

Distributed Systems



Dr. Sudhir N. Dhage
Professor
Department of Computer Engineering
Sardar Patel Institute of Technology,
Andheri, Mumbai.
Email: sudhir_dhage@spit.ac.in

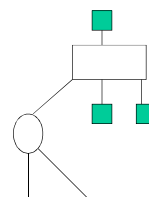
CONTENTS

1. DEFINITION
2. GOALS
3. HARDWARE CONCEPTS
4. SOFTWARE CONCEPTS
5. THE CLIENT-SERVER MODEL

Distributed Computing System Models

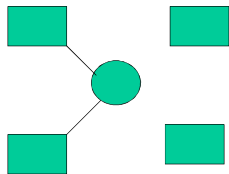
- Minicomputer Model
- Workstation Model
- Workstation Server Model
- Processor Model
- Hybrid Model

Minicomputer Model



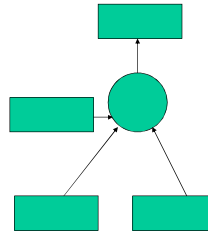
- Extension of centralized Time Sharing Model.
- Resource Sharing.
- Minicomputer has evolved into the mid-range server and a part of a N/W.
- AS-400

Workstation Model



- Several W/S are interconnected by High speed N/W.
- Effective use of Computational Power.

Workstation Server Model



- Interconnection of few server class machine with large number of workstations.
- Diskless Terminals are allowed.
- E.g.. Novell NetWare

Processor Pool Model

- No concept of home machine.
- Better utilization of processing power as compared to W/S model.

Hybrid Model

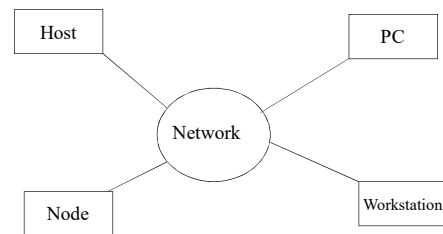
- A combination of two or more online marketing payment models.

DEFINITION

DISTRIBUTED SYSTEM

A collection of independent computers that appears to its users as a single coherent system.

Definition



What is a distributed system?

A very broad definition:

– *A set of autonomous processes communicating among themselves to perform a task*

Issues:

- Un-reliability of communication
- Lack of global knowledge
- Lack of synchronization and causal ordering
- Concurrency control
- Failure and recovery

Major Advantages

- Resource Sharing
- Higher Performance
- Fault Tolerance
- Scalability

The Rise of Distributed Systems

- Computer hardware prices are falling and power increasing.
- Network connectivity is increasing.
 - Everyone is connected with fat pipes.
- It is *easy* to connect hardware together.

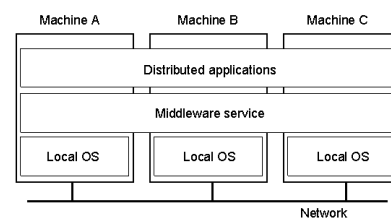
Why Distributed Systems are becoming Popular?



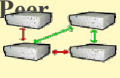
- Inherently distributed applications
- Information sharing among distributed users
- Resource sharing
- Better price performance
- Shorter response time higher throughput
- Extensible and incremental growth
- Better flexibility in meeting users' needs

The main features of a distributed system

- **Functional Separation**
- **Inherent distribution**
- **Reliability**
- **Scalability**
- **Economy**

MIDDLEWARE



Types of Distributed Systems			
Clusters  <p>Groups of PCs (ordinary or specialized) brought specifically together to work collectively on problems.</p>	Grids  <p>Clusters can be combined to form a "Grid", a system of massive collective computing power which is designed to be easily used by "plugging in" to it.</p>	Peer 2  <p>A system whereby individual users or nodes can communicate with each other by themselves. Examples of such a system would be Napster.</p>	Others <p>The WWW is a distributed system! (of information). It is actually peer to peer, but is worth mentioning separately as a good example.</p> <p>CORBA can be used to create a distributed system of programming objects, almost like a distributed developer system</p>

GOALS

- CONNECTING USERS AND RESOURCES
- TRANSPARENCY
- OPENNESS
- SCALABILITY

CONNECTING USERS AND RESOURCES

- ✓ Easy to access remote resources
- ✓ Share resources with other users in controlled way
- ✓ Easy to collaborate and exchange information

