UiT

THE ARCTIC
UNIVERSITY
OF NORWAY

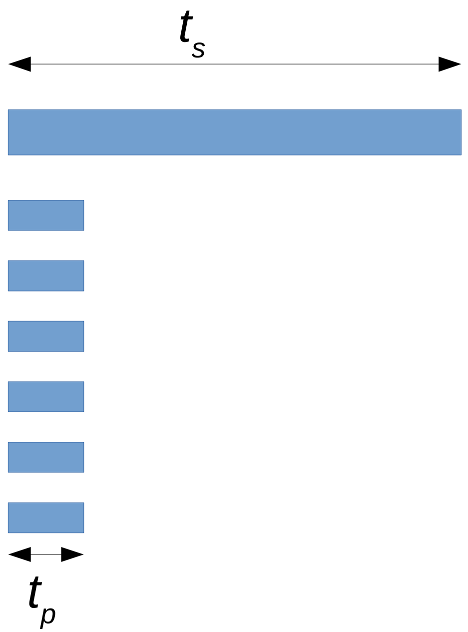
Lecture 2: Parallel Computers

Parallell Programming (INF-3201)

John Markus Bjørndalen

Speedup Factor

$$S(p) = \frac{Execution\, time\, using\, one\, processor\,(best\, sequential\, algorithm)}{Execution\, time\, using\, multiprocessor\, with\, p\, processors} = \frac{t_s}{t_p}$$



$S(p)$: increase in speed by using multiprocessor.

Use best sequential algorithm with single processor system.

Underlying algorithm for parallel implementation is usually different.

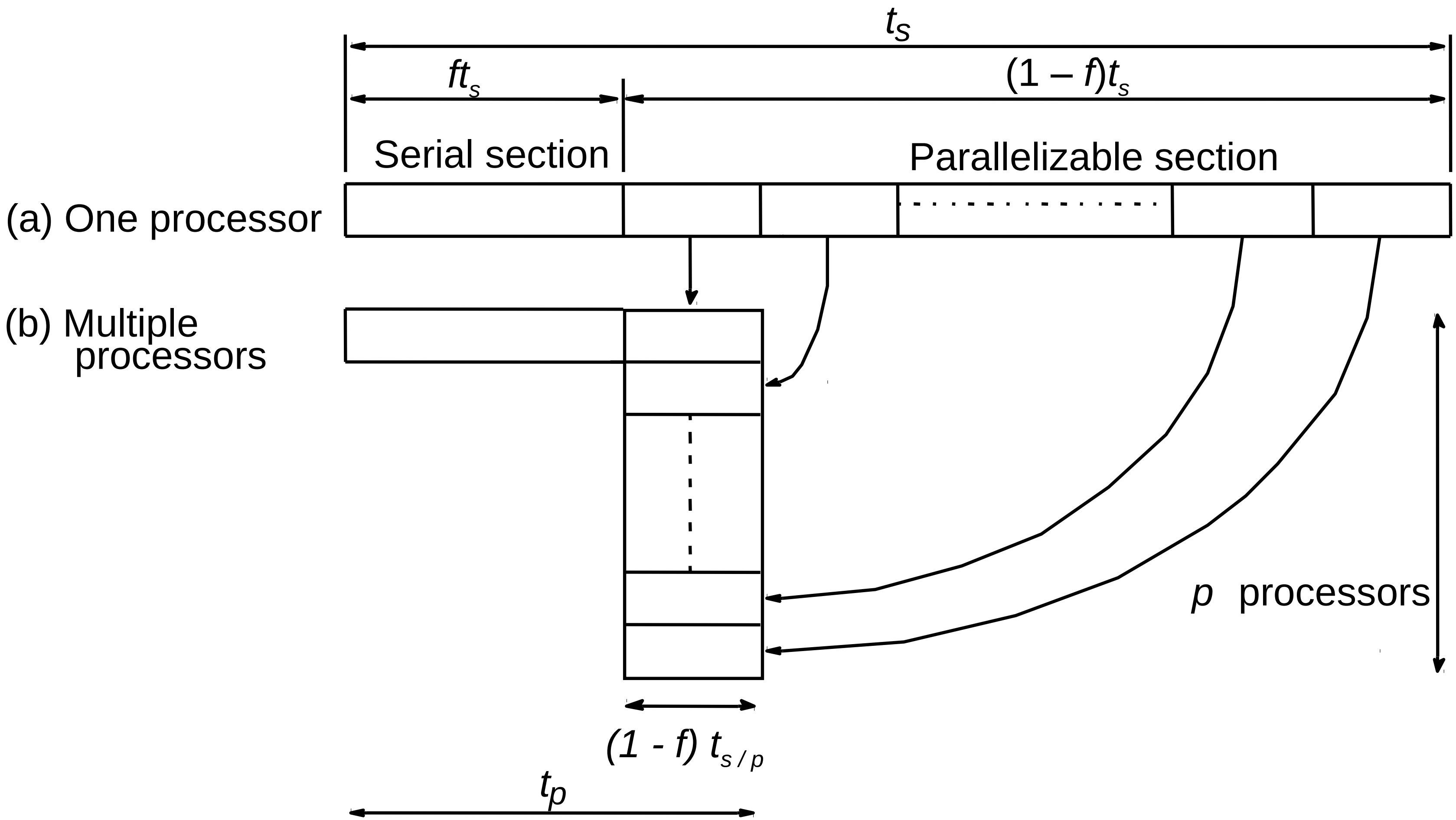
Speedup Factor

Can also be computed in terms of computational steps as (sequential and parallel) time complexity.

Ignore communication overhead in parallel implementation

$$S(p) = \frac{\text{Number of computational steps using one processor}}{\text{Number of parallel computational steps with } p \text{ processors}}$$

Maximum Speedup - Amdahl's law



f = sequential fraction

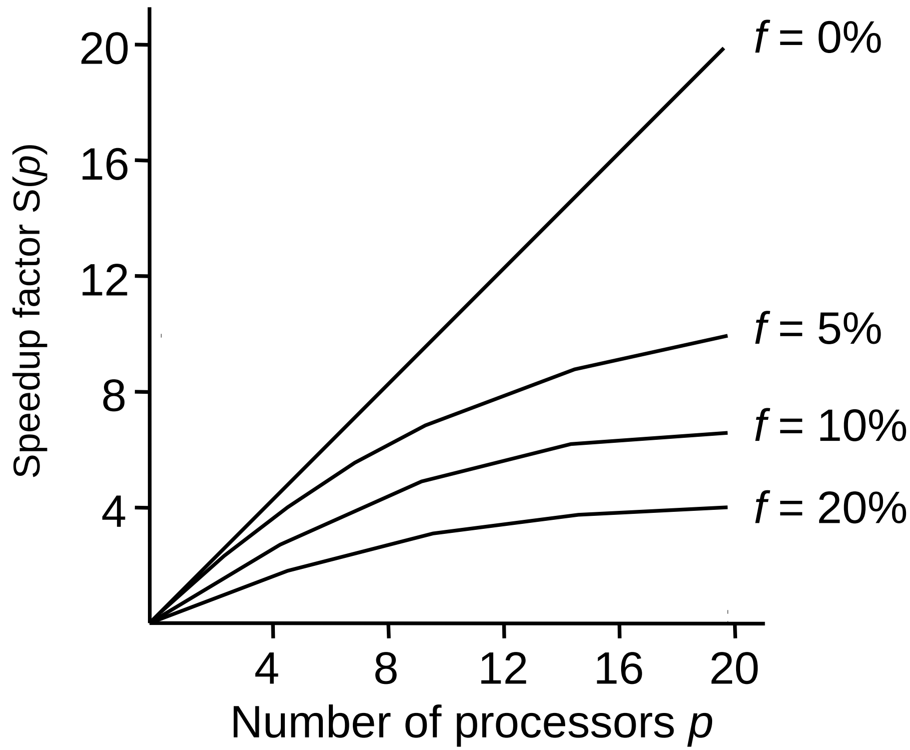
Maximum Speedup - Amdahl's law

Speedup factor is given by:

