BITS Pilani

Hyderabad Campus

SS ZG 526: Distributed Computing (CS1)

Introduction

Prof. Geetha

Associate Professor, Dept. of Computer Sc. & Information Systems
BITS Pilani Hyderabad Campus

geetha@hyderabad.bits-pilani.ac.in

Course Scope & Objectives

No	Objective
CO1	This course will cover various hardware architectures for building distributed systems, and their communication models.
CO2	This will help students understand the design aspects of various software applications that can be deployed on various distributed systems.
CO3	This will provide an understanding of the complexities and resource management issues that are critical in a large distributed system.
CO4	This course will cover algorithmic aspects of building/designing distributed systems in domains like IoT, P2P, Cluster, Grid computing etc.

Course Material

Text Book

No.	Author(s), Title, Edition, Publishing House
T1	Ajay D. Kshemkalyani, and Mukesh Singhal “Distributed Computing: Principles, Algorithms, and Systems”, Cambridge University Press, 2008 (Reprint 2013).

Reference Books

No.	Author(s), Title, Edition, Publishing House
R1	John F. Buford, Heather Yu, and Eng K. Lua, “P2P Networking and Applications”, Morgan Kaufmann, 2009 Elsevier Inc.
R2	Kai Hwang, Geoffrey C. Fox, and Jack J. Dongarra, “Distributed and Cloud Computing: From Parallel processing to the Internet of Things”, Morgan Kaufmann, 2012 Elsevier Inc.

Course Evaluation Scheme

No.	Name	Type	Duration	Weight	Day, Date, Session, Time
EC-1	Quiz-I/ Assignment-I	Online	-	5%	September 10-20, 2020
	Quiz-II	Online		5%	October 20-30, 2020
	Assignment-II	Online		10%	November 10-20, 2020
EC-2	Mid-Semester Test	Closed Book	2 hours	30%	Saturday, 10/10/2020 (AN) 2 PM – 4 PM
EC-3	Comprehensive Exam	Open Book	3 hours	50%	Saturday, 28/11/2020 (AN) 2 PM – 5 PM

Distributed Computing

Modular Overview

- **M1: Introduction to Distributed Computing**
- **M2: Logical Clocks & Vector Clocks**
- **M3: Global state and snapshot recording algorithms**
- **M4: Message ordering and Termination detection**
- **M5: Distributed Mutual Exclusion**
- **M6: Deadlock Detection**
- **M7: Consensus and Agreement Algorithm**
- **M8: Peer to Peer Computing and Overlay graphs**
- **M9: Cluster computing, Grid Computing**
- **M10: Internet of Things**

What is a distributed system?

- A computing environment which deals with all forms of computing, information access, and information exchange across multiple processing platforms connected by computer networks

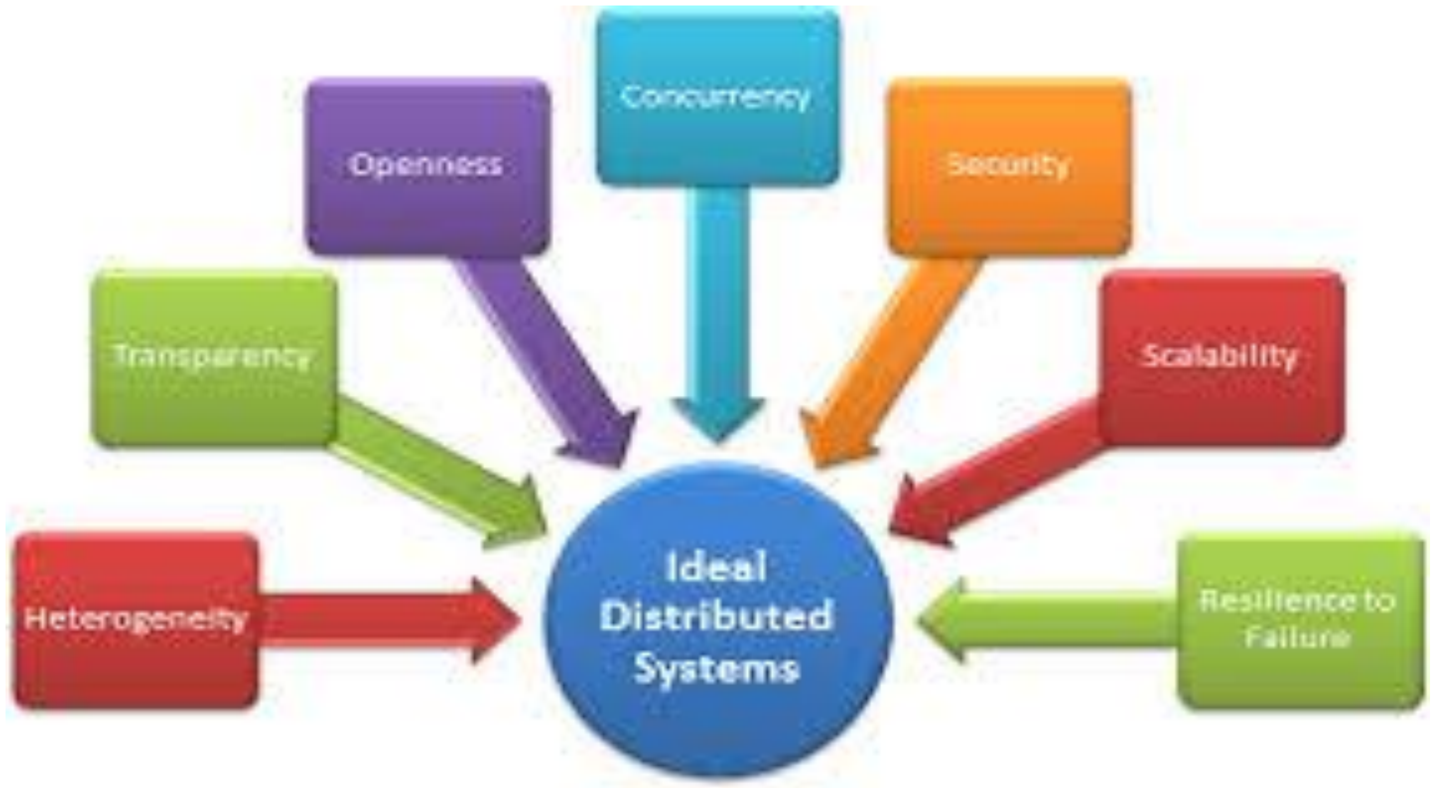
Ex: Banking systems; Communication systems, Information systems (WWW),Manufacturing and process control, Inventory Systems, General purpose automation systems etc,

- So, what is a good definition of a distributed system?
- A distributed system is a collection of independent entities that cooperate to solve a problem that cannot be individually solved.
- Distributed systems have been in existence since many years

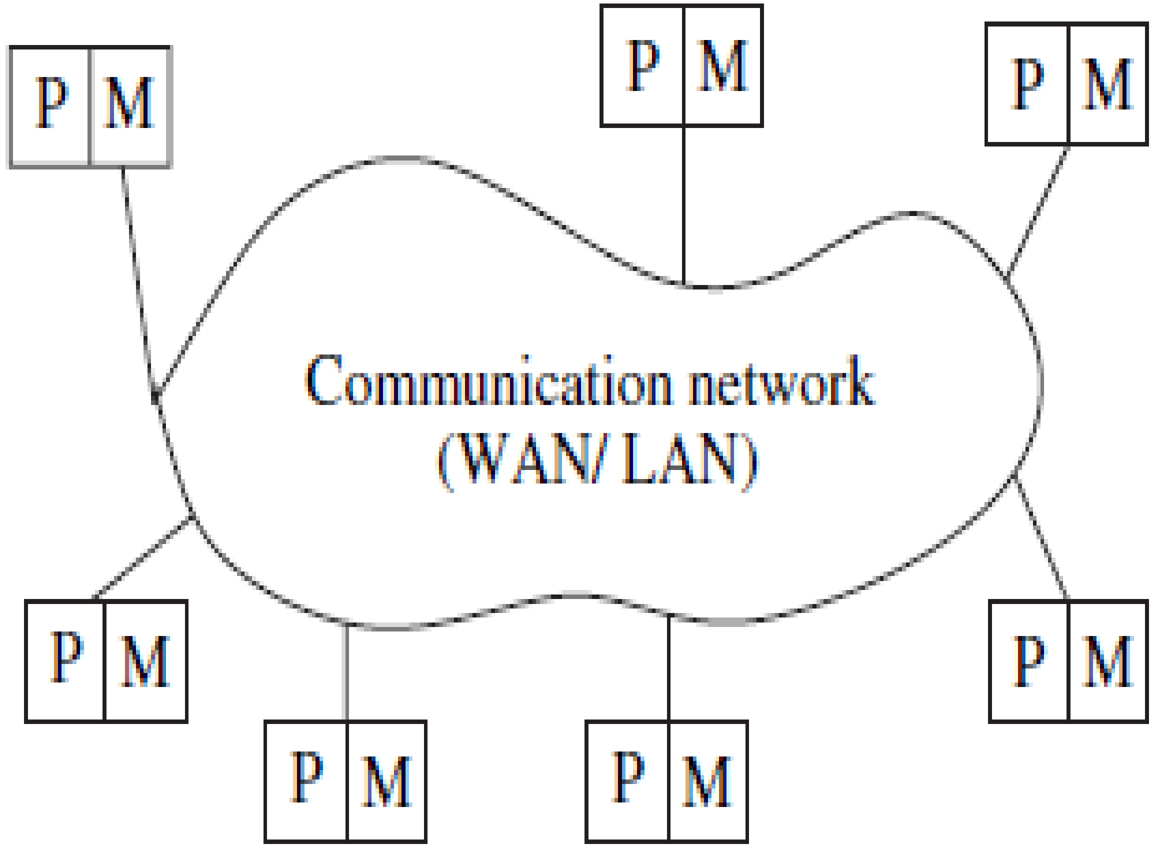
Characteristics of a distributed system

- Autonomous processors communicating over a communication network make a distributed system
  - ✓No common physical clock
  - ✓No shared memory
  - ✓Geographical separation
  - ✓Autonomy and heterogeneity

Ideal Distributed System?

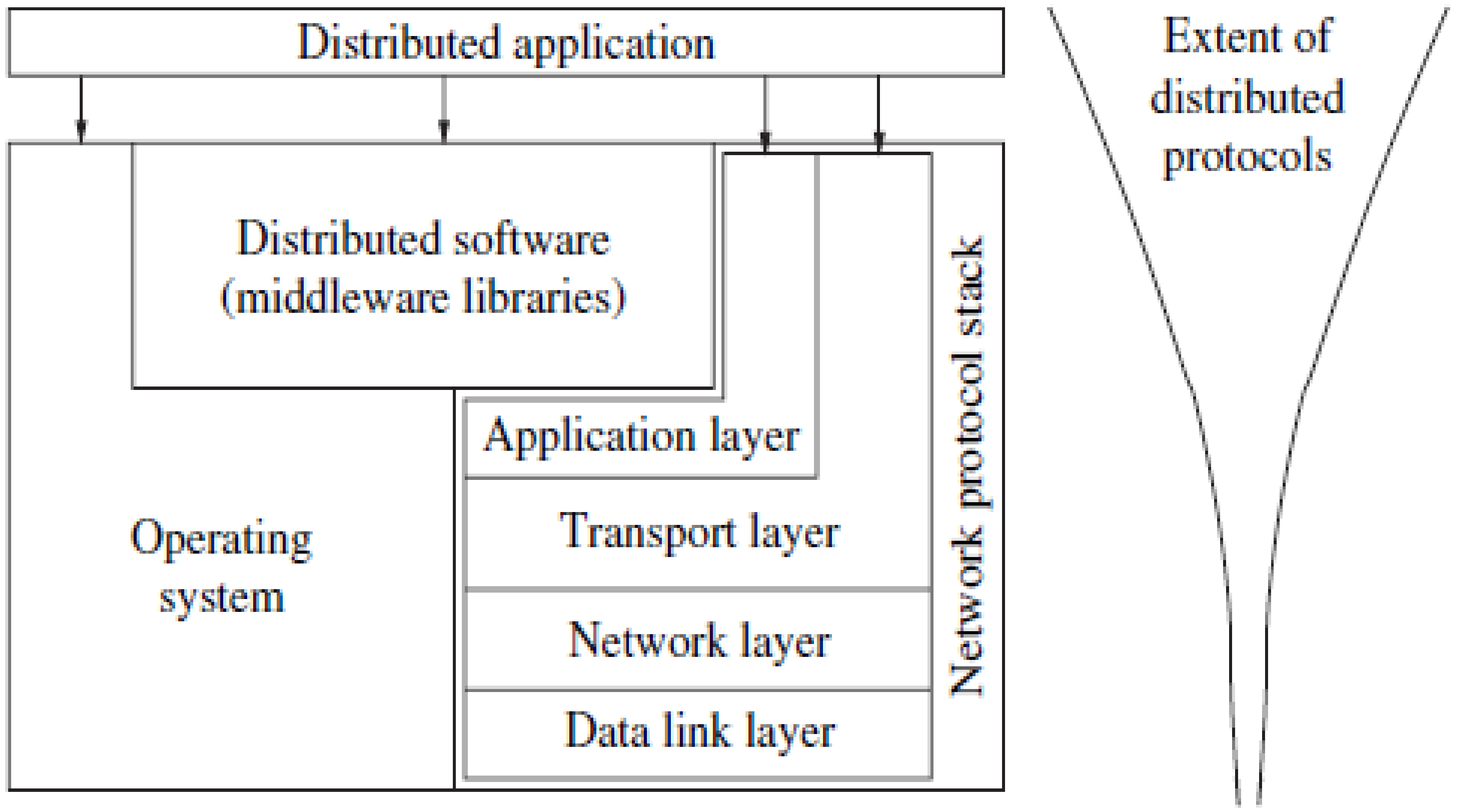


Distributed System model



P	processor(s)
M	memory bank(s)

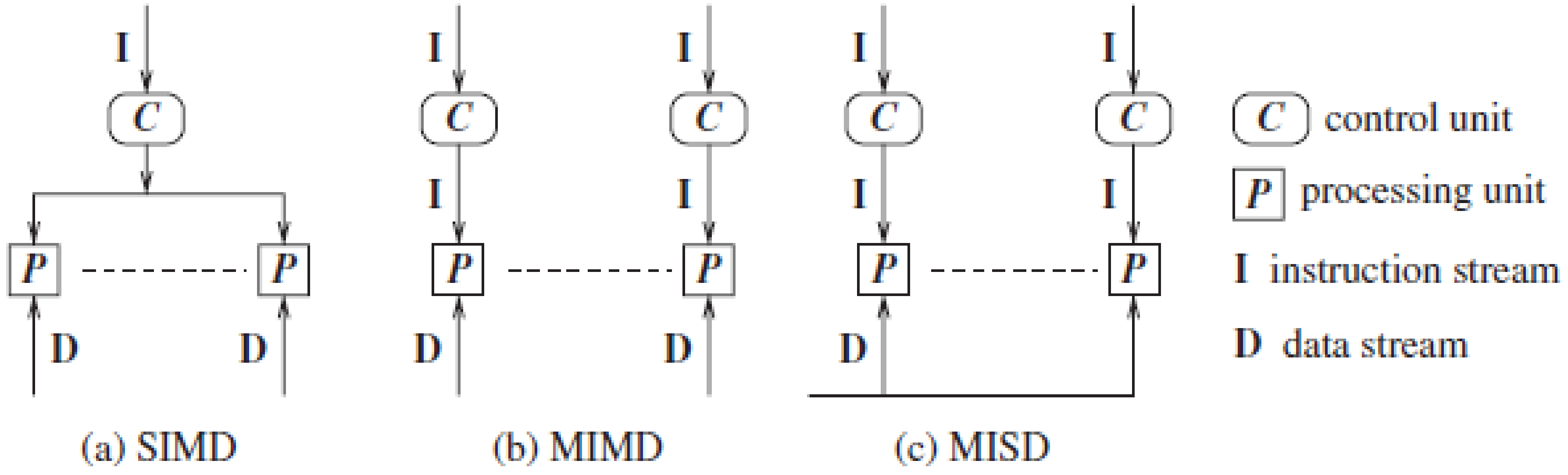
Software Components and their interaction in Distributed environment



Why do we need distributed systems?

