Ein Laufzeitsystem für hochgradig parallele Simulationen

Luc Bläser

ETH Zürich / LBC Informatik

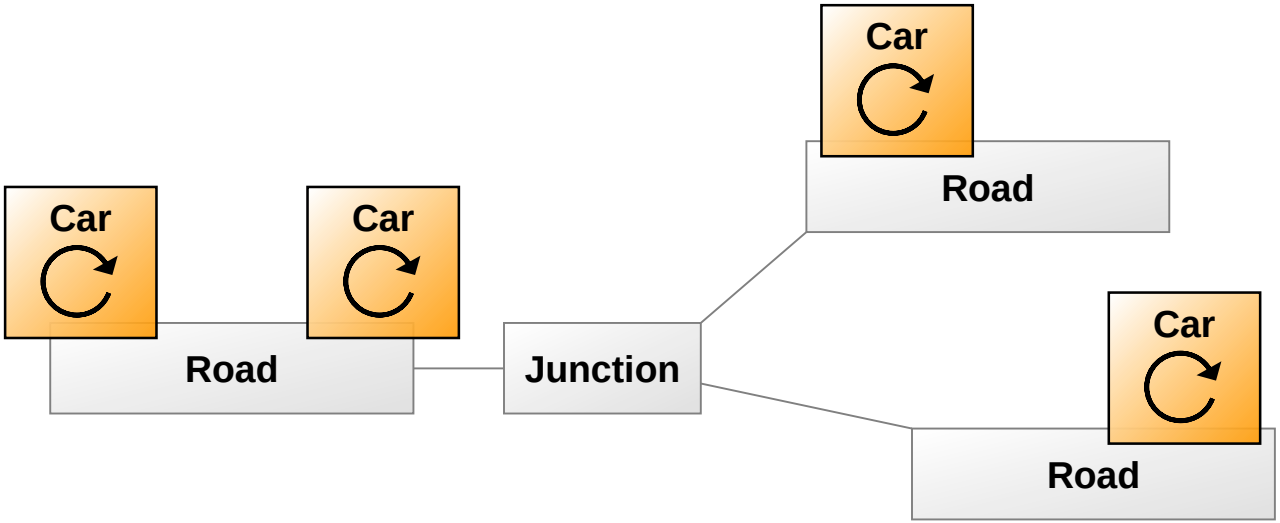
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

LBC Informatik
Software-Ingenieure ETH

Seminar für Verkehrssimulation
TU Berlin, 6. Juni 2008

Motivation

- Parallele Simulation
  - Selbstaktive Agenten mit eigenem Prozess
  - Agenten laufen selbständig und parallel in einer virtuellen Zeit

