Operating System

Chapter 8: Distributed Operating Systems

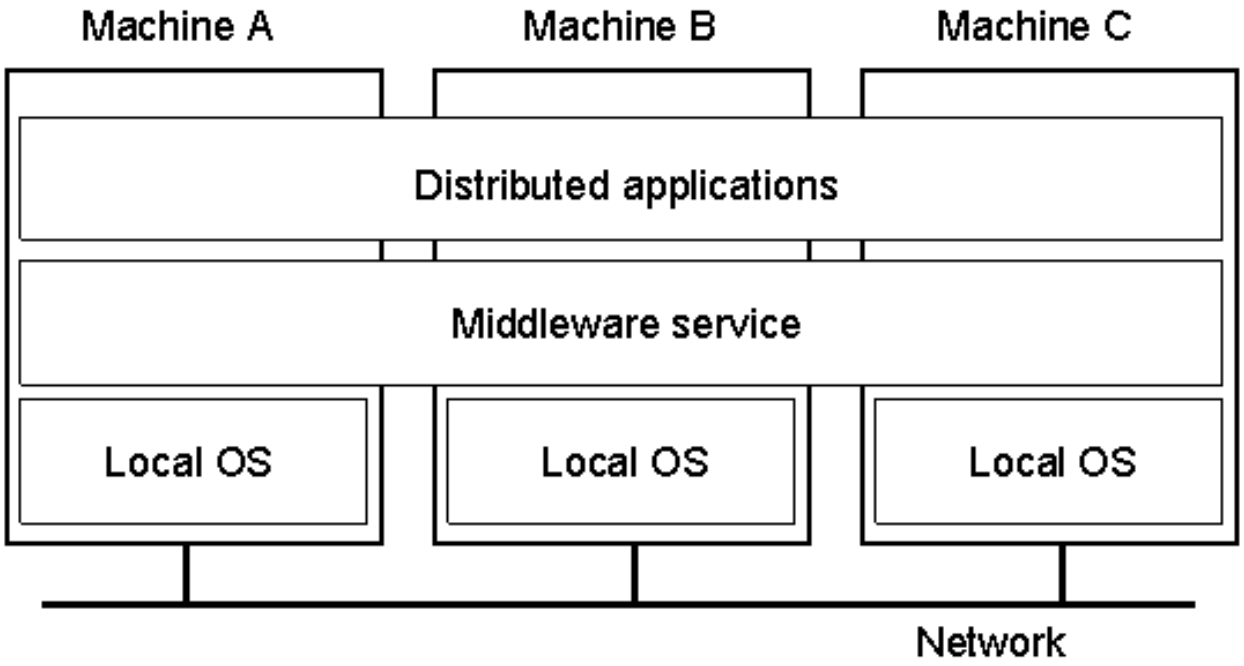
Prepared By:
Amit K. Shrivastava
Asst. Professor
Nepal College Of Information Technology

Introduction to Distributed Systems

- Why do we develop distributed systems?
  - availability of powerful yet cheap microprocessors (PCs, workstations), continuing advances in communication technology,
- **What is a distributed system?**
  - A distributed system is a collection of independent computers that appear to the users of the system as a single system.
- Examples:
  - Network of workstations
  - Distributed manufacturing system (e.g., automated assembly line)

  Network of branch office computers

An Abstract View Distributed Systems



A distributed system organized as middleware.
Note that the middleware layer extends over multiple machines.

Advantages of Distributed Systems Over Centralized Systems

- **Economics:** a collection of microprocessors offer a better price/performance than mainframes. Low price/performance ratio: cost effective way to increase computing power.
- **Speed:** a distributed system may have more total computing power than a mainframe. Ex. 10,000 CPU chips, each running at 50 MIPS. Not possible to build 500,000 MIPS single processor since it would require 0.002 nsec instruction cycle. Enhanced performance through load distributing.
- **Inherent distribution:** Some applications are inherently distributed. Ex. a supermarket chain.
- **Reliability:** If one machine crashes, the system as a whole can still survive. Higher availability and improved reliability.
- **Incremental growth:** Computing power can be added in small increments. Modular expandability
- **Another deriving force:** the existence of large number of personal computers, the need for people to collaborate and share information.

