Verteilte Betriebssysteme

10. Kapitel
Globaler Zustand

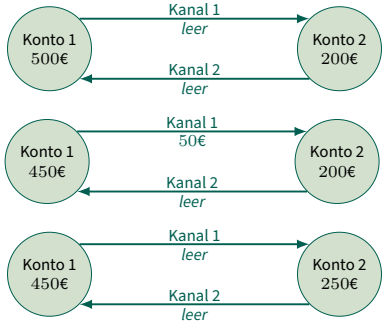
Matthias Werner
Professur Betriebssysteme

10.1 Motivation

- Mitunter ist der globale verteilte Zustand einer verteilten Anwendung (oder ein Teil davon) interessant
- Problem der **Beobachtbarkeit**
  - Lokale Zustände können nur lokal beobachtet werden
  - Kommunikation kann **nur indirekt** über Sende- und Empfangsereignisse beobachtet werden, nicht direkt
- Beispiel: verbundene Geldkonten

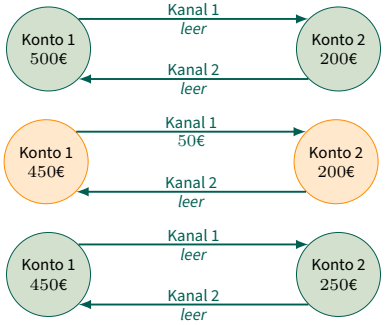
Beispiel: verbundene Geldkonten

- Bei mehreren, zusammengehörigen Geldkonten kann jederzeit Geld überwiesen werden oder eingehen
- Was ist der **Gesamtbetrag** auf den verbundenen Konten?



Beispiel: verbundene Geldkonten (Forts.)

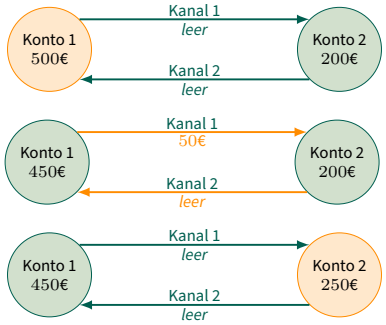
- 1. Fall:



- **Beobachten**: $450€ + 200€ = 650€$ ➡ **inkorrekt**
- **Erkenntnis**: Konzept des **Kanals** als zu untersuchendes Objekt

Beispiel: verbundene Geldkonten (Forts.)

- 2. Fall:



- **Beobachten:** $500€+50€+250€= 800€$ ➡ **inkorrekt**
- **Erkenntnis:** Konsistenz ist ein Problem (mal wieder 😛)

10.2 Konzepte

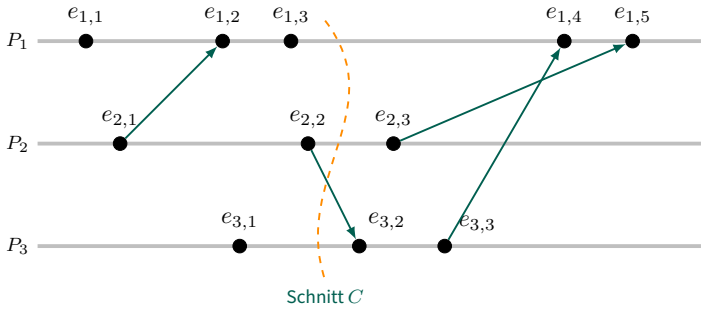
Definitionen: Historien und Schnitte

- Gegeben sei eine Menge von Prozessen $\mathbf{P} = \{P_1, P_2, ..., P_n\}$, wobei jeder Prozess auf einem anderen Knoten läuft
- Die Folge **lokaler Ereignisse** im Prozess P_i heißt **Historie** von P_i: $h_i =< e_{i,1}, e_{i,2}, \dots >$
  - **Beachte**: Auch Sende-/Empfangsereignisse sind lokale Ereignisse
- Betrachtung einer Historie bis zu einem bestimmten Zeitpunkt k ➡ **Präfix** der Historie $h_i(k) =< e_{i,1}, e_{i,2}, \dots, e_{i,k} >$
- $s_{i,k}$ bezeichnet den **Zustand** eines Prozesses P_i unmittelbar nach Ereignis $e_{i,k}$
- Der **initiale Zustand** heißt $s_{i,0}$

Definitionen: Historien und Schnitte (Forts.)