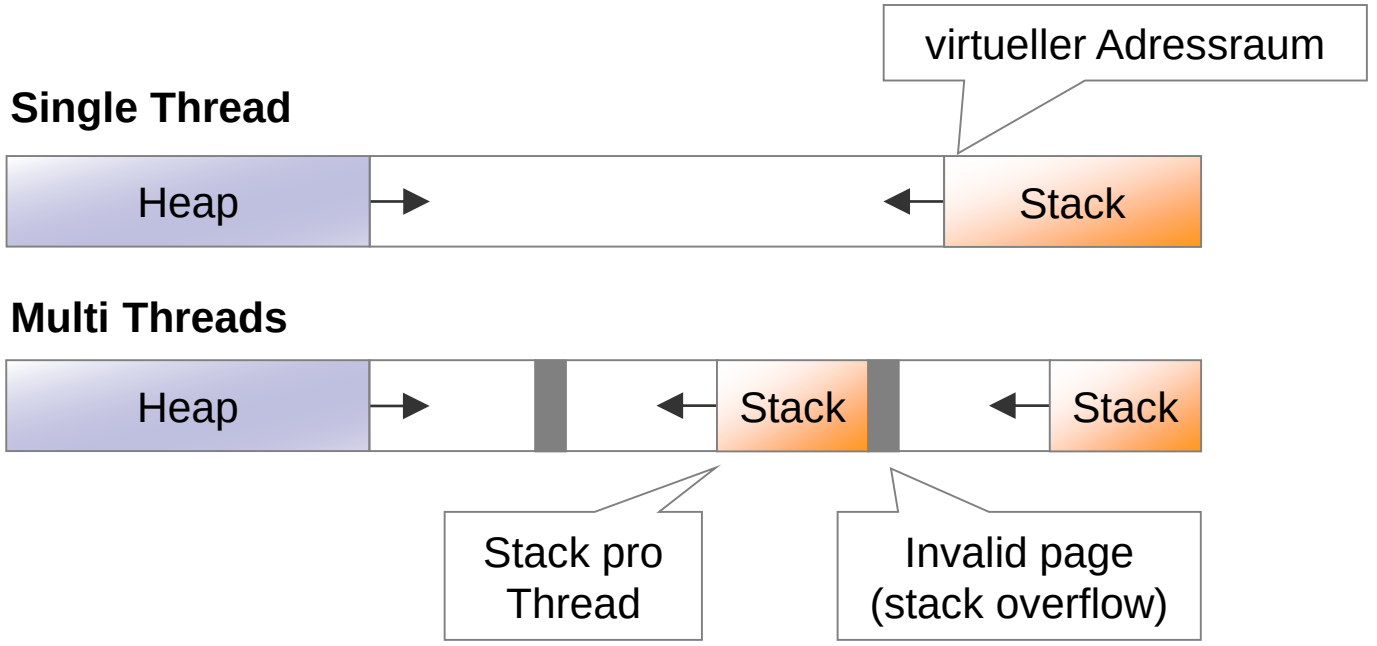


- Systemanforderungen
  - Sehr viele Leichtgewichtsprozesse
  - Sehr schnelle Nebenläufigkeit

Heutige Laufzeitsysteme

Schwergewichtige Threads

- Stack: dynamische Größe durch Verwendung virtuellen Speichers
- Stack-Grösse auf Page Granularität

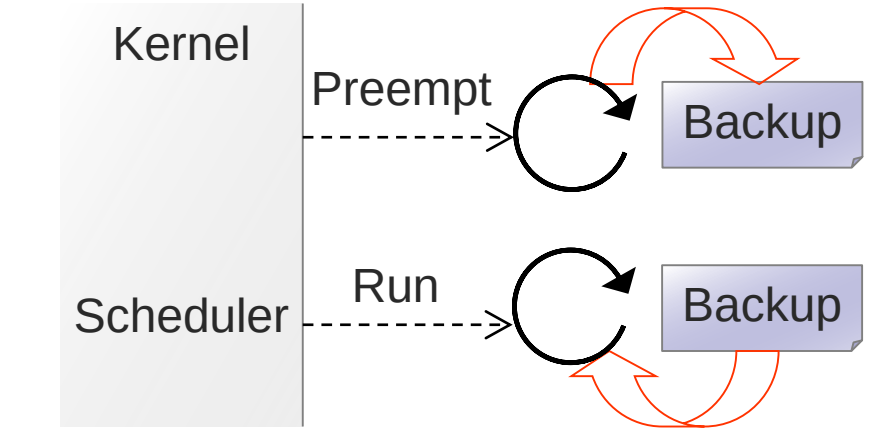


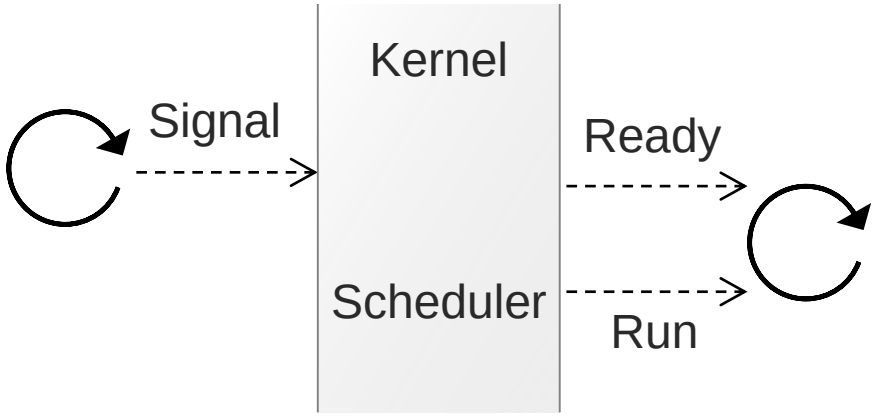
- Stack entweder zu klein oder zu gross
- Knappheit des virtuellen Speichers
  - 256K pro Stack => maximal ca. 10‘000 Threads möglich

