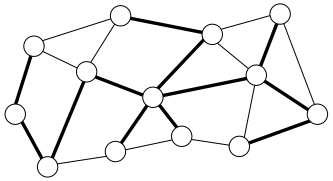
Verteilte Spannbaumkonstruktion?

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



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

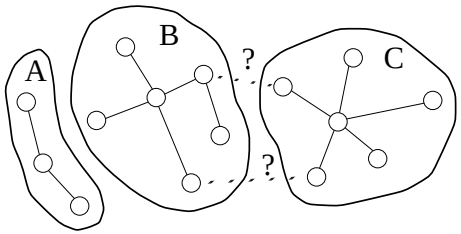
---

Problem: - Wer initiiert die Spannbaumkonstruktion?

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

- *Lösung 1*: Election, Gewinner startet Echo-Algorithmus
(bzw. Variante Echo-Election: Die Echos bilden bereits einen Spannbaum)

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



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

Election und Spannbaumkonstruktion

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

Präziser: Sei C_e die Nachrichtenkomplexität des Election-Problems, und C_t diejenige des Spannbaum-Problems:

(a) Es gilt für C_t: $C_t \leq C_e + 2e$ (wg. obiger "Lösung 1")

(b) Es gilt für C_e: $C_e \leq C_t + O(n)$ (wg. Kplx. Baumelection)

Interpretation:

(a) Hat man mittels Election einen "leader" bestimmt, lässt sich ein eindeutiger Spannbaum einfach ermitteln

(b) Hat man einen Spannbaum, dann lässt sich ein "leader" einfach (d.h. effizient) bestimmen

Hierbei wird 2e bzw. O(n) als "klein" gegenüber C_e und C_t angesehen

---

Aus der unteren Schranke $\Omega(e + n \log n)$ für die Nachrichtenkomplexität des Election-Problems folgt aus (b):

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

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

--> Der genannte GHS-Algorithmus ist grössenordnungsmässig optimal!

Anonyme Netze

- Keine Knotenidentitäten
  - bzw.: alle (oder mehrere) *identische* Identitäten
  - bzw.: ggf. vorhandene Knotenidentitäten werden nicht benutzt

- Frage: Was geht dann noch? (Insbes. Symmetriebrechung!?)

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

...und *alles* geht wie gehabt!

  - 1. Idee: Leader gibt sich einen (neuen) Namen; restliche Knoten führen unter sich eine neue Election durch
  - 2. Idee: Leader fragen, wie man heissen soll; dieser denkt sich Namen aus
  - Konkretisierung bzw. Variante der 2. Idee:
    - es sei also die Existenz eines "Leaders" angenommen
    - dieser kann mit dem Echo-Algorithmus einen Spannbaum bilden (das klappt auch bei anonymen Knoten!)
    - die Echos melden ("rekursiv") die Grösse (= Anzahl der Knoten) jedes gebildeten Unterbaumes zurück
    - jeder Knoten merkt sich für jede "ausgehende" Kante die Grösse des daran hängenden Unterbaumes
    - nach Konstruktion des Baumes wird (ausgehend vom Initiator als Wurzel) der Baum durchlaufen - dabei wird jedem Unterbaum ein disjunktes Intervall natürlicher Zahlen passender Grösse zugeordnet

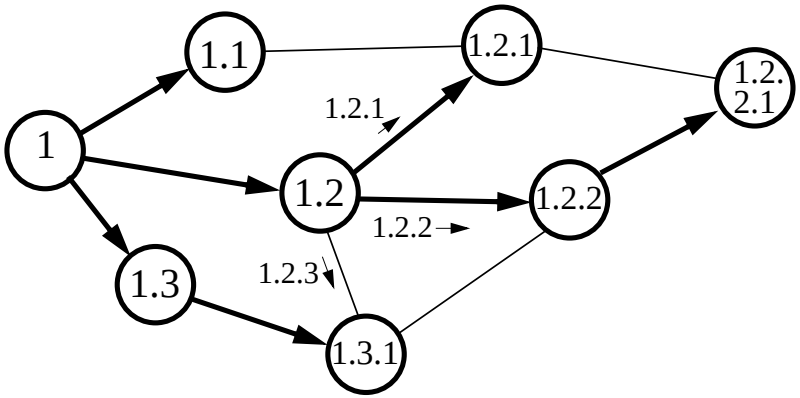
---

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

De-Anonymisierung

Es geht auch direkter ohne explizite Baumkonstruktion,
indem der Leader das Netz mit verschieden benannten Nachrichten
flutet und jeder Knoten die Identität der ersten Nachricht übernimmt,
die ihn erreicht - dazu numeriert ein Knoten seine Kanäle bzw. Nachrichten
durch und konkateniert seine eigene Identität zu dieser Nummer

==> Alle Nachrichten und damit alle Knoten heissen verschieden!



- Einziger (!) Initiator gibt sich selbst einen Namen “1”

- Jeder Prozess numeriert seine Ausgangskanäle 1,...,k

- Jede von Prozess X über Kanal j versendete Nachricht bekommt zusätzlich den Namen “X.j” dazugepackt

- Ein (noch) anonymer Prozess nimmt den Namen der ersten Nachricht als seinen Namen

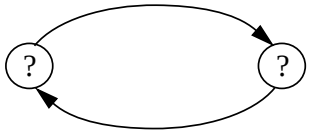
Fragen:
- wann ist das Verfahren beendet?
- wie erfährt der Initiator dies?

Election in anonymen Netzen?

- In anonymen Netzen geht manches (z.B. Election?) nicht mehr mit deterministischen Algorithmen
  --> randomisierte Verfahren helfen gelegentlich!