Advantages of Distributed Systems over Independent PCs

- Data sharing: allow many users to access to a common data base
- Resource Sharing: expensive peripherals like color printers
- Communication: enhance human-to-human communication, e.g., email, chat
- Flexibility: spread the workload over the available machines

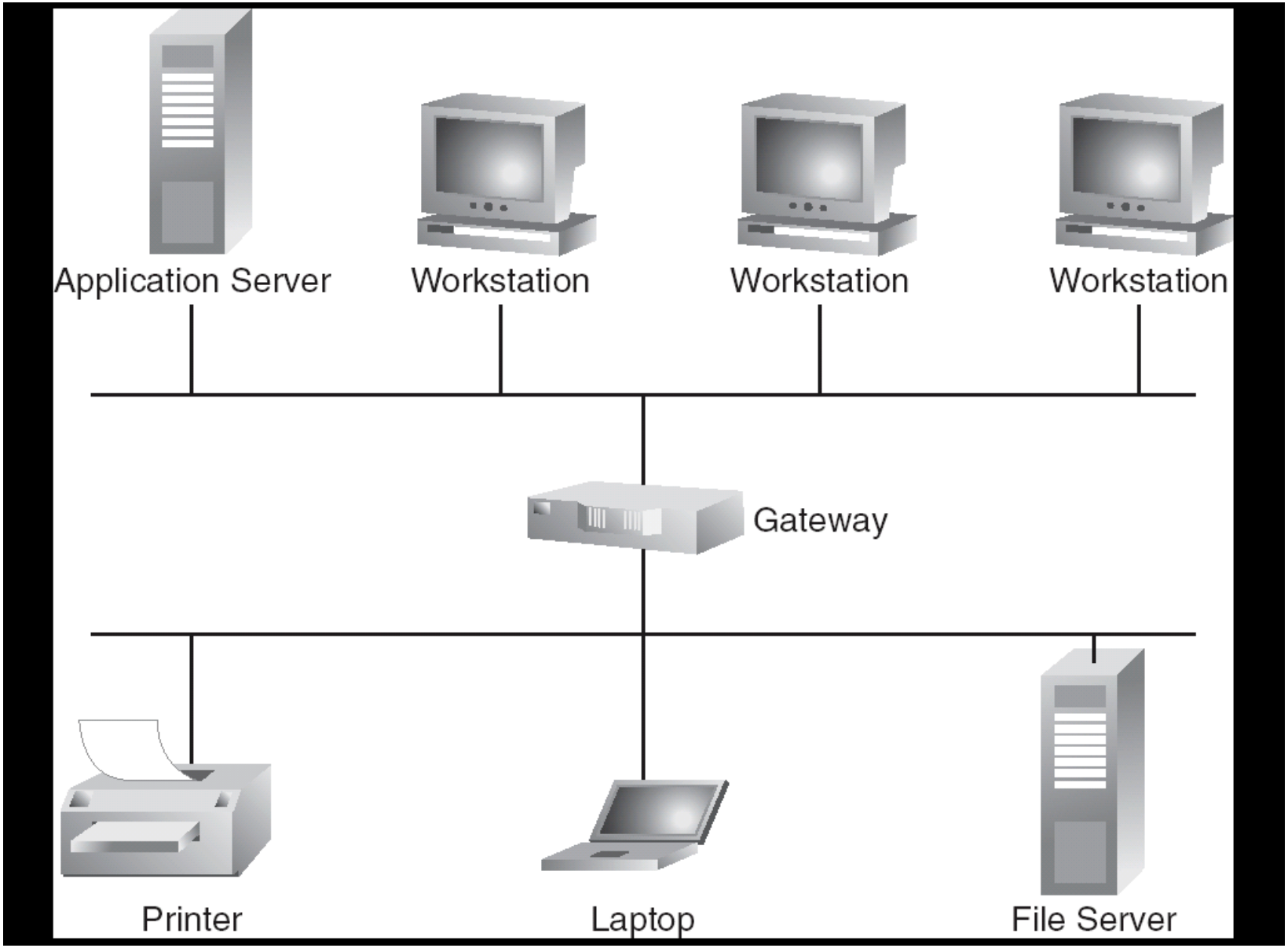
Disadvantages of Distributed Systems

- Software: difficult to develop software for distributed systems
- Network: saturation, lossy transmissions
- Security: easy access also applies to secrete data.

