

# Embedded Parallel Computing

## Lecture 4 -

### Multiple-instruction multiple-data streams (MIMD) parallel architectures

Tomas Nordström

Course webpage: <<http://www.hh.se/DO8003>>

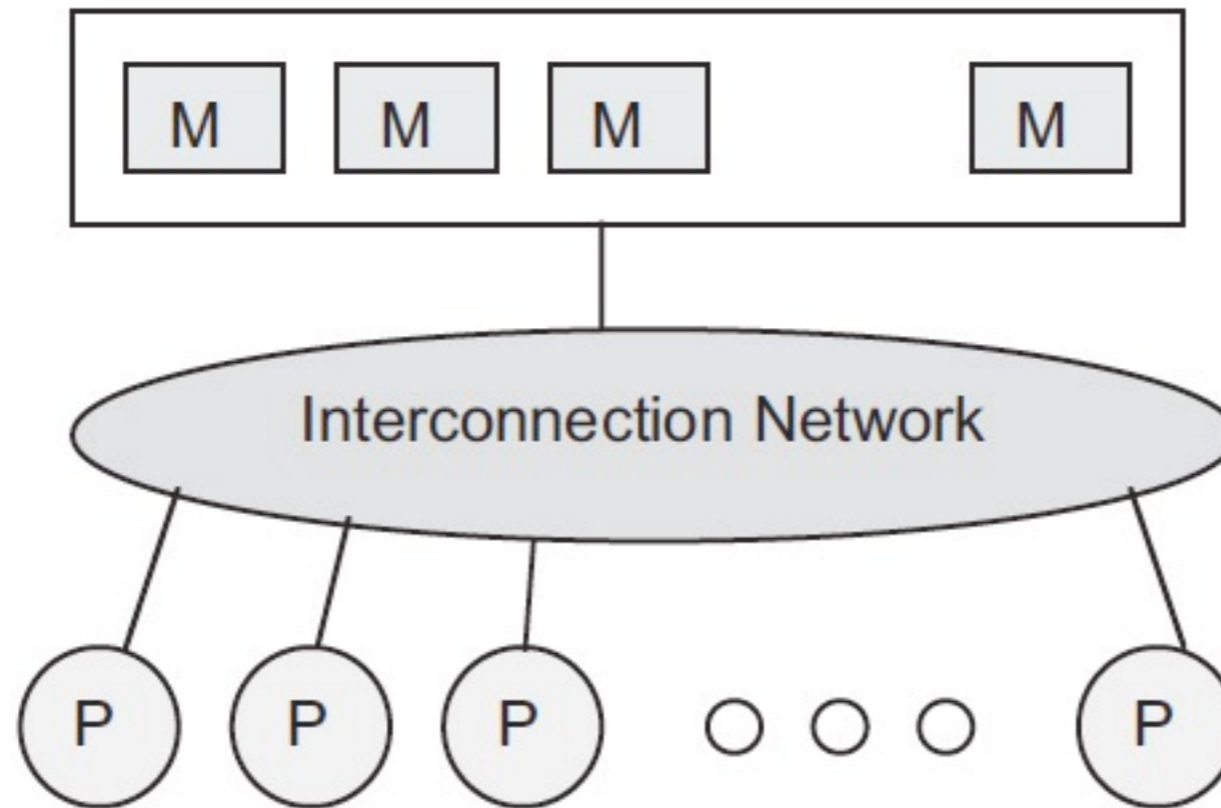
Course responsible and examiner: Tomas Nordström,  
[Tomas.Nordstrom@hh.se](mailto:Tomas.Nordstrom@hh.se); Room E313; Tel: +46 35 16 7334

# Outline

## MIMD

- Shared Memory [Ch4 in ACAPP]
- Distributed Memory [Ch5 in ACAPP]
- You may pay less attention to Sections 3.5, 4.4.3 - 4.4.5, 4.5.1 and 4.5.2.

