Introduction to Multiprocessors (Part I)

Prof. Cristina Silvano

Politecnico di Milano

Outline

- ❑ Key issues to design multiprocessors
- ❑ Interconnection network
- ❑ Centralized shared-memory architectures
- ❑ Distributed shared-memory architectures
- ❑ Message passing architectures
- ❑ Communication models
- ❑ Introduction to the problem of cache coherence

Multiprocessors

- **A new era in Computer Architecture**
- Multiprocessors now play a major role from embedded to high end general-purpose computing:
  - Main goals: very high-end performance; scalability; reliability.
- Multiprocessors refers to tightly coupled processors whose coordination and usage is controlled by a single operating system and that usually share memory through a shared address space
- **Multicores** when all cores are in the same chip (named **manycores** when more than 32 cores)
- Multiple Instruction Multiple Data **(MIMD)**
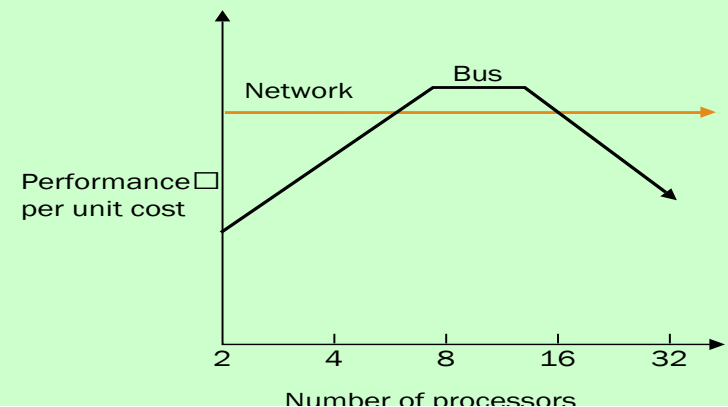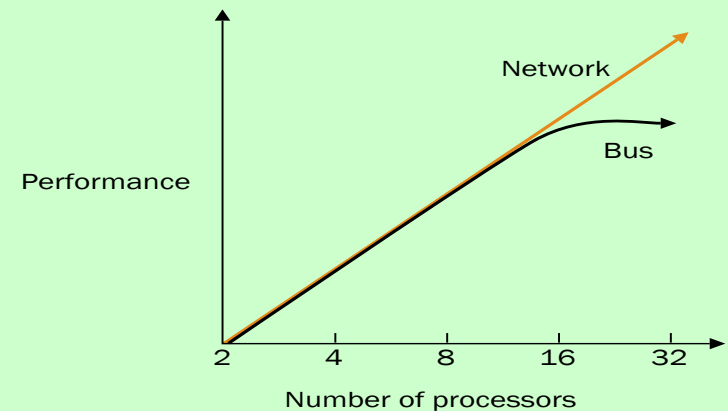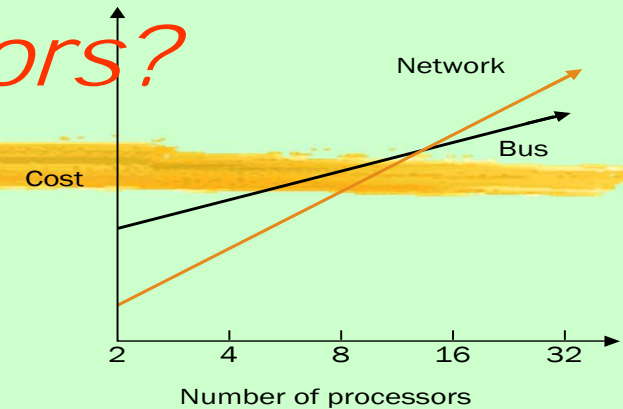
Key issues to design multiprocessors

- *How many processors?*
- *How powerful are processors?*
- *How do connect processors?*
- *How do parallel processors share data?*
- *Where to place the physical memory?*
- *How do parallel processors cooperate and coordinate?*
- *How to program processors?*
- *How to maintain cache coherence?*
- *How to maintain memory consistency?*
- *How to evaluate system performance?*

How do connect processors?

Single bus vs. interconnection network:

❑ **Single-bus** approach imposes constraints on the number of processors connected to it (up to now, 36 is the largest number of processors connected in a commercial single bus system) ⇒ **saturation**.

❑ To connect many processors with a high bandwidth, the system needs to use more than a single bus ⇒ **introduction of an interconnection network.**

Cost

Network

Bus

2   4   8   16   32

Number of processors

Performance

Network

Bus

2   4   8   16   32

Number of processors

Performance per unit cost

Bus

Network

2   4   8   16   32

Number of processors

Cost/Performance Tradeoffs

❑ The **network-connected** machine has a smaller initial cost, then the costs scale up more quickly than the **bus-connected** machine.

❑ Performance for both machines scales linearly until the bus reaches its limit, then performance is flat no matter how many processors are used.

❑ When these two effects are combined $\Rightarrow$ the **network-connected** machine has consistent performance per unit cost, while the **bus-connected** machine has a 'sweet spot' plateau (8 to 16 processors).

❑ **Network-connected** MPs have better cost/performance on the left of the plateau (because they are less expensive), and on the right of the plateau (because they have higher performance).

Arch. of Single-Bus Multiprocessors

❑ In **Single-Bus Multiprocessors,** the connection medium *(the bus)* is between the processors and memory $\Rightarrow$ the medium is used on every memory access.
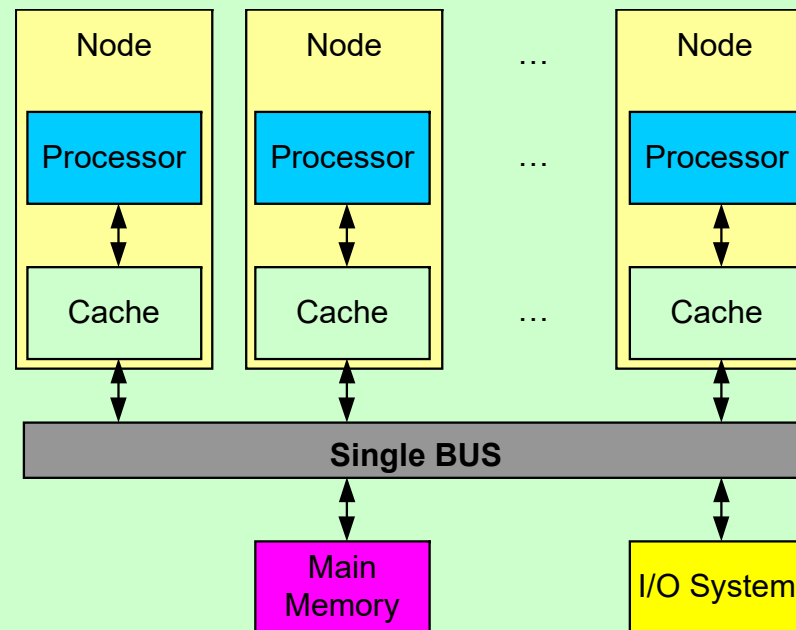
Arch. of Network-Connected MPs

❑ In **Network-Connected Multiprocessors,** memory is attached to each processor, and the connection medium *(the network)* is between the nodes ⇒ the medium is used only for interprocessor communication.
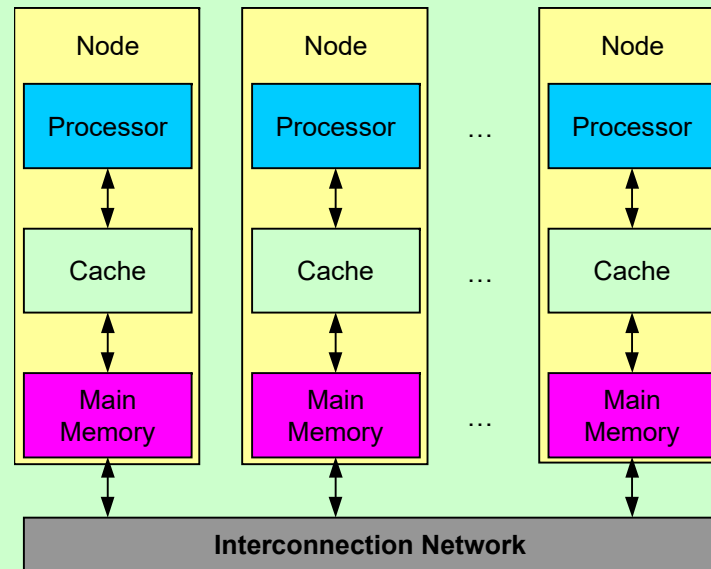
Network Topologies

- Between the high cost/performance of a **fully connected network** (with a dedicated communication link between every node) and the low cost/performance of a **single bus,** there are a set of **network topologies** that constitutes a wide range of trade-offs in cost/performance metric.
- Examples of topology of the interconnection network:
  - **Single bus**
  - **Ring**
  - **Mesh**
  - **N-cube**
  - **Crossbar Network**
- The topology of the interconnection network can be different for **data** and **address** buses.

Network Representation and Costs

❏ Networks are represented as **graphs** where:

   ➢ **Nodes** (shown as black square) are processor-memory nodes

   ➢ **Switch** (shown as red circle) whose links go to processor-memory nodes and to other switches

   ➢ **Arcs** representing a link of the communication network (all links are bidirectional $\Rightarrow$ information can flow in either direction).

❏ **Network costs** include:

   ➢ Number of switches

   ➢ Number of links on a switch to connect to the network

   ➢ Width (number of bits) per link

   ➢ Length of links when the network is mapped to a physical machine

Single-bus and Ring

❑ **Single-bus**
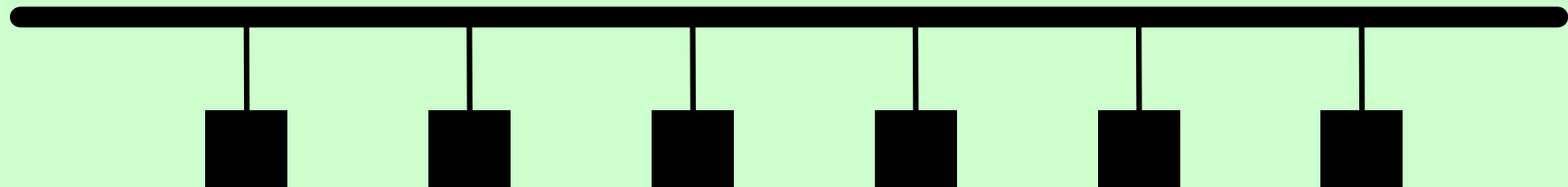
❑ **Ring:** capable of many simultaneous transfers (like a segmented bus). Some nodes are not directly connected ⇒ the communication between some nodes needs to pass through intermediate nodes to reach the final destination (multiple-hops).
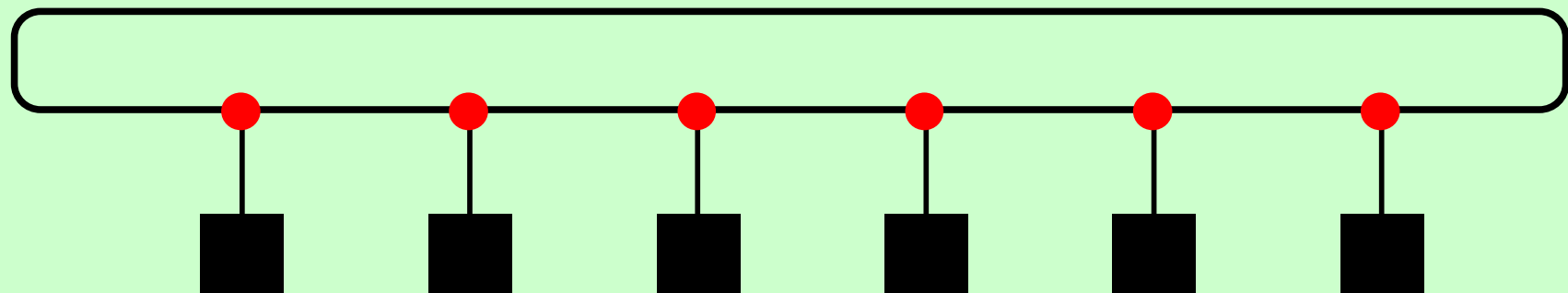
Network Performance Metrics

P = **number of nodes**

M = **number of links**

b = **bandwidth of a single link**

❑ **Total Network Bandwidth (best case): M x b**
number of links multiplied by the bandwidth of each link

➢ For the single-bus topology, the total network bandwidth is just the bandwidth of the bus: *(1 x b)*

➢ For the ring topology **P = M** and the total network bandwidth is **P** times the bandwidth of one link: *(P x b)*

Network Performance Metrics

❑ **Bisection Bandwidth** *(worst case):* This is calculated by dividing the machine into two parts, each with half the nodes. Then you sum up the bandwidth of the links that cross that imaginary dividing line.

 ➢ For the ring topology is two times the link bandwith: *(2 x b),*

 ➢ For the single bus topology is just the bus bandwidth: *(1 x b).*

❑ Some network topologies are **not** symmetric: where to draw the bisection line? To find the worst case metric, we need to choose the division that yields the most pessimistic network performance (that is calculate all possible bisection bandwidths and pick the smallest)

Bisection Bandwidth Calculation

❑ **Single-bus**
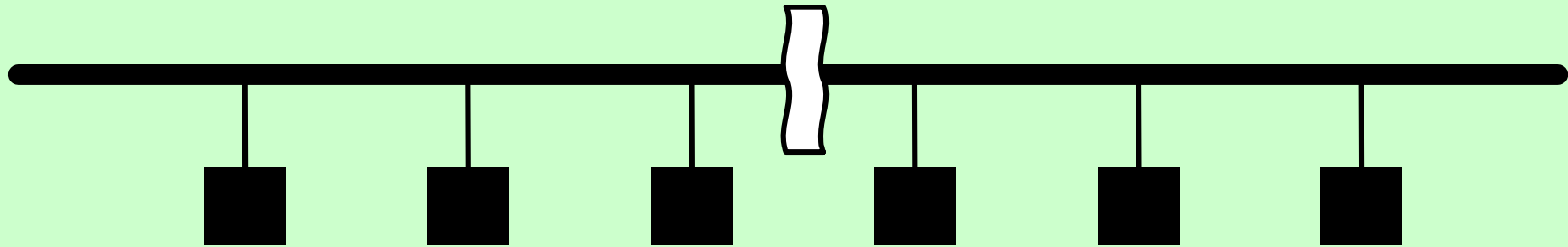
❑ **Ring**
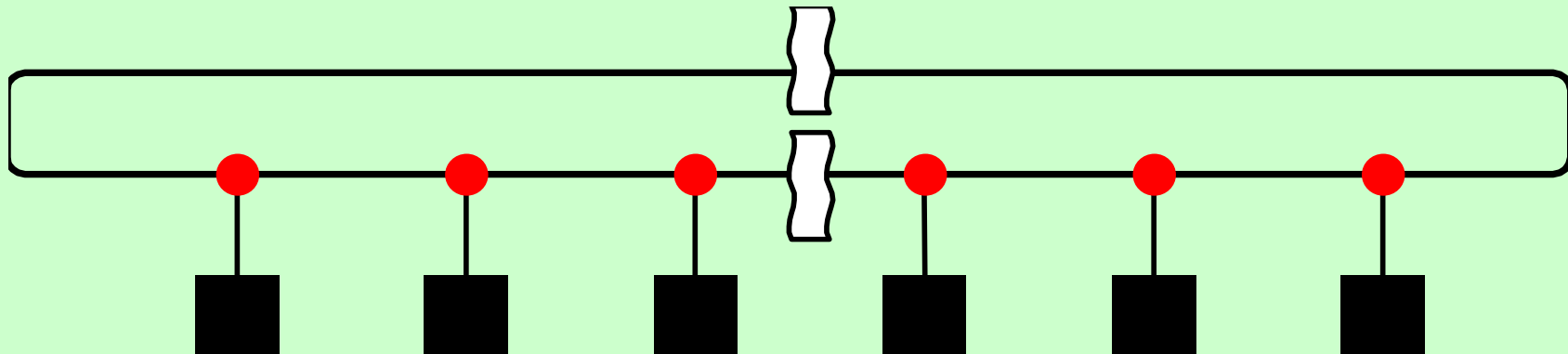
Single-bus vs Ring

	Best case Total Bandwidth	Worst case Bisection Bandwidth
Single Bus	b	b
Ring	$P\,b$	$2\,b$

❑ Single-bus is as fast as the single link.

❑ Given the bandwidth b of a single link:

  ➢ The ring is only **twice** as fast as the single-bus in the worst case

  ➢ The ring is **P times** faster in the best case.

Crossbar Network

❑ **Crossbar Network or fully connected network:**
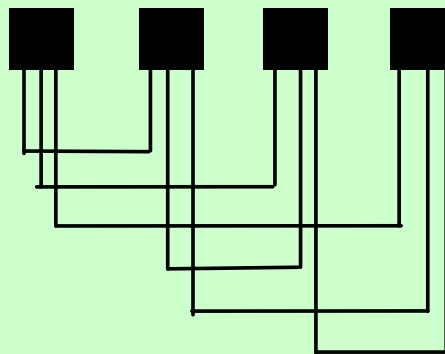every processor has a bidirectional dedicated communication link to every other processor
$\Rightarrow$ very high cost

❑ Total Bandwidth: $\{(P \times (P-1))/2\} \times b$
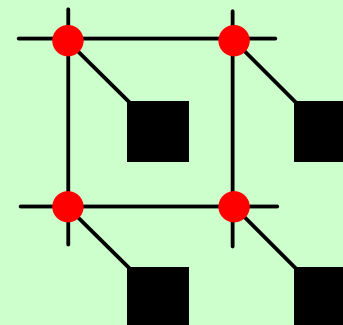
❑ Bisection Bandwidth: $(P/2)^2 \times b$

❑ Example: $P = 4$ nodes
$\Rightarrow$ Total BW = $6 \times b$; Bisection BW = $4 \times b$

Bidimensional Mesh (2D Mesh)

- Given P nodes: $N = \sqrt{P}$
- $N \times (N - 1)$ **horizontal channels**
- $N \times (N - 1)$ **vertical channels**
- Number of links per internal switch = 5
- Number of links per external switch = 3
- Total Bandwidth: $\{2 N (N - 1)\} \times b$
- Bisection Bandwidth: $N \times b$
- Example: $P = 4; N = 2 \Rightarrow$ Total Bandwidth = $4 \times b$; Bisection Bandwidth = $2 \times b$
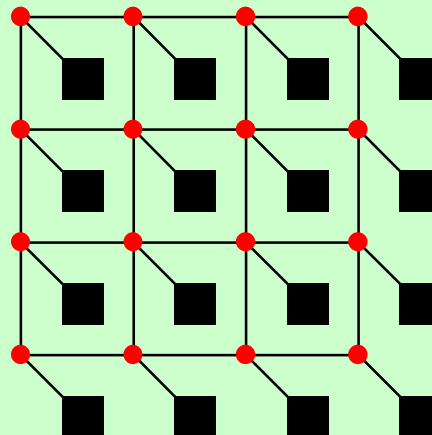
Bidimensional Mesh (2D Mesh)

❑ Example: *P = 16; N = 4* $\Rightarrow$
    12 horizontal channels
    12 vertical channels
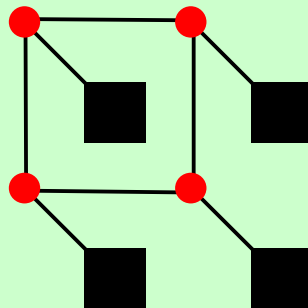   Total Bandwidth = *24 x b*
    Bisection Bandwidth = *4 x b*

Hypercube

- Boolean N-cube with $P = 2^N$ nodes.
- Each node has N neighborhood nodes.
- Number of links per switch = $N+1$
- Total Bandwidth: $\{(N \times P) / 2\} \times b$
- Bisection Bandwidth: $2^{(N-1)} \times b$
- Example: $P = 4\ N = 2 \Rightarrow$ Total Bandwidth = $4 \times b$; Bisection Bandwidth = $2 \times b$

Hypercube

❑ Example: **P = 8 N = 3** $\Rightarrow$ Total Bandwidth = **12 x b**;
  Bisection Bandwidth = **4 x b**
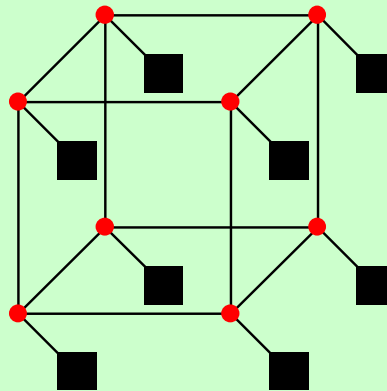
Hypercube

❑ Example: **P = 16 N = 4** $\Rightarrow$ Total Bandwidth = **32 x b**;
   Bisection Bandwidth = **8 x b**
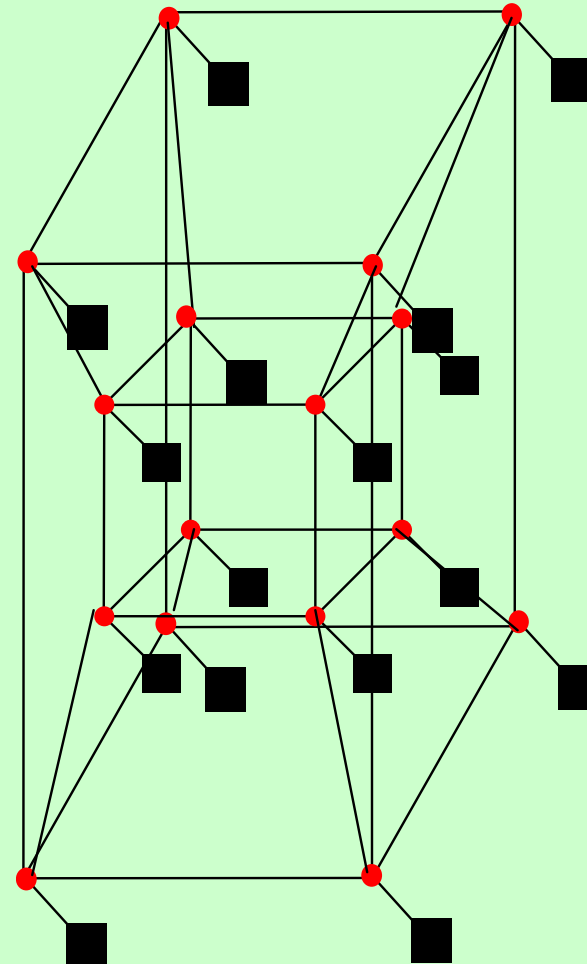
How do parallel processors share data?

❑ Memory Address Space Model:

❑ Single logically shared address space

❑ Multiple and private address spaces

Main Memory

| MM0 |
| MM1 |
| MM2 |
| MM3 |

P0, P1, P2, P3

Main Memory

MM0	P0
MM1	P1
MM2	P2
MM3	P3

Memory Address Space Model

❑ **Single logically shared address space:** A memory reference can be made by any processor to any memory location through loads/stores
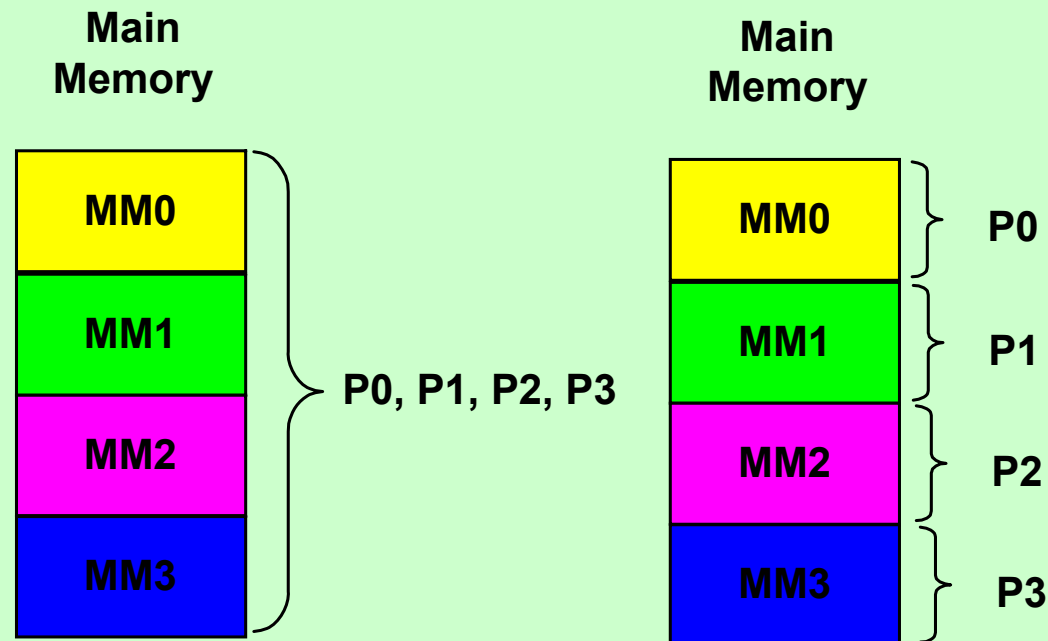⇒ **Shared Memory Architectures.**

➤ The address space is shared among processors: The same physical address on 2 processors refers to the same location in memory.

❑ **Multiple and private address spaces:** The processors communicate among them through send/receive primitives ⇒ **Message Passing Architectures.**

➤ The address space is logically disjoint and cannot be addressed by different processors: the same physical address on 2 processors refers to 2 different locations in 2 different memories.

Shared Addresses

❑ The processors communicate among them through shared variables in memory.

❑ In shared memory architectures, communication among threads occurs through a shared address space.

❑ Implicit management of the communication through **load/store operations** to access any memory locations.

  ➢ Oldest and most popular model

❑ Shared memory does not mean that there is a single centralized memory: the shared memory can be centralized or distributed over the nodes.

❑ Shared memory model imposes the **cache coherence problem** among processors.

Multiple and Private Addresses

❑ The processors communicate among them through sending/receiving messages: **message passing protocol**

❑ Explicit management of the communication through send/receive primitives to access private memory locations.

❑ The memory of one processor cannot be accessed by another processor without the assistance of software protocols.

❑ **No cache coherence problem** among processors.
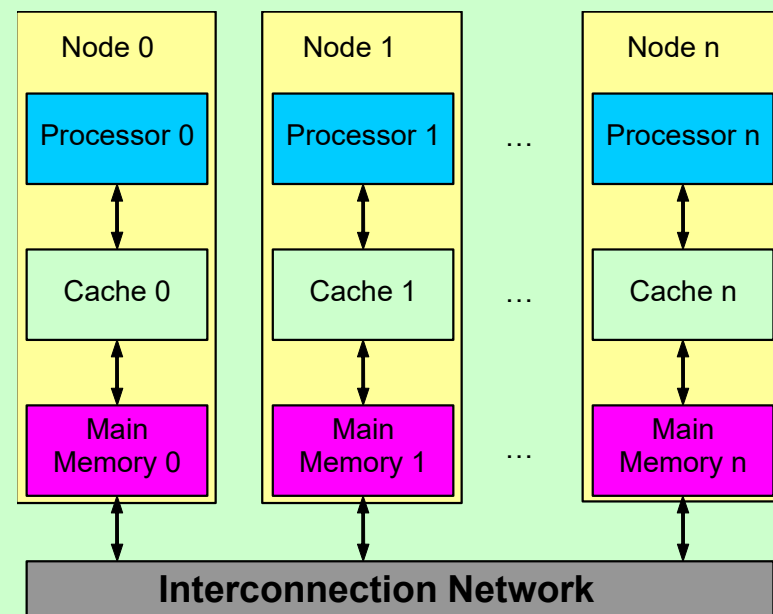
Where to place the physical memory?

❏ Physical Memory Organization:

❏ Centralized Memory   ❏ Distributed Memory
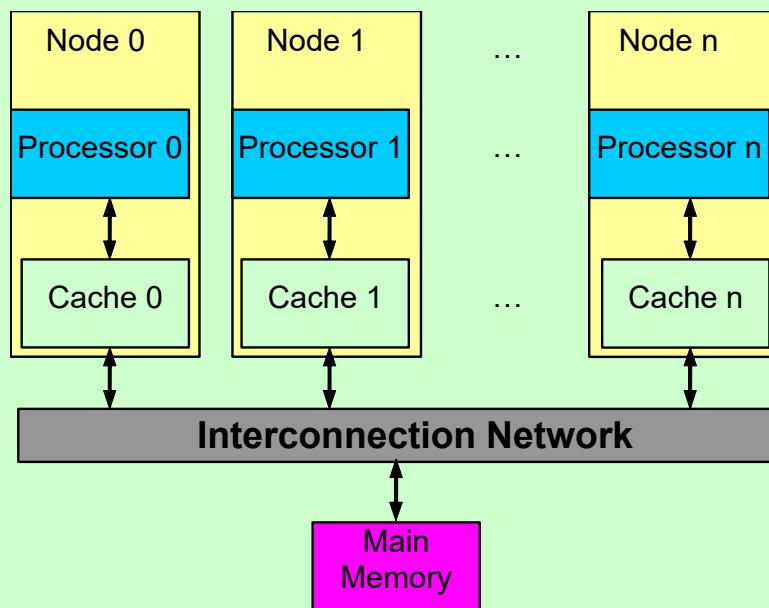
Uniform Memory Access (UMA)   Non Uniform Memory Access (NUMA)

Physical Memory Organization

❑ **Centralized Memory**:

➢ **UMA (Uniform Memory Access):** The access time to a memory location is *uniform* for all the processors: no matter which processor requests it and no matter which word is asked.

❑ **Distributed Memory:** The physical memory is divided into memory modules distributed on each single processor.

➢ **NUMA (Non Uniform Memory Access):** The access time to a memory location is *non uniform* for all the processors: it depends on the location of the data word in memory and the processor location.

Physical Memory Organization

❑ Multiprocessor systems can have single address space and distributed physical memory.

❑ The concepts of addressing space (single/multiple) and the physical memory organization (centralized/distributed) are **_orthogonal_** to each other.
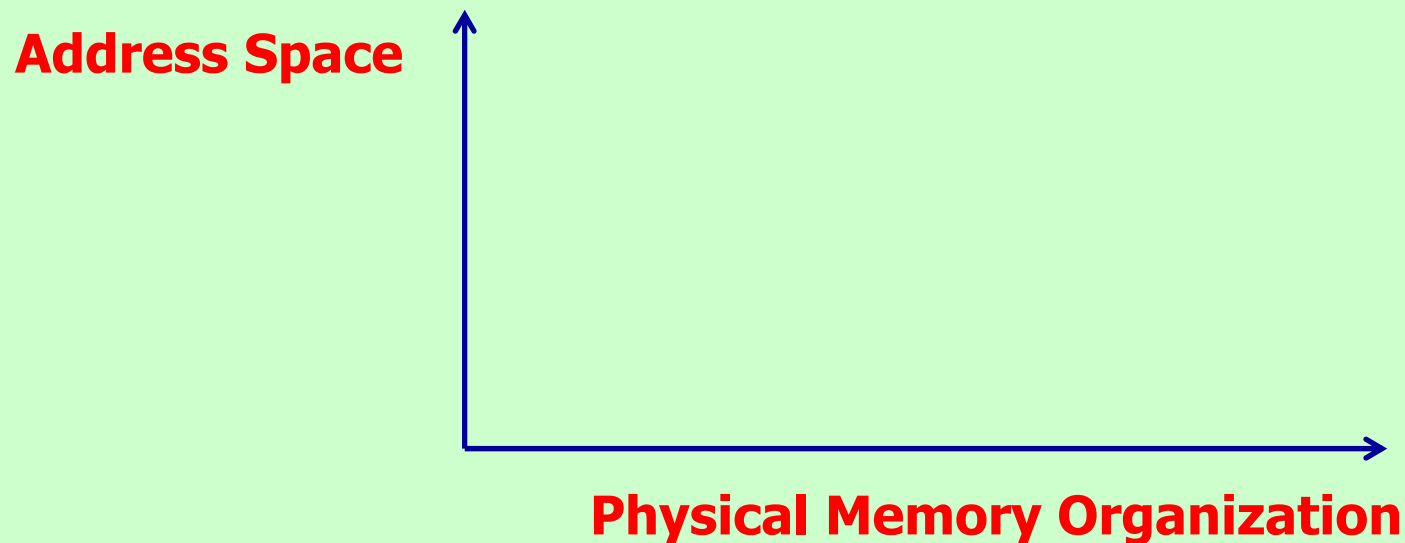
Address Space

Physical Memory Organization

Address Space vs. Physical Mem. Org.

Physical Memory Organization

❑ Centralized Memory   ❑ Distributed Memory

Address Space

❑ Single Logically Shared Address Space (Shared-Memory Architectures)

Address Space vs. Physical Mem. Org.

Physical Memory Organization

❑ Centralized Memory   ❑ Distributed Memory

Address Space

❑ Multiple and Private Address Spaces (Message Passing Architectures)
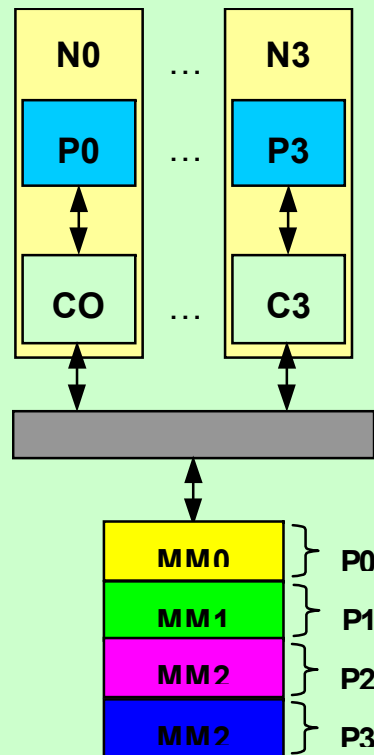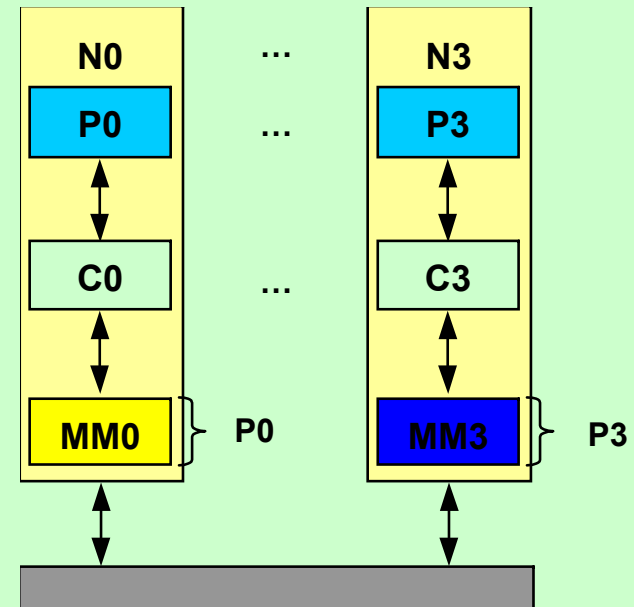
Address Space vs. Physical Mem. Org.

❑ Centralized Memory

❑ Distributed Memory

❑ Single Logically Shared Address Space (Shared-Memory Architectures)
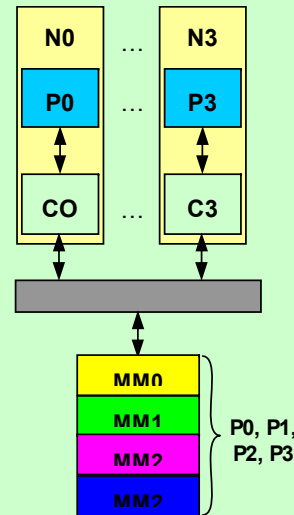
❑ Multiple and Private Address Spaces (Message Passing Architectures)
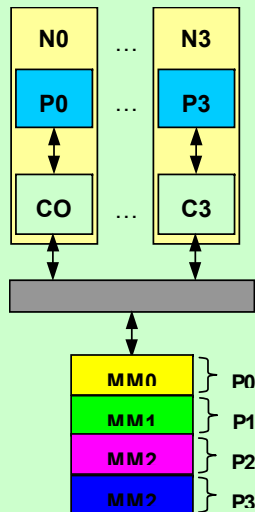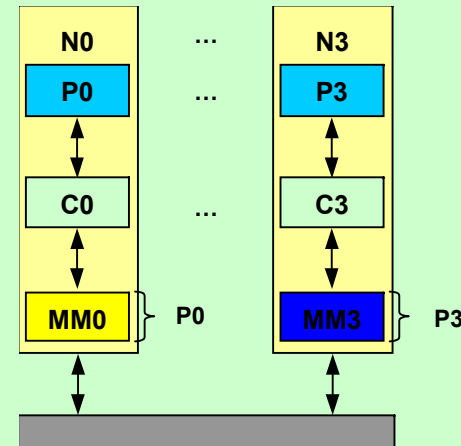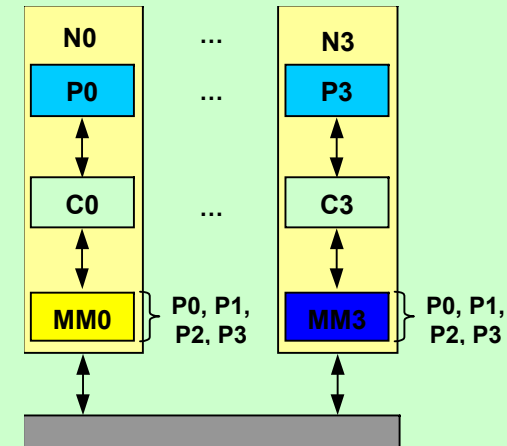
Multicore single-chip multiprocessor

❑ Most of existing **multicores** are single-chip small-scale (<= 8 nodes) with single-bus and **Centralized Shared-Memory (CSM)** architectures:

  ➢ **Symmetric Multiprocessors (SMPs)**

  ➢ **Uniform Memory Access (UMA)** time

  ➢ Snooping protocols to support cache coherence

Node	Node	...	Node
Processor	Processor	...	Processor
Cache	Cache	...	Cache

Single BUS

Main Memory

I/O System
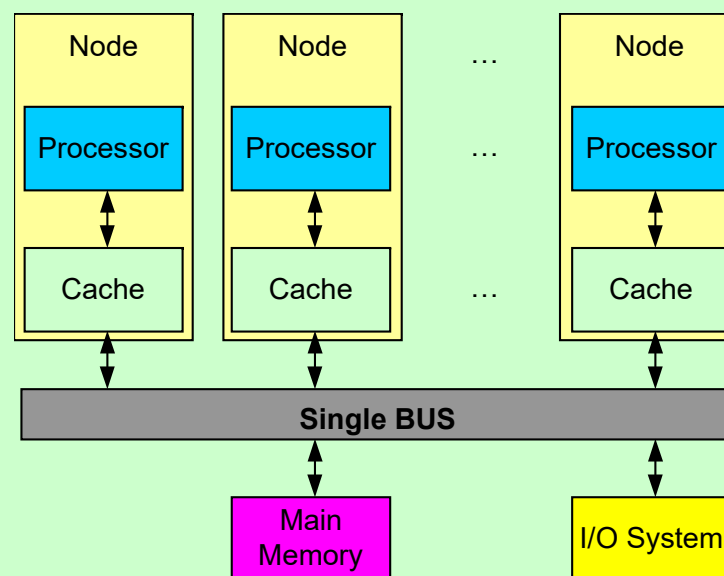
Multicore single-chip multiprocessor

➤ Most of existing **multicores** are typically **SMPs**

➤ **Small number of cores (<=8)**

➤ Share a single centralized memory with uniform memory access **(UMA)**

➤ There is typically one level of shared cache (L3), and one or more levels of private per-core cache (L1 & L2)
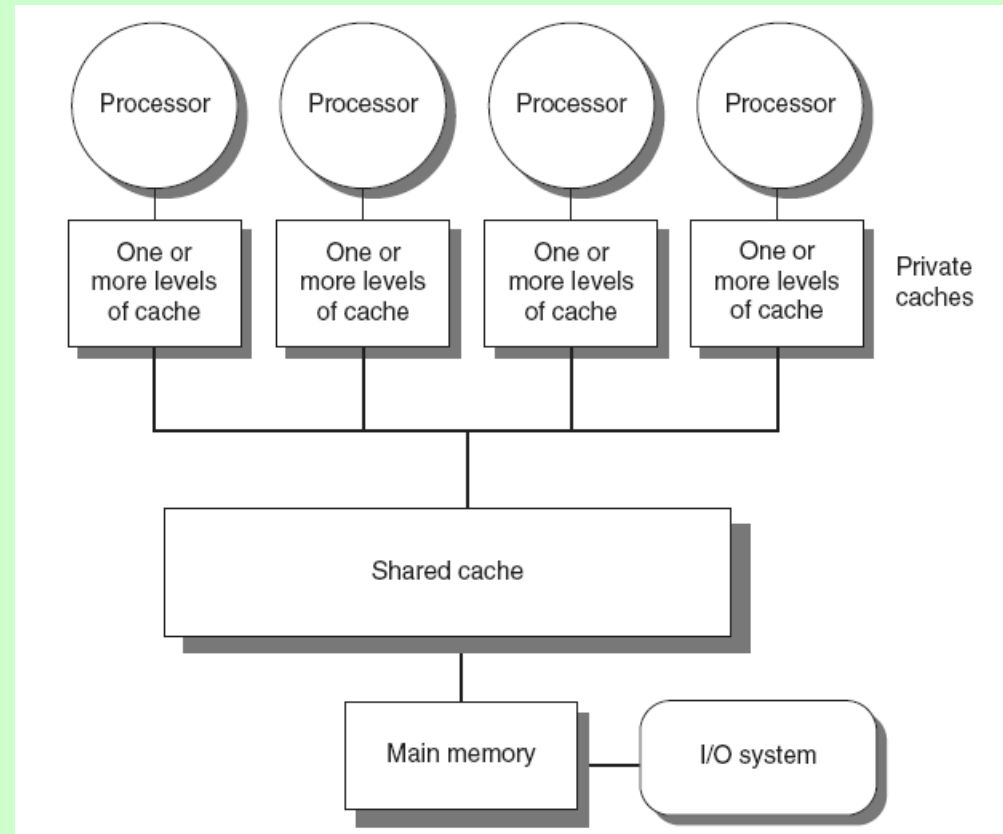
Multicore single-chip multiprocessor

- As the number of processors grows (**>8**), any centralized resource in the system becomes a bottleneck

- To increase the communication bandwidth -> multiple buses as well as **interconnection networks** (such as crossbars or small point-to-point networks), where memory or a shared cache can be configured into multiple physical banks

- Basic idea: to boost the effective memory bandwidth while retaining uniform access time to memory

Prof. Cristina Silvano, Politecnico di Milano

Multicore single-chip multiprocessor

❑ Multicore single-chip multiprocessor with UMA through a banked shared cache and using an interconnection network rather than a bus.
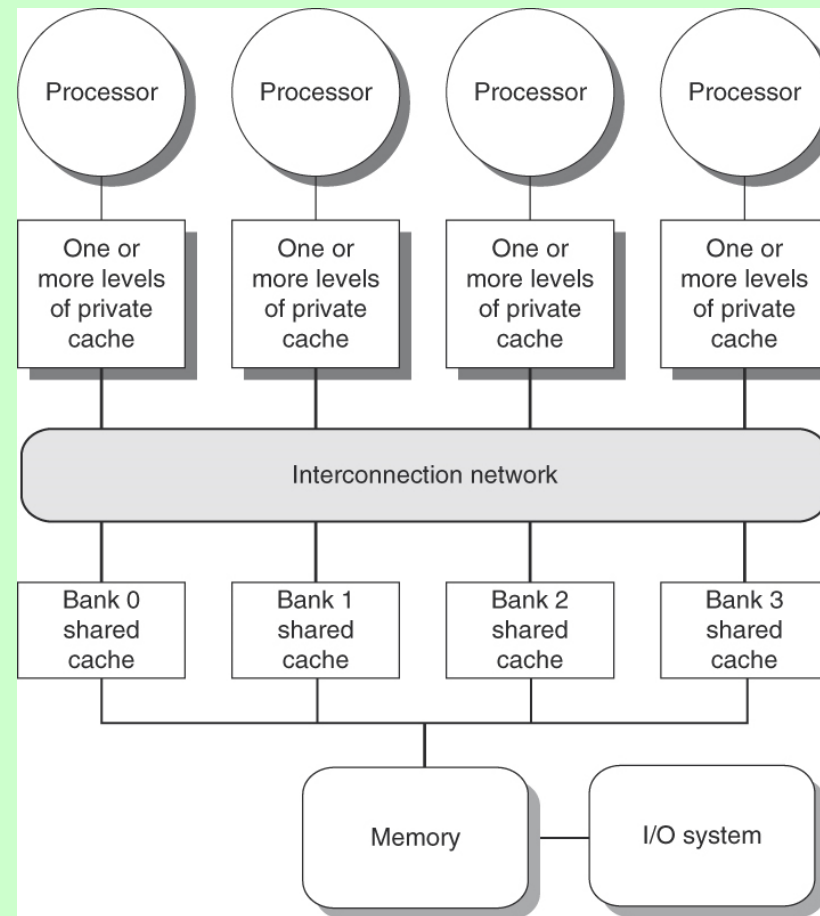
Distributed Shared-Mem. (DSM) Arch.

❑ Shared-Memory Architectures with scalable interconnect and large processor count (from 8 to 32): Large-scale MP designs

❑ **NUMA (Non Uniform Memory Access):** Memory access time depends on processor and memory location.

❑ **CC-NUMA (Cache Coherent NUMA)**

N0	...	N3
P0	...	P3
C0	...	C3
MM0	P0, P1, P2, P3	MM3	P0, P1, P2, P3

Interconnection Network
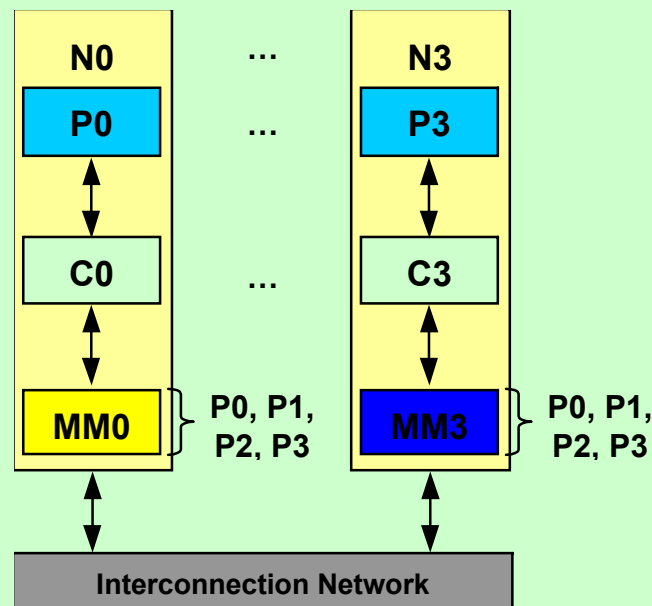
- Each processor core shares the entire memory space
- Access time to the memory attached to the processor's node will be much faster than the access time to remote memories (NUMA)

Distributed Shared-Mem. (DSM) Arch.

❑ Characterized by local memory modules distributed to the nodes, but components of a single global address space, i.e. any processor can access the local memory of any other processor.

❑ Whenever a processor addresses the module of the shared-memory on the local node, the access is much faster than the access to a memory module placed on a remote node (NUMA).

❑ Larger coherent caches reduce the time spent on remote load operations.

❑ Directory-based protocols to support cache coherence

❑ Interconnection network provides low latency and high reliability

Multicore DSM

- ❑ Memory physically distributed among processors (rather than centralized)
- ❑ Non-uniform memory access/latency **(NUMA)**
- ❑ Have a larger processor count (from 8 to 32)
- ❑ Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks
- ❑ Typical multicore: single-chip multiprocessor with memory and I/O attached and an interface to an interconnection network that connects all the nodes
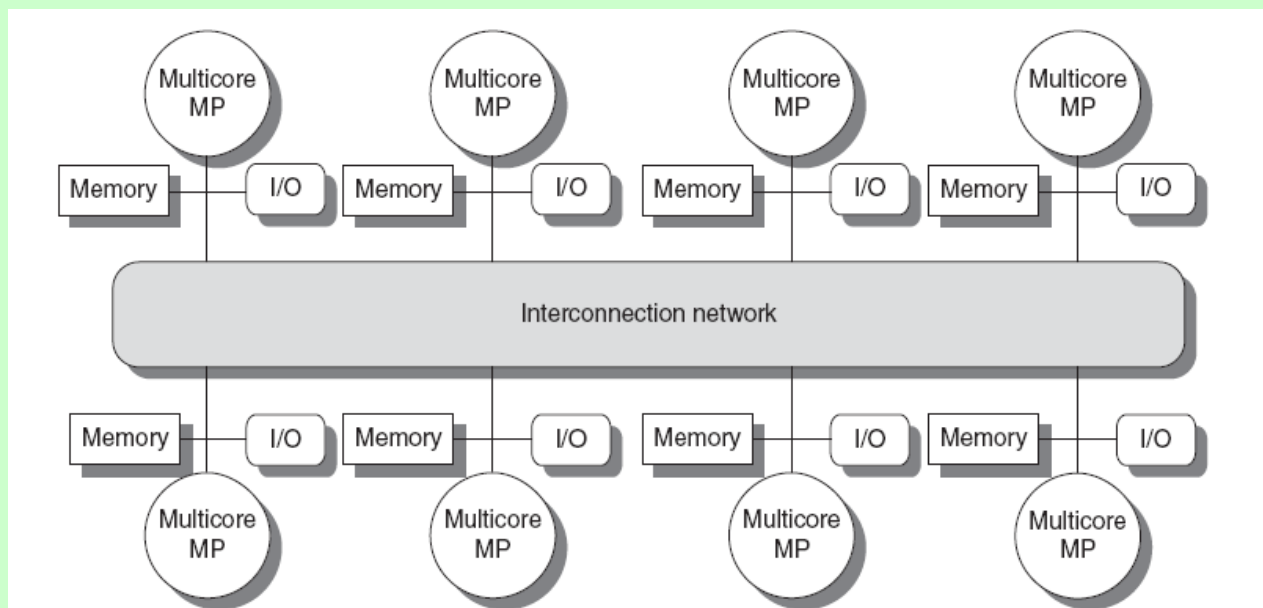
Distributed Memory Architectures

❑ **Clusters** made by individual computers with multiple and private address spaces and connected by a scalable interconnection network (Message passing architectures)

N0	...	N3
P0	...	P3
C0	...	C3
MM0 — P0		MM3 — P3

Interconnection Network
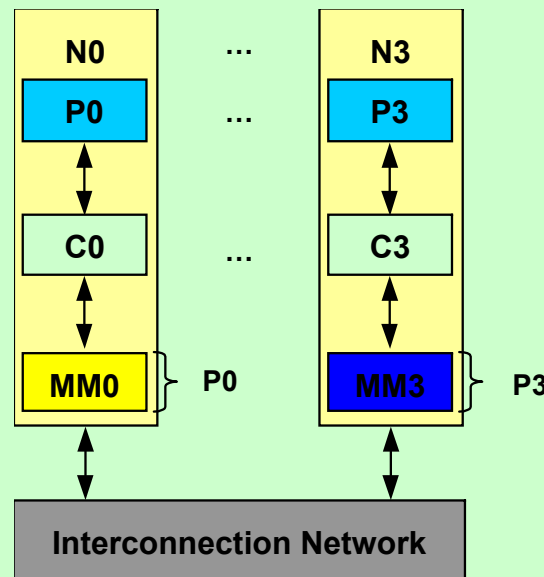
Parallel Framework for Communication

❑ Programming Model:

- ➤ **Multiprogramming** : lots of jobs, no communication
- ➤ **Shared address space:** communicate via load/store memory operations
- ➤ **Message passing:** communicate via send and receive messages
- ➤ **Data Parallel:** several processors operate on several data sets simultaneously and then exchange information globally and simultaneously (shared or message passing): **SIMD** (Single Instruction Multiple Data) architectures

❑ Debate over this topic:
ease of programming, large scaling.

Challenges of Parallel Processing

❑ Limited intrinsic parallelism available in programs

➢ Programmer or compiler help, to expose more parallelism

❑ Relatively high cost of communication

➢ A memory access will cost:

▪ 35 to 50 cycles between cores in the same chip

▪ 100 to 500+ cycles between cores in different chip

➢ It's fundamental to have cache memories

Importance of caches in MPs

- ❑ Caches reduce average access latency
- ❑ Caches reduce contention
- ❑ Because, if several cores want to access the same data, each core will access it from its own cache
- ❑ When shared data are cached, the shared values may be replicated in multiple caches
- ❑ But the use of multiple copies of same data introduces a problem: **cache coherence**
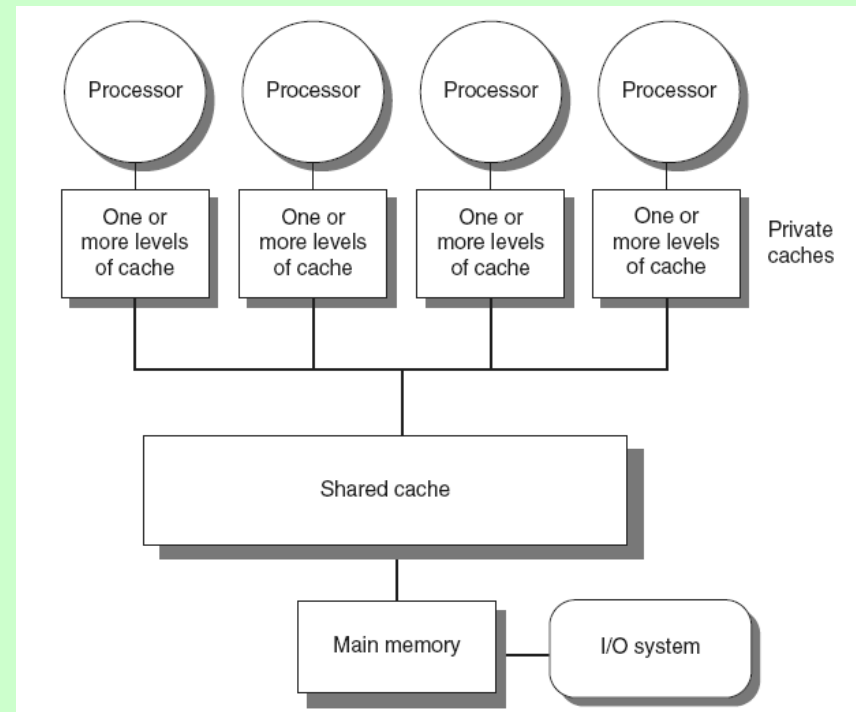
The Problem of Cache Coherence

❑ A program running on multiple processors will normally have **multiple copies** of the same data in several caches.

❑ In a coherent multiprocessor the caches should provide both **migration** and **replication** of shared data items.

❑ **Migration**: movement of data (faster access)

  ➢ A data item can be moved to a local cache and used there in a transparent fashion

  ➢ Reduces both the latency to access a shared data item that is allocated remotely and the bandwidth demand on the shared memory

❑ **Replication** multiple copies of data (reduced contention)

  ➢ For shared data that are being simultaneously read: caches make a copy of the data item in the local cache

  ➢ Reduces both latency of access and contention for a read shared

The Problem of Cache Coherence

❑ Let's consider two processors with **write-through** memory:
Processors may see different values through their caches

Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

Potential Solutions

❑ HW-based solutions to maintain coherency:
Cache Coherence Protocols

❑ Key issue to implementi a cache coherence protocol is tracking the state of any sharing copy of a data block

❑ Two classes of protocols:

➢ Snooping Protocols

➢ Directory-Based Protocols

Cache Coherence Protocols

Two classes of protocols:

- ❑ **Snooping protocols:** each core tracks the sharing status of each block
  - ➢ A cache controller monitors (*snoops*) on the bus, to see what is being requested by another cache
- ❑ **Directory-based protocols:** the sharing status of each block is kept in one location, called the directory
  - ➢ In SMPs: a single directory
  - ➢ In DSMs: multiple, distributed directories (one for each main memory module)

Examples of Snooping Protocols

❏ **Write invalidate protocol**

  ➤ A processor writing an item to memory **invalidates** all other copies

  ➤ Most common protocol

❏ **Write update protocol**

  ➤ A processor writing an item to memory **updates** all other copies

  ➤ More expensive, less common protocol

Cache Coherence vs. Memory Consistency

❑ **Cache Coherence**
  ➢ All reads by any processor must return the most recently written value
  ➢ Writes to the same location by any two processors are seen in the same order by all processors

❑ **Memory Consistency**
  ➢ When a written value will be returned by a read?
  ➢ In other words, in what order must a processor observe the data writes of another processor?
  ➢ If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

Problem of Memory Consistency

❑ Coherence and consistency are **complementary**
  ➢ **Coherence** defines the behavior of reads and writes to the same memory location
  ➢ **Consistency** defines the behavior of reads and writes with respect to accesses to other memory locations
❑ Assumptions for now:
  ➢ A write does not complete (and allow the next write to occur) until all processors have seen the effect of that write
  ➢ The processor does not change the order of any write with respect to any other memory access
    ▪ If a processor writes A followed by B any processor that sees the new value of B must also see the new value of A

Cache Coherence vs. Memory Consistency

❑ A cache coherence protocol ensures that all writes by one processor are eventually visible to other processors, for one memory address

  ➢ i.e., updates are not lost

❑ A memory consistency model gives the rules on when a write by one processor can be observed by a read on another, across different addresses

  ➢ Equivalently, what values can be seen by a load

❑ A cache coherence protocol is not enough to ensure sequential consistency

  ➢ But if sequentially consistent, then caches must be coherent

❑ Combination of cache coherence protocol plus processor memory reorder buffer used to implement a given architecture's memory consistency model

Communication Models

Multiple independent instruction streams, two main kinds:

❑ **Shared Memory**

  ➢ Processors communicate with shared address space via load/store operations: hardware global cache coherence

  ➢ Easy on small-scale multicores

  ➢ Advantages:

    ▪ Ease of programming

    ▪ Lower latency

    ▪ Easier to use hardware controlled caching

❑ **Message Passing**

  ➢ Processors have private memories, communicate via send/receive messages: no hardware global cache coherence

  ➢ Advantages:

    ▪ Less hardware, easier to design

    ▪ Focuses attention on costly non-local operations

51

Shared Memory Model
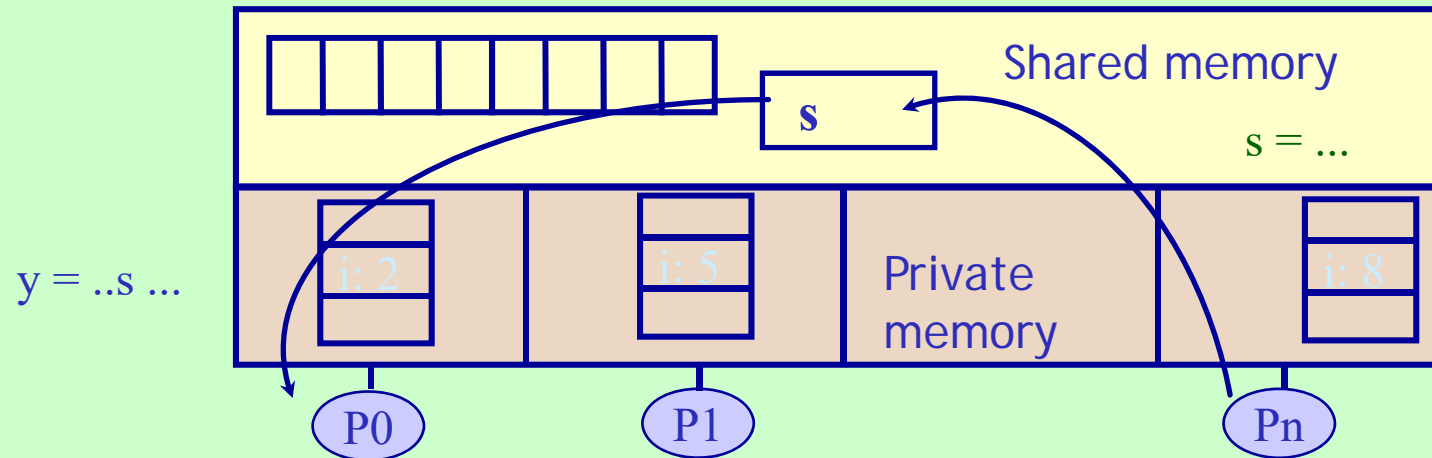
- Program is a collection of threads of control.
  - Can be created dynamically, mid-execution, in some languages
- Each thread has a set of private variables, e.g., local stack variables
- Also a set of shared variables, e.g., static variables, shared common blocks, or global heap.
  - Threads communicate implicitly by writing and reading shared variables.
  - Threads coordinate by synchronizing on shared variables

Message Passing Model

Private memory

s: 12   s: 14   receive Pn,s   s: 11

i: 2   i: 3   i: 1

P0   y = ..s ...   P1   send P1,s   Pn

Network
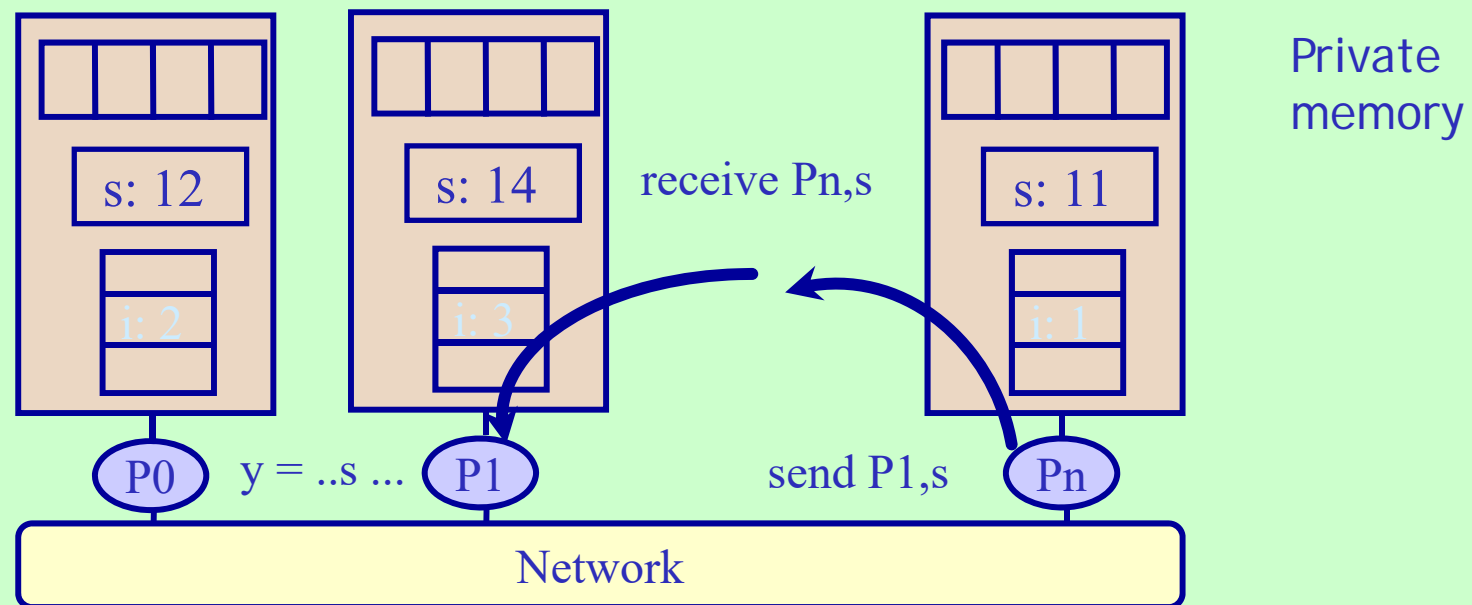
❑ Program consists of a collection of named processes.
  ➢ Usually fixed at program startup time
  ➢ Thread of control plus local address space -- NO shared data.
  ➢ Logically shared data is partitioned over local processes.
❑ Processes communicate by explicit send/receive pairs
  ➢ Coordination is implicit in every communication event.
  ➢ MPI (Message Passing Interface) is the most commonly used SW

Which is better? SM or MP?

❑ Which is better, Shared Memory or Message Passing?

➢ It depends on the program!

➢ Both are "communication Turing complete"

▪ i.e. can build Shared Memory with Message Passing and vice-versa

Pros/Cons of Shared Memory

❑ Advantages of Shared Memory:
  ➢ Implicit communication (via loads/stores)
  ➢ Low latency
  ➢ Low overhead when shared data are cached

❑ Disadvantages of Shared Memory:
  ➢ Complex to build in a way that scales well
  ➢ Requires synchronization operations
  ➢ Hard to control data placement within caching system

Pros/Cons of Message Passing

❑ Advantages of Message Passing

- ➢ Explicit Communication (via sending/receiving messages)
- ➢ Easier to control data placement (no automatic caching)

❑ Disadvantages of Message Passing

- ➢ Message passing overhead can be quite high
- ➢ More complex to program
- ➢ Introduces question of reception technique (interrupts/polling)

Data Parallel Model

- Operations can be performed in parallel on each element of a large regular data structure, such as an array
- The Control Processor broadcast to many replicated PEs
- Condition flag per PE so that can skip
- **Data distributed in each memory**
- Early 1980s VLSI => **SIMD (Single Instruction Multiple data)** rebirth
- Data parallel programming languages lay out data to processor

Data Parallel Model

❑ SIMD led to Data Parallel Programming languages

❑ SIMD programming model led to
Single Program Multiple Data (SPMD) model

➢ All processors execute identical program

❑ Data parallel programming languages still useful, do communication all at once:
"Bulk Synchronous" phases in which all communicate after a global barrier

Convergence in Parallel Architecture

❑ Complete computers connected to scalable network via communication assist

❑ Different programming models place different requirements on communication

➢ **Shared address space:** tight integration with memory to capture memory events that interact with others + to accept requests from other nodes

➢ **Message passing:** send messages quickly and respond to incoming messages: tag match, allocate buffer, transfer data, wait for receive posting

➢ **Data Parallel:** fast global synchronization

Characteristics of four commercial multicores

Feature	AMD Opteron 8439	IBM Power 7	Intel Xenon 7560	Sun T2
Transistors	904M	1200 M	2300 M	500 M
Nominal Power	137 W	140 W	130 W	95 W
Max cores/chip	6	8	8	8
Multithreading	No	SMT	SMT	Fine-grained
Threads/core	1	4	2	8
Clock rate	2.8 GHz	4.1 GHz	2.7 GHz	1.6 GHz
Multicore coherence	Snooping	Directory	Directory	Directory

Prof. Cristina Silvano, Politecnico di Milano

Multicores

❑ Cores have become the new building block

❑ Intel, with a single architecture design (the Nehalem core) has created all of its multiprocessor platforms:

  ➢ Xeon, i3, i5, i7 are all based on the same core