19Z603-DISTRIBUTED COMPUTING

SYLLABUS

INTRODUCTION, MESSAGE PASSING AND RPC : Definition - System models - Design issues of distributed operating systems - Message Passing: Features and Issues–Buffering - Process addressing - Failure handling RPC: Model - Implementation - Stub generation - RPC Messages - Marshaling - Server management – Call semantics (10)

SYNCHRONIZATION : Clock synchronization - Physical clocks - Logical clocks - Election algorithms – Mutual exclusion- Deadlocks (8)

PROCESS AND RESOURCE MANAGEMENT: Process migration: Features - Mechanism. Resource Management: Load balancing approach - Load sharing approach (9)

CLOUD AS A DISTRIBUTED ENVIRONMENT : The Vision of Cloud Computing - Defining a Cloud - Historical Developments -Cloud Computing Reference Model –Cloud Deployment Models - Public, Private, Community, Hybrid Clouds - Cloud Delivery Models - IaaS, PaaS, SaaS - Characteristics and Benefits - Challenges. (8)

CLOUD TECHNOLOGIES : Technologies for Infrastructure as a service - Platform as a Service– Software as a service – Cloud Storage: MapReduce, GFS, HDFS - Cloud container: Docker. (10)

TEXT BOOKS AND REFERENCES

TEXT BOOKS:

1. Pradeep K Sinha , "Distributed Operating Systems: Concepts and Design", Prentice Hall of India, New Delhi, 2009.
2. Rajkumar Buyya, Christian Vecchiola, Thamarai Selvi S , "Mastering Cloud Computing", Tata McGraw Hill Education Private Limited,, New Delhi, 2013.

REFERENCES:

1. Andrew S Tanenbaum, Marteen Van Steen , "Distributed Systems Principles and Paradigms", Pearson Education / Prentice Hall of India, New Delhi, 2007.
2. George Coulouris, Jean Dollimore , "Distributed Systems Concept and Design", Pearson Education, New Delhi, 2006.
3. David S Linthicium , "Cloud Computing and SOA Convergence in your Enterprise", Pearson, USA, 2010.
4. Sébastien Goasguen , "Docker in the Cloud -Recipes for AWS, Azure, Google, and More", O'Reilly Media, USA, 2016.

COURSE OUTCOMES

CO1: Depict the issues, models of distributed systems and Describe message passing and develop RPC based client-server programs

CO2:Analyze and apply synchronization, deadlock management techniques

CO3:Paraphrase, apply process management and resource management in distributed systems

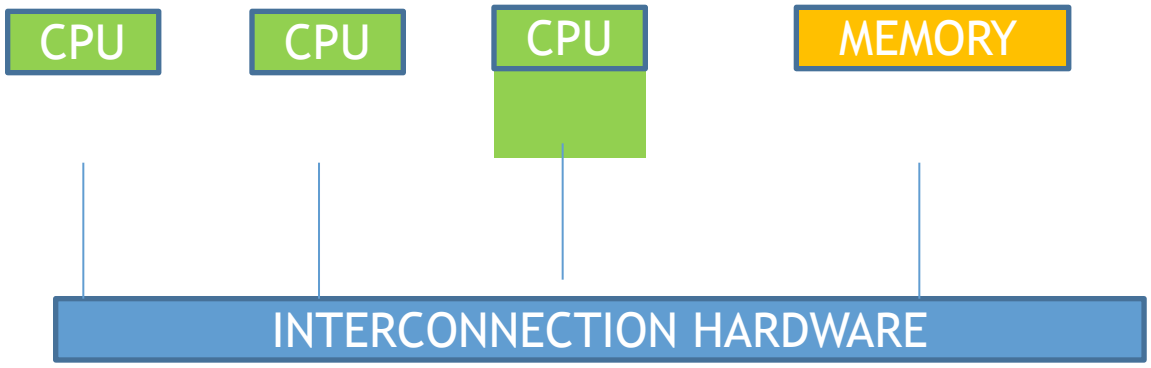
CO4:Understand the concepts, key technologies and strengths of cloud computing

CO5: Explore different cloud programming tools and technologies and deploy cloud based application

Computer Architecture

Multiprocessors

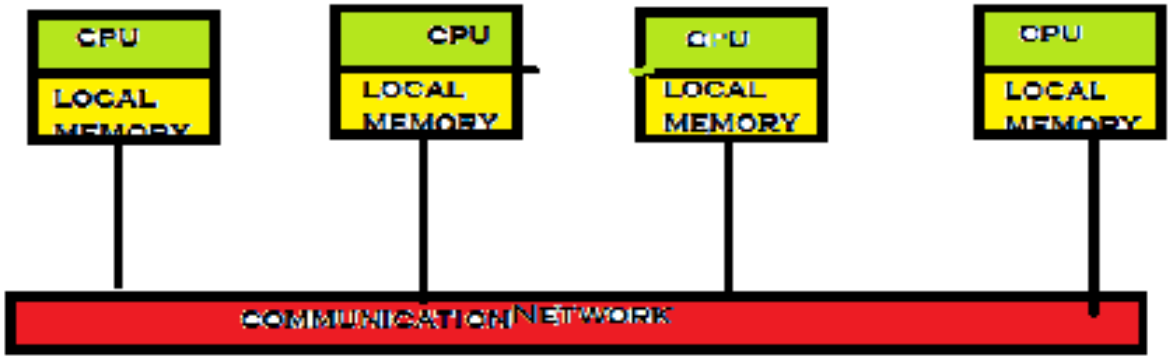
1. Tightly Coupled Systems(Parallel Processing Systems)
   - Single System Memory Shared by all Processors
   - Small and limited bandwidth



Computer Architecture

2. Loosely Coupled Systems(Distributed Computing Systems)

- Processors have their own local memory and communicate between them
- Expandable and Unlimited no of processor



Distributed Computing System

A distributed computing system is a collection of processors connected by a communication network, where each processor has its own local memory and other peripherals. Communication between any two processors in the system occurs through message passing over the communication network.

Evolution of Distributed Computing Systems

Early Computers

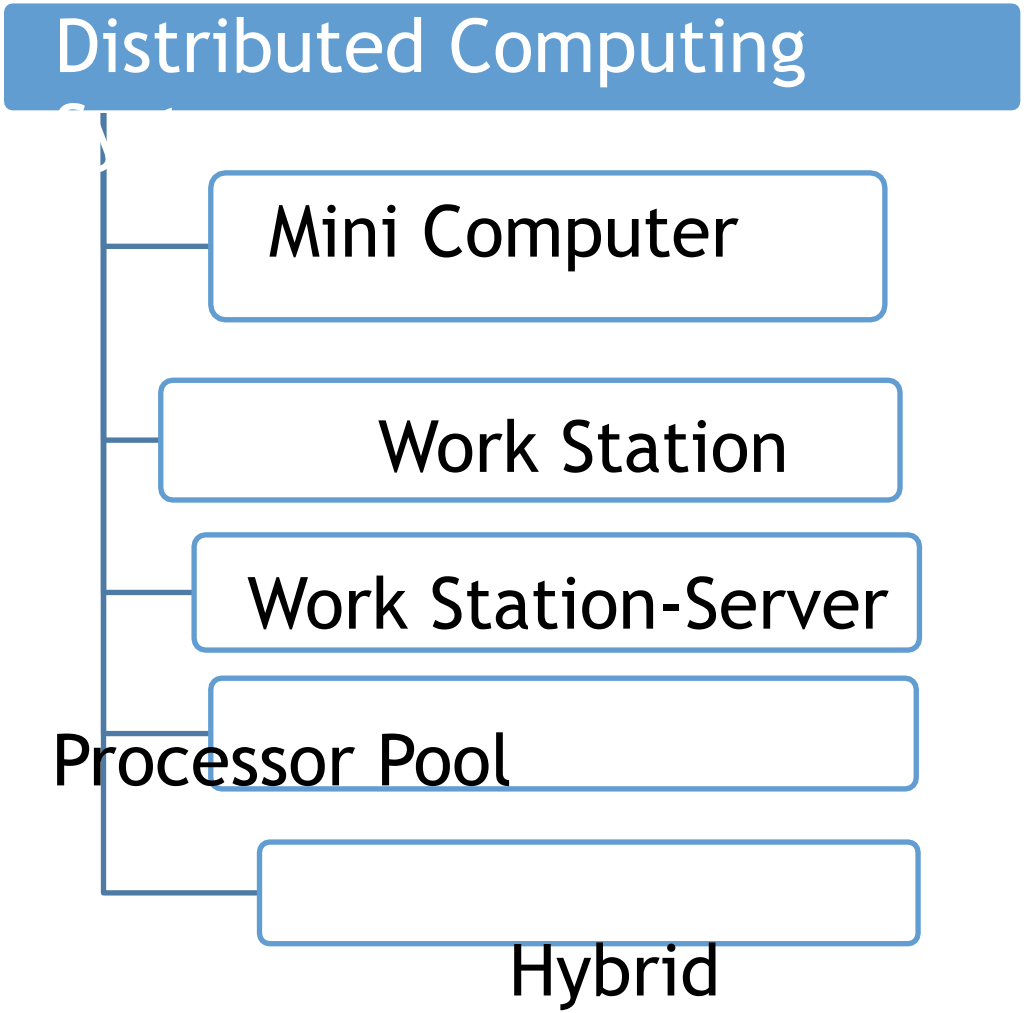
- Job SetUp Time
- Batch Processing
- Multiprogramming

