Bolum 10

MULTIPROCESSOR ARCHITECTURE

° Classification of Parallel Architectures

Flynn’s Taxonomy

° • SISD: Single instruction single data

– Classical von Neumann architecture

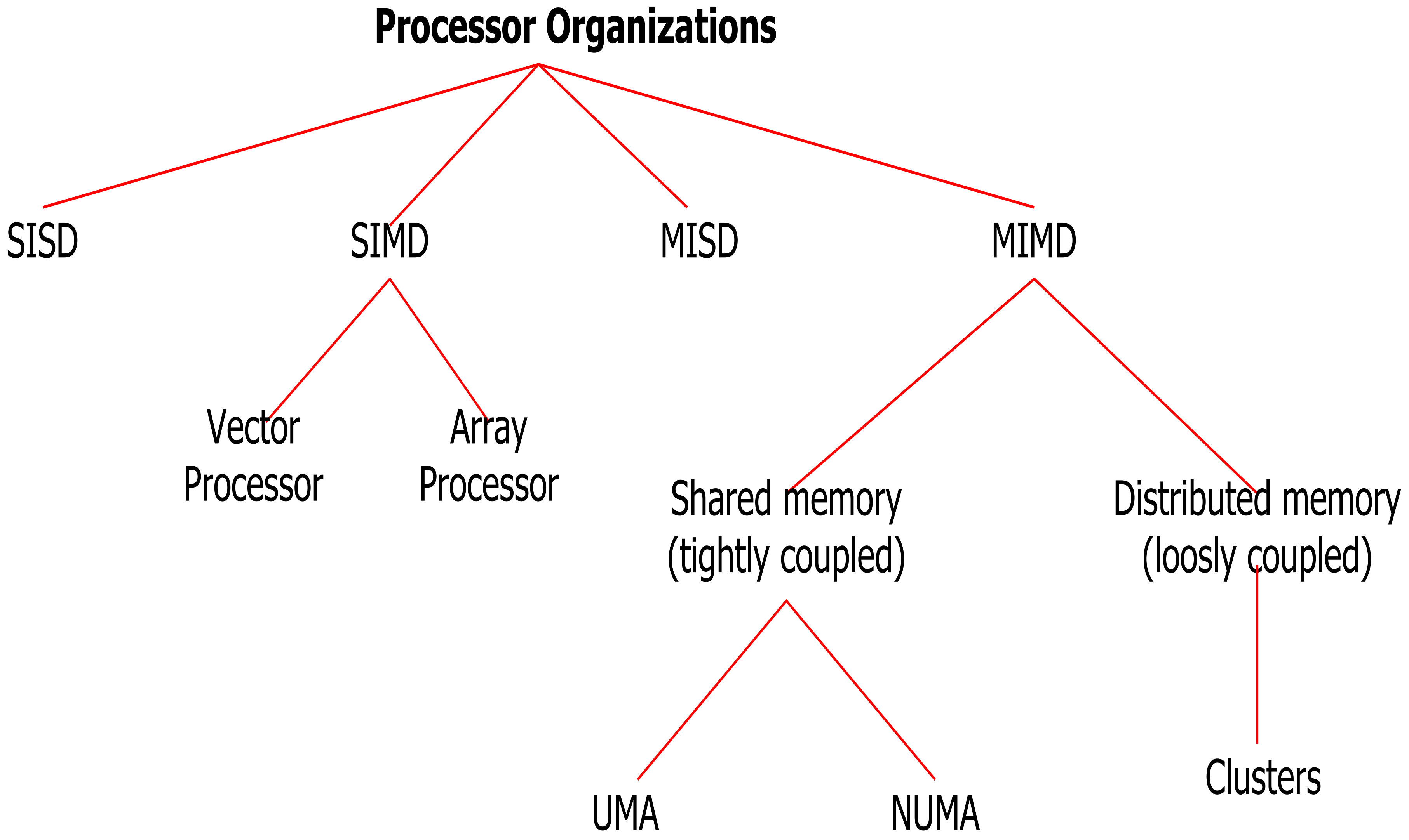
° • SIMD: Single instruction multiple data

° • MISD: Multiple instructions single data

– Non existent, just listed for completeness

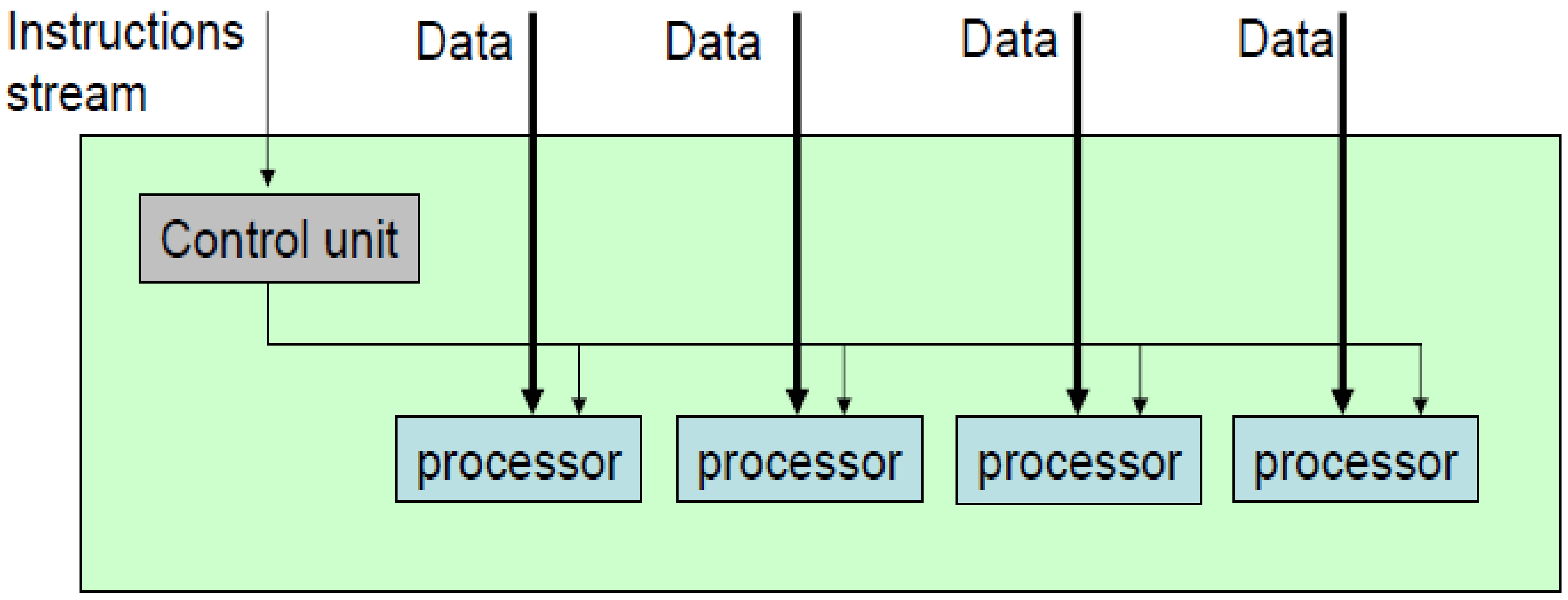
° • MIMD: Multiple instructions multiple data

