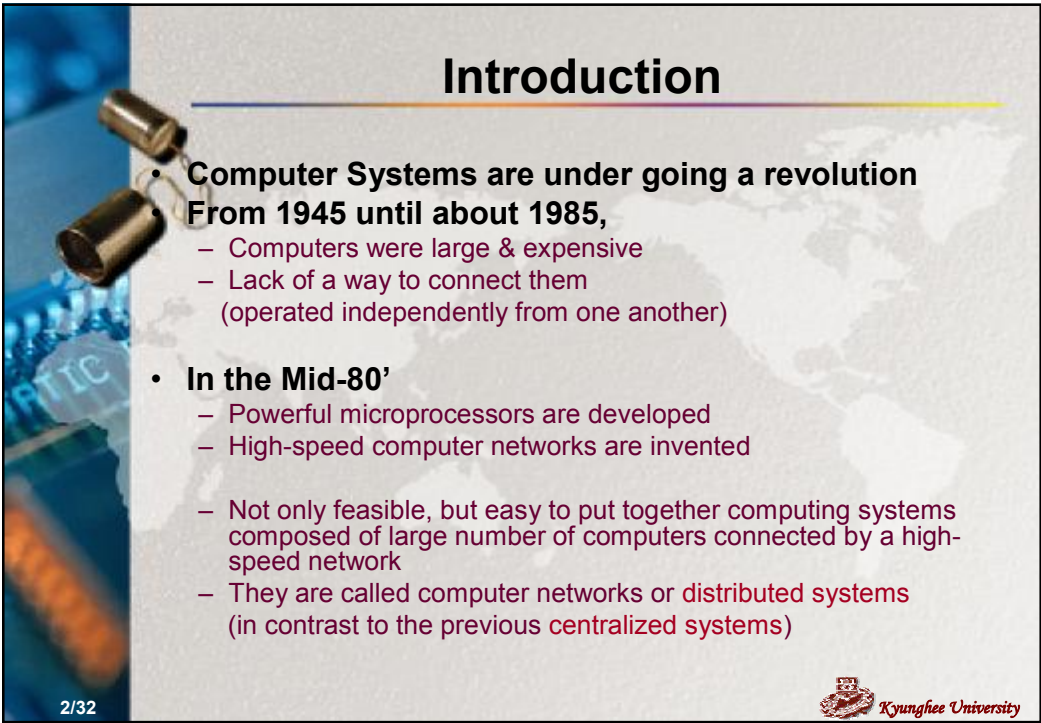



Introduction of Distributed System

Jaehun Bang


1/32



Introduction

- **Computer Systems are under going a revolution**
From 1945 until about 1985,
 - Computers were large & expensive
 - Lack of a way to connect them
(operated independently from one another)
- **In the Mid-80'**
 - Powerful microprocessors are developed
 - High-speed computer networks are invented
 - Not only feasible, but easy to put together computing systems composed of large number of computers connected by a high-speed network
 - They are called computer networks or distributed systems
(in contrast to the previous centralized systems)

2/32



1.1. Definition of a Distributed System

- “A Distributed system is a collection of independent computers that appears to its users as a single coherent system”
- Two aspect
 - Hardware: the machines are autonomous
 - Software: the users think they are dealing with a single system
- Characteristics
 - Differences between the various computers and the ways in which they communicate are hidden from users
 - Users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place

3/32



1.1. Definition of a Distributed System

- Characteristics
 - Relatively easy to expand or scale
 - Normally be continuously available
(although certain parts may be temporarily out of order)
 - Offering a single system view : **→ Middleware**
(in heterogeneous environments)

4/32



1.2. Goals

- A distributed system **should easily connect users to resources;**
- it should **hide the fact that resources** are distributed across a network
- it **should be open**
- and it **should be scalable**

5/32



1.2.1 Connecting Users and Resources

- **Is to make it easy for users to access remote resources, and to share them with other users in a controlled way**
 - Reason for sharing resources: economics
- **Easier to collaborate and exchange information**
 - Internet, groupware
- **Security** is becoming more and more important as connectivity and sharing increase
 - Tracking communication to build up a preference profile of a specific user

6/32



1.2.2. Transparency

- Is to hide the fact that its processes and resources are physically distributed across multiple computers → transparent

① access transparency

: **hiding differences in data representation and how a resource is accessed**

- Ex) to send an integer from intel-based workstation to SUN SPARC machine
- Ex) different naming convention

② location transparency

: **users cannot tell where a resource is physically located in the system** → “naming”

- Ex) assigning only logical names to resources

7/32



1.2.2. Transparency

③ migration transparency

: **resources can be moved without affecting how that resource can be accessed**

④ relocation transparency

: **resources can be relocated while they are being accessed without the user or application noticing anything**

- Ex) when mobile users can continue to use their wireless laptop while moving from place to place without ever being (temporarily) disconnected

⑤ replication transparency

: **resources may be replicated to increase availability or to improve performance by placing a copy close to the place where it is accessed**