Heutige Laufzeitsysteme

Langsame Kontextwechsel

- Asynchroner Wechsel
  - Sicherung aller Register bei Preemption
  - Vorreservierter Backup-Speicher pro Thread (ca. 1KB)
- Synchroner Wechsel
  - Indirekter Wechsel via Scheduler
  - System Call via Software Interrupt





Heutige Laufzeitsysteme

Fazit

- Sehr kleine Anzahl Threads
  - Grosse und grobgranulare Stacks
  - Substanzieller Register-Backup für Preemption
- Langsame Kontext-Wechsel
  - Indirekt via Scheduler
  - System Call per Interrupt

$\Rightarrow$ Effiziente Parallele Simulation nicht möglich

Neues Simulationssystem

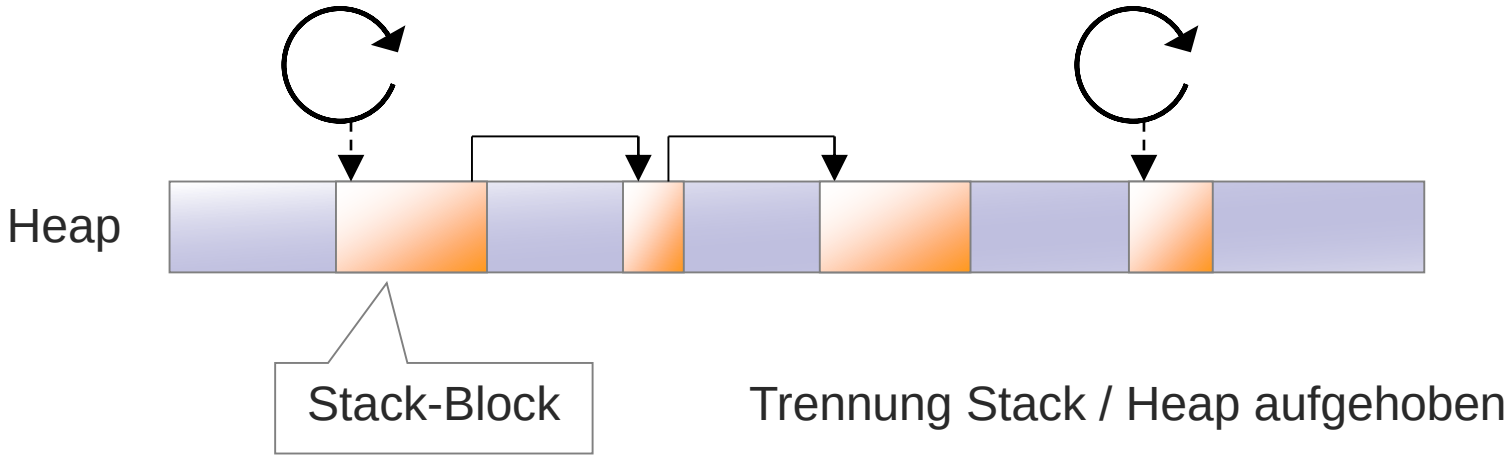
Betriebssystem für hochgradige effiziente Nebenläufigkeit

- Leichtgewichtsprozesse
  - Dynamische Mikro-Stacks
- Schnelle Kontextwechsel
  - Ausschliesslich direkte synchrone Wechsel
  - Sparsame Preemption
- Effiziente Synchronisation
  - Protokollbasierte Kommunikation
  - Systemverwaltete Monitors
- Effiziente Speicherverwaltung
  - Hierarchische Speicherverwaltung
  - Keine virtuelle Speicherverwaltung