# Shared-Memory MIMD

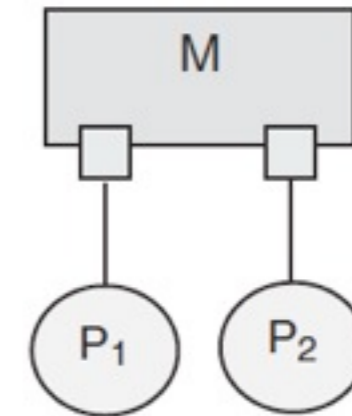


**Figure 4.1** Shared memory systems.

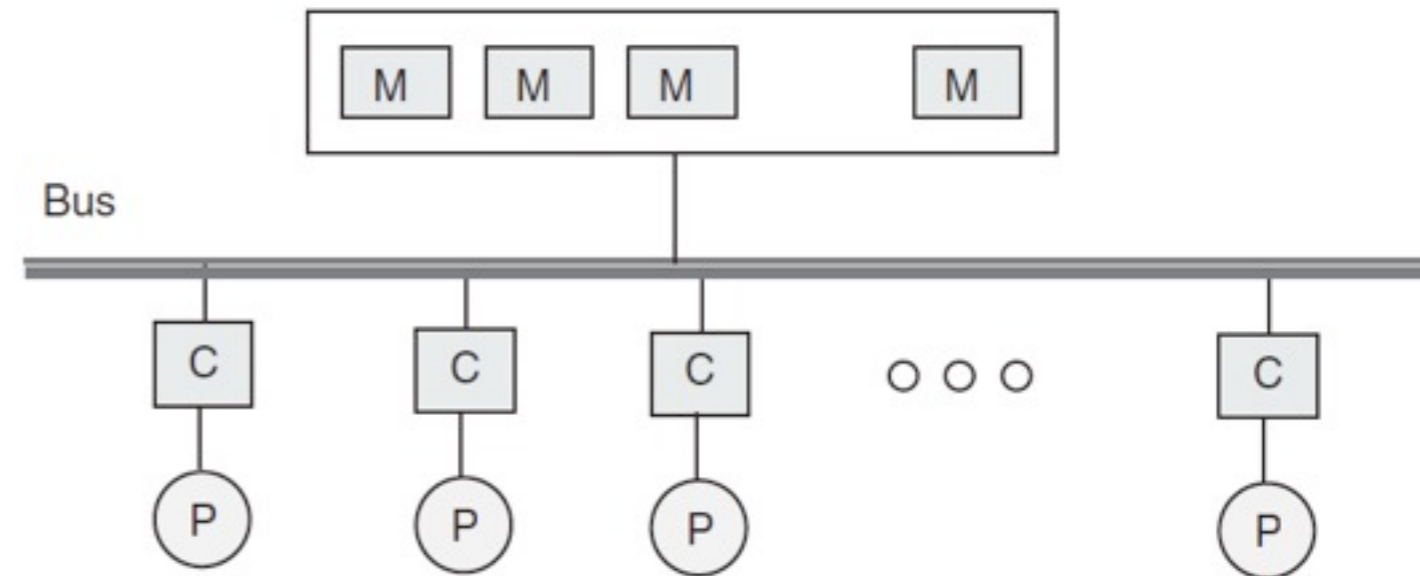
# Classification

## Shared-Memory MIMD

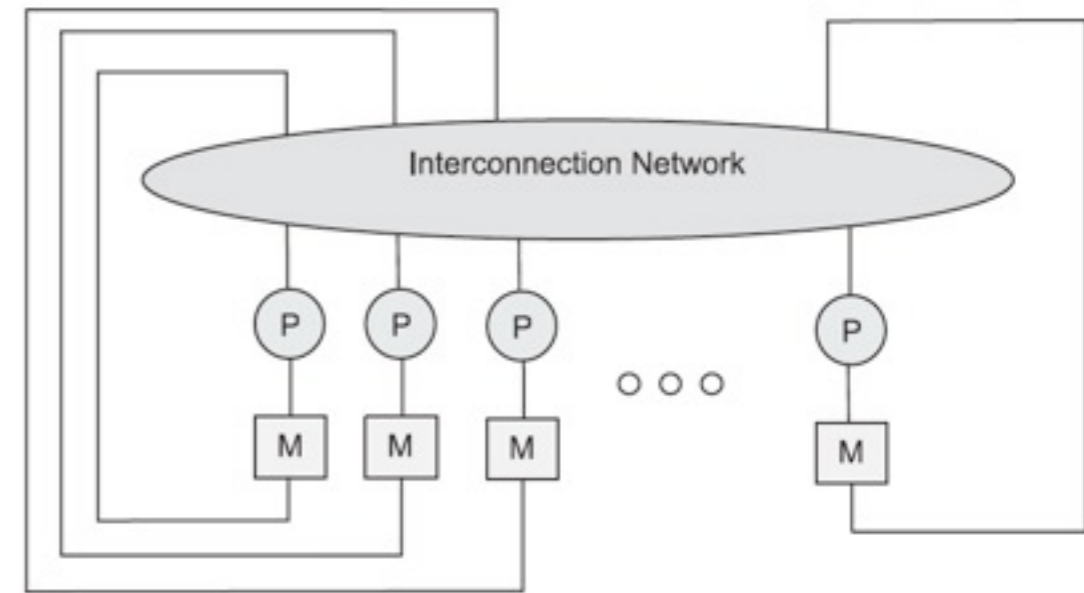
- Uniform Memory Access (UMA)
  - ▶ Multi-port memory



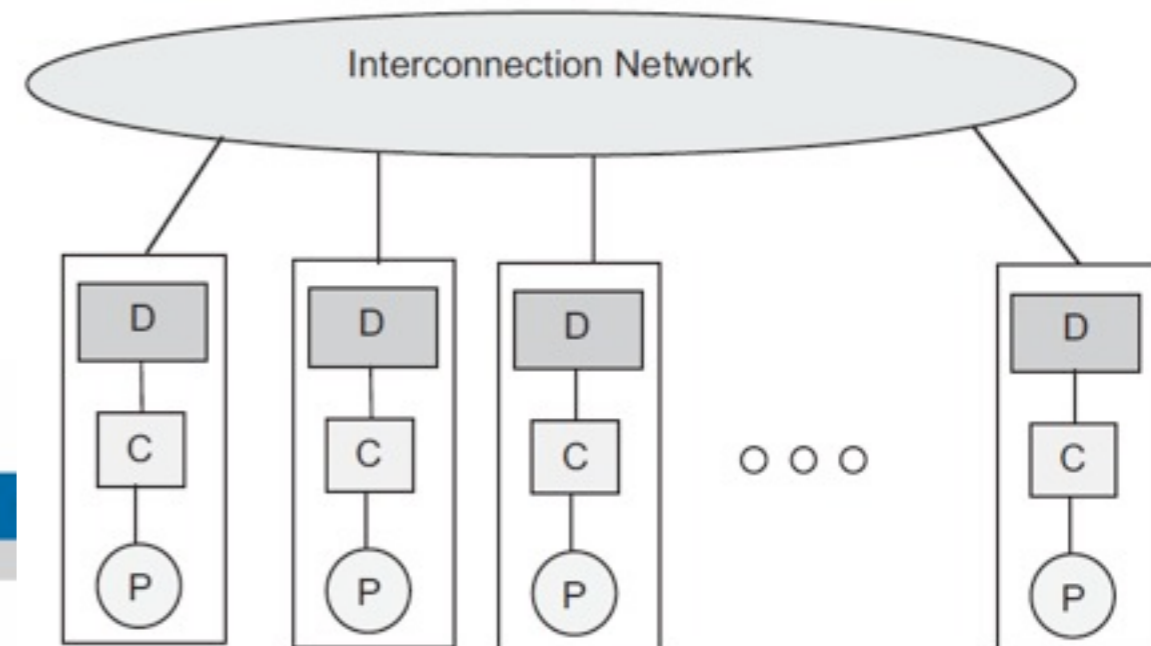
- ▶ Bus-based UMA



- Nonuniform Memory Access (NUMA)



