



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

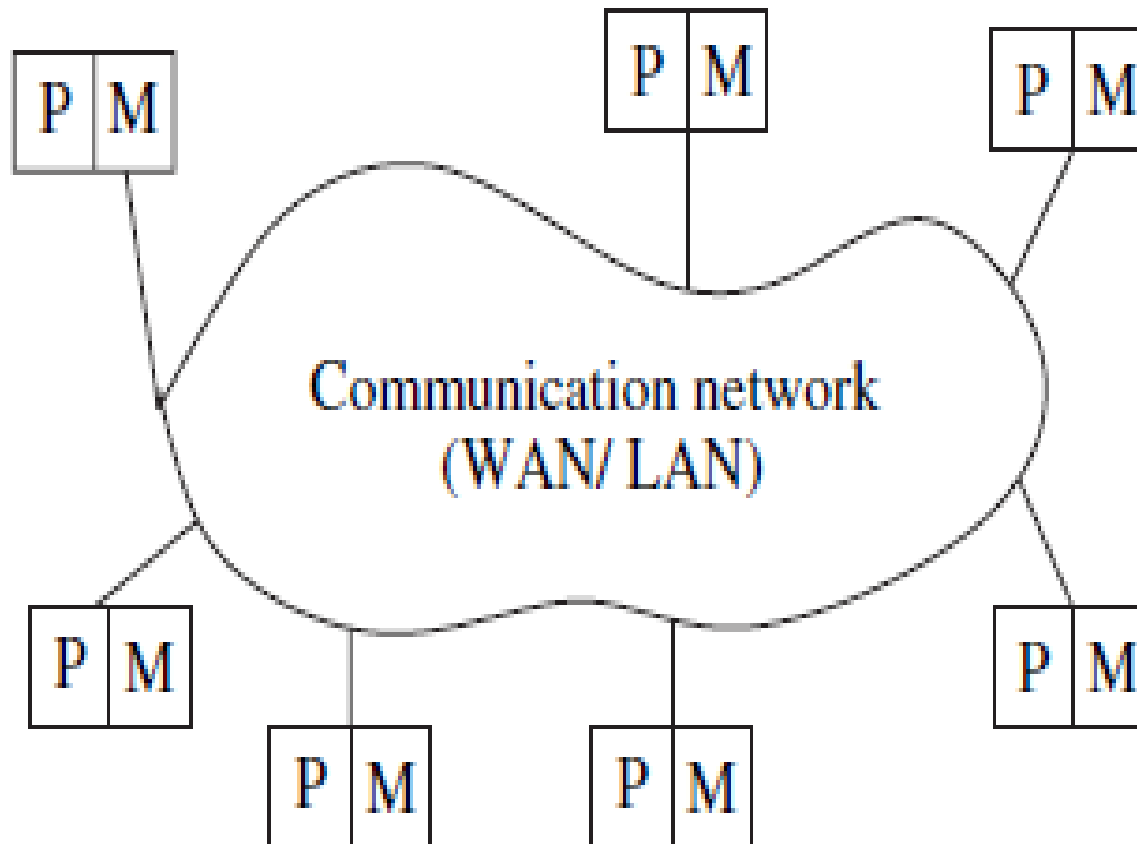


- ❖ Autonomous processors communicating over a communication network make a distributed system
 - ✓ No common physical clock
 - ✓ No shared memory
 - ✓ Geographical separation
 - ✓ Autonomy and heterogeneity

Ideal Distributed System?

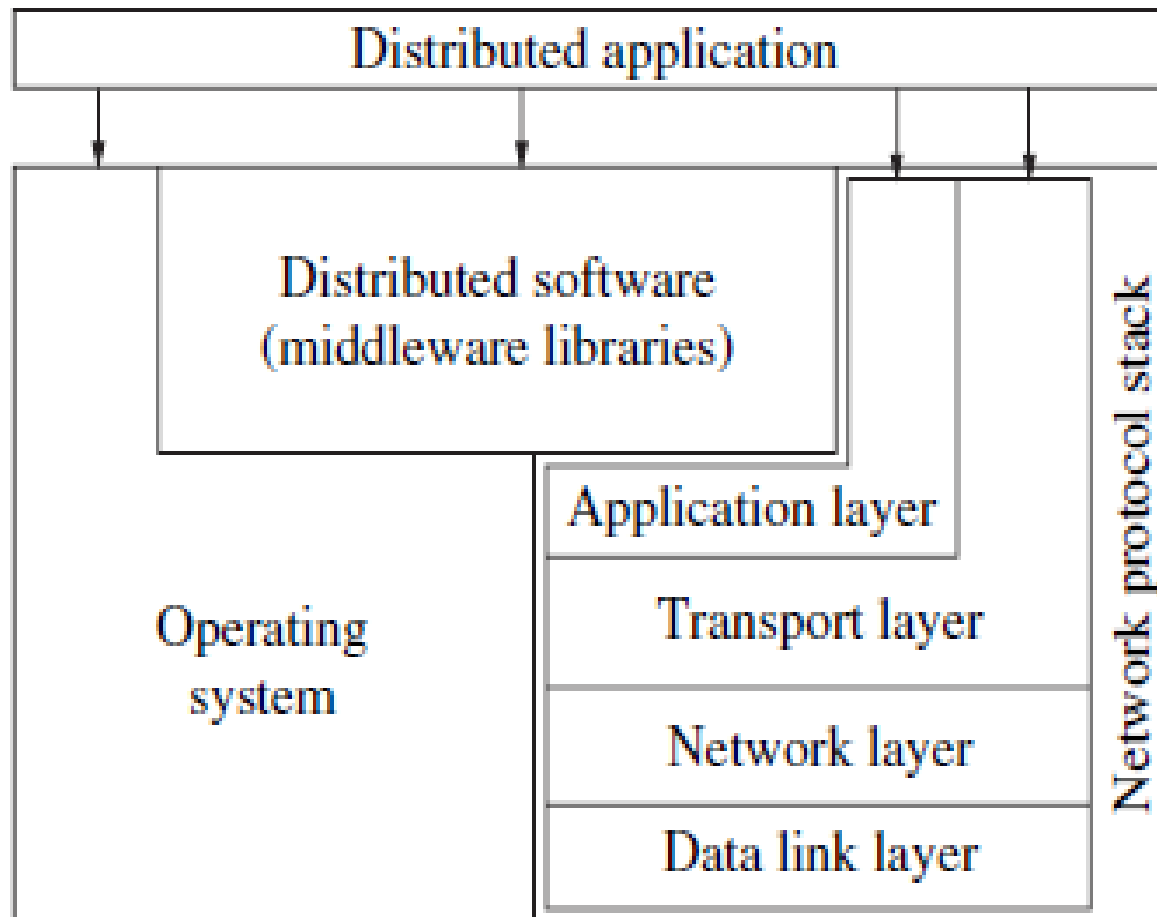


Distributed System model



P processor(s)
M memory bank(s)

Software Components and their interaction in Distributed environment

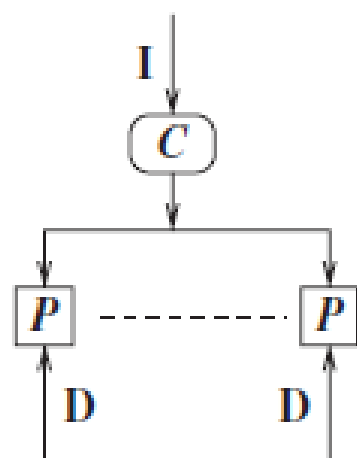




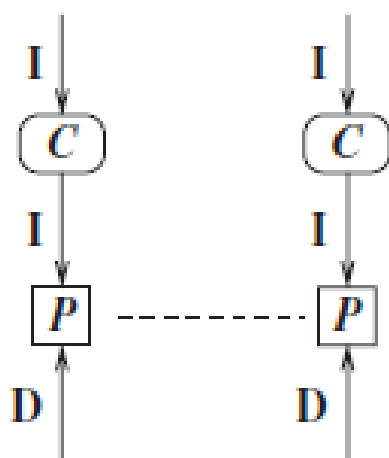
Why do we need distributed systems?