Leichtgewichtsprozesse

Mikro-Stacks

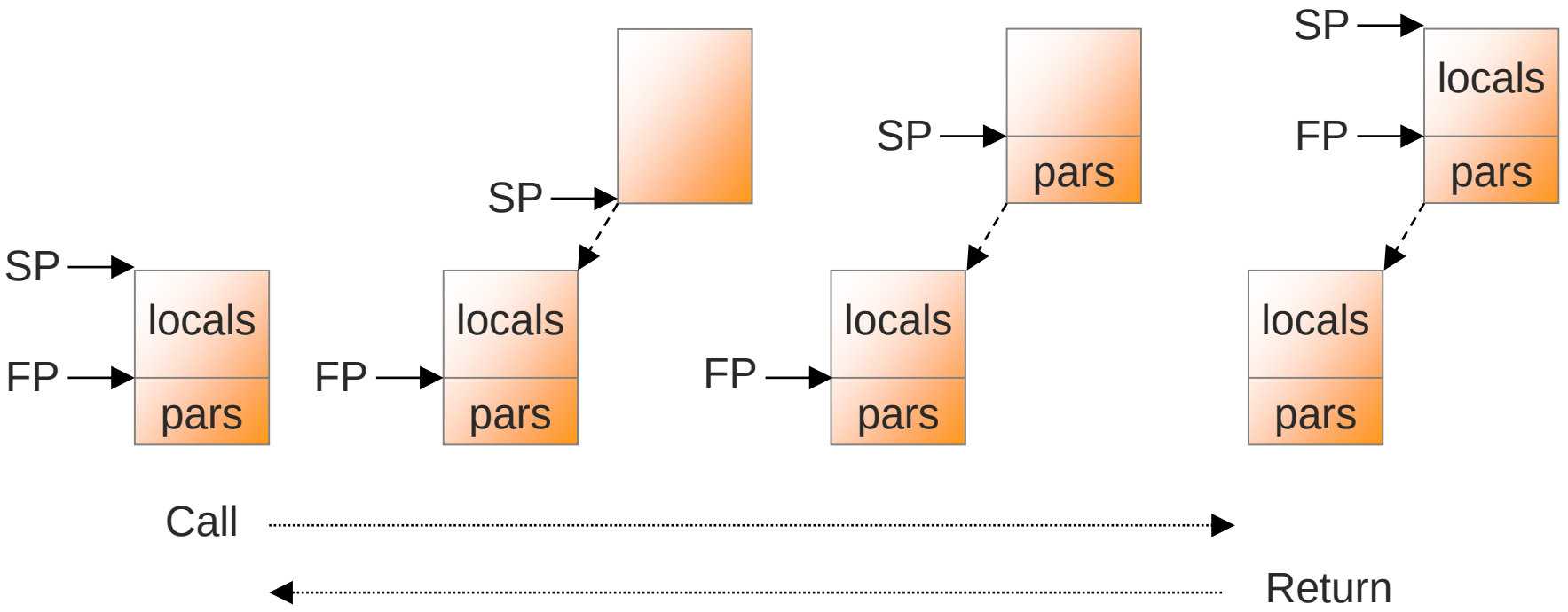
- Beliebig kleine Stacks
  - Größe nicht auf Page Granularität fixiert
- Stack als Liste von Blöcken beliebiger Größe
  - Dynamisches Wachsen und Schrumpfen



- Initialgröße von Compiler berechnet
  - Kommunikation statt Methoden: Weniger Prozeduraufrufe
  - Stack der Agenten/Komponenten bleibt meist gleich groß

Leichtgewichtsprozesse

- Dynamische Stacks
  - Wachsen bei Prozeduraufruf und Schrumpfen bei Prozedurende
  - Compiler fügt Code bei Prozeduraufruf und Rücksprung ein

