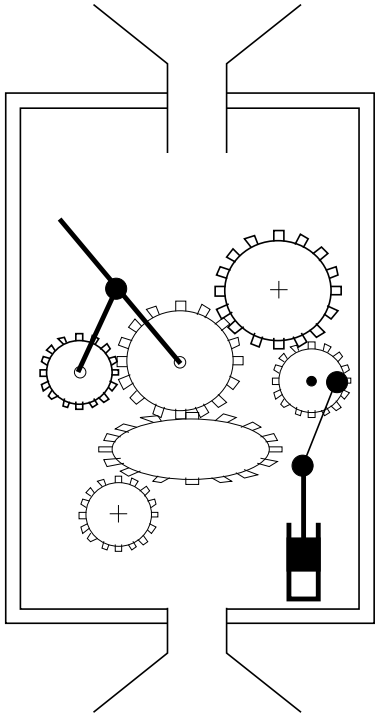
Local Reference Counting (LRC)

Y. Ichisugi, A. Yonezawa: Distributed Garbage Collection Using Group Reference Counting, TR 90-014, Univ. of Tokyo

M. Rudalics: Implementation of Distributed Reference Counts, Internal Report, J. Kepler University, Linz

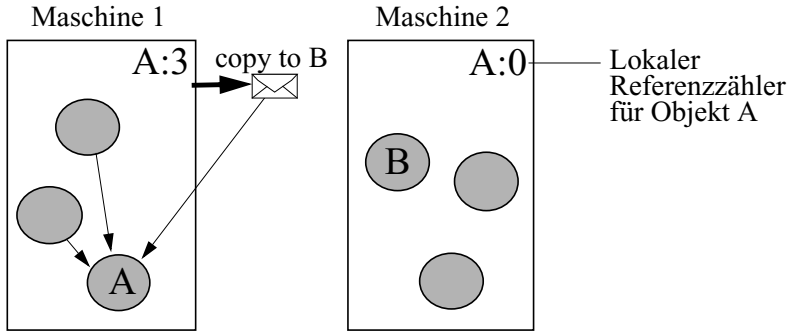
J. Piquer: *Indirect Reference Counting,* Proc. PARLE 91, 150-165

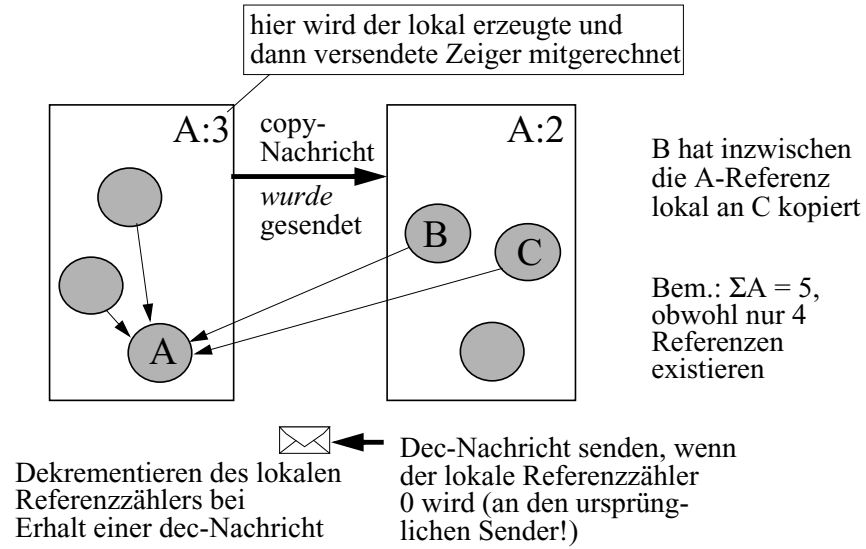


E.W. Dijkstra, C.S. Scholten: Termination Detection for Diffusing Computations. Inf. Proc. Lett. 11 (1980), 1-4

LRC-Garbage-Collection-Verfahren

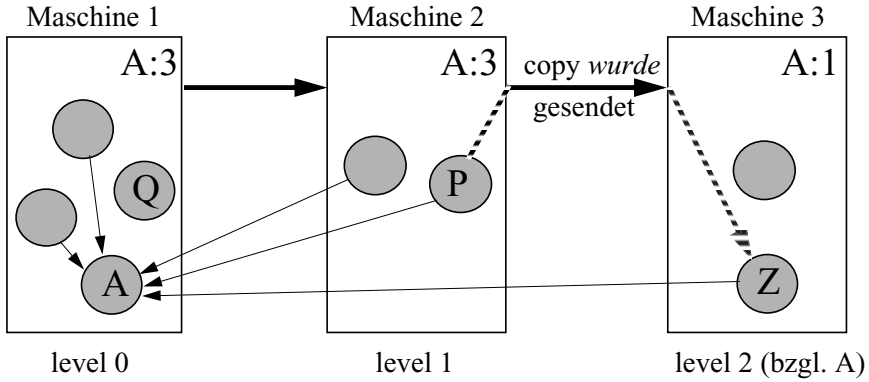
- Pro *Maschine* wird (potentiell) für jedes Objekt ein Referenzzähler gehalten, z.B. für Objekt A:





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

LRC: Weitervererben von Referenzen



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

- Korrektheit (safety): Referenzbaum

  Lokaler Referenzzähler auf level i+1 > 0
  --> lokaler Referenzzähler auf level i > 0

  ==> Falls eine Referenz existiert, dann ist der Referenzzähler auf level 0 > 0

  - wieso eigentlich *Baum*?
  - wie genau ist "level" definiert?

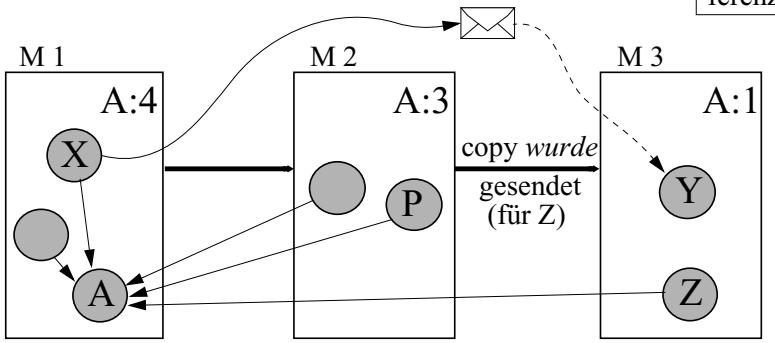
  ==> "Garbage-Kriterium": Referenzzähler auf level 0 = 0

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

Der "Remote-ref"-Baum

- Was geschieht, wenn X auf Maschine 1 eine copy-Nachricht mit einer A-Referenz an Y auf Maschine 3 schickt?
  - Maschine 1 erhöht ihren lokalen A-Zähler beim Senden
  - Maschine 3 erhöht ihren lokalen A-Zähler beim Empfang

  die schon eine A-Referenz hat



- Aber: Wann und an wen (hier: Maschine 1 und/oder 2) soll 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?)

LRC: Formale Beschreibung

- Ergänzen existierender atomarer Aktionen bzw. zusätzliche Aktionen:

C_p: {p ist erreichbar und hat eine r-Referenz}
send copy(r) **to** q;
$LRC_p(r) := LRC_p(r) + 1$

atomare Aktion: Änderung von LRC "gleichzeitig" mit dem Versenden der copy-Nachricht

R_p: {Bei Empfang einer copy(r)-Nachricht von q}
Installiere die r-Referenz;
if $LRC_p(r) = 0$
then {$LRC_p(r) := 1$; $FIRST_p(r) := q$}
else {$LRC_p(r) := LRC_p(r)+1$; **send** dec(r) **to** q}
fi

r-Referenz existierte schon bzw. p befindet sich schon im Referenzbaum

D_p: {p hat eine r-Referenz}
Lösche die r-Referenz;
$LRC_p(r) := LRC_p(r) - 1$;
if $LRC_p(r) = 0$ **then call** collect(r) **fi**

X_p: {Bei Empfang einer dec(r)-Nachricht}
$LRC_p(r) := LRC_p(r) - 1$;
if $LRC_p(r) = 0$ **then call** collect(r) **fi**

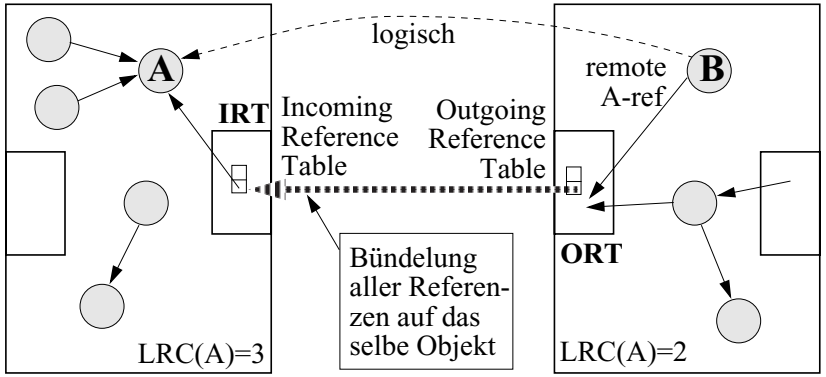
ggf. rekursives Freigeben!

proc collect(r):
if $FIRST_p(r) = \varnothing$ **then** "*r is garbage*"
else send dec(r) **to** $FIRST_p(r)$; **fi**

r wurde lokal erzeugt

- beim *lokalen* Kopieren von r wird lediglich $LRC_p(r)$ erhöht
- wieso wird eine dec-Nachricht gesendet, wenn schon eine r-Referenz existiert; könnte man darauf u.U. verzichten?

Implementierungssicht: Remote-reference-Tabellen



- ORT: enthält Schattenobjekte als lokale Stellvertreter ("Proxy") für alle entfernten Objekte
  - beachte: hier ist Σ LRC(A) = 5 = Anzahl A-Referenzen (inkl. bei IRT!),

- IRT / ORT: Die Einträge bzgl. eines bestimmten Objektes (z.B. A) lassen sich als ein *einziges* Objekt, *verteilt* auf mehrere Maschinen auffassen

z.B. weil vom *lokalen Collector* als Garbage erkannt

Idee: Wenn ein Teilobjekt in ORT *gelöscht* wird, dann wird das entsprechende Gegenstück in der IRT gelöscht
(falls es sonst keine Teile in anderen ORT gibt)

--> ORT muss dazu eine Nachricht an IRT senden

- Remote-Referenzen sind ggf. *mehrfach indirekt*
  - u.U mehrfache Adressumsetzung bei *Zugriff* über eine Remote-Referenz, falls der Zugriff tatsächlich entlang der Referenzkette erfogt

LRC: Eigenschaften

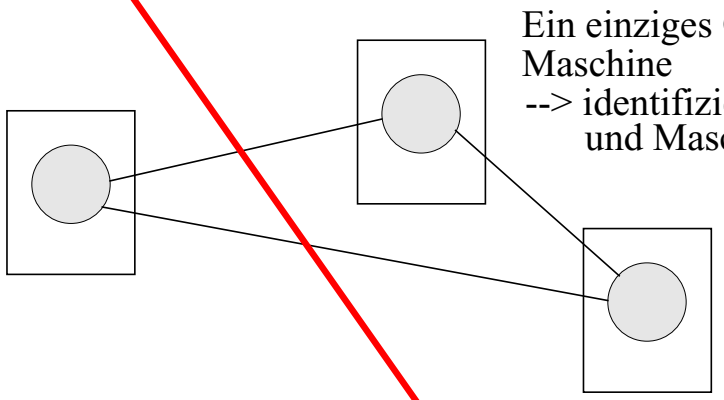
- Vergleich zu Verfahren von Lermen/Maurer, Rudalics etc:
  - keine Zusatznachricht bei copy
  - kein Verzögern von copy
  - oft: keine Zusatznachricht bei delete (gelegentlich: Abbau des Referenzbaumes)
  - kein Verzögern bei delete
  - FIFO nicht notwendig
  - Gesamtzahl der Nachrichten: genausoviele dec wie copy (wieso?)

Generell gilt: zyklischer Garbage zwischen verschiedenen Objekten lässt sich mit Referenzzählern nicht erkennen!