Multiple user can access

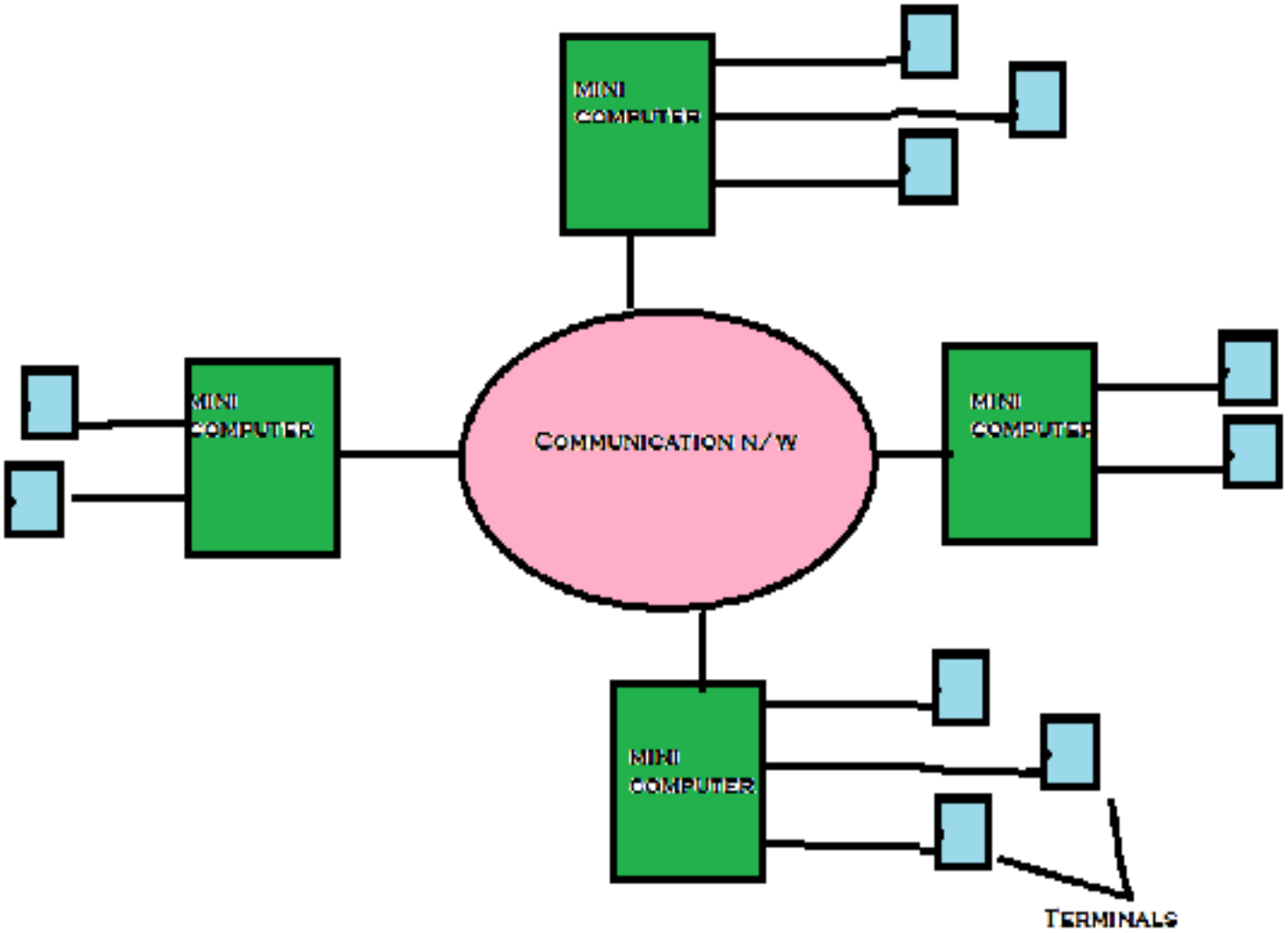
- Time Sharing (resource share and access from different place)
- Mini Computers
- Workstation
- MicroProcessors
- LANs/WANs
  - Merging of Computers and Networking-

    ➔Distributed Computing Systems(1970s)

