Unit 1:
Introduction to Distributed System

References:

1. G. Coulouris, J. Dollimore and T. Kindberg; **Distributed Systems Concepts and Design,4th Edition.**
2. Andrew S. Tanenbaum and Maarten van Steen**; Distributed Systems: Principles and Paradigms, 2nd Edition.**

Unit I: Introduction to Distributed Systems

1.1 Distributed Systems:

- Definition, characteristics, and types of distributed systems.

1.2 Goals of Distributed Systems:

- Scalability, transparency, reliability, and fault tolerance.

1.3 Distributed System Architectures

1.3.1 Client-server model:

- architecture, advantages, and limitations

1.3.2 Peer-to-peer model:

- decentralized approach, applications

1.3.3 Hybrid models:

- combining client-server and peer-to-peer

Defining Distributed Systems

- Various definition of distributed systems have been given in the literature, for example:

  A collection of logically related data that is distributed over different processing nodes of computer network.

- Definition above does not provide all characteristics of distributed systems.

Defining Distributed Systems

- It is difficult to find a definition that provides all characteristics of distributed systems.
- Another way is a definition that gives a loose characterization for distributed systems such as:
  - A distributed system is a collection of independent computers that appear to the users of the system as a single computer.
- With any definition, sharing of resources is a main motivation for constructing distributed systems.

Defining Distributed Systems

- We define distributed systems more precisely as :
  - A distributed system is one in which hardware or software components located at networked computers communicate and coordinate their actions only by **message passing**.
- Definition above covers the entire range of distributed systems in which networked computers can usefully be deployed.

Defining Distributed Systems

- Networks of computers are everywhere!
- Examples of networks of computers are:
  - Mobile phone networks
  - Corporate networks
  - Factory networks
  - Campus networks
  - Home networks
  - In-car networks
  - On board networks in aero planes and trains

Defining Distributed Systems

- Our definition of distributed systems has the following significant consequences:
  - Concurrency
    - Tasks carry out independently
  - No global clock
    - Tasks coordinate their actions by exchanging messages

Defining Distributed Systems

- Independent Failures
  - Faults in the network result in the isolation of the computers that are connected to it.
  - Each component of the system can fail independently, leaving the others still running.

Defining Distributed Systems

- A collection of independent computers that appears to its users as a single coherent system.
- One that looks like an ordinary system to its users, but runs on a set of autonomous processing elements (PEs) where each PE has a separate physical memory space and the message transmission delay is not negligible.
- There is close cooperation among these PEs. The system should support an arbitrary number of processes and dynamic extensions of PEs.

Defining Distributed Systems

- Distributed Systems encounter number of terminologies:
  - distributed, network, parallel, concurrent, and decentralized.
- Parallel means lockstep actions on a data set from a single thread of control.
- Distributed means that the cost or performance of a computation is governed by the communication of data and control.
- A system is centralized if its components are restricted to one site, decentralized if its components are at different sites with no or limited or close coordination
- When a decentralized system has no or limited coordination, it is called networked; otherwise, it is termed distributed indicating a close coordination among components at different sites.

Characteristics of Distributed System(1)

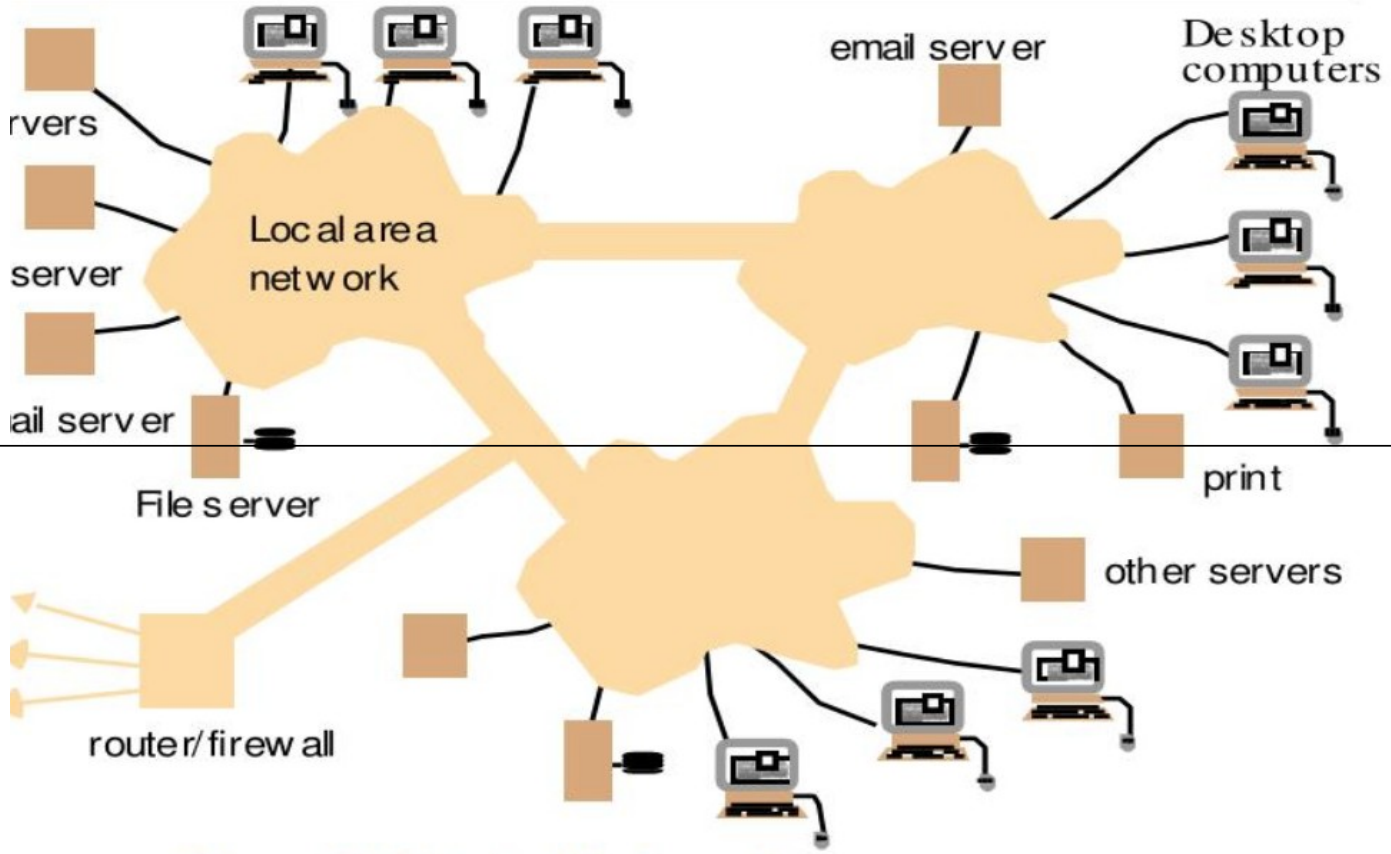
- **Resource Sharing:**
  It is the ability to use any Hardware, Software, or Data anywhere in the System.
- **Openness:**
  It is concerned with Extensions and improvements in the system (i.e., How openly the software is developed and shared with others)
- **Concurrency:**
  It is naturally present in Distributed Systems, that deal with the same activity or functionality that can be performed by separate users who are in remote locations. Every local system has its independent Operating Systems and Resources.

Characteristics of Distributed System(2)

- **Scalability:**
  It increases the scale of the system as a number of processors communicate with more users by accommodating to improve the responsiveness of the system.
- **Fault tolerance:**
  It cares about the reliability of the system if there is a failure in Hardware or Software, the system continues to operate properly without degrading the performance the system.
- **Transparency:**
  It hides the complexity of the Distributed Systems to the Users and Application programs as there should be privacy in every system.

