

Folienkopien zur Vorlesung

Verteilte Algorithmen

F. Mattern

Departement Informatik, ETH Zürich

WS 05/06 (3-stündig inkl. Übungen)

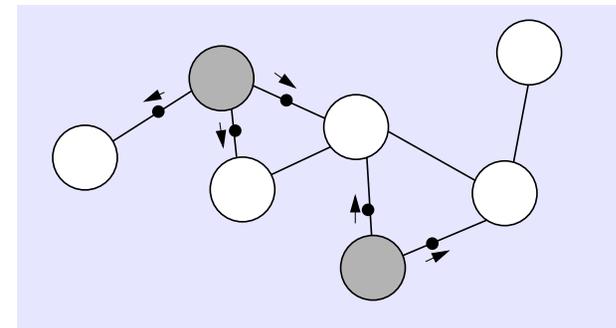
© F. Mattern, 2005

Verteilte Algorithmen

F. Mattern

Departement Informatik

ETH Zürich



Verteilte Algorithmen

= Algorithmen für / in verteilten Systemen

Grundkonzept der Informatik
(praktisch / theoretisch)

Interessantes Gebiet der
praktischen Informatik
sowie wichtiges Anwendungsfeld

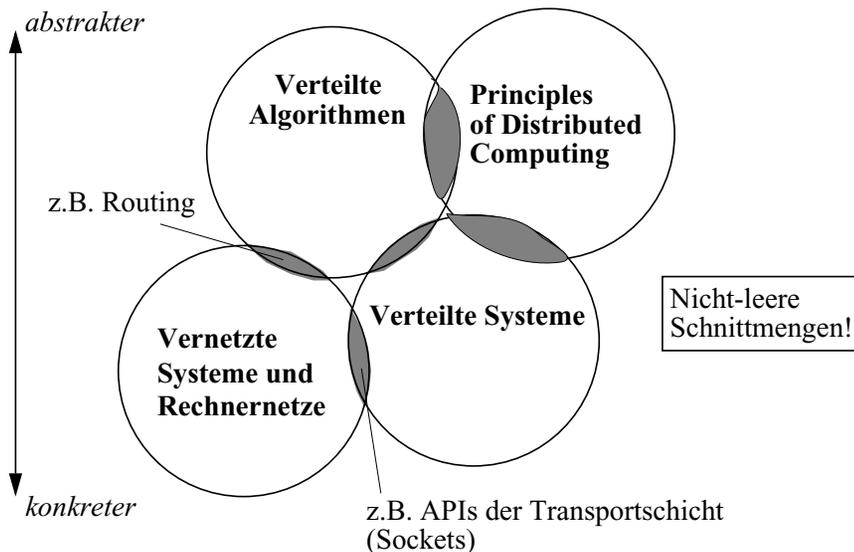
*Achtung: Prüfungsrelevant ist der gesamte
Stoff der Vorlesung, nicht alleine der Inhalt
dieser Foliensammlung!*

- Was ist "verteilt" an einem verteilten Algorithmus?

Vorläufige Antwort: *Zustand* und *Kontrolle* auf verschiedene "Orte"

Einordnung der Vorlesung

- 3-stündige Vorlesung (inkl. Übungen)
- Sinnvolle Vorkenntnisse:
 - Verteilte Systeme (< 10% Überschneidung)
 - Grundkenntnisse der Informatik und Mathematik (Vordiplom)
- Folienkopien im Internet (pdf-Format):
 - www.vs.inf.ethz.ch/edu/WS0506/VA/



- **Mobile Computing**
- **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.a. 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.

Literatur:

- F. Mattern: Verteilte Basisalgorithmen. Springer-Verlag
- G. Tel: Topics in Distributed Algorithms. Cambridge Univ. Press
- G. Tel: Introduction to Distributed Algorithms. Cambridge University Press, 2nd edition
- [] V. Barbosa: An Introduction to Distributed Algorithms, MIT Press
- N. Lynch: Distributed Algorithms, Morgan Kaufmann
- V. K. Garg: Principles of Distributed Systems, Kluwer

Artikel aus Fachzeitschriften (wird in der Vorlesung bekanntgegeben).

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...

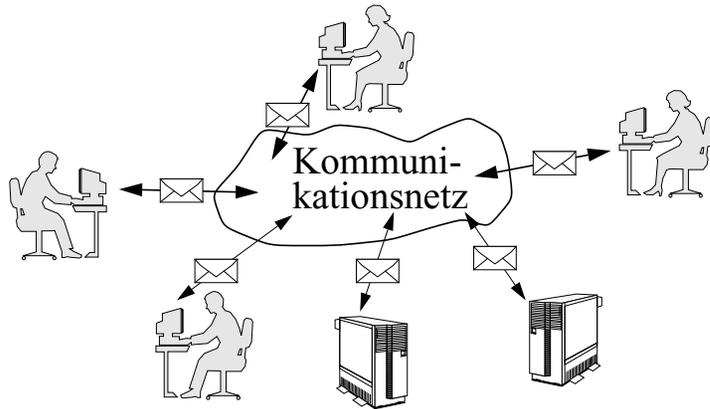
-
- Techniken, Einsichten, Zusammenhänge,...
 - Basisverfahren, grundsätzliche Probleme,...
 - Abstraktion, Modellbildung, Formalisierung,...
 - Problemlösungstechniken, Analysetechniken,...
 - Qualitätsbewertung, Komplexitätsabschätzung,...
 - Verifikationstechniken

(verallgemeinerbare)
Erkenntnisse

Notwendige mathematische Grundbegriffe (Relationen)

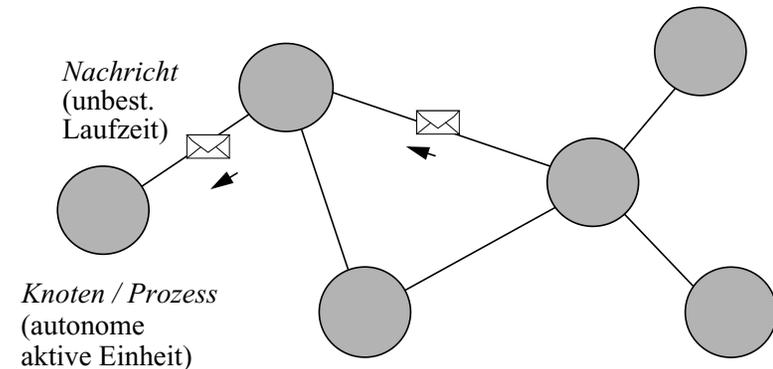
- Kartesisches Produkt, Relation, binäre Relation
- Halbordnung, lineare (totale) Ordnung
- Äquivalenzrelation
- Transitiv, reflexiv, symmetrisch, asymmetrisch, antisymmetrisch
- Infimum, Supremum, Verband, Hasse-Diagramm
- Bitmatrix, Adjazenzmatrix von Graphen
- Reflexiv-transitive Hülle
- Prädikat

Verteiltes System



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

Verteiltes System II



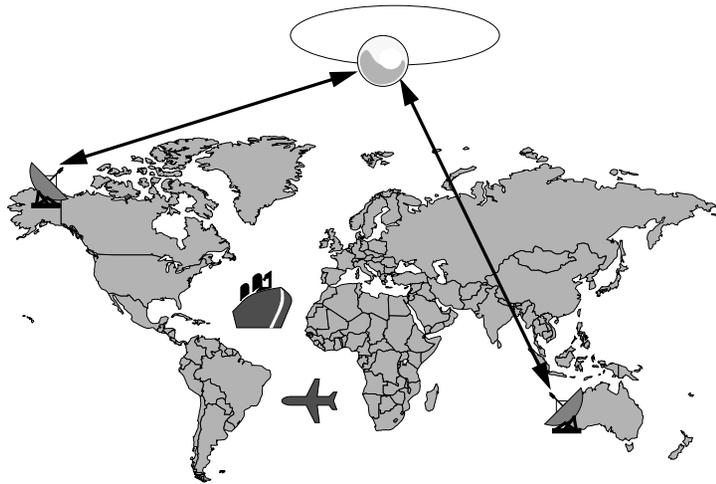
Physisch verteiltes System:

Mehrrechnersystem ... Rechnernetze

Logisch verteiltes System: Prozesse (Objekte, Agenten)

- Verteilung des Zustandes (keine globale Sicht)
- Keine gemeinsame Zeit (globale, genaue "Uhr")

Verteiltes System III

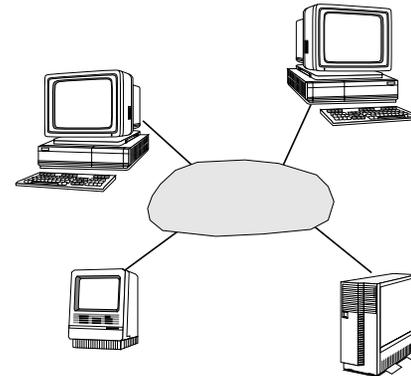


Auch die "reale Welt" ist ein verteiltes System:

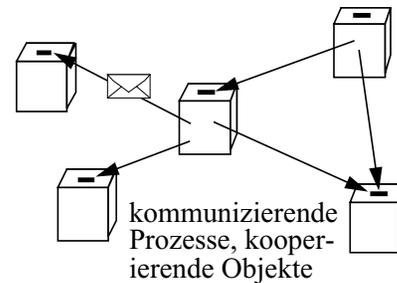
- Viele gleichzeitige ("parallele") Aktivitäten
- Exakte globale Zeit nicht erfahrbar / vorhanden
- Keine konsistente Sicht des Gesamtzustandes
- Kooperation durch explizite Kommunikation
- *Ursache* und *Wirkung* zeitlich (und räumlich) getrennt

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

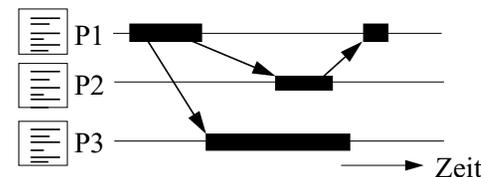
Sichten verteilter Systeme



- Rechnernetz mit Rechenknoten:
- Multicomputer (Parallelrechner)
 - LAN = Local Area Network
 - WAN = Wide Area Network
 - Routing, Adressierung....



Objekte eines Betriebssystems bzw. einer Programmiersprache
 ⇒ "Programmierersicht" (Client, Server...)



- Aktionen, Ereignisfolgen
- Konsistenz, Korrektheit

zunehmende Abstraktion

Parallele ↔ verteilte Algorithmen

Sequentieller Algorithmus



Paralleler Algorithmus

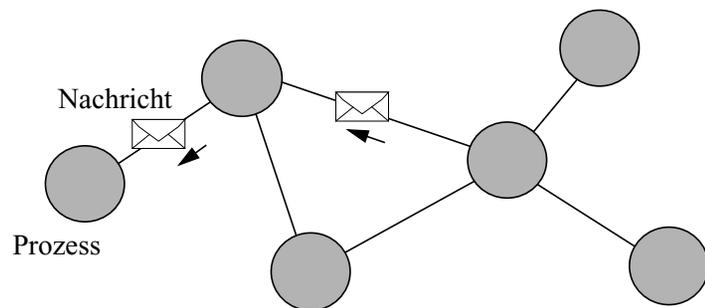
- mehrere, oft gleichartige Prozesse
- Synchronisation oft über gemeinsamen Speicher
- Zweck: Beschleunigung gegenüber seq. Algorithmus



Verteilter Algorithmus

- mehrere Prozesse kommunizieren über Nachrichten
- gemeinsames Ziel → Kooperation
- ideal: keine zentrale Kontrolle

↑
oft asynchron



Aspekte verteilter Systeme

im Vergleich zu *sequentiellen* Systemen:

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



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

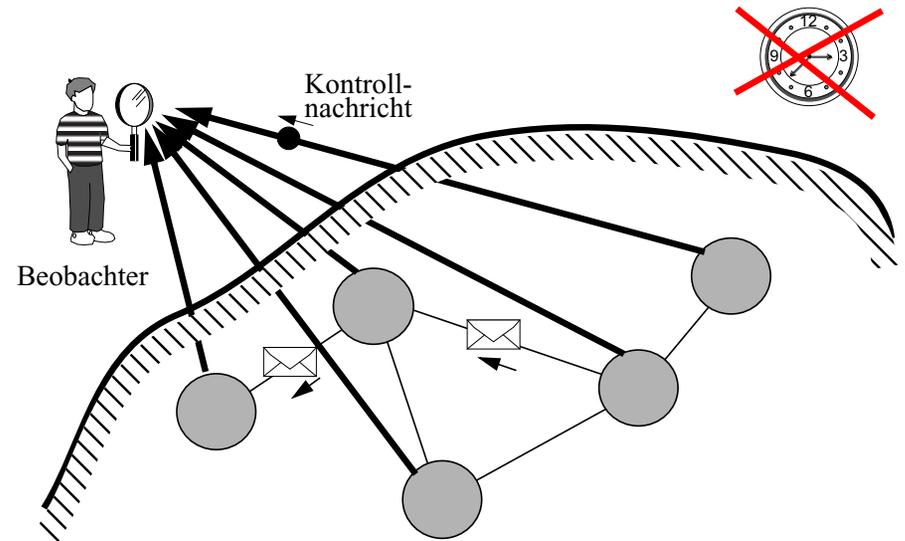
⇒ gute Werkzeuge (“Tools”) und Methoden
- z.B. Middleware als Software-Infrastruktur

⇒ 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)

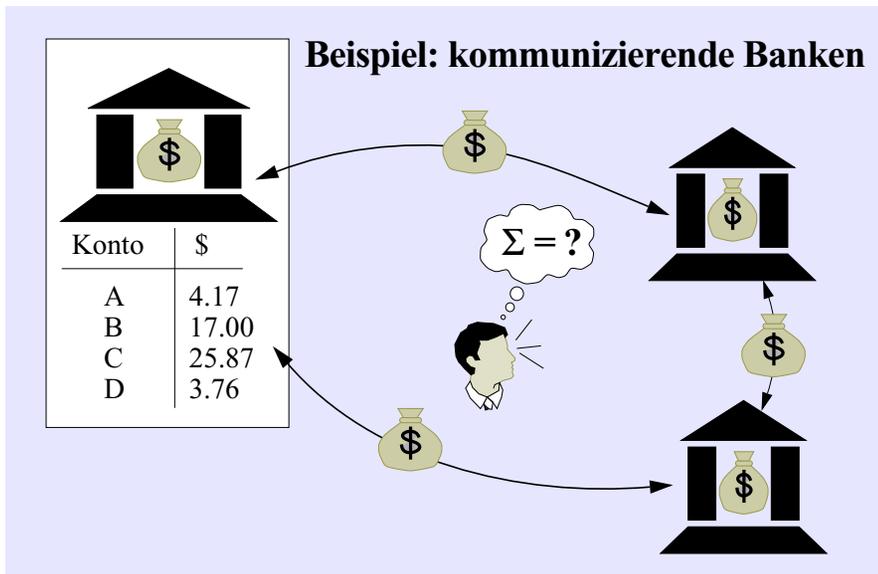
"Axiom": Mehrere Prozesse können "niemals" gleichzeitig beobachtet werden

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)



- 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...)

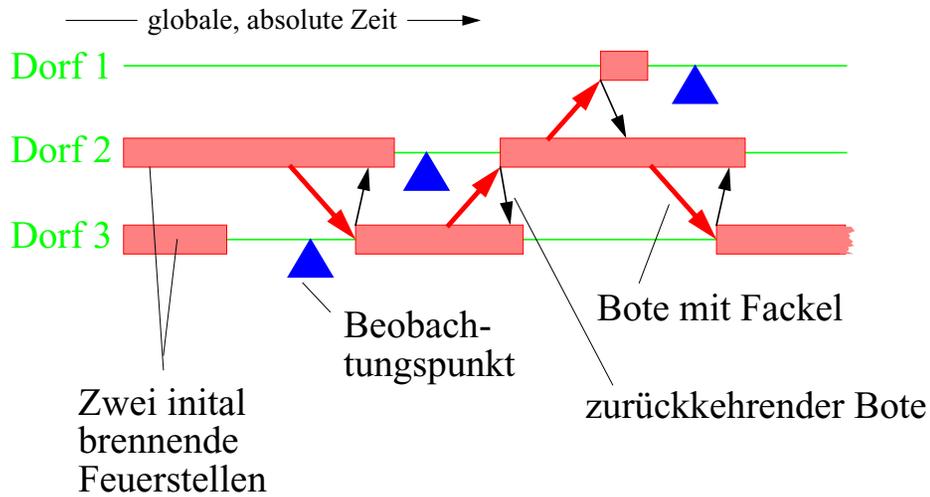


- Feststellen der Terminierung ist wichtig
 - kein warmes Essen bis zum nächsten Gewitter...



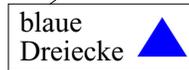
Falsche Beobachtungen

Beobachter kann **nicht** alles **gleichzeitig** betrachten!



Für alle Feuerstellen gilt (zu einem Zeitpunkt):

- kein Feuer brennt
- kein Bote ist unterwegs

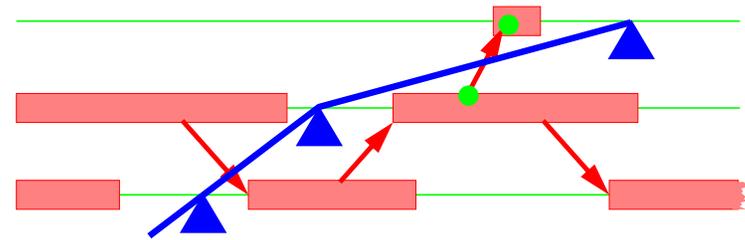


Aber: Es gibt keinen **einzigsten Zeitpunkt**, wo kein Feuer brennt!

⇒ Beobachtung liefert ein **falsches** ("schiefes") Bild!

⇒ **Terminierungserkennungs-Problem**

Was läuft schief?



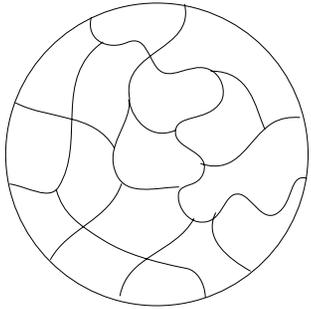
- Es scheint, als ob **alle Feuer aus sind** und **genausoviele Boten angekommen** wie aufgebrochen sind
 - dann wäre tatsächlich "alles aus" (bis zum nächsten Blitzschlag)

- Täuschungsgrund: Beobachtung ist **nicht kausaltreu**

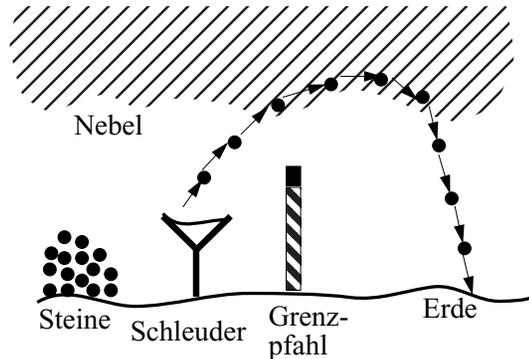


- **Wirkung** wird registriert, nicht aber die zugehörige **Ursache**
- **inkonsistente** Sicht durch schiefen Schnitt
- Würde man **alles gleichzeitig** beobachten, dann könnte man niemals eine Wirkung ohne ihre Ursache wahrnehmen!

Wie stellt man fest, dass der ewige Friede ausgebrochen ist?



Die Welt ist vollständig in einzelne Gebiete parzelliert



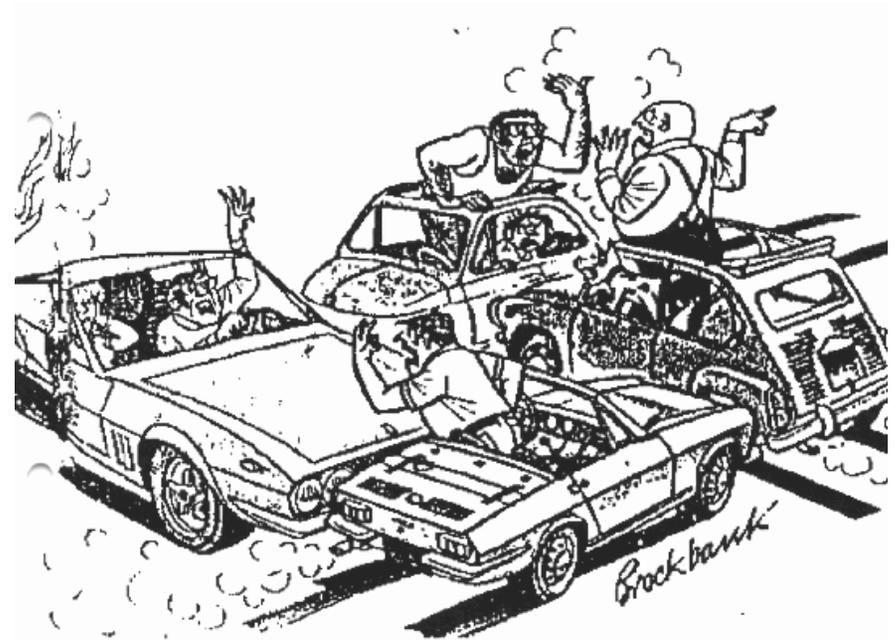
Regel: Nur wer von einem Stein getroffen wird, darf Steine wegschleudern

- Urknall: Ein einziger Meteorit traf die Erde...
- Steine fliegen beliebig hoch und brauchen beliebig lang...
- Niemand hat den Gesamtüberblick...

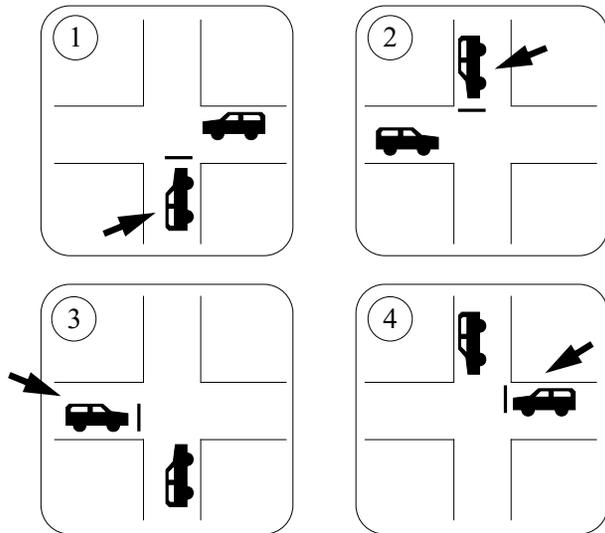
- Der Friede ist ewig ("stabil"; "monotone Eigenschaft")
- Wie stellt man sicher fest, dass er gilt (wenn er gilt)?

==> *Terminierungserkennungs-Problem*

Das Deadlock-Problem



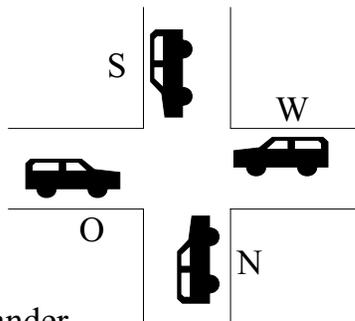
Phantom-Deadlocks



Vier Einzelbeobachtungen der Autos N, S, O, W

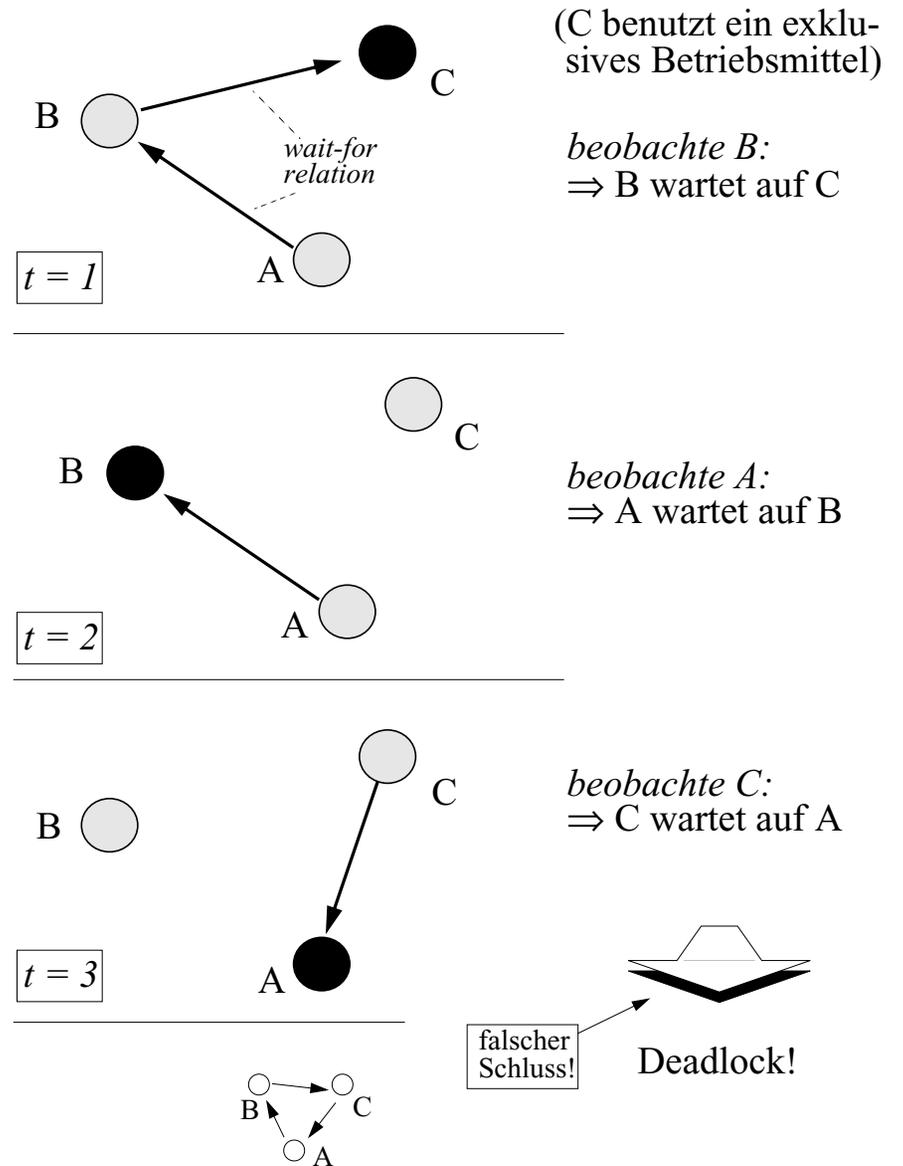
- 1) N wartet auf W
- 2) S wartet auf O
- 3) O wartet auf N
- 4) W wartet auf S

zu notwendigerweise
verschiedenen Zeitpunkten
liefert den *falschen* Eindruck,
als würden zu einem *einzigem*
Zeitpunkt alle zyklisch aufeinander
warten (→ Verklemmung)



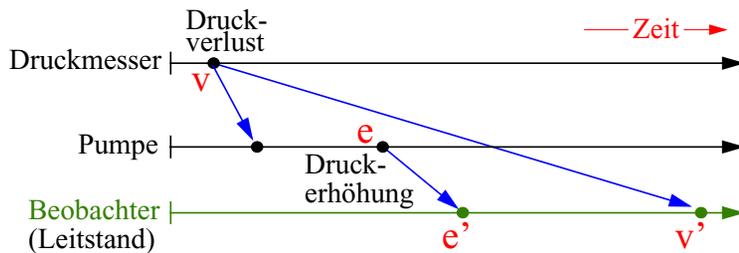
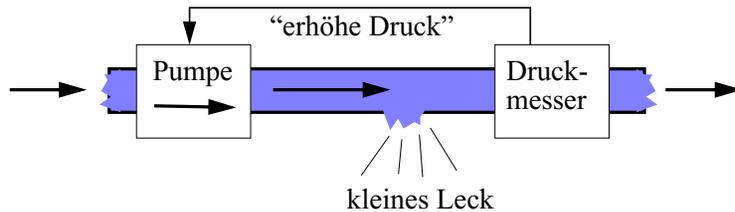
==> *verteiltes Deadlockproblem*

Phantom-Deadlocks



Kausal (in)konsistente Beobachtungen

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

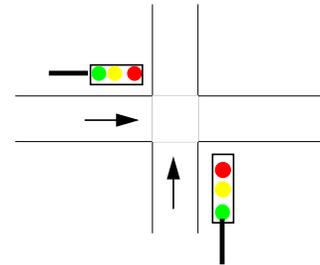


Falsche Schlussfolgerung des Beobachters:
 Es erhöhte sich der Druck (aufgrund einer unbegründeten Aktivität der Pumpe), es kam zu einem Leck, was durch den abfallenden Druck angezeigt wird.

==> **"Causal order-Problem"**

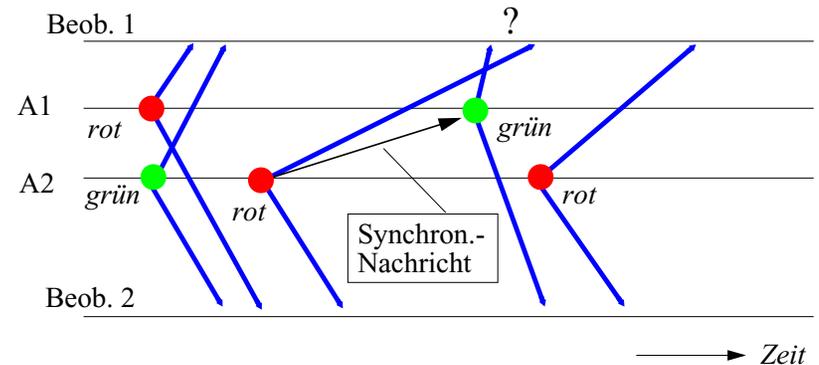
Beispiel: Verteilte Ampelsteuerung

(hier für 2 Ampeln)



Konsistenzproblem bei *mehreren Beobachtern*

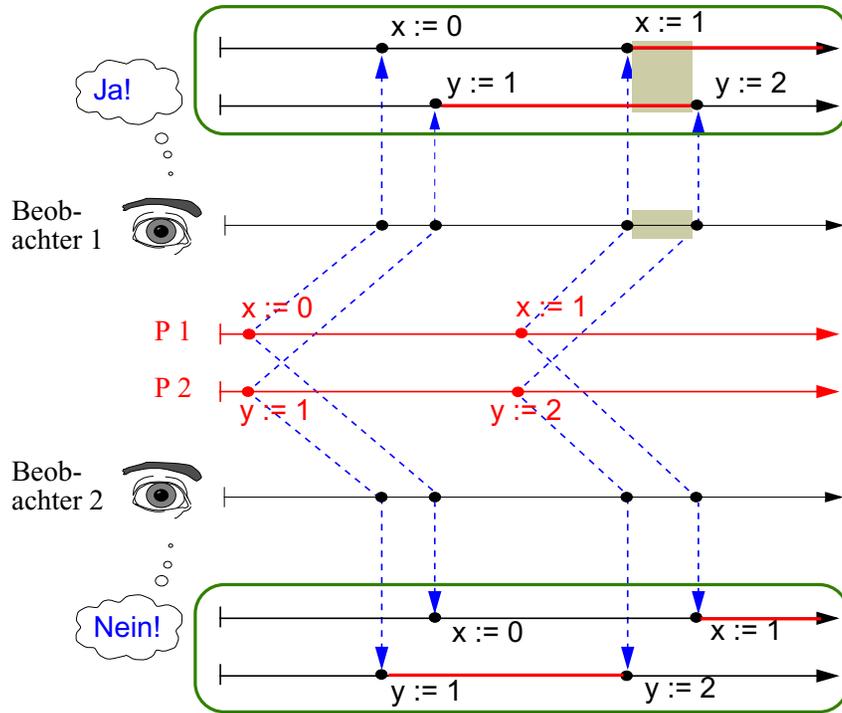
- Jede Ampel darf autonom auf **rot** schalten
- Eine Ampel darf nur dann auf **grün** schalten, wenn sie erfahren hat, dass die andere rot ist
- Umschalten der Ampel ist ein **Ereignis** ● (*atomar*: zeitlos, nicht unterbrechbare Aktion)



- **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 $x = y$?



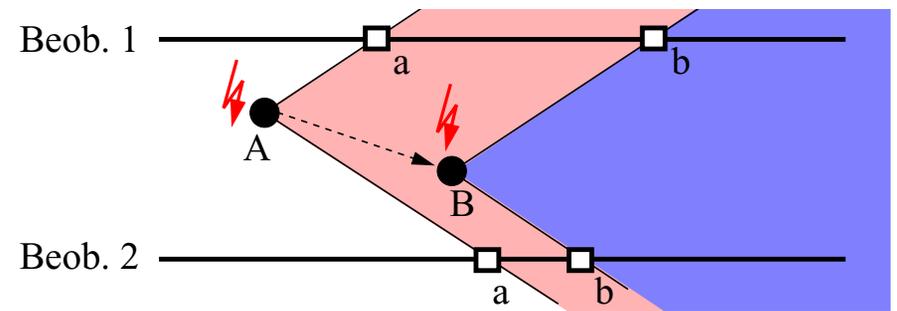
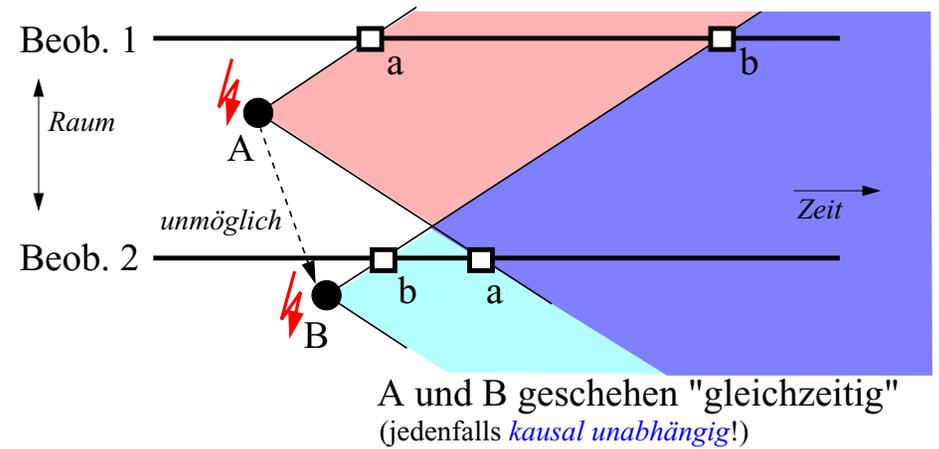
- Beide Beobachtungen sind gleich "richtig"
- Die Beobachter stimmen bzgl. $x = y$ nicht überein!

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

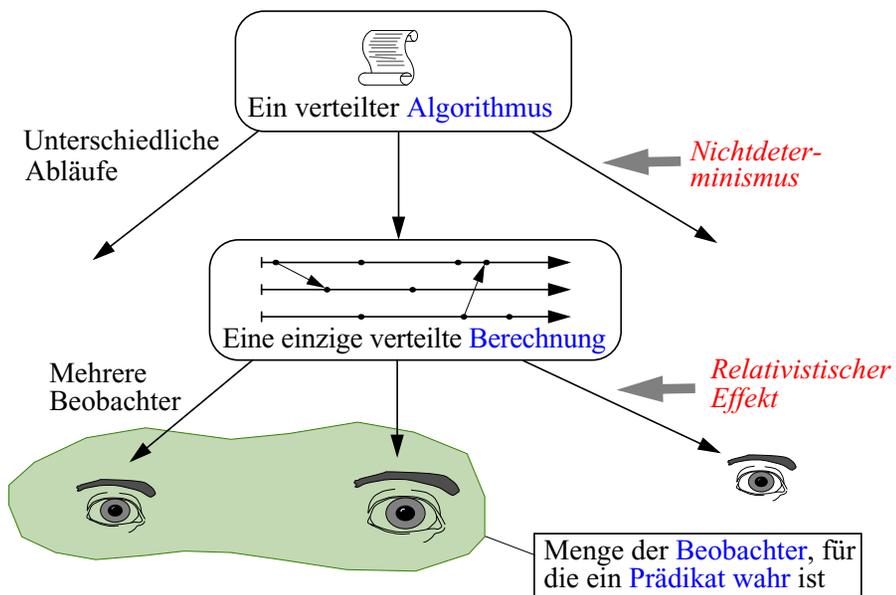


beobachterinvariant
 \Rightarrow objektive Tatsache

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

Die "Beobachtungsvielfalt"

- Verschiedene Beobachter sehen *verschiedene Wirklichkeiten*



Konsequenz:

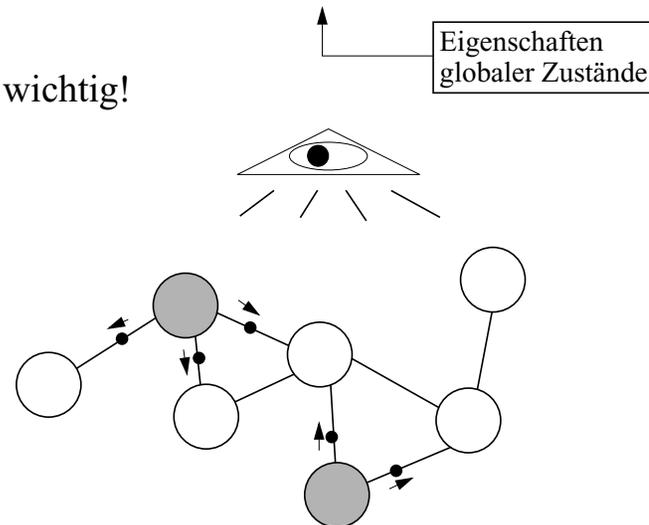
Es ist naiv (d.h. falsch!), einen **verteilten Debugger** zu entwickeln, mit dem man solche (im sequentiellen Fall "richtigen") Fragen beantworten kann!

Ursache:

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

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

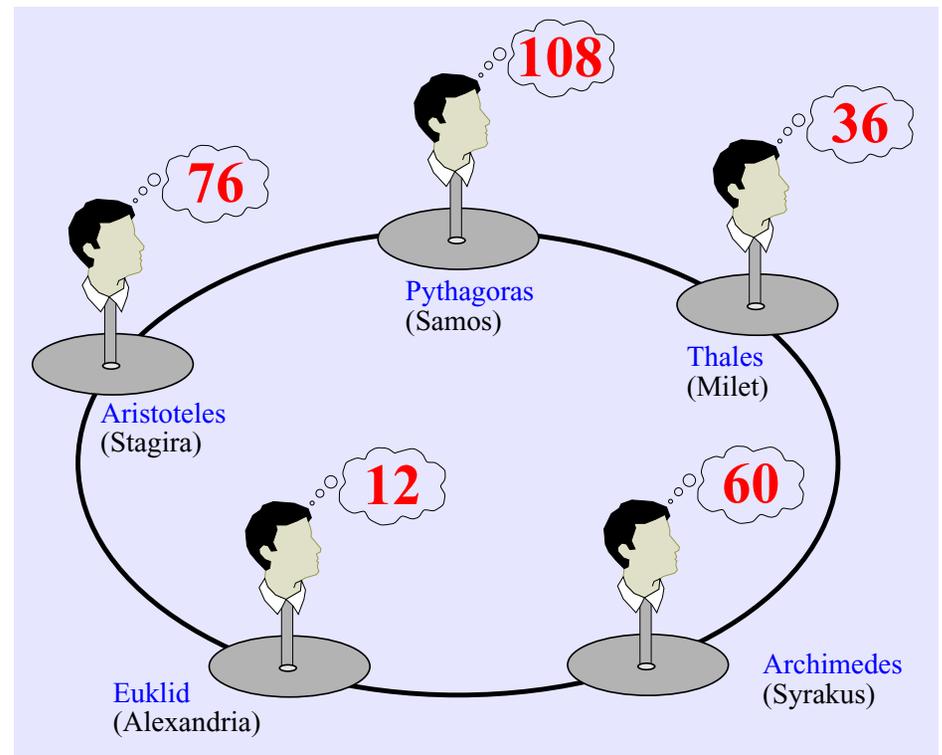
- Welche? → fundamentale Aspekte analysieren
- Wie? → *Algorithmen*

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 $\{a_1, \dots, a_n\}$ 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 $q \cdot t = p$
 - 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 $\text{ggT}(x,0)$?)
 - Kanonische Erweiterung auf $n \geq 1$ Argumente
 - Bsp: $\text{ggT}(108, 36, 60, 12, 76) = 4$
-

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

Satz von Euklid

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

- Offenbar ist $\text{ggT}(x,x) = x$ für alle x
- Man setzt nun noch $\text{ggT}(x,0) = x$ für alle x
- Obiger Satz legt eine rekursive Realisierung nahe

$$\text{ggT}(x, y) \rightarrow \text{ggT}(y, \text{mod}(x, y))$$

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

\Rightarrow *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

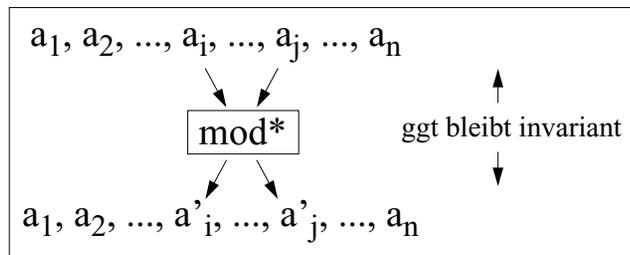
- Invariante (für $x \geq y > 0$):

$$\text{ggT}(x,y) = \text{ggT}(y, \text{mod}^*(x,y))$$

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

- Idee: Ersetze x durch $\text{mod}^*(x,y)$ ← Ist i.a. 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 $\text{mod}^*(a_i, a_j)$

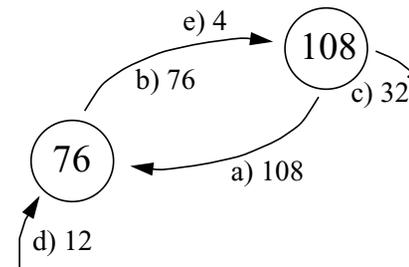
⇒ konvergiert gegen den ggT

- 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 < Mi then
    Mi := mod(Mi-1,y)+1;
    send <Mi> to all neighbours;
fi
```

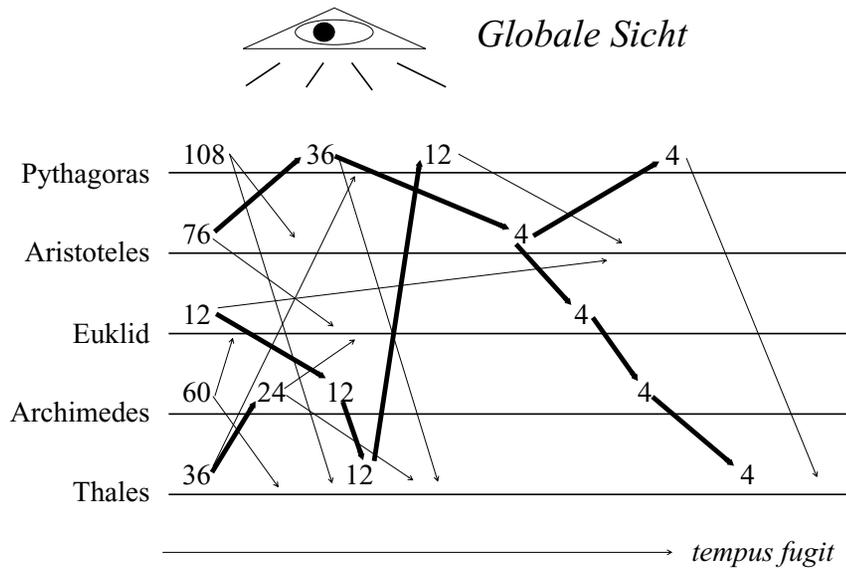
Jeder Prozess P_i hat seine eigene Variable M_i .

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 (?):

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

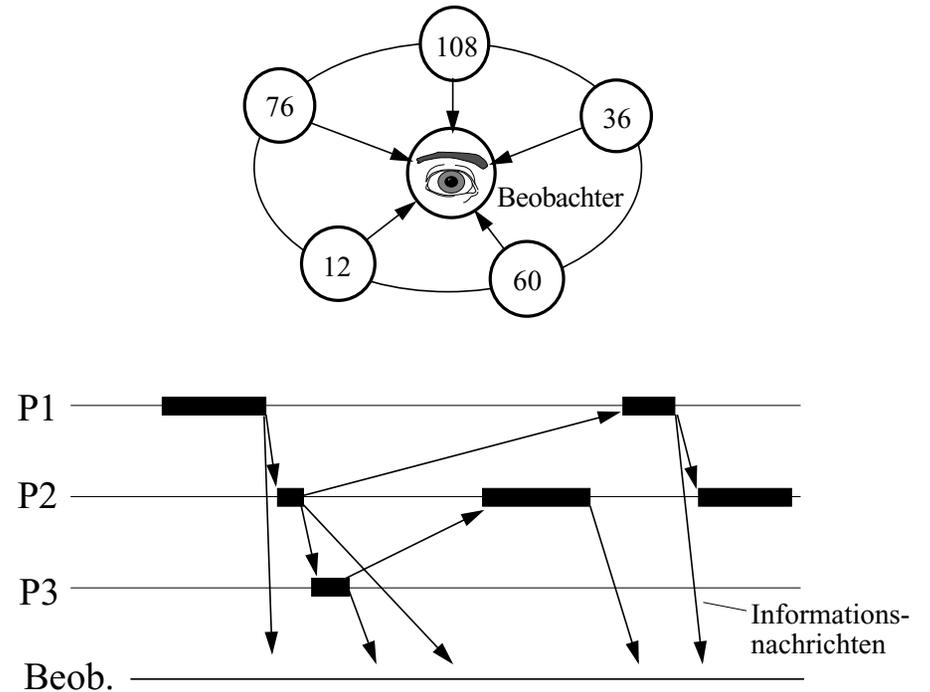
Das ist unrealistisch!

Beweisbare math. Eigenschaft

Diese "stabile" Stagnationseigenschaft ist problemunabhängig!

- Was ist adäquat?
- Wie feststellen?

Globaler ggT-Beobachter?



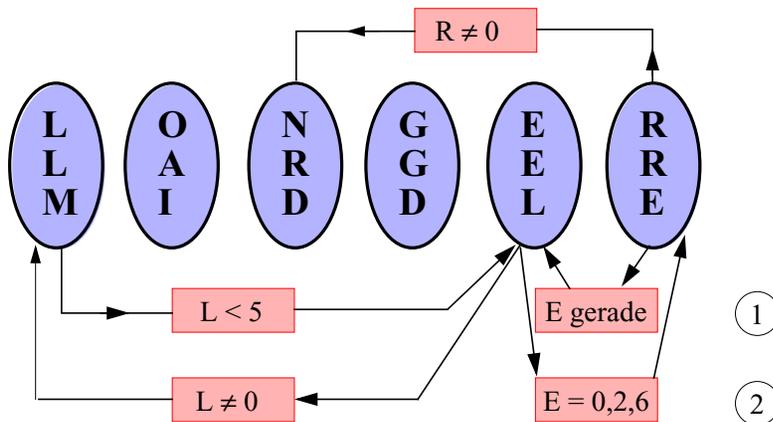
Bekommt der Beobachter ein "richtiges Bild" des Geschehens?

- 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	→	Pro Spalte ein Prozess
+LARGER	+283563		
MIDDLE	491126		

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

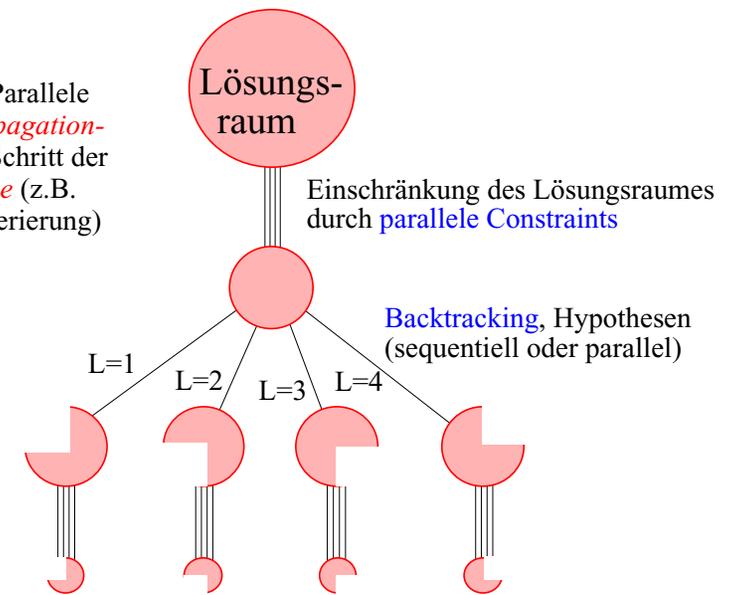


Endergebnis:
 $1 \leq L \leq 4$
 $M \geq 2$
 $R = 1,3,5,6,8$
 $E = 0,2,6$
 sonst keine Einschränkung

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

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"?

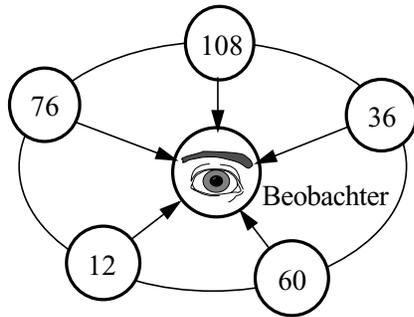
Bemerkungen:

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

Offensichtlich *gleiches Schema* wie ggT-Berechnung!

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

- a) Man zeichne Raum-Zeit-Diagramme für verschiedene Abläufe des verteilten ggT-Algorithmus
- b) Wie kann man beweisen, dass für *jeden* denkbaren Ablauf das Endergebnis stets der ggT ist?



- c) 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 :

```

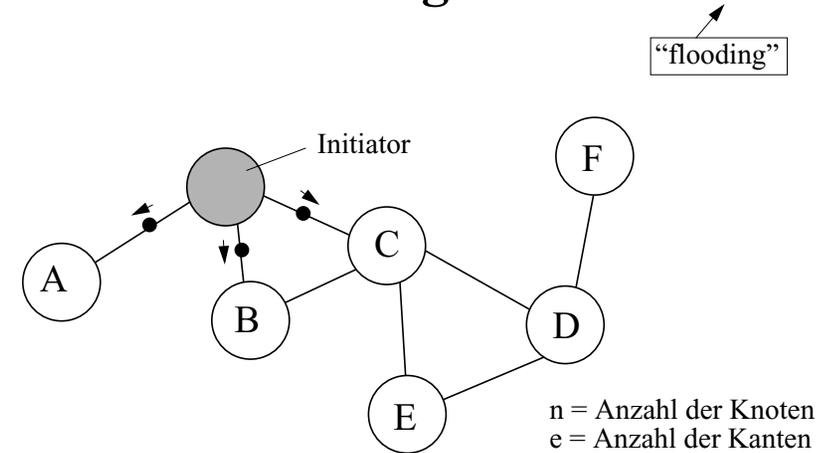
{Eine Nachricht  $\langle y \rangle$  ist eingetroffen}
if  $y < M_i$  then
   $M_i := \text{mod}(M_i - 1, y) + 1$ ;
  send  $\langle M_i \rangle$  to all neighbours;
fi
    
