Background of Cloud computing

1. Distributed Systems

What is a distributed system?

Distributed System: a group of independent/autonomous computers that are networked together appear to the user as a one computer.

- Work together to achieve a common goal.
- Clouds can be DS ?

-- History --

- Problems that are larger than what a single machine can handle
- Computer Networks, Message passing were invented to facilitate distributed systems
- ARPANET eventually became Internet

What is a distributed system?

- A collection of independent computers
- A communication facility to pass messages
- No shared memory
- No shared clock
- Each computer has its own operating system

Where they are used?

- **Strategic Systems (Defense/Intelligence)**
- **Bioinformatics (Genome/ Molecular Analysis)**
- **Visualization and Graphics (VR)**
- **Economics and Finance (Fin-Tech)**
- **Scientific Computing (Weather forecasting)**

Advantages and Disadvantages

Advantages

- Communication and resource sharing possible
- Economics – price-performance ratio
- Reliability, scalability
- Potential for incremental growth

Disadvantages

- Distribution-aware PLs, OSs and applications
- Network connectivity essential
- Security and privacy

Design Goals/ Characteristics

- **Distributed Transparency**
  - Location
  - Migration
  - Replication
  - Concurrency

- **Openness**
- **Scalability**
- **Fault-tolerance**
- **High availability**
- **Recoverability**
- **Performance Predictability**
- **Security**

Transparency in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several competitive users
Concurrency	Hide that a resource may be performed by several heterogeneous resources(h/w)
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Parallel vs. Distributed Systems

A concurrent system could be Parallel or Distributed:
Two possible Views to make the distinction

View 1:

- Parallel System: a particular tightly-coupled form of distributed computing
- Distributed System: a loosely-coupled form of parallel computing

View 2:

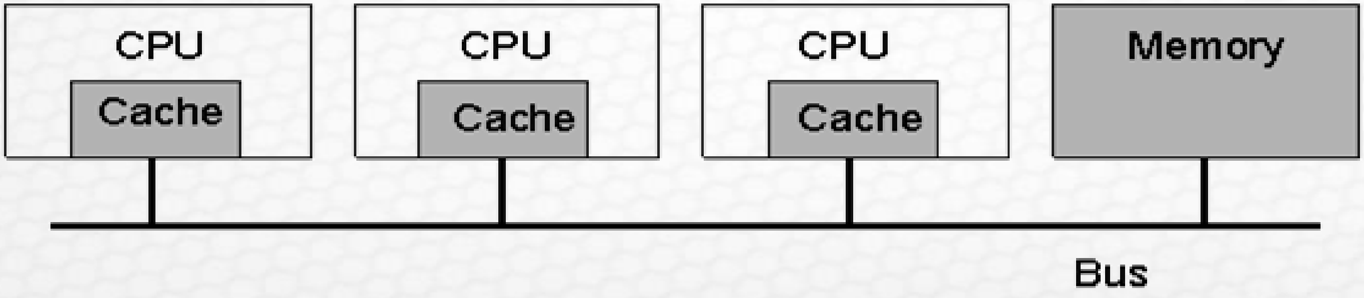
- Parallel System: processors access a shared memory to exchange information
- Distributed System: uses a "distributed memory". Massage passing is used to exchange information between the processors as each one has its own private memory.

Hardware Concepts: Multiprocessors(1/2)

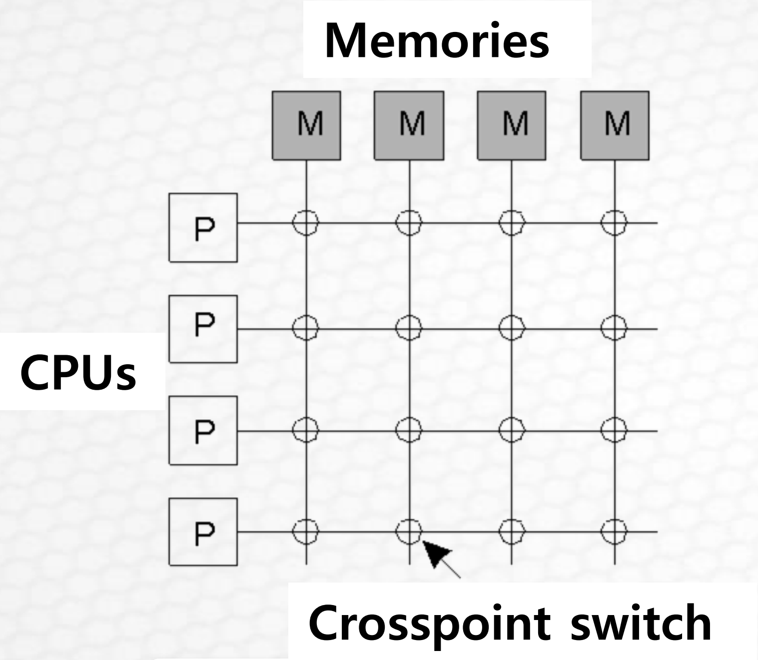
Multiprocessor dimensions

- Memory: could be shared or be private to each CPU
- Interconnect: could be shared (bus-based) or switched

