Parallel Programming Concepts

Parallel Computing Hardware / Models / Connection Networks

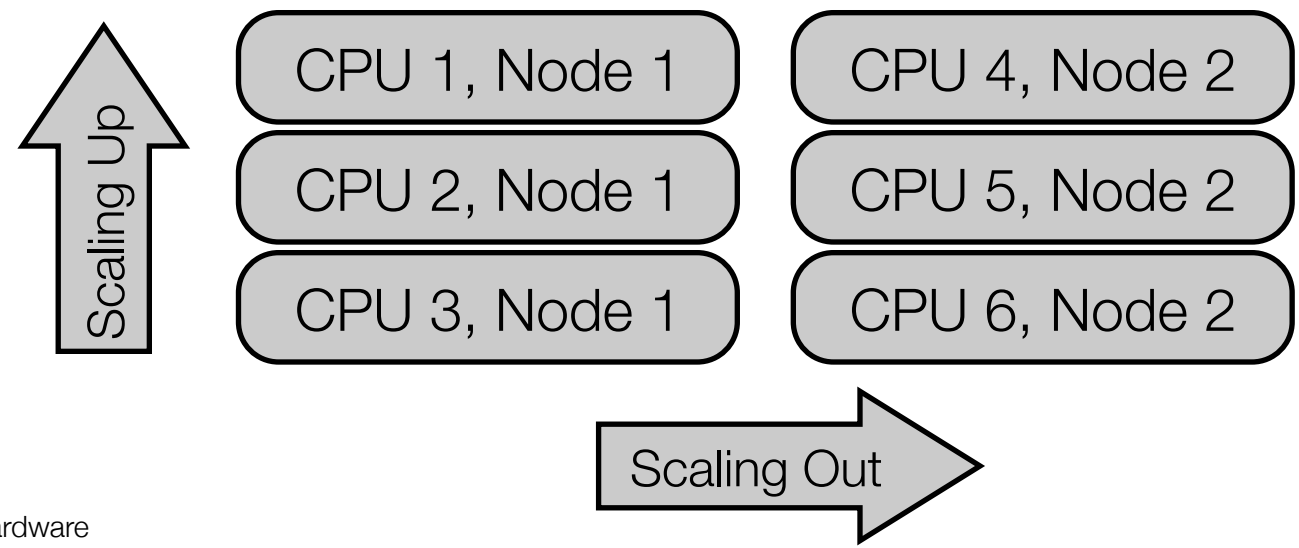
Andreas Polze, Peter Tröger

Sources:

Clay Breshears: The Art of Concurrency, Chapter 6
Blaise Barney: Introduction to Parallel Computing
Culler, David E. et al.: LogP: Towards a Realistic Model of Parallel Computation.
Valiant, Leslie G.: A bridging model for parallel computation. In: Commun. ACM 33 (1990), Nr. 8, S. 103-111
Andreas Polze; Vorhersagbares Rechnen in Multicomputersystemen Habilitationsschrift an der Mathematisch-Naturwissenschaftlichen Fakultät der Humboldt-Universität zu Berlin, November 2001.

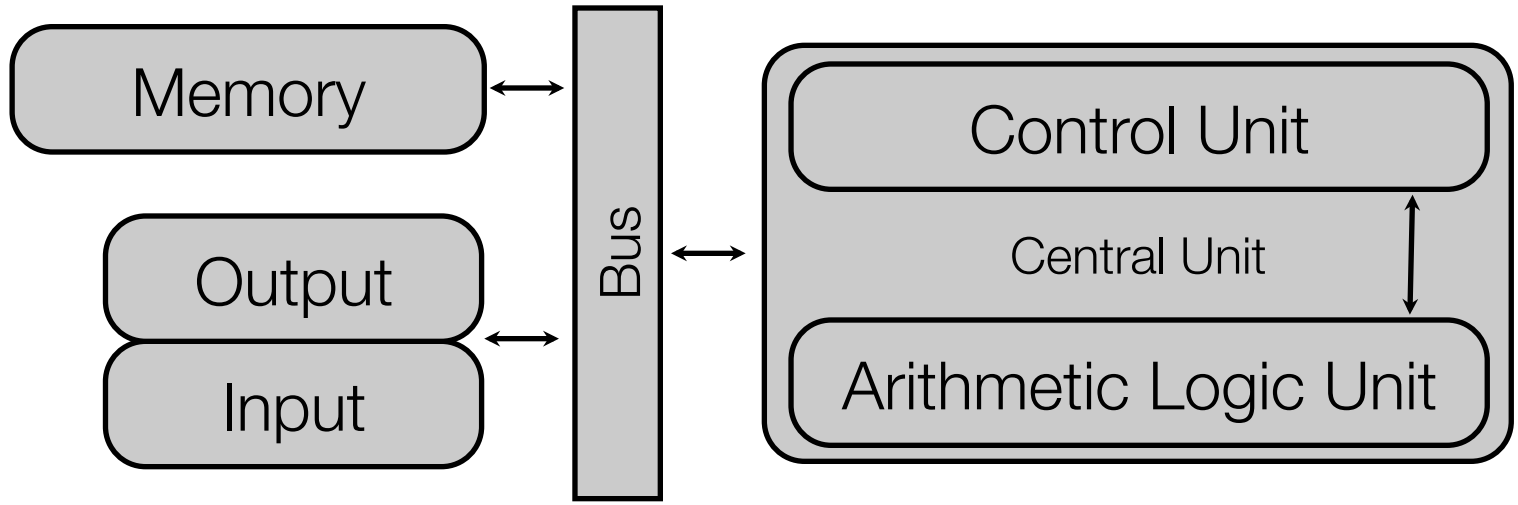
Reason for choosing a parallel architecture

- Performance - do it faster
- Throughput - do more of it in the same time
- Availability - do it without interruption
- Price / performance - do it as fast as possible for the given money
- Scalability - be able to do it faster with more resources
- Scavenging - do it with what I already have



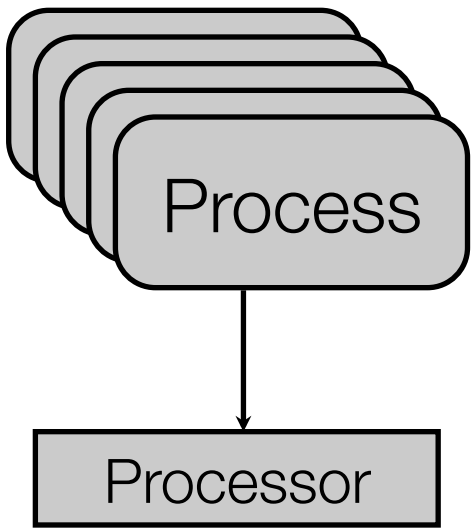
Machine Model

- First computers had fixed programs (electronic calculator)
- *von Neumann architecture* (1945, for EDVAC project)
  - Instruction set used for assembling programs stored in memory
  - Program is treated as data, which allows program exchange under program control and self-modification
  - von Neumann bottleneck



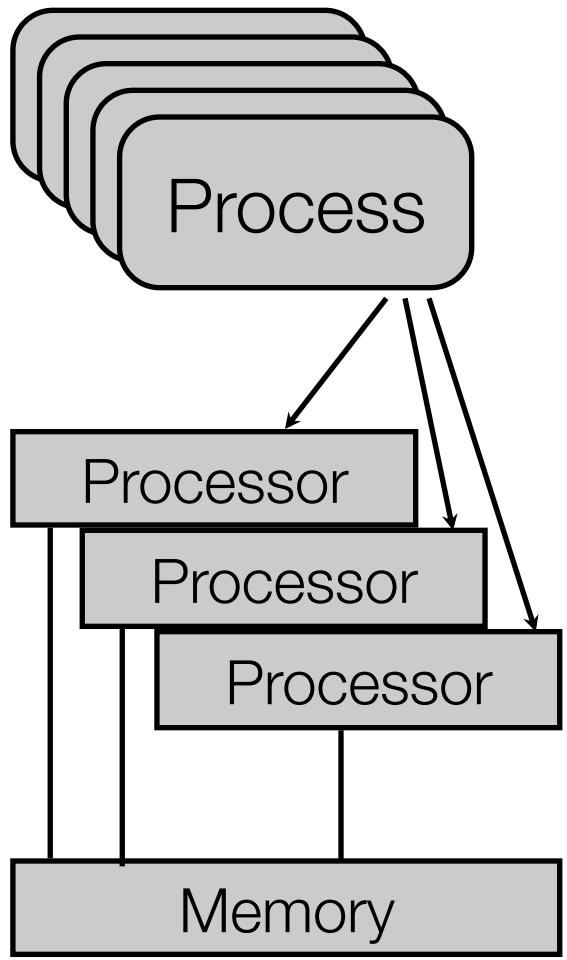
Parallel Computers - Vocabulary

Uniprocessor System

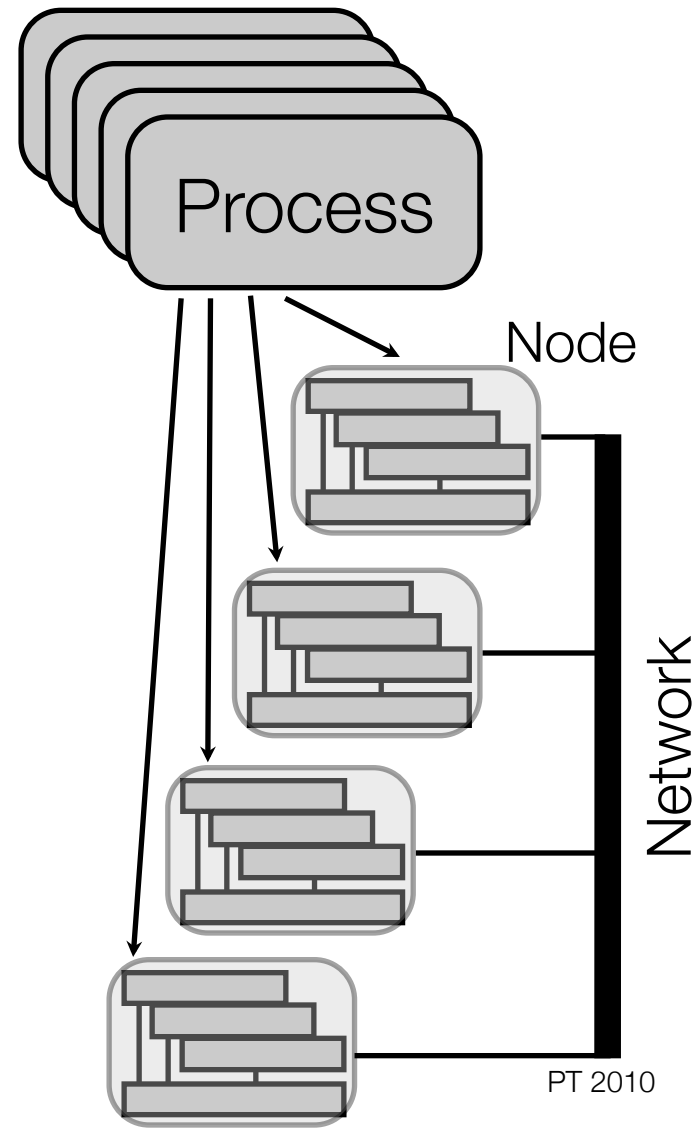


- Pipelining
- Super-scalar
- VLIW
- Branch prediction
- ...

Multiprocessor System



Multicomputer System



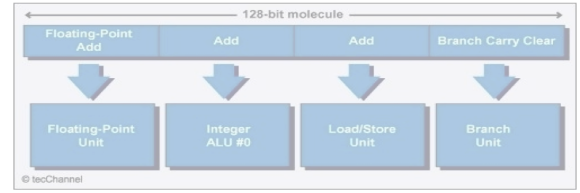
RISC vs. CISC - Computer Architecture History

- CISC - Complex Instruction Set Computer
  - VAX, Intel X86, IBM 360/370, etc.
- Large number of complex instructions
- Variable length instructions
- Extensive manipulation of low-level computational elements and events such as memory, binary arithmetic, and addressing

- RISC - Reduced Instruction Set Computer
  - MIPS, DEC Alpha, SUN Sparc, IBM 801
- Small number of instructions
- instruction size constant
- Fewer addressing modes
- instructions that can be overlapped and made to execute in one machine cycle or less (pipelining)

- RISC designs lend themself to eploitation of instruction level parallelism
- Very Long Instruction Word – VLIW – Transmeta Crusoe
- Explicitely Parallel Instruction Set Computing – EPIC – Intel Itanium

Instruction-Level Parallelism

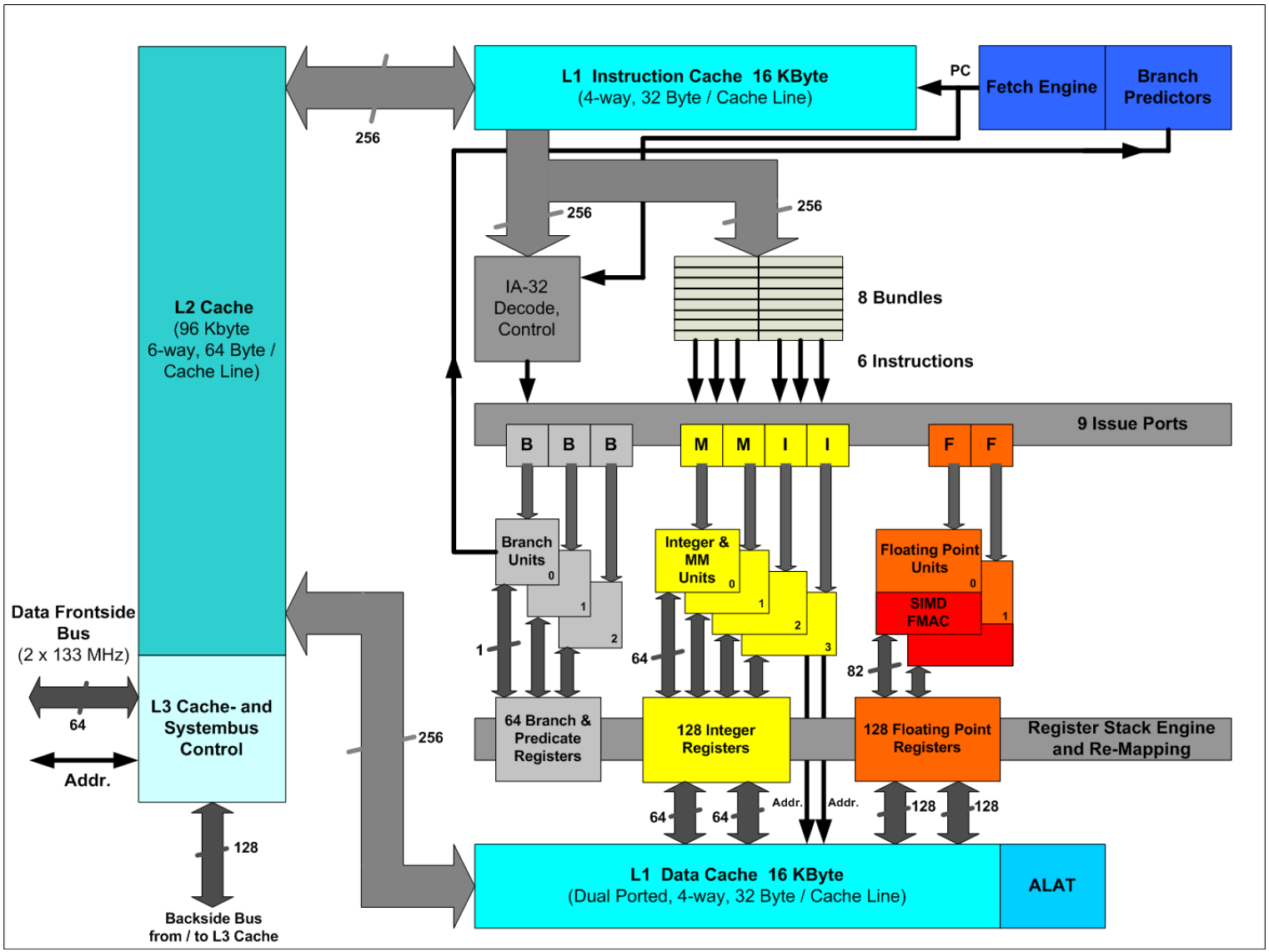


- Processor hardware optimizes instruction stream execution
  - Sub-steps of sequential instructions are executed in parallel (**pipelining**)
  - Execution of multiple instructions in parallel (**superscalar** architecture)
  - Re-arrangement of the order of instructions (**out-of-order execution**)
- **Very Long Instruction Word (VLIW)**
  - Fisher et al., 1980‘s
  - Compiler identifies instructions to be executed in parallel (code bloat)
  - Less hardware complexity, higher compiler complexity
  - VLIW processors usually designed as multiple RISC execution units
  - Success with IA-64 (EPIC) and Transmeta Crusoe, embedded market

EPIC – Itanium architecture (X64)

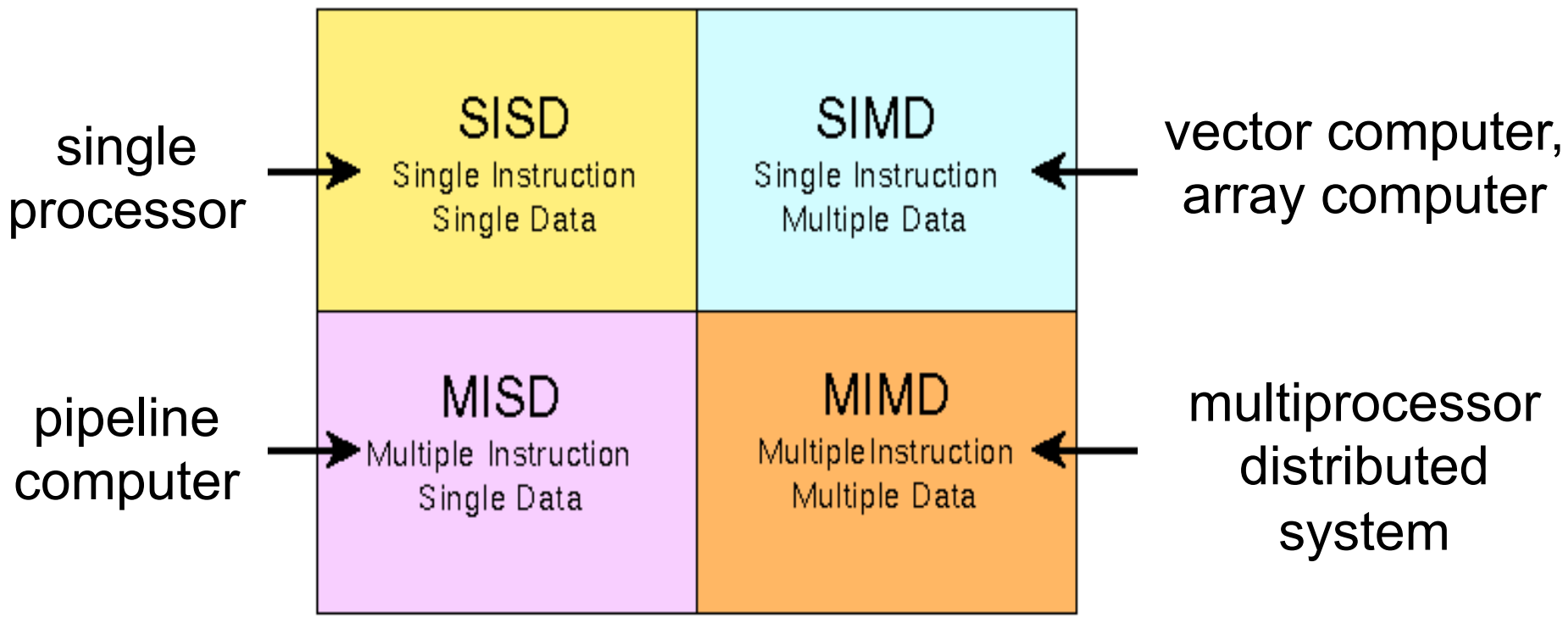
- 64-bit register-rich explicitly-parallel architecture
- implements predication, speculation, and branch prediction
  - hardware register renaming for parameter passing
  - parallel execution of loops
- Speculation, prediction, predication, and renaming controlled by compiler:
  - Each 128-bit instruction word contains three instructions; stop-bits control parallel execution
  - fetch mechanism can read up to two instruction words per clock from the L1 cache into the pipeline
  - processor can execute six instructions per clock cycle
  - thirty functional execution units for particular subsets of instruction set in eleven groups.
  - each unit executes at a rate of one instruction per cycle unless execution stalls waiting for data
  - common instructions can be executed in multiple units.

Itanium architecture – 30 functional units



- Six general-purpose ALUs, two integer units, one shift unit
- Four data cache units
- Six multimedia units, two parallel shift units, one parallel multiply, one population count
- Two 82-bit floating-point multiply-accumulate units, two SIMD floating-point multiply-accumulate units (two 32-bit operations each)[52]
- Three branch units

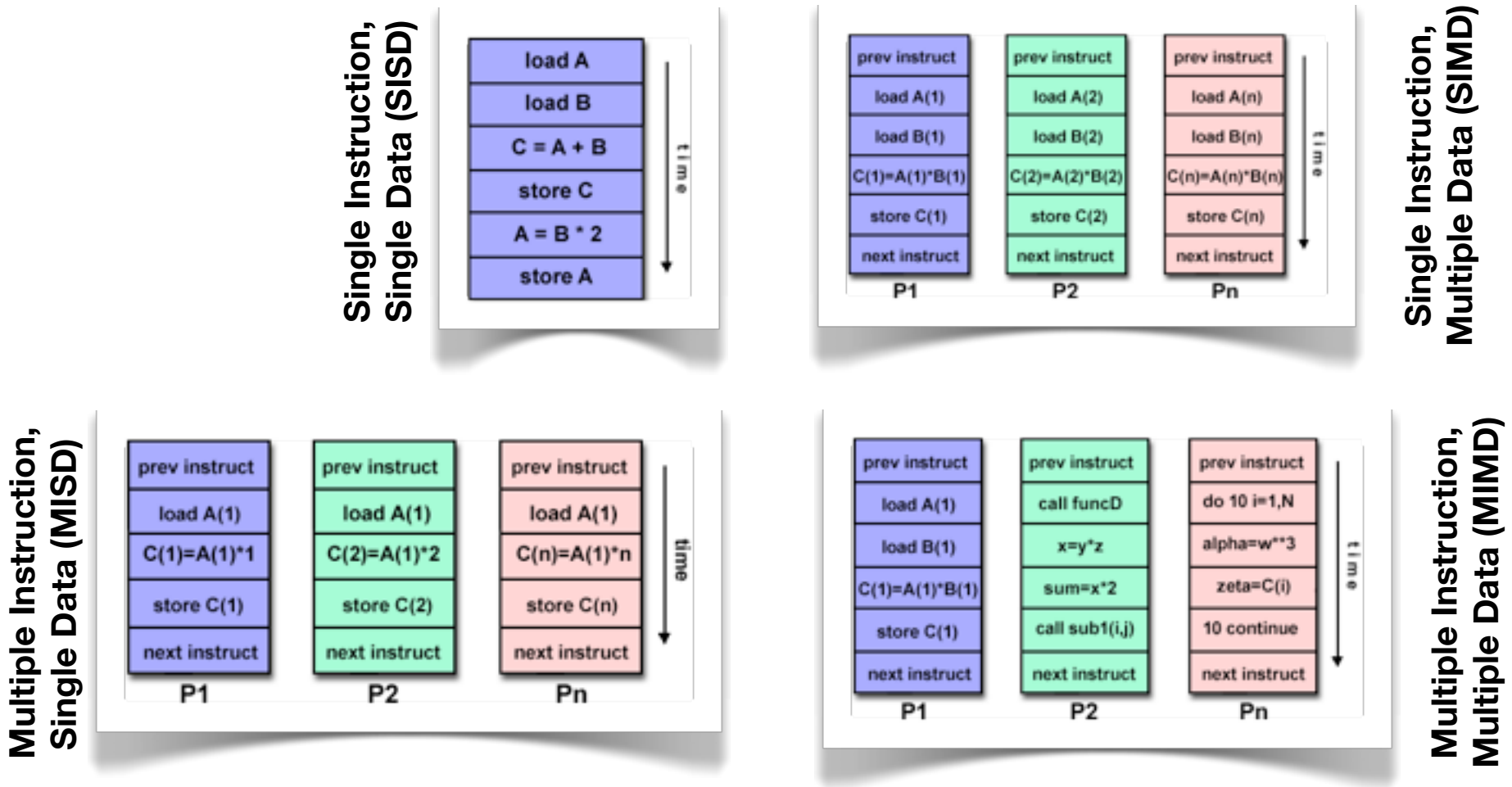
Computer Classification



M.Flynn, *Very High Speed Computing Systems*,
Proceedings of the IEEE, vol 54, 1966, pp. 1901-1909(9)

Multiprocessor: Flynn‘s Taxonomy (1966)

- Classify multiprocessor architectures among **instruction** and data **dimension**



Multiprocessor Systems

- **Symmetric Multiprocessing (SMP)**
  - Set of equal processors in one system (more SM-MIMD than SIMD)
  - Processors share access to main memory over one bus
    - Demands synchronization and operating system support
  - Today, every SMP application also works on a uniprocessor machine
- **Asymmetric multiprocessing (ASMP)**
  - Specialized processors for I/O, interrupt handling or operating system (DEC VAX 11, OS-360, IBM Cell processor)
  - Typically master processor with main memory access and slaves
- Large multiprocessor work with NUMA / COMA memory hierarchy

SMP for Scalability and Availability

- Advantages
  - Performance increase by simple addition of processor card
  - Common shared memory programming model
  - Easy hardware partitioning, in-built redundancy possible
- Disadvantages
  - Scale-up is limited by hardware architecture
  - Complex tuning of the application needed
  - Failover between partitions is solution-dependent
- Solves performance and availability problems rather in hardware & operating system than in software

Classification by granularity

$$\text{Granularity} = \frac{t_{\text{basic communication}}}{t_{\text{basic computation}}}$$

Few powerful processor elements:

- *Coarse grain parallel computers:* Cray Y-MP with 8-16 GFlop-Pes

Many relatively weak processor elements:

- *Fine grain parallel computers*: CM-2 (64k 1-bit-processors), MasPar MP-1 (up to 16344 4-bit PEs), C.mmp, KSR-1

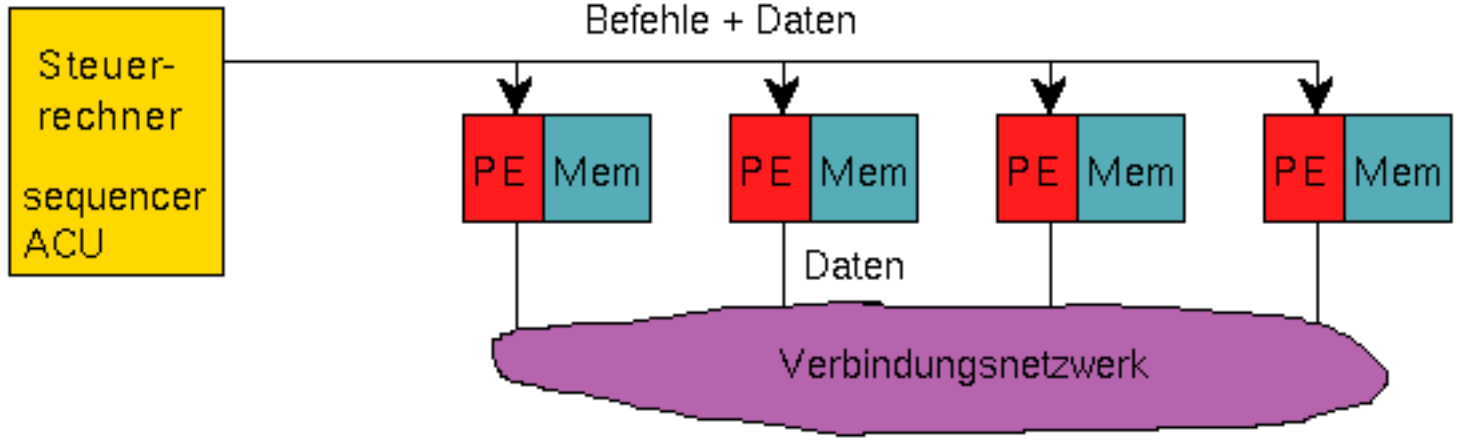
Less than 1000 workstation-class processor elements

- *Medium grain parallel computers*: CM-5, nCUBE2, Paragon XP/S

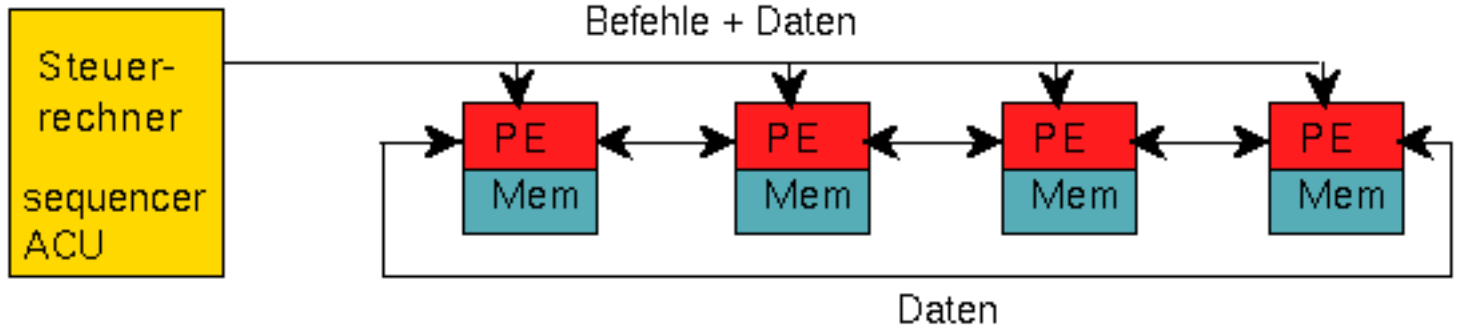
Problem: many algorithms / implementations show limited amount of inherent parallelism

