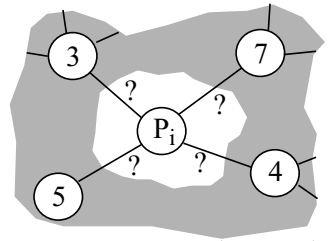


Nachbarschaftswissen

- Zweck: Lerne Identitäten Deiner Nachbarn kennen
(Idee: "Ich heiße i, wie heißt Du?")

- Hier nachrichtengesteuert für P_i :
(vgl. verteiltes Approximationsschema)



Netz zusammenhängend mit bidirektionalen Kanten

```

{Nachricht <j> kommt an}
if not vorgestellt then
    send <i> to all neighbors;
fi;
Nachbarn := Nachbarn  $\cup$  {j};
vorgestellt := true;
    
```

↙ mehrere?

Einer muss "spontan" mit der Vorstellung beginnen

Verteilte Terminierung

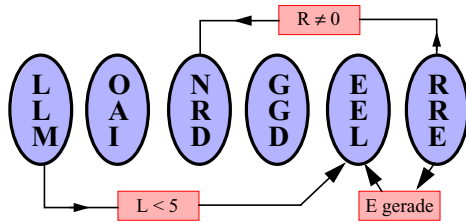
- *global terminiert*, wenn
 - vorgestellt = true bei allen Prozessen und keine Nachricht unterwegs
 - *oder*: alle kennen alle
- *lokal terminiert* (genügt meistens!), wenn zu jeder inzidenten Kante die zugehörigen Nachbarknoten bekannt sind
- 2e Nachrichten (bei Ring also 2n); Zeitkomplexität?

Verwendung als:

- vorgeschalteter eigener Algorithmus
- "piggybacking" der Vorstellungsnachrichten mit eigentlichen (jew. "ersten") Anwendungsnachrichten

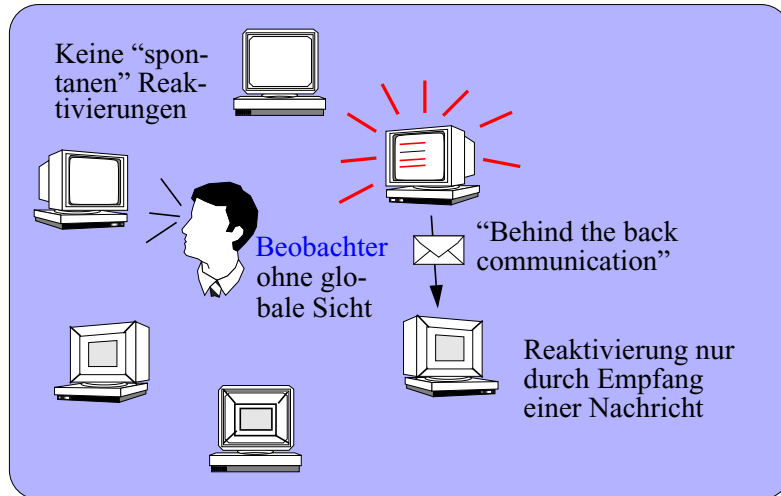
Das Problem der Terminierung

- Bsp: Zahlenrätsel (oder ggT) auf einem PC-Cluster



- pro Spalte (bzw. "Philosoph") jeweils ein Display
 - dort jeweiligen Zustand und neue Wertemengen anzeigen

aktiv oder passiv

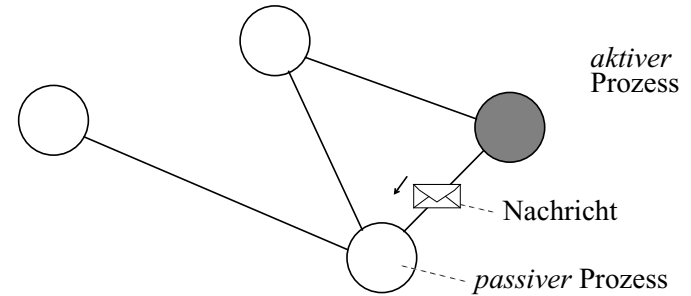


Erkennung der verteilten Terminierung?

alle passiv und keine Nachricht unterwegs

Terminierungserkennung

- Dem Zahlenrätsel- und dem ggT-Beispiel gemeinsam ist ein etwas abstrakteres Modell:



- Es gibt aktive und passive Prozesse sowie Nachrichten

- dabei kann eine ankommende Nachricht einen passiven Prozess wieder reaktivieren

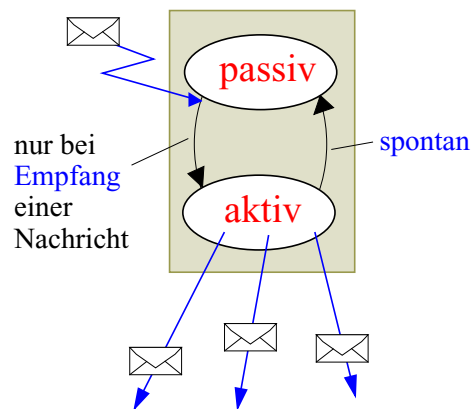
- Man möchte "irgendwie" erkennen, ob alle Prozesse passiv sind und keine Nachricht mehr unterwegs ist

- Stagnationszustand beim verteilten Approximationsparadigma

Verteilte Terminierung: Modell und Problemdefinition

Nachrichtengesteuertes Modell einer vert. Berechnung:

- Prozesse sind *aktiv* oder *passiv*
- Nur aktive Prozesse versenden Nachrichten
- Prozess kann "spontan" passiv werden
- Prozess wird durch ankommende Nachricht reaktiviert