$$S(p) = \frac{t_s}{ft_s + (1 - f)t_s/p} = \frac{p}{1 + (p - 1)f}$$

This equation is known as Amdahl's law

Speedup against number of processors



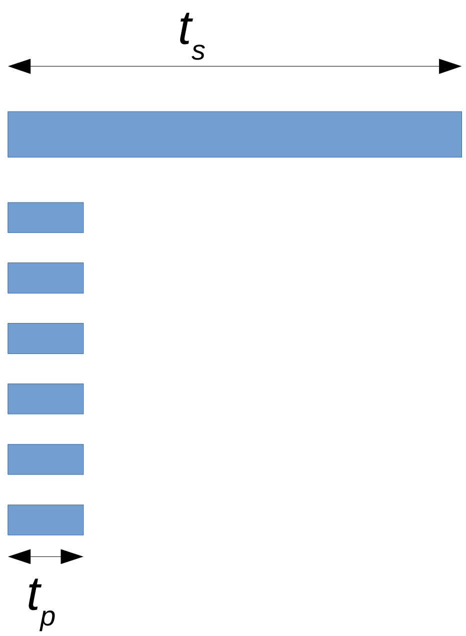
Maximum speedup

$$S(p) = \frac{t_s}{ft_s + (1-f)t_s/p} = \frac{p}{1+(p-1)f}$$

$$S(p)_{p\rightarrow\infty} = \frac{1}{f}$$

Example: f = 5% $\Rightarrow$ maximum S(p) = 20

Maximum Speedup



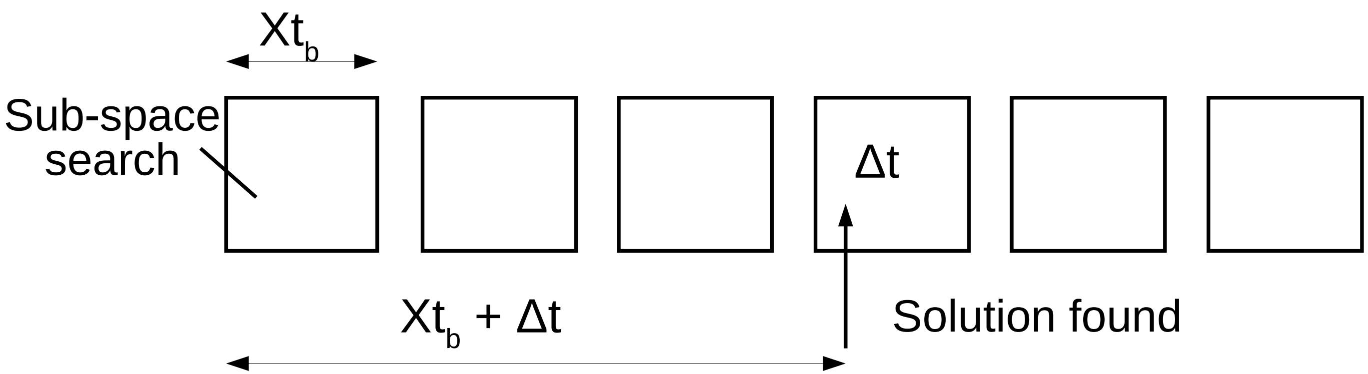
Maximum speedup: usually *p* with *p* processors (linear speedup).

Is it possible to get superlinear speedup (greater than *p*)?

- Yes, but usually a specific reason such as:
  - Extra memory in multiprocessor system
  - Nondeterministic algorithm

Superlinear Speedup example - Searching

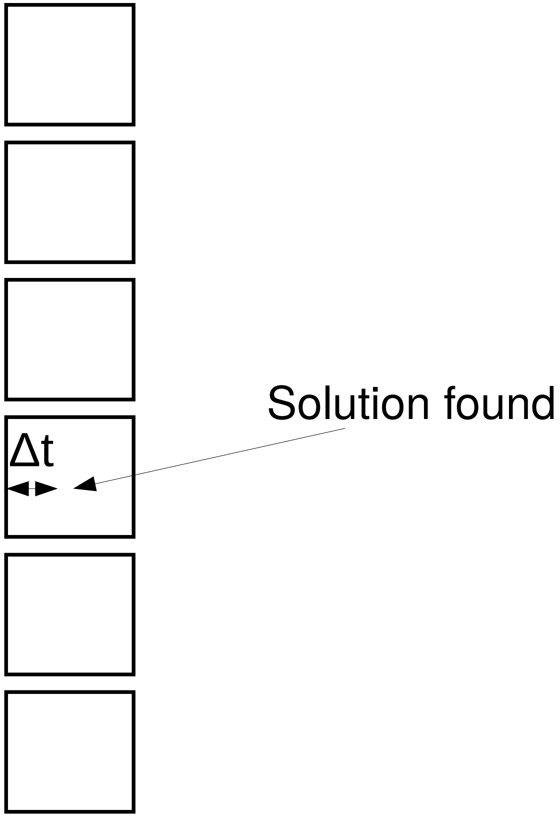
Sequential:



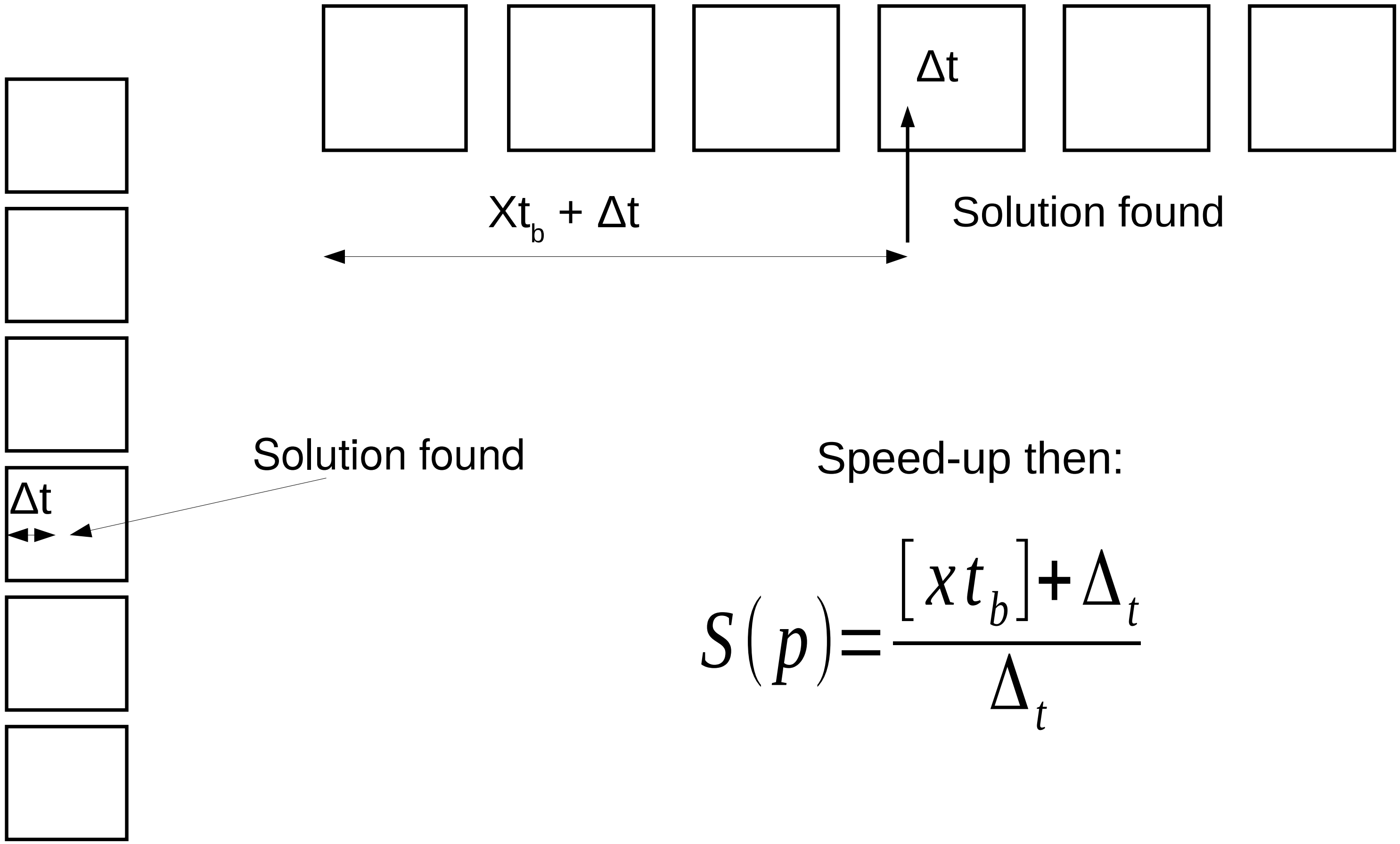
X = number of blocks searched before block with solution

Superlinear Speedup example - Searching

Parallel



Superlinear Speedup example - Searching



Speed-up then:

$$S(p) = \frac{[x t_b] + \Delta_t}{\Delta_t}$$

NB: the book uses $t_b = t_s/p$

Efficiency

- How efficiently processors are being used on the computation (instead of idling, overheads etc)

