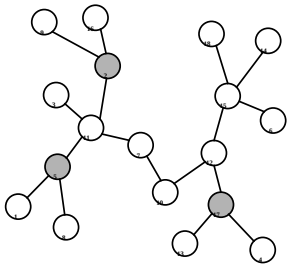
Election auf Bäumen

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



- Explosionsphase:
  - Election-Ankündigung wird zu den Blättern propagiert

flooding!

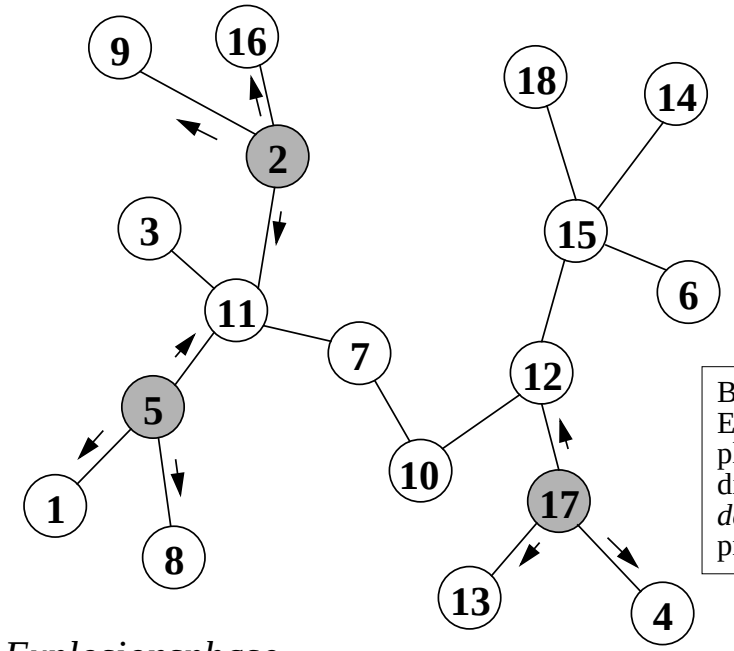
- Kontraktionsphase
  - von aussen zum "Zentrum" das Maximum propagieren

0 für unbeteiligte Knoten

- Informationsphase (notw.?)
  - Zentrum informiert alle Knoten über Gewinner

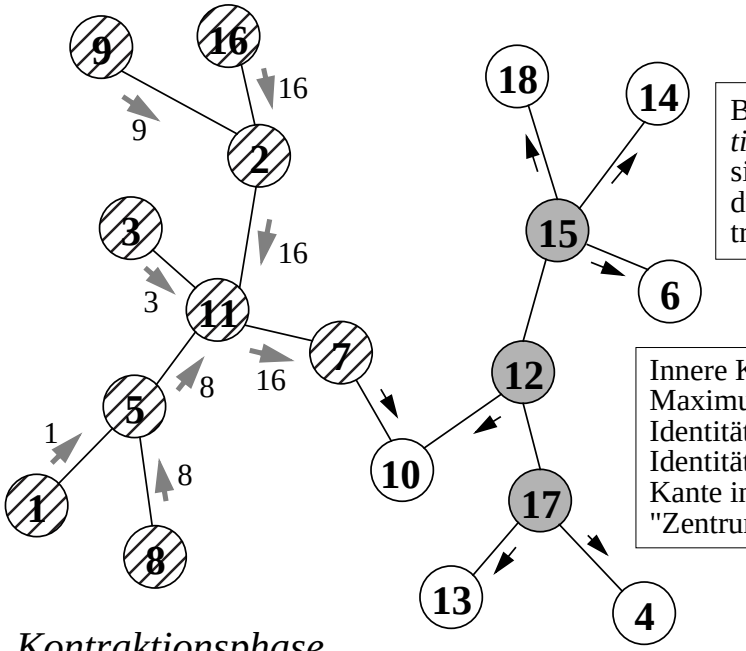
flooding!