Distributed Computing System Models



Mini Computer Model



Mini Computer Model

Different databases on different remote machines

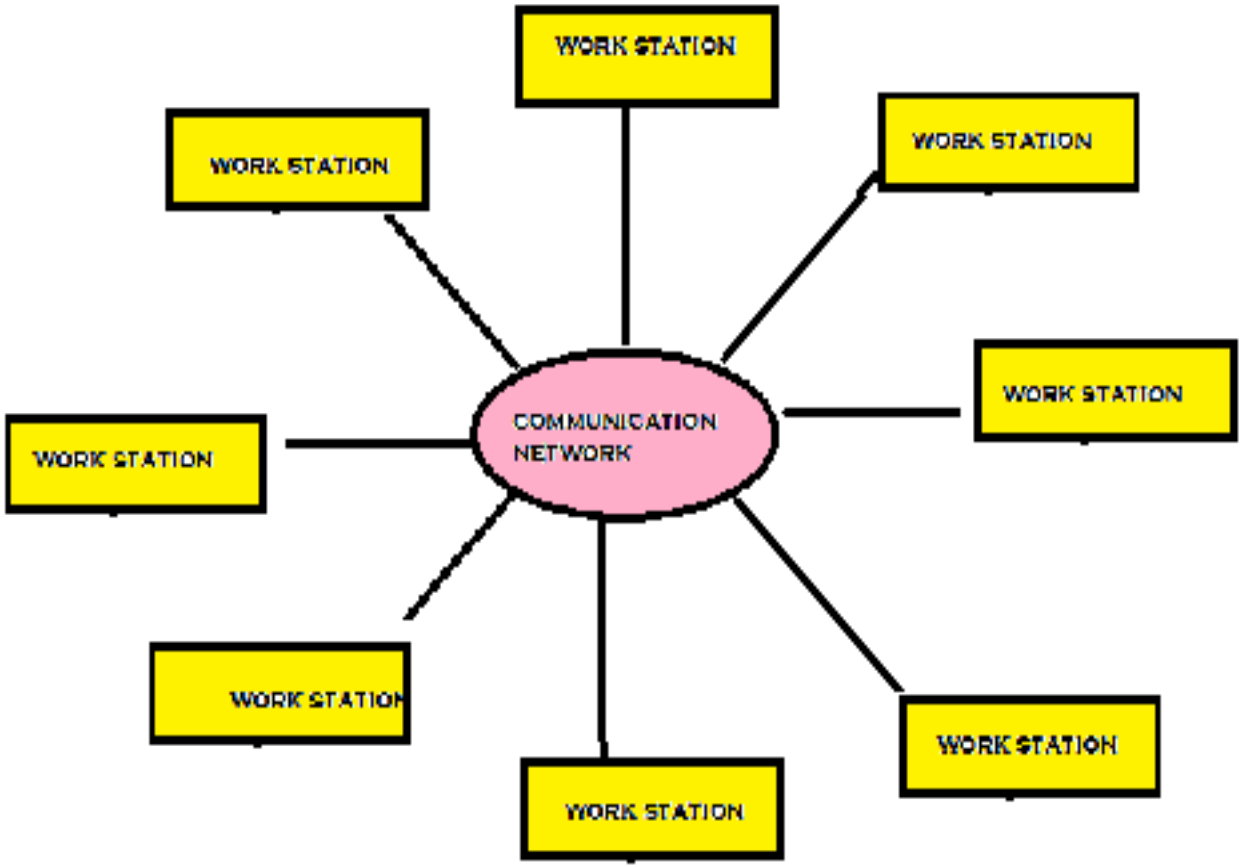
Extension of time sharing systems

Multiple users simultaneous login

Example

ARPANET

Work Station Model



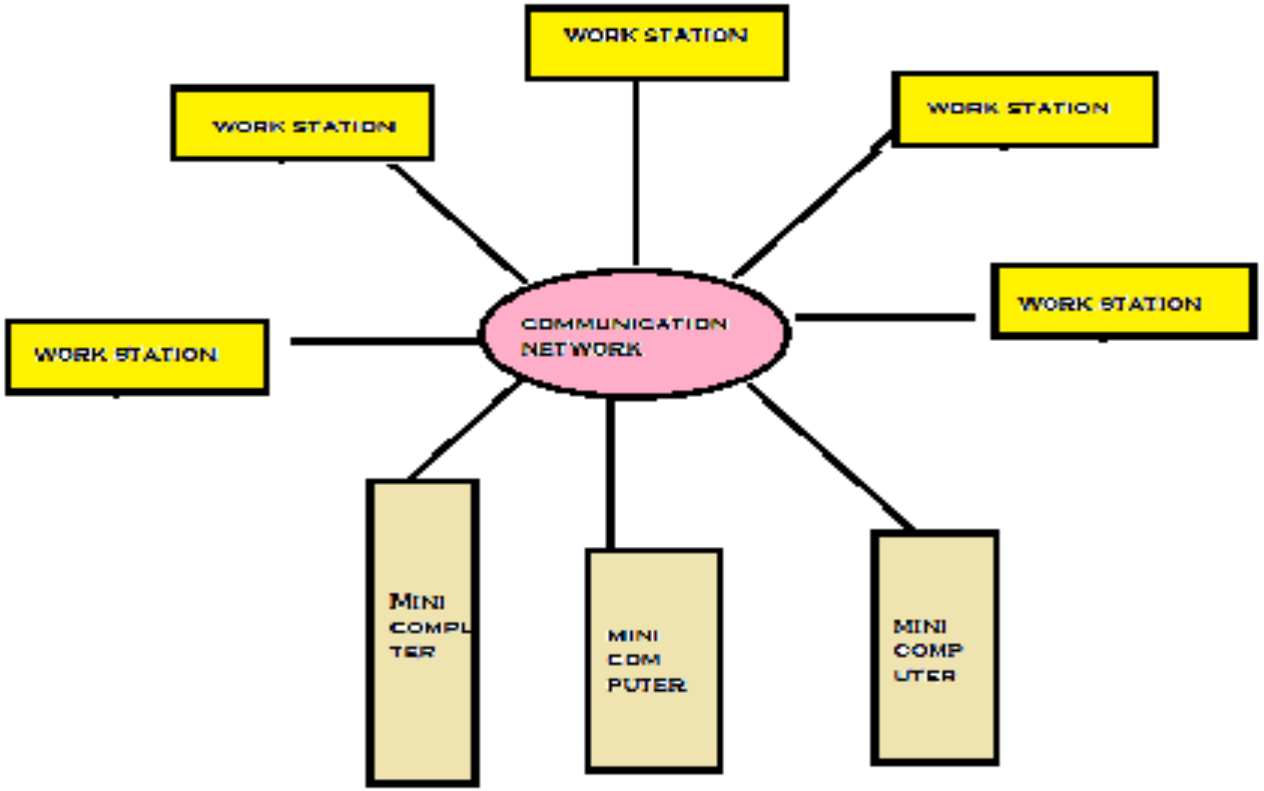
Work Station Model

Idle workstation in big campus

Issues

- Finding idle ws
- How to transfer job
- Suppose if it starts working?
  - Allow remote process to share the resources of the workstation along with its own logged on users processes.
  - Kill the remote process
  - Migrate remote process back to its home workstation

Work Station Server Model



Work-Station Server Model

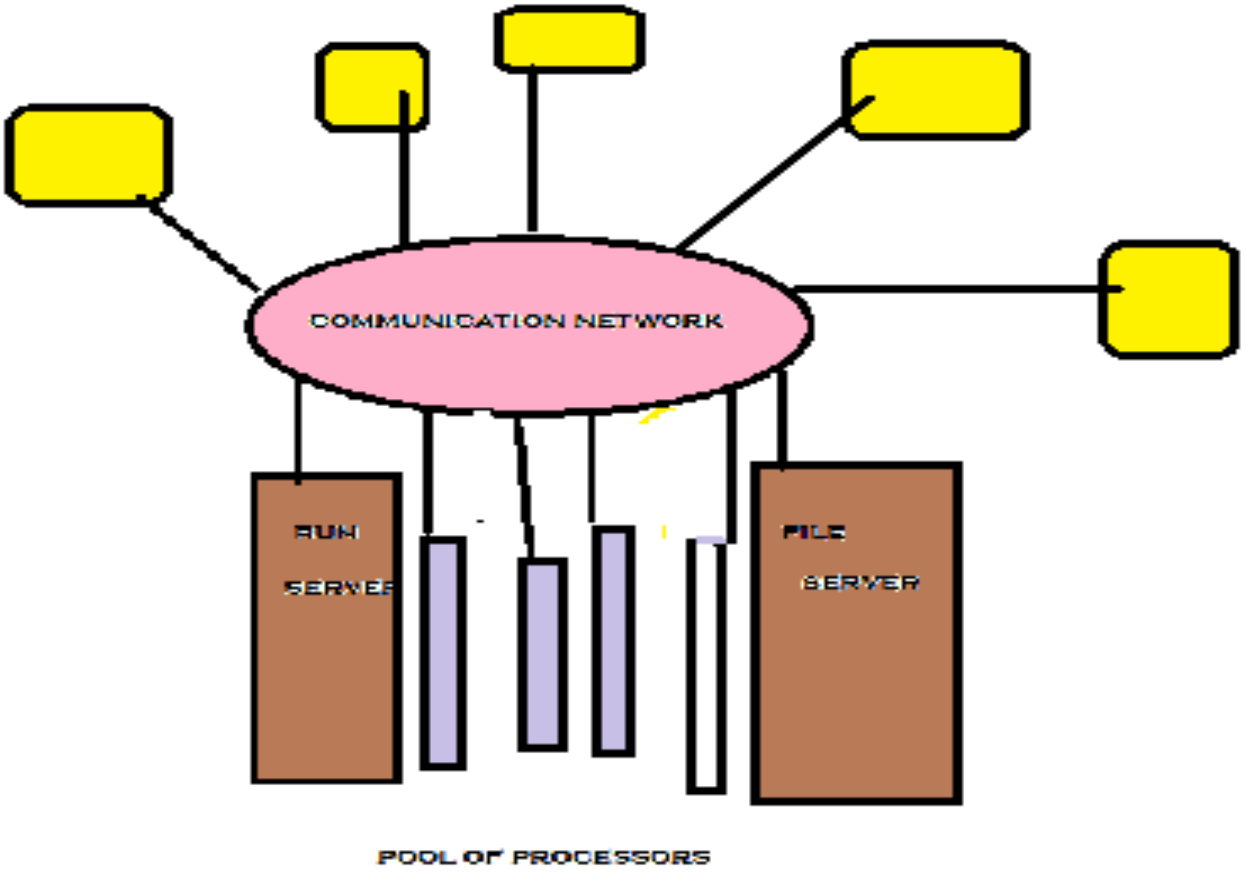
- Consists of few minicomputers and several workstations interconnected by a communication network.

