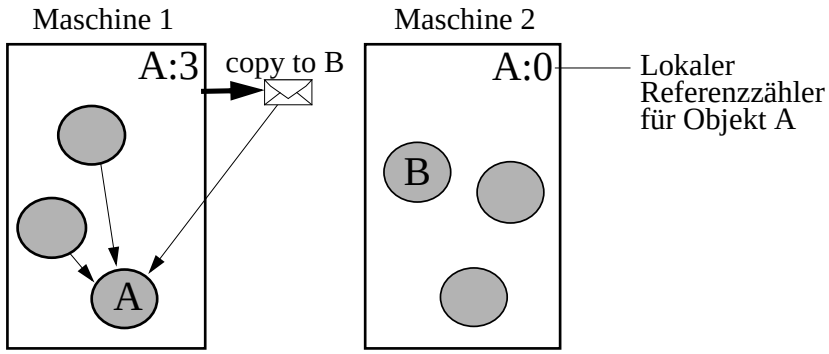
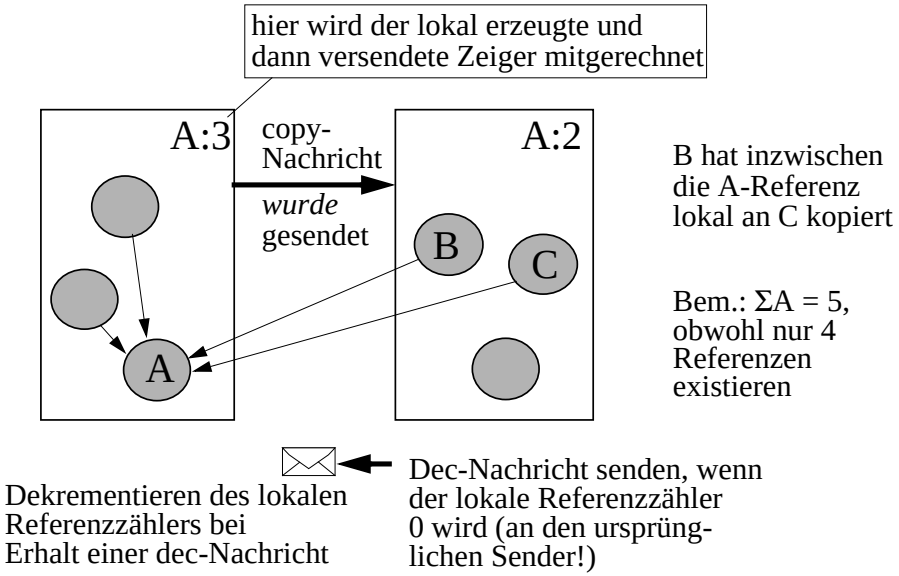
LRC-Garbage-Collection-Verfahren

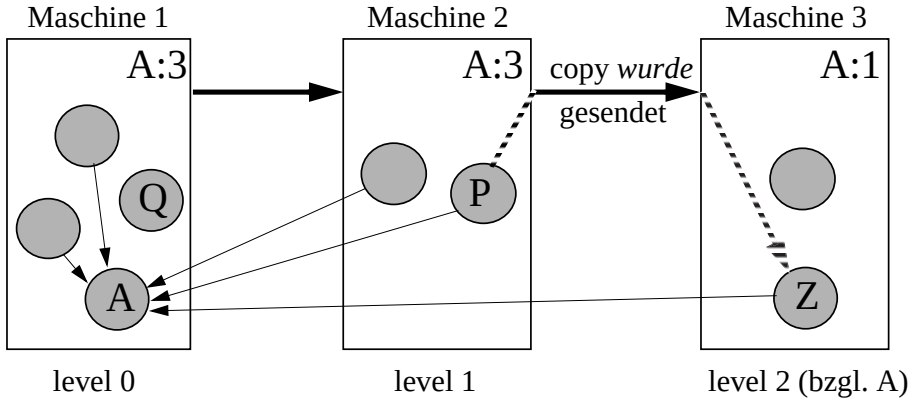
- Pro *Maschine* wird (potentiell) für jedes Objekt ein Referenzzähler gehalten, z.B. für Objekt A:





Beachte: Es wird angenommen, dass der *lokale* Referenzzähler *atomar* mit den Operationen copy, delete, Empfang einer dec-Nachricht aktualisiert werden kann

LRC: Weitervererben von Referenzen



- Maschine 1 muss nicht informiert werden, wenn z.B. Objekt P eine (Kopie seiner) A-Referenz an Z schickt!

- Korrektheit (safety):

Referenzbaum

Lokaler Referenzzähler auf level i+1 > 0
--> lokaler Referenzzähler auf level i > 0

==> Falls eine Referenz existiert, dann ist der Referenzzähler auf level 0 > 0

- wieso eigentlich *Baum*?
- wie genau ist "level" definiert?

