Folienkopien zur Vorlesung

Verteilte Algorithmen

F. Mattern

Departement Informatik, ETH Zürich

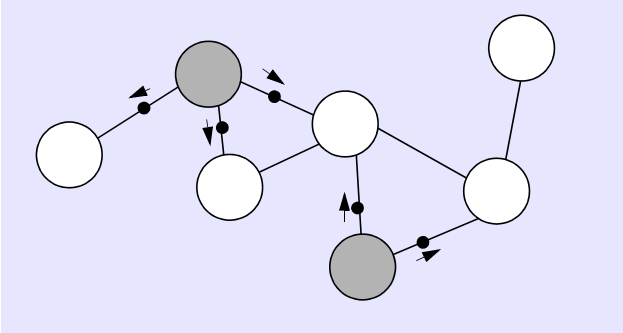
FS 2020 (3-stündig inkl. Übungen)

Achtung: Prüfungsrelevant ist der gesamte Stoff der Vorlesung, nicht alleine der Inhalt dieser Foliensammlung!

Verteilte Algorithmen

F. Mattern

Departement Informatik
ETH Zürich



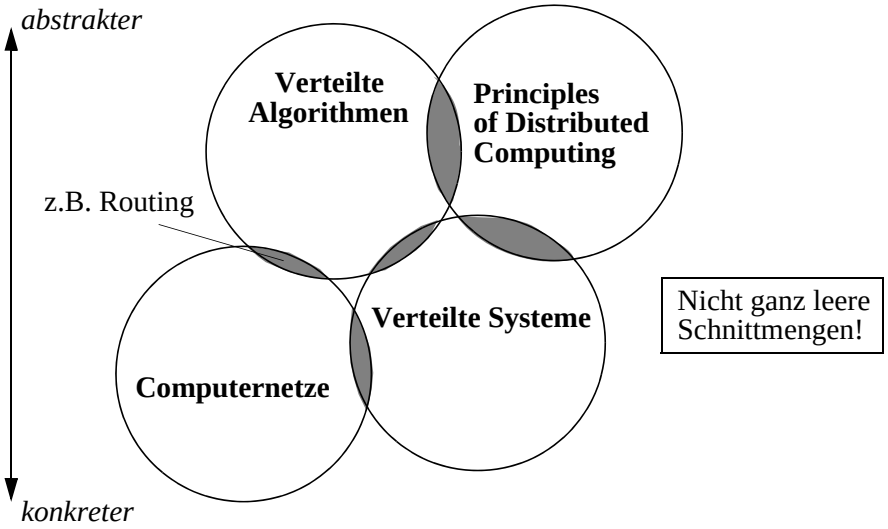
Verteilte Algorithmen
= Algorithmen für / in verteilten Systemen

Was ist "verteilt" an einem verteilten Algorithmus?

Vorläufige Antwort: *Zustand* und *Ablaufkontrolle* auf verschiedene "Orte"
Daher werden uns prinzipielle Konzepte wie "Zustand" intensiv beschäftigen!

Einordnung der Vorlesung

- 3-stündige Vorlesung (mit integrierter Übungen) auf deutsch
  - eine einzige Pause (flexibel)
- Sinnvolle Vorkenntnisse:
  - Verteilte Systeme
  - Grundkenntnisse der Informatik und Mathematik
- Folienkopien schriftlich; Prüfung mündlich
- Homepage der Vorlesung:
  - `www.vs.inf.ethz.ch/edu/ ...`



Nicht ganz leere Schnittmengen!

- Ubiquitous Computing

- ...

Vorlesungsankündigung "Verteilte Algorithmen"

Verteilte Algorithmen sind Verfahren, die dadurch charakterisiert sind, dass mehrere autonome Prozesse gleichzeitig Teile eines gemeinsamen Problems in kooperativer Weise bearbeiten und der dabei erforderliche Informationsaustausch ausschliesslich über Nachrichten erfolgt. Derartige Algorithmen kommen im Rahmen verteilter Systeme zum Einsatz, bei denen kein gemeinsamer Speicher existiert und die Übertragungszeit von Nachrichten i.Allg. nicht vernachlässigt werden kann. Da dabei kein Prozess eine aktuelle konsistente Sicht des globalen Zustands besitzt, führt dies zu interessanten Problemen.

Im einzelnen werden u.a. folgende **Themen** behandelt:

Modelle verteilter Berechnungen;
Raum-Zeitdiagramme;
virtuelle Zeit; logische Uhren und Kausalität;
Wellenalgorithmen;
verteilte und parallele Graphtraversierung;
Berechnung konsistenter Schnappschüsse;
wechselseitiger Ausschluss;
Election und Symmetriebrechung;
verteilte Terminierung;
Garbage-Collection in verteilten Systemen;
Beobachten verteilter Systeme;
Berechnung globaler Prädikate.

Themenspektrum

- Wellenalgorithmen, Traversierungsverfahren
- Verteilte Terminierung
- Verteiltes Garbage-Collection
- Globaler Zustand, Schnappschuss
- Konsistente Beobachtungen und globale Prädikate
- Logische Zeit, Kausalität, Konsistenz
- Wechselseitiger Ausschluss
- Election
- Kommunikationssemantik (synchron, asynchron, kausal)

Algorithmen

- Prinzipielle Phänomene und Begriffe
  - Kausalität, Parallelität,...
- Anwendungsbezug
  - verteilte Betriebssysteme, Datenbanken, Anwendungen, Tools...

- Techniken, Einsichten, Zusammenhänge,...
- Basisverfahren, grundsätzliche Probleme,...
- Abstraktion, Modellbildung, Formalisierung,...
- Problemlösungstechniken, Analysetechniken,...
- Qualitätsbewertung, Komplexitätsabschätzung,...
- Verifikationstechniken

(verallgemeinerbare) Erkenntnisse

Literatur

Folienkopien der Vorlesung

F. Mattern: Verteilte Basisalgorithmen. Springer-Verlag

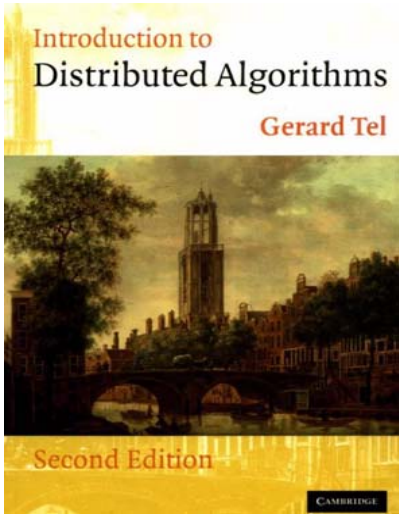
A. Kshemkalyani, M. Singhal: Distributed Computing: Principles, Algorithms, and Systems, Cambridge University Press

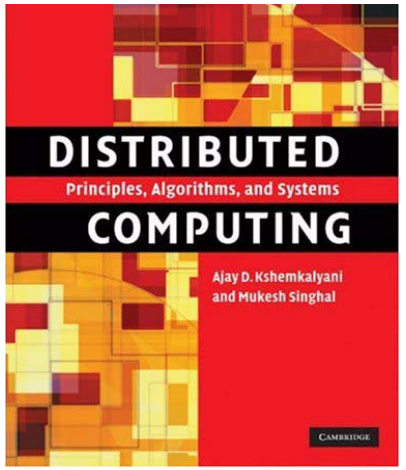
G. Tel: Introduction to Distributed Algorithms. Cambridge University Press, 2nd edition

N. Lynch: Distributed Algorithms, Morgan Kaufmann

V. K. Garg: Principles of Distributed Systems, Kluwer

Artikel aus Fachzeitschriften (wird in der Vorlesung bekanntgegeben)



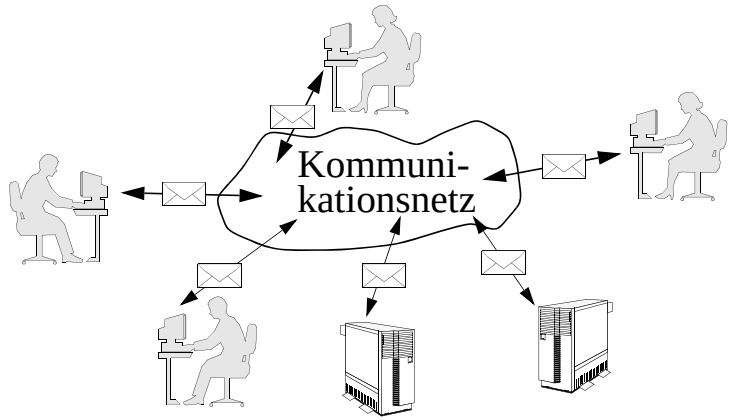


Notwendige mathematische Grundbegriffe (gut wiederholen!)

- Kartesisches Produkt, Relation, binäre Relation
- Halbordnung, lineare (totale) Ordnung
- Äquivalenzrelation
- Transitiv, reflexiv, symmetrisch, asymmetrisch, antisymmetrisch
- Infimum, Supremum, Verband, Hasse-Diagramm
- Graphen: gerichtet & azyklisch ("DAG"), vollständig
- Spannbaum
- Bitmatrix, Adjazenzmatrix von Graphen
- Transitive (bzw. reflexiv-transitive) Hülle
- Prädikat
- Harmonische Reihe, Approximation durch log n
- Erwartungswert
- ggt, Satz von Euklid
- Umrechnung von Logarithmen verschiedener Basen

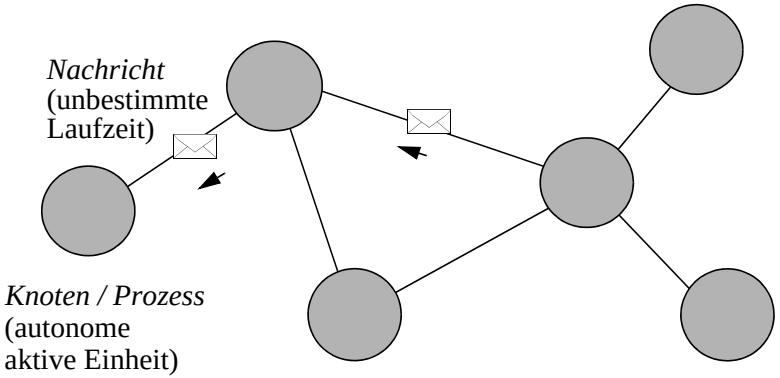
- Bekannt aus der Vorlesung "Verteilte Systeme":
  - Lamport-Uhren und "happened before"-Relation
  - synchrone / asynchrone Kommunikation
  - Hypercube

Verteiltes System



- Rechner, Personen, Prozesse, “Agenten”,... sind an *verschiedenen Orten*
- Autonome Handlungsträger, die jedoch gelegentlich *kooperieren* (und dazu *kommunizieren*)

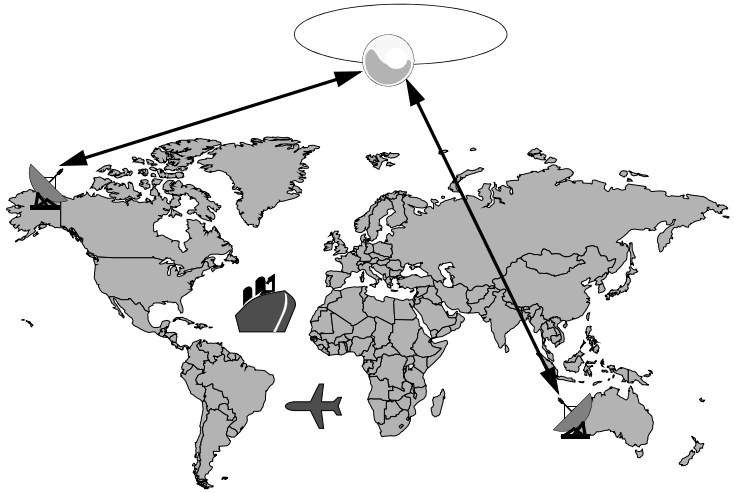
Verteiltes System II



Physisch verteiltes System:
Mehrrechnersystem ... Cluster ... Rechnernetz

Logisch verteiltes System: Prozesse (aktive Objekte)
- Verteilung des Zustandes (keine globale Sicht)
- Keine gemeinsame Zeit (globale, genaue "Uhr")

Verteiltes System III

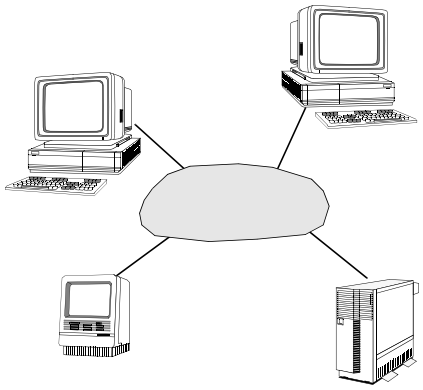


Auch die "reale Welt" ist ein verteiltes System:

- Viele gleichzeitige (“parallele”) Aktivitäten
- Exakte globale Zeit nicht erfahrbar / vorhanden
- Keine konsistente Sicht des Gesamtzustandes
- Kooperation durch explizite Kommunikation
- *Ursache* und *Wirkung* zeitlich (und räumlich) getrennt

- “Inkonsistente Zustände”: Kriegsende zwischen England und Frankreich war in den Kolonialgebieten erst später bekannt!
- Heute: “zeitkompakter Globus” - weitgehend synchronisierte Uhren

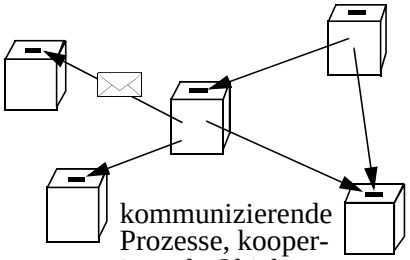
Sichten verteilter Systeme



Computernetz mit "Rechenknoten":

- Compute-Cluster
- LAN (Local Area Network)
- Internet

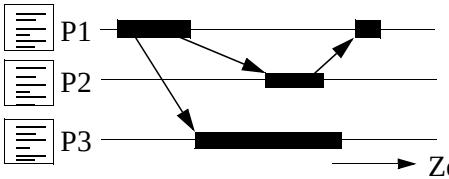
$\Rightarrow$ Routing, Adressierung,..



kommunizierende Prozesse, kooperierende Objekte

Objekte in *Betriebssystemen, Middleware, Programmiersprachen*

$\Rightarrow$ "Programmierersicht" (Client, Server...)



Algorithmen- und Protokollebene

- Aktionen, Ereignisfolgen
- Konsistenz, Korrektheit

zunehmende Abstraktion

Parallele $\leftrightarrow$ verteilte Algorithmen

Sequentieller Algorithmus

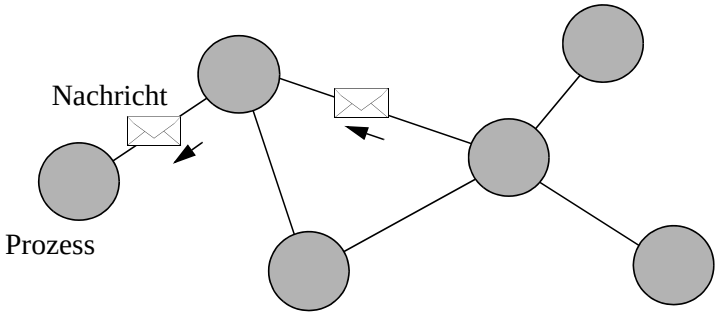
Paralleler Algorithmus

- mehrere, oft gleichartige Prozesse
- Synchronisation oft über gemeinsamen Speicher
- Zweck: Beschleunigung gegenüber seq. Algorithmus

Verteilter Algorithmus

- mehrere Prozesse kommunizieren über Nachrichten
- gemeinsames Ziel $\rightarrow$ Kooperation
- idealerweise: keine zentrale Kontrolle

oft asynchron



Aspekte verteilter Systeme

im Vergleich zu *sequentiellen* Systemen:

- Grösse und Komplexität → jede(r) ist anders
- Heterogenität → jede(r) ist anders
- Nebenläufigkeit → vieles gleichzeitig
- Nichtdeterminismus → morgen anders als heute...
- Zustandsverteilung → niemand weiss alles

- Programmierung *komplexer*
- Test und Verifikation *aufwendiger*
- Verständnis der Phänomene *schwieriger*

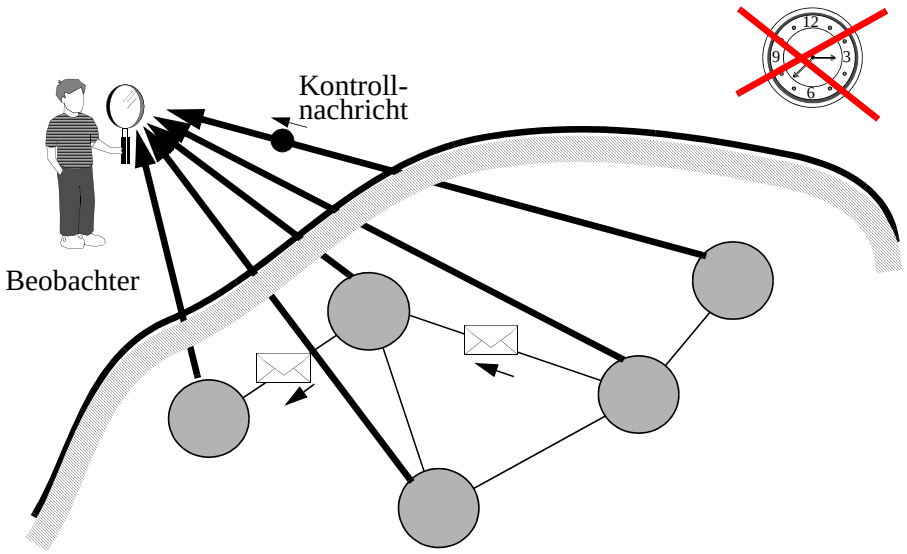
⇒ gute Werkzeuge (“Tools”) und Methoden

⇒ adäquate Modelle, Algorithmen, Konzepte
- zur Beherrschung der neuen Phänomene

Ziel: Verständnis der grundlegenden Phänomene, Kenntnis der geeigneten Konzepte und Verfahren

Phänomene verteilter Berechnungen

Beobachten verteilter Berechnungen



Beobachten geht nur über das Empfangen von "Kontrollnachrichten" (mit unbestimmter Laufzeit)

"Axiom": Mehrere Prozesse können "niemals" gleichzeitig beobachtet werden

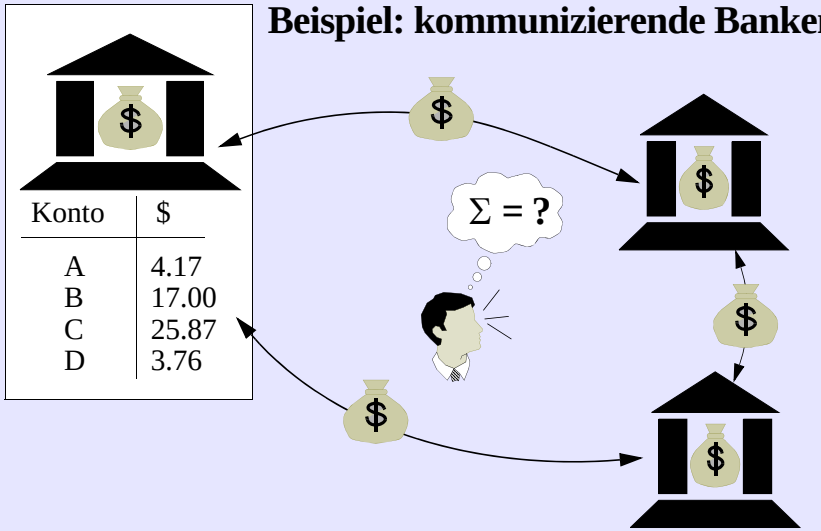
was heisst das?

"Korollar": Aussagen über den globalen Zustand sind schwierig

Ein erstes Beispiel: Wieviel Geld ist in Umlauf?

- konstante Geldmenge, oder
- monotone Inflation (→ Untergrenze)

Beispiel: kommunizierende Banken



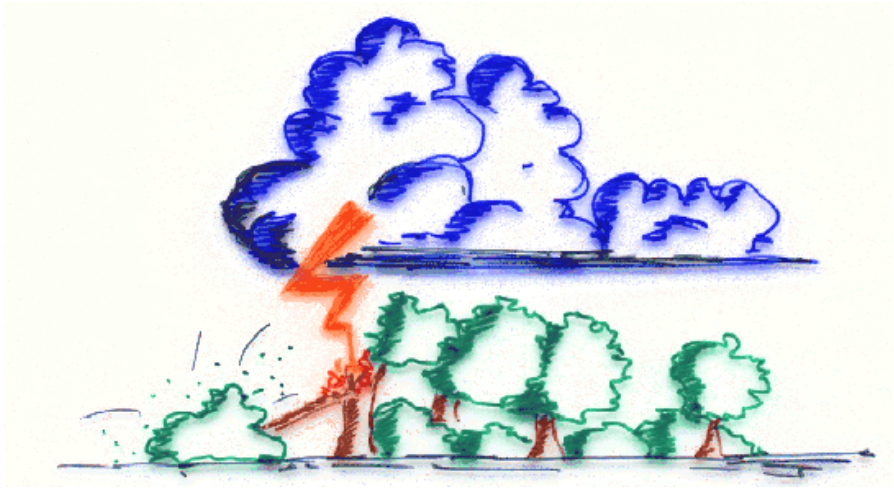
- Modellierung:
  - verteilte Geldkonten
  - ständige Transfers zwischen den Konten
- Erschwerte Bedingungen:
  - niemand hat eine globale Sicht
  - es gibt keine gemeinsame Zeit (“Stichtag”)

→ Geht das dann überhaupt?
→ Ist das überhaupt ein wichtiges Problem?

==> *Schnappschussproblem*

Beispiel: Prähistorische Gesellschaft

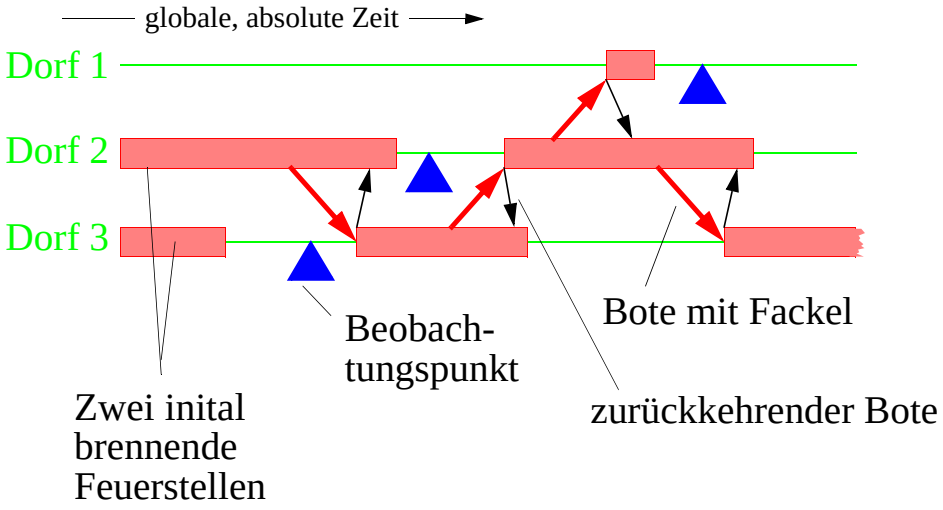
- Einzelne, versprengte Stämme
- Beschränktes technisches Wissen
  - Feuermachen unbekannt
  - Feuerhüter ist ein angesehener Beruf!
- Feuer erlischt → von einem anderen Stamm holen
- Nur lokale Sicht
  - Glimmt noch ein Fünkchen Hoffnung?
  - Oder ist der Ofen aus?
- Alle Feuer erloschen und kein Feuerträger unterwegs:
  - Warten auf einen Blitz (→ grosser Feuerzauber...)



- Feststellen der Terminierung ist wichtig
  - kein warmes Essen bis zum nächsten Gewitter...

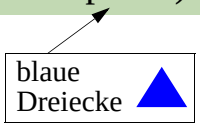
Falsche Beobachtungen

Beobachter kann nicht alles gleichzeitig betrachten!



Für alle Feuerstellen gilt (zu einem Zeitpunkt):

- kein Feuer brennt
- kein Bote ist unterwegs

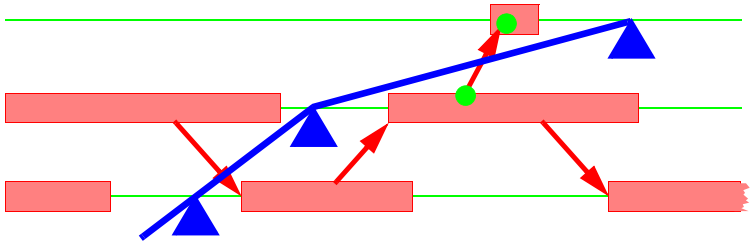


Aber: Es gibt keinen *einzigen Zeitpunkt*, wo kein Feuer brennt!

⇒ Beobachtung liefert ein *falsches (“schiefes”) Bild!*

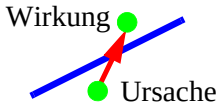
==> *Terminierungserkennungs-Problem*

Was läuft schief?



- Es scheint, als ob alle Feuer aus sind und genauso-viele Boten angekommen wie aufgebrochen sind
  - dann wäre tatsächlich “alles aus” (bis zum nächsten Blitzschlag)

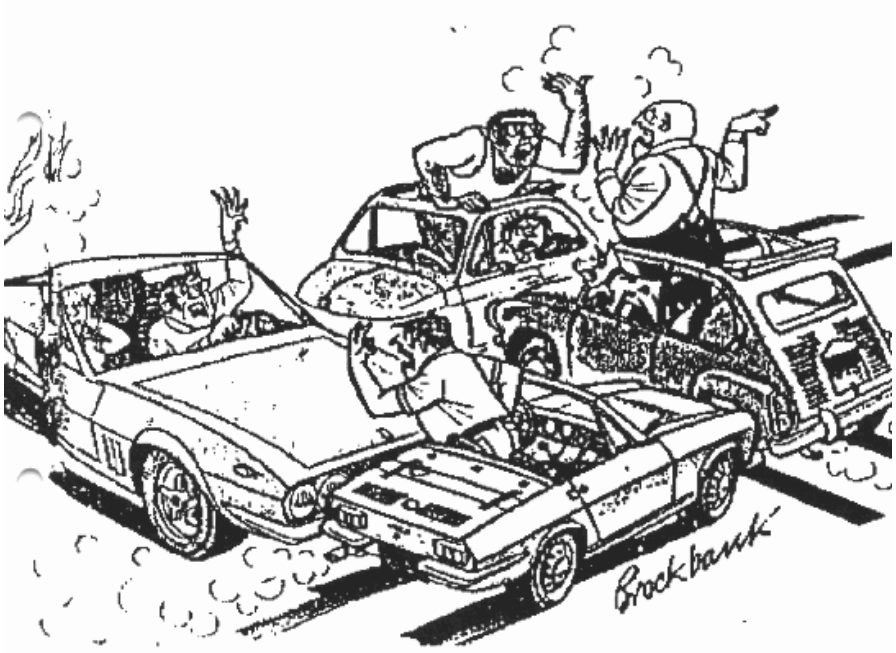
- Täuschungsgrund: Beobachtung ist nicht kausaltreu



  - Wirkung wird registriert, nicht aber die zugehörige Ursache
  - inkonsistente Sicht durch schiefen Schnitt

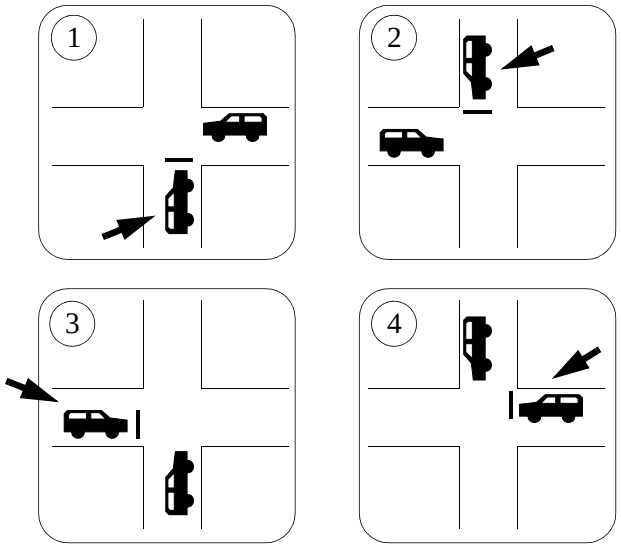
- Würde man alles gleichzeitig beobachten, dann könnte man niemals eine Wirkung ohne ihre Ursache wahrnehmen!

Das Deadlock-Problem



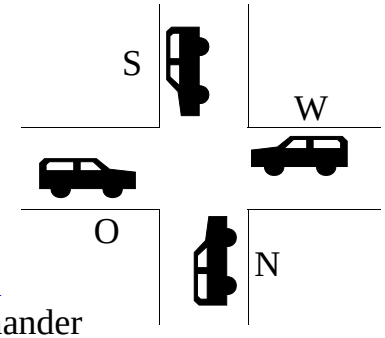
Das Deadlock-Problem

Phantom-Deadlocks



Vier Einzelbeobachtungen der Autos N, S, O, W

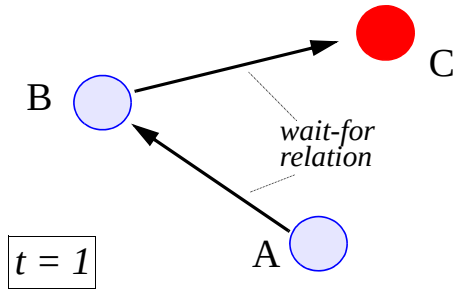
1) N wartet auf W
2) S wartet auf O
3) O wartet auf N
4) W wartet auf S



zu (im Allgemeinen) verschiedenen Zeitpunkten liefert den falschen Eindruck, als würden zu einem einzigen Zeitpunkt alle zyklisch aufeinander warten (→ Verklemmung)

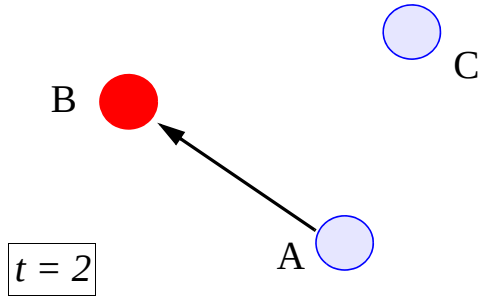
==> *verteiltes Deadlockproblem*

Phantom-Deadlocks

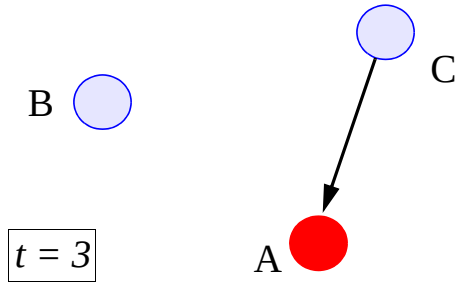


(C benutzt ein exklusives Betriebsmittel)

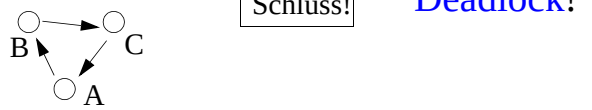
beobachte B:
⇒ B wartet auf C



beobachte A:
⇒ A wartet auf B



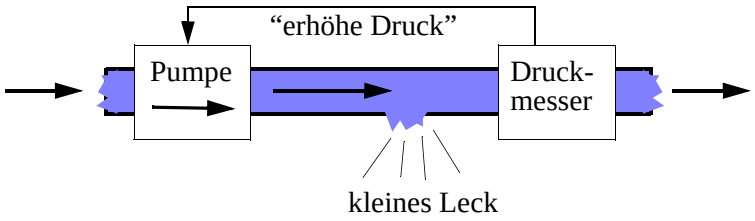
beobachte C:
⇒ C wartet auf A

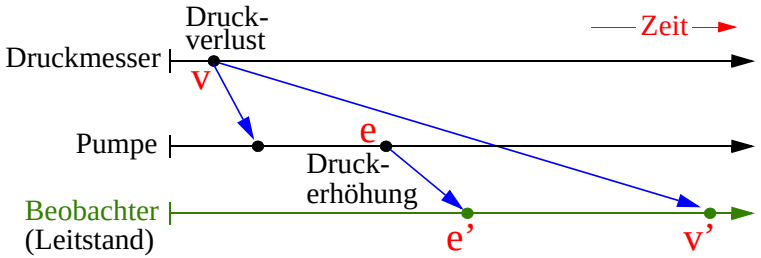


Deadlock!

Kausal (in)konsistente Beobachtungen

- Gewünscht: Eine Ursache stets vor ihrer (u.U. indirekten) Wirkung beobachten





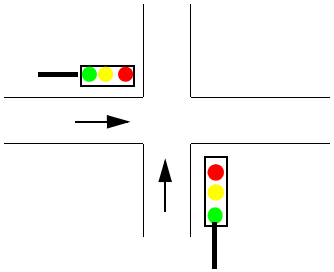
Falsche Schlussfolgerung des Beobachters:

Es erhöhte sich der Druck (aufgrund einer unbegründeten Aktivität der Pumpe), es kam zu einem Leck, was durch den abfallenden Druck angezeigt wird.

==> *"Causal order-Problem"*

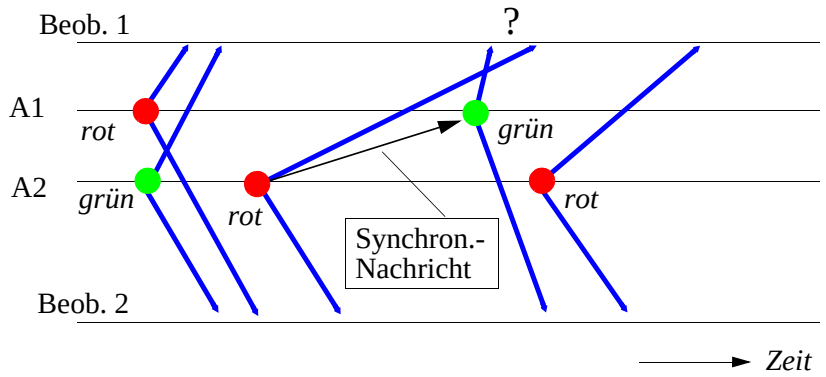
Beispiel: Verteilte Ampelsteuerung

(hier für 2 Ampeln)



Konsistenzproblem bei *mehreren Beobachtern*

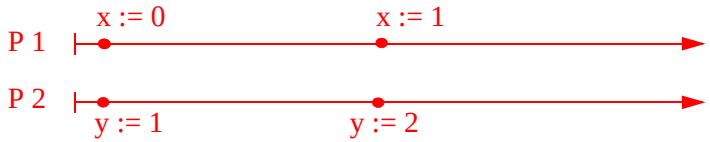
- Jede Ampel darf autonom auf rot schalten
- Eine Ampel darf nur dann auf grün schalten, wenn sie erfahren hat, dass die andere rot ist
- Umschalten der Ampel ist ein *Ereignis* ● (*atomar*: zeitlos, nicht unterbrechbare Aktion)

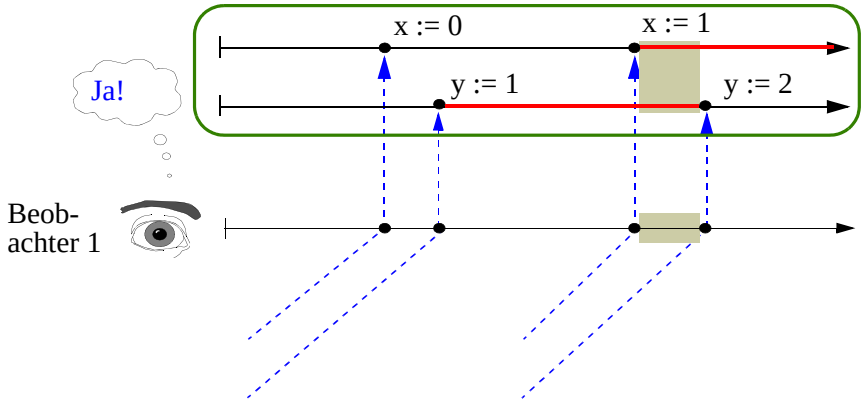


- Welcher Beobachter hat Recht? (Wieso?)
- Ähnliche Probleme (in komplexerem Umfeld) tauchen in der Praxis tatsächlich auf (z.B. Flugkontrollsystem)

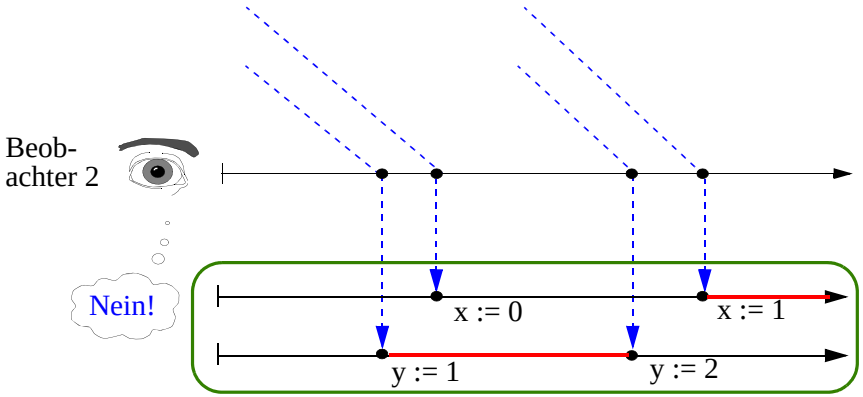
Das Problem globaler Prädikate

Frage: Gilt in der vorliegenden Berechnung x = y ?





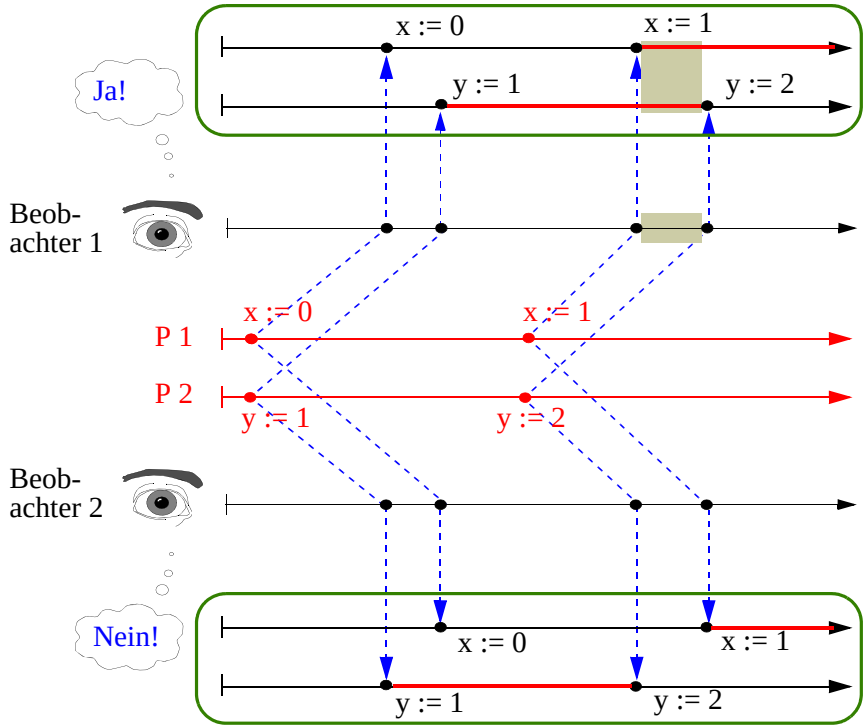
Das Problem globaler Prädikate



- Beide Beobachtungen sind gleich “richtig”
- Die Beobachter stimmen bzgl. x = y nicht überein!

Aber was denn nun: gilt x=y in dieser Berechnung oder nicht?

Frage: Gilt in der vorliegenden Berechnung x = y ?



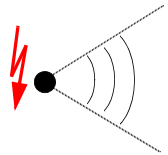
- Beide Beobachtungen sind gleich “richtig”
- Die Beobachter stimmen bzgl. x = y nicht überein!

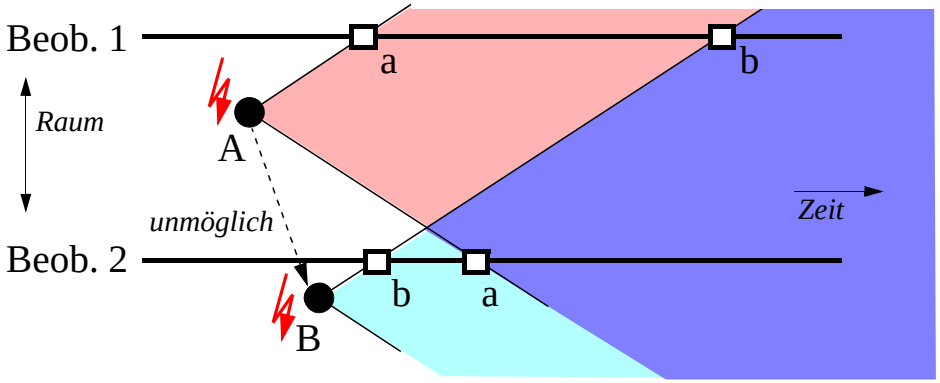
Aber was denn nun: gilt x=y in dieser Berechnung oder nicht?

Relativierung der Gleichzeitigkeit

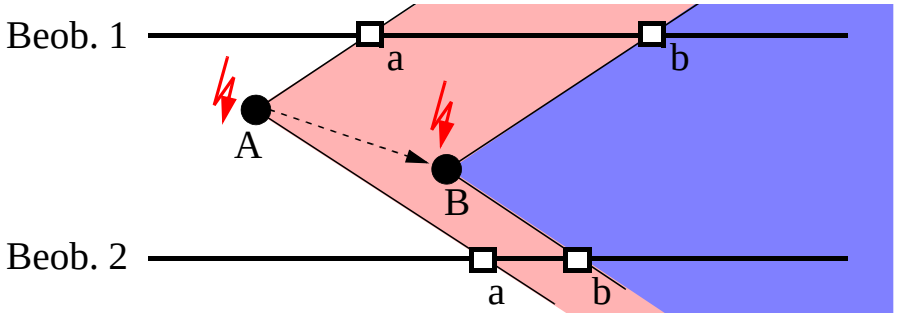
Zwei "kausal unabhängige" Ereigniss können in beliebiger Reihenfolge beobachtet werden!

Lichtkegel-Prinzip der relativistischen Physik





A und B geschehen "gleichzeitig"
(jedenfalls *kausal unabhängig*!)

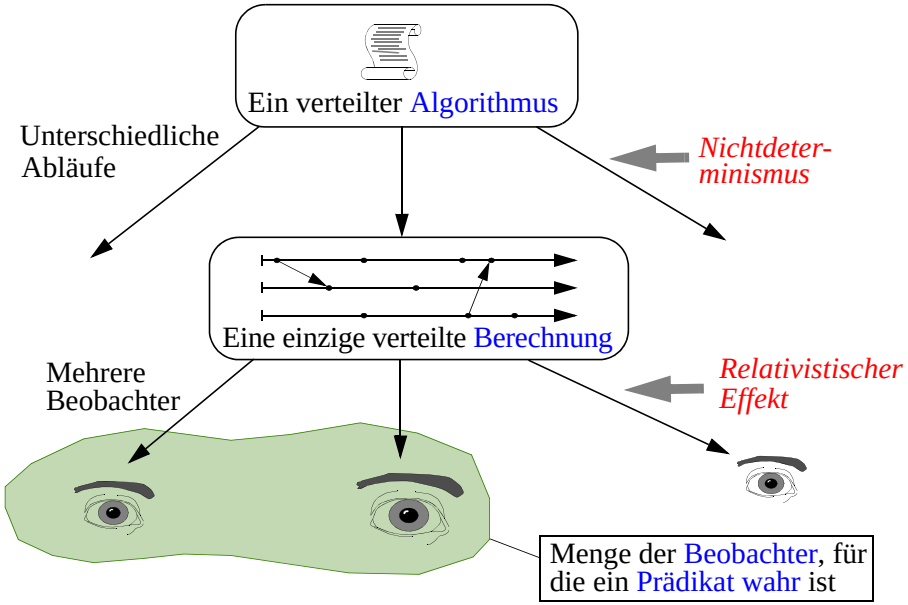


beobachterinvariant
⇒ *objektive Tatsache*

B liegt im Kegel von A →
B hängt kausal ab von A →
Alle Beobachter sehen B nach A

Die "Beobachtungsvielfalt"

- Verschiedene Beobachter sehen *verschiedene Wirklichkeiten*



Konsequenz:

Es ist naiv (d.h. falsch!), einen verteilten Debugger zu entwickeln, mit dem man solche (im sequentiellen Fall "richtigen") Fragen beantworten kann!

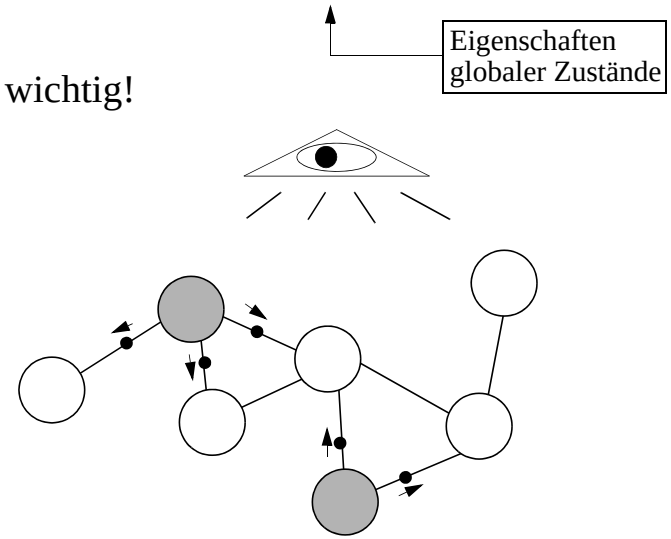
Ursache:

Bei sequentiellen Berechnungen fallen Berechnung und Beobachtung zusammen, im verteilten Fall nicht!

Halbordnung / lineare Ordnung

Globale Prädikate...

... sind aber wichtig!



Bsp.:

- Wieviel Geld ist in Umlauf?
- Ist die parallele Approximation gut genug?
- Ist die verteilte Berechnung terminiert?
- Ist ein bestimmtes Objekt "Garbage"?
- Konsistenter Sicherungspunkt (vert. Datenbank)?
- Wie hoch ist die momentane Last?
- Liegt eine zyklische Wartebedingung vor?

Einige globale Prädikate lassen sich "eindeutig" bestimmen

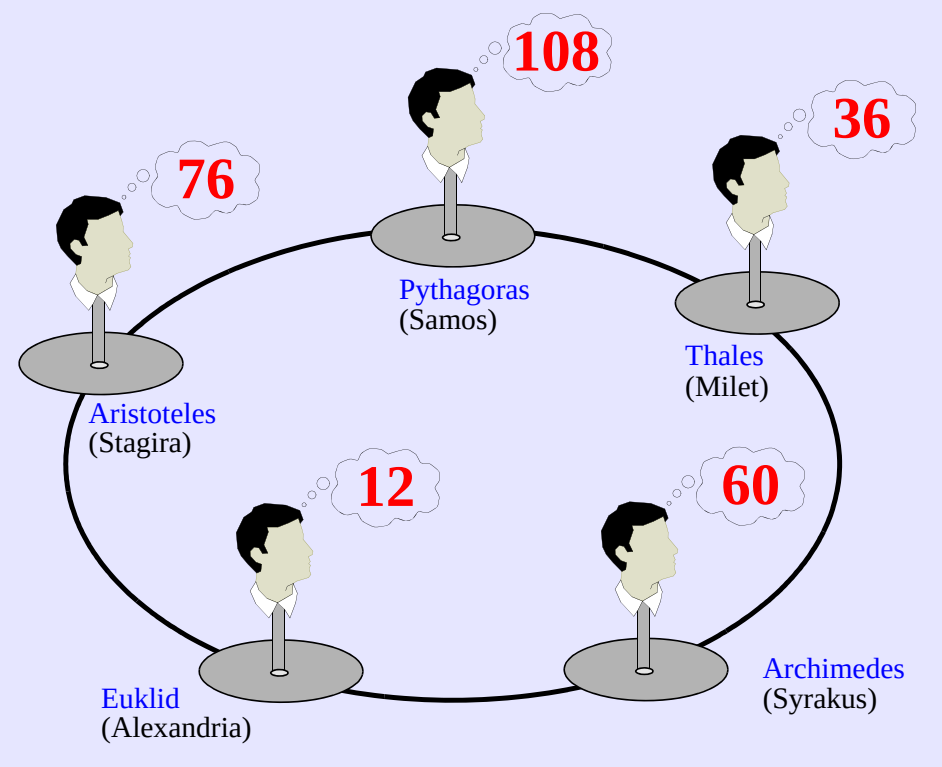
- Welche? → fundamentale Aspekte analysieren
- Wie? → *Algorithmen*

u.a. Thema der Vorlesung

Erste Beispiele für verteilte Algorithmen

Ein erster verteilter Algorithmus: Verteilte Berechnung des ggT

Fünf in Raum und Zeit verteilte griechische Philosophen wollen zusammen den grössten gemeinsamen Teiler ihres Alters berechnen...



- Was ist der ggT einer Menge $\{a_1,...,a_n\}$ nat. Zahlen?
- Wie funktioniert der *euklidische Algorithmus*?

"Elemente" Buch 7, Satz 1 und 2

Der grösste gemeinsame Teiler (ggT)

- Nachfolgend werden nur natürliche Zahlen betrachtet
- t heisst *Teiler* einer Zahl p, falls es $q \neq 0$ gibt mit $q\,t = p$
  - Bsp: 6 ist Teiler von 30 (da $5 \times 6 = 30$)
    3 ist Teiler von 30 (da $10 \times 3 = 30$)
    aber: 8 ist kein Teiler von 30
- 1 ist Teiler jeder Zahl (klar nach Definition)

- t heisst *gemeinsamer Teiler* von u, v
  wenn t Teiler von u ist und t Teiler von v ist
  - Bsp: 6 ist gemeinsamer Teiler von 30 und 18.
    Aber 3, 2, 1 sind auch gemeinsamer Teiler.
    Jedoch gibt es keine anderen gemeinsamer Teiler
    $\Rightarrow$ 6 ist der grösste gemeinsame Teiler (ggT).
- Zu je zwei Zahlen $\neq 0$ gibt es stets einen ggT, da die Menge der gemeinsamen Teiler mindestens die 1 enthält und endlich ist (was aber ist ggT(x,0)?)
- Kanonische Erweiterung auf $n \geq 1$ Argumente
  - Bsp: ggT(108, 36, 60, 12, 76) = 4

- *Anwendung*: Z.B. Kürzen von Brüchen

Satz von Euklid

Der ggT zweier positiver ganzer Zahlen x,y (mit x≥y>0) ist gleich dem ggT von y und dem Rest, der bei ganzzahliger Division von x durch y entsteht

- Offenbar ist ggT(x,x) = x für alle x
- Man setzt nun noch ggT(x,0) = x für alle x

- Obiger Satz legt eine rekursive Realisierung nahe

```
ggt(x,y)  -->  ggt(y,mod(x,y)
```

  - für x > y werden in der Rekursion beide Argumente jeweils kleiner
  - Rekursion geht irgendwann mit ggT(...,0) zuende
  - oft: iterative Formulierung (statt rekursiver)

⇒ *Euklidischer Algorithmus*

Vorteil: Zur Bestimmung des ggT sind Primfaktorzerlegung ("Schulmethode") oder Probedivisionen nicht notwendig

Hinweis: Der Beweis des Algorithmus (und des Satzes) ist eine interessante Wiederholungsübung!

Das (abstrakte) Lösungsprinzip

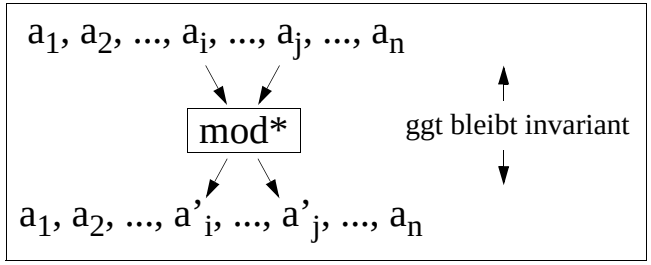
- *Invariante* (für x ≥ y > 0):

  ggT(x,y) = ggT(y,mod*(x,y))

  Wie *Restfunktion* mod, soll jedoch y liefern, falls x ganzzahliges Vielfaches von y ist. Also: mod*(a,b) = mod(a-1,b)+1

- *Idee:* Ersetze x durch mod*(x,y) ← Ist i.Allg. kleiner als x

- Erweiterung auf n Zahlen:



Frage: Wieso wurde mod zu mod* modifiziert?

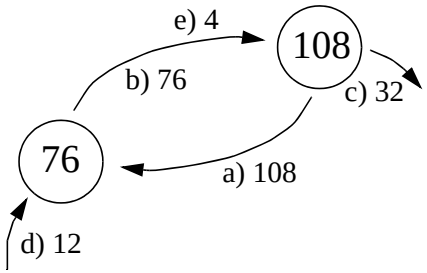
Greife zwei beliebige Elemente a_i, a_j ($i \neq j$) heraus und ersetze den grösseren Wert durch mod*(a_i,a_j)

⇒ konvergiert gegen den ggT

- Dies nun "verteilt" realisieren! ← In "Teamarbeit", damit es schneller geht...

Nachrichtenaustausch

Prinzip: Kooperation durch Kommunikation



zwischen den Nachbarn auf dem Ring

- Nachrichten sind beliebig lange unterwegs
- Parallele Aktivitäten

Idee:

- Initial beide Nachbarn spontan informieren
- Danach nur auf eintreffende Nachrichten *reagieren*
- Neue Erkenntnisse an alle Nachbarn übermitteln

Verhaltensbeschreibung eines Prozesses P_i:

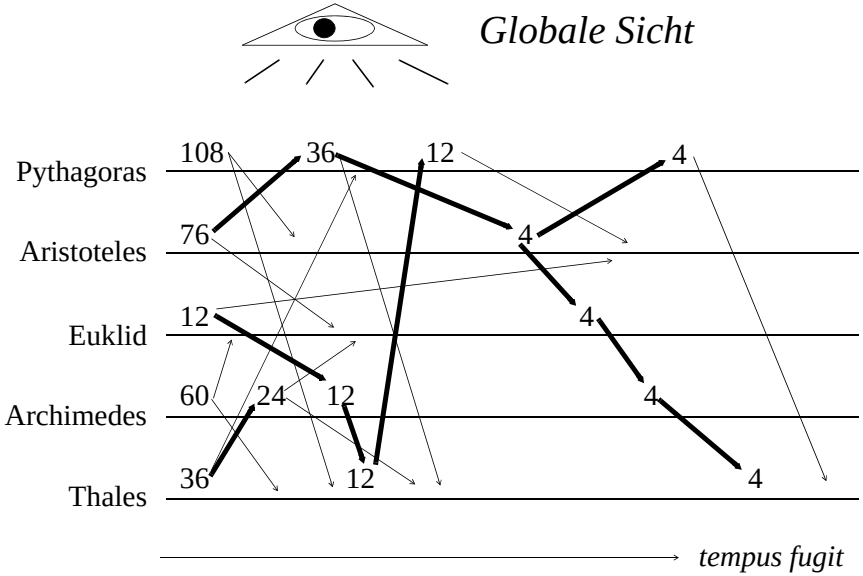
{Eine Nachricht <y> ist eingetroffen}
if $y < M_i$ **then**
  $M_i := \text{mod}(M_i\text{-}1,y)+1$;
  send $<M_i>$ **to** all neighbours;
fi

Jeder Prozess P_i hat seine eigene Variable M_i.

Jeder Prozess hat das gleiche prinzipielle Verhalten.

Atomare Aktion:
In der Zwischenzeit u.U. eintreffende Nachrichten werden im "Hausbriefkasten" zwischengepuffert...

Berechnungsablauf und Zeitdiagramm



Ablauf einer *möglichen* "verteilten Berechnung" mit einem *Zeitdiagramm*.

Nicht-deterministisch, abhängig von der Nachrichtenlaufzeit!

- *Terminiert* wenn (?):

Das ist unrealistisch!

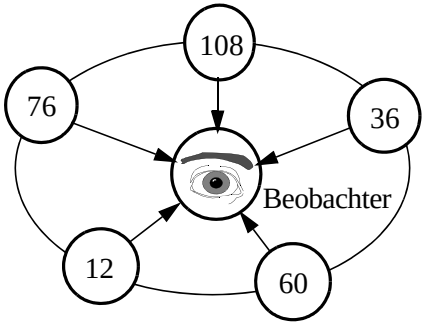
  (a) jeder Prozess den ggT kennt,
  (b) alle den gleichen Wert haben,
  (c) alle Prozesse passiv sind und keine Nachricht unterwegs ist.

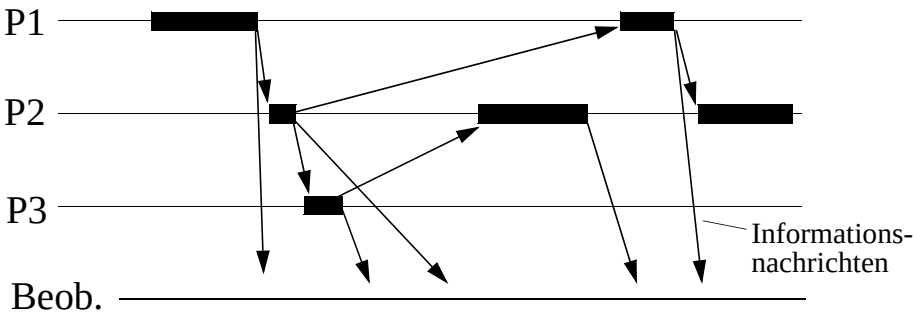
Beweisbare math. Eigenschaft

Diese "stabile" Stagnationseigenschaft ist *problemunabhängig*!

- Was ist adäquat?
- Wie feststellen?

Globaler ggT-Beobachter?





Bekommt der Beobachter ein "richtiges Bild" des Geschehens?

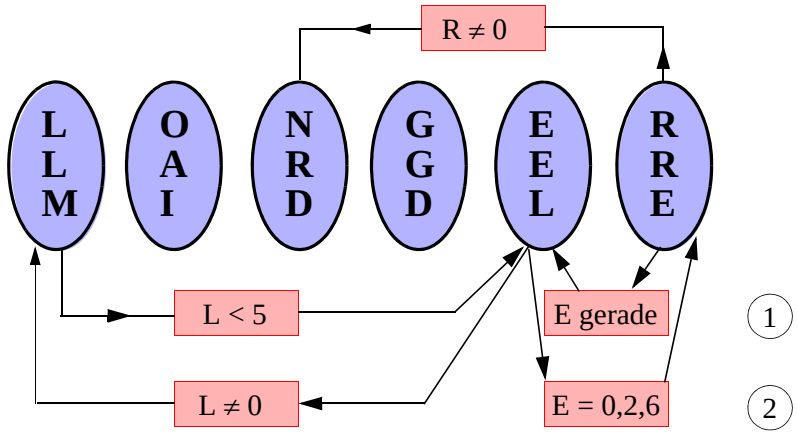
- was heisst das genau?
- und wenn Informationsnachrichten verschieden schnell sind?
- könnte der Beobachter einen Zwischenzustand (z.B. "alle haben den Wert 12") irrtümlich als Endzustand interpretieren?
- wie stellt er überhaupt das Ende der Berechnung fest?

Ein weiteres Beispielproblem: Paralleles Lösen von "Zahlenrätseln"

```
  LONGER          207563
+ LARGER    →   + 283563
---------       --------
  MIDDLE          491126
```

Pro Spalte ein Prozess

- Reaktives Verhalten: Auslösen atomarer Aktionen
- Propagieren neuer Erkenntnisse (= Einschränkungen)
  ("parallele Constraint-Propagation")



Endergebnis:

$1 \le L \le 4$
$M \ge 2$
$R = 1,3,5,6,8$
$E = 0,2,6$
sonst keine Einschränkung

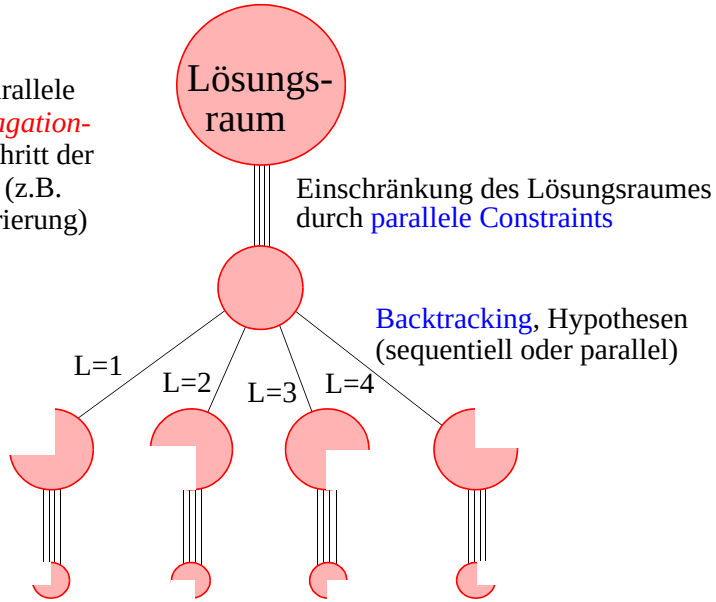
Probleme:

1) Keine eindeutige Lösung
   → Backtrack-Algorithmus
2) Entdeckung der "Stagnation"?
   (Ende der Parallelphase)

Offensichtlich *gleiches Schema* wie ggT-Berechnung!

Der aufgesetzte Backtrack-Algorithmus

Abwechselnd: Parallele *Constraint-Propagation-Phase* und ein Schritt der *Backtrack-Phase* (z.B. Hypothesengenerierung)



- Hypothese = beliebige Menge von Constraints
  (von einem “Orakel” statt von einer Spalte)

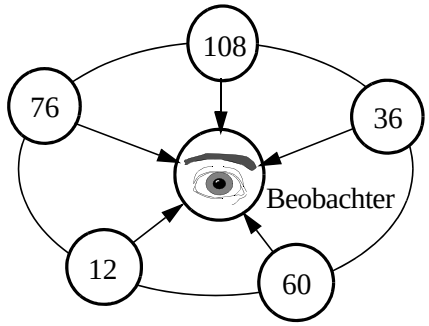
- Einheit, die die Hypothesen generiert und verwaltet, muss die Terminierung einer Constraint-Phase feststellen können
  - wie würde man hier die Terminierung zweckmässigerweise definieren?
  - problembezogen wie bei ggT ("alle Werte identisch") hier nicht einfach
  - "alle passiv und keine sinnvolle Nachricht mehr unterwegs"?

Bemerkungen:

- nicht unbedingt beste Parallelisierungsstrategie!
- Problem ist (in Verallgemeinerung) NP-vollständig

Übungen (1) zur Vorlesung "Verteilte Algorithmen"...

a) Wie kann man beweisen, dass für *jeden* denkbaren Ablauf das Endergebnis stets der ggT ist?



b) Bleibt der Algorithmus (und/oder der Beweis) korrekt, wenn im Algorithmus $y<M_i$ durch $y \leq M_i$ ersetzt wird?

Verhaltensbeschreibung eines Prozesses P_i:

{Eine Nachricht $<y>$ ist eingetroffen}
if $y < M_i$ **then**
  $M_i := \text{mod}(M_i\text{-}1,y)+1$;
  send $<M_i>$ **to** all neighbours;
fi

...Übungen (1)

c) Man vergleiche die verteilte Berechnung des ggT-Algorithmus für zwei Zahlen mit dem üblichen sequentiellen ggT-Algorithmus für zwei Zahlen

d) Genügt es auch, nur in Uhrzeigerrichtung eine Nachricht zu senden (anstatt an beide Nachbarn)?

e) Kann statt des Ringes eine andere Topologie verwendet werden? Welche?

f) Wie kann man erreichen, dass ein ggT-Beobachter (der über jede Wertänderung eines Prozesse informiert wird) eine "kausaltreue" (vgl. Aufg. g) Beobachtung macht?

g) Formalisieren Sie für Zeitdiagramme den Begriff (potentiell, indirekt) "kausal abhängig" als Halbordnung über "Ereignissen"

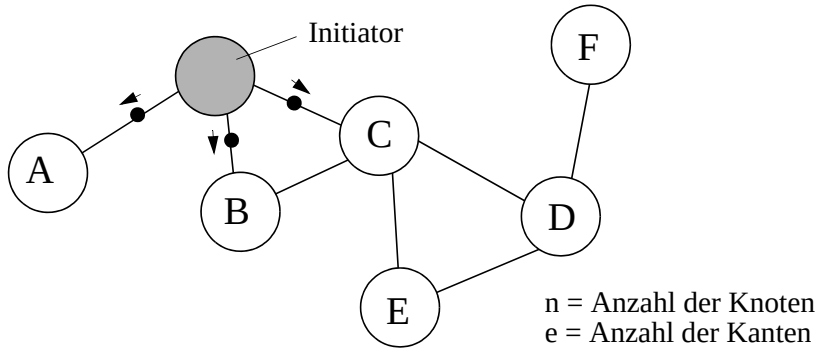
h) Beobachtungen sind eine lineare Ordnung von (beobachteten) Ereignissen. In welcher Beziehung steht die oben erwähnte Halbordnung zu dieser linearen Ordnung? Können Sie eine Vermutung darüber anstellen, was der Schnitt aller möglichen kausaltreuen Beobachtungen einer verteilten Berechnung aussagt?

i) Wie kann der Beobachter die Terminierung erkennen?

Flooding, Echo-Algorithmus, Broadcast

Informationsverteilung durch “Fluten”

“flooding”



n = Anzahl der Knoten
e = Anzahl der Kanten

- Voraussetzung: zusammenhängende Topologie
- Prinzip: jeder erzählt *neues* Gerücht allen anderen Freunden
- Kein Routing etc. notwendig

wieder das gleiche Prinzip wie beim ggT und beim Zahlenrätsel!

- Wieviele Nachrichten werden versendet?
  - jeder Knoten sendet über alle seine inzidenten Kanten ($\rightarrow$ 2e)
  - jedoch nicht über seine Aktivierungskante zurück ($\rightarrow$ -n)
  - Ausnahme: Initiator ($\rightarrow$ +1)

  $\Rightarrow$ *Also*: 2e - n + 1

- Frage: Wie *Terminierung* feststellen?
  d.h.: wie erfährt der Sender (= Initiator), wann alle erreicht wurden?
  (das ist für “verlässlichen” Broadcast notwendig)

Flooding-Algorithmus - eine etwas formalere Spezifikation

- Zwei *atomare Aktionen* für jeden Prozess:

- wechselseitig ausgeschlossen
- "schlagartig"?
- ununterbrechbar?

Assertion (muss wahr sein, damit Aktion ausgeführt wird)

R: {Eine Nachricht <info> kommt an}
if not informed **then**
send <info> **to** all other neighbors;
informed := true;
fi

Natürlich auch "merken" der per Nachricht erhaltenen Information <info>

I: {not informed}
send <info> **to** all neighbors;
informed := true;

- initial sei informed = false
- Aktion R wird nur bei Erhalt einer Nachricht ausgeführt
  - "message driven"
- Aktion I wird vom Initiator *spontan* ausgeführt
  - darf es mehrere *konkurrente* Initiatoren geben?

Terminierungserkennung von Flooding

1) Jeder Prozess informiert (evtl. indirekt) den Initiator (oder einen Beobachter) per *Kontrollnachricht*, wenn er eine *Basisnachricht* erhält; Initiator zählt bis 2e-n+1

   - Nachteile?
     - n und e müssen dem Initiator bekannt sein
     - indirektes Informieren kostet u.U. viele Einzelnachrichten
   - Nachrichtenkomplexität?
   - Variante: Prozess sendet Kontrollnachricht, wenn er *erstmalig* eine Basisnachricht erhält; Initiator zählt bis n-1
     - n muss dem Initiator bekannt sein
     - Terminierung in dem Sinne, dass alle informiert sind - es können dann aber noch (an sich nutzlose) Basisnachrichten unterwegs sein

2) *Überlagerung* eines geeigneten Kontrollalgorithmus, der die Berechnung des Flooding-Verfahrens beobachtet und die Terminierung meldet
   - später mehr zu überlagerten Terminierungserkennungsverfahren

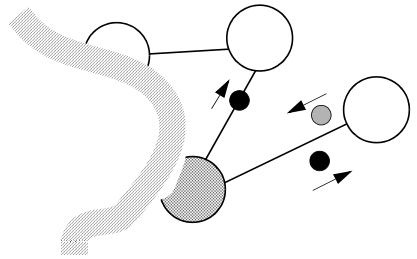
3) *Bestätigungsnachrichten* (acknowledgements)
   - direktes Bestätigen einer Nachricht funktioniert nicht
   - indirekte Bestätigungsnachrichten: ein Knoten sendet erst dann ein ack, wenn er selbst für alle seine Nachrichten acks erhalten hat
   - klappt diese Idee? auch wenn der Graph Zyklen enthält? wieso?

Flooding mit Quittungsmeldungen

(Originalversion des *Echo-Algorithmus* von Chang 1982)

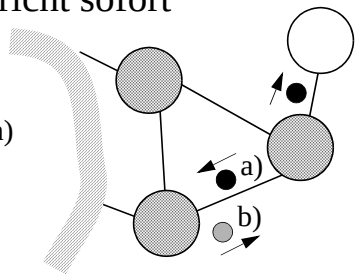
Prinzip: Ein Prozess sendet eine Quittung für eine empfangene Nachricht erst dann, wenn er für alle von ihm selbst gesendeten Nachrichten Quittungen erhalten hat

- Ein Knoten mit Grad 1 ("Blatt") sendet sofort eine Quittung zurück



Nachricht des zugrundeliegenden (Flooding)-Algorithmus

- Ein Knoten, der bereits eine Basisnachricht erhalten hat, quittiert jede weitere Basisnachricht sofort
  - *Prinzip*: "bin schon informiert"
  - *Wirkung*: Zyklen werden aufgebrochen (als wäre die Kante gar nicht vorhanden)
  - *Konsequenz*: es entsteht ein Baum



- Terminiert, wenn Initiator alle Quittungen erhalten hat

- Wieviele Quittungen / Nachrichten insgesamt?

Der Echo-Algorithmus

- Hier: Variante "Propagation of Information with Feedback" (PIF) von A. Segal
- Ähnliches Verfahren 1980 von Dijkstra/Scholten: "Diffusing Computations"

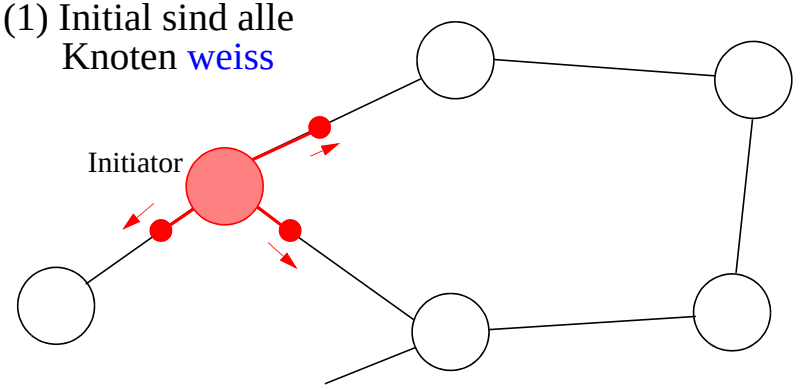
Ausgehend von einem einzigen Initiator:

- Paralleles *Traversieren* eines bel. (zusammenhängenden ungerichteten) Graphen mit 2e Nachrichten
- Terminierung klar durch "Vollzugsmeldung"
- Idee: *Indirektes* acknowledge
- Hinwelle durch "*Explorer*": Verteilen von Information
- Rückwelle durch "*Echos*": Einsammeln einer verteilten Information
- Aufbau eines *Spannbaums* ("Echo-Kanten": jeder Knoten sendet genau ein Echo)

Paralleler *Wellenalgorithmus*:

- virtueller broadcast
- *Basisalgorithmus* für andere Verfahren

(1) Initial sind alle Knoten weiss



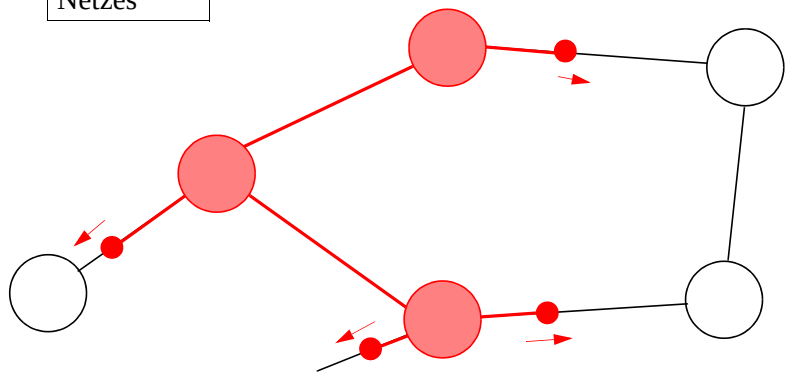
- Der (eindeutige) Initiator wird rot und sendet (rote) *Explorer* über alle seine Kanten

---

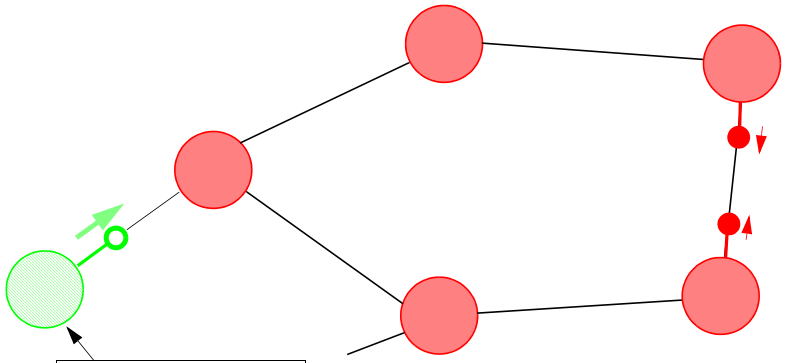
(2) Ein weisser Knoten, der einen Explorer bekommt, sendet Explorer über alle seine anderen Kanten ("flooding") und merkt sich die "erste" Kante

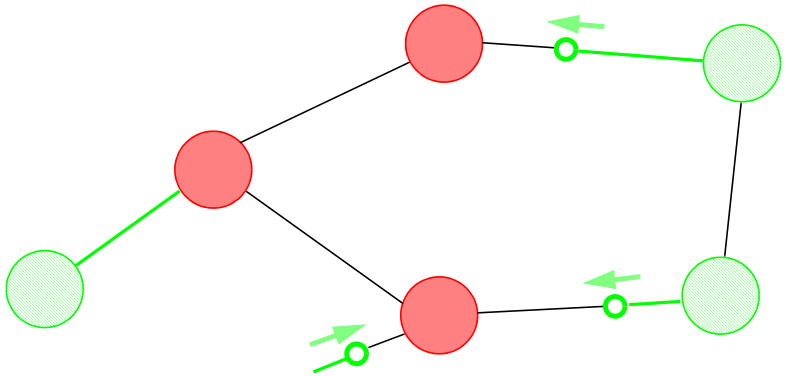
"Fluten" des Netzes

über die ein Explorer eintraf



(3) Ein (roter) Knoten, der über alle seine Kanten einen Explorer *oder* ein Echo erhalten hat, wird grün und sendet ein (grünes) *Echo* über seine "erste" Kante

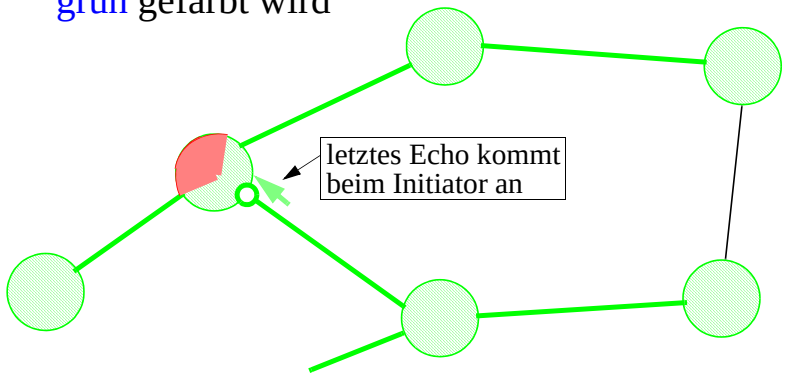




Beachte: *Atomare Aktionen* → Explorer können sich höchstens auf Kanten begegnen, nicht aber bei Knoten!

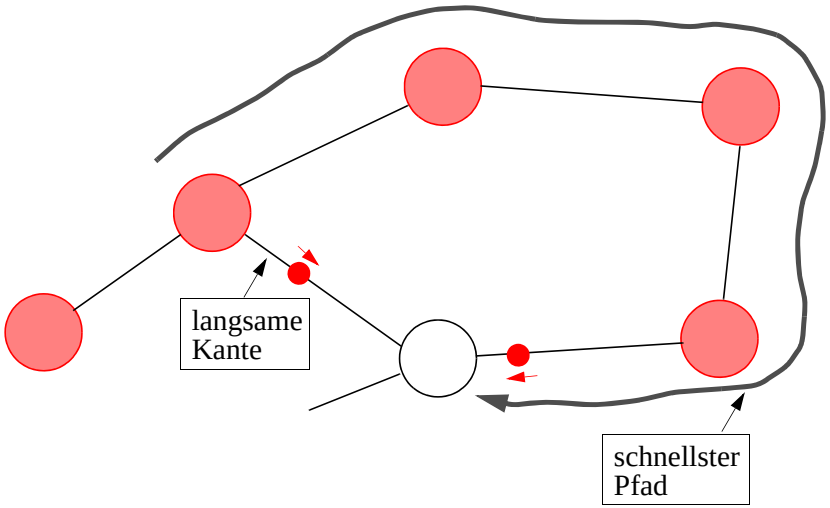
Auf einer Kante, wo sich zwei Explorer begegnen, wird der Zyklus aufgebrochen

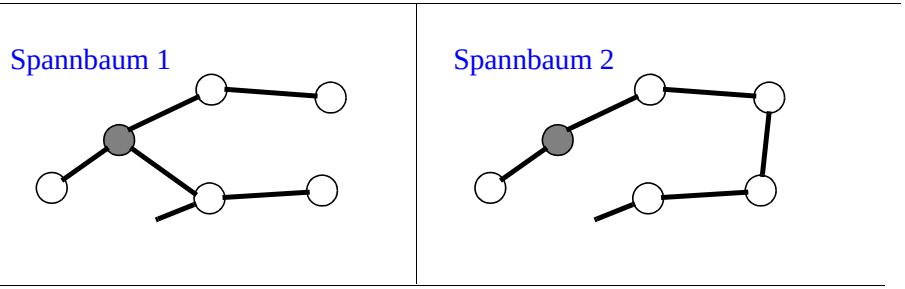
(4) Das Verfahren ist beendet, wenn der Initiator grün gefärbt wird



- Grüne Kanten bilden einen *Spannbaum*
  - alle Knoten bis auf den Initiator haben eine "erste" Kante
  - “grüner Graph” ist zusammenhängend
- Über jede Kante laufen genau 2 Nachrichten:
  - entweder ein Explorer und ein gegenläufiges Echo, oder zwei Explorer, die sich begegnen → Nachrichtenkomplexität = 2e
- Verfahren ist *schnell*: parallel und "bester" (?) Baum
- Ereignis "rot werden" in jedem Prozess definiert eine Welle (*Hinwelle*)
- Ereignis "grün werden" in jedem Prozess → *Rückwelle*
- Es darf nicht mehr als einen Initiator geben:
  - was geschieht sonst?
  - wie kann man dem Problem mehrerer Initiatoren begegnen?

Echo-Algorithmus ist *nicht-deterministisch*, es können (je nach Geschwindigkeit einer "Leitung") *verschiedene* Spannbäume entstehen!





- Inwiefern ist die PIF-Variante besser als die Originalversion?
  - Nachrichtenkomplexität
  - Einfachheit / Eleganz

Echo-Algorithmus...

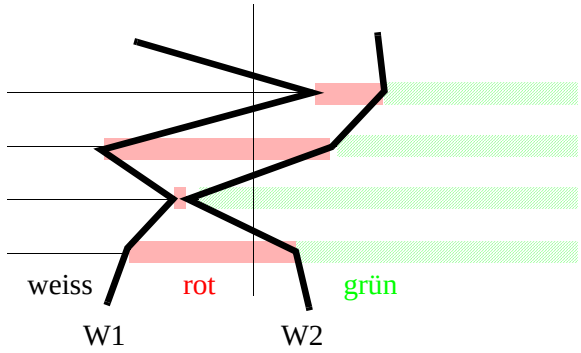
- Jeder Knoten wird erst *rot* und dann *grün*
- Rote Phase ist bei "Blättern" allerdings recht kurz
- Ein grüner Knoten hat keine weisse Nachbarn

⇒ Eine von einem grünen Knoten versendete Basisnachricht wird nicht von einem weissen Knoten empfangen

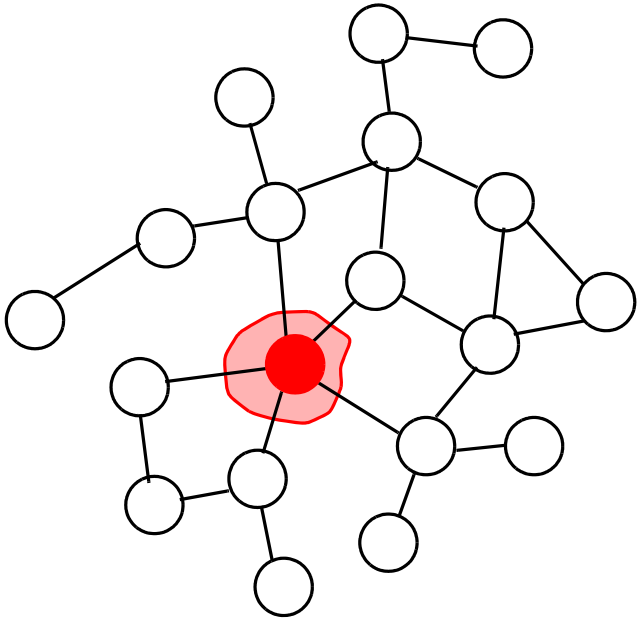
- beachte: gilt nur für *direkte* Nachrichten, nicht für *Nachrichtenketten!*

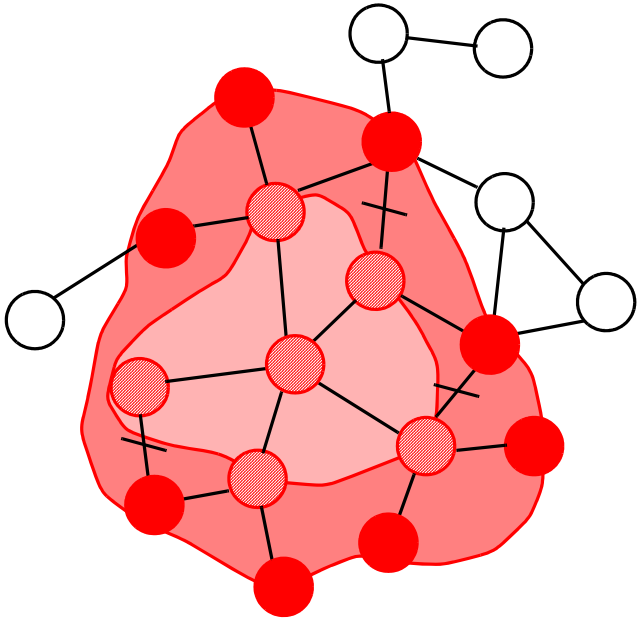
⇒ Weisse und grüne Phase sind in "gewisser Weise" disjunkt

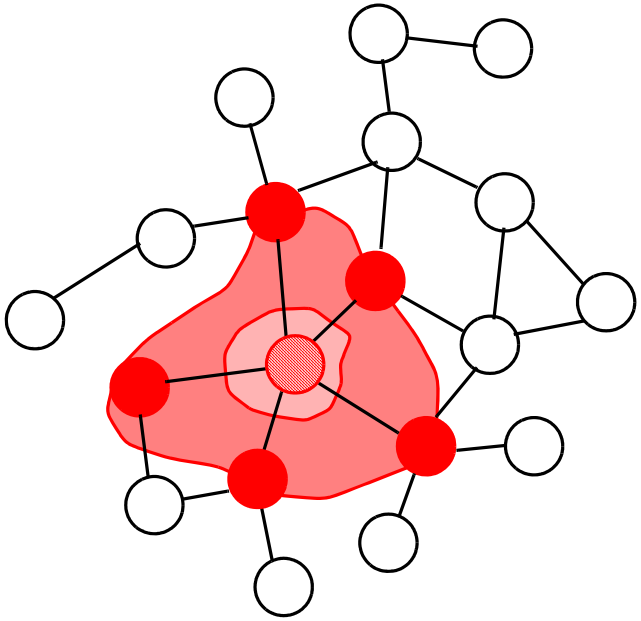
- obwohl es globale Zeitpunkte geben kann, wo ein Knoten bereits grün ist, während ein anderer (nicht direkt benachbarter!) noch weiss ist!