- Cache-Only Memory Architecture



# Bus-based Symmetric Multiprocessors

- Simplest shared memory architecture
  - ▶ No expensive multiport memory or interface circuitry (and still not message passing)
  - ▶ But the bus can be a bottleneck!
    - Alleviate the bus contention problem with caches
    - Maximum number of processors (with cache) that the bus can support:

$$N \leq \frac{BI}{(1-h)V}$$

$N$  = number of processors

$h$  = cache hit rate

$B$  = bus bandwidth (cycles/s)

$I$  = processor duty cycle (fetches/cycle)

$V$  = peak processor speed (fetches/s)

# Basic Cache Coherence Methods

- Cache-Memory Coherence

- write-through
- write-back

Serial	Event	Write-Through		Write-Back	
		Memory	Cache	Memory	Cache
1		X		X	
2	P reads X	X	X	X	X
3	P updates X	X'	X'	X	X'

- Cache-Cache Coherence

- ▶ write-invalidate
- ▶ write-update

Serial	Event	Write-Update		Write-Invalidate	
		P's Cache	Q's Cache	P's Cache	Q's Cache
1	P reads X	X		X	
2	Q reads X	X	X	X	X
3	Q updates X	X'	X'	INV	X'
4	Q updates X'	X''	X''	INV	X''

- Shared Memory System Coherence

# Snooping Protocols

- Watch the bus (snoop) and carry out appropriate coherency command when needed
  - ▶ Write-Invalidate with Write-Through

**TABLE 4.3 Write-Invalidate Write-Through Protocol**

State	Description
Valid [VALID]	The copy is consistent with global memory.
Invalid [INV]	The copy is inconsistent.
Event	Actions
Read-Hit	Use the local copy from the cache.
Read-Miss	Fetch a copy from global memory. Set the state of this copy to Valid.
Write-Hit	Perform the write locally. Broadcast an Invalid command to all caches. Update the global memory.
Write-Miss	Get a copy from global memory. Broadcast an invalid command to all caches. Update the global memory. Update the local copy and set its state to Valid.
Block replacement	Since memory is always consistent, no write-back is needed when a block is replaced.

# Snooping Protocols cont.

## Write-Invalidate with Write-Through



- ▶ Memory is always consistent

**TABLE 4.3 Write-Invalidate Write-Through Protocol**

State	Description
Valid [VALID]	The copy is consistent with global memory.
Invalid [INV]	The copy is inconsistent.
Event	Actions
Read-Hit	Use the local copy from the cache.
Read-Miss	Fetch a copy from global memory. Set the state of this copy to Valid.
Write-Hit	Perform the write locally. Broadcast an Invalid command to all caches. Update the global memory.
Write-Miss	Get a copy from global memory. Broadcast an invalid command to all caches. Update the global memory. Update the local copy and set its state to Valid.
Block replacement	Since memory is always consistent, no write-back is needed when a block is replaced.

# Snooping Protocols cont.

## Write-Invalidate with Write-Back (ownership protocol)

- ▶ Many can read from their caches until someone writes and take ownership of the block