SIMD Computers

Arrayrechner

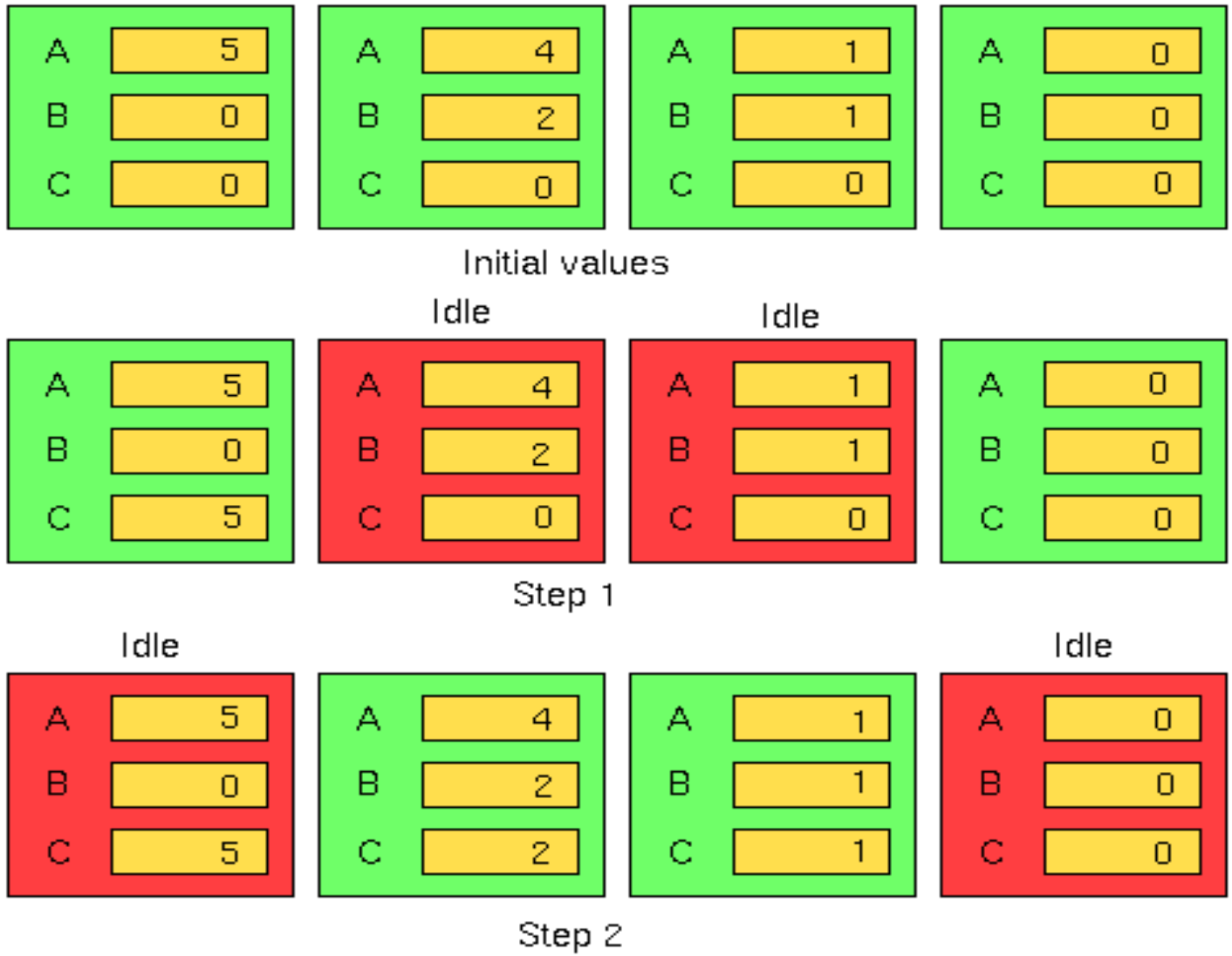


Vektorrechner



SIMD Problems

```
if (B == 0)
        C = A;
else
        C = A/B;
```



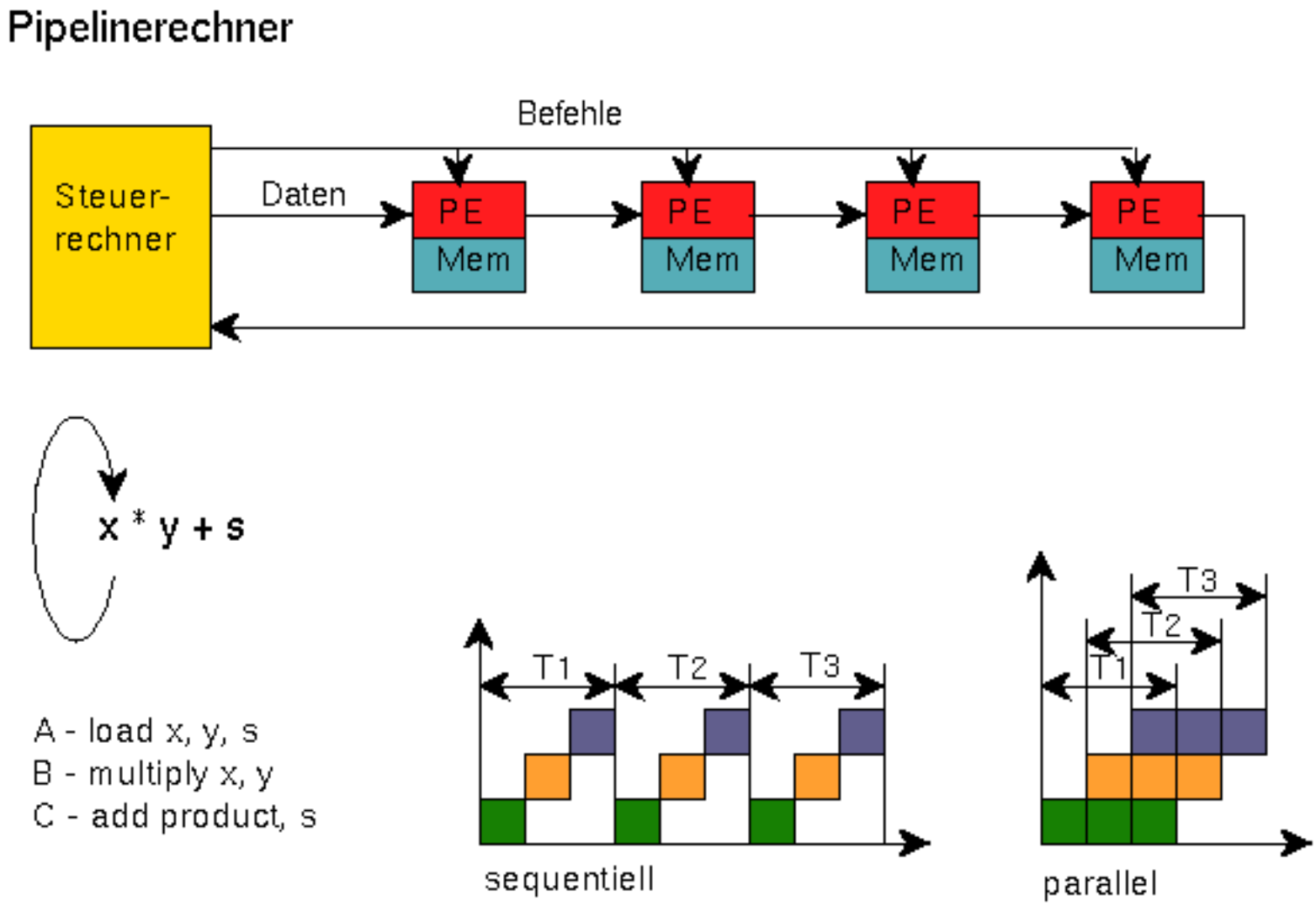
SIMD Vector Pipelines

- Vector processors have high-level operations for data sets
- Became famous with Cray architecture in the 70‘s
- Today, vector instructions are part of the standard instruction set
  - AltiVec
  - Streaming SIMD Extensions (SSE)
    - Example: Vector addition

```
vec_res.x = v1.x + v2.x;
vec_res.y = v1.y + v2.y;
vec_res.z = v1.z + v2.z;
vec_res.w = v1.w + v2.w;
```

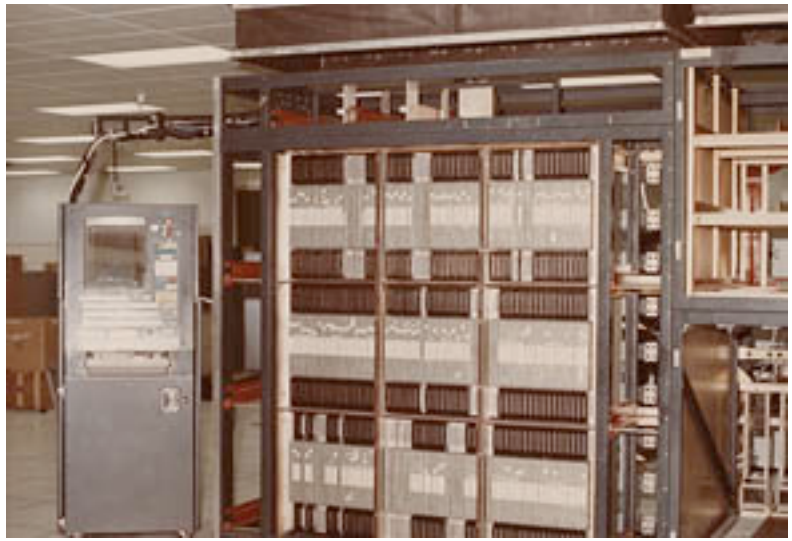
```
movaps xmm0,address-of-v1
(xmm0=v1.w | v1.z | v1.y | v1.x)

addps xmm0,address-of-v2
(xmm0=v1.w+v2.w | v1.z+v2.z | v1.y+v2.y | v1.x+v2.x)

movaps address-of-vec res,xmm0
```

SIMD Pipelining



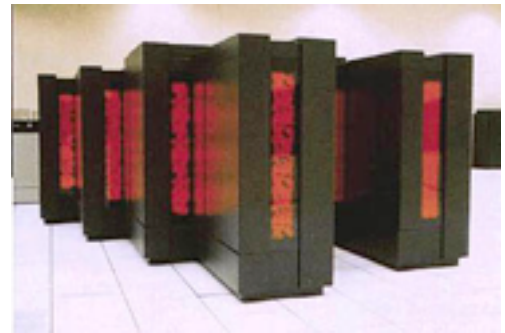
SIMD Examples

ILLIAC IV (1974)

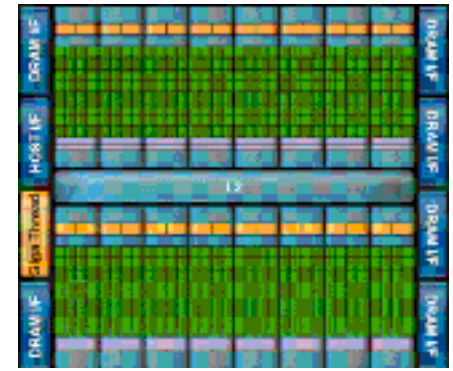


- Good for problems with high degree of regularity, such as graphics/image processing
- Synchronous (lockstep) and deterministic execution
- Typically exploit data parallelism
- Today: GPGPU Computing, Cell processor, SSE, AltiVec

Cray Y-MP



Thinking Machines CM-2 (1985)



Fermi GPU

Illiac IV

- Supercomputer for vector processing from University of Illinois (1966)
- One control unit fetches instructions
  - Handed over to a set of processing elements (PE‘s)
  - Each PE has own memory, accessible by control unit
- Intended for 1 GFLOPS, ended up with 100 MFLOPS at the end
- Main work on bringing the data to the SIMD machine
  - Parallelized versions of FORTRAN language
- Credited as fastest machine until 1981
  - Computational fluid dynamics (NASA)

(C) Wikipedia

CM2 – Connection Machine

W. Daniel Hillis: The Connection Machine.
1985 (MIT Press Series in Artificial Intelligence)
ISBN 0-262-08157-1

CM2 at Computer Museum, Mountain View, CA

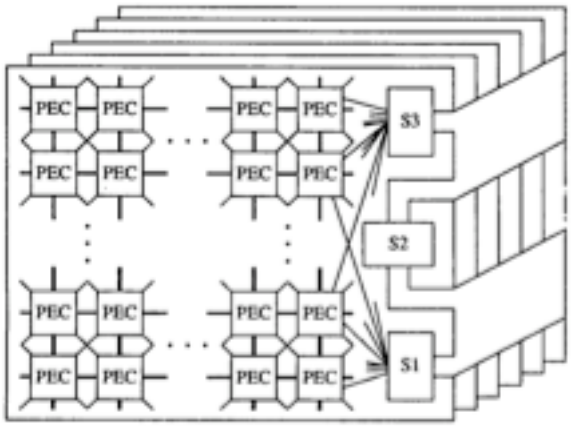
Hersteller:	**Thinking Machines Corporation, Cambridge, Massachusetts**
Prozessoren:	**65.536 PEs (1-Bit Prozessoren)** **Speicher je PE: 128 KB (maximal)** **Peak-Performance: 2.500 MIPS (32-Bit Op.)** **10.000 MFLOPS (Skalar,32Bit)** **5.000 MFLOPS (Skalar,64Bit)**
Verbindungsnetzwerke:	**-globaler Hypercube** **- 4-faches, rekonfigurierbares Nachbarschaftsgitter**
Programmiersprachen:	**-CMLisp (ursprüngliche Variante)** **- *Lisp (Common Lisp Erweiterung)** **-C*(Erweiterung von C)** **-CMFortran (Anlehnung an Fortran 90)** **-C/Paris (C+Assembler Bibliotheksroutinen)**

MasPar MP-1

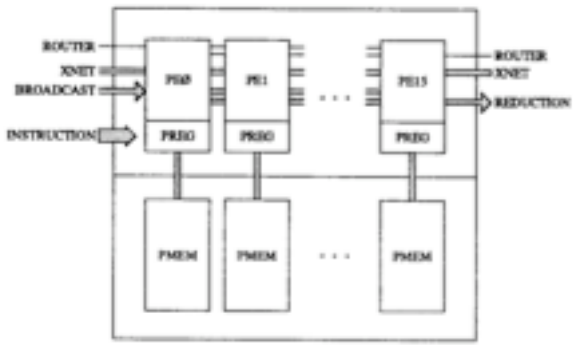
Hersteller:	**MasPar Computer Corporation, Sunnyvale, California**
Prozessoren:	**16.384 PEs (4-Bit Prozessoren) Spei-cher je PE: 64 KB (maximal) Peak-Performance: 30.000 MIPS (32-Bit Op.) 1.500 MFLOPS (32-Bit) 600 MFLOPS (64-Bit)**
Verbindungsnetzwerke:	**3-stufiger globaler crossbar switch (Router) 8-faches Nachbarschaftsgitter (unabh.)**
Programmiersprachen	**- MPL (Erweiterung von C) - MPFortran (Anlehnung an Fortran 90**

MasPar MP-1 Architecture

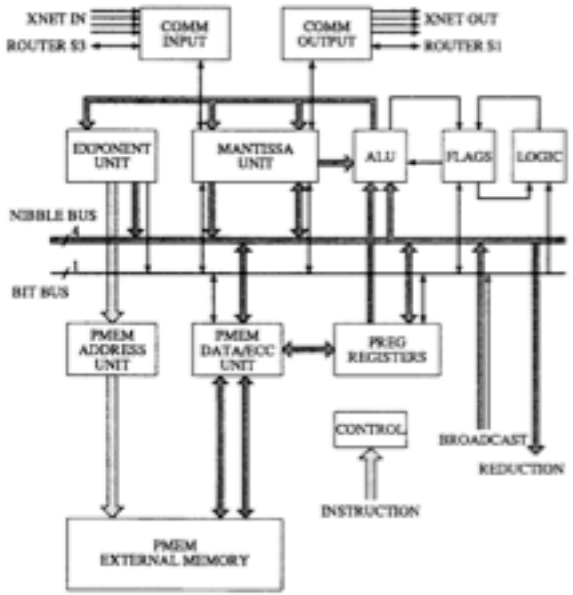
- Processor Chip contains 32 identical PEs
- PE is mostly data path logic, no instruction fetch/decode



Interconnection structure



Processor element



Inside a PE

Nickolls, J.R.; MasPar Comput. Corp., Sunnyvale, CA
The design of the MasPar MP-1: a cost effective massively parallel computer
Compcon Spring '90. Intellectual Leverage. Digest of Papers. Thirty-Fifth IEEE Comp. Soc. Intl. Conf..

Distributed Array Processor (DAP 610)

The Distributed Array Processor (DAP) produced by International Computers Limited (ICL) was the world's first commercial massively parallel computer. The original paper study was complete in 1972 and building of the prototype began in 1974.

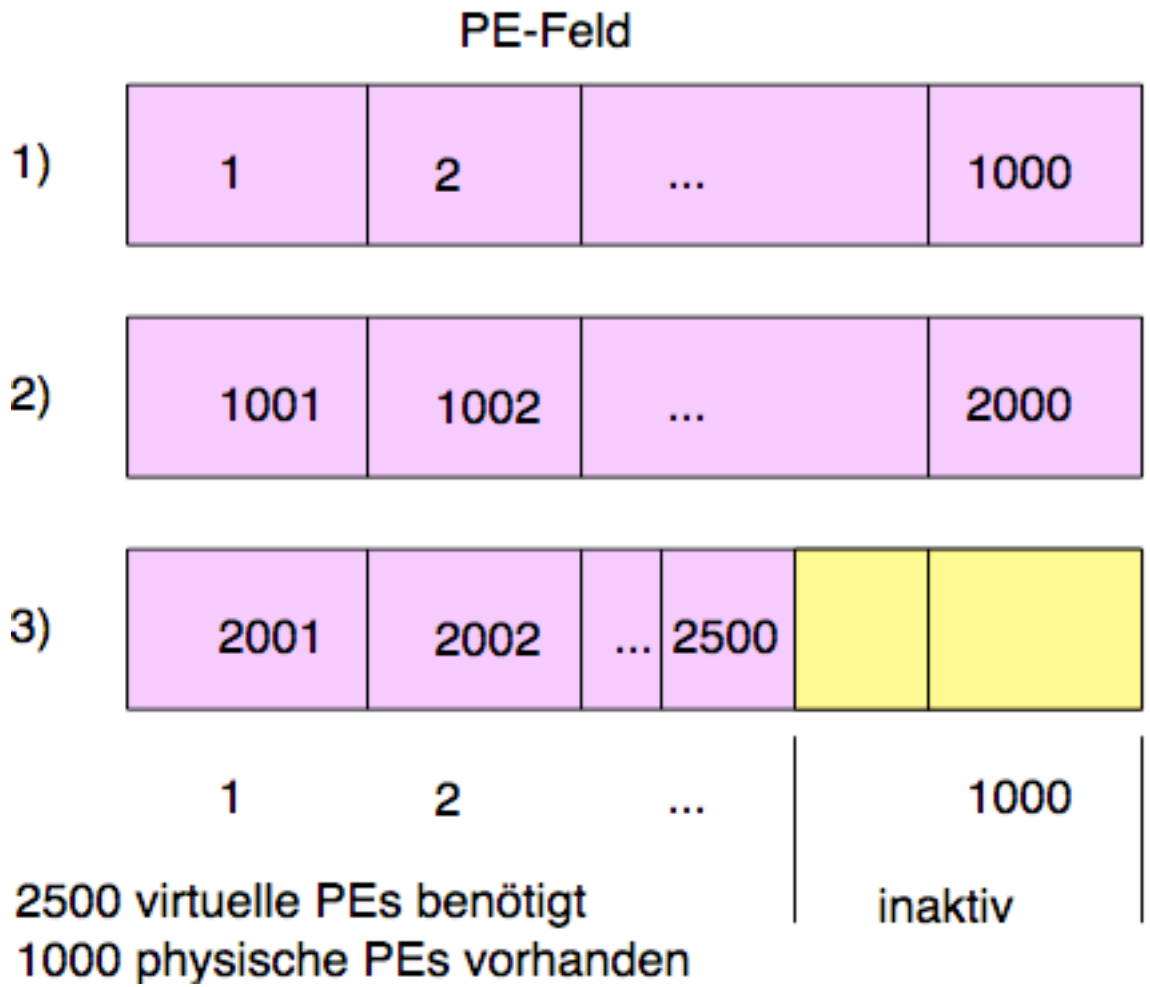
The ICL DAP had 64x64 single bit processing elements (PEs) with 4096 bits of storage per PE. It was attached to an ICL mainframe and could be used as normal memory. (from Wikipedia).

Early mainframe coprocessor...

Hersteller:	**Active Memory Technology (AMT), Reading, England**
Prozessoren:	**4.096 PEs (1-Bit Prozessoren + 8-Bit Koprozessoren)** **Speicher je PE: 32 KB** **Peak-Performance:** **40.000 MIPS (1-Bit Op.)** **20.000 MIPS (8-Bit Op.)** **560 MFLOPS**
Verbindungsnetzwerk:	- **4-faches Nachbarschaftsgitter** - **(kein globales Netzwerk)**
Programmiersprache:	**- Fortran-Plus (in Anlehnung an Fortran 90)**

Problems with synchronous parallelism: virtual processor elements

- Even thousands of PEs may not be sufficient…



SIMD communication – programming is complex

- Activation of a group of PEs
- Selection of a previously defined connection network
- Pair-wise data exchange among active PEs

```
PARALLEL ring[3..8]
    PROPAGATE.rechts(x)
ENDPARALLEL
```

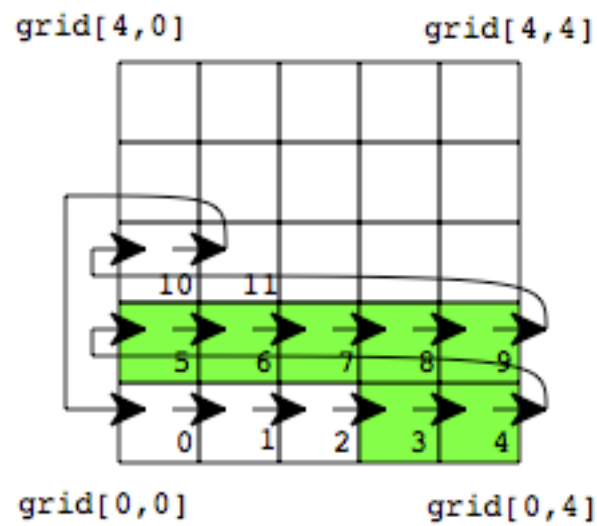
wird abgebildet auf:

a) ein Schritt nach rechts

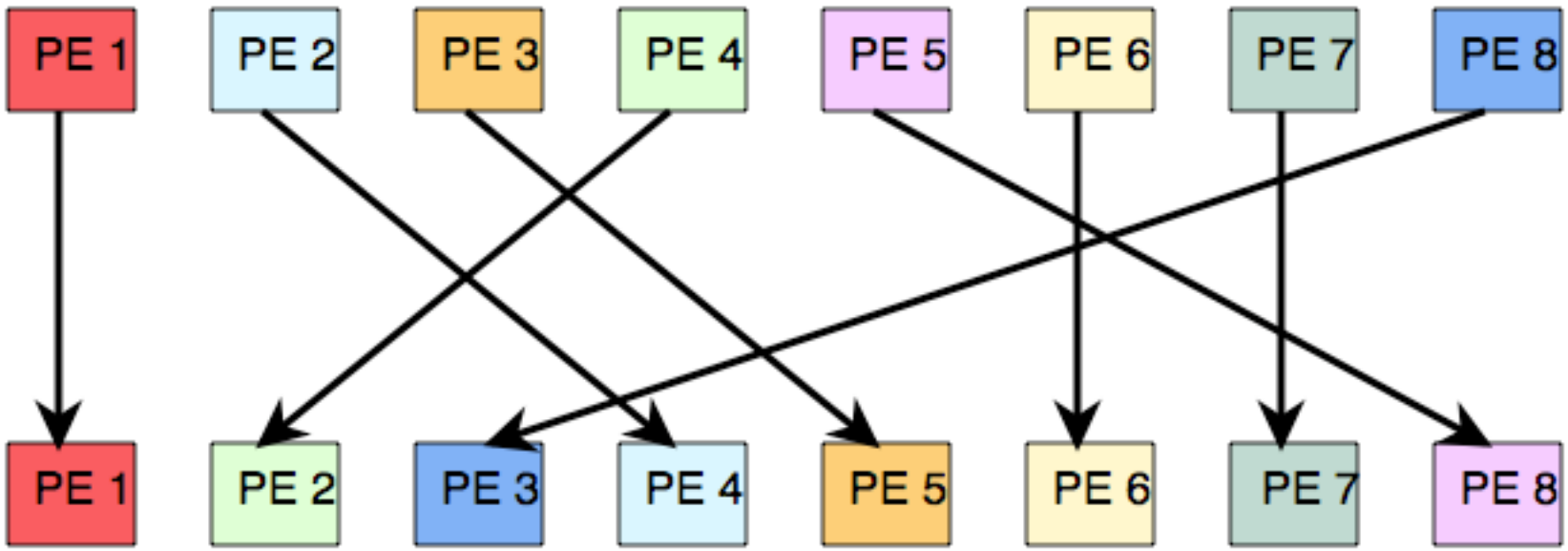
```
PARALLEL
    grid[0..1],[3..4];
    grif[1],   [0..3]
  "grid[i,j] -> grid[i,j+1]"
```

b) eine Zeile höher

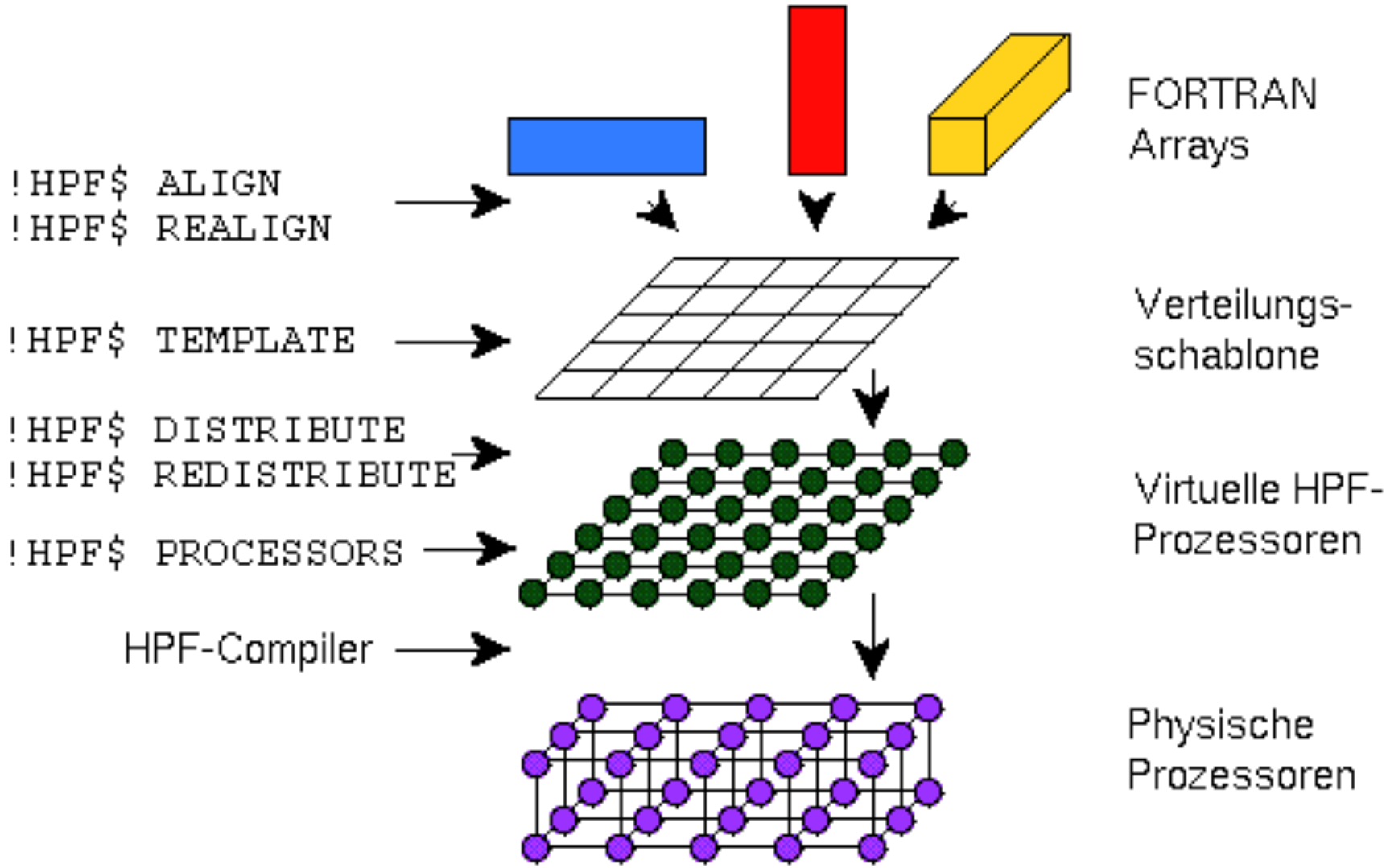
```
PARALLEL
    grid[0],[4]
  "grid[i,j] -> grid[i+1,0]"
```



