3. Parallele und verteilte Simulation

Simulation realer Systeme ist eine komplexe Aufgabe ⇒

- Lange Laufzeiten
- Hoher Speicherbedarf
- Dezentrale Entwicklung einzelner Module

Eine Möglichkeit der Effizienzsteigerung

Einsatz paralleler und verteilter Systeme

naheliegender Weise auch in der Simulation

Literatur zu diesem Kapitel:

- H. Mehl. Methoden verteilter Simulation. Vieweg 1994
- R. Fujimoto. Parallel and distributed Simulation. Proc. Winter Simulation Conference 1999, S. 122-131.
- K. Pawlikowski. Distributed Stochastic Discrete-Event Simulation. In Simulation in Research and Development, L. Bobrowski and R. Bogacz (eds.), Polish Society of Computer Simulation and Polish Academy of Sciences, Warszawa, 2000, pp. 8-17

Wir unterschieden

- Parallele Simulation
  - Abarbeitung auf einem Parallelrechner mit tatsächlichem oder virtuellem gemeinsamen Speicher
  - Zentrales Management der Ereignisliste als globale Datenstruktur
- Verteilte Simulation
  - Abarbeitung auf einem verteilten System
  - Dezentrale Verwaltung der Ereignisliste, Prozesse kommunizieren über Nachrichtenaustausch

Unterscheidung in der Literatur nicht eindeutig

Wir betrachten hier primär verteilte Simulation im obigen Sinne

Ziele paralleler/verteilter Simulation

- Effizienzsteigerung (Laufzeit, Speicherplatz)
- Integration unterschiedlicher Module
- Fehlertoleranz

Historische Entwicklung paralleler/verteilter Simulation

- Stark ansteigendes Interesse Ende der 80er / Anfang der 90er Jahre,
- Vielzahl theoretischer Ansätze,
- nur wenige praktisch erfolgreiche Anwendungen,
- deshalb folgte eine Phase der Ernüchterung,

aber

- Die parallele Abarbeitung von Replikationen ist ein auch in der Praxis sehr erfolgreicher Ansatz (siehe 3.1 MRIP)
- Parallele/verteilte Simulation ist ein sehr interessantes Forschungsgebiet (siehe 3.2 SRIP)
  (viele generelle Probleme der parallelen Programmierung treten auf)
- In beschränkten Einsatzgebieten existieren durchaus positive Ergebnisse
- Infrastruktur für die Realisierung paralleler Simulationsprotokolle existiert

Wir betrachten im Folgenden verteilte Simulation ereignisdiskreter und stochastischer Systeme

Man kann grundsätzlich unterscheiden

- Parallelisierung durch mehrfache Abarbeitung eines Modells auf mehreren Prozessoren mit unterschiedlichen Sequenzen von Zufallszahlen (multiple replications in parallel (MRIP))
- Parallelisierung durch Verteilung eines Modells auf mehreren Prozessoren (single replication in parallel (SRIP))
  - Verteilung der Datenstruktur (räumlich)
  - Verteilung der Simulationszeit (zeitlich)

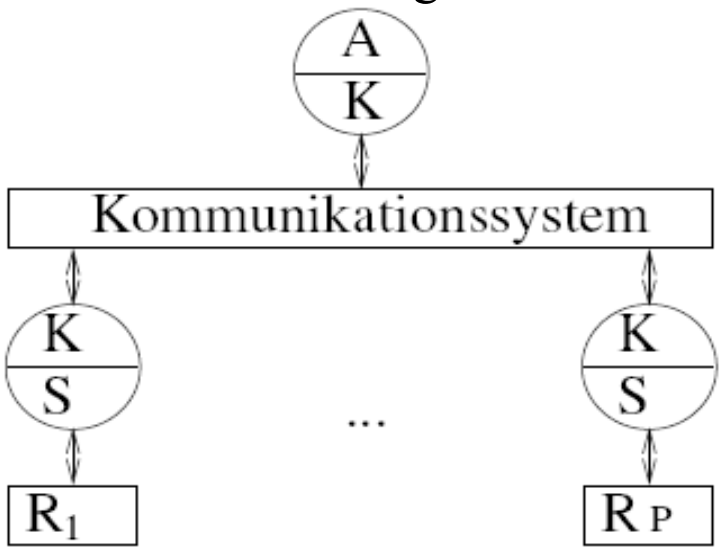
MRIP ist deutlich einfacher zu realisieren als SRIP!

Anforderungen an verteilte/parallele Simulation

- Effizienzsteigerung (genauere Resultate in vorgegebener Zeit, Analyse größerer Modelle)
- Einfache Realisieurng
- Möglichst keine bzw. wenige Änderungen am Modell
- Möglichst automatische Parallelisierung
- Teilmodelle unterschiedlicher Simulatoren nach Festlegung der Schnittstellen koppelbar, wobei jedes Teilmodell auf einem separaten Rechner läuft

3.1 MRIP

Struktur der Realisierung



K Kommunikationsschnittstelle
S Simulatorkern R Replikation
A Analysator

- Auf jedem Prozessor läuft ein Simulator
- Simulatoren senden Zwischenergebnisse für alle Resultatmaße zu Kontrollzeitpunkten an den Analysator (Kontrollzeitpunkts sind lokal definiert, z.B. all t Modellzeiteinheiten, alle k Ereignisse, …)
- Komunikation zwischen Analysator und Simulatoren über Kommunikationsschnittstelle (Sockets, MPI, ….)

Notwendige Eigenschaften der Simulatoren

- Erzeugung von P unabhängigen ZZ-Strömen
- Ausgabe der Zwischenergebnisse über die Kommunikationsschnittstelle

Welche Leistungssteigerungen können wir erwarten?