TRANSPARENCY

Transparency is the one in which certain aspects of distribution are made invisible to the application programmer

DIFFERENT FORMS OF TRANSPARENCY:

- Access
- Location
- Migration
- Relocation
- Replication
- Concurrency
- Failure
- Persistence

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 is replicated
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

OPENNESS

An open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services

SCALABILITY

A system is said to be scalable if it remains effective when there is a significant increase in the number of resources and the number of users

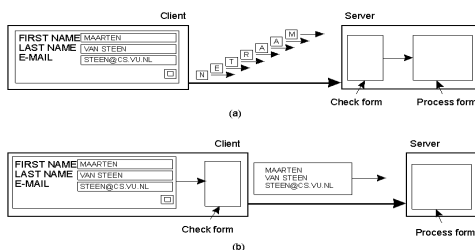
SCALABILITY CHALLENGES

- ✓ Controlling the cost of physical resources
- ✓ Controlling the performance loss
- ✓ Preventing software resources running out

SCALING TECHNIQUES

- ASYNCHRONOUS COMMUNICATION
- DISTRIBUTION
- REPLICATION
- CACHING

ASYNCHRONOUS COMMUNICATION

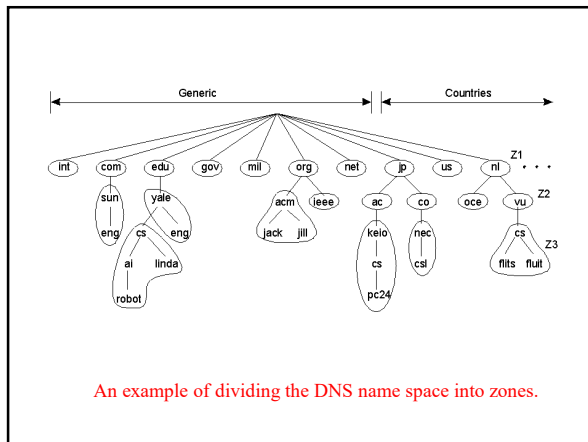


The difference between letting:

- a) a server or
- b) a client check forms as they are being filled

DISTRIBUTION

It involves taking a component, splitting it into smaller parts and subsequently spreading those parts across the system



REPLICATION

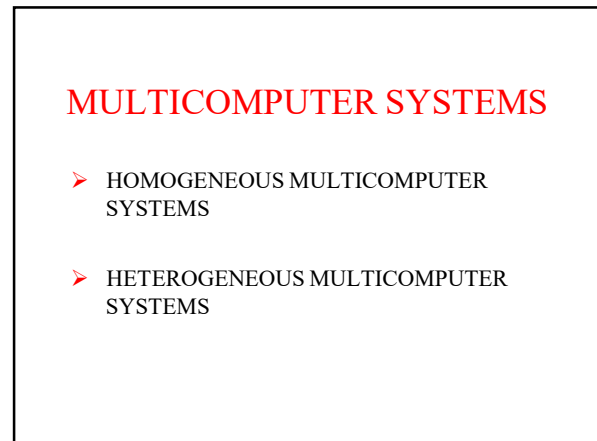
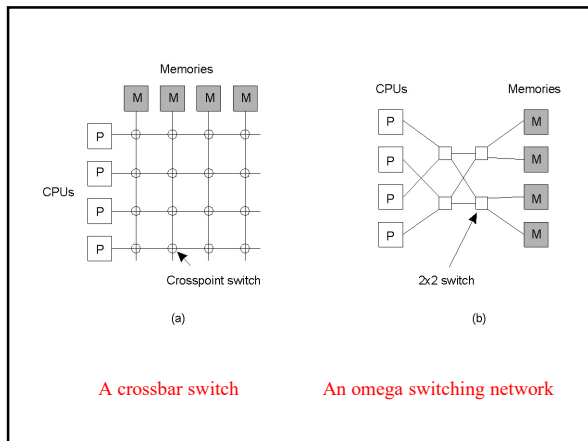
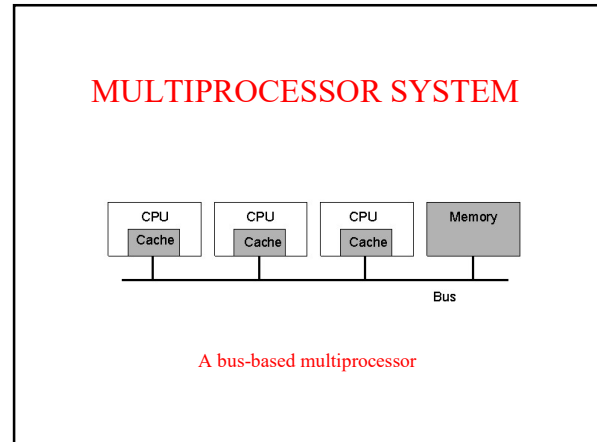
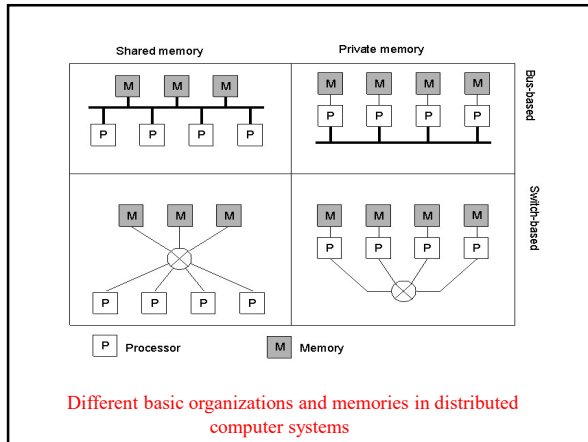
In this method, the components are replicated across the distributed system and are generally placed in the proximity of the client accessing the resource.