Permutations – arbitrary data exchange



High Performance Fortran

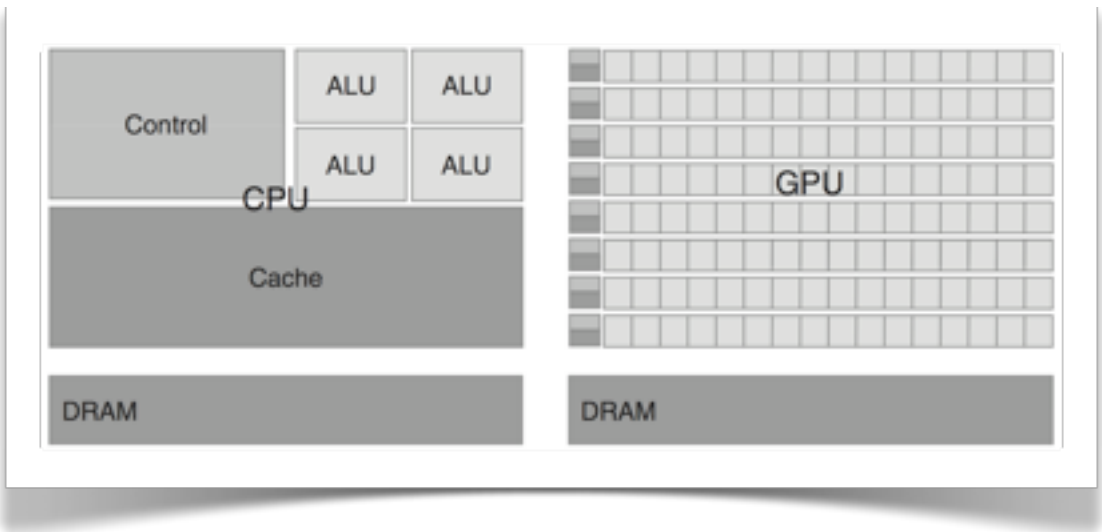


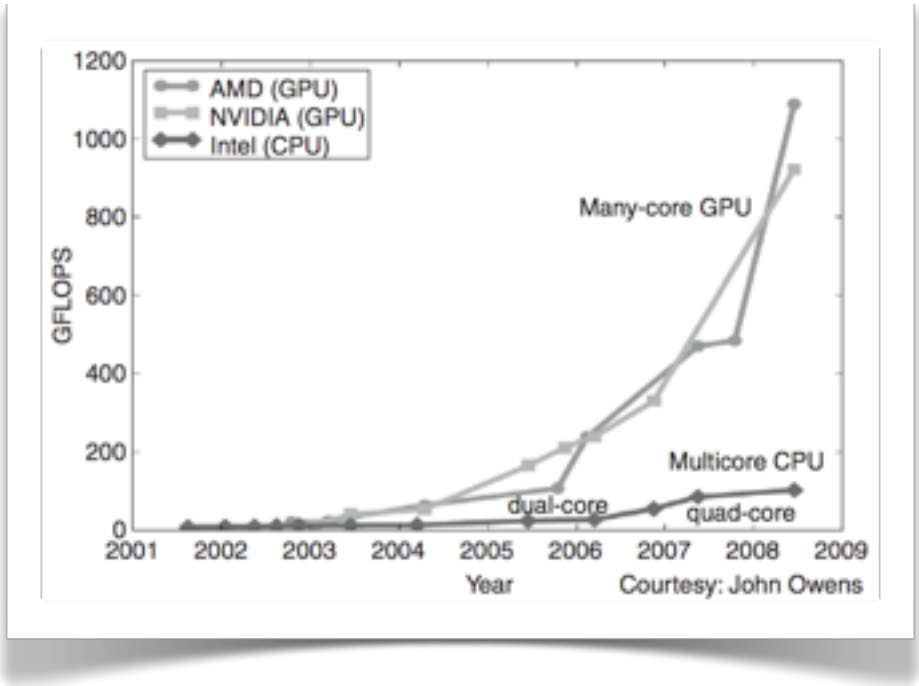
Data distribution in HPF

```
!HPF$ PROCESSORS :: prc(5), chess_board(8, 8)

!HPF$ PROCESSORS :: cnfg(-10:10, 5)

!HPF$ PROCESSORS :: mach( NUMBER_OF_PROCESSORS() )

  REAL :: a(1000), b(1000)

  INTEGER :: c(1000, 1000, 1000), d( 1000, 1000, 1000)

!HPF$ DISTRIBUTE (BLOCK) ONTO prc :: a

!HPF$ DISTRIBUTE (CYCLIC) ONTO prc :: b

!HPF$ DISTRIBUTE (BLOCK(100), *, CYCLIC) ONTO cnfg :: c

!HPF$ ALIGN (i,j,k) WITH d(k,j,i) :: c
```

GPGPU Computing – SIMD + multithreading

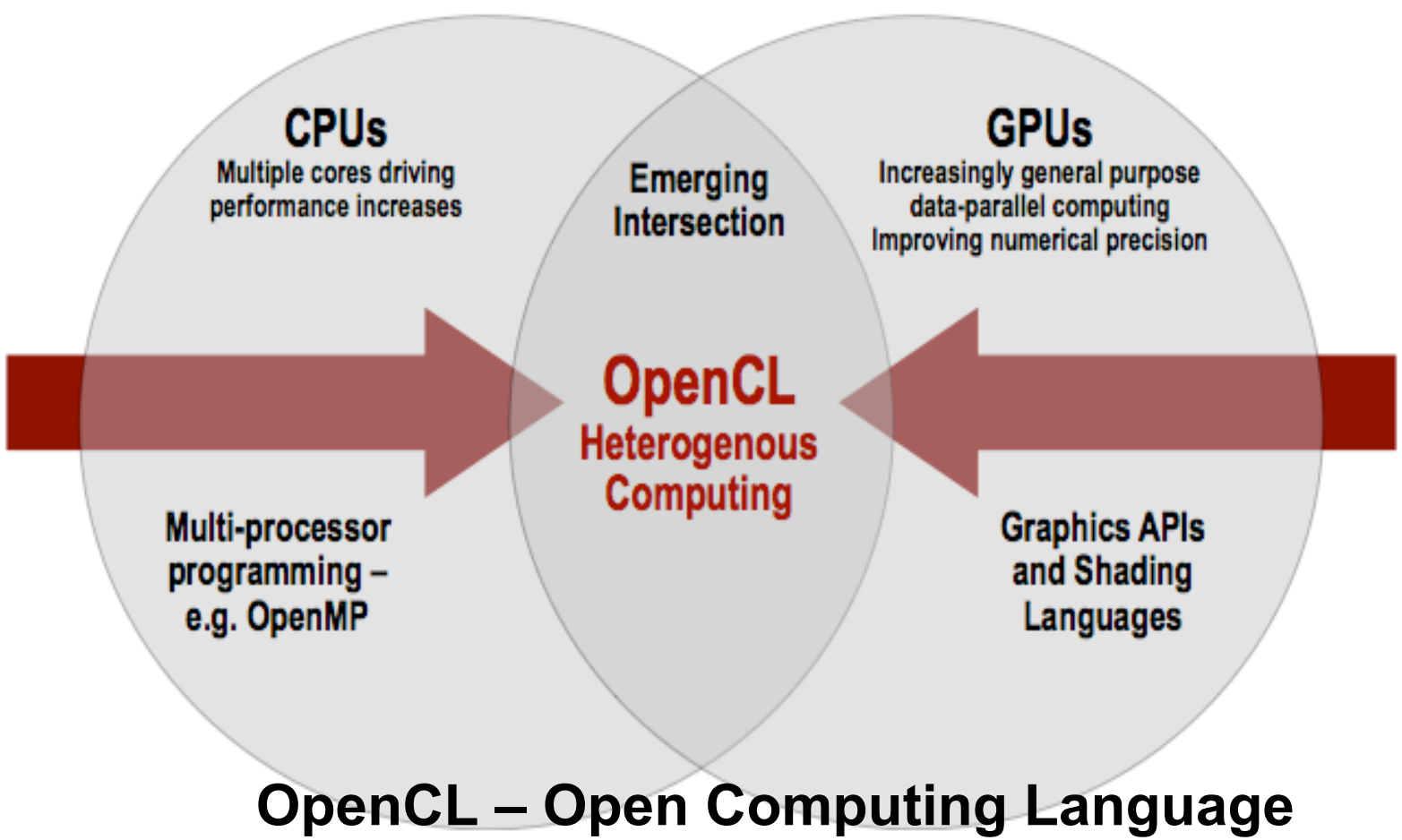
- Pure SIMD approach, different design philosophy
- Driven by video / game industry development, recent move towards general purpose computations
- Offloading parallel computation to the GPU is still novel





(C) Kirk & Hwu

Programming Models #1: OpenCL, CUDA



OpenCL – Open Computing Language

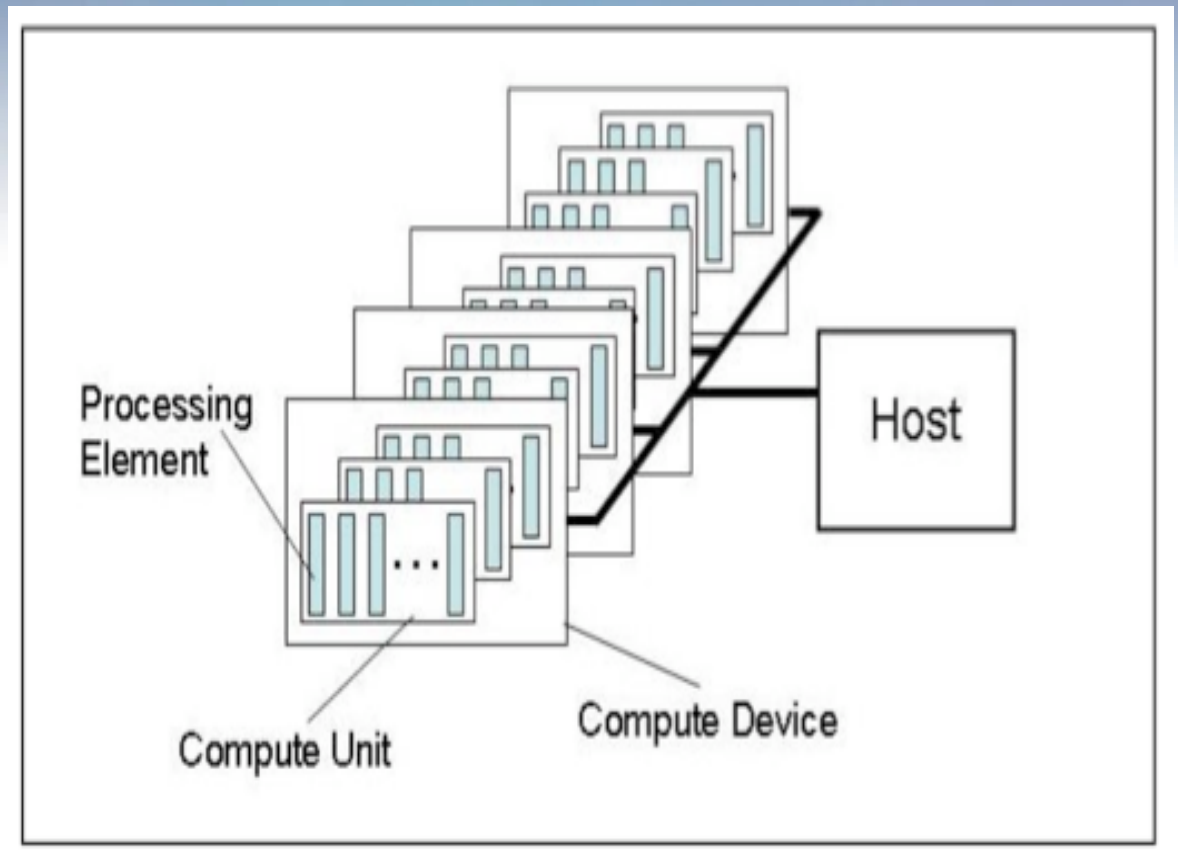
CUDA – Compute Unified Device Architecture

Open standard for portable, parallel programming of heterogeneous parallel computing CPUs, GPUs, and other processors

OpenCL Design Goals

- Use all computational resources in system
  - Program GPUs, CPUs, and other processors as peers
  - Support both data- and task- parallel compute models
- Efficient C-based parallel programming model
  - Abstract the specifics of underlying hardware
- Abstraction is low-level, high-performance but device-portable
  - Approachable – but primarily targeted at expert developers
  - Ecosystem foundation – no middleware or “convenience” functions
- Implementable on a range of embedded, desktop, and server systems
  - HPC, desktop, and handheld profiles in one specification
- Drive future hardware requirements
  - Floating point precision requirements
  - Applicable to both consumer and HPC applications

OpenCL Platform Model



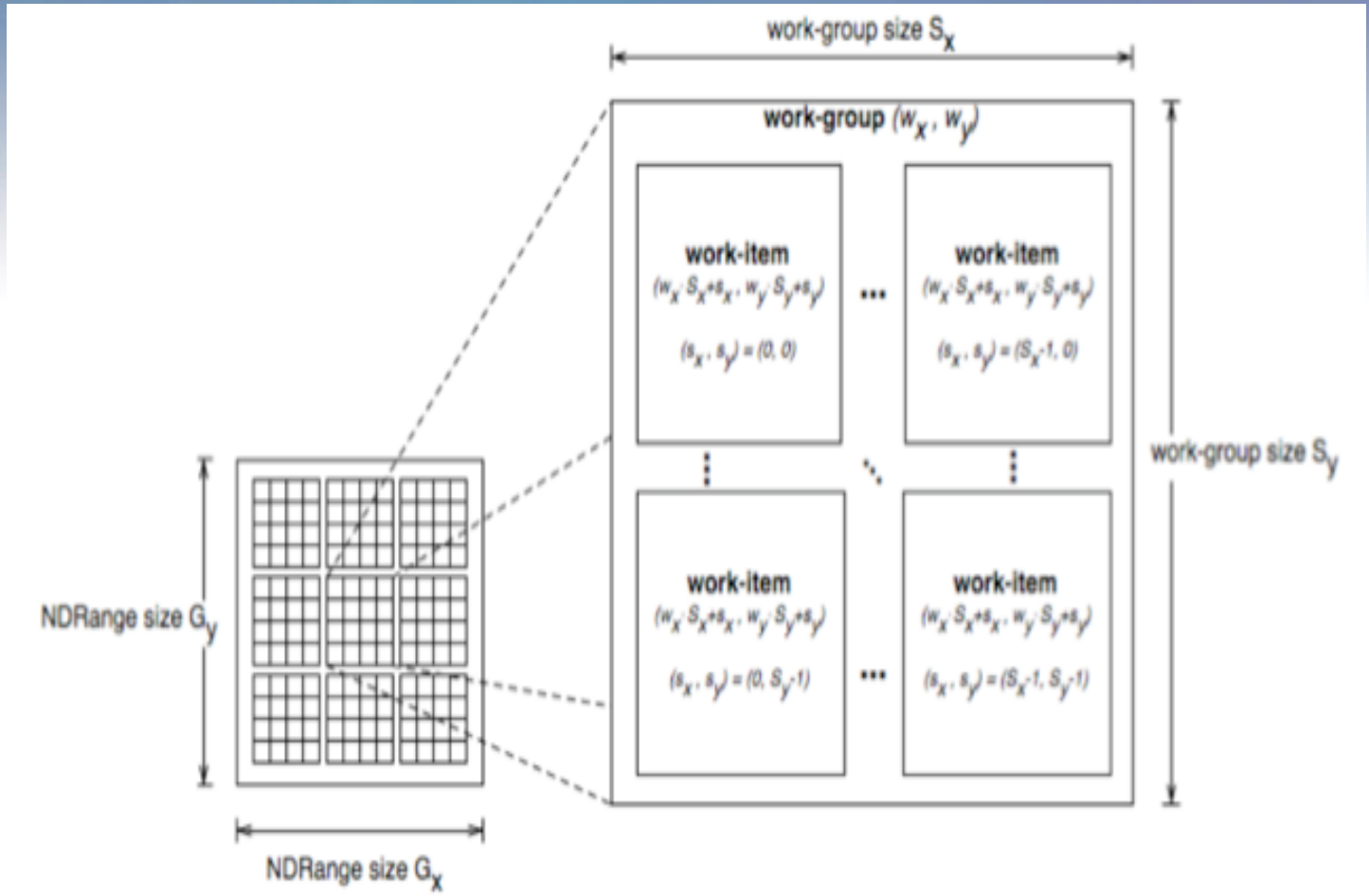
- One Host + one or more Compute Devices
  - Each Compute Device is composed of one or more Compute Units
  - Each Compute Unit is further divided into one or more Processing Elements

OpenCL Execution Model

- OpenCL Program:
  - Kernels
    - Basic unit of executable code — similar to a C function
    - Data-parallel or task-parallel
  - Host Program
    - Collection of compute kernels and internal functions
    - Analogous to a dynamic library

- Kernel Execution
  - The host program invokes a kernel over an index space called an NDRange
    - NDRange = “N-Dimensional Range”
    - NDRange can be a 1, 2, or 3-dimensional space
  - A single kernel instance at a point in the index space is called a work-item
    - Work-items have unique global IDs from the index space
  - Work-items are further grouped into work-groups
    - Work-groups have a unique work-group ID
    - Work-items have a unique local ID within a work-group

Kernel Execution



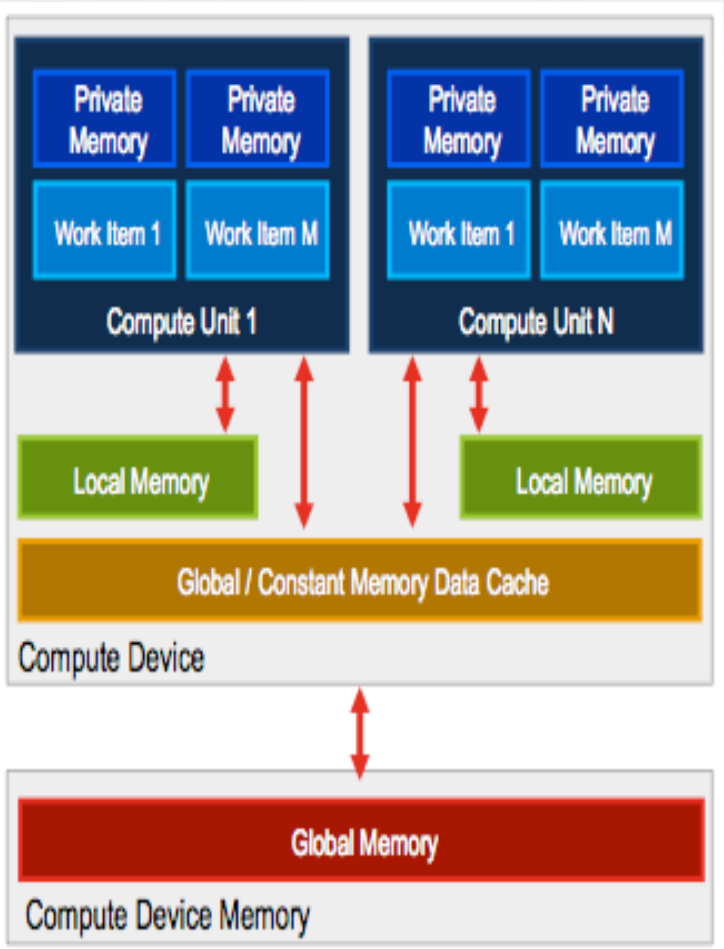
- Total number of work-items = G_x x G_y
- Size of each work-group = S_x x S_y
- Global ID can be computed from work-group ID and local ID

Contexts and Queues

- Contexts are used to contain and manage the state of the “world”
- Kernels are executed in contexts defined and manipulated by the host
  - Devices
  - Kernels - OpenCL functions
  - Program objects - kernel source and executable
  - Memory objects
- Command-queue - coordinates execution of kernels
  - Kernel execution commands
  - Memory commands - transfer or mapping of memory object data
  - Synchronization commands - constrains the order of commands
- Applications queue compute kernel execution instances
  - Queued in-order
  - Executed in-order or out-of-order
  - Events are used to implement appropriate synchronization of execution instances

OpenCL Memory Model

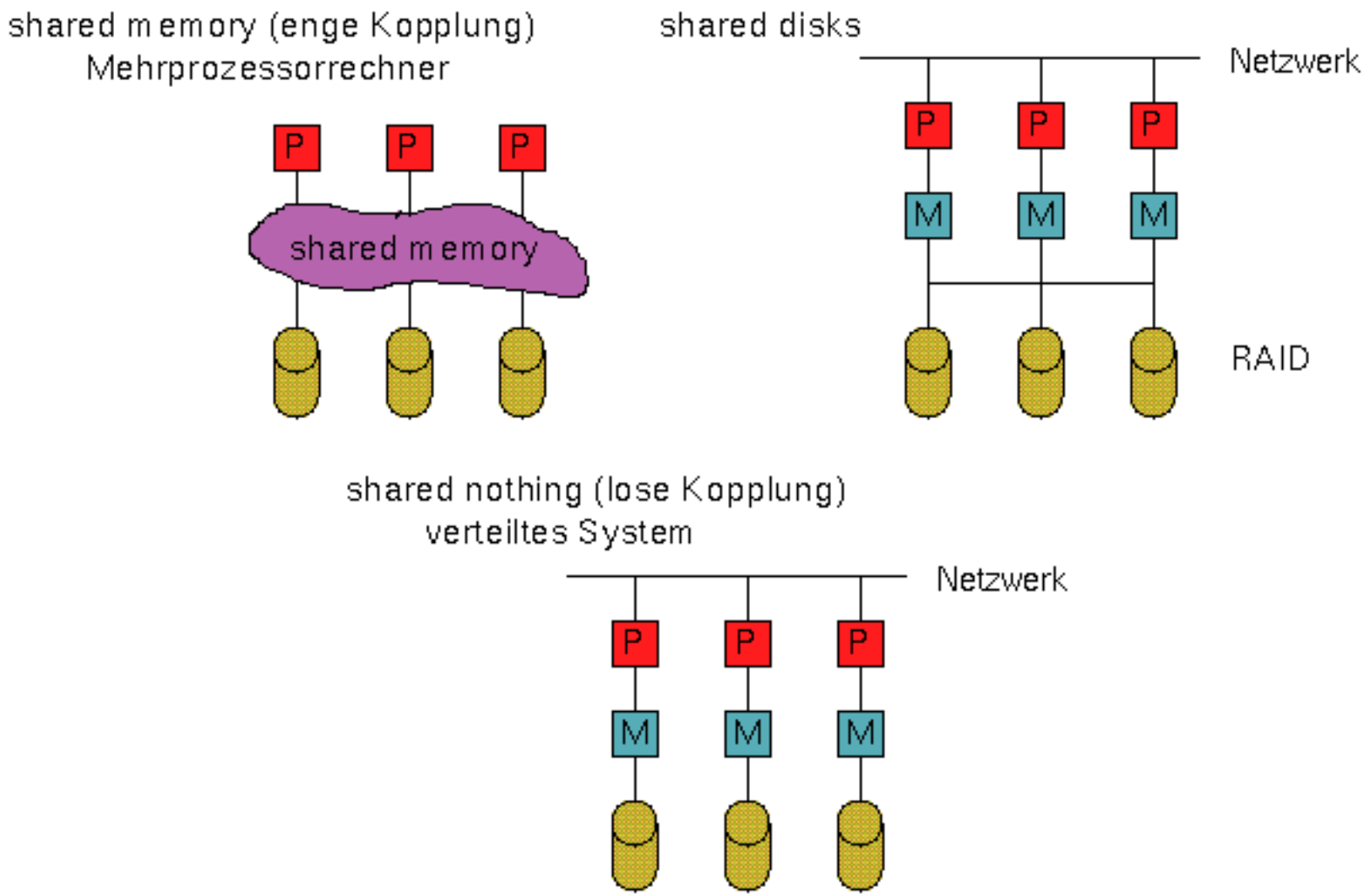
- Shared memory model
  - Relaxed consistency
- Multiple distinct address spaces
  - Address spaces can be collapsed depending on the device's memory subsystem
- Address spaces
  - Private - private to a work-item
  - Local - local to a work-group
  - Global - accessible by all work-items in all work-groups
  - Constant - read only global space
- Implementations map this hierarchy
  - To available physical memories



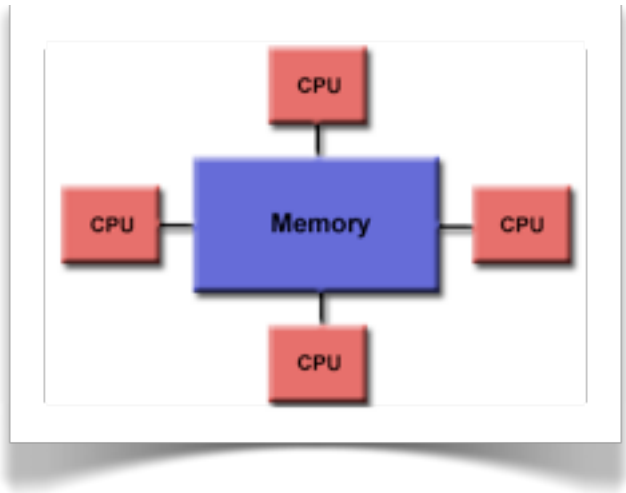
Multiple Instruction Multiple Data (MIMD)

- Most common parallel hardware architecture today
  - Example: All many-core processors, clusters, distributed systems
- From software perspective [Pfister]
  - **SPMD - Single Program Multiple Data**
    - Sometimes denoted as ‚application cluster‘
    - Examples: Load-balancing cluster or failover cluster for databases, web servers, application servers, ...
  - **MPMD - Multiple Program Multiple Data**
    - Multiple implementations work together on one parallel computation
    - Example: Master / worker cluster, map / reduce framework

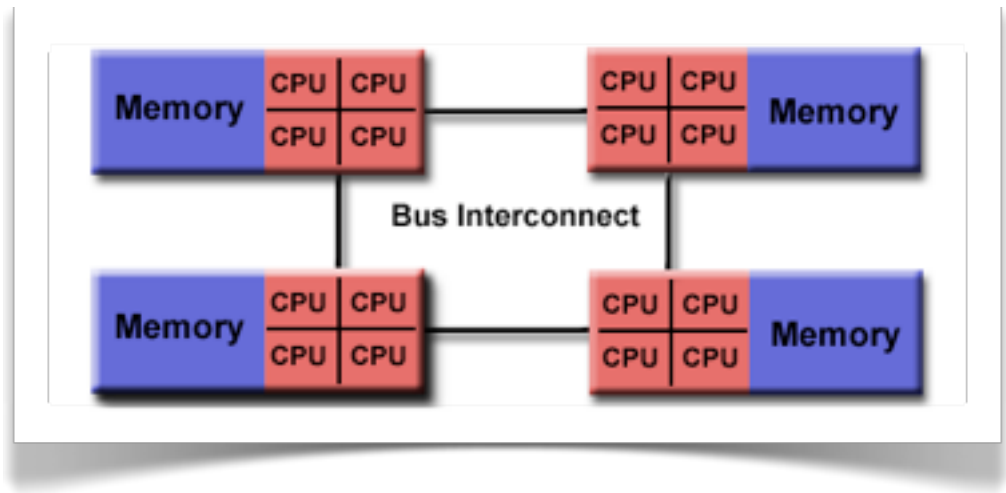
MIMD Classification



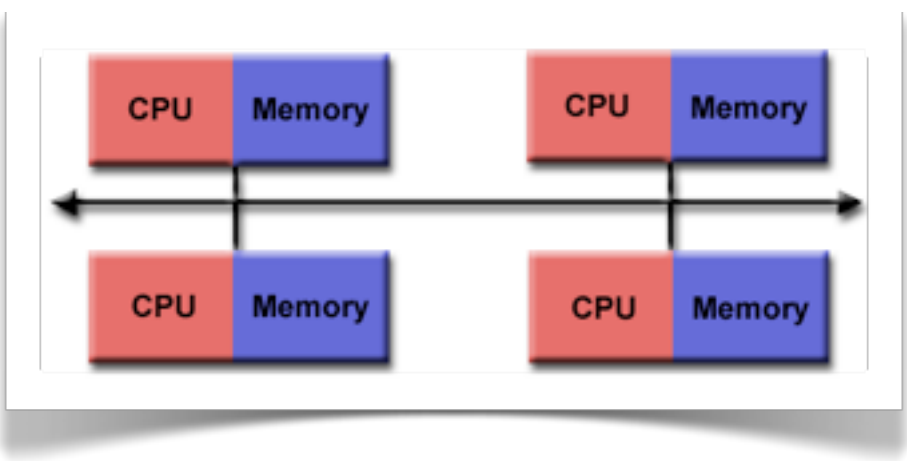
Memory Architectures



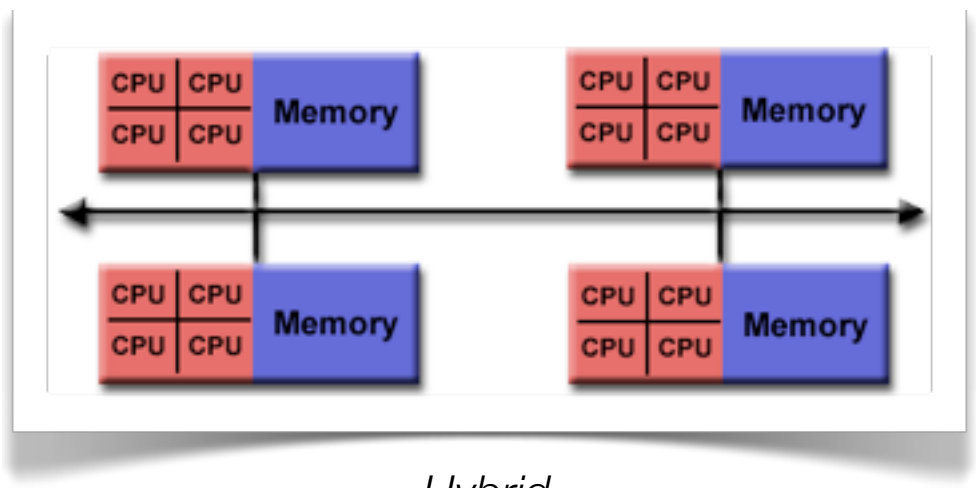
Uniform Memory Access (UMA)



