DISTRIBUTED OPERATING SYSTEMS
UNIT I
Evolution to Group Communication

Dr.K.Geetha
Associate Professor of Computer Science
Periyar Arts College
Cuddalore

Syllabus

- **SEMESTER –III MAIN PAPER -9 DISTRIBUTED OPERATING SYSTEMS**
- **UNIT-I**

Evolution –Models – Popularity - Distributed Operating System – Issues – Distributed Computed Environment - Features of a Good Message Passing – Issues-Synchronization –Buffering - – Multi data gram Messages – Encoding and Decoding of Message Data – Process Addressing – Failure Handling – Group Communication.

- **UNIT-II**

The RPC Model –Transparency – Implementation – Stub – Messages – Marshaling – Server Management –Parameter Passing Semantics – Call Semantics – Communication protocols – Complicated – Client server Binding – Exception Handling – Security – Special types – Heterogeneous – Light Weight – Optimization

- **UNIT-III**

Clock Synchronization – Event Ordering – Mutual Exclusion – Deadlock – Election Algorithms - Process Migration – Threads.

- **UNIT–IV**

Meet Hadoop: Data - Data Storage and Analysis - Comparison with Other Systems - A Brief History of Hadoop - The Apache Hadoop Project – Map Reduce: A Weather Dataset - Analyzing the Data with UNIX Tools - Analyzing the Data with Hadoop - Scaling Out – Hadoop Streaming - Hadoop Pipes

Syllabus

- **UNIT-V**

The Configuration API - Configuring the Development Environment - Running Locally on Test Data - Running on a Cluster - The Map Reduce Web UI - Using a Remote Debugger - Tuning a Job - Map Reduce Workflows

- **TEXT BOOKS**

1. Pradeep K. Sinha, "Distributed Operating System Concepts and Design", PHI, New Delhi,2007.

2. Tom White, "Hadoop: The Definitive Guide", Published by O'Reilly Media, Third Edition, 2009

Operating systems

- Memory Management
  - Allocation
  - De Allocation
  - Keeping track of memory space
  - Handling free areas
- Processor Management
  - Process status
  - Allocation
  - Release

Operating Systems

- Device Management
  - Hardware devices
  - Allocation
  - Release
  - Keeping track
- Information Management
  - keeps the track of information its location.
  - Who gets what data, when
  - Open,read,write and close files

Operating Systems

- Protection and Security
- Networking
- Accounting
- Error Detection
- Resource Allocation
- Sharing
- Resource Manager

Distributed Operating System

To make it possible

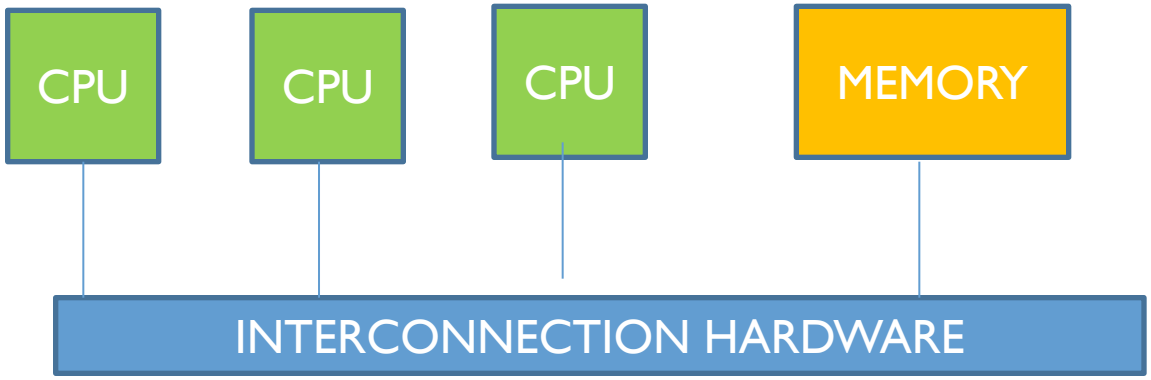
- A collection of independent computers

appears to be as a single computer

Computer Architecture

Multiprocessors

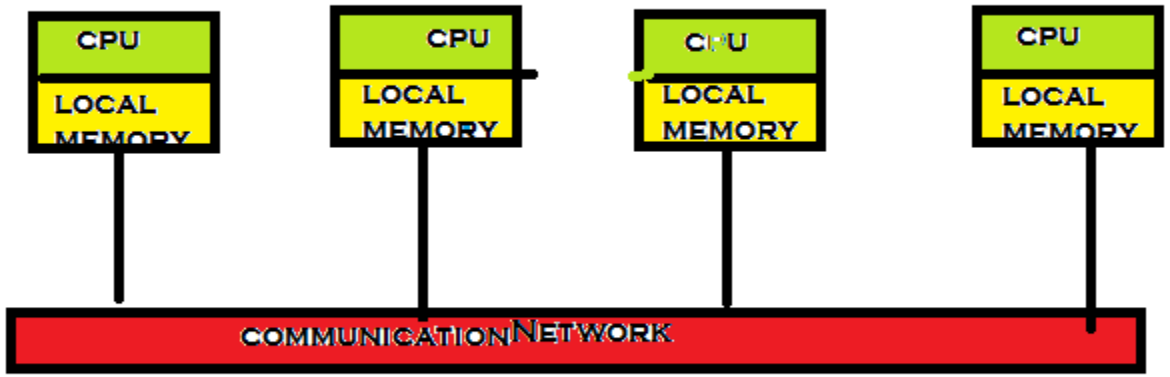
1. Tightly Coupled Systems(Parallel Processing Systems)
   - Single System Memory Shared by all Processors