- Durch Vereinigungsbildung lokaler Historien entsteht eine **globale Historie** $H = h_1 \cup h_2 \cup \ldots \cup h_n$
- Ein **Schnitt** C einer globalen Historie ist die Vereinigung lokaler Präfixe:
  $C \stackrel{def}{=} h_1(c_1) \cup h_2(c_2) \cup \ldots \cup h_n(c_n)$
- Verbunden mit einem Schnitt C ist
  - ein **globale Zustand** des Systems $S(C) = (s_{1,c_1}, s_{2,c_2}, \dots, s_{n,c_n})$,
  - die (linkseitige) **Grenzlinie** des Schnitts $E(C) = (e_{1,C_1}, e_{2,C_2}, \dots, e_{n,C_n})$

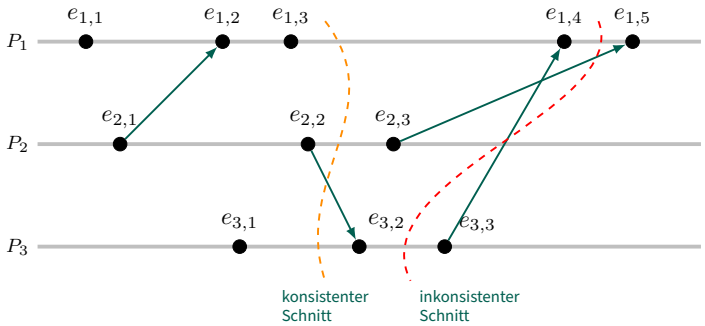
Schnitt



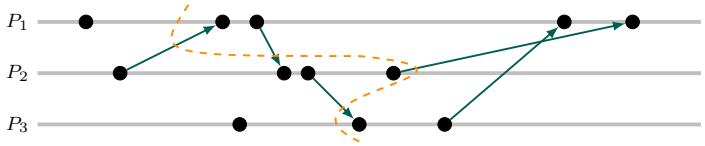
- Der Schnitt hat die (linksseitige) Grenzlinie $(e_{1,3}, e_{2,2}, e_{3,1})$

Konsistenter Schnitt

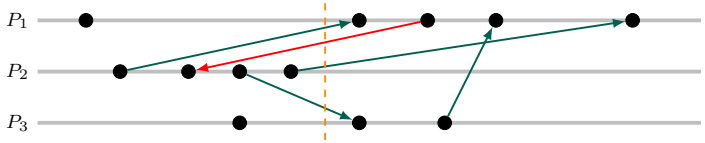
- Ein Schnitt partitioniert die globale Historie H in zwei Teile, den Schnitt C (Vergangenheit) und sein Komplement $H - C$ (Zukunft)
- Ein Schnitt C heißt **konsistent**, wenn $\forall e \in C, e' \prec e \Rightarrow e' \in C$
- ➡ Es darf keine kausalen Abhängigkeiten aus der „Zukunft“ geben



„Gummibandtransformation“



- Zur „Überprüfung“, ob ein Schnitt konsistent ist, kann man die lokalen Zeitachsen so dehnen oder stauchen, dass die Schnittlinie eine vertikale Gerade ergibt
- ➡ Alle Pfeile müssen in die Zukunft (nach rechts) zeigen
- Beispiel mit inkonsistentem Schnitt



10.3 Terminierung und Schnappschuss

- Mit Hilfe des konsistenten Schnittes kann das Problem des konsistenten globalen Zustands neu formuliert werden

Ein konsistenter Zustand ist die Summe der Teilzustände an einem konsistenten Schnitt.