- All replicas have the same name
- Support location transparency

8/32



1.2.2. Transparency

⑥ concurrency transparency

: **hide that a resource may be shared by several competitive users**

- Issue: concurrent access to a shared resource leaves that resource in a consistent state
 - Consistency can be achieved through locking mechanisms

⑦ failure transparent

: **a user does not notice that a resource fails to work properly, and that the system subsequently recovers from that failure**

- Difficulty in masking failures lies in the inability to distinguish between a dead resource and a painfully slow resource

9/32



1.2.2. Transparency

⑧ persistence transparency

: **masking whether a resource is in volatile memory or perhaps somewhere on a disk**

- In object-oriented databases user is unaware that the server is moving state between primary and secondary memory

• Degree of Transparency

- Trade-off between a high degree of transparency and the performance of a system

10/32



1.2.3. Openness

- **Open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of these services**
 - In computer networks
 - : standard rules govern the **format contents, meaning of messages sent and received**
 - Formalized in **protocols**
 - In Distributed system
 - : services are specified through interfaces (IDL: interface Definition Language)
 - Specify the names of the functions that are available together with type of the parameters, return values, possible exceptions
 - Hard part: the semantics of interfaces (by means of natural language)
→ **informal way**

11/32



1.2.3. Openness

- Once properly specified, an interface definition
 - Allows an arbitrary process to talk another process through that interface
 - Allows **two independent parties to build different implementations of those interface**
- **Proper specifications** are complete & neutral
 - **Complete**: everything that is necessary to make an implementation has indeed been specified (in real world not at all complete)
 - **Neutral**: specifications do not prescribe what the implementation should look like they should be neutral

Important for
interoperability
and portability

12/32



1.2.3. Openness

- **Interoperability**

- Two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard

- **Portability**

- An application developed for a distributed system A can be executed, without modification, on a different distributed system B that implements the same interfaces as A

- **Flexibility**

- It should be easy to configure the system out of different components possibly from different developers
- Easy to add new components or replace existing ones without affecting those components that stay in place

13/32



1.2.3. Openness

- **Separating Policy from Mechanism**

- To achieve flexibility, the system is organized as a collection of relatively small and easily replaceable or adaptable components
 - Implies that should provide definitions of the high-level interface and the internal parts of the system (how parts interact)
 - A component does not provide the optional policy for a specific user or application
ex : caching in WWW

14/32



1.2.3. Openness

- **Ex: caching policy**

- Browsers allow a user to adapt their caching policy by specifying the size of cache, whether a cached document should always be checked for consistency, or only once per session
- But, the user can not influence other caching parameters, how long a document may remain in the cache, or which document should be removed when the cache fills up. Impossible to make caching decisions based on the content of document.

- **We need a separation between policy & mechanism**

- Browser should ideally provide facilities for only storing documents (mechanism)
- Allow users to decide which documents are stored and for how long (policy)
- In practice, this can be implemented by offering a rich set of parameters that the user can set dynamically
- Even better is the a user can implement his own policy in the form of a component that can be plugged into the browser. the component must have an interface that the browser can understand.

15/32



1.2.4 Scalability

- **Measured 3 different dimensions**

- ① **Size** : can easily add more users and resources to the system
- ② **Geographically scalable system** : the users and resources may lie far apart
- ③ **Administratively scalable** : it can be easy to manage even if it spans many independent administrative organizations.

- **Some loss of performance as the system scales up**

16/32



Scalability problems

1. Size

: Confronted with the limitations of

- ① Centralized services : A single server for all users
 - problem : the server become a bottleneck as the number of user grows
 - Unavoidable using a single server : service for managing highly confidential information such as medical records, bank accounts.....
 - Copying the server to several locations to enhance performance → Vulnerable to security attack
- ② Centralized Data : A single on-line telephone book
 - problem : saturate all the communication lines into and out of a single database

17/32



③ Centralized Algorithm : Doing routing based on complete information.


- Problem : theoretical point of view the optimal way to do routing in collect complete information about load on all machines and lines and then run a graph theory algorithm to compute all the optimal route.
- Problem : messages would overload part of network
- Solution : Decentralized Algorithms should be used

Characteristics

- ① No machine has complete information about the system state.
- ② Machines make decisions based only on local information.
- ③ Failure of one machine does not ruin the algorithms.
- ④ There are no implicit assumption that a global check exists.
 - Clock synchronization is tricky in WAN

18/32







• 2. Geographical Scalability

- **Problem**
 - ① LAN used on synchronous communication.
WAN had to use synchronous communication.
 - ② In WAN, inherently unreliable (point-to-point)
In WAN, highly reliable communication (broad casting)
- Geographical scalability is related to the problems of centralized solutions that hinder size scalability.

19/32




• 3. Administrative scalability

- How to scale a distributed system across multiple, independent administrative domains.
- Problem needs to solved :
 - **Conflicting policies** with respect to resource usage management, and security.
- The trust does not expand naturally across domain boundaries
- If a distributed system expands to another domain, two **types of security measures need to be taken.**

Distributed system

- ① **Has at protect itself** against malicious attacks from the new domain
- ② **The new domain has to protect itself** against malicious attack from the distributed system.

