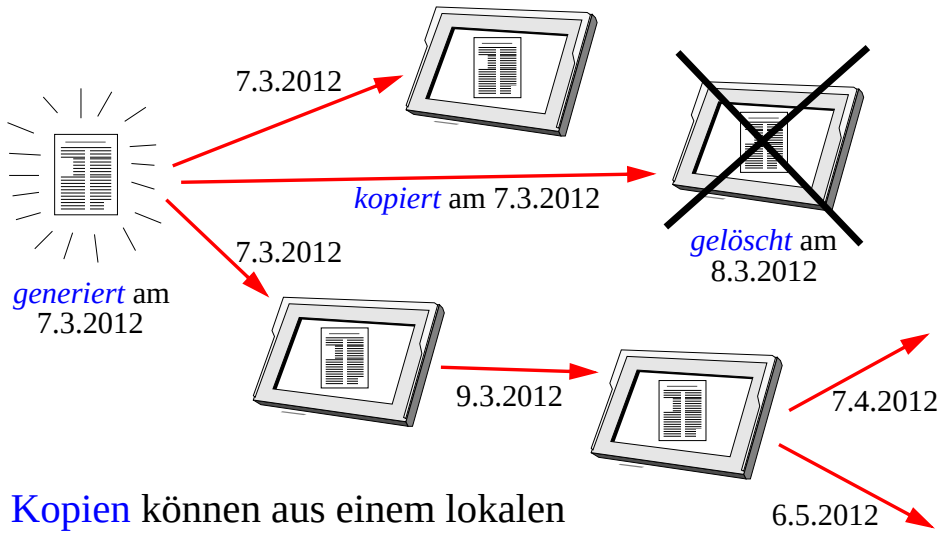
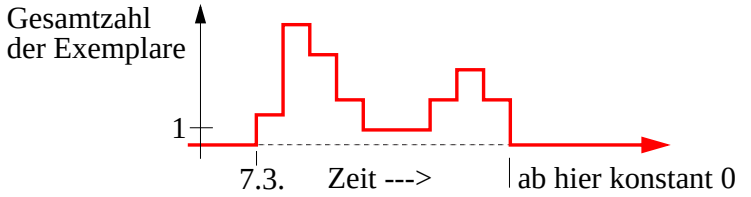
Kausaltreues Beobachten

Aussterben einer Menge von Kopien

- Ein Beispiel: on-line Kopien einer Zeitung:



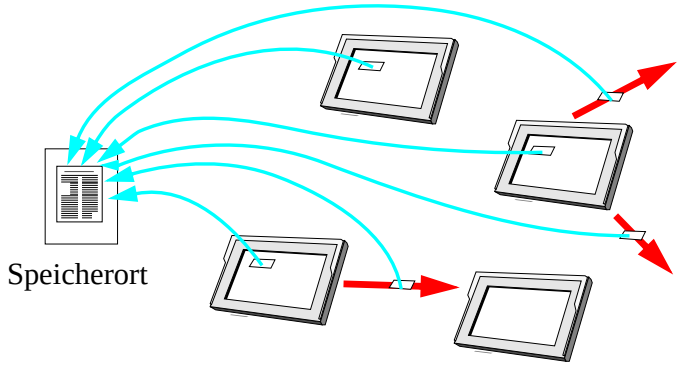
- Kopien können aus einem lokalen Exemplar erzeugt und verteilt werden
- Exemplare werden gelegentlich gelöscht



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

Beobachten des Referenzzählers?

