Probabilistische Algorithmen

- Der klassische "totale" Korrektheitsbegriff von Algorithmen kann auf zweierlei Weise abgeschwächt werden:

---

1. Sogenannte *Las Vegas-Algorithmen:*

   - Abschwächung der Terminierungsforderung
   - also: "Partiell korrekt und Terminierung mit *Wahrscheinlichkeit* 1"
   - beachte: die (worst-case) Laufzeit solcher Algorithmen ist unbeschränkt!
   - Beispiel: obiger Election-Algorithmus für anonyme Ringe

---

2. Sogenannte *Monte Carlo-Algorithmen:*

   - Abschwächung der partiellen Korrektheit
   - "terminiert stets, ist aber nur mit Wahrscheinlichkeit $p < 1$ partiell korrekt"
   - also: $\exists$ *Restwahrscheinlichkeit* $\varepsilon = 1\text{-}p > 0$, dass das Ergebnis falsch ist!
   - nur verwenden, wenn:
     - ε sehr *klein* ist (oft: als Parameter des Algorithmus, etwa abhängig von der Laufzeit und damit "beliebig klein" *wählbar*)
     - dadurch deutliche Vorteile erzielbar (Problem effizienter oder überhaupt erst lösbar)
   - beachte den "Sonderfall" $p=1$ (also $\varepsilon=0$): ein solcher Monte Carlo-Algorithmus wäre *total korrekt* (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 = random (1,...,n), mit n = 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:

    FIFO-Kanäle notwendig?

    - nur diese führen eine *neue Election-Runde* durch
    - daher enthalten Nachrichten auch eine Rundenkennung (alte Nachrichten werden in der nächsten Runde einfach ignoriert)

- Ohne Beweis: *Erwartungswert bzgl. Rundenzahl* $\leq e\ (n/n-1)$
  - vgl. mit vorherigem Algorithmus für Ringgrösse 2

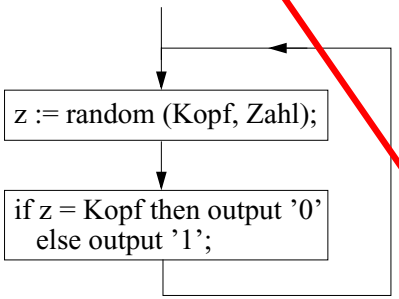
  2.718281828...

- 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.

Zufällige reellwertige Zahlen?

- Wenn man im Verfahren von Itai/Rodeh zufällige reelle Zahlen (z.B. zwischen 0 und 1) für die id wählen könnte...
  - wie hoch wäre dann die Wahrscheinlichkeit, dass zwei Prozesse sich für die gleiche Identität entscheiden?
  - wie hoch wäre dann die Rundenzahl bzw. die Nachrichtenkomplexität?

- Realisierung solcher Zufallszahlengeneratoren?



- liefert eine unendliche Folge von 0en und 1en
- Verwende diese als die Nachkommastellen von $0.\cdots$ im Dualsystem
- Zurückführung des Problems auf perfekten binären Zufallsentscheider

- Unendliche Folgen lassen sich aber nicht in endlicher Zeit generieren und mit endlich vielen Bits speichern...

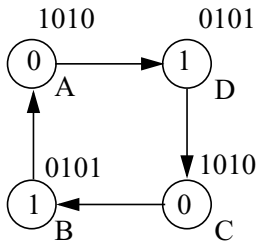
- Wie wäre es statt dessen mit einer "lazy" Variante, die notwendige Nachkommastellen nur auf Anfrage produziert?
  - etwa: liefere zunächst nur die ersten 32 Nachkommastellen; wenn mehr benötigt werden, fordert man das Objekt "zufällige reellwertige Zahl zwischen 0 und 1" auf, weitere Stellen zu liefern
  - Denkübung: hilft das (grundsätzlich) bei unserem Problem?

Genügt ein einziges Zufallsbit?

- 1993 haben S. Kurtz, C. Lin, S. Mahaney einen anderen Las-Vegas-Algorithmus für das Election-Problem vorgestellt, der hier *grob skizziert* wird:

für Ringe bekannter Grösse

  - jeder Knoten bestimmt *ein* zufälliges Bit und sendet es nach "rechts"
  - jeder Knoten leitet n-1 Mal ein empfangenes Bit nach "rechts" weiter
  - danach hat jeder Knoten *alle* n Bits (zyklisch verschoben!) gesehen
  - jeder Knoten prüft, ob bei "Ringshift" *seines* gesehenen Bitstrings dieser mit weniger als n shifts erhalten wird (nichttrivialer Automorphismus)
  - falls ja (selten!) --> gesamter Algorithmus wird wiederholt
  - falls nein: unter den n verschiedenen Bitstrings gibt es genau einen maximalen (bei Interpretation als Dualzahl)
  - der eindeutige Prozess, der diesen gesehen hat, ist der Leader



- symmetrische Lösung: alle Knoten führen den gleichen Algorithmus aus
- "common sense of orientation" wird vorausgesetzt
- in nebenstehendem Szenario ging es nicht gleich in der ersten Runde gut...

Beachte: die Laufzeit des Algorithmus ist prinzipiell unbegrenzt; wenn er hält, ist ein Leader allerdings eindeutig bestimmt

- Denkübungen (nicht ganz einfach!):
  - wie hoch ist die Best-case-Nachrichtenkomplexität?
  - macht es einen Unterschied, ob die Ringgrösse eine Primzahl ist?
  - kann man die Wahrscheinlichkeit abschätzen, dass eine einzige Runde (für eindeutiges Maximum) genügt?
  - wie hoch mag die *erwartete* Nachrichtenkomplexität sein?

Schätzung der Ringgrösse?

- Für den vorherigen Algorithmus ist die Kenntnis der Ringgrösse n entscheidend.

- Bei *unbekannter Ringgrösse* lässt sich diese aufgrund von zyklischen Wiederholungen im Bitstring mit wenigen Läufen mit hoher Wahrscheinlichkeit korrekt schätzen...

  - Wieviele Läufe sind für eine gewisse Sicherheit notwendig?
  - Konsequenzen, wenn man sich unerkanntermassen irrt?

Wir wollen das an dieser Stelle nicht weitertreiben...
Aus "philosophischer Sicht" ist allerdings interessant:

- Was ist an minimaler (struktureller) Information notwendig, um Symmetrie zu brechen? (Beispiele: Ringgrösse ist eine unbekannte Primzahl; obere Schranke für die Ringgrösse...)

- Unter welchen minimalen Voraussetzungen ist eine deterministische oder probabilistische Lösung möglich?

- Wie "sicher" und effizient können probabilistische Algorithmen für dieses Anwendungsproblem sein?

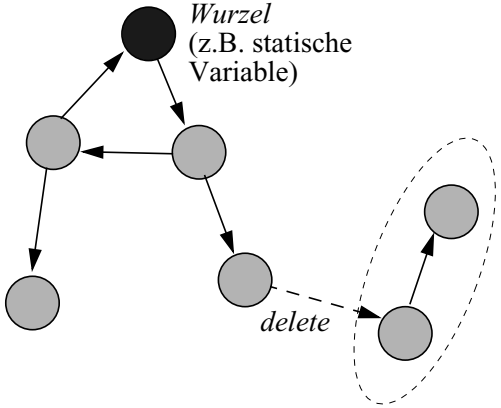
Satz (ohne Beweis): *Für anonyme Netze unbekannter Grösse existiert kein (det. oder prob. Las Vegas) Election-Algorithmus.*

- Daher stellt sich die Frage, ob die Grösse zumindest mit hoher Wahrscheinlichkeit korrekt abgeschätzt werden kann!

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

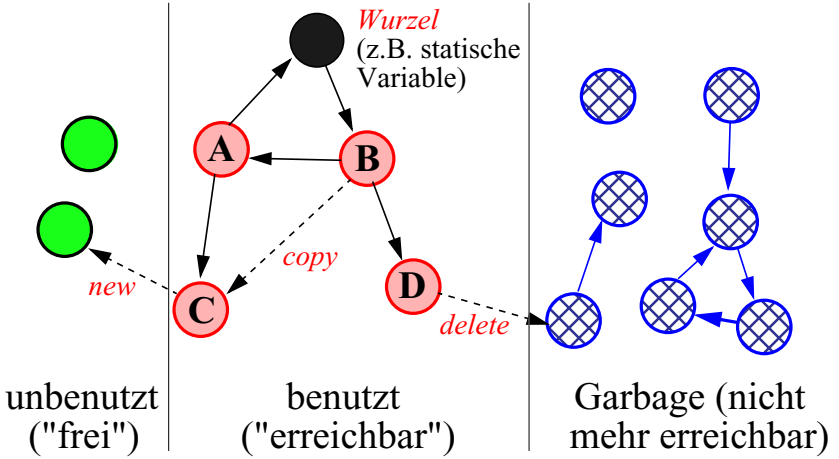
- Typischerweise läuft auch der Garbage-collector (echt oder im Zeitmultiplex) parallel zur Anwendung

⇒ 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* ⇔ ∃ Pfad von der Wurzel dorthin
- "Garbage sein" ist stabiles Prädikat (vgl. Terminierung!)
- *Mutator* ↔ *Collector* spielen mit-/gegeneinander

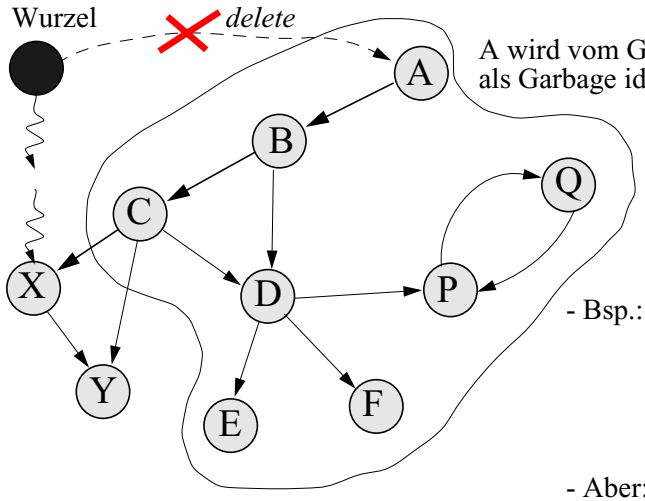
Anwendungsprogramm
(manipuliert Zeiger zwischen Objekten mittels *copy*, *delete* und *new*)

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"



A wird vom Garbage-Collector als Garbage identifiziert

- Bsp.: A Garbage
  → B Garbage
  → C Garbage
  → D Garbage
  → E, F Garbage
- Aber: X, Y kein Garbage!

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

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-Entdeckungverfahren:
  - friere alle Prozese ein
  - ermittle Gesamtzahl der versendeten und empfangenen Nachrichten
  - falls identisch und alle Prozesse passiv sind: terminiert
  - ansonsten: "auftauen" aller Prozesse
- Eingefrorener globaler Zustand ist konsistent!
  - "in aller Ruhe" ein Objekt nach dem anderen betrachten

"Behind the back copy"-Problem

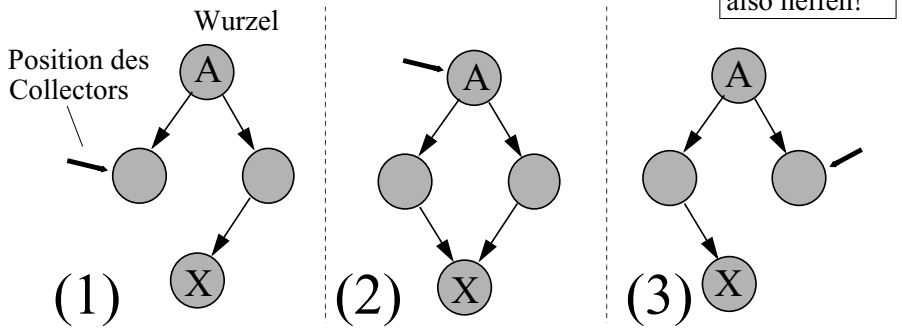
*Concurrent / parallel / on-the-fly-G*arbage-Collection:

- 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!

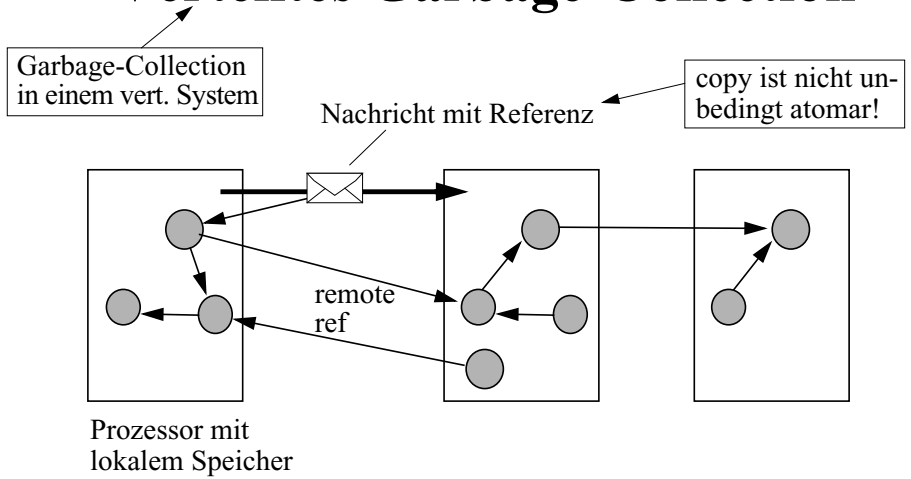
Mutator muss dem Collector also helfen!