- Vergleich zu WRC:
  - Zähler einfacher handzuhaben als die Akkumulation von beliebig kleinen Gewichtsfragmenten
  - keine Komplikation bei RW=1
- Nachteil (gegenüber anderen Referenzzähler-Methoden):
  - Objekte besitzen i.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)

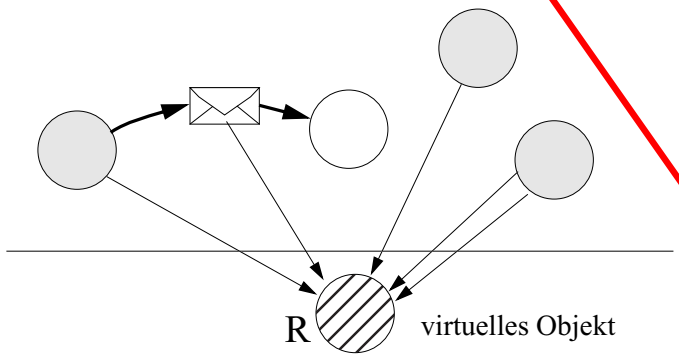
Transformation von LRC in einen Terminierungserkennungs-Algorithmus

Konzeptionelle Sicht:



Ein einziges Objekt pro Maschine
--> identifiziere Objekt und Maschine

- Betrachte zunächst die Variante, wo ein Objekt u.U. mehrere R-Referenzen besitzen kann
- Entspricht dem Fall, dass ein aktiver Prozess noch eine weitere Nachricht empfängt



Die Transformation

1) Identifziere Maschine / Prozess / Objekt

2) Füge ein einziges zusätzliches (virt.) Objekt "R" hinzu

3) *Senden einer Nachricht* --> Kopieren der R-Referenz
   --> *inkrementiere den (!) lokalen Zähler ("für R")*

4) *Empfang einer Nachricht*
   --> *inkrementiere den lokalen Zähler*

auch wenn schon eine lokale R-Referenz existiert...!?

5) *Passiv werden* --> Lösche alle (!) R-Referenzen
   --> *dekrementiere den lokalen Zähler entsprechend oft*

wie oft genau?

6) *Bei Empfang einer dec-Nachricht*
   --> *dekrementiere den lokalen Zähler*

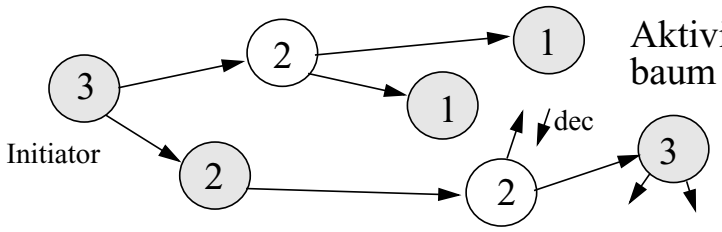
müsste man sich also alle merken

7) *Wenn der lokale Zähler 0 wird*
   --> *sende dec-Nachricht an alle (!) Sender*

8) *Terminiert, wenn der Zähler* des Erzeugers von R (also *des Initiators der Berechnung) 0 wird*

- Optimierung: Behalte nur *erste* R-Referenz, alle anderen R-Referenzen werden sofort wieder gelöscht

nur einen merken!



Das Resultat der Transformation

S_p: {Zustand = *aktiv*}
send <message> **to** ... ; $LRC_p := LRC_p + 1$;

R_p: {Eine Nachricht von q ist angekommen}
receive ...;
if $LRC_p = 0$
then {$LRC_p := 1$; $FIRST_p := q$; Zustand := *aktiv*;}
else {**send** <dec> **to** q; **if** Zustand = *passiv*
then {Zustand := *aktiv*;
$LRC_p := LRC_p + 1$;}}

I_p: {Zustand = aktiv}
Zustand := *passiv*; $LRC_p := LRC_p - 1$;
if $LRC_p = 0$ **then send** <dec> **to** $FIRST_p$;

X_p: {Bei Empfang einer <dec>-Nachricht}
receive <dec>; $LRC_p := LRC_p - 1$;
if $LRC_p = 0$ **then send** <dec> **to** $FIRST_p$;

- vgl. dies mit "indirekten Acknowledgements" bzw. dem Echo-Algorithmus
- dies ist der *Terminierungserkennungsalgorithmus von Dijkstra und Scholten* (in etwas anderer Darstellung als im Originalartikel, dessen Studium sich lohnt!): E.W. Dijkstra, C.S. Scholten: Termination Detection for Diffusing Computations. Inf. Proc. Lett. 11 (1980), 1-4
- Voraussetzung: Es gibt einen einzigen initial aktiven Prozess ("environment")
- die letzte Zeile von I_p und X_p wird für das "environment" ersetzt durch:
  if $LRC_p = 0$ **then** *"terminated"*;
- die letzte if-Bedingung in der Aktion R_p ergibt sich aus der Überlegung, dass ein bereits aktiver Prozess nicht "noch aktiver" wird

Verteilte Berechnungen

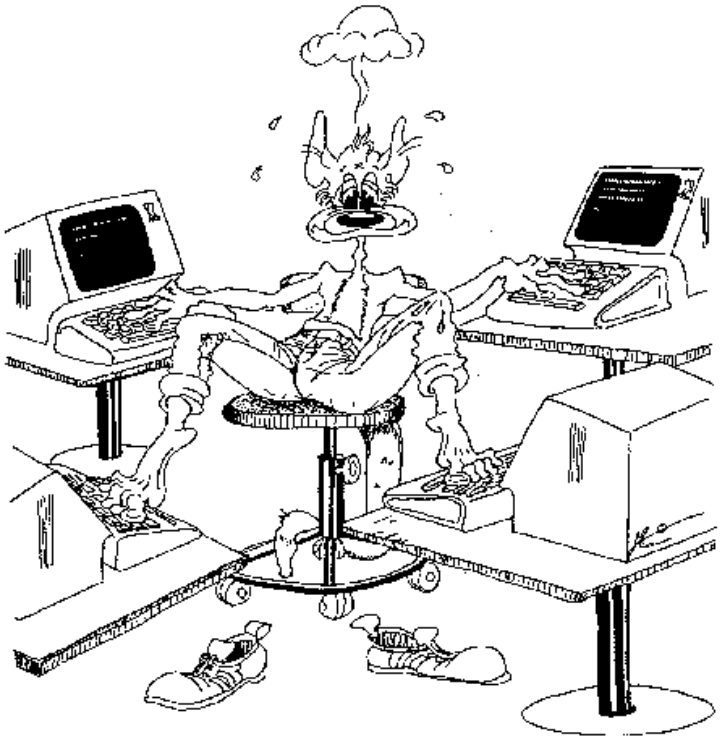


Bild: R. G. Herrtwich, G. Hommel

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 ==> 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!

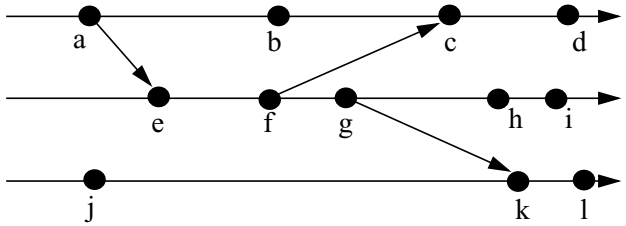
Gummiband-transformation

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 "Querpfeile")



  - 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: $\mathbf{x} < \mathbf{z} \wedge \mathbf{z} < \mathbf{y}$

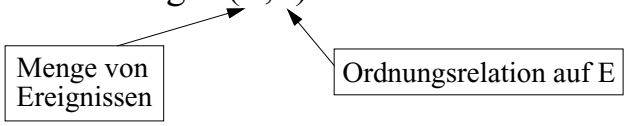
- Wieso ist das eine partielle *Ordnung*?
  - d.h. wieso ist die Relation "gerichtet" und zyklenfrei?
  - oder muss man das (zur Def. von "Berechnung") axiomatisch fordern?

- Relation wird oft als "happened before" bezeichnet
  - eingeführt von Lamport (1978)
  - aber Vorsicht: damit ist nicht direkt eine "zeitliche" Aussage getroffen!

- Interpretationen von e < e':
  - es gibt eine Kausalkette von e nach e'
  - e kann e' beeinflussen
  - e' hängt (potentiell) von e ab
  - e' "weiss" / kennt e

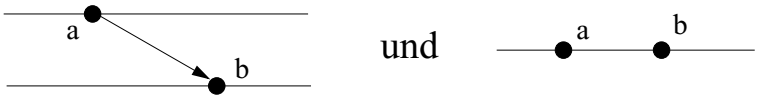
Erster Definitionsversuch

- Verteilte Berechnung = (E,<)



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

Beachte:



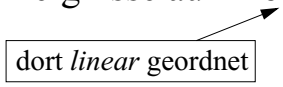
wären identisch!

==> *Prozesse* einführen

- Was sind Prozesse (formal / abstrakt)?

  - Partitioniere E in disjunkte Mengen $E_1,...E_n$

  - Interpretation: E_i = Ereignisse auf Prozess P_i

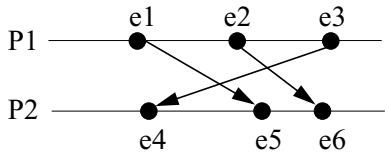


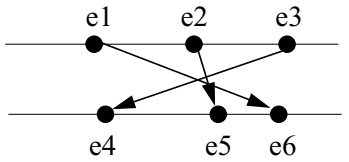
Zweiter Definitionsversuch

- Verteilte Berechnung = $(E_1,...E_n,<)$ mit:

  - alle E_i paarweise disjunkt
  - < ist (irreflexive) Halbordnung auf $E_1 \cup ... \cup E_n$
  - < ist lineare Ordnung auf jedem E_i

- Noch nicht befriedigend:





	e1	e2	e3	e4	e5	e6
e1		x	x	x	x	x
e2			x	x	x	x
e3				x	x	x
e4					x	x
e5						x
e6						

"Spalte" < "Zeile"

gleiche Kausalrelation
==> nicht unterscheidbar, obwohl wesentlich verschiedene Diagramme (also verschiedene Berechnungen)!

==> Nachrichten einführen
(eindeutige Zuordnung zusammengehöriger send-/receive-Ereignisse)

Dritter Definitionsversuch

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

  was wäre bei syn. Komm. anders?

  1) [*Ereignisse*] Alle E_i sind paarweise disjunkt

  2) [*Nachrichten*] Sei $E = E_1 \cup ... \cup E_n$
  Für $\Gamma \subseteq S \times R$ mit $S,R \subseteq E$ und $S \cap R = \emptyset$ 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

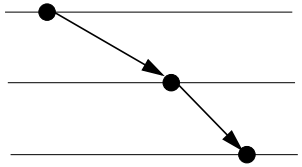
  4) $(s,r) \in \Gamma \Longrightarrow s < r$

  5) $<$ ist eine irreflexive Halbordnung auf E

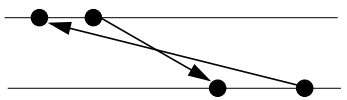
  6) $<$ ist die kleinste Relation, die 3) - 5) erfüllt
  (d.h.: es stehen keine weiteren als die dadurch festgelegten Ereignisse in <-Relation zueinander)

  [*Kausalrelation*]

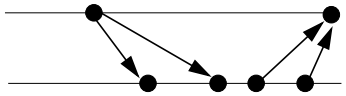
- "Gegenbeispiele":



nicht möglich wegen (2)



nicht möglich wegen (5)



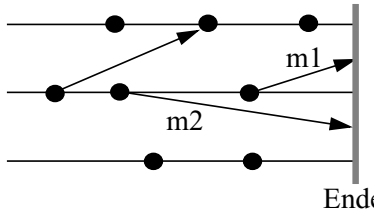
nicht möglich wegen (2)

Bemerkungen zur Definition

- Die $s \in S$ heissen *Sende-*, die $r \in R$ *Empfangsereignisse*
  - die anderen Ereignisse werden *interne Ereignisse* genannt

- *Darstellung* einer so definierten verteilten Berechnung ist mit *Zeitdiagrammen* möglich
  - E_i als Prozesslinie mit Ereignissen
  - Γ als Pfeile
  - $<$ garantiert Zyklenfreiheit

- Definition erlaubt (wegen "höchstens" in Punkt 2) die Modellierung von *In-transit-Nachrichten*:



  - die beiden Nachrichten m1 und m2 sind nie angekommen
  - mögliche Interpretationen:
    - sind verlorengegangen
    - waren bei Berechnungsende noch unterwegs

- Kann man Einzelprozesse als *Folgen von Ereignissen* auffassen?

- Ergibt sich bei einer vert. Berechnung aus einem *einzigen* Prozess eine (wie üblich definierte) *sequentielle* Berechnung?

- Wieso ist diese Definition so "statisch" (statt einer Definition über *Folgen* von Ereignissen oder Zuständen)?