Non-Uniform Memory Access (NUMA)



Distributed Memory



Hybrid

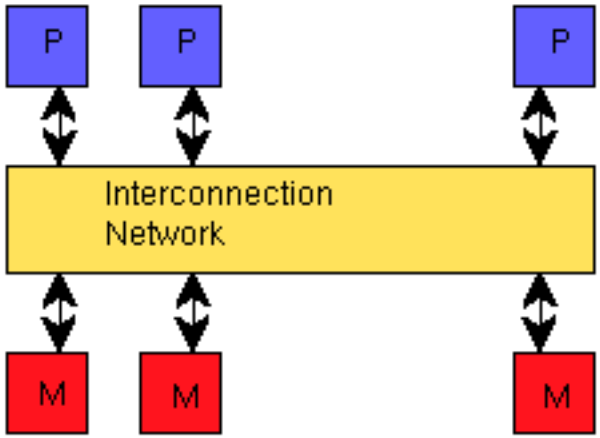
Shared Memory vs. Distributed Memory System

- **Shared memory (SM) systems**
  - SM-SIMD: Single CPU vector processors
  - SM-MIMD: Multi-CPU vector processors, OpenMP
  - Variant: Clustered shared-memory systems (NEC SX-6, CraySV1ex)
- **Distributed memory (DM) systems**
  - DM-SIMD: processor-array machines; lock-step approach; front processor and control processor
  - DM-MIMD: large variety in interconnection networks
- **Distributed (Virtual) shared-memory systems**
  - High-Performance Fortran, TreadMarks

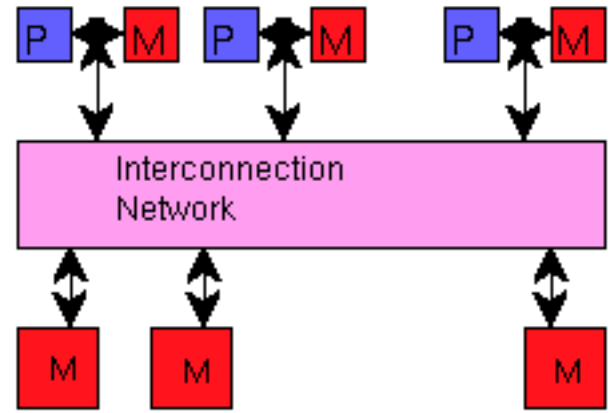
Shared Memory Architectures

- All processors act independently, access the same global address space
- Changes in one memory location are visible for all others
- **Uniform memory access (UMA) system**
  - Equal load and store access for all processors to all memory
  - Default approach for majority of SMP systems in the past
- **Non-uniform memory access (NUMA) system**
  - Delay on memory access according to the accessed region
  - Typically realized by processor interconnection network and local memories
  - Cache-coherent NUMA (CC-NUMA), completely implemented in hardware
  - About to become standard approach with recent X86 chips

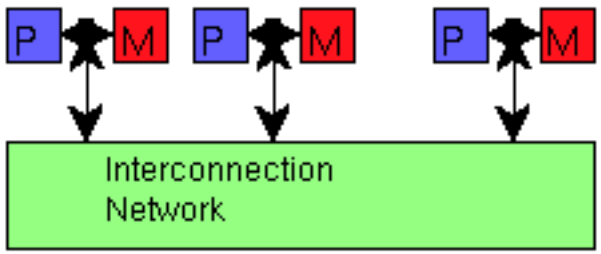
NUMA Classification



Uniform-memory-access
shared-address-space
computer (UMA)



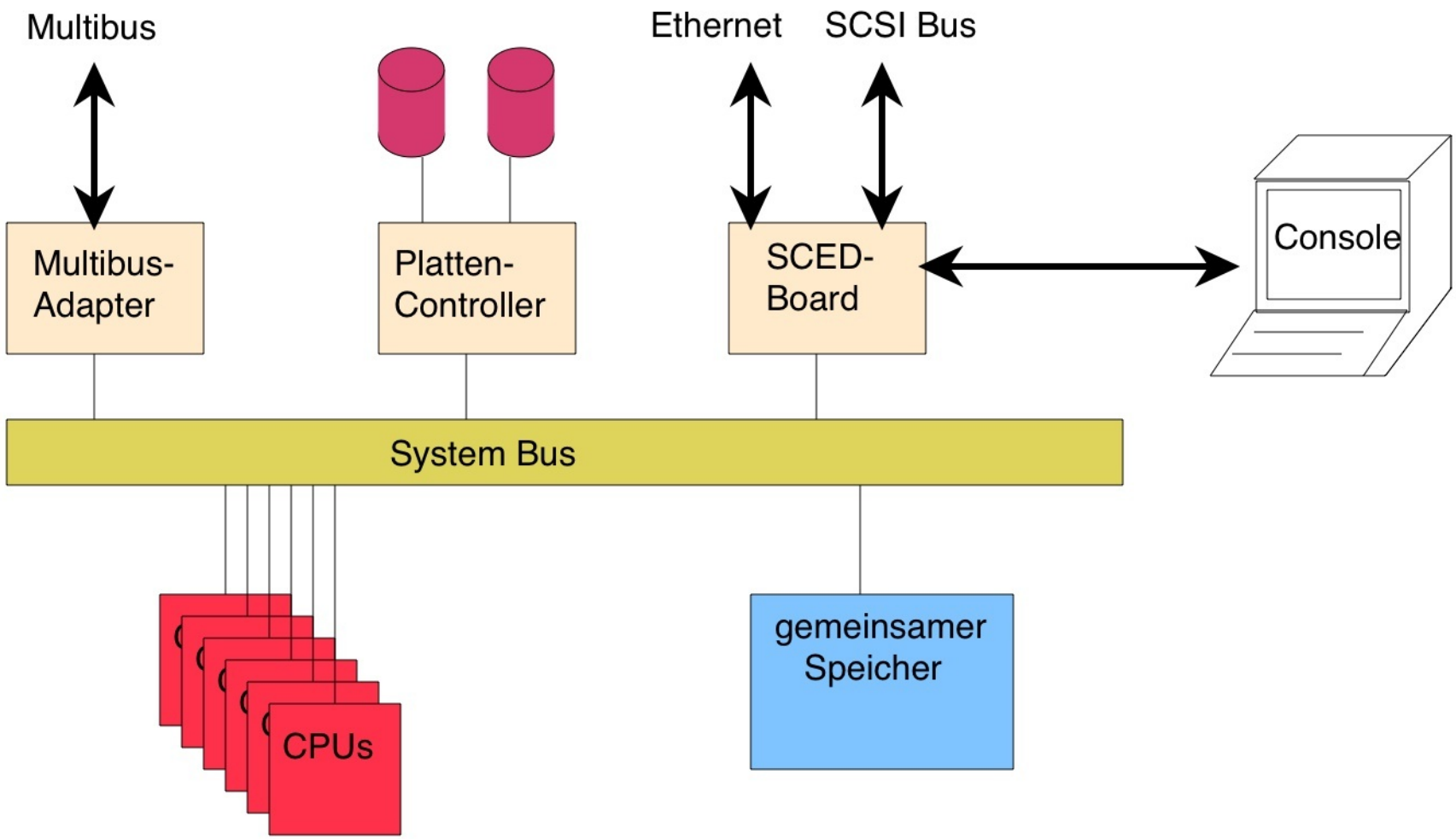
Non-Uniform-memory-access
shared-address-space
computer with local and
global memories (NUMA)



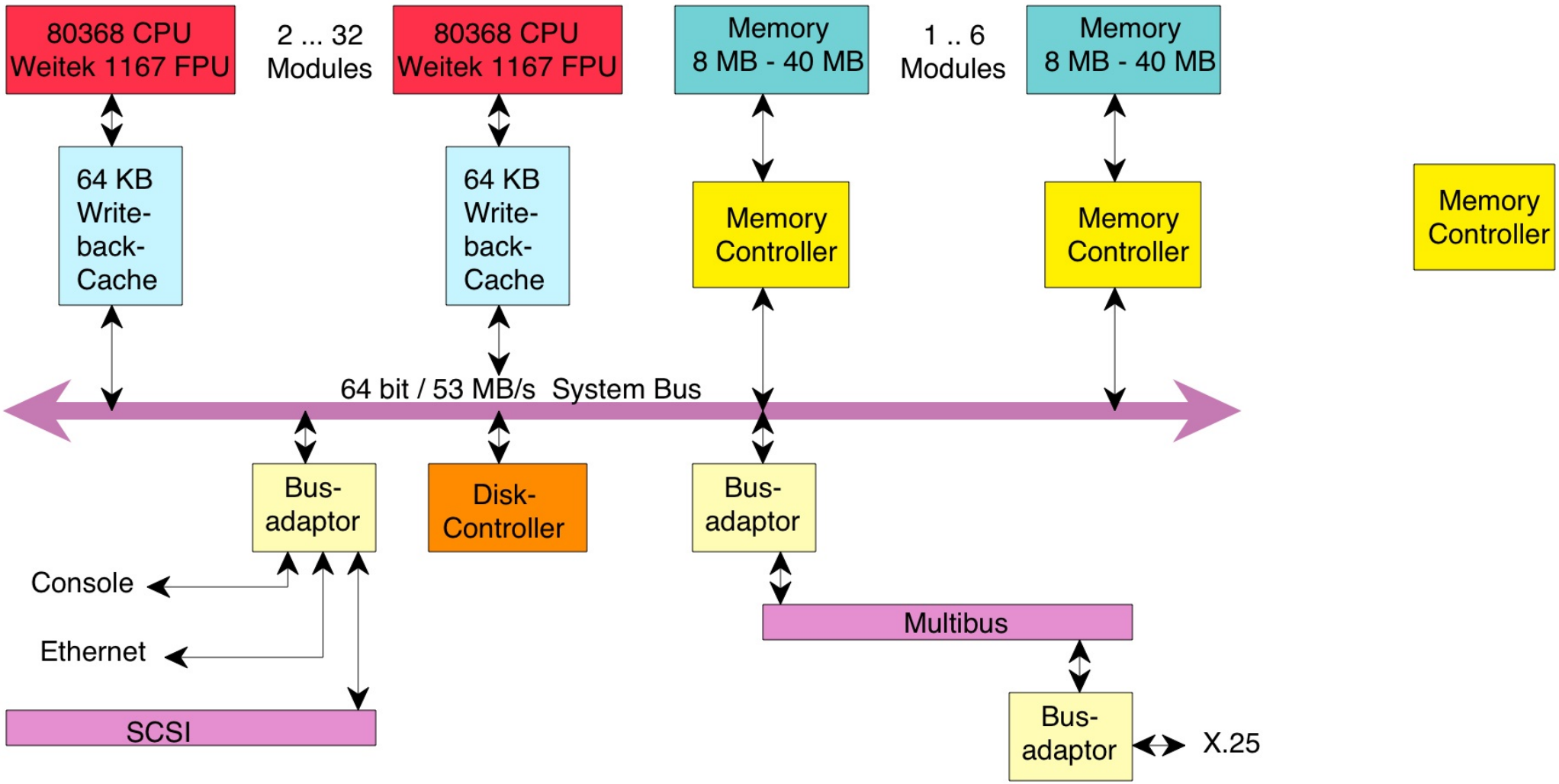
Non-uniform-memory-access
shared-address-space
computer with
local memory only (NUMA)

MIMD Computer Systems

- Sequent Balance



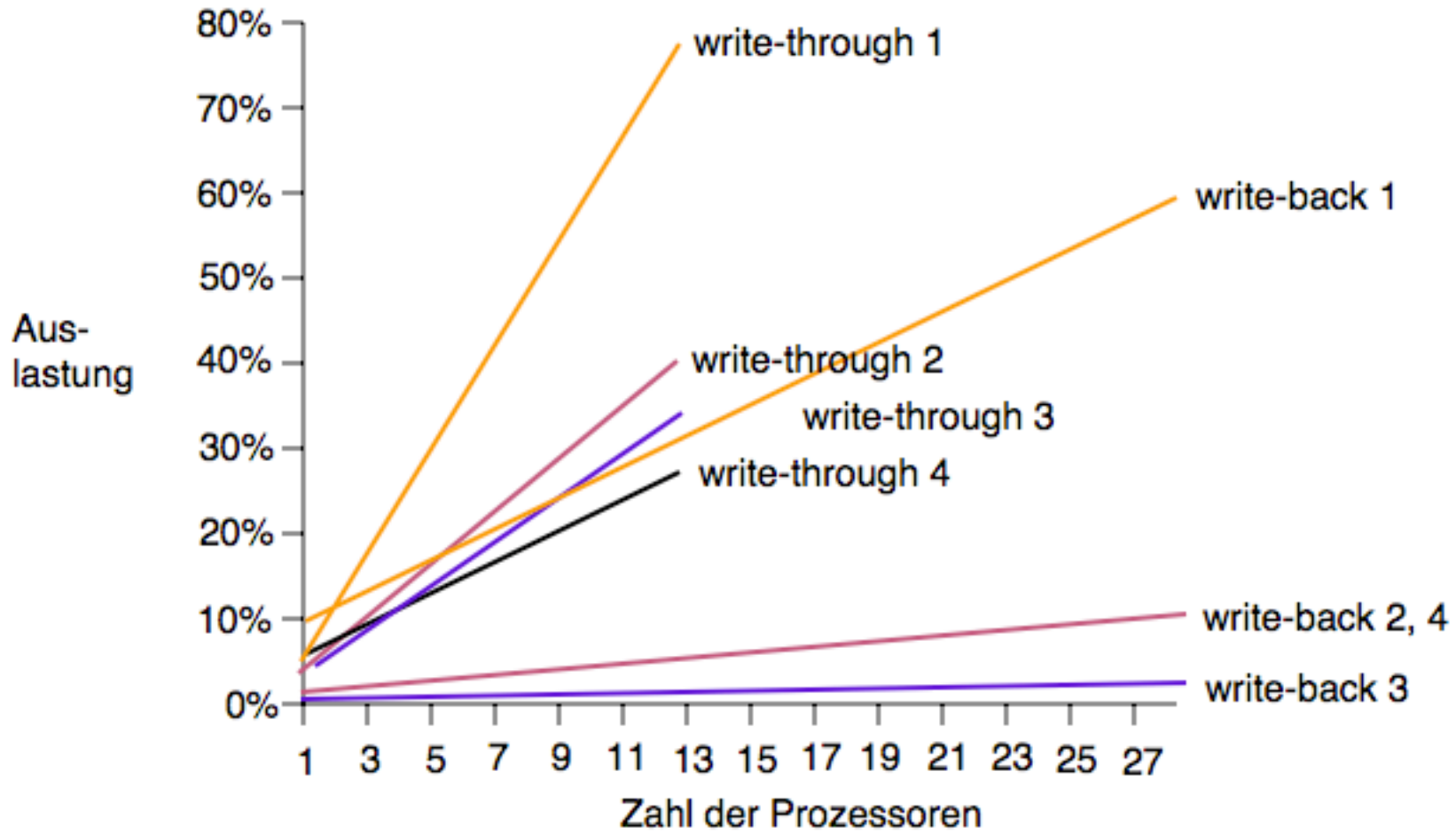
Sequent Symmetry



Sequent was bought by IBM in 1999. IBM produced several Intel-based servers based on Sequent's later NUMA architecture…

Caches – managing bus contention

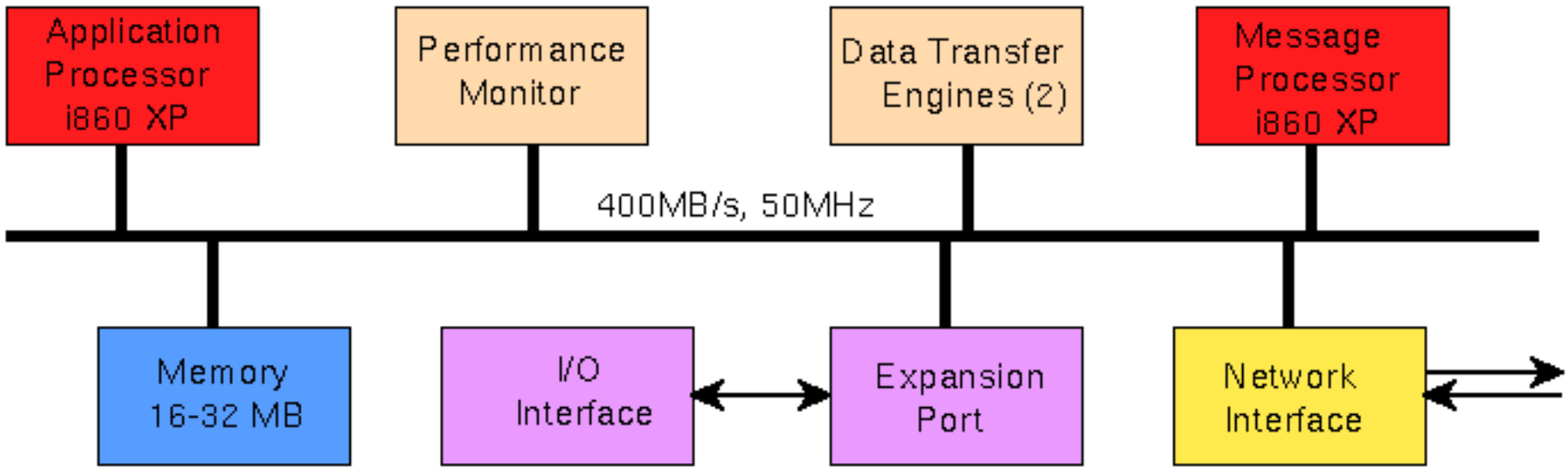
- Effect of write-through and write-back cache coherency protocols on Sequent Symmetry



1) Butterfly Switch Simulator
2) 2D Monte Carlo Simulation
3) Ray Tracing
4) Parallel Linpack Benchmark

Intel Paragon XP/S

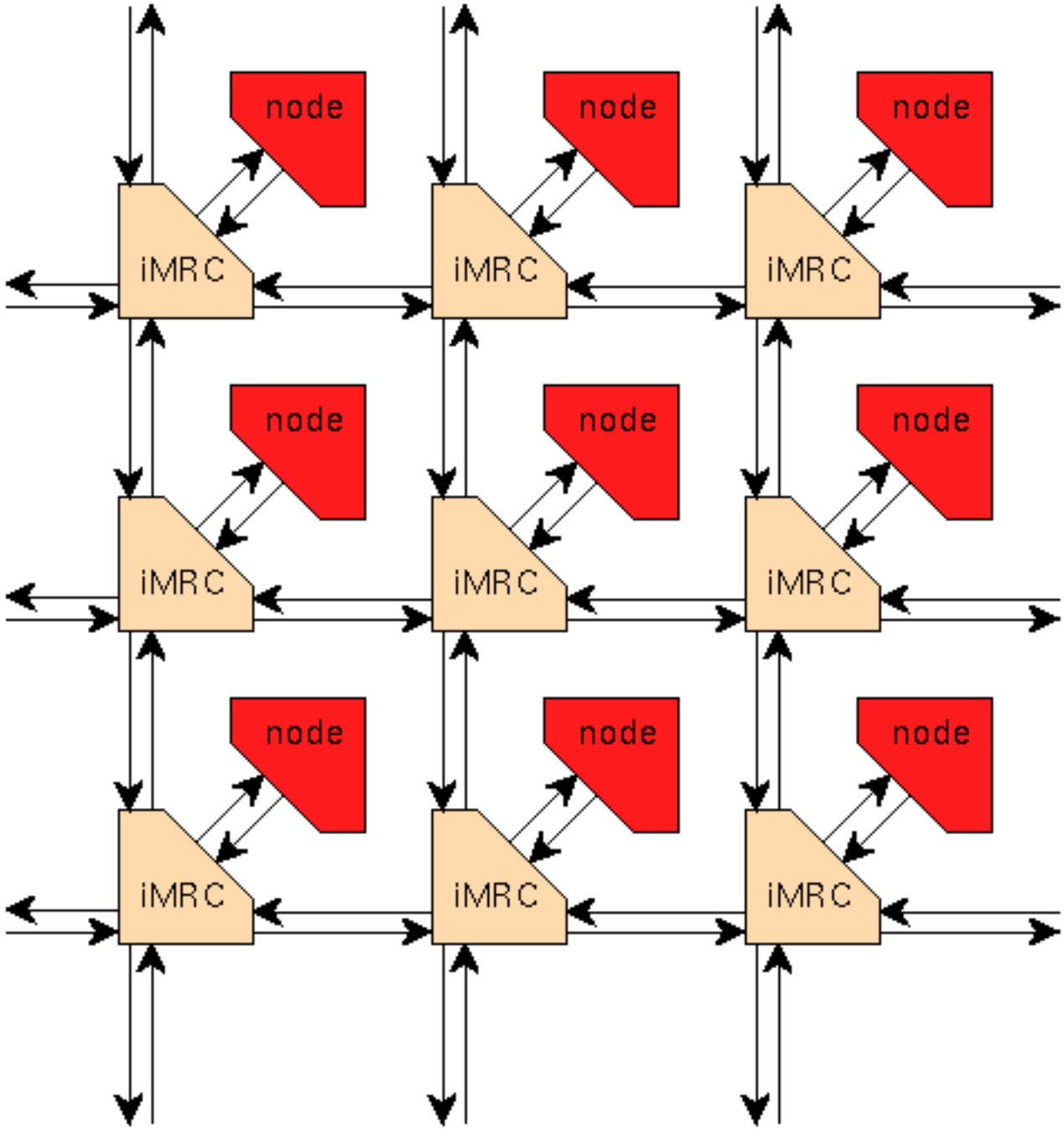
- i860 RISC processor (64 bit, 50 MHz, 75 MFlops)
- Standard OS (Mach) on each node
- Cluster in a box