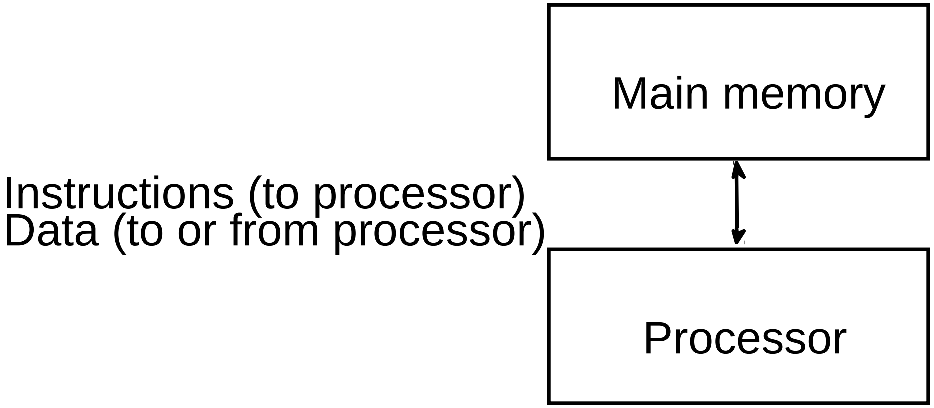
$$E = \frac{\text{Execution time using one processor}}{\text{Execution time using a multiprocessor x number of processors}}$$

$$E = \frac{t_s}{t_p \times p} = \frac{S(p)}{p} \times 100\%$$

- E = 100% $\Leftrightarrow$ S(p) = p

Conventional Computer

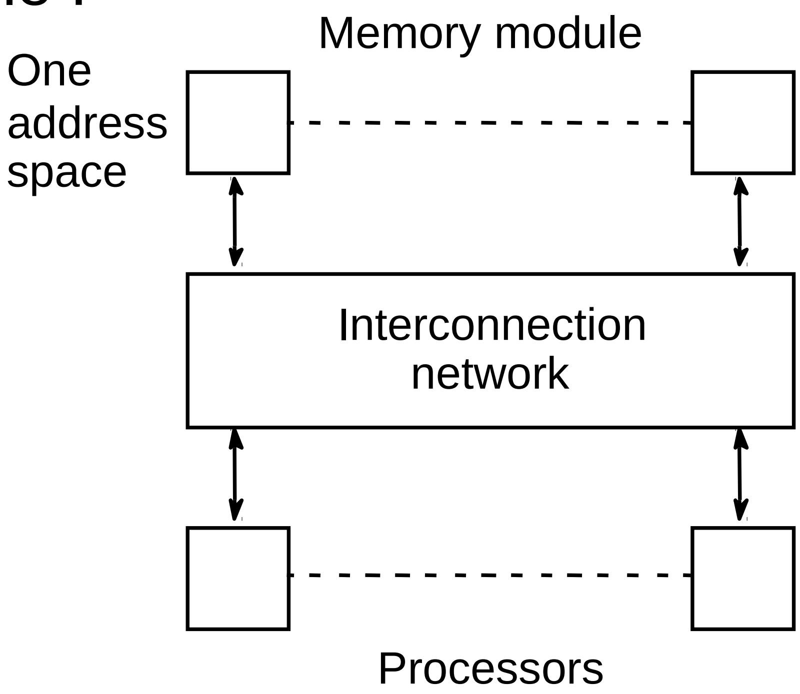
Consists of a processor executing a program stored in a (main) memory:



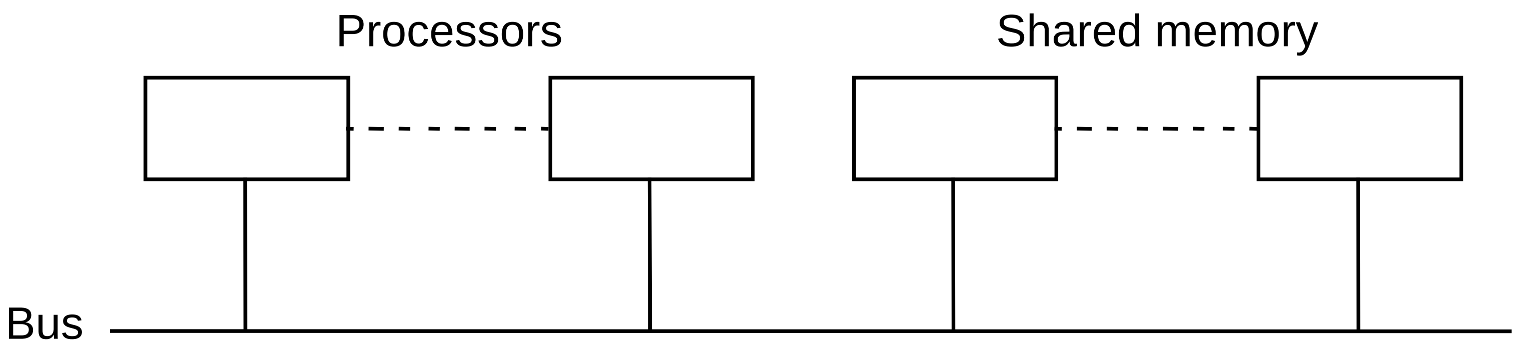
Each main memory location located by its address. Addresses start at 0 and extend to 2^{b-1} when there are *b* bits (binary digits) in address.

Shared Memory Multiprocessor System

Natural way to extend single processor model - have multiple processors connected to multiple memory modules, such that each processor can access any memory module :

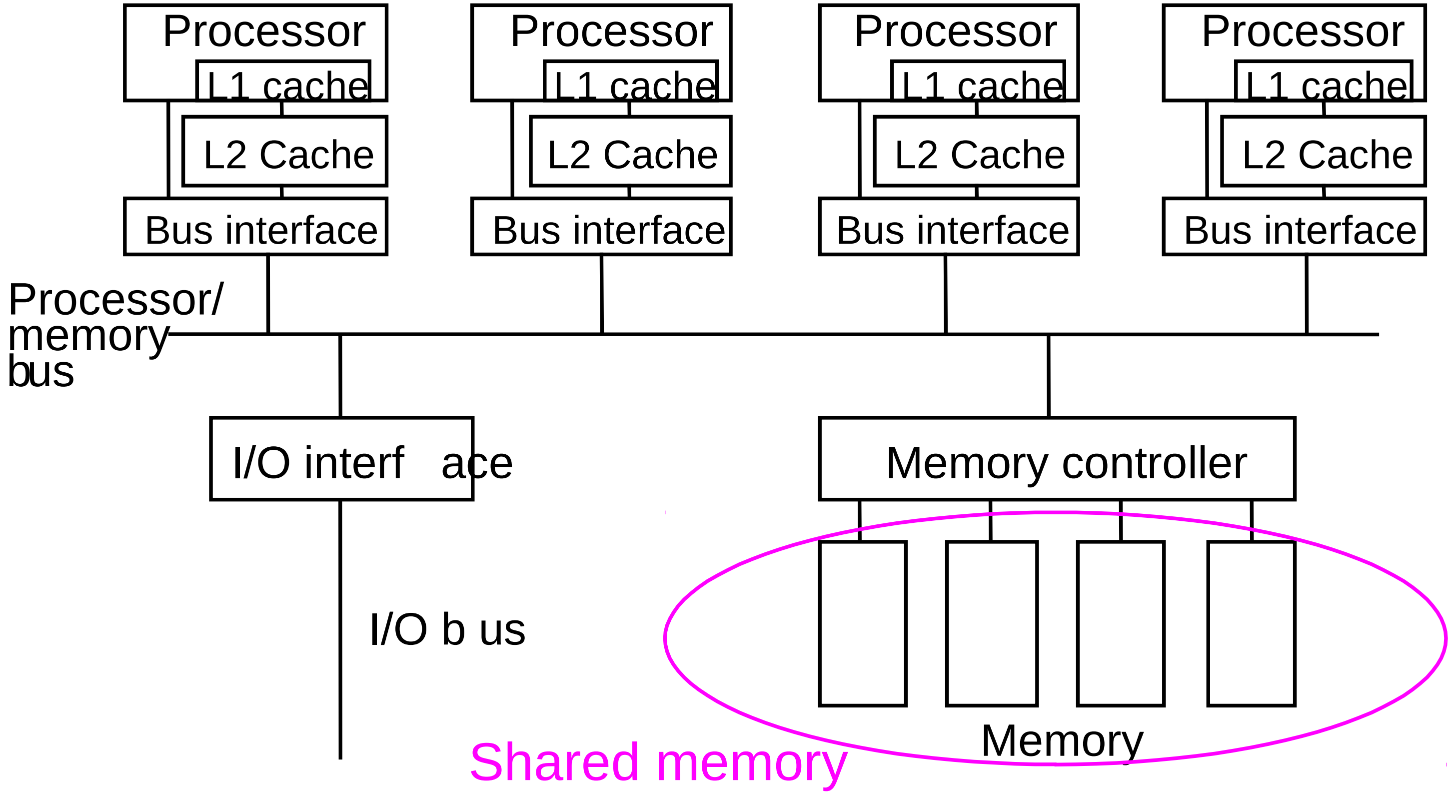


Simplistic view of a small shared memory multiprocessor



Quad Pentium Shared Memory Multiprocessor

NB: not quad-core!

