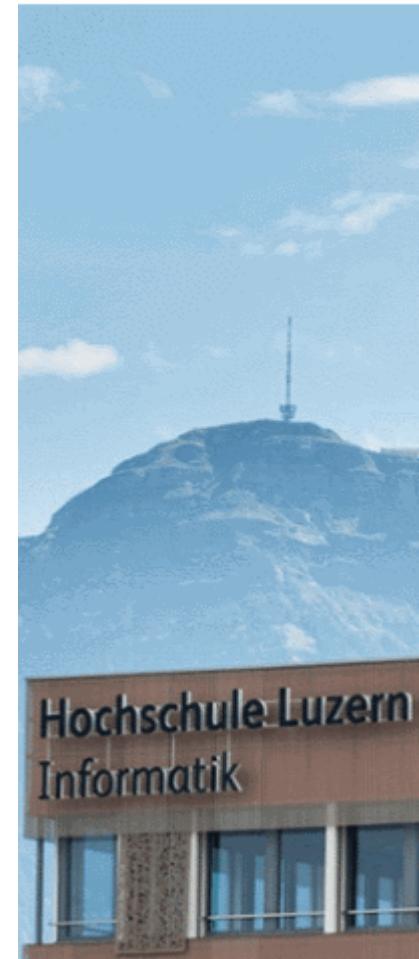


IT-Systemengineering & -Operations

# **Virtualisierung im DC Parallele Architekturen**

Vers. 1.0

Bruno Joho



# Service- & System Management

## Parallele Rechnerarchitekturen

Bruno Joho

# Ziele

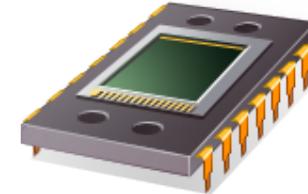
- Sie kennen und verstehen moderne multi Core Architekturen, Memory Architekturen, Cache Verwaltungen und Verbindungsnetzwerke.
- Sie können Einflüsse der Chip Architektur auf das Performanceverhalten eines Systems abschätzen.
- Sie verstehen den Einsatz der multi Core Prozessoren in der Virtualisierung.
- Sie verstehen die Implikationen der modernen Prozessoren auf bei der Ausführung ihrer Programme.

# Parallele Rechnerarchitekturen



- Symmetrische Multi Prozessoren
- Multi Core Prozessoren
- Speicherorganisation
- Simultaneous Multi Threading
- Leistungsmasse
- Cache Koherenz
- Verbindungsnetzwerke
- Skaliert ihre Applikation?

# Symmetrische Multiprozessoren (SMP)



Ein stand-alone Computer mit den folgenden Charakteristika:

- Zwei oder mehr gleichartige Prozessoren mit vergleichbaren Möglichkeiten.
- Prozessoren teilen sich das gleiche main Memory und sind über einen Bus oder andere interne Verbindungen zusammengeschaltet.
- Prozessoren teilen sich die I/O Devices.
- Alle Prozessoren können die gleichen Funktionen ausführen (daher die Bezeichnung „symmetrisch“).
- Das System wird durch ein integriertes OS kontrolliert. Dieses stellt die Interaktion her zwischen Prozessoren und deren Programmen auf dem Level von Job, Task, File und Daten-Elementen.

# Vorteile SMP

## Performance

- Ein System mit mehreren Prozessoren führt zu einer grösseren Performance, wenn die Arbeit parallel ausgeführt werden kann.

## Availability

- Der Ausfall eines Prozessors führt nicht zum Stillstand des ganzen Rechners.

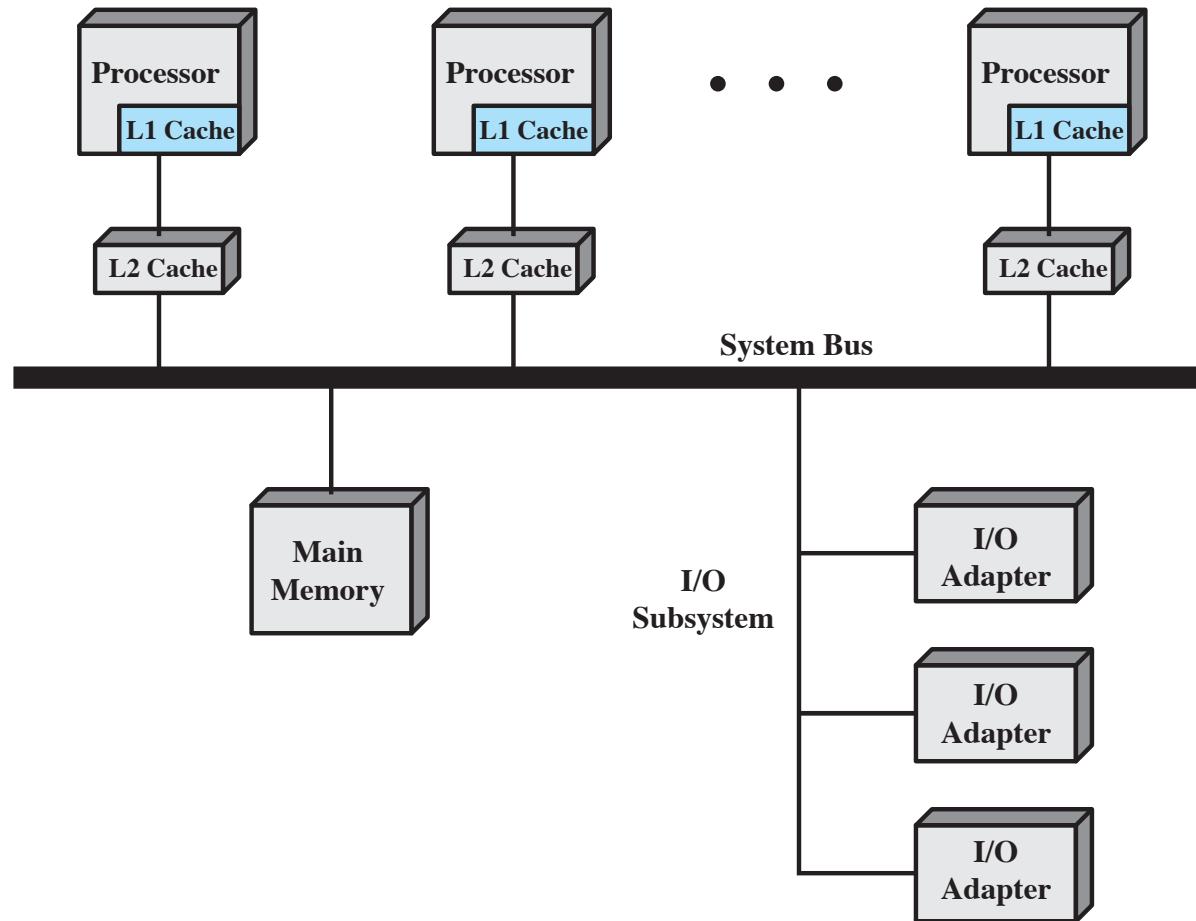
## Scaling

- Hersteller können eine Palette von Produkten mit verschiedenen Preisen und Leistungen anbieten.

## Incremental Growth

- Zusätzliche Prozessoren können dazu gefügt werden um die Performance zu erhöhen.

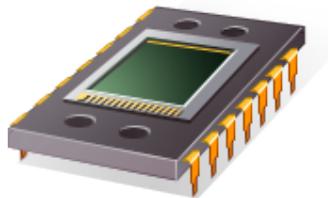
# SMP Organisation



# Parallele Rechnerarchitekturen

- Symmetrische Multi Prozessoren
-  - Multi Core Prozessoren
- Speicherorganisation
- Simultaneous Multi Threading
- Leistungsmasse
- Cache Koherenz
- Verbindungsnetzwerke
- Skaliert ihre Applikation?

# Multicore Computer



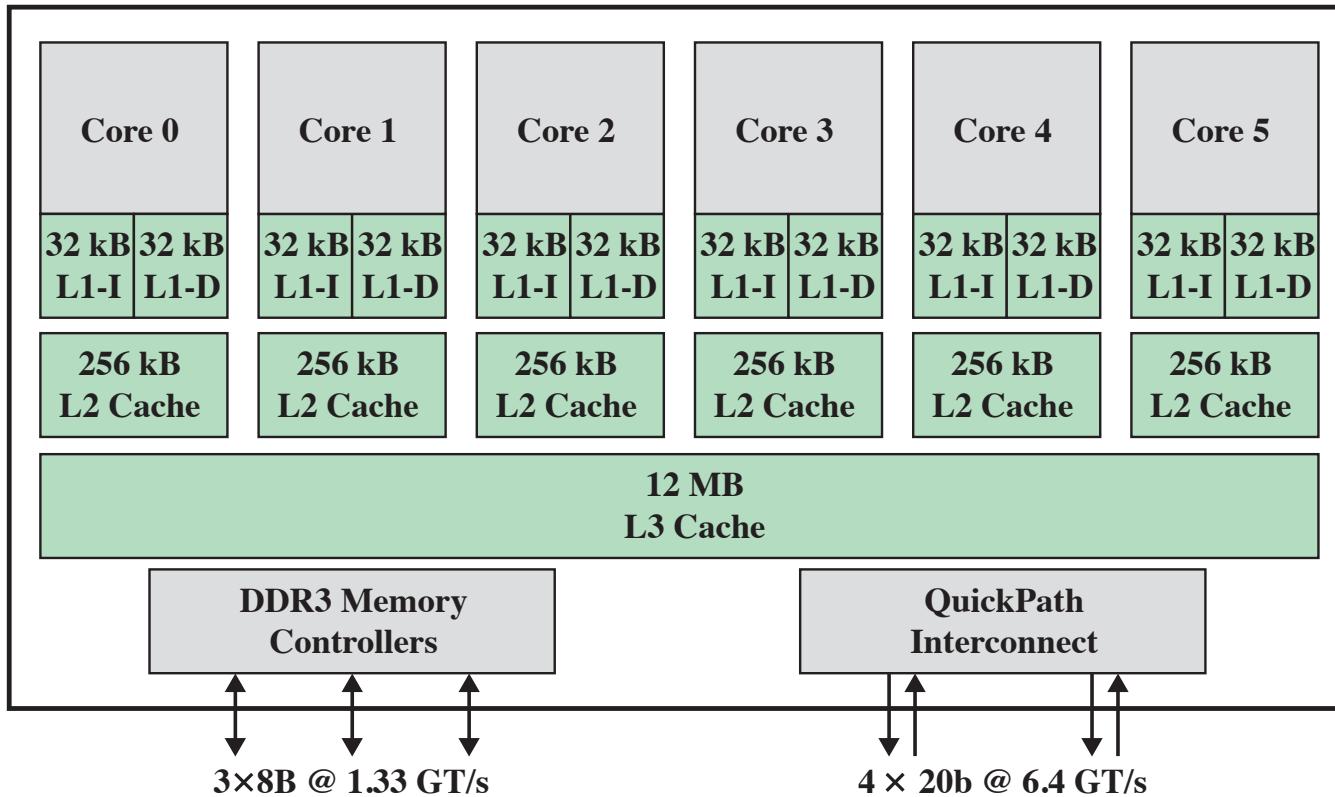
- Besser bekannt als Chip Multiprozessor.
- Kombiniert 2 oder mehr Prozessoren (Cores) auf einem Silikon Die.
- Jeder Core beinhaltet alle Komponenten von einem unabhängigen Prozessor.
- Zusätzlich beinhalten multicore Chips L2 Cache und unter Umständen sogar L3 Caches.

# Vergleich Anzahl Prozessoren

Kategorie	Architektur		# Prozessoren
Kommunikationsmodell II	Message Passing		8 - 256
	Shared Adress	NUMA	8 – 256
		UMA	2 – 64
Physische Verbindung	Network		8 – 256
	Bus		3 - 32

Mit steigender Anzahl Cores wird beim Kommunikationsmodell das Nachrichtenmodell und bei der physischen Verbindung das Netzwerkmodell bevorzugt.

# Multicore System: Intel Core i7-990X



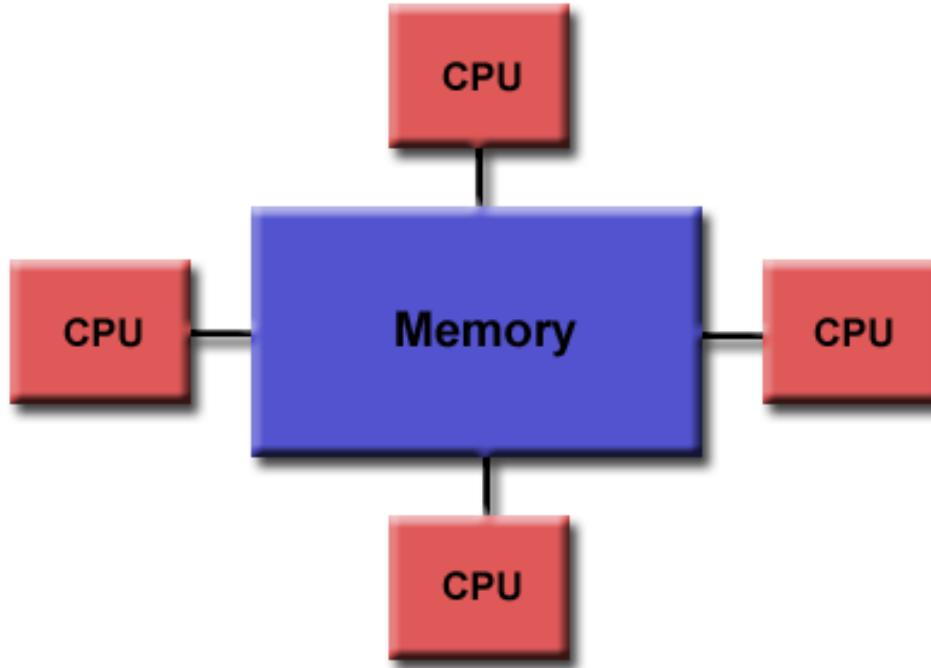
Nicht ersichtlich hier ist das Core zu Core Netzwerk (vergleiche Folie im Teil „Verbindungsnetzwerke“)

# Parallele Rechnerarchitekturen

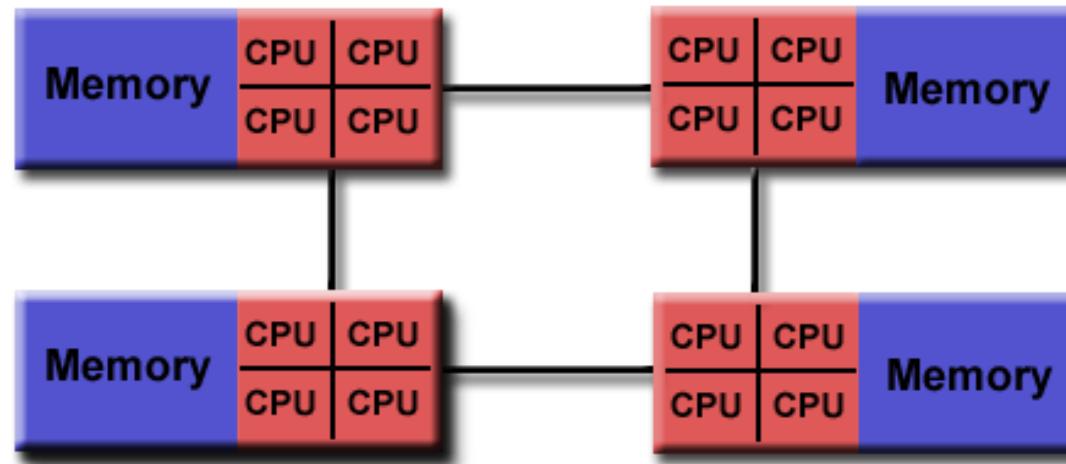
- Symmetrische Multi Prozessoren
- Multi Core Prozessoren
-  - Speicherorganisation
- Simultaneous Multi Threading
- Leistungsmasse
- Cache Koherenz
- Verbindungsnetzwerke
- Skaliert ihre Applikation?