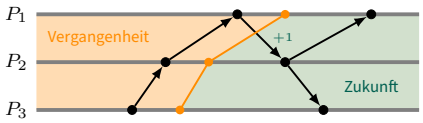
- Wir betrachten zunächst “kleines” Zustandsproblem: **Terminierung**
- **Terminierungsproblem**
  - Feststellen, ob bei einem konsistenten Schnitt alle Prozesse endgültig terminiert sind.
- Darauf aufbauend betrachten wir das allgemeinere **Schnappschussproblem**
- **Schnappschussproblem**
  - Feststellen der lokalen Zustände und gesendeten Nachrichten zu einem konsistenten Schnitt

Terminierung – Modell

- **Aktivitätszustände**: verteilte Komponenten (Knoten) in einem verteilten Algorithmus können bzgl. der Terminierung zwei (lokale) Zustände haben:
  - **Aktiv**:
    - Nachrichten können verschickt werden
    - Kann jederzeit „von allein“ in Zustand „terminiert“ wechseln
  - **Terminiert**:
    - Kann keine Nachrichten verschicken
    - Kann **nur durch empfangene Nachrichten** in den Zustand „aktiv“ wechseln
  - Kurz wird nur vom „**Zustand**“ gesprochen
- Wenn ein Terminierungsalgorithmus (Kontrollalgorithmus) bei mindestens einem Knoten einen aktiven Anwendungsalgorithmus(teil) entdeckt, ist dieser offensichtlich nicht terminiert
  ➡ wir “kondensieren” Aktivitätszeiten zu einem Punkt (**Atommodell**)
- ➡ Nur die noch nicht ausgelieferten Nachrichten sind interessant

Einfaches Zählverfahren

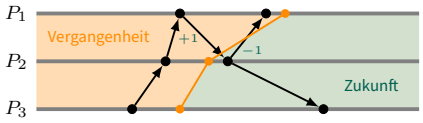
- Ein „Beobachter“ besucht nacheinander jeden Knoten und summiert getrennt die **gesendeten** und die **empfangenen** Basisnachrichten auf
- An ungleichen Summen lässt sich erkennen, dass Nachrichten abgeschickt, aber noch nicht angekommen sind!
- Daher: Sind beide Summen identisch, dann ist keine Nachricht mehr unterwegs und der Algorithmus ist terminiert, *oder*?



- Beobachter meldet:
  - 3 Nachrichten gesendet
  - 2 Nachrichten empfangen
- ➡ $3 \neq 2$, keine Terminierung

Einfaches Zählverfahren (Forts.)

- Leider funktioniert dieses Verfahren so **nicht**, da die Bedingung nur **notwendig**, aber **nicht hinreichend** ist
- Beispiel: Beobachter meldet 3 Nachrichten empfangen und 3 Nachrichten gesendet ➡ **Phantom-Terminierung**
- Dies liegt an Nachrichten die in der „Zukunft“ gesendet wurden aber in der „Vergangenheit“ empfangen werden
  - Diese gleichen die Summendifferenz aus



- Beobachter meldet:
  - 3 Nachrichten gesendet
  - 3 Nachrichten empfangen
- ➡ $3 = 3$, Terminierung (falsch!)

Lösungsideen

- Vermeiden von Nachrichten aus der Zukunft
  ➡ **Einfrieren** der Kommunikation im gefährlichen Bereich
- Nachträgliches Überprüfen mit zwei Wellen
  ➡ **Doppelzählverfahren**
- Differenzierteres Zählen
  ➡ **Vektorverfahren**