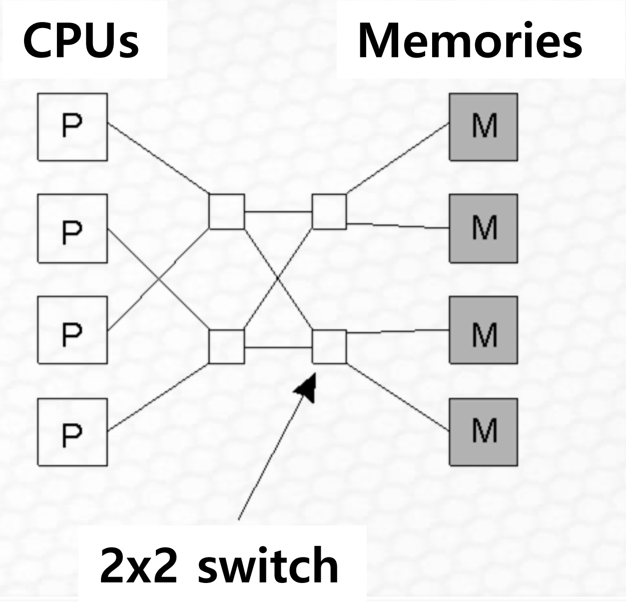
A bus-based multiprocessor



Hardware Concepts: Multiprocessors(2/2)

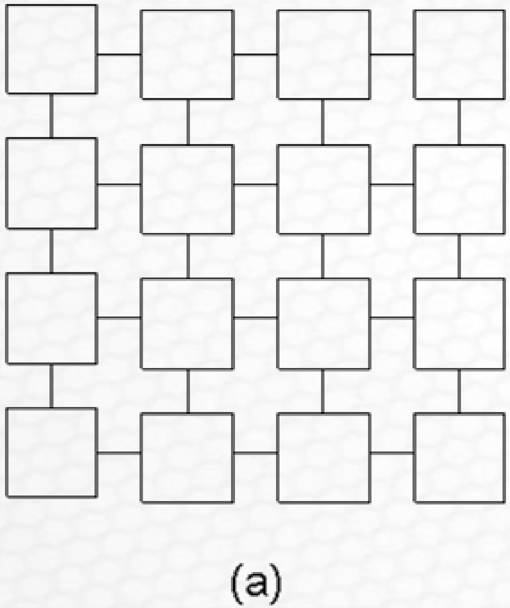


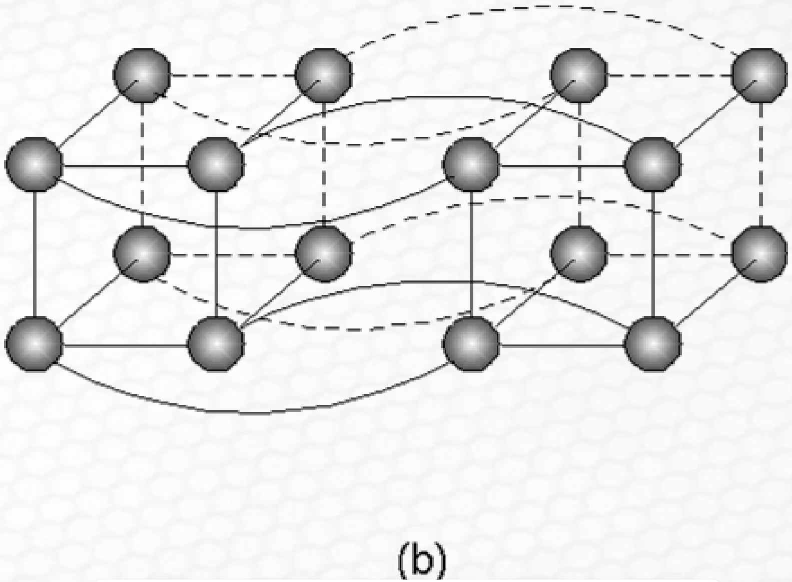
(a) A crossbar switch



(b) An omega switching network

Homogeneous Multicomputer Systems





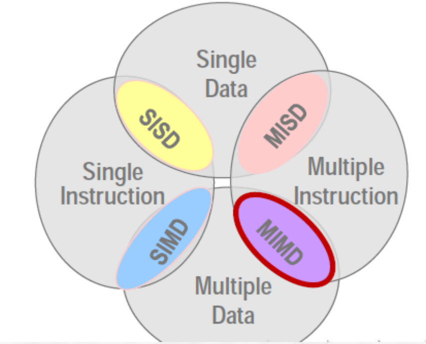
(a) Grid

(b) Hypercube

Hardware Model

- **The four possible combinations: (Flynn's Taxonomy)**
  -SISD: in traditional uniprocessor computers
  -MISD: Multiple concurrent instructions operating on the same data element. Not useful!
  -SIMD: Single instruction operates on multiple data elements in parallel
  -MIMD: Covers parallel & distributed systems and machines that contain multiple computers