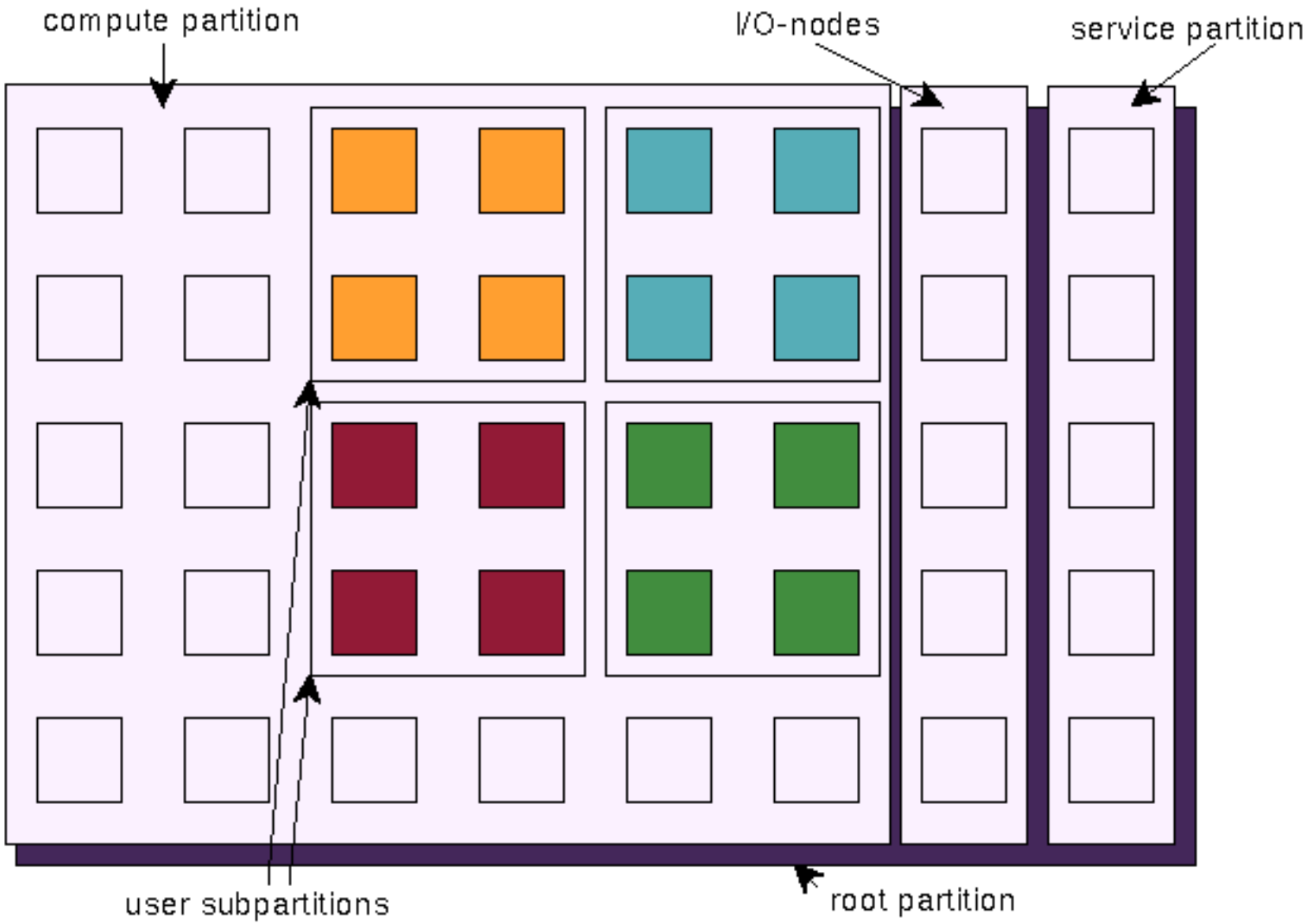
Paragon node

Intel
Paragon XP/S –

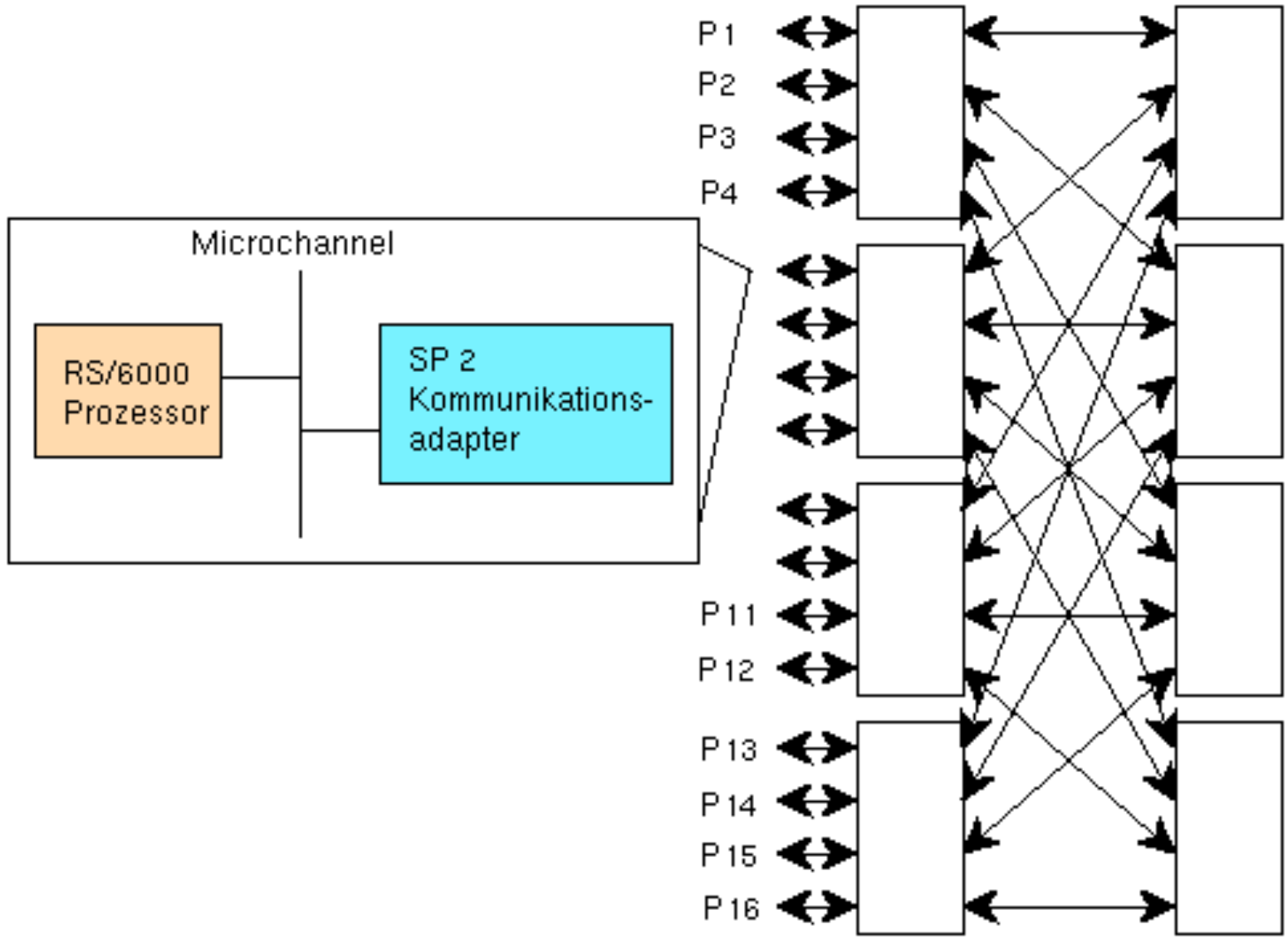
interconnection
network



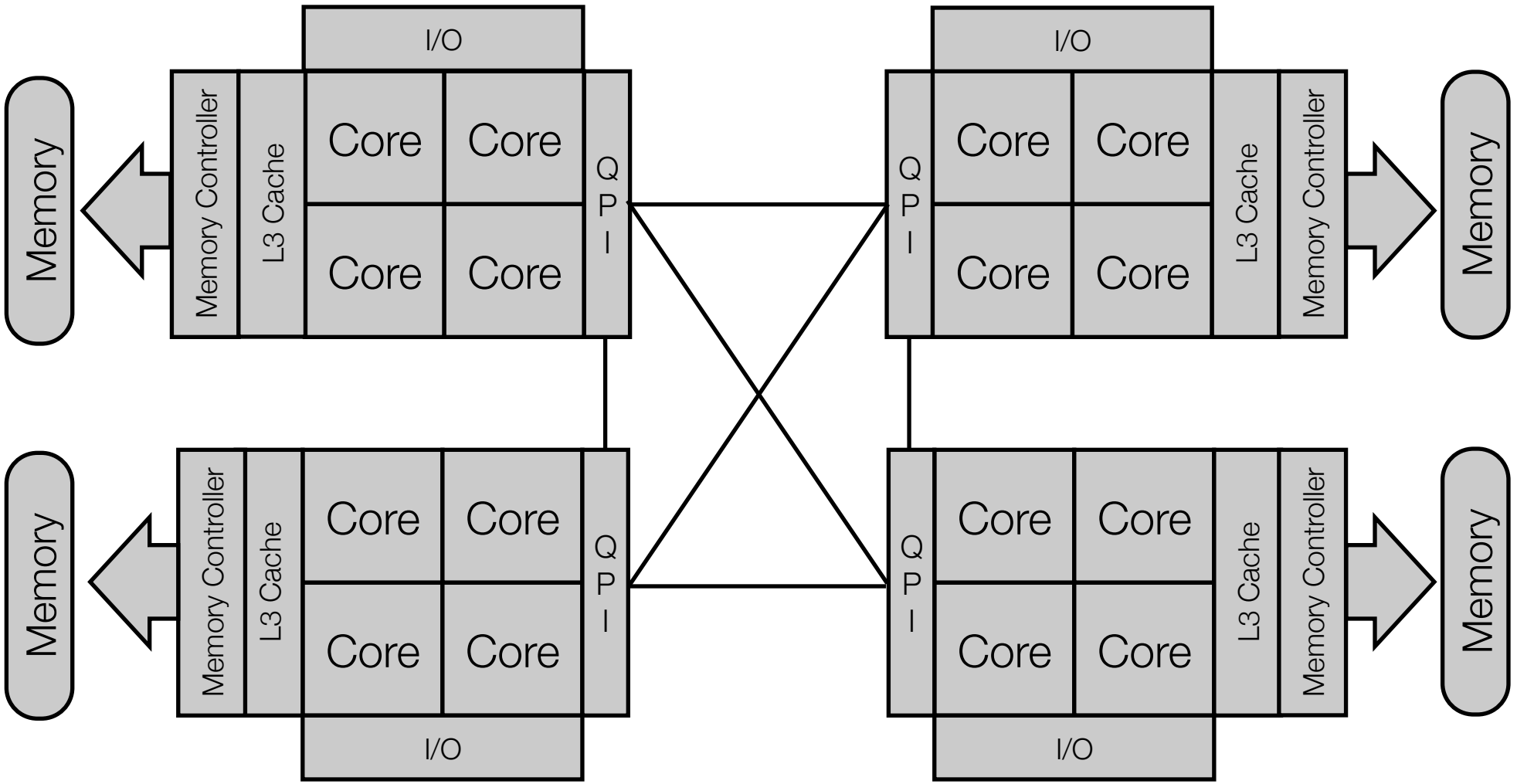
Intel Paragon XP/S - partitioning



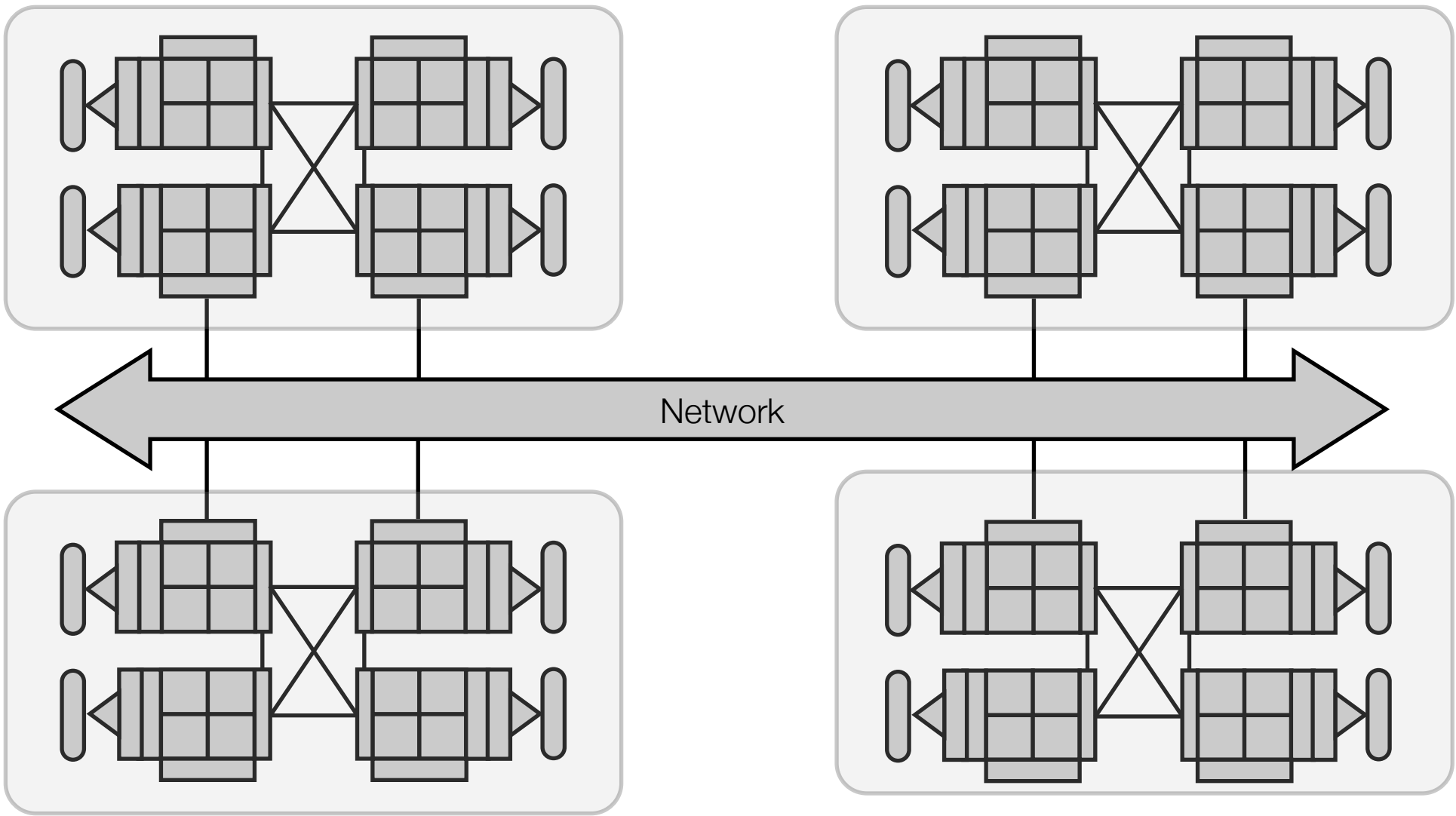
IBM SP/2


P1
P2
P3
P4
Microchannel
RS/6000
Prozessor
SP 2
Kommunikations-
adapter
P11
P12
P13
P14
P15
P16

Example: Intel Nehalem SMP System

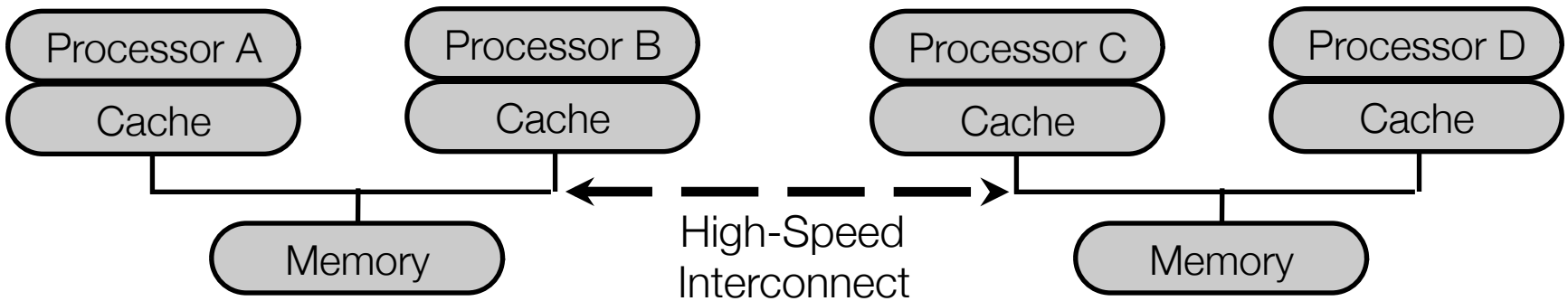


An Intel Nehalem Cluster:
SMP + NUMA + Distributed Memory



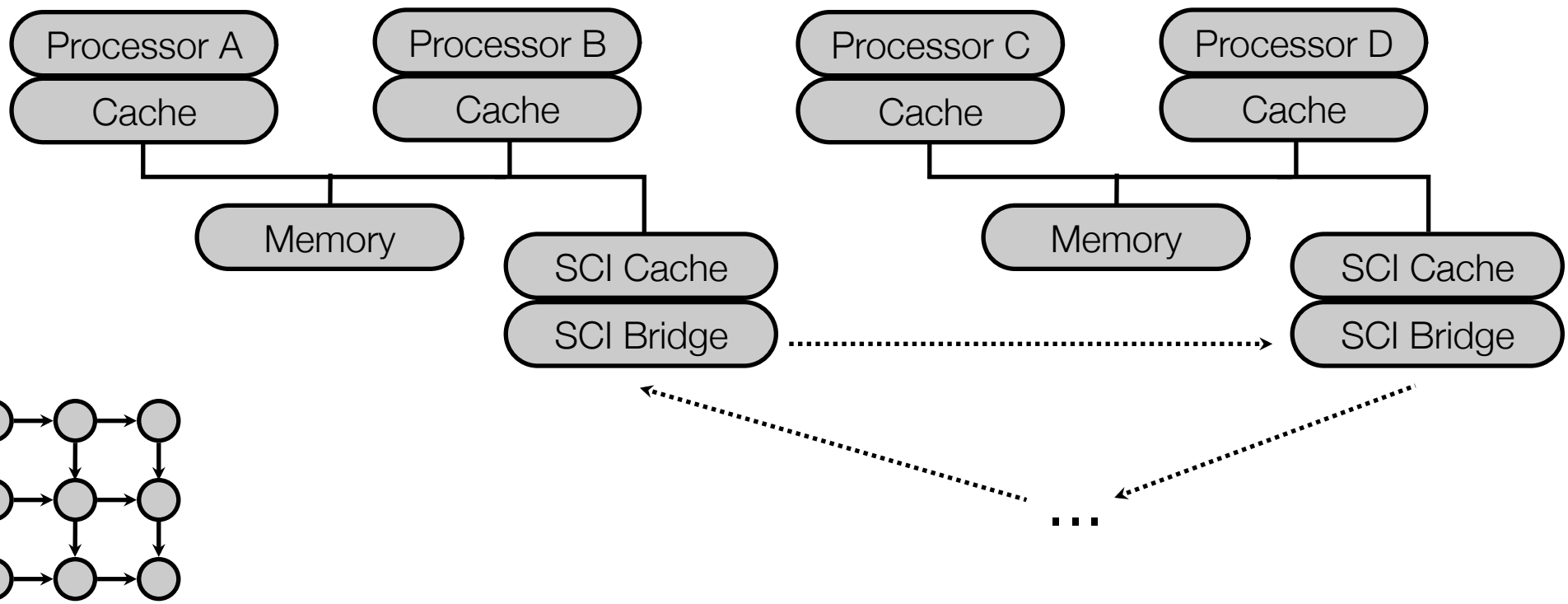
CC-NUMA

- Still SMP programming model, but non-NUMA aware software scales poorly
- Different implementations lead to diffuse understanding of „node“, typical:
  - Region of memory where every byte has the same distance from each processor
- Tackles scalability problems of pure SMP architectures, while keeping the location independence promise
- Recent research tendency towards non-cache-coherent NUMA approach (Intel Single Chip Cloud Computer)



Scalable Coherent Interface

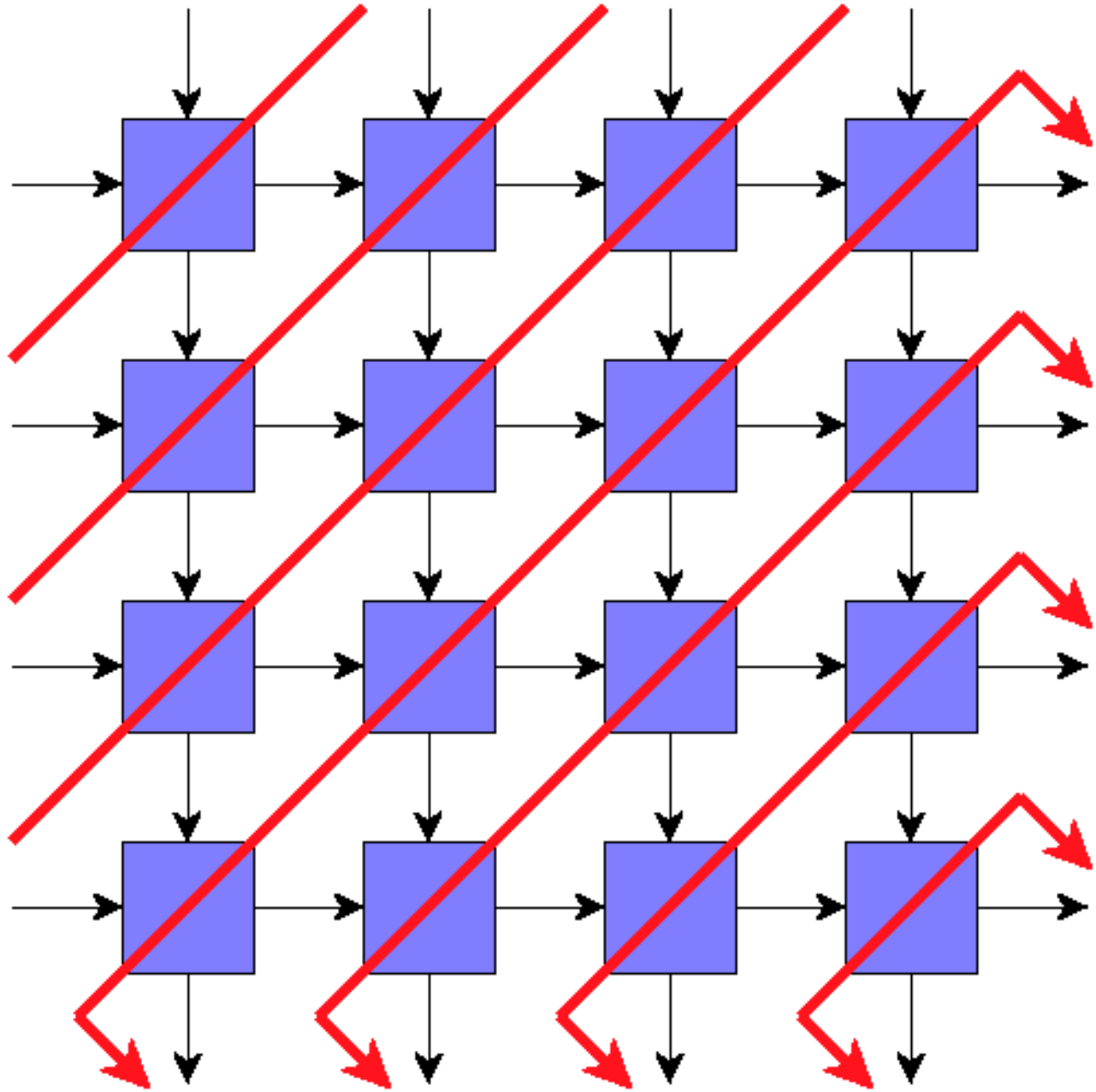
- ANSI / IEEE standard for NUMA interconnect, used in HPC world
  - 64bit global address space, translation by SCI bus adapter (I/O-window)
  - Used as 2D / 3D torus



Experimental Approaches

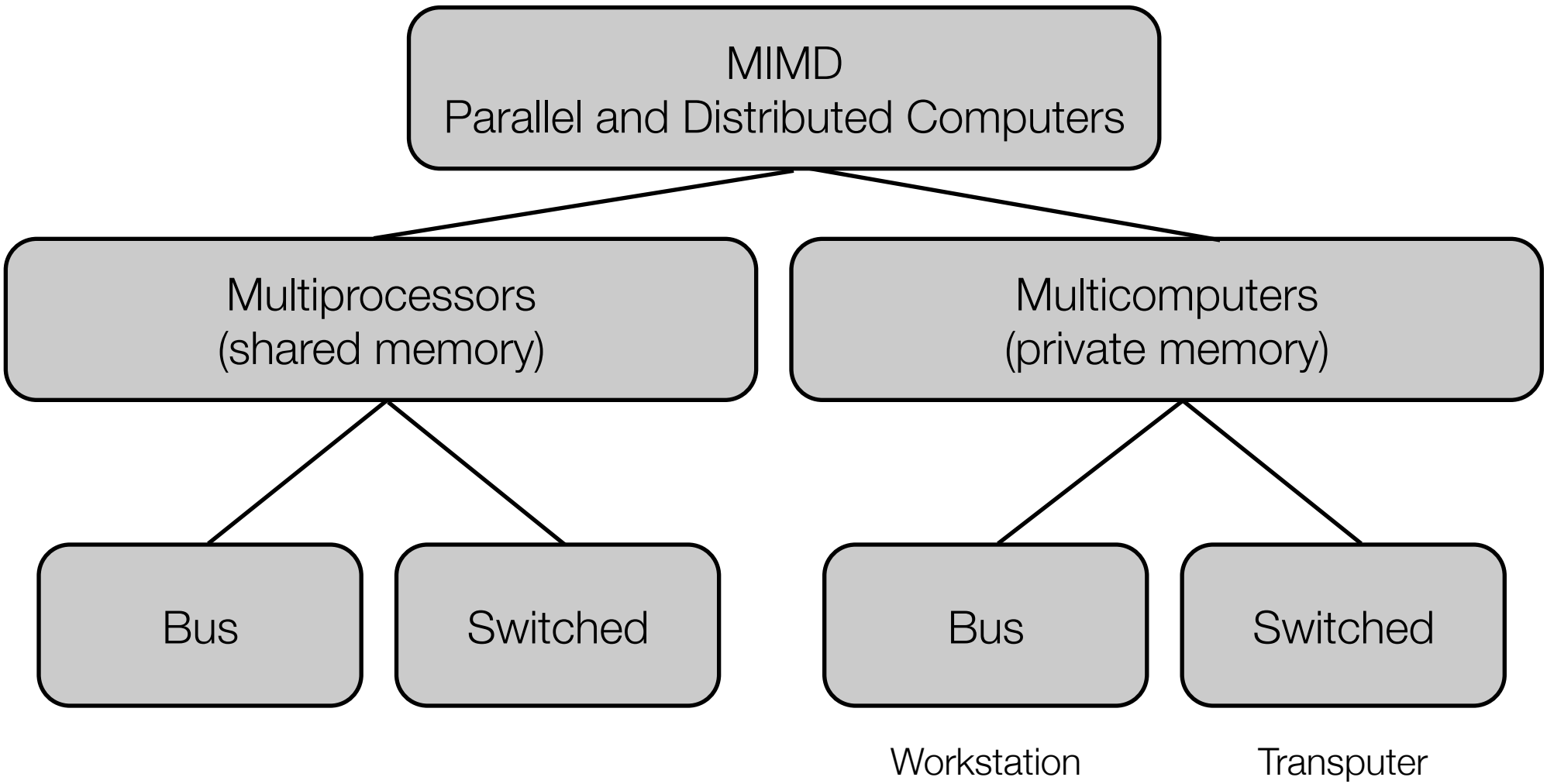
Systolic Arrays

- Data flow architectures
- Problem: common clock – maximum signal path restricted by frequency
- Fault contention: single faulty processing element will break entire machine



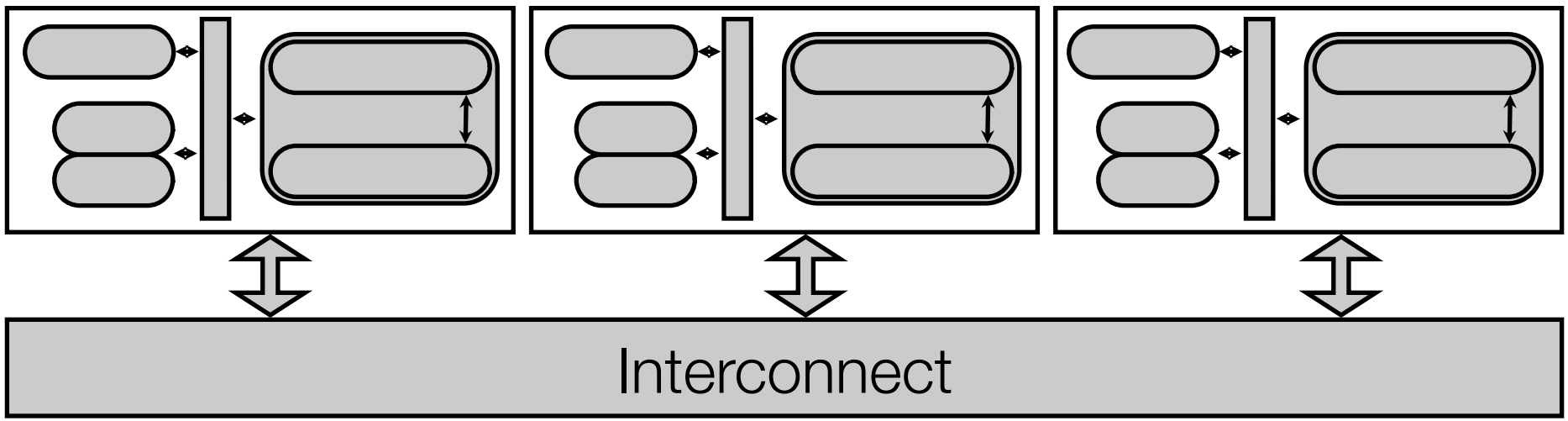
Systolische Arrays

Another Taxonomy (Tanenbaum)



Another Taxonomy (Foster)

- Multicomputer
  - Number of von Neumann computers, connected by a network (DM-MIMD)
  - Each computer runs own program and sends / receives messages
  - Local memory access is less expensive than remote memory access

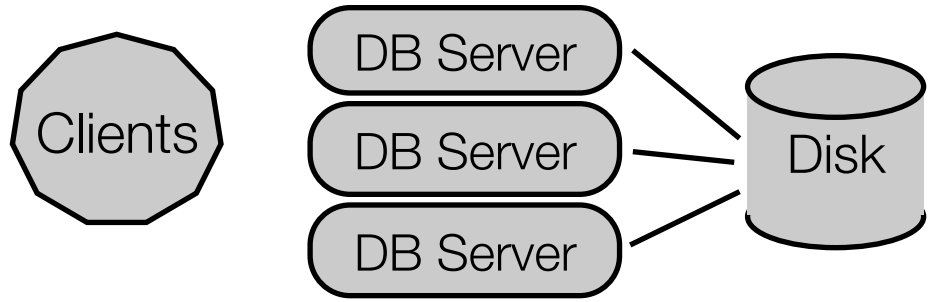


Multicomputer Systems - Clusters

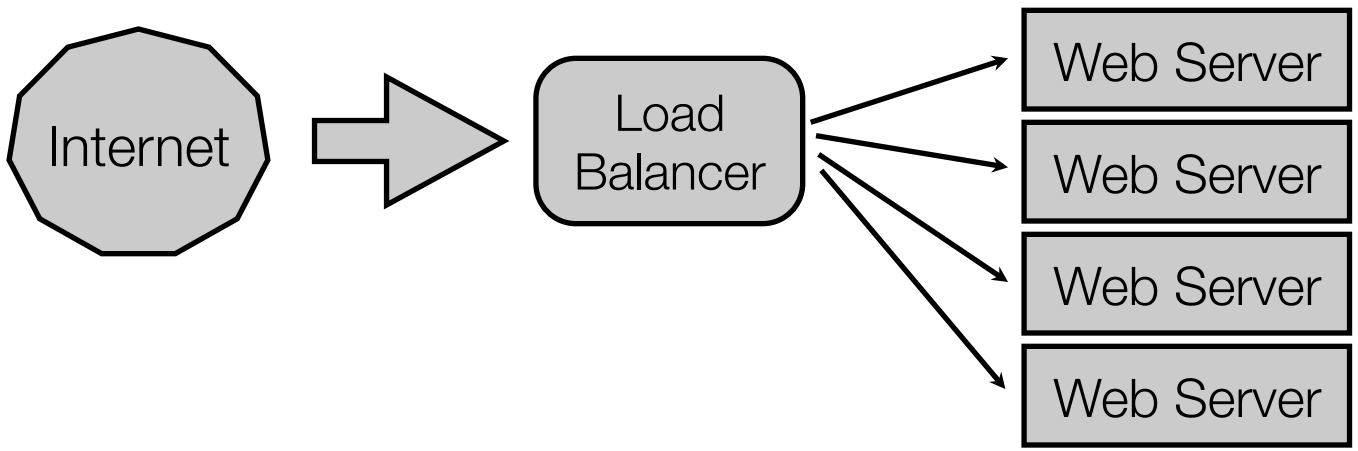
- Collection of stand-alone workstations/PC‘s connected by a local network
  - Cost-effective technique to connect small-scale computers to a large-scale parallel computer
  - Low cost of both hardware and software
  - **Users are builders**, have control over their own system (hardware infrastructure and software), low costs as major issue
- Distributed processing as extension of **DM-MIMD**
  - Communication between processors is orders of magnitude slower
  - PVM, MPI as widely accepted programming standards
    - Used with cheap LAN hardware