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

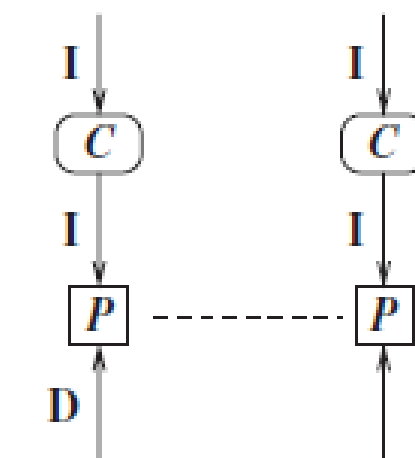
Flynn's Taxonomy





(a) SIMD



(b) MIMD



(c) MISD

 control unit
 processing unit
I instruction stream
D data stream

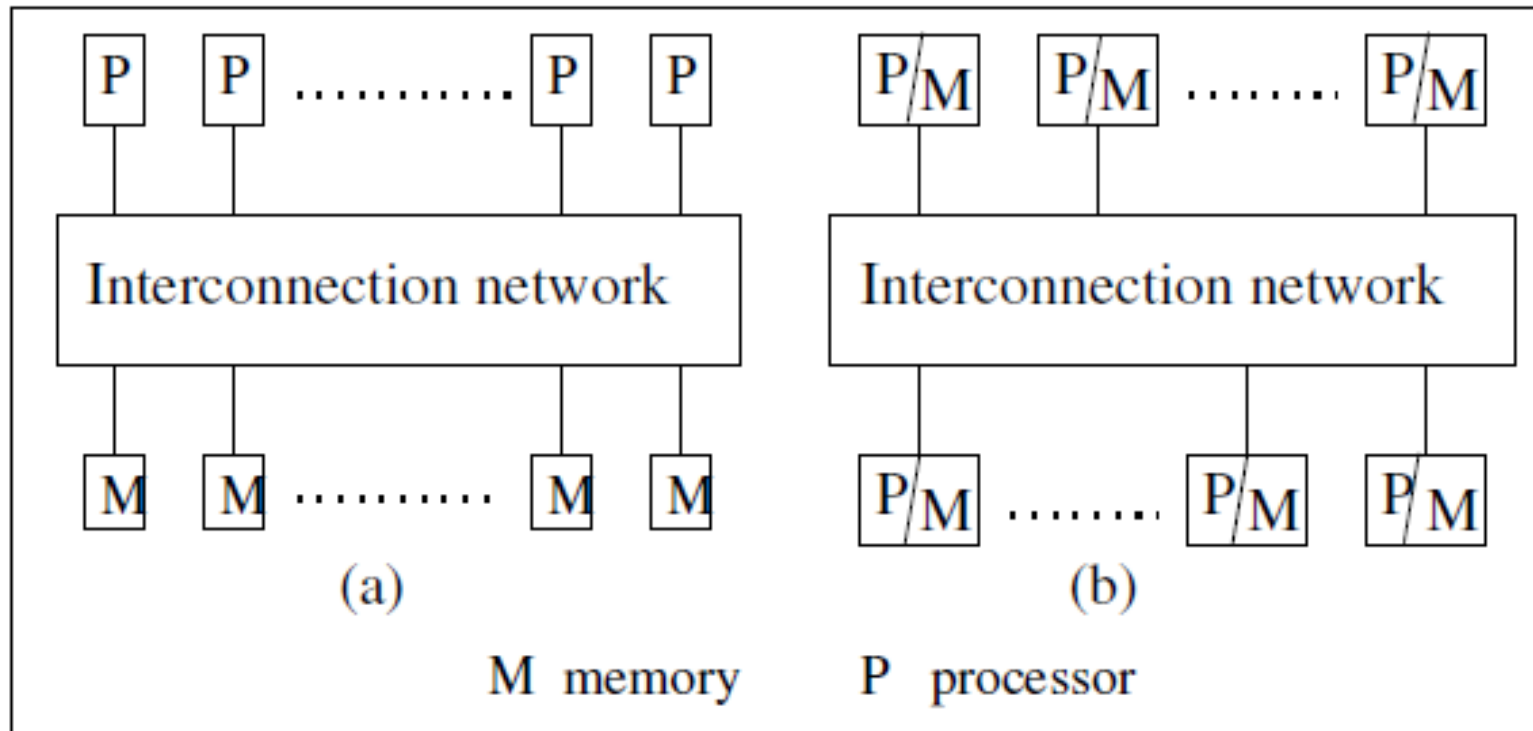
1. **Single instruction stream, single data stream (SISD)**
2. **Single instruction stream, multiple data streams (SIMD)**
 - ❖ Vector architectures
 - ❖ Multimedia extensions
 - ❖ Graphics processor units
3. **Multiple instruction streams, single data stream (MISD)**
 - ❖ No commercial implementation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



Parallel Systems

- Multiprocessor systems (direct access to shared memory, UMA model)
 - ▶ Interconnection network - bus, multi-stage switch
 - ▶ 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

Multiprocessor/Multi computer Systems



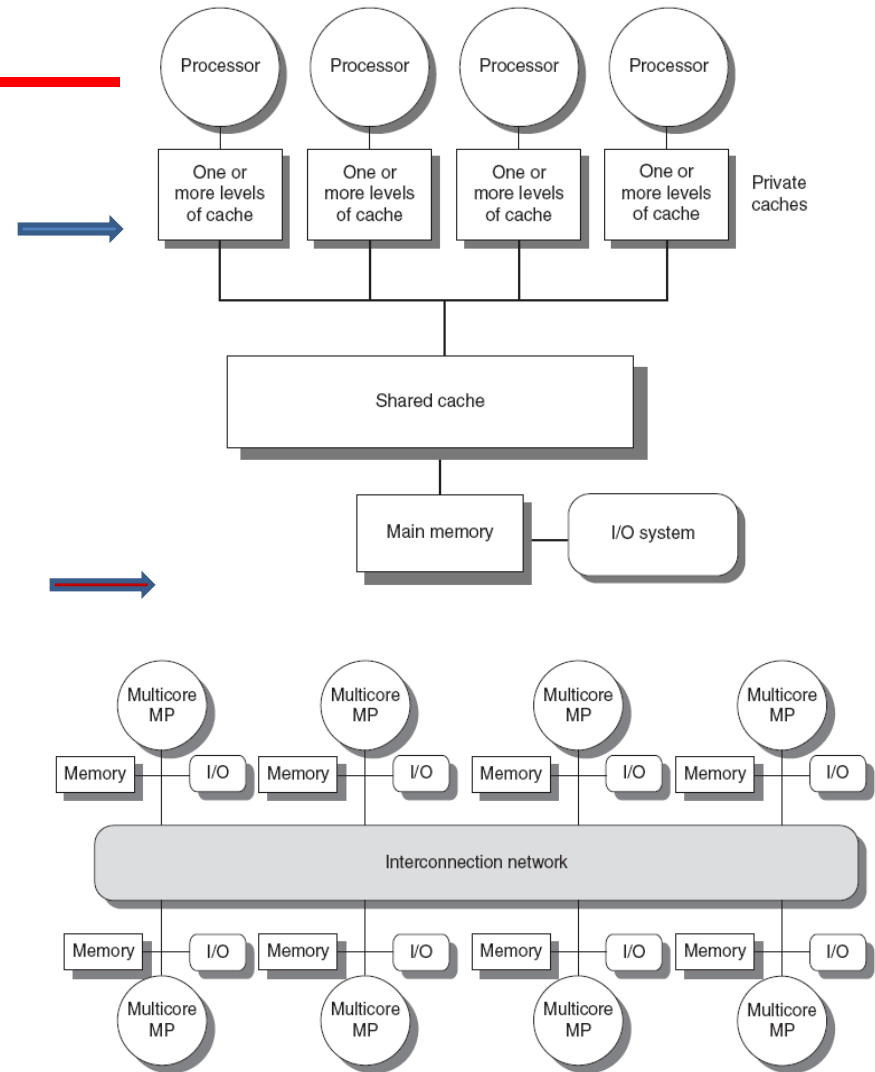
Two standard architectures for parallel systems. In both architectures, the processors may locally cache data from memory.

(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

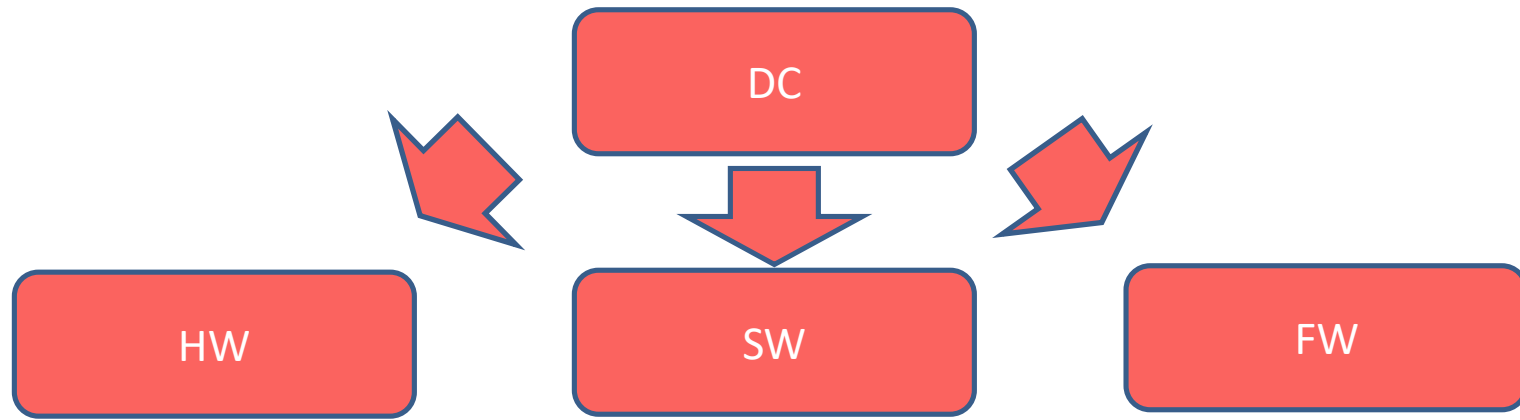


How does a distributed computing system work?



- 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?



© Can Stock Photo - csp11263846

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

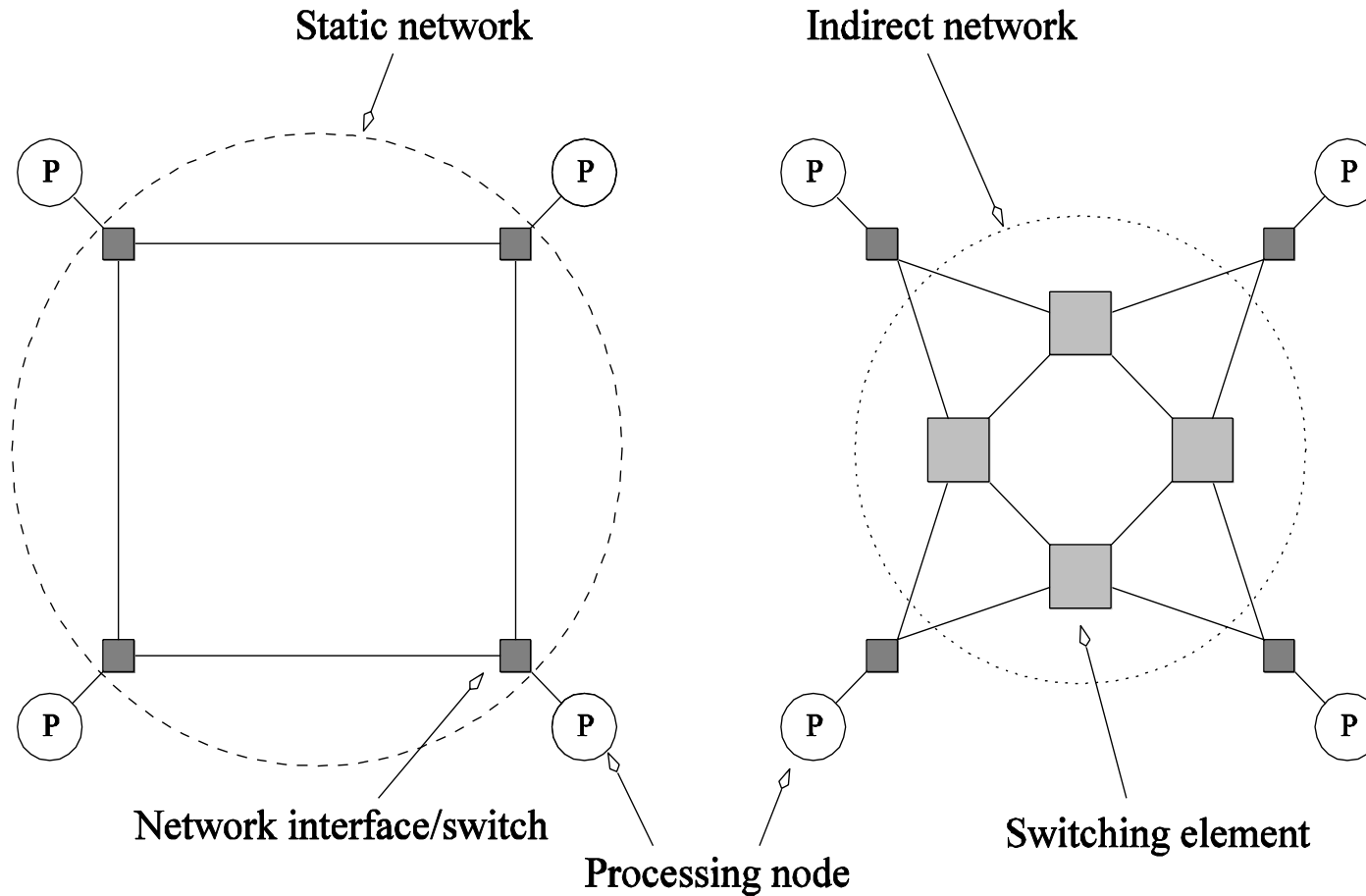
❖ **Indirect connection networks** – **Indirect (dynamic)** networks have no fixed neighbors. The communication topology can be changed dynamically based on the application demands. Ex:

➤ Bus networks

➤ Multistage networks

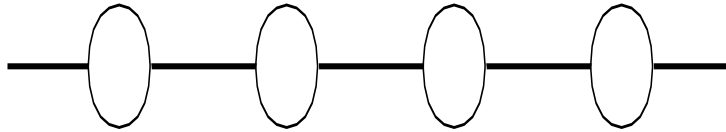
➤ Cross bars

Static and Dynamic Interconnection Networks

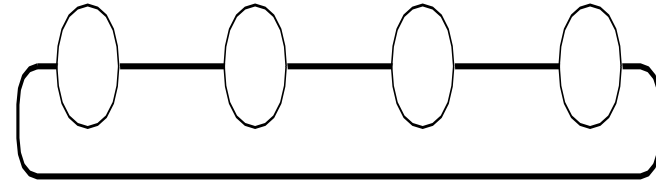


Classification of interconnection networks:
(a) static network (b) dynamic network

Network Topologies: Linear Arrays



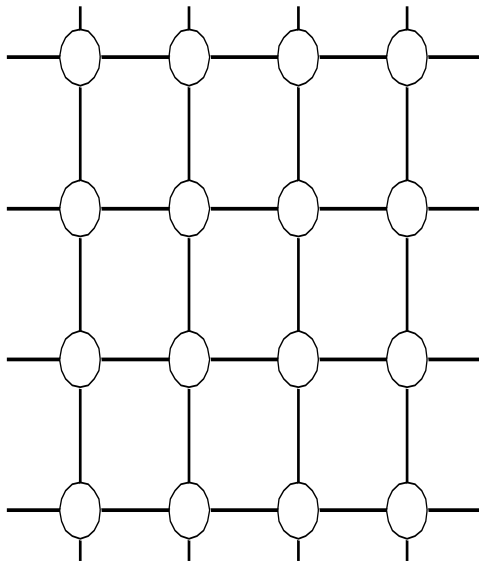
(a)



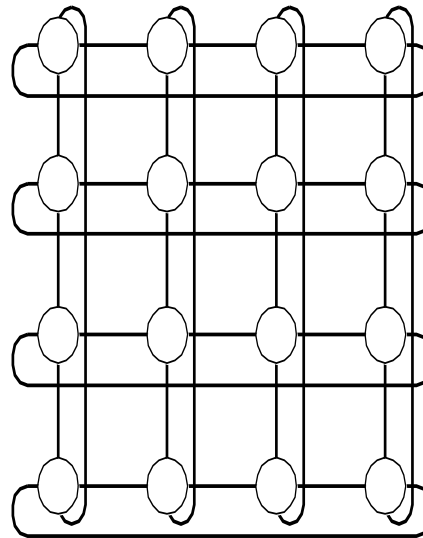
(b)

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

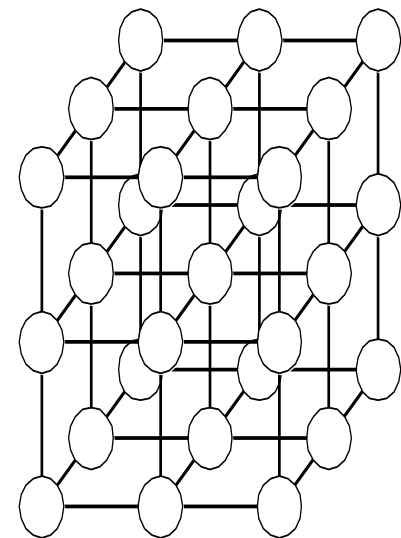
Network Topologies: Two- and Three Dimensional Meshes



(a)



(b)

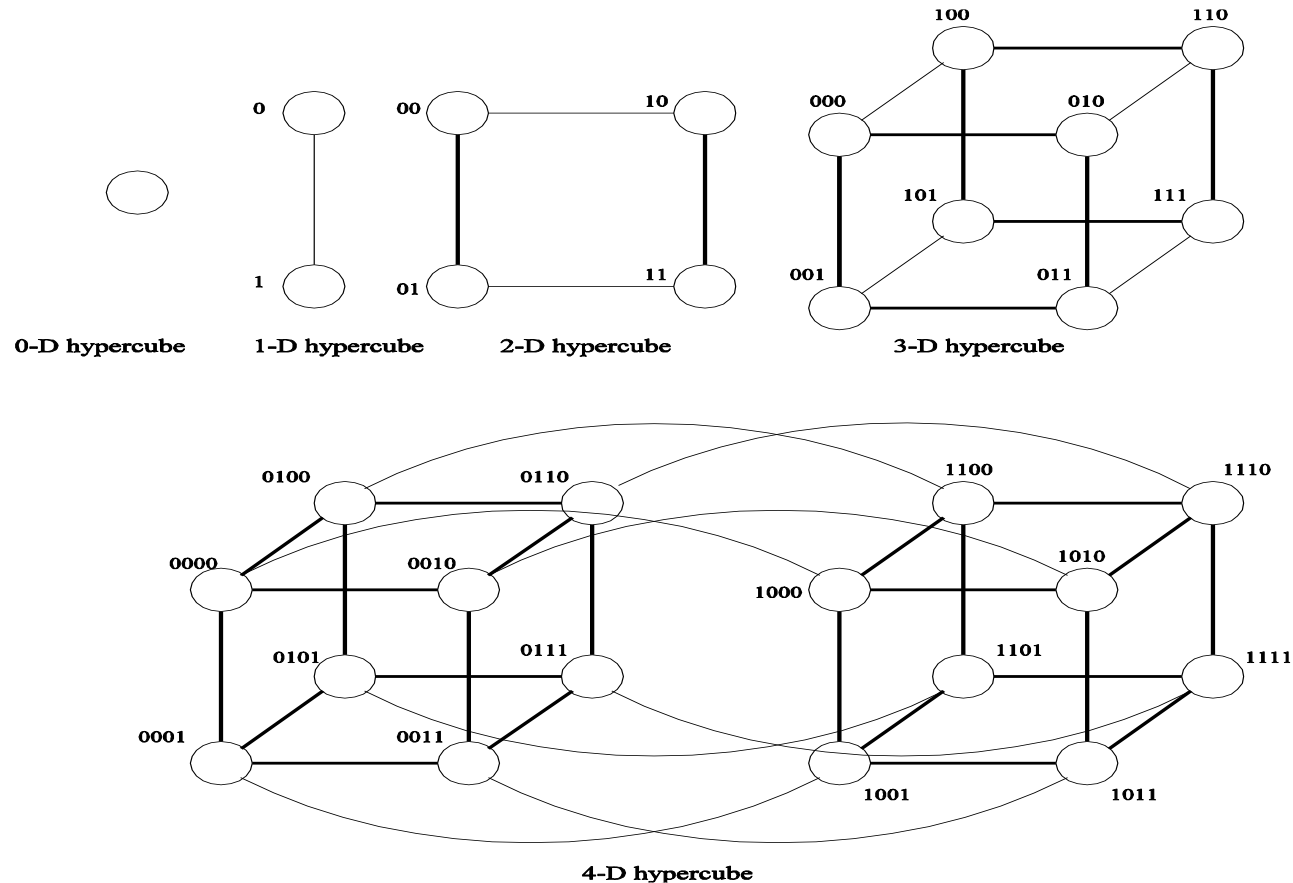


(c)

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.

