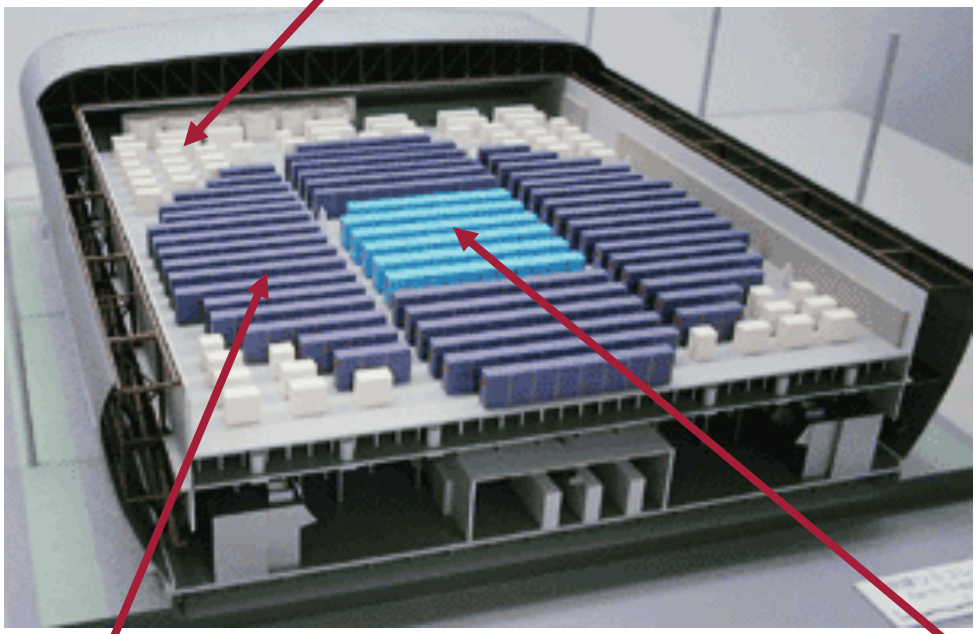
1. Parallelverarbeitung

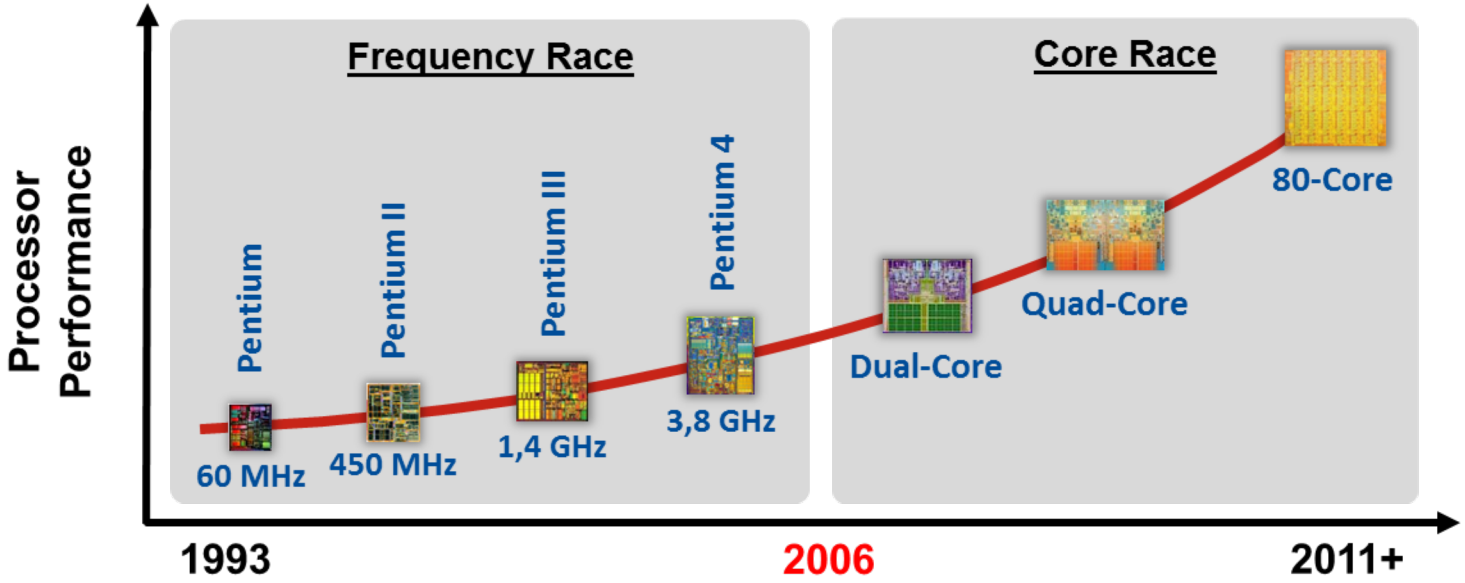
Supercomputer (Beispiel Earth Simulator, 2001)



Typische Probleme im Supercomputing

- **Grand Challenges:** Probleme, die ohne paralleles Rechnen in „vernünftiger Zeit“ nicht lösbar sind.
  - Klimaforschung
  - Erdbebensimulation
  - Simulation von Galaxien
- Rechenintensive Probleme mit „kurzer“ Deadline
  - Wettervorhersage
  - Simulationen im Entwicklungsprozess eines Produkts, z.B. virtuelle Crash-Tests bei Automobilentwicklung
  - Anwendungen mit Realzeitanforderungen, z.B. virtuelle Realität

Multi-Core CPUs



- **Prinzip:** Es werden mehrere vollständige Prozessoren (= Cores) auf einem Chip integriert
  - Anzahl der Cores verdoppelt sich ca. alle 2 Jahre (gem. Moore's Law.)
- Heutzutage allgegenwärtig, da andere Methoden der Leistungssteigerung (z.B. höhere Taktfrequenz) nicht mehr greifen.

➢ **Nicht parallelisierte Programme können nur einen Bruchteil der Leistungsfähigkeit nutzen (z.B. 25 % bei Quadcore)**

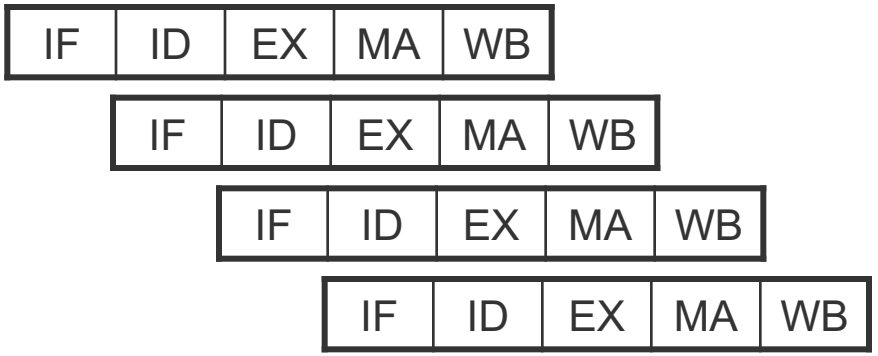
Parallelität in Rechnersystemen

Parallelität tritt auf unterschiedlichen Ebenen auf:

- **Bitebenenparallelität**: Datenbits werden zu Datenworten zusammengefasst und parallel verarbeitet.
- **Befehlsebenenparallelität**: Maschinenbefehle werden implizit parallel ausgeführt.
  - Pipelining: Überlappung der einzelnen Phasen der Befehlsausführung.
  - Superskalere Architektur: Betrieb mehrerer Pipelines.
- **Programmebenenparallelität**:
  - Parallelität wird durch geeignete Programmierkonstrukte oder Tools explizit im Programm festgelegt.

Exkurs: Parallelität auf Befehlsebene

- Pipeline-Architektur: Überlappende Ausführung der Verarbeitungsphasen eines Maschinenbefehls:
  - Instruction Fetch (IF)
  - Instruction Decode (ID)
  - Execute (EX)
  - Memory Access (MA)
  - Write Back (WB)



  - Ziel: Pro Takt wird ein Befehl abgeschlossen.
- Superskalare Architektur: Mehrere Pipelines (bzw. einzelne Stufen einer Pipeline) werden parallel betrieben.
  - Ziel: Pro Takt werden mehrere Befehle abgeschlossen.