- ✓ Inherently distributed computation
- ✓ Resource sharing
- ✓ Access to remote resources
- ✓ Increased performance/cost ratio
- ✓ Reliability
- ✓ Availability, integrity, fault-tolerance
- ✓ Scalability, Modularity and incremental expandability

Flynn's Taxonomy



1. **Single instruction stream, single data stream (SISD)**
2. **Single instruction stream, multiple data streams (SIMD)**
   - **Vector architectures**
   - **Multimedia extensions**
   - **Graphics processor units**
3. **Multiple instruction streams, single data stream (MISD)**
   - **No commercial implementation**
4. **Multiple instruction streams, multiple data streams (MIMD)**
   - **Tightly-coupled MIMD**
   - **Loosely-coupled MIMD**

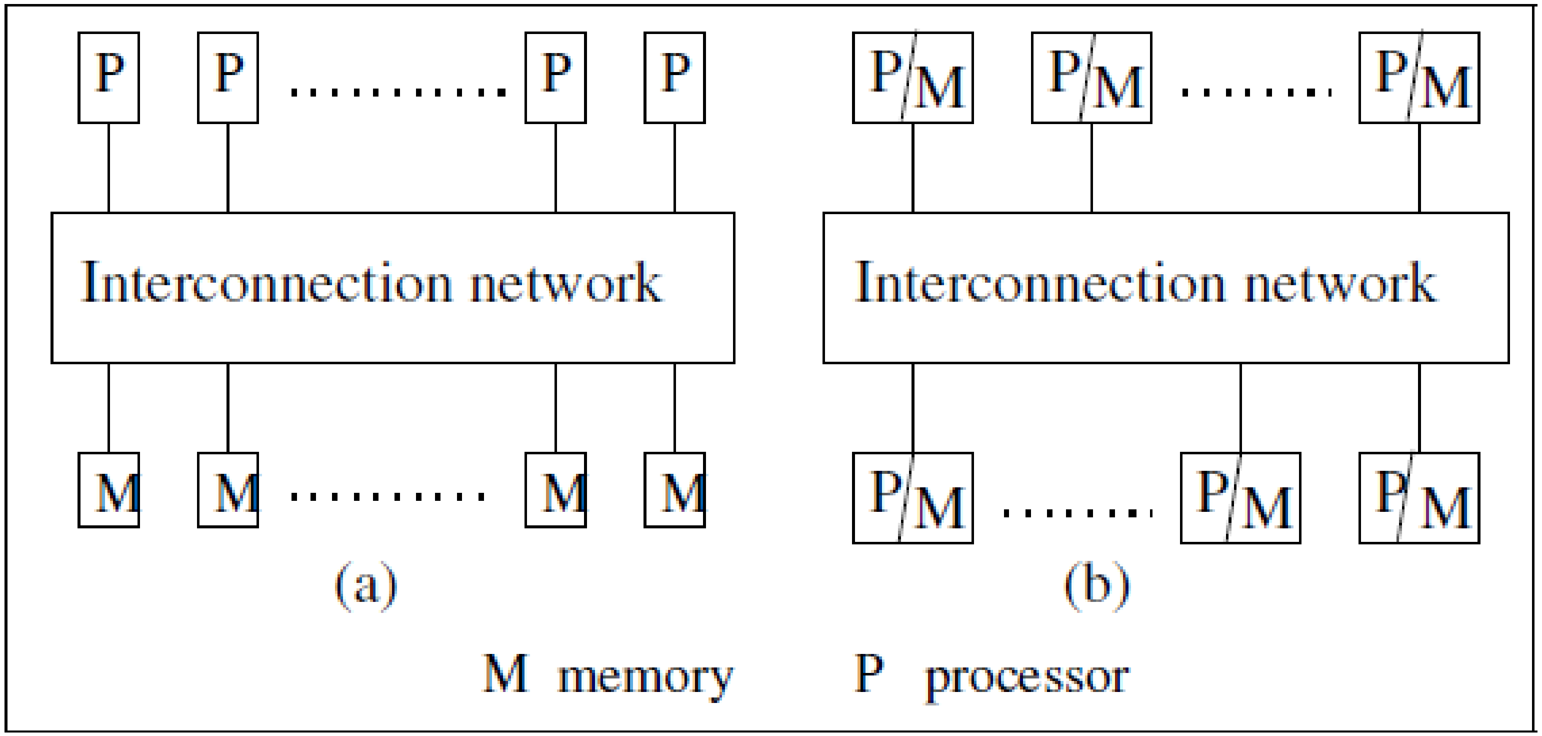
Distributed vs Parallel vs Multiprocessor systems

Parallel Systems

- Multiprocessor systems (direct access to shared memory, UMA model)
  - Interconnection network - bus, multi-stage sweitch
  - E.g., Omega, Butterfly, Clos, Shuffle-exchange networks
  - Interconnection generation function, routing function
- Multicomputer parallel systems (no direct access to shared memory, NUMA model)
  - bus, ring, mesh (w w/o wraparound), hypercube topologies
  - E.g., NYU Ultracomputer, CM* Conneciton Machine, IBM Blue gene
- Array processors (colocated, tightly coupled, common system clock)
  - Niche market, e.g., DSP applications

Contd..

Multiprocessor/Multi computer Systems



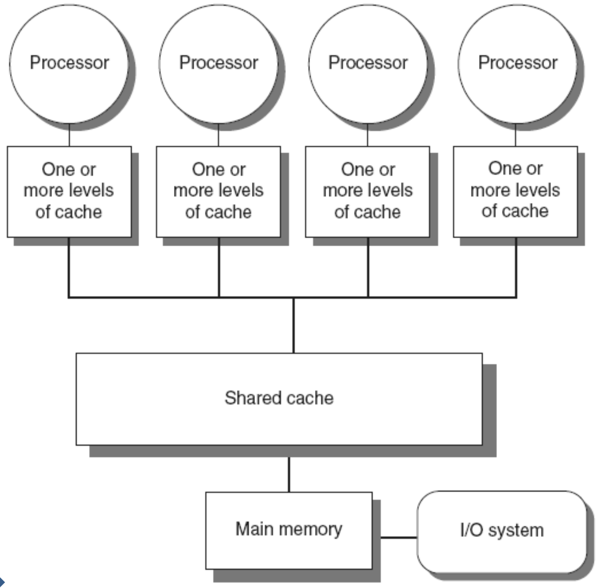
Two standard architectures for parallel systems. In both architectures, the processors may locally cache data from memory.

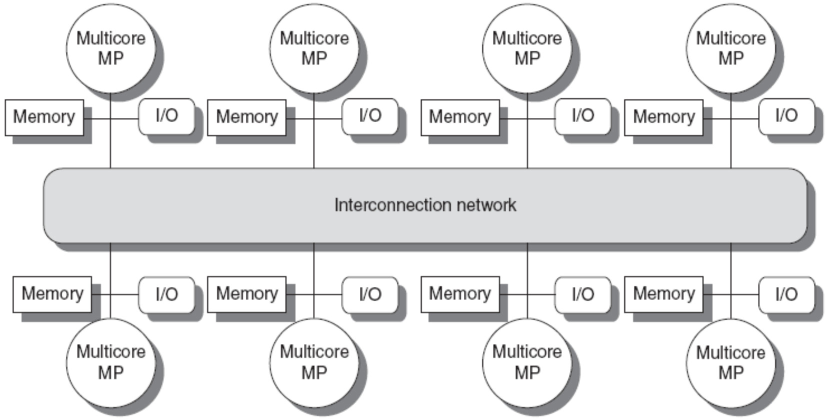
(a) Uniform memory access (UMA) multiprocessor system.

(b) Non-uniform memory access (NUMA) multiprocessor.

Types of multiprocessing

- Centralized Shared Memory (CSM) or Symmetric multiprocessors (SMP)
  - Small number of cores
  - Share single memory with uniform memory latency (UMA)
- Distributed shared memory (DSM)
  - Memory distributed among processors
  - Non-uniform memory access/latency (NUMA)
  - Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks


Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Private caches
Shared cache
Main memory
I/O system

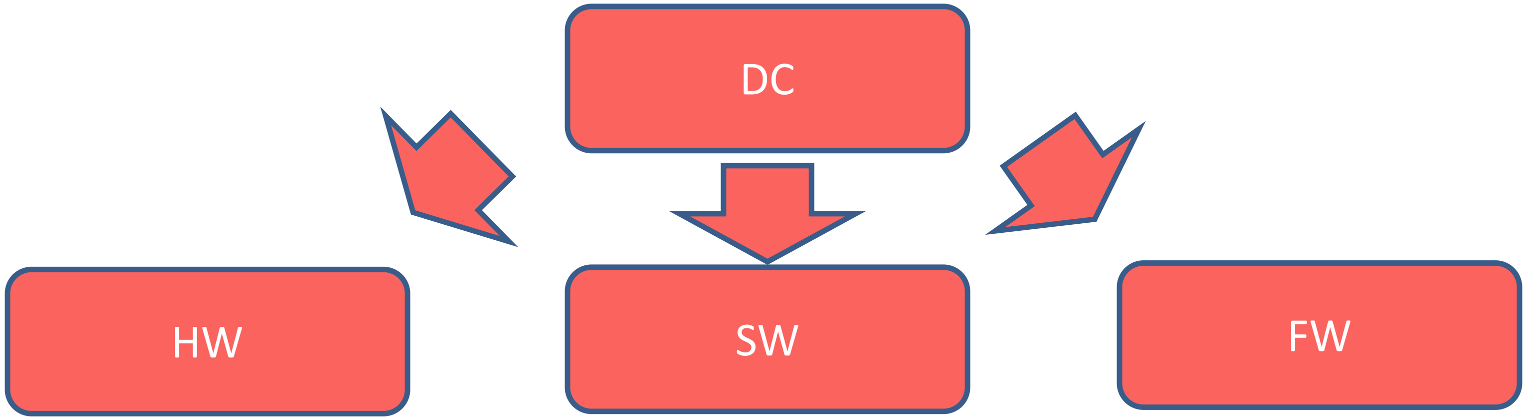


How does a distributed computing system work?

- The machines that are part of a distributed system may be computers, physical servers, virtual machines, containers, or any other node that can connect to the network, have local memory, and communicate by passing messages
- Two general ways that distributed systems function are:

1. Each machine works toward a common goal and the end-user views results as one cohesive unit
2. Each machine has its own end-user and the distributed system facilitates sharing resources or communication services

What all things constitute a distributed computing environment?



Tag cloud representation of distributed computing

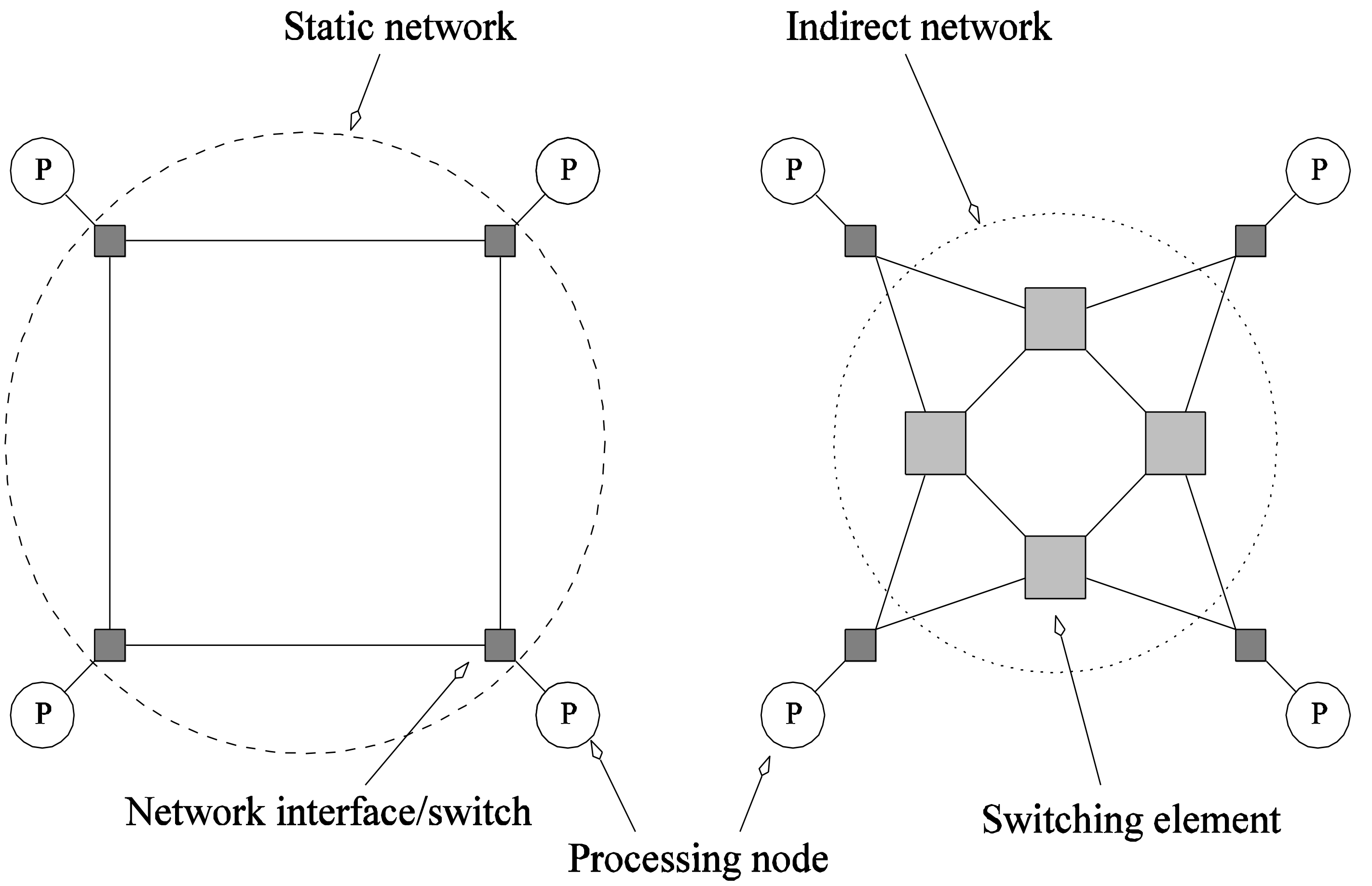
Interconnection networks in Distributed Computing