- **Distributed systems are MIMD**



Distributed Systems Models

- **Minicomputer model (e.g., early networks)**
  - Each user has local machine
  - Local processing but can fetch remote data (files, databases)
- **Workstation model (e.g., Sprite)**
  - Processing can also migrate
- **Client-server Model (e.g., V system, world wide web)**
  - User has local workstation
  - Powerful workstations serve as servers (file, print, DB servers)
- **Processor pool model (e.g., Amoeba, Plan 9)**
  - Terminals are Xterms or diskless terminals
  - Pool of backend processors handle processing

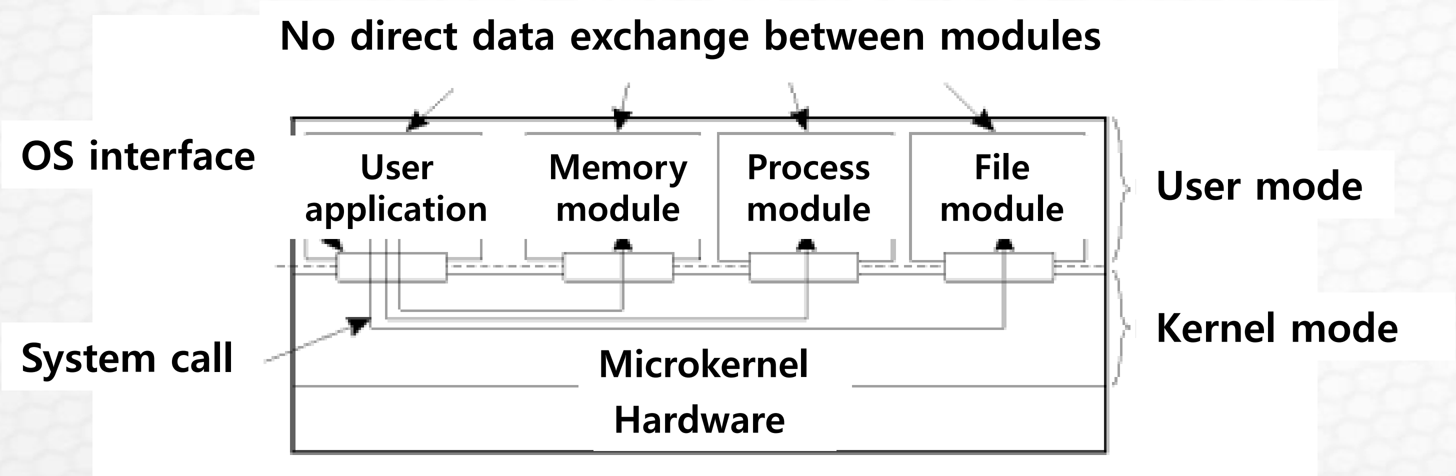
Uniprocessor Operating Systems(1/2)

- An OS acts as a resource manager or an arbitrator
  - Manages CPU, I/O devices, memory
- OS provides a virtual interface that is easier to use than hardware
- Structure of uniprocessor operating systems
  - Monolithic (e.g., MS-DOS, early UNIX)
    - One large kernel that handles everything
  - Layered design
    - Functionality is decomposed into N layers
    - Each layer uses services of layer N-1 and implements new service(s) for layer N+1

Uniprocessor Operating Systems(2/2)

Microkernel architecture

- Small kernel
- user-level servers implement additional functionality



Distributed Operating System

- **Manages resources in a distributed system**
  - **Seamlessly and transparently to the user**
- **Looks to the user like a centralized OS**
  - **But operates on multiple independent CPUs**
- **Provides transparency**
  - **Location, migration, concurrency, replication,…**
- **Presents users with a virtual uniprocessor**