Diskful workstations-Work station model is a Network of personal workstations, each with its own disk and a local file system

Diskless workstations

- Cheaper-few minicomputers equipped with large, fast disks that are accessed over the network
- Diskless Workstation are preferred-Backup and H/W maintenance are easier
- No process migration needed
- Request Response protocol is used to access the services of the server machines

Processor Pool Model



Processor Pool Model

- Most of the time user does not need computing power but once in a while user needs a very large amount of computing power for a short time.
- In workstation server model, processor is allocated to each user but in Processor poor model Processors are gathered-Users share whenever needed
- Terminals are diskless workstations
- Run Server- Special server manages and allocated the processors in the pool to different users on a demand basis

Processor Pool Model

As compared to workstation -server model , Processor Pool Model

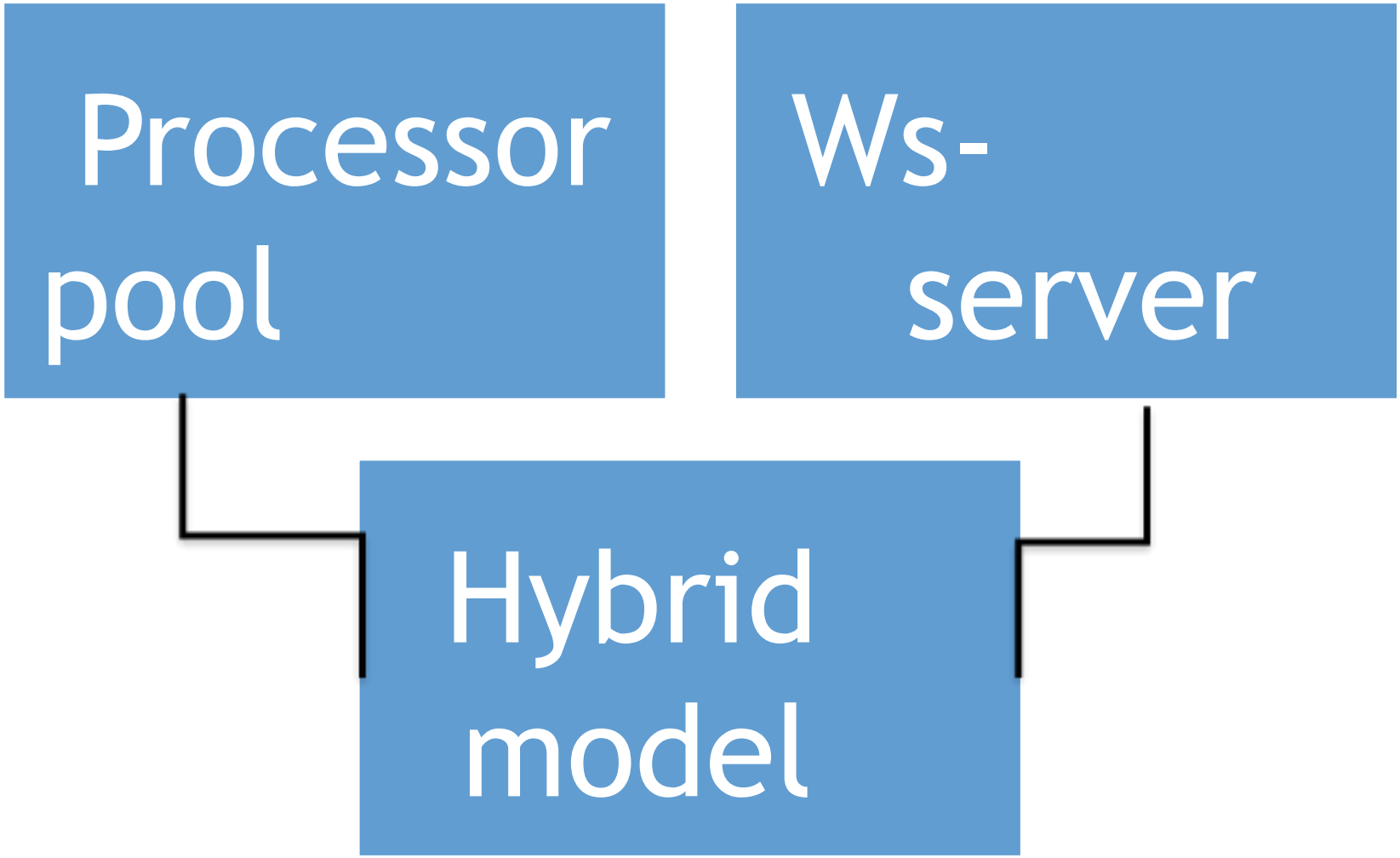
- Allows Better utilization of processors
- Provides Greater Flexibility- System services can be easily expanded without the need to install any more computers
- Considered to be unsuitable for high performance interactive applications – especially those using graphics or window systems

  - due to slow speed of communication between the computer on which the application program is executed and the terminal via which the user is interacting

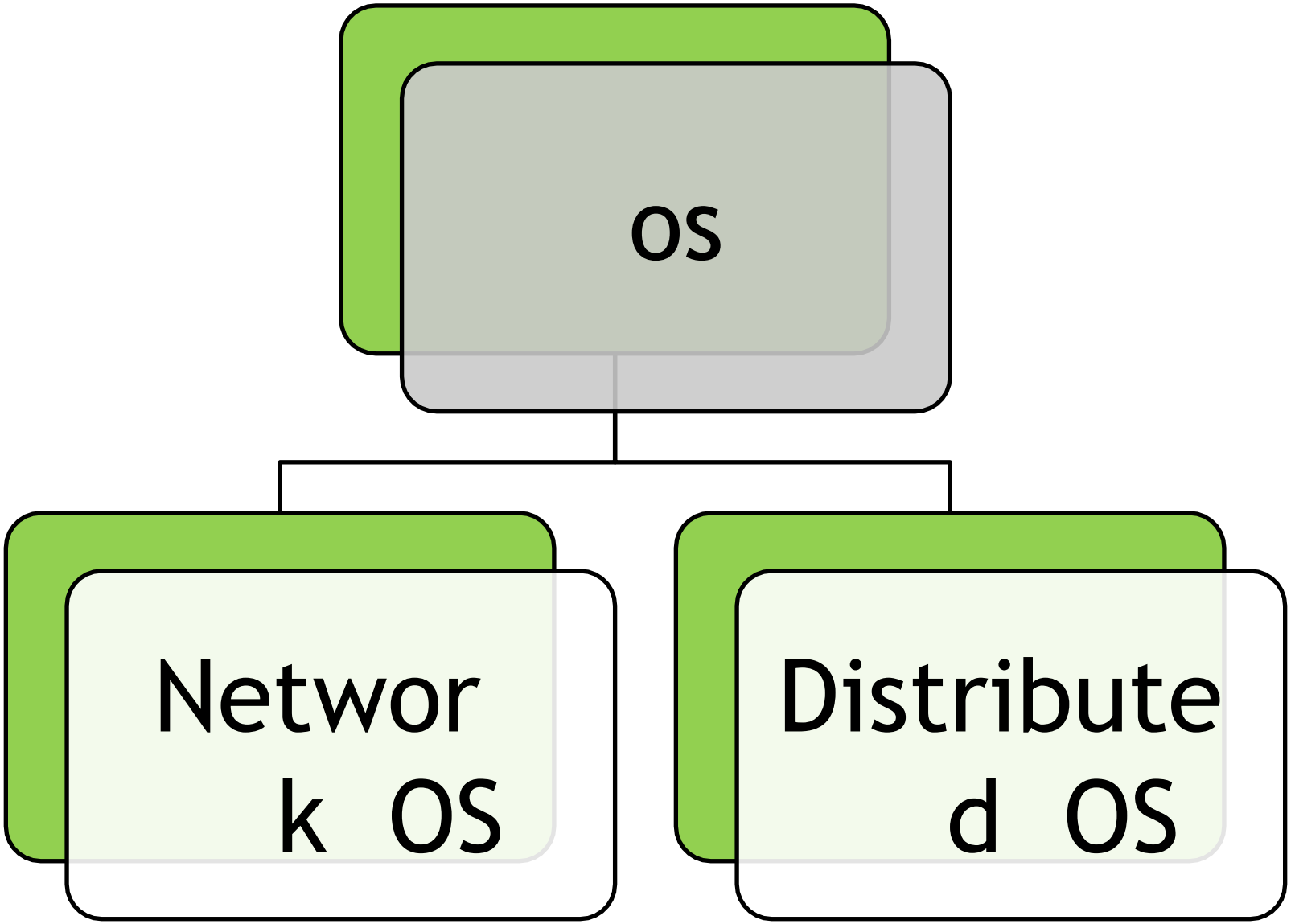
Hybrid Model

- Out of the four models WS-Server model is widely used for building distributed computing systems Because Suitable for interactive jobs
- Processor-pool model is suitable for Massive applications which need computations
- Hybrid model is based on the WS-Server model but with the addition of a pool of processors
- Processors in the pool can be allocated dynamically for computations that are too large for workstations

Hybrid Model



What is a Distributed OS?