  Vorläufige Antwort:
  - wir haben den Begriff "globaler Zustand" noch nicht definiert
  - Folgen sind nicht eindeutig bestimmt

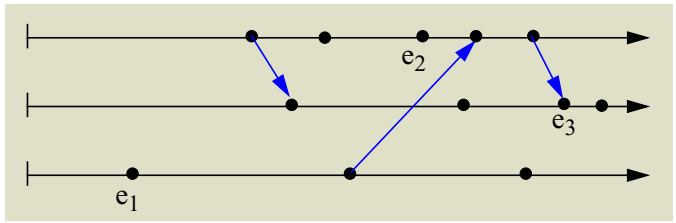
Zeitdiagramme

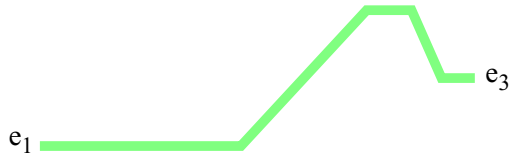
- Theorem [*Kausalkette*]:

  In einem Zeitdiagramm gilt für je zwei Ereignisse e, e' die Relation e<e' genau dann, wenn es einen Pfad von e nach e' gibt

  Nachrichtenpfeile + Teilstücke auf Prozessachsen von links nach rechts

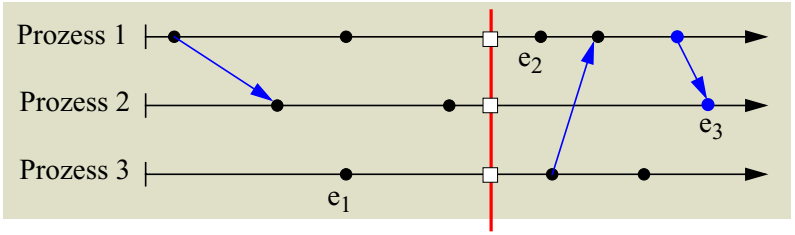
- Bew.: induktiv, Transitivität, "kann beeinflussen",...





- Beispiel: $e_1 < e_3$, aber *nicht* $e_1 < e_2$

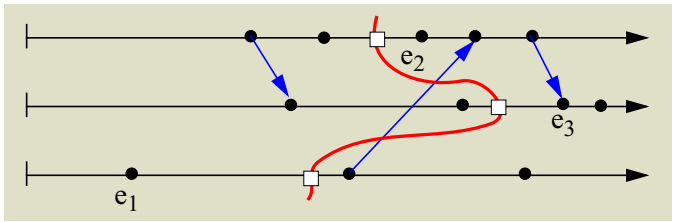
Gummibandtransformation



vertikaler Zeitschnitt

Diagramme sind daher äquivalent

Ein anderes Bild der *gleichen* Berechnung:



- Gummibandtransformation
  - abstrahiert von metrischer Struktur ("Zeit")
  - Stauchen und Dehnen der Prozessachsen
  - lässt topologische Struktur invariant (Kausalitätspfade von links nach rechts)
  - vertikale Zeitschnitte werden "verbogen" (umgekehrt ist interessant, wann sich krumme Zeitschnitte "geradebiegen" lassen!)

  Nachrichtenpfeile sollen aber nie rückwärts laufen

- Bei Fehlen einer absoluten Zeit sind alle Diagramme, die sich so deformieren lassen, gleich "richtig" (äquivalent)
  - kann man so immer eine schiefe Beobachtungslinie "senkrecht biegen"?
  - nein, leider nur wenn die Beobachtung kausaltreu ist!

Globale Zustände und Endzustände

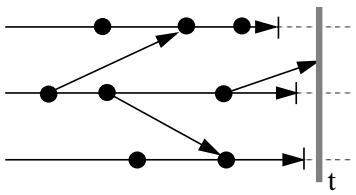
- Was ist ein *globaler Zustand* einer vert. Berechnung?
  - Charakteristikum verteilter Systeme: dieser ist auf die Prozesse verteilt und nicht unmittelbar ("gleichzeitig") zugreifbar!

- Damit vielleicht: *Ablauf einer vert. Berechnung* als Folge solcher Zustände definieren?
  - geht nicht so einfach, wie wir noch sehen werden...

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

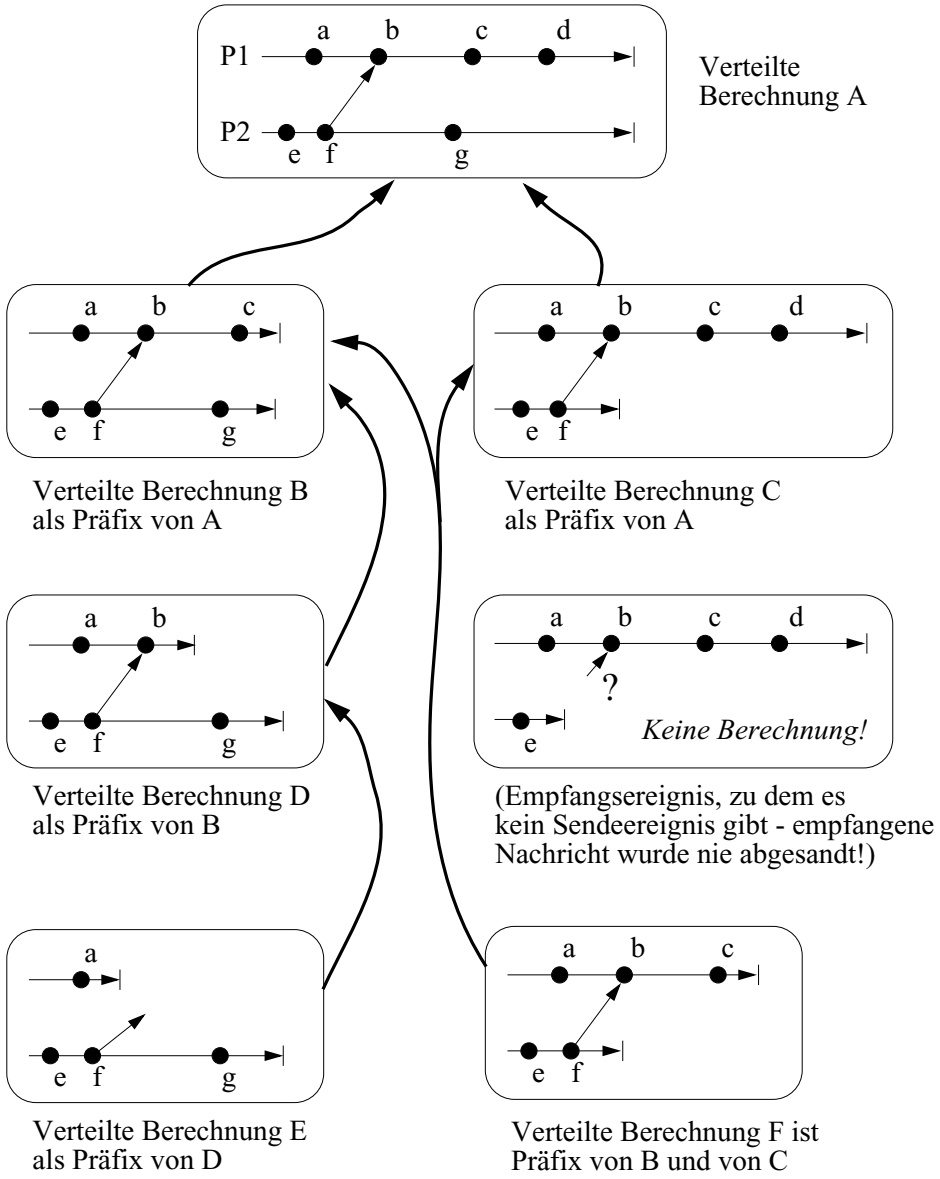
- *Globaler Zustandsraum*:
  - Kreuzprodukt lokaler Zustandsräume
  - Menge von Nachrichten ("in transit")

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



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

Präfixe von Berechnungen



Präfix-Berechnungen

- Gegeben sei eine Berechnung B = (E_1,...E_n,Γ,<);
  wann ist eine Berechnung B' eine Präfixberechnung von B?
  - E_i' ist eine *Teilmenge* von E_i
  - Γ' und <' sind *Einschränkungen* von Γ und < auf diese Teilmengen

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

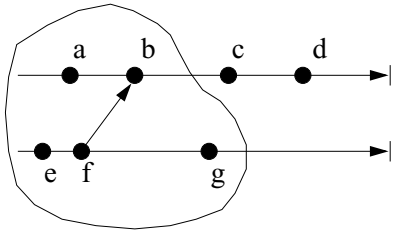
$$\forall\, x \in E', y \in E: \quad y < x \implies y \in E'$$

==> lokale Berechnungen sind wirkliche Anfangsstücke

==> unmögliche Diagramme, wo Nachrichten empfangen aber nicht gesendet werden, sind "automatisch" ausgeschlossen (dies wurde allerdings schon durch Γ' als Einschränkungen von Γ garantiert)

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

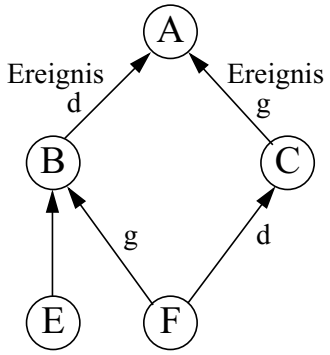
Beispiel:



(1): {a, b, e, f, g}

(2): {b, g}

Präfix-Relation



- Präfixdiagramm
  - kein (Wurzel)baum, aber
  - *gerichtet* und *zyklenfrei*
- Präfixrelation ist *transitiv*
- ==> 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")

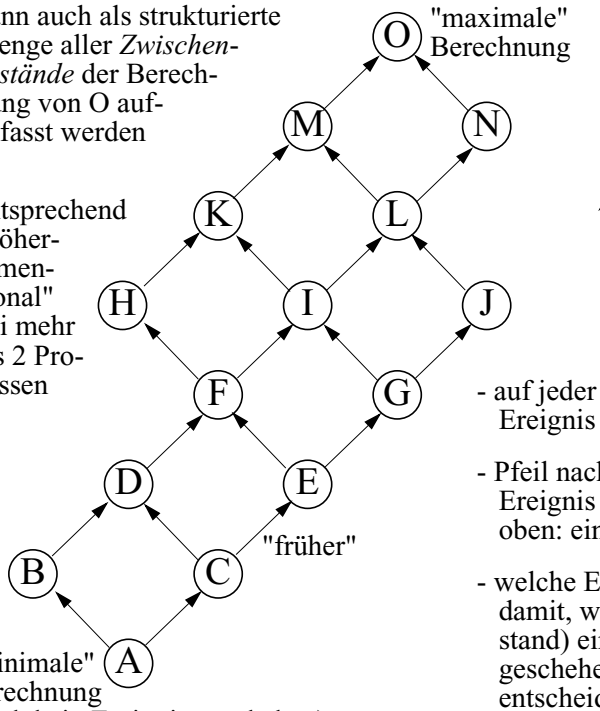
- Konsequenz:
  - direkter Vorgänger i.a. indefinit
  - verteilte Berechnung ist *keine Folge* von Zuständen, sondern - notgedrungen - eine geeignet definierte Halbordnung!

- Vgl. dies mit *sequentiellen* Berechnungen: Hier wird *jeder* Präfix einer Berechnung (genauer: dessen Endzustand) durchlaufen! (--> 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



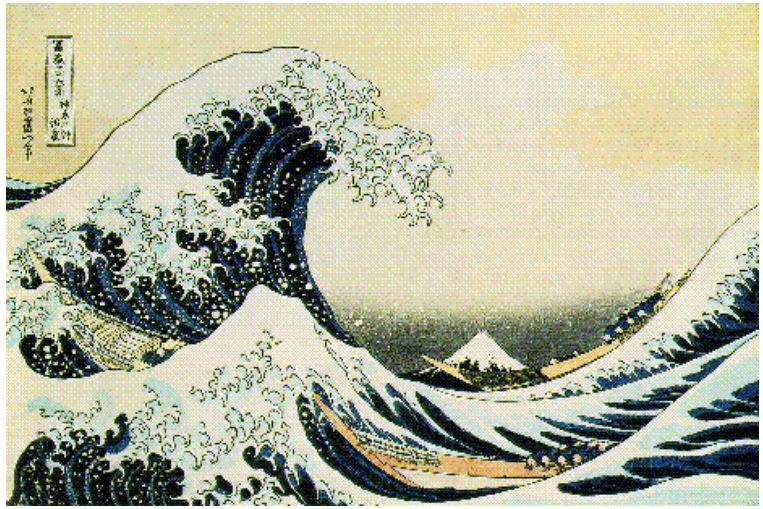
an dieser Stelle würde ein "unmögliches" Diagramm stehen

- entsprechend "höherdimensional" bei mehr als 2 Prozessen

- auf jeder Ebene kommt ein Ereignis hinzu
- Pfeil nach rechts oben: ein Ereignis von P1; nach links oben: ein Ereignis von P2
- welche Ereignismenge (und damit, welcher Zwischenzustand) einer Ebene *wirklich* geschehen ist, lässt sich nicht entscheiden --> keine vernünftige Frage!

- Ein Zwischenzustand hat also i.a. mehrere direkte Vorgänger- und Nachfolgezustände!
- Berechnung bewegt sich in diffuser Weise in diesem Zustandsraum von unten nach oben
  - von den Ecken her ausgefranstes n-dimensionales Gitter, mehr dazu später!

Wellenalgorithmen



Katsushika Hokusai (1760-1849): Die grosse Woge, Metropolitan Museum of Art, New York

This well-known masterpiece shows Mt. Fuji behind raging waves off the seacoast. Hokusai created "Mt. Fuji Off Kanagawa" (popularly known in the West as "The Wave") as part of his subscription series, "Thirty-Six Views of Mt. Fuji," completed between 1826 and 1833. This is one of the best-known Japanese woodblock prints, and with others of this period inspired the entire French Impressionist school. By making Mt. Fuji only rather small in the background the artist expresses, in a novel way, the elemental power of nature.

