RAMAIAH
Institute of Technology

CSE20-Distributed Systems

Bheemapa H

UNIT-1

Overview

- **Introduction**
- **Relation to computer system components**
- **Motivation**
- **Relation to parallel multiprocessor/multicomputer systems**
- **Message-passing systems versus shared memory systems**

Introduction

Definition:

“A distributed system is a collection of independent entities that cooperate to solve a problem that cannot be individually solved”

- Autonomous processors communicating over a communication network
- Some characteristics

  - ✓ No common physical clock
  - ✓ No shared memory
  - ✓ Geographical separation
  - ✓ Autonomy and heterogeneity.

Distributed System Model

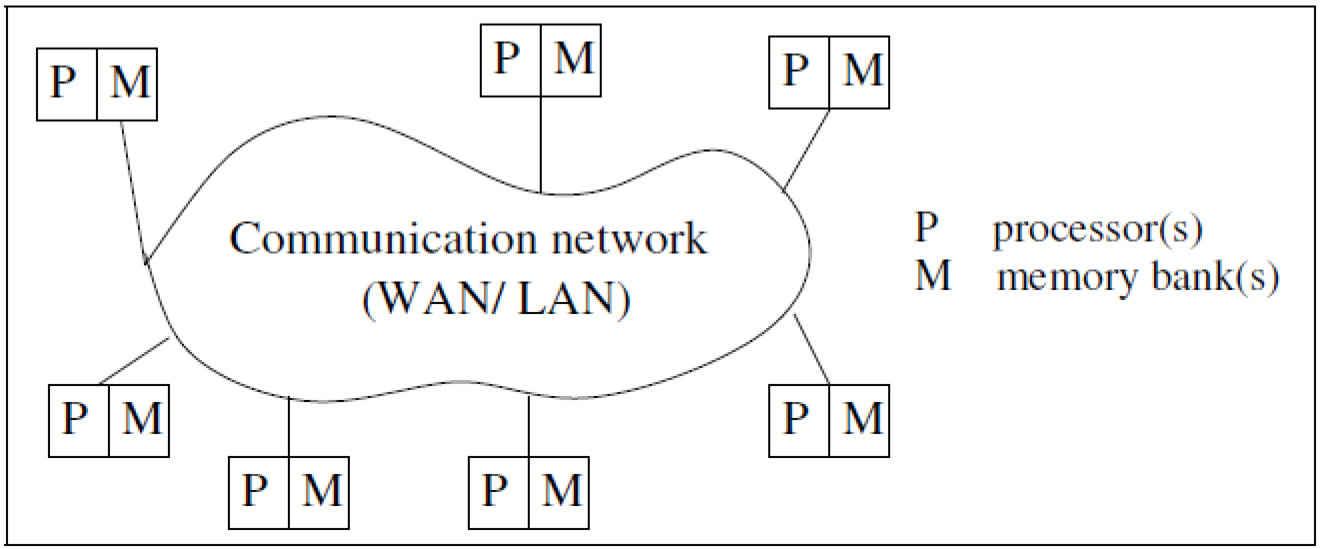


Figure 1.1: A distributed system connects processors by a communication network.

Relation between Software Components

- The distributed system uses a layered architecture to break down the complexity of system design.

A ***distributed execution(****computation)* is the execution of processes across the distributed system to collaboratively achieve a common goal.

- ✓ **The remote procedure call (RPC) mechanism.**
- ✓ Middleware such as CORBA, DCOM (distributed component object model), Java, and RMI(remote method invocation) technologies and the message-passing interface (MPI).