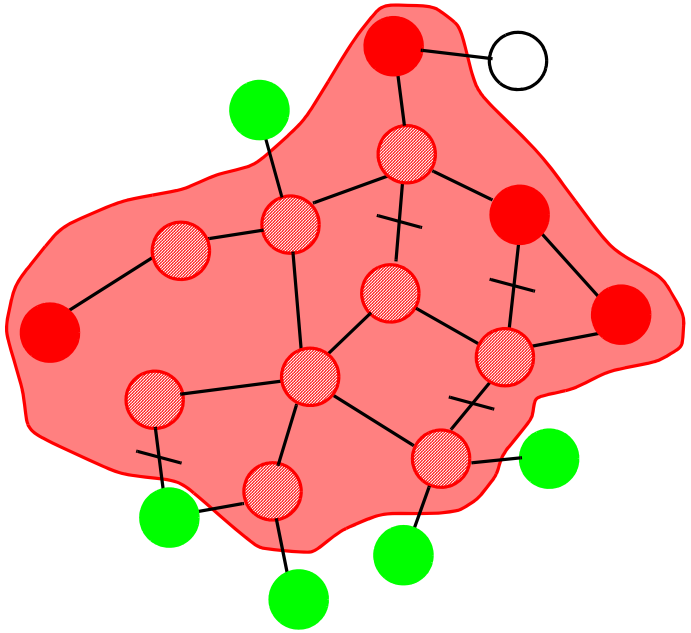


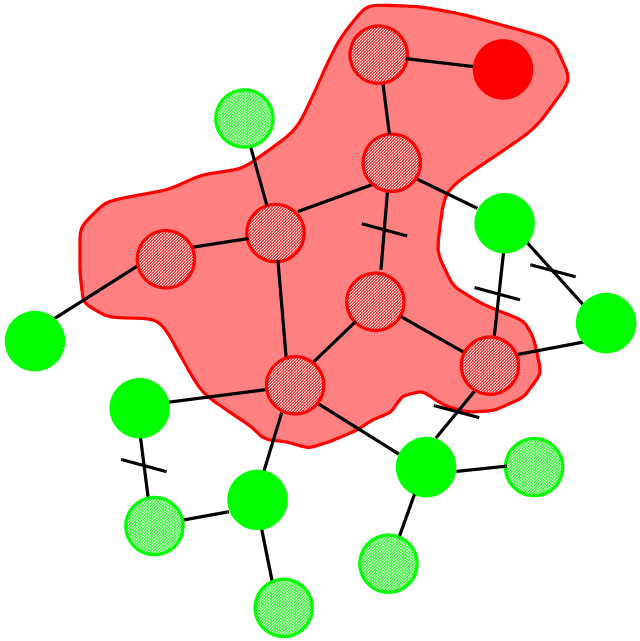
- "Rot werden" und "grün werden" definieren zwei *Wellen*
  - mit diesen Wellen kann Information transportiert werden (verteilen bzw. akkumulieren)
  - Echo-Algorithmus wird daher oft als Basis für andere Verfahren verwendet

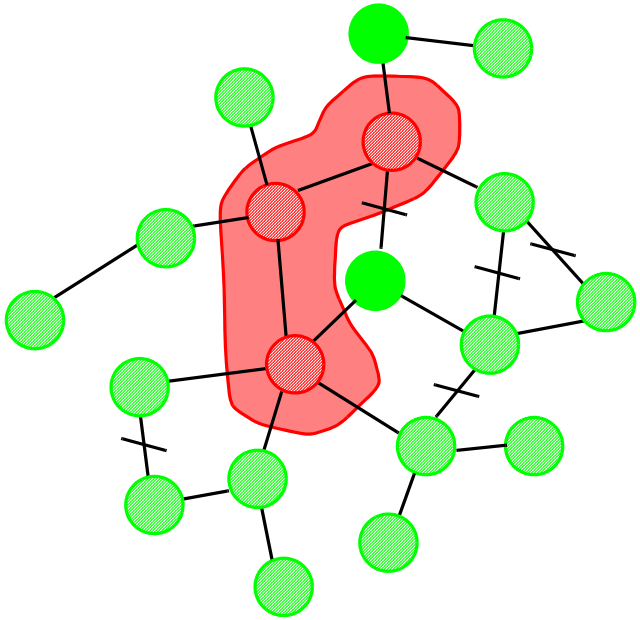


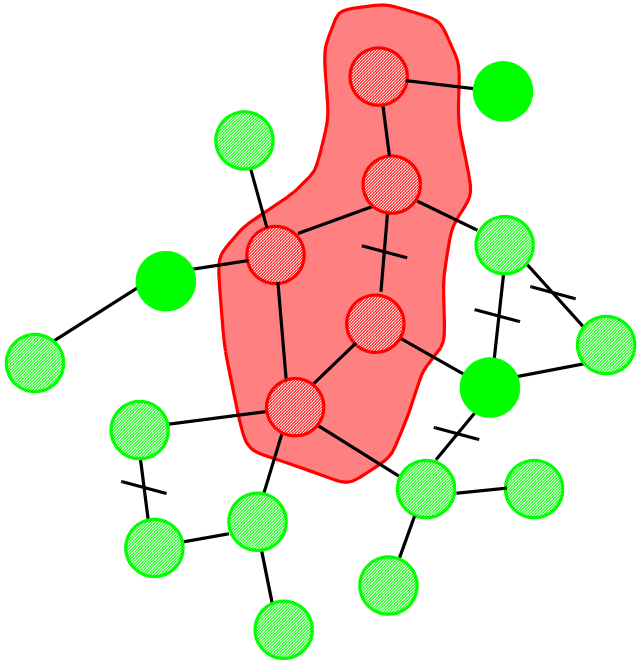


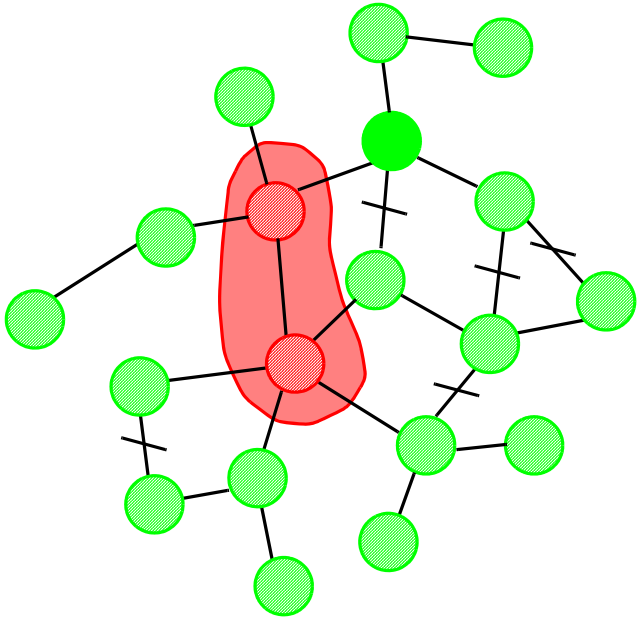


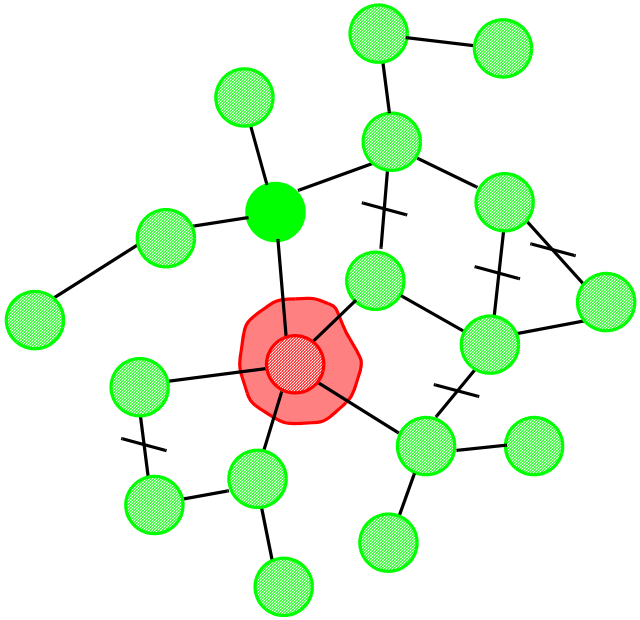


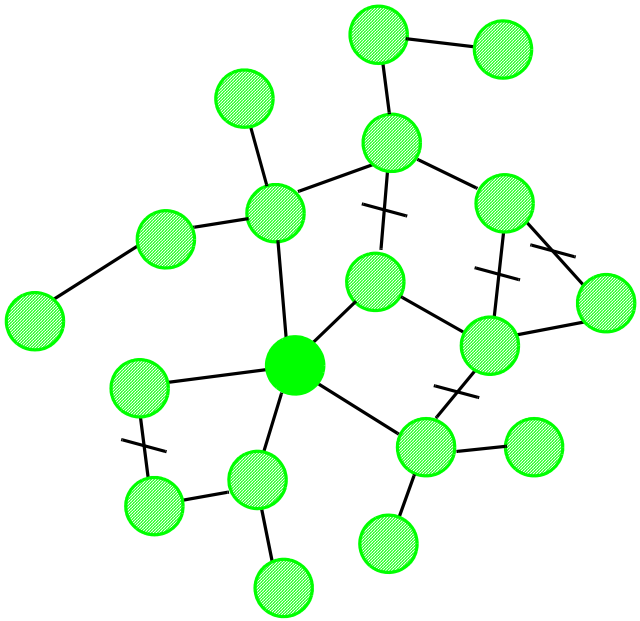








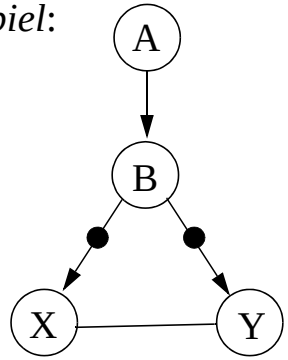




Verbesserung des Echo-Algorithmus?

- *Idee*: vermeide Besuch von Knoten, von denen man weiss, dass sie von anderen Explorern besucht werden

Beispiel:



Nachricht von B an X enthält Information, dass Y nicht besucht zu werden braucht

- *Voraussetzung*: Identitäten der Nachbarn bekannt

Schema (Modifikation gegenüber PIF-Echo):

Menge von "Tabu-Knoten"

```
receive <..., z>
...
y := neighbors \ z
send <..., z ∪ y> to all y
//  Registrieren, über welche Kanäle
//  Echos oder Explorer eintreffen müssen
```

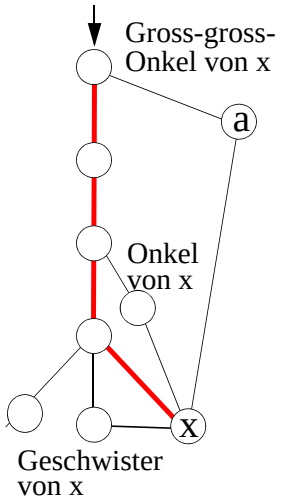
Statt neighbors ohne Vorgänger im Original

- Initiator i: send <..., neighbors ∪ {i} >

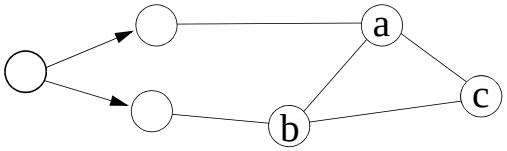
Was wird gespart?

- Keine Nachricht an einen Vorgänger oder einen direkten Nachbarn eines Vorgängers

- Allerdings: Obwohl x nicht an a sendet, sendet a an x! Über diese Kante fliesst dann auch ein Echo zurück → nichts gespart, da 2 Nachrichten über die Kante!



- Auch im folgenden Fall spart man nicht (immer?) etwas:



Knoten a und b werden "gleichzeitig" erreicht: wissen nichts voneinander:
→ senden beide gegenseitig und an c

- Wieviel wird bei vollständigen Graphen gespart? Und bei Bäumen? Spielt der "Vermaschungsgrad" eine Rolle?

- Ersparnis nicht ganz klar
  - interessante Extremfälle, z.B. Baum oder vollständiger Graph

- Ersparnis an Nachrichten durch Nachteile erkauft
  - lange Nachrichten (O(n))
  - Nachbaridentitäten müssen bekannt sein

Das Märchen von der verteilten Terminierung

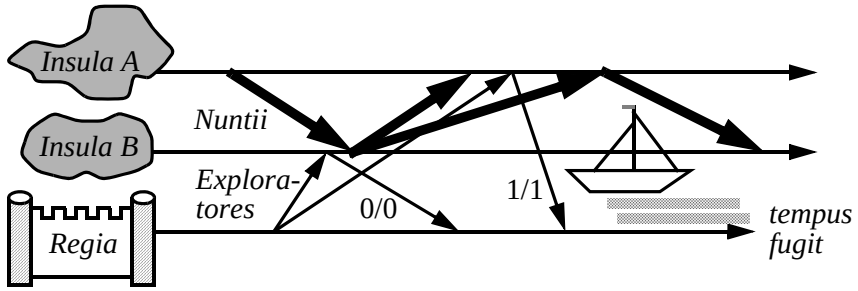
Als einst der König von Polymikronien merkte, dass die Zeit gekommen war, sein aus unzähligen Inseln bestehendes Reich gerecht unter seinen Enkelkindern aufzuteilen, sandte er Botschaften an die weit über das Land verteilt lebenden Weisen aus, auf dass diese ihm einen klugen Vorschlag unterbreiten mögen.

Wohl wusste der König um den eigenwilligen aber steten Lebenswandel seiner Ratgeber, welche den ganzen Tag nichts taten, als zu essen und zu denken: War einer von ihnen bei Speis und Trank, so konnte alleine eine königliche Botschaft oder eine Nachricht eines anderen Weisen ihn zum Denken anregen - er griff meist sogleich zu Feder, Papier, Tinte und Siegel, um einigen übrigen Mitgliedern des so weit verteilten königlichen Konsiliums eine neue Weisheit zuzusenden. Hungrig vom Denken wandt er sich alsbald wieder der stets fürstlich gedeckten Tafel zu.

Als indes die Jahre vergingen und der König immer älter wurde, ohne dass er von den Weisen einen Rat erhalten hätte, seufzte er, schickte nach seinem geheimen Hofrat und sprach zu ihm: "Ich weiss wohl, wie schwierig ein gerechter Plan zur Aufteilung meines Erbes ist, und ich kenne die Regeln meines Konsiliums, wonach man mir erst kundtut, wenn die königliche Sache so erschöpfend beraten wurde, dass ein jeder der Weisen zufrieden ist und keine Botschaft mehr unterwegs ist. Alleine die königliche Post bereitet mir Sorge - ist etwa ein Schiff in den Stürmen der Meere gesunken oder hat sich ein Bote in der Weite des Reiches verirrt?"

"O königliche Hoheit", entgegnete der geheime Hofrat und sprach weiter: "Unermesslich gross ist Euer Reich, und gar lange brauchen die Segler, um von einem Eiland zu einem anderen zu gelangen. Rein niemand vermag die Zeit vorher abzuschätzen, und es wird sogar berichtet, dass in der ein oder anderen Nacht ein Postboot ein anderes überholt. Aber die Segelkünste der Seefahrer, die hohe Schule der Schiffsbaumeister und die pflichtbewusste Ergebenheit Eurer Diener sorgen dafür, dass nicht eine einzige Botschaft verloren gehen kann. Lasset uns also Kundschafter aussenden, mein König, um jeden Ratgeber zu befragen, wieviele Botschaften er empfangen und versandt hat. So können wir leicht Gewissheit darüber erlangen, ob summa summarum so viele Nachrichten versandt wie empfangen wurden und das Konsilium des Königs Sache abschliessend beraten hat."

Fortsetzung→



Übungen (2) zur Vorlesung "Verteilte Algorithmen"...

a) Man beweise die Korrektheit des im Märchen beschriebenen "Doppelzählverfahrens" zur Feststellung der verteilten Terminierung.

b) Man beweise die Korrektheit des Echo-Algorithmus:
- der Initiator terminiert erst, wenn alle Knoten informiert wurden ("safety"),
- nach endlicher Zeit terminiert der Initiator ("liveness").

Überlegen Sie sich, was für Beweistechniken Sie einsetzen können (Invarianten, Induktion...) und wie genau / formal die Spezifikation des Algorithmus sein sollte, damit Sie im Beweis formal argumentieren können. Geben Sie ggf. eine formalere Spezifikation des Algorithmus an.

→ *Fortsetzung Märchen*

Der König war hoch erfreut über diese weisen Worte, schöpfte neue Zuversicht und beauftragte sogleich den Hofmathematicus, einen Plan auszuarbeiten. Dieser erschien alsbald mit einer grossen Leinwand und sprach: "Majestät, auf diesem Szenario seht Ihr, dass des Hofrats Plan versagen kann - die zu den Eilanden A und B gesandten Kundschafter berichten, dass so viele Botschaften empfangen wie versandt wurden - summa summarum nur eine Botschaft. Nichtsdestotrotz sind noch Nachrichten unterwegs. Die Sache ist wohl so, dass die Kundschafter sehr klug vorgehen müssen, um sich nicht täuschen zu lassen und des Königs Zählung zu verfälschen, alldieweil wir primo die Uhr noch nicht erfinden konnten und somit auch keine einheitliche Reichszeit haben, secundo wir den Rundfunk noch nicht kennen und tertio die ehrwürdigen Sitten es verbieten, dass die Weisen an einem gemeinsamen Ort zusammenkommen." Der König war erstaunt über die gar wundersamen Worte und meinte: "Nun, das weiss ich wohl, denn ich bin der König. Was also rät Er mir ?" "Hoheit, mein Plan sieht vor, die Kundschafter erneut auszusenden, sobald der letzte bei Hofe eingetroffen ist. Wird alsodann das Ergebnis genau bestätigt und sind die Summen gleich, so ist keine Nachricht mehr unterwegs." "Vortrefflich", meinte der König, der nichts verstanden hatte. "Kümmere Er sich nur sogleich um die Instruktion der Kundschafter!"

Der Hofmathematicus tat wie befohlen, verbesserte seinen Plan noch verschiedentlich, und bald waren die Kundschafter mit allen königlichen Vollmachten versehen auf den besten Seglern des Reiches unterwegs zu den Weisen.

Als endlich der Schluss der Weisen bei Hofe eintraf, war der König so voll Glück, dass er den Hofmathematicus bald darauf zu seinem ersten Hofinformaticus ernannte. Und dieser forscht, wenn er nicht gestorben ist, noch heute in einem stillen Turm des königlichen Palastes... an einer noch besseren Lösung zum Problem der verteilten Terminierung!

Papier war kostbar - so verzichtete der Hofinformaticus leider darauf, einen Korrektheitsbeweis seines Planes niederzuschreiben. Einer Randbemerkung seiner Schriften entnehmen wir, dass er später ein Verfahren ersann, bei dem nicht in jedem Fall die Inseln zwei- oder mehrfach von den Kundschaftern aufgesucht werden müssen. Desweitern gelang es ihm offenbar, die Zahl der notwendigen Kundschafterfahrten durch eine einfache Funktion der Zahl der Inseln und Botschaften zu beschränken. Leider reichte wiedereinmal der Rand zur Darstellung der Methode nicht aus. Wer hilft mit bei der Rekonstruktion und Verifikation der Verfahren ?

Zeitkomplexität

Beachte: Algorithmen oft nichtdeterministisch → *mehrere* mögl. Berechnungen!

Variable Zeitkomplexität eines vert. Algorithmus = max. "Zeit" aller Berechnungen des Algo unter:

Z1: Lokale Berechnungen erfolgen in Nullzeit
Z2: Eine Nachricht benötigt *maximal* 1 Zeiteinheit

Einheitszeitkomplexität eines vert. Algorithmus = max. "Zeit" aller Berechnungen des Algo unter:

E1: Lokale Berechnungen erfolgen in Nullzeit
E2: Eine Nachricht benötigt *exakt* 1 Zeiteinheit

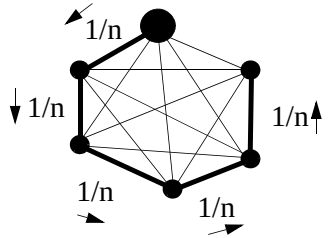
Behauptung:
Es gilt *nicht* immer: variable Zeitkplx ≤ Einheitszeitkplx

- Grund: Einheitszeitkomplexität erlaubt nicht alle Berechnungen!
- Frage: Gilt Umkehrung?

Bsp. Echo-Algorithmus auf vollständigem Graph

(1) Einheitszeitkomplexität = 3 ← Phase 1: Alle werden rot
Phase 2: Alle bis auf Initiator werden grün
Phase 3: Initiator wird grün

(2) Variable Zeitkomplexität ≥ n

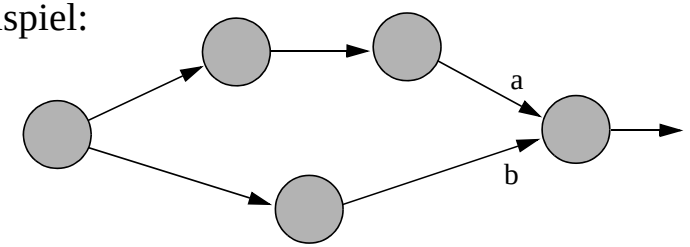


- Explorer "aussen": 1/n Zeiteinheiten
- Jede sonstige Nachricht 1 Zeiteinheit
- Entarteter Baum Tiefe n-1 nach einer Zeiteinheit aufgebaut
- Echo beim Initiator nach n Einheiten

Zeitkomplexität: Welche Definition?

- *Einheitszeitkomplexität*: Einige Berechnungen bleiben unberücksichtigt!
  - Nicht bei var. Ztkplx! (Wieso?)
  - unwahrscheinliche?

Beispiel:



Trifft a vor b ein → sehr lange Berechnung, sonst terminiert

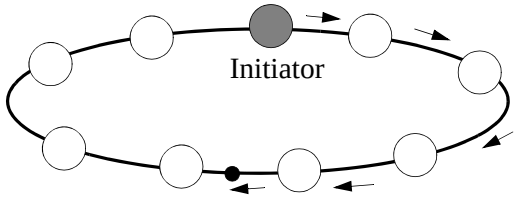
- mag vielleicht in 10% aller Fälle der Fall sein...
- aber wie oft wirklich?
- systemabhängig!

- *Variable Zeitkomplexität*: Resultat wird u.U. durch extrem unwahrscheinliche Berechnungen bestimmt
- Für worst-case: variable Ztkplx? Aber: average case?
- Genauer: Wahrscheinlichkeitsverteilung → Erwartungswert
  - systemabhängig
  - schwierig
  - jeden Tag anders...

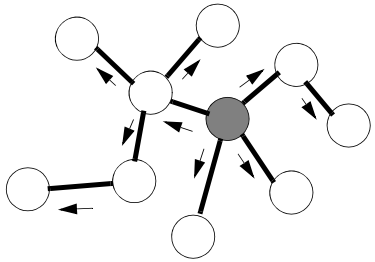
Broadcast auf speziellen Topologien

- Echo-Algorithmus realisiert einen Broadcast
  - auf bel. zusammenh. Topologie
  - Verteilen von Information ausgehend von einem Initiator
  - liefert sogar "Vollzugsmeldung" durch Echo-Nachrichten

- Auf *speziellen* Topologien lässt sich der Broadcast auch effizienter realisieren
  - Beispiel *Ring*: Ein "Token" zirkuliert mit der Information; alle sind informiert, wenn das Token wieder beim Initiator eingetroffen ist
  - Evtl. kann einer anderen Topologie ein Ring überlagert werden



- Beispiel *(Spann)baum* (tatsächlich Unterschied zum Echo-Algorithmus?)



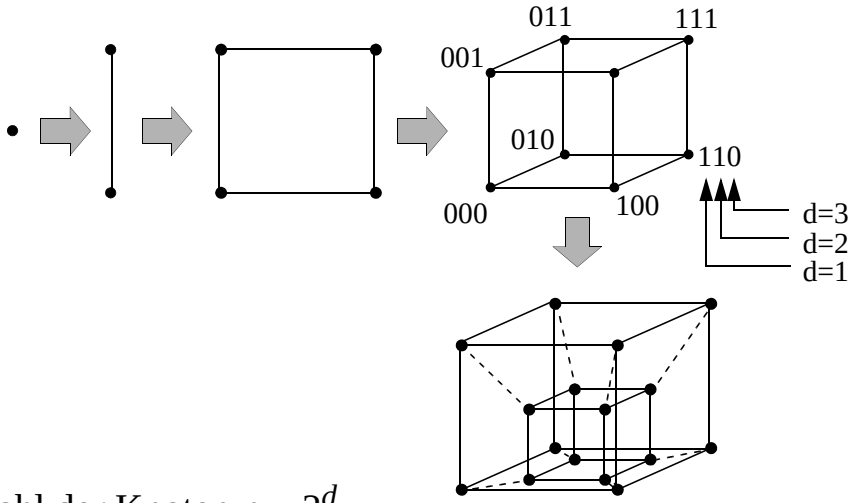
Vorausgesetzt wird jeweils, dass der Algorithmus "weiss", dass eine spezifische Topologie vorliegt!

- Beispiel *vollständiger Graph* (als Denkübung)

Hypercubes

- Hypercube = "Würfel der Dimension d"
- Rekursives Konstruktionsprinzip
  - Hypercube der Dimension 0: Einzelrechner
  - Hypercube der Dimension d+1:

    „Nimm zwei Würfel der Dimension d und verbinde korrespondierende Ecken“



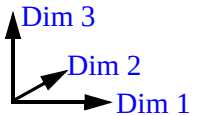
- Anzahl der Knoten $n = 2^d$
- Anzahl der Kanten = $d\ 2^{d-1}$ (Ordnung O(n log n))
  - viele Wegalternativen (Fehlertoleranz, Parallelität!)
  - maximale Weglänge: d = log n
  - mittlere Weglänge: d/2 (Beweis als Denkübung!)

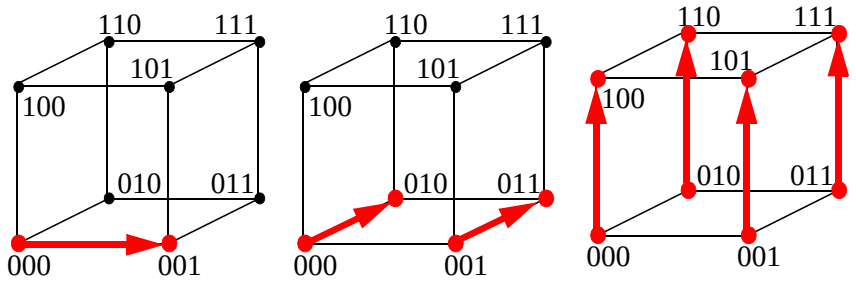
wieviele verschiedene Wege der Länge k gibt es insgesamt?

- Knotengrad = d (nicht konstant bei Skalierung!)
- Einfaches Routing von einzelnen Nachrichten
  - xor von Absende- und Zieladresse...

Broadcast in Hypercubes (1)

- Initiator habe die Nummer 00...00 (binär)
- Wir verzichten hier auf Vollzugsmeldung
  (also keine Acknowledgemnents oder Endeerkennung)





- Analog zum rekursiven Aufbau des Hypercube:
  - zunächst in Dimension 1 senden: Teil-Hypercube der Dimension 1 ist damit informiert
  - dann senden alle Knoten der Dimension 1 in Dimension 2
  - dann Dimension 3 etc.
- Nach d "Takten" sind alle Knoten informiert
  - Zeitkomplexität ist daher d (unter welchem Zeitmass?)
  - Nachrichtenkomplexität: $1 + 2 + 4 + ... + 2^{d-1} = 2^{d-1} - 1 + 2^{d-1} = n-1$
    (jeder Knoten, ausgenommen der Initiator, erhält genau eine Nachricht)
- Welche Komplexität hat ein *optimaler* Broadcast-Algo.?
- Geht es besser?

was heisst überhaupt "besser"?

  - Algorithmus startet ziemlich "langsam": am Anfang geschieht wenig parallel!
  - Kann man dies durch *gleichzeitiges* Versenden "in alle Richtungen" beschleunigen?

Broadcast in Hypercubes (2)

- Ein anderes Verfahren (Vergleich als Denkübung!)
- Initiator sendet an alle seine Nachbarn:

  0...01, 0...010, 0...100, ... , 10...0

  in "kanonischer" Numerierung

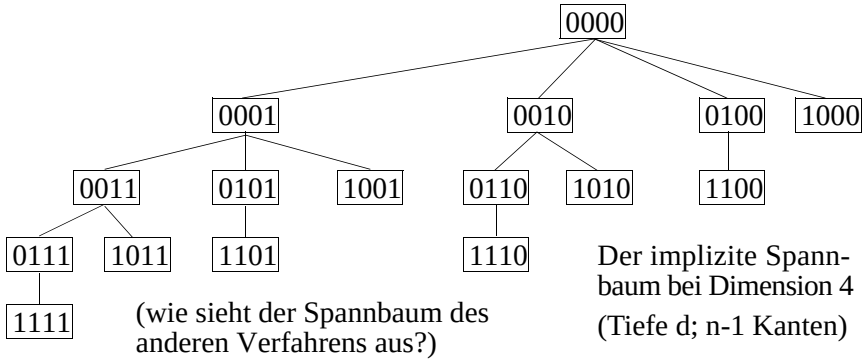
  am besten gleichzeitig, wenn dies technisch geht!

- Ein Knoten mit der Nummer 0...01x...y...z leitet die Information an alle seine "höheren" Nachbarn weiter:

  linkeste 1 | beliebiges Restmuster

  0...00**1**1x...y...z
  0...0**1**01x...y...z
  0...**1**001x...y...z
  ...
  10...001x...y...z

  Von welchem (eindeutigen) Knoten A wird Knoten B informiert?
  Setze *vorderste 1* von B auf 0
  → = Nummer von A



(wie sieht der Spannbaum des anderen Verfahrens aus?)

Der implizite Spannbaum bei Dimension 4
(Tiefe d; n-1 Kanten)

- Der Algorithmus wird z.B. in Mehrprozessorsystemen verwendet
- Wie effizient ist der Algorithmus? (Geht es besser?)
- Denkübung: Formuliere Algorithmus für einen beliebigen Initiator (schliesslich sind Hypercubes symmetrisch...)
- Denkübung: Vergleich mit Flooding bzw. Echo-Algorithmus

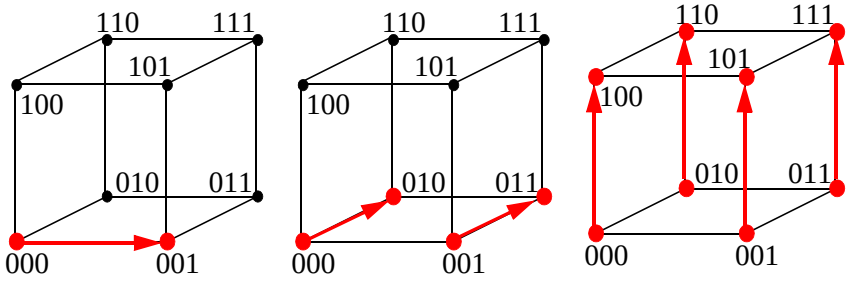
Noch ein anderer (besserer?) Algorithmus

- Beobachtungen:
  - Ein Baum verwendet im Hypercube relativ wenig Kanten → schlechte Ausnutzung potentielle Parallelität
  - Es gibt *mehrere* Spannbäume in Hypercubes → diese nutzen?

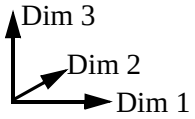
- Sender 0...0 teilt die Nachricht in d Pakete
- Sender startet für jedes Paket eine eigene "Welle":
- *1. Paket* in *Dimension 1* senden → 0...01
- Dann: Alle informierten Knoten (also 0...0 und 0...01) senden dieses Paket in Dimension 2
- Etc. Welle für Paket 1 breitet sich analog zur rekursiven Definition des Hypercubes in einer jeweils zusätzlichen Dimension aus
- Das *2. Paket* wird erst in *Dimension 2*, dann 3,..., d und erst zuletzt in Dimension 1 gesendet
- Das *3. Paket*: Dimensionsreihenfolge 3, 4, ..., d,1, 2
- Etc.: das *d.-Paket* in Dimensionsreihenfolge d, 1, 2,..., d-1

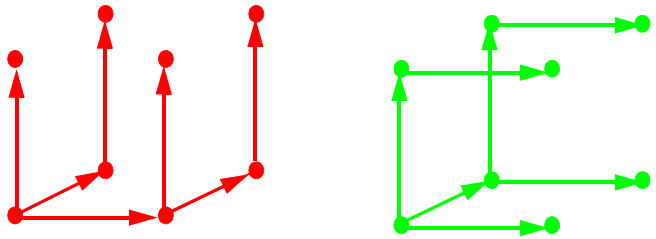
Denkübungen:
- Können so die Pakete gleichzeitig verschickt werden?
- Ist dann in jedem "Takt" pro Kante nur eine Nachricht unterwegs?
- Wieviele (kantendisjunkte ?) Spannbäume gibt es in einem Hypercube?

Veranschaulichung des Algorithmus



Die drei “Takte” der Welle von Paket 1

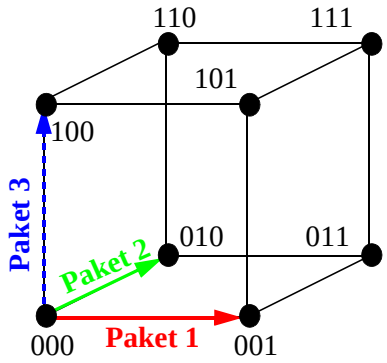




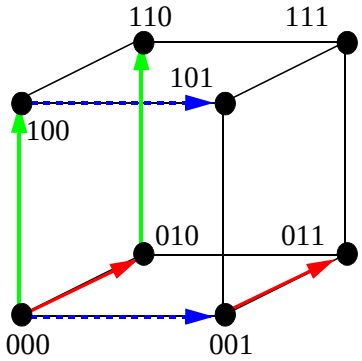
Die Spannbäume bzgl. Paket 1 und Paket 2

Parallelausführung der drei Wellen

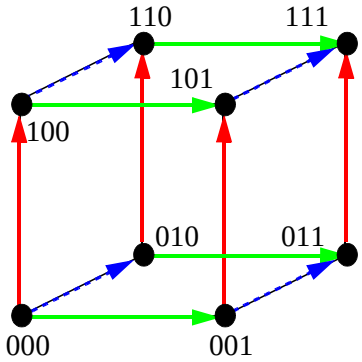
Takt 1



Takt 2



Takt 3



Takt
1 2 3

Paket 1: 1. 2. 3. Dim.

Paket 2: 2. 3. 1. Dim.

Paket 3: 3. 1. 2. Dim.

Pro Takt laufen die Pakete in jeweils unterschiedlicher Dimension!

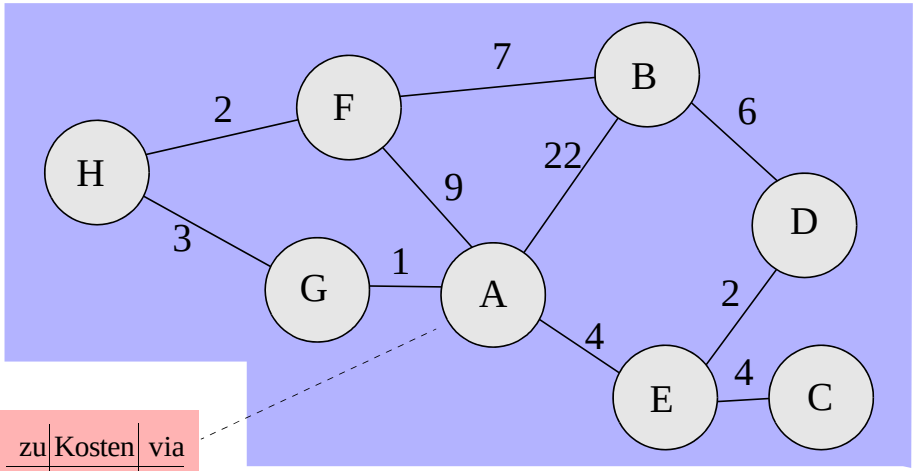
Es können also tatsächlich die drei Wellen parallel ausgeführt werden, ohne dass diese sich gegenseitig stören!

→ Dies ist das (im Prinzip) schnellere Verfahren!

Beachte: Ein global synchronisierter Takt ist gar nicht nötig!

Verteilte Berechnung von Routingtabellen für kürzeste Wege

Gegeben: ungerichteter zusammenhängender Graph
mit bewerteten Kanten (Kosten, Länge...)



zu	Kosten	via
A	0	-
B	22	B
C	∞	?
D	∞	?
E	4	E
F	9	F
G	1	G
H	∞	?

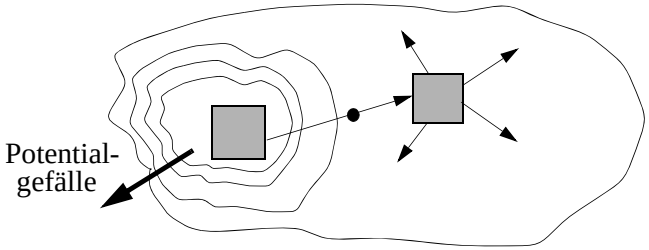
Anfangs-tabelle für Knoten A

- Jeder kennt anfangs die Kosten zu seinen Nachbarn
- "Spontanstart": Sende eigene Tabelle an Nachbarn
- Bei Empfang einer Tabelle über Verbindung mit Kosten g:
  Für alle Zeilen i der Tabelle:
  Falls Nachricht.Kosten[i]+g < Knoten.Kosten[i]:
  ersetze Zeile (Kosten := Kosten+g; via := Absender)
- Falls sich Tabelle verändert hat:
  Neue Tabelle an alle Nachbarn (Ausnahme: Sender)
- Wie Terminierung feststellen?

- Ist eine verteilte Version des Bellman-Ford-Algorithmus
  - ähnlich dem bekannten Dijkstra-Algorithmus für kürzeste Wege
  - “Relexationsprinzip” (Bellman 1958, Dijkstra 1959, Ford 1962)

Das Paradigma der vert. Approximation

Prinzip:
- Anfang: Informiere alle Nachbarn spontan
- Bei Empfang einer Nachricht:
  - berechne neue Approximation
  - falls diese "besser": informiere Nachbarn



- *Nachrichtengesteuert* (aber "Spontanstart")
- Alle Prozesse arbeiten gleich, alle sind beteiligt
- *Nichtdeterministischer* Ablauf, determin. Ergebnis
- Beliebige stark zusammenhängende Topologie
- Assoziative Operatoren (min, max, ∩, ∪, +, and, or, ...)
- *Stagnation* bei globalem Gleichgewicht ("Optimum")
  → Potentialunterschiede ausgeglichen
  → Terminierungsproblem

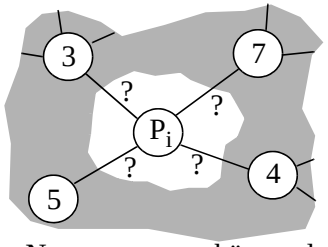
Beispiele ("Instanzen der Algorithmenklasse"):
- ggT
- Zahlenrätsel
- Verteilen von Information ("Wissensausgleich")
- Routingmatrizen (inkl. Spannbaum)
- Maximale Identität ("election")
- Lastausgleich (Approx. eines dyn. Optimums)
- Relaxationsverfahren (Lösen von DGL)

(noch) nicht behandelt

Nachbarschaftswissen

- Zweck: Lerne Identitäten Deiner Nachbarn kennen
  (Idee: "Ich heisse i, wie heisst Du?")

- Hier nachrichtengesteuert für P_i:
  (vgl. verteiltes Approximationsschema)



Netz zusammenhängend
mit bidirektionalen Kanten

```
{Nachricht <j> kommt an}
if not vorgestellt then
    send <i> to all neighbors;
fi;
Nachbarn := Nachbarn ∪ {j};
vorgestellt := true;
```

mehrere?

Einer muss "spontan" mit
der Vorstellung beginnen

- *global terminiert*, wenn
  - vorgstellt = true bei allen Prozessen und keine Nachricht unterwegs
  - *oder*: alle kennen alle

- *lokal terminiert* (genügt meistens!), wenn zu jeder inzidenten Kante die zugehörigen Nachbarknoten bekannt sind

- 2e Nachrichten (bei Ring also 2n); Zeitkomplexität?

Verwendung als:

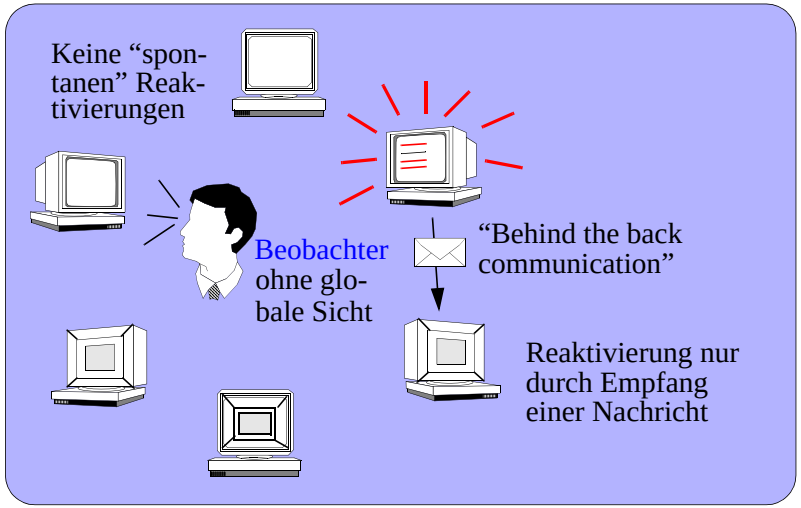
- vorgeschalteter eigener Algorithmus
- "piggybacking" der Vorstellungsnachrichten mit eigentlichen (jew. "ersten") Anwendungsnachrichten

Verteilte Terminierung

Das Problem der Terminierung

- Veranschaulichung: z.B. verteilter ggT auf PC-Cluster
  - pro "Prozess" jeweils ein Display
  - dort jeweiligen Zustand und neue Wertemengen anzeigen

aktiv oder *passiv*

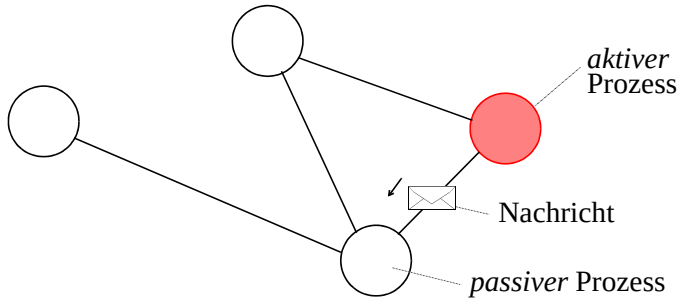


Erkennung der verteilten Terminierung?

alle passiv und keine Nachricht unterwegs

Erkennung der verteilten Terminierung

- Abstrakteres Modell, das dem ggT-Beispiel (und weiteren Beispielen) zugrunde liegt:



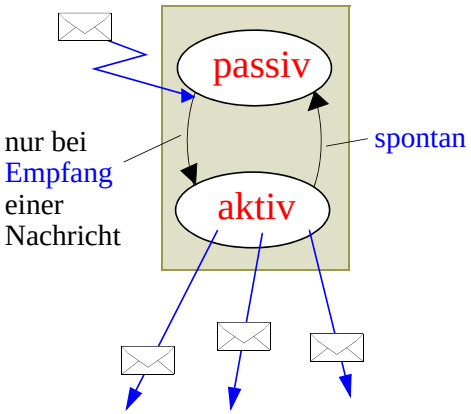
- Es gibt aktive und passive Prozesse sowie Nachrichten
  - dabei kann eine ankommende Nachricht einen passiven Prozess wieder reaktivieren

- Man möchte "irgendwie" erkennen, ob *alle* Prozesse *passiv* sind und *keine Nachricht* mehr *unterwegs* ist
  - Stagnationszustand beim verteilten Approximationsparadigma

Verteilte Terminierung: Modell und Problemdefinition

Nachrichtengesteuertes Modell einer vert. Berechnung:

- Prozesse sind *aktiv* oder *passiv*
- Nur aktive Prozesse versenden Nachrichten
- Prozess kann "spontan" passiv werden
- Prozess wird durch ankommende Nachricht reaktiviert



Problem:

Feststellen, ob (zu *einem* Zeitpunkt)
- alle Prozesse passiv sind
- keine Nachricht unterwegs ist

- "globales Prädikat"
- "stabiler Zustand"

Die Aktionen der Basisberechnung im nachrichtengesteuerten Modell

Prozess p:

"guard": Prädikat über dem lokalen Zustand

S_p: {Zustand = *aktiv*}
send message <M> **to** ...

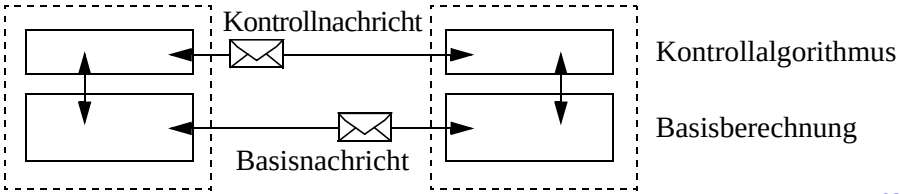
R_p: {Eine Nachricht ist angekommen}
receive <M> ; Zustand := *aktiv*

I_p: {Zustand = aktiv}
Zustand := *passiv*

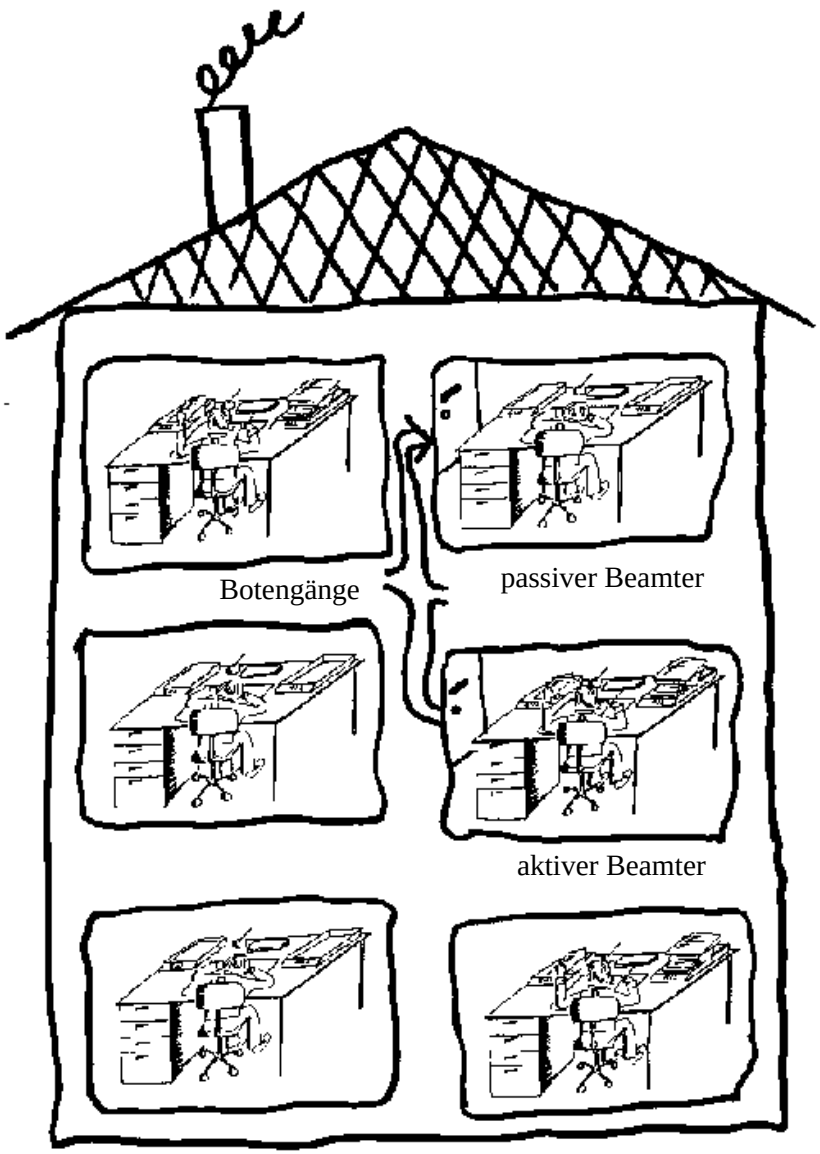
Typischerweise werden die Aktionen als "atomar" betrachtet

Abstraktes Verhalten einer verteilten Berechnung hinsichtlich Terminierung (evtl. existieren weitere Aktionen)

- Durch einen "überlagerten" Kontrollalgorithmus werden weitere Aktionen hinzugefügt
- "Anreichern" der Basisberechnung für Zwecke des Kontrollalgorithmus
  - z.B. Verändern spezifischer (lokaler) Variablen
- Überlagerter Algorithmus soll Basisberechnung nicht stören
  - darf aber gemeinsame Variablen, die der lokalen Kommunikation mit dem Basisalgorithmus dienen, lesen und schreiben

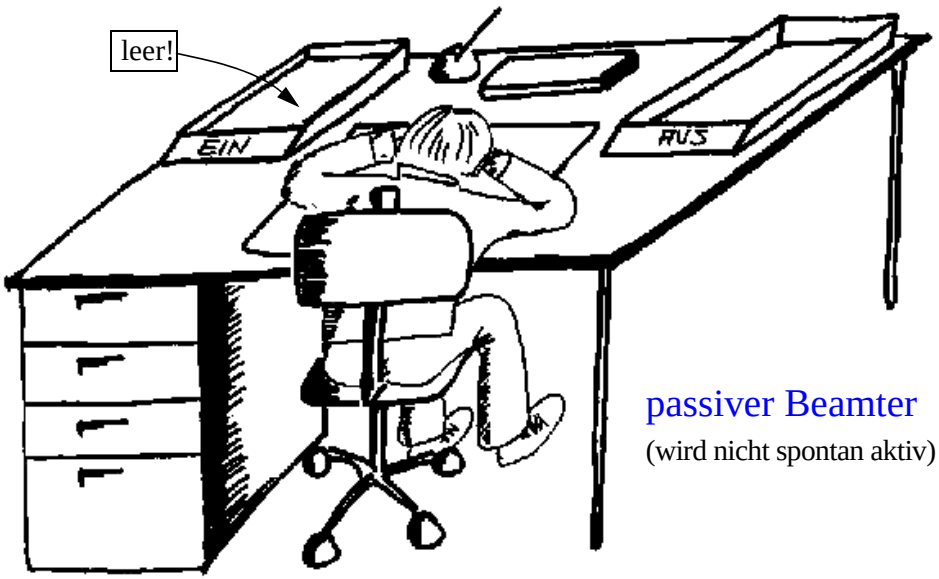


Eine typische Behörde



aktiver Beamter

Die Funktionsweise der Behörde



passiver Beamter
(wird nicht spontan aktiv)

(1) Publikumsverkehr nur bis 12 Uhr
(2) Schliesst erst, wenn alle Vorgänge bearbeitet
(3) Vorgänge werden von Beamten erledigt
(4) Die Bearbeitung eines Vorganges *kann* neue Vorgänge für andere Beamte auslösen
(5) Aktenaustausch per (bel. langsame) Boten
(6) Keiner hat den Gesamtüberblick
(7) Beamte sind *aktiv* oder *passiv*
(8) Ein Beamter wird nicht spontan aktiv

Terminiert, wenn alle passiv und nichts "unterwegs"

Das ist ein stabiler Zustand!

Variante: Beamter lässt sich während der Arbeit nicht stören (anklopfen/warten auf "herein") →

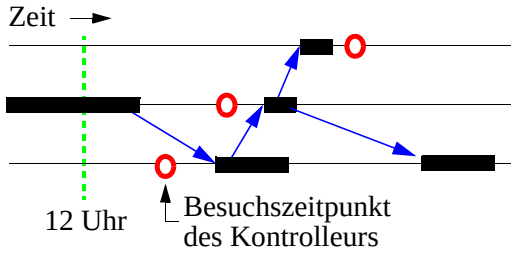
! → Beamte *scheinen* immer passiv (Atommodell)

Das schiefe Bild des Kontrolleurs

- *Kontrolleur* wandert durch die Behörde, um die Terminierung feststellen zu können

- *Problem*: Wie stellt der Kontrolleur fest, ob der stabile Terminierungszustand eingetreten ist?

- Die *Illusion* Kontrolleurs:

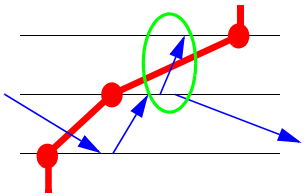


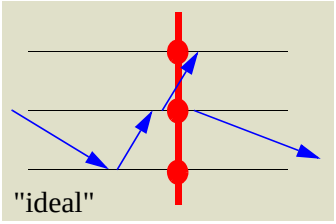
"behind the back communication"

- Alle Beamten stets passiv

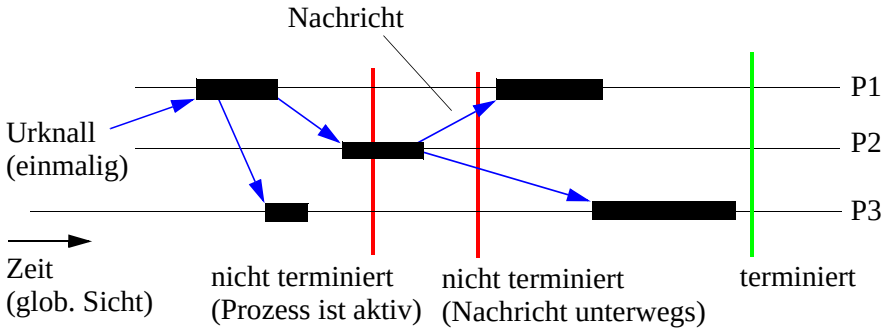
- Σ Nachrichten versendet = Σ Nachrichten empfangen
(jedenfalls nach dem Eindruck des Kontrolleurs)

- Kontrolleur macht sich ein *schiefes Bild!*



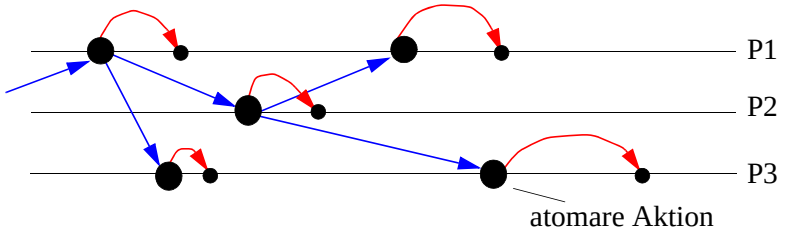


Zeitdiagramme und Atommodell



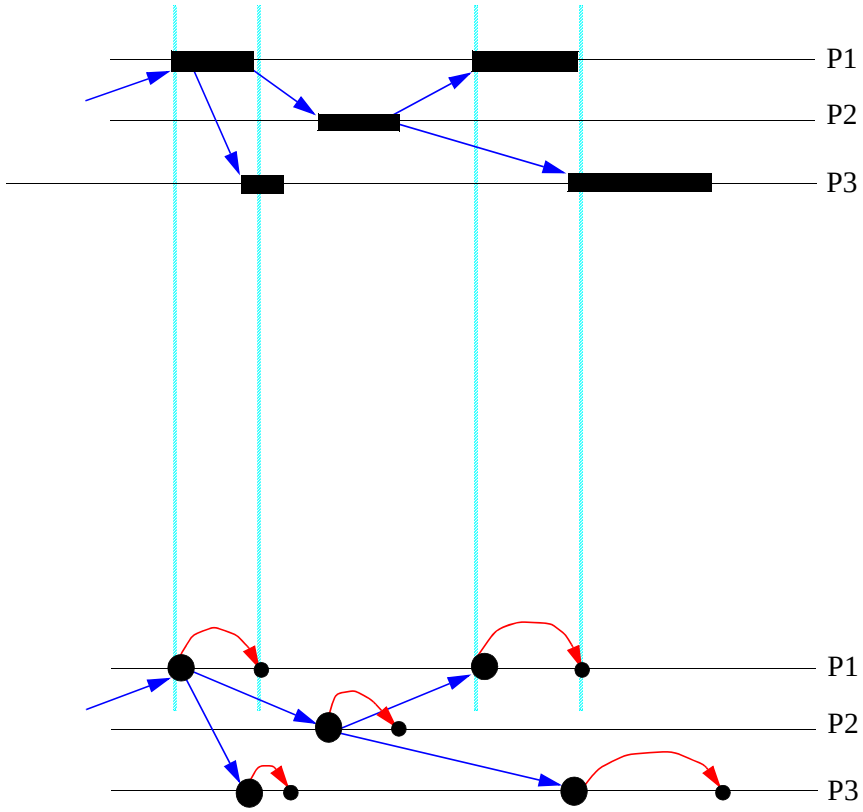
Idee: Dauer der Aktivitätsphasen "gegen Null" gehen lassen

Modellierung: Prozess sendet (virtuelle) Nachricht an sich selbst, sobald er aktiv wird; ist "unterwegs", solange er aktiv ist

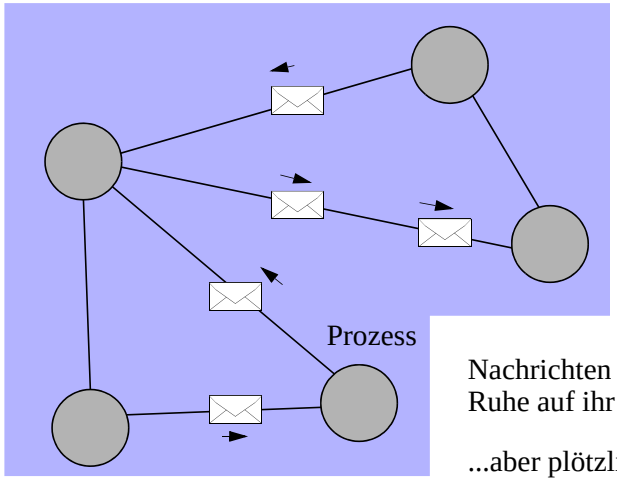


Terminiert (Atommodell) $\Leftrightarrow$
Keine (echte oder virtuelle) Nachricht unterwegs

Zur Lösung des Terminierungsproblems also feststellen, *ob* noch Nachrichten unterwegs sind



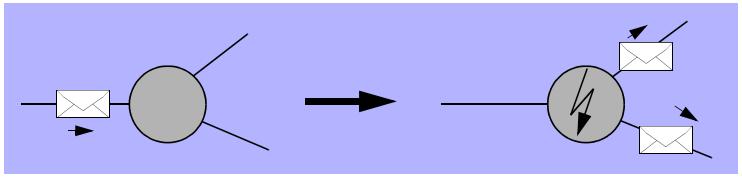
Globale Sicht "atomarer" Berechnungen



idealisierter Beobachter

Nachrichten fliessen in aller Ruhe auf ihr Ziel zu...

...aber plötzlich "explodiert" ein Prozess, wenn er von einer Nachricht getroffen wird!



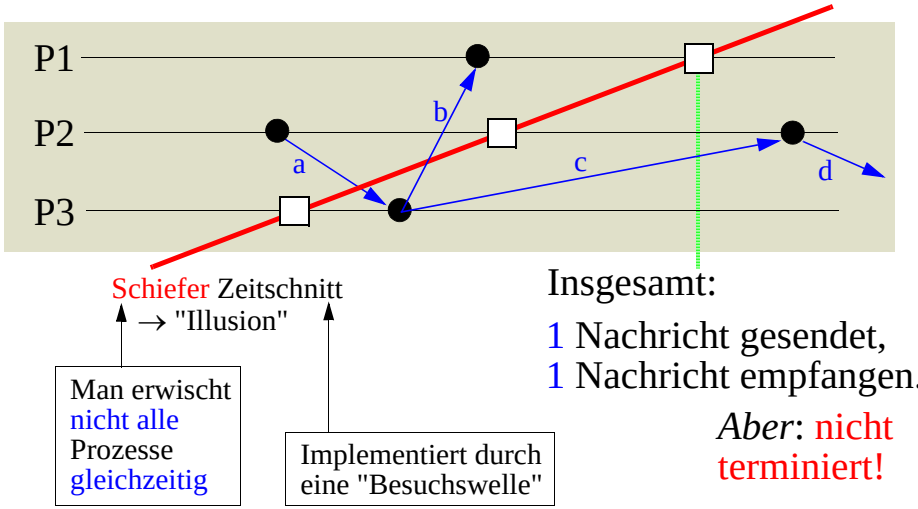
Terminiert, wenn in der globalen Sicht kein ✉ existiert

- Statt im "passiv/aktiv-Modell" genügt es offenbar, im Atommodell die Terminierungserkennung zu lösen (wieso?)

Verteilte Terminierung: Lösungen durch Zählen von Nachrichten?

- Genügt das (verteilte) Zählen von gesendeten und empfangenen Nachrichten?

- Einfaches Zählen genügt nicht, Gegenbeispiel:



Insgesamt:

1 Nachricht gesendet,
1 Nachricht empfangen.

Aber: nicht terminiert!

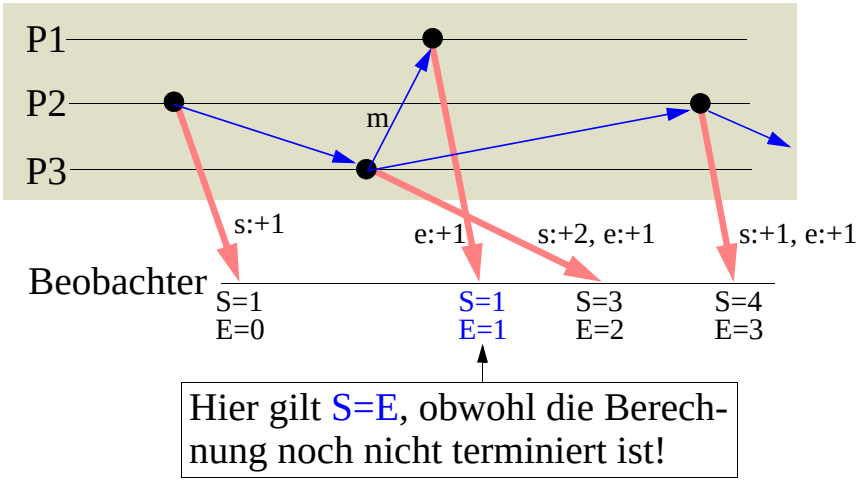
Ursache (informell):

- Nachricht aus der "Zukunft"
  - kompensiert die Zähler
- Inkonsistenter Schnitt
  - ist nicht äquivalent zu einem senkrechten Schnitt

Lösung durch "Ursachenvermeidung"? Ideen vielleicht:

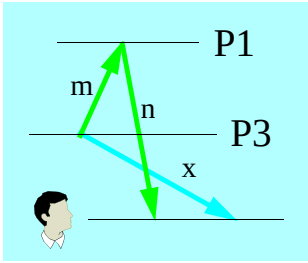
- Nachrichten aus der Zukunft *vermeiden* oder zumindest *erkennen*?
- Senkrechten Schnitt simulieren durch *Einfrieren* der Prozesse?

Beobachter über gesendete und empfangene Nachrichten informieren?



- Gleiches Szenario wie eben: Beobachter erfährt, dass m *empfangen* wurde, aber nicht, dass m *gesendet* wurde!

- Man beachte auch, dass hier eine Nachricht (x) in indirekter Weise (via m und n) "überholt" wurde!



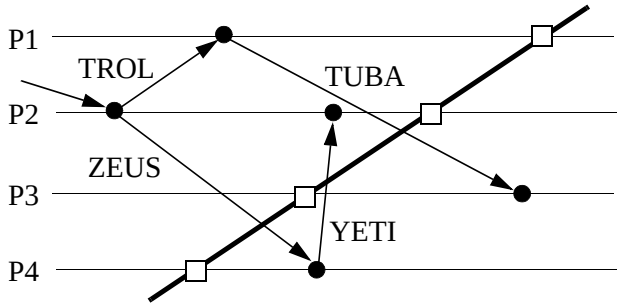
Vermutung: Wenn Informationsnachrichten nicht (indirekt) überholt werden können, dann kann das Phänomen eines "schiefen Bildes" nicht auftreten!

- worauf gründet sich die Vermutung?
- kann man solchermassen korrekte ("kausaltreue") Beobachtungen erzwingen?

Nachrichten eindeutig benennen?

Prinzip: Jede Nachricht bekommt einen (global) eindeutigen Namen:

- TROL, ZEUS, TUBA, YETI,... (?)
- Nachricht kennt ihren Namen
- Sender weiss, welche Nachrichten gesendet wurden
- Empfänger weiss, welche Nachrichten empfangen wurden



ZEUS und TUBA müssen Schnitt von links nach rechts überquert haben, YETI von rechts nach links (relativ zum Schnitt!)

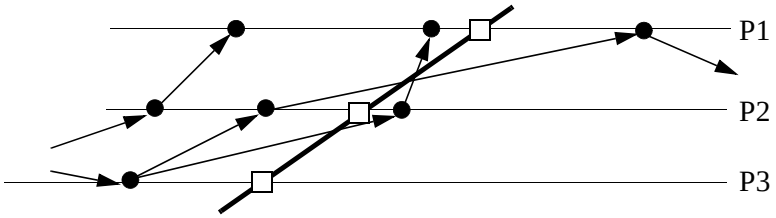
- Welle akkumuliert Namen der gesendeten und Namen der empfangenen Nachrichten
- Wenn eine gesendete nicht empfangen wurde, muss sie den Schnitt überquert haben $\Rightarrow$ Terminierung nicht melden
- Terminiert, wenn alle "bekanntermassen gesendeten" auch empfangen wurden? (Beweis?)
  - Tip: Wenn keine Nachricht den Schnitt (von links nach rechts??) überquert, ist der Lebensfaden des Systems gerissen; rechts des Schnittes kann dann keine Aktivität mehr entfacht werden (wieso?)

Eindeutige Nachrichtennamen?

- Sender könnte Nachrichten fortlaufend numerieren und seinen eigenen eindeutigen Namen hinzufügen
  - lässt sich einfacher verwalten als beliebige (global eindeutige) Namen
- Es genügt offenbar auch eine fortlaufende Numerierung pro Sender-Empfänger-Beziehung ("Kanal")
  - z.B. 17.4.239 ("239. Nachricht von Knoten 17 an Knoten 4")
  - Verwaltungsaufwand ist allerdings recht hoch
  - Denkübung: Wie lässt sich der Verwaltungsaufwand reduzieren, wenn die Kanäle die FIFO-Eigenschaften haben? (Tipp: Nachrichten zählen)

Genügt pauschales Zählen pro Kanal?

(anstatt Nachrichten pro Kanal individuell zu betrachten)



- Welle stellt folgendes fest:
  - auf Kanal P2P1 sind 2 Nachrichten gesendet und 2 Nachrichten empfangen worden
  - dennoch überquerte eine Nachricht den Schnitt von P2 nach P1 (d.h. die Terminierung ist nicht eingetreten)

- Denkübung: Ist auch bei FIFO-Kanälen eine solche Täuschung möglich oder wäre bei ausgeglichenen send/receive-Kanalzählern dann tatsächlich keine Nachricht auf diesem Kanal unterwegs?

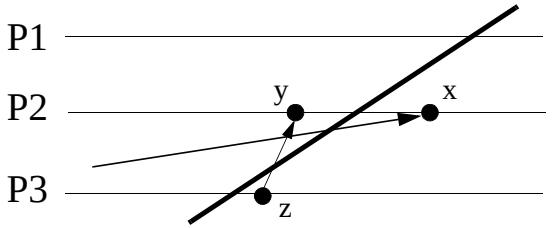
- Denkübung: Und wenn entlang eines Schnittes *alle* Kanalzähler bzgl. send/receive ausgeglichen sind?
  - Kanäle sind *unidirektional*: d.h., wenn sich zwei Prozesse gegenseitig Nachrichten zuschicken, gibt es für beide Richtungen getrennte Kanäle

Zählen pro Kanal

Behauptung: Wenn entlang eines Schnittes *pro Kanal* gleich viele Nachrichten gesendet wie empfangen wurden, dann ist die Berechnung terminiert

Betrachte frühestes Ereignis (x) *nach* dem Schnitt:

Bei globaler (von links nach rechts fliessender) *Zeit* in der Abb. ist dies klar; wenn man ohne solche graphischen Veranschaulichungen auskommen will, muss man statt dessen die *Kausalrelation* bemühen!

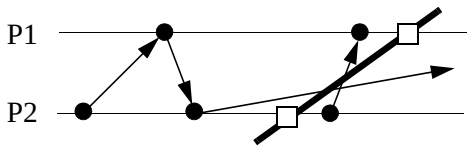


Wir zeigen durch *Widerspruch*: es gibt kein frühestes Ereignis nach dem Schnitt ⇒ Terminierung

- Dies ist ein Ereignis mit Empfang einer Nachricht, deren Sendeereignis *links* des Schnittes liegt
- Zugehöriger Kanalzähler kann nicht getäuscht werden, da für eine *Kompensationsnachricht* gilt: Empfangen (y) vor dem Schnitt, gesendet (z) danach

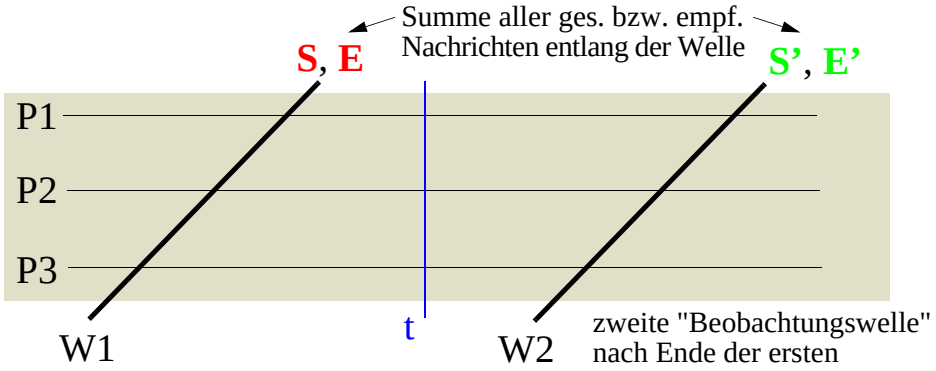
"aus der Zukunft"

- Sendeereignis der Kompensationsnachricht wäre *früheres* Ereignis (als x) *nach* dem Schnitt ⇒ *Widerspruch*
  - Senden ist immer früher als das Empfangen einer Nachricht!
  - z früher als y, y früher als x ⇒ z früher als x



Denkübung: Wieso ist das eigentlich kein Gegenbeispiel?

Das Doppelzählverfahren



Behauptung: S=E=S'=E' ⇒ terminiert

d.h. keine Nachricht unterwegs

Beweis (Skizze; lässt sich auch formalisieren):

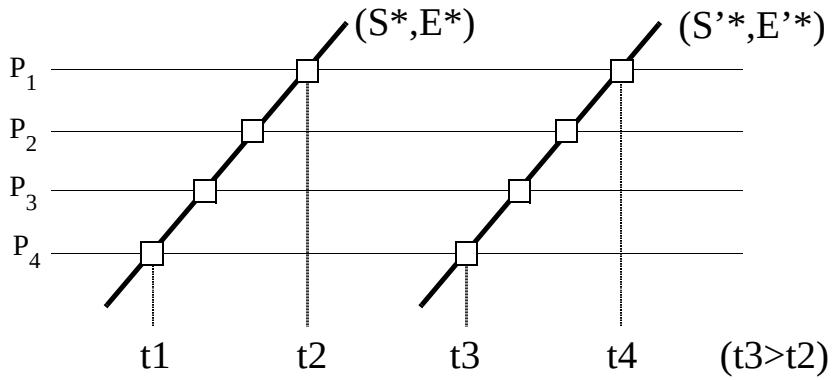
S=S' ⇒ Keine Nachricht zwischen W1, W2 gesendet.
E=E' ⇒ ———— "" ———— empfangen.
⇒ Werte bei t = Werte von W1.

Also: S=E ⇒ zum globalen Zeitpunkt t gilt:

Anzahl gesendeter = Anzahl empfangener Nachrichten
⇒ zum Zeitpunkt t ist keine Nachricht unterwegs
⇒ zum Zeitpunkt t terminiert
⇒ Berechnung war nach W1 terminiert □

Es gelingt also, für einen bestimmten Zeitpunkt eine kausaltreue Beobachtung (als senkrechten Schnitt) im Nachhinein zu *rekonstruieren*!

Formalerer Beweis des Verfahrens



Notation:

- Lokaler Send-Zähler des Prozesses P_i zur Zeit t: $s_i(t)$
- Lokaler Empf.-Zähler des Prozesses P_i zur Zeit t: $e_i(t)$
- $S(t) := \sum s_i(t)$ $E(t) := \sum e_i(t)$

Lemmata:

(1) $t \le t' \Rightarrow s_i(t) \le s_i(t'), e_i(t) \le e_i(t')$ [Def.]
(2) $t \le t' \Rightarrow S(t) \le S(t'), E(t) \le E(t')$ [Def., (1)]
(3) $E^* \le E(t2)$ [(1), e_i wird "eingesammelt" vor t2]
(4) $S'^* \ge S(t3)$ [(1), s_i wird "eingesammelt" nach t3]
(5) Für alle t: $E(t) \le S(t)$ [Induktion über die atomaren Aktionen]

Beweis:

$E^* = S'^* \Rightarrow E(t2) \ge S(t3)$ [(3), (4)]
$\Rightarrow E(t2) \ge S(t2)$ [(2)]
$\Rightarrow E(t2) = S(t2)$ [(5)]
$\Rightarrow$ terminiert zum Zeitpunkt t2 □

Zwei Zähler genügen!

Anzahl der "in-transit" Nachrichten bei t2 = 0

Eigenschaften des Doppelzählverfahrens

- Vergleich des Empfangszählers der ersten Welle mit mit dem Sendezähler der zweiten Welle (d.h. E = S') ist ein hinreichendes Kriterium

- Falls Terminierung nicht entdeckt wird: Benutze zweite Welle der vorherigen Runde als erste Welle der neuen Runde

---

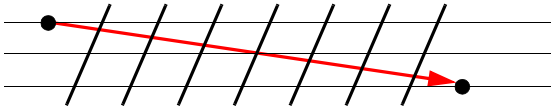
"ablaufinvariant"

- Algorithmus ist *reentrant*: Lokaler Zustand des Prozesses wird durch Kontrollalgorithmus nicht geändert
  ⇒ jeder Prozess kann unabhängig eine eigene / neue Kontrollrunde starten ("symmetrischer Algorithmus")

- Viele Varianten (unterschiedliche zugrundeliegende Wellenalgorithmen)

- Anzahl der Kontrollrunden ist a priori nicht durch die Anzahl der Basisnachrichten begrenzt

  - es kann eine ganz langsame Basisnachricht geben, während der beliebig viele Kontrollrunden gestartet werden können



  - allerdings ist folgende Variante denkbar: ein Prozess, der eine Basisnachricht erhält ohne eine neue auszusenden, startet eine Doppelrunde (Zeitkomplexität?)

Safety- und Liveness-Eigenschaften

Safety: Something bad will never happen...
(Typisch: "für alle eingenommenen Zustände gilt...")

- Bsp.: Nie mehr als 1 Prozess im kritischen Abschnitt
- Bsp.: Variable "Gehaltskonto" wird nie negativ
- Bsp.: Invariante wird nicht verletzt

oft auch "progress" — *schliesslich*

Liveness: Something good will eventually happen...
(Typisch: "es wird ein Zustand eingenommen, so dass...")

- Bsp.: Variable "Verkehrsampel" bekommt schliesslich den Wert "grün"
- Bsp.: Programm terminiert
- Bsp.: erfolgte Terminierung wird schliesslich auch gemeldet

Korrektheit: Algorithmus erfüllt Safety *und* Liveness

Beispiel verteilte Terminierung:

- aufgesetzter Kontrollalgorithmus
  - sagt "ja", wenn Basisberechnung terminiert ist
  - sagt "weiss nicht" sonst ("nein" geht nicht!)

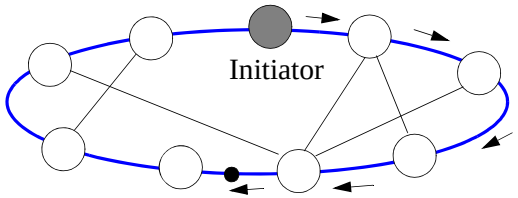
- safe, aber nicht live: meldet stets "weiss nicht"
- live, aber nicht safe: meldet stets "ja"

⇒ "Kunst": Algorithmus, der safe *und* live zugleich ist!
⇒ Es ist stets safety *und* liveness zu zeigen!

Kontrolltopologien

- Die für das Doppelzählverfahren benötigten "Wellen" können unterschiedlich realisiert werden, z.B.:

1.) *Ring / Hamilton'scher Zyklus:*



Kontrollnachricht ("Token") besucht zyklisch zwei Mal alle Prozesse und akkumuliert dabei die Zählerstände

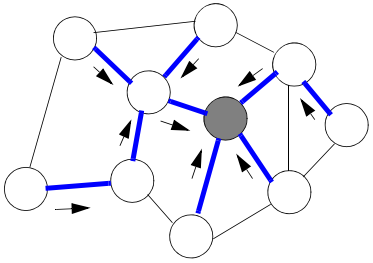
"Logischer" Ring genügt!

- in einem zusammenhängenden ungerichteten Graphen kann ein logischer Ring immer gefunden werden, indem man einen Spannbaum "umfährt"

- Zeit- und Nachrichtenkomplexität: O(n)

2.) *Spannbaum:*

- vereinfachter Echo-Algorithmus: an den Blättern reflektierte Welle sammelt die Zählerstände in akkumulierender Weise ein



- auch hier werden *zwei* solche zum Initiator hinfliessende Wellen benötigt

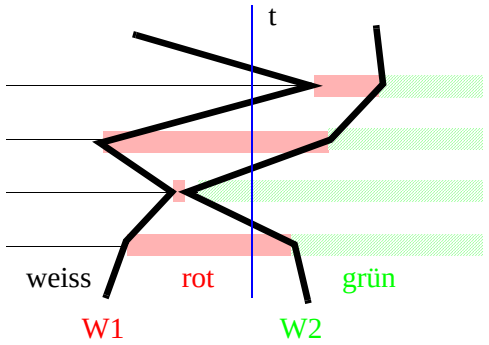
- bei nicht-entartetem Spannbaum: Viele Nachrichten parallel unterwegs ⇒ bessere Zeitkomplexität!

3.) *Echo-Algorithmus:* Explorer-Welle und Echo-Welle

Echo-Algorithmus für die beiden Wellen des Doppelzählverfahrens?

- Anwendbar für beliebige zusammenhängende Topologien

- Ausnutzen der beiden "Halbwellen" eines einzigen Laufs des Echo-Algorithmus für die beiden Kontrollrunden!



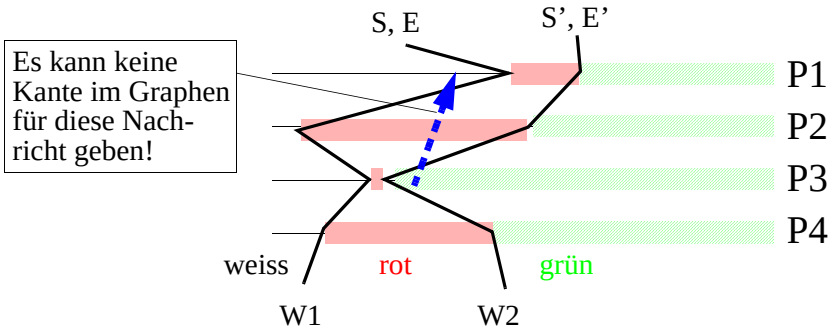
Der Echo-Algorithmus wird als *Transportdienst* zur Realisierung von *zwei* Wellen benutzt, wobei durch die Echo-Nachrichten sowohl die bei W1 lokal gemerkte Information als auch die bzgl. W2 relevante lokale Information an den Initiator übermittelt wird.

- Welle W1: "rot werden"; Welle W2: "grün werden"

- *Behauptung*: Das so realisierte Doppelzählverfahren ist korrekt

- *Problem*: Früherer Beweis lässt sich nicht direkt anwenden, da sich W1 und W2 *überlappen*!

Bemerkung:
Auf vorhandenen Spannbäumen kann man aber *nicht* einfach die vom Initiator ausgesendete Hinwelle und die reflektierte Rückwelle verwenden!



- "Trick": Es gibt keine nach W2 gesendete Nachricht, die vor W1 ankommt (grüne Knoten haben keine weissen Nachbarn!)

  - Wenn ein Knoten grün wird, sind seine Nachbarn bereits vorher rot geworden

  → Täuschung der Zähler durch Kompensation mittels Nachrichten "rückwärts" über 2 Wellen ist unmöglich!

- *Beweisskizze*:

  1) Im roten Gebiet (d.h. zwischen W1 und W2) findet keine Aktivität statt, wenn das Terminierungskriterium S=E=S'=E' gilt:
  Dazu müsste eine vor W1 (im weissen Gebiet) ausgesendete Nachricht im roten Gebiet ankommen. Dann ist aber E' > E.

  2) Es kann Nachrichten geben, die beide Wellen "vorwärts" überqueren (d.h. im weissen Gebiet gesendet werden und im grünen ankommen). Solche Nachrichten werden auf S (und S') als gesendet registriert, jedoch weder auf E noch auf E' als empfangen registriert. Da eine Kompensation der Zähler S, E mittels Nachrichten "rückwärts" über 2 Wellen unmöglich ist (wie im Gegenbsp. zum einfachen Zählen), ist dann S > E.

  Also: Es gibt keine Nachricht, die W1 überquert; nach W1 findet daher keine Aktivität statt; das System ist nach W1 terminiert

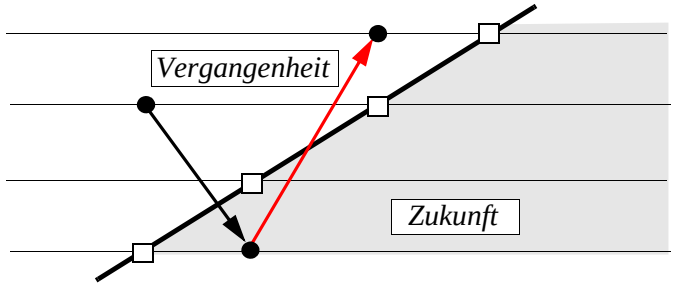
Zeitzonenverfahren

- Erkenntnis: Beim *einfachen Zählen* war eine *irreführende Kompensation* der Zähler durch Nachrichten aus der Zukunft möglich

- Idee: Entlang des Schnittes (induziert durch Welle!) zählen, aber Nachricht aus der Zukunft *erkennen*

- Bei Nachricht aus der Zukunft → Schnitt inkonsistent

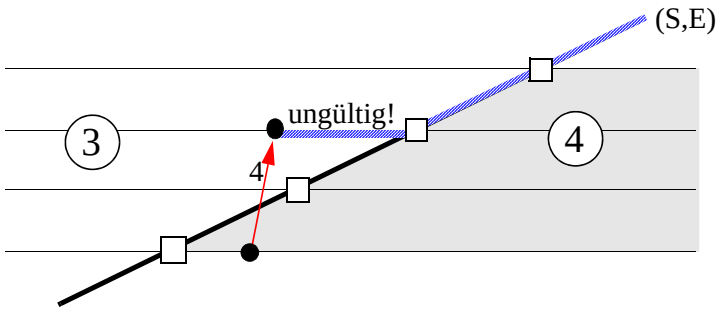
- Keine solche Nachricht → Schnitt konsistent
  ("äquivalent" zu senkrechtem Schnitt entsprechend Gummibandtransformation)



- Zählergebnis verwerfen, wenn inkonsistent

- Sequenz von Wellen, bis Terminierung festgestellt
  (Liveness ist klar: Schnitt *nach* Terminierung ist konsistent)

Prinzip: *Erkennen* inkonsistenter Schnitte



Lokale Nachrichtenzähler mit einer Welle akkumulieren, aber Ergebnis *invalidieren*, wenn eine Nachricht aus der Zukunft empfangen wurde

---

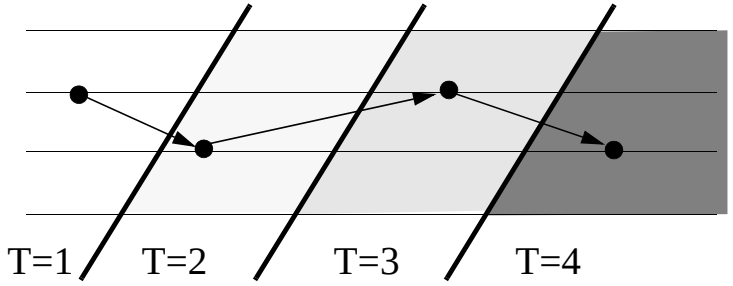
Implementierung:

- "Piggybacking" der Zeitzonennummer auf Nachrichten
- Flag setzen, wenn eine Nachricht aus einer höheren Zeitzone (z.B. "4" im Bild) empfangen wurde
- Welle registriert evtl. gesetztes Flag (und setzt es zurück) und erhöht die Zeitzone

Terminiert, wenn Welle kein gesetztes Flag feststellt und die beiden akkumulierten Nachrichtenzähler (Anzahl gesendete, Anzahl empfangene) übereinstimmen

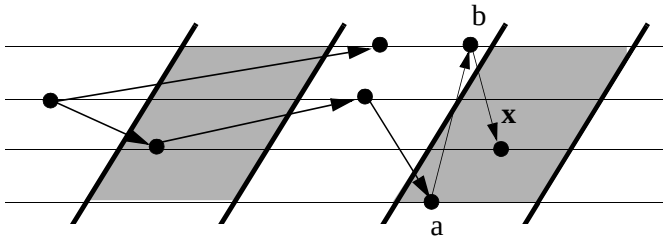
(Formaler Korrektheitsbeweis?)

- Wiederholte Ausführung des zugrundeliegenden Wellenalgorithmus:



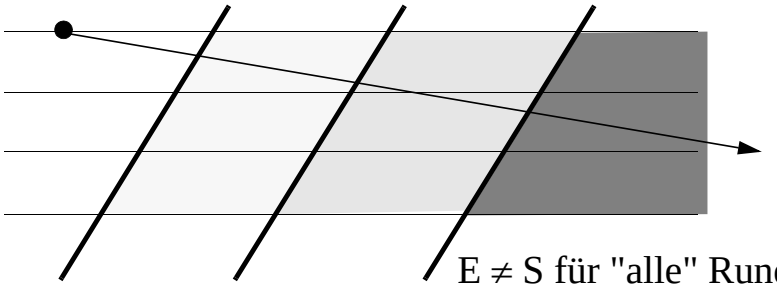
- Zeitzone bei jeder Runde inkrementieren

- "Zyklische" schwarz/weiss-Zeit genügt
  - höhere Zeitzone → "andere" Zeitzone



- *Jede* Nachricht aus der *Zukunft* wird erkannt
  - Nachricht aus der Zukunft trägt *andere* Farbe (Nachricht rückwärts über zwei oder mehr Wellen existiert nicht)
- Nachrichten aus der *Vergangenheit* können zu zu "Fehlalarmen" führen (vgl. Nachricht x)
  - Fehlalarme gefährden nicht die safety, evtl. aber die liveness
  - zyklische Zeit erfordert u.U. eine (einzige) zusätzliche Runde

- Anzahl notwendiger Kontrollrunden? Unbeschränkt!



E ≠ S für "alle" Runden...

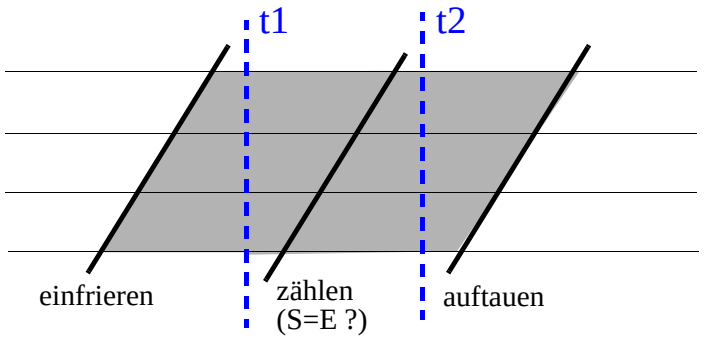
- Vergleich von Doppelzähl- und Zeitzonenverfahren?

  - Aufwand an Kontrollnachrichten und Speicher unwesentlich verschieden
  - Eingriff in die Basisnachrichten bei Zeitzonenverfahren nötig!
  - Zeitzonenverfahren ist nicht "reentrant"
    - lokaler Zustand wird verändert
    - dadurch nicht symmetrisch: Wellen mehrerer Initiatoren ("multi-source") können sich gegenseitig stören

⇒ Doppelzählverfahren scheint eleganter und einfacher / universeller einsetzbar
(allerdings u.U. eine "Runde" mehr nötig)

- Denkübung: Könnte man nicht Nachrichten aus der Zukunft von vornherein *vermeiden*?
  - z.B. durch Einfrieren des Systems (dann in Ruhe zählen und evtl. in einer weiteren Runde wieder auftauen): ist das korrekt?
  - klappt das Vermeidungsprinzip auch ohne Einfrieren?

Einfrieren?



- "Einfrieren" heisst: Kein Prozess nimmt mehr eine Nachricht an

- Folglich wird im eingefrorenen Gebiet auch keine Nachricht ausgesendet

- Nachrichten, die in diesem Gebiet eigentlich ankommen würden, werden als noch unterwegs befindlich angesehen

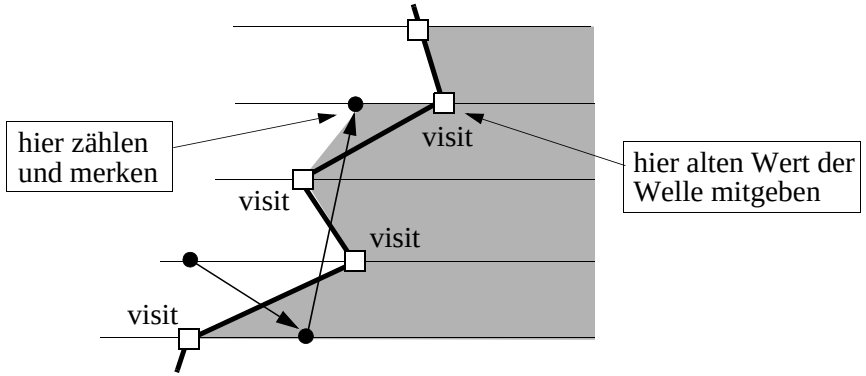
- Man beweise (formal) als Denkübung: Wenn bei der Zählwelle S=E gilt, dann ist die Berechnung terminiert
  - Tip: Man versuche, aus S=E herzuleiten, dass S'=E' bei einem "senkrechten Schnitt" (also z.B. t1 oder t2) gilt

- Informell: Wenn man alles eingefroren hat, dann kann keine Nachricht aus der Zukunft die Zählwelle überqueren

- Wie gut und praktikabel ist dieses Verfahren?

Noch ein anderes Erkennungsprinzip

- Idee: *Vermeiden* inkonsistenter Schnitte durch Vorziehen der Schnittlinie ← d.h. des Kontrollereignisses zum Zählen
  - Prinzip wird uns später beim Schnappschussproblem noch nützlich sein!