Programmebenenparallelität

- **Bit- und Befehlsebenenparallelität** finden heute breite Verwendung.
- Warum macht Programmebenenparallelität Sinn?
  - **Performance**: Steigerung der absoluten Rechenleistung durch Verwendung mehrerer Prozessoren wird möglich.
  - **Software-Engineering:** Einfachere Programmerstellung durch Ausnutzung natürlicher Parallelität der Anwendung.
- **Paralleles Rechnen**: Beschleunigung der Berechnung eines Problems durch den Einsatz **mehrerer Prozessoren** und **Programmebenenparallelität.**

Wichtige Metriken der Parallelverarbeitung

- **Sequentielle Laufzeit T_S** : Zeit, die zwischen dem Programmstart und dem Programmende bei der Ausführung auf einem sequentiellen Rechner verstreicht.
- **Parallele Laufzeit T_p**: Zeit zwischen dem Start und dem Ende der parallelen Programmausführung auf p Prozessoren.
- **Speedup: $S := T_S / T_P$**
  - Maß für die durch Parallelverarbeitung erzielte Beschleunigung
- **Effizienz: $E := S / p$**
  - Maß für den Ausnutzungsgrad des Parallelrechners

2. Parallelrechner

1. Klassifikation von Parallelrechner-Architekturen
2. Verbindungsnetzwerke für Parallelrechner
3. Trends bei Parallelrechner-Architekturen

2. Parallelrechner

1. **Klassifikation von Parallelrechner-Architekturen**
2. Verbindungsnetzwerke für Parallelrechner
3. Trends bei Parallelrechner-Architekturen

Klassifikation von Parallelrechnern

- Klassifikation anhand der **logischen Organisation** des Parallelrechners.
- Logische Organisation: Wie stellen sich dem Programmierer die folgenden Aspekte dar:
  - **Kontrollstruktur**
    Mechanismen zur Darstellung paralleler Abläufe (Tasks)
  - **Kommunikationsmodell**
    Mechanismen zur Kommunikation zwischen parallelen Tasks
- Wichtig: Es werden zur Klassifikation nur Eigenschaften betrachtet, die direkt von der Hardware realisiert und nicht mittels Software emuliert werden.

Klassifikation anhand der Kontrollstruktur (Flynn, 1972)

- Wie viele unterschiedliche Befehle können gleichzeitig bearbeitet werden?
- Auf wie vielen Datenworten wird ein Befehl gleichzeitig angewendet?

- **SISD:** Single Instruction Stream, Single Data Stream
  - Konventioneller Rechner mit von Neumann Architektur
- **SIMD:** Single Instruction Stream, Multiple Data Streams
- **MISD:** Multiple Instruction Streams, Single Data Stream
  - kein Vertreter bekannt
- **MIMD:** Multiple Instruction Streams, Multiple Data Streams

SIMD Parallelrechner

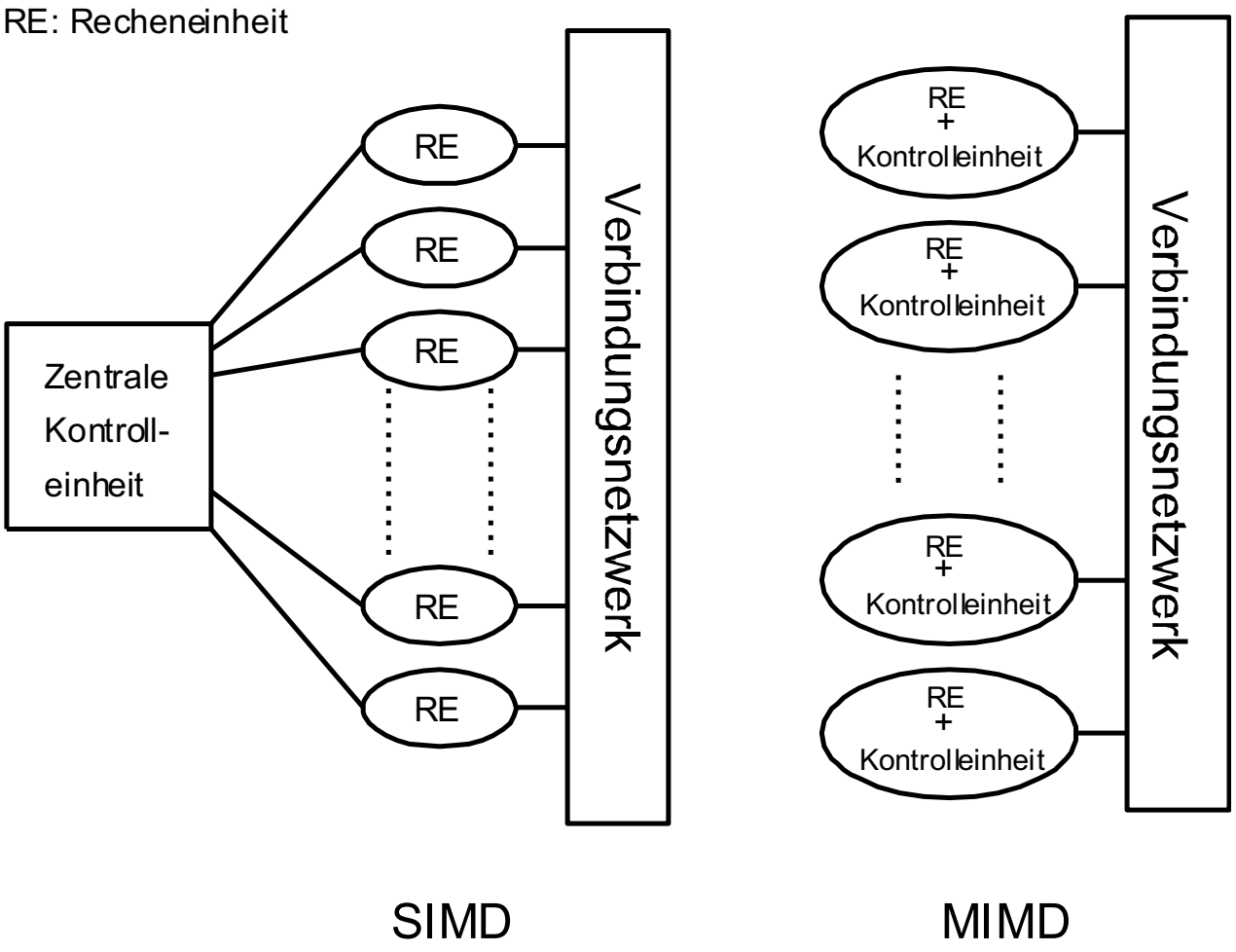
- Zentrale Kontrolleinheit schickt Befehle zu den einzelnen Recheneinheiten.
- Zu jedem Zeitpunkt führen die Recheneinheiten den selben Befehl aus.
  - Einzelne Recheneinheiten können ggf. maskiert (d.h. deaktiviert) werden.
- SIMD Parallelrechner sind nur für reguläre Probleme geeignet, z.B. Vektoraddition:

```
for (int i=0; i<100; i++)
    c[i]=a[i]+b[i];
```

MIMD Parallelrechner

- Jede Recheneinheit besitzt **eigene Kontrolleinheit**.
- Recheneinheiten können zu einem Zeitpunkt unterschiedliche Befehle ausführen.
- Programmierung oftmals mittels SPMD (Single Program, Multiple Data) Konzept:
  - Jeder Recheneinheit führt eine Instanz des selben Quell-Programms aus.
  - Vereinfachung für den Programmierer, da nicht für jede Recheneinheit ein eigenes Programm erstellt werden muss.
  - Die Recheneinheiten bearbeiten in der Regel unterschiedliche Programmteile (z.B. unterschiedliche Funktionen).
  - SPMD „gleich mächtig“ wie MIMD.

Vergleich SIMD-MIMD



SIMD vs. MIMD

Vorteile SIMD

- Realisierung von SIMD Parallelrechnern ist einfacher, da diese nur eine Kontrolleinheit benötigen.
- SIMD Parallelrechner benötigen weniger Speicher, da das Programm nur einmal gespeichert werden muss.