- 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

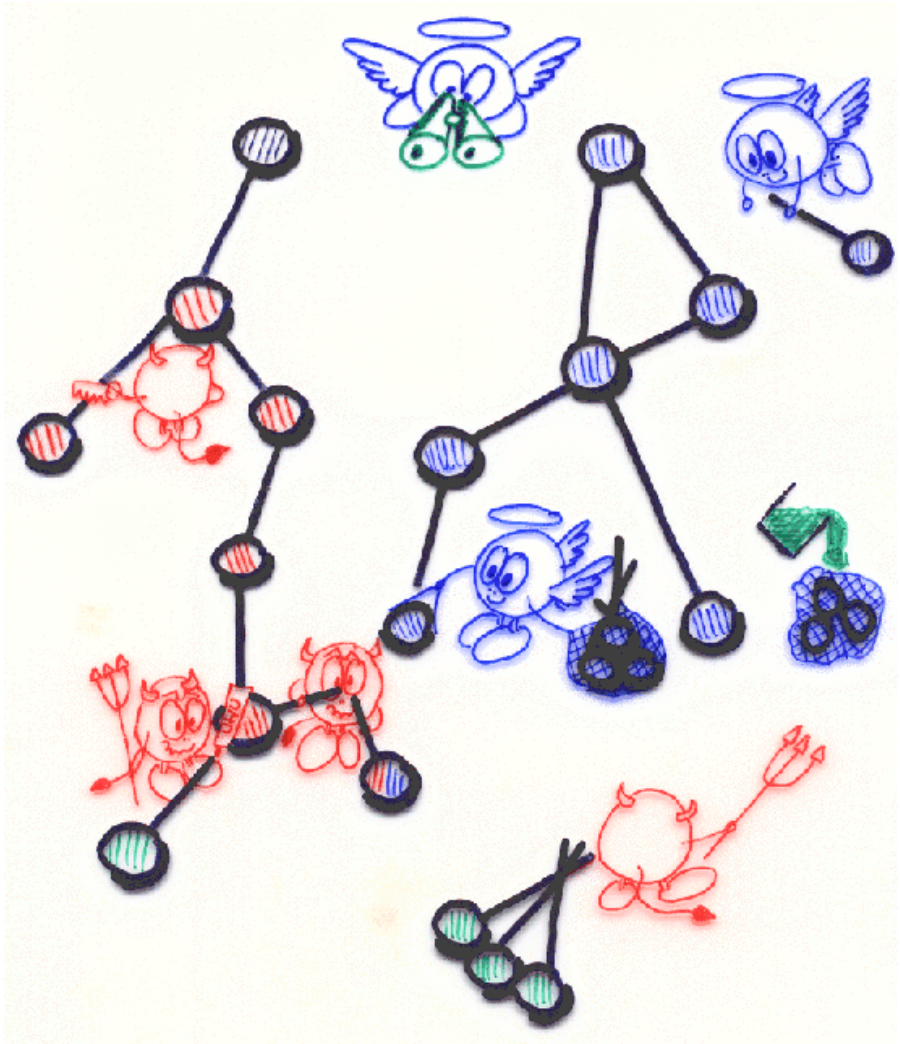


(1) Prozessorüberschreitende Referenzen ("*remote ref*")

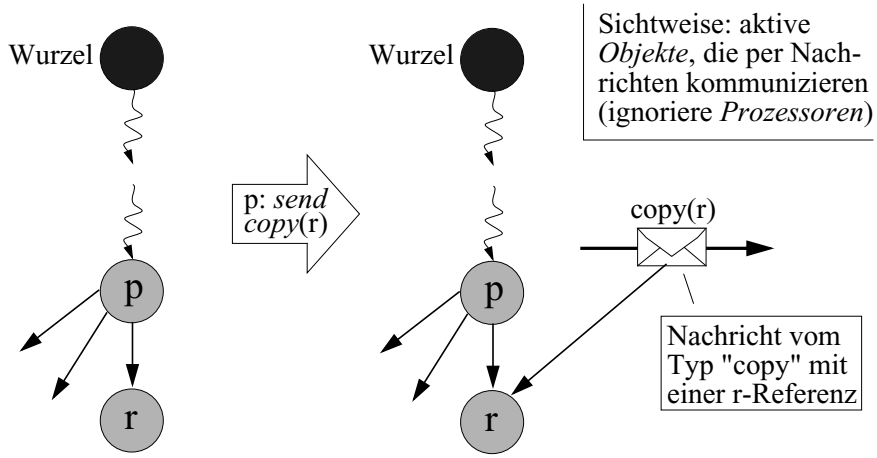
(2) Nachrichten enthalten (Objekte mit) Referenzen auf andere Objekte ("*remote copy*")

- 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)

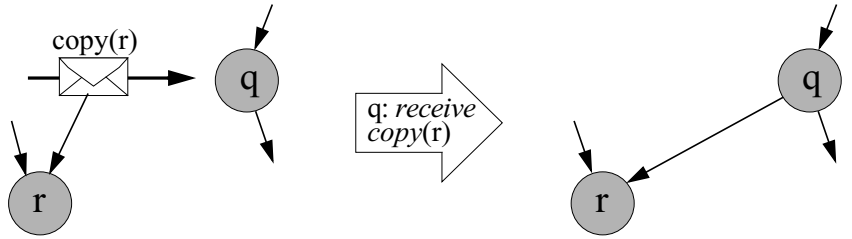


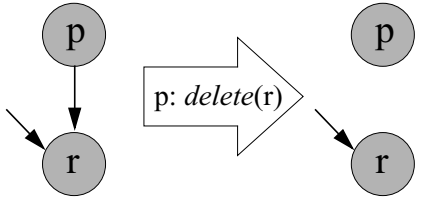
Wirkung der Mutator-Operationen



Sichtweise: aktive *Objekte*, die per Nachrichten kommunizieren (ignoriere *Prozessoren*)

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

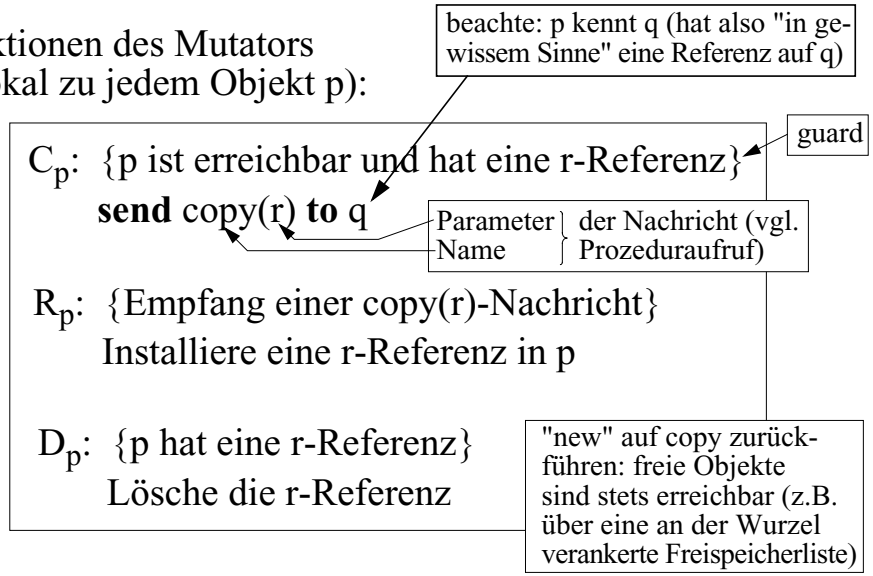




- "new" hier nicht relevant
- jede Aktion ändert den globalen Graphen "etwas"
- Folge solcher Änderungen →" Berechnung"
- hier: "Interleaving-Modell": Operationen sind atomar ("zeitlos") → es gibt keine gleichzeitige Aktionen → verschränkte Ausführung

Formalisierung des verteilten Garbage-Collection-Problems

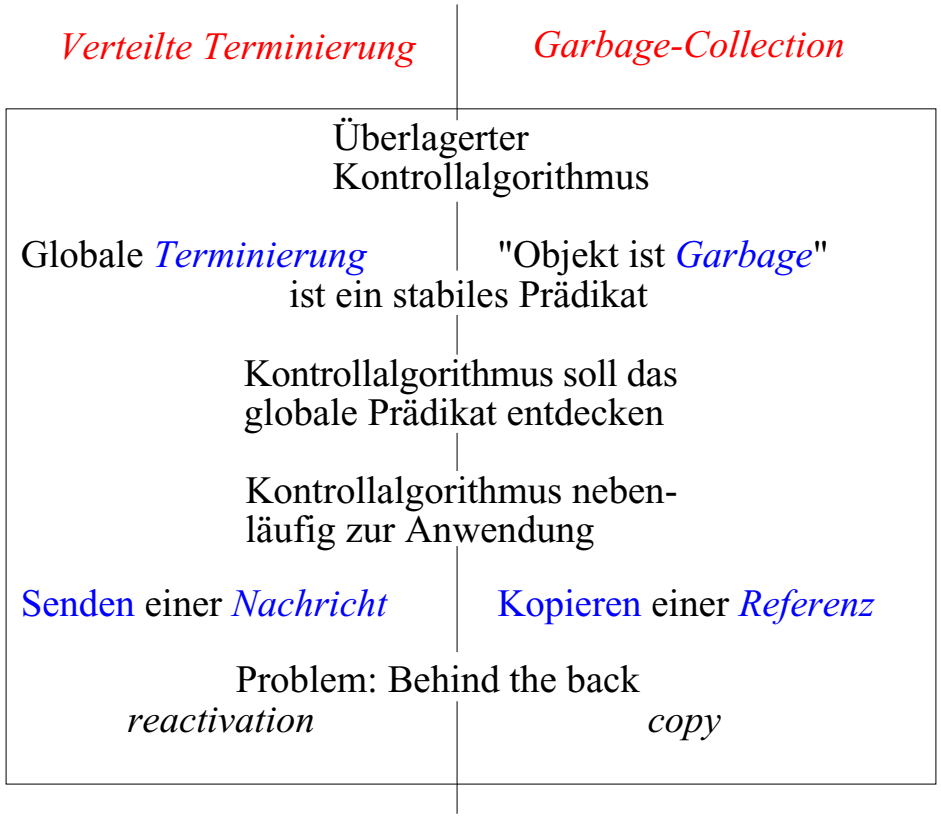
Aktionen des Mutators
(lokal zu jedem Objekt p):



- So "sieht" der Collector die Basisberechnung
- Vergleiche dies mit Basisaktionen beim Problem der verteilten Terminierung!
- 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

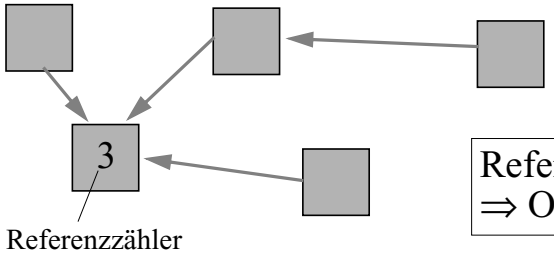
- Interessante Analogie zwischen beiden Problemen:



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

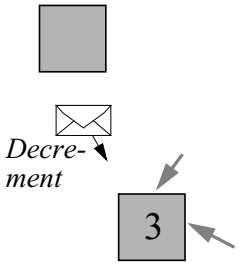
Referenzzähler-Verfahren

- 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



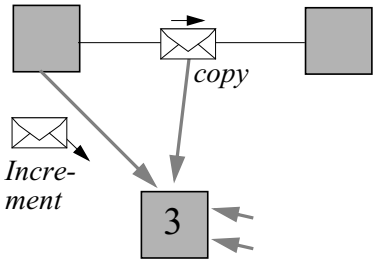
Referenzähler = 0
⇒ Objekt ist Garbage

Löschen einer Referenz:



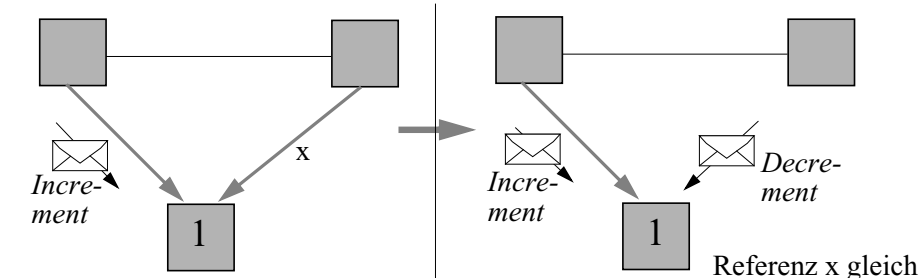
3 ⇒ 2

Kopieren einer Referenz:



3 ⇒ 4

Kopieren kann zu *Fehlinterpretationen* führen!