Wellenalgorithmen

Häufige Probleme bei verteilten Algorithmen / Systemen:

- *Broadcast* einer Information
- Globale *Synchronisation* zwischen Prozessen
- *Triggern eines Ereignisses* in jedem Prozess
- *Einsammeln* von verteilten Daten

Trennung von Phasen

==> *Wellenalgorithmen*

- *Alle* Prozesse müssen sich beteiligen
- *Basisalgorithmen* ("Bausteine") für andere Algorithmen
  (wechselseitiger Ausschluss, Terminierungerkennung, 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 ≤ visit
2. visit ≤ conclude

} (==> init ≤ conclude)

Wellenalgorithmen (2)

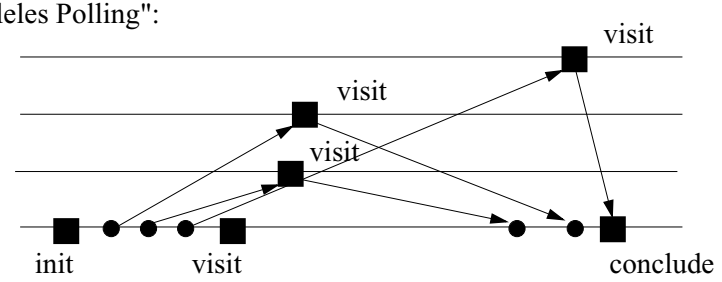
Aus 1: Über die Kausalketten lässt sich *Information* vom Initiator an alle Prozesse *verteilen*

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

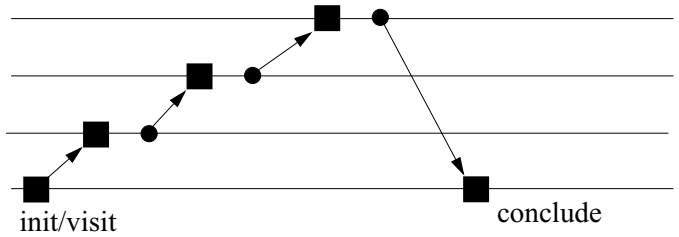
Bem.: a) init und conclude oft im selben Prozess ("Initiator")
b) init oder conclude kann mit visit verschmelzen (daher '≤' statt '<')

Beispiele:

"paralleles Polling":



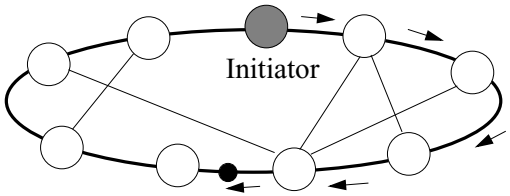
"Ring":



Einige Wellenalgorithmen

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

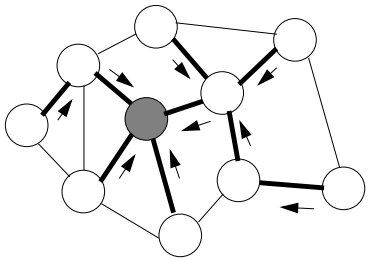
1.) *Ring* / *Hamiltonscher Zyklus* mit umlaufenden Token



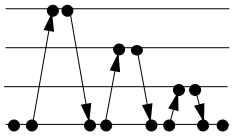
- "logischer" Ring genügt!
- in einem zusammenhängenden ungerichteten Graphen kann ein logischer Ring immer gefunden werden, indem man einen Spannbaum "umfährt"

2.) *Spannbaum*

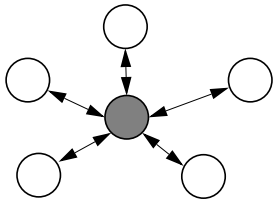
- an den Blättern reflektierte Welle (vereinfachter Echo-Algorithmus: flooding mit rek. acknowledgement)
- bei nicht-entartetem Spannbaum sind viele Nachrichten parallel unterwegs



3.) *(logischer) Stern*



sequentielles Polling



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

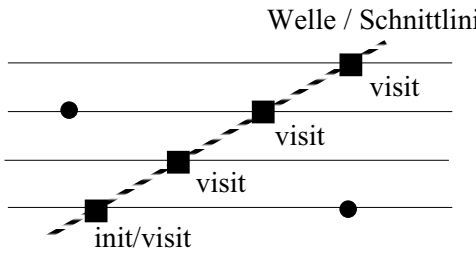
---

4.) *Echo-Algorithmus ist ein Wellenalgorithmus*

- visit-Ereignis entweder erster Erhalt eines Explorers oder Senden des Echos
- definiert so sogar zwei verschiedene "parallele" Wellen bzw. "Halbwellen"!

Wellen und Schnittlinien

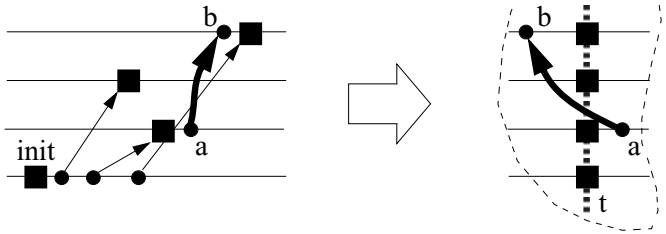
- Verbinden der visit-Ereignisse zu einer *Schnittlinie*:



Trennt die Menge der Ereignisse in *Vergangenheit* und *Zukunft*

oBdA: "gerade" Schnittlinie
(--> Gummibandtransf.)

- Falls ein Wellenalgorithmus einer anderen verteilten Anwendung überlagert wird: Lässt sich dann die Schnittlinie immer *senkrecht* zeichnen (Gummibandtransformation), so dass keine Nachricht "echt" rückwärts läuft?



keine bzgl. < linksabgeschl. Menge!

- Beispiel: zu dem dadurch festgelegten "globalen Zeitpunkt" t wäre eine Anwendungsnachricht zwar angekommen, aber noch nicht abgesendet!

  ==> die Welle würde ein unmögliches ("inkonsistentes") Bild liefern

  ==> Visit-Ereignisse lassen sich (hier) nicht als "virtuell gleichzeitg" ansehen

- Wichtige Fragen: Unter welchen Umständen definieren die visit-Ereignisse einen "Schnitt", der als *senkrechte* Linie aufgefasst werden kann? Lässt sich das erzwingen?
  (damit hätten wir so etwas wie globale Zeit --> später)

Eigenschaften von Wellenalgorithmen

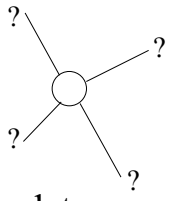
wodurch ist eigentlich der suggestive Name gerechtfertigt?

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

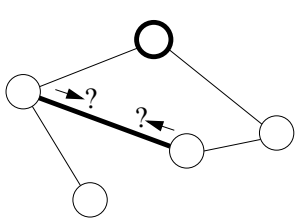
- Begründung: Da jedes visit-Ereignis vom init-Ereignis kausal abhängig ist, muss jeder andere Prozess mindestens eine Nachricht empfangen

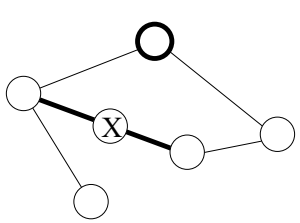
- mindestens n Nachrichten, falls conclude und init auf gleichem Prozess stattfinden (dann muss auch der Initiator eine Nachricht empfangen)

Satz: Ohne Kenntnis der Nachbaridentitäten werden *mindestens e Nachrichten* benötigt



Bew.: Über jede Kante muss eine Nachricht gesendet werden, wenn die Identität der Nachbarn unbekannt ist:





- würde man im linken Szenario die Kante nicht durchlaufen, dann würde im rechten Szenario Knoten X nicht erreicht und er könnte kein von init abhängiges visit-Ereignis ausführen

Wellenalgorithmen und Spannbäume

Satz: Die Kanten, über die ein Knoten (≠Initiator) erstmals eine Nachricht erhielt, bilden einen *Spannbaum*

(Voraussetzung: der Algorithmus wird an einer einzigen Stelle initiiert)

Beweis:

- Jeder Nicht-Initiator erhält mindestens eine Nachricht

- Es gibt also n-1 solche ersten Kanten

- Wir müssten noch einsehen:
  - die Menge der entsprechenden Kanten ist zyklenfrei,
  - *oder* die Menge dieser Kanten ist zusammenhängend

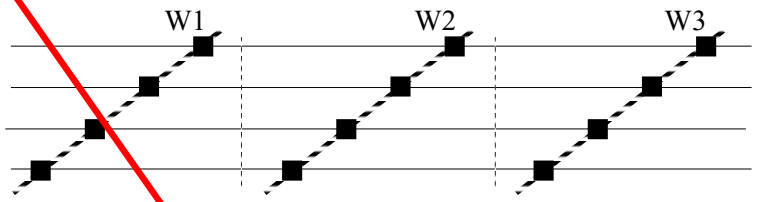
  } Denkübung: wieso?

- Damit und mit n-1 folgt die Baumeigenschaft

- man überlege sich, wieso der Beweis falsch wird, wenn ein Wellenalgorithmus an mehreren Stellen unabhängig initiiert wird!

- wir kennen solche Spannbäume bereits vom Echo-Algorithmus!

Folgen von Wellen



- Wellennummer j kann mit jeder Nachricht mitgeführt werden.

Start:

```
j := 1 (* Wellennummer *)
x := lokale Berechnung
init(j)  (* ggf. broadcast von x *)
```

j-te Iteration

Ersetze conclude durch:

```
j := j+1
if (* noch nicht fertig *)
   x := lokale Berechnung
   init(j)  (* ggf. broadcast von x *)
fi
```

Satz: Zu jedem globalen Zeitpunkt unterscheiden sich die Wellennummern zweier Prozesse um höchstens 1

Beachte zum Beweis den Zusammenhang zwischen Zeit und Kausalität

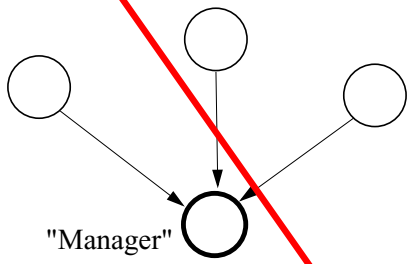
==> Phasensynchronisation der Prozesse möglich

Bem.: Verschiedene unabhängige Wellen (ggf. verschiedener Initiatoren) können sich in transparenter Weise "parallel" überlagern, wenn eindeutige Identitäten mitgeführt werden

Halbwellen

- Abschwächung des Wellenbegriffs, z.B. *kein init*:

--> *kontrahierende* Halbwelle



- Bsp.: Prozesse melden unaufgefordert ihre einzusammelnden Werte einem zentralen Prozess (*Stern*) bzw. allen Prozessen (*vollst. Graph*)

- Analog auch für *Bäume*, wo die Blätter "spontan" ihr Ergebnis weiter nach innen melden

- Manager bzw. Wurzel des Baumes kann conclude durchführen, wenn alle Nachbarknoten geantwortet haben (also ihr visit ausgeführt haben)

- Andere Auffassung: es gibt kein *eindeutiges* init, sondern i.a. mehrere unabhängige (--> "multisource-Algorithmus")

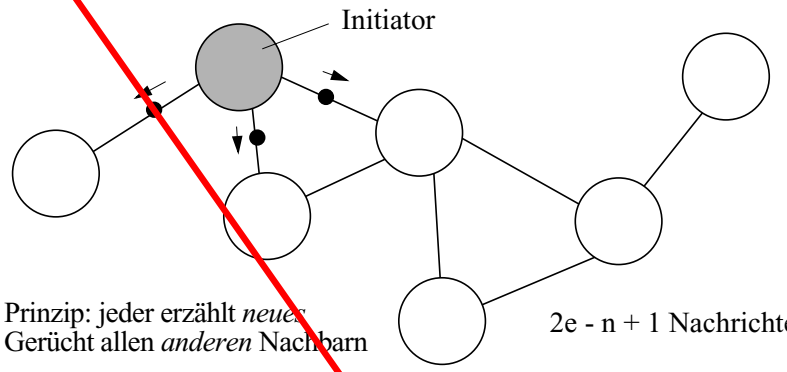
- Entsprechend: *kein conclude*: --> *expandierende* Halbwelle

- Es gibt keinen Prozess, der von der *Terminierung* des Algorithmus erfährt
- Beispiel: *flooding*-Algorithmus (entspricht erster Halbwelle des Echo-Algorithmus)

Der flooding-Algorithmus als Halbwelle

- Algorithmus kennen wir bereits!