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



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

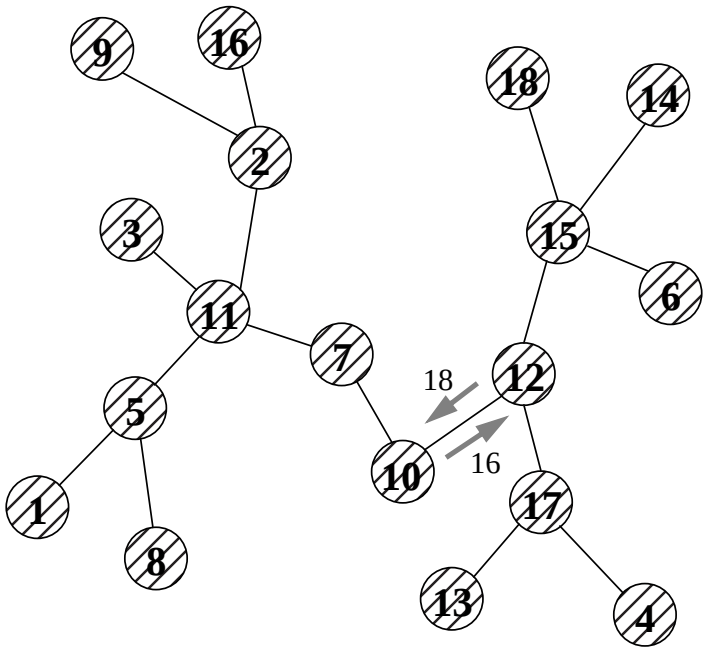
Explosionsphase



Blätter *reflektieren* Explosionsnachricht durch eine Kontraktionsnachricht

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

Kontraktionsphase



- Begegnung zweier Kontraktionsnachrichten auf (genau) einer Kante im Zentrum

- Die beiden Knoten wissen, dass sie nun das Maximum kennen

- Sie können dies nun ggf. per flooding verbreiten (Informationsphase)

- Terminierung der flooding-Phase (falls notwendig) einfach durch erneute Reflexion / Kontraktion ("indirektes acknowledge")

Nachrichtenkomplexität von Baumelection

Folgender Satz / Beweis ist *falsch* - wieso?

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

Beweis induktiv:

1) n=1 --> m(1) = 0 ✓ (offensichtlich korrekt)

2) Schritt von n auf n+1:

- Füge an einen Baum aus n Knoten ein Blatt an; über die neue Kante fliessen genau 2 Nachrichten

- Also: m(n+1) = m(n)+2 = 2n-2+2 = 2(n+1)-2 ✓

- wo genau liegt der Fehler?

- korrekter Wert der Nachrichtenkomplexität --> nächste Folie!