Increment-Nachricht bzgl.
Referenz x ist noch unterwegs

Referenz x gleich
wieder gelöscht!

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

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

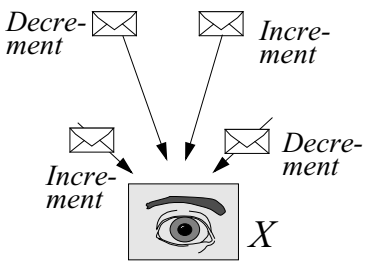
Wie dies garantieren?

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

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

→ (indirekte) Überholungen vermeiden

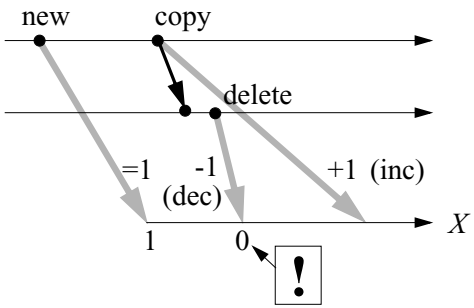
Garbage-Identifikation als konsistentes Beobachtungsproblem



Objekt X "beobachtet" die copy- und delete-Operationen, die es selbst betreffen

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

- Diese Beobachtung sollte kausaltreu sein!
  - zumindest darf ein dec nicht vor "seinem" inc empfangen werden

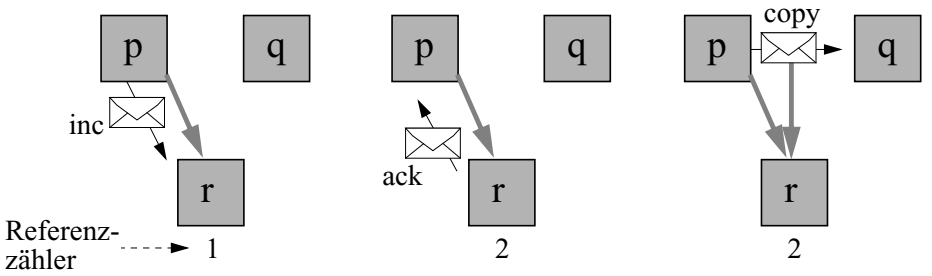


- *Jedes* Objekt beobachtet jedes andere
  - "Causal Order" (Globalisierung von FIFO) bzgl. Kommunikation gefordert

- 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):

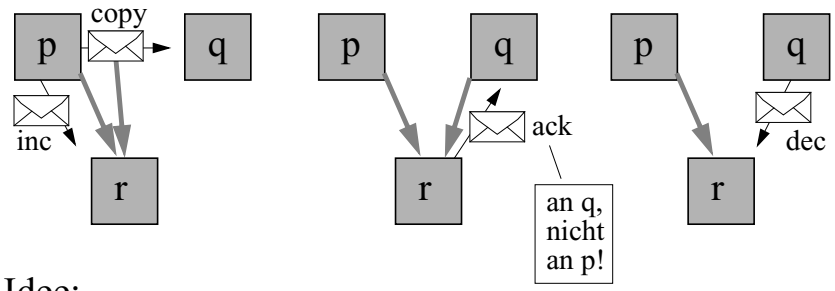


- Korrektheitskriterium erfüllt:
  inc wird vor einem kausal abhängigen dec empfangen
  (d.h. Objekt r erfährt über die Existenz einer Kopie einer r-Referenz, bevor es vom Löschen dieser oder einer "solchen" Referenz erfährt)

- 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

---

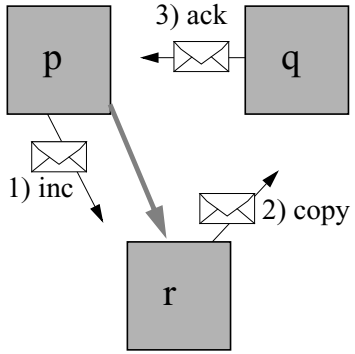
- Implementierungsskizze:
  - Zählen von empfangenen copy und ack-Nachrichten
  - dec-Nachricht erst senden, wenn Zähler ACK und COPY übereinstimmen
    - dann die Zähler ACK und COPY beide dekrementieren
    - "individuelle" Zuordnung von copy zu ack nicht notwendig!
    - Abschwächung $|ACK|>0 \wedge |COPY|>0$ möglich? Konsequenzen?

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

wieso?

Varianten von Rudalics

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



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

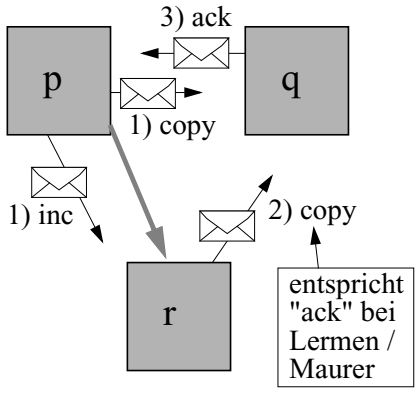
Bedingung: p darf seine r-Referenz erst löschen, wenn alle erwarteten ack-Nachrichten eingetroffen sind.

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:



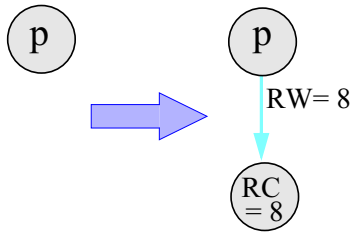
- Idee: ack erst senden, wenn *beide* copy-Nachrichten empfangen wurden
- q installiert die r-Referenz bei Empfang der *ersten* copy-Nachricht
- Unter welchen Bedingungen dürfen p bzw. q dec-Nachrichten senden?

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

Die Referenzgewichts-Methode

(WRC: "Weighted Reference Counting")

Neues Objekt generieren ("new"):

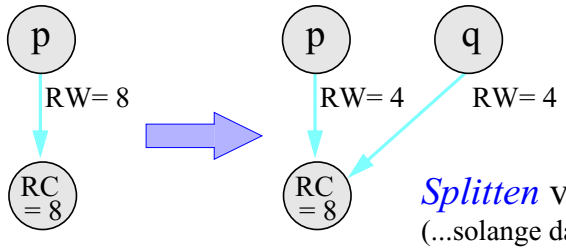


Reference *Weight*

Reference *Count*

(obwohl dadurch die Referenzen nicht tatsächlich "gezählt" werden)

Referenz kopieren ("copy "):



Splitten von RW

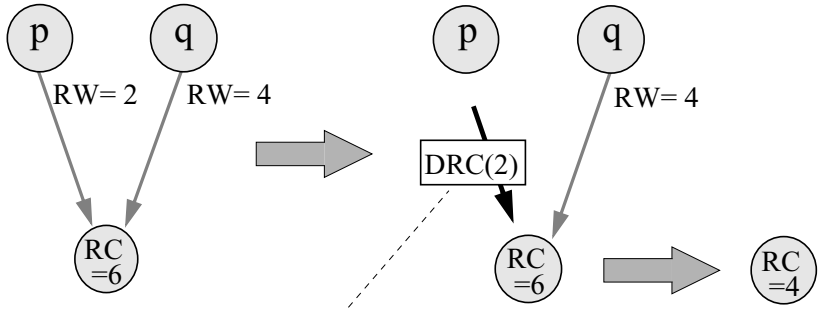
(...solange das geht!)

ansonsten (hier nicht genauer behandelt):

1) "Recharge"
2) Indirektionsobjekt

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

Referenz löschen ("delete"):



Decrement-Reference-Count-Nachricht

Invariante: RC = ∑ RW + ∑ DRC

RC = 0 → Objekt ist Garbage

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

- Logarithmische Kompression (2er-Potenzen!) von RW
  - mit nur 2 Bit pro Zeiger lassen sich so RW bis max. 8 darstellen
  - statt 8 kann ggf. auch ein (etwas?) grösserer Maximalwert gewählt werden
  - RC so nicht komprimierbar ⇒ 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 (→ Zusatzaufwand)
- Analogie zur *Kredit-Methode* bei vert. Terminierung!

Kredit-Methode und WRC

Terminierungserkennung mit der Kreditmethode		WRC-Garbage-Collection
- Senden einer Nachricht (Splitten des Kreditwertes)	⟷	- Kopieren einer Referenz (Splitten des RW)
- Passiv werden (Kredit an Urprozess zurückgeben)	⟷	- R-Referenz löschen (Decrement-Nachricht an R)
Invariante: ∑ Kredit =1	⟷	*Invariante*: RC = ∑ RW + ∑ DRC
- Gesamtkredit beim Urprozess = 2^0=1	⟷	- RC = 0 bei R
Terminierung	⟷	*R ist Garbage*

Also: Kreditmethode entspricht WRC-Garbage-Collection!

Garbage-Collection und Terminierung

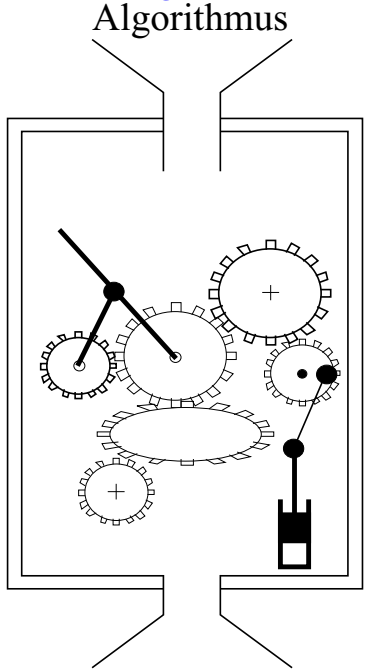
Theorem:

nicht notwendigerweise verteilte

Jeder *Garbage-Collection*-Algorithmus kann automatisch in einen Algorithmus zur Feststellung der *verteilten Terminierung* transformiert werden

Bemerkung: Für beide Probleme wurden viele nicht-triviale (und auch manche falsche!) Lösungen publiziert