Network Architecture

- Local-Area Network (LAN) – designed to cover small geographical area.
  - Multiaccess bus, ring, or star network
  - Speed ≈ 10 – 100 megabits/second upto 40 mbps
  - Broadcast is fast and cheap
  - Nodes:
    - ✓ usually workstations and/or personal computers
    - ✓ a few (usually one or two) mainframes

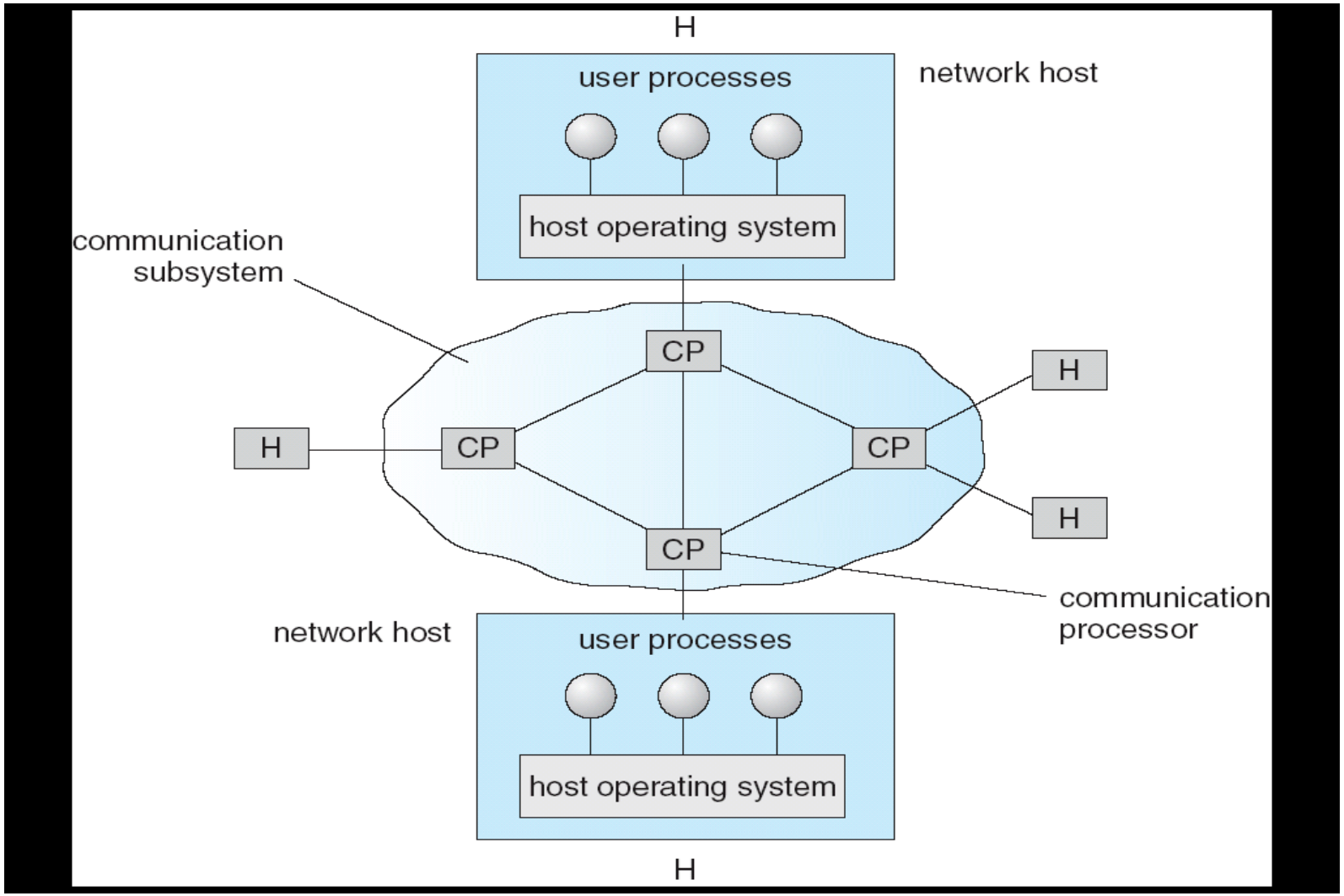
Depiction of typical LAN



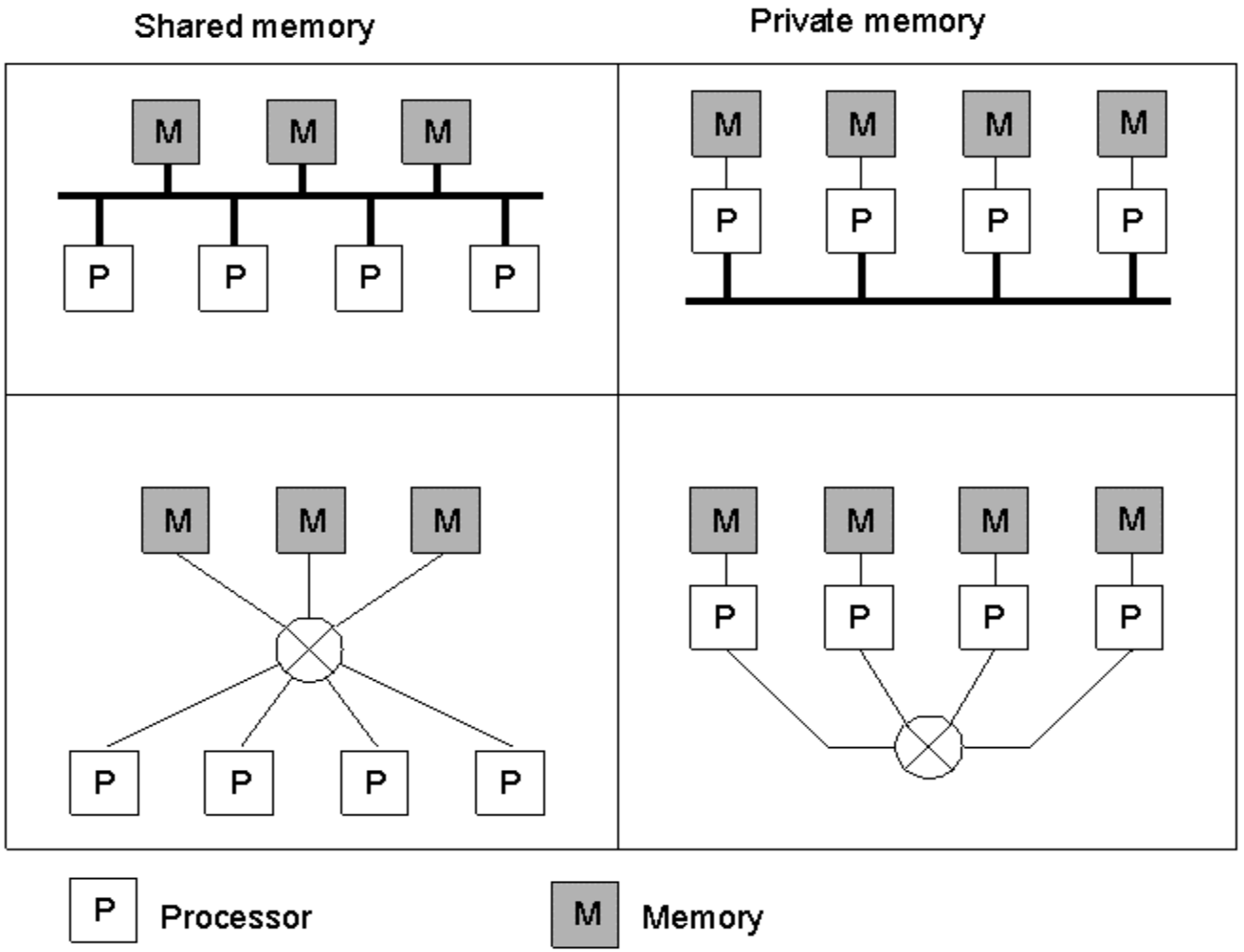
Network Architecture(contd..)

