Parallel Computing I

Einführung in das
Hochleistungsrechnen

Parallele Rechnerarchitekturen

3. Mai 2012
Paralleles Rechnen
SS 2012
Thorsten Grahs

Parallele Rechnerarchitekturen

- Harvard/Princeton-Architekturen
- von-Neumann-Flaschenhals
- CISC/RISC-Rechner
- Fließbandverarbeitung (pipelining)
- Flynns Klassifizierung von Parallelrechnern
- Speicherorganisation (distributed/shared)
- Verbindungsnetzwerke/Topologien

Die von–Neumann–Architektur I

John von Neumann (1903 – 1957), Mathematiker

- Princeton, Institute for Advanced Studies
- **E**lectronic **N**umerical **I**ntegrator and **C**omputer

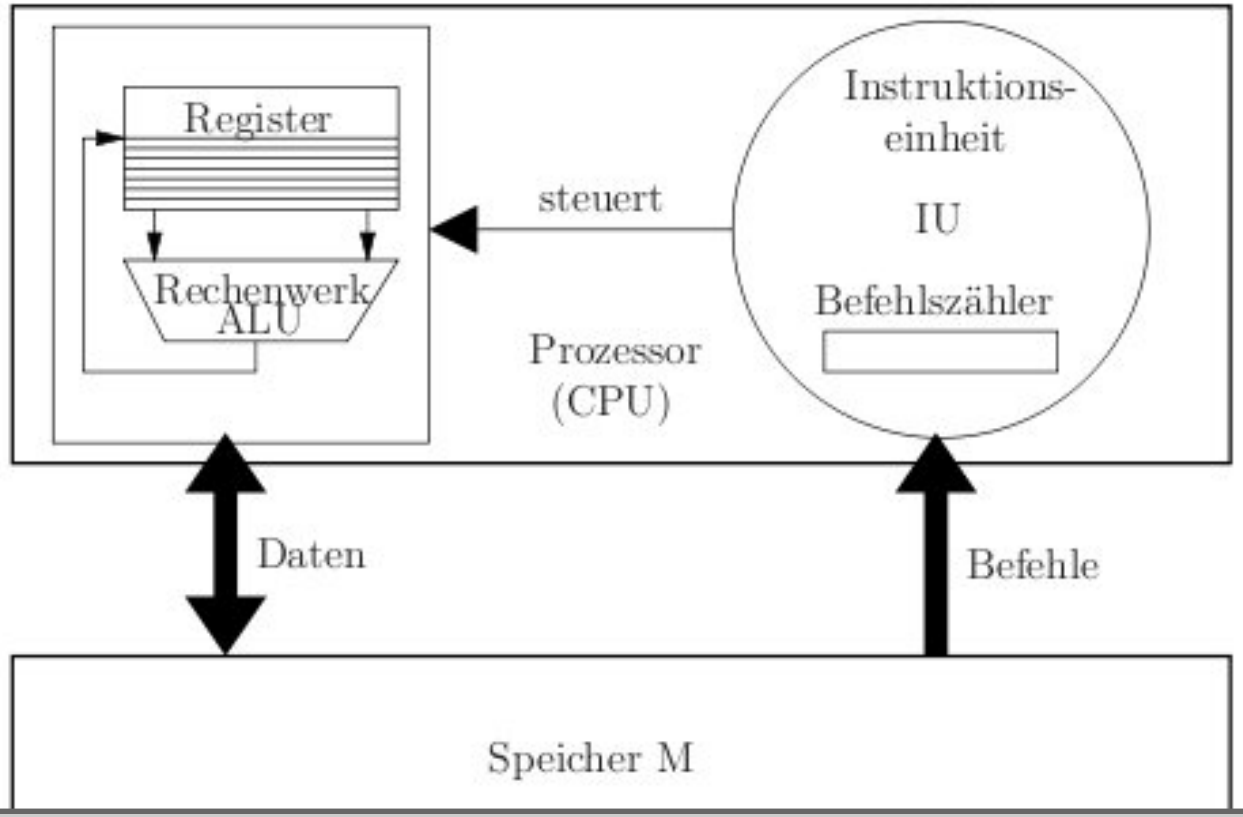
  ENIAC-Programm (University of Pennsylvania)

  Jesper P. Eckert & John W. Mauchly
- **E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer (EDVA) (1946)

  Einsatz: Balisitkberechnungen (US-Army)
- *First Draft of a Report on the EDVAC*, 1945

Der von–Neumann–Architektur II

- Referenzmodell für klassischen (sequenziellen) Computer
- Befehle & Daten werden in einem Speicher vorgehalten
- Erster universeller Computer (vorher: Lochkarten/Hardware)



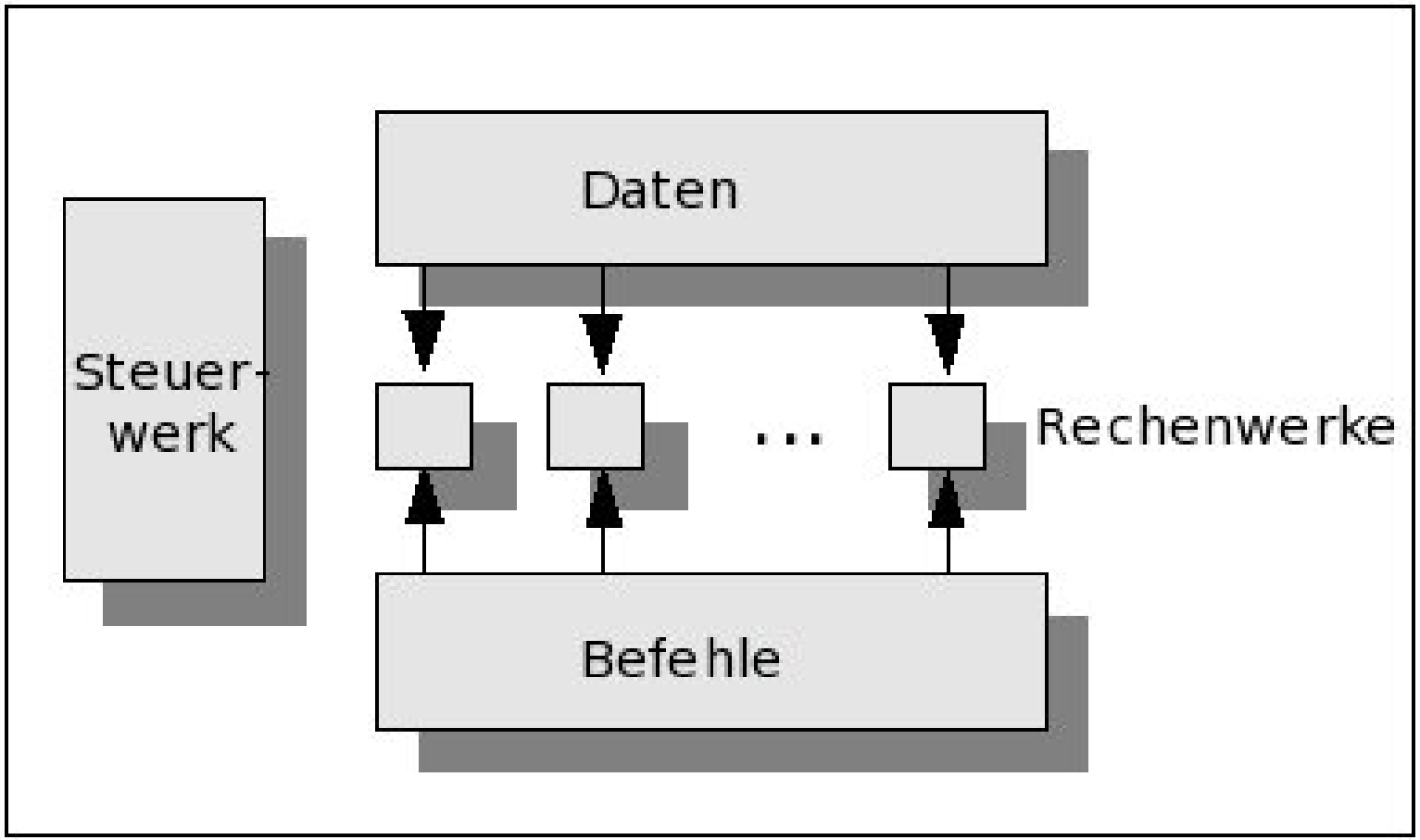
Die Harvard–Architektur

(Parallele) Harvard-Architektur

- Referenzmodell für schnelle CPUs (Pipelining)
- Befehle & Daten werden in getrennten Speichern vorgehalten
- Vorteile:
  - Befehle/Daten können gleichzeitig geladen/geschrieben werden
  - Einfacherer Speicherschutz (Zugriffsrechte)
  - getrennter Daten- u. Instruktions-Bus
- RISC (Reduced Instruction Set Computer)
  Verzicht auf komplexe Befehlssätze welche Speicherzugriffe (langsam) mit arithmetischen Operationen (schnell) kombinieren

Die Harvard–Architektur II

⇒ Liest in einem Zyklus Befehl und Datum (parallel)
während v.N. nacheinander lesen muss (2 Zyklen)