Ein Garbage-Collection-Algorithmus

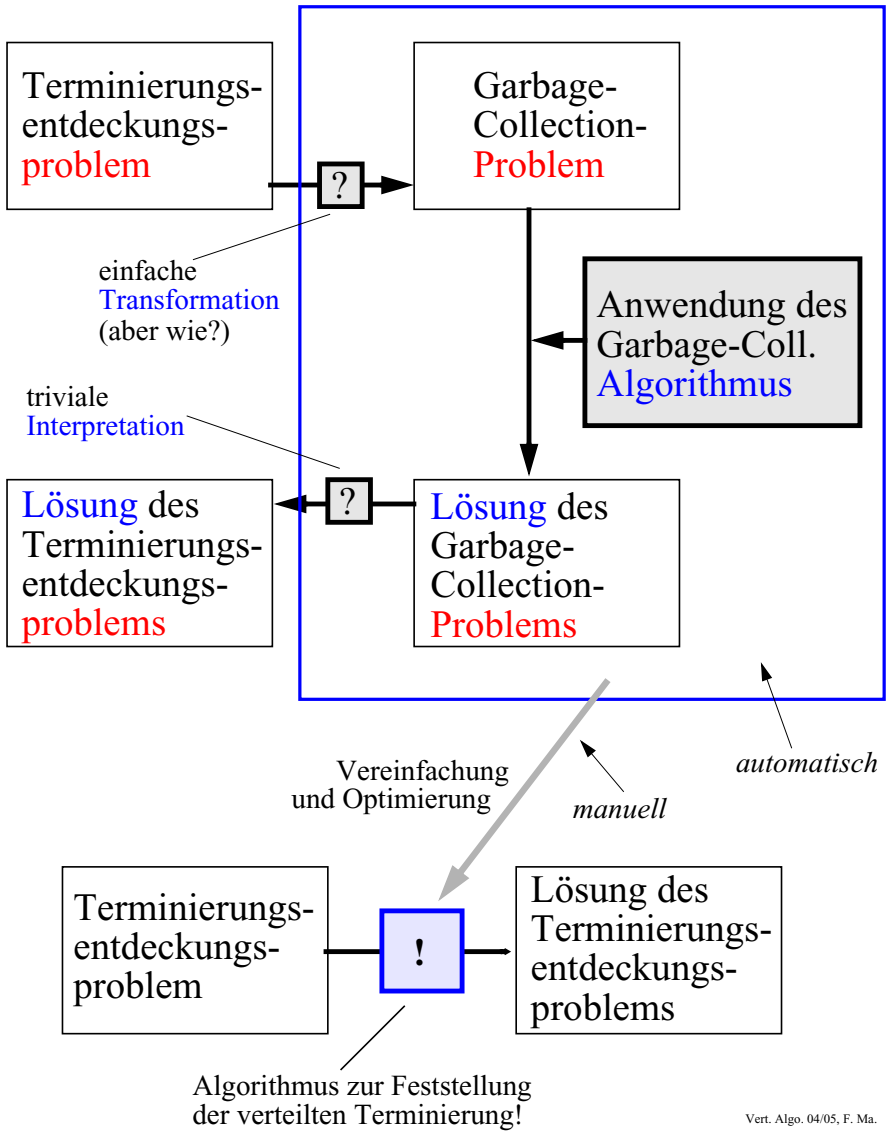


mechanische Transformation

Ein verteilter Terminierungsentdeckungs-Algorithmus

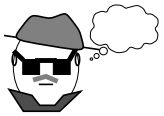
Problemtransformation

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

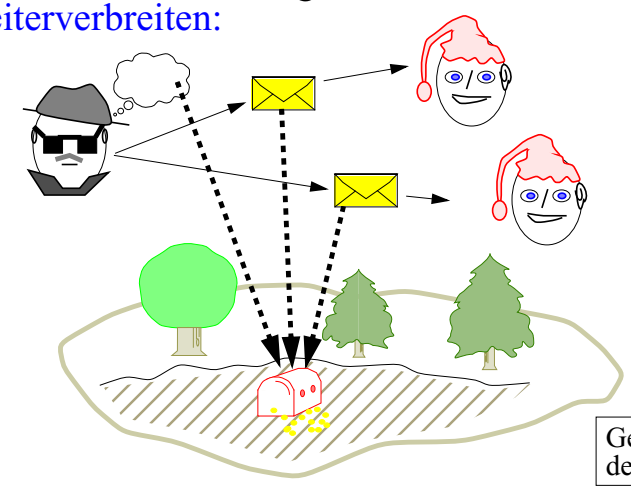


Vom Ende einer Geheimniskrämerei

Die vier goldenen Regeln der Geheimniskrämerei:

1) Es gibt Geheim-nisträger  und unein-geweihte Personen 

2) Nur ein Geheimnisträger kann das Geheimnis weiterverbreiten:



3) Wer das Geheimnis erfährt, wird zum Geheimnisträger

4) Ein Geheimnisträger kann das Geheimnis (endgültig) vergessen

---

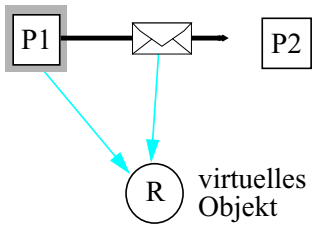
es gibt keine Geheimnisträger und keine Nachrichten mit dem Geheimnis

Geheimniskrämerei terminiert ⇔
Schatz nicht mehr zugreifbar ⇔ *Schatz ist "Garbage"*

→ "watchdog" beim Schatz meldet Terminierung...

Die Transformation

- Jeder Prozess wird in ein Wurzelobjekt transformiert
- Ein zusätzliches virtuelles Objekt R wird hinzugefügt



1) Prozess P aktiv ⇔
P besitzt Referenz auf R

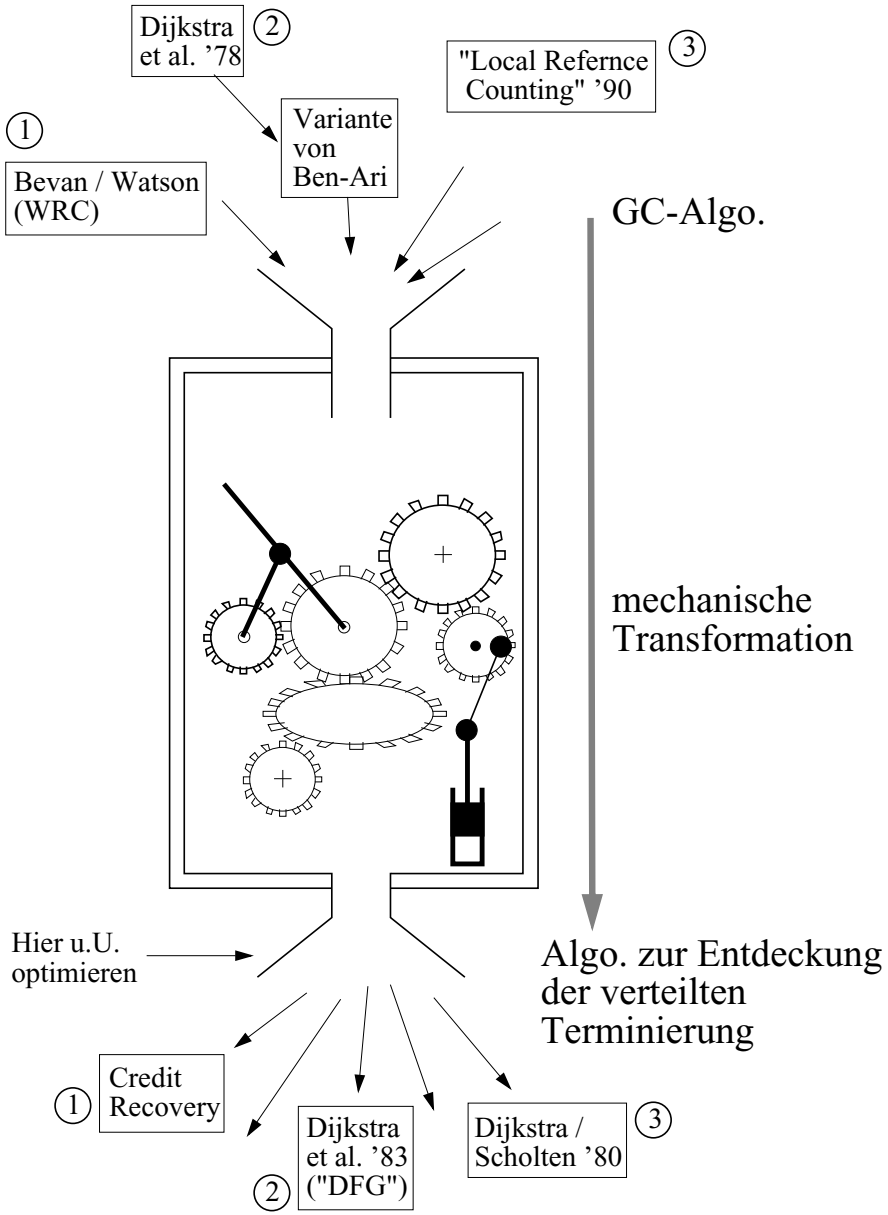
2) Jede Nachricht enthält
Referenz auf R

- Die beiden Regeln lassen sich ("induktiv") erfüllen:
  - ein (aktives) Objekt / Prozess sendet eine Kopie seiner R-Referenz mit jeder Nachricht
  - ein reaktivierter Prozess erhält eine R-Referenz
  - ein Prozess, der passiv wird, löscht seine R-Referenz

R Garbage ⇔ Es gibt keine Referenz auf R
⇔ Alle Prozesse passiv und keine Nachricht unterwegs ⇔ Verteilte Berechnung terminiert

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

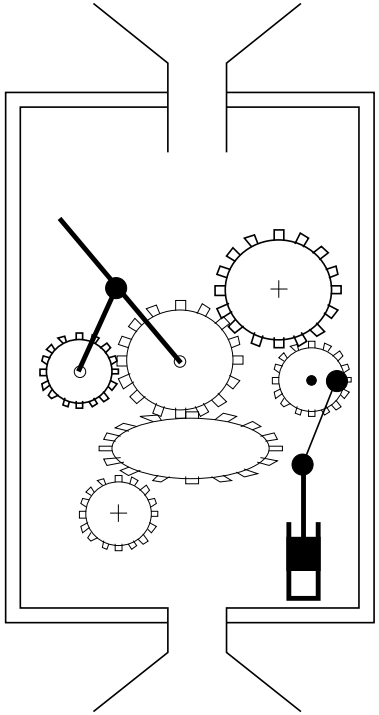
- Übung: man mache dies für konkrete GC-Algorithmen aus der Literatur
- beachte: es entstehen keine Zyklen von Referenzen, daher sind auch Referenzzählverfahren anwendbar!



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

Die Patent-Story

WRC-Garbage-Collection-Algorithm patentiert:
Europäische Patentnummer 86309082.5



Ist der resultierende Terminierungs-erkennungsalgorithmus auch durch das Patent geschützt?

Das WRC-Patent

Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 225 755**
A2

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number: 86309082.5

(22) Date of filing: 20.11.86

20.11.86

(51) Int. Cl.: **G 06 F 12/02**

ICL

(30) Priority: 04.12.85 GB 8529890

(71) Applicant: INTERNATIONAL COMPUTERS LIMITED ICL House, Putney, London, SW15 1SW (GB)

Watson, Paul

(43) Date of publication of application: 16.06.87 Bulletin 87/25

(72) Inventor: Watson, Paul, 146, Hilda Park Chester-Le-Street, Co-Durham DH2 2JY (GB)

(84) Designated Contracting States: BE DE FR GB NL

(74) Representative: Guyatt, Derek Charles et al, STC Patents Edinburgh Way, Harlow Essex CM20 2SH (GB)

(54) Garbage collection in a computer system

(57) A computer system is described, having memory cells organised in a directed graph structure by means of pointers. Each cell has a reference count, and each pointer a weight value. If a new pointer to a cell is created by copying an existing pointer, the new and existing pointers are given weights whose sum equals the old value of the existing pointer. In this way, the sum of the weights of the pointers to any cell are maintained equal to its reference count. If a pointer is destroyed, the reference count of the cell to which it points is reduced by the weight of the pointer. Thus, when the reference count of a cell reaches zero, it is safe to assume that there are no more existing pointers to it, and hence that cell may be reclaimed (garbage-collected) for re-use.

nächste Folie →

(84) Designated Contracting States: BE DE FR GB NL

(54) Garbage collection in a computer system

(57) A computer system is described, having memory cells organised in a directed graph structure by means of pointers. Each cell has a reference count, and each pointer a weight value. If a new pointer to a cell is created by copying an existing pointer, the new and existing pointers are given weights whose sum equals the old value of the existing pointer. In this way, the sum of the weights of the pointers to any cell are maintained equal to its reference count. If a pointer is destroyed, the reference count of the cell to which it points is reduced by the weight of the pointer. Thus, when the reference count of a cell reaches zero, it is safe to assume that there are no more existing pointers to it, and hence that cell may be reclaimed (garbage-collected) for re-use.

Invariante!

Ein hierarchisches Patent

CLAIMS:-

1. A *computer system* having storage means containing a plurality of memory cells, at least some of which contain pointers to others of the cells thereby defining a directed graph structure in which each cell represents a node of the graph, wherein

(a) each pointer has a *variable weight value* associated with it,

(b) each cell contains a variable reference count value,

(c) whenever a cell is *allocated* in the storage means, its reference count is initially set to a predetermined value and the weights of any pointers to that cell are set to non-zero values such that the sum of those weights (or the weight if there is only one such pointer) is equal to the reference count of that cell,

(d) whenever a new pointer to a cell is created by *copying* an existing pointer, the new pointer and the existing pointer are given non-zero weight values whose *sum equals the old weight value* of the existing pointer, but the reference count of the cell is unaltered,

(e) whenever a pointer is *destroyed*, the reference count of the cell to which it points is reduced by the weight of that pointer, and

(f) cells with reference count equal to zero are reclaimed.

2. A system according to Claim 1 wherein the weight of each pointer is a *power of two*, and wherein, whenever a pointer is created by copying an existing pointer of weight greater than one, the weight of the existing pointer is divided by two and the result is stored as the weight value of the new and existing pointers.

3. A system according to Claim 2 wherein the weight of each pointer is *encoded in a compressed form as the logarithm* to the base two of the weight.

4. A system according to any preceding claim wherein, whenever a pointer is to be created by copying an existing pointer of weight equal to one, a *dummy cell* is created into which the existing pointer is copied along with its weight, the dummy cell is given a reference count greater than one, and the new and existing pointers are both made to point to this dummy cell, and are given non-zero weights whose sum equals the reference count of the dummy cell.

5. A system according to any preceding claim in which the system is a *multi-processor distributed computer system*.

10 CLAIMS

US-Version 4755939: hier wird nun eine *Methode* patentiert, nicht mehr, wie im europäischen Patent, ein *Computersystem*!