```

...Übungen (1)

- d) Man vergleiche die verteilte Berechnung des ggT-Algorithmus für zwei Zahlen mit dem üblichen sequentiellen ggT-Algorithmus für zwei Zahlen
- e) Genügt es auch, nur in Uhrzeigerrichtung eine Nachricht zu senden (anstatt an beide Nachbarn)?
- f) Kann statt des Ringes eine andere Topologie verwendet werden? Welche?
- g) Formalisieren Sie für Zeitdiagramme den Begriff (potentiell, indirekt) "kausal abhängig" als Halbordnung über "Ereignissen"
- h) Wie kann man erreichen, dass ein ggT-Beobachter (der über jede Wertänderung eines Prozesses informiert wird) eine "kausaltreue" Beobachtung macht?
- i) 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?
- j) Wie kann der Beobachter die Terminierung erkennen?

Flooding, Echo-Algorithmus, Broadcast

Informationsverteilung durch “Fluten”



- 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 ($\rightarrow +1$)

$$\Rightarrow \text{Also: } 2e - n + 1$$

- Frage: Wie *Terminierung* feststellen?

d.h.: wie erfährt der Sender (= Initiator), wann alle erreicht wurden?
(das ist für “sicheren” oder “synchronen” Broadcast notwendig)

Flooding-Algorithmus - eine etwas formale Spezifikation

- Zwei *atomare* Aktionen für jeden Prozess:

- wechselseitig ausgeschlossen
- "schlagartig"?
- ununterbrechbar?

Assertion (muss wahr sein,
damit Aktion ausgeführt wird)

R: {Eine Nachricht $\langle \text{info} \rangle$ kommt an}
if not informed then
 send $\langle \text{info} \rangle$ **to** all other neighbors;
 informed := true;
fi

Natürlich auch
"merken" der
per Nachricht
erhaltenen
Information
 $\langle \text{info} \rangle$

I: {not informed}
 send $\langle \text{info} \rangle$ **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 (ggf. 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 ggf. 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)

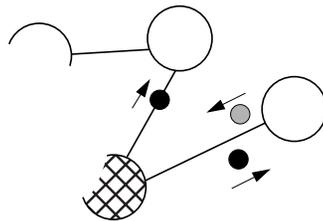
- direktes Bestätigen einer Nachricht funktioniert nicht
- indirekte Bestätigungsnachrichten: ein Knoten sendet erst dann ein ack, wenn er selbst für alle seine Nachrichten acks erhalten hat
- klappt diese Idee? auch wenn der Graph Zyklen enthält? wieso?

Flooding mit Quittungsmeldungen

(Originalversion des *Echo-Algorithmus* von Chang '82)

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

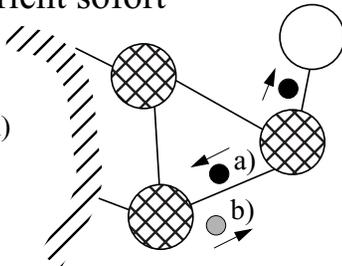
- Ein Knoten mit Grad 1 ("Blatt") sendet sofort eine Quittung zurück



Nachrichte 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

(PIF-Variante von A. Segal, 1983)

Propagation of Information with Feedback

- Ähnliches Verfahren 1980 von Dijkstra/Scholten: "Diffusing Computations"

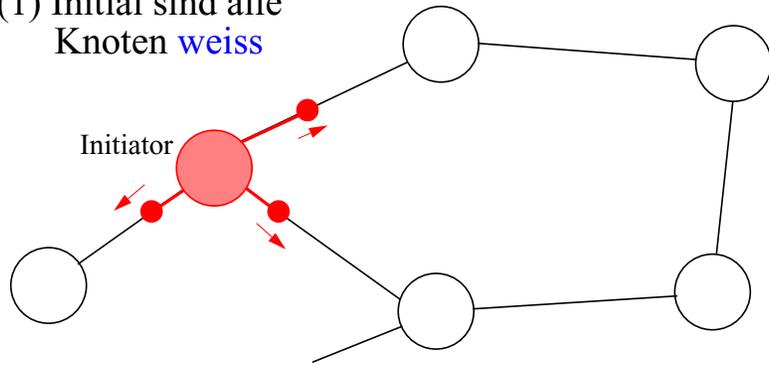
Ausgehend von einem einzigen Initiator:

- 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 *spannenden Baumes* ("Echo Kanten": jeder Knoten sendet genau ein Echo)

Paralleler *Wellenalgorithmus*:

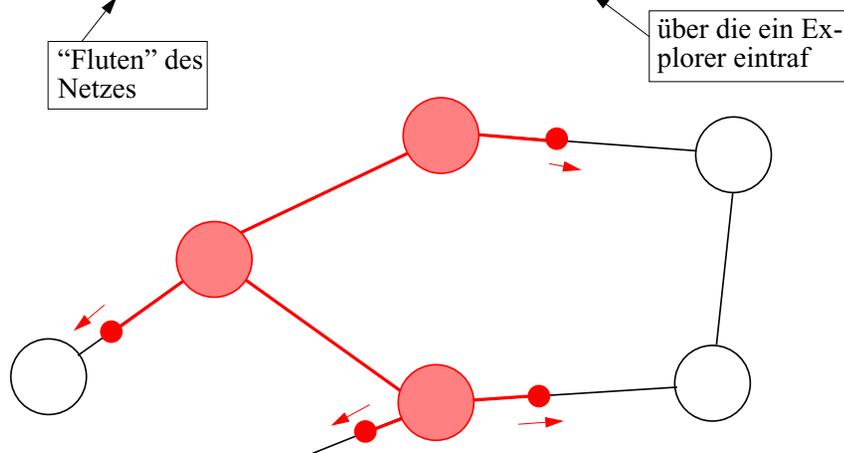
- virtueller broadcast
- *Basisalgorithmus* für andere Verfahren ("underlying algorithm"; "superposition")

(1) Initial sind alle Knoten **weiss**

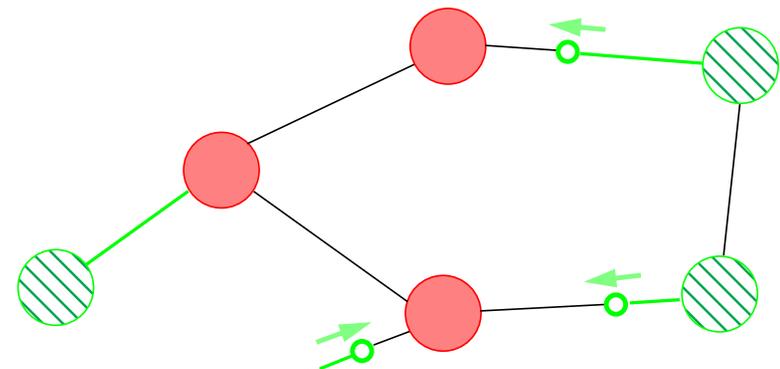
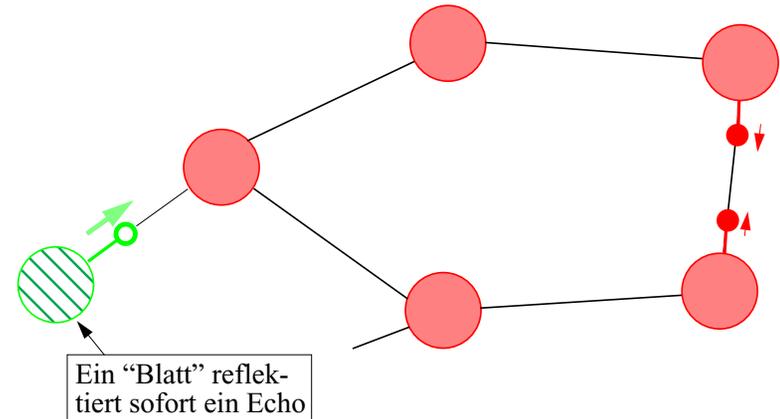


- Der (eindeutige) **Initiator** wird **rot** und sendet (rote) **Explorer** über alle seine Kanten

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



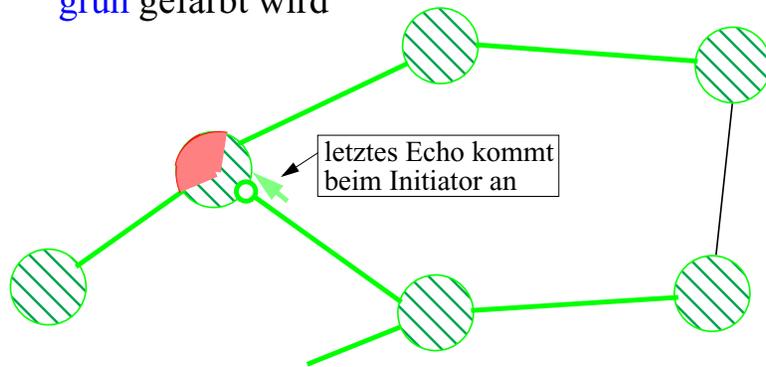
(3) Ein (roter) Knoten, der über alle seine Kanten einen Explorer *oder* ein Echo erhalten hat, wird **grün** und sendet ein (grünes) **Echo** über seine "erste" Kante



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

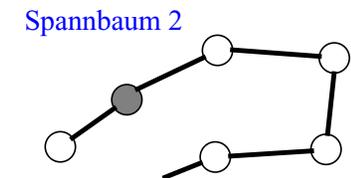
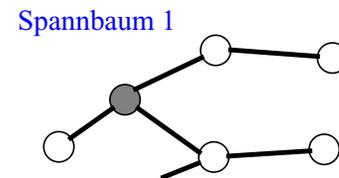
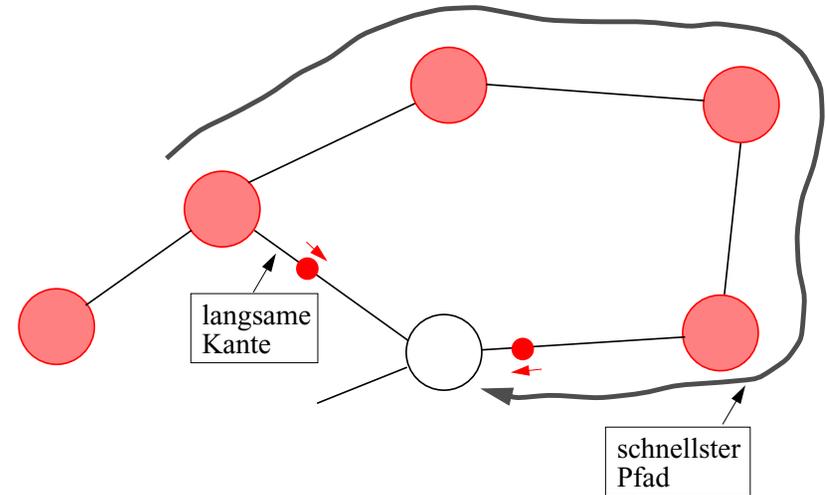
Auf einer Kante, wo sich zwei Explorer begegnen, wird der **Zyklus aufgebrochen**

(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 Spannbaume* entstehen!



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

Echo-Algorithmus...

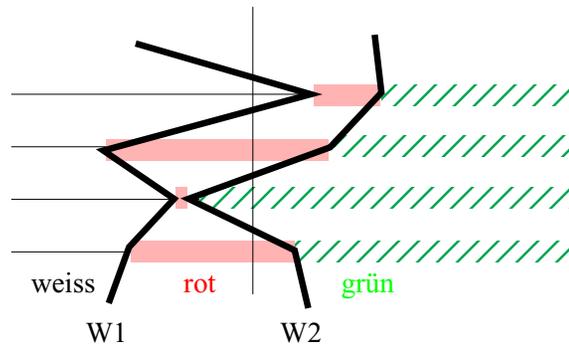
- Jeder Knoten wird *erst rot* und *dann grün*
- Rote Phase ist bei "Blättern" allerdings recht kurz
- Ein *grüner Knoten* hat *keine weisse Nachbarn*

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

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

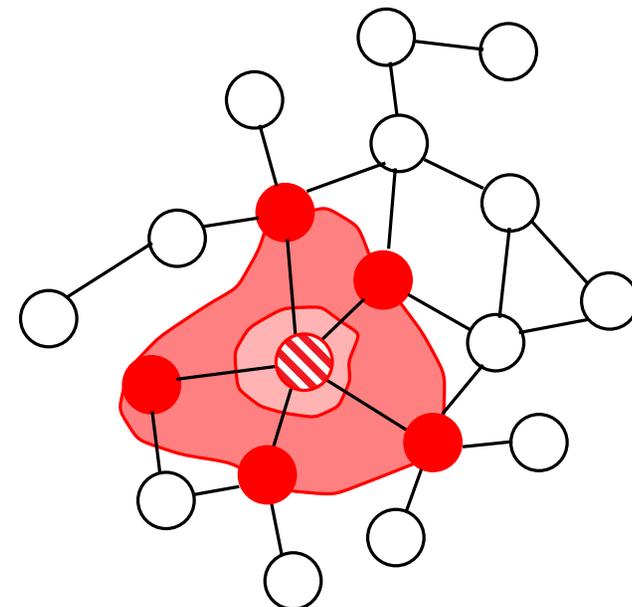
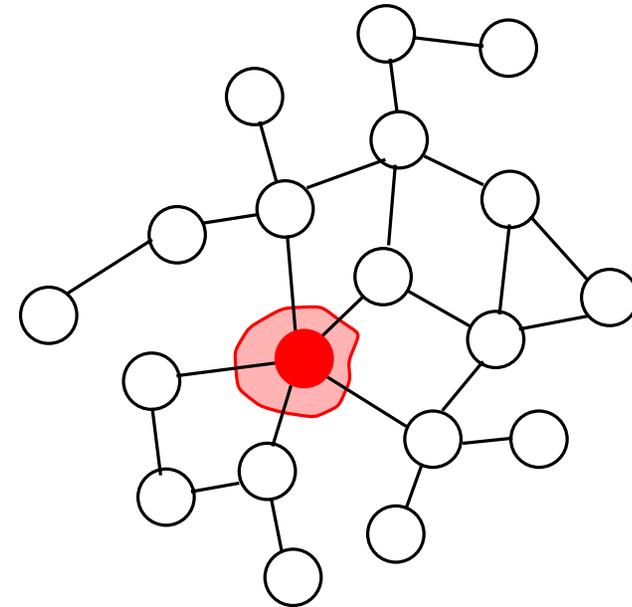
⇒ *Weisse und grüne Phase* sind in "gewisser Weise" *disjunkt*

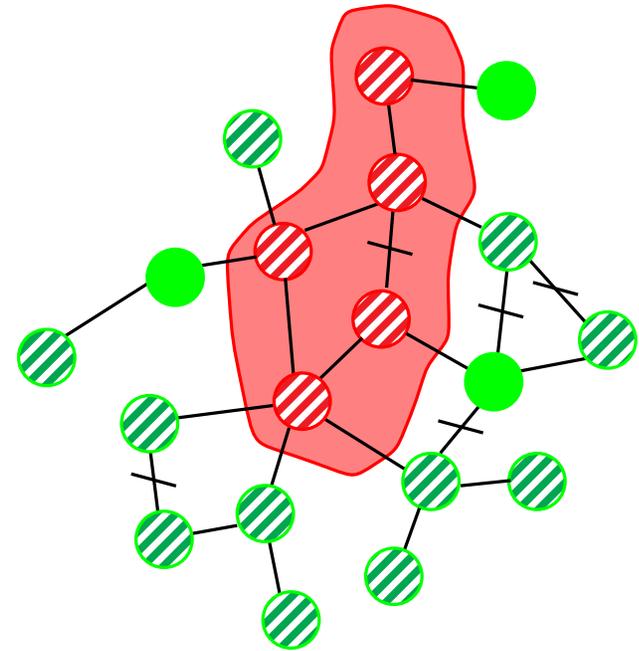
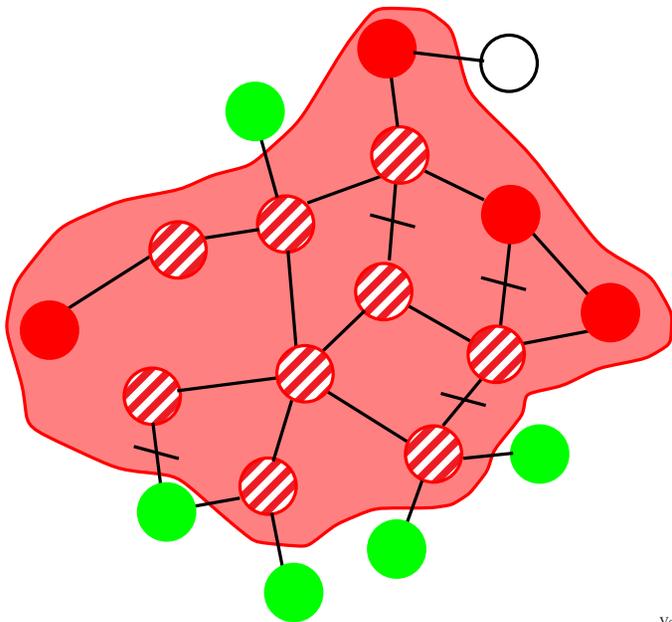
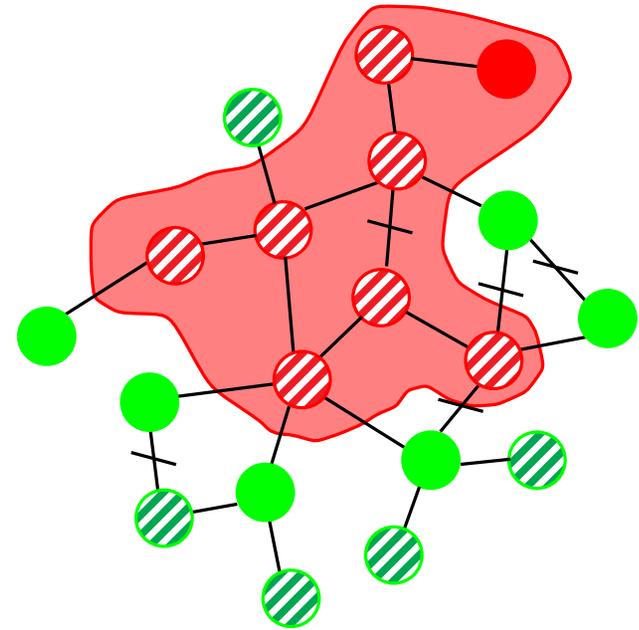
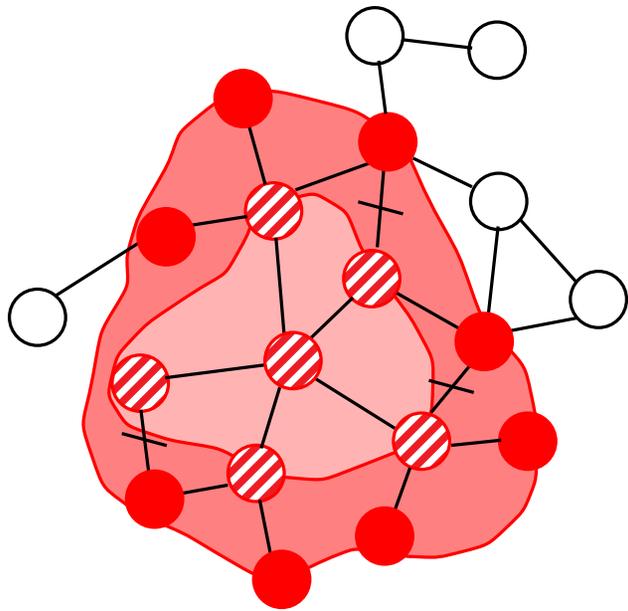
- obwohl es globale Zeitpunkte geben kann, wo ein Knoten bereits grün ist, während ein anderer (nicht direkt benachbarter!) noch weiss ist!

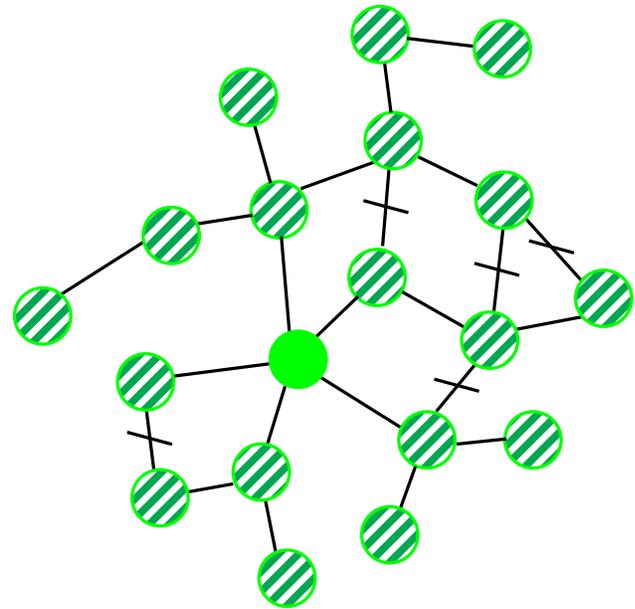
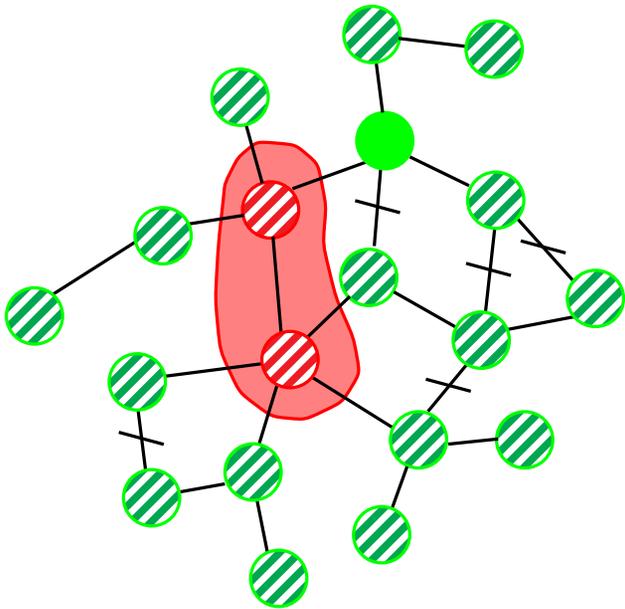
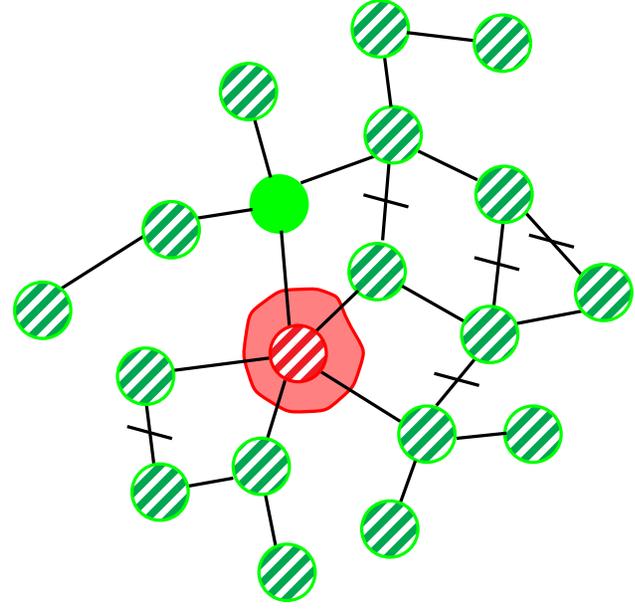
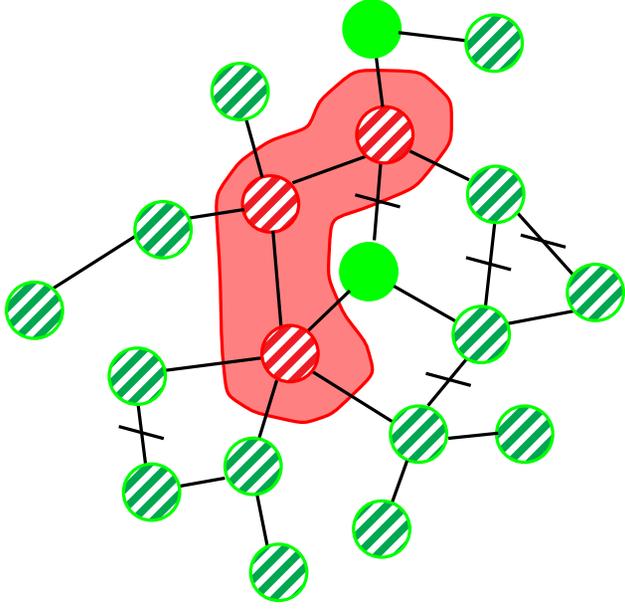


- "Rot werden" und "grün werden" definieren zwei *Wellen*

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





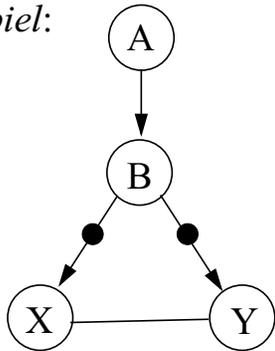


Verbesserung des Echo-Algorithmus?

[Helary et. al.]

- *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):

```

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

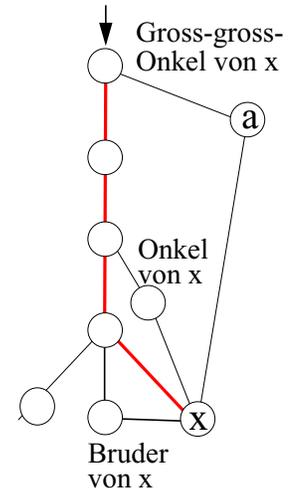
Menge von "Tabu-Knoten"

Statt neighbors ohne Vorgänger im Original

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

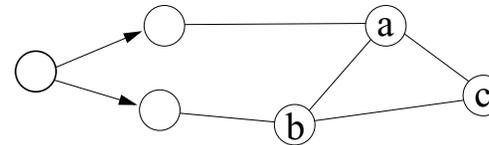
Was wird gespart?

- Keine Nachricht an einen Vorgänger oder einen direkten Nachbarn eines Vorgängers (z.B. "Brüder" und "Onkel")



- 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 in diesem 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 "Vermaassungsgrad" 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

[F. Mattern - Informatik-Spektrum 8:6, pp. 342-343, 1985]

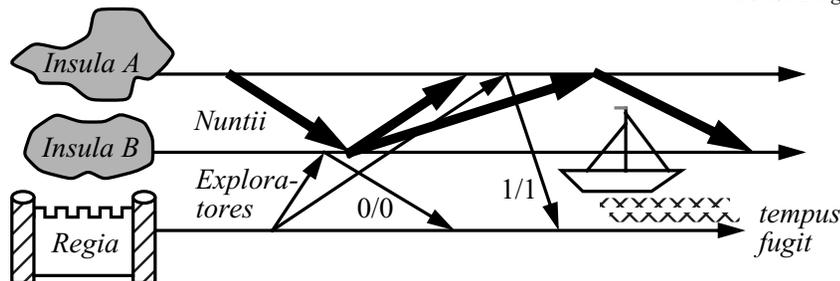
Als einst der König von Polymikronien merkte, dass die Zeit gekommen war, sein aus unzähligen Inseln bestehendes Reich gerecht unter seinen Enkelkindern aufzuteilen, sandte er Botschaften an die weit über das Land verteilt lebenden Weisen aus, auf dass diese ihm einen klugen Vorschlag unterbreiten mögen.

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"...

- Man beweise die Korrektheit des im Märchen beschriebenen "Doppelzählverfahrens" zur Feststellung der verteilten Terminierung.
- 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 formilere 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 alsdann 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. Desweiterem gelang es ihm offenbar, die Zahl der notwendigen Kundschafterfahrten durch eine einfache Funktion der Zahl der Inseln und Botschaften zu beschränken. Leider reichte wiederum der Rand zur Darstellung der Methode nicht aus. Wer hilft mit bei der Rekonstruktion und Verifikation der Verfahren ?

Zeitkomplexität

Beachte: Algorithmen i.a. 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.

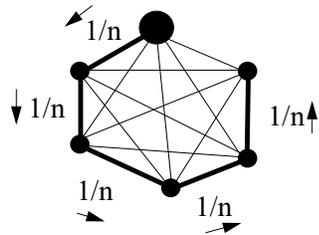
- Grund: Einheitszeitkplx erlaubt nicht alle Berechnungen!
- Frage: Gilt Umkehrung?

Bsp. Echo-Algorithmus auf vollständigem Graph

(1) Einheitszeitkomplexität = 3

(2) Variable Zeitkomplexität ≥ n

- Phase 1: Alle werden rot
- Phase 2: Alle bis auf Initiator werden grün
- Phase 3: Initiator wird grün



- 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

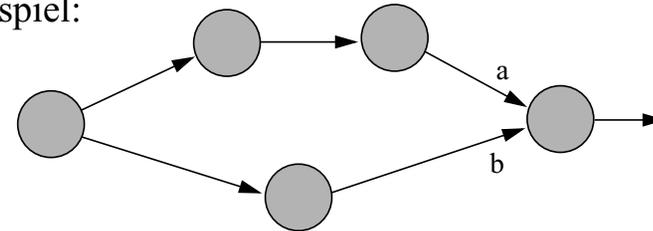
Zeitkomplexität: Welche Definition?

- *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...

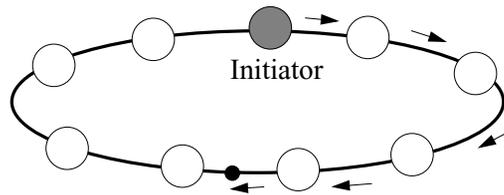
Broadcast auf speziellen Topologien

- Echo-Algorithmus realisiert einen Broadcast
 - Verteilen von Information ausgehend von einem Initiator
 - für beliebige (zusammenhängende) Topologien
 - liefert sogar "Vollzugsmeldung" durch Echo-Nachrichten

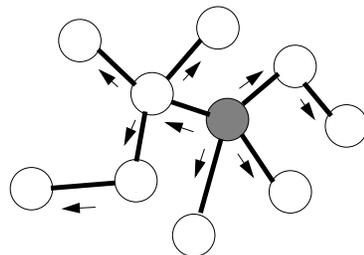
auf bel. zusammenh. Topologie

- 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
- ggf. kann einer anderen Topologie ein Ring überlagert werden



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



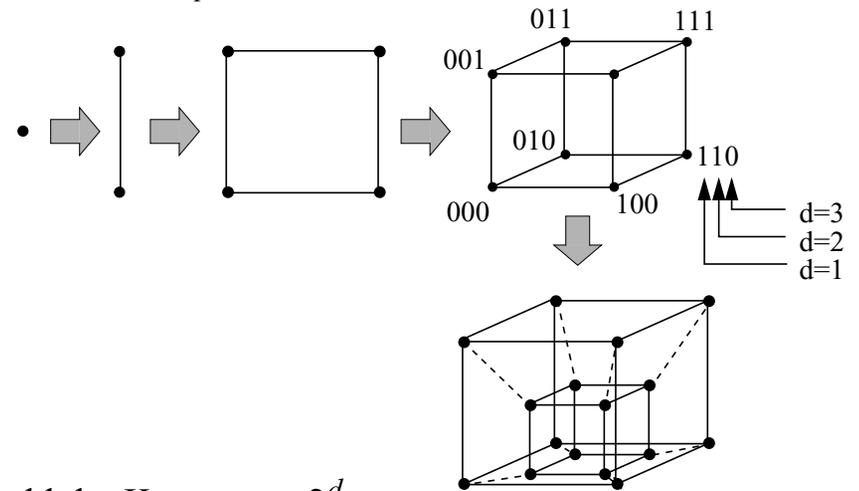
vorausgesetzt wird jeweils, dass der Algorithmus "weiss", dass eine spezifische Topologie vorliegt!

- 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 $O(n \log n)$)

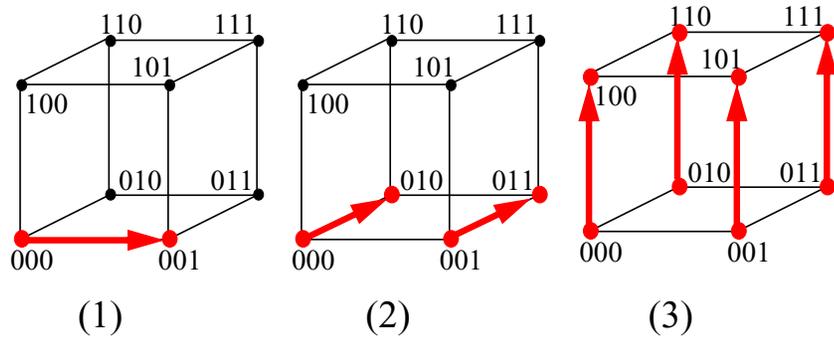
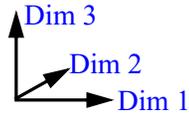
- viele Wegalternativen (Fehlertoleranz, Parallelität!)
- maximale Weglänge: $d = \log n$
- mittlere Weglänge: $d/2$ (Beweis als Denkübung!)

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 Acknowledgements oder Endeerkennung)



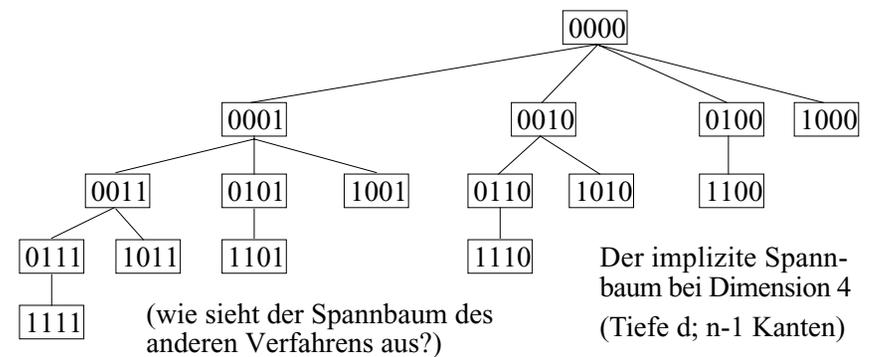
- Analog zum **rekursiven Aufbau** des Hypercube:
 - zunächst in **Dimension 1** senden: Teil-Hypercube der Dimension 1 ist damit informiert
 - dann senden alle Knoten der Dimension 1 in **Dimension 2**
 - dann **Dimension 3** etc.
- Nach **d "Takten"** sind alle Knoten informiert
 - **Zeitkomplexität** ist daher **d** (unter welchem Zeitmass?)
 - **Nachrichtenkomplexität**: $1 + 2 + 4 + \dots + 2^{d-1} = 2^d - 1$ (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:
 - am besten gleichzeitig, wenn dies technisch geht!
 - 0...01, 0...010, 0...100, ..., 10...0
 - in "kanonischer" Numerierung
 - linkeste 1
 - beliebiges Restmuster
- Ein Knoten mit der Nummer 0...01x...y...z leitet die Information an alle seine "höheren" Nachbarn weiter:
 - 0...0011x...y...z
 - 0...0101x...y...z
 - 0...1001x...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



- 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:

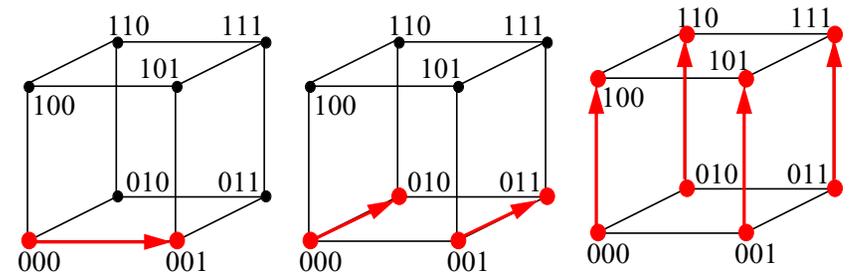
- Ein Baum verwendet im Hypercube relativ wenig Kanten → schlechte Ausnutzung potentielle Parallelität
- Es gibt *mehrere* Spannbäume in Hypercubes → diese nutzen?

- 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 $d, 1, 2, \dots, d-1$

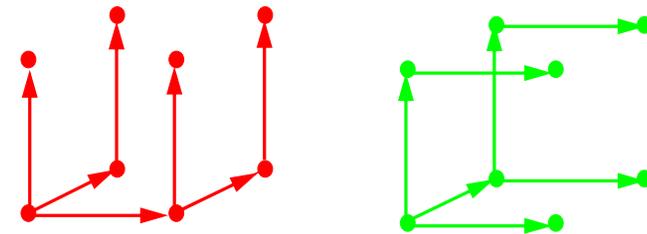
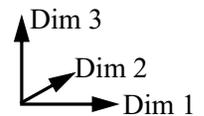
Denkübungen:

- 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

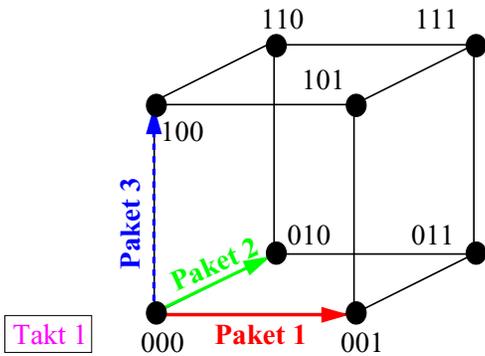


Die drei "Takte" der Welle von Paket 1

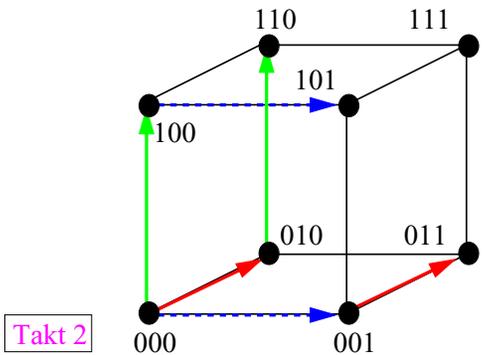


Die Spannbäume bzgl. Paket 1 und Paket 2

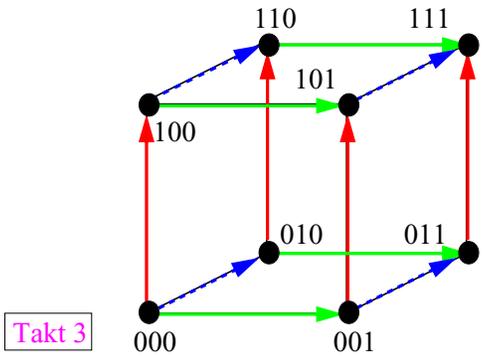
Parallelausführung der drei Wellen



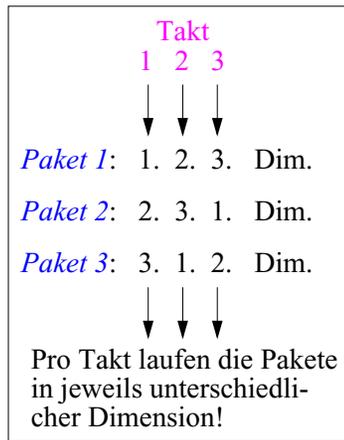
Takt 1



Takt 2



Takt 3



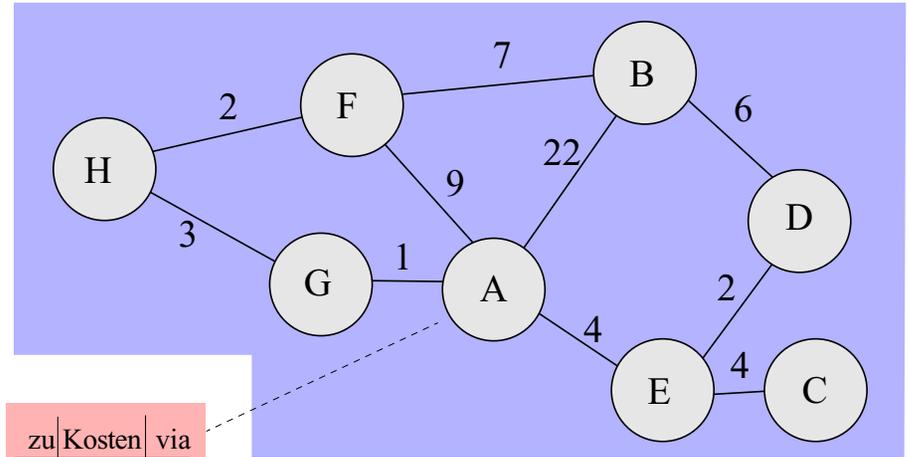
Es können also tatsächlich die **drei Wellen parallel** ausgeführt werden, ohne dass diese sich gegenseitig stören!

→ Dies ist das (im Prinzip) **schnellere Verfahren!**

Beachte: Ein globaler 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	∞	?

Anfangstabelle für Knoten A

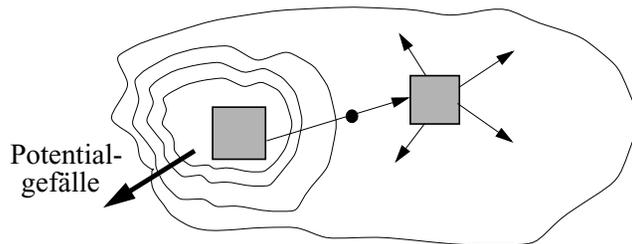
- Jeder kennt **anfangs** die **Kosten** zu seinen **Nachbarn**
- "Spontanstart": **Sende eigene Tabelle** an Nachbarn
- Bei **Empfang** einer Tabelle über Verbindung mit Kosten g:
Für alle Zeilen i der Tabelle:
Falls $Nachricht.Kosten[i]+g < Knoten.Kosten[i]$:
ersetze Zeile (Kosten := Kosten+g; via := 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
 - "Relaxationsprinzip" (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



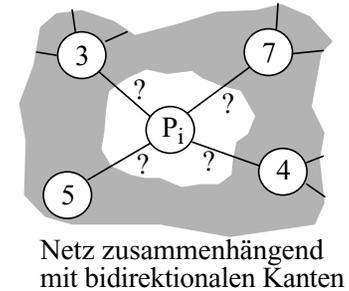
- Nachrichtengesteuert (aber "Spontanstart")
- Alle Prozesse arbeiten gleich, alle sind beteiligt
- Nichtdeterministischer Ablauf, determin. Ergebnis
- Beliebige stark zusammenhängende Topologie
- Assoziative Operatoren (min, max, \cap , \cup , +, and, or, ...)
- Stagnation bei globalem Gleichgewicht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

Beispiele ("Instanzen der Algorithmeklasse"):

- ggT
 - Zahlenrätsel
 - Verteilen von Information ("Wissensausgleich")
 - Routingmatrizen (inkl. Spannbaum)
 - Maximale Identität ("election")
 - Lastausgleich (Approx. eines dyn. Optimums)
 - Relaxationsverfahren (Lösen von DGL)
- } (noch)
} nicht be-
} handelt