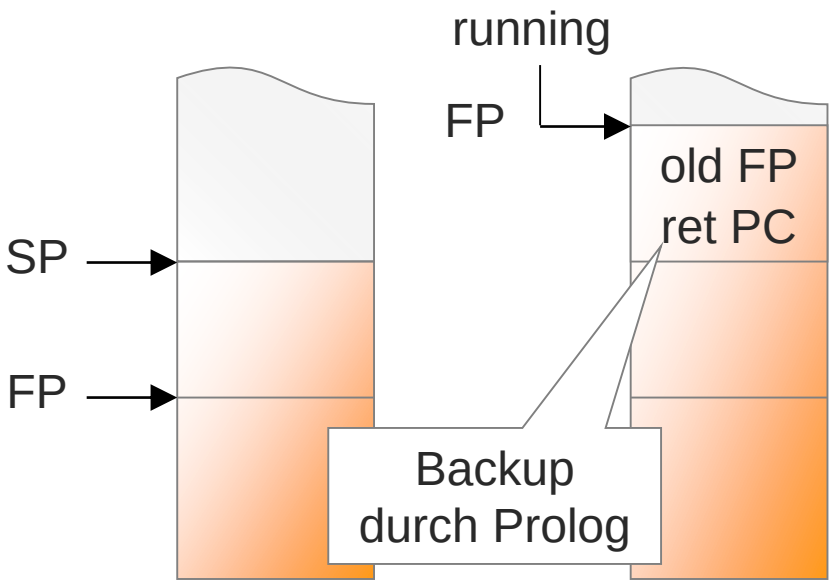


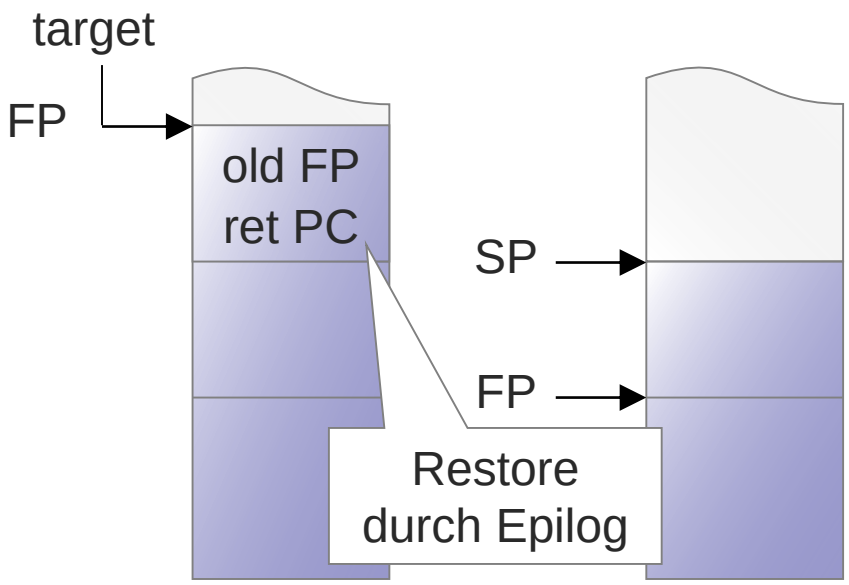
- Systemaufrufe und Interrupte
  - Auf Prozessor-zugeordnetem System-Stack (Run to completion)

Synchroner Kontextwechsel

- Systemaufruf via Prozeduraufruf
  - Kein Software Interrupt
  - Keine Kernel Protection wegen sicherer Sprache
- Direkter Wechsel zum Zielprozess

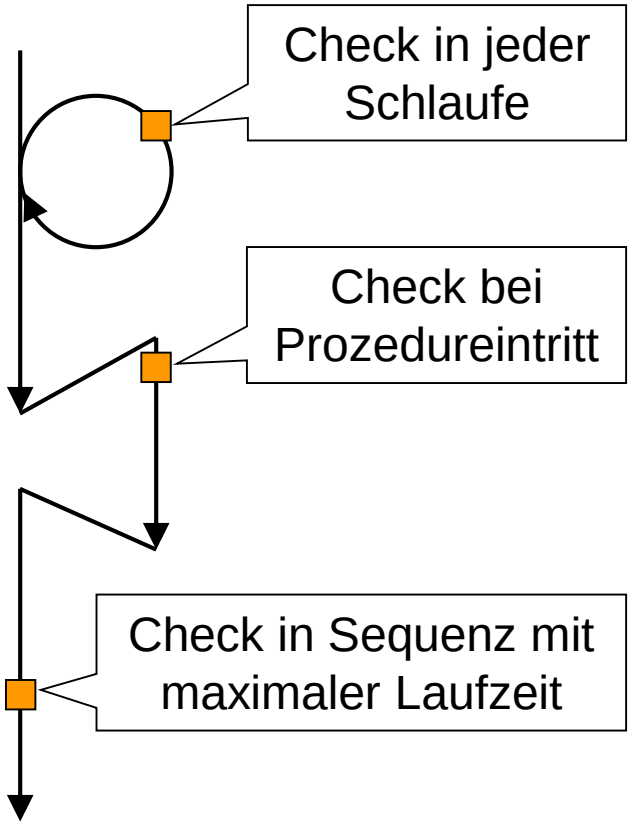
```
PROCEDURE Switch(target: Process);
BEGIN
  running := REGISTER.FP;
  REGISTER.FP := target
END Switch;
```