I claim:

1. A method of garbage collection in a computer system having storage means containing a plurality of memory cells, at least some of which contain pointers to others of the cells thereby defining a directed graph structure in which each cell represents a node of the graph, wherein the method comprises steps as follows:
   - (a) each pointer is provided with a variable weight value,
   - (b) each cell is provided with a variable reference count value,
   - (c) whenever a cell is allocated in the storage means, the reference count of the cell is initially set to a predetermined value and the weights of any pointers to that cell are set to non–zero values such that the sum of those weights (or the weight if there is only one such pointer) is equal to the reference count of that cell,
   - (d) whenever a new pointer to a cell is created by copying an existing pointer, the new pointer and the existing pointer are given non–zero weight values whose sum equals the old weight value of the existing pointer, but the reference count of the cell is unaltered,
   - (e) whenever a pointer is destroyed, the reference count of the cell to which the pointer points is reduced by the weight of that pointer,
   - (f) and cells with reference count equal to zero are reclaimed.

2. A method according to claim 1 wherein the weight of each pointer is a power of two, and wherein, whenever a pointer is created by copying an existing pointer of weight greater than one, the weight of the existing pointer is divided by two and the result is stored as the weight value of the new and existing pointers.

3. A method according to claim 2 wherein the weight of each pointer is encoded in a compressed form as the logarithm to the base two of the weight.

4. A method according to claim 3 wherein, whenever a pointer is to be created by copying an existing pointer of weight equal to one, a dummy cell is created into which the existing pointer is copied along with its weight, the dummy cell is given a reference count greater than one, and the new and existing pointers are both made to point to this dummy cell, and are given non–zero weights whose sum equals the reference count of the dummy cell.

5. A method according to claim 3 in which the system is a multi–processor distributed computer system.

6. A method according to claim 2 wherein, whenever a pointer is to be created by copying an existing pointer of weight equal to one, a dummy cell is created into which the existing pointer is copied along with its weight, the dummy cell is given a reference count greater than one, and the new and existing pointers are both made to point to this dummy cell, and are given non–zero weights whose sum equals the reference count of the dummy cell.

7. A method according to claim 2 in which the system is a multi–processor distributed computer system.

8. A method according to claim 1 wherein, whenever a pointer is to be created by copying an existing pointer of weight equal to one, a dummy cell is created into which the existing pointer is copied along with its weight, the dummy cell is given a reference count greater than one, and the new and existing pointers are both made to point to this dummy cell, and are given non–zero weights whose sum equals the reference count of the dummy cell.

9. A method according to claim 8 in which the computer system is a multiple–processor distributed computer system.

10. A method according to claim 1 in which the computer system is a multi–processor distributed computer system.

Patentsuche

- Sucht man nach Garbage-Collection-Verfahren (z.B. bei den WWW-Sites der Patentämter), findet man einiges...

Search Results
Query: (garbage collection)
139 of 2569032 matched (Jan. 2000: ein Jahr zuvor: 93 of 2461228)

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

5652883 Computer method and system for conservative-stack and generational heap garbage collection

5561785 System for allocating and returning storage and collecting garbage using subpool of available blocks

5560003 System and hardware module for incremental real time garbage collection and memory management

5446901 Fault tolerant distributed garbage collection system and method for collecting network objects

5819299 Process for distributed garbage collection

5832529 Methods, apparatus, and product for distributed garbage collection

5033930 Garbage collecting truck

5398334 System for automatic garbage collection using strong and weak encapsulated pointers

4755939 Garbage collection in a computer system

5901540 Garden tool for collection and removal of debris

...

Andere Garbage-Collection-Patente

- Es gibt viele Patente zum Thema "Garbage Collection", und manches hätte man vielleicht selbst erfinden können

- Titel: Fault tolerant distributed garbage collection system and method for collecting network objects
- Veröffentlichungsnr.: US5446901
- Veröffentlichungsdatum: 1995-08-29
- Erfinder: OWICKI SUSAN S (US); BIRRELL ANDREW D (US); NELSON CHARLES G (US); WOBBER EDWARD P (US)
- Anmelder: DIGITAL EQUIPMENT CORP (US)
- Klassifikationssymbol (IPC): G06F12/00

A distributed computer system includes a multiplicity of concurrently active processes. Each object is owned by one process. Objects are accessible to processes other than the object's owner. Each process, when it receives a handle to an object owned by any other process, sends a first "dirty" message to the object's owner indicating that the object is in use. When a process permanently ceases use of an object handle, it sends a second "clean" message to the object's owner indicating that the object is no longer in use. Each object's owner receives the first and second messages concerning usage of that object, stores data for keeping track of which other processes have a handle to that object and sends acknowledgement messages in return. The receiver of an object handle does not use the handle until its first message is acknowledged. Periodically, the object's owner sends status request messages to other processes with outstanding handles to that object to determine if any of those processes have terminated and updates its stored object usage data accordingly. A garbage collection process collects objects for which the usage data indicates that no process has a handle. The first and second messages include sequence numbers, wherein the sequence numbers sent by any process change in value monotonically in accordance with when the message is sent. Object owners ignore any message whose sequence number indicates that it was sent earlier than another message for the same object that previously received from the same process.

Andere Garbage-Collection-Patente (2)

- Man findet auch Querverweise auf viele andere Patente, auf wissenschaftliche Literatur dazu etc., z.B.:

5355483 : Asynchronous garbage collection
INVENTORS: Serlet; Bertrand, Paris, France
ASSIGNEES: NeXT Computers, Redwood City, CA
ISSUED: Oct. 11, 1994
FILED: July 18, 1991
SERIAL NUMBER: 732453

...With this method, the process being collected communicates its memory state ("a memory snapshot") to a garbage collecting process (GC), and the GC process scans the memory and sends back the information about garbage. As a result, the present invention permits garbage collection to be performed asynchronously. The process being scanned for garbage is interrupted only briefly, to obtain the memory snapshot. The process then runs without interruption while the garbage collection is being performed. The present invention makes the assumption that if an object is garbage at the time of the memory snapshot it remains garbage any time later,...

Beispiele für Softwarepatente (1)

Es gibt viel ganz offensichtliche Dinge, die patentiert wurden, das gilt auch für Algorithmen. Einiges davon ist in der fachlich beschlagenen Öffentlichkeit bekannt geworden, z.B. das LZW-Komprimierungspatent (US4558302) beim gif-Bildformat. Es gibt aber auch noch viele andere "interessante" Patente. Vieldiskutiert ist die Frage, ob Algorithmen (oder ganz allgemein "Intellectual Property") überhaupt patentiert werden soll.

Any word processor with a separate mode that the user selects when they wish to type in a mathematical formula. [US5122953]

A word processor which marks and makes correction to a document using two additional different colors. [US5021972]

Inventor(s): Nishi; Toshio
June 4, 1991
A word processor, including a keyboard through which characters can be inputted, a memory device for storing inputted character arrays and a display device capable of multi-color displays, is so programmed that corrections and additions are automatically displayed in a different color from the rest for the convenience of editing. ... When a completed document is finally stored in a document file, however, such color codes are deleted such that the document can be outputted in one color.

Statically allocating an initial amount of memory when a program is first loaded according to a size value contained in the program header. [US5247674]

Applicant(s): Fujitsu Limited, Kawasaki, Japan
Issued/Filed Dates: Sept. 21, 1993 / May. 13, 1991
A memory allocation system includes a unit for storing the information about the amount of memory required at the time of initializing each executable program in the control information of the file storing the program. The amount required is determined when the program is translated, assembled or compiled and linked. The memory allocation system also includes a unit for reading the information, indicating the amount of memory required at the time of initializing the program stored in the control information of the file, when loading of program is requested.

Beispiele für Softwarepatente (2)

Assigning a client request to a server process by first examining all the server processes not handling the maximum number of clients, and then assigning it to the server process currently servicing the fewest clients. [US5249290]

Applicant(s): AT&T Bell Laboratories, Murray Hill, NJ
Issued/Filed Dates: Sept. 28, 1993 / Feb. 22, 1991
A server of a client/server network uses server processes to access shared server resources in response to service requests from client computers connected to the network. The server uses a measured workload indication to assign a received client service request to a server process... a busy indicator provides a measured workload indication for each active process. The server uses the busy indicator to assign a new client service request to the least busy process.

Method for canonical ordering of binary data for portable operating systems. [US4956809]

Applicant(s): Mark Williams Company, Chicago, IL
Issued/Filed Dates: Sept. 11, 1990 / Dec. 29, 1988
...The method includes converting all binary data accessed from a file or communications channel from the canonical order to the natural order of the host computer before using the binary data in the host computer and converting all binary data which is to be sent to a file or communications channel from the natural order of the host computer to the canonical order before sending the binary data.

Remembering file access behavior and using it to control the amount of read-ahead the next time the file is opened. [US5257370]

Using of multiple read only tokens and a single read-write token to control access to a portion of a file in a distributed file system. [US5175851]

Quicksort implemented using a linked list of pointers to the objects to be sorted. [US5175857]

Intercepting calls to a network operating system by replacing the first few instructions of an entry point by a call to an intercept routine. [US5257381]

Beispiele für Softwarepatente (3)

Distinguishing nested structures by color. [US4965765]

Applicant(s): International Business Machines Corp., Armonk, NY
Issued/Filed Dates: Oct. 23, 1990 / May. 16, 1986
A method of distinguishing between nested expressions, functions, logic segments or other text by using a different color for each nesting level.

Das berüchtigte "xor-Cursor-Patent": **Method for dynamically viewing image elements stored in a random access memory array.** [US4197590]

Issued/Filed Dates: April 8, 1980 / Jan. 19, 1978
...An XOR feature allows a selective erase that restores lines crossing or concurrent with erased lines. The XOR feature permits part of the drawing to be moved or "dragged" into place without erasing other parts of the drawing. ...supporting another image on the display without destruction of the initially stored image... logically exclusively ORing together the accessed data for each element of the stored image and the data for the corresponding element of the image to be superimposed, and for reentering the resultant logical data into the same memory locations, said display then being generated from the resultant contents of said memory.

A parallelizing compiler that estimates the execution time for each of a number of different parallelization conversions and then selects the one that it thinks will be the fastest. [US5151991]

Any document storage system that has a digital camera to scan in documents, stores the documents on an optical disk, and uses character recognition software to construct an index. [US4941125]

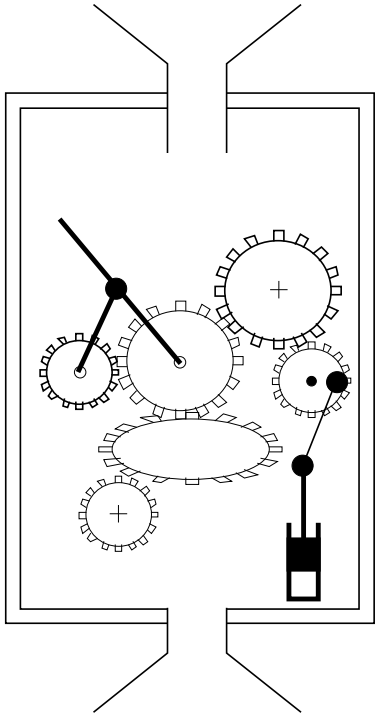
Siehe auch zu obigen Beispielen:
www.base.com/software-patents/examples.html

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

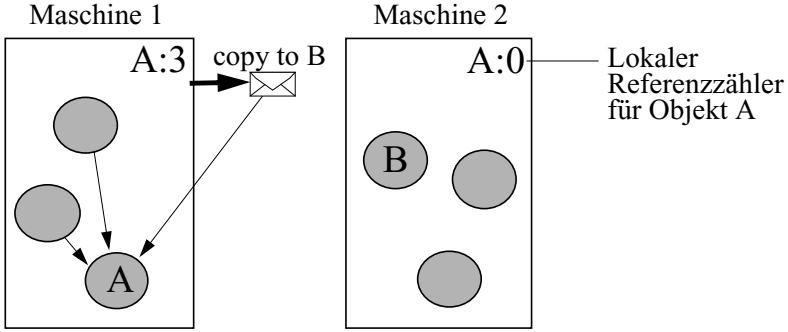
J. Piquer: *Indirect Reference Counting,* Proc. PARLE 91, 150-165

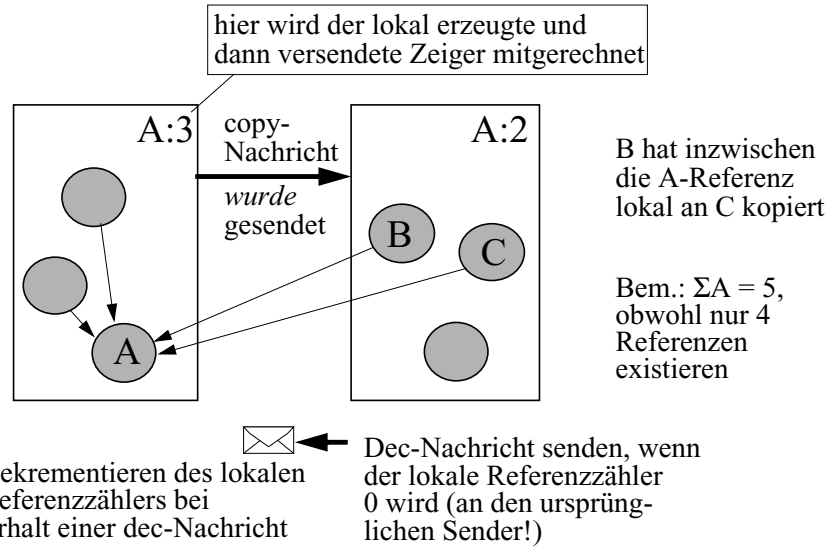


E.W. Dijkstra, C.S. Scholten: Termination Detection for Diffusing Computations. Inf. Proc. Lett. 11 (1980), 1-4

LRC-Garbage-Collection-Verfahren

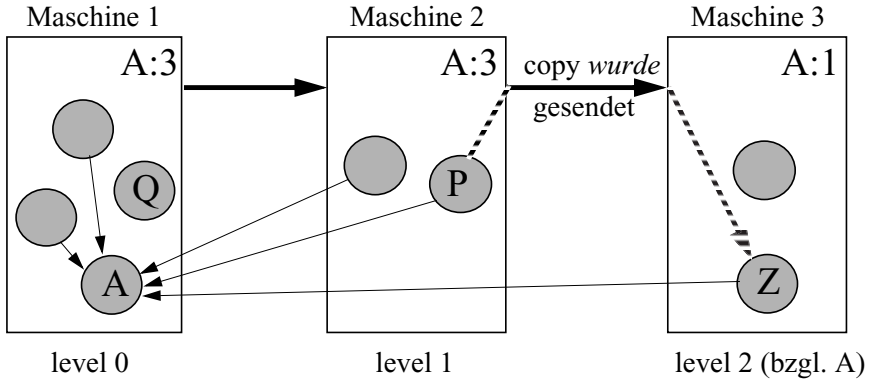
- Pro *Maschine* wird (potentiell) für jedes Objekt ein Referenzzähler gehalten, z.B. für Objekt A:





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?