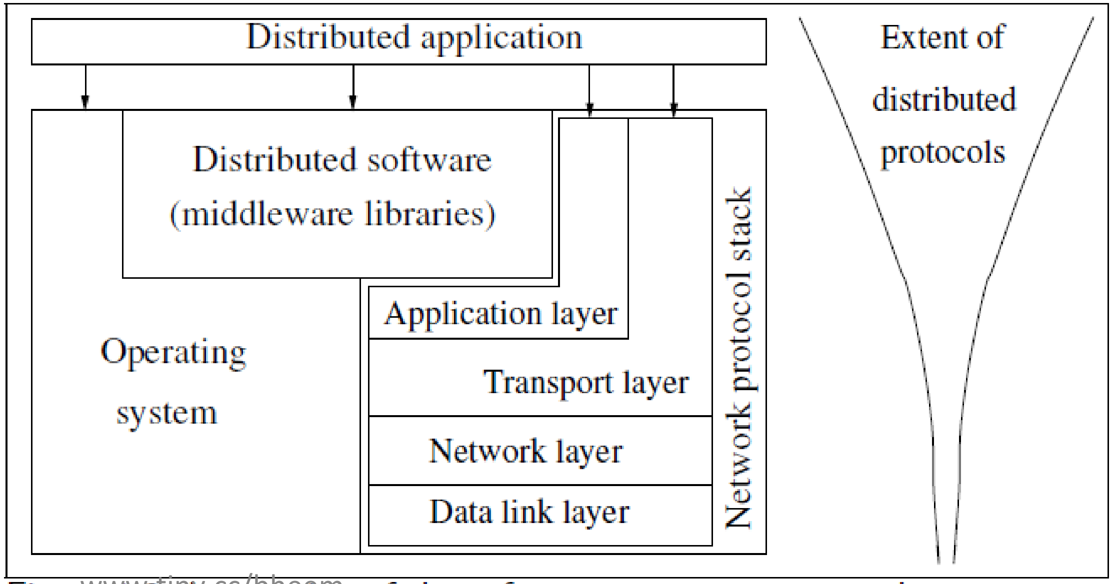


Figure 1.2: Interaction of the software components at each process

Motivation for Distributed System

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

• Characteristics of parallel systems

- Multiprocessor systems (direct access to shared memory, UMA model)
    - Interconnection network - bus, multi-stage switch
    - E.g., Omega, Buttery, 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

- In UMA, where Single memory controller is used- Single, Multiple and Crossbar.
- In NUMA, where different memory controller is used- Tree and hierarchical.

UMA vs. NUMA Models

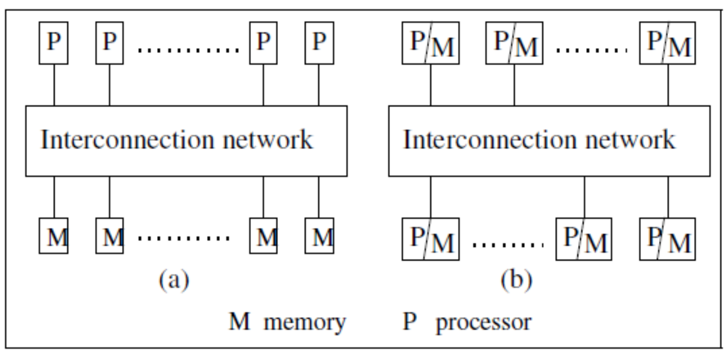


Figure 1.3: Two standard architectures for parallel systems. (a) Uniform memory access (UMA) multiprocessor system. (b) Non-uniform memory access (NUMA) multiprocessor. In both architectures, the processors may locally cache data from memory.

Omega Network

- n processors, n memory banks
- $log\ n$ stages: with $n/2$ switches of size 2x2 in each stage
- Interconnection function: Output i of a stage connected to input j of next stage:

$$j = \begin{cases} 2i & \text{for } 0 \leq i \leq n/2 - 1 \\ 2i + 1 - n & \text{for } n/2 \leq i \leq n - 1 \end{cases}$$

- Routing function: in any stage s at any switch:
  to route to dest. j,
  if $s + 1$th MSB of $j = 0$ then route on upper wire
  else [$s + 1$th MSB of $j = 1$] then route on lower wire

n= 8

1 --- $\log_2 8 = 3$ (the logarithm of 8 to base 2 is equal to 3, because $2^3 = 8$
2 – n/2= 8/2= 4 switches