Computer Architecture

2. Loosely Coupled Systems(Distributed Computing Systems)

Processors have their own local memory



Distributed Computing System

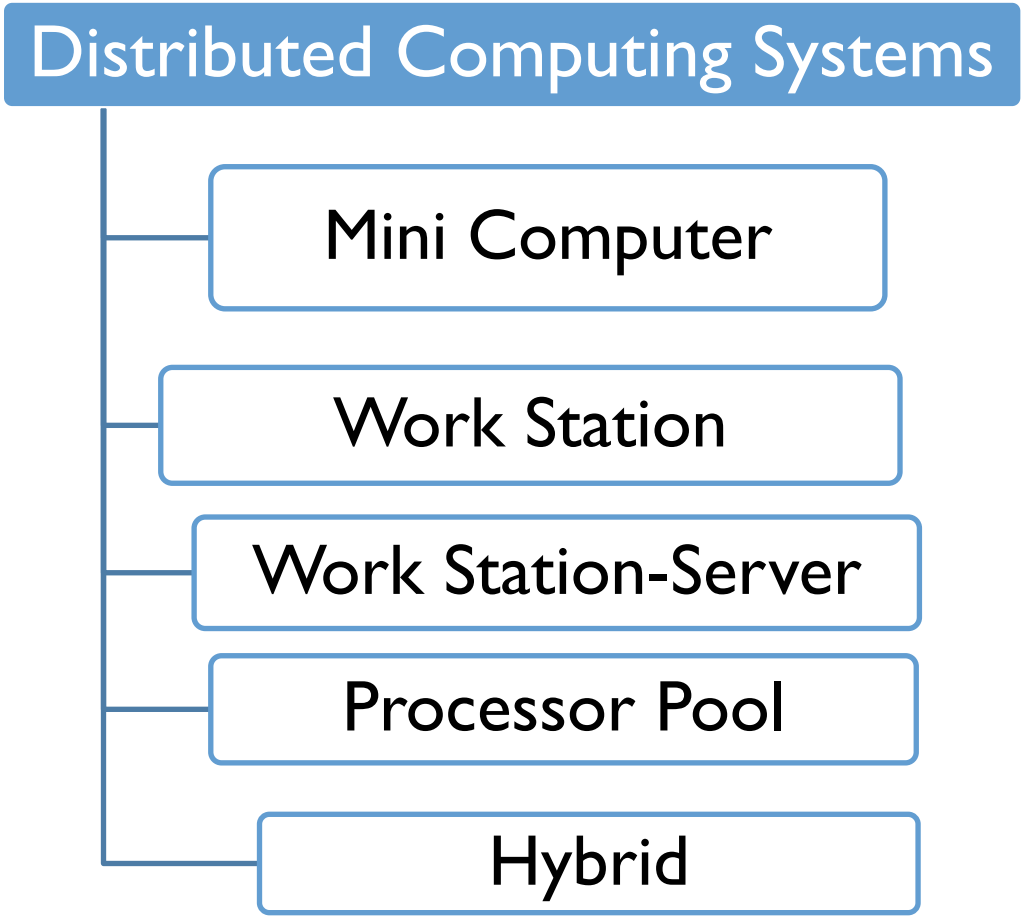
- Collection of processors interconneted by Network
- Each processor has local memory and peripherals
- Communicate by message passing