Einige Notationen

- P Prozessoren (homogen)
- D Beobachtungen zwischen Kontrollpunkten
- N_{min} Beobachtungen werden insgesamt benötigt
- N_P Beobachtungen werden von jedem Prozessor generiert
- f sei der sequentielle Anteil, von jedem Prozessor separat zu leisten
  z.B. zur Simulation der transienten Phase bei einer stationären Simulation nehmen wir an, dass $f \cdot N_{min}$ Beobachtungen sequentiell erzeugt werden

Die P Prozessoren müssen also $(1-f) \cdot N_{min}$ Beobachtungen erzeugen, so dass
$P \cdot N_P = (1-f) \cdot N_{min}$ gilt

Durch Anpassung des Gesetzes von Amdahl erhalten wir folgende obere Schranke

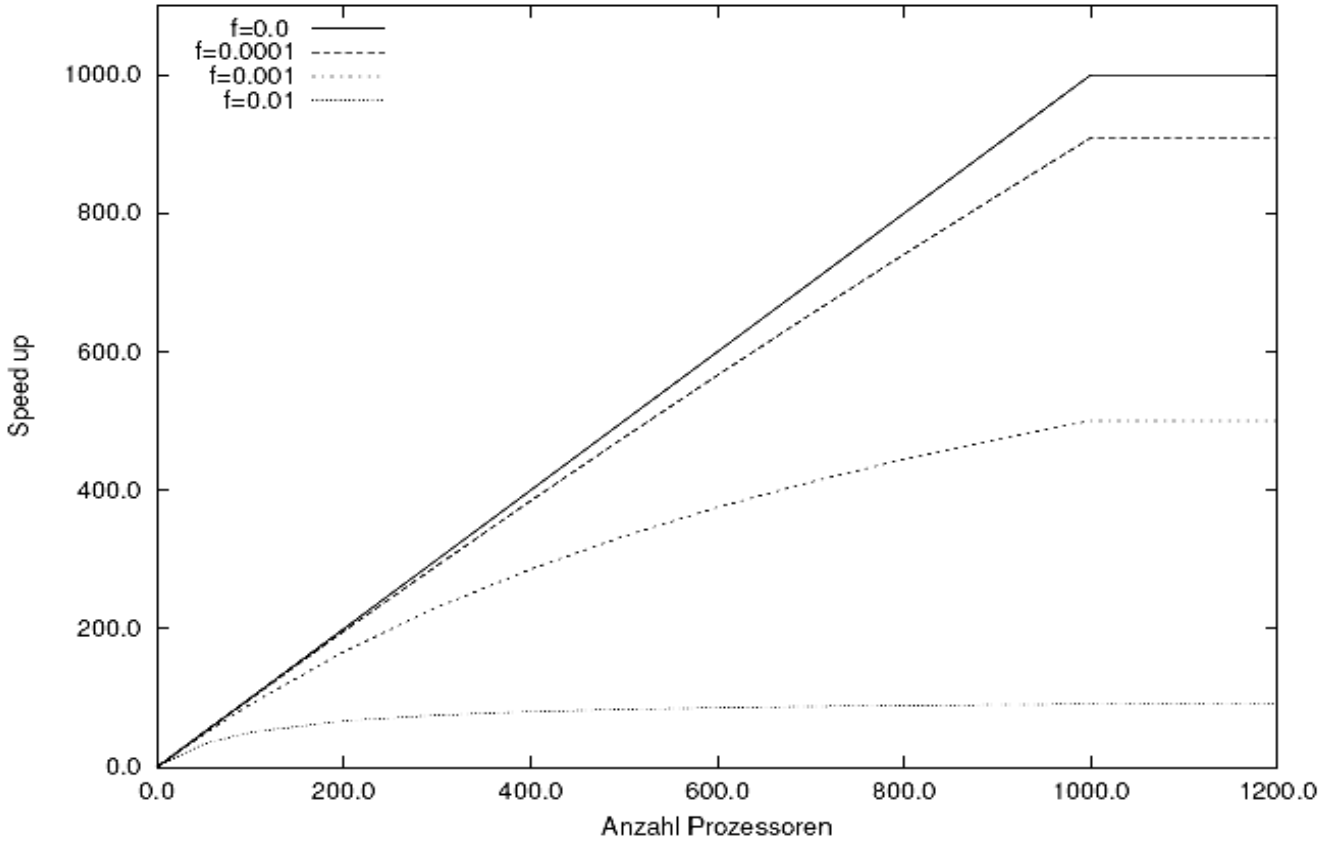
$$S = \begin{cases} \dfrac{1}{f + (1-f)\cdot P} & \text{falls } P \le \dfrac{(1-f)\cdot N_{\min}}{D} \\ \dfrac{N_{\min}}{f \cdot N_{\min} + D} & \text{falls } P \ge \dfrac{(1-f)\cdot N_{\min}}{D} \end{cases}$$

Der maximale Speedup beträgt $S_{\max} = \dfrac{N_{\min}}{f \cdot N_{\min} + D}$

bei Verwendung von $P_{\max} = \dfrac{(1-f)\cdot N_{\min}}{D}$ Prozessoren

Bei realen Simulationen sind N_{min}, f und D von der betrachteten Trajektorie abhängig, so dass Ergebnisse auf Basis der Erwartungswerte (oder anderer Kenngrößen) berechnet werden müssen

Verlauf des Speedups für $E(D) = 1$ und $E(N_{min}) = 1000$



Schon geringe sequentielle Anteile führen zu einer signifikanten Reduktion des Speedups

Praktische Anwendung des Ansatzes

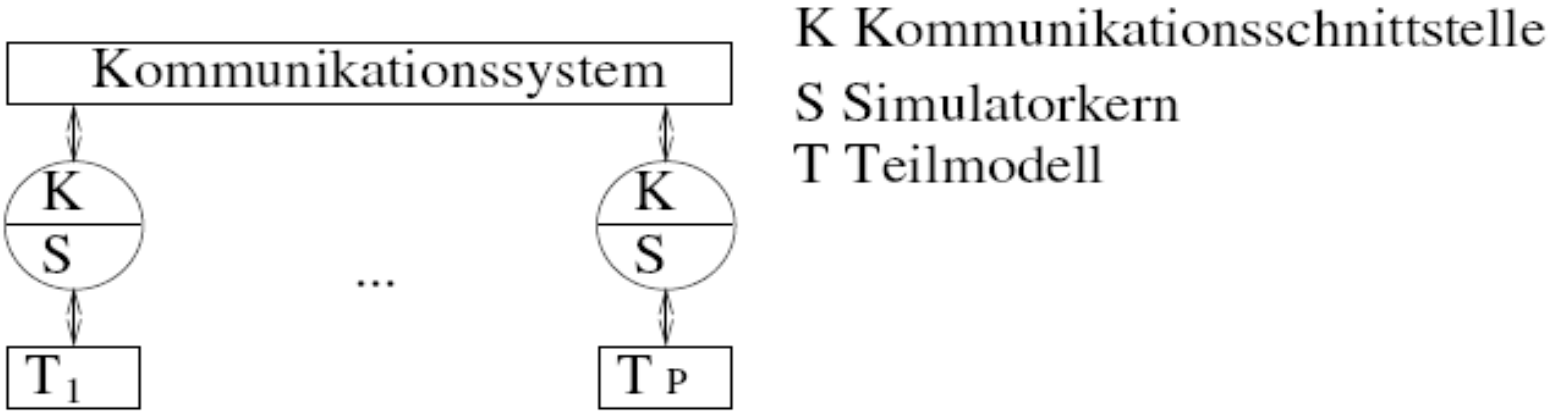
- Terminierende Simulation
  - Analyse durch unabhängige Replikationen
  - Simulation ohne sequentiellen Anteil, d.h. f = 0
  - Replikationen unabhängig so lange ZZ-Ströme nicht überlappen
- Stationäre Simulation
  - Falls eine transiente Phase existiert (was in der Regel der Fall ist), muss diese von jedem Prozessor durchlaufen werden
  - f > 0 gilt damit in jedem Fall
  - Der tatsächliche Wert von f hängt vom Verhältnis der Länge der transienten Phase zur Länge des beobachteten Intervalls ab bzw. muss mit der notwendigen Batchgröße einer sequentiellen Simulation verglichen werden
  - Unabhängigkeit der Prozesse bei Vewendung nicht überlappender ZZ-Ströme