Welle ohne conclude



Prinzip: jeder erzählt *neues* Gerücht allen *anderen* Nachbarn

2e - n + 1 Nachrichten

visit:

```
V:  {Eine Informationsnachricht kommt an}
    if not informed then
       informed := true;
       send <info> to all other neighbors;
    fi
```

informed = false

init:

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

Aktion I nur beim einzigen Initiator

- auch für solche expandierenden Halbwellen gilt (bei einem einzigen Initiator), dass die "ersten Kanten" einen *spannenden Wurzelbaum* bilden
- diesen Baum kann man benutzen, um (entsprechend einer kontrahierenden Halbwelle) *Acknowledgements* zum Initiator zu senden (dazu müssen die Blätter des Baumes aber erkennen, dass sie Blätter sind - wie geht das?)
- man erhält nach dieser Idee den *Echo-Algorithmus*!

Virtuell gleichzeitiges Markieren

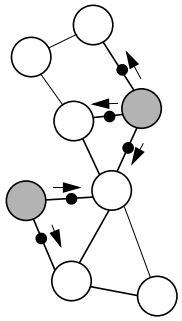
- Als Anwendung eines (Halb)wellenalgorithmus (≈ flooding)
- Voraussetzung hier: FIFO-Kanäle (d.h. keine Überholungen)

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

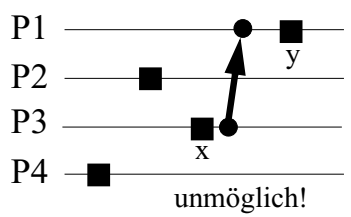
```
V:  {Eine Marker-Nachricht kommt an}
    if not marked then
       marked := true;   // = visit
       send <marker> to all neighbors;
    fi
```

!

mehrere Initiatoren sind zulässig!

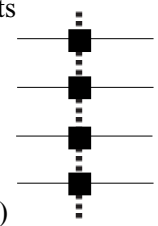


- Über *jede* Kante läuft in beide Richtungen ein Marker
- Eine andere Nachricht ("Basisnachricht"), die von einem markierten Knoten gesendet wird, kommt erst dann an, wenn der Empfänger auch bereits markiert ist



unmöglich!

Es ist daher stets rechts nebenstehende *Sicht* als Gummibandtransformation *möglich*, wo Basisnachrichten nicht rückwärts verlaufen und alle visits simultan sind (*wieso?*)



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

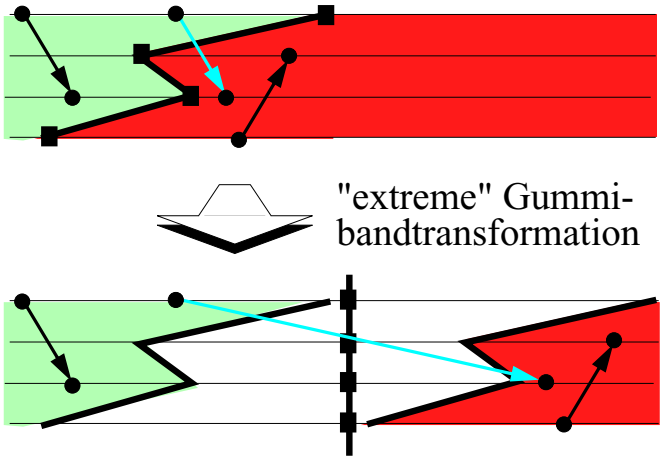
Fragen:
- geht virtuell gleichzeitiges Markieren auch ohne FIFO-Kanäle? (FIFO abschwächen, FIFO erzwingen, FIFO "simulieren"...)
- geht das auch mit weniger als 2e Nachrichten?

Vertikale Schnittlinien?

- Voraussetzung: Keine Nachricht läuft von der "Zukunft" in die "Vergangenheit" einer Schnittlinie
  - solche Schnittlinien heissen *konsistent*
    (linke Hälfte ist dann linksabgeschlossen bzgl. der Kausalrelation, d.h. linke Hälfte ist eine Präfixberechnung)

- Dann lässt sich diese Schnittlinie *vertikal* zeichnen, ohne dass dabei Nachrichten von rechts nach links laufen
  - als hätte die zugehörige Welle alle Prozesse gleichzeitig besucht!
  - offenbar nützlich für Terminierungserkennung, Sicherungspunkte...!

- "Konstruktiver" Beweis: Auseinanderschneiden entlang der Schnittlinie und den rechten Teil "ganz" über den weitesten rechts liegenden Punkt des linken Teils hinausschieben



- Zerschnittene Nachrichten reparieren ("verlängern")
- Schnittereignisse in die Lücke senkrecht untereinander legen

Anwendung von Wellenalgorithmen

- Als "Unterprogramm" innerhalb anderer Anwendungen, z.B.
  - Broadcast einer Information an alle Prozesse
  - Einsammeln verteilter Daten
  - Traversieren aller Prozese eines Prozessgraphen
  - Berechnung einer globalen Funktion, deren Parameter auf alle Prozesse verteilt sind (z.B. globales Minimum)
  - ...

- Abstrakt: um Schnitte durch ein Zeitdiagramm zu legen
  (Problem jedoch 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

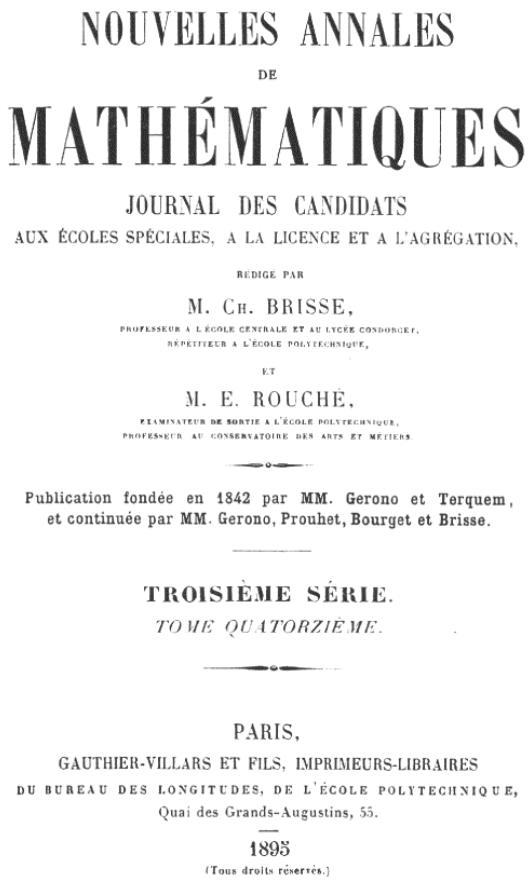
Sequentielle Traversierungsverfahren

- Unterklasse der Wellenalgorithmen (auf ungerichteten und zusammenhängenden Graphen) mit:
  - einem einzigen Initiator (bei dem init und conclude stattfindet)
  - *totaler Ordnung* auf allen visit-Ereignissen

- Interpretation: Ein *Token* wandert durch alle Prozesse und kehrt zum Initiator zurück
  - Visit-Ereignis: erstmalige Ankunft (oder Weiterreichen) des Tokens
- Relativ hohe Zeitkomplexität, da keine wesentliche Parallelität

---

- Bekannte einfache Verfahren für *spezielle Topologien*:
  - *Stern* (bzw. vollst. Graph): sequentielles "polling"
  - *Ring*
  - *Baum* (bzw. Spannbaum)
  - n-dimensionales *Gitter*: Verallgemeinerung des "ebenenweise" Durchlaufens eines 3-dimensionale Gitters (einen Schritt in der Dimension k+1, wenn k-dimensionales Untergitter durchlaufen...)
  - n-dimensionale *Hyperwürfel*: Traversiere ("rekursiv") einen n-1-dim. Hyperwürfel bis auf den letzten Schritt, gehe in Richtung der n-ten Dimension und traversiere dort den n-1-dim. Hyperwürfel...
  - Hamiltonsche Graphen (z.B. auch Ring und Hyperwürfel als Sonderfall)



NOUVELLES ANNALES

DE

MATHÉMATIQUES

JOURNAL DES CANDIDATS

AUX ÉCOLES SPÉCIALES, A LA LICENCE ET A L'AGRÉGATION,

RÉDIGÉ PAR

M. CH. BRISSE,

PROFESSEUR A L'ÉCOLE GÉNÉRALE ET AU LYCÉE CONDORCET,
RÉPÉTITEUR A L'ÉCOLE POLYTECHNIQUE,

ET

M. E. ROUCHÉ,

EXAMINATEUR DE SORTIE A L'ÉCOLE POLYTECHNIQUE,
PROFESSEUR AU CONSERVATOIRE DES ARTS ET MÉTIERS.

Publication fondée en 1842 par MM. Gerono et Terquem,
et continuée par MM. Gerono, Prouhet, Bourget et Brisse.

TROISIÈME SÉRIE.

TOME QUATORZIÈME.

PARIS,

GAUTHIER-VILLARS ET FILS, IMPRIMEURS-LIBRAIRES

DU BUREAU DES LONGITUDES, DE L'ÉCOLE POLYTECHNIQUE,

Quai des Grands-Augustins, 55.

1895

(Tous droits réservés.)

Nouvelles Annales de Mathématiques 14 (1895)

LE PROBLÈME DES LABYRINTHES;

PAR M. G. TARRY.

Tout labyrinthe peut être parcouru en une seule course, en passant deux fois en sens contraire par chacune des allées, sans qu'il soit nécessaire d'en connaître le plan.

Pour résoudre ce problème, il suffit d'observer cette règle unique :

Ne reprendre l'allée initiale qui a conduit à un carrefour pour la première fois que lorsqu'on ne peut pas faire autrement.

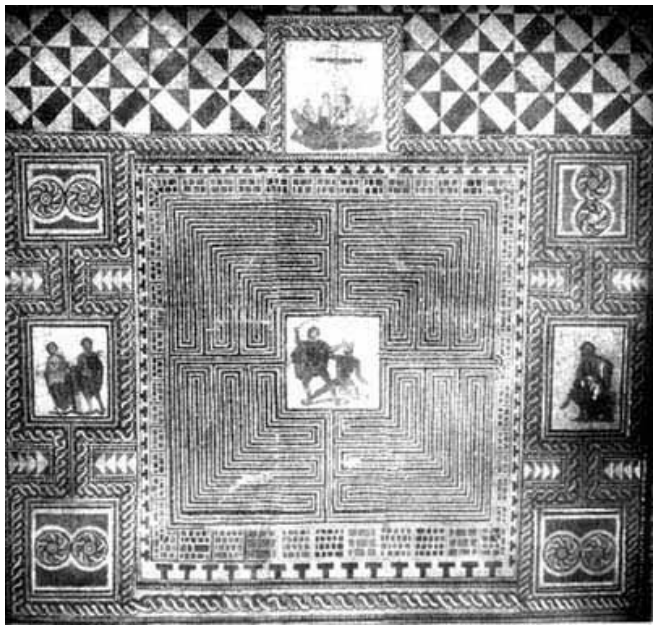
. . .

En suivant cette marche pratique, un voyageur perdu dans un labyrinthe ou dans des catacombes, retrouvera forcément l'entrée avant d'avoir parcouru toutes les allées et sans passer plus de deux fois par la même allée.

Ce qui démontre qu'un labyrinthe n'est jamais inextricable, et que le meilleur fil d'Ariane est le fil du raisonnement.

Gaston Tarry (1843-1913) studied mathematics 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.