Evolution of Distributed Computing Systems

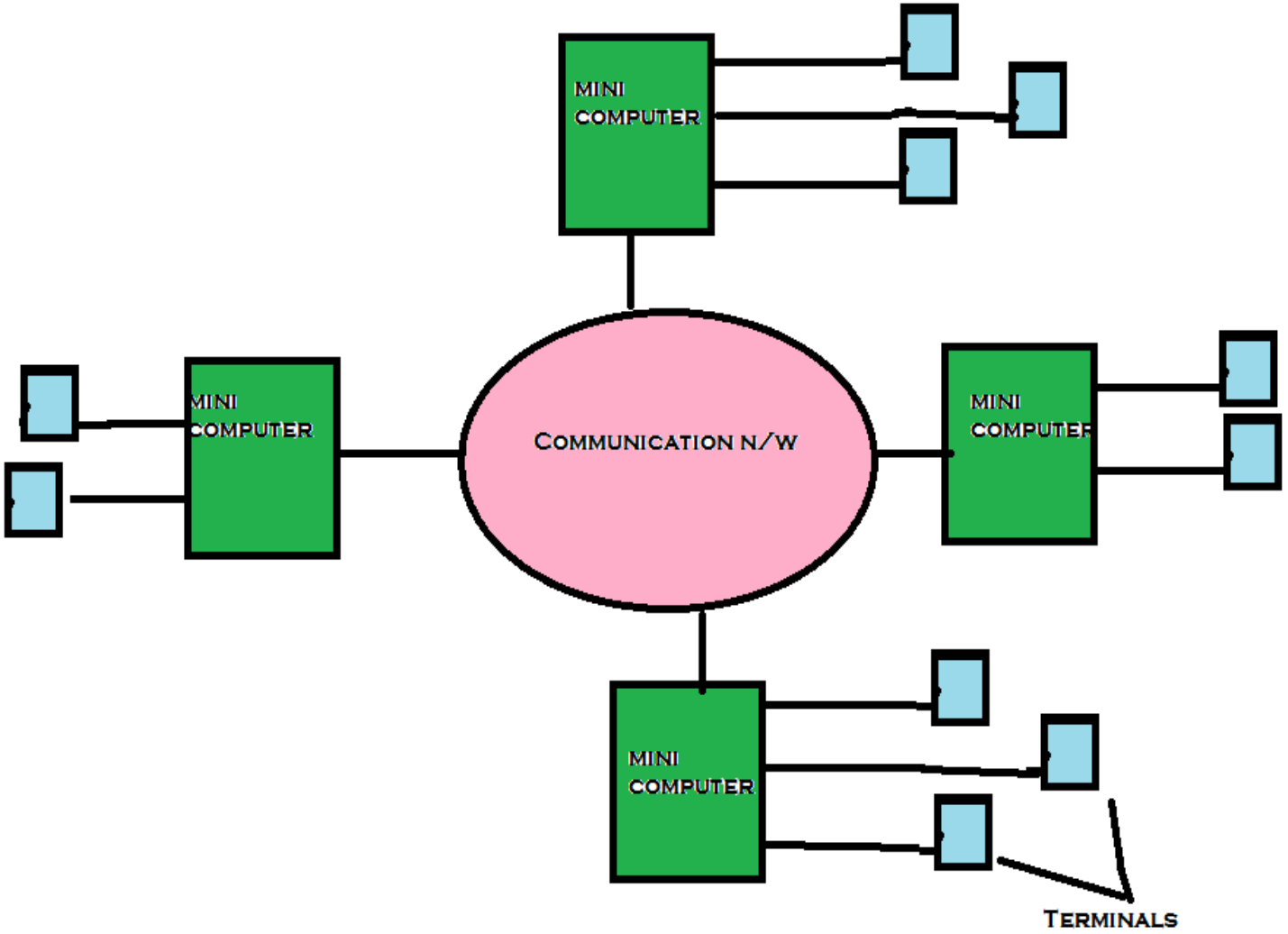
Early Computers

- Job SetUp Time
- Batch Processing
- Time Sharing
- Mini Computers
- MicroProcessors
- LANs/WANs
  - Merging of Computers and Networking-
    →Distributed Computing Systems(1970s)