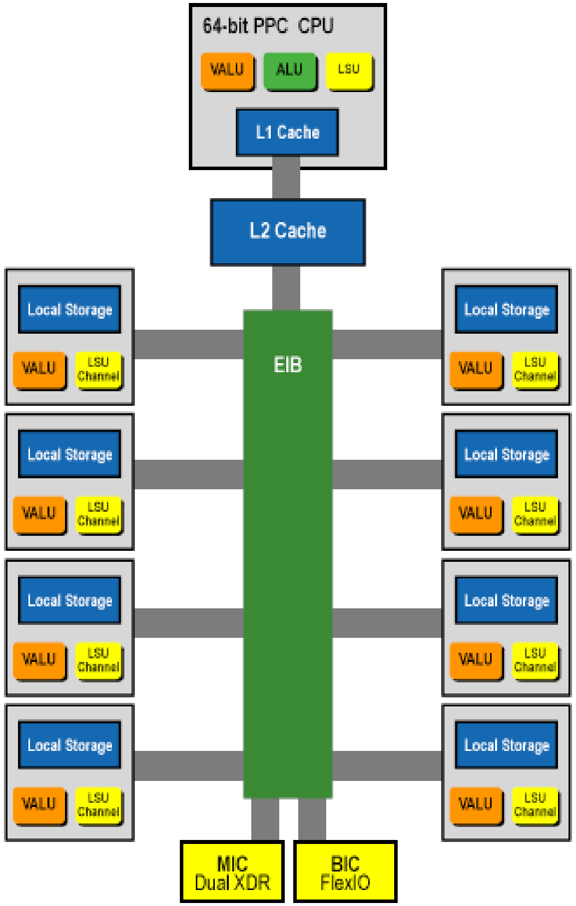


Uniform Memory Access (UMA)

Heterogenous architectures

Example: IBM Cell Architecture

Used in Playstations and some supercomputing areas.

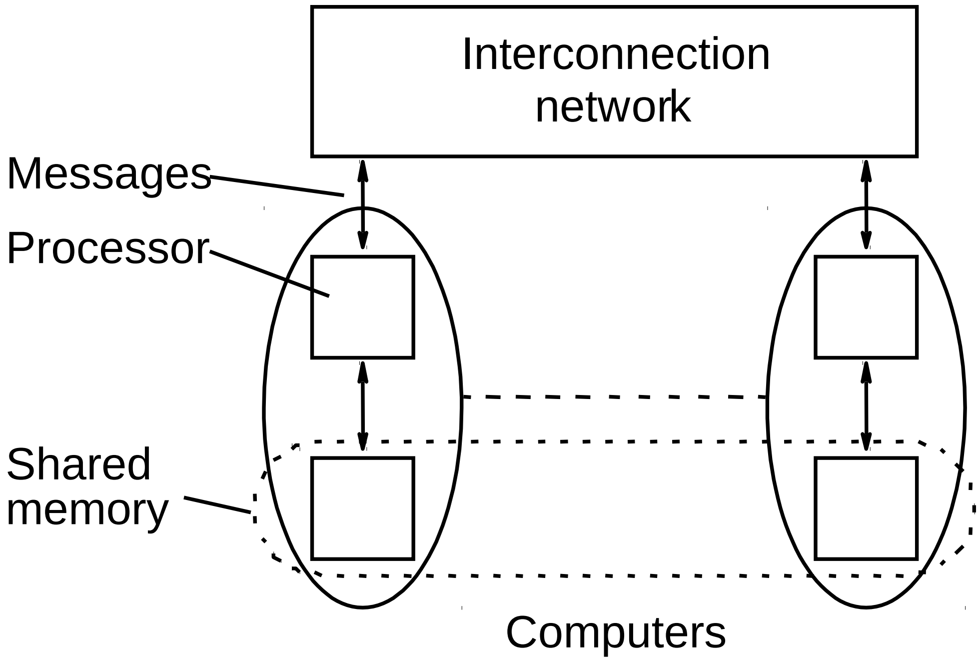


The CELL Architecture

Figure from http://arstechnica.com/features/2005/02/cell-2/

Distributed Shared Memory

Making main memory of group of interconnected computers look like a single memory with single address space. Then can use shared memory programming techniques.



Programming Shared Memory Multiprocessors

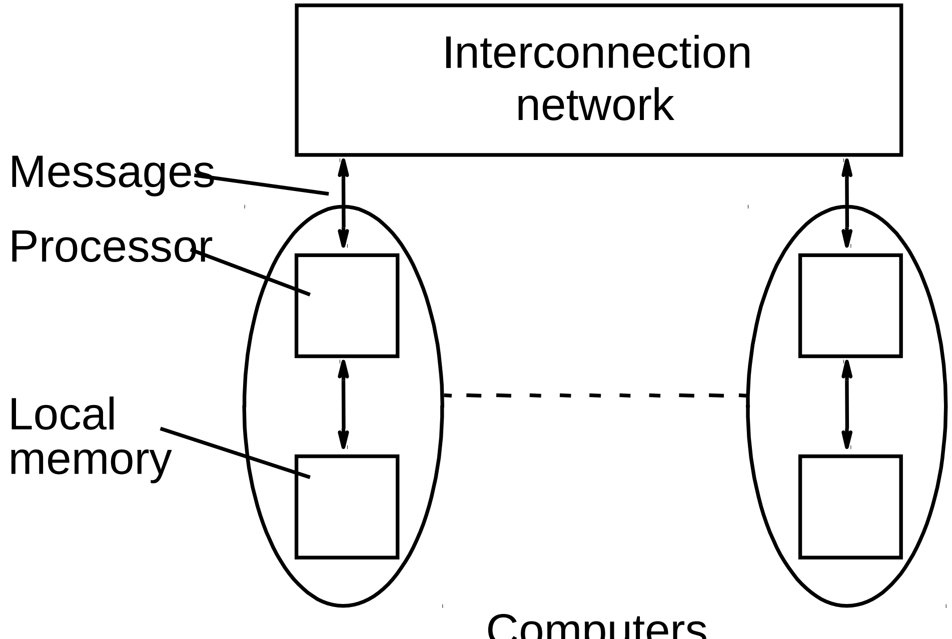
- Threads - programmer decomposes program into individual parallel sequences, (threads), each being able to access variables declared outside threads.
  - Example: Pthreads
- Sequential programming language with added syntax to declare shared variables and specify parallelism.
  - Example: Intel Cilk++
- Sequential programming language with preprocessor compiler directives to declare shared variables and specify parallelism.
  - Example: OpenMP

Programming Shared Memory Multiprocessors

- Languages – languages especially designed for concurrency or parallel programming
  - Example: Occam-pi, Go
- Libraries
  - Example: Intel TBB (template library), new concurrency mechanisms in Java

Message-Passing Multicomputer

Complete computers connected through an interconnection network:



Key concepts and issues

- Concepts
  - Bandwidth: data rate (bits/sec)
  - Throughput: average rate sucessfully transmitted.
  - Latency:
    - Network latency: time to make a message transfer through the network
    - Communication latency: total time to send a message, including software overhead and interface delays
    - Message latency: time to send a zero-length message
  - Diameter: minimum number of links between the two farthest nodes in the network

    $\Rightarrow$ Maximum distance a message must travel $\Rightarrow$ network latency
- Issues in network design:
  - Bandwidth, latency and cost.