Problem:

Feststellen, ob (zu einem Zeitpunkt)

- alle Prozesse passiv sind
- keine Nachricht unterwegs ist

- "globales Prädikat"
- "stabiler Zustand"

Die Aktionen der Basisberechnung im nachrichtengesteuerten Modell

Prozess p:

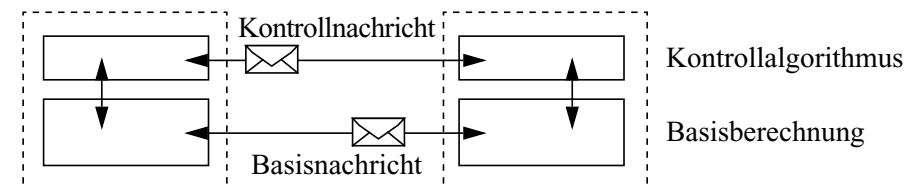
- $S_p: \{ \text{Zustand} = \text{aktiv} \}$
send message $\langle M \rangle$ **to ...**
- $R_p: \{ \text{Eine Nachricht ist angekommen} \}$
receive $\langle M \rangle$; **Zustand := aktiv**
- $I_p: \{ \text{Zustand} = \text{aktiv} \}$
Zustand := passiv

"guard": Prädikat über dem lokalen Zustand

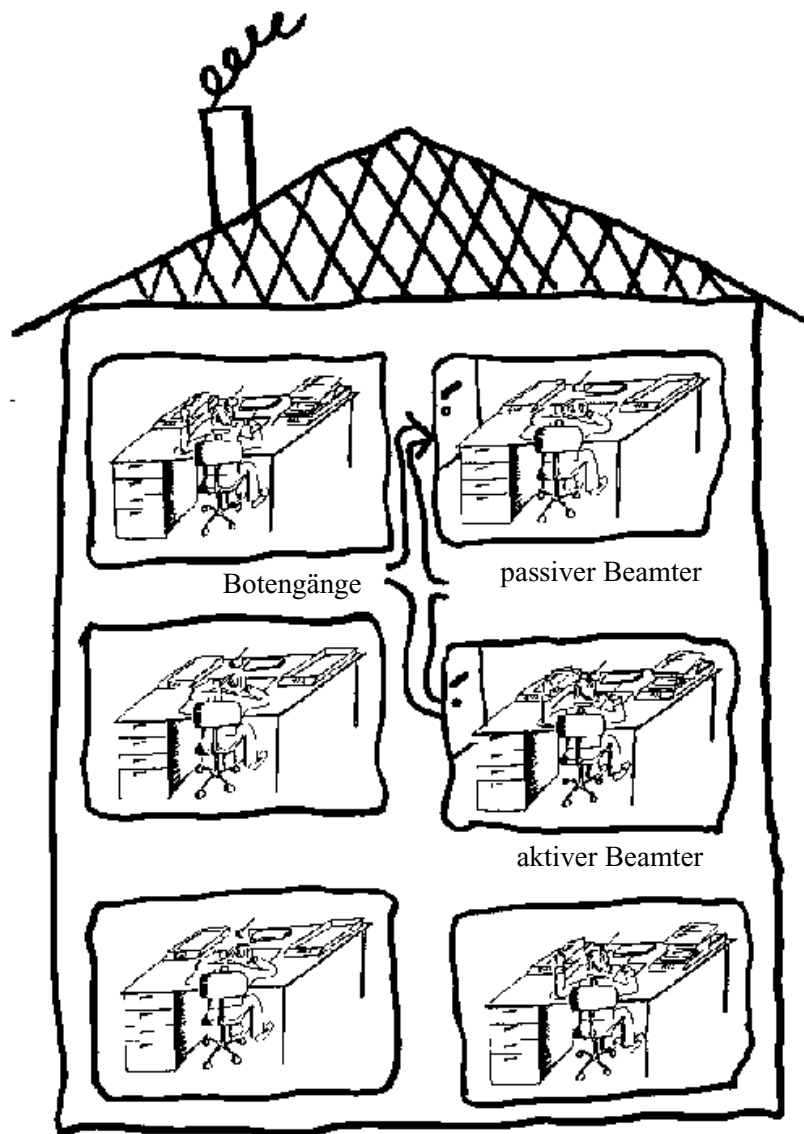
Typischerweise werden die Aktionen als "atomar" betrachtet

Abstraktes Verhalten einer verteilten Berechnung hinsichtlich Terminierung (ggf. existieren weitere Aktionen)

- Durch einen "überlagerten" Kontrollalgorithmus werden weitere Aktionen hinzugefügt
- "Anreichern" der Semantik der Basisberechnung für Zwecke des Kontrollalgorithmus
 - z.B. Verändern spezifischer (lokaler) Variablen
- Überlagerter Algorithmus soll Basisberechnung nicht stören
 - darf aber die Variablen, die der lokalen Kommunikation mit dem Basisalgorithmus dienen, lesen und schreiben