Vorteile MIMD

- Nicht auf die Verarbeitung von regulären Problemen beschränkt.
- MIMD Parallelrechner lassen sich einfach aus Standardkomponenten aufbauen, z.B. PC Cluster.

Multicore CPU vs. GPU

- GPU-Design verwendet einen weitaus größeren Teil der Transistoren für Recheneinheiten.
  - Hohe Rechenleistung, aber beschränkt auf reguläre Probleme, da SIMD Parallelität.

(Quelle: NVIDIA CUDA™ Programming Guide)

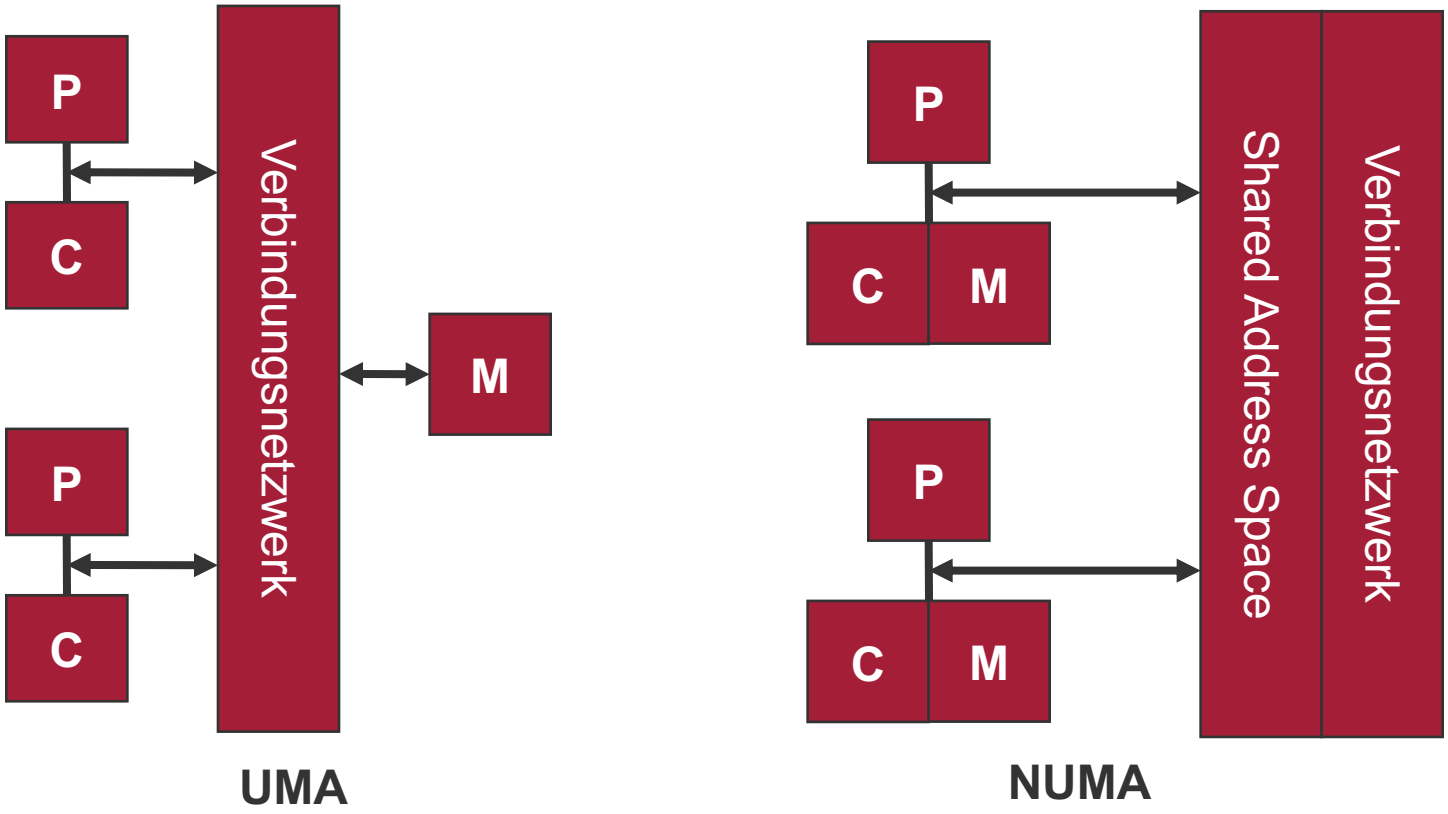
Klassifikation anhand des Kommunikationsmodells

- Klassifikation anhand der **Semantik der Kommunikation** zwischen den Recheneinheiten.
- Parallelrechner mit gemeinsamem Adressraum **(Shared-Address-Space)**
  - Kommunikation über Zugriff auf Speicheradressen
  - Read/Write Semantik
  - auch Multiprozessor genannt
- Nachrichten basierte Parallelrechner **(Message Passing)**
  - Kommunikation über Austausch von Nachrichten
  - Send/Receive Semantik
  - auch Multicomputer genannt

Parallelrechner mit gemeinsamem Adressraum

- **UMA (Uniform Memory Access)**
  - Recheneinheiten und Speicherelemente sind direkt verbunden
  - Für jedes Rechenelement ist die Zugriffszeit auf jedes Datenwort des gemeinsamen Adressraums gleich groß.
- **NUMA (Non-Uniform Memory Access)**
  - Speicherelemente sind bei Recheneinheiten lokalisiert.
  - Die Zugriffszeiten für Datenworte unterscheiden sich.
  - Parallele Programme müssen diese Eigenschaft berücksichtigen, um effizient ausgeführt werden zu können. (Lokalität der Daten).

Parallelrechner mit gemeinsamem Adressraum



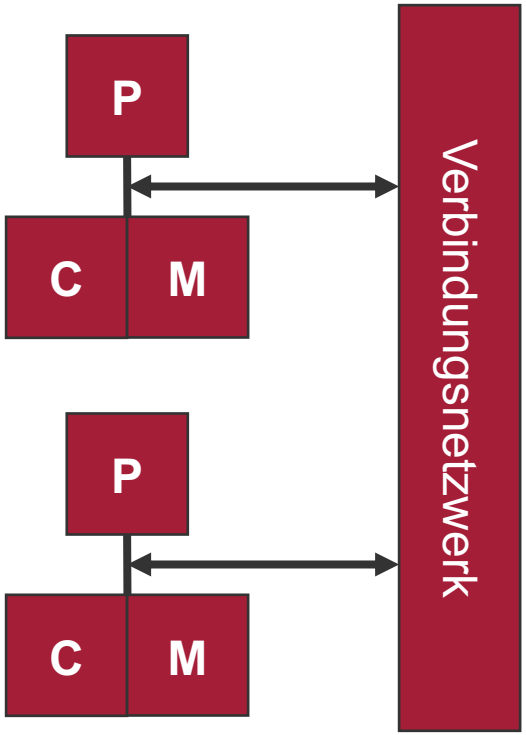
- P = Prozessor, C = Cache, M = Memory
- Cache Kohärenz Problematik (→ VL Rechnerarchitektur)

Nachrichten basierte Parallelrechner

- Speicherelemente sind bei Recheneinheiten lokalisiert.
- Jeder **Knoten** (Recheneinheit + Speicherelement) besitzt privaten Adressraum.
- Knoten haben ID zur Adressierung von Nachrichten.
- Minimale Funktionalität umfasst:
  - Abfrage der eigenen ID (rank)
  - Abfrage der Anzahl der Knoten (size)
  - Send Operation (send)
  - Receive Operation (receive)
- Oft auch **hierarchisches Design**: Die einzelnen Knoten sind Parallelrechnern mit gemeinsamem Adressraum.

Nachrichten basierte Parallelrechner

- Ähnlich zu NUMA, hier aber keine HW Unterstützung für gemeinsamen Adressraum.



Shared Address Space vs. Message Passing

UMA

- Programmierung einfach, ähnlich wie von Neumann Rechner.
- Alle Speicherzugriffe gehen über Verbindungsnetzwerk.
  - Anzahl der Prozessoren beschränkt oder sehr aufwändiges Verbindungsnetzwerk erforderlich.

NUMA

- Lokaler Speicher entlastet Verbindungsnetzwerk.
- Programmierer muss Lokalität der Daten berücksichtigen.

