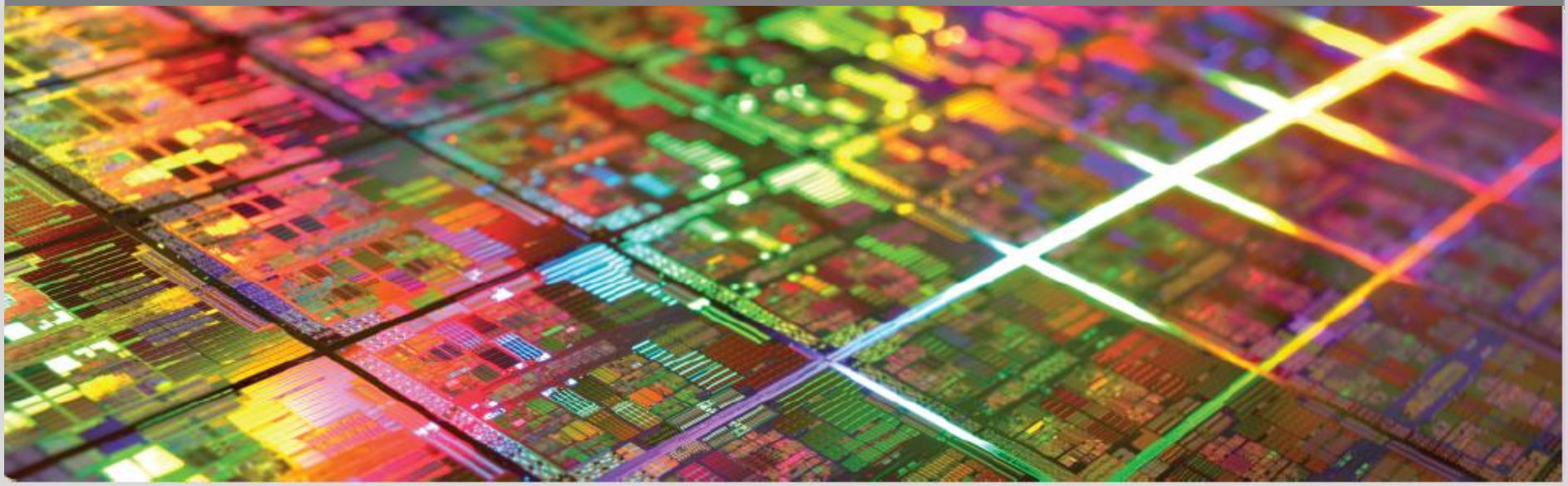
Karlsruher Institut für Technologie

# Rechnerstrukturen

## Vorlesung im Sommersemester 2011

**Prof. Dr. Wolfgang Karl**

Fakultät für Informatik – Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung



# Vorlesung Rechnerstrukturen

## Kapitel 3: Multiprozessoren – Parallelismus auf Prozess-/ Blockebene

- 3.1 Motivation

# Multiprozessorsysteme

## Motivation

- Allgemeine Grundlagen, parallele Programmierung, Verbindungsstrukturen, Leistungsfähigkeit
- Speichergekoppelte Multiprozessoren: SMP und DSM, Cache-Kohärenz und Speicherkonsistenz, Rechnerbeispiele
- Nachrichtengekoppelte Multiprozessoren, Beispielrechner

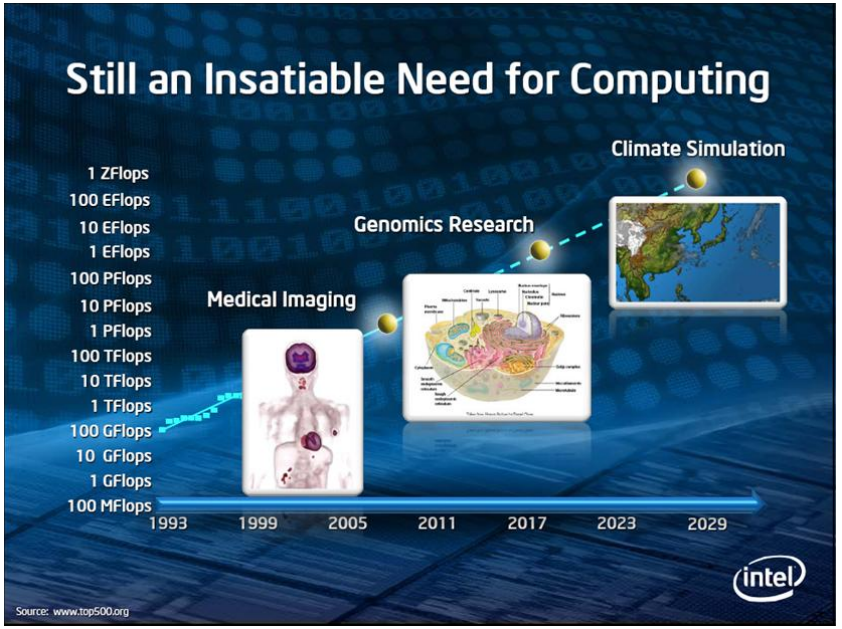
# Multiprozessorsysteme

## Einordnung:

- Klassifikation nach Flynn: MIMD-Rechner

## Warum Multiprozessorsysteme?

- Hohe Anforderungen von Anwendungen an die Rechenleistung
- Technisch-wissenschaftlicher Bereich
  - Rechnergestützte Simulation
  - Strömungsmechanik
  - Modellierung der globalen klimatischen Veränderungen
  - Evolution von Galaxien
  - Struktur von Materialien
  - .....
- Kommerzieller Bereich
  - Server, Datenbank-Anwendungen,
  - WEB




http://download.intel.com/pressroom/archive/reference/ISC_2010_Skaugen_keynote.pdf

# Multiprozessorsysteme

## Motivation

- Höchstleistungsrechner:
  - TOP500-Liste
    - Führt die schnellsten Rechner der Welt auf
    - Erscheint immer im Juni und im November eines Jahres
    - http://www.top500.org
    - Beispiel: TOP500 Liste (November 2010)

| | NAME/MANUFACTURER/COMPUTER | LOCATION | COUNTRY | CORES | $R_{max}$ Pflop/s |
|---|---|---|---|---|---|
| 1 | **Tianhe-1A** NUDT 6-core Intel X5670 2.93 GHz + Nvidia M2050 GPU w/custom interconnect | NUDT/NSCC/Tianjin | China | 186,368 | 2.57 |
| 2 | **Jaguar** Cray XT-5 6-core AMD 2.6 GHz w/custom interconnect | DOE/SC/ORNL | USA | 224,162 | 1.76 |
| 3 | **Nebulae** Dawning TC3600 Blade Intel X5650 2.67 GHz, NVidia Tesla C2050 GPU w/ Iband | NSCS | China | 120,640 | 1.27 |
| 4 | **Tsubame 2.0** HP Proliant SL390s G7 nodes (Xeon X5670 2.93GHz) , NVIDIA Tesla M2050 GPU w/Iband | TiTech | Japan | 73,278 | 1.19 |
| 5 | **Hopper** Cray XE-6 12-core AMD 2.1 GHz w/custom interconnect | DOE/SC/LBNL | USA | 153,408 | 1.05 |

Quelle: http://www.top500.org

# Multiprozessorsysteme

## Motivation

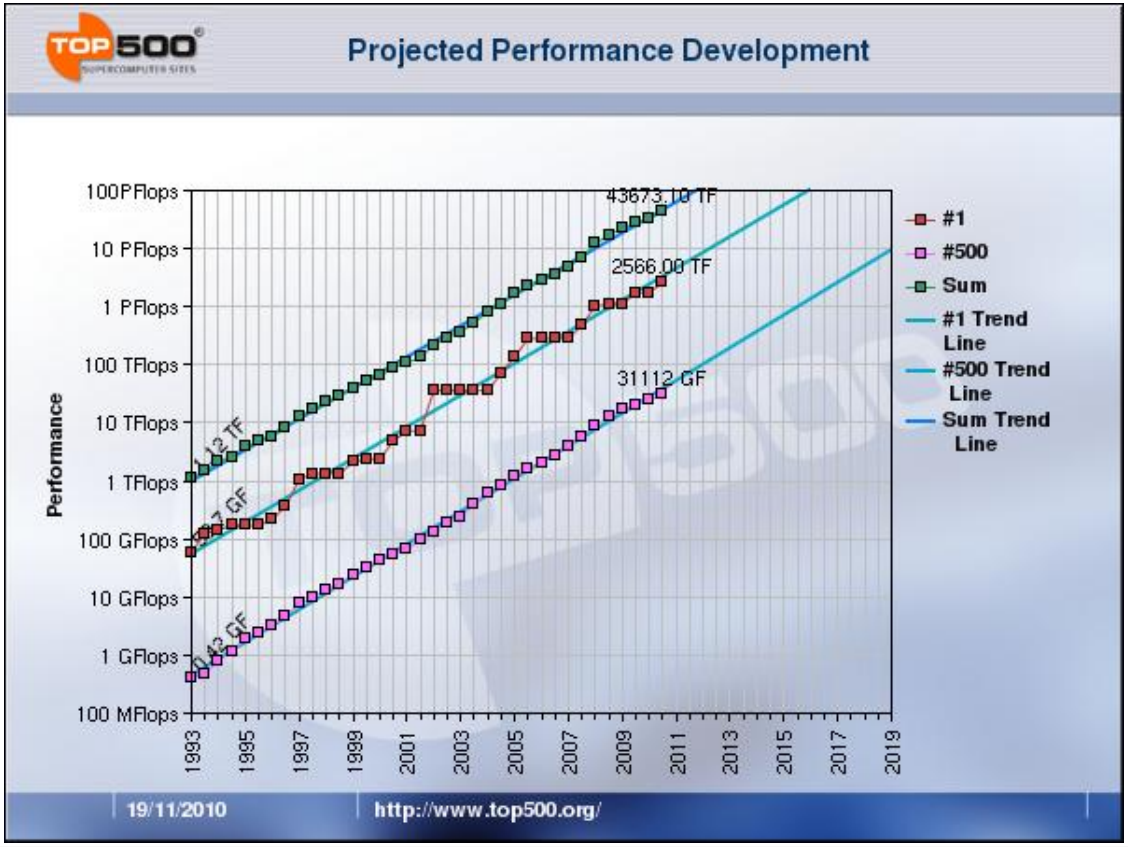
- Höchstleistungsrechner:
  - TOP500-Liste


Quelle: http://www.top500.org

# Multiprozessorsysteme

## Motivation

- Höchstleistungsrechner:
  - TOP500-Liste

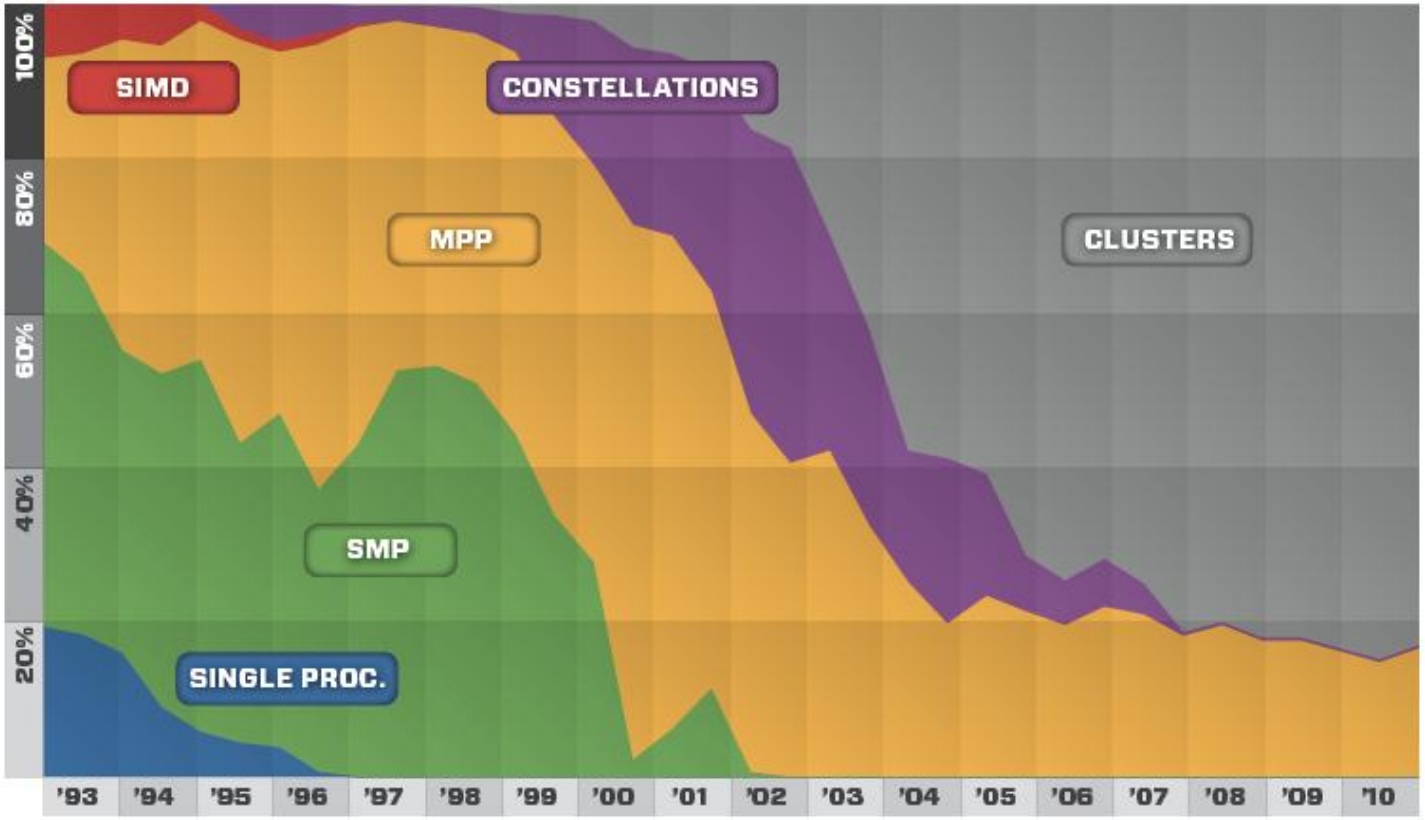



Quelle: http://www.top500.org

# Multiprozessorsysteme

## Motivation

- Höchstleistungsrechner:
  - TOP500-Liste: **Architektur**

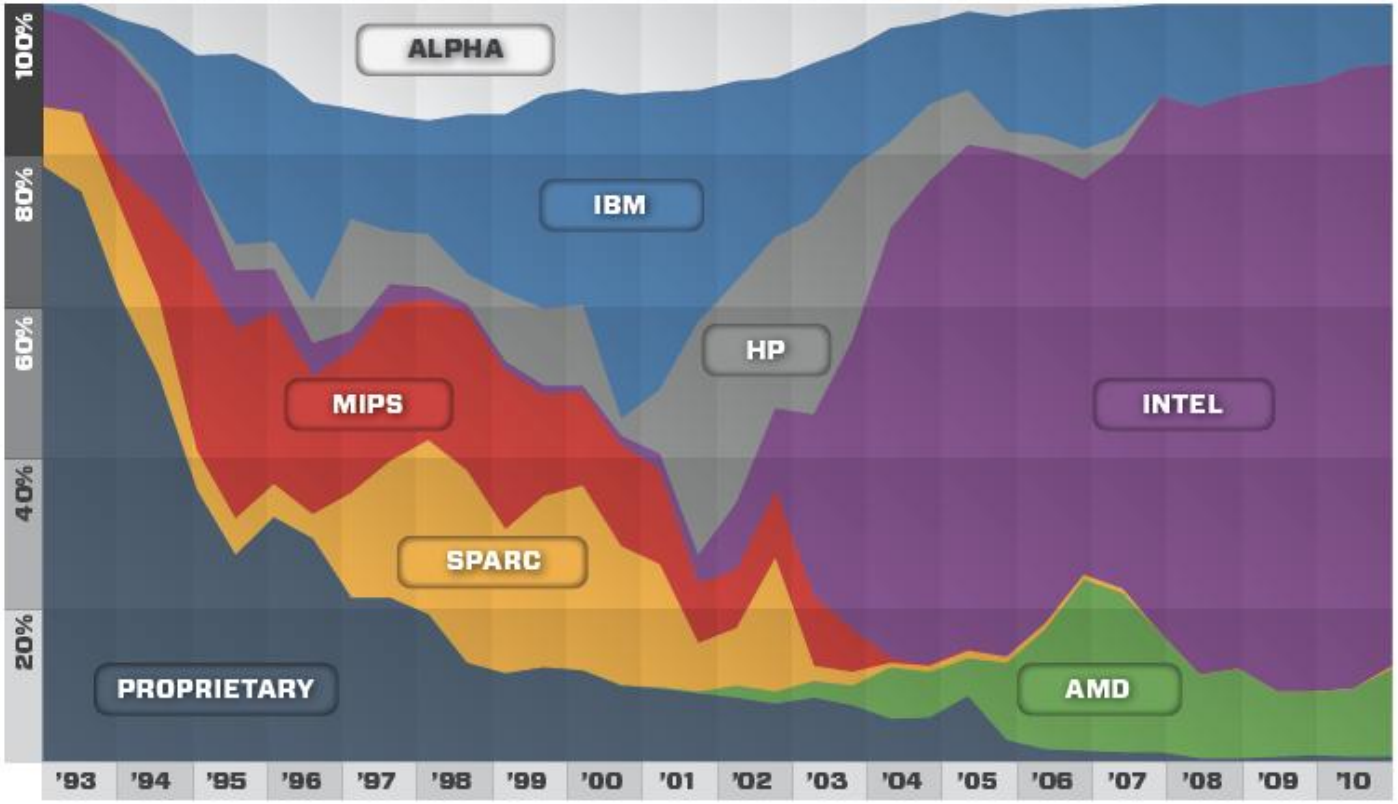



Quelle: http://www.top500.org

# Multiprozessorsysteme

## Motivation

- Höchstleistungsrechner:
  - TOP500-Liste: **Prozessortyp**

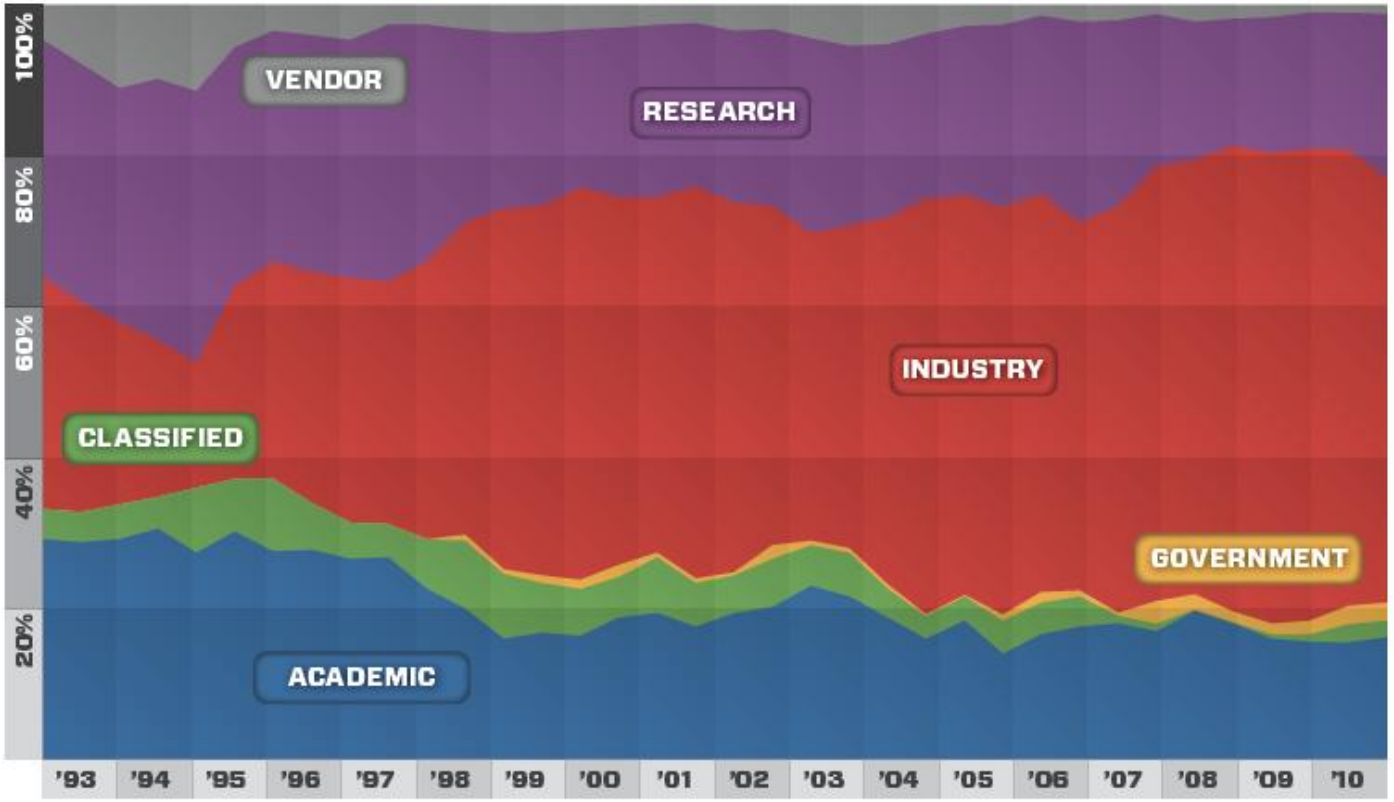



Quelle: http://www.top500.org

# Multiprozessorsysteme

## Motivation

- Höchstleistungsrechner:
  - TOP500-Liste: **Prozessortyp**




Quelle: http://www.top500.org

# Multiprozessorsysteme

## Motivation

- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 1 (November 2009): Jaguar Cray XT5-HE

| | XT5 | XT4 | Total |
|---|---|---|---|
| Cabinets | 200 | 84 | 284 |
| Compute Blades | 4,672 | 1,958 | 6,630 |
| Quad-core Opteron Processors | 37,376 | 7,832 | 45,208 |
| Cores | 149,504 | 31,328 | 180,832 |
| Peak TeraFLOPS | 1,375 | 263 | 1,639 |
| Nodes | 18,688 | 7,832 | 26,520 |
| Memory (TB) | 300 | 62 | 362 |
| Number of disks | 13,440 | 2,774 | 16,214 |
| Disk Capacity (TB) | 10,000 | 750 | 10,750 |
| I/O Bandwidth (GB/s) | 240 | 44 | 284 |

Table 1 - Jaguar System Configuration

Bland A.S., Kendall R.A., Kothe D.B., Rogers J.H., Shipman G.M. Jaguar: The World's Most Powerful Computer, http://www.nccs.gov/computing-resources/jaguar/documentation/2009-cug-meeting/

# Multiprozessorsysteme

## Motivation

- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 1 (November 2009): Jaguar Cray XT5-HE
  - Massiv-paralleler Multiprozessor mit verteiltem Speicher
  - Knoten: symmetrischer Multiprozessor
    - „Each node has two AMD Opteron model 2356 2.3 GHz quad-core “Barcelona” processors
    - connected to each other through a pair of HyperTransport5 connections.
    - Each of the Opteron processors has 2 MB of level 3 cache shared among the four cores
    - and a DDR2 memory controller connected to a pair of 4 GB DDR2-800 memory modules.
    - The HyperTransport connections between the processors provide a cache coherent shared memory node with eight cores, 16 GB of memory, and 25.6 GB per second of memory bandwidth.
    - The node has a theoretical peak processing performance of 73.6 billion floating point operations per second (GF).