– Most common and general parallel machine


Processor Organizations
SISD
SIMD
MISD
MIMD
Vector Processor
Array Processor
Shared memory (tightly coupled)
Distributed memory (loosely coupled)
UMA
NUMA
Clusters

Single Instruction Multiple Data

- Also known as Array-processors
- A single instruction stream is broadcasted to multiple processors, each having its own data stream
  - Still used in graphics cards today



Single Instruction stream, Multiple Data stream

Single stream of instruction is broadcast to a number of processor

Each processor operates on its own data

Each processor has its own memories

All processors executes the same program but operate on different data

Multiple Instructions Multiple Data (I)

- Each processor has its own instruction stream and input data
- Very general case
  - every other scenario can be mapped to MIMD
- Further breakdown of MIMD usually based on the memory organization
  - Shared memory systems
  - Distributed memory systems

- SIMD architectures can exploit significant data-level parallelism for:
  - matrix-oriented scientific computing
  - media-oriented image and sound processors
- SIMD is more energy efficient than MIMD
  - Only needs to fetch one instruction per data operation
  - Makes SIMD attractive for personal mobile devices

- **High performance computer**

Large computing capacity
Required to compute large amount of data in a reasonable amount of time

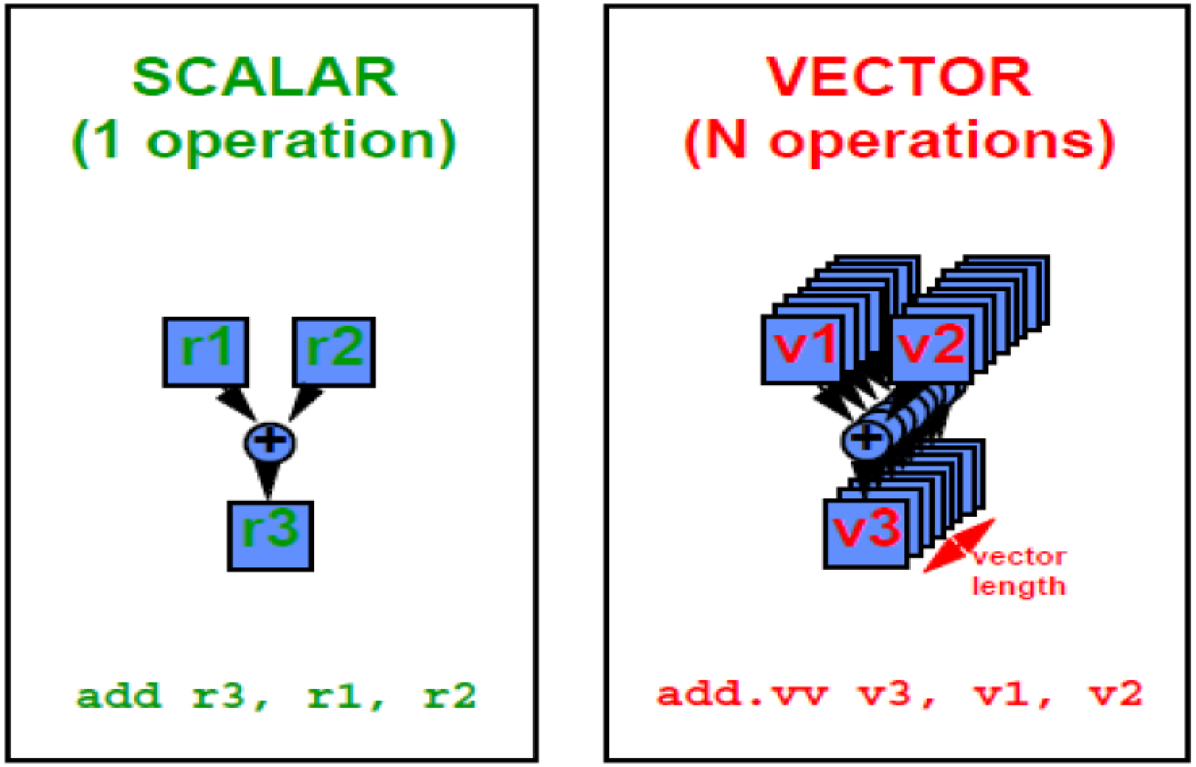
- Vector processing
- Multiprocessing
- Distributed computer system

Vector Processor

- **Also called an Array Processor.**
- **Runs multiple mathematical operations on multiple data elements simultaneously.**
- **Common in supercomputers of the 1970's 80's and 90's.**
- **Today most CPU designs contains at least some vector processing instructions, typically referred to as SIMD.**

- A processor can operate on an entire vector in one instruction
- Work done automatically in parallel (simultaneously)
- The operand to the instructions are complete vectors instead of one element
- Reduce the fetch and decode bandwidth
- Data parallelism
- [illegible]