20/32



Scaling Techniques

- ① Hiding communication latencies
- ② Distribution
- ③ Replication

21/32



① Hiding communication latencies

- **Solution** : Try to avoid waiting for responses to remote service requests as much as possible.
 - Asynchronous communication is a solution.

In reality, many applications not use of asynchronous communication

- **Example** : Interactive applications.
 - by moving part of computations which usually done at the server to the client process.
 - Accessing DB case Figure 1-4.

22/32



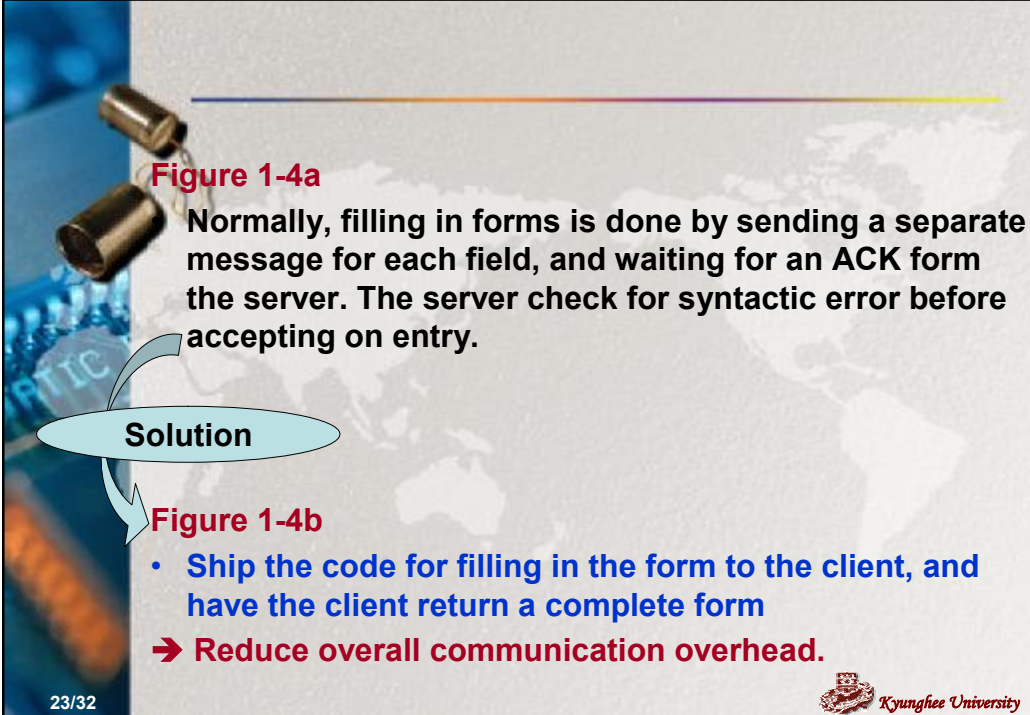


Figure 1-4a


Normally, filling in forms is done by sending a separate message for each field, and waiting for an ACK from the server. The server checks for syntactic error before accepting an entry.