Bland A.S., Kendall R.A., Kothe D.B., Rogers J.H., Shipman G.M. Jaguar: The World’s Most Powerful Computer, http://www.nccs.gov/computing-resources/jaguar/documentation/2009-cug-meeting/

## Motivation

- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 1 (November 2009): Jaguar Cray XT5-HE

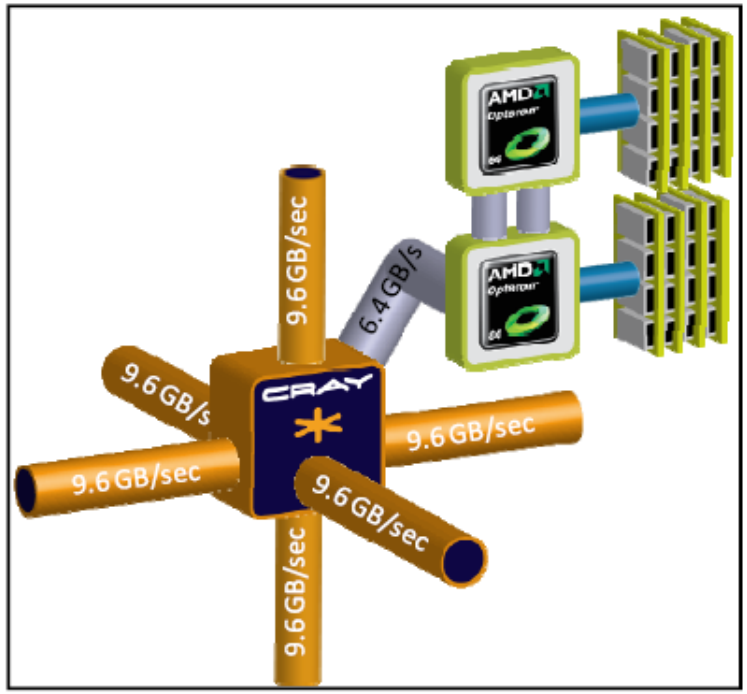


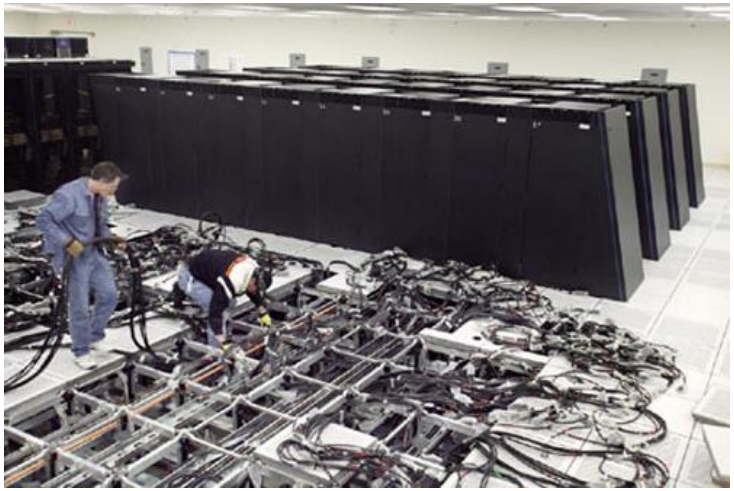

Figure 1 - XT5 Node Configuration

Bland A.S., Kendall R.A., Kothe D.B., Rogers J.H., Shipman G.M. Jaguar: The World's Most Powerful Computer, http://www.nccs.gov/computing-resources/jaguar/documentation/2009-cug-meeting/

# Multiprozessorsysteme

## Motivation

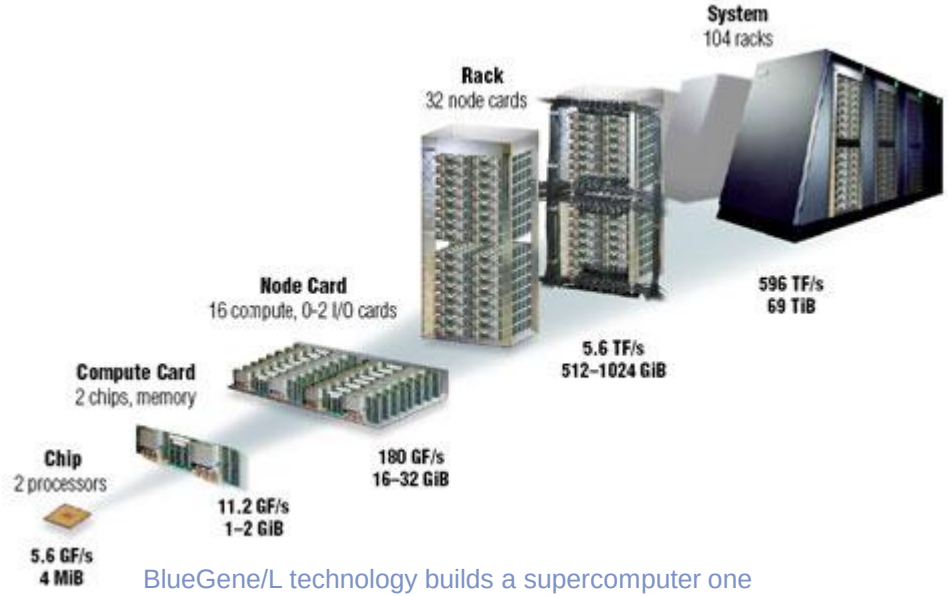
- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 12 (November 2010): BlueGene/L, Modell: eServer Blue Gene Solution
    - Standort:  DOE/NNSA/LLNL
    - Anzahl Prozessoren (PowerPC 440 700 MHz, 2.8 GFlops) : 212992
    - Speicher: 73728 GB
    - Leistung: 478200  GFLOPS (Linpack)
    - Installation: 2007



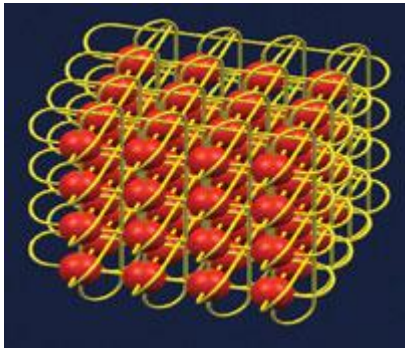
Quelle: https://asc.llnl.gov/computing_resources/bluegenel/photogallery.html

## Motivation

- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 12 (November 2010): BlueGene/L, Modell: eServer Blue Gene Solution




BlueGene/L technology builds a supercomputer one dual-processor chip at a time. Chips are aggregated into compute cards, which are then assembled into node cards. Each rack holds 2 node cards, and the full machine now comprises 104 racks.



BlueGene/L uses a three-dimensional (3D) torus network in which the nodes (red balls) are connected to their six nearest-neighbor nodes in a 3D mesh. In the torus configuration, the ends of the mesh loop back, thereby eliminating the problem of programming for a mesh with edges. Without these loops, the end nodes would not have six near neighbors.

Quelle: https://asc.llnl.gov/computing_resources/bluegenel/photogallery.html

# Multiprozessorsysteme

## Motivation

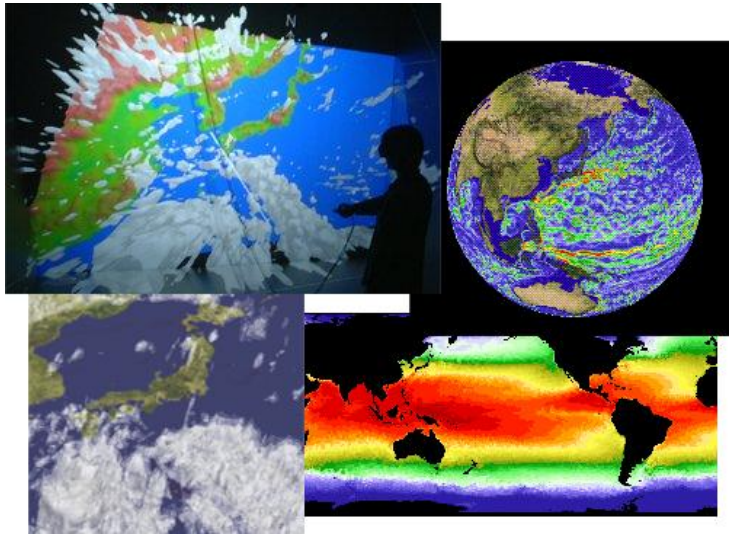
- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 9 (November 2010): JUGENE - Blue Gene/P Solution
    - Rechenleistung: 1 Petaflop/s
    - Prozessoren: 73 728 / Cores: 294 912
      - Prozessortyp: 32-bit PowerPC 450 core 850 MHz
    - Compute node: 4-way SMP processor
    - Hauptspeicher: 144 Terabyte,
    - Racks: 72
    - Leistungsaufnahme: 2,2 Megawatt


http://www.fz-juelich.de/portal/DE/Forschung/Informationstechnologie/Supercomputer/JUGENE.html?nn=363164

# Multiprozessorsysteme

## Motivation

- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 1 (November 2002): Earth Simulator
    - Anzahl Prozessoren: 5120
    - Leistung: 35,86 TFLOPS (Linpack),
    - Anwendung: Klimaforschung

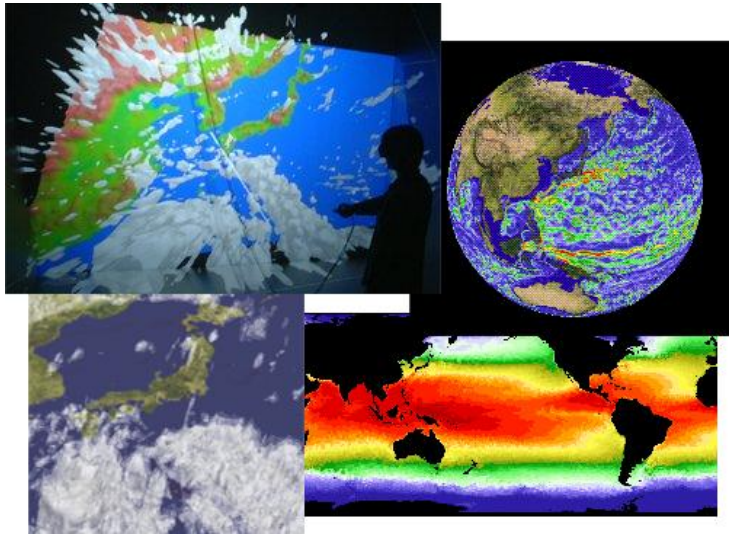


Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

# Multiprozessorsysteme

## Motivation

- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 1 (November 2002): Earth Simulator
    - Anzahl Prozessoren: 5120
    - Leistung: 35,86 TFLOPS (Linpack),
    - Anwendung: Klimaforschung

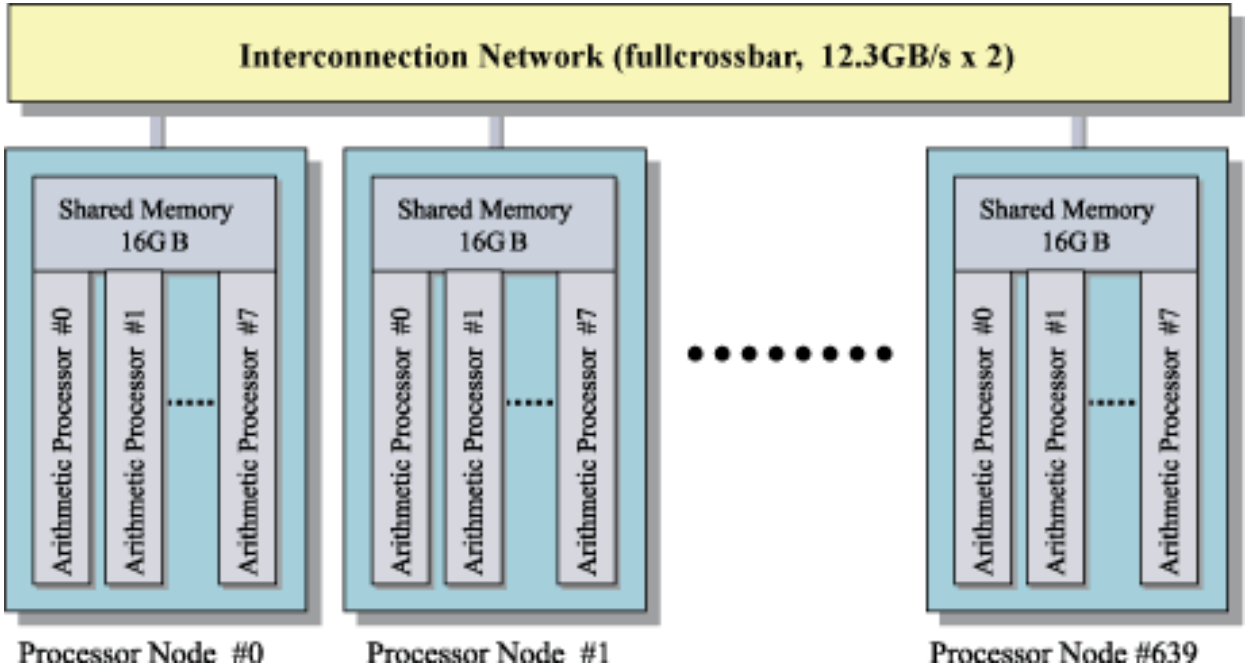


Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

# Multiprozessorsysteme

## Motivation

- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 1 (November 2002): Earth Simulator
    - Konfiguration

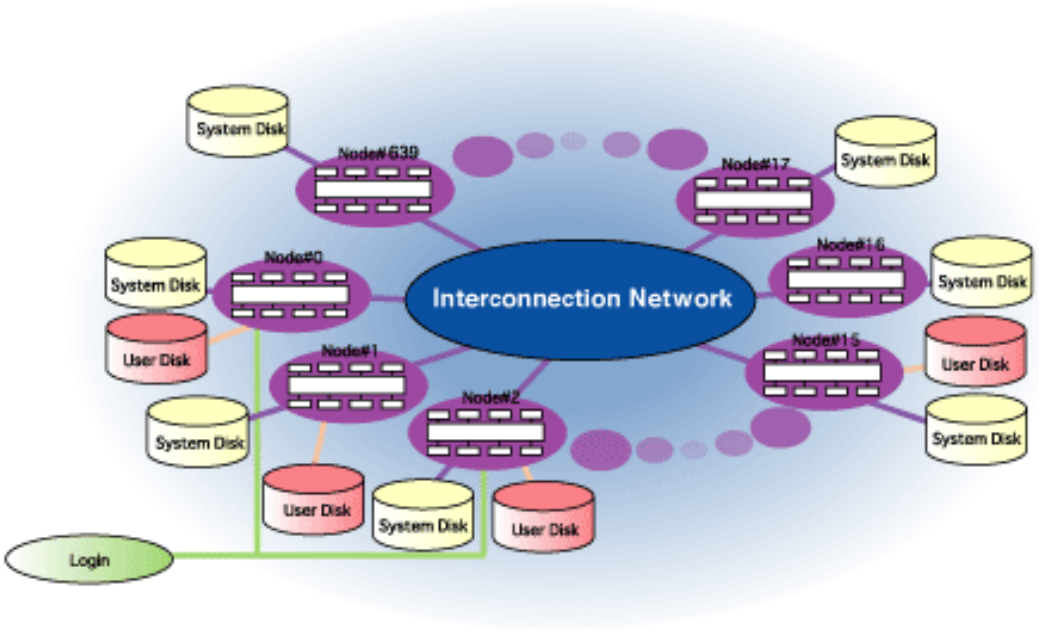



Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

# Multiprozessorsysteme

## Motivation

- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 1 (November 2002): Earth Simulator
    - Konfiguration

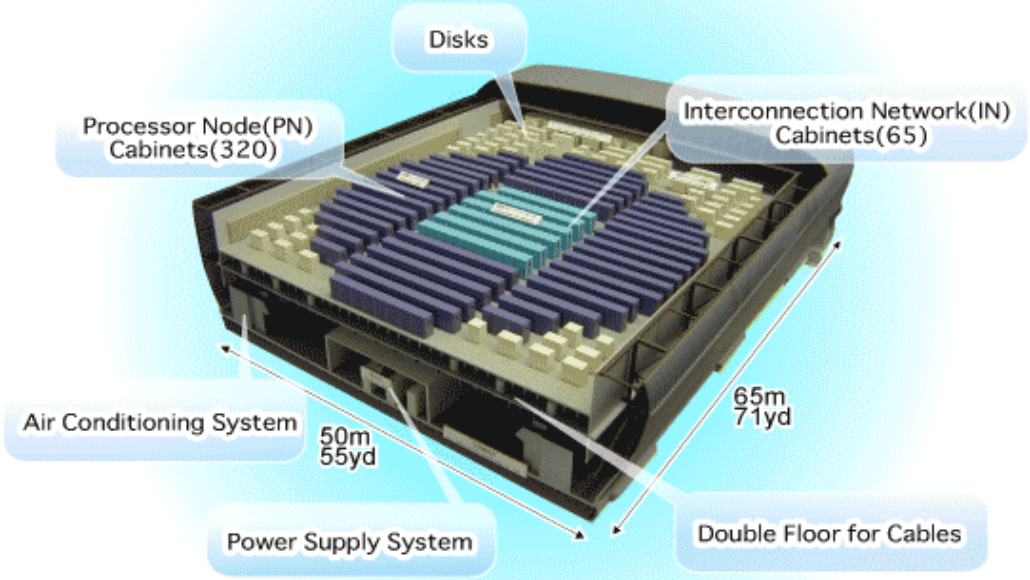



Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

# Multiprozessorsysteme

## Motivation

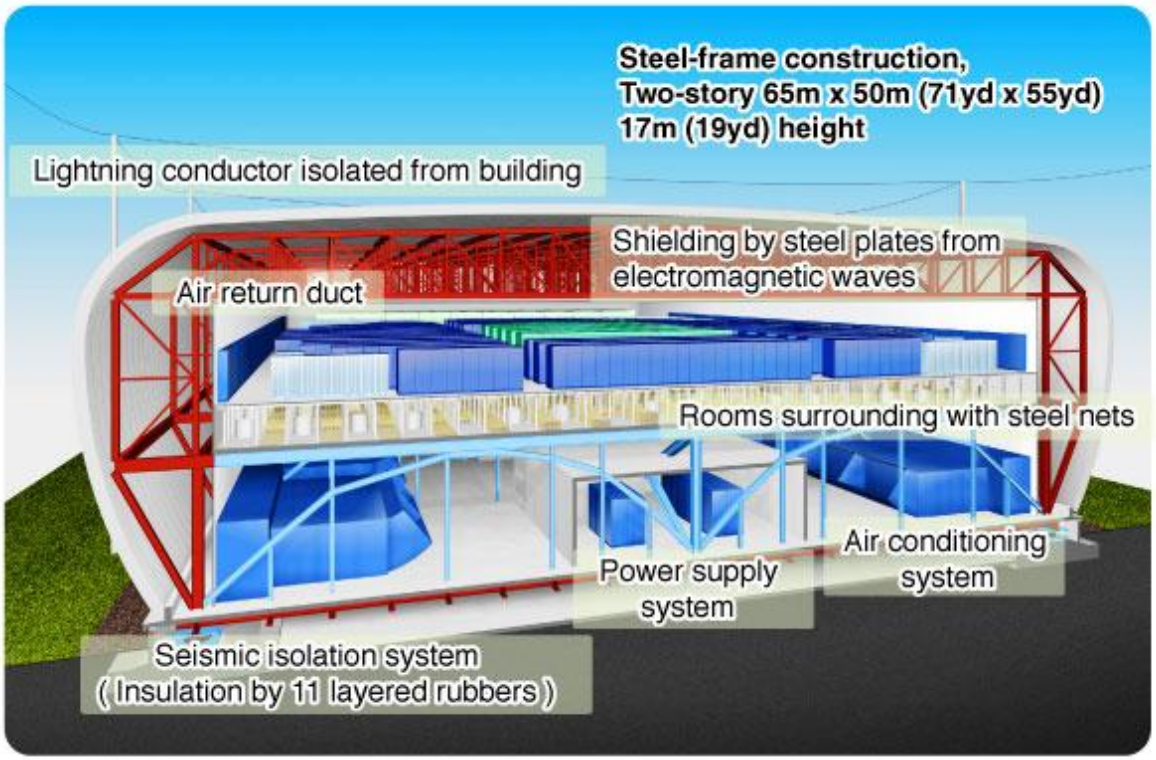
- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 1 (November 2002): Earth Simulator

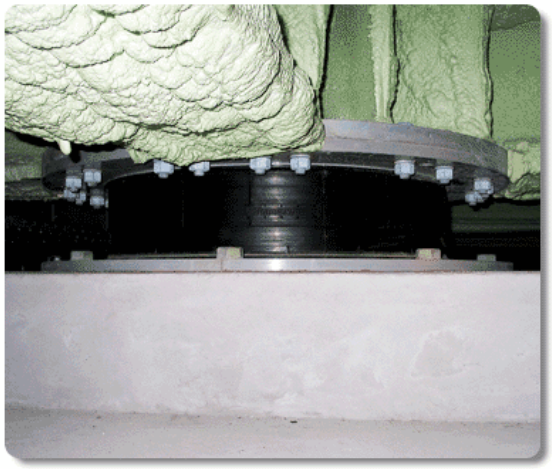



Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

## Motivation

- Beispiele von Höchstleistungsrechnern
  - Top500 Nr. 1 (November 2002): Earth Simulator



Quelle: The Earth Simulator Center;
http://www.es.jamstec.go.jp/esc/research/Perception/index.en.html

# Multiprozessorsysteme

## Motivation

- Beispiele von Höchstleistungsrechnern
  - Barcelona Supercomputer Center: MareNostrum - BladeCenter JS21 Cluster, PPC 970 (10240) 2.3 GHz, Myrinet

By courtesy of Barcelona Supercomputing Center - www.bsc.es

# Vorlesung Rechnerstrukturen

## Kapitel 3: Multiprozessoren – Parallelismus auf Prozess-/ Blockebene

- 3.1 Motivation
- 3.2 Allgemeine Grundlagen

## Parallele Architekturen

- Definition Parallelrechner:
  - *„A collection of processing elements that communicate and cooperate to solve large problems“* (Almase and Gottlieb, 1989)
  - Betrachtung einer parallelen Architektur als eine Erweiterung des Konzepts einer konventionellen Rechnerarchitektur um eine Kommunikationsarchitektur