Labyrinthe



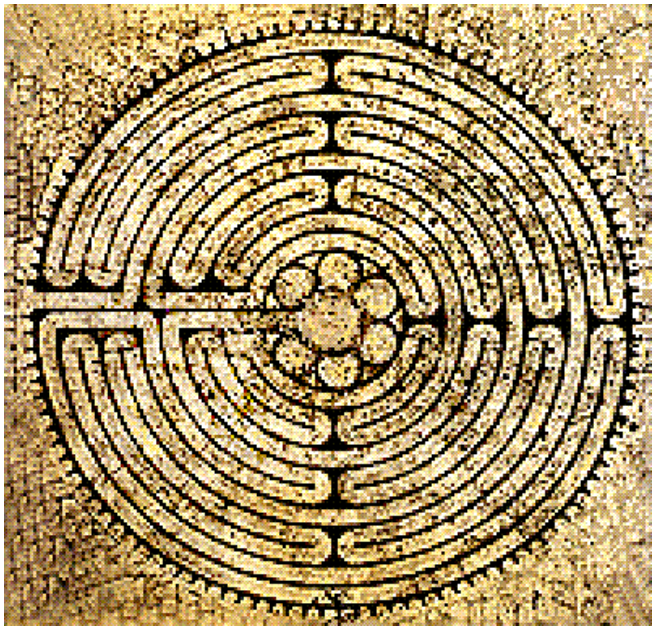
Römisches Mosaik (aus Loig bei Salzburg) illustriert die Geschichte von Theseus und Minotaurus im Labyrinth

Arcera, Spanien

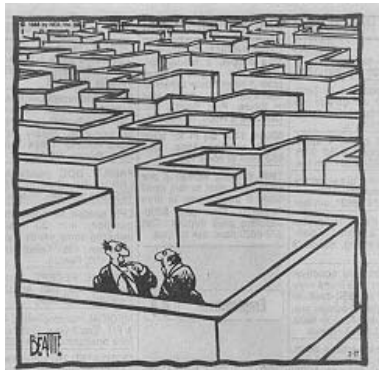
O'odham-Volk (Papago-Indianer) aus Süd-Arizona

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

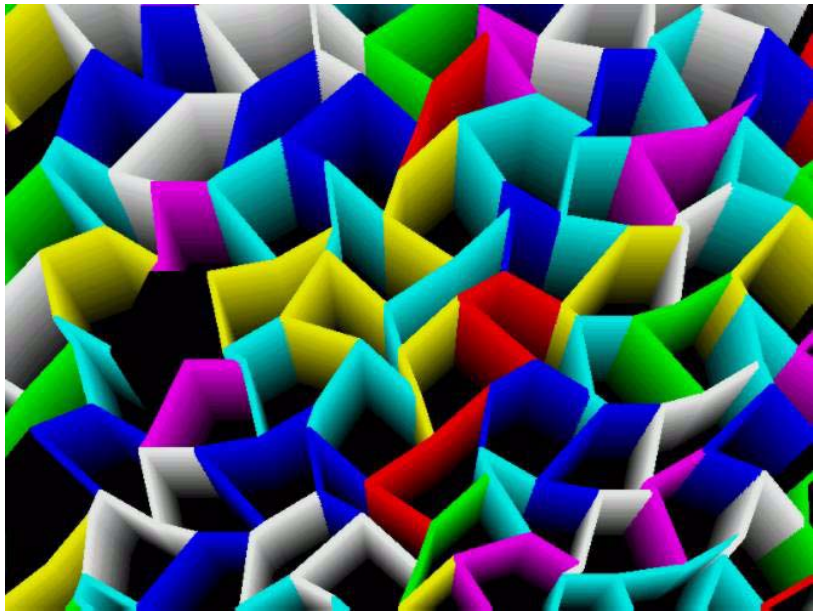
Labyrinthe (2)



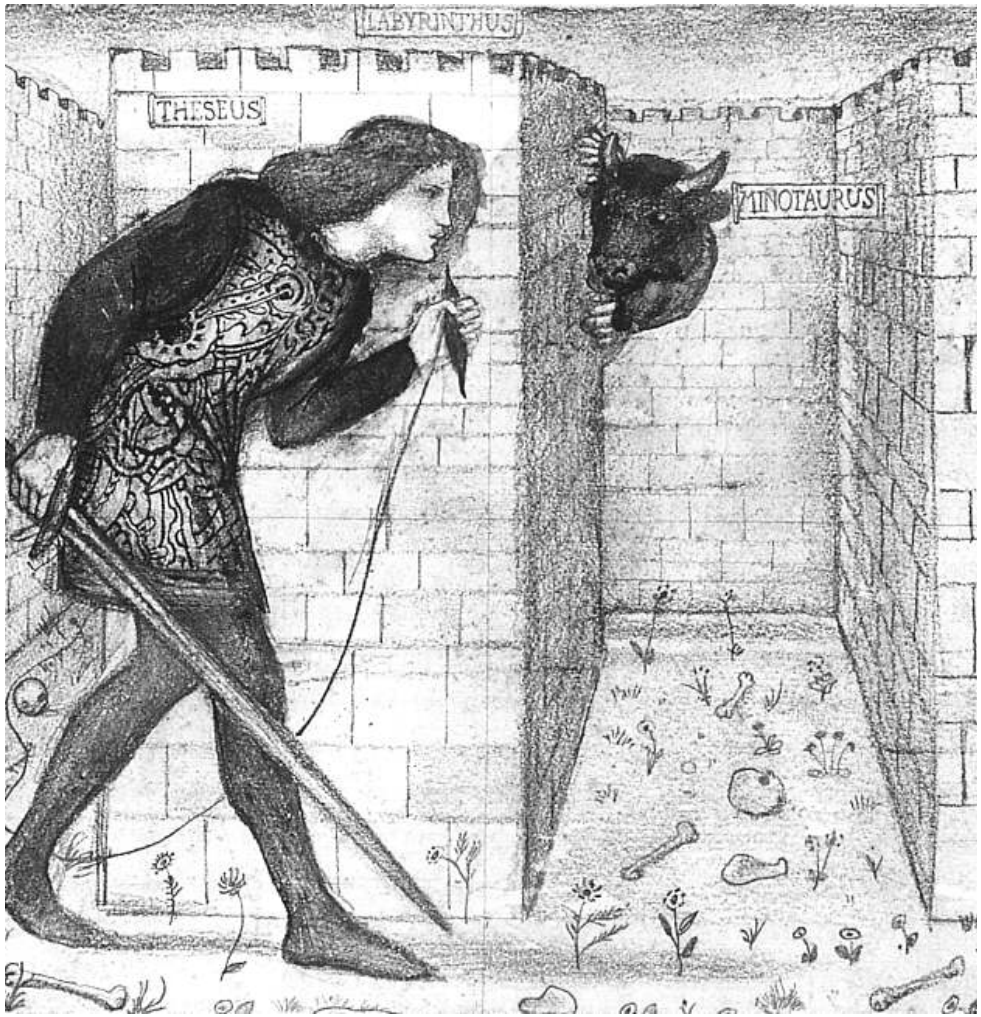
Labyrinthe (3)



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



Labyrinthe (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 (≠ 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 ==> Token ist beim Initiator

Beweis: Für jeden Nicht-Initiator p gilt: Wenn p das Token hat, dann hat p das Token k-Mal (auf jeweils unterschiedlichen Kanälen, Regel R1) erhalten und auf k-1 (jeweils unterschiedlichen Kanälen) gesendet ==> es gibt noch mindestens einen unbenutzten "Ausgangskanal" ==> das Token bleibt nicht bei p.

(2) Alle Kanäle des Initiators werden in beiden Richtungen genau 1 Mal vom Token durchlaufen

Beweis: Für jeden "Ausgangskanal" klar, sonst wäre der Algorithmus nicht terminiert: Nach (1) bleibt das Token nicht bei einem andereren Prozess stecken. Das Token muss genauso oft zum Initiator zurückgekehrt sein, wie es von dort weggeschickt wurde, und zwar über jeweils andere Kanäle (Regel R1) ==> Jeder "Eingangskanal" des Inititators wurde benutzt.

(3) Für jeden besuchten Prozess p gilt: Alle Kanäle von p wurden in beide Richtungen durchlaufen

Beweis durch Widerspruch: Betrachte den ersten (= "frühesten") besuchten Prozess p, für den dies nicht gilt. Nach (2) ist dies nicht der Initiator. Sei Vater(x) derjenige Prozess, von dem x erstmalig das Token erhielt. Für Vater(p) gilt nach Wahl von p, dass alle Kanäle in beide Richtungen durchlaufen wurden. ==> p hat das Token an Vater(p) gesendet. Wegen Regel R2 hat daher p das Token auf allen anderen (Ausgangs)kanälen gesendet. Das Token musste dazu aber genauso oft auf jeweils anderen Kanälen (R1) zu p zurückkommen. ==> Alle Eingangskanäle wurden ebenfalls benutzt.

(4) Alle Prozesse wurden besucht

Beweis: Andernfalls gäbe es eine Kante von einem besuchten Prozess q zu einem unbesuchten Prozess p, da der Graph zusammenhängend ist. Dies steht aber im Widerspruch zu (3), da diese Kante vom Token durchlaufen wurde.

Die Welleneigenschaft ergibt sich i.w. aus (1) und (4)

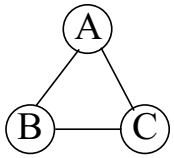
Depth-first-Algorithmen und Spannbäume

Klassischer Depth-first-search-Algorithmus:

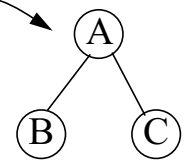
- Token geht erst dann zurück, wenn alles andere "abgegrast" ist
- Token kehrt um, sobald es auf einen bereits besuchten Knoten trifft
- jede Kante wird in jede Richtung genau 1 Mal durchlaufen
- Tarry-Algorithmus lässt sich zu Depth-first-Traversierung spezialisieren

==> 2e Nachrichten, 2e Zeitkomplexität

- Depth-first-search-Algorithmen liefern beim Durchlaufen eines Graphen einen Spannbaum (Wurzel = Initiator)
  (wie jeder Wellenalgorithmus!)



wieso ist der rechte Baum *kein* von einem Depth-first-search-Algorithmus erzeugter Spannbaum des linken Graphen mit Initiator A?



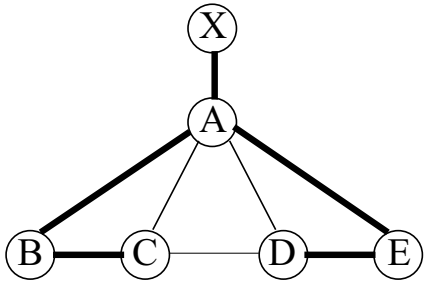
- Eine Charakterisierung solcher Spannbäume:

Jede Kante des Graphen, die keine Kante des 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

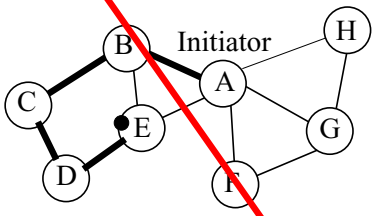
- Mit folgender Regel lässt sich der Nichtdeterminismus des Tarry-Algorithmus soweit einschränken, dass das klassische Depth-first-Traversierungsverfahren resultiert:

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?

Depth-first mit Besuchslisten

Voraussetzung: Nachbaridentitäten der Knoten sind bekannt



Token bei E enthält die Namen der bereits besuchten Knoten A, B, C, D, E; Knoten E wird also das Token nicht an B oder A weitersenden (sondern zurück an D)

Idee: - Token merkt sich bereits besuchte Knoten und wird nicht dorthin propagiert

- Jeder Knoten (≠ Initiator) wird also nur 1 Mal besucht (und sendet Token 1 Mal zurück)

==> 2n-2 Nachrichten, 2n-2 Zeitkomplexität

Vorteil: Bei "dichten" Netzen ($e \gg n$) effizienter!

Nachteil: Hohe Bit-Komplexität (d.h. "lange" Nachrichten)

Bem.: Nachbarschaftswissen lässt sich immer mit 2e Nachrichten erreichen! (Algorithmus dafür haben wir früher kennengelernt)

Frage: Lässt sich vielleicht ein Wellenalgorithmus angeben, bei dem man immer oder zumindest manchmal mit n-1 Nachrichten auskommt (wenn Nachbarschaftswissen vorhanden ist; dann ist das eine untere Schranke, wie wir wissen), statt mit 2n-2?

Depth-first mit Auftragslisten

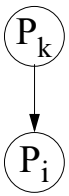
(Voraussetzung wieder: Nachbaridentitäten der Knoten sind bekannt)

- Idee: Token enthält zwei Mengen <...,z,s>:

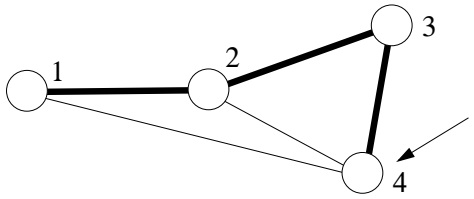
  z: Menge der bereits besuchten Knoten
  s: Menge noch zu besuchender Knoten ("Aufträge")

- Initial sendet Initiator P_i an einen Nachbarn P_j:
  <..., {i}, Nachbarn\{j}>

- Bei Empfang von <...,z,s> von P_k bei P_i:



y := Nachbarn \ z
case
  y = ∅ and s = ∅ --> terminated
  y = ∅ and s ≠ ∅ --> send <... z ∪ {i}, s\{i}> to P_k (zurück)
  y ≠ ∅ --> send <... z ∪ {i}, s∪ y \ {i} \ {m} > to $P_m \in$ y



Bereits hier nach drei Nachrichten terminiert

- Beendet, wenn s leer wird (wieso? Beweis?)
- Anderer Knoten als Initiator stellt Terminierung fest (Nachteil? Vorteil, da schneller?)
- Nachrichten- / Zeitkomplexität: n-1 bis 2(n-1)

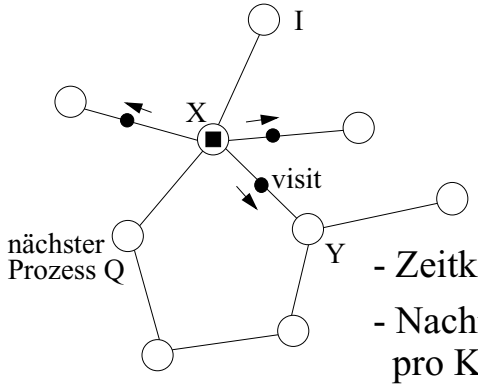
Traversierungsverfahren von Awerbuch

- Voraussetzung: Nachbaridentitäten unbekannt
- Idee: Prozess weiss spätestens dann, wenn das Token ihn besucht, an welche Nachbarn es nicht weitergereicht werden soll, weil es dort schon war

- Wenn Token einen Prozess besucht:

Sende *visit-Nachrichten* an "alle" Nachbarn;
warte auf *Ack* bevor Token weitergereicht wird
--> alle Nachbarn wissen, wo Token bereits war

Ein Prozess schickt Token niemals an einen Prozess, von dem er eine visit-Nachricht erhalten hat
(Ausnahme: Vaterknoten beim Backtrack)



Keine visit-Nachricht notwendig an:
- Vaterprozess (hier: I)
- nächsten Prozess (hier: Q)

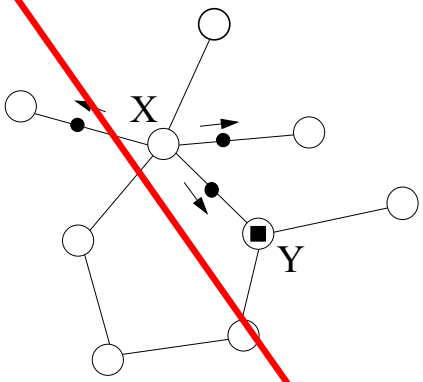
- Zeitkomplexität: 4n-2
- Nachrichtenkomplexität: pro Kante i.a. 2 visit, 2 ack (bzw. Token)... --> ca. 4e

- Frage: Muss Y eine visit-Nachricht an X schicken?
- Frage: Wozu überhaupt auf ack-Nachrichten warten?