Message Passing

- Geringerer Hardwareaufwand
- Viele Prozessoren möglich, da Speicherzugriffe immer lokal sind.
- Programmierung schwierig

Typen Nachrichten basierter Parallelrechnern

- **Massively Parallel Processors (MPPs)**
  - Große Zahl an Knoten, meistens mit Standard Prozessoren
  - Speziell gefertigte Hochleistungs-Verbindungsnetzwerke
  - Robustheit und Redundanz
- **Cluster**
  - Aus (autarken) Standard PCs oder Workstations aufgebaut.
  - Standard Netzwerke, z.B. Ethernet oder Myrinet
- **Network of Workstations (Desktop Grids)**
  - Heterogene Komponenten
  - Meist gleichzeitig Verwendung als Arbeitsplatzrechner
- **Computational Grids**
  - Geographisch verteilte Parallelrechner aller Klassen
  - Unterschiedliche Besitzer/ administrative Domänen

2. Parallelrechner

1. Klassifikation von Parallelrechner-Architekturen
2. **Verbindungsnetzwerke für Parallelrechner**
3. Trends bei Parallelrechner-Architekturen

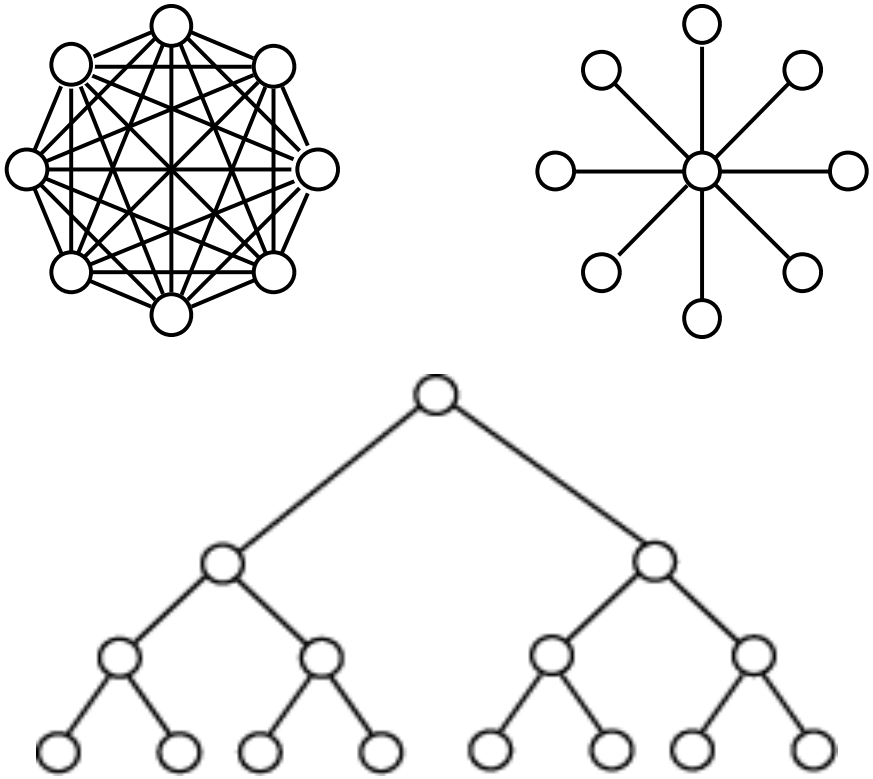
Verbindungsnetzwerke

- Verbindungsnetzwerke ermöglichen den Datentransfer zwischen
  - einzelnen Rechenelementen und
  - zwischen Rechen- und Speicherelementen.
- **Statische Verbindungsnetzwerke**
  - Feste Punkt-zu-Punkt Verbindungen (Links) zw. Elementen
- **Dynamische Verbindungsnetzwerke**
  - Elemente sind an Eingangs- bzw. Ausgangs-Ports angebunden.
  - **Aktive Komponenten** stellen dynamisch einen Pfad zwischen Eingangs- und Ausgangs-Ports her.
  - Zusätzliche Funktionalität:
    - Zwischenspeicherung von Daten, falls Ausgangs-Port belegt ist.
    - Multicasting: Eingangs-Port wird mit mehreren Ausgangs-Ports verbunden

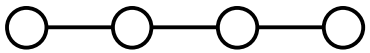
Leistungsbewertung von Verbindungen

- **Latenz:** Übertragungszeit für eine Nachricht, die keine Nutzdaten enthält.
  - Latenz umfasst Hardware- und Protokoll-Overhead
  - Verringerung der Latenz z.B. durch
    - Verkürzung von Schaltzeiten
    - einfache Protokolle und ggf. Unterstützung durch spezielle Protokoll-Prozessoren.
- **Bandbreite:** Anzahl der Bits, die pro Zeiteinheit übertragen werden können.
  - Erhöhung der Bandbreite z.B. durch parallele Übertragung (Bitebenenparallelität).

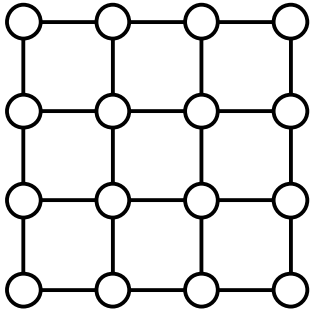
Statische Verbindungsnetzwerke: Vollständig, Stern, Baum



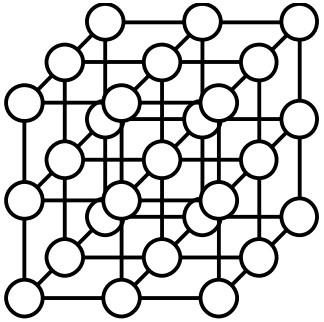
Statische Verbindungsnetzwerke: Array, Torus