# Parallele Architekturen

## Rechnerarchitektur

- **Abstraktion**
  - Benutzer-/System-Schnittstelle
  - Hardware-/Software-Schnittstelle
- **Architektur**
  - Spezifiziert die Menge der Operationen an den Schnittstellen und die Datentypen, auf denen diese operieren
- **Organisation**
  - Realisierung der Abstraktionen

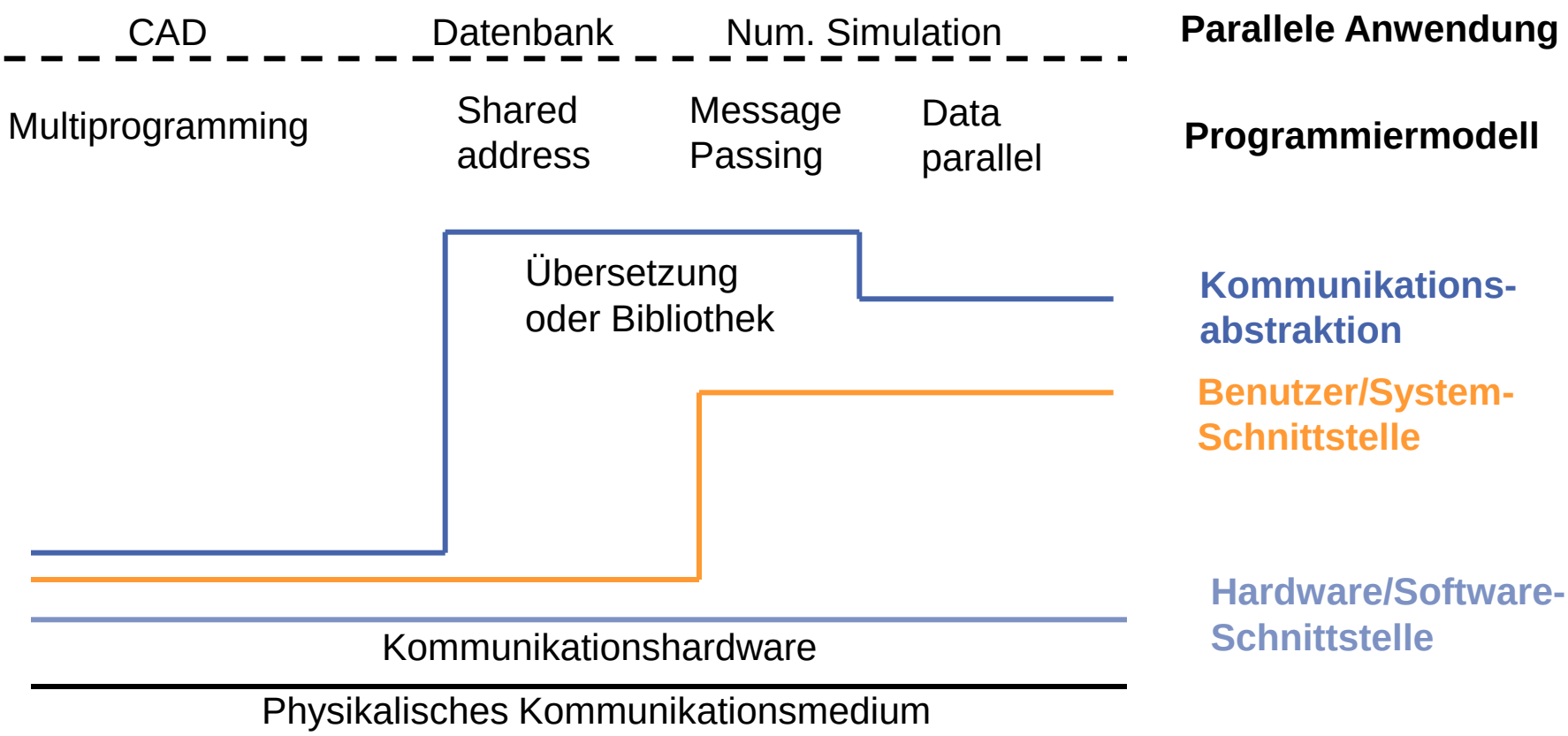
# Parallele Architekturen

## Kommunikationsarchitektur

- **Abstraktion**
  - Benutzer-/System-Schnittstelle
  - Hardware-/Software-Schnittstelle
- **Architektur**
  - Spezifiziert die Kommunikations- und Synchronisationsoperationen
- **Organisation**
  - Realisierung dieser Operationen

# Parallele Architekturen

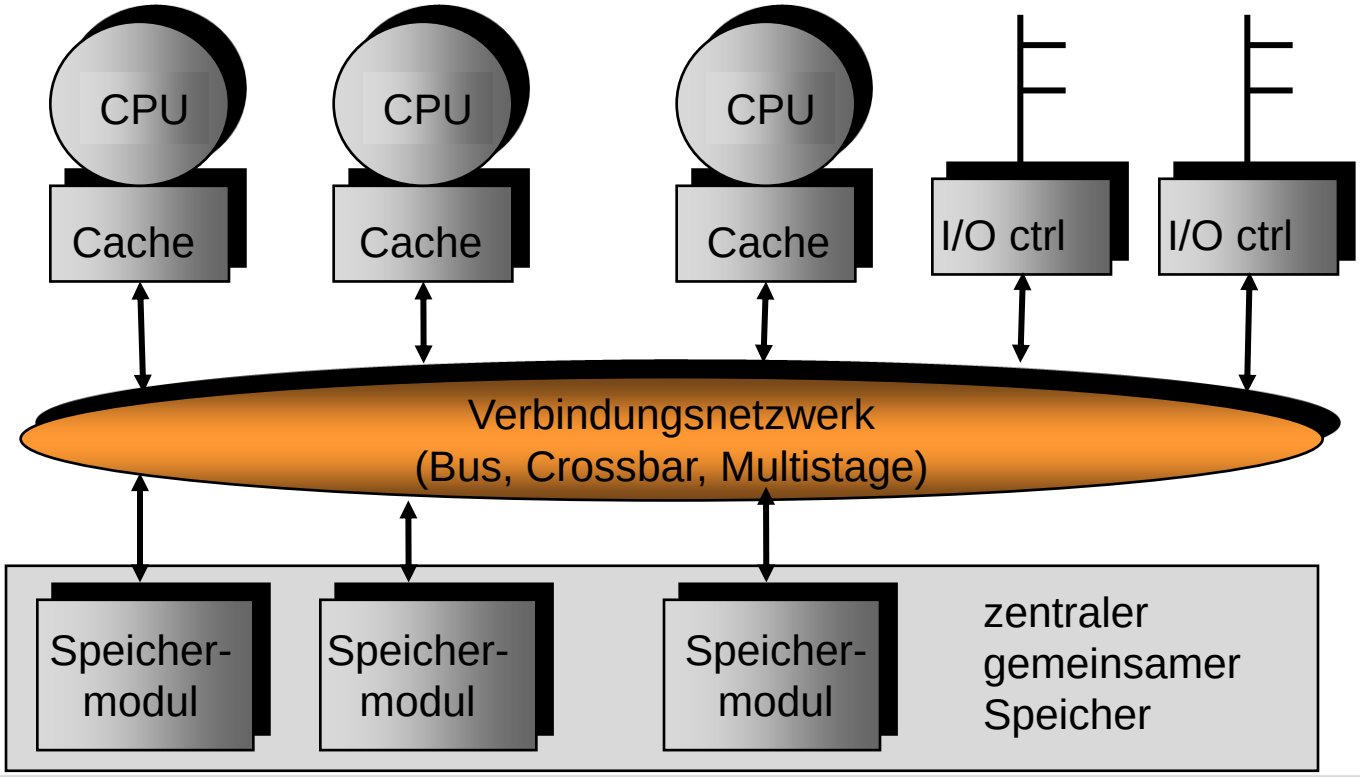
## Abstraktion



# Parallele Architekturen

## Multiprozessor mit gemeinsamem Speicher

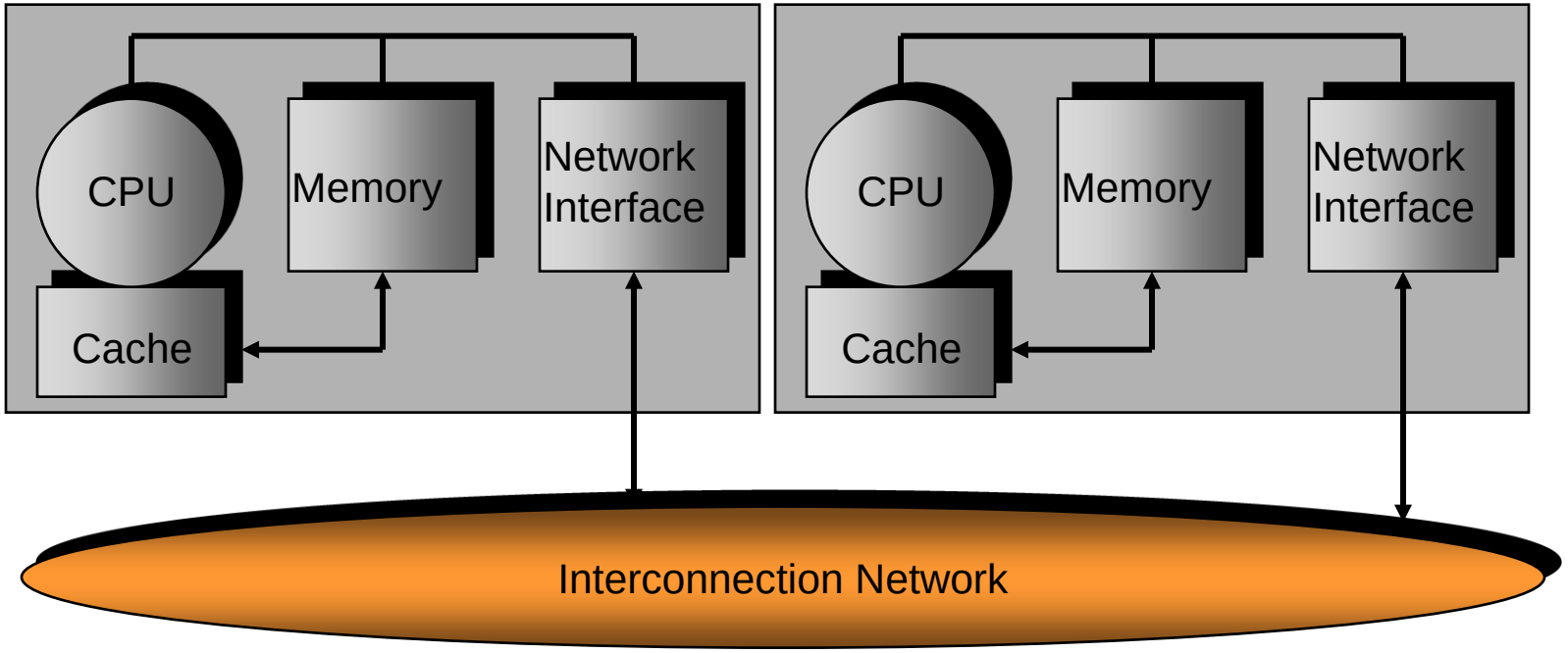
- **UMA**: Uniform Memory Access
- Beispiel: **symmetrischer Multiprozessor (SMP)**
  - Gleichberechtigter Zugriff der Prozessoren auf die Betriebsmittel



# Parallele Architekturen

## Multiprozessor mit verteiltem Speicher

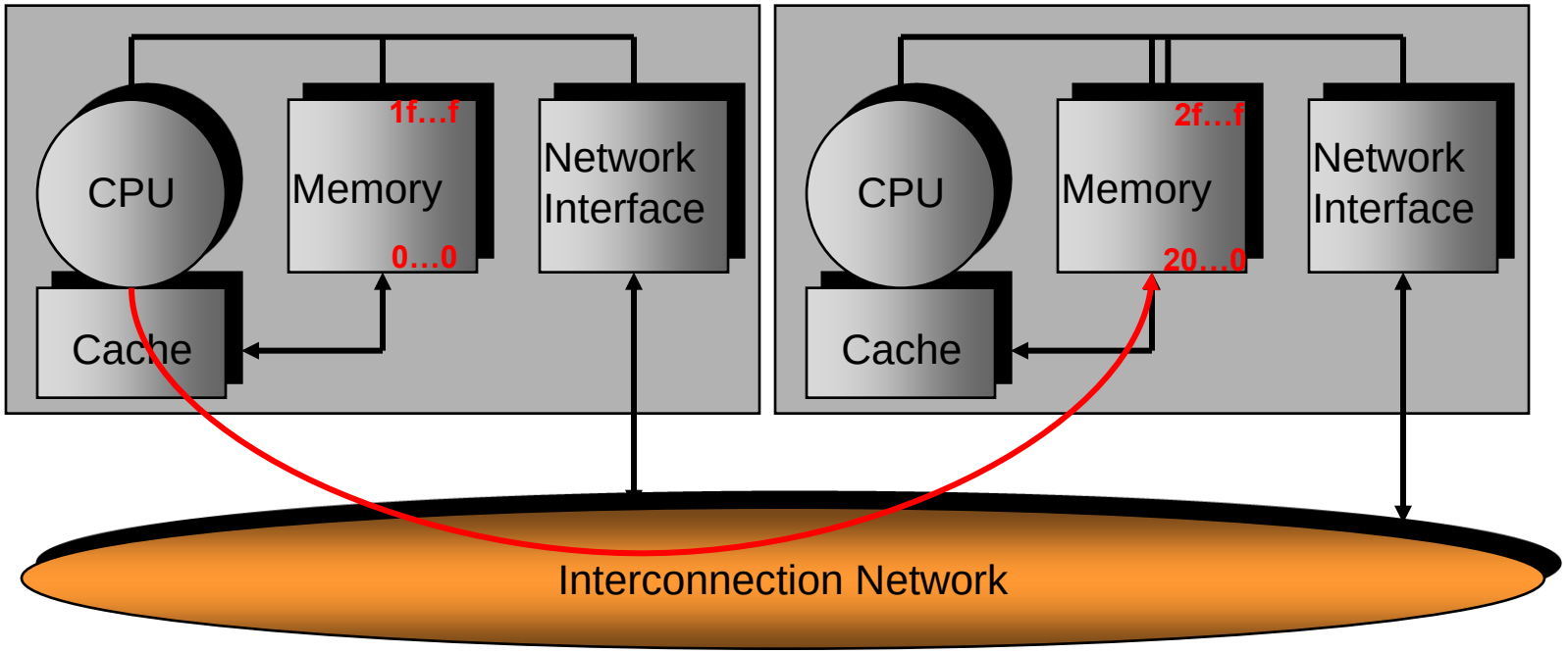
- **NORMA** No Remote Memory Access
- Beispiel: **Cluster**



# Parallele Architekturen

## Multiprozessor mit verteiltem gemeinsamen Speicher

- **NUMA**: Non-Uniform Memory Access
- **CC-NUMA**: Cache-Coherent Non-Uniform Memory Access
  - Globaler Adressraum: Zugriff auf entfernten Speicher



# Parallele Architekturen

## Programmiermodell

- Abstraktion einer parallelen Maschine, auf der der Anwender sein Programm formuliert
- Spezifiziert, wie Teile des Programms parallel abgearbeitet werden, wie Informationen ausgetauscht werden und welche Synchronisationsoperationen verfügbar sind, um die Aktivitäten zu koordinieren
- Anwendungen werden auf der Grundlage eines parallelen Programmiermodells formuliert

# Parallele Architekturen

## Programmiermodell

- **Multiprogramming**
  - Menge von unabhängigen sequentiellen Programmen
  - Keine Kommunikation oder Koordination

# Parallele Architekturen

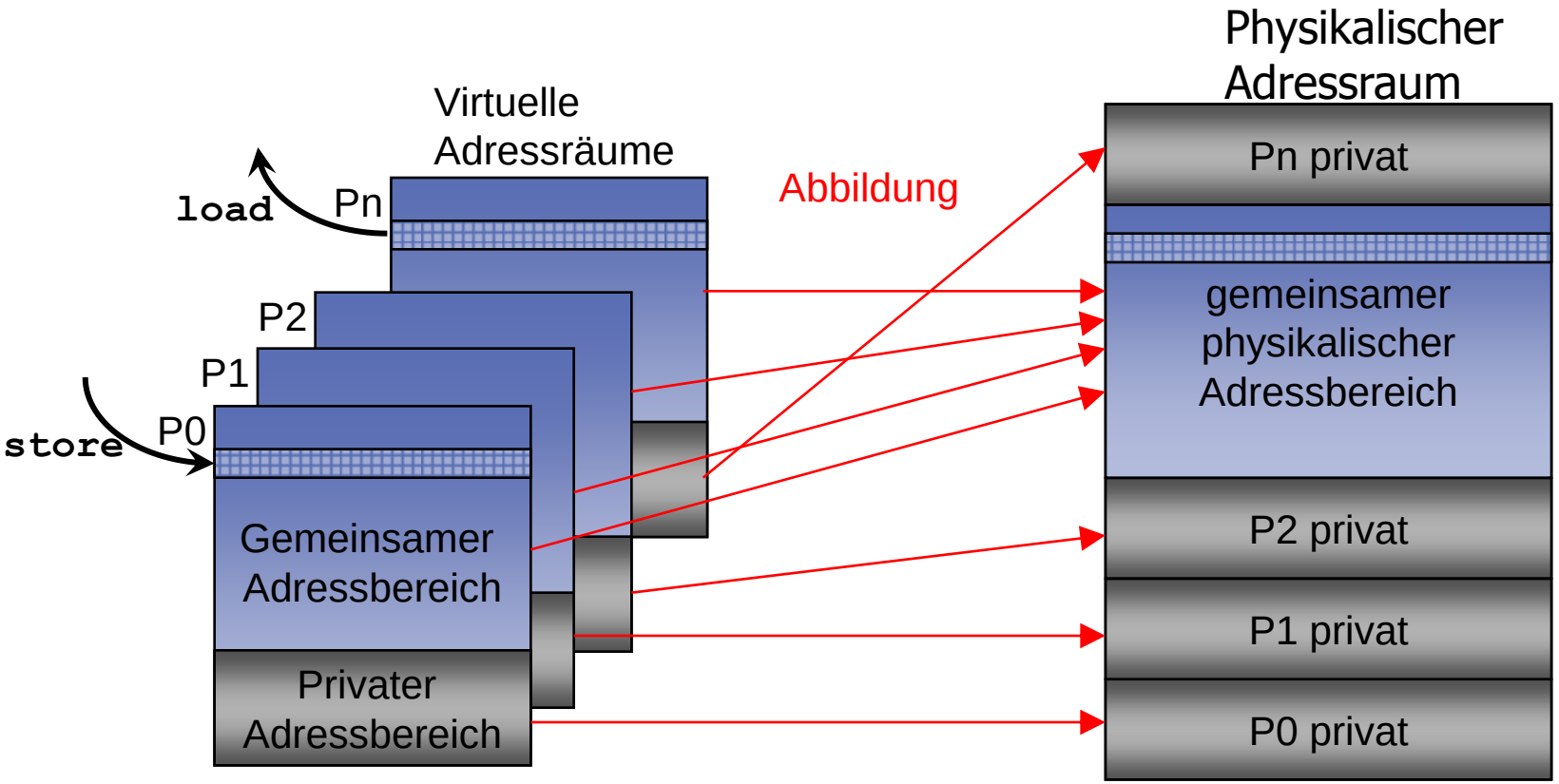
## Programmiermodell

- **Gemeinsamer Speicher (Shared Memory)**
  - Kommunikation und Koordination von Prozessen (Threads) über **gemeinsame Variablen** und Zeiger, die gemeinsame Adressen referenzieren

  - **Kommunikationsarchitektur**
    - Verwendung konventioneller Speicheroperationen für die Kommunikation über gemeinsame Adressen
    - Atomare Synchronisationsoperationen

## Programmiermodell

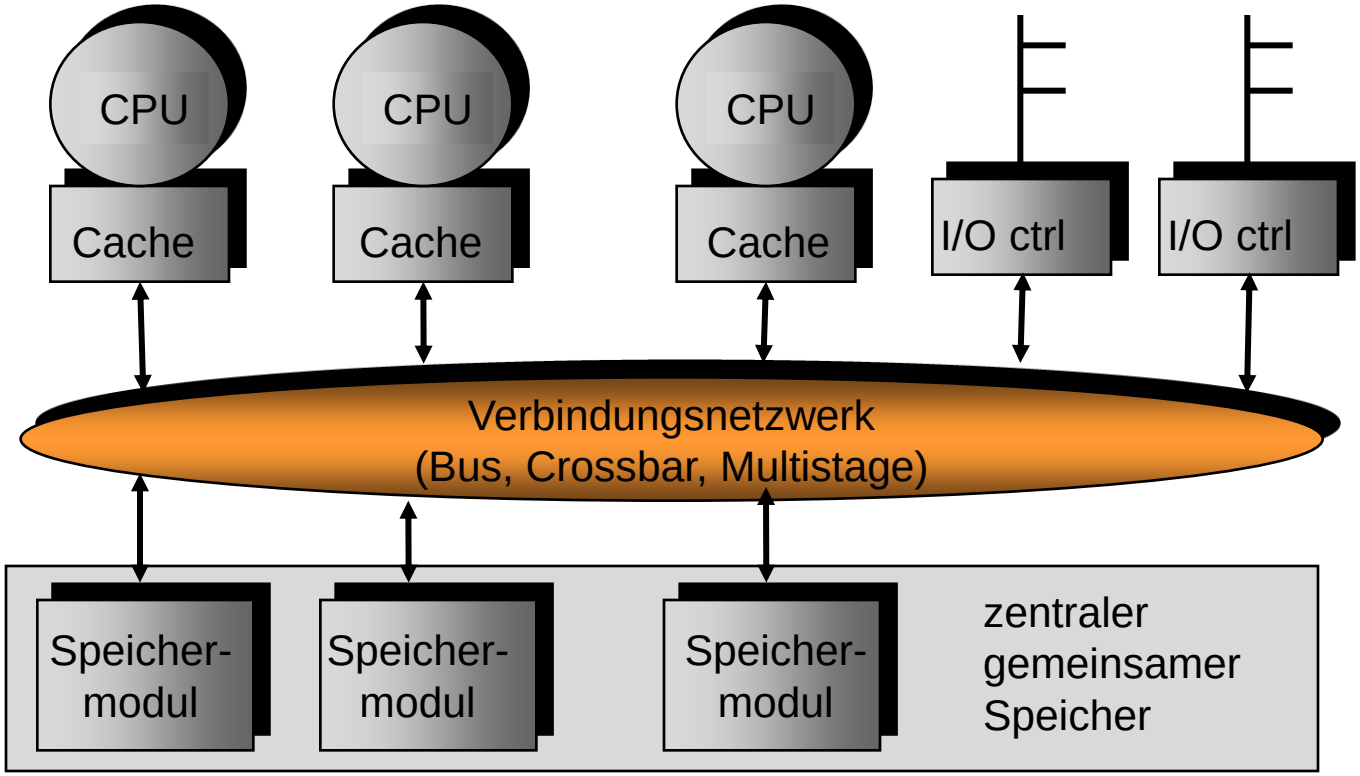
- **Gemeinsamer Speicher (Shared Memory)**