- Interconnection networks are composed of switching elements
- Topology is the pattern to connect the individual switches to other elements, like processors, memories and other switches
- A network allows exchange of data between processors in the distributed computing system

Types of Interconnection networks

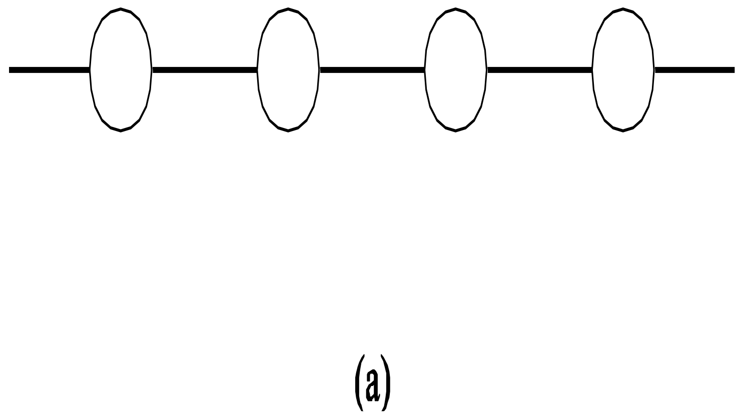
- **Direct connection networks** – Direct (static) networks have point-to-point connections between neighboring nodes. Ex:
  - **Rings**
  - **Meshes**
  - **Cubes**
- **Indirect connection networks** – Indirect (dynamic) networks have no fixed neighbors. The communication topology can be changed dynamically based on the application demands. Ex:
  - **Bus networks**
  - **Multistage networks**
  - **Cross bars**

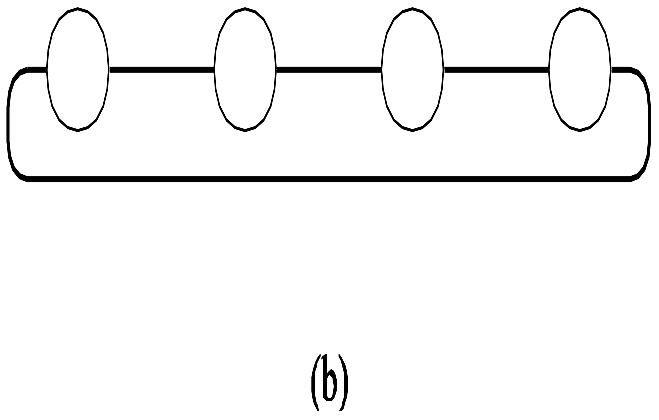
Static and Dynamic Interconnection Networks



Classification of interconnection networks: (a) static network (b) dynamic network

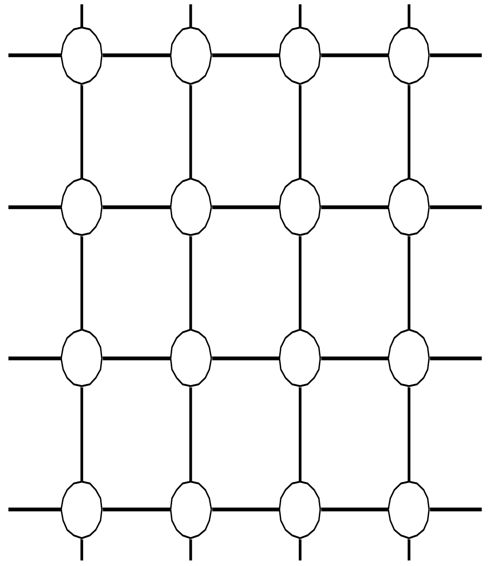
Network Topologies: Linear Arrays

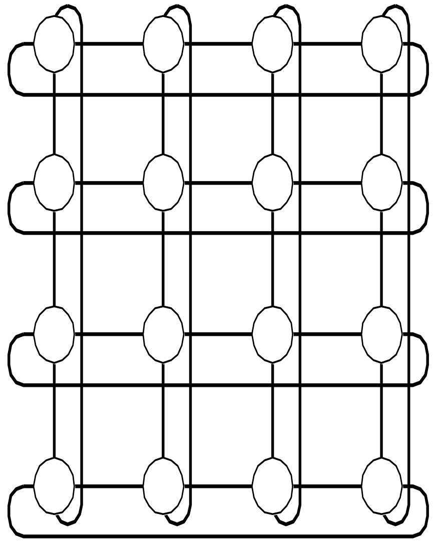


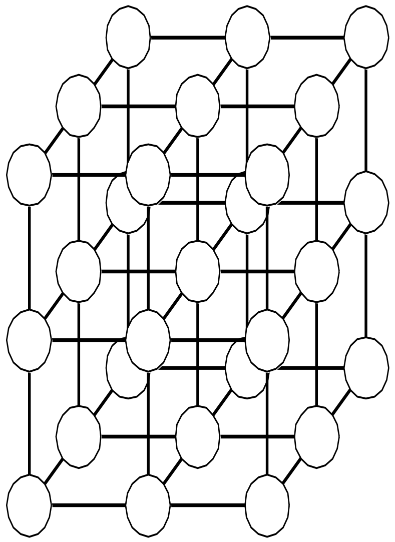


Linear arrays: (a) with no wraparound links; (b) with wraparound link.

Network Topologies: Two- and Three Dimensional Meshes


(a)

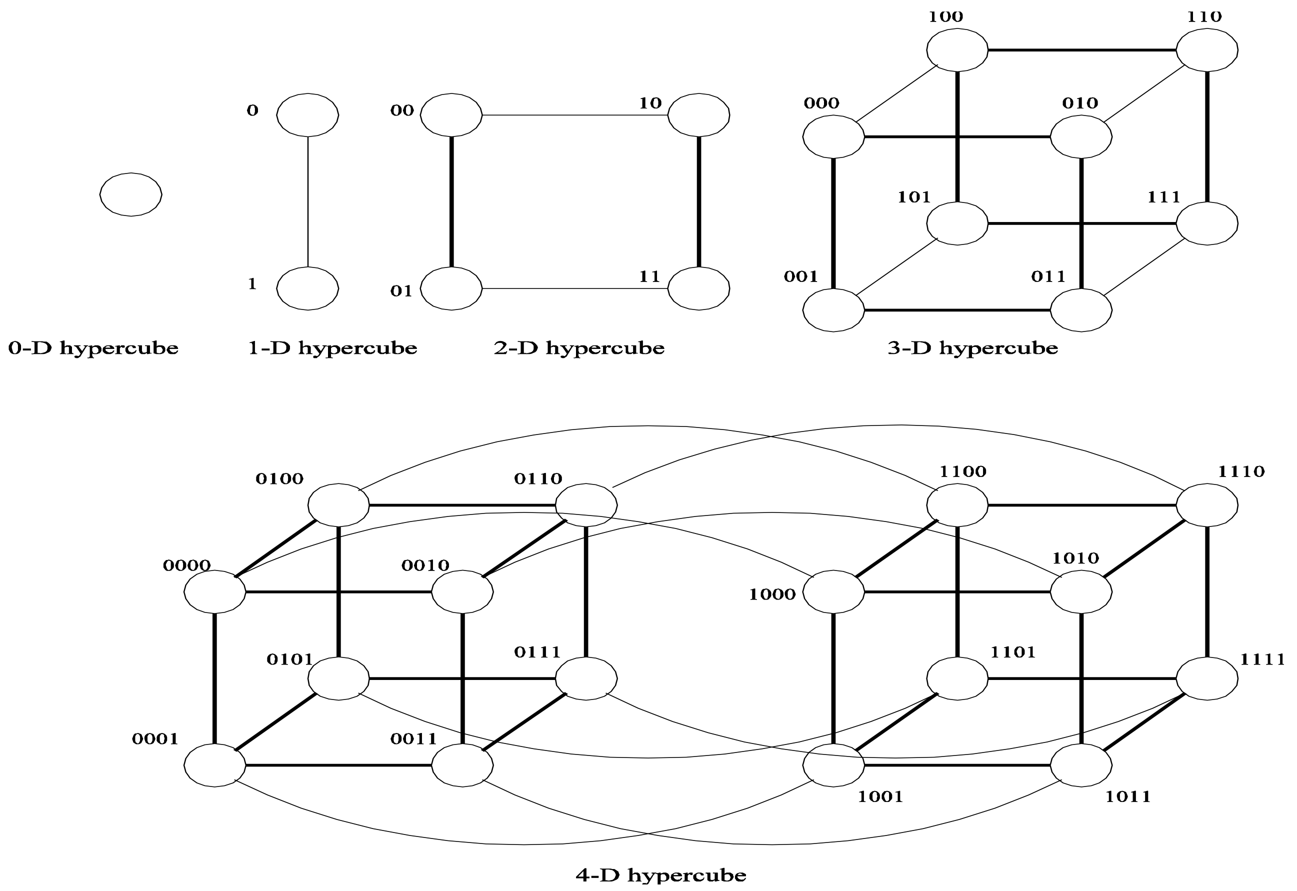

(b)


(c)

Two and three dimensional meshes: (a) 2-D mesh with no wraparound; (b) 2-D mesh with wraparound link (2-D torus); and (c) a 3-D mesh with no wraparound.

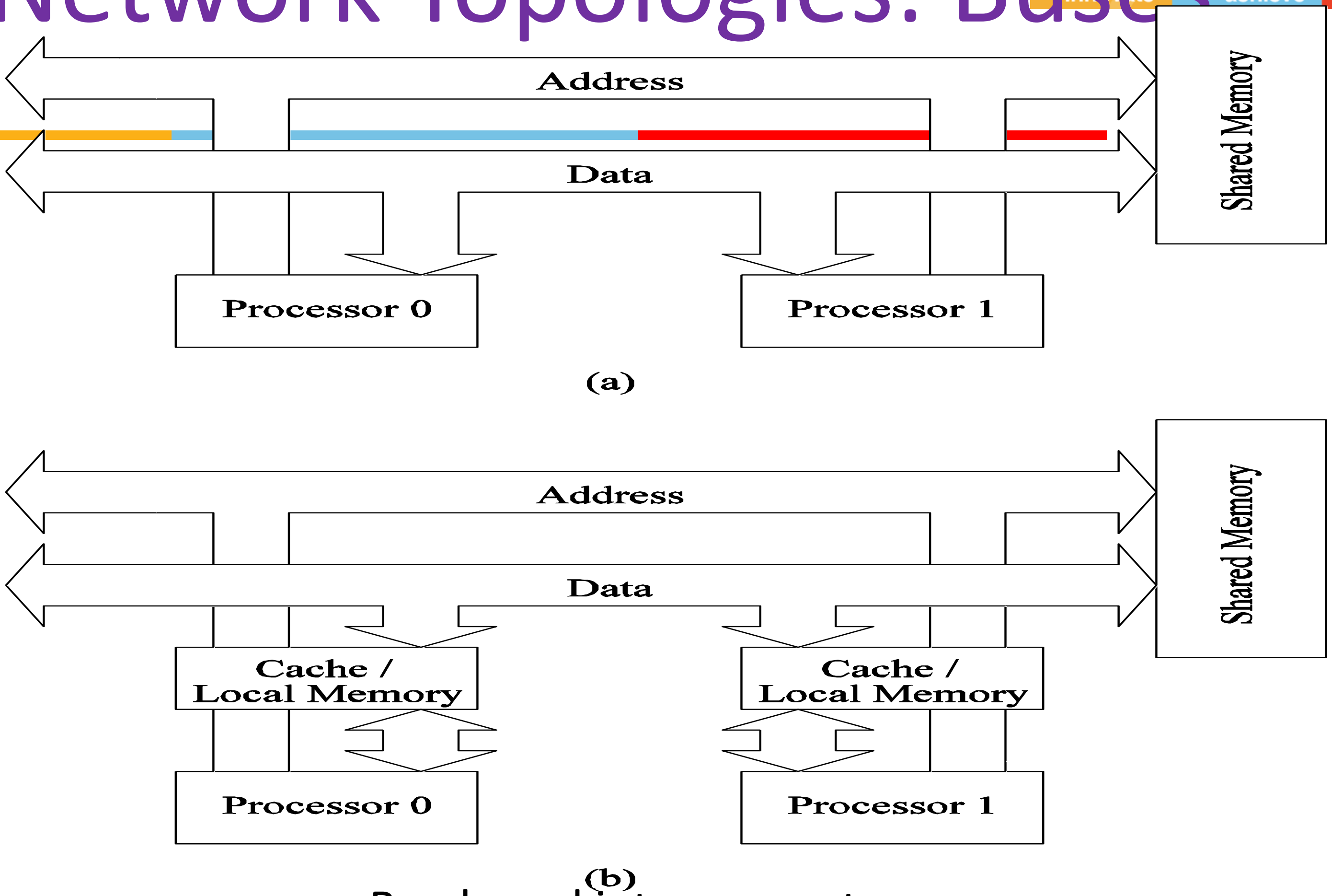
(3-D weather modelling, structural modelling physical simulations)

Network Topologies: Hypercubes and their Construction



Construction of hypercubes from hypercubes of lower dimension.