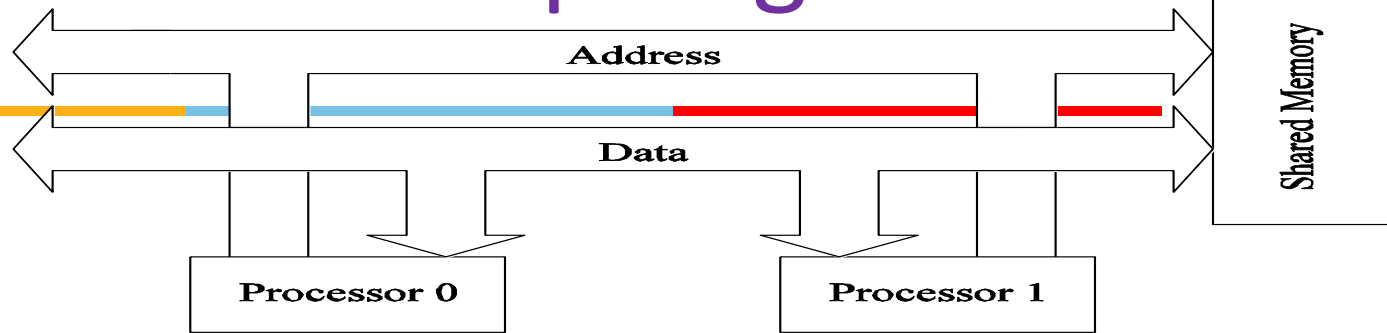
(3-D weather modelling, structural modelling physical simulations)

Network Topologies: Hypercubes and their Construction

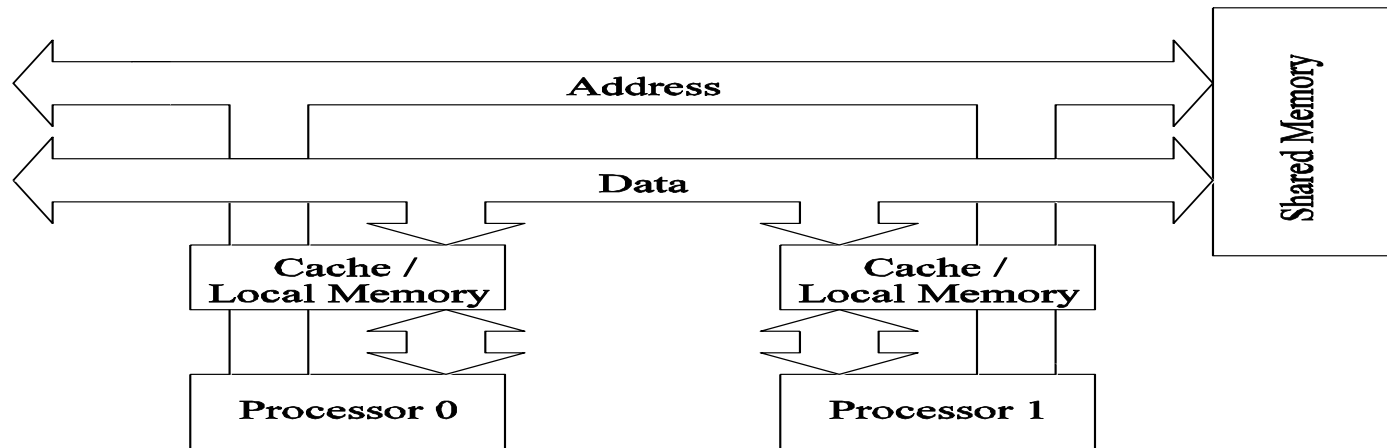


Construction of hypercubes from hypercubes of lower dimension.

Network Topologies: Buses



(a)



(b)

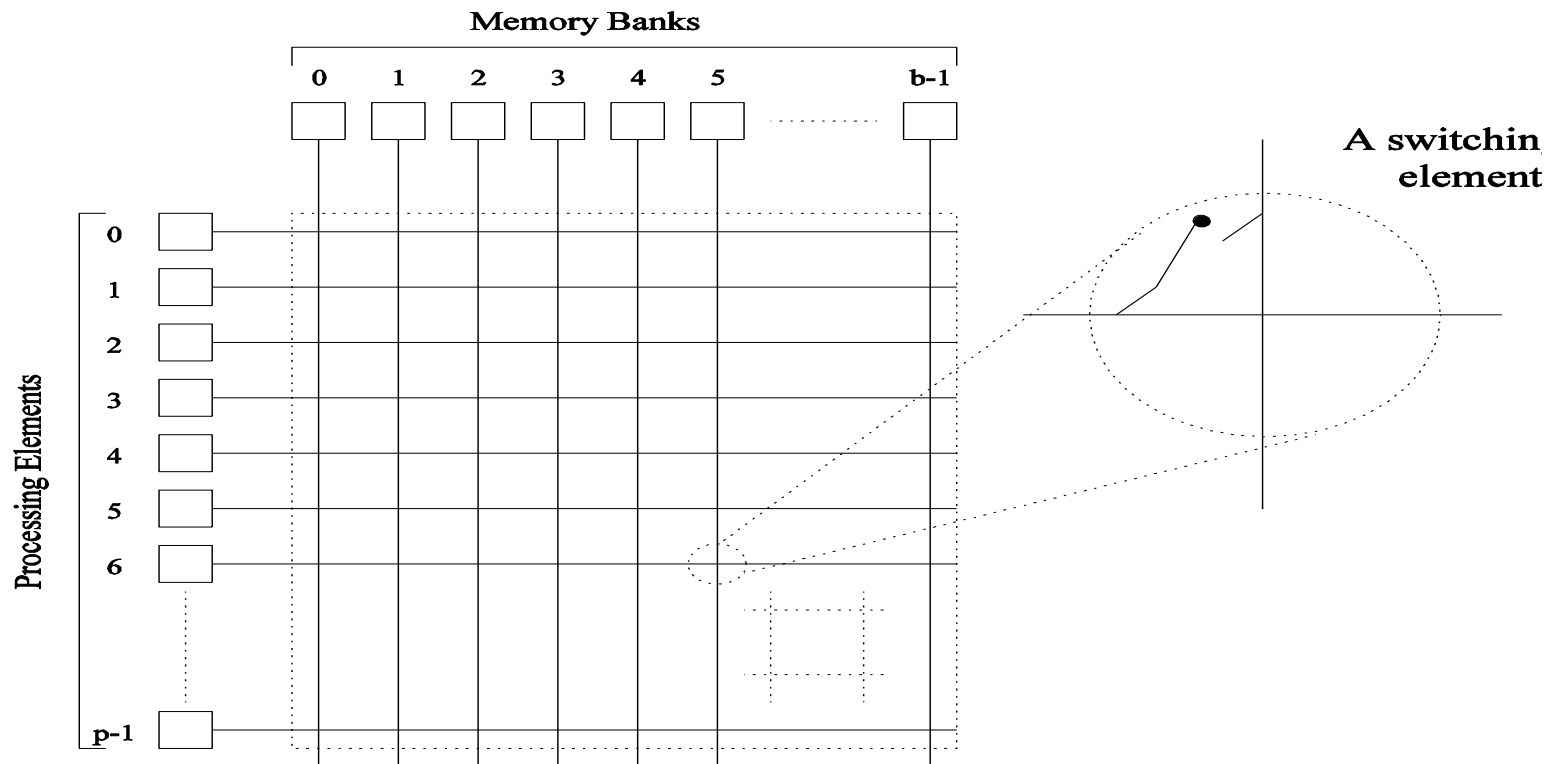
Bus-based interconnects

(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.

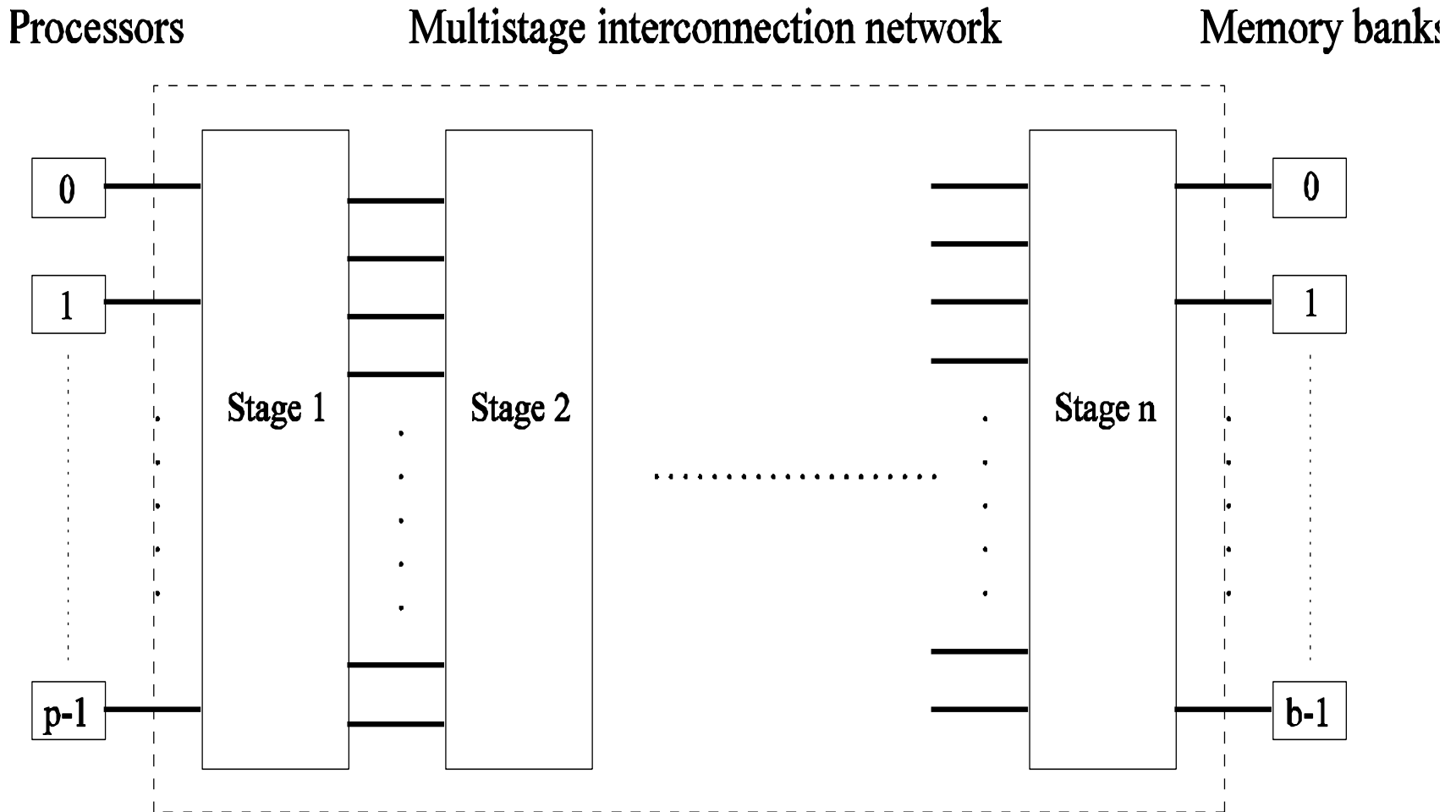
Network Topologies: Crossbars

A crossbar network uses an $p \times m$ 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