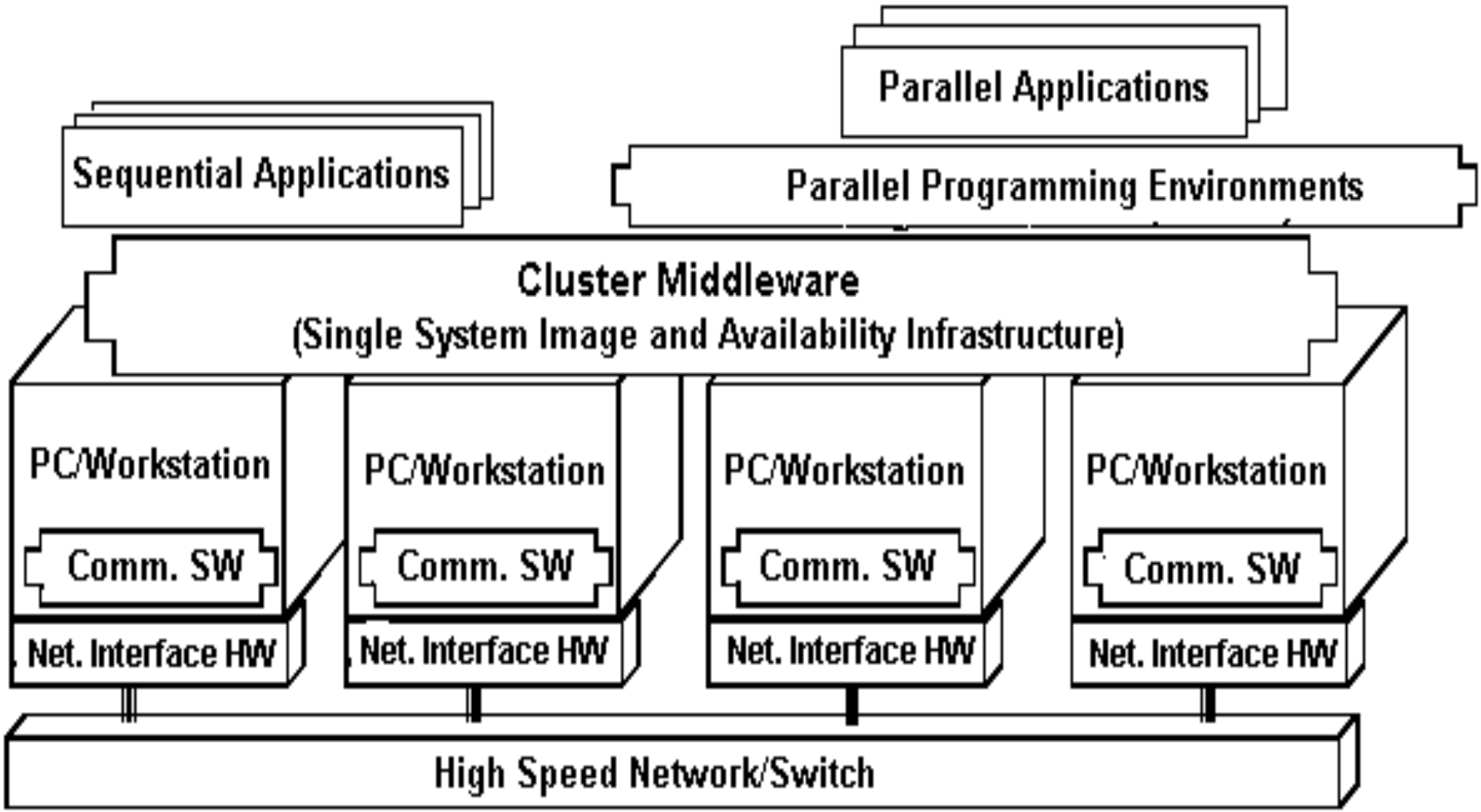
Lowly Parallel Processing

- Current market for large-scale parallel systems is small
  - High price, small amount of users
- Clusters provide cheap availability



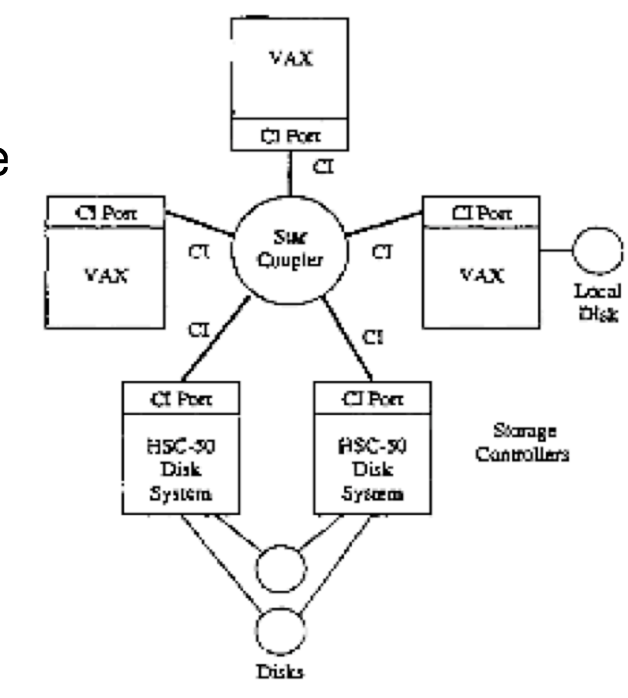
- Parallel processing with small amount of standard systems
  - Most coordination functionality realized by software, only feasible for coarse-grained parallel activities




Parallel Applications
Sequential Applications
Parallel Programming Environments
Cluster Middleware
(Single System Image and Availability Infrastructure)
PC/Workstation
Comm. SW
Net. Interface HW
PC/Workstation
Comm. SW
Net. Interface HW
PC/Workstation
Comm. SW
Net. Interface HW
PC/Workstation
Comm. SW
Net. Interface HW
High Speed Network/Switch

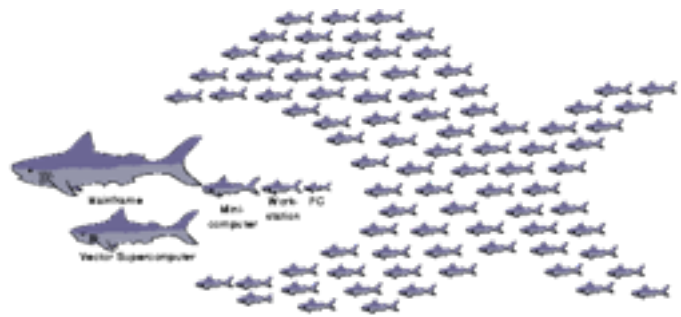
History

- 1977: ARCnet (Datapoint)
  - LAN protocol, such as Ethernet, DATABUS programming language
  - Single computer with terminals
  - Transparent addition of ‚compute resource‘ and ‚data resource‘ computers
- May 1983: VAXCluster (DEC)
  - Cluster of VAX computers, no single-point-of-failure
  - Every component that could fail was duplicated
  - High-speed message-oriented interconnect
  - Distributed version of VMS operating system
- Distributed lock manager for shared resources



NOW

- Berkeley Network Of Workstations (NOW) - 1995
- Building large-scale parallel computing system with COTS hardware
- GLUnix operating system
  - Transparent remote execution, network PID‘s
  - Load balancing
  - Virtual Node Numbers (for communication)
- Network RAM - idle machines as paging device
- Collection of low-latency, parallel communication primitives - ‘active messages’
- Berkeley sockets, shared address space parallel C, MPI



NOW

MareNostrum

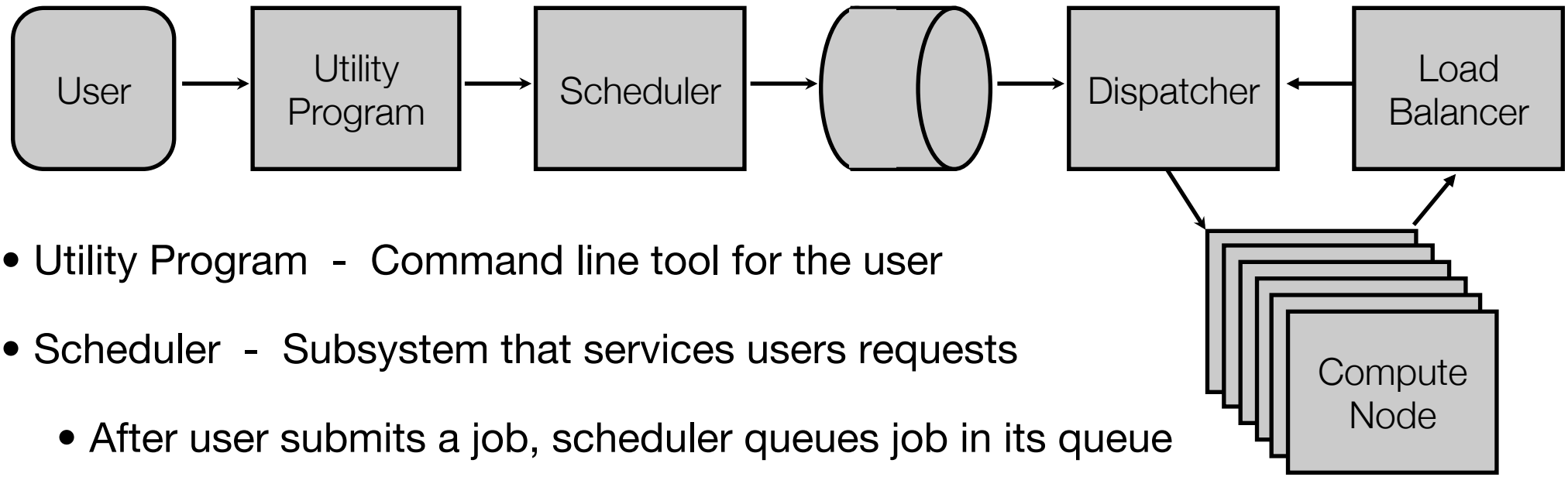
- Peak Performance of 94,21 Teraflops
- 10.240 IBM Power PC 970MP processors at 2.3 GHz (2560 JS21 blades)
- 44 racks with 6 blade centers each, 120m^2
- Each blade center with 28 blades, each blade with 2 processors
- 8 GB shared memory and local harddisk per blade card, standard Linux
- Operating system from network (Gigabit), Myrinet interconnect

(C) bsc.es

Cluster System Classes

- **High-availability (HA) clusters** - Improvement of cluster availability
  - Linux-HA project (multi-protocol heartbeat, resource grouping)
- **Load-balancing clusters** - Server farm for increased performance / availability
  - Linux Virtual Server (IP load balancing, application-level balancing)
- **High-performance computing (HPC) clusters** - Increased performance by splitting tasks among different nodes
  - Speed up the computation of one distributed job (FLOPS)
- **High-throughput computing (HTC) clusters** - Maximize the number of finished jobs
  - All kinds of simulations, especially parameter sweep applications
  - Special case: Idle Time Computing for cycle harvesting

Simple Queuing Management System



- Utility Program - Command line tool for the user
- Scheduler - Subsystem that services users requests
  - After user submits a job, scheduler queues job in its queue
  - Makes decision based on scheduling policy
- Queue - Collection of jobs, order based on attributes/policy
- Dispatcher - Performs the submission of jobs in queue
- Load Balancer - Selects appropriate set of compute nodes, based on monitoring

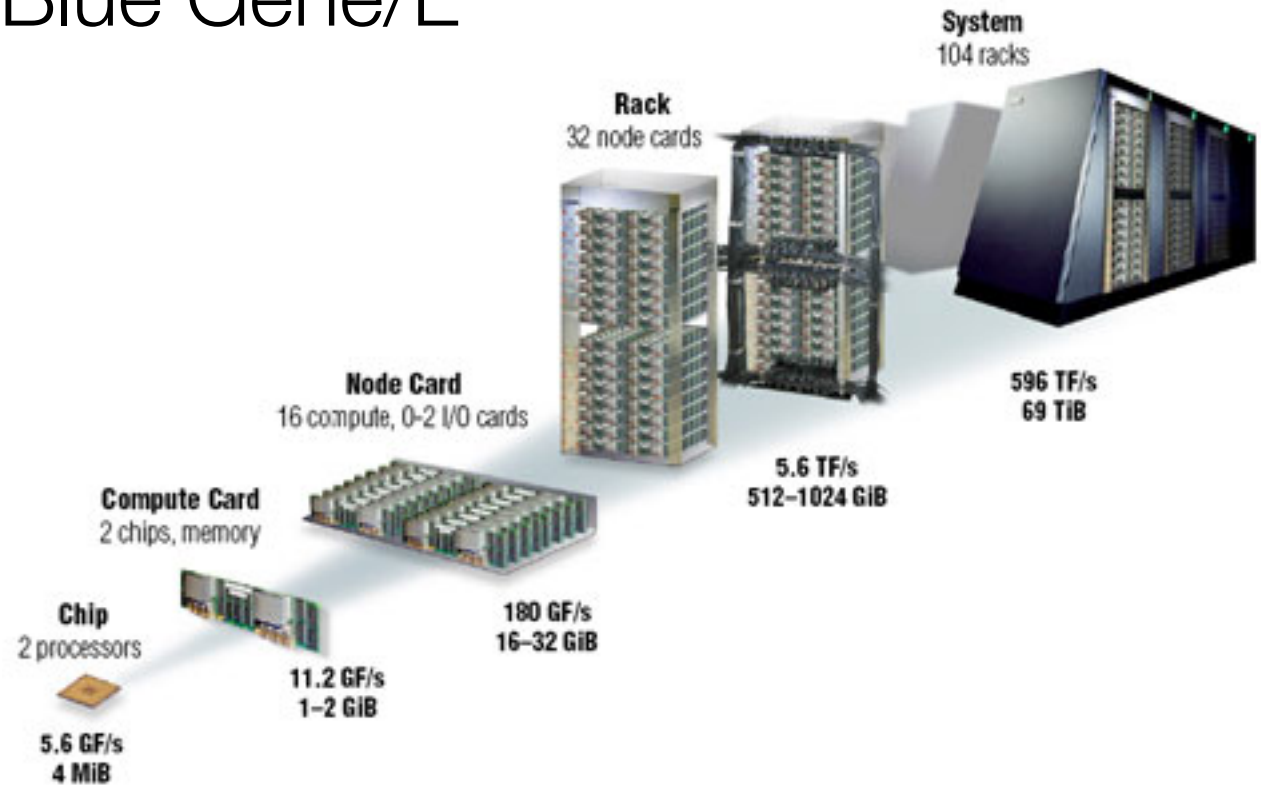
Clusters for Scalability and Availability

- Advantages
  - Intrinsic availability due to node independence
  - Simple commodity hardware, low costs
  - Application-level compatibility with uniprocessors
- Disadvantages
  - Single system image mostly not possible, therefore explicit programming
  - Complex administration
  - Power consumption

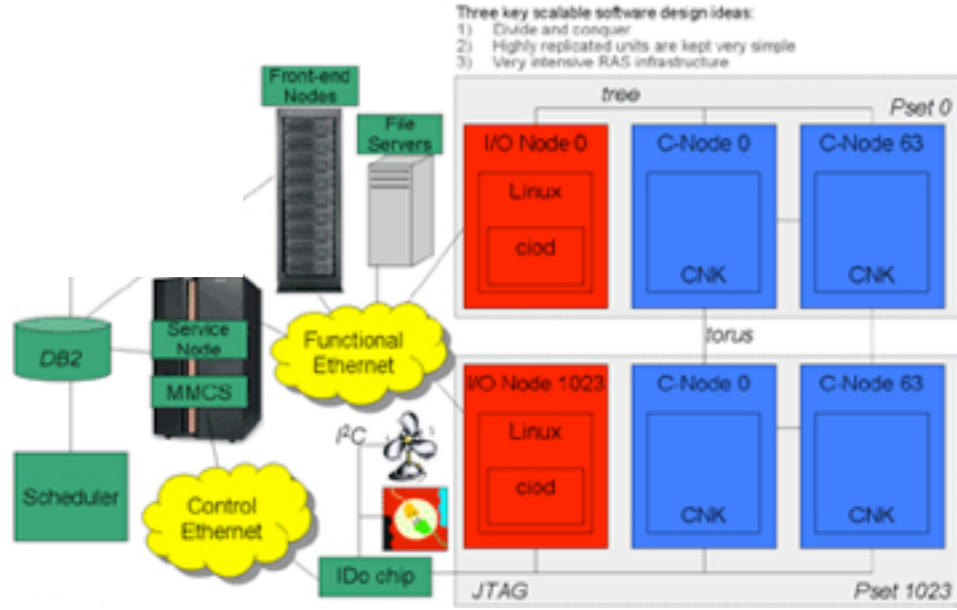
Multicomputer Systems - Massively Parallel Processing (MPP)

- Hierarchical SIMD / MIMD architecture with **many** interconnected processors
  - Standard components (in contrast to mainframes)
  - Host processor(s) / nodes, responsible for loading program and data to PE‘s
  - High-performance interconnect (bus, ring, 2D mesh, hypercube, tree, ...)
  - For embarrassingly parallel applications, mostly simulations (atom bomb, climate, earthquake, airplane, car crash, ...)
- Examples
  - Distributed Array Processor (1979), 64x64 single bit processing elements
  - Goodyear Massively Parallel Processor (1985), 128x128 single bit PE‘s
  - Earth Simulator (2004), 640 nodes x 8 vector processors per node
  - BlueGene/L (2007), 106.496 nodes x 2 PowerPC processors (700MHz)

Blue Gene/L



(C) LLNL



- Proprietary ASIC with 2 PowerPC 440 cores on a chip
- Proprietary communication networks organized as 3D torus (memory mapped)
- 596 TerraFLOPS, MTBF 6 days

MPP Properties

- Standard components (processors, harddisks, ...)
- Specific non-standardized interconnection network
  - Low latency, high speed
  - Distributed file system
- Specific packaging of components and nodes for cooling and upgradeability
  - Whole system provided by one vendor (IBM, HP)
  - Extensibility as major issue, in order to save investment
- Distributed processing as extension of **DM-MIMD**
- Single System View
  - Common file system, central job scheduling

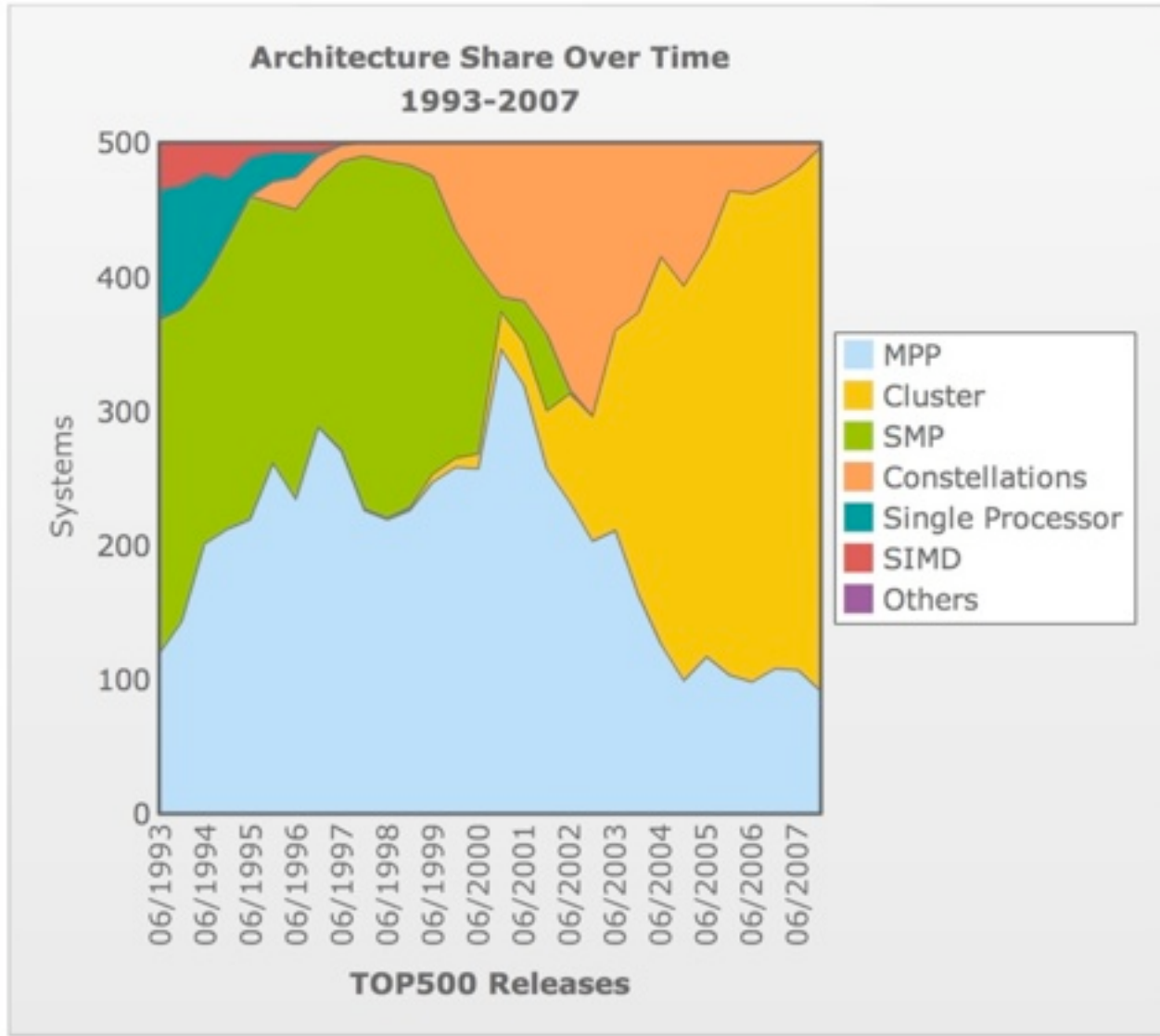
MPP for Scalability and Availability

- Advantages
  - Performance scalability only limited by application, and not by hardware
  - Proprietary wiring of standard components as alternative to mainframes
- Disadvantages
  - Specialized interconnect network
  - Demands custom operating system and aligned applications
  - No major consideration of availability
  - Power consumption, cooling

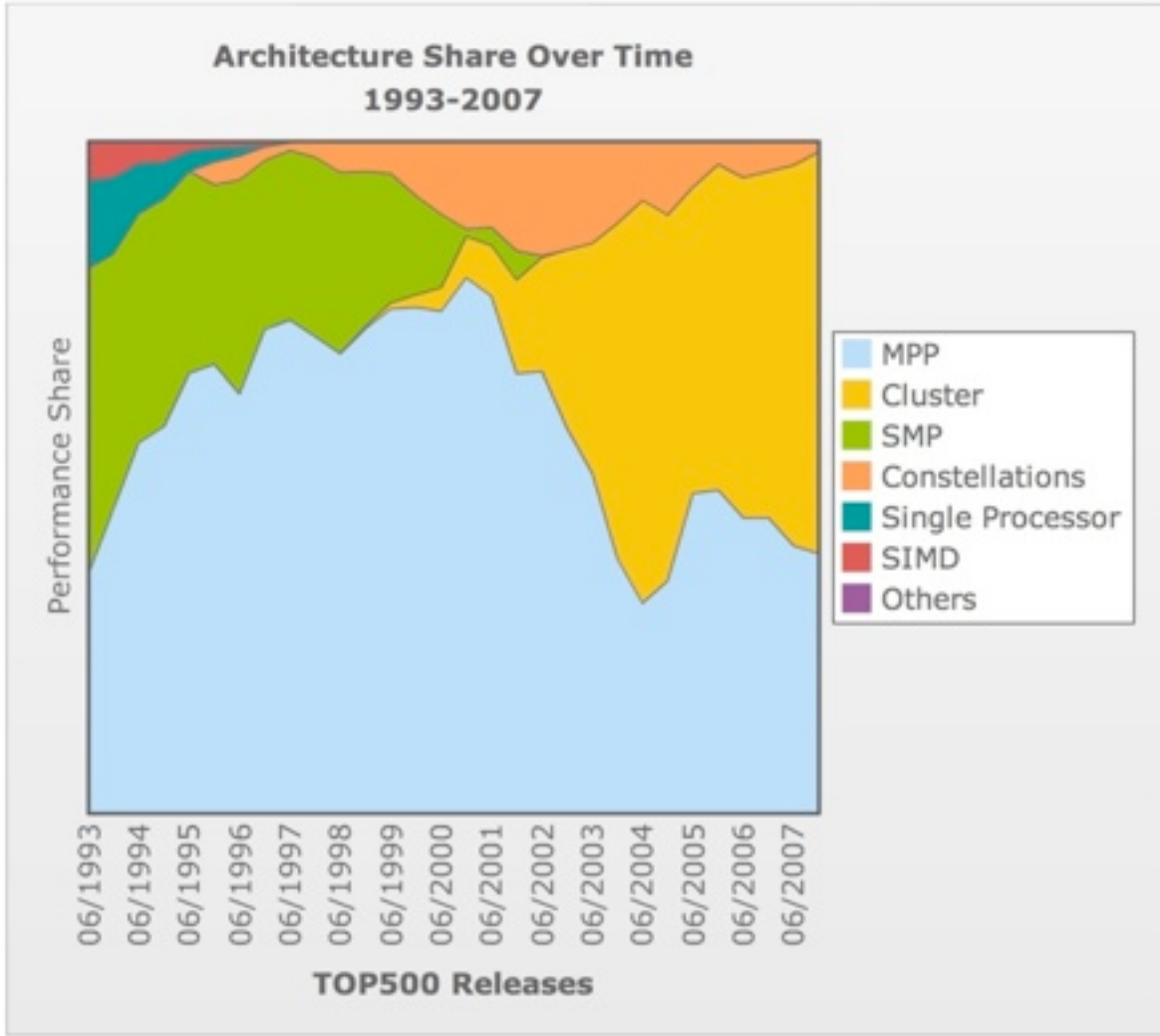
Cluster / MPP System Benchmarking

- TOP500.org
  - Collection started in 1993, updated every 6 months
  - 500 most powerful computers / clusters in the world
  - Comparison through Linpack benchmark results
- Linpack
  - Measures floating point rate of execution
  - Solving of a dense system of linear equations, grown since 1979
  - Compiler optimization is allowed, no changes to the code
  - 3 tests: Linpack Fortran n=100, Linpack n=1000, Linpack Highly Parallel Computing Benchmark (used for TOP500 ranking)

Top 500 - Clusters vs. MPP (# systems)



Top 500 - Clusters vs. MPP (performance)



	MPP	SMP	Cluster	Distributed
Number of nodes	O(100)-O(1000)	O(10)-O(100)	O(100) or less	O(10)-O(1000)
Node Complexity	Fine grain	Medium or coarse grained	Medium grain	Wide range
Internode communication	Message passing / shared variables (SM)	Centralized and distributed shared memory	Message Passing	Shared files, RPC, Message Passing, IPC
Job scheduling	Single run queue on host	Single run queue mostly	Multiple queues but coordinated	Independent queues
SSI support	Partially	Always in SMP	Desired	No
Address Space	Multiple	Single	Multiple or single	Multiple
Internode Security	Irrelevant	Irrelevant	Required if exposed	Required
Ownership	One organization	One organization	One or many organizations	Many organizations

K. Hwang and Z. Xu, Scalable Parallel Computing: Technology, Architecture, Programming; WCB/McGraw-Hill, 1998

Distributed System

- Tanenbaum (Distributed Operating Systems):
  „A distributed system is a collection of independent computers that appear to the users of the system as a single computer.“
- Coulouris et al.:
  „... [system] in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.“
- Lamport:
  „A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.”
- Consequences: concurrency, no global clock, independent failures
- Challenges: heterogeneity, openness, security, scalability, failure handling, concurrency, need for transparency

Grid Computing

- „... coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.“
  Foster, Kesselman, Tueke „The Anatomy of the Grid“, 2001
- Analogy to the power grid
- Request-driven usage of standardized services
- Reliable, high availability, low costs
- Innovation was the coordinated distribution, not the power itself
- Resource coordination without central control
- Open standards (Open Grid Forum)
- Service quality (reliability, throughput, availability, security)

SMP vs. Cluster vs. Distributed System

- Clusters are composed of computers, SMPs are composed of processors
  - High availability is cheaper with clusters, but demands other software
  - Scalability is easier with a cluster
  - SMPs are easier to maintain from administrators point of view
  - Software licensing becomes more expensive with a cluster
- Clusters for **capability computing**, integrated machines for **capacity computing**
- Cluster vs. Distributed System
  - Both contain of multiple nodes for parallel processing
  - Nodes in a distributed system have their own identity
  - Physical vs. virtual organization

Interconnection networks

Optimization criteria

- Connectivity – ideally direct links between any two stations
- High number of parallel connections