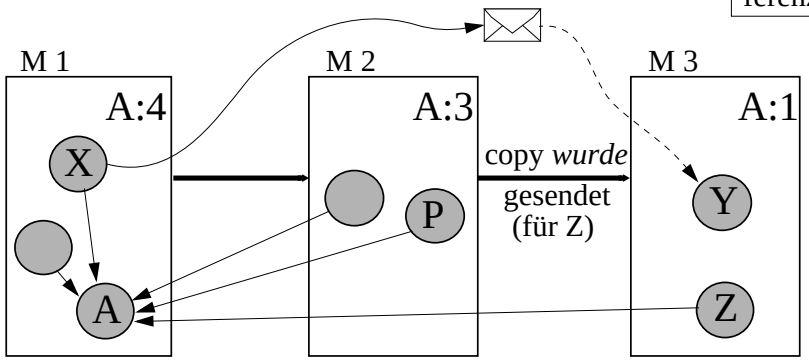
==> "Garbage-Kriterium": Referenzzähler auf level 0 = 0

- Denkübungen: wie steht es mit der *liveness*?
- was geschieht, wenn P eine A-Referenz an Q (Maschine 1) sendet? (--> Zyklen ?)

Der "Remote-ref"-Baum

- Was geschieht, wenn X auf Maschine 1 eine copy-Nachricht mit einer A-Referenz an Y auf Maschine 3 schickt?
  - Maschine 1 erhöht ihren lokalen A-Zähler beim Senden
  - Maschine 3 erhöht ihren lokalen A-Zähler beim Empfang

die schon eine A-Referenz hat



- Aber: Wann und an wen (hier: Maschine 1 und/oder 2) soll ggf. eine dec-Nachricht von Maschine 3 gesendet werden?

?

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

(2) wenn der lok. Referenzzähler 0 wird auf M3 --> an M1 und M2 senden

(3) M2 "adoptiert" Y bei Empfang der copy-Nachricht --> bei Empfang des copy eine dec-Nachricht an M1 senden (als hätte Y seine A-Referenz gleich gelöscht und dann sofort eine lokale Kopie von Z erhalten)

- Beachte bei der Lösung (3):
  - eindeutige Vorgängerbeziehung; keine Zyklen --> *Baum* ("level" klar bestimmt)
  - neuer "Adoptivvater" M2 braucht hierbei nicht informiert zu werden
  - genausogut hätte M1 Objekt Z adoptieren können (M3 sendet dann dec-Nachricht an M2 bei Empfang der copy-Nachricht von X) --> Optimierungspotential: wähle stets den Vater mit niedrigstem level... (wieso?)

LRC: Formale Beschreibung

- Ergänzen existierender atomarer Aktionen bzw. zusätzliche Aktionen:

atomare Aktion: Änderung von LRC "gleichzeitig" mit dem Versenden der copy-Nachricht

C_p: {p ist erreichbar und hat eine r-Referenz}
send copy(r) **to** q;
$LRC_p(r) := LRC_p(r) + 1$

R_p: {Bei Empfang einer copy(r)-Nachricht von q}
Installiere die r-Referenz;
if $LRC_p(r) = 0$
then {$LRC_p(r) := 1$; $FIRST_p(r) := q$}
else {$LRC_p(r) := LRC_p(r)+1$; **send** dec(r) **to** q}
fi

r-Referenz existierte schon bzw. p befindet sich schon im Referenzbaum

D_p: {p hat eine r-Referenz}
Lösche die r-Referenz;
$LRC_p(r) := LRC_p(r) - 1$;
if $LRC_p(r) = 0$ **then call** collect(r) **fi**

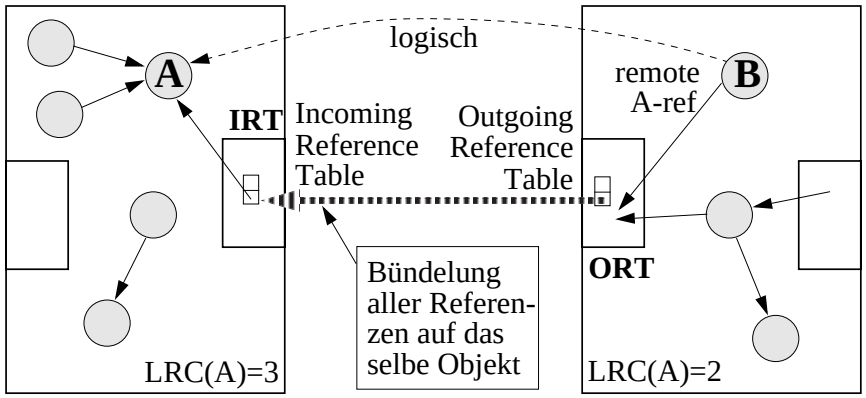
X_p: {Bei Empfang einer dec(r)-Nachricht}
$LRC_p(r) := LRC_p(r) - 1$;
if $LRC_p(r) = 0$ **then call** collect(r) **fi**

ggf. rekursives Freigeben!

proc collect(r):
r wurde lokal erzeugt
if $FIRST_p(r) = \varnothing$ **then** "*r is garbage*"
else send dec(r) **to** $FIRST_p(r)$; **fi**

- beim *lokalen* Kopieren von r wird lediglich $LRC_p(r)$ erhöht
- wieso wird eine dec-Nachricht gesendet, wenn schon eine r-Referenz existiert; könnte man darauf u.U. verzichten?