Skaler ve Vektörel işlem

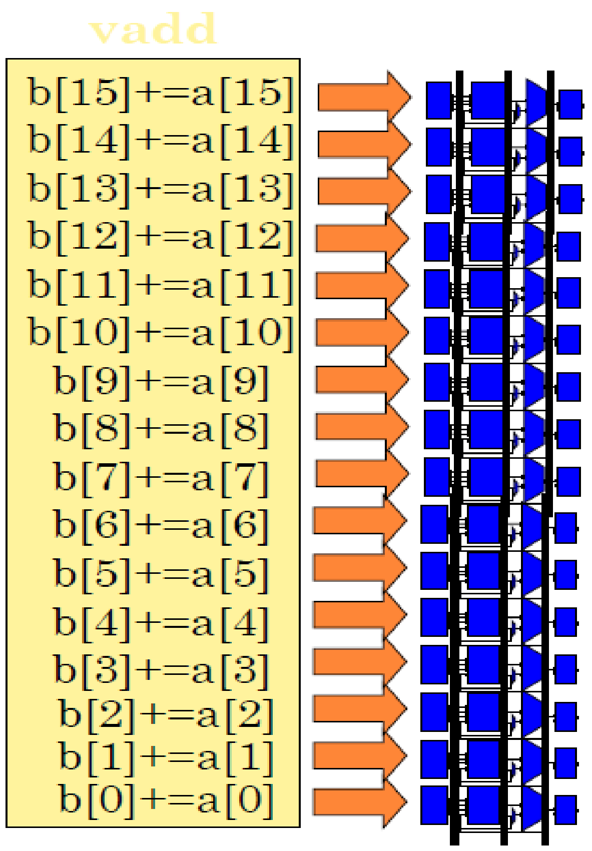


Vector Processors (Cont'd)

```
// C code
for(i=0;i<16; i++)
  b[i]+=a[i]
```

```
// Vectorized code
set     vl,16
vload   vr0,b
vload   vr1,a
vadd    vr0,vr0,vr1
vstore  vr0,b
```

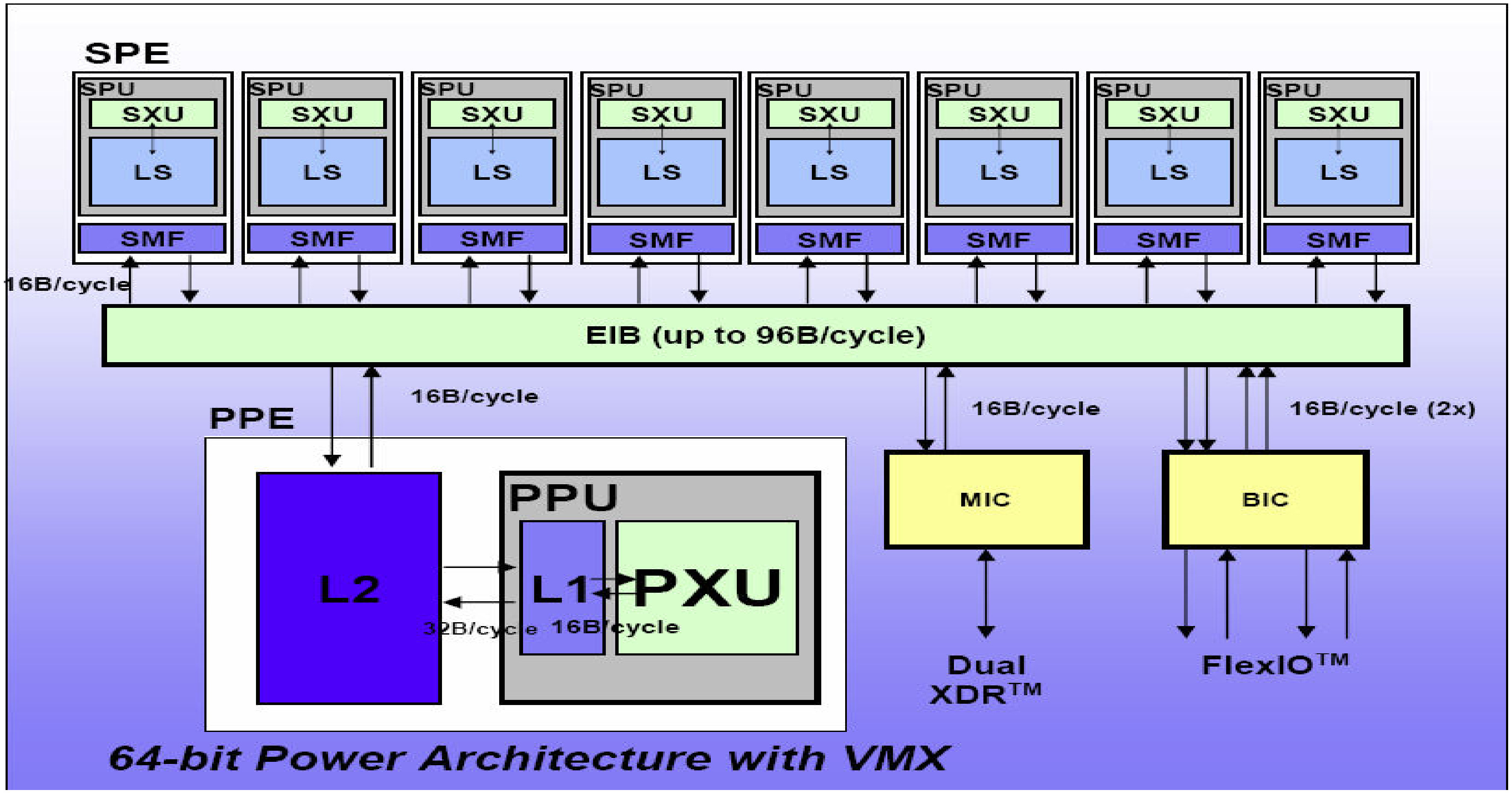
Each vector instruction holds many units of independent operations



16 Vector Lanes

16x speedup

° Typical Vector Processor (Cell Processor)



8 Synergestic Processing Elements (SPE)
64 bit Power Processing Element Control Core (PPE)
Main Control Unit in PPE

Single Instruction Multiple Data - SIMD
Element Interconnection Bus – EIB

MULTIPROCESSING

Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system.
The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them.

□ The term ‘processor’ in multiprocessor can mean either a CPU or an input-output processor (IOP).

□ There are some similarities between multiprocessor and multicomputer systems since both support concurrent operations. However, there exists an important distinction between the two.

□ In case of multicomputer systems, several autonomous computers are connected through a network that may or may not communicate with each other.

On the other hand, in a multiprocessor system, processors interact with each other through an operating system and cooperate in the solution of a problem.

Multiprocessor

Use large number of processor design for workstation or PC market

Has an efficient high bandwidth medium
for communication among
the processor
memory
I/O

Provide High performance but cheaper than vector processing

Multiprocessing performance

- Many computation can proceed in parallel
- Difficulty:
  - the application must be broken down into small task that can be assigned to individual processor
  - Processors must communicate with each other to exchange data

Advantages of Multiprocessing

 A benefit derived from multiprocessing is improved system performance.

This happens because computations can proceed in parallel in one of two ways:

a) Multiple independent jobs can be made to operate in parallel.

b) A single job can be partitioned into multiple parallel tasks.

 Increased Reliability: A failure or error in one part has a limited effect on the rest of the system. If a fault causes one processor to fail, a second processor can be assigned to perform the functions of the disabled processor.

 Increased Throughput: An increase in the number of processors completes the work in less time. It is important to note that doubling the number of processors does not halve the time to complete a job.

Classification Multiprocessors

Multiprocessors are classified by the way their memory is organized.

There are two main kinds of multiprocessing systems:-

- Tightly Coupled Systems

.

- Loosely Coupled Systems

Tightly Coupled Systems

 A multiprocessor system with common shared memory is classified as a shared- memory or tightly coupled multiprocessor.
This does not prevent each processor from having its own local memory.

 In fact, most commercial tightly coupled multiprocessors provide a cache memory with each CPU.
In addition, there is a global common memory that all CPUs can access.
Information can be therefore be shared among the CPUs by placing it in the common global memory.

 Symmetric multiprocessing (SMP) involves a multiprocessor system architecture where two or more identical processors connect to a single, shared main memory, have full access to all I/O devices, and are controlled by a single operating system instance that treats all processors equally, reserving none for special purposes

Loosely Coupled Systems

 An alternative model of microprocessor is the distributed memory or loosely coupled system.

Each processor element in a loosely coupled system has its own private local memory.

 The processors are tied together by a switching scheme designed to route information from one processor to another through a message-passing scheme.

 Loosely coupled systems are most efficient when the interaction between tasks is minimal unlike tightly coupled systems can tolerate a higher degree of interaction between tasks

The structure of general-purpose multiprocessors

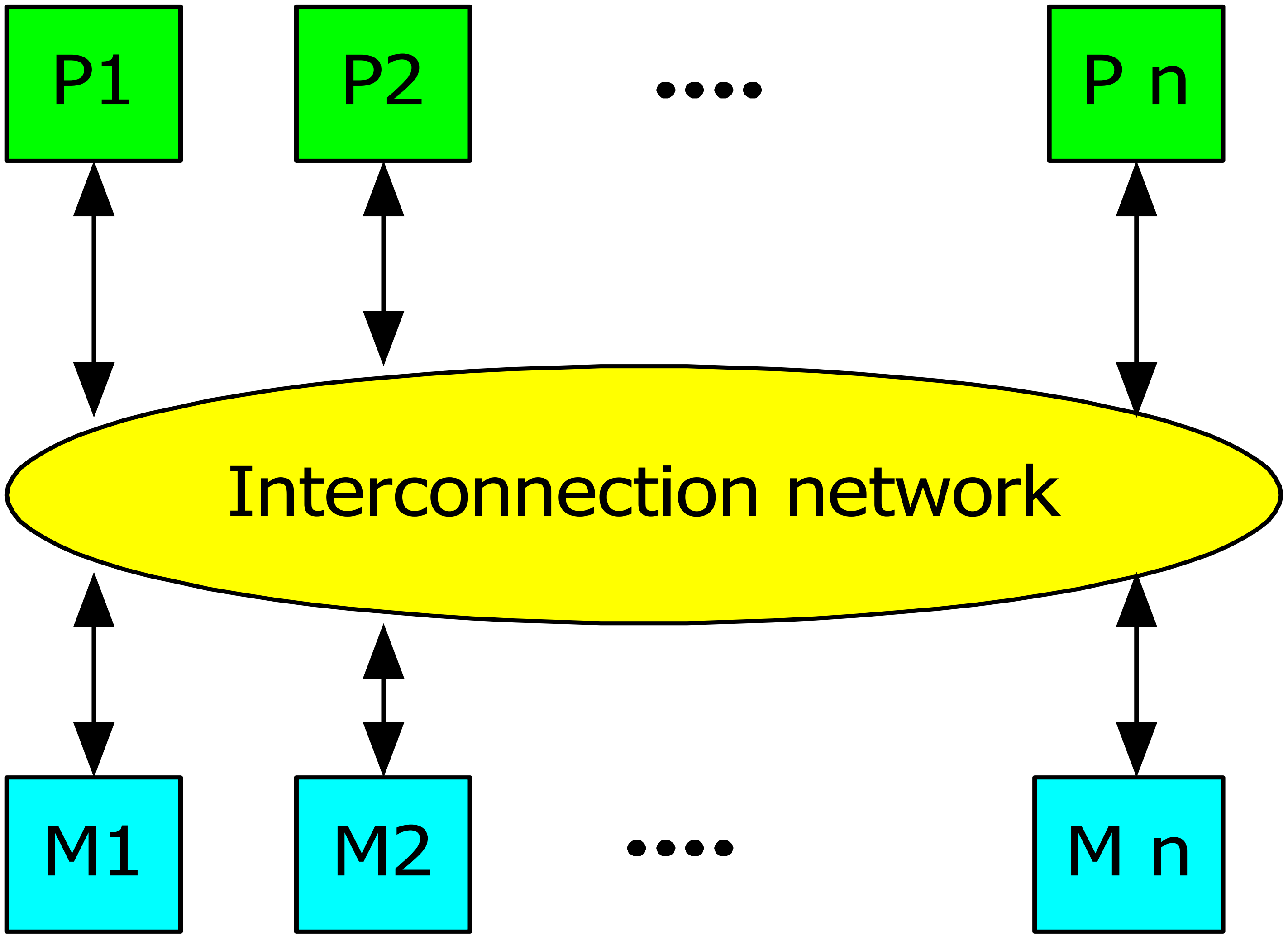
- **UMA multiprocessor**
  Thightly coupled system
- **NUMA multiprocessor**
  Thightly coupled system
- **Distributed memory system**
  Loosely coupled system

UMA & NUMA

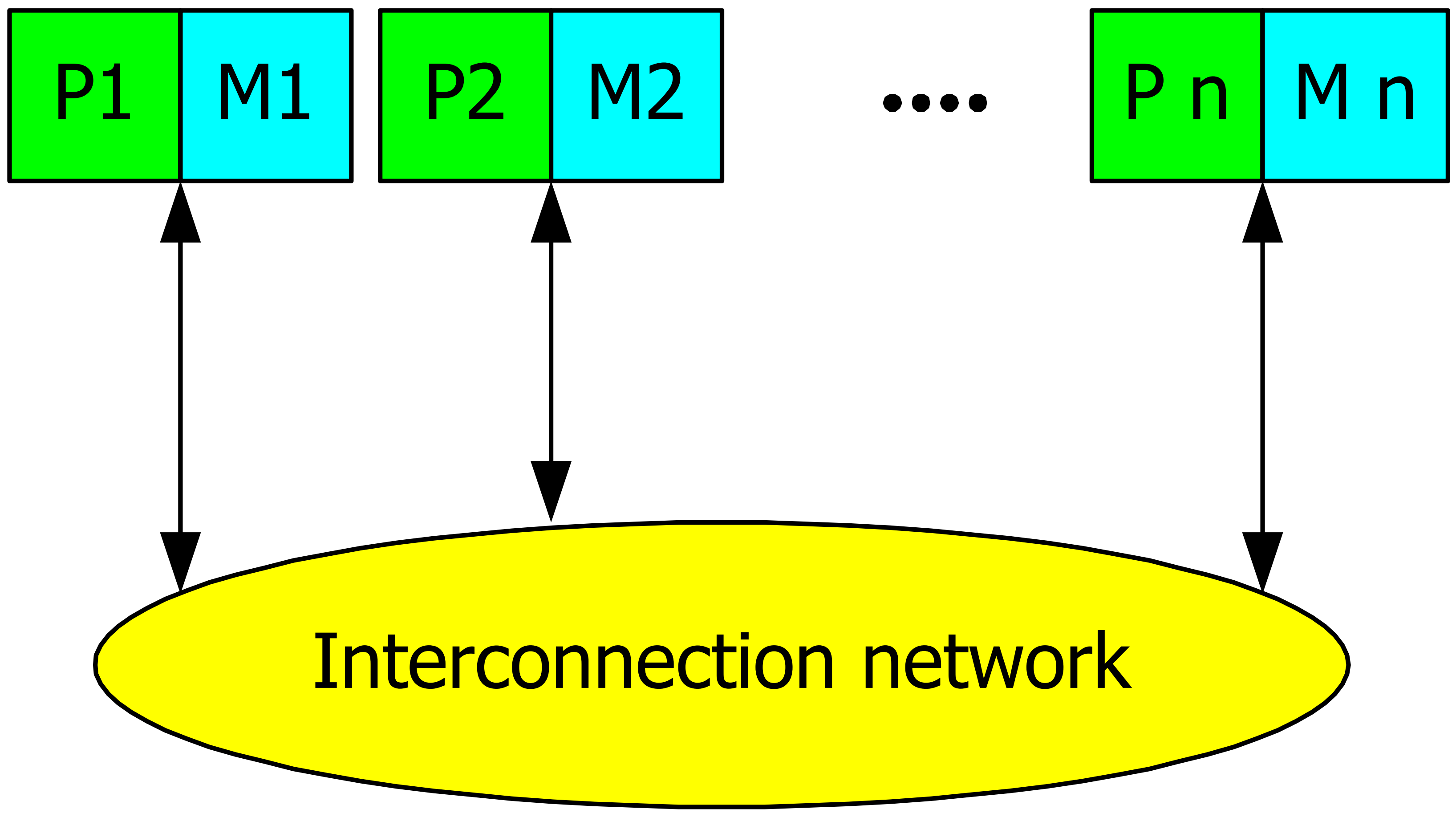
 Uniform memory access (UMA) is a shared memory architecture used in parallel computers.
All the processors in the UMA model share the physical memory uniformly.
The UMA model is suitable for general purpose and time sharing applications by multiple users.

 Non-uniform memory access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor.
Under NUMA, a processor can access its own local memory faster than non-local memory (memory local to another processor or memory shared between processors).

A UMA multiprocessor



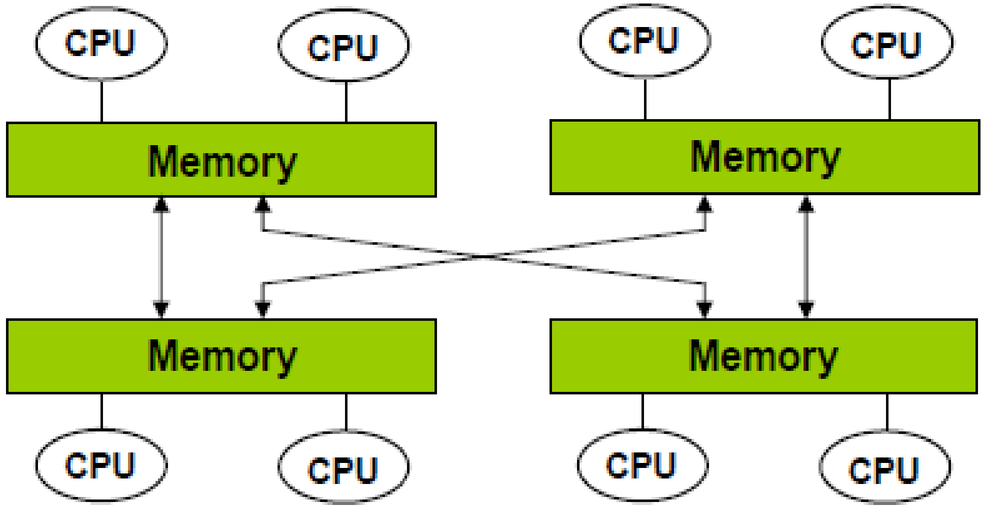
A NUMA multiprocessor



NUMA architectures (I)