# Shared Memory Architektur (**UMA**)

- Alle Prozessoren (P) haben direkten Zugriff auf alle Speicherstellen (M)
- einfacher und schneller Kommunikationsmechanismus
- Private-Memory: mehrere lokale Addressräume



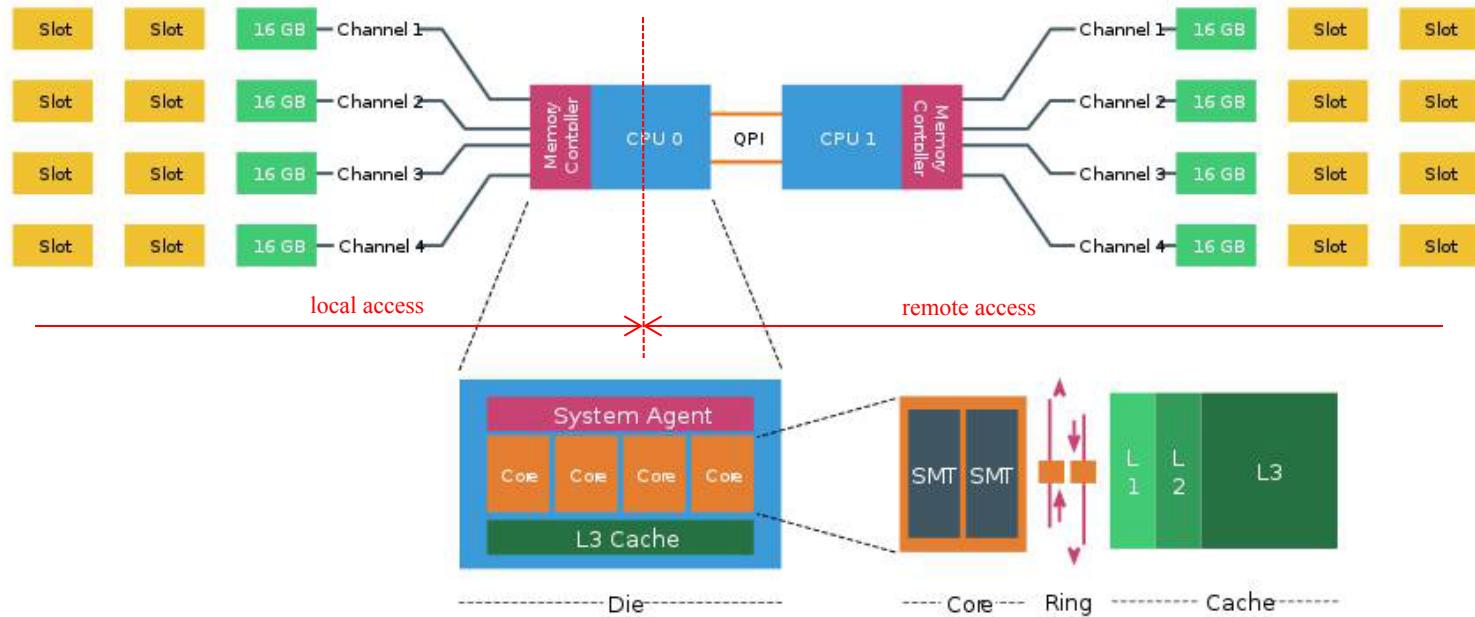
# Shared Memory Architektur (NUMA)



- meistens werden 2 oder mehr SMPs physikalisch verbunden.
- Shared-Memory: ein globaler linearer Addressraum
- Ein SMP kann direkt auf das Memory der anderen SMPs zugreifen.
- Nicht alle Prozessoren haben die gleiche Zugriffszeit auf das Memory. Speicherzugriffszeit =  $f(\text{Speicheradresse})$
- Memory Zugriff auf „remote“ Memory ist langsamer.
- Wenn Cache Kohärenz unterhalten wird, reden wir von CC-NUMA.

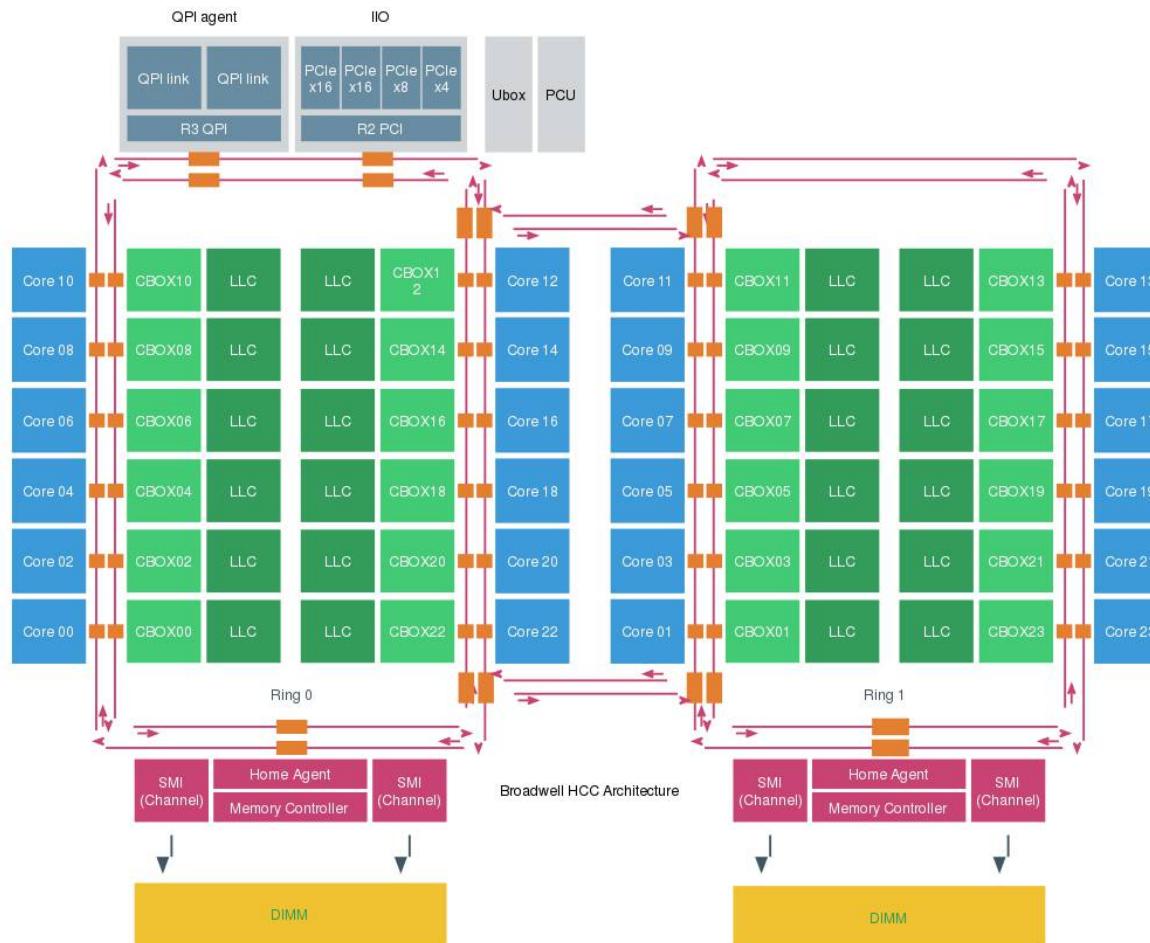
# System mit 2 NUMA Nodes und 8 Prozessoren

Ein 2 Socket (2 NUMA Nodes) hat eine Memory Latency von ~ 100ns. Memory Access (random) auf einen entfernten Node addiert zusätzliche 50ns.



Die OSs ignorieren meistens die Geschwindigkeitseinbussen von der NUMA Memory Lokalität. Für spezielle Applikationen haben Operating Systeme APIs für explizite Memory Kontrolle. Linux, Solaris, und Microsoft Windows stellen dazu System Calls für NUMA bereit.

# Data Access



- vom LLC (last level cache) - Zugriff auf **CBOX**
- local memory – Zugriff auf **home agent** und **integrated memory controller**
- remote NUMA node – **QPI Agent**

# LLC Daten Platzierung

- LLC (last level cache): L1 und L2 sind zum Core lokale (private) Caches. LLC ist (ge)shared über alle Cores.
- Der ganze LLC ist aufgeteilt in 2,5MB grosse Stücke auf die alle Cores vom System zugreifen und verwenden können.
- In vielen Abhandlungen wird geschrieben, dass ein Core ein Stück dieser LLC assoziiert ist. Das ist aber nur eine physikalisches Konstrukt.
- Ein Core kann die Platzierung der Daten nicht kontrollieren. Er kann nur auf den ganzen LLC zugreifen.
- Auf den LLC wird über den on-die ring zugegriffen und die Latenz richtet sich nach den Anzahl hops die gemacht werden müssen.

*Vergleichen Sie mit dem Abschnitt über Speicherprotokolle später in dieser Vorlesung.*

# Parallele Rechnerarchitekturen

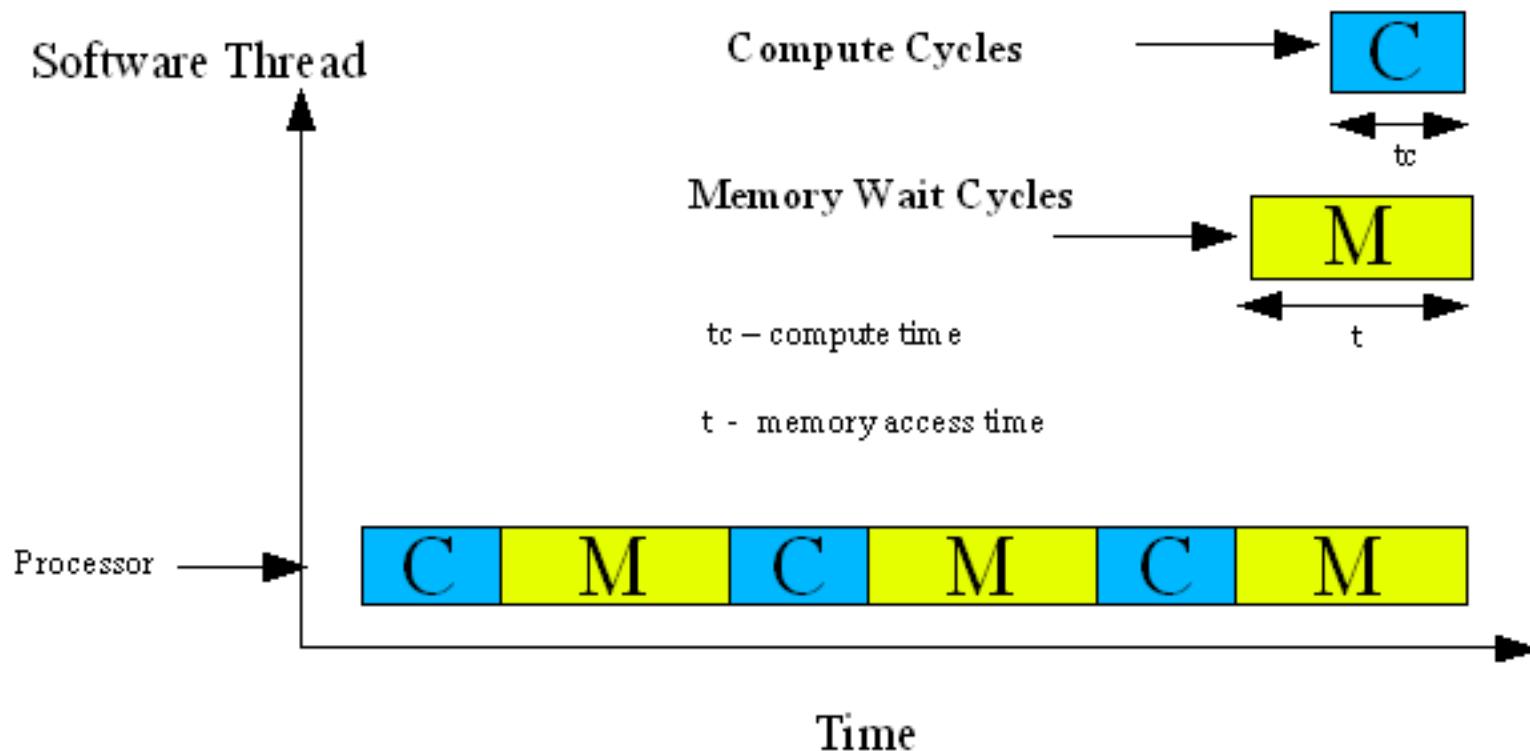
- Symmetrische Multi Prozessoren
- Multi Core Prozessoren
- Speicherorganisation
-  - Simultaneous Multi Threading
- Leistungsmasse
- Cache Koherenz
- Verbindungsnetzwerke
- Skaliert ihre Applikation?

# Memory Stalls

Wartende Applikation	Typische Anzahl Zyklen/Instruktion	Memory Stalls (%)
Transactional Database	3-6	>75%
Web Server	1,5-2,5	~50%
Entscheidungs-unterstützte DB	1-1,5	~10-50%

# Warten, warten....

50% von der Prozessor Zeit ist Warten auf das Memory



# Wie lange wartet eine CPU?

## **Beispiel CPU mit 3.3 GHz Taktrate.**

Verglichen mit den anderen Komponenten hochgerechnet auf erfassbare Dimensionen von Menschen.

Wenn ein einziger CPU Zyklus mit einer Sekunde repräsentiert wird statt mit 0.3 Nanosekunden, dann braucht das Abholen von Daten im Level 1 Cache 3 Sekunden, vom Level 3 Cache 45 Sekunden und jene vom DRAM 6 Minuten.

Das Warten auf Disk reads braucht Monate und Jahre wenn die Daten von einem entfernten Datacenter gelesen werden müssen.