CACHING

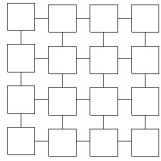
It works on the same concept as replication but in this case the decision to replicate the data is made by the client and not by the owner of the resource.

HARDWARE CONCEPTS

- BUS BASED ORGANISATION
- SWITCH BASED ORGANISATION

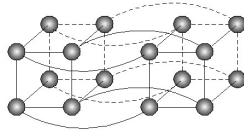


HOMOGENEOUS MULTICOMPUTER SYSTEMS



(a)

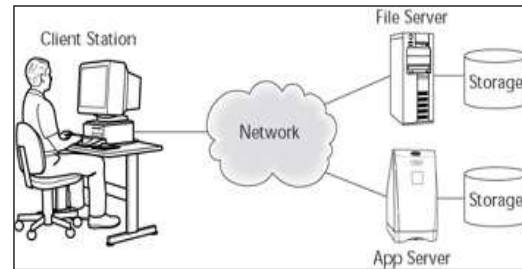
Grid



(b)

Hypercube

HETEROGENEOUS MULTICOMPUTER SYSTEMS



SOFTWARE CONCEPTS

- DISTRIBUTED OPERATING SYSTEM
- NETWORK OPERATING SYSTEM
- MIDDLEWARE

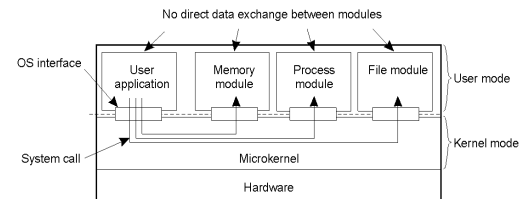
OVERVIEW

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

DISTRIBUTED OPERATING SYSTEM

- UNIPROCESSOR OPERATING SYSTEM
- MULTIPROCESSOR OPERATING SYSTEM
- MULTICOMPUTER OPERATING SYSTEM
- DISTRIBUTED SHARED MEMORY ACCESS