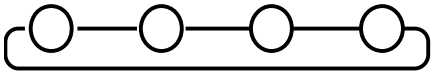
1D Array



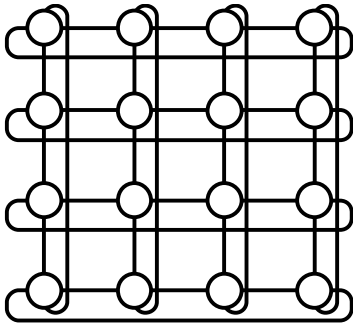
2D Array



3D Array

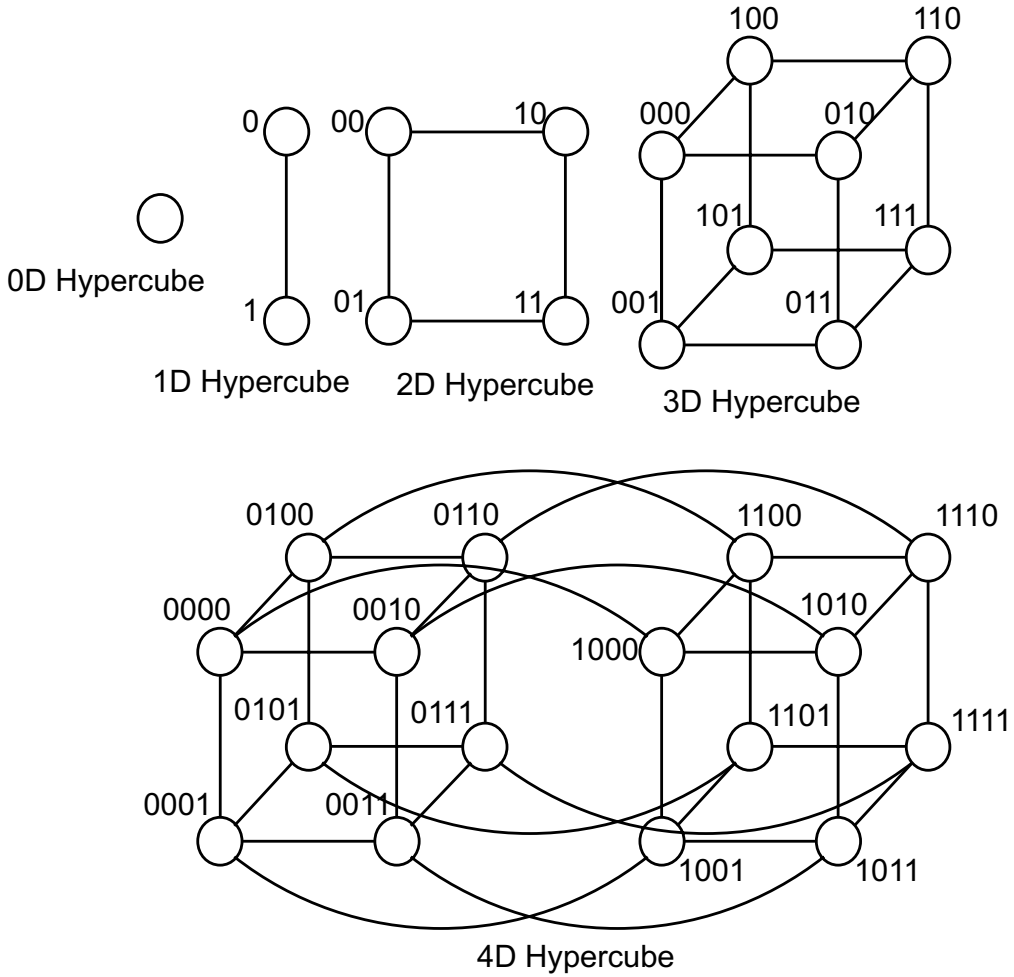


1D Torus (Ring)



2D Torus

Statische Verbindungsnetzwerke: Hypercube



Leistungsbewertung statischer Verbindungs-netzwerke

- **Kosten:** Gesamtzahl der Links
- **Entfernung:** Kürzester Pfad (Anzahl der Links) zwischen zwei Elementen.
- **Durchmesser:** Maximal vorkommende Entfernung zwischen zwei Elementen des Netzwerks.
- **Bisektionsbreite:** Minimale Anzahl von Links, die durchtrennt werden müssten, um das Netzwerk in zwei (ungefähr) gleiche Hälften zu teilen.
  - Wenn jedes Element in der einen Hälfte eine Nachricht an ein Element der anderen Hälfte schickt, müssen alle Nachrichten über die Links der Bisektionsbreite übertragen werden
  - „Bisektionsbreite ist der Flaschenhals des Netzwerks“

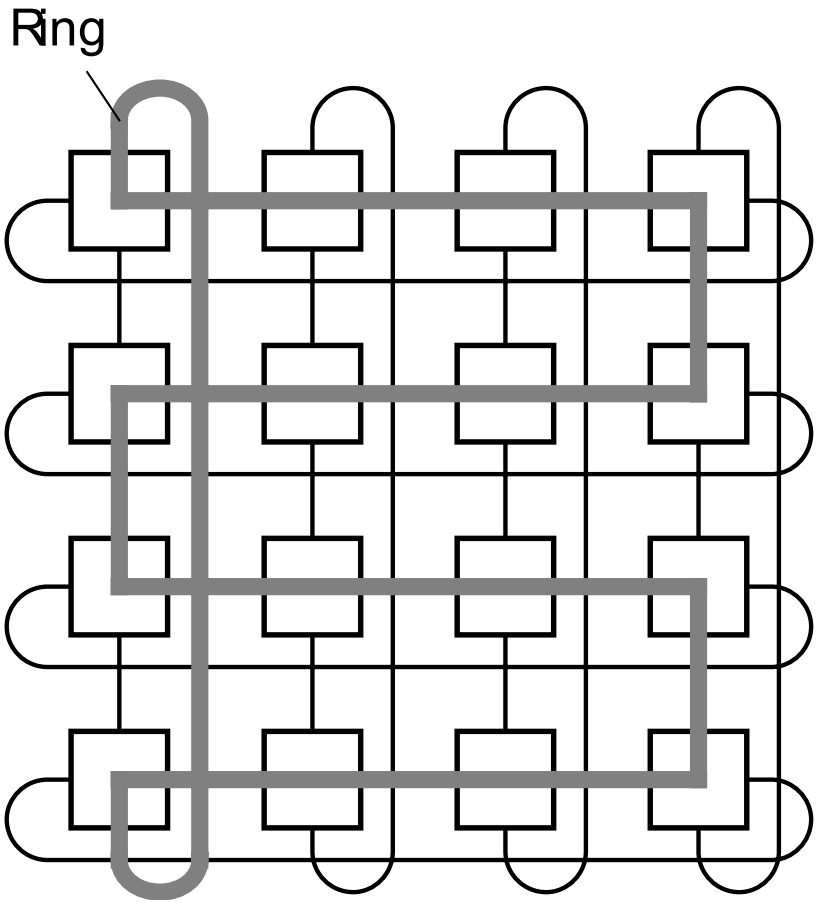
Leistungsbewertung statischer Verbindungs-netzwerke

Netzwerk	Kosten	Durchmesser	Bisektionsbreite
Vollständig	$p\,(p - 1) / 2$	1	$p^2 / 4$
Stern	$p - 1$	2	1
2D Array	$2(p - \sqrt{p})$	$2(\sqrt{p} - 1)$	$\sqrt{p}$
2D Torus	$2\,p$	$2\,\lfloor \sqrt{p}/2 \rfloor$	$2\,\sqrt{p}$
Hypercube	$(p \log p) / 2$	$\log p$	$p / 2$

Einbettung statischer Netzwerke

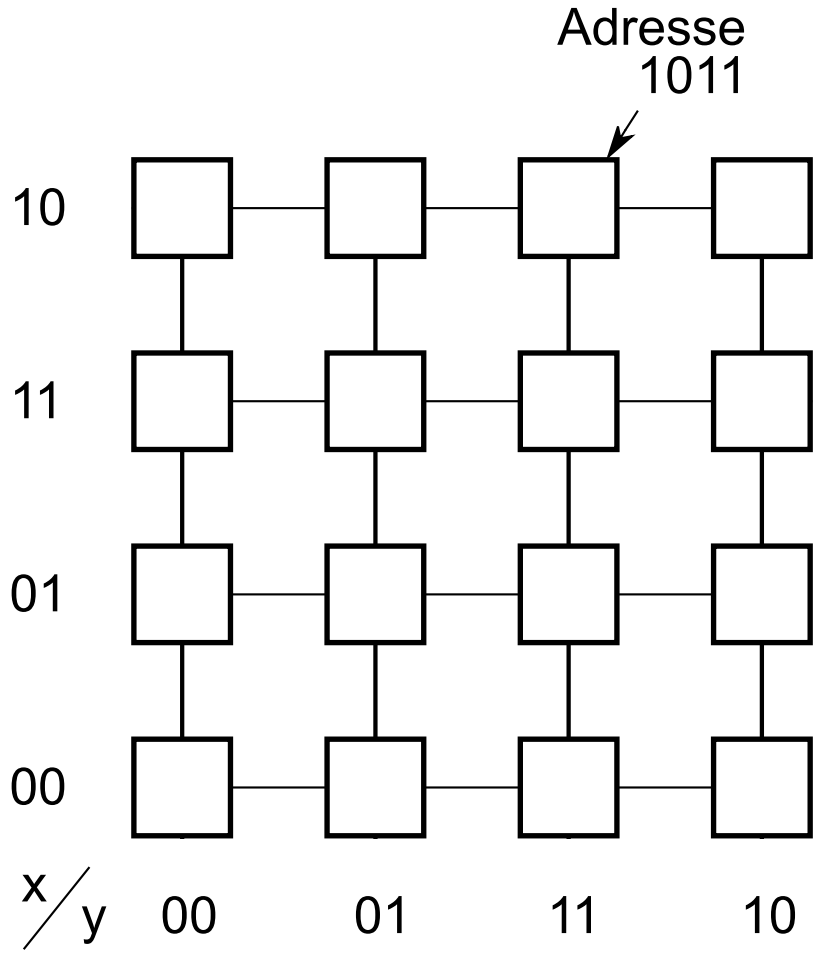
- Eine **Einbettung (Embedding)** ist eine Abbildung der Knoten eines Verbindungsnetzwerks auf die Knoten eines Zielnetzwerks mit einer anderen Topologie.
  - Parallele Programme sind häufig speziell für ein bestimmtes Verbindungsnetzwerk optimiert.
  - Durch Einbettung sollen solche Programme ohne Änderungen in anderen Verbindungsnetzwerken effizient ausgeführt werden können.
- Maß für die Güte einer Einbettung: **Ausdehnung (Dilation)** Die größte vorkommende Anzahl an Links im Zielnetzwerk, auf die ein einzelner Link im eingebetteten Netzwerk abgebildet („ausgedehnt“) wird.
- Eine Einbettung ist **perfekt**, wenn ihre Ausdehnung 1 ist.