Types of Distributed System

- Client/Server Systems
- Peer-to-Peer Systems
- Middleware
- Three-tier
- N-tier



Advantages of Distributed Systems over Centralized Systems

1. **Economics:** a collection of microprocessors offer a better price/performance than mainframes.
2. **Speed**: A distributed system may have more total computing power than a mainframe. Ex. 10,000 CPU chips, each running at 50 MIPS(million instruction per second). Not possible to build 500,000 MIPS single processor since it would require 0.002 nsec instruction cycle. Enhanced performance through load distributing.
3. **Inherent distribution**: Some applications are inherently distributed. Ex. Computerized worldwide Airline reservation, computerized banking system in which a customer can deposit/withdraw money from his/her account from any branch of the bank.
4. **Reliability**: If one machine crashes, the system as a whole can still survive. Higher availability and improved reliability.
5. **Incremental growth**: Computing power can be added in small increments. Modular expandability
6. **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

1. **Data sharing**: allow many users to access to a common databases

2. **Resource Sharing**: Expensive peripherals like color printers

3. **Communication**: enhance human-to-human communication, e.g., email, chat

4. **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
- Distribution of control
  - Hard to detect faults
  - Administration issues
- Performance
- Interconnect & servers must scale

Goals of D.S.

- Transparency
- Fault Tolerance
- Reliability
- Performance
- Scalability

General Examples of Distributed Systems

- Internet/Intranets/extranets
- Mobile networks
- DNS : Distributed Database system

Other Examples of Distributed Systems

- Cluster
  - A type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers cooperatively working together as a single, integrated computing resource. The computers may be standard per uniprocessor or multiprocessor.
  - A cluster can be used for providing highly scalable services such as search engines provide for users all over the Internet.

Other Examples of Distributed Systems

- Grid
  - A type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements.

Resource sharing and the Web

- The World Wide Web
  - The World Wide Web is an evolving system for publishing and accessing resources and services across the Internet.

(Figure 4)

Resource sharing and the Web

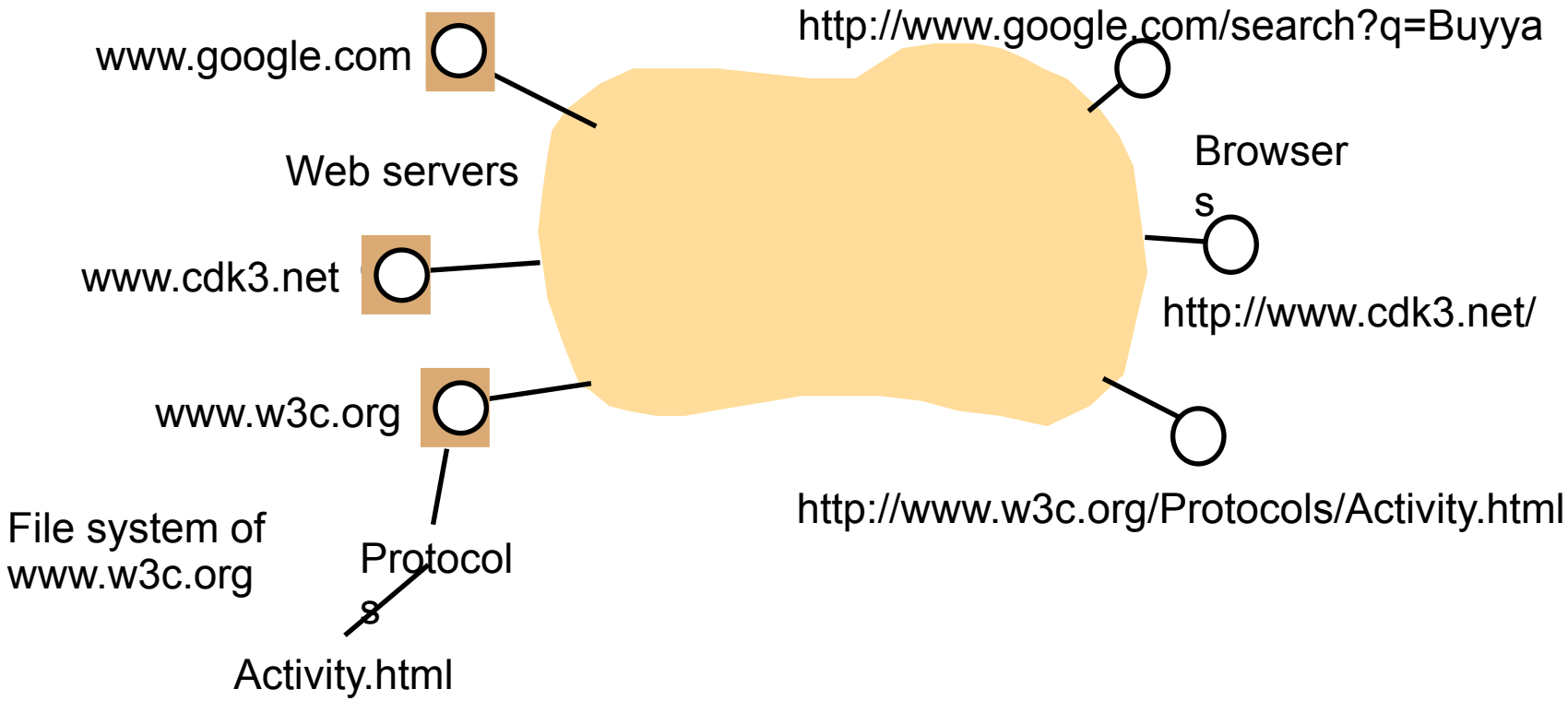


Figure 4. Web servers and web browsers.

Design Challenges of Distributed Systems

- Designers of distributed systems need to take the following challenges into account:
  - Heterogeneity
    - Heterogeneous components must be able to interoperate.
  - Openness
    - Interfaces should allow components to be added or replaced.
  - Security
    - The system should only be used in the way intended.

Design Challenges of Distributed Systems

- Scalability
  - System should work efficiently with an increasing number of users.
  - System performance should increase with inclusion of additional resources.
- Failure handling
  - Failure of a component (partial failure) should not result in failure of the whole system.

- Transparency
  - Distribution should be hidden from the user as much as possible.

Heterogeneity

- Heterogeneous components that must be able to interoperate, apply to all of the following:
  - Networks
  - Hardware architectures
  - Operating systems
  - Programming languages
  - Program written by multiple programmer

Heterogeneity

- Examples that mask differences in network, operating systems, hardware and software to provide heterogeneity are
  - Middleware
  - Internet protocols
  - Mobile code

Heterogeneity

- Middleware
  - Middleware applies to a software layer.
  - Middleware provides a programming abstraction.
  - Middleware masks the heterogeneity of the underlying networks, hardware, operating systems and programming languages.
  - The Common Object Request Broker (CORBA) is a middleware example.

Heterogeneity

- Mobile code
  - Mobile code is the code that can be sent from one computer to another and run at the destination.
  - Java applets are the example of mobile codes.
- Virtual machine
  - Virtual machine provides a way of making code executable on any hardware.

Openness

- Distributed systems must be extensible.
- Openness of a computer system is the characteristic that determines whether the system can be extended and re-implemented in various ways.

Openness

- The first step in openness is publishing the documentation of software components and interfaces of the components to make them available to software developers.
- **Monolithic Kernel**: systems calls are trapped and executed by the kernel. All system calls are served by the kernel, e.g., UNIX.
- **Microkernel**: provides minimal services
  - IPC
  - some memory management
  - some low-level process management and scheduling
  - low-level i/o (E.g. multiple system interfaces.)

Reliability

- Distributed system should be more reliable than a single system.
  - **Availability**: fraction of time the system is usable.
  - **Redundancy** improves it.
    - Need to maintain **consistency**
    - Need to be **secure**
  - **Fault tolerance**: need to mask failures, recover from errors.