- Wide-Area Network (WAN) – links geographically separated sites
  - Point-to-point connections over long-haul lines (often leased from a phone company)
  - Speed ≈ 1.544 – 45 megbits/second
  - Broadcast usually requires multiple messages
  - Nodes:
    - usually a high percentage of mainframes

Communication Processors in a Wide-Area Network



Hardware Concepts

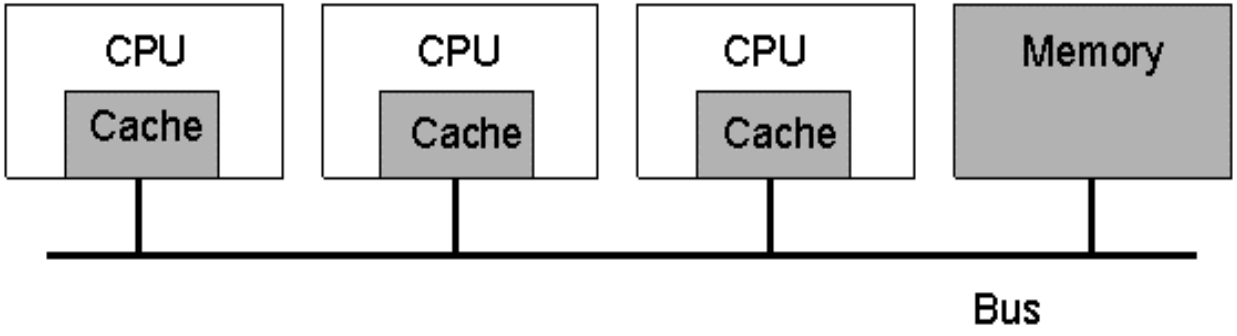


individual wires from machines to machine, with many wiring patterns

Different basic organizations and memories in distributed computer systems: Multiprocessors Vs Multicomputers

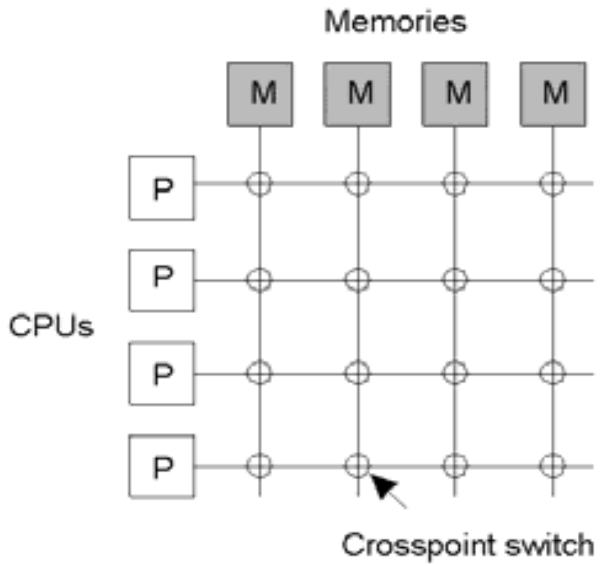
Multiprocessors (1)

A bus-based multiprocessor.

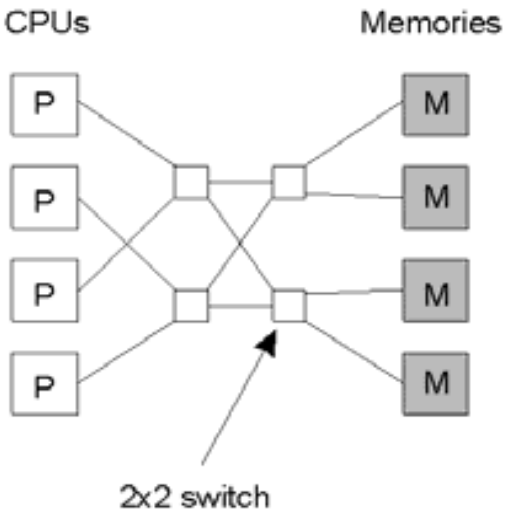


Multiprocessors (2)

a) A crossbar switch
b) An omega switching network



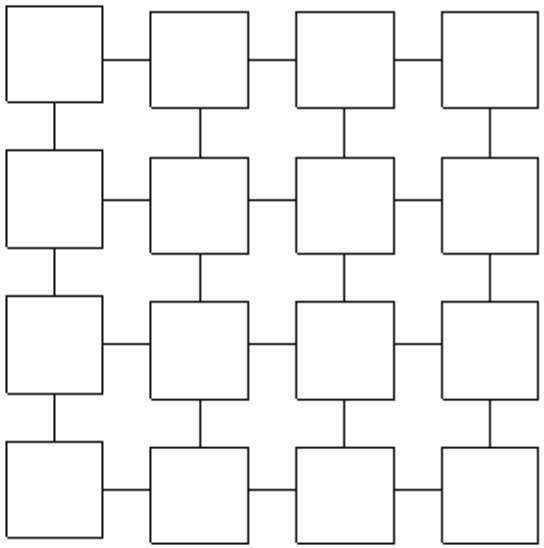
(a)