UNIPROCESSOR OPERATING SYSTEM

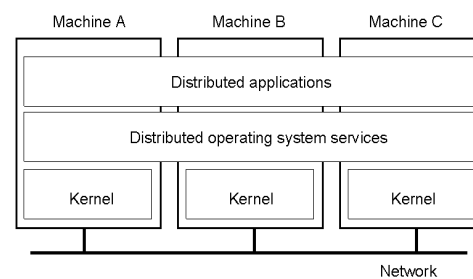


Separating applications from operating system code through a microkernel

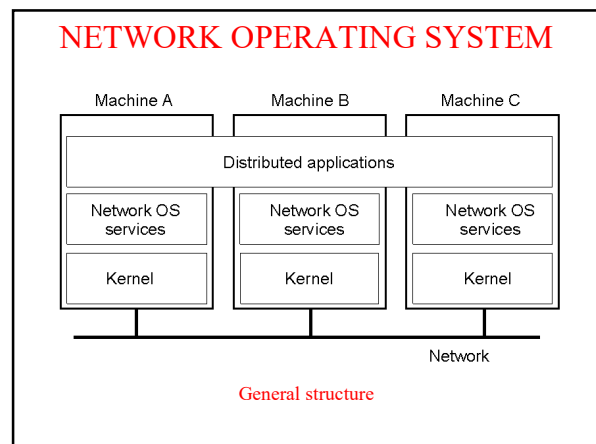
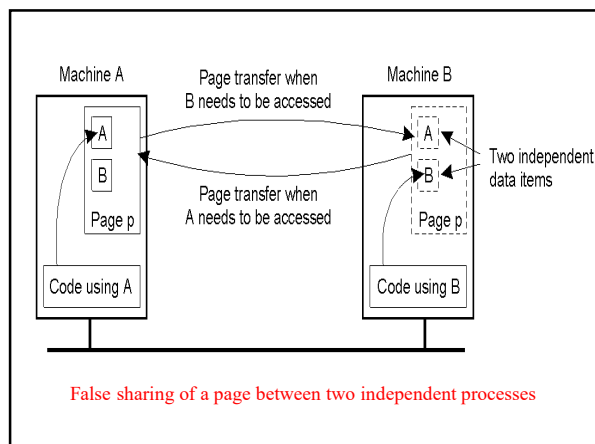
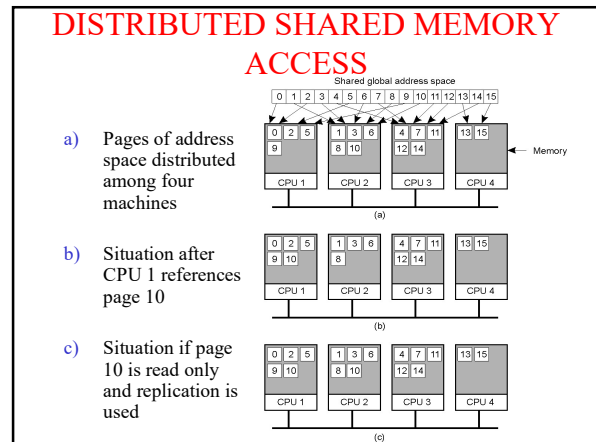
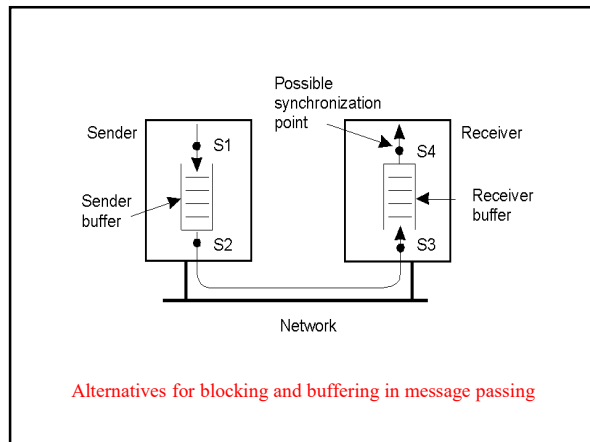
MULTIPROCESSOR OPERATING SYSTEM