- Manches geht noch, wenn wenigsten die Knotenzahl bekannt ist (aber auch das hilft nicht immer!)

Satz: *Es gibt keinen stets terminierenden Election-Algorithmus für anonymen Ringe* ← selbst wenn die Ringgrösse den Knoten bekannt ist



- hier für Ringgrösse 2
- jeder Knoten befindet sich (bzgl. des Election-Algorithmus) in einem bestimmten Zustand z

Bew.: - Betrachte *Konfigurationen* (hier Quadrupel aus den Zuständen der beiden Knoten und Kanäle)

- Kanalzustand = Nachrichten, die unterwegs sind

- Anfangkonfiguration des Algorithmus ist *symmetrisch*: (z,z,∅,∅), wegen Anonymität

- Alle lokalen Algorithmen sind definitionsgemäss identisch --> Knoten können sich jeweils gleich (quasi "zum gleichen Zeitpunkt") verhalten

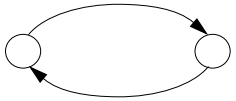
- Konfigurationsfolge kann daher aus lauter symmetrischen Konfigurationen bestehen: (z,z,∅,∅) --> (z',z',k,k) --> ... --> etc.

- Falls die Folge endlich ist, könnte der Endzustand symmetrisch sein --> keine Symmetriebrechung möglich!

Ein probabilistischer Election-Algorithmus

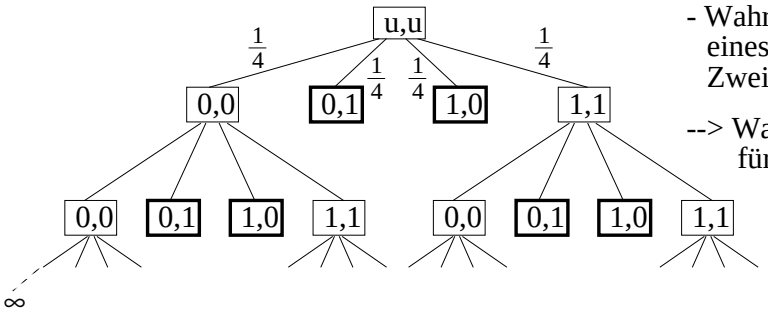
```
state := undecided;
while state = undecided do
{ mine := random(0,1);      ← zufälliges Ergebnis 0 oder 1
  send <mine> to neighbor;
  receive <his>;
  if (mine,his) = (1,0) then state := win;
  if (mine,his) = (0,1) then state := lose;
}
```

- hier: *anonyme* Ringe der Grösse 2



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

- Geht es nicht auch mit einem *deterministischen* Algorithmus?
- Beachte: *"korrekt" ≠ "korrekt mit Wahrscheinlichkeit 1"*!
  - "korrekt" heisst: Algorithmus *hält stets* und liefert richtiges Ergebnis
  - obiger Algorithmus ist aber (in diesem Sinne) *nicht korrekt*, da es nicht terminierende Konfigurationsfolgen wie etwa
    - (0,0) --> (0,0) --> (0,0) --> (0,0) --> (0,0) --> (0,0) .. . oder
    - (0,0) --> (1,1) --> (0,0) --> (1,1) --> (0,0) --> (1,1)... etc. gibt!
  - alle diese Folgen haben aber die Dichte 0 (wenn "random" gut genug ist...)
    ==> Algorithmus terminiert mit Wahrscheinlichkeit 1
    ==> Algorithmus ist "korrekt mit Wahrscheinlichkeit 1"!



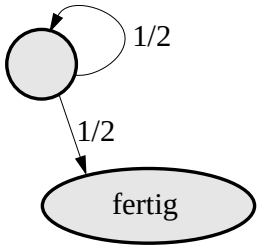
- Wahrscheinlichkeit eines unendlichen Zweiges = 0

--> Wahrscheinlichkeit für leader = 1

Mittlere Nachrichtenkomplexität?

- Beachte: Keine Schranke für *worst-case* Komplexität!

- Die mittlere ("erwartete") Nachrichtenkomplexität lässt sich mittels *Markow-Ketten* leicht ermitteln

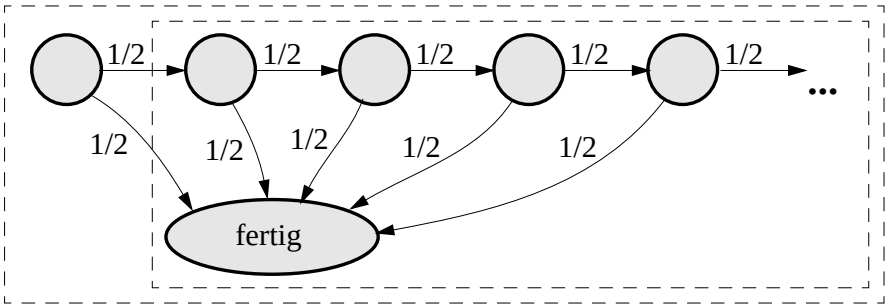


- wie hoch ist die erwartete Weglänge W, wenn mit den angegebenen Wahrscheinlichkeiten (hier jeweils 1/2) zum Folgezustand verzweigt wird?

- ist dieser Ansatz (gewichtete Summe) korrekt?

$$1\,\frac{1}{2} + 2\,\frac{1}{4} + 3\,\frac{1}{8} + 4\,\frac{1}{16} + \ldots + \quad = ?$$

- Durch "Aufbröseln" der Schleife ergibt sich diese Darstellung:



- hier ist man mit Wahrscheinlichkeit 1/2 nach einem Schritt fertig, oder es liegt (nach einem Schritt) wieder die gleiche Situation vor!