Einige weitere Beobachtungen

- Gemessene Werte für den Speedup ähneln den berechneten Werten
- MRIP erfordert nur Kommunikation zwischen Simulatoren und Analysator und keinen zusätzlichen Synchronisationsaufwand
- Kommunikationsaufwand kann durch Wahl von D beeinflusst werden (D ist in Grenzen frei wählbar)
- Automatisierung des Ansatzes ist relativ einfach
- Prozessoren bzw. Rechner müssen nicht identisch sein
- Spezielle statistische Verfahren zur Erkennung der transienten Phase und zur Konfidenzintervallberechnung auf Basis von MRIP existieren

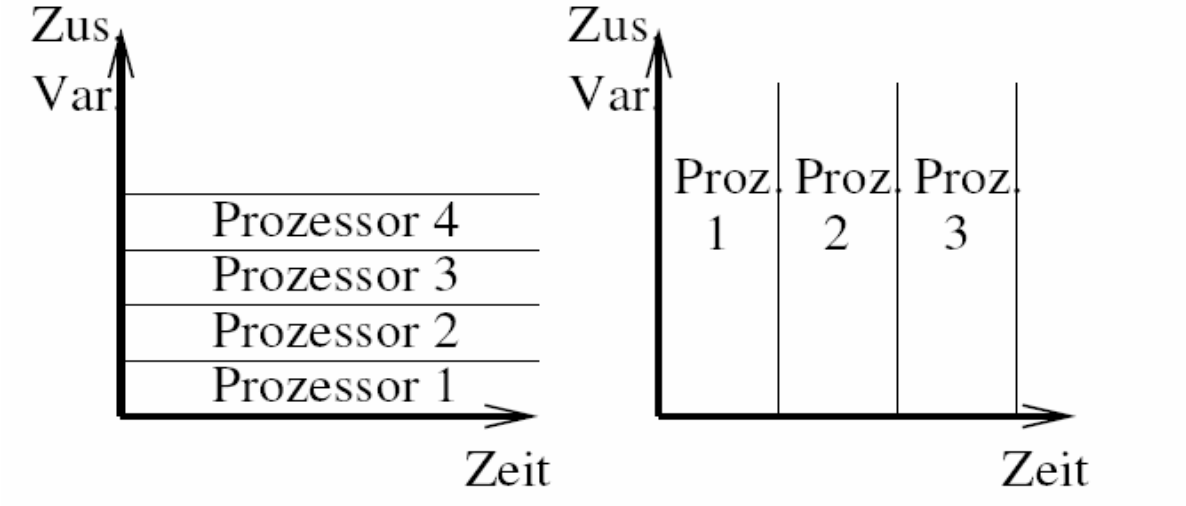
3.2 SRIP

Allgemeinere Form der Verteilung durch Aufteilung des Simulationsprozesses in mehrere logische Prozess (LPs), die jeweils einen Teil des Modells beschreiben



- Prozessoren kommunizieren in der Regel zur Synchronisation und zum Ergebnisaustausch
- Resultate werden in den Prozessoren während der Simulation oder durch einen separaten Analyseprozess berechnet
- Aufteilung erfordert die Festlegung eines Teilmodellbegriffs und dessen Umsetzung in das Simulationsparadigma

Mögliche Aufteilungen der Simulationsprozesse



Raum-/Zeitdiagramme

- Raum ist definiert durch die Zustandvariablen
  - Datenstruktur zur Zustandsbeschreibung wird auf verschiedene Prozessoren verteilt
  - In Abhängigkeit vom lokalen Zustand und der Kenntnis über den Zustand auf anderen Prozessoren werden Ereignisse generiert
- Zeit ist gegeben durch die Modellzeit
  - Jeder Prozessor behandelt ein Teilintervall
- Kombination beider Ansätze möglich

3.2.1 Verteilung bzgl. der Zeit

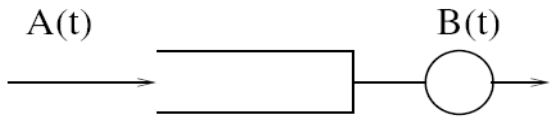
- Unterteilung der Zeitachse in Intervalle $I_i = [T_i, T_{i+1})$
- Prozessor i simuliert das System in I_i
- Theoretisch beliebige Parallelisierbarkeit
- Voraussetzung: Zustand am Ende von I_i entspricht Zustand am Anfang von I_{i+1}
- Um überhaupt Parallelität zu erreichen, muss der Zustand am Ende des Intervalls von Simulation des Intervalls bekannt sein

$\Rightarrow$ Konkrete Anwendungen nur in speziellen Fällen

Beispiele

- Endzustand hängt nicht/kaum vom Anfangszustand ab
- Falscher Anfangszustand ist reparierbar
- Schätzung des Anfangszustand und erneuet Simulation bei falscher Schätzung
- T_i dynamisch wählen, so dass am Ende des Intervalls jeweils ein spezieller Zustand erreicht wird (z.B. Regenerationszustand, dann identisch zu MRIP)
- (Verallgemeinerte) Rekurrenzgleichungen

Rekurrenzgleichungen am Beispiel GI/GI/1:



Notationen:

- r_i Zwischenankunftszeit des i-ten Auftrags (unabhängig identisch verteilt)
- s_i Bedienzeit des i-ten Auftrags (unabhängig identisch verteilt)
- Wartezeitd es i-ten Auftrags $d_i = (d_{i-1} + s_{i-1} – r_i)^+$ wobei $(x)^+=\max(0, x)$

$\Rightarrow$ r_i und s_i können aus unabhängigen ZZ-Strömen generiert werden

$\Rightarrow$ Bei Kenntnis von r_i und s_i können alle Wartezeiten berechnet werden

$\Rightarrow$ Aus den Wartezeiten und Ankunftszeiten sind die Populationen im System berechenbar

Mathematisches Modell zur weiteren Berechnung: max/+ Semiring
Semiring $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$ mit