Performance

- Performance loss due to communication delays:
  - fine-grain parallelism: high degree of interaction
  - coarse-grain parallelism

  - *(**Granularity** is the extent to which a system is broken down into small parts, either the system itself or its description or observation)*
- Performance loss due to making the system fault tolerant

Security

- Security of a computer system is the characteristic that the resources are accessible to authorized users and used in the way they are intended.

- Security for information resources has three components:
  - Confidentiality
    - Protection against disclosure to unauthorized individual.

Security

- Integrity
  - Protection against alteration or corruption.
- Availability
  - Protection against interference with the means to access the resources.

- Security Mechanisms are:
  - Encryption
  - Authentication
  - Authorization

Security

Security Challenges have not yet been fully met are:

- Denial of service attack:
  - Bombarding the service with a large number of pointless requests that the serious users are unable to use it.
- Security of mobile code:
  - Mobile code needs to be handled with care.
  - Example: Executable attachments in email.

Scalability

- Scalable distributed systems operate effectively and efficiently at many different scales, ranging from a small Intranet to the Internet.
- Scalable distributed systems remain effective when there is a significant increase in the number of resources and the number of users.

Scalability

- System should work efficiently with an increasing number of users.
- System performance should increase with inclusion of additional resources
- Techniques that require resources linearly in terms of the size of the system are not scalable. (e.g., broadcast based query won't work for large distributed systems.

Scalability

- Challenges of designing scalable distributed systems are:
  - Controlling the cost of physical resources
    - Cost should linearly increase with the system size.
  - Controlling the performance loss
    - For example, in hierarchically structured data, search performance loss due to data growth should not be beyond O(*log n*), where n is the size of data.

Scalability

- Preventing software resources running out
  - An example is the numbers used as Internet addresses (IP)(32 bit->128-bit)
  - Y2K like problem.
- Avoiding performance bottlenecks
  - Using decentralized algorithms to avoid having performance bottlenecks.

- Caching and replication in Web are examples of providing scalability.

Failure handling

- Failures in distributed systems are partial, that is some components fail while others continue to function.

- Techniques for dealing with failures:
  - Detecting failures
    - E.g. Checksums
  - Masking failures
    - E.g. Retransmission of corrupt messages
    - E.g. File redundancy

Failure handling

- Tolerating failures
  - E.g. Exception handling
  - E.g. Timeouts
- Recovery from Failure
  - E.g. Rollback mechanisms
- Redundancy
  - E.g. Redundant components

Concurrency

- With concurrency, services and applications can be shared by clients in a distributed system.
- For an object to be safe in a concurrent environment, its operations must be synchronized in such a way that its data remains consistent.
- Concurrency can be achieved by standard techniques such as semaphores, which are used in most operating systems.

Transparency

- Transparency is defined as the hiding of the separation of components in a distributed systems from the user and the application programmer.
- With transparency the system is perceived as a whole rather than a collection of independent components.

Transparency

- Forms of transparencies:
  - Access transparency
    - Enables local and remote resources to be accessed using identical operations.
  - Location transparency
    - Enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).

Transparency

- Concurrency transparency
  - Enables several processes to operate concurrently using shared resources without interference between them.
- Replication transparency
  - Enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

Transparency

- Failure transparency
  - Enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
- Mobility transparency
  - Allows the movement of resources and clients within a system without affecting the operation of users or programs.

Transparency

- Performance transparency
  - Allows the system to be reconfigured to improve performance as loads vary.
- Scaling transparency
  - Allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Transparency

- The two most important transparencies are access and location transparency referred to together as network transparency.
- Presence or absence of network transparency most strongly affects the utilization of distributed resources.

Forms of 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 or is migrated to newer version
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 shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Pitfalls when Developing Distributed Systems

- False assumptions made by first time developer:
  - The network is reliable.
  - The network is secure.
  - The network is homogeneous.
  - The topology does not change.
  - Latency is zero.
  - Bandwidth is infinite.
  - Transport cost is zero.
  - There is one administrator.

True Distributed System

- A distributed system with all the challenges like transparency, scalability, Dependability, Performance, and Flexibility, being solved to its full extent, is called a True Distributed System (TDS).

- However, because of the different problems (like a need of new component (network), security and software complexity faced during the development of a distributed system, a TDS is virtually possible.

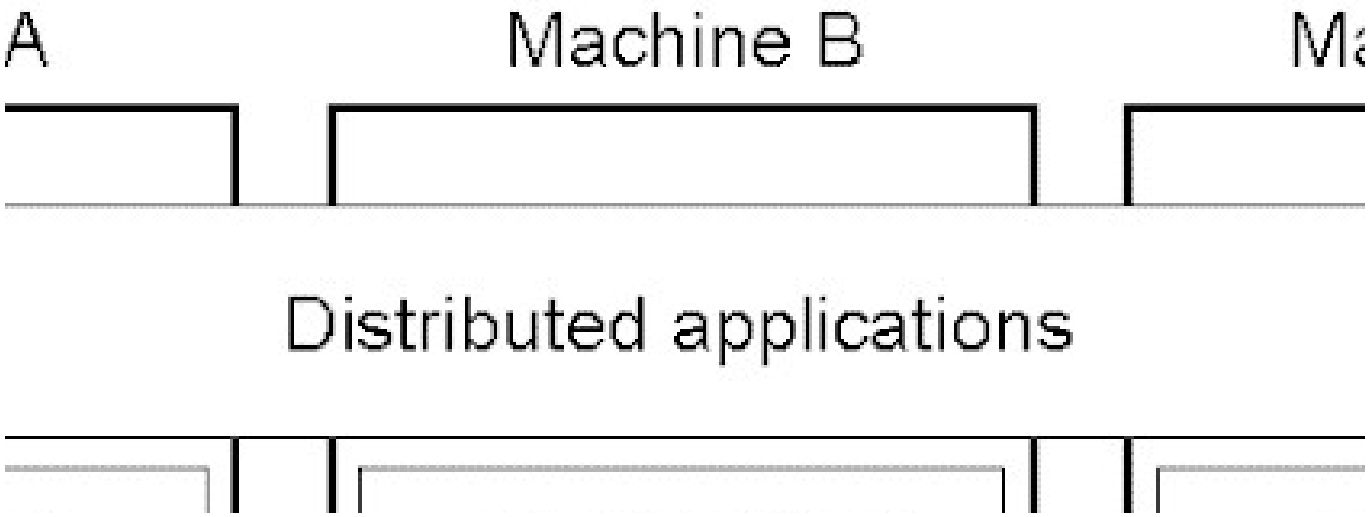
Software Concept in Distributed System

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

An overview between

- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)
- Middleware

Network Operating System



Loosely-coupled operating system for heterogeneous multi-computers (LAN and WAN). Weak transparency.

NOS: characteristics

- **Network Operating System**
  - extension of centralized operating systems
  - offer local services to remote clients
  - each processor has own operating system
  - user owns a machine, but can access others (e.g. rlogin, telnet)
  - no global naming of resources
  - system has little fault tolerance
  - e.g. UNIX, Windows NT, 2000

Distributed System as a Middleware Service

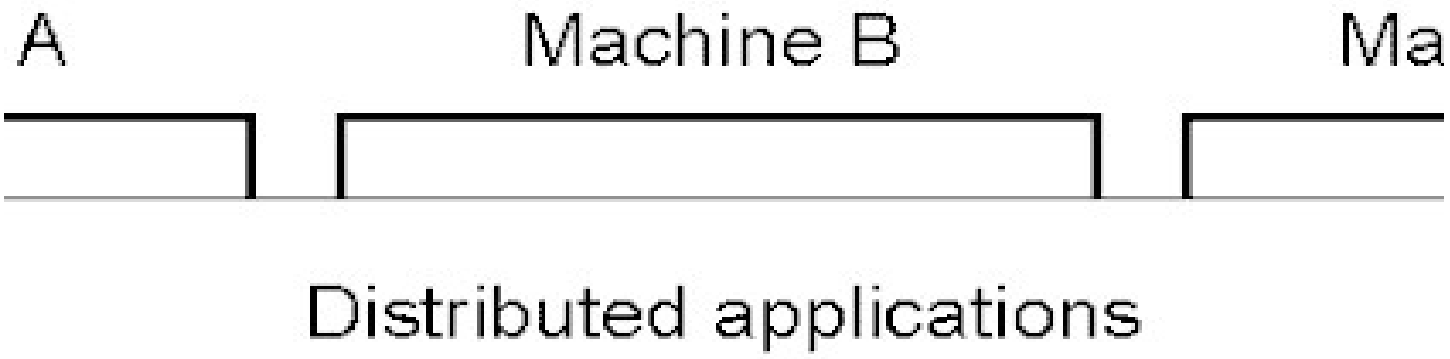
Distributed applications

Additional layer on the top of NOS implementing general-purpose services. Better transparency.