# Parallele Architekturen

## Multiprozessor mit gemeinsamem Speicher

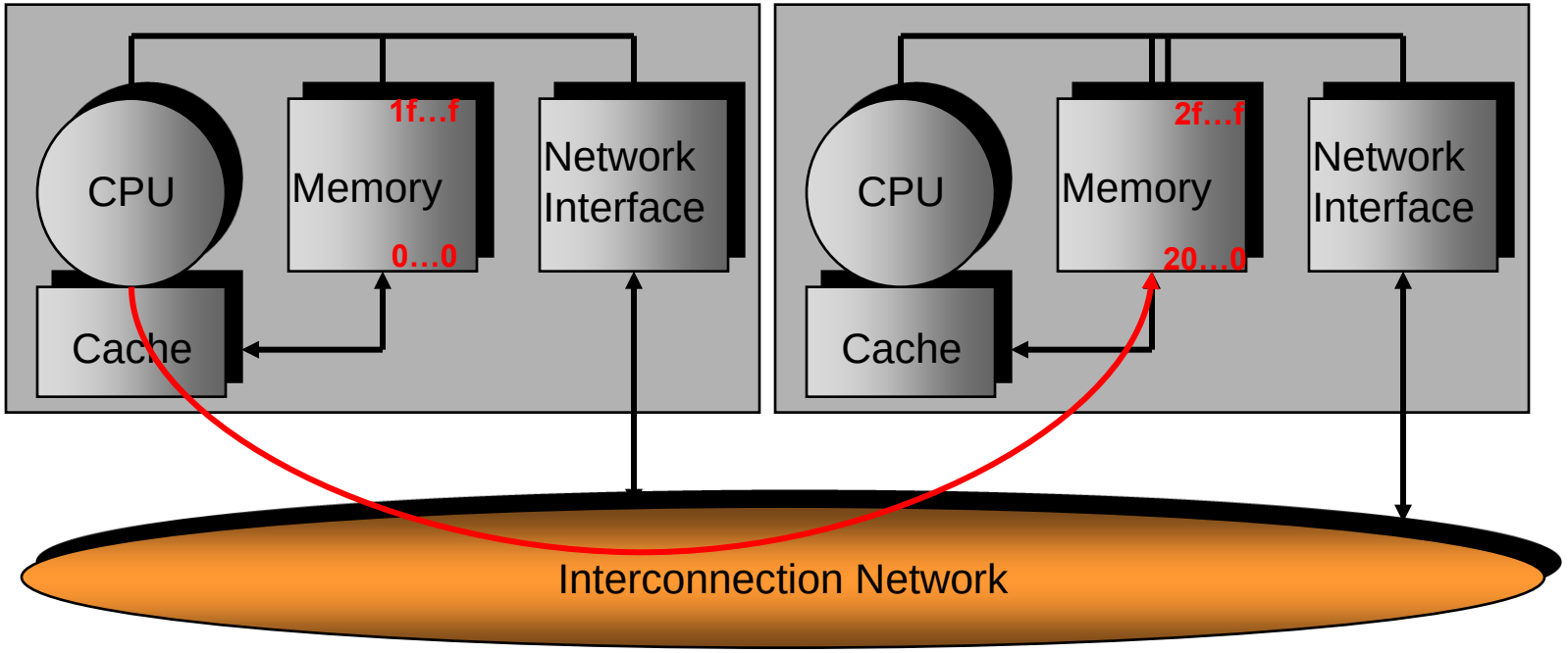
- UMA: Uniform Memory Access



# Parallele Architekturen

## Multiprozessor mit verteiltem gemeinsamen Speicher

- NUMA: Non-Uniform Memory Access
- CC-NUMA: Cache-Coherent Non-Uniform Memory Access
  - Globaler Adressraum: Zugriff auf entfernten Speicher



# Parallele Architekturen

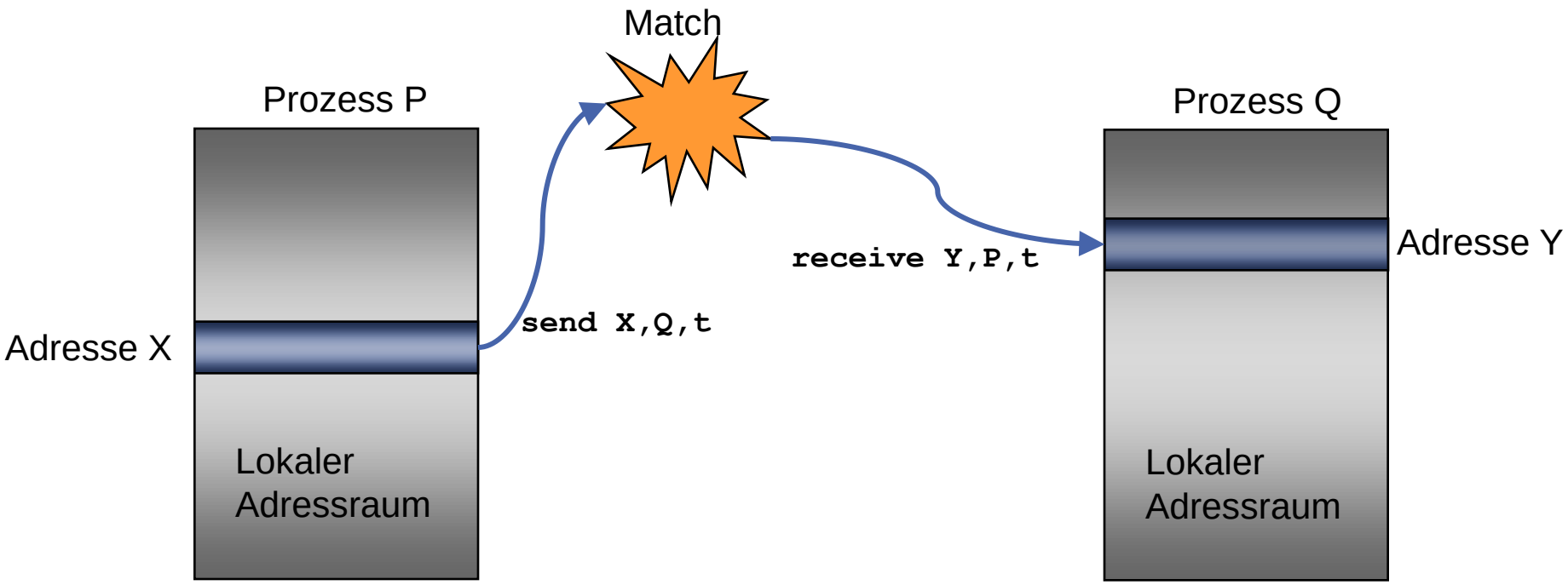
## Programmiermodell

- **Nachrichtenorientiertes Programmiermodell (Message Passing)**
  - Kommunikation der Prozesse (Threads) mit Hilfe von **Nachrichten**
    - Kein gemeinsamer Adressbereich
  - **Kommunikationsarchitektur**
    - Verwendung von korrespondierenden Send- und Receive-Operationen
    - **Send**: Spezifikation eines lokalen Datenpuffers und eines Empfangsprozesses (auf einem entfernten Prozessor)
    - **Receive**: Spezifikation des Sende-Prozesses und eines lokalen Datenpuffers, in den die Daten ankommen

# Parallele Architekturen

## Programmiermodell

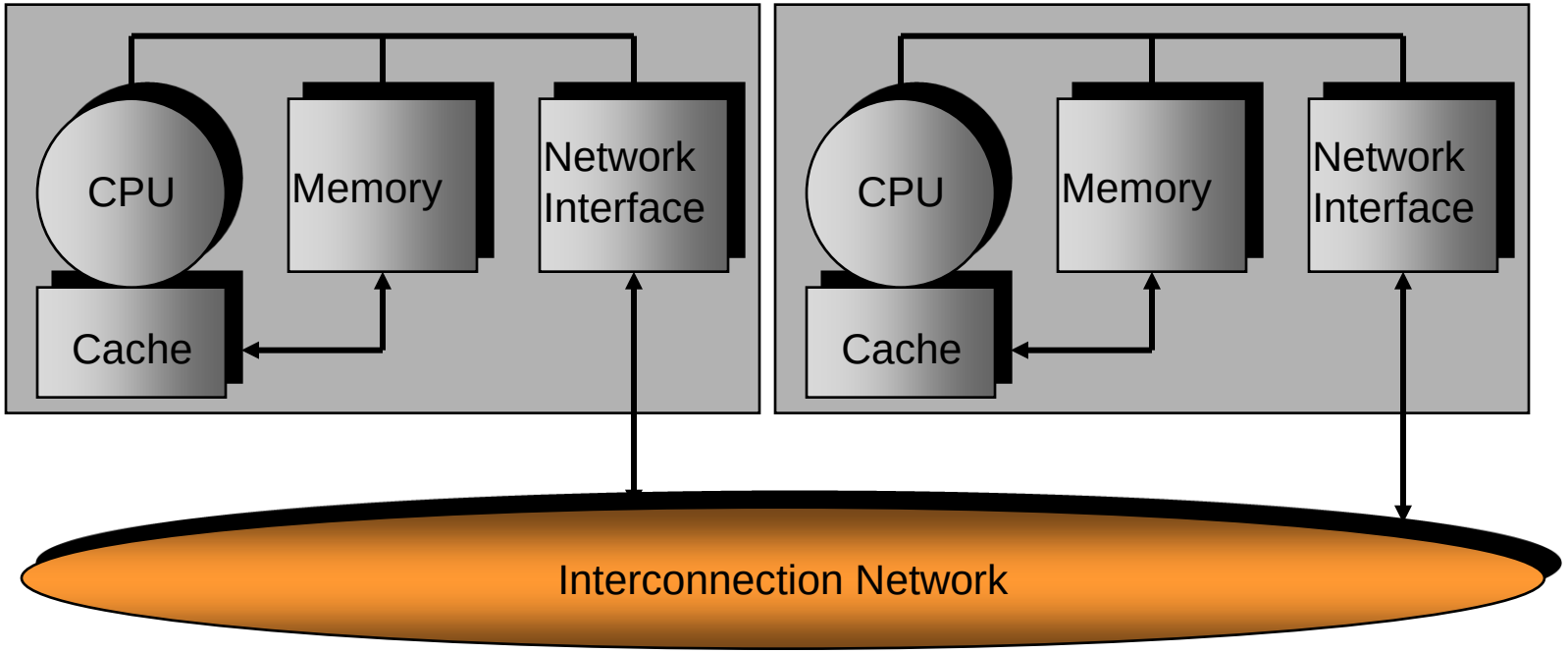
- **Nachrichtenorientiertes Programmiermodell (Message Passing)**



# Parallele Architekturen

## Multiprozessor mit verteiltem Speicher

- NORMA No Remote Memory Access



# Parallele Architekturen

## Programmiermodell

- **Datenparallelismus**
  - Gleichzeitige Ausführung von Operationen auf getrennten Elementen einer Datenmenge (Feld, Matrix)
  - Typischerweise in Vektorprozessoren

# Vorlesung Rechnerstrukturen

## Kapitel 3: Multiprozessoren – Parallelismus auf Prozess-/ Blockebene

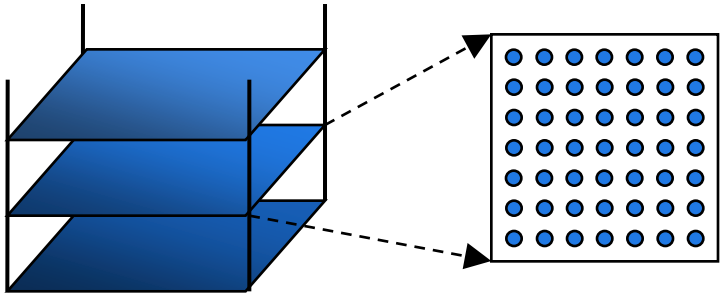
- 3.1 Motivation
- 3.2 Allgemeine Grundlagen
- 3.3 Parallele Programmierung

## Fallstudie: OCEAN - Simulation der Ozean-Strömung

- Benchmark-Programm aus der SPLASH-Benchmark-Suite
  - **Modellierung des Erdklimas**
    - Gegenseitige Beeinflussung der Atmosphäre und der Ozeane, die ¾ der Erdoberfläche ausmachen
  - **Simulation der Bewegung der Wasserströmung im Ozean**
    - Strömung entwickelt sich unter dem Einfluss mehrerer physikalischer Kräfte, einschließlich atmosphärischer Effekte, dem Wind und der Reibung am Grund des Ozeans;
    - Vertikale Reibung an den „Rändern“: führt zu Wirbelströmung
    - Ziel: Simulation dieser Wirbelströme über der Zeit

## Fallstudie: OCEAN - Simulation der Ozean-Strömung

- **Diskretisierung:**
  - **Raum**:
    - Modellierung des Ozeansbeckens als ein Gitter von diskreten Punkten
    - Jede Variable (Druck, Geschwindigkeit, …) hat einen Wert an jedem Gitterpunkt
    - Zweidimensionales Gitter:



Modellierung des Ozeans
in einem rechteckigen Becken

  - **Zeit**:
    - Endliche Folge von Zeitschritten

## Fallstudie: OCEAN - Simulation der Ozean-Strömung

- **Lösung der Bewegungsgleichungen:**
  - An allen Gitterpunkten in einem Zeitschritt
  - In jedem Zeitschritt werden die Variablen neu berechnet
  - Wiederholung der Berechnung mit jedem Zeitschritt
  - Jeder Zeitschritt besteht aus mehreren Phasen

## Fallstudie: OCEAN - Simulation der Ozean-Strömung

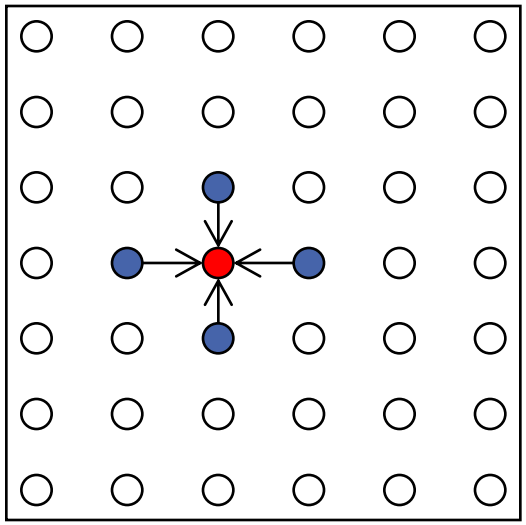
- **Lösung der Bewegungsgleichungen:**
  - Je mehr Gitterpunkte verwendet werden, desto feiner ist die Auflösung der Diskretisierung und desto genauer ist die Simulation
  - Für einen Ozean wie den Atlantik, der etwa eine Fläche von 2000km x 2000km umspannt bedeutet ein Gitter mit 100 x 100 Punkten eine Distanz von 20 km in jeder Dimension
  - Kürzere physikalische Intervalle zwischen den Zeitschritten führen zu einer höheren Simulationsgenauigkeit
  - Simulation der Ozeanbewegung über einen Zeitraum von 5 Jahren mit einer Aktualisierung des Zustands alle 8 Stunden erfordert 5500 Zeitschritte

## Fallstudie: OCEAN - Simulation der Ozean-Strömung

- **Lösung der Bewegungsgleichungen**
  - Lösung einer einfachen partiellen Differentialgleichung auf einem Gitter mit Hilfe einer finiten Differenzenmethode (Gauss-Seidel-Verfahren)
  - Reguläres zweidimensionales Gitter mit *(n+2)*(n+2)* Punkten (eine Ebene des Ozeanbeckens
  - Randwerte sind fest
  - Die inneren *n*n* Gitterpunkte werden mit Hilfe des Lösers berechnet, ausgehend von Anfangswerten

## Fallstudie: OCEAN - Simulation der Ozean-Strömung

- **Lösung der Bewegungsgleichungen**
  - Gitter



Berechnungsvorschrift für einen Gitterpunkt:

```
A[i,j]=0,2  x  (A[i,j]+
               +A[i,j-1]+A[i-1,j]
               +A[i,j+1]+A[i+1,j]
```

Wiederholte Berechnung, bis Verfahren konvergiert

## Fallstudie: OCEAN - Simulation der Ozean-Strömung

- **Sequentielle Version des Lösers:**

```
(1) int n,                                  /*size of matrix: (n+2)x(n+2)
(2) float **A, diff=0;
(3) main()
(4) begin
(5)   read(n)                               /*read input parameters*/
(6)   A←malloc (2-d array of size n+2 by n+2 doubles)
(7)   initialize(A);                        /*initialize matrix A*/
(8)   Solve (A);
(9)  end main
```

# Parallele Programmierung

## Fallstudie: OCEAN - Simulation der Ozean-Strömung

- **Sequentielle Version des Lösers:**

```
(1) procedure Solve (A)                    /*solve equation system*/
(2)   float **A;
(3) begin
(4)  int i,j,done=0
(5)  float temp;
(6)  while (!done) do                      /*outermost loop over sweeps*/
(7)    diff=0;                             /*initialize maximum diff*/
(8)    for i←1 to n do                     /*sweep over nonborder points*/
(9)       for j←1 to n do
(10)          temp=A[i,j];
(11)          A[i,j]←0.2*(A[i,j]+A[i,j-1]+A[i-1,j]+A[i,j+1]+A[i+1,j]);
(12)          diff += abs(A[i,j]-temp);
(13)       end for
(14)     end for
(15)     if (diff/(n*n) < TOL) then done=1;
(16)   end while
(17) end procedure
```

# Parallele Programmierung

## Parallelisierungsprozess

- Festlegen der Aufgaben, die parallel ausgeführt werden kann
  - Aufteilen der Aufgaben und der Daten auf Verarbeitungsknoten
    - Berechnung
    - Datenzugriff
    - Ein-/Ausgabe
  - Verwalten des Datenzugriffs, der Kommunikation und der Synchronisation
- Ziel: Hohe Leistung
  - Schnellere Lösung der parallelen Version gegenüber der sequentiellen Version
    - Ausgewogene Verteilung der Arbeit unter den Verarbeitungsknoten
    - Reduzierung des Kommunikations- und Synchronisationsaufwandes

# Parallele Programmierung

## Parallelisierungsprozess

- Ausführung der Schritte bei der Parallelisierung
  - Durch den Programmierer
  - Auf den verschiedenen Ebenen der Systemsoftware
    - Compiler
    - Laufzeitsystem
    - Betriebssystem
  - Ideal: Automatische Parallelisierung
    - Sequentielles Programm wird automatisch in ein effizientes paralleles Programm transformiert
    - Parallelisierende Compiler
    - Parallele Programmiersprachen
    - Noch nicht vollständig möglich!

## Parallelisierungsprozess

- **Definitionen**
  - **Task**:
    - Beliebige Aufgabe, die durch ein Programm auszuführen ist
    - Kleinste Parallelisierungseinheit
    - Möglichkeiten beim Beispiel Ocean:
      - Berechnung eines Gitterpunkts in jeder Berechnungsphase,
      - die Berechnung einer Reihe von Gitterpunkten,
      - die Berechnung einer beliebigen Teilmenge von Gitterpunkten
    - **Granularität**
      - grobkörnig
      - feinkörnig!

## Parallelisierungsprozess

- **Definitionen**
  - **Prozess** oder **Thread**
    - Paralleles Programm setzt sich aus mehreren kooperierenden Prozessen zusammen, von denen jeder eine Teilmenge der Tasks ausführt
    - Tasks werden über Prozessen zugewiesen
    - Beispiel Ocean:
      - Falls die Berechnung einer Reihe von Gitterpunkten als Task angesehen wird, dann kann eine feste Anzahl von Reihen einem Prozess zugewiesen werden
      - Aufteilung einer Ebene in mehrere Streifen
    - Kommunikation der Prozesse untereinander und Synchronisation
  - **Prozessor**
    - Ausführung eines Prozesses

# Parallele Programmierung

## Parallelisierungsprozess

- **Definitionen**
  - Unterscheidung Prozess und Prozessor
  - Prozessor:
    - Physikalische Ressource
  - Prozess
    - Abstraktion, Virtualisierung von einem Multiprozessor
    - Anzahl der Prozesse muss nicht gleich der Anzahl der Prozessoren eines Multiprozessorsystems sein

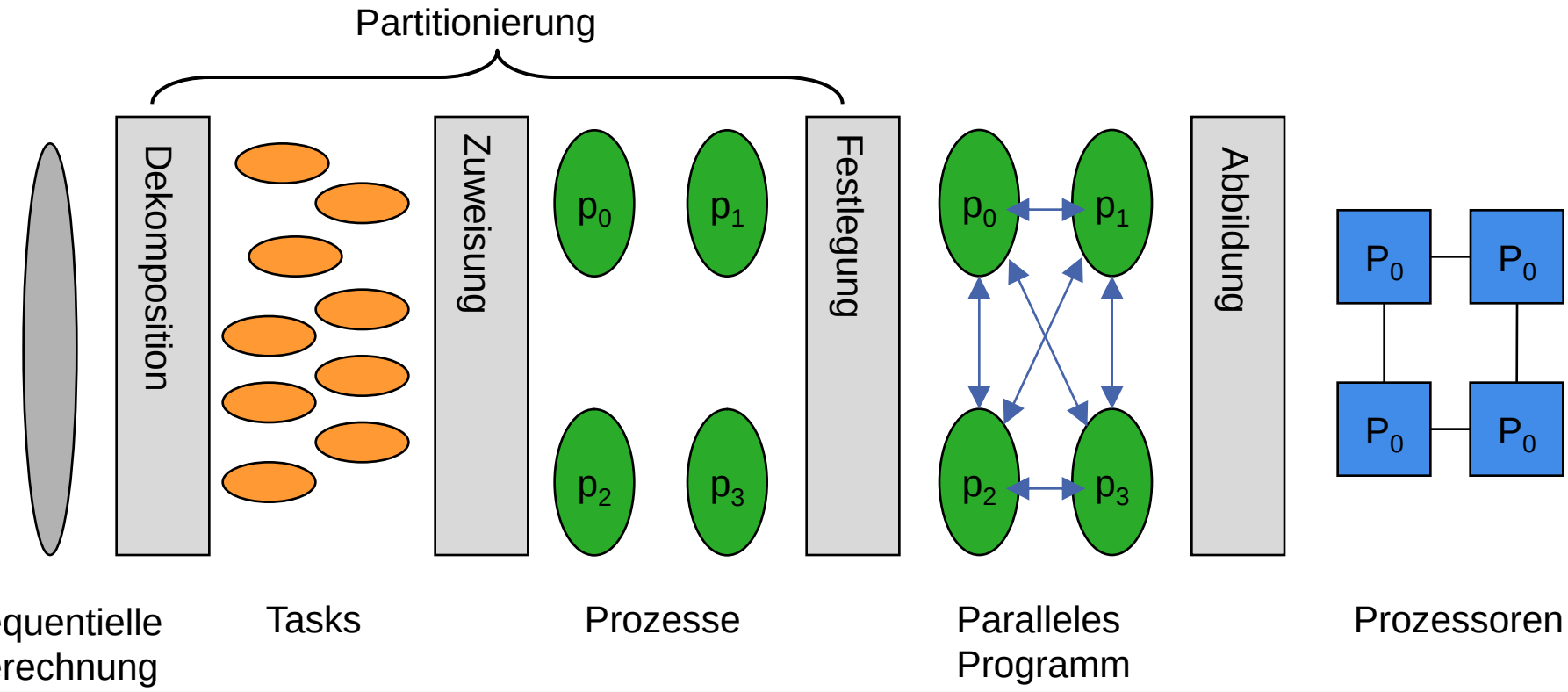
## Parallelisierungsprozess

- **Schritte bei der Parallelisierung**
  - Ausgangspunkt ist ein sequentielles Programm
  - **Aufteilung** oder **Dekomposition**
    - der Berechnung in Tasks
  - **Zuweisung**
    - der Tasks zu Prozessen
  - **Zusammenführung (Orchestration)**
    - Festlegung des notwendigen Datenzugriffs, der Kommunikation und der Synchronisation zwischen den Prozessen
  - **Abbildung**
    - der Prozesse an die Prozessoren