Types of Distributed OSs

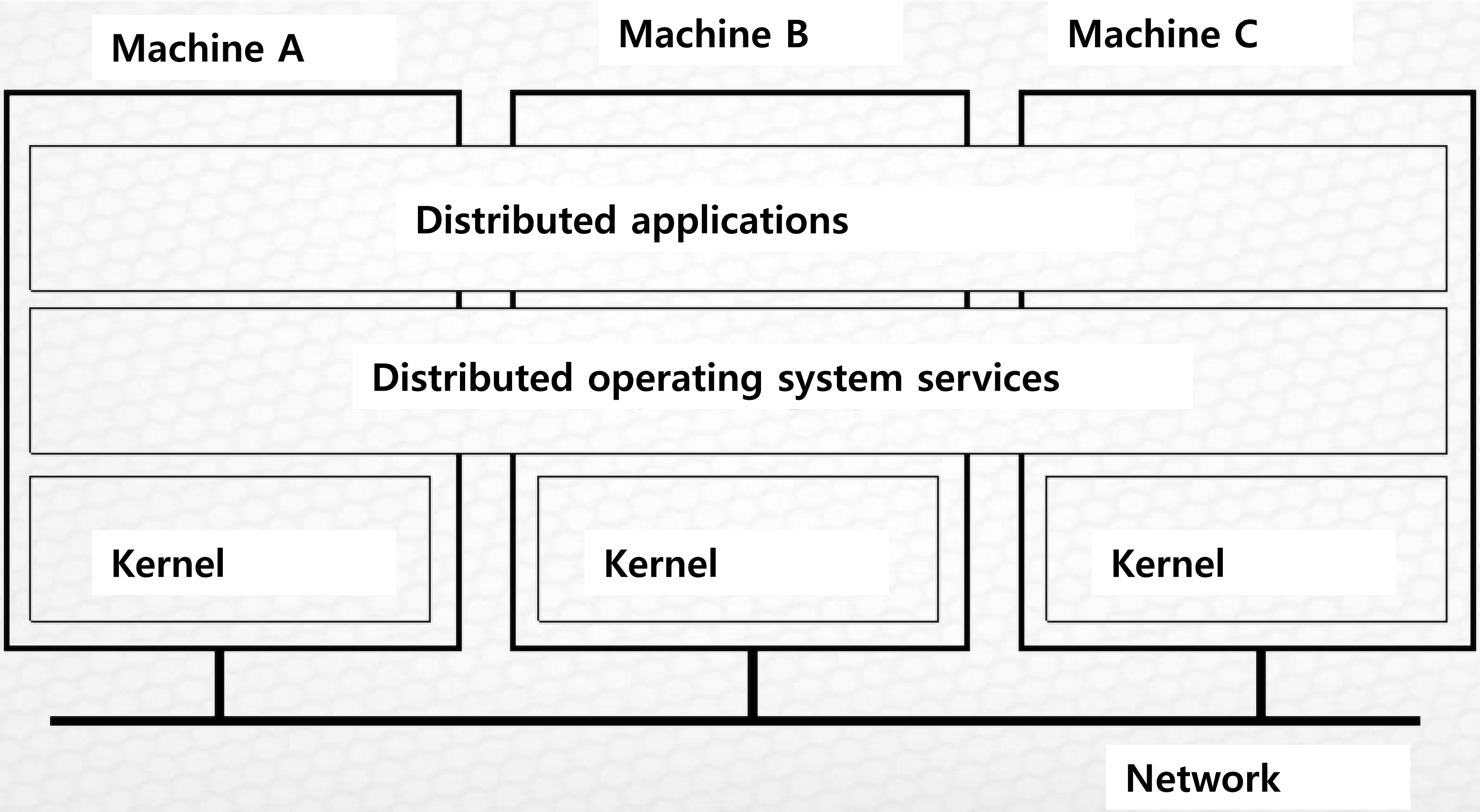
System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

Multiprocessor Operating Systems (1/2)

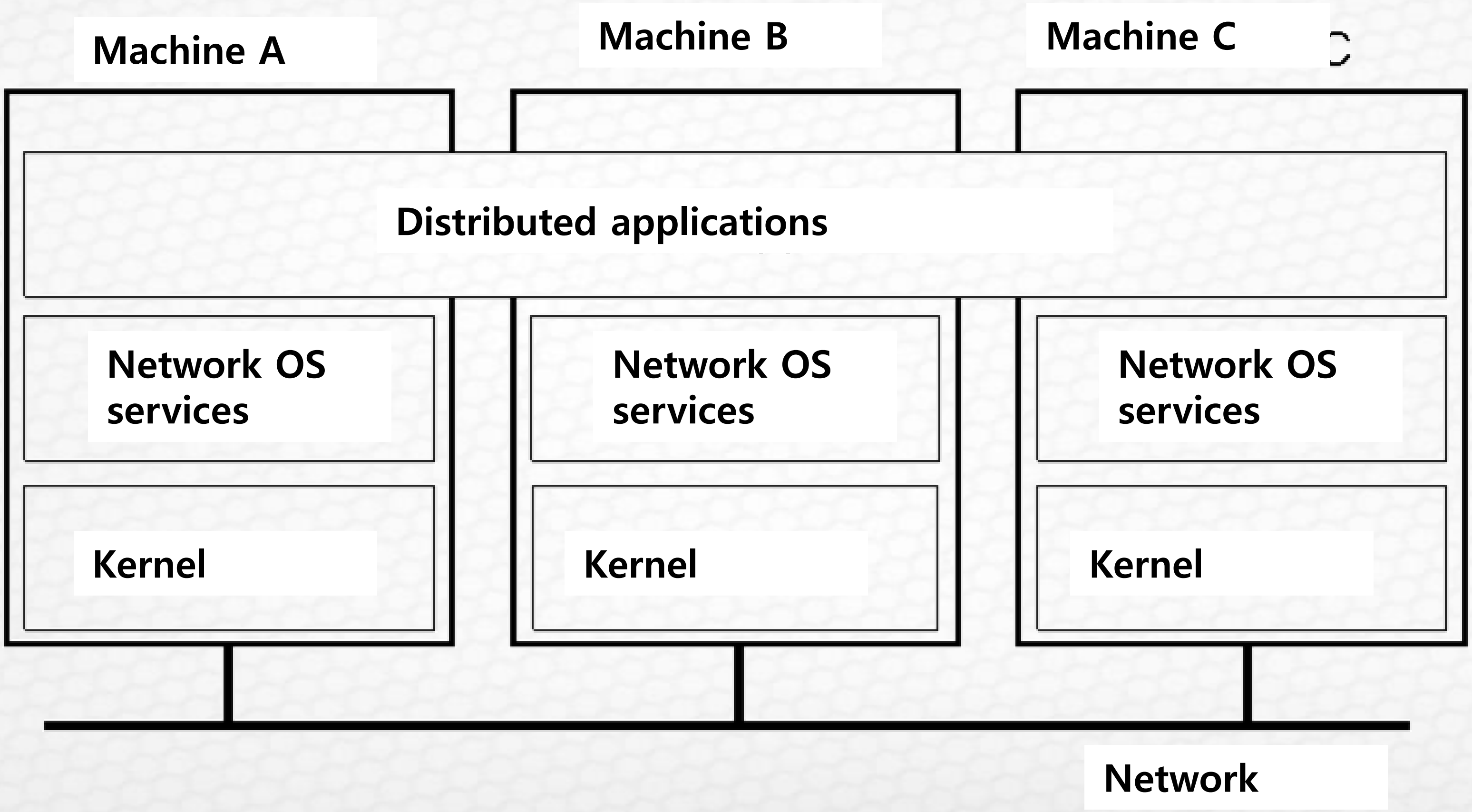
- Like a uniprocessor operating system
- Manages multiple CPUs transparently to the user
- Each processor has its own hardware cache