Distributed Computing System Models



Mini Computer Model



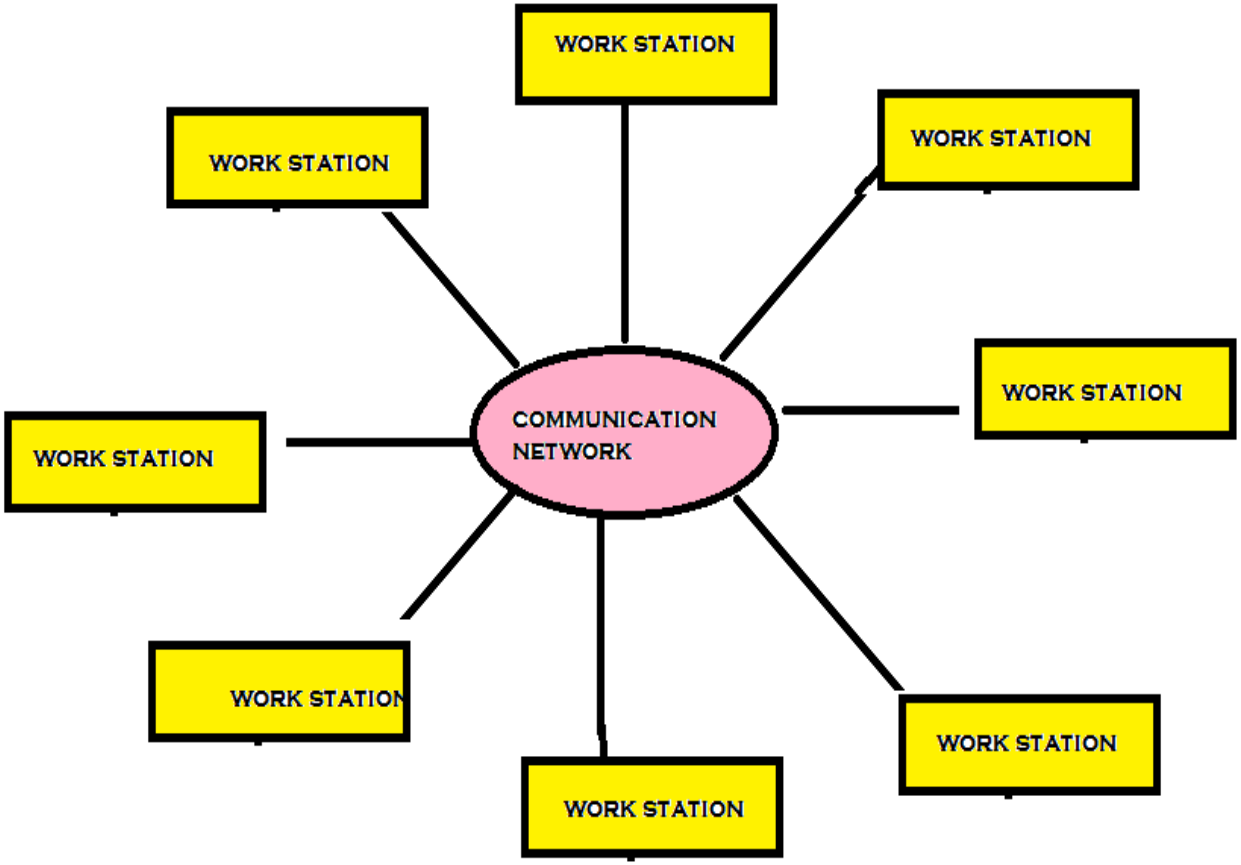
Mini Computer Model

- 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

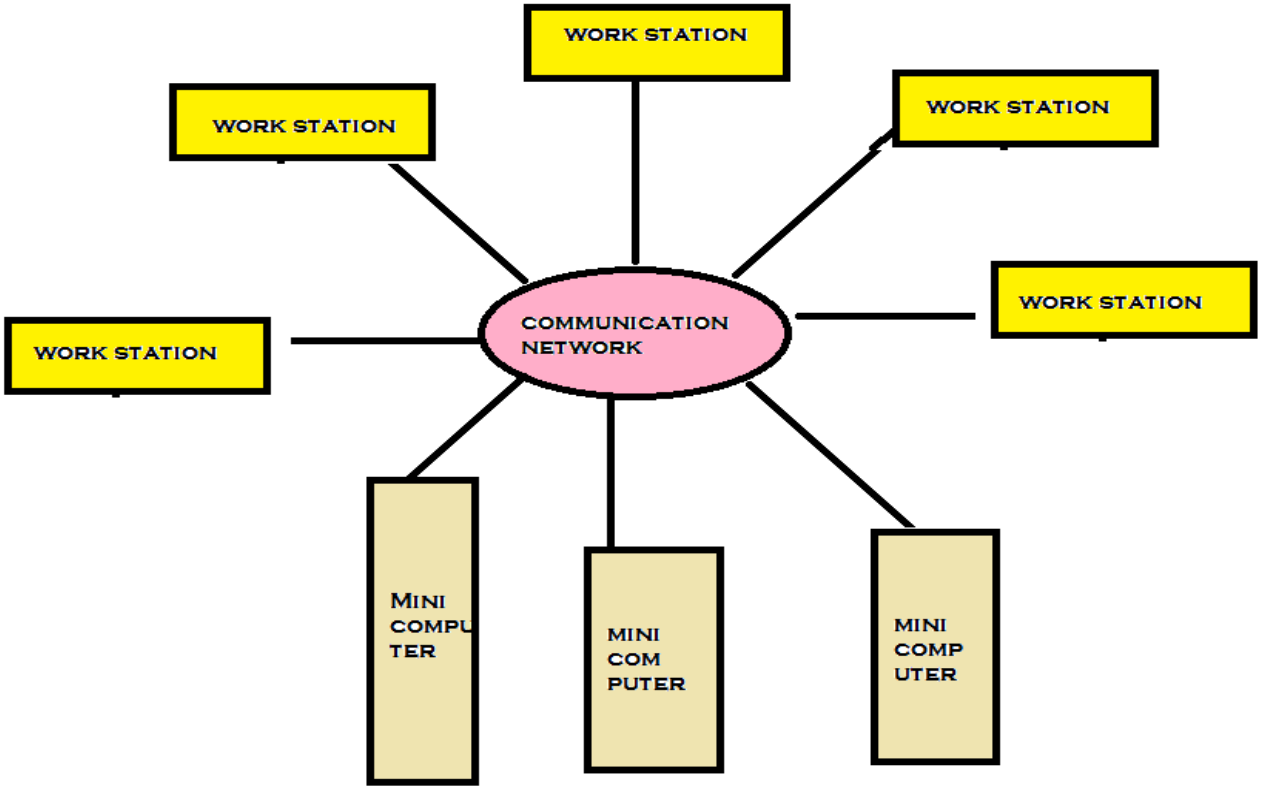
- Issues
  - Finding idle ws
  - How to transfer job
  - Suppose if it starts working?

Work station model -issues

Third issue solving

- Simultaneous performance
- kill the remote process
- migrate the remote process to home

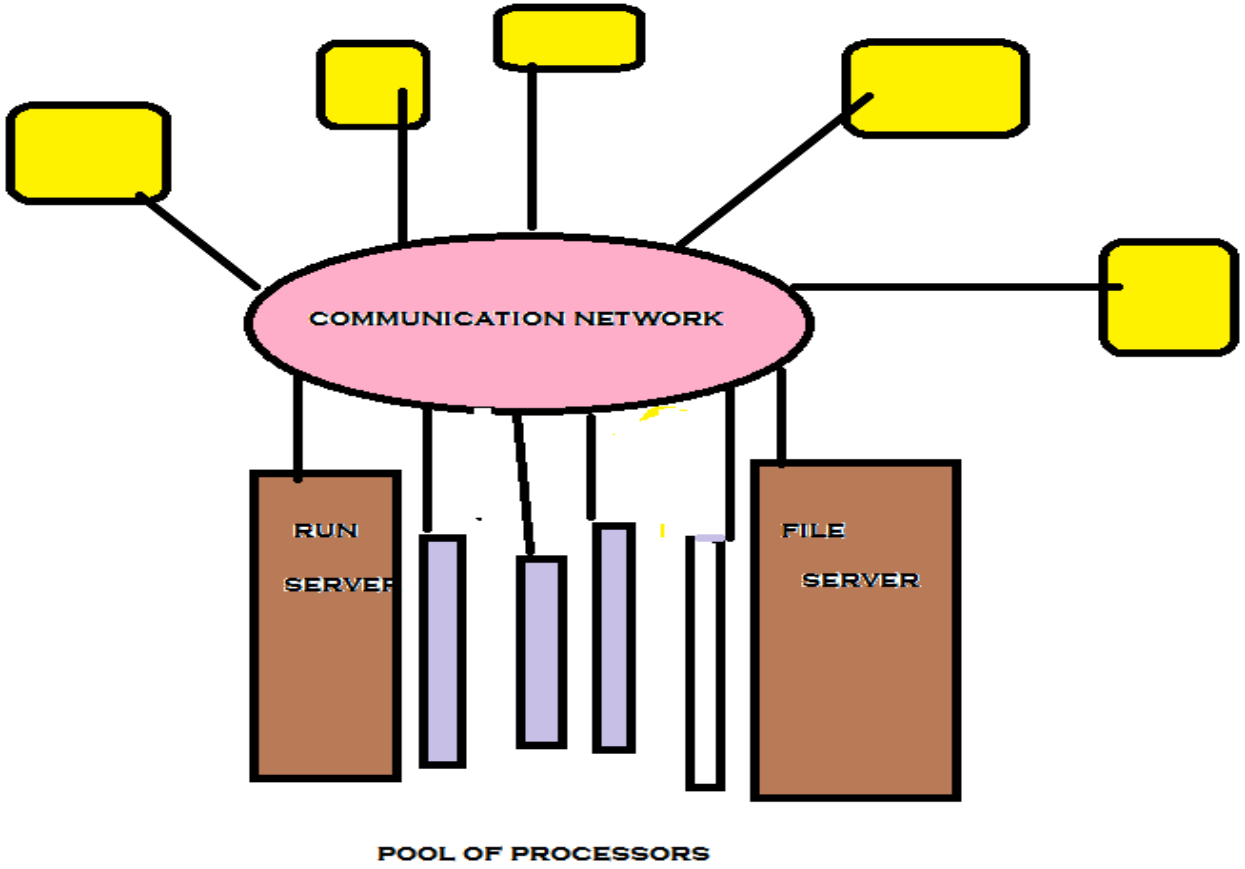
Work Station Server Model



Work-Station Server Model

- Diskful workstations
- Diskless workstations
- Cheaper
- Fast
- No process migration needed
- Request Response protocol