# Parallele Programmierung

## Parallelisierungsprozess

- Schritte bei der Parallelisierung und die Beziehung zwischen Tasks, Prozessen und Prozessoren



## Parallelisierungsprozess

- **Dekomposition**
  - Aufteilung der Berechnung in eine Menge von Tasks
    - Tasks können dynamisch während der Ausführung generiert werden
    - Anzahl der Tasks kann während der Ausführung variieren
    - Maximale Anzahl der Tasks, die zu einem Zeitpunkt zur Ausführung verfügbar sind, ist eine obere Grenze für die Anzahl der Prozesse, die effektiv genutzt werden können
  - Ziel:
    - Finden von parallel ausführbaren Anteilen
    - Verwaltungsaufwand (Overhead) gering halten

## Parallelisierungsprozess

- **Dekomposition**
  - **Beispiel: Ocean**
    - Programm strukturiert in geschachtelten Schleifen
      - Betrachtung einzelner Schleifen oder der geschachtelten Schleifen
      - Können Iterationen parallel ausgeführt werden?
      - Betrachtung über Schleifengrenzen

## Parallelisierungsprozess

- **Dekomposition**
  - **Beispiel: Ocean**
    - Betrachtung einzelner Schleifen oder der geschachtelten Schleifen

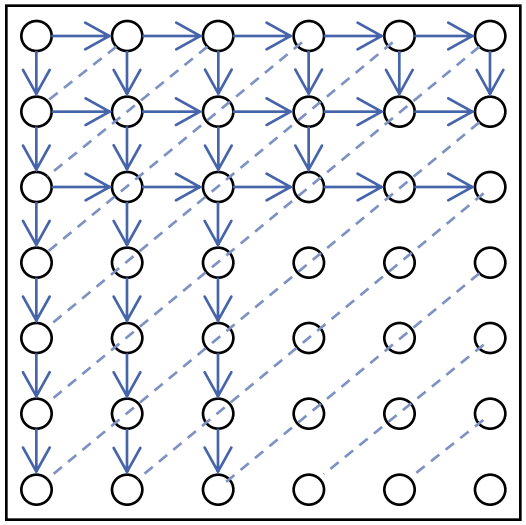
```
(1) while (!done) do
(2)    diff=0;
(3)    for i←1 to n do
(4)       for j←1 to n do
(5)          temp=A[i,j];
(6)          A[i,j]←0.2*(A[i,j]+A[i,j-1]+A[i-1,j]+A[i,j+1]+A[i+1,j]);
(7)          diff += abs(A[i,j]-temp);
(8)       end for
(9)     end for
(10)     if (diff/(n*n) < TOL) then done=1;
(11) end while
```

## Parallelisierungsprozess

- **Dekomposition**
  - **Beispiel: Ocean**
    - Betrachtung einzelner Schleifen oder der geschachtelten Schleifen
      - Äußere Schleife (Zeile 1-11) durchläuft das gesamte Gitter → Iterationen sind nicht unabhängig, da Daten, die in einer Iteration geändert werden, in der nächsten Iteration gebraucht werden
      - Innere Schleifen (Zeile 3-9)→Iterationen sind sequentiell abhängig, da in jeder inneren Schleife A[i,j-1] gelesen wird, der in der vorherigen Iteration geschrieben wurde

## Parallelisierungsprozess

- **Dekomposition**
  - **Beispiel: Ocean**
    - Betrachtung der Abhängigkeiten (Granularität Gitterpunkte)



Abhängigkeiten

Verbinden Punkte, zwischen den keine Abhängigkeiten bestehen

## Parallelisierungsprozess

- **Dekomposition am Beispiel: Ocean**
  - Aufteilen der Arbeit in einzelne Gitterpunkte, so dass die Aktualisierung eines Gitterpunktes eine Task ist
  - 1. Möglichkeit:
    - Beibehalten der Schleifenstruktur
    - erfordert Punkt-zu-Punkt-Synchronisation wegen Beachtung der Abhängigkeiten
    - Der neue Wert eines Gitterpunktes in einem Durchlauf ist berechnet, bevor er von dem westlichen oder südlichen Punkten verwendet wird
    - Verschiedene Schleifenschachtelungen und verschiedene Durchläufe können gleichzeitig ausgeführt werden
    - Hoher Aufwand!

## Parallelisierungsprozess

- **Dekomposition am Beispiel: Ocean**
  - Aufteilen der Arbeit in einzelne Gitterpunkte, so dass die Aktualisierung eines Gitterpunktes eine Task ist
  - 2. Möglichkeit:
    - Ändern der Schleifenstruktur
    - Erste FOR-Schleife (Zeile3) geht über Anti-Diagonale und die innere Schleife geht über die Elemente der Anti-Diagonalen
      - Parallele Ausführung der inneren Schleife
      - Globale Synchronisation zwischen den Interationen der äußeren Schleife
    - Hoher Aufwand
      - Globale Synchronisation findet immer noch häufig statt
    - Lastungleichheit
      - wegen unterschiedlich vielen Elementen auf den Anti-Diagonalen

## Parallelisierungsprozess

- **Dekomposition am Beispiel: Ocean**
  - 3. Möglichkeit: Asynchrone Methode
    - Ignorieren der Abhängigkeiten zwischen den Gitterpunkten für einen Durchlauf
    - Globale Synchronisation zwischen den Iterationen, aber keine Änderung der Durchlaufordnung
    - Prozess aktualisiert alle Punkte, sequentielle Ordnung
    - Punkte können auf mehrere Prozesse aufgeteilt werden, dann ist die Ordnung nicht vorhersagbar, sondern hängt von der Zuteilung der Punkte zu Prozessen, der Anzahl der Prozesse und wie schnell die verschiedenen Prozesse relativ zueinander während der Laufzeit ausgeführt werden, ab
    - Ausführung ist nicht deterministisch!
    - Anzahl der Durchläufe bis zur Konvergenz kann von der Anzahl der Prozesse abhängen

## Parallelisierungsprozess

- **Dekomposition am Beispiel: Ocean**
  - 3. Möglichkeit: Asynchrone Methode
    - Dekomposition in individuelle innere Schleifeniterationen
    - for_all: weist darunter liegende HW/SW an, dass Schleifeiterationen parallel ausgeführt werden können

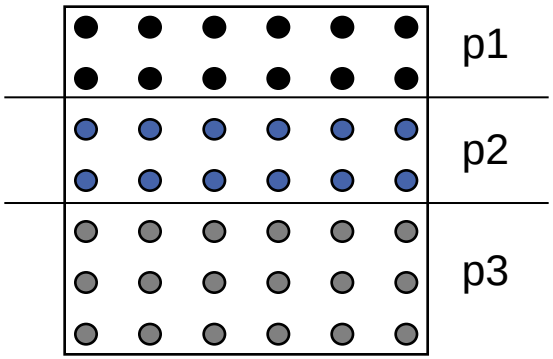
```
(1) while (!done) do
(2)     diff=0;
(3)     for_all i←1 to n do
(4)        for_all j←1 to n do
(5)           temp=A[i,j];
(6)           A[i,j]←0.2*(A[i,j]+A[i,j-1]+A[i-1,j]+A[i,j+1]+A[i+1,j]);
(7)           diff += abs(A[i,j]-temp);
(8)        end for_all
(9)      end for_all
(10)     if (diff/(n*n) < TOL) then done=1;
(11)   end while
```

## Parallelisierungsprozess

- **Zuweisung**
  - Spezifikation des Mechanismus, mit dessen Hilfe die Tasks auf Prozesse aufgeteilt werden
  - Ziel:
    - ausgewogene Lastverteilung: Lastbalanzierung
    - Reduzierung der Interprozess-Kommunikation
    - Reduzierung des Aufwands zur Laufzeit
    - Statische oder dynamische Zuweisung

## Parallelisierungsprozess

- **Zuweisung**
  - Beispiel:



# Parallele Programmierung

## Parallelisierungsprozess

- **Festlegung**
  - Architektur und Programmiermodell sowie die Programmiersprache spielen eine Rolle
    - Um die zugewiesenen Tasks ausführen zu können, benötigen die Prozesse Mechanismen
      - für den Zugriff auf die Daten,
      - für die Kommunikation (Austausch von Daten)
    - für die Synchronisation untereinander
  - Fragen
    - Organisation der Datenstrukturen
    - Ablauf der Tasks
    - Explizite oder implizite Kommunikation
  - Ziel:
    - Reduzierung des Kommunikations- und Synchronisationsaufwandes (aus der Sicht des Prozessors)
    - Erhalten der Lokalität der Datenzugriffe, soweit möglich
    - Reduzierung des Parallelisierungsaufwandes
  - Rechnerarchitekt: bereitstellen effizienter Mechanismen, die die Festlegung vereinfachen

# Parallele Programmierung

- **Parallelisierungsprozess**
- **Festlegung**
  - Programmiermodelle:
    - Shared-Memory-Programmiermodell
    - Nachrichten-orientiertes Programmiermodell
    - Datenparalleles Programmiermodell

## Parallelisierungsprozess

- **Festlegung**
  - **Shared-Memory-Programmiermodell: Primitive**

| Name | Syntax | Funktion |
|---|---|---|
| `CREATE` | `CREATE(p,proc,args)` | Generiere Prozess, der die Ausführung bei der Prozedur **proc** mit den Argumenten **args** startet |
| `G_MALLOC` | `G_MALLOC(size)` | Allokation eines gemeinsamen Datenbereichs der Größe **size** Bytes |
| `LOCK` | `LOCK(name)` | Fordere wechselseitigen exklusiven Zugriff an |
| `UNLOCK` | `UNLOCK(name)` | Freigeben des Locks |

## Parallelisierungsprozess

- **Festlegung**
  - **Shared-Memory-Programmiermodell: Primitive**

| Name | Syntax | Funktion |
|---|---|---|
| `BARRIER` | `BARRIER(name,number)` | Globale Synchronisation für `number` Prozesse |
| `WAIT_FOR_END` | `WAIT_FOR_END(number)` | Warten, bis `number` Prozesse terminieren |
| `WAIT_FOR_FLAG` | `while (!flag); or WAIT(flag)` | Warte auf gesetztes flag; entweder wiederholte Abfrage (spin) oder blockiere; |
| `SET FLAG` | `flag=1; or SIGNAL(flag)` | Setze flag; weckt Prozess auf, der `flag` wiederholt abfragt |

## Parallelisierungsprozess

- **Festlegung**
  - **Message Passing: Primitive**

| Name | Syntax | Funktion |
|---|---|---|
| `CREATE` | `CREATE(procedure)` | Erzeuge Prozess, der bei `procedure` startet |
| `SEND` | `SEND(src_addr,size,dest,tag)` | Sende `size` Bytes von Adresse `src_addr` an `dest` Prozess mit `tag` Identifier |
| `RECEIVE` | `RECEIVE(buffer_addr, size,src,tag)` | Empfange eine Nachricht mit der Kennung `tag` vom `src`-Prozess und lege `size` Bytes in Puffer bei `buffer_addr` ab |
| `BARRIER` | `BARRIER(name,number)` | Globale Synchronisation von `number` Prozessen |

# Vorlesung Rechnerstrukturen

## Kapitel 3: Multiprozessoren – Parallelismus auf Prozess-/ Blockebene

- 3.1 Motivation
- 3.2 Allgemeine Grundlagen
- 3.3 Parallele Programmierung
- 3.4 Quantitative Maßzahlen

# Quantitative Maßzahlen

## Parallelitätsprofil

- misst die entstehende Parallelität in einem parallelen Programm bzw. bei der Ausführung auf einem Parallelrechner.
- Gibt eine Vorstellung von der inhärenten Parallelität eines Algorithmus/Programms und deren Nutzung auf einem realen oder ideellen Parallelrechner

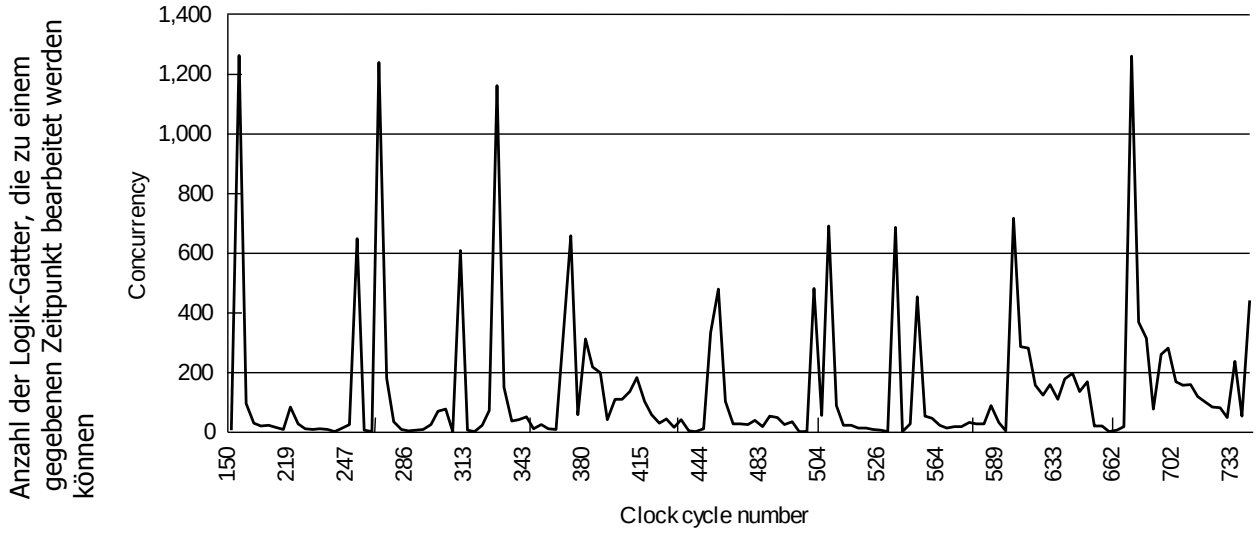
- Grafische Darstellung:
  - Auf der x-Achse wird die Zeit und auf der y-Achse die Anzahl paralleler Aktivitäten angetragen.
  - ➔ Perioden von Berechnungs- Kommunikations- und Untätigkeitszeiten sind erkennbar.

# Quantitative Maßzahlen

## Parallelitätsprofil

- Zeigt an, wie viele Tasks einer Anwendung zu einem Zeitpunkt parallel ausgeführt werden können
- Parallelitätsgrad PG(t):
  - Anzahl der Tasks, die zu einem Zeitpunkt parallel bearbeitet werden können

Beispiel: Parallele ereignisgesteuerte Simulation der Logik-Synthese




Quelle: D. Culler: Parallel Computer Architecture. Morgan Kaufmann Publishers, 1999, p.87

# Quantitative Maßzahlen

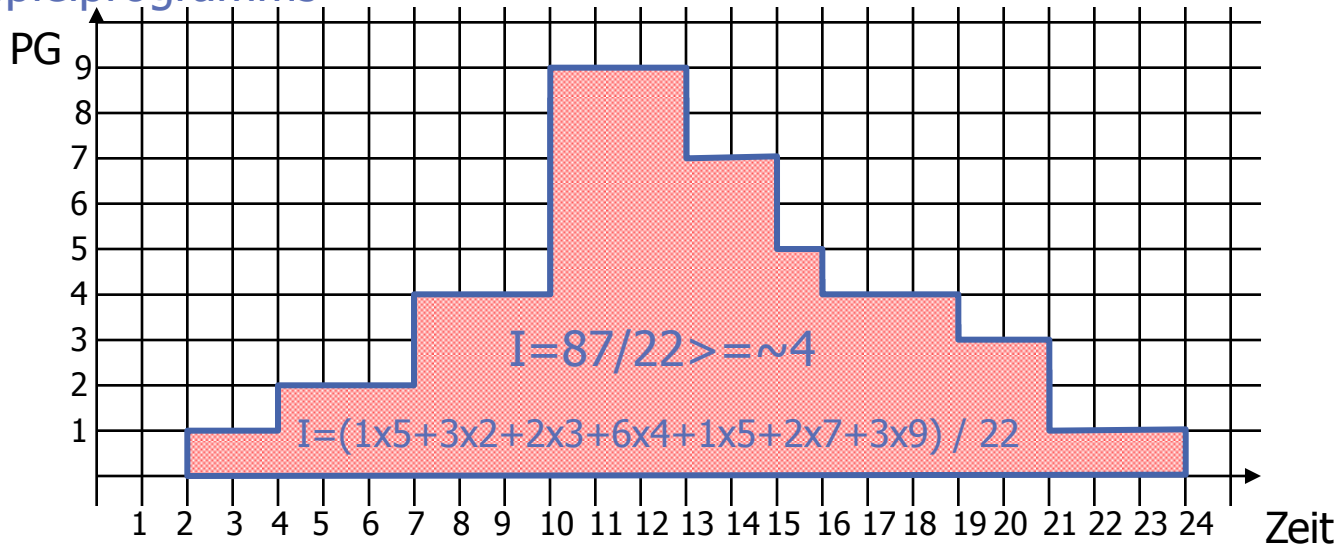
## Parallelitätsprofil

- **Parallelindex I** (Mittlerer Grad des Parallelismus):

$$I = \frac{1}{t_2 - t_1} \int_{t1}^{t2} PG(t)dt \qquad I = \left( \sum_{i=1}^{m} i * t_i \right) \Big/ \left( \sum_{i=1}^{m} t_i \right)$$

PG Bereich  Ausführungszeit

Parallelitätsprofil eines Beispielprogramms



# Quantitative Maßzahlen

## Vergleich von Multiprozessorsystemen zu Einprozessorsystemen

- Leistungsangaben zu Multiprozessorsystemen werden mit Leistungsangaben zu Einprozessorsystemen in Beziehung gesetzt
- Notwendig:
  - Programm das auf beiden zu vergleichenden Systemen ablaufen kann

# Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Definitionen:**
  - P(1): Anzahl der auszuführenden (Einheits-) Operationen (Tasks) des Programms auf einem Einprozessorsystem.
  - P(n): Anzahl der auszuführenden (Einheits-) Operationen (Tasks) des Programms auf einem Multiprozessorsystem mit n Prozessoren.
  - T(1): Ausführungszeit auf einem Einprozessorsystem in Schritten (oder Takten).
  - T(n): Ausführungszeit auf einem Multiprozessorsystem mit n Prozessoren in Schritten (oder Takten).
- **Vereinfachende Voraussetzungen:**
  - T(1) = P(1),
    - da in einem Einprozessorsystem (Annahme: einfacher Prozessor) jede (Einheits-) Operation in genau einem Schritt ausgeführt werden kann.
  - $T(n) \leq P(n)$,
    - da in einem Multiprozessorsystem mit n Prozessoren ($n \geq 2$) in einem Schritt mehr als eine (Einheits-)Operation ausgeführt werden kann.

# Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Beschleunigung S(n) (Speedup):**

$$S(n) = \frac{T(1)}{T(n)}$$

  - Gibt die Verbesserung in der Verarbeitungsgeschwindigkeit an
  - Wert bezieht sich auf das jeweils bearbeitete Programm oder kann als Mittelwert eine Menge von Programmen angesehen werden
  - Üblicherweise gilt:

# Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Effizienz E(n)**

$$E(n) = \frac{S(n)}{n}$$

  - Gibt die relative Verbesserung in der Verarbeitungsgeschwindigkeit an
  - Leistungssteigerung wird mit der Anzahl der Prozessoren n normiert

  - Üblicherweise gilt:

# Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Beschleunigung (Speed-Up), Effizienz:**
  - Algorithmenunabhängige Definition
  - Man setzt den besten bekannten sequentiellen Algorithmus für das Einprozessorsystem in Beziehung zum vergleichbaren parallelen Algorithmus für das Multiprozessorsystem.
    - Absolute Beschleunigung
    - Absolute Effizienz

# Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Beschleunigung (Speed-Up), Effizienz:**
  - Algorithmenabhängige Definition
  - Man benutzt den parallelen Algorithmus so, als sei er sequentiell, und misst dessen Laufzeit auf einem Einprozessorsystem.
  - Der für die Parallelisierung erforderliche Zusatzaufwand an Kommunikation und Synchronisation kommt „ungerechterweise“ auch für den sequentiellen Algorithmus zum Tragen.
    - Relative Beschleunigung
    - Relative Effizienz

# Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Mehraufwand R(n) für die Parallelisierung:**

$$R(n) = \frac{P(n)}{P(1)}$$

  - Beschreibt den bei einem Multiprozessorsystem erforderlichen Mehraufwand für die Organisation, Synchronisation und Kommunikation der Prozessoren
  - Es gilt:

$$1 \leq R(n)$$

    - Anzahl der auszuführenden Operationen eines parallelen Programms größer ist als diejenige des vergleichbaren sequentiellen Programms

# Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- **Auslastung U(n):**

$$U(n) = \frac{I(n)}{n} = R(n) \cdot E(n) = \frac{P(n)}{n \cdot T(n)}$$

  - Entspricht dem normierten Parallelindex
  - Gibt an, wie viele Operationen (Tasks) jeder Prozessor im Durchschnitt pro Zeiteinheit ausgeführt hat

# Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- Folgerungen
  - Alle definierten Ausdrücke haben für n = 1 den Wert 1.
  - Der Parallelindex gibt eine obere Schranke für die Leistungssteigerung:

$$1 \leq S(n) \leq I(n) \leq n$$

  - Die Auslastung ist eine obere Schranke für die Effizienz:

$$\frac{1}{n} \leq E(n) \leq U(n) \leq 1$$

# Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- Zahlenbeispiel:
  - Ein Einprozessorsystem benötige für die Ausführung von 1000 Operationen 1000 Schritte.
  - Ein Multiprozessorsystem mit 4 Prozessoren benötige dafür 1200 Operationen, die aber in 400 Schritten ausgeführt werden können.
  - Damit gilt:

$$P(1) = T(1) = 1000, P(4) = 1200, T(4) = 400$$

  - Daraus ergibt sich:

$$S(4) = 2,5 \text{ und } E(4)=0,625$$

  - Die Leistungssteigerung verteilt sich als im Mittel zu 62,5% auf alle Prozessoren

# Quantitative Maßzahlen

- **Vergleich von Multiprozessorsystemen zu Einprozessorsystemen**
- Zahlenbeispiel:
  - Parallelindex und Auslastung:

$$I(4) = 3 \text{ und } U(4) = 0,75$$

  - Es sind im Mittel drei Prozessoren gleichzeitig tätig, d.h., jeder Prozessor ist nur zu 75% der Zeit aktiv.
  - Mehraufwand:

$$R(4) = 1,2$$

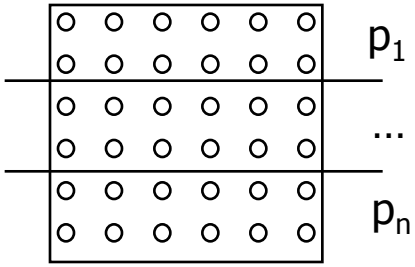
  - Bei Ausführung auf dem Multiprozessorsystem sind 20% mehr Operationen als bei Ausführung auf einem Einprozessorsystem notwendig.