Einbettung: Ring in Torus



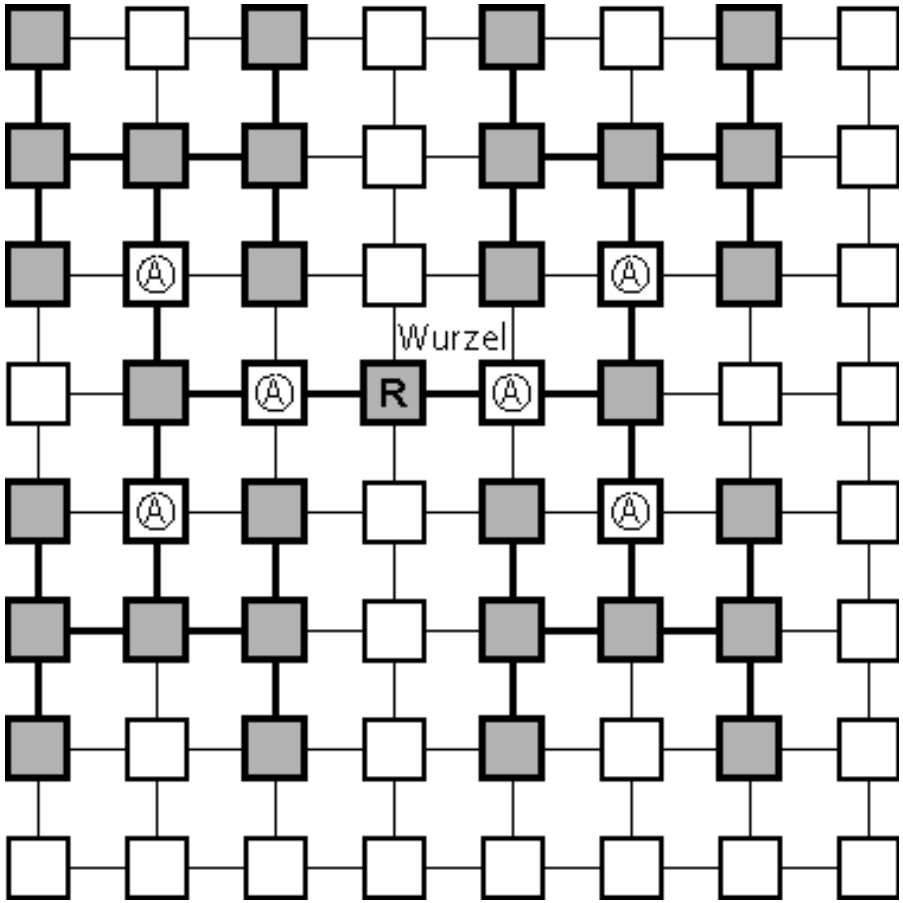
- Ausdehnung ist 1

Einbettung: 2D Array in Hypercube



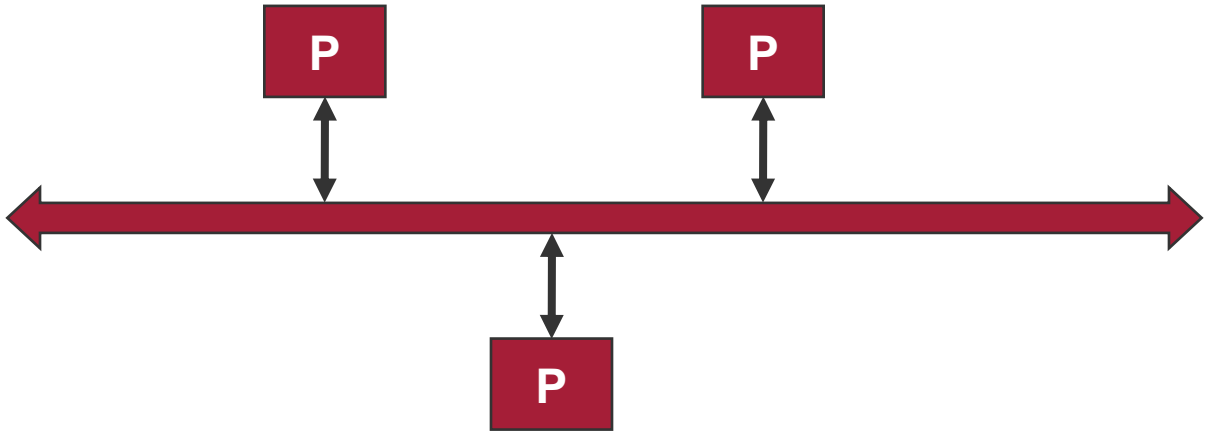
- Die Koordinaten der Knoten des 2D Arrays werden im **Gray Code** nummeriert.
- Ausdehnung ist 1