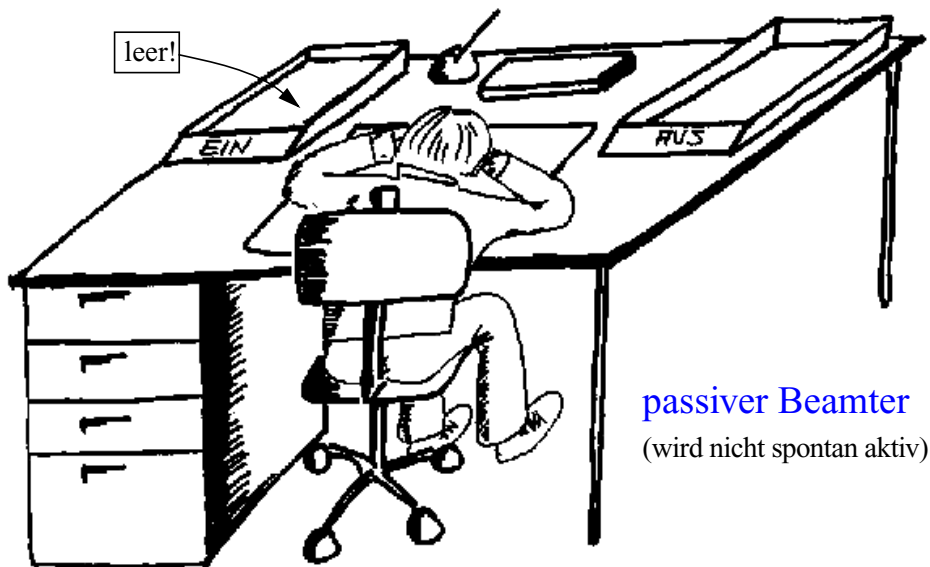


Eine typische Behörde



Die Funktionsweise der Behörde

- (1) Publikumsverkehr nur bis 12 Uhr
- (2) Schliesst erst, wenn **alle Vorgänge bearbeitet**
- (3) Vorgänge werden von Beamten erledigt
- (4) Die Bearbeitung eines Vorganges **kann neue Vorgänge** für andere Beamte **auslösen**
- (5) Aktenaustausch per (bel. langsame) Boten
- (6) **Keiner** hat den **Gesamtüberblick**
- (7) Beamte sind **aktiv** oder **passiv**
- (8) Ein Beamter wird **nicht spontan aktiv**



Terminiert, wenn **alle passiv** und **nichts "unterwegs"**

Das ist ein stabiler Zustand!

Variante: Beamter lässt sich während der Arbeit nicht stören (anklopfen/warten auf "herein") -->

! → **Beamte scheinen immer passiv (Atommodell)**

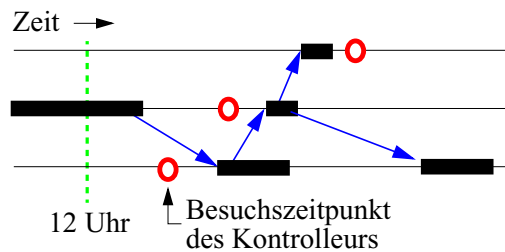
Speedup ist durch die Maximalzahl gleichzeitig aktiver Beamten begrenzt

Dieser ist oft erstaunlich niedrig...

Das schiefe Bild des Kontrolleurs

- *Kontrolleur* wandert durch die Behörde, um die **Terminierung feststellen** zu können
- *Problem*: **Wie** stellt der Kontrolleur fest, ob der stabile Terminierungszustand eingetreten ist?

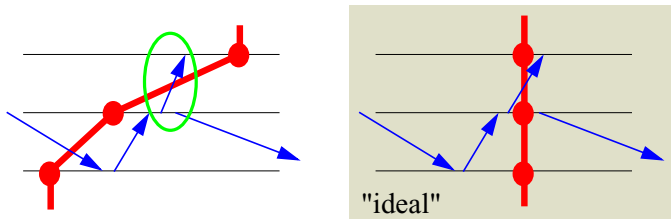
- Die *Illusion* Kontrolleurs:



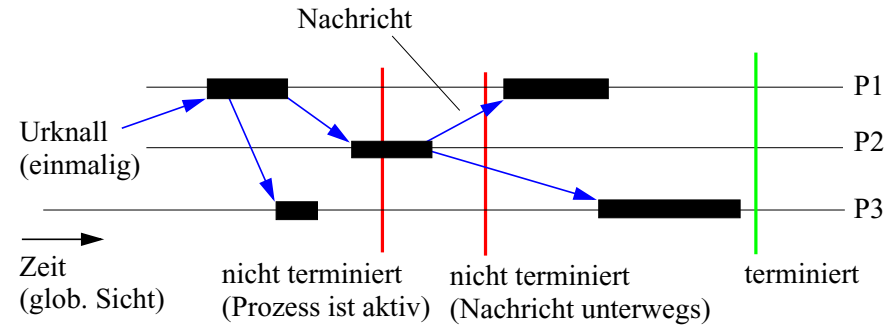
"behind the back communication"

- Alle Beamten stets passiv
- \sum Nachrichten versendet = \sum Nachrichten empfangen

- Kontrolleur macht sich ein *schiefes Bild!*

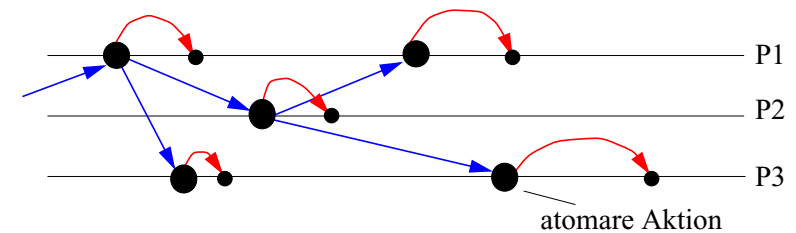


Zeitdiagramme und Atommodell



Idee: Dauer der **Aktivitätsphasen "gegen Null"** gehen lassen

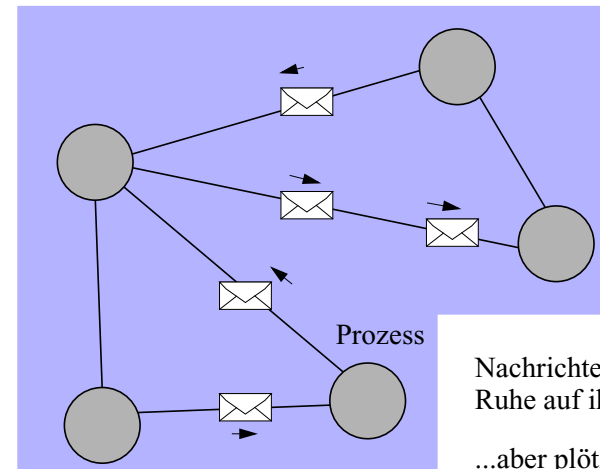
Modellierung: Prozess sendet (**virtuelle**) **Nachricht an sich selbst**, sobald er aktiv wird; ist "unterwegs", solange er aktiv ist