Differences

•A network operating system is made up of **software** and associated protocols that allow a set of computer network to be used together.

• A distributed operating system is an ordinary centralized operating system but runs on multiple independent CPUs.

Metrics	Network OS	Distributed OS
System Image	The Users are aware that multiple systems are used	Virtual Uniprocessor
	The User knows in which machine his job is executed	Unaware of this information
	User know where his/her information is stored	Does not know
	Explicit commands for file transfer	Same Commands
	Control over file placement is manual	Automatic

Differences

Metrics	Network OS	Distributed OS
Autonomy	Each Computer is independent	Not independent
	Have local OS	Common OS
	Degree of autonomy is high	Low
Fault Tolerance Capability	Little or No fault tolerance capability	High Fault tolerance
Example	Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux	AIX operating system for IBM RS/6000 computers. Solaris operating system for SUN multiprocessor workstations. .

ISSUES IN DESIGNING DISTRIBUTED OS

- Differences in the complexity of the Design between traditional system and Distributed system
- Centralized Os-User can request status information and it is available

Distributed OS- cannot have Complete information about the system and not available to the user.

- Centralized Os- Resources are nearer

Distributed Os-Far away

- Centralized Os- Common Clock

Distributed OS- No Common clock,Lack of Up to date information (Delivery of messages – delayed/lost)

ISSUES

- Transparency
- Reliability
- Flexibility
- Performance
- Scalability
- Heterogeneity
- Security

ISSUES IN DESIGNING DISTRIBUTED OS

A lot of issues But

flexible,
efficient,
reliable,
secure, and
easy to use.

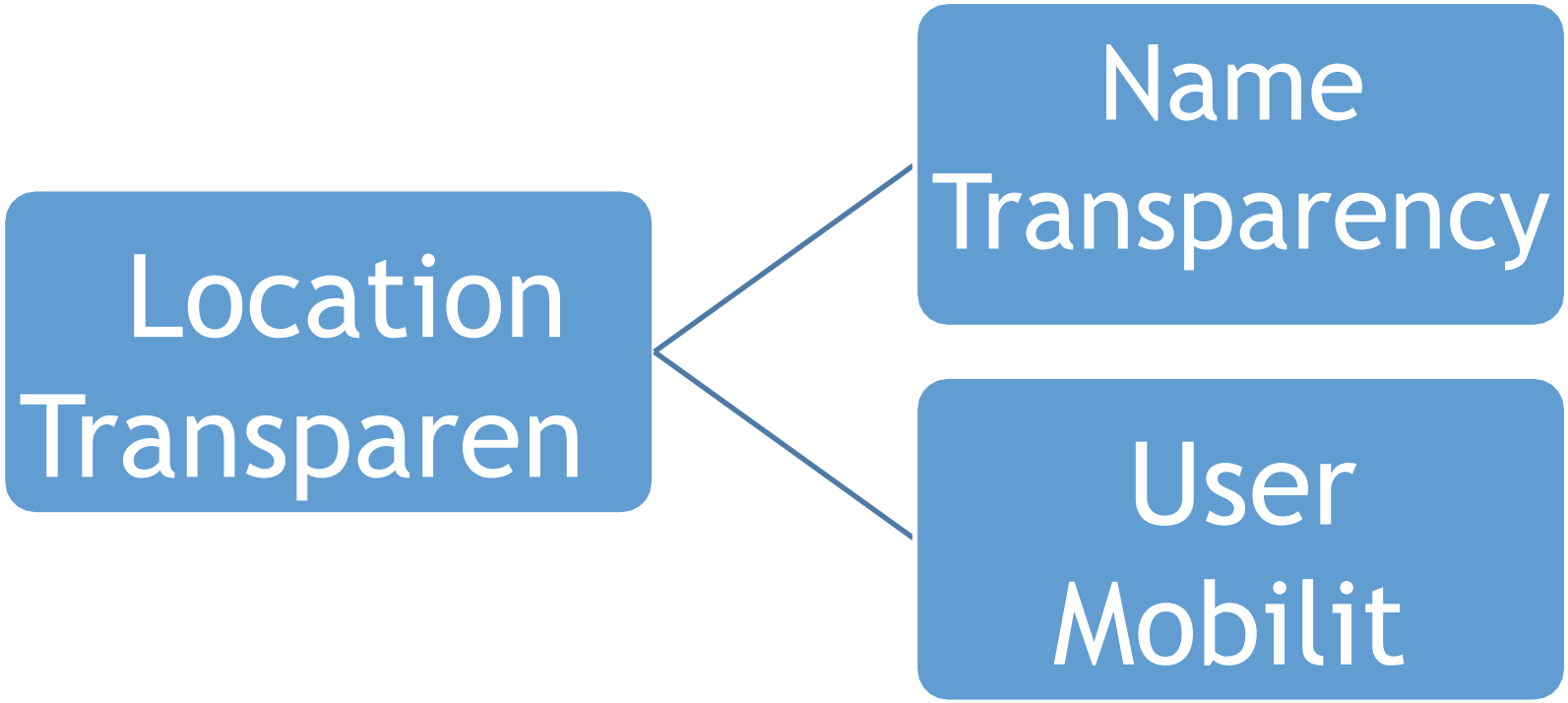
Transparency

- Single Virtual Uniprocessor image
- Eight forms of transparency - ISO 1992 Reference Model for Open Distributed Processing
  - Access transparency
  - Location transparency,
  - Replication transparency,
  - Failure transparency,
  - Migration transparency
  - Concurrency transparency,
  - Performance transparency, and
  - Scaling transparency

Transparency

- Access transparency
- User should not be able to recognize a resource as local or remote
  - Remote resources in the same way as local
  - Uniform system calls
  - Suitable for limited types of application due to its Performance limitations

Transparency



Transparency

Name Transparency

- Name of resource does not reveal physical location of the resource
- Movement of files need not change the names
- Resource names are unique

User Mobility

- Same resources from different locations are accessed with same name.
- Users access to resources without any extra effort

Transparency

Replication Transparency

- Creating Replicas of files and resources on different nodes of distributed systems – Better performance and reliability
- Should be transparent to users
  - Issues
    - Naming of replicas (name of replica should be mapped to user supplied name by the system)
    - Replication control (No.of copies? When to create? Where to place?)

Transparency

- Failure Transparency
- Masking from users- partial failure in the system
- DOS will continue to function in degraded form when failure occurs.
  - communication link failure,
  - a machine failure, or
  - storage device crash
- All types of failures cannot be handled in a user transparent manner.
- Theoretically possible, Practically not justified