Middleware Examples

- **Examples: Sun RPC, CORBA, DCOM, Java RMI** (distributed object technology)
- **Built on top of transport layer** in the ISO/OSI 7 layer reference model: application (protocol), presentation (semantic), session (dialogue), transport (e.g. TCP or UDP), network (IP, ATM etc), data link (frames, checksum), physical (bits and bytes)
- **Most are implemented over the internet protocols**
- **Masks heterogeneity of underlying networks, hardware, operating system and programming languages – so provides a uniform programming model with standard services**
- **3 types of middleware:**
  - transaction oriented (for distributed database applications)
  - message oriented (for reliable asynchronous communication)
  - remote procedure calls (RPC) – the original OO middleware

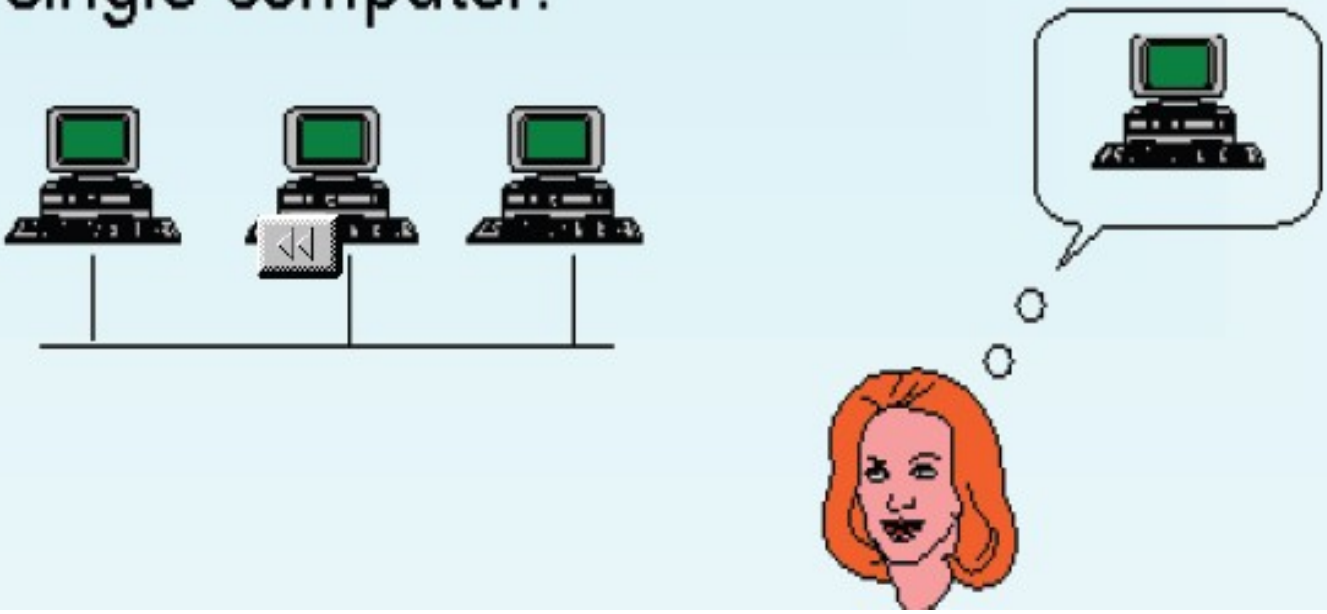
Distributed Operating System (1)



Tightly-coupled operating system for multi-processors and homogeneous multi-computers. Strong transparency.

Distributed Operating System (2)

Goal: An operating system which manages a collection of independent computers and makes them appear to the users of the system as a single computer.



Distributed Operating System (3)

- A **distributed operating system** is the logical aggregation of operating system software over a collection of independent, networked, communicating, and physically separate computational nodes (a distributed system)

- Individual nodes each hold a specific software subset of the global aggregate operating system.

- Each subset is a composite of two distinct service provisioners.
  - *Kernel* - that directly controls that node's hardware.
  - *System management components* - that coordinate the node's individual and collaborative activities. These components abstract microkernel functions and support user applications.

- The collection of kernel and system management components work together to make the distributed systems to appear as a single system image.

DOS: characteristics (1)

- Distributed Operating Systems
  - Allows a **multiprocessor** or **multicomputer** network resources to be integrated as a single system image
  - Hide and manage hardware and software resources
  - provides transparency support
  - provide heterogeneity support
  - control network in most effective way
  - consists of low level commands + local operating systems + distributed features
  - Inter-process communication (IPC)

DOS: characteristics (2)

- remote file and device access
- global addressing and naming
- trading and naming services
- synchronization and deadlock avoidance
- resource allocation and protection
- global resource sharing
- deadlock avoidance
- communication security
- no examples in general use but many research systems: Amoeba, Chorus etc. see Google “distributed systems research”

OS vs DOS

OS

- Manages the local resources to serve the local user

DOS

- Manages the global resources (systems connected in a distributed network) and integrates them in such a way that all the systems acts as a single coherent system.

NOS vs DOS

1. Networking operating system (NOS) also referred to as the Dialogue.
2. Network OS does not reside on every computer, the client only has enough software to boot the hardware and contact the server. All the subsequent operations are done on the server, and the only role of the client is to relay the input and output between the server and the user. Thus, network OS cannot handle computation intensive programs due to the stress it puts in the server.
3. NOS are loosely coupled operating system for heterogeneous multi-computers (LAN & WAN) whose main goal is offer the local services to remote client.

NOS

1. Distributed operating system (DOS) also referred to as the Middleware.
2. DOS is a middleware shared on all system that knows which computers are overloaded and which ones are idle. It would then balance the tasks available so that each computer in the group is sharing equal load. Relatively less hardware and the software resources are required on the user terminals.
3. DOS is tightly-coupled operating system for multi-processors and homogenous multi-computers whose main goal is hide & manage hardware resources.

DOS

Hardware Concepts

Multiprocessors

- **Multiprocessor** is the use of two or more central processing units (CPUs) within a single computer system – ***MIMD***
- The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them
- The term **multiprocessing** may be sometimes used an analogous to *multitasking*, or *multiprogramming*

Multiprocessing Systems

- Generally accepted definition of a multiprocessing/multicomputer system:
  - Multiple processors, each with its own CPU and Memory.
  - Interconnection hardware
  - Processors fail independently
  - There exists a shared state
  - Appears to users as single system

Multicomputer

- A *multicomputer consists of separate computing nodes connected to each other over a network*

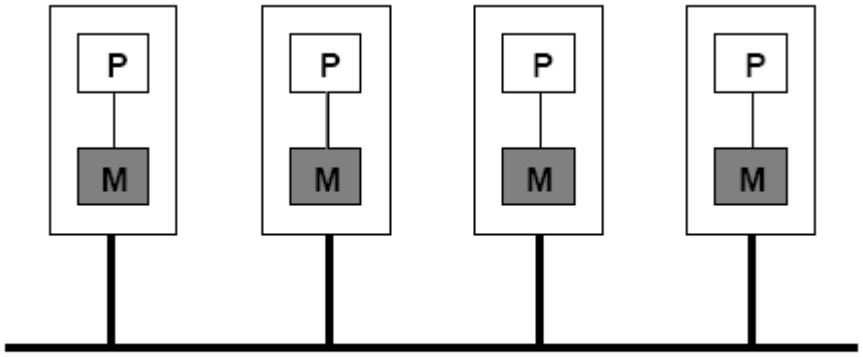


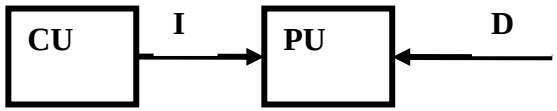
Figure 1: A multicomputer.