→ Maintain consistency of cached data

Multicomputer Operating Systems (2/2)



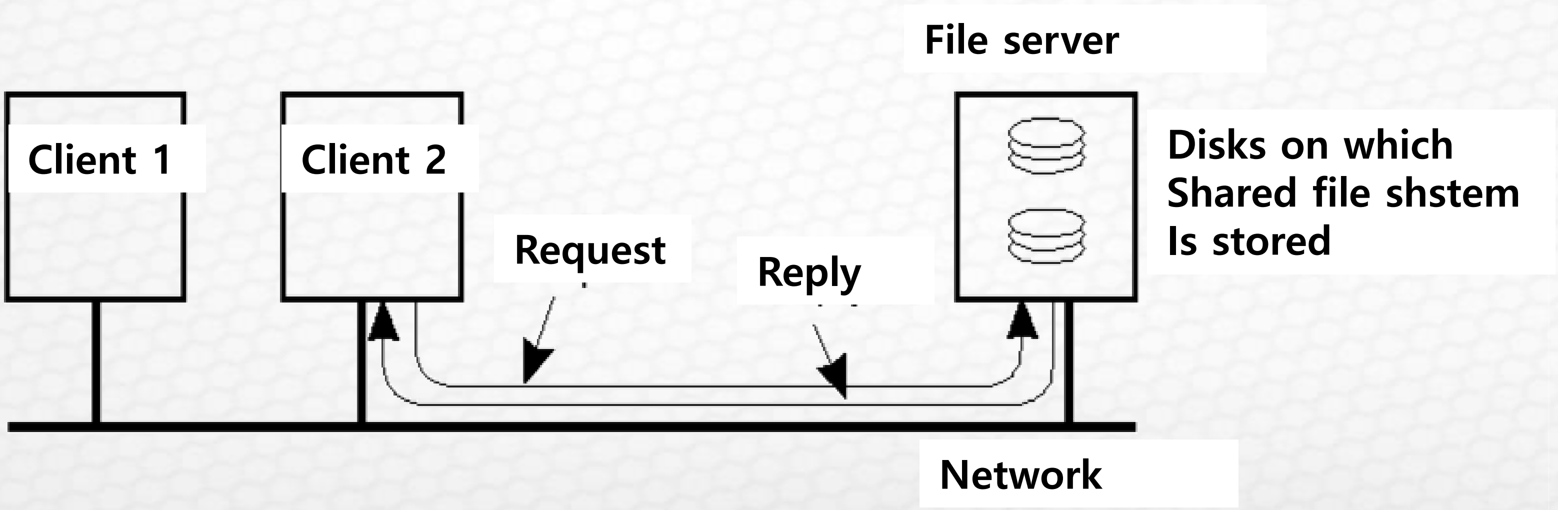
Network Operating System (1/2)



Network Operating System (2/2)