- Strategie: Im Augenblick, wo Nachricht aus der Zukunft ankommt, "kurz" vorher schon das Wesentliche des "visit" Ereignisses ausführen

  ⇒ Nachricht verläuft nun ganz in der Zukunft

  ⇒ Schnitt ist immer konsistent (→ Zählen ist korrekt)

- Bemerkungen:
  - es gibt keinen "Dominoeffekt" (mehrfaches Vorziehen ist nicht möglich; "vorgezogene" Zukunft liegt rechts vom linkesten Visit-Ereignis)
  - formal: Hüllenoperation → Rechtsabschluss bzgl. der Kausalitätsrelation

---

- Denkübung: Man vergleiche die bisher vorgestellten Terminierungserkennungsverfahren bzgl. ihrer "Qualität" (=?)
  - z.B.: diese Methode kann im Vergleich zum Zeitzonenverfahren die Terminierung u.U. eine Runde früher feststellen, dafür benötigt sie etwas mehr Speicherplatz (zum "Merken" der Werte)

Diskussion von Übung 1

Wir wollen an dieser Stelle mündlich und an der Tafel folgende Teilaufgaben aus Übung 1 besprechen, da dies für das Verständnis der folgenden Aspekte ("synchrone Kommunikation") wesentlich ist:

-) *Formalisieren* Sie für *Zeitdiagramme* den Begriff (potentiell, indirekt) "*kausal abhängig*" als *Halbordnung* über "*Ereignissen*".

-) Beobachtungen sind eine *lineare Ordnung* von (beobachteten) Ereignissen. In welcher Beziehung steht die oben erwähnte Halbordnung zu dieser linearen Ordnung?

Bemerkung: insbesondere "kausaltreue" Beobachtungen sind hier relevant

-) Was ist der Schnitt aller (kausaltreuen) Beobachtungen?

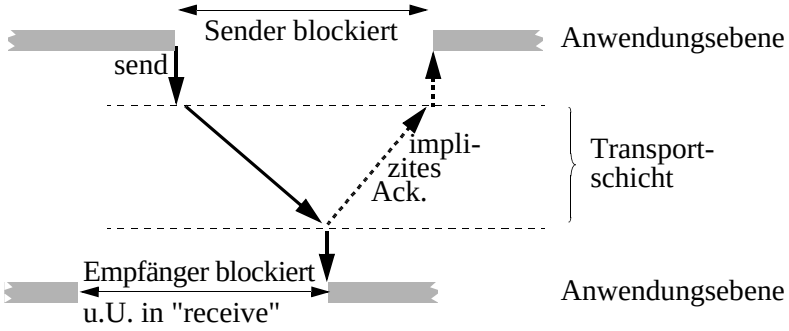
Synchrones / asynchrones Senden

- *Asynchrones Senden:* Absender wartet nach dem Absenden der Nachricht nicht (“no wait send”)

  → erfordert u.U. Puffer bei der Realisierung (in der Transportschicht)

- *Synchrones Senden:* Absender wartet blockierend, bis die Nachricht angekommen ist ("blocking send")

  → wartet auf explizite Antwort oder implizites Acknowledgement



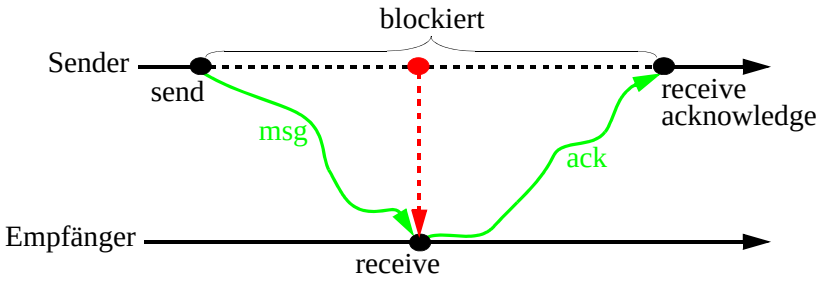
- Aus Sicht des ("bewusstlosen") Senders ist die Nachricht im Augenblick des Sendens auch schon angekommen
  - als wäre die Nachricht "unendlich schnell" (senkrechte Pfeile!)
  - Denkübung: Kann man nicht immer ein Zeitdiagramm per Gummiband-transformation "schadlos" so verzerren, dass ein Nachrichtenpfeil senkrecht verläuft?

i.w. gleiche Berechnung bei Abstraktion von der Realzeit

Achtung: Begriffe synchron / asynchron werden in der Kommunikationswelt verschieden mit leicht unterschiedlichen spezifischen Bedeutungen benutzt!

Synchrone Kommunikation

- Synchrone Kommunikation: syn chron
  “send” und “receive” geschehen virtuell gleichzeitig



- Sender ist blockiert, bis er vom Empfang seiner Nachricht erfährt
  - Sendezeitpunkt ist innerhalb des Blockadeintervalls beliebig verschiebbar

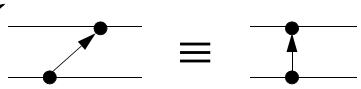
- Konsequenz:
  Man darf so tun, als wären Nachrichten nie unterwegs!
  - “blitzschnelle” Nachrichten
  - vereinfacht viele Argumentationen
  - formale Unterscheidung synchron / asynchron durch Kausalbeziehung (synchron: Abhängigkeit des *Senders* vom *Empfänger*)

Modellierung synchroner Kommunikation “Als ob” Nachrichten instantan wären?

- Wie definieren wir verteilte Abläufe ("Berechnungen") mit synchroner Kommunikation?
  - bzw: *charakterisieren* wir diejenigen verteilten Berechnungen, die mit synchroner Kommunikation *realisiert* werden können?

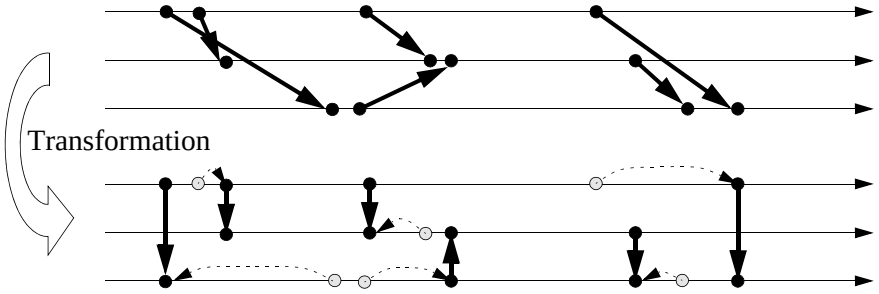
- Vorschlag:
  Synchron = virtuell gleichzeitig
  = als ob Nachrichten "instantan" wären

mit geeigneter Gummiband-transformation



- aber sind instantane Nachrichten nicht unrealistisch?

- Kann man *immer* eine geeignete Gummibandtransformation finden, sodass *alle* Nachrichtenpfeile senkrecht verlaufen?



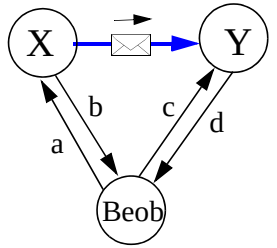
Falls man bei einer verteilten Berechnung ein Phänomen beobachten kann, welches bei instantanen Nachrichten unmöglich wäre, dann soll diese Berechnung nicht mit synchronen Nachrichten realisierbar sein

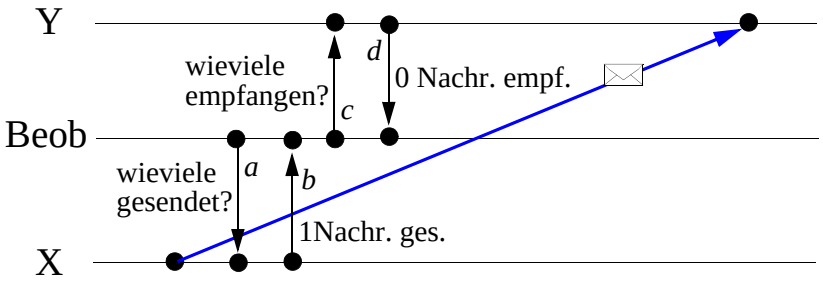
⇒ Nachrichtensemantik soll dann nicht als "synchron" bezeichnet werden

Beispiel:

Beobachter fragt zunächst X zur Zahl der Nachrichten, die dieser an Y gesendet hat

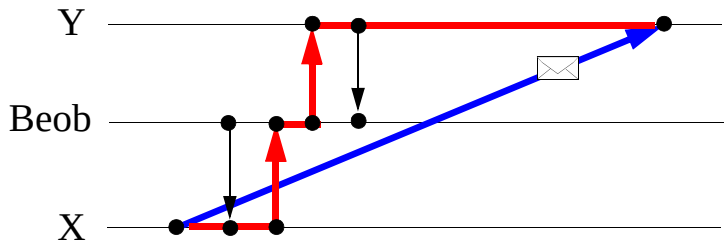
Und fragt danach Y, wieviele Nachrichten dieser von X erhalten hat





Beobachter erfährt, dass eine Nachricht von X nach Y eine *gewisse Zeit unterwegs ist* ⇒ nicht synchron!

Senkrechte Nachrichtenpfeile?

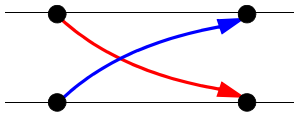


- Die Nachricht von X nach Y wird *in indirekter Weise* von einer *Kette* anderer Nachrichten *überholt*

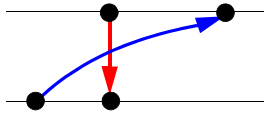
- Die direkte Nachricht kann daher durch eine Gummibandtransformation *nicht senkrecht* gemacht werden

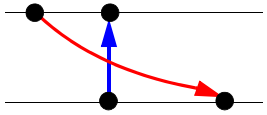
  (sonst würde eine Nachricht der Kette rückwärts in der Zeit verlaufen)

- Eine andere Berechnung, die mit synchroner Kommunikation unmöglich ist ($\Rightarrow$ Deadlock):



Obwohl jeder *einzelne* Pfeil senkrecht gestellt werden kann, kann das Diagramm nicht so gezeichnet werden, dass *beide* senkrecht verlaufen!





Charakterisierungen synchroner Kommunikation

- Sind diese alle äquivalent? (Beachte: einige Charakterisierungen sind weniger formal als andere!)

- Funktioniert ein Algorithmus, der unter der Voraussetzung syn. Komm. gemacht wurde, auch bei asynchroner Kommunikation? Und umgekehrt?

---

1) Bestmögliche Approximation *instantaner* Kommunikation

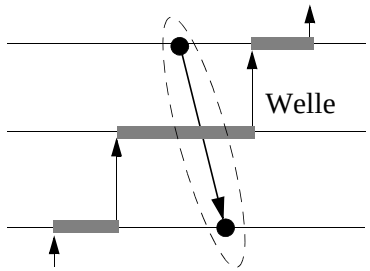
   (d.h.: ohne Realzeituhren lässt sich nicht beweisen, dass eine Nachricht nicht instantan übermittelt wurde)

2) Raum-Zeitdiagramm kann so gezeichnet werden, dass *alle Nachrichtenpfeile senkrecht verlaufen*

3) *Kommunikationskanäle* erscheinen immer *leer*

   (d.h., man sieht nie Nachrichten, die unterwegs sind)

4) Korrespondierende *Sende-/Empfangsereignisse* bilden eine *einzige atomare Aktion*



- was aber genau heisst "atomar"?

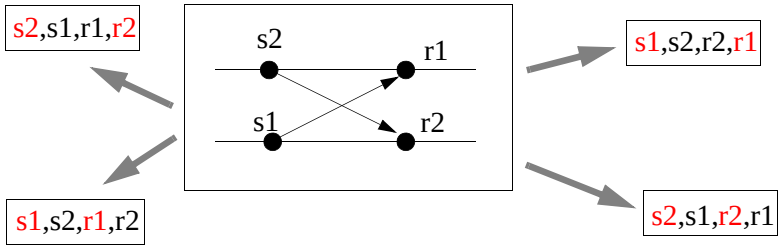
- geschieht das "kombinierte Ereignis" vor oder nach der Welle?

Menge aller Ereignisse mit Kausalitätsrelation

die zur selben Nachricht gehören

5) ∃ *lineare Erweiterung* von (E, <), so dass ∀ korrespondierenden Kommunikationsereignisse s,r gilt:
s ist direkter Vorgänger von r

- Motivation: korrespond. Ereignisse bilden eine *einzige atomare Aktion*



- Das Beispiel hat 4 verschiedene Linearisierungen: bei allen ist ein Paar korrespondierender Sende-Empfangsereignisse durch andere Ereignisse getrennt - daher kann die Berechnung *nicht synchron realisiert werden*

6) Der Graph der *Nachrichten-Scheduling-Relation* ist *zyklenfrei*

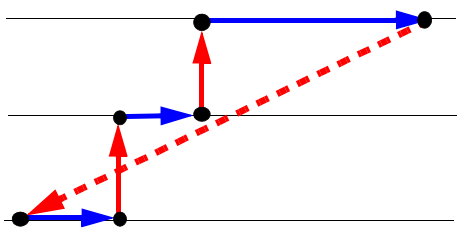
Def. der (transitiven) Nachrichten-*Scheduling-Relation* '<' :

m '<' n ⇔ send(m) < receive(n)

dies ist die klassische Kausalitätsrelation auf der Ereignismenge

- Dann können ganze *Nachrichten* (d.h. korrespondierende Sende-/Empfangsereignisse s, r) als Ganzes gescheduled werden (s unmittelbar vor r), andernfalls ist das nicht möglich

7) Wenn man *Nachrichtenpfeile in beliebiger Richtung* durchläuft, Prozesslinien aber nur von links nach rechts, dann entsteht *kein Zyklus*



- *Interpretation*: Richtung der Nachrichtenpfeile ignorieren ⇒
  - send / receive ist "symmetrisch"
  - "identifiziere" send / receive

- Falls ein solcher Zyklus existiert ⇒ es gibt keine "erste" Nachricht
- Falls kein Zyklus existiert ⇒ Nachrichtenschedule existiert

8) Synchrone Kausalitätsrelation << ist eine *Halbordnung*

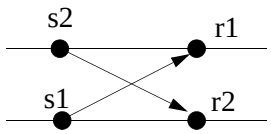
Definition von << : vgl. dies mit der klassischen Kausalrelation!

1. Falls a vor b auf dem selben Prozess, dann a << b
2. x << s ⇔ x << r ("common past") } für alle korrespond. s, r und für alle Ereignisse x
3. s << x ⇔ r << x ("common future") } für alle korrespond. s, r und für alle Ereignisse x
4. Transitiver Abschluss

Interpretation: korrespondierende s, r stehen *nicht* in Relation, aber bzgl. der synchronen Kausalitätsrelation werden sie "identifziert"

haben gemeinsame Vergangenheit und Zukunft

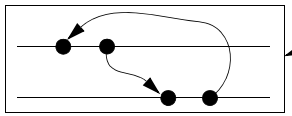
"Gegen"beispiel:



a) **s1 << r2** (1) "common future" von (s1, r1)
b) **r1 << r2** (a, 3)
c) **s2 << r1** (1)
d) **r2 << r1** (c, 3)

Zyklus, aber r1 ≠ r2 (d.h. << ist hier keine Halbordnung!)

- Vergleiche diese Charakterisierung mit Charakterisierung 6
- Wieso ist die Definition von << sinnvoll?

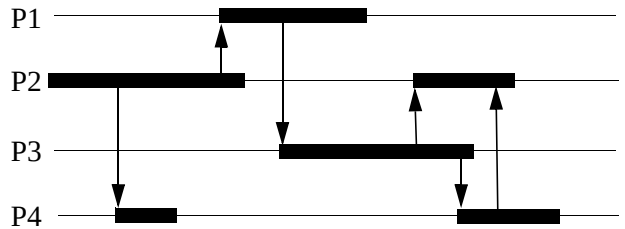


Hier ist die klassische Kausalrelation < keine Halbordnung, daher ist dies noch nicht einmal mit *asynchronen* Nachrichten realisierbar!

Zum Nachlesen: www.vs.inf.ethz.ch/publ/papers/mattern-dc-1996.pdf

Terminierungserkennung bei synchroner Basiskommunikation

- Nachrichtenpfeile senkrecht in Zeitdiagrammen



- Abstrakte Basisberechnung mit zwei Aktionen modellieren:

$state_p$ mit Werten *aktiv* oder *passiv*

X_p: {$state_p$ = *aktiv*}
    $state_q$:= *aktiv* // "atomares" Aktivieren von q

I_p: $state_p$:= *passiv*

Nachricht hier unerheblich...

- Aufgabe: Für dieses abstrakte Modell einer Basisberechnung einen überlagerten Kontrollalgorithmus angeben

- Aktion X_p grenzt an Telepathie mit "spukhaften Fernwirkungen"

Verhaltensmodelle vert. Anwendungen und Definition der vert. Terminierung

Nachrichtengesteuertes Modell ("Transaktionsmodell"):

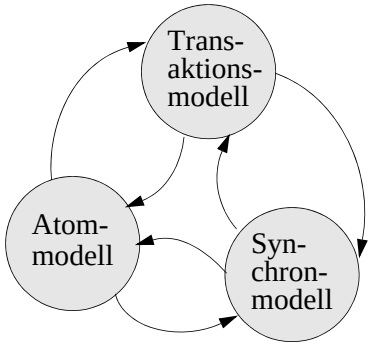
- Alle Prozesse passiv und keine Nachricht unterwegs

Atommodell:
(Prozesse sind immer passiv)

- Keine Nachricht unterwegs

Synchronmodell:
(Nachrichten haben Laufzeit 0)

- Alle Prozesse passiv



Simulation (aller Eigenschaften) eines Modells A in / mit einem Modell B?

- Lassen sich die jeweiligen Lösungsalgorithmen für ein anderes Modell verwenden / transformieren?

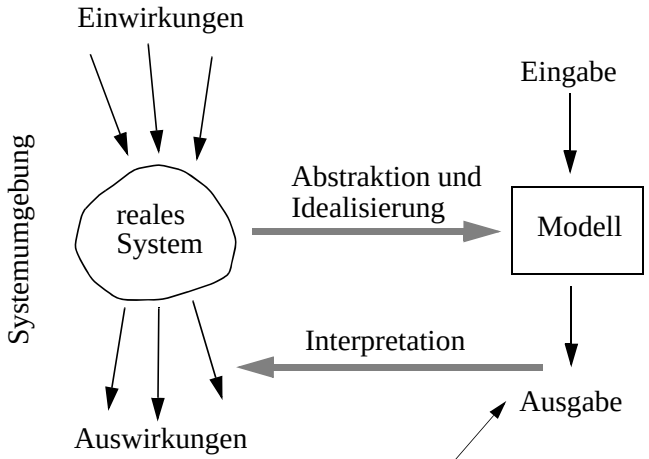
  Denkübung: Man überlege sich geeignete Transformationen an

- Was ist das "richtige" Modell?
  - Hängt von der Anwendung ab
  - Hängt evtl. von Abstraktionsniveau und Sichtweise ab
  - Ein Modell lässt sich u.U. *realisieren* (Unterschied zu Naturwissenschaften!)
    - z.B. Asynchronität "maskieren" und eine virtuell synchrone Sicht liefern
    - z.B. Aktionen eines Prozesses wechselseitig ausschliessen

Modelle und Modellierung - ein kurzer Exkurs

- Modell = *vereinfachtes* Abbild der komplexen Realität

soll sich bzgl. *relevanter* Aspekte (=?) analog zur Realität darstellen oder verhalten



"...dass die Folgen der Bilder stets wieder die Bilder seien von den naturnotwendigen Folgen der abgebildeten Gegenstände" (Heinrich Hertz)

- Abstraktion → Reduktion, Vergröberung!

- Modelle dienen u.a. dem Begreifen der Wirklichkeit
  - in diesem Sinne bereits prähistorische Höhlenmalereien
  - Spielen von Kindern
  - Voraussetzung für planvolles Handeln

- Durchspielen am Modell → *Simulation*
  ("Modellexperiment")

Heinrich Hertz (1857 - 1894)

> Wir machen uns unsere Scheinbilder oder Symbole der äusseren Gegenstände, und zwar machen wir sie von solcher Art, dass die denknotwendigen Folgen der Bilder stets wieder die Bilder seien von den naturnotwendigen Folgen der abgebildeten Gegenstände... so können wir an ihnen, wie an Modellen, in kurzer Zeit die Folgen entwickeln.
>
> -- Heinrich Hertz: Einleitung zu "Prinzipien der Mechanik", 1894

- Arbeitsgebiete: Funkenentladung, Induktion, Elektrodynamik, Thermodynamik
- geboren in Hamburg 1857
- 1877 immatrikulierte sich Hertz an der Technischen Hochschule München, kümmerte sich aber wenig um den Vorlesungsbetrieb, sondern besuchte mehr die Museen und Theater
- Wechsel nach Berlin, Physikstudium unter Helmholtz und Kirchhoff
- 1879 Goldmedaille der Humboldt Universität für die Lösung einer Preisaufgabe (Nachweis einer trägen Masse von elektrischen Selbstinduktionsströmen)
- Promotion "Über die Induktion in rotierenden Kugeln"
- Assistentenstelle bei Helmholtz
- 1883 Habilitation ("Versuche über die Glimmentladung") an der Uni Kiel
- 1885 Ruf an die TH Karlsruhe
- 1886 Heirat mit Elisabeth Doll, der Tochter eines Kollegen; zwei Töchter
- 1889 Professor für Physik in Bonn
- gestorben 1894 an einer Sepsis, noch nicht einmal 37-jährig

Beispiele für Modelle

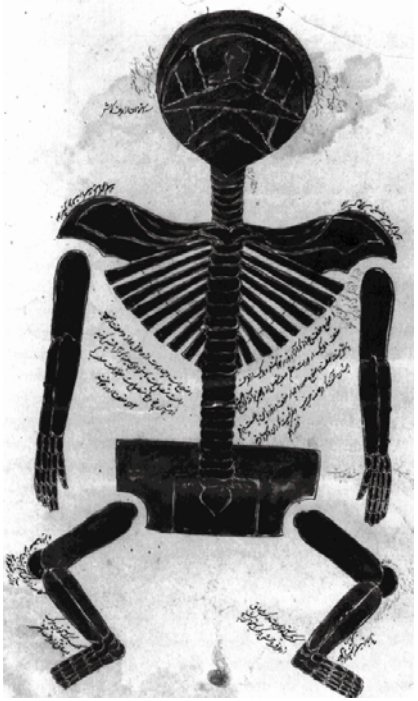
- Spielzeugeisenbahn ("Modell"eisenbahn)
- Planetarium, Globus, Strassenkarte
- Sandkasten
- Architekturmodell ← bereits in der Antike: Holzmodelle von geplanten Bauwerken
- Flugsimulator
- Training von Astronauten
  - Zentrifuge (Beschleunigung)
  - Parabelflüge (Schwerelosigkeit)
  - isländische Kraterwüste (Mondlandschaft)
- Windkanalmodelle (→ "digitaler Windkanal")
- Differentialgleichungssystem
- Schach und andere Spiele
- Puppe ("Dummy")
- Ratte ("Tiermodell")
- ...

Korrektes Modellieren ist eine "Kunst":

> Welche Aspekte werden beibehalten?
> Wovon wird abstrahiert?

- Konkrete Modelle
  - z.B. massstabsgerechte Verkleinerung
  - z.B. Simulation hydraulischer Strömungssysteme durch elektrische Schaltkreise
- Abstrakte Modelle (→ "Computersimulation")

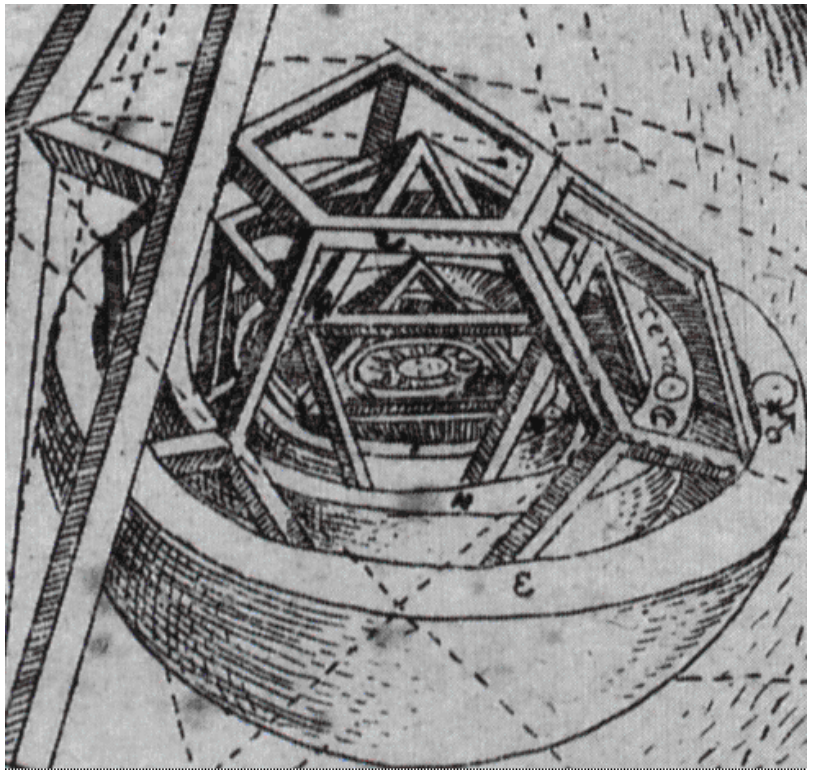
Ein wirklichkeitsfremdes Modell des menschlichen Skeletts (Persien, 14. Jhd.)



- Was hat eigentlich ein "model" mit "Modell" zu tun?

Ein Modell der Planetenbahnen aus verschachtelten platonischen Körpern

(J. Kepler, 1597)



Zweck der Modellierung

- Zweck von Modellierung bei der *Simulation der* (evtl. prognostizierten) *Wirklichkeit*

  durch Experimente am Modell

  - Optimierung (z.B. Bestimmung von Engpässen)
  - Entscheidungshilfe (Auswahl von Entwurfsalternativen)
  - Prognose (z.B. Wetter)
  - Validierung (z.B. geplanter Mikroprozessor)
  - Theorienbildung (z.B. kognitive Vorgänge)
  - Animation, Erklärung (pädagog. Hilfsmittel, "Demonstrationsmodell")

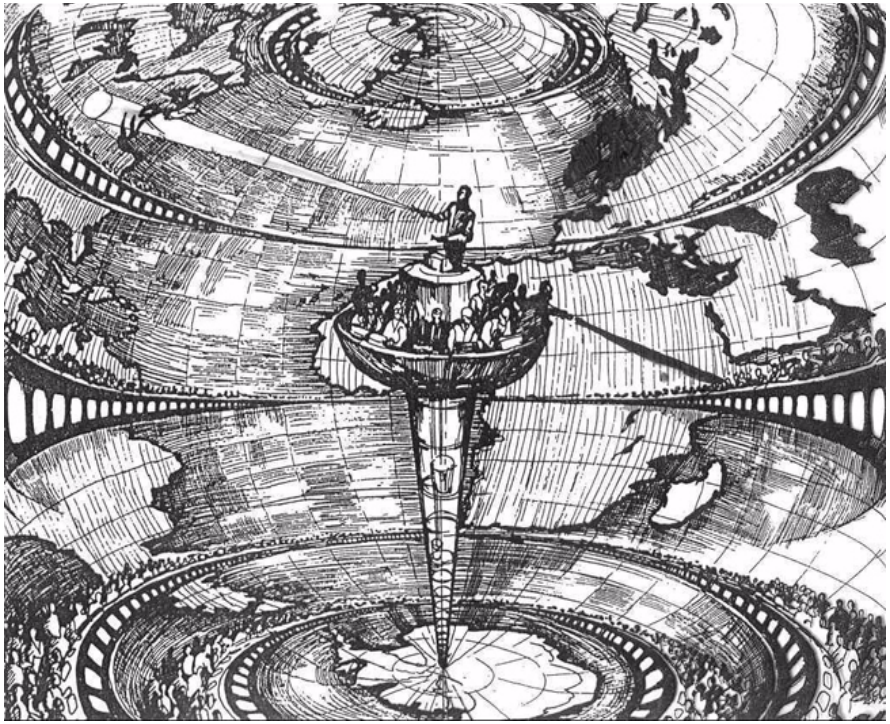
- Es geht um das Erkennen und Beherrschen der Realität

- Geschichte der Modellierung und Simulation
  - Höhlenmalereien, Landkarten, Globus
  - Formalisierung der Physik (Differentialgleichungen)
  - Sandkastenspiele der Militärs (Schach)
  - Computer ab ca. 1950 (zunächst militärische und ökonomische Anw.)

The history of mankind is a history of model building
[Rivett: Principles of Model Building]

The craft of modelling is central to the way in which we understand the world arround us
[Kreutzer: System Simulation]

Historische Sicht der Wettervorhersage mittels numerischer Simulation



Mehrere 10000 menschliche Rechner in einer kugelförmigen Galerie um einen Dirigenten berechnen die Wettervorhersage. Eine auf die Wände gemalte Erdkarte definiert das Rechengitter, das eine räumliche Auflösung von einem Grad hat. Die Ergebnisse der Berechnung werden in einem Eimer zu einem Telegraphen herabgelassen, der die Wetterprognose verbreitet. (Nach L.F. Richardson, *Weather Prediction by Numerical Processes*, 1922)

Implementierung von Modellen

- Im Unterschied zu den *Naturwissenschaften* geht es in der *Informatik* oft nicht um die möglichst detailgetreue Nachbildung der Wirklichkeit, sondern um die *Implementierung* eines Modells als "ausgedachte Wirklichkeit"

- Weil das ausgedachte Modell "schön" ist, z.B.:
  - als hätte man einen sehr grossen Hauptspeicher (→ "virtueller Speicher")
  - als ob ich einen Rechner exklusiv für mich alleine hätte (→ "virt. Maschine")
  - als würden nie Bits auf einer Leitung verloren gehen
  - als ob Nachrichten keine Zeit bräuchten
  - als ob ich eine tolle Figur wäre (Chat, Avatar, Computerspiel, 2nd life...)
  - als ob...

- Das heisst: Man erschafft sich eine "Wirklichkeit"
  - künstliche Wirklichkeit = virtuelle Realität

Welt am Draht (1973)

Film von Rainer Werner Fassbinder (nach dem Roman "Simulacron-3" von D.F. Galouye)

Plot Summary (Wikipedia): Am "Institut für Kybernetik und Zukunftsforschung" wurde ein Supercomputer namens Simulacron-1 entwickelt, der imstande ist, eine ganze Welt künstlich zu simulieren. Diese virtuelle Realität wird von "Simulationseinheiten" bevölkert, die in etwa dasselbe Leben führen wie wir und ein Bewusstsein besitzen. Ausser einer "Kontaktperson" wissen die simulierten Menschen jedoch nicht, dass ihre Welt nur eine Simulation bzw. ein Simulakrum ist.

Fred Stiller wird zum neuen Direktor des Instituts befördert, nachdem sein Vorgänger unter ungeklärten Umständen ums Leben gekommen ist. Zuvor hatte dieser gegenüber seinem Mitarbeiter Günther Lause noch angedeutet, eine "ungeheure Entdeckung" gemacht zu haben. Einige Tage später verschwindet Lause wie vom Erdboden verschluckt, unmittelbar, bevor er Stiller über die Entdeckung in Kenntnis setzen konnte. Stiller macht sich auf die Suche nach Lause, doch niemand ausser ihm scheint den einstigen Sicherheitschef des Instituts überhaupt gekannt zu haben. Während Stiller weitere Nachforschungen anstellt, ereignen sich innerhalb der Computersimulation seltsame Dinge...

Schliesslich entdeckt er, dass auch seine eigene Welt nicht die wirkliche Welt ist, sondern ebenfalls eine Simulation, die von einer höheren Ebene aus programmiert wurde.

- Mit einem kurzen Textauszug beenden wir unseren Exkurs in die Modellierung und Simulation

"Dank Hannon J. Fuller haben wir eine bessere Methode gefunden. Wir können eine ganze Gesellschaft, ein Milieu, elektronisch simulieren. Wir können sie mit individuellen Entsprechungen bevölkern - mit sogenannten reagierenden Identitätseinheiten. Durch Manipulierung des Milieus, durch Reizung der ID-Einheiten vermögen wir das Verhalten in hypothetischen Situationen zu beurteilen."

...

"Der Simulator ist das elektromathematische Modell eines durchschnittlichen Gemeinwesens. Es erlaubt Verhaltensvoraussagen auf weite Sicht. Diese Vorhersagen sind noch um ein Vielfaches präziser als die Ergebnisse einer ganzen Armee von Meinungsforschern, die unsere Stadt durchkämmen."

...

Am Fenster starrte ich auf eine vertraute Strassenszene hinunter... Aber warum sollte hier etwas verschieden sein! Meine Analogstadt musste ein Abbild dieser Stadt sein, wenn sie ihren Zweck erfüllen sollte. Ich sah genauer hin und erkannte einen Unterschied. Viele Personen rauchten gemächlich ihre Zigaretten... Und es war klar, dass eine der simulelektronischen Funktionen meiner unechten Welt darin bestanden hatte, die Erfolgschancen einer Tabak-Prohibition zu prüfen.

...

Ich liess mich in Halls Sessel Fallen und begiff erst jetzt, dass ich wirklich aus der Illusion in die Realität emporgestiegen war.

...

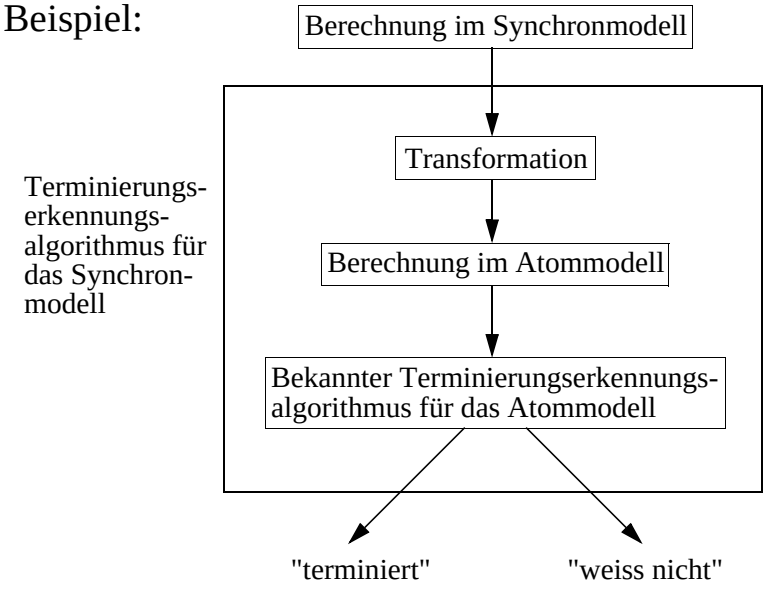
"Es wird dir hier gefallen, Doug, obwohl es vielleicht nicht ganz so drollig ist wie auf deiner Welt. Hall hatte einen Sinn für das Romantische, als er den Simulator programmierte. Die Attrappennamen wie *Mittelmeer*, *Riviera*, *Pazifik*, *Himalaja* und so weiter verraten doch immerhin viel Phantasie... Du wirst sehen, dass unser Mond nur ein Viertel so gross ist wie der eure. Aber Du wirst dich schon an die Unterschiede gewöhnen."

Ich legte den Arm um ihre Hüften und zog sie an mich. Ich war auch überzeugt davon.

Exkurs-Ende

Modelltransformation

- Viele "Probleme" lassen sich definieren / lösen im Transaktionsmodell, Atommodell und im Synchronmodell
  - Frage: Wie wird ein Lösungsalgorithmus (z.B. zur Erkennung der verteilten Terminierung) für ein anderes Modell adaptiert?
- Idee: Transformation des *Modells*, nicht der Algorithmus
- Beispiel:



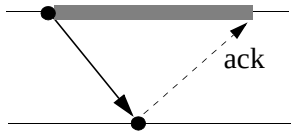
- Transformation muss natürlich *korrekt* sein: Wenn die transformierte Berechnung beendet ist, dann ist auch die Originalberechnung beendet ($\rightarrow$ safety); wenn die Originalberechnung beendet ist, dann ist schliesslich auch die transformierte Berechnung beendet ($\rightarrow$ liveness)

Beispiel für eine Modelltransformation

Gegeben:
- Berechnung im Atommodell
- Terminierungsalgorithmus für Synchronmodell

statt Atommodell

Was tun? $\rightarrow$ Einführung von "passiv" und "aktiv" (Modelltransformation)



- Sende ack zurück bei Empfang einer Nachricht
- Sender wird "aktiv" beim Senden, "passiv", wenn alle Nachrichten quittiert

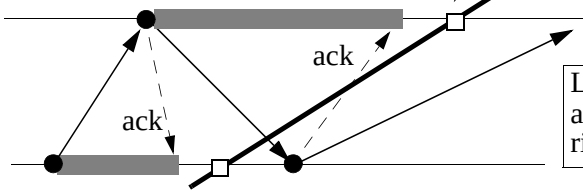
Dann gilt: Alle "passiv" $\Rightarrow$ keine Nachricht unterwegs

stellt Terminierungsalgo. für Synchronmodell fest

ist Terminierungsdefinition für das Atommodell

Aber was ist damit?:

Zu jeder (bekannten) Nachricht wurde ein ack empfangen und alle sind passiv. Terminiert?

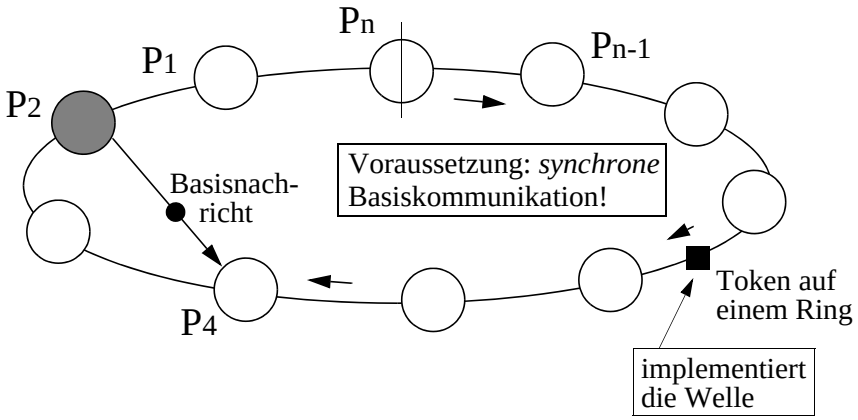


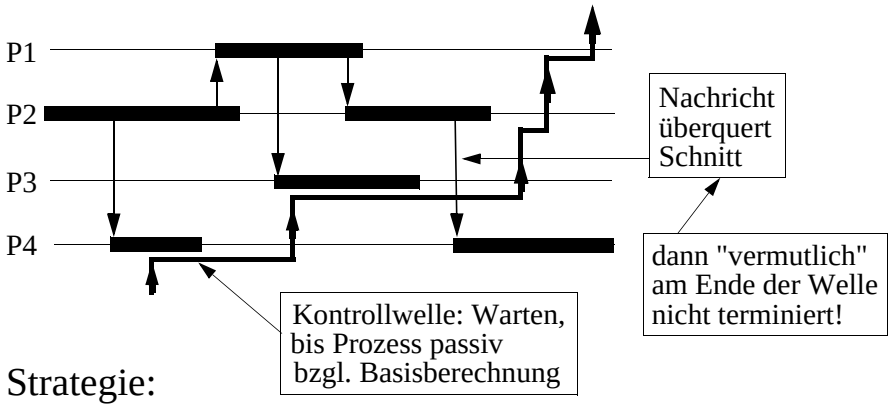
Lässt sich ein entspr. Szenario auch mit senkrechten Nachrichtenpfeilen realisieren?

- "Alle sind passiv" (entlang einer schiefen Schnittlinie!) ist kein korrektes Terminierungskriterium im synchronen Fall!
- Genausowenig wie "alle passiv und alle acks angekommen" im Transaktionsmodell!

Der DFG-Algorithmus

- Dijkstra, Feijen, Van Gasteren (1983)
- Zur Terminierungserkennung bei syn. Basiskommunikation





Strategie:

- Prozesse und Token können schwarz oder weiss sein
- Prozess wird schwarz, wenn er an einen Prozess mit einer höheren Nummer etwas sendet
- Welle testet, ob ein Prozess schwarz ist und färbt Prozess ("auf der Rückflanke") weiss
- Terminiert, wenn (entlang Welle) alle weiss

DFG-Algorithmus: Verhaltensregeln

Regel 1: Ein Prozess, der eine Basisnachricht [an einen Prozess mit einem höheren Index] sendet, wird schwarz.

Regel 2: Wenn Prozess P_n passiv ist, kann er den Terminierungstest initiieren, indem er ein weisses Token an P_{n-1} sendet.

Regel 3: Ein aktiver Prozess behält das Token, bis er passiv wird.

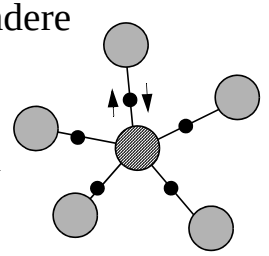
Regel 4: Ein passiver Prozess P_i ($i \neq n$), der das Token hat, reicht ein schwarzes Token weiter an P_{i-1}, wenn er oder das Token schwarz ist, ansonsten reicht er ein weisses Token weiter.

Regel 5: Ein Prozess, der das Token weiterreicht, wird weiss.

Regel 6: Wenn Prozess P_n ein weisses Token erhält, meldet er *Terminierung*.

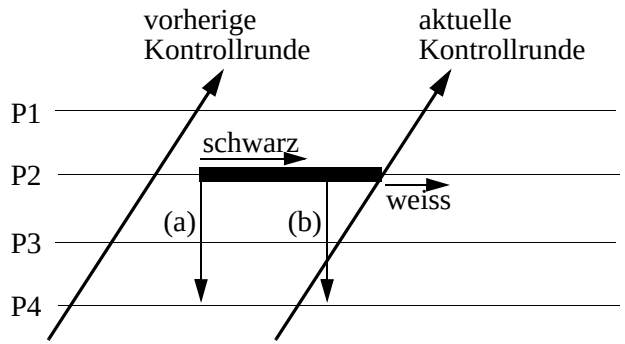
Regel 7: Wenn Prozess P_n ein schwarzes Token erhält, startet er eine neue Runde.

- Korrektheit? (Safety; Liveness)
- Muss das Token mittels synchroner Kommunikation propagiert werden?
- Wieso klappt der Algorithmus nicht für asynchrone Basiskommunikation?
- Worst-case Nachrichtenkomplexität? "Detection delay"?
- Muss der Initiator eindeutig sein? Mehrere Initiatoren?
- Statt Kontrollring andere Realisierung der Kontrollwelle?
- Welche Farbe haben die Prozesse initial?



z.B. Stern mit einem zentralen Initiator, der *parallel* Token zu den Prozessen schickt, um deren Farbe zu ermitteln

DFG-Algorithmus - "falscher Alarm"



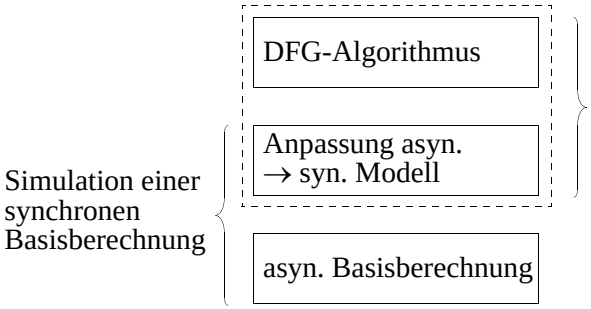
- Die beiden Situationen (a) und (b) werden nicht unterschieden, obwohl nur (b) gefährlich ist

- Konsequenz: Wenn im Gebiet zwischen den beiden Runden eine Nachricht an einen höheren Prozess gesendet wird, ist in jedem Fall noch eine weitere Runde nötig

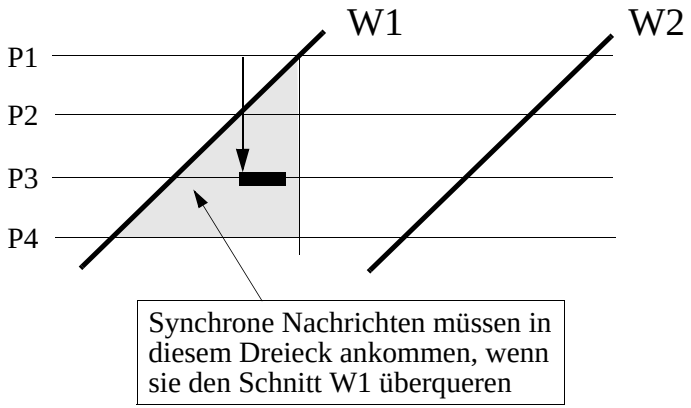
- Vereinfachung von Regel 1 (Konsequenz?)

Regel 1': Ein Prozess, der eine Basisnachricht sendet, wird schwarz

---



Empfangsflags bei syn. Kommunikation



Idee: Feststellen, *ob* im Dreieck eine Nachricht ankommt

- W1 *schärft* ein Empfangsflag (d.h. setzt es zurück)
- Empfang einer Basisnachricht *setzt* das flag
- W2 (gestartet nach Ende von W1) *prüft*, ob ein flag gesetzt wurde

Wenn

einfach realisierbar!

- W1 keine aktiven Prozesse "durchtrennt" hat
- W2 kein gesetztes flag feststellt

dann terminiert (nach Ende von W1 spätestens)

Rolle von W1 und W2 kann zusammengefasst werden

- kombinierte Welle testet erst und setzt dann das flag zurück

---

Denkübungen:

- Exakter Beweis? (Ohne mit "senkrechten" Pfeilen die Geometrie zu bemühen)
- Prinzip auf asynchrone Kommunikation übertragen? (Modelltransformation)

Übungen (3)

1) Man zeige, dass der Algorithmus von Arora et al.
("Distributed termination detection algorithm for distributed computations", Information Processing Letters, Vol 22 No 6, pp. 311-314, 1986) fehlerhaft ist. Dazu
   a) gebe man ein möglichst einfaches Gegenbeispiel an,
   b) lokalisiere man den Fehler im Korrektheitsbeweis,
   c) identifiziere man den eigentlichen Denkfehler der Autoren.

2) Wenn man davon ausgehen kann, dass sich Nachrichten nicht überholen (FIFO-Kanäle), lässt sich das "flushing-" oder "channel-sweeping-Prinzip" anwenden: Um sicherzustellen, dass auf einem Kanal keine Basisnachricht mehr unterwegs ist (oder: unterwegs war?), schiebt man mit einer Kontrollnachricht eventuelle Basisnachrichten aus dem Kanal heraus. Man setze diese Idee in einen Algorithmus zur Feststellung der verteilten Terminierung um, und zwar:
a) für Modelle mit aktiven und passiven Prozessen, wo Nachrichten beliebig lange unterwegs sind, die Nachrichtenkanäle jedoch die FIFO-Eigenschaft besitzen;
b) für das allgemeine Modell (wo die Kanäle nicht unbedingt FIFO sind), indem man die gefundene Lösung entsprechend adaptiert (ohne die FIFO-Eigenschaft für die Basiskommunikation zu erzwingen!)

3) In der Vorlesung wurden drei Berechnungsmodelle (nachrichtengesteuertes Modell, Atommodell, Synchronmodell) vorgestellt, für die man das Terminierungsproblem lösen kann. Man zeige für jedes der Modelle, wie man eine Lösung in diesem Modell als Lösung für eines der anderen Modelle verwenden kann bzw. in eine entsprechende Lösung in systematischer Weise transformieren kann.

Information Processing Letters 22 (1986) 311–314
North-Holland

ACHTUNG: Algorithmus ist falsch!

DISTRIBUTED TERMINATION DETECTION ALGORITHM FOR DISTRIBUTED COMPUTATIONS

R.K. ARORA, S.P. RANA and M.N. GUPTA

1. Introduction

The distributed termination problem requires taking of explicit or implicit snapshots of the state of distributed processes and then testing a termination criterion over the snapshot. The majority of the algorithms in the literature [3,4,5,6,7,11], relegates this responsibility to a unique process in the system. Because of its specialized role, the unique process becomes a point of centralized control and hence a performance bottleneck. In this paper we are interested in fully distributed algorithms, where all processes have identical code added to them for the purpose of solving the termination problem.

Few algorithms [2,10] do belong to the above category. However, we further depart from these and we avoid the use of time-stamps and clock-synchronization.

After introducing the termination problem in Section 2, a new algorithm is developed with the correctness arguments in Section 3. The last section concludes with the comments on the performance aspects of the presented algorithm.

2. Distributed termination problem

Consider a distributed program P consisting of n communicating sequential processes $p_1, p_2, \ldots, p_n$. Let a local predicate c_i be associated with process p_i, $1 \leqslant i \leqslant n$. Let C be the conjunction of local predicates c_i where the values of the c_i's correspond to the same time instant for all the processes. We refer to C as the distributed termination condition. The necessary condition for a process p_i $(1 \leqslant i \leqslant n)$ to terminate is the truth of c_i. However, it can only be terminated after asserting the truth of C.

When c_i is true, $1 \leqslant i \leqslant n$, the corresponding process p_i is said to be in *passive* state, otherwise it is in *active* state. An active process may change the state of a passive process by engaging into basic communication with that process. The basic communication refers to the communication inherent in the distributed program P. In addition to basic communication, control communication, which takes place around the Hamiltonian ring in anti-clockwise direction, is superimposed for the purpose of detecting distributed termination.

A passive process never initiates a basic communication; however, a process in any state can initiate or engage in control communication.

An active process p_i, $1 \leqslant i \leqslant n$, may engage in basic communication only with the processes belonging to the set N_i, referred to as the set of neighbours of p_i.

3. Detection of distributed termination

Algorithms for termination of distributed programs primarily consist of two distinct phases viz. a detection phase followed by a termination phase. In the detection phase, the truth of the distributed termination condition is asserted, whereas the termination phase actually terminates the program.

First, we focus our attention on the detection phase. In the next section, comments on the termination phase will be provided.

3.1. Detection phase

The detection phase employs control—messages are referred to as *probe-messages*. The processes are assumed to have unique identifications. A probe-message always carries the identification of its initiator process.

In the algorithm below, we allow probe-messages to propagate along a unidirectional ring, linking the n processes. The underlying strategy in the presented algorithm is briefly stated as follows: "Upon satisfaction of a local condition, a process sends a probe-message with its identification to a successor process. Whenever a process receives a probe-message, it again makes a check and depending upon this check it either forwards the probe-message or purges it. However, if the

probe-message received by a process is its own message, the last process concludes the truth of the distributed termination condition and thus enters into termination phase."

The algorithm is elaborated by answering the following questions:

- What local information is to be satisfied so as to initiate a probe message?
- How is a response to an incoming probe message generated by a process?

In the present algorithm, each process is required to maintain the following information:

(1) its own state information (active or passive), and

(2) the information about the state of its neighbours.

This information is maintained by modifying the code in the following manner: Whenever a process sends a message to a neighbouring node, it records the state of the neighbouring node as active. Whenever a node becomes passive, it sends an I-am-passive message to all its neighbours. As soon as a process receives an I-am-passive message from a process, it records the state of the last process as passive, or remains passive, if it was already passive.

The algorithm is fully described as follows.

Algorithm for process p_i ($1 \leqslant i \leqslant n$)

1. *Upon becoming passive*:
state(p_i) := passive;
/ * state(p_i) has been used for recording the status of p_i * /
for each $p \in$ {neighbours of p_i} **do**
send I-am-passive to p;

2. *Upon receiving an I-am-passive message from* p ($p \in$ {neighbours of p_i}):
begin
state$_i$(p) := passive;
/ * state$_i$(p) is used for recording the status of process p in p_i * /
if state$_i$(p_j) = passive for all $p_j \in$ {neighbours of p_i}
and state(p_i) = passive
then send a probe-message to successor(p_i)
/ * successor(p_i) is used for referring successor of p_i on the ring * /
end

3. *Upon receiving a probe-message initiated by a process other than* p_i:
begin
if state$_i$(p_j) = passive for al $p_j \in$ {neighbour of p_i}
and state(p_i) = passive
then forward the probe-message to successor(p_i)
else purge the above probe-message

4. *Upon receiving a probe-message initiated by* p_i *itself*:
enter in termination phase;

3.2. Termination phase

To finally terminate the process of the distributed program P, let each process make use of Boolean variables SENDTM and RCVTM with initial value ***false***. For a process, SENDTM is set to ***true*** upon its forwarding the termination-message to the successor while RCVTM is set to ***true*** upon its receiving a termination-message from the predecessor.

Based upon the above, the steps to be taken in terminating the processes are briefly given as follows:

1. *Upon determining the distributed termination condition by a process* p_i:
```
begin
  SENDTM := true;
  send termination-message to successor(p_i)
  end
```
2. *Upon receipt of termination-message by* p_i:
```
begin
  RCVTM := true;
  if SENDTM then terminate
else begin
  SENDTM := true;
  send termination-message to successor(p_i);
  terminate
  end
end
```


Correctness of the algorithm

To establish the correctness of the above algorithm so as to ensure that the false termination condition is not detected and any deadlock situation does not arise, we need to state and prove the following assertions:

(1) There is at least one process that succeeds in initiating the probe-message.

(2) There is at least one process that gets its probe-message back, i.e., it succeeds in entering the termination phase by detecting the truth of the distributed termination condition.

(3) No process can enter the termination phase without the distributed termination condition being ***true***.

Proof of assertion (1)

Consider a process, say p, that is last to become passive. As p belongs to the set the numbers of which are neighbours of p, all the neighbours of p will qualify to initiate probe-messages.

Further, consider any neighbour p_j of p. Since p is the last process to become passive, p_j and all neighbours of p_j are already passive. Thus, p_j has either received all I-am-passive messages except from p when p becomes passive or I-am-passive messages are in transit. Eventually, p_j will receive all I-am-passive messages. Upon receipt of the last I-am-passive message from one of its neighbours, p_j will thus qualify for initiation of a probe-message. Hence, the assertion follows.

Proof of assertion (2)

Once the distributed termination condition has become ***true***, all processes are in passive state and only I-am-passive messages may be in transit. Such messages will eventually be delivered causing one or more processes to initiate probe-messages. Let p be the process among the above processes to initiate the probe-message last. Obviously, when p initiates the probe-message, all processes would be passive, and p has received all I-am-passive messages. Thus, the probe-message of p would be forwarded by each process and hence would reach back to p. Thus, there is at least one process in the system getting its probe-message back.

Proof of assertion (3)

The proof follows by contradiction. Consider a probe-message initiated by a process, say p. Let p get its probe-message back and hence conclude the truth of the distributed termination condition. Further, assume that there is an active process, say p_j, thus violating the above conclusion.

Obviously, p_j has become active after forwarding the probe-message of p. In other words, some active process p_i has communicated with p_j after p_j's forwarding the probe-message of p. There are two cases. Either p_i has not yet received the probe-message of p or has already forwarded it. If p_i has already forwarded it, this means that p_i in turn has been activated by some other process. Continuing the argument in this manner, we can conclude that there is some process, say p_k, that before receiving the probe-message of p communicates with a process that has already forwarded the probe-message of p. This means that process p_k must purge the probe-message upon receiving it. But, according to our assumption, all processes forward the probe-message of p, hence a contradiction.

4. Concluding remarks

We have presented a fully distributed and symmetric algorithm for solving the distributed termination problem that does not make use of time-stamp and clock-synchronization. We have looked into three aspects:

- initiation of probe-messages to detect the distributed termination condition,
- detection phase,
- termination phase.

The issuance of probe-message from a process has been based upon the concept of neighbouring processes, i.e., that set of processes with which the process enters into basic communication for solving the distributed termination problem. The probe-message is issued from a process only when itself and all its neighbours have become passive. This results in considerably less control message traffic as compared to other reported approaches [2,10].

The detection phase is based upon two factors:

- maintenance of local information so as to initiate or respond to probe-messages,
- propagation of probe-message.

The local information in terms of keeping the status of neighbours of a process in its control section is automatically generated when basic communication takes place between the process and its neighbours.

The probe-message once issued from a process propagates around the ring in anti-clockwise direction and either reaches back its originating process thereby determining the truth of the distributed termination condition or gets purged on its way on encountering either an active process or a passive process with at least one of its neighbours in active state. Finally, upon determining the truth of the distributed termination condition, the termination phase has been discussed where a process gets terminated only after it has both received and forwarded a termination-message issued by a process or processes.

Thus, we find that our presented algorithm is simple and involves less message overheads.

References

[1] K.R. Apt and J.L. Richier, Real time clocks versus virtual clocks, Tech. Rept. #84-34, LITP, University of Paris 7, 1984.

[2] R.K. Arora and N.K. Sharma, A methodology to solve distributed termination problem, Information Systems 8 (1) (1983) 37–39.

[3] K.M. Chandy and J. Misra, Termination detection of diffusing computations in communicating sequential processes, Tech. Rept. TR#144, The University of Texas at Austin, 1980; also: ACM-TOPLAS 4 (1) (1982) 37–43.

[4] E.W. Dijkstra and C.S. Scholten, Termination detection for diffusing computations, Inform. Process. Lett. 11 (1) (1980) 1–4.

[5] N. Francez, Distributed termination, ACM-TOPLAS 2 (1) (1980) 42–45.

[6] N. Francez and M. Rodeh, Achieving distributed termination without freezing, Tech. Rept. TR#72, IBM Israel Scientific Center, 1979.

[7] N. Francez, M. Rodeh and M. Sintzoff, Distributed termination with interval assertions, Tech. Rept. TR#186, Computer Science Dept., TECHNION—Israel Institute of Technology, 1980.

[8] C.A.R. Hoare, Communicating sequential processes, Comm. ACM 21 (8) (1978) 666–777.

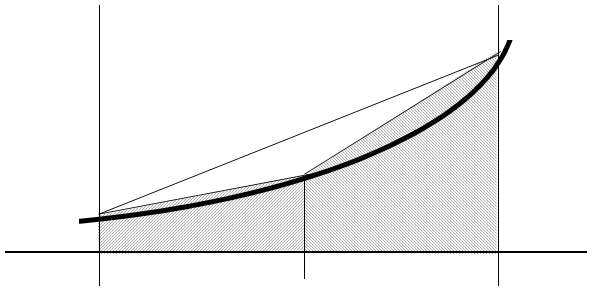
[9] L. Lamport, Time clocks and ordering of events in a distributed system, Comm. ACM 21 (7) (1978) 558–565.

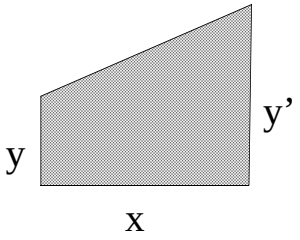
[10] S.P. Rana, A distributed solution of the distributed termination problem, Inform. Process. Lett. 17 (1983) 43–46.

[11] R.W. Toper, Termination detection for distributed computations, Inform. Process. Lett. 18 (1984) 33–36.

Parallele Berechnungsschemata und verteilte Terminierung

Einfaches Beispiel: Berechnung einer Fläche
(numerische Integration: Trapezmethode)





$$A = x \frac{y + y'}{2}$$ (wieso?)

Prinzip:
- x-Intervall iterativ / rekursiv halbieren bis zu einem *Stoppkriterium*
- Teiltrapezflächen dann aufaddieren

Beachte: Mathematische Aspekte hier ausgeklammert!

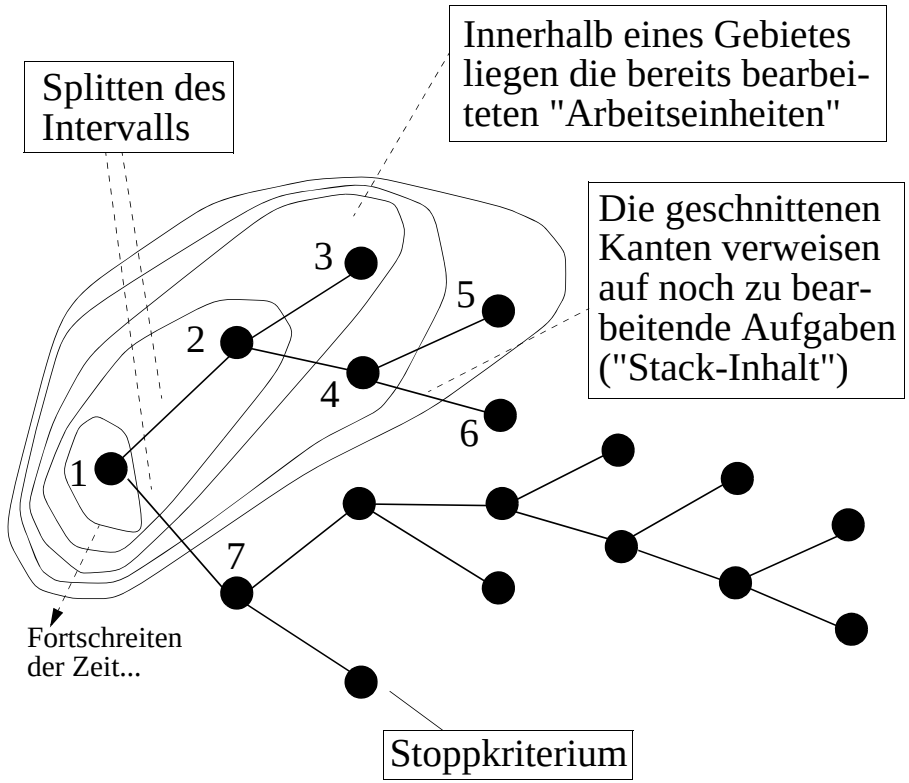
Mögliche *Stoppkriterien*:
- berechnete Trapezfläche bis auf ε gleich halber Fläche des vorher. Intervalls
- Steigung Sekante / Mitteltangente nahezu gleich

Sequentielle Berechnung

- Iterative bzw. rekursive Berechnung ist kanonisch

Systematisches Abarbeiten eines Binärbaums von Teilaufgaben; Unerledigtes auf einen Stapel (= Stack)

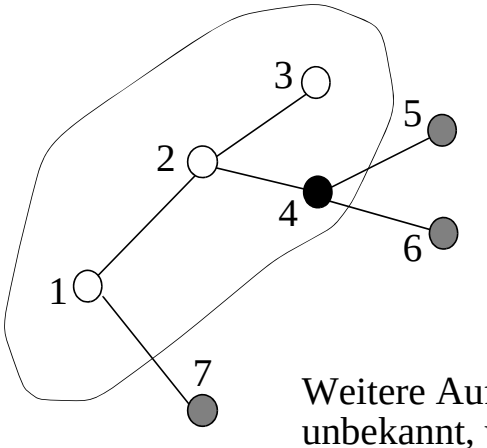
- *Visualisierung:*



- Teilflächenwerte stehen an den Blättern!

Parallelisierung

"Schnappschuss" der sequentiellen Berechnung:

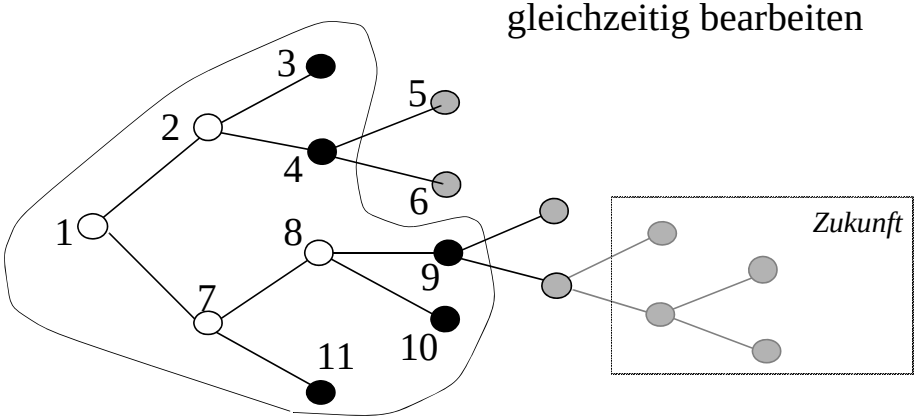


- Aufgabe 4 wird gerade bearbeitet
- Aufgaben 5, 6 und 7 sind bereits "generiert"
- Aufgaben 1, 2 und 3 sind bereits bearbeitet

Weitere Aufgaben sind noch unbekannt, werden erst später generiert

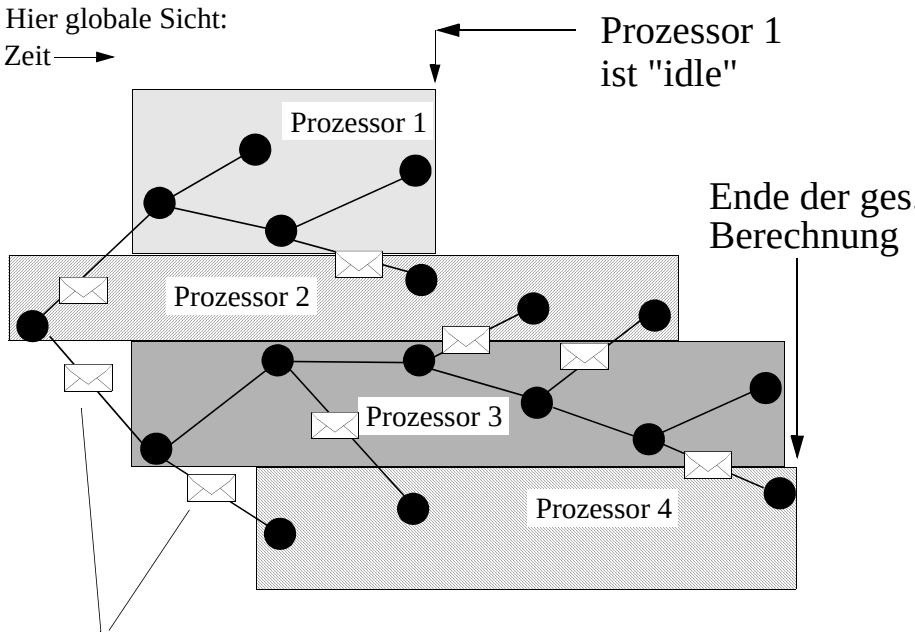
---

Parallelisierung: Aufgaben 3, 4, 9, 10, 11 (an der "Front") evtl. gleichzeitig bearbeiten

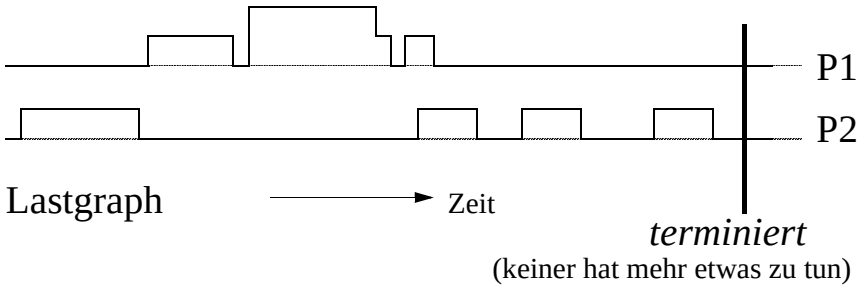


"Zeitfront" dehnt sich in mehrere Richtungen gleichzeitig aus

Lastausgleich und Terminierung

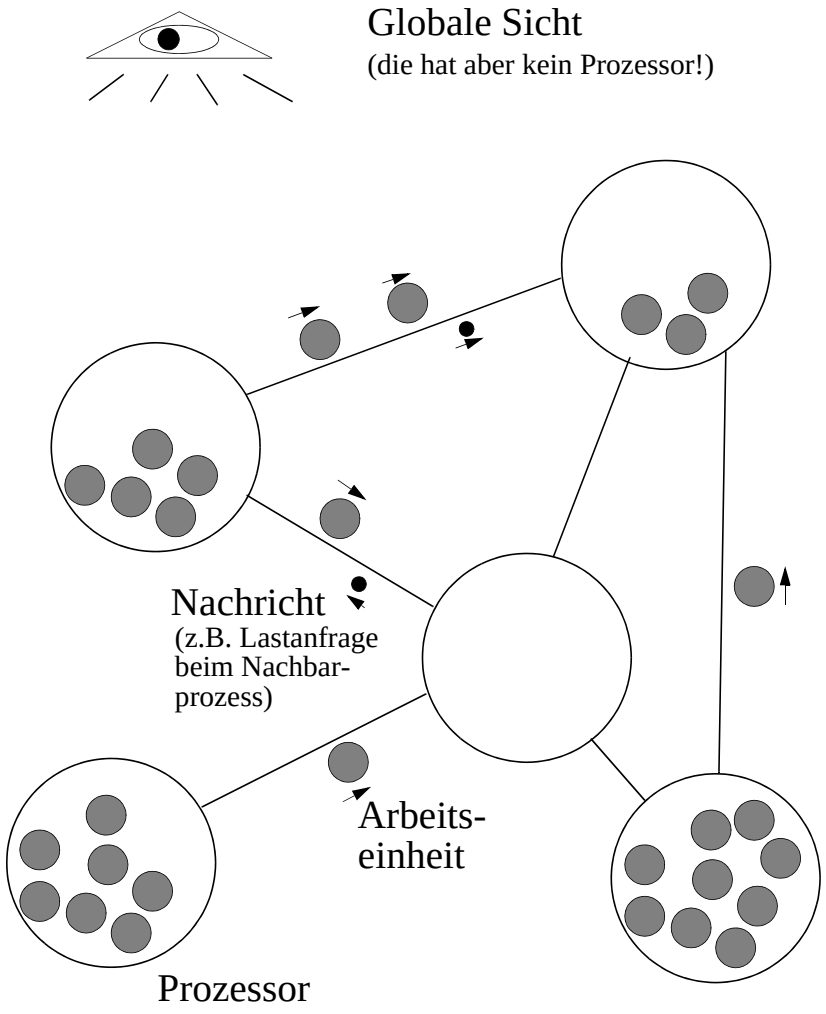


Übermittlung von (neu entstandenen) Arbeitseinheiten an andere Prozessoren zum Zweck des Lastausgleichs



Terminiert, wenn Last überall 0 und nichts "unterwegs"

Globaler Schnappschuss einer Berechnung



US Patent 6 112 225

August 29, 2000 / Filed: March 30, 1998
Assignee: **I**nternational **B**usiness **M**achines Corporation (Armonk, NY)

SUMMARY OF THE INVENTION

Broadly, the present invention concerns a system for processing a computer executable "aggregate" task **by dividing it into subtasks and distributing the subtasks** "on demand" to remotely located subscribing computers via a computer **network** such as the public Internet. Application programs running on the subscribing computers obtain subtasks from a coordinating computer "on demand", and manage execution of the obtained subtasks during their idle processing time. Ultimately, the subscribing computers **submit the results of their processing back** to the coordinating computer.

What is claimed is:

1. A method performing a computing task comprising operations of:

a coordinating computer receiving an aggregate computing task divisible into multiple independent subtasks;

announcing an opportunity for other computers to participate in the aggregate computing task, and in response, one or more subscribing computers submitting requests to participate in the aggregate computing task, the subscribing computers including one or more computers having principal functions distinct from the aggregate computing task;

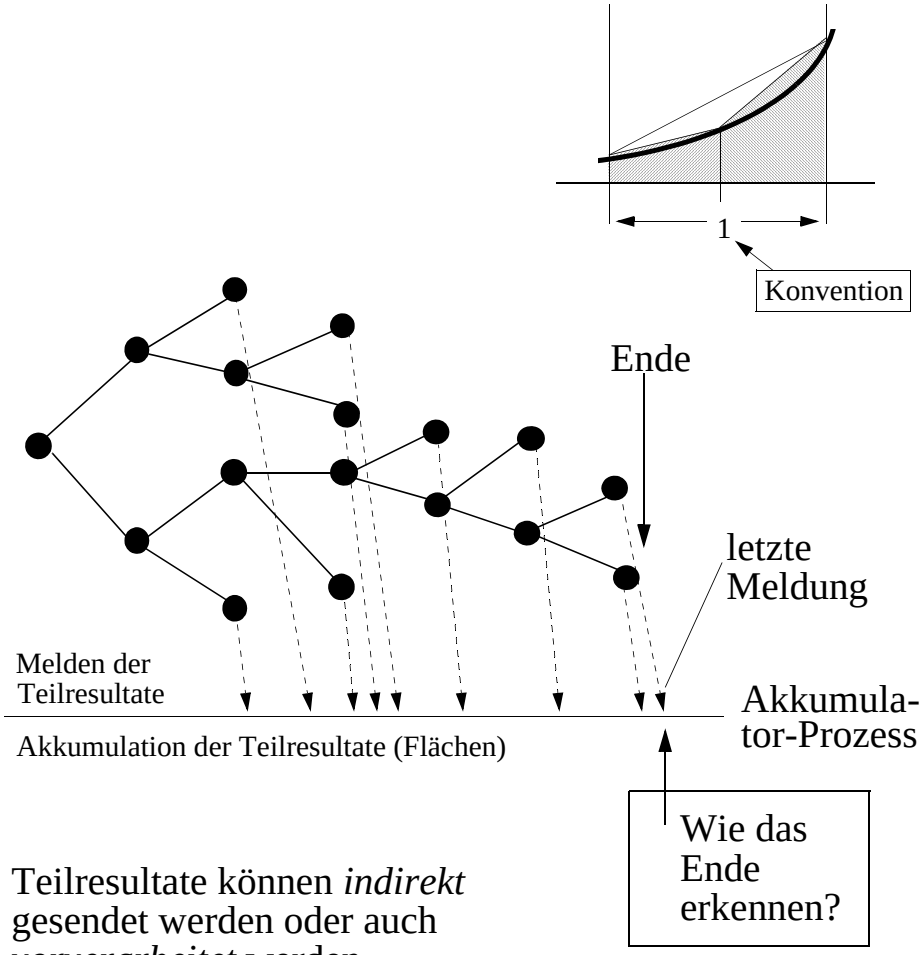
the coordinating computer receiving the requests from subscribing computers, and in response, the coordinating computer assigning the subtasks by distributing the subtasks among the subscribing computers, and also sending an idle time activation program to each subscribing computer;

each subscribing computer installing the idle time activation program, whereupon the idle time activation program causes the subscribing computer to perform operations including working toward completion of the assigned subtask when the subscribing computer is in a predefined idle state with respect to the subscribing computer's principal functions, and halting work toward completion of the assigned subtask when the subscribing computer is not in the predefined idle state; and

in response to each subscribing computer's completion of its assigned subtask, transmitting results of the completed subtask to the coordinating computer.

2. ...

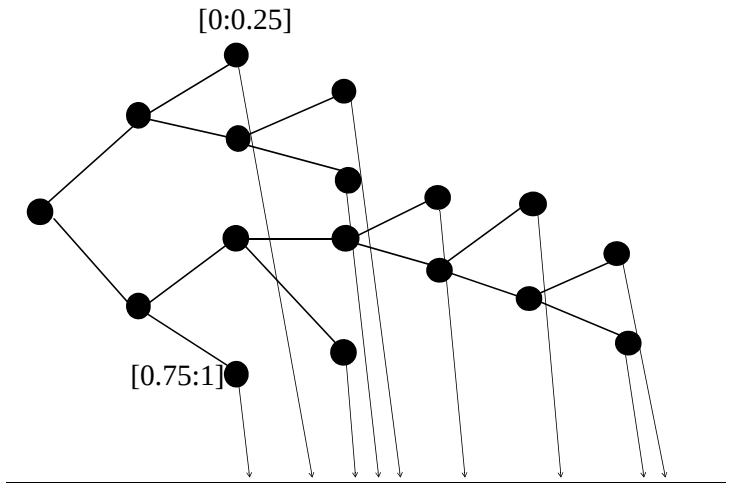
Parallele numerische Integration: Erkennung der Terminierung



Wie das Ende erkennen?

Teilresultate können *indirekt* gesendet werden oder auch *vorverarbeitet* werden...

Lösung des Terminierungsproblems

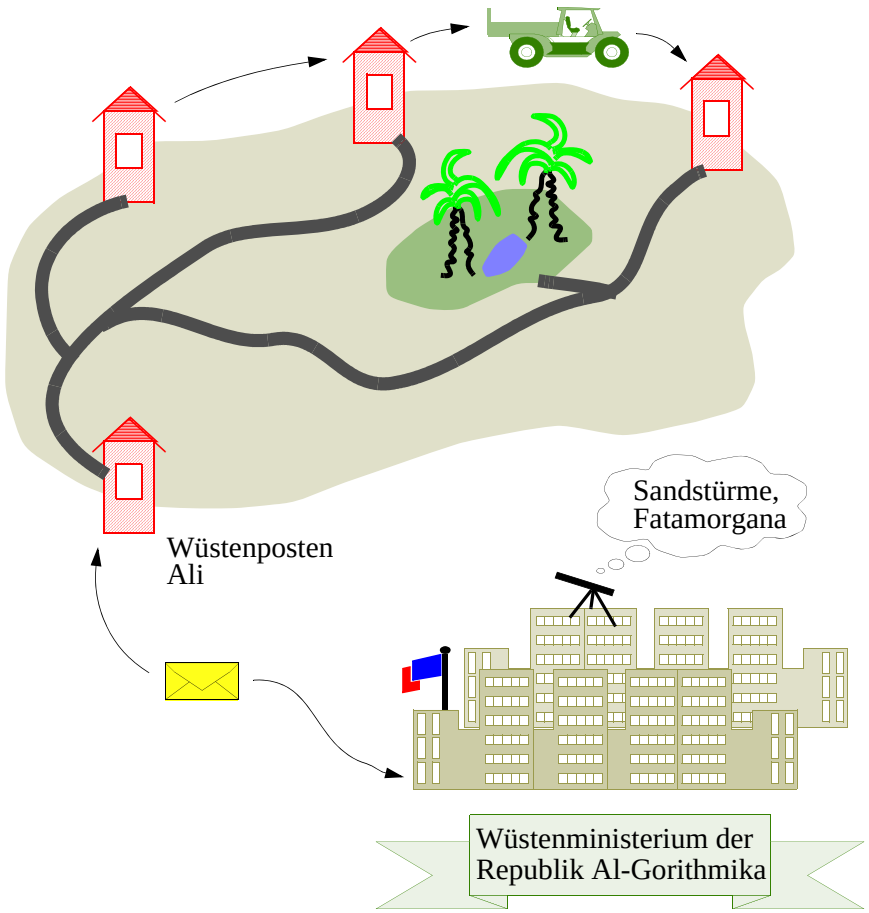


Intervallgrössen $\frac{1}{4}$ $\frac{1}{4}$ $\frac{1}{8}$ $\frac{1}{8}$ $\frac{1}{8}$ $\frac{1}{16}$ $\frac{1}{32}$ $\frac{1}{64}$ $\frac{1}{64}$ $\Sigma = 1$!

Statt $\frac{1}{2^i}$ besser nur *i* versenden (bleibt "klein")

Lässt sich die Lösung auf allgemeine verteilte Berechnungen übertragen?

Das Problem der Wüstenposten



Vorschriften und Probleme...

- Reisende dürfen die Wüste nur "kontrolliert" (bei einem "wachen" Posten) betreten

- Nur ein wacher Posten lässt Personen in die Wüste einreisen

- Nur aus der Wüste kommende Reisende können einen Posten wecken

- Kontrolleure sollen die Beobachtungen der Wüstenposten an das Ministerium melden

Problem: Wann ist die Wüste leer?

- Fatamorgana und Sandstürme trüben die globale Sicht des Ministeriums

- Zählen der Reisende geht oft "schief", wie früher bereits eingesehen

Eine orientalische Lösung

- Reisende erhalten *Eintrittskarten*
- Rückgabe bei Austritt aus der Wüste
- Ministerium gibt feste Zahl von Tickets aus
- Ministerium sammelt Tickets wieder ein

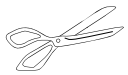
Invariante: Gesamtzahl der Tickets

- Aktiver Posten soll einen nichtleeren Vorrat an Tickets haben

- Wenn das Ministerium alle Tickets wieder eingesammelt hat, ist kein Reisender in der Wüste unterwegs und sind alle Posten passiv → *terminiert*

Was tut ein Posten, wenn er sein letztes Ticket verkaufen müsste?

- Neue anfordern → bürokratischer Aufwand!
- Orientalischer Trick: *Ticket halbieren* und nur das halbe Ticket verkaufen...



Puzzle-Abteilung im Ministerium...

Die Kreditmethode

- Idee:

  Verallgemeinerung des beim Integrationsbeispiel gefundenen Prinzips

  Also: Akkumulation eines Wertes, bis dieser =1

Bedingungen:

(1) Urprozess startet die verteilte Berechnung
(2) Prozesse und Nachrichten haben einen Kreditanteil $\in Q^+$
(3) Summe aller Kreditanteile stets = 1 ← Invariante
(4) Aktiver Prozess hat Kreditanteil > 0
(5) Nachricht hat Kreditanteil > 0

Theorem: Berechnung dann *terminiert*, wenn der Urprozess die Kreditsumme 1 wiedererlangt hat

---

Man muss also die Bedingungen erfüllen ("*safety*") und dafür sorgen, dass der Urprozess die gesamte Kreditsumme schliesslich wiedererhält ("*liveness*")

Kreditmethode - Realisierung

(1) Wird ein Prozess passiv, übermittelt er seinen Kreditanteil an den Urprozess.

(2) Der Kreditanteil einer ankommenden Nachricht wird dem Empfänger zugeschlagen.

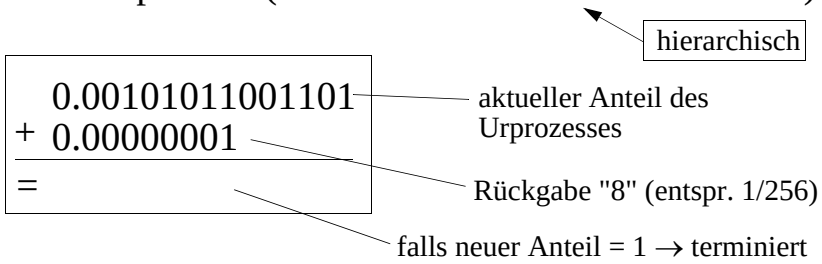
(3) Ausgesandte Nachricht erhält Hälfte des Anteils des Senders (und Sender behält andere Hälfte).

Implementierung:

(1) Problem: Gleitpunktzahlen für Kreditanteile unbrauchbar
Lösung: Bruchdarstellung

(2) Problem: Nenner schnell zu gross
Lösung: Da stets nur negative 2er-Potenzen
→ nur Exponent des Nenners ("negativer Logarithmus")
→ halbieren: KREDIT := KREDIT + 1

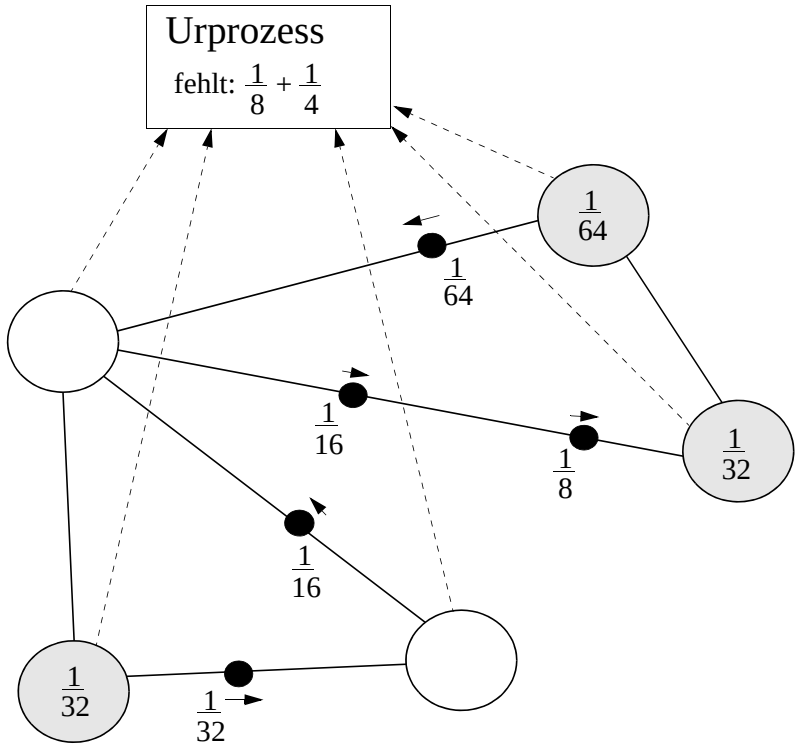
(3) Rekombination (Addition) der Anteile
beim Urprozess (bzw. auch den anderen Prozessen)



(4) Problem: Länge der Bitleiste
Lösung: Anzahl *fehlender* Kreditanteile stets beschränkt ("klein") → speichere *Komplement* als Menge

Kreditmethode - ein Beispiel

Alle Kreditanteile sind von der Form $\frac{1}{2^k}$

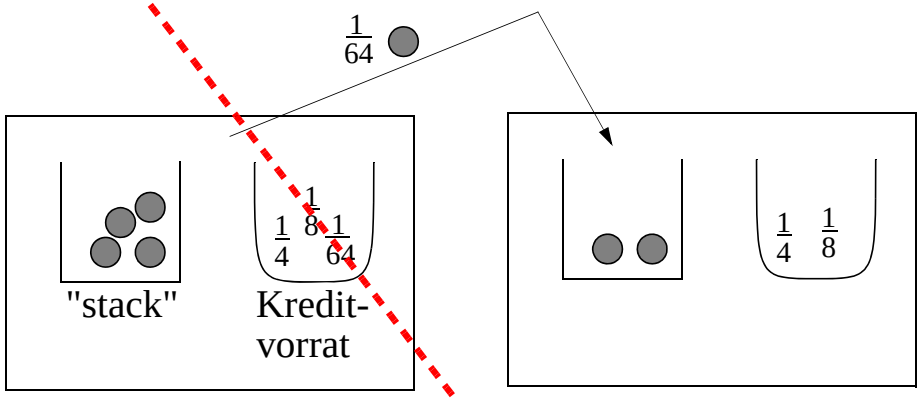


Der Urprozess muss nicht mehr "Krümel" halten, als davon im System (d.h. in aktiven Prozessen, in Basisnachrichten oder in Kontrollnachrichten zum Urprozess) sind.

(Der Urprozess kann die ihm noch fehlenden Krümel sogar oft kompakter speichern!)

Kreditmethode beim parallelen Berechnungsschema

1) Kreditanteile werden weitgehend dezentral von den Prozessoren verwaltet; lokale Arbeitseinheiten benötigen keinen Kredit.



2) Bei Versenden einer Arbeitseinheit: Diese bekommt einen Anteil aus dem lokalen Kreditvorrat

3) Lokaler Kreditvorrat darf bei einem "aktiven" Prozessor nie leer werden: evtl. muss ein Kreditanteil (der letzte) halbiert werden

4) Anteil eines ankommenden Arbeitseinheit wird dem Empfänger zugeschlagen; evtl. (falls gleicher Anteil dort schon vorhanden): iterativ rekombinieren (d.h. soweit möglich aufaddieren)

5) Wenn stack leer (oder auf explizite Anforderung): Kreditvorrat an den zentralen Prozess übermitteln

Rekombination der Kreditanteile

- Einsammeln der Kredite auch durch beliebigen Wellenalgorithmus möglich (initiiert durch Urprozess)
  - dabei evtl. Rekombination eingesammelter Kredite "on the fly"

- Rekombination der Kreditanteile beim *Urprozess*:
  - Initial: D = {0};
  - Bei Empfang von <KRÜMEL>:

steht für $2^{-KREDIT}$

```
K := KRÜMEL;
while K ∉ D begin
   D := D ∪ {K};
   K := K-1;
end;
D := D-{K}
if D = ∅ then terminated fi;
```

  - Wieso klappt dieses Schema?

Rekombination - Ein Beispiel

Initial: D = {0}

$\frac{1}{4}$ ("2") $\rightarrow \cup\{2\}, \cup\{1\}, -\{0\}$ $\rightarrow$ D = {2, 1}

$\frac{1}{2}$ ("1") $\rightarrow -\{1\}$ $\rightarrow$ D = {2}

$\frac{1}{64}$ ("6") $\rightarrow$ $\rightarrow$ D = {6, 5, 4, 3}

$\frac{1}{8}$ ("3") $\rightarrow$ $\rightarrow$ D = {6, 5, 4}

$\frac{1}{64}$ ("6") $\rightarrow$ $\rightarrow$ D = {5, 4}

$\frac{1}{16}$ ("4") $\rightarrow$ $\rightarrow$ D = {5}

$\frac{1}{32}$ ("5") $\rightarrow$ $\rightarrow$ D = {}

$\Rightarrow$ *Terminierung!*

$\Sigma = 1$

Prinzip: Binäre Subtraktion!

Kreditmethode - Bewertung

- Topologievoraussetzungen?
- Zentraler Urprozess: Engpass?
  - evtl. weitere Hierarchiestufen einführen
  - Kreditanteile durch einen Wellenalgorithmus einsammeln
- (Worst-case) Nachrichtenkomplexität?
  - wenn passiv werdende Prozesse ihren Kreditanteil stets zurücksenden
  - wenn nicht mehr benötigte Anteile von einer Welle eingesammelt werden
- lok. Speicheraufwand? (Insbes. beim Urprozess)
  - wie gross kann die Menge dort gehaltener "Krümel" werden?
- lok. Berechnungsaufwand?
- "Detection Delay" (nach erfolgter Terminierung)?
- Qualitativer Vergleich mit anderen Verfahren?
- Varianten?