- Multicomputer generally differ from each other in three ways:
  - Node resources: processor, memory
  - Network connection: star, ring etc.
  - Homogeneity: same nodes with same physical architecture (processor, system bus, memory)

Flynn's Classification – Processor architectures

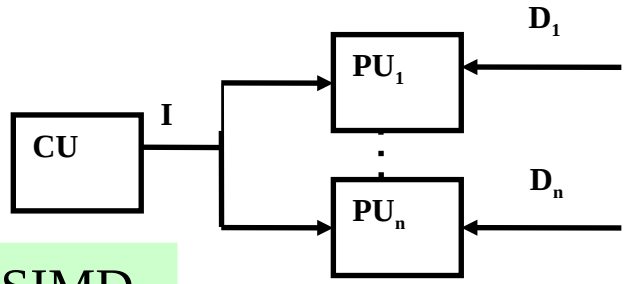
- Flynn, 1966+1972 classification of computer systems in terms of instruction and data stream organizations
- Based on Von-Neumann model (separate processor and memory units
- 4 machine organizations
  - SISD - Single Instruction, Single Data
  - SIMD - Single Instruction, Multiple Data
  - MISD - Multiple Instruction, Single Data
  - MIMD - Multiple Instruction, Multiple Data

Flynn Architectures (1)



SISD

Serial Processor (8080,M6800,i8080)

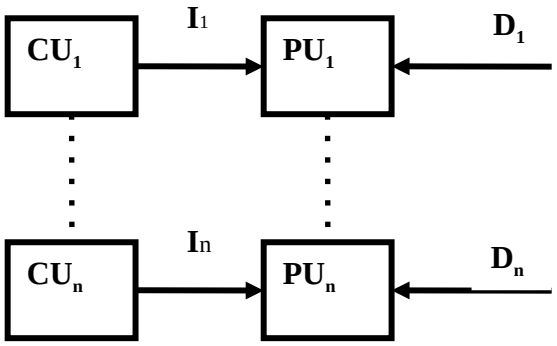


SIMD

Array Processor (Supercomputers, ICL DAP, Thinking Machine CM-1 and CM-2)

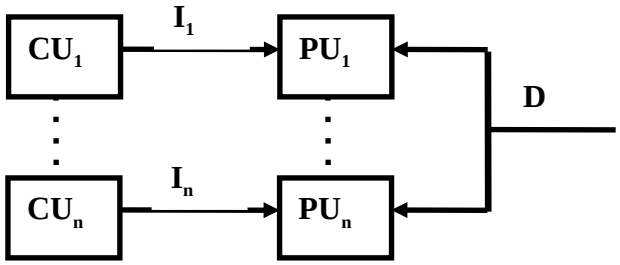
CU – control unit
PU – processor unit
I – instruction stream
D – data stream

Flynn Architectures (2)



MIMD

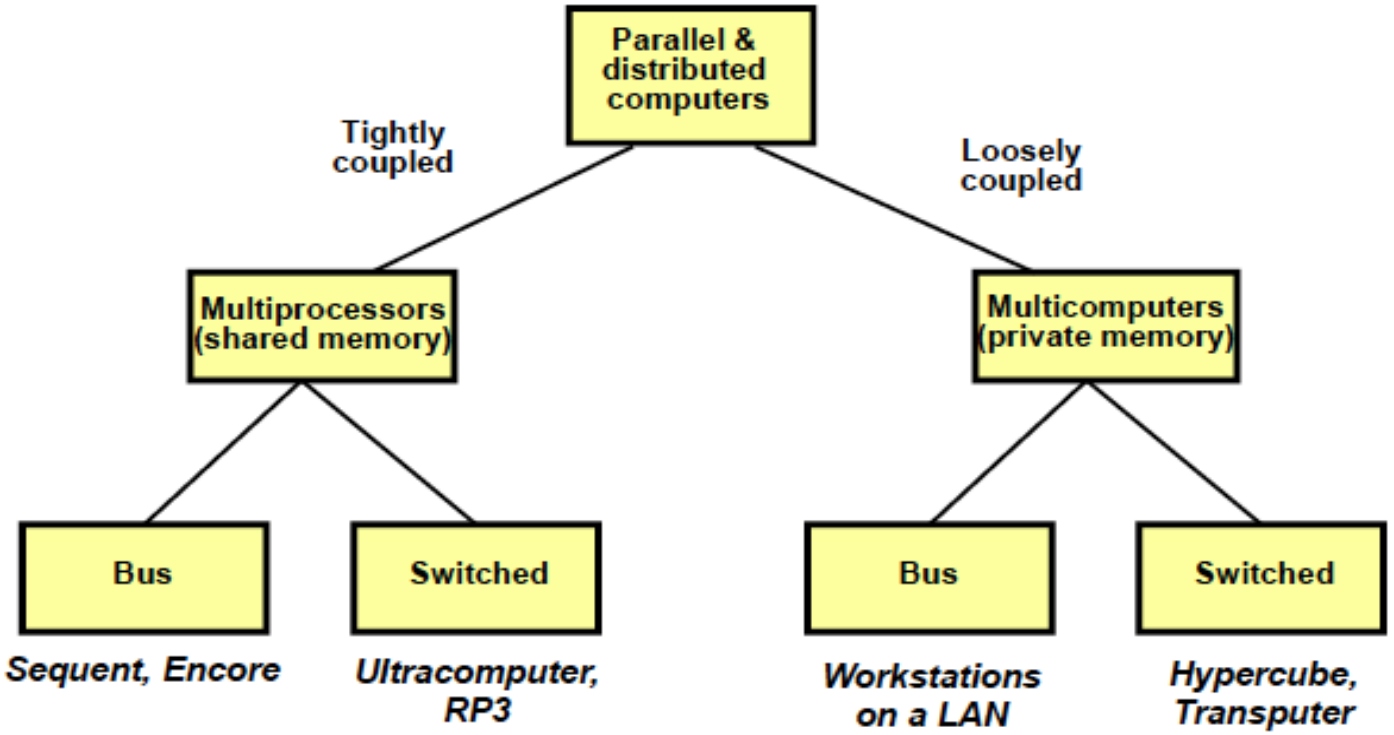
Multiprocesor (All Distributed systems)



MISD

No real examples
- possibly some pipeline architectures (Cray-1, CDC cyber 205, PIC18)