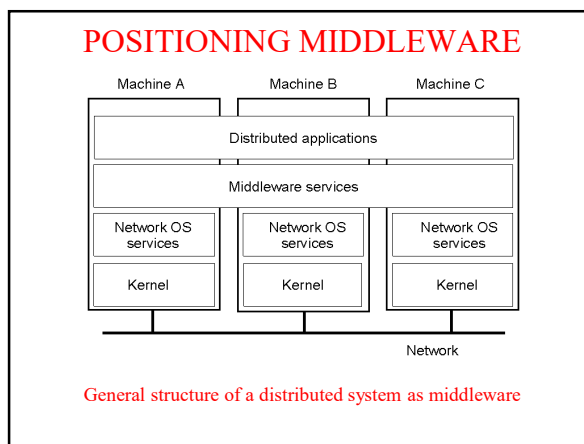
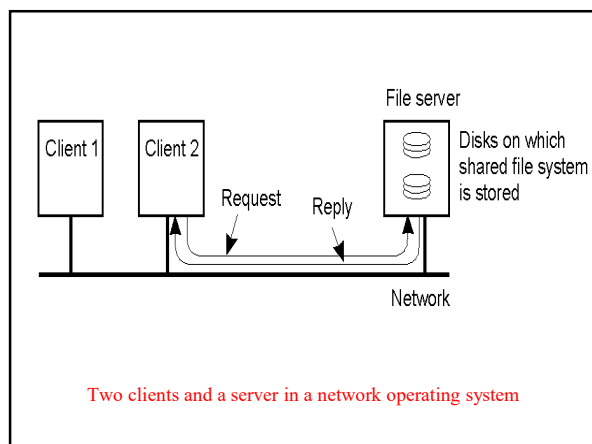
- SEMAPHORE
can be thought of as an integer with two operations down and up
- MONITOR
can be thought of as a module consisting of variables and procedures

MULTICOMPUTER OPERATING SYSTEM



General structure





Models of Middleware

1. File System Model

-Basically used by Unix, Unix treated each H/W as a file, Treating each m/c over the N/W as a file.
 -Writing and reading from any m/c is reading and writing the data from file.

2. RPC Model

In this, you call a procedure at remote location from your current m/c

3. Distributed File system model

-Using Distributed file system like CODA or NFS

4. Distributed Object model

-object is divided into two parts Implementation and Interface

5. Distributed Document model

-achieved through linking

Services Offered by Middleware

1. Access Transparency

-How data is hidden

2. Naming

Service used to locate specific resource over the system

3. Persistence

-Giving permanent storage to each resource also known as persistence

4. Distributed Transaction

-Allowing use of a resource located on a remote machine from a local machine.

5. Security

-Protect resource with the help security mechanism.

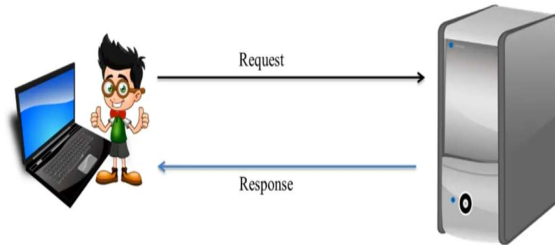
Comparison between Systems

- A comparison between multiprocessor operating systems, multicomputer operating systems, network operating systems, and middleware based distributed systems.

Item	Distributed OS		Network OS	Middleware-based OS
	Hydros	Hydromp		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basic for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Client	Open	Open

Client – Server Architecture

2-Tier Architecture – A 2 Tier Architecture is where a client talks directly to a server



Client – Server Architecture

3-Tier Architecture – A middleware is used.