Implementierungssicht: Remote-reference-Tabellen



- ORT: enthält Schattenobjekte als lokale Stellvertreter ("Proxy") für alle entfernten Objekte
  - beachte: hier ist Σ LRC(A) = 5 = Anzahl A-Referenzen (inkl. bei IRT!),

- IRT / ORT: Die Einträge bzgl. eines bestimmten Objektes (z.B. A) lassen sich als ein *einziges* Objekt, *verteilt* auf mehrere Maschinen auffassen

z.B. weil vom *lokalen Collector* als Garbage erkannt

Idee: Wenn ein Teilobjekt in ORT *gelöscht* wird, dann wird das entsprechende Gegenstück in der IRT gelöscht
(falls es sonst keine Teile in anderen ORT gibt)

--> ORT muss dazu eine Nachricht an IRT senden

- Remote-Referenzen sind ggf. *mehrfach indirekt*
  - u.U mehrfache Adressumsetzung bei *Zugriff* über eine Remote-Referenz, falls der Zugriff tatsächlich entlang der Referenzkette erfogt

LRC: Eigenschaften

- Vergleich zu Verfahren von Lermen/Maurer, Rudalics etc:
  - keine Zusatznachricht bei copy
  - kein Verzögern von copy
  - oft: keine Zusatznachricht bei delete (gelegentlich: Abbau des Referenzbaumes)
  - kein Verzögern bei delete
  - FIFO nicht notwendig
  - Gesamtzahl der Nachrichten: genausoviele dec wie copy (wieso?)

Generell gilt: zyklischer Garbage zwischen verschiedenen Objekten lässt sich mit Referenzzählern nicht erkennen!

- Vergleich zu WRC:
  - Zähler einfacher handzuhaben als die Akkumulation von beliebig kleinen Gewichtsfragmenten
  - keine Komplikation bei RW=1

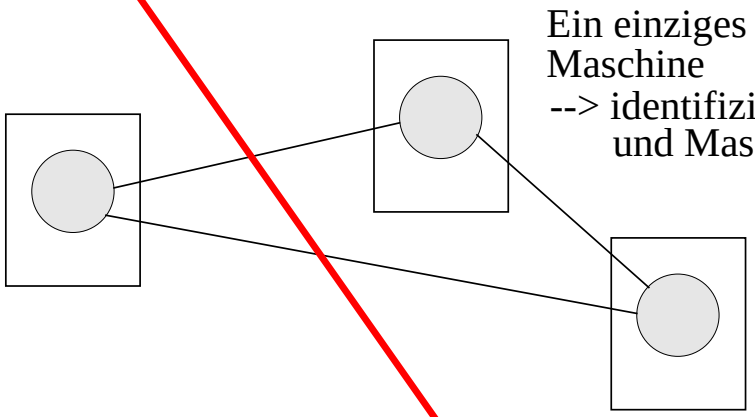
- Nachteil (gegenüber anderen Referenzzähler-Methoden):
  - Objekte besitzen i.a. mehrere (Referenz)zähler (angesiedelt z.B. in mehreren ORT-Tabellen) --> höherer Speicheraufwand

- Implementierung:
  - typischerweise je einen *lokalen Garbage-Collector* pro Maschine; kann nach anderem Verfahren arbeiten, z.B. mark&sweep (Zyklenerkennung!)
  - Proxyobjekte in der IRT werden dabei als Wurzelobjekte angesehen

- Migration von Objekten lässt sich einfach realisieren!
  - nur Zielmaschine und Quellmaschine sind betroffen
  - wie realisiert? (vgl. jeweils Objekte A und B in vorheriger Skizze)
  - wieso einfacher als bei anderen GC-Verfahren?
  - Denkübung: Präzisierung (z.B. Zyklen?)...
    ... und Optimierung (z.B. Verkürzung von Indirektionsketten)

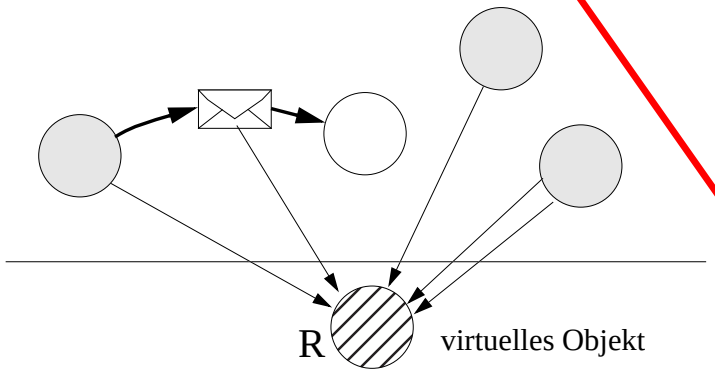
Transformation von LRC in einen Terminierungserkennungs-Algorithmus

Konzeptionelle Sicht:



Ein einziges Objekt pro Maschine
--> identifiziere Objekt und Maschine

- Betrachte zunächst die Variante, wo ein Objekt u.U. mehrere R-Referenzen besitzen kann
- Entspricht dem Fall, dass ein aktiver Prozess noch eine weitere Nachricht empfängt



Die Transformation

1) Identifziere Maschine / Prozess / Objekt

2) Füge ein einziges zusätzliches (virt.) Objekt "R" hinzu

3) *Senden einer Nachricht* --> Kopieren der R-Referenz
   --> *inkrementiere den (!) lokalen Zähler ("für R")*

4) *Empfang einer Nachricht*
   --> *inkrementiere den lokalen Zähler*
   ← auch wenn schon eine lokale R-Referenz existiert...!?