Transparency

Migration Transparency –

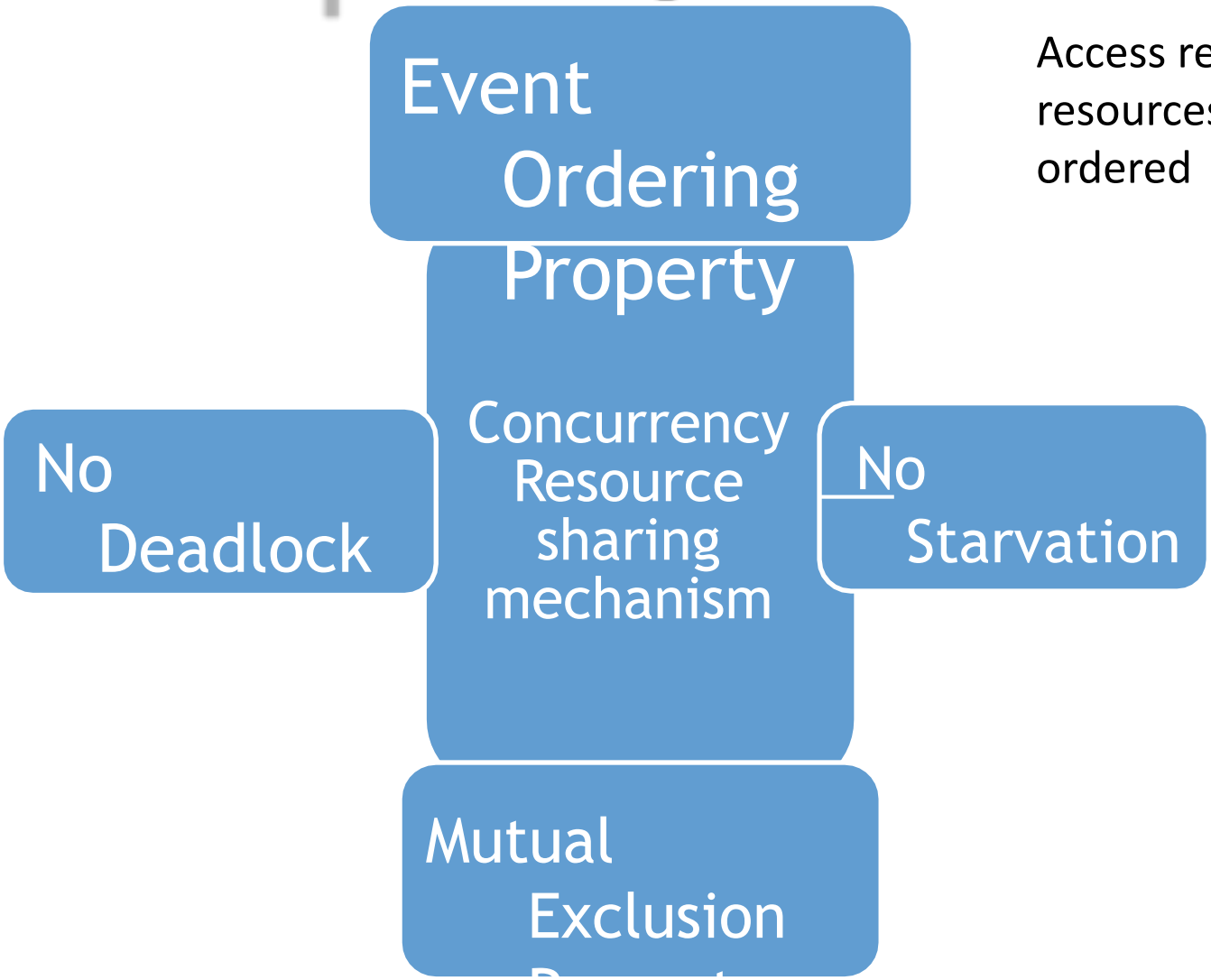
(Performance ,Reliability, Security)

- Migration decisions should be automatic
- Migration should not require any change in name
- If the receiver process migrates to another location before receiving messages from the sender , IPC should ensure that migrated process receives the message.

Concurrency Tranparency

- Multiple user- spatially separated use the system concurrently
- Concurrent update of the same file by two processes should not be allowed

Transparency



Access request to resources -> properly ordered

Transparency

Performance Transparency

To allow the system to be automatically reconfigured

- Processor – Loads should be distributed(No Idle Processors)
- Processing capability should be uniformly distributed among the available jobs

Requirement

- Intelligent resource allocation and migration facility

Scaling Transparency

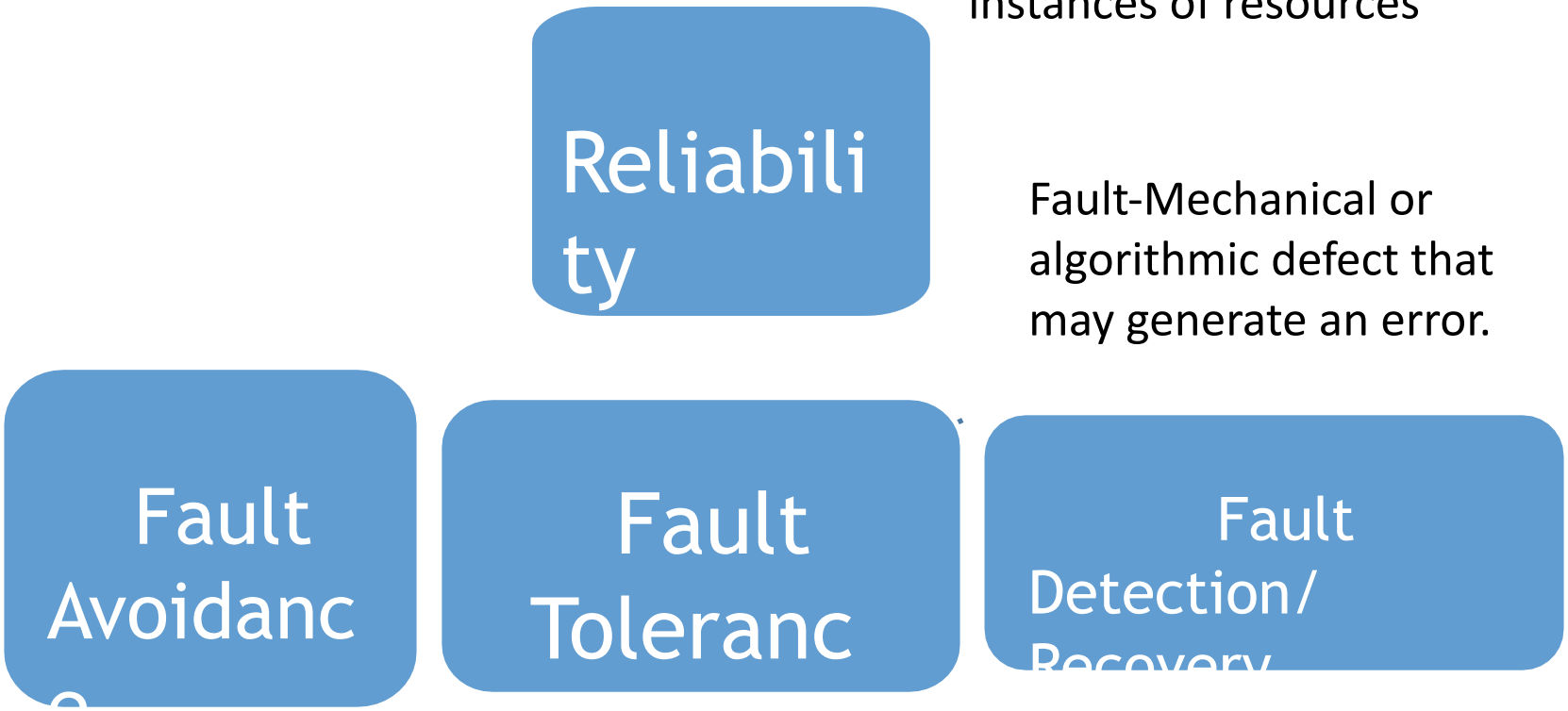
To allow system to expand without disturbing the user activities.

Requirement

- Use of Open System Architecture
- Use of scalable algorithms

2. Reliability

DS expected to be more reliable – Due to Multiple instances of resources



Fault-Mechanical or algorithmic defect that may generate an error.

Fault->Failure

Reliability

System Failure

- Fail-stop –System stop functioning after changing to the state in which failure can be detected
- Byzantine –System continues to function but produces wrong result.

System bugs causes this failure.

Reliability

1. Fault Avoidance

- Designing the components to minimize the occurrence of faults
- Use high reliability components
- The Designers of software components has to test to make the hardware components highly reliable .

2.Fault Tolerance

- Ability of the system to function in the event of partial system failure

Techniques to improve fault tolerance

i. Redundancy Techniques

- To avoid single point of failure -Hardware and software components are replicated.
- Additional system overhead is needed to maintain replicas to ensure that they are consistent.

Reliability

How much replication is enough?

System is said to be *k-fault tolerant* if it can continue to function even in the event of the failure of *k* components.

K fail stop failures – k+1 replicas

K byzantine failure - min 2k+1 replicas (voting mechanism)

ii. Distributed Control

Distributed control mechanism is used in name servers, scheduling algo to avoid single points of failure.