- daraus ergibt sich der *Rekursionsansatz* W = 1/2 + 1/2 (1+W)

- dies liefert W=2 als Lösung

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

Probabilistische Algorithmen

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

---

1. Sogenannte *Las Vegas-Algorithmen:*

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

---

2. Sogenannte *Monte Carlo-Algorithmen:*

- Abschwächung der partiellen Korrektheit
- "terminiert stets, ist aber nur mit Wahrscheinlichkeit $p < 1$ partiell korrekt"
- also: $\exists$ *Restwahrscheinlichkeit* $\varepsilon = 1-p > 0$, dass das Ergebnis falsch ist!
- nur verwenden, wenn:
  - ε sehr *klein* ist (oft: als Parameter des Algorithmus, etwa abhängig von der Laufzeit und damit "beliebig klein" *wählbar*)
  - dadurch deutliche Vorteile erzielbar (Problem effizienter oder überhaupt erst lösbar)
- beachte den "Sonderfall" $p=1$ (also $\varepsilon=0$): 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* --> ∃ anderen Knoten gleicher Identität (merken mittels Flag!)
      - *ja* --> 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* ≤ e (n/n-1)

  2.718281828...

  - vgl. mit vorherigem Algorithmus für Ringgrösse 2

- Berechnungsdauer ist im Prinzip aber unbegrenzt!

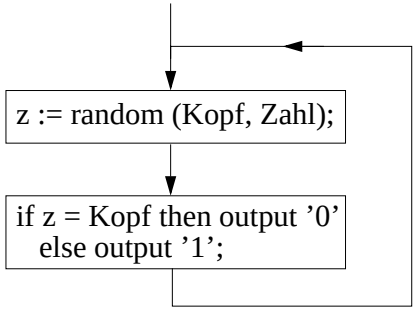
- Algorithmus ist partiell korrekt: *Wenn* er hält, dann mit genau einem leader!

- Verallgemeinerung auf *allgemeine Netze* mit Echo-Algorithmus (statt Ring) ist möglich:
  - wenn die durch Echos gemeldete Baumgrösse ≠ n ist, neue Runde starten etc.

Zufällige reellwertige Zahlen?

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

- Realisierung solcher Zufallszahlengeneratoren?



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

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

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

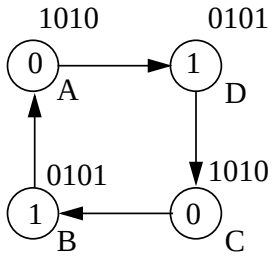
Genügt ein einziges Zufallsbit?

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

für Ringe bekannter Grösse

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

---



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

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

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

Schätzung der Ringgrösse?

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

- Bei *unbekannter Ringgrösse* lässt sich diese aufgrund von zyklischen Wiederholungen im Bitstring mit wenigen Läufen mit hoher Wahrscheinlichkeit korrekt schätzen...
  - Wieviele Läufe sind für eine gewisse Sicherheit notwendig?
  - Konsequenzen, wenn man sich unerkanntermassen irrt?

Wir wollen das an dieser Stelle nicht weitertreiben...

Aus “philosophischer Sicht” ist allerdings interessant:

- Was ist an minimaler (struktureller) Information notwendig, um Symmetrie zu brechen? (Beispiele: Ringgrösse ist eine unbekannte Primzahl; obere Schranke für die Ringgrösse...)
- Unter welchen minimalen Voraussetzungen ist eine deterministische oder probabilistische Lösung möglich?
- Wie “sicher” und effizient können probabilistische Algorithmen für dieses Anwendungsproblem sein?

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

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

Kenntnis der Knotenzahl?

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

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

- *Bestimmung der Grösse anonymer Ringe mit einem Monte Carlo-Algorithmus* (hier nur Andeutung des Prinzips):
  - Jeder Prozess p hält einen konservativen Schätzwert $\bar{n}_p \leq n$ (initial: $\bar{n}_p = 2$).
  - Prozess p generiert ein Token mit einer Zufallsmarke aus {1,...,r} und sendet es $\bar{n}_p$ Schritte weit. Das Token enthält dazu den Wert $\bar{n}_{tok}$ (= $\bar{n}_p$ des Senders p).
  - Prozess p erhöht sein $\bar{n}_p$ um 1 und sendet ein neues Token, wenn:
    - er ein Token empfängt mit $\bar{n}_{tok} > \bar{n}_p$, oder
    - er ein Token empfängt mit $\bar{n}_{tok} = \bar{n}_p$, aber dessen Marke nicht der eigenen Marke entspricht.

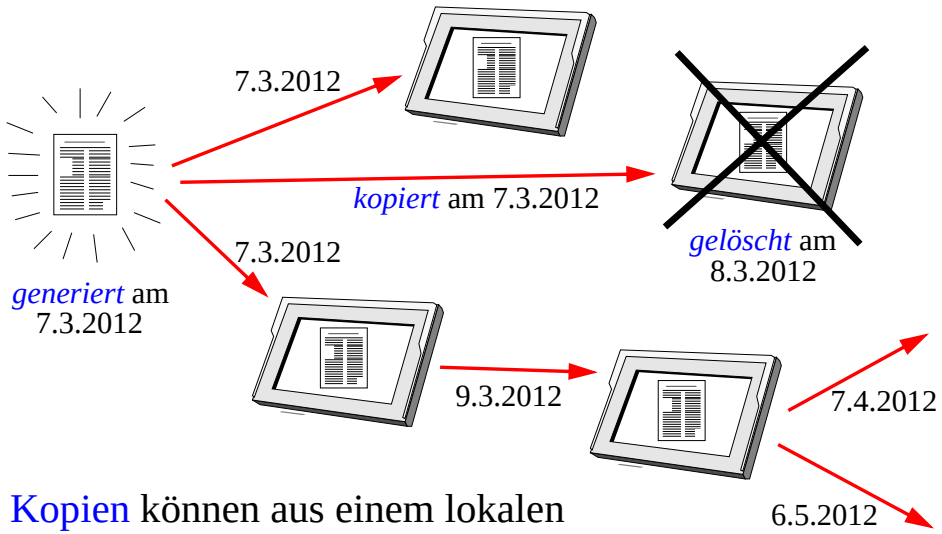
- Wir notieren ohne Beweis: Es werden höchstens $O(n^3)$ Nachrichten generiert; es sind am Ende alle $\bar{n}_p$ identisch mit $\bar{n}_p \leq n$; und es ist $\bar{n}_p = n$ mit einer Wahrscheinlichkeit $\geq 1-(n-1)\,(1/r)^{n/2}$. (Die Wahl von r ist also "kritisch".)
- Varianten, bei denen der Schätzwert $\bar{n}_p$ gelegentlich schneller wächst, sind möglich.