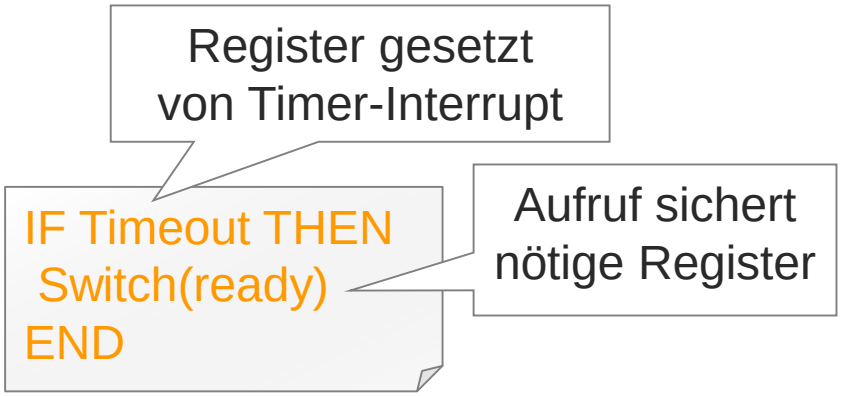




Sparsame Preemption

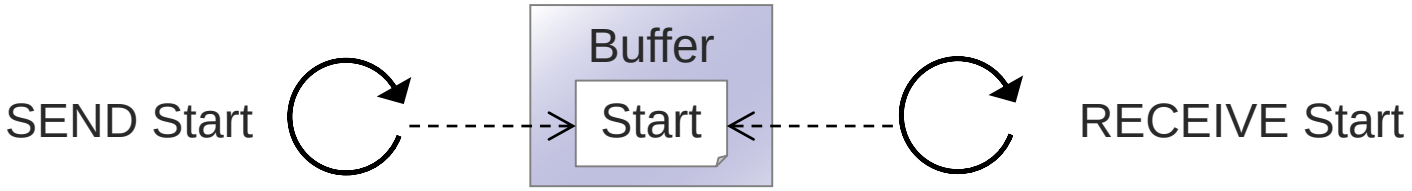
- Compiler fügt Runtime-Checks in Maschinencode ein
  - Checks in Intervallen garantierter Maximallänge ausgeführt
  - Checks initiieren Preemption nach Ablauf eines Zeitintervalls
  - Preemption sichert nur verwendete Register auf Stack
  - Sehr schnelle Checks (<0.1% Overhead)





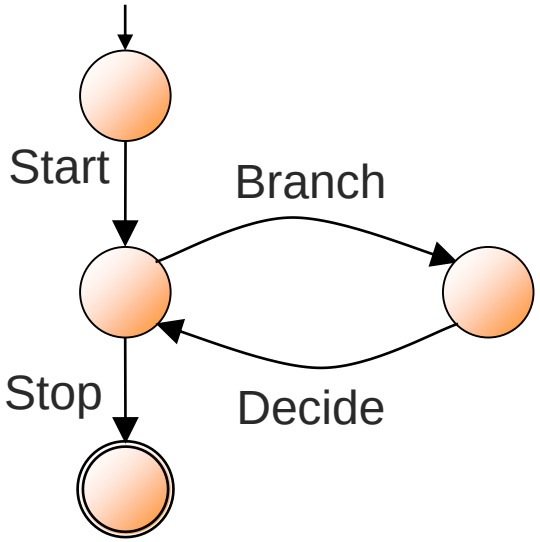
Protokollbasierte Kommunikation

- Nachrichtenaustausch zwischen Prozessen
  - Automatisch synchronisiert via internem Buffer
  - Maximale Nachrichtengrösse im Protokoll limitiert Buffergrösse