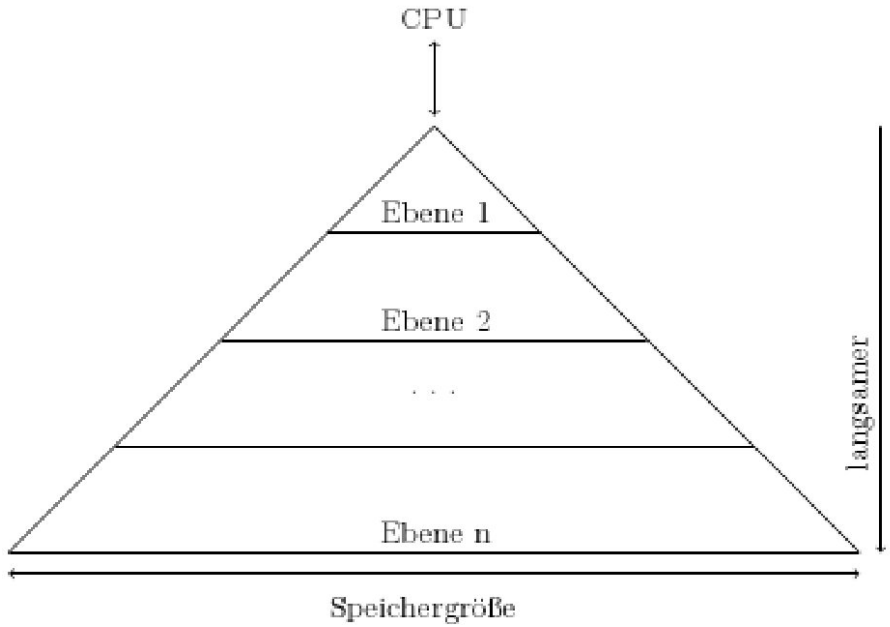


v.Neumann-Flaschenhals (bottle neck)

Speed-Up

Diskrepanz zwischen CPU- und Speicher-Taktrate

CPUs$\approx 100\times$ schneller als Speicher
(Stand 2010)



Abhilfe

- RISC-Befehlssatz
- Pipelining
- Abhilfe: Level 1/2/3-Caching

CISC – Complex Instrution Set Computers (vor 1985)

Warum CISC

- Einfacheres Programmieren durch komplexere Operationen in einem Befehl
  (Software war teuer)
- Beschränkter Speicher, deshalb wurden kürzere Programme favorisiert
  (Komplexe Befehle $\Rightarrow$ kompaktere Programme)

Nachteile:

- Komplexere Konstruktionen können geringere Taktrate implizieren
- Befehls-Pipelines sind schwerer zu realisieren
- längere Entwicklungszyklen
- viele Fehler
- nur ca. 30% der Befehle wurden genutzt

RISC – Reduced Instrution Set Computers (vor 1985)

- Speicher wurde billiger und schneller
- Dekodieren und Ausführen der Befehle wird zum limitierenden Faktor

Vorteile:

- Einfacheres Design
- leichter zu Debuggen
- kürzere Entwicklungszyklen
- billiger zu produzieren
- schneller Ausführung
- besseres Optimierungspotential
  (viele Abhängigkeiten zwischen komplexen Befehlssätzen)

Warum ist RISC schneller?

CISC – kurze Programme mit komplexen Befehlssätzen

RISC – längere Programme mit einfachen Befehlssätzen

Techniken in RISC-Rechner:

- Befehls-pipelines
  paralleles Abarbeiten von Befehlsketten
  (holen, dekodieren, ausführen, schreiben)
- Cache Speicher
  kleine und schnelle Speicherpuffer zwischen Hauptspeicher und CPU
- superskalare Ausführung
  Gruppierung von Befehlen zur parallelen Ausführung

Pipelining I

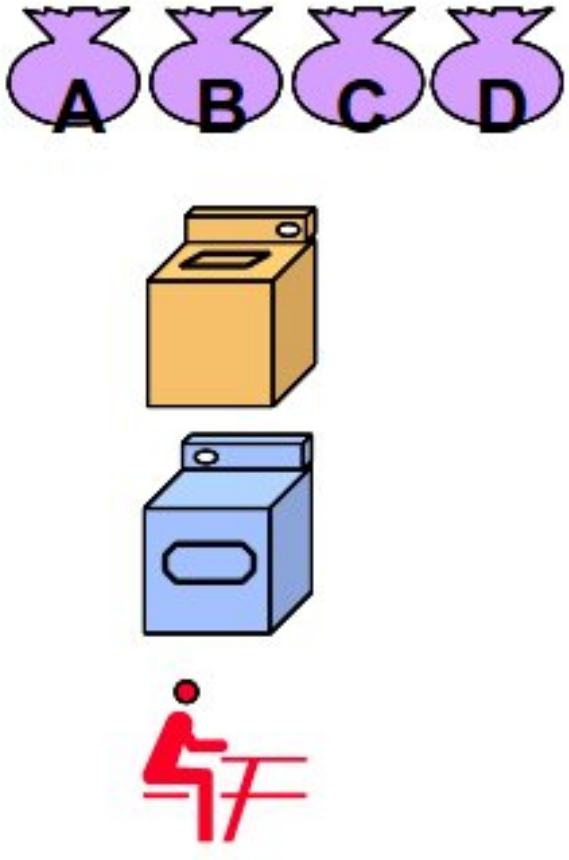
Fließbandverarbeitung

Zerlegen von Tasks in mehrere Teilschritte (Vorbild: Autoproduktion)

Pipelining II

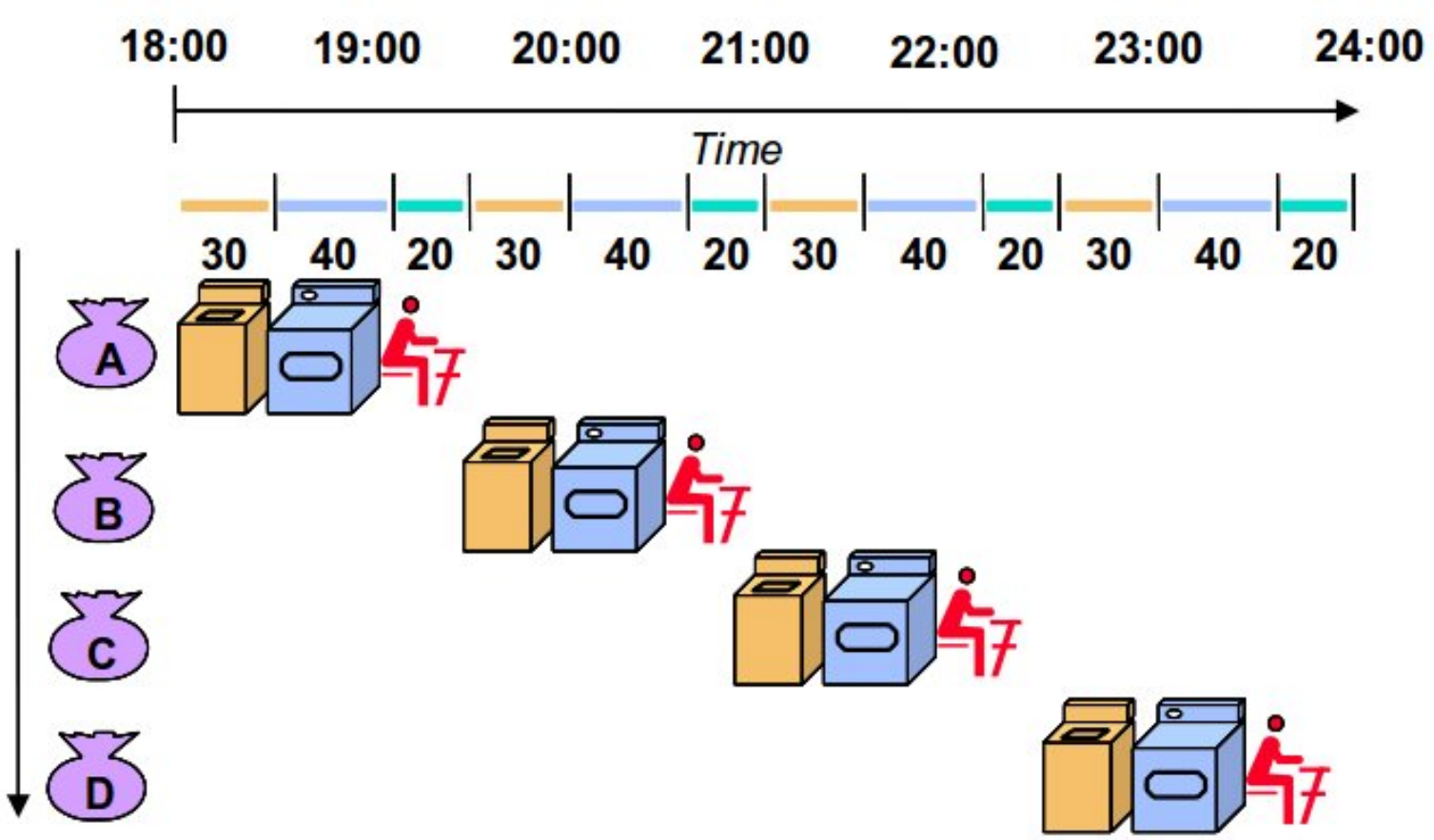
Beispiel: Waschsalon

- Anna, Britta, Caspar und Daniel haben jeweils eine Ladung schmutziger Klamotten zu waschen, trockenen und zusammenzulegen
- Waschen dauert 30 Minuten
- Trocknen dauert 40 Minuten
- Zusammenlegen 20 Minuten