Kausaltreues Beobachten

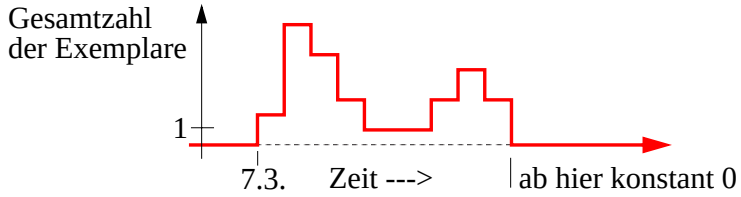
***Beob*achten *ver*teilter Systeme**

Aussterben einer Menge von Kopien

- Ein Beispiel: on-line Kopien einer Zeitung:



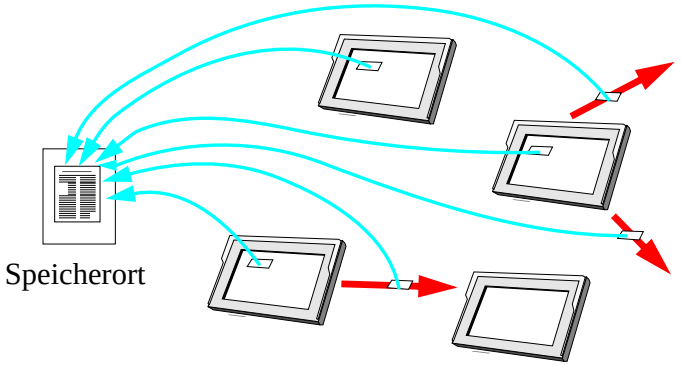
- Kopien können aus einem lokalen Exemplar erzeugt und verteilt werden
- Exemplare werden gelegentlich gelöscht



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

Beobachten des Referenzzählers?

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



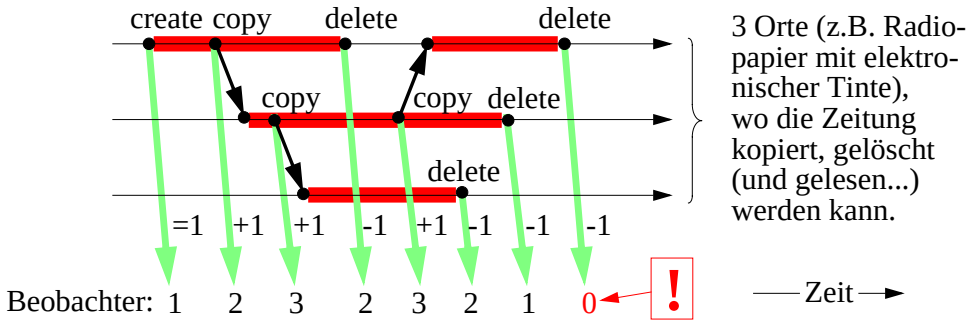
- Copy --> Übermitteln der Referenz (=Zugriffspfad / Adresse)
- Delete --> Löschen der Referenz

- "Beobachter" befindet sich z.B. am Speicherort
- *Referenzzähler* = 0 --> Objekt physisch löschen
  - Begründung: Objekt kennt ja sowieso keiner mehr... (ist Garbage!)
  - aber natürlich nur bei kausaltreuer Beobachtung korrekt!

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

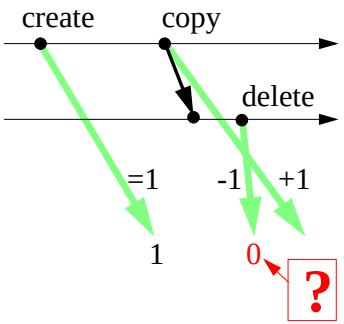
Zählen der Exemplare?

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



3 Orte (z.B. Radiopapier mit elektronischer Tinte), wo die Zeitung kopiert, gelöscht (und gelesen...) werden kann.

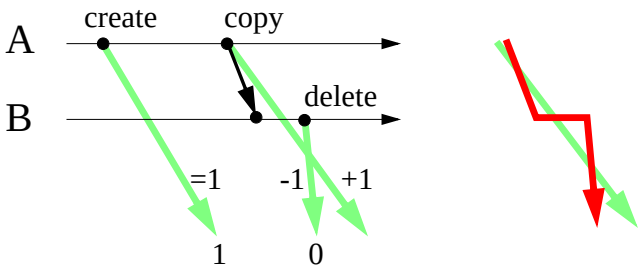
- Aber Achtung: Beobachtung ist u.U. nicht kausaltreu!