Interconnection Networks

- With direct links between computers/processors
  - Exhaustive interconnections
  - 2- and 3-dimensional meshes
  - Hypercube

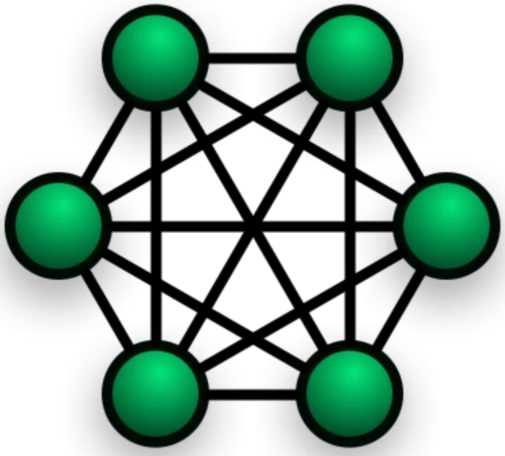
- Using switches:
  - Crossbar
  - Trees
  - Multistage interconnection networks

Exhaustive interconnections

- connect every computer to every other computers with links
  - fully connected network
- Latency: diameter =
- Cost: # links for c computers ?
  - c(c-1)/2

1

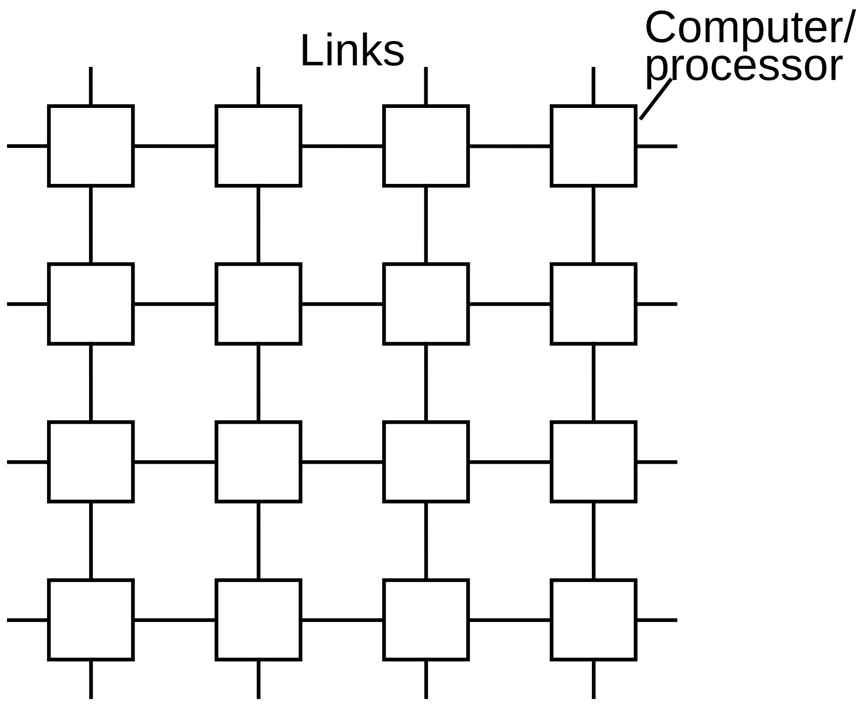
⇒ suitable only for a very small system



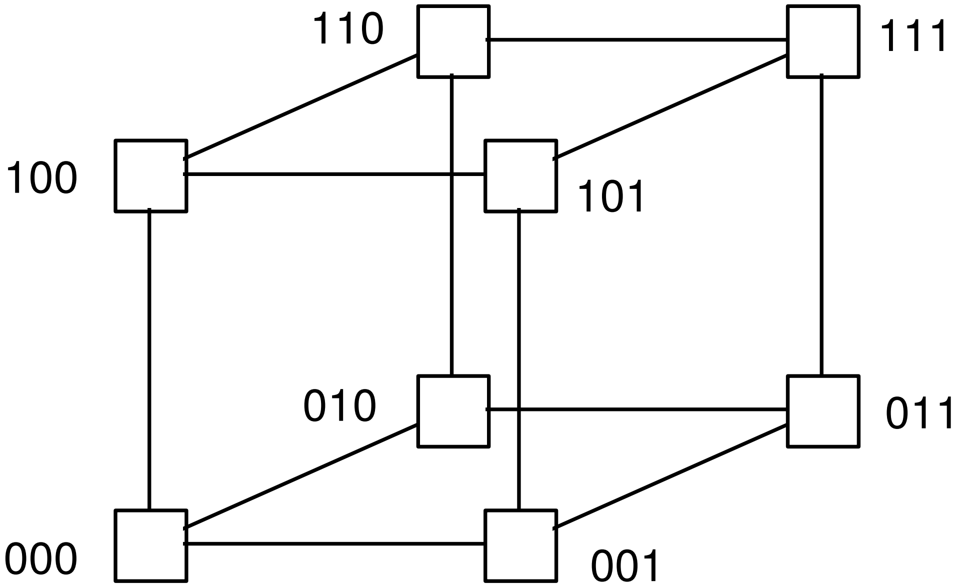
Two-dimensional array (mesh)

For p^2 processors , diameter = 2 (p - 1)

Also three-dimensional - used in some large high performance systems.

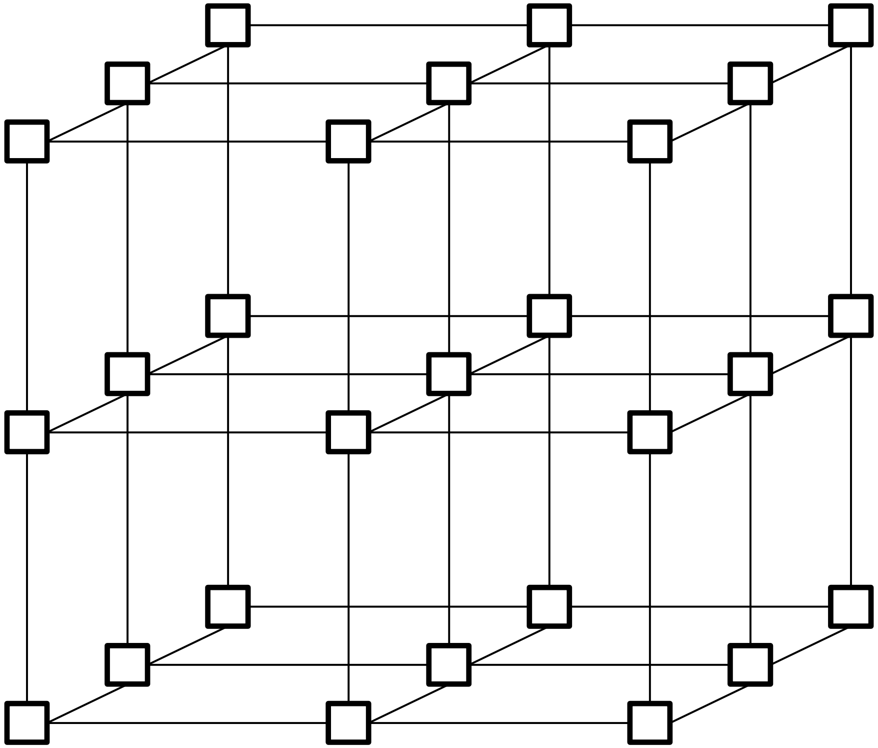


Three-dimensional hypercube



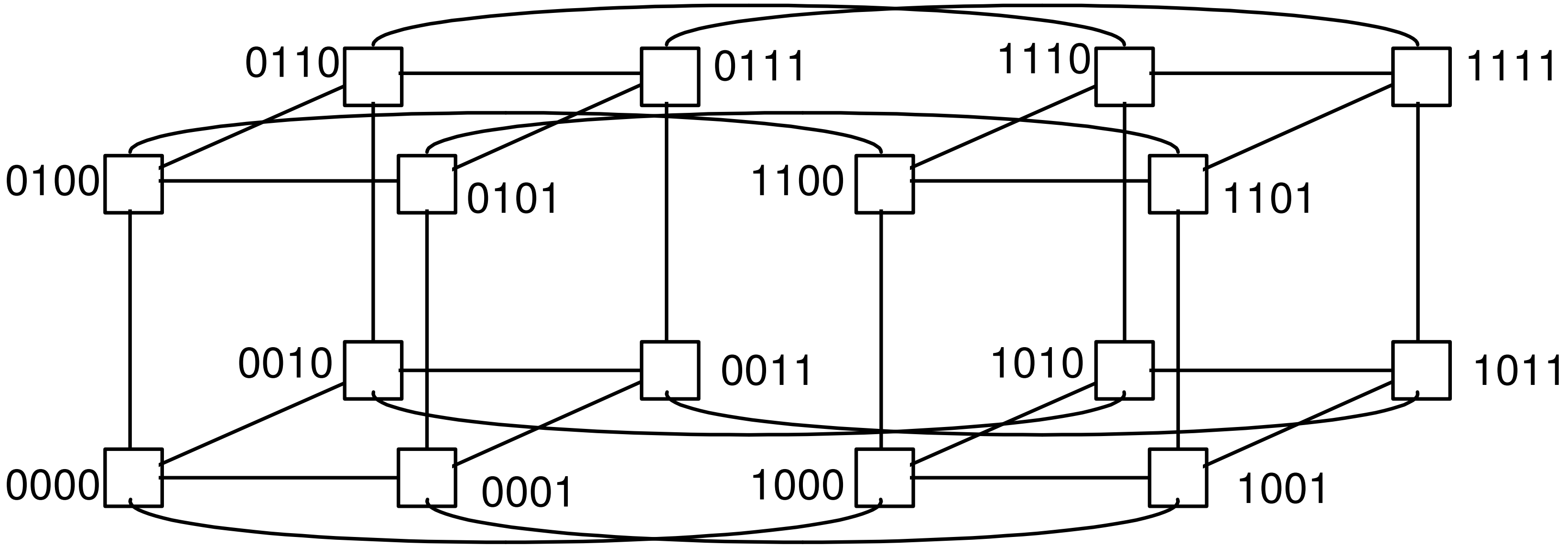
For p processors, diameter = $\log_2 p$

3D-hypercube or 3D-mesh ?