Processor Pool Model



Processor Pool Model

- Most of the time user does not need computing power
- Processors are gathered
- Users share whenever needed
- Terminals are diskless workstations
- Better utilization of processors
- Greater Flexibility
- Unsuitable for graphics or window system
- Low speed

Hybrid Model

- Out of the four models WS-Server model is popular

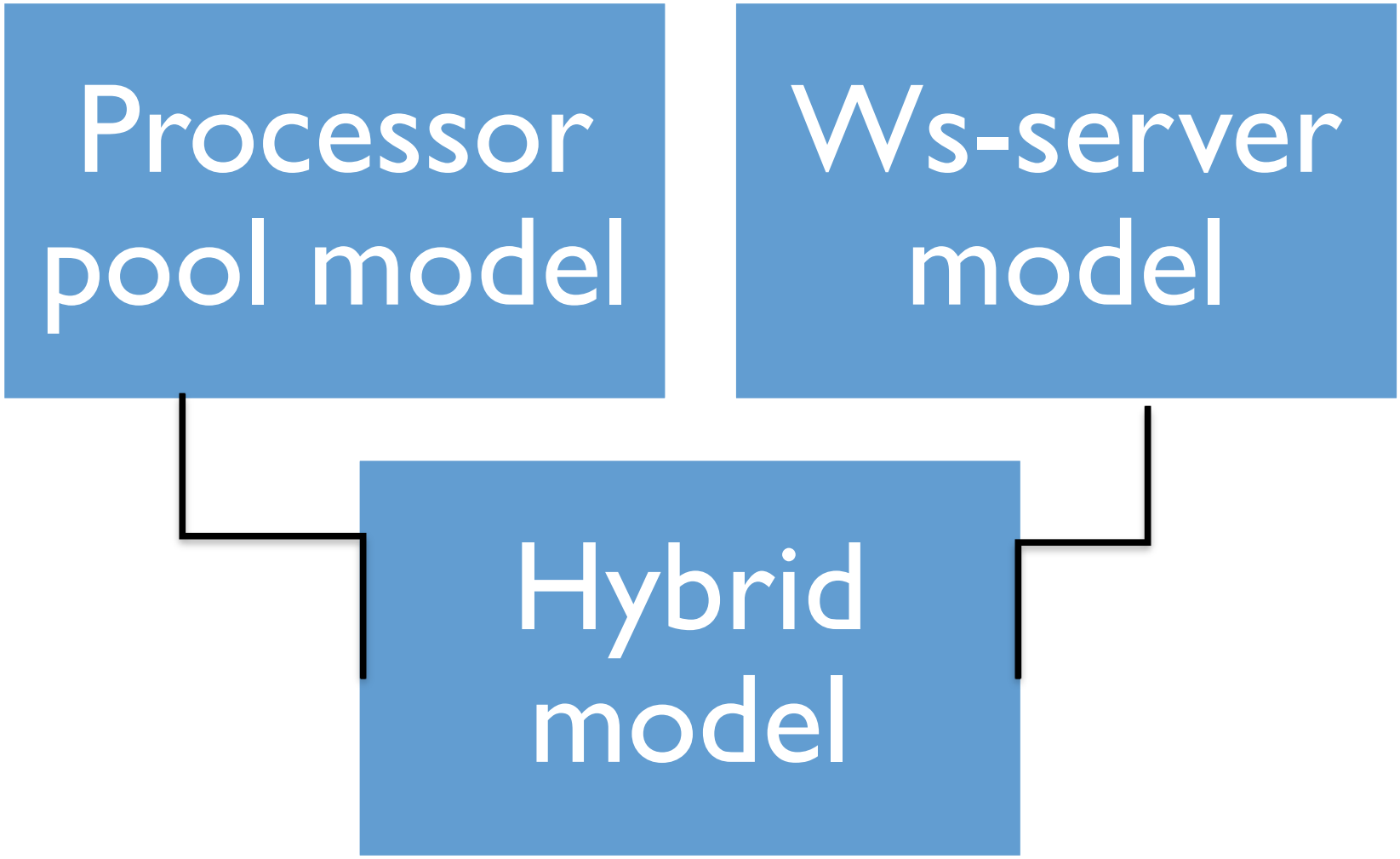
Because

Suitable for interactive jobs

Processor-pool model is suitable for

Massive applications which need computations

Hybrid Model



Distributed Systems & Its Popularity

- Traditional centralized systems
- Distributed systems

DS are difficult to design and implement

- Effectively using the resources
- Security and Communication is a problem
- Performance is network dependent

Distributed Systems & Its Popularity

- Special softwares needed to
  - Handle loss of messages
  - Prevent overloading of messages
  - Handle shared resources
- Inherently Distributed systems
- Air line reservation system
- Banking systems

Distributed Systems & Its Popularity

Information Sharing among Distributed Users

- Efficient Person-Person Communication
- Sharing information over great distances
- Eg Proje
- Computer Supported co operative work(CSCW) or groupware

Resource Sharing

- Sharing of S/W libraries and database

Distributed Systems & Its Popularity

Better Price-Performance Ratio

Important reason for popularity

Increased power

Decreased price of processors

Increasing speed of networks

Resource sharing

Distributed Systems & Its Popularity

Shorter Response time and higher Throughput

- Response time
- Throughput

Higher Reliability

- Degree of tolerance against errors
- Increased Availability
- But reliability comes at the cost of performance