- Es gilt: Es gibt keinen Algorithmus zur Feststellung der verteilten Terminierung, der eine bessere Worst-case-Nachrichtenkomplexität als O(m+n) hat (ohne Beweis)
  - m = Anzahl der Basisnachrichten; n = Anzahl der Prozesse
  - wie könnte man eine solche Aussage beweisen?
  - Kreditmethode ist daher "worst-case-optimal"!

Variante: "Nachlaufverfahren"

- Idee: Kontrollnachrichten laufen auf "benutzten" Kanälen den Basisnachrichten hinterher, um die Kredite zurückzufordern
- Überholen dabei keine Basisnachrichten (wie realisieren?)
- Spalten sich bei Prozessen "geeignet" auf
- Kehren (direkt oder indirekt) zum Urprozess zurück, wenn kein benutzter Kanal mehr existiert

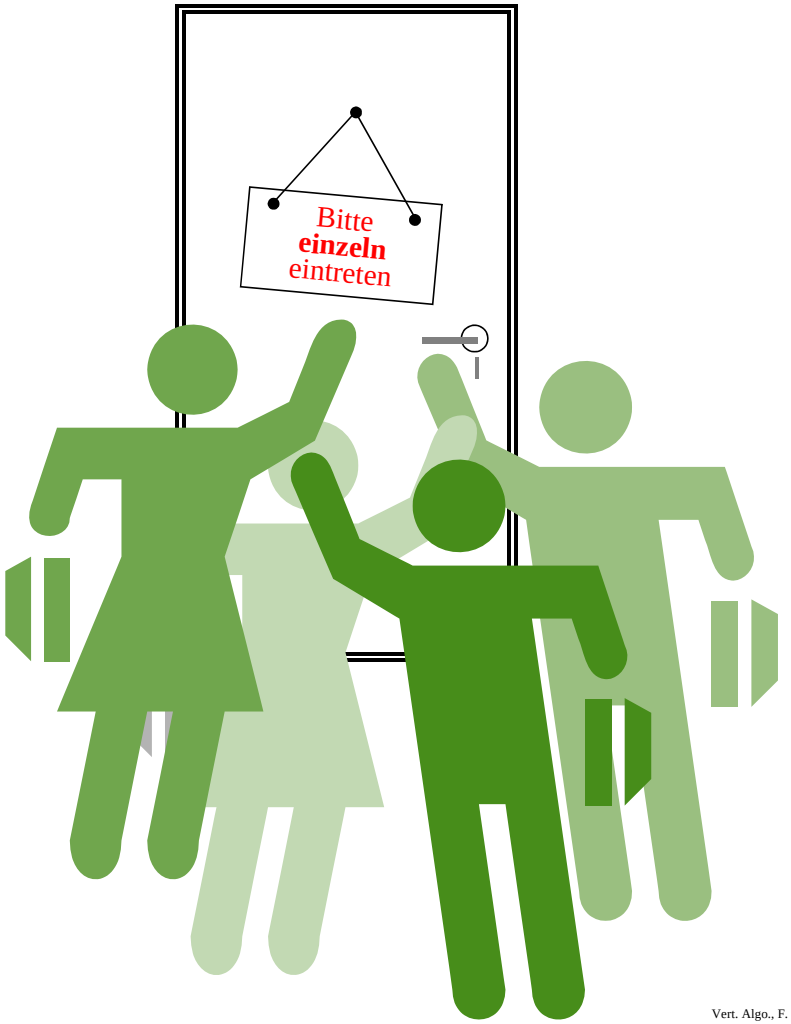
---

- Eine spezielle Ausprägung davon:
  Hinter *jeder* Nachricht *direkt* herlaufen, um den Kredit (über diesen Kanal) baldmöglichst wieder zurückzuholen
  - *Diesen* Typ von Kontrollnachrichten kann man sich sparen! ("Verheiraten" mit der zugehörigen Basisnachricht)
  - Kreditrückgabe wird sofort initiiert, wenn der Empfänger schon aktiv ist
  - Aber wenn der Empfänger passiv war, wann dann?
    - Empfänger wartet mit der Rückgabe des Kredits (d.h. mit dem Acknowledgement), bis er selbst Acknowledgements zu allen von ihm selbst ausgesendeten Nachrichten erhalten hat
  - Kreditwerte zu versenden, kann man sich sparen: Man bekommt schliesslich genau den Wert zurück, den man versendet hat!
  - Was bleibt vom Verfahren also übrig? ($\rightarrow$ Neuformulierung: Jeder Prozess zählt gesendete Nachrichten und empfangene Acknowledgements...)

Diese Idee *indirekter Acknowledgements* ("diffusing computations"-Verfahren von Dijkstra und Scholten, 1980) erinnert sehr an den *Echo-Algorithmus!*

Man könnte also überhaupt verteilte Berechnungen ganz allgemein als Instanzen (einer einfachen Variante) des Echo-Algorithmus hinsichtlich der Explorer-Nachrichten auffassen. Die Echo-Nachrichten übernehmen dann, wie gehabt, die Erkennung der Terminierung!

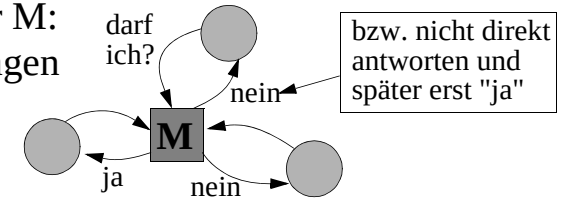
Wechselseitiger Ausschluss



Wechselseitiger Ausschluss: Request- (bzw. Permission)-basierte Prinzipien

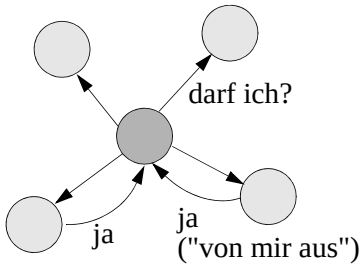
1) Zentraler Monitor M:
Einen einzigen fragen

- Engpass, schlecht skalierbar, nicht fehlertolerant



2) *Alle* fragen

- dezentral und symmetrisch, aber viele Nachrichten
- Algorithmen von Lamport (1978) und Ricart / Agrawala (1981) → Vorlesung "Verteilte Systeme"



3) Kompromiss: dezentrale, symmetrische Lösung, bei der nur *einige* Prozesse um Erlaubnis gefragt werden?

- es müssen natürlich "ausreichend viele" sein

Fairnessforderung: Jeder request wird “schliesslich” erfüllt

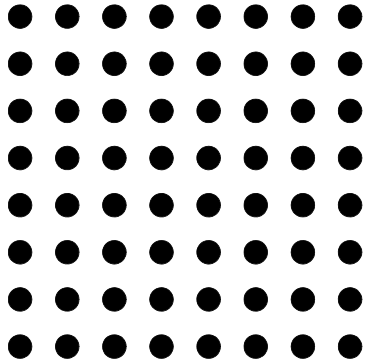
- d.h., es wird z.B. keiner ausgeschlossen, weil zwei andere sich dauernd "den Ball zuspielen"

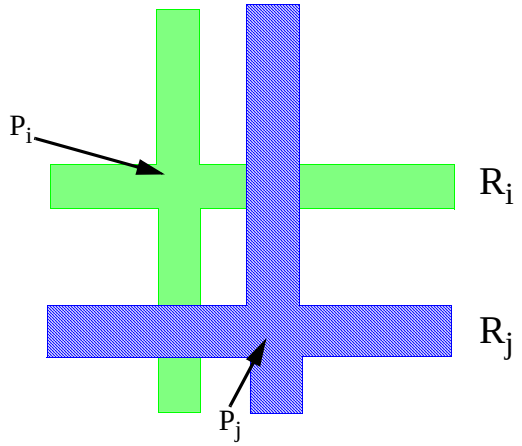
Maekawa’s $\sqrt{n}$ -Algorithmus

"... the algorithm is optimal in terms of the number of messages..."

Idee *in etwa:*

- Anordnung der Prozesse in einem $\sqrt{n} \times \sqrt{n}$ - Gitter

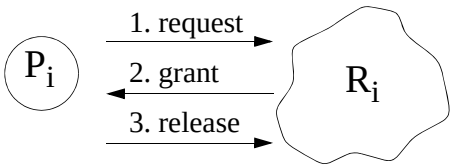




- Prozess P_i hat eine Menge von Prozessen R_i, die er (mit request-Nachrichten) *um Erlaubnis fragen* muss
  - hier symbolisiert durch Prozesse in der Spalte / Zeile von P_i

- Die "request-granting" Mengen für je zwei Prozesse *überschneiden* sich garantiert! ($\forall i,j: R_i \cap R_j \neq \emptyset$)

- Grundidee:

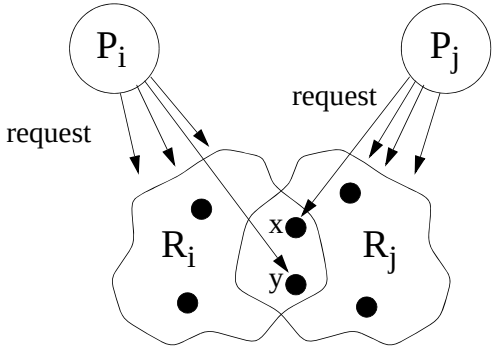


- Nachrichtenkomplexität: $3\,|R_i|$
  → minimale Mächtigkeit der R_i?

Ein Prozess wartet auf "grant" seiner Menge. Erst dann darf er den kritischen Abschnitt betreten. Nach Verlassen die Menge mit "release" informieren.

Ein "grant" wird zu einem Zeitpunkt nur *einem einzigen* Bewerber erteilt.

Deadlock-Problematik



Beachte: Zweckmässigerweise ist i.Allg. $P_k \in R_k$, falls dies möglich ist.

Im Szenario könnte dann z.B. $x = P_j$ und $y = P_i$ sein.

- y antwortet P_i mit "grant", nicht jedoch P_j
- x antwortet P_j mit "grant", nicht jedoch P_i

$\Rightarrow$ *Deadlock*, P_i und P_j warten auf weitere Zusage!

Lösung erfordert weitere Nachrichtentypen zur Deadlockvermeidung (bzw. Deadlockbehebung)

$\Rightarrow$ soll hier nicht behandelt werden ($\rightarrow$ Literatur)

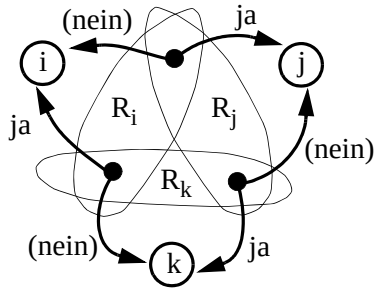
$\Rightarrow$ erhöht die Nachrichtenkomplexität jedoch nur um konstanten Faktor

---

Ist auch ein Deadlock möglich, wenn $|R_i \cap R_j| = 1$ für alle i, j ?

Ja, siehe nebenstehendes Szenario!

- Prozese i, j und k wenden sich gleichzeitig an ihre entsprechenden Mengen
- jeweils ein Prozess daraus antwortet mit "ja"



Gitteranordnung ist nicht optimal

- Zu jedem Paar von Prozessen P_i, P_j sind mindestens *zwei* Elemente im Schnitt von R_i, R_j (statt minimal einem)

- "Minimale" R_i können mit Hilfsmitteln der *projektiven Geometrie* bestimmt werden ($\rightarrow$ $|R_i| \approx \sqrt{n}$ statt $2\sqrt{n}-1$)

  - *Endliche projektive Ebene* entsteht aus affiner Ebene unter Hinzunahme von "uneigentlichen Punkten" (=Parallelenbündel) und der "uneigentlichen Geraden" (=Menge aller u. Punkte) und erfüllt folgende 3 Axiome:

    - zu je zwei Punkten gehört genau eine damit inzidente Gerade
    - zu je zwei Geraden gehört genau ein gemeinsamer Punkt
    - es gibt 4 Geraden, wovon keine 3 durch den selben Punkt gehen

  - Für eine *endliche projektive Ebene der Ordnung k* gilt:

    - jede Gerade enthält k Punkte
    - jeder Punkt liegt auf k Geraden

    } typischer *Dualismus* der projektiven Geometrie!

    - Ebene hat k(k-1)+1 Punkte und ebensoviele Geraden

  - *Idee*: Punkte $\Leftrightarrow$ Prozesse, Geraden $\Leftrightarrow$ request-granting-Mengen R_i.
    $n = k(k-1)+1 \rightarrow$ Jede Menge enthält $O(\sqrt{n})$ Elemente

  - *Leider* existiert nicht für jedes k eine endliche projektive Ebene der Ordnung k (mindestens jedoch dann, wenn k eine Primzahlpotenz ist); 1988 gezeigt (3000 Stunden Rechenzeit, erschöpfende Suche): es gibt keine projektive Ebene der Ordnung 10

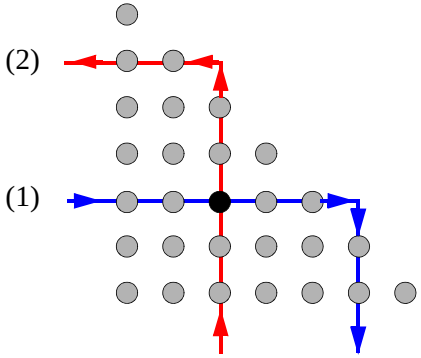
Beispiel n=13, k=4:
(Überprüfe, ob $|R_i \cap R_j| = 1$)

Beachte: Umnumerierung, so dass $i \in R_i$ (für alle i) ist möglich und zweckmässig!

$R_1 = \{1, 2, 3, 4\}$
$R_2 = \{1, 5, 6, 7\}$
$R_3 = \{1, 8, 9, 10\}$
$R_4 = \{1, 11, 12, 13\}$
$R_5 = \{2, 5, 8, 11\}$
$R_6 = \{2, 6, 9, 12\}$
$R_7 = \{2, 7, 10, 13\}$
$R_8 = \{3, 5, 10, 12\}$
$R_9 = \{3, 6, 8, 13\}$
$R_{10} = \{3, 7, 9, 11\}$
$R_{11} = \{4, 5, 9, 13\}$
$R_{12} = \{4, 6, 10, 11\}$
$R_{13} = \{4, 7, 8, 12\}$

Eine $\sqrt{2}\sqrt{n}$ Request-granting-Menge

- Prozesse werden *dreiecksförmig* angeordnet:



Schema 1 für R_i: Nimm alle Prozesse der *Zeile* von P_i sowie alle Prozesse derjenigen *Spalte*, die eins weiter rechts liegt als der rechteste Prozess der Zeile

Schema 2 für R_i: Nimm alle Prozesse der *Spalte* von P_i sowie alle Prozesse derjenigen *Zeile*, die eins weiter oben liegt als der oberste Prozess der Spalte

- Für beide Schemata gilt jeweils: Je zwei solche Mengen schneiden sich an mindestens einer Stelle (wieso?)

- Beispiel:

1
2 3
4 5 6
7 8 9 10

Mächtigkeit der R_i: ca. $\sqrt{2}\sqrt{n}$ (wieso?)

Schema 1	Schema 2
$R_1 = \{1, 3, 5, 8\}$	$R'_1 = \{1, 2, 4, 7\}$
$R_2 = \{2, 3, 6, 9\}$	$R'_2 = \{1, 2, 4, 7\}$
$R_3 = \{2, 3, 6, 9\}$	$R'_3 = \{1, 3, 5, 8\}$
$R_4 = \{4, 5, 6, 10\}$	$R'_4 = \{1, 2, 4, 7\}$
$R_5 = \{4, 5, 6, 10\}$	$R'_5 = \{1, 3, 5, 8\}$
$R_6 = \{4, 5, 6, 10\}$	$R'_6 = \{2, 3, 6, 9\}$
$R_7 = \{7, 8, 9, 10\}$	$R'_7 = \{1, 2, 4, 7\}$
$R_8 = \{7, 8, 9, 10\}$	$R'_8 = \{1, 3, 5, 8\}$
$R_9 = \{7, 8, 9, 10\}$	$R'_9 = \{2, 3, 6, 9\}$
$R_{10} = \{7, 8, 9, 10\}$	$R'_{10} = \{4, 5, 6, 10\}$

- Schema 1 und 2 sind jeweils für sich unausgewogen: Prozess 10 bzw. 1 kommt viel häufiger als andere vor

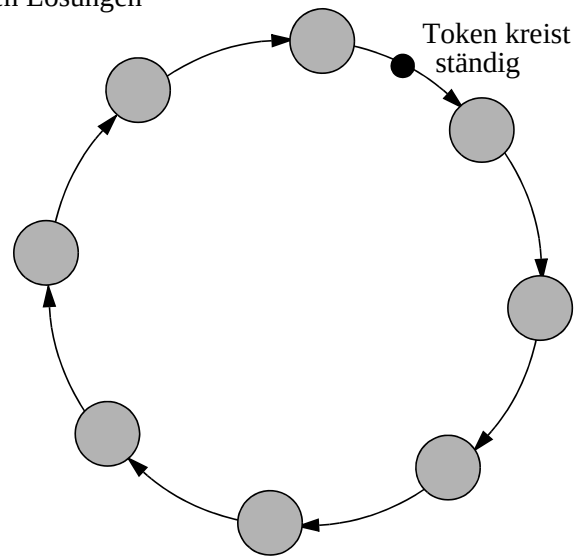
  - aber: jede Zahl kommt genau 8 mal in *allen* Mengen vor (Lastsymmetrie!) (Denkübung: Beweis, dass *stets* alle gleich oft vorkommen)
  - Vorschlag: "alternierende" Benutzung der beiden Schemata (wie?) Frage: Überschneiden sich R_i (Schema 1) und R'_j (Schema 2) jeweils? (Gilt immer $i \in R_i$ und $i \in R'_i$?)

Man kann zeigen:
$|R_i| = O(\sqrt{n})$ ist optimal
(für "symmetrische" Lösungen).

Wir lassen das hier aber weg.

Die Token-Ring-Lösung

Als Alternative zu den Request-/Permission-basierten Lösungen



- Nur der Tokeninhaber darf
  - Safety ist klar
  - Liveness: Token muss weitergegeben werden
  - Fairness intuitiv gegeben

- Probleme?
  - Bei vielen Prozesse → lange Wartezeiten, Gefahr von Tokenverlust
  - Anzahl der Einzelnachrichten nicht begrenzt (ständiges Kreisen)
  - Für jedes Betriebsmittel eigenes Token vorsehen?

Token-Request-basierte Algorithmen

- Token soll nicht dauernd nutzlos unterwegs sein
  - Token wandert nur bei Bedarf

- Grundidee: "Token zu mir" an alle (?) anderen senden
  - per *broadcast* (falls entspr. Kommunikationsprimitiv existiert)
  - oder z.B. mittels *Echo-Algorithmus*
  - Aufwand ist hoch, wenn man nicht weiss, wo das Token sein könnte

- Fairness muss aber gewahrt bleiben

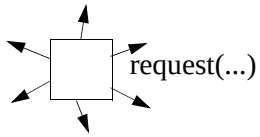
- Safety ist vergleichsweise trivial (Tokenbesitz)

- Liveness mit möglichst wenig Aufwand garantieren

Algorithmus von Ricart / Agrawala bzw. Suzuki / Kasami

1) Es gibt ein einziges Token; nur Besitzer darf den kritischen Abschnitt betreten

2) Request-Nachricht mit Zeitstempel an *alle* senden



3) Token hat "Gedächtnis":

Prozess-nummer

Zeitstempel des letzten Besuchs (evtl. implizit 0, wenn Prozess noch unbekannt)

1	xxx
2	xxx
3	xxx
:	:
n	xxx

4) Jeder Prozess registriert Anforderungen *aller* anderen

5) Nach Verlassen des kritischen Abschnitts eines Prozesses wandert das Token zu denjenigem Prozess, der am "längsten" wartet. (Beim Prozess vermerkte Anforderung muss jünger als der im Token registrierte Zeitstempel des letzten Besuchs sein)

→ Nachrichtenkomplexität: n (n-1 für request, + 1 Token)
(bzw. 0, wenn inzwischen kein anderer wollte)

Fragen:
- Liveness? Fairness?
- Inwiefern kann man statt physischer Zeit auch einen logischen Zeitbegriff verwenden (z.B. "Lamport-Zeit")?
- Wie lange muss ein Prozess max. (auf Mitbewerber) warten?
- Ist FIFO notwendig?

Token-Request-basierte Algorithmen

aber wie bekommt man diesen?

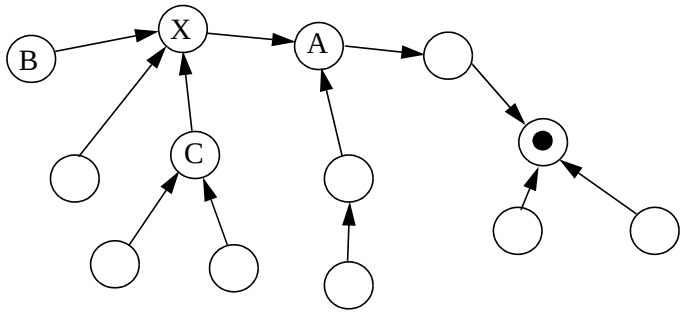
- Andere Idee (statt Broadcast): *Spannbaum* verwenden
  - "Suchnachrichten" wandern nur auf Kanten des Baumes
  - Token benutzt ebenfalls nur Baumkanten
  - Aufwand: Maximal n-1 Einzelnachrichten, um jeden der n Knoten über den Tokenwunsch zu informieren

---

- Bessere Idee: Spannbaum mit *gerichteten Kanten*
  - Kanten zeigen immer in Richtung des Tokenbesitzers
  - typischerweise O(log n) Einzelnachrichten, um den Tokenbesitzer zu erreichen (aber bei speziellen, z.B. bösartig entarteten Bäumen?)
  - wenn Token seinen Ort wechselt, müssen Kantenrichtungen aktualisiert werden! (Aufwand?)
  - → "Lift-Algorithmus" nächste Seite

Der Lift-Algorithmus

(van de Snepscheut; Raymond; Naimi-Trehel)



- Token wandert entgegen der Pfeilrichtung zum anfordernden Prozess und dreht jede durchlaufene Kante um
- Ein Prozess sendet (bis er das Token erhält) nur ein einziges Mal einen request auf seiner ausgehenden Kante
  - nochmals "ungeduldig" nachfragen hilft auch nichts
  - Prozess merkt sich aber, wer ihn alles um Weiterleitung des requests bat
- Bei Empfang des Tokens wird es (in fairer Weise) in eine der anfordernden Richtungen weitergeleitet
  - vorher selbst benutzen, wenn Bedarf besteht? ← vgl. "Lift"
  - Fairness: Nur beschränkt oft in eine andere Richtung weiterleiten, bevor es über eine "wartende Kante" weitergeleitet wird
  - falls Anforderungen aus mehreren Richtungen vorliegen: Dem ausgesendeten Token sofort einen request hinterhersenden (Optimierung: Token und request zusammenfassen)
- Denkübung: Sind Nachrichtenüberholungen (request überholt Token) ein Problem?

Welcher Baum beim Lift-Verfahren?

- Baum mit (inneren) Knoten vom *Grad k*:

  → längster Weg hat Länge $O(\log_k n)$; durchschnittliche Weglänge ebenfalls

  → bestimmt die Nachrichtenkomplexität bei *schwacher Last*

  - Knoten im "Zentrum" werden mehr belastet als weiter aussen liegende Knoten!

  - Grad an Fairness?

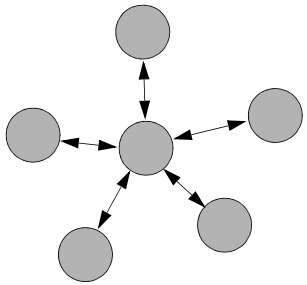
  fast nie mehrere gleichzeitige Konkurrenten

Nachrichtenkomplexität bei *starker Last*:

- Idee: Jedes Weitersenden des Tokens ein "Treffer"
- Genauer: Token *traversiert* den Baum (wegen Fairness)
  - → n Knotenbesuche bei 2(n-1) Token-Nachrichten
  - → weiterer Faktor 2, da gleiche Anzahl von requests
  - → ca. 4 Nachrichten pro Betreten des kritischen Abschnittes

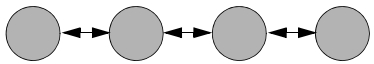
Algorithmus wird bei starker Last "besser"! (Overhead-Amortisation)

- *Stern*:



$O(\log_n n) = O(1)$

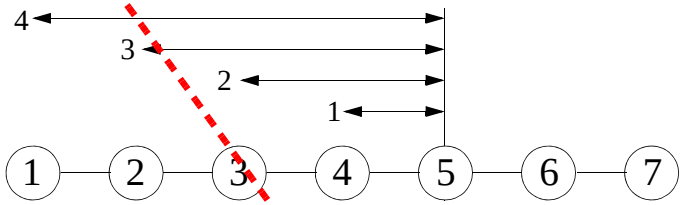
- Kürzeste Wege → beste Topologie? (D.h. wenn schon Baum, dann diesen?)
- Unterschied zur Lösung mit zentralem Manager? (Und was war daran eigentlich schlecht?)
- Beachte: Einen Spannbaum gibt es immer; ein Stern erfordert aber u.U. zusätzliche (logische oder physische) Verbindungen!

- *Lineare Kette* als entarteter Baum 

→ mittlere Weglänge ≈ n/3 (Beweis: einfaches kombinator. Nachrechnen)

Mittlere Weglänge im linearen Fall

(Für diejenigen, denen das Rechnen Spass macht)



1
1 + 2
1 + 2 + 3
1 + 2 + 3 + 4
. . . . . .
1 + 2 + 3 + (n-1)

$$\sum_{j=1}^{n-1}\sum_{i=1}^{j} i = \sum_{j=1}^{n-1}\frac{j(j+1)}{2} = \frac{1}{2}\sum_{j=1}^{n-1} j^2 + \frac{1}{2}\sum_{j=1}^{n-1} j$$

Denn es gilt:

$$\sum_{k=1}^{n} k^2 = \frac{n\,(n+1)\,(2n+1)}{6}$$

$$= \frac{1}{2}\,\frac{(n-1)\,n\,(2(n-1)+1)}{6} + \frac{1}{2}\,\frac{n\,(n-1)}{2}$$

Gemittelt über alle n(n-1)/2 ungerichteten Paare {i,j} mit i≠j:

$$\rightarrow \quad \frac{2(n-1)+1}{6} + \frac{1}{2} = \frac{2n+2}{6} = \frac{1}{3}(n+1) \approx \underline{\underline{n/3}}$$

(evtl. abzüglich der Fälle, wo das Token bereits "vor Ort" ist).

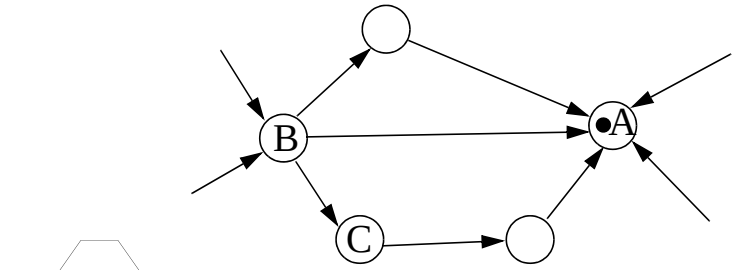
Verallgemeinerung auf allg. Graphen

- Gerichtet, azyklisch und schwach zusammenhängend
- Kann man die Kanten jedes ungerichteten Graphen so orientieren, dass kein Zyklus entsteht?
  - Bsp. Stadtplanung: Einbahnstrassenrichtung so festlegen, dass man nicht im Kreis fahren kann
  - Ja: Willkürliche Ordnung auf den Knoten festlegen (z.B. Knoten nummerieren) und Kanten entsprechend dieser Ordnungsrelation orientieren!

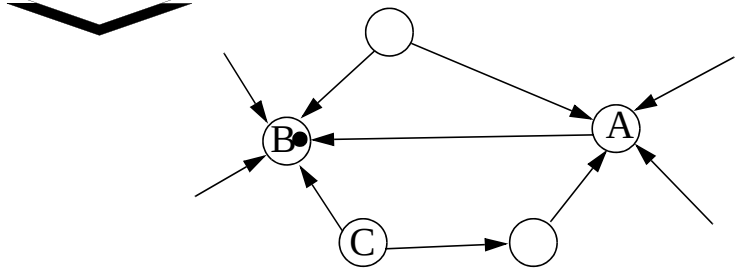
- Präzisere Forderung: Jeder gerichtete Weg soll beim (eindeutigen) Tokenbesitzer enden
  - Klar, dass dann Tokenbesitzer nur eingehende Kanten hat
  - Auch das geht!
    - Starte Echo-Algorithmus vom Tokenbesitzer aus
    - Durch die Flussrichtung der Echos entsteht ein auf diesen Knoten hin gerichteter Baum
    - Nun müssen noch die Nicht-Baumkanten orientiert werden
    - Dazu bekommt jeder Knoten eine Ordnungsnummer aus:
      - seiner Höhe (= Entfernung zur Wurzel, d.h. dem Tokenbesitzer) als primärem Ordnungskriterium,
      - einem eindeutigen sekundären Ordnungskriterium
    - "Flussrichtung" der Kanten von der höheren zur niedrigeren Ordnungsnummer → ist azyklisch!

Tokenanforderung

- Schicke request über irgendeine ausgehende Kante
  - klappt mit jeder Kante (Weg zum Token ist aber i.Allg. verschieden lang)
  - auch denkbar: über alle Pfade gleichzeitig (→ mehr Aufwand!)



Regel: Wenn das Token wandert, werden *alle* Ausgangskanten des Empfängers invertiert



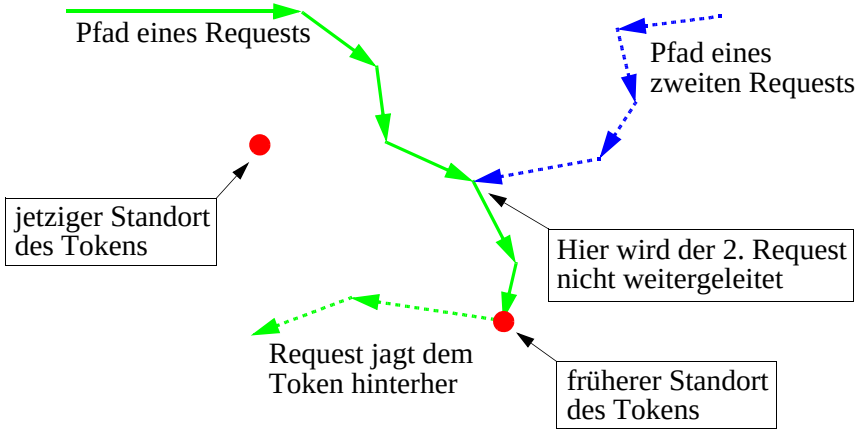
- Behauptung: *Zyklenfreiheit ist eine Invariante*

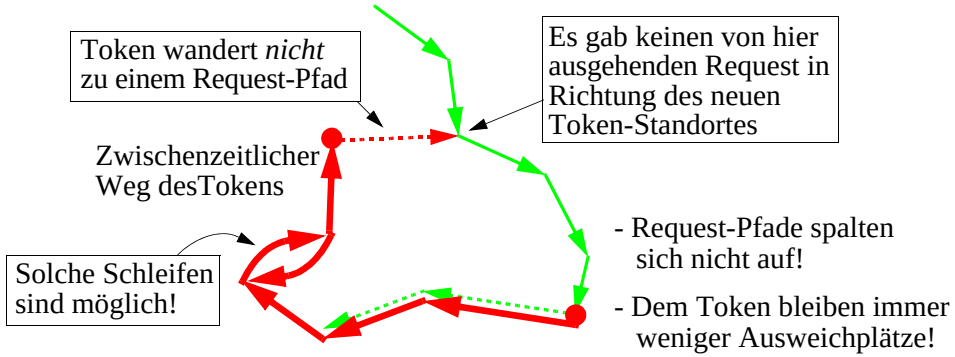
Beweis: Neue Zyklen können höchstens durch die invertierten Kanten entstanden sein - mit diesen ist jedoch kein Zyklus möglich, da sie alle zum neuen Tokenbesitzer gerichtet sind, welcher keine ausgehenden Kanten besitzt

- Weitere Invariante: *Jeder gerichtete Pfad führt zum Tokenbesitzer*

Glücklose Token-Jagd?

- Wieso holt ein Request das Token schliesslich immer ein?





Zum Lift-Algorithmus auf allg. Graphen

- Wenn die Richtung einer Kante als "Wegweiser" im ausgehenden Knoten gespeichert ist, dann muss den Nachbarn des neuen Tokenbesitzers eine Meldung gesendet werden, damit diese den Wegweiser entsprechend setzen
  - tatsächlich allen, oder kann man sich das für einige sparen?
  - "Gewinn" des Verfahrens: Nachbarn haben nun i.Allg. einen kürzeren Weg zum Token
  - Denkübung: muss der Empfang aller dieser Meldungen abgewartet werden (Quittungen!), bevor das Token weiterreisen darf?

- Variante: Statt alle Nachbarn zu informieren:
  - Nur die Richtung der einen vom Token durchlaufenen Kante ändern

    → spart Meldungen an Nachbarn
  - Beim neuen Tokenbesitzer alle ausgehenden "Wegweiser zum Token" löschen

    → es entstehen ungerichtete Kanten, die zukünftig nicht mehr benutzt werden!

- Denkübungen:
  - Vergleich dieser Variante mit obigem "Originalverfahren"
  - Vergleich mit spannendem gerichteten Baum als Topologie
  - welche Entfernung legt das Token im Mittel zurück?
  - ist ein vollst. Graph ein interessanter Sonderfall?

Klassifikation: Token ⇔ Request

1) Token-basierte Lösungen für wechselseitigen Ausschluss

- Safety ist trivial
- Fairness bei Tokenweitergabe beachten
- Wie fordern Prozesse das Token an? ← → unterschiedliche Lösungen
- Topologie
  - → "Reiseweg des Tokens"
  - → Zeitaufwand durch sequentielle Nachrichtenketten
- Fehlertoleranz:
  - wie Tokenverlust feststellen?
  - wer darf neues Token generieren? (Eindeutigkeit notwendig!)
- Nur anwendbar, wenn für das exklusive Betriebsmittel von vornherein ein Token eingerichtet wird → nicht immer möglich!

- Beispiel für *a priori unbekannte exklusive Betriebsmittel:* "Reservierungszeiten für einen Tennisplatz"
  - "Ich benötige ihn übermorgen von 10.28 - 12.17 exklusiv"
  - Dafür lässt sich nicht von vornherein ein Token generieren!
  - Vielleicht: ein Token "Tennis <Zeitintervall>" dynamisch generieren?
  - Aber: wer garantiert, dass ein anderer dies nicht gleichzeitig tut?
  - Zurück zu einer zentralen Lösung mit allen Nachteilen? ("Tennisplatz als Monitor")
- Anderes Bsp. für ein "abstraktes Betriebsmittel": Terminvereinbarung mit einer beliebigen Menge von Teilnehmern
- Exklusives Generieren eines Tokens unter symmetrischen Bedingungen → *Election-Problem* (→ später)

Klassifikation (2)

2) Request-basierte Lösungen

- wen sollen die Prozesse fragen? ("request set")
- Safety sicherstellen!
- Deadlockfreiheit ist nicht trivial

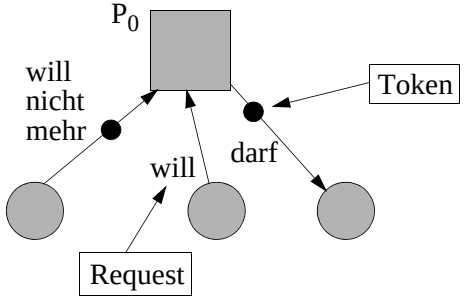
} hierfür existieren verschiedene Lösungen

---

3) Zentraler Manager

bzw. Server oder Monitor

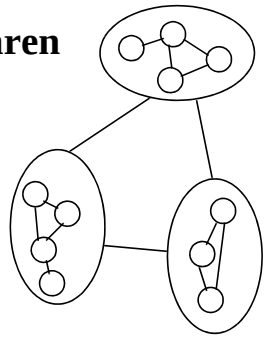
- ist entarteter Fall *beider* Klassen!



---

4) Hierarchische / hybride Verfahren

- bei grossen Systemen mehrstufig (mittels "Stellvertreter")
- auf verschiedenen Stufen / in verschiedenen Clustern evtl. unterschiedliche Verfahren



Wechselseitiger Ausschluss: Kriterien

- Nachrichtenkomplexität

für die Qualität eines Lösungsalgorithmus

- Symmetrie
  - syntaktisch: gleicher Algorithmus für alle
  - semantisch: gleiche Last für alle etc.

Inwiefern würde man den Lift-Algorithmus als symmetrisch bezeichnen?

- Fehlertoleranz
  - Verhalten des Algorithmus bei Fehlern
    - z.B. bei Nachrichtenverlust
    - oder nach Abbruch von aussen wegen Deadlock der Anwendung
  - Zusatzaufwand, um (etwa bei erkanntem Fehler) wieder einen konsistenten Zustand herzustellen

- Grad an Fairness
  - inwieweit werden Requests in ihrer zeitlich globalen Reihenfolge bedient??

- Zeitbedarf zwischen Freigabe und Benutzung durch einen anderen Prozess
  - (minimale) Länge sequentieller Nachrichtenketten
  - Bsp.: beim Lift-Algorithmus ist Zeitbedarf i.Allg. O(log n) (statt O(1) wie bei einigen anderen Algorithmen)!
  - verschiedene Lastsituationen berücksichtigen:
    - schwache Last → nur selten mehr als ein Konkurrent
    - hohe Last → Betriebsmittel fast ständig in Benutzung

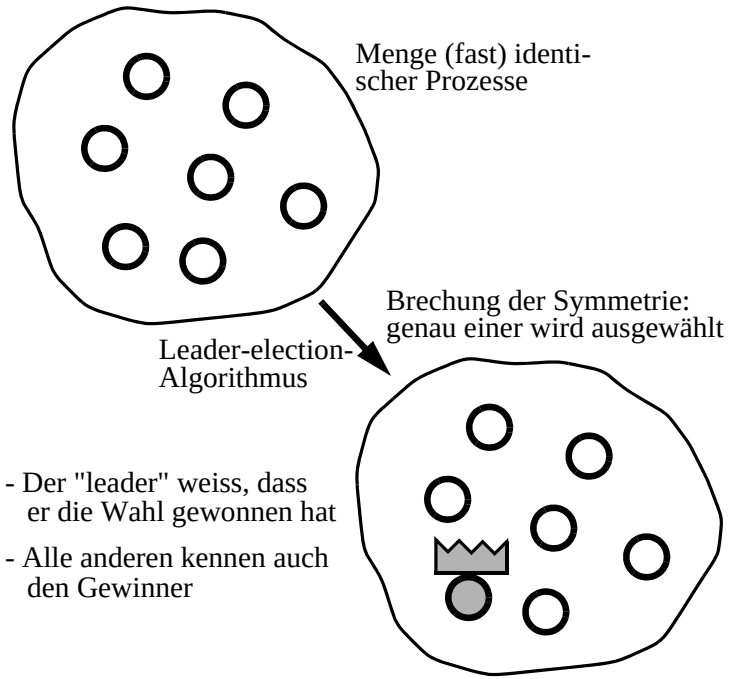
- Effizienz / Einfachheit der Implementierung
  - z.B.: wie wird broadcast / multicast ("request an alle") realisiert? (als effiziente Systemoperation; auf Ring; mit Echo-Algorithmus...)
  - wird eine spezielle Topologie vorausgesetzt (Ring, Baum,....) bzw. muss jeder Prozess jeden anderen kennen?

Übungen (4)

(1) Fischer Alois fängt jede Minute einen Fisch.
(a) Wie lange fischt er im Mittel bei der "Stoppregel" *grösserer Fisch als der erste gefangene oder kein Fisch mehr übrig* bei einem Teich/See/Meer von n verschieden grossen Fischen mit z.B. n=100, n=1000, $n=10^6$, $n=2^{31}$, $n=10^{13}$?
(b) ... wenn schneller Fische geboren werden als Alois fangen kann? (Wobei Fische nicht wachsen, sondern gleich mit ihrer endgültigen Grösse geboren werden.)
(c) ... wenn "grösserer" durch "kleinerer" in der Stoppregel ersetzt wird? Diskutieren Sie dies im Vergleich zu (a)!
Sie mögen dies simulieren, wenn Sie an den theoretischen Ergebnissen zweifeln. Mitteln Sie dabei jedes "Makroexperiment" über viele, typischerweise O(n), Einzelexperimente (also Kollegen von Alois). Vergleichen Sie die Mittelwerte mehrerer (Makro)experiment (stabil, verlässlich?).Wenn Sie stochastisch simulieren, achten Sie auf einen guten Zufallszahlengenerator. (Lesen Sie dazu z.B.: PARK, S.K. and MILLER, K.W., *Random Number Generators: Good Ones are Hard to Find*, Comm. of the ACM 31:10, pp. 1192-1201, 1988.)

(2) Wartezeit bis zum ersten echten Rekord.
(a) Nach wievielen Jahren ist in einem Jahrhundert im Mittel der erste echte (d.h. verschieden vom ersten Jahr, das immer einen unechten "Rekord" darstellt) Rekord "kältester Januar" fällig?
(b) Und in einem Jahrtausend? Darf hier etwas anderes herauskommen?
(c) Wie wahrscheinlich ist es, dass in einem Jahrhundert bzw. Jahrtausend der erste echte Rekord gerade auf das 3. Jahr fällt? Und auf das 100.? Und, bei "Jahrtausend", auf ein Jahr nach dem 100.?
(d) Gehen Sie spazieren. Wievielen Menschen begegnen Sie im Mittel, bis Sie einem grösseren begegnen? (Oder sind Sie selbst der grösste? Kann man diese Möglichkeit vernachlässigen?)

Das Election-Problem



Anwendung z.B.:

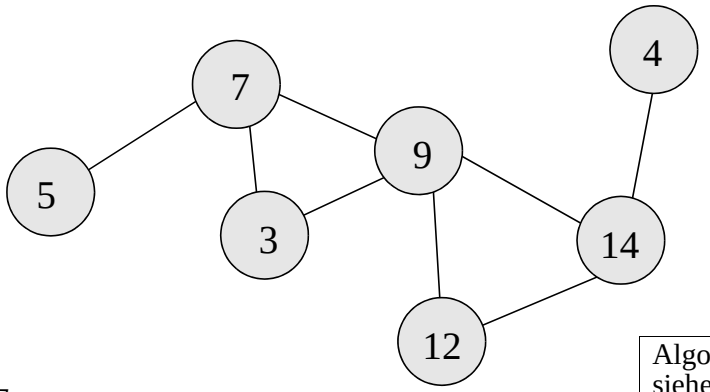
- Monitorstation bei Token-Ring-LANs festlegen
- Generierung eines eindeutigen Tokens
- "Root bridge" für Spannbaum bei Ethernet-LANs
- "Symmetrisierung" anderer Algorithmen
  (Verwendung als vorgeschalteter Basisalgorithmus)

Leader-Election mit "message extinction"

- Vorauss.: Knoten haben unterschiedliche Nummern > 0
- Gesucht: verteilter, symmetrischer Algorithmus zur Bestimmung der grössten Identität (= "Leader"), so dass
  - jeder Knoten schliesslich den grössten kennt
  - jeder Knoten unabhängig (gleichzeitig) den Algorithmus initiieren darf

---

- Bekanntes Schema der *verteilten Approximation* anwenden, um die maximale Identität zu ermitteln:



Algorithmus siehe nächste Seite

Fragen:

- Anzahl der Nachrichten?
- Terminierung?
  - z.B. mit Echos bei Echo-Algorithmus (mit "message extinction")?
- Bessere Algorithmen für gleiches Problem?
- Spezielle Topologien (Ring, Baum)...?

Der Leader-Election-Algorithmus

- Nachrichtengesteuerte Spezifikation des Algorithmus ("*message driven*")
- *Atomare Aktionen* ← vereinfacht u.a. auch die Verifikation
- Jeder Prozess mit Identität p hat lokale Variable M_p
- M_p (initial 0) enthält am Ende die Identität des Leaders

Name der Aktion

assertion bzw. "guard"

Alle Prozesse verhalten sich gleich ("syntaktische Symmetrie")

I_p: $\{M_p = 0\}$
$M_p := p;$
send $\langle M_p \rangle$ **to** all neighbours

R_p: {Eine Nachricht $\langle j \rangle$ ist eingetroffen}
if $M_p < j$ **then**
$M_p := j;$
send $\langle M_p \rangle$ **to** all neighbours
fi

andernfalls "message exctinction"

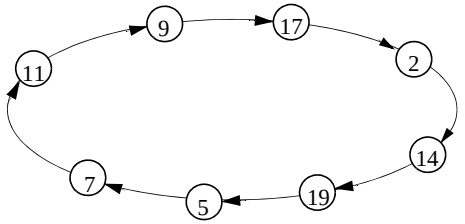
?

T_p: {Terminierung wurde entdeckt}
if $M_p = p$ **then** *"I am the leader"* **fi**

andernfalls ist M_p der Leader

Denkübung: Funktioniert das Verfahren auch dann, wenn nur einer oder einige wenige (statt alle) Prozesse starten (also I_p rechtzeitig ausführen)?

Leader-Election auf einem Ring



- Voraussetzung hier: unidirektionaler Ring, wobei alle Identitäten verschieden sind

1. Idee:
- Jeder Prozess wacht irgendwann auf (spätestens wenn er eine Nachricht von einem anderen erhält)
- Startet vollständigen Ringumlauf
- Meldet, ob unterwegs einen grösseren getroffen

"Bully-Algorithmus"

$\Rightarrow n^2$ Einzelnachrichten (n = Anzahl der Prozesse)

---

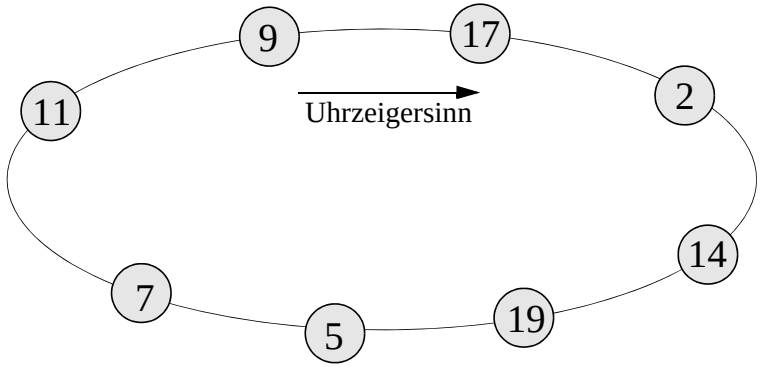
2. Idee: Message extinction anwenden!

I_p: $\{M_p = 0\}$
$M_p := p;$
send $\langle M_p \rangle$ **to** next

R_p: {Eine Nachricht $\langle j \rangle$ ist eingetroffen}
if $M_p < j$ **then**
$M_p := j;$
send $\langle M_p \rangle$ **to** next
fi
if $j = p$ **then** *"I am the leader"* **fi**

Ring-Election: Nachrichtenkomplexität

- Message-Extinction-Prinzip (Chang und Roberts, 1979)



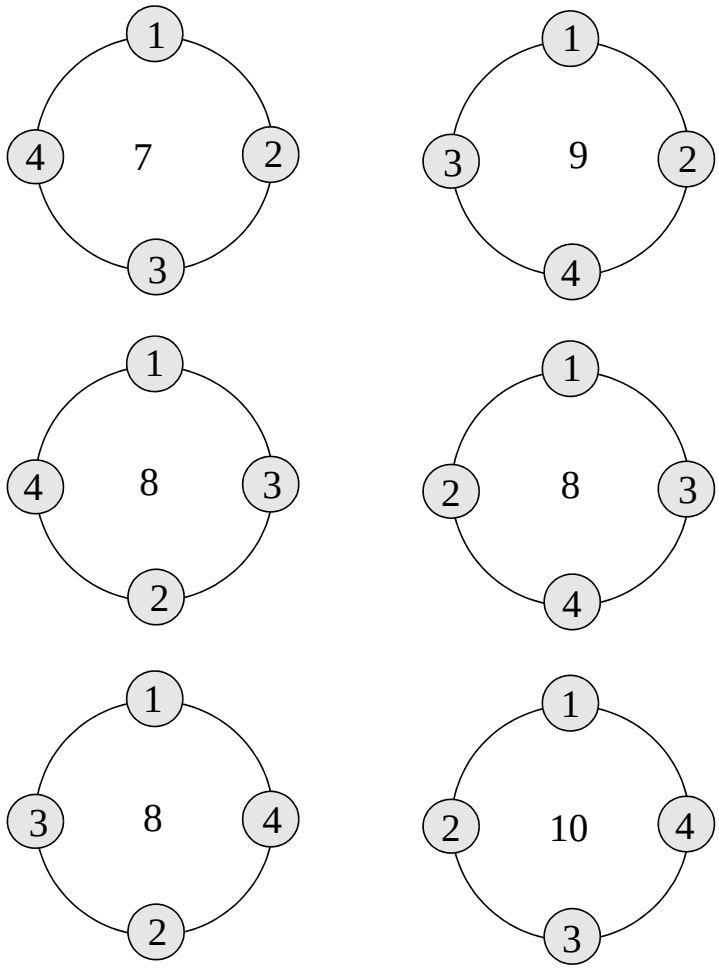
- *Worst-case Nachrichtenkomplexität* bei k Startern:
  n + (n-1) + (n-2) +...+ (n-k+1) = **nk - k(k-1)/2**
  $\rightarrow O(n^2)$ bei Ringgrösse n und k=n

- Wie hoch aber ist die *mittlere* Nachrichtenkomplexität?
  - bei “zufälliger” Permutation der Identitäten

- Denkübung: Wie hoch ist die (Einheits)zeitkomplexität?
  - wenn alle gleichzeitig starten?
  - beim zeitversetzten Starten?

Mittlere Nachrichtenkomplexität (1)

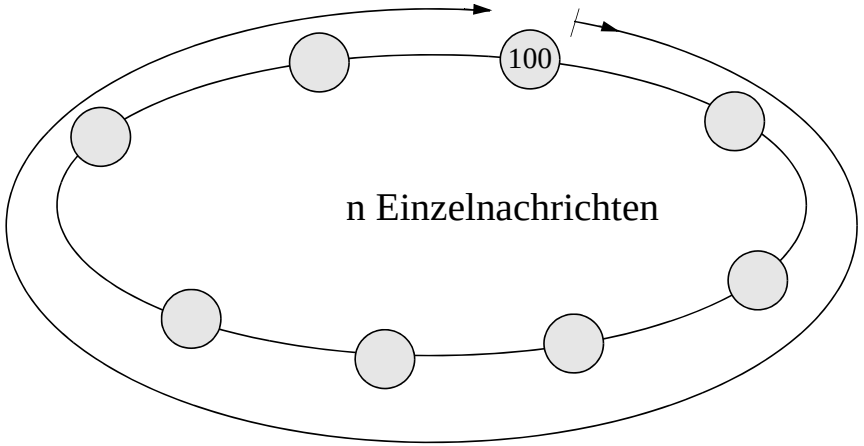
- Beispiel: Sei k = n = 4
- Über alle Permutationen mitteln (wieviele?)

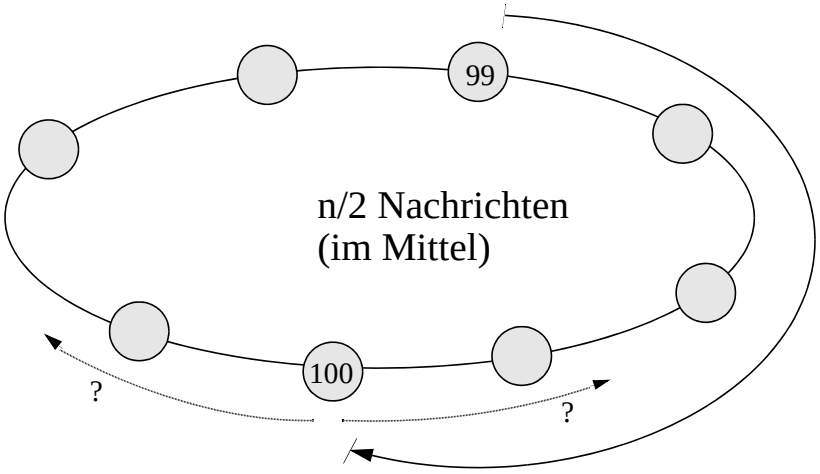


50/6 = 8.333... Nachrichten im Mittel

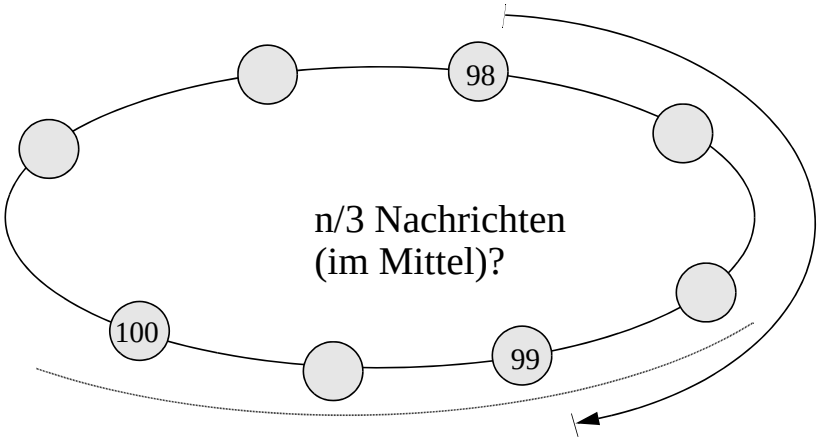
Mittlere Nachrichtenkomplexität (2)

Sei n = 100; id = 1, 2, ..., 100





Mittlere Nachrichtenkomplexität (3)



Vermutung: Drittgrösster sollte im Mittel nach n/3 Schritten auf einen grösseren Prozess (hier: 99 oder 100) treffen

- Lässt sich durch explizites Nachrechnen aller ca. n^2 Fälle auch beweisen...

- Allgemeiner Beweis für die generalisierte Vermutung "im Mittel n/i Schritte beim i-t grössten"?

Mittlere Nachrichtenkomplexität (4)

- Grösste Identität: n Nachrichten (immer)
- Zweitgrösste: im Mittel n/2
- Drittgrösste: im Mittel n/3

...

- i-t grösster: im Mittel n/i ← stimmt das?

Wenn das stimmt →

einfach aufaddieren?

$$\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \ldots + \frac{n}{k} = n \sum_{i=1}^{k} \frac{1}{i} = n\, H_k$$

Def. der sogenannten *harmonischen Reihe*

Was ergibt sich für n = k = 4 ?

$$4\,\frac{12+6+4+3}{12} = 25/3 = 8.333...$$

Stimmt mit oben berechnetem Wert überein!
⇒ *Vermutung*: $n\, H_k$ ist korrekt

- Man kann tatsächlich beweisen:

Die mittlere Nachrichtenkomplexität beträgt

$$n \sum_{i=1}^{k} \frac{1}{i} = n\, H_k \approx \mathbf{n \ln k}$$

Wir wollen dies gleich genauer herleiten...

Unabhängige Fälle?

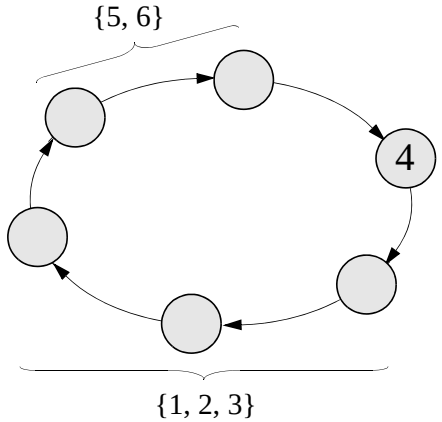
Im allgemeinen sind die "Zufallsvariablen", die die Länge der von Prozess i initiierten Nachrichtenkette repräsentieren, nicht unabhängig voneinander!

Beispiel: Gegeben 6 Prozesse 1, 2, ... 6 im Ring.
Sei *A* das Zufallsereignis "Nachrichtenkette von Prozess 4 hat die Länge 4" und *B* das Zufallsereignis "Nachrichtenkette von Prozess 5 hat die Länge 2"

Aus *A* folgt nebenstehende Situation; dann kann die von 5 initiierte Nachrichtenkette jedoch nur die Länge 1 oder 5 haben, nicht jedoch 2, wie von *B* gefordert

⇒ $\text{prob}(A \cap B) = 0$, obwohl $\text{prob}(A) \times \text{prob}(B) \neq 0$

⇒ A und B sind *nicht unabhängig* voneinander!



Allerdings (hier ohne Rechtfertigung):
Die Zufallsvariablen in obigem Beispiel sind *paarweise unkorreliert*

Die harmonische Reihe H_n

Beh.: H_n divergiert

Bew.: Fasse 4 Reihenglieder ab 1/4 zusammen,
(jew. grösser als 1/8), 8 Reihenglieder ab 1/8...

Beh. (o. Bew.) $\sum_i \frac{1}{i^r}$ konvergiert gdw. $r > 1$

($r = 2 \rightarrow \pi^2/6$ ... Riemann'sche Zeta-Funktion)

$\Rightarrow$ Harmonische Reihe "divergiert gerade noch"

Beh. (o.Bew.) $\gamma = \lim_{n \to \infty} (H_n - \ln n) = 0.5772156649...$

Euler'sche Konstante

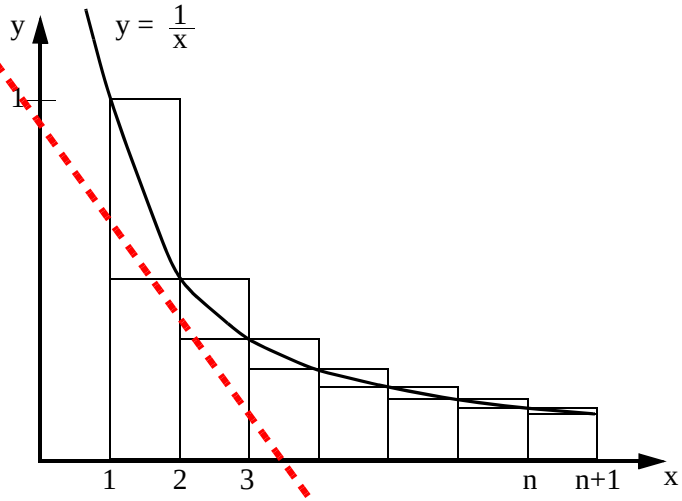
$\Rightarrow$ H_n lässt sich zumindest für grosse n durch ln n abschätzen

- diese Abschätzung lässt sich sogar noch präzisieren...

Abschätzung der harmonischen Reihe H_n

Beh.: $\ln n < H_n < 1 + \ln n$

Bew.:



- *Fläche der Obertreppe* von 1 bis n+1: H_n

- *Fläche unter Kurve* von 1 bis n+1: $\int_1^{n+1} \frac{1}{x}\, dx = \ln(n+1)$ (wg. $\ln(1) = 0$)

$\Rightarrow H_n > \ln(n+1) \Rightarrow \underline{H_n > \ln n}$

- *Fläche der Untertreppe* von 1 bis n: $H_n - 1$

- *Fläche unter Kurve* von 1 bis n: $\int_1^{n} \frac{1}{x}\, dx = \ln(n)$

$\Rightarrow H_n - 1 < \ln(n) \Rightarrow \underline{H_n < 1 + \ln n}$

Wartezeit bis zum ersten Rekord

- Wie weit kommt die von einem "x-beliebigen" Knoten initiierte Nachrichtenkette im Mittel?

- wann trifft man auf einen stärkeren als man selbst?
- wie lange überlebt man im Mittel?

In einem Teich schwimmen n Fische verschiedener Grösse - wieviele Fische muss man im Mittel noch fangen, bis man einen fängt, der grösser als der erste gefangene Fisch ist? (Oder bis der Teich leer ist)

Lösungsansatz (aber darf man wirklich so vorgehen?):

Fängt man den grössten zuerst → n Fänge ("Pech", Teich wird leer)
Fängt man den zweitgrössten zuerst → n/2 Fänge im Mittel
Fängt man den drittgrössten zuerst → n/3 ...
... → n/i ...

Zurückführung auf ein bekanntes math. Modell: *Urnenmodell ohne Zurücklegen*

Z.B. n=100, 7.grösster Fisch initial:

→ 6 schwarze und 93 weisse Kugeln

→ Sei p die Wahrscheinlichkeit, eine schwarze Kugel zu ziehen ("Treffer")

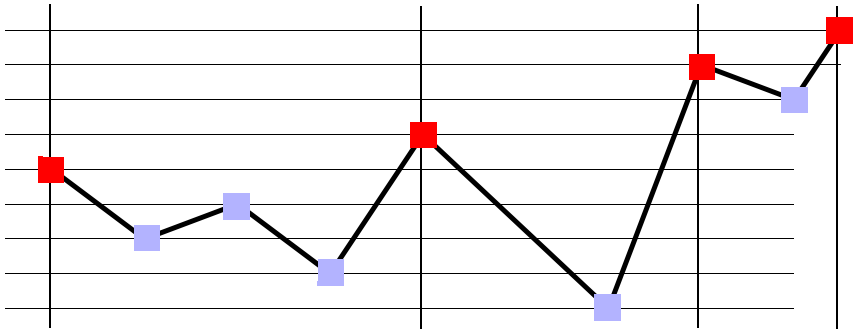
→ Beispiel: Wahrscheinlichkeit, eine weisse und dann eine schwarze zu ziehen (d.h. genau die Entfernung 2 zu schaffen): (1-p) mal Wahrscheinlichkeit für "Treffer bei 6 schwarzen und 92 weissen"

→ ... alles gewichtet aufsummieren... (aufwendig!)

Warten auf einen neuen Rekord...

- Rekord = grösserer Wert als alle vorangehenden
- "left to right maxima" (einer Zahlenfolge)

- Z.B. "heissester August seit Anfang des Jahrhunderts"



Das interessiert uns! (Mittelwert, Erwartungswert)

- Anwendung: z.B. Verlobungshäufigkeit bei deletion-sort

Festhalten und weitersuchen

Rekorde werden immer seltener!

- Wieviele Rekorde gibt es eigentlich?
  - z.B. heissester August in einem Jahrhundert...
  - Annahme: keine Korrelation (d.h. zufällige Permutation vorausgesetzt)

- Betrachte Bitfolge: i-te Stelle $\Leftrightarrow$ i-ter Wert ein Rekord

  Bsp: 1001011000000100000000001000000000000000010000...

  $\rightarrow$ Rekorde traten im 1., 4., 6., 7., 14.,... Jahr auf

  (Angeblich ist die Summe der Positionen der Rekorde, also im Beispiel 1+4+6+7+14+..., im Mittel n: ist das plausibel?)

- Wieviele 1en bei gegebener Länge n?

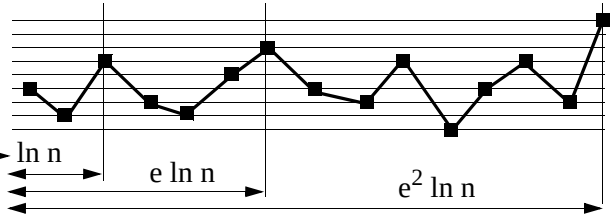
  Antwort:
  - damit i-tes Bit auf 1 steht, muss der i-te Zahlenwert der Folge $\geq$ alle ersten i Zahlenwerte sein
  - Wahrscheinlichkeit dafür ist $1/i$ (stimmt's? wieso?)

$\rightarrow$ Mittlere Anzahl von 1en = $1 + 1/2 + 1/3 + ...$
- darf man wirklich einfach so aufaddieren?

$\rightarrow$ Mittlere Anzahl von Rekorden ist $H_n \approx \ln n$

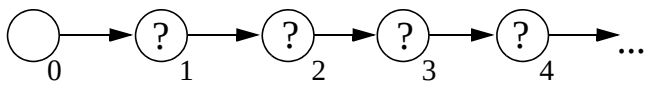
- Bsp: es gibt ca. 5 heisseste Auguste im Jahrhundert ...und wieviele in einem Jahrtausend?

$\Rightarrow$ Für einen Rekord mehr (im Mittel) muss das Zeitintervall ca. e = 2.71828... mal länger sein



Das haben wir noch immer nicht wirklich bewiesen!

Verteilung der Lebensdauer



Zu jedem Zeitpunkt ein Schritt...

- Wahrscheinlichkeit, *genau* bis zur Position i zu gelangen (und dort besiegt zu werden)?

- p(Lebenszeit = i) = p(Lebenszeit $\geq$ i) - p(Lebenszeit > i)
  - Vgl.: 18% Studis $\geq$ 16. Semester; 13% Studis $\geq$ 17. Semester ($>$ 16. Semester)
    $\Rightarrow$ 5% *im* 16. Semester

Wahrscheinlichkeit für "grösster unter den ersten i"

- Also: $p(\text{Lebenszeit} = i) = \frac{1}{i} - \frac{1}{i+1} = \frac{1}{i(i+1)}$ (für i<n)

- Aber: $p(\text{Lebenszeit} = n) = \frac{1}{n}$ ("erfolgreicher Durchmarsch")

Beispiel n=4:

gewichtete Summe = 25/12 $\rightarrow$ 25/3 = 8.333...
Nachrichten im Mittel (vgl. früheres Ergebnis!)

1	$\frac{1}{1\times2} = \frac{1}{2}$	= 50%
2	$\frac{1}{2\times3} = \frac{1}{6}$	= 16.7%
3	$\frac{1}{3\times4} = \frac{1}{12}$	= 8.3%
4	$\frac{1}{4}$	= 25%

100%

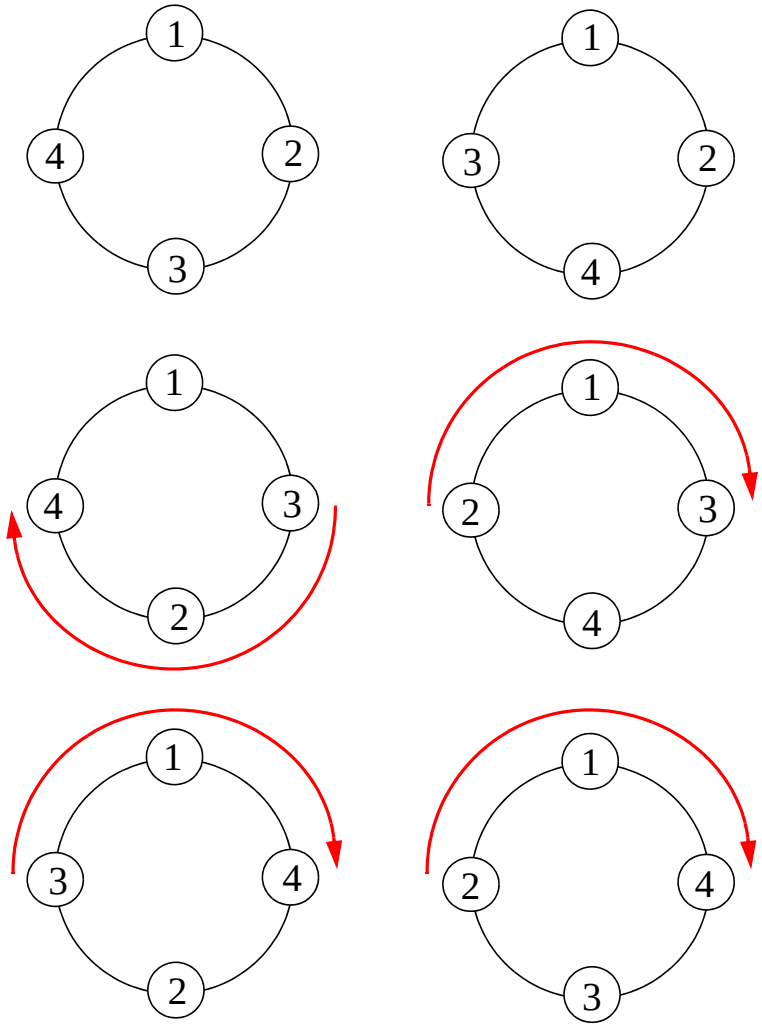
vgl. dazu nächstes Bild

Bem.: Es gilt

$$\frac{1}{n} + \sum_{i<n} \frac{1}{i(i+1)} = 1$$

(Induktionsbeweis als einfache Übung)

Beispiel: Lebensdauer 2 bei n = 4

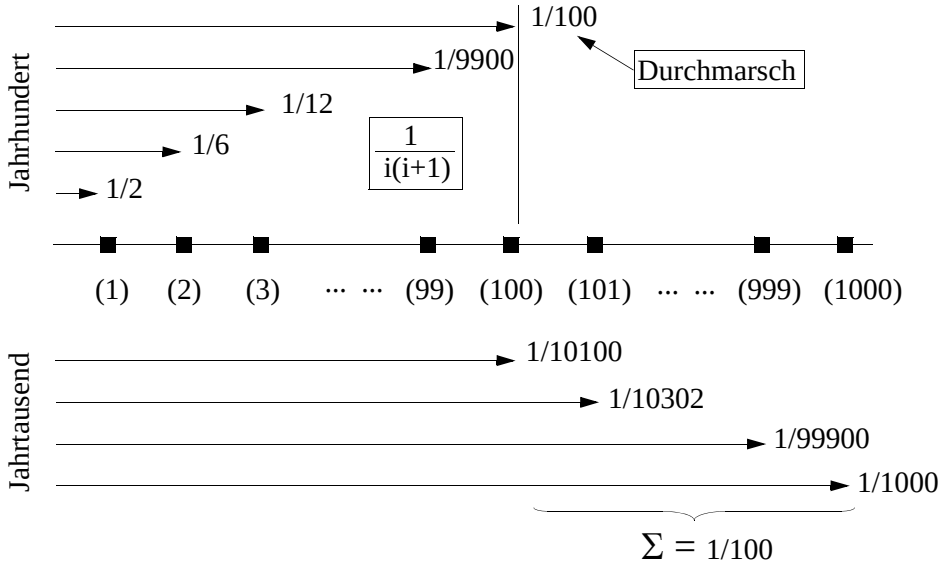


6 Bilder — jeweils 4 Starter

In 4 von 6 × 4 = 24 Situationen, also 1/6 aller Fälle (= 16.7%),
wird für einen Initiator die Lebensdauer 2 exakt erreicht

Rekord im Jahr i eines Jahrhunderts?

- Bzw.: Wahrscheinlichkeit, bei Position (i) "unterzugehen":



- "Durchmarsch" ist viel wahrscheinlicher, als an der Position davor unterzugehen

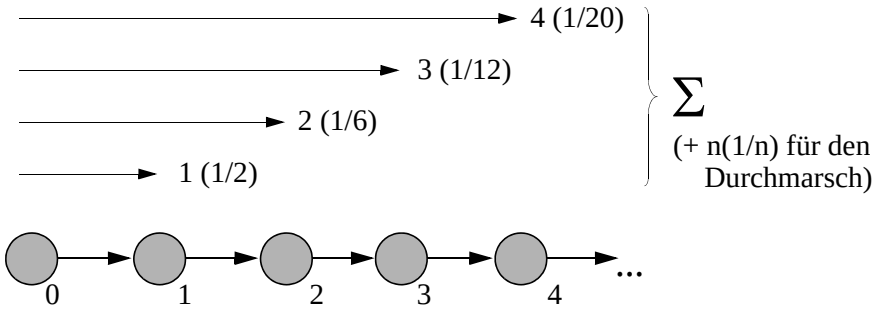
- Nur 50% Chance, über den ersten hinwegzukommen

- p(*kein* echter Rekord in einem Jahrhundert) = 1/100,
- p(... in einem Jahrtausend) = 1/1000

Rekord würde jenseits liegen

Nun also: Wartezeit bis zum ersten Rekord

- Gesucht: Die *mittlere* Lebensdauer
- Dazu: Wahrscheinlichkeit mit Weglänge gewichten und alle (disjunkten) Einzelfälle aufsummieren



- Daraus folgt allgemein für den Erwartungswert:

$$n\,\frac{1}{n} + 1\,\frac{1}{1\times 2} + 2\,\frac{1}{2\times 3} + 3\,\frac{1}{3\times 4} + \ldots + (n-1)\,\frac{1}{(n-1)n} = \underline{\underline{H_n}}$$

Also: Eine von einem "x-beliebigen" Prozess gestartete Nachricht induziert im Mittel einen Weg der Länge H_n

- darf man das nun mit n multiplizieren, wenn jeder Prozess startet?
- ja; aber generell Vorsicht bei solchen Überlegungen!

Damit haben wir also $nH_n \approx \ln n$ als mittlere Nachrichtenkomplexität des Chang/Roberts-Algorithmus!

Qualität des Chang/Roberts-Algorithmus

- *Satz* (ohne Bew.):

Für unidirektionale Ringe der Grösse n und n Initiatoren ist $n\,H_n$ die optimale mittlere Nachrichtenkomplexität beim *Election-Problem*

- mit anderen Worten: es gibt keinen anderen Algorithmus, der "besser" ist!

→ Chang/Roberts-Algorithmus ist "average case optimal"!

- *Satz* (Rotem et al., ohne Bew.):

Für die Nachrichtenzahl m des Chang/Roberts-Algorithmus gilt für alle ε: $\lim_{n \to \infty}$ prob $(m < (1+\varepsilon)\, n\, H_n) = 1$

Interpretation: Bei grösseren Ringen werden fast nie mehr als $n\,H_n$ Nachrichten benötigt

Tschebyscheff- / Chernoff-Ungleichungen

- Die Nachrichtenzahl des Chang/Roberts-Algorithmus lässt sich auch genauer abschätzen:

  - aus der *Tschebyscheff-Ungleichung* (engl.: "Chebyshev") folgt:

$$\text{prob}\,(m \geq (1+\varepsilon)\, n\, H_n) \leq \frac{\pi^2/6}{\varepsilon^2\, H_n^{\,2}}$$

(vgl. dazu Lehrbücher zur Wahrscheinlichkeitstheorie)

  - noch besser lässt sich dies mit der *Chernoff-Ungleichung* abschätzen:

$$\text{prob}\,(m \geq (1+\varepsilon)\, n\, H_n) \leq \exp(-\varepsilon^2\, n H_n/3)$$

- Einige Beispiele (Wahrscheinlichkeit, dass die Nachrichtenzahl unter dem angegebenen Wert liegt):

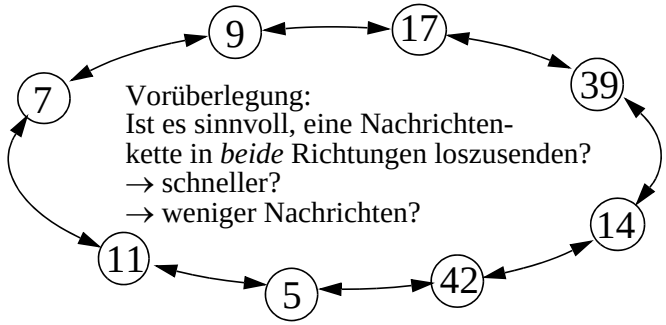
	Tschebyscheff	*Chernoff*
ε=1, n=10	0.31	0.00046
ε=1, n=100	0.078	10^{-67}
ε=0.5, n=100	0.31	10^{-17}
ε=0.1, n=100	--	0.2

  - die Tschebyscheff-Ungleichung ist also recht konservativ!
  - auch die Chernoff-Ungleichung ist nur eine Abschätzung; *die Nachrichtenzahl liegt also fast immer sehr nahe am Mittelwert!*

Zur Chernoff-Ungleichung vergleiche man z.B.:

- C. Lavault: Evaluation des algorithmes distribués. Hermes, Paris 1995
- T. Hagerup. C. Rüb: A Guided Tour of Chernoff Bounds. Inform. Proc. Lett. 33, 305-308, 1990

Probabilistisches Election-Verfahren für bidirektionale Ringe



- Message-Extinction-Prinzip in naheliegender Weise verallgemeinern:

I_p: {M = 0}
M := p;
Wähle mit Wahrscheinlichkeit 1/2 eine Richtung;
send <M> **to** Nachbar in diese Richtung;

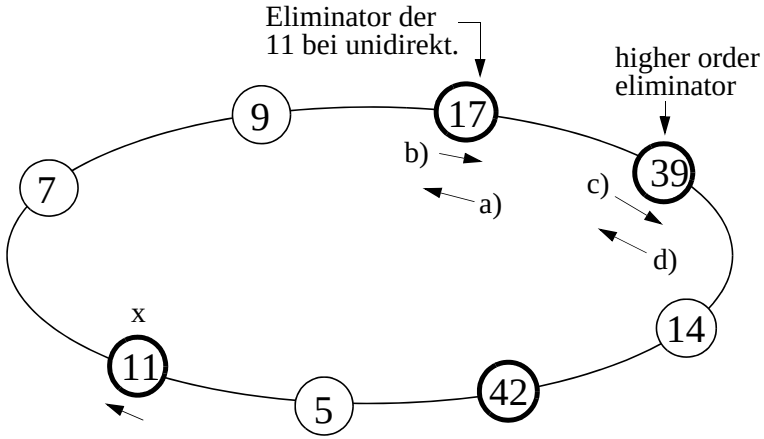
Identität des Prozesses

Uhrzeigersinn oder Gegen-Uhrzeigersinn

R_p: {Eine Nachricht <j> ist eingetroffen}
if M < j **then**
  M := j;
  send <M> **to** ... // an anderen Nachbarn
fi // *weiter*senden
if j = p **then** *"I am the leader"* **fi**

- ist Wahrscheinlichkeit 1/2 eine gute Wahl?
- geht es nicht besser "irgendwie" deterministisch statt probabilistisch?
- wie hoch ist die mittlere Nachrichtenkomplexität?

Mittlere Nachrichtenkomplexität



- Annahme: Jede Einzelnachricht braucht gleich lang

- In der Hälfte aller Fälle kommt der Eliminator der Nachricht "x" auf halbem Weg entgegen
  - z.B. Fall a) bei den Knoten 11 und 17
  - → sollte 1/4 aller Nachrichten (gegenüber unidirekt. Fall) sparen (wieso?)

- Aber: höchste läuft immer ganz durch
  (jedoch: spielt für $n \to \infty$ eine "asymptotisch geringe" Rolle)

- Asymptotische mittlere Nachrichtenkomplexität ist *geringer* als 0.75 n ln n (Grund: "Higher order eliminators")
  - z.B. 39 verkürzt den Weg der 11 "etwas" im Fall b), d)
  - 42 als higher order eliminator würde hier aber nichts nützen!

Resultat von Lavault (Beweis schwierig!):
Asymptot. Nachrichtenkomplexität $n \to \infty$ ist

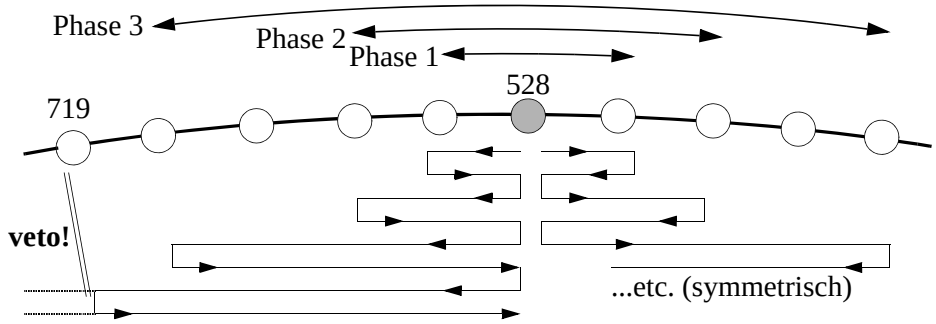
$$0.5\ \sqrt{2}\ n \ln n \approx 0.7071\ n \ln n$$

Als *untere Schranke* für das bidir. Election-Problem kennt man 0.5 n ln n (→ "Lücke")

Hirschberg / Sinclair-Election-Algorithmus

- Idee: Jeder Knoten versucht, sukzessive Gebiete der Grösse 2^i (i=1,...) zu erobern

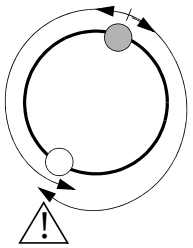
bidirektionaler Ring!



- Ein unterwegs angetroffener grösserer Knoten legt Veto ein
  - → Initiator über Rückmeldung informieren
  - → Initiator wechselt von *aktiv* nach *passiv*

nur noch Nachrichten weiterleiten ("relay")

Gewinnsituation:



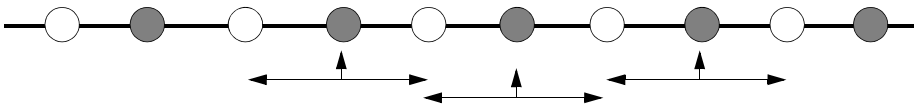
- Nachricht läuft in bereits selbst erobertes Gebiet
- *oder*: Nachricht trifft bei Initiator selbst wieder ein
- es bleibt genau ein Gewinner! (wieso?)

Komplexitätsanalyse

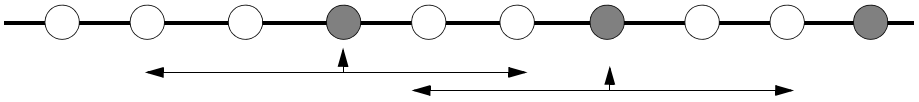
- Ein Prozess kann nur dann eine Kette der Länge 2^i starten, wenn er im Abstand 2^{i-1} in beiden Richtungen überlebt hat
  - Dichte überlebender Prozesse nimmt also exponentiell ab
- Innerhalb eines Bereiches von $1+ 2^{i-1}$ benachbarter Prozesse kann also *höchstens einer* eine Kette der Länge 2^i starten

---

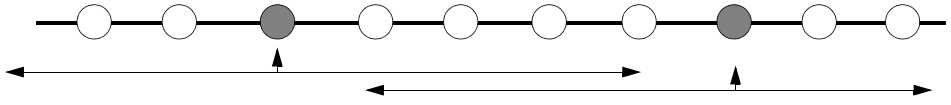
- Jeweils mind. 1 dazwischenliegender Prozess nach Phase 1:



- Jeweils mind. 2 dazwischenliegende Prozesse nach Phase 2:



- Jeweils mind. 4 dazwischenliegende Prozesse nach Phase 3:



- max. n/2 Prozesse können Ketten der Länge 2 initiieren
- max. n/3 Prozesse können Ketten der Länge 4 initiieren
- max. n/5 Prozesse können Ketten der Länge 8 initiieren

  ...
- max. $n/(1+2^{i-1})$ Prozesse können Ketten der Länge 2^i initiieren

Maximal 8 n $\log_2$ n Nachrichten

- Also: höchstens $n/(1+2^{i-1})$ Prozesse initiieren eine Nachrichtenkette der Länge 2^i in Phase i
- Bei jeder solchen Kette wird jede Kante max. 4 Mal durchlaufen
- In Phase i gibt es also höchstens $4 \times 2^i \times n / (1+2^{i-1}) < 8n$ Nachrichten
- Es gibt höchstens $1+\lceil \log_2 n \rceil$ Phasen

$\Rightarrow$ ca. 8 n $\log_2$ n $\approx$ 11.53 n ln n Nachrichten *maximal*

(Worst-case-Komplexität!)

- *Mittlere* Nachrichtenkomplexität schwieriger abzuschätzen
- *Zeitkomplexität*: $2 + 4 + 8 + 16 +... + 2^i < 2^{i+1} \approx 4n$

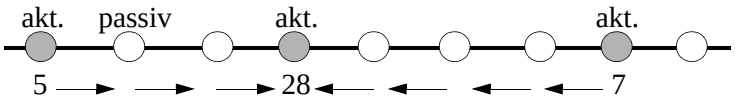
Vergleiche dies alles mit dem Chang/Roberts-Algorithmus

Peterson's Election-Algorithmus (1. Variante)

- Prinzip: Anzahl aktiver Knoten pro "Phase" mind. halbieren
  - bidirektionaler Ring
  - anfangs sind alle *aktiv*
  - *passive* Knoten reichen nur noch Nachrichten weiter ("relay")

- Idee: Pro Phase bekommt ein Knoten die Identitäten seiner rechten und linken *noch aktiven* Nachbarn...

Vgl. dies mit iterierter Anwendung des Algorithmus für Nachbarschaftswissen!



...und überlebt nur, wenn er der grösste *aller drei* ist!

- Ein Überlebender bleibt aktiv und startet eine neue Phase: Sendet seine Identität in *beide* Richtungen
  (Initial tun das alle Initiatoren)

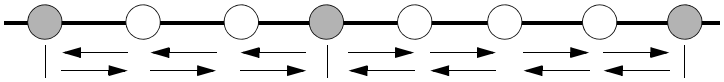
- *Gewonnen*, wenn die eigene Identität empfangen wird

Beachte: In obigem Beispiel wird die 5 von der 28 "passiviert". Bald darauf (in der nächsten Phase) erhält die 5 erneut eine Nachricht "28", um diese weiterzuleiten. Hätte die 5 nicht gleich beim ersten Mal die "28" einfach weiterleiten sollen, so dass Knoten 28 die Nachricht nicht erneut über die Strecke 28 → 5 senden muss? (Vgl. Chang/Roberts!) Nein! Knoten 5 weiss nicht, ob die 28 die gegenwärtige Phase tatsächlich überlebt - wenn nicht, dann wäre die Nachricht "28" fälschlicherweise weitergeleitet worden!

Zeitkomplexität des Algorithmus als Denkübung (dominiert hier die letzte Phase?)

Nachrichtenkomplexität

- Pro Phase laufen 2 Nachrichten über *jede* Kante
  - für global *synchrone Phasen* leicht einsichtig
  - aber auch für nicht synchronisierte Phasen richtig!



- Wenn ein Knoten Phase i überlebt, überlebt sein "linker" aktiver Nachbar diese Phase nicht!

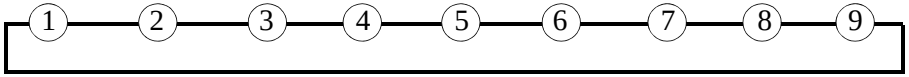
  → max. $\log_2 n$ Phasen

  → max. $2 n \log_2 n$ Nachrichten

"in jeder Phase überlebt immer nur einer von 3" ist *falsch* - wieso?

- wie sieht eine Anordnung aus, bei der *maximal viele* Nachrichten entstehen?

- *Sortierte Anordnung*: jeweils durch "rechten" Nachbarn eliminiert, ausgenommen grösster Knoten im Ring



→ dann nur 2 Phasen ⇒ 4n Nachrichten! (beachte Terminierungserkennung!)

- Mittlere Nachrichtenkomplexität:
  - ein Knoten überlebt eine Phase mit Wahrscheinlichkeit 1/3 (wirklich? wieso?)
  - es gibt also im Mittel $\log_3 n$ Phasen

  → $2 n \log_3 n \approx 1.26 n \log_2 n \approx 1.82 n \ln n$ Nachrichten

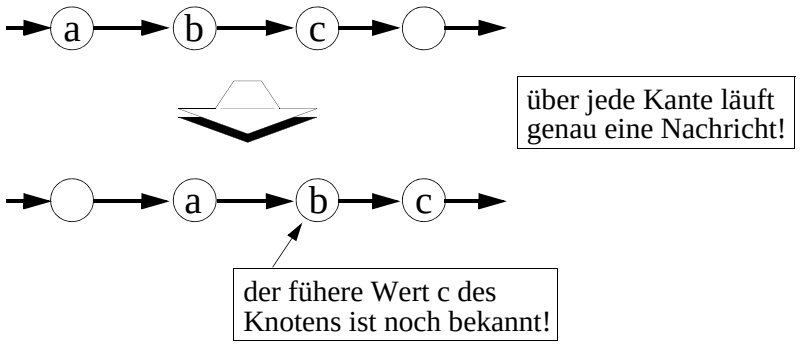
Variante für unidirektionale Ringe

- Man glaubte zunächst, dass jeder Election-Algorithmus auf *unidirektionalen* Ringen mindestens $O(n^2)$ Nachrichten im Worst-case-Fall benötigt

- Das stimmt nicht: Der Peterson-Algorithmus lässt sich auf unidirektionalen Ringen *simulieren*!