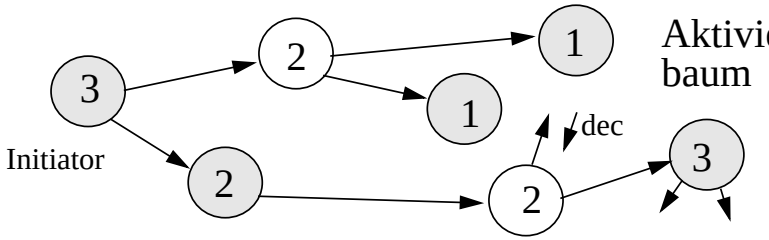
5) *Passiv werden* --> Lösche alle (!) R-Referenzen
   --> *dekrementiere den lokalen Zähler entsprechend oft*
   ← wie oft genau?

6) *Bei Empfang einer dec-Nachricht*
   --> *dekrementiere den lokalen Zähler*

7) *Wenn der lokale Zähler 0 wird*
   --> *sende dec-Nachricht an alle (!) Sender*
   ← müsste man sich also alle merken

8) *Terminiert, wenn der Zähler* des Erzeugers von R (also *des Initiators der Berechnung) 0 wird*

- Optimierung: Behalte nur *erste* R-Referenz, alle anderen R-Referenzen werden sofort wieder gelöscht
  ← nur einen merken!



Das Resultat der Transformation

S_p: {Zustand = *aktiv*}
send <message> **to** ... ; $LRC_p := LRC_p + 1$;

R_p: {Eine Nachricht von q ist angekommen}
receive ...;
if $LRC_p = 0$
then {$LRC_p := 1$; $FIRST_p$:= q; Zustand := *aktiv*;}
else {**send** <dec> **to** q; **if** Zustand = *passiv*
then {Zustand := *aktiv*;
$LRC_p := LRC_p + 1$;}}

I_p: {Zustand = aktiv}
Zustand := *passiv*; $LRC_p := LRC_p - 1$;
if $LRC_p = 0$ **then send** <dec> **to** $FIRST_p$;

X_p: {Bei Empfang einer <dec>-Nachricht}
receive <dec>; $LRC_p := LRC_p - 1$;
if $LRC_p = 0$ **then send** <dec> **to** $FIRST_p$;

- vgl. dies mit "indirekten Acknowledgements" bzw. dem Echo-Algorithmus
- dies ist der *Terminierungserkennungsalgorithmus von Dijkstra und Scholten* (in etwas anderer Darstellung als im Originalartikel, dessen Studium sich lohnt!): E.W. Dijkstra, C.S. Scholten: Termination Detection for Diffusing Computations. Inf. Proc. Lett. 11 (1980), 1-4
- Voraussetzung: Es gibt einen einzigen initial aktiven Prozess ("environment")
- die letzte Zeile von I_p und X_p wird für das "environment" ersetzt durch:
if $LRC_p = 0$ **then** *"terminated"*;
- die letzte if-Bedingung in der Aktion R_p ergibt sich aus der Überlegung, dass ein bereits aktiver Prozess nicht "noch aktiver" wird