Interconnection Topologies for Multiprocesors

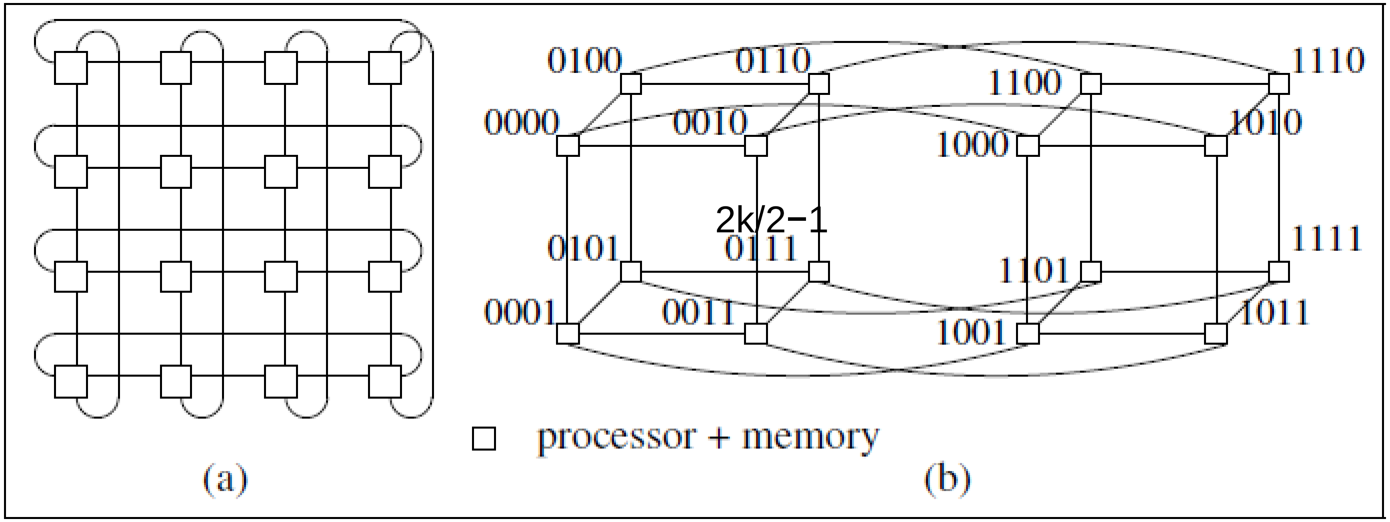


Figure 1.5: (a) 2-D Mesh with wraparound (a.k.a. torus) (b) 3-D hypercube

Flynn's Taxonomy

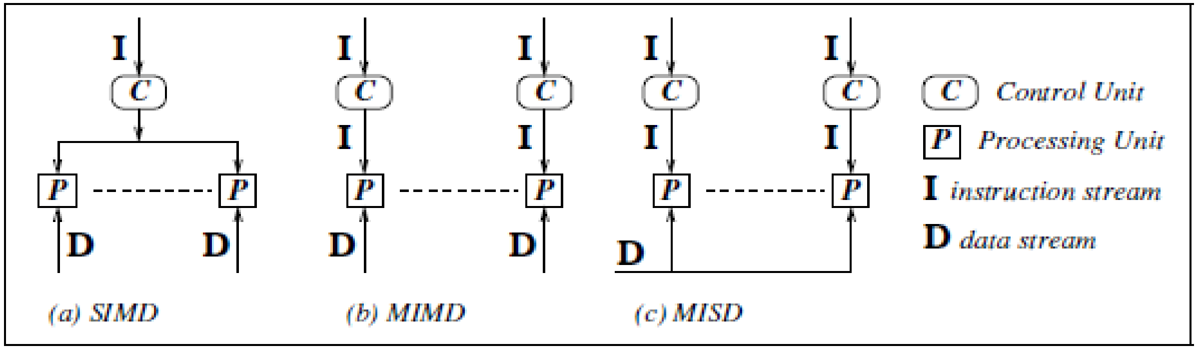


Figure 1.6: SIMD, MISD, and MIMD modes.