Nachrichtenkomplexität

(1) Explosionsphase: n-2+k

Anzahl der Initiatoren

- es gibt k-1 Begegnungskanten von Explosionsnachrichten

(2) Kontraktionsphase: n

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

(3) Informationsphase: n-2

- keine Nachricht über die Zentrumskante

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

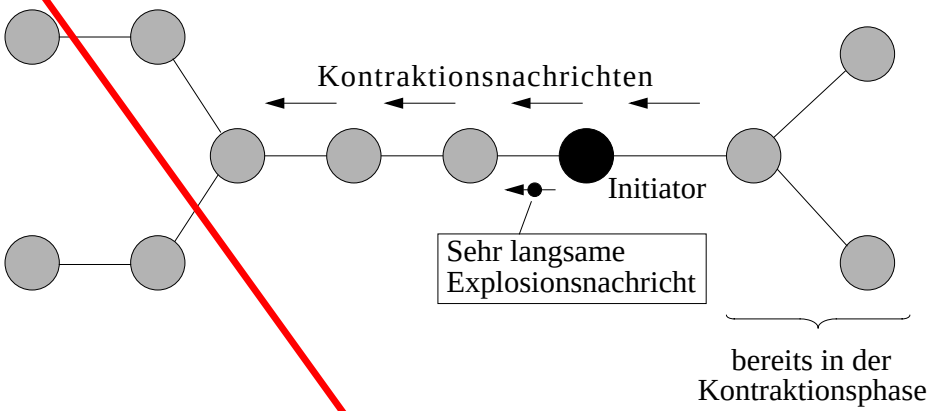
--> 3(n-1) für k=1

--> 4(n-1) für k=n

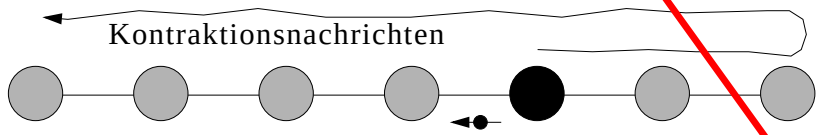
- Wesentlich effizienter als Ringe!
- Wieso? (Ringe sind symmetrischer!)
- Wieso Verfahren nicht "einfach" auf Ringe anwenden?

(eine Kante entfernen)

Schaden Nachrichtenüberholungen?



- Kann eine Kontraktionsnachricht eine Explosionsnachricht überholen?
- Muss man das vermeiden?
- Lassen sich vielleicht sogar z.T. Nachrichten durch Zusammenfassen (Explosions / Kontraktion) sparen?

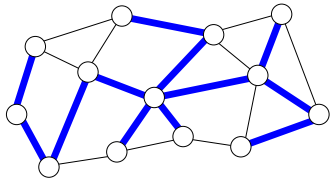


- Wie würde in diesem Fall das Ende erkannt?

Election auf allgemeinen Graphen

zusammenhängend

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



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

"Echo-Election" für allgemeine Graphen

zusammenhängend mit bidirekt. Kanten

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

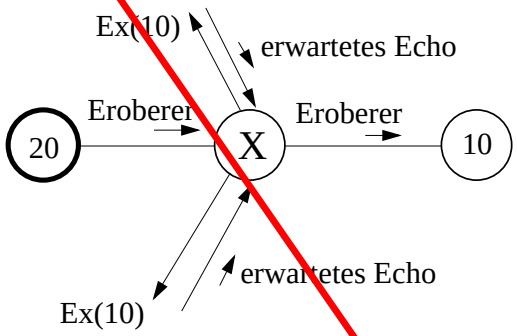
- Also:
  - Jeder Initiator startet "seinen" Echo-Algorithmus
  - Explorer und Echos führen die Identität des Initiators mit
  - Schwächere Nachrichten (Explorer und Echos) werden "verschluckt" (d.h. nicht weitergegeben)
  - Stärkste Welle setzt sich überall durch (und informiert so neben dem Gewinner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

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

- Vermutung bzgl. der worst-case und der average-case Nachrichtenkomplexität?

Varianten, z.B. "Adoption"

- Idee: Stärkerer Knoten ("Eroberer") läuft nicht besiegten Explorern hinterher, sondern "adoptiert" dessen Echos



Beispiel: Knoten X wird erst von 10, dann von 20 erobert.

Eroberer-Nachrichten (= Explorer des stärkeren) laufen "direkt" in Richtung des gegnerischen Initiatorknotens

- Kommt statt erwartetem Echo dann ein fremder Eroberer:
  - Fremder > : verloren, dieser erobert nun...
  - Fremder < : Jetzt doch Explorer in diese Richtung senden, da der "Vasall" offenbar nicht stark genug war, um sich durchzusetzen

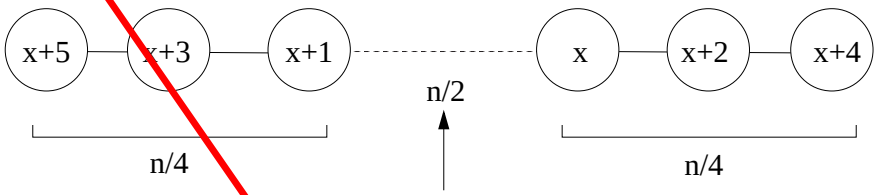
---

- *Eigenschaften* dieser Variante:
  - Nur ein einziges Echo pro Knoten (das ist oft praktisch!)
    (Interpretation: "Verschiedene" Echo-Wellen vereinigen sich)
  - Nicht jeder Knoten wird über seinen Misserfolg informiert
    (Gewinner kann aber über "seinen"Baum eine Informationswelle starten)

- Es sind noch einige andere Varianten denkbar...