# Quantitative Maßzahlen

- **Skalierbarkeit eines Parallelrechners**
  - das Hinzufügen von weiteren Verarbeitungselementen führt zu einer kürzeren Gesamtausführungszeit, ohne dass das Programm geändert werden muss.
  - Insbesondere meint man damit eine lineare Steigerung der Beschleunigung mit einer Effizienz nahe bei Eins.
  - Wichtig für die Skalierbarkeit ist eine angemessene Problemgröße.
  - Bei fester Problemgröße und steigender Prozessorzahl wird ab einer bestimmten Prozessorzahl eine Sättigung eintreten. Die Skalierbarkeit ist in jedem Fall beschränkt.
  - Skaliert man mit der Anzahl der Prozessoren auch die Problemgröße (scaled problem analysis), so tritt dieser Effekt bei gut skalierenden Hardware- oder Software-Systemen nicht auf.

# Quantitative Maßzahlen

- **Gesetz von Amdahl**
- Beispiel:
  - Gegeben: ein zweidimensionales *k* x *k*-Gitter
  - 1. Berechnungsphase: Ausführung einer Operation auf allen Gitterpunkten
    - Annahme: keine Abhängigkeiten zwischen den Gitterpunkten
    - Parallele Berechnung auf n Prozessoren




  - 2. Berechnungsphase: Berechnung der Summe der k2 berechneten Werte der Gittersumme
    - Jeder Prozessor addiert seine $k^2/n$ berechneten Werte zur globalen Summe

# Quantitative Maßzahlen

- **Gesetz von Amdahl**
- Beispiel:
  - Problem:
    - Akkumulation der globalen Summe muss serialisiert werden!
    - 2. Phase benötigt $k^2$ Zeiteinheiten unabhängig von *n*
    - Ausführungszeit des parallelen Programms: $k^2/n + k^2$
    - Ausführungszeit des sequentiellen Programms: $2k^2$
    - Möglicher Speedup S:

$$\frac{2k^2}{\frac{k^2}{n} + k^2} = \frac{2n}{n+1}$$

    - Selbst bei einer hohen Anzahl Prozessoren nicht mehr als 2.

# Quantitative Maßzahlen

- **Gesetz von Amdahl**
  - Gesamtausführungszeit T(n)

$$T(n) = \underbrace{T(1) \cdot \frac{1-a}{n}}_{\text{Ausführungszeit des parallel ausführbaren Programmteils 1-a}} + \underbrace{T(1) \cdot a}_{\text{Ausführungszeit des sequentiell ausführbaren Programmteils a}}$$

a: Anteil des Programmteils, der nur sequentiell ausgeführt werden kann

  - Beschleunigung

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

  - Für $n \to \infty$ : $S(n) = \frac{1}{a}$

# Quantitative Maßzahlen

- **Gesetz von Amdahl**
- Beispiel:
  - Erhöhung der Parallelität
    - Aufteilung der 2. Berechnungsphase in zwei weiteren Teilphasen:
    - 1. Teilphase: Jeder Prozessor berechnet die Summe seiner berechneten Werte
      - Kann vollständig parallel abgearbeitet werden
    - 2. Teilphase: Akkumulation der Teilsummen
      - Weiterhin seriell!
    - Ausführungszeit $T(n)=k^2/n+k^2/n+n$
    - Beschleunigung $S(n)=n*2k^2/(2k^2+n^2)$
      - Wenn n groß genug, dann nahezu linear!

# Quantitative Maßzahlen

- **Gesetz von Amdahl**
- Diskussion
  - Amdahls Gesetz zufolge kann eine kleine Anzahl von sequentiellen Operationen die mit einem Parallelrechner erreichbare Beschleunigung signifikant begrenzen.
  - Beispiel: $a$ = 1/10 des parallelen Programms kann nur sequenziell ausgeführt werden,
    ➔ das gesamte Programm kann maximal zehnmal schneller als ein vergleichbares, rein sequenzielles Programm sein.
  - Jedoch: viele parallele Programme haben einen sehr geringen sequenziellen Anteil (a«1)

# Quantitative Maßzahlen

- **Synergetischer Effekt und superlinearer Speedup**
- Theorie : einen „**superlinearen Speedup**“ kann es nicht geben:
  - Jeder parallele Algorithmus lässt sich auf einem Einprozessorsystem simulieren, indem in einer Schleife jeweils der nächste Schritt jedes Prozessors der parallelen Maschine emuliert wird.

- Ein „**superlinearer Speed-up**“ kann real beobachtet werden bei
  - parallelem Backtracking (depth-first search)
  - Beim Programmlauf auf einem Rechner passen die Daten nicht in den Hauptspeicher des Rechners (häufiger Seitenwechsel), aber:
    bei Verteilung auf die Knoten des Multiprozessors können die parallelen Programme vollständig in den Cache- und Hauptspeichern der einzelnen Knoten ablaufen.

# Quantitative Maßzahlen

- **Weitere grundsätzliche Probleme bei Multiprozessoren**
  - Verwaltungsaufwand (Overhead)
    - Steigt mit der Zahl der zu verwaltenden Prozessoren
  - Möglichkeit von Systemverklemmungen (deadlocks)
  - Möglichkeit von Sättigungserscheinungen
    - können durch Systemengpässe (bottlenecks) verursacht werden.

# Verbindungsstrukturen

## Verbindungsnetze in Multiprozessoren

- Ermöglichen die Kommunikation und Kooperation zwischen den Verarbeitungselementen (Knoten)
  - Zuverlässiger Austausch von Informationen
- Einsatz eines Verbindungsnetzwerks
  - Multiprozessor mit verteiltem Speicher (nachrichtenorientierter Multiprozessor)
    - Verbinden physikalisch jeden Knoten für das Versenden von Nachrichten
    - Direkte Send/Receive-Kommunikation zwischen den Knoten
  - Multiprozessor mit gemeinsamem Speicher
    - Ermöglicht den Zugriff aller Knoten auf den gemeinsamen Speicher
    - Kommunikation durch Lesen und Schreiben auf gemeinsame Daten

# Verbindungsstrukturen

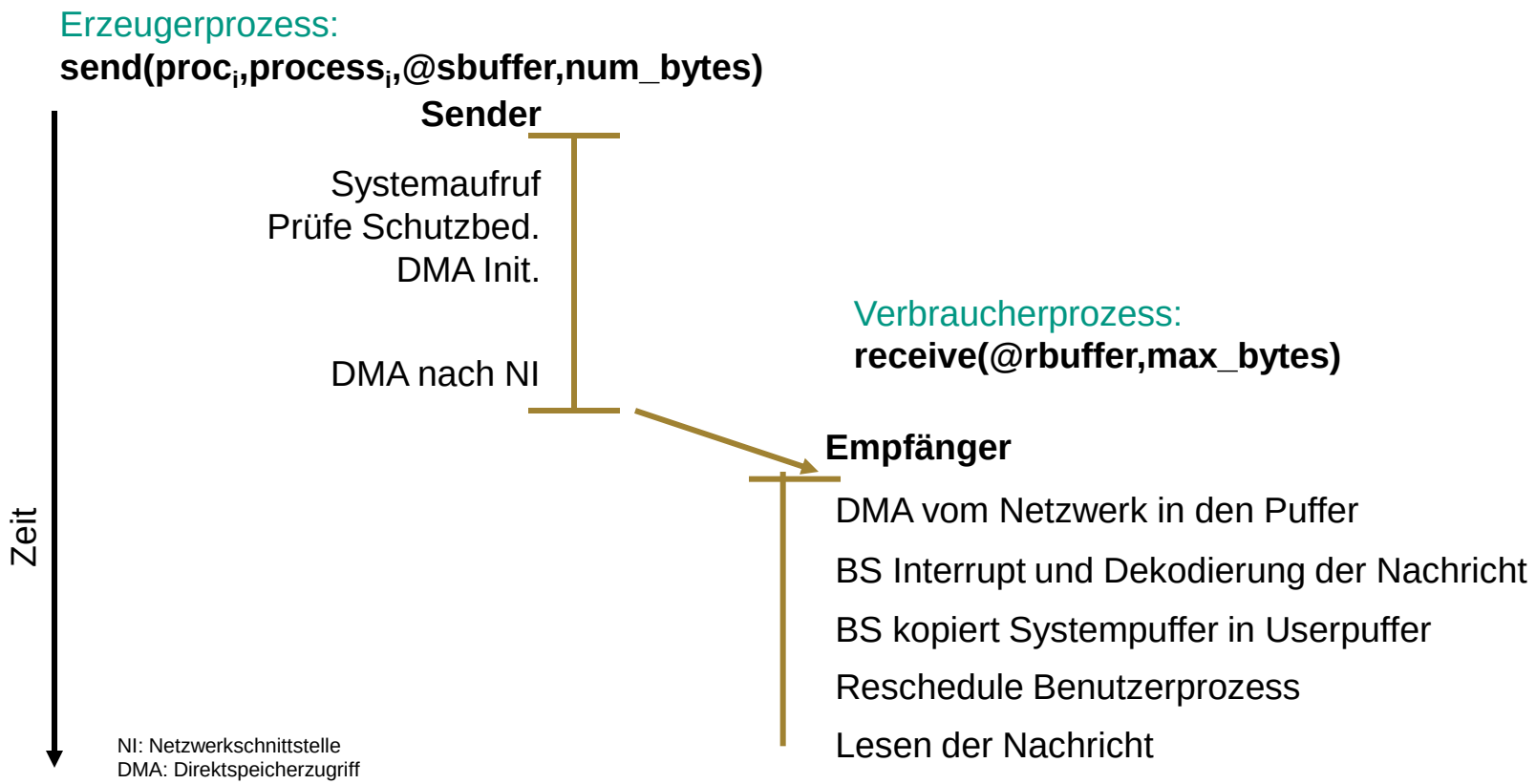
## Charakterisierung von Verbindungsnetzwerken

- **Latenz**
  - **Übertragungszeit einer Nachricht $T_{msg}$**
    - die Zeit, die für das Verschicken einer Nachricht von einer bestimmten Länge zwischen zwei Prozessoren benötigt wird
  - Die Übertragungszeit setzt sich zusammen aus:
    - der **Startzeit $t_s$** (Message Startup Time):
      - Die Zeit, die benötigt wird, um die Kommunikation zu initiieren
    - **Transferzeit $t_w$** pro übertragenem Datenwort:
      - hängt von der physikalischen Bandbreite des Kommunikationsmediums ab.
    - Voraussetzung:
      - Verbindungsnetz ist konfliktfrei, da sonst die Übertragungszeit nicht fest berechnet werden kann

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Latenz** (Übertragungszeit einer Nachricht, latency)
  - **Software-Overhead**



# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Latenz** (Übertragungszeit einer Nachricht, latency)
  - **Kanalverzögerung** (channel delay)
    - Dauer für die Belegung eines Kommunikationskanals durch eine Nachricht
    - Kanal:
      - Physikalische Verbindung zwischen Schalterelementen oder Knoten mit einem Puffer zum Halten der Daten während ihrer Übertragung
    - Verbindung (link)
      - Menge von Leitungen

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Latenz** (Übertragungszeit einer Nachricht, latency)
  - **Schaltverzögerung, Routing-Verzögerung** (switching delay, routing delay)
    - Zeit, einen Weg zwischen zwei Knoten aufzubauen
    - Pfadberechnung oder Wegefindung (Routing)
      - die Art, wie der Weg einer Nachricht vom Sende- zum Zielknoten berechnet wird
      - Zu einer Verbindungsstruktur kann es mehrere Wegefindungsalgorithmen geben
      - einfache Implementierung in Verbindungselementen mit Hilfe eines schnellen Hardware-Algorithmus

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Latenz** (Übertragungszeit einer Nachricht, latency)
  - **Blockierungszeit** (contention time)
    - Wird verursacht, wenn zu einem Zeitpunkt mehr als eine Nachricht auf eine Netzwerkressource zugreifen
  - **Blockierung** (contention)
    - Ein Verbindungsnetzwerk heißt blockierungsfrei, falls jede gewünschte Verbindung zwischen Prozessoren oder zwischen Prozessoren und Speichern unabhängig von schon bestehenden Verbindungen hergestellt werden kann

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Durchsatz oder Übertragungsbandbreite** (bandwidth)
  - Maximale Übertragungsleistung des Verbindungsnetzwerkes oder einzelner Verbindungen, meist in Megabits pro Sekunde (MBit/s) oder Megabytes pro Sekunde (MB/s)

- **Bisektionsbandbreite** (bisection bandwidth)
  - Maximale Anzahl von Megabytes pro Sekunde, die das Netzwerk über die Bisektionslinie, die das Netzwerk in zwei gleiche Hälften teilt, transportieren kann

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Diameter oder Durchmesser r** (diameter):
  - maximale Distanz für die Kommunikation zweier Prozessoren, also die Anzahl der Verbindungen, die durchlaufen werden müssen. Man spricht auch von der maximalen Pfadlänge zwischen zwei Knoten.

- **Verbindungsgrad eines Knotens *P*** (node degree, connectivity)
  - ist definiert als die Anzahl der direkten Verbindungen, die von einem Knoten zu anderen Knoten bestehen.

- **Mittlere Distanz $d_a$** (average distance) zwischen zwei Knoten
  - Anzahl der Links auf dem kürzesten Pfad zwischen zwei Knoten
  - $P/d_a$ ist die maximale Anzahl neuer Nachrichten, die von jedem Knoten in einem Zyklus in das Netzwerk eingebracht werden können

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Komplexität oder Kosten:**
  - Kosten für die Implementierung einer Hardware
  - Aufwand für das Verbindungsnetz gemessen in der Anzahl und der Art der Schaltelemente und Verbindungsleitungen.

- **Erweiterbarkeit:**
  - Multiprozessoren können begrenzt, stufenlos oder nur durch Verdopplung der Anzahl der Prozessoren erweiterbar sein.

- **Skalierbarkeit:**
  - Fähigkeit, die wesentlichen Eigenschaften des Verbindungsnetzes auch bei beliebiger Erhöhung der Knotenzahl beizubehalten.

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Ausfallstoleranz oder Redundanz :**
  - Verbindungen zwischen Knoten sind selbst dann noch zu schalten, wenn einzelne Elemente des Netzes (Schaltelemente, Leitungen) ausfallen.
  - Ein fehlertolerantes Netz muss also zwischen jedem Paar von Knoten mindestens einen zweiten, redundanten Weg bereitstellen.
    Die Eigenschaft eines Systems, bei Ausfall einzelner Komponenten unter deren Umgehung funktionstüchtig zu bleiben, wenn auch mit verminderter Leistung, wird als Graceful degradation bezeichnet.

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Art des Datentransfers:**
  - **Durchschalte- oder Leitungsvermittlung** (circuit switching):
    - Eigenschaft eines Netzes eine direkte Verbindung zwischen zwei oder mehreren Knoten eines Netzes zu schalten.
      Die physikalische Verbindung bleibt für die gesamte Dauer der Informationsübertragung bestehen.
    - Blockierungsfreie Kommunikation
    - Kurze Latenz
    - Gut geeignet für lange Nachrichten, da die Zeit zum Aufsetzen einer Nachricht im Verhältnis zur Übertragungszeit kurz ist
    - Übertragungszeit einer Nachricht der Länge L über eine Distanz d beträgt: $l/b+d\delta$, mit individueller Schaltverzögerung $\delta$ und der Bandbreite b eines Kanals

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Art des Datentransfers:**
  - **Paketvermittlung (packet switching):**
    - Datenpakete fester Länge oder Nachrichten variabler Länge werden entsprechend einem Wegefindungsalgorithmus (routing) vom Absender zum Empfänger geschickt
    - Nachrichten mit Adresse und Daten werden durch das Netzwerk verschickt
    - Adresse wird in jedem Knoten gelesen und die Nachricht wird zum nächsten Knoten weitergeleitet, bis die Nachricht das Ziel erreicht
    - Günstig für kurze Nachrichten

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Art des Datentransfers:**
  - **Paketvermittlung (packet switching):**
  - Übertragungsmodi: **Store-and-forward-Modus**
    - Jeder Knoten enthält einen Puffer zum Aufnehmen der vollständigen Nachricht
    - Nachricht wird von jedem Zwischenknoten in Empfang genommen, vollständig zwischengespeichert und dann weiter übertragen
    - Nachfolgende Pakete werden nacheinander verschickt
    - Gegenüber Circuit Switching: höhere Bandbreite, aber auch höhere Latenz
    - Übertragungszeit einer Nachricht der Länge L über eine Distanz d von einer Quelle zum Ziel beträgt $d(L/b+\delta)$

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Art des Datentransfers:**
  - **Paketvermittlung (packet switching):**
  - Übertragungsmodi: **Cut-through** oder **wormhole**
    - Phit und Flusskontrolle
    - Eine Nachricht selbst wird in eine Anzahl von Übertragungseinheiten (phits – physical transfer units – oder auch flits – flow control digits – genannt) zerlegt.
    - Ein Phit ist dabei die Datenportion, die zu einem Zeitpunkt zwischen zwei Knoten übertragen werden kann.
    - Bei der Nachrichtenübertragung zwischen nicht benachbarten Sender- und Empfängerknoten sind Puffer nötig.

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Art des Datentransfers:**
  - **Paketvermittlung (packet switching):**
  - Übertragungsmodi: **Virtual-cut-through-Modus:**
    - Nachricht wird aufgeteilt in Zellen (Flits) fester Größe
    - Der Kopfteil der Nachricht enthält die Empfängeradresse und bestimmt den einzuschlagenden Weg. Flits mit Daten folgen dem Kopf auf dem Pfad von der Quelle zum Ziel
    - Bei Ankunft des Kopfs einer Nachricht wird dieser dekodiert . Nachfolgende Flits werden automatisch an den nächsten Knoten auf dem ausgewählten Pfad weitergeleitet , ein Flit pro Zeiteinheit gemäß einer Pipeline-Verarbeitung
    - Kopf-Information wird festgehalten bis letztes Flit angekommen ist.
    - ankommende Daten werden nur im Konfliktfall im Knoten vollständig zwischengespeichert.
    - In jedem Knoten werden Puffer bereit gehalten, die auch ein maximal großes Nachrichtenpaket zwischenspeichern können

# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Art des Datentransfers:**
  - **Paketvermittlung (packet switching):**
  - Übertragungsmodi: **Wormhole-routing-Modus:**
    - solange keine Übertragungskanäle blockiert sind, mit den Virtual-cut-through-Modus identisch.
    - Falls der Kopfteil der Nachricht auf einen Kanal trifft, der gerade belegt ist, wird er abgeblockt. Alle nachfolgenden Übertragungseinheiten der Nachricht verharren dann ebenfalls an ihrer augenblicklichen Position, bis die Blockierung aufgehoben ist. Durch das Verharren werden die Puffer nachfolgender Kanäle auch für weitere Nachrichten blockiert.

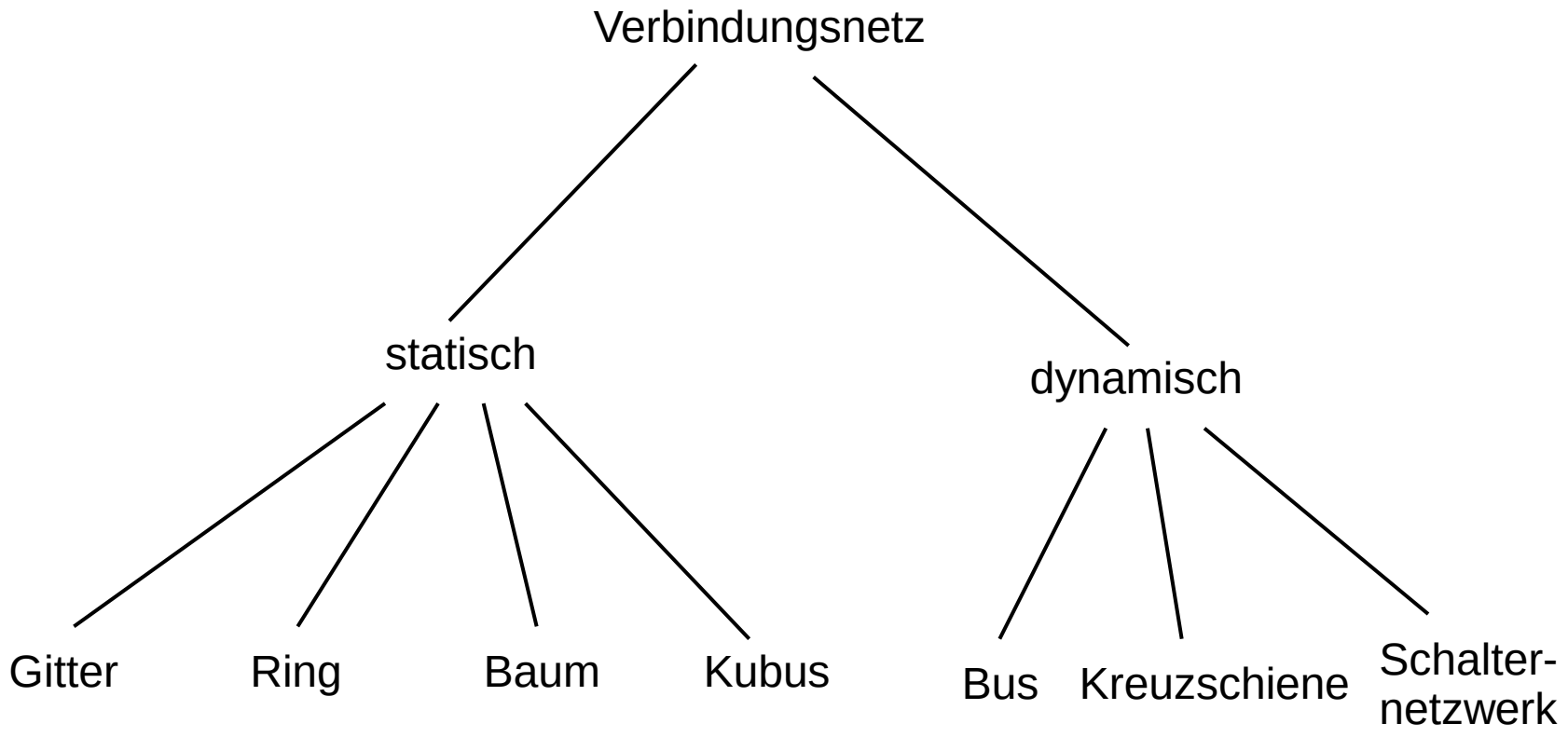
# Verbindungsstrukturen

## Charakterisierung von Verbindungsnetzwerken

- **Art des Datentransfers:**
  - **Paketvermittlung (packet switching):**
  - Übertragungsmodi: **Buffered wormhole routing:**
    - Kompromisslösung zwischen Virtual-cut-through- und Wormhole-routing-Modus eingesetzt,
    - begrenzter Puffer zur Aufnahme kleinerer Pakete vorhanden
    - größere Pakete werden im Blockierungsfall – ähnlich dem Wormhole-routing-Modus – in den Puffern mehrerer Knoten zwischengespeichert.