TABLE 4.5 Write-Invalidate Write-Back Protocol

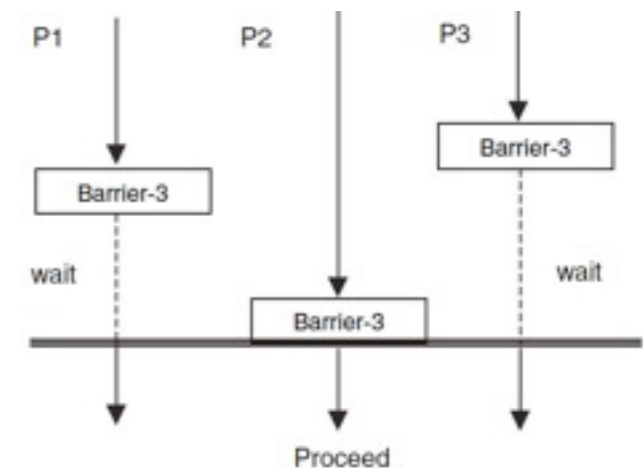
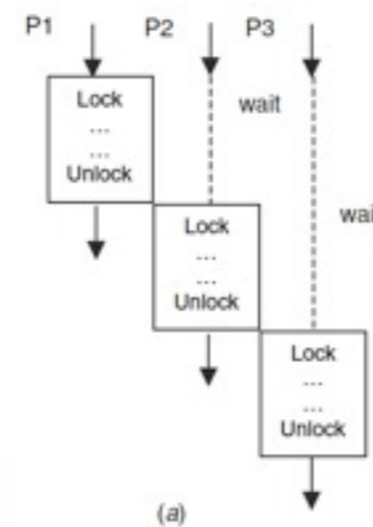
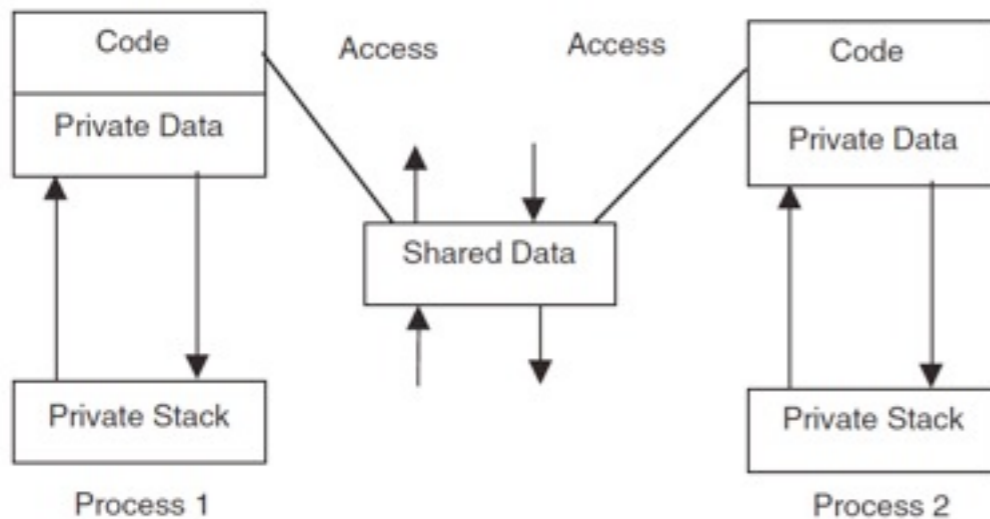
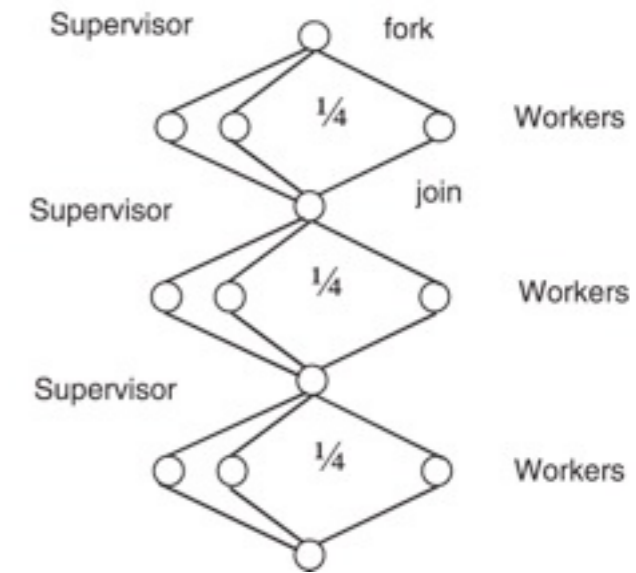
State	Description
Shared (Read-Only) [RO]	Data is valid and can be read safely. Multiple copies can be in this state.
Exclusive (Read-Write) [RW]	Only one valid cache copy exists and can be read from and written to safely. Copies in other caches are invalid.
Invalid [INV]	The copy is inconsistent.
Event	Action
Read-Hit	Use the local copy from the cache.
Read-Miss	If no Exclusive (Read-Write) copy exists, then supply a copy from global memory. Set the state of this copy to Shared (Read-Only). If an Exclusive (Read-Write) copy exists, make a copy from the cache that set the state to Exclusive (Read-Write), update global memory and local cache with the copy. Set the state to Shared (Read-Only) in both caches.
Write-Hit	If the copy is Exclusive (Read-Write), perform the write locally. If the state is Shared (Read-Only), then broadcast an Invalid to all caches. Set the state to Exclusive (Read-Write).
Write-Miss	Get a copy from either a cache with an Exclusive (Read-Write) copy, or from global memory itself. Broadcast an Invalid command to all caches. Update the local copy and set its state to Exclusive (Read-Write).
Block replacement	If a copy is in an Exclusive (Read-Write) state, it has to be written back to main memory if the block is being replaced. If the copy is in Invalid or Shared (Read-Only) states, no write-back is needed when a block is replaced.

# Directory Based Protocols

- A Central directory maintains information about all blocks in a central data structure.
- Updates are then only concerning the affected caches
- While Central directory includes everything in one location, it becomes a bottleneck and suffers from large search time. To alleviate this problem, the same information can be handled in a distributed fashion by allowing each memory module to maintain a separate directory.

# Shared Memory Programming

- Main programming constructs:
  - ▶ Task (process, threads) creation
  - ▶ Communication
  - ▶ Synchronization