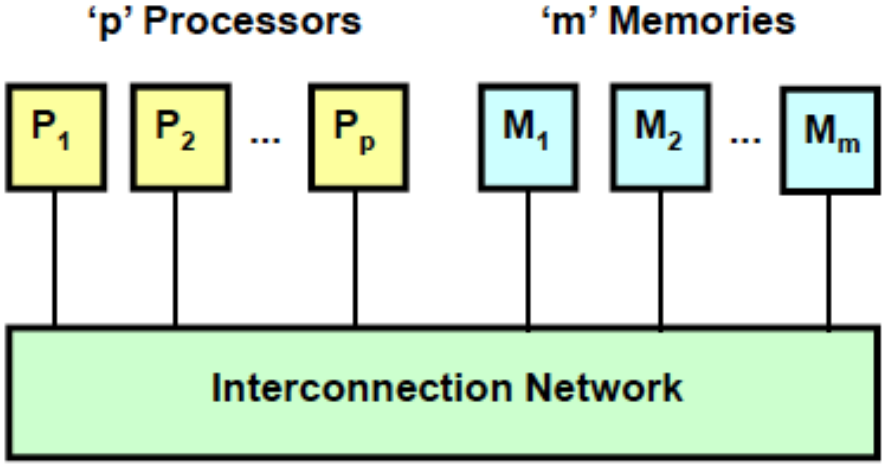
Parallel and Distributed Computers



A taxonomy of parallel & distributed computer systems

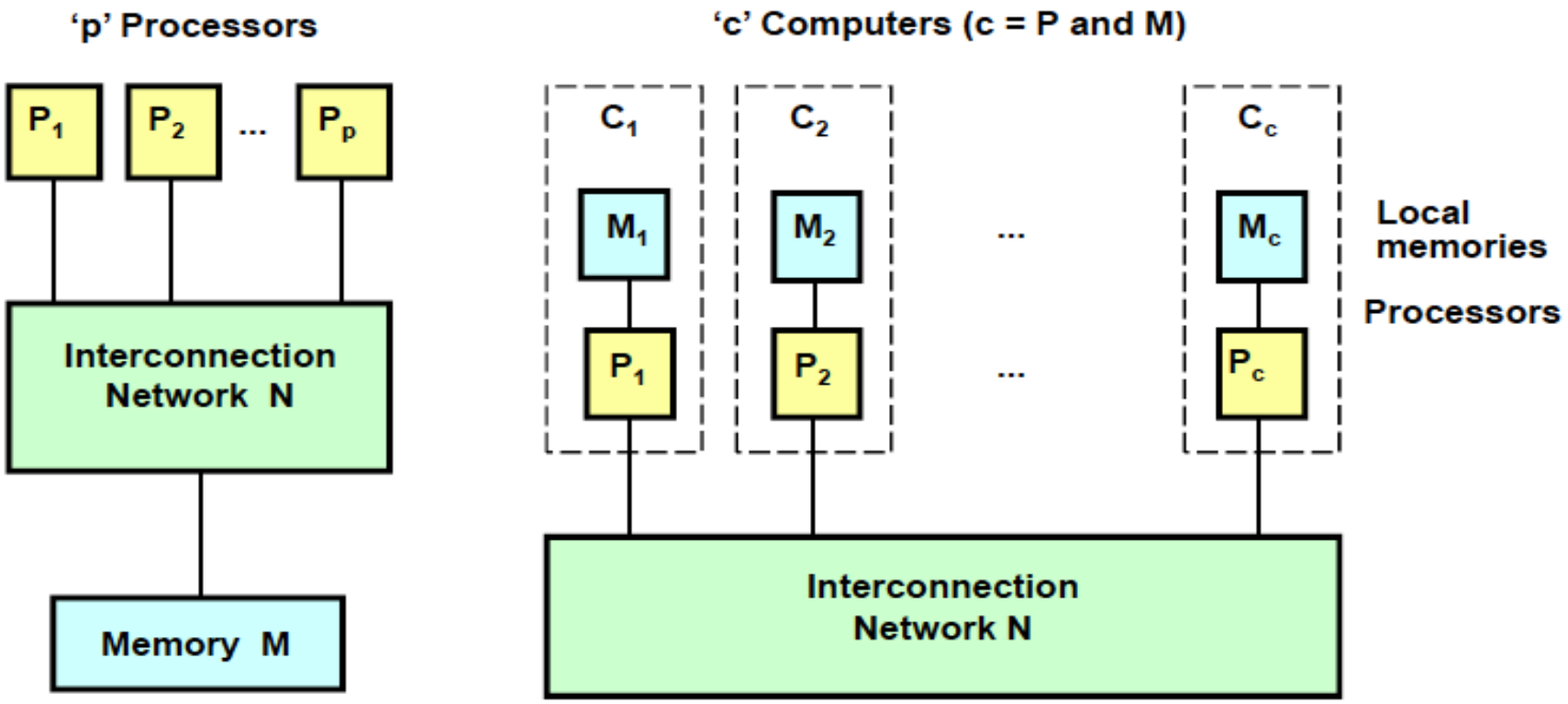
Structural Classification

- Computer system is essentially
  - ‘p’ processing elements = (CPU + registers + cache)
  - ‘m’ memory units
  - joined by an inter-connection network
- Memory may be local to a processor, shared or both



Shared Memory (Multiprocessor)

Distributed Memory (Multicomputer or distributed computer system)



Shared Memory

- If processor A writes 0x55 to its address 2000, then processor B will read 0x55 from *its* address 2000. This is a *multiprocessor*

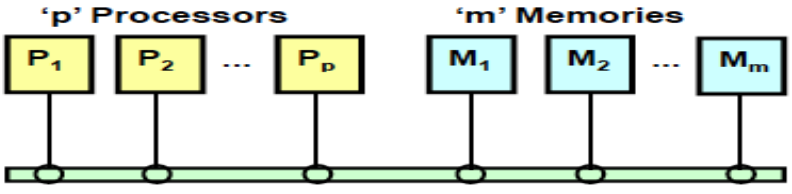
- Obviously, some mechanism is needed to resolve *contention* for the shared resource

Processor-Memory Interconnection Network

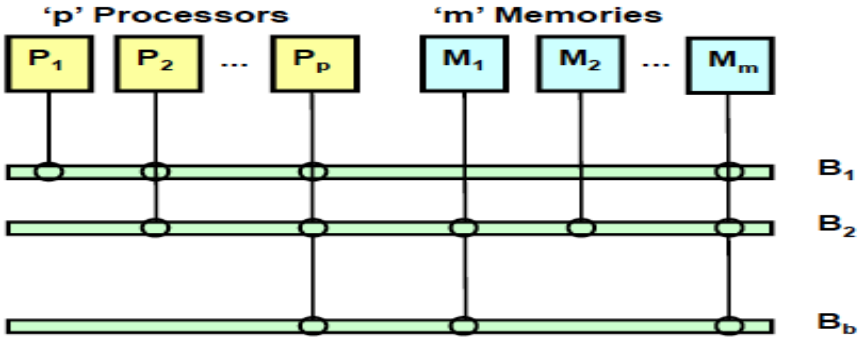
Multiprocessor interconnections may be

- Bussed (time shared)
  - only one bus write at any time
  - must prevent bus contention at the bus interface ports
  - BREQ signals etc
  - limited to about 64 processors
- Switched
  - multiple simultaneous writes
  - requires fast (parallel) bus switches - not cheap!