Nachrichtenkomplexität von Echo-Election

- *Worst-case* Nachrichtenkomplexität ist $O(n^2)$



Strecke kann u.U. von jedem der n/2 skizzierten Initiatoren durchlaufen werden (wenn diese geeignet zeitversetzt "zünden")

- *Mittlere* Nachrichtenkomplexität ist $O(e \log k)$

Anzahl der Initiatoren

Hier nur *Beweisskizze:*

- Ein Knoten wird im Mittel $H_k \approx \log k$ mal erobert (--> Anzahl der Rekorde!)
- Eroberter Knoten sendet e/n Nachrichten im Mittel (Explorer und Echos)
  --> n H_k (e/n) = e H_k Nachrichten insgesamt.

- Bemerkungen:
  - Empirische Untersuchungen zeigen, dass die Adoptionsvariante bei typischen Graphen und mehreren Initiatoren ca. 30% - 50% derNachrichten spart. (Bei nur einem Initiator sind die Verfahren identisch!)
  - Es gibt Verfahren mit geringerer worst-case Nachrichtenkomplexität, diese sind allerdings um einiges aufwendiger!

Nachrichtenkomplexität: untere Schranke

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

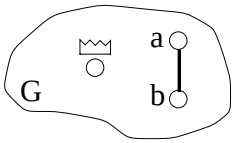
was dominiert typischerweise?

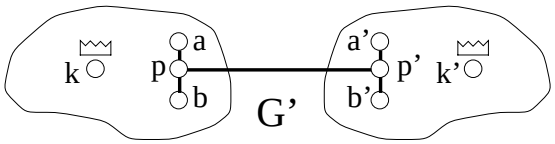
Beweis: *n log n* ist untere Grenze, da der Satz auch für Ringe gilt (vgl. dortigen Satz ohne Beweis)

Also noch zu zeigen: *e* ist untere Schranke

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

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





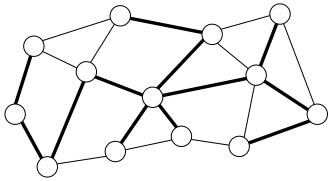
Da zwischen den beiden Teilen keine Nachricht ausgetauscht wird, gewinnen u.U. *zwei* Prozesse k, k'!

Beachte bei diesem Beweis:

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

Verteilte Spannbaumkonstruktion?

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



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

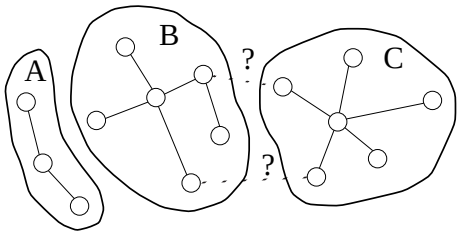
---

Problem: - Wer initiiert die Spannbaumkonstruktion?

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

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

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



- Initial: Jeder Knoten ist ein Fragment
- Algorithmus von Gallager, Humblet, Spira (GHS) 1983 mit 2e + 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

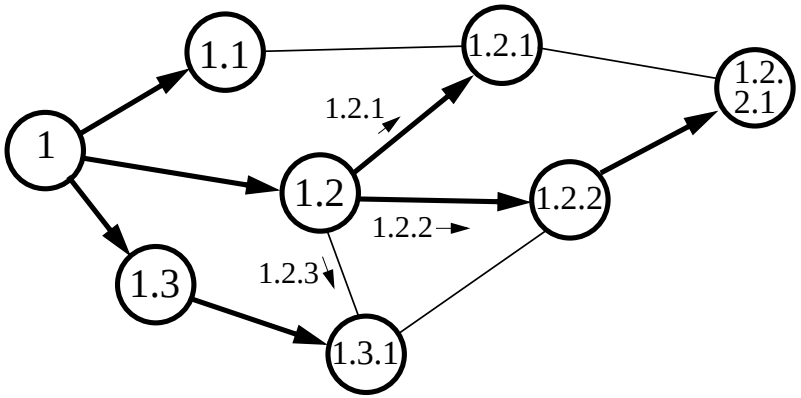
---

- 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?