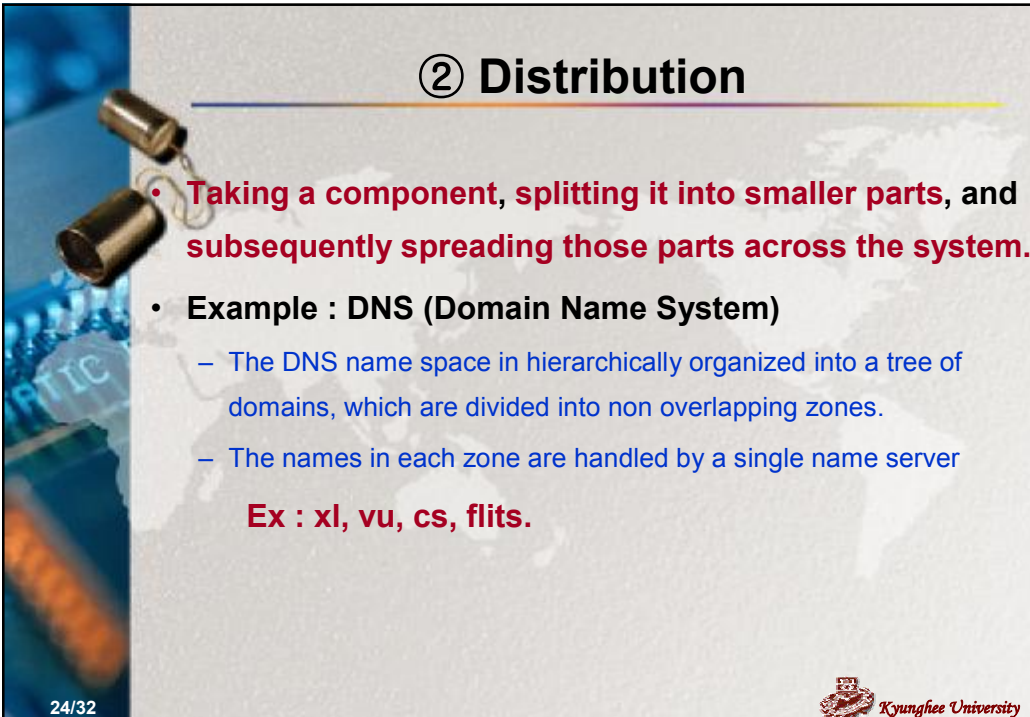
Solution

Figure 1-4b

- Ship the code for filling in the form to the client, and have the client return a complete form
- Reduce overall communication overhead.

23/32

 Kyunghee University




② Distribution

- Taking a component, splitting it into smaller parts, and subsequently spreading those parts across the system.
- Example : DNS (Domain Name System)
 - The DNS name space is hierarchically organized into a tree of domains, which are divided into non overlapping zones.
 - The names in each zone are handled by a single name server

Ex : xl, vu, cs, flits.

24/32

 Kyunghee University

③ Replication

- Scalability problems often appear in the form of performance degradation, it is a good idea to actually replicate components across a distributed system.
- **Replication increases availability and load balance.**
- **Having a copy nearby can hide much of the communication latency problems.**
- Caching is a special form of replication.
- **Leads to consistency problems.**

25/32



1.3 Hardware Concept

Even though distributed systems consist of multiple CPUs there are several ways the H/W can be organized
=> How they are interconnected & communicate

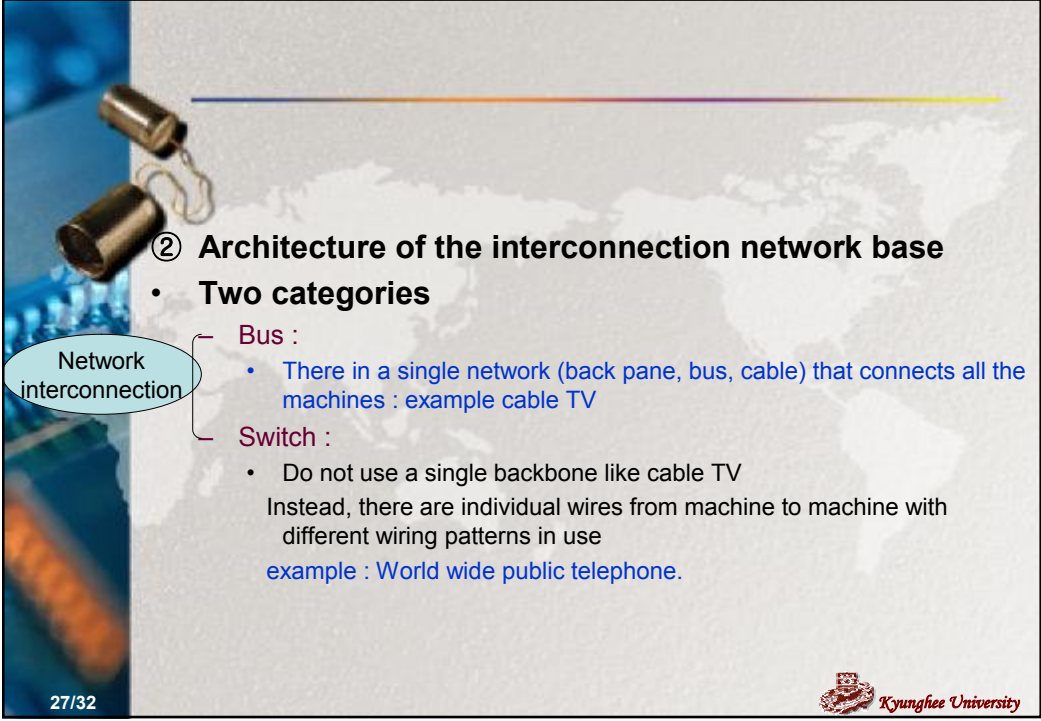
① Classification

- **Shared Memory => multiprocessors :**
 - a single physical address space that is shared by all CPUs
- **Non shared Memory => Multicomputers :**
 - Every machine has its own private memory.
 - Common example : a collection of PC connected by a network