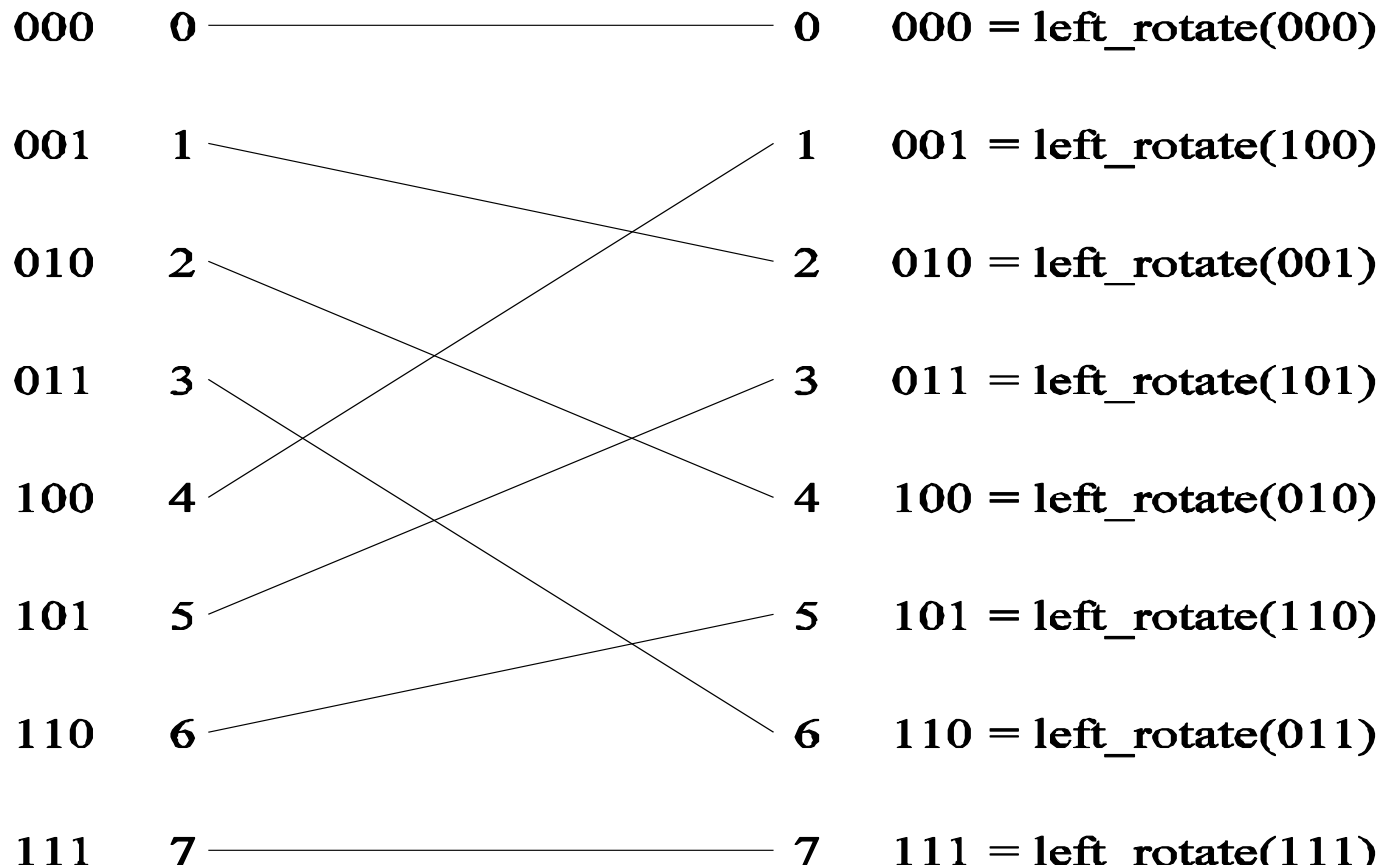
- ❖ One of the most commonly used multistage interconnects is the Omega network.
- ❖ This network consists of $\log p$ stages, where p is the number of inputs/outputs (processing nodes as well as memory banks)
- ❖ At each stage, input i is connected to output j if:
$$j = \begin{cases} 2i, & 0 \leq i \leq p/2 - 1 \\ 2i + 1 - p, & p/2 \leq i \leq p - 1 \end{cases}$$
- ❖ 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:



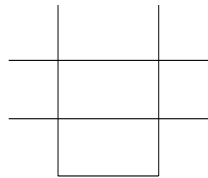
A perfect shuffle interconnection for eight inputs and outputs.

Network Topologies:

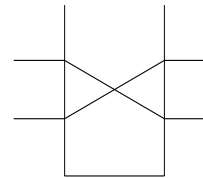
Multistage Omega Network



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



(a)



(b)

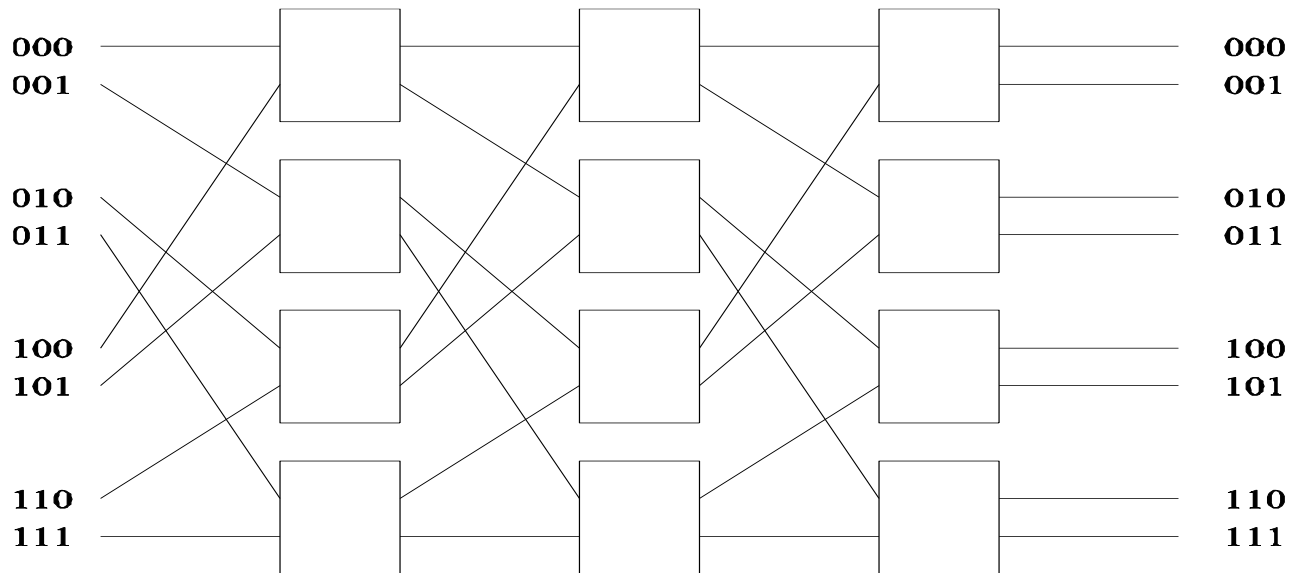
Two switching configurations of the 2×2 switch:
(a) Pass-through; (b) Cross-over.

Network Topologies:

Multistage Omega Network



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



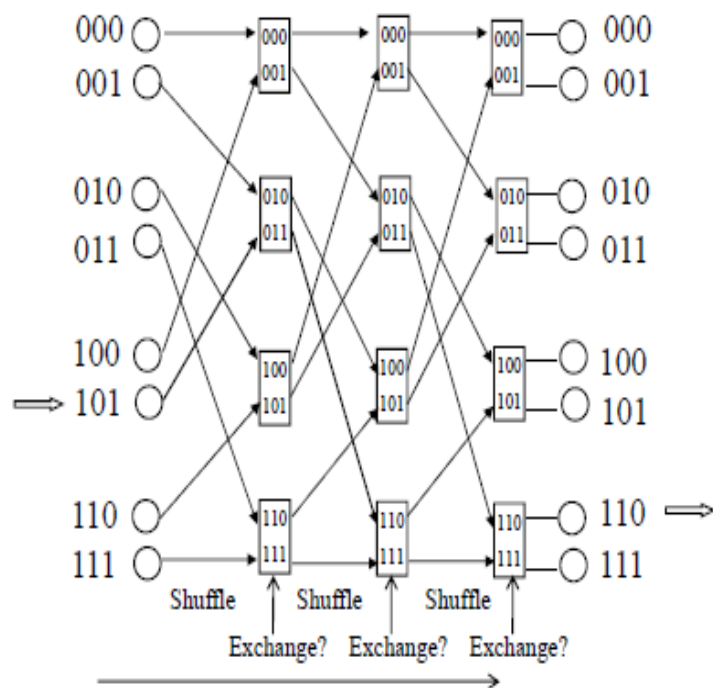
A complete omega network connecting eight inputs and eight outputs.