# Verbindungsstrukturen

## Topologie: Klassifizierung



# Verbindungsstrukturen

## Topologie:

- **Statische Verbindungsnetze**
  - Nach Aufbau des Verbindungsnetzes bleiben die Verbindungen fest
  - Gute Leistung für Probleme mit vorhersagbaren Kommunikationsmustern zwischen benachbarten Knoten

# Verbindungsstrukturen

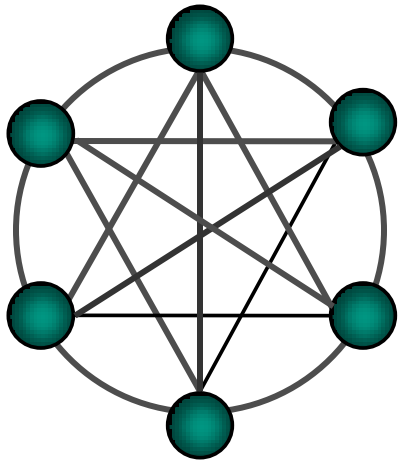
**Topologie:**

- **Statische Verbindungsnetze**
- Vollständige Verbindung
  - Jeder Knoten ist mit jedem anderen Knoten verbunden
  - Höchste Leistungsfähigkeit
    - Arbeitet für alle Anwendungen mit allen Arten von Kommunikationsmustern effizient
  - Aber: nicht praktikabel in Parallelrechnern
    - Netzwerkkosten steigen quadratisch mit der Anzahl der Prozessoren

# Verbindungsstrukturen

**Topologie:**

- **Statische Verbindungsnetze**
- Vollständige Verbindung
  - Jeder Knoten ist mit jedem anderen Knoten verbunden
  - Höchste Leistungsfähigkeit
    - Arbeitet für alle Anwendungen mit allen Arten von Kommunikationsmustern effizient
  - Aber: nicht praktikabel in Parallelrechnern
    - Netzwerkkosten steigen quadratisch mit der Anzahl der Prozessoren



# Verbindungsstrukturen

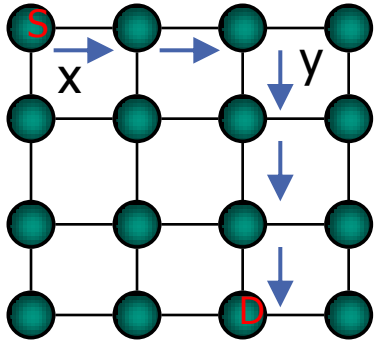
## Topologie:

- **Statische Verbindungsnetze**
- **Gitterstrukturen**
  - 1-dimensionales Gitter (**lineares Feld, Kette**)
    - Verbindet N Knoten mit (N-1) Verbindungen
    - Endknoten haben den Grad 1, Zwischenknoten den Grad 2 und sind mit benachbarten Knoten verbunden
    - Diameter r ist N-1
    - Disjunkte Bereiche des linearen Netzwerkes können gleichzeitig genutzt werden, aber es sind mehrere Schritte notwendig, um eine Nachricht zwischen zwei nicht benachbarte Knoten zu verschicken

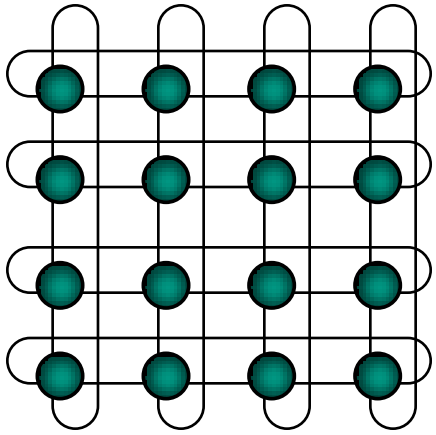
# Verbindungsstrukturen

- **Topologie:**
- **Statische Verbindungsnetze**
- **Gitterstrukturen**
  - **k-dimensionales Gitter mit N Knoten**
    - Innere Knoten haben den Grad 2k, wobei die 2k benachbarten Knoten miteinander verbunden sind
    - In einem k-dimensionalen Netzwerk mit Knoten in jeder Dimension beträgt der Diameter

2-dim. Gitter



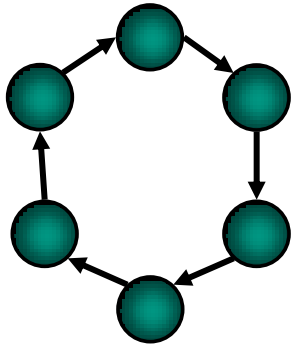

2-dim. Torus



# Verbindungsstrukturen

## Topologie:

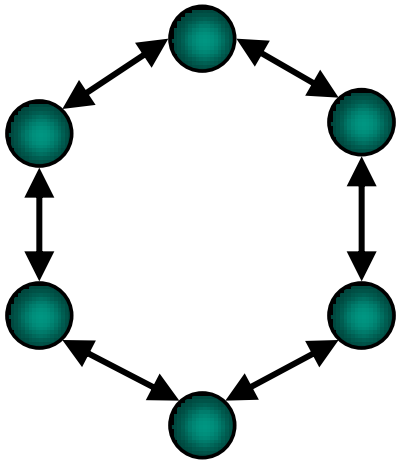
- **Statische Verbindungsnetze**
- **Ring**
  - Erhält man, wenn man die Endknoten eines linearen Feldes miteinander verbindet
  - Unidirektionaler Ring mit N Knoten
    - Nachrichten werden in einer Richtung vom Quellknoten zum Zielknoten verschickt
    - Diameter r ist N-1
    - Bei Ausfall einer Verbindung bricht die Kommunikation zusammen



# Verbindungsstrukturen

## Topologie:

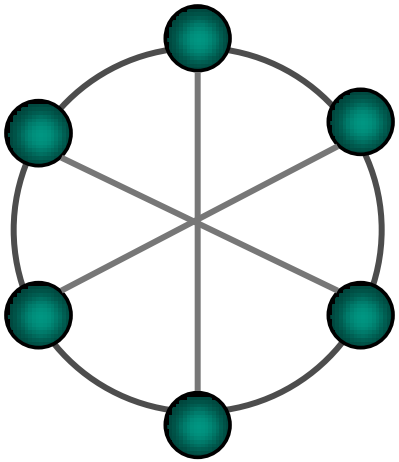
- **Statische Verbindungsnetze**
- **Bidirektionaler Ring mit N Knoten**
  - symmetrisches Netzwerk
  - Der längste Pfad, den eine Nachricht nehmen muss, ist nicht länger als N/2
  - Bei Ausfall einer Verbindung bricht die Kommunikation noch nicht zusammen, während zwei Ausfälle von Verbindungen das Netzwerk in zwei disjunkte Teilnetzwerke aufteilen



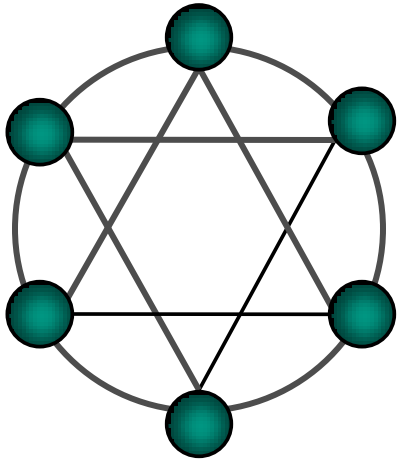
# Verbindungsstrukturen

## Topologie:

- **Statische Verbindungsnetze**
- **Chordaler Ring**
  - Hinzufügen redundanter Verbindungen
    - erhöht Fehlertoleranzeigenschaft des Verbindungsnetzwerkes
    - Höherer Knotengrad und kleinerer Diameter gegenüber Ring



Chordaler Ring mit Knotengrad 3



Chordaler Ring mit Knotengrad 4

# Verbindungsstrukturen

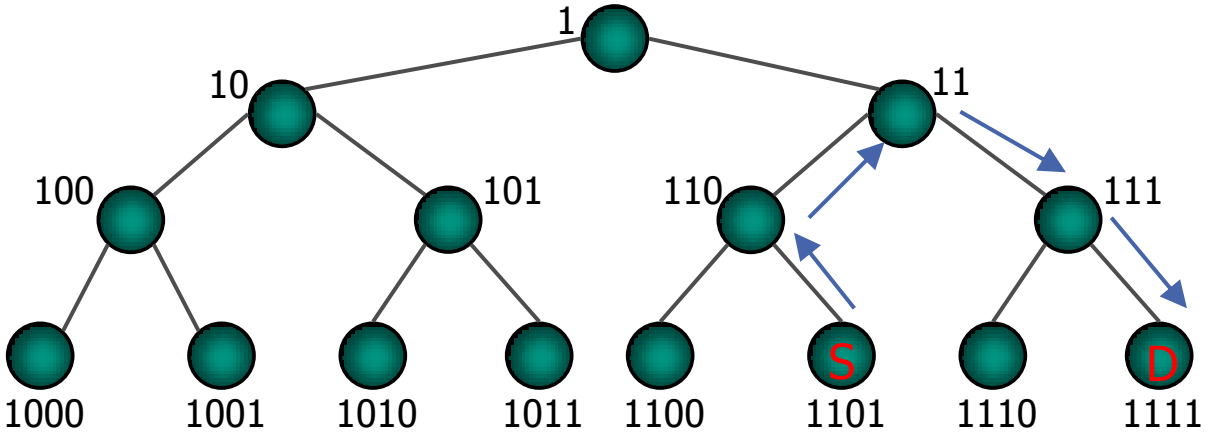
**Topologie:**

- **Statische Verbindungsnetze**
- **Baumstrukturen**
  - Binärer Baum mit m-Ebenen:
  - Auf Ebene m: $N=2^m-1$ Knoten
  - Diameter: 2(m-1)
  - Adressierung der Knoten:
    - Die Knotennummer auf Ebene m besteht aus m Bits
    - Der Wurzelknoten hat die Nummer 1
    - Die Nummer des linken Kindknotens erhält man durch Hinzufügen einer 0 an die niederwertige Stelle der Adresse des Elternknoten
    - Die Nummer des rechten Kindknotens erhält man durch Hinzufügen einer 1 an die niederwertige Stelle der Adresse des Elternknoten

# Verbindungsstrukturen

## Topologie:

- **Statische Verbindungsnetze**
- **Baumstrukturen: Routing**
  - Finde gemeinsamen Elternknoten P von S und D
  - Gehe von S nach P und von P nach D



# Verbindungsstrukturen

## Topologie:

- Statische Verbindungsnetze
- Baumstrukturen: Routing

  - Die Binärdarstellung der Adresse eines Quellknotens S auf Ebene i sei $S_i S_{i-1} \ldots S_1$ und die der Adresse des Zielknotens D auf Ebene j sei $D_j D_{j-1} \ldots D_1$
  - Finde die gemeinsamen höchstwertigen Bits von S und D, so dass die Adresse des Elternknotens P gleich $D_j D_{j-1} \ldots D_x = S_i S_{i-1} \ldots S_{(i-j+x)}$ ist

  - Steige von S (i-j+x) Ebenen auf nach P
  - for k=x-1 step 1 until 0

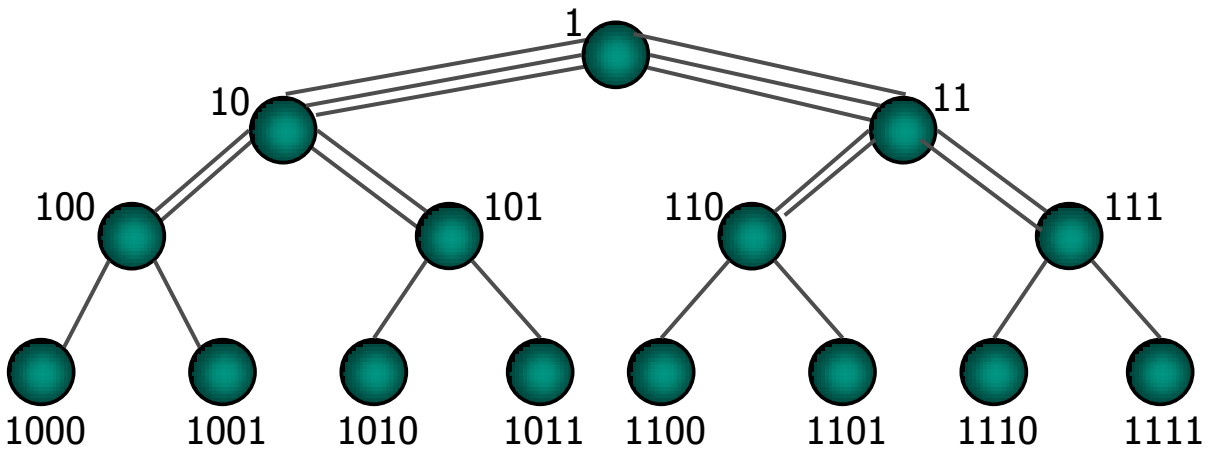
    {steige nach links ab, falls $D_x$=0

    steige nach rechts ab, falls $D_x$=1}

# Verbindungsstrukturen

**Topologie:**

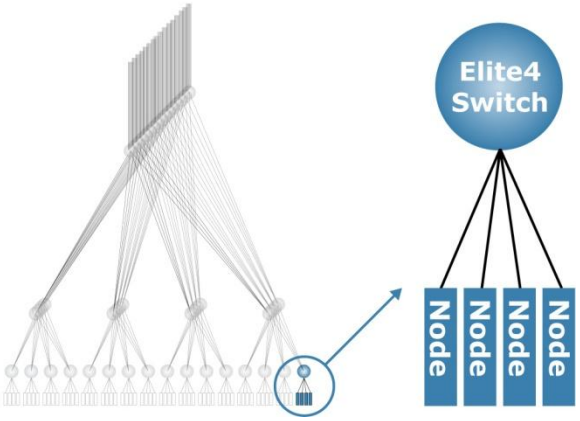
- **Statische Verbindungsnetze**
- **Baumstrukturen**
- **Fat Tree:**
  - Lösung des Blockierungsproblems in Richtung Wurzel
  - Kommunikationskanäle werden größer, je näher man sich der Wurzel nähert

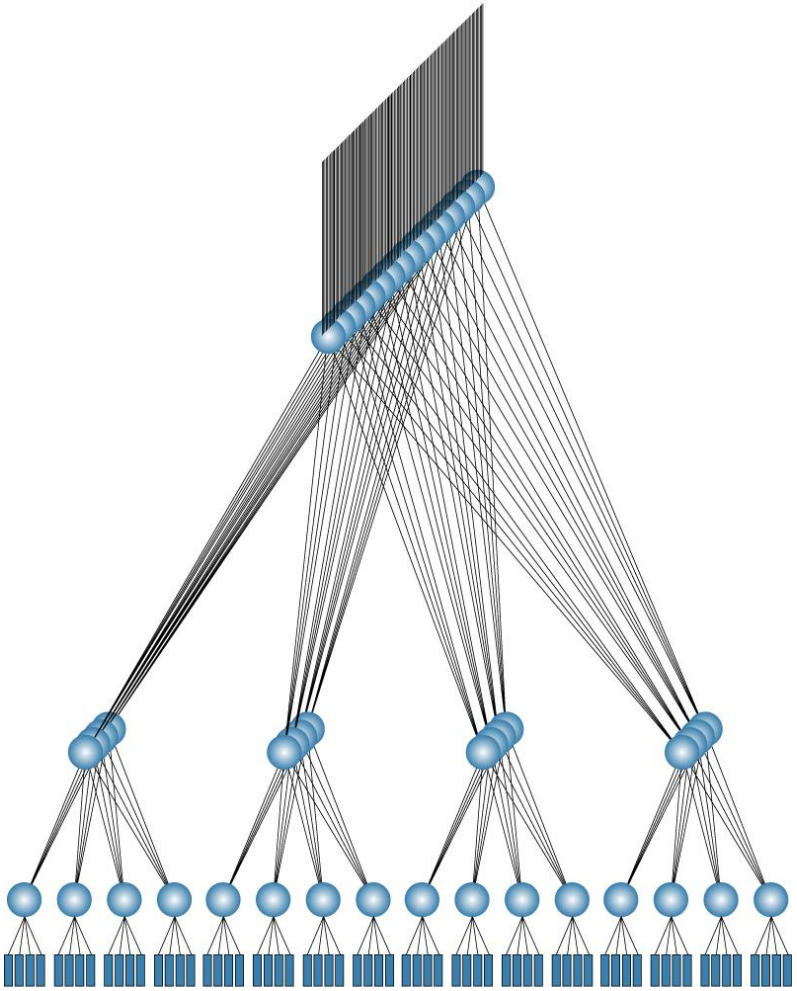


# Verbindungsstrukturen

## Topologie:

- **Statische Verbindungsnetze**
- Beispiel: Quadrics QSnet



Quelle: D. Addison, J. Beecroft, D. Hewson, M. McLaren (Quadrics Ltd.), Fabrizio Petrini (LANL): A network for Supercomputing Applications. Hot Chips, August 2003, http://www.c3.lanl.gov/~fabrizio/quadrics.html

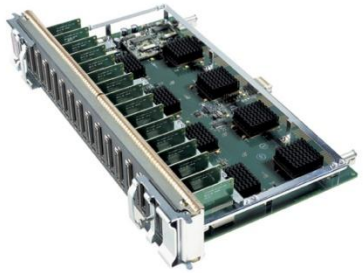
# Verbindungsstrukturen

## Topologie:

- **Statische Verbindungsnetze**
- Beispiel: Quadrics QSnet

Elan 4 network interface card:

Elite 4 Switch Component:



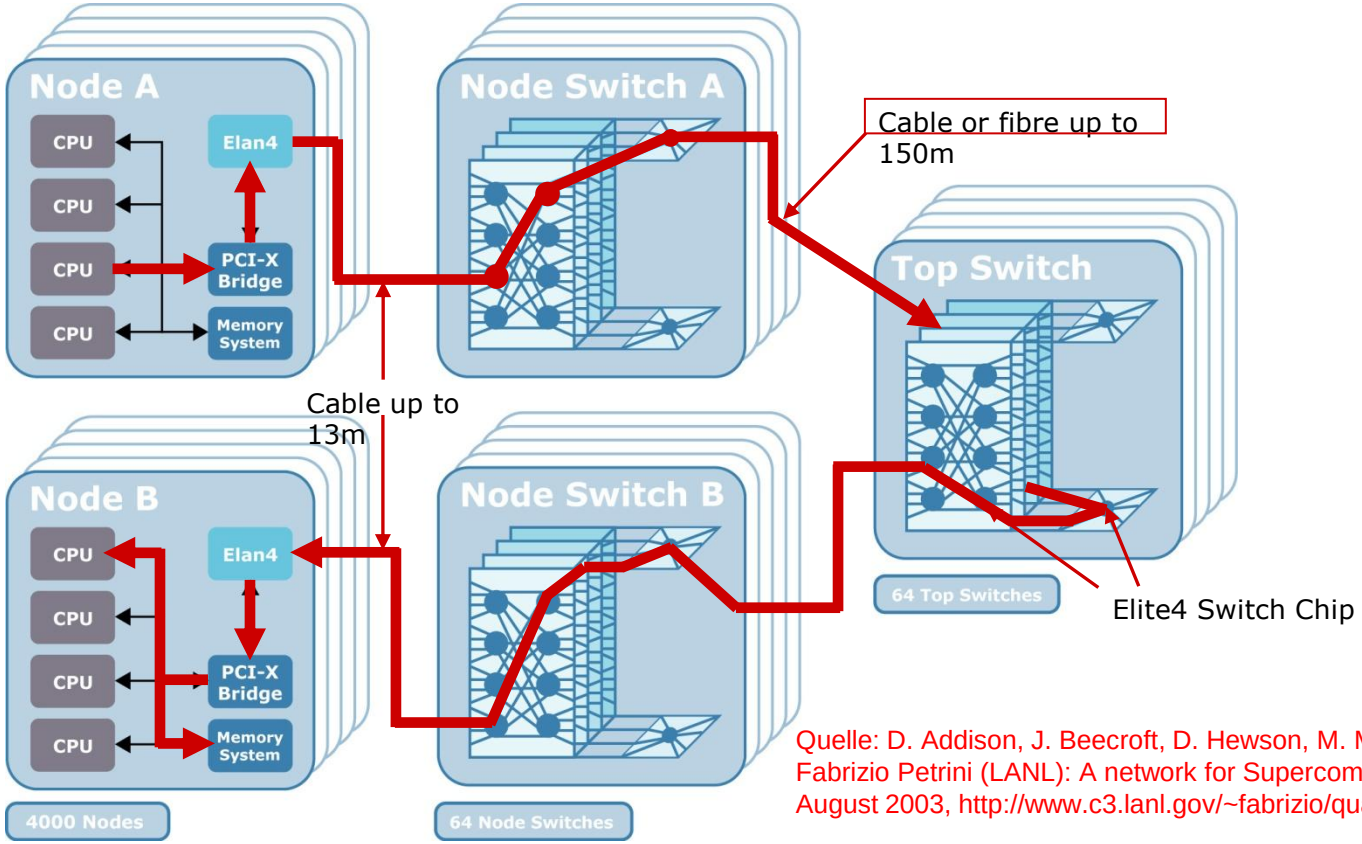
QsNet II Switch:

Quelle: D. Addison, J. Beecroft, D. Hewson, M. McLaren (Quadrics Ltd.), Fabrizio Petrini (LANL): A network for Supercomputing Applications. Hot Chips, August 2003, http://www.c3.lanl.gov/~fabrizio/quadrics.html