3.Fault Detection and Recovery

Use of H/W and S/W mechanism to determine the occurrence of failure and then to correct system to an acceptable state.

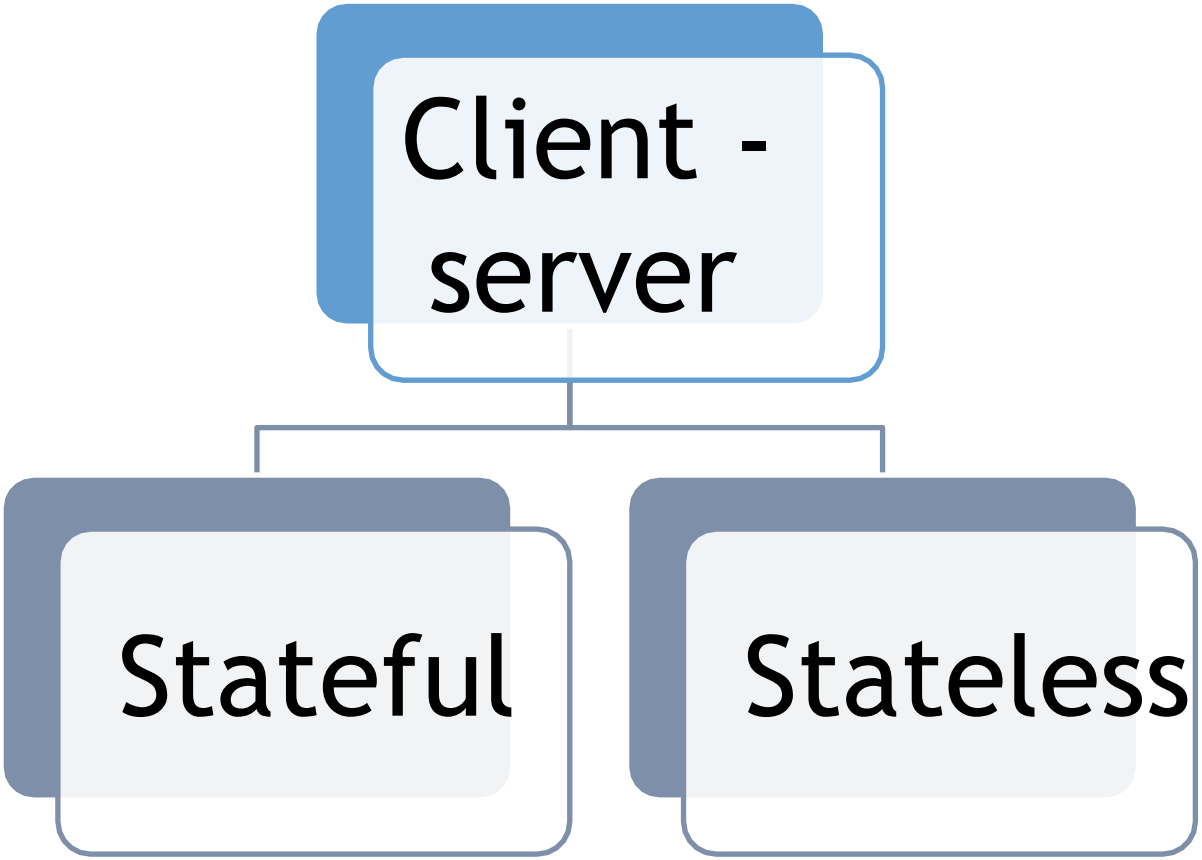
Techniques:

i. Atomic Transactions

- Collection of operations that takes place indivisibly in presence of failure
- All operations goes to completion or none-incase of failure data object restore to original state if system supports trans mechanism

Reliability

ii. **Stateless Servers**



Stateful Server: Depends on history of serviced request.

Stateless Server – Crash Recovery- Easy- No client state info is maintained.

Reliability

iii. **Acknowledgements and time-outs based retransmissions**

Node crash/Communication link failure-message lost

- Duplicate messages are a problem here
- Detection and handling of Duplicate messages

  - Generating and assigning Sequence Numbers to messages

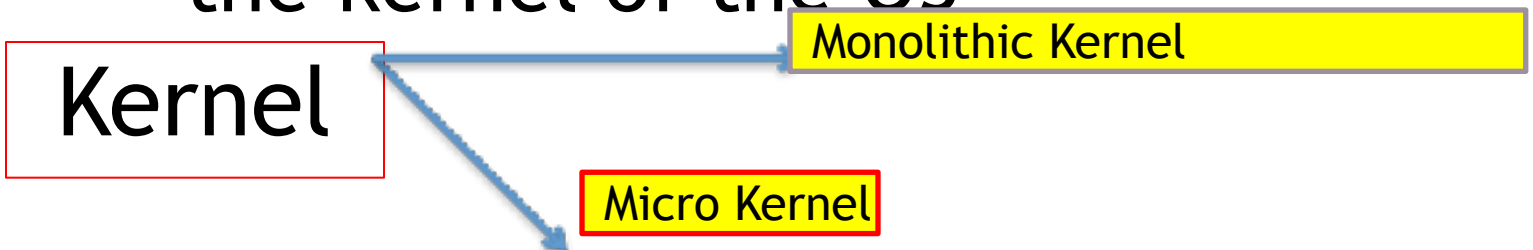
3.Flexibility

Why DOS should be flexible?

Ease of modification - Some parts of design needs to be replaced/modified if bugs detected or not suitable for changed system

Ease of enhancement -User should have the flexibility to add new service or the change the existing style.

The important design factor is Designing the kernel of the OS



Flexibility

Kernel of an OS is its central controlling part that provides basic system facilities with separate address space inaccessible to user processes. User cant replace or modify

Monolithic Kernel

- All functions are provided by the kernel
- Big structure
- UNIX

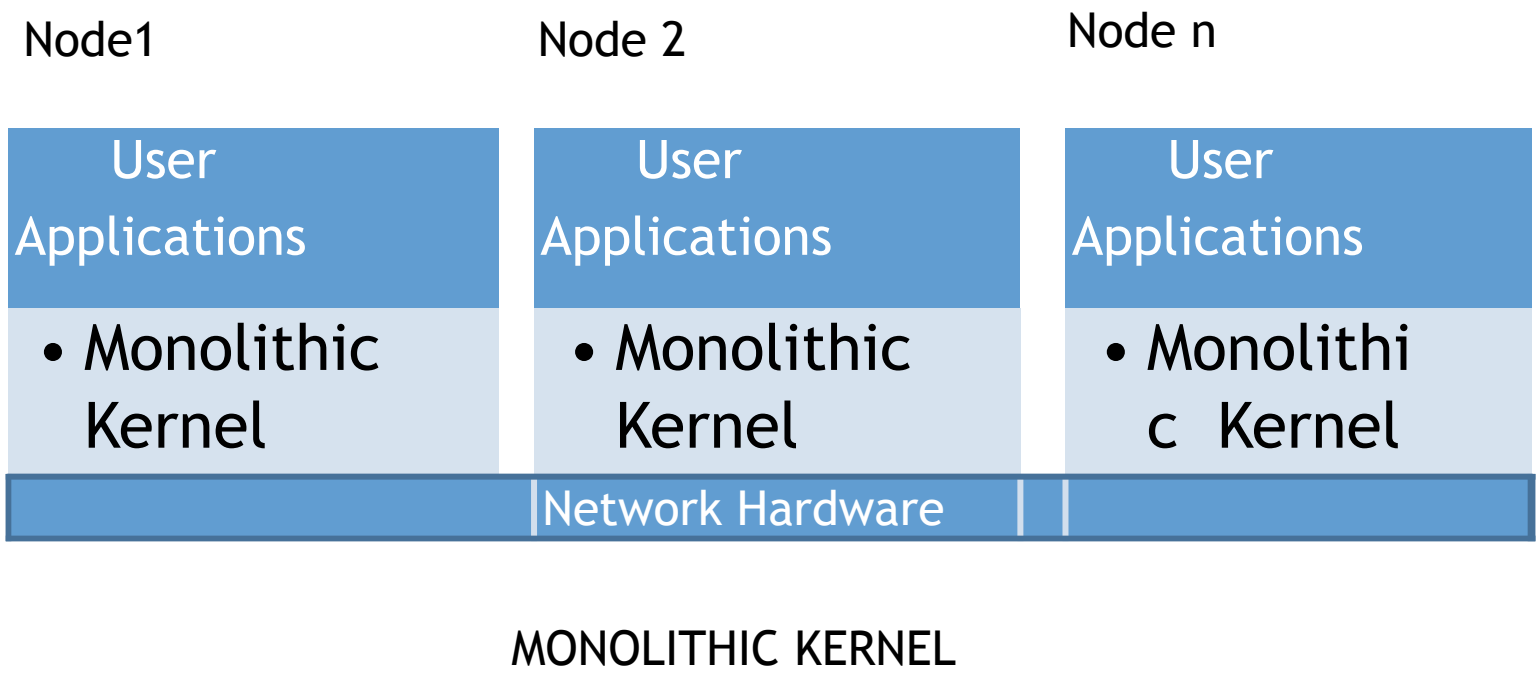
- Micro Kernel

To Keep Kernel as small as possible

OS provides minimum facilities - Services provided is IPC -Low Device mgmt and Mem. Management