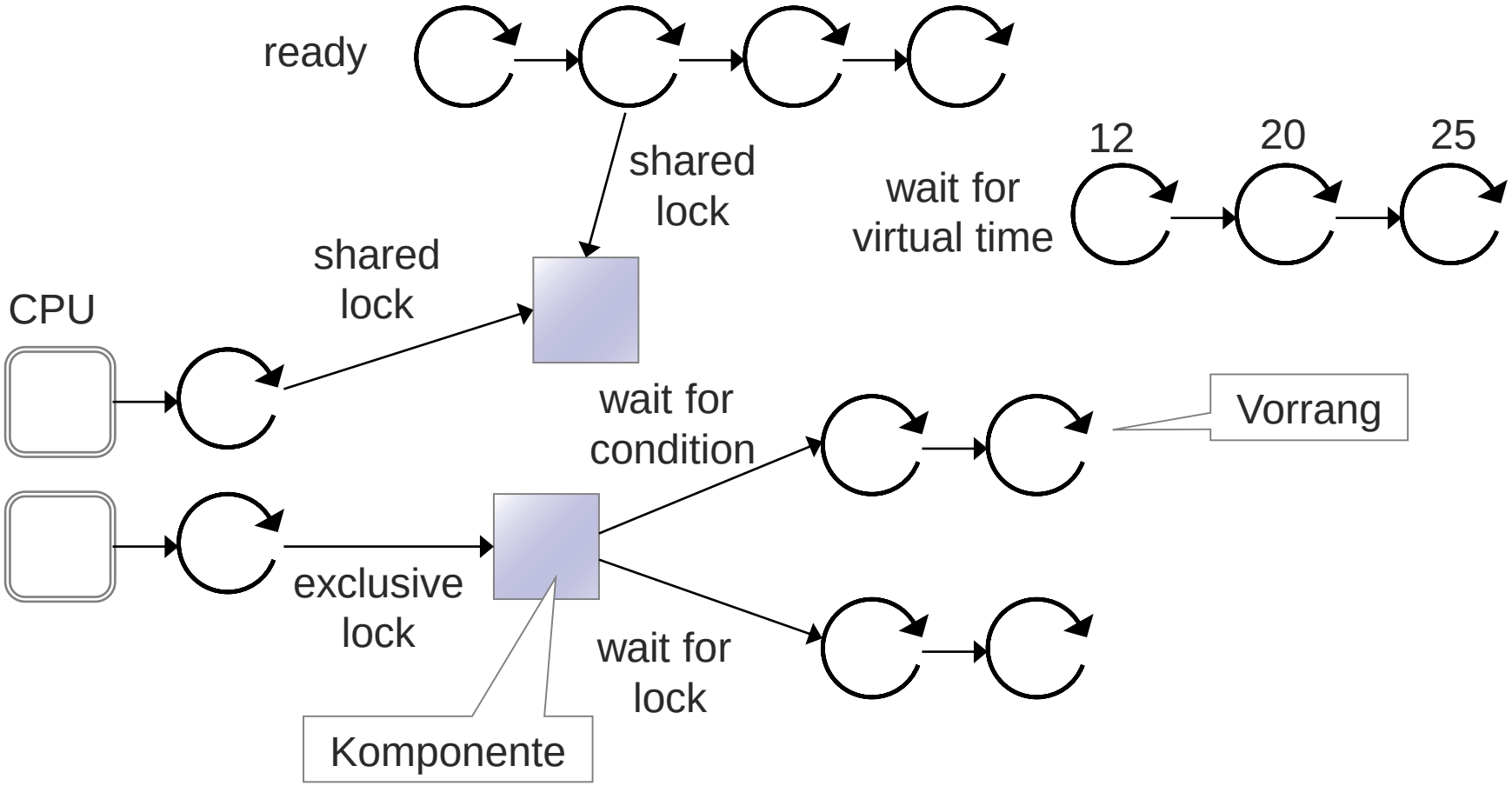
- Compiler generiert Zustandsmaschine für Protokoll
  - Sender überprüft Protokoll durch Zustandsübergang

```
IN Start
{
  IN Branch(next: LinkId)
  OUT Decide
}
IN Stop
```