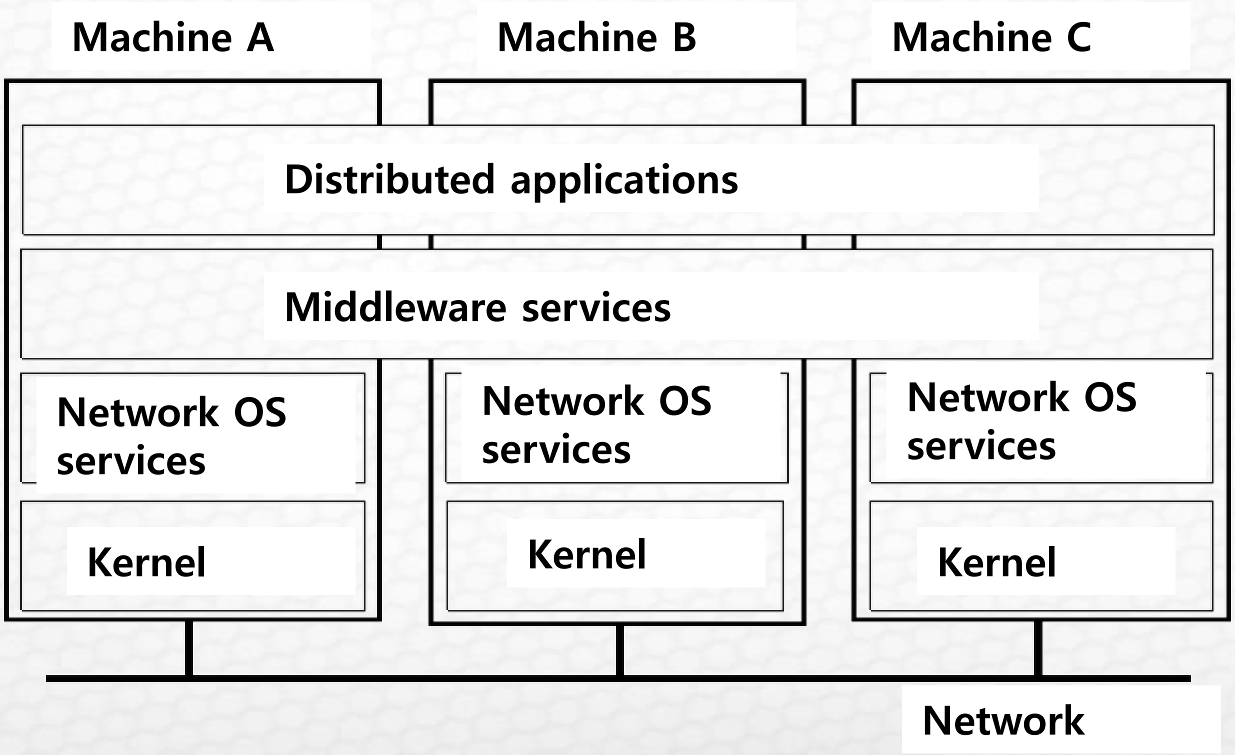
Employs a client-server model

- Minimal OS kernel
- Additional functionality as user processes



Middleware-based Systems

General structure of a distributed system as middleware



Distributed system organizations

Microcomputer model

- several multiuser systems

Workstations/PCs model

- each user has own WS/PC to do work
- each user shares files and other resources

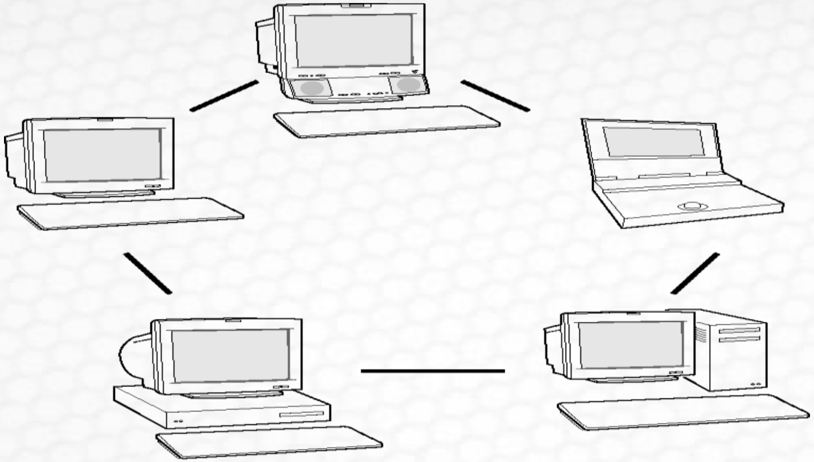
Processor pool model

LANs, MANs, WANs, WWW

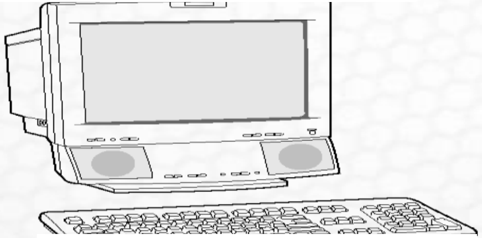
Distributed System : HW vs SW

		Software	
		Loosely-coupled	**Tightly-coupled**
Hardware	**Loosely-coupled**	■ Software Type: Network OS ■ Multicomputer ■ Each machine running its OS. OSes may differ.	■ Software Type: Integrated Distributed System ■ Group of Shared machines work like one computer but do not have shared memory
	Tightly-coupled	■ Doesn't make sense	■ Software Type: Multiprocessor Timesharing System ■ E.g. UNIX machine with several processors and several terminals