memory

26/32






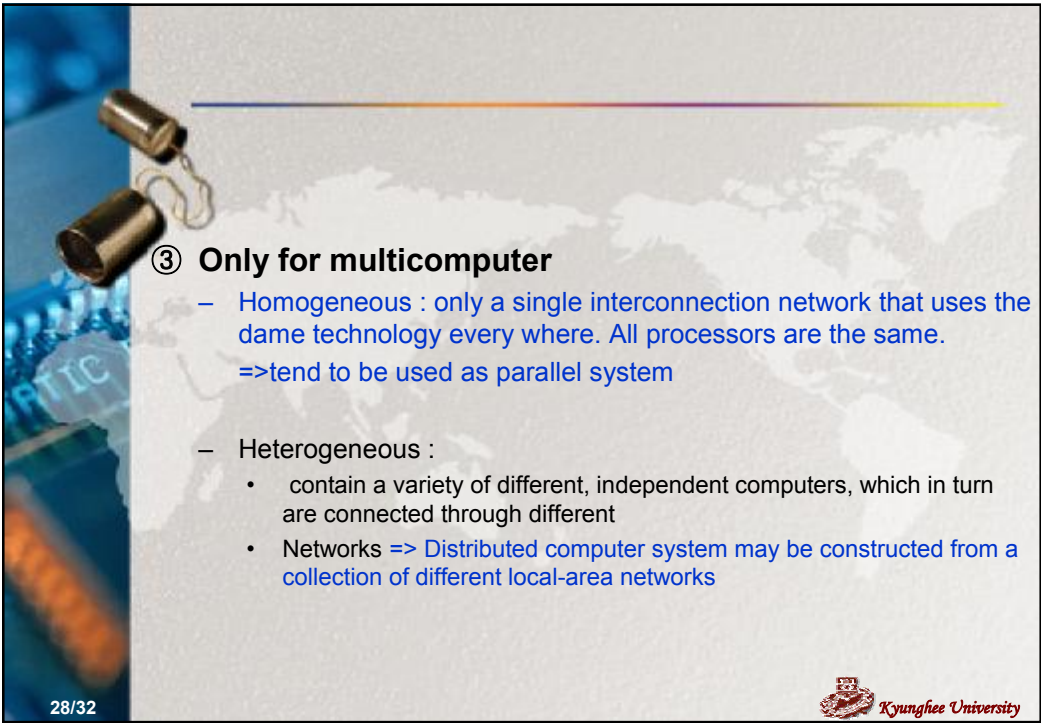
② **Architecture of the interconnection network base**

- **Two categories**
 - **Bus :**
 - There in a single network (back pane, bus, cable) that connects all the machines : example cable TV
 - **Switch :**
 - Do not use a single backbone like cable TV
 - Instead, there are individual wires from machine to machine with different wiring patterns in use
 - example : World wide public telephone.

Network interconnection

27/32


 Kyunghee University



③ **Only for multicomputer**

- **Homogeneous :** only a single interconnection network that uses the same technology every where. All processors are the same.
=>tend to be used as parallel system
- **Heterogeneous :**
 - contain a variety of different, independent computers, which in turn are connected through different
 - Networks => Distributed computer system may be constructed from a collection of different local-area networks

28/32

 Kyunghee University

1.3.1 Multiprocessors

- Share a single key property : **all the CPUs have direct access to the shared memory.**
- **Coherent** : since there is only one memory, if CPU A writes a word to memory and then CPU B reads that word back a microsecond later, B will get the value just written.
- **Problem** : with as few as 4 or 5 CPUs, the bus will usually be **overloaded and performance will drop drastically**

29/32



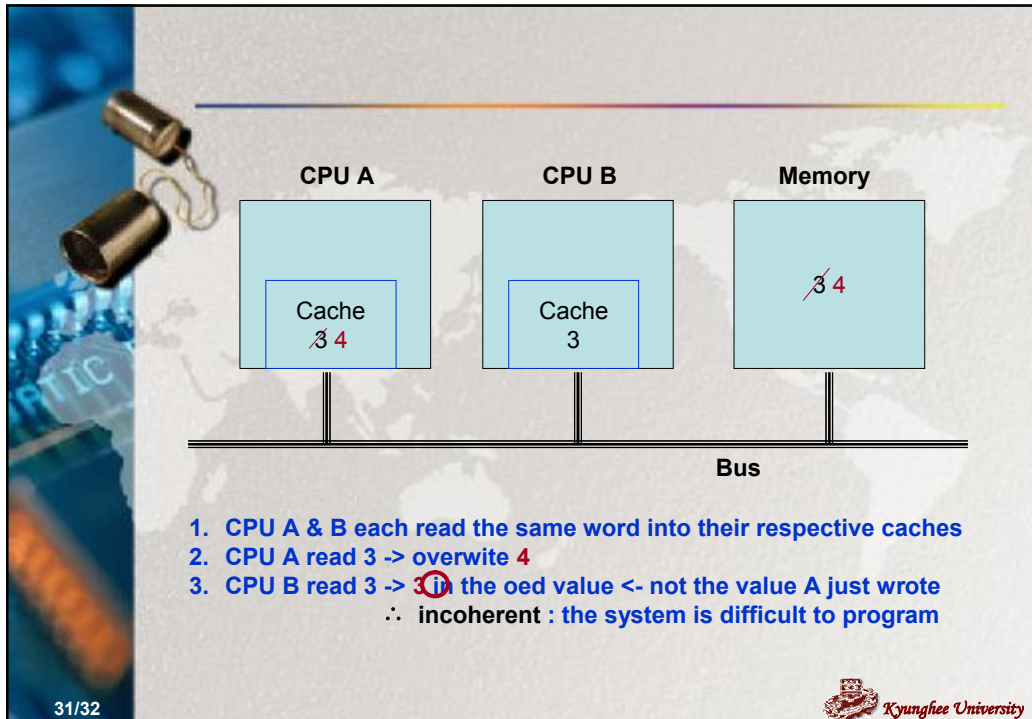
- **Solution** : **is to add** a high-speed cache memory between the CPU & the bus.
[Fig 1-7] -> **reduce bus traffic**
hitrate : the probability of success (the word requested is in the cache)
ex: cache size : 512KB to 1MB are common
hit rate 90% or more