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



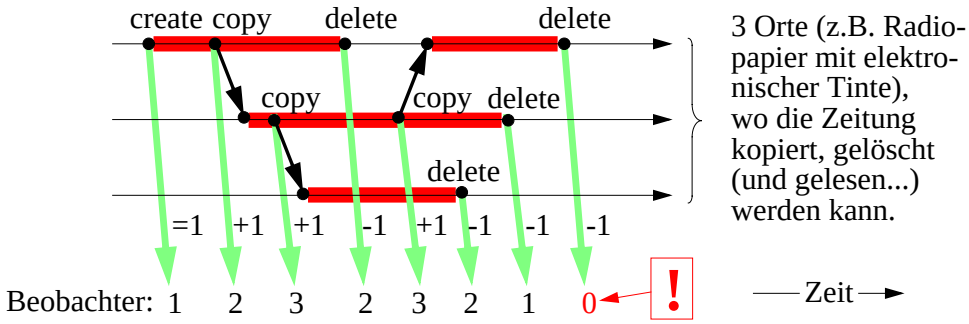
- Copy --> Übermitteln der Referenz (=Zugriffspfad / Adresse)
- Delete --> Löschen der Referenz

- "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!

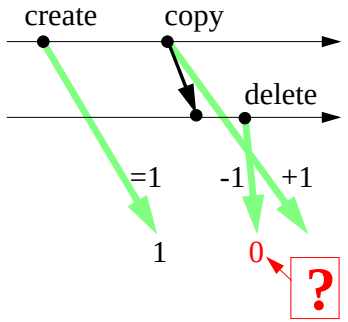
--> *Garbage-Collection*-Problem in verteilten Systemen!

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!

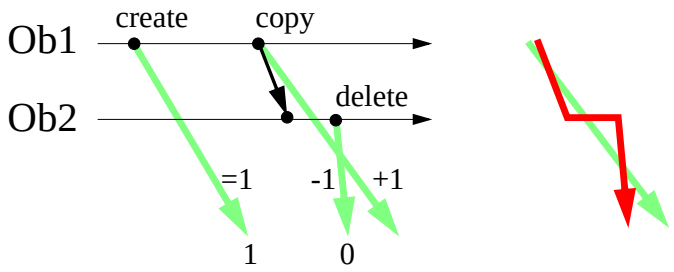


- Beachte: delete-Ereignis ist eine *kausale Konsequenz* des copy-Ereignisses ("ohne copy auch kein delete")
- Beobachter sieht jedoch die Konsequenz *vor* ihrer Ursache!

- 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

---

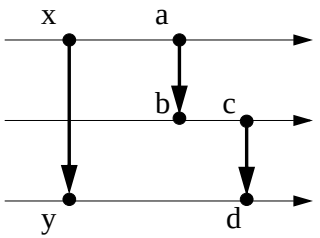
- Nicht-kausaltreue Beobachtungen sind die Ursache für viele konzeptuelle Probleme verteilter Systeme! **!**
  - z.B. verteilte Terminierung, Schnappschuss, Deadlockerkennung...

- 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

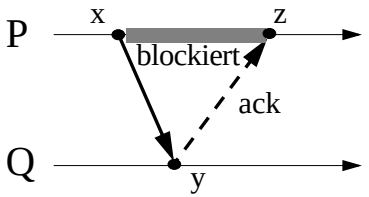
$t(y) = t(x) < t(a) = t(b) < ... < t(c) = t(d) \quad ==> \quad t(y) < t(d)$

Informell: "Blitzschnelle" Nachrichten kann man nicht überholen

---

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

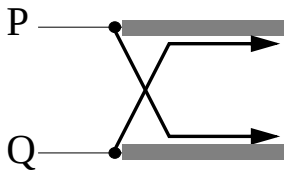
   - aber das ist ja nichts anderes als eine Implementierung / Simulation synchroner Kommunikation!



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

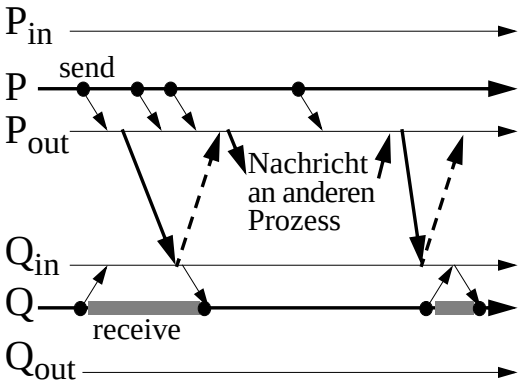
(Nachrichtenpfeile verlaufen nie "rückwärts" in der Zeit)

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!