- SISD: Single Instruction Stream Single Data Stream (traditional)
- SIMD: Single Instruction Stream Multiple Data Stream
  - scientific applicaitons, applications on large arrays
  - vector processors, systolic arrays, Pentium/SSE, DSP chips
- MISD: Multiple Instruciton Stream Single Data Stream
  - E.g., visualization
- MIMD: Multiple Instruction Stream Multiple Data Stream
  - distributed systems, vast majority of parallel systems

Terminology

- Coupling
  - I Interdependency/binding among modules, whether hardware or software (e.g., OS, middleware)
- Parallelism: T(1)=T(n).
  - I Function of program and system
- Concurrency of a program
  - Measures productive CPU time vs. waiting for synchronization operations
- Granularity of a program
  - Amt. of computation vs. amt. of communication
  - Fine-grained program suited for tightly-coupled system

Message-passing vs. Shared Memory

- 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

Primitives for distributed communication

- 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-speced buer
- 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

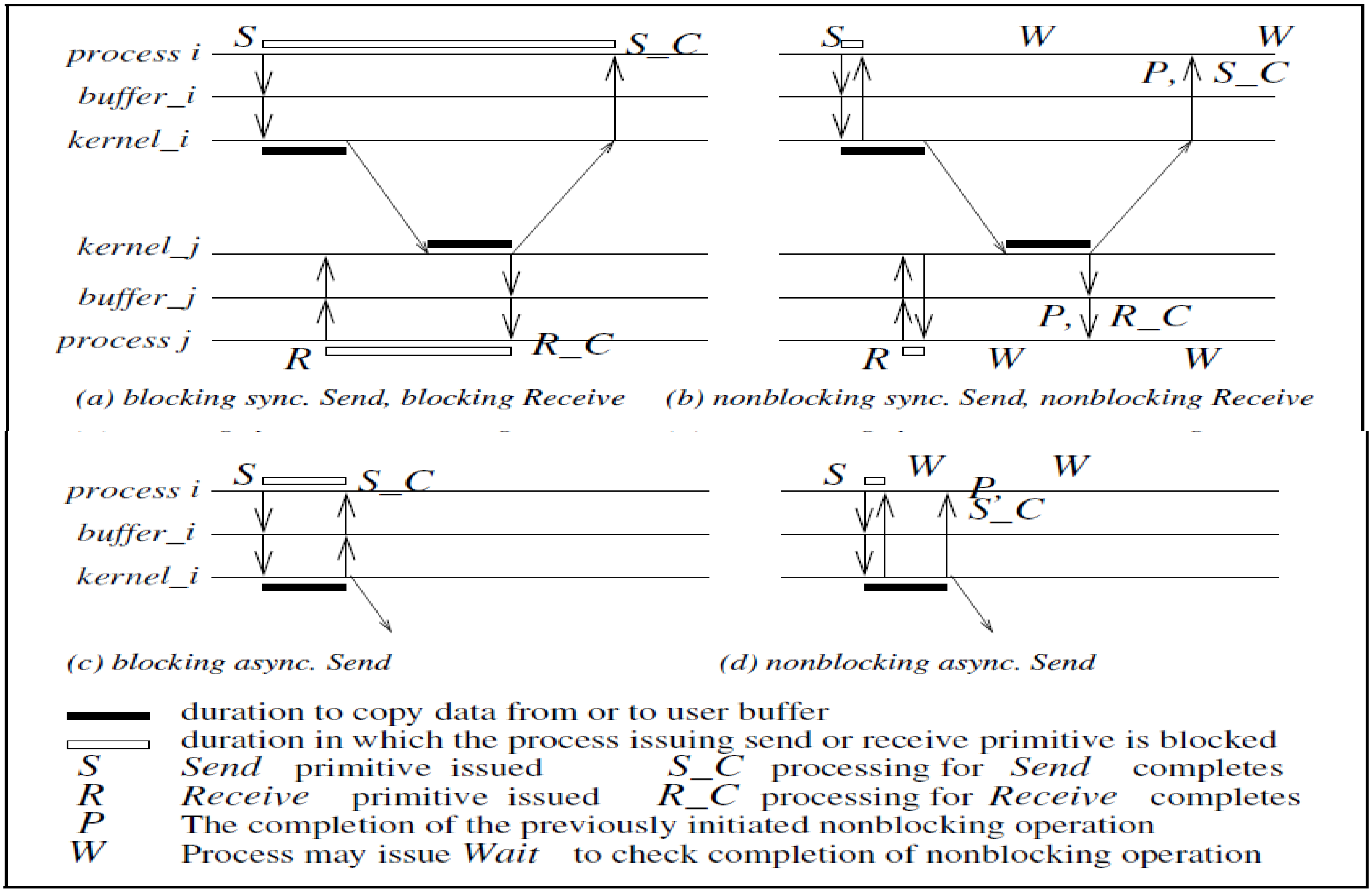
Non-blocking Primitive

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

Figure 1.7: A nonblocking *send* primitive. 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

Illustration of 4 send and 2 receive primitives



Omega, Butterfly Interconnects

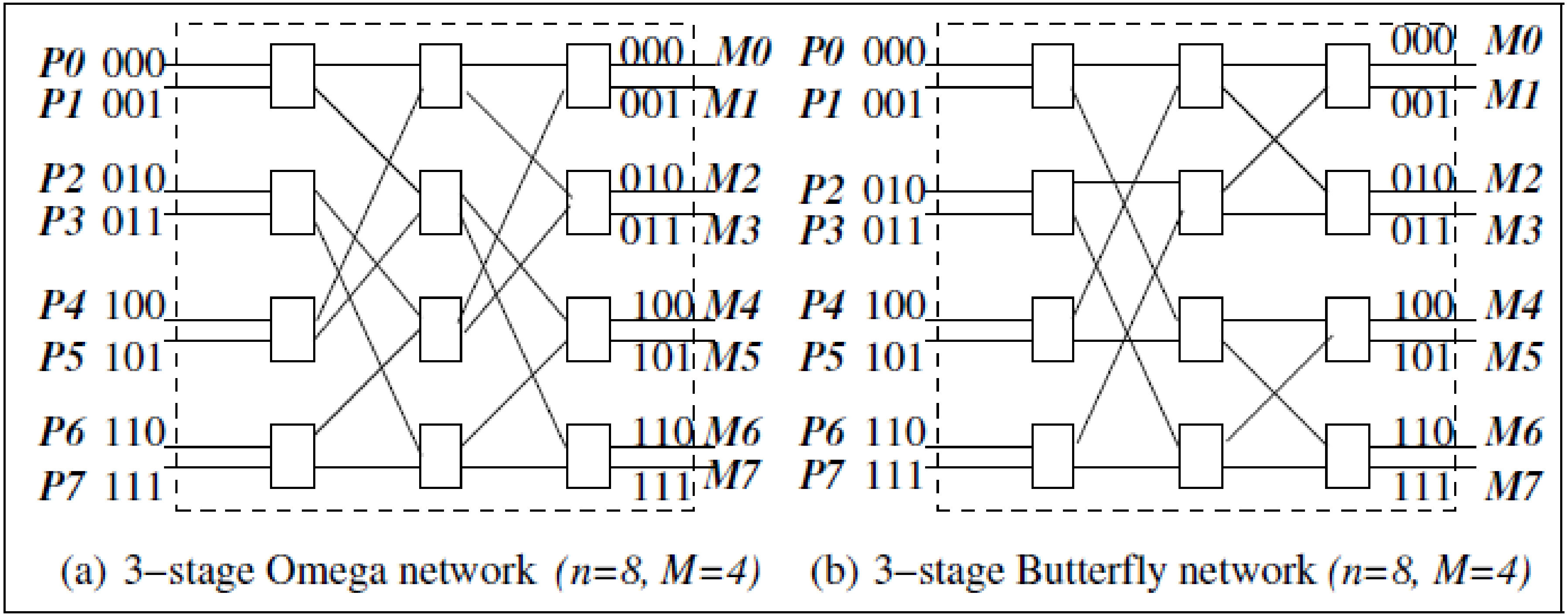


Figure 1.4: Interconnection networks for shared memory multiprocessor system
(a) Omega network (b) Butterfly network.