Einbettung: Baum in 2D Array



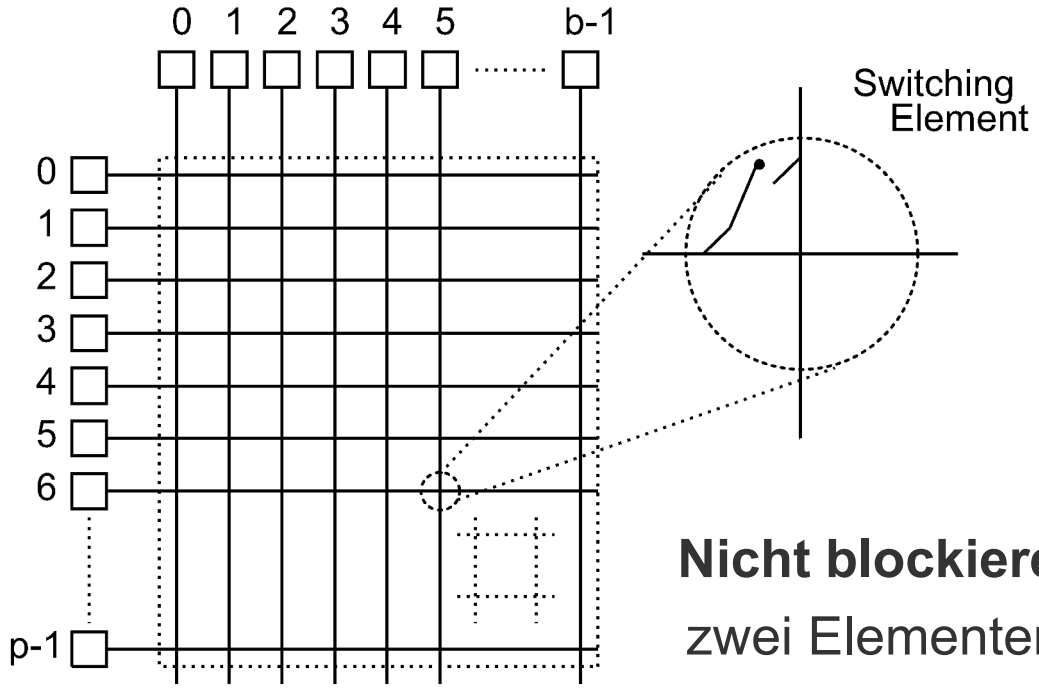
- Ausdehnung ist 2

Dynamisches Verbindungsnetzwerk: Bus



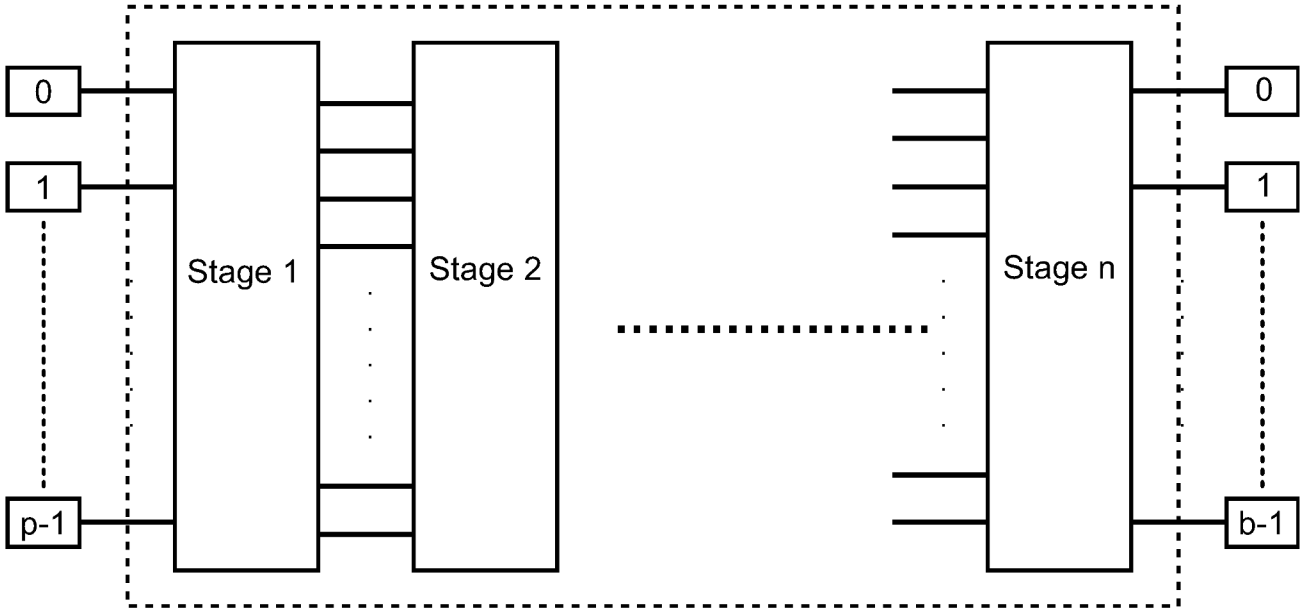
- Alle Elemente teilen sich ein Übertragungsmedium.
- Kosten steigen linear mit der Anzahl der Elemente.
- Distanz zwischen zwei Elementen ist konstant.
- Broadcasting Operationen nicht teurer als Punkt-zu-Punkt Kommunikation.
- **Nachteil:** Elemente teilen sich Bandbreite des Mediums.

Dynamisches Verbindungsnetzwerk: Crossbar-Switch



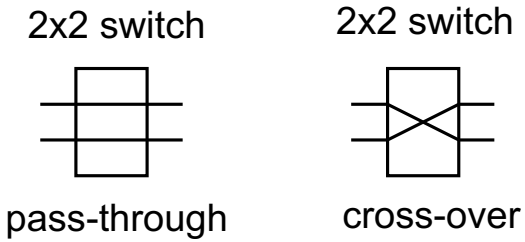
Nicht blockierend: Verbindung zw. zwei Elementen blockiert nicht die Verbindung zweier anderer Elemente

Dynamisches Verbindungsnetzwerk: Multistage



Beispiel: Omega Multistage-Netzwerk

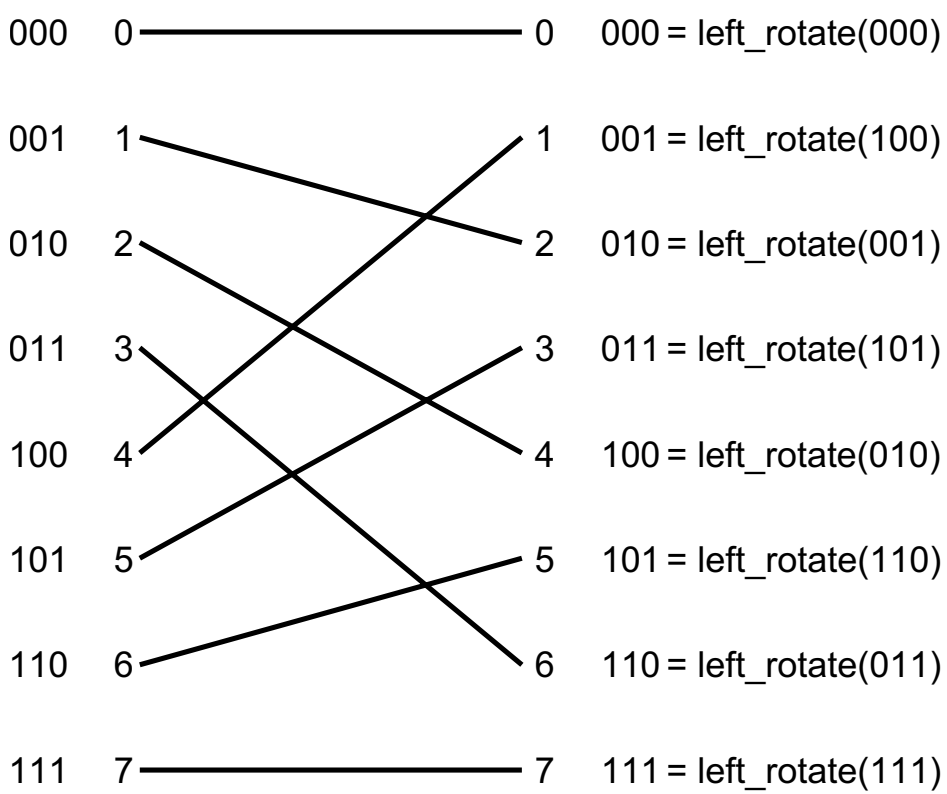
- p Eingänge und p Ausgänge
- log p Stufen (mit jeweils p Eingängen und p Ausgängen).
- Jede Stufe besteht aus p/2 2x2-Switch-Elementen.
- Jedes Switch-Element kennt 2 verschieden Verbindungsmodi:
  - **pass-through** Modus
  - **cross-over** Modus



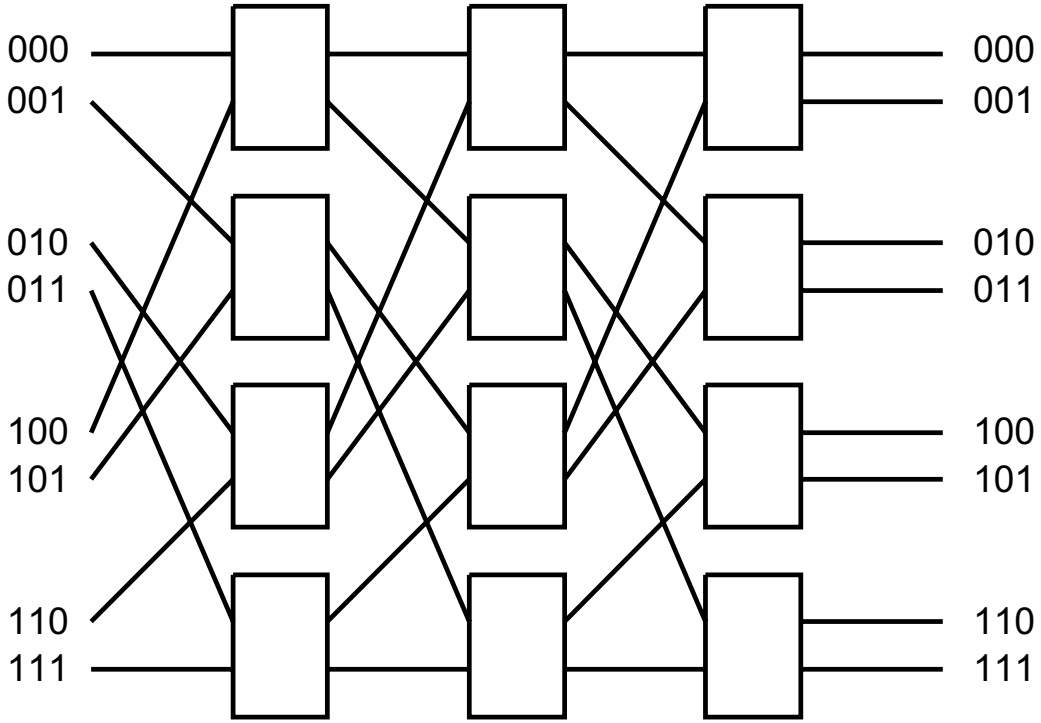
- Die einzelnen Stufen sind nach dem **perfect shuffle** Prinzip verbunden.