Terminiert (Atommodell) \iff **Keine** (echte oder virtuelle) **Nachricht unterwegs**

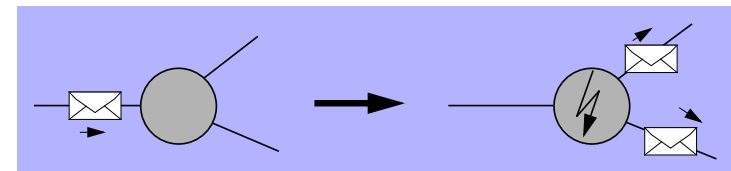
Zur Lösung des Terminierungsproblems also feststellen, **ob noch Nachrichten unterwegs sind**


Globale Sicht 'atomarer' Berechnungen



Nachrichten fließen in aller Ruhe auf ihr Ziel zu...
...aber plötzlich "explodiert" ein Prozess, wenn er von einer Nachricht getroffen wird!

idealisierter Beobachter

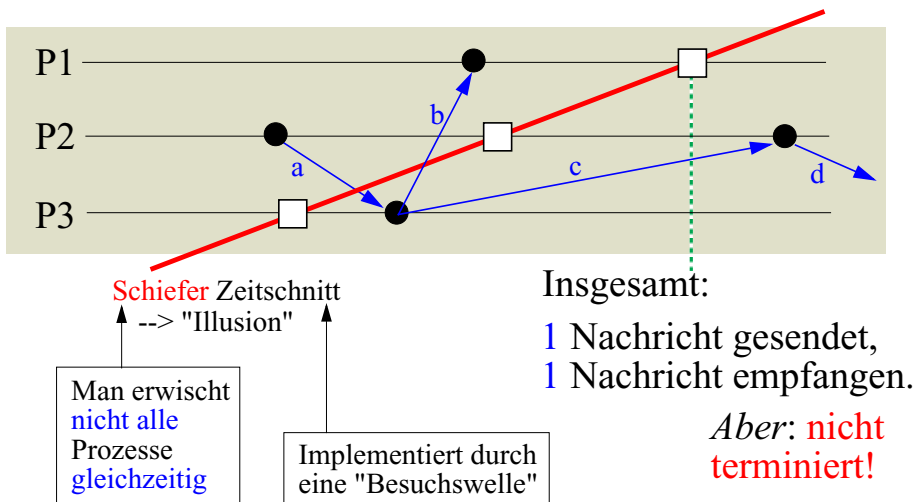


Terminiert, wenn in der globalen Sicht kein  existiert

- Statt im "passiv/aktiv-Modell" genügt es offenbar, im Atommodell die Terminierungserkennung zu lösen (wieso?)
- Wie sehen die Aktionen der Basisberechnung in diesem Modell aus?

Verteilte Terminierung: Lösungen durch Zählen von Nachrichten?

- Genügt das (verteilte) **Zählen** von **gesendeten** und **empfangenen** Nachrichten?
- Einfaches Zählen genügt nicht, **Gegenbeispiel:**



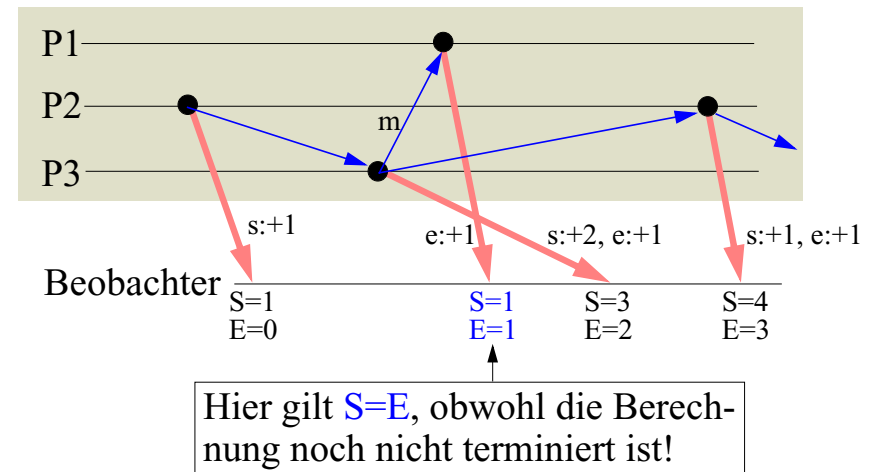
Ursache (informell):

- **Nachricht** aus der **"Zukunft"**
 - kompensiert die Zähler
- **Inkonsistenter Schnitt**
 - ist nicht äquivalent zu einem senkrechten Schnitt

Lösung durch "Ursachenvermeidung"? Ideen vielleicht:

- Nachrichten aus der Zukunft *vermeiden* oder zumindest *erkennen*?
- Senkrechten Schnitt simulieren durch *Einfrieren* der Prozesse?