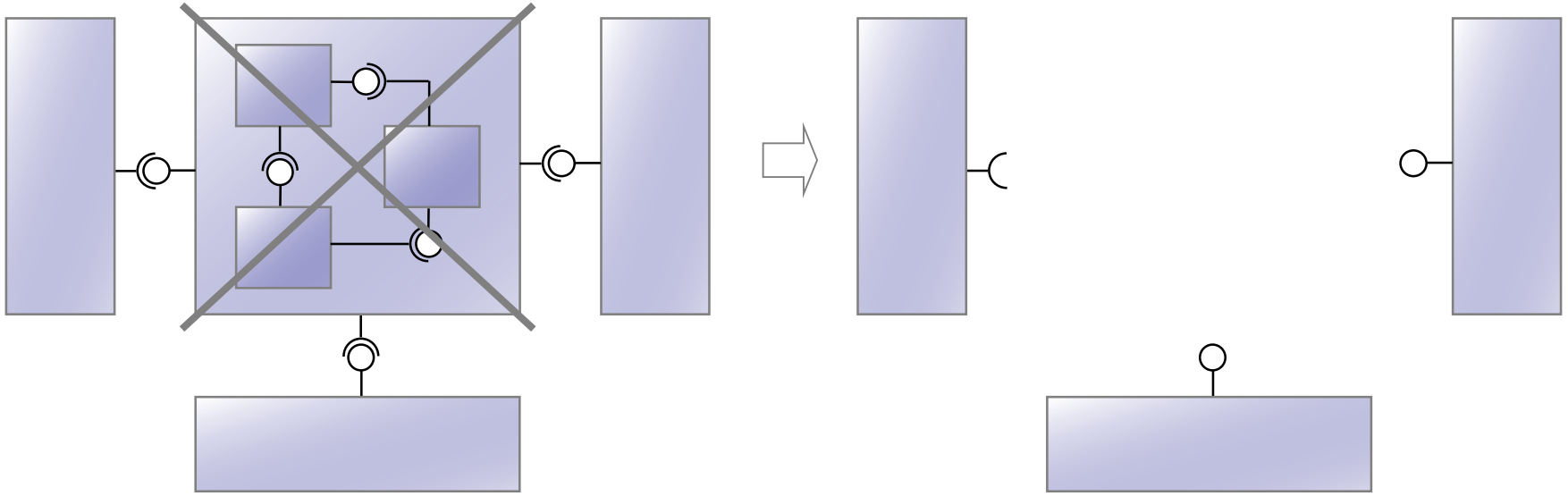
Systemverwaltete Monitore

- Warten auf systemüberwachte Bedingung
  - Wechsel zu wartendem Prozess, sobald Bedingung erfüllt wird
  - Direkte Lock-Weitergabe



Hierarchische Speicherverwaltung

- Basierend auf hierarchischen Komponenten
- Hierarchien definieren Existenzabhängigkeiten
  - Entfernen einer Komponente mit deren Unterkomponenten
  - Automatische Schnittstellen-Entkopplung beim Entfernen
- Keine Garbage Collection nötig



Virtuelle Speicherverwaltung

Nicht benötigt

- Alle Prozesse laufen im gleichen Adressraum
- Speichersichere Programme brauchen keine Isolierung
- Stack basieren nicht auf Pages (dynamisches Wachsen statt Overflow Page Faults)
- NIL Checks durch Segmentierung oder instrumentierte Checks
- Reeller Speicher ist heute meist so groß wie der virtuelle Speicher (Swapping überflüssig)
- Keine Unterscheidung zwischen reellen und virtuellen Adressen

Skalierung und Performance

- Maximale Anzahl Threads / Leichtgewichtsprozesse

Simulation OS	Windows .NET	Windows JVM	Active Oberon
5'010'000	1'890	10'000	15'700

4GB Hauptspeicher, City Programm

- Ausführungsgeschwindigkeit

Programm (sec)	Simulation OS	C#	Java	Oberon AOS
ProducerCons.	**16**	19	130	60
Eratosthenes	**1.8**	6.8	4.6	5.8
TokenRing	**2.1**	22	22	18
TrafficSim 1000 Autos	**2.4**	1980	-	-
TrafficSim 260'000 Autos	**76min**	-	-	-

C++: 210min

6 CPUs Intel Xeon **700MHz**, C# & Java Windows Server Enterprise Edition

Schlussfolgerung

Für parallele Simulationen braucht es neue Laufzeitsysteme

- Wichtige neue Features
  - Leichtgewichtsprozesse
  - Mikro-Stacks
  - Schnelle Kontextwechsel
  - Systemoptimierte Synchronisation
  - Kommunikation & Monitore
- Hinderliche alte Hüte
  - Stack / Thread Aufteilung
  - Virtuelle Speicherverwaltung
  - Prozess / Thread Unterscheidung
  - Garbage Collection
  - Pseudo-Monitore von Java / .NET