# Summary

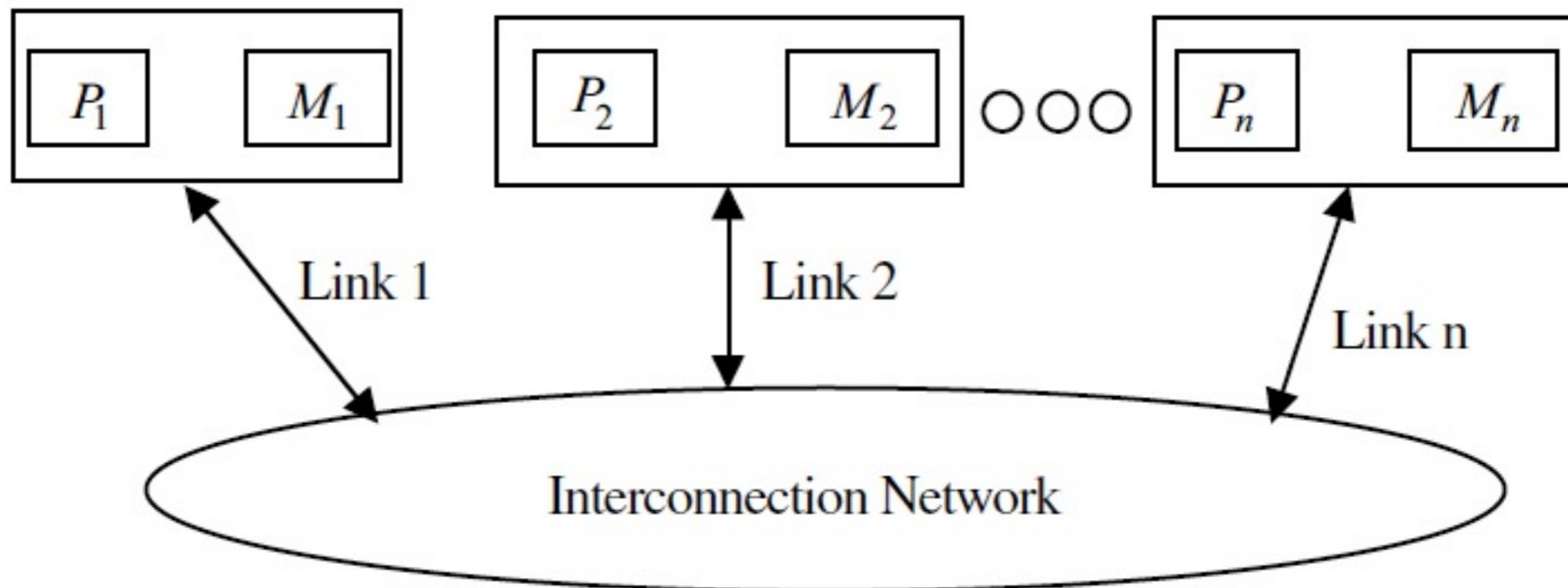
## Shared Memory Architectures

- Shared memory multiprocessors are usually **bus-based** or **switch-based**. In all cases, each processor has **equal access to the global memory shared by all processors**.
- **Communication** among processors is achieved by **writing to and reading from memory**.
- **Synchronization** among processors is achieved using **locks** and **barriers**.
- The main challenges of shared memory systems are performance degradation due to **contention and cache coherence problems**.
- The performance of a shared memory system becomes an issue when the **interconnection network** connecting the processors to global memory becomes a **bottleneck**.
- **Local caches** are typically used to alleviate the bottleneck problem.
- However, **scalability remains the main drawback** of a shared memory system.

# Summary cont.

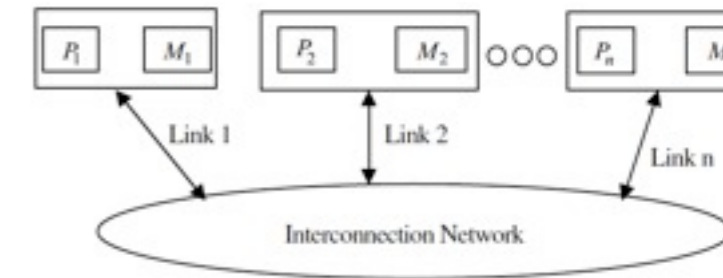
- The introduction of  **caches**  has created a  **consistency problem**  among caches and between memory and caches.
- **Cache coherence schemes**  can be categorized into two main categories:  **snooping protocols**  and  **directory-based protocols** .
  - ▶  **Snooping protocols**  are based on watching bus activities and carry out the appropriate coherency commands when necessary.
  - ▶ In cases when the broadcasting techniques used in snooping protocols are unpractical, coherency commands need to be sent to only those caches that might be affected by an update.
  - ▶ This is the idea behind  **directory-based protocols** . *Cache coherence protocols that somehow store information on where copies of blocks reside are called directory schemes.*

# Message Passing MIMD



- No global memory
- Move data between processors by *message passing*. (Typically as send/receive pairs, which needs to be explicitly programmed)

- ICN normally a static network.
  - ▶ In particular: hypercubes and nearest neighbor 2D and 3D mesh have been suggested.
- Important parameters: *link bandwidth* and *network latency*

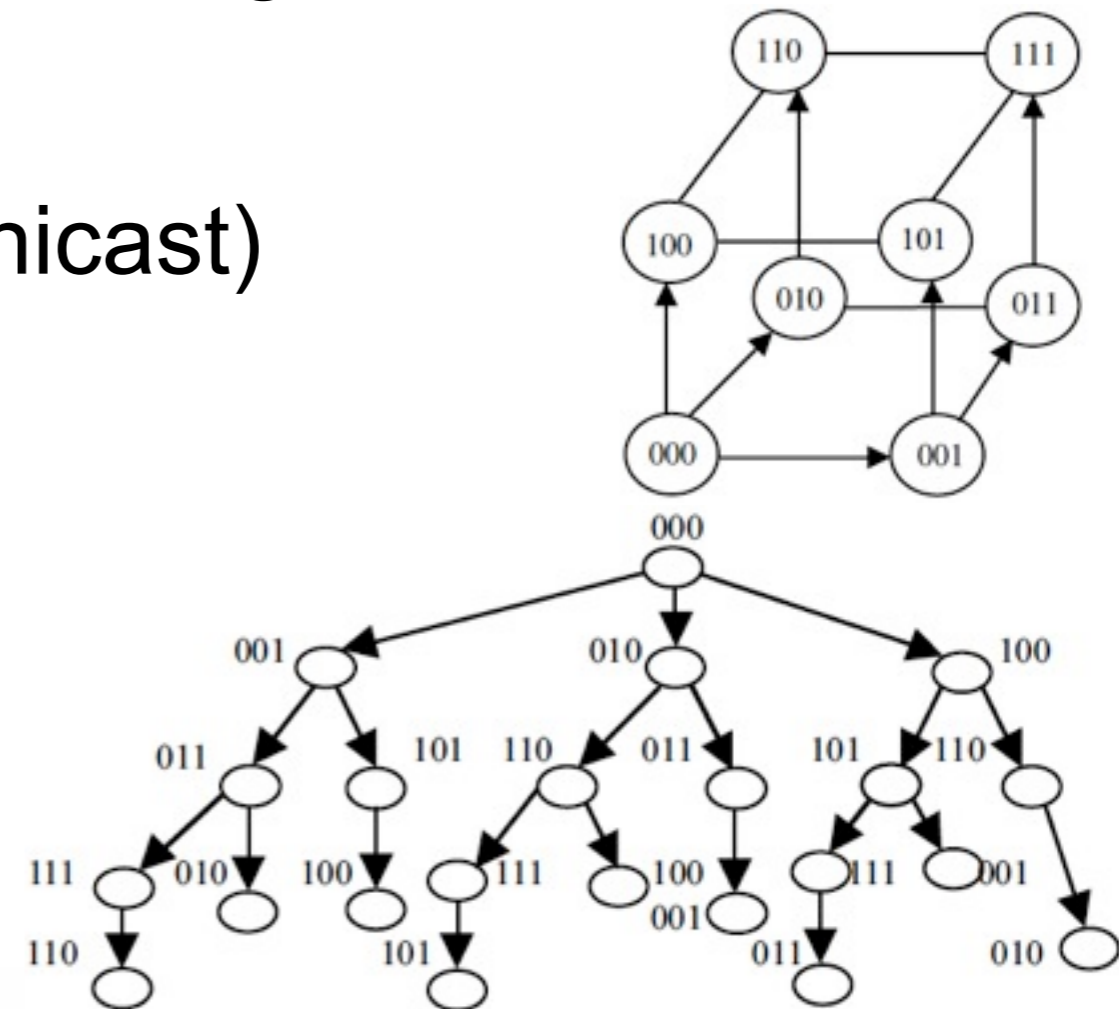


# Programming Concept

- Program divided into concurrent processes that can be distributed among the processors
- Data exchanged among processors can not be shared, it is rather copied (using send/receive)
- Elimination of the need for synchronization constructs like semaphores!
- Process granularity