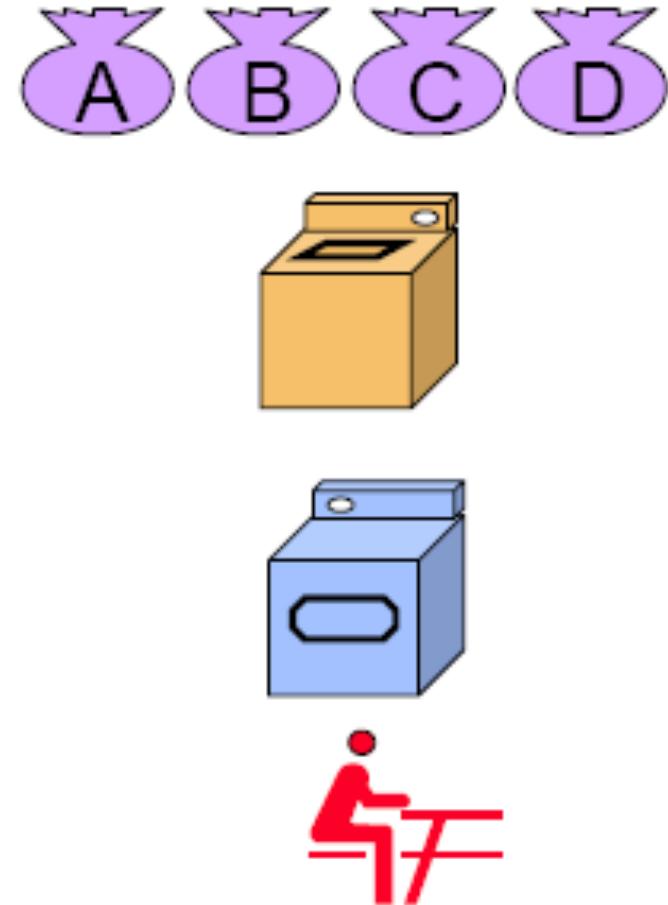
Zur Erinnerung der Skalierungsfaktor beträgt 3.3 Milliarden zu 1. Events die sich in weniger als 50 ms ereignen stufen Menschen als gleichzeitig ein.

# Multithreading

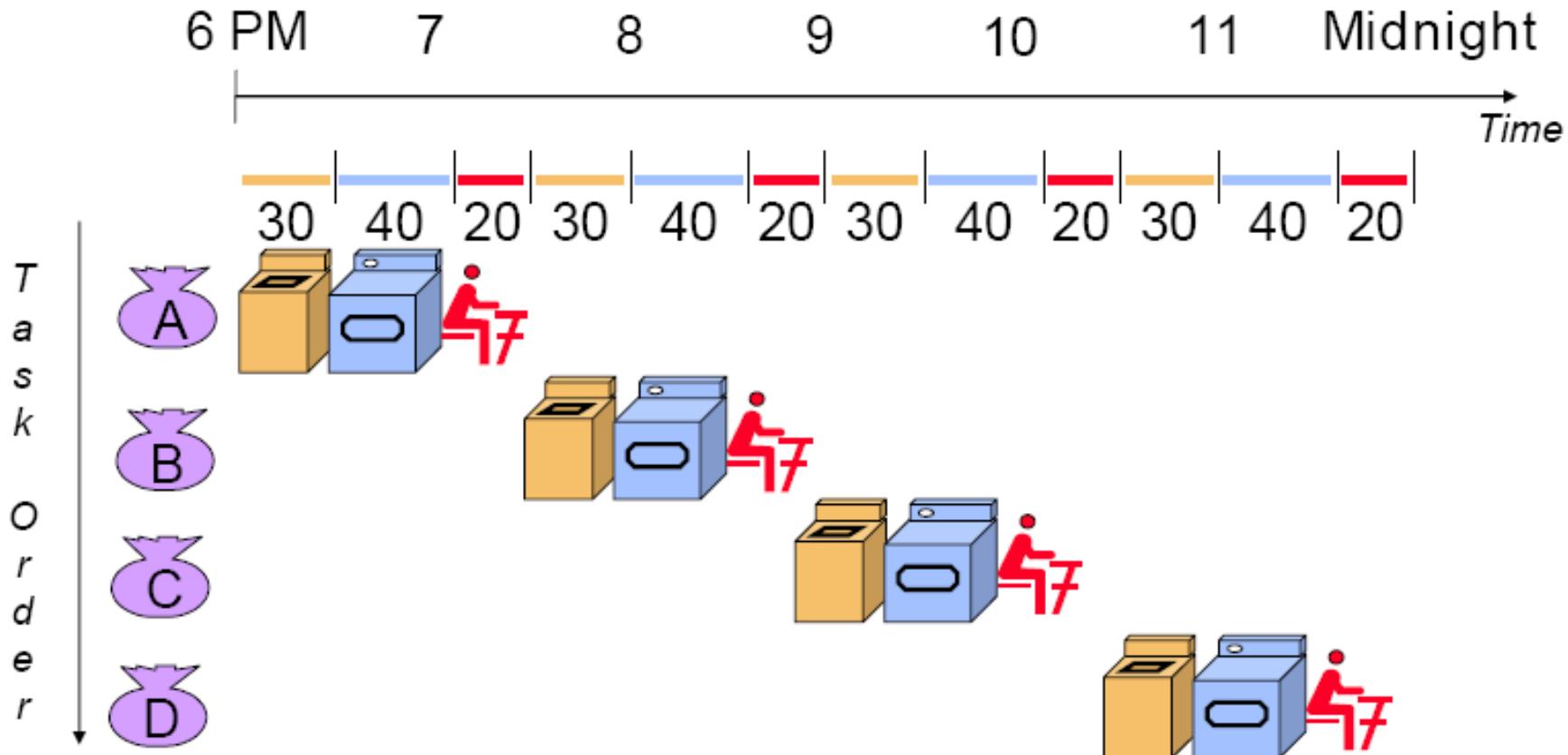
## Beispiel Waschmaschine

Bruno, Margrith und Michi haben alle Kleider zum waschen, trocknen und bügeln.

- Waschen braucht 30 Minuten
- Der Trockner braucht 40 Minuten
- Bügeln braucht 20 Minuten.

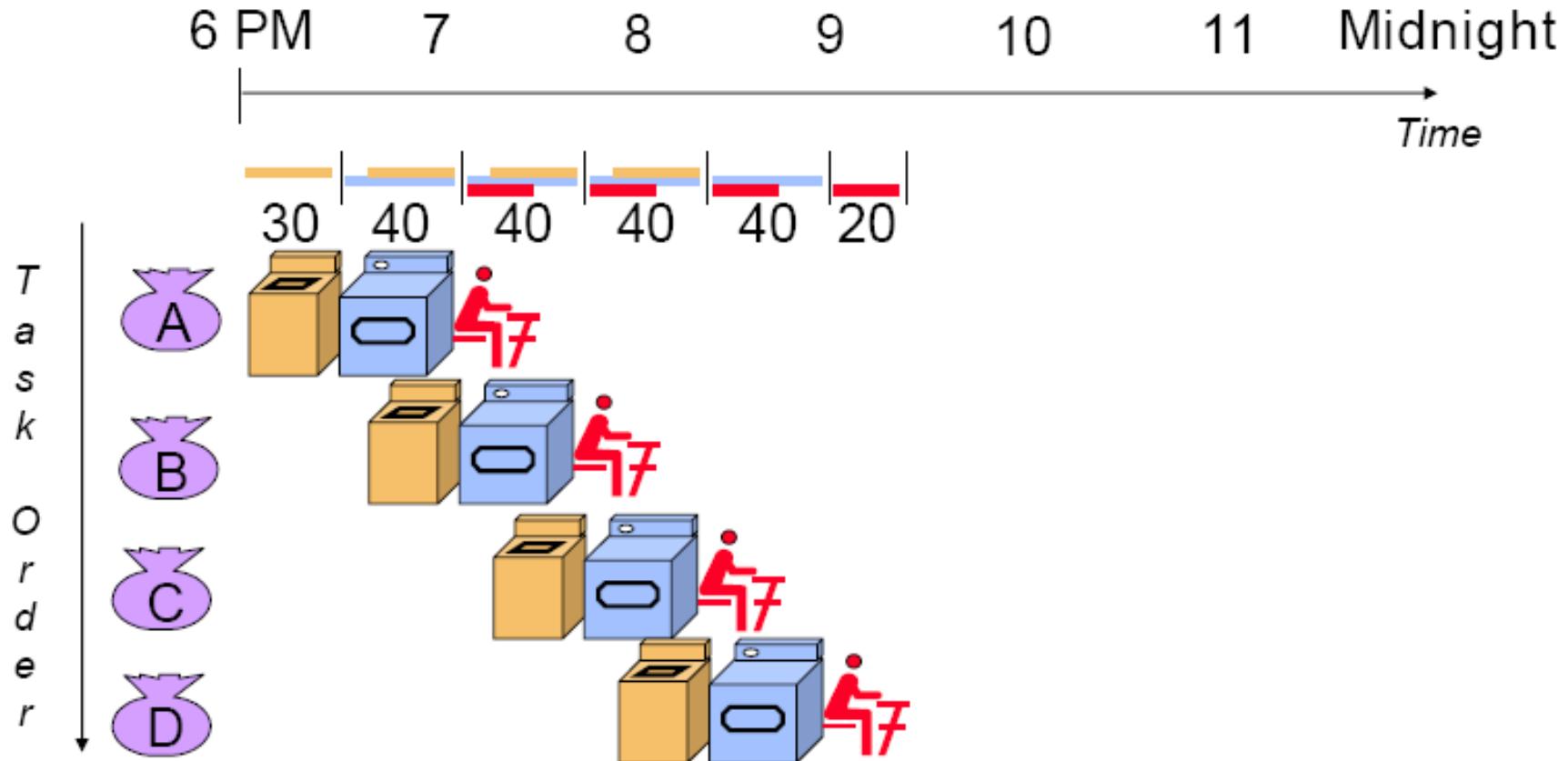


# Sequentiell



Wenn sie Pipelining kennen würden wie lange hätten sie?

# Pipelined

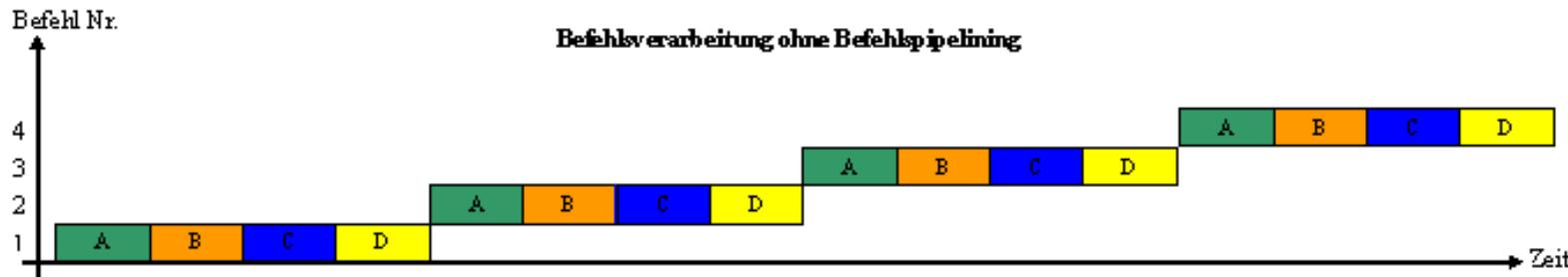
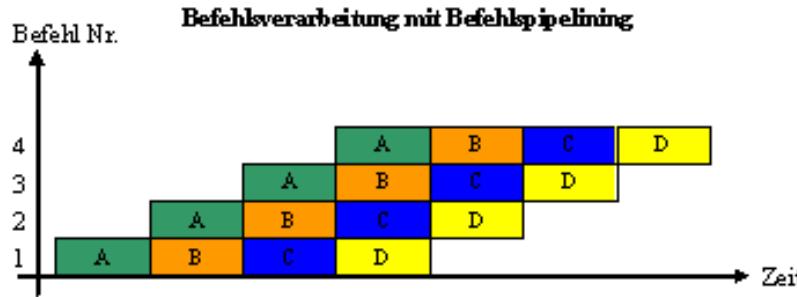


Pipelined Waschen beträgt 3,5 Stunden für 4 Waschgänge!

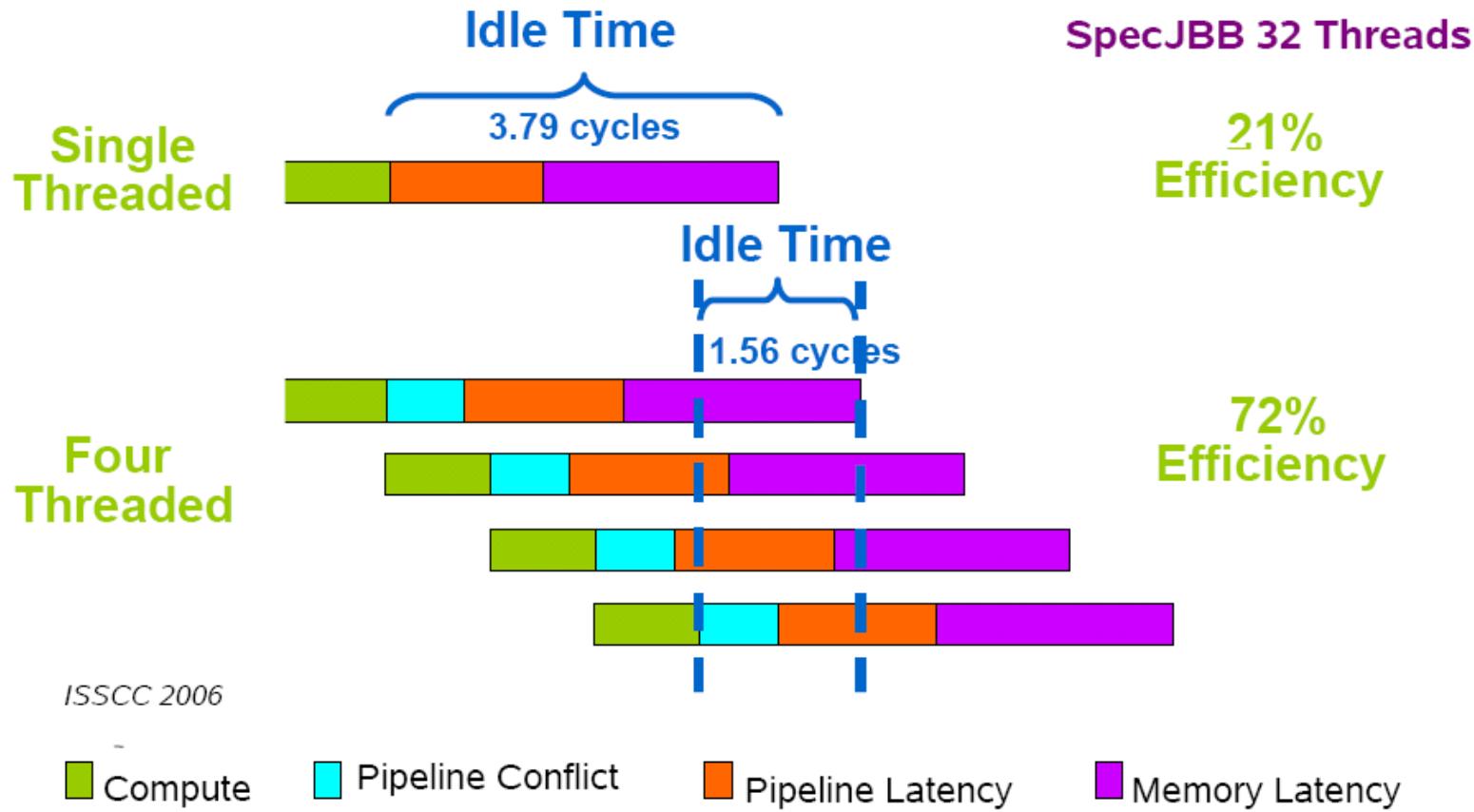
# Simultanes Multi Threading

**SMT erhöht die Zahl der Transistoren um 10%**

(CMP erhöht die Zahl der Transistoren um 50%)