- $\mathbb{K}$ einer Menge
- $\oplus$, $\otimes$ sind assoziativ, $\oplus$ ist kommutativ, $\oplus$, $\otimes$ sind rechts- und links-distributiv
- $\mathbb{0}$ und $\mathbb{1}$ sind die Identitäten von $\oplus$ und $\otimes$, $\mathbb{0} \neq \mathbb{1}$ und für alle $k \in \mathbb{K}$ gilt
  $k \otimes \mathbb{0} = \mathbb{0} \otimes k = \mathbb{0}$

Semiring für unsere Berechnungen:

- $\mathbb{K} = \mathbb{R} \cup \{-\infty\}$
- max := Addition mit Identität $-\infty$ (Symbol $\oplus$)
- + := Multiplikation mit Identität 0 (Symbol $\otimes$)

Es gilt unter anderem:

1. $a, b \in \mathbb{K} \Rightarrow a \oplus b, a \otimes b \in \mathbb{K}$ (Abgeschlossenheit)
2. $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ und $(a \otimes b) \otimes c = a \otimes (b \otimes c)$ (Assoziativität)
3. $a \oplus b = b \oplus a$ (Kommutativität der Addition)
4. $a \otimes -\infty = -\infty \otimes a = -\infty$
   (Absorbierendes neutrales Element der Addition bzgl. der Multiplikation)
5. $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$ (Distributivität)

Operationen können auf Matrizen erweitert werden

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \otimes \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} (A_{11} \otimes B_{11}) \oplus (A_{12} \otimes B_{21}) & (A_{11} \otimes B_{12}) \oplus (A_{12} \otimes B_{22}) \\ (A_{21} \otimes B_{11}) \oplus (A_{22} \otimes B_{21}) & (A_{21} \otimes B_{12}) \oplus (A_{22} \otimes B_{22}) \end{pmatrix}$$

Vorgehen bei der Berechnung der Wartezeiten in Matrixdarstellung

$D_i = M_i \otimes D_{i-1} = M_i \otimes M_{i-1} \otimes \ldots \otimes M_2 \otimes D_1$

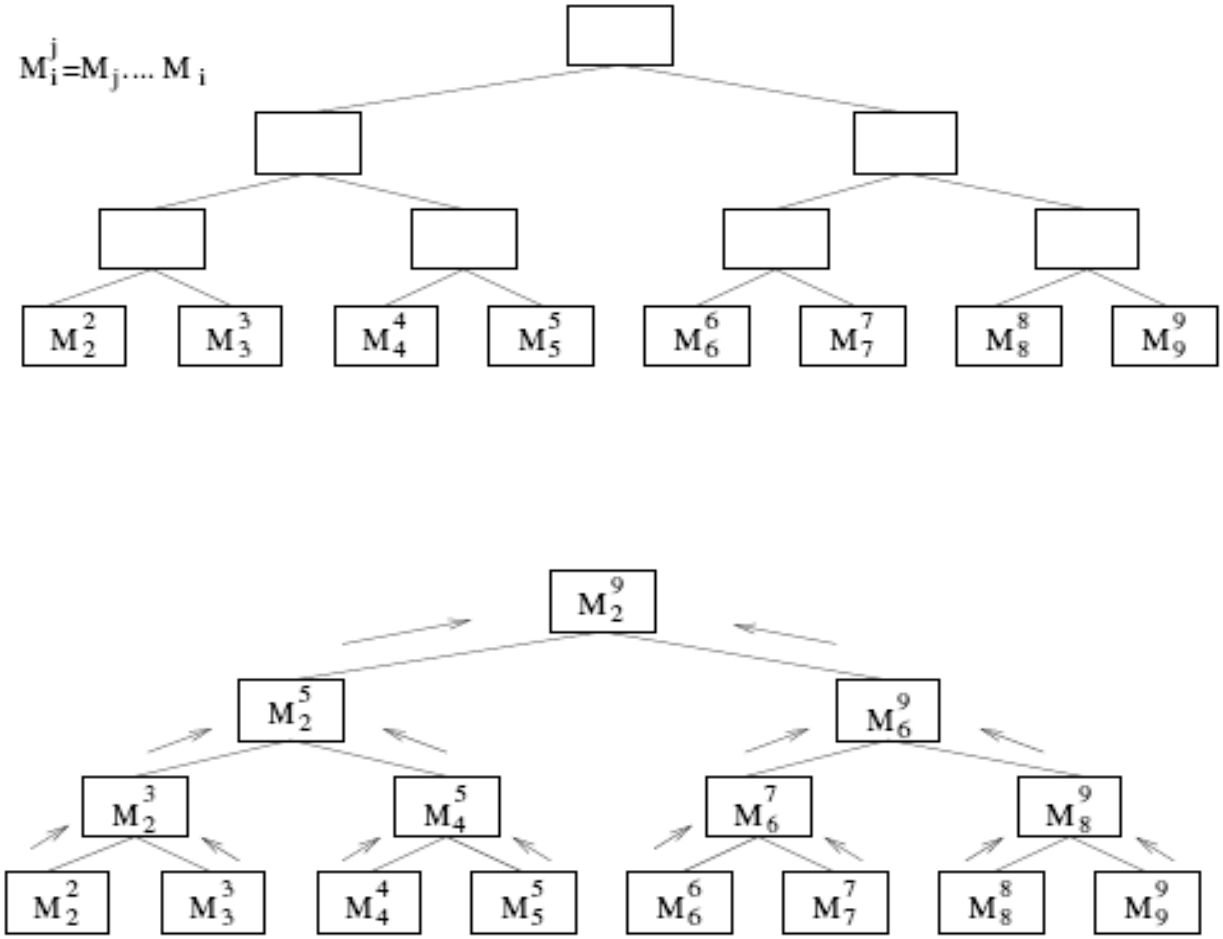
mit $$D_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, D_i = \begin{pmatrix} d_i \\ 0 \end{pmatrix} \text{und} M_i = \begin{pmatrix} s_{i-1} - r_i & 0 \\ -\infty & 0 \end{pmatrix}$$