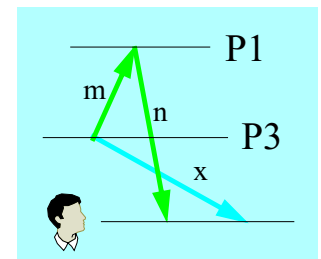
Beobachter über gesendete und empfangene Nachrichten informieren?



- Gleiches Szenario wie eben: Beobachter erfährt, dass m **empfangen** wurde, aber nicht, dass m **gesendet** wurde!

- Man beachte auch, dass hier eine Nachricht (x) **in indirekter Weise** (via m und n) **"überholt"** wurde!

Vermutung: Wenn Informationsnachrichten **nicht** (indirekt) **überholt** werden können, dann kann das Phänomen eines **"schiefen Bildes"** **nicht auftreten!**

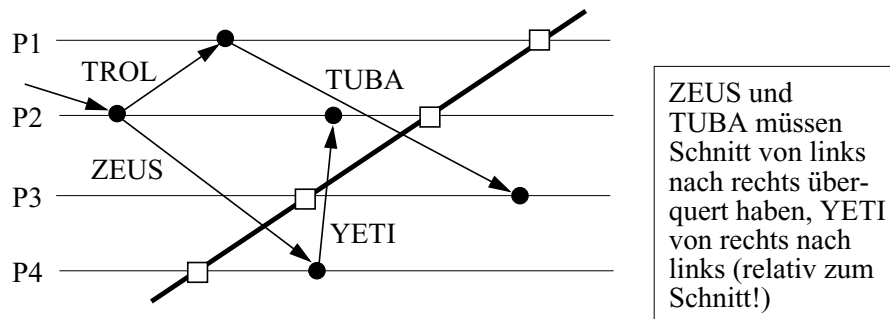


- worauf gründet sich die Vermutung?
- kann man solchermassen korrekte ("kausaltreue") Beobachtungen erzwingen?

Nachrichten eindeutig benennen?

Prinzip: Jede Nachricht bekommt einen (global) eindeutigen Namen:

- TROL, ZEUS, TUBA, YETI,... (?)
- Nachricht kennt ihren Namen
- Sender weiss, welche Nachrichten gesendet wurden
- Empfänger weiss, welche Nachrichten empfangen wurden



- Welle akkumuliert Namen der gesendeten und Namen der empfangenen Nachrichten
- Wenn eine gesendete nicht empfangen wurde, muss sie den Schnitt überquert haben ==> Terminierung nicht melden
- Terminiert, wenn alle "bekanntermassen gesendeten" auch empfangen wurden? (Beweis?)
 - Tip: Wenn keine Nachricht den Schnitt (von links nach rechts??) überquert, ist der Lebensfaden des Systems gerissen; rechts des Schnittes kann dann keine Aktivität mehr entfacht werden (wieso?)

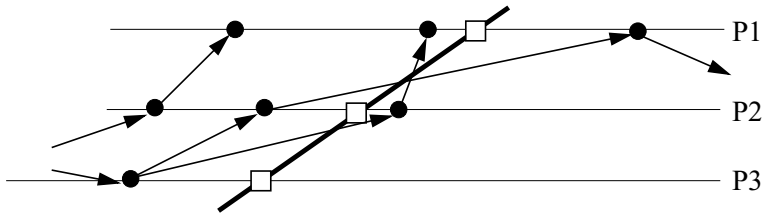
Eindeutige Nachrichtennamen?

- Sender könnte Nachrichten fortlaufend nummerieren und seinen eigenen eindeutigen Namen hinzufügen
 - lässt sich einfacher verwalten als beliebige (global eindeutige) Namen
- Es genügt wohl auch eine fortlaufende Numerierung pro Sender-Empfänger-Beziehung ("Kanal")
 - z.B. 17.4.239 ("239. Nachricht von Knoten 17 an Knoten 4")
 - Verwaltungsaufwand ist recht hoch (bei FIFO benötigt man keine Mengen, es genügen $O(n^2)$ Zähler)

Frage: Wie geht das ganze überhaupt initial los?

Genügt pauschales Zählen pro Kanal? Beweisskizze für das Kanalzählerkriterium

(anstatt Nachrichten pro Kanal individuell zu betrachten)



- Welle stellt folgendes fest:

- auf Kanal P2P1 sind 2 Nachrichten gesendet und 2 Nachrichten empfangen worden
- dennoch überquerte eine Nachricht den Schnitt von P2 nach P1!

- Denkübung: Wäre das bei FIFO-Kanälen korrekt?

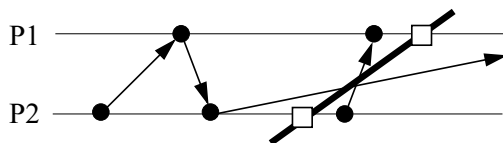
- d.h. wäre dann bei ausgeglichenen Kanalzählern keine Nachricht auf diesem Kanal unterwegs?

Behauptung (auch bei non-FIFO!):

Wenn entlang eines Schnittes *alle* Kanalzähler bzgl. send/receive ausgeglichen sind, dann überquert keine Nachricht den Schnitt

- Wieso? (intuitives Argument?)

- Beweis?

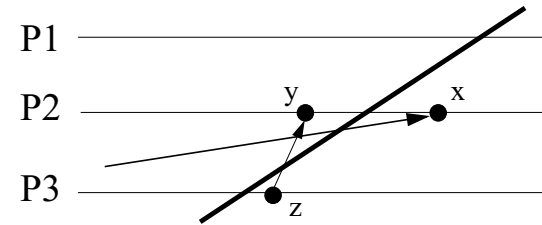


wieso ist das eigentlich kein Gegenbeispiel?