Network Topologies: Buses



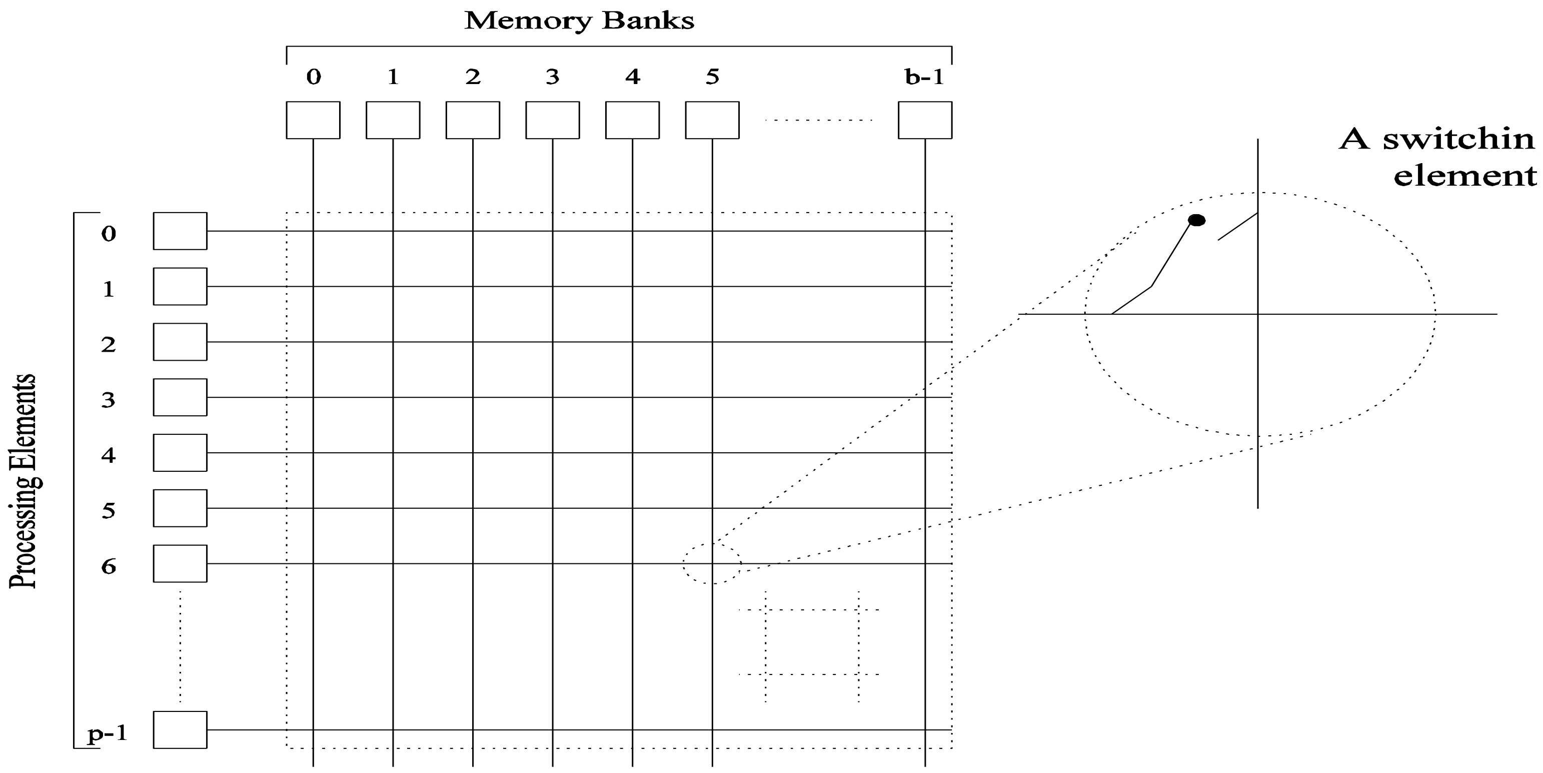
Bus-based interconnects

(a) with no local caches; (b) with local memory/caches.

Since much of the data accessed by processors is local to the processor, a local memory can improve the performance of bus-based machines.

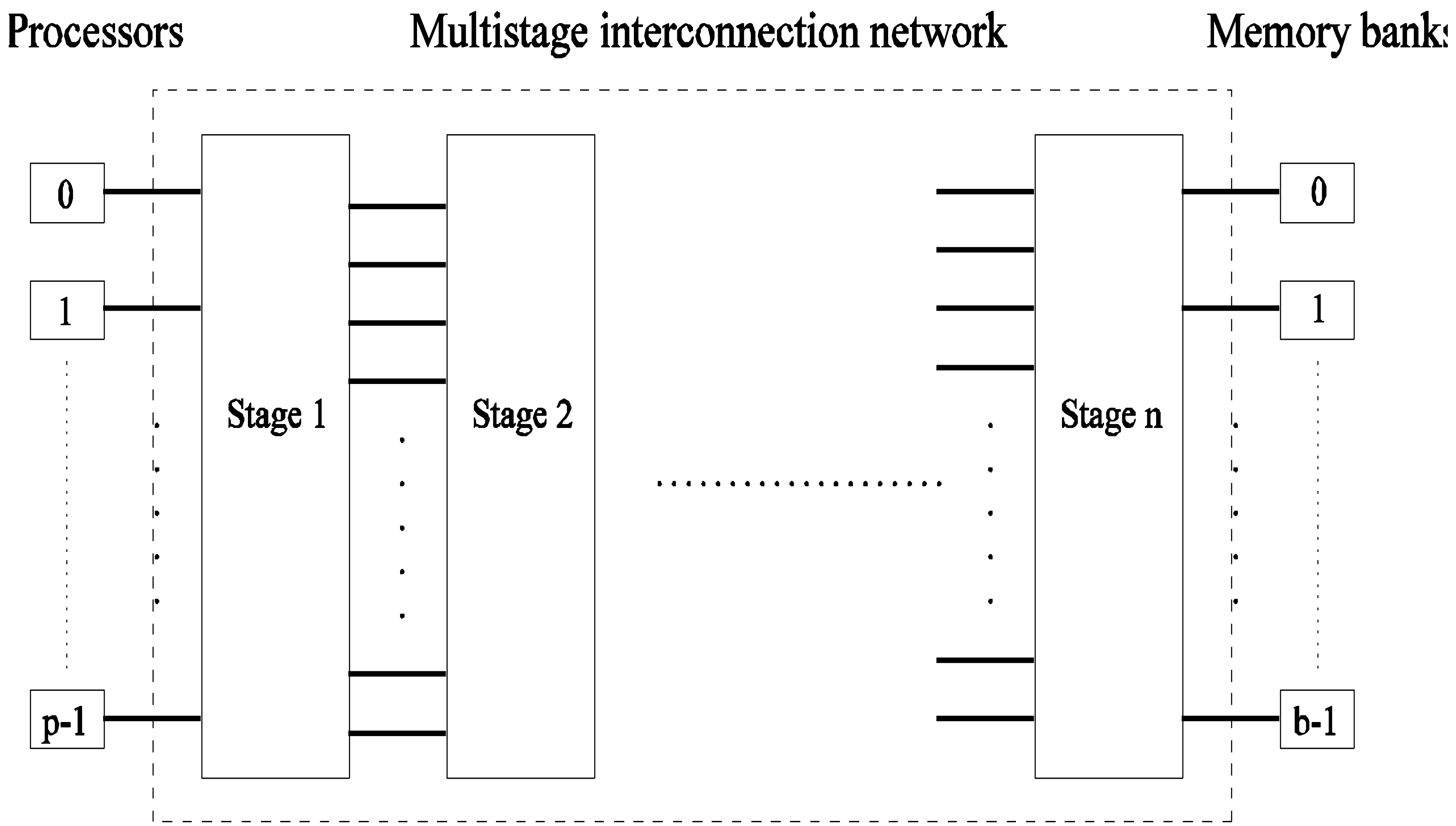
Network Topologies: Crossbars

A crossbar network uses an *p×m* grid of switches to connect *p* inputs to m outputs in a non-blocking manner.



A completely non-blocking crossbar network connecting *p* processors to b memory banks.

Network Topologies: Multistage Networks



The schematic of a typical multistage interconnection network.

Network Topologies: Multistage Omega Network

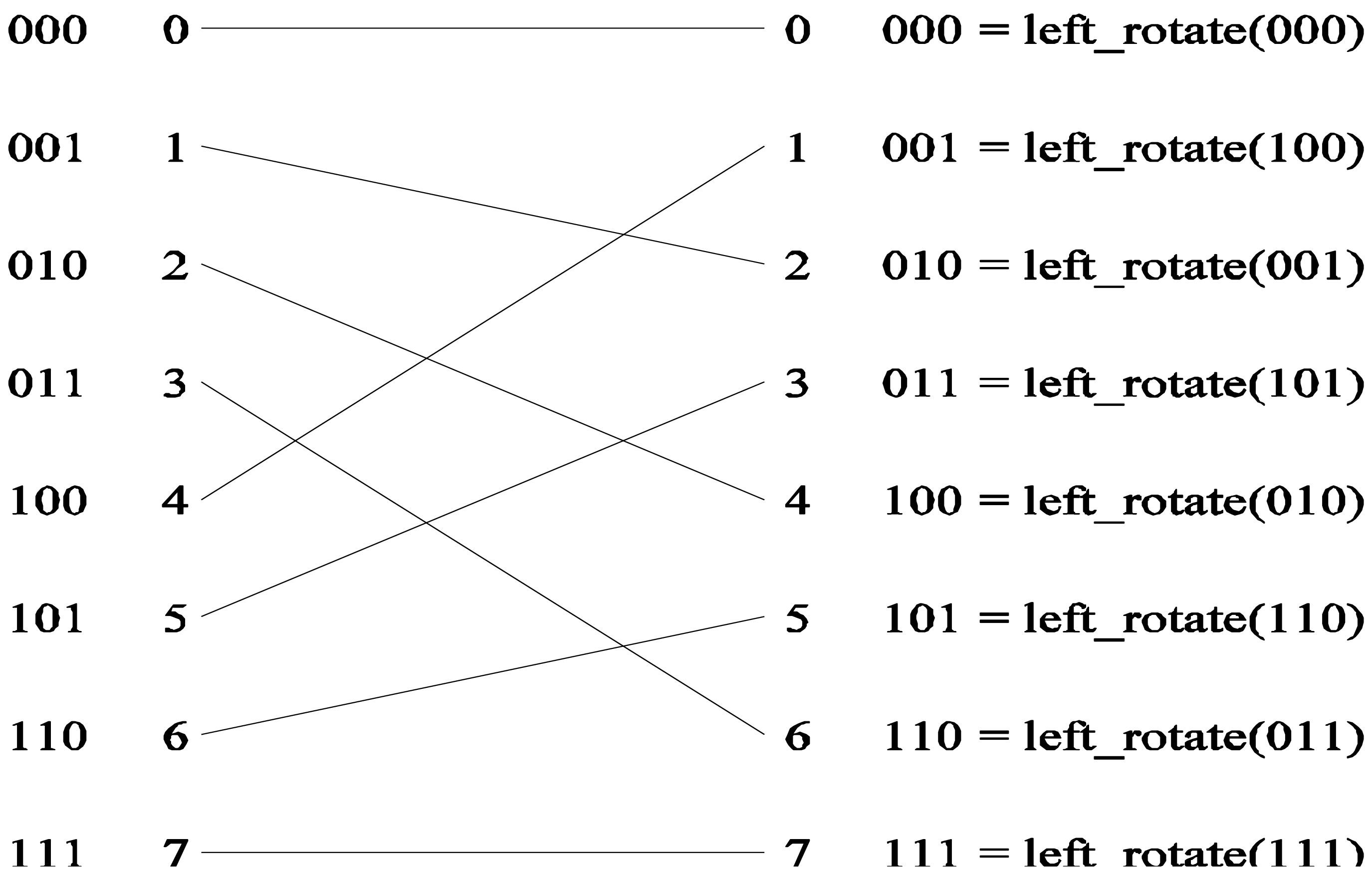
- One of the most commonly used multistage interconnects is the Omega network.
- This network consists of *log p* stages, where *p* is the number of inputs/outputs (processing nodes as well as memory banks)
- At each stage, input *i* is connected to output *j* if:

$$j = \begin{cases} 2i, & 0 \leq i \leq p/2 - 1 \\ 2i + 1 - p, & p/2 \leq i \leq p - 1 \end{cases}$$

- This is actually a left-rotation operation on the binary representation of i to obtain j

Network Topologies: Multistage Omega Network

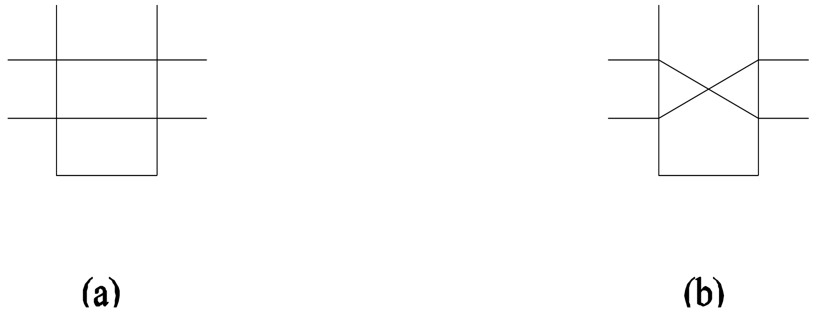
Each stage of the Omega network implements a perfect shuffle as follows:


000 0 — 0 000 = left_rotate(000)
001 1 — 1 001 = left_rotate(100)
010 2 — 2 010 = left_rotate(001)
011 3 — 3 011 = left_rotate(101)
100 4 — 4 100 = left_rotate(010)
101 5 — 5 101 = left_rotate(110)
110 6 — 6 110 = left_rotate(011)
111 7 — 7 111 = left_rotate(111)

A perfect shuffle interconnection for eight inputs and outputs.

Network Topologies: Multistage Omega Network

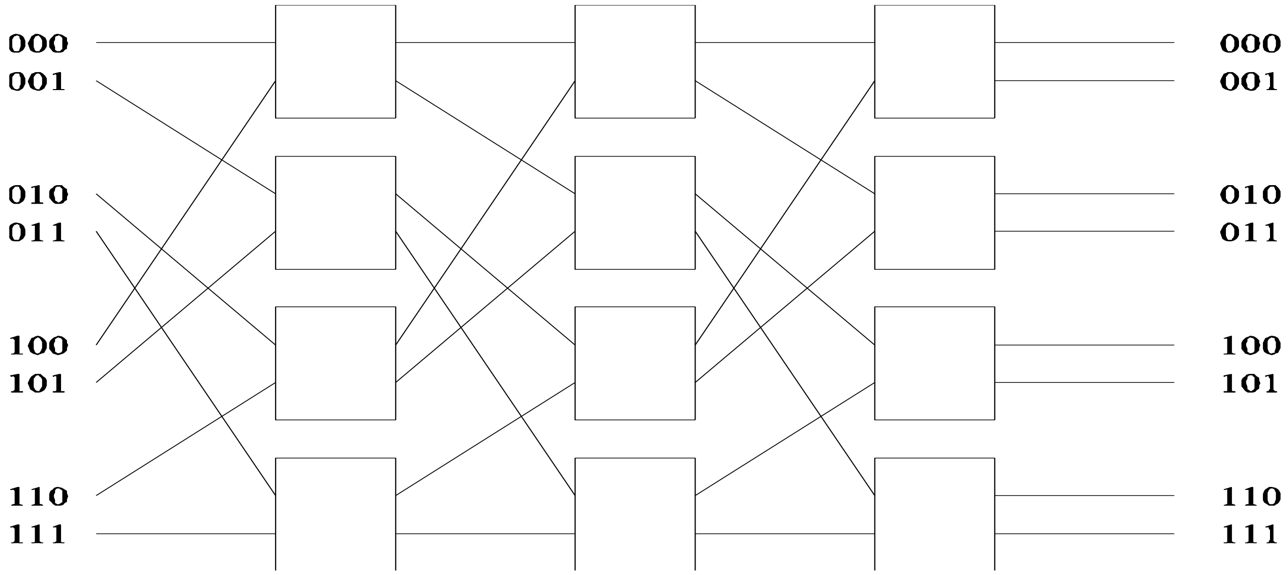
- The perfect shuffle patterns are connected using 2×2 switches.
- The switches operate in two modes – crossover or pass through.