Nachbarschaftswissen

- Zweck: Lerne Identitäten Deiner Nachbarn kennen
(Idee: "Ich heiße i, wie heißt Du?")
- Hier nachrichtengesteuert für P_i :
(vgl. verteiltes Approximationsschema)



```

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

↓ mehrere?
Einer muss "spontan" mit
der Vorstellung beginnen

- global terminiert, wenn
 - vorgestellt = 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?

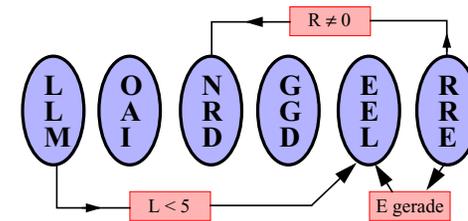
Verwendung als:

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

Verteilte Terminierung

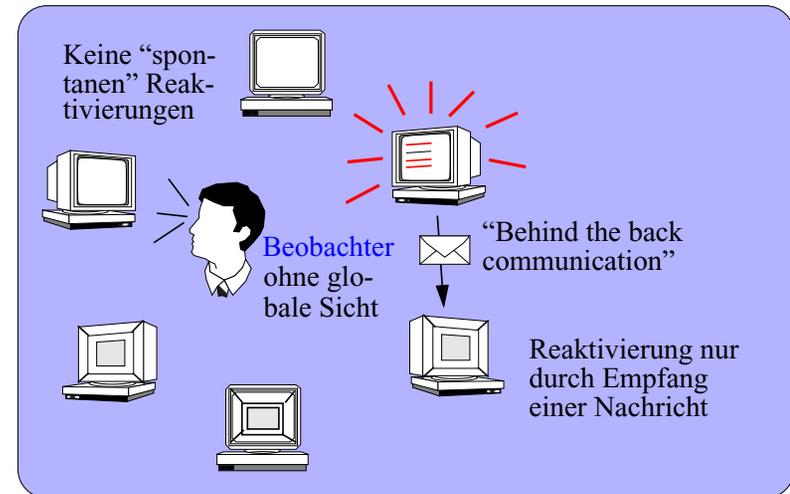
Das Problem der Terminierung

- Bsp: Zahlenrätsel (oder ggT) auf einem PC-Cluster



- pro Spalte (bzw. "Philosoph") jeweils ein Display
- dort jeweiligen Zustand und neue Wertemengen anzeigen

aktiv oder passiv



Erkennung der verteilten Terminierung?

alle passiv und keine Nachricht unterwegs

Die Aktionen der Basisberechnung im nachrichtengesteuerten Modell

Eine typische Behörde

Prozess p:

"guard": Prädikat über dem lokalen Zustand

$S_p: \{Zustand = aktiv\}$
send message $\langle M \rangle$ to ...

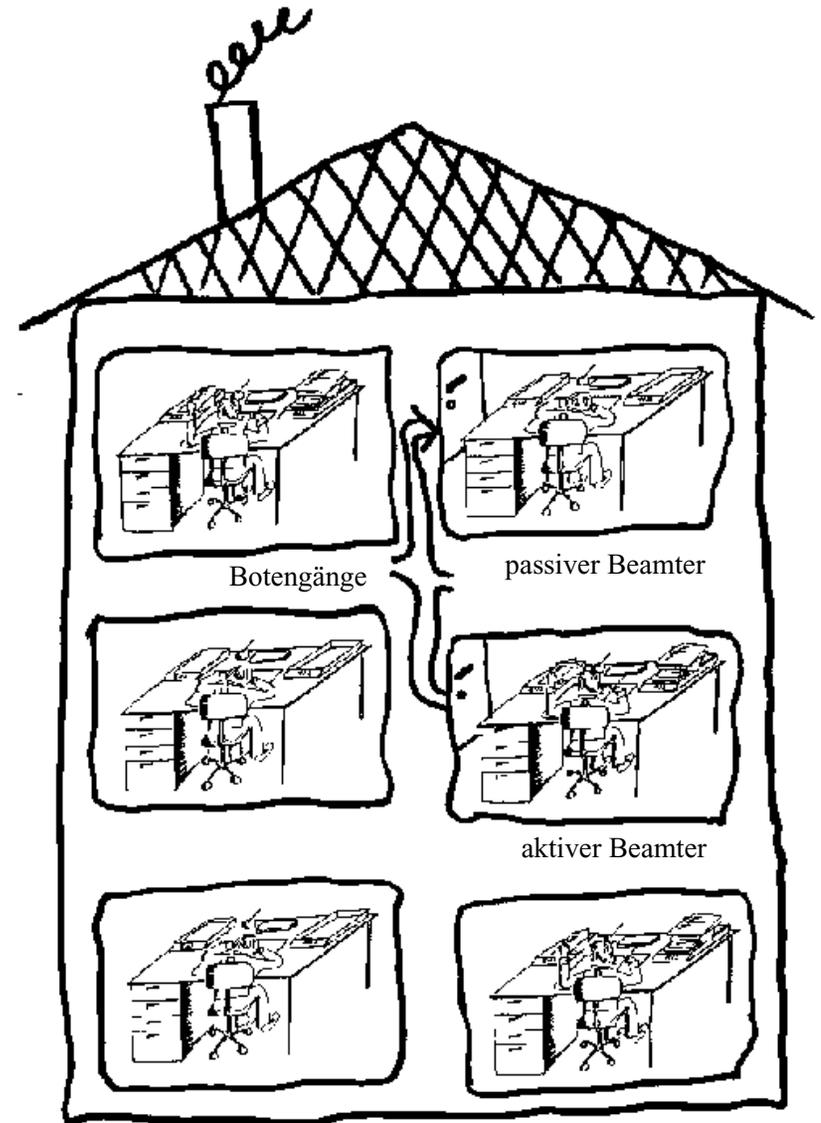
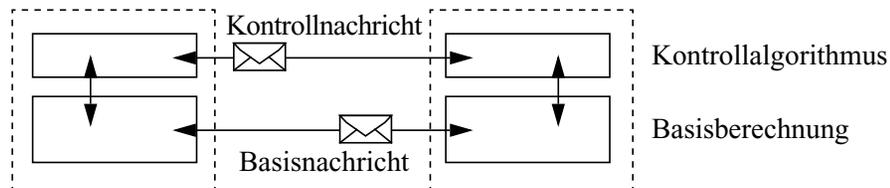
$R_p: \{Eine\ Nachricht\ ist\ angekommen\}$
receive $\langle M \rangle$; Zustand := *aktiv*

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

Typischerweise werden die Aktionen als "atomar" betrachtet

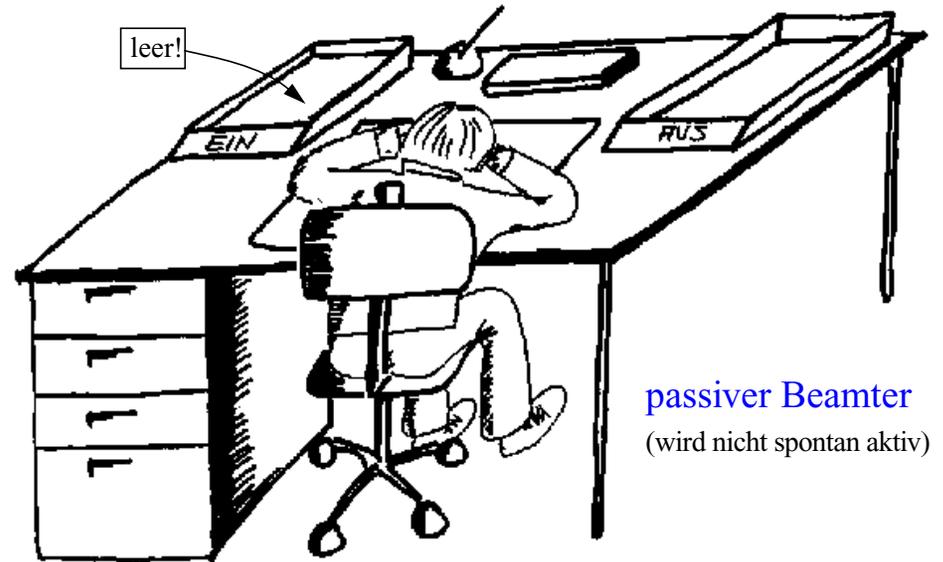
Abstraktes Verhalten einer verteilten Berechnung hinsichtlich Terminierung (ggf. existieren weitere Aktionen)

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





aktiver Beamter



passiver Beamter
(wird nicht spontan aktiv)

Die Funktionsweise der Behörde

- (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)**

Speedup ist durch die Maximalzahl gleichzeitig aktiver Beamten begrenzt

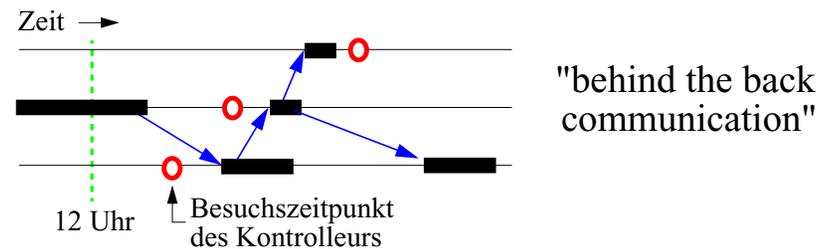
Dieser ist oft erstaunlich niedrig...

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?

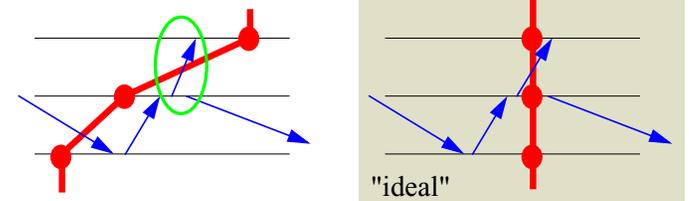
- Die *Illusion* Kontrolleurs:



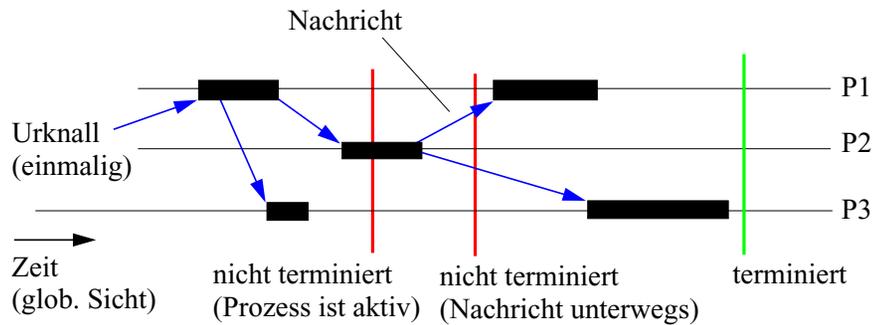
- Alle Beamten stets passiv

- \sum Nachrichten versendet = \sum Nachrichten empfangen

- Kontrolleur macht sich ein *schiefes Bild!*

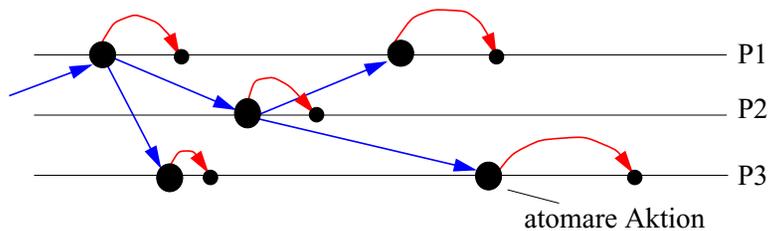


Zeitdiagramme und Atommodell



Idee: Dauer der **Aktivitätsphasen "gegen Null"** gehen lassen

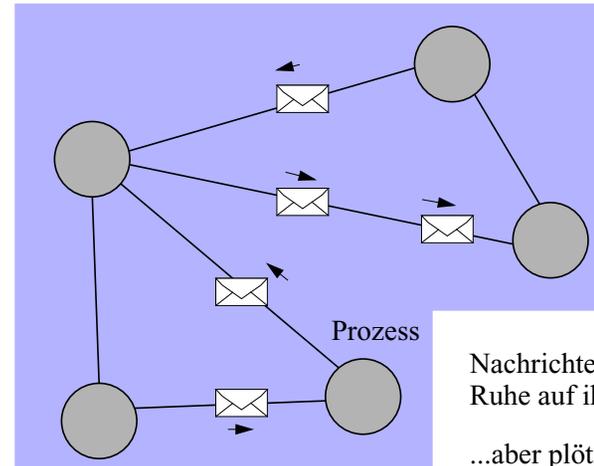
Modellierung: Prozess sendet (**virtuelle**) **Nacht** **an sich selbst**, sobald er aktiv wird; ist "unterwegs", solange er aktiv ist



Terminiert (Atommodell) \Leftrightarrow
Keine (echte oder virtuelle) **Nacht unterwegs**

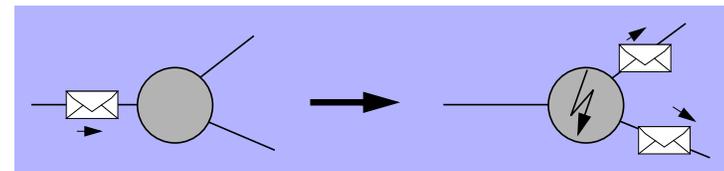
Zur Lösung des Terminierungsproblems also feststellen, **ob noch Nachrichten unterwegs sind**

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!

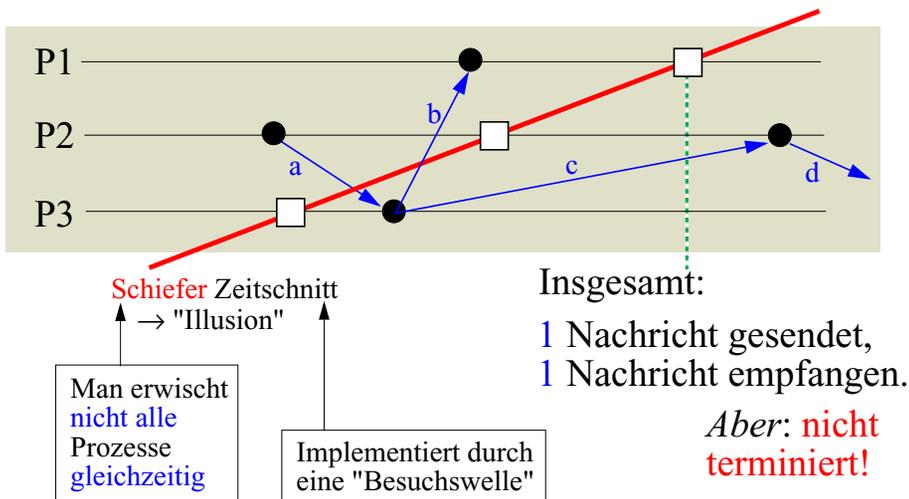


Terminiert, wenn in der globalen Sicht kein existiert

- Statt im "passiv/aktiv-Modell" genügt es offenbar, im Atommodell die Terminierungserkennung zu lösen (wieso?)
- Wie sehen die Aktionen der Basisberechnung in diesem Modell aus?

Verteilte Terminierung: Lösungen durch Zählen von Nachrichten?

- Genügt das (verteilte) **Zählen** von **gesendeten** und **empfangenen** Nachrichten?
- Einfaches Zählen genügt nicht, **Gegenbeispiel:**



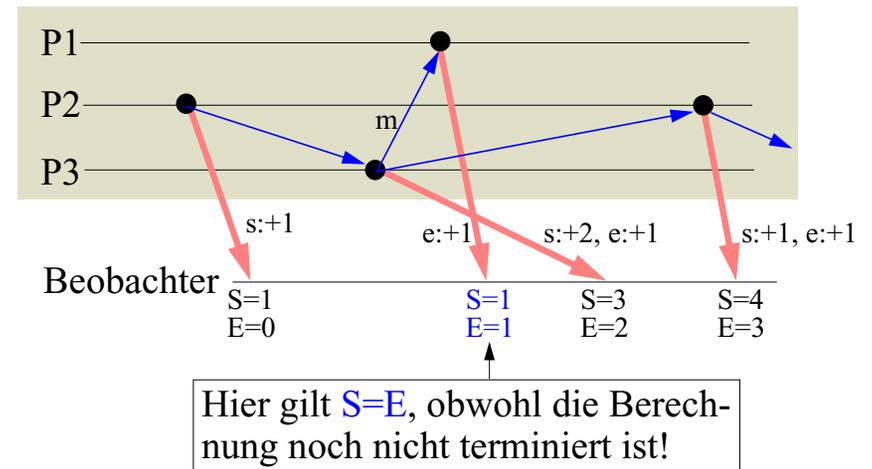
Ursache (informell):

- **Nachrichte 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?

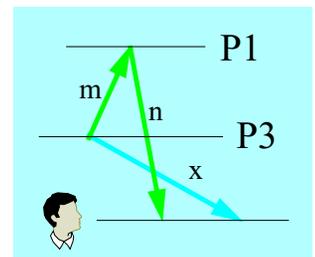
Beobachter über gesendete und empfangene Nachrichten informieren?



- 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!

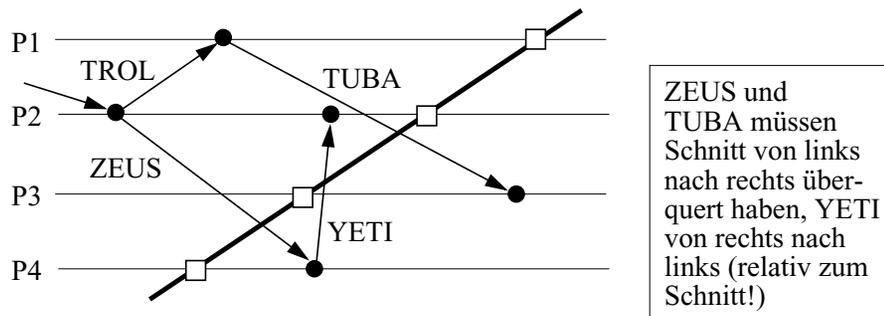


- worauf gründet sich die Vermutung?
- kann man solchermassen korrekte ("kausaltreue") Beobachtungen erzwingen?

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



- 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?)

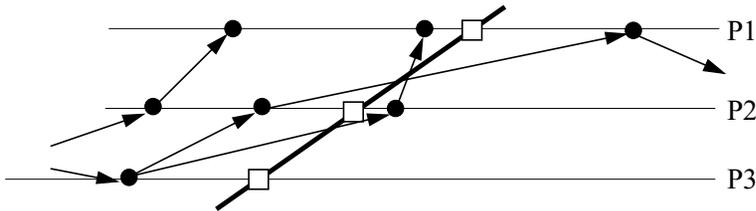
Frage: Wie geht das ganze überhaupt initial los?

Eindeutige Nachrichtennamen?

- Sender könnte Nachrichten fortlaufend nummerieren und seinen eigenen eindeutigen Namen hinzufügen
 - lässt sich einfacher verwalten als beliebige (global eindeutige) Namen
- Es genügt wohl auch eine fortlaufende Numerierung pro Sender-Empfänger-Beziehung ("Kanal")
 - z.B. 17.4.239 ("239. Nachricht von Knoten 17 an Knoten 4")
 - Verwaltungsaufwand ist recht hoch (bei FIFO benötigt man keine Mengen, es genügen $O(n^2)$ Zähler)

Genügt pauschales Zählen pro Kanal? Beweisskizze für das Kanalzählerkriterium

(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!

- Denkübung: Wäre das bei FIFO-Kanälen korrekt?

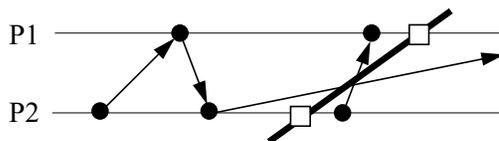
- d.h. wäre dann bei ausgeglichenen Kanalzählern keine Nachricht auf diesem Kanal unterwegs?

Behauptung (auch bei non-FIFO!):

Wenn entlang eines Schnittes *alle* Kanalzähler bzgl. send/receive ausgeglichen sind, dann überquert keine Nachricht den Schnitt

- Wieso? (intuitives Argument?)

- Beweis?

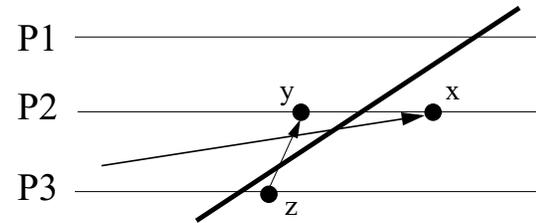


wieso ist das eigentlich kein Gegenbeispiel?

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 \Rightarrow Terminierung

- Dies ist ein Ereignis mit Empfang einer Nachricht, deren Sendereignis *links* des Schnittes liegt

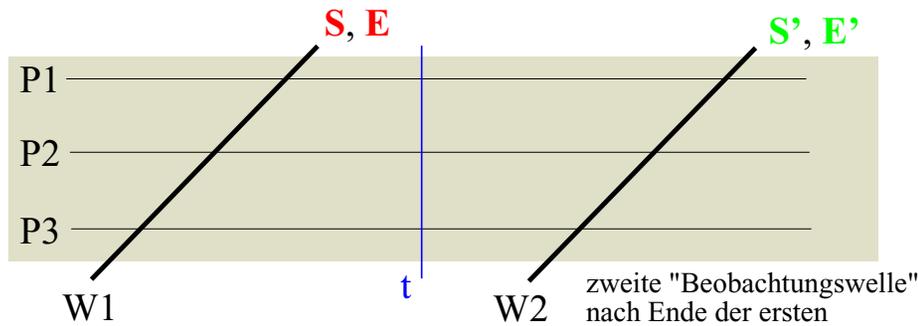
- Zugehöriger Kanalzähler kann nicht getäuscht werden, da für eine *Kompensationsnachricht* gilt: Empfangen (y) vor dem Schnitt, gesendet (z) danach

- Sendereignis der Kompensationsnachricht wäre *früheres* Ereignis *nach* dem Schnitt \Rightarrow *Widerspruch*

- Senden ist immer früher als das Empfangen einer Nachricht!
- z früher als y, y früher als x \Rightarrow z früher als x

Zählen pro Kanal ist aber etwas aufwendig ($O(n^2)$ Zähler); geht es nicht doch mit "ganz pauschalen" Zählern?

Das Doppelzählverfahren



Behauptung: $S=E=S'=E' \Rightarrow$ terminiert

d.h. keine Nachricht unterwegs

Beweis (Skizze; lässt sich auch formalisieren):

$S=S' \Rightarrow$ Keine Nachricht zwischen W1, W2 gesendet.

$E=E' \Rightarrow$ " " " " empfangen.

\Rightarrow Werte bei $t =$ Werte von W1.

Also: $S=E \Rightarrow$ zum globalen Zeitpunkt t gilt:

Anzahl gesendeter = Anzahl empfangener Nachrichten

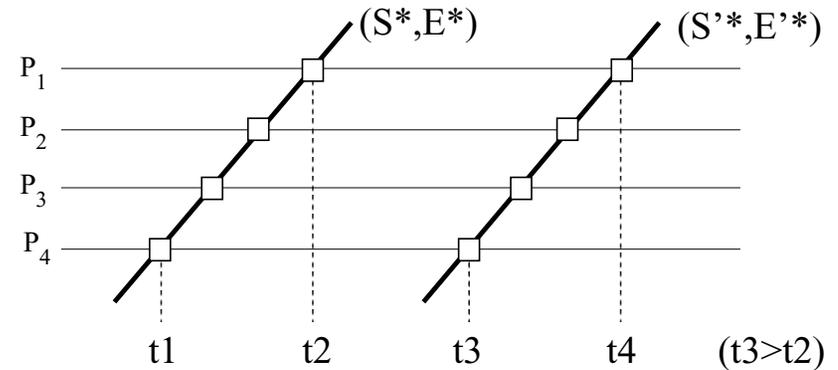
\Rightarrow zum Zeitpunkt t ist keine Nachricht unterwegs

\Rightarrow zum Zeitpunkt t terminiert

\Rightarrow Berechnung war nach W1 terminiert \square

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

Beweis:

$E^* = S'^* \Rightarrow E(t2) \geq S(t3)$ [(3), (4)]

$\Rightarrow E(t2) \geq S(t2)$ [(2)]

$\Rightarrow E(t2) = S(t2)$ [(5)]

\Rightarrow terminiert zum Zeitpunkt $t2$ \square

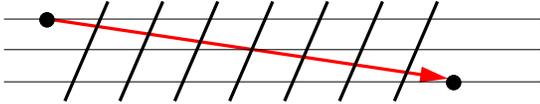
Zwei Zähler genügen!

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

Eigenschaften des Doppelzählverfahrens

- Vergleich des Empfangszählers der ersten Welle 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 (zugrundeliegender Wellenalgorithmus)
 - 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 x 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 x wird schliesslich positiv
- Bsp.: Programm terminiert
- Bsp.: erfolgte Terminierung wird schliesslich auch gemeldet

Korrektheit: Algorithmus erfüllt *Safety und Liveness*

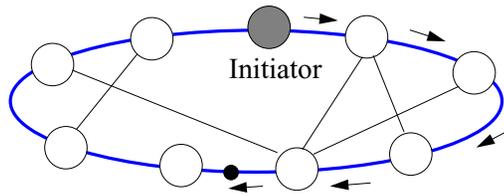
Beispiel verteilte Terminierung:

- 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* ist!
⇒ Es ist stets *safety und liveness* zu zeigen!

Kontrolltopologien

- Die für den Terminierungsalgorithmus benötigten "Wellen" können unterschiedlich realisiert werden:

1.) Ring / Hamilton'scher Zyklus:

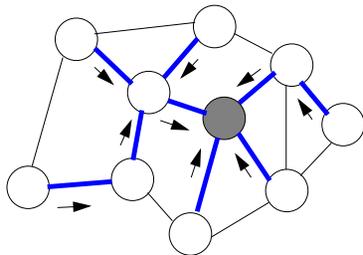


Kontrollnachricht ("Token") besucht zyklisch zwei Mal alle Prozesse und akkumuliert dabei die Zählerstände
"Logischer" Ring genügt!

- in einem zusammenhängenden ungerichteten Graphen kann ein logischer Ring immer gefunden werden, indem man einen Spannbaum "umfährt"
- Zeit- und Nachrichtenkomplexität: $O(n)$

2.) Spannbaum:

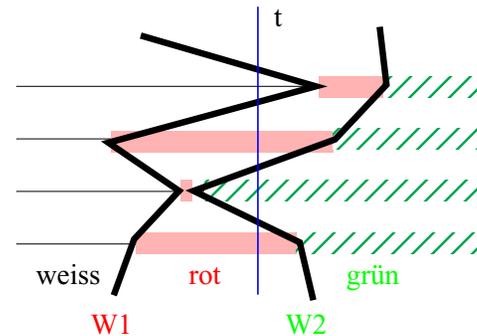
- vereinfachter Echo-Algorithmus: an den Blättern reflektierte Welle sammelt die Zählerstände in akkumulierender Weise ein



- auch hier werden *zwei* solche zum Initiator hinfließende Wellen benötigt
- bei nicht-entartetem Spannbaum: Viele Nachrichten parallel unterwegs \Rightarrow bessere Zeitkomplexität!

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!

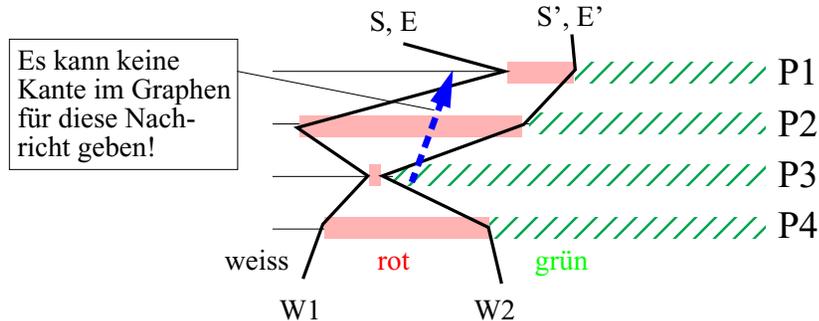


Der Echo-Algorithmus wird als *Transportdienst* zur Realisierung von *zwei* Wellen benutzt, wobei durch die Echo-Nachrichten sowohl die bei W1 lokal gemerkte Information als auch die bzgl. W2 relevante lokale Information an den Initiator übermittelt wird.

- Welle W1: "rot werden"; Welle W2: "grün werden"
- *Behauptung*: Das so realisierte Doppelzählverfahren ist korrekt
- *Problem*: Formeller oder informaler Beweis lassen sich so nicht anwenden, da sich W1 und W2 *überlappen*!

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

Zeitzoneverfahren



- "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 $S > E$.

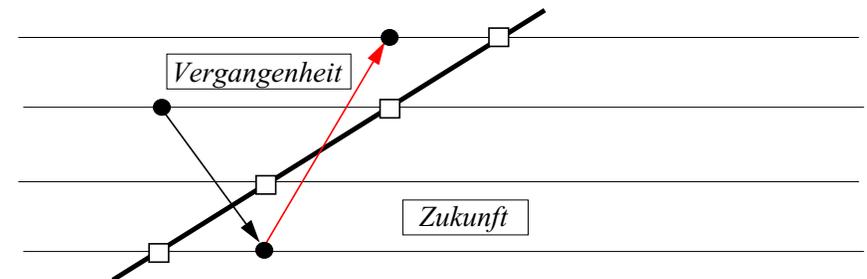
Also: Es gibt keine Nachricht, die W1 überquert; nach W1 findet daher keine Aktivität statt; das System ist nach W1 terminiert

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

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

- Nachricht aus der Zukunft → Schnitt inkonsistent

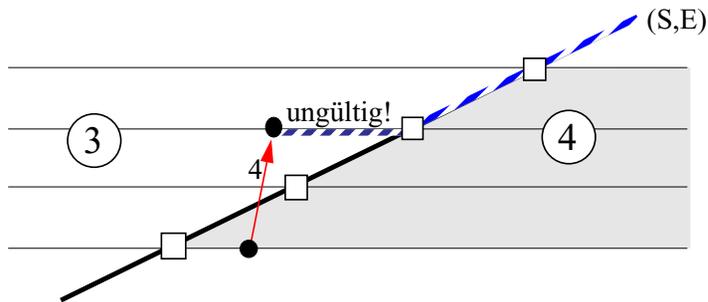
- Keine solche Nachricht → Schnitt konsistent ("äquivalent" zu senkrechtem Schnitt entsprechend Gummibandtransformation)



- 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

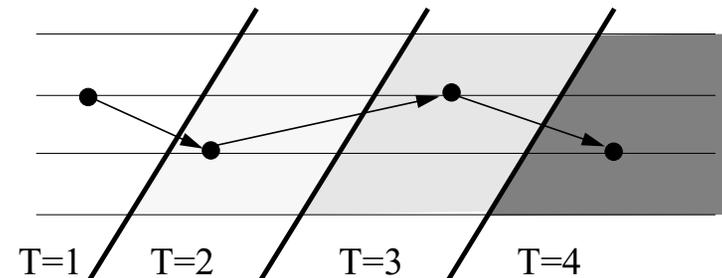
Implementierung:

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

Terminiert, wenn Welle kein gesetztes Flag feststellt und die beiden akkumulierten Zähler übereinstimmen

(Formaler Korrektheitsbeweis?)

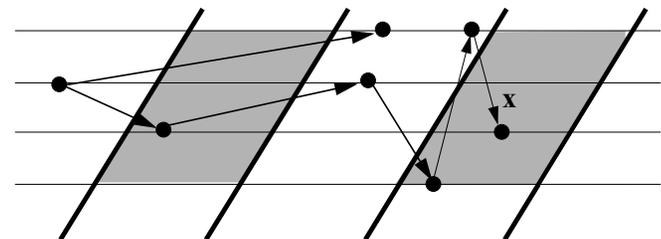
- Wiederholte Ausführung des zugrundeliegenden Wellenalgorithmus:



- Zeitzone bei jeder Runde inkrementieren

- "Zyklische" schwarz/weiss-Zeit genügt

- höhere Zeitzone → "andere" Zeitzone



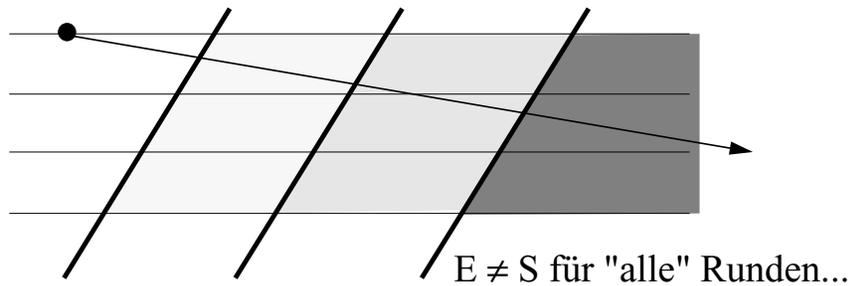
- Jede Nachricht aus der Zukunft wird erkannt

- Nachricht aus der Zukunft trägt *andere* Farbe (Nachricht rückwärts über zwei oder mehr Wellen existiert nicht)

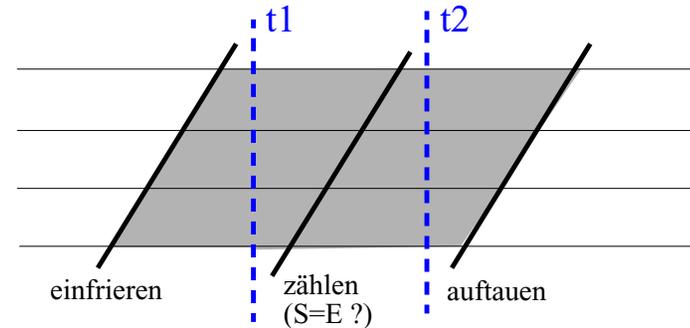
- Einige Nachrichten aus der Vergangenheit führen zu "Fehlalarmen" (vgl. Nachricht x)

- evtl. eine (einzige) zusätzliche Runde nötig

- Anzahl notwendiger Kontrollrunden? Unbeschränkt!



Einfrieren?



- 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 ggf. in einer weiteren Runde wieder auftauen): ist das korrekt?
- klappt das Vermeidungsprinzip auch ohne 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

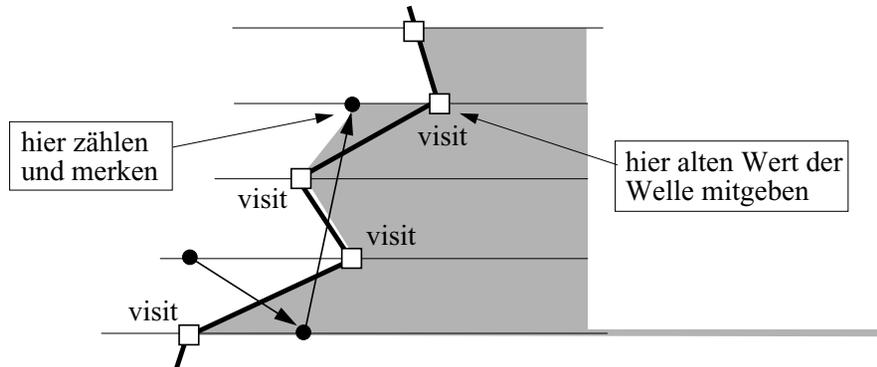
- Wie gut und praktikabel ist dieses Verfahren?

Noch ein anderes Erkennungsprinzip

- Idee: *Vermeiden* inkonsistenter Schnitte durch Vorziehen der Schnittlinie

d.h. des Kontrollereignisses zum Zählen

- Prinzip wird uns später beim Schnapsschussproblem noch nützlich sein!



- Strategie: Im Augenblick, wo Nachricht aus der Zukunft ankommt, "kurz" vorher 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
- 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)

Besprechung von Übung 1

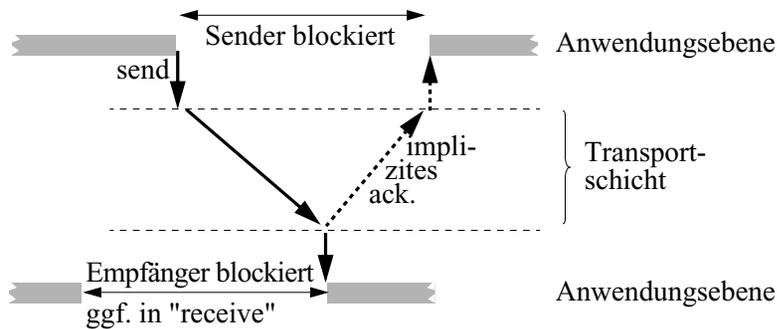
Wir wollen an dieser Stelle mündlich und an der Tafel folgende Teilaufgaben aus Übung 1 besprechen, da dies für das Verständnis der folgenden Aspekte ("synchrone Kommunikation") wesentlich ist:

- g) *Formalisieren* Sie für *Zeitdiagramme* den Begriff (potentiell, indirekt) "*kausal abhängig*" als *Halbordnung* über "*Ereignissen*".
- i) Beobachtungen sind eine *lineare Ordnung* von (beobachteten) Ereignissen. In welcher Beziehung steht die oben erwähnte Halbordnung zu dieser linearen Ordnung?

Synchrones / asynchrones Senden

- *Asynchrones Senden*: Absender wartet nach dem Absenden der Nachricht nicht ("no wait send")
 - erfordert ggf. 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



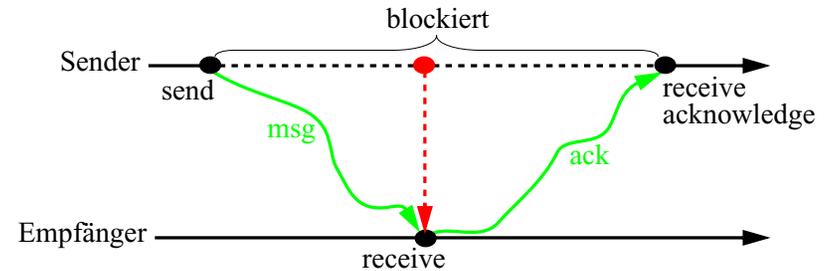
- Aus Sicht des ("bewusstlosen") Senders ist die Nachricht im Augenblick des Sendens auch schon angekommen
 - als wäre die Nachricht "unendlich schnell" (senkrechte Pfeile!)
 - Denkübung: Kann man nicht immer ein Zeitdiagramm per Gummibandtransformation "schadlos" so verzerren, dass ein Nachrichtenpfeil 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**
 - als wäre der Sender vor und nach dem virtuellen Sendezeitpunkt "idle"

- **Konsequenz:**
 Man darf so tun, **als wären Nachrichten nie unterwegs!**

- "blitzschnelle" Nachrichten
- vereinfacht viele Argumentationen
- formale Unterscheidung synchron / asynchron durch Kausalrelation (synchron: Abhängigkeit des Senders vom Empfänger)

Modeling Synchronous Communication

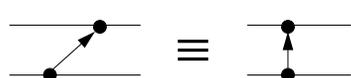
- How do we *define* distributed computations with synchronous message passing?
 - or: *characterize* those distributed computations that can be *realized* with synchronous communications?

- Proposition:

Synchronous = virtually simultaneous

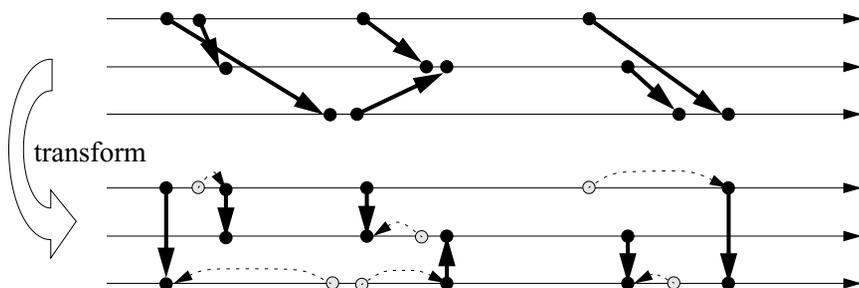
= as if msg transmission were instantaneous

suitable rubber band transformation?



- But: aren't instantaneous message transmissions unrealistic?

- Can we *always* apply a suitable rubber band transformation such that all message arrows become vertical?



“As if” Messages were Instantaneous?

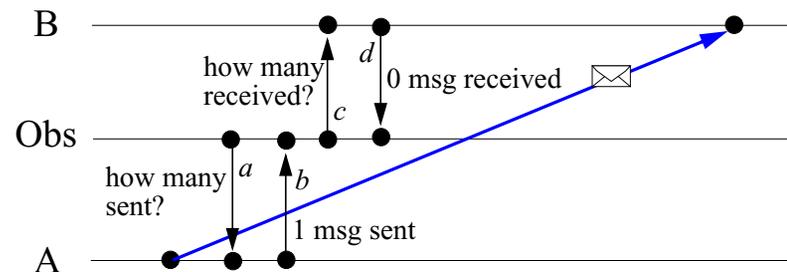
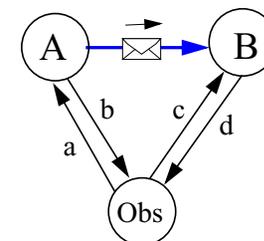
If for a distributed computation a phenomenon can be observed which is impossible with instantaneous messages, the computation must not be realizable with synchronous message passing semantics

⇒ message passing should then not be called “synchronous”

Example:

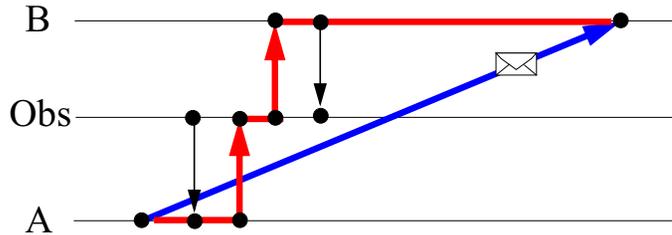
The observer first asks A about the number of messages it sent to B

Then it asks B about the number of messages it received from A



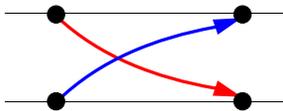
Observer learns that a message from A to B is *in transit* for a certain *duration* ⇒ not synchronous!

Vertical Message Arrows?



- The message from A to B is *overtaken in an indirect way* by a *chain* of other messages
- The direct message can therefore *not be made vertical* by a rubber band transformation
(A message of the chain would then go backwards in time)

- Another computation which is not possible with synchronous communications (\Rightarrow deadlock):



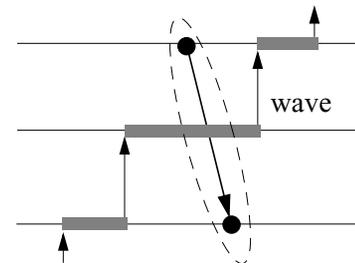
Although each *single* arrow can be made vertical, it is not possible to draw the diagram in such a way that *both* arrows are vertical!



Various Characterizations of Synchronous Communications

- Question: are they all equivalent?
- Note: some characterizations are informal or less formal than others

- 1) Best possible approximation of *instantaneous* communications
(i.e., without clocks, it is not possible to prove that a message was not transmitted instantaneously)
- 2) Space-time diagrams can be drawn such that *all message arrows are vertical*
- 3) *Communication channels* always appear to be *empty*
(i.e., messages are never seen to be in transit)
- 4) Corresponding *send-receive events* form one *single atomic action*



- But what exactly does “atomic” mean?
- Does the combined event happen before or after the wave? Should this be possible with synchronous communication?

8) Synchronous causality relation \ll is a *partial order*

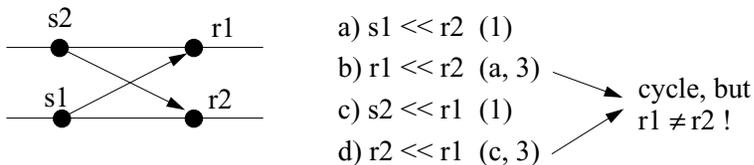
Definition of \ll : compare this to the classical causality relation!

1. If a before b on the same process, then $a \ll b$
2. $x \ll s$ iff $x \ll r$ ("common past")
3. $s \ll x$ iff $r \ll x$ ("common future")
4. Transitive closure

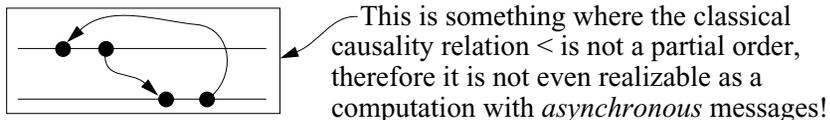
Interpretation: corresponding s, r are *not* related, but with respect to the synchronous causality relation they are "identified"

they have the same past and future

Example:

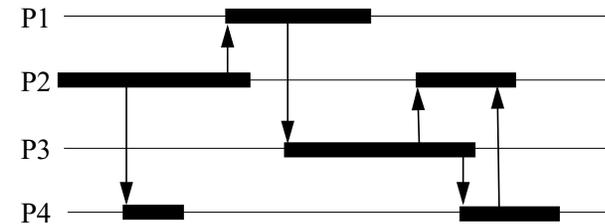


- Compare this characterization to characterization 6 ("scheduling relation")
- Why is the definition of \ll sensible?



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

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...

Verhaltensmodelle vert. Anwendungen und Definition der vert. Terminierung

Nachrichtengesteuertes Modell ("Transaktionsmodell"):

- Alle Prozesse passiv und keine Nachricht unterwegs

Atommodell:

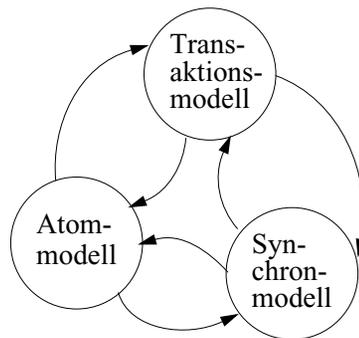
(Prozesse sind immer passiv)

- Keine Nachricht unterwegs

Synchronmodell:

(Nachrichten haben Laufzeit 0)

- Alle Prozesse passiv



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

Aufgabe: Man gebe jeweils geeignete Transformationen an!

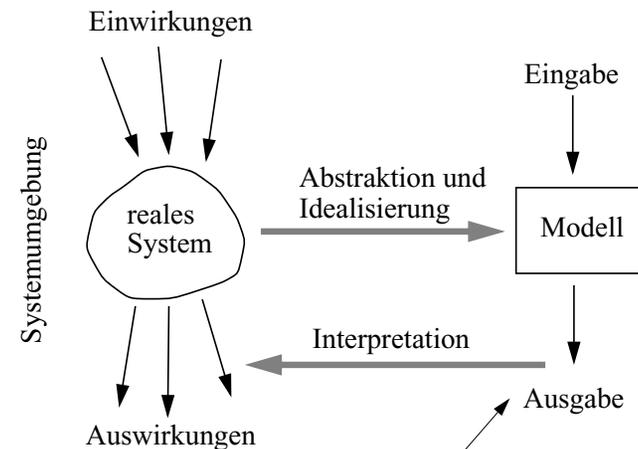
- Was ist das "richtige" Modell?

- Hängt von der Anwendung ab!
- Hängt ggf. von Abstraktionsniveau und Sichtweise ab!
- Ein Modell lässt sich ggf. *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, Vergrößerung!
- 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 denknötigen 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



Vert. Algo. 05/06, F. Ma. 117

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 (Schwereelosigkeit)
 - 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")

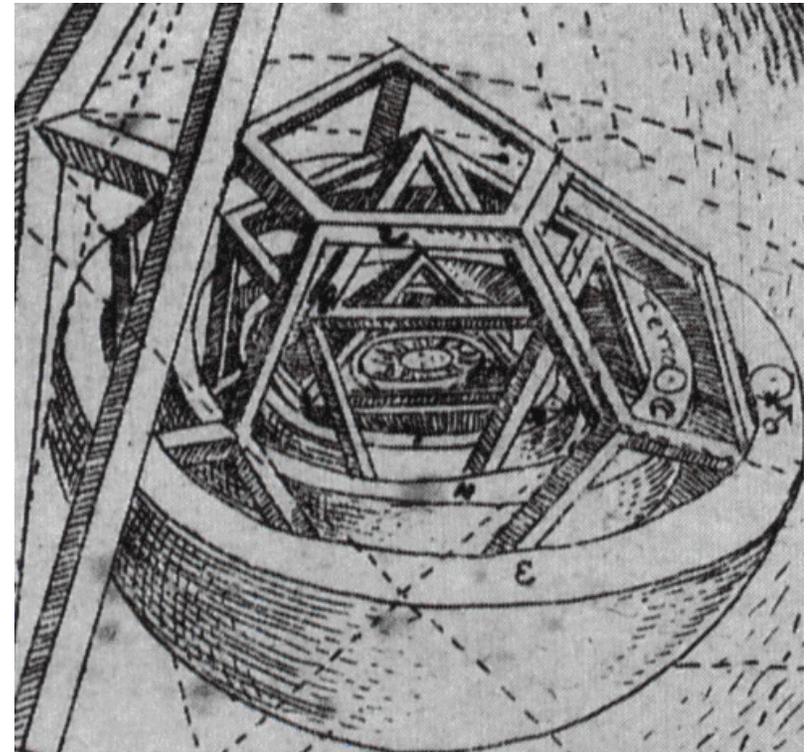
Vert. Algo. 05/06, F. Ma. 118

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



Ein Modell der Planetenbahnen aus verschachtelten platonischen Körpern

(J. Kepler, 1597)



Zweck der Modellierung

- Zweck von Modellierung bei der *Simulation der* (ggf. 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 around us*
[Kreutzer: System Simulation]

Implementierung von Modellen

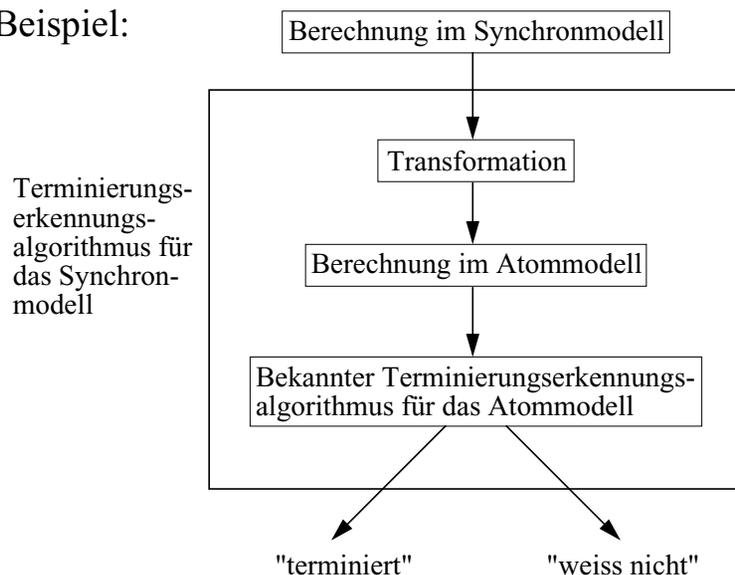
- Im Unterschied zu den *Naturwissenschaften* geht es in der *Informatik* oft nicht um die möglichst detailgetreue Nachbildung der Wirklichkeit, sondern um die *Implementierung* eines Modells als "ausgedachte Wirklichkeit"
- Weil das ausgedachte Modell "schön" ist, z.B.:
 - als hätte man einen sehr grossen 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 (IRC, MUD, Chat, Avatar, Computerspiel,...)
 - als ob...
- Das heisst: Man erschafft sich eine "Wirklichkeit"
 - künstliche Wirklichkeit = virtuelle Realität

Exkurs-Ende

Modelltransformation

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

- Beispiel:



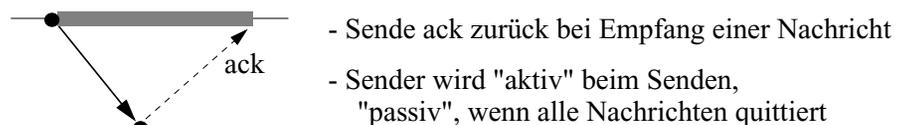
- Transformation muss natürlich *korrekt* sein: Wenn die transformierte Berechnung beendet ist, dann ist auch die Originalberechnung beendet (\rightarrow safety); wenn die Originalberechnung beendet ist, dann ist schliesslich auch die transformierte Berechnung beendet (\rightarrow liveness)

Ein Beispiel für eine Modelltransformation

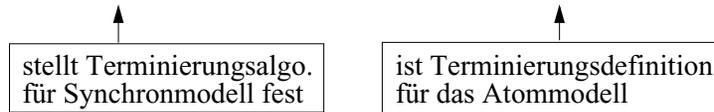
Gegeben:

- Berechnung im Atommodell
 - Terminierungsalgorithmus für Synchronmodell
- (statt Atommodell)

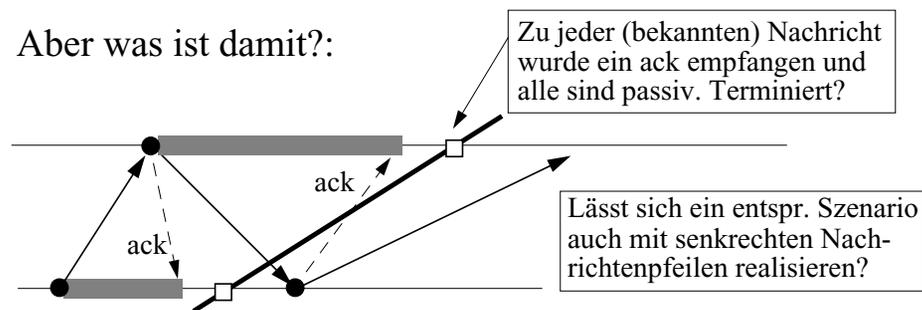
Was tun? \rightarrow Einführung von "passiv" und "aktiv" (Modelltransformation)



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



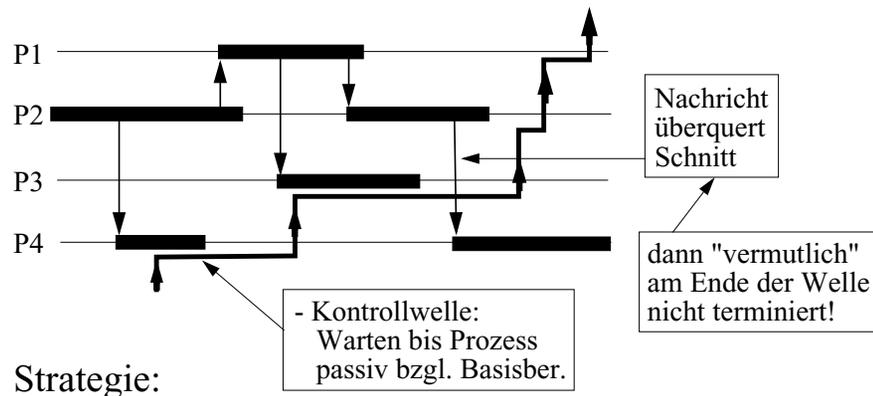
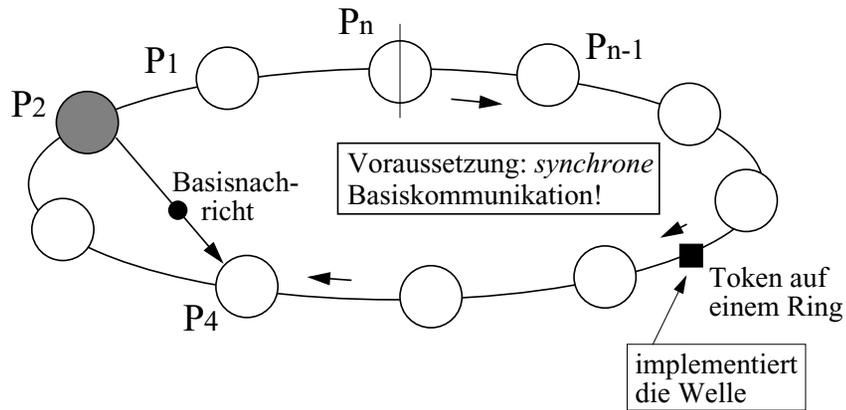
Aber was ist damit?:



- "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



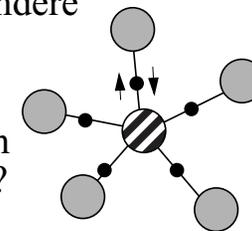
Strategie:

- 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 alle weiss

DFG-Algorithmus: Verhaltensregeln

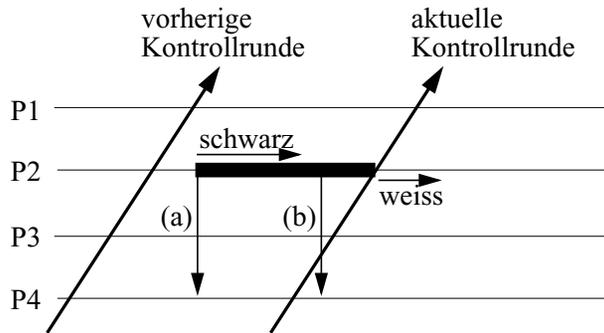
- Regel 1:** Ein Prozess, der eine Basisnachricht [an einen Prozess mit einem höheren Index] sendet, wird schwarz.
- Regel 2:** Wenn Prozess P_n passiv ist, kann er den Terminierungstest initiieren, indem er ein weisses Token an P_{n-1} sendet.
- Regel 3:** Ein aktiver Prozess behält das Token, bis er 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?



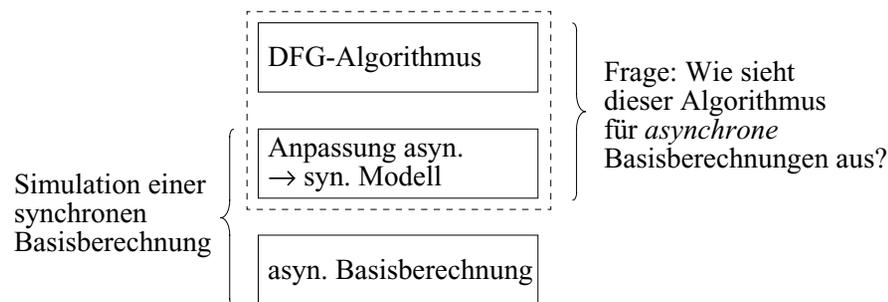
z.B. Stern mit einem zentralen Initiator, der *parallel* Token zu den Prozessen schickt, um deren Farbe zu ermitteln

DFG-Algorithmus - "falscher Alarm"

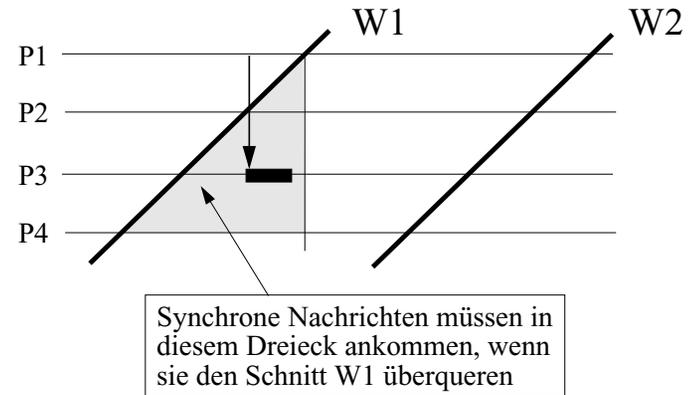


- Die beiden Situationen (a) und (b) werden nicht unterschieden, obwohl nur (b) gefährlich ist
- Konsequenz: Wenn im Gebiet zwischen den beiden Runden eine Nachricht an einen höheren Prozess gesendet wird, ist in jedem Fall noch eine weitere Runde nötig
- Vereinfachung von Regel 1 (Konsequenz?)

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)

ACHTUNG: Algorithmus ist falsch!

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, Atommmodell, 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.

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, \dots, p_n . Let a local predicate c_i be associated with process p_i , $1 \leq i \leq n$. 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 ($1 \leq i \leq n$) 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, $1 \leq i \leq n$, 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 , $1 \leq i \leq n$, 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 ($1 \leq i \leq n$)

1. *Upon becoming passive:*
 $state(p_i) := \text{passive};$
/ $state(p_i)$ has been used for recording the status of p_i */*
for each $p \in \{\text{neighbours of } p_i\}$ **do**
 send I-am-passive to p ;
2. *Upon receiving an I-am-passive message from p ($p \in \{\text{neighbours of } p_i\}$):*
begin
 $state_i(p) := \text{passive};$
 / $state_i(p)$ is used for recording the status of process p in p_i */*
 if $state_i(p_j) = \text{passive}$ for all $p_j \in \{\text{neighbours of } p_i\}$
 and $state(p_i) = \text{passive}$
 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 $state_i(p_j) = \text{passive}$ for all $p_j \in \{\text{neighbour of } p_i\}$
 and $state(p_i) = \text{passive}$
 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_j 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 for-

warded 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.

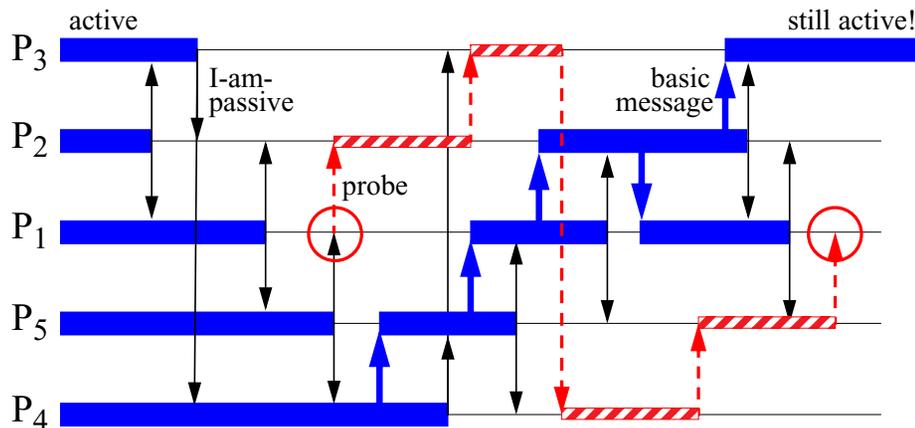
References

- [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.

Zu Übung 3: Gegenbeispiel zum Algorithmus von Arora, Rana, Gupta

- Man überlege zunächst:
 - setzen die Autoren *synchrone* Kommunikation bei der Basisberechnung voraus?
 - ... bei den Kontrollnachrichten ("I-am-passive", probe)?

- Hier: Ringtopologie sowohl für Kontroll- als auch Basisnachrichten

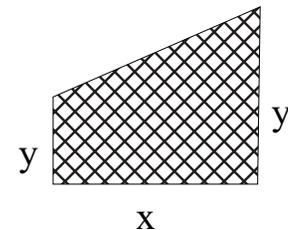
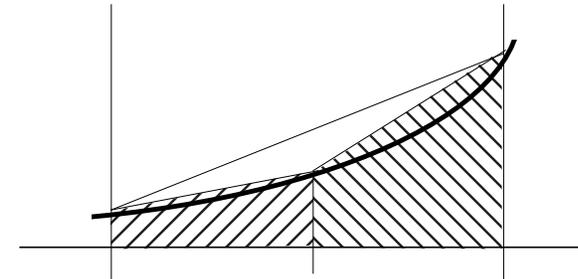


- Siehe auch: Letter to the Editor "Comments on Distributed Termination Detection Algorithm for Distributed Computation", Inf. Proc. Lett. 23 (1986) pp. 163

- Aber auch die Antwort der Autoren: Letter to the Editor "More Comments on Distributed Termination Detection Algorithm for Distributed Computation", Inf. Proc. Lett. 29 (1988) pp. 53-55

Parallele Berechnungsschemata und verteilte Terminierung

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



$$A = x \frac{y + y'}{2}$$

Prinzip:

- x-Intervall iterativ / rekursiv halbieren bis zu einem *Stop-Kriterium*
- Teiltrapezflächen dann aufaddieren

Beachte: Mathematische Aspekte hier ausgeklammert!

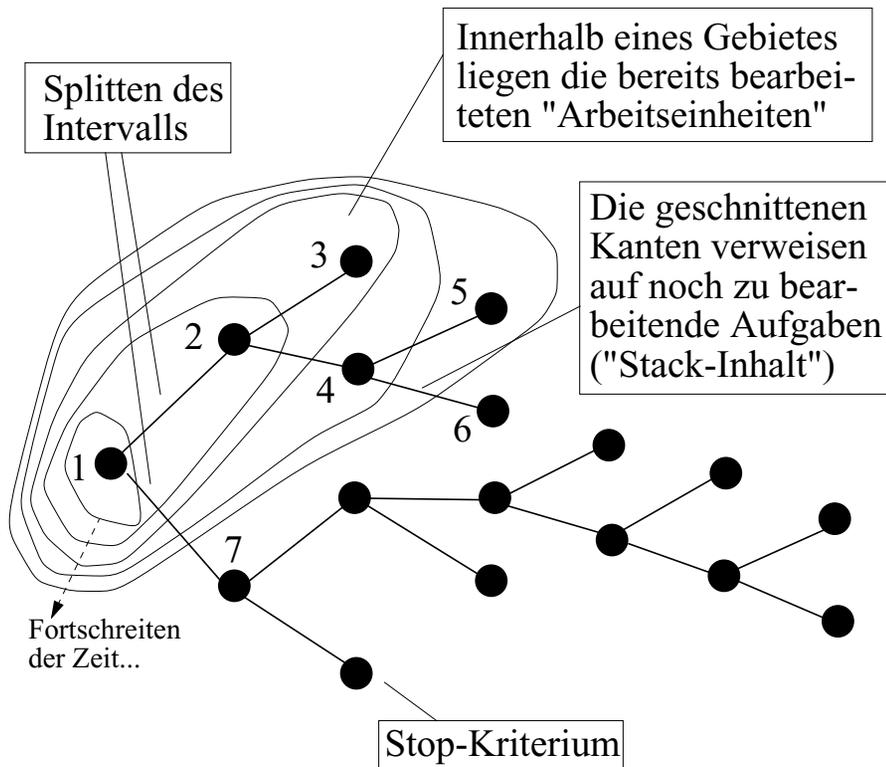
Mögliche *Stop-Kriterien*:

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

Sequentielle Berechnung

- Iterative bzw. rekursive Berechnung ist (uns) klar

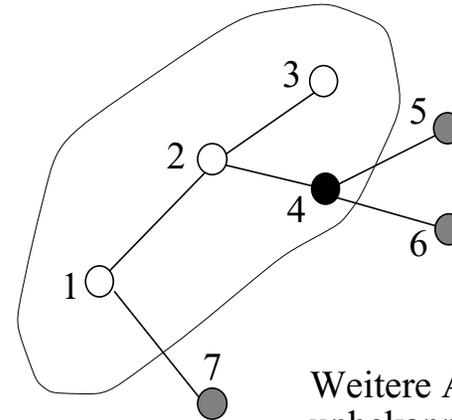
- *Visualisierung:*



- Teilflächenwerte stehen an den Blättern!

Parallelisierung

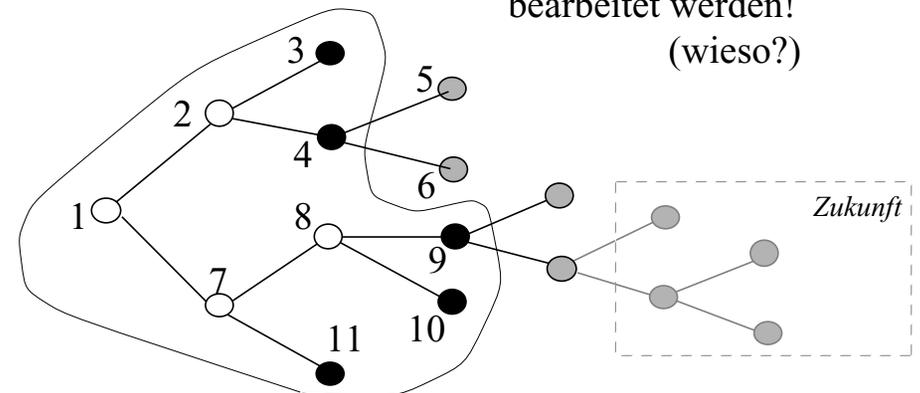
"Schnappschuss" der sequentiellen Berechnung:



- 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

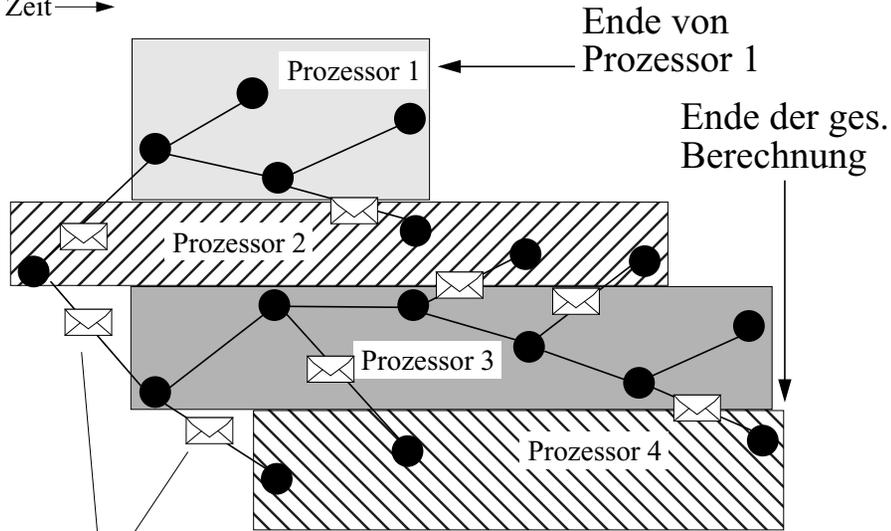
Parallelisierung: Aufgaben 3,4,9,10,11 können gleichzeitig bearbeitet werden!
(wieso?)



"Zeithorizont" dehnt sich in mehrere Richtungen gleichzeitig aus

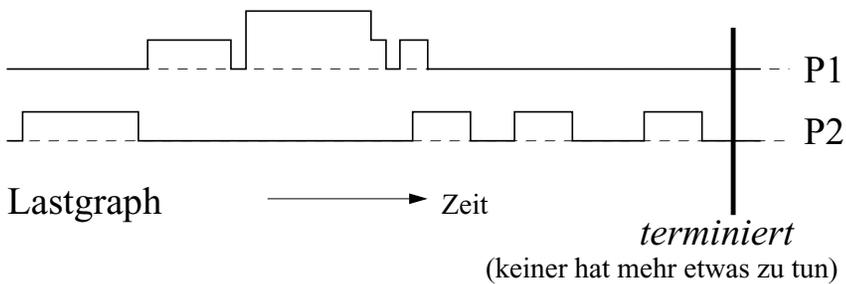
Lastausgleich und Terminierung

Hier globale Sicht:
Zeit →

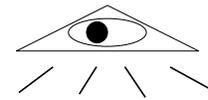


Übermittlung von Arbeitseinheiten an andere Prozessoren

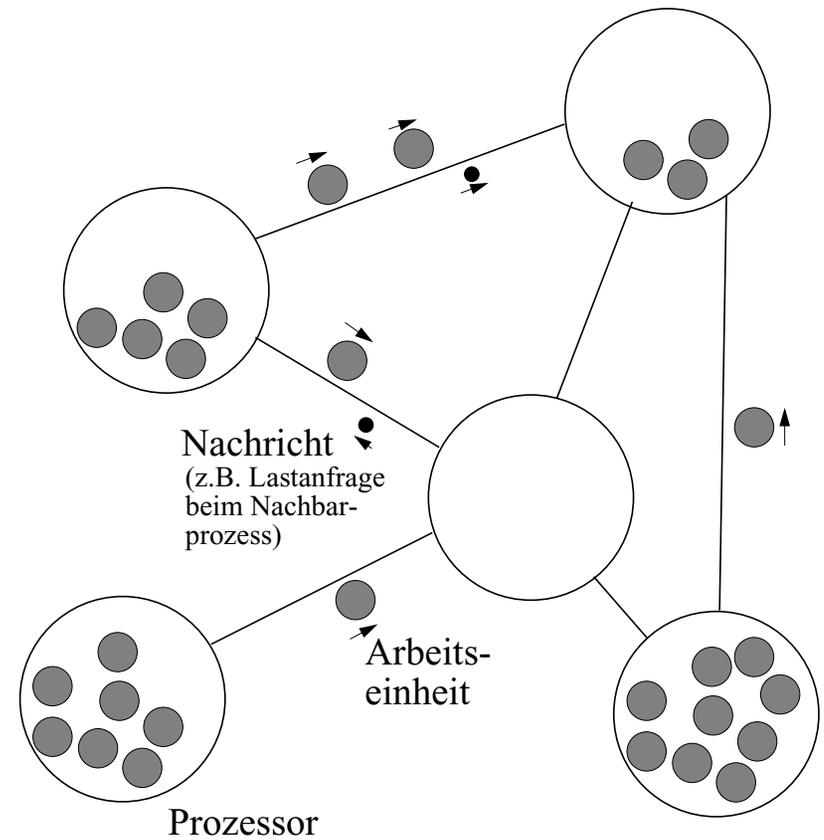
- a) neu entstandene Einheiten
- b) Lastausgleich



Globaler Schnappschuss einer Berechnung

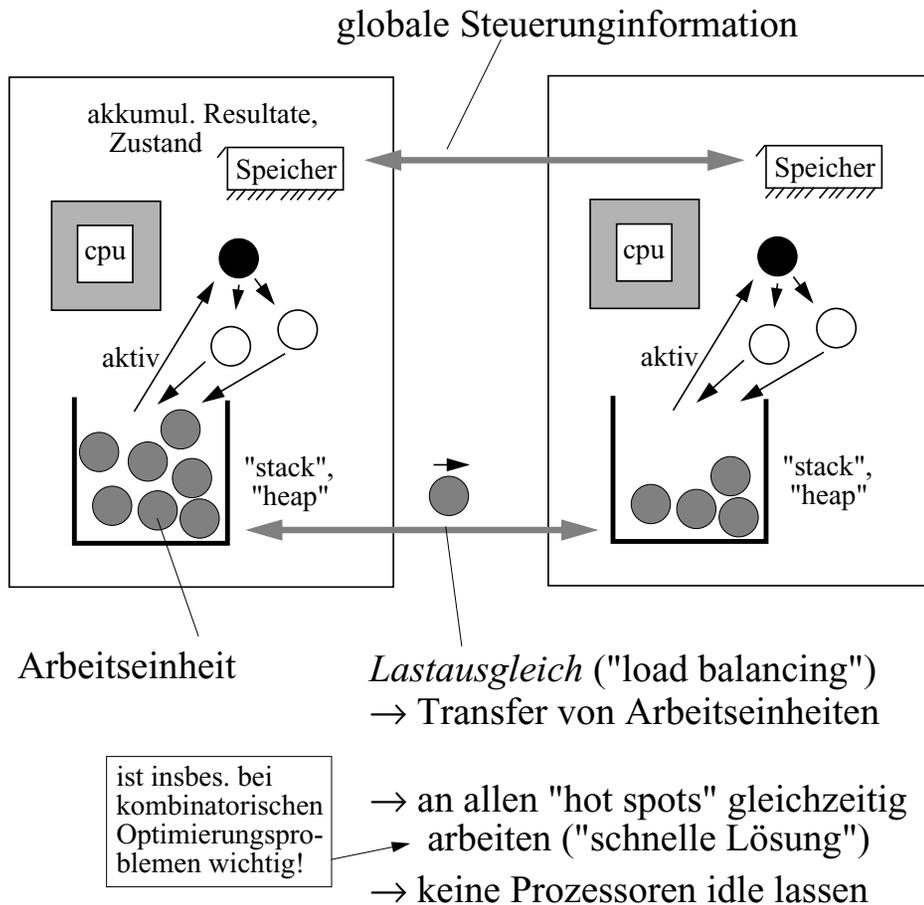


Globale Sicht
(hat kein Prozessor!)



Terminiert, wenn Last überall 0 und nichts "unterwegs"

"Operationale" Sichtweise



Was ist Last? (Mass für Last?)

Wie funktioniert Lastausgleich?

→ verschiedene Modelle, Strategien, Heuristiken, Verfahren...

US Patent 6 112 225

August 29, 2000 / Filed: March 30, 1998

Assignee: International Business Machines Corporation (Armonk, NY)

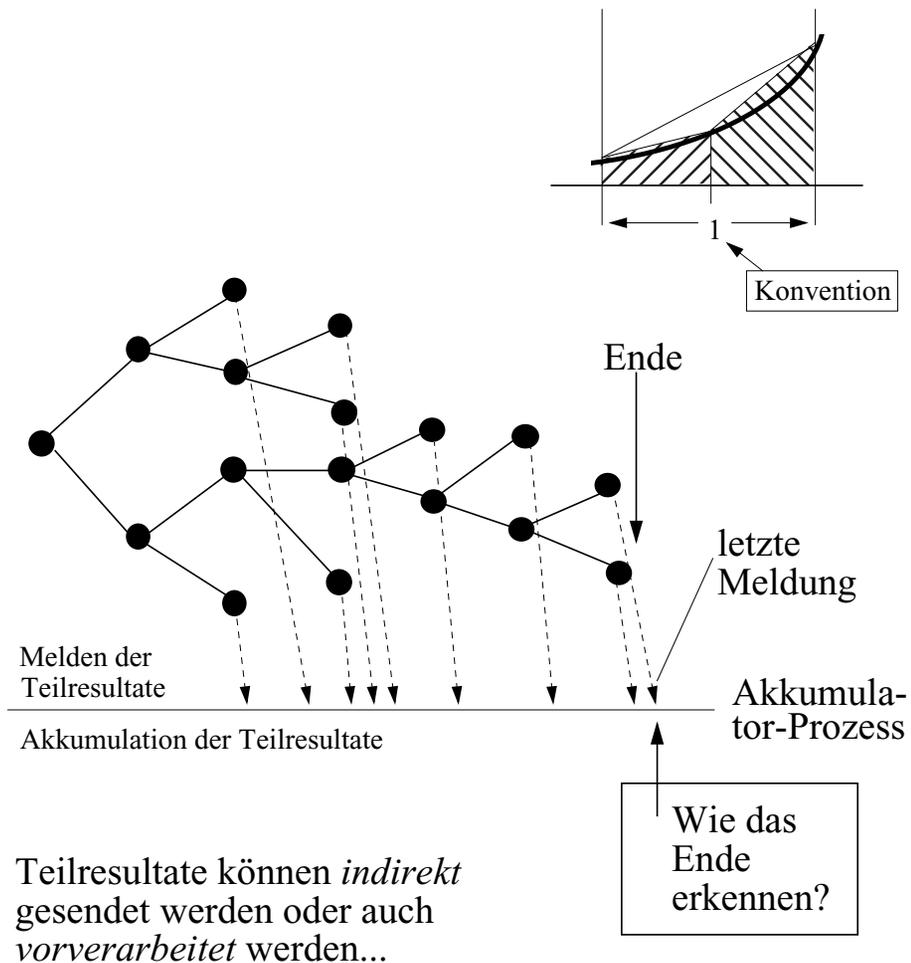
SUMMARY OF THE INVENTION

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.

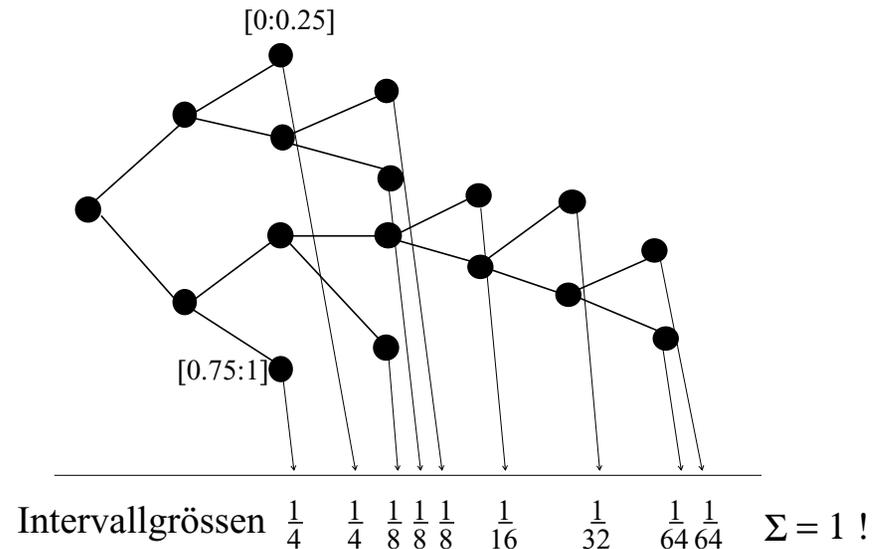
What is claimed is:

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



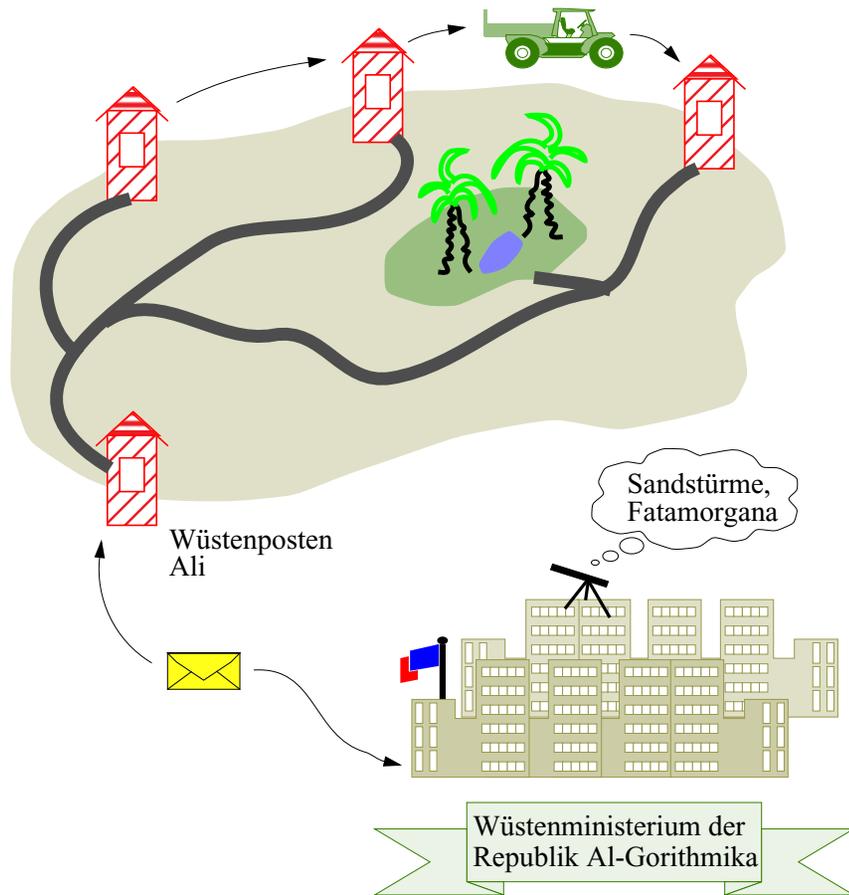
Lösung des Terminierungsproblems



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 \rightarrow *terminiert*

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

- Neue anfordern \rightarrow 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 $\in \mathbb{Q}^+$
- (3) Summe aller Kreditanteile stets = 1
- (4) Aktiver Prozess hat Kreditanteil > 0
- (5) Nachricht hat Kreditanteil > 0

Invariante

Theorem: Dann Berechnung *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 wiedererhält ("*liveness*")

Rekombination der Kreditanteile

- Einsammeln der Kredite auch durch beliebigen Wellenalgorithmus möglich (initiiert durch Urprozess)
 - dabei Rekombination eingesammelter Kredite ggf. hierarchisch

- Rekombination der Kreditanteile beim *Urprozess*:

- Initial: $D = \{0\}$;

steht für $2^{-\text{KREDIT}}$

- Bei Empfang von $\langle \text{KRÜMEL} \rangle$:

```

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: $D = \{0\}$

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

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

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

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

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

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

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

\rightarrow Terminierung!

$$\underline{\underline{\Sigma = 1}}$$

\Rightarrow Prinzip: Binäre Subtraktion!