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

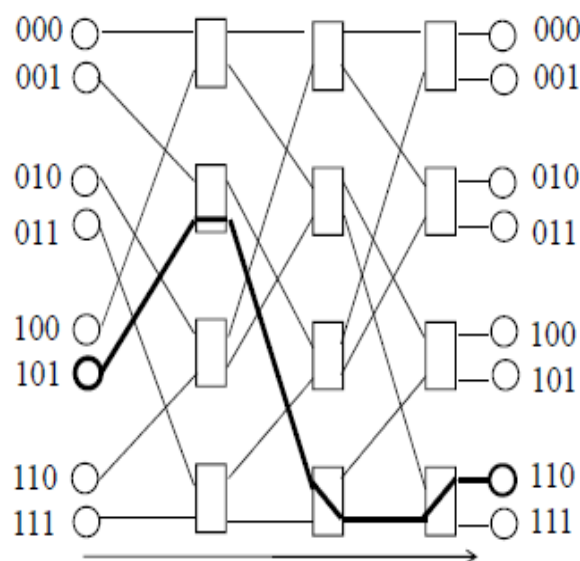
Routing in Omega network



((Unique route between a source and a destination - rules))



$\text{Shuffle}(x_{q-1}, x_{q-2}, \dots, x_0) = x_{q-2}, \dots, x_0, x_{q-1}$
 $\text{Exchange}(x_{q-1}, x_{q-2}, \dots, x_0) = x_{q-1}, x_{q-2}, \dots, \bar{x}_0$

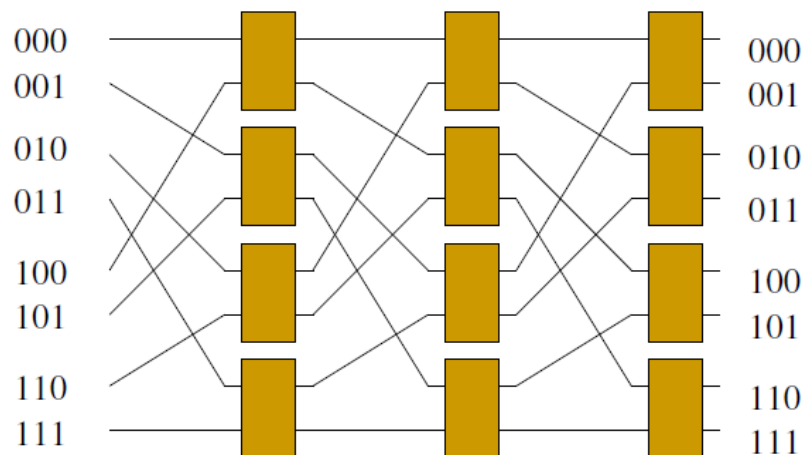


The destination $d_2 d_1 d_0$ is coded in message header

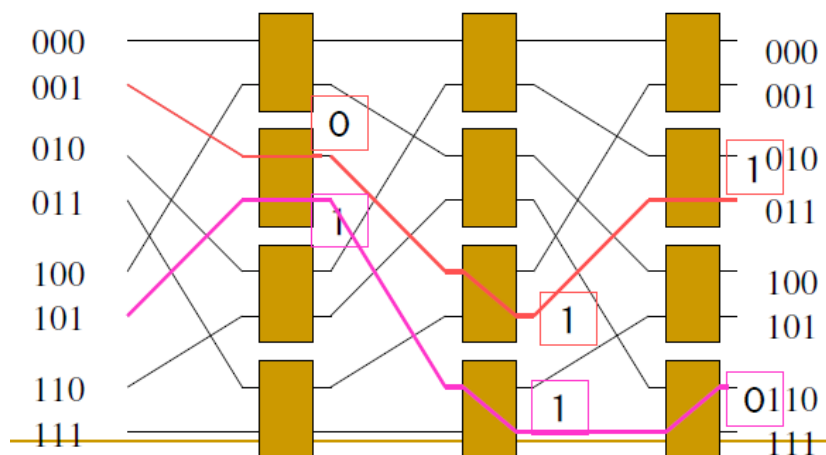
Example: to route from source 101 to destination 110 ($101 \text{ xor } 110 = 011$)

$101 \rightarrow 011 \rightarrow 011 \rightarrow 110 \rightarrow 111 \rightarrow 111 \rightarrow 110$
 shuffle straight shuffle exchange cross

Omega network - example



(a)



(b)

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

Answer = $\frac{1}{2} n \log_2 n = 12$

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

First step is to take the xor of the source and destination, which will give the sequence.

Ex: 1 \Rightarrow 3

Answer: 001- \rightarrow 010- \rightarrow 010- \rightarrow 100- \rightarrow 101- \rightarrow 011- \rightarrow 011 (the sequence is straight-cross-straight)

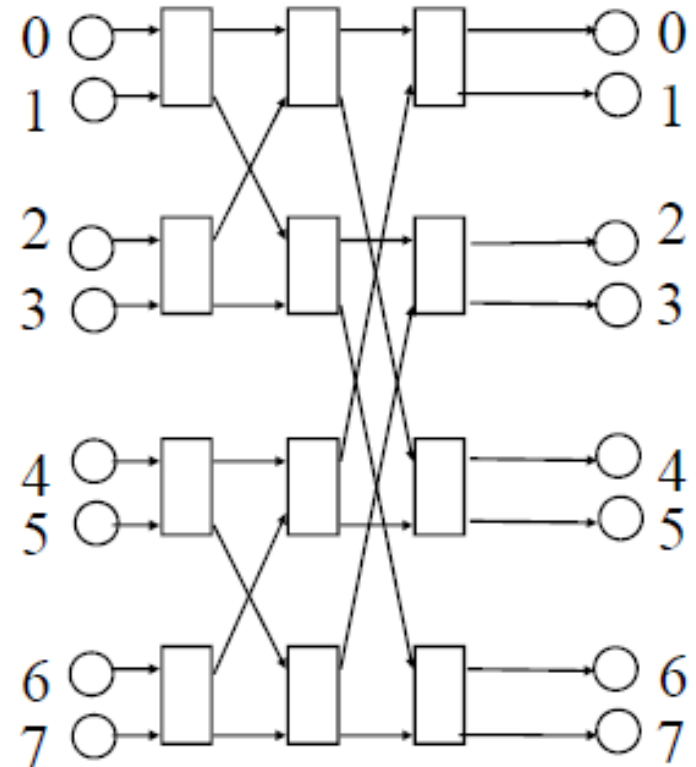
5 \Rightarrow 6

Answer: 101- \rightarrow 011- \rightarrow 011- \rightarrow 110- \rightarrow 111- \rightarrow 111- \rightarrow 110 (the sequence is straight-cross-cross)

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



A butterfly network

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
- **$S = T_{\text{serial}} / T_{\text{parallel}}$**

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

Granularity in Distributed Computing



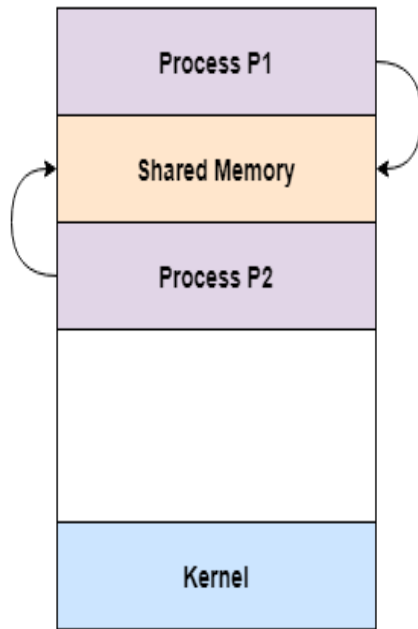
- ❖ The ratio of the amount of computation to the amount of communication within the parallel/distributed program is termed as granularity
- ❖ If the degree of parallelism is coarse-grained (fine-grained), there are relatively many more (fewer) productive CPU instruction executions, compared to the number 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

Communication Paradigms for Distributed Computing



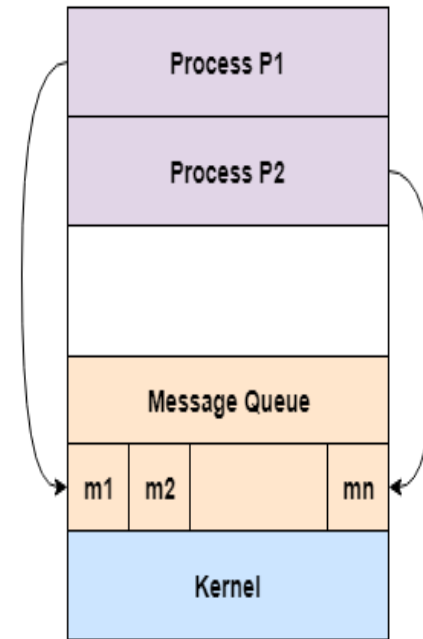
- ❖ Recap on interaction of software components in Distributed Computing (CORBA, RPC, DCOM, RMI etc.,)
- ❖ A distributed computation involves a number of processes communicating with one another and inter process communication mechanism is fairly complex
- ❖ many **dimensions of variability** in distributed 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