---

1. "Shift" in Ringrichtung um eine Position bzgl. aktiver Knoten:



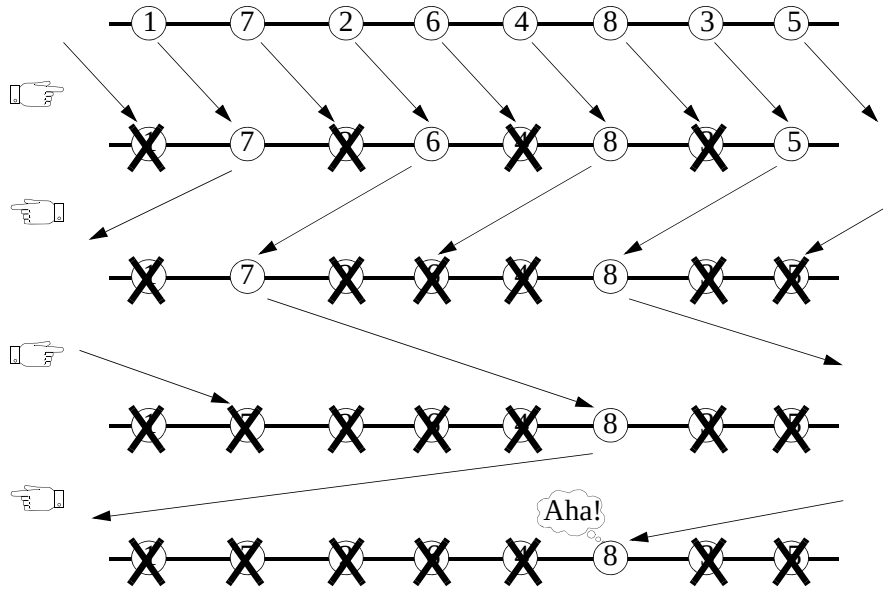
2. Nun kann (der neue) Knoten a an seinen Nachbarn b seinen Wert senden - damit kennt b sowohl a als auch c
(als hätte b Nachrichten von a und c erhalten!)

Damit kostet eine solche Phase insges. auch nur 2n Nachrichten!

Peterson's Election-Algorithmus (2. Variante)

- Idee einer Optimierung:
  - anstatt sich mit beiden Nachbarn "gleichzeitig" zu vergleichen, sollte ein Knoten sich nur dann mit seinem anderen Nachbarn vergleichen, wenn er den ersten Vergleich gewonnen hat

- Phasen im / gegen den Uhrzeigersinn wechseln sich ab:



  - lässt sich auch wieder unidirektional simulieren!
  - in jeder Phase werden n Nachrichten gesendet (passive Knoten: "relay")

- Denkübung:
  Man mache sich Gedanken zur Abschätzung der worst-case und der average-case Nachrichtenkomplexität

---

Einen ähnlichen, aber etwas effizienteren Algorithmus gaben Lisa Higham und Teresa Przytycka 1996 an: A Simple, Efficient Algorithm for Maximum Finding on Rings. Inf. Process. Lett. 58(6): 319-324

Nachrichtenkomplexität

- **Behauptung:**
  Für die *Anzahl der Phasen* r gilt: $r \leq \log_\varphi n + O(1)$ (Basis $\varphi = (1+\sqrt{5})/2$)
  $\Rightarrow$ Anzahl der Nachrichten $\leq 1.44\, n \log_2 n + c$

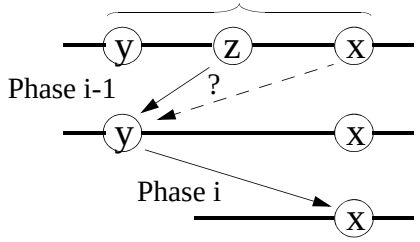
- **Lemma:** $a_i \leq a_{i-2} - a_{i-1}$ (für $i > 1$)

Def: Anzahl Überlebende von Phase i

Anzahl der "Opfer" von Phase i-1

Bew.: Betrachte 2 benachbarte Knoten x, y zu Beginn von Phase i

Gab es einen Knoten zwischen x und y in Phase i-1?



(1) x überlebe Phase i
$\Rightarrow x > y$

(2) y hat Phase i-1 überlebt
$\Rightarrow y > z$

(1) und (2) $\Rightarrow x \neq z$

Also muss es ein z geben, das in Phase i-1 Opfer wurde
Hier: x hat seinen linken Nachbarn in der vorherigen Phase verloren

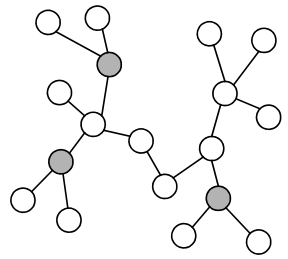
$\Rightarrow$ Für jeden Überlebenden in Phase i (hier: x) gibt es mindestens ein Opfer (hier: z) in Phase i-1 □
(Also: Pro "Doppelphase" mindestens Halbierung der Knotenzahl!)

- Aus $a_i \leq a_{i-2} - a_{i-1}$ folgt $a_{i-2} \geq a_{i-1} + a_i$, also $a_i \geq a_{i+1} + a_{i+2}$
- Ferner gilt $a_{r-1} = 1 = Fib(2)$, $a_{r-2} \geq 2 = Fib(3)$ } $a_{r-3} \geq a_{r-2} + a_{r-1} \geq Fib(2)+Fib(3) = Fib(4)$
- Also: $n = a_0 \geq Fib(r+1)$ → Ungleichung nach r auflösen!

Weil Fib exponentiell zur Basis φ wächst ($Fib(k) \approx \varphi^k / \sqrt{5}$), folgt die Behauptung

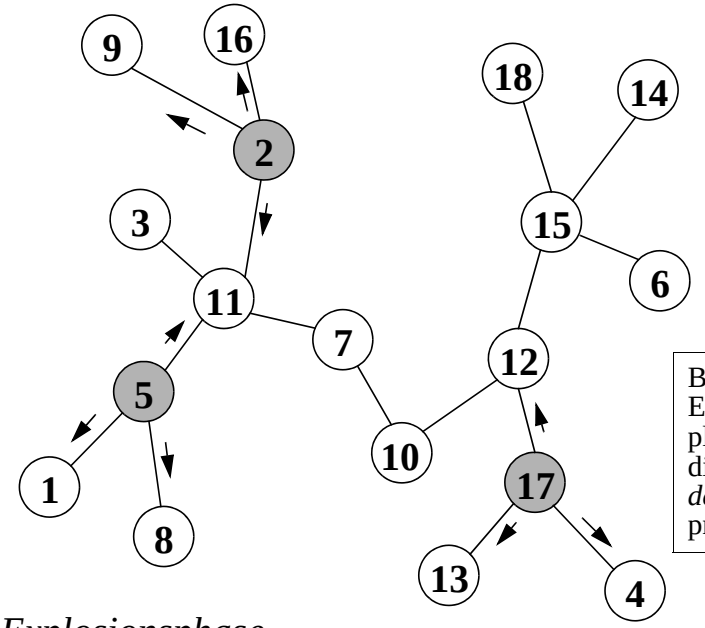
Election auf Bäumen

- Geht dies besser / effizienter als z.B. mit dem Message-Extinction-Prinzip für allg. Graphen?
- Und im Vergleich zu den Verfahren auf Ringen?



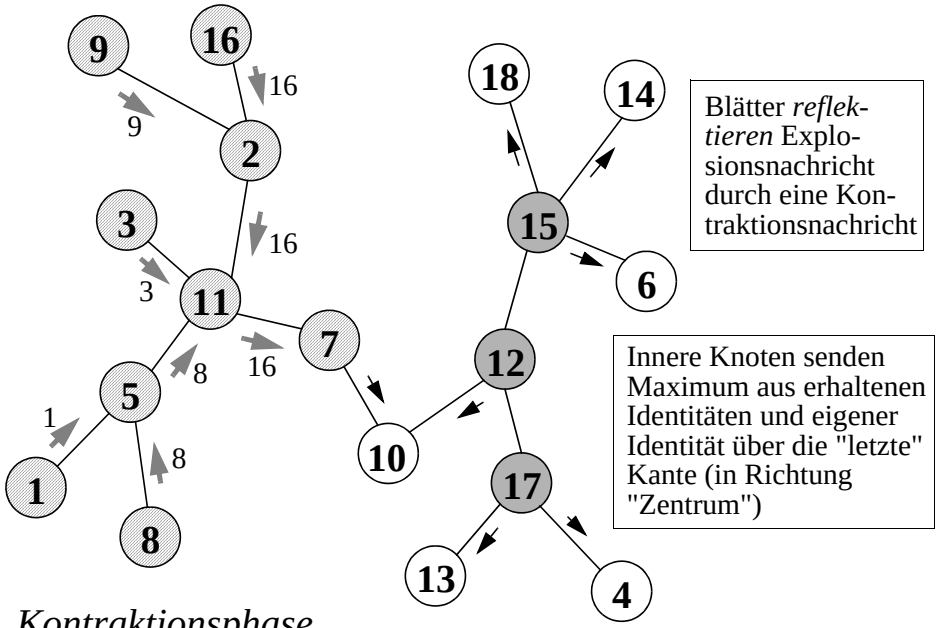
- Explosionsphase:
  - Election-Ankündigung wird zu den Blättern propagiert (flooding!)
- Kontraktionsphase
  - von aussen zum "Zentrum" das Maximum propagieren (0 für unbeteiligte Knoten)
- Informationsphase (optional)
  - Zentrum informiert alle Knoten über Gewinner (flooding!)

- Explosionsphase kann an mehreren Stellen "zünden"
- "Vereinigung" der Explosionsphasen
- Evtl. sind Teile noch in Explosionsphase, während andere Teile schon in der Kontraktionsphase sind
- "Zentrum" ist nicht determiniert



Bei erstmaligem Erhalt einer Explosionsnachricht diese in alle *anderen* Richtungen propagieren

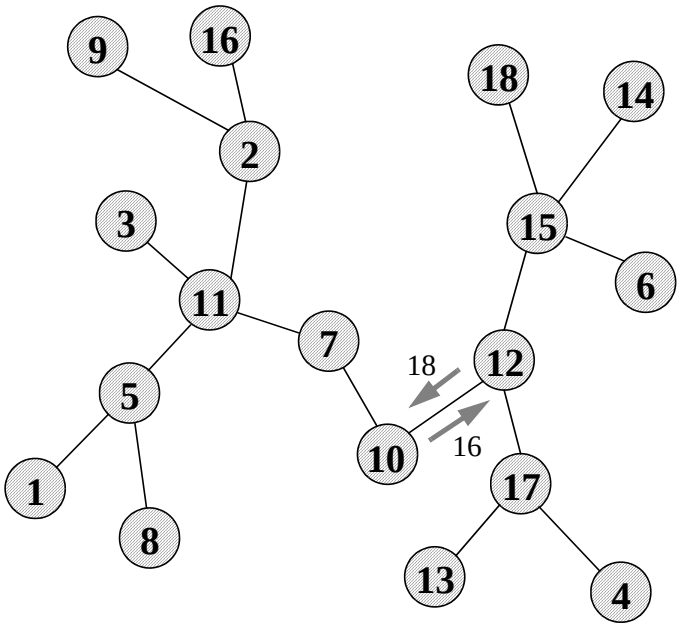
Explosionsphase



Blätter *reflektieren* Explosionsnachricht durch eine Kontraktionsnachricht

Innere Knoten senden Maximum aus erhaltenen Identitäten und eigener Identität über die "letzte" Kante (in Richtung "Zentrum")

Kontraktionsphase



- Begegnung zweier Kontraktionsnachrichten auf (genau) einer Kante im "Zentrum"
- Die beiden Knoten dieser Zentrumskante wissen, dass sie nun das Maximum kennen
  - sie können dies nun evtl. per flooding verbreiten (Informationsphase)
- Terminierung der Informationsphase (falls notwendig) durch erneute Reflexion / Kontraktion ("indirektes acknowledge")

Nachrichtenkomplexität von Baumelection

Folgender Satz / Beweis ist *falsch* - wieso?

Beh.: Der Baumelection-Algorithmus hat bei *einem* Initiator und n Knoten die Komplexität m(n) = 2n - 2
(ohne Berücksichtigung der Informationsphase)

Beweis induktiv:

1) n=1 → m(1) = 0 ✓ (offensichtlich korrekt)

2) Schritt von n auf n+1:

- Füge an einen Baum aus n Knoten ein Blatt an; über die neue Kante fliessen genau 2 Nachrichten
- Also: m(n+1) = m(n)+2 = 2n-2+2 = 2(n+1)-2 ✓

- wo genau liegt der Fehler?
- korrekter Wert der Nachrichtenkomplexität → nächste Seite!

Nachrichtenkomplexität von Baumelection

(1) Explosionsphase: n-2+k

Anzahl der Initiatoren

- es gibt k-1 Begegnungskanten von Explosionsnachrichten

(2) Kontraktionsphase: n

- über alle Kanten eine Nachricht, nur über die Zentrumskante zwei

(3) Optionale Informationsphase: n-2

- keine Nachricht über die Zentrumskante

$\sum = 3n + k - 4$ (mit Information aller Knoten)

→ 3(n-1) für k=1

→ 4(n-1) für k=n

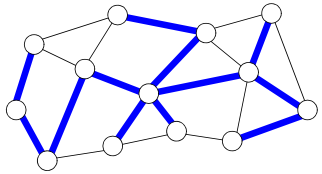
- Wesentlich effizienter als Ringe!
  - Grund: Ringe sind symmetrischer
- Wieso Verfahren nicht "einfach" auf Ringe anwenden?

(eine Kante entfernen)

Election auf allgemeinen Graphen

zusammenhängend mit bidirekt. Kanten

- Wieso versagt folgende einfache Idee für allg. Graphen?
  - verwende den Echo-Algorithmus, um einen (einzigen) Spannbaum zu konstruieren
  - führe dann Election auf diesem Baum aus



- Wie wäre es damit:
  - jeder Initiator startet seinen eigenen Echo-Algorithmus, mit dem er (über die Echo-Nachrichten) die grösste Identität erfährt
  - jeder Initiator weiss somit, ob er der grösste ist oder nicht (und kennt auch die Identität des grössten)
  - vgl. dies mit dem "Bully-Algorithmus" für Ringe: jeder macht einen vollständigen (!) Ringdurchlauf und prüft dabei, ob er der grösste ist
  - ist das korrekt?
  - effizient?

"Echo-Election" für allgemeine Graphen

zusammenhängend mit bidirekt. Kanten

- *Generelle Idee*: Wie Chang / Roberts-Algorithmus (also "message exctinction"), jedoch *Echo-Algorithmus* anstelle des zugrundeliegenden Ring-Verfahrens

- Also:
  - Jeder Initiator startet "seinen" eigenen Echo-Algorithmus
  - Explorer und Echos führen die Identität des Initiators mit
  - Schwächere Nachrichten (Explorer bzw. Echos) werden "verschluckt" (d.h. nicht weitergegeben)
  - Stärkste Welle setzt sich überall durch (und informiert so neben dem Gewinner auch alle Verlierer)
  - Alle anderen Wellen stagnieren irgendwo endgültig
    (zumindest der Gewinner sendet keine Echos für diese Wellen
    $\rightarrow$ kein anderer Initiator bekommt alle seine Echos)

- *Veranschaulichung*: "Gleichzeitiges" Einfärben des Graphen mit verschieden dominanten Farben

---

Denkübung: Man mache sich Gedanken zur Abschätzung der worst-case und der average-case Nachrichtenkomplexität

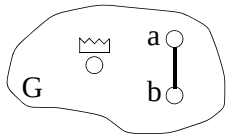
Nachrichtenkomplexität: untere Schranke

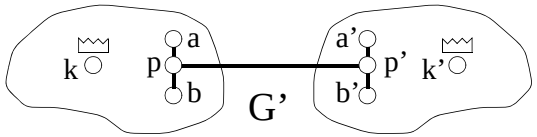
Satz: *Zur Lösung des Election-Problems in allgemeinen Graphen werden mindestens e Nachrichten benötigt*

Beweis durch Widerspruch:

Angenommen, es gäbe einen Election-Algorithmus A, der weniger als e Nachrichten für einen Graphen G (aus mindestens 2 Knoten) benötigt
⇒ es gibt eine Kante $\overline{ab}$, über die *keine Nachricht* fliesst

Konstruiere dann G' so: Zwei Kopien von G (mit unterschiedlichen Identitäten, aber der gleichen relativen Ordnung), die mit einer Kante $\overline{pp'}$ verbunden werden, wobei p bzw. p' in die unbenutzten Kanten $\overline{ab}$ bzw. $\overline{a'b'}$ neu eingefügt werden





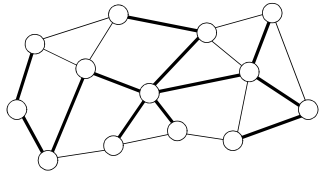
Da zwischen den beiden Teilen keine Nachricht ausgetauscht wird, gewinnen dann *zwei* Knoten k, k'!

Beachte bei diesem Beweis:

- die Knoten haben (weder in G noch in G') ein "globales Wissen": sie kennen nicht die Struktur oder Gesamtgrösse des Graphen, sie kennen auch nicht die Identitäten ihrer Nachbarn
- die Knoten wissen insbesondere nicht, ob Situation G oder G' vorliegt
- der Algorithmus sei (wie bisher auch immer angenommen) deterministisch
- bei Anwendung von Algorithmus A auf G' wird sich daher jeder einzelne Knoten *genauso verhalten* wie der entsprechende im Graphen G
- Knoten p und p' seien keine Initiatoren
- bzgl. der Knotenidentitäten wird nicht vorausgesetzt, dass es sich um Nummern handelt; Algorithmus A braucht daher auch nicht den "grössten" Knoten zu bestimmen, sondern nur irgend einen eindeutigen
- G bzw. G' haben (wie üblich) keine Schleifen oder Mehrfachkanten

Verteilte Spannbaumkonstruktion

- Gegeben: Zusammenhängendes Netz (mit bidir. Kanten)
- Alle Knotenidentitäten seien verschieden
- Bestimmung eines Spannbaums ist oft wichtig:



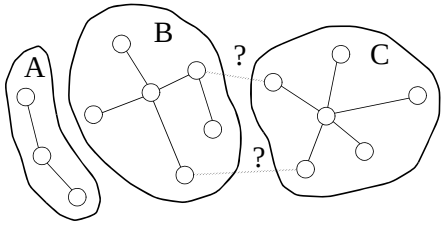
- z.B. um entlang dieses Baumes Information zu verteilen und einzusammeln
- Routing ist wesentlich einfacher, wenn Zyklen ausgeschlossen sind

---

Problem: - Wer initiiert die Spannbaumkonstruktion?
- Basteln alle Initiatoren am gleichen Baum?
(Problem: Zyklenbildung trotz Dezentralität vermeiden!)

- *Lösung 1*: Election, Gewinner startet Echo-Algorithmus
(bei Echo-Election bilden die Echos des Gewinners bereits einen Spannbaum)

- *Lösung 2*: Dezentral werden Fragmente gebildet, die wachsen und sich nach und nach vereinigen



- Initial: Jeder Knoten ist ein Fragment
- Algorithmus von Gallager, Humblet, Spira (1983) mit 2e + 5 n log n Nachrichten (im Detail nicht ganz einfach, hier nicht weiter behandelt)
- Problem: Zyklenbildung vermeiden und sparsam mit Nachrichten umgehen!

Election und Spannbaumkonstruktion

Beh.: Election und Spannbaumkonstruktion sind in allgemeinen Graphen vergleichbar schwierige Probleme

Präziser: Sei C_e die Nachrichtenkomplexität des Election-Problems, und C_t diejenige des Spannbaum-Problems:

(a) Es gilt für C_t: $C_t \leq C_e + 2e$ (wegen obiger "Lösung 1")

(b) Es gilt für C_e: $C_e \leq C_t + O(n)$ (wegen Komplexität Baumelection)

Interpretation:

(a) Hat man mittels Election einen "leader" bestimmt, lässt sich ein eindeutiger Spannbaum einfach ermitteln

(b) Hat man einen Spannbaum, dann lässt sich ein "leader" einfach (d.h. effizient) bestimmen

Hierbei wird 2e bzw. O(n) als "klein" gegenüber C_e und C_t angesehen

---

Aus der unteren Schranke Ω(e) für die Nachrichtenkomplexität des Election-Problems folgt aus (b):

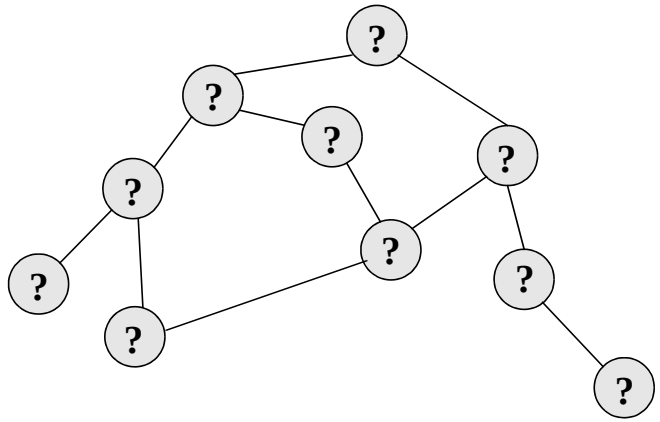
Das Spannbaum-Problem hat eine Nachrichtenkomplexität von mindestens Ω(e)

Denn sonst: Konstruiere den Spannbaum effizienter und löse mit zusätzlichen O(n) Nachrichten das Election-Problem!

Anonyme Netze

- Keine Knotenidentitäten
  - bzw. alle (oder mehrere) *identische* Identitäten
  - bzw. evtl. vorhandene Knotenidentitäten werden nicht benutzt

- Frage: Was geht dann noch? (Insbes. Symmetriebrechung!?)



- Es gilt: Falls Election in anonymen Netzen geht, dann können die Knoten individuelle Namen bekommen

→ "De-Anonymisierung"

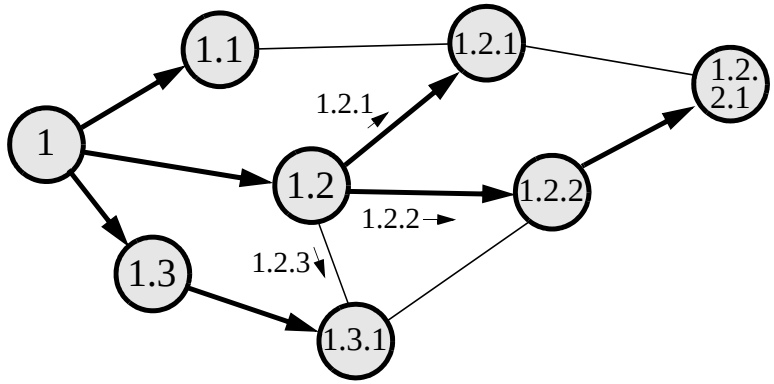
...und *alles* geht wie gehabt!

(Der Satz lässt vermuten, dass Election in anonymen Netzen *nicht* geht, denn sonst wäre Anonymität kein *grundsätzliches* Problem!)

De-Anonymisierung mit Leader

Der *Leader* flutet das Netz; sendet verschieden benannten Nachrichten an seine Nachbarn. Jeder Knoten übernimmt die Identität der ersten Nachricht, die ihn erreicht - dazu numeriert ein Knoten seine Kanäle bzw. Nachrichten durch und konkateniert seine eigene Identität zu dieser Nummer.

$\Rightarrow$ Alle Nachrichten und damit alle Knoten heissen verschieden!



- Der Leader als Initiator gibt sich selbst den Namen “1”
- Jeder Prozess numeriert seine Ausgangskanäle 1,...,k
- Jede von Prozess X über Kanal j versendete Nachricht bekommt zusätzlich den Namen “X.j” dazugepackt
- Ein (noch) anonymer Prozess nimmt den Namen der ersten Nachricht als seinen Namen

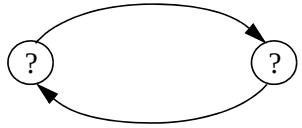
Fragen: - wann ist das Verfahren beendet?
- wie erfährt der Initiator dies?

Election in anonymen Netzen?

- In anonymen Netzen geht manches (z.B. Election!) nicht mehr mit deterministischen Algorithmen
  $\rightarrow$ randomisierte Verfahren helfen gelegentlich!
- Manches geht noch, wenn wenigsten die Knotenzahl bekannt ist (aber auch das hilft nicht immer!)

Satz: *Es gibt keinen stets terminierenden Election-Algorithmus für anonyme Ringe*

selbst wenn die Ringgrösse den Knoten bekannt ist



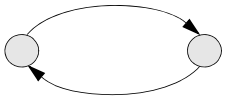
- hier für Ringgrösse 2
- jeder Knoten befindet sich (bzgl. des Election-Algorithmus) in einem bestimmten Zustand z

Bew.: - Betrachte *Konfigurationen* (hier Quadrupel aus den Zuständen der beiden Knoten und Kanäle)
  - Kanalzustand = Nachrichten, die dort unterwegs sind
- Anfangkonfiguration des Algorithmus ist *symmetrisch*: (z,z,$\varnothing$,$\varnothing$), wegen Anonymität
- Alle lokalen Algorithmen sind definitionsgemäss identisch $\rightarrow$ Knoten *können* sich jeweils gleich (quasi "zum gleichen Zeitpunkt") verhalten
- Konfigurationsfolge kann daher aus lauter symmetrischen Konfigurationen bestehen: (z,z,$\varnothing$,$\varnothing$) $\rightarrow$ (z’,z’,k,k) $\rightarrow$ ... $\rightarrow$ etc.
- Falls die Folge endlich ist, könnte der Endzustand symmetrisch sein $\rightarrow$ keine Symmetriebrechung möglich!

Probabilistische Election für anonyme Ringe der Grösse 2

```
state := undecided;
while state = undecided do
{ mine := random(0,1);
  send <mine> to neighbor;
  receive <his>;
  if (mine,his) = (1,0) then state := win;
  if (mine,his) = (0,1) then state := lose;
}
```

zufälliges Ergebnis 0 oder 1

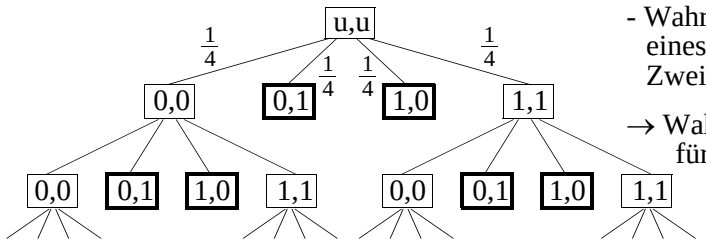


- jeder Knoten führt den gleichen nebenstehenden Algorithmus aus
- Verallgemeinerung auf grössere Ringe?

- Geht das nicht auch in *deterministischer* Weise?
- Beachte: *"korrekt" ≠ "korrekt mit Wahrscheinlichkeit 1"*!
  - "korrekt" heisst: Algorithmus *hält stets* (!) und liefert richtiges Ergebnis
  - obiger Algorithmus ist aber (in diesem Sinne) *nicht korrekt*, da es nicht terminierende Konfigurationsfolgen wie etwa
    - (0,0) → (0,0) → (0,0) → (0,0) → (0,0) → (0,0) ... oder
    - (0,0) → (1,1) → (0,0) → (1,1) → (0,0) → (1,1)... etc. gibt!
  - alle diese Folgen haben aber die Dichte 0 (wenn "random" gut genug ist...)

⇒ Algorithmus terminiert mit Wahrscheinlichkeit 1

⇒ Algorithmus ist "korrekt mit Wahrscheinlichkeit 1"!

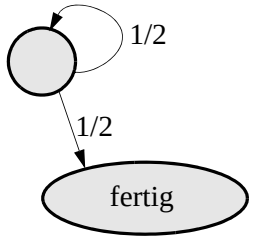


- Wahrscheinlichkeit eines unendlichen Zweiges = 0

→ Wahrscheinlichkeit für leader = 1

Mittlere Nachrichtenkomplexität?

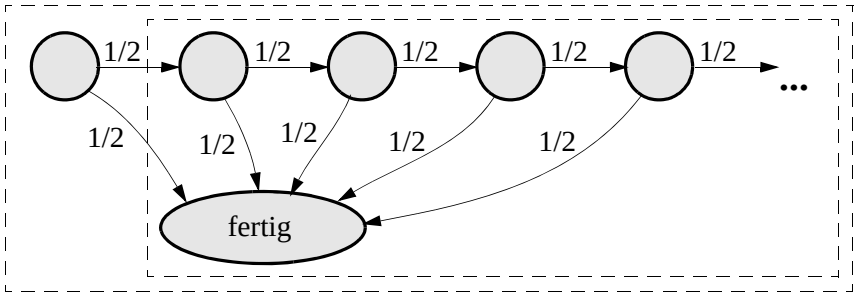
- Beachte: Keine Schranke für *worst-case* Komplexität!
- Die mittlere ("erwartete") Anzahl der Schleifendurchläufe ("Rundenzahl") lässt sich mittels *Markow-Ketten* ermitteln:



- wie hoch ist die erwartete Weglänge W, wenn mit den angegebenen Wahrscheinlichkeiten (hier jeweils 1/2) zum Folgezustand verzweigt wird?
- ist dieser Ansatz (gewichtete Summe) korrekt?

$$1\,\frac{1}{2} + 2\,\frac{1}{4} + 3\,\frac{1}{8} + 4\,\frac{1}{16} + \ldots + \quad = ?$$

- Durch "Aufbröseln" der Schleife ergibt sich diese Darstellung:



- Hier ist man mit Wahrscheinlichkeit 1/2 nach einem Schritt fertig, oder es liegt (nach einem Schritt) wieder die gleiche Situation vor!
- Daraus ergibt sich der *Rekursionsansatz* W = 1/2 + 1/2 (1+W)
- Dies liefert W=2 als Lösung
- Somit ist die mittlere Nachrichtenkomplexität 4

Probabilistische Algorithmen

- Der klassische "totale" Korrektheitsbegriff von Algorithmen kann auf zweierlei Weise abgeschwächt werden:

---

1. Sogenannte *Las Vegas-Algorithmen:*

   - Abschwächung der Terminierungsforderung
   - also: "partiell korrekt und Terminierung mit *Wahrscheinlichkeit* 1"
   - beachte: die (worst-case) Laufzeit solcher Algorithmen ist unbeschränkt!
   - Beispiel: obiger Election-Algorithmus für anonyme Ringe

---

2. Sogenannte *Monte Carlo-Algorithmen:*

   - Abschwächung der partiellen Korrektheit
   - "terminiert stets, ist aber nur mit Wahrscheinlichkeit $p < 1$ partiell korrekt"
   - also: $\exists$ *Restwahrscheinlichkeit* $\varepsilon = 1\text{-}p > 0$, dass das Ergebnis falsch ist!
   - nur verwenden, wenn:
     - ε sehr *klein* ist (oft: als Parameter des Algorithmus, etwa abhängig von der Laufzeit und damit "beliebig klein" *wählbar*)
     - dadurch deutliche Vorteile erzielbar (Problem effizienter oder überhaupt erst lösbar)

- beachte den "Sonderfall" $p = 1$ (also $\varepsilon = 0$): ein solcher Monte Carlo-Algorithmus wäre *total korrekt*, denn er hält stets und das Ergebnis ist dabei ("mit Wahrscheinlichkeit 1") korrekt!

Las Vegas-Election-Algorithmus für anonyme Ringe bekannter Grösse

A. Itai, M. Rodeh: Symmetry breaking in distributive networks. In Proc. FOCS'81, pp. 150-158, 1981(basierend auf dem Chang/Roberts-Prinzip)

- Prinzip:
  - wähle eigene Identität id = random (1,...,n), mit n = Ringgrösse
    (oder ein anderer Wert, z.B. 2n, n^2 oder einfach 2 ?)
  - message extinction wie gehabt
  - Nachrichten enthalten einen "hop counter": Zählt Anzahl besuchter Knoten
  - falls eine Nachricht mit eigener Identität empfangen wird:
    - prüfe, ob hop counter = n
      - *nein* $\rightarrow$ $\exists$ anderen Knoten gleicher Identität (merken mittels Flag!)
      - *ja* $\rightarrow$ gewonnen! (aber falls Flag gesetzt, gibt es andere Gewinner!)
  - falls es mehrere Gewinner gibt:
    - nur diese führen eine *neue Election-Runde* durch
    - daher enthalten Nachrichten auch eine Rundenkennung (alte Nachrichten werden in der nächsten Runde einfach ignoriert)

    (FIFO-Kanäle notwendig?)

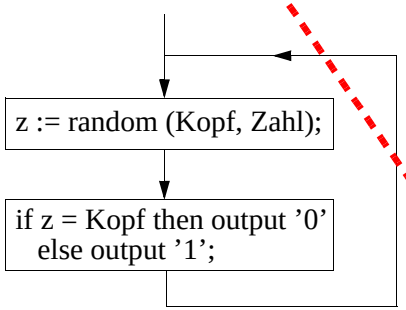
- Ohne Beweis: *Erwartungswert bzgl. Rundenzahl* $\leq e\ (n/n-1)$
  - vgl. mit vorherigem Algorithmus für Ringgrösse 2

  (2.718281828...)
- Nachrichten- / Zeitkomplexität ist im Prinzip aber unbegrenzt!
- Algorithmus ist partiell korrekt: *Wenn* er hält, dann mit genau einem leader!
- Verallgemeinerung auf *allgemeine Netze* mit Echo-Algorithmus (statt Ring) ist möglich
  - wenn die durch Echos gemeldete Baumgrösse $\neq n$ ist, neue Runde starten etc.

Zufällige reellwertige Zahlen?

- Wenn man im Verfahren von Itai/Rodeh zufällige reelle Zahlen (z.B. zwischen 0 und 1) für die id wählen könnte...
  - wie hoch wäre dann die Wahrscheinlichkeit, dass zwei Prozesse sich für die gleiche Identität entscheiden?
  - wie hoch wäre dann die Rundenzahl bzw. die Nachrichtenkomplexität?

- Realisierung solcher Zufallszahlengeneratoren?



- liefert eine unendliche Folge von 0en und 1en
- Verwende diese als die Nachkommastellen von 0.··· im Dualsystem
- Zurückführung des Problems auf perfekten binären Zufallsentscheider

- Unendliche Folgen lassen sich aber nicht in endlicher Zeit generieren und mit endlich vielen Bits speichern...

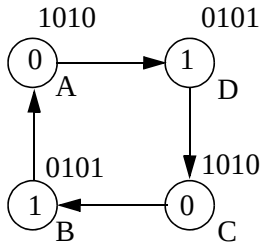
- Wie wäre es statt dessen mit einer "lazy" Variante, die notwendige Nachkommastellen nur auf Anfrage produziert?
  - etwa: liefere zunächst nur die ersten 32 Nachkommastellen; wenn mehr benötigt werden, fordert man das Objekt "zufällige reellwertige Zahl zwischen 0 und 1" auf, weitere Stellen zu liefern
  - Denkübung: hilft das (grundsätzlich) bei unserem Problem?

Genügt ein einziges Zufallsbit?

- 1993 haben S. Kurtz, C. Lin, S. Mahaney einen anderen Las-Vegas-Algorithmus für das Election-Problem vorgestellt, der hier *grob skizziert* wird:

für Ringe bekannter Grösse

  - jeder Knoten bestimmt *ein* zufälliges Bit und sendet es nach "rechts"
  - jeder Knoten reicht n-1 Mal ein empfangenes Bit nach "rechts" weiter
  - danach hat jeder Knoten *alle* n Bits (zyklisch verschoben!) gesehen
  - jeder Knoten prüft, ob bei "Ringshift" *seines* gesehenen Bitstrings dieser mit weniger als n shifts erhalten wird (nichttrivialer Automorphismus)
  - falls ja (selten!) --> gesamter Algorithmus wird wiederholt
  - falls nein: unter den n verschiedenen Bitstrings gibt es genau einen maximalen (bei Interpretation als Dualzahl)
  - der eindeutige Prozess, der diesen gesehen hat, ist der Leader



- symmetrische Lösung: alle Knoten führen den gleichen Algorithmus aus
- "common sense of orientation" wird vorausgesetzt
- in nebenstehendem Szenario ging es nicht gleich in der ersten Runde gut...

Beachte: die Laufzeit des Algorithmus ist prinzipiell unbegrenzt; wenn er hält, ist ein Leader allerdings eindeutig bestimmt

- Denkübungen (nicht ganz einfach!):
  - wie hoch ist die Best-case-Nachrichtenkomplexität?
  - macht es einen Unterschied, ob die Ringgrösse eine Primzahl ist?
  - kann man die Wahrscheinlichkeit abschätzen, dass eine einzige Runde (für eindeutiges Maximum) genügt?
  - wie hoch mag die *erwartete* Nachrichtenkomplexität sein?

Schätzung der Ringgrösse?

- Für den vorherigen Algorithmus ist die Kenntnis der Ringgrösse n entscheidend.

- Bei *unbekannter Ringgrösse* lässt sich diese aufgrund von zyklischen Wiederholungen im Bitstring mit wenigen Läufen mit hoher Wahrscheinlichkeit korrekt schätzen...
  - Wieviele Läufe sind für eine gewisse Sicherheit notwendig?
  - Konsequenzen, wenn man sich unerkanntermassen irrt?

Wir wollen das an dieser Stelle nicht weitertreiben...
Aus "philosophischer Sicht" ist allerdings interessant:

- Was ist an minimaler (struktureller) Information notwendig, um Symmetrie zu brechen? (Beispiele: Ringgrösse ist eine unbekannte Primzahl; obere Schranke für die Ringgrösse...)
- Unter welchen minimalen Voraussetzungen ist eine deterministische oder probabilistische Lösung möglich?
- Wie "sicher" und effizient können probabilistische Algorithmen für dieses Anwendungsproblem sein?

Satz (ohne Beweis): *Für anonyme Netze unbekannter Grösse existiert kein (det. oder prob. Las Vegas) Election-Algorithmus.*

- Daher stellt sich die Frage, ob die Grösse zumindest mit hoher Wahrscheinlichkeit korrekt abgeschätzt werden kann!

Kenntnis der Knotenzahl?

Satz (ohne Beweis): *Für anonyme Netze unbekannter Grösse existiert kein (det. oder prob. Las Vegas) Election-Algorithmus.*

- Daher stellt sich die Frage, ob die Grösse zumindest mit hoher Wahrscheinlichkeit korrekt abgeschätzt werden kann!

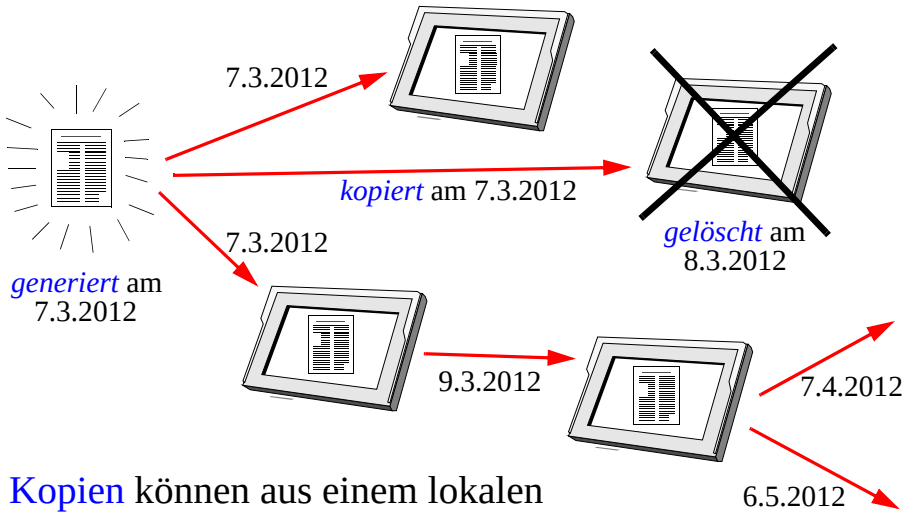
- *Bestimmung der Grösse anonymer Ringe mit einem Monte Carlo-Algorithmus* (hier nur Andeutung des Prinzips):
  - Jeder Prozess p hält einen konservativen Schätzwert $\bar{n}_p \leq n$ (initial: $\bar{n}_p = 2$).
  - Prozess p generiert ein Token mit einer Zufallsmarke aus $\{1,...,r\}$ und sendet es $\bar{n}_p$ Schritte weit. Das Token enthält dazu den Wert $\bar{n}_{tok}$ (= $\bar{n}_p$ des Senders p).
  - Prozess p erhöht sein $\bar{n}_p$ um 1 und sendet ein neues Token, wenn:
    - er ein Token empfängt mit $\bar{n}_{tok} > \bar{n}_p$, oder
    - er ein Token empfängt mit $\bar{n}_{tok} = \bar{n}_p$, aber dessen Marke nicht der eigenen Marke entspricht.

- Wir notieren ohne Beweis: Es werden höchstens $O(n^3)$ Nachrichten generiert; es sind am Ende alle $\bar{n}_p$ identisch mit $\bar{n}_p \leq n$; und es ist $\bar{n}_p = n$ mit einer Wahrscheinlichkeit $\geq 1-(n-1)\,(1/r)^{n/2}$. (Die Wahl von r ist also "kritisch".)
- Varianten, bei denen der Schätzwert $\bar{n}_p$ gelegentlich schneller wächst, sind möglich.

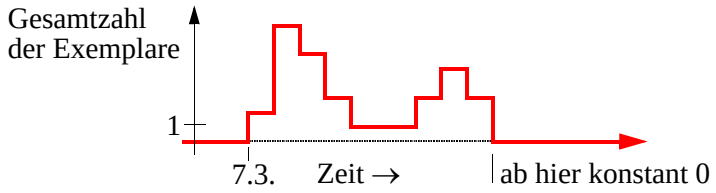
Garbage-Collecton in verteilten Systemen

Aussterben einer Menge von Kopien

- Ein Beispiel: On-line-Kopien einer Zeitung:



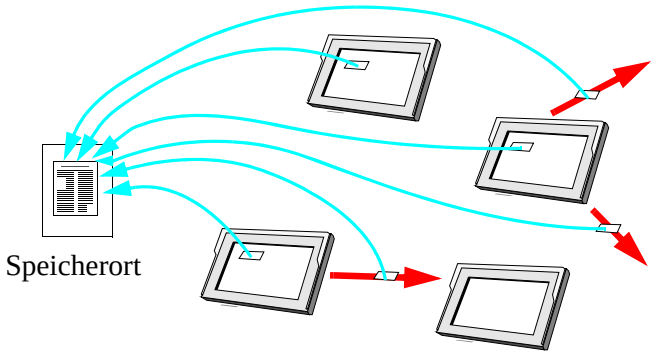
- Kopien können aus einem lokalen Exemplar erzeugt und verteilt werden
- Exemplare werden gelegentlich gelöscht



- Interessante Frage (erst *nach* der Erzeugung sinnvoll):
  Ist die Zeitung schon "ausgestorben"?
- Präziser: Ist die Gesamtzahl der Exemplare = 0?

Beobachten des Referenzzählers?

- Andere Idee: "copy by reference" statt "copy by value"
- Also: Zeitungsexemplare als Nur-lese-Kopien halten und lediglich eine *Referenz* auf den Speicherort übergeben
  - *kopiert* wird also nur ein kurzer *Verweis*, z.B. www.faz.ffm.de/07_03_12
  - im WWW also eine URL
  - bei Bedarf könnte eine echte Kopie angefordert werden ("copy by need")



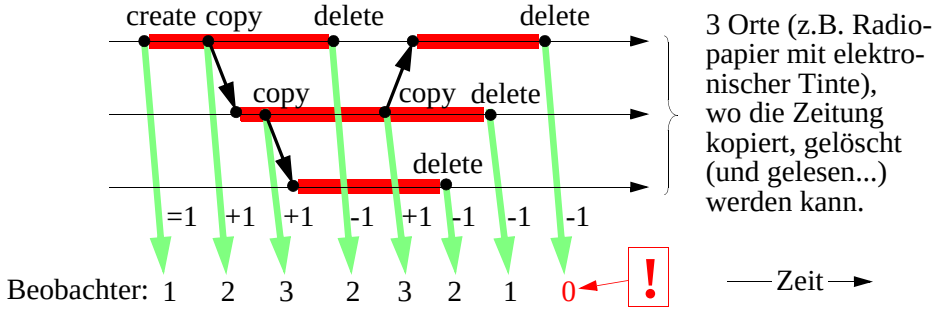
- Copy → Übermitteln der Referenz (= Zugriffspfad / Adresse)
- Delete → Löschen der Referenz

- "Beobachter" befindet sich z.B. am Speicherort
- *Referenzzähler* = 0 → Objekt physisch löschen
  - Begründung: Objekt kennt ja sowieso keiner mehr... (ist Garbage!)
  - aber natürlich nur bei kausaltreuer Beobachtung korrekt!

→ *Garbage-Collection*-Problem in verteilten Systemen!

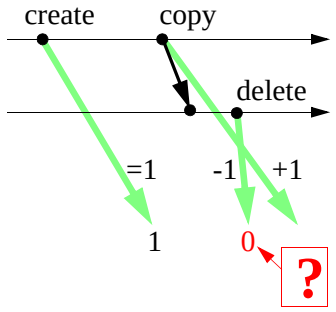
Zählen der Exemplare?

- Lösungsansatz: Beobachter wird informiert über
  - einmaligen Erzeugungsvorgang ("create-Ereignis")
  - jeden Kopiervorgang ("copy")
  - jeden Löschvorgang ("delete")



3 Orte (z.B. Radiopapier mit elektronischer Tinte), wo die Zeitung kopiert, gelöscht (und gelesen...) werden kann.

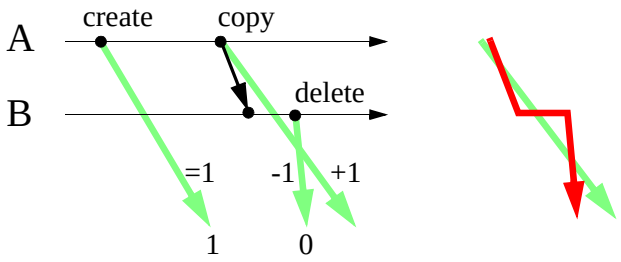
- Aber Achtung: Beobachtung ist u.U. nicht kausaltreu!



- Beachte: delete-Ereignis ist eine *kausale Konsequenz* des copy-Ereignisses ("ohne copy auch kein delete")
- Beobachter sieht jedoch die Konsequenz *vor* ihrer Ursache!
- Beobachter meint *fälschlicherweise*, dass die Zeitung unwiederbringlich verloren sei!

Was ist der Grund für das Problem?

- Kennen wir das nicht schon von der vert. Terminierung?



- Eine Einzelnachricht (Meldung von "copy") wurde in *indirekter* Weise *überholt* (Pfad von copy-Nachricht und delete-Meldung)

- Auf dem Überholpfad können Ereignisse liegen (hier z.B. "delete"), die "Konsequenzen" des überholten Ereignisses ("Ursache", hier "copy") darstellen

- Vermutung: Vermeiden von (direkten oder indirekten) Überholungen löst das Problem, d.h. liefert immer *kausaltreue Beobachtungen*

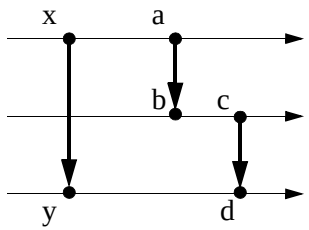
- Nicht-kausaltreue Beobachtungen sind die Ursache für viele konzeptuelle Probleme verteilter Systeme! **!**
  - z.B. verteilte Terminierung, Schnappschuss, Deadlockerkennung...

- Wie könnte man also das Überholen von Nachrichten durch Pfade anderer Nachrichten vermeiden? **?**

Vermeiden (indirekter) Überholungen?

1. Idee: Verwendung von *synchroner* Kommunikation

   - zumindest bei Meldungen an den Beobachter



Falls Nachrichtenpfeile *senkrecht* sind (Nachrichten also keine Übertragungszeit benötigen), kann es keine direkten oder indirekten Überholungen geben: jede *später* ausgesandte Nachricht (die den Anfang eines Überholpfades bilden könnte), kommt auch *später* an

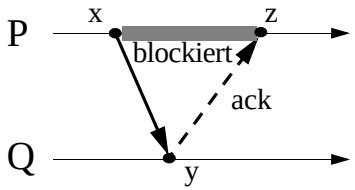
$t(y) = t(x) < t(a) = t(b) < ... < t(c) = t(d) \quad \Rightarrow \quad t(y) < t(d)$

Informell: "Blitzschnelle" Nachrichten kann man nicht überholen

---

2. Idee: Sender so lange blockieren, bis der Empfänger ein *Acknowledgement* zurückgesandt hat

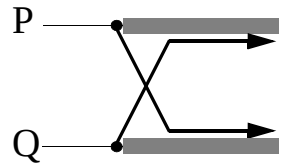
   - aber das ist ja nichts anderes als eine Implementierung / Simulation synchroner Kommunikation!



Jede bei P *nach* x (und damit nach z) gestartete Nachrichtenkette kommt bei Q garantiert *nach* y an

(Nachrichtenpfeile verlaufen nie "rückwärts" in der Zeit)

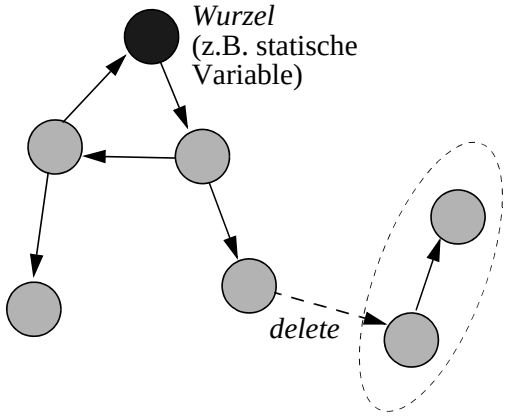
Problem bei einer solchen "Betriebsart" verteilter Systeme sind *Deadlocks*, wenn etwa "gleichzeitig" P an Q und Q an P sendet



Andere, bessere Idee zur Vermeidung indirekter Überholungen?

Garbage-Collection

- "Verpointerte" Objekte



Diese beiden Objekte sind nicht mehr von der Wurzel aus erreichbar und werden zu "garbage"

Ein *Garbage-Collector* soll solche Objekte identifizieren und deren Speicher wiederverwenden

- Wenn man die Objekte als "aktiv" ansieht, hat man im Grunde genommen ein paralleles / verteiltes System
  - Vgl. Puppentheater: Eine einzige Person bedient die Puppen im Zeitmultiplex; man kann dies aber als paralleles System autonomer Objekte ansehen

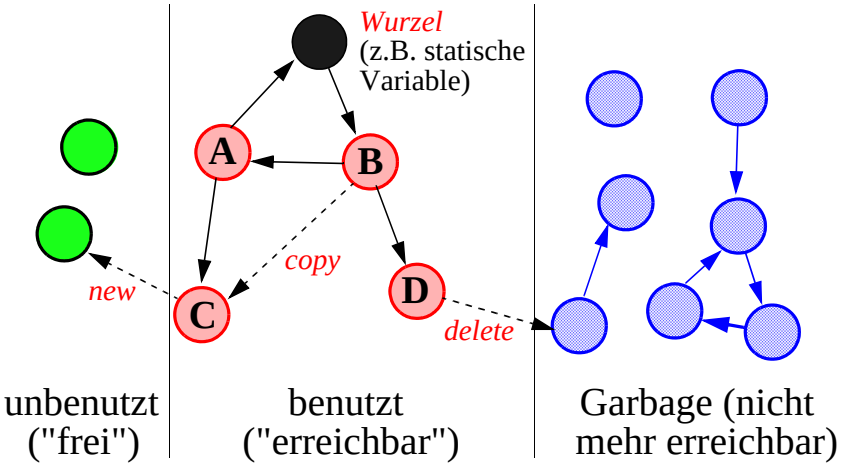
- Typischerweise läuft auch der Garbage-Collector (echt oder im Zeitmultiplex) parallel zur Anwendung

⇒ Algorithmen zur Identifikation von Garbage-Objekten im parallelen (oder verteilten) Fall nützen auch bei sequentiellen objektorientierten Systemen

Garbage-Collection ist allerdings (insbesondere in einer verteilten Umgebung) nicht trivial!

Garbage-Collection-Modell

- Zweck: Recycling von "verbrauchtem", ungenutztem Speicher
- Bei Sprachen mit dynamischem Speicher und Zeigerstrukturen
  - historisch: LISP (bereits in den frühen 1960er Jahren)
  - Interesse heute: objektorientierte Sprachen (+ evtl. parallele Implementierung)
  - statische Variablen und Variablen im Laufzeitkeller ("Stack") stets erreichbar, dynamische Variablen auf dem Speicher-Heap u.U. jedoch "abgehängt"



- *copy:* Füge *neue Referenz* zwischen 2 erreichbaren (!) Objekten hinzu
  z.B.: B.refvar := A.refvar

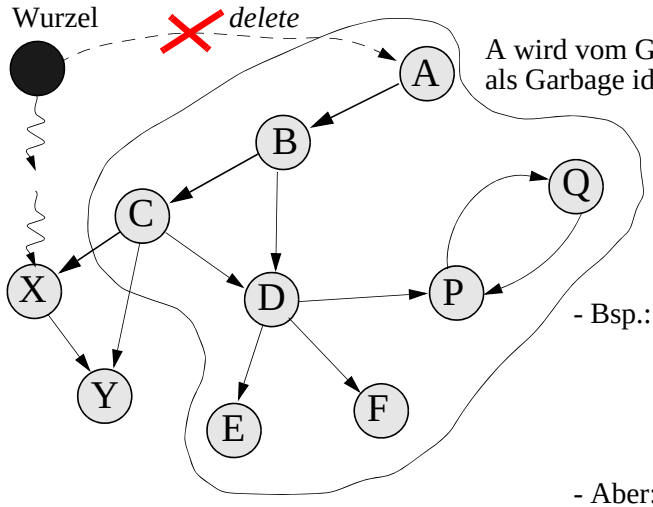
- Objekt *erreichbar* ⇔ Es gibt Pfad von der Wurzel dorthin
- "Garbage sein" ist stabiles Prädikat (vgl. Terminierung!)
- *Mutator* ↔ *Collector* spielen mit-/gegeneinander

Anwendungsprogramm	Kontrollprogramm
(manipuliert Zeiger zwischen Objekten mittels *copy*, *delete* und *new*)	identifiziert Garbage

Rekursives Freigeben

- Falls ein Objekt als Garbage erkannt wird:
  - sollten seine ausgehenden Referenzen gelöscht werden,
  - damit kann eine evtl. grössere daran "aufgehängte" Substruktur als Garbage erkannt werden!

  → rekursives Erkennen von Garbage-Objekten, "pointer chasing"



A wird vom Garbage-Collector als Garbage identifiziert

- Bsp.: A Garbage
  → B Garbage
  → C Garbage
  → D Garbage
  → E, F Garbage
- Aber: X, Y kein Garbage!

- P, Q sind dann auch Garbage, manche Garbage-Collectoren können solchen "zyklischen Garbage" jedoch nicht erkennen!

Das "Stop the World"-Paradigma

Mark & sweep-Algorithmus:

Das geschieht meist automatisch, weil Weiterarbeit unmöglich ist!

1) Mutator anhalten ("out of memory")
2) Alle erreichbaren Objekte markieren (ausgehend von der Wurzel) → Graph-Traversierung
3) Garbage (= alle unmarkierten Objekte) einsammeln
4) Mutator wieder starten

"sweep" durch den Hauptspeicher und z.B. in eine Freispeicherliste einketten

- "Stop the world"-Paradigma → schlechte Lösung für Realzeit- und interaktive Anwendungen!
  - erst recht untragbar in verteilten Systemen!

- Vergleiche dies mit dem folgenden (schlechten!) Terminierungs-Entdeckungverfahren:
  - friere alle Prozese ein
  - ermittle Gesamtzahl der versendeten und empfangenen Nachrichten
  - falls identisch und alle Prozesse passiv sind: terminiert
  - ansonsten: "Auftauen" aller Prozesse
- Eingefrorener globaler Zustand ist konsistent!
  - "in aller Ruhe" ein Objekt nach dem anderen betrachten

"Behind the back copy"-Problem

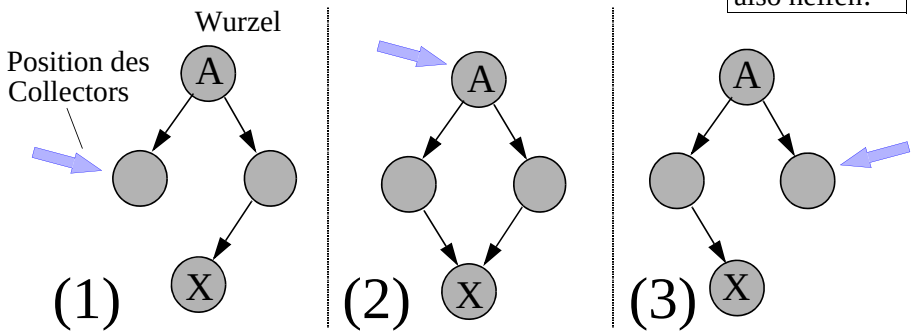
Concurrent / parallel / On-the-Fly-Garbage-Collection:

- Collector versucht, Garbage-Objekte zu identifizieren, *während* der Mutator aktiv ist
- Verhindert somit längere Wartezeiten der Anwendung

Traversiert den Graphen und markiert erreichbare Objekte

Collector könnte von gleichzeitig aktivem Mutator getäuscht werden → Kooperation notwendig!

Mutator muss dem Collector also helfen!



- Knoten X wird als nicht erreichbar angesehen...
- ...obwohl es *stets* einen Weg von A zu X gibt!

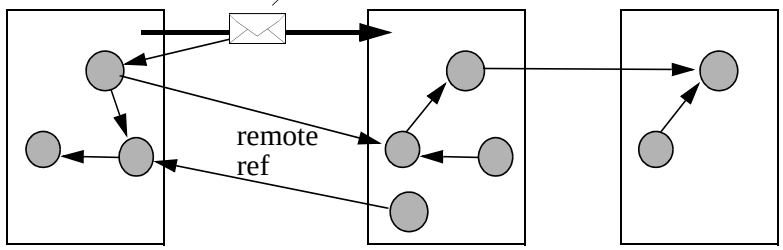
- Manipulationen "hinter dem Rücken" des Collectors
- Collector rekonstruiert aus seinen zusammengesetzten lokalen Beobachtungen einen falschen (nie existenten) Graphen
- dabei ist Beobachtungszeitpunkt = Besuchszeitpunkt

Verteiltes Garbage-Collection

Garbage-Collection in einem vert. System

Nachricht mit Referenz

copy ist nicht unbedingt atomar!



Prozessor mit lokalem Speicher

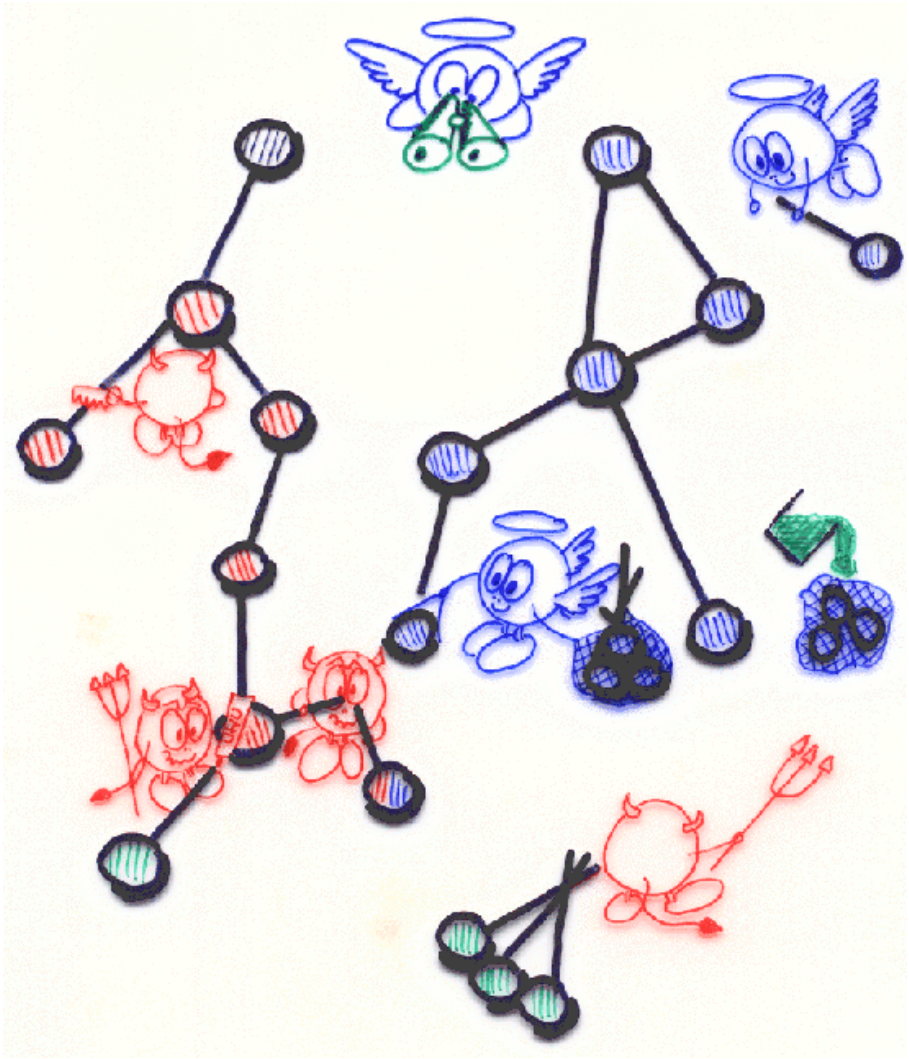
(1) Prozessorüberschreitende Referenzen ("*remote ref*")

(2) Nachrichten enthalten (Objekte mit) Referenzen auf andere Objekte ("*remote copy*")

- Referenzen in Nachrichten dürfen nicht übersehen werden!

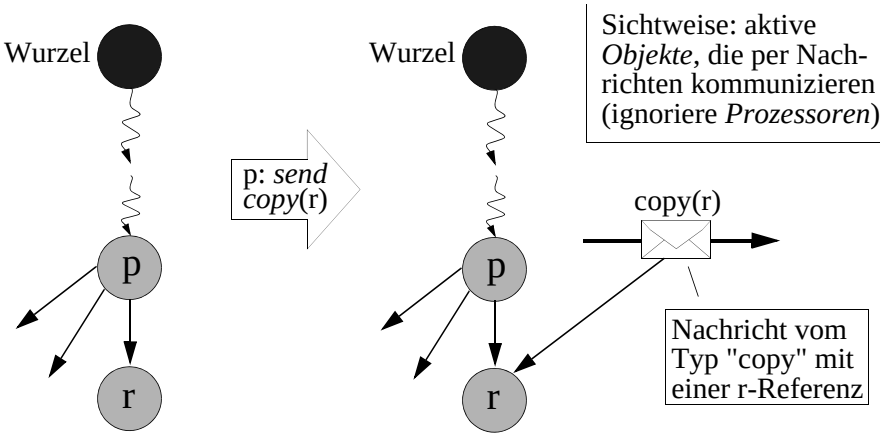
- GC typischerweise dezentral, parallel und hierarchisch
  - *lokale* GC (Annahme: alle eingehenden remote refs kommen von einem erreichbaren Objekt)
  - *globale* GC (Suchen prozessorübergreifender "Garbage-Ketten")
- GC aufwendiger als im nicht-verteilten Fall
- Mehrer Mutator / Collector (einen pro Prozessor ?)
  - Koordination zwischen den vielen Prozessen notwendig
- Pragmatisches:
  - "Stop the world" ist hier schon gar nicht angemessen
  - Kontrollkommunikation minimieren (Kosten, Effizienz)

Verteiltes Garbage-Collection



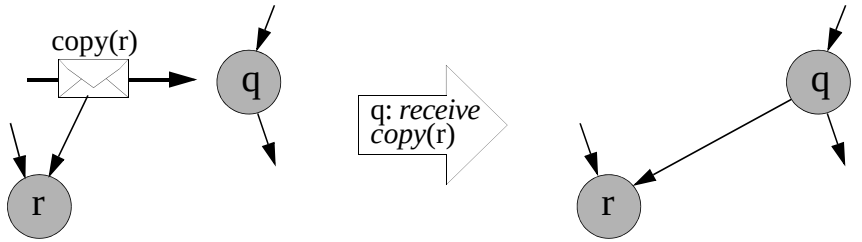
Wirkung der Mutator-Operationen

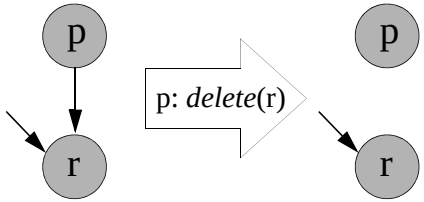
(auf den globalen Graphen der "verzeigerten" Objekte)



Sichtweise: aktive *Objekte*, die per Nachrichten kommunizieren (ignoriere *Prozessoren*)

Beachte: Copy-Operation ist nicht atomar ⇒ Aufspalten in zwei Aktionen *send / receive copy*

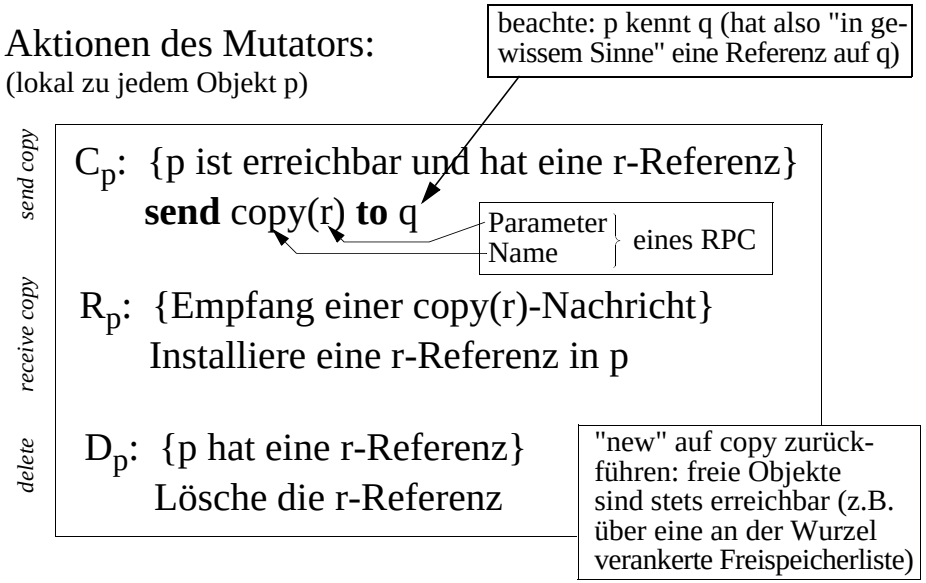




- "new" hier nicht relevant
- jede Aktion ändert einen kleinen Teil des globalen Graphen
- Folge solcher Änderungen → "Berechnung"
- hier: "Interleaving-Modell": Operationen sind atomar ("zeitlos") → es gibt keine gleichzeitige Aktionen → verschränkte Ausführung

Formalisierung des verteilten Garbage-Collection-Problems

Aktionen des Mutators:
(lokal zu jedem Objekt p)



- So "sieht" der Collector die Basisberechnung
- Vergleiche dies mit Basisaktionen beim Problem der verteilten Terminierung!
- Zu tun: Ergänzung um die Bedürfnisse des Collectors:
  - Ergänzung der 3 Aktionen mit weiteren Anweisungen, um die notwendige Kooperation mit dem Collector zu erreichen
  - evtl. zusätzliche atomare Aktionen für überlagerte Kontrollnachrichten
- Bedingungen an eine korrekte Lösung:
  - *Safety*: Wenn ein Objekt eingesammelt wird, dann ist es Garbage
  - *Liveness*: Wenn ein Objekt Garbage ist, dann wird es *schliesslich* eingesammelt

Verteilte Terminierung und Garbage-Collection

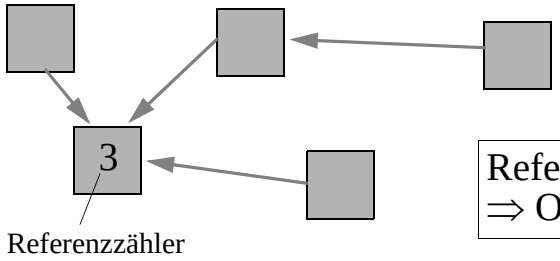
- Interessante Analogie zwischen beiden Problemen:

Verteilte Terminierung	*Garbage-Collection*
Überlagerter Kontrollalgorithmus	
Globale *Terminierung* ist ein stabiles Prädikat	"Objekt ist *Garbage*"
Kontrollalgorithmus soll das globale Prädikat entdecken	
Kontrollalgorithmus nebenläufig zur Anwendung	
Senden einer *Nachricht*	Kopieren einer *Referenz*
Problem: Behind the back	
reactivation	*copy*

- Können Lösungen des einen Bereiches auf den anderen Problembereich angewendet werden?

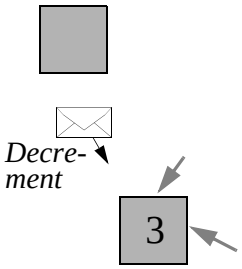
Referenzzähler-Verfahren

- Idee: Jedes Objekt weiss, wie oft es referenziert wird
  - wird dieser Referenzzähler 0 → Garbage
- Nachteil: Zyklischer Garbage wird so nicht entdeckt
- Zugehörigen Referenzzähler "atomar" zusammen mit der copy- oder delete-Operation aktualisieren
  - relativ einfach in einem nicht-verteilten System
- In einem verteilten System mit Kontrollnachrichten: *Increment*- / *Decrement*-Nachrichten



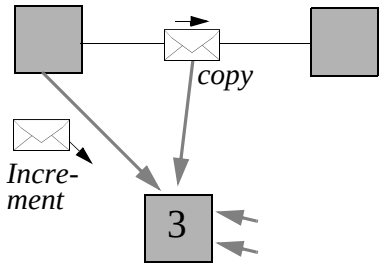
Referenzähler = 0
⇒ Objekt ist Garbage

Löschen einer Referenz:



3 ⇒ 2 (bei Ankunft von Decrement)

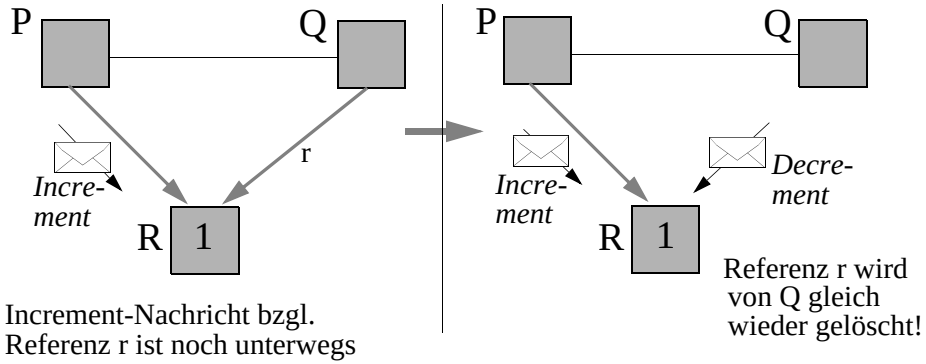
Kopieren einer Referenz:



3 ⇒ 4

Fehlinterpretation bei Referenzzählern

Kopieren kann zu *Fehlinterpretationen* führen!



Increment-Nachricht bzgl. Referenz r ist noch unterwegs

Referenz r wird von Q gleich wieder gelöscht!

Decrement schneller als vorangehendes (!) Increment
⇒ Referenzzähler wird 0, jedoch kein Garbage!

Korrektheitsbedingung:
Empfang von *Inc* früher als alle kausal abhängigen *Dec*

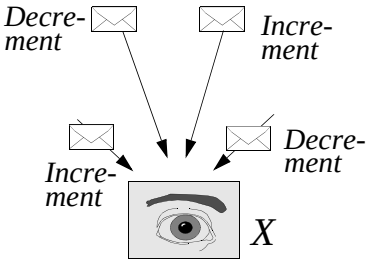
Wie dies garantieren?

1) Synchrone Kommunikation (→ "atomare" Operation)
2) Acknowledge von Inc-Nachrichten (+ warten)
3) "Causal Order" realisieren...

Objekt hat eine *kausaltreue Sicht* der Ereignisse (die es betreffen)

→ (indirekte) Überholungen vermeiden

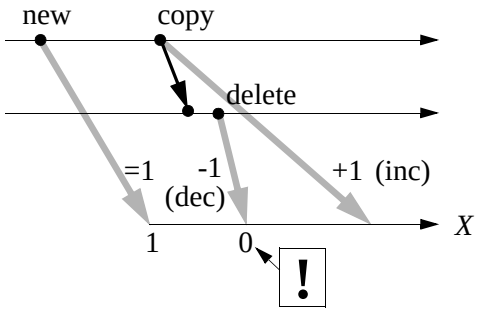
Garbage-Identifikation als konsistentes Beobachtungsproblem



Objekt X "beobachtet" die copy- und delete-Operationen, die es selbst betreffen

- die Increment- und Decrement-Nachrichten dienen der Beobachtung
- damit feststellen, ob *alle* copy durch delete kompensiert wurden

- Diese Beobachtung sollte kausaltreu sein!
  - zumindest darf ein dec nicht vor "seinem" inc empfangen werden

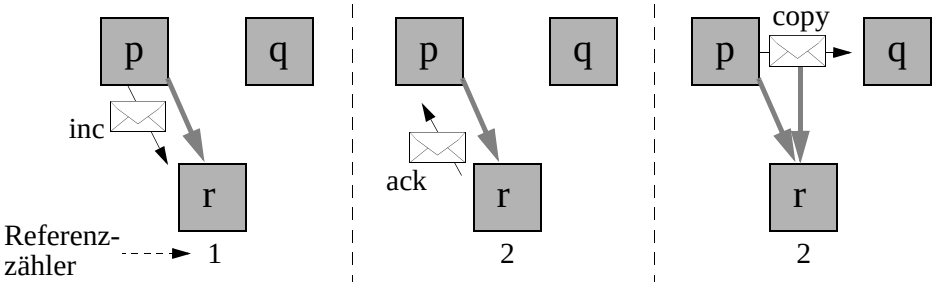


- *Jedes* Objekt beobachtet (im Prinzip) jedes andere
  - "Causal Order" (Globalisierung von FIFO) bzgl. Kommunikation gefordert

- Wie realisiert man kausaltreue Beobachtungen?
  - in diesem konkreten Fall mehrere Möglichkeiten (→ diverse Algorithmen!), z.B. copy solange verzögern, bis Bestätigung für inc-Nachricht eingetroffen
  - oder: kausaltreue Beobachter / "Causal Order"-Kommunikation allgemein implementieren?

Verteiltes Reference-Counting: Lösungen

- Erste Idee: jede Increment-Nachricht bestätigen und warten auf das ack, bevor das copy losgeschickt wird



- Korrektheitskriterium erfüllt:

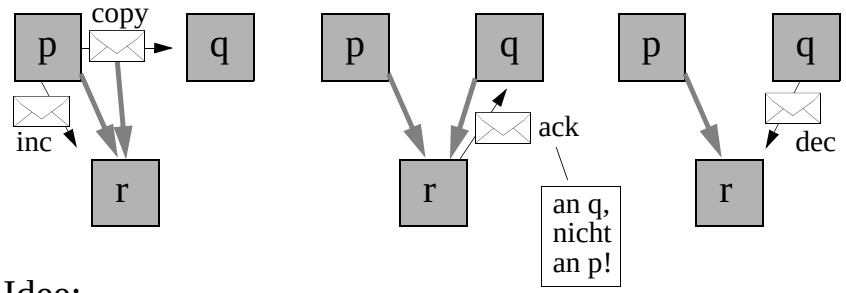
  inc wird vor einem kausal abhängigen dec empfangen

  (d.h. Objekt r erfährt von der Existenz einer neuen Kopie einer r-Referenz, bevor es vom Löschen dieser Referenz erfährt)

- Nachteile der Methode:
  - copy wird verzögert
  - zusätzliche Nachricht (ack)

→ insgesamt 3 Nachrichten pro copy-Operation

Variante von Lermen und Maurer



Idee:

Senden einer dec-Nachricht (bei Löschen der Referenz) erst dann, wenn das Objekt bereits ein zugehöriges ack (bzgl. inc) vom Zielobjekt der Referenz empfangen hat

$\Rightarrow$ Korrektheitskriterium erfüllt

- Beachte: Referenz selbst kann stets gelöscht werden, nur das Senden von dec muss verzögert werden

---

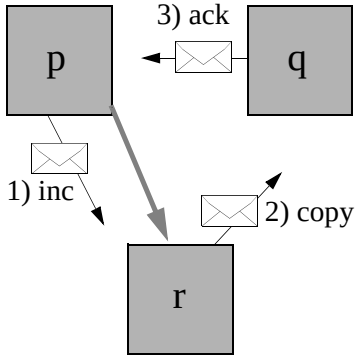
- Implementierungsskizze:
  - Zählen von empfangenen copy und ack-Nachrichten
  - dec-Nachricht erst senden, wenn Zähler ACK und COPY übereinstimmen
    - dann die Zähler ACK und COPY beide dekrementieren
    - "individuelle" Zuordnung von copy zu ack nicht notwendig! (Denkübung: Abschwächung zu |ACK|>0 $\wedge$ |COPY|>0 möglich?)

- *Vorteil*: kein Verzögern von Basisaktionen
- *Nachteil*: Ack-Nachricht und FIFO-Kanäle notwendig

wieso?

Varianten von Rudalics

1) 3-Nachrichten-Protokoll ("zyklisch"):



Idee: q bekommt Referenz auf r von r selbst; nachdem r seinen Zähler inkrementiert hat (veranlasst durch inc).

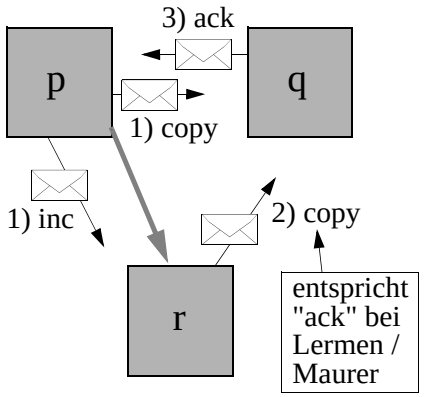
Bedingung: p darf seine r-Referenz erst löschen, wenn alle erwarteten ack-Nachrichten eingetroffen sind.

Denkübung: Wäre es auch möglich, dass das ack von r an p (statt q) gesendet wird?

- *Vorteil*: kein FIFO notwendig (falls FIFO garantiert ist: ack-Nachricht einsparen $\Rightarrow$ nur zwei Nachrichten pro copy!)
- *Nachteil*: Kopieren dauert länger (2 Nachrichten in Sequenz)

---

2) 4-Nachrichten-Protokoll:



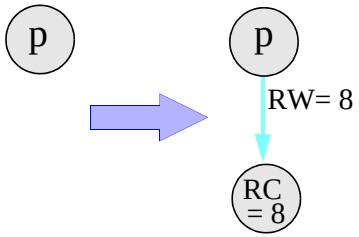
- Idee: ack erst senden, wenn *beide* copy-Nachrichten empfangen wurden
- q installiert die r-Referenz bei Empfang der *ersten* copy-Nachricht
- Unter welchen Bedingungen dürfen p bzw. q dec-Nachrichten senden?

- Vorteil: kein FIFO notwendig; keine Verzögerung
- Nachteil: 4 Nachrichten (davon 3 in Sequenz)

Die Referenzgewichts-Methode

(WRC: "Weighted Reference Counting")

Neues Objekt generieren ("new"):

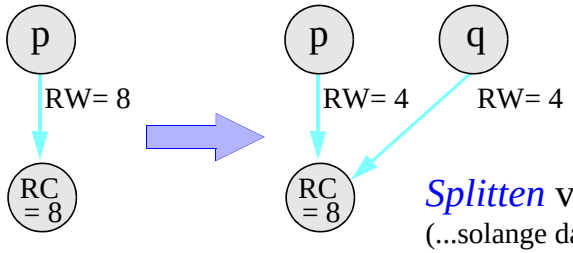


Reference *Weight*

Reference *Count*

(obwohl dadurch die Referenzen nicht tatsächlich "gezählt" werden)

Referenz kopieren ("copy"):



Splitten von RW

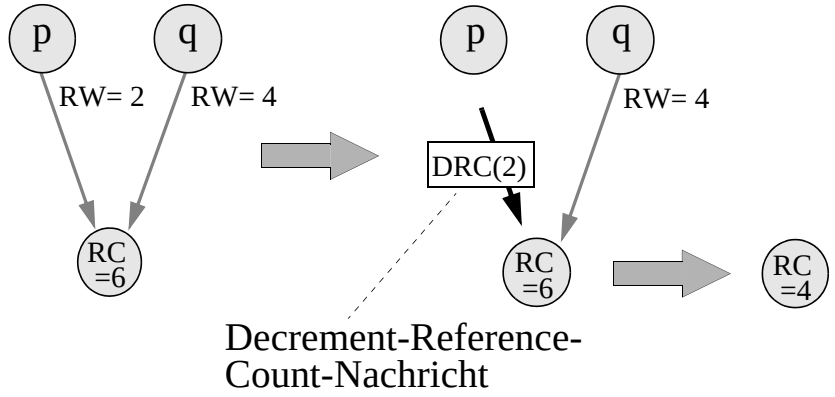
(...solange das geht!)

ansonsten (hier nicht genauer behandelt):

→ 1) "Recharge", oder:
2) Indirektionsobjekt

- Beachte: Es wird *keine Increment-Nachricht* benötigt!

Referenz löschen ("delete"):



Decrement-Reference-Count-Nachricht

Invariante: RC = ∑ RW + ∑ DRC

RC = 0 → Objekt ist Garbage

→ alle Referenzen dieses Objektes auf andere Objekte löschen (DRC-Nachrichten senden)

- Logarithmische Kompression (2er-Potenzen!) von RW
  - mit nur 2 Bit pro Zeiger lassen sich so RW bis max. 8 darstellen
  - statt 8 kann evtl. auch ein (etwas?) grösserer Maximalwert gewählt werden
  - RC so nicht komprimierbar ⇒ int-Variable mit "vielen" Bits pro Objekt
- Im Vergleich zu vorherigen Verfahren: Keine Verzögerung bei copy / delete; bei copy keine zusätzlichen Nachrichten!
  - RW = 1 (→ Zusatzaufwand) sollte ein eher seltenes Ereignis sein
- Analogie zur *Kredit-Methode* bei vert. Terminierung!

Kredit-Methode und WRC

Terminierungserkennung mit der Kreditmethode		WRC-Garbage-Collection
- Senden einer Nachricht (Splitten des Kreditwertes)	⟷	- Kopieren einer Referenz (Splitten des RW)
- Passiv werden (Kredit an Urprozess zurückgeben)	⟷	- R-Referenz löschen (Decrement-Nachricht an R)
Invariante: ∑ Kredit =1	⟷	*Invariante*: RC = ∑ RW + ∑ DRC
- Gesamtkredit beim Urprozess = 2^0=1	⟷	- RC = 0 bei R
Terminierung	⟷	*R ist Garbage*

Also: Kreditmethode "entspricht" WRC-Garbage-Collection!

Garbage-Collection und Terminierung

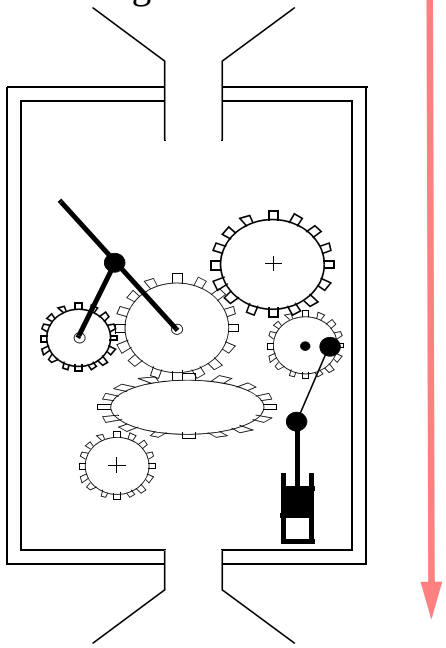
Theorem:

nicht notwendigerweise verteilte

Jeder *Garbage-Collection*-Algorithmus kann automatisch in einen Algorithmus zur Feststellung der *verteilten Terminierung* transformiert werden

Bemerkung: Für beide Probleme wurden viele nicht-triviale (und auch manche falsche!) Lösungen publiziert

Ein Garbage-Collection-Algorithmus

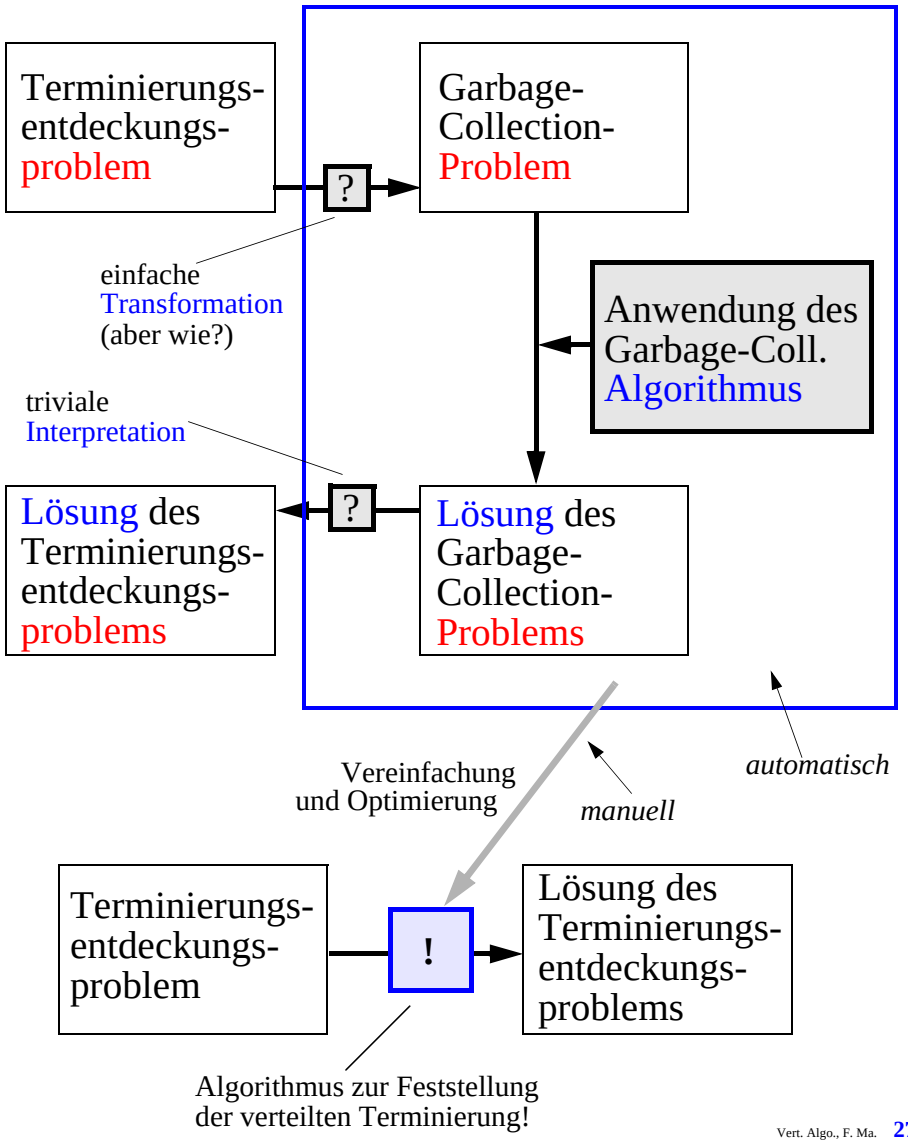


mechanische Transformation

Ein verteilter Terminierungsentdeckungs-Algorithmus

Problemtransformation

- Es wird das *Problem*, nicht der *Algortihmus* transformiert!