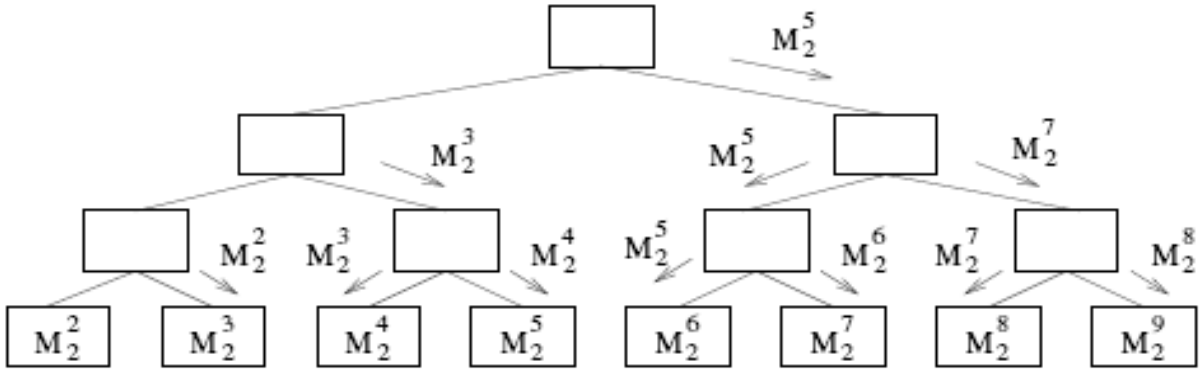
Also $M_2^i = M_i \otimes \ldots \otimes M_2 \Rightarrow D_i = M_2^i \otimes D_1$, so dass gilt

$$\begin{pmatrix} d_i \\ 0 \end{pmatrix} = \begin{pmatrix} \max(s_{i-1} - r_i + d_{i-1}, 0) \\ \max(-\infty + d_{i-1}, 0) \end{pmatrix} = \begin{pmatrix} s_{i-1} - r_i & 0 \\ -\infty & 0 \end{pmatrix} \otimes \begin{pmatrix} d_{i-1} \\ 0 \end{pmatrix}$$

Simulation kann mit parallelem Präfixalgorithmus durchgeführt werden
Ablauf:

- Parallele Generierung der ZZs
- Schrittweise Berechnung der Unterprodukt $M_i \otimes \ldots \otimes M_j$ $(i > j)$


$M_i^j = M_j \ldots M_i$
M_2^2
M_3^3
M_4^4
M_5^5
M_6^6
M_7^7
M_8^8
M_9^9
M_2^9
M_2^5
M_6^9
M_2^3
M_4^5
M_6^7
M_8^9
M_2^2
M_3^3
M_4^4
M_5^5
M_6^6
M_7^7
M_8^8
M_9^9



Blattknoten berechnen abschließend in einem Schritt $D_i = M_2^i \otimes D_1$

$\Rightarrow$ Simulationslauf der Länge n mit Speedup O(p) und Effizienz O(1) bei Verwendung von p = O(log n) Prozessoren

Verfahren erweiterbar auf

- Warteschlangennetze mit „feedforward“-Struktur (d.h. keine Zyklen)
- Spezielle zeitbehaftetet Petri-Netze
- Aber nicht für größere Modellklassen (jedenfalls nicht in max/+)

3.2.1 Verteilung bzgl. des Raumes

- Unterteilung des Simulationsmodells in disjunkte Teile
- Jeder LP simuliert ein Teilmodell
- Lokale Ereignisse im Teilmodell werden lokal von einem LP simuliert
- Globale Ereignisse zwischen Teilmodellen erfordern Kommunikation zwischen LPs

Partitionierung des Modells

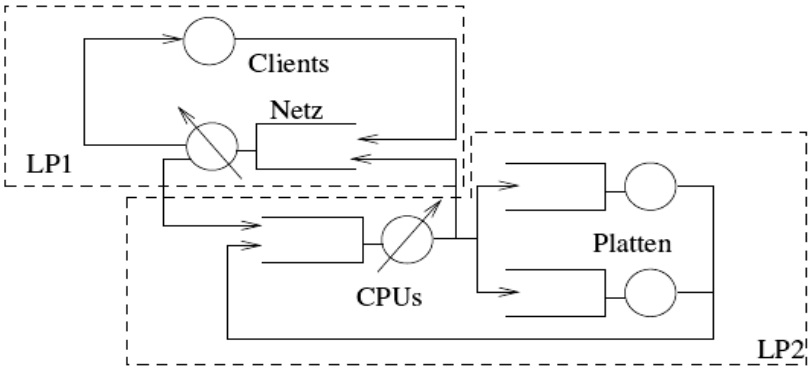
Unterteilung der statischen Struktur in disjunkte Teile

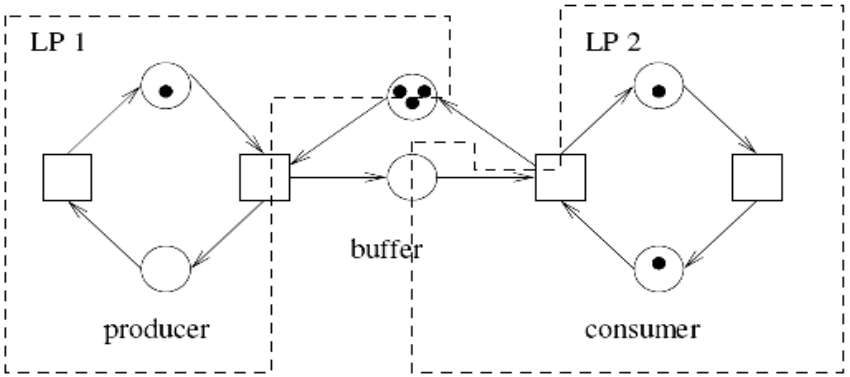
Unterteilung auf Basis der physikalischen System-/Modellstruktur

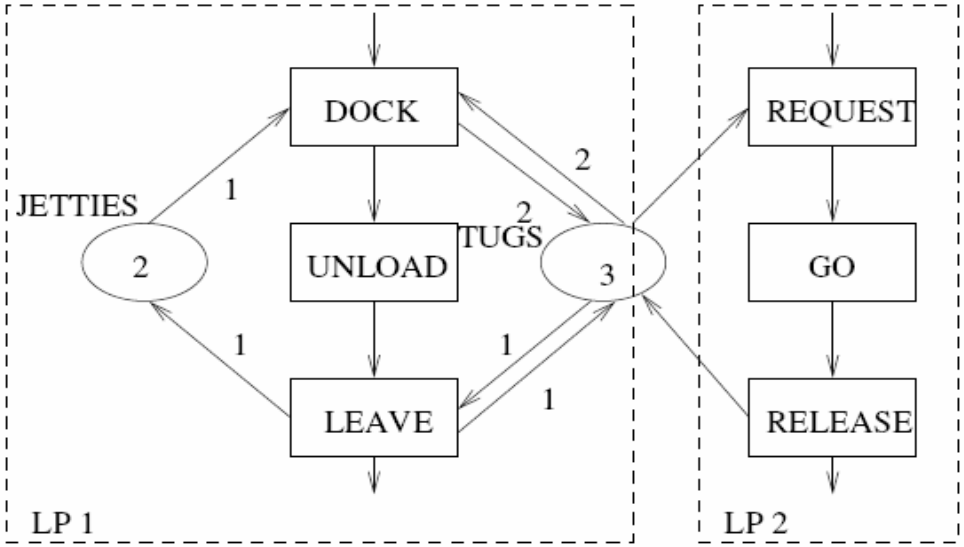
- Menge von Stationen in einem Warteschlangennetz
- Menge von Stellen und Transitionen in einem Petri-Netz
- Menge von Prozessen und Ressourcen in einem prozessorietierten Simulator

Partitionierung erfolgt während der Modellspezifikation, unabhängig von der zur Simulation verwendeten Rechnerarchitektur!