Verteilte Berechnungen

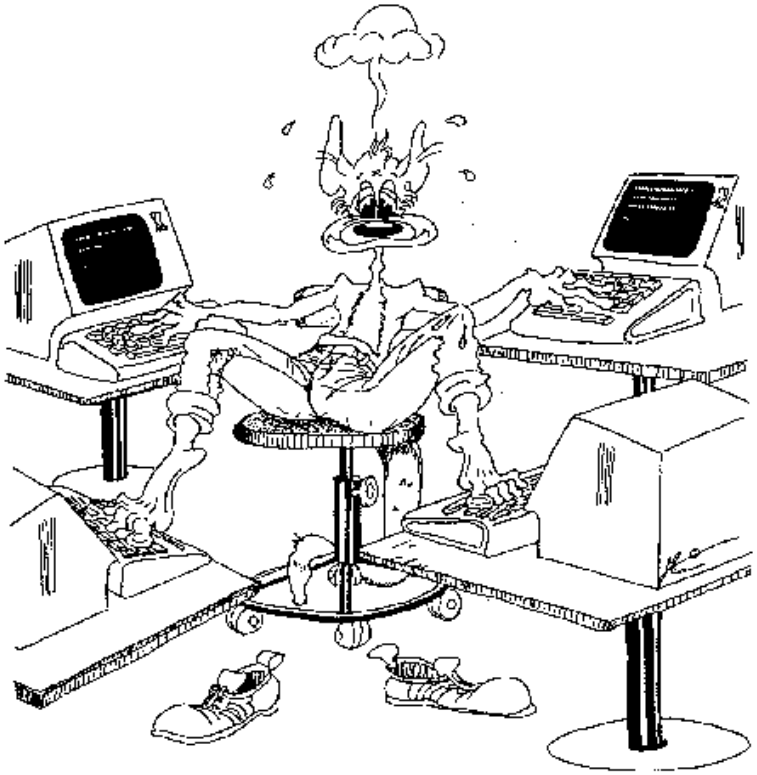


Bild: R. G. Herrtwich, G. Hommel

Verteilte Berechnungen - Vorüberlegungen

- Vert. Programm / Algorithmus:
  - mehrere "Programmtexte" für unterschiedliche Prozesstypen
  - "send", "receive" oder ähnliche Konstrukte für nachrichtenbasierte Kommunikation

- Verteilte Berechnung, "naive" Charakterisierung:
  - Ausführung eines vert. Programms / Algorithmus
  - viele u.U. gleichartige "Instanzen" von Prozessen
  - Kooperation (und Synchronisation) durch Kommunikation
  - mehrere gleichzeitige / parallele Kontrollflüsse ("threads")
  - mehrere Kontrollpunkte (Programmzählerstände)

- Bem.: Es gibt i.a. mehrere verschiedene Berechnungen zu einem verteilten Algorithmus! (Nichtdeterminismus)

- Anweisungen, Statements ==> atomare Aktionen
  - "atomar": Zwischenzustände von aussen nicht sichtbar (als ob diese keine zeitlichen Ausdehnungen hätten!)
  - auch grössere Einheiten zu atomaren Aktionen zusammenfassen
  - Abstraktion zu "Ereignissen"

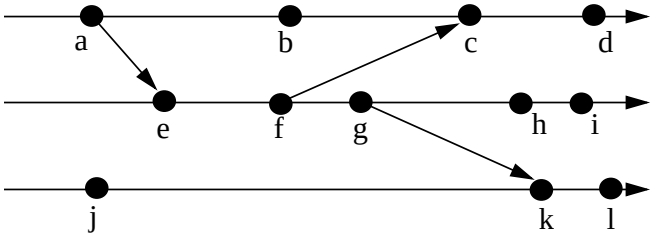
- *Visualisierung* einer vert. Berechnung durch *Zeitdiagramme*
  - Vorsicht: es gibt i.a. mehrere "äquivalente" Zeitdiagramme!

Gummiband-transformation

Verteilte Berechnungen

- Vorübung: Was ist eine sequentielle Berechnung (in einem geeigneten Modell)?

- Ziel: *Formale Definition*, ausgehend von *Zeitdiagrammen*
  - Zeitdiagramme sind bereits ein *Modell* (d.h. Abstraktion von der Realität): "punktförmige" Ereignisse, die bei miteinander kommunizierenden Prozessen stattfinden (Nachrichten als "Querpfeile")



  - interessant: von links nach rechts verlaufende "Kausalitätspfade"
  - auf graphische Darstellungsmöglichkeiten bei der Definition verzichten

---

- Menge von Ereignissen E
  - interne Ereignisse; korrespondierende send/receive-Ereignisse

- Direkte Kausalrelation ’<:’
  - s <: r für korrespondierende send/receive-Ereignisse s,r
  - a <: b wenn a direkter lokaler Vorgänger von b

- Eigentliche *Kausalrelation* ’<’ dann als transitive Hülle der direkten Kausalrelation

Die Kausalrelation

- Definiere eine Relation ‘<‘ auf der Menge E aller Ereignisse:

> “Kleinste” Relation auf E, so dass **x** < **y** wenn:
>
> 1) **x** und **y** auf dem gleichen Prozess stattfinden und **x** vor **y** kommt, *oder*
>
> 2) **x** ist ein Sendeereignis und **y** ist das korrespondierende Empfangsereignis, *oder*
>
> 3) $\exists$ **z** so dass: **x** < **z** $\wedge$ **z** < **y**

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

- Relation wird oft als “happened before” bezeichnet
  - eingeführt von Lamport (1978)
  - aber Vorsicht: damit ist nicht direkt eine "zeitliche" Aussage getroffen!

- Interpretationen von e < e’:
  - es gibt eine Kausalkette von e nach e’
  - e kann e’ beeinflussen
  - e’ hängt (potentiell) von e ab
  - e’ "weiss" / kennt e

Erster Definitionsversuch

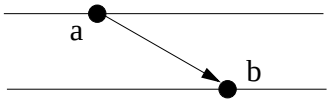
- Verteilte Berechnung = (E,<)

Menge von Ereignissen

Ordnungsrelation auf E

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

Beachte:



und

wären identisch!