**Shared Memory
Process Communication
Model**



Message Passing Model

**Message Passing
Process Communication
Model**

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, handlek)           // handlek is a return parameter
...
...
Wait(handle1, handle2, ..., handlek, ..., handlem)    // Wait always blocks
```

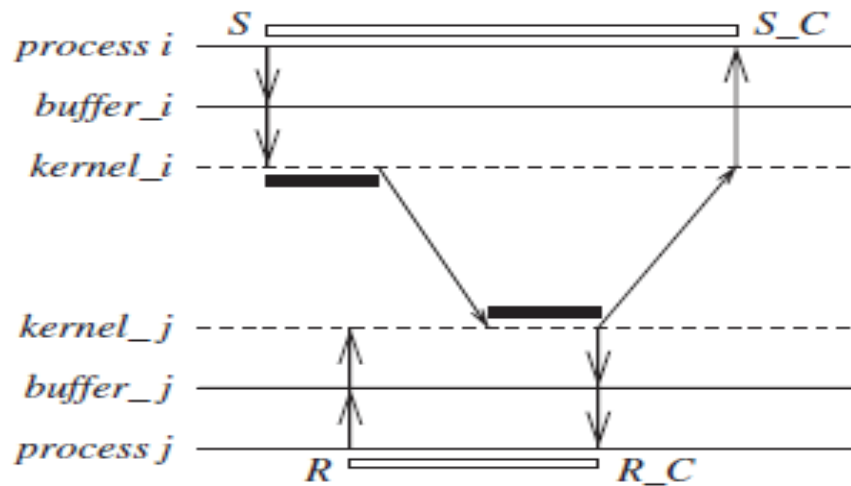
- When the Wait call returns, at least one of its parameters is posted
- Return parameter returns a system-generated handle
 - Use later to check for status of completion of call
 - Keep checking (loop or periodically) if handle has been posted
 - Issue Wait(*handle₁*, *handle₂*, . . .) call with list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

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

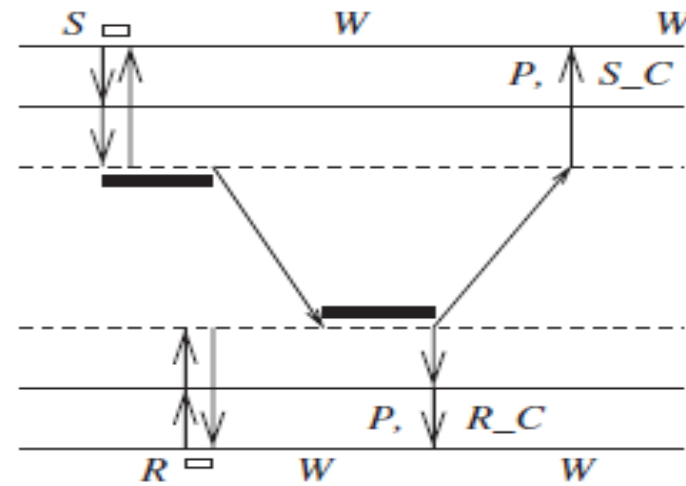
innovate

achieve

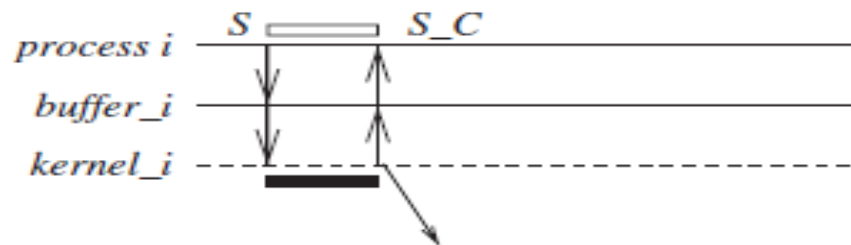
lead



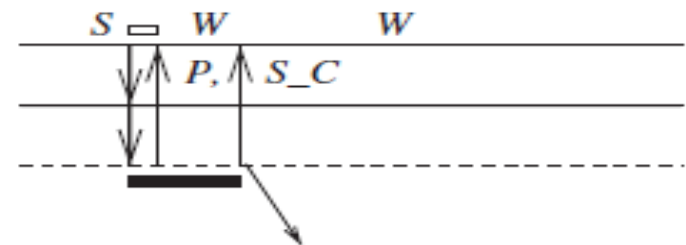
(a) Blocking sync. *Send*, blocking *Receive*



(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

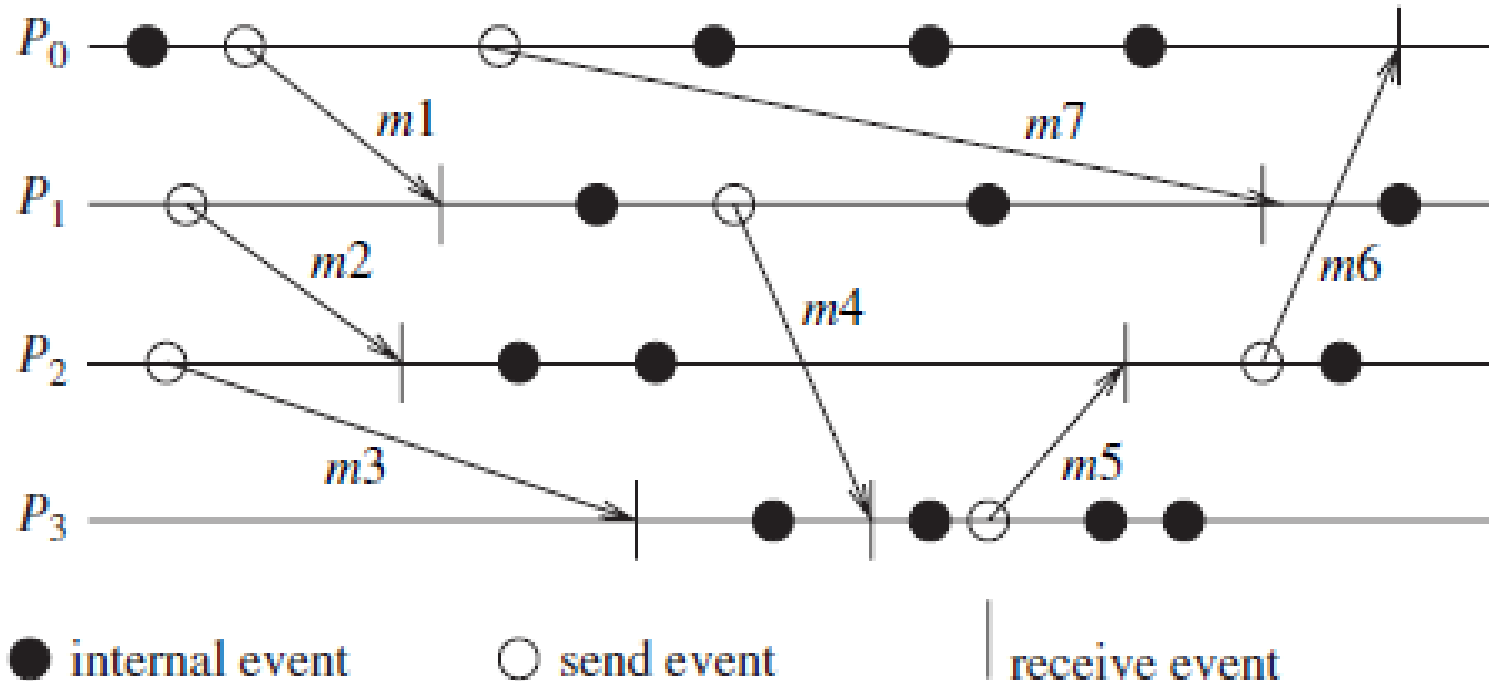
R *Receive* primitive issued

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

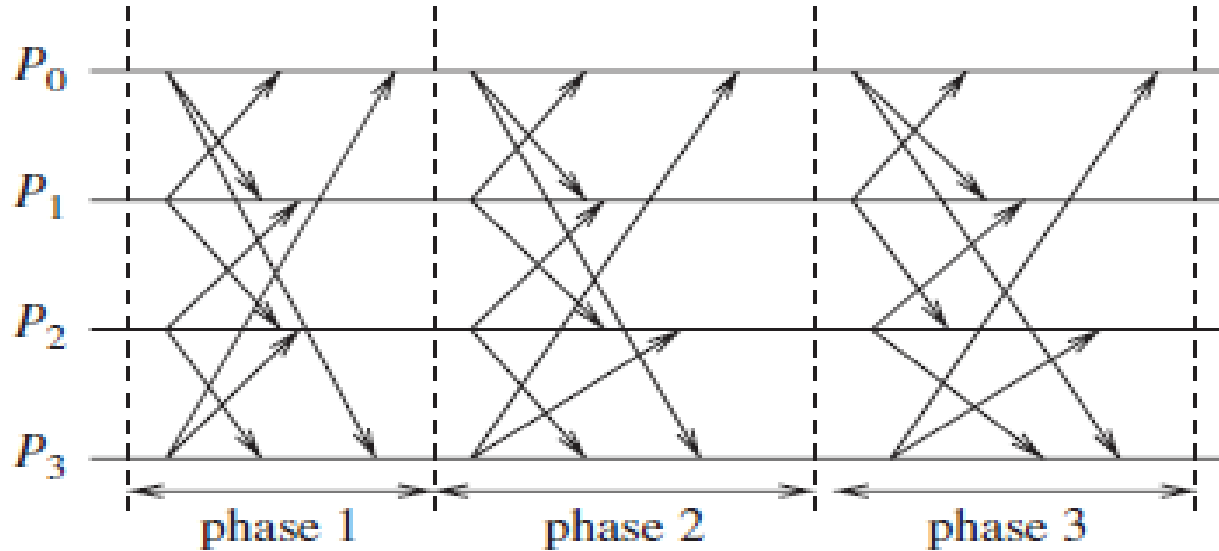


Processor synchrony



step

Synchronous Executions in a Message-passing System



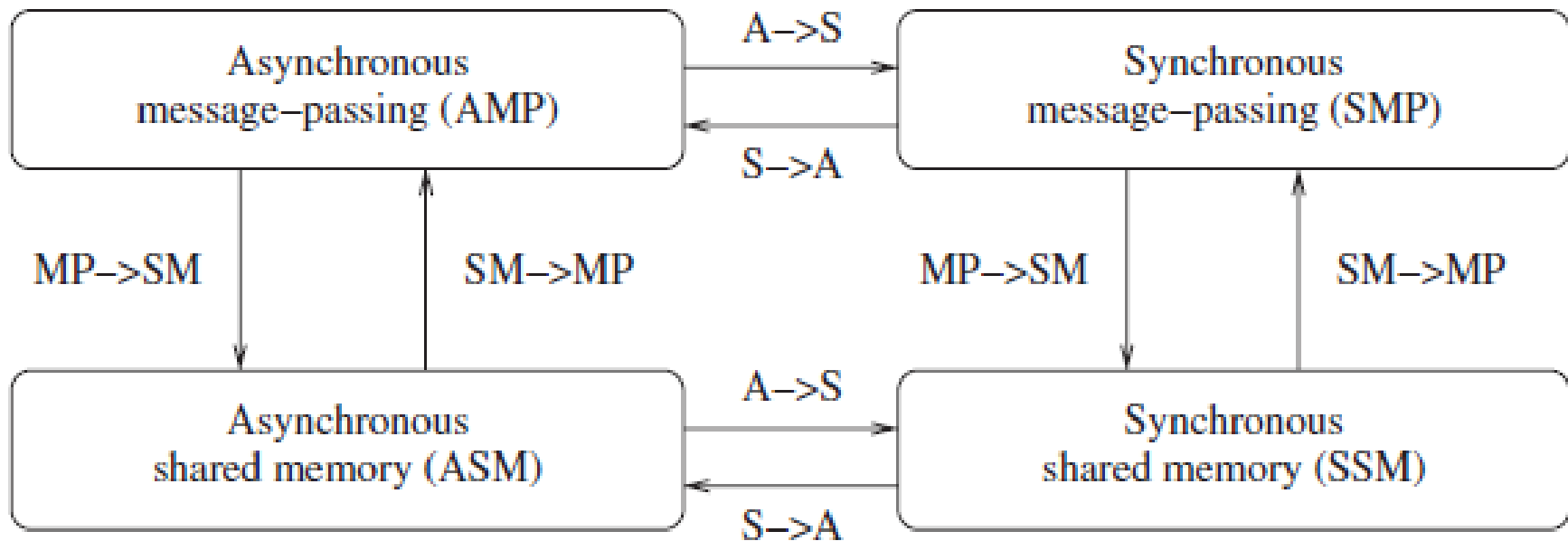
Synchronous execution in a message-passing system In any round/step/phase: (send | internal) * (receive | internal)*

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

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

System emulations: **virtual synchrony**

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

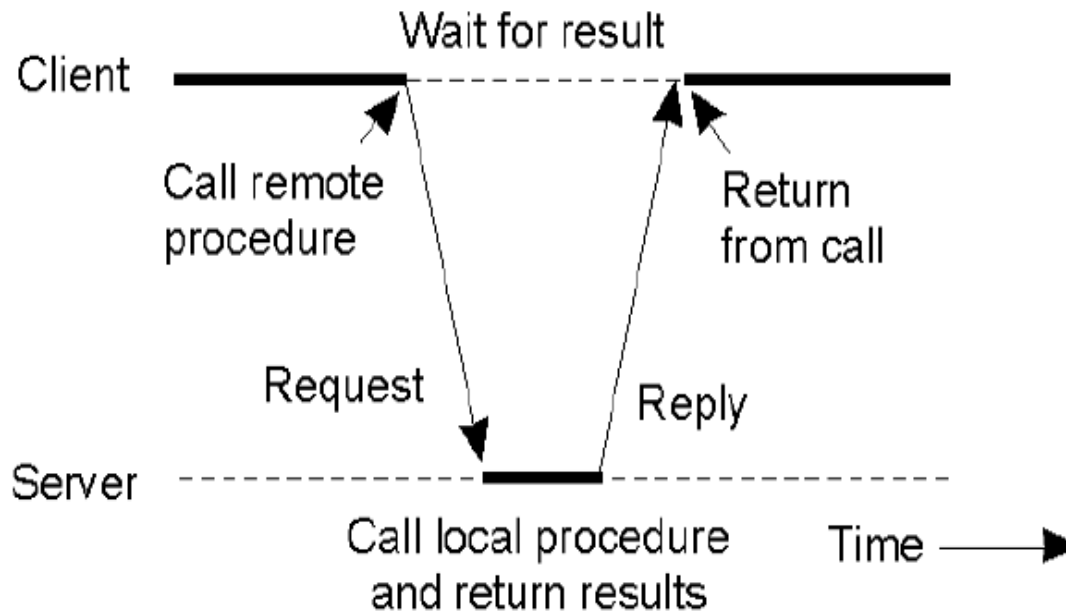


- ❖ Sync \leftrightarrow async, and shared memory \leftrightarrow msg-passing emulations
Assumption: failure-free system
- ❖ System A emulated by system B
- ❖ If not solvable in B, not solvable in A
- ❖ If solvable in A, solvable in B

Distributed Communication Models

- ❖ Remote Procedure Call (RPC)
- ❖ Publish/Subscribe Model
- ❖ Message Queues

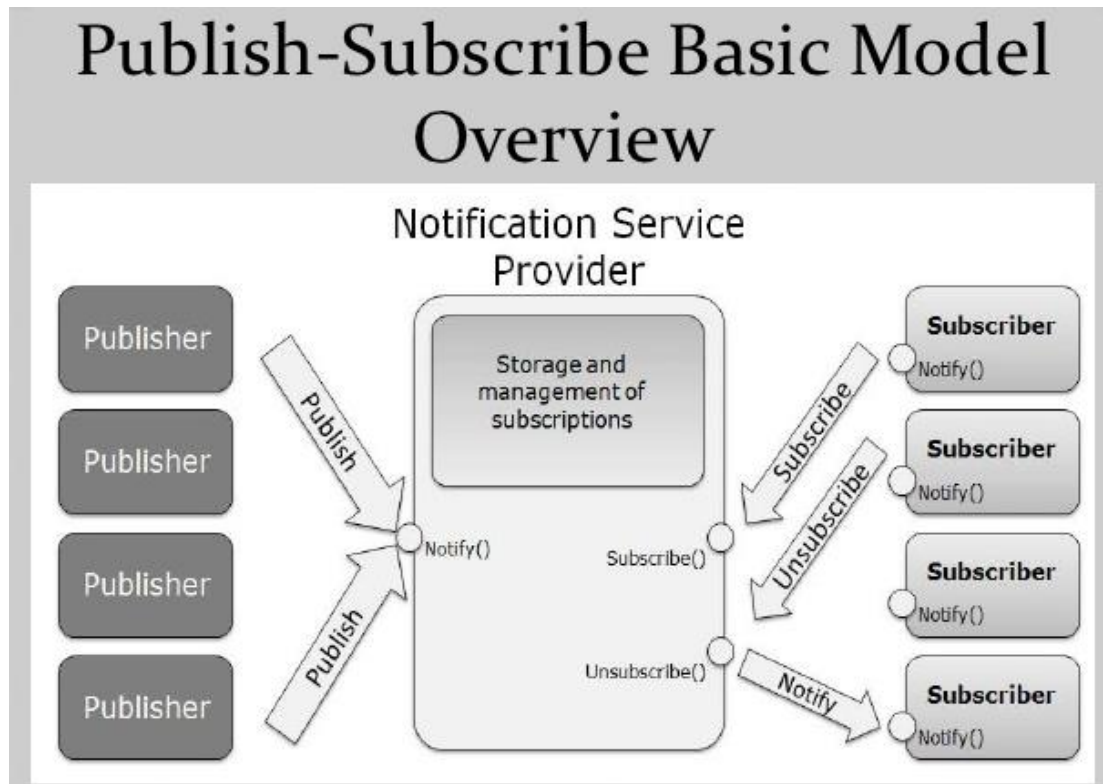
RPC between Client & Server



Source: <http://web.cs.wpi.edu/~rek/DCS/D04/Communication.pdf>