Pipelining III

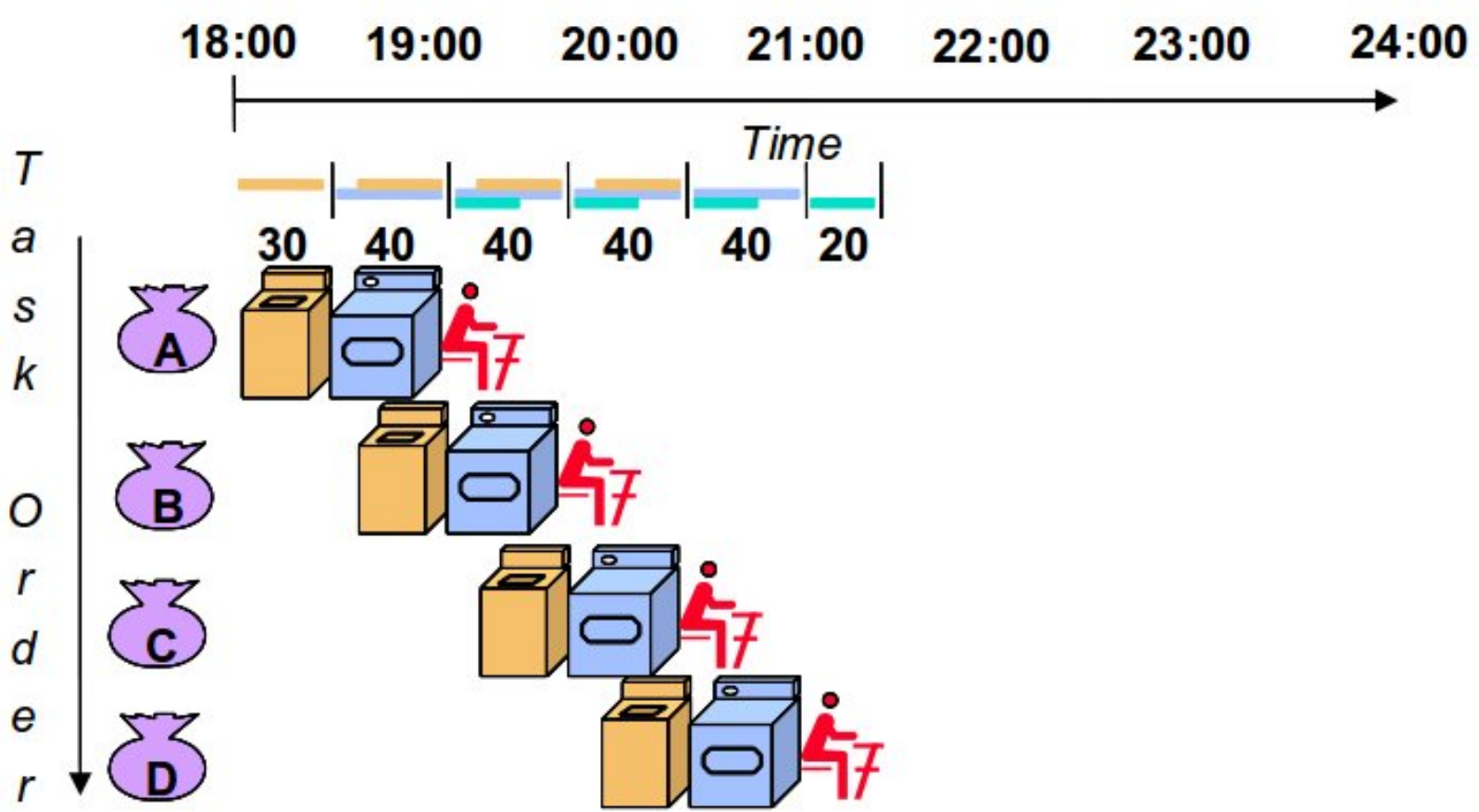
Beispiel: Waschsalon



Sequentielle Durchführung des Waschvorgangs (6 h für 4 Wäscheladungen)

Pipelining IV

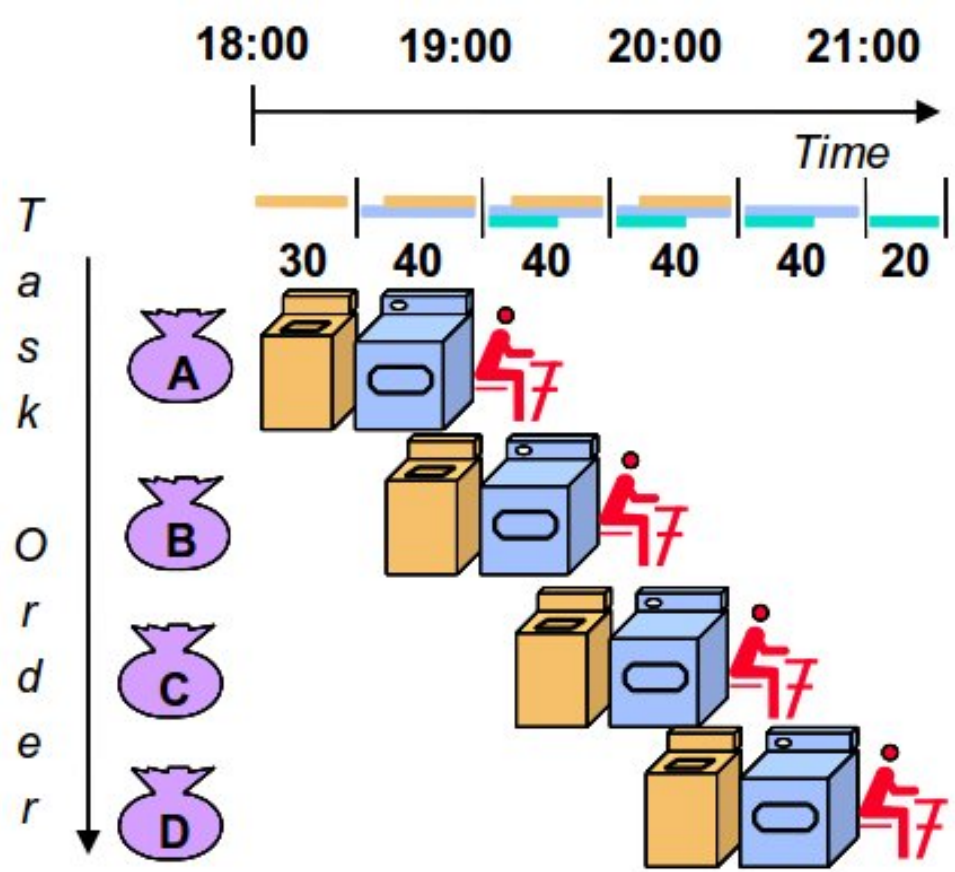
Beispiel: Waschsalon



Fließbandverarbeitung des Waschvorgangs (3,5 h für 4 Wäscheladungen)

Pipelining V

Beispiel: Waschsalon



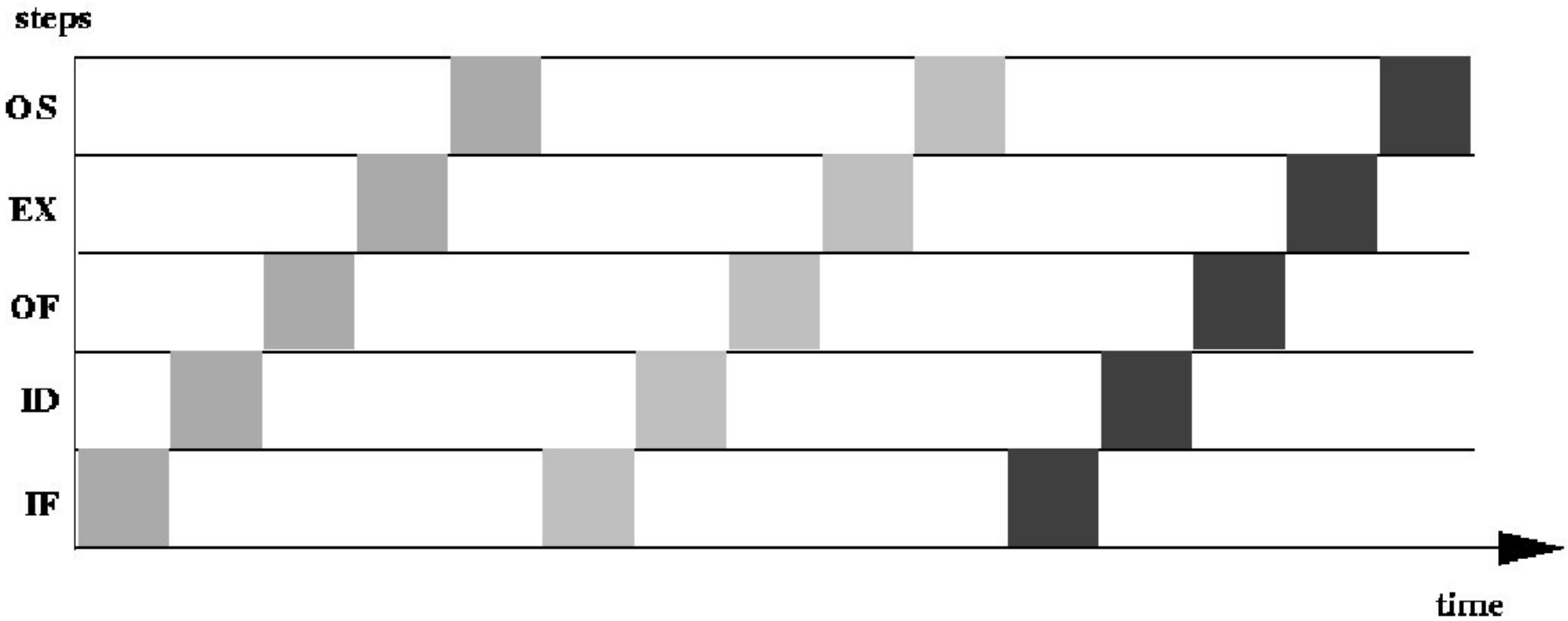
- Pipelining erhöht Durchsatz (Operationen/Zeit)
- Abhängig vom längsten Bearbeitungsschritt (Operanden)
- Bearbeitungszeit für einzelnen Schritt ggf. länger
- Unbalancierte Länge der Operationen reduziert Speedup
- Potentieller Speedup $=\#$ Einzeloperationen

Fließbandverarbeitung des Waschvorgangs (3,5 h für 4 Wäscheladungen)