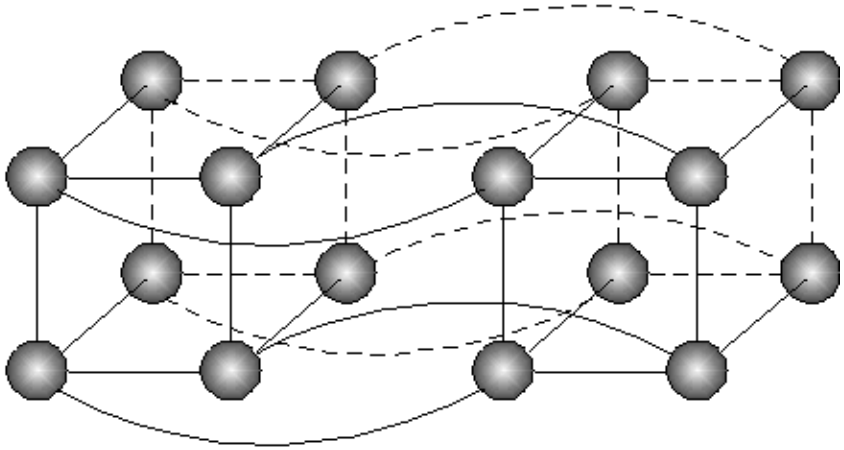
(b)

Homogeneous Multicomputer Systems

a) Grid
b) Hypercube



(a)



(b)

Tightly coupled vs. loosely coupled

Heterogeneous Multicomputer Systems

- Most distributed systems today are built on top of heterogeneous multicomputers and interconnection networks
- No global system view
- Sophisticated software needed to support distributed applications

Software Concepts

- **Software more important for users**
- **Two types:**
    1. **Network Operating Systems**
    2. **Distributed Operating Systems**

Software Concepts(contd..)

Network Operating Systems

- loosely-coupled software on loosely-coupled hardware
- A network of workstations connected by LAN
- each machine has a high degree of autonomy
  - rlogin machine
  - rcp machine1:file1 machine2:file2
- Files servers: client and server model
- Clients mount directories on file servers
- Best known network OS:
  - Sun's NFS (network file servers) for shared file systems
- a few system-wide requirements: format and meaning of all the messages exchanged

Software Concepts(contd..)

Distributed Operating Systems

- tightly-coupled software on loosely-coupled hardware
- provide a single-system image or a virtual uniprocessor
- a single, global interprocess communication mechanism, process management, file system; the same system call interface everywhere
- Ideal definition:

  “ A distributed system runs on a collection of computers that do not have shared memory, yet looks like a single computer to its users.”

Communication in Distributed Systems

OSI : APSTNDP

- The most important issue for any distributed system is the communication modelOne (older) communication model is the ISO OSI Reference Model. This model structures communication in seven layers. Layer 1 is the physical layer, which is concerned with moving bits. Layer 2, the data link later, breaks the bit stream into a frame stream for error control and flow control. Layer 3, the network layer, handles routing. Layer 4, the transport layer, allows reliable end-to-end connections. Layer 5, the session layer, groups messages into activities for recovery purposes. Layer 6, the presentation layer, defines abstract data structures for use by applications. Finally, layer 7, the application layer, contains file transfer, job entry, virtual terminal, and various other standard protocols. The ISO OSI model, once of great interest to some, is rapidly losing out to the TCP/IP model.

Communication in Distributed Systems(contd.)

- The TCP model really has two layers. The IP layer moves packets from source to destination over an arbitrary collection of networks. On top of this is the TCP layer, which provides end-to-end connections. All the applications go above this. What happens below the IP layer is a bit vague, but there must be a data link layer and a physical layer of some kind, only these are not defined by the model. The TCP/IP protocol is the glue holding the Internet together.

ATM(Asynchronous Transfer Model)

- ATM can be used across both wide area and local area networks. The goal is to support a wide range of communication requirements such as those needed for voice and video as well as data transmission. ATM supports point-to-point communication via one or more switches. Typically, each computer in the network is connected to a switch via two optical fibres: one taking traffic to the switch, the other relaying traffic from the switch.