Distributed Systems & Its Popularity

Extensibility and Incremental Growth

Gradually extended to include resources

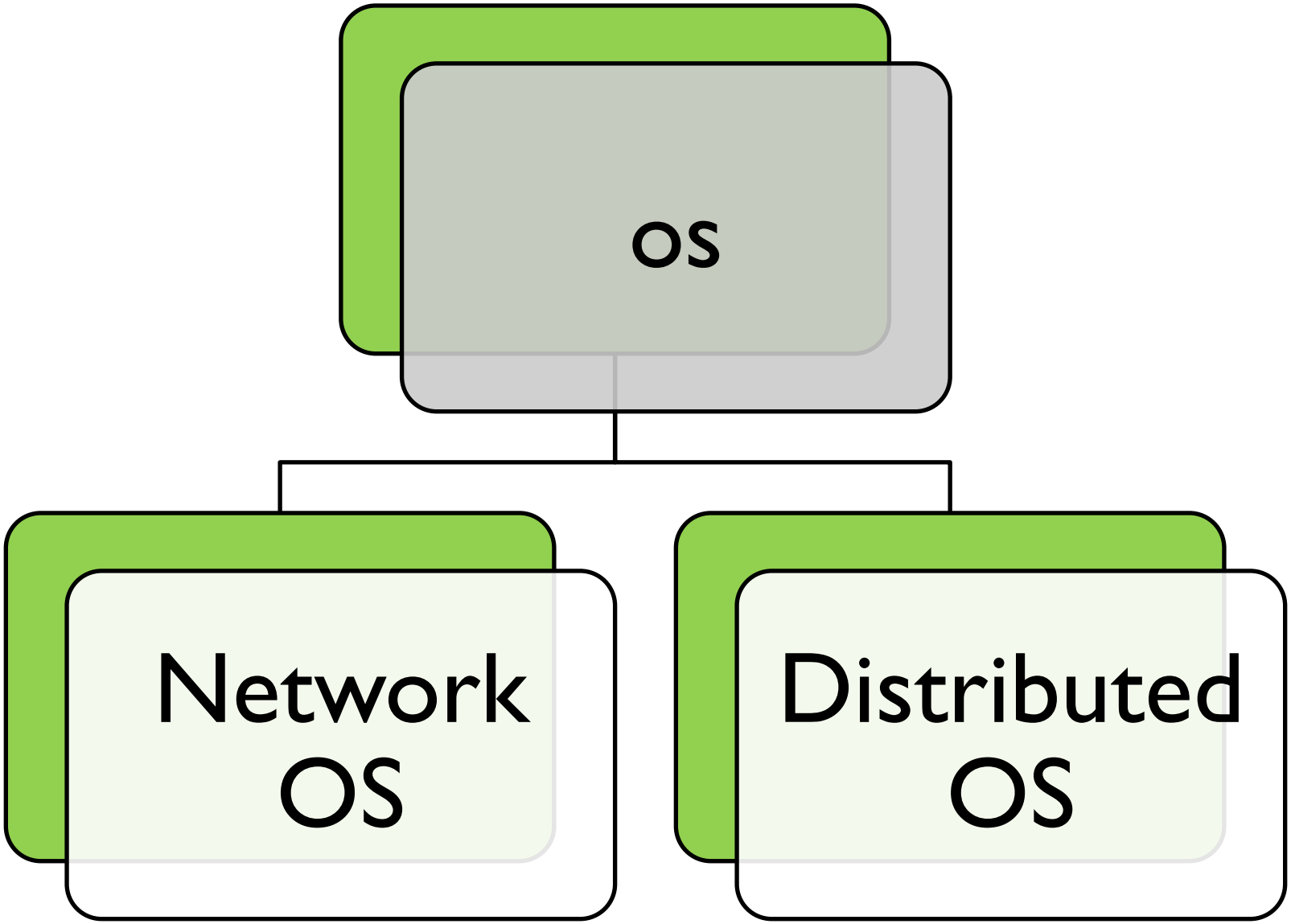
Called open Distributed Systems

Better Flexibility

Combination of different types of computers

Flexible to do any task

What is a Distributed OS?



Differences

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 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

ISSUES IN DESIGNING DISTRIBUTED OS

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

  A distributed OS cannot have Complete information about the system is not available
- Centralized Os- Resources are nearer

  Distributed Os-Faraway
- Centralized Os- Common Clock

  Distributed OS- No Common clock, Lack of UP to date information

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.