Behauptung: Wenn entlang eines Schnittes pro Kanal gleich viele Nachrichten gesendet wie empfangen wurden, dann ist die Berechnung terminiert

Betrachte frühestes Ereignis (x) *nach* dem Schnitt:

Bei globaler (von links nach rechts fließender) *Zeit* in der Abb. ist dies klar; wenn man ohne solche graphischen Veranschaulichungen auskommen will, muss man statt dessen die *Kausalrelation* bemühen!

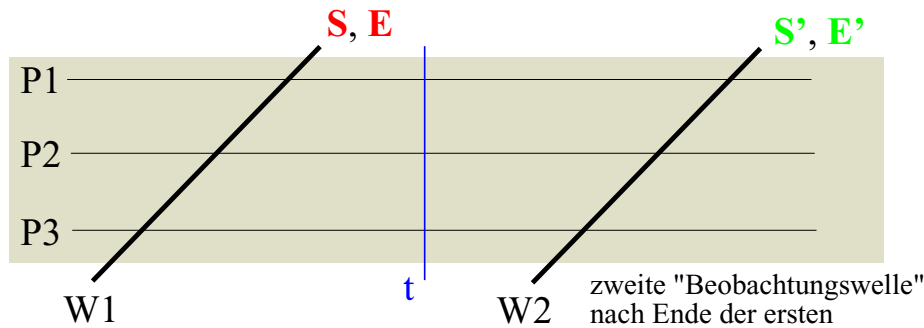


Wir zeigen durch *Widerspruch*: es gibt kein frühestes Ereignis nach dem Schnitt \implies Terminierung

- Dies ist ein Ereignis mit Empfang einer Nachricht, deren Sendereignis *links* des Schnittes liegt
- Zugehöriger Kanalzähler kann nicht getäuscht werden, da für eine *Kompensationsnachricht* gilt: Empfangen (y) vor dem Schnitt, gesendet (z) danach
- Sendereignis der Kompensationsnachricht wäre *früheres* Ereignis *nach* dem Schnitt \implies *Widerspruch*
 - Senden ist immer früher als das Empfangen einer Nachricht!
 - z früher als y, y früher als x \implies z früher als x

Zählen pro Kanal ist aber etwas aufwendig ($O(n^2)$ Zähler); geht es nicht doch mit "ganz pauschalen" Zählern?

Das Doppelzählverfahren



Behauptung: $S=E=S'=E' \implies$ terminiert

d.h. keine Nachricht unterwegs

Beweis (Skizze; lässt sich auch formalisieren):

$S=S' \implies$ Keine Nachricht zwischen W1, W2 gesendet.

$E=E' \implies$ " " " " empfangen.

\implies Werte bei t = Werte von W1.

Also: $S=E \implies$ zum globalen Zeitpunkt t gilt:

Anzahl gesendeter = Anzahl empfangener Nachrichten

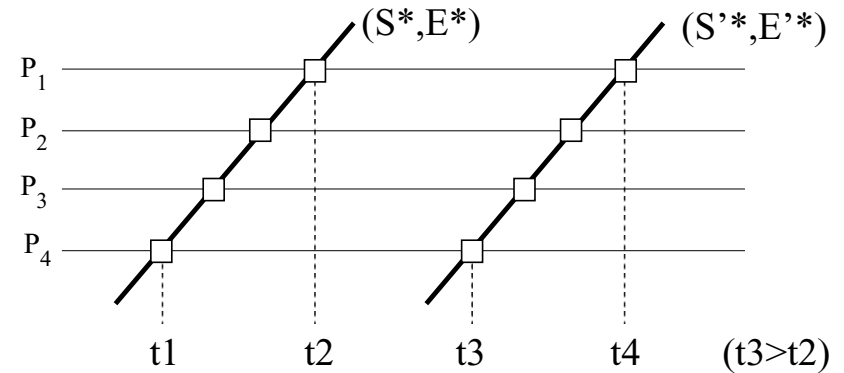
\implies zum Zeitpunkt t ist keine Nachricht unterwegs

\implies zum Zeitpunkt t terminiert

\implies Berechnung war nach W1 terminiert \square

Es gelingt also, für einen bestimmten Zeitpunkt eine **kausaltreue Beobachtung** (als senkrechten Schnitt) im Nachhinein zu *rekonstruieren*!

Formalerer Beweis des Verfahrens



Notation:

- Lokaler Send-Zähler des Prozesses P_i zur Zeit t : $s_i(t)$
- Lokaler Empf.-Zähler des Prozesses P_i zur Zeit t : $e_i(t)$
- $S(t) := \sum s_i(t)$ $E(t) := \sum e_i(t)$

Lemmata:

- (1) $t \leq t' \implies s_i(t) \leq s_i(t')$, $e_i(t) \leq e_i(t')$ [Def.]
- (2) $t \leq t' \implies S(t) \leq S(t')$, $E(t) \leq E(t')$ [Def., (1)]
- (3) $E^* \leq E(t2)$ [(1), e_i wird "eingesammelt" vor $t2$]
- (4) $S'^* \geq S(t3)$ [(1), s_i wird "eingesammelt" vor $t3$]
- (5) Für alle t : $E(t) \leq S(t)$ [Induktion über die atomaren Aktionen]

Beweis:

$$E^* = S'^* \implies E(t2) \geq S(t3) \text{ [(3), (4)]}$$

$$\implies E(t2) \geq S(t2) \text{ [(2)]}$$

$$\implies E(t2) = S(t2) \text{ [(5)]}$$

$$\implies \text{terminiert zum Zeitpunkt } t2 \text{ } \square$$

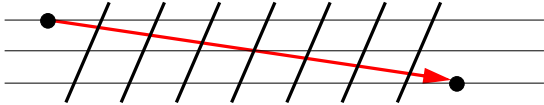
Zwei Zähler genügen!