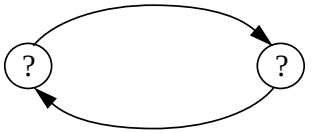
Election in anonymen Netzen?

- 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.

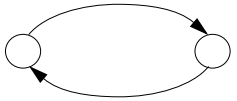
- Falls die Folge endlich ist, könnte der Endzustand symmetrisch sein --> keine Symmetriebrechung möglich!

Ein probabilistischer Election-Algorithmus

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

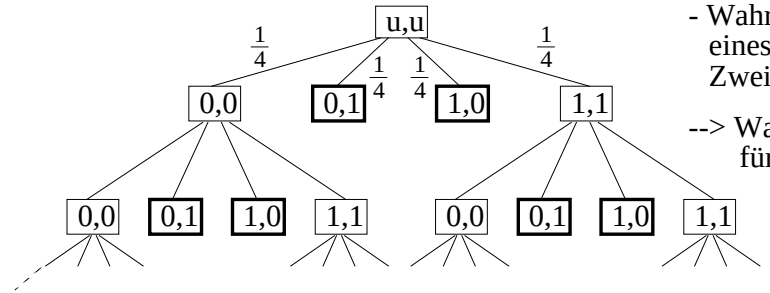
zufälliges Ergebnis 0 oder 1

- hier: *anonyme* Ringe der Grösse 2



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

- Geht es nicht auch mit einem *deterministischen* Algorithmus?
- Beachte: *"korrekt" ≠ "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"!

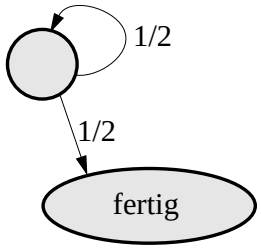


- Wahrscheinlichkeit eines unendlichen Zweiges = 0

--> Wahrscheinlichkeit für leader = 1

Mittlere Nachrichtenkomplexität?

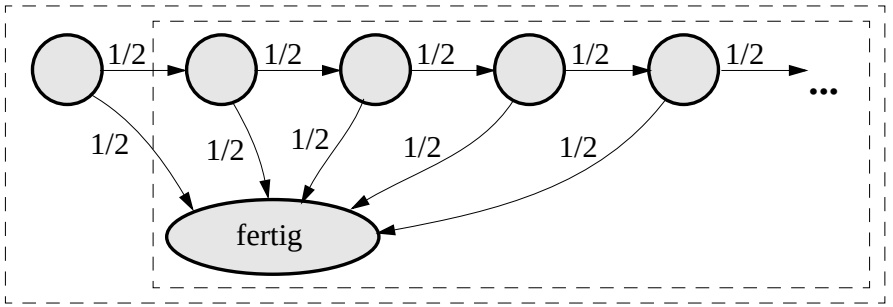
- Beachte: Keine Schranke für *worst-case* Komplexität!
- Die mittlere ("erwartete") Nachrichtenkomplexität lässt sich mittels *Markow-Ketten* leicht ermitteln



- 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* --> $\exists$ 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* $\leq e\ (n/n-1)$
  - vgl. mit vorherigem Algorithmus für Ringgrösse 2

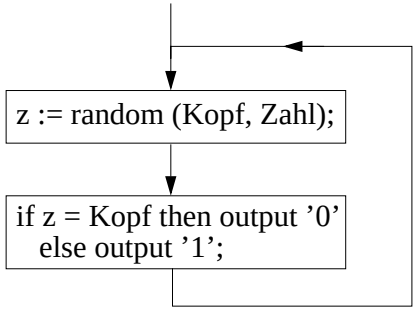
  2.718281828...

- 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...

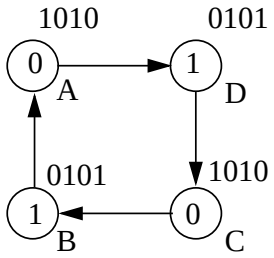
- Wie wäre es statt dessen mit einer "lazy" Variante, die notwendige Nachkommastellen nur auf Anfrage produziert?
  - etwa: liefere zunächst nur die ersten 32 Nachkommastellen; wenn mehr benötigt werden, fordert man das Objekt "zufällige reellwertige Zahl zwischen 0 und 1" auf, weitere Stellen zu liefern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

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.