cache


- Another problem of cache : **memory incoherent**

30/32






- **Another problem of bus-based Multiprocessor**
 - ↓
 - limited scalability** even when using “cache”
 - Crossbar switch [refer to Fig 5.8-3]
 - **The virtue of the crossbar switch**
 - Many CPUs can be accessing memory at the same time although if two CPUs try to access the same memory simultaneously, one of them will have to wait
 - **Downside of the crossbar switch**
 - Exponential growth of crosspoint switch number if # of CPU(n) grow large. (n^2)
 - **Omega network (require fewer switch) [Fig 1-8-6]**
 - With proper settings of the switches, every CPU can access every memory
- 32/32
- Kyunghee University




• Drawback of Switching Network

- There may be several switching stages between the CPU and Memory
 - > Consequently, to ensure low latency between CPU & memory, switching has to be extremely fast, which is expensive
reduce the cost of switching
- Hierarchical system : NUMA (NonUniform Memory Access) machine
 - Some memory is associated with each CPU
 - Each CPU can access its own local memory quickly, but accessing anybody else memory is slower
 - Another complication :
placement of the programs and data becomes critical in order to make access go to the local memory

33/32




Kyunghee University



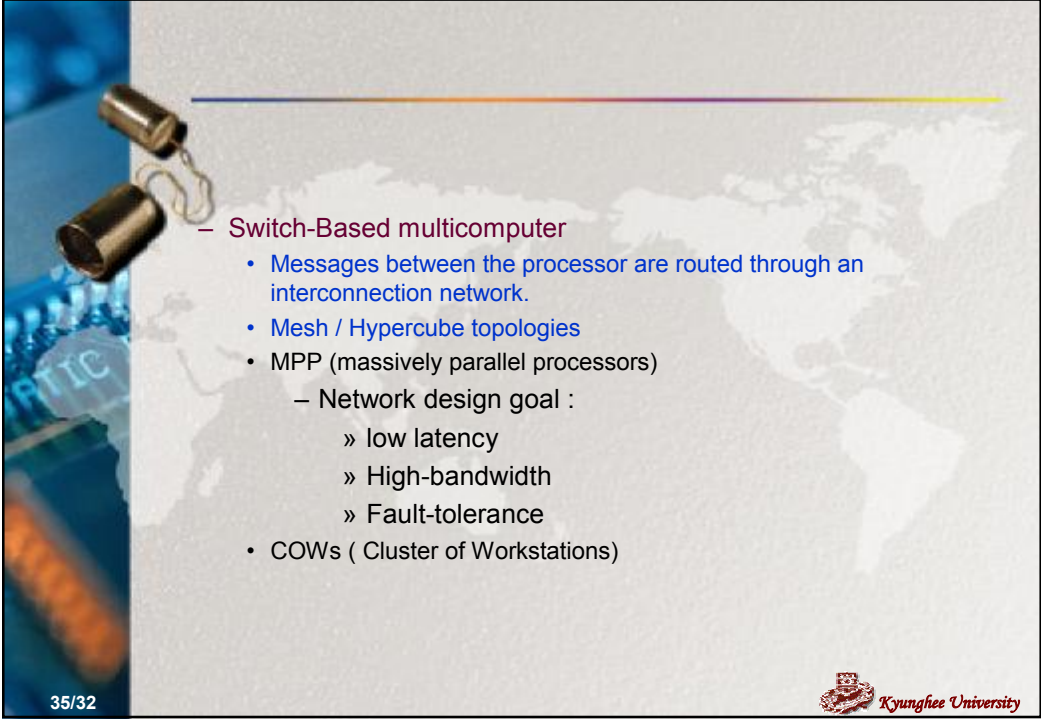
1.3.2. Homogeneous Multicomputer Systems

- **Relatively easy to build compare to Multiprocessors**
Each CPU has a direct connection to its own local memory
- **How the CPUs communicate with each other (CPU-to-CPU communication)**
Volume of traffic is much lower than that of CPU-to-memory
 - SAN (System Area Networks) : connected through a single interconnection network
 - Bus-based multicomputer
 - The processors are connected through a shared multiaccess network
 - Limited scalability
 - Broad-cast