# Simultanes Multi Threading (2)



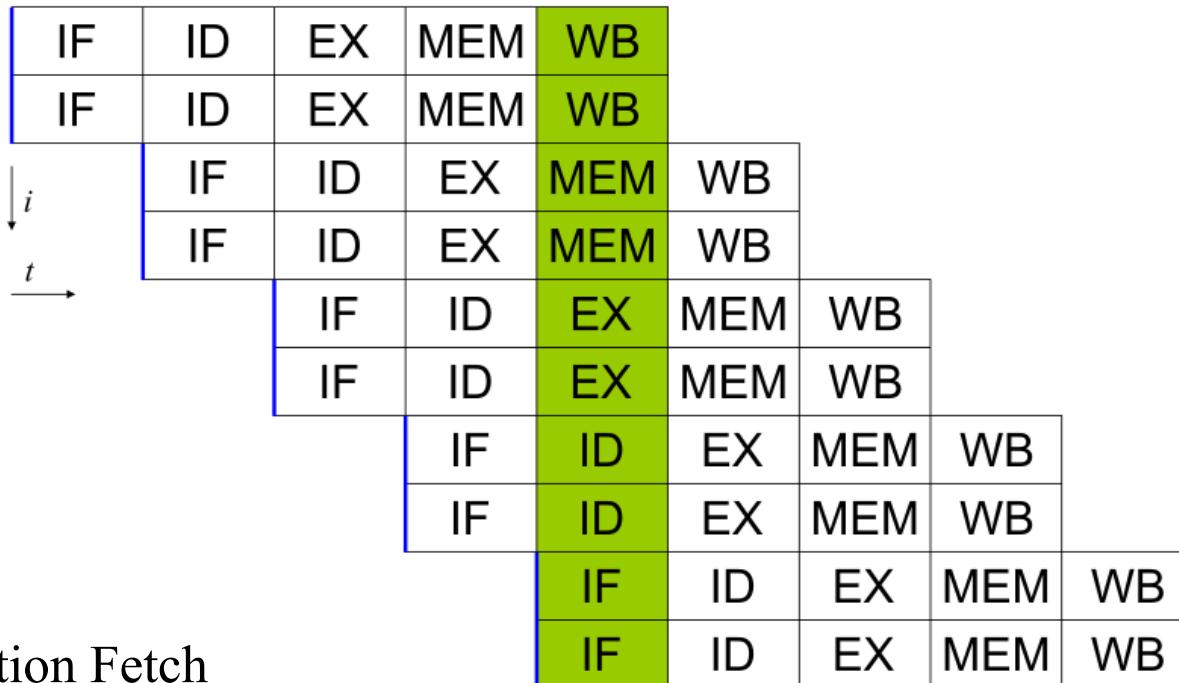
# Superscalar Architecture

- Superscalar auch *Instruction Level Parallel* genannt.
- SC Prozessor erledigt mehr als eine Instruktion während eines Clock-Cycle indem er mehrere Instruktionen auf mehrere redundante Funktionseinheiten im Prozessor legt.
- Eine funktionale Einheit ist nicht eine CPU sondern eine Ausführungsressource innerhalb einer einzigen CPU wie zB: eine „*arithmetic logic unit*“, ein „*bit shifter*“ oder ein „*multiplier*“.

# Superscalar Pipeline

2 fetching und dispatching Instruktionen gleichzeitig.  
Zyklus können 2 Instruktionen bearbeitet werden.

Pro Prozessor-



IF = Instruction Fetch

ID = Instruction Decode

EX = Execute

MEM = Memory access

WB = Register write back

$i$  = Instruction number  
 $t$  = Clock cycle [i.e., time])

# Wie passt SMT ins Bild vom Modul CNA?

Sie erinnern  
sich?

Auszug aus  
Tannenbaum's:  
„Structured  
Computer Organization“  
Kap. 2

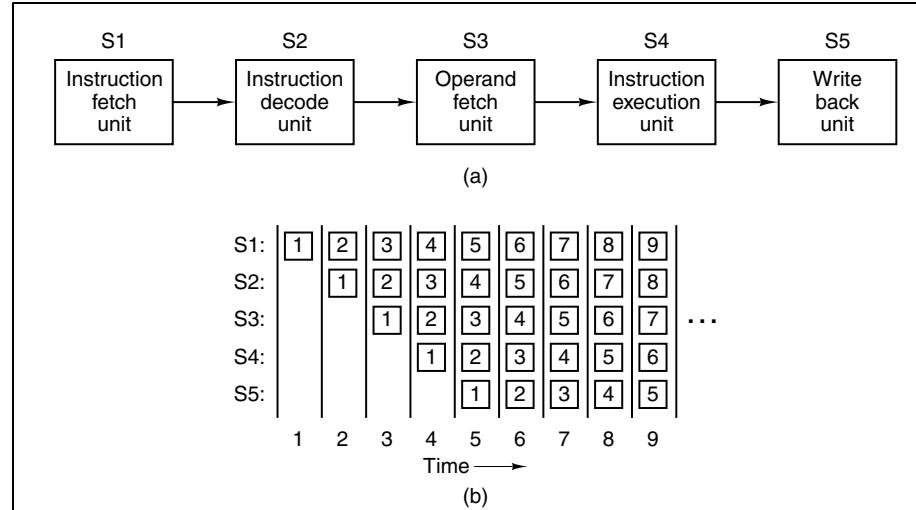


Figure 2-4. (a) A five-stage pipeline. (b) The state of each stage as a function of time. Nine clock cycles are illustrated.

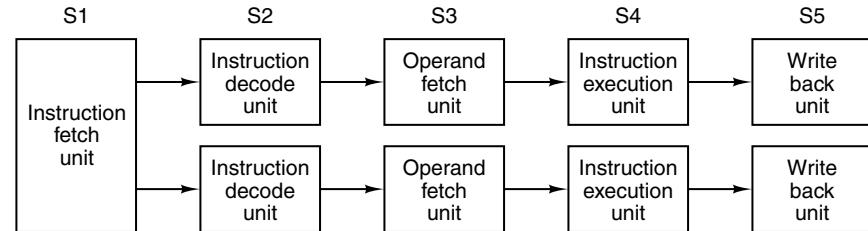
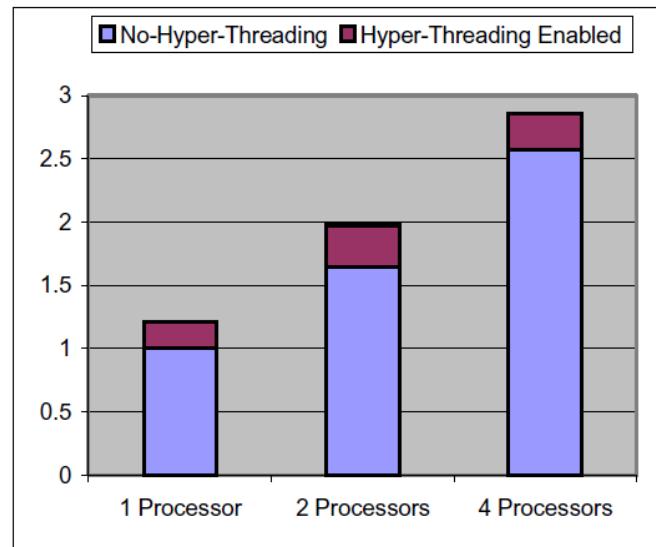


Figure 2-5. Dual five-stage pipelines with a common instruction fetch unit.

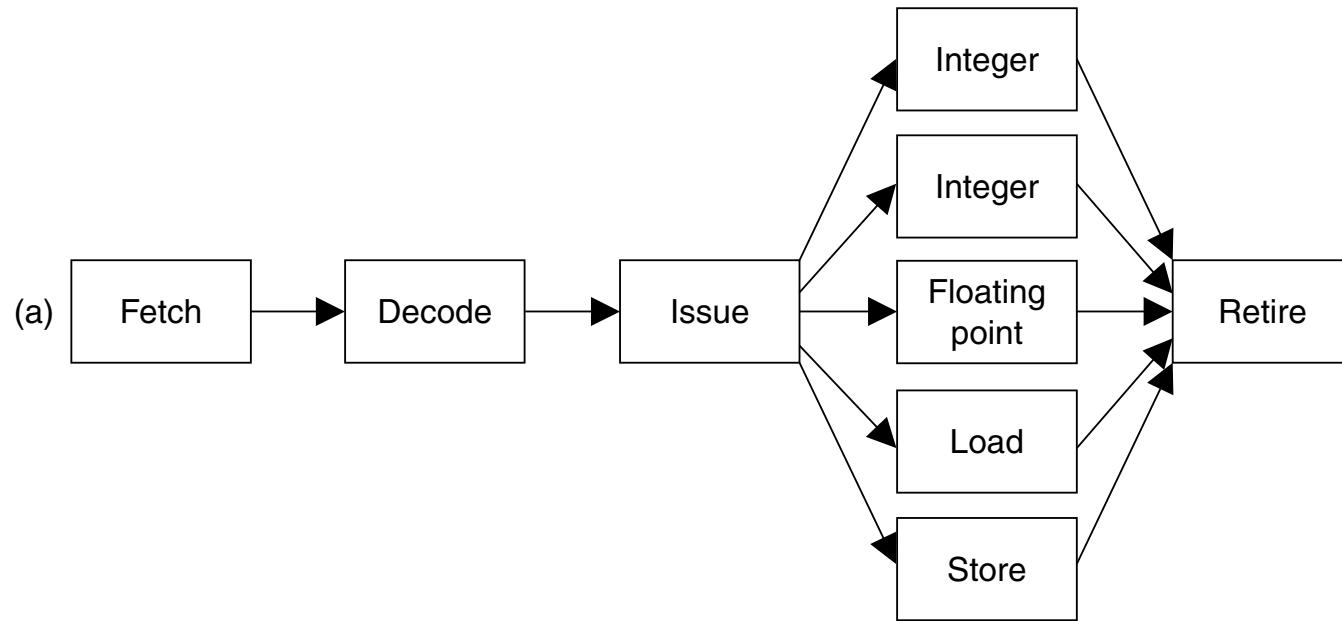
## Hyper-Threading

- ◆ Ein Core wird so erweitert, dass er gleichzeitig 2 Threads bearbeiten kann (sogar 2 Threads von 2 unterschiedlichen Prozessen).
- ◆ Bei neueren Intel-Prozessoren:  
z.B. Core i7 und Xeon:  
2 Threads pro Core:  
4 Cores, 8 Threads
- ◆ ein (komplexes) Steuerwerk
- ◆ nur Teile der Architektur doppelt vorhanden, z.B.:
  - ◆ mehrere vollständige Registersätze
  - ◆ Pipeline (ALU, FPU, ..) nur 1 mal vorhanden



Online transaction processing performance (OLTP), Intel Technology Journal Q1, 2002

# Pipeline for handling VLIW



Auszug aus Tannenbaum's: „Structured Computer Organization“ Kap. 8

# CMT in Oracle's Niagara

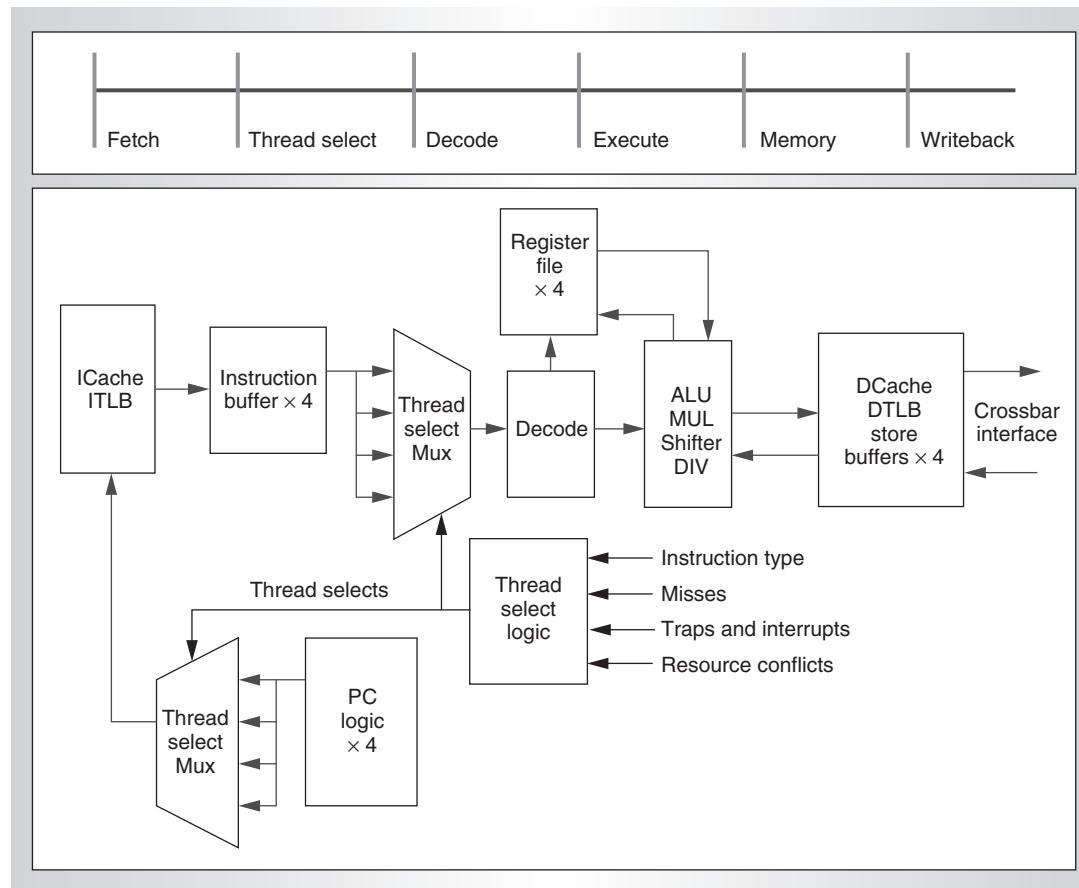


Figure 3. Sparc pipeline block diagram. Four threads share a six-stage single-issue pipeline with local instruction and data caches. Communication with the rest of the machine occurs through the crossbar interface.

Auszug aus Paper von IEEE: „Niagara: a 32-way multithreaded SPARC processor“

# Parallele Rechnerarchitekturen

- Symmetrische Multi Prozessoren
- Multi Core Prozessoren
- Speicherorganisation
- Simultaneous Multi Threading
-  - Leistungsmasse
- Cache Koherenz
- Verbindungsnetzwerke
- Skaliert ihre Applikation?