Omega Netzwerk
Perfect Shuffle Prinzip

Beispiel: p=8



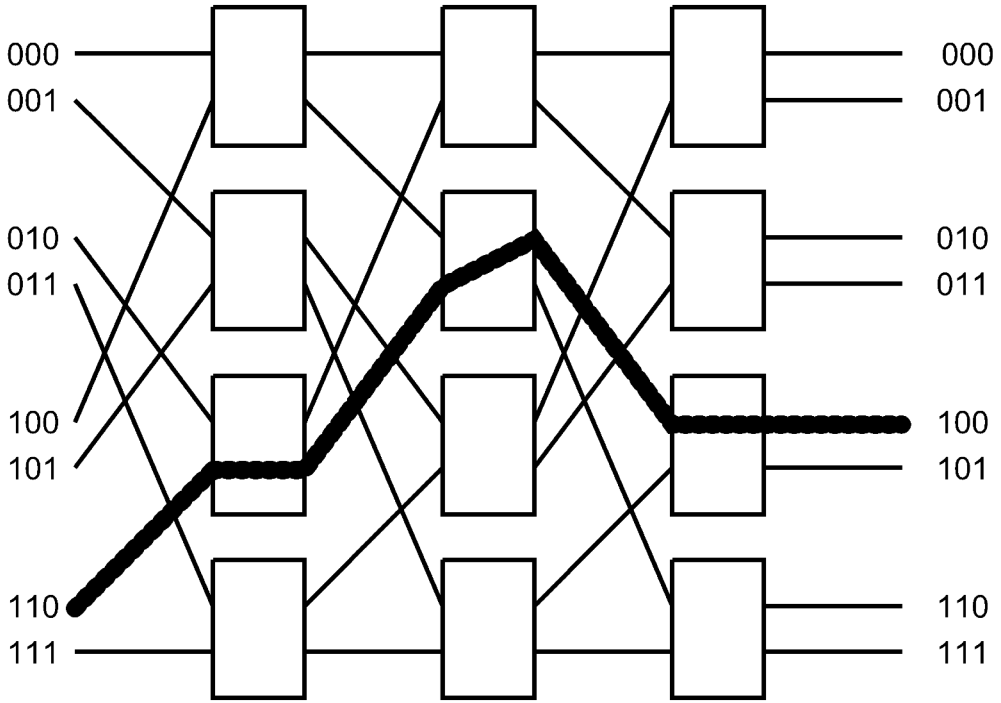
Aufbau eines Omega Netzwerks



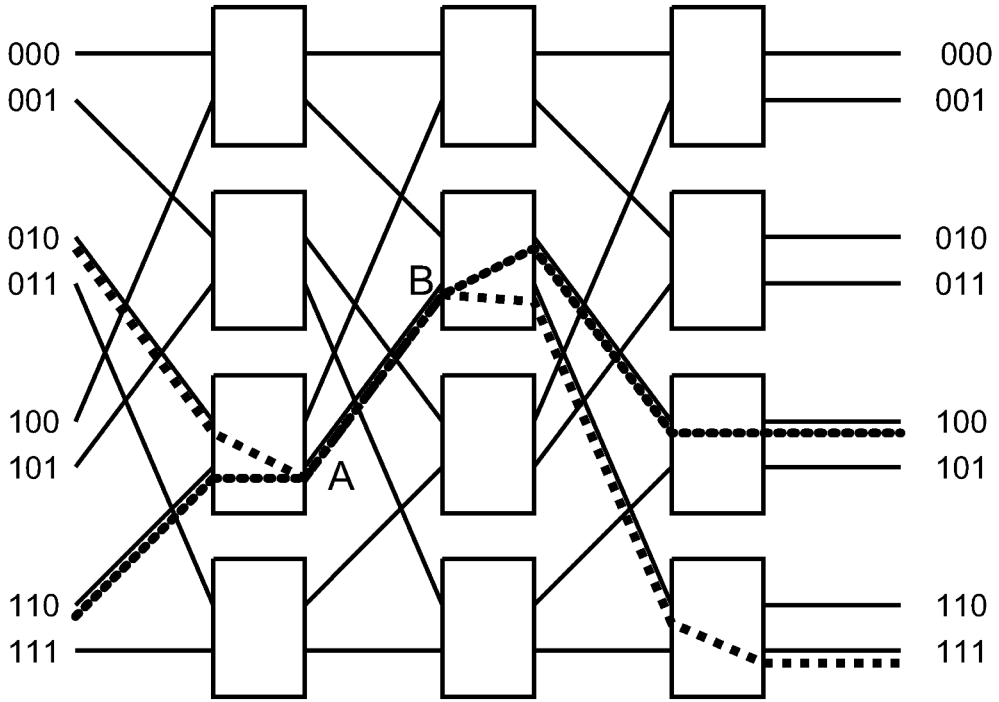
Routing im Omega Netzwerk

- Jede Stufe ist für die Verarbeitung einer Bitposition der Adressen zuständig.
  - Stufe 1 beginnt mit höchstwertigstem Bit
- In jeder Stufe wird das betreffende Bit der Senderadresse mit dem Bit der Empfängeradresse verglichen.
  - Bits sind gleich: Switch-Element arbeitet im pass through Modus.
  - Bits sind unterschiedlich: Switch-Element arbeitet im cross over Modus.

Routing im Omega Netzwerk



Blockierung im Omega Netzwerk



Vergleich dynamischer Verbindungsnetzwerke

- Bus:
  - Skalierbar bezüglich Aufwand
  - Nicht skalierbar bezüglich Leistung
- Crossbar
  - Skalierbar bezüglich Leistung
  - Nicht skalierbar bezüglich Aufwand
- Multistage
  - Kompromiss aus Leistung und Aufwand

2. Parallelrechner

1. Klassifikation von Parallelrechner-Architekturen
2. Verbindungsnetzwerke für Parallelrechner
3. **Trends bei Parallelrechner-Architekturen**

Entwicklung der Parallelrechner

- **Top500** Liste der 500 leistungsstärksten Rechner
- 2 mal jährlich aktualisiert (Juni/November) seit 1993
- Leistungsbewertung basiert auf Linpack Benchmark
  - Lösen eines linearen Gleichungssystems (Ax=b)
    - Dichtbesetzte Matrix A mit zufällig erzeugten Einträgen
    - Problemgröße N frei wählbar
  - R_{MAX} ist die maximale erreichte Anzahl an Fließkomma-operationen pro Sekunde (Flops).
  - N_{MAX} ist die Problemgröße bei der R_{MAX} erreicht wurde.
  - R_{PEAK} gibt die theoretisch erreichbare Anzahl an Fließkomma-operationen pro Sekunde an.
- Aktuelle Liste (Quelle: www.top500.org)

Prognostizierte Eigenschaften künftiger Exascale-Architekturen

- **Ziel:** Steigerung der Rechenleistung um Faktor 1000 bis zum Jahr 2020 (Petaflop → Exaflop)
- Leistung wird im Wesentlichen durch extreme Parallelität erzielt:
  - 10^8 – 10^9 Cores (ca. 100 Cores pro Chip)
  - 10 – 100 Threads pro Core (zum Verbergen von Speicher- und Netzwerklatenzen)
  - Kombination verschiedener Arten von Cores
    - MIMD vs. SIMD
    - teilweise auch applikationsspezifische Cores
  - ➢ Programmierung extrem schwierig

Prognostizierte Eigenschaften künftiger Exascale-Architekturen

- Gesamtsystem besitzt deutlich geringere Zuverlässigkeit
  - Hohe Anzahl von Komponenten
  - Aufgrund der hohen Integrationsdichte weisen Transistoren zunehmend probabilistisches Verhalten auf.
  - ➢ Fehlertoleranz ist wichtiges Querschnittsthema
- Energieverbrauch
  - Ca. 100 MW Leistungsaufnahme
  - „Politisch-ökonomische Schmerzgrenze“: 25 MW
  - ➢ Energieverbrauch wird wichtiges Merkmal einer Applikation (gleichranging mit der erzielten parallelen Effizienz)