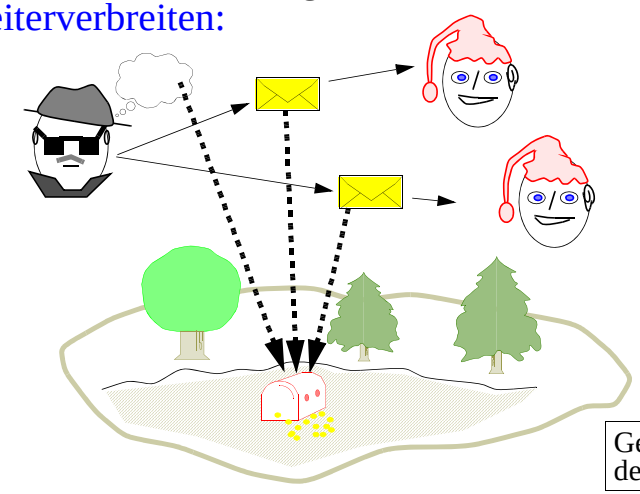


Vom Ende einer Geheimniskrämerei

Die vier goldenen Regeln der Geheimniskrämerei:

1) Es gibt Geheim-nisträger  und unein-geweihte Personen 

2) Nur ein Geheimnisträger kann das Geheimnis weiterverbreiten:



3) Wer das Geheimnis erfährt, wird zum Geheimnisträger

4) Ein Geheimnisträger kann das Geheimnis (endgültig) vergessen

---

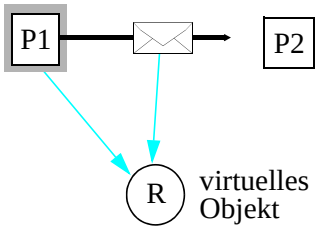
es gibt keine Geheimnisträger und keine Nachrichten mit dem Geheimnis

Geheimniskrämerei terminiert ⇔
Schatz nicht mehr zugreifbar ⇔ *Schatz ist "Garbage"*

→ "watchdog" beim Schatz meldet Terminierung...

Die Transformation

- Jeder Prozess wird in ein Wurzelobjekt transformiert
- Ein zusätzliches virtuelles Objekt R wird hinzugefügt



1) Prozess P aktiv ⇔
   P besitzt Referenz auf R

2) Jede Nachricht enthält
   eine Referenz auf R

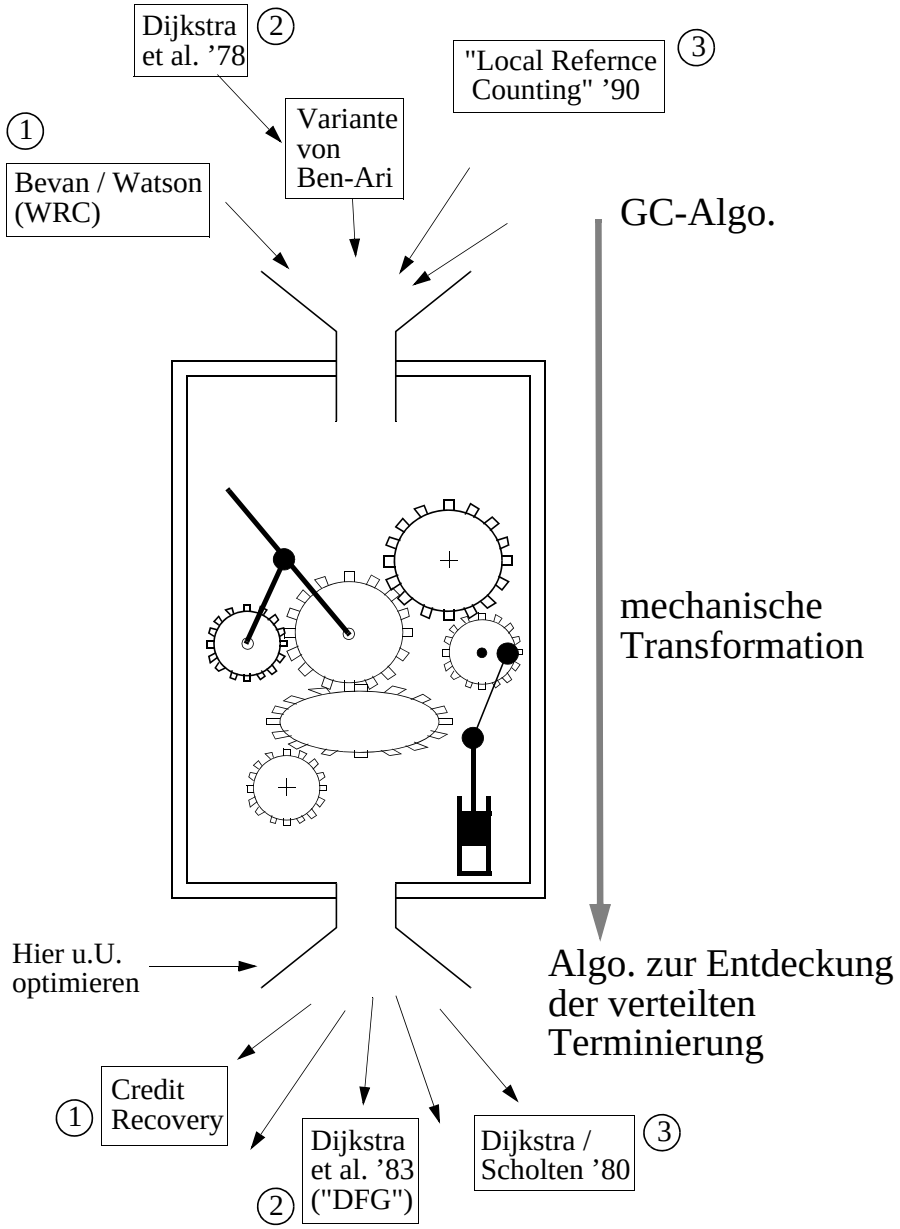
- Die beiden Regeln lassen sich ("induktiv") erfüllen:
  - ein (aktives) Objekt / Prozess sendet eine Kopie seiner R-Referenz mit jeder Nachricht
  - ein reaktivierter Prozess erhält eine R-Referenz
  - ein Prozess, der passiv wird, löscht seine R-Referenz

R Garbage ⇔ Es gibt keine Referenz auf R
⇔ Alle Prozesse passiv und keine Nachricht unterwegs ⇔ Verteilte Berechnung terminiert

- Also: verwende *irgendeinen* GC-Algorithmus
  (es entstehen keine Referenzzyklen, daher auch Referenzzählverfahren möglich!)

  → interpretiere Berechnung als GC-Problem

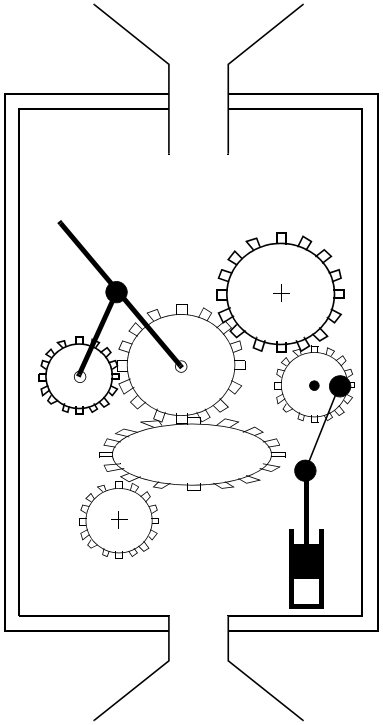
  → melde Terminierung, wenn R als Garbage erkannt



Bekannte Garbage-Collection-Verfahren werden so in bekannte und brauchbare Algorithmen zur Erkennung der verteilten Terminierung transformiert!

Die Patent-Story

WRC-Garbage-Collection-Algorithm patentiert:
Europäische Patentnummer 86309082.5



Ist der resultierende Terminierungs-erkennungsalgorithmus auch durch das Patent geschützt?

Das WRC-Patent

Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 225 755**
A2

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number: 86309082.5

(22) Date of filing: 20.11.86

20.11.86

(51) Int. Cl.: **G 06 F 12/02**

ICL

(30) Priority: 04.12.85 GB 8529890

(71) Applicant: INTERNATIONAL COMPUTERS LIMITED ICL House, Putney, London, SW15 1SW (GB)

Watson, Paul

(43) Date of publication of application: 16.06.87
Bulletin 87/25

(72) Inventor: Watson, Paul, 146, Hilda Park
Chester-Le-Street, Co-Durham DH2 2JY (GB)

(84) Designated Contracting States: BE DE FR GB NL

(74) Representative: Guyatt, Derek Charles et al, STC
Patents Edinburgh Way, Harlow Essex CM20 2SH (GB)

(54) Garbage collection in a computer system

(57) A computer system is described, having memory cells organised in a directed graph structure by means of pointers. Each cell has a reference count, and each pointer a weight value. If a new pointer to a cell is created by copying an existing pointer, the new and existing pointers are given weights whose sum equals the old value of the existing pointer. In this way, the sum of the weights of the pointers to any cell are maintained equal to its reference count. If a pointer is destroyed, the reference count of the cell to which it points is reduced by the weight of the pointer. Thus, when the reference count of a cell reaches zero, it is safe to assume that there are no more existing pointers to it, and hence that cell may be reclaimed (garbage-collected) for re-use.

nächste Seite →

(84) Designated Contracting States: BE DE FR GB NL

(54) Garbage collection in a computer system

(57) A computer system is described, having memory cells organised in a directed graph structure by means of pointers. Each cell has a reference count, and each pointer a weight value. If a new pointer to a cell is created by copying an existing pointer, the new and existing pointers are given weights whose sum equals the old value of the existing pointer. In this way, the sum of the weights of the pointers to any cell are maintained equal to its reference count. If a pointer is destroyed, the reference count of the cell to which it points is reduced by the weight of the pointer. Thus, when the reference count of a cell reaches zero, it is safe to assume that there are no more existing pointers to it, and hence that cell may be reclaimed (garbage-collected) for re-use.

Invariante!

Ein hierarchisches Patent

CLAIMS:-

1. A *computer system* having storage means containing a plurality of memory cells, at least some of which contain pointers to others of the cells thereby defining a directed graph structure in which each cell represents a node of the graph, wherein

(a) each pointer has a *variable weight value* associated with it,

(b) each cell contains a variable reference count value,

(c) whenever a cell is *allocated* in the storage means, its reference count is initially set to a predetermined value and the weights of any pointers to that cell are set to non-zero values such that the sum of those weights (or the weight if there is only one such pointer) is equal to the reference count of that cell,

(d) whenever a new pointer to a cell is created by *copying* an existing pointer, the new pointer and the existing pointer are given non-zero weight values whose *sum equals the old weight value* of the existing pointer, but the reference count of the cell is unaltered,

(e) whenever a pointer is *destroyed*, the reference count of the cell to which it points is reduced by the weight of that pointer, and

(f) cells with reference count equal to zero are reclaimed.

2. A system according to Claim 1 wherein the weight of each pointer is a *power of two*, and wherein, whenever a pointer is created by copying an existing pointer of weight greater than one, the weight of the existing pointer is divided by two and the result is stored as the weight value of the new and existing pointers.

3. A system according to Claim 2 wherein the weight of each pointer is *encoded in a compressed form as the logarithm* to the base two of the weight.

4. A system according to any preceding claim wherein, whenever a pointer is to be created by copying an existing pointer of weight equal to one, a *dummy cell* is created into which the existing pointer is copied along with its weight, the dummy cell is given a reference count greater than one, and the new and existing pointers are both made to point to this dummy cell, and are given non-zero weights whose sum equals the reference count of the dummy cell.

5. A system according to any preceding claim in which the system is a *multi-processor distributed computer system*.

10 CLAIMS

US-Version 4755939: hier wird nun eine *Methode* patentiert, nicht mehr, wie im europäischen Patent, ein *Computersystem*!

I claim:

1. A method of garbage collection in a computer system having storage means containing a plurality of memory cells, at least some of which contain pointers to others of the cells thereby defining a directed graph structure in which each cell represents a node of the graph, wherein the method comprises steps as follows:
   - (a) each pointer is provided with a variable weight value,
   - (b) each cell is provided with a variable reference count value,
   - (c) whenever a cell is allocated in the storage means, the reference count of the cell is initially set to a predetermined value and the weights of any pointers to that cell are set to non-zero values such that the sum of those weights (or the weight if there is only one such pointer) is equal to the reference count of that cell,
   - (d) whenever a new pointer to a cell is created by copying an existing pointer, the new pointer and the existing pointer are given non-zero weight values whose sum equals the old weight value of the existing pointer, but the reference count of the cell is unaltered,
   - (e) whenever a pointer is destroyed, the reference count of the cell to which the pointer points is reduced by the weight of that pointer,
   - (f) and cells with reference count equal to zero are reclaimed.

2. A method according to claim 1 wherein the weight of each pointer is a power of two, and wherein, whenever a pointer is created by copying an existing pointer of weight greater than one, the weight of the existing pointer is divided by two and the result is stored as the weight value of the new and existing pointers.

3. A method according to claim 2 wherein the weight of each pointer is encoded in a compressed form as the logarithm to the base two of the weight.

4. A method according to claim 3 wherein, whenever a pointer is to be created by copying an existing pointer of weight equal to one, a dummy cell is created into which the existing pointer is copied along with its weight, the dummy cell is given a reference count greater than one, and the new and existing pointers are both made to point to this dummy cell, and are given non-zero weights whose sum equals the reference count of the dummy cell.

5. A method according to claim 3 in which the system is a multi-processor distributed computer system.

6. A method according to claim 2 wherein, whenever a pointer is to be created by copying an existing pointer of weight equal to one, a dummy cell is created into which the existing pointer is copied along with its weight, the dummy cell is given a reference count greater than one, and the new and existing pointers are both made to point to this dummy cell, and are given non-zero weights whose sum equals the reference count of the dummy cell.

7. A method according to claim 2 in which the system is a multi-processor distributed computer system.

8. A method according to claim 1 wherein, whenever a pointer is to be created by copying an existing pointer of weight equal to one, a dummy cell is created into which the existing pointer is copied along with its weight, the dummy cell is given a reference count greater than one, and the new and existing pointers are both made to point to this dummy cell, and are given non-zero weights whose sum equals the reference count of the dummy cell.

9. A method according to claim 8 in which the computer system is a multiple-processor distributed computer system.

10. A method according to claim 1 in which the computer system is a multi-processor distributed computer system.

Patentsuche

- Sucht man nach Garbage-Collection-Verfahren (z.B. bei den WWW-Sites der Patentämter), findet man einiges...

Search Results
Query: (garbage collection)
139 of 2569032 matched

man hätte besser auch nach "garbage collector" etc. gefragt

5652883 Computer method and system for conservative-stack and generational heap garbage collection

5561785 System for allocating and returning storage and collecting garbage using subpool of available blocks

5560003 System and hardware module for incremental real time garbage collection and memory management

5446901 Fault tolerant distributed garbage collection system and method for collecting network objects

5819299 Process for distributed garbage collection

5832529 Methods, apparatus, and product for distributed garbage collection

5033930 Garbage collecting truck

5398334 System for automatic garbage collection using strong and weak encapsulated pointers

4755939 Garbage collection in a computer system

5901540 Garden tool for collection and removal of debris

...

Andere Garbage-Collection-Patente

- Es gibt viele Patente zum Thema "Garbage Collection", und manches hätte man vielleicht selbst erfinden können

- Titel: Fault tolerant distributed garbage collection system and method for collecting network objects
- Veröffentlichungsnr.: US5446901
- Veröffentlichungsdatum: 1995-08-29
- Erfinder: OWICKI SUSAN S (US); BIRRELL ANDREW D (US); NELSON CHARLES G (US); WOBBER EDWARD P (US)
- Anmelder: DIGITAL EQUIPMENT CORP (US)
- Klassifikationssymbol (IPC): G06F12/00

A distributed computer system includes a multiplicity of concurrently active processes. Each object is owned by one process. Objects are accessible to processes other than the object's owner. Each process, when it receives a handle to an object owned by any other process, sends a first "dirty" message to the object's owner indicating that the object is in use. When a process permanently ceases use of an object handle, it sends a second "clean" message to the object's owner indicating that the object is no longer in use. Each object's owner receives the first and second messages concerning usage of that object, stores data for keeping track of which other processes have a handle to that object and sends acknowledgement messages in return. The receiver of an object handle does not use the handle until its first message is acknowledged. Periodically, the object's owner sends status request messages to other processes with outstanding handles to that object to determine if any of those processes have terminated and updates its stored object usage data accordingly. A garbage collection process collects objects for which the usage data indicates that no process has a handle. The first and second messages include sequence numbers, wherein the sequence numbers sent by any process change in value monotonically in accordance with when the message is sent. Object owners ignore any message whose sequence number indicates that it was sent earlier than another message for the same object that previously received from the same process.

Andere Garbage-Collection-Patente (2)

- Man findet auch Querverweise auf viele andere Patente, auf wissenschaftliche Literatur dazu etc., z.B.:

5355483 : Asynchronous garbage collection
INVENTORS: Serlet; Bertrand, Paris, France
ASSIGNEES: NeXT Computers, Redwood City, CA
ISSUED: Oct. 11, 1994
FILED: July 18, 1991
SERIAL NUMBER: 732453

...With this method, the process being collected communicates its memory state ("a memory snapshot") to a garbage collecting process (GC), and the GC process scans the memory and sends back the information about garbage. As a result, the present invention permits garbage collection to be performed asynchronously. The process being scanned for garbage is interrupted only briefly, to obtain the memory snapshot. The process then runs without interruption while the garbage collection is being performed. The present invention makes the assumption that if an object is garbage at the time of the memory snapshot it remains garbage any time later,...

Beispiele für Softwarepatente (1)

Es gibt viel ganz offensichtliche Dinge, die patentiert wurden, das gilt auch für Algorithmen. Einiges davon ist in der fachlich beschlagenen Öffentlichkeit bekannt geworden, z.B. das LZW-Komprimierungspatent (US4558302) beim gif-Bildformat. Es gibt aber auch noch viele andere "interessante" Patente. Vieldiskutiert ist die Frage, ob Algorithmen (oder ganz allgemein "Intellectual Property") überhaupt patentiert werden soll.

Any word processor with a separate mode that the user selects when they wish to type in a mathematical formula. [US5122953]

A word processor which marks and makes correction to a document using two additional different colors. [US5021972]

Inventor(s): Nishi; Toshio
June 4, 1991
A word processor, including a keyboard through which characters can be inputted, a memory device for storing inputted character arrays and a display device capable of multi-color displays, is so programmed that corrections and additions are automatically displayed in a different color from the rest for the convenience of editing. ... When a completed document is finally stored in a document file, however, such color codes are deleted such that the document can be outputted in one color.

Statically allocating an initial amount of memory when a program is first loaded according to a size value contained in the program header. [US5247674]

Applicant(s): Fujitsu Limited, Kawasaki, Japan
Issued/Filed Dates: Sept. 21, 1993 / May. 13, 1991
A memory allocation system includes a unit for storing the information about the amount of memory required at the time of initializing each executable program in the control information of the file storing the program. The amount required is determined when the program is translated, assembled or compiled and linked. The memory allocation system also includes a unit for reading the information, indicating the amount of memory required at the time of initializing the program stored in the control information of the file, when loading of program is requested.

Beispiele für Softwarepatente (2)

Assigning a client request to a server process by first examining all the server processes not handling the maximum number of clients, and then assigning it to the server process currently servicing the fewest clients. [US5249290]

Applicant(s): AT&T Bell Laboratories, Murray Hill, NJ
Issued/Filed Dates: Sept. 28, 1993 / Feb. 22, 1991
A server of a client/server network uses server processes to access shared server resources in response to service requests from client computers connected to the network. The server uses a measured workload indication to assign a received client service request to a server process... a busy indicator provides a measured workload indication for each active process. The server uses the busy indicator to assign a new client service request to the least busy process.

Method for canonical ordering of binary data for portable operating systems. [US4956809]

Applicant(s): Mark Williams Company, Chicago, IL
Issued/Filed Dates: Sept. 11, 1990 / Dec. 29, 1988
...The method includes converting all binary data accessed from a file or communications channel from the canonical order to the natural order of the host computer before using the binary data in the host computer and converting all binary data which is to be sent to a file or communications channel from the natural order of the host computer to the canonical order before sending the binary data.

Remembering file access behavior and using it to control the amount of read-ahead the next time the file is opened. [US5257370]

Using of multiple read only tokens and a single read-write token to control access to a portion of a file in a distributed file system. [US5175851]

Quicksort implemented using a linked list of pointers to the objects to be sorted. [US5175857]

Intercepting calls to a network operating system by replacing the first few instructions of an entry point by a call to an intercept routine. [US5257381]

Beispiele für Softwarepatente (3)

Distinguishing nested structures by color. [US4965765]

Applicant(s): International Business Machines Corp., Armonk, NY
Issued/Filed Dates: Oct. 23, 1990 / May. 16, 1986
A method of distinguishing between nested expressions, functions, logic segments or other text by using a different color for each nesting level.

Das berüchtigte "xor-Cursor-Patent": **Method for dynamically viewing image elements stored in a random access memory array.** [US4197590]

Issued/Filed Dates: April 8, 1980 / Jan. 19, 1978
...An XOR feature allows a selective erase that restores lines crossing or concurrent with erased lines. The XOR feature permits part of the drawing to be moved or "dragged" into place without erasing other parts of the drawing. ...supporting another image on the display without destruction of the initially stored image... logically exclusively ORing together the accessed data for each element of the stored image and the data for the corresponding element of the image to be superimposed, and for reentering the resultant logical data into the same memory locations, said display then being generated from the resultant contents of said memory.

A parallelizing compiler that estimates the execution time for each of a number of different parallelization conversions and then selects the one that it thinks will be the fastest. [US5151991]

Any document storage system that has a digital camera to scan in documents, stores the documents on an optical disk, and uses character recognition software to construct an index. [US4941125]

Siehe auch zu obigen Beispielen:
www.base.com/software-patents/examples.html

Bemerkungen zum WRC-Patent...

From: munnari!tasis.utas.oz.au!ben@uunet.UU.NET (Ben Lian)
Subject: Watson & Watson's Weighted Reference Counting
Date: 19 Jun 89 11:55:52 GMT

... I thought you'd like to know that the Watson & Watson algorithm was PATENTED (yes, patented) by ICL in the early '80s. According to one of my colleagues, the patent covers all uses of the algorithm in which the reference weight is split (in some arbitrary ratio). This suggests that the patent covers most ANY application of the algorithm. Bevan's algorithm, also in the PARLE 87 proceedings, is an elegant subset of Watson & Watson's. (Ironically, Bevan won the 'best paper' award at the conference!) The patent permits the use of the algorithm in research contexts only; use in a commercial product is not allowed. (Which means that our Department's VLSI parallel combinator reduction machine will never see the light of day.)

Ben Lian

P.S. On hearing that the algorithm had been patented, my colleague ran up and down the corridor of our hut, exclaiming: "How could they do this? They might as well patent the addition operator!"


```
-------------------------------------------------------------------------------
Benjamin Y H Lian              ACSnet: ben@tasis.utas.oz
Dept. of EE & CS               ARPA  : ben%tasis.utas.oz.au@uunet.uu.net
University of Tasmania         BITnet: munnari!tasis.utas.oz!ben@
GPO Box 252C                           uunet.uu.net
Hobart, Tasmania 7001          UUCP  : {enea,hplabs,mcvax,uunet,ukc}!
A U S T R A L I A                      munnari!tasis.utas.oz!ben
```


Ein weiterer Brief von Ben Lian

As I see it, since the algorithm itself is patented, then any technique which makes use of weighted reference counts in any form is covered by the patent. This is regardless of different terminology and/or the strategy used to split reference weights.

I hope this has been helpful. If you have any further questions you might perhaps like to write to Andrew Partridge in my Department. He will be able to provide you with more accurate information—I am only an interested by-stander! Alternatively, you can write to Derek Guyatt or Paul Watson (as per the enclosed letter from the latter).

Yours sincerely

Ben Lian
(ben@tasis.uucp)

Watson war nicht der erste...

From: grit@carlos.llnl.gov (Dale Grit)
Message-Id: <8907101526.AA12531@carlos.llnl.gov>
To: ben@tasis.utas.oz
Subject: weighted reference counting
Status: R

I had heard about weighted reference counts in 1980 at a workshop at MIT, so I asked Arvind what he knew about them. He responded with the following:

"On reference weights: I brought the idea to the attention of Bob Thomas while he was doing his thesis at Irvine. I had heard it from Ken Weng in course of our many informal discussions. It is explained in a foot note in Ken Weng's Ph.D. thesis. I have mentioned the idea in my dataflow course at MIT as early as 1980. Paul (not our old buddy Ian) Watson gave a talk about it at MIT in 1986 (I think). He was quite surprised to learn that we knew about it so thoroughly. He acknowledges it in his Parle paper. He for a while was erroneously attributing the idea to me but I had him correct it. I did not know about the ICL patent. I will be surprised if it was filed in 1982. In any case I would not worry about it because it can't hold up in court if challanged. In fact 70 to 80% patents don't hold up when challanged."

If you need more information, please contact Arvind at

arvind@au-bon-pain.lcs.mit.edu

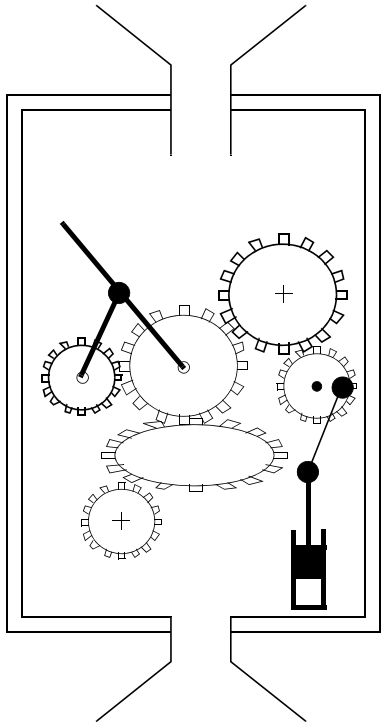
Dale Grit,
Computer Science Dept.,
Colorado State University
on leave at LLNL
grit@lll-crg.llnl.gov

Local Reference Counting (LRC)

Y. Ichisugi, A. Yonezawa: Distributed Garbage Collection Using Group Reference Counting, TR 90-014, Univ. of Tokyo

M. Rudalics: Implementation of Distributed Reference Counts, Internal Report, J. Kepler University, Linz

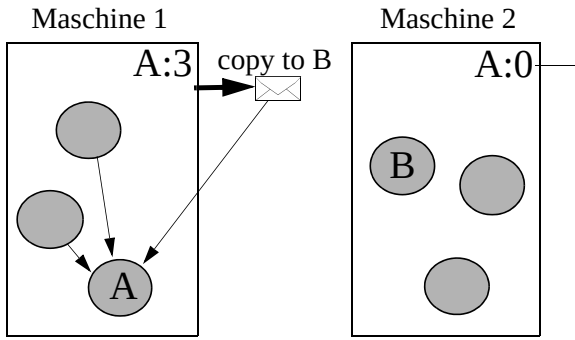
J. Piquer: *Indirect Reference Counting*, Proc. PARLE 91, 150-165

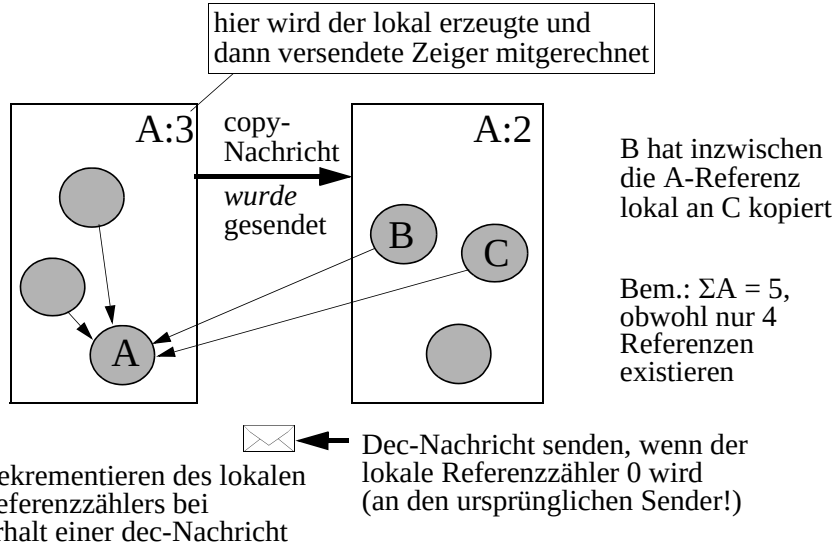


E.W. Dijkstra, C.S. Scholten: Termination Detection for Diffusing Computations. Inf. Proc. Lett. 11 (1980), 1-4

LRC-Garbage-Collection-Verfahren

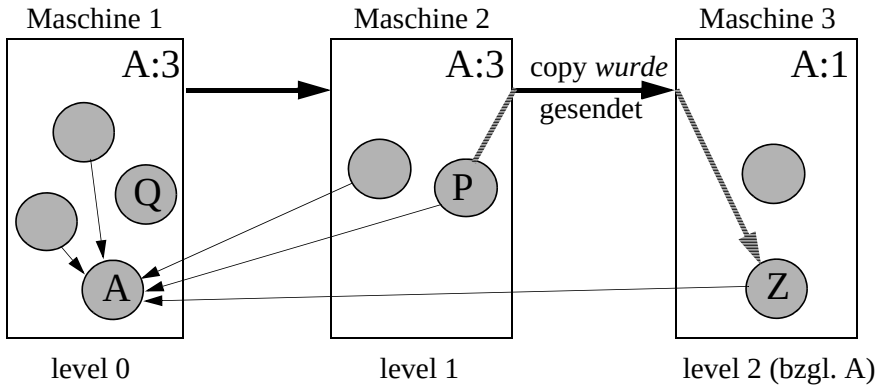
- Pro *Maschine* wird (potentiell) für jedes Objekt ein Referenzzähler gehalten, z.B. für Objekt A:





Beachte: Es wird angenommen, dass der *lokale* Referenzzähler *atomar* mit den Operationen copy, delete, Empfang einer dec-Nachricht aktualisiert werden kann

LRC: Weitervererben von Referenzen



- Maschine 1 muss nicht informiert werden, wenn z.B. Objekt P eine (Kopie seiner) A-Referenz an Z schickt!

- Korrektheit (safety):

Referenzbaum

Lokaler Referenzzähler auf level i+1 > 0
→ lokaler Referenzzähler auf level i > 0

⇒ Falls eine Referenz existiert, dann ist der Referenzzähler auf level 0 > 0

- wieso eigentlich *Baum*?
- wie genau ist "level" definiert?

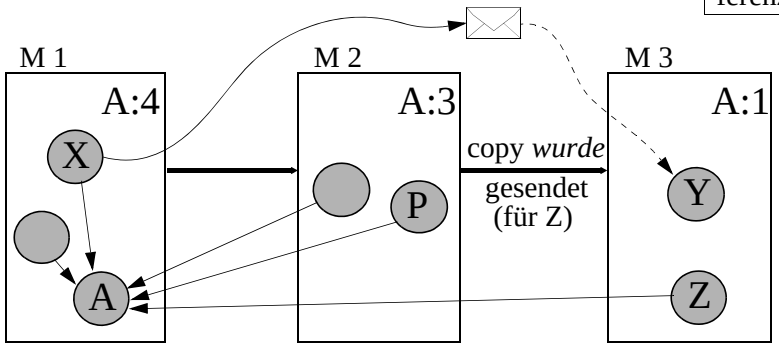
⇒ "Garbage-Kriterium": Referenzzähler auf level 0 = 0

- Denkübungen: wie steht es mit der *liveness*?
- was geschieht, wenn P eine A-Referenz an Q (Maschine 1) sendet? (→ Zyklen ?)

Der "Remote-ref"-Baum

- Was geschieht, wenn X auf Maschine 1 eine copy-Nachricht mit einer A-Referenz an Y auf Maschine 3 schickt?
  - Maschine 1 erhöht ihren lokalen A-Zähler beim Senden
  - Maschine 3 erhöht ihren lokalen A-Zähler beim Empfang

die schon eine A-Referenz hat



- Aber: Wann und an wen (hier: Maschine 1 und/oder 2) soll u.U. eine dec-Nachricht von Maschine 3 gesendet werden?

?

(1) wenn Y seine A-Referenz löscht → an M1 senden, und wenn Z seine A-Referenz löscht → an M2 senden

(2) wenn der lok. Referenzzähler 0 wird auf M3 → an M1 und M2 senden

(3) M2 "adoptiert" Y bei Empfang der copy-Nachricht → bei Empfang des copy eine dec-Nachricht an M1 senden (als hätte Y seine A-Referenz gleich gelöscht und dann sofort eine lokale Kopie von Z erhalten)

- Beachte bei der Lösung (3):
  - eindeutige Vorgängerbeziehung; keine Zyklen → *Baum* ("level" klar bestimmt)
  - neuer "Adoptivvater" M2 braucht hierbei nicht informiert zu werden
  - genausogut hätte M1 Objekt Z adoptieren können (M3 sendet dann dec-Nachricht an M2 bei Empfang der copy-Nachricht von X) → Optimierungspotential: wähle stets den Vater mit niedrigstem level... (wieso?)

LRC: Formale Beschreibung

- Ergänzen existierender atomarer Aktionen bzw. zusätzliche Aktionen:

C_p: {p ist erreichbar und hat eine r-Referenz}
send copy(r) **to** q;
$LRC_p(r) := LRC_p(r) + 1$

atomare Aktion: Änderung von LRC "gleichzeitig" mit dem Versenden der copy-Nachricht

R_p: {Bei Empfang einer copy(r)-Nachricht von q}
Installiere die r-Referenz;
if $LRC_p(r) = 0$
then {$LRC_p(r) := 1$; $FIRST_p(r) := q$}
else {$LRC_p(r) := LRC_p(r)+1$; **send** dec(r) **to** q}
fi

r-Referenz existierte schon bzw. p befindet sich schon im Referenzbaum

D_p: {p hat eine r-Referenz}
Lösche die r-Referenz;
$LRC_p(r) := LRC_p(r) - 1$;
if $LRC_p(r) = 0$ **then call** collect(r) **fi**

X_p: {Bei Empfang einer dec(r)-Nachricht}
$LRC_p(r) := LRC_p(r) - 1$;
if $LRC_p(r) = 0$ **then call** collect(r) **fi**

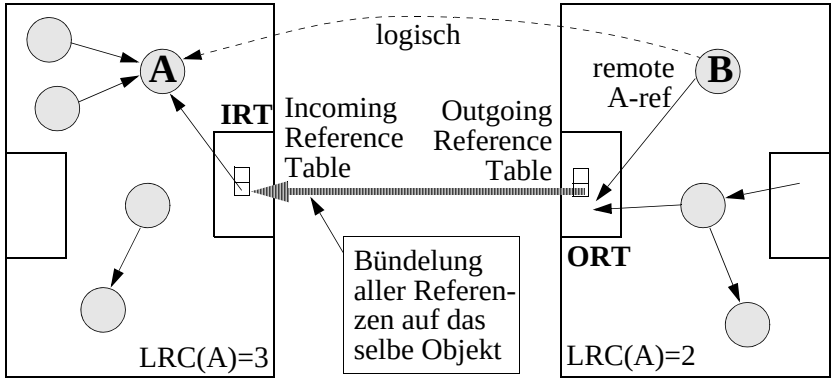
evtl. rekursives Freigeben!

proc collect(r):
if $FIRST_p(r) = \varnothing$ **then** "*r is garbage*"
else send dec(r) **to** $FIRST_p(r)$; **fi**

r wurde lokal erzeugt

- beim *lokalen* Kopieren von r wird lediglich $LRC_p(r)$ erhöht
- wieso wird eine dec-Nachricht gesendet, wenn schon eine r-Referenz existiert; könnte man darauf u.U. verzichten?

Implementierungssicht: Remote-reference-Tabellen



- ORT: enthält *Schattenobjekte* als lokale Stellvertreter ("Proxy") für alle entfernten Objekte
  - beachte: hier ist Σ LRC(A) = 5 = Anzahl A-Referenzen (inkl. bei IRT!),

- IRT / ORT: Die Einträge bzgl. eines bestimmten Objektes (z.B. A) lassen sich als ein *einziges* Objekt, *verteilt* auf mehrere Maschinen auffassen

z.B. weil vom *lokalen Collector* als Garbage erkannt

Idee: Wenn ein Teilobjekt in ORT *gelöscht* wird, dann wird das entsprechende Gegenstück in der IRT gelöscht
(falls es sonst keine Teile in anderen ORT gibt)

→ ORT muss dazu eine Nachricht an IRT senden

- Remote-Referenzen sind evtl. *mehrfach indirekt*
  - u.U mehrfache Adressumsetzung bei *Zugriff* über eine Remote-Referenz, falls der Zugriff tatsächlich entlang der Referenzkette erfogt

LRC: Eigenschaften

- Vergleich zu Verfahren von Lermen/Maurer, Rudalics etc:
  - keine Zusatznachricht bei copy
  - kein Verzögern von copy
  - oft: keine Zusatznachricht bei delete (gelegentlich: Abbau des Referenzbaums)
  - kein Verzögern bei delete
  - FIFO nicht notwendig
  - Gesamtzahl der Nachrichten: genauso viele dec wie copy (wieso?)

Generell gilt: zyklischer Garbage zwischen verschiedenen Objekten lässt sich mit Referenzzählern nicht erkennen!

- Vergleich zu WRC:
  - Zähler einfacher handzuhaben als die Akkumulation von beliebig kleinen Gewichtsfragmenten
  - keine Komplikation bei RW=1

- Nachteil (gegenüber anderen Referenzzähler-Methoden):
  - Objekte besitzen i.Allg. mehrere (Referenz)zähler (angesiedelt z.B. in mehreren ORT-Tabellen) → höherer Speicheraufwand

- Implementierung:
  - typischerweise je einen *lokalen Garbage-Collector* pro Maschine; kann nach anderem Verfahren arbeiten, z.B. mark&sweep (Zyklenerkennung!)
  - Proxyobjekte in der IRT werden dabei als Wurzelobjekte angesehen

- Migration von Objekten lässt sich einfach realisieren
  - nur Zielmaschine und Quellmaschine sind betroffen
  - wie realisiert? (vgl. jeweils Objekte A und B in vorheriger Skizze)
  - wieso einfacher als bei anderen GC-Verfahren?
  - Denkübung: Präzisierung (z.B. Zyklen?)...
    ... und Optimierung (z.B. Verkürzung von Indirektionsketten)

Verteilte Berechnungen

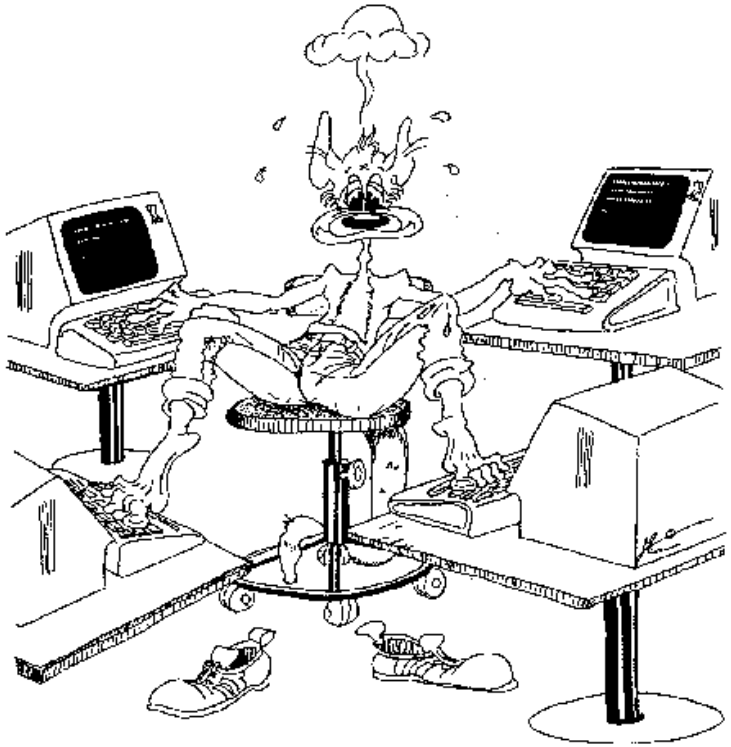


Bild: R. G. Herrtwich, G. Hommel

Verteilte Berechnungen - Vorüberlegungen

- Vert. Programm / Algorithmus:
  - mehrere "Programmtexte" für unterschiedliche Prozesstypen
  - "send", "receive" oder ähnliche Konstrukte für nachrichtenbasierte Kommunikation

- Verteilte Berechnung, "naive" Charakterisierung:
  - Ausführung eines vert. Programms / Algorithmus
  - viele u.U. gleichartige "Instanzen" von Prozessen
  - Kooperation (und Synchronisation) durch Kommunikation
  - mehrere gleichzeitige / parallele Kontrollflüsse ("threads")
  - mehrere Kontrollpunkte (Programmzählerstände)

- Bem.: Es gibt i.Allg. mehrere verschiedene Berechnungen zu einem verteilten Algorithmus! (Nichtdeterminismus)

- Anweisungen, Statements ⇒ atomare Aktionen
  - "atomar": Zwischenzustände von aussen nicht sichtbar (als ob diese keine zeitlichen Ausdehnungen hätten!)
  - auch grössere Einheiten zu atomaren Aktionen zusammenfassen
  - Abstraktion zu "Ereignissen"

- *Visualisierung* einer vert. Berechnung durch *Zeitdiagramme*
  - Vorsicht: es gibt i.Allg. mehrere "äquivalente" Zeitdiagramme!

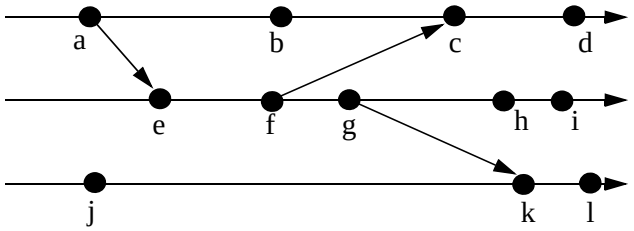
Gummiband-transformation

Verteilte Berechnungen

- Vorübung: Was ist eine sequentielle Berechnung (in einem geeigneten Modell)?

- Ziel: *Formale Definition*, ausgehend von *Zeitdiagrammen*

  - Zeitdiagramme sind bereits ein *Modell* (d.h. Abstraktion von der Realität): "punktförmige" Ereignisse, die bei miteinander kommunizierenden Prozessen stattfinden (Nachrichten als "Querpfeile")



  - interessant: von links nach rechts verlaufende "Kausalitätspfade"
  - Formale Definition möglichst ohne Verwendung graphischer Darstellungen

---

- Menge von Ereignissen E
  - interne Ereignisse; korrespondierende send/receive-Ereignisse

- Direkte Kausalrelation '<:'
  - s <: r für korrespondierende send/receive-Ereignisse s,r
  - a <: b wenn a direkter lokaler Vorgänger von b

- Eigentliche *Kausalrelation* '<' dann als transitive Hülle der direkten Kausalrelation

Die Kausalrelation

- Definiere eine Relation '<' auf der Menge E aller Ereignisse:

"Kleinste" Relation auf E, so dass **x** < **y** wenn:

1) **x** und **y** auf dem gleichen Prozess stattfinden und **x** vor **y** kommt, *oder*

2) **x** ist ein Sendeereignis und **y** ist das korrespondierende Empfangsereignis, *oder*

3) $\exists$ **z** so dass: **x** < **z** $\wedge$ **z** < **y**

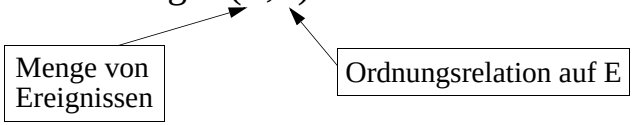
- Wieso ist das eine partielle *Ordnung*?
  - d.h. wieso ist die Relation "gerichtet" und zyklenfrei?
  - oder muss man das (zur Def. von "Berechnung") axiomatisch fordern?

- Relation wird oft als "happened before" bezeichnet
  - eingeführt von Lamport (1978)
  - aber Vorsicht: damit ist nicht direkt eine "zeitliche" Aussage getroffen!

- Interpretationen von e < e':
  - es gibt eine Kausalkette von e nach e'
  - e kann e' beeinflussen
  - e' hängt (potentiell) von e ab
  - e' "weiss" / kennt e

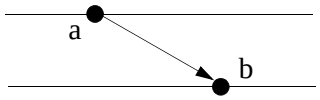
Erster Definitionsversuch

- Verteilte Berechnung = (E,<)

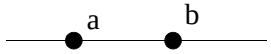


- ist eine verteilte Berechnung also das selbe wie eine Ordnungsrelation?
- zumindest eine spezielle mit mehr Struktur?

Beachte:



und



wären identisch!

⇒ *Prozesse* einführen

- Was sind Prozesse (formal / abstrakt)?

  - Partitioniere E in disjunkte Mengen $E_1,...E_n$

  - Interpretation: E_i = Ereignisse auf Prozess P_i

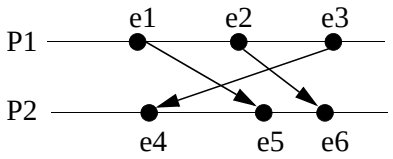
dort *linear* geordnet

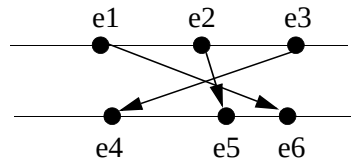
Zweiter Definitionsversuch

- Verteilte Berechnung = $(E_1,...E_n,<)$ mit:

  - alle E_i paarweise disjunkt
  - < ist (irreflexive) Halbordnung auf $E_1 \cup ... \cup E_n$
  - < ist lineare Ordnung auf jedem E_i

- Noch nicht befriedigend:





	e1	e2	e3	e4	e5	e6
e1		x	x	x	x	x
e2			x	x	x	x
e3				x	x	x
e4					x	x
e5						x
e6						

"Spalte" < "Zeile"

gleiche Kausalrelation

⇒ nicht unterscheidbar, obwohl wesentlich verschiedene Diagramme (also verschiedene Berechnungen)!

⇒ Nachrichten einführen

(eindeutige Zuordnung zusammengehöriger send-/receive-Ereignisse)

Dritter Definitionsversuch

- (*n-fach) verteilte Berechnung* (mit asynchroner Nachrichten-Kommunikation) = $(E_1,...E_n,\Gamma,<)$ mit:

  was wäre bei syn. Komm. anders?

1) [*Ereignisse / Prozesse*] Alle E_i sind paarweise disjunkt

2) [*Nachrichten*] Sei $E = E_1 \cup ... \cup E_n$
   Für $\Gamma \subseteq S \times R$ mit $S,R \subseteq E$ und $S \cap R = \varnothing$ gilt:
   - für jedes $s \in S$ gibt es *höchstens* ein $r \in R$ mit $(s,r) \in \Gamma$
   - für jedes $r \in R$ gibt es *genau* ein $s \in S$ mit $(s,r) \in \Gamma$

3) $<$ ist eine lineare Ordnung auf jedem E_i

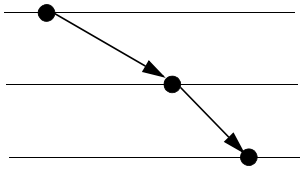
4) $(s,r) \in \Gamma \Rightarrow s < r$

5) $<$ ist eine irreflexive Halbordnung auf E

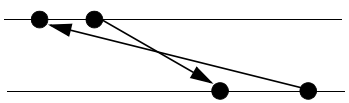
6) $<$ ist die kleinste Relation, die 3) - 5) erfüllt
   (d.h.: es stehen keine weiteren als die dadurch festgelegten Ereignisse in Relation zueinander)

[*Kausalrelation*]

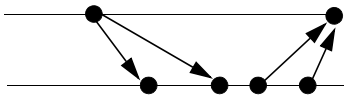
- "Gegenbeispiele":



nicht möglich wegen (2)



nicht möglich wegen (5)



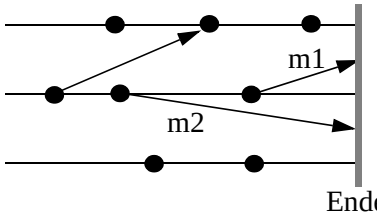
nicht möglich wegen (2)

Bemerkungen zur Definition

- Die $s \in S$ heissen *Sende-*, die $r \in R$ *Empfangsereignisse*
  - die anderen Ereignisse werden *interne Ereignisse* genannt

- *Darstellung* einer so definierten verteilten Berechnung ist mit *Zeitdiagrammen* möglich
  - E_i als Prozesslinie mit Ereignissen
  - Γ als Pfeile
  - $<$ garantiert Zyklenfreiheit

- Definition erlaubt (wegen "höchstens" in Punkt 2) die Modellierung von *In-transit-Nachrichten*:



  - die beiden Nachrichten m1 und m2 sind nie angekommen
  - mögliche Interpretationen:
    - sind verlorengegangen
    - waren bei Berechnungsende noch unterwegs

- Kann man Einzelprozesse als *Folgen von Ereignissen* auffassen?

- Ergibt sich bei einer vert. Berechnung aus einem *einzigen* Prozess eine (wie üblich definierte) *sequentielle* Berechnung?

- Wieso ist diese Definition so "statisch" (statt einer Definition über *Folgen* von Ereignissen oder Zuständen)?

  Vorläufige Antwort:
  - wir haben den Begriff "globaler Zustand" noch nicht definiert
  - Folgen sind nicht eindeutig bestimmt

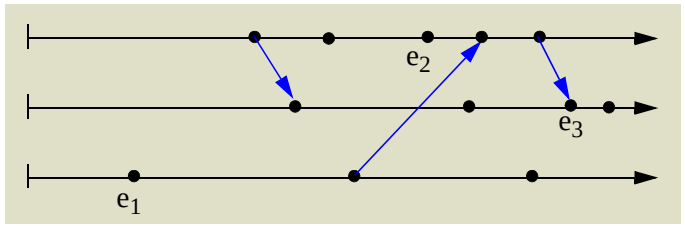
Zeitdiagramme

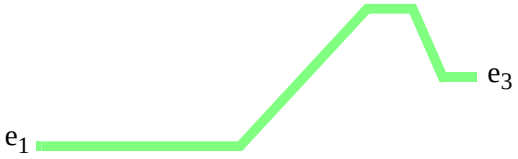
- Theorem [*Kausalkette*]:

  In einem Zeitdiagramm gilt für je zwei Ereignisse e, e' die Relation e < e' genau dann, wenn es einen Pfad von e nach e' gibt

  Nachrichtenpfeile + Teilstücke auf Prozessachsen von links nach rechts

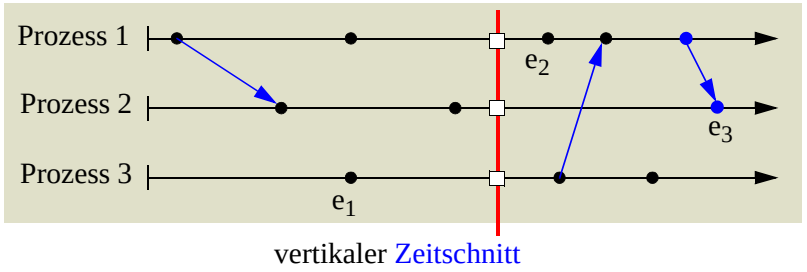
- Bew.: induktiv, Transitivität, "kann beeinflussen",...





- Beispiel: $e_1 < e_3$, aber *nicht* $e_1 < e_2$

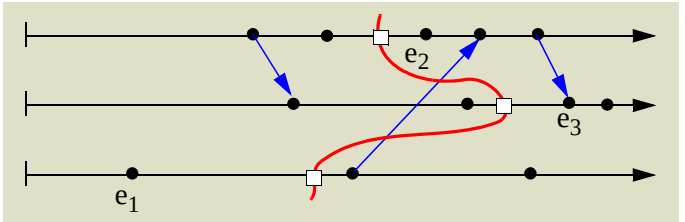
Gummibandtransformation



vertikaler Zeitschnitt

Diagramme sind daher äquivalent

Ein anderes Bild der *gleichen* Berechnung:



- Gummibandtransformation
  - abstrahiert von metrischer Struktur ("Zeit")
  - Stauchen und Dehnen der Prozessachsen
  - lässt topologische Struktur invariant (Kausalitätspfade von links nach rechts)
  - vertikale Zeitschnitte werden "verbogen" (umgekehrt ist interessant, wann sich krumme Zeitschnitte "geradebiegen" lassen!)

  Nachrichtenpfeile sollen aber nie rückwärts laufen

- Bei Fehlen einer absoluten Zeit sind alle Diagramme, die sich so deformieren lassen, gleich "richtig" (äquivalent)
  - kann man so immer eine schiefe Beobachtungslinie "senkrecht biegen"?
  - nein: leider nur, wenn die Beobachtung kausaltreu ist!

Globale Zustände und Endzustände

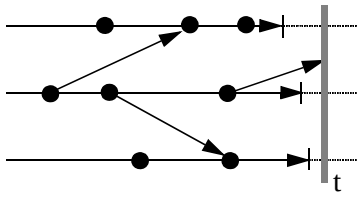
- Was ist ein *globaler Zustand* einer vert. Berechnung?
  - Charakteristikum verteilter Systeme: dieser ist auf die Prozesse verteilt und nicht unmittelbar ("gleichzeitig") zugreifbar!

- Damit vielleicht: *Ablauf einer vert. Berechnung* als Folge solcher Zustände definieren?
  - geht nicht so einfach, wie wir noch sehen werden...

- *Lokaler Zustand* eines Prozesses ist klar
  - Def. Zustandsraum (Kreuzprodukt Variablenzustände...) klar
  - Zustand zwischen zwei atomaren Aktionen / Ereignissen konstant (d.h. Ereignisse transformieren lokale Zustände)

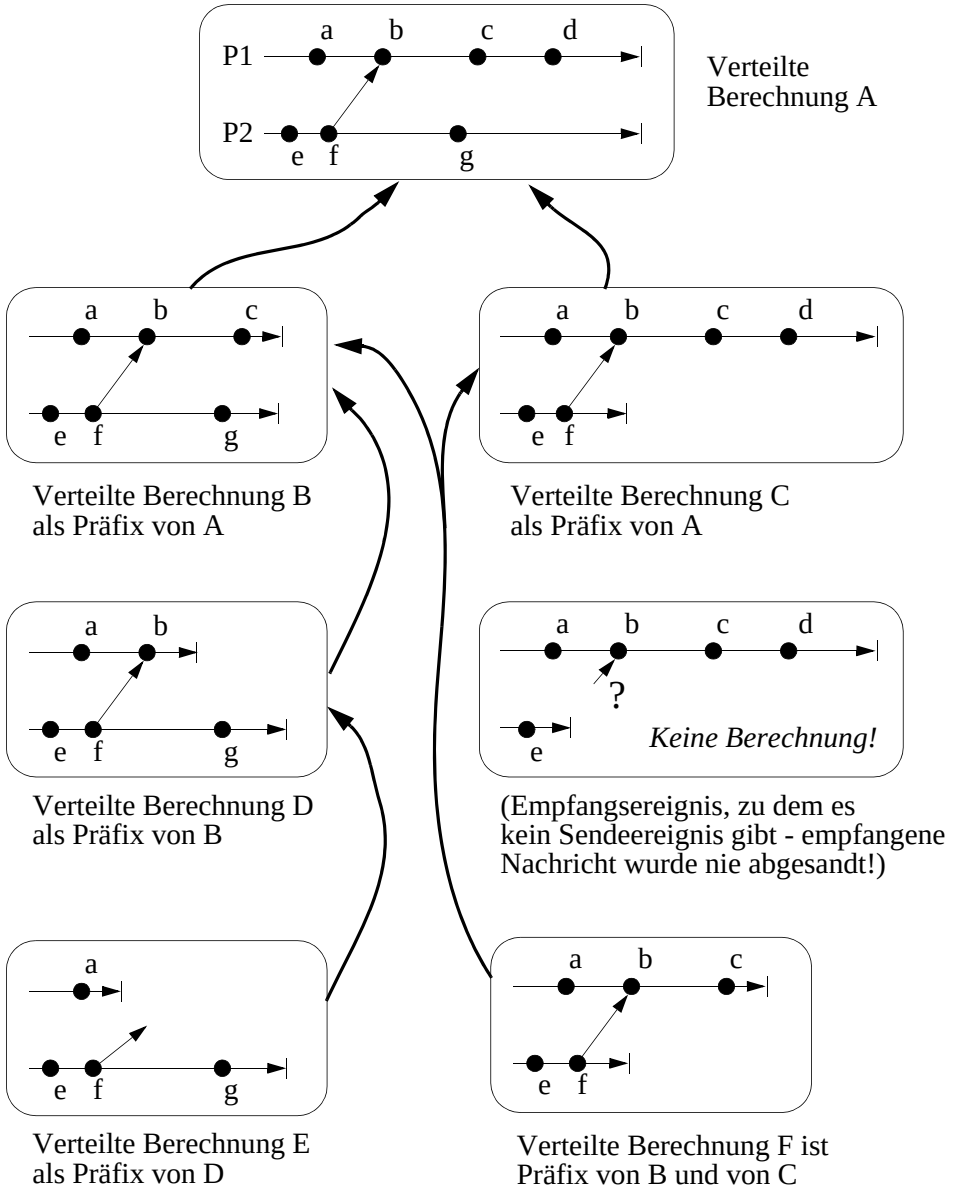
- *Globaler Zustandsraum*:
  - Kreuzprodukt lokaler Zustandsräume
  - Menge von Nachrichten ("in transit")

- Beachte: "Globaler Zustand" lässt sich zunächst nur für einen bestimmten (gemeinsamen) *Zeitpunkt* definieren!



- benutze hilfsweise den *Endzustand* der verteilten Berechnung
- dieser lässt sich am Ende einfach einsammeln (wird nicht mehr verändert)
- Zwischenzustände lassen sich dann evtl. als Endzustände geeigneter Präfixberechnungen definieren

Präfixe von Berechnungen



Präfix-Berechnungen

- Gegeben sei eine Berechnung B = $(E_1,...E_n,\Gamma,<)$;
  wann ist eine Berechnung B' eine Präfixberechnung von B?
  - E_i' ist eine *Teilmenge* von E_i
  - Γ' und <' sind *Einschränkungen* von Γ und < auf diese Teilmengen

- Wir müssen aber noch fordern, dass E' *linksabgeschlossen* bzgl. der Kausalrelation < ist:

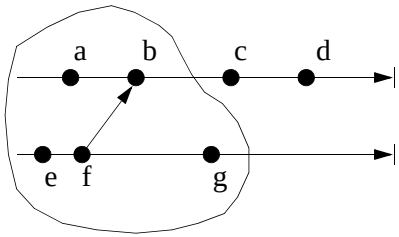
$$\forall\, x \in E', y \in E:\quad y < x \;\Rightarrow\; y \in E'$$

  $\Rightarrow$ lokale Berechnungen sind wirkliche Anfangsstücke

  $\Rightarrow$ unmögliche Diagramme, wo Nachrichten empfangen aber nicht gesendet werden, sind "automatisch" ausgeschlossen (dies wurde allerdings schon durch Γ' als Einschränkungen von Γ garantiert)

- Zu einer gegebenen Berechnung B = $(E_1,...E_n,\Gamma,<)$ ist eine Präfixberechnung B' eindeutig bestimmt duch:
  - (1) die Menge aller ihrer Ereignisse E', *oder*
  - (2) die Menge der jeweils lokal letzten Ereignisse
    ("Front" von E' ist eine kompaktere Repräsentation!)

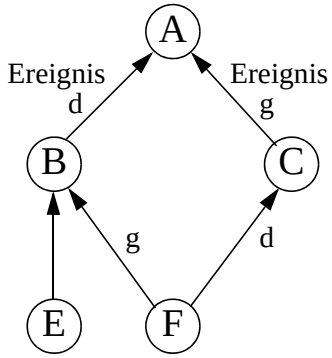
Beispiel:



(1): {a, b, e, f, g}

(2): {b, g}

Präfix-Relation



- Präfixdiagramm
  - kein (Wurzel)baum, aber
  - *gerichtet* und *zyklenfrei*
- Präfixrelation ist *transitiv*
- $\Rightarrow$ Präfixrelation ist eine *Halbordnung*!

- Frage: Entstand "im Verlaufe der Berechnung" A aus B oder aus C?
  - d.h.: durchlief die Berechnung von A vorher den Endzustand von B oder den von C als Zwischenzustand?
  - äquivalent: geschah Ereignis d vor g oder g vor d?
  - beachte: *beides* ist unmöglich (wenn d und g nicht "gleichzeitig")

- Konsequenz:
  - direkter Vorgänger i.Allg. indefinit
  - verteilte Berechnung ist *keine Folge* von Zuständen, sondern - notgedrungen - eine geeignet definierte Halbordnung!

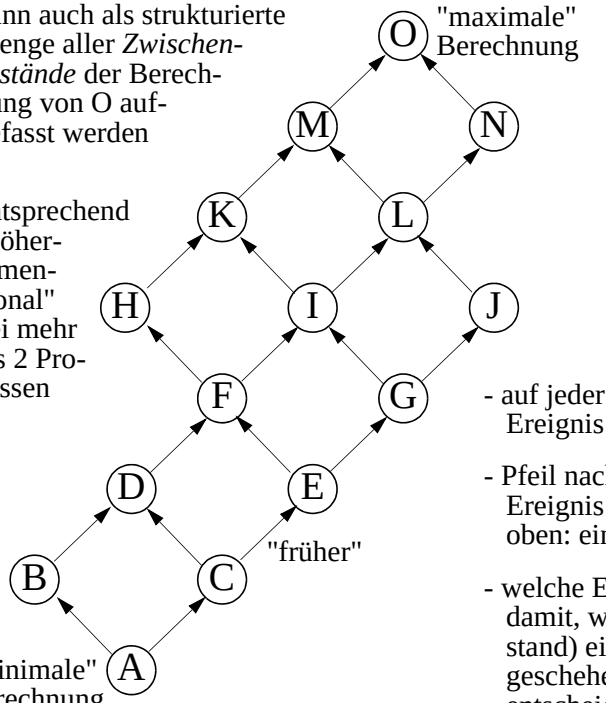
- Vgl. dies mit *sequentiellen* Berechnungen: Hier wird *jeder* Präfix einer Berechnung (genauer: dessen Endzustand) durchlaufen! ($\rightarrow$ Def. als Folge möglich)

Der Präfix-Verband

- *Verband* von Mengen "geschehener" Ereignisse
  - zur Wiederholung: was ist ein Verband als mathematische Struktur?

- kann auch als strukturierte Menge aller *Zwischenzustände* der Berechnung von O aufgefasst werden

- entsprechend "höherdimensional" bei mehr als 2 Prozessen

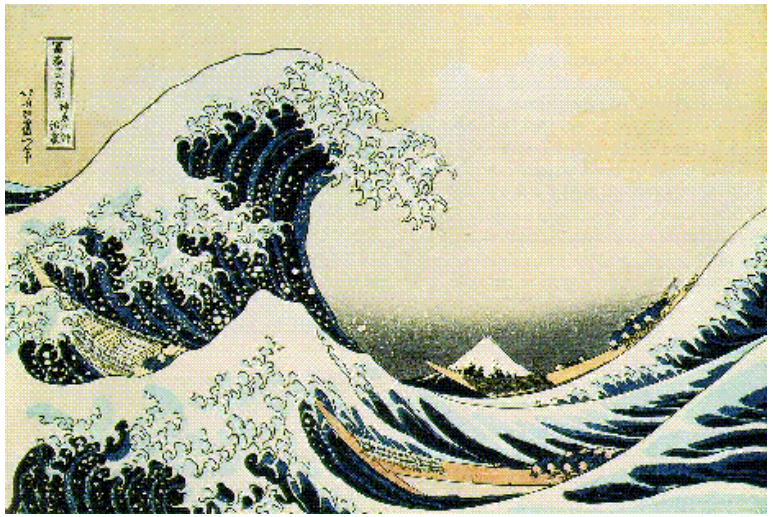


an dieser Stelle würde ein "unmögliches" Diagramm stehen

- auf jeder Ebene kommt ein Ereignis hinzu
- Pfeil nach rechts oben: ein Ereignis von P1; nach links oben: ein Ereignis von P2
- welche Ereignismenge (und damit, welcher Zwischenzustand) einer Ebene *wirklich* geschehen ist, lässt sich nicht entscheiden → keine vernünftige Frage!

- Ein Zwischenzustand hat also i.Allg. mehrere direkte Vorgänger- und Nachfolgezustände!
- Berechnung bewegt sich in diffuser Weise in diesem Zustandsraum von unten nach oben
  - von den Ecken her ausgefranstes n-dimensionales Gitter, mehr dazu später!

Wellenalgorithmen



Katsushika Hokusai (1760-1849): Die grosse Woge, Metropolitan Museum of Art, New York

This well-known masterpiece shows Mt. Fuji behind raging waves off the seacoast. Hokusai created "Mt. Fuji Off Kanagawa" (popularly known in the West as "The Wave") as part of his subscription series, "Thirty-Six Views of Mt. Fuji," completed between 1826 and 1833. This is one of the best-known Japanese woodblock prints, and with others of this period inspired the entire French Impressionist school. By making Mt. Fuji only rather small in the background the artist expresses, in a novel way, the elemental power of nature.

Wellenalgorithmen

Häufige Probleme bei verteilten Algorithmen / Systemen:

- *Broadcast* einer Information
- Globale *Synchronisation* zwischen Prozessen
- *Triggern eines Ereignisses* in jedem Prozess
- *Einsammeln* von verteilten Daten

Trennung von Phasen

$\Rightarrow$ *Wellenalgorithmen*

- *Alle* Prozesse müssen sich beteiligen
- *Basisalgorithmen* ("Bausteine") für andere Algorithmen (wie wechselseitiger Ausschluss, Terminierungerkennung, Election...)

Abstraktere Definition:

Algo. ist evtl. nicht-deterministisch!

Wellenalgorithmus, wenn für jede seiner Berechnung gilt:

1. Berechnung enthält ein *init*-Ereignis
2. Alle Prozesse besitzen ein *visit*-Ereignis
3. Berechnung enthält ein *conclude*-Ereignis

Und es gilt folgendes für alle visit-Ereignisse:

a) init $\leq$ visit
b) visit $\leq$ conclude

$(\Rightarrow$ init $\leq$ conclude)

Wellenalgorithmen (2)

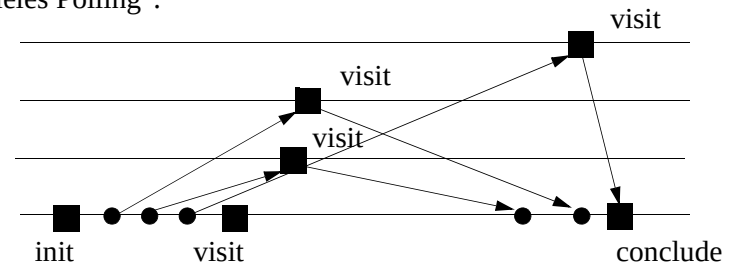
Aus a): Über die Kausalketten lässt sich *Information* vom Initiator an alle Prozesse *verteilen*

Aus b): - Nach conclude wurde jeder besucht ($\rightarrow$ Terminierung!)
- Über die Kausalketten lässt sich *Information* von allen Prozessen *einsammeln*

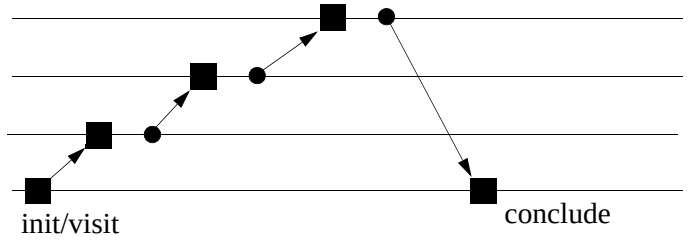
Bem.: - init und conclude oft im selben Prozess ("Initiator")
- init oder conclude kann mit visit verschmelzen (daher '$\leq$' statt '$<$')

Beispiele:

"paralleles Polling":



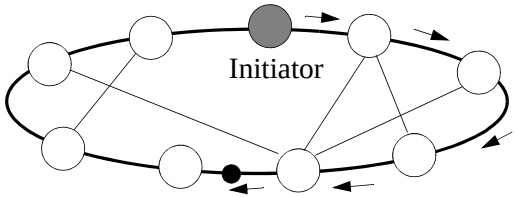
"Ring":



Einige Wellenalgorithmen

- init-, visit-, conclude-Ereignisse jeweils sinnvoll festlegen!

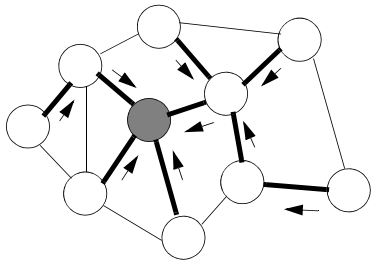
1.) *Ring* / *Hamiltonscher Zyklus* mit umlaufenden Token



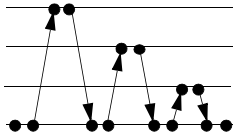
- "logischer" Ring genügt!
- in einem zusammenhängenden ungerichteten Graphen kann ein logischer Ring immer gefunden werden, indem man einen Spannbaum "umfährt"

2.) *Spannbaum*

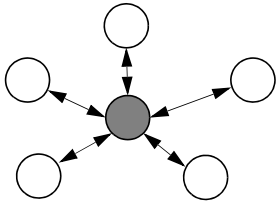
- an den Blättern reflektierte Welle (vereinfachter Echo-Algorithmus: flooding mit rek. acknowledgement)
- bei nicht-entartetem Spannbaum sind viele Nachrichten parallel unterwegs



3.) *(logischer) Stern*



sequentielles Polling



- "Polling" entweder sequentiell (jeweils höchstens eine Nachricht unterwegs) oder parallel

---

4.) *Echo-Algorithmus ist ein Wellenalgorithmus*

- visit-Ereignis entweder erster Erhalt eines Explorers oder Senden des Echos
- definiert so sogar zwei verschiedene "parallele" Wellen bzw. "Halbwellen"!

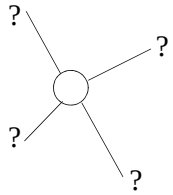
Eigenschaften von Wellenalgorithmen

wodurch ist eigentlich der suggestive Name gerechtfertigt?

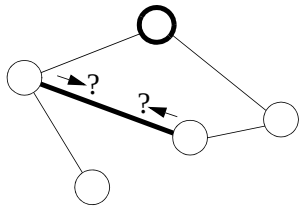
Satz: Es werden *mindestens n-1 Nachrichten* benötigt

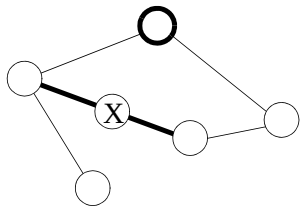
- Begründung: Da jedes visit-Ereignis vom init-Ereignis kausal abhängig ist, muss jeder andere Prozess mindestens eine Nachricht empfangen
- mindestens n Nachrichten, falls conclude und init auf gleichem Prozess stattfinden (dann muss auch der Initiator eine Nachricht empfangen)

Satz: Ohne Kenntnis der Nachbaridentitäten werden *mindestens e Nachrichten* benötigt



Bew.: Über jede Kante muss eine Nachricht gesendet werden, wenn die Identität der Nachbarn unbekannt ist:





- würde man im linken Szenario die Kante nicht durchlaufen, dann würde im rechten Szenario Knoten X nicht erreicht und er könnte kein von init abhängiges visit-Ereignis ausführen

Wellenalgorithmen und Spannbäume

Satz: Die Kanten, über die ein Knoten (≠Initiator) erstmals eine Nachricht erhielt, bilden einen *Spannbaum*

(Voraussetzung: der Algorithmus wird an einer einzigen Stelle initiiert)

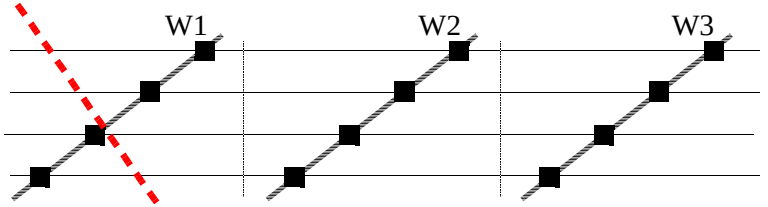
Beweis:

- Jeder Nicht-Initiator erhält mindestens eine Nachricht
- Es gibt also n-1 solche ersten Kanten
- Wir müssten noch einsehen:
  - die Menge der entsprechenden Kanten ist zyklenfrei,
  - *oder* die Menge dieser Kanten ist zusammenhängend

  } Denkübung: wieso?
- Damit und mit n-1 folgt die Baumeigenschaft

- man überlege sich, wieso der Beweis falsch wird, wenn ein Wellenalgorithmus an mehreren Stellen unabhängig initiiert wird!
- wir kennen solche Spannbäume bereits vom Echo-Algorithmus!

Folgen von Wellen



- Wellennummer j kann mit jeder Nachricht mitgeführt werden.

Start:

```
j := 1 (* Wellennummer *)
x := lokale Berechnung
init(j)  (* evtl. broadcast von x *)
```

j-te Iteration

Ersetze conclude durch:

```
j := j+1
if (* noch nicht fertig *)
   x := lokale Berechnung
   init(j)  (* evtl. broadcast von x *)
fi
```

Satz: Zu jedem globalen Zeitpunkt unterscheiden sich die Wellennummern zweier Prozesse um höchstens 1

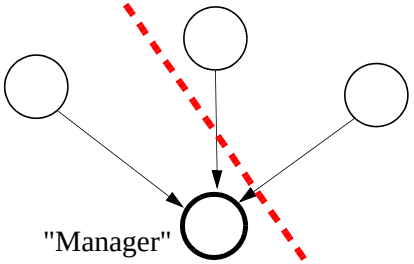
Beachte zum Beweis den Zusammenhang zwischen Zeit und Kausalität

==> Phasensynchronisation der Prozesse möglich

Bem.: Verschiedene unabhängige Wellen (evtl. verschiedener Initiatoren) können sich in transparenter Weise "parallel" überlagern, wenn eindeutige Identitäten mitgeführt werden

Halbwellen

- Abschwächung des Wellenbegriffs, z.B. *kein init*:
  --> *kontrahierende* Halbwelle



- Bsp.: Prozesse melden unaufgefordert ihre einzusammelnden Werte einem zentralen Prozess (*Stern*) bzw. allen Prozessen (*vollst. Graph*)
- Analog auch für *Bäume*, wo die Blätter "spontan" ihr Ergebnis weiter nach innen melden

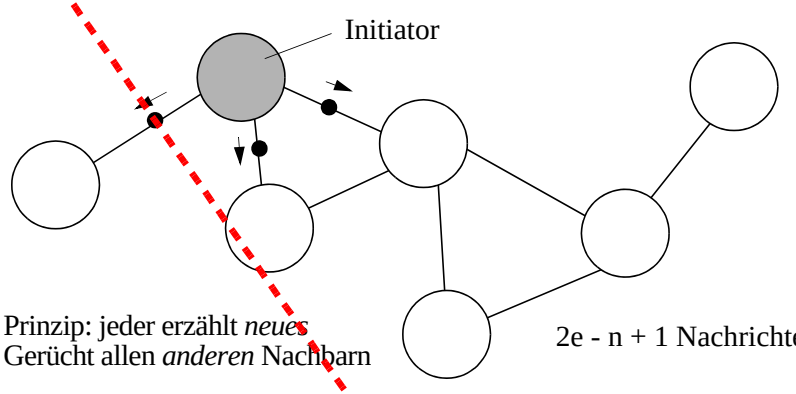
- Manager bzw. Wurzel des Baumes kann conclude durchführen, wenn alle Nachbarknoten geantwortet haben (also ihr visit ausgeführt haben)
- Andere Auffassung: es gibt kein *eindeutiges* init, sondern i.Allg. mehrere unabhängige (--> "multisource-Algorithmus")

- Entsprechend: *kein conclude*: --> *expandierende* Halbwelle
  - Es gibt keinen Prozess, der von der *Terminierung* des Algorithmus erfährt
  - Beispiel: *flooding*-Algorithmus (entspricht erster Halbwelle des Echo-Algorithmus)

Der Flooding-Algorithmus als Halbwelle

- Algorithmus kennen wir bereits!

Welle ohne conclude



Prinzip: jeder erzählt *neues* Gerücht allen *anderen* Nachbarn

2e - n + 1 Nachrichten

visit:

```
V:  {Eine Informationsnachricht kommt an}
    if not informed then
       informed := true;
       send <info> to all other neighbors;
    fi
```

informed = false

init:

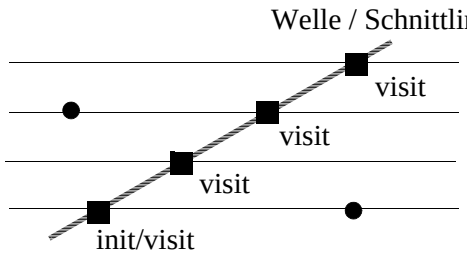
```
I:  {not informed}
    informed := true;
    send <info> to all neighbors;
```

Aktion I nur beim einzigen Initiator

- auch für solche expandierenden Halbwellen gilt (bei einem einzigen Initiator), dass die "ersten Kanten" einen *spannenden Wurzelbaum* bilden
- diesen Baum kann man benutzen, um (entsprechend einer kontrahierenden Halbwelle) *Acknowledgements* zum Initiator zu senden (dazu müssen die Blätter des Baumes aber erkennen, dass sie Blätter sind - wie geht das?)
- man erhält nach dieser Idee den *Echo-Algorithmus*!

Wellen und Schnittlinien

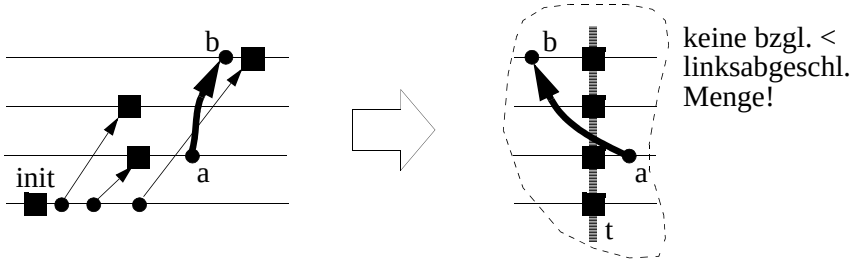
- Verbinden der visit-Ereignisse zu einer *Schnittlinie*:



Trennt die Menge der Ereignisse in *Vergangenheit* und *Zukunft*

oBdA: "gerade" Schnittlinie
(→ Gummibandtransf.)

- Falls ein Wellenalgorithmus einer anderen verteilten Anwendung überlagert wird: Lässt sich dann die Schnittlinie immer *senkrecht* zeichnen (Gummibandtransformation), so dass keine Nachricht "echt" rückwärts läuft?



keine bzgl. < linksabgeschl. Menge!

- Beispiel: zu dem dadurch festgelegten "globalen Zeitpunkt" t wäre eine Anwendungsnachricht zwar angekommen, aber noch nicht abgesendet!

  ⇒ die Welle würde ein unmögliches ("inkonsistentes") Bild liefern

  ⇒ Visit-Ereignisse lassen sich (hier) nicht als "virtuell gleichzeitg" ansehen

- Wichtige Fragen: Unter welchen Umständen definieren die visit-Ereignisse einen "Schnitt", der als *senkrechte* Linie aufgefasst werden kann? Lässt sich das erzwingen?
  (damit hätten wir so etwas wie globale Zeit → später)

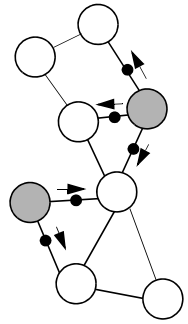
Virtuell gleichzeitiges Markieren

- Als Anwendung eines Wellenalgorithmus (≈ flooding)
- Voraussetzung hier: FIFO-Kanäle (d.h. keine Überholungen)

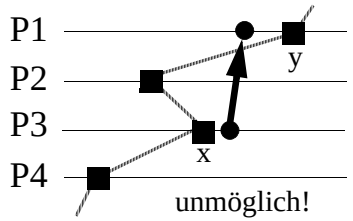
```
I:  {not marked}
    marked := true;   //  = init = visit
    send <marker> to all neighbors;
```

```
V:  {Eine Marker-Nachricht kommt an}
    if not marked then
       marked := true;   // = visit
       send <marker> to all neighbors;
    fi
                                  !
```

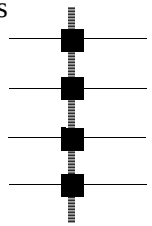
mehrere Initiatoren sind zulässig!



- Über *jede* Kante läuft in beide Richtungen ein Marker
- Eine andere Nachricht ("Basisnachricht"), die von einem markierten Knoten gesendet wird, kommt erst dann an, wenn der Empfänger auch bereits markiert ist



Es ist daher stets rechts nebenstehende *Sicht* als Gummibandtransformation *möglich*, wo Basisnachrichten nicht rückwärts verlaufen und alle visits simultan sind (*wieso?*)



Eine "rückwärts" über den Schnitt laufende Basisnachricht kann es nicht geben: Wenn P3 und P1 benachbart sind, dann wurde bei x ein Marker an P1 gesendet, der nicht (wie von der gezeichneten Nachricht) überholt werden kann

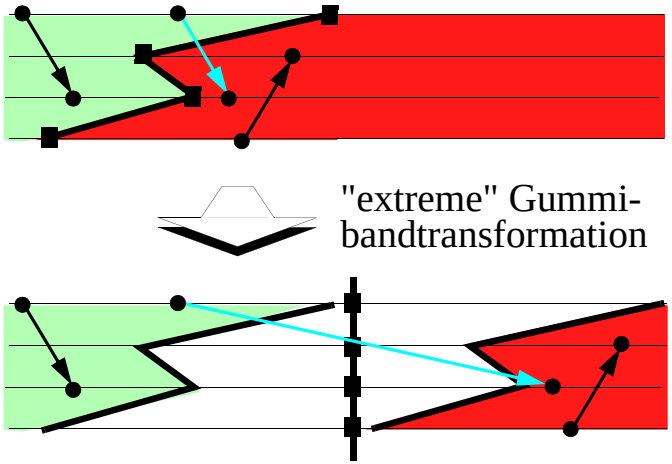
Fragen: - geht virtuell gleichzeitiges Markieren auch ohne FIFO-Kanäle?
(FIFO abschwächen, FIFO erzwingen, FIFO "simulieren"...)
- geht das auch mit weniger als 2e Nachrichten?

Vertikale Schnittlinien?

- Voraussetzung: Keine Nachricht läuft von der "Zukunft" in die "Vergangenheit" einer Schnittlinie
  - solche Schnittlinien heissen *konsistent*
    (linke Hälfte ist dann linksabgeschlossen bzgl. der Kausalrelation, d.h. linke Hälfte ist eine Präfixberechnung)

- Dann lässt sich diese Schnittlinie *vertikal* zeichnen, ohne dass dabei Nachrichten von rechts nach links laufen
  - als hätte die zugehörige Welle alle Prozesse gleichzeitig besucht!
  - offenbar nützlich für Terminierungserkennung, Sicherungspunkte...!

- "Konstruktiver" Beweis: Auseinanderschneiden entlang der Schnittlinie und den rechten Teil "ganz" über den weitesten rechts liegenden Punkt des linken Teils hinausschieben



- Zerschnittene Nachrichten reparieren ("verlängern")
- Schnittereignisse in die Lücke senkrecht zueinander legen

Anwendung von Wellenalgorithmen

- Als "Unterprogramm" innerhalb anderer Anwendungen, z.B.
  - Broadcast einer Information an alle Prozesse
  - Einsammeln verteilter Daten
  - Traversieren aller Prozese eines Prozessgraphen
  - Berechnung einer globalen Funktion, deren Parameter auf alle Prozesse verteilt sind (z.B. globales Minimum)
  - ...

- Abstrakt: um Schnitte durch ein Zeitdiagramm zu legen
  (Problem jedoch evtl.: Konsistenz des Schnittes feststellen / erzwingen)
  - Terminierungserkennung
  - Schnappschuss
  - ...

- Wellenalgorithmen sind typische *Basisalgorithmen*
  - verrichten *Dienste* einer niedrigeren Schicht
  - sind *Bausteine* für komplexere Verfahren
  - bestimmen oft qualitative Eigenschaften wie Nachrichten- oder Zeitkomplexität der aufgesetzten Verfahren entscheidend mit
  - lassen sich (sofern die Rahmenbedingungen stimmen) gegeneinander austauschen

- Es gibt viele verschiedene Wellenalgorithmen
  - für die verschiedensten Topologien
  - mit unterschiedlichen Voraussetzungen
  - mit unterschiedlichen Qualitätseigenschaften

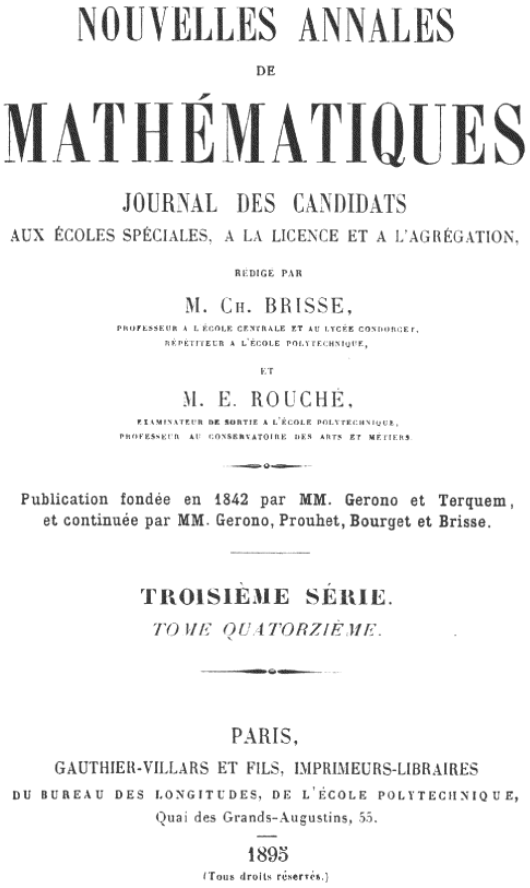
Sequentielle Traversierungsverfahren

- Unterklasse der Wellenalgorithmen (auf ungerichteten und zusammenhängenden Graphen) mit:
  - einem einzigen Initiator (bei dem init und conclude stattfindet)
  - *totaler Ordnung* auf allen visit-Ereignissen

- Interpretation: Ein *Token* wandert durch alle Prozesse und kehrt zum Initiator zurück
  - Visit-Ereignis: erstmalige Ankunft (oder Weiterreichen) des Tokens
- Relativ hohe Zeitkomplexität, da keine wesentliche Parallelität

---

- Bekannte einfache Verfahren für *spezielle Topologien*:
  - *Stern* (bzw. vollst. Graph): sequentielles "polling"
  - *Ring*
  - *Baum* (bzw. Spannbaum) traversieren ("depth first")
  - n-dimensionales *Gitter*: Verallgemeinerung des "ebenenweise" Durchlaufens eines 3-dimensionale Gitters (einen Schritt in der Dimension k+1, wenn k-dimensionales Untergitter durchlaufen...)
  - n-dimensionale *Hyperwürfel*: Traversiere ("rekursiv") einen n-1-dim. Hyperwürfel bis auf den letzten Schritt, gehe in Richtung der n-ten Dimension und traversiere dort den n-1-dim. Hyperwürfel...
  - Hamiltonsche Graphen (z.B. auch Ring und Hyperwürfel als Sonderfall)



NOUVELLES ANNALES

DE

MATHÉMATIQUES

JOURNAL DES CANDIDATS

AUX ÉCOLES SPÉCIALES, A LA LICENCE ET A L'AGRÉGATION,

RÉDIGÉ PAR

M. Ch. BRISSE,

PROFESSEUR A L'ÉCOLE CENTRALE ET AU LYCÉE CONDORCET,
RÉPÉTITEUR A L'ÉCOLE POLYTECHNIQUE,

ET

M. E. ROUCHÉ,

EXAMINATEUR DE SORTIE A L'ÉCOLE POLYTECHNIQUE,
PROFESSEUR AU CONSERVATOIRE DES ARTS ET MÉTIERS.

Publication fondée en 1842 par MM. Gerono et Terquem,
et continuée par MM. Gerono, Prouhet, Bourget et Brisse.

TROISIÈME SÉRIE.

TOME QUATORZIÈME.

PARIS,

GAUTHIER-VILLARS ET FILS, IMPRIMEURS-LIBRAIRES

DU BUREAU DES LONGITUDES, DE L'ÉCOLE POLYTECHNIQUE,

Quai des Grands-Augustins, 55.

1895

Nouvelles Annales de Mathématiques 14 (1895)

LE PROBLÈME DES LABYRINTHES;

PAR M. G. TARRY.

Tout labyrinthe peut être parcouru en une seule course, en passant deux fois en sens contraire par chacune des allées, sans qu'il soit nécessaire d'en connaître le plan.

Pour résoudre ce problème, il suffit d'observer cette règle unique :

Ne reprendre l'allée initiale qui a conduit à un carrefour pour la première fois que lorsqu'on ne peut pas faire autrement.