ATM(contd..)

- All data to be transmitted is divided into fixed-sized packets call **cells**, where each cell has a 5-byte header and a 48-byte data field. Applications communicate via **virtual channels (VC).** The data to be transmitted on a particular VC can have certain timing behaviour associated with it, such as its bit rate, period or its deadline. An **adaptation layer** provides specific services to support the particular class of user data; the precise behaviour of this layer is variable to suit the data transmission needs of a particular system. It is within the adaption layer that end-to-end error correction and synchronization, and the segmentation and reassembly of user data into the ATM cells are performed.

RPC(Remote Procedure Call)

- The overriding goal behind the remote procedure call paradigm is to make distributed communication as simply as possible. Typically, RPCs are used for communication between programs written in the same language, for example, Ada or Java. A procedure (server) is identified as being one that can be called remotely. From the server specification, it is possible to generate automatically two further procedures: a **client stub** and a **server stub.** The client stub is used in place of the server at the site on which the remote procedure call originates. The server stub is used on the same site as the server procedure. The purpose of these two procedures is to provide the link between the client and the server in a transparent way (and thereby meet all the requirements laid out in the previous section). The stubs are sometimes called **middleware** as they sit between the application and the operating system**.**

Group Communication

- The remote procedure call (or remote method call) is a common form of communication between clients and servers in a distributed systems. However, it does restrict communication to be between two processes. Often, when groups of processes are interacting (for example, performing an atomic action), it is necessary for communication to be sent to the whole group. A **multicast** communication paradigm provides such a facility. Some networks, for example Ethernet, provide a hardware multicast mechanism as part of their data link layer. If this is not the case, then further software protocols must be added.
- All networks and processors are, to a greater or lesser extent, unreliable. It is therefore possible to design a family of group communication protocols, each of which provides a multicast communication facility with specific guarantees:

Group Communication(contd..)

- **unreliable multicast –** no guarantees of delivery to the group is provided; the multicast protocol provides the equivalent of a datagram-level of service;
- **reliable multicast** – the protocol makes a best-effort attempt to deliver the message to the group, but offers no absolute guarantee of delivery;
- **atomic multicast** – the protocol guarantees that if one process in the group receives the message then all members of the group receive the message; hence the message is delivered to all of the group or none of them;
- **ordered atomic multicast –** as well as guaranteeing the atomicity of the multicast, the protocol also guarantees that all members of the group will receive messages from different senders in the same order.
- The more guarantees the protocols give, the greater the cost of their implementation.

Client-Server Model

- *Coordination model in a distributed system.*
- It defines:
  - Which process may begin the interaction
  - Which process may answer
  - How error conditions may be managed.
- Although an Internet system provides a basic communication service, the protocol software cannot initiate control with, or accept contact from, a remote computer.
- Most network applications use a form of communication known as **the client –server paradigm.** A server application waits passively for contact, while a client application initiates communication actively.

Client-Server Model(contd..)

Process classification

- **Client process,** the process that requires a service
- **Server process,** the process that provides the required service
- The client requires a service, the server provides the service and makes available the results to the client

Client-Server Model(contd..)

Client functions

In generally, client software:

- is an arbitrary application program that becames a client temporarily when remote access is needed, but also performs other computation locally.
- is invoked locally by a user, and executes only for one session
- runs locally on a user personal computer
- actively initiates contact with a server
- can access multiple services as needed, but actively contacts one remote server at a time.
- does not require special hardware or a sophisticated operating system

Client-Server Model(contd..)

Server functions

- Is a special purpose, privileged program dedicated to providing one service, but can handle multiple remote clients at the same time.
- Run on a shared computer(i.e. not a user’s personal computer).
- Wait passively for contact from arbitrary remote clients
- Accepts contact from arbitrary clients, but offers a single service
- Requires powerful hardware and a sophisticated operating system

Clock Synchronization

- Synchronizing clocks in a distributed system is complicated due to the fact that each local clock runs at a slightly different rate. Furthermore, message transport takes a finite, variable, and unknown amount of time. As a consequence, it is never possible to have all the clocks in the distributed system record the same time, at least in the absence of an external way to achieve synchronization.

Clock Synchronization(contd..)