# Verbindungsstrukturen

## Topologie:

- **Statische Verbindungsnetze**
- Beispiel: Quadrics QSnet

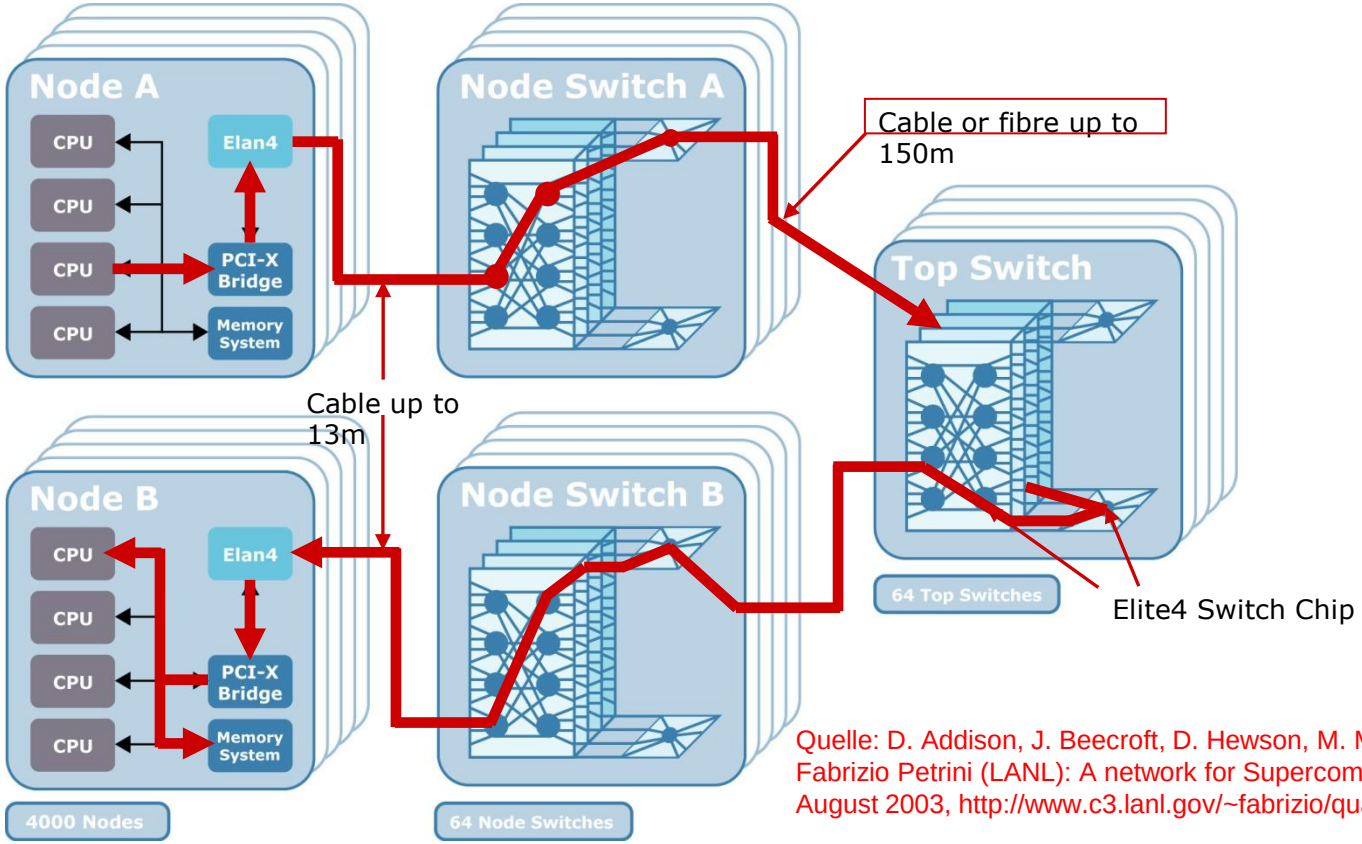



Quelle: D. Addison, J. Beecroft, D. Hewson, M. McLaren (Quadrics Ltd.), Fabrizio Petrini (LANL): A network for Supercomputing Applications. Hot Chips, August 2003, http://www.c3.lanl.gov/~fabrizio/quadrics.html

# Verbindungsstrukturen

## Topologie:

- **Statische Verbindungsnetze**
- Beispiel: Quadrics QSnet




Quelle: D. Addison, J. Beecroft, D. Hewson, M. McLaren (Quadrics Ltd.), Fabrizio Petrini (LANL): A network for Supercomputing Applications. Hot Chips, August 2003, http://www.c3.lanl.gov/~fabrizio/quadrics.html

# Verbindungsstrukturen

**Topologie:**

- **Statische Verbindungsnetze**
- **K-ärer n-Kubus (Cubes, Würfel)**
  - Allgemeine Form eines Kubus-Verbindungsnetzwerkes
  - Ringe, Gitter, oder Hyperkubi sind topologisch isomorph zu einer Familie von K-ären n-Kubus Netzwerken
    - n ist die Dimension
    - Radius K ist die Anzahl der Knoten, die einen Zyklus in einer Dimension bilden
  - Enthält $N=K^n$ Knoten
  - Die Knoten werden über eine n-stellige binäre Radius K Zahl der Form $a_0,a_1,\ldots,a_{n-1}$ adressiert
    - Jede Stelle $0 \leq a_i > K$ stellt die Position des Knotens in der entsprechenden i-ten Dimension dar, mit $0 \leq i \leq n-1$
    - Ein Nachbarknoten in der i-ten Dimension zu einem Knoten mit Adresse $a_0,a_1,\ldots,a_{n-1}$ kann erreicht werden mit $a_0,a_1,\ldots,a_{(i \pm 1)}$ mod k…$a_{n-1}$.
  - Knotengrad ist 2n und der Diameter ist

# Verbindungsstrukturen

**Topologie:**

- **Statische Verbindungsnetze**
- **Hyperkubus (Hypercubes)**
  - Verallgemeinerter Würfel:
    - die $N = 2^n$ Prozessoren sind Ecken eines *n*-dimensionalen Würfels, wobei die Verbindungen dann die Kanten des Würfels darstellen.
  - Komplexität ist $(N \cdot \log_2 N)/2$.
  - Diameter beträgt $\log_2 N$.
  - Lange Zeit häufigste Verbindungsstruktur bei den nachrichtengekoppelten Multiprozessoren,
    aber:
    - Skalierbarkeit:
      - Jede Erweiterung benötigt mindestens die Verdopplung der Prozessorenanzahl.
      - Aus räumlichen Anordnungsgründen begrenzt.

# Verbindungsstrukturen

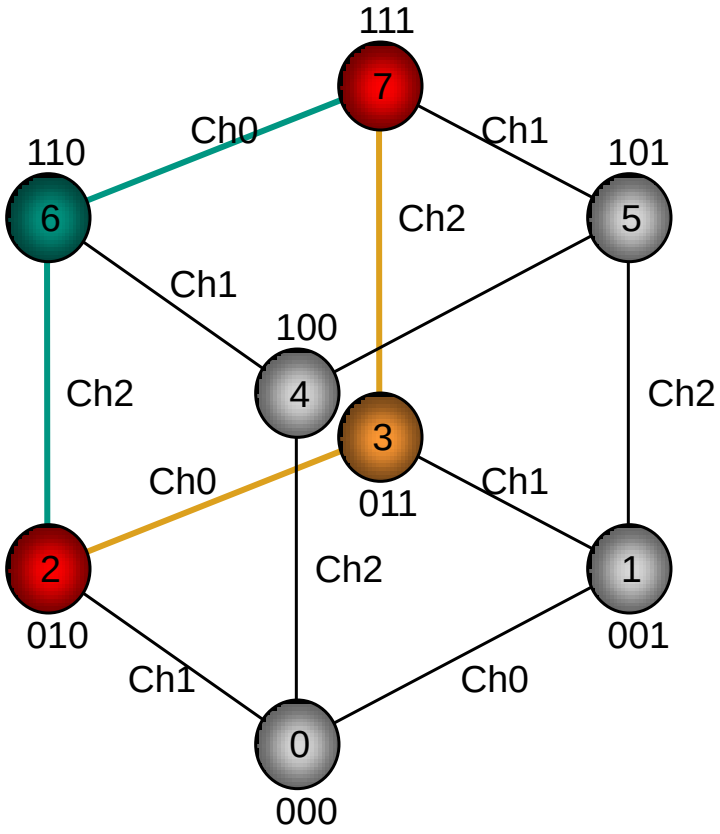
## Topologie:

- **Statische Verbindungsnetze**
- **Hyperkubus**
  - **e-Cube Routing**
    - Die Knotennummern werden als Binärzahlen geschrieben, dadurch unterscheiden sich benachbarte Knoten in genau einer Stelle, die zudem die Richtung der Verbindung angeben kann (Hamming Distanz)
    - Eine einfache Wegewahl:
      die Bits in Start- und Zieladresse werden mittels einer XOR-Verbindung verknüpft und das Resultat bestimmt die möglichen Wege.

# Verbindungsstrukturen

## Topologie:

- **Statische Verbindungsnetze**
- Hyperkubus
  - Beispiel:
    - A = (010) und B = (111)
    - W=A XOR B = 101
    - (010) → (011) → (111),
    - (010) → (110) → (111)



# Verbindungsstrukturen

## Topologie:

- **Dynamische Verbindungsnetzwerke:**
  - Geeignet für Anwendungen mit variablen und nicht regulären Kommunikationsmustern
- **Bus:**
  - Wird von den am Bus angeschlossenen Prozessoren gemeinsam benützt
  - Ein Datentransport zu einem Zeitpunkt
  - Nachricht von einer Quelle zu jedem Ziel in einem Schritt
  - Busbandbreite = w * f
    - w: Anzahl der Datenleitungen (Busbreite)
    - F: Frequenz
    - Bestimmt maximale Anzahl der Prozessoren, d. h. die Bandbreite muss mit dem Produkt der Anzahl der Prozessoren und ihrer Geschwindigkeit abgestimmt werden

# Verbindungsstrukturen

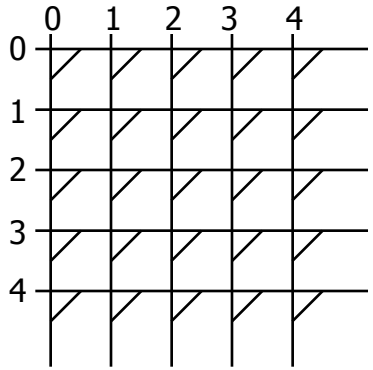
**Topologie:**

- **Dynamische Verbindungsnetzwerke:**
- **Bus:**
  - Reduzierung des Busverkehrs
    - Verwendung von Cache-Speichern mit Cache-Kohärenz-Protokollen
  - Verwendung von sog. Split-Phase Busprotokollen
    - Das Protokoll gibt den Bus nach der Übertragung einer Speichereferenzanforderung wieder frei
    - Wenn der Speicher bereit ist, das Datum zu liefern, fordert dieser den Bus an und schickt die Daten als Antwort
    - Ermöglicht, dass andere Prozessoren in der Zwischenzeit den Bus anfordern können, vorausgesetzt, dass ein verschränkter Speicher vorliegt oder Pipelining möglich ist

# Verbindungsstrukturen

## Topologie:

- **Dynamische Verbindungsnetzwerke:**
- **Kreuzschienenverteiler (Crossbar)**
  - Vollständig vernetztes Verbindungswerk mit allen möglichen Permutationen der N Einheiten, die über das Netzwerk verbunden werden



# Verbindungsstrukturen

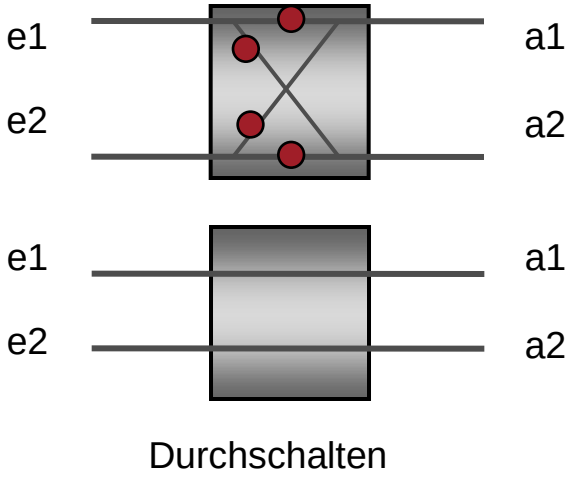
## Topologie:

- **Dynamische Verbindungsnetzwerke:**
- **Kreuzschienenverteiler (Crossbar)**
  - Hardware-Einrichtung, die so geschaltet werden kann, dass in einer Menge von Prozessoren alle möglichen disjunkten Paare von Prozessoren gleichzeitig und blockierungsfrei miteinander kommunizieren können.
    - In Abhängigkeit vom Zustand der Schaltelemente im Kreuzschienenverteiler können dann je zwei beliebige Elemente aus den verschiedenen Mengen miteinander kommunizieren.
    - Alle N! Permutationen sind möglich
    - An den Kreuzungspunkten sitzen Schaltelemente: hoher Hardware-Aufwand
    - Kosten: $N^2$ Schaltelemente (bei N Knoten pro Dimension)
    - Ein Schaltelement entspricht einem Paar von Quelle und Ziel, so dass die Darstellung einer Permutation als eine Liste solcher Paare direkt zu der korrekten Schaltung der Schalterelemente führt.

# Verbindungsstrukturen

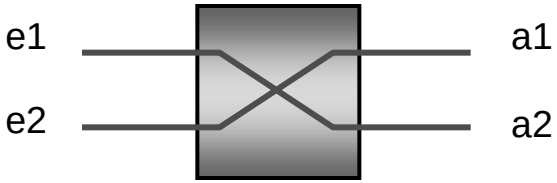
## Topologie:

- **Dynamische Verbindungsnetzwerke:**
- **Schalterelemente (2x2 Kreuzschienenverteiler)**
  - bestehen aus Zweierschaltern mit zwei Eingängen und zwei Ausgängen, die entweder durchschalten oder die Ein- und Ausgänge überkreuzen können




Durchschalten

- Kontakt, der geöffnet oder geschlossen werden kann




Vertauschen

# Verbindungsstrukturen

**Topologie:**

- **Dynamische Verbindungsnetzwerke:**
- **Mehrstufige Verbindungsnetzwerke (Schalternetzwerke, Permutationsnetzwerke)**
  - Kompromiss zwischen der niedrigeren Leistungsfähigkeit von Bussen und hohem Hardware-Aufwand von Kreuzschienenverteilern
  - Oft 2 x 2 Kreuzschienenverteiler (Schalterelement) als Grundelement

# Verbindungsstrukturen

**Topologie:**

- **Dynamische Verbindungsnetzwerke:**
- **Permutationsnetze**
  - p Eingänge des Netzes können gleichzeitig auf p Ausgänge geschaltet werden und somit wird eine Permutation der Eingänge erzeugt.
- **Einstufige Permutationsnetze**
  - enthalten eine einzelne Spalte von Zweierschaltern,
- **Mehrstufige Permutationsnetze**
  - enthalten mehrere solcher Spalten
  - Spalten: Stufen des Permutationsnetzwerkes

# Verbindungsstrukturen

**Topologie:**

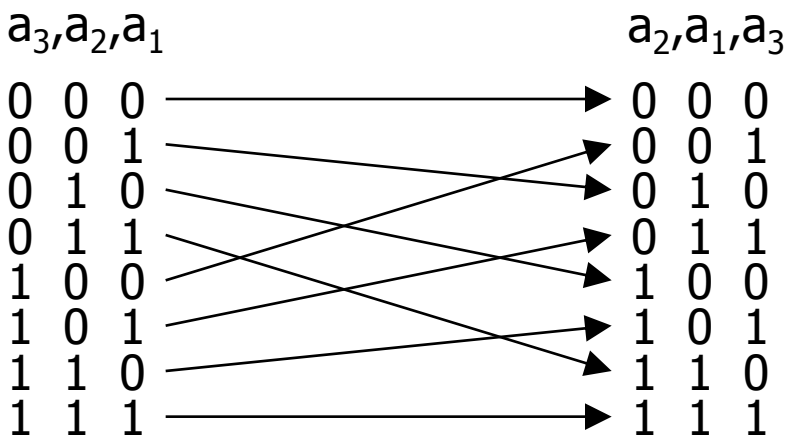
- **Dynamische Verbindungsnetzwerke:**
- **Permutationsnetze**
- **reguläre Permutationsnetzwerke:**
  - p Eingänge, p Ausgänge und k Stufen mit jeweils p/2 Zweierschaltern, wobei die Zahl p normalerweise eine Zweierpotenz ist.

- **Irreguläre Permutationsnetzwerke**
  - weisen gegenüber der vollen regulären Struktur Lücken auf

# Verbindungsstrukturen

## Topologie:

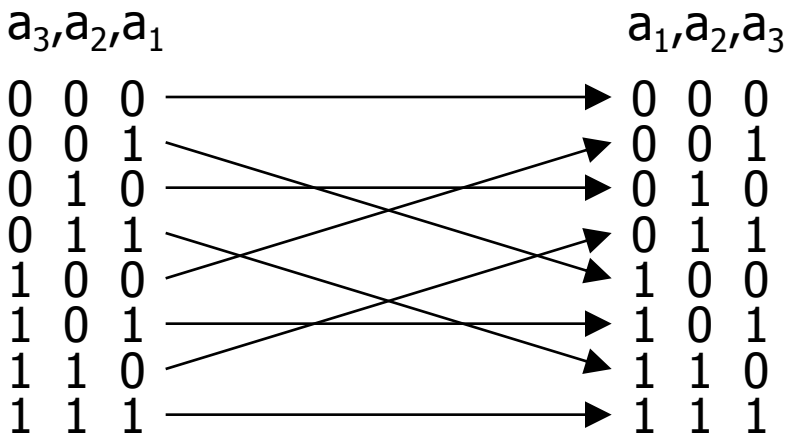
- **Dynamische Verbindungsnetzwerke:**
- **Permutationen**
  - **Mischpermutation M** (Perfect Shuffle):
    - $M(a_n,a_{n-1},\ldots a_2,a_1)=(a_{n-1},\ldots a_2,a_1,a_n)$



# Verbindungsstrukturen

## Topologie:

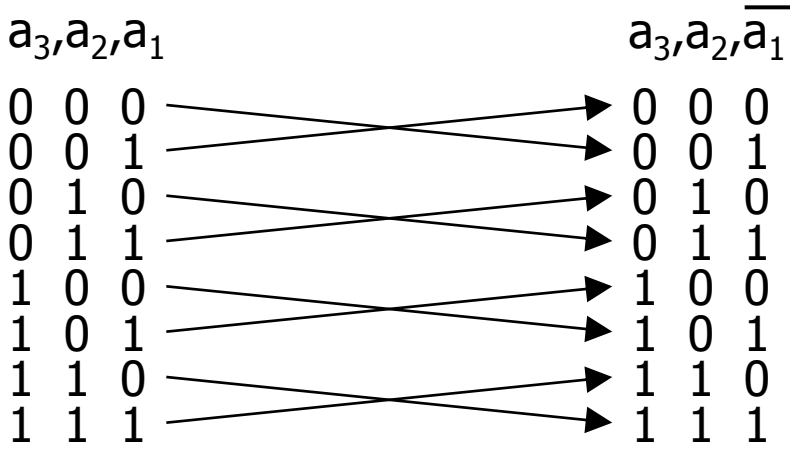
- **Dynamische Verbindungsnetzwerke:**
- **Permutationen**
  - **Kreuzpermutation K** (Butterfly):
    - $K(a_n,a_{n-1},\ldots a_2,a_1)=(a_1,a_{n-1}\ldots a_2,a_n)$



# Verbindungsstrukturen

## Topologie:

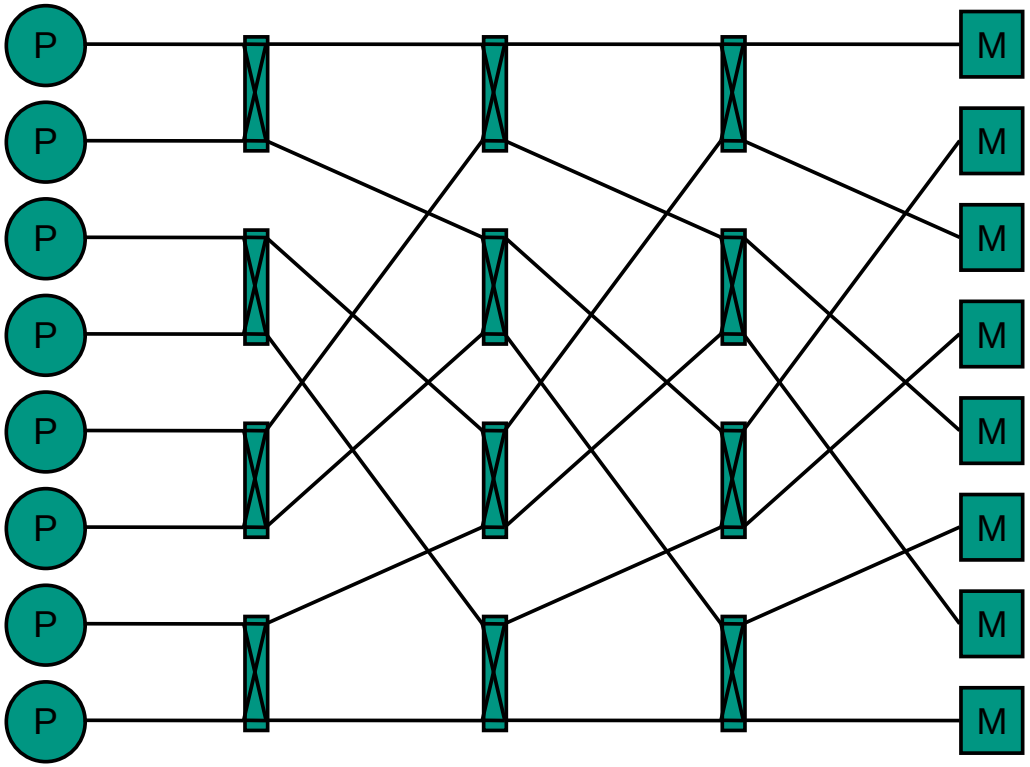
- **Dynamische Verbindungsnetzwerke:**
- **Permutationen**
  - **Tauschpermutation T** (Butterfly):
    - Negation des niedrigwertigen Bits
    - $T(a_n,a_{n-1},\ldots a_2,a_1)=(a_n,a_{n-1}\ldots a_2,\overline{a_1})$



# Verbindungsstrukturen

## Topologie:

- Dynamische Verbindungsnetzwerke:
- Speichergekoppeltes Omega-Netzwerk mit 8 Eingängen und 8 Ausgängen



# Verbindungsstrukturen

## Topologie:

- Dynamische Verbindungsnetzwerke:
- Speichergekoppeltes Omega-Netzwerk mit 8 Eingängen und 8 Ausgängen