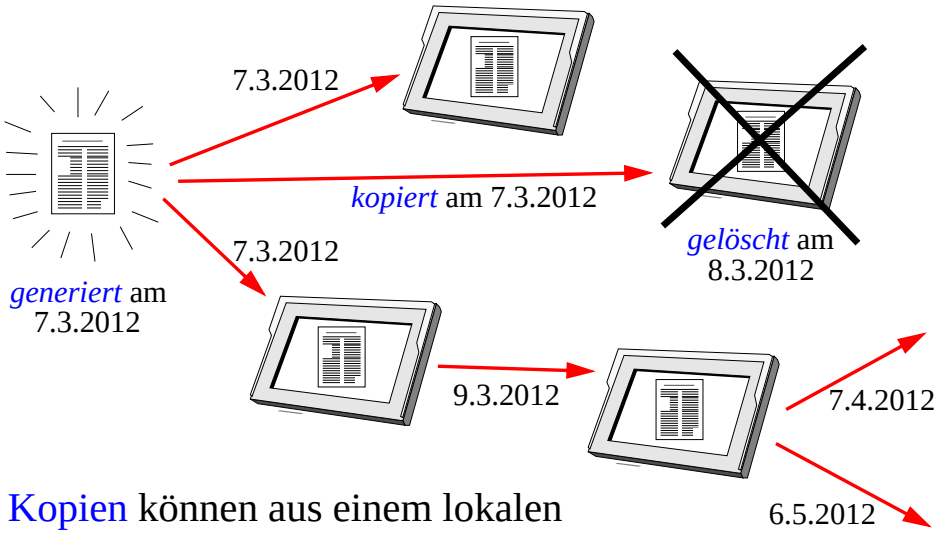
- Wir notieren ohne Beweis: Es werden höchstens $O(n^3)$ Nachrichten generiert; es sind am Ende alle $\bar{n}_p$ identisch mit $\bar{n}_p \leq n$; und es ist $\bar{n}_p = n$ mit einer Wahrscheinlichkeit $\geq 1-(n-1)\,(1/r)^{n/2}$. (Die Wahl von r ist also "kritisch".)
- Varianten, bei denen der Schätzwert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

Kausaltreues Beobachten

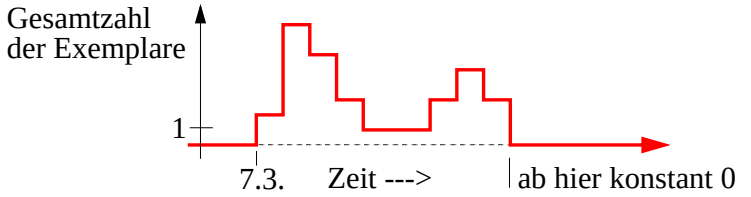
***Beob*achten *ve*rteilter Systeme**

Aussterben einer Menge von Kopien

- Ein Beispiel: on-line Kopien einer Zeitung:



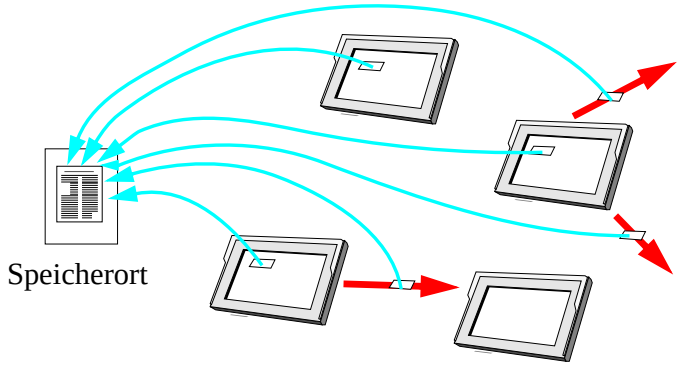
- Kopien können aus einem lokalen Exemplar erzeugt und verteilt werden
- Exemplare werden gelegentlich gelöscht