# Einfache Leistungsmasse

$$V(n) \cdot \frac{T(1)}{T(n)}$$

$$E(n) \cdot \frac{V(n)}{n}$$

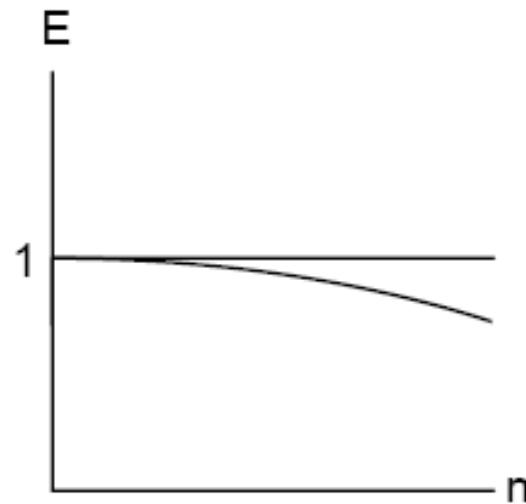
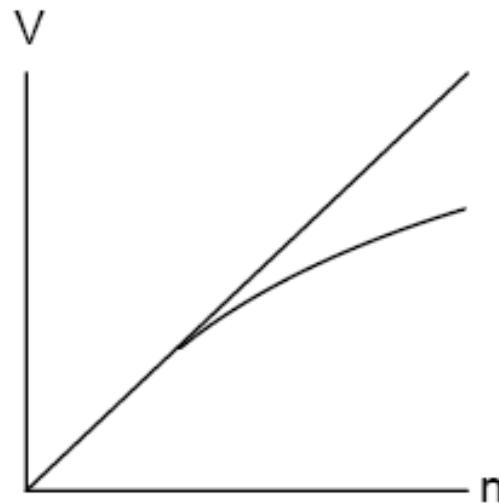
$V(n)$ : Leistungsverbesserung

$E(n)$ : Effizienz

$T(1)$ : Ausführungszeit für einen Prozessor

$T(n)$ : Ausführungszeit für  $n$  Prozessoren

$n$ : Anzahl Prozessoren



# Amdahls Gesetz (Beispiel)

Gesammte Rechenzeit = 1

Nicht Parallelisierter Anteil =  $a = 20\%$

Parallelisierbar =  $1-a = 80\%$

$$T = a + \frac{(1-a)}{n}$$

1.  $n = 1 \quad T = 0.2 + (1-0.2) / 1 = 1$

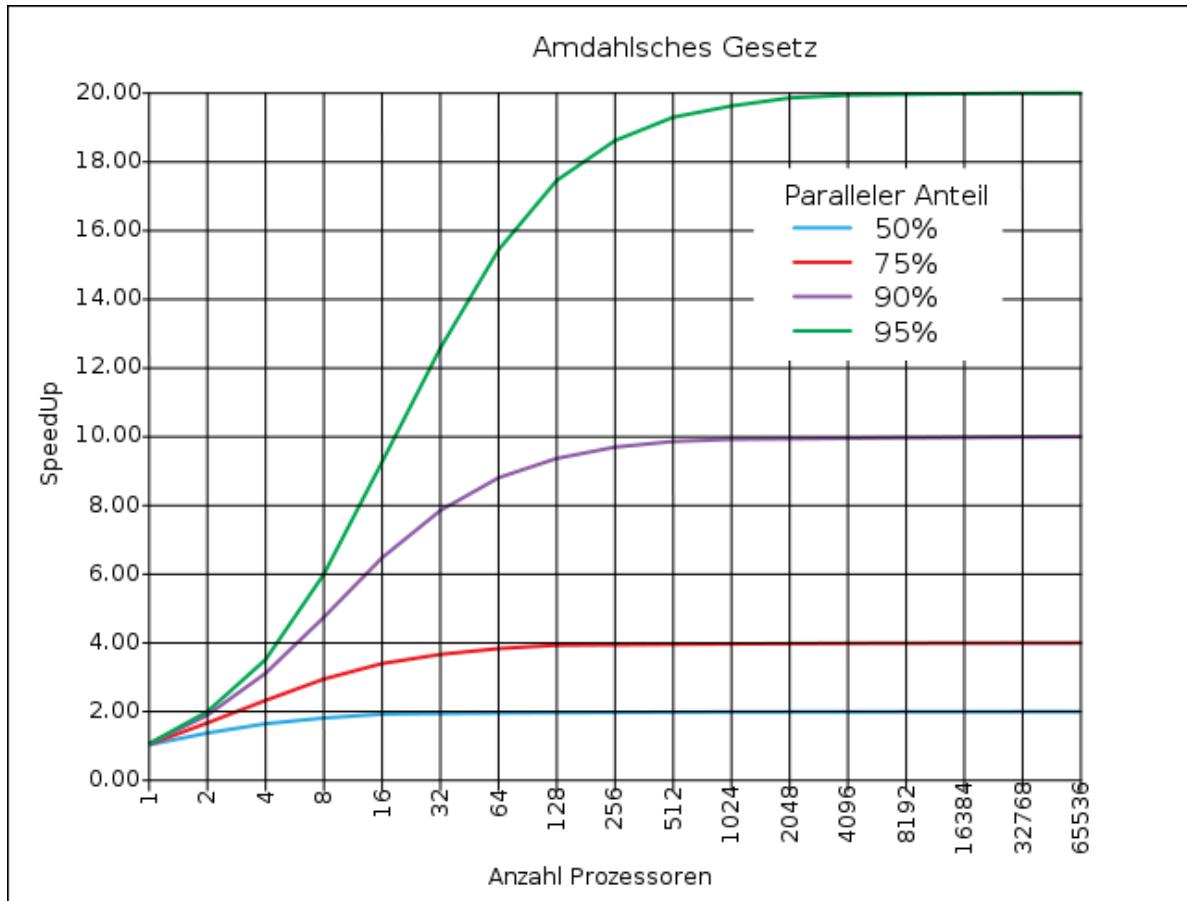
2.  $n = 2 \quad T = 0.2 + (1-0.2) / 2 = 0.6$

3.  $n = 10 \quad T = 0.2 + (1-0.2) / 10 = 0.28$

4.  $n = 100 \quad T = 0.2 + (1-0.2) / 100 = 0.202$

Konvergiert gegen 0.2 das ist der nicht parallele Anteil (20%)

# Amdahl

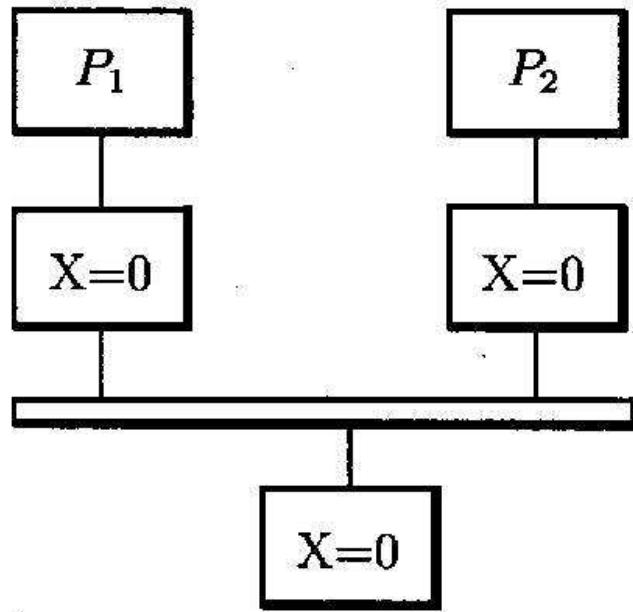


Der Geschwindigkeitsgewinn bei der Verwendung von parallel arbeitenden Prozessoren bei der Bearbeitung eines parallelen Problems (Wikipedia)

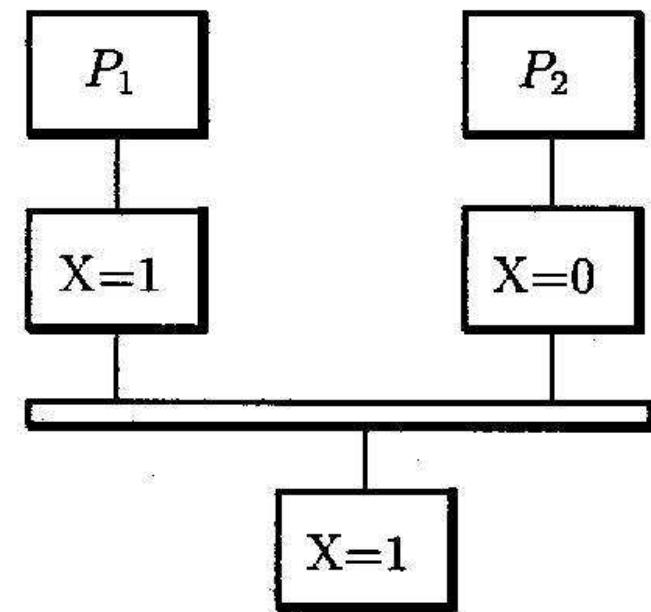
# Parallele Rechnerarchitekturen

- Symmetrische Multi Prozessoren
- Multi Core Prozessoren
- Speicherorganisation
- Simultaneous Multi Threading
- Leistungsmasse
-  - Cache Koherenz
- Verbindungsnetzwerke
- Skaliert ihre Applikation?

# Cachekohärenz (Beispiel 1: write through cache)

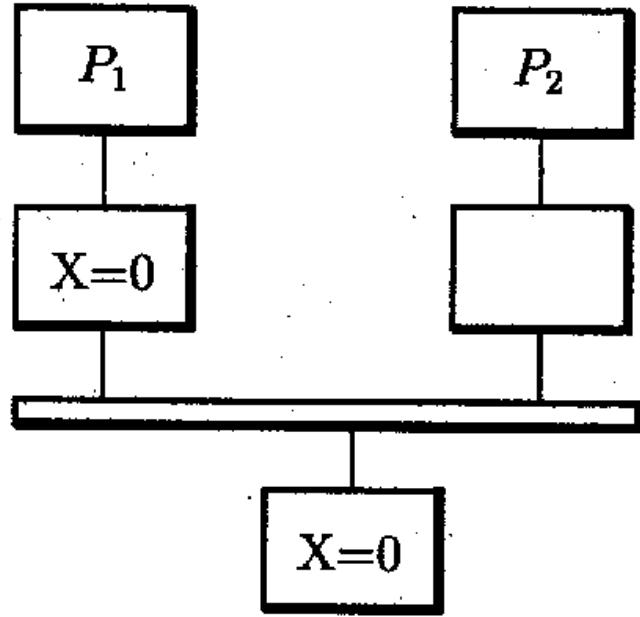


Die Variable X befindet sich in den Caches von P1 und P2 sowie im Hauptspeicher

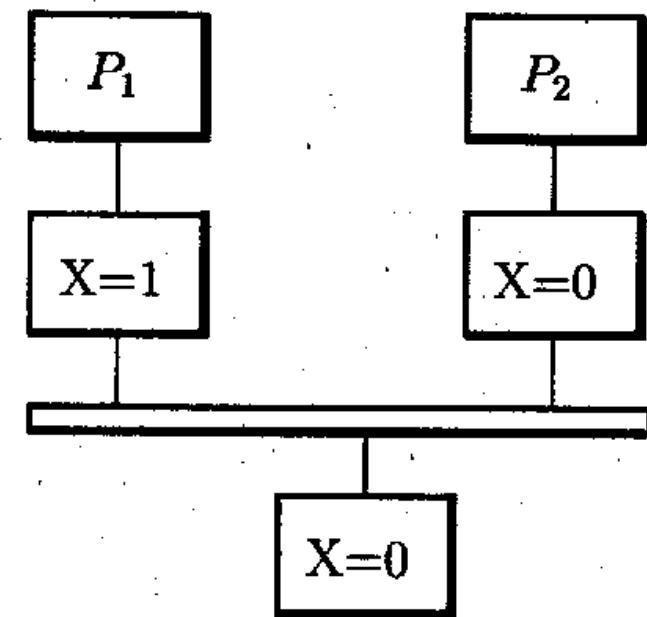


P1 schreibt X=1,  
Der Hauptspeicher wird beim **write through Cache** ebenfalls verändert. P2 liest den alten (inkohärenten) Wert aus dem eigenen Cache.

## Cachekohärenz (Beispiel 2: write back cache)



Die Variable X befindet sich im Cache von P1 und im Hauptspeicher



P1 schreibt X=1,  
Der Hauptspeicher wird nicht sofort verändert (**write back cache**). P2 liest nun den alten, inkohärenten Wert aus dem Hauptspeicher.

# Lösung Cachekohärenz Probleme in Distributed Shared-Memory Architekturen

Cachekehärenzprotokolle:

Es werden Snoopy und Directory-Protokolle unterschieden.

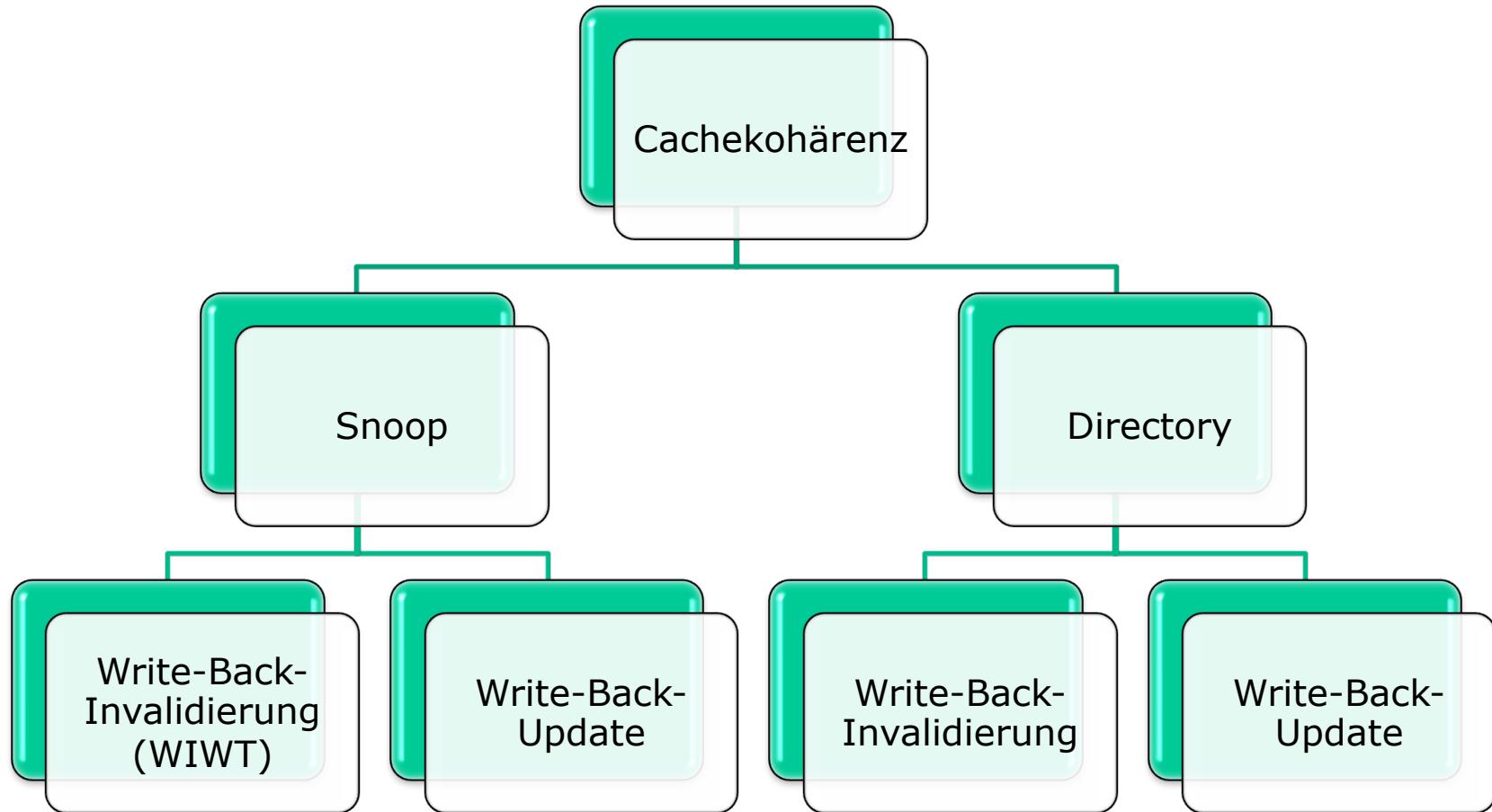
Verwendung eines gemeinsamen Caches:

Grosser Hardwareaufwand, Sequentialisierung der Speicherzugriffe.