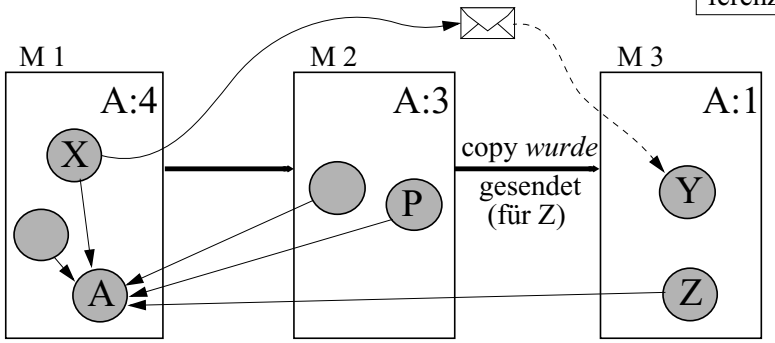
  ⇒ "Garbage-Kriterium": Referenzzähler auf level 0 = 0

- 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?

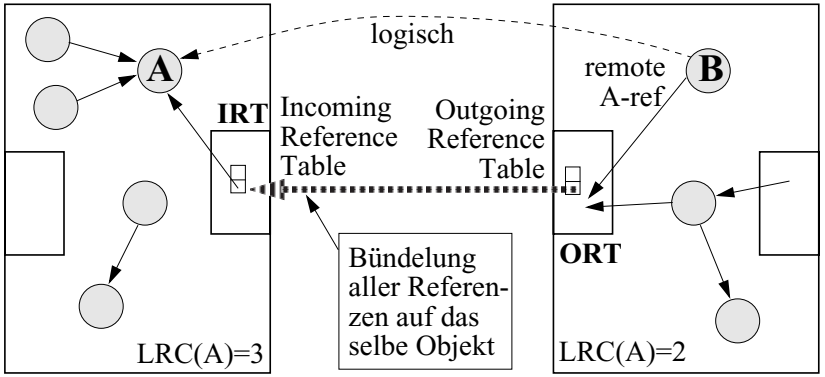
  ? (1) wenn Y seine A-Referenz löscht → an M1 senden, und wenn Z seine A-Referenz löscht → an M2 senden

  ? (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?)

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)

Verteilte Berechnungen

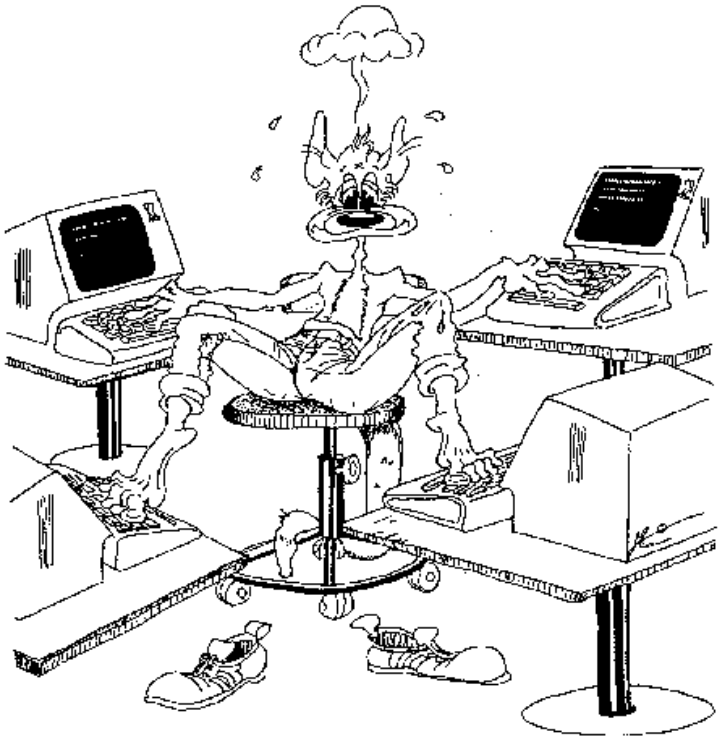


Bild: R. G. Herrtwich, G. Hommel

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!

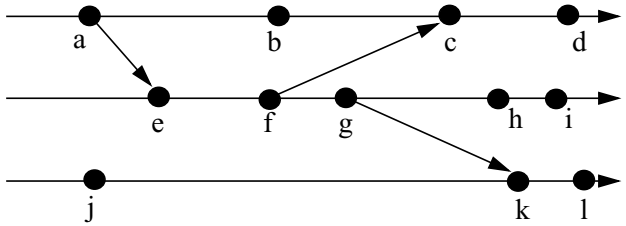
Gummiband-transformation

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

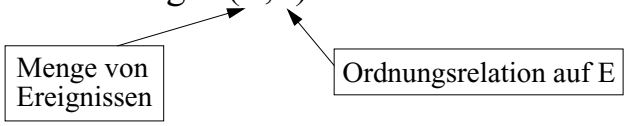
- Wieso ist das eine partielle *Ordnung*?
  - d.h. wieso ist die Relation "gerichtet" und zyklenfrei?
  - oder muss man das (zur Def. von "Berechnung") axiomatisch fordern?

- 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

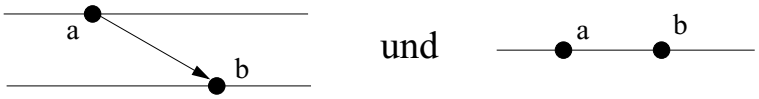
Erster Definitionsversuch

- Verteilte Berechnung = (E,<)



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

Beachte:



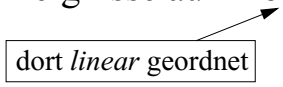
wären identisch!

⇒ *Prozesse* einführen

- Was sind Prozesse (formal / abstrakt)?

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

  - Interpretation: E_i = Ereignisse auf Prozess P_i

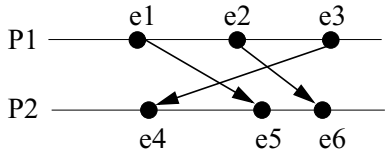


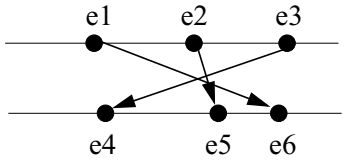
Zweiter Definitionsversuch

- 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

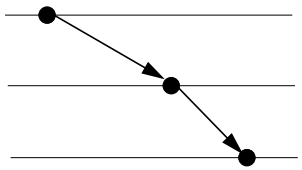
  4) $(s,r) \in \Gamma \Rightarrow s < r$

  5) $<$ ist eine irreflexive Halbordnung auf E

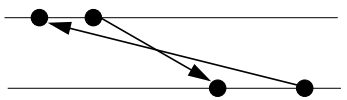
  6) $<$ ist die kleinste Relation, die 3) - 5) erfüllt
  (d.h.: es stehen keine weiteren als die dadurch festgelegten Ereignisse in <-Relation zueinander)

  [*Kausalrelation*]

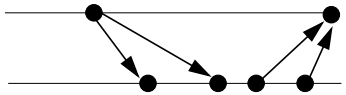
- "Gegenbeispiele":



nicht möglich wegen (2)



nicht möglich wegen (5)



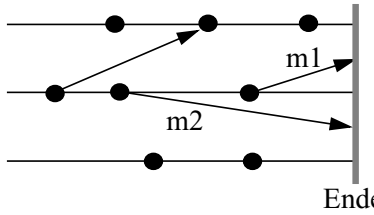
nicht möglich wegen (2)

Bemerkungen zur Definition

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

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

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



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

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

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

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

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

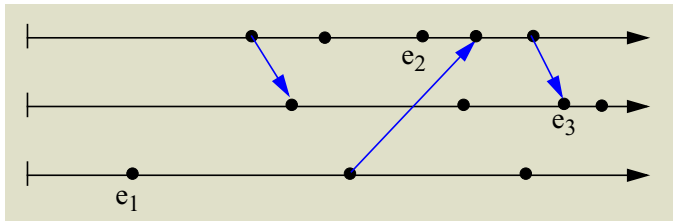
Zeitdiagramme

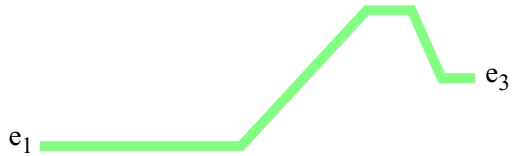
- Theorem [*Kausalkette*]:

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

  Nachrichtenpfeile + Teilstücke auf Prozessachsen von links nach rechts

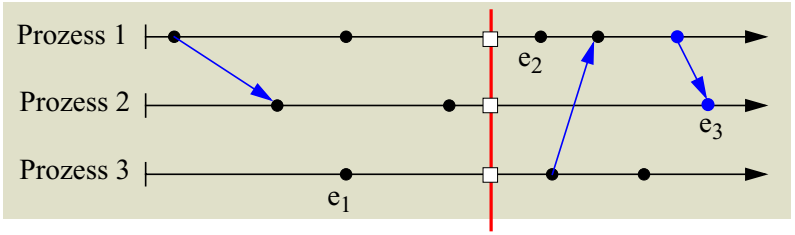
- Bew.: induktiv, Transitivität, "kann beeinflussen",...





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

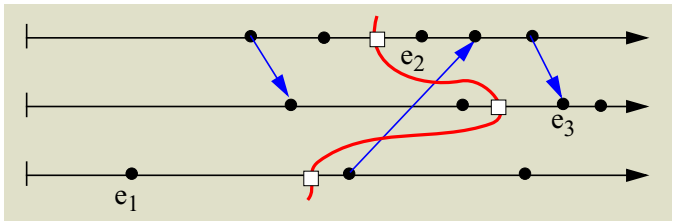
Gummibandtransformation



vertikaler Zeitschnitt

Diagramme sind daher äquivalent

Ein anderes Bild der *gleichen* Berechnung:



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

Nachrichtenpfeile sollen aber nie rückwärts laufen

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

Globale Zustände und Endzustände

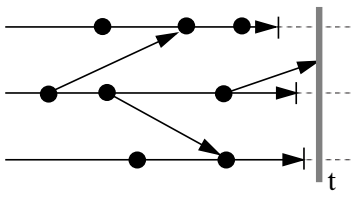
- Was ist ein *globaler Zustand* einer vert. Berechnung?
  - Charakteristikum verteilter Systeme: dieser ist auf die Prozesse verteilt und nicht unmittelbar ("gleichzeitig") zugreifbar!