Two switching configurations of the 2 × 2 switch:
(a) Pass-through; (b) Cross-over.

Network Topologies: Multistage Omega Network

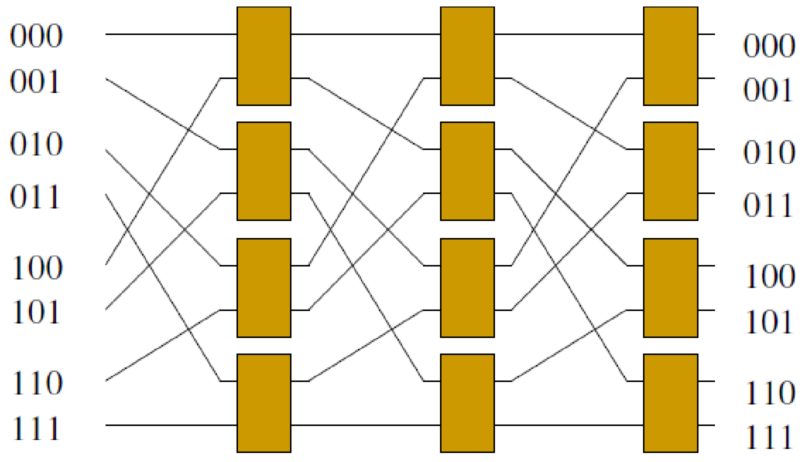
A complete Omega network with the perfect shuffle interconnects and switches can now be illustrated:



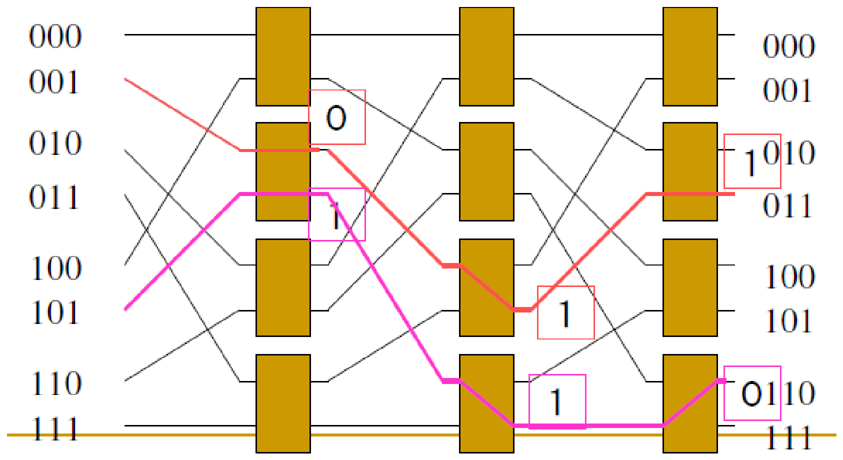
A complete omega network connecting eight inputs and eight outputs.

An omega network has *p/2 × log p* switching nodes, and the cost of such a network grows as *(p log p).*

Omega network - example



(a)



(b)

1. For the omega network given above in (a), find the number of switching elements

Answer = ½ $n\log_2 n$ = 12

2. How to apply the perfect shuffle technique for destination routing in (b)?

Answer = Check the destination tag from MSB If 0 use upper link, else use lower link.

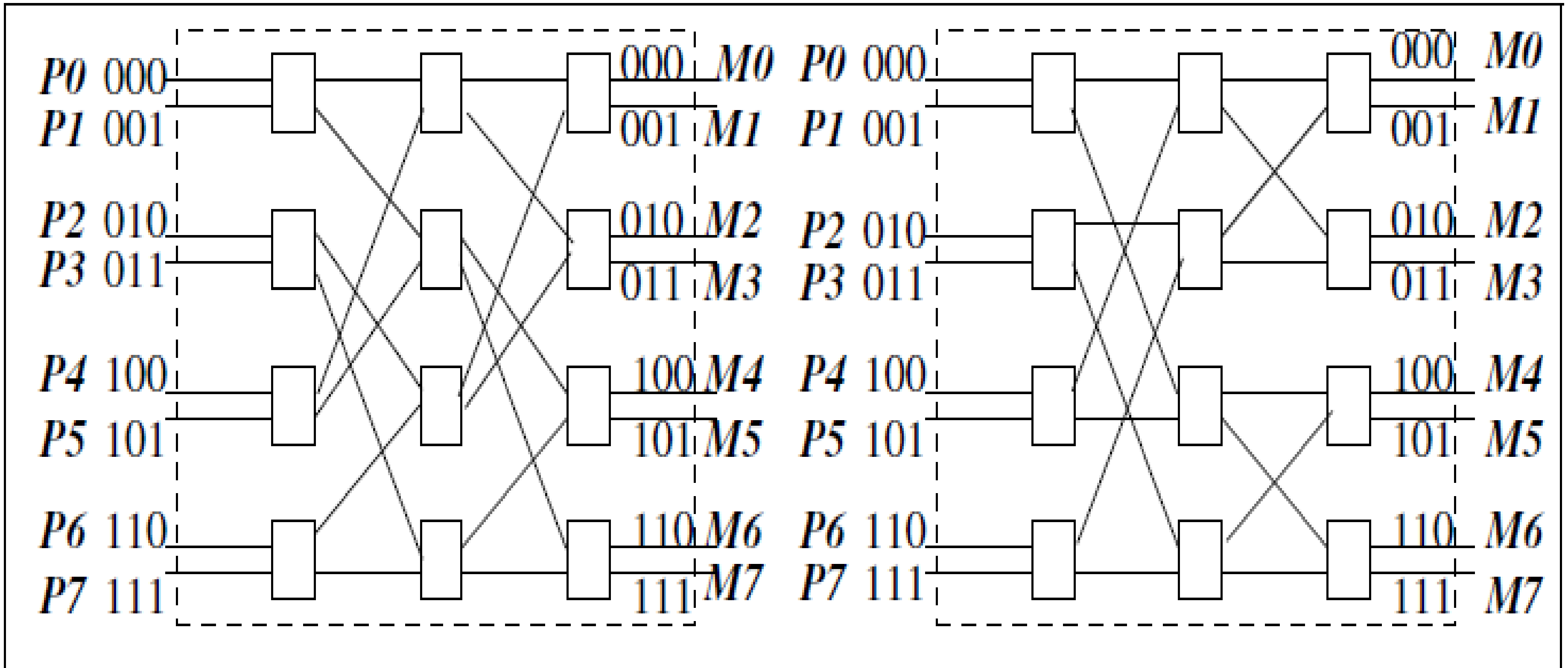
Ex: 1 => 3

5 => 6

Network Topologies: Butterfly Networks

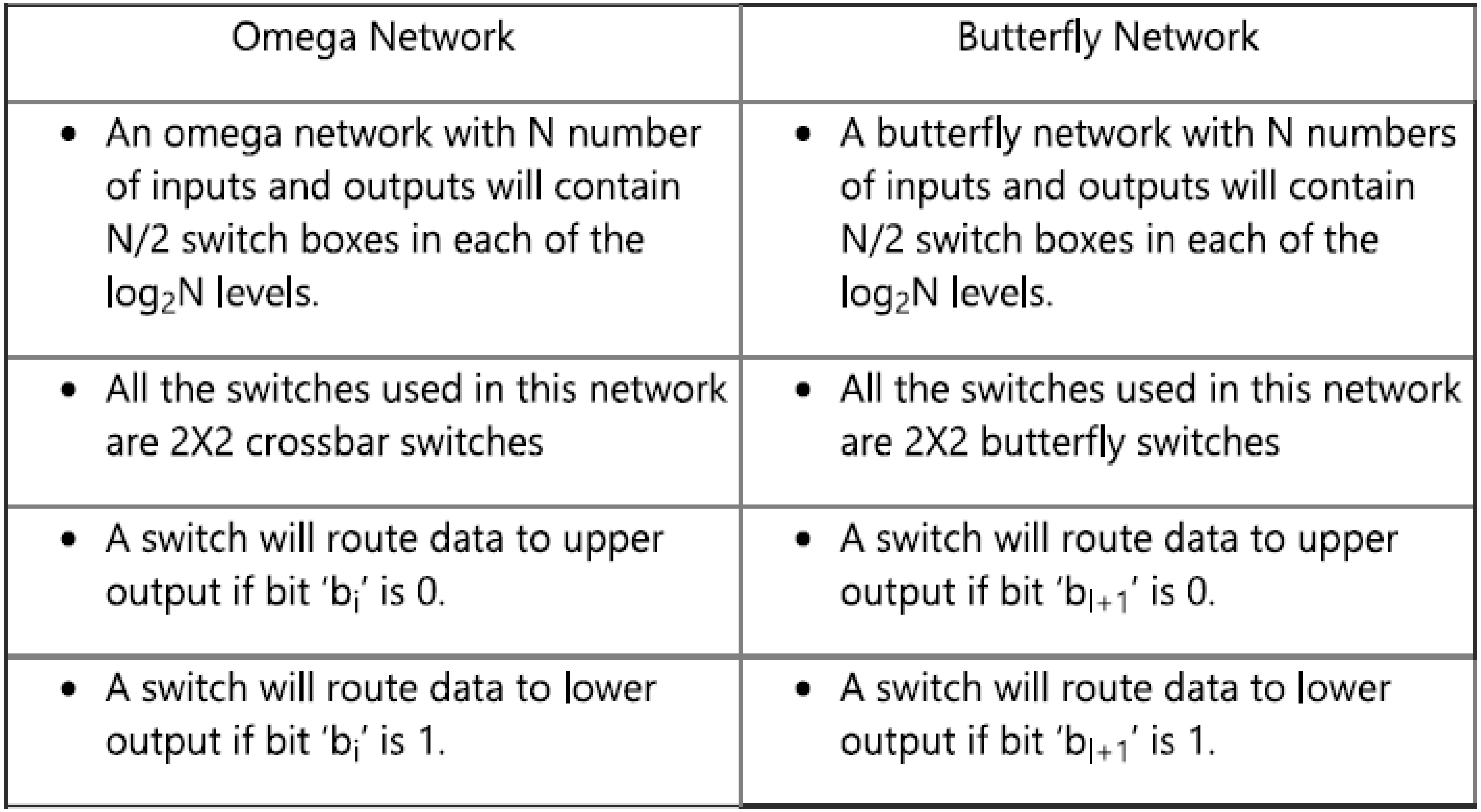
- A butterfly network is a technique to link multiple computers into a high-speed network
  - This form of multistage interconnection network topology can be used to connect different nodes in a multiprocessor system
  - Butterfly networks have lower diameter than other topologies like a linear array, ring and 2-D mesh. This implies that in butterfly network, a message sent from one processor would reach its destination in a lower number of network hops
  - Butterfly networks have higher bisection bandwidth than other topologies. This implies that in butterfly network, a higher number of links need to be broken in order to prevent global communication

Omega, Butterfly Interconnects



(a) 3–stage Omega network (*n*=8, *M*=4)   (b) 3–stage Butterfly network (*n*=8, *M*=4)

Comparing Omega networks and butterfly networks

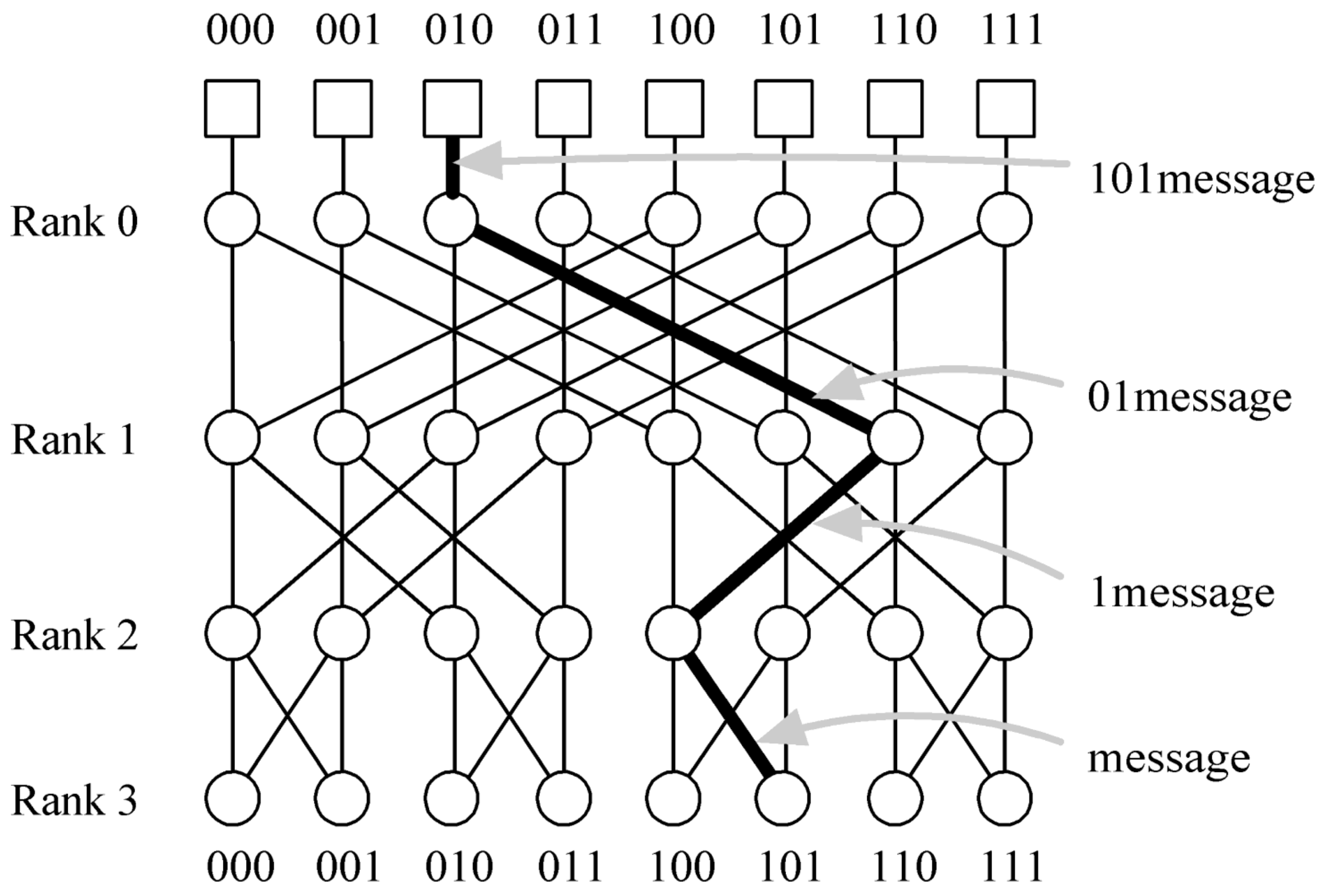


Omega Network	Butterfly Network
• An omega network with N number of inputs and outputs will contain N/2 switch boxes in each of the $\log_2 N$ levels.	• A butterfly network with N numbers of inputs and outputs will contain N/2 switch boxes in each of the $\log_2 N$ levels.
• All the switches used in this network are 2X2 crossbar switches	• All the switches used in this network are 2X2 butterfly switches
• A switch will route data to upper output if bit 'b_i' is 0.	• A switch will route data to upper output if bit 'b_{l+1}' is 0.
• A switch will route data to lower output if bit 'b_i' is 1.	• A switch will route data to lower output if bit 'b_{l+1}' is 1.