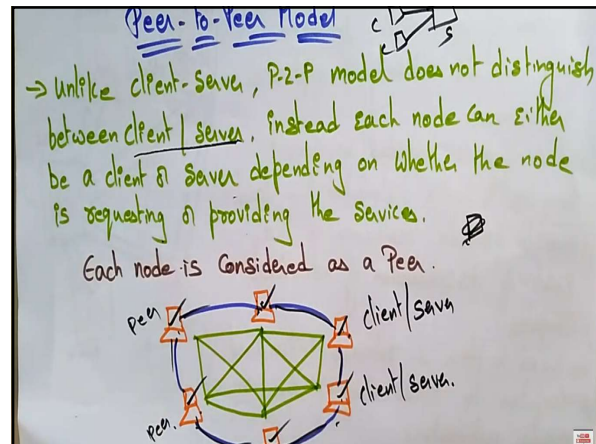
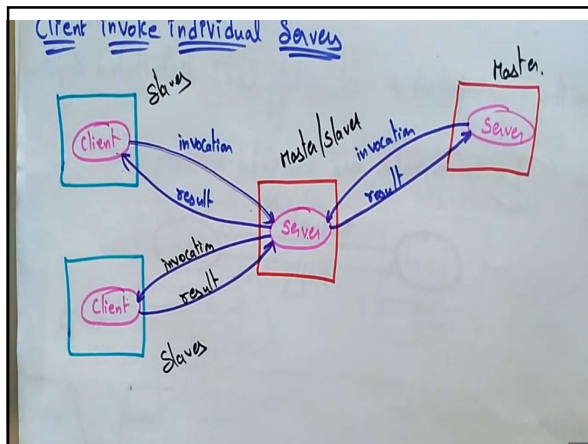
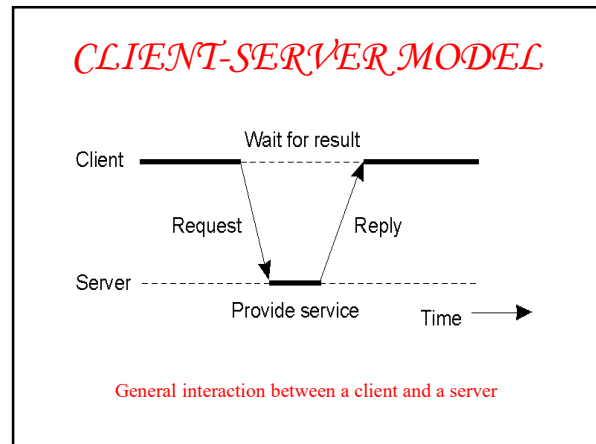
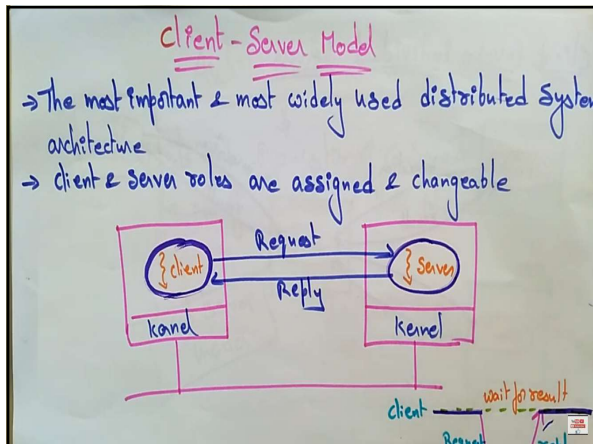


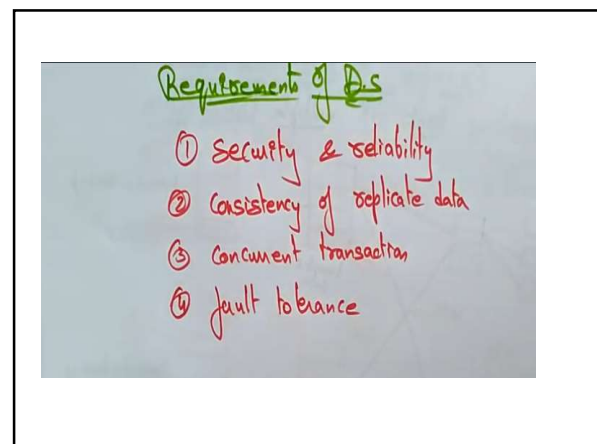
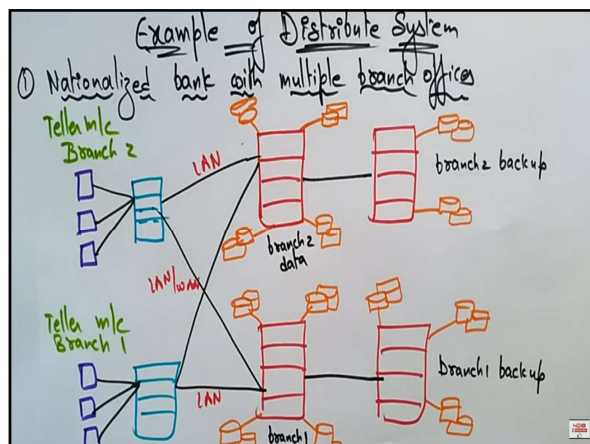
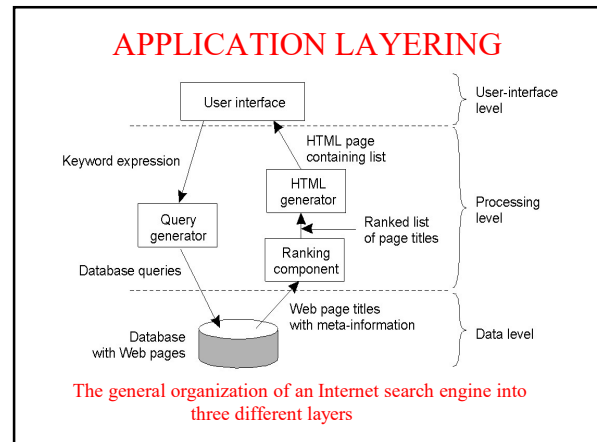
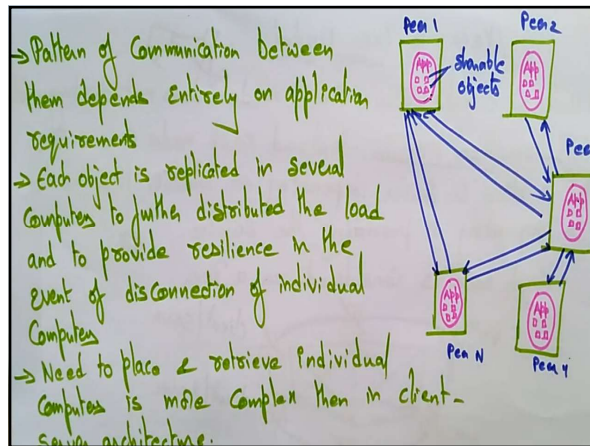
Architectural Model