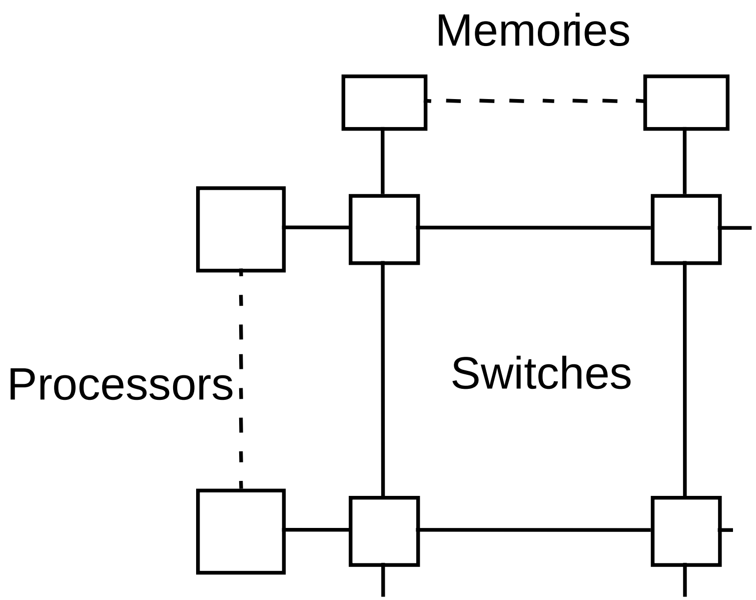


For p^3 processors, diameter = 3(p-1)