# Routing

- Adaptive vs Deterministic path selection
- Centralized vs Distributed routing decisions
- Message types
  - ▶ one-to-one (point-to-point or unicast)
  - ▶ broadcast (one-to-all)
  - ▶ multicast (one-to-many)



CERES

CENTRE FOR RESEARCH ON EMBEDDED SYSTEMS

# Routing (Potential) Problems

- Deadlock
  - ▶ Two messages each hold the resources required by the other in order to move
- Livelock
  - ▶ Messages keep going around the network and never reaches destination
- Starvation
  - ▶ A process wants to inject a message into the network but it is never allowed to enter

# Switching Mechanisms

- Switching mechanisms refer to the mechanisms used to remove data from an input channel and place it on an output channel.
- Examples: *store-and-forward*, *circuit-switching*, *virtual cut-through*, *wormhole*, and *pipelined circuit-switching*.

# Switching Mechanism

## Circuit-Switching

- In circuit-switching networks, the path between the source and destination is first determined, all links along that path are reserved, and no buffers are needed in each node
- The source and destination are guaranteed a certain bandwidth and maximum latency when communication is established between them

# Switching Mechanism

## Store-and-Forward

- The main idea is to offer dynamic bandwidth allocation to messages as they flow through the network, thus avoiding the main drawback of the circuit switching mechanism. Two main types of store-and-forward networks are common. These are packet-switched and virtual cut-through networks.

# Switching Mechanism

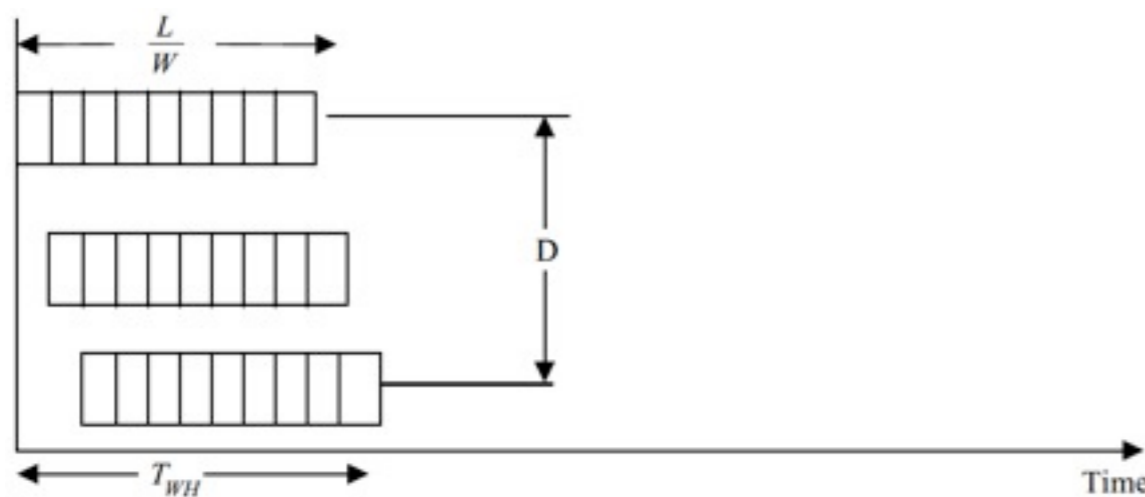
## S-a-F Packet Switched

- Each message is divided into smaller fixed size parts, called packets, before being transmitted. Each node must contain enough buffers to hold received packets before transmitting them.
- A complete path from source to destination may not be available at the start of transmission. As links become available, packets are moved from node to node until they reach the destination node.
- Since packets are routed separately through the network, they may *follow different paths to the destination node*. This may lead to *packets arriving out of order* at the destination.

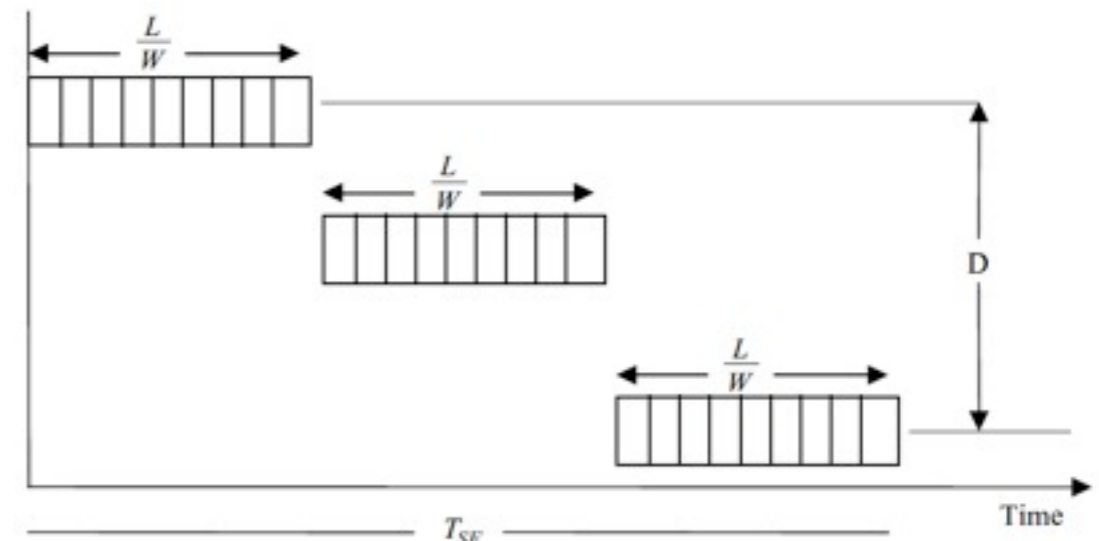
# Switching Mechanism

## S-a-F Virtual Cut-Through

- A packet is stored at an intermediate node only if the next required channel is busy
- The packet is sent out to the adjacent node towards its destination before it is completely received!