Some memory is closer to a certain processor than other memory

- The whole memory is still addressable from all processors
- Depending on what data item a processor retrieves, the access time might vary strongly



NUMA architectures (II)

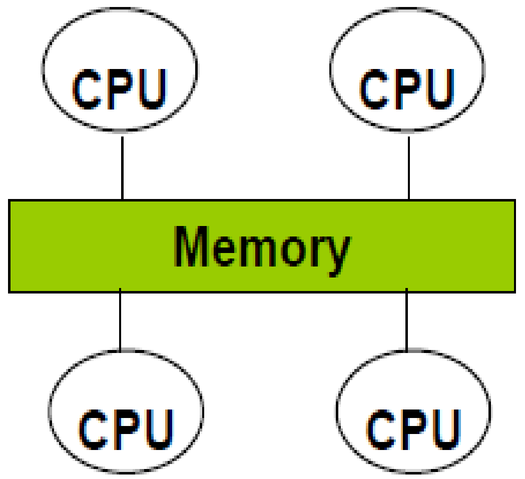
- Reduces the memory bottleneck compared to SMPs
- More difficult to program efficiently
  - E.g. first touch policy: data item will be located in the memory of the processor which uses a data item first
- To reduce effects of non-uniform memory access, caches are often used
  - ccNUMA: cache-coherent non-uniform memory access architectures
- Largest example as of today: SGI Origin with 512 processors

Shared memory systems (I)

- All processes have access to the same address space
  - E.g. PC with more than one processor
- Data exchange between processes by writing/reading shared variables
  - Shared memory systems are easy to program
  - Current standard in scientific programming: OpenMP
- Two versions of shared memory systems available today
  - Symmetric multiprocessors (SMP)
  - Non-uniform memory access (NUMA) architectures

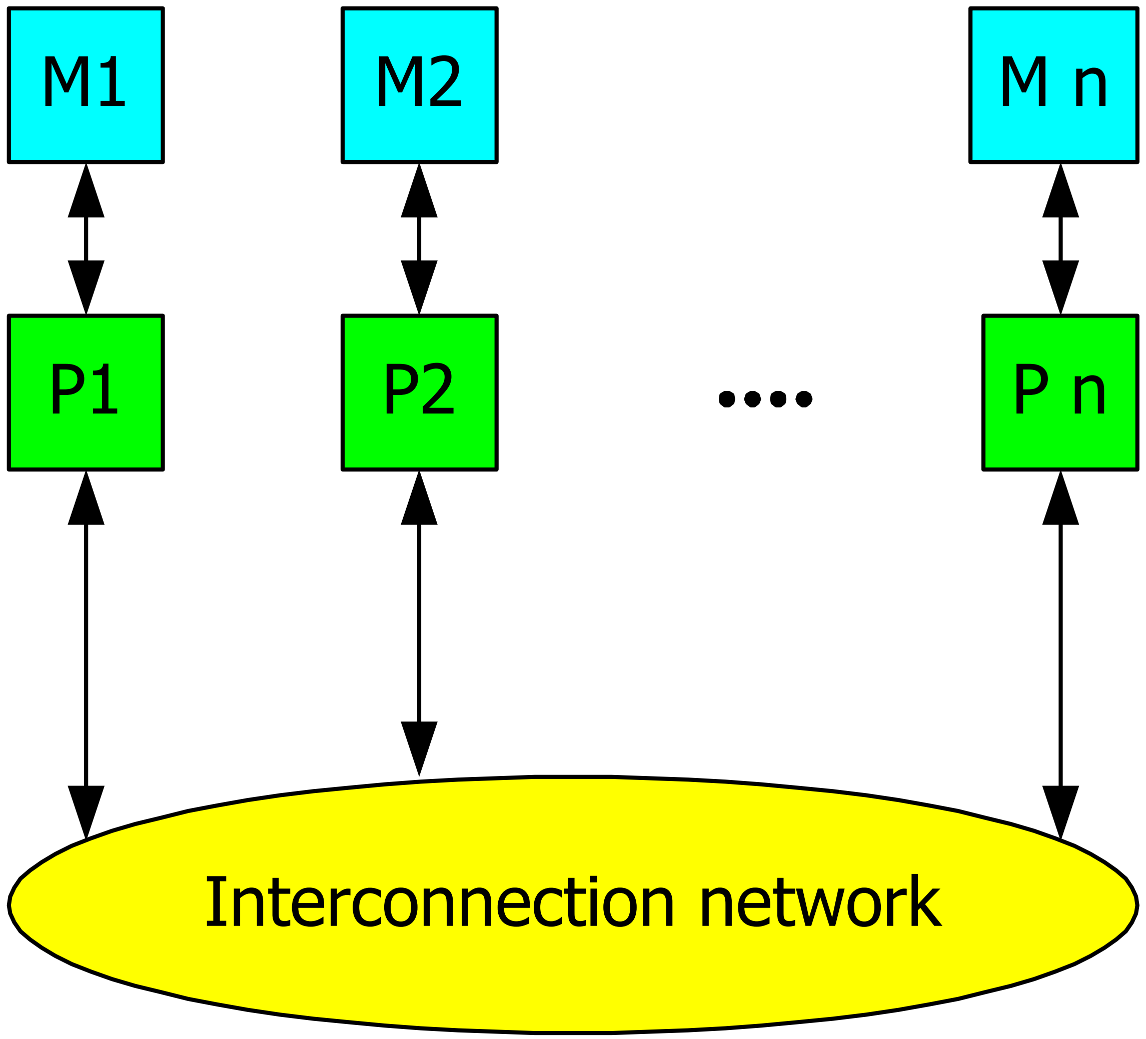
Symmetric multi-processors (SMPs)

- All processors share the same physical main memory



- Memory bandwidth per processor is limiting factor for this type of architecture
- Typical size: 2-32 processors