Kreditmethode - Bewertung

- Topologievoraussetzungen?
- Zentraler Urprozess: Engpass?
 - ggf. weitere Hierarchiestufen einführen
 - Kreditanteile durch einen Wellenalgorithmus einsammeln
- (Worst-case) Nachrichtenkomplexität? (= Overhead)
 - wenn passive 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
 - 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 (Kreditanteile)
- 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 wieder zurückzuholen
 - *Diesen* Typ von Kontrollnachrichten kann man sich sparen! ("Verheiraten" mit der zugehörigen Basisnachricht)
 - Acknowledgement wird sofort generiert, wenn der Empfänger aktiv ist.
 - Aber wenn der Empfänger passiv ist, 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? (→ 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 als Instanzen (einer einfachen *Variante*) des Echo-Algorithmus hinsichtlich der Explorer-Nachrichten auffassen. Die Echo-Nachrichten übernehmen dann wie gehabt die Erkennung der Terminierung!

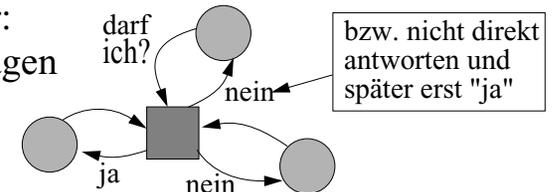
Wechselseitiger Ausschluss



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

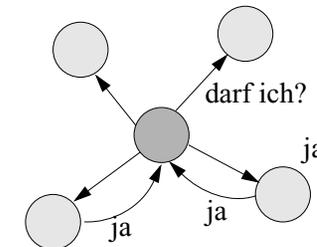
1) Zentraler Monitor:
Einen einzigen fragen

- 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

- Lassen sich Schemata als Ausprägung eines gemeinsamen allgemeinen Prinzips verstehen?

- "ein für alle mal" verifizieren
- abstraktere Sicht liefert hoffentlich tieferes Verständnis
- ggf. neue Varianten mit interessanten Eigenschaften

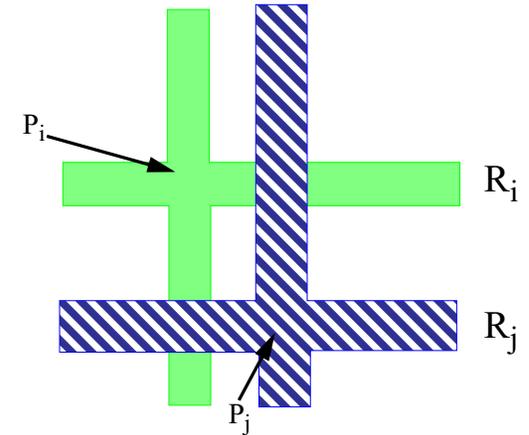
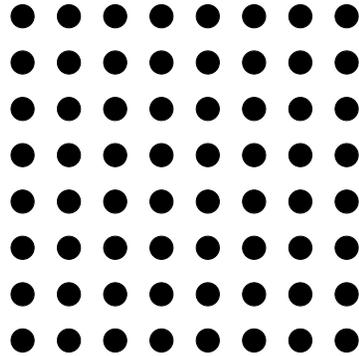
Siehe dazu Artikel von B. Sanders (ACM TOCS 5, 284-299, '87)

Maekawa's \sqrt{n} -Algorithmus (1985)

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

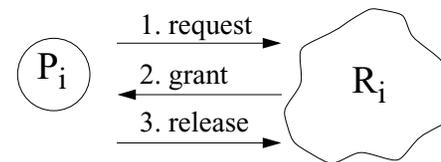
Idee *in etwa*:

- 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:

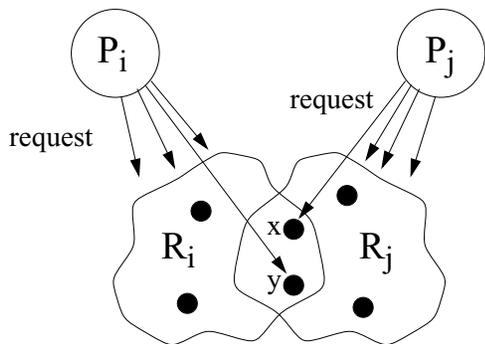


Ein Prozess wartet auf "grant" seiner Menge. Erst dann darf er den kritischen Abschnitt betreten. Nach Verlassen Menge mit "release" informieren.

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

Eine Erlaubnis ("grant") wird zu einem Zeitpunkt nur *einem* Bewerber erteilt.

Deadlock-Problematik



Beachte: Zweckmässigerweise ist oft $P_k \in R_k$, 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
- ⇒ *Deadlock*, P_i und P_j warten auf weitere Zusage!

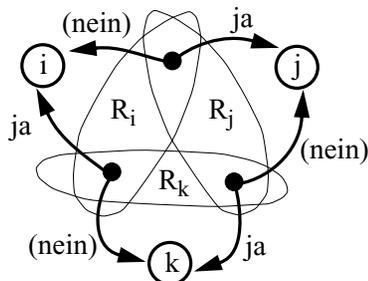
Lösung erfordert weitere Nachrichtentypen zur Deadlockvermeidung (bzw. Deadlockbehebung → Symmetriebrechung)

- ⇒ soll hier nicht behandelt werden (→ Literatur)
- ⇒ 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!

- Prozesse 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 (→ $|R_i| \approx \sqrt{n}$ statt $2\sqrt{n}-1$)
 - *Endliche projektive Ebene* entsteht aus affiner Ebene unter Hinzunahme von "uneigentlichen Punkten" (=Parallelenbündel) und 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
 - 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

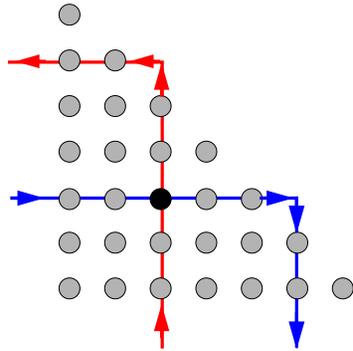
Beispiel $n=13, k=4$:
 (Überprüfe, ob $|R_i \cap R_j| = 1$)

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

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

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

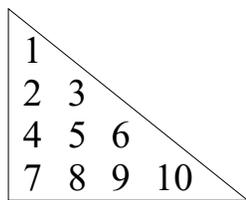
Schema 2 für R_i : Nimm alle Prozesse der Spalte von P_i sowie alle Prozesse derjenigen Zeile, die eins weiter oben liegt als der oberste Prozess der Spalte

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

Wir lassen das hier aber weg.

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

- Beispiel:



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

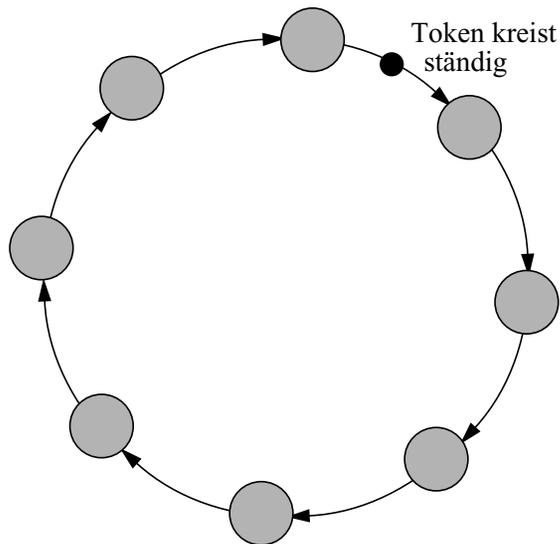
Schema 1	Schema 2
$R_1 = \{1, 3, 5, 8\}$	$R'_1 = \{1, 2, 4, 7\}$
$R_2 = \{2, 3, 6, 9\}$	$R'_2 = \{1, 2, 4, 7\}$
$R_3 = \{2, 3, 6, 9\}$	$R'_3 = \{1, 3, 5, 8\}$
$R_4 = \{4, 5, 6, 10\}$	$R'_4 = \{1, 2, 4, 7\}$
$R_5 = \{4, 5, 6, 10\}$	$R'_5 = \{1, 3, 5, 8\}$
$R_6 = \{4, 5, 6, 10\}$	$R'_6 = \{2, 3, 6, 9\}$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

- 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?

Mehr dazu (und weitere Verfahren) siehe: Luk, Wong, "Two New Quorum Based Algorithms for Distributed Mutual Exclusion", Proc. 17th Int. Conf. Distributed Computing Systems, 1997, pp. 100-106

Die Token-Ring-Lösung



- 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

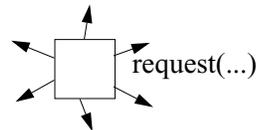
- Safety ist vergleichsweise trivial (Tokenbesitz)

- Liveness mit möglichst wenig Aufwand garantieren

Ricart / Agrawala 1983 und Suzuki / Kasami 1985

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

2) Request-Nachricht mit (log.) Zeitstempel an *alle* senden



3) Token hat "Gedächtnis":

Prozessnummer	Zeitstempel des letzten Besuchs (ggf. 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 wird das Token an denjenigen Prozess geschickt, der am "längsten" wartet (Anforderung muss jünger als Zeitstempel des letzten Besuchs beim Prozess sein)

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

- Fragen:
- Wie lange muss ein Prozess max. (auf Mitbewerber) warten?
 - Liveness? Fairness?
 - Geht es auch mit weniger Nachrichten?
 - Geht auch physische Zeit statt Lamport-Zeit?

Token-Request-basierte Algorithmen

aber wie bekommt man diesen?

- Andere Idee: *Spannbaum* verwenden

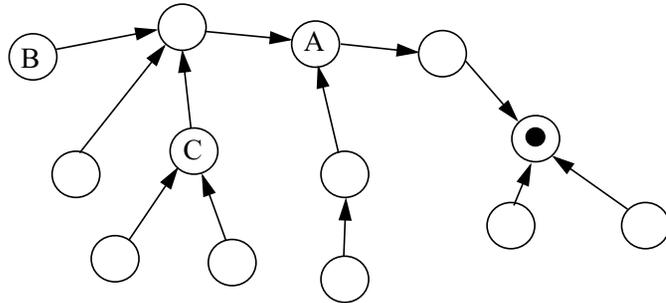
- "Suchnachrichten" wandern nur auf Kanten des Baumes
- Token benutzt ebenfalls nur Baumkanten
- Aufwand: Maximal $n-1$ Einzelnachrichten, um jeden der n Knoten über den Tokenwunsch zu informieren

- 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 gutartig / böartig entarteten Bäumen?)
- wenn Token seinen Ort wechselt, müssen Kantenrichtungen aktualisiert werden! (Aufwand?)

Der "O(log n)"-Algorithmus

Das wäre aber besser als das "optimale" $O(\sqrt{n})$ -Verfahren!?!



Verallgemeinerung auf allg. Graphen

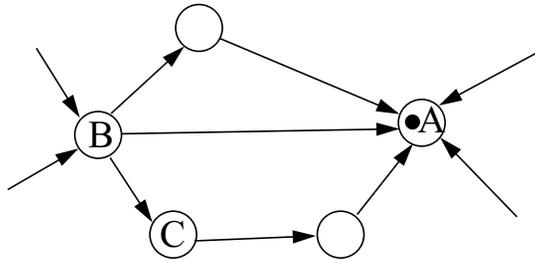
- Gerichtet, azyklisch und schwach zusammenhängend
- Kann man die Kanten jedes ungerichteten Graphen so orientieren, dass kein Zyklus entsteht?
 - Bsp. Stadtplanung: Einbahnstrassenrichtung so festlegen, dass man nicht im Kreis fahren kann
 - Ja: Willkürliche Ordnung auf den Knoten festlegen (z.B. Knoten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

- Token wandert entgegen der Pfeilrichtung zum anfordernden Prozess und dreht jede durchlaufene Kante um
- Ein Prozess sendet (bis er das Token erhält) nur ein Mal ein 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?
 - 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 ein request hinterhersenden (Optimierung: Token und request zusammenfassen)
- Sind Nachrichtenüberholungen ein Problem?
 - z.B.: request überholt Token

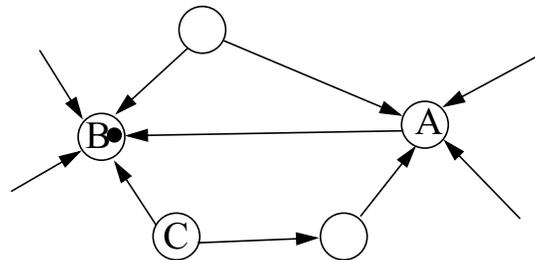
- Präzisere Forderung: Jeder gerichtete Weg soll beim (eindeutigen) Tokenbesitzer enden
 - Klar, dass dann Tokenbesitzer nur eingehende Kanten hat
 - Auch das geht!
 - Starte Echo-Algorithmus vom Tokenbesitzer aus
 - Durch die Flussrichtung der Echos entsteht ein auf diesen Knoten hin gerichteter Baum
 - Nun müssen noch die Nicht-Baumkanten orientiert werden
 - Dazu bekommt jeder Knoten eine Ordnungsnummer aus:
 - seiner Höhe (= Entfernung zur Wurzel = 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.a. verschieden lang)
 - auch denkbar: über alle Pfade gleichzeitig... (→ mehr Aufwand!)



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



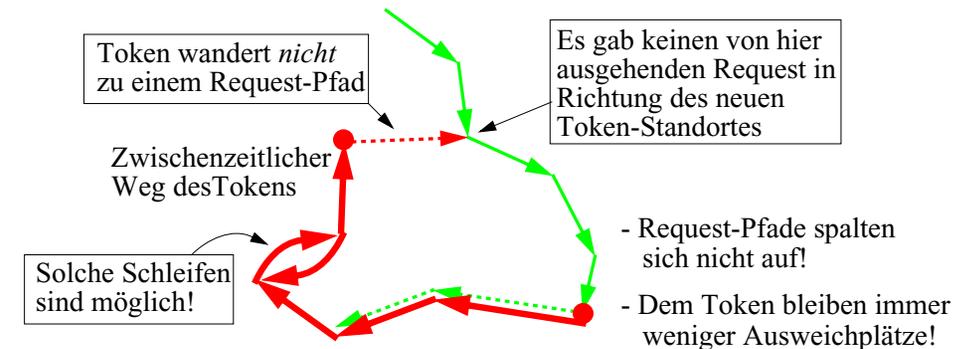
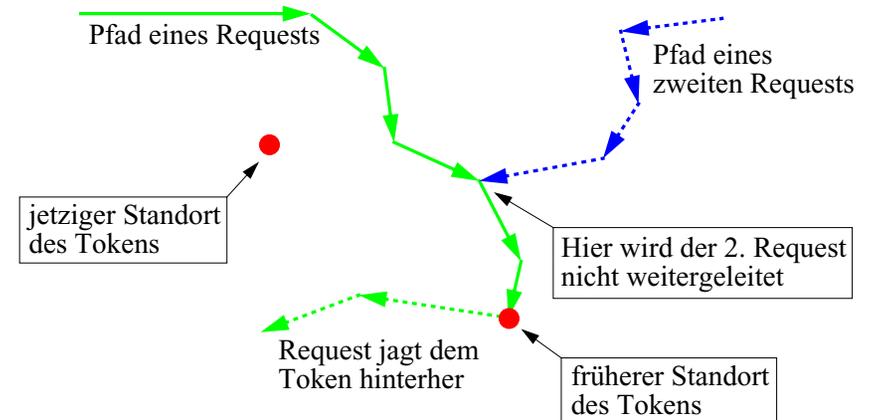
- Behauptung: *Zyklenfreiheit ist eine Invariante*

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 endet beim Tokenbesitzer*

Glücklose Token-Jagd?

- Wieso holt ein Request das Token schliesslich immer ein?



- Request-Pfade spalten sich nicht auf!
- Dem Token bleiben immer weniger Ausweichplätze!

Welcher Baum beim $O(\log n)$ -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!

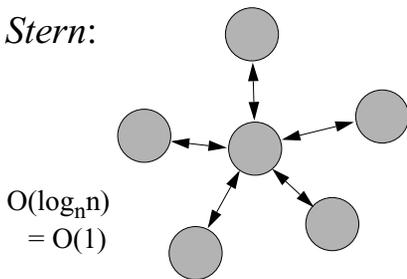
fast nie mehrere gleichzeitige Konkurrenten

- Grad an Fairness?

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 also "besser"!

- *Stern*:

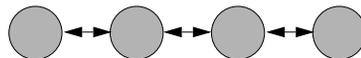


- Kürzeste Wege → beste Topologie? (D.h. wenn schon Baum, dann diesen?)

- Unterschied zur Lösung mit zentralem Manager? (Und was war daran so 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 $\approx n/3$ (Beweis: einfaches kombinator. Nachrechnen)

Klassifikation: Token \Leftrightarrow Request

1) Token-basierte Lösungen

- 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)

Vergleich der Nachrichtenkomplexität

(pro Betreten des kritischen Abschnittes bzw. Anforderung des Betriebsmittels)

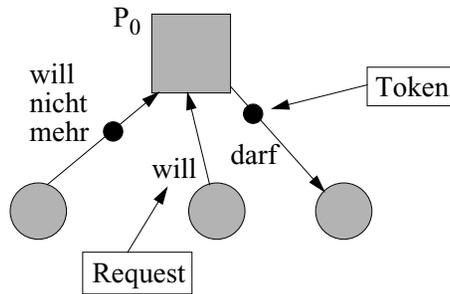
2) Request-basierte Lösungen

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

3) Zentraler Manager

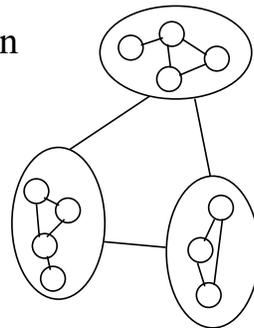
bzw. Server oder Monitor

- ist entarteter Fall beider Klassen!



4) Hierarchische / hybride Verfahren

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



Token-Ring (LeLann,1977)	1 ... ∞
Lamport (1978)	3 (n-1)
Ricart / Agrawala (1981)	2 (n-1)
("an optimal algorithm...")		
Ricart / Agrawala (1983)	n
Maekawa (1985)	$O(\sqrt{n})$
Lift-Algorithmus auf Baum (1987)	$O(\log_k n)$
Zentraler Manager	2

Vorlesung "vert. Systeme"

- Offenbar soll es aber eben nicht alleine auf die Nachrichtenkomplexität ankommen!

- "Qualitative" Merkmale oft wichtiger!

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 $O(\log n)$ -Algorithmus als symmetrisch bezeichnen?

- Fehlertoleranz

- Verhalten des Algorithmus bei Fehlern
 - z.B. 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 wird zeitlich globale Reihenfolge der Requests eingehalten?

- Zeitbedarf zwischen Freigabe und Benutzung durch einen anderen Prozess

- (minimale) Länge sequentieller Nachrichtenketten
- Bsp.: bei $O(\log n)$ -Algorithmus ist Zeitbedarf auch $O(\log n)$ (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 "Stopregel" "grösserer Fisch als der erste gefangene oder kein Fisch mehr übrig" bei einem Teich/See/Meer von n verschieden grossen Fischen mit $n=1000$, $n=10^6$, $n=2^{31}$, $n=10^{13}$?

(b) ... wenn schneller Fische geboren werden als Alois fangen kann?

(c) ... wenn "grösserer" durch "kleinerer" in der Stopregel ersetzt wird? Diskutieren Sie dies im Vergleich zu (a)!

Sie mögen dies simulieren, wenn Sie an den theoretischen Ergebnissen zweifeln. 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.) Mitteln Sie jedes "Makroexperiment" über viele, typischerweise $O(n)$ Einzelerperimente (also Kollegen von Alois). Vergleichen Sie die Mittelwerte mehrerer Makroexperiment (stabil, verlässlich?). Ermitteln Sie auch die relative Häufigkeit von $m = 1, \dots, 10$ sowie von $m = n - 1$ und $m = n$ ("leergefischt") Fängen.

(2) Wartezeit bis zum ersten echten Rekord:

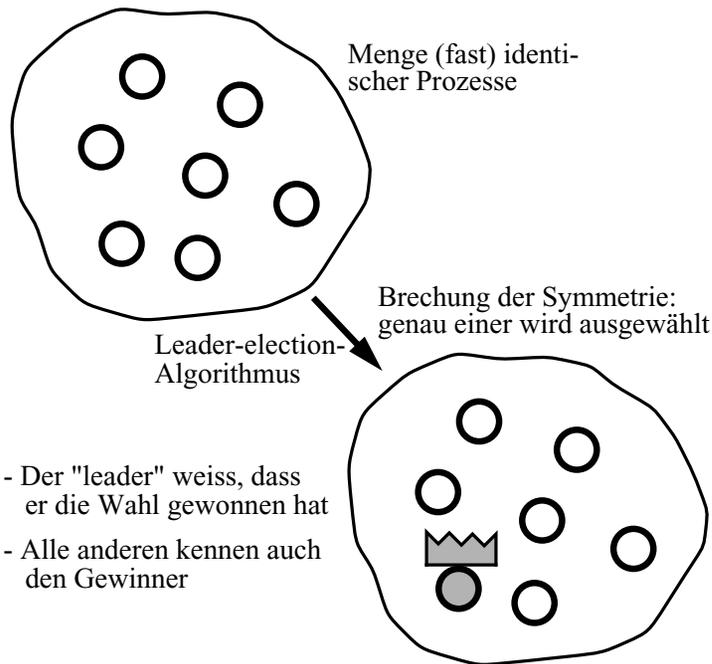
(a) Nach wievielen Jahren ist in einem Jahrhundert im Mittel der erste echte (d.h. verschieden vom ersten Jahr, das immer einen unechten "Rekord" darstellt) Rekord "kältester Januar" fällig?

(b) Und in einem Jahrtausend? Darf hier etwas anderes herauskommen? Diskutieren! (Ungläubige mögen wieder simulieren.)

(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 ganz vernachlässigen?)

Das Election-Problem

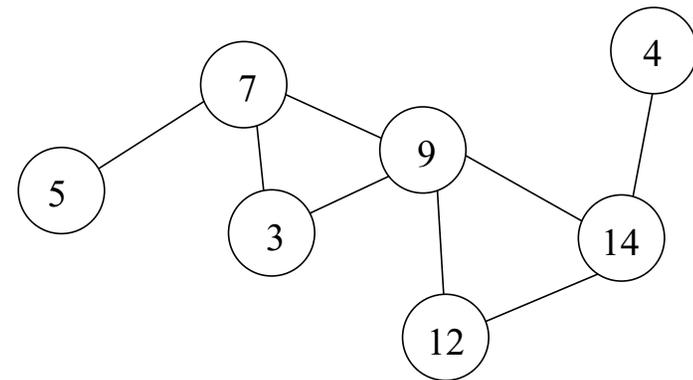


Anwendung z.B.:

- 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"

- Alle Knoten haben unterschiedliche Nummern > 0
- Verteilter, symmetrischer Algorithmus zur Bestimmung der grössten Identität
- Jeder Knoten soll schliesslich grössten kennen
- Jeder Knoten darf unabhängig (gleichzeitig) den Algorithmus initiieren
- Schema der verteilten Approximation anwenden

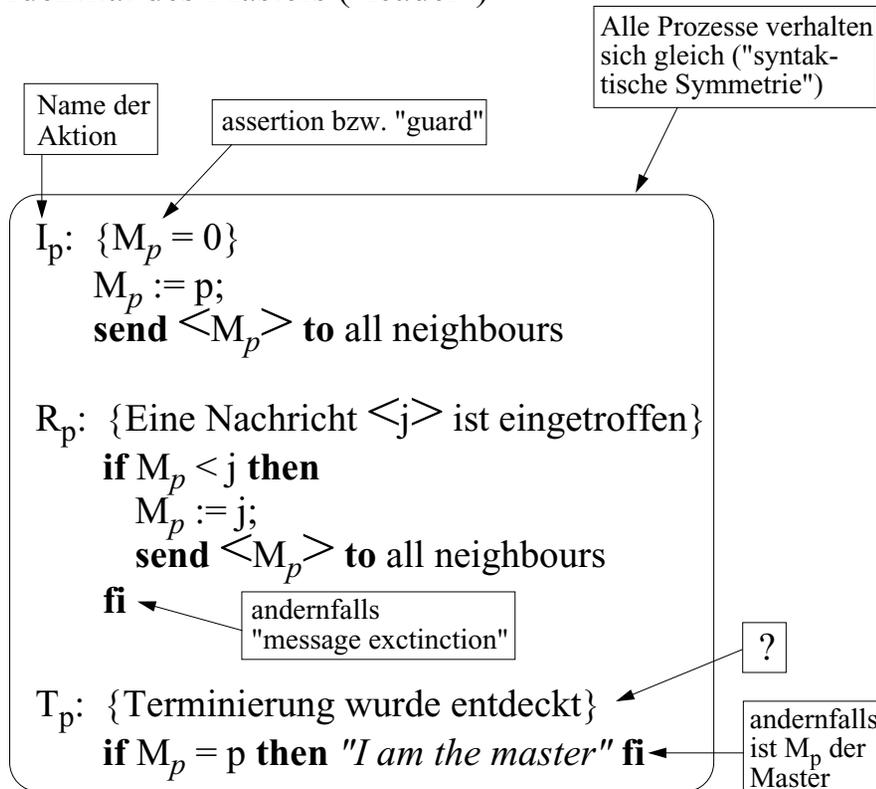


Fragen:

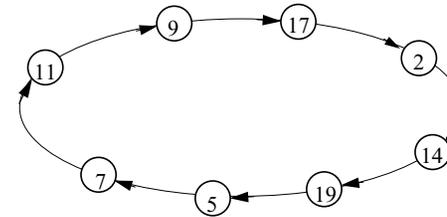
- Anzahl der Nachrichten?
- Terminierung?
- Bessere Algorithmen für gleiches Problem?
- Spezielle Topologien (Ring, Baum)...?

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 ist initial 0; am Ende enthält M_p die Identität des Masters ("leader")



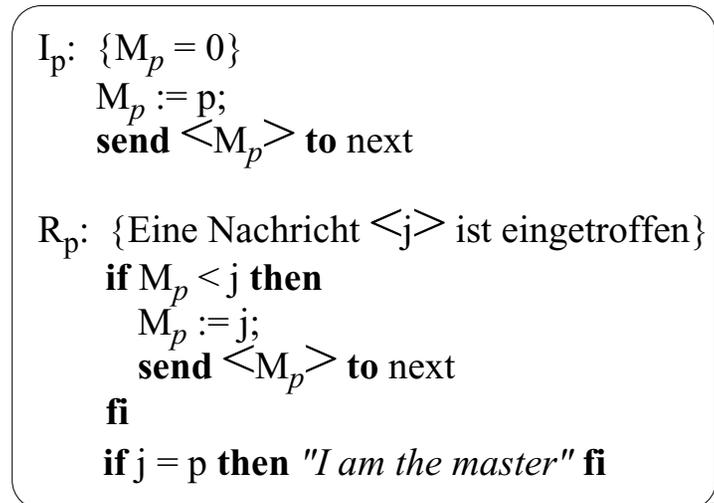
Leader-Election auf einem Ring



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

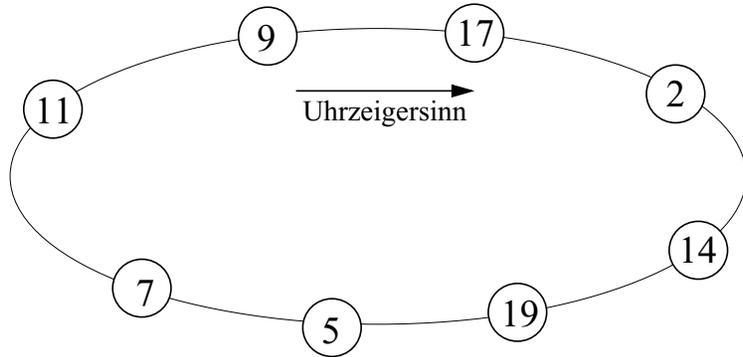
1. Idee: - Jeder Prozess wacht irgendwann auf (spätestens wenn er eine Nachricht von einem anderen erhält)
- "Bully-Algorithmus"
- Startet vollständigen Ringumlauf
 - Meldet, ob unterwegs einen grösseren getroffen
- ⇒ n^2 Einzelnachrichten ($n = \text{Anzahl der Prozesse}$)

2. Idee: Message-extinction anwenden!



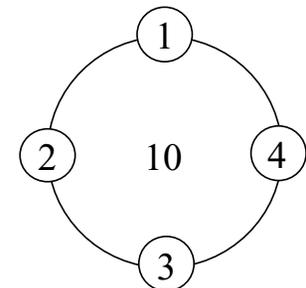
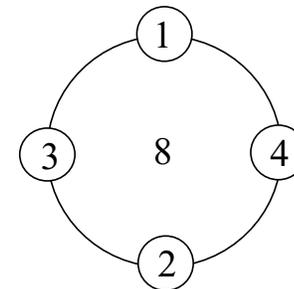
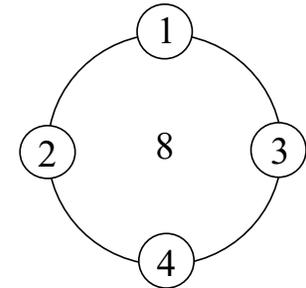
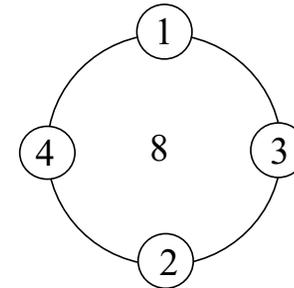
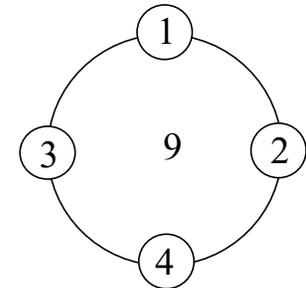
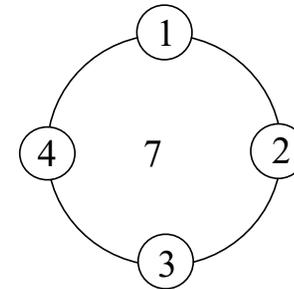
Election: Nachrichtenkomplexität

- Message-extinction-Prinzip von Chang und Roberts 1979
 - war einer der ersten verteilten Algorithmen überhaupt



Mittlere Nachrichtenkomplexität (1)

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

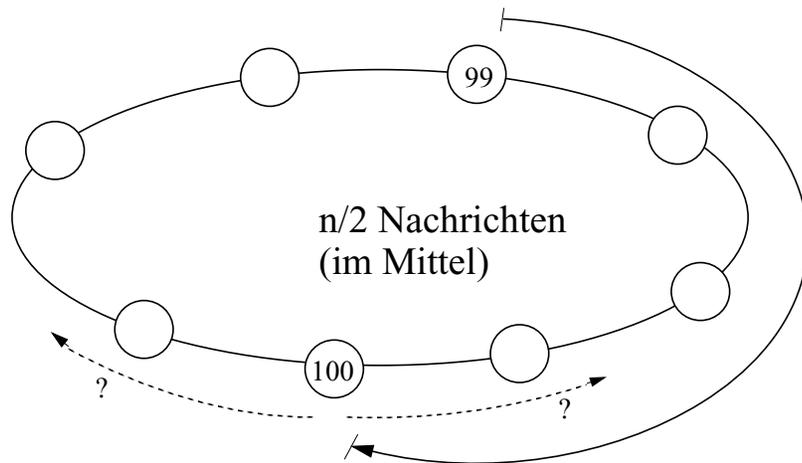
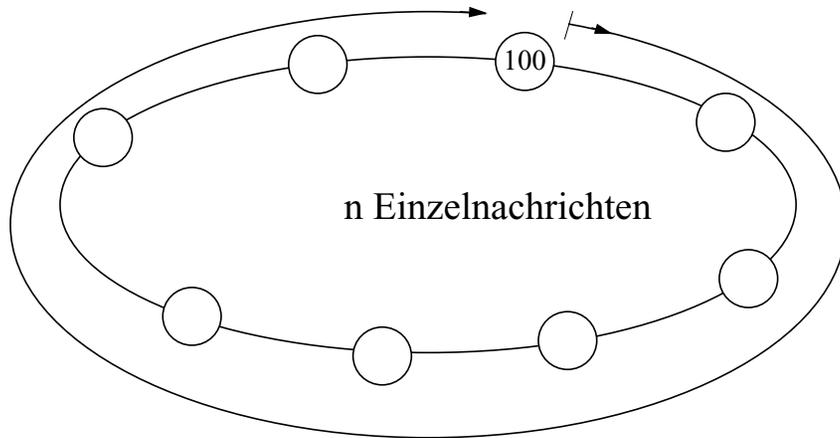


- *Worst-case Nachrichtenkomplexität* bei k Startern:
 $n + (n-1) + (n-2) + \dots + (n-k+1) = nk - k(k-1)/2$
 $\rightarrow O(n^2)$ bei Ringgröße n und $k=n$
- Wie hoch ist die *mittlere* Nachrichtenkomplexität?
 - bei "zufälliger" Permutation der Identitäten
- Wie hoch ist die *(Einheits)zeitkomplexität*?
 - wenn alle gleichzeitig starten?
 - beim zeitversetzten Starten?

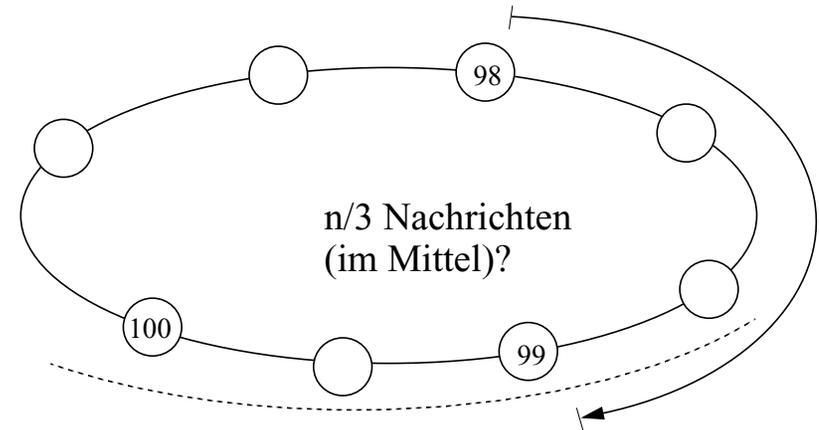
$50/6 = 8.333\dots$ Nachrichten im Mittel

Mittlere Nachrichtenkomplexität (2)

Sei $n = 100$; $id = 1, 2, \dots, 100$



Mittlere Nachrichtenkomplexität (3)



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

- Lässt sich durch explizites Nachrechnen aller ca. n^2 Fälle auch beweisen...
- Allgemeiner Beweis für die Vermutung "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?

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

Def. der sogenannten harmonischen Reihe

Was ergibt sich für n = k = 4 ?

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

Stimmt mit früherem Wert überein!
 ⇒ *Vermutung*: $n H_k$ ist korrekt

- Man kann tatsächlich exakt zeigen:

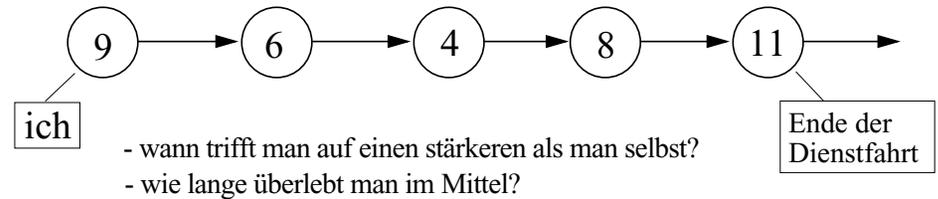
Die mittlere Nachrichtenkomplexität beträgt

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

Wir wollen dies gleich genauer herleiten...

Wartezeit bis zum ersten Rekord

- Wie weit kommt die von einem "x-beliebigen" Knoten initiierte Nachrichtenkette 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 → ...

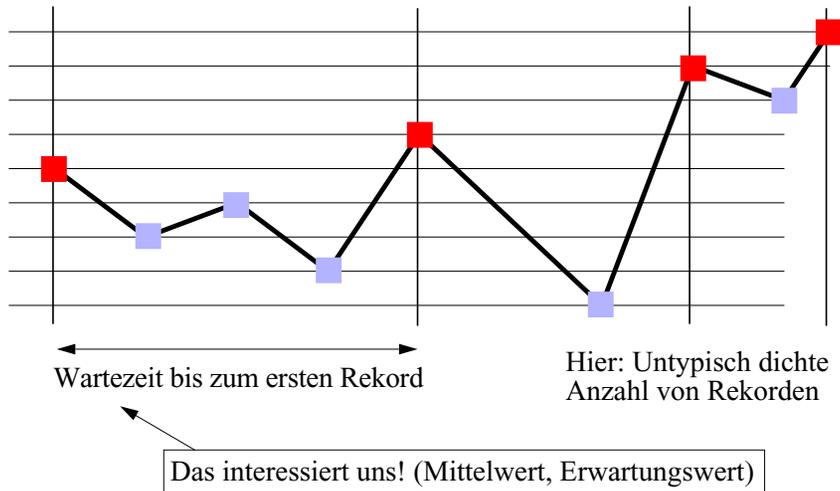
Zurückführung auf ein bek. math. Modell: *Urnenmodell ohne Zurücklegen*

- 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"



- 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 1 gdw. i-ter Wert ein Rekord

Bsp: 1001011000000100000000001000000000000001
 (Rekorde traten im 1., 4., 6., 7., 14.,... Jahr auf)

- Wieviele 1en bei gegebener Länge n?

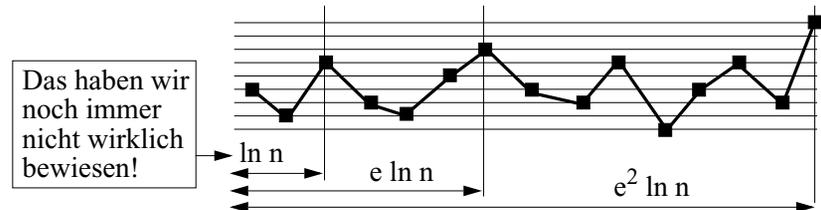
Antwort: - damit i-tes Bit auf 1 steht, muss der i-te Zahlenwert der Folge \geq alle ersten i Zahlenwerte sein
 - Wahrscheinlichkeit dafür ist $1/i$ (stimmt's? wieso?)

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

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

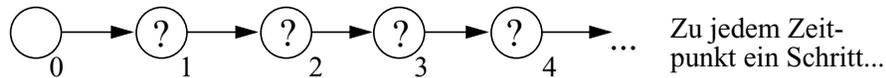
- Bsp: es gibt ca. 5 heisseste Auguste im Jahrhundert
 ... und wieviele in einem Jahrtausend?

⇒ 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



- Wahrscheinlichkeit, *genau* bis zur Position i zu gelangen (und dort besiegt zu werden)?
- $p(\text{Lebenszeit} = i) = p(\text{Lebenszeit} \geq i) - p(\text{Lebenszeit} > i)$
 - Vgl.: 18% Studis $\geq 16.$ Semester; 13% Studis $\geq 17.$ Semester \Rightarrow 5% *im* 16. Semester $> 16.$ 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 = $25/12 \rightarrow 25/3 = 8.333\dots$
Nachrichten im Mittel (vgl. früheres Ergebnis!)

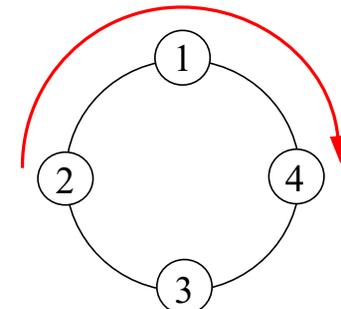
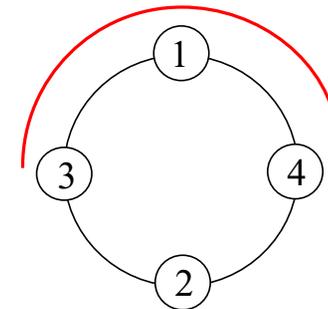
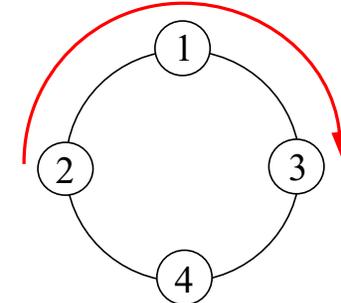
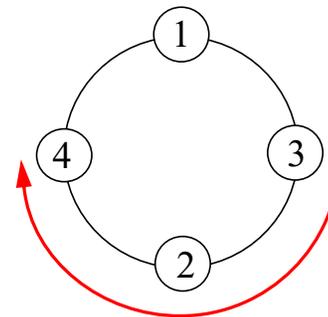
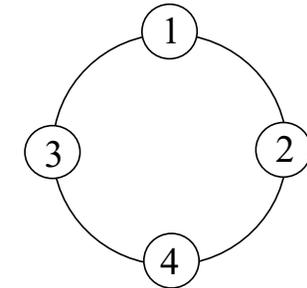
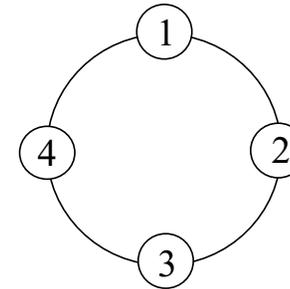
1	$\frac{1}{1 \times 2} = \frac{1}{2} = 50\%$
2	$\frac{1}{2 \times 3} = \frac{1}{6} = 16.7\%$
3	$\frac{1}{3 \times 4} = \frac{1}{12} = 8.3\%$
4	$\frac{1}{4} = 25\%$

vgl. dazu nächstes Bild

100%

Bem.: Es gilt
 $\frac{1}{n} + \sum_{i < n} \frac{1}{i(i+1)} = 1$
 (Induktionsbeweis als einfache Übung)

Lebensdauer 2 bei $n=4$



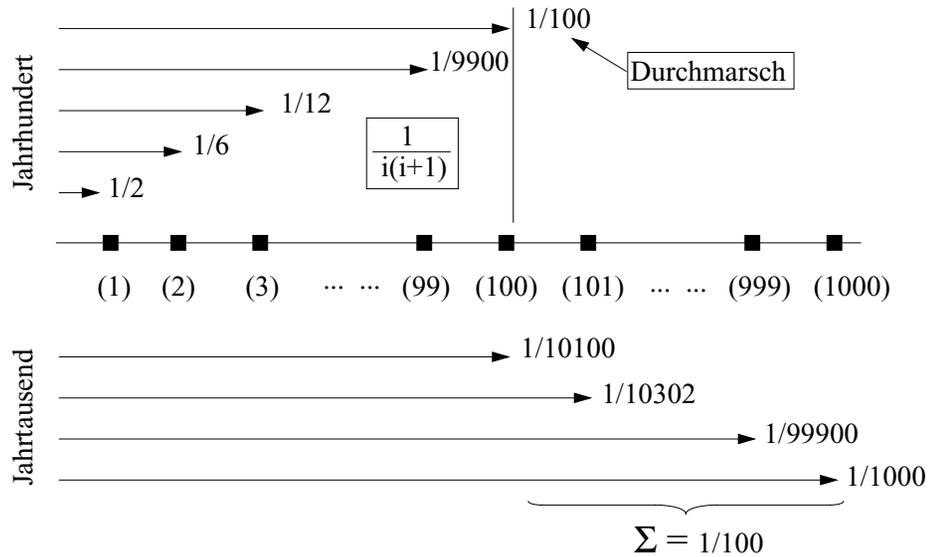
6 Bilder

jeweils 4 Starter

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

Rekord im Jahr i eines Jahrhunderts?

- Oder: Wahrscheinlichkeit, bei Position (i) "unterzugehen":



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

- Nur 50% Chance, über den ersten hinwegzukommen

- $p(\text{kein echter Rekord in einem Jahrhundert}) = 1/100$,

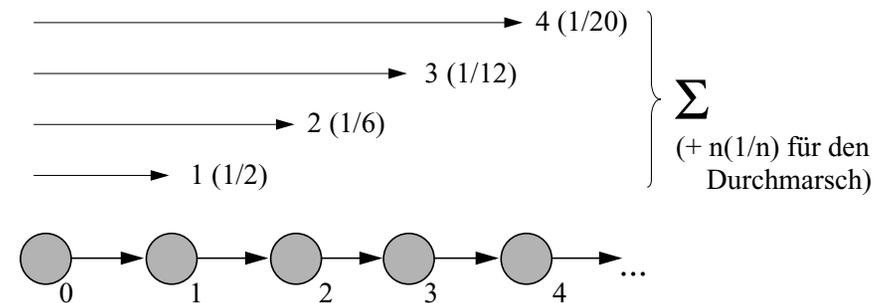
- $p(\dots \text{ in einem Jahrtausend}) = 1/1000$

Rekord würde jenseits liegen

Nun also: Wartezeit bis zum ersten Rekord

- Gesucht: Die *mittlere* Lebensdauer

- Dazu: Wahrscheinlichkeit mit Weglänge gewichten und alle Einzelfälle aufsummieren



- Daraus folgt allgemein für den Erwartungswert:

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

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, da jeder Prozess startet?
- ja; aber generell Vorsicht bei solchen Überlegungen!

Damit haben wir also nH_n als mittl. Nachrichtenkomplexität!

Qualität des Chang/Roberts-Algorithmus

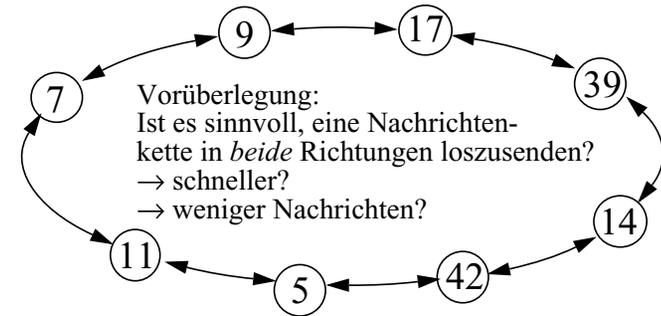
- *Satz* (ohne Bew.):

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"!

Probabilistisches Election-Verfahren für bidirektionale Ringe



- Message-extinction-Prinzip in naheliegender Weise verallgemeinern:

$I_p: \{M = 0\}$
 $M := p;$ Identität des Prozesses Uhrzeigersinn oder Gegen-Uhrzeigersinn

Wähle mit Wahrscheinlichkeit 1/2 eine Richtung;
send $\langle M \rangle$ **to** Nachbar in diese Richtung;

$R_p: \{ \text{Eine Nachricht } \langle j \rangle \text{ ist eingetroffen} \}$
if $M < j$ **then**
 $M := j;$
send $\langle M \rangle$ **to** ... /* an anderen Nachbarn */
fi /* weitersenden */
if $j = p$ **then** "I am the master" **fi**

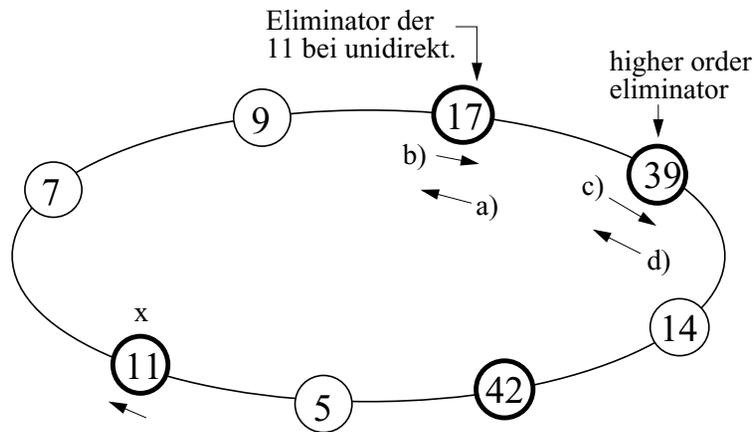
- ist Wahrscheinlichkeit 1/2 eine gute Wahl?
- geht es nicht besser deterministisch als probabilistisch?
- wie hoch ist die mittlere Nachrichtenkomplexität?

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

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

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

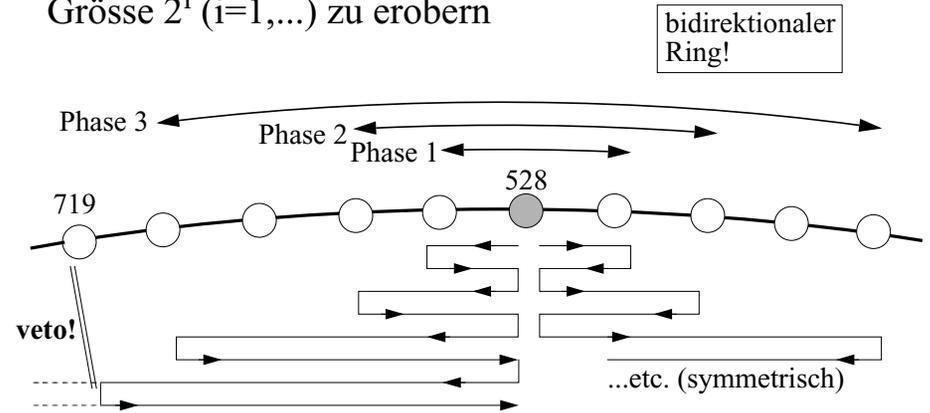
Mittlere Nachrichtenkomplexität



- 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 $n \rightarrow \infty$ 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!

Hirschberg / Sinclair-Election-Algorithmus

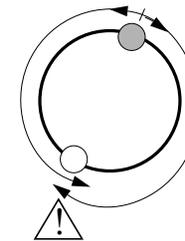
- Idee: Jeder Knoten versucht, sukzessive Gebiete der Grösse 2^i ($i=1, \dots$) zu erobern



- 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?)