Einfrieren des Systems

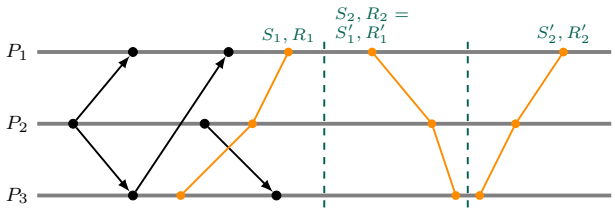
- Erste Welle **friert** das System ein
  - Kein Prozess nimmt mehr eine (Basis-)Nachricht an
  - Zwischenzeitlich ankommende Nachrichten werden gepuffert und als „noch unterwegs“ betrachtet
  - ➡ Im eingefrorenen Teil des Systems, werden keine Nachrichten mehr gesendet
- Zweite Welle **summiert** die gesendeten und die empfangenen Nachrichten auf
  - Sind beide Summen gleich, ist das System terminiert
- Dritte Welle **taut** das System wieder auf

Problem

Verfahren setzt die Nebenläufigkeit **stark** herab und greift in den Basisalgorithmus ein!

Doppelzählverfahren

- Ein Beobachter besucht **zweimal** hintereinander alle Knoten und bestimmt jeweils die Summen der empfangenen und gesendeten Nachrichten
- Sind **alle vier Summen** gleich, dann ist der Basisalgorithmus terminiert: $S_1 = R_1 = S_2 = R_2$
- Falls keine Terminierung: Benutze zweite Welle der vorherigen Runde als erste Welle der neuen Runde

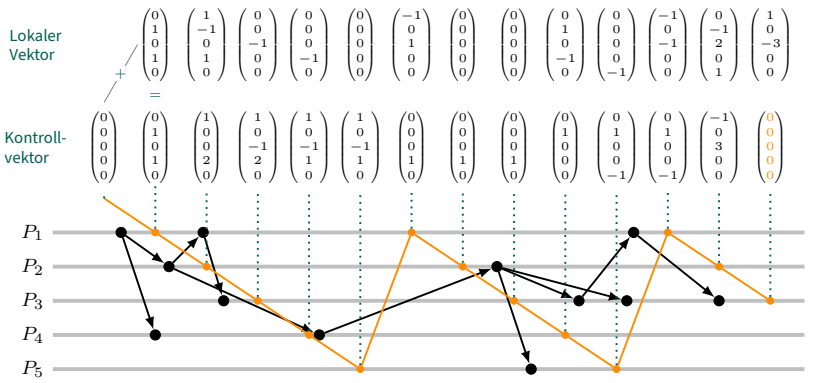


Doppelzählverfahrens: Eigenschaften

- Anzahl der Kontrollrunden ist a priori nicht durch die Anzahl der Basisnachrichten begrenzt
  - Es kann eine ganz langsame Basisnachricht geben; während ihrer Übertragung können **beliebig viele** Kontrollrunden gestartet werden
  - Variante beseitigt diese Eigenschaft: Prozess, der eine Basisnachricht erhält ohne eine neue auszusenden, **startet** eine neue Runde
- Das Doppelzählverfahren ist **ablaufinvariant**
  - Die lokalen Zustände der besuchten Prozesse werden nicht verändert
  - ➡ Mehrere konkurrierende Initiatoren können gleichzeitig die mögliche Terminierung testen

Vektorverfahren

- Jeder Prozess P_i führt einen lokalen Vektor $\vec{V}_i$ der Länge n
- $\vec{V}_i$ wird mit dem Nullvektor initialisiert
- Wenn P_i eine Basisnachricht an P_j verschickt, wird $\vec{V}_i[j]$ um 1 erhöht
- Wenn P_j eine Basisnachricht erhält, wird $\vec{V}_j[j]$ um 1 erniedrigt
- Ein Kontrollvektor $\vec{C}$ besucht reihum alle Prozesse
  - Bei einem Besuch, wird der lokale Vektor $\vec{V}$ zu $\vec{C}$ hinzuaddiert und $\vec{V}$ selbst wieder auf den Nullvektor gesetzt
  - Terminierung wird erkannt, sobald der Kontrollvektor zum **Nullvektor** wird
  - Keine Täuschung der Zähler möglich!