Connection Topology



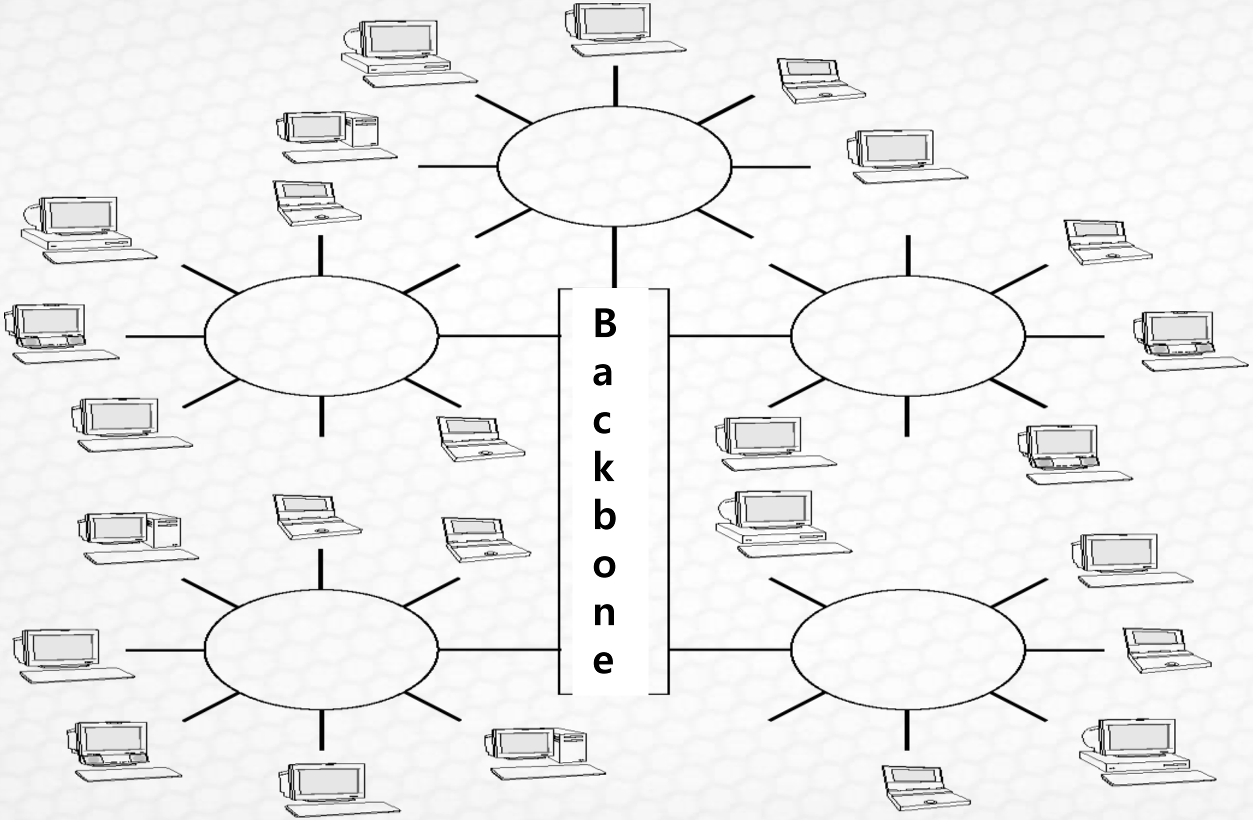
(a) Physical view



(b) Logical view

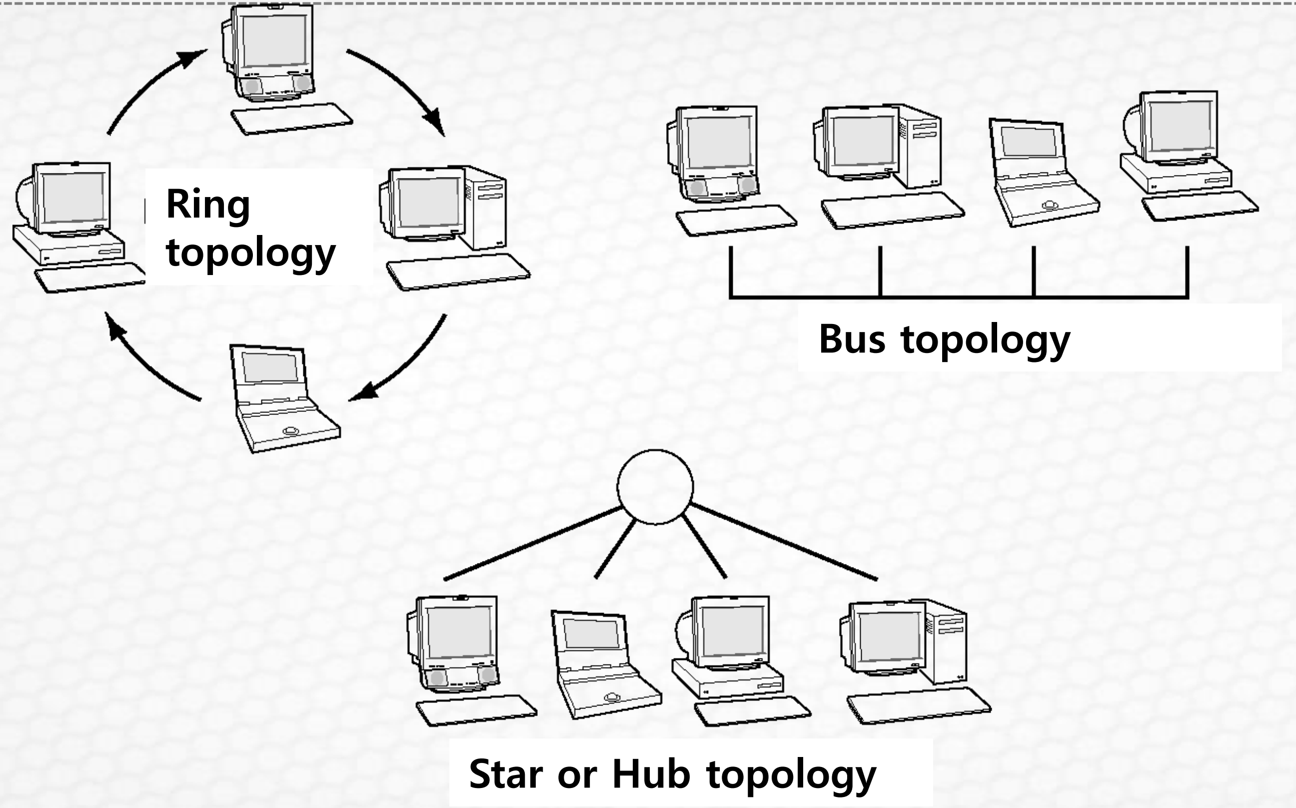
Computers in a Networked Environment. ***(Galli, p. 3)***