Unterteilung der Daten:

Für Daten, auf die mehr als nur ein Prozessor zugreift, existiert jeweils nur eine Kopie. Die Datenunterteilung übernimmt der Compiler oder der Programmierer. Effizienzverlust aufgrund konservativer Annahmen.

# Protokolle (Algorithmen)



# Cachekohärenz - Protokolle

## Snoopingbasiert (Snoop)

Da alle Zugriffe über gleiches Medium verlaufen (Bus, Switch) können die Cachecontroller beobachten und erkennen welche Blöcke sie selbst gespeichert haben.

## Verzeichnisbasiert (Directory)

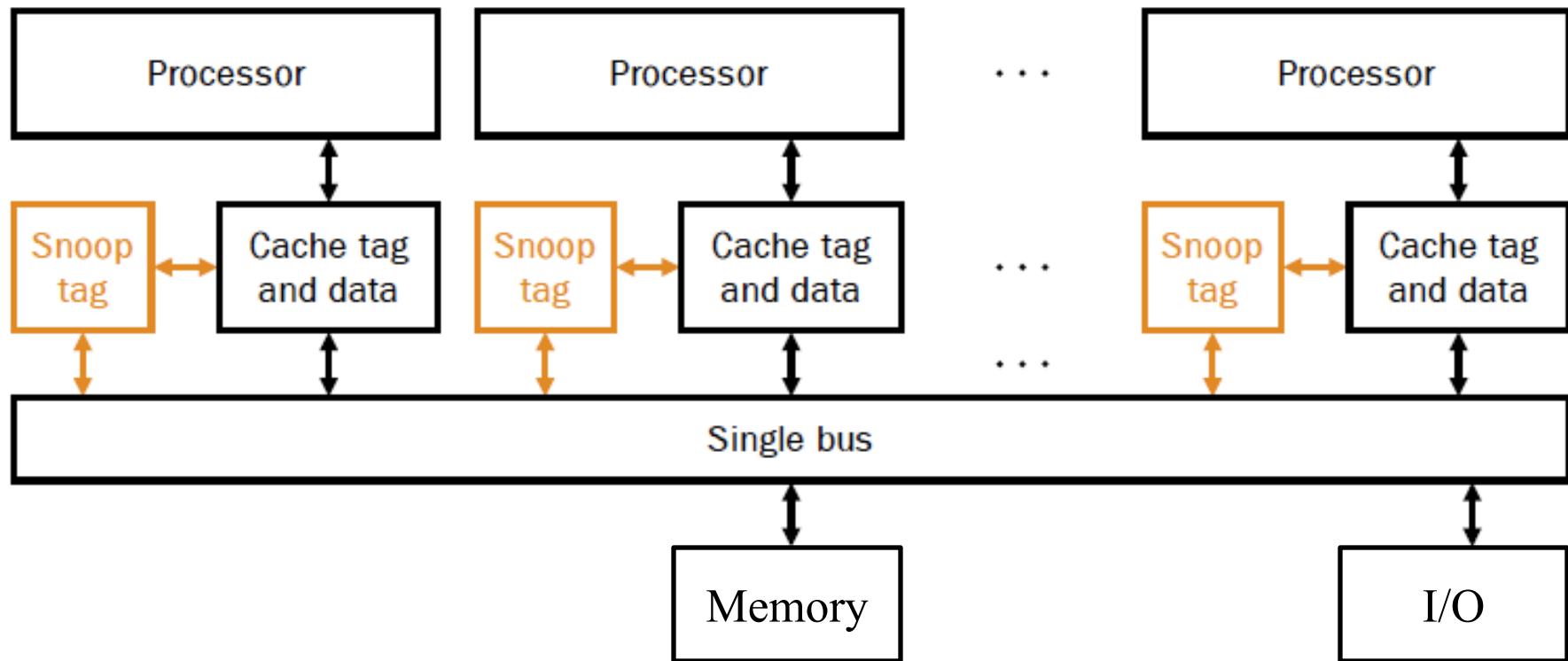
Zentrale Liste mit Status aller Cache Blöcken. Welcher Prozessor hat was?

*Read-Only = Shared,    exclusive write = Exclusive)*

Ab 64 Prozessoren oder Cores werden ausschliesslich Directory basierende Protokolle eingesetzt, da die Bandbreite des Busses nicht ausreichend skaliert

# Cachekohärenz: Snoopy

Snoopy Caches hören auf den Verbindungsbus mit um alle Schreiboperationen auf Cache Werte die sie auch selber halten zu detektieren



## Cachekohärenz: Snoopy (2)

- Erweiterung der Status-Bits jeder Cachezeile.
- Zusätzliche Cache-Controller, die das jeweilige Cachekohärenz-Protokoll implementieren.
- Zur Vermeidung von Zugriffskonflikten zwischen CPUs werden Adressen-Tags und Status-Bits dupliziert (Snoop tag).
- Protokolle werden üblicherweise durch endliche Automaten dargestellt. Die Zustände sind den Cachezeilen zugeordnet und repräsentieren die aktuelle Situation.
- Klassifikation von Protokollen: Write-Invalidate / Write-Update für Write-Through / Write-Back Caches.

# Cachekohärenz: Directory (1)

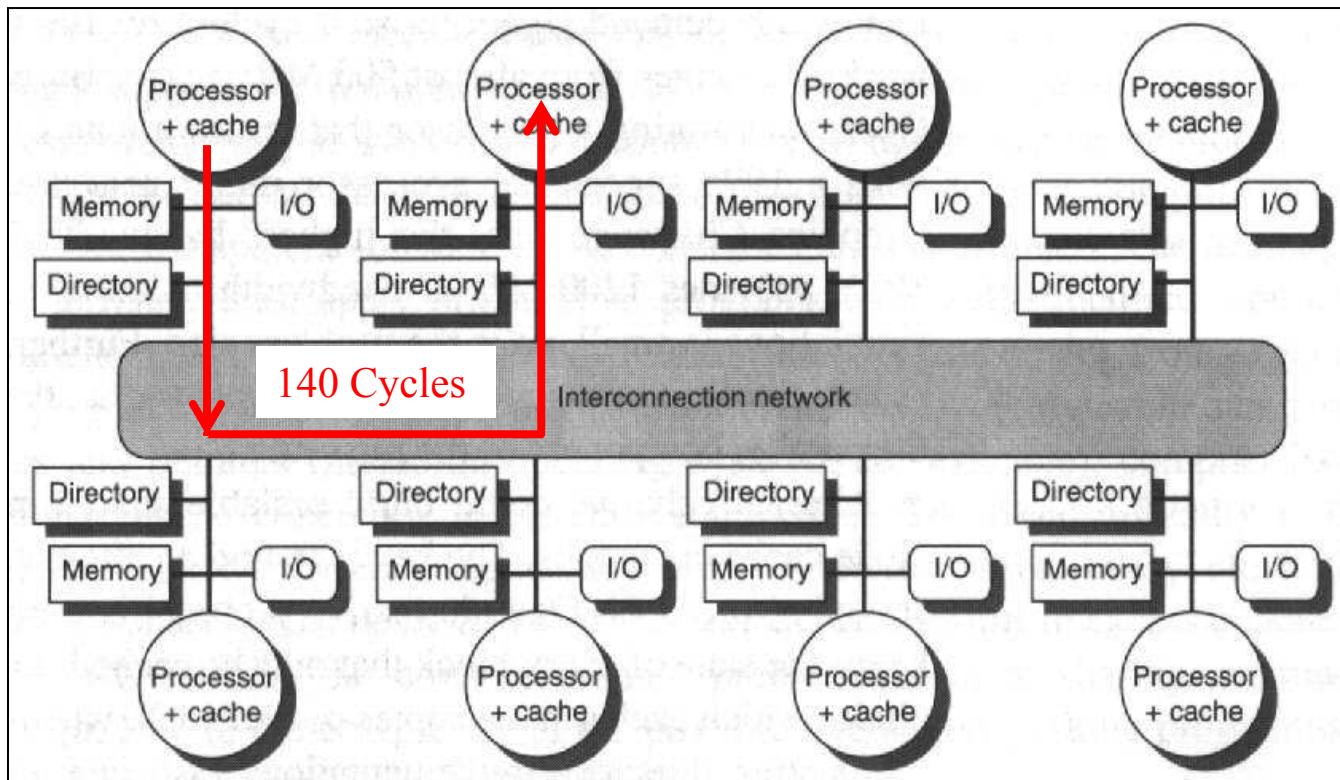
Snoopy Protokolle sind aufwendig aufgrund der notwendigen globalen Kommunikation.

Alternative: Directory Protokoll (Eigenschaften):

- Directories enthalten den Zustand von Speicherblöcken (welche Caches haben Kopien, welche sind dirty, etc ...).
- Übliche Zustände:
  - *Shared* (*Block ist in einem oder mehreren Caches, Wert ist up-to-date*),
  - *Uncached* (*Block ist in keinem Cache*),
  - *Exclusive* (*Block gehört einem Cache, Wert ist nicht up-to-date*).
- Ein Bitvektor zeigt zudem an, in welchen Caches der jeweilige Block gespeichert ist.

# Cachekohärenz: Directory (2)

- Pro Speicherblock existiert ein Eintrag in einem Directory.
- *Verteilte Directories:* Zuordnung zum jeweiligen Hauptspeicher. Enthält nur Informationen über Blöcke in diesem Hauptspeicher.

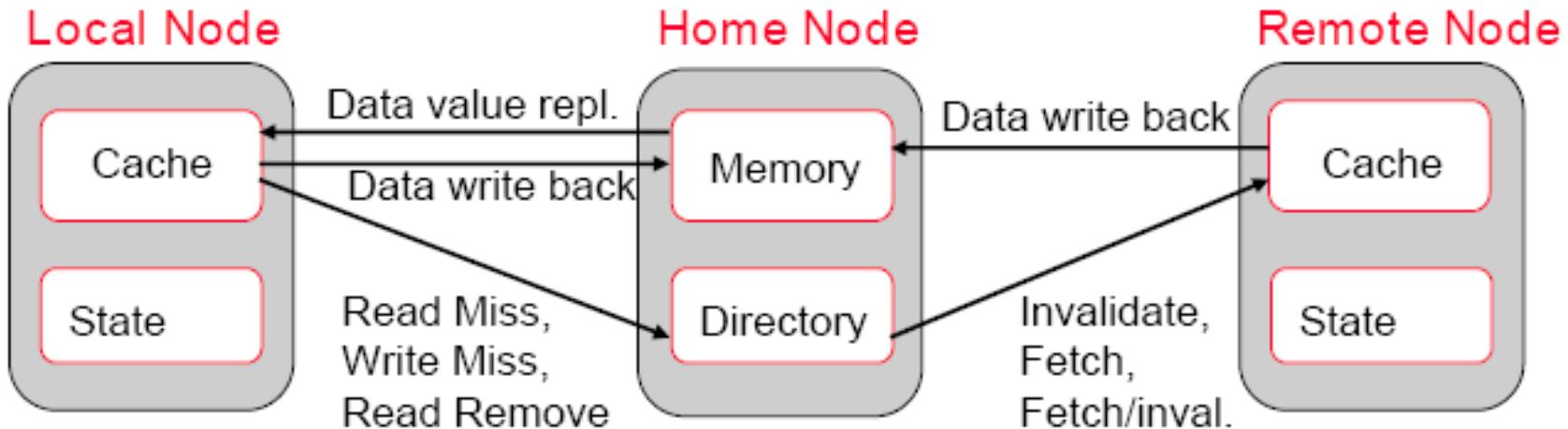


# Cachekohärenz: Directory (3)

Beteiligte Einheiten:

- **Local Node:** Ursache der gesamten Transaktion.
- **Home Node:** Hauptspeicher und Directory, in denen die Adresse der Schreib- oder Leseanforderung liegt.
- **Remote Node:** Knoten, der eine Kopie des jeweiligen Cacheblocks besitzt.

Nachrichten:

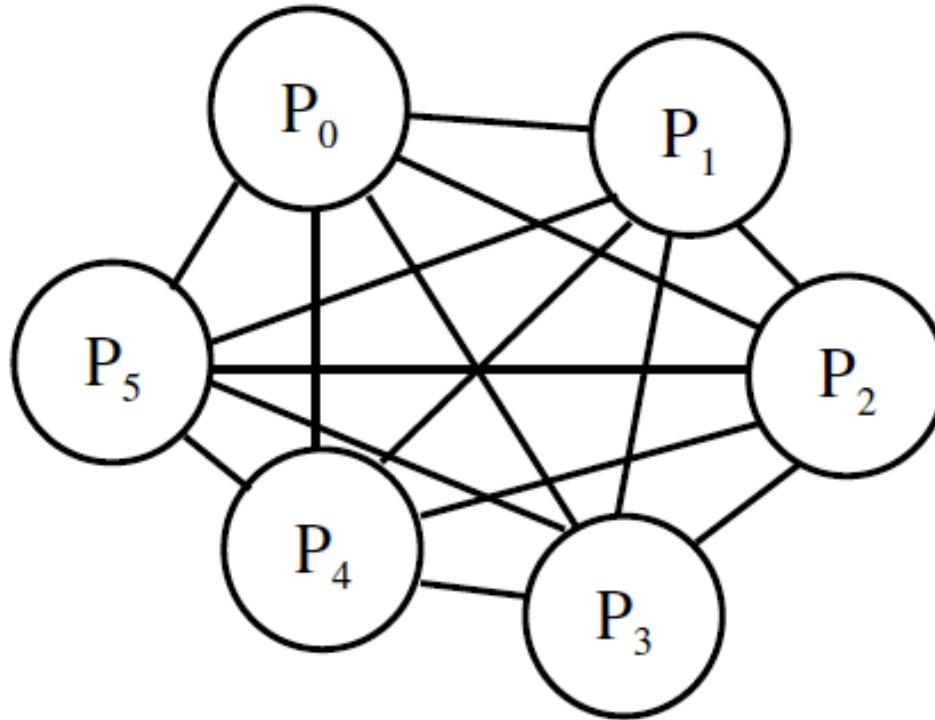


# Parallele Rechnerarchitekturen

- Symmetrische Multi Prozessoren
- Multi Core Prozessoren
- Speicherorganisation
- Simultaneous Multi Threading
- Leistungsmasse
- Cache Koherenz
-  - Verbindungsnetzwerke
- Skaliert ihre Applikation?