- One approach to synchronizing clocks, logical clocks, was devised by Leslie Lamport (1978). This scheme does not maintain absolute times, but does maintain relative ordering. It makes it possible to tell which of two events happened first. In it, each machine has a clock and puts its current time in every message. When a message arrives at a machine whose clock has a lower value than that contained in the message, the receiver’s clock is fast forwarded to a value one tick more than the value contained in the message. In this way, any two messages that are casually related bear different numbers (time stamps), with the first one having the lower number. Events that are not causally related do not exhibit the "happens before" property.

Processes and Processors in Distributed System

- **Basic idea**: we build virtual processors in software, on top of physical processors:
- **Processor**: Provides a set of instructions along with the capability of automatically executing a series of those instructions.
- **Thread:** A minimal software processor in whose context a series of instructions can be executed. Saving a thread context implies stopping the current execution and saving all the data needed to continue the execution at a later stage.
- **Process:** A software processor in whose context one or more threads may be executed. Executing a thread, means executing a series of instructions in the context of that thread.

Context Switching (1/2)

- **Processor context:** The minimal collection of values stored in the registers of a processor used for the execution of a series of instructions (e.g., stack pointer, addressing registers, program counter).
- **Thread context:** The minimal collection of values stored in registers and memory, used for the execution of a series of instructions (i.e., processor context, state).
- **Process context:** The minimal collection of values stored in registers and memory, used for the execution of a thread (i.e., thread context, but now also at least MMU register values).

Context Switching (2/2)

- **Observation 1:** Threads share the same address space. Thread context switching can be done entirely independent of the operating system.
- **Observation 2:** Process switching is generally more expensive as it involves getting the OS in the loop, i.e., trapping to the kernel.
- **Observation 3:** Creating and destroying threads is much cheaper than doing so for processes.

Scheduling in Distributed System

- In general, job scheduling is composed of at least two inter-dependent steps: the allocation of processes to workstations (space-sharing) and the scheduling of the processes over time (time-sharing), while there exist several optional complementary steps to further improve the performance.
- When a job is submitted to the system, job placement will be done, i.e., to decide which workstations to run the job cooperatively (space-sharing). Along with the job submission, a description of the attributes of the job is also submitted to the system in order to specify the resource requirement, such as memory size requirement, expected CPU time, deadline time, etc. In the meantime, the system always maintains an information table, either distributed or centralized, to record the current resource status of each workstation, e.g., CPU load, free memory size, etc. Then, a matchmaking frame will do matching work to find the most suitable set of workstations to meet the requirement of the job.

Scheduling in Distributed System(contd..)

- *Local scheduling, where each workstation independently schedules its processes, is an attractive time-sharing option* for its ease of construction, scalability, fault-tolerance, etc. Meanwhile, coordinated scheduling of parallel jobs across the nodes of a multiprocessor (*coscheduling) is also indispensable in a distributed system. Without coscheduling, the* processes constituting a parallel job might suffer high communication latencies because of *processor thrashing. By* coordinated scheduling across cooperating processes, each local scheduler is able to make independent decisions that tend to schedule the processes of a parallel application in a coordinated manner across processors, in order to fully exploit the computing resource of a distributed system.

Properties of a Good Scheduler

- *General purpose: a scheduling approach should make few assumptions about and have few restrictions to the types* of applications that can be executed. Interactive jobs, distributed and parallel applications, as well as non-interactive batch jobs, should all be supported with good performance.
- *Efficiency: it has two meanings: one is that it should improve the performance of scheduled jobs as much as* possible; the other is that the scheduling should incur reasonably low overhead so that it won't counter attack the benefits.
- *Fairness: sharing resources among users raises new challenges in guaranteeing that each user obtains his/her* fair share when demand is heavy. In a distributed system, this problem could be exacerbated such that one user consumes the entire system.

Properties of a Good Scheduler(contd..)

- *Dynamic: the algorithms employed to decide where to process a task should respond to load changes, and* exploit the full extent of the of the resources available.
- *Transparency: the behavior and result of a task's execution should not be affected by the host(s) on which it* executes. In particular, there should be no difference between local and remote execution. No user effort should be required in deciding where to execute a task or in initiating remote execution; a user should not even be aware of remote processing, except may be better performance. Further, the applications should not be changed greatly. It is undesirable to have to modify the application programs in order to execute them in the system.