Pipelining VI

Logische Phasen im Rechner

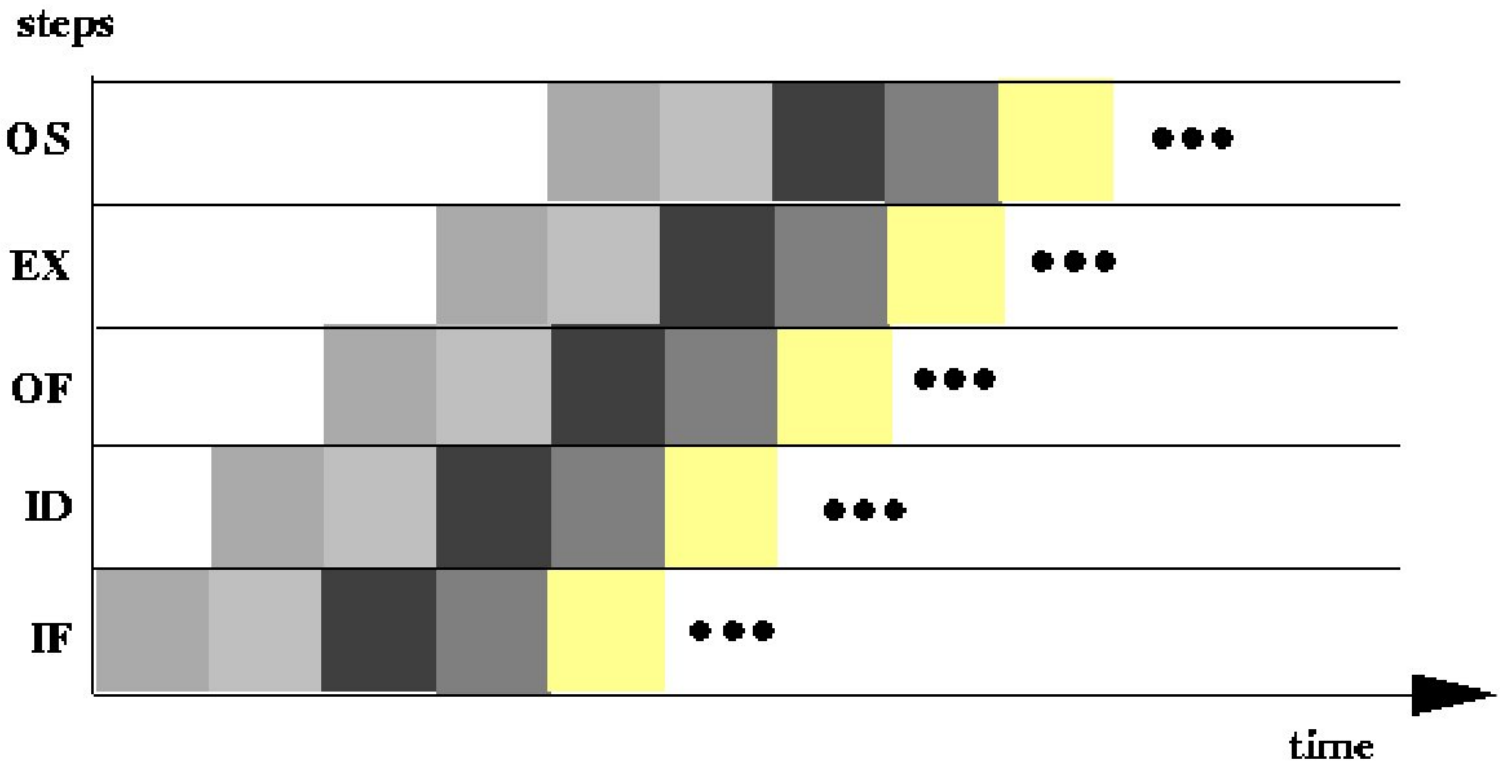
- **IF** Instruction Fetch **ID** Instruction Decode
- **OF** Operand Fetch **EX** Execution **OS** Operand Store



Pipelining VII

Logische Phasen im Rechner

- **IF** Instruction Fetch **ID** Instruction Decode
- **OF** Operand Fetch **EX** Execution **OS** Operand Store



Probleme – Pipelining I

Datenabhängigkeit

Eine Anweisung hängt vom Ergebnis der Vorgängeranweisung ab.

```
a=x+y;
b=a*c;
```

- Die Berechnung von `b` muss auf die Beendigung von a warten.
- Auflösbare Datenabhängigkeit:
  Die Berechnung von a kann vorgezogen werden

Out of order execution

Compiler kann Ausführungsreihenfolge verändern, um Datenabhängigkeiten aufzulösen.

Probleme – Pipelining II – Verzweigungen

Bedingte Kontrollflussabhängigkeit

Der Kontrollfluss hängt vom Ergebnis der Vorgängeranweisung ab.

```
if(d>2)
{x=0.;}
else
{x=1.;}
```

Out of order execution

Compiler kann Ausführungsreihenfolge verändern, um Kontrollflussabhängigkeiten zu entschärfen.

Branch prediction

Hardware sagt wahrscheinlichsten Zweig voraus, berechnet die Anweisungen spekulativ, verwirft u.U. die Ergebnisse.

Probleme – Pipelining III – Inlining

Unbedingte Kontrollflussabhängigkeit

Der Kontrollfluss muss unterbrochen werden.

```
double mult(a,b) {return(a*b);}
...
for(i=0;i<n;i++) x[i]=mult(x[i],1+1./2.);
```

Inlining

Präprozessor ersetzt Funktionsaufruf.

```
#define mult(a,b) {(a)*(b)}
...
for(i=0;i<n;i++) x[i]=mult(x[i],1+1./2.);
```

Probleme – Pipelining IV – loop unrolling

Unbedingte Kontrollflussabhängigkeit

Der Kontrollfluss muss unterbrochen werden.

```
for(i = 0; i < n; i++)
x[i] = a * x[i];
```

Inlining

Abrollen der Schleife – Erhöhung des Schleifeninkrements

```
for(i = 0; i < n; i+=4)
{
x[i] = a * x[i];
x[i + 1] = a * x[i + 1];
x[i + 2] = a * x[i + 2];
x[i + 3] = a * x[i + 3];
}
```

Probleme – Pipelining IV – loop unrolling

Unbedingte Kontrollflussabhängigkeit

Der Kontrollfluss muss unterbrochen werden.

```
for(i = 0; i < n; i++)
x[i] = a * x[i];
```

Inlining

Abrollen der Schleife – Erhöhung des Schleifeninkrements

```
for(i = 0; i < n; i+=4)
{
x[i] = a * x[i];
x[i + 1] = a * x[i + 1];
x[i + 2] = a * x[i + 2];
x[i + 3] = a * x[i + 3];
}
```

Prozessor

Leistungsfähigkeit einer CPU – Rechenoperationen pro Zeiteinheit

FLoating point OPerations per second (FLOPs)

Berechnet sich aus

- der Anzahl der Pipelines
- Operationen pro Pipeline
- Durchschnittlichen Zeiteinheit pro pipline-Phase

Interessiert an

- Peak Performance (Maximal-Leistung)
- Durchschnittlicher Leistung (LinPack-Test)
- Verhältnis zu Speicherleistung

Peak Performance

Prozessor

Taktrate 2,4 GHz

- der Anzahl der Pipelines
  # ALUs (Arithmetic Logical Units)
- Operationen pro Pipeline
- #OPsPerALU im Allg. zwei arithmetische Operationen gleichzeitig
- Durchschnittlichen Zeiteinheit pro pipline-Phase
  Taktrate Prozessor, 2,4 Ghz $\Rightarrow$ $\frac{1}{2{,}4 \times 10^9}$ Sek. $\approx$ 417 psec

Peak Performance

$$R_{peak} = \frac{\#ALUs \cdot \#OPpALU}{Zeiteinheit} = \frac{2 \cdot 2}{417 psec} \approx 9,6 \text{GFLOPs}$$

Speichertakt vs. Prozessertakt

Zeit für das wiederholte Lesen eines Datums aus dem Speicher

DRR3-1600 RAM (Double Data Rate Dynamic Random Acess Memory)
Speicherfrequenz 400 Hz, Busfrequenz 1600 Hz

- Adresse über das Bussystem zum Speicher bringen.
  (*Benötigt ca. 1 Takt der Geschwindigkeit des Bussystem*)
- Zwischen Eingang der Adresse und Erhalt des Datums vergeht Speicherlatenzzeit. (*Benötigt ca. 4 Takte d. Speichergeschwindigkeit*)
- Nach Lesen/Schreiben von Speicherdaten wird dieser aufgefrischt.
  (*Benötigt ca. 20-40 Takte der Speichergeschwindigkeit*)
- Transport der Daten vom Speicher
  (*benötigt ca. 1 Takt der Busgeschwindigkeit*)

$$\underbrace{\frac{1+1}{1600 \cdot 10^6}}_{\textit{Busgeschwindigkeit}} + \underbrace{\frac{8}{400 \cdot 10^6}}_{\textit{Speicherlatenzzeit}} + \underbrace{\frac{30}{400 \cdot 10^6}}_{\textit{Auffrischen}} \approx 96\,\textit{nsec}$$

Speichertakt vs. Prozessortakt

Rechner

2,4 Ghz, DRR3-1600 RAM, 1600 FSB 400 DRAM

- Peak Performance $R_{peak} = 9,6$ GFLOPS
- $\Longleftrightarrow$ Taktrate 417 psec
- Operation (Lesen eines Datums) 96 nsec

$$\Rightarrow \quad \frac{96\,nsec}{417\,psec} \approx 230$$

Das heißt

- 230 Prozessortakte vergehen während des Lesens/Holen eines Datums
- bzw. 920 Operationen können in der Zeit ausgeführt werden

Klassifikation nach Flynn

Michael J. Flynn (*1934) Professor Emeritus, Stanford University

Klassifizierung der Rechnerarchitekturen nach der Anzahl der vorhandenen Befehls- und Datenströme.