Connection Topology



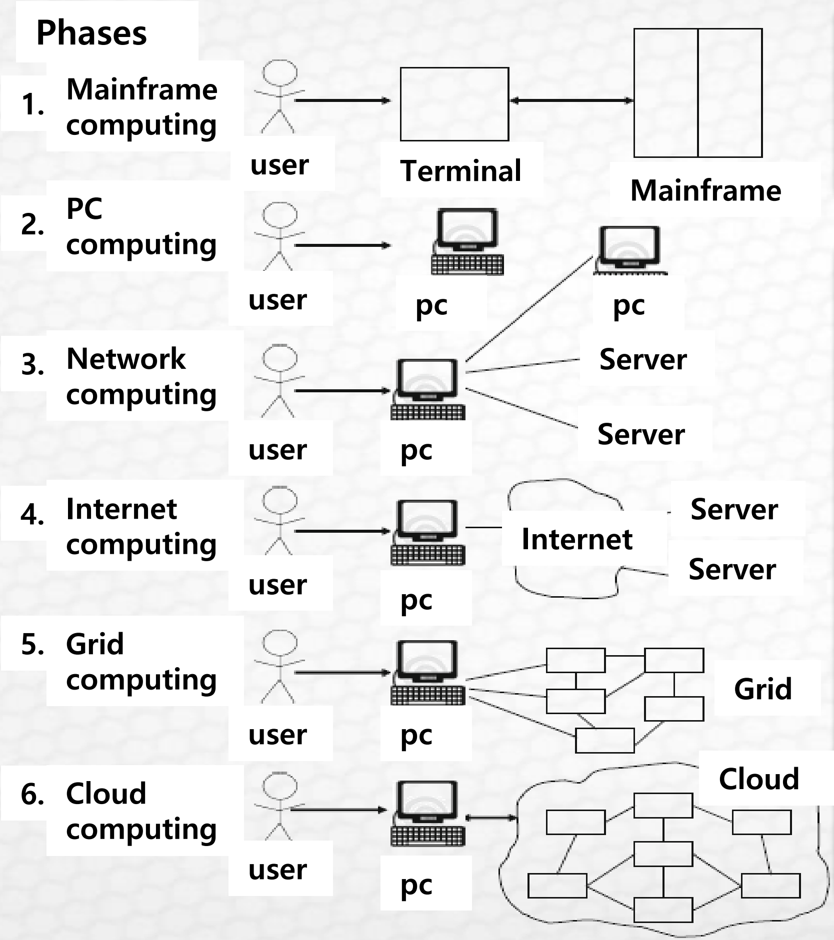
Connecting LAN Subnets with a Backbone. ***(Galli, p.6)***

Connection Topology



Common Wired LAN Topologies. ***(Galli, p.7)***

Six Computing Paradigm



It looks like that cloud computing is a return to the original mainframe c omputing paradigm

However, there are several important differences

- Mainframe computing offers finite
- Computing power
- Cloud computing provides almost infinite power and capacity
- In mainframe computing, dummy terminals acted as user interface devices
- In cloud computing, powerful PCs can provide local computing power and caching support

2. Issues in Distributed Systems

Global Knowledge

Unable to determine up-to-date global state

- no global memory
- no common clock
- unpredictable message delays

Need device-efficient distributed control

- e.g. how to get a consensus

Need method for ordering events

Naming

All objects are named

Need to map name onto its location

Need a directory (or directories)

- replicated (to maintain consistency)
- partitioned (which partition helps me?)

Scalability, Process Synch

Scalability

- Can system grow without performance degradation?
- Want to avoid centralized components

Process synchronization

- Enforce mutual exclusion to shared resources
- Deal with potential for deadlock

Compatibility

- Possible at different levels
- Binary level: all processing elements run same binary code
- Execution level: same source code can be compiled and run on all nodes
- Protocol level: all processing elements support same protocols

Resource management

Data migration: bring data to the location

- distributed file system
- distributed shared memory

Computation migration

- e.g. RPC
- e.g. send a query for info computed remotely instead of requesting raw data

Distributed scheduling

- process migration

Security

Authentication

➡ verify user identification

Authorization

➡ determine user privileges