Variante von Cidon

- Idee: Wie Awerbuch, aber keine acks senden!

- Was geschieht, wenn visit-Nachricht noch nicht da?
  (also der Kanal XY langsam ist im Vergleich zum Umweg, den Token nahm)



- X erhält Token auf "falscher" Kante von Y --> ignorieren
  (aber wie eine visit-Nachricht von Y behandeln, Y sendet kein visit mehr an X)

- Y erhält visit-Nachricht von X verspätet, nachdem bereits das Token an X gesendet wurde --> Token neu generieren und an einen anderen Nachbarn schicken

---

- Zeitkomplexität: 2n-2 ← Kann es (für beliebige Graphen) ein schnelleres Traversierungs-verfahren geben?

- Nachrichtenkomplexität:
  max. 4e (max. 2 visit, 2 Token pro Kante)

Der Phasenalgorithmus

- Voraussetzung (hier): bidirektionale Nachrichtenkanäle

- *Grundprinzip (Initiator):* ← Prinzip geht auch auf *gerichteten* Graphen

  {noch nicht terminiert}
  - Nachricht an alle Nachbarn senden
  - Lokale Berechnung durchführen
  - Phase := Phase + 1 /* lok. Variable; initial 0 */
  - Eine Nachricht von allen Nachbarn empfangen

- Nicht-Initiatoren steigen erst nach Erhalt einer ersten Nachricht in einen solchen Zyklus ein

---

Eigenschaften: ← Aufgabe: Hiermit die *Wellen-eigenschaft* nachweisen! ← Was sind visit- und conclude-Ereignise?

- Falls ein Prozess in der i-ten Phase ist, befinden sich seine Nachbarn in der i-ten, i-1-ten, oder i+1-ten (wieso?)

- Zwei Prozesse der Entfernung q haben zu jedem Zeitpunkt einen "Phasenunterschied" von max. q (wieso?)
  - Denkübung: können alle Prozesse gleichzeitig in Phase 1 (oder allg.: k) sein?

- Mit D = Durchmesser --> Ist der (einzige) Initiator am Ende von Phase D, wurde jeder vom Initiator (indirekt) erreicht
  - Denkübung: hat der Initiator dann bereits von allen Prozessen indirekt gehört?
  - wenn ich weiss, dass jeder in einer Phase > 0 ist, gibt es dann eine effektiv nutzbare Nachrichtenkette von jedem zu mir?

- Eine obere Schranke von D (z.B. n) muss bekannt sein, damit die Terminierung festgestellt werden kann

Algorithmus von Finn

- Wellenalgorithmus für (stark zusammenhängende) *gerichtete* Graphen (unterscheide daher *in-neighbors* und *out-neighbors*)

- Prozesse müssen eindeutige Identitäten besitzen

- *Idee des Algorithmus*: Solange es Prozesse gibt, die man kennt, aber deren Nachbarn man noch nicht alle kennt, ist man noch nicht fertig

  --> Hüllenbildung, dadurch Kennenlernen des gesamten Graphen

---

- Jeder Prozess P_i besitzt zwei Mengenvariablen A, B, die so initialisiert sind: A = {i}; B = ∅

R_i

Bei Empfang von <A',B'>:
A := A∪A'; B:= B∪B';
if {über alle Eingangskanäle irgendwann mind. eine Nachricht erhalten} **then** B:= B∪{i}; **fi**;
if {A oder B hat sich geändert} **then**
send <A,B> **to all** out-neighbors; **fi**

I_1

{noch nicht gestartet}
send <A,B> **to all** out-neighbors

init-Ereignis (was geschieht eigentlich bei *mehreren* Initiatoren?)

C_i

{A = B}
conclude

jeder Prozess kann conclude ausführen, insbesondere aber der Initiator

Finns Verfahren ist ein Wellenalgorithmus

(1) $x \in B \Longrightarrow$ *in-neighbors(x)* $\subseteq A$
ist eine Invariante für alle Paare (A,B)

- Aktionen C und I verändert (A,B) nicht.
- Aktion I erzeugt ein neues Paar ({i},∅).
- Die beiden update-Aktionen A := A∪A'; B:= B∪B' in Aktion R erhalten ebenfalls die Invariante: Für ein x ∈ B nach dem update war x ∈ B vor dem update oder x ∈ B'. Folglich war auch bereits vorher in-neighbors(x) ⊆ A oder in-neighbors(x) ⊆ A', und damit auch noch nach dem update.
- Falls in R das Statement B:= B∪{i} ausgeführt wird, sind alle y ∈ in-neighbors(i) bereits in A, da Prozess i laut Algorithmus von allen Nachbarn eine Nachricht erhalten hat, in deren A-Menge die Identität des Senders (also desNachbarn) enthalten ist.
- Man beachte ferner, dass dies nicht nur eine Invariante ist, sondern sogar *immer gilt,* da schon initial (A,B) = ({i},∅) in jedem Prozess ist. □

---

(2) *Wenn* A = B in einem Prozess P_i gilt, dann ist dort A = B = {Menge *aller* Prozessidentitäten}

- Es gilt schon initial i ∈ A, folglich (da nun A=B) auch i ∈ B. Da die Mengen nie vermindert werden, bleibt i in beiden Mengen.
- Wegen obiger Invarianten (1) ist in-neighbors(i) ⊆ A.
- Wegen A = B also auch in-neighbors(i) ⊆ B.
- Aus der Invarianten (1) folgt entsprechend in-neighbors(in-neighbors(i)) ⊆ A.
- Damit gilt induktiv ∀k: in-neighbors^k(i) ⊆ A = B.
- Wegen des starken Zusammenhangs ist jeder Prozess in in-neighbors^j(i) für ein gewisses j. □

(3) Sei *visit* = Aktion R bei erstem Versenden der eigenen Identität in der Menge B
Bei conclude (d.h. A=B) im Initiator gilt:

a) Jeder Prozess $P_j \neq$Initiator hat $visit_j$ ausgeführt und $visit_j <$ conclude,

b) init < $visit_j$

a): Wegen (2) "kennt" der Initiator bei conclude P_j (d.h. $j \in A = B$); er kann davon nur über eine Nachrichtenkette erfahren haben, an deren Anfang P_j seine Identität j versendet hat.

b): P_j versendet seine Identität nicht spontan, sondern nur nach (indirekter) Aufforderung durch den Initiator. □

Es ist nun noch zu zeigen, dass der Initiator tatsächlich nach endlicher Zeit conclude ausführt, d.h. dass bei ihm A = B wird (liveness)!

(4) Beim Initiator wird schliesslich A=B
(Voraussetzungen: stark zusammenhängender Graph, endliche Nachrichtenlaufzeiten und Aktionsdauern etc.)

- Über jede Kante des Graphen läuft schliesslich mindestens eine Nachricht ("flooding"): Das erste Empfangen eine Nachricht von einem Nachbarn vergrössert echt die eigene Menge A --> Welle wird an alle Nachbarn weiterverteilt.

- Jeder Prozess P_i verbreitet schliesslich seine eigene Identität i an alle Nachbarn über die B-Mengen, da B := B ∪ {i} ausgeführt wird.

- Jede Identität erreicht schliesslich mittels Flooding den Initiator über die B-Mengen.

- Beim Initiator gilt so schliesslich: B = {Menge aller Prozessidentitäten}.

- $A \supseteq B$ ist eine Invariante, wie man leicht überprüft.

- Aus den letzten beiden Eigenschaften folgt, dass schliesslich A = B gilt. □

Mit (3) und (4) ist alles gezeigt:
Finns Algorithmus ist ein Wellenalgorithmus!

- obere Schranke für Durchmesser etc. braucht nicht bekannt zu sein
- Nachrichtenkomplexität ≤ 2 n e (wieso?)
- höhere Bit-Komplexität als z.B. der Echo-Algorithmus

Eingesetzte Beweistechniken:

- atomare Aktionen
- Invarianten
- monotone Approximation

Es gibt verschiedene Wellenalgorithmen

- Topologiespezifische, z.B. für
  - Ring
  - Baum
  - allg. Graph

  } hierfür spezialisierte Verfahren u.U. besonders effizient

- Voraussetzungen bzgl. Knotenidentitäten
  - eindeutig oder
  - anonym

- Voraussetzungen bzgl. notwendigem "Wissen", z.B.
  - Nachbaridentitäten
  - Anzahl der Knoten (bzw. obere Schranke)
  - ...

- Voraussetzungen bzgl. Kommunikationssemantik
  - synchron, asynchron, FIFO-Kanäle, bidirektionale Kanäle...?

- Qualitätseigenschaften
  - Sequentiell oder parallel (bzw. "Parallelitätsgrad")
  - Anzahl möglicher Initiatoren (mehr als einer?)
  - Zeitkomplexität
  - Nachrichtenkomplexität (worst/average case)
  - Bitkomplexität (Länge der Nachrichten)
  - Dezentralität (kein Engpass?)
  - Symmetrie (alle lok. Algorithmen identisch?)
  - Fehlertoleranz (Fehlermodell? Grad and Fehlertoleranz?)
  - Einfachheit (--> 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 Wellenalgorithmen!

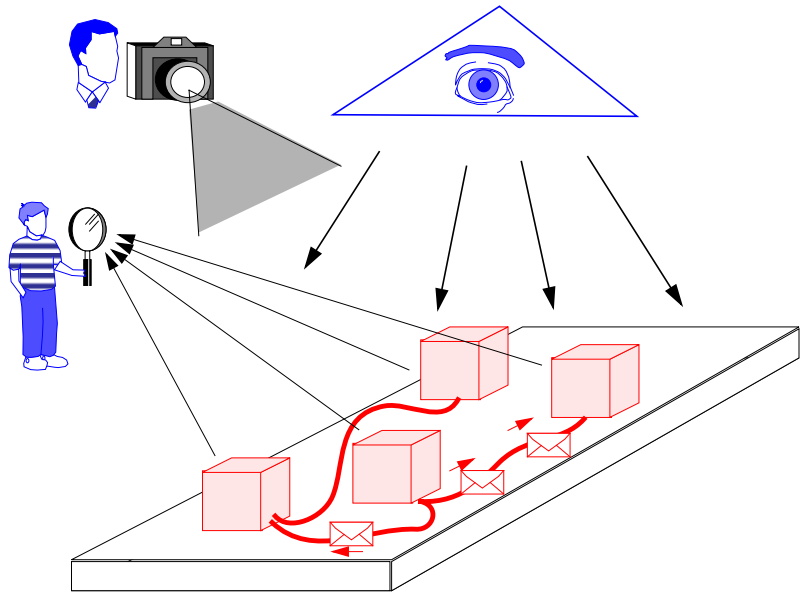
Übungen (5)

Man *vergleiche* die in der Vorlesung und den Übungen betrachteten *Wellenalgorithmen*. Dazu gehören insbesondere der Echo-Algorithmus, der Phasenalgorithmus, Finns Algorithmus, Depth-first-Traversierung (bzw. Verfahren von Tarry), Awerbuchs Verfahren, Variante von Cidon, sowie Verfahren für Graphen mit bekanntem spannendem Baum, bekanntem Hamilton'schen Zyklus und Verfahren auf Sterntopologien (bzw. auch vollständigen Graphen) mit dem Initiator als Sternmittelpunkt.