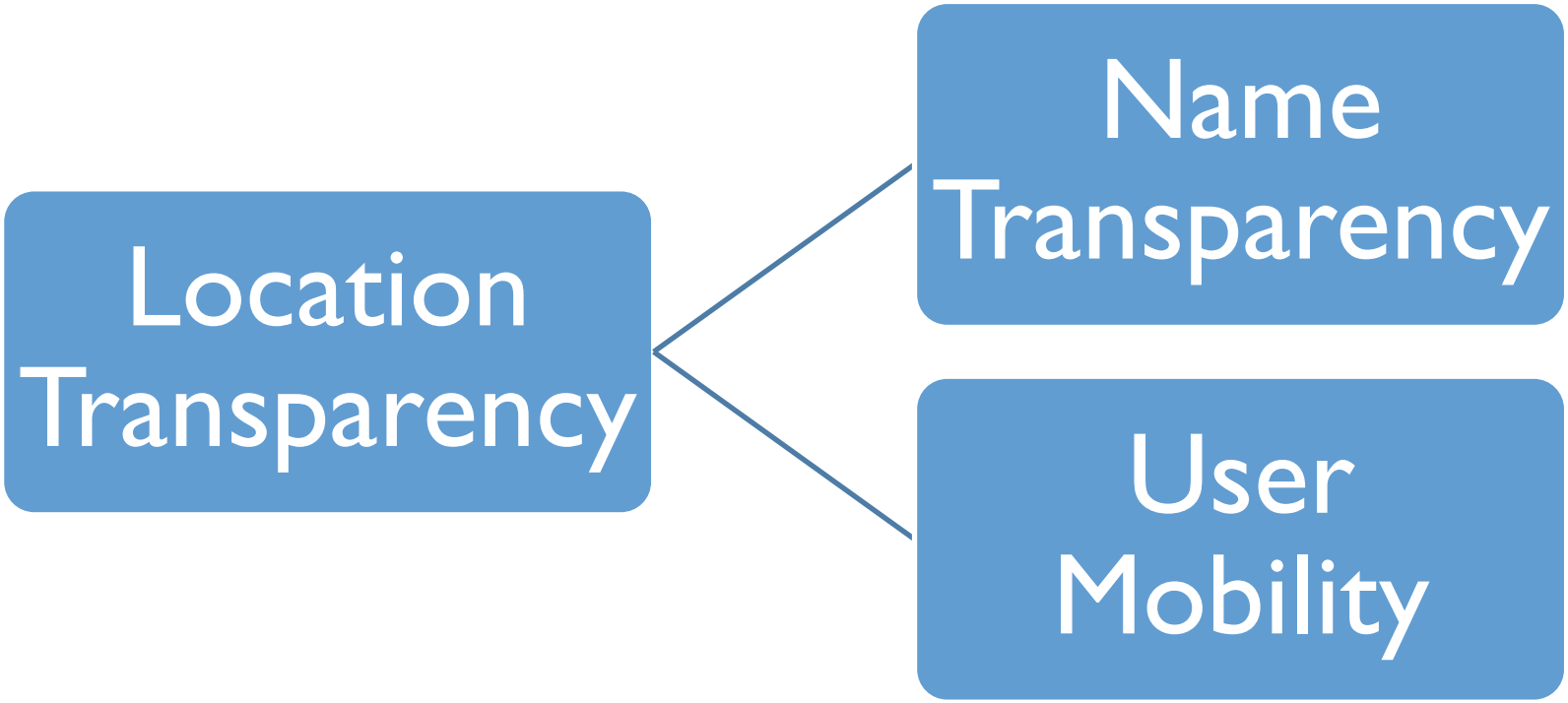
Transparency

- Single Virtual Uniprocessor image
- Eight forms of transparency
  - Access transparency
  - Location transparency,
  - Replication transparency,
  - Failure transparency,
  - Migration transparency
  - Concurrency transparency,
  - Performance transparency, and
  - Scaling transparency

Transparency

- Access transparency
- user cannot recognize a resource local or remote
  - Remote resources in the same way as local
  - Uniform system calls
  - Distributed shared memory concept
  - Suitable for limited types
  - Performance limitation

Transparency



Transparency

Name Transparency

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

User Mobility

- Same name for accessing from different locations
- Users access without any extra effort

Transparency

- Replication Transparency
  - Creating Replicas of files on another systems
  - Should be transparent
    - Issues
      - Naming of replicas and
      - Replication control.

Transparency

- Failure Transparency
  - 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 possible