- 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
  - vgl. auch "Hyperlinks" im WWW
  - bei Bedarf könnte eine echte Kopie angefordert werden ("copy by need")



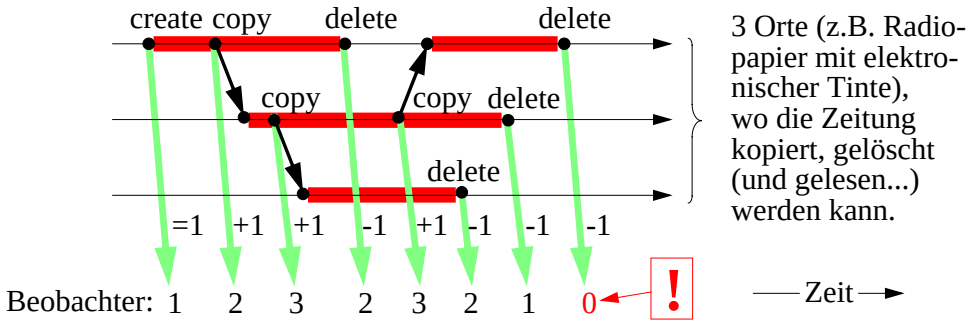
- Copy --> Übermitteln der Referenz (=Zugriffspfad / Adresse)
- Delete --> Löschen der Referenz

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

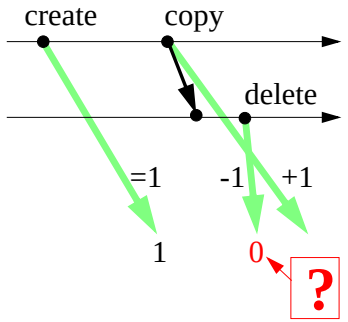
--> *Garbage-Collection*-Problem in verteilten Systemen!

Zählen der Exemplare?

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



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

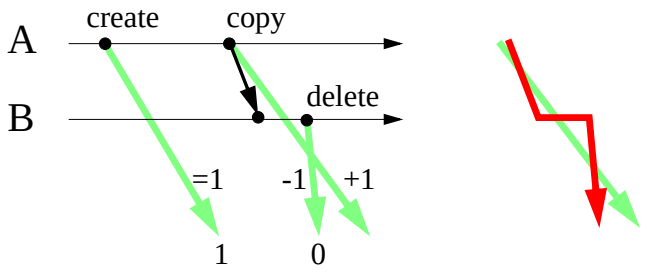


- Beachte: delete-Ereignis ist eine *kausale Konsequenz* des copy-Ereignisses ("ohne copy auch kein delete")
- Beobachter sieht jedoch die Konsequenz *vor* ihrer Ursache!

- Beobachter meint *fälschlicherweise*, dass die Zeitung unwiederbringlich verloren sei!

Was ist der Grund für das Problem?

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



- Eine Einzelnachricht (Meldung von "copy") wurde in indirekter Weise *überholt* (Pfad von copy-Nachricht und delete-Meldung)

- Auf dem Überholpfad können Ereignisse liegen (hier z.B. "delete"), die "Konsequenzen" des überholten Ereignisses ("Ursache", hier "copy") darstellen

- Vermutung: Vermeiden von (direkten oder indirekten) Überholungen löst das Problem, d.h. liefert immer kausaltreue Beobachtungen

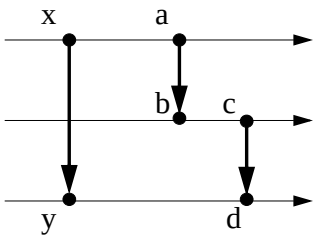
---

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

- Wie könnte man also das Überholen von Nachrichten durch Pfade anderer Nachrichten vermeiden? **?**

Vermeiden (indirekter) Überholungen?