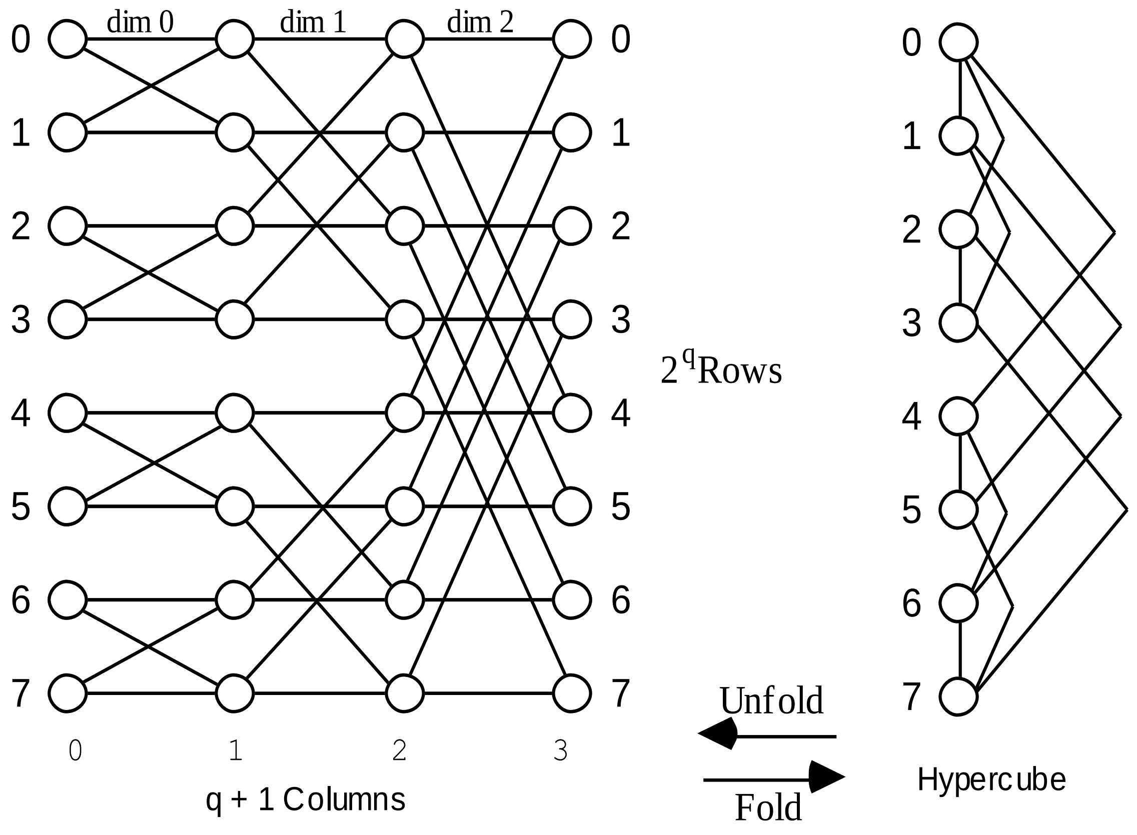
Butterfly interconnection function

- Unlike the Omega network, the generation of the interconnection pattern between a pair of adjacent stages depends not only on n but also on the stage number s.
- Let there be M = n/2 switches per stage, and let a switch be denoted by the tuple <x,s> where $x \in [0, M-1]$ and stage $s \in [0, \log_2 n-1]$
- The two outgoing edges from any switch <x,s> are as follows
- There is an edge from switch <x,s> to switch <y,s+1> if (i) x = y or (ii) x XOR y has exactly one 1 bit, which is in the s +1 th MSB
- For stage s, apply the rule above for $M/2^s$ switches.

Butterfly Network

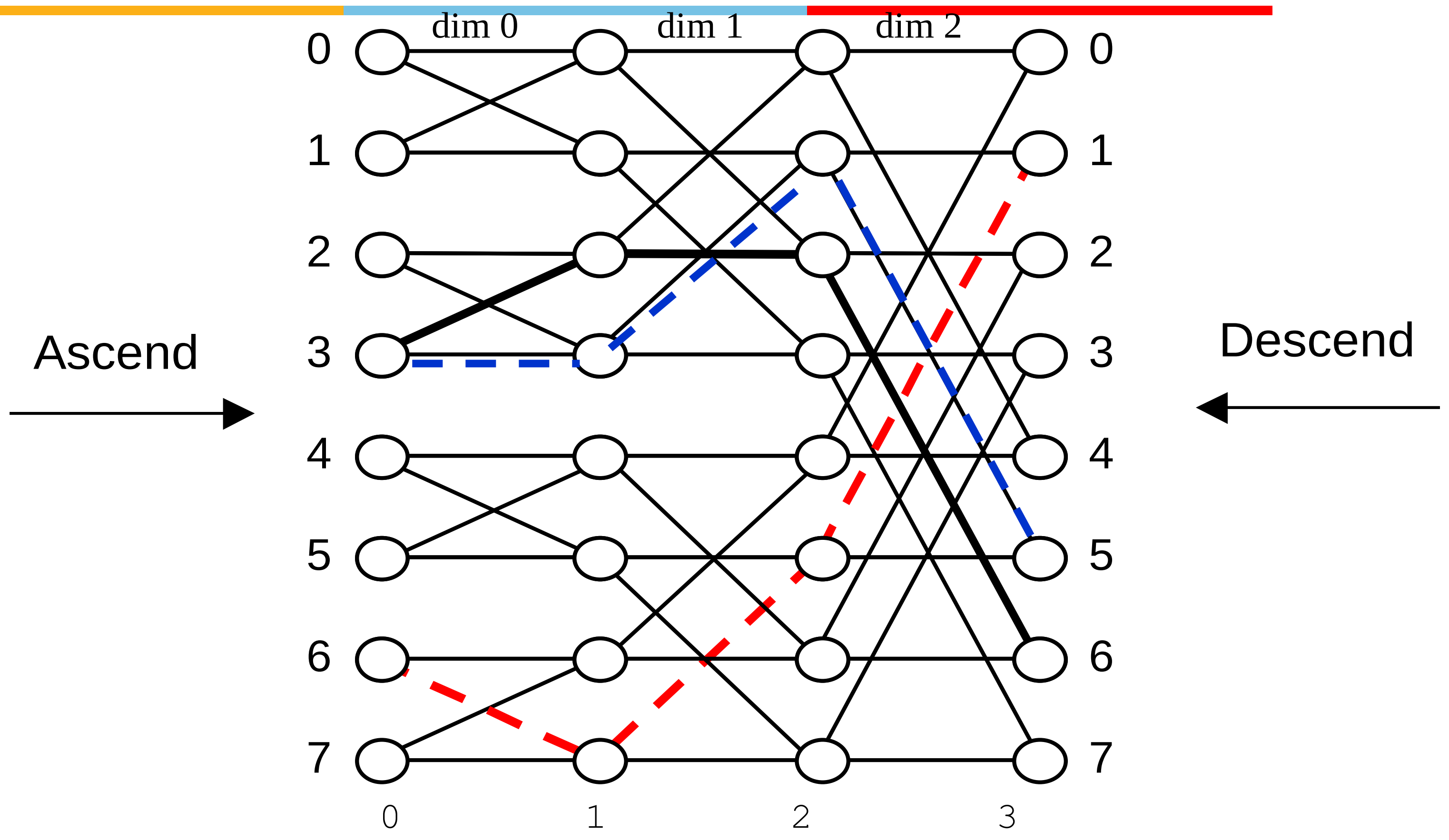


Unfolded 3-cube or 32-node butterfly network



Dimension-order routing between nodes *i* and *j* in *q*-cube can be viewed as routing from node *i* in column 0 (*q*) to node *j* in column *q* (0) of the butterfly

Butterfly Network - example



Number of cross links taken = length of path in hypercube

From node 3 to 6: routing tag = 011 ⊕ 110 = 101 “cross-straight-cross”
From node 3 to 5: routing tag = 011 ⊕ 101 = 110 “cross-cross-straight”
From node 6 to 1: routing tag = 110 ⊕ 001 = 111 “cross-cross-cross”

Coupling

- ✓ The degree of coupling among a set of modules, whether hardware or software, is measured in terms of the interdependency and binding and/or homogeneity among the modules
- ✓ When the degree of coupling is high (low), the modules are said to be tightly (loosely) coupled
- ✓ SIMD and MISD architectures generally tend to be tightly coupled because of the common clocking of the shared instruction stream or the shared data stream

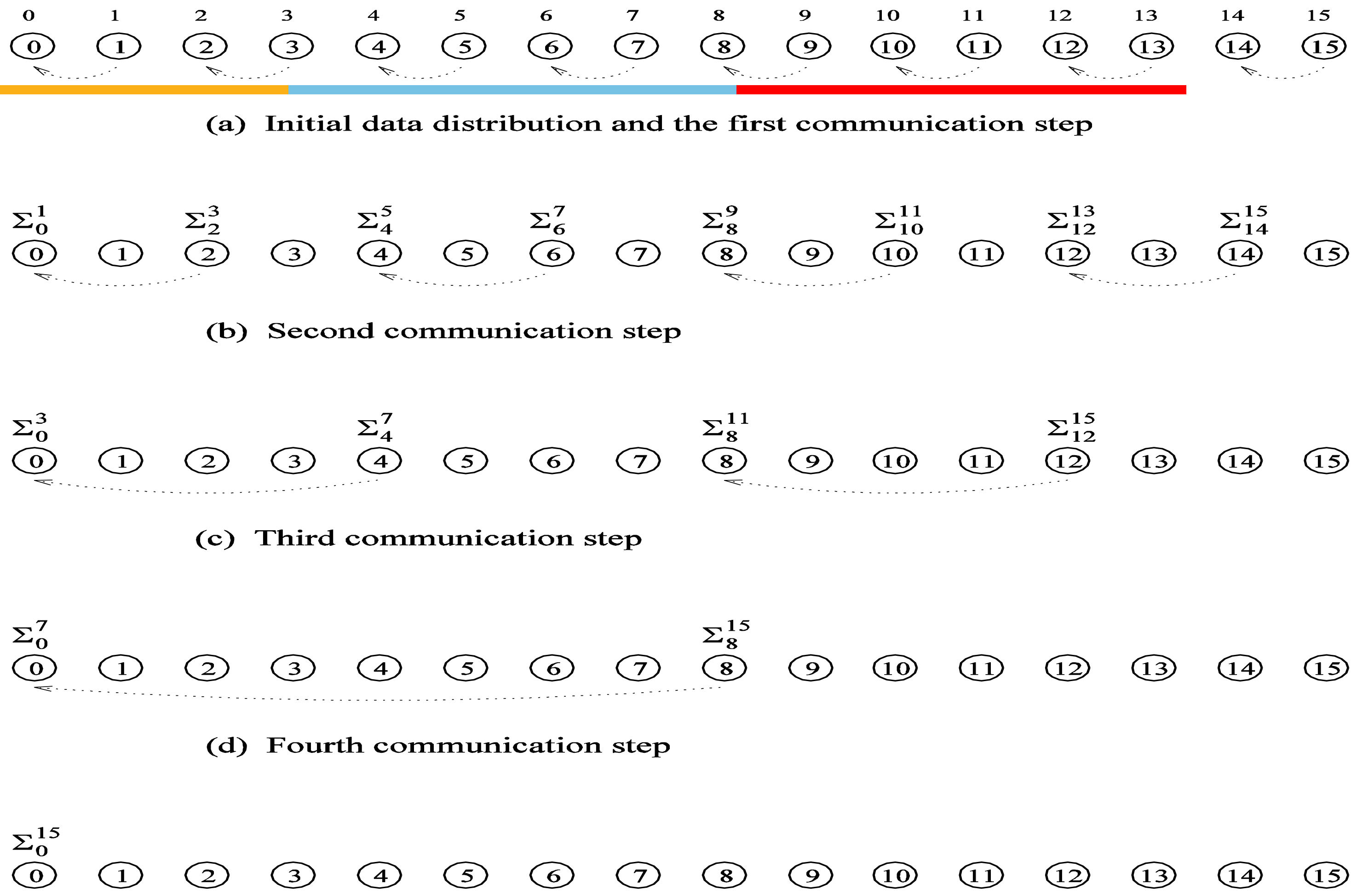
Coupling in MIMD Architectures

1. Tightly coupled multiprocessors (with UMA shared memory)
2. Tightly coupled multiprocessors (with NUMA shared memory or that communicate by message passing)
3. Loosely coupled multi computers (without shared memory) physically collocated
4. Loosely coupled multi computers (without shared memory and without common clock) that are physically remote

Parallelism

- This is a measure of the relative speedup of a specific program, on a given machine
- The speedup depends on the number of processors and the mapping of the code to the processors
- Speedup (***S***) is the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with ***p*** identical processing elements
- It is expressed as the ratio of the time T(1) with a single processor, to the time T(n) with n processors.
- Parallelism within a parallel/distributed program is an aggregate measure of the percentage of time that all the processors are executing CPU instructions productively, as opposed to waiting for communication (either via shared memory or message-passing) operations to complete
- $S = T_{serial}/T_{parallel}$

Parallelism example



(a) Initial data distribution and the first communication step
(b) Second communication step
(c) Third communication step
(d) Fourth communication step

(e) Accumulation of the sum at processing element 0 after the final communicatio

Computing the global sum of 16 partial sums using 16 processing elements . Σ^j_i denotes the sum of numbers with consecutive labels from *i* to *j*.