Beispiele


Clients
Netz
LP1
Platten
CPUs
LP2


LP 1
LP 2
buffer
producer
consumer


DOCK
REQUEST
JETTIES
2
1
2
2
TUGS
3
UNLOAD
GO
1
1
1
LEAVE
RELEASE
LP 1
LP 2

- Partitionierung ist modell(typ)spezifisch
- Schnittestellen sind modell(typ)spezifisch
- In manchen Fällen (nicht bei Warteschlangennetzen oder Petri-Netzen) kann sich die Zahl der LPs zur Laufzeit ändern
- In allen Fällen haben lokale Ereignisse u.U. globale Effekte (d.h. ändern Zustandsvariablen, die zu anderen LPs gehören) und erfordern damit Kommunikation
- Verteilung der LPs auf Prozessoren ist ein zusätzliches Scheduling-Problem

Ziel der Partitionierung und Aufteilung der LPs auf Prozessoren:
Möglichst geringe Kommunikation

- Ereignisse sollten möglichst nur LPs auf einem Prozessor beeinflussen
- Partitionierung des Modells so, dass „fast alle“ Ereignisse LP-lokal
- Scheduling so, dass „fast alle“ Ereignisse lokal auf einem Prozessor

Problem der Kommunikation zwischen LPs auf unterschiedlichen Prozessoren

- Kommunkationsoverhead
- Sicherstellung zeitlicher Konsistenz

Man unterscheidet synchrones und asynchrones Vorgehen:

Synchrones Vorgehen

- Existenz einer (virtuellen) globalen Uhr mit Zeittakt Δ
- Synchronisation der Prozesse alle Δ Zeiteinheiten
- LP simuliert die Ereignisse im Intervall [i·Δ, (i+1)·Δ) lokal
- Am Intervallende Synchronisation und Nachrichtenaustausch zwischen LPs
- Overhead durch zusätzliche Kommunikation alle Δ Zeiteinheiten

Bewertung des Vorgehens

+ Einfach realiserbares Konzept
– Oft ineffizient
– Anwendung nur auf spezielle Modelltypen
  - Δ existiert in der Regel nicht und muss künstlich eingeführt werden
    - großes Δ verfälscht das Verhalten
    - kleines Δ sorgt für zusätzlichen Overhead

Asynchrones Vorgehen

- LPs simulieren asynchron ihre Teilmodelle
- Es existieren lokale Uhren, aber keine globale Uhr
- Interaktion zwischen LPs erfolgt über Nachrichten mit lokalem (!) Zeitstempel
- Empfangene Nachrichten werden im Simulationslauf des Emfängers berücksichtigt

Bewertung des Vorgehens

\+ Theoretisch breit anwendbar

– Problem der Kausalitätsverletzung
(LP_i empfängt Nachricht von LP_j mit Zeitstempel t_j kleiner als lokale Zeit lt_i)

– Effizienz hängt stark von der Systemstruktur ab

Im wesentlichen werden heute asynchrone Verfahren verwendet

Kausalitätsverletzungen werden durch Protokolle

- vermieden (konservative Protokolle)
- behoben (optimistische Protokolle)

Konservative Protokolle

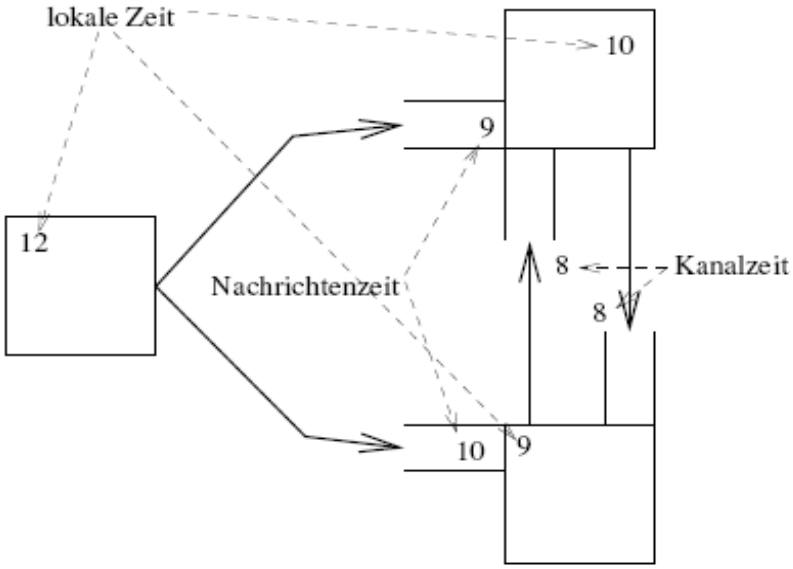
Voraussetzung: Zuverlässige FIFO Kanäle zwischen kommunizierenden LPs (jeder Kanal ist Ausgangs- und Eingangskanal für genau einen LP)

- LP_i hat lokale Uhr lt_i
- Ereignisse erhalten lokale Zeitstempel und werden wie üblich in der Ereignisliste gespeichert
- Alle Nachrichten erhalten (lokale) Zeitstempel $t_i = lt_i$ zum Zeitpunkt ihrer (lokalen) Generierung
- kt_i := Kanalzeit Kanal i
  (Zeitstempel der letzten empfangenen Nachricht auf diesem Kanal)
- LPi führt lokales Ereignis zum Zeitpunkt lt_i aus, falls $kt_k \geq lt_i$ für alle Eingangskanäle k
- Blockierung von LP_i bei leerem Eingangskanal oder einem Eingangskanal k mit $kt_k < lt_i$

Probleme:

- Pufferüberläufe in den Kanälen
- Deadlocks

Beispiel für einen Deadlock



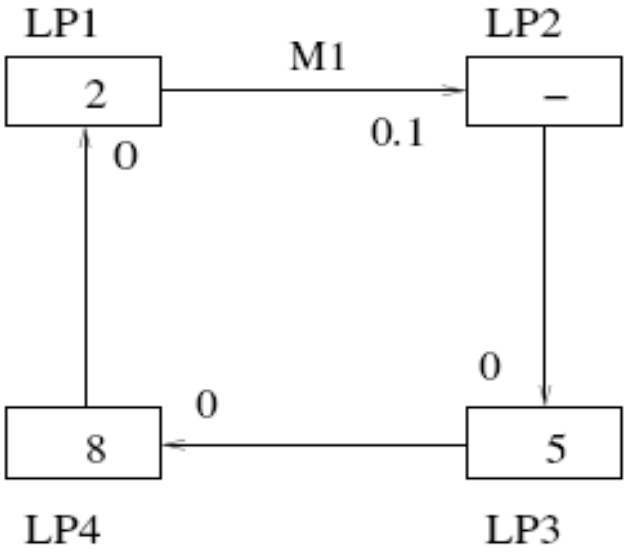
Behandlung von Deadlocks

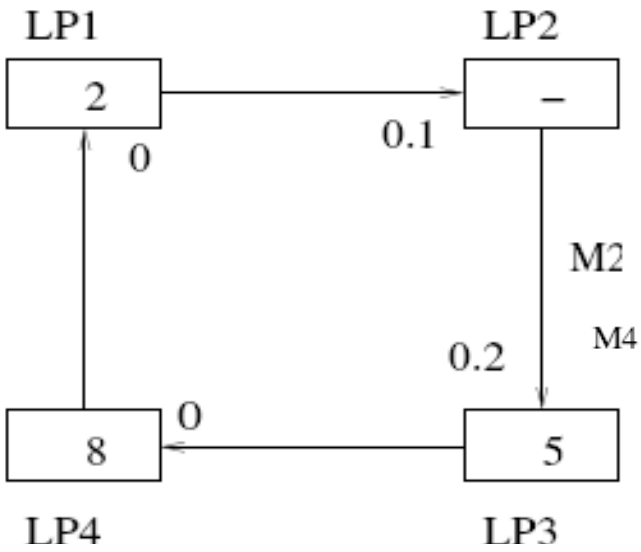
Deadlockvermeidung

- Sendung von Nullnachrichten mit Zeitstempel
  (d.h. Garantien, keine Nachricht mit kleinerem Zeitstempel als t_i zu schicken)
- Nullnachrichten erfordern einen lokalen Blick in die Zukunft
  (auch wenn Nachrichten auf allen Eingangskanälen ankämen, würde lokal keine Nachricht mit kleinerem Zeitstempel generiert)
- Verfahren funktioniert immer dann, wenn auf jedem Zyklus von LPs eine Zeitverzögerung existiert

Problem des Ansatzes: zahlreiche Nachrichten bei u.U. geringem Zeitfortschritt

Beispiel mit Zeitverzögerung $\Delta = 0.1$ auf allen Kanälen



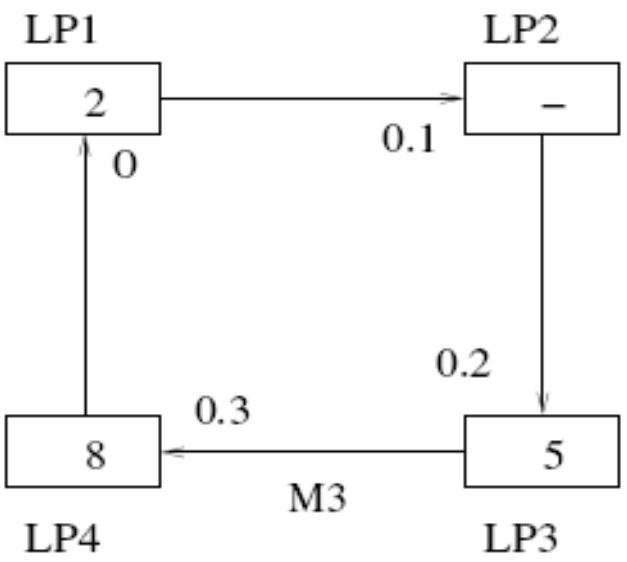


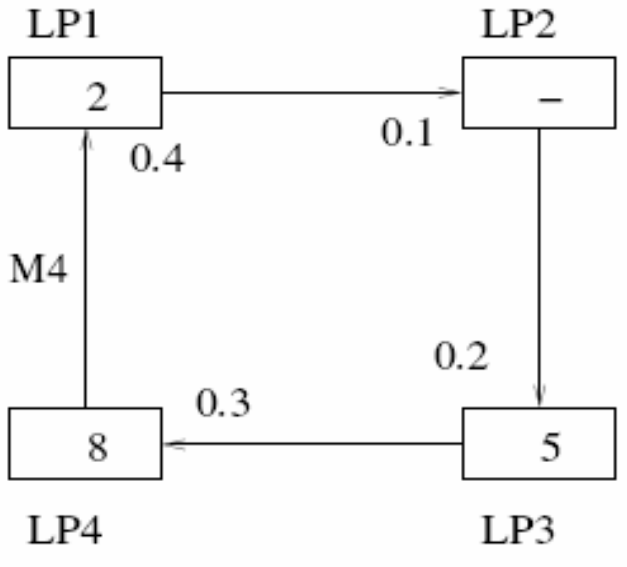
M1 = (0.1, (LP1,2))

M2 = (0.2, (LP1,2),(LP2,–))

M3= (0.3, (LP1,2),(LP2,–),(LP3,5))

M4 = (0.4, (LP1,2),(LP2,–),(LP3,5),(LP4,8))





Verbesserungen:

- Nullnachrichten auf Anforderung
- Nullnachrichten nur von blockierten LPS
- Löschen nicht aktueller Nullnachrichte
- Bündelung von Nullnachrichten
- Carrier Null Message Ansatz

Alternative Deadlockvermeidung und –behebung

- Entdeckung und Behebung durch globalen Prozess
- Kreisender Marker
  - Alle K Kanäle werden zyklisch besucht
  - Marker speichert lokale Ereigniszeit und setzt LP auf Zustand weiß
  - Bei Sendung oder Empfang einer Nachricht wird LP auf Zustand rot gesetzt
  - Falls Marker K Kanäle mit weißen Prozessen nacheinander besucht hat, wird der LP mit der nächsten lokalen Ereigniszeit aufgeweckt und kann fortfahren

Vermeidung von Pufferüberläufen

- Flusskontrolle zwischen LPs

Problem konservativer Ansätze

- Langsamer Zeitfortschritt (und schlechte Effizienz) durch zahlreiche Kontrollnachrichten

Vorteil konservativer Ansätze

- Relativ einfache Realisierung
  (auch zur Parallelisierung sequentieller Simulatoren)