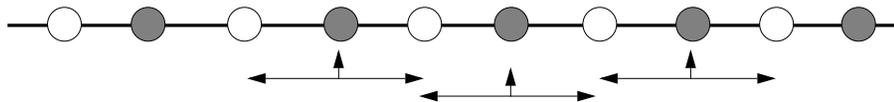
Resultat von Lavault (Beweis schwierig!):
 Asymptot. Nachrichtenkomplexität $n \rightarrow \infty$ ist
 $0.5 \sqrt{2} n \ln n \approx 0.7071 n \ln n$

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

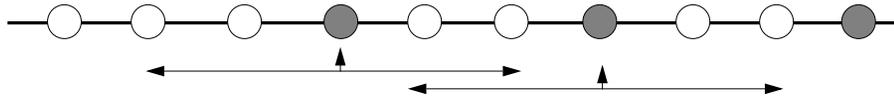
Komplexitätsanalyse

- 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

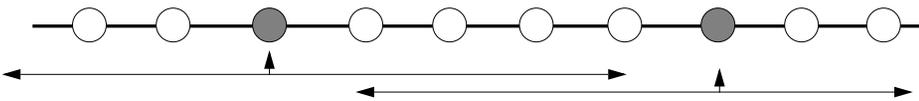
- Jeweils mind. 1 dazwischenliegender Prozess nach Phase 1:



- 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. $8 n \log_2 n \approx 5.55 n \ln n$ Nachrichten *maximal*
(Worst-case-Komplexität!)

- Aber: Wie hoch ist die *mittlere* Nachrichtenkomplexität?
- *Zeitkomplexität*: $2 + 4 + 8 + 16 + \dots + 2^i < 2^{i+1} \approx 4n$

Vergleiche dies alles mit dem Chang/Roberts-Algorithmus

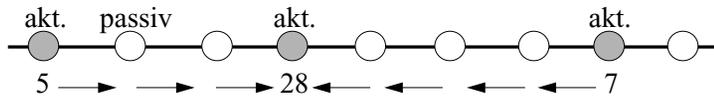
- welcher Algorithmus ist "in der Praxis" besser?

Peterson's Election-Algorithmus (1. Variante)

- Prinzip analog zum Hirschberg/Sinclair-Verfahren:
Anzahl aktiver Knoten pro "Phase" mindestens 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!

Im Unterschied zu Hirschberg / Sinclair gibt es keine Echos / Vetos!

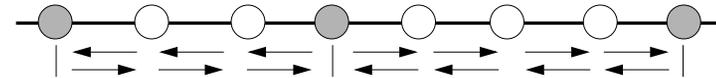
- Ein Überlebender bleibt aktiv und startet eine neue Phase: Sendet seine Identität in beide Richtungen (Initial tun das alle Initiatoren)
- *Gewonnen*, wenn die eigene Identität empfangen wird

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 - dann wäre die Nachricht "28" fälschlicherweise weitergeleitet worden!

Zeitkomplexität des Algorithmus als Denkübung (dominiert auch 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!



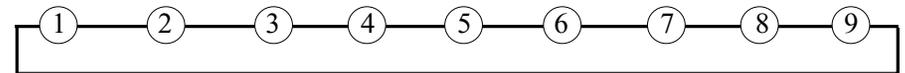
- Wenn ein Knoten Phase i überlebt, überlebt sein "linker" aktiver Nachbar diese Phase nicht!

- max. $\log_2 n$ Phasen
- max. $2 n \log_2 n$ Nachrichten

"in jeder Phase überlebt einer von *dreien*" 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



- in diesem Fall nur 2 Phasen ⇒ 4n Nachrichten! (beachte Terminierungserkennung!)

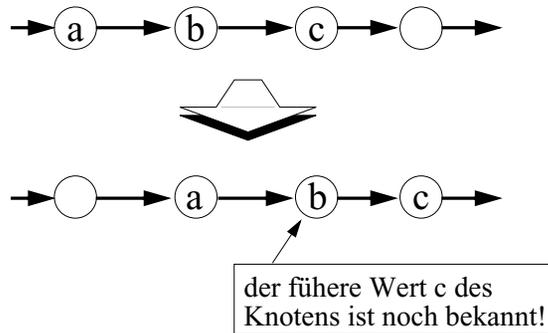
- **Mittlere Nachrichtenkomplexität:**

- ein Knoten überlebt eine Phase mit Wahrscheinlichkeit 1/3
- 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 ein 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:

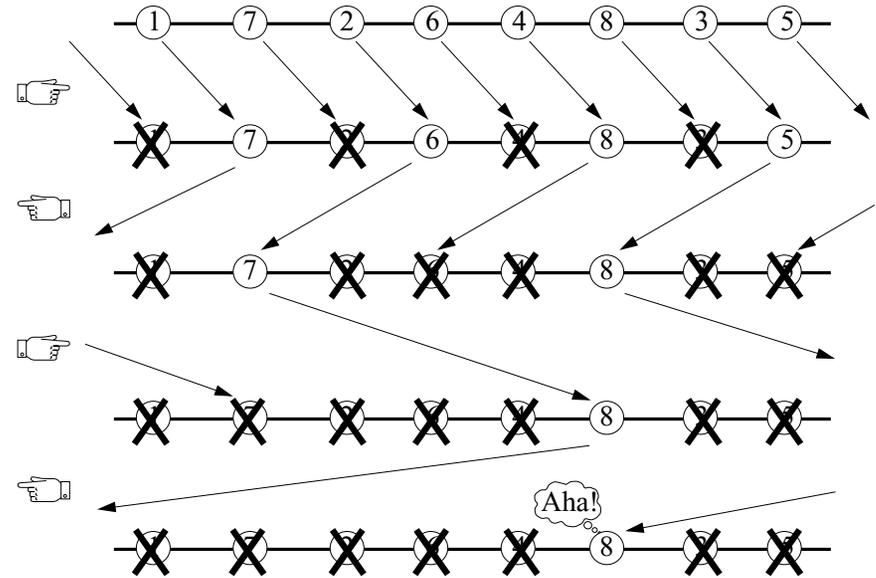


2. Nun kann (der neue) Knoten a an seinen Nachbarn b seinen Wert senden - damit kennt b sowohl a als auch c (als hätte b Nachrichten von a und c erhalten!)

Damit kostet eine solche Phase global 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")

- Denkbungen:

- 1) Wie kann man auch bei asynchronen Nachrichten und nicht gleichzeitigem Start der Knoten "eine Art" global getakteter Phasen erreichen?
- 2) Man formuliere den Algorithmus aus "Sicht eines Knotens":
Wie reagiert ein Knoten auf das Eintreffen einer bestimmten Nachricht?
- 3) Man mache sich Gedanken zur Abschätzung der worst-case und der average-case Nachrichtenkomplexität!

Nachrichtenkomplexität

$$\text{Basis } \phi = (1 + \sqrt{5})/2$$

- Behauptung:

Für die Anzahl der Phasen r gilt: $r \leq \log_{\phi} n + O(1)$

\Rightarrow Anzahl der Nachrichten $\leq 1.44 n \log_2 n + c$

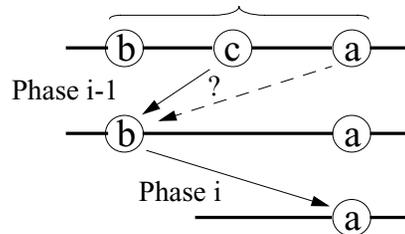
- Lemma: $a_i \leq a_{i-2} - a_{i-1}$ (für $i > 1$)

Def: Anzahl Überlebende von Phase i

Anzahl der "Opfer" von Phase $i-1$

Bew.: Betrachte 2 benachbarte Knoten a, b zu Beginn von Phase i

Gab es einen Knoten zwischen a und b in Phase $i-1$?



(1) a überlebe Phase i
 $\Rightarrow a > b$

(2) b hat Phase $i-1$ überlebt
 $\Rightarrow b > c$

(1) und (2) $\Rightarrow a \neq c$

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

\Rightarrow Für jeden Überlebenden in Phase i (hier: a) gibt es mindestens ein Opfer (hier: c) in Phase $i-1$ \square

(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 $a_i \geq a_{i+1} + a_{i+2}$

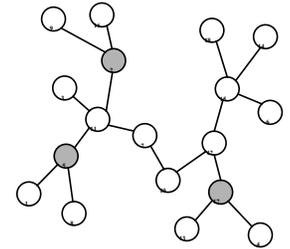
- Ferner gilt $a_{r-1} = 1 = \text{Fib}(2)$
 $a_{r-2} \geq 2 = \text{Fib}(3)$ $\left. \vphantom{\begin{matrix} a_{r-1} = 1 = \text{Fib}(2) \\ a_{r-2} \geq 2 = \text{Fib}(3) \end{matrix}} \right\} a_{r-3} \geq a_{r-2} + a_{r-1} \geq \text{Fib}(2) + \text{Fib}(3) = \text{Fib}(4)$

- Also: $n = a_0 \geq \text{Fib}(r+1)$ \rightarrow Ungleichung nach r auflösen!

Weil Fib exponentiell zur Basis ϕ wächst ($\text{Fib}(k) \approx \phi^k / \sqrt{5}$), 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 (notw.?)

- Zentrum informiert alle Knoten über Gewinner

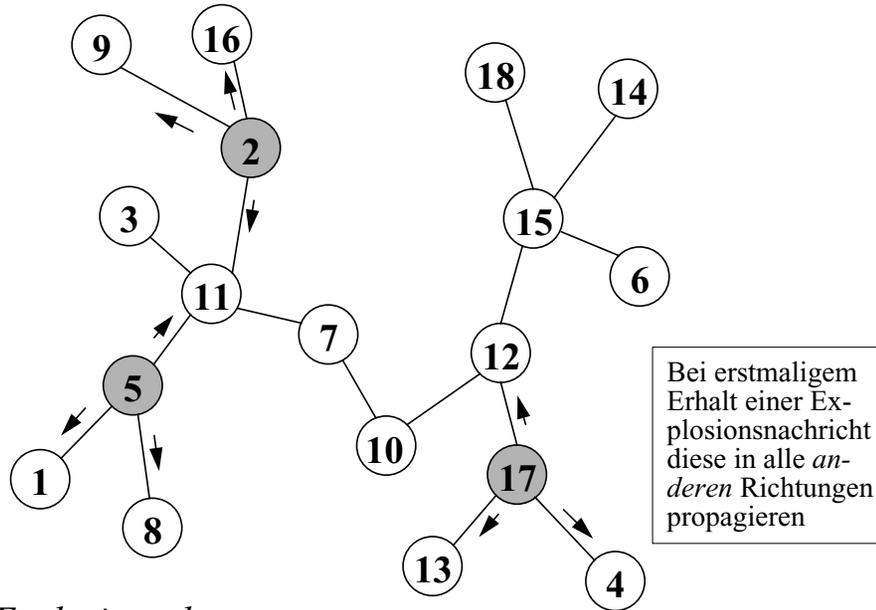
flooding!

- Explosionsphase kann an mehreren Stellen "zünden"

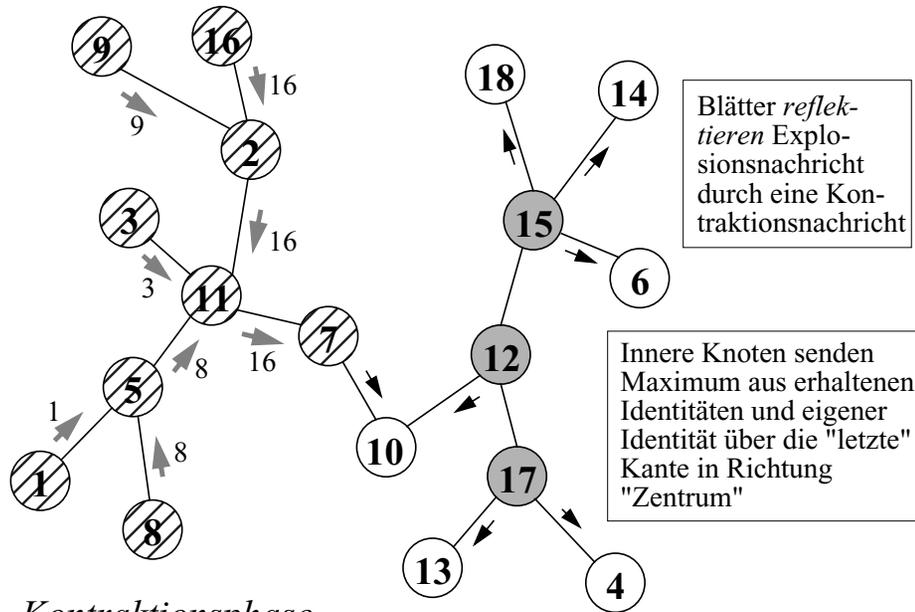
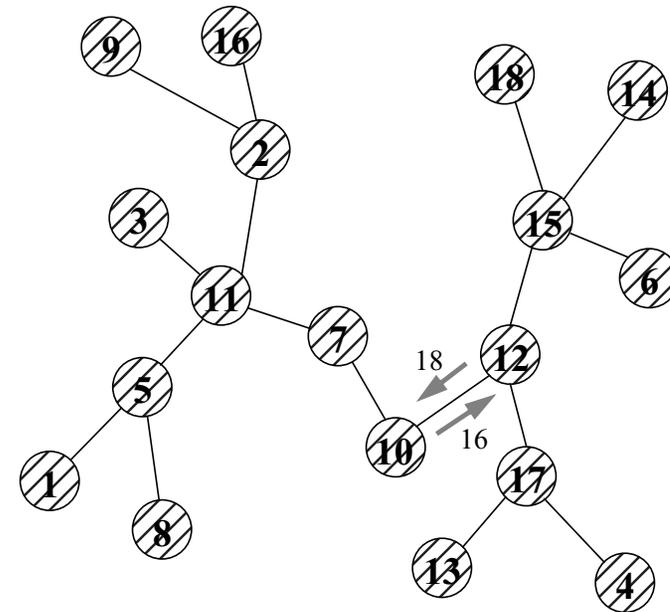
- "Vereinigung" der Explosionsphasen

- Ggf. Teile in Explosionsphase während andere Teile schon in Kontraktionsphase

- Zentrum ist nicht determiniert



Explosionsphase



Kontraktionsphase

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

Nachrichtenkomplexität

- (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 Zentrums-kante zwei
- (3) Informationsphase: $n-2$
- keine Nachricht über die Zentrums-kante

$$\Sigma = 3n + k - 4 \quad (\text{mit Information aller Knoten})$$

$$\rightarrow 3(n-1) \text{ für } k=1$$

$$\rightarrow 4(n-1) \text{ für } k=n$$

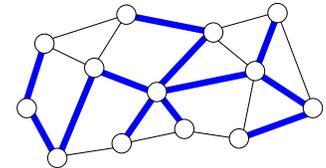
-
- Wesentlich effizienter als Ringe!
 - Wieso? (Ringe sind symmetrischer!)
 - Wieso Verfahren nicht "einfach" auf Ringe anwenden?

(eine Kante entfernen)

Election auf allgemeinen Graphen

zusammenhängend

- Wieso versagt folgende einfache Idee für allg. Graphen?
- verwende den Echo-Algorithmus, um einen (einzig) 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 den 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:* Chang / Roberts-Algorithmus (also "message extinction"), jedoch *Echo-Algorithmus* anstelle des zugrundeliegenden Ring-Verfahrens
- Also:
 - Jeder Initiator startet "seinen" Echo-Algorithmus
 - Explorer und Echos führen die Identität des Initiators mit
 - Schwächere Nachrichten (Explorer und 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 → kein anderer Initiator bekommt jemals ein Echo)

- *Veranschaulichung:* "Gleichzeitiges" Einfärben des Graphen mit verschieden dominanten Farben
- Vermutung bzgl. der worst-case und der average-case Nachrichtenkomplexität?

Nachrichtenkomplexität: untere Schranke

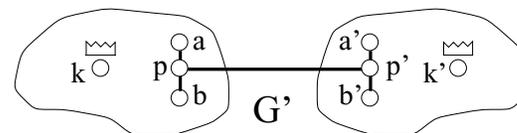
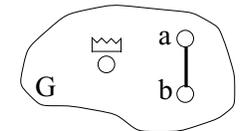
Satz: Die Nachrichtenkomplexität für das Election-Problem in allg. Graphen beträgt mindestens $\Omega(e + n \log n)$
 was dominiert typischerweise?

Beweis: $n \log n$ ist untere Grenze, da der Satz auch für Ringe gilt (vgl. dortigen Satz ohne Beweis)

Also noch zu zeigen: e ist untere Schranke

Widerspruchsbeweis: Angenommen, es gäbe einen Election-Algorithmus A, der weniger als e Nachrichten für einen Graphen G benötigt ⇒ es gibt eine Kante \overline{ab} , über die *keine* Nachricht fließt

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



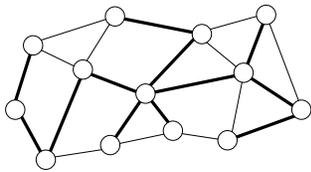
Da zwischen den beiden Teilen keine Nachricht ausgetauscht wird, gewinnen u.U. zwei Prozesse k, k'!

Beachte bei diesem Beweis:

- bei Anwendung von Algorithmus A auf G' kann sich jeder Knoten genauso wie der entsprechende im Graphen G verhalten
- die Knoten haben (weder in G noch in G') ein "globales Wissen": sie kennen nicht die Struktur oder Gesamtgröße des Graphen, sie kennen nicht die Identitäten ihrer Nachbarn
- die Knoten wissen insbesondere nicht, ob Situation G oder G' vorliegt
- Knoten p und p' seien keine Initiatoren
- bzgl. der Knotenidentitäten wird nur vorausgesetzt, dass auf diesen eine lineare Ordnung definiert ist (nicht, dass es sich um Nummern handelt)

Verteilte Spannbaumkonstruktion?

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



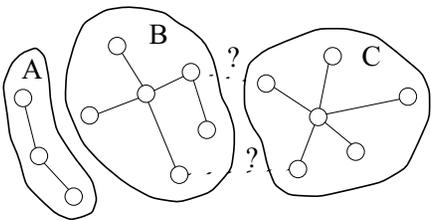
- z.B. um entlang dieses Baumes Information zu verteilen und einzusammeln
- Routing ist wesentlich einfacher, wenn Zyklen ausgeschlossen sind

Problem: - Wer initiiert die Spannbaumkonstruktion?

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

- **Lösung 1:** Election, Gewinner startet Echo-Algorithmus
(bzw. Variante Echo-Election: Die Echos bilden 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 (GHS) 1983 mit $2e + 5n \log n$ Nachrichten (im Detail nicht ganz einfach, hier nicht weiter behandelt)
- Problem: Sparsam mit Nachrichten umgehen!

Election und Spannbaumkonstruktion

Beh.: Election und Spannbaumkonstruktion sind in allgemeinen Netzen 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 : $C_t \leq C_e + 2e$ (wg. obiger "Lösung 1")
- (b) Es gilt für C_e : $C_e \leq C_t + O(n)$ (wg. Kplx. 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 $\Omega(e + n \log n)$ für die Nachrichtenkomplexität des Election-Problems folgt aus (b):

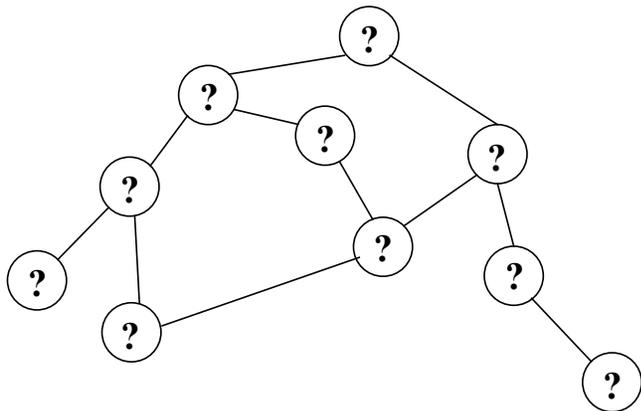
Das Spannbaum-Problem hat eine Nachrichtenkomplexität von mindestens $\Omega(e + n \log n)$

Denn sonst: Konstruiere den Spannbaum effizienter und löse mit zusätzlichen $O(n)$ Nachrichten das Election-Problem!

→ Der genannte GHS-Algorithmus ist grössenordnungsmässig optimal!

Anonyme Netze

- Keine Knotenidentitäten
 - bzw.: alle (oder mehrere) *identische* Identitäten
 - bzw.: ggf. vorhandene Knotenidentitäten werden nicht benutzt
- Frage: Was geht dann noch? (Insbes. Symmetriebrechung!?)



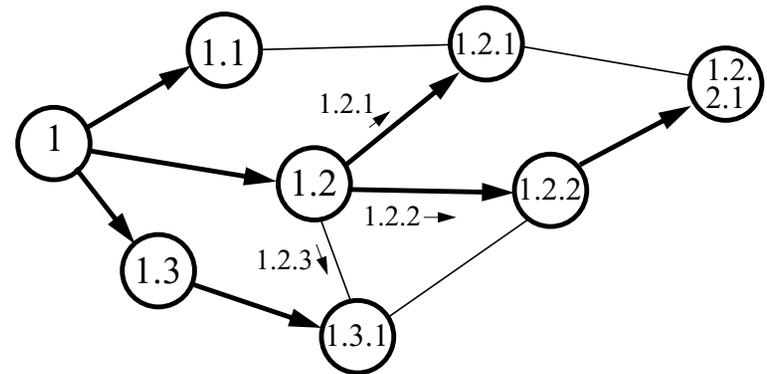
- Es gilt: Falls Election in anonymen Netzen geht, dann können die Knoten individuelle Namen bekommen

↑ ...und *alles* geht wie gehabt!

- Der Satz lässt vermuten, dass Election in anonymen Netzen *nicht* geht, sonst wäre Anonymität kein *grundsätzliches* Problem!

De-Anonymisierung

Z.B.: Der Leader flutet das Netz mit verschieden benannten Nachrichten und 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
 ⇒ Alle Nachrichten und damit alle Knoten heissen verschieden!



- Einziger (!) Initiator gibt sich selbst einen 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

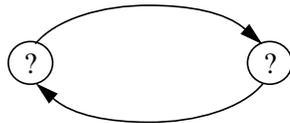
Fragen: - wann ist das Verfahren beendet?
 - wie erfährt der Initiator dies?

Election in anonymen Netzen?

- In anonymen Netzen geht manches (z.B. Election?) nicht mehr mit deterministischen Algorithmen
→ randomisierte Verfahren helfen gelegentlich!
- Manches geht noch, wenn wenigstens die Knotenzahl bekannt ist (aber auch das hilft nicht immer!)

Satz: *Es gibt keinen stets terminierenden Election-Algorithmus für anonymen 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 unterwegs sind
 - Anfangskonfiguration des Algorithmus ist *symmetrisch*: $(z, z, \emptyset, \emptyset)$, wegen Anonymität
 - Alle lokalen Algorithmen sind definitionsgemäss identisch --> Knoten können sich jeweils gleich (quasi "zum gleichen Zeitpunkt") verhalten
 - Konfigurationsfolge kann daher aus lauter symmetrischen Konfigurationen bestehen: $(z, z, \emptyset, \emptyset) \rightarrow (z', z', k, k) \rightarrow \dots \rightarrow \text{etc.}$
 - Falls die Folge endlich ist, könnte der Endzustand symmetrisch sein --> keine Symmetriebrechung möglich!