==> *Prozesse* einführen

- Was sind Prozesse (formal / abstrakt)?
  - Partitioniere E in disjunkte Mengen $E_1,...E_n$
  - Interpretation: E_i = Ereignisse auf Prozess P_i

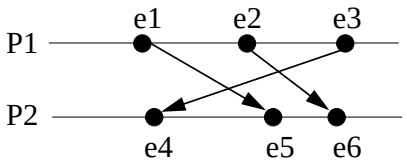
dort *linear* geordnet

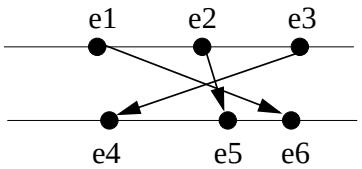
Zweiter Definitionsversuch

- Verteilte Berechnung = $(E_1,...E_n,<)$ mit:

  - alle E_i paarweise disjunkt
  - $<$ ist (irreflexive) Halbordnung auf $E_1 \cup ... \cup E_n$
  - $<$ ist lineare Ordnung auf jedem E_i

- Noch nicht befriedigend:





	e1	e2	e3	e4	e5	e6
e1		x	x	x	x	x
e2			x	x	x	x
e3				x	x	x
e4					x	x
e5						x
e6						

"Spalte" < "Zeile"

gleiche Kausalrelation
==> nicht unterscheidbar, obwohl wesentlich verschiedene Diagramme (also verschiedene Berechnungen)!

==> Nachrichten einführen
(eindeutige Zuordnung zusammengehöriger send-/receive-Ereignisse)

Dritter Definitionsversuch

- (*n-fach) verteilte Berechnung* (mit asynchroner Nachrichten-Kommunikation) = $(E_1,...E_n,\Gamma,<)$ mit:

  was wäre bei syn. Komm. anders?

  1) [*Ereignisse*] Alle E_i sind paarweise disjunkt

  2) [*Nachrichten*] Sei $E = E_1 \cup ... \cup E_n$
  Für $\Gamma \subseteq S \times R$ mit $S,R \subseteq E$ und $S \cap R = \varnothing$ gilt:
  - für jedes $s \in S$ gibt es *höchstens* ein $r \in R$ mit $(s,r) \in \Gamma$
  - für jedes $r \in R$ gibt es *genau* ein $s \in S$ mit $(s,r) \in \Gamma$

  3) $<$ ist eine lineare Ordnung auf jedem E_i

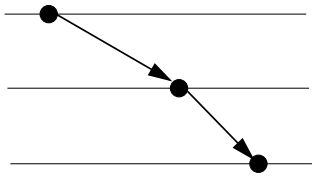
  4) $(s,r) \in \Gamma \Longrightarrow s < r$

  5) $<$ ist eine irreflexive Halbordnung auf E

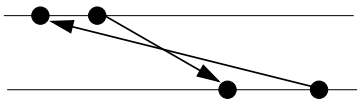
  6) $<$ ist die kleinste Relation, die 3) - 5) erfüllt
  (d.h.: es stehen keine weiteren als die dadurch festgelegten Ereignisse in $<$-Relation zueinander)

  [*Kausalrelation*]

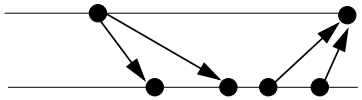
- "Gegenbeispiele":



nicht möglich wegen (2)



nicht möglich wegen (5)



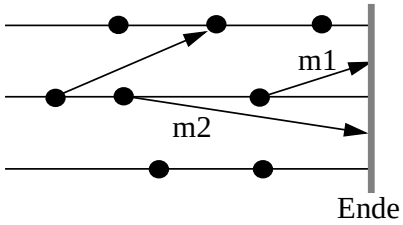
nicht möglich wegen (2)

Bemerkungen zur Definition

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

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

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



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

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

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

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

  Vorläufige Antwort:

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

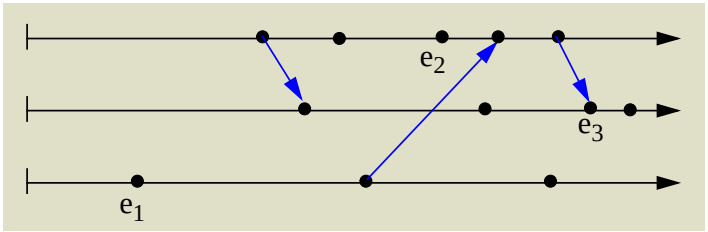
Zeitdiagramme

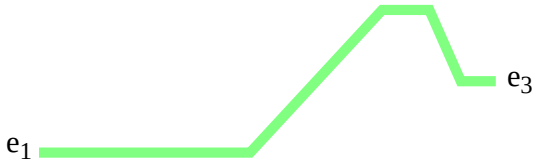
- Theorem [*Kausalkette*]:

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

  Nachrichtenpfeile + Teilstücke auf Prozessachsen von links nach rechts

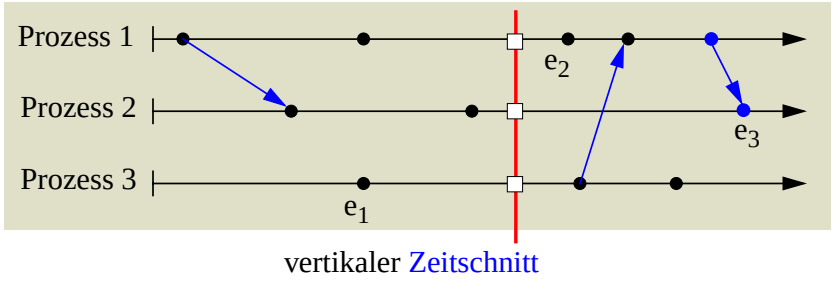
- Bew.: induktiv, Transitivität, "kann beeinflussen",...