Parallelism example (continued)

- If an addition takes constant time, say, t_c and communication of a single word can be performed in $t_s + t_w$, we have the parallel time

  $T_P = \Theta(\log n)$
- We know that $T_S = \Theta(n)$ (as the problem can be solved in $\Theta(n)$ on a single processing element)
- Now, Speedup S is given by $S = \Theta\ (n / \log n)$

Concurrency: Definition

- ✓ A broader term that means roughly the same as parallelism of a program, but is used in the context of distributed programs
- ✓ The parallelism/concurrency in a parallel/distributed program can be measured by the ratio of the number of local (non-communication and non-shared memory access) operations to the total number of operations, including the communication or shared memory access operations

Concurrency

❖The maximum number of tasks that can be executed simultaneously at any time in a parallel algorithm is called its *degree of concurrency*

❖If ***C***(***W***) is the degree of concurrency of a parallel algorithm, then for a problem of size ***W***, no more than ***C***(***W***) processing elements can be employed effectively

Degree of Concurrency: Numeric Example

Consider solving a system of equations in certain number of variables by using Gaussian elimination ($W = \Theta(n^3)$)

- The n variables must be eliminated one after the other, and eliminating each variable requires $\Theta(n^2)$ computations
- At most $\Theta(n^2)$ processing elements can be kept busy at any time
- Since $W = \Theta(n^3)$ for this problem, the degree of concurrency $C(W)$ is $\Theta(W^{2/3})$
- Given p processing elements, the problem size should be at least $\Omega(p^{3/2})$ to use them all

Granularity in Distributed Computing

- **The ratio of the amount of computation to the amount of communication within the parallel/distributed program is termed as granularity**
- If the degree of parallelism is coarse-grained (fine-grained), there are relatively many more (fewer) productive CPU instruction executions, compared to the number of times the processors communicate either via shared memory or message passing and wait to get synchronized with the other processors
- Programs with fine-grained parallelism are best suited for tightly coupled systems
  - These typically include SIMD and MISD architectures, tightly coupled MIMD multiprocessors (that have shared memory), and loosely coupled multi computers (without shared memory) that are physically co located

Effect of Granularity on Performance

- Often, using fewer processors improve performance of distributed systems.

- Using fewer than the maximum possible number of processing elements to execute a parallel/distributed algorithm is called *scaling down* a parallel /distributed system

- A naive way of scaling down is to think of each processor in the original case as a virtual processor and to assign virtual processors equally to scaled down processors

Building Granularity: Example

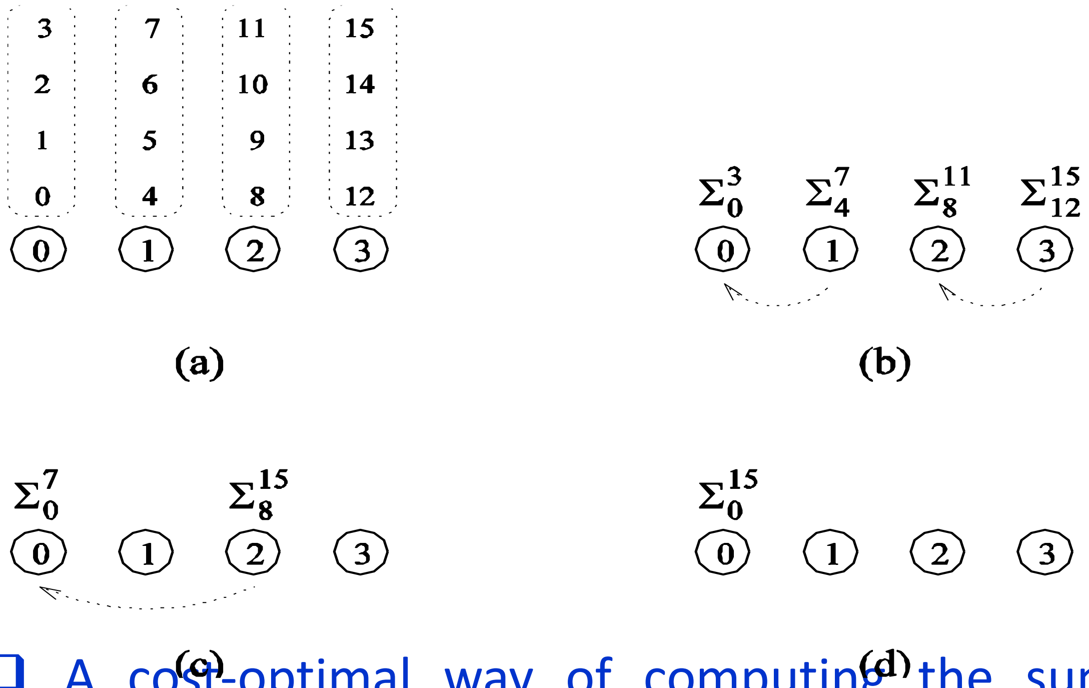
- Consider the problem of adding ***n*** numbers on ***p*** processing elements such that ***p*** < ***n*** and both ***n*** and ***p*** are powers of 2.
- Use the parallel algorithm for ***n*** processors, except, in this case, we think of them as virtual processors.
- Each of the ***p*** processors is now assigned ***n* / *p*** virtual processors.
- The first **log *p*** of the **log *n*** steps of the original algorithm are simulated in **(*n* / *p*) log *p*** steps on ***p*** processing elements.
- Subsequent **log *n* - log *p*** steps do not require any communication.

Building Granularity: Example (continued)

- Since the number of processing elements decreases by a factor of ***n* / *p***, the computation at each processing element increases by a factor of ***n* / *p***.
- The overall parallel execution time of this parallel system is $\Theta((n/p)\log p)$.

Building Granularity: Example (continued)

- ❑ Can we build granularity in the example in a cost-optimal fashion?
- ➢ Each processing element locally adds its ***n* / *p*** numbers in time Θ (***n***/ ***p***).
- ➢ The ***p*** partial sums on ***p*** processing elements can be added in time Θ(***n*** /***p***).


3 7 11 15
2 6 10 14
1 5 9 13
0 4 8 12
0 1 2 3
(a)
Σ_0^3 Σ_4^7 Σ_8^{11} Σ_{12}^{15}
(b)
Σ_0^7 Σ_8^{15}
(c)
Σ_0^{15}
(d)

Parallel Runtime is

$$T_P = \Theta(n/p + \log p).$$

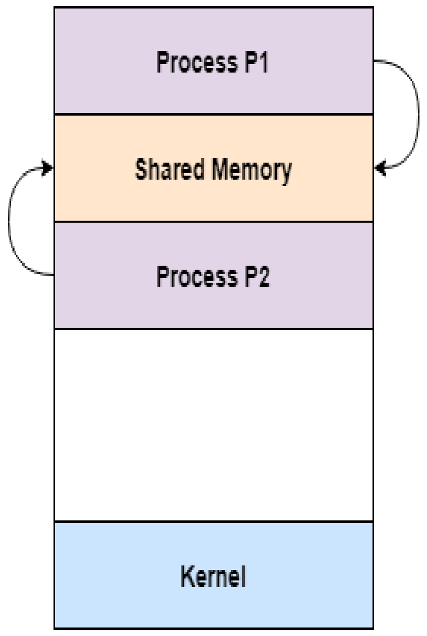
- ❑ A cost-optimal way of computing the sum of 16 numbers using four processing elements

Communication Paradigms for Distributed Computing

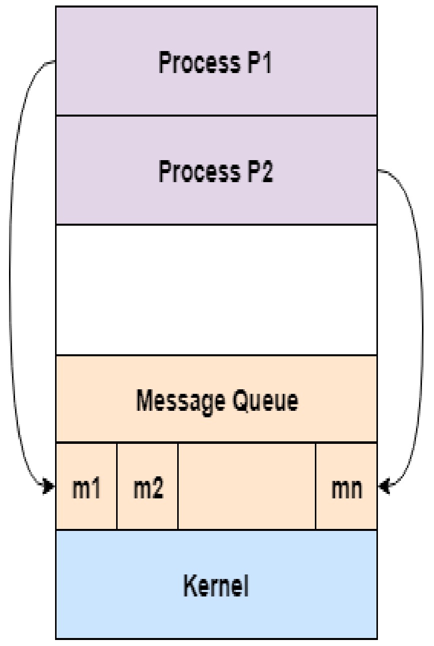
- Recap on interaction of software components in Distributed Computing (CORBA, RPC, DCOM,RMI etc.,)
- A distributed computation involves a number of processes communicating with one another and inter process communication mechanism is fairly complex
- many dimensions of variability in distributed systems like network topology, inter process communication mechanisms, failure classes, and security mechanisms
- Two major paradigms are:

1. **Shared Memory Process Communication Model**
2. **Message Passing Process Communication Model**

Shared Memory vs Message Passing



Shared Memory Model



Message Passing Model

Shared Memory Process Communication Model

Message Passing Process Communication Model

Shared Memory vs Message Passing

slno	Shared memory	Message Passing
1	Shared memory region is used for communication	Message passing facility is used for communication
2	For communication between processes on a single processor or multiprocessor systems where the communicating processes reside on the same machine as the communicating processes share a common address space	For a distributed environment where communicating processes reside on remote machines connected through a network
3	code for reading and writing the data from the shared memory should be written explicitly by the Application programmer	No such code required here as the message passing facility provides mechanism for communication and synchronization of actions performed by the communicating processes
4	provides maximum speed of computation as communication is done through shared memory so system calls are made only to establish the shared memory	time consuming as message passing is implemented through kernel intervention (system calls)
5	processes need to ensure that they are not writing to the same location simultaneously	useful for sharing small amounts of data as conflicts need not to be resolved
6	Faster communication strategy	Relatively slower communication strategy

Shared Memory vs Message Passing-Tradeoffs

	Shared Memory Systems	Message passing systems
Interface to Communication	implicit	explicit
Complexity of the Architecture	Leverages conventional architecture	Code needs to be rewritten
Convenience	Serial code runs without modification	Message passing libraries available
Protocols	Communication occurs as part of the shared memory system	Communication protocols are fully under user control
Prediction on future usage		Message passing with SMP

Message-passing and Shared Memory - emulation

- Emulating MP over SM:
  - Partition shared address space
  - Send/Receive emulated by writing/reading from special mailbox per pair of processes
- Emulating SM over MP:
  - Model each shared object as a process
  - Write to shared object emulated by sending message to owner process for the object
  - Read from shared object emulated by sending query to owner of shared object

Implementing shared memory paradigm

- SystemV IPC
- POSIX IPC
- OpenMP API

Implementing Message passing paradigm

- MPI
- MPICH

Classification of primitives

- Synchronous (send/receive)
  - Handshake between sender and receiver
  - Send completes when Receive completes
  - Receive completes when data copied into buffer
- Asynchronous (send)
  - Control returns to process when data copied out of user-specified buffer

- Blocking (send/receive)
  - Control returns to invoking process after processing of primitive (whether sync or async) completes
- Nonblocking (send/receive)
  - Control returns to process immediately after invocation
  - Send: even before data copied out of user buffer
  - Receive: even before data may have arrived from sender

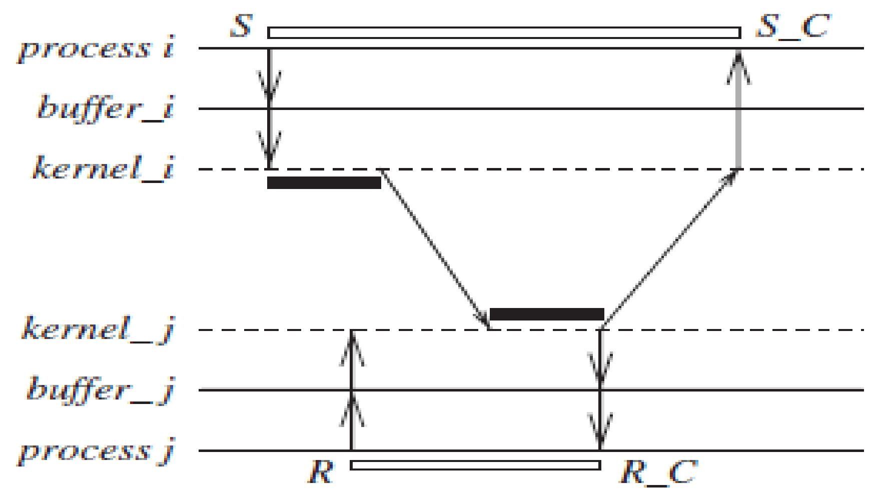
Non-blocking Primitive

nonblocking send primitive

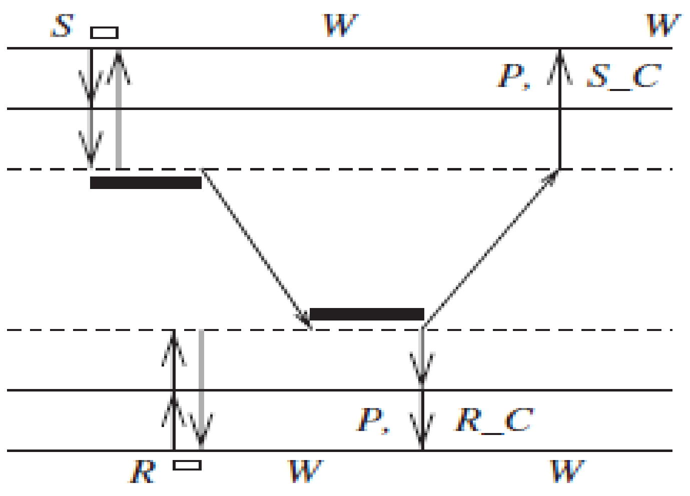
```
Send(X, destination, $handle_k$)          //$handle_k$ is a return parameter
...
...
Wait($handle_1$, $handle_2$, . . . , $handle_k$, . . . , $handle_m$)          //Wait always blocks
```

- ➢ When the Wait call returns, at least one of its parameters is posted
- ➢ Return parameter returns a system-generated handle
  - ➢ Use later to check for status of completion of call
  - ➢ Keep checking (loop or periodically) if handle has been posted
  - ➢ Issue Wait(handle1, handle2, . . .) call with list of handles
  - ➢ Wait call blocks until one of the stipulated handles is posted

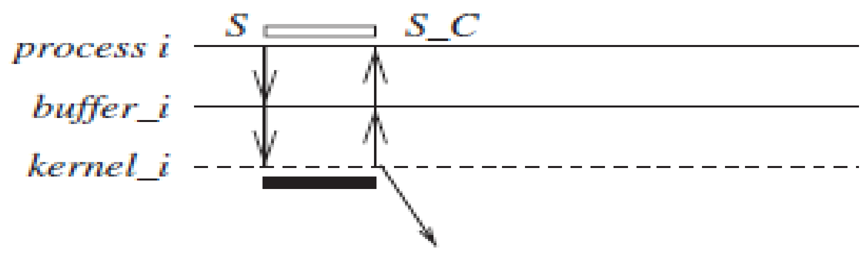
Blocking/non blocking; Synchronous/asynchronous; send/receive primitives