Anzahl der "in-transit" Nachrichten bei $t2 = 0$

Eigenschaften des Doppelzählverfahrens

- Vergleich des Empfangszählers der ersten Welle mit dem Sendezähler der zweiten Welle (d.h. $E = S'$) ist ein hinreichendes Kriterium
- Falls Terminierung nicht entdeckt wird: Benutze zweite Welle der vorherigen Runde als erste Welle der neuen Runde

"ablaufinvariant"

- Algorithmus ist *reentrant*: Lokaler Zustand des Prozesses wird durch Kontrollalgorithmus nicht geändert
=> jeder Prozess kann unabhängig eine eigene / neue Kontrollrunde starten ("symmetrischer Algorithmus")
 - Viele Varianten (zugrundeliegender Wellenalgorithmus)
 - Anzahl der Kontrollrunden ist a priori nicht durch die Anzahl der Basisnachrichten begrenzt
 - es kann eine ganz langsame Basisnachricht geben, während der beliebig viele Kontrollrunden gestartet werden können
- 
- allerdings ist folgende Variante denkbar: ein Prozess, der eine Basisnachricht erhält ohne eine neue auszusenden, startet eine Doppelrunde (Zeitkomplexität?)

Safety- und Liveness-Eigenschaften

Safety: Something bad will never happen...
(Typisch: "für alle eingenommenen Zustände gilt...")

- Bsp.: Nie mehr als 1 Prozess im kritischen Abschnitt
- Bsp.: Variable x wird nie negativ
- Bsp.: Invariante wird nicht verletzt

oft auch "progress"

schliesslich

Liveness: Something good will eventually happen...
(Typisch: "es wird ein Zustand eingenommen, so dass...")

- Bsp.: Variable x wird schliesslich positiv
- Bsp.: Programm terminiert
- Bsp.: erfolgte Terminierung wird schliesslich auch gemeldet

Korrektheit: Algorithmus erfüllt *Safety und Liveness*

Beispiel verteilte Terminierung:

- aufgesetzter Kontrollalgorithmus
 - sagt "ja", wenn Basisberechnung terminiert ist
 - sagt "weiss nicht" sonst ("nein" geht nicht!)
- safe, aber nicht live: meldet stets "weiss nicht"
- live, aber nicht safe: meldet stets "ja"

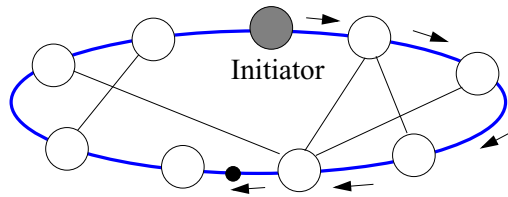
=> "Kunst": Algorithmus, der *safe und live* ist!

=> Es ist stets *safety und liveness* zu zeigen!

Kontrolltopologien

- Die für den Terminierungsalgorithmus benötigten "Wellen" können unterschiedlich realisiert werden:

1.) Ring / Hamilton'scher Zyklus:

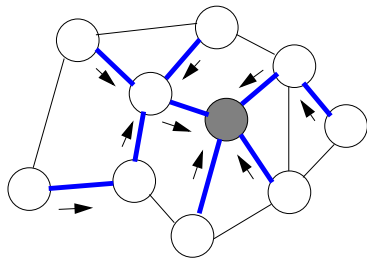


Kontrollnachricht ("Token") besucht zyklisch zwei Mal alle Prozesse und akkumuliert dabei die Zählerstände
 "Logischer" Ring genügt!

- in einem zusammenhängenden ungerichteten Graphen kann ein logischer Ring immer gefunden werden, indem man einen Spannbaum "umfährt"
- Zeit- und Nachrichtenkomplexität: $O(n)$

2.) Spannbaum:

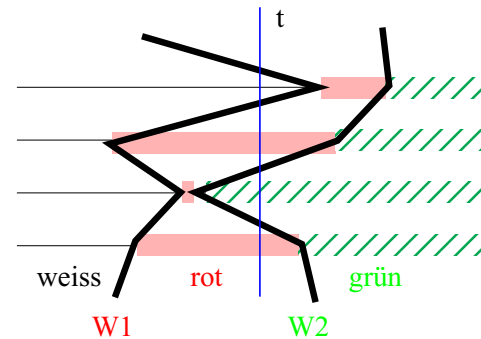
- vereinfachter Echo-Algorithmus: an den Blättern reflektierte Welle sammelt die Zählerstände in akkumulierender Weise ein



- auch hier werden *zwei* solche zum Initiator hinfließende Wellen benötigt
- bei nicht-entartetem Spannbaum: Viele Nachrichten parallel unterwegs \implies bessere Zeitkomplexität!

Echo-Algorithmus für die beiden Wellen des Doppelzählverfahrens?

- Anwendbar für beliebige zusammenhängende Topologien
- Ausnutzen der beiden "Halbwellen" eines einzigen Laufs des Echo-Algorithmus für die beiden Kontrollrunden!