1. Idee: Verwendung von *synchroner* Kommunikation
   - zumindest bei Meldungen an den Beobachter



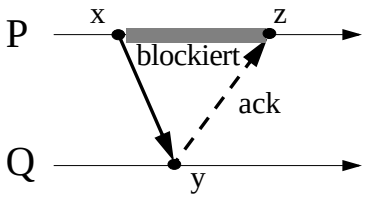
Falls Nachrichtenpfeile *senkrecht* sind (Nachrichten also keine Übertragungszeit benötigen), kann es keine direkten oder indirekten Überholungen geben: jede *später* ausgesandte Nachricht (die den Anfang eines Überholpfades bilden könnte), kommt auch *später* an

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

Informell: "Blitzschnelle" Nachrichten kann man nicht überholen

---

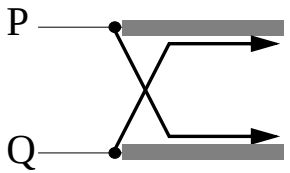
2. Idee: Sender so lange blockieren, bis der Empfänger ein *Acknowledgement* zurückgesandt hat
   - aber das ist ja nichts anderes als eine Implementierung / Simulation synchroner Kommunikation!



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

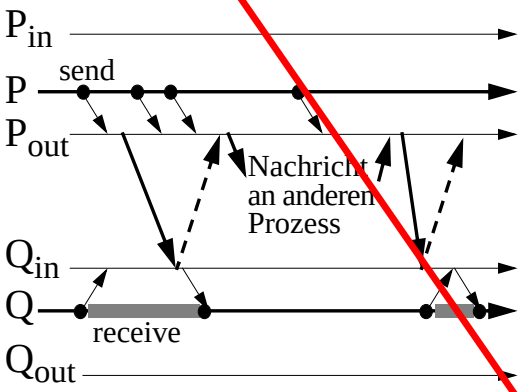
(Nachrichtenpfeile verlaufen nie "rückwärts" in der Zeit)

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



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?

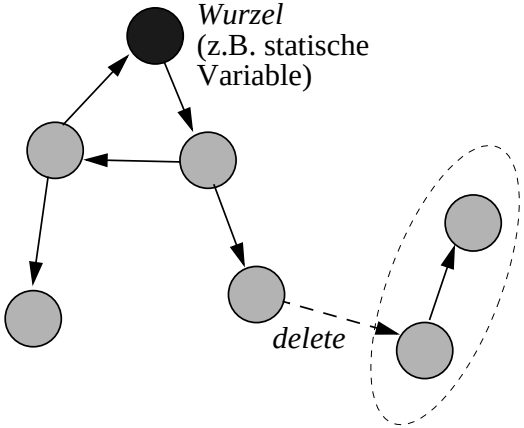
hilft hier die logische Zeit von Lamport?

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

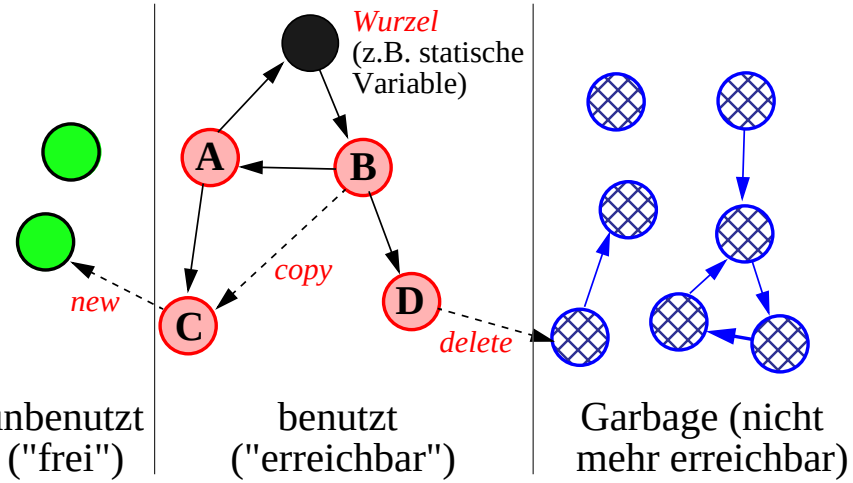
- Typischerweise läuft auch der Garbage-collector (echt oder im Zeitmultiplex) parallel zur Anwendung

==> 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)	identifiziert Garbage