A distributed memory system

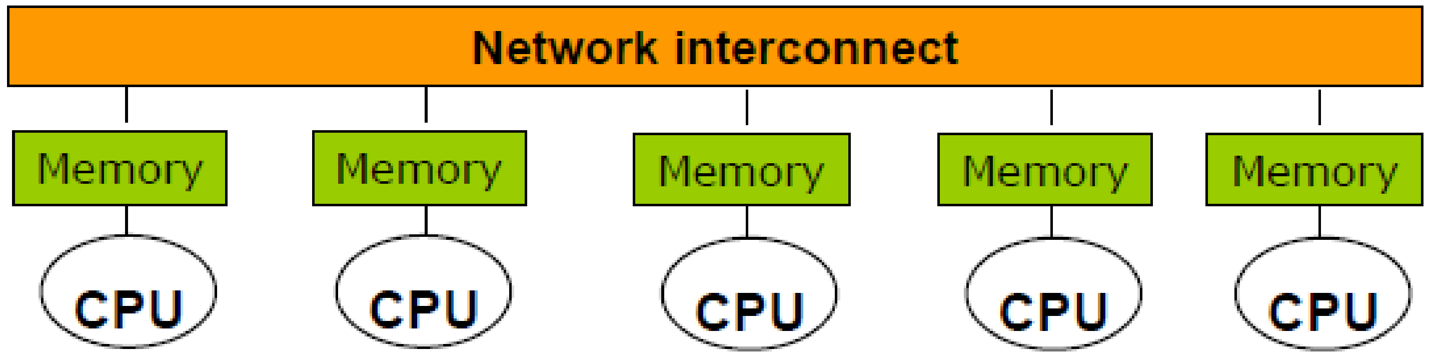


Distributed memory machines (I)

Each processor has its own address space

Communication between processes by explicit data exchange

- Sockets
- Message passing
- Remote procedure call / remote method invocation



Distributed memory machines (II)

Performance of a distributed memory machine strongly depends on the quality of the network interconnect and the topology of the network interconnect

- Of-the-shelf technology: e.g. fast-Ethernet, gigabit-Ethernet
- Specialized interconnects: Myrinet, Infiniband, Quadrics, 10G Ethernet …

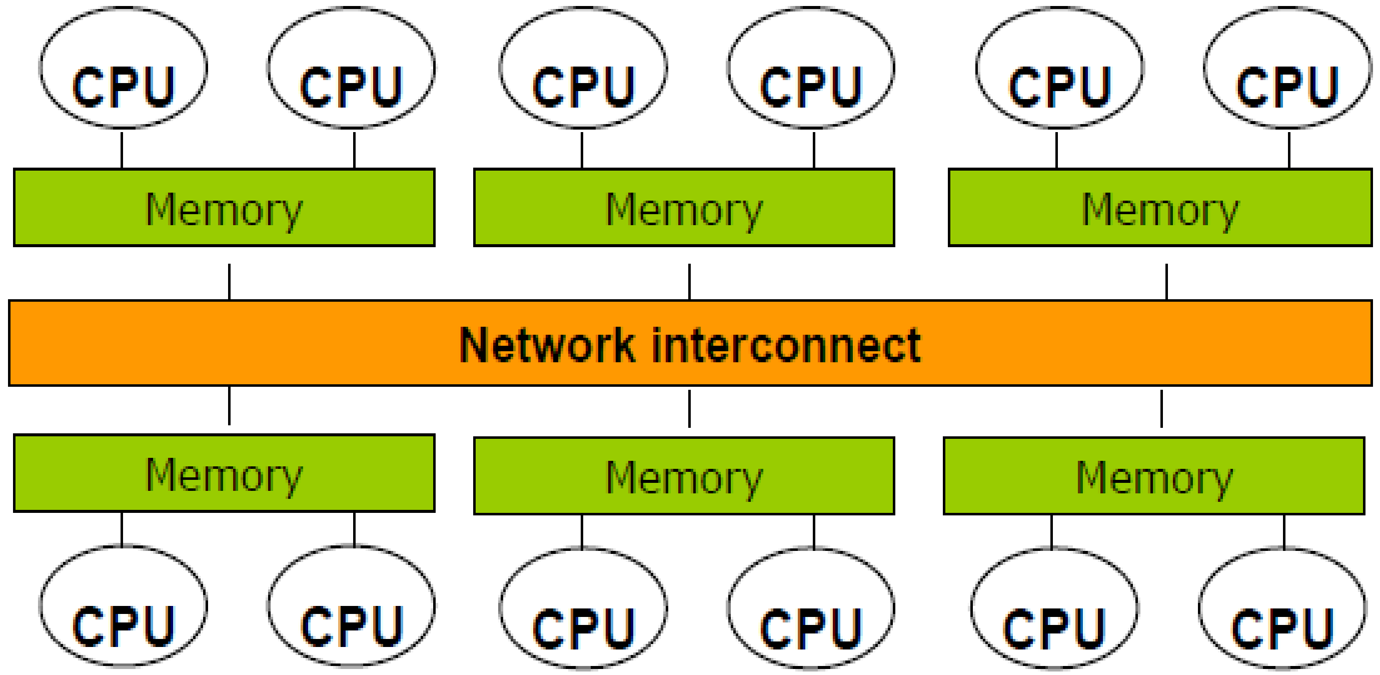
Distributed memory machines (III)

Two classes of distributed memory machines:

- Massively parallel processing systems (MPPs)
  - Tightly coupled environment
  - Single system image (specialized OS)
- Clusters
  - Of-the-shelf hardware and software components such as
    - Intel P4, AMD Opteron etc.
    - Standard operating systems such as LINUX, Windows, BSD UNIX

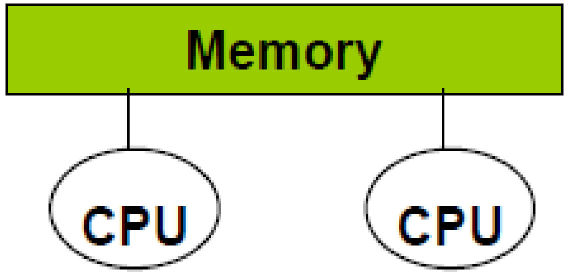
Hybrid systems

E.g. clusters of multi-processor nodes

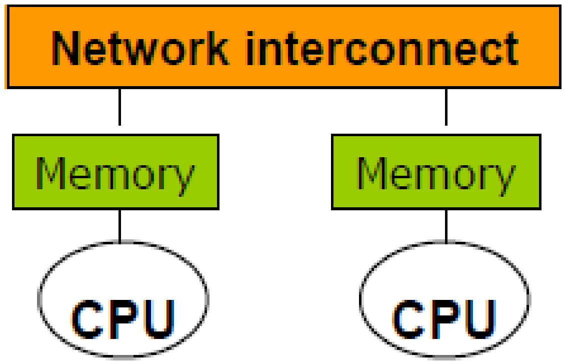


Shared memory vs. distributed memory machines

Shared memory machines:
- Compiler directives (e.g. Open MP)
- Threads (e.g. POSIX Threads)