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

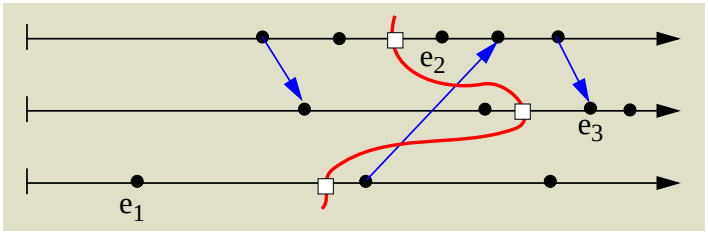
Gummibandtransformation



vertikaler Zeitschnitt

Diagramme sind daher äquivalent

Ein anderes Bild der *gleichen* Berechnung:



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

Nachrichtenpfeile sollen aber nie rückwärts laufen

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

Globale Zustände und Endzustände

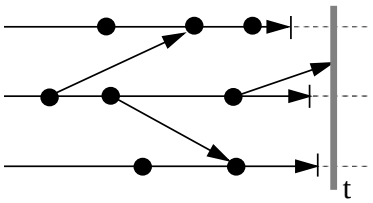
- Was ist ein *globaler Zustand* einer vert. Berechnung?
  - Charakteristikum verteilter Systeme: dieser ist auf die Prozesse verteilt und nicht unmittelbar ("gleichzeitig") zugreifbar!

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

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

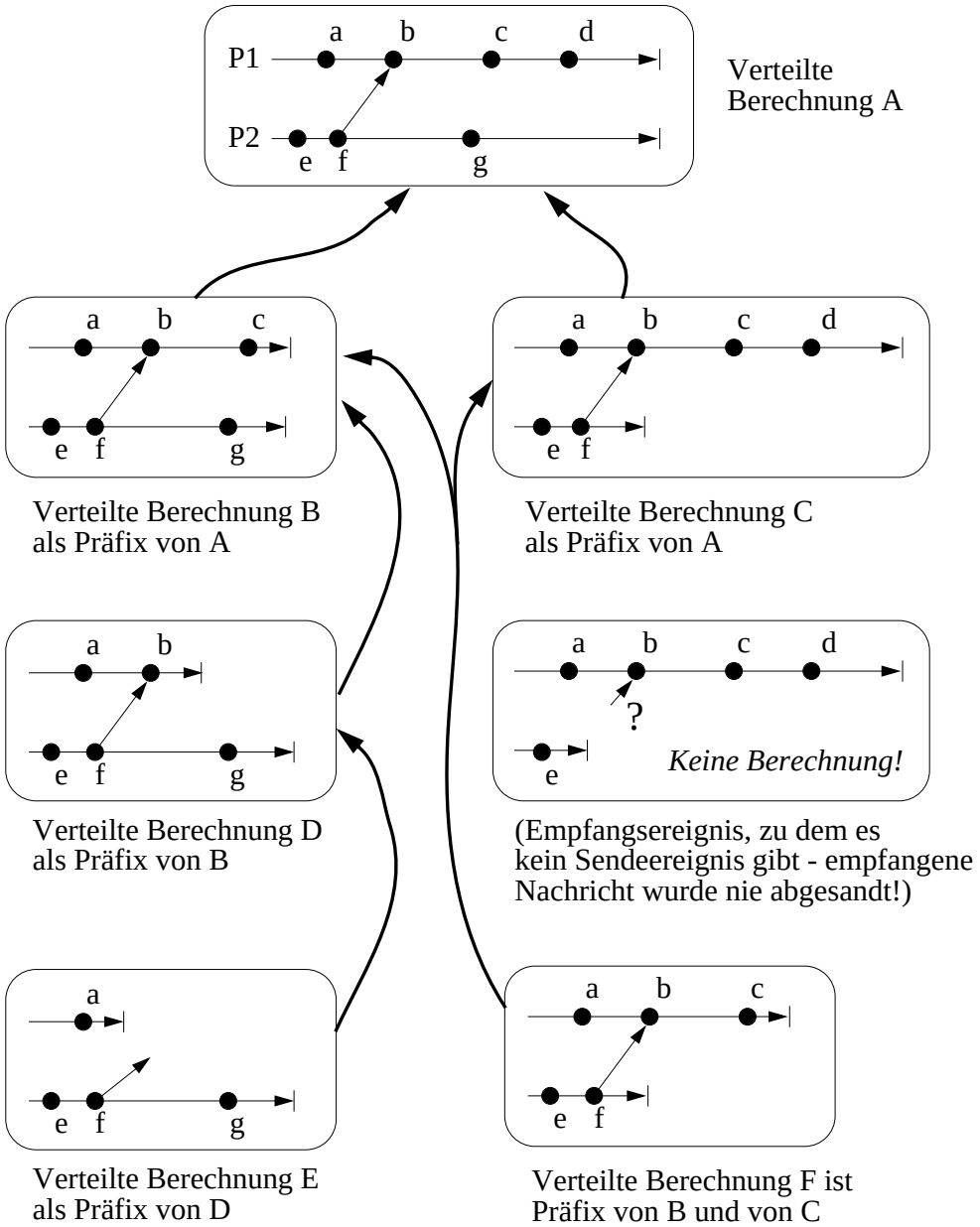
- *Globaler Zustandsraum*:
  - Kreuzprodukt lokaler Zustandsräume
  - Menge von Nachrichten ("in transit")