	Single Instruction	Multiple Instruction
Single Data	**SISD**	**MISD**
Multiple Data	**SIMD**	**MIMD**

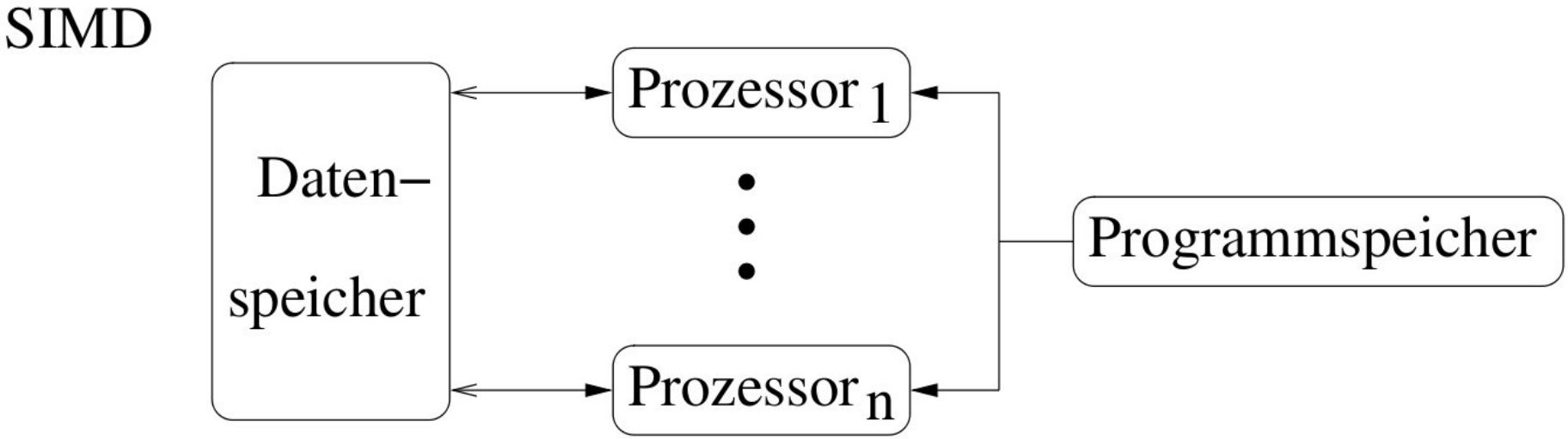
SISD – Single Instruction Single Data

- Sequentielle Rechner
- Klassischer Rechner mit einem Rechenwerk
- Princeton Architektur (v.Neumann)
- Harvard-Architektur

SISD

SIMD – Single Instruction Multiple Data

- Parallele Systemarchitektur
- Moderne Prozessoren mit mehreren Rechenwerken
- Daten werden in sog. Vektor-Registern gespeichert
- Jeder Befehl verarbeitet ein Vektor-Register
- Weiterentwicklung der Harvard Architektur



SIMD – Single Instruction Multiple Data

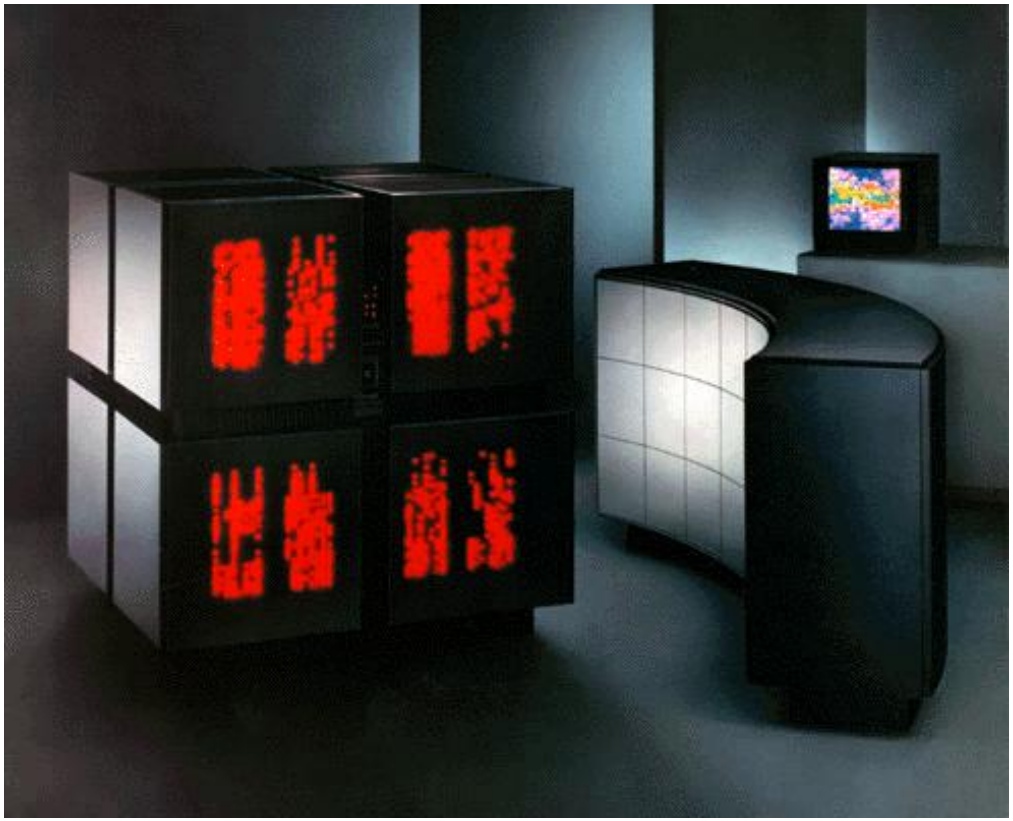
- Array- oder Vektorprozessor (SIMD durch parallele Register)
- Taktsynchrone Steuerung zur Abarbeitung einer Operation auf verschiedenen Daten (z.B. Addition zweier Vektoren)

Beispiel: Cray-1 (1976)

- 80 MFLOPS, 64 Vektorregister
- Gewicht: 5.5 Tonnen
- Los Alamos National Lab. (Kernwaffentestberechnung)
- European Centre for Medium Range Weather Forecasts (10tägige Wettervorhersage)

SIMD – Single Instruction Multiple Data

Thinking Maschines



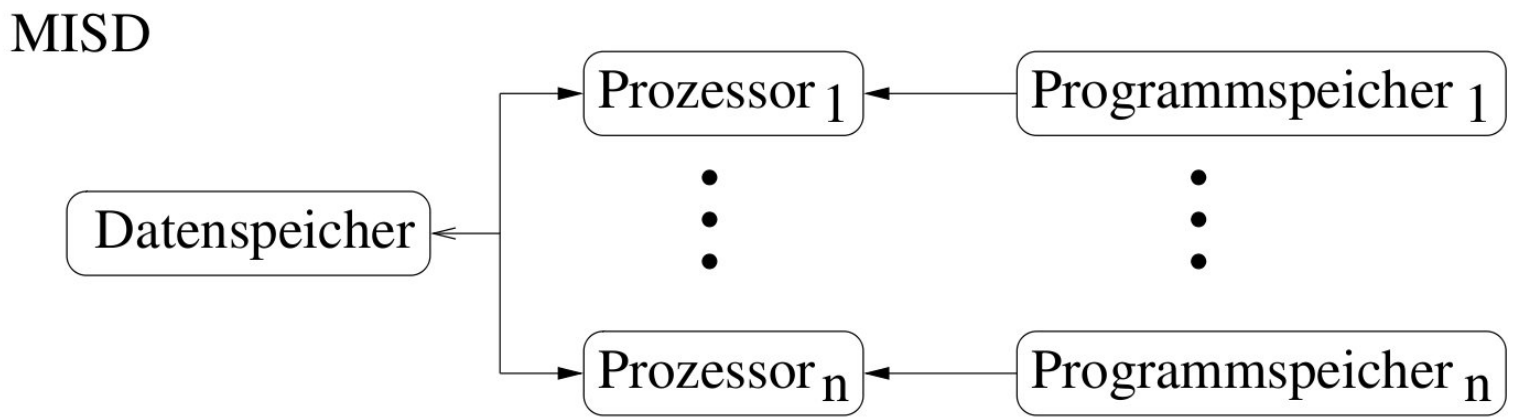
Beispiel: Connection Maschine

CM-1 (1985)

- 201 MFlops
- 65536 1-Bit Prozessoren
- mit eigenem 4K-Bit Speicher
- Taktrate 4 Mhz
- SIMD Maschine mit einer Kontrolleinheit
- CM-5 (1991)
  MISD-Architektur

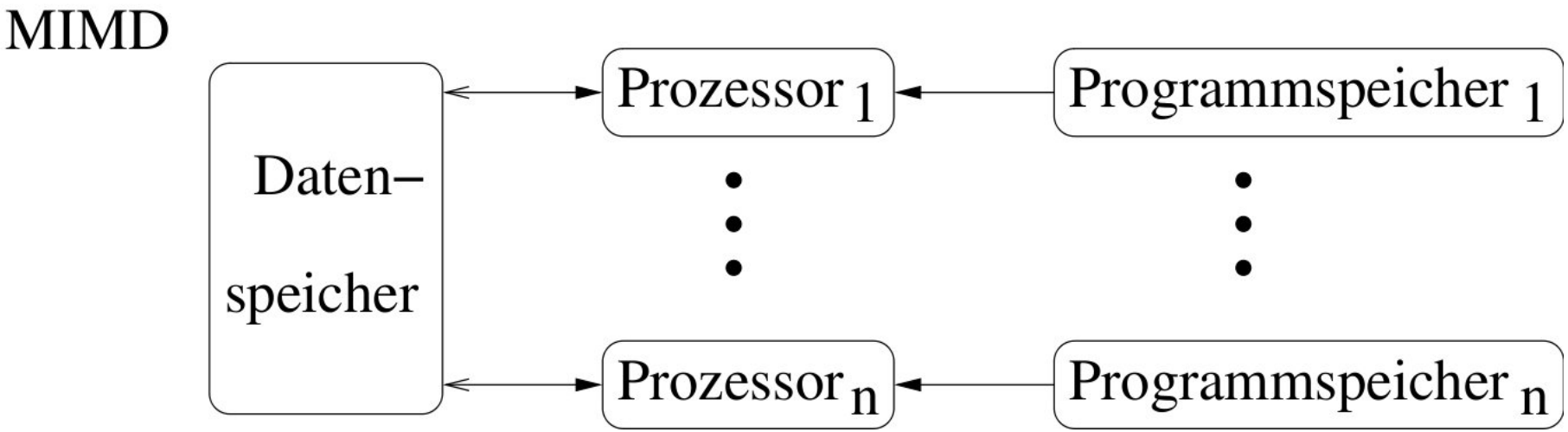
MISD – Multiple Instruction Single Data

- Parallele oder Serielle Systemarchitektur
- Einsatzgebiet:
  Redundante Berechnungen od. verschiedene Lösungen eines Problems
- Oft wird diese Klasse auch als leer angegeben
- Beispiele
  - Schachprozessoren
  - Optimierungsprozessoren