- Beachte: delete-Ereignis ist eine *kausale Konsequenz* des copy-Ereignisses ("ohne copy auch kein delete")
- Beobachter sieht jedoch die Konsequenz *vor* ihrer Ursache!
- Beobachter meint *fälschlicherweise*, dass die Zeitung unwiederbringlich verloren sei!

Was ist der Grund für das Problem?

- Kennen wir das nicht schon von der vert. Terminierung?



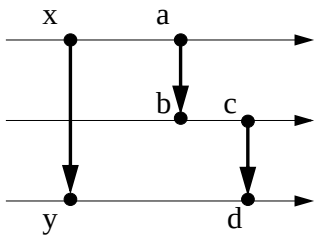
- Eine Einzelnachricht (Meldung von "copy") wurde in indirekter Weise *überholt* (Pfad von copy-Nachricht und delete-Meldung)
- Auf dem Überholpfad können Ereignisse liegen (hier z.B. "delete"), die "Konsequenzen" des überholten Ereignisses ("Ursache", hier "copy") darstellen
- Vermutung: Vermeiden von (direkten oder indirekten) Überholungen löst das Problem, d.h. liefert immer kausaltreue Beobachtungen

- Nicht-kausaltreue Beobachtungen sind die Ursache für viele konzeptuelle Probleme verteilter Systeme! **!**
  - z.B. verteilte Terminierung, Schnappschuss, Deadlockerkennung...
- Wie könnte man also das Überholen von Nachrichten durch Pfade anderer Nachrichten vermeiden? **?**

Vermeiden (indirekter) Überholungen?

1. Idee: Verwendung von *synchroner* Kommunikation

- zumindest bei Meldungen an den Beobachter



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

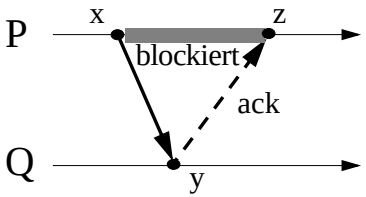
$t(y) = t(x) < t(a) = t(b) < ... < t(c) = t(d) \quad ==> \quad t(y) < t(d)$

Informell: "Blitzschnelle" Nachrichten kann man nicht überholen

---

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

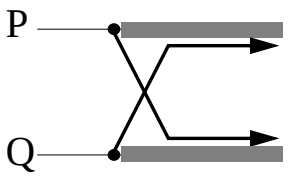
- aber das ist ja nichts anderes als eine Implementierung / Simulation synchroner Kommunikation!



Jede bei P *nach* x (und damit nach z) gestartete Nachrichtenkette kommt bei Q garantiert *nach* y an

(Nachrichtenpfeile verlaufen nie "rückwärts" in der Zeit)

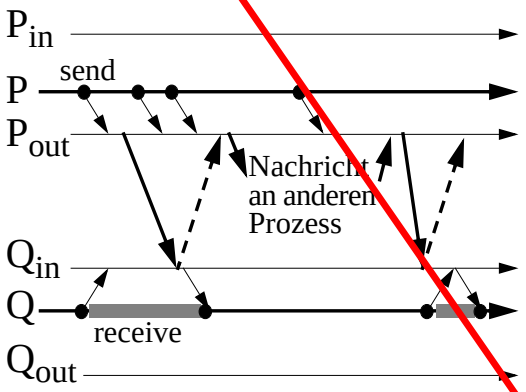
Problem bei einer solchen "Betriebsart" verteilter Systeme sind *Deadlocks*, wenn etwa "gleichzeitig" P an Q und Q an P sendet



Andere, bessere Idee zur Vermeidung indirekter Überholungen?

Eine deadlockfreie Realisierung?

- 3. Idee: Verwendung von Sende- und Empfangspuffern
  - jeder Prozess P hat jeweils einen solchen FIFO-Puffer P_{out} bzw. P_{in}
  - bei "receive" wendet sich der Prozess P an seinen Eingangspuffer P_{in}
    - bekommt älteste Nachricht, sofern vorhanden, zurück
    - blockiert, falls z.Z. keine Nachricht vorhanden
  - bei "send" übergibt der Prozess P die Nachricht seinem Sendepuffer P_{out}



Regel:
Die nächste Nachricht wird erst dann vom Sendepuffer an den entspr. Empfangspuffer geschickt, wenn die von ihm vorher versandte Nachricht bestätigt wurde.

- Beweis, dass es keine (indirekten) Überholungen gibt, nutzt u.a. die FIFO-Eigenschaft der Puffer aus!
- Beachte, dass durch die Puffer (im Gegensatz zu vorherigen Lösungen) der Sender vom Empfänger *entkoppelt* ist!
  - Sender wird nicht durch langsamen Empfänger behindert
  - keine Deadlocks bei gegenseitigem / zyklischen Senden

- andere Lösungen für das Vermeiden indirekter Überholungen (bzw. kausaltreue Beobachtungen)?
- wie gut ist diese Lösung im Vergleich dazu?
- wie kann man die Korrektheit formal beweisen?
- wofür kann man somit realisierte "kausaltreuen" Berechnungen ansonsten sinnvoll verwenden?

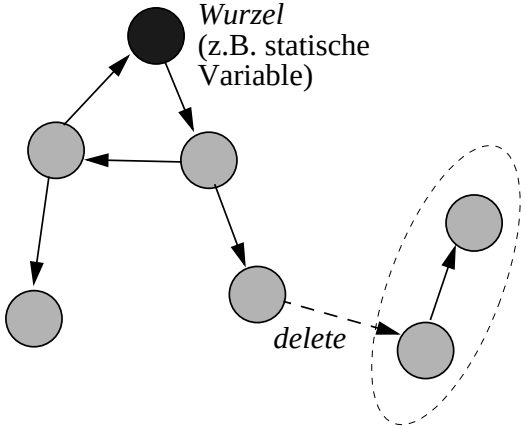
hilft hier ein logisches Zeitkonzept?

darauf kommen wir später zurück!