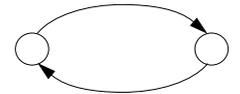
Ein probabilistischer Election-Algorithmus

```

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

- hier: *anonyme* Ringe der Größe 2

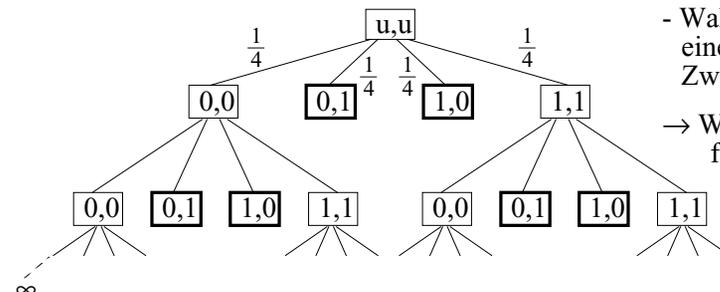


- jeder Knoten führt den gleichen nebenstehenden Algorithmus aus
- Verallgemeinerung auf grössere Ringe?

- Geht es nicht auch mit einem *deterministischen* Algorithmus?

- Beachte: "*korrekt*" \neq "*korrekt mit Wahrscheinlichkeit 1*"!

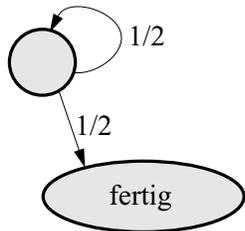
- "*korrekt*" heisst: Algorithmus *hält stets* und liefert richtiges Ergebnis
- obiger Algorithmus ist aber (in diesem Sinne) *nicht korrekt*, da es nicht terminierende Konfigurationsfolgen wie etwa
 - $(0,0) \rightarrow (0,0) \rightarrow (0,0) \rightarrow (0,0) \rightarrow (0,0) \rightarrow (0,0) \dots$ oder
 - $(0,0) \rightarrow (1,1) \rightarrow (0,0) \rightarrow (1,1) \rightarrow (0,0) \rightarrow (1,1) \dots$ etc. gibt!
- alle diese Folgen haben aber die Dichte 0 (wenn "random" gut genug ist...)
 - \Rightarrow Algorithmus terminiert mit Wahrscheinlichkeit 1
 - \Rightarrow Algorithmus ist "*korrekt mit Wahrscheinlichkeit 1*"!



- Wahrscheinlichkeit eines unendlichen Zweiges = 0
- \rightarrow Wahrscheinlichkeit für leader = 1

Mittlere Nachrichtenkomplexität?

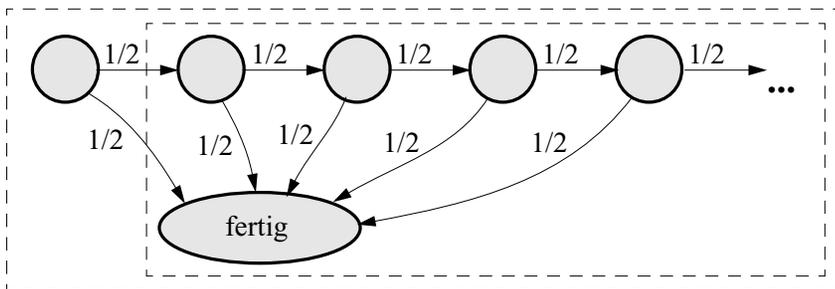
- Beachte: Keine Schranke für *worst-case* Komplexität!
- Die mittlere ("erwartete") Nachrichtenkomplexität lässt sich mittels *Markow-Ketten* leicht ermitteln



- 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?

$$1 \frac{1}{2} + 2 \frac{1}{4} + 3 \frac{1}{8} + 4 \frac{1}{16} + \dots = ?$$

- 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
- Bem.: Hinweise zur Varianz etc. findet man in entsprechenden Mathematik-Lehrbüchern

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* $\epsilon = 1-p > 0$, 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 $\epsilon=0$): ein solcher Monte Carlo-Algorithmus wäre *total korrekt* (hält stets und das Ergebnis ist dabei ("mit Wahrscheinlichkeit 1") korrekt)!

Las Vegas-Election-Algorithmus für anonyme Ringe bekannter Grösse

- 1981 von Itai/Rodeh: Basiert auf Chang/Roberts-Verfahren

- Prinzip:

oder ein anderer Wert, z.B.
 $2n$, n^2 oder einfach 2 ?

- wähle eigene Identität $id = \text{random}(1, \dots, n)$, mit $n = \text{Ringgrösse}$
- 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...

- Berechnungsdauer 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:

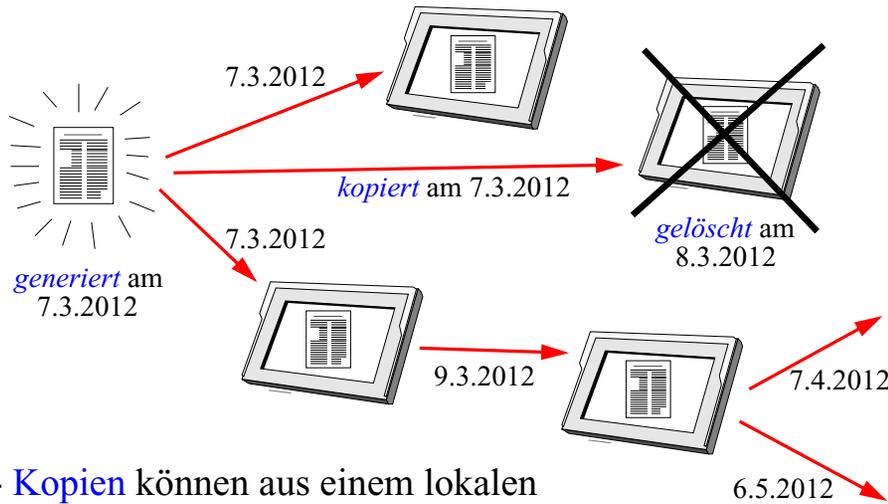
- wenn die durch Echos gemeldete Baumgrösse $\neq n$ ist, neue Runde starten etc.

Kausaltreues Beobachten

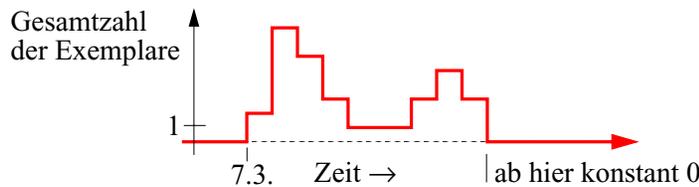


Aussterben einer Menge von Kopien

- Ein **Beispiel**: on-line Kopien einer Zeitung:



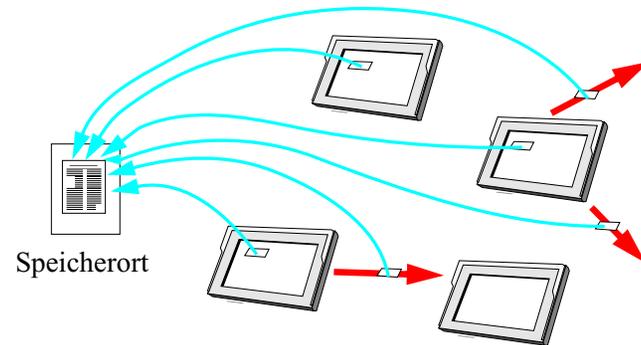
- **Kopien** können aus einem lokalen Exemplar erzeugt und verteilt werden
- Exemplare werden gelegentlich **gelöscht**



- Interessante Frage (erst *nach* der Erzeugung sinnvoll):
Ist die Zeitung schon "ausgestorben"?
- Präziser: Ist die Gesamtzahl der Exemplare = 0?

Beobachten des Referenzzählers?

- Bei Hochgeschwindigkeitsnetzen könnte man "copy by reference" statt "copy by value" verwenden
- Also: Zeitungsexemplare als Nur-lese-Kopien halten und lediglich eine **Referenz** auf den Speicherort übergeben
 - *kopiert* wird also nur ein kurzer *Verweis*, z.B. `nptp://faz.ffm.de/07_03_12`
 - im WWW also eine URL
 - bei Bedarf könnte eine echte Kopie angefordert werden ("copy by need")



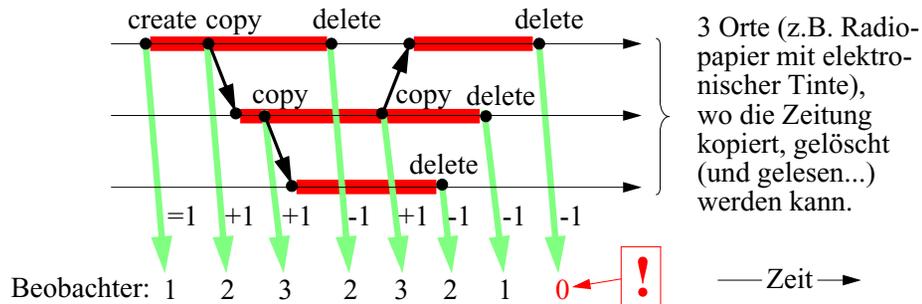
- **Copy** → Übermitteln der Referenz (=Zugriffspfad / Adresse)
- **Delete** → Löschen der Referenz

- "**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!

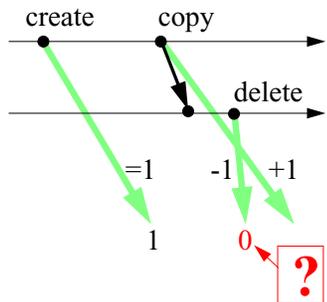
→ **Garbage-Collection**-Problem in verteilten Systemen!

Zählen der Exemplare?

- Lösungsansatz: **Beobachter** wird informiert über
 - einmaligen Erzeugungsvorgang ("create-Ereignis")
 - jeden Kopiervorgang ("copy")
 - jeden Löschvorgang ("delete")



- Aber Achtung: **Beobachtung** ist u.U. **nicht kausaltreu!**

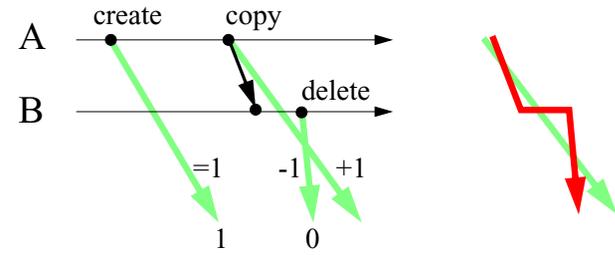


- Beachte: **delete**-Ereignis ist eine **kausale Konsequenz** des **copy**-Ereignisses ("ohne copy auch kein delete")
- Beobachter sieht jedoch die Konsequenz *vor* ihrer Ursache!

- 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

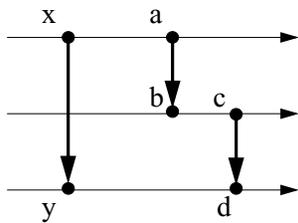
- **Nicht-kausaltreue Beobachtungen** sind die Ursache für viele konzeptuelle Probleme verteilter Systeme! **!**
- z.B. verteilte Terminierung, Schnappschuss, Deadlockerkennung...

- **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



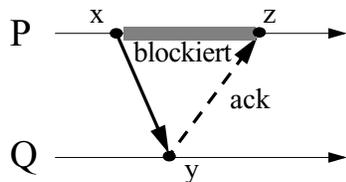
Falls Nachrichtenpfeile *senkrecht* sind (Nachrichten also keine Übertragungszeit benötigen), kann es keine direkten oder indirekten Überholungen geben: jede *später* ausgesandte Nachricht (die den Anfang eines Überholpfades bilden könnte), kommt auch *später* an

$$t(y) = t(x) < t(a) = t(b) < \dots < t(c) = t(d) \Rightarrow t(y) < t(d)$$

Informell: "Blitzschnelle" Nachrichten kann man nicht überholen

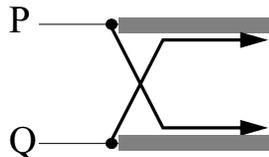
2. Idee: Sender so lange blockieren, bis der Empfänger ein *Acknowledgement* zurückgesandt hat

- 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



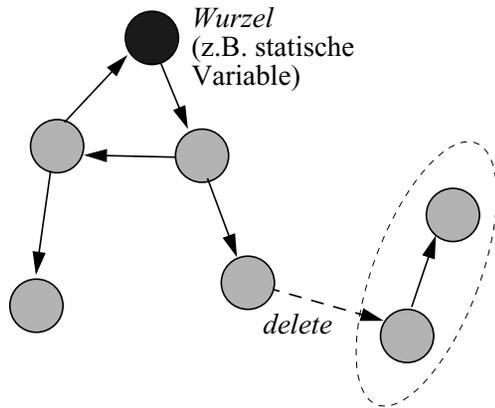
Andere, bessere Idee zur Vermeidung indirekter Überholungen?

Garbage-Collecton in verteilten Systemen



Garbage-Collection

- "Verpointerte Objekte"



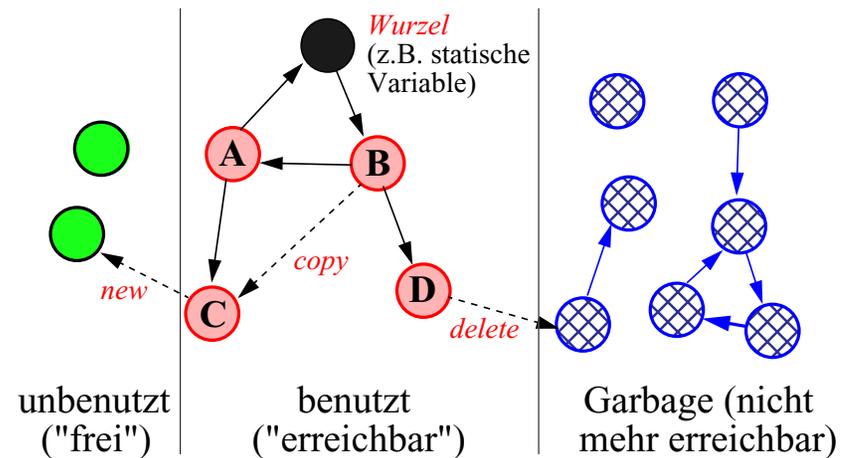
Diese beiden Objekte sind nicht mehr von der Wurzel aus erreichbar und werden zu "garbage"

Ein Garbage-collector soll solche Objekte identifizieren und deren Speicher wiederverwenden

- Wenn man diese Objekte als "aktiv" ansieht, hat man schon "fast" ein paralleles / verteiltes System
 - Vgl. Puppentheater: Auch wenn eine einzige Person die Figuren im Zeitmultiplex bedient, kann man dies als ein paralleles System autonomer Objekte betrachten
 - 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", unbenutztem Speicher
- Bei Sprachen mit dynamischem Speicher und Zeigerstrukturen
 - historisch: LISP (bereits in den frühen 1960er Jahren)
 - Interesse heute: objektorientierte Sprachen (+ ggf. parallele Implementierung)
 - statische Variablen und Variablen im Laufzeitkeller ("Stack") stets erreichbar, dynamische Variablen auf der Halde ("Heap") u.U. jedoch "abgehängt"



- copy: Füge Referenz zwischen 2 erreichbaren (!) Objekten hinzu
z.B.: B.refvar := A.refvar

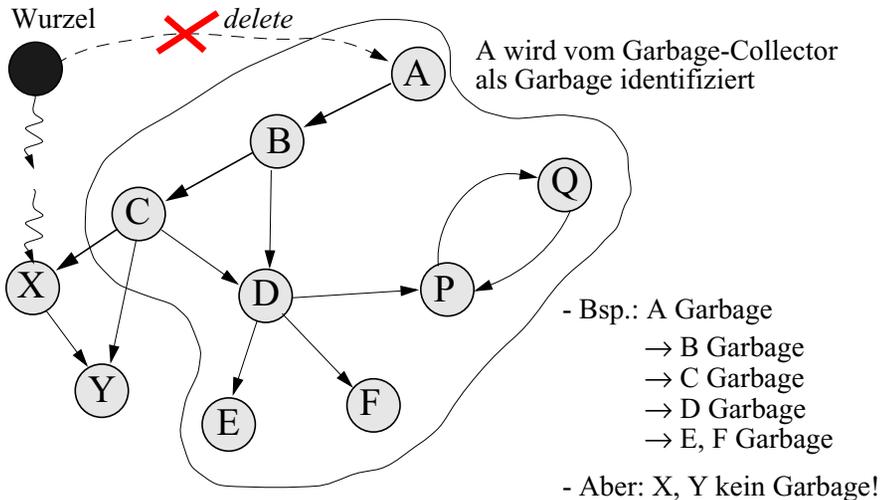
- Objekt erreichbar $\Leftrightarrow \exists$ Pfad von der Wurzel dorthin
- "Garbage sein" ist stabiles Prädikat (vgl. Terminierung!)
- Mutator \leftrightarrow Collector spielen mit-/gegeneinander

Anwendungsprogramm
(manipuliert Zeiger zwischen Objekten mittels copy, delete und new)

Kontrollprogramm
identifiziert Garbage

Rekursives Freigeben

- Falls ein Objekt als Garbage erkannt wird:
 - sollten seine ausgehenden Referenzen gelöscht werden,
 - damit kann ggf. eine grössere daran "aufgehängte" Substruktur als Garbage erkannt werden!
 - rekursives Erkennen von Garbage-Objekten, "pointer chasing"



- P, Q sind dann auch Garbage, manche Garbage-Collectoren können solchen "zyklischen Garbage" jedoch nicht erkennen!

Freezing: "Stop the World"-Prinzip

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
- 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-Entdeckungsverfahren:

- friere alle Prozesse 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!

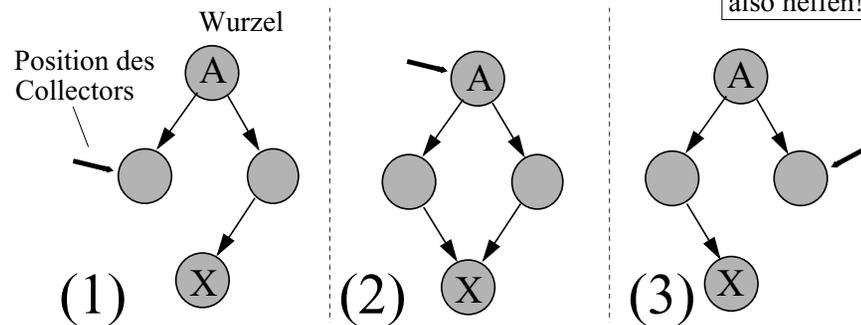
- "in aller Ruhe" ein Objekt nach dem anderen betrachten

"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 lange Wartezeiten der Anwendung

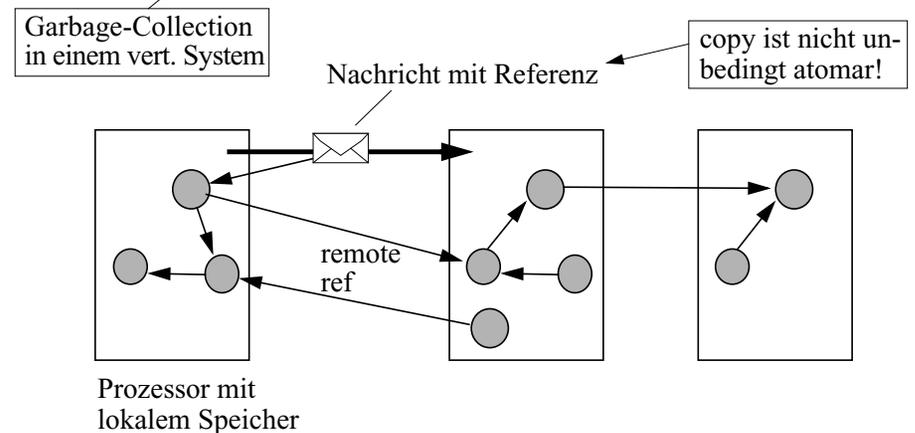
Traversiert den Graphen und markiert erreichbare Objekte
 Collector kann von gleichzeitig aktivem Mutator getäuscht werden → Kooperation notwendig!



- Knoten X wird als nicht erreichbar angesehen...
- ...obwohl es immer 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



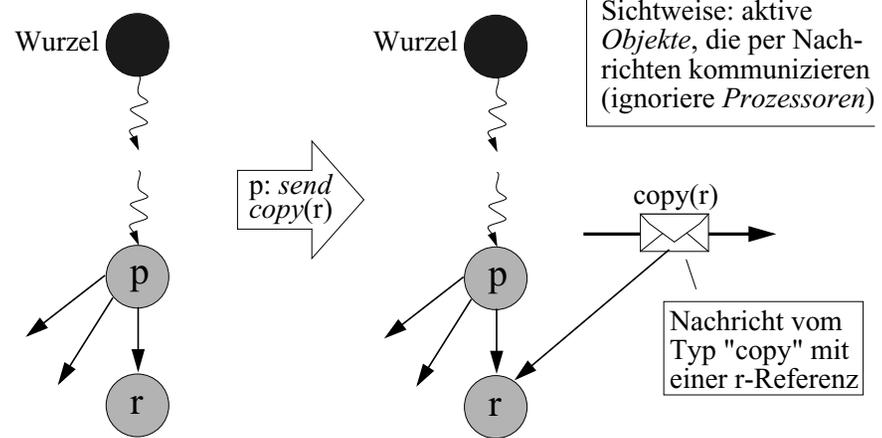
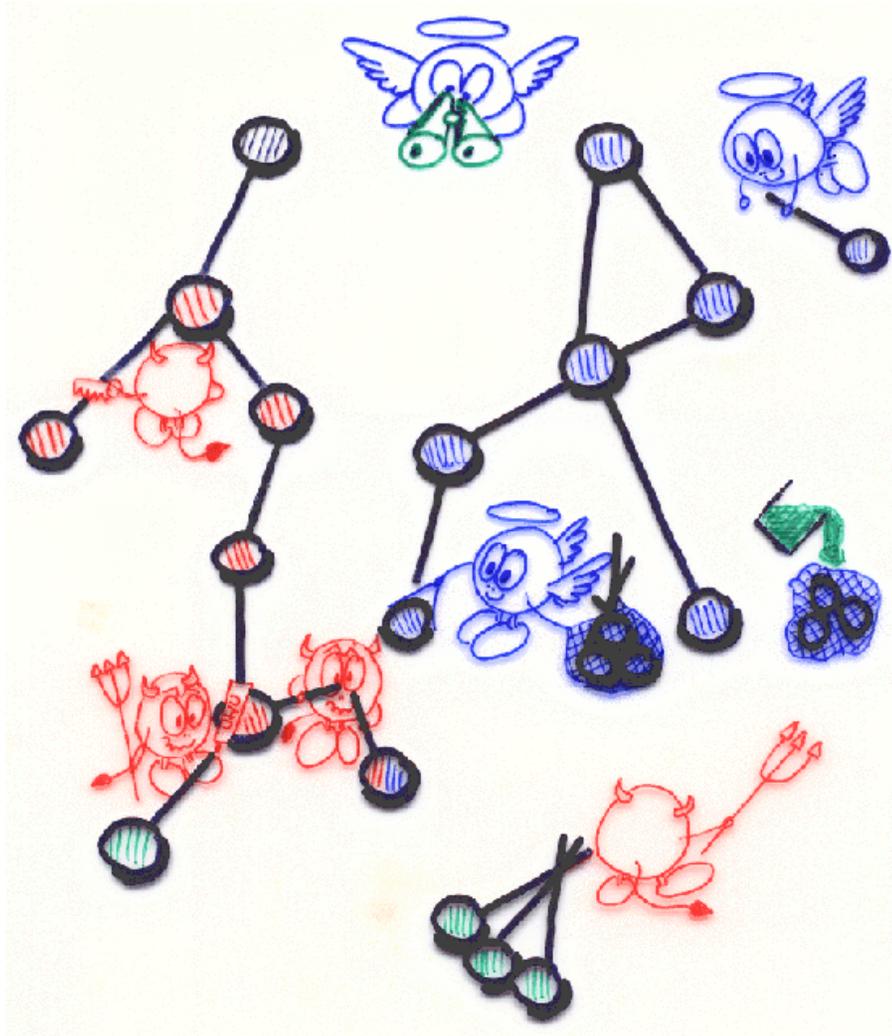
- (1) Prozessorüberschreitende Referenzen ("remote ref")
- (2) Nachrichten enthalten (Objekte mit) Referenzen auf andere Objekte ("remote copy")
 - Referenzen in Nachrichten dürfen nicht übersehen werden!

- 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
- Viele Mutator / Collector (pro Prozessor einen?)
 - Synchronisation zwischen den vielen Prozessen notwendig

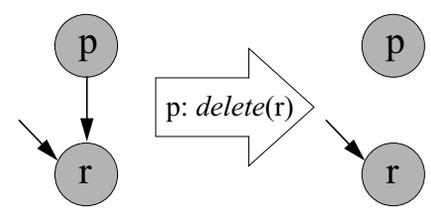
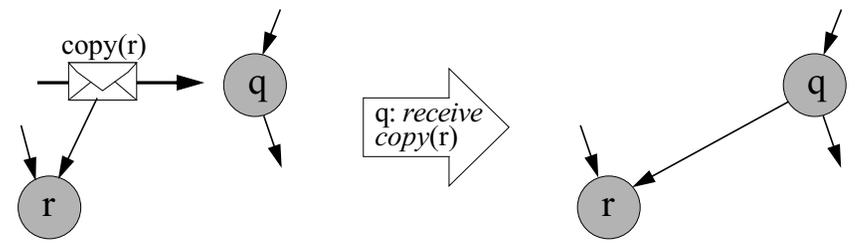
- Pragmatisches:

- "Stop the world" ist hier schon gar nicht angemessen
- Kontrollkommunikation minimieren (Kosten, Effizienz)

Wirkung der Mutator-Operationen



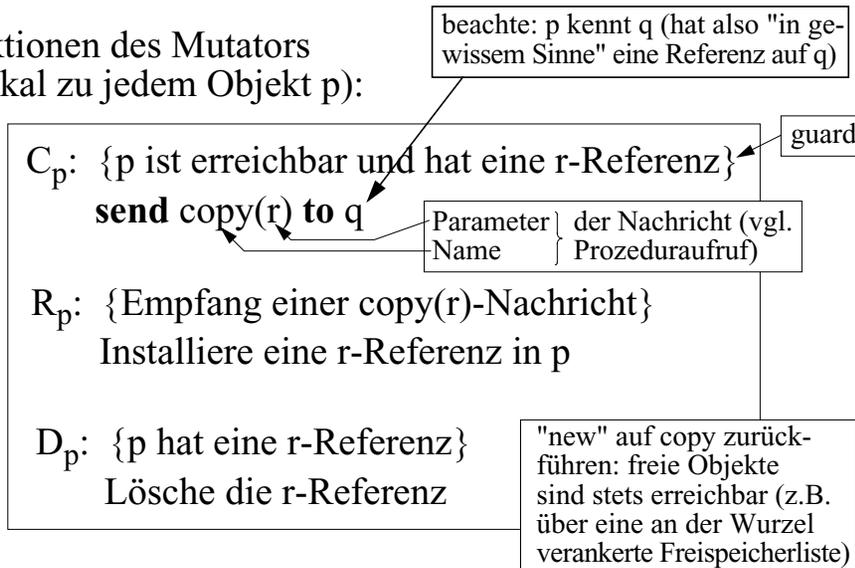
Beachte: Copy-Operation ist nicht atomar \Rightarrow
Aufspalten in zwei Aktionen *send* / *receive copy*



- "new" hier nicht relevant
- jede Aktion ändert den globalen Graphen "etwas"
- Folge solcher Änderungen \rightarrow "Berechnung"
- hier: "Interleaving-Modell": Operationen sind atomar ("zeitlos") \rightarrow es gibt keine gleichzeitige Aktionen \rightarrow verschränkte Ausführung

Formalisierung des verteilten Garbage-Collection-Problems

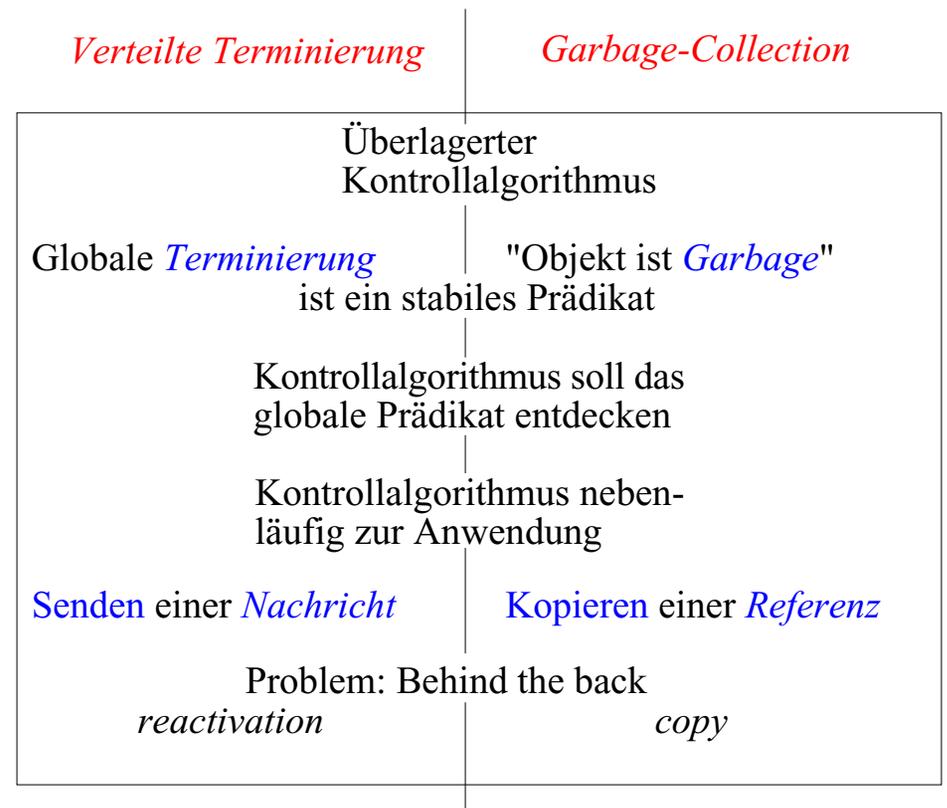
Aktionen des Mutators
(lokal zu jedem Objekt p):



- So "sieht" der Collector die Basisberechnung
- Vergleiche dies mit Basisaktionen beim Problem der verteilten Terminierung!
- Aufgabe: Überlagerung mit Aktionen des Collectors:
 - zusätzliche atomare Aktionen
 - Ergänzung der 3 Aktionen mit weiteren Statements, um die notwendige Kooperation mit dem Collector zu erreichen
- 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:

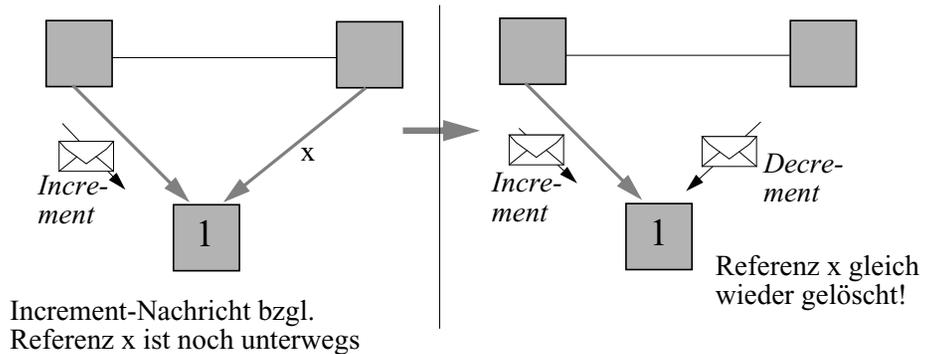


- Können Lösungen des einen Bereiches auf den anderen Problembereich angewendet werden?

Referenzzähler-Verfahren

- Idee: Jedes Objekt weiss, wie oft es referenziert wird
 - wird dieser Referenzzähler 0 → Garbage
- Nachteil: Zyklischer Garbage wird nicht entdeckt
- Zugehörigen Referenzzähler "atomar" zusammen mit der copy- oder delete-Operation aktualisieren
 - relativ einfach in einem nicht-verteilten System
- In einem verteilten System:
 - increment / decrement* Nachrichten

Kopieren kann zu *Fehlinterpretationen* führen!



Decrement schneller als vorangehendes Increment
 ⇒ Referenzzähler wird 0, jedoch kein Garbage!

Korrektheitsbedingung:

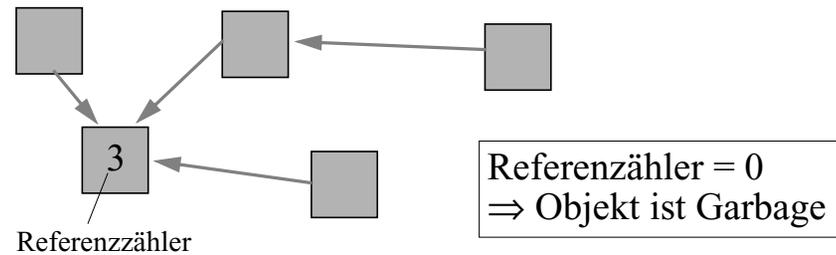
Empfang von *Inc* früher als alle kausal abhängigen *Dec*

Wie dies garantieren?

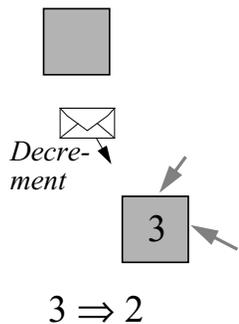
- 1) Synchroner Kommunikation (→ "atomare" Operation)
- 2) Acknowledge von Inc-Nachrichten (+ warten)
- 3) "Causal Order" realisieren...

Objekt hat eine *kausaltreue Sicht* der Ereignisse (die es betreffen)

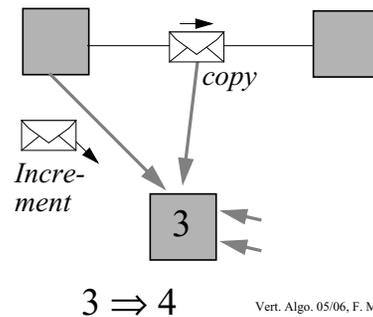
→ (indirekte) Überholungen vermeiden



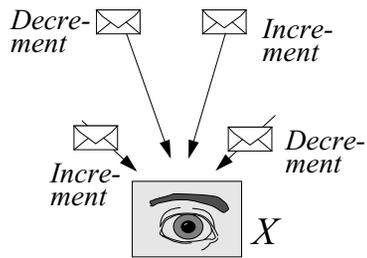
Löschen einer Referenz:



Kopieren einer Referenz:



Garbage-Identifikation als konsistentes Beobachtungsproblem

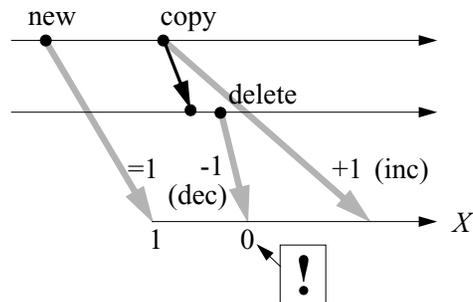


Objekt X **“beobachtet”** die copy- und delete-Operationen, die es selbst betreffen

- die **increment-** und **decrement-** Nachrichten dienen der Beobachtung
- damit feststellen, ob *alle* copy durch delete kompensiert wurden

- Diese Beobachtung sollte **kausal** sein!

- zumindest darf ein dec nicht vor “seinem” inc empfangen werden



- **Jedes** Objekt beobachtet **jedes andere**

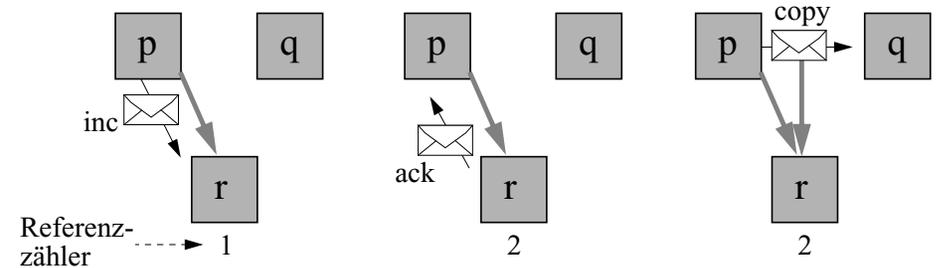
- “Causal Order” (Globalisierung von FIFO) bzgl. Kommunikation gefordert

- **Wie** realisiert man kausaltreue Beobachtungen?

- in diesem **konkreten Fall** diverse Möglichkeiten (→ **viele Algorithmen!**), z.B. copy solange verzögern, bis Bestätigung für inc-Nachricht eingetroffen
- oder: kausaltreue Beobachter / "Causal Order"-Kommunikation **allgemein** implementieren?

Verteiltes Reference-Counting: Lösungen

→ Jede increment-Nachricht bestätigen (warten auf ack bevor das copy losgeschickt wird):



- Korrektheitskriterium erfüllt:

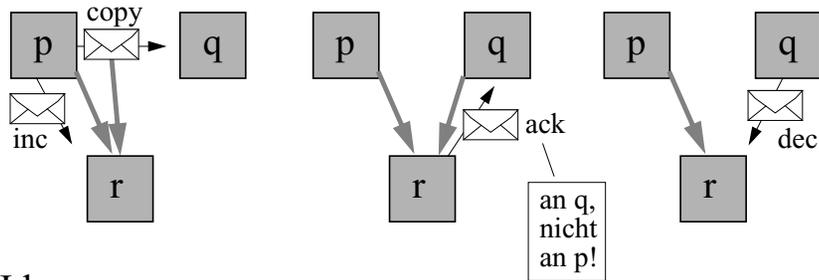
inc wird vor einem kausal abhängigen dec empfangen (d.h. Objekt r erfährt über die Existenz einer Kopie einer r-Referenz, bevor es vom Löschen dieser oder einer "solchen" Referenz erfährt)

- 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 eine zugehöriges ack (bzgl. inc) vom Zielobjekt der Referenz empfangen hat

⇒ Korrektheitskriterium erfüllt

- Beachte: Referenz kann stets gelöscht werden, nur das Senden von dec muss verzögert werden

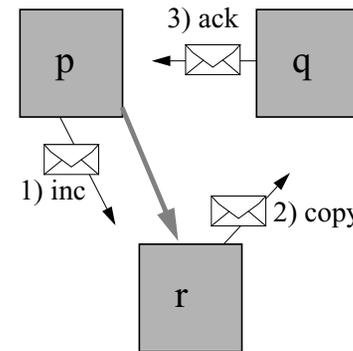
- Implementierungsskizze:

- 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!
 - Abschwächung $|ACK|>0 \wedge |COPY|>0$ möglich? Konsequenzen?

- Vorteil: kein Verzögern von Basisaktionen wieso?
- Nachteil: Ack-Nachricht und FIFO-Kanäle notwendig

Varianten von Rudalics

1) 3-Nachrichten-Protokoll ("zyklisch"):



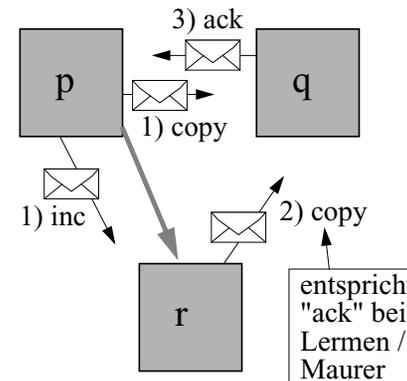
Idee: q bekommt Referenz auf r von r selbst; nachdem r seinen Zähler inkrementiert hat (veranlasst durch inc).

Bedingung: p darf seine r-Referenz erst löschen, wenn alle erwarteten ack-Nachrichten eingetroffen sind.

Frage: Wäre es auch möglich, dass das ack an p von r (statt q) gesendet wird?

- Vorteil: kein FIFO notwendig (falls FIFO garantiert ist: ack-Nachricht einsparen ⇒ nur zwei Nachrichten pro copy!)
- Nachteil: Kopieren dauert länger (2 Nachrichten)

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

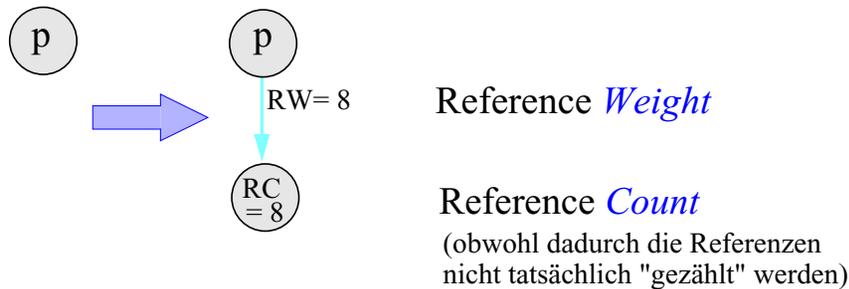
- Unter welchen Bedingungen dürfen p bzw. q dec-Nachrichten senden?

- Vorteil: kein FIFO notwendig; keine Verzögerung
- Nachteil: 4 Nachrichten (aber nur 3 "sequentiell")

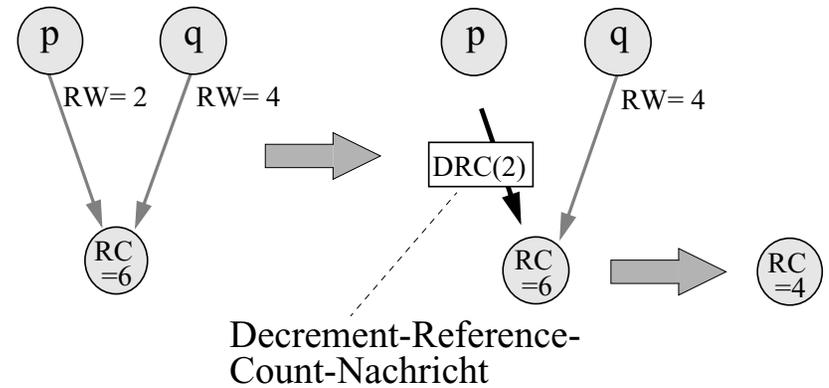
Die Referenzgewichts-Methode

(WRC: "Weighted Reference Counting")

Neues Objekt generieren ("new"):



Referenz löschen ("delete"):

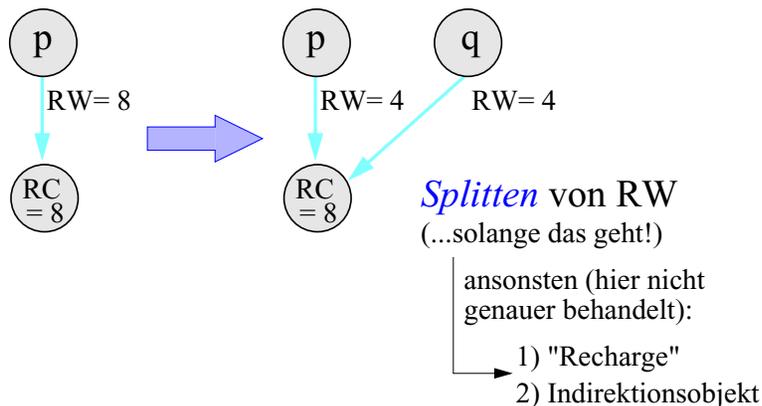


Invariante: $RC = \sum RW + \sum DRC$

$RC = 0 \rightarrow$ Objekt ist Garbage

\rightarrow alle Referenzen dieses Objektes auf andere Objekte löschen (DRC-Nachrichten senden)

Referenz kopieren ("copy"):



- Logarithmische Kompression (2er-Potenzen!) von RW

- mit nur 2 Bit pro Zeiger lassen sich so RW bis max. 8 darstellen
- statt 8 kann ggf. auch ein (etwas?) grösserer Maximalwert gewählt werden
- RC so nicht komprimierbar \Rightarrow int-Variable mit "vielen" Bits pro Objekt

- Keine Verzögerung bei copy / delete und bei copy keine zusätzlichen Nachrichten!

- RW = 1 sollte ein eher seltenes Ereignis sein (\rightarrow Zusatzaufwand)

- Analogie zur Kredit-Methode bei vert. Terminierung!

- Beachte: Es wird *keine Increment-Nachricht* benötigt!

Kredit-Methode und WRC

Garbage-Collection und Terminierung

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: $\sum \text{Kredit} = 1$

Invariante: $RC = \sum RW + \sum DRC$

- Gesamtkredit beim
Urprozess = $2^0 = 1$

- $RC = 0$ bei R

Terminierung

R ist Garbage

Also: Kreditmethode entspricht WRC-Garbage-Collection!

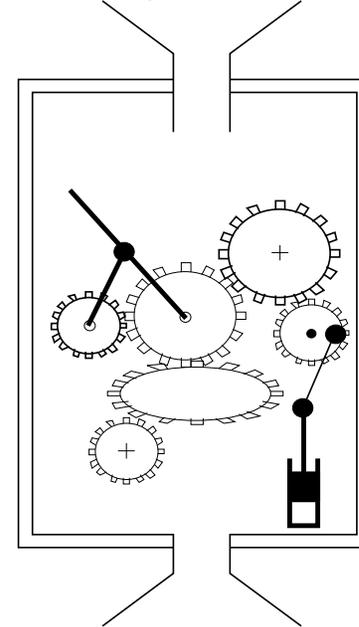
nicht notwendigerweise verteilte

Theorem:

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

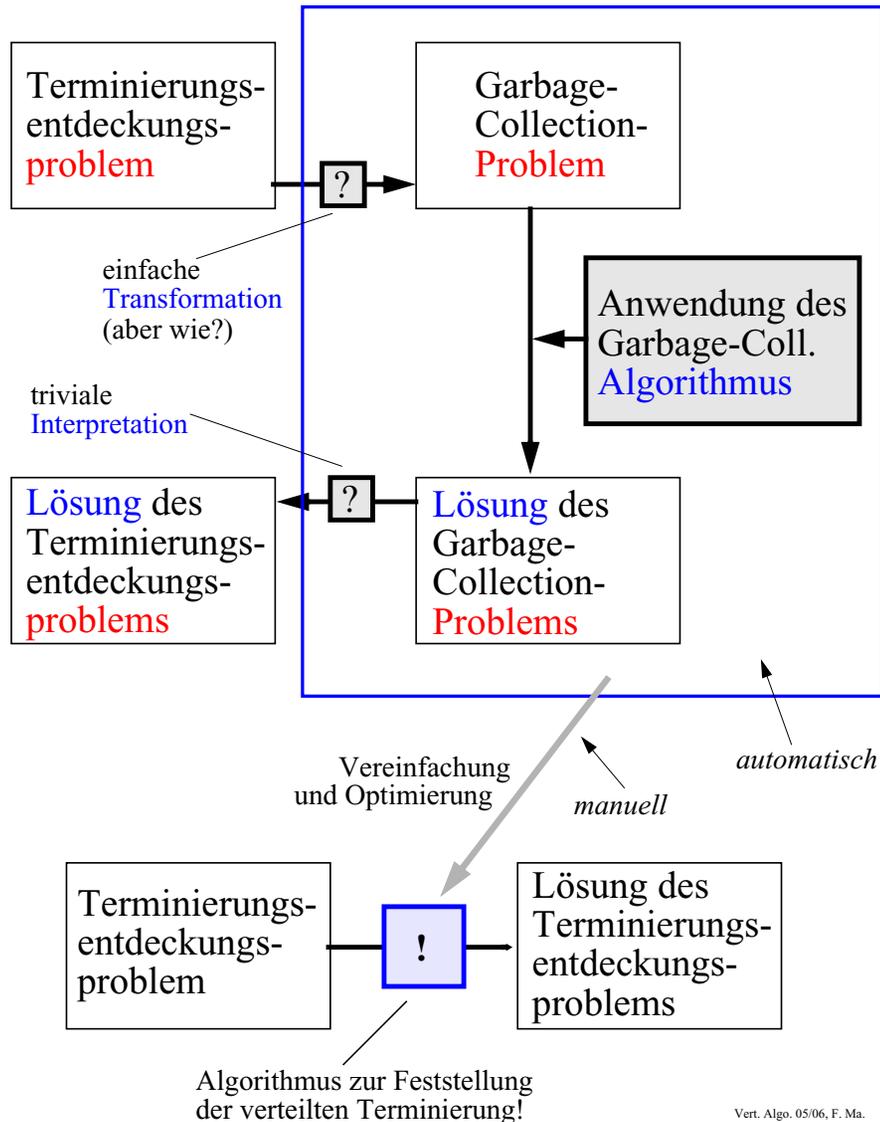


mechanische
Transformation

Ein *verteilter Termini-
erungsentdeckungs*-Algorithmus

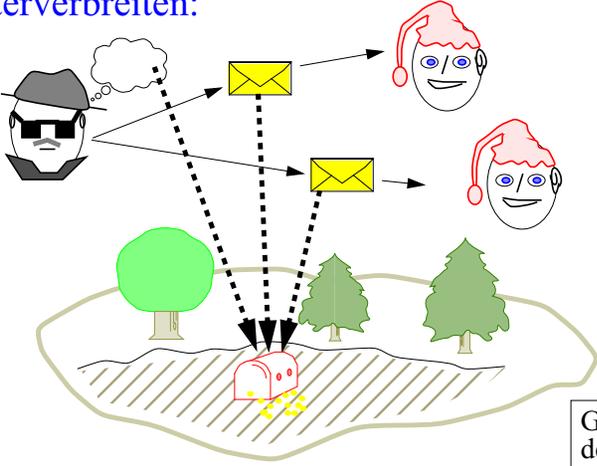
Problemtransformation

- Es wird das *Problem*, nicht der *Algorithmus* transformiert!



Vom Ende einer Geheimniskrämerei

Die vier goldenen Regeln der Geheimniskrämerei:

- 1) Es gibt **Geheimnisträger**  und **uneingeweihte 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**

Geheimnis: Ort des Schatzes

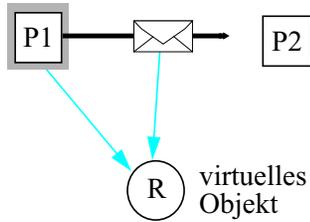
es gibt keine Geheimnisträger und keine Nachrichten mit dem Geheimnis

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** \Leftrightarrow P besitzt **Referenz auf R**
- 2) Jede **Nachrichte** enthält **Referenz auf R**

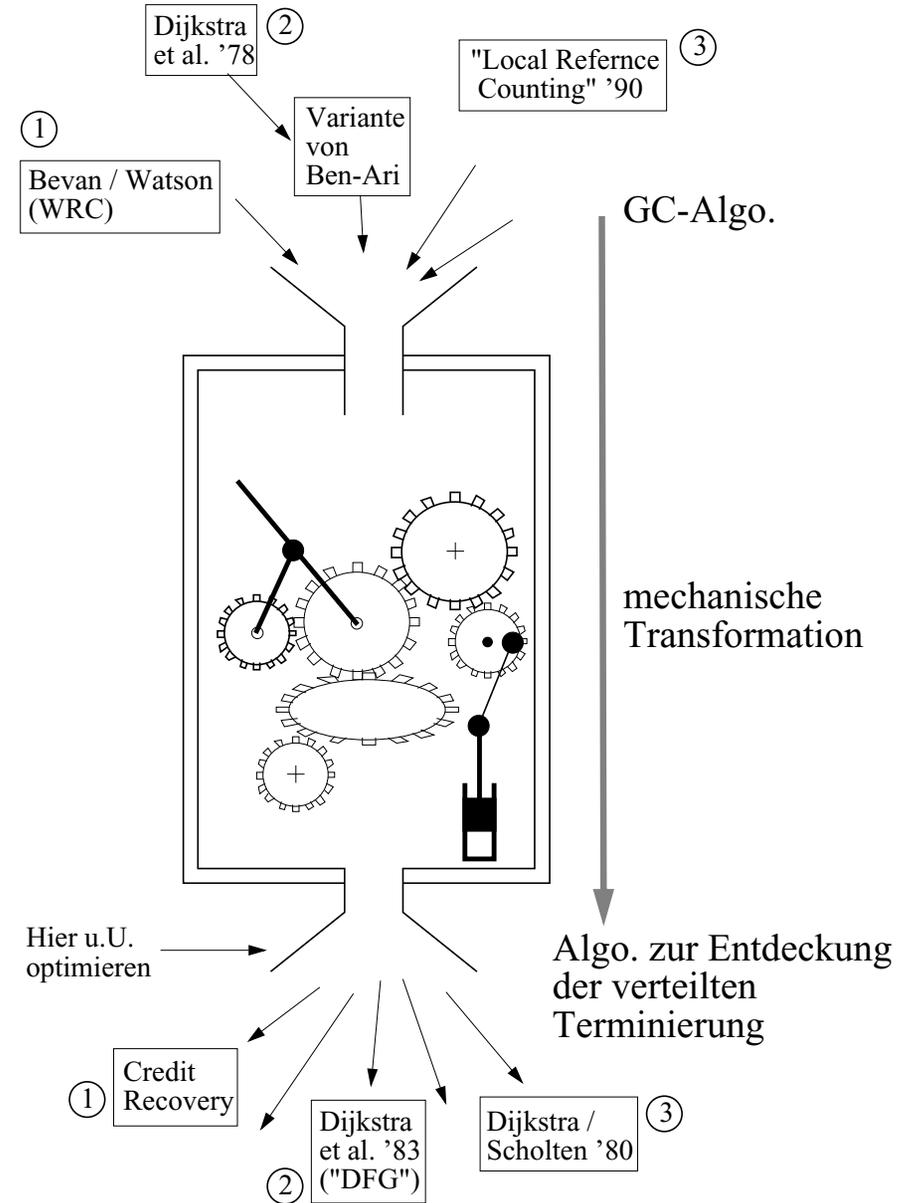
- Die beiden Regeln lassen sich ("induktiv") erfüllen:

- 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 \Leftrightarrow Es gibt keine Referenz auf R
 \Leftrightarrow Alle Prozesse passiv und keine Nachricht unterwegs \Leftrightarrow **Verteilte Berechnung terminiert**

- Also: verwende *irgendeinen* **GC-Algorithmus**
 - \rightarrow interpretiere Berechnung als GC-Problem
 - \rightarrow melde Terminierung, wenn R als Garbage erkannt

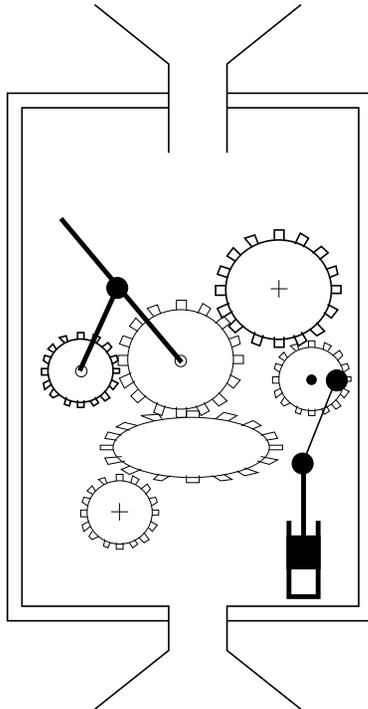
- **Übung:** man mache dies für konkrete GC-Algorithmen aus der Literatur
- beachte: es entstehen keine Zyklen von Referenzen, daher sind auch Referenzzählverfahren anwendbar!



Bekannte Garbage-Collection-Verfahren werden so in bekannte und brauchbare Algorithmen zur Erkennung der verteilten Terminierung transformiert!

Die Patent-Story

WRC-Garbage-Collection-Algorithmus 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

① Publication number: **0 225 755
A2**

⑫ EUROPEAN PATENT APPLICATION

⑲ Application number: 86309082.5

⑤ Int. Cl.: **G 06 F 12/02**

⑳ Date of filing: 20.11.86

20.11.86

ICL

③① 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 or-
ganised 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 refer-
ence 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 exist-
ing pointers to it, and hence that cell may be reclaimed (gar-
bage-collected) for re-use.

nächste Folie →

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.

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 1 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 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.
5. A method according to claim 1 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.

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*.

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 (Jan. 2000: ein Jahr zuvor: 93 of 2461228)

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

...

man hätte besser auch nach "garbage collector" etc. gefragt

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. Viieldiskutiert 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 bystander! 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 challenged. In fact 70 to 80% patents don't hold up when challenged."

If you need more information, please contact Arvind at
arvind@au-bon-pain.lcs.mit.edu

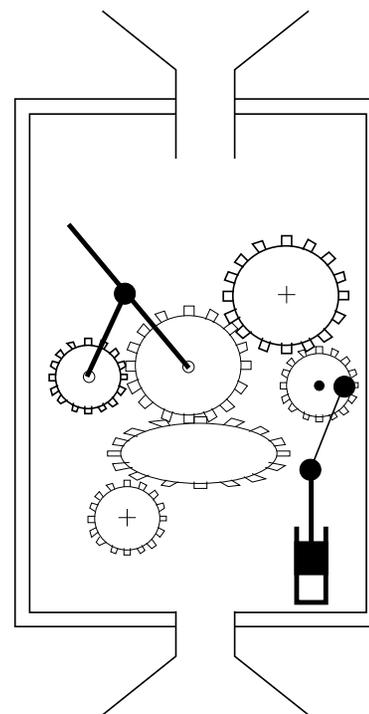
Dale Grit,
Computer Science Dept.,
Colorado State University
on leave at LLNL
grit@lll-crg.llnl.gov

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

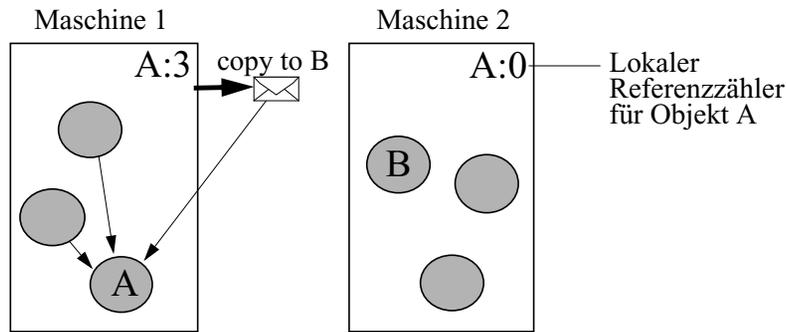
J. Piquer: *Indirect Reference Counting*, Proc. PARLE 91, 150-165



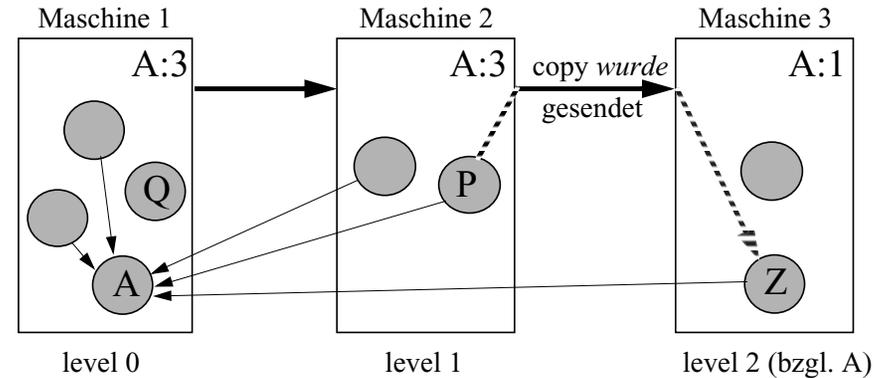
E.W. Dijkstra, C.S. Scholten: Termination Detection for Diffusing Computations. Inf. Proc. Lett. 11 (1980), 1-4

LRC-Garbage-Collection-Verfahren

- Pro *Maschine* wird (potentiell) für jedes Objekt ein Referenzzähler gehalten, z.B. für Objekt A:

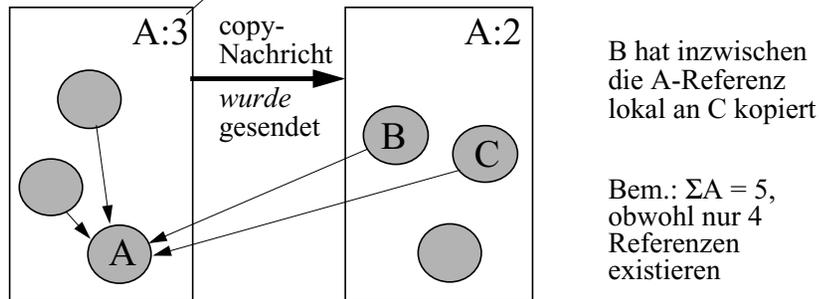


LRC: Weitervererben von Referenzen



- Maschine 1 muss nicht informiert werden, wenn z.B. Objekt P eine (Kopie seiner) A-Referenz an Z schickt!

hier wird der lokal erzeugte und dann versendete Zeiger mitgerechnet



 Dec-Nachricht senden, wenn der lokale Referenzzähler bei Erhalt einer dec-Nachricht dekrementiert werden kann

Beachte: Es wird angenommen, dass der lokale Referenzzähler *atomar* mit den Operationen copy, delete, Empfang einer dec-Nachricht aktualisiert werden kann

- Korrektheit (safety):

Referenzbaum

Lokaler Referenzzähler auf level $i+1 > 0$
 \rightarrow lokaler Referenzzähler auf level $i > 0$

- wieso eigentlich Baum?

\Rightarrow Falls eine Referenz existiert, dann ist der Referenzzähler auf level $0 > 0$

- wie genau ist "level" definiert?

\Rightarrow "Garbage-Kriterium": Referenzzähler auf level $0 = 0$

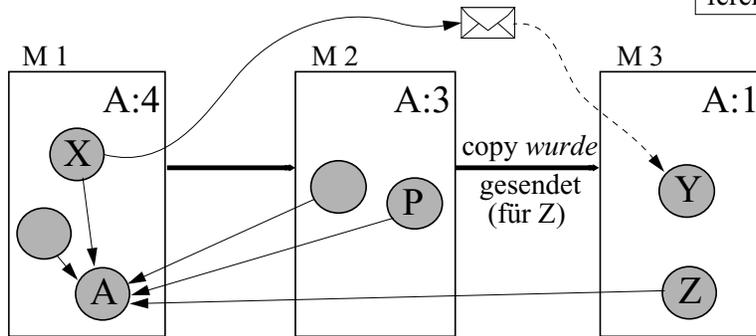
- Denkübingen: wie steht es mit der *liveness*?
- was geschieht, wenn P eine A-Referenz an Q (Maschine 1) sendet? (\rightarrow Zyklen ?)

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



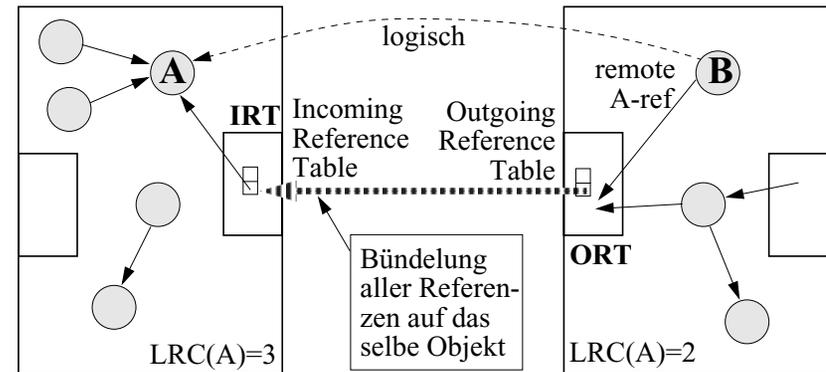
- Aber: Wann und an wen (hier: Maschine 1 und/oder 2) soll ggf. eine dec-Nachricht von Maschine 3 gesendet werden?

- ?
- (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?)

Implementierungssicht: Remote-reference-Tabellen



- ORT: enthält Schattenobjekte als lokale Stellvertreter ("Proxy") für alle entfernten Objekte

- beachte: hier ist $\sum LRC(A) = 5 = \text{Anzahl A-Referenzen (inkl. bei IRT!)}$,

- 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 ggf. *mehrfach indirekt*

- u.U mehrfache Adressumsetzung bei *Zugriff* über eine Remote-Referenz, falls der Zugriff tatsächlich entlang der Referenzkette erfolgt

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 Referenzbaumes)
- kein Verzögern bei delete
- FIFO nicht notwendig
- Gesamtzahl der Nachrichten: genausoviele 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.a. 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

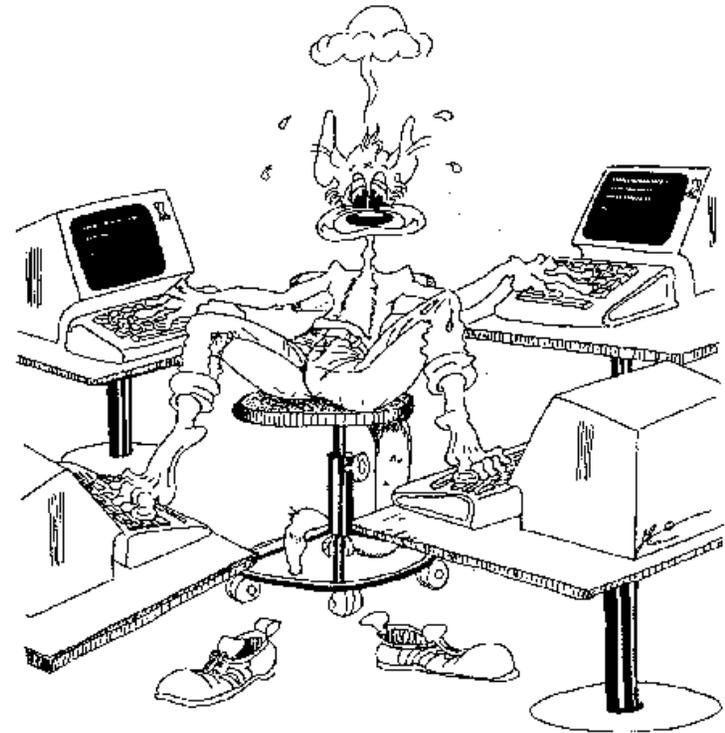


Bild: R. G. Herrtwich, G. Hommel

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.a. mehrere verschiedene Berechnungen zu einem verteilten Algorithmus! (Nichtdeterminismus)