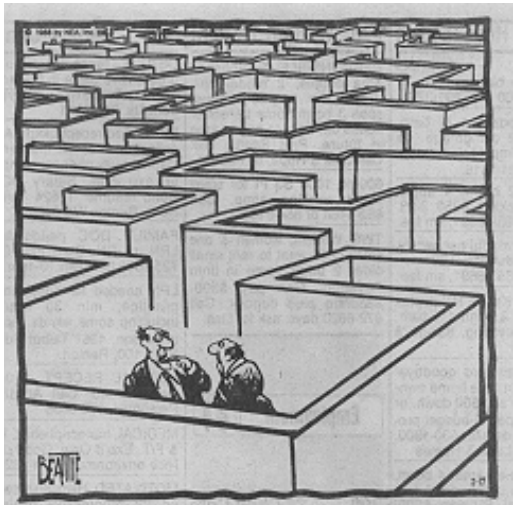
. . .

En suivant cette marche pratique, un voyageur perdu dans un labyrinthe ou dans des catacombes, retrouvera forcément l'entrée avant d'avoir parcouru toutes les allées et sans passer plus de deux fois par la même allée.

Ce qui démontre qu'un labyrinthe n'est jamais inextricable, et que le meilleur fil d'Ariane est le fil du raisonnement.

Gaston Tarry (1843-1913) studied mathematics in Paris. He spent his whole career working in Algeria. He was interested in geometry and published numerous articles in various journals from 1882 until his death. He did extensive work on magic squares and on number theory.

Labyrinth



"The exit? Sure... take a right, then left, left again... no wait... a *right*, then... no, wait..."

Römisches Mosaik (aus Loig bei Salzburg) illustriert die Geschichte von Theseus und Minotaurus im Labyrinth

Labyrinth (2)

Was ist von der klassischen Regel zu halten, immer mit der einen Hand die Wand entlang tasten?

- findet man dann immer wieder heraus?
- findet man den Goldtopf, der irgendwo im Labyrinth versteckt ist?

Der Algorithmus von Tarry

Traversieren *beliebiger* (zusammenhängender, unger.) Graphen
(Nachbaridentitäten der Knoten brauchen nicht bekannt zu sein!)

- Zwei Regeln zum Propagieren eines Tokens, die - *wenn immer möglich* - von einem Prozess angewendet werden:

R1: Ein Prozess schickt das Token niemals zwei Mal über die gleiche Kante *(Wo sagt Tarry das?)*

R2: Ein Prozess (≠ Initiator) schickt das Token erst dann an denjenigen Prozess zurück, von dem er es erstmalig erhielt, wenn er keine andere unbenutzte Kante mehr hat

→ Algorithmus ist nichtdeterministisch!

- Beh.: Algorithmus terminiert

  Bew.: Max. 2e Mal wird das Token versendet...

- Beh.: Wenn der Algorithmus terminiert ist, ist das Token bei jedem Prozess vorbeigekommen und wieder zum Initiator zurückgekehrt

  → Tarry-Algorithmus ist ein Traversierungsalgorithmus

  → "Ziel" kann auch eine andere Stelle als der Eingang sein

  Bew.: ...

Tarrys Verfahren ist ein Wellenalgorithmus

(1) Terminierung ⇒ Token ist beim Initiator

Beweis: Für jeden Nicht-Initiator p gilt: Wenn p das Token hat, dann hat p das Token k-Mal (auf jeweils unterschiedlichen Kanälen, Regel R1) erhalten und auf k-1 (jeweils unterschiedlichen Kanälen) gesendet
⇒ es gibt noch mindestens einen unbenutzten "Ausgangskanal"
⇒ das Token bleibt nicht bei p.

(2) Alle Kanäle des Initiators werden in beiden Richtungen genau 1 Mal vom Token durchlaufen

Beweis: Für jeden "Ausgangskanal" klar, sonst wäre der Algorithmus nicht terminiert: Nach (1) bleibt das Token nicht bei einem andereren Prozess stecken. Das Token muss genauso oft zum Initiator zurückgekehrt sein, wie es von dort weggeschickt wurde, und zwar über jeweils andere Kanäle (Regel R1) ⇒ Jeder "Eingangskanal" des Inititators wurde benutzt.

(3) Für jeden besuchten Prozess p gilt: Alle Kanäle von p wurden in beide Richtungen durchlaufen

Beweis durch Widerspruch: Betrachte den ersten (= "frühesten") besuchten Prozess p, für den dies nicht gilt. Nach (2) ist dies nicht der Initiator. Sei Vater(x) derjenige Prozess, von dem x erstmalig das Token erhielt. Für Vater(p) gilt nach Wahl von p, dass alle Kanäle in beide Richtungen durchlaufen wurden. ⇒ p hat das Token an Vater(p) gesendet. Wegen Regel R2 hat daher p das Token auf allen anderen (Ausgangs)kanälen gesendet. Das Token musste dazu aber genauso oft auf jeweils anderen Kanälen (R1) zu p zurückkommen. ⇒ Alle Eingangskanäle wurden ebenfalls benutzt.

(4) Alle Prozesse wurden besucht

Beweis: Andernfalls gäbe es eine Kante von einem besuchten Prozess q zu einem unbesuchten Prozess p, da der Graph zusammenhängend ist. Dies steht aber im Widerspruch zu (3), da diese Kante vom Token durchlaufen wurde.

Die Welleneigenschaft ergibt sich i.w. aus (1) und (4)

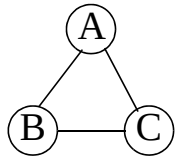
Depth-first-Algorithmen und Spannbäume

Klassischer Depth-first-search-Algorithmus:

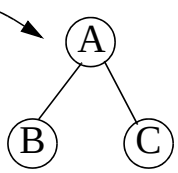
- Token geht erst dann zurück, wenn alles andere "abgegrast" ist
- Token kehrt um, sobald es auf einen bereits besuchten Knoten trifft
- jede Kante wird in jede Richtung genau 1 Mal durchlaufen
- Tarry-Algorithmus lässt sich zu Depth-first-Traversierung spezialisieren

⇒ 2e Nachrichten, 2e Zeitkomplexität

- Depth-first-search-Algorithmen liefern beim Durchlaufen eines Graphen einen Spannbaum (Wurzel = Initiator)
(wie jeder Wellenalgorithmus!)



wieso ist der rechte Baum *kein* von einem Depth-first-search-Algorithmus erzeugter Spannbaum des linken Graphen mit Initiator A?



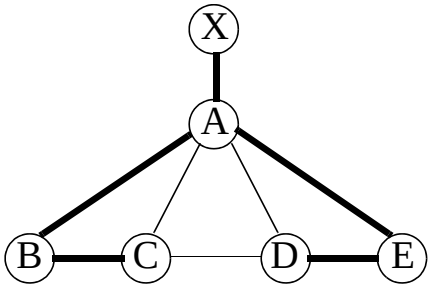
- Eine Charakterisierung solcher Spannbäume:

Jede Kante des Graphen, die keine Kante des Spannbaums ist, verbindet zwei Knoten, die auf dem gleichen Ast liegen

Weg von der Wurzel zu einem Blatt

- wieso?
- kann der rechte Spannbaum vom Tarry-Algorithmus erzeugt werden?

Tarry-Algorithmus und Spannbäume



- Der fett eingezeichnete Baum (mit Wurzel X) ist kein Depth-first-Spannbaum (wegen der Kante CD)

- Dennoch kann der Baum mit dem Tarry-Algorithmus über folgende Traversierung erzeugt werden:
  X, A, B, C, A, E, D, A, C, D, C, B, A, D, E, A, X

bis hierhin auch mit depth-first!

alternativ auch D oder C, aber nicht B oder X

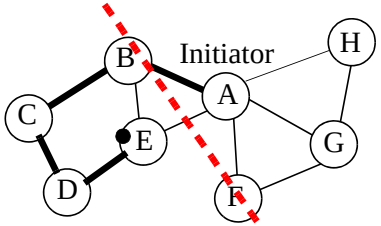
- Mit folgender Regel lässt sich der Nichtdeterminismus des Tarry-Algorithmus soweit einschränken, dass das klassische Depth-first-Traversierungsverfahren resultiert:

R3: Ein Prozess schickt ein empfangenes Token sofort über die gleiche Kante zurück, wenn dies nach R1 und R2 gestattet ist

- Damit ist in obigem Beispiel X, A, B, C, A, E... nicht mehr gestattet!
- Denkübung: *Wieso* wird durch R1-R3 die Depth-first-Traversierung realisiert?
- Denkübung: Kann mit dem Algorithmus *jeder* Spannbaum realisiert werden?
- Denkübung: Hat die Depth-first-Traversierung implementierungstechnische Vorteile gegenüber dem allgemeineren Tarry-Algorithmus?

Depth-first mit Besuchslisten

Voraussetzung: Nachbaridentitäten der Knoten sind bekannt



Token bei E enthält die Namen der bereits besuchten Knoten A, B, C, D, E; Knoten E wird also das Token nicht an B oder A weitersenden (sondern zurück an D)

Idee: - Token merkt sich bereits besuchte Knoten und wird nicht dorthin propagiert

- Jeder Knoten (≠ Initiator) wird also nur 1 Mal besucht (und sendet Token 1 Mal zurück)

⇒ 2n-2 Nachrichten, 2n-2 Zeitkomplexität

Vorteil: Bei "dichten" Netzen (e ≫ n) effizienter!

Nachteil: Hohe Bit-Komplexität (d.h. "lange" Nachrichten)

Bem.: Nachbarschaftswissen lässt sich immer mit 2e Nachrichten erreichen! (Algorithmus dafür haben wir früher kennengelernt)

Frage: Lässt sich vielleicht ein Wellenalgorithmus angeben, bei dem man immer oder zumindest manchmal mit n-1 Nachrichten auskommt (wenn Nachbarschaftswissen vorhanden ist; dann ist das eine untere Schranke, wie wir wissen), statt mit 2n-2?

Depth-first mit Auftragslisten

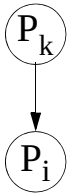
(Voraussetzung wieder: Nachbaridentitäten der Knoten sind bekannt)

- Idee: Token enthält zwei Mengen <...,z,s>:

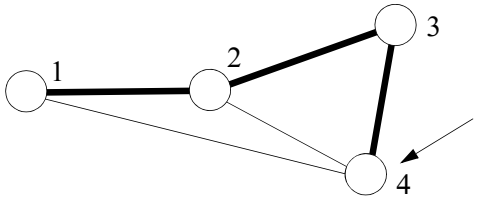
  z: Menge der bereits besuchten Knoten
  s: Menge noch zu besuchender Knoten ("Aufträge")

- Initial sendet Initiator P_i an einen Nachbarn P_j:
  <..., {i}, Nachbarn\{j}>

- Bei Empfang von <...,z,s> von P_k bei P_i:



y := Nachbarn \ z
case
  y = ∅ and s = ∅ --> terminated
  y = ∅ and s ≠ ∅ --> send <... z ∪ {i}, s\{i}> to P_k (zurück)
  y ≠ ∅ --> send <... z ∪ {i}, s∪ y \ {i} \ {m} > to $P_m \in y$



Bereits hier nach drei Nachrichten terminiert

- Beendet, wenn s leer wird (wieso? Beweis?)
- Anderer Knoten als Initiator stellt Terminierung fest (Nachteil? Vorteil, da schneller?)
- Nachrichten- / Zeitkomplexität: n-1 bis 2(n-1)

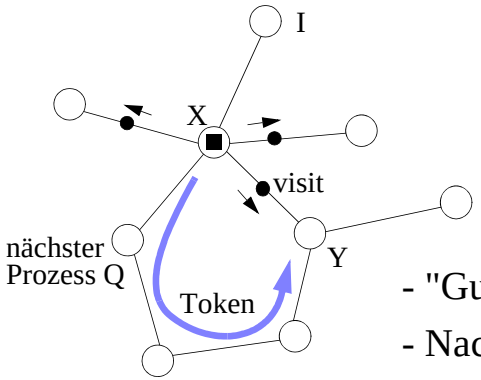
Traversierungsverfahren von Awerbuch

- Annahme: Nachbaridentitäten unbekannt
- Idee: Prozess weiss spätestens dann, wenn das Token ihn besucht, an welche Nachbarn es nicht weitergereicht werden soll (weil es dort schon war)

- Wenn Token einen Prozess besucht:

Sende *visit-Nachrichten* an die Nachbarn; warte auf *ack* von allen, bevor Token weitergereicht wird
→ Nachbarn wissen, dass Token hier bereits war

Ein Prozess schickt Token niemals an einen Prozess, von dem er eine visit-Nachricht erhalten hat
(Ausnahme: Vorgängerknoten beim Backtrack)



Keine visit-Nachricht notwendig an:
- Vorgängerprozess (hier: I)
- nächsten Prozess (hier: Q)

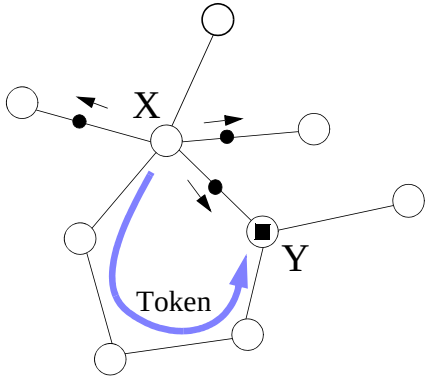
- "Gute" Zeitkomplexität: 4n-2
- Nachrichtenkomplexität: pro Kante i.Allg. 2 visit und 2 ack (bzw. Token)... → ca. 4e

- Frage: Muss Y eine visit-Nachricht an X schicken?
- Frage: Wozu überhaupt auf ack-Nachrichten warten?

Variante von Cidon

- Idee: Wie Awerbuch, aber *keine acks* senden!

- Was geschieht bei Y, wenn visit-Nachricht noch nicht da?
  (also der Kanal XY langsam ist im Vergleich zum Umweg, den Token nahm)



- X erhält Token auf "falscher" Kante (von Y) → ignorieren
  (aber wie eine visit-Nachricht von Y behandeln; Y sendet kein visit mehr an X)

- Y erhält visit-Nachricht von X verspätet, nachdem bereits das Token an X gesendet wurde → Token neu generieren und an einen anderen Nachbarn schicken

---

- Zeitkomplexität: 2n-2 ← Kann es (für beliebige Graphen) ein schnelleres Traversierungs-verfahren geben?

- Nachrichtenkomplexität: max. 4e
  (max. 2 visit und 2 Token pro Kante)

Der Phasenalgorithmus

- Voraussetzung (hier): bidirektionale Nachrichtenkanäle

- *Grundprinzip (Initiator):* ← Prinzip geht auch auf *gerichteten* Graphen

  {noch nicht terminiert}
  - Nachricht an alle Nachbarn senden
  - Lokale Berechnung durchführen
  - Phase := Phase + 1 // lok. Variable; initial 0
  - Eine Nachricht von allen Nachbarn empfangen

- Nicht-Initiatoren steigen erst nach Erhalt einer ersten Nachricht in einen solchen Zyklus ein

---

Eigenschaften: ← Aufgabe: Hiermit die *Wellen-eigenschaft* nachweisen! ← Was sind visit- und conclude-Ereignise?

- Falls ein Prozess in der i-ten Phase ist, befinden sich seine Nachbarn in der i-ten, i-1-ten, oder i+1-ten (wieso?)

- Zwei Prozesse der Entfernung q haben zu jedem Zeitpunkt einen "Phasenunterschied" von max. q (wieso?)
  - Denkübung: können alle Prozesse gleichzeitig in Phase 1 (oder allg.: k) sein?

- Mit D = Durchmesser → Ist der (einzige) Initiator am Ende von Phase D, wurde jeder vom Initiator (indirekt) erreicht
  - Denkübung: hat der Initiator dann bereits von allen Prozessen indirekt gehört?
  - wenn ich weiss, dass jeder in einer Phase > 0 ist, gibt es dann eine effektiv nutzbare Nachrichtenkette von jedem zu mir?

- Eine obere Schranke von D (z.B. n) muss bekannt sein, damit die Terminierung festgestellt werden kann

Algorithmus von Finn

- Wellenalgorithmus für (stark zusammenhängende) *gerichtete* Graphen (unterscheide daher *in-neighbors* und *out-neighbors*) !
- Prozesse müssen eindeutige Identitäten besitzen

- *Idee des Algorithmus*: Solange es Prozesse gibt, die man kennt, aber deren Nachbarn man noch nicht alle kennt, ist man noch nicht fertig

  → Hüllenbildung; dadurch Kennenlernen des gesamten Graphen

- Jeder Prozess P_i besitzt zwei Mengenvariablen A, B, die so initialisiert sind: A = {i}; B = ∅

R_i

Bei Empfang von <A',B'>:
A := A∪A'; B:= B∪B';
if {über alle Eingangskanäle irgendwann mind. eine Nachricht erhalten} **then** B:= B∪{i}; **fi**;
if {A oder B hat sich geändert} **then**
send <A,B> **to all** out-neighbors; **fi**

I_1

{noch nicht gestartet}
send <A,B> **to all** out-neighbors

init-Ereignis (was geschieht eigentlich bei *mehreren* Initiatoren?)

C_i

{A = B}
conclude

jeder Prozess kann conclude ausführen, insbesondere aber der Initiator

Finns Verfahren ist ein Wellenalgorithmus

(1) $x \in B \Longrightarrow$ *in-neighbors*$(x) \subseteq A$
ist eine Invariante für alle Paare (A,B)

- Aktionen C und I verändert (A,B) nicht.
- Aktion I erzeugt ein neues Paar ({i},∅).
- Die beiden update-Aktionen A := A∪A'; B:= B∪B' in Aktion R erhalten ebenfalls die Invariante: Für ein $x \in B$ nach dem update war $x \in B$ vor dem update oder $x \in B'$. Folglich war auch bereits vorher in-neighbors(x) ⊆ A oder in-neighbors(x) ⊆ A', und damit auch noch nach dem update.
- Falls in R das Statement B:= B∪{i} ausgeführt wird, sind alle $y \in$ in-neighbors(i) bereits in A, da Prozess i laut Algorithmus von allen Nachbarn eine Nachricht erhalten hat, in deren A-Menge die Identität des Senders (also desNachbarn) enthalten ist.
- Man beachte ferner, dass dies nicht nur eine Invariante ist, sondern sogar *immer gilt*, da schon initial (A,B) = ({i},∅) in jedem Prozess ist. □

---

(2) *Wenn* A = B in einem Prozess P_i gilt, dann ist dort A = B = {Menge *aller* Prozessidentitäten}

- Es gilt schon initial $i \in A$, folglich (da nun A=B) auch $i \in B$. Da die Mengen nie vermindert werden, bleibt i in beiden Mengen.
- Wegen obiger Invarianten (1) ist in-neighbors(i) ⊆ A.
- Wegen A = B also auch in-neighbors(i) ⊆ B.
- Aus der Invarianten (1) folgt entsprechend in-neighbors(in-neighbors(i)) ⊆ A.
- Damit gilt induktiv ∀k: in-neighbors^k(i) ⊆ A = B.
- Wegen des starken Zusammenhangs ist jeder Prozess in in-neighbors^j(i) für ein gewisses j. □

(3) Sei *visit* = Aktion R bei erstem Versenden der eigenen Identität in der Menge B
Bei conclude (d.h. A=B) im Initiator gilt:

a) Jeder Prozess $P_j \neq$Initiator hat $visit_j$ ausgeführt und $visit_j <$ conclude,

b) init < $visit_j$

a): Wegen (2) "kennt" der Initiator bei conclude P_j (d.h. $j \in A = B$); er kann davon nur über eine Nachrichtenkette erfahren haben, an deren Anfang P_j seine Identität j versendet hat.

b): P_j versendet seine Identität nicht spontan, sondern nur nach (indirekter) Aufforderung durch den Initiator. □

Es ist nun noch zu zeigen, dass der Initiator tatsächlich nach endlicher Zeit conclude ausführt, d.h. dass bei ihm A = B wird (liveness)!

(4) Beim Initiator wird schliesslich A=B

(Voraussetzungen: stark zusammenhängender Graph, endliche Nachrichtenlaufzeiten und Aktionsdauern etc.)

- Über jede Kante des Graphen läuft schliesslich mindestens eine Nachricht ("flooding"): Das erste Empfangen eine Nachricht von einem Nachbarn vergrössert echt die eigene Menge A --> Welle wird an alle Nachbarn weiterverteilt.

- Jeder Prozess P_i verbreitet schliesslich seine eigene Identität i an alle Nachbarn über die B-Mengen, da $B := B \cup \{i\}$ ausgeführt wird.

- Jede Identität erreicht schliesslich mittels Flooding den Initiator über die B-Mengen.

- Beim Initiator gilt so schliesslich: B = {Menge aller Prozessidentitäten}.

- $A \supseteq B$ ist eine Invariante, wie man leicht überprüft.

- Aus den letzten beiden Eigenschaften folgt, dass schliesslich A = B gilt. □

Mit (3) und (4) ist alles gezeigt:
Finns Algorithmus ist ein Wellenalgorithmus!

- obere Schranke für Durchmesser etc. braucht nicht bekannt zu sein
- Nachrichtenkomplexität $\leq 2\ n\ e$ (wieso?)
- höhere Bit-Komplexität als z.B. der Echo-Algorithmus

Eingesetzte Beweistechniken:

- atomare Aktionen
- Invarianten
- monotone Approximation

Es gibt verschiedene Wellenalgorithmen

- Topologiespezifische, z.B. für
  - Ring
  - Baum
  - allg. Graph

  } hierfür spezialisierte Verfahren u.U. besonders effizient

- Voraussetzungen bzgl. Knotenidentitäten
  - eindeutig oder
  - anonym
- Voraussetzungen bzgl. notwendigem "Wissen", z.B.
  - Nachbaridentitäten
  - Anzahl der Knoten (bzw. obere Schranke)
  - ...
- Voraussetzungen bzgl. Kommunikationssemantik
  - synchron, asynchron, FIFO-Kanäle, bidirektionale Kanäle...?

- Qualitätseigenschaften
  - Sequentiell oder parallel (bzw. "Parallelitätsgrad")
  - Anzahl möglicher Initiatoren (mehr als einer?)
  - Zeitkomplexität
  - Nachrichtenkomplexität (worst/average case)
  - Bitkomplexität (Länge der Nachrichten)
  - Dezentralität (kein Engpass?)
  - Symmetrie (alle lok. Algorithmen identisch?)
  - Fehlertoleranz (Fehlermodell? Grad and Fehlertoleranz?)
  - Einfachheit ($\rightarrow$ Verifizierbar, einsichtig...)
  - Praktikabilität, Implementierbarkeit
  - Skalierbarkeit (auch für grosse Systeme geeignet?)
  - ...

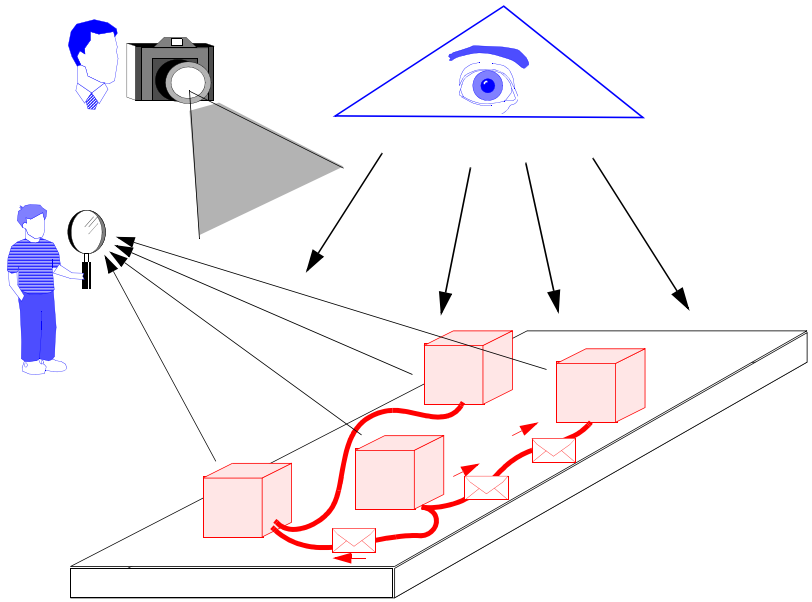
Wellenalgorithmen: Zusammenfassung

- Es gibt viele Wellenalgorithmen, wir kennen u.a.:
  - Echo-Algorithmus ("Flooding mit indirektem Acknowledge")
  - Traversierung von Ringen, Gittern, Hypercubes, Sterntopologien,...
  - Paralleles Durchlaufen von (Spann)bäumen
  - Paralleles Polling auf Sternen
  - Tarry-Algorithmus, Depth-first-Traversierungen
  - Verfahren von Awerbuch und Variante von Cidon
  - ~~Phasenalgorithmus~~
  - ~~Algorithmus von Finn~~

- Anwendung von Wellenalgorithmen (u.a.):
  - *Broadcast*
  - *Einsammeln* verteilter Daten ("gather"); evtl. mit dezentraler Akkumulation
  - Konstruktion eines *Spannbaums*
  - *Phasensynchronisation* von Prozessen
  - *Triggern eines Ereignisses* in jedem Prozess
  - Implementierung von *Schnittlinien* ($\rightarrow$ Schnappschuss etc.)
  - *Basisalgorithmus* für andere Verfahren (Deadlock, Terminierung,...)

- Es gibt viele Wellenalgorithmen $\Rightarrow$ welcher ist der beste?
  - Es gibt sicherlich keinen "allgemein besten" - je nach Voraussetzungen wird man nur eine Teilmenge davon in Betracht ziehen können, ferner gibt es sehr unterschiedliche Qualitätskriterien (vgl. frühere Aufzählung)!

Globale Zustände, Beobachtungen, Prädikate



Kausal-konsistente globale Zustände

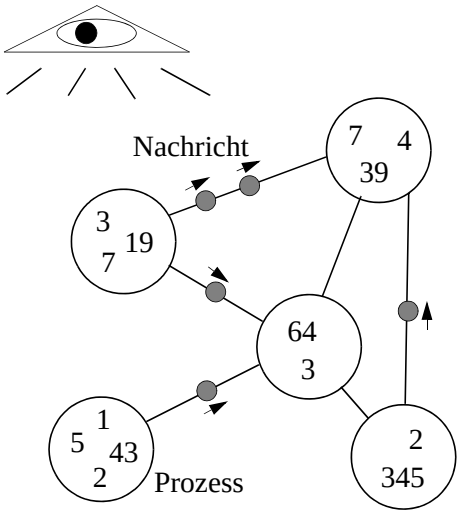
Webster:

State = a set of circumstances or attributes characterizing a person or thing at a given time

Gibt es "globale Zeit" in einem vert. System?

Globaler Zustand (zu einem Zeitpunkt):

Alle lokalen *Prozesszustände* + alle *Nachrichten*, die "unterwegs" sind



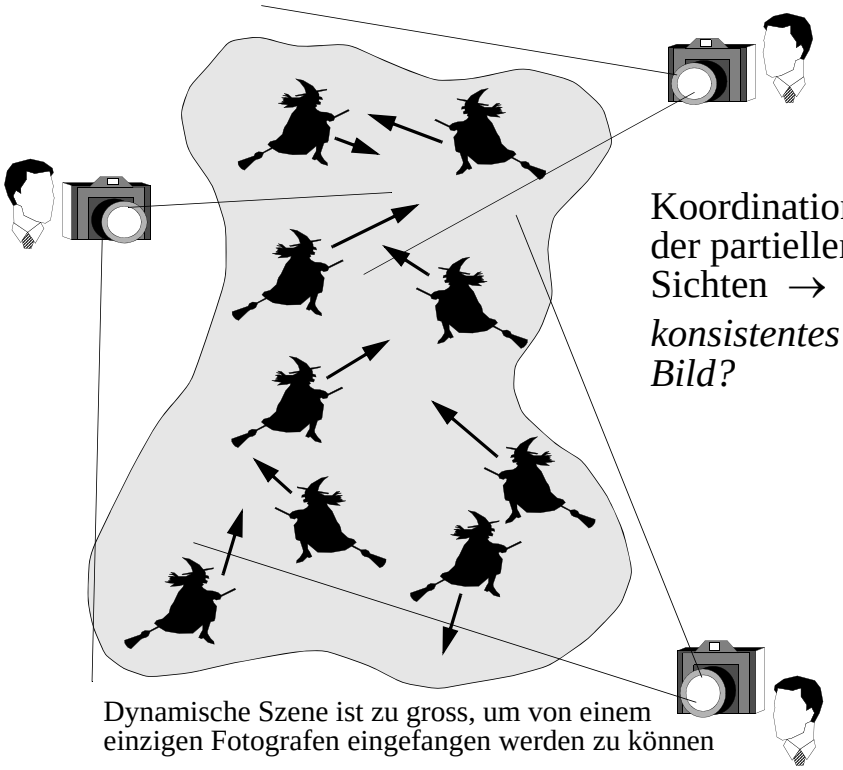
Schnappschuss

Problem:

Prozesszustände sind nur einzeln ("nacheinander") erfahrbar → Konsistenz?

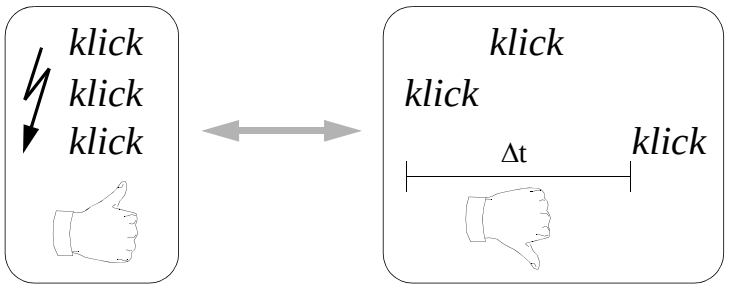
als ob alles gleichzeitig wahrgenommen wird

Synchronisierte lokale Schnappschüsse?



Koordination der partiellen Sichten → *konsistentes Bild?*

Dynamische Szene ist zu gross, um von einem einzigen Fotografen eingefangen werden zu können



Das Schnappschussproblem

Problem: "Momentaner" *Schnappschuss* des globalen Zustands, ohne das System anzuhalten

Realität:

- *Volkszählung*: Stichzeitpunkt (geht hier nicht)
- *Inventur*: Einfrieren (unpraktisch)

Anwendungen:

- konsistenter Aufsetzpunkt für verteilte Datenbanken
- wie hoch ist die momentane Last?
- Testen verteilter Systeme: gilt eine globale Eigenschaft?
- Deadlock: Existiert eine zykl. Wartebedingung?
- ist die verteilte Berechnung terminiert?
- ist ein bestimmtes Objekt "Garbage"?
- ...

} *Prädikate* über glob. Zuständen

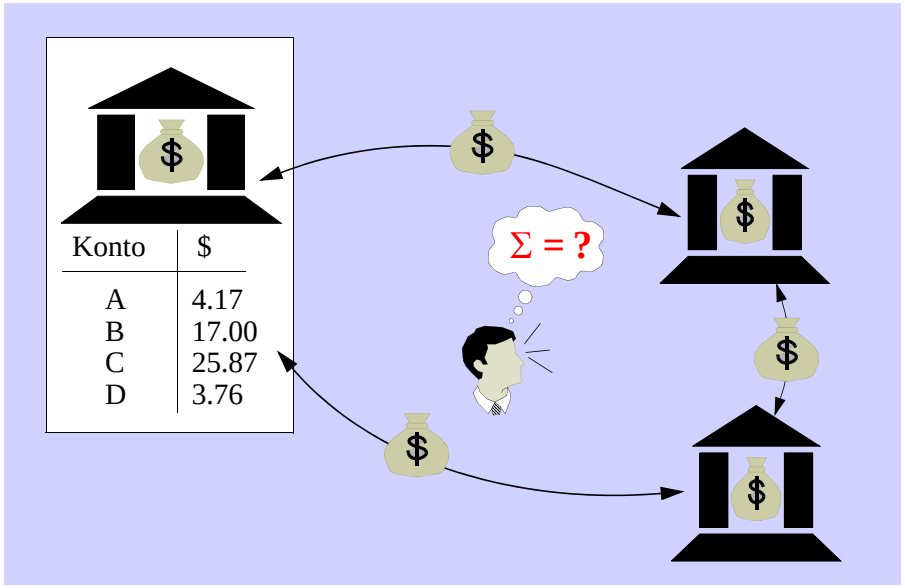
Schwierigkeiten:

- unmöglich, alle Prozesse gleichzeitig zu erwischen
- unbestimmte Nachrichtenlaufzeiten
- Nachrichten, die unterwegs sind, sieht man nicht
- ermittelter Zustand ist i.Allg. veraltet und
  - ... u.U. nie "wirklich" so gewesen
  - ... u.U. inkonsistent

zumindest dies ausschliessen!

→ *Schnappschussalgorithmus*

Beispiel: Kommunizierende Banken



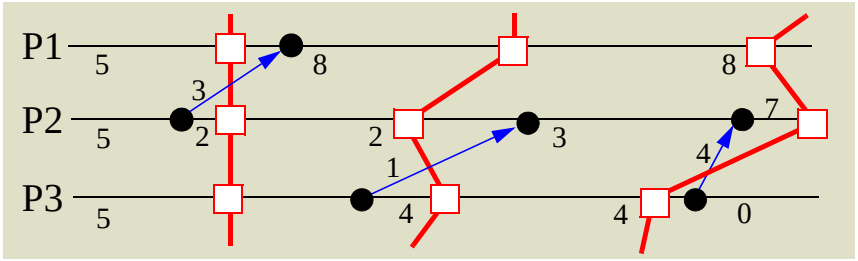
- Modellierung:
  - ständige Transfers zwischen den Konten bzw. Banken
  - lokale atomare Aktionen: alle Geldkonten *einer* Bank können "gleichzeitig" (also atomar und damit "lokal konsistent") untersucht werden

- Wieviel Geld ist in Umlauf?
  - konstante Geldmenge, oder
  - monotone Inflation (→ untere Schranke für momentane Geldmenge)

- Erschwerte Bedingungen:
  - niemand hat eine globale Sicht
  - gemeinsame Zeit?

(In)konsistente Schnitte

Beispiel: Wieviel Geld ist in Umlauf?



idealer (senkrechter) Schnitt → 15

Unerreichbar

konsistenter Schnitt → 15

Äquivalent zu senkrechtem Schnitt (lässt sich problemlos "geradebiegen")

inkonsistenter Schnitt → 19 (+4 ?)

Lässt sich nicht "geradebiegen"

manches doppelt (bzw. gar nicht) gezählt

"Gummiband-transformation"

→ verändert Metrik

→ erhält Topologie

Wie wir noch einsehen werden:

Kausaltreues Beobachten ⇔ Folge konsistenter Schnitte
- wie erreicht man das?

Schnappschussalgorithmen: Zweck

auch wenn der Zustand nur möglich gewesen *wäre*, aber gar nicht *wirklich* auftrat!

aber was heisst schon "wirklich"?

- Liefern *konsistenten, möglichen, vergangenen* Zustand

  - *konsistenter Zustand* = globaler Zustand entlang einer konsistenten Schnittlinie (keine Nachricht "aus der Zukunft")

  - nur über konsistenten Zuständen können *globale Prädikate* sinnvoll bestimmt werden (da diese "äquivalent" zu Zuständen entlang senkrechter Schnittlinien sind); in diesem Sinne sind solche Algorithmen wichtig!

  - ein Prädikat heisst *stabil* (oder monoton), wenn es nie wieder aufhört zu gelten, nachdem es einmal gilt (es also in allen zukünftigen Zuständen gilt); z.B. Terminierung, Garbage, Deadlock...

- Falls Prädikat stabil: "Entdecken" dieses Prädikats (⇒ "stable property detection algorithm")

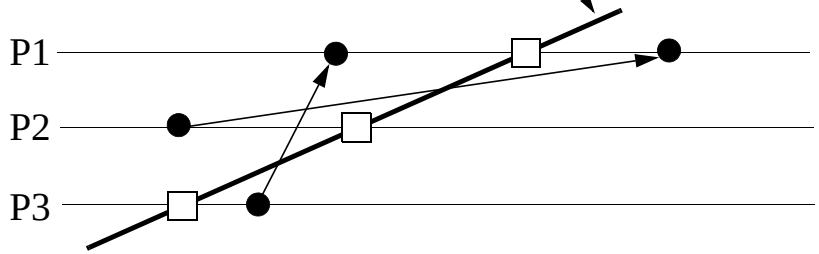
  - bei stabilen Prädikaten ist "möglich... vergangen" sogar brauchbar! (es gilt dann sicherlich jetzt)

Zum Nachdenken:

- aber wenn die betrachtete Eigenschaft nicht stabil ist, was dann?
- wer garantiert eigentlich, dass eine Eigenschaft (wirklich) stabil ist?

Ein Schnappschussalgorithmus

Schnittlinie durch irgend einen *Wellenalgorithmus* realisieren



Prozesse, Nachrichten: *schwarz* oder *rot*

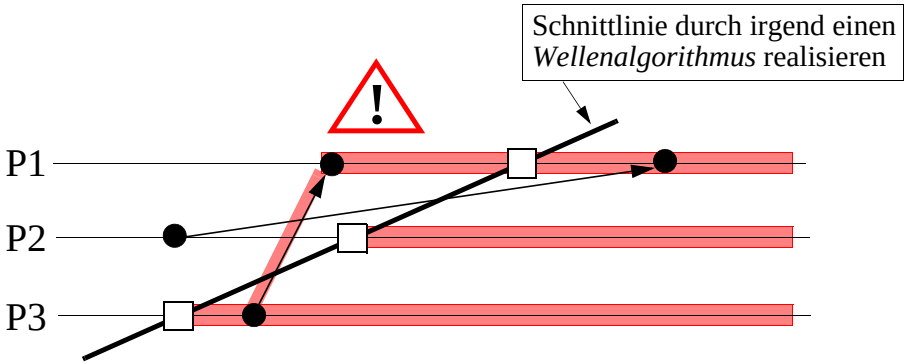
Schnappschussaugenblick: schwarz → rot
(dann: lokalen Zustand dem Initiator melden)

Prozess wird rot bei a) Erhalt expliziter Aufforderung
b) Erhalt einer roten Nachricht

Beh.: Schnappschuss ist *konsistent*.
Bew.: Keine "Nachricht aus der Zukunft"

Nachrichten, die *unterwegs* sind?
- Schwarze Nachrichten, die bei rot ankommen
- Sende (bei Empfang) *Kopie* davon an Initiator
- Problem: Wann letzte Kopie dort eingetroffen?

Ein Schnappschussalgorithmus



Prozesse, Nachrichten: *schwarz* oder *rot*

Schnappschussaugenblick: schwarz → rot
(dann: lokalen Zustand dem Initiator melden)

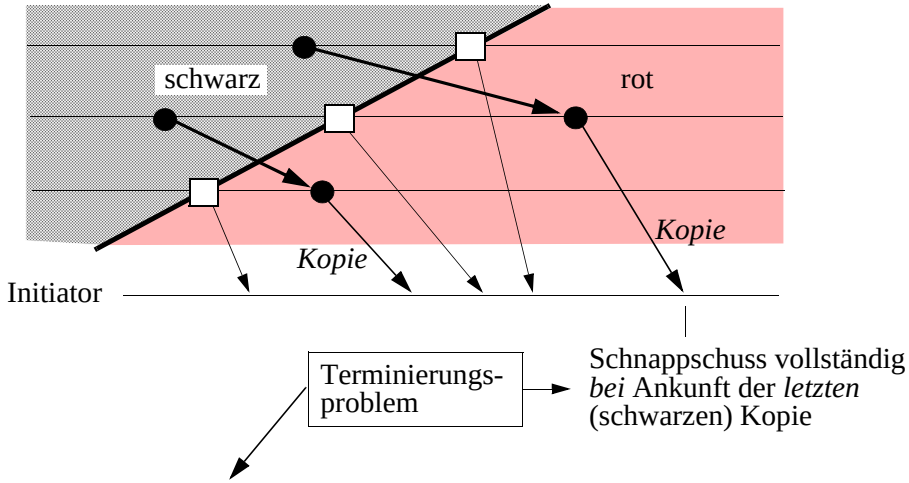
Prozess wird rot bei a) Erhalt expliziter Aufforderung
b) Erhalt einer roten Nachricht

Beh.: Schnappschuss ist *konsistent*.
Bew.: Keine "Nachricht aus der Zukunft"

Nachrichten, die *unterwegs* sind?
- Schwarze Nachrichten, die bei rot ankommen
- Sende (bei Empfang) *Kopie* davon an Initiator
- Problem: Wann letzte Kopie dort eingetroffen?

Schnappschussalgorithmus: Nachrichten

- In-transit-Nachrichten?
  - schwarze Nachrichten, die von einem roten Prozess empfangen werden
  - Sende (bei Empfang) eine Kopie davon an den Initiator
  - Problem: Wann hat der Initiator die letzte Kopie erhalten?



- Z.B. "Defizitzähler" als Teil des konsistenten Zustands
  - zähle gesendete und empfangene schwarze Nachrichten
  - globale Differenz = Anzahl zu erwartender schwarzer Kopien
- Beliebiger Wellenalgorithmus als "*Basisalgorithmus*"
  → unterschiedliche Schnappschussalgorithmen

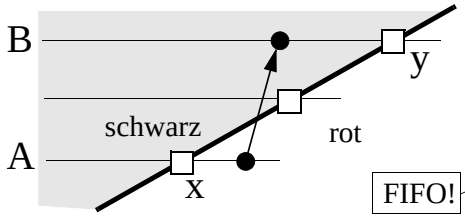
- FIFO-Kommunikation keine Voraussetzung
- Letzte In-transit-Nachricht abwarten
  - das kann aber lange dauern; geht es nicht auch schneller? (ja! wie?)
- "Repeated snapshot": Farben vertauschen

Der Chandy/Lamport-Algorithmus für konsistente Schnappschüsse

- *Idee:* 1) Setzt FIFO-Kanäle voraus ("flushing-Prinzip")
  Marker schieben In-transit-Nachrichten aus den Kanälen heraus

  2) Flooding als zugrundeliegendes Wellenverfahren

- Keine Nachricht aus der Zukunft ⇒ Schnitt ist *konsistent*
  - vgl. auch frühere Ausführungen zu "virtuell gleichzeitiges Markieren"



- bei x wird ein *Marker* von A nach B gesendet (falls A und B benachbart sind)
- Nachricht kann den Marker nicht überholen ⇒ nach x gesendete Nachricht kommt auch *nach* y an

---

- *In-transit-Nachrichten* bei FIFO-Kanälen:
  - Nach der *letzten* schwarzen Nachricht folgt ein Marker
  - Empfang eines Markers informiert den Empfänger, dass nun über diesen Kanal keine schwarzen Nachrichten mehr unterwegs sind

---

- *Vorteil*: Farben müssen nicht (in Nachrichten) mitgeführt werden
- *Nachteile*:
  - bei dichten Netzen grosse Zahl von Kontrollnachrichten
  - FIFO ist notwendig
  - lokale Zustände müssen i.Allg. zum Initiator gebracht werden (z.B. mittels Echo-Nachrichten)

Der Chandy/Lamport-Algorithmus (2)

(Zum Nachlesen und Studieren: ACM TOCS Vol. 3, 1985, 63-75)

- Globaler Zustand besteht aus den *Prozesszuständen* und allen *Kanalzuständen*

- Im Unterschied zum Nicht-FIFO-Fall sind Kanalzustände *Folgen* von Nachrichten, keine Mengen

  Channel State = sequence of messages sent along the channel before the sender's state is recorded, excluding the sequence of messages received along the channel before the receiver's state is recorded

Marker-Sending Rule for a Process p. For each channel *c*, incident on, and directed away from *p*:

p sends one marker along *c* after *p* records its state and before *p* sends further messages along *c*

Marker-Receiving Rule for a Process q. On receiving a marker along channel *c*:

if *q* has not recorded its state **then**
  q records its state;
  q records the state of *c* as the empty sequence
else
  q records the state of *c* as the sequence of messages received along *c* after *q*'s state was recorded and before *q* received the marker along *c*

- Wie verhält sich ein Initiator? Kann der Algorithmus "spontan" von mehreren Prozessen unabhängig voneinander initiiert werden?

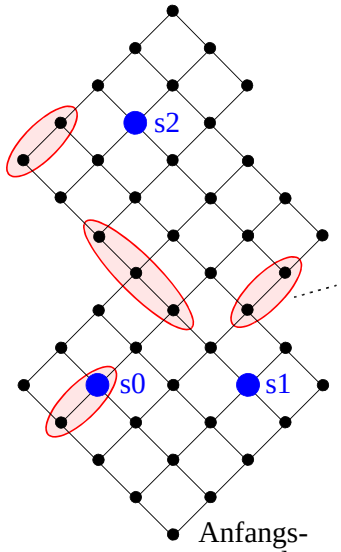
- Wieso ist manchmal der Kanalzustand die leere Folge? Hat ein mit diesem Algorithmus gewonnener globaler Zustand *immer* einige leere Kanäle?

Türken einen Tag unter "Hausarrest" gestellt

ANKARA (dpa). Mehr als 50 Millionen Einwohner der Türkei standen gestern unter "Hausarrest". Weil die Wählerlisten für ein am 6. September anstehendes Referendum überprüft werden sollten, durfte die Bevölkerung den ganzen Tag die Wohnungen nicht verlassen. In der Volksbefragung soll über Fortdauer oder Aufhebung des seit 1980 bestehenden politischen Betätigungs-Verbots für die früheren Ministerpräsidenten Ecevit und Demirel entschieden werden. Während landesweit zehntausende von Helfern unterwegs waren, um in dem wie ausgestorben wirkenden Land die Eintragungen in die Wählerlisten in den Wohnungen der Wähler zu kontrollieren, attackierten die beiden Oppositionspolitiker die umständliche Methode der Zählung.

Eigenschaften verteilter Berechnungen mit Schnappschüssen "entdecken"?

- Ein wiederholt angewendeter Schnappschussalgorithmus könnte z.B. zuerst s1, dann s2 liefern (s2 ist "später" als s1)
  - dazwischen sind Lücken!



Menge der konsistenten Zustände der Berechnung, geordnet entsprechend der "später-Relation" (entspricht der Präfixbeziehung)

Eigenschaft sei in diesen Bereichen gültig (wird jedoch von s1 oder s2 nicht "entdeckt", allenfalls von s0)

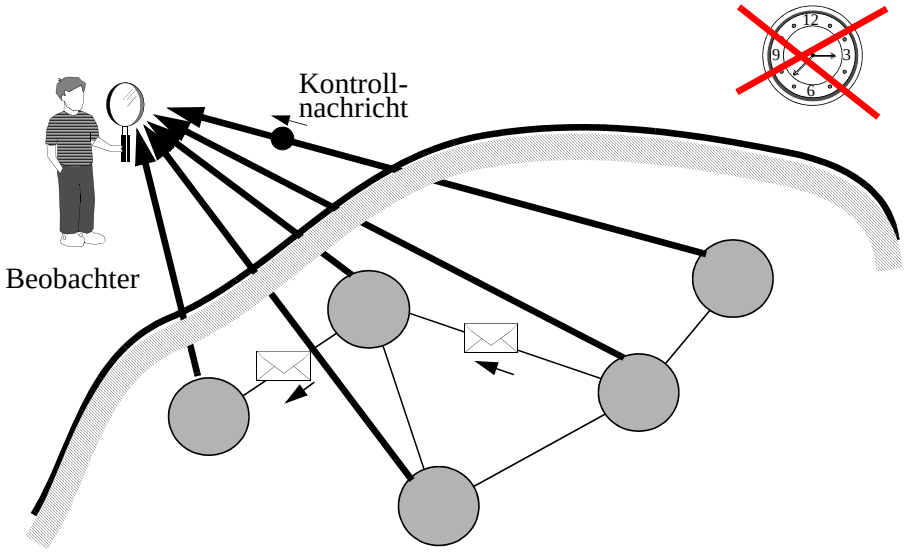
wenn dies in einem Zustand gilt, *dann* auch in allen späteren

- Sinnvoll, wenn die Eigenschaft stabil ist
  - beachte: wir wissen nicht, ob "in Wirklichkeit" s0 oder s1 eintrat!

- Wir hätten gerne eine lückenlose "Folge" konsistenter Schnappschüsse als eine "*Beobachtung*" der Berechnung
- Allerdings sind Berechnungen nur *halbgeordnete* Mengen (konsistenter) Zustände (also keine Folgen)!

Beobachten verteilter Berechnungen

wie früher schon erwähnt...:

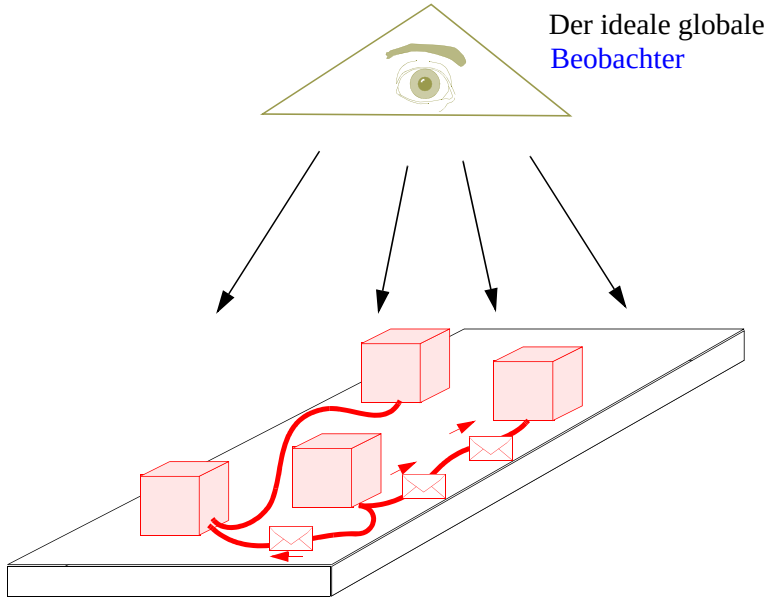


Beobachten geht nur über das Empfangen von "Kontrollnachrichten" (mit unbestimmter Laufzeit)

"Axiom": Mehrere Prozesse können "niemals" gleichzeitig beobachtet werden

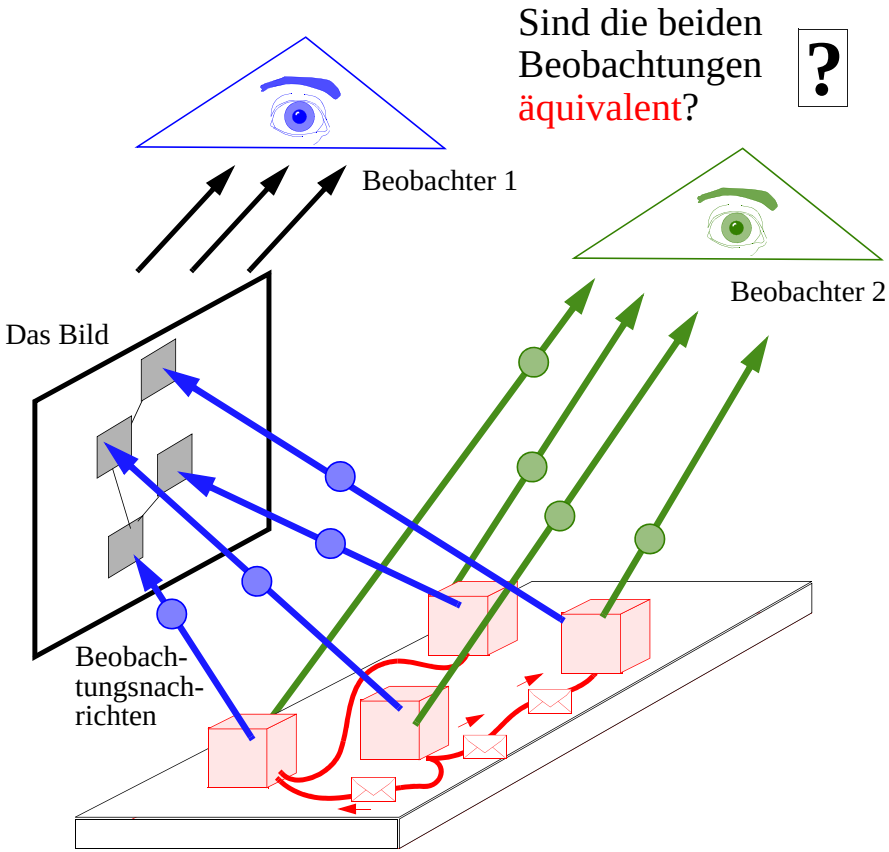
"Korollar": Aussagen über den globalen Zustand sind schwierig

Verteilte Berechnung und Beobachtung



Eine verteilte Berechnung

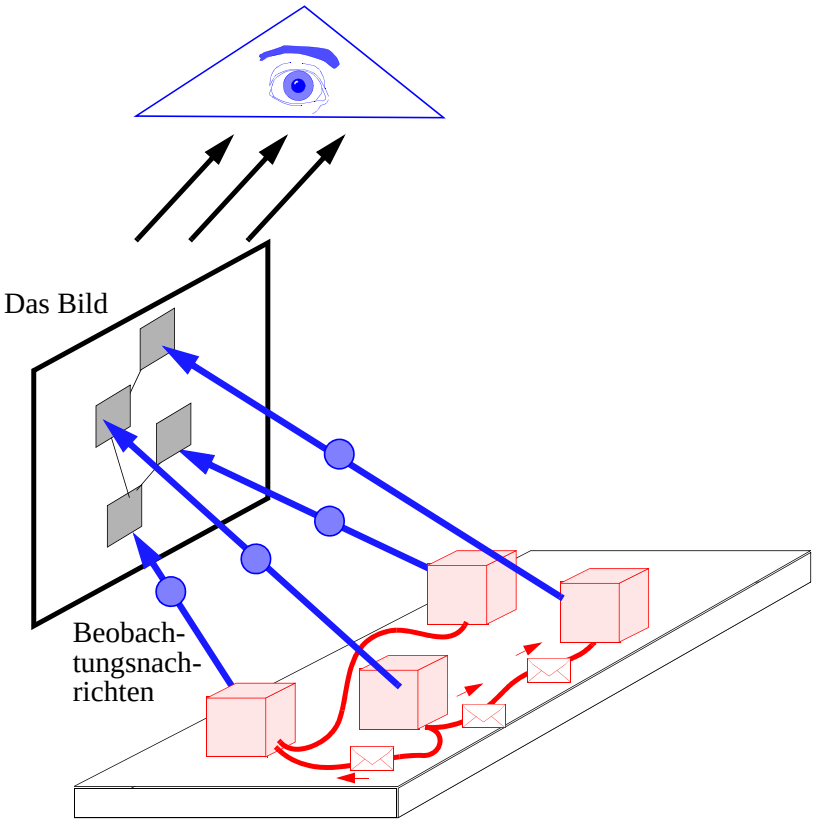
Beobachtungen...



Probleme:

- Zeitverzögerung der Beobachtung
- Konsistenz des Bildes
- Verzerrung des Verhaltens ("Heisenberg’sche Unschärfe")

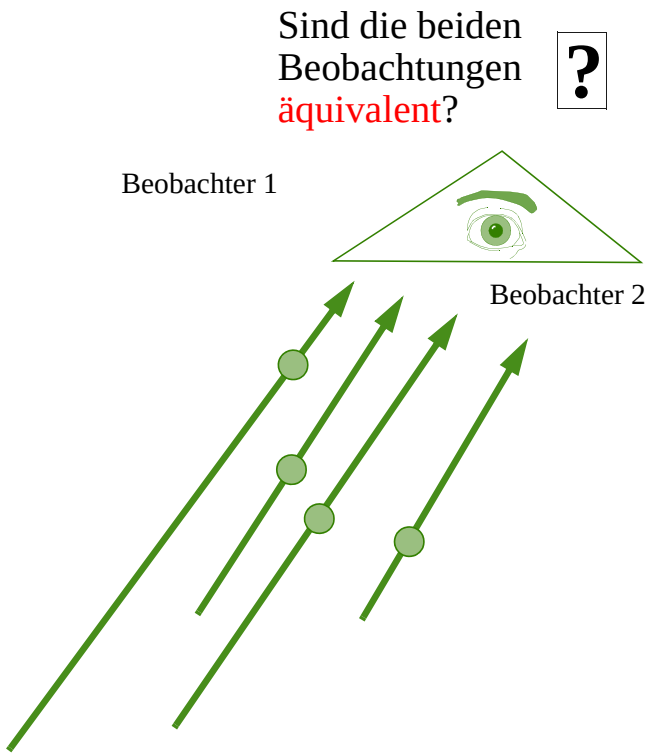
Beobachtungen...



Probleme:

- Zeitverzögerung der Beobachtung
- Konsistenz des Bildes
- Verzerrung des Verhaltens ("Heisenberg’sche Unschärfe")

Das Paradigma der Beobachtung



- “Korrektes” Beobachten verteilter Systeme ist ein wichtiges praktisches Problem

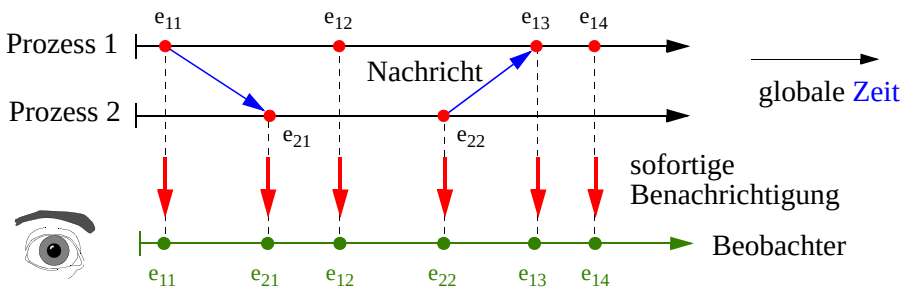
- Kausaltreue (auch: "kausal konsistente") Beobachtungen stellen das Kernproblem vieler verteilter Algorithmen dar

---

⇒ Wie realisiert man kausaltreue Beobachter?

⇒ Was sind, formal gesehen, *kausaltreue Beobachter / Beobachtungen ?*

Ideale Beobachtungen



wesentliche Vorkommnisse, z.B.
Nachricht empfangen / senden

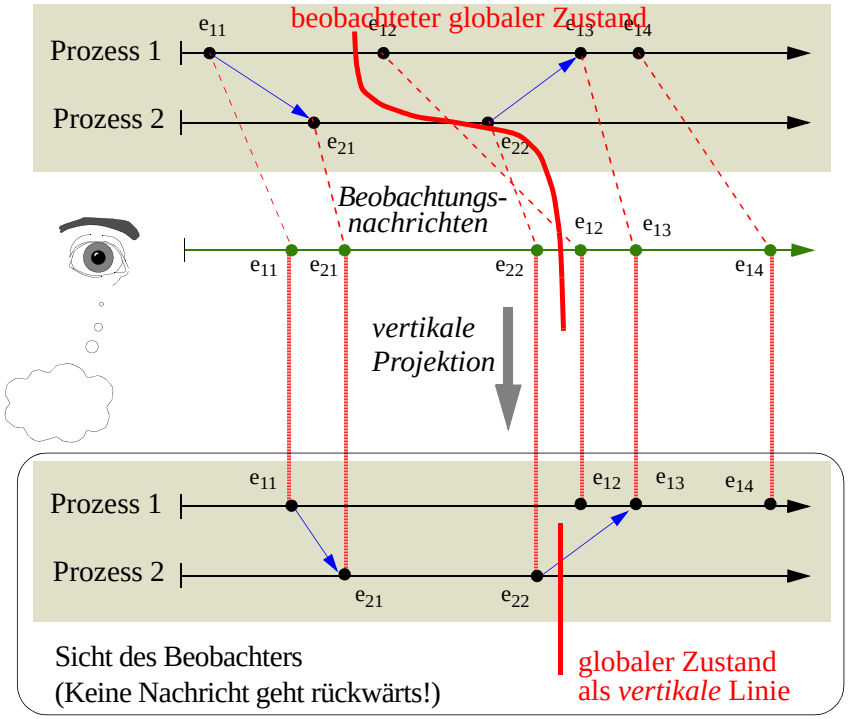
- Beobachtet werden Ereignisse bei den Prozessen
  - Meldung nach aussen über Beobachtungsnachrichten

- Eine solche ideale Beobachtung möchten wir haben, können das aber nicht garantiert bekommen

- Statt dessen könnten wir etwas bekommen, was wir nicht haben wollen...
  (nämlich eine inkonsistente Beobachtung, bei der sich Beobachtungs-nachrichten so überholen, dass Ursache und Wirkung vertauscht werden)

Kausaltreue Beobachtungen

- Ursachen werden stets vor ihren Wirkungen angezeigt
  - dies hoffen wir zu bekommen! (aber wie?)



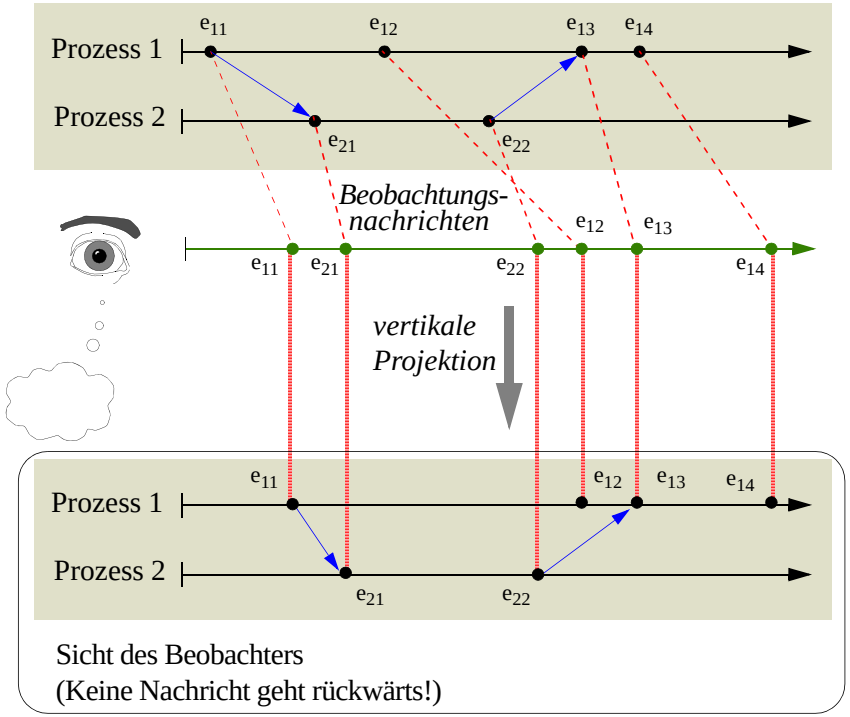
- Beobachtersicht ist nur in irrelevanter Weise verzerrt
  (d.h. ist eine Gummibandtransformation des echten Ablaufes)

- Beobachteter globaler Zustand ist daher *möglich*

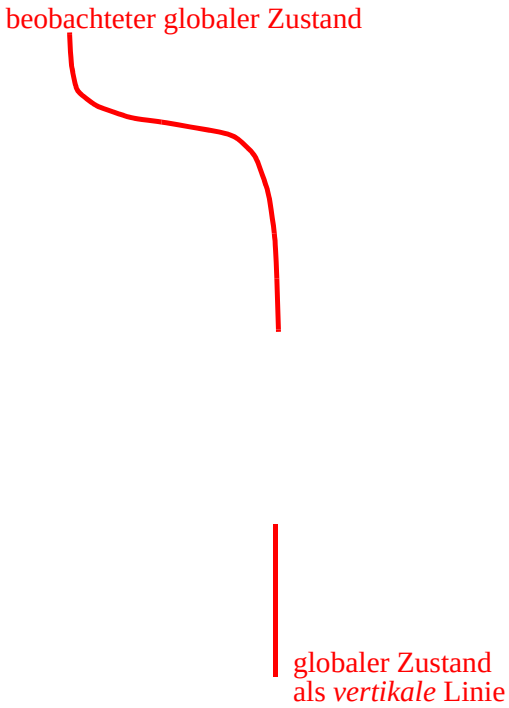
- *gültige*, nicht zu widerlegende Interpretation
- nicht entscheidbar, ob *tatsächlich eingetreten*

Kausaltreue Beobachtungen

- Ursachen werden stets vor ihren Wirkungen angezeigt
  - dies hoffen wir zu bekommen! (aber wie?)



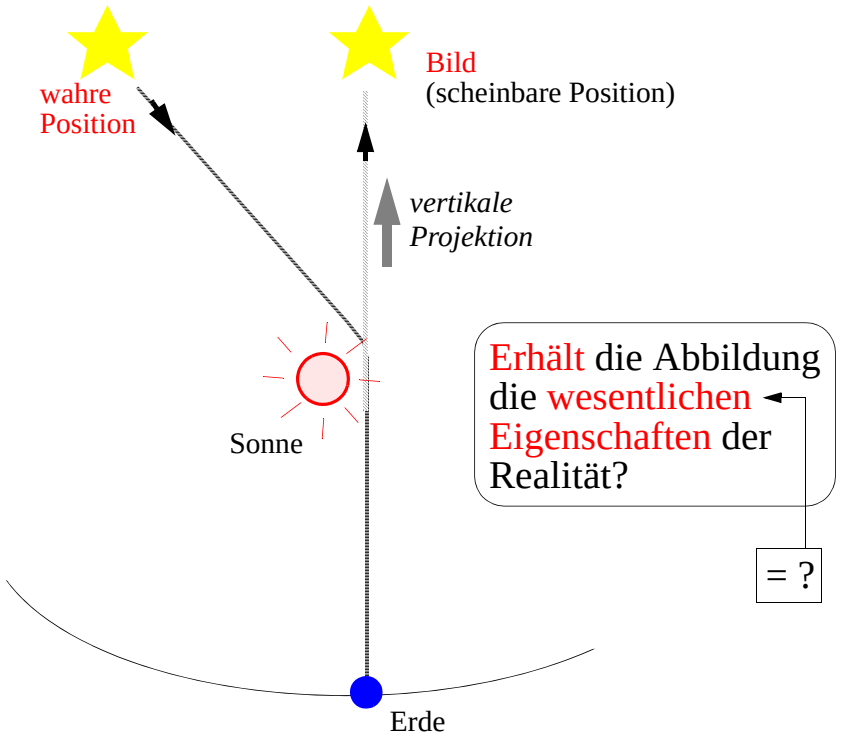
- Beobachtersicht ist nur in irrelevanter Weise verzerrt
  (d.h. ist eine Gummibandtransformation des echten Ablaufes)



- Beobachteter globaler Zustand ist daher *möglich*

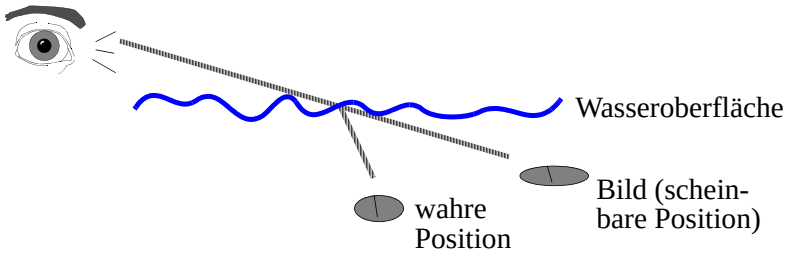
  - *gültige*, nicht zu widerlegende Interpretation
  - nicht entscheidbar, ob *tatsächlich eingetreten*

Bild, Schein und Wirklichkeit

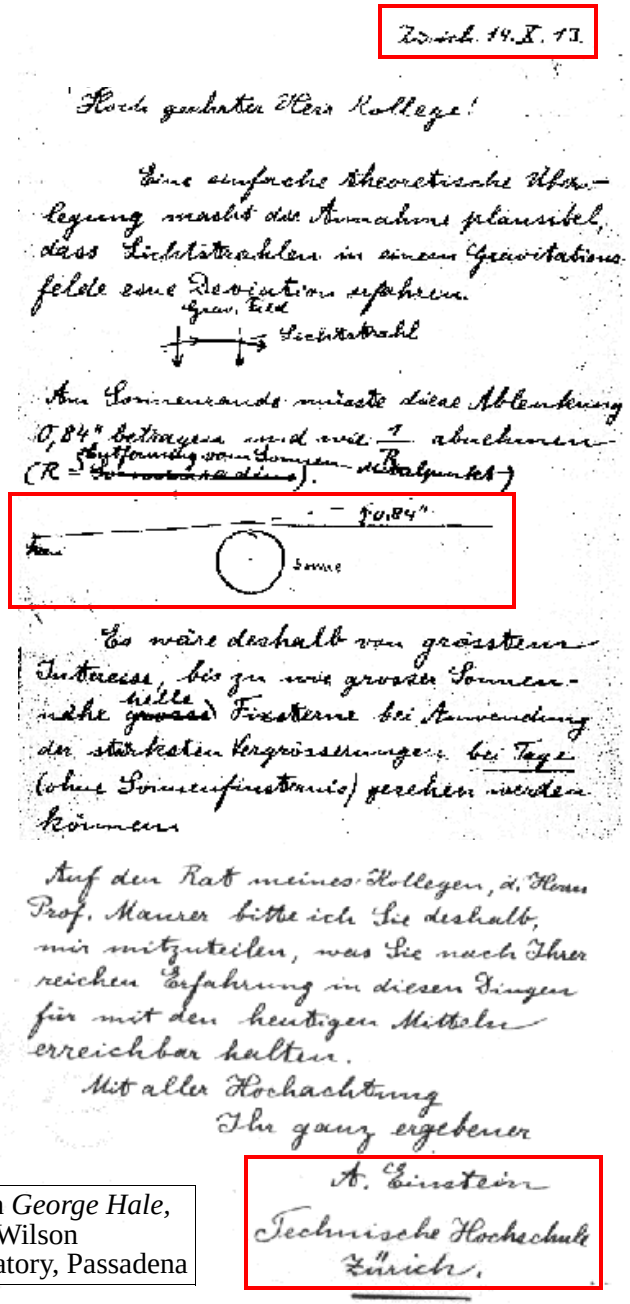


Erhält die Abbildung die wesentlichen Eigenschaften der Realität?

= ?



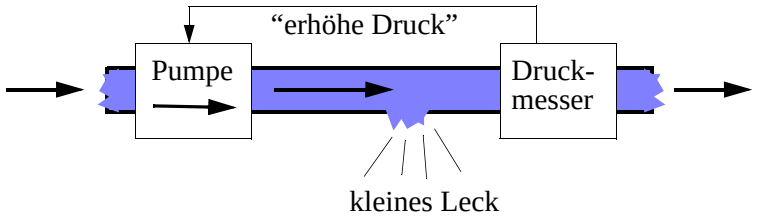
- Offenbar kann man mit einem gegenüber der Realität "etwas" verzerrten Bild ganz gut leben...

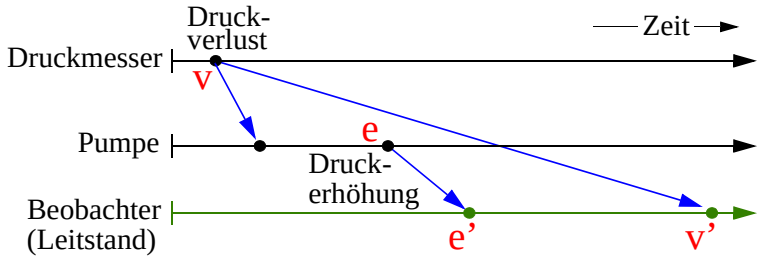


Brief an *George Hale*, Mount Wilson Observatory, Passadena

Kausal (in)konsistente Beobachtungen

wie früher schon erwähnt...:





Falsche Schlussfolgerung des Beobachters:

Es erhöhte sich der Druck (aufgrund einer unbegründeten Aktivität der Pumpe), es kam zu einem Leck, was durch den abfallenden Druck angezeigt wird.

Forderung (“kausaltreue Beobachtung”):

Ursache stets vor (u.U. indirekter) Wirkung beobachten!

Frage:

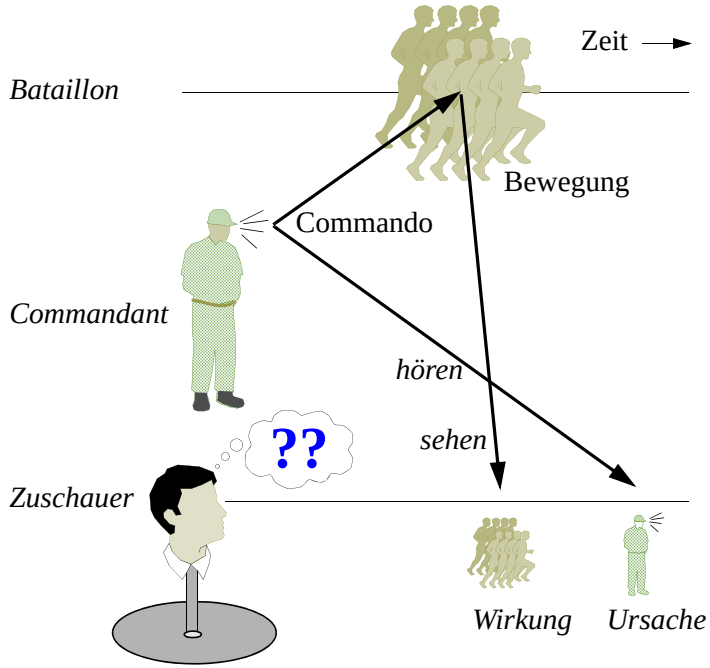
Wie implementiert man kausal konsistente Beobachter?

Kausal (in)konsistente Beobachtungen

Das Beobachtungsproblem ist nicht neu...

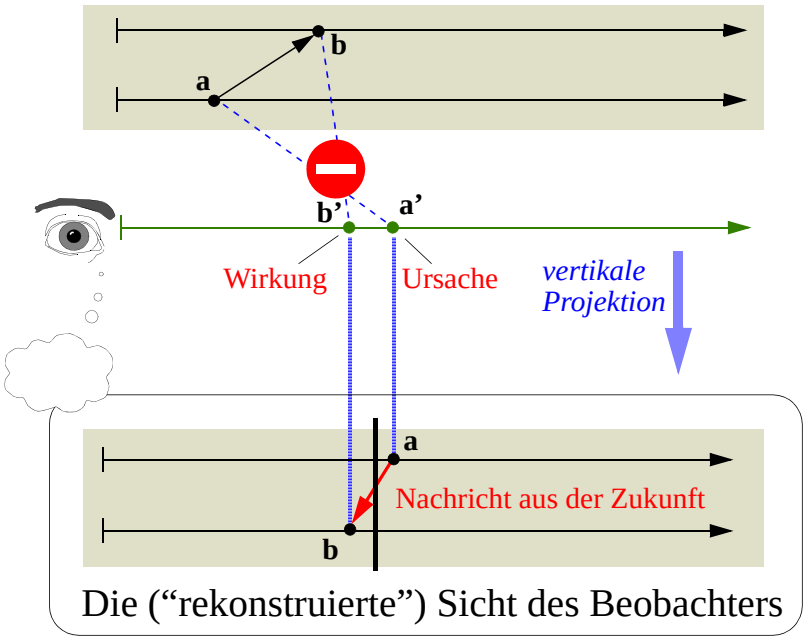
Wenn ein Zuschauer von der Ferne das Exercieren eines Bataillons verfolgt, so sieht er übereinstimmende Bewegungen desselben plötzlich eintreten, *ehe* er die Commandostimme oder das Hornsignal hört; aber aus seiner Kenntnis der *Causalzusammenhänge* weiss er, dass die Bewegungen die *Wirkung* des gehörten Commandos sind, dieses also jenen *objectiv* vorangehen muss, und er wird sich sofort der Täuschung bewusst, die in der Umkehrung der Zeitfolge in seinen Perceptionen liegt.

Christoph von Sigwart (1830-1904) *Logik* (1889)



Kausal inkonsistente Beobachtungen

- "Überholungen" von Benachrichtigungen:



- In der Interpretation des Beobachters:
  - Nachricht fliesst rückwärts in der Zeit
  - Kausalität ist verletzt (Wirkung vor Ursache!)
  - Beobachteter globaler Zustand (nach b, vor a) ist inkonsistent

---

- Wir hätten gerne eine kausaltreue Beobachtung, wo die Beobachtersicht nur in irrelevanter Weise verzerrt ist
  - d.h. eine Gummibandtransformation des echten Ablaufes ist (bei der keine Nachricht rückwärts verläuft)
  - wie erzwingt man das?

Kausaltreue Beobachtungen

lineare Erweiterung oder Einbettung

Definition:
Eine *kausaltreue Beobachtung* einer Berechnung ist eine Linearisierung der zugehörigen (partiellen) Kausalordnung (E,<)

- mit anderen Worten: Jeder "Trace" von Ereignissen, in dem eine Wirkung niemals vor ihrer Ursache erscheint, heisst "kausaltreue Beobachtung"

Bem.: Es gibt i.Allg. viele unterschiedliche Linearisierungen!

- wieviele? (Abschätzung in Abhängigkeit der Ereignis- und Prozesszahl?)
- eine sequentielle Berechnung besitzt offenbar nur eine einzige Linearisierung

---

- kausal unabhängige Ereignisse können stets in unterschiedlicher Reihenfolge wahrgenommen werden
- alle kausaltreuen Beobachtungen sind gleichermassen "wahr"
- alle kausaltreuen Beobachter sind sich bzgl. Kausalitätsrelation einig!

Schnitt aller Linearisierungen einer Halbordnung = Halbordnung
(Theorem von Szpilrajn: "Sur l'extension de l'ordre partiel", Fund. Math., 1930)

⇒ der "unstrittige Kern" aller Beobachter ist die Kausalrelation der Berechnung selbst!

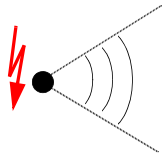
⇒ in den "wesentlichen" Aspekten stimmen alle Beobachter überein! (die Kausalbeziehung ist also ein beobachterinvariantes, objektives Faktum)

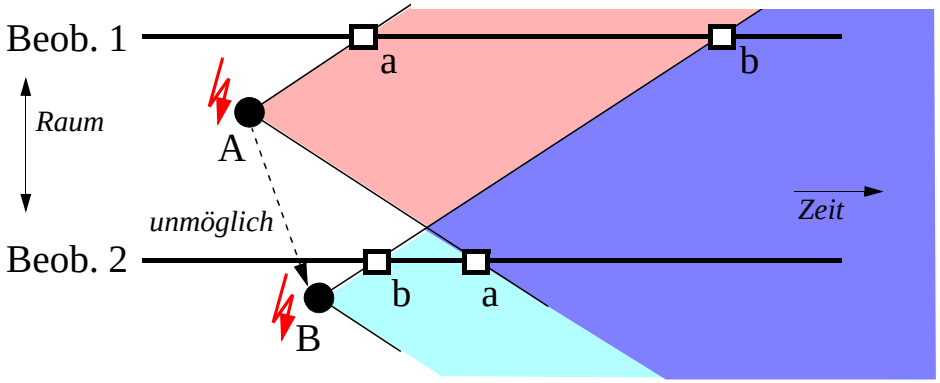
⇒ vert. Berechnung ist durch die Menge aller ihrer Beobachtungen charakterisiert

Relativierung der Gleichzeitigkeit

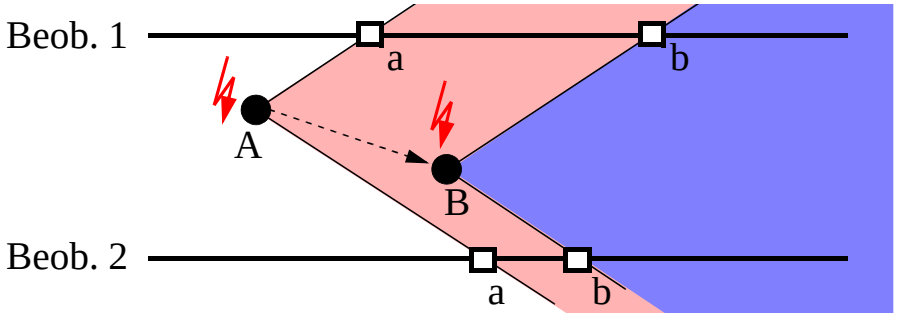
Zwei "kausal unabhängige" Ereigniss können in beliebiger Reihenfolge beobachtet werden!

Lichtkegel-Prinzip der relativistischen Physik





A und B geschehen "gleichzeitig"
(jedenfalls *kausal unabhängig*!)

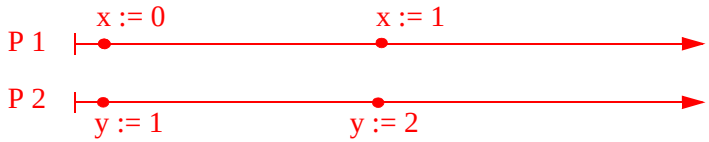


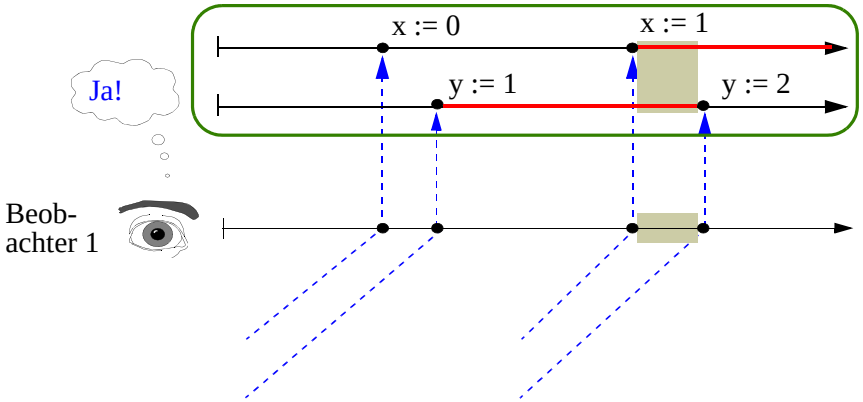
beobachterinvariant
⇒ *objektive Tatsache*

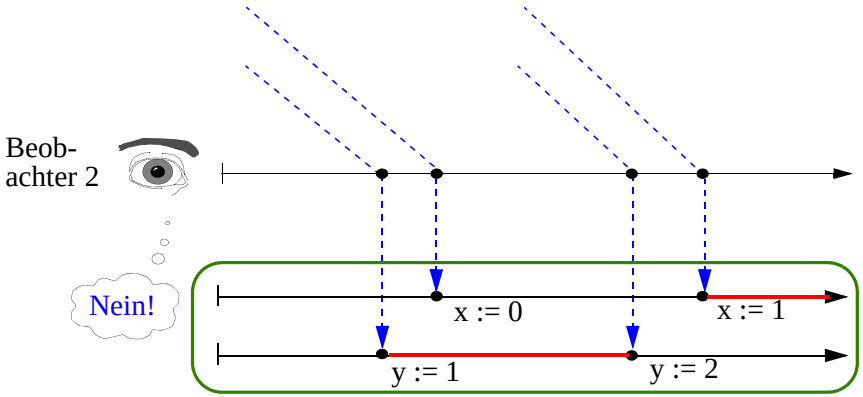
B liegt im Kegel von A →
B hängt kausal ab von A →
Alle Beobachter sehen B nach A

Das "Entdecken" globaler Prädikate

Frage: Gilt in dieser Berechnung $\Phi \equiv (x = y)$?





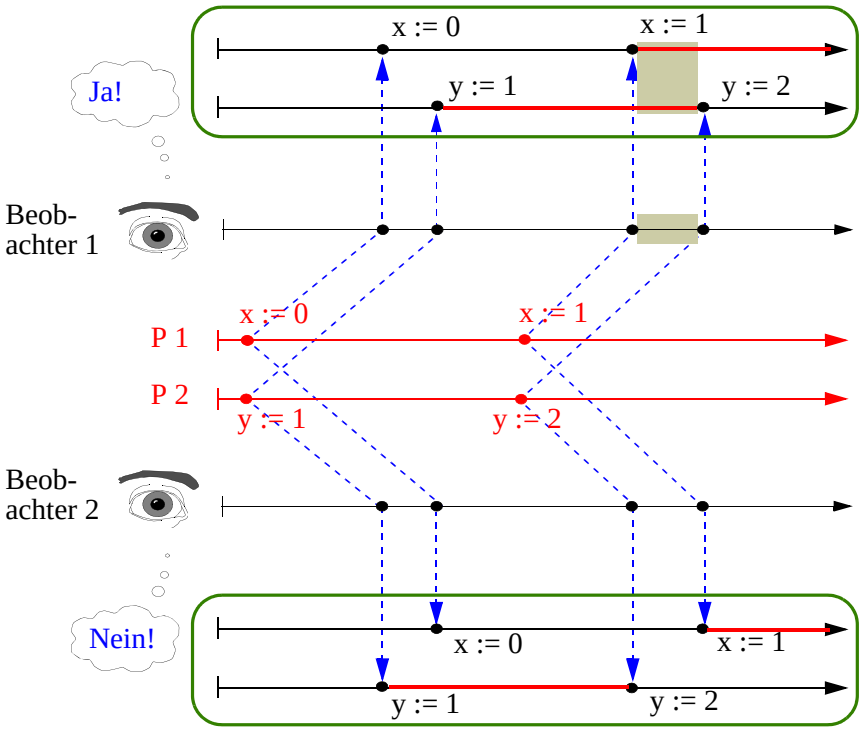


- Meldungen alle gleich schnell
- Beide Beobachtungen sind gleich “richtig”
- Die Beobachter stimmen bzgl. x = y nicht überein!

Aber was denn nun: gilt x=y in dieser Berechnung oder nicht?

Das "Entdecken" globaler Prädikate

Frage: Gilt in dieser Berechnung $\Phi \equiv (x = y)$?

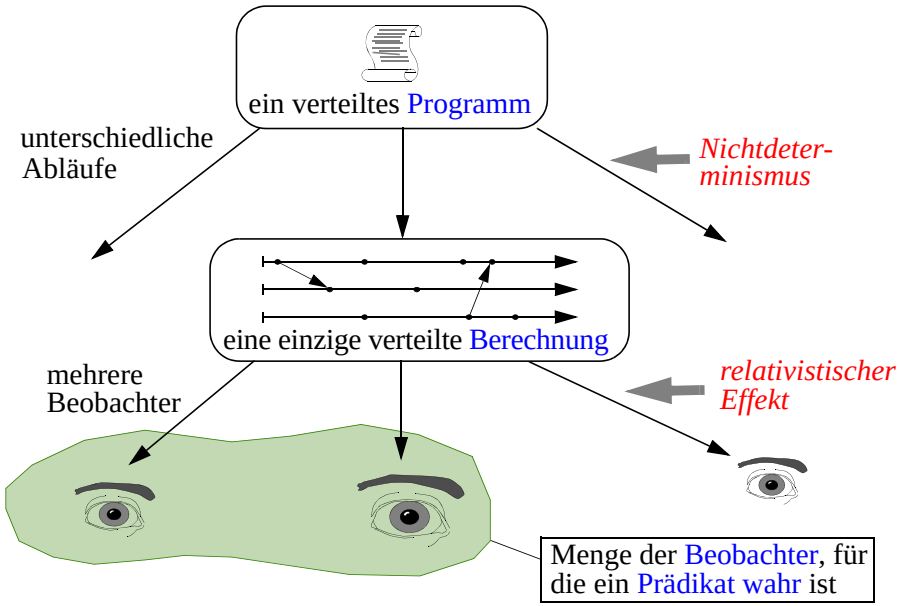


- Meldungen sind alle gleich schnell
- Beide (kausaltreuen!) Beobachtungen sind gleich "richtig"
- Die Beobachter stimmen bzgl. x = y nicht überein!

Aber was denn nun: gilt x=y in dieser Berechnung oder nicht?

"Possible Worlds"

- Verschiedene Beobachter sehen *verschiedene Wirklichkeiten*
  → Frage, ob ein bestimmtes Prädikat gilt, ist u.U. sinnlos!