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

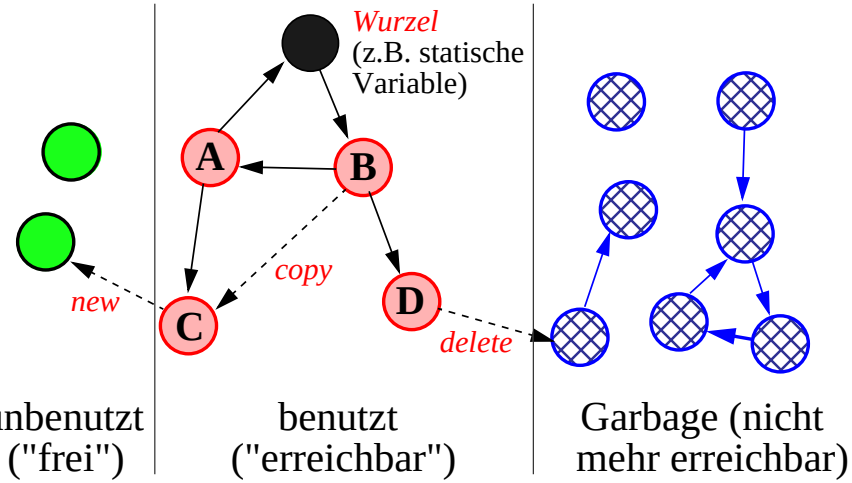
- Typischerweise läuft auch der Garbage-collector (echt oder im Zeitmultiplex) parallel zur Anwendung

==> 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

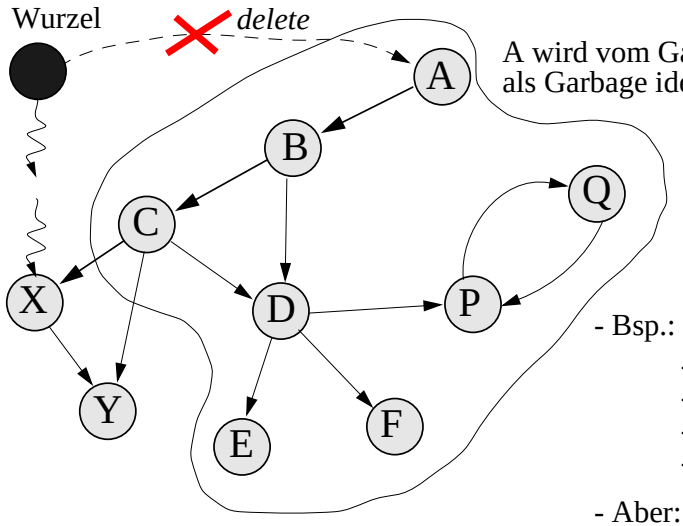
- Objekt *erreichbar* <==> ∃ Pfad von der Wurzel dorthin
- "Garbage sein" ist stabiles Prädikat (vgl. Terminierung!)
- *Mutator* <--> *Collector* spielen mit-/gegeneinander

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

Rekursives Freigeben

- Falls ein Objekt als Garbage erkannt wird:
  - sollten seine ausgehenden Referenzen gelöscht werden,
  - damit kann 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

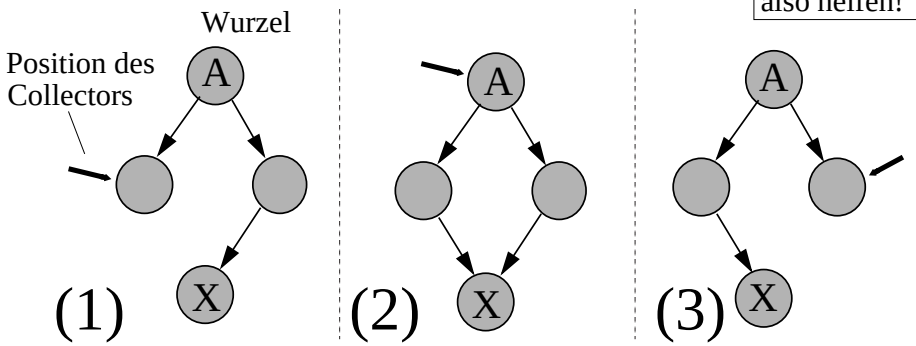
*Concurrent / parallel / on-the-fly-G*arbage-Collection:

- 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!

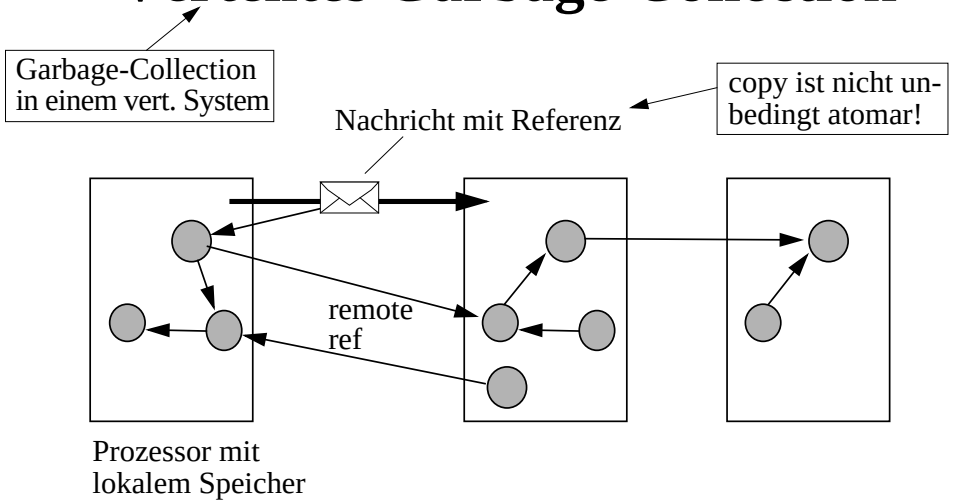
Mutator muss dem Collector also helfen!