- Anweisungen, Statements \Rightarrow 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.a. mehrere "äquivalente" Zeitdiagramme!

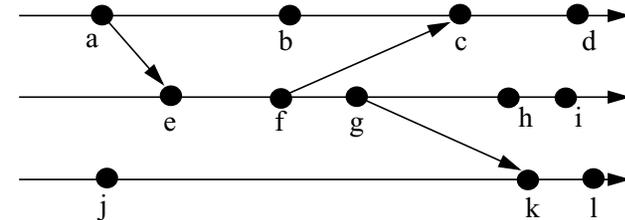
Gummiband-
transformation

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 "Querpfleile")



- interessant: von links nach rechts verlaufende "Kausalitätspfade"
- auf graphische Darstellungsmöglichkeiten bei der Definition verzichten

- 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

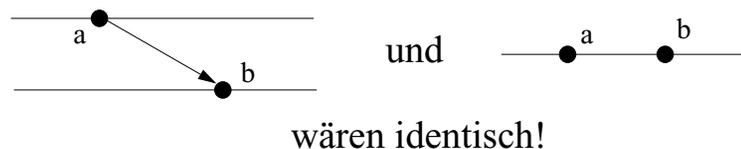
- Verteilte Berechnung = $(E, <)$

Menge von Ereignissen

Ordnungsrelation auf E

- ist eine verteilte Berechnung also das selbe wie eine Ordnungsrelation?
- zumindest eine spezielle mit mehr Struktur?

Beachte:



\Rightarrow Prozesse einführen

- Was sind Prozesse (formal / abstrakt)?

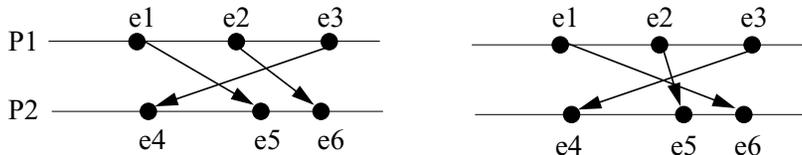
- Partitioniere E in disjunkte Mengen E_1, \dots, E_n
- Interpretation: $E_i =$ Ereignisse auf Prozess P_i

dort *linear* geordnet

Zweiter Definitionsversuch

- Verteilte Berechnung = $(E_1, \dots, E_n, <)$ mit:
 - alle E_i paarweise disjunkt
 - $<$ ist (irreflexive) Halbordnung auf $E_1 \cup \dots \cup E_n$
 - $<$ 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
 \Rightarrow nicht unterscheidbar, obwohl
 wesentlich verschiedene Diagramme
 (also verschiedene Berechnungen)!

\Rightarrow Nachrichten einführen
 (eindeutige Zuordnung zusammen-
 gehöriger send-/receive-Ereignisse)

Dritter Definitionsversuch

- (n -fach) verteilte Berechnung (mit asynchroner Nachrichten-Kommunikation) = $(E_1, \dots, E_n, \Gamma, <)$ mit:

was wäre bei syn. Komm. anders?

- 1) [Ereignisse] Alle E_i sind paarweise disjunkt
- 2) [Nachrichten] Sei $E = E_1 \cup \dots \cup E_n$
 Für $\Gamma \subseteq S \times R$ mit $S, R \subseteq E$ und $S \cap R = \emptyset$ 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) $(s, r) \in \Gamma \Rightarrow s < r$

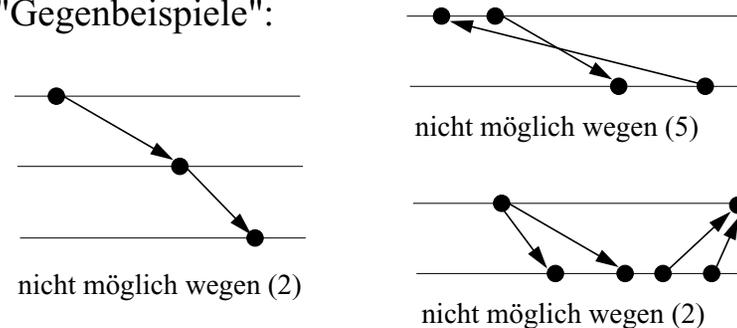
5) $<$ ist eine irreflexive Halbordnung auf E

6) $<$ ist die kleinste Relation, die 3) - 5) erfüllt

(d.h.: es stehen keine weiteren als die dadurch festgelegten Ereignisse in $<$ -Relation zueinander)

[Kausalrelation]

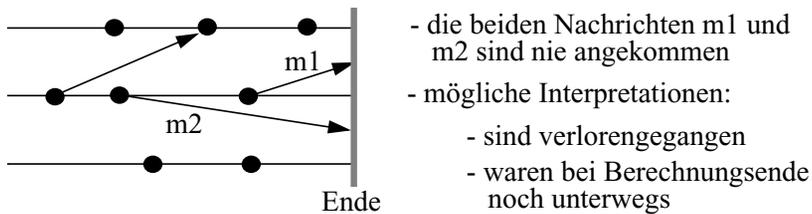
- "Gegenbeispiele":



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*:



- Kann man Einzelprozesse als *Folgen von Ereignissen* auffassen?
- Ergibt sich bei einer vert. Berechnung aus einem *einzigem* 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

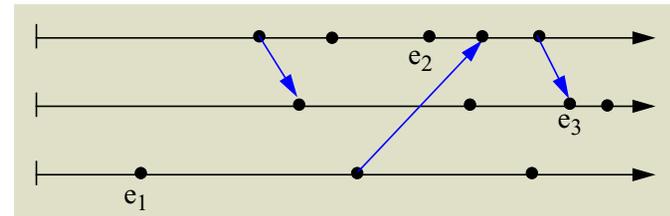
Zeitdiagramme

- 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

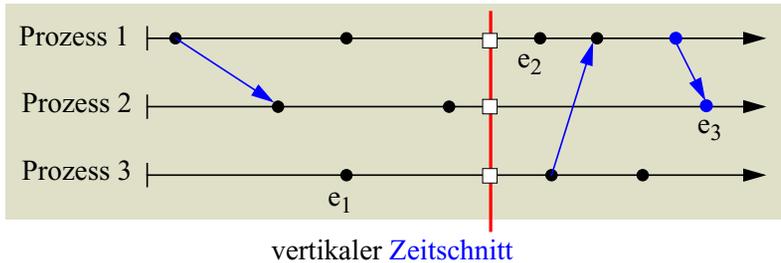
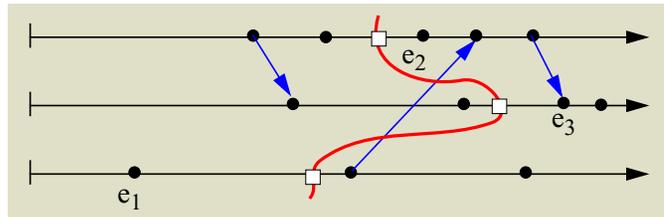


Diagramme sind daher äquivalent

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...

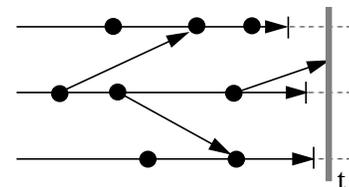
- *Lokaler Zustand* eines Prozesses ist klar

- Def. Zustandsraum (Kreuzprodukt Variablenzustände...) klar
- Zustand zwischen zwei atomaren Aktionen / Ereignissen konstant (d.h. Ereignisse transformieren lokale Zustände)

- *Globaler Zustandsraum*:

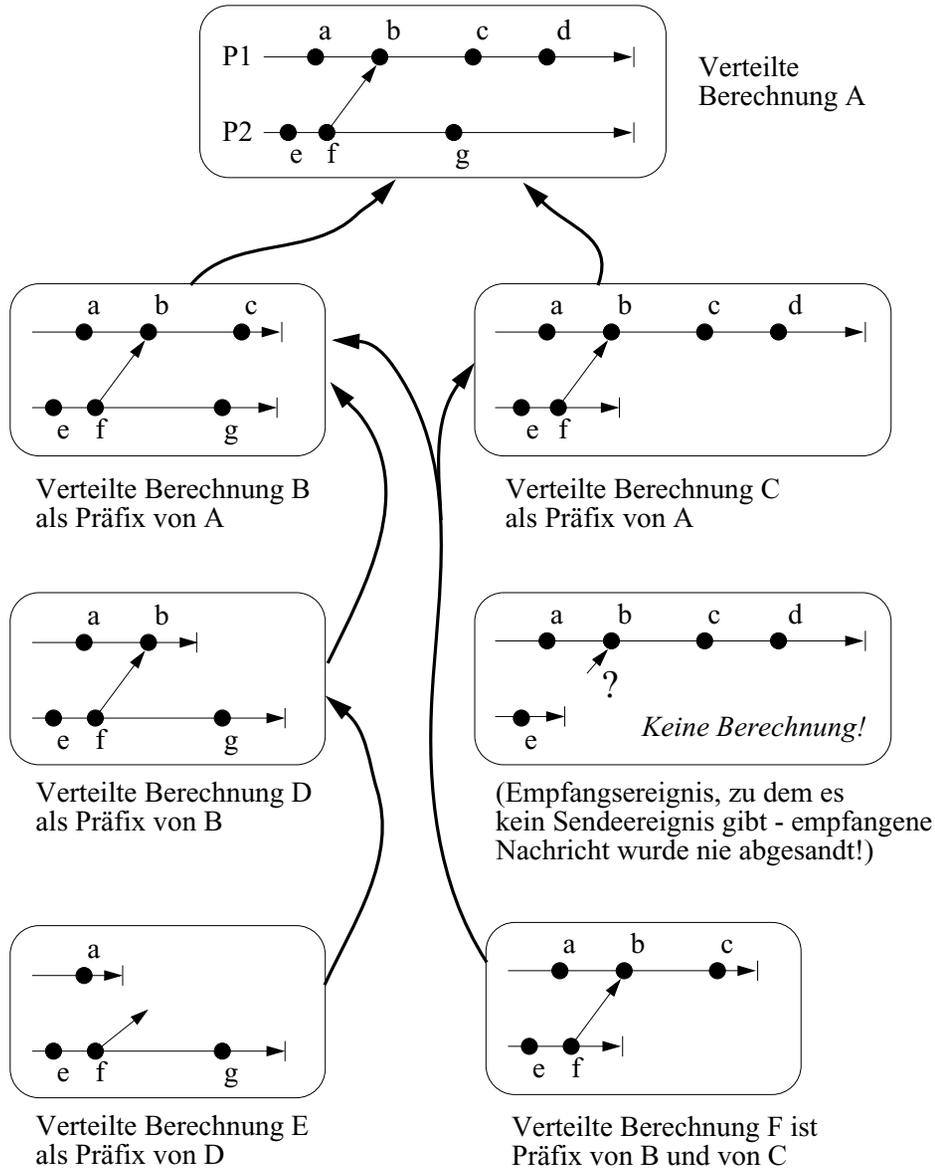
- Kreuzprodukt lokaler Zustandsräume
- Menge von Nachrichten ("in transit")

- Beachte: "Globaler Zustand" lässt sich zunächst nur für einen bestimmten (gemeinsamen) *Zeitpunkt* definieren!



- benutze hilfswise den *Endzustand* der verteilten Berechnung
- dieser lässt sich am Ende einfach einsammeln (wird nicht mehr verändert)
- Zwischenzustände lassen sich dann ggf. als Endzustände geeigneter Präfixberechnungen definieren

Präfixe von Berechnungen



Präfix-Berechnungen

- Gegeben sei eine Berechnung $B = (E_1, \dots, E_n, \Gamma, <)$;
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

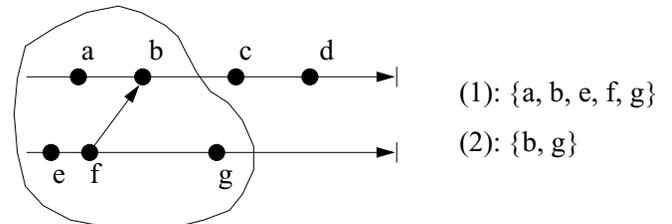
- Wir müssen aber noch fordern, dass E' *linksabgeschlossen* bzgl. der Kausalrelation $<$ ist:

$$\forall x \in E', y \in E: y < x \Rightarrow y \in E'$$

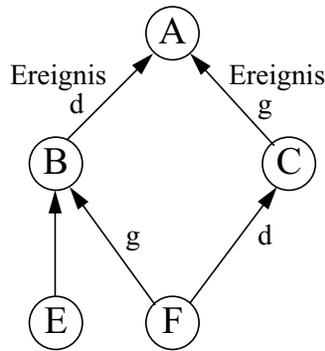
- \Rightarrow lokale Berechnungen sind wirkliche Anfangsstücke
- \Rightarrow unmögliche Diagramme, wo Nachrichten empfangen aber nicht gesendet werden, sind "automatisch" ausgeschlossen (dies wurde allerdings schon durch Γ' als Einschränkungen von Γ garantiert)

- Zu einer gegebenen Berechnung $B = (E_1, \dots, E_n, \Gamma, <)$ ist eine Präfixberechnung B' eindeutig bestimmt durch:
 - (1) die Menge aller ihrer Ereignisse E' , *oder*
 - (2) die Menge der jeweils lokal letzten Ereignisse ("Front" von E' ist eine kompaktere Repräsentation!)

Beispiel:



Präfix-Relation

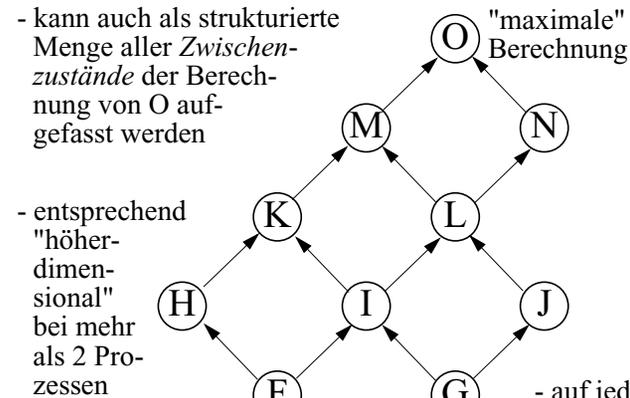


- Präfixdiagramm
 - kein (Wurzel)baum, aber
 - *gerichtet* und *zyklenfrei*
- Präfixrelation ist *transitiv*
- \Rightarrow Präfixrelation ist eine *Halbordnung!*

- 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.a. 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 \rightarrow keine vernünftige Frage!

- Ein Zwischenzustand hat also i.a. mehrere direkte Vorgänger- und Nachfolgezustände!
- Berechnung bewegt sich in diffuser Weise in diesem Zustandsraum von unten nach oben
 - \uparrow von den Ecken her ausgefranztes 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

⇒ *Wellenalgorithmen*

- *Alle* Prozesse müssen sich beteiligen
- *Basisalgorithmen* ("Bausteine") für andere Algorithmen (wechselseitiger Ausschluss, Terminierungserkennung, Election...)

Abstraktere Definition:

Algo. ist ggf. 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:

1. $init \leq visit$
 2. $visit \leq conclude$
- } ($\Rightarrow init \leq conclude$)

Wellenalgorithmen (2)

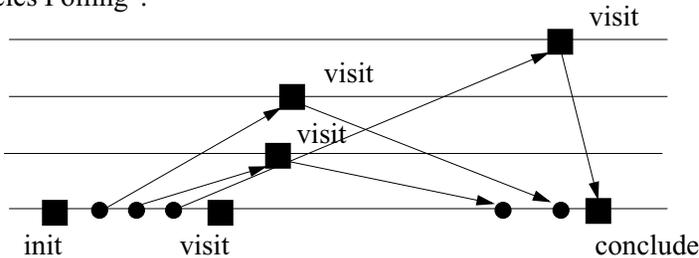
Aus 1: Über die Kausalketten lässt sich *Information* vom Initiator an alle Prozesse *verteilen*

Aus 2: - Nach conclude wurde jeder besucht (→ Terminierung!)
 - Über die Kausalketten lässt sich *Information* von allen Prozessen *einsammeln*

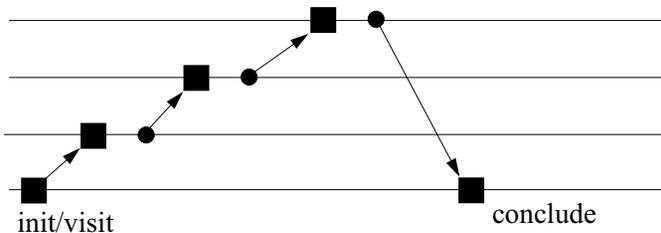
Bem.: a) init und conclude oft im selben Prozess ("Initiator")
 b) init oder conclude kann mit visit verschmelzen (daher '≤' statt '<')

Beispiele:

"paralleles Polling":



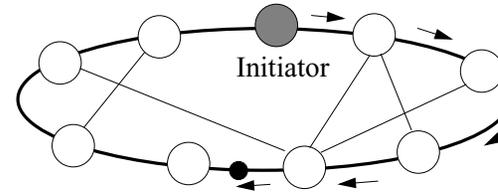
"Ring":



Einige Wellenalgorithmen

- init-, visit-, conclude-Ereignisse jeweils sinnvoll festlegen!

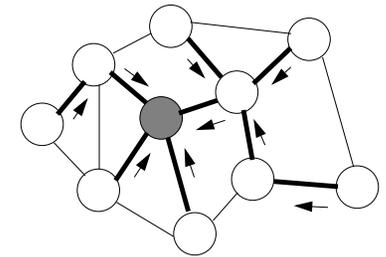
1.) *Ring / Hamiltonscher Zyklus* mit umlaufenden Token



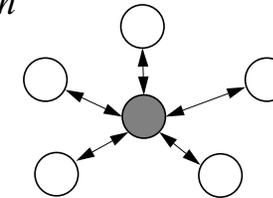
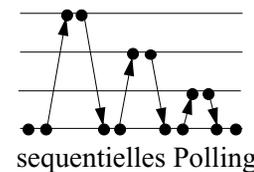
- "logischer" Ring genügt!
 - in einem zusammenhängenden ungerichteten Graphen kann ein logischer Ring immer gefunden werden, indem man einen Spannbaum "umfährt"

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*



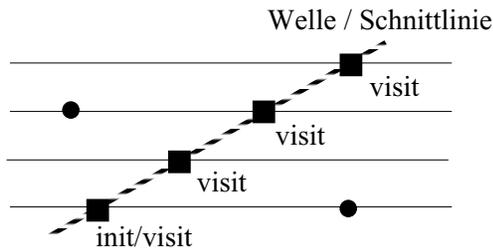
- "Polling" entweder sequentiell (jeweils höchstens eine Nachricht unterwegs) oder parallel

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"!

Wellen und Schnittlinien

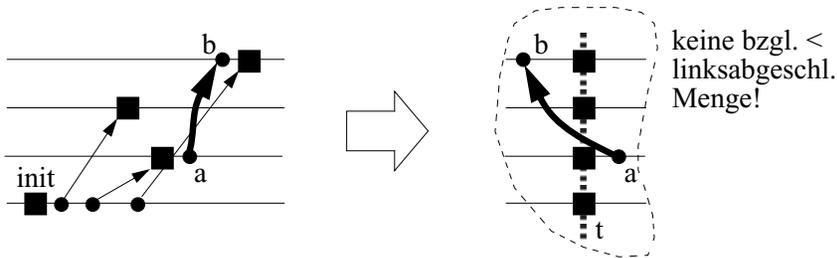
- Verbinden der visit-Ereignisse zu einer *Schnittlinie*:



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?



- 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 gleichzeitig" 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)

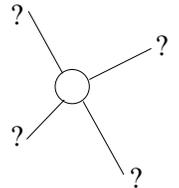
Eigenschaften von Wellenalgorithmien

wodurch ist eigentlich der suggestive Name gerechtfertigt?

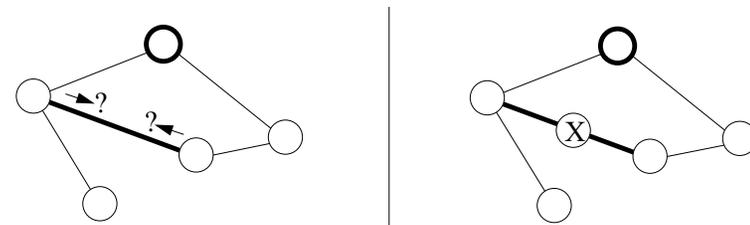
Satz: Es werden *mindestens n-1* Nachrichten benötigt

- 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 (\neq 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 zyklensfrei, } Denkübung:
 - oder die Menge dieser Kanten ist zusammenhängend } wieso?
- Damit und mit $n-1$ folgt die Baumeigenschaft

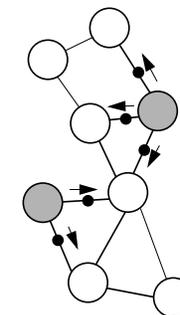
Virtuell gleichzeitiges Markieren

- Als Anwendung eines (Halb)wellenalgorithmus (\approx flooding)
- Voraussetzung hier: FIFO-Kanäle (d.h. keine Überholungen)

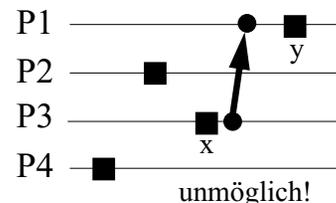
I: {not marked}
 marked := true; // = init = visit
send <marker> to all neighbors;

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?*)



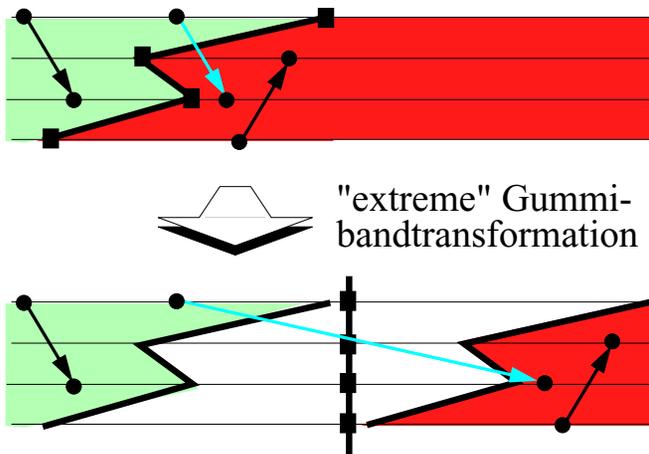
Eine "rückwärts" über den Schnitt laufende Basisnachricht kann es nicht geben: Wenn P3 und P1 benachbart sind, dann wurde bei x ein Marker an P1 gesendet, der nicht (wie von der gezeichneten Nachricht) überholt werden kann

- Fragen:
- geht virtuell gleichzeitiges Markieren auch ohne FIFO-Kanäle? (FIFO abschwächen, FIFO erzwingen, FIFO "simulieren"...)
 - geht das auch mit weniger als $2e$ Nachrichten?

- 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!

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 untereinander legen

Anwendung von Wellenalgorithmen

- Als "Unterprogramm" innerhalb anderer Anwendungen, z.B.
 - Broadcast einer Information an alle Prozesse
 - Einsammeln verteilter Daten
 - Traversieren aller Prozesse 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 ggf.: 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 Wellenalgorithmien (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)
- *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. BRISSÉ,
PROFESSEUR A L'ÉCOLE GÉNÉRALE ET AU LYCÉE CONDORCET,
RÉPÉTITEUR A L'ÉCOLE POLYTECHNIQUE,

ET
M. E. ROUCHE,
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
(Tous droits réservés.)

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 intricable, et que le meilleur fil d'Ariane est le fil du raisonnement.

Gaston Tarry (1843-1913) studied mathematics at secondary school in Paris and joined the civil service. 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.



Römisches Mosaik (aus Loig bei Salzburg) illustriert die Geschichte von Theseus und Minotaurus im Labyrinth



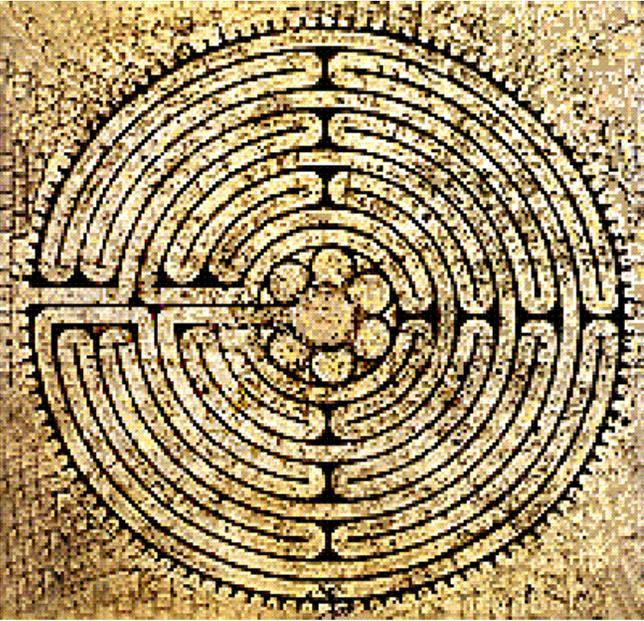
Arcera, Spanien



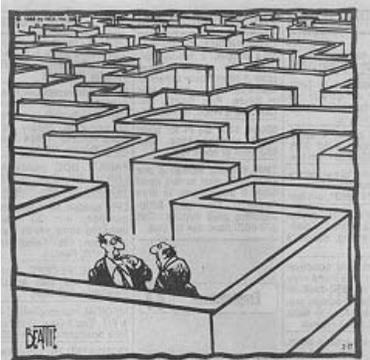
O'odham-Volk
(Papago-Indianer)
aus Süd-Arizona

Literatur:
Janet Bord: Irrgärten und
Labyrinth, DuMont
Buchverlag 1976

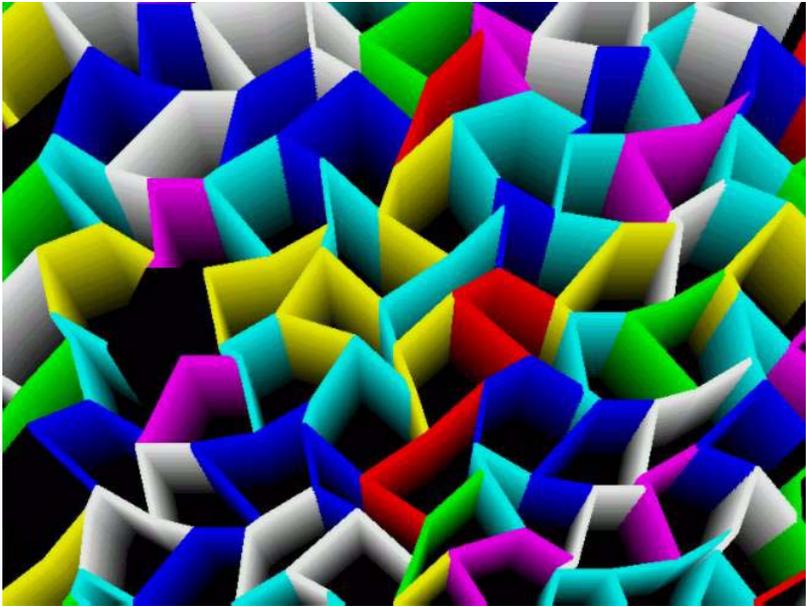
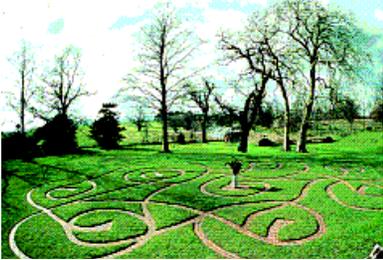
Labyrinthe (2)



Labyrinthe (3)



"The exit? Sure... take a right, then left, left again... no wait... a *right*, then... no, wait..."



Labyrinth (4)



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 (\neq 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.: ...

Tarry's Verfahren ist ein Wellenalgorithmus

(1) Terminierung \Rightarrow 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 \Rightarrow es gibt noch mindestens einen unbenutzten "Ausgangskanal" \Rightarrow 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 anderen 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) \Rightarrow Jeder "Eingangskanal" des Initiators 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. $\Rightarrow p$ 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. \Rightarrow 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)

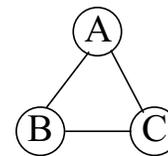
Depth-first-Algorithmen und Spannbäume

Klassischer Depth-first-search-Algorithmus:

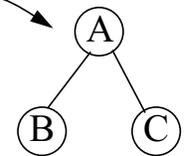
- Token geht erst dann zurück, wenn alles andere "abgegrast" ist
- Token kehrt um, sobald es auf einen bereits besuchten Knoten trifft
- jede Kante wird in jede Richtung genau 1 Mal durchlaufen
- Tarry-Algorithmus lässt sich zu Depth-first-Traversierung spezialisieren

\Rightarrow $2e$ 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?



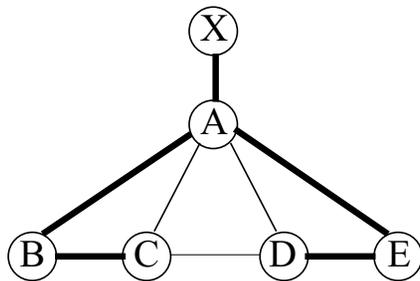
- Eine Charakterisierung solcher Spannbäume:

Jede Kante des Graphen, die keine Kante des Spannbaumes 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?

Es gibt verschiedene Wellenalgorithmmen

- 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 (→ 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~~

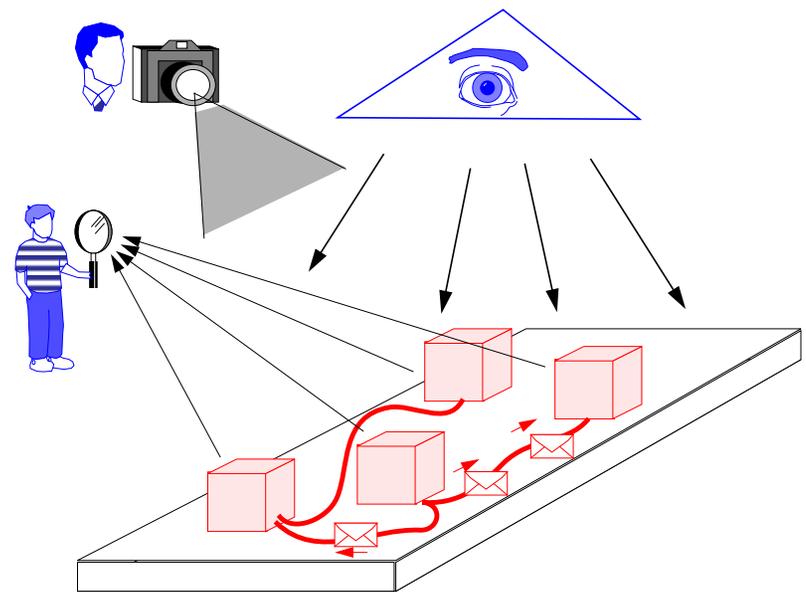
- Anwendung von Wellenalgorithmen (u.a.):

- *Broadcast*
- *Einsammeln* von verteilten Daten ("gather")
- Konstruktion eines *Spannbaumes*
- *Phasensynchronisation* von Prozessen
- *Triggern eines Ereignisses* in jedem Prozess
- Implementierung von *Schnittlinien* (→ Schnappschuss etc.)
- *Basisalgorithmus* für andere Verfahren (Deadlock, Terminierung,...)
- Bestimmung des *glob. Infimums* (z.B.: "ist ein flag gesetzt?")

- Es gibt viele Wellenalgorithmen ⇒ 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)!
- *Aufgabe*: Diesbezüglicher Vergleich aller Wellenalgorithmien!

Globale Zustände, Beobachtungen, Prädikate



Kausal-konsistente globale Zustände

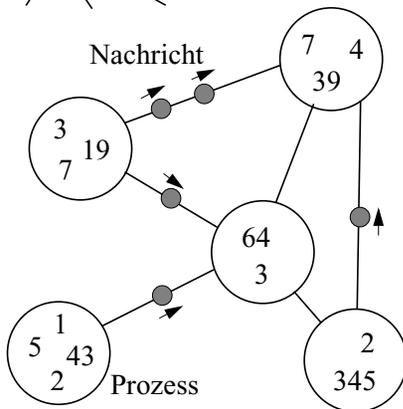
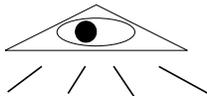
Webster:

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



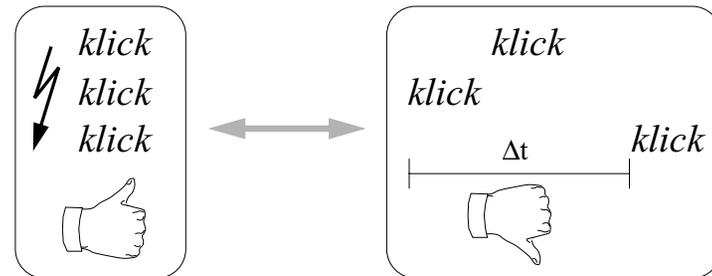
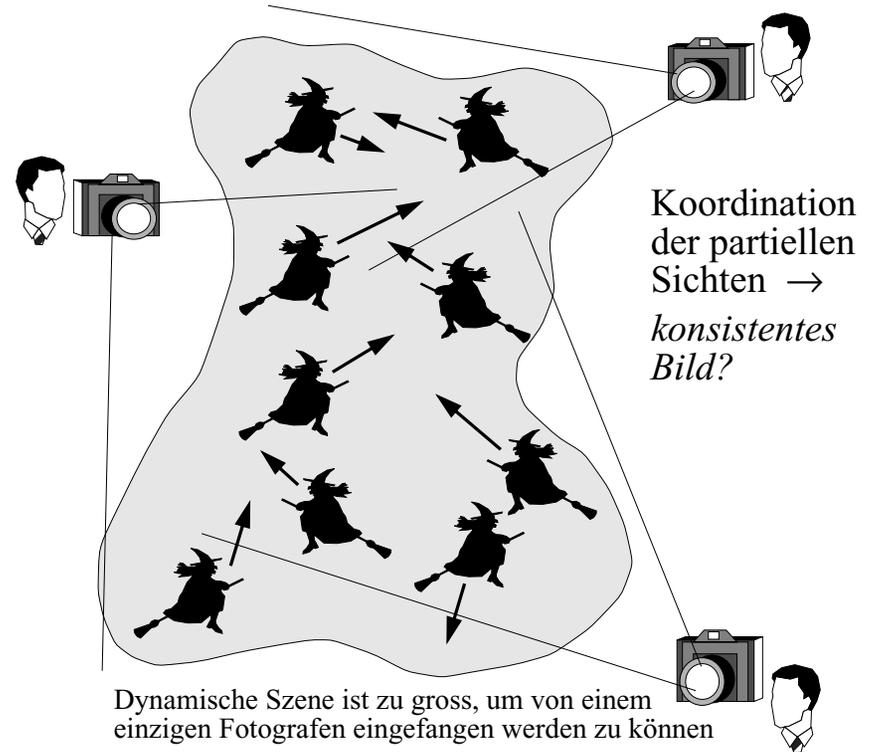
Schnappschuss

Problem:

Prozesszustände sind nur einzeln ("nacheinander") erfahrbar → Konsistenz?

als ob alles gleichzeitig wahrgenommen wird

Synchronisierte lokale Schnappschüsse?



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 vert. 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

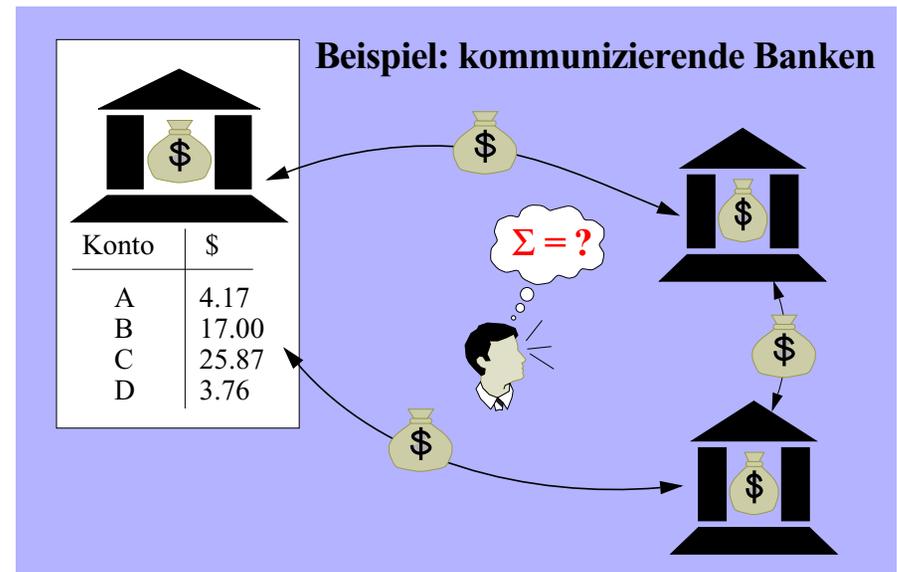
Schwierigkeiten:

- unmöglich, alle Prozesse gleichzeitig zu erwischen
- unbestimmte Nachrichtenlaufzeiten
- Nachrichten, die unterwegs sind, sieht man nicht
- ermittelter Zustand ist i.a. veraltet
- ... u.U. nie "wirklich" so gewesen
- ... u.U. inkonsistent

zumindst 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?

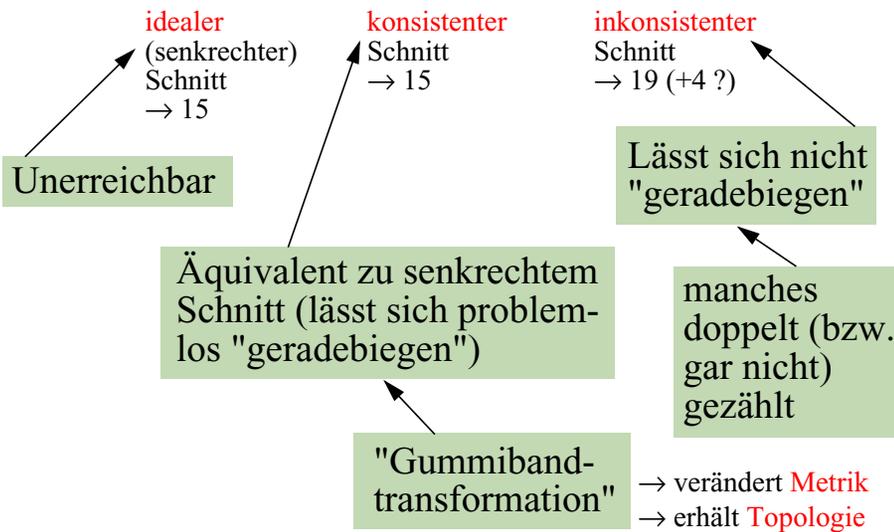
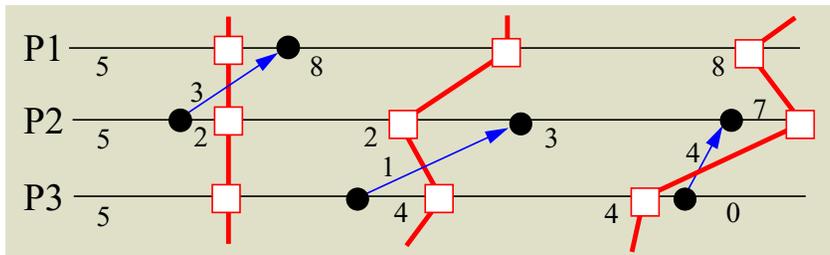
- **konstante** Geldmenge, oder
- **monotone** Inflation (→ untere Schranke für momentane Geldmenge)

- Erschwerte Bedingungen:

- niemand hat eine **globale Sicht**
- gemeinsame **Zeit**?

(In)konsistente Schnitte

Beispiel: Wieviel Geld ist in Umlauf?



Wie wir noch einsehen werden:

Kausaltreues Beobachten \Leftrightarrow konsistente 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"?

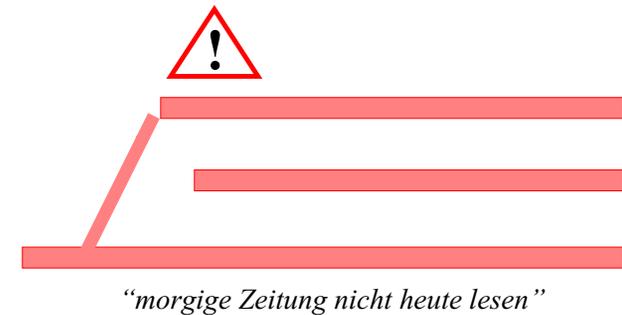
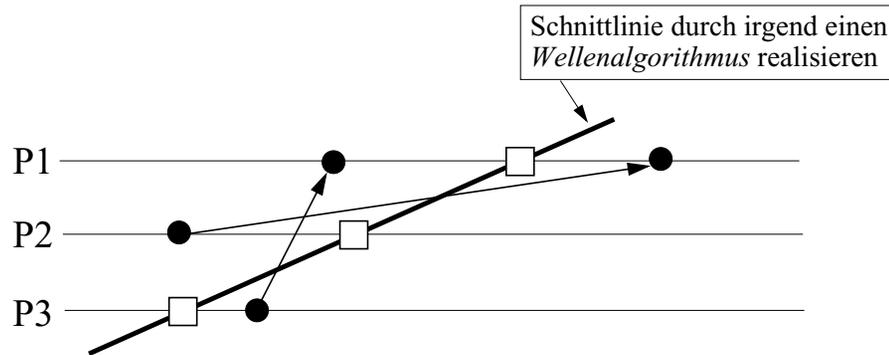
- Liefern *konsistenten, möglichen, vergangenen Zustand*

- *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 gilt (also in allen zukünftigen Zuständen gilt); z.B. Terminierung, Garbage, Deadlock...

- Falls Prädikat stabil: "entdecken" dieses Prädikat (\Rightarrow "stable property detection algorithm")

- bei stabilen Prädikaten ist "möglich... vergangen" sogar brauchbar! (es gilt dann jetzt *sicherlich*)
- aber wenn die betrachtete Eigenschaft nicht stabil ist, was dann?
- wer garantiert eigentlich, dass eine Eigenschaft (wirklich) stabil ist?

Ein Schnappschussalgorithmus



Prozesse, Nachrichten: *schwarz* oder *rot*
Schnappschussaugenblick: *schwarz* → *rot*
(dann: lokalen Zustand dem Initiator melden)
Prozess wird *rot*, wenn a) Aufforderung erhalten,
b) Erhalt einer roten Nachricht

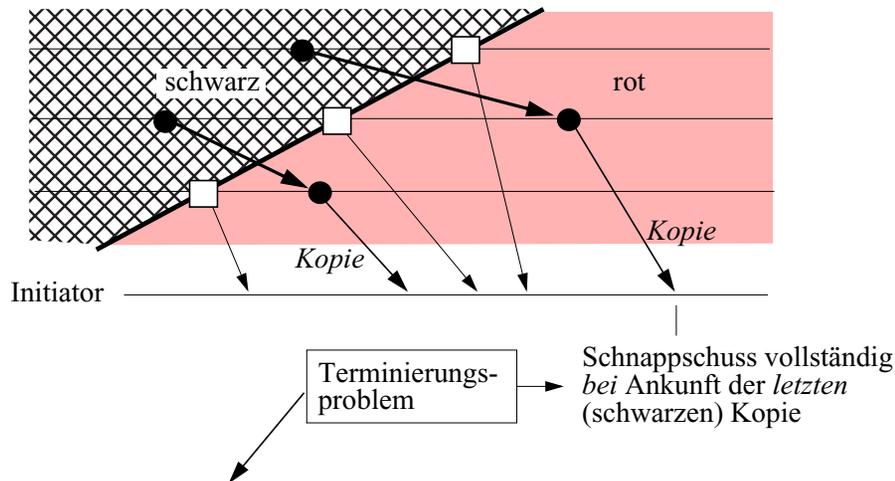
Beh.: Schnappschuss ist *konsistent*.
Bew.: Keine "Nachricht aus der Zukunft"

Nachrichten, die *unterwegs* sind?
- *Schwarze* Nachrichten, die bei *rot* ankommen
- Sende *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

- Wellenalgorithmus als "Basisalgorithmen" -> 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

(erster Schnappschussalgorithmus, 1985 veröffentlicht in ACM TOCS 3, 63-75)

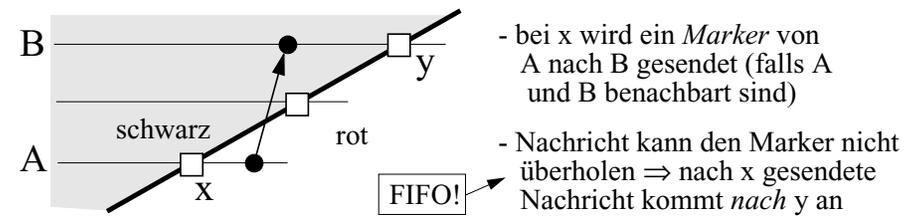
- Idee: 1) Setzt FIFO-Kanäle voraus ("flushing-Prinzip")

Marker schieben In-transit-Nachrichten aus den Kanälen heraus

2) Flooding als zugrundeliegendes Wellenverfahren

- $\neg \exists$ Nachricht aus der Zukunft \Rightarrow Schnitt ist *konsistent*

- vgl. auch frühere Ausführungen zu "virtuell gleichzeitiges Markieren"



- 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 ankommen

- Vorteil: Farben müssen nicht (in Nachrichten) mitgeführt werden

- Nachteile:

- bei dichten Netzen grosse Zahl von Kontrollnachrichten
- FIFO ist notwendig
- lokale Zustände müssen i.a. zum Initiator gebracht werden (z.B. mittels Echo-Nachrichten)

Der Chandy/Lamport-Algorithmus (2)

- 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*

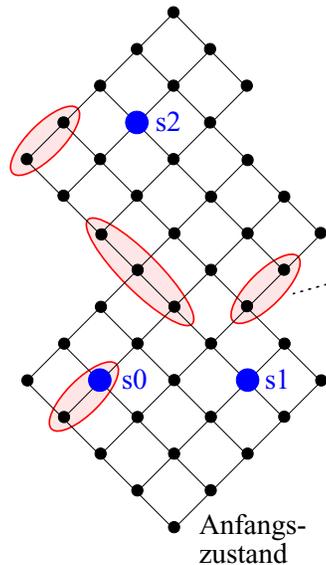
Türken einen Tag unter "Hausarrest" gestellt

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.

- Wie verhält sich ein Initiator? Kann der Algorithmus "spontan" von mehreren Prozessen unabhängig voneinander initiiert werden?
- Wieso ist im ersten Fall der Kanalzustand die leere Folge? Hat jeder so gewonnene globale Zustand einige leere Kanäle?

Eigenschaften verteilter Berechnungen mit Schnappschüssen "entdecken"?

- Ein **wiederholt angewendeter Schnappschussalgorithmus** könnte 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)

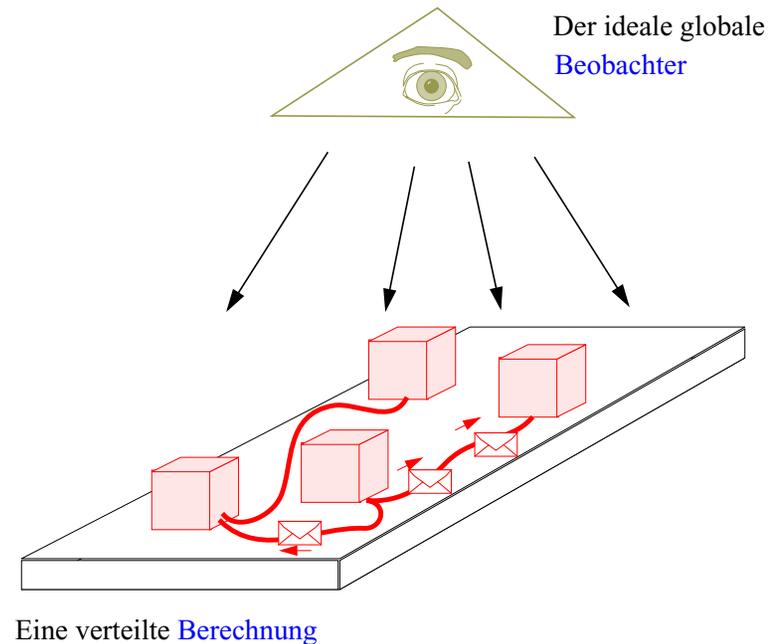
Eigenschaft sei in diesen Bereichen gültig (wird jedoch von s1 oder s2 nicht "entdeckt", allenfalls von s0)

wenn dies in einem Zustand gilt, dann auch in allen späteren

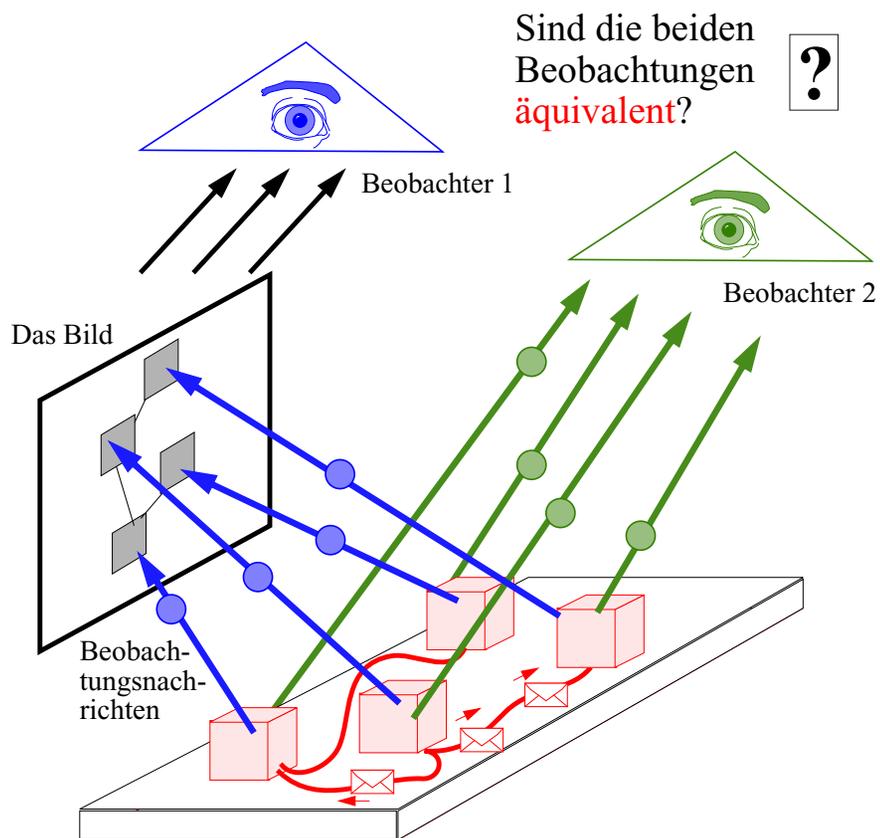
- Sinnvoll, wenn die Eigenschaft **stabil** ist - aber ansonsten?
 - beachte: wir wissen nicht, ob "in Wirklichkeit" s0 oder s1 eintritt!

- Wir hätten gerne eine **lückenlose "Folge" konsistenter Schnappschüsse** als eine "**Beobachtung**" der Berechnung
- Allerdings sind Berechnungen nur **halbgeordnete** Mengen (konsistenter) Zustände (also keine Folgen)!

Verteilte Berechnung und Beobachtung



Beobachtungen...



Probleme:

- Zeitverzögerung der Beobachtung
- Konsistenz des Bildes
- Verzerrung des Verhaltens ("Heisenberg'sche Unschärfe")

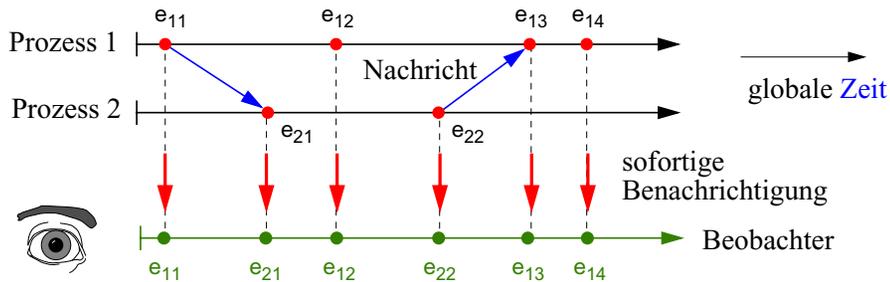
Das Paradigma der Beobachtung

- "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* ?

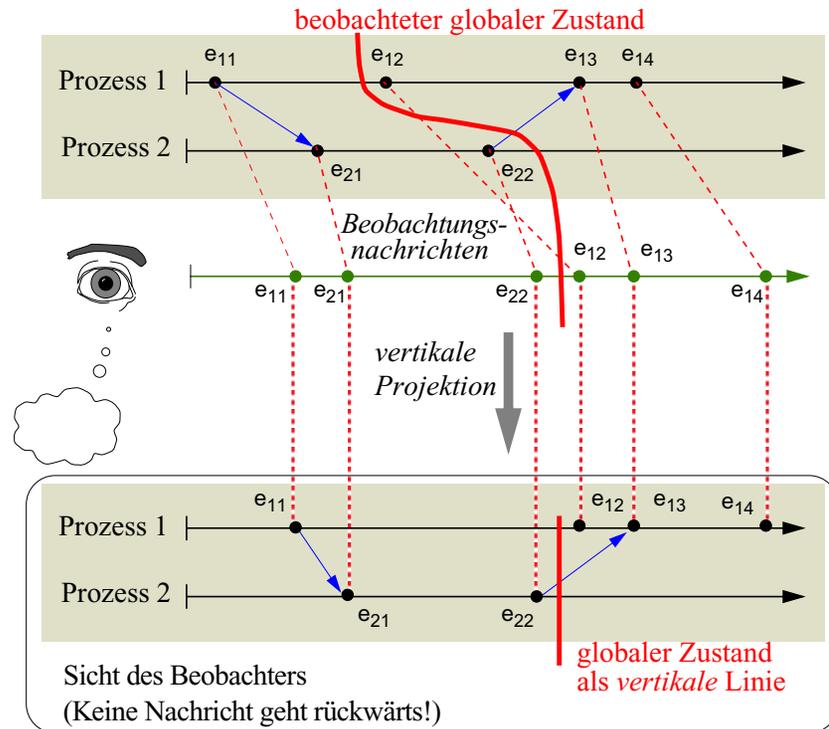
Ideale Beobachtungen



- 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 Beobachtungsnachrichten so überholen, dass Ursache und Wirkung vertauscht werden)

Kausaltreue Beobachtungen

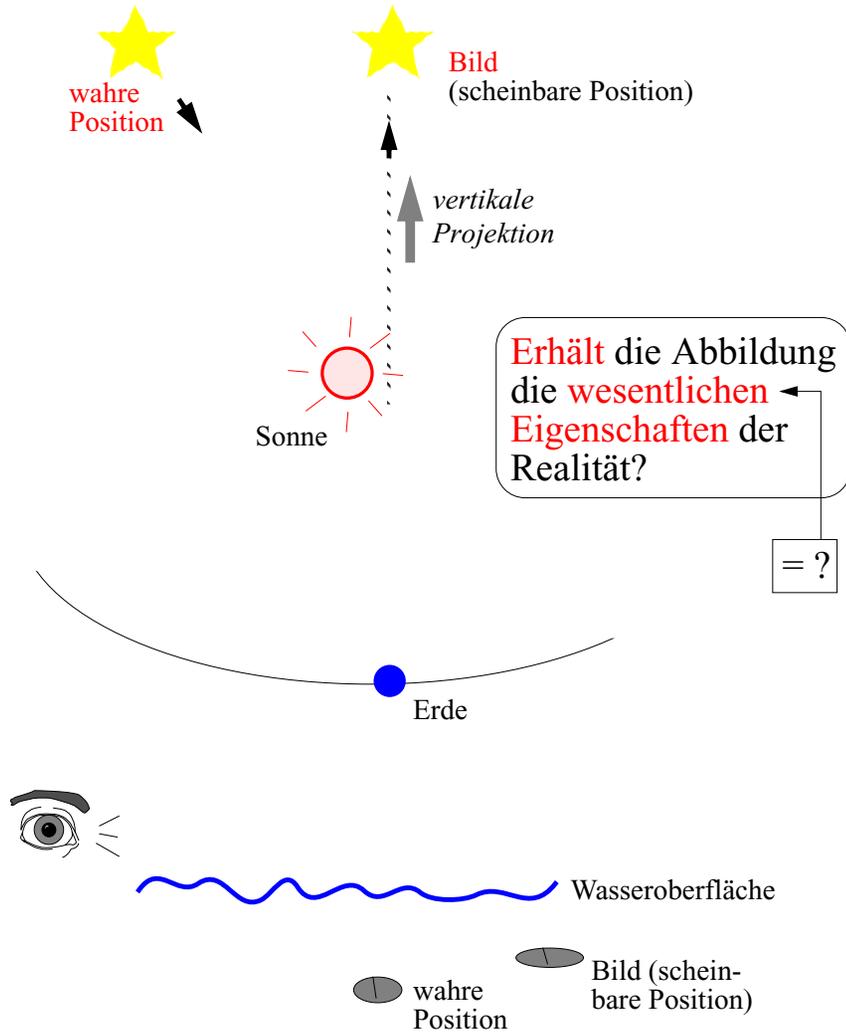
- **Ursachen** werden stets **vor** ihren **Wirkungen** angezeigt
- dies hoffen wir zu bekommen! (aber wie?)



- 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

Bild, Schein und Wirklichkeit

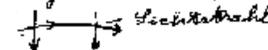


- Offenbar kann man mit einem gegenüber der Realität "etwas" verzerrten Bild ganz gut leben...

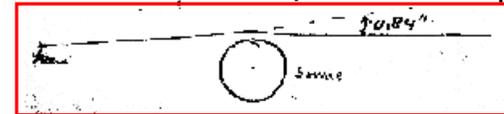
Zürich, 19. I. 13.

Hoch geehrter Herr Kollege!

Eine einfache theoretische Überlegung macht die Annahme plausibel, dass Lichtstrahlen in einem Gravitationsfeld eine ^{geringfügige} Deviation erfahren.



Aus Sonnenrande müsste diese Ablenkung $0,84''$ betragen und wird $\frac{1}{2}$ abnehmen (auf Grund der Sonnen-^{Wärmestrahlung} ~~Wärmestrahlung~~ ^{Abstand} ~~Abstand~~ ^{zum Beobachter} ~~zum Beobachter~~ ^{R = Sonnenradius} ~~R = Sonnenradius~~).



Es wäre deshalb von grösstem Interesse, bis zu wie grosser Sonnen-nähe gewisse Fixsterne bei Anwendung der stärksten Vergrösserungen bei Tage (ohne Sonnenfinsternis) gesehen werden können.

Auf den Rat meines Kollegen, d. Herrn Prof. Maurer bitte ich Sie deshalb, mir mitzuteilen, was Sie nach Ihrer reichen Erfahrung in diesen Dingen für mit den heutigen Mitteln erreichbar halten.

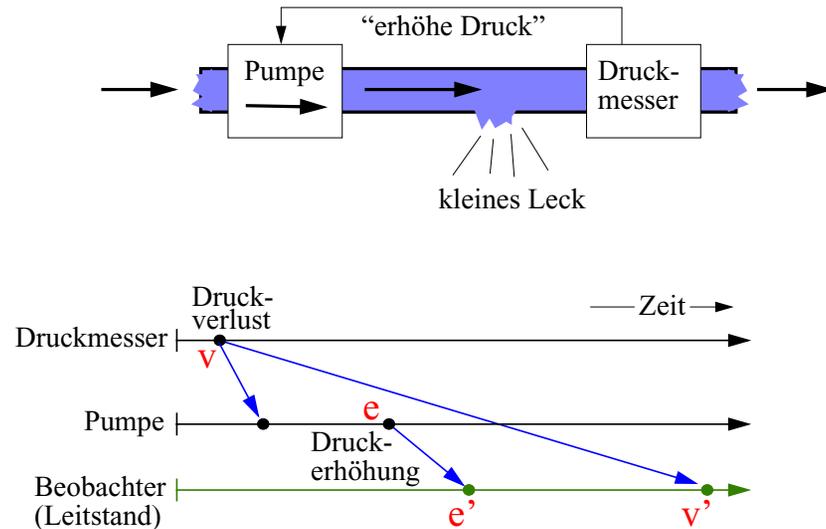
Mit aller Hochachtung
Ihr ganz ergebener

A. Einstein
Technische Hochschule
Zürich.

Brief an George Hale,
Mount Wilson
Observatory, Pasadena

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 ("kausal-treue Beobachtung"):

Ursache stets vor (u.U. indirekter) Wirkung beobachten!

Frage:

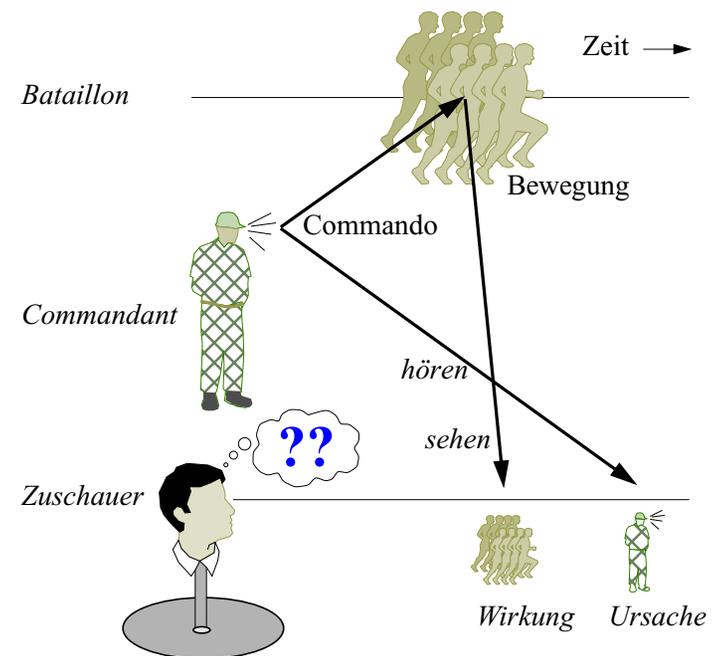
Wie implementiert man kausal konsistente Beobachter?

Kausal (in)konsistente Beobachtungen

Das Beobachtungsproblem ist nicht neu...

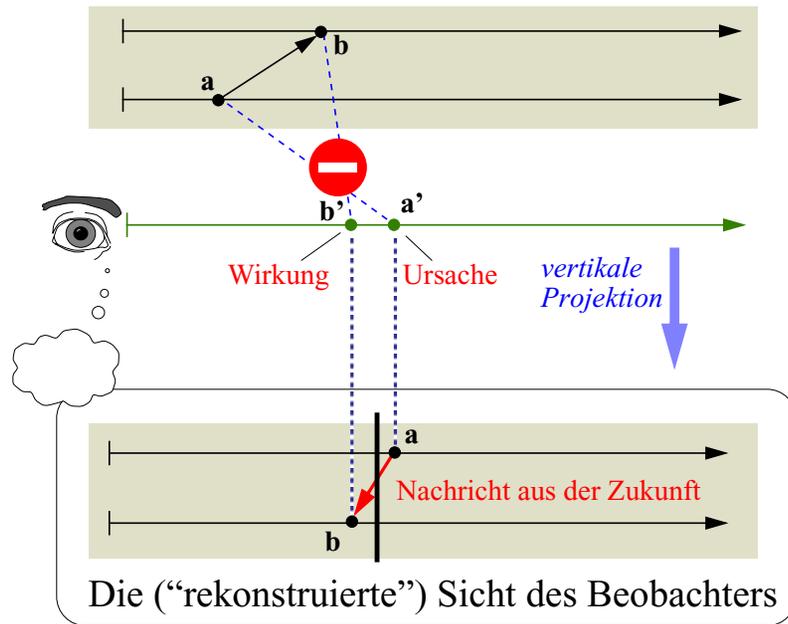
Wenn ein Zuschauer von der Ferne das Exerzieren eines Bataillons verfolgt, so **sieht** er übereinstimmende Bewegungen desselben plötzlich eintreten, *ehe* er die Commandostimme oder das Hornsignal **hört**; aber aus seiner Kenntnis der *Causalzusammenhänge* weiss er, dass die Bewegungen die *Wirkung* 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 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

Definition:

lineare Erweiterung oder Einbettung

Eine *kausaltreue Beobachtung* einer Berechnung ist eine Linearisierung der entsprechenden (partiellen) Kausalordnung $(E, <)$

Mit anderen Worten:

Jeder "Trace" von Ereignissen, in dem eine Wirkung niemals vor ihrer Ursache erscheint, heisst "kausaltreue Beobachtung"

Bemerkung:

Es gibt i.a. viele unterschiedliche Linearisierungen!

- wieviele? (Größenordnung 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, 1930)

⇒ der "unstrittige Kern" aller Beobachter ist die Kausalrelation der Berechnung selbst!

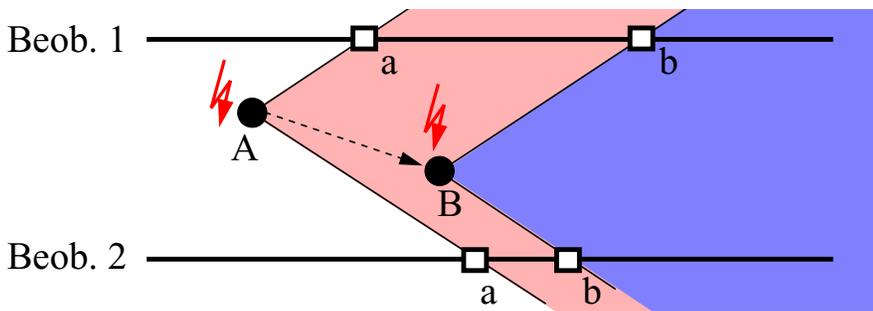
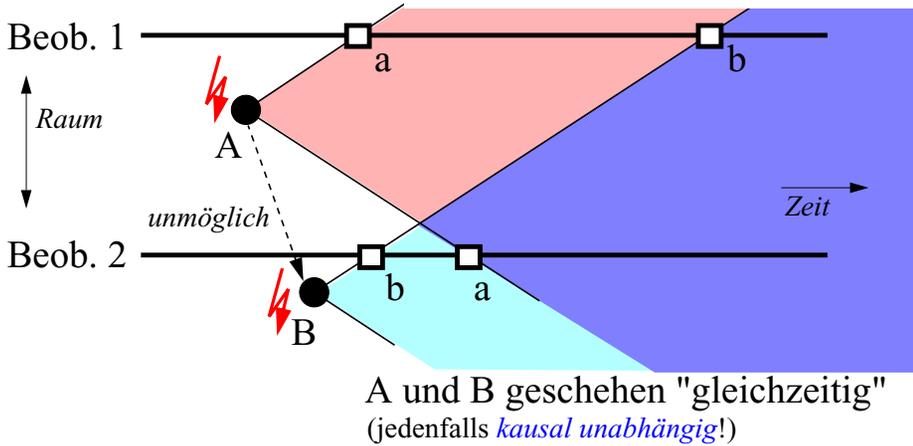
⇒ in den "wesentlichen" Aspekten stimmen alle Beobachter überein! (die Kausalbeziehung ist also ein beobachterinvariantes, objektives Faktum)

⇒ vert. Berechnung ist durch die Menge aller ihrer Beobachtungen charakterisiert

Relativierung der Gleichzeitigkeit

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

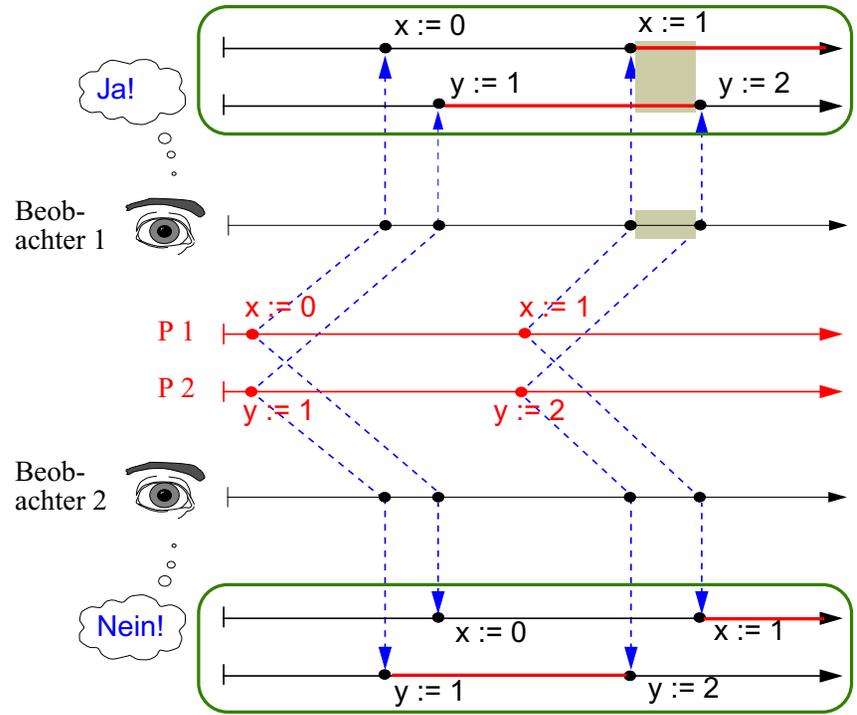
Lichtkegel-Prinzip der relativistischen Physik   



beobachterinvariant
 \Rightarrow objektive Tatsache \rightarrow Alle Beobachter sehen B nach A

Das "Entdecken" globaler Prädikate

Frage: Gilt in dieser Berechnung $\Phi \equiv (x = y)$?

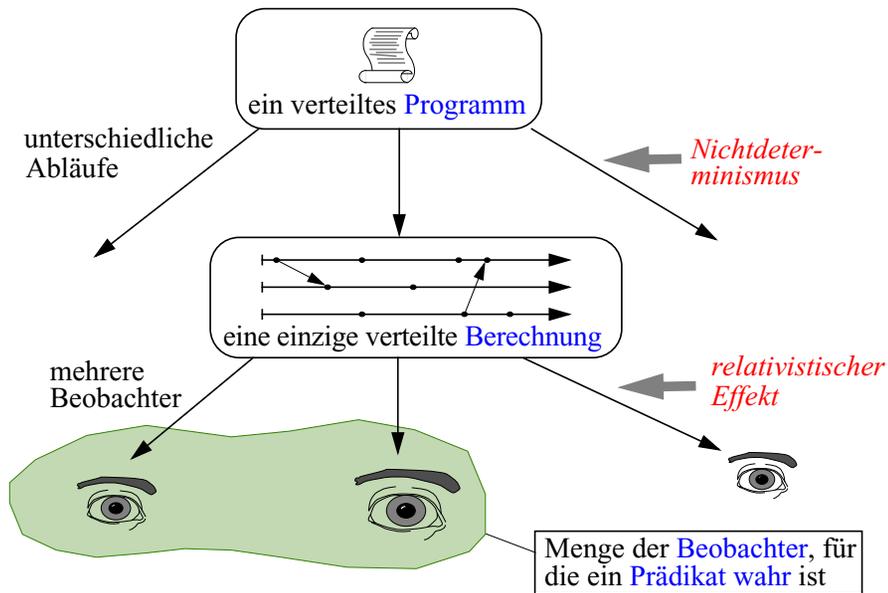