*Virtual cut-through*



*packet switched*

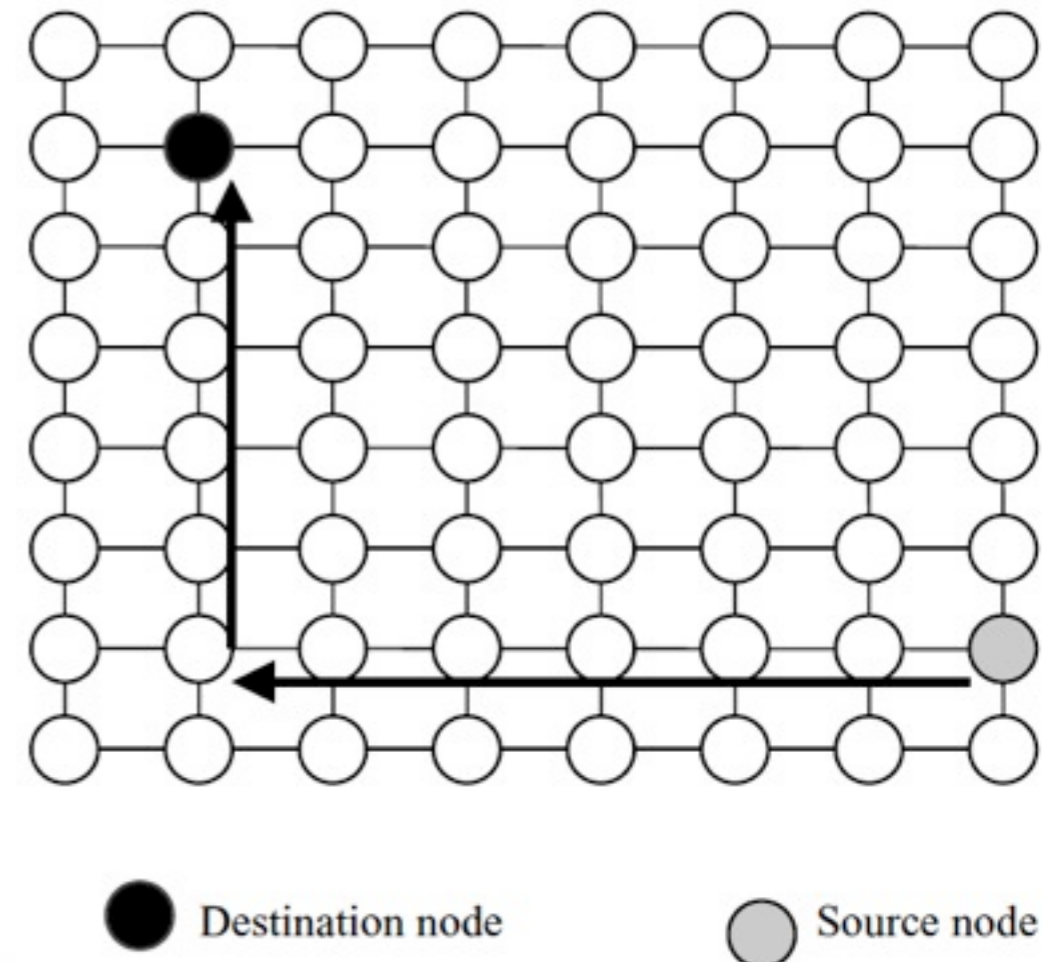
# Switching Mechanism

## Wormhole Routing

- In order to reduce the size of the required buffers and decrease the incurred network latency for Virtual Cut-Through, a technique called *wormhole routing* has been introduced
- Here, a packet is divided into smaller units called *flits* (flow control bits).
  - ▶ These flits move in a pipeline fashion with a header flit leading the way to the destination node.
  - ▶ When the header flit is blocked due to network congestion, the remaining flits are also blocked.
  - ▶ Only a buffer that can store a flit is required for a successful operation of the wormhole routing technique.

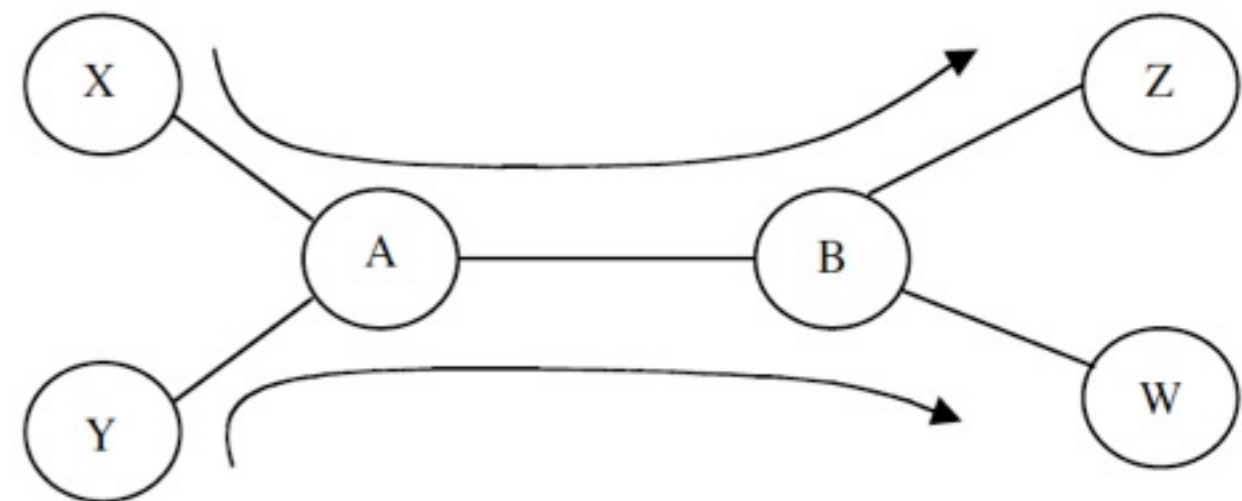
# Example Wormhole Routing in Mesh Networks