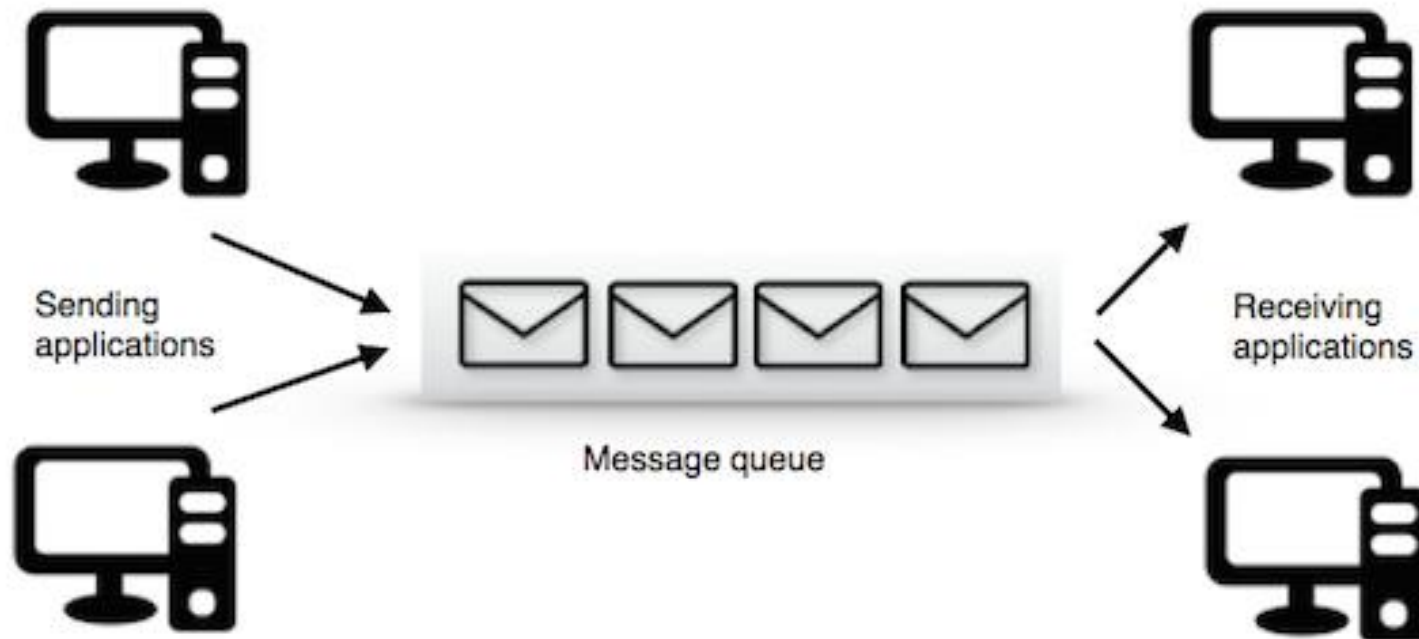
Real life example for RPC: Use of RPC in CERN Research

Publish/ Subscribe Model



Real life examples of Pub/Sub model: Google Cloud Pub/Sub

Distributed Message Queues



Real world example: ERP servers sending messages to CRM

Design Issues & Challenges – System Perspective



- ❖ Communication mechanisms
- ❖ Processes
- ❖ Naming
- ❖ Synchronization
- ❖ Data storage and access
- ❖ Consistency and replication
- ❖ Fault-tolerance
- ❖ Security
- ❖ Scalability and modularity
- ❖ API and transparency

Algorithmic Challenges

- ❖ Designing useful execution models and frameworks: to reason with and design correct distributed programs
- ❖ Dynamic distributed graph algorithms and routing algorithms
- ❖ Load balancing: to reduce latency, increase throughput, done dynamically
- ❖ Real-time scheduling: difficult without global view, network delays make task harder
- ❖ Performance modeling and analysis: Network latency to access resources must be reduced
- ❖ Synchronization/coordination mechanisms
- ❖ Group communication, multicast, and ordered message delivery
- ❖ Monitoring distributed events and predicates
- ❖ Distributed program design and verification tool
- ❖ Time and global state
- ❖ Debugging distributed programs
- ❖ Data replication, consistency models, and caching
- ❖ World Wide Web design: caching, searching, scheduling
- ❖ Distributed shared memory abstraction
- ❖ Reliable and fault-tolerant distributed systems

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 $T_s = 10$ time units be the sequential time of execution in a distributed system and let $T_p = 8$ 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

Major references



- ❖ Ajay D. Kshemkalyani, and Mukesh Singhal, Chapter 1, “Distributed Computing: Principles, Algorithms, and Systems”, Cambridge University Press, 2008.
- ❖ <http://www.ois.com/Products/what-is-corba.html>
- ❖ <http://wiki.c2.com/?PublishSubscribeModel>
- ❖ <https://www.slideshare.net/ishraqabd/publish-subscribe-model-overview-13368808/5>
- ❖ <https://www.cloudamqp.com/blog/2014-12-03-what-is-message-queuing.html>
- ❖ https://en.wikipedia.org/wiki/Message_queue