Distributed memory machines:
- Message Passing (e.g. MPI, PVM)
- Distributed Shared Memory (e.g. UPC, CoArrayFortran)
- Remote Procedure Calls (RPC/RMI)



Message passing widely used in (scientific) parallel programming
- price/performance ratio of ‘message passing hardware’
- very general concept

MIMD (III)

Important metrics:

- Latency:
  - minimal time to send a very short message from one processor to another
  - Unit: ms, µs
- Bandwidth:
  - amount of data which can be transferred from one processor to another in a certain time frame
  - Units: Bytes/sec, KB/s, MB/s, GB/s
    Bits/sec, Kb/s, Mb/s, Gb/s,
    baud

Performance Metrics (I)

Speedup: how much faster does a problem run on *p* processors compared to 1 processor?

$$S(p) = \frac{T_{total}(1)}{T_{total}(p)}$$

– Optimal: *S(p)* = *p* (linear speedup)

Parallel Efficiency: Speedup normalized by the number of processors

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

– Optimal: *E(p)* = *1.0*

Performance Metrics (II)

- Example: Application A takes 35 min. on a single processor, 27 on two processors and 18 on 4 processors.

$$S(2) = \frac{35}{27} = 1.29 \qquad E(2) = \frac{1.29}{2} = 0.645$$

$$S(4) = \frac{35}{18} = 1.94 \qquad E(4) = \frac{1.94}{4} = 0.485$$

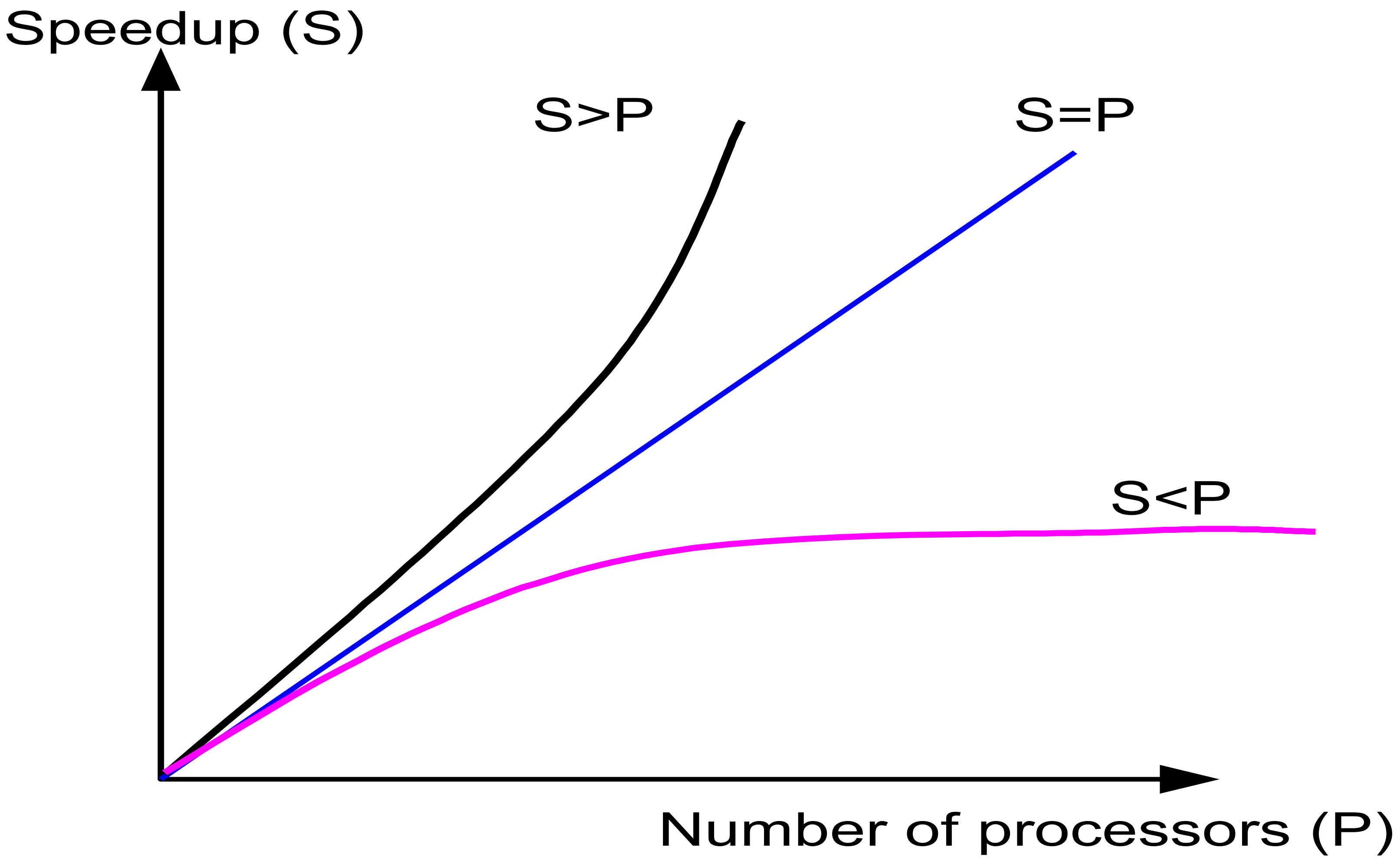
Amdahl's Law (I)

Most applications have a (small) sequential fraction, which limits the speedup

$$T_{total} = T_{sequential} + T_{parallel} = fT_{Total} + (1-f)T_{Total}$$

f: fraction of the code which can only be executed sequentially

$$S(p) = \frac{T_{total}(1)}{(f + \frac{1-f}{p})T_{total}(1)} = \frac{1}{f + \frac{1-f}{p}}$$


Speedup (S)
S>P
S=P
S<P
Number of processors (P)