Vektorverfahren: Diskussion

- **Verbesserung**
  - Falls die Komponente des als nächstes zu besuchenden Knotens 0 ist, diesen zunächst auslassen
  - ➡ Potentiell nutzlose Besuche eines solchen Knotens werden vermieden
- **Vorteile**
  - Unabhängig von der Netztopologie
  - Meist geringe Anzahl von Kontrollnachrichten
  - Basisnachrichten bleiben unberührt (Zähler wird **nicht** mitgeschickt)
- **Nachteile**
  - Länge der Kontrollnachrichten (Vektor) $\mathcal{O}(n)$
  - Verfahren ist **nicht** ablaufinvariant

Konsistente Schnappschüsse

Konsistenter Schnappschuss

Ein konsistenter Schnappschuss entspricht der Summe der Zustände der lokalen Knoten **und** Kanäle zu einem **konsistenten Schnitt**.

- Kanäle können nicht direkt beobachtet werden
- **Aber**: Bei festgestellter Terminierung sind Kanäle leer
- ➡ Jede Summe lokaler Zustände (ohne Berücksichtigung der Kanäle) nach (bei) Festellung einer korrekten Terminierung, bilden eine konsistenten Schnappschuss
- Verteilte Terminierungsalgorithmen lassen sich aber **nicht direkt** zur Schnappschusserhebung einsetzen, da
  - auch Schnappschüsse **vor Terminierung** interessant sind
  - manche Algorithmen **nicht terminieren**
- **Aber:** Ideen lassen sich nutzen

Einfrieren des Systems

- Nutzung eines Wellenalgorithmus (z.B. Echo)
- Erste Welle friert das System ein
  - Kein Prozess sendet nach Empfang einer Ersten-Wellen-Nachricht mehr eine Nachricht
- Zweite Welle summiert lokalen Zustände auf
- Dritte Welle taut das System wieder auf
- **Problem**: Basisalgorithmus wird erheblich beeinträchtigt.

Türken einen Tag unter "Hausarrest" gestellt

ANKARA (dpa). Mehr als 50 Millionen Einwohner der Türkei standen gestern unter "Hausarrest". Weil die Wählerlisten für ein am 6. September anstehendes Referendum überprüft werden sollten, durfte die Bevölkerung den ganzen Tag die Wohnungen nicht verlassen. In der Volksbefragung soll über Fortdauer oder Aufhebung des seit 1980 bestehenden politischen Betätigungs-Verbots für die früheren Ministerpräsidenten Ecevit und Demirel entschieden werden. Während landesweit Zehntausende von Helfern unterwegs waren, um in dem wie ausgestorben wirkenden Land die Eintragungen in die Wählerlisten in den Wohnungen der Wähler zu kontrollieren, attackierten die beiden Oppositionspolitiker die umständliche Methode der Zählung.

Quelle: F. Mattern, Verteilte Basisalgorithmen

Chandy-Lamport Algorithmus (1985)

- **Annahmen**:
  - Gerichtete FIFO-Kommunikationskanäle
  - Zusammenhängender Kommunikationsgraph (Spezialfall: vollständig vernetzt)
  - Nachrichten des Basisalgorithmus werden nur einmal gesendet (Fehler sind ausgeschlossen)
  - Kanalübermittlung und lokale Zustandsnahme sind zeitbeschränkt (synchron)
- **Eigenschaften**:
  - Algorithmus kann von beliebigem Prozess ausgelöst werden
  - Nebenläufige Auslösung möglich
  - Der Algorithmus hat keinen Einfluss auf den Basisalgorithmus

CHANDY-LAMPORT Algorithmus (1985) (Forts.)

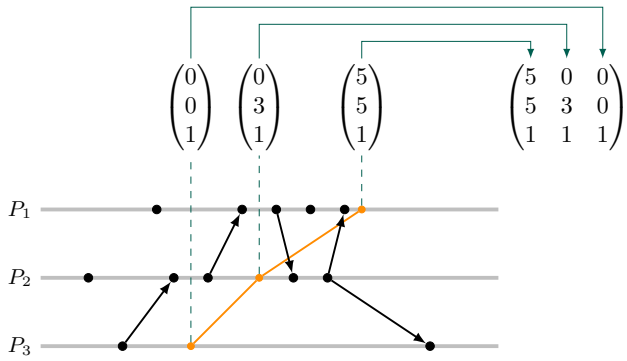
```
1  I: {NOT marked}:     // initiator
2    S_i := Z_L;        // take local state
3    marked := TRUE;
4    DO FOR ALL channel co IN set-of-out-channels
5      SEND <Marker> VIA co;
6    DONE
7
8  R_i: { message <Marker> is received via channel C}
9    IF NOT marked THEN
10     marked := TRUE;
11     S_i := Z_L;        // take local state
12     S_{i,C} := {};
13     DO FOR ALL channel co IN set-of-out-channels
14       SEND <Marker> VIA co;
15     DONE
16   ELSE
17     S_i := S_i + S_{i,C}
18   FI
19
20 R_i: { message Msg != <Marker> is received via channel C}
21   IF marked THEN
22     S_{i,C} := S_{i,C} + Msg;
23   FI;
```


CHANDY-LAMPORT Algorithmus (1985) (Forts.)

- Die Marker trennen auf den Kanälen die Nachrichten **vor** und **nach** dem Nehmen des lokalen Schnappschusses des Senders
- Das Verfahren terminiert, wenn alle Teilnehmer über alle Eingangskanäle einen Marker erhalten haben
  - Globaler Zustand: Summe aller lokalen (Knoten-)Zustände + aller (In-)Kanalzustände
  - Der Zustand kann eingesammelt werden, z.B. durch Senden der Knoten und Kanalzustände an den Initiator
- **Modifikation**: Lokale Schnappschüsse und Empfangs-Ereignisse werden **sofort** an den Initiator gesendet und dort der konsistente Zustand zusammengesetzt

Nutzung der Vektorzeit

- Konstruieren einer **Schnittmatrix**
- Kann erkannt werden, ob die Schnittmatrix für einen konsistenten Schnitt steht?



Konsistente Schnittmatrix

- Sei $\mathbf{S}$ eine Schnittmatrix
  - $\mathrm{dia}(\mathbf{S})$ sei der **Diagonalenvektor**

    $\mathrm{dia}(\mathbf{S}) = (s_{1,1}, s_{2,2}, \ldots, s_{n,n})^T$

    $\mathbf{S} = \begin{pmatrix}5&0&0\\5&3&0\\1&1&1\end{pmatrix} \Rightarrow \mathrm{dia}(\mathbf{S}) = \begin{pmatrix}5\\3\\1\end{pmatrix}$
  - $\sup(\mathbf{S})$ sei der **maximum row vector**

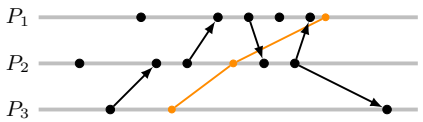
    $\sup(\mathbf{S}) = (\max(s_{1,1}, \ldots, s_{1,n}), \ldots, \max(s_{n,1}, \ldots, s_{n,n}))^T$

    $\mathbf{S} = \begin{pmatrix}5&0&0\\5&3&0\\1&1&1\end{pmatrix} \Rightarrow \sup(\mathbf{S}) = \begin{pmatrix}5\\5\\1\end{pmatrix}$

Konsistente Schnittmatrix (Forts.)