# Vernetzung der Prozessoren

- 1000 Prozessoren,  $10^6$  Verbindungen, jede z.B. 64 Bit breit also  $6,4 * 10^7$  Leitungen.
- Bei einer 100-lagigen Platine wäre das eine Platinenbreite von ca. 6 km.

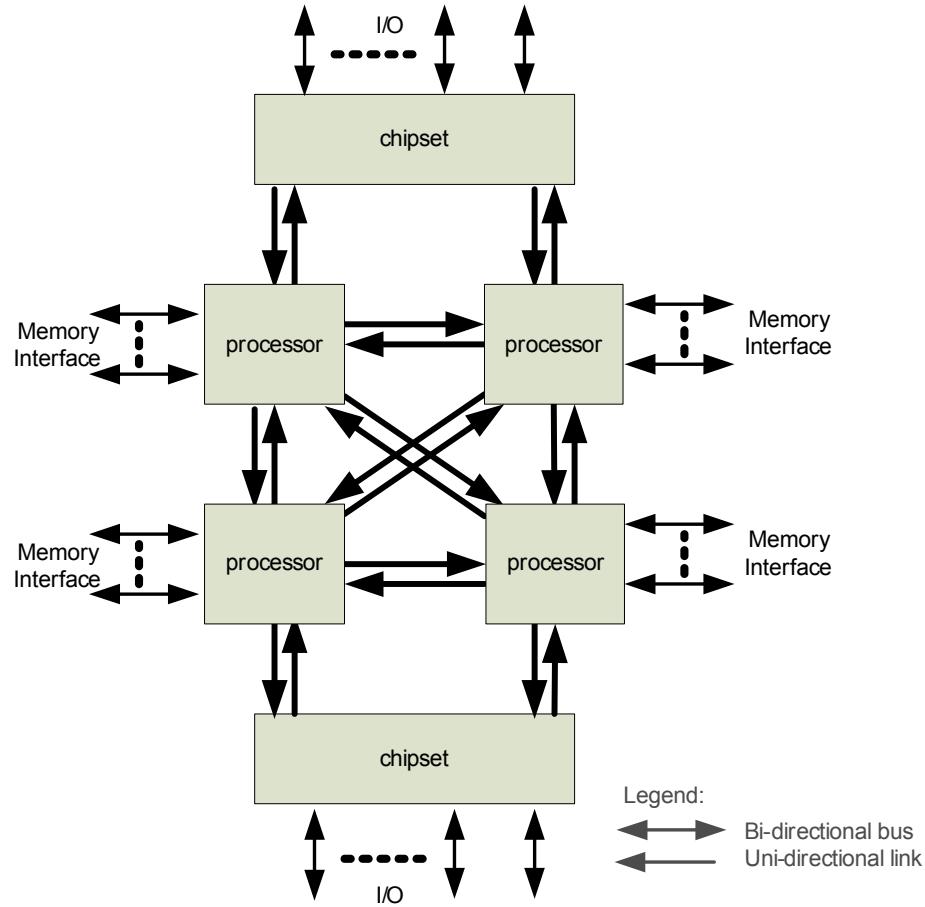


# Prozessor Interconnect

Uni direktionale Verbindungen (QPI, quick path interface) zwischen den Prozessoren erlauben Intel viel grössere Geschwindigkeiten.

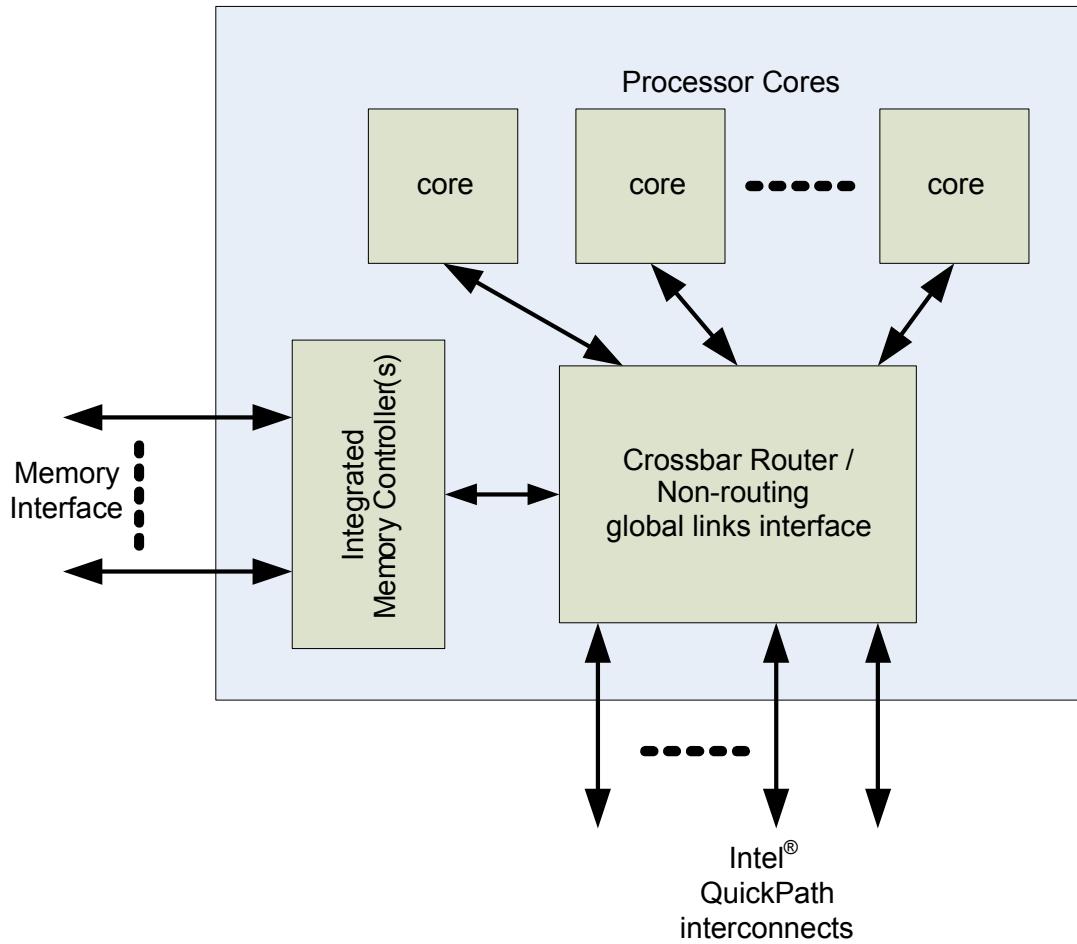
**4 Sockets** (Prozessoren) verlangen bei einer „jeder zu jedem“ Verbindung „nur“ 12 Verbindungen. Das ist noch machbar.

**8 Sockets** (Prozessoren) sind das aber schon 56 Verbindungen. Das ist sehr aufwändig in Anbetracht, dass die Verbindungsbreite mit 64 Bit noch um einiges komplizierter wird.



# Quick Path Interconnects

Ein Crossbar Switch optimiert Aufwand und Ertrag der Core zu Core Verbindungen.



# Verbindungsnetzwerke

## Entwurfskriterien

- hohe Leistung -> viele Leitungen
- niedrige Kosten -> wenig Leitungen

## Klassifikationskriterien

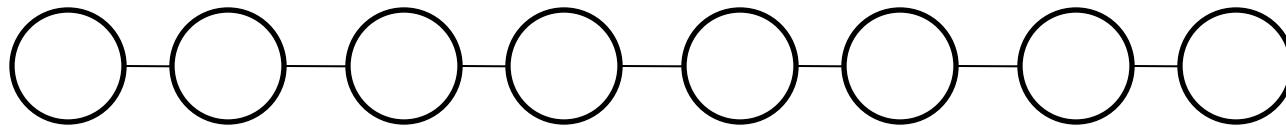
- Topologie
  - Wie sind die Prozessoren miteinander verbunden (gemeinsames Medium vs. geschaltete Verbindung) ?
  - Statisch oder dynamisch veränderbar (Verbindungen invariant vs. auf Anforderung etabliert).
- Routing
  - Wie werden die Nachrichten verteilt (store and forward, wormhole, cut-trough, circuit switching, packet switching) ?
  - Welcher Algorithmus zur Pfadbestimmung wird verwendet (deterministisch, off-line, adaptiv, randomized ) ?

# Topologie Bewertungskriterien

- **Inzident:** Knoten ist inzident mit einer Kante wenn der Knoten wenigstens an einem Ende der Kante liegt.
- **Durchmesser D:** Maximum der Anzahl der Kanten im kürzesten Pfad zwischen zwei beliebigen Knoten des Netzwerks.
- **Grad k des Netzwerks:** Maximale Grad (Zahl der inzidenten Kanten) der Netzwerk-Knoten.
- **Halbierungsbandbreite (Bisektionsbreite):** Wird ein Netzwerk in zwei etwa gleich grosse Teile getrennt, so ist die Halbierungsbandbreite die minimale Anzahl an Verbindungsleitungen, die dabei durch- trennt werden müssen. Eine hohe *Bisektionsbreite* verhindert Flaschenhälse.
- **Symmetrie:** Die Knoten und/oder Links in einem symmetrischen Netz verhalten sich gleich, egal welchen Knoten oder welchen Link man betrachtet.
- **Skalierbarkeit:** *Ein Verbindungsnetzwerk ist skalierbar, wenn* einer Erweiterung des Netzwerks keine Beschränkungen entgegen stehen.
- **Konnektivität:** Minimale Anzahl Knoten oder Links die durchtrennt werden müssen, damit das Netz nicht mehr funktionstüchtig ist.

# Lineares Feld

Das lineare Feld ist das einfachste Verbindungsnetzwerk



Aufgrund der besonderen Stellung der Endknoten ist das lineare Feld asymmetrisch.

Grad der Knoten:

2 für alle inneren Knoten

1 für Randknoten

Durchmesser:

$N-1$

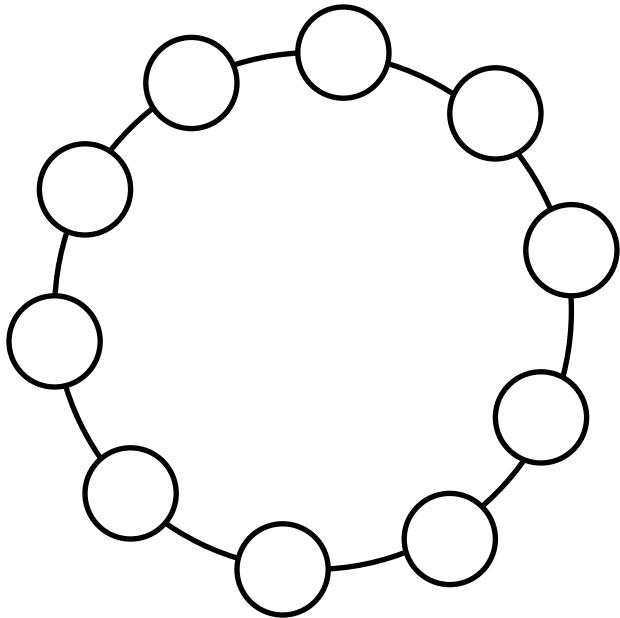
Halbierungsbandbreite:

1

Skalierbarkeit:

keine Beschränkungen für die Skalierbarkeit.

# Ring



Der Ring ist ein asymmetrisches Netzwerk. Im Gegensatz zum linearen Feld können die Verbindungen im Ring uni- oder bidirektional ausgelegt sein. Wir gehen von bidirektionalen Verbindungen aus.

Grad der Knoten:

2

Durchmesser:

$N/2$  im bidirektionalen Fall

$N-1$  im unidirektionalen Fall

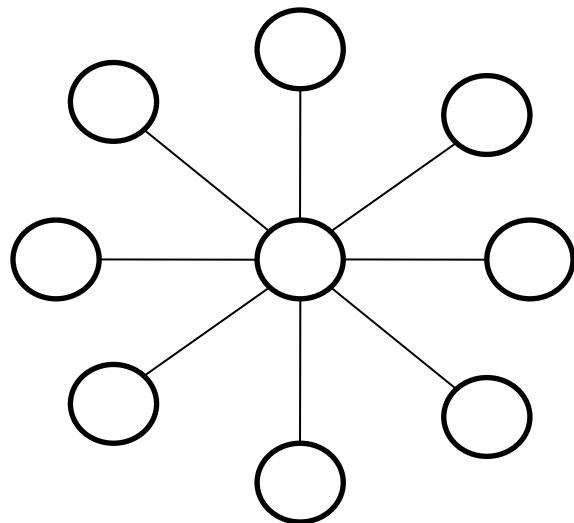
Halbierungsbandbreite:

2

Skalierbarkeit:

keine Hardwarebeschränkungen für Skalierbarkeit.

# Stern



Der Stern ist ein asymmetrisches Netzwerk mit einem herausragenden Zentrumsknoten

Grad der Knoten:

N-1 für Zentrumsknoten

1 für alle anderen Knoten

2

N/2

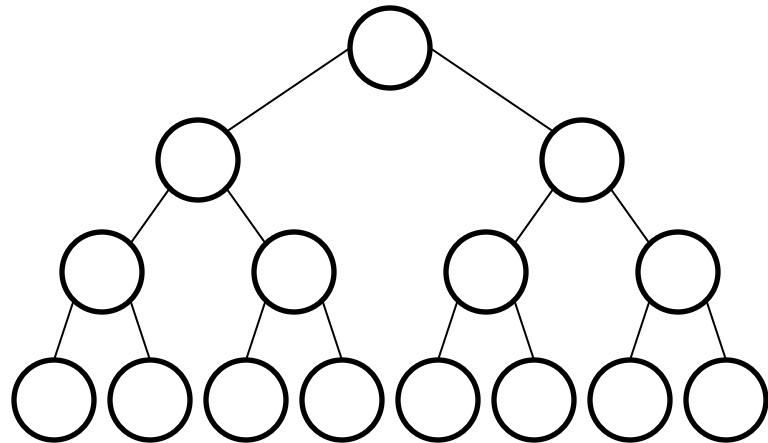
Durchmesser:

Halbierungsbandbreite:

Skalierbarkeit:

Da der Grad des Zentrumsknotens linear mit der Grösse des Netzwerkes steigt, ist der Stern nur für kleine Systeme geeignet.

# Baum



Ein Baum ist ein asymmetrisches Netzwerk. Als Beispiel wird ein ausgewogener binärer Baum betrachtet.