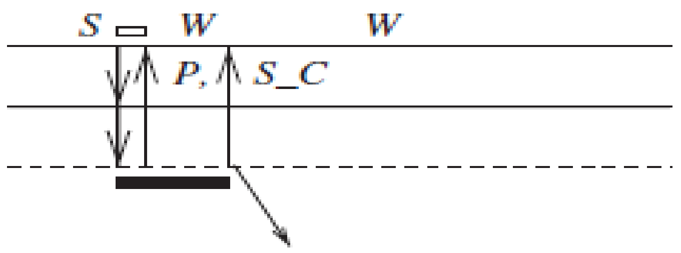
(a) Blocking sync. *Send*, blocking *Receive*



(b) Nonblocking sync. *Send*, nonblocking *Receive*



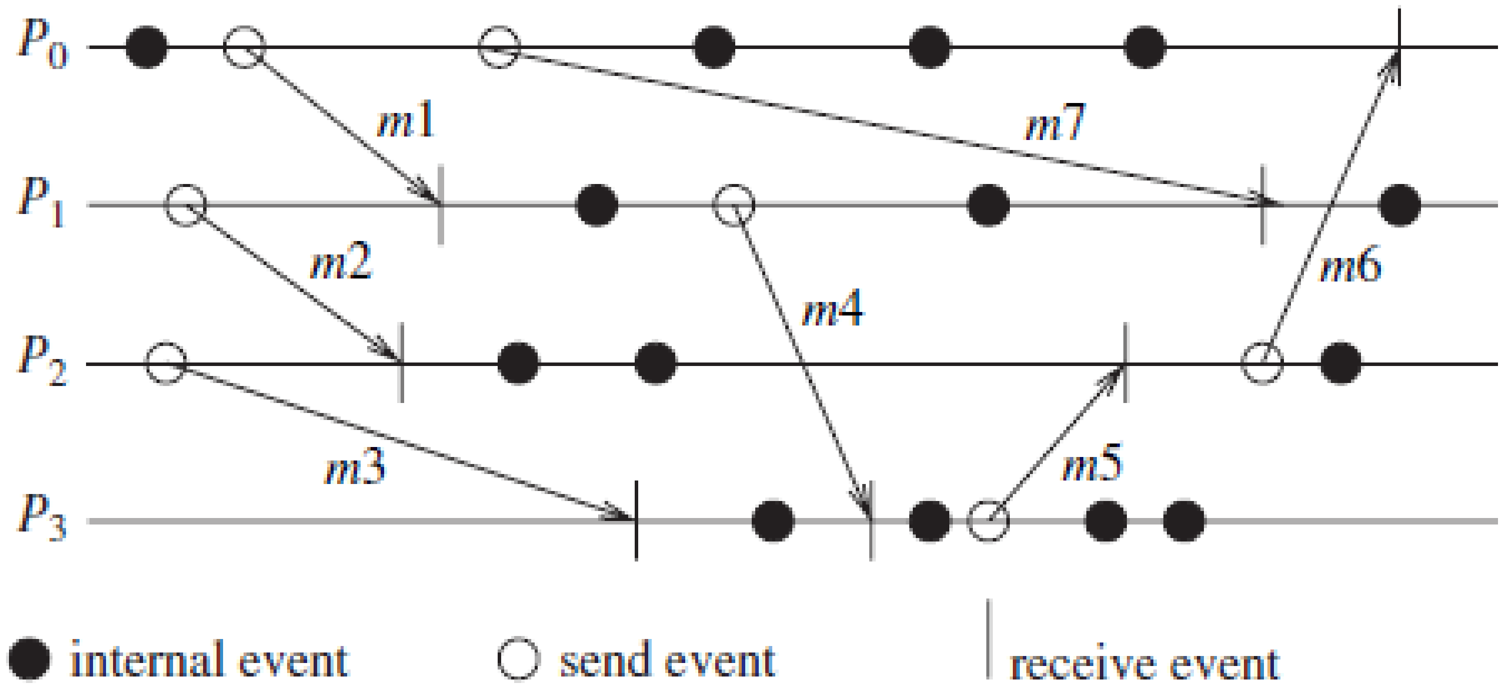
(c) Blocking async. *Send*



(d) Non-blocking async. *Send*

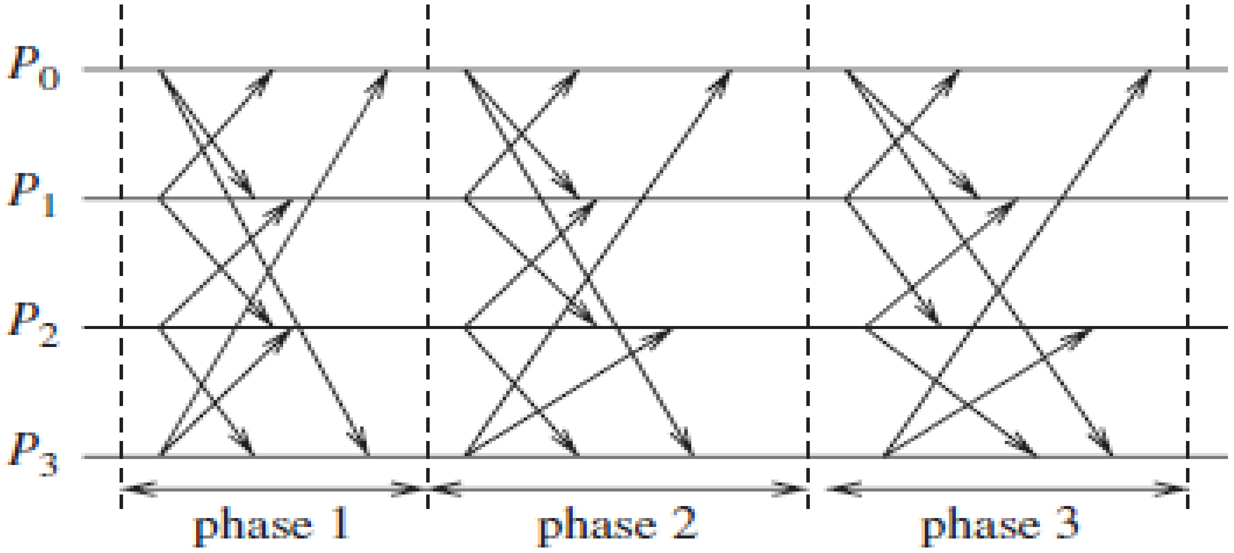
▬	Duration to copy data from or to user buffer		
▭	Duration in which the process issuing send or receive primitive is blocked		
S	*Send* primitive issued	*S_C*	processing for *Send* completes
R	*Receive* primitive issued	*R_C*	processing for *Receive* completes
P	The completion of the previously initiated nonblocking operation		
W	Process may issue *Wait* to check completion of nonblocking operation		

Asynchronous Executions in a Message-passing System



Processor synchrony → step

Synchronous Executions in a Message-passing System



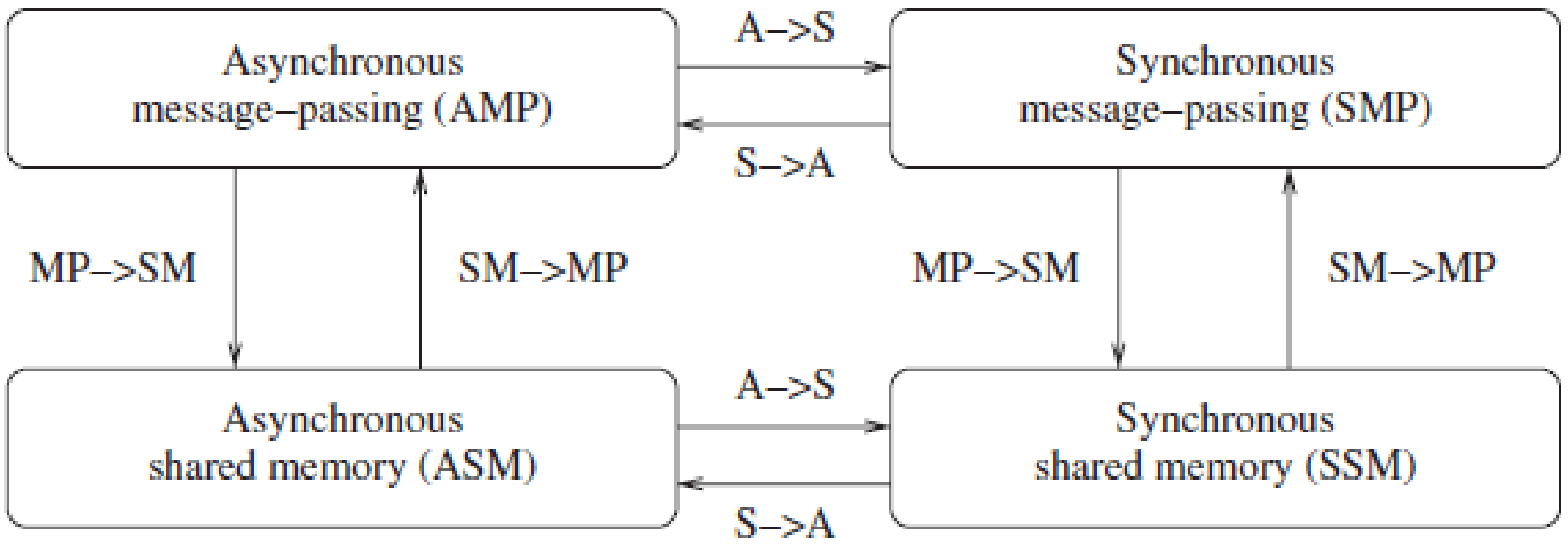
Synchronous execution in a message-passing system In any round/step/phase: (send | internal) ∗ (receive | internal)∗

Sync Execution(int k, n) //k rounds, n processes.

```
(1) for r = 1 to k do
(2)     proc i sends msg to (i + 1) mod n and (i − 1) mod n;
(3)     each proc i receives msg from (i + 1) mod n and (i − 1) mod n;
(4)     compute app-specific function on received values.
```

System emulations: virtual synchrony

Emulations among the principal system classes in a failure-free system



Asynchronous message–passing (AMP)
Synchronous message–passing (SMP)
Asynchronous shared memory (ASM)
Synchronous shared memory (SSM)
A–>S
S–>A
MP–>SM
SM–>MP
MP–>SM
SM–>MP
A–>S
S–>A

- ❖ Sync ↔ async, and shared memory ↔ msg-passing emulations
  Assumption: failure-free system
- ❖ System A emulated by system B
- ❖ If not solvable in B, not solvable in A
- ❖ If solvable in A, solvable in B

Design challenges – system perspective

- Communication mechanisms: E.g., Remote Procedure Call (RPC), remote object invocation (ROI), message-oriented vs. stream-oriented communication
- Processes: Code migration, process/thread management at clients and servers, design of software and mobile agents
- Naming: Easy to use identifiers needed to locate resources and processes transparently and scalably
- Synchronization
- Data storage and access
  - Schemes for data storage, search, and lookup should be fast and scalable across network
  - Revisit file system design
- Consistency and replication
  - Replication for fast access, scalability, avoid bottlenecks
  - Require consistency management among replicas
- Fault-tolerance: correct and efficient operation despite link, node, process failures
- Distributed systems security
  - Secure channels, access control, key management (key generation and key distribution), authorization, secure group management
- Scalability and modularity of algorithms, data, services
- Some experimental systems: Globe, Globus, Grid

Design challenges – algorithm perspective

- Useful execution models and frameworks: to reason with and design correct distributed programs
  - Interleaving model
  - Partial order model
  - Input/Output automata
  - Temporal Logic of Actions
- Dynamic distributed graph algorithms and routing algorithms
  - System topology: distributed graph, with only local neighborhood knowledge
  - Graph algorithms: building blocks for group communication, data dissemination, object location
  - Algorithms need to deal with dynamically changing graphs
  - Algorithm efficiency: also impacts resource consumption, latency, traffic, congestion

algorithm perspective – contd..

- Time and global state
  - 3D space, 1D time
  - Physical time (clock) accuracy
  - Logical time captures inter-process dependencies and tracks relative time progression
  - Global state observation: inherent distributed nature of system
  - Concurrency measures: concurrency depends on program logic, execution speeds within logical threads, communication speeds
- Group communication, multicast, and ordered message delivery
  - Group: processes sharing a context, collaborating
  - Multiple joins, leaves, fails
  - Concurrent sends: semantics of delivery order
- Monitoring distributed events and predicates
  - Predicate: condition on global system state
  - Debugging, environmental sensing, industrial process control, analyzing event streams
- Distributed program design and verification tools
- Debugging distributed programs

algorithm perspective – contd..

- Data replication, consistency models, and caching
  - Fast, scalable access;
  - coordinate replica updates;
  - optimize replica placement
- World Wide Web design: caching, searching, scheduling
  - Global scale distributed system; end-users
  - Read-intensive; prefetching over caching
  - Object search and navigation are resource-intensive
  - User-perceived latency
- Distributed shared memory abstraction
  - Wait-free algorithm design: process completes execution, irrespective of actions of other processes, i.e., $n - 1$ fault-resilience
  - Mutual exclusion
    - ★ Bakery algorithm, semaphores, based on atomic hardware primitives, fast algorithms when contention-free access
  - Register constructions
    - ★ Revisit assumptions about memory access
    - ★ What behavior under concurrent unrestricted access to memory? Foundation for future architectures, decoupled with technology (semiconductor, biocomputing, quantum . . .)
  - Consistency models:
    - ★ coherence versus access cost trade-off
    - ★ Weaker models than strict consistency of uniprocessors

Applications and emerging challenges

- Mobile systems
- Sensor networks
- Ubiquitous or pervasive computing
- Peer-to-peer computing
- Publish/subscribe, content distribution
- Distributed agents
- Distributed data mining
- Grid computing
- Cloud computing
- Internet of Things
- Security

Recap Quiz

1. Which of the following is not one of the communication mechanisms in distributed computing?

(a) RPC (b) RMI (c) ROI (d) RTI

2. Let Ts = 10 time units be the sequential time of execution in a distributed system and let Tp = 8 time units be the parallel time. What is the speedup? Assume other delays are negligible.

(a) 1.25 (b) 1.0 (c) 0.8 (d) 1.8

3. In distributed computing processor synchrony is achieved through units of execution called

(a) Task (b) step (c) process (d) thread

4. What is the granularity of a distributed computing environment if the amount of computation is 100 units and the amount of communication is 36 units?

(a) 0.36 (b) 2.78 (c) 1.36 (d) 0.64

5. The maximum number of tasks that can be executed simultaneously at any time in a distributed computing environment is called the degree of

(a) Efficiency (b) granularity (c) consistency (d) concurrency

Recap Quiz key

Q1	Q2	Q3	Q4	Q5
d	a	b	b	d