34/32




Kyunghee University

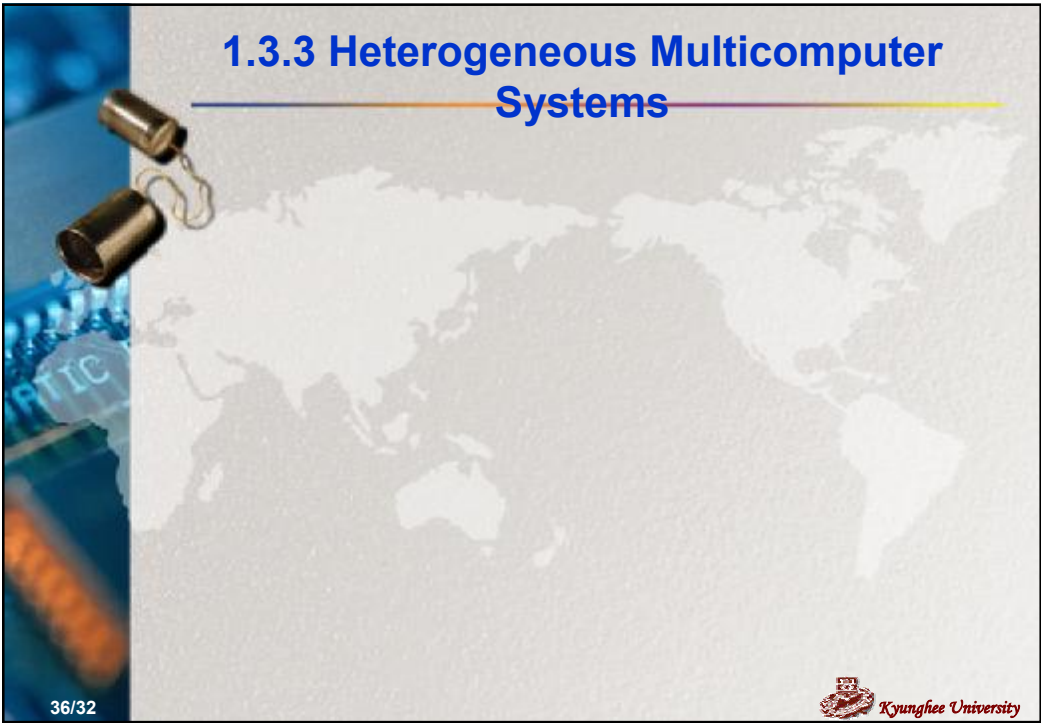


– Switch-Based multicomputer

- Messages between the processor are routed through an interconnection network.
- Mesh / Hypercube topologies
- MPP (massively parallel processors)
 - Network design goal :
 - » low latency
 - » High-bandwidth
 - » Fault-tolerance
- COWs (Cluster of Workstations)


35/32

 Kyunghee University



1.3.3 Heterogeneous Multicomputer Systems

36/32

 Kyunghee University

1.4. Software Concepts

- **Distributed system :**

- Matter of software not matter of HW
- Act as resource managers for the underlying H/W
- Attempt to hide the intricacies and heterogeneous nature of underlying hardware by providing a virtual machine on which applications can be easily executed.

- **OS for distributed computers**

- Tightly coupled system :
 - Try to maintain a single, global view of the resources it manages : DOS (Distributed OS)
 - >need for managing multiprocessors and homogeneous multicomputers
- Loosely-coupled system : Collection of computers each running their own operating system : NOS (Network OS)
 - >local services are made available to remote clients

37/32



※Enhancements to the services of network OS are needed such that better support for distribution transparency ->middleware :

lie at the heart of modern distributed system

※Figure 1-10 : overview between DOS, NOS, and MW

System	Description	Main goal
DOS	Tightly-coupled OS for multiprocessor and homogeneous multicomputers	Hide & manage Hardware resource
NOS	Loosely-coupled OS for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middlew are	Additional layer a top of NOS implementing general-purpose services	Provide distribution transparency

38/32



1.4.1. DOS

- Uniprocessor OS
- Multiprocessor OS
- Multicomputer OS
- Distributed Shared Memory Systems

39/32




Uniprocessor OS

- Allow uses/applications an easy way of sharing resources
(CPU, Main Memory, Disks, peripheral device)
- Virtual machine : To an application, it **appears as if it has its own resources**, and that there may be **several applications executing on the same system at the same time**, each with **their own set of resources**
sharing resources : applications are protected from each other


40/32






- **Kernel mode** : all instructions are permitted to executed, and the whole memory and collection of all registers is accessible during execution
 - ※ OS should be in full control of how the H/W resources are need & shared
- **User mode** : Memory & register access is restricted.
- Only way to switch from user mode to kernel mode is through “system call”.
- **Monolithic OS** :
 - Run in a single address space
 - Difficult to adapt the system
 - Not good idea for openness, SE, reliability or maintain ability