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



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

Präfixe von Berechnungen



Präfix-Berechnungen

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

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

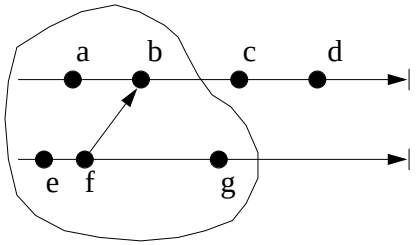
$$\forall\, x \in E',\, y \in E:\quad y < x \;\Longrightarrow\; y \in E'$$

==> lokale Berechnungen sind wirkliche Anfangsstücke

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

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

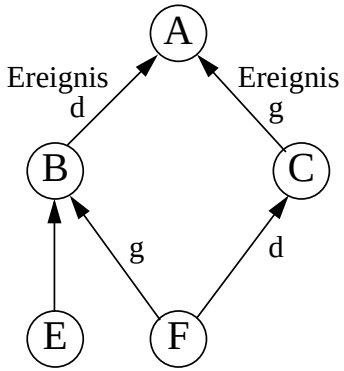
Beispiel:



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

(2): {b, g}

Präfix-Relation



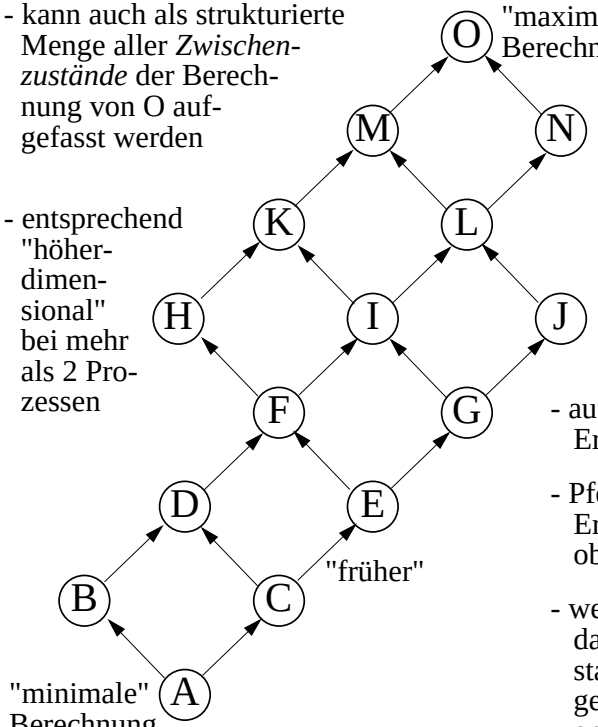
- Präfixdiagramm
  - kein (Wurzel)baum, aber
  - *gerichtet* und *zyklenfrei*
- Präfixrelation ist *transitiv*
- ==> Präfixrelation ist eine *Halbordnung*!

- Frage: Entstand "im Verlaufe der Berechnung" A aus B oder aus C?
  - d.h.: durchlief die Berechnung von A vorher den Endzustand von B oder den von C als Zwischenzustand?
  - äquivalent: geschah Ereignis d vor g oder g vor d?
  - beachte: *beides* ist unmöglich (wenn d und g nicht "gleichzeitig")
- Konsequenz:
  - direkter Vorgänger i.a. indefinit
  - verteilte Berechnung ist *keine Folge* von Zuständen, sondern - notgedrungen - eine geeignet definierte Halbordnung!
- Vgl. dies mit *sequentiellen* Berechnungen: Hier wird *jeder* Präfix einer Berechnung (genauer: dessen Endzustand) durchlaufen! (--> Def. als Folge möglich)

Der Präfix-Verband

- *Verband* von Mengen "geschehener" Ereignisse
  - zur Wiederholung: was ist ein Verband als mathematische Struktur?

- kann auch als strukturierte Menge aller *Zwischenzustände* der Berechnung von O aufgefasst werden
- entsprechend "höherdimensional" bei mehr als 2 Prozessen



an dieser Stelle würde ein "unmögliches" Diagramm stehen

- auf jeder Ebene kommt ein Ereignis hinzu
- Pfeil nach rechts oben: ein Ereignis von P1; nach links oben: ein Ereignis von P2
- welche Ereignismenge (und damit, welcher Zwischenzustand) einer Ebene *wirklich* geschehen ist, lässt sich nicht entscheiden --> keine vernünftige Frage!

- Ein Zwischenzustand hat also i.a. mehrere direkte Vorgänger- und Nachfolgezustände!
- Berechnung bewegt sich in diffuser Weise in diesem Zustandsraum von unten nach oben
  - von den Ecken her ausgefranstes n-dimensionales Gitter, mehr dazu später!