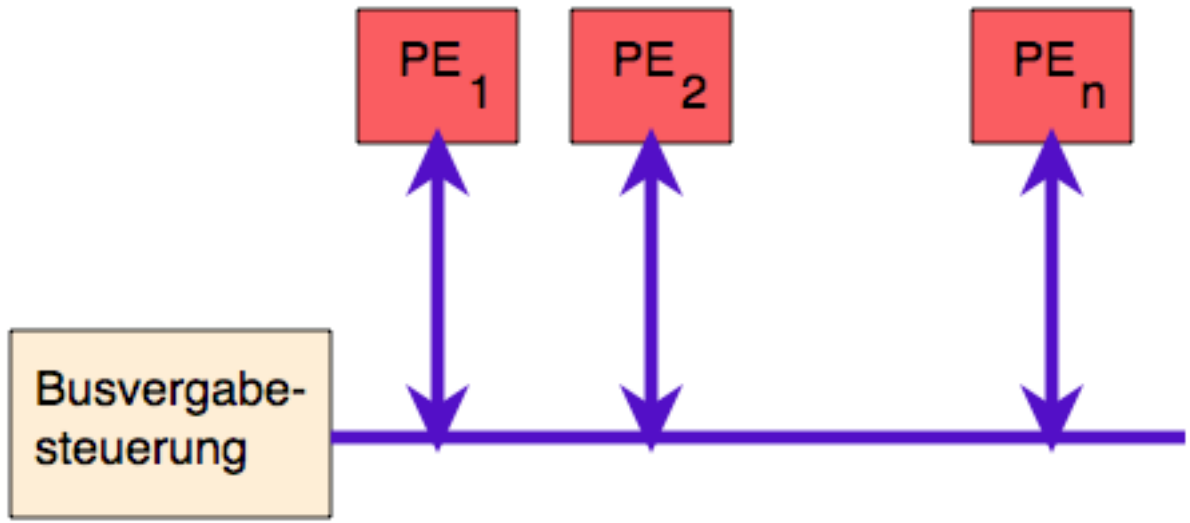
Cost model

- Production cost - # connections
- operational cost – distance among PEs

- Bus networks, switching networks, point-to-point interconnects

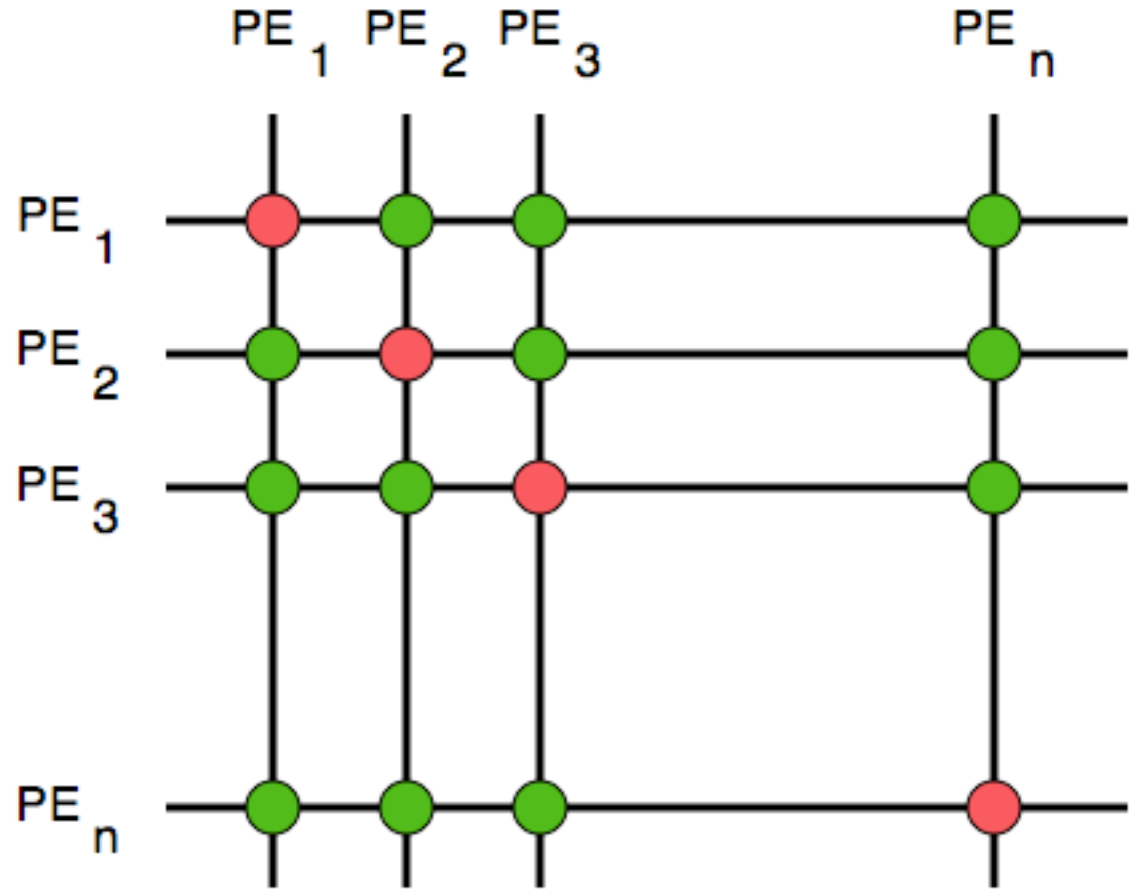
Bus network

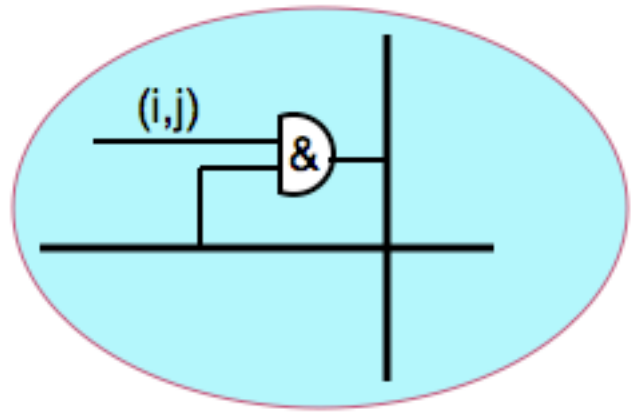
- Optimal #connection per PE: 1
- Constant distance among any two PEs



Crossbar switch (Kreuzschienenverteiler)

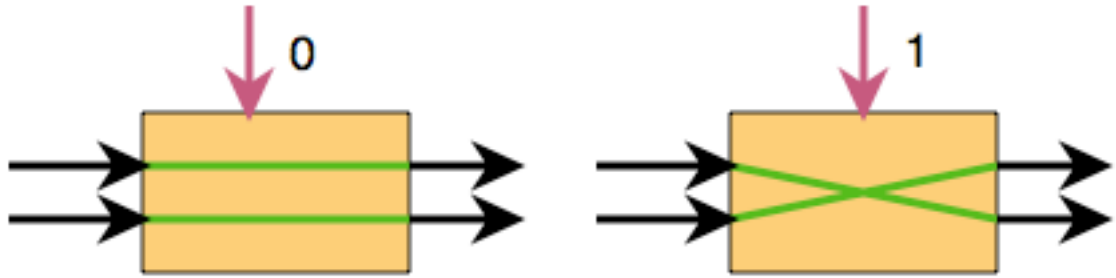
- Arbitrary number of permutations
- Collision-free data exchange
- High cost, quadratic growth
- n * (n-1) connection points

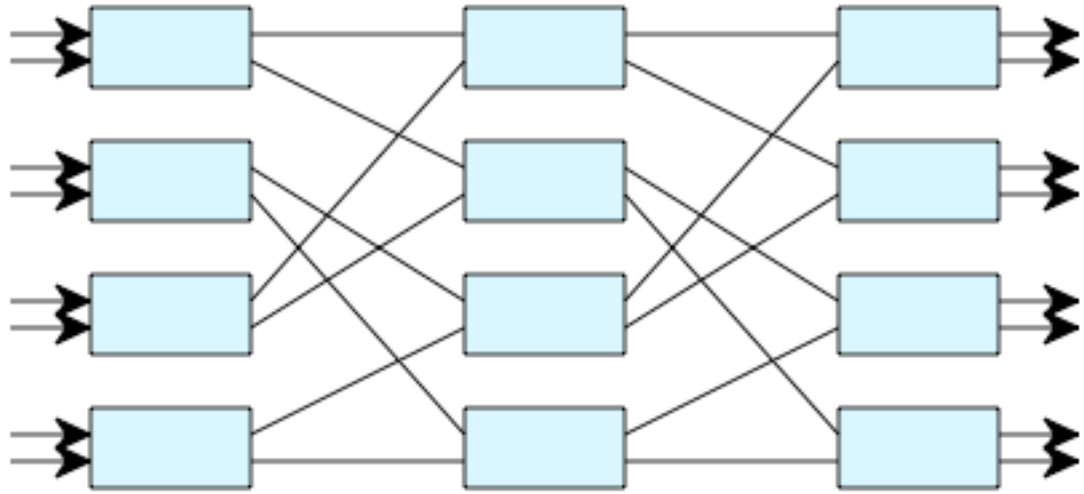




Delta networks

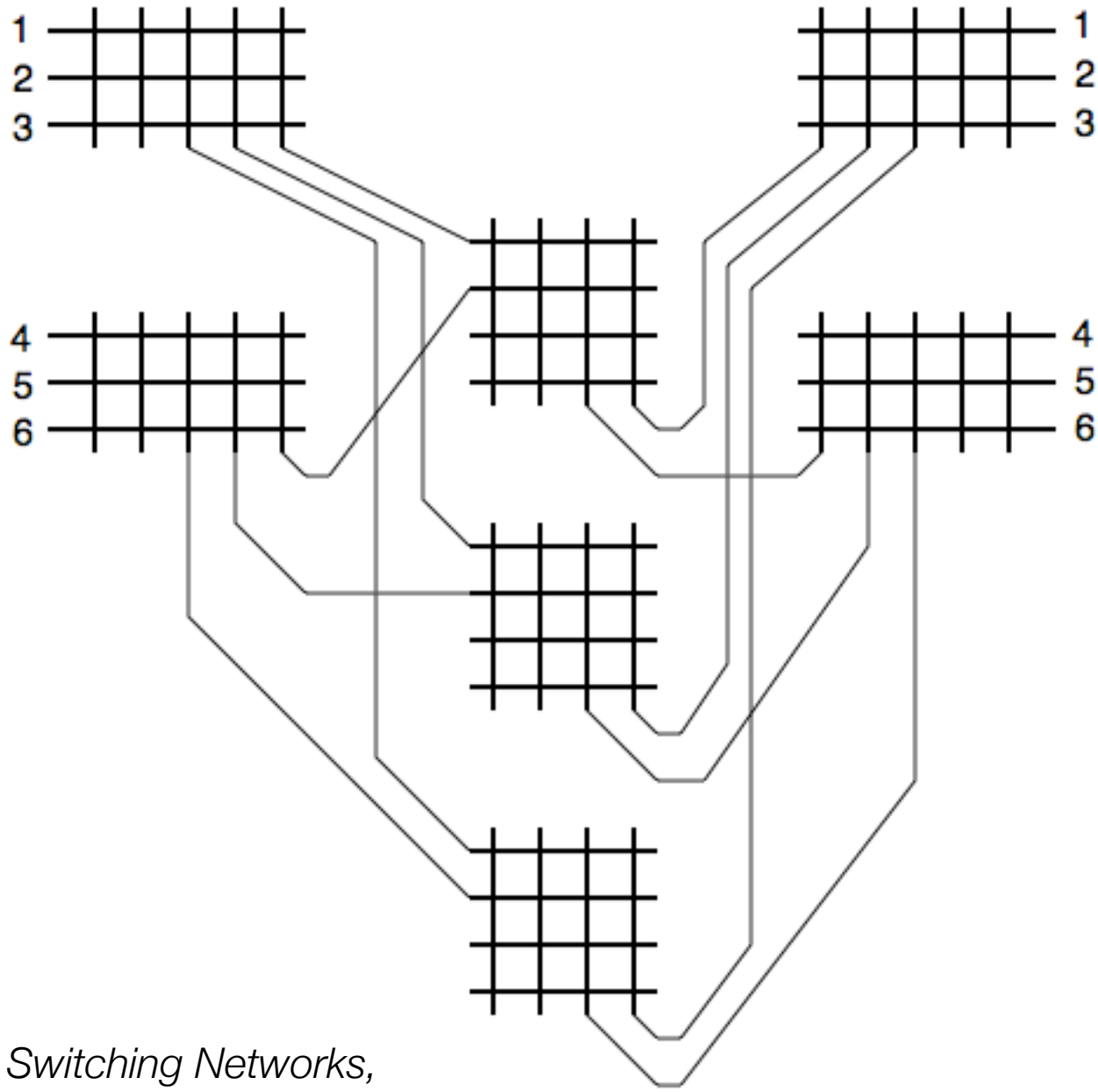
- Only n/2 log n delta-switches
- Limited cost
- Not all possible permutations operational in parallel





Clos coupling networks

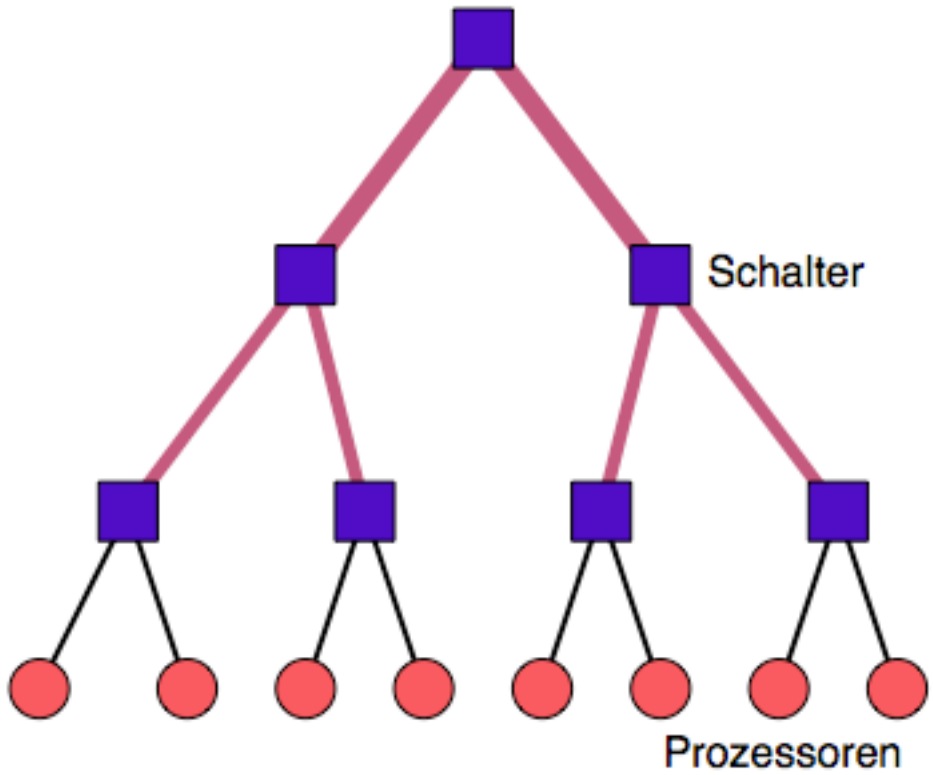
- Combination of delta network and crossbar

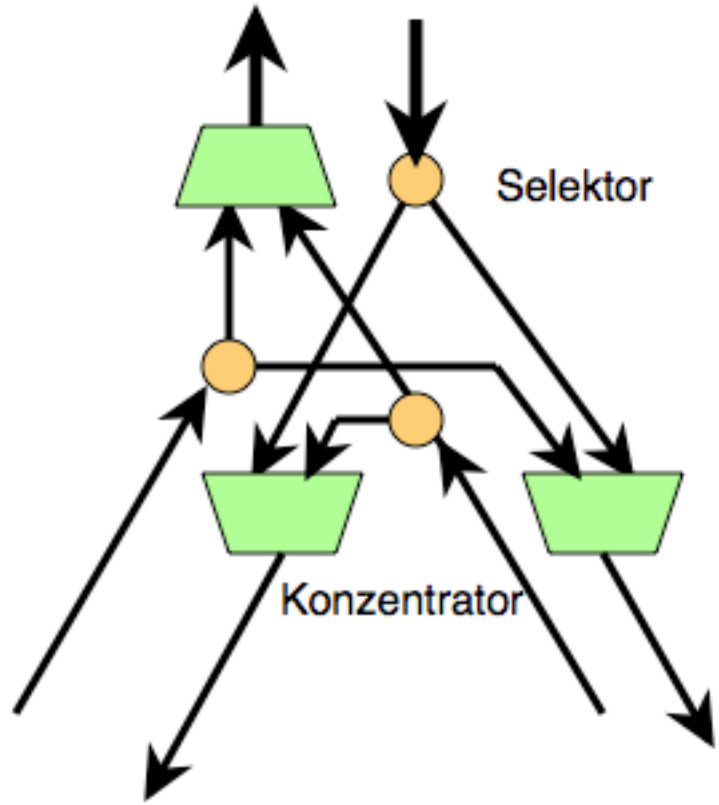


C.Clos, *A Study of Nonblocking Switching Networks, Bell System Technical Journal, vol. 32, no. 2, 1953, pp. 406-424(19)*

Fat-Tree networks

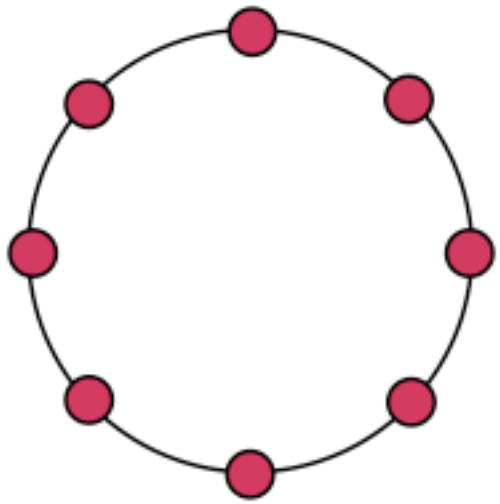
- PEs arranged as leafs on a binary tree
- Capacity of tree (links) doubles on each layer



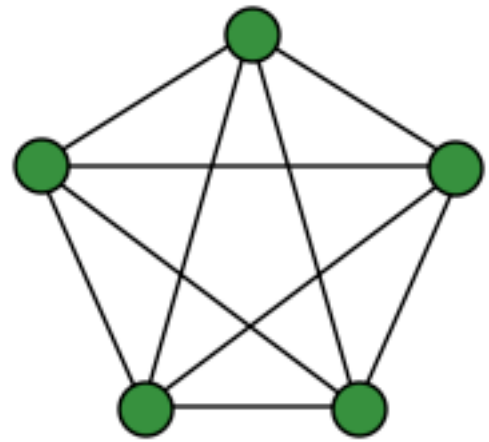


Point-to-point networks: ring and fully connected graph

- Ring has only two connections per PE (almost optimal)
- Fully connected graph – optimal connectivity (but high cost)



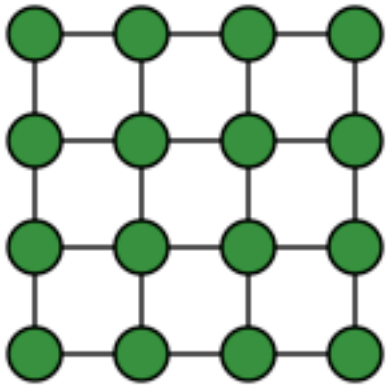
Ring



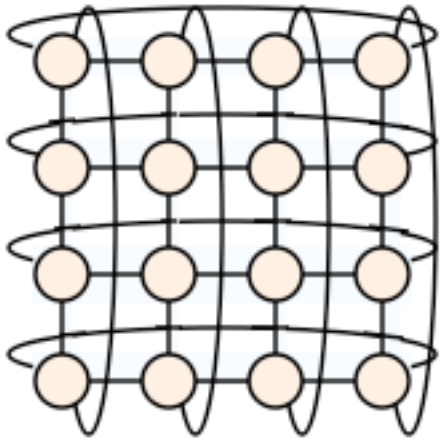
vollständiger Graph

Mesh and Torus

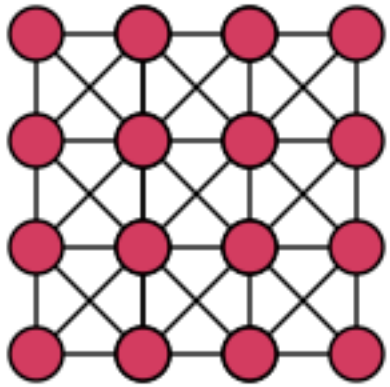
- Compromise between cost and connectivity



Quadratisches Gitter (4-way)



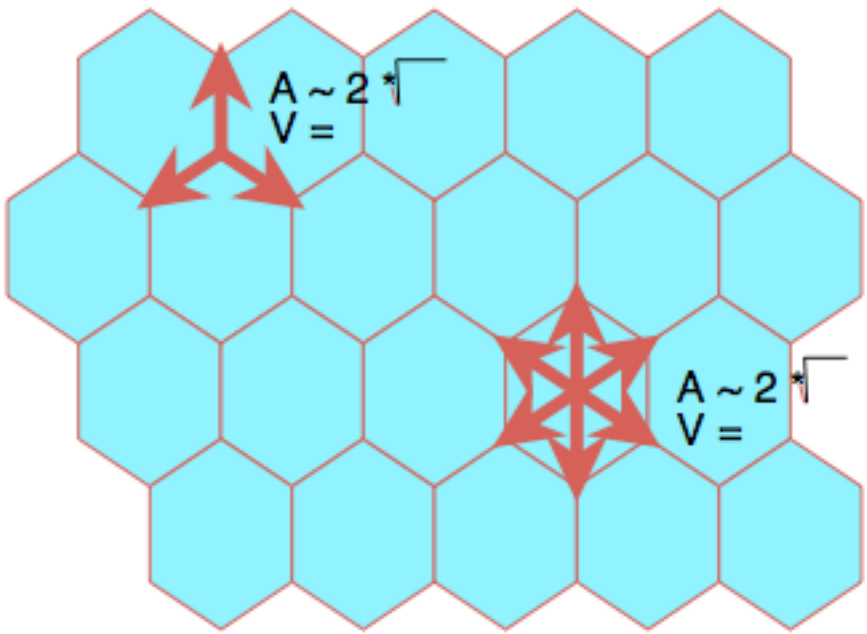
Quadratischer Torus (4-way)

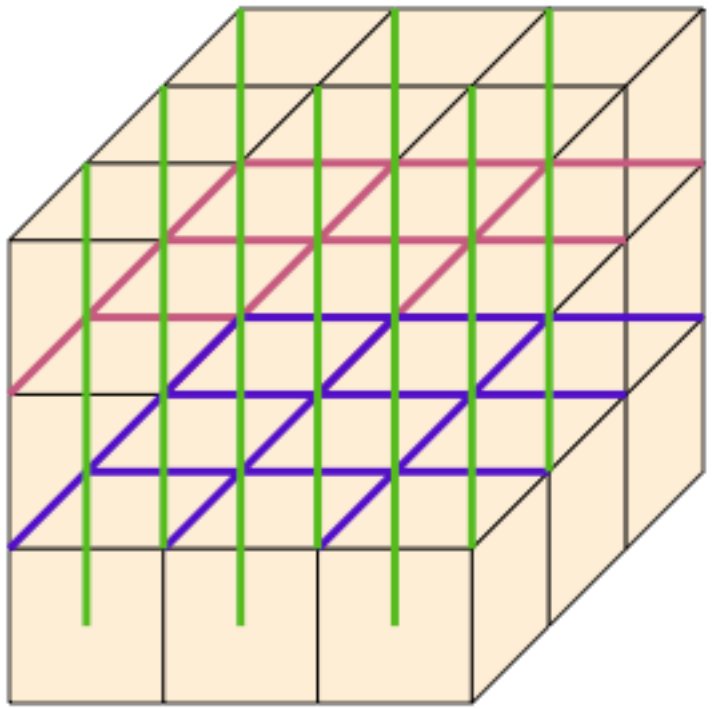


Quadratisches Gitter (8-way)

Cubic Mesh

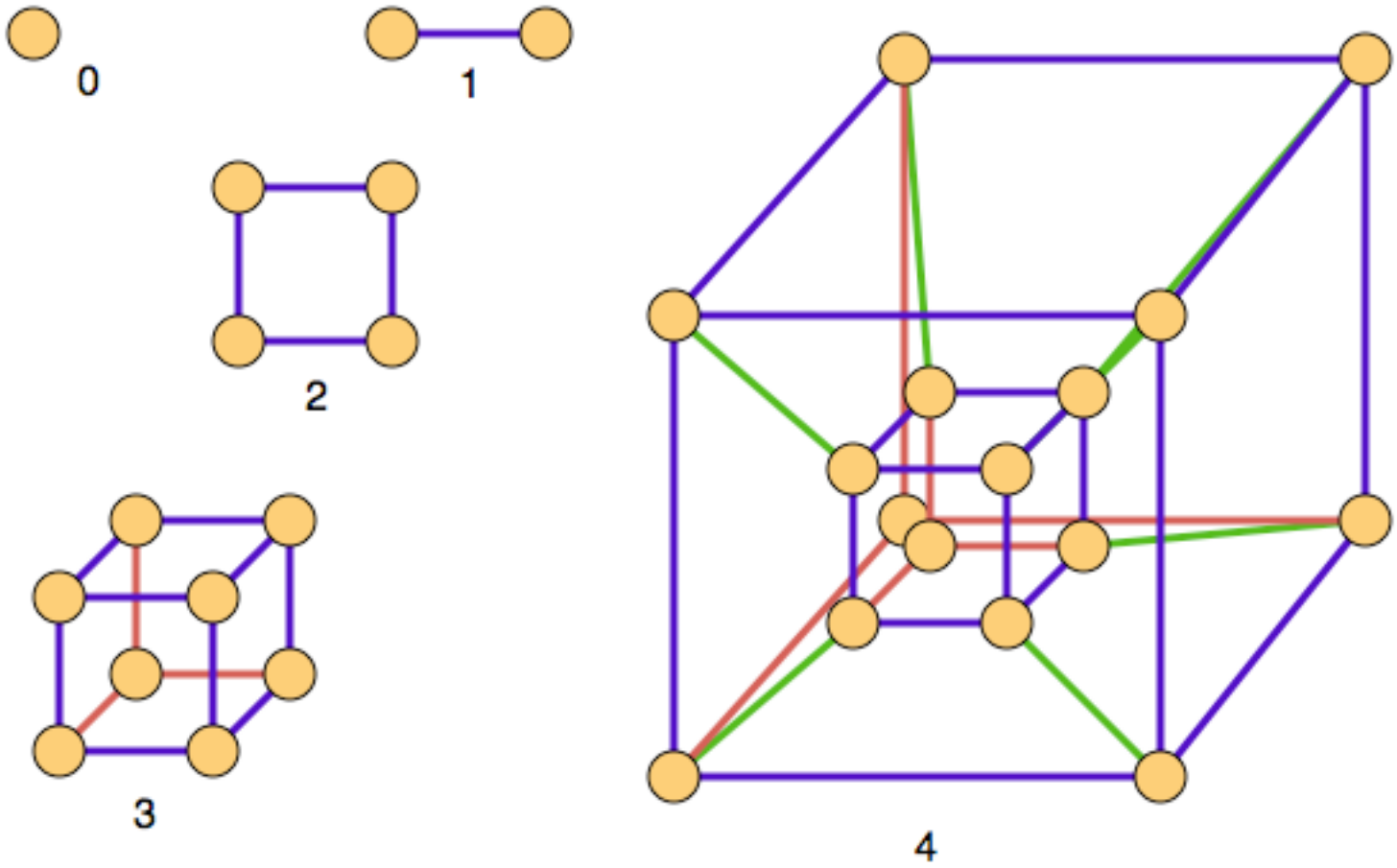
- PEs are arranged in a cubic fashion
- Each PE has 6 links to neighbors





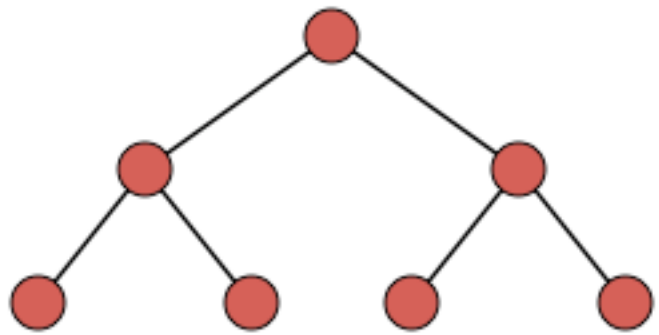
Hypercube

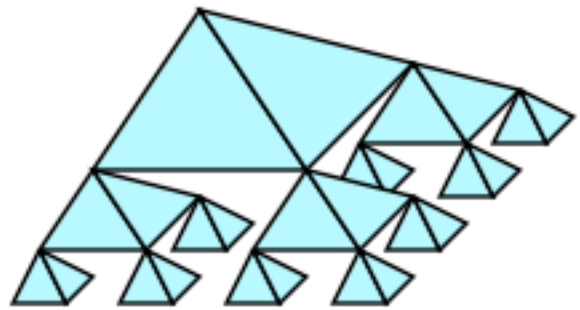
- Dimensions 0-4, recursive definition



Binary tree, quadtree

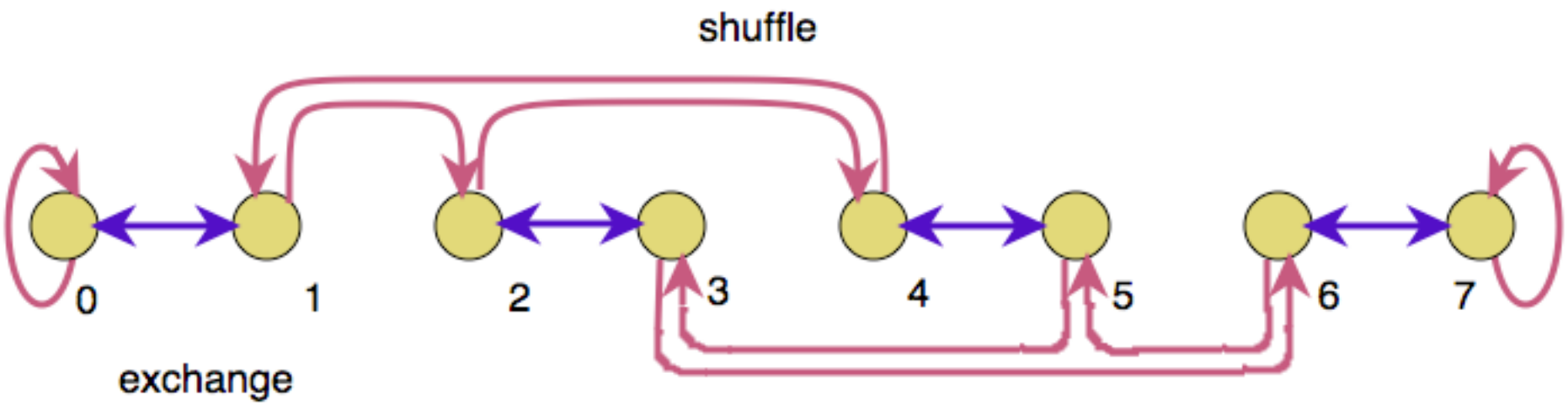
- Logarithmic cost
- Problem of bottleneck at root node





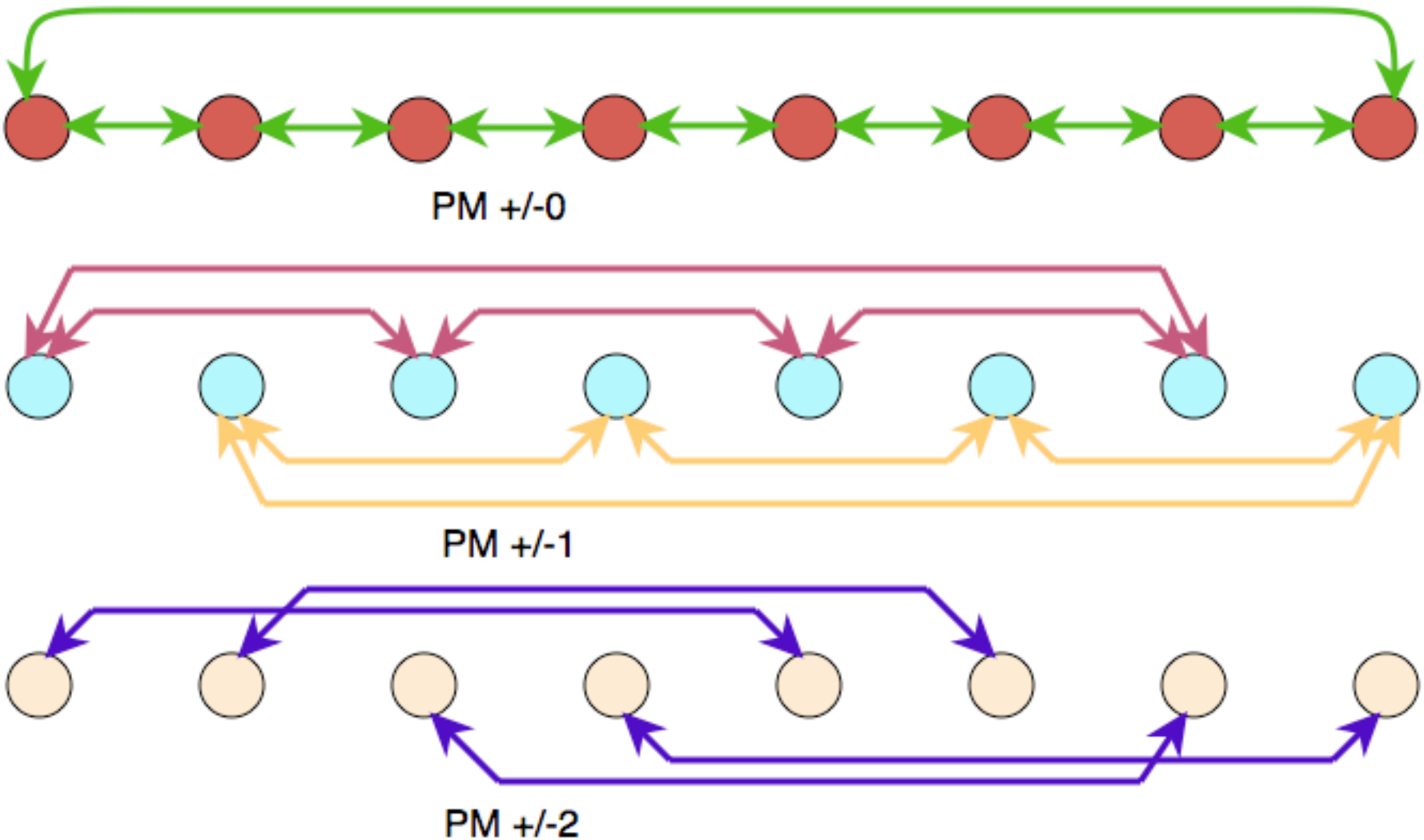
Shuffle-Exchange network

- Logarithmic cost
- Uni-directional shuffle network + bi-directional exchange network



Plus-Minus-Network

- PM 2i – 2*m-1 separate unidirectional interconnection networks



Comparison of networks

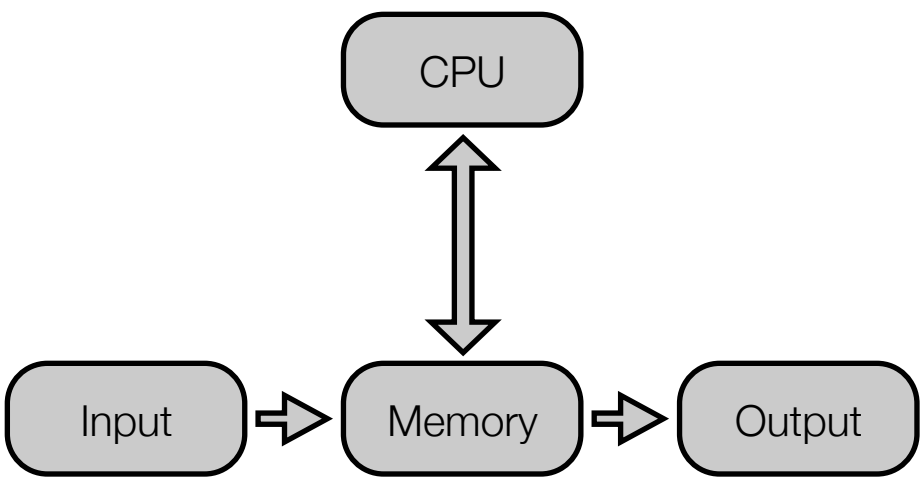
Netz1 simuliert Netz2 ->	Gitter(2D)	PM 2i	Shuffle-Exchange	Hypercube
Gitter(2D)	—	$\frac{\texttt{sqrt(2)}}{2}$	`sqrt(n)`	`sqrt(n)`
PM 2i	1	—	$\log_2 n$	2
Shuffle-Exchange	$2 * \log_2 n$	$2 * \log_2 n$	—	$\log_2 n + 1$
Hypercube	$\log_2 n$	$\log_2 n$	$\log_2 n$	—

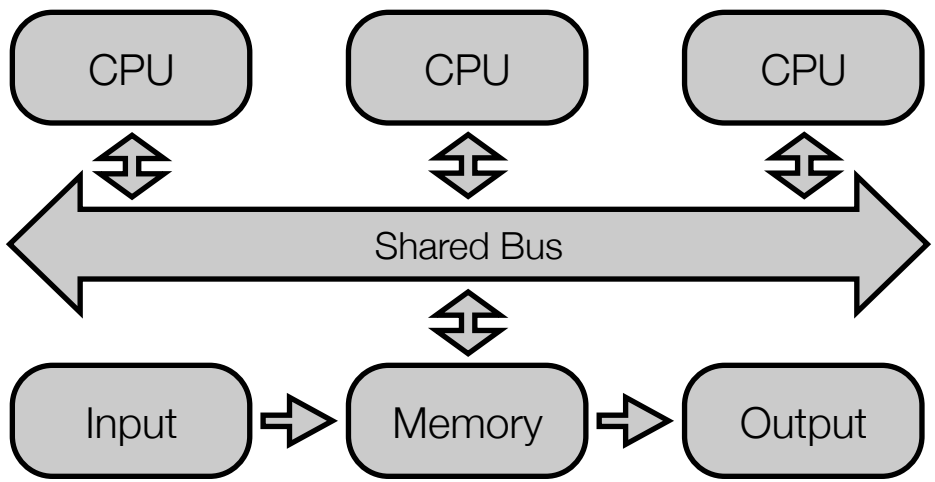
Theoretical Models for Parallel Computers

- Simplified parallel machine model, for theoretical investigation of algorithms
  - Difficult in the 70‘s and 80‘s due to large diversity in parallel hardware design
- Should improve algorithm robustness by avoiding optimizations to hardware layout specialities (e.g. network topology)
- Resulting computation model should be independent from programming model
- Vast body of theoretical research results
- Typically, formal models adopt to hardware developments

(Parallel) Random Access Machine

- RAM assumptions: Constant memory access time, unlimited memory
- PRAM assumptions: Non-conflicting shared bus, no assumption on synchronization support, unlimited number of processors
- Alternative models: BSP, LogP





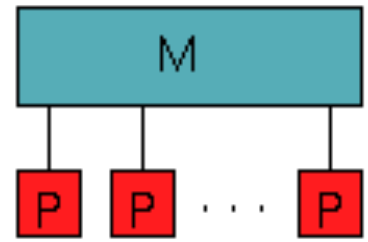
PRAM Extensions

- Rules for memory interaction to classify hardware support of a PRAM algorithm
- Note: Memory access assumed to be in lockstep (synchronous PRAM)
- *Concurrent Read, Concurrent Write (CRCW)*
  - Multiple tasks may read from / write to the same location at the same time
- *Concurrent Read, Exclusive Write (CREW)*
  - One thread may write to a given memory location at any time
- *Exclusive Read, Concurrent Write (ERCW)*
  - One thread may read from a given memory location at any time
- *Exclusive Read, Exclusive Write (EREW)*
  - One thread may read from / write to a memory location at any time

PRAM Extensions

- Concurrent write scenario needs further specification by algorithm
  - Ensures that the same value is written
  - Selection of arbitrary value from parallel write attempts
  - Priority of written value derived from processor ID
  - Store result of combining operation (e.g. sum) into memory location
- PRAM algorithm can act as starting point (unlimited resource assumption)
  - Map ‚logical‘ PRAM processors to restricted number of physical ones
  - Design scalable algorithm based on unlimited memory assumption, upper limit on real-world hardware execution
  - Focus only on concurrency, synchronization and communication later

PRAM



PRAM extensions

Exclusive-read, exclusive-write (EREW) PRAM

- exklusiver Zugang zu jedem Speicherwort
- schwächstes Modell: Speicherverwaltung muß Minimum an Nebenläufigkeit unterstützen

Concurrent-read, exclusive-write (CREW) PRAM

- mehrere simultane Lesezugriffe auf Speicherwort
- Schreibzugriffe serialisiert

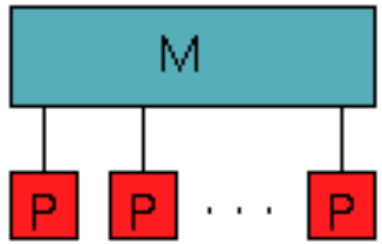
Exclusive-read, concurrent-write (ERCW) PRAM

- parallele Schreibzugriffe auf dasselbe Speicherwort
- Lesezugriffe serialisiert

Concurrent-read, concurrent-write (CRCW) PRAM

- mächtigstes Modell
- kann auf EREW simuliert werden

PRAM



Protokolle für parallele Schreibzugriffe

PRAM write operations

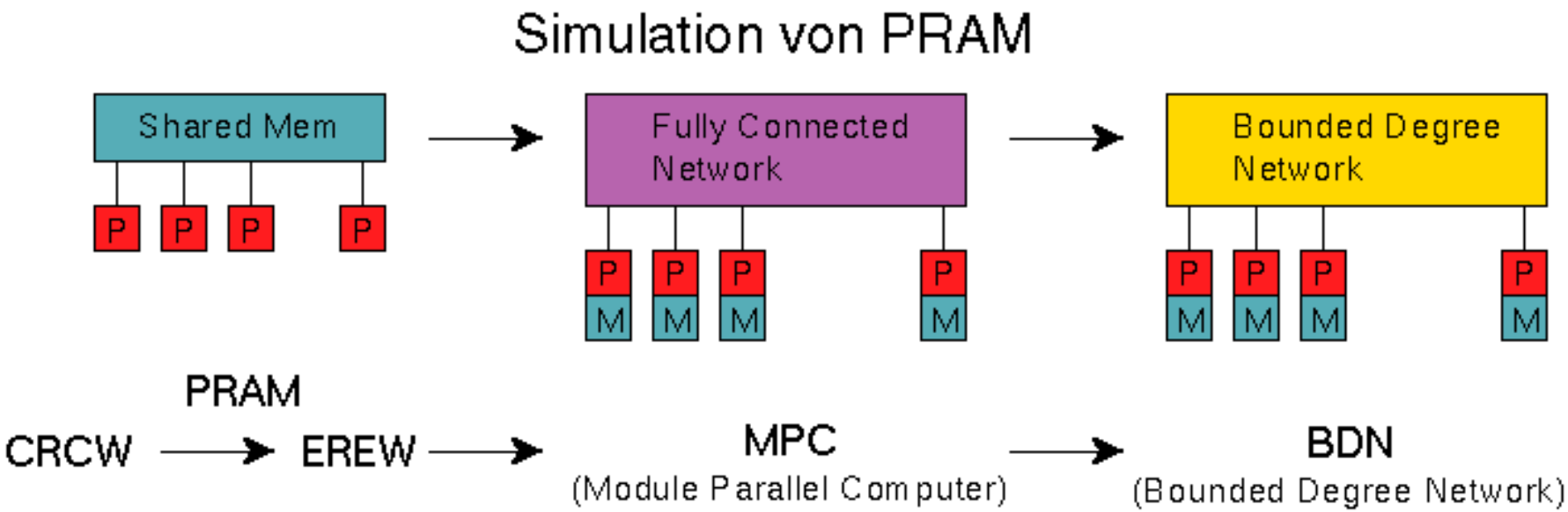
Common: nebenäufiges Schreiben ist erlaubt, wenn alle Prozessoren denselben Wert schreiben

Arbitrary: einer der Prozessoren kann erfolgreich schreiben, die anderen scheitern

Priority: Prozessoren haben vorherdefinierte Priorität, Prozessor mit höchstem Wert hat Erfolg, die anderen scheitern

Sum: Die Summe aller Werte wird geschrieben
(Combining: jeder assoziative Operator ist denkbar)

PRAM Simulation



	Deterministische Simulation	randomized Simulation
CRCW → EREW	O(log n)	
EREW → MPC	O(log n/log log n)	O(log log n log*n)
CRCW → MPC	O(log n)	O(log n)
MPC → BDN	$O(\log^2 n)$	O(log n)
EREW → BDN	$O(\log^2 n/\log \log n)$	O(log n)
CRCW → BDN	$O(\log^2 n/\log \log n)$	O(log n)

Tim J. Harris, *A Survey of PRAM Simulation Techniques*, ACM Comp. Surveys, Vol. 26, No. 2, June 1994

Example: Parallel Sum

- General parallel sum operation works with any associative and commutative combining operation (multiplication, maximum, minimum, logical operations, ...)
  - Typical reduction pattern
- PRAM solution: Build binary tree, with input data items as leaf nodes
  - Internal nodes hold the sum, root node as global sum
  - Additions on one level are independent from each other
  - PRAM algorithm: One processor per leaf node, in-place summation
  - Computation in $O(\log_2 n)$

```
int sum=0;
for (int i=0; i<N; i++) {
   sum += A[i];
}
```

Example: Parallel Sum

```
for all l levels (1..log2n){
  for all i items (0..n-1) {
    if (((i+1) mod 2^l) = 0) then
      X[i] := X[i-2^(l-1)]+X[i]
  }
}
```

- Example: n=8:
  - l=1: Partial sums in X[1], X[3], X[5], [7]
  - l=2: Partial sums in X[3] and X[7]
  - l=3: Parallel sum result in X[7]
- Correctness relies on PRAM lockstep assumption (no synchronization)

Bulk-Synchronous Parallel (BSP) Model

- *Leslie G. Valiant. A Bridging Model for Parallel Computation, 1990*
- Success of von Neumann model
  - Bridge between hardware and software
  - High-level languages can be efficiently compiled based on this model
  - Hardware designers can optimize the realization of this model
- Similar model for parallel machines
  - Should be neutral about the number of processors
  - Program are written for *v* virtual processors that are mapped to *p* physical ones, were *v* >> *p* -> chance for the compiler

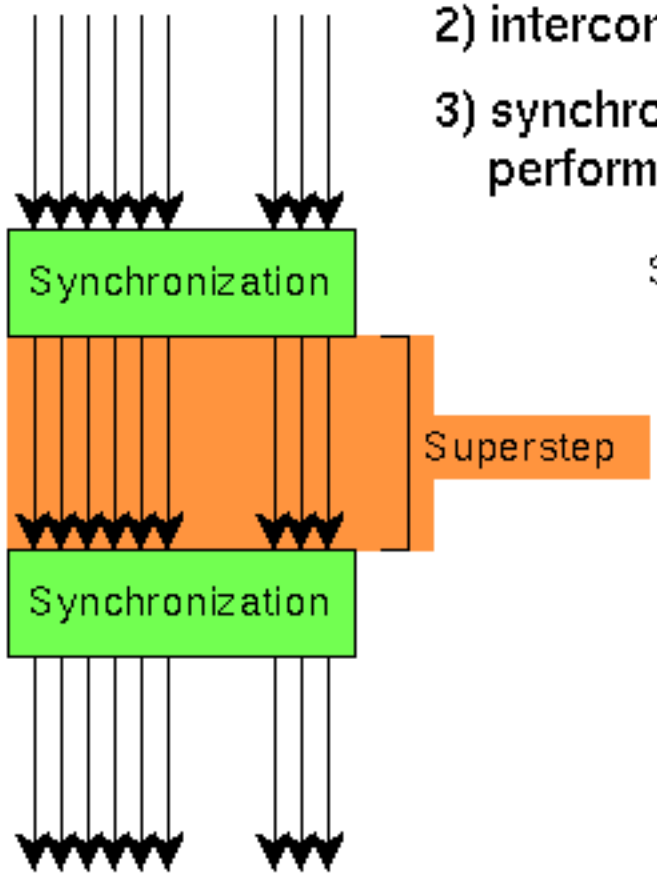
BSP

Bulk Synchronous Parallel Model

1) processor/memory modules

2) interconnection network

3) synchronizer,
   performs barrier synchronization



Superstep: - Prozessor vollführt lokale Berechnungen
- empfängt/sendet Nachrichten

jeder Prozessor kann höchstens h-Nachrichten empfangen und h-Nachrichten senden
(h - relation)

L.G.Valiant, *A Bridging Model for Parallel Computation*, CACM, 33(8), 103-11, August 1990

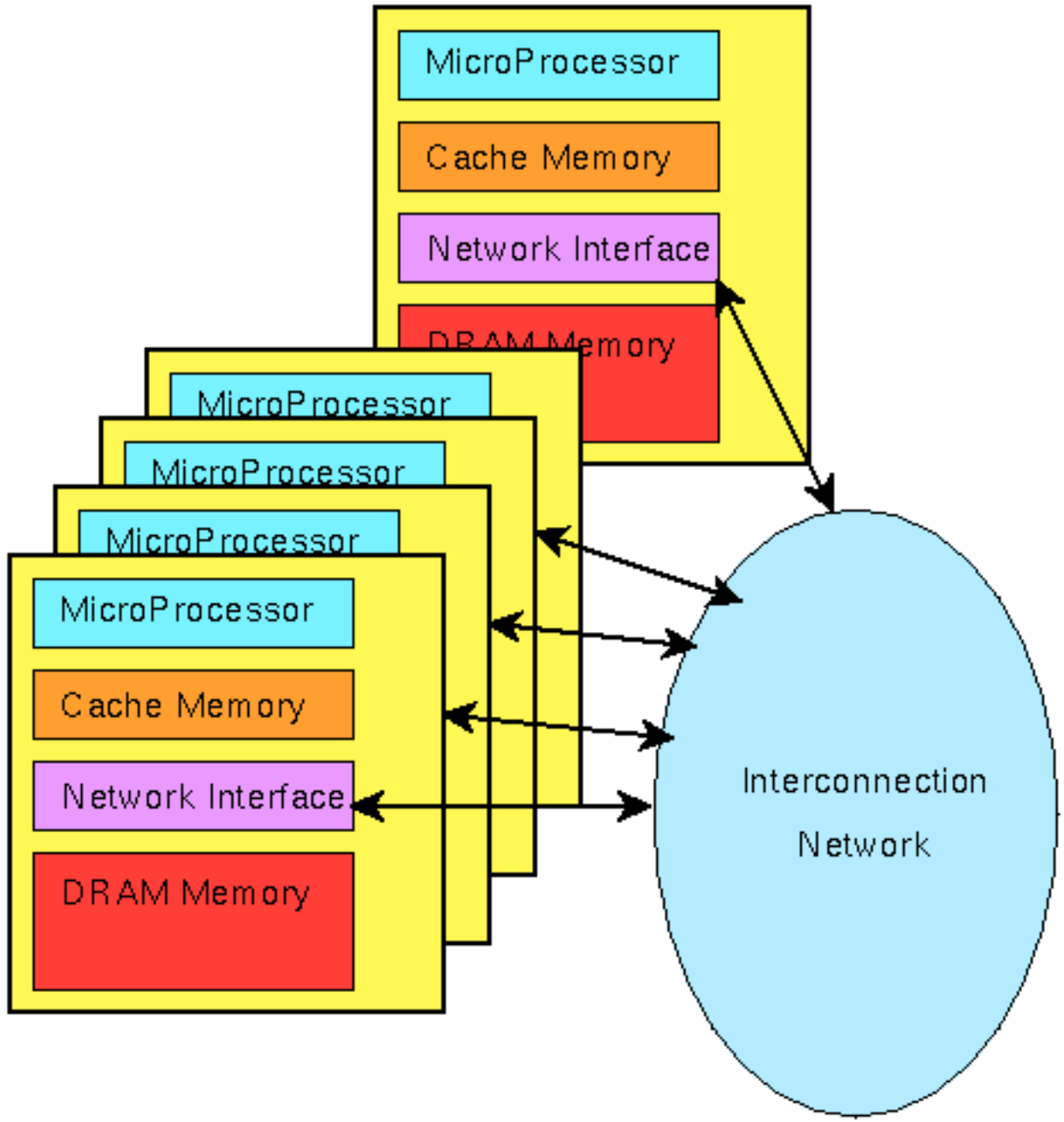
Bulk-Synchronous Parallel (BSP) Model

- Bulk-synchronous parallel computer (BSPC) is defined by:
  - *Components*, each performing processing and / or memory functions
  - *Router* that delivers messages between pairs of components
  - Facilities to synchronize components at regular intervals *L (periodicity)*
- Computation consists of a number of *supersteps*
  - Each L, global check is made if the *superstep* is completed
- Router concept splits computation vs. communication aspects, and models memory / storage access explicitely
- Synchronization may only happen for some components, so long-running serial tasks are not slowed down from model perspective
- *L* is controlled by the application, even at run-time

LogP

- *Culler et al., LogP: Towards a Realistic Model of Parallel Computation, 1993*
- Criticism on overly simplification in PRAM-based approaches, encourage exploitation of ‚formal loopholes' (e.g. no communication penalty)
- Trend towards multicomputer systems with large local memories
- Characterization of a parallel machine by:
  - **P**: Number of processors
  - **g**: Gap: Minimum time between two consecutive transmissions
    - Reciprocal corresponds to per-processor communication bandwidth
  - **L**: Latency: Upper bound on messaging time from source to target
  - **o**: Overhead: Exclusive processor time needed for send / receive operation
- L, o, G in multiples of processor cycles

LogP architecture model



Architectures that map well on LogP:

—Intel iPSC, Delta, Paragon,

—Thinking Machines CM-5, Ncube,

—Cray T3D,

—Transputer MPPs: MeikoComputing Surface, Parsytec GC.

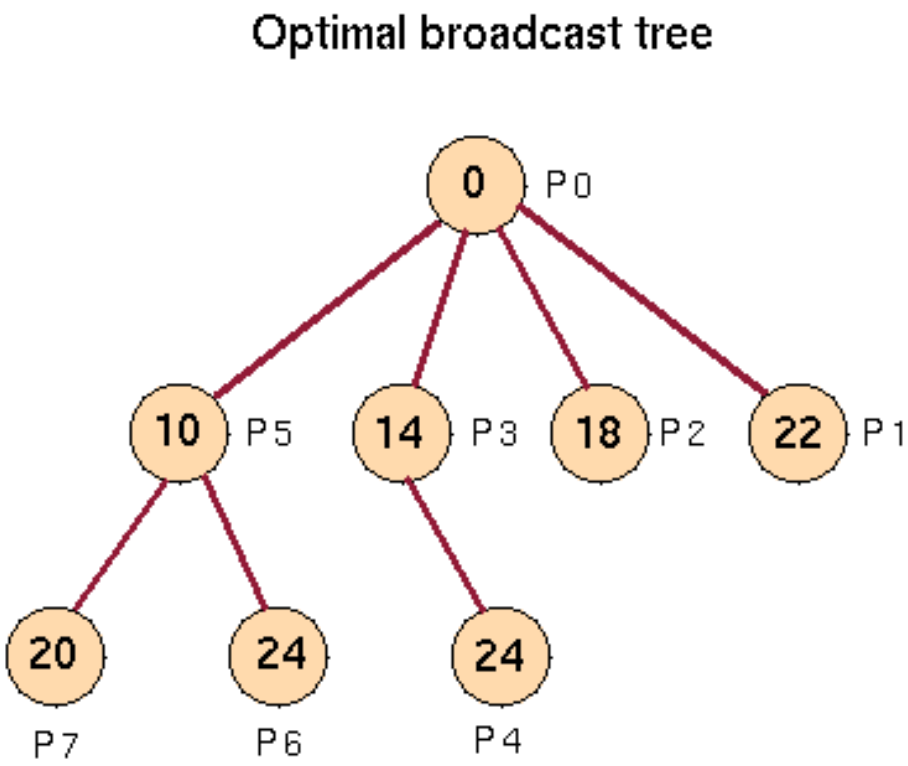
LogP

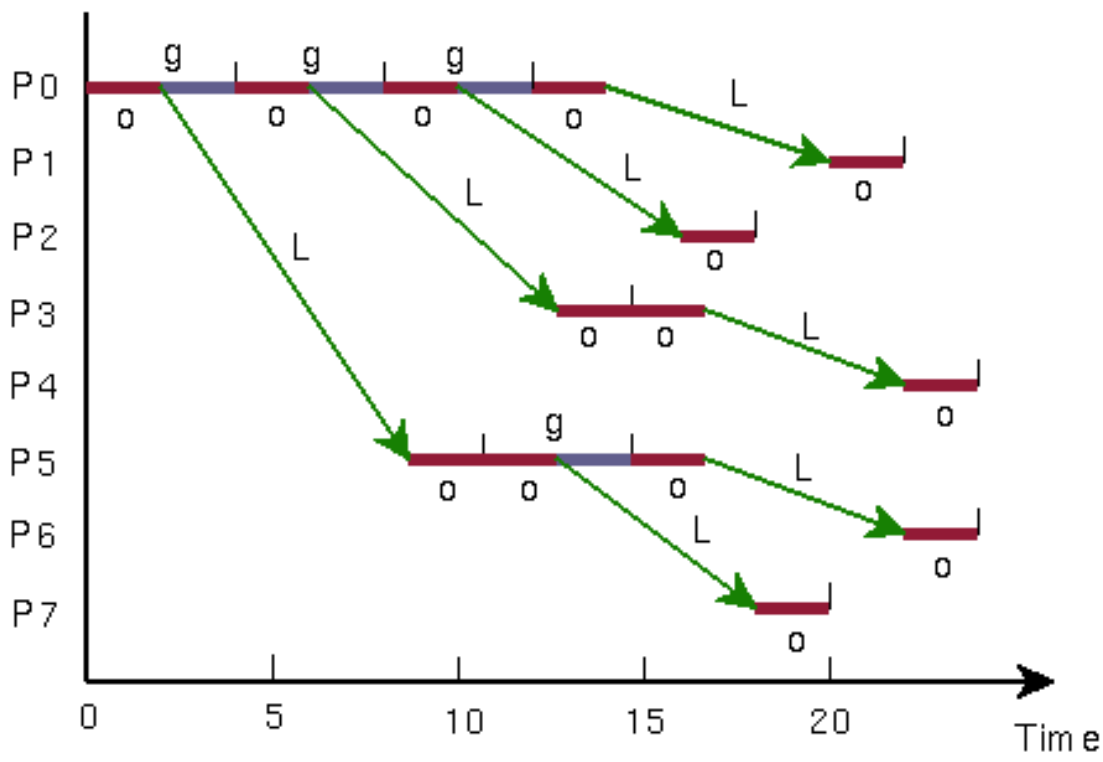
- Analyzing an algorithm - must produce correct results under all message interleaving, prove space and time demands of processors
- Simplifications
  - With infrequent communication, bandwidth limits (g) are not relevant
  - With streaming communication, latency (L) may be disregarded
- Convenient approximation: Increase overhead (o) to be as large as gap (g)
- Encourages careful scheduling of computation, and overlapping of computation and communication
- Can be mapped to shared-memory architectures
  - Reading a remote location requires *2L+4o* processor cycles

LogP

- Matching the model to real machines
  - Saturation effects: Latency increases as function of the network load, sharp increase at saturation point - captured by capacity constraint
  - Internal network structure is abstracted, so ‚good' vs. ‚bad' communication patterns are not distinguished - can be modeled by multiple *g's*
  - LogP does not model specialized hardware communication primitives, all mapped to send / receive operations
  - Separate network processors can be explicitly modeled
- Model defines 4-dimensional parameter space of possible machines
  - Vendor product line can be identified by a curve in this space

LogP – optimal broadcast tree





LogP – optimal summation