41/32




Kyunghee University

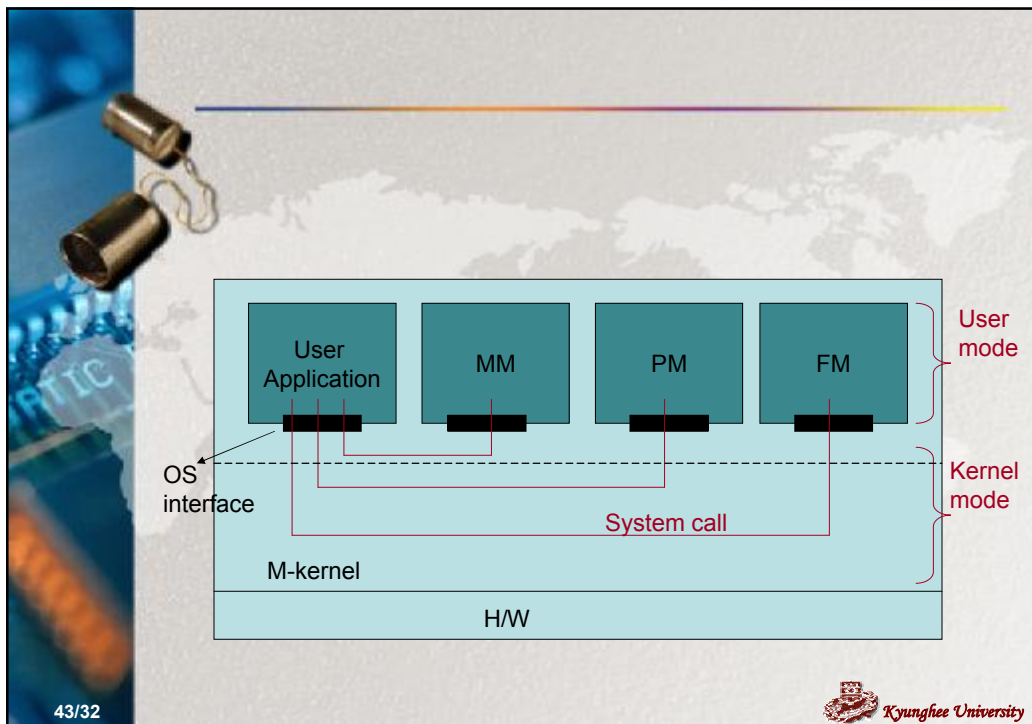


- **Microkernel OS** :
 - Containing only the code that must execute in kernel mode
 - Only contain the code for setting device registers, switching the CPU between processes, manipulating MMU, and capturing hardware interrupts.
 - Contains the code to pass system calls to calls on the appropriate user-level OS modules, and to return their results.

42/32



Kyunghee University



43/32

- **Main Benefits to using Microkernels**
 - **Flexibility**
 - A large part of OS is executed in user mode, it is relatively easy to recompile or re-install the entire system
 - User level modules can be place on different machines
 - **Disadvantages**
 - Different from the way current OS work (meets massive resistance)
 - Extra communication cost ->performance loss

44/32

Multiprocessor OS


- An important is to support for multiple processors having access to a shared memory.
- Data have to be **protected** against concurrent access to guarantee **consistency** but can't easily handle multiple CPUs since they have been designed as monolithic programs that **can be executed only with a single thread of control** => need redesigning and reimplementing the entire kernel

45/32

• Goal of multiprocessor OS :


- To support high performance through multiple CPUs
- Is to make the # of CPUs transparent to the application.
- communication between different part of applications uses the same primitives as these in multitasking uniprocessor OS.
- All communication is done by manipulating data at shared memory locations => protect that data against simultaneous access
=> protection is done through synch primitives : semaphore / Monitor
- ※ Explain Semaphore / Monitor

46/32




- **Semaphore :**
 - Error-prone except when need for simply protecting shared data
 - Problem : easily lead to unstructured code (goto statement)
- **Monitor :**
 - Programming – language construct similar to an object in O-based programming
 - Problem : they are programming – language constructs
 - java provides a notion of monitors by essentially allowing each object to protect itself against concurrent access through synchronized statements, and operations **wait** and **notify** on objects.

47/32




Kyunghee University

Multicomputer OS




- Different (totally) structure and complexity than multiprocessor OS
- Means of communication : **Message passing** (not enough for shared memory)
- [Fig 1-14]
 - Each mode has its own kernel containing modules for managing local resources such as memory, the local CPU, a local disk, and soon.
- Each kernel support **parallel** and **concurrent execution** of various tasks
 - Software implementation of shared memory =>by means of message passing
 - Assigning a task to a processor
 - Masking transparent storage
 - General interprocess communication

48/32



Kyunghee University




- **Message-passing**
 - Semantics of message-passing primitive vary between different systems

Differences

- Considering whether or not messages are buffered
- Take into account when a sending or receiving process is blocked

- **Two places where messages can be buffered**
 - Sender's side
 - Receiver's side
- **4 synchronization points : at which a sender or receiver can block.**

49/32



Kyunghee University