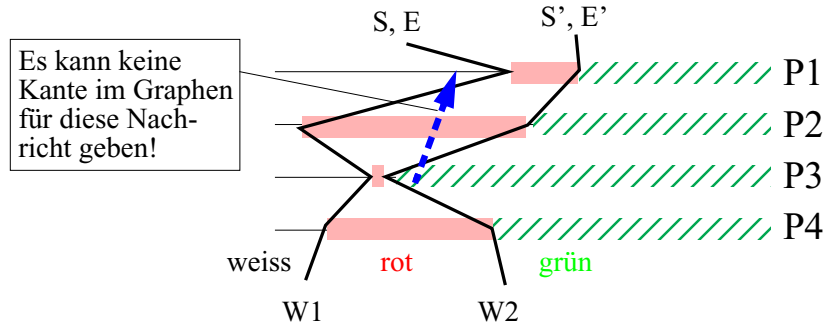


Der Echo-Algorithmus wird als *Transportdienst* zur Realisierung von *zwei* Wellen benutzt, wobei durch die Echo-Nachrichten sowohl die bei W1 lokal gemerkte Information als auch die bzgl. W2 relevante lokale Information an den Initiator übermittelt wird.

- Welle W1: "rot werden"; Welle W2: "grün werden"
- *Behauptung*: Das so realisierte Doppelzählverfahren ist korrekt
- *Problem*: Formeller oder informaler Beweis lassen sich so nicht anwenden, da sich W1 und W2 *überlappen*!

Bemerkung:
 Auf vorhandenen Spannbäumen kann man aber *nicht* einfach die vom Initiator ausgesendete Hinwelle und die reflektierte Rückwelle verwenden!

Zeitzoneverfahren



- "Trick": Es gibt keine nach W2 gesendete Nachricht, die vor W1 ankommt (grüne Knoten haben keine weissen Nachbarn!)

- Wenn ein Knoten grün wird, sind seine Nachbarn bereits vorher rot geworden

--> Täuschung der Zähler durch Kompensation mittels Nachrichten "rückwärts" über 2 Wellen ist unmöglich!

- *Beweisskizze:*

- 1) Im roten Gebiet (d.h. zwischen W1 und W2) findet keine Aktivität statt, wenn das Terminierungskriterium $S=E=S'=E'$ gilt: Dazu müsste eine vor W1 (im weissen Gebiet) ausgesendete Nachricht im roten Gebiet ankommen. Dann ist aber $E' > E$.
- 2) Es kann Nachrichten geben, die beide Wellen "vorwärts" überqueren (d.h. im weissen Gebiet gesendet werden und im grünen ankommen). Solche Nachrichten werden auf S (und S') als gesendet registriert, jedoch weder auf E noch auf E' als empfangen registriert. Da eine Kompensation der Zähler S, E mittels Nachrichten "rückwärts" über 2 Wellen unmöglich ist (wie im Gegenbsp. zum einfachen Zählen), ist dann $S > E$.

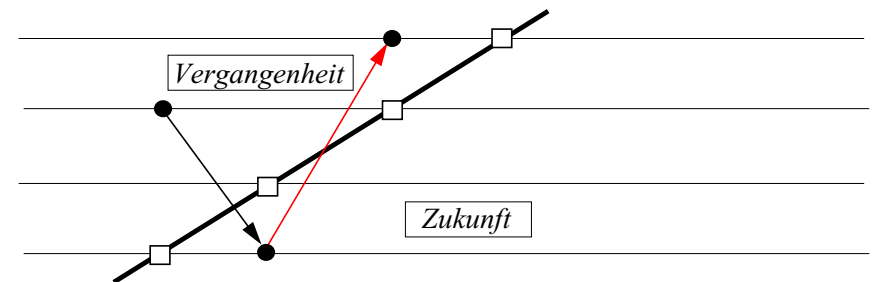
Also: Es gibt keine Nachricht, die W1 überquert; nach W1 findet daher keine Aktivität statt; das System ist nach W1 terminiert

- Erkenntnis: Beim *einfachen Zählen* war eine *irreführende Kompensation* der Zähler durch Nachrichten aus der Zukunft möglich

- Idee: Entlang des Schnitts (induziert durch Welle!) zählen, aber Nachricht aus der Zukunft *erkennen*

- Nachricht aus der Zukunft --> Schnitt inkonsistent

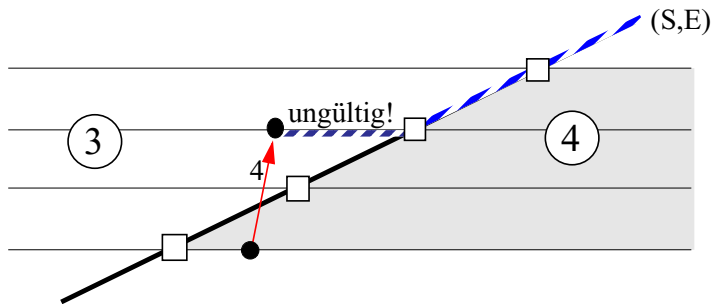
- Keine solche Nachricht --> Schnitt konsistent ("äquivalent" zu senkrechtem Schnitt entsprechend Gummibandtransformation)



- Zählergebnis verwerfen, wenn inkonsistent

- Sequenz von Wellen, bis Terminierung festgestellt (Liveness ist klar: Schnitt nach Terminierung ist konsistent)

Prinzip: *Erkennen inkonsistenter Schnitte*



Lokale Nachrichtenzähler mit einer Welle akkumulieren,
aber Ergebnis *invalidieren*, wenn eine Nachricht
aus der Zukunft empfangen wurde

Implementierung:

- "Piggybacking" der Zeitzonenummer auf Nachrichten
- Flag setzen, wenn eine Nachricht aus einer höheren
Zeitzone (z.B. "4" im Bild) empfangen wurde
- Welle registriert Flag (und setzt es zurück)
und erhöht die Zeitzone

Terminiert, wenn Welle kein gesetztes Flag feststellt
und die beiden akkumulierten Zähler übereinstimmen

(Formaler Korrektheitsbeweis?)