- 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

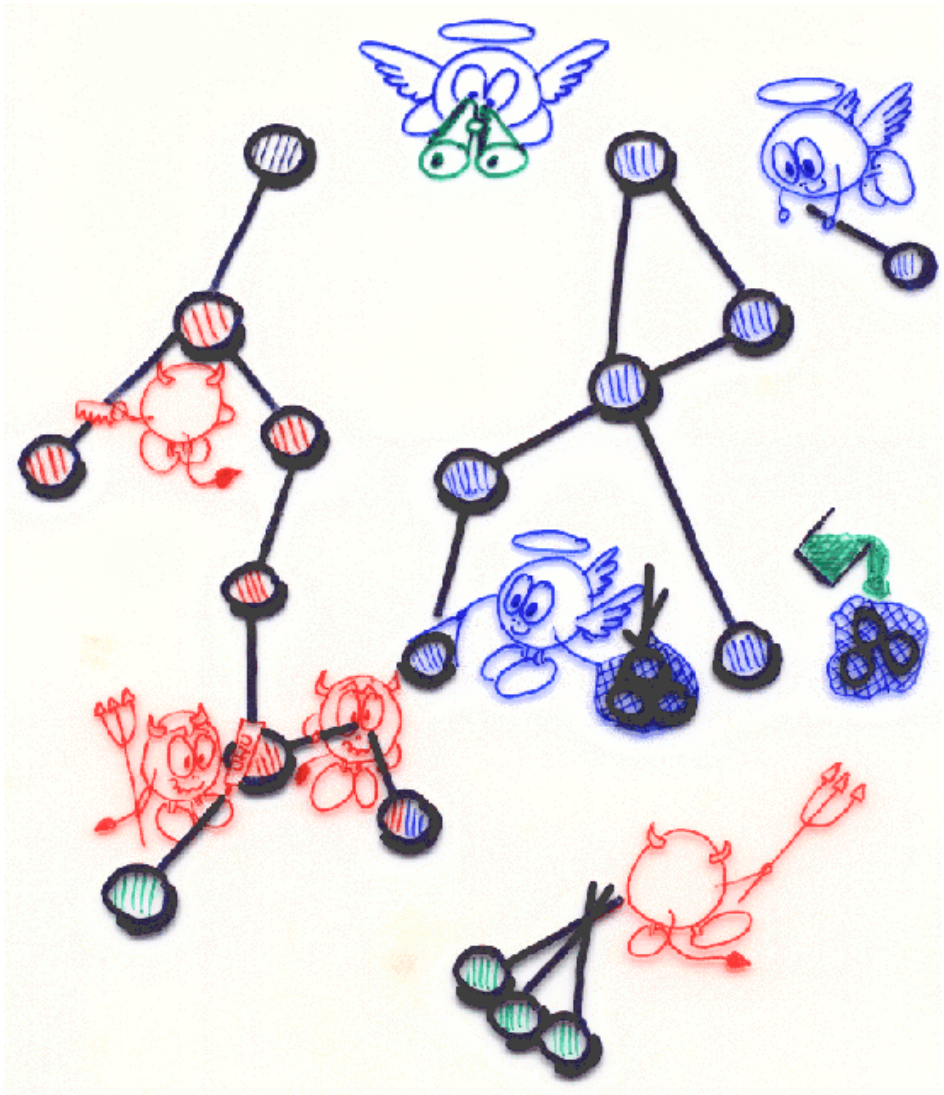


(1) Prozessorüberschreitende Referenzen ("*remote ref*")

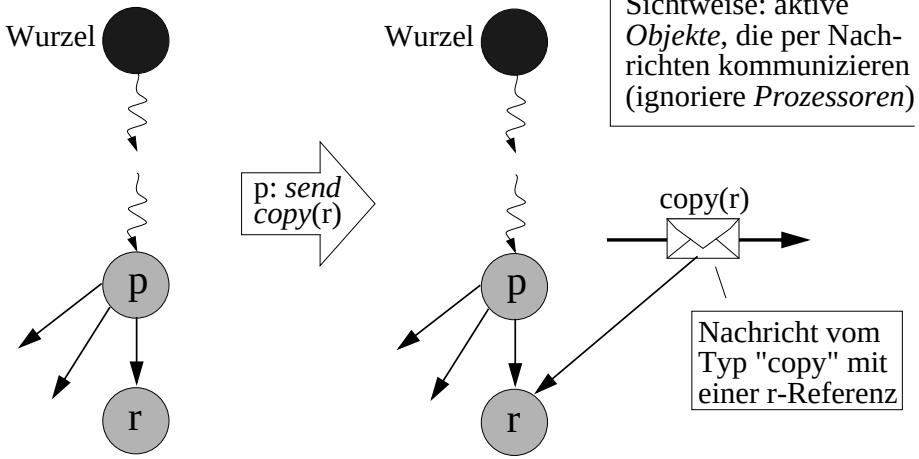
(2) Nachrichten enthalten (Objekte mit) Referenzen auf andere Objekte ("*remote copy*")

- 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)