Optimistische Protokolle

LPs führen immer ihr nächstes lokales Ereignis aus

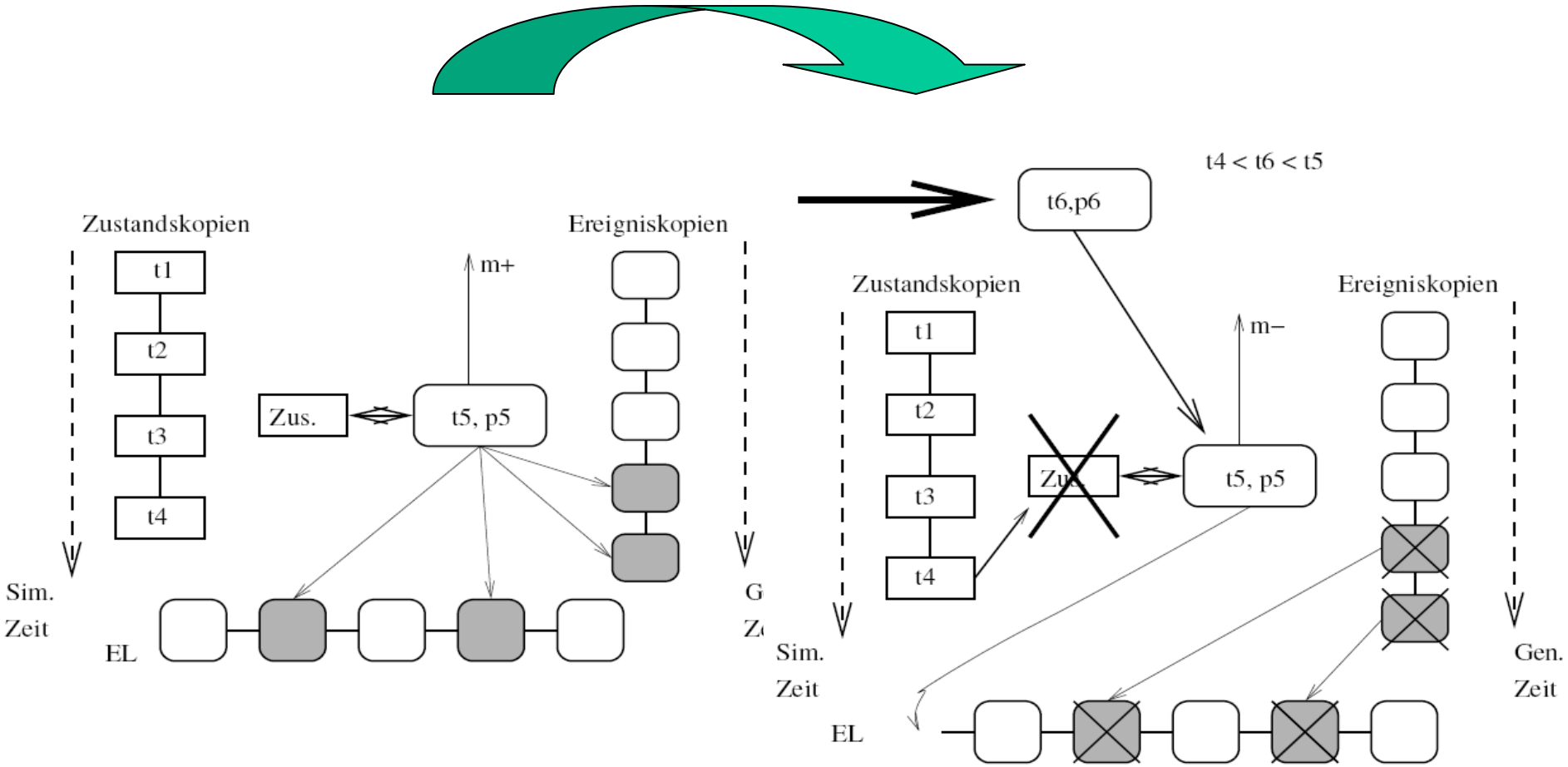
Bei Empfang einer Nachricht mit Zeitstempel t < lt

1. Zustand zurücksetzen (d.h. Zustände vorher speichern!)
2. Antinachrichten verschicken, für alle zwischen t und lt verschickten Nachrichten
   a. Antinachrichten bedingen „Zurücksetzen“ der Zeit beim Empfänger
   b. U.U. Generierung weiterer Antinachrichten beim Empfänger

Bewertung des Vorgehens

+ Potenzielle Parallelität wird besser genutzt
+ FIFO Übertragung von Nachrichten nicht notwendig
– Höherer Verwaltungsaufwand
– Hoher Speicherplatzbedarf (durch Zustandsspeicherung)
– Komplexe Realisierung
– u.u. großer Overhead

Schematischer Ablauf:



Behandlung von Antinachrichten

- Löschen nicht bearbeiteter Nachrichten
- Zurücksetzen bearbeiteter Nachrichten, verschiedene Möglichkeiten
  - „aggressive cancellation“ versendet Antinachrichten für alle Nachrichten aus dem Intewrvall (t,lt]
  - „lazy cancellation“ versendet nur Antinachrichten, falls eine Nachricht nicht erneut beim geänderten Simulationsablauf wieder versendet würde

  Je nach Modell (zum Teil deutliche) Vorteile für die eine oder die andere Variante

Speicherverwaltung

- Prozesszustände umfassen Zeitpunkt, Werte der Zustandsvariablen, gesendete Nachrichten
  GVT = min ($\min_i(lt_i)$, $\min_n(mt_n)$)
  mit mt_n Zeitstempel unbearbeiteter Nachrichten
- Bestimmung einer unteren Schranke für GVT mittels Schnappschussalgorithmus
- Zustände zum Zeitpunkt t_i < GVT können gelöscht werden
- Irreversible Ereignisse (z.B. Bildschrimausgabe, statistische Auswertung) erst bei t_i < GVT ausführen

Behandlung von Speichermangel

- Inkrementelle Zustandsspeicherung
- Zustandssicherung nur alle s Ereignisse
- Zurücksetzen nicht bearbeiteter Nachrichten
- Künstliches Zurücksetzen oder Anhalten von LPs
- Beschränkter Optimismus

$\Rightarrow$ In allen Fällen Kompromiss zwischen Speicher- und Zeitbedarf

Hybride Protokolle

- Optimismus in konservativen Protokollen
  (spekulative Simulation durch Nutzung der Wartezeit zur Weiterführung der lokalen Simulation, ohne Nachrichten zu verschicken)
- Konservativismus in optimistischen Protokollen
  (Reduzierung der Geschwindigkeit bei steigendem Speicherplatzbedarf, probabilistische Verhaltensvorhersage)
- Anwendungsspezifische Protokolle