MIMD – Multiple Instruction Multiple Data

- Parallele Systemarchitektur
- Ausführungseinheiten sind unabhängig und arbeiten asynchron.
- Prozessoren verarbeiten verschiedene Operationen auf unterschiedliche Daten
- Beispiel: Moderne Cluster-Systeme



Top 500 Rechner

- Ab ca. 2000 basierten alle Top 10 und die meisten TOP500 Supercomputer auf MIMD-Architekturen

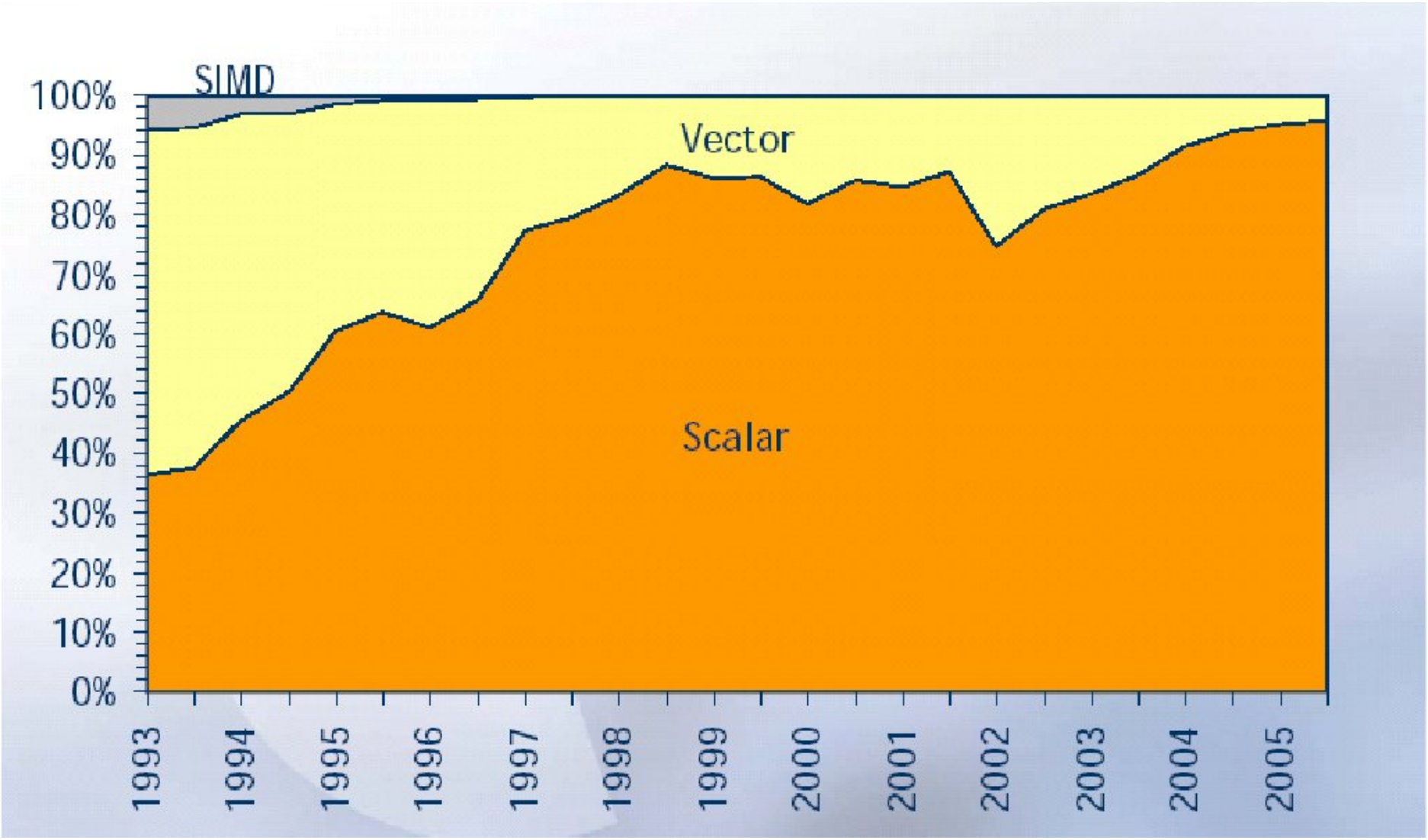
  `http://www.top500.org`

K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect

Site:	RIKEN Advanced Institute for Computational Science (AICS)
System URL:	
Manufacturer:	Fujitsu
Cores:	705024
Power:	12659.89 kW
Memory:	1410048 GB
Interconnect:	Custom
Operating System:	Linux

- RIKEN Advanced Institute for Computational Science (AICS), Kobe 10.51 Petaflop/s on Linpack benchmark (705,024 SPARC64 Cores)

MIMD vs. SIMD


SIMD
Vector
Scalar
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005

Green 500 Rechner

- Da ca. 2% des jährlichen CO_2-Ausstoßes auf den Energieverbrauch von Rechner zurückgeht, wird Leistung pro Kilowatt wichtiger

  `http://www.green500.org`

Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)
1	2026.48	IBM - Rochester	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	85.12
2	2026.48	IBM Thomas J. Watson Research Center	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	85.12
3	1996.09	IBM - Rochester	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	170.25
4	1988.56	DOE/NNSA/LLNL	BlueGene/Q, Power BQC 16C 1.60 GHz, Custom	340.50
5	1689.86	IBM Thomas J. Watson Research Center	NNSA/SC Blue Gene/Q Prototype 1	38.67
6	1378.32	Nagasaki University	DEGIMA Cluster, Intel i5, ATI Radeon GPU, Infiniband QDR	47.05
7	1266.26	Barcelona Supercomputing Center	Bullx B505, Xeon E5649 6C 2.53GHz, Infiniband QDR, NVIDIA 2090	81.50
8	1010.11	TGCC / GENCI	Curie Hybrid Nodes - Bullx B505, Nvidia M2090, Xeon E5640 2.67 GHz, Infiniband QDR	108.80
9	963.70	Institute of Process Engineering, Chinese Academy of Sciences	Mole-8.5 Cluster, Xeon X5520 4C 2.27 GHz, Infiniband QDR, NVIDIA 2050	515.20
10	958.35	GSIC Center, Tokyo Institute of Technology	HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows	1243.80

Exkurs: GPUs/Cell-Prozessoren

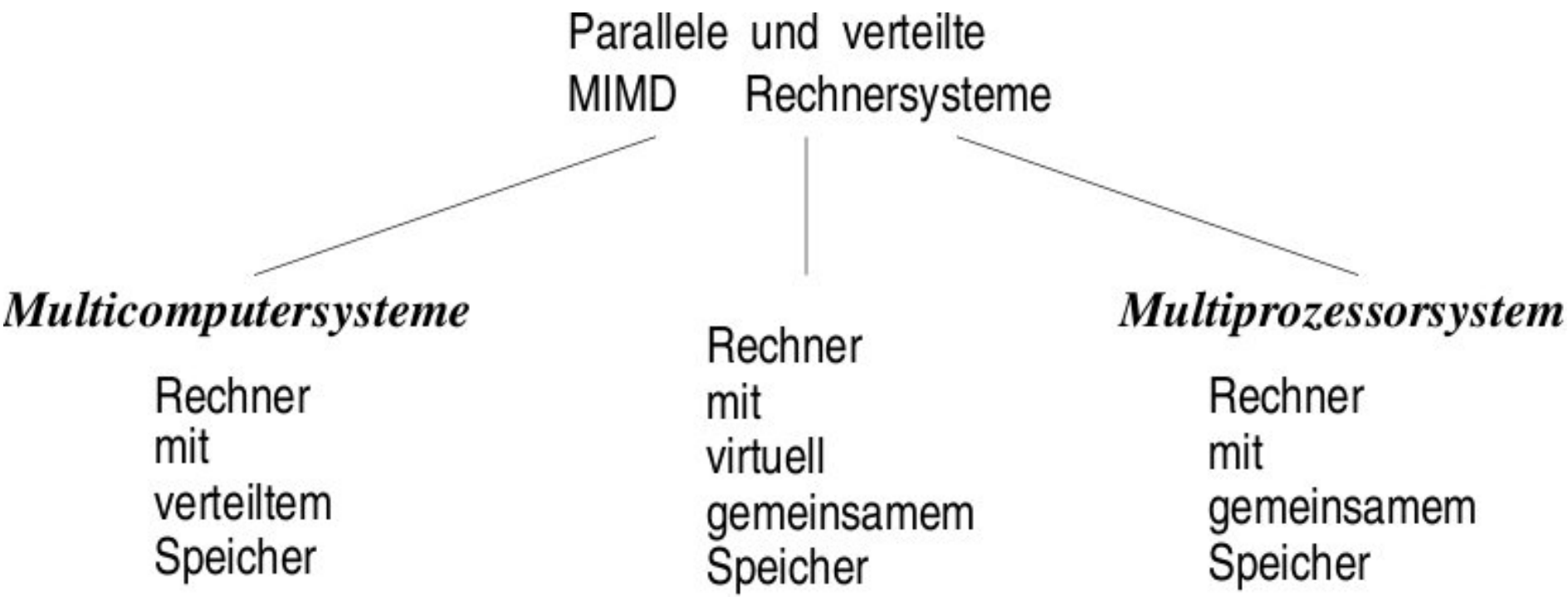
GPUs/Cell-Prozessoren

- Paradigmenwechsel?
- GPUs rechnet man allgemein zu den SIMD-Architekturen
- Sie besitzen wesentlioch mehr ALUs, als herkömmliche Rechner
- Mehrer Kontroller mit Cache-Speicher
- Ein Kontroller verwaltet mehrere ALUs