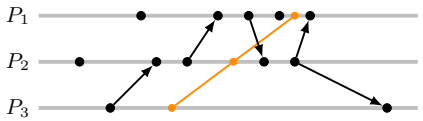
Konsistente Schnittmatrix

Ein Schnitt ist genau dann konsistent, wenn für seine Schnittmatrix $\mathbf{S}$ gilt: $\mathrm{dia}(\mathbf{S}) = \sup(\mathbf{S})$.



inkonsistenter Schnitt

$$\mathrm{dia}(\mathbf{S}) = \begin{pmatrix}5\\3\\1\end{pmatrix} \neq \sup(\mathbf{S}) = \begin{pmatrix}5\\5\\1\end{pmatrix}$$



konsistenter Schnitt

$$\mathrm{dia}(\mathbf{S}) = \begin{pmatrix}4\\3\\1\end{pmatrix} = \sup(\mathbf{S}) = \begin{pmatrix}4\\3\\1\end{pmatrix}$$

Vom Schnitt zum Schnappschuss

- Wie kommt man zum konsistenten Schnitt?
  - **Trivialer Ansatz**:
    Wiederholt Vektoren anfordern und abwarten, bis $\mathrm{dia}(\mathbf{S}) = \sup(\mathbf{S})$
- Was ist mit den Nachrichten?
  - **Trivialer Ansatz**:
    Lokale Sende- und Empfangsereignisse zählen. Wenn die Summen zu einem konsistenten Schnitt gleich sind, sind keine Nachrichten unterwegs
- **Nachteil**: Es können sehr viele[1] Versuche nötig sein, um zum Schnappschuss zu kommen

[1] Im Extremfall sogar unendlich viele

Matterns Lösung (1993)

- Initiator denkt sich einen Zeitstempel s **in der Zukunft** aus und schickt ihn an alle Knoten
- Alle Knoten merken ihn sich und schicken eine Bestätigung (falls lokale Zeit nicht kleiner s ➡ negative Bestätigung)
- Wenn der Initiator von allen Knoten positive Bestätigung erhalten hat, setzt er seine Vektorzeit auf s und sendet eine **Dummy-Nachricht**
- Jeder Knoten nimmt lokalen Schnappschuss, unmittelbar bevor er sein Vektorzeit zu einem Wert $\geq s$ setzt
- Der globale Schnappschuss ist die Summe der lokalen Schnappschüsse zuzüglich aller Nachrichten, die nach einem lokalen Schnappschuss mit einem Zeitstempel kleiner s ankommen

Diskussion

- Der Initiator darf bis zum Eintreffen aller Bestätigungen nicht aktiv werden
  ➡ Basis-Algorithmen auf dem Initiator werden beeinträchtigt
- Terminierung ist nicht offensichtlich
  ➡ Terminierungstest nötig (z.B. Nachrichten zählen)
- **Optimierungen**
  - Initiator als „virtueller“ Prozess ➡ kein Einfrieren
  - Wenn s hinreichend groß ist, kann auf Bestätigungen verzichtet werden

Kausaler Multicast

- Wenn kausaler Multicast (➡ Kapitel 4) verwendet wird, erhält man „automatisch“ konsistente Schnitte
- Müssen noch Kanäle beobachtet werden
- Ansatz nach ACHARYA/BADRINATH (1992)
  - Jeder Prozess P_i zählt Sende- und Empfangsnachrichten zu/von jedem Prozess in $\vec{S}_i$ und $\vec{R}_i$, d.h. z.B. eine von P_j empfangene Nachricht inkrementiert $r_{i,j}$
  - Nachrichten mit Sequenznummern zwischen $r_{j,i} + 1$ und $s_{i,j}$ sind im Kanal zwischen P_i und P_j unterwegs
- Ansatz nach ALAGAR/VENKATESAN (1994)
  - Ab lokalem Schnappschuss werden „alte“ Nachrichten protokolliert
  - Initiator sendet Terminate-Nachricht zum Beenden der Protokollierung, nachdem er eine Bestätigung für das Nehmen der lokalen Schnappschüsse erhalten hat

Literatur

[TaSt02] Tanenbaum , A. S. und M. van Steen: *Distributed Systems*. Prentice Hall, 2002 , Abschnitt 6.2

[CDK05] Coulouris , G. , J. Dollimore und T. Kindberg: *Distributed Systems: Concepts and Design*. Prentice Hall, 2005, Kapitel 18