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

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

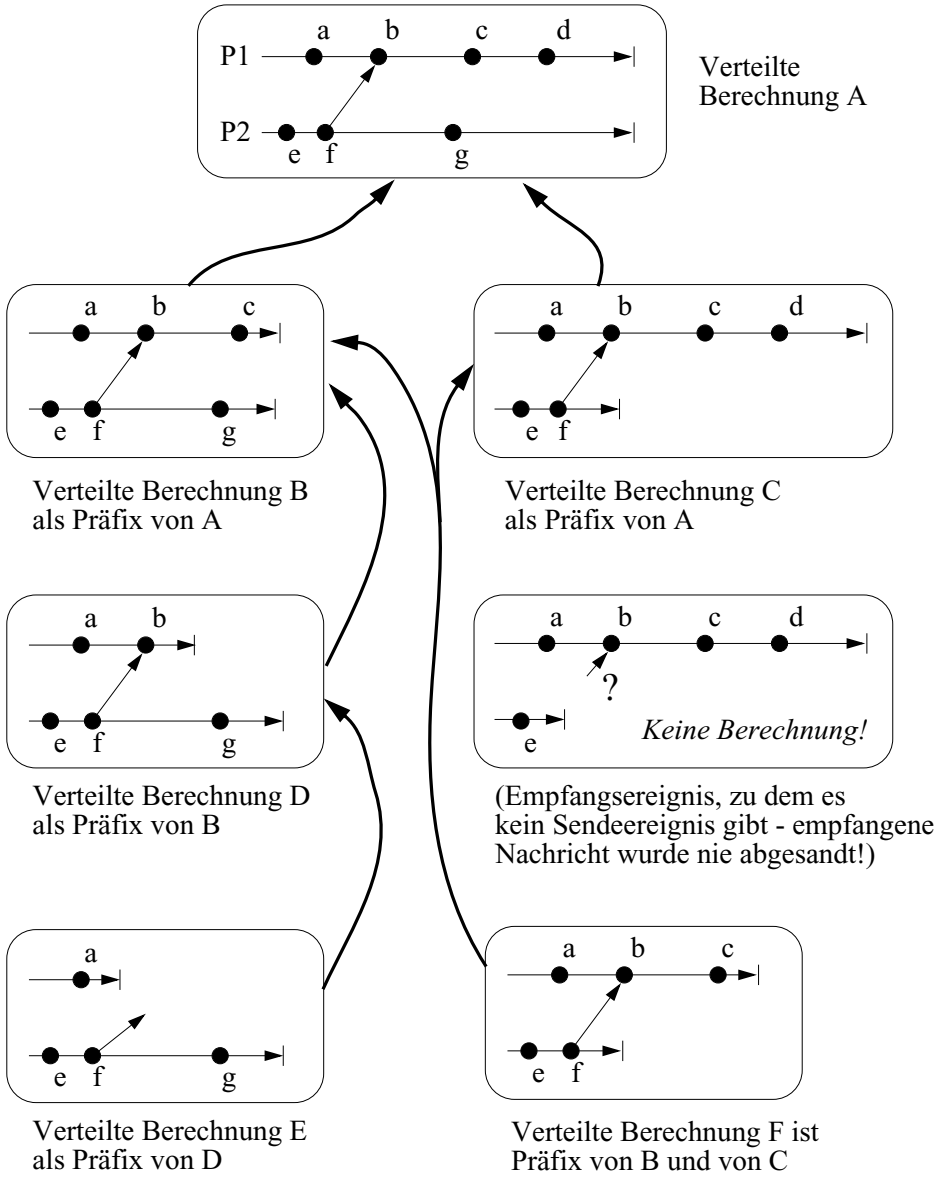
- *Globaler Zustandsraum*:
  - Kreuzprodukt lokaler Zustandsräume
  - Menge von Nachrichten ("in transit")

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



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

Präfixe von Berechnungen



Präfix-Berechnungen

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

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

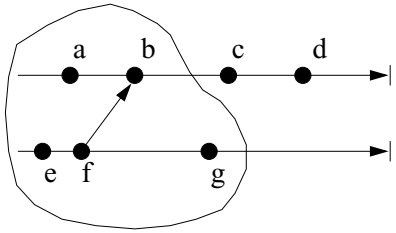
$$\forall\, x \in E', y \in E:\ \ y < x \ \Rightarrow\ y \in E'$$

  $\Rightarrow$ lokale Berechnungen sind wirkliche Anfangsstücke

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

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

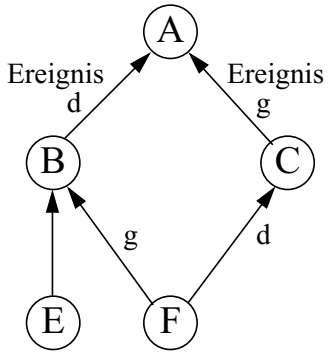
Beispiel:



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

(2): {b, g}

Präfix-Relation



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

- Frage: Entstand "im Verlaufe der Berechnung" A aus B oder aus C?
  - d.h.: durchlief die Berechnung von A vorher den Endzustand von B oder den von C als Zwischenzustand?
  - äquivalent: geschah Ereignis d vor g oder g vor d?
  - beachte: *beides* ist unmöglich (wenn d und g nicht "gleichzeitig")

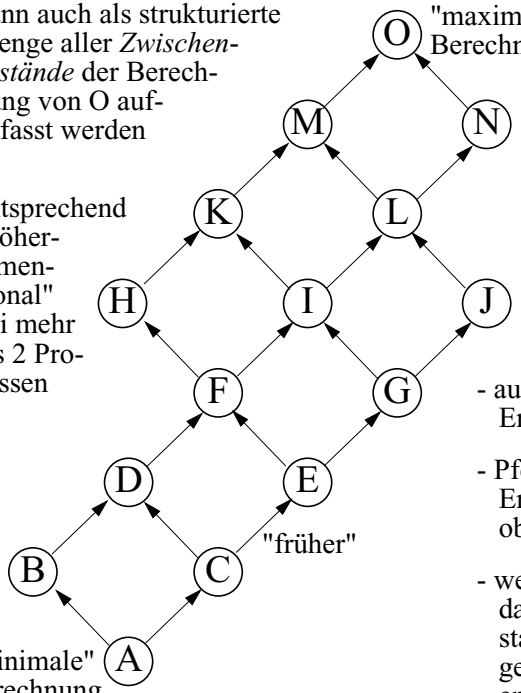
- Konsequenz:
  - direkter Vorgänger i.a. indefinit
  - verteilte Berechnung ist *keine Folge* von Zuständen, sondern - notgedrungen - eine geeignet definierte Halbordnung!

- 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



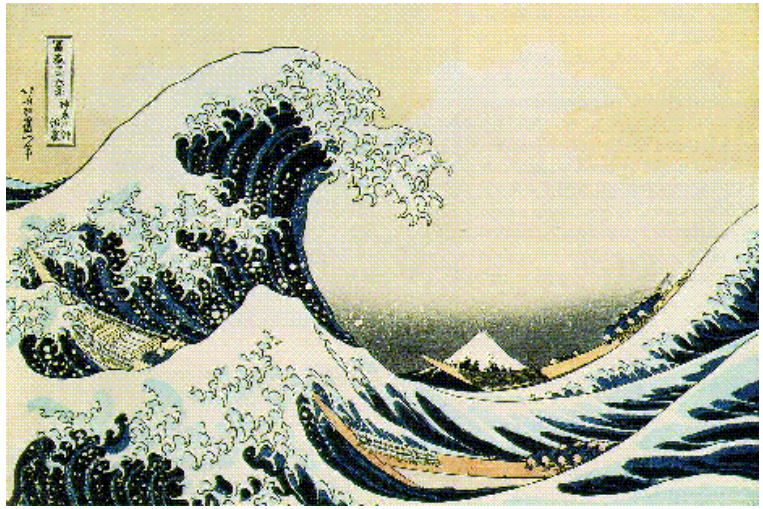
an dieser Stelle würde ein "unmögliches" Diagramm stehen

- 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

$\Rightarrow$ *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 $\leq$ visit
2. visit $\leq$ conclude

$\}$ ($\Rightarrow$ init $\leq$ conclude)

Wellenalgorithmen (2)

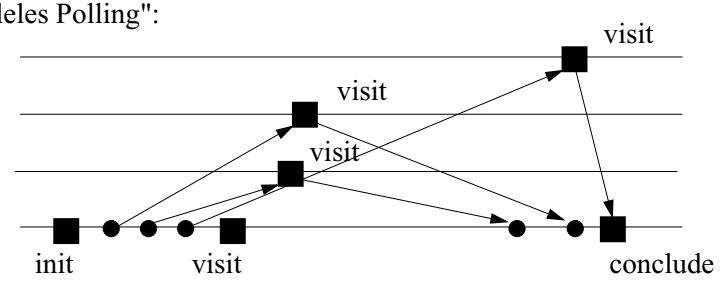
Aus 1: Über die Kausalketten lässt sich *Information* vom Initiator an alle Prozesse *verteilen*

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

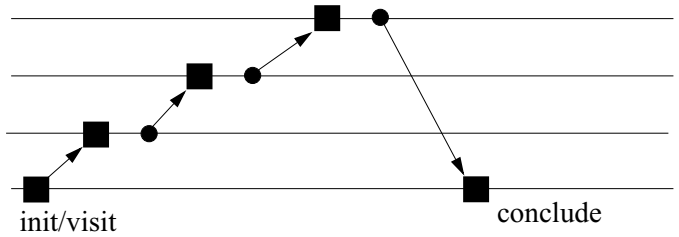
Bem.: a) init und conclude oft im selben Prozess ("Initiator")
b) init oder conclude kann mit visit verschmelzen (daher '$\leq$' statt '$<$')

Beispiele:

"paralleles Polling":



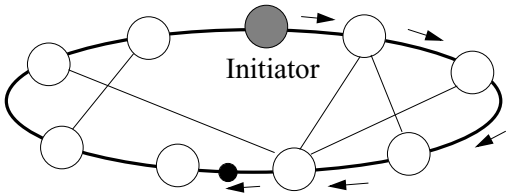
"Ring":



Einige Wellenalgorithmen

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

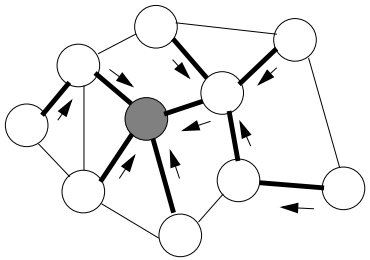
1.) *Ring* / *Hamiltonscher Zyklus* mit umlaufenden Token



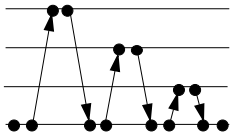
- "logischer" Ring genügt!
- in einem zusammenhängenden ungerichteten Graphen kann ein logischer Ring immer gefunden werden, indem man einen Spannbaum "umfährt"

2.) *Spannbaum*

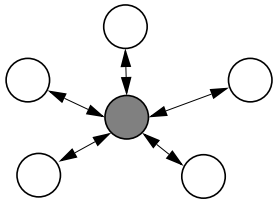
- an den Blättern reflektierte Welle (vereinfachter Echo-Algorithmus: flooding mit rek. acknowledgement)
- bei nicht-entartetem Spannbaum sind viele Nachrichten parallel unterwegs



3.) *(logischer) Stern*



sequentielles Polling



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

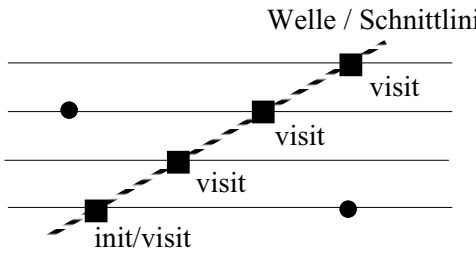
---

4.) *Echo-Algorithmus ist ein Wellenalgorithmus*

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

Wellen und Schnittlinien

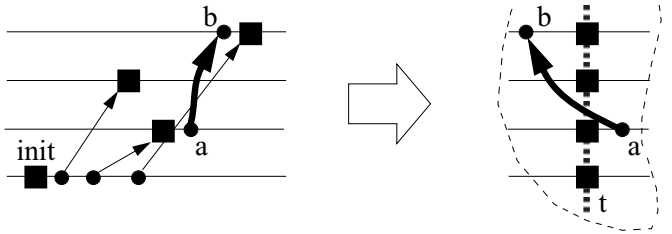
- Verbinden der visit-Ereignisse zu einer *Schnittlinie*:



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)

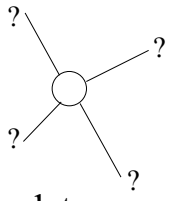
Eigenschaften von Wellenalgorithmen

wodurch ist eigentlich der suggestive Name gerechtfertigt?

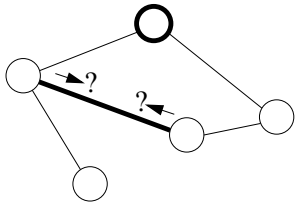
Satz: Es werden *mindestens n-1 Nachrichten* benötigt

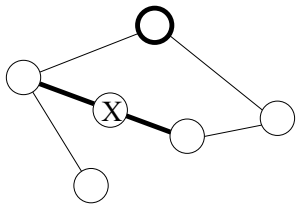
- Begründung: Da jedes visit-Ereignis vom init-Ereignis kausal abhängig ist, muss jeder andere Prozess mindestens eine Nachricht empfangen
- mindestens n Nachrichten, falls conclude und init auf gleichem Prozess stattfinden (dann muss auch der Initiator eine Nachricht empfangen)

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!