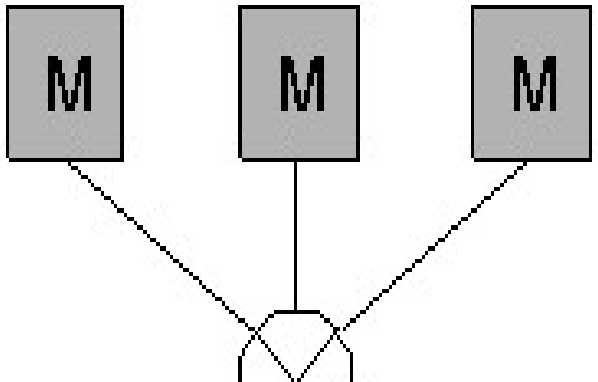
Bussed Systems

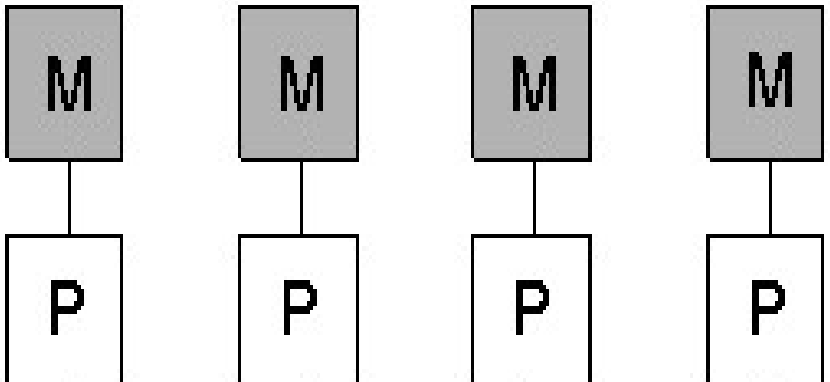


- Single shared bus
  - widely used



- Multiple busses
  - relieve bus contention
  - provides some redundancy





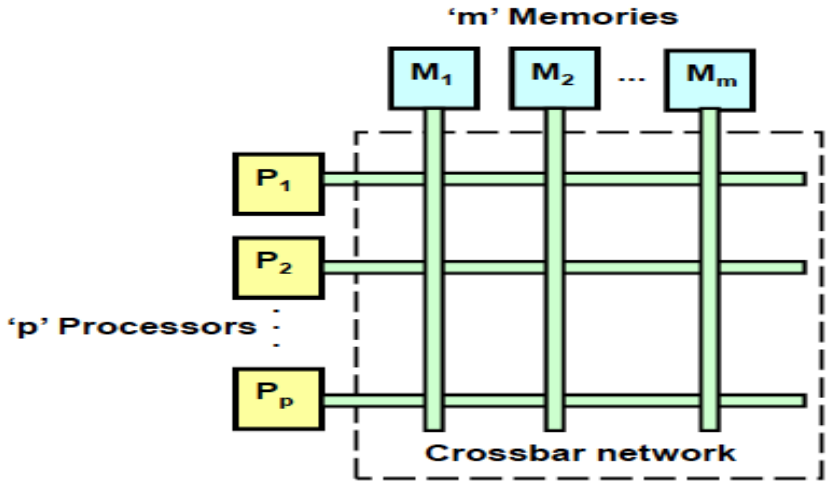
Bus Based Multiprocessor System

- **Drawback:**
  - As few as 4 or 5 CPUs, the bus will be overloaded.
  - Solution: Cache memory.
  - Problem with cache: Incoherent.
  - Solution to incoherent: snoopy write through cache.

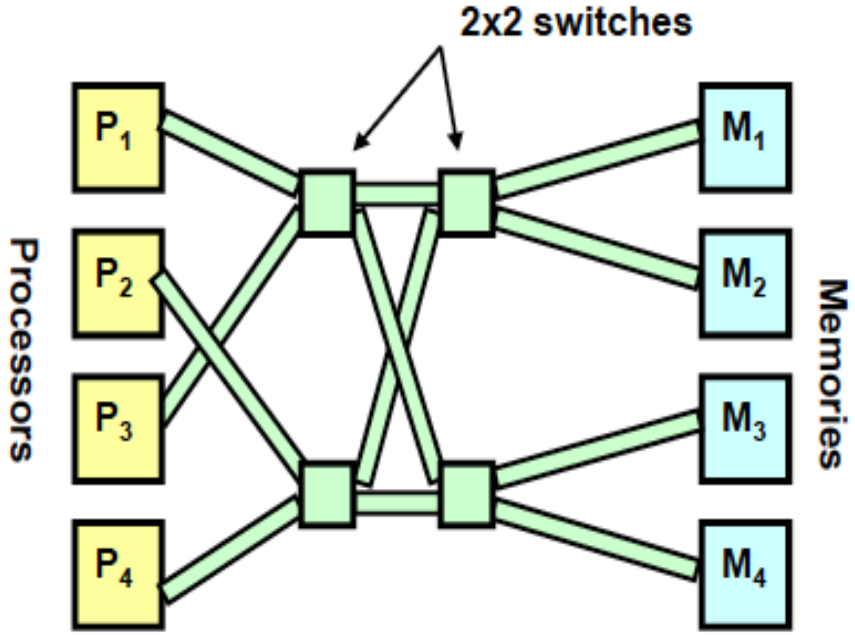
- **Write through:** Whenever a word is written to the cache, it is written through to memory as well.
- **Snoopy cache:** a cache is always“snooping”on the bus. Whenever it sees a write occurring to a memory address present in its cache, it either invalids the entry or updates it.

Switched Systems

- **Crossbar switch**
  - max(m,p) writes at any time
  - requires fast m×p bus switch



- **Crosspoint switch**
  - cheaper but slower!!
  - used in "Omega" networks



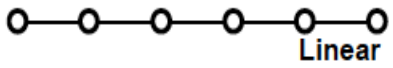
Cross switch

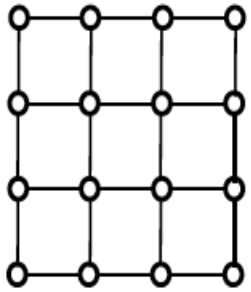
- **Advantage:**
  - Many CPUs can be accessing memory at the same time.
- **Disadvantage:**
  - Need n^2 crosspoints switches with **n** CPUs and **n** memories.
- One solution: use omega network.

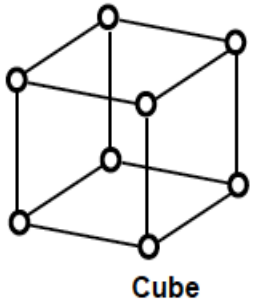
- Interconnections (topology) may be either
  - Static – fixed by hardware
  - Dynamic – re-configurable in software, perhaps even during program execution

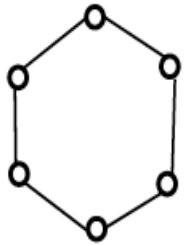
Static Topologies

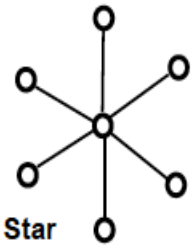
- **Common arrangements are array, ring, star, cube, tree, and complete interconnection of processors.**

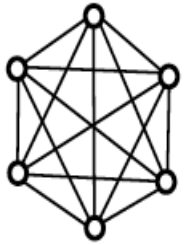


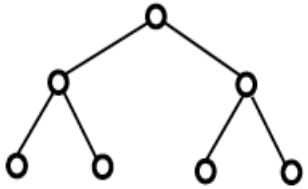








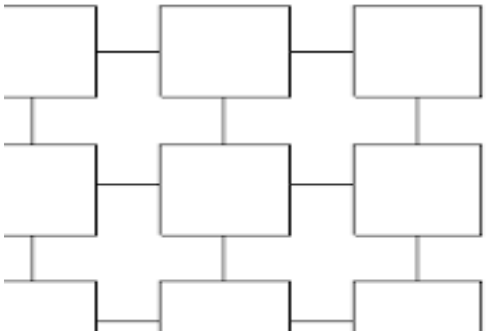




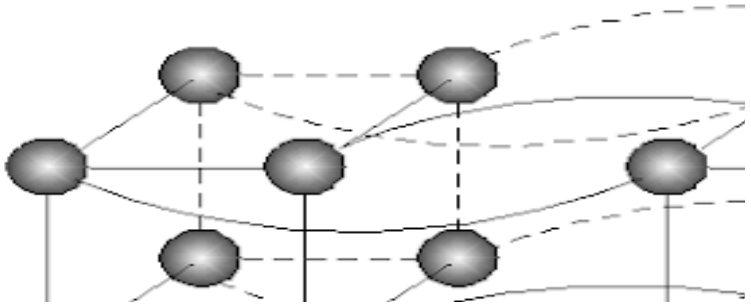
Cube (or hypercube) gives good balance between

- internode length (communications latency)
- number of neighbouring nodes (cost of switching circuitry).