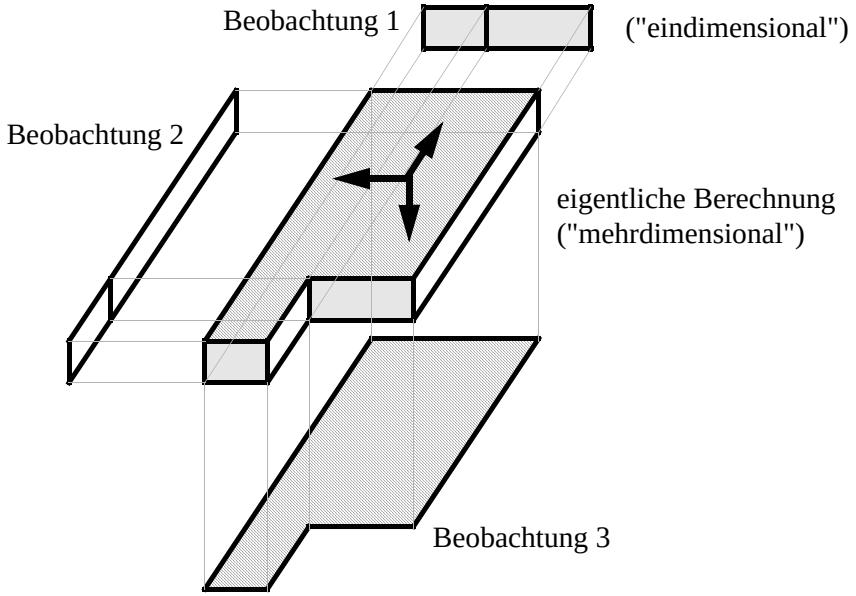


Konsequenz:

Es ist naiv (d.h. falsch!), einen verteilten Debugger zu entwickeln, mit dem man solche (im sequentiellen Fall "vernünftigen") Fragen beantworten kann!

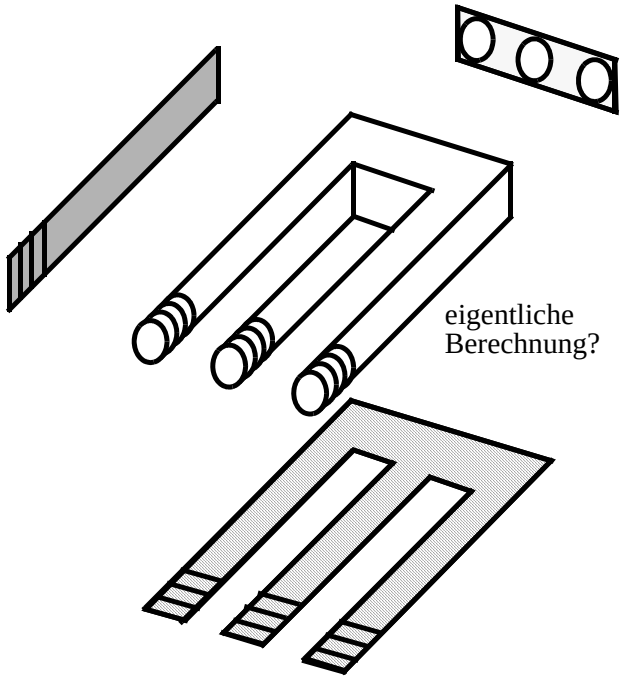
- im sequentiellen Fall: Berechnung = Beobachtung
- im verteilten Fall aber: Berechnung ≠ Beobachtung
- Gültigkeit von Prädikaten ist eine Eigenschaft einer Beobachtung, nicht einer Berechnung!
- gibt es sinnvolle beobachterinvariante Prädikate?

Beobachtung, Bild und Wirklichkeit



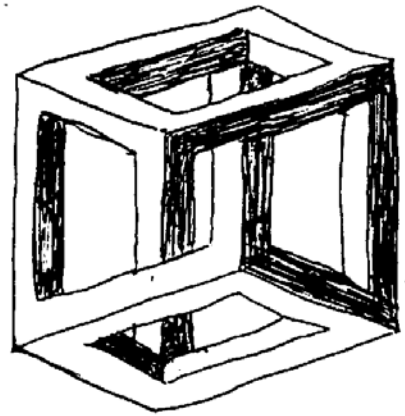
- Einige Eigenschaften gehen durch die Abbildung verloren
  - hier: kausale Unabhängigkeit
- Essentielle Eigenschaften sollen erhalten bleiben
  - hier: kausale Abhängigkeit
- Lässt sich die eigentliche Berechnung aus allen Beobachtungen rekonstruieren?
  - ja: Schnitt aller Linearisierungen ist Halbordnung selbst

Wirklichkeit ?

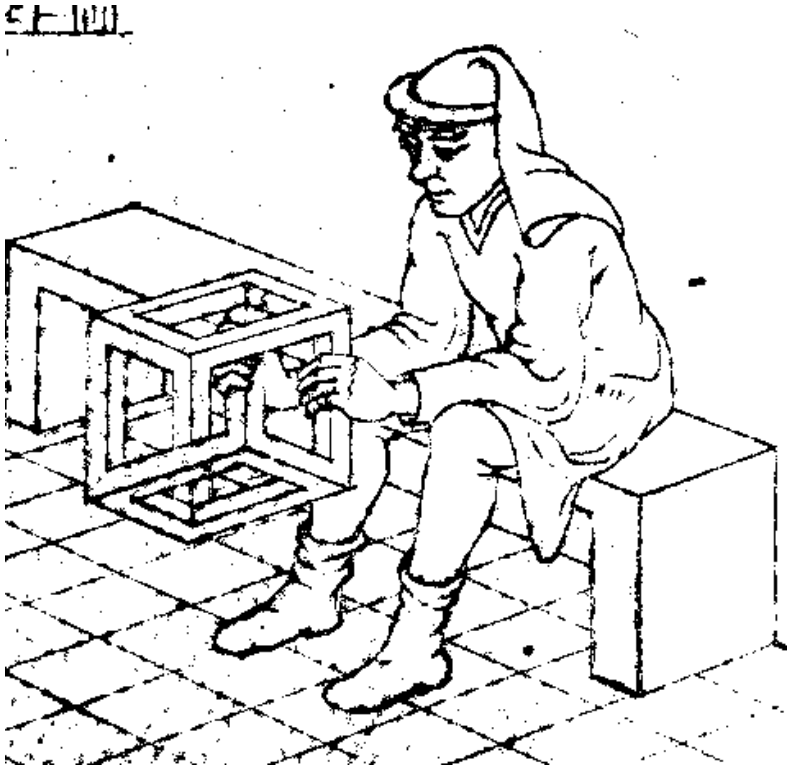


- Sind die verschiedenen Beobachtungen “verträglich”?
- Die beobachtete Wirklichkeit kann weitaus merkwürdiger sein, als eine Beobachtung vermuten lässt!
- Lassen sich die verschiedenen Beobachtungen zu einem konsistenten Bild zusammenfügen?

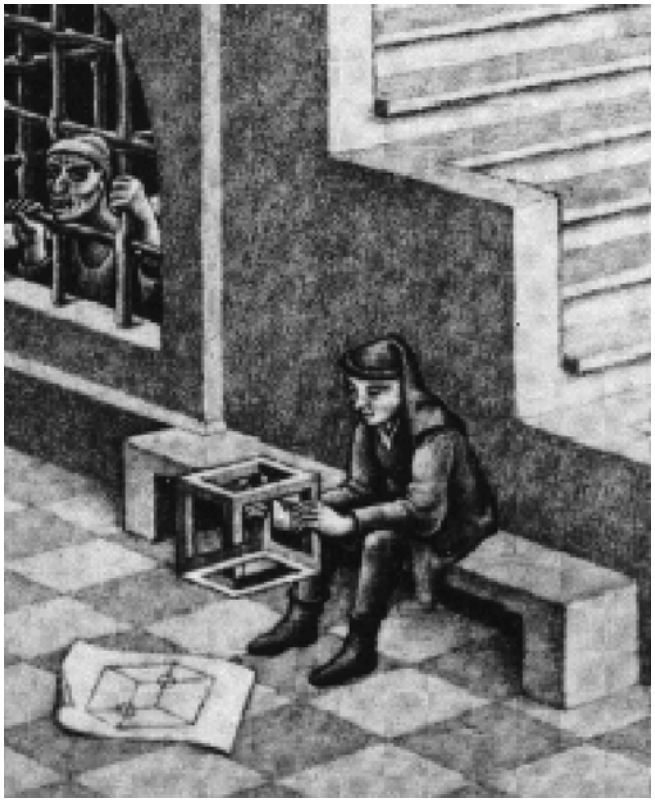
Ein inkonsistentes Bild



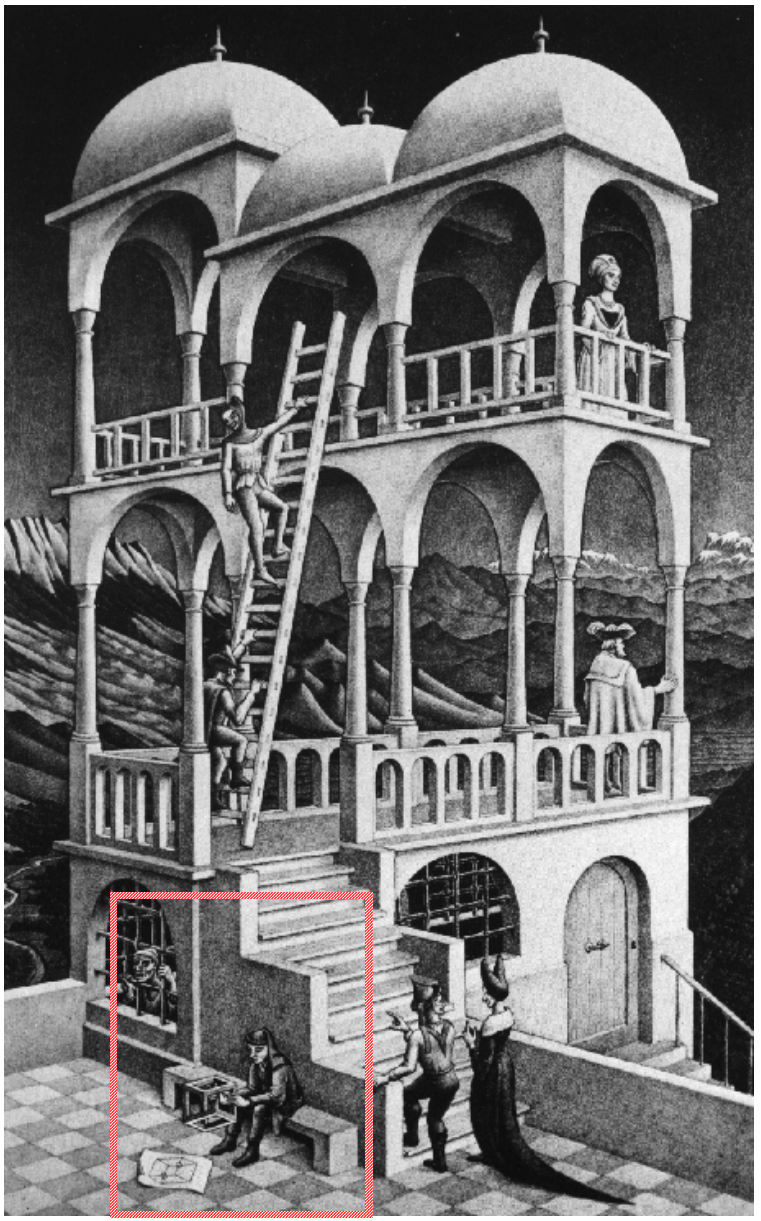
Wie heisst der Künstler?



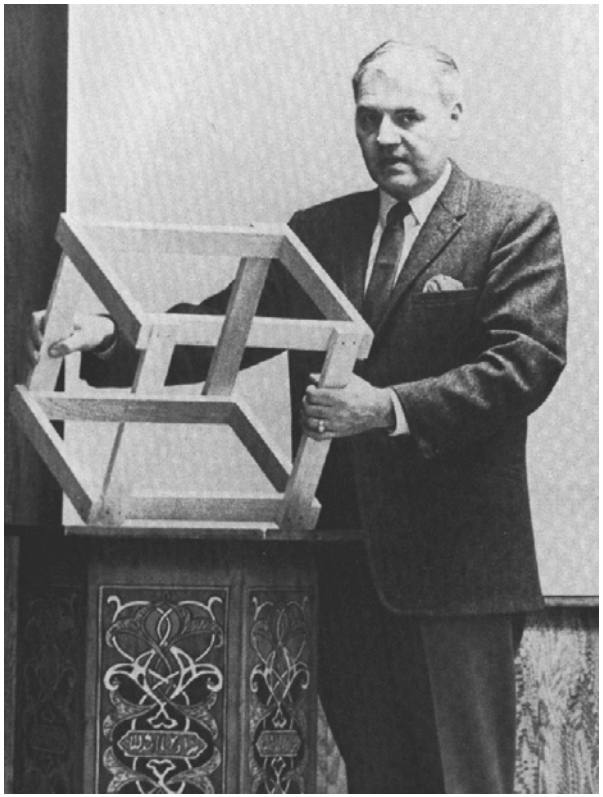
Wie heisst der Künstler?



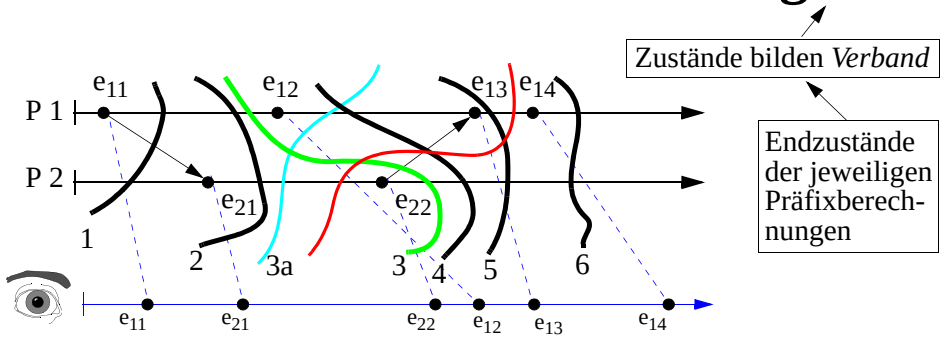
M.C. Escher: Belvedere (1958)



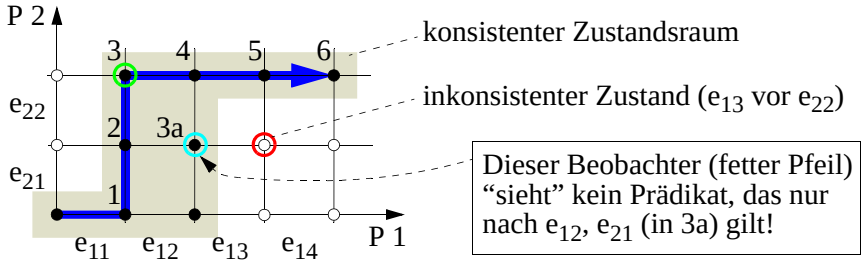
Ein Schnappschuss als Beweis



Das n-dimensionale Zustandsgitter



- *Beobachtung* =
  - Lineare Folge von Ereignissen
  - Folge damit assoziierter globaler Zustände



- *Beobachtung* = Pfad von links unten nach rechts oben
  (*kausaltreue Beobachtung*, wenn dieser sich nur im "erlaubten" Bereich aufhält)

- Ein Beobachter sieht alle *Ereignisse*, aber nur eine Teilmenge aller möglichen *Zustände*!
  → Beobachter kann Prädikate übersehen!

- Mit einem Schnappschussalgorithmus kann nur eine *einzige* (oft *lückenhafte*) von vielen Beobachtungen erlangt werden

Fragen zum Beobachtungsbegriff

1) Gibt es zu jedem kons. Zustand s einer Berechnung (mind.) einen Beobachter, der diesen "wahrnimmt"?

2) Seien s, s' zwei bel. (kons.) Zustände. Gibt es immer eine Beobachtung Obs mit $s \in Obs$ und $s' \in Obs$?

3) Können zwei konkurrente Zustände s, s' (d.h. weder s<s' noch s'<s) in der selben Beobachtung auftauchen?

(Ordnungsrelation entspr. Verband bzw. Präfixberechnung)

4) Es sei $s \in Obs$ und s später als s'. Ist dann $s' \in Obs$?

5) Sei $s \notin Obs$. Gibt es stets ein $s' \in Obs$, welches später als s ist? ... früher als s ist?

6) In einer Beobachtung Obs = $s_1,...,s_k$ sei $s_i \in Obs$ und $e \in s_i$ (Zustand aufgefasst als Menge der bis dahin vergangenen Ereignisse)

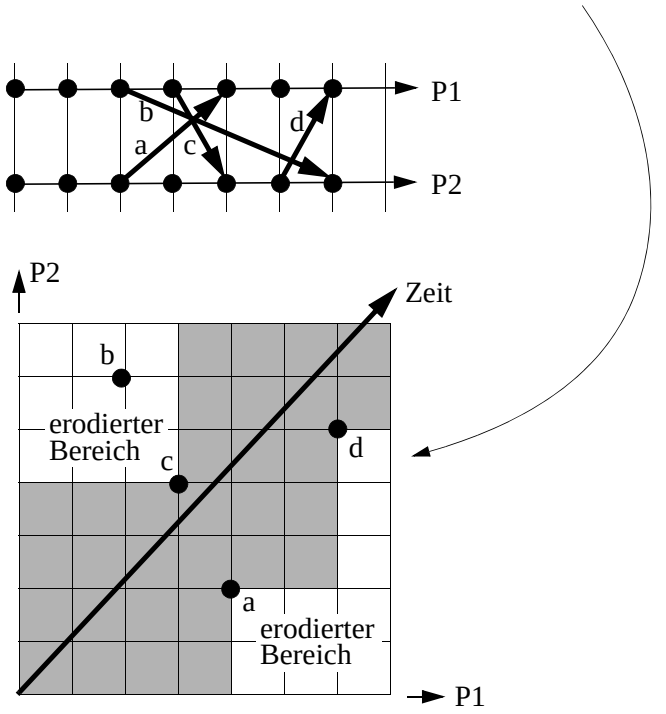
a) Gilt dann $e \in sj$ für alle $sj \in Obs$ mit $j > i$? Interpretation?

b) Es sei $e' < e$. Gilt dann $e' \in si$?

7) Sind alle Beobachtungen der selben (endlichen) Berechnung gleich lang?

Der erodierte Zustandshyperwürfel

- Hier: zwei Prozesse → 2-dimensionaler Würfel



- "Erosion" der inkonsistenten Zustände von den Ecken her

  - keine Nachricht wird empfangen, bevor sie gesendet wurde

  - ein Prozess blockiert in einer Empfangsanweisung, bis eine Nachricht verfügbar ist (und das zugehörige send somit ausgeführt wurde)

  - Nachrichten zur Synchronisation: sorgen dafür, dass kein Prozess sich zu schnell entfernt (zwingen Prozesse in den "Schlauch" von links unten nach rechts oben)

Der Verband konsistenter Zustände

- Zu jeder Präfixberechnung gehört ein konsistenter Zustand

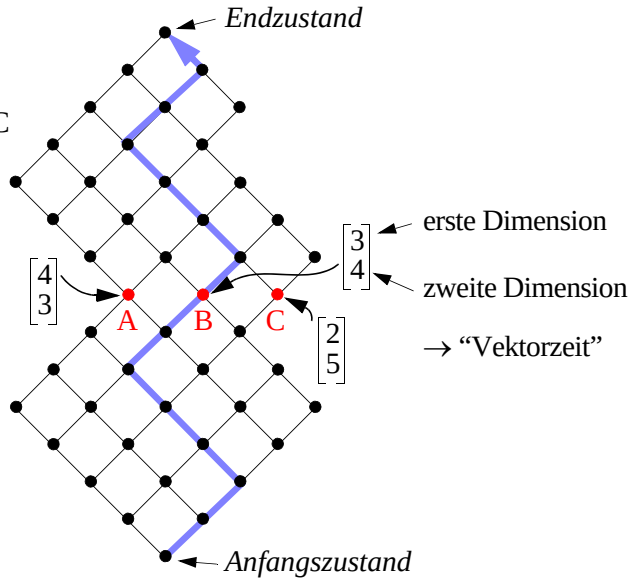
- Die "echte" Folge globaler Zustände ist ein Pfad durch den Verband (ist aber unbekannt, wenn keine exakte globale Zeit existiert)

- drei "gleichzeitige" globale Zustände A, B, C

- Frage, durch welchen davon die Berechnung verlief, ist sinnlos!

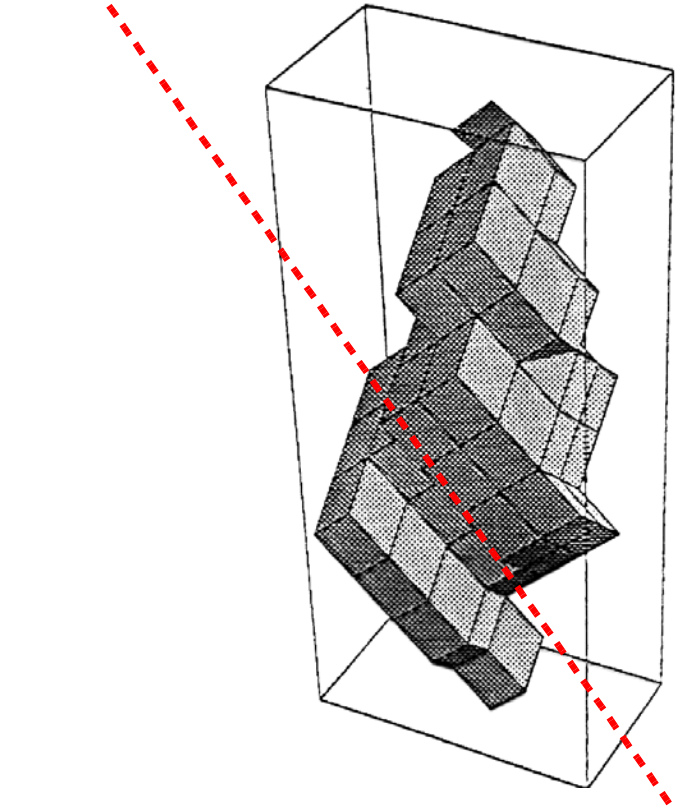
- Äquivalenzklasse [A, B, C] (alle Zustände mit sieben Ereignissen)

- man weiss nur, dass die Berechnung durch diese *Klasse* lief



- Konsistente Zustände bilden einen *Verband*
  - früherer, späterer globaler Zustand; abgeschlossen bzgl. "sup" und "inf"
  - visualisiert als eine kompakte Menge (ohne "Löcher")
  - Unterverband des Verbandes *aller* globalen Zustände

Eine 3-fach verteilte Berechnung

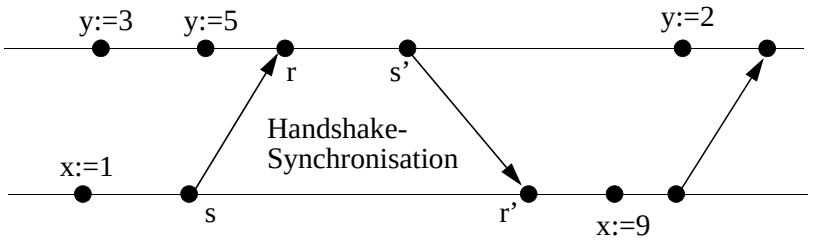


- 3-dimensionales Gebilde (als "kompakte Menge")
- Synchronisation → "Kante" oder "Kerbe" auf der Oberfläche
- "Engpässe" werden so sichtbar!

Kann man auch noch Berechnungen mit mehr Prozessen so visualisieren?

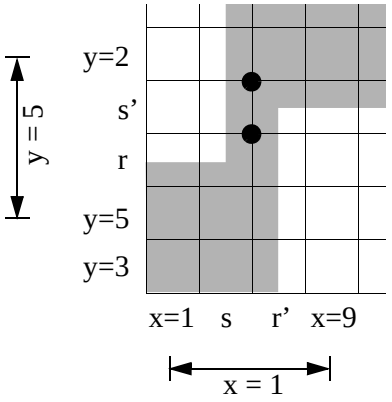
Synchronisationsengpässe

- Hier sieht *jeder* kausaltreue Beobachter "kurzzeitig" y=5, x=1:



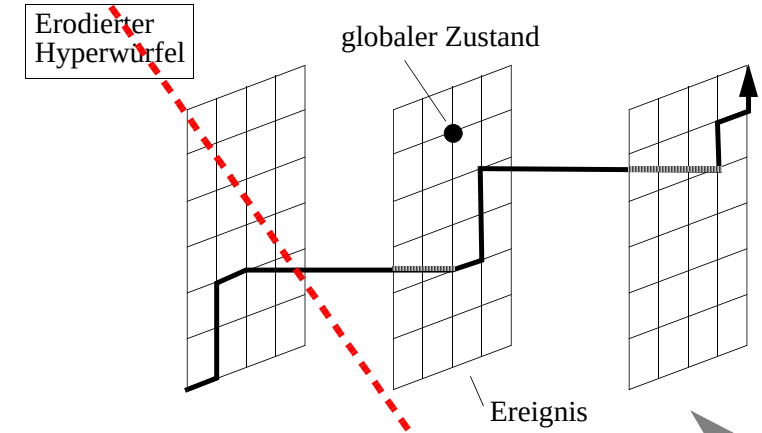
- Der Zustand y=5, x=1 wird also auf alle Fälle angenommen; er ist daher "unvermeidlich"

- Verallgemeinerung: "Barrier Synchronisation"
  (erst wenn alle Prozesse die Barriere erreicht haben, dürfen sie weiterlaufen)



Durch die beiden markierten Zustände muss jeder Beobachter; dort gilt sicherlich x=1, y=5

Dualismus der Diagramme



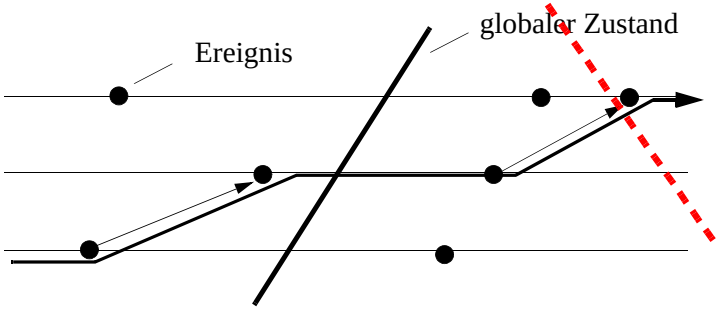
Hyper-ebene

Punkt → globaler Zustand
Schnitt(ebene) → Ereignis
Pfad → Kette von Zuständen

Beide Diagramme repräsentieren die gleiche Berechnung

Punkt → Ereignis
Schnitt → globaler Zustand
Pfad → Kette von Ereignissen

Zeitdia-gramm



Konsequenzen

- Prädikate gelten i.Allg. nur relativ zu Beobachtern, nicht für eine Berechnung als solche!
- Debugging: Nächster Schritt / Zustand nicht eindeutig
- Debugging: “stop when [condition]” i.Allg. sinnlos
- Polynomielle Zahl von Zuständen
- Exponentielle Zahl von Beobachtungen

→ i.Allg. ziemlich hoffnungslos!

- Ein einzelner Beobachter kann einen Zustand, in dem ein Prädikat gilt, "verpassen"

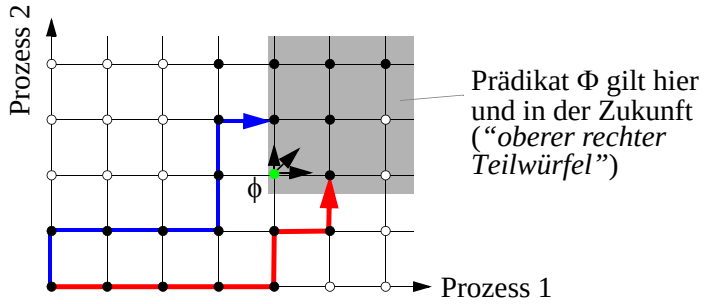
Genauere Spezifikation von Gültigkeit notwendig, z.B.:

- Prädikat Φ ist *aus Sicht von Beobachter X* erfüllt
- Prädikat Φ wird *von wenigstens einem* Beobachter wahrgenommen
- Prädikat Φ wird *von allen* Beobachtern wahrgenommen

Stabile Prädikate globaler Zustände

- Informell: "einmal wahr, immer wahr" (Monotonie)
- Def.: Φ stabil, wenn für $c1 < c2$ gilt: $\Phi(c1) \Rightarrow \Phi(c2)$

Halbordnung im Zustandsverband



- Sind *beobachterunabhängig*!
  - jeder Beobachter muss durch den oberen rechten Teilwürfel (Fairness...)
  - lassen sich daher einfach mit einer einzigen Beobachtung feststellen (jede andere Beobachtung wird Φ früher oder später ebenfalls entdecken)
- Für zwei Beobachtungen B_1, B_2 gilt: Falls B_1 das Prädikat Φ "entdeckt", dann gibt es einen *gemeinsamen späteren Zustand* (Verbandseigenschaft!) von B_1, B_2, bei dem Φ gilt
  - spätestens der Endzustand (bei endlichen Berechnungen)
  - B_1 kann z.B. die "echte" Ereignisfolge in Realzeit sein, B_2 eine Beobachtung
- Ein *gelegentlicher* (konsistenter) Schnappschuss genügt!
  - wenn der Schnappschussalgorithmus die Gültigkeit von Φ ermittelt, dann gilt Φ "jetzt" tatsächlich
  - wenn Φ "jetzt" gilt, dann meldet dies ein (jetzt gestarteter) Schnappschussalg.
- Es gibt wichtige stabile Prädikate, z.B. Terminierung, Garbage, Deadlock... (aber woher weiss man eigentlich, ob bzw. dass ein Prädikat stabil ist?)

Logische Zeit in verteilten Systemen

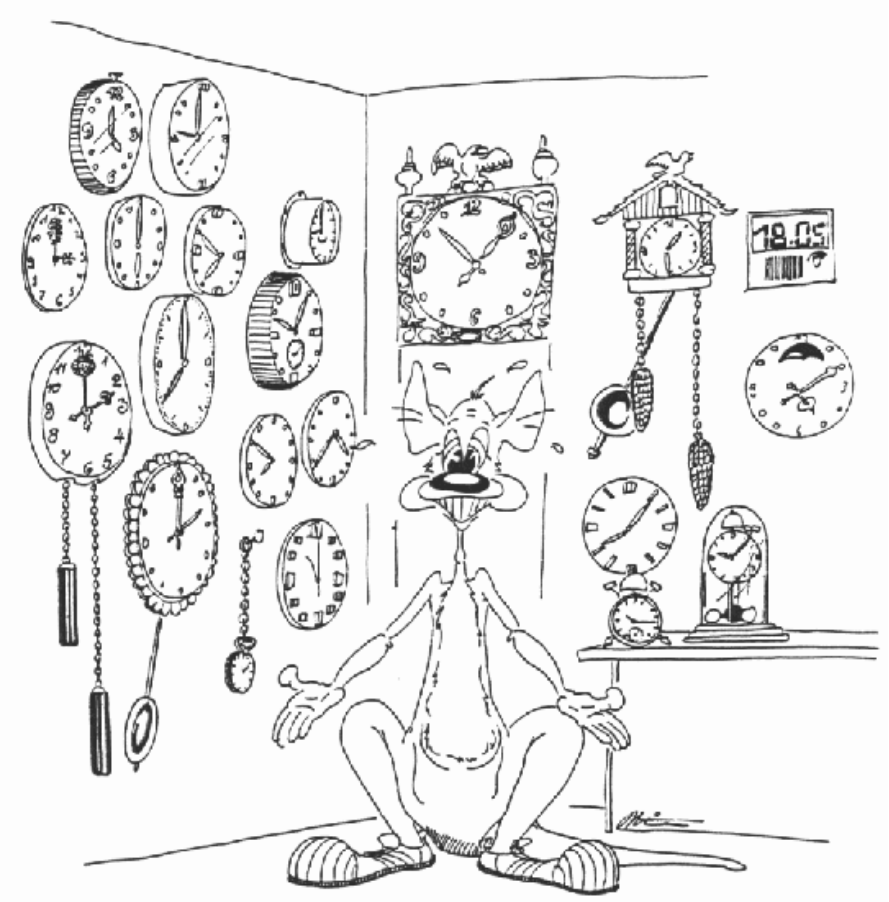


Bild: R. G. Herrtwich, G. Hommel

Zeit ?

Quid est ergo tempus?
Si nemo ex me quaerat, scio,
si quaerenti explicare velim, nescio.

Augustinus (354-430)

Was also ist die Zeit?
Wenn niemand mich danach fragt,
weiss ich's,
will ich's aber einem Fragenden erklären,
weiss ich's nicht.

Time is money.

Benjamin Franklin (1706-1790)

Time is how long we wait.

Richard Feynman (1918-1988, Nobelpreis Physik 1965)

Das im menschlichen Bewusstsein verschieden erlebte Vergehen von Gegenwart zu Vergangenheit sowie von erwarteter Zukunft zu Gegenwart.

Brockhaus 1983

Der Pfeil der Zeit - Vergangenheit, Gegenwart, Zukunft

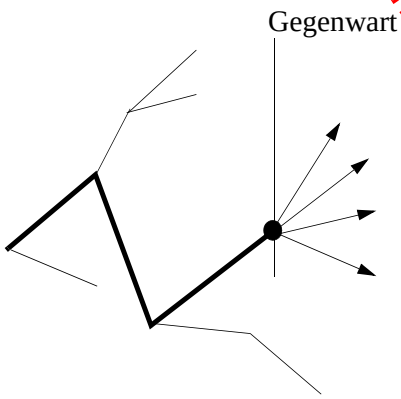
Die Zeit geht nicht, sie steht still,
Wir ziehen durch sie hin...

Gottfried Keller, Die Zeit geht nicht

tempus fugit

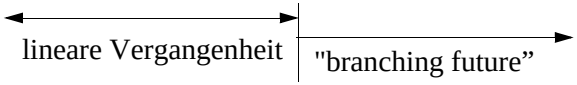
Time goes, you say?
Ah no! Alas, time stays, we go.

Austin Dobson, The Paradox of Time



Two roads diverged in a yellow wood,
And sorry I could not travel both.
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;
...
Then took the other, as just as fair,
...
I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I -
I took the one less traveled by,
And that has made all the difference.

Robert Frost (1874-1963)
The Road Not Taken (1916)



- *Rückblickend* scheint Zeit immer linear zu sein...

Zeit

...ist das häufigste (?) Substantiv der deutschen Sprache

- kommt aber in manchen Sprachen überhaupt nicht vor (z.B. Hopi)

Zeitalter	Arbeitszeit
Zeitdruck	Atomzeit
Zeitfrage	Endzeit
Zeitgeber	Frühzeit
Zeitgeist	Gleitzeit
Zeitgenosse	Gründerzeit
Zeitlupe	Halbwertszeit
Zeitnehmer	Jahreszeit
Zeitpunkt	Lebenszeit
Zeitraffer	Neuzeit
Zeitraum ⟷	Ortszeit
Zeitwende	Raumzeit
Zeitzeichen	Regierungszeit
	Spätzeit
	Steinzeit
	Tageszeit
	Teilzeit
	Uhrzeit
	Weltzeit

- Zeit kann man sparen, stehlen, rauben, nehmen, gewinnen, verlieren, absitzen, stoppen, verschwenden, vertreiben, totschlagen,...

"Die Zeit" über die "Zeit"

Stimmt es, dass "Zeit" das am häufigsten benutzte Substantiv der deutschen Sprache ist?

Jan Kny, Hamburg

Sosehr es dieser Zeitung schmeichelt: Es kommt darauf an, wer da was zählt. Tatsächlich hat ein Hobbylinguist namens Helmut Meier 1967 eine *Deutsche Sprachstatistik* veröffentlicht, in der auf Platz 90 der Liste der meistbenutzten deutschen Wörter und als erstes Substantiv das Wort Zeit steht - als nächste Hauptwörter folgen Herr, Jahre, Mann und Paragraph.

Fragt man dagegen beim Mannheimer Institut für deutsche Sprache nach, so erhält man eine ganz andere Liste, die aus einem 400 Millionen Wörter umfassenden Textkorpus ermittelt wurde: Die Top ten darin sind Jahr, Mark, Uhr, Prozent, Frau, Land, Million, Mensch, Tag und Stadt.

Paragraph und Prozent lassen stutzen und darauf schliessen, was für Texte die Sprachforscher in ihre Untersuchungen einbezogen haben mögen. Bei Herrn Meier findet man Gott und die Welt, Glaube und Gesetz ziemlich weit vorne - ein Hinweis auf juristische und theologische Quellen, wahrscheinlich hat der Mann die Bibel ausgezählt. Die Mannheimer Forscher haben vor allem Zeitungstexte ausgewertet. Was dagegen der Mann auf der Strasse sagt, protokolliert zum Glück niemand.

Christoph Drösser
Aus: "Die Zeit", 29.04.1999

Uhren

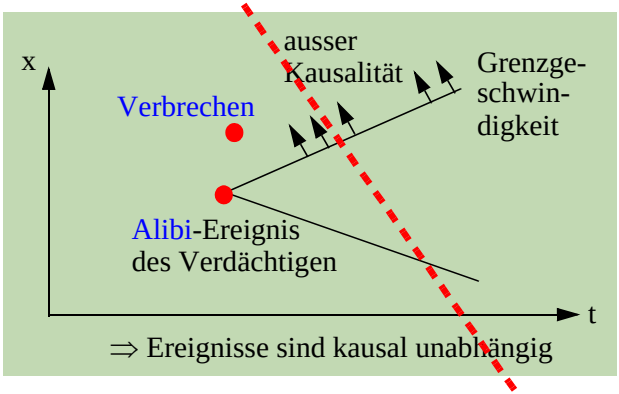
Ich halte ja eine Uhr für überflüssig.
Sehen Sie, ich wohne ja ganz nah beim Rathaus. Und jeden Morgen, wenn ich ins Geschäft gehe, da schau ich auf die Rathausuhr hinauf, wieviel Uhr es ist, und da merke ich's mir gleich für den ganzen Tag und nütze meine Uhr nicht so ab.

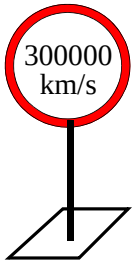
Karl Valentin

Kommt Zeit, kommt Rat

1. Volkszählung: Stichzeitpunkt in der Zukunft
   - liefert eine gleichzeitige, daher kausaltreue "Beobachtung"

2. Kausalitätsbeziehung zwischen Ereignissen ("Alibi-Prinzip")
   - wurde Y später als X geboren, dann kann Y unmöglich Vater von X sein
     → Testen verteilter Systeme: Fehlersuche/ -ursache





"speed limit of causality" (P. Langevin)

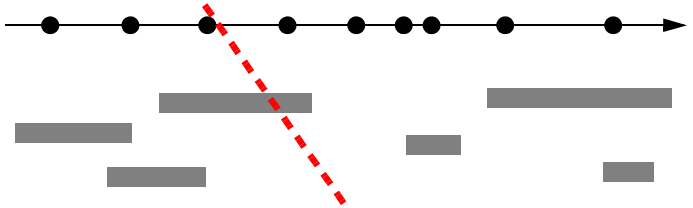
3. Wechselseitiger Ausschluss
   - bedient wird, wer am längsten wartet

4. Viele weitere nützliche Anwendungen in unserer "verteilten realen Welt"
   - z.B. kausaltreue Beobachtung durch "Zeitstempel" der Ereignisse

Le temps est un grand maître, il règle bien des choses.
Corneille, Sertorius

Zeitmodelle

- *Punkte* "in der Zeit" zusammen mit einer Relation "später"
- Oder: Zeit-*Intervalle* zusammen mit "später", "überlappt"...



Was ist das korrekte / geeignete Modell?

- Sind die beiden Modelle / Sichten kompatibel?
  - z.B. Start- und Endzeitpunkt bei Intervallen → Punktmodell?
- Ist das Zeitpunktmodell besser? Oder Zeitintervallmodell?
  - vgl. z.B. passe simple / imparfait im Französischen
  - wann tritt das Ereignis "Sonne wird rot" am Abend ein?

⇒ *was* möchte man mit so einem Modell *wie* und *wozu* modellieren?

Eigenschaften der "Realzeit"

Formale Struktur eines Zeitpunktmodells:

- transitiv
- irreflexiv } → lin. Ordnung
- linear
- unbeschränkt ("Zeit ist ewig": kein Anfang oder Ende)
- dicht (es gibt immer einen Zeitpunkt dazwischen)
- kontinuierlich
- metrisch
- homogen
- vergeht "von selbst" → jeder Zeitpunkt wird schliesslich erreicht

Typische Modelle: reelle Zahlen, rationale Zahlen (?)

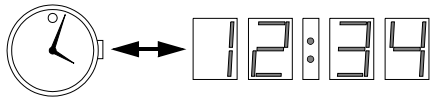
Welche Eigenschaften benötigen wir wirklich?

- was wollen wir mit "Zeit" anfangen?

→ "billigeren" Ersatz für fehlende globale Realzeit!

Zeit und Uhren in der Informatik

- Hardwarezähler als Uhren



→ Zeit ist *diskret* (nicht dicht)

- Uhrenüberlauf z.B. bei langen Simulationsläufen

→ Zeit ist nicht ewig sondern *beschränkt*

- Bei ereignisorientierter Sicht:

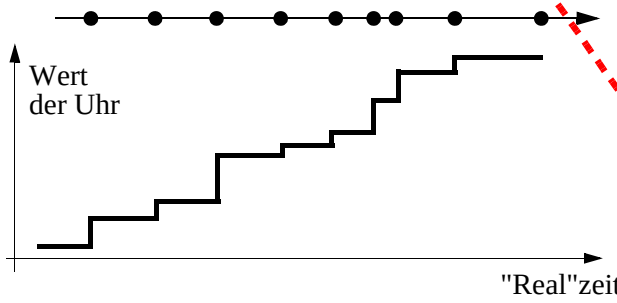
  - Zwischen zwei Ereignissen geschieht nichts

→ Uhren brauchen nicht kontinuierlich zu laufen

→ Uhren nur bei Ereignissen verändern

- "Weltsicht": Zeit *ist* das Stattfinden von Ereignissen

- Beispiel für diese Weltsicht: Ereignisgesteuerte Simulation



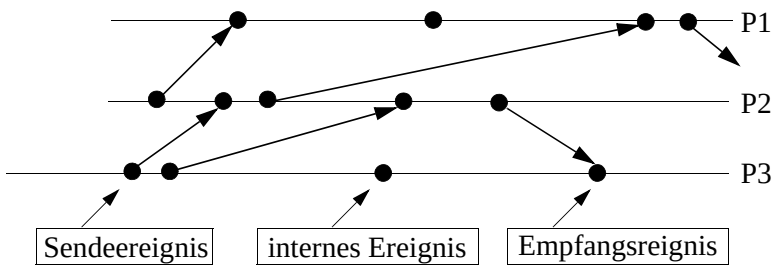
Nur bei den *Ereignissen* wird der Zustand der Welt betrachtet und gegebenenfalls verändert!

Wir nennen also Konzepte und Dinge "Zeit" und "Uhren", obwohl diese nicht alle idealen Eigenschaften besitzen!

was aber sind die *wesentlichen* Eigenschaften?

Kausalrelation

- "Happened before" (Lamport, 1978); zur Wiederholung:



- interessant: von links nach rechts verlaufende "Kausalitätspfade" (bestehend aus Nachrichtenpfeilen + Teilstücken auf Prozessachsen)

- *Kausalrelation* '<' auf der Menge E aller Ereignisse:

"Kleinste" transitive Relation auf E, mit **x** < **y** wenn:

1) **x** und **y** auf dem gleichen Prozess stattfinden und **x** vor **y** kommt, *oder*

2) **x** ist ein Sendeereignis und **y** ist das korrespondierende Empfangsereignis

- In einem Zeitdiagramm gilt für je zwei Ereignisse e, e' die Relation e < e' genau dann, wenn es einen Kausalitätspfad von e nach e' gibt

Logische Zeitstempel von Ereignissen

- Verteilte Berechnung abstrakt: n Prozesse, halbgeordnete Ereignismenge E, Nachrichten (Sende- / Empfangsereignis)

- Zweck: Ereignissen eine Zeit geben

- Gesucht: Abbildung C: E → H

Clock

"Zeitbereich": Halbgeordnete Menge → "früher", "später"

- Für e ∈ E heisst C(e) *Zeitstempel* von e

- C(e) bzw. e *früher* als C(e') bzw. e', wenn C(e) < C(e')

- Wie soll H aussehen?

z.B.:

- **N** (lineare Ordnung)
- **R** (bzw. Gleitpunktzahlen)
- Potenzmenge von E
- $\mathbf{N}^n$ (d.h. n-dim. Vektoren)

- Sinnvolle Forderung:

Kausalrelation ("Pfad im Diagramm")

Uhrenbedingung: e < e' ⇒ C(e) < C(e')

Ordnungshomomorphismus

Zeitrelation "früher"

Interpretation ("Zeit ist kausaltreu"):

Wenn ein Ereignis e ein anderes Ereignis e' beeinflussen kann, dann muss e einen kleineren Zeitstempel als e' haben

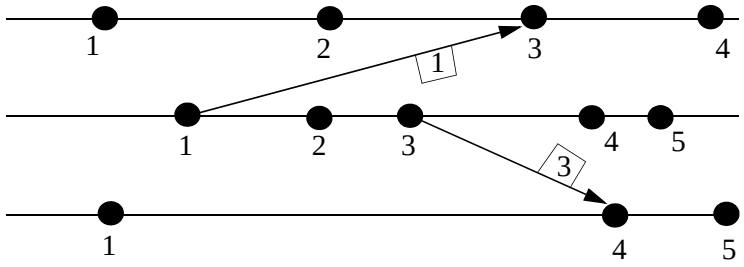
Logische Uhren von Lamport

Commun. ACM 1978:
Time, Clocks, and the Ordering of Events in a Distributed System

C: (E,<) → (**N**,<) Zuordnung von Zeitstempeln

Kausal-relation

e < e' ⇒ C(e) < C(e') Uhrenbedingung



Protokoll zur Implementierung der Uhrenbedingung:

- Lokale Uhr (= Zähler) *tickt* "bei" *jedem* Ereignis
- Sendeereignis: Uhrwert mitsenden (*Zeitstempel*)
- Empfangsereignis: *max*(lokale Uhr, Zeitstempel)

zuerst! danach "ticken"

Behauptung:
Protokoll respektiert Uhrenbedingung

Beweis: Kausalitätspfade sind monoton...

Kausale Unabhängigkeit

- *Def.* ["kausal unabhängig"]: a||b ⇔: ¬(a < b) ∧ ¬(a > b)

Beachte: || ist *nicht transitiv*! (Gilt eigentlich a||a ?)

- *Behauptung*: C(a) = C(b) ⇒ a||b

Bew.:

C(a) = C(b) ⇒ ¬(C(a) < C(b)) ∧ ¬(C(a) > C(b))

⇒ ¬(a < b) ∧ ¬(a > b) Verwende ¬(C(x) < C(y)) ⇒ ¬(x < y) aus der Uhrenbedingung!

⇒ a||b (Def.)

- Gilt die Umkehrung der Uhrenbedingung?

Nein, C(e) < C(e') ⇒ e < e' gilt nicht!
Es gilt nur: C(e) < C(e') ⇒ e < e' *oder* e||e'

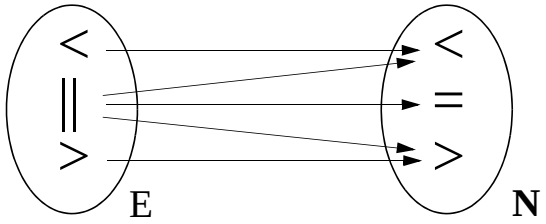
vgl. Beispiel

Das heisst:

Aus den Zeitstempeln lässt sich nicht (immer) schliessen, ob zwei Ereignisse kausal voneinander abhängig sind oder ob sie unabhängig sind!

⇒ Aber wozu sind solche Zeitstempel dann gut?

Lamport-Zeit: Strukturverlust



- Negation geht verloren
- Ordnungshomomorphismus, jedoch kein Isomorphismus
- E ist Halbordnung, N ist lineare Ordnung

→ zwei kausal unabhängige Ereignisse werden vermöge der Abbildung (scheinbar) vergleichbar!

Strukuturverlust ist ein wichtiger *Defekt*: Zweck der Zeitstempel ist, auf die Beziehung der Ereignisse zurückzuschliessen! Geht das besser? Anderer "Zeitbereich"?

Lamport-Zeit: Nicht-Injektivität

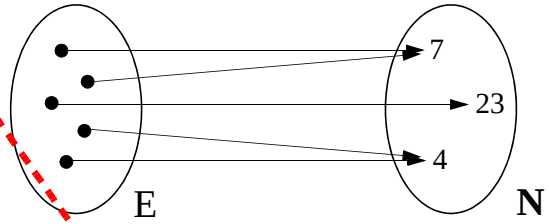


Abbildung ist nicht injektiv

- Wichtig z.B. für: "Wer die kleinste Zeit hat, gewinnt"
- Lösung:

  Lexikographische Ordnung (C(e),i), wobei i die Prozessnummer bezeichnet, auf dem e stattfindet

  Ist injektiv, da alle lokalen Ereignisse verschiedene Zeitstempel C(e) haben ("break ties")

  - *lin*. Ordnung $(a,b) < (a',b') \Leftrightarrow a<a' \vee a=a' \wedge b<b'$

  → alle Ereignisse haben *verschiedene* Zeitstempel

  → Kausalitätserhaltende Abb. $(E,<) \rightarrow (\mathbf{N}\times\mathbf{N}, <)$

Zu jeder Menge von Ereignissen gibt es nun ein eindeutig "frühestes"!

Vektorzeit: Motivation

Umkehrung der Uhrenbedingung gilt nicht für Lamport-Zeit

- $C(e) < C(e') \Rightarrow e < e'$ gilt nicht!
- es gilt nur: $C(e) < C(e') \Rightarrow e < e'$ oder $e \| e'$

Zeit := vergangene Zeit ← viele Uhren messen die Zeit, indem sie *vergangene* Sekunden zählen

:= Vergangenheit

:= Menge vergangener Ereignisse

Kausalrelation

Zeit(e) := $\{e' | e' \leq e\}$ = *Kegel* von e

Genauer: Zeitstempel eines Ereignisses e

Kann durch lokal späteste Ereignisse repräsentiert werden (linksabgeschlossen)

Hiervon gibt es n Stück (n = Anzahl der Prozesse)

!

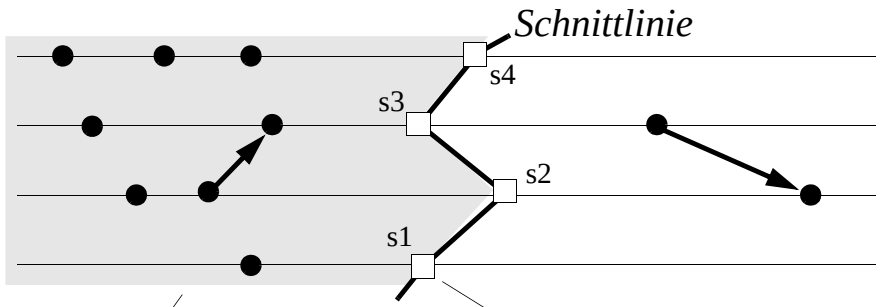
→ *Zeitstempel* ist n-dimensionaler Vektor

→ *Zeit* ist Menge n-dimensionaler Vektoren

→ *Uhr* ist ein array C[1:n]

zum Anzeigen von Zeitvektoren

Schnittlinien von Zeitdiagrammen



Vergangenheit V

$\{e \mid \exists s_i : e <_l s_i\}$ ($<_l$ ist die prozesslokale Kausalrelation)

- Schnittlinie trennt Zeitdiagramm / Ereignismenge in zwei disjunkte Mengen "Vergangenheit" / "Zukunft"

- Bemerkung: $e \in V \wedge e' <_l e \Rightarrow e' \in V$
  (linksabgeschlossen bzgl. lokaler Kausalrelation)

Denkübung: Man vergleiche den Begriff des *Schnittes* (insbes. des *konsistenten Schnittes*, vgl. nächste Seite) mit dem früher erwähnten Begriff der *Präfixberechnung*! Man beachte auch die Halbordnungsstruktur bzw. Verbandsstruktur dieser Begriffe.

Schnitte und konsistente Schnitte

Def. **Schnitt**:

$S \subseteq E$ heisst *Schnitt* von E, falls $e \in S \wedge e' <_l e \Rightarrow e' \in S$

(d.h. Schnitt wird mit seiner Vergangenheit identifiziert)

Def. **konsistenter Schnitt**:

$S \subseteq E$ heisst *konsistent*, falls $e \in S \wedge e' < e \Rightarrow e' \in S$

Def.: Schnitt S *später* als S' $:\Leftrightarrow$ $S' \subseteq S$

bzw. $\subset$ bei "strikt später"

Schreibweise auch: $S' < S$

Beh.: Jeder konsistente Schnitt ist ein Schnitt

Bew.: $<_l \subseteq <$

Bem.: Schnitt(linie) inkonsistent $\Leftrightarrow$
$\exists$ "Nachricht aus der Zukunft"

Bem.: Schnitt(linie) konsistent $\Leftrightarrow$
Schnittereignisse paarweise kausal unabhängig

Bew. als Übung

Bem.: Schnitt(linie) konsistent $\Leftrightarrow$
lässt sich senkrecht darstellen (Gummibandtransf.)

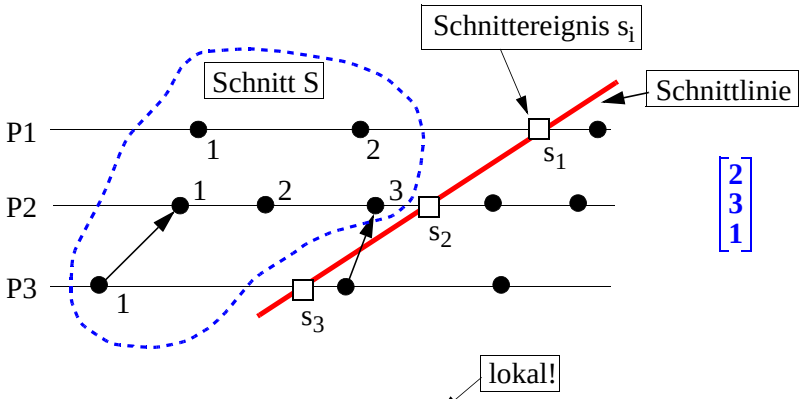
Bew. bereits bekannt: Diagramm auseinanderschneiden und versetzen

Vektorzeitstempel von Schnitten

Zeitstempel $\tau(S)$ eines Schnittes S ist ein Vektor aus $\mathbf{N}^n$

alle Ereignisse, die links von einer Schnittlinie liegen

Anzahl der Prozesse



lokal!

Def. $\tau(S)[i] := |\{e \in E_i \mid e <_l s_i\}| := |S \cap E_i|$

(für jede Komponente i mit $1 \leq i \leq n$)

Interpretation:

$\tau(S)[i]$ ist die "Stelle", wo die Prozessachse von P_i durch die Schnittlinie geschnitten wird

Beachte: man kann sowohl zu *konsistenten* als auch *inkonsistenten* Schnitten den zugehörigen Zeitvektor definieren!

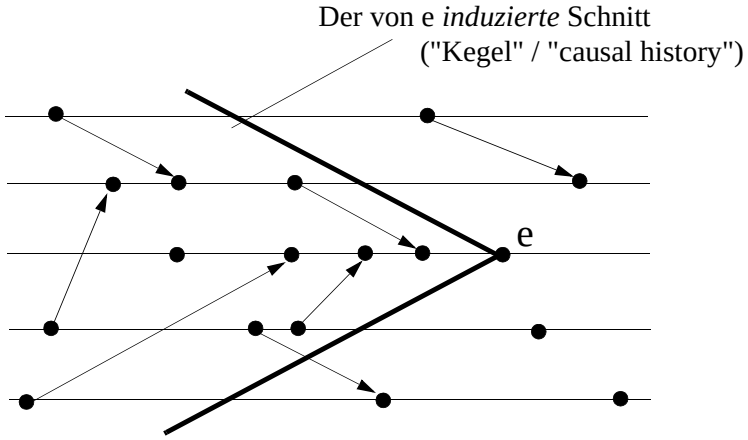
Kausale Vergangenheit eines Ereignisses

Def. *kausale Vergangenheit* $\downarrow(e)$ eines Ereignisses e:

$$\downarrow(e) = \{e' \mid e' \leq e\}$$

Beh.: $\downarrow(e)$ ist ein konsistenter Schnitt

Bew. als Übung



Beh.: $e' \leq e \Leftrightarrow e' \in \downarrow(e)$

Beh.: $e \parallel e' \Leftrightarrow \neg(e \in \downarrow(e')) \wedge \neg(e' \in \downarrow(e))$

(Bew. klar)

Vektorzeitstempel von Ereignissen

Kausale Vergangenheit

- Bezeichne $\downarrow e$ den *Kegel* $\{e' \in E \mid e' \leq e\}$ von e

  - jeder Kegel ist ein konsistenter Schnitt (da linksabgeschlossen)
    ("Kegelmantel" könnte also als senkrechte Linie gezeichnet werden!)

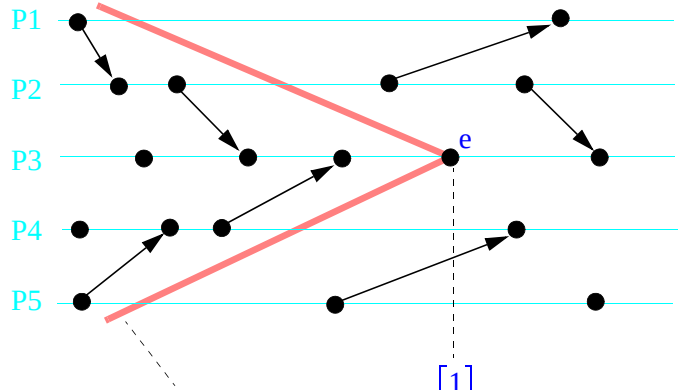
  - Repräsentation durch die n lokal am weitesten rechts liegenden Ereignisse

- Dann *definiere* $\tau(e) := \tau(\downarrow e)$

Menge der Ereignisse von P_i

Also: $\tau(e)[i] := |\{e' \in E \mid e' \leq e\} \cap E_i| := |\{e' \in E_i \mid e' \leq e\}|$

  - das heisst: Zeitstempel eines Ereignisses = Zeitstempel seines induz. Kegels



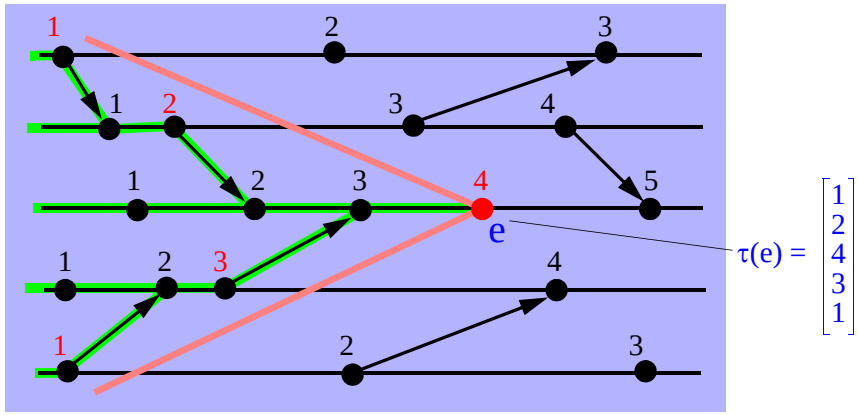
Schnittlinie in der Form eines Kegels (vgl. "Lichtkegel" der relativistischen Physik): $\downarrow e$ ist ein konsistenter Schnitt

$$\begin{bmatrix} 1 \\ 2 \\ 4 \\ 3 \\ 1 \end{bmatrix}$$

$\tau(e)$: Vektorzeitstempel von e

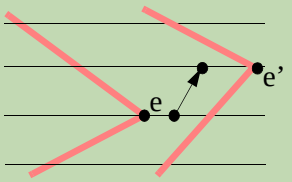
$\tau(e) < \tau(e')$

- Jeder Prozess numeriert seine Ereignisse lokal durch

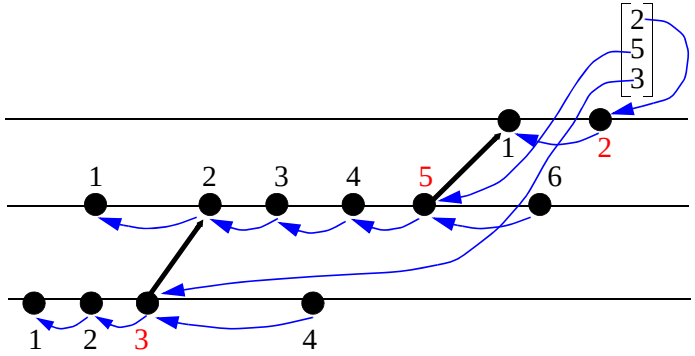


- Vektor $\tau(e)$ repräsentiert gesamte kausale Vergangenheit des Ereignisses e
  - Veranschaulichung durch einen "Kegel" $\{e' \mid e' \leq e\}$
  - Nummer des jeweils lokal letzten kausal vorangehenden Ereignisses steht in der jeweiligen Komponente des Vektors

- Interpretation von $\tau(e) < \tau(e')$:
  - e liegt in der kausalen Vergangenheit von e'
  - Kegel von e ist ganz im Kegel von e' enthalten



Vektorzeitstempel: Interpretation



- Zeigt auf jeweils jüngstes kausal vergangenes lok. Ereignis
- Damit implizit auch auf alle vorangehenden
  (wegen lokaler totaler Ordnung)
- Vektor repräsentiert *gesamte kausale Vergangenheit*
- Kodiert "*Wissen*" über (jedes einzelne) vergangene Ereignis
  Genauer: Vektorzeit repräsentiert die Kausalrelation in isomorpher Weise!

Denkübungen:

- wie stellt man fest, ob e' im Kegel von e mit $\tau(e)$ liegt?
- gibt es eine noch kompaktere Kodierung?

"Zeitstempelmathematik": ≤, ||, sup

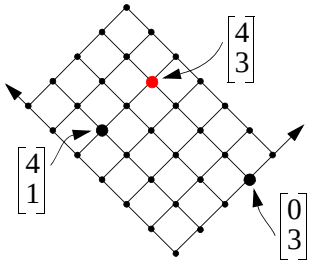
$$\begin{bmatrix}1\\3\\4\\3\\2\end{bmatrix} \leq \begin{bmatrix}1\\7\\4\\6\\2\end{bmatrix} \qquad \begin{bmatrix}1\\3\\4\\3\\7\end{bmatrix} \parallel \begin{bmatrix}5\\3\\8\\3\\2\end{bmatrix}$$

wenn komponentenweise ≤ "konkurrent"

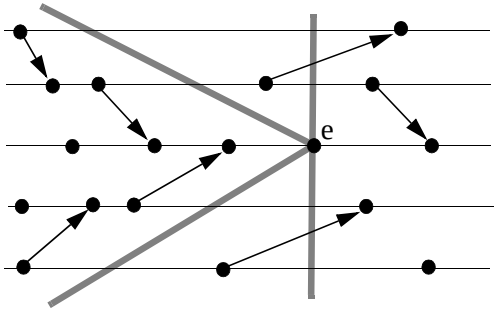
Bemerkung: '<' ist dann definiert als: ≤, aber ≠

$$\sup\left(\begin{bmatrix}1\\4\\2\\3\\7\end{bmatrix}, \begin{bmatrix}8\\3\\4\\3\\2\end{bmatrix}\right) = \begin{bmatrix}8\\4\\4\\3\\7\end{bmatrix}$$

sup = komponentenweises Maximum



Vektorzeit und ideale Beobachter



$$\begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix} \ldots \begin{bmatrix}2\\4\\5\\4\\3\end{bmatrix}$$

Wahrnehmungen des idealen Beobachters

$$\tau(e) = \begin{bmatrix}1\\2\\4\\3\\1\end{bmatrix} \qquad id(e) = \begin{bmatrix}1\\3\\4\\3\\2\end{bmatrix}$$

- Numeriere Ereignisse lokal
- Idealisierter Beobachter nimmt "Ticken" sofort wahr
- Geeignete Datenstruktur hierfür: Vektor / array
- Für *jeden* Beobachter gilt stets: $\tau(e) \leq id(e)$ $(\forall e)$
  (komponentenweise '≤' ; idealer Beobachter sieht stets die gesamte kausale Vergangenheit sowie evtl. einige weitere Ereignisse)

- $\tau(e)$ = Infimum aller idealen Sichten id(e)
- Beachte: id(e) hängt vom Zeitdiagramm ab!
- Aber $\tau(e)$ ist invariant bzgl. Gummibandtransformation!

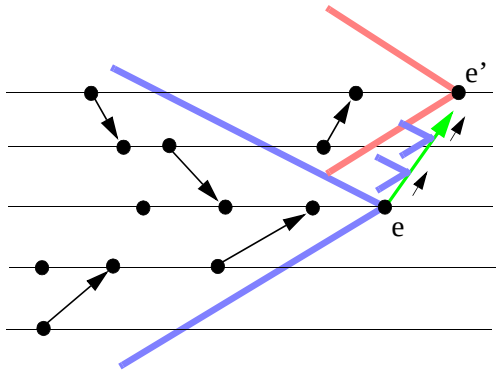
Daher nur zur Motivation

Implementierung der Vektorzeit

- Idee: Analog zur Lamport-Zeit
  (hier allerdings stets vektoriell!)

- Nachrichten enthalten die gesamte kausale Vergangenheit des Senders ⇒ Zeitvektor des Sendeereignisses

- Bei Empfang einer Nachricht:
  - Vereinigung der Kegel ← Wissen über vergangene Ereignisse vereinigen
  - d.h. Supremum der Vektoren



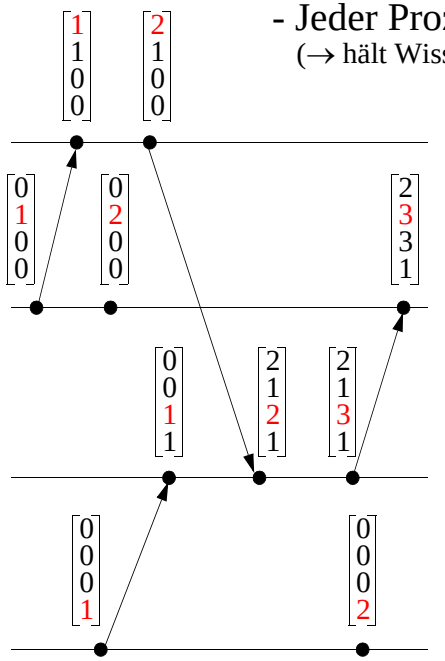
Mitschleppen des Kegels des Sendeereignisses und Vereinigung mit dem Kegel des Empfangsereignisses

→ "induktiv": ein Ereignis hat ein "vollständiges Wissen" über alle seine vergangenen Ereignisse

Propagieren des Zeitwissens

(→ Implementation der Vektorzeit)

- Jeder Prozess besitzt eine *Vektoruhr*
  (→ hält Wissen über vergangene Ereignisse)



- *bei jedem Ereignis:*
  eigene Komponente erhöhen
- *beim Senden:*
  neuen Vektor mitsenden
- *beim Empfangen:*
  komponentenweises Maximum der beiden Vektoren

Vereinigung der beiden Kegel

Kausalrelation — bzgl. Zeitvektor

- *Behauptung*: $e < e' \Leftrightarrow \tau(e) < \tau(e')$

- *Anschauliche Interpretation*:
  - $\tau(e) \leq \tau(e') \Leftrightarrow$ es gibt eine Kausalkette von e zu e'

monoton bzgl. Zeitvektoren!

- *Korollar*: $e \parallel e' \Leftrightarrow \tau(e) \parallel \tau(e')$

Interpretation: Genau die "gleichzeitigen" Ereignisse beeinflussen sich nicht ggs.

Kausal- und Zeitstruktur: Isomorphie

"Hauptsatz": $e < e' \Leftrightarrow \tau(e) < \tau(e')$

- Umkehrung der Uhrenbedingung
- gilt nicht für Lamport-Zeit

Beweis:

(1) $e \leq e' \Leftrightarrow e \in \downarrow e'$
wegen Def. von Kegel

(2) $e \in \downarrow e' \Leftrightarrow \downarrow e \subseteq \downarrow e'$
klar nach Def. von Kegel (= kons. Schnitt)

(3) $\downarrow e \subseteq \downarrow e' \Leftrightarrow \tau(\downarrow e) \leq \tau(\downarrow e')$
weil "später" sich jeweils überträgt

(4) $\tau(\downarrow e) \leq \tau(\downarrow e') \Leftrightarrow \tau(e) \leq \tau(e')$
nach Def. von $\tau(e)$

Irreflexivität folgt aus Injektivität von τ

Anschauliche Interpretation: Wissen über lokale Zeit über Kausalkette propagiert

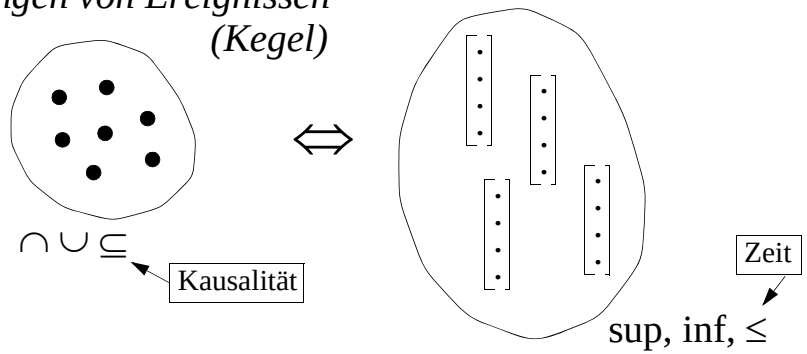
---

Verschärfung: Falls $e \in E_i$ (und $e \neq e'$):
$e < e' \Leftrightarrow \tau(e)[i] < \tau(e')[i]$

Effizienter: Nur eine Komponente testen

Rechnen mit Ereignismengen

Mengen von Ereignissen (Kegel) ⇔ *Zeitvektoren*



Mengentheoretische Operationen ⇔ Algebraische Operationen ($\rightarrow$ "rechnen")

Verbandsstruktur auf 2^E (Ideale) ⇔ Produduktverband auf $\mathbf{N}^n$

Ordnungstheoretische Eigenschaften ⇔ Algebraische Eigenschaften

---

Vektorzeit und Vektoruhren ermöglichen eine "operationale Manipulation" der Kausalrelation

Eigenschaften der Vektorzeit

$\tau'(e) := \sum_i \tau(e)[i]$ hat Eigenschaften der Lamport-Zeit:

1) $e < e' \Rightarrow \tau'(e) < \tau'(e')$ (Uhrenbedingung)
2) Umkehrung gilt nicht
3) τ' ist nicht injektiv
4) $\tau'(e)$ ist ein Skalar

Beachte: $\tau'(e)$ ist das "Volumen" des Kegels, die Lamportzeit die längste vorangehende Kette von e.

Beh.:
$\tau(e) < \tau(e') \Rightarrow t(e) < t(e')$

Realzeitpunkte

Bew.: $\tau(e) < \tau(e') \Rightarrow e < e' \Rightarrow t(e) < t(e')$

Bem.: Gilt nicht für die Lamport-Zeit!

Frage: Gibt es kompaktere Zeitstempel als Vektoren der Länge n? (Für die auch die Umkehrung der Uhrenbedingung gelten.)

nicht leicht!

Wäre für die Praxis sehr wichtig (Zeitstempel in Nachrichten können unangenehm lang werden...)

Anwendungen der Vektorzeit

<Momo trifft Professor Hora>:

> Und überall hingen, lagen und standen Uhren. Da gab es auch *Weltzeituhren in Kugelform, welche die Zeit für jeden Punkt der Erde anzeigten...*
> "Vielleicht", meinte Momo,
> "braucht man dazu eben so eine Uhr."
> Meister Hora schüttelte lächelnd den Kopf.
> "Die Uhr allein würde niemand nützen.
> *Man muss sie auch lesen können."*
>
> *Michael Ende, Momo*

- Debugging
  - Lokalisierung von Fehlern (“kann [nicht] Ursache sein...)
  - Race conditions; Synchronisationsfehler (kausale Unabhängigkeit)
  - Effizientes Replay

!

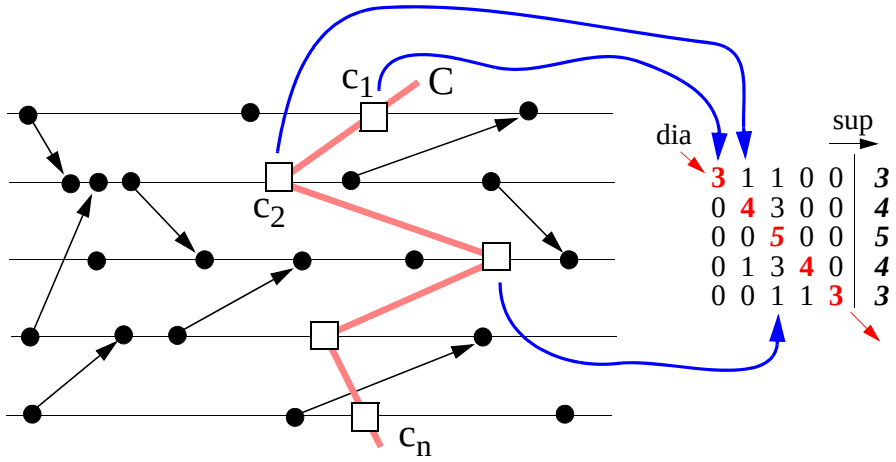
- Leistungsanalyse
  - “Flaschenhals” im Zeitverband; Synchronisationsgrad
  - Kausal unabhängige Ereignisse können parallel ausgeführt werden
- Implementierung konsistenter Schnappschüsse
  - Menge lokaler Schnappschüsse mit paarweise konkurrenten Ereignissen
- Realisierung von kausaltreuen Beobachtern
  - Causal broadcast
  - Causal order

Schnittmatrix

- Schnittmatrix $ eines Schnittes C (mit Schnittereign. c_i):

$\$(C) := (\tau(c_1), \tau(c_2),...,\tau(c_n))$

d.h. Schnittereignisvektoren c_i als Spaltenvektoren



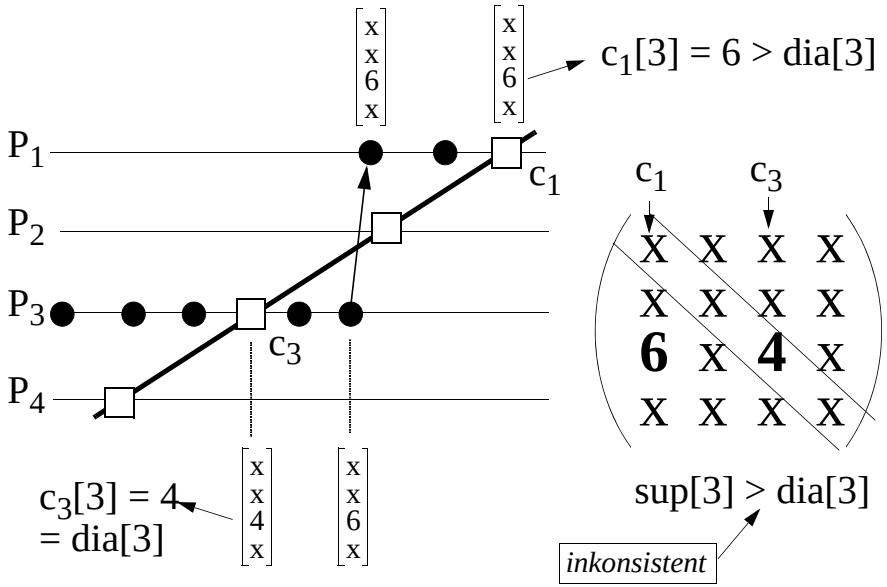
Frage: Kann man an den Schnittmatrizen etwas über die Schnitte erkennen? (z.B.: ob später, früher; ob konsistent...)

C konsistent $\Leftrightarrow$ dia($) = sup($)

Diagonalvektor — Zeilenmaximum

(d.h. Maximum einer Zeile ist stets das Diagonalelement)

Das “sup = dia”-Konsistenzkriterium



Ein Prozess (P_1) verschieden von P_3 weiss (bei c_1) etwas über lokale Ereignisse auf P_3, von denen P_3 selbst noch nichts weiss (d.h. die *nach* c_3 geschehen)

$\Leftrightarrow$

Es gibt einen Pfad von einem Ereignis auf P_3 *nach* c_3 zu einem Ereignis *vor* c_1

$\Leftrightarrow$

[Generalisierung über alle Indizes i≠j]

Der Schnitt ist inkonsistent

Konsistenzkriterium

Beweis hier ohne "anschauliche" Zeitdiagramme

- Beh.: S inkonsistent ==> $S \neq S^*$
- Bew.: S^* ist stets konsistent (vgl. oben)

- Korollar: S inkonsistent ==> $\tau(S) \neq \tau(S^*)$
  (Versch. Schnitte haben versch. Zeitstempel)

- Beh.: S inkonsistent ==> $dia(\$) \neq sup(\$)$
- Bew.: $dia(\$) = \tau(S)$ und (Verträglichkeit)
  $sup(\$) \overset{Def.}{=} sup(\tau(s_1),...,\tau(s_n)) = \tau((\downarrow s_1) \cup ... (\downarrow s_n)) \overset{Def.}{=} \tau(S^*)$.
  Wende nun obiges Korollar an.

- Beh.: S konsistent ==> $dia(\$) = sup(\$)$
- Bew.: $\downarrow s_i$ liegt ganz in S, d.h. $\downarrow s_i \subseteq S$

Denn: 1) $x \in \downarrow s_i$ ==> $x \leq s_i$
2) $y \in S \wedge x \leq y$ ==> $x \in S$
Wegen $s_i \in S$: $x \in \downarrow s_i$ ==> $x \in S$
Also $\downarrow s_i \subseteq S$

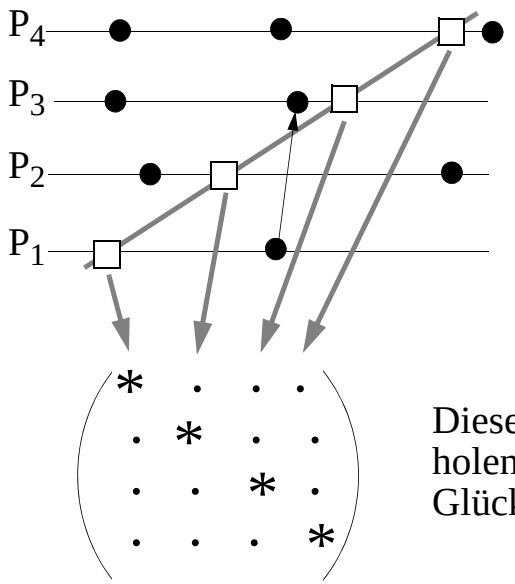
==> $S^* \subseteq S$. Umkehrung gilt sowieso:
==> $S^* = S$ (vgl. vorh. Lemma)
==> $\tau(S^*) = \tau(S)$ ==> $sup(\$) = dia(\$)$.

Daraus folgt das Konsistenzkriterium:

S konsistent $\Leftrightarrow$ $dia(\$) = sup(\$)$

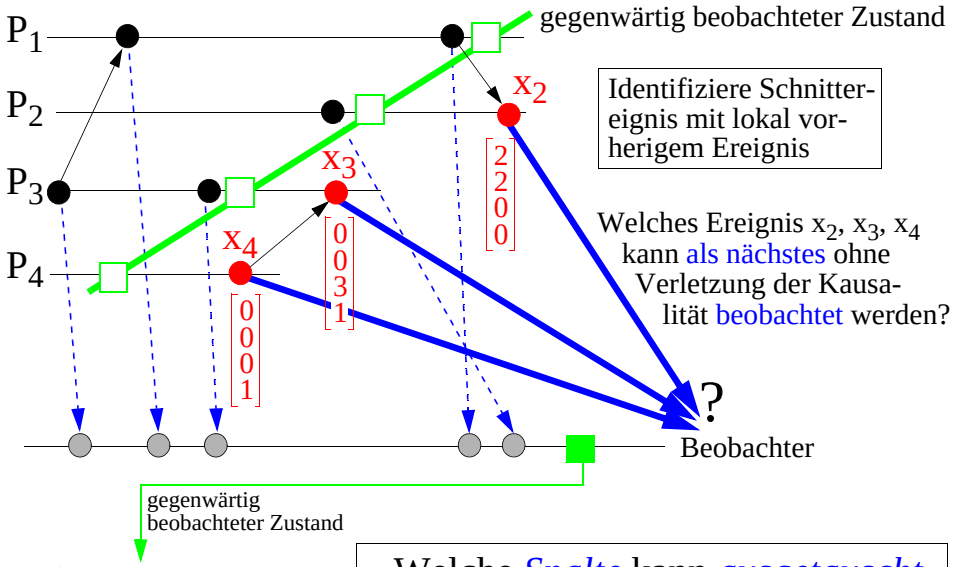
Implementierung konsistenter Schnappschüsse mit Vektorzeit?

Ein erster Ansatz: Alle Prozesse auffordern, ihren lokalen Zustand zu senden und testen, ob konsistent:



Dieses solange wiederholen, bis man einmal Glück hat...

Realisierung kausaltreuer Beobachter



$$\begin{pmatrix} \mathbf{2} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 1 & 0 & \mathbf{2} & 0 \\ 0 & 0 & 0 & \mathbf{0} \end{pmatrix}$$

- Welche *Spalte* kann *ausgetauscht* werden? (x_2, x_4, aber nicht x_3)
- Beobachter merkt sich dia($); es muss *Zeitstempel* ≤ *dia($)* sein (ausgenommen die Diagonalkomponente)

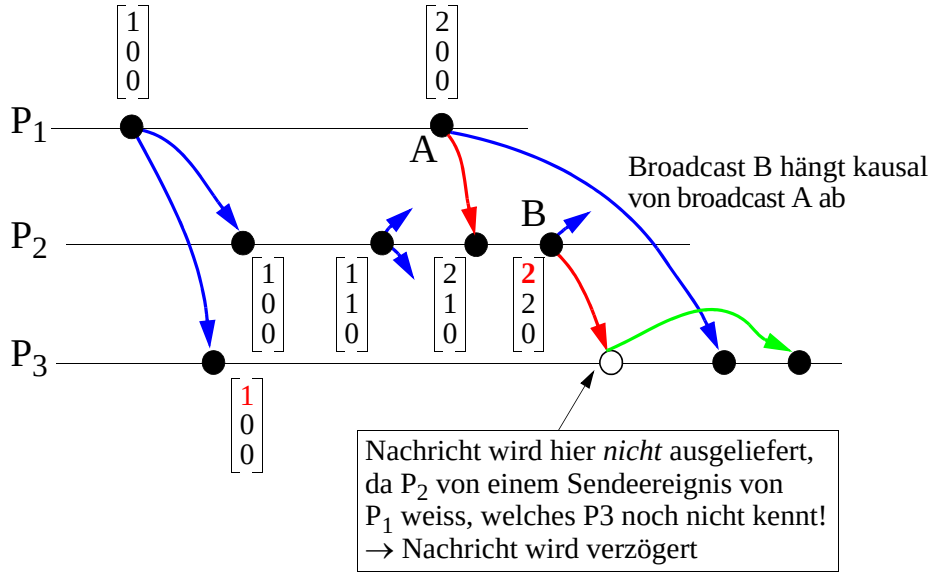
- Strategie: dia($) = sup($) → stets konsistent halten!
- Beobachter benötigt *nur* den *Diagonalvektor*, keine Matrix, um momentanen Zustand zu beschreiben

---

Prinzip: Verwende Vektorzeit, um (indirektes) *Wissen* über "kausal frühere" Sendeereignisse zu kodieren:

"Dieses Ereignis hängt von einem anderen ab, das ich eigentlich erhalten haben müsste; also warte ich das andere erst ab..."

Implementierung von kausalem Broadcast



- *Prinzip*: Verwende die Vektorzeit, um (indirektes) Wissen über "kausal frühere" Ereignisse zu kodieren

  -"dieses Ereignis hängt von einem anderen ab, das ich eigentlich erhalten haben müsste; also warte ich das andere erst ab..."

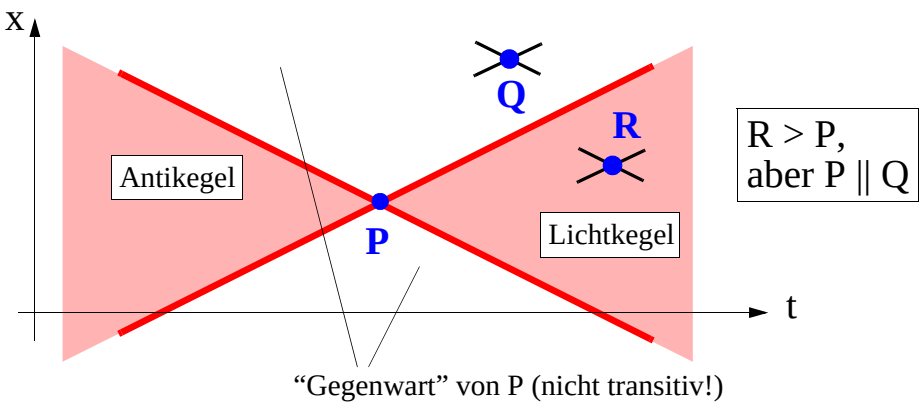
- Nur Broadcast-Ereignisse sind hier relevant

- Verallgemeinerung des Sequenznummerverfahrens zur Implementierung von FIFO bei Nicht-FIFO-Kanälen

---

Denkübung: Vektoren sind relativ aufwendig: Geht kausaler Broadcast, causal order, kausaltreue Beobachtung auch mit weniger aufwendigen Datenstrukturen?

Vektorzeit und Minkowski-Raumzeit



Raumzeit	*Vektorzeit*
Halbordnung	Halbordnung
2-dimensionale Kegel bilden Verband (bzgl. Schnitt)	Zeitvektoren bilden Verband (sup)
Lorentz-Transformation lässt Lichtkegel invariant	Gummiband-Transformation lässt Kausalrelation invariant
Raumzeitkoordinaten ermöglichen Test, ob potentiell kausal abhängig: Mit u= (x_1, t_1), v= (x_2, t_2) prüfe $c^2(t_2-t_1)^2 - (x_2-x_1)^2 >= 0$	Zeitvektoren ermöglichen einfachen Test, ob potentiell kausal abhängig: (prüfe, ob in allen Komponenten kleiner)

"Von Stund an sollen Raum für sich und Zeit für sich völlig zu Schatten herabsinken, und nur noch eine Art Union der beiden soll Selbständigkeit bewahren."

Resümee: Themen der Vorlesung

- Beispiele für verteilte Berechnungen und Algorithmen
  - verteilte ggT-Berechnung
  - verteilte Approximation
- Grundalgorithmen
  - Flooding
  - Echo-Algorithmus (Wellenalgorithmus, Spannbaum)
- Verteilte Terminierung
  - Doppelzählverfahren
  - Zeitzonenverfahren
  - für synchrone Kommunikation: DFG-Verfahren
  - Kreditmethode

Grundphänomen "inkonsistenter Sicht"; problemspezifische Lösungen dafür

- Wechselseitiger Ausschluss
  - Grundprinzipien
  - Maekawa (sqrt(n))
  - Token-basierte Verfahren

Synchronisation in vert. Sys. (viele wollen, einer darf; Sicherheit, Deadlockfreiheit, Fairness)

- Election
  - Chang/Roberts-Verfahren (Ring); bidirektionale Varianten
  - Peterson's Algorithmen: O(n log n) worst case
  - Election auf Bäumen
  - untere Schranke O(e) für Nachrichtenkomplexität bei allg. Netzen
  - Election in anonymen Netzen (probabilistische Algorithmen)

Symmetriebrechung in vert. Sys.: verteilte Wahl eines "Repräsentanten"

Resümee: Themen (2)

- Garbage-Collection
  - Mutator, collector, Formalisierung
  - Behind the back copy
  - Verteiltes Garbage-Collection
  - Referenzzähler (verschiedene Lösungen; z.B. WRC, LRC)
  - Implementierungstechniken

- Garbage-Collection $\Rightarrow$ Terminierungserkennung

- Wellenalgorithmen
  - Eigenschaften, Spannbäume,...
  - sequentiell (Token traversiert Graph) oder parallel (z.B. Echo-Algorithmus)
  - Methode von Tarry (1895) als Verallgemeinerung von depth-first

Resümee: Themen (3)

- Schnappschuss, Konsistenz, Beobachtungen, Prädikate,...
  - Kausalrelation, kausale Vergangenheit...
  - Halbordnung, Verband,...
  - Schnitt, globaler Zustand
  - Kausal konsistente Beobachtung
  - Globale Prädikate, stabile Prädikate
  - Schnappschussalgorithmen

- Logische Zeit
  - Uhrenbedingung
  - Lamport-Uhren
  - Vektorzeit: Eigenschaften und Implementierung
  - Schnittmatrix
  - Konsistenzkriterium
  - Implementierung von kausaltreuen Beobachtern