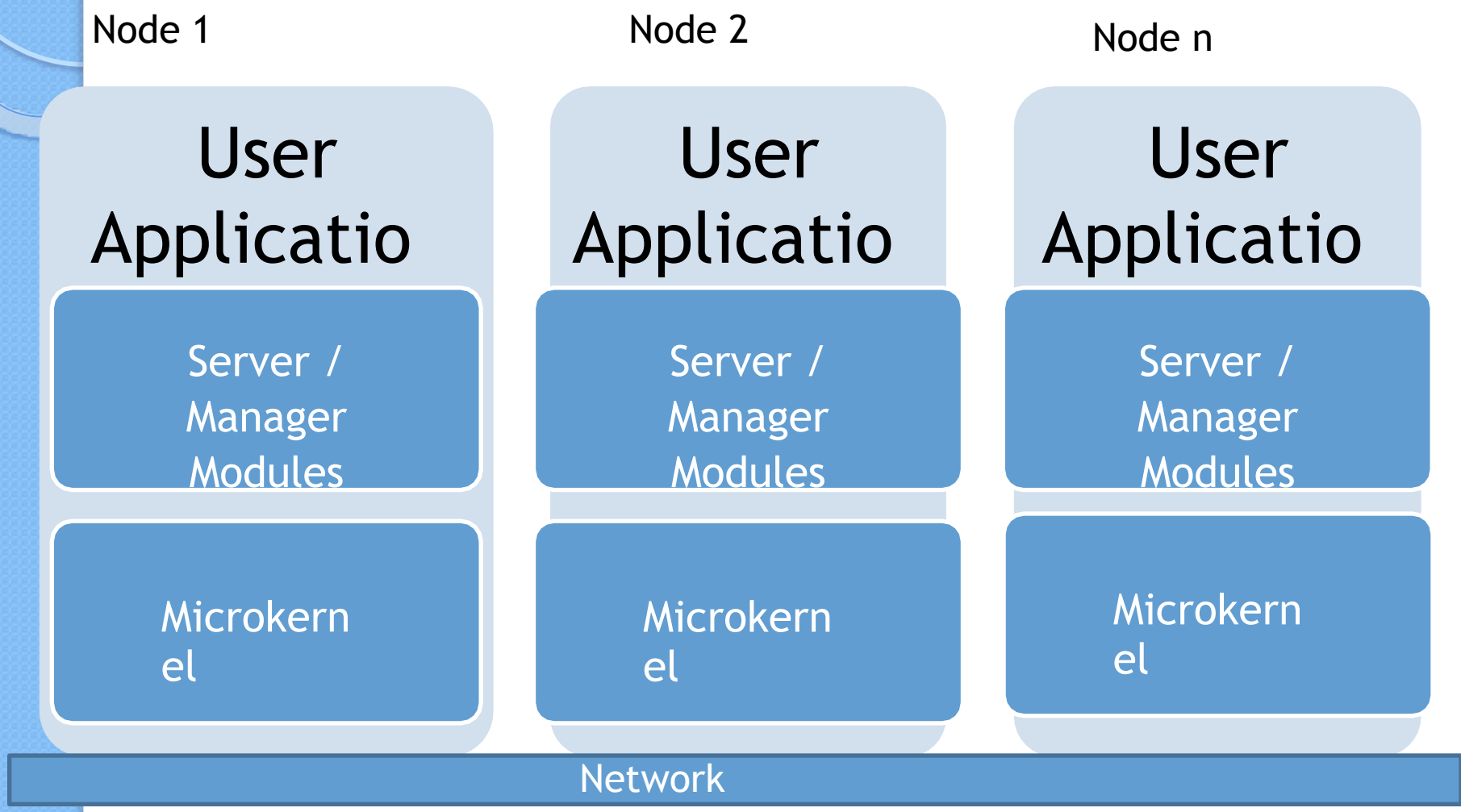
other services are implemented as user level server processes.

Flexibility



MONOLITHIC KERNEL

Microkernel



Flexibility

Monolithic Model	Microkernel Model
Major OS services are provided by the kernel	Only minimal facilities and services are provided
Kernel has a large monolithic structure	Size of the kernel is small
No such thing	User level server processes services
Large size reduces flexibility	Increased flexibility
Reduced Configurability	Highly modular in nature
complex	Easy to modify
complex	Easy to add services
Changes can be done by interrupting users	Without interrupting users, the changes can be performed In the OS
No	the servers have to use some form of message-based interprocess communication mechanism
No	Message passing requires context switches

Performance

Design principles in order to achieve Good Performance are

1. Batch if possible
   - Large chunks of data- transfer instead of small across network
   - Piggybacking acknowledgement (Message Exchange)
2. Cache whenever possible
   - Makes data available
   - Saving large amount of Computing time and bandwidth
3. Minimize copying data

   Data path to send the message

   Senders Stack to Message buffer

   Message buffer in sender address space to message buffer in kernel address space

   From kernel to Network Interface Board

   Receiving Same in reverse

Performance

4. *Minimize network traffic.*
   - migrating a process closer to the resources
   - to cluster two or more processes that frequently communicate with each other on the same node of the system

5. *Take advantage of fine-grain parallelism for multiprocessing*
   - Threads-used for structuring the server processes-Simultaneously service requests from clients
   - concurrency control of simultaneous accesses by multiple processes to a shared resource

Scalability

capability of a system to adapt to increased service load.

PRINCIPLES FOR DESIGNING SCALABLE SYSTEMS

1. Avoiding centralized entities (such as single file server or single database)

- The failure of the centralized entity often brings the entire system down. Hence, the system cannot tolerate faults
- Even if the centralized entity has enough processing and storage capacity, the capacity of the network saturated
- In a wide-area network consisting several interconnected LAN increases network traffic.

The increased users increases the complexity

Scalability

2. Avoid centralized algorithms.

- *This algorithm collects information from all nodes.*
- *Processing info on single node and distributing the results to rest of the nodes- Not acceptable from a scalable point of view.*

Eg: Scheduling algo - selects lightly loaded node -host selection time too long.

The complexity of the algorithm is *O(n2)*

- It creates heavy network traffic and quickly
- consumes network bandwidth.Therefore, in the design of a distributed operating system, only decentralized algorithms should be used.

3. Perform Most operations on a client workstations - Caching

Heterogeneity

Interconnected set of dissimilar hardware and software

Incompatibilities in heterogeneous DS:

- Communication protocols
- Topologies
- Servers operating at different node of system may be different.

Pros

-Provides Flexibility - Different computer platforms for different applications.

Heterogeneity

- We need translator which converts each format in the system to the format used in receiving node.
- n formats - (n-1) pieces of translation software at each node - n(n-1) pieces of translation software in the system.
- Complex

Intermediate standard data format -Less no. of conversions

Security

- In a centralized system, all users are authenticated by the system at login time.

- system can easily check whether a user is authorized to perform the requested operation on an accessed resource.

- In a distributed system this is not possible.

So Design should include to know

- Sender of the message should know that the message was received by the intended receiver
- Message sent by a genuine sender
- The content of the message is not altered

Cryptography and integrity(Trusting smaller servers rather than clients)