Grad der Knoten:

1 für alle Endknoten (Blätter)

2 für Wurzelknoten

3 für innere Knoten

Durchmesser:

$2 * \lceil \lg(N) \rceil$

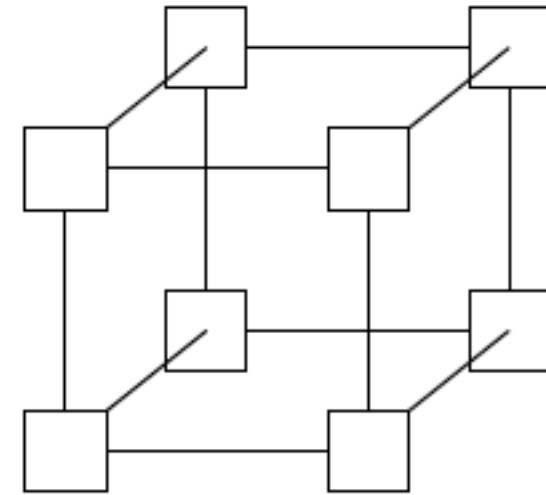
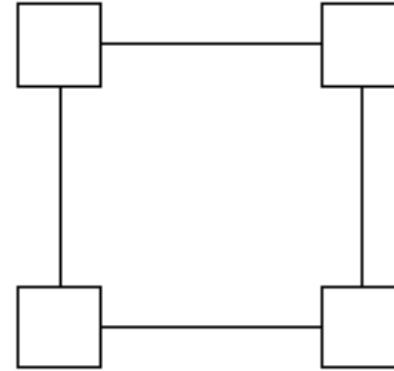
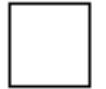
Halbierungsbandbreite:

1

Skalierbarkeit:

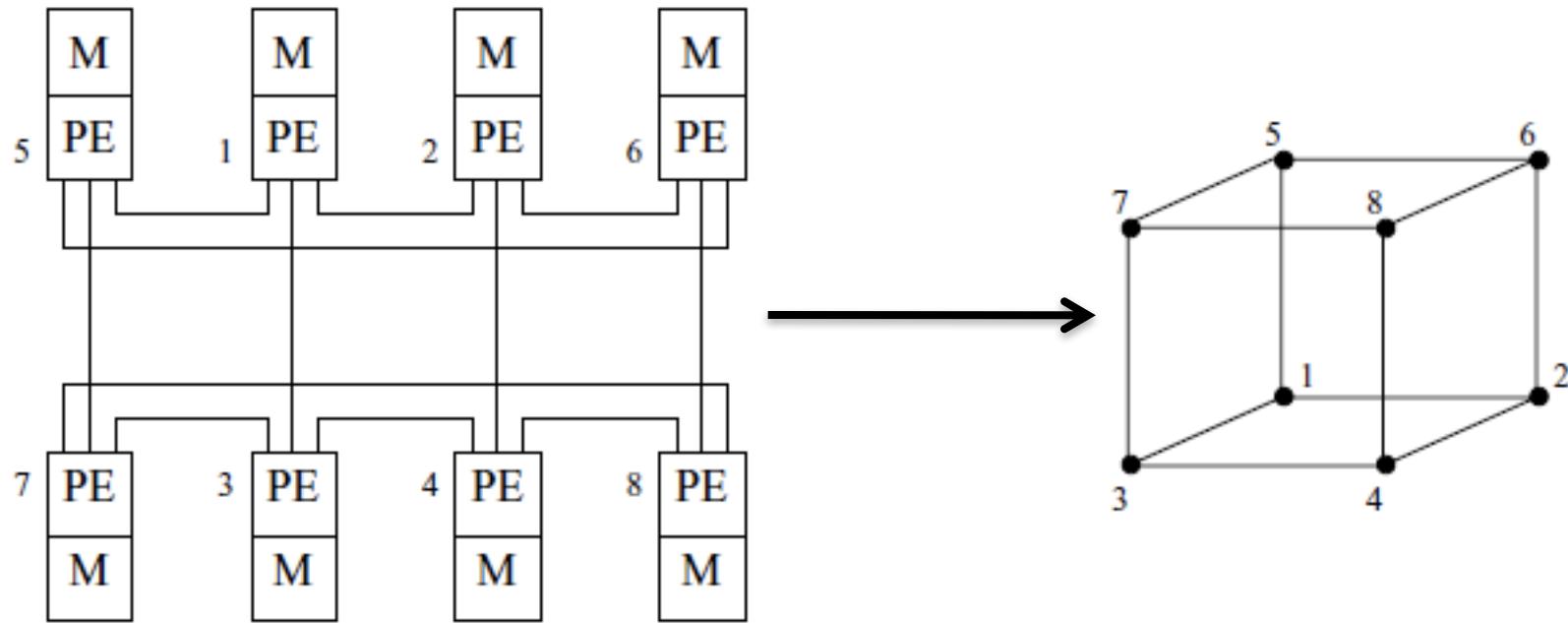
Da der Grad der Knoten konstant ist, stehen der Skalierbarkeit keine Hardwareprobleme entgegen.

# Hypercube



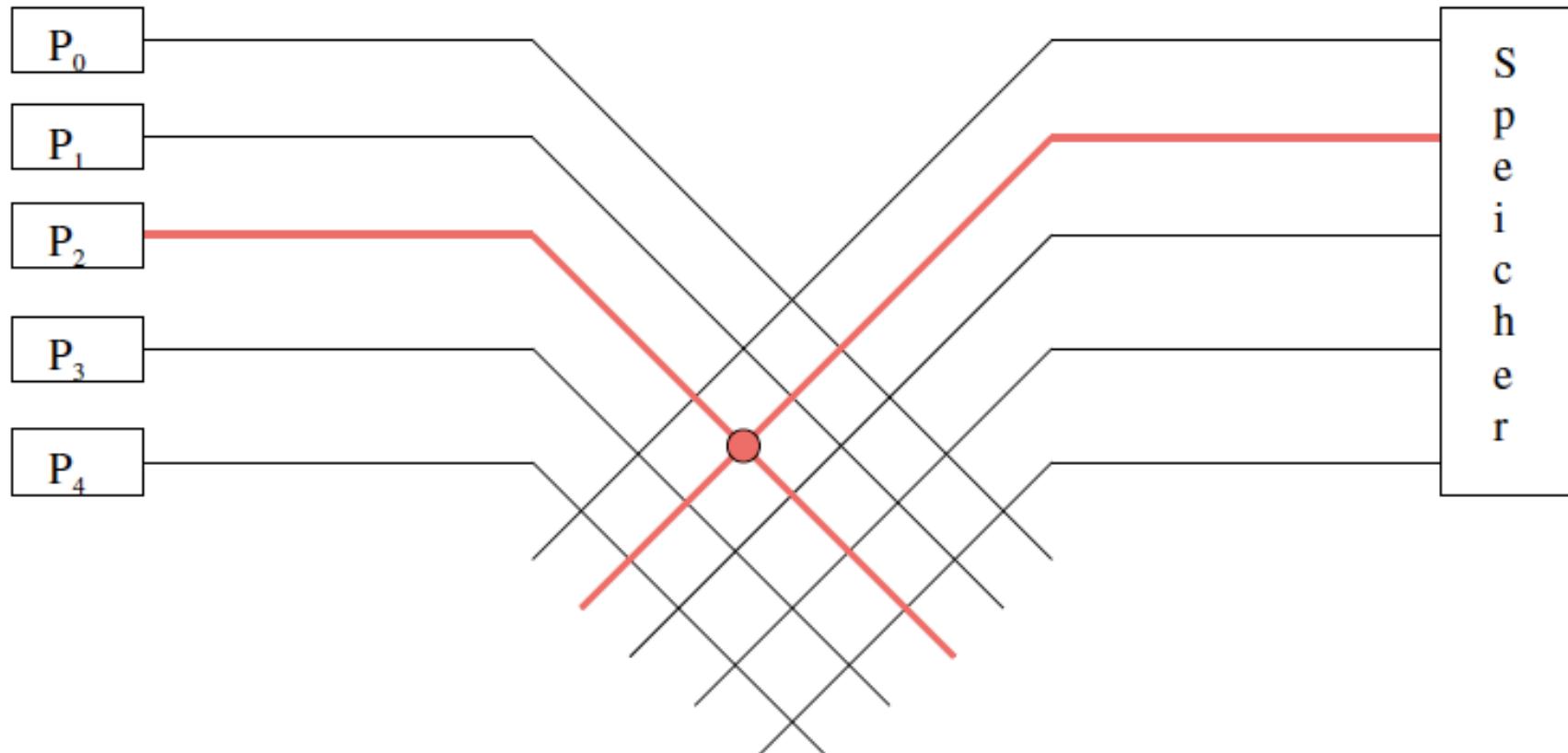
Grad der Knoten:	$N$
Durchmesser:	$\lg(N)$
Halbierungsbandbreite:	$N/2$
Skalierbarkeit:	mässig, wird bei Systolic Array verwendet.

# Layout eines Hypercubes



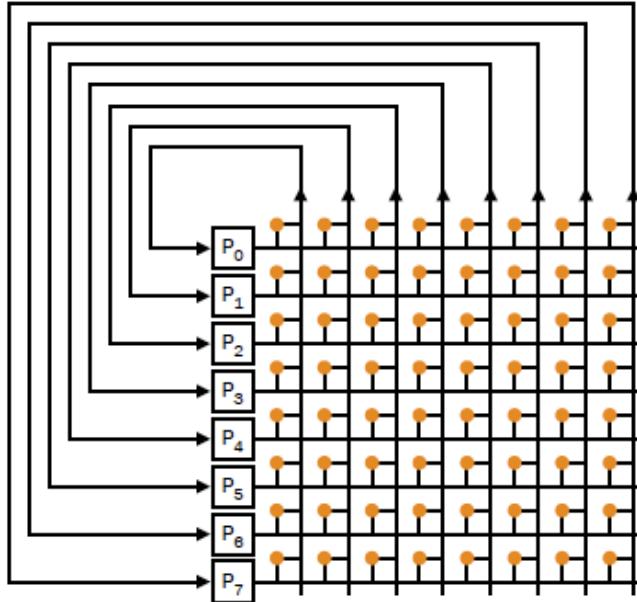
# Koppelnetze

Ein anderer Ansatz...

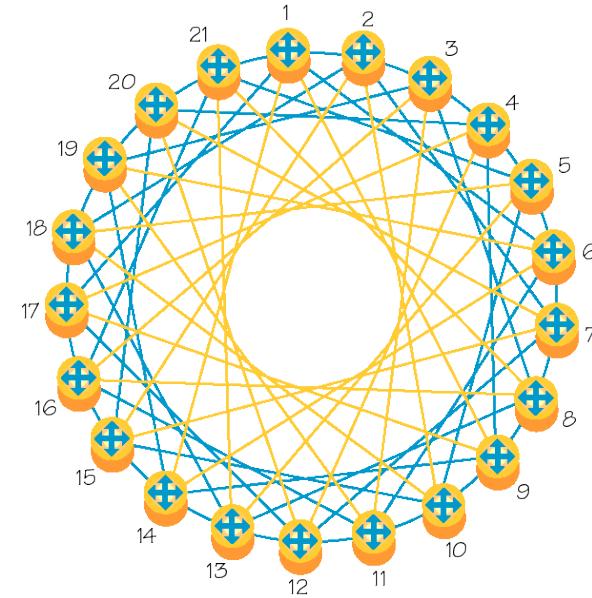


# Koppelnetze (2)

## Crossbar Switch



8 x 8 Crossbar Switch



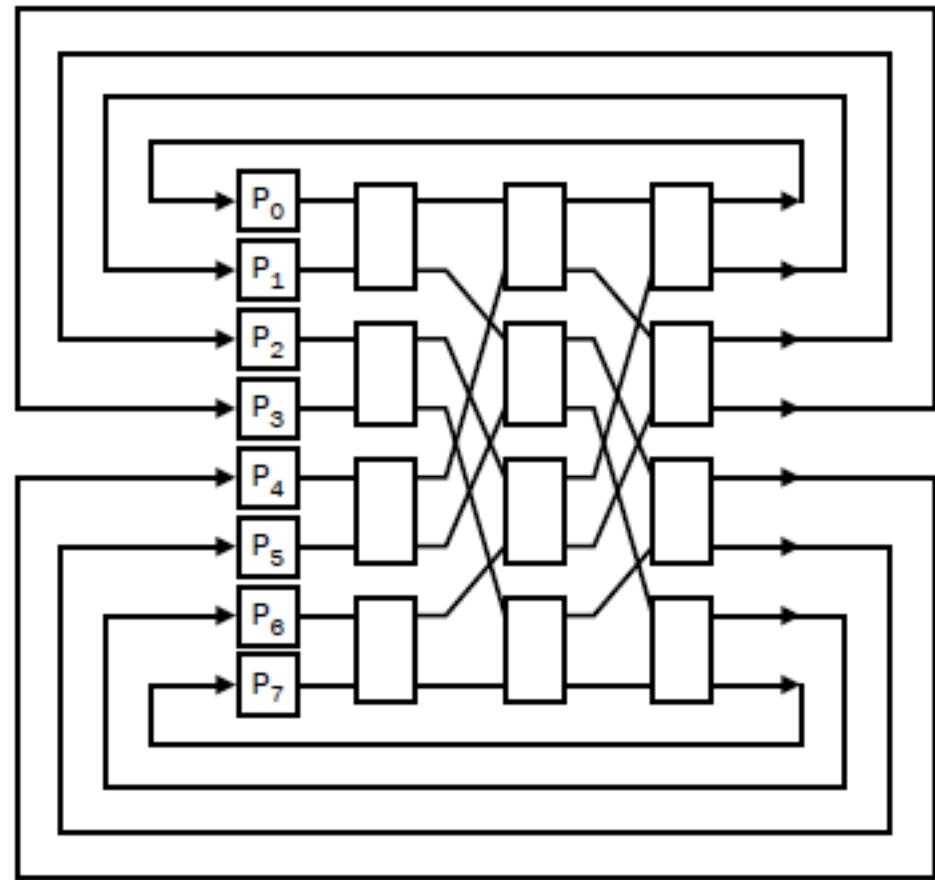
Grad der Knoten: z.B.: 3 Ringen = 2R

Jeder Punkt ist mit jedem Punkt (oder auch Port genannt) verbunden über eine konstante Anzahl Hops (Weg über einen anderen Knoten). Crossbar Switches sind „nicht blockierend“

# Koppelnetze (3)

## Omega Netzwerk

Omega Netzwerke sind  
blockierend



# Parallele Rechnerarchitekturen

- Symmetrische Multi Prozessoren
- Multi Core Prozessoren
- Speicherorganisation
- Simultaneous Multi Threading
- Leistungsmasse
- Cache Koherenz
- Verbindungsnetzwerke
-  - Skaliert ihre Applikation?

# Wie bilde ich skalierbare Applikationen?

- Skalierbare Algorithmen
- Locking – verwenden von fein granularen Locking Mechanismen pro Objekt
- Verwenden von Worker Thread Pools
- Ausführen von Skalier-Tests
- Beobachten der skalierbaren Applikation
- Identifizieren der Orte wo wait-time involviert sind
- Identifizieren von heissen synchronization locks
- Identifizieren der nicht skalierbaren Algorithmen

Lucerne University of  
Applied Sciences and Arts

**HOCHSCHULE  
LUZERN**

Engineering & Architecture



Questions?

# Referenzen

- Computer Architecture – a Quantitative Approach, Hennessy & Patterson, Morgan Kaufmann, ISBN: 1-558860-188-0
- <http://queue.acm.org/detail.cfm?id=2513149>
- William Stallings, OS Internal and Design Principles”, 8<sup>th</sup> edition
- [https://computing.llnl.gov/tutorials/parallel\\_comp/](https://computing.llnl.gov/tutorials/parallel_comp/)