Deals with organization of components across the net of computers & their interrelationship.

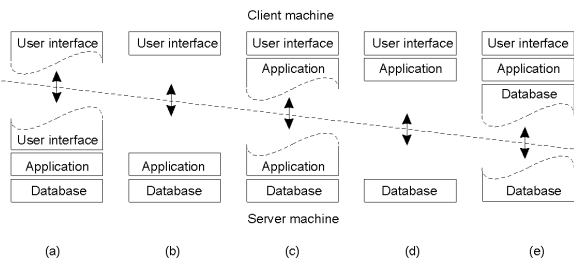
client server model

Peer-Peer model.



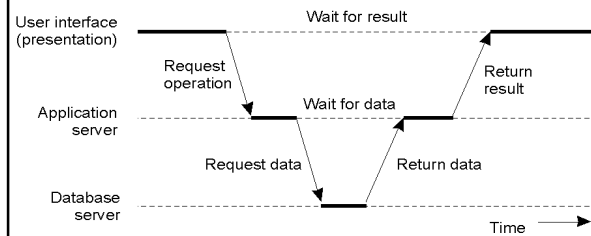


MULTITIERED ARCHITECTURES



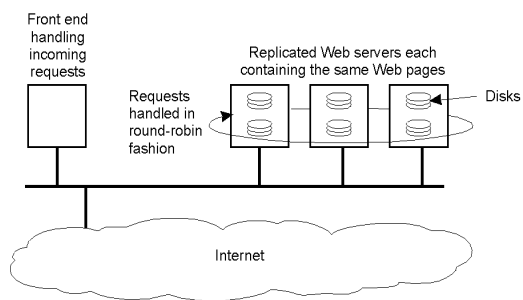
Alternative client-server organizations (a) – (e)

THREE-TIERED ARCHITECTURE



An example of a server acting as a client

MODERN ARCHITECTURE



An example of horizontal distribution of a Web service

Why is it hard to design them?

- The usual problem of concurrent systems:
 - Arbitrary interleaving of actions makes the system hard to verify
- +
- No globally shared memory (therefore hard to collect global state)
- No global clock
- Unpredictable communication delays

Models for Distributed Algorithms

- Topology: Completely connected, Ring, Tree etc.
- Communication: Shared memory / Message passing (reliable? Delay? FIFO/Causal? Broadcast/multicast?)
- Synchronous/asynchronous
- Failure models: Fail stop, Crash, Omission, Byzantine...
- *An algorithm needs to specify the model on which it is supposed to work*

Complexity Measures

- Message complexity: no. of messages
- Communication complexity / Bit Complexity: no. of bits
- Time complexity:
 - For synchronous systems, no. of rounds
 - For asynchronous systems, different definitions are there.