- Many routing techniques have been suggested: dimension-ordered, dimension reversal, turn model, and message flow model
- **Dimension-ordered (X-Y) Routing:**
- Do each dimension at a time. This strict monotonic order -> dead-lock free
- $gx = dx - sx$ ;  $gy = dy - sy$
- place the  $gx$  and  $gy$  in the first two flits and decrement them as they pass through a node until they become 0.



# Virtual channels

- The principle of virtual channel was introduced in order to allow the design of deadlock-free routing algorithms.
- Virtual channels provide an inexpensive method to increase the number of logical channels without adding more wires.
- Each physical link is actually divided into a number of unidirectional virtual channels. (Time sharing)



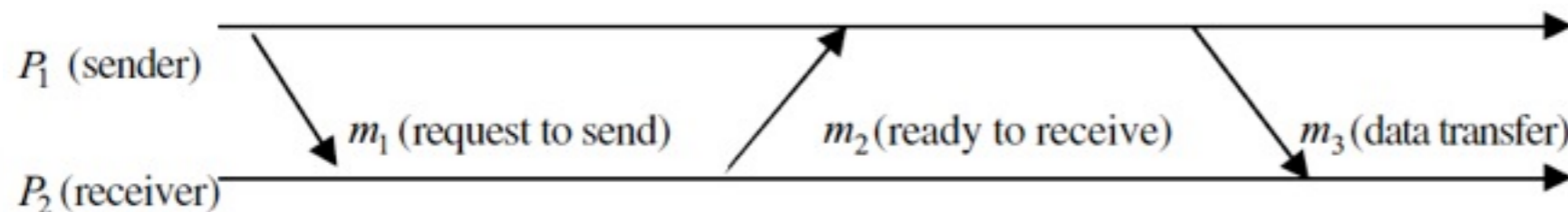
# Switching Mechanism Comparison



Switching Mechanism	Advantages	Disadvantages
Circuit switching	<ol style="list-style-type: none"><li>1. Suitable for long messages</li><li>2. Deadlock-free</li></ol>	Wasting of bandwidth
Store-and-forward	<ol style="list-style-type: none"><li>1. Simple</li><li>2. Suitable for interactive traffic</li><li>3. Bandwidth on demand</li></ol>	<ol style="list-style-type: none"><li>1. Buffer for every packet</li><li>2. Potential long latency</li><li>3. Potential deadlock</li></ol>
Virtual cut-through	<ol style="list-style-type: none"><li>1. Good for long messages</li><li>2. Possible deadlock avoidance</li><li>3. Elimination of data-link protocol</li></ol>	<ol style="list-style-type: none"><li>1. Need for multiple message buffers</li><li>2. Wasting of bandwidth</li><li>3. Mainly used with profitable routing</li></ol>
Wormhole	<ol style="list-style-type: none"><li>1. Good for long messages</li><li>2. Reduced need for buffering</li><li>3. Reduced effect of path length</li></ol>	<ol style="list-style-type: none"><li>1. Possibility for deadlock</li><li>2. Inability to support backtracking</li></ol>

# Message Passing Programming Models

- A message passing architecture uses a set of primitives that allows processes to communicate with each other. These include the *send*, *receive*, *broadcast*, and *barrier primitives*.
- The basic programming model used in message passing architectures is based on the idea of *matching a send request on one processor with a receive request on another*.
  - ▶ In such scheme, send and receive are *blocking*; that is, send blocks until the corresponding receive is executed before data can be transferred.
  - ▶ Implementation of the send/ receive among processes requires a three-way protocol



# Message Passing vs Shared Memory Architectures

- Your reflections on Chapter 5.7?

# Summary

- Shared memory systems may be easier to program, but are difficult to scale up to a large number of processors. If scalability to larger and larger systems (as measured by the number of processing units) was to continue, systems had to use message passing techniques.
- *It is apparent that message passing systems are the only way to efficiently increase the number of processors managed by a multiprocessor system.*
- There are, however, a number of problems associated with message passing systems.
  - ▶ These include communication overhead and difficulty of programming.

# Questions

- Study-support questions



# Links

Shared-memory systems <[http://en.wikipedia.org/wiki/Shared\\_memory](http://en.wikipedia.org/wiki/Shared_memory)>;  
SMP - Symmetric Multiprocessor System <[http://en.wikipedia.org/wiki/Symmetric\\_multiprocessor](http://en.wikipedia.org/wiki/Symmetric_multiprocessor)>  
Non-Uniform Memory Access <[http://en.wikipedia.org/wiki/Non-Uniform\\_Memory\\_Access](http://en.wikipedia.org/wiki/Non-Uniform_Memory_Access)>  
Memory coherence <[http://en.wikipedia.org/wiki/Memory\\_coherence](http://en.wikipedia.org/wiki/Memory_coherence)>  
Cache coherence <[http://en.wikipedia.org/wiki/Cache\\_coherence](http://en.wikipedia.org/wiki/Cache_coherence)>  
  
Message passing systems <[http://en.wikipedia.org/wiki/Message\\_passing](http://en.wikipedia.org/wiki/Message_passing)>;