Homogeneous Multicomputer Systems – Processor Arrays



Grid



Hypercube

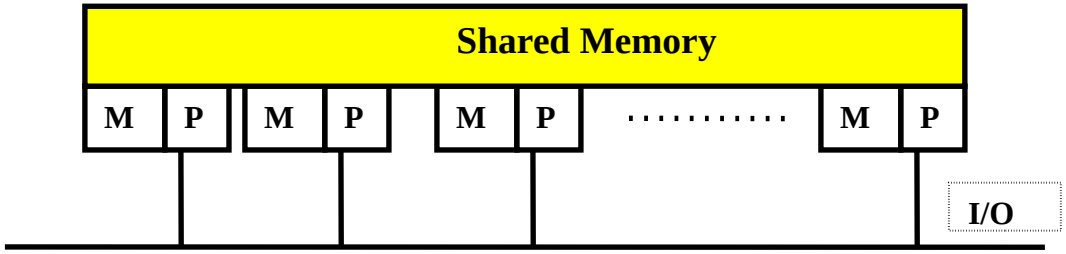
Multiprocessors or Multicomputer Coupling

- **Loosely Coupled**
  - Each processor or computer having a separate memory
- **Tightly Coupled**
  - Each processor or computer **sharing a common memory**
  - Processors may have separate memory units along with a shared memory

Loosely coupled multi-computer systems

- ***Loosely-coupled*** multiprocessor systems (often referred to as clusters) are based on multiple standalone single or dual processor computers interconnected via a high speed communication system
- A Linux Beowulf cluster is an example of a loosely-coupled system.
- **Distributed Memory Multi-computer or multiprocessor**
- **IPC by message passing**
- **Typically PC or workstation clusters**
- **Physically distributed components**
- **Characterized by longer message delays and limited bandwidth**

Closely (Tightly) coupled multi-computer systems



- ***Tightly-coupled*** multiprocessor systems contain multiple CPUs that are connected at the bus level.
- E.g. Chip multiprocessors, also known as multi-core computing, involves more than one processor placed on a single chip and can be thought of the most extreme form of tightly-coupled multiprocessing
- These CPUs may have access to a central shared memory (SMP or UMA), or may participate in a memory hierarchy with both local and shared memory (NUMA).
- **IPC via Shared Memory**

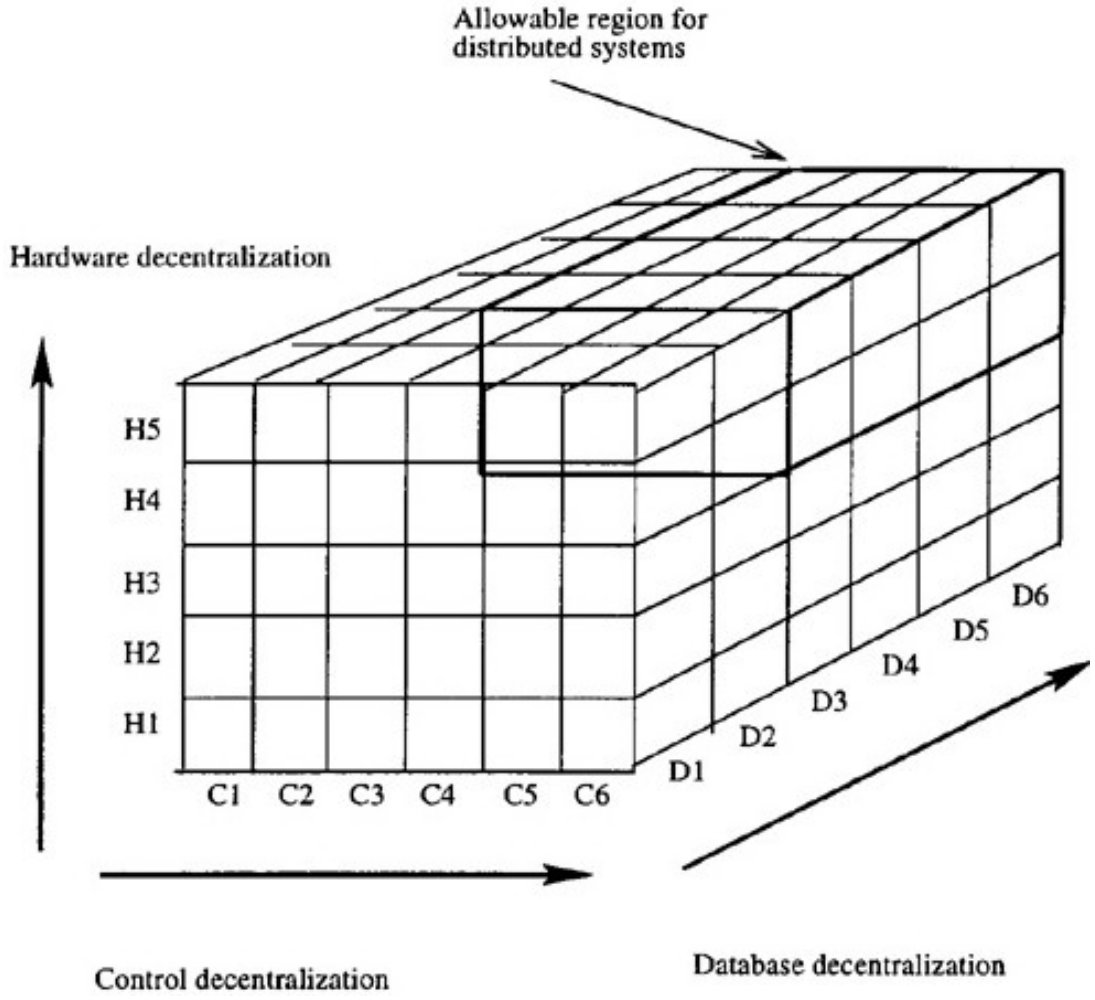
Tightly Coupled vs Loosely Coupled

- **Performance:** Tightly-coupled systems perform better
- **Size**: Tightly-coupled systems are physically smaller than loosely-coupled systems
- **Cost:** Nodes in a loosely-coupled system are usually inexpensive than a tightly coupled system
- **Power consumption:** Tightly-coupled systems tend to be much more energy efficient than clusters.

Enslow's Model of DS

- Enslow (Scientist) proposed that distributed systems can be examined using three dimensions of hardware, control, and data.

- Distributed system = distributed hardware + distributed control + distributed data



Enslow's model of distributed systems

(Some researchers also considered computer networks and parallel computers as part of distributed system)

- a system can be classified as a distributed system if all three categories (hardware, control, and data) reach a certain degree of decentralization.

- Several points in the dimension of hardware organization are as follows:
  - H1. A single CPU with one control unit.
  - H2. A single CPU with multiple ALUs (arithmetic and logic units). There is only one control unit

- H3. Separate specialized functional units, such as one CPU with one floating-point coprocessor.
- H4. Multiprocessors with multiple CPUs but only one single I/O system and one global memory.
- H5. Multicomputers with multiple CPUs, multiple I/O systems and local memories.

- Similarly, points in the control dimension in order of increasing decentralization are the following:
  - C1. Single fixed control point. Note that physically the system may or may not have multiple CPUs.

- C2. Single dynamic control point. In multiple CPU cases the controller changes from time to time among CPUs.
- C3. A fixed master/slave structure. For example, in a system with one CPU and one coprocessor, the CPU is a fixed master and the coprocessor is a fixed slave.
- C4. A dynamic master/slave structure. The role of master/slave is modifiable by software.
- C5. Multiple homogeneous control points where copies of the same controller are used.
- C6. Multiple heterogeneous control points where different controllers are used

- The database has two components that can be distributed: files and a directory that keeps track of these files

- Distribution can be done in one of two ways, or a combination of both: replication and partition

- A database is partitioned if it is split into sub-databases and then each sub-database is assigned to different sites

- D1. Centralized databases with a single copy of both files and directory.
- D2. Distributed files with a single centralized directory and no local directory.
- D3. Replicated database with a copy of files and a directory at each site.

- D4. Partitioned database with a master that keeps a complete duplicate copy of all files.
- D5. Partitioned database with a master that keeps only a complete directory.
- D6. Partitioned database with no master file or directory.

- A system is a distributed one if it has:
  - Multiple processing elements (PEs).
  - Interconnection hardware.
  - Shared states.