Die zu untersuchenden *Kriterien* sollen u.a. sein:
-- Nachrichtenkomplexität (worst case, average case),
-- Zeitkomplexität (worst case, average case),
-- Bitkomplexität,
-- Anwendbarkeit auf gerichtete / ungerichtete Graphen,
-- Anwendbarkeit, falls Nachrichten sich überholen können,
-- Anwendbarkeit bei anonymen Graphen,
-- Berechnung von nicht-idempotenten (assoziative, kommutativen) Operationen,
-- Kenntnis von Parametern wie Durchmesser, Nachbaridentitäten etc.

Die wichtigsten Angaben ordne man zweckmässigerweise in *Tabellenform* an. Man *diskutiere* die relativen Vor- und Nachteile. Gibt es einen besten oder schlechtesten Wellenalgorithmus? Aussagen zu den Kriterien sollten *begründet* werden, sofern sie nicht offensichtlich sind oder in der Vorlesung bewiesen wurden.

Globale Zustände, Beobachtungen, Prädikate



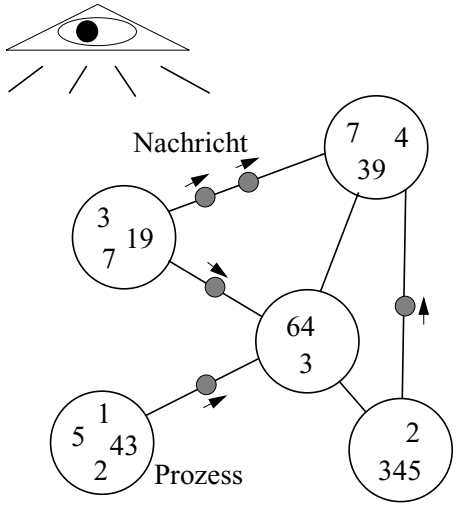
Kausal-konsistente globale Zustände

Webster:

State = a set of circumstances or attributes characterizing a person or thing at a given time

Gibt es "globale Zeit" in einem vert. System?

Globaler Zustand (zu einem Zeitpunkt):
Alle lokalen *Prozesszustände* +
alle *Nachrichten*, die "unterwegs" sind



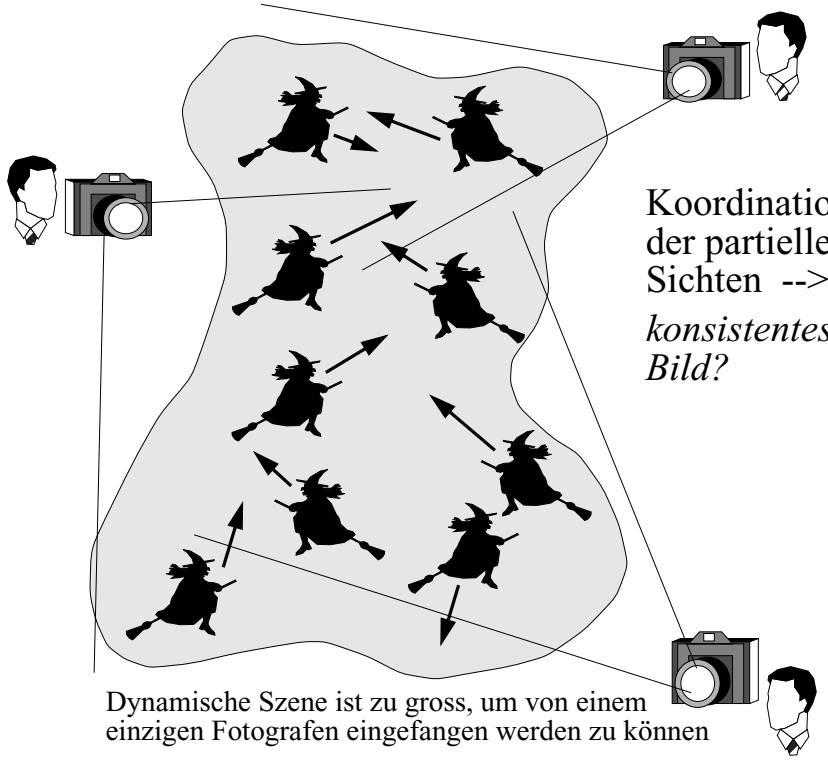
Schnappschuss

Problem:

Prozesszustände sind nur einzeln ("nacheinander") erfahrbar --> Konsistenz?

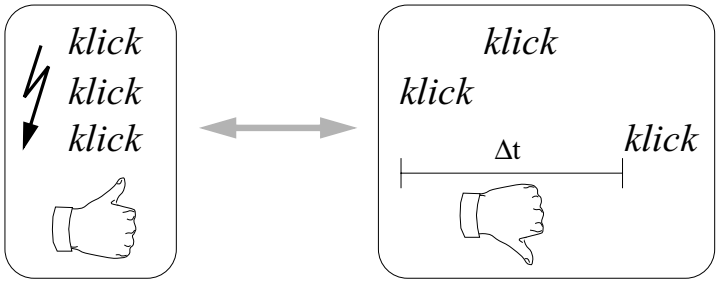
als ob alles gleichzeitig wahrgenommen wird

Synchronisierte lokale Schnappschüsse?



Koordination der partiellen Sichten --> *konsistentes Bild?*

Dynamische Szene ist zu gross, um von einem einzigen Fotografen eingefangen werden zu können



Das Schnappschussproblem

Problem: "Momentaner" *Schnappschuss* des globalen Zustands, ohne das System anzuhalten

Realität:

- *Volkszählung*: Stichzeitpunkt (geht hier nicht)
- *Inventur*: Einfrieren (unpraktisch)

Anwendungen:

- konsistenter Aufsetzpunkt für 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

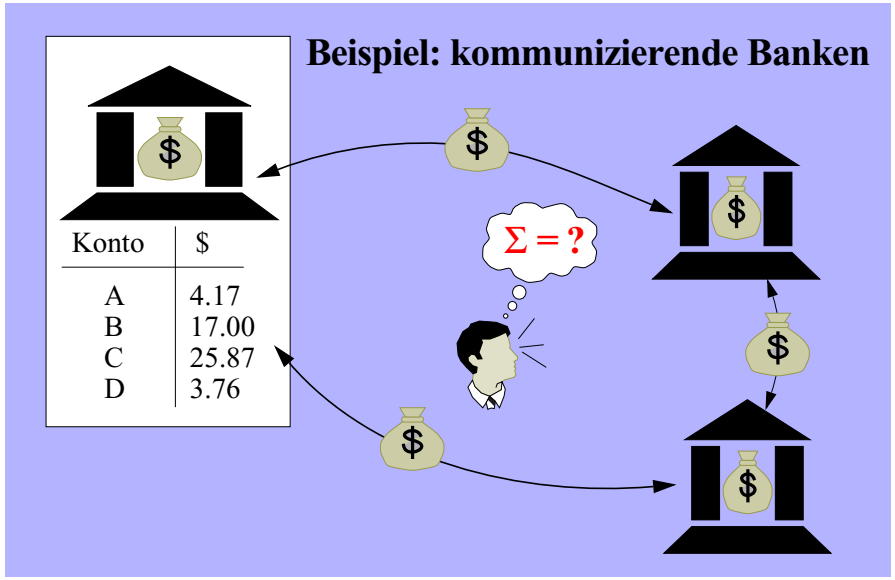
Schwierigkeiten:

- 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

zumindest dies ausschliessen!

→ *Schnappschussalgorithmus*

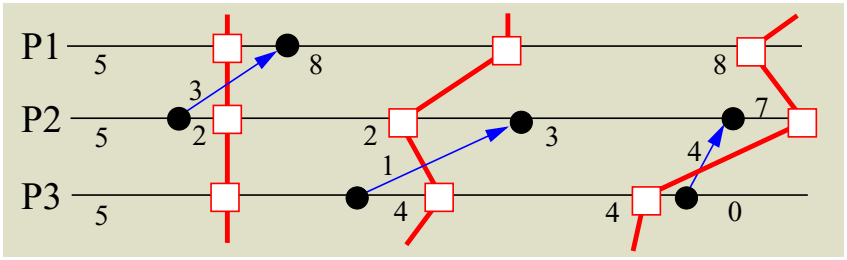
Beispiel: Kommunizierende Banken



- Modellierung:
  - ständige Transfers zwischen den Konten bzw. Banken
  - lokale atomare Aktionen: alle Geldkonten *einer* Bank können "gleichzeitig" (also atomar und damit "lokal konsistent") untersucht werden
- Wieviel Geld ist in Umlauf?
  - konstante Geldmenge, oder
  - monotone Inflation (--> untere Schranke für momentane Geldmenge)
- Erschwerte Bedingungen:
  - niemand hat eine globale Sicht
  - gemeinsame Zeit?

(In)konsistente Schnitte

Beispiel: Wieviel Geld ist in Umlauf?



idealer (senkrechter) Schnitt --> 15

konsistenter Schnitt --> 15

inkonsistenter Schnitt --> 19 (+4 ?)

Unerreichbar

Lässt sich nicht "geradebiegen"

Äquivalent zu senkrechtem Schnitt (lässt sich problemlos "geradebiegen")

manches doppelt (bzw. gar nicht) gezählt

"Gummiband-transformation"

--> verändert Metrik

--> erhält Topologie

Wie wir noch einsehen werden:

Kausaltreues Beobachten <==> 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...

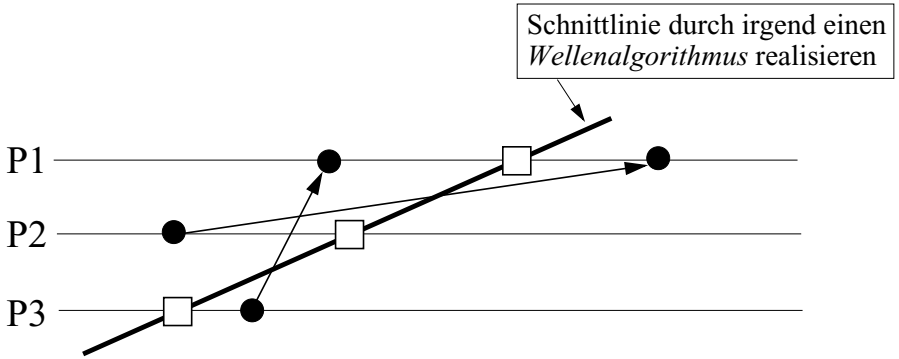
- Falls Prädikat stabil: "entdecken" dieses Prädikat (==> "stable property detection algorithm")

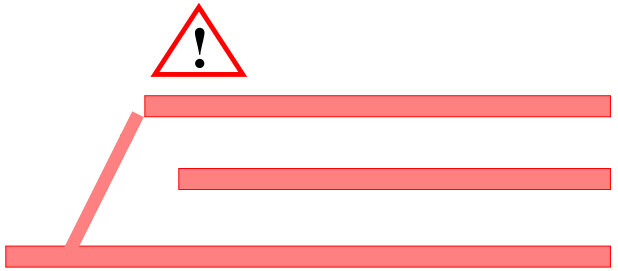
- bei stabilen Prädikaten ist "möglich... vergangen" sogar brauchbar! (es gilt dann jetzt sicherlich)

- aber wenn die betrachtete Eigenschaft nicht stabil ist, was dann?

- wer garantiert eigentlich, dass eine Eigenschaft (wirklich) stabil ist?

Ein Schnappschussalgorithmus





"morgige Zeitung nicht heute lesen"

Prozesse, Nachrichten: *schwarz* oder *rot*

Schnappschussaugenblick: schwarz --> rot
(dann: lokalen Zustand dem Initiator melden)

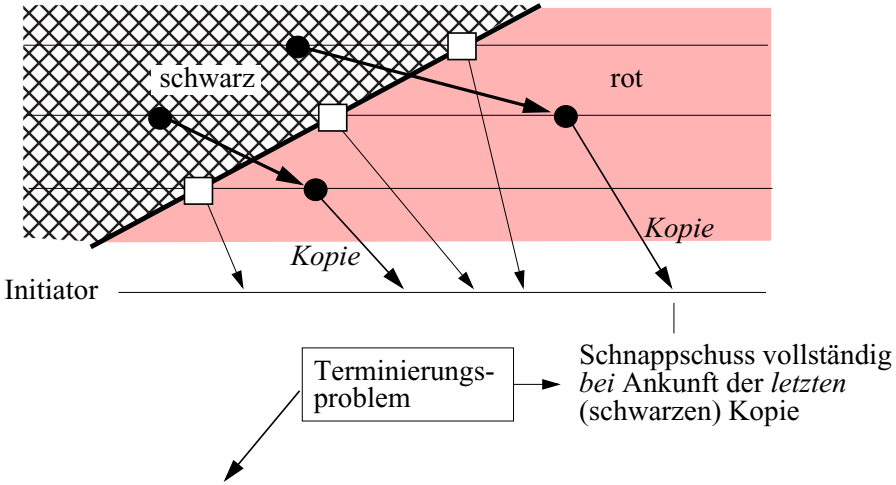
Prozess wird rot, wenn a) Aufforderung erhalten,
b) Erhalt einer roten Nachricht

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