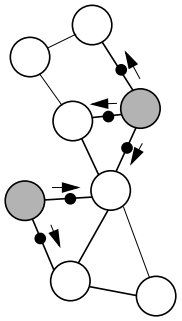
Virtuell gleichzeitiges Markieren

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

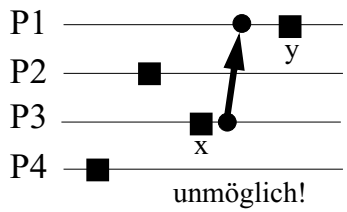
```
I:  {not marked}
    marked := true;   //  = init = visit
    send <marker> to all neighbors;
```

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

mehrere Initiatoren sind zulässig!

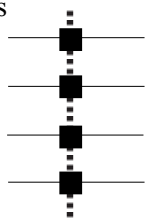


- Über *jede* Kante läuft in beide Richtungen ein Marker
- Eine andere Nachricht ("Basisnachricht"), die von einem markierten Knoten gesendet wird, kommt erst dann an, wenn der Empfänger auch bereits markiert ist



unmöglich!

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

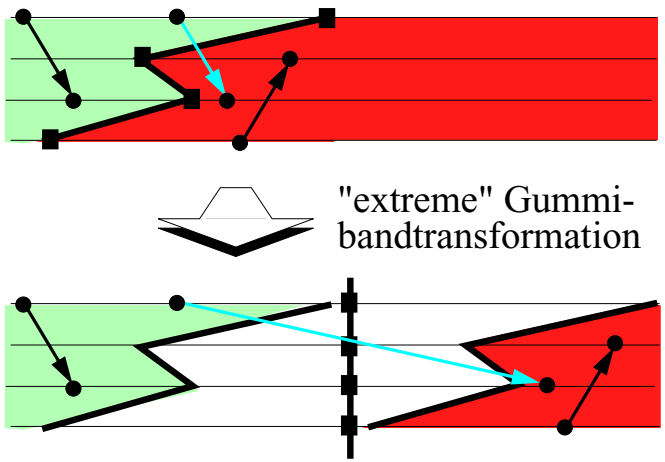


Eine "rückwärts" über den Schnitt laufende Basisnachricht kann es nicht geben: Wenn P3 und P1 benachbart sind, dann wurde bei x ein Marker an P1 gesendet, der nicht (wie von der gezeichneten Nachricht) überholt werden kann

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



"extreme" Gummibandtransformation

- 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 CENTRALE ET AU LYCÉE CONDORCET,
RÉPÉTITEUR A L'ÉCOLE POLYTECHNIQUE,
ET
M. E. ROUCHÉ,
EXAMINATEUR DE SORTIE A L'ÉCOLE POLYTECHNIQUE,
PROFESSEUR AU CONSERVATOIRE DES ARTS ET MÉTIERS
Publication fondée en 1842 par MM. Gerono et Terquem,
et continuée par MM. Gerono, Prouhet, Bourget et Brisse.
TROISIÈME SÉRIE.
TOME QUATORZIÈME.
PARIS,
GAUTHIER-VILLARS ET FILS, IMPRIMEURS-LIBRAIRES
DU BUREAU DES LONGITUDES, DE L'ÉCOLE POLYTECHNIQUE,
Quai des Grands-Augustins, 55.
1895

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.

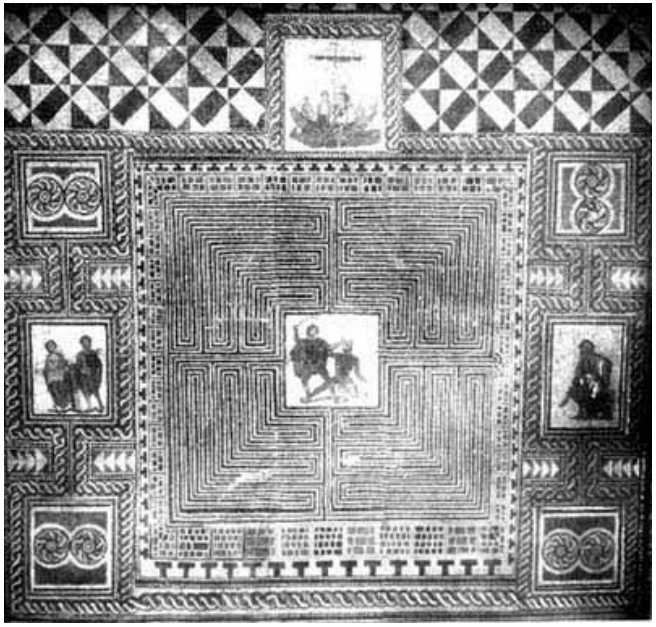
. . .

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



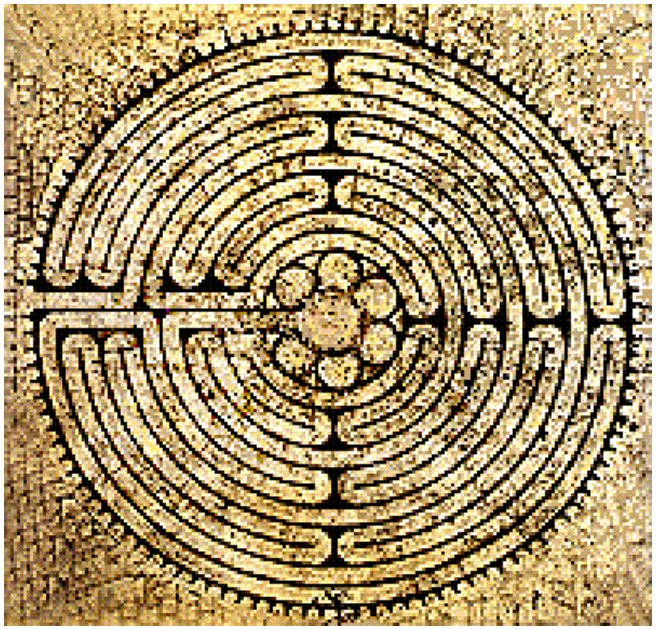
Römisches Mosaik (aus Loig bei Salzburg) illustriert
die Geschichte von Theseus und Minotaurus im Labyrinth

Arcera, Spanien

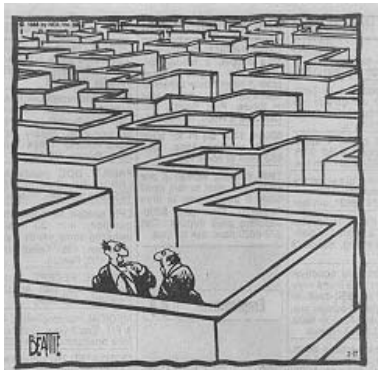
O'odham-Volk
(Papago-Indianer)
aus Süd-Arizona

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

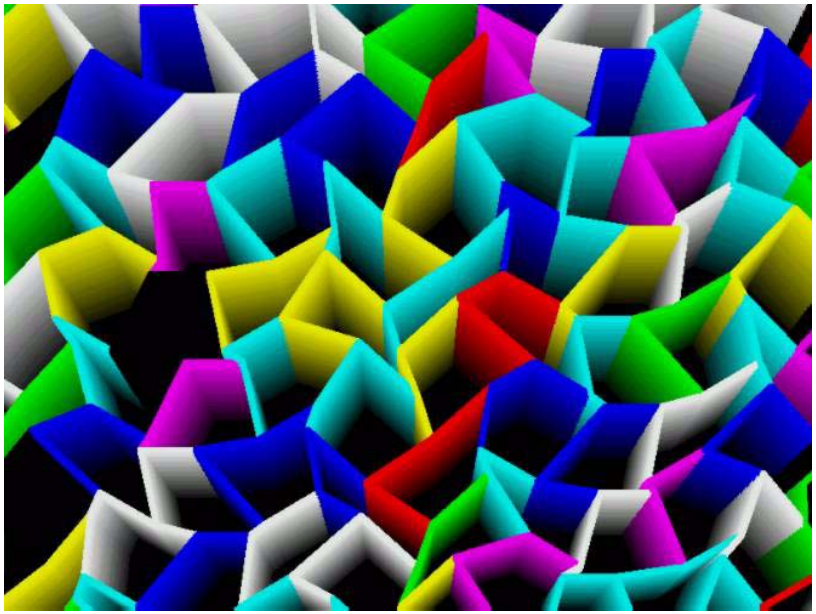
Labyrinthe (2)



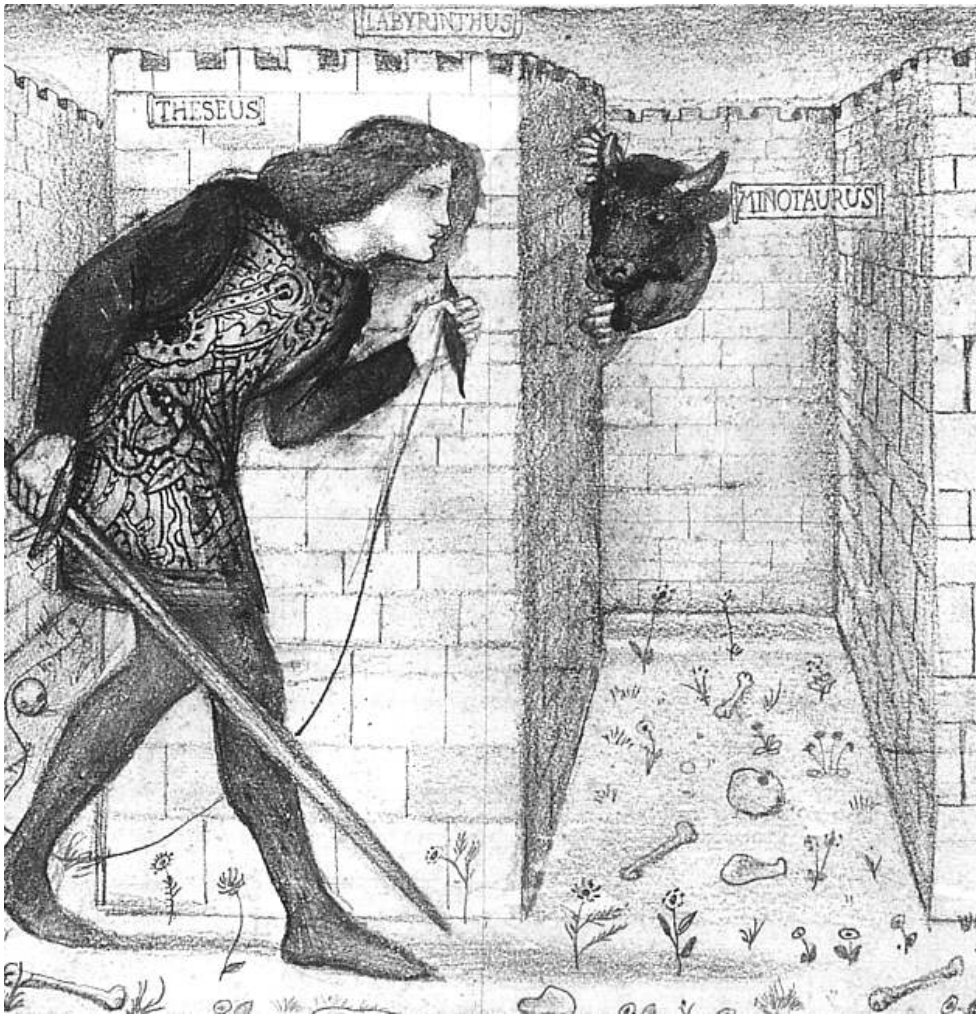
Labyrinthe (3)



"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)

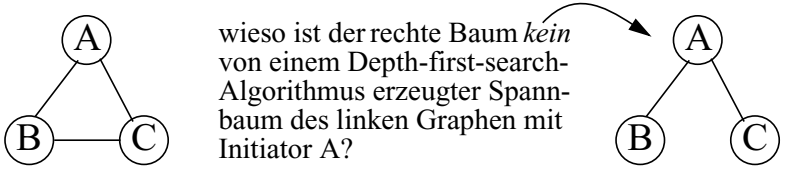
Depth-first-Algorithmen und Spannbäume

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

$\Rightarrow$ 2e Nachrichten, 2e Zeitkomplexität

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



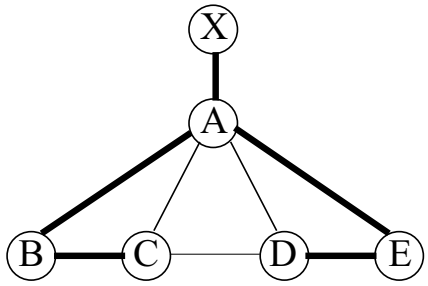
- Eine Charakterisierung solcher Spannbäume:

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 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

- 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!