$\Rightarrow$ Massiver Parallelrechner in der GPU realisiert

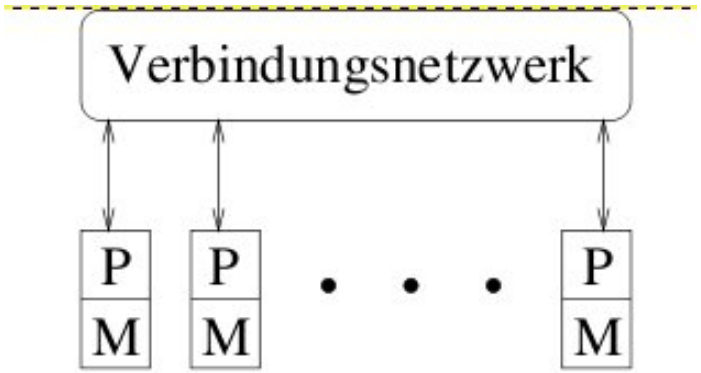
Speicherorganisation von Parallelrechnern

MIMD



DMM und SMM

DMM – Distributed Memory Maschines

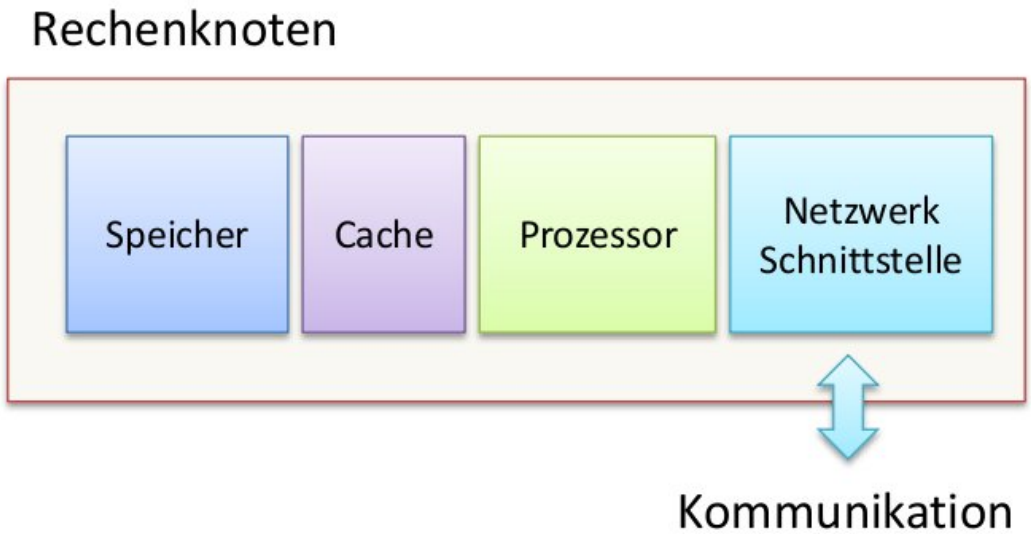


SMM – Shared Memory Maschines

Rechner mit verteiltem Speicher I

DMM – Distributed Memory Maschines

- Besteht aus mehreren Verarbeitungseinheiten (Knoten)
- Netzwerk, welches die Knoten physikalisch verbindet.
- Knoten bestehen aus
  - Prozessor
  - lokalem Speicher
  - ggf. Netzwerkcontroller (I/O)



Rechner mit verteiltem Speicher II

DMM – Distributed Memory Maschines

- Lokale Speicher sind privat
- Nachrichtenaustausch/Kommunikation über das Netzwerk

- Kommunikation zwischen kooperierenden sequentielen Prozessen
- Prozesse P_A, P_B auf verschidenen Knoten A und B
- P_A sendet Nachricht an P_B:
  - P_A führt Sendebefehl aus mit Nachricht und Ziel P_B
  - P_B führt Empfangsbefehl aus, mit Angabe des Empfangspuffers und sendenden Prozesses P_A

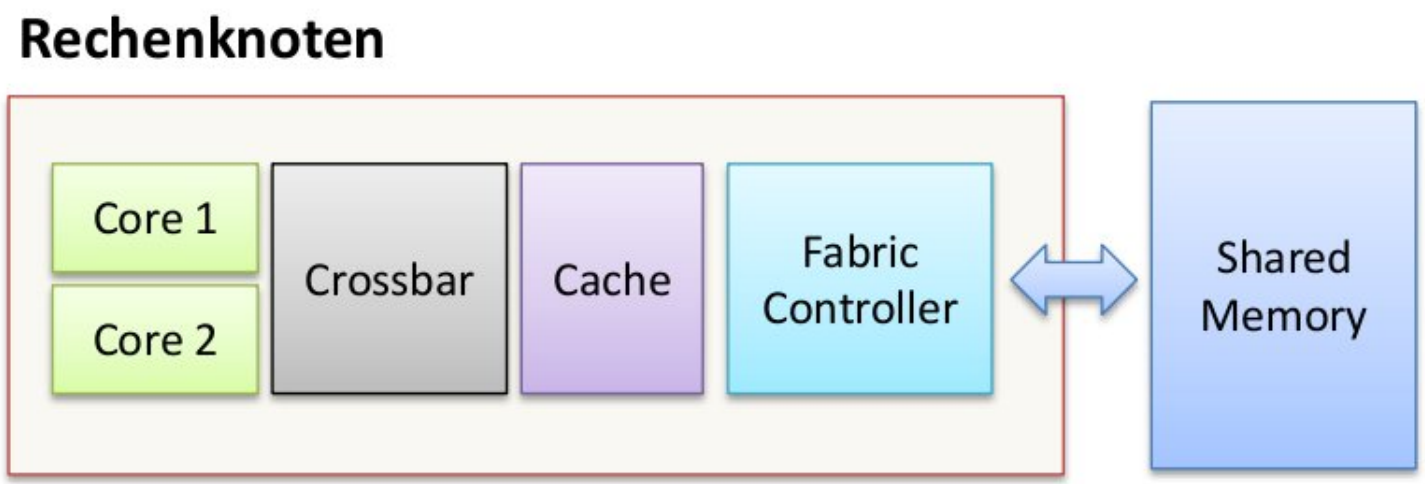
message-passing programming model

Message Passing Interface (MPI)

Rechner mit gemeinsammen Speicher

SMM – Shared Memory Maschines

- mehrere Prozessoren oder Prozessorkerne
- **gemeinsammer** oder **globaler** Speicher
- einheitlicher Adressraum
- Verbindung zwischen Prozessoren und Speicher (nicht notwendigerweise Netzwerk)



Rechner mit gemeinsammen Speicher II

SMM – Shared Memory Maschines

- Zugriff auf gemeinsamme Variablen

  $\Rightarrow$ zu vermeiden sind **gleichzeitiges unkoordiniertes Schreiben** verschiedener Prozessoren

Vorteile SMM

- einfachere Programmierung
- geringere Kommunikation
- bessere Speicherausnutzung

Nachteile SMM

- Erfordert hohe Übertragungskapazität des Verbindungsnetzwerks (Damit schneller Zugriff der Prozessoren auf Speicher möglich ist)

  $\Rightarrow$ grosse Anzahl Prozessoren schwer zu realisieren

Verbindungsnetzwerk

Gestaltungskriterien eines Netzwerks (interconnection)

- **Topologie**
  Form der Verschaltung der einzelnen Prozessoren od. Speichereinheiten
- **Routingtechniken**
  realisiert Nachrichtenübertragung zw. Prozessoren/Speicher verschiedener Prozessoren

Topologie

- beschreibt geometrische Struktur
- Graph mit Prozessoren, Speicher und Switches als Knoten
- statische vs. dynamische Netzwerke

Routingtechnik

- beschreibt **wie** eine Nachricht übertragen wird
- **entlang welchen Pfades** eine Nachricht übertragen wird

Pfade in Clusternetzwerke

Beschreibung des Netzwerks durch Kommunikationsgraphen G=(V,E)

- V Knotenmenge
- E Kantenmenge
- Verbindung zwischen Knoten **u** und **v** (Pfad)

  Eine Folge von Knoten $(v_0, \ldots, v_k)$ heißt Pfad der Länge k zwischen v_0 und v_k, falls $(v_i, v_{i+1}) \in E$ für $0 \leq i < k$

Bewertungskriterien

- Durchmesser
- Grad
- Bisektionsbreite
- Knoten- und Kantenkonnektivität

Bewertungskriterien in Clusternetzwerke

Durchmesser $\delta(G)$

Maximale Distanz zwischen zweibeliebigen Prozessoren

$$\delta(G) = \max_{u,v \in V} \min_{\phi} \{k \mid k \text{ ist Länge des Pfades } \phi \text{ von } u \text{ nach } v\}$$

- Maß für die Kommunikationsdauer im Netzwerk
- Beschreibt, wie lange es dauert, bis eine von einem beliebigen Prozessor abgeschickten Nachricht bei einem beliebigen anderen Prozessor ankommt.