Some Fundamental Problems

- Ordering events in the absence of a global clock
- Capturing the global state
- Mutual exclusion
- Leader election
- Clock synchronization
- Termination detection
- Constructing spanning trees
- Agreement protocols

An experimental file server is up 75% of the time and down for 25% of the time due to bugs. How many times does this file server have to be replicated to give an availability of at least 99% ?

- A) 2
- B) 4
- C) 8
- D) 16

Lets us check each option

a) if there are 2 replications, probability of failing both at same time = $.25 * .25 = .0625 = 6.25\%$

Availiability = $100 - 6.25 = 93\%$ //Not the answer

b) In case of 4 replications probability of failing all at same time = $(0.25)^4 = 1.00 = 1\%$

Availiability = $100 - 1 = 99\%$ //Hence the answer

No need to check furthur as all options are more than 4

An experimental file server is up 3/4 of the time and down 1/4 of the time due to bugs. How many times does this fileserver have to be replicated to give an availability of atleast 99%?

Availability, A, is 0.99; thus, the server can be down with probability $1 - A = 0.01$. The probability of server failure, Pf, is 0.25. Thus, we are looking for the number of servers, n, such that $(0.25)^n < 0.01$. That is, the probability that all n servers will be down must be below $1 - A$. Here are some possibilities, which show that four server replicas are necessary.

Number of Servers (n) Probability All n Servers Are Down ((Pf)ⁿ)

1	0.250
2	0.063
3	0.016
4	0.004

Message Passing

In a distributed system, processes executing on different omputers often need to communicate with each other to achieve some common goal. Interprocess communication basically requires information sharing among two or more processes.

The two basic methods for information sharing are as follows:

1. Original sharing, or shared-data approach
2. Copy sharing, or message-passing approach

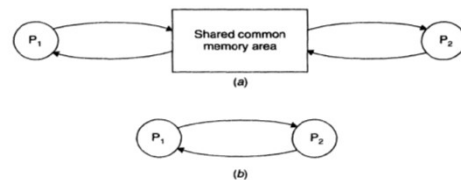


Fig. The two basic interprocess communication paradigms: (a) The shared-data approach. (b) The message-passing approach.

DESIRABLE FEATURES OF A GOOD MESSAGE-PASSING SYSTEM

1. **Simplicity**
2. **Uniform Semantics**
3. **Efficiency**
4. **Reliability**
5. **Correctness**
6. **Flexibility**
7. **Security**
8. **Portability**

ISSUES IN IPC BY MESSAGE PASSING

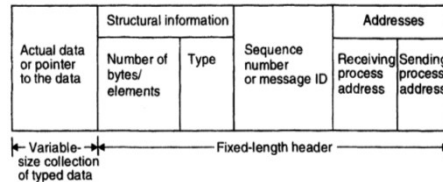


Fig. A typical message structure.

In the design of an IPC protocol for a message-passing system, the following important issues need to be considered:

- Who is the sender?
- Who is the receiver?
- Is there one receiver or many receivers?
- Is the message guaranteed to have been accepted by its receiver(s)?
- Does the sender need to wait for a reply?
- What should be done if a catastrophic event such as a node crash or a communication link failure occurs during the course of communication?
- What should be done if the receiver is not ready to accept the message: Will the message be discarded or stored in a buffer? In the case of buffering, what should be done if the buffer is full?
- If there are several outstanding messages for a receiver, can it choose the order in which to service the outstanding messages?

DESIGN ISSUES IN IPC BY MESSAGE PASSING

- SYNCHRONIZATION
 - BLOCKING PRIMITIVE (SYNCHRONOUS) : If its invocation blocks execution of its invoker
 - NON-BLOCKING (ASYNCHRONOUS) : Does not block execution
- 2 cases
 - 1st case
 - Blocking send
 - Blocking Rec
 - 2nd case
 - Non-Blocking Send
 - Non-Blocking Rec

Thanks

?