- Meldungen sind alle gleich schnell
- Beide (kausaltreuen!) Beobachtungen sind gleich "richtig"
- Die Beobachter stimmen bzgl. $x = y$ nicht überein!

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!**

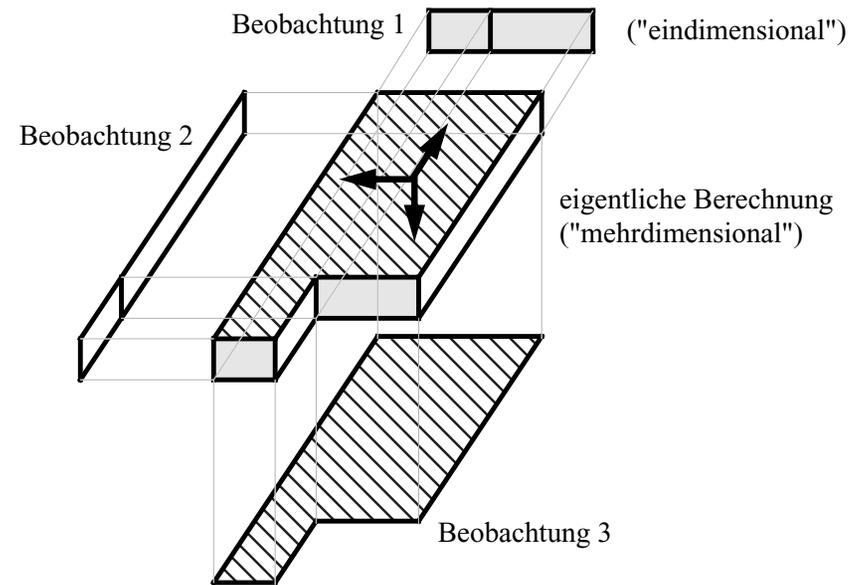


Konsequenz:

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

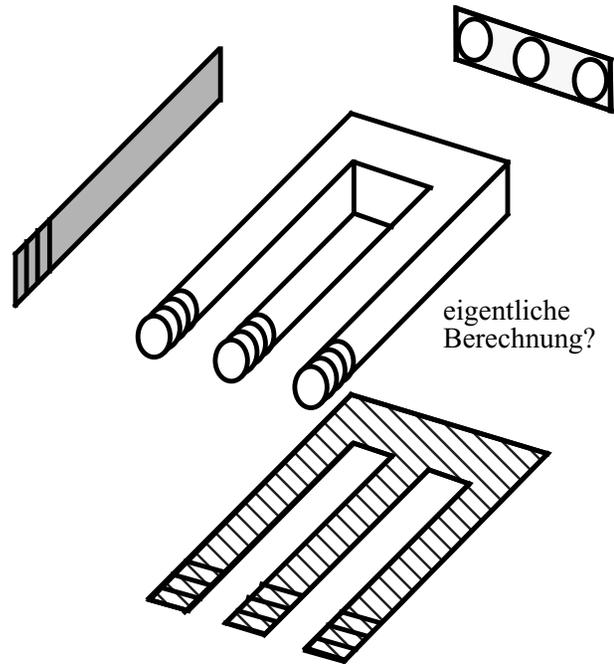
- im sequentiellen Fall: Berechnung = Beobachtung
- im verteilten Fall aber: Berechnung \neq Beobachtung
- **Gültigkeit von Prädikaten ist eine Eigenschaft einer Beobachtung, nicht einer Berechnung!**
- gibt es sinnvolle **beobachterinvariante Prädikate**?

Beobachtung, Bild und Wirklichkeit



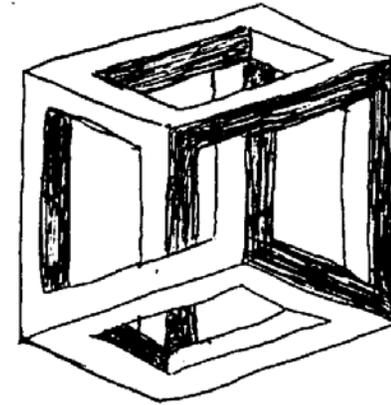
- Einige Eigenschaften gehen durch die Abbildung verloren
 - hier: kausale Unabhängigkeit
- Essentielle Eigenschaften sollen erhalten bleiben
 - hier: kausale Abhängigkeit
- Lässt sich die eigentliche Berechnung aus allen Beobachtungen rekonstruieren?
 - ja: Schnitt aller Linearisierungen ist Halbordnung selbst

Wirklichkeit ?

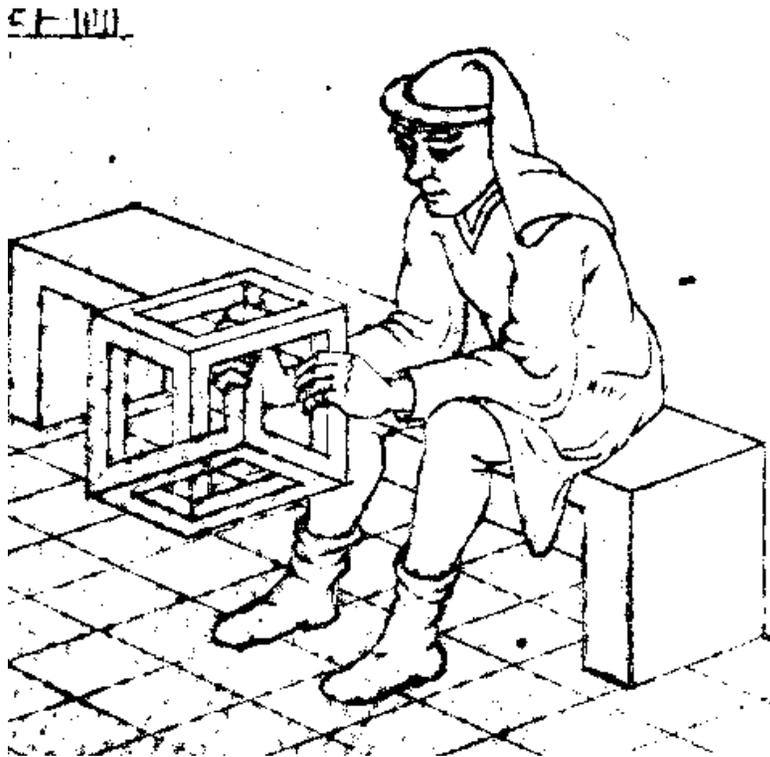


- Sind die verschiedenen Beobachtungen “verträglich”?
- Die beobachtete Wirklichkeit kann weitaus merkwürdiger sein, als eine Beobachtung vermuten lässt!
- Lassen sich die verschiedenen Beobachtungen zu einem konsistenten Bild zusammenfügen?

Ein inkonsistentes Bild



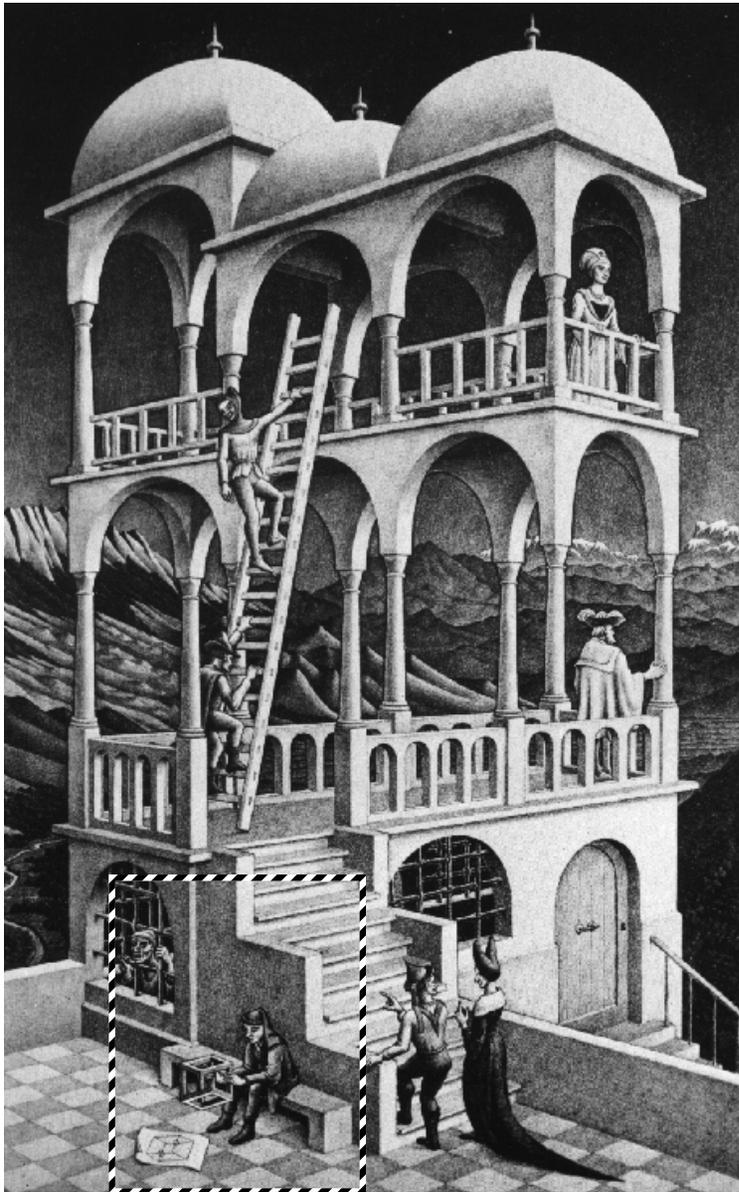
Wie heisst der Künstler?



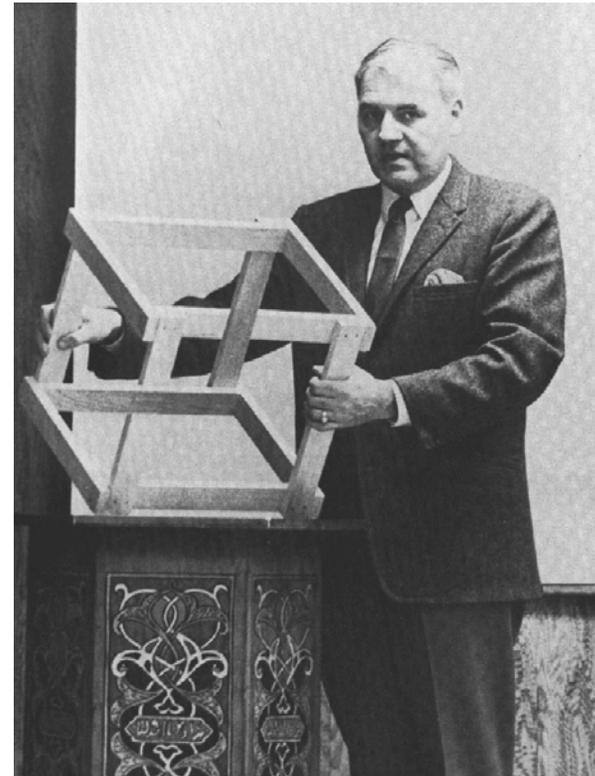
Wie heisst der Künstler?



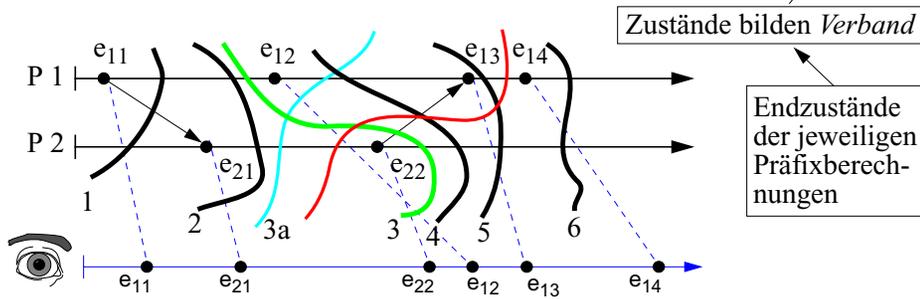
M.C. Escher: Belvedere (1958)



Ein Schnappschuss als Beweis

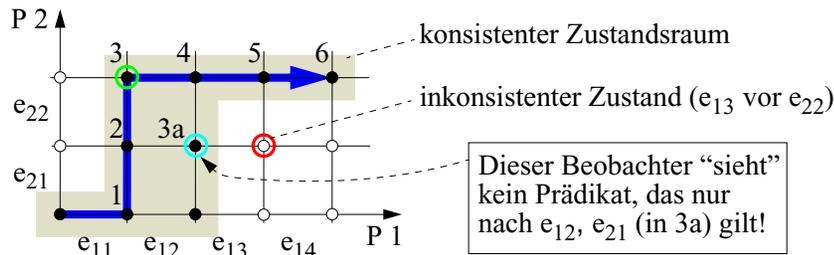


Das n-dimensionale Zustandsgitter



- *Beobachtung* =

- Lineare Folge von Ereignissen
- Folge damit assoziierter globaler Zustände

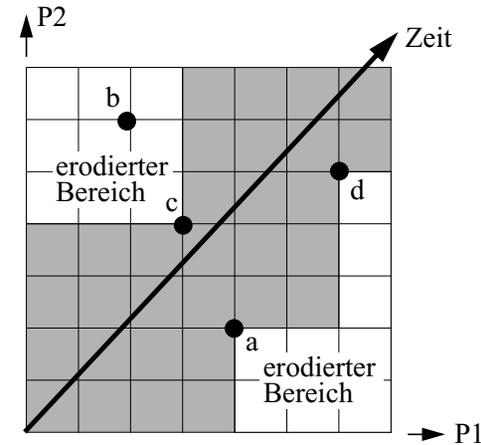
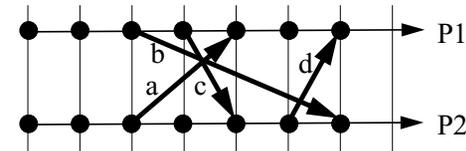


- *Beobachtung* = Pfad von links unten nach rechts oben
(*kausaltreue Beobachtung*, wenn dieser sich nur im "erlaubten" Bereich aufhält)

- Ein Beobachter sieht *alle Ereignisse*, aber nur eine *Teilmenge* aller möglichen *Zustände*!
→ Beobachter kann Prädikate übersehen!
- Mit einem Schnappschussalgorithmus kann nur eine *einzige* (i.a. *lückenhafte*) von vielen Beobachtungen erlangt werden

Der erodierte Zustandshyperwürfel

- Hier: zwei Prozesse → 2-dimensionaler Würfel

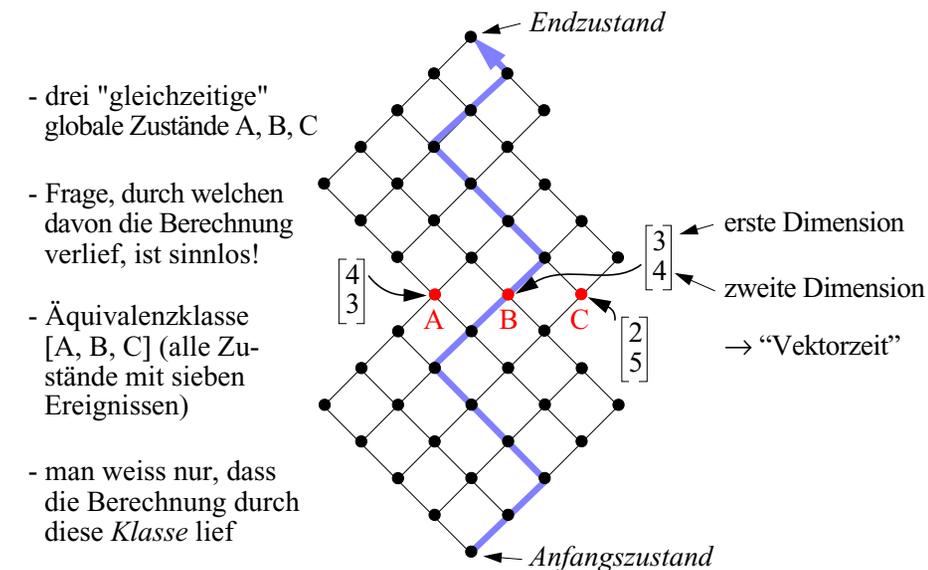


- "Erosion" der inkonsistenten Zustände von den Ecken her

- 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

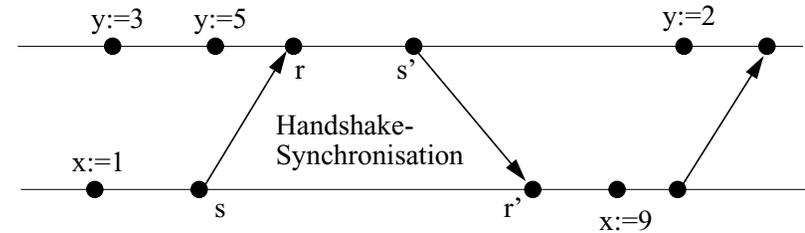
- Zu jeder Präfixberechnung gehört ein konsistenter Zustand.
- Die "echte" Folge globaler Zustände ist ein Pfad durch den Verband (ist aber unbekannt, wenn keine exakte globale Zeit existiert).



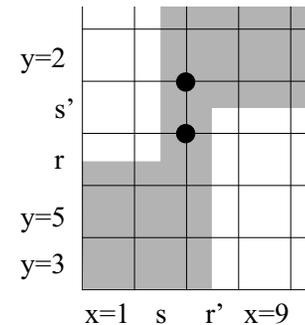
- drei "gleichzeitige" globale Zustände A, B, C
- Frage, durch welchen davon die Berechnung verlief, ist sinnlos!
- Äquivalenzklasse [A, B, C] (alle Zustände mit sieben Ereignissen)
- man weiss nur, dass die Berechnung durch diese Klasse lief
- Konsistente Zustände bilden einen *Verband*
 - früherer, späterer globaler Zustand; abgeschlossen bzgl. "sup" und "inf"
 - visualisiert als eine kompakte Menge (ohne "Löcher")
 - Unterverband des Verbandes *aller* globalen Zustände

Synchronisationsengpässe

- Hier sieht *jeder* kausaltreue Beobachter "kurzzeitig" $y=5, x=1$:



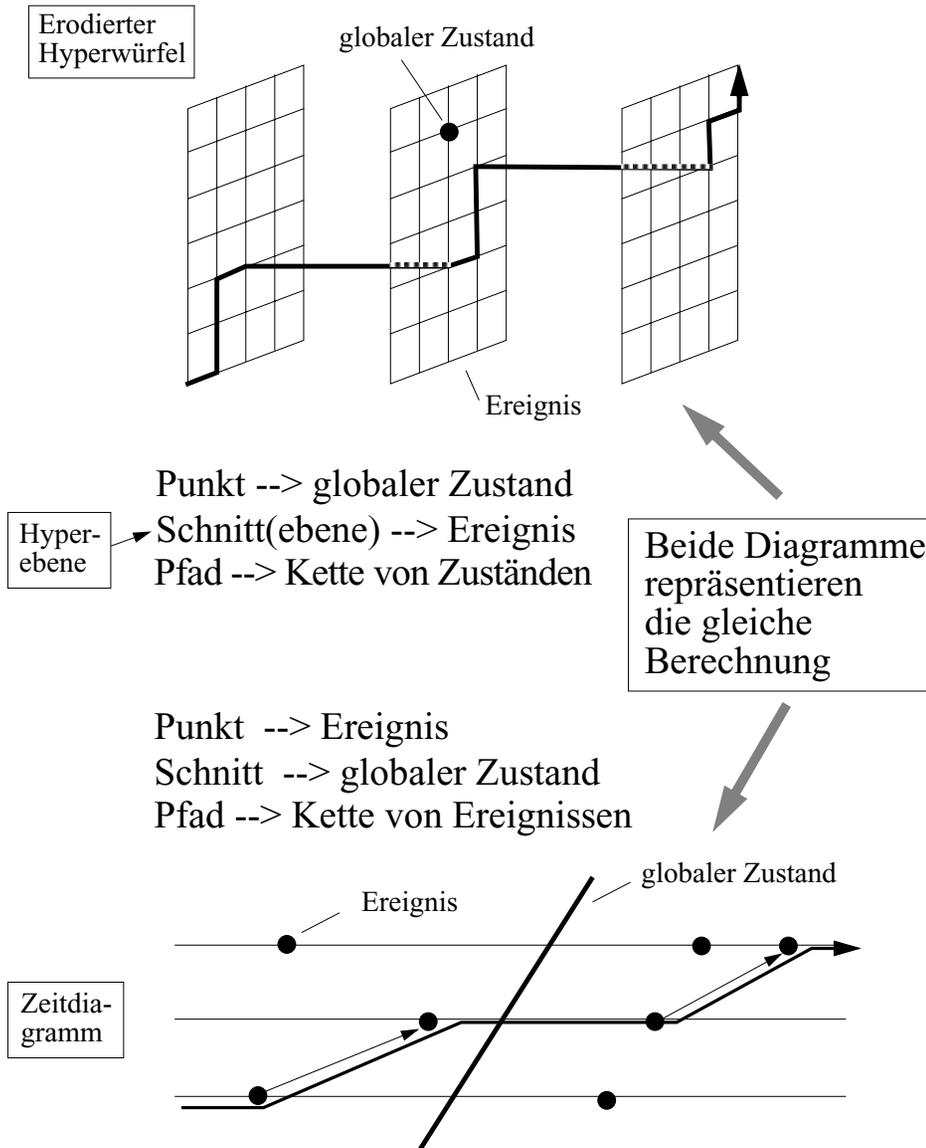
- Der Zustand $y=5, x=1$ wird also auf alle Fälle angenommen; er ist daher "unvermeidlich"
- Verallgemeinerung: "Barrier Synchronisation" (erst wenn alle Prozesse die Barriere erreicht haben, dürfen sie weiterlaufen)



Durch die beiden markierten Zustände muss jeder Beobachter; dort gilt jedoch $x=1, y=5$

- ⇒ Mass für die Parallelität bzw. Unabhängigkeit?
(ein gutes numerisches Mass für "Parallelität" zu finden ist schwierig!)

Dualismus der Diagramme



Konsequenzen

- Prädikate gelten i.a. nur relativ zu Beobachtern, nicht für eine Berechnung als solche!
- Debugging: Nächster Schritt / Zustand nicht eindeutig
- Debugging: "stop when <condition>" i.a. sinnlos
- Polynomielle Zahl von Zuständen
- Exponentielle Zahl von Beobachtungen } → i.a. ziemlich hoffnungslos!
- Ein einzelner Beobachter kann einen Zustand, in dem ein Prädikat gilt, "verpassen"

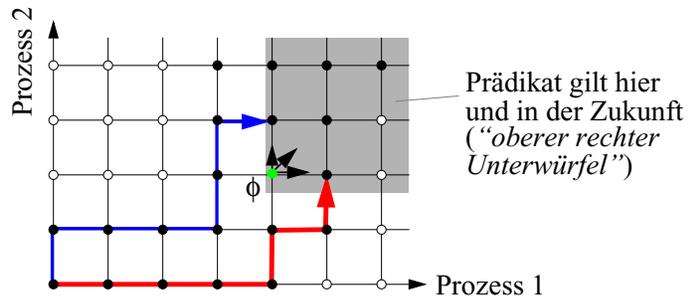
Genauere Spezifikation von Gültigkeit notwendig, z.B.:

- 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

- Informell: Monoton - "einmal wahr, immer wahr"
- Def.: wenn $c1 < c2$, dann $\Phi(c1) \Rightarrow \Phi(c2)$

↑
Halbordnung im Zustandsverband



- Sind *beobachterunabhängig!*
 - jeder Beobachter muss durch den oberen rechten Würfel (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 $B_1 \Phi$ "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) Schnapschuss genügt!
 - wenn der Schnapschussalgorithmus die Gültigkeit von Φ ermittelt, dann gilt Φ "jetzt" tatsächlich
 - wenn Φ "jetzt" gilt, dann meldet dies ein (jetzt gestarteter) Schnapschussalg.
- 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

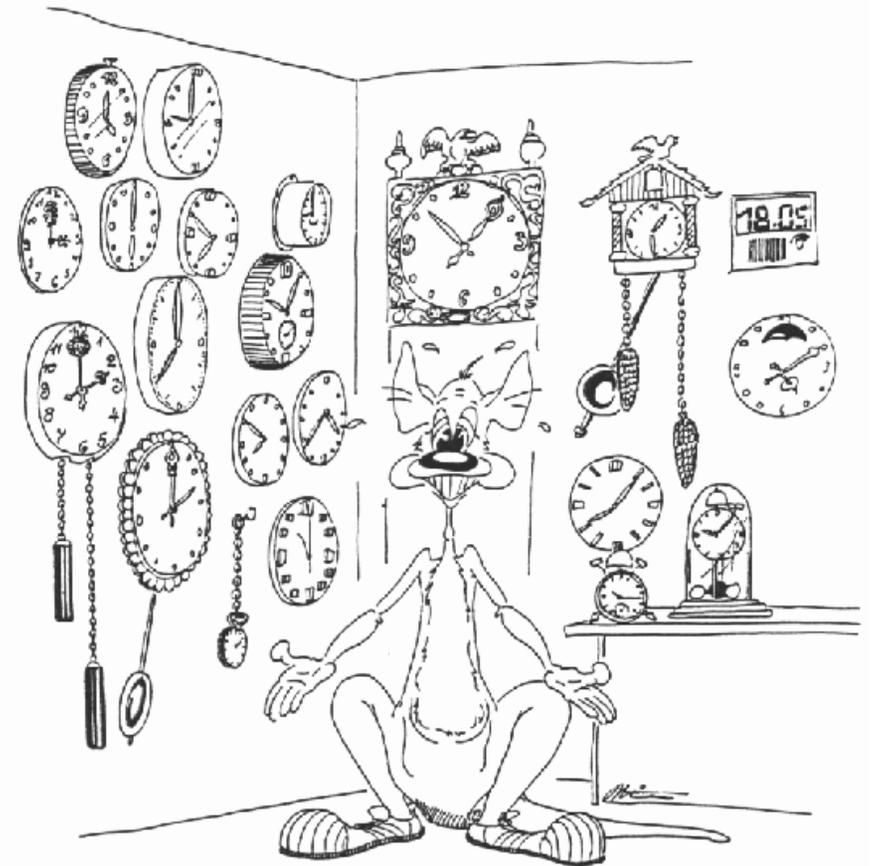
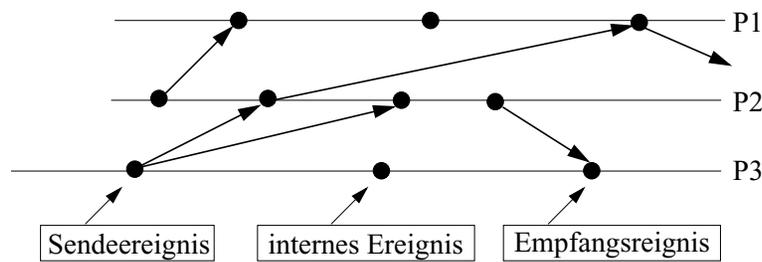


Bild: R. G. Herrtwich, G. Hommel

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 ("dazwischen" egal)

- Gesucht: Abbildung $C: E \rightarrow H$

Clock

↑

"Zeitbereich":
Halbgeordnete Menge
--> "früher", "später"

- Für $e \in E$ heisst $C(e)$ *Zeitstempel* von e

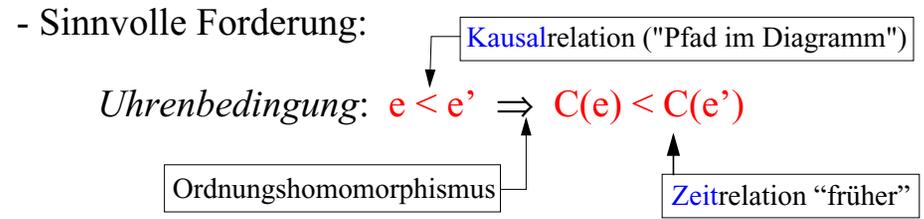
- $C(e)$ bzw. e *früher* als $C(e')$ bzw. e' , wenn $C(e) < C(e')$

- Wie soll H aussehen?

↑

z.B.:

- \mathbb{N} (lineare Ordnung)
- \mathbb{R} (bzw. REAL-Datentyp)
- Potenzmenge von E
- \mathbb{N}^n (d.h. n-dim. Vektoren)

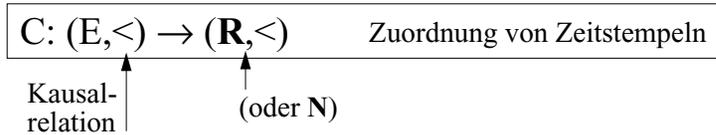


Interpretation ("Zeit ist kausaltreu"):

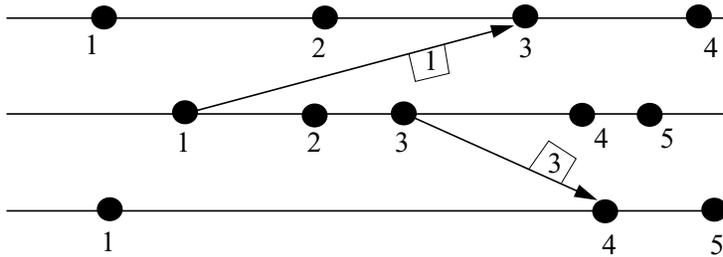
Wenn ein Ereignis e ein anderes Ereignis e' beeinflussen kann, dann muss e einen kleineren Zeitstempel als e' haben

Logische Uhren von Lamport

Commun. ACM 1978:
Time, Clocks, and the Ordering of Events in a Distributed System



$e < e' \Rightarrow C(e) < C(e')$ Uhrenbedingung



Protokoll zur Implementierung der Uhrenbedingung:

- Lokale Uhr (= Zähler) *tickt* "bei" *jedem* Ereignis
- Sendeereignis: Uhrwert mitsenden (*Zeitstempel*)
- Empfangsereignis: $\max(\text{lokale Uhr, Zeitstempel})$

\uparrow zuerst! danach "ticken"

Behauptung:

Protokoll respektiert Uhrenbedingung

Beweis: Kausalitätspfade sind monoton...

Eigenschaften der Lamport-Zeit

- Was bleibt von den Eigenschaften der Realzeit?

- + lin. Ordnung, unbeschränkt
- + respektiert Kausalität
- diskret
- vergeht nicht von alleine

"Critical path" -->
 Parallelitätsmass

- Zeitstempel = Länge der längsten vorangehenden Kette

- Uhrenbedingung \implies

- lokal aufsteigende Zeitstempel
- Sendeereignis kleineren Zeitstempel als Empfang
- $C(a) < C(b) \implies \neg (b < a)$

Zukunft kann Vergangenheit nicht beeinflussen!

Bew.: $b < a \implies C(b) < C(a)$
 $\implies \neg(C(a) < C(b))$

Kausale Unabhängigkeit

- Def. ["kausal unabhängig"]: $a \parallel b \Leftrightarrow \neg(a < b) \wedge \neg(a > b)$

Beachte: \parallel ist *nicht transitiv*! (Gilt eigentlich $a \parallel a$?)

- Behauptung: $C(a) = C(b) \Rightarrow a \parallel b$

Bew.:

$C(a) = C(b) \Rightarrow \neg(C(a) < C(b)) \wedge \neg(C(a) > C(b))$

$\Rightarrow \neg(a < b) \wedge \neg(a > b)$

Verwende $\neg(C(x) < C(y)) \Rightarrow \neg(x < y)$ aus der Uhrenbedingung!

$\Rightarrow a \parallel b$ (Def.)

- Gilt die Umkehrung der Uhrenbedingung?

Nein, $C(e) < C(e') \Rightarrow e < e'$ gilt nicht!

Es gilt nur: $C(e) < C(e') \Rightarrow e < e'$ *oder* $e \parallel e'$

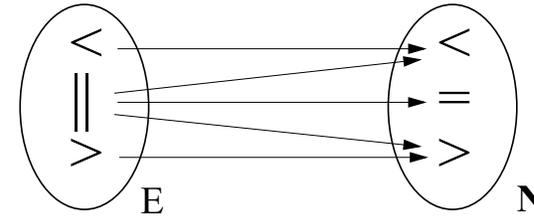
vgl. Beispiel

Das heisst:

Aus den Zeitstempeln lässt sich nicht (immer) schliessen, ob zwei Ereignisse kausal voneinander abhängig sind, oder ob sie unabhängig sind!

\Rightarrow Aber wozu sind solche Zeitstempel dann gut?

Lamport-Zeit: Strukturverlust



- Negation geht verloren

- Ordnungshomomorphismus, jedoch nicht Isomorphism

- E ist Halbordnung, N ist lineare Ordnung

--> zwei kausal unabhängige Ereignisse werden vermöge der Abbildung plötzlich (scheinbar) vergleichbar!

Strukturverlust ist ein wichtiger *Defekt*: Zweck der Zeitstempel ist, auf die Beziehung der Ereignisse zurückzuschliessen! Geht das besser? Anderer "Zeitbereich"?

Lamport-Zeit: Nicht-Injektivität

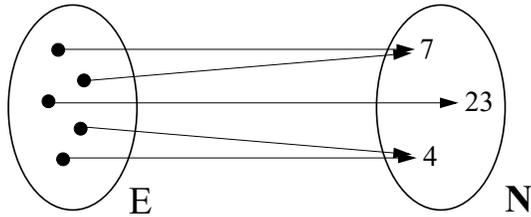


Abbildung ist nicht injektiv

- 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. $(E, <) \rightarrow (N \times N, <)$

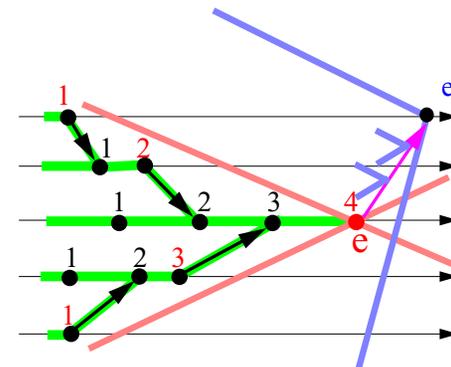
Zu jeder Menge von Ereignissen gibt es nun ein eindeutig "frühestes"!

Vektorzeit

Andere Zeiten, andere Sitten

Quot tempora tot astra
G. Bruno (1548-1600)

"relativistische" Weltansicht, vgl. auch
- Nicolaus Kopernicus
- Galileo Galilei



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 $e \parallel e'$

Zeit := vergangene Zeit

viele Uhren messen die Zeit, indem sie *vergangene* Sekunden zählen

:= Vergangenheit

:= Menge vergangener Ereignisse

vgl. dies mit der *Lamport-Zeit* ("lokal vergangen")

Kausalrelation

$Zeit(e) := \{e' \mid e' \leq e\} = \text{Kegel von } e$

Genauer: Zeitstempel eines Ereignisses

Kann durch lokal späteste Ereignisse repräsentiert werden (linksabgeschlossen)

Hiervon gibt es n Stück (n = Anzahl der Prozesse)

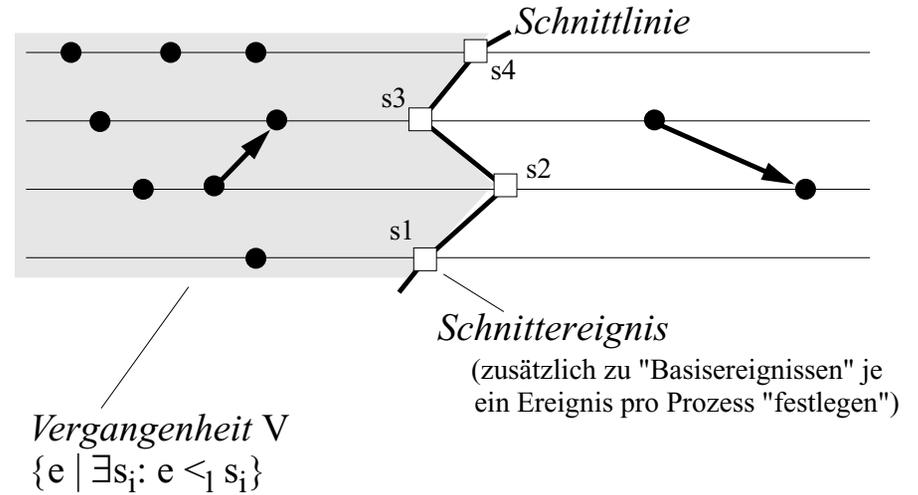
→ *Zeitstempel* ist n-dimensionaler Vektor

→ *Zeit* ist Menge n-dimensionaler Vektoren

→ *Uhr* ist ein array $C[1:n]$

zum Anzeigen von Zeitvektoren

Schnitt und Schnittlinie



- 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.

Konsistente Schnitte

Def. **Schnitt**:

$S \subseteq E$ heisst *Schnitt* von E , falls $e \in S \wedge e' <_1 e \Rightarrow e' \in S$
(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.: $<_1 \subseteq <$

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.)

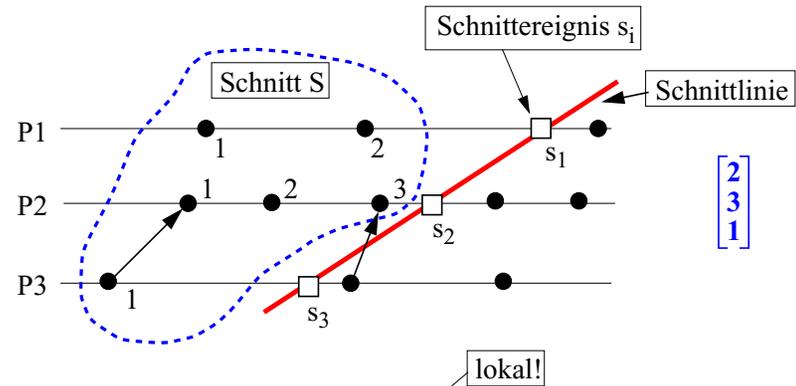
Bew. bereits bekannt: Diagramm auseinanderschneiden und versetzen

Vektorzeitstempel von Schnitten

Zeitstempel $\tau(S)$ eines Schnittes S ist ein Vektor aus \mathbb{N}^n

alle Ereignisse, die links von einer Schnittlinie liegen

Anzahl der Prozesse



Def. $\tau(S)[i] := |\{e \in E_i \mid e <_1 s_i\}| := |S \cap E_i|$

(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 zu *konsistenten* und *inkonsistenten* Schnitten den zugehörigen Zeitvektor definieren!

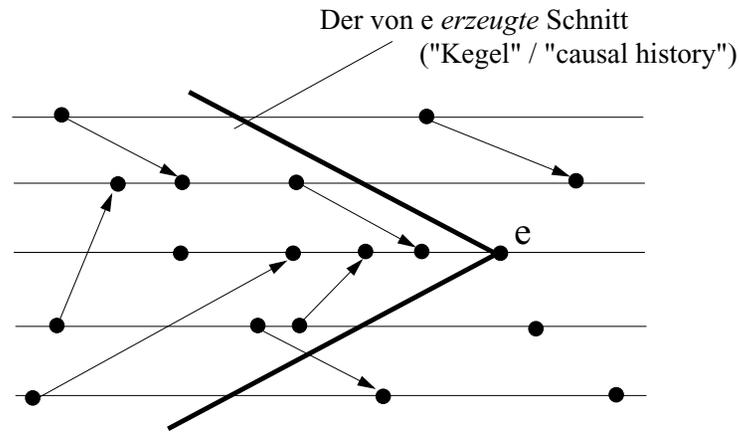
Kausale Vergangenheit eines Ereignisses

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!)

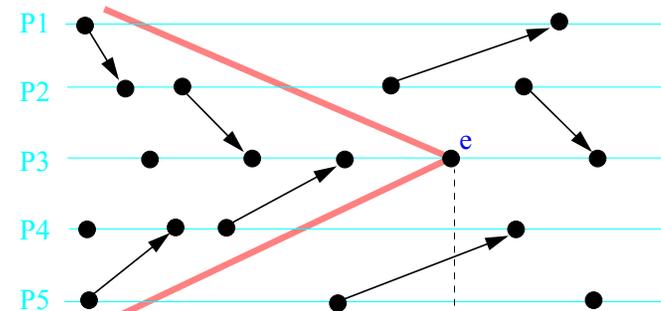
- Repräsentation durch die n lokal am weitesten rechts liegenden Ereignisse

- 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 Kegels



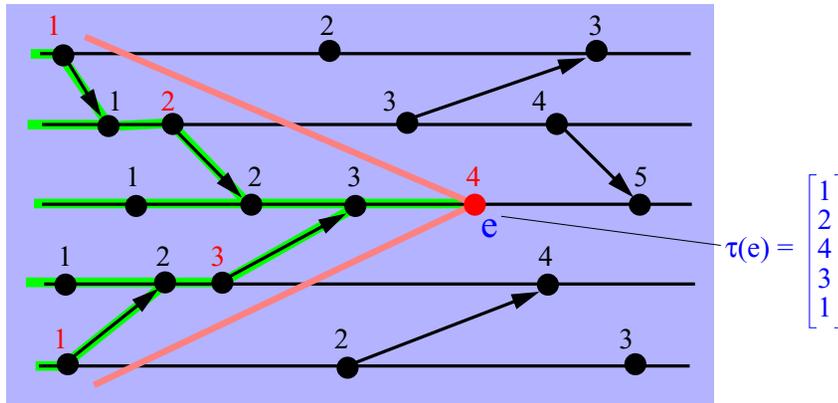
Schnittlinie in der Form eines Kegels (vgl. "Lichtkegel" der relativistischen Physik): $\downarrow e$ ist ein konsistenter Schnitt

$\begin{bmatrix} 1 \\ 2 \\ 4 \\ 3 \\ 1 \end{bmatrix}$

$\tau(e)$: Vektorzeitstempel von e

$$\tau(e) < \tau(e')$$

- Jeder Prozess numeriert seine Ereignisse lokal durch

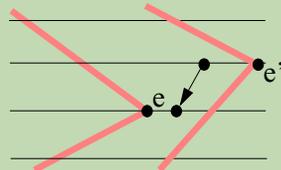


- Vektor $\tau(e)$ repräsentiert gesamte **kausale Vergangenheit** des Ereignisses e

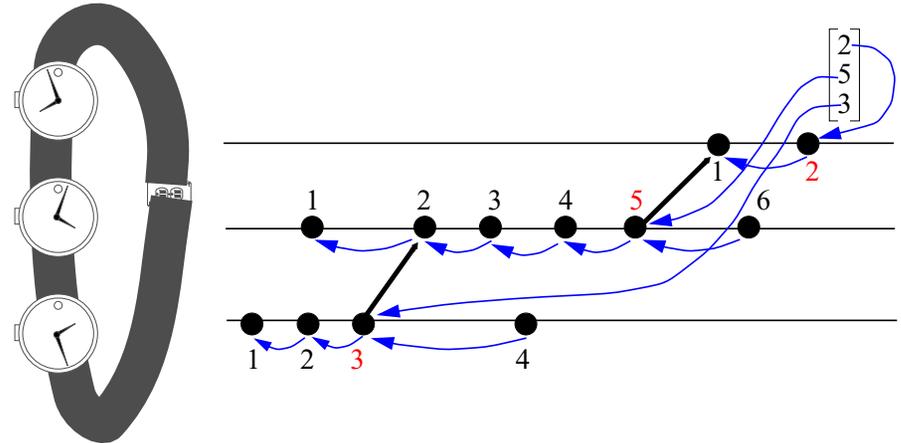
- Veranschaulichung durch einen "Kegel" $\{e' \mid e' \leq e\}$
- Nummer des jeweils lokal letzten kausal vorangehenden Ereignisses steht in der jeweiligen Komponente des Vektors

- 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?

Def. "Zeitstempelarithmetik"

$$\begin{bmatrix} 1 \\ 3 \\ 4 \\ 3 \\ 2 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 7 \\ 4 \\ 6 \\ 2 \end{bmatrix}$$

vergleichbar
(komponentenweise \leq)

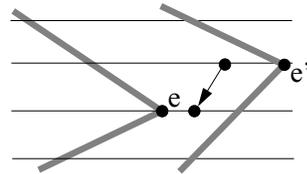
$$\begin{bmatrix} 1 \\ 3 \\ 4 \\ 3 \\ 7 \end{bmatrix} \parallel \begin{bmatrix} 5 \\ 3 \\ 8 \\ 3 \\ 2 \end{bmatrix}$$

"konkurrent"

'<' definiert als " \leq aber \neq "

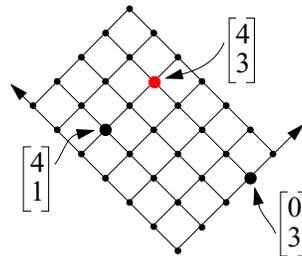
Interpretation von $\tau(e) < \tau(e')$:

- e liegt in der kausalen Vergangenheit von e'
- Kegel von e ist im Kegel von e' enthalten

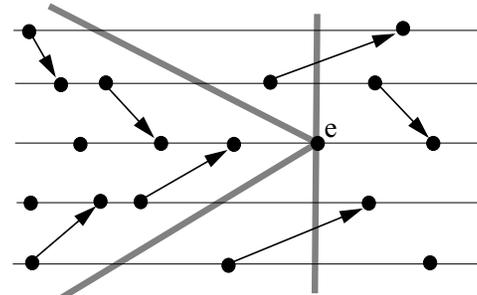


$$\sup \left(\begin{bmatrix} 1 \\ 4 \\ 2 \\ 3 \\ 7 \end{bmatrix}, \begin{bmatrix} 8 \\ 3 \\ 4 \\ 3 \\ 2 \end{bmatrix} \right) = \begin{bmatrix} 8 \\ 4 \\ 4 \\ 3 \\ 7 \end{bmatrix}$$

sup = komponentweises Maximum



Vektorzeit und ideale Beobachter



$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \dots \begin{bmatrix} 2 \\ 4 \\ 5 \\ 4 \\ 3 \end{bmatrix}$$

Wahrnehmungen des
idealen Beobachters

$$\tau(e) = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 3 \\ 1 \end{bmatrix}$$

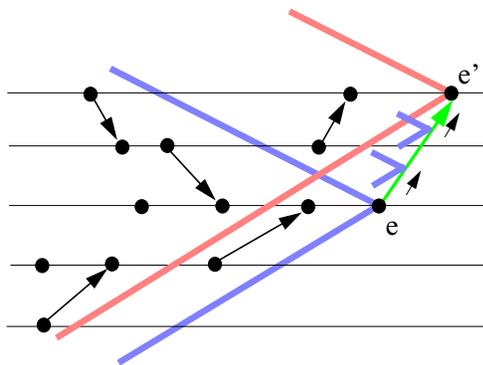
$$\text{id}(e) = \begin{bmatrix} 1 \\ 3 \\ 4 \\ 3 \\ 2 \end{bmatrix}$$

- Numeriere Ereignisse lokal
- Idealisierter Beobachter nimmt "Ticks" sofort wahr
- Geeignete Datenstruktur hierfür: Vektor / array
- Für *jeden* Beobachter gilt stets: $\tau(e) \leq \text{id}(e)$ ($\forall e$)
(komponentenweise ' \leq '; idealer Beobachter sieht stets die gesamte kausale Vergangenheit und evtl. einige weitere Ereignisse)
- $\tau(e)$ = Infimum aller idealen Sichten $\text{id}(e)$
- Beachte: $\text{id}(e)$ 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 \Rightarrow Zeitvektor des Sendeereignisses
- Bei Empfang einer Nachricht:
 - Vereinigung der Kegel \leftarrow Wissen über vergangene Ereignisse vereinigen
 - \Rightarrow Supremum der Vektoren



Mitschleppen des Kegels des Sendeereignisses und Vereinigung mit dem Kegel des Empfangereignisses

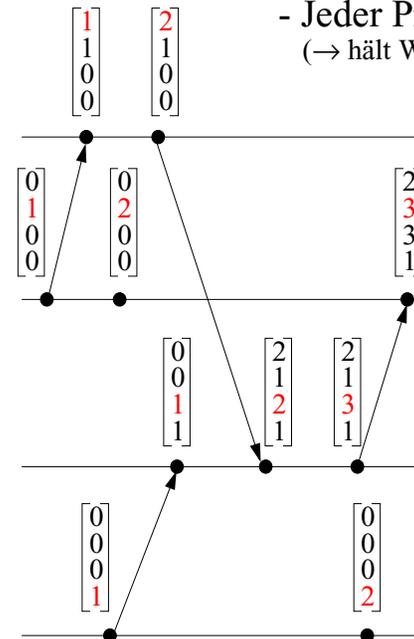
\rightarrow "induktiv": ein Ereignis hat ein "vollständiges Wissen" über alle seine vergangenen Ereignisse

Propagieren des Zeitwissens

Andere Zeiten, andere Sitten

(\rightarrow Implementation der Vektorzeit)

- Jeder Prozess besitzt eine *Vektoruhr*
(\rightarrow 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 \downarrow 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
Lampport-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 $\tau(e)$

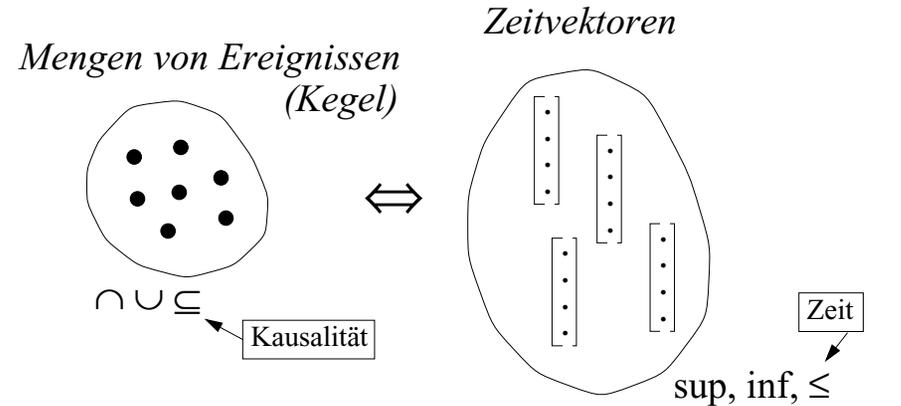
Irreflexivität folgt aus Injektivität von τ

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



Mengentheoretische
Operationen

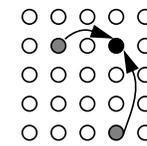
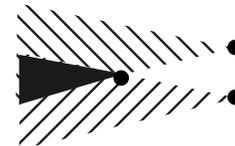
\Leftrightarrow

Algebraische Operationen
(\rightarrow "rechnen")

Verbandsstruktur
auf 2^E (Ideale)

\Leftrightarrow

Produktverband auf \mathbb{N}^n



Ordnungstheoretische
Eigenschaften

\Leftrightarrow

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

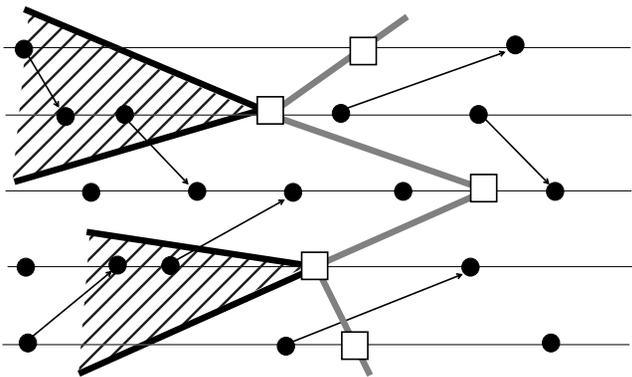
- Implementierung konsistenter Schnappschüsse

- Menge lokaler Schnappschüsse mit paarweise konkurrenten Ereignissen

- Realisierung von kausaltreuen Beobachtern

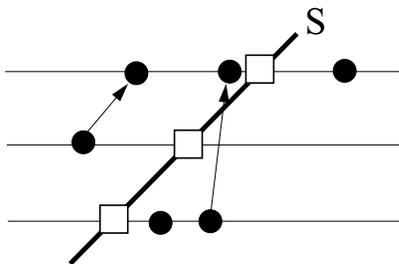
- Causal broadcast
- Causal order

Zeitvektoren der Schnittereignisse



- Betrachte nun die Zeitvektoren der Schnittereignisse s_i eines Schnittes S.
- Liegen die Kegel $\downarrow s_i$ ganz im Schnitt? (Gilt $\downarrow s_i \subseteq S$?)

Nicht immer:



Konsistente Hülle eines Schnittes

Für einen Schnitt S definiere $S^* := \downarrow s_1 \cup \downarrow s_2 \cup \dots \cup \downarrow s_n$
(Vereinigung aller Schnittereigniskegel)

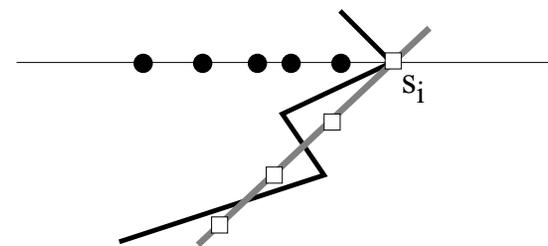
Beh.: Für jeden Schnitt S ist S^*

- a) eindeutig ← klar nach Konstruktion
- b) ein Schnitt ← - jeder Kegel ist ein kons. Schnitt (Übung)
- c) konsistent ← - kons. Schnitte sind bzgl. Vereinigung abgeschlossen (Verband --> Übung)

Beh.: Es gilt $S \subseteq S^*$ (d.h. S^* ist später/gleich S)

Bew.: $S \cap E_i \subseteq \downarrow s_i$

(Alle Ereignisse des Schnittes auf dem i-ten Prozess werden durch $\downarrow s_i$ abgedeckt)



Fragen:

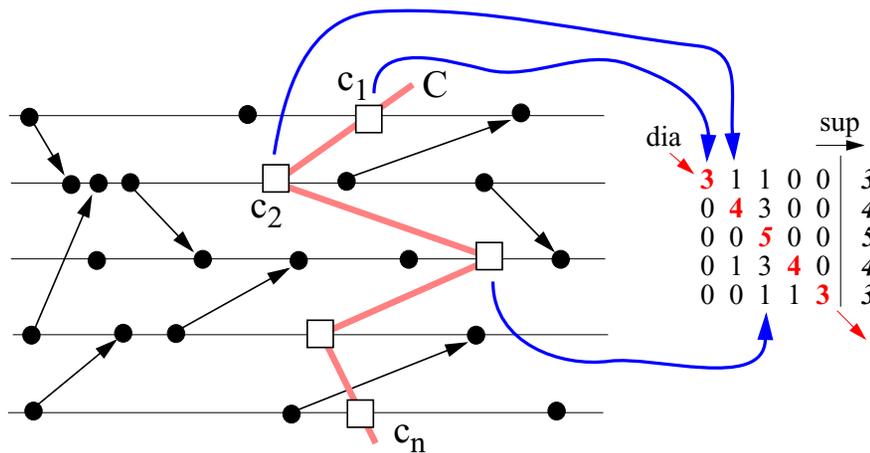
- a) Gibt es u.U. einen Schnitt S' mit $S \subset S' \subset S^*$?
- b) ... konsistenten Schnitt...?

Schnittmatrix

- Schnittmatrix $\$$ eines Schnittes C (mit Schnittereign. c_i):

$$\$(C) := (\tau(c_1), \tau(c_2), \dots, \tau(c_n))$$

d.h. Schnittereignisvektoren c_i als Spaltenvektoren



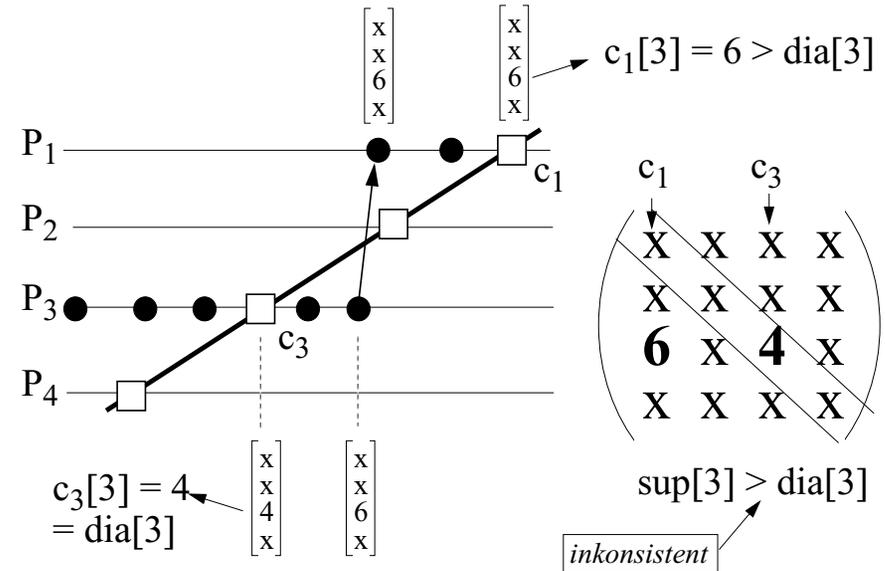
Frage: Kann man an den Schnittmatrizen etwas über die Schnitte erkennen? (z.B.: ob später, früher; ob konsistent...)

$$C \text{ konsistent} \Leftrightarrow \text{dia}(\$) = \text{sup}(\$)$$

Diagonalvektor Zeilenmaximum

(d.h. Maximum einer Zeile ist 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

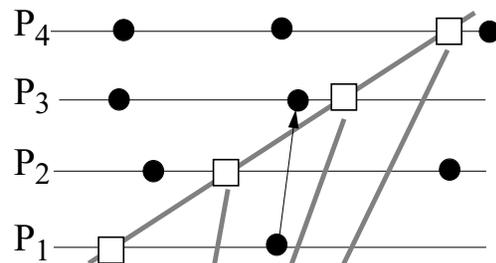
\Leftrightarrow

[Generalisierung über alle Indizes $i \neq j$]

Der Schnitt ist inkonsistent

Implementierung konsistenter Schnappschüsse mit Vektorzeit

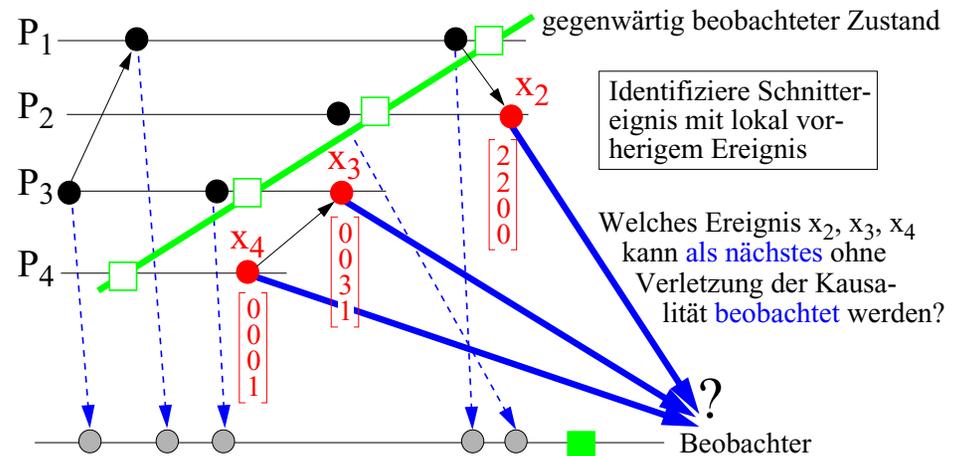
Ein erster Ansatz: Alle Prozesse auffordern, ihren lokalen Zustand zu senden und testen, ob konsistent:



$$\begin{pmatrix} X & \cdot & \cdot & \cdot \\ \cdot & X & \cdot & \cdot \\ \cdot & \cdot & X & \cdot \\ \cdot & \cdot & \cdot & X \end{pmatrix}$$

Dieses solange wiederholen, bis man einmal Glück hat...

Realisierung kausaltreuer Beobachter



$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

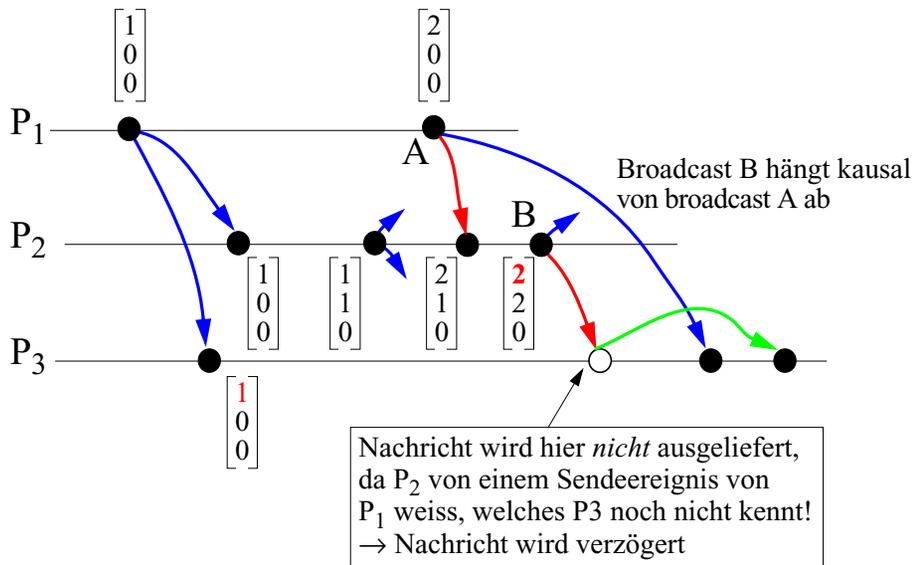
- Welche *Spalte* kann *ausgetauscht* werden? (x_2, x_4 , aber nicht x_3)
- Beobachter merkt sich $dia(\$)$; es muss $Zeitstempel \leq dia(\$)$ sein (ausgenommen die Diagonalkomponente)

- Strategie: $dia(\$) = sup(\$)$ → stets konsistent halten!
- Beobachter benötigt *nur* den *Diagonalvektor*, keine Matrix, um momentanen Zustand zu identifizieren

Prinzip: Verwende Vektorzeit um (indirektes) *Wissen* über "kausal frühere" Sendeereignisse zu kodieren:

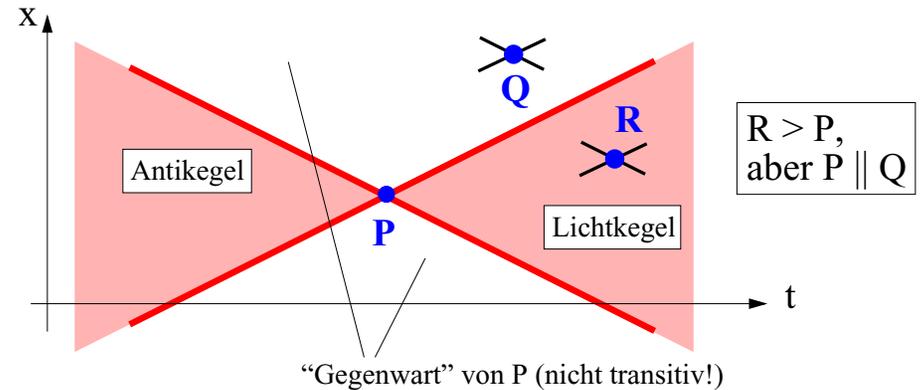
"Dieses Ereignis hängt von einem anderen ab, das ich eigentlich erhalten haben müsste; also warte ich das andere erst ab..."

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 relevante Ereignisse
- Vektoren sind Spezialfälle der Matrixuhren
 - alle Elemente einer Spalte identisch → Reduktion zu einem Vektor
- **Verallgemeinerung** des Sequenznummervorgangs zur Implementierung von **FIFO** bei Nicht-FIFO-Kanälen

Vektorzeit und Minkowski-Raumzeit



Raumzeit

Halbordnung

2-dimensionale Kegel bilden **Verband** (bzgl. Schnitt)

Lorentz-Transformation lässt **Lichtkegel invariant**

Raumzeitkoordinaten ermöglichen **Test**, ob potentiell **kausal abhängig**:
 Mit $u = (x_1, t_1)$, $v = (x_2, t_2)$ prüfe
 $c^2(t_2 - t_1)^2 - (x_2 - x_1)^2 \geq 0$

Vektorzeit

Halbordnung

Zeitvektoren bilden **Verband** (sup)

Gummiband-Transformation lässt **Kausalrelation invariant**

Zeitvektoren ermöglichen einfachen **Test**, ob potentiell **kausal abhängig**:
 (prüfe, ob in allen Komponenten kleiner)

Denkübung:

Vektoren sind relativ aufwendig: Geht kausaler Broadcast, causal order, kausaltreue Beobachtung auch mit **weniger aufwendigen** Datenstrukturen?

Resümee: Konzepte der Vorlesung

- Prinzipielle Phänomene und Begriffe herausarbeiten
 - Kausalität, Konsistenz, verteilte Berechnung, safety und liveness,...
- Geeignete Modelle und Abstraktionen entwickeln
 - z.B. Zeitdiagramme, Atommodell, Zustandsgitter, Gummibandtransform.
- Problemlösungs-, Analyse- und Verifikationstechniken
 - z.B. Beweis über Invarianten
- Techniken, Einsichten, Zusammenhänge, ...
 - Komplexitätsanalyse
 - Transformationen zwischen Problemklassen
 - Problemverständnis von einem höheren Standpunkt



Resümee: Themen der Vorlesung

- Beispiele für verteilte Berechnungen und Algorithmen
 - verteilte ggT-Berechnung
 - verteiltes Lösen von Zahlenrätseln
 - verteilte Approximation
- Grundalgorithmen
 - Flooding
 - Echo-Algorithmus (Wellenalgorithmus, Spannbaum)
- Verteilte Terminierung
 - Doppelzählverfahren
 - Zeitonenverfahren
 - für synchrone Kommunikation: DFG-Verfahren
 - Kreditmethode

Grundphänomen "inkonsistenter Sicht"; nur problemspezifische Lösungen dafür
- Wechselseitiger Ausschluss
 - Grundprinzipien
 - Maekawa
 - Token-basierte Verfahren

Synchronisation in vert. Sys. (viele wollen, einer darf; Sicherheit, Deadlockfreiheit, Fairness)
- Election
 - Chang/Roberts-Verfahren (Ring); bidirektionale Varianten
 - Hirschberg/Sinclair und Peterson's Algorithmen: $O(n \log n)$ worst case
 - Election auf Bäumen
 - untere Schranke $O(e)$ für Nachrichtenkomplx. 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,...
- Sequentielle Traversierungsverfahren
 - Methode von Tarry (1895)
- Parallele Traversierungsverfahren
 - Verteilen von Information ("flooding"), Echo-Algorithmus

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
-

Ende

ETH Life, erschienen am 29.10.2001
www.ethlife.ethz.ch/tages/show/0,1046,0-8-1385,00.html

Er ist der neue "Mister NET": Christian Sengstag... ETH Life befragte ihn zu seinen Plänen und Zielen.

...wagen wir einen Ausblick: Wie wird an der ETH in zehn Jahren gelehrt werden?

Sengstag: Der grösste Unterschied wird sein: Es gibt keine klassischen Vorlesungen mehr.

Warum?

Sengstag: Weil sie als ineffizient erkannt werden. Wer noch auf diese Art doziert, sieht sich je länger je mehr der Konkurrenz von guten Lernprogrammen ausgesetzt. Aus heute noch frontal Dozierenden werden Diskussionsleiter und Moderatoren für gezielte Problemlösungen. Zudem werden gedruckte Lehrbücher zur Seltenheit werden: alles muss und wird online verfügbar sein.