Ziel

Kleiner Durchmesser bei der Nachrichtenübertragung

Bewertungskriterien in Clusternetzwerke II

Grad $g(G)$

Maximale Grad eines Knotens des Netzwerks.
Der Grad eines Knotens ist die Anzahl seiner adjazenten Kanten.
(Adjanzente Kanten = ein- bzw. auslaufenden Kanten eines Knotens)

$$g(G) = \max\{g(v) \mid g(v) \text{ Grad von } v \in V\}$$

- Beschreibt den Hardwareaufwand im Clusternetzwerk

Ziel

Kleiner Grad für jeden Knoten

Bewertungskriterien in Clusternetzwerke III

Bisektionsbandbreite

Minimale Anzahl der Kanten, die aus dem Netzwerk entfernt werde müssen, um das Netzwerk in zwei gleiche Teile zu zerlegen

$$B(G) = \min_{\|U_1 - U_2\| \leq 1} |\{(u,v) \in E \mid u \in U_1,\ v \in U_2\}|,\ U_1,\ U_2 \text{ Partition von } V$$

- Bereits $B(G)+1$ Nachrichten können das Netzwerk sättigen, falls sie zur gleichen Zeit gesendet werden.
- Maß für die Belastbarkeit des Netzwerks bei gleichzeitiger Übertragung von Nachrichten.
- Wichtig bei all-to-all-Kommunikation

Ziel

Hohe Bisektionsbandbreite zur Erreichung eines hohen Durchsatzes

Bewertungskriterien in Clusternetzwerke IV

Knotenkonektivität *VC* und Kantenkonnektivität *EC*

Minimale Anzahl von Knoten/Kanten, die gelöscht werden müssen, um das Netzwerk zu unterbrechen

$$VC(G) = \min_{M \subset V} \{|M| \quad \exists u, v \in V \backslash M, \nexists \text{ Pfad } \phi \text{ von } u \text{ nach } v \in G_{V \backslash M}\}$$

$$EC(G) = \min_{F \subset E} \{|F| \quad \exists u, v \in V, \nexists \text{ Pfad } \phi \text{ von } u \text{ nach } v \in G_{E \backslash F}\}$$

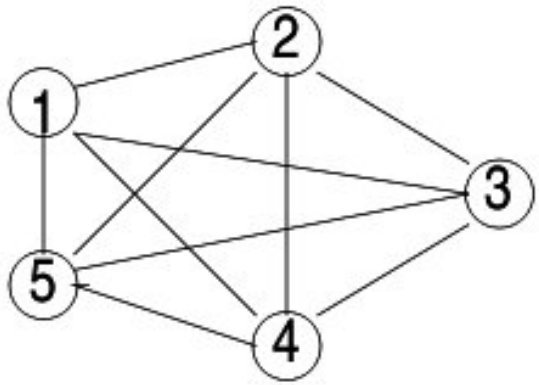
- Maß für die unabhängigen Wege zwischen zwei beliebigen Prozessoren
- Beschreibt Ausfallsicherheit bzw Zuverlässigkeit im Netzwerk

Ziel

Hohe Konnektivität zur Erreichung einer hohen Ausfallsicherheit/Zuverlässigkeit im Netzwerk

Clustertopologie – vollständiger Graph

Jeder Knoten ist mit jedem verbunden (n # Prozessoren $= |V|$)



$g(G)$:	n-1	Grad	# Prozessoren N=16	15
$\delta(G)$:	1	Durchmesser		1
$EC(G)$:	n-1	Kantenkonnektivität		15
$B(G)$:	$\frac{n}{2}^2$	Bisektionsbandbreite		64

Wegen des hohen Knotegrades nur für eine kleine Anzahl n geeignet.

Clustertopologie – lineares Feld

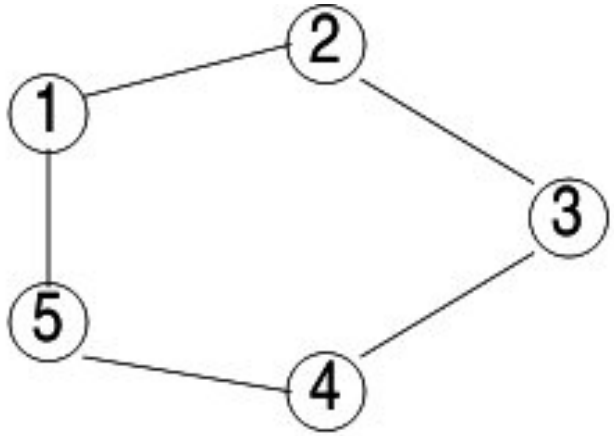
lineare Anordnung, d.h. **bidirektionale** Verbindung zwischen benachbarten Prozessoren

$g(G)$:	2	Grad	# Prozessoren N=16	2
$\delta(G)$:	n-1	Durchmesser		15
$EC(G)$:	1	Kantenkonnektivität		1
$B(G)$:	1	Bisektionsbandbreite		1

Geringe Fehlertoleranz, da bereits beim AUsfall eines Knotens das Netzwerk unterbrochen ist. (EC(G)=1).

Clustertopologie – Ring

Prozessoren sind in einer Ring-Topologie angelegt.

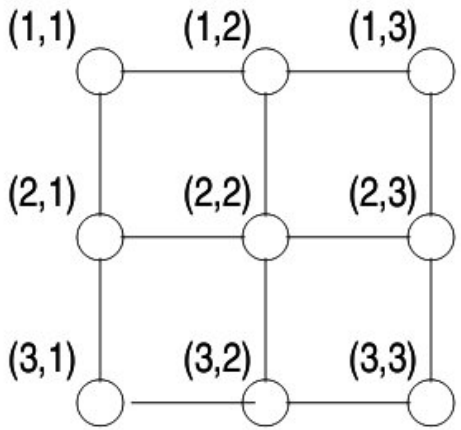


$g(G)$:	2	Grad	# Prozessoren N=16	2
$\delta(G)$:	$\lfloor \frac{n}{2} \rfloor$	Durchmesser		8
$EC(G)$:	2	Kantenkonnektivität		2
$B(G)$:	2	Bisektionsbandbreite		2

Wird für kleine Prozessoranzahl und komplexe Netzwerke eingesetzt

Clustertopologie – d-dimensionales Gitter

Gitter oder Feld aus $n = n_1 \cdot n_2 \cdot \ldots \cdot n_d$ Knoten in Form eines d-dimensionalen Gitters (# Prozessoren: $n = r^d$)

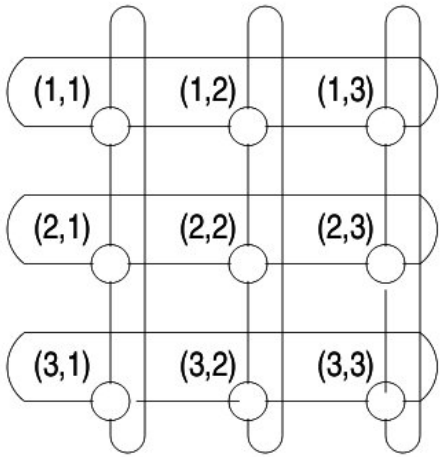


$g(G)$:	2d	Grad	# Prozessoren N=16	8
$\delta(G)$:	$d(\sqrt[d]{n} - 1)$	Durchmesser		4
$EC(G)$:	d	Kantenkonnektivität		4
$B(G)$:	$n^{\frac{d-1}{d}}$	Bisektionsbandbreite		8

2-dim. Gitter wurde im Terascale-Prozessor (Intel) realisiert (80 Cores).

Clustertopologie – d-dimensionales Torus

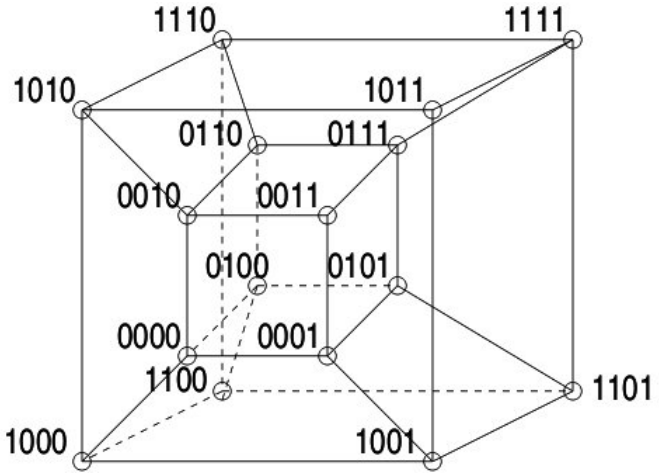
Variante des d-dim. Gitters. Enthält zusätzlich für jede Dim. $j = 1, \ldots, d$ zwischen $(x_1, \ldots, x_{j-1}, 1, x_{j+1}, \ldots, x_d)$ und $(x_1, \ldots, x_{j-1}, n_j, x_{j+1}, \ldots, x_d)$ zusätzliche Kanten. Realisiert als 3d-Torus in Cray XT3 und XT4.



$g(G)$:	$2d$	Grad	# Prozessoren N=16	8
$\delta(G)$:	$d\lfloor \frac{\sqrt[d]{n}}{2} \rfloor$	Durchmesser		4
$EC(G)$:	$2d$	Kantenkonnektivität		8
$B(G)$:	$2n^{\frac{d-1}{d}}$	Bisektionsbandbreite		16

Clustertopologie – d-dimensionaler Hyperwürfel

hypercube: Besitzt $n = 2^d$ Knoten, zwischen denen entsprehend niedrigdimensionale Würfel existieren. Jedem Knoten wird ein binäres Wort der Länge k als Name zugeordnet.



$g(G)$:	$\log n$	Grad	# Prozessoren N=16	4
$\delta(G)$:	$\log n$	Durchmesser		4
$EC(G)$:	$\log n$	Kantenkonnektivität		4
$B(G)$:	$\frac{n}{2}$	Bisektionsbandbreite		8

Clusterperformance – weitere Parameter

- **Latenzzeit**
  Zeit für das Abschicken und Empfangen eines Datenpaketes der Länge Null.
- **Bandbreite**
  Übertragungsleistung des Netzwerks in Byte/sec.
- **Nicht blockierend**
  Keiner der Pfade kann durch eine andere Kommunikation belegt sein.
- **Skalierbarkeit**
  Kann das Netzwerk für wachsende Prozessorzahl wachsende Bandbreiten bei konstanten Latenzzeiten liefern?