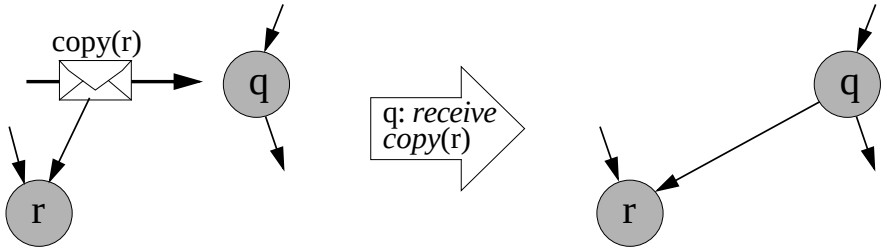


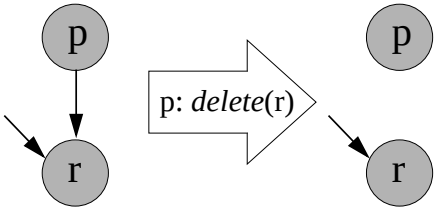
Wirkung der Mutator-Operationen



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):

beachte: p kennt q (hat also "in gewissem Sinne" eine Referenz auf q)

guard

C_p: {p ist erreichbar und hat eine r-Referenz}
send copy(r) **to** q

Parameter Name } der Nachricht (vgl. Prozeduraufruf)

R_p: {Empfang einer copy(r)-Nachricht}
Installiere eine r-Referenz in p

D_p: {p hat eine r-Referenz}
Lösche die r-Referenz

"new" auf copy zurückführen: freie Objekte sind stets erreichbar (z.B. über eine an der Wurzel verankerte Freispeicherliste)

- 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

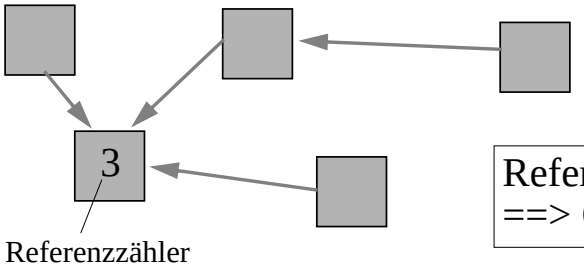
- Interessante Analogie zwischen beiden Problemen:

Verteilte Terminierung	*Garbage-Collection*
Überlagerter Kontrollalgorithmus	
Globale *Terminierung* ist ein stabiles Prädikat	"Objekt ist *Garbage*"
Kontrollalgorithmus soll das globale Prädikat entdecken	
Kontrollalgorithmus nebenläufig zur Anwendung	
Senden einer *Nachricht*	Kopieren einer *Referenz*
Problem: Behind the back *reactivation*	*copy*

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

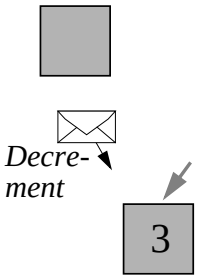
Referenzzähler-Verfahren

- Idee: Jedes Objekt weiss, wie oft es referenziert wird
  - wird dieser Referenzzähler 0 --> Garbage
- Nachteil: Zyklischer Garbage wird 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



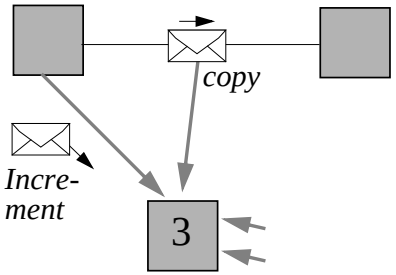
Referenzähler = 0
==> Objekt ist Garbage

Löschen einer Referenz:



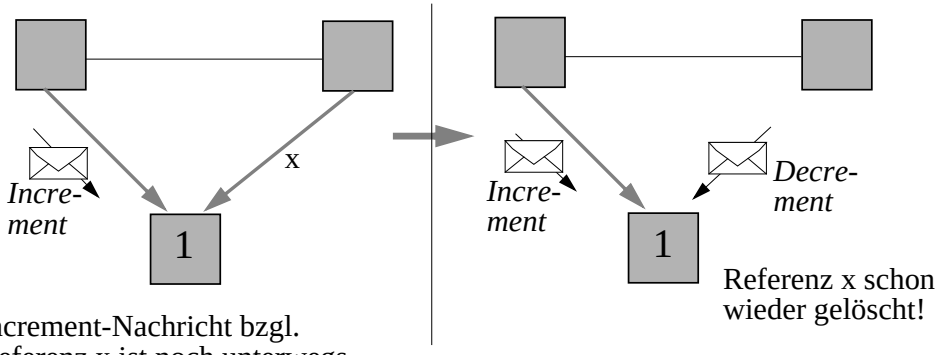
3 ==> 2

Kopieren einer Referenz:



3 ==> 4

Kopieren kann zu *Fehlinterpretationen* führen!



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

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

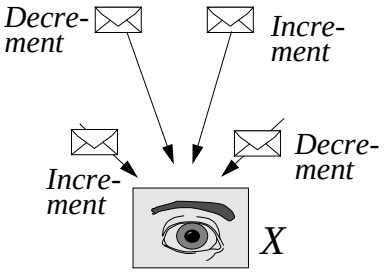
Wie dies garantieren?

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

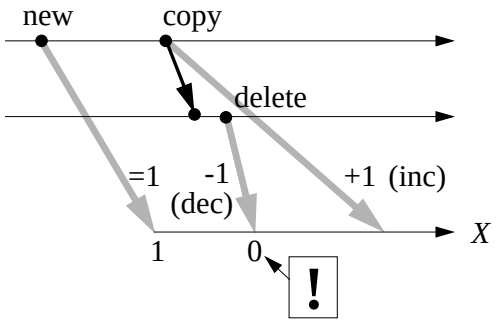
Garbage-Identifikation als konsistentes Beobachtungsproblem



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 / new 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?