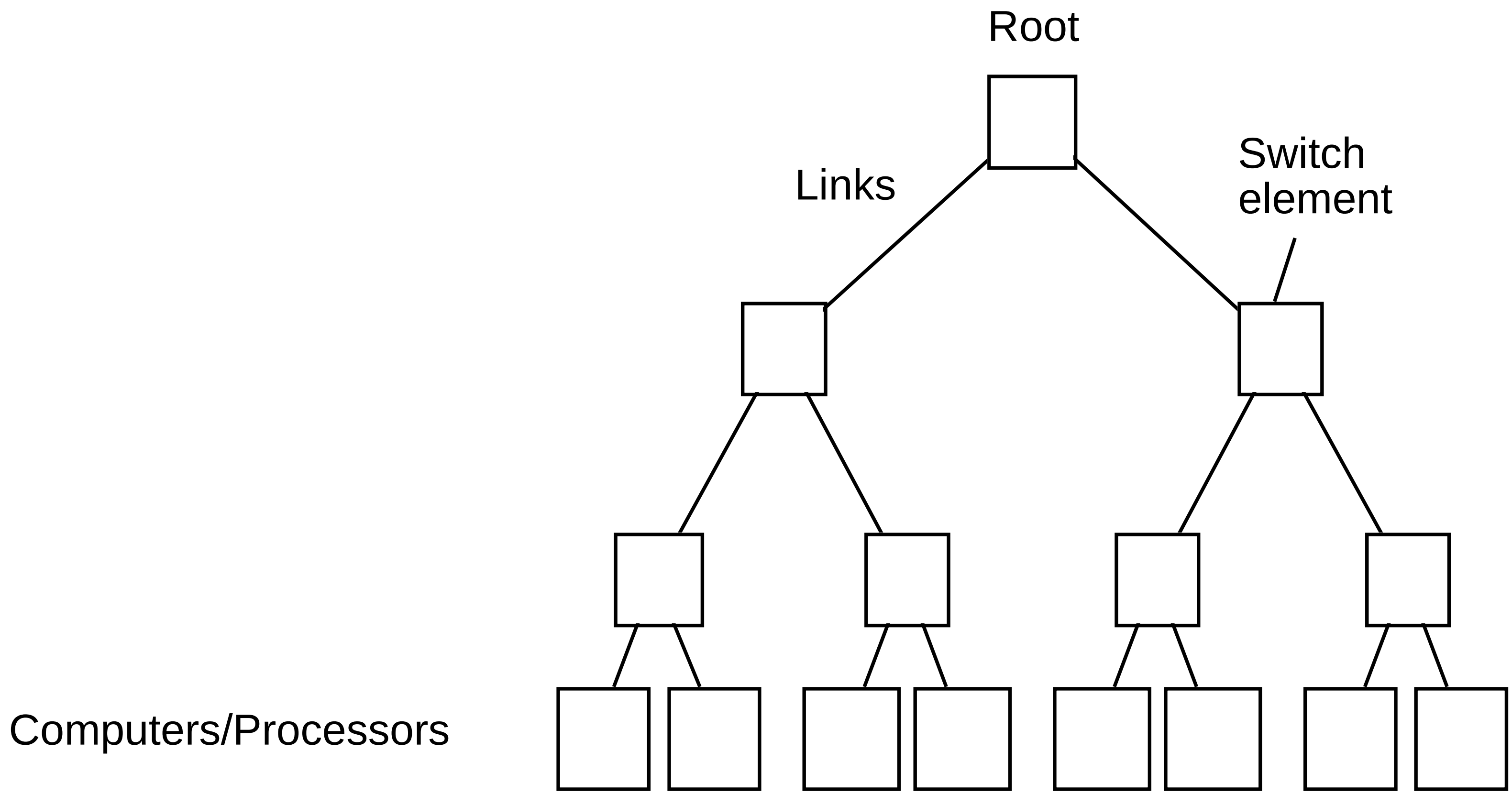
Four-dimensional hypercube



Crossbar switch



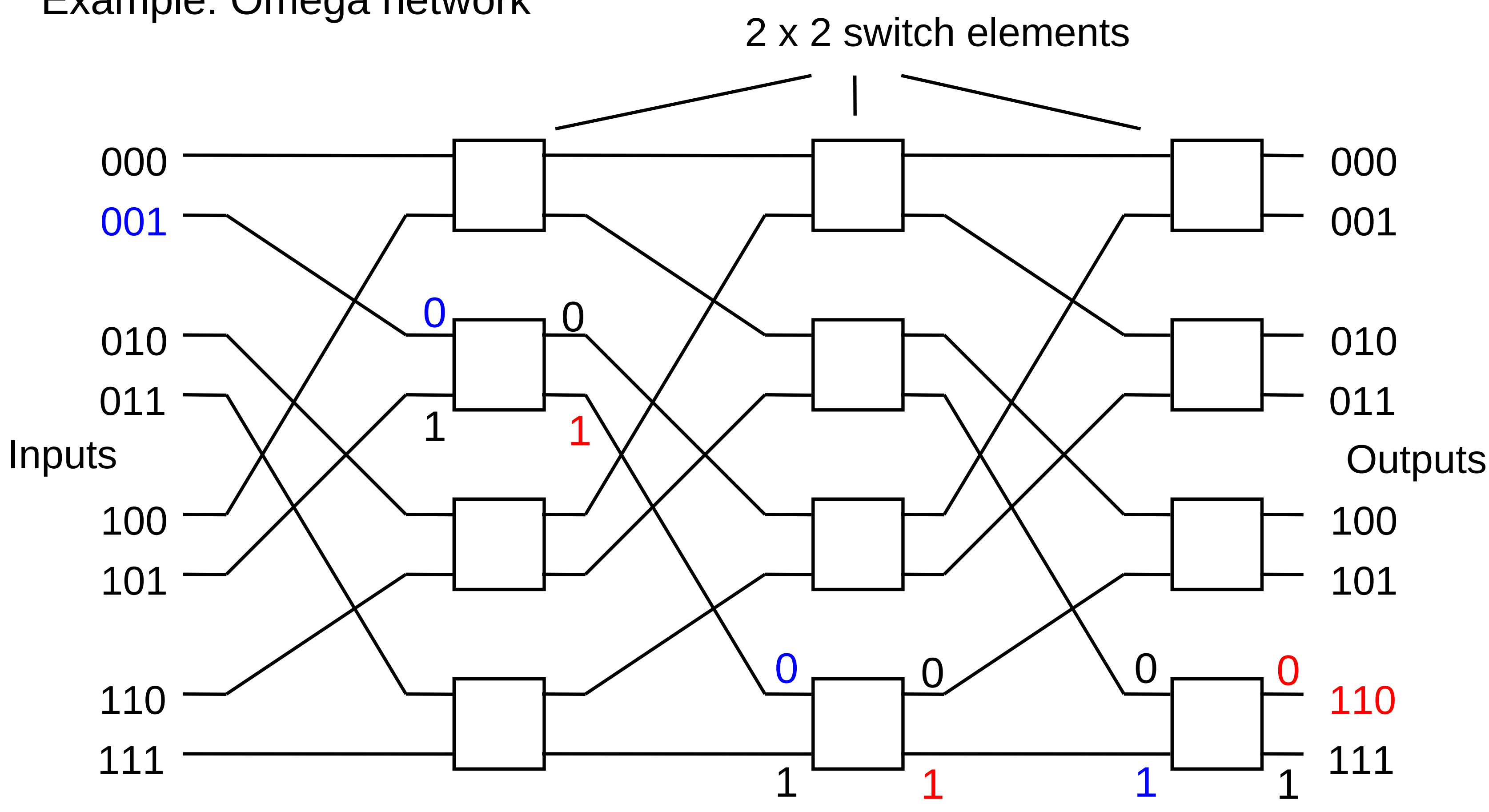
Tree



- Complete binary tree
  - p processors $\Rightarrow$ height = $\log_2 p$

Multistage Interconnection Network

Example: Omega network

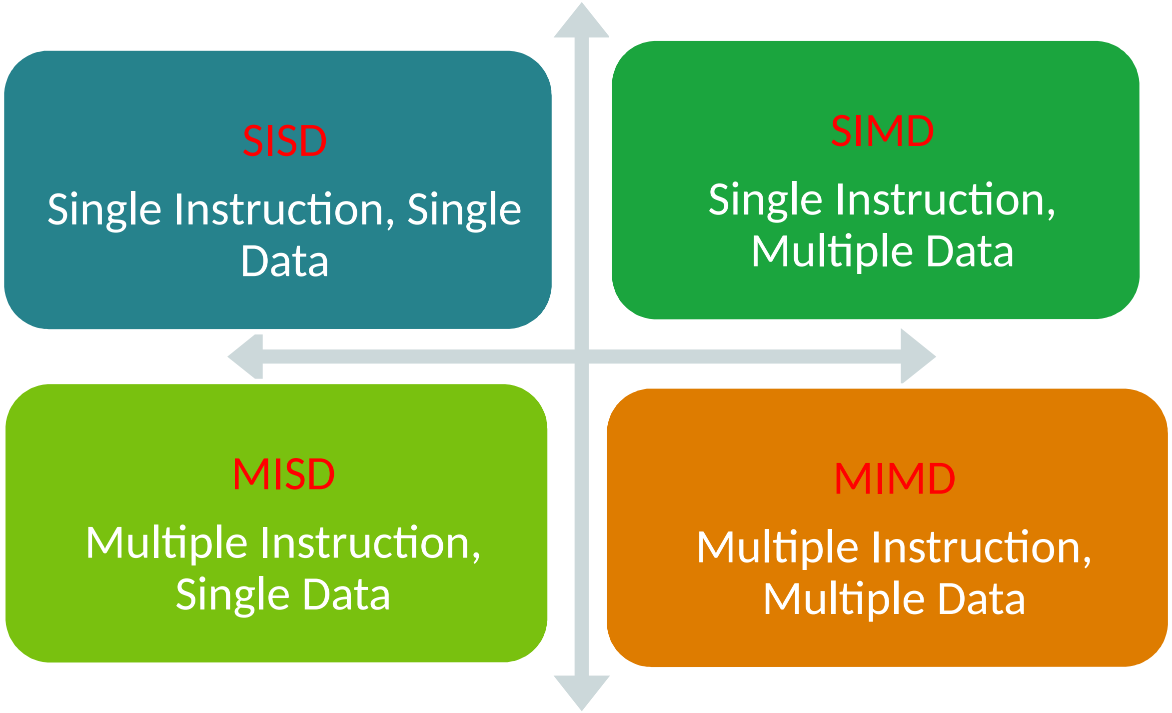


N inputs/outputs: $\log_2 N$ stages and N/2 switches at each stage

Flynn's Classifications

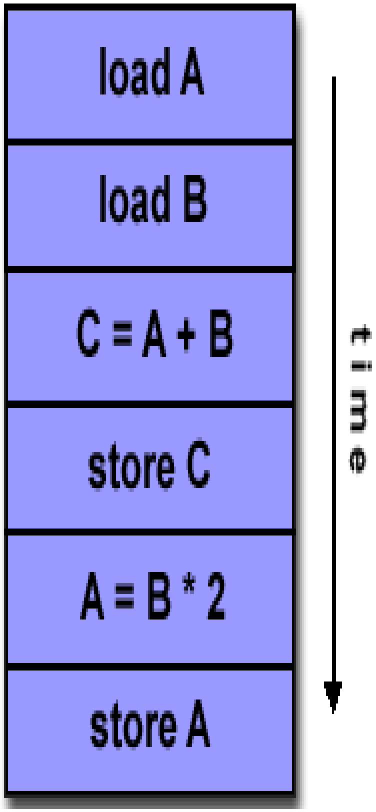
- Flynn (1966) created a classification for computers based on the two dimensions of Instruction and Data.
- Each dimension has two possible states: Single and Multiple.

$\Rightarrow$ 4 possible classifications



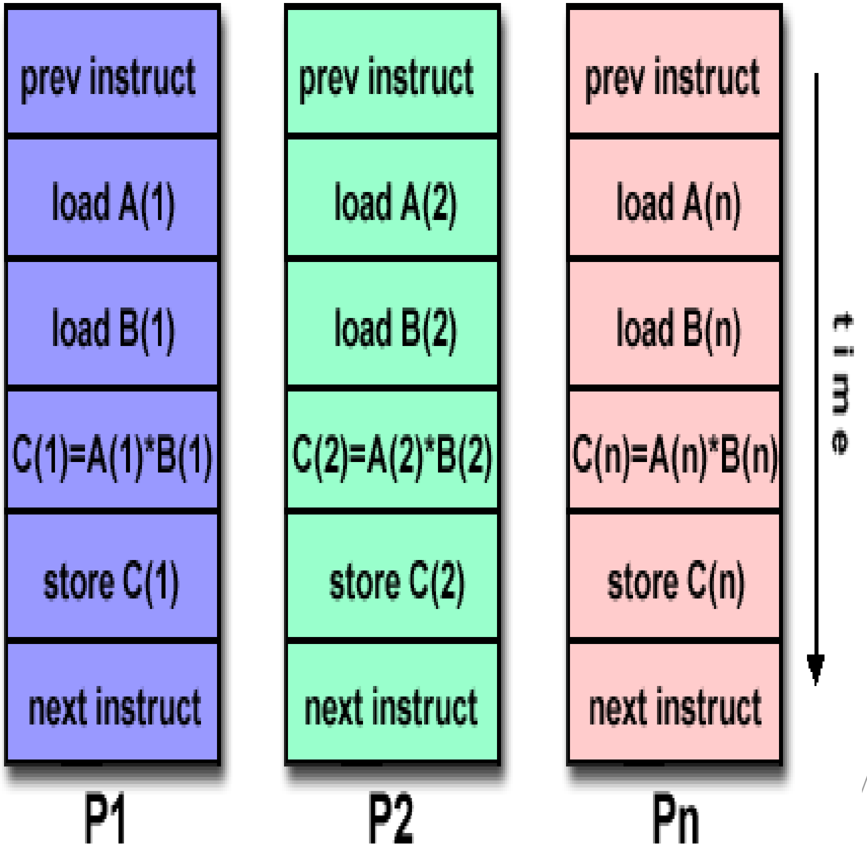
Single Instruction, Single Data (SISD)

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is executed by the CPU during one clock cycle
- Single data: only one data stream is used as input during one clock cycle
- Deterministic execution
- The oldest and the most common type of (single-core) computer