Transparency

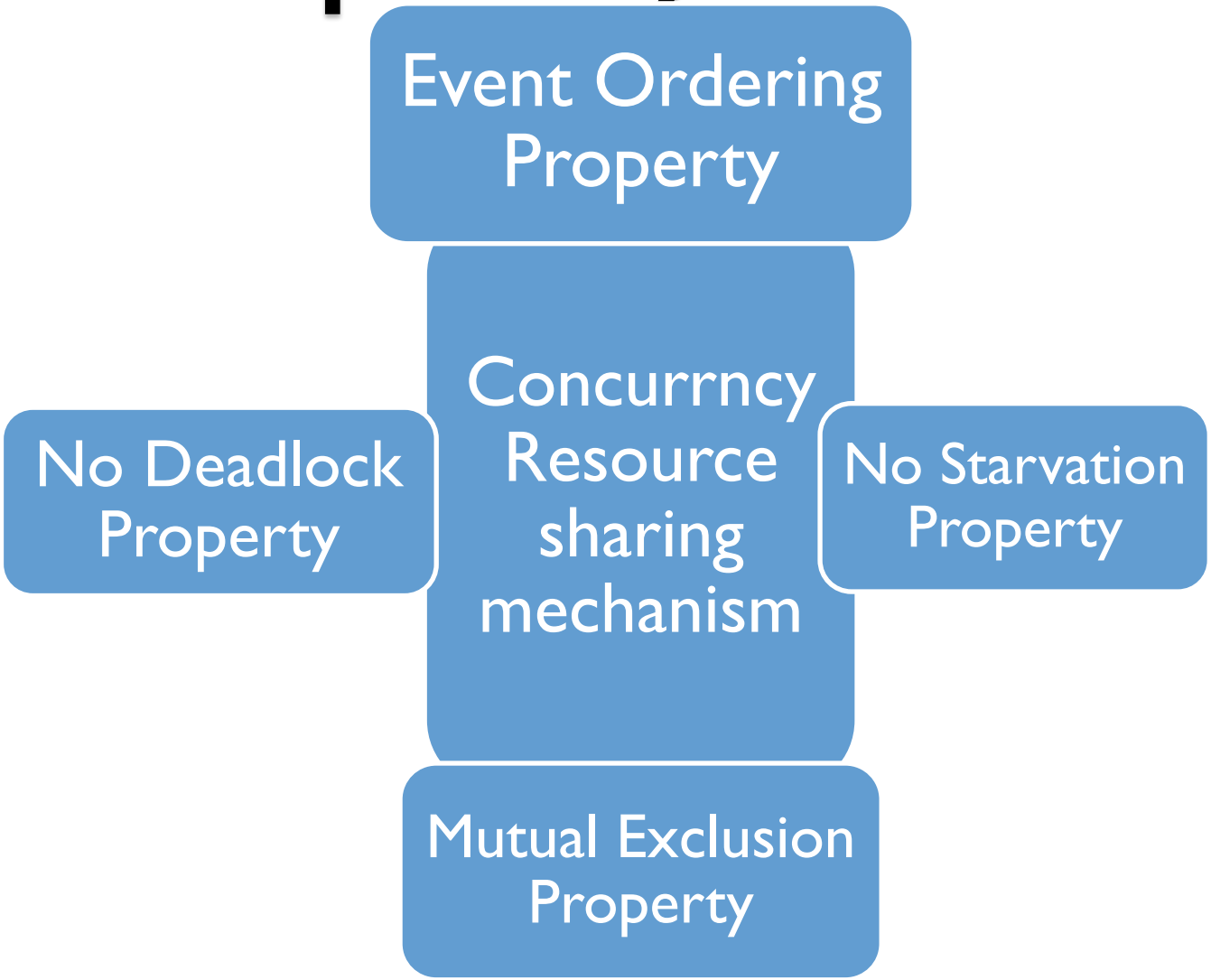
Migration Transparency

- Migration decisions should be automatic
- Migration should not require any change in name
- If the receiver moves to another location the sender need not resend it.

Concurrency Tranparency

- Using the system concurrently
- Concurrent update of the same file should not be allowed

Transparency



Transparency

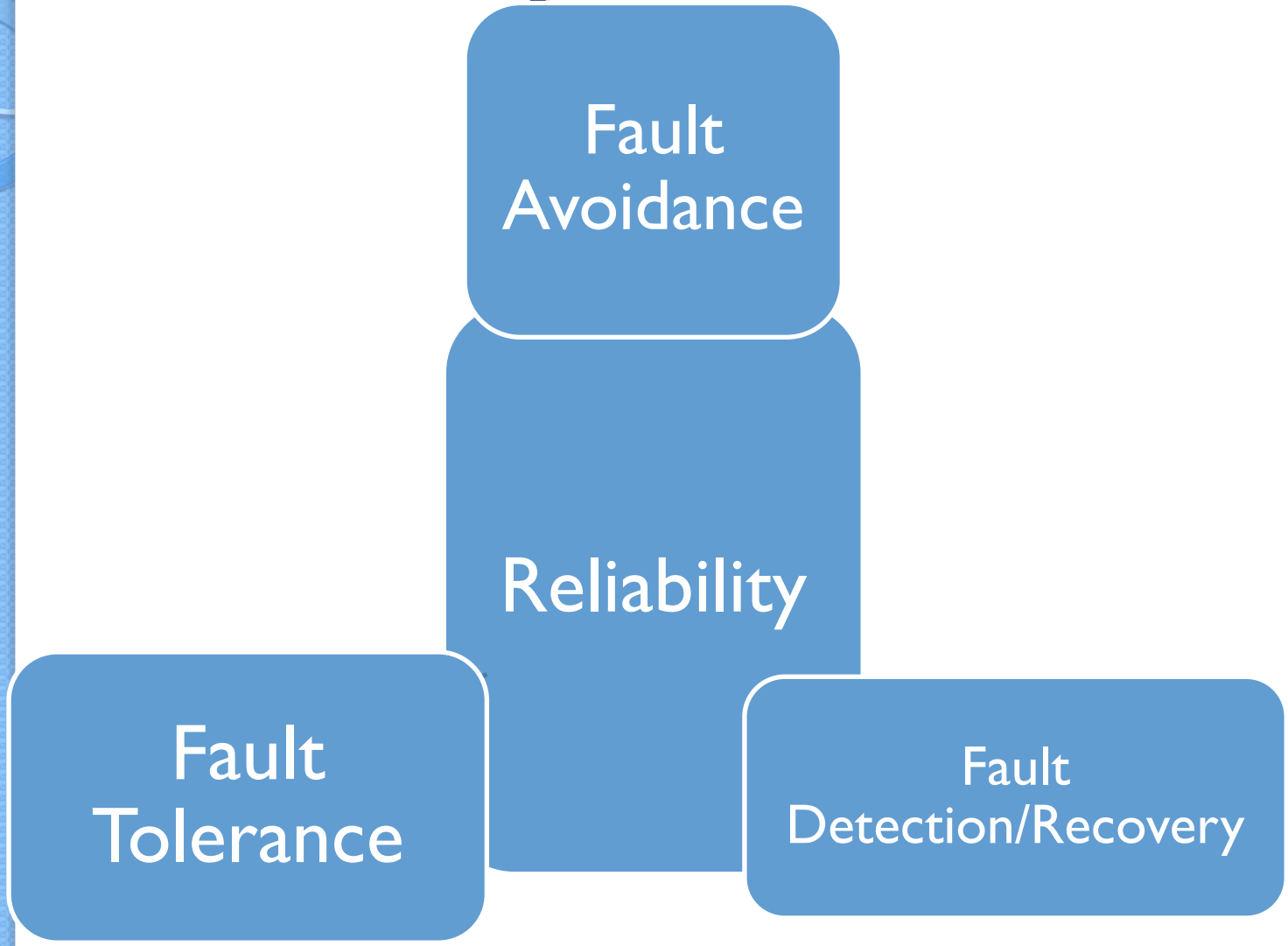
Performance Transparency

- Loads vary dynamically
- Processors should be uniformly distributed
- To allow system to reconfigure
- Intelligent resource allocation and migration facility

Scaling Transparency

- To allow system to expand
- Open System Architecture
- Use of scalable algorithms

Reliability



Reliability

Fault Avoidance

- To design to minimize the occurrence of faults
- The Designers should test.