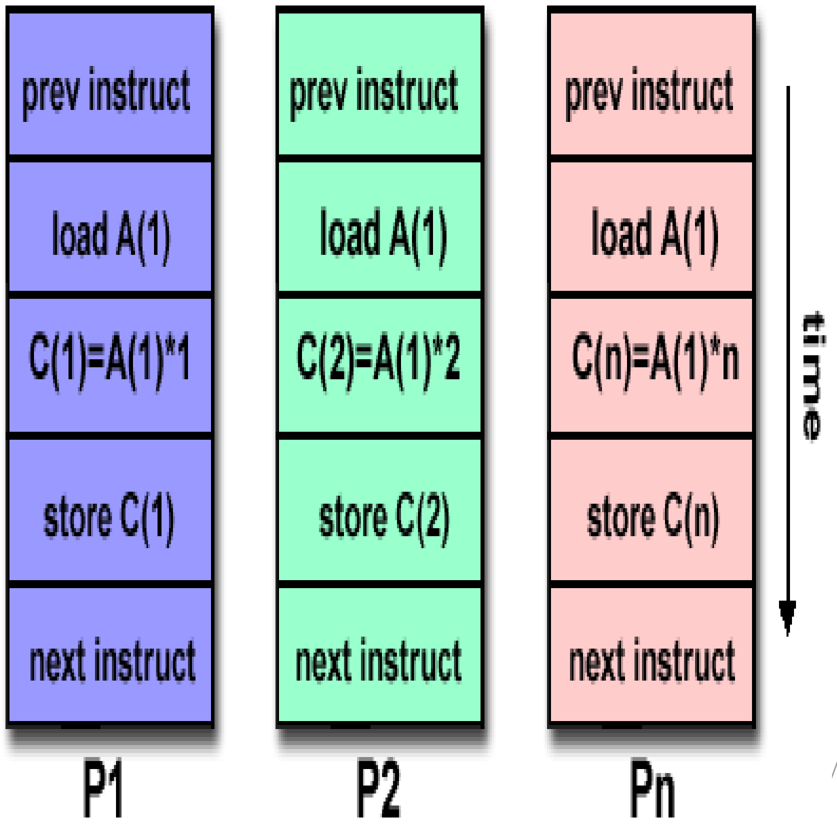
Single Instruction, Multiple Data (SIMD):

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- Synchronous (lockstep) and deterministic execution
- Best suited for graphics/image processing.
  - Graphics Processing Units (GPU)



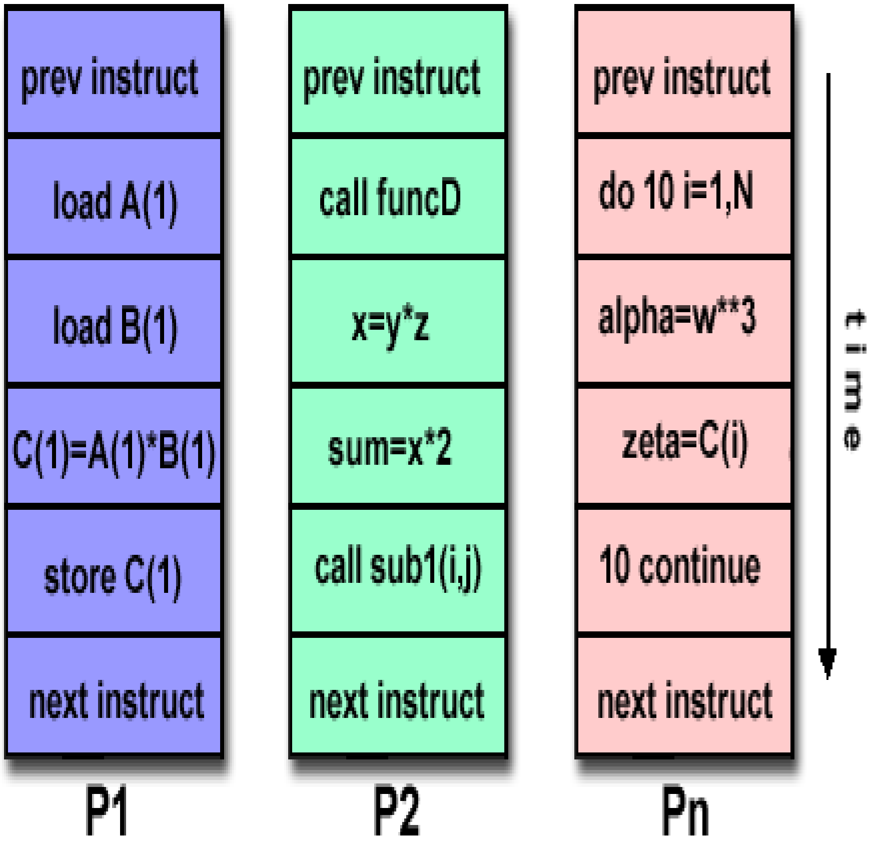
Multiple Instruction, Single Data (MISD)

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- Few actual examples of this class have ever existed.
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.



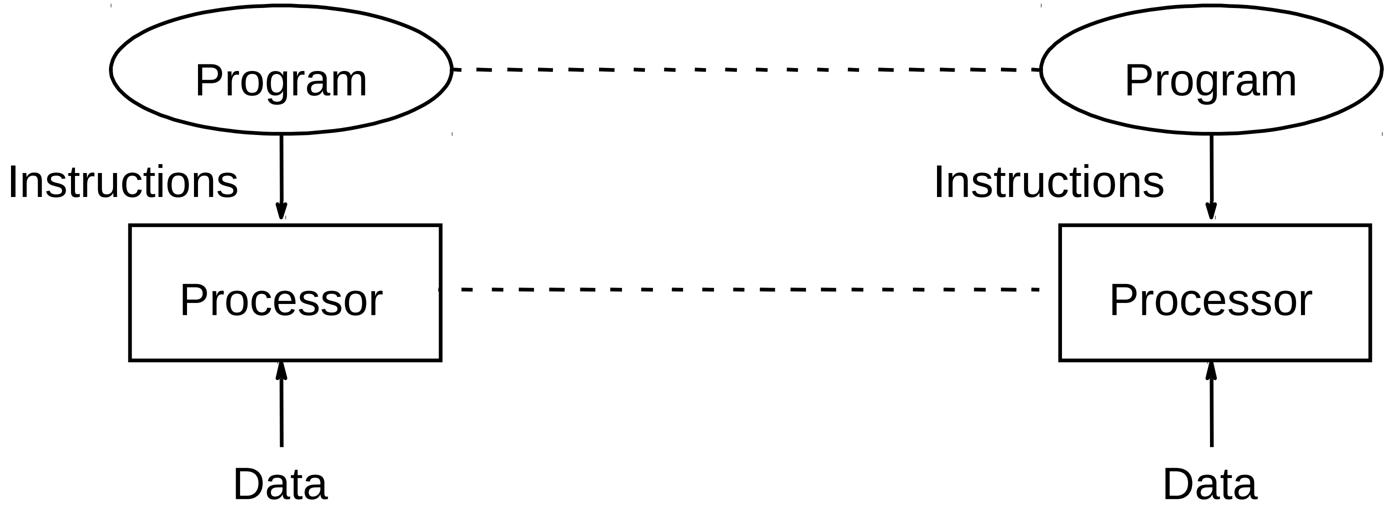
Multiple Instruction, Multiple Data (MIMD)

- Currently, the most common type of parallel computer.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Examples: most current supercomputers, clusters, multi-core computers.
- Note: many MIMD architectures also include SIMD sub-components



Other classifications

- Multiple Program Multiple Data (MPMD)
  - Each processor will have its own program to execute



- Single Program Multiple Data (SPMD)
  - Single source program written and each processor executes its personal copy of this program
  - Parts of the program can executed by certain processors and not others depending on processor ID.

Networked Computers as a Computing Platform

- A network of computers became a very attractive alternative to expensive supercomputers and parallel computer systems for high-performance computing in early 1990's.

- Several early projects. Notable:
  - Berkeley NOW (network of workstations) project.
  - NASA Beowulf project.

Key advantages:

- Very high performance workstations and PCs readily available at low cost.

- The latest processors can easily be incorporated into the system as they become available.

- Existing software can be used or modified.

Software Tools for Clusters

- Based upon Message Passing Parallel Programming:

- Parallel Virtual Machine (PVM) - developed in late 1980’s. Became very popular.

- Message-Passing Interface (MPI) - standard defined in 1990s.

- Both provide a set of user-level libraries for message passing. Use with regular programming languages (C, C++, ...)

Cluster Interconnects

- Originally fast Ethernet on low cost clusters
- Gigabit Ethernet - easy upgrade path

More Specialized/Higher Performance

- Myrinet - 2.4 Gbits/sec
- InfiniBand
- SCI (Scalable Coherent Interface)
- QsNet
- …

References

- Blaise Barney, “Introduction to Parallel Computing”, Livermore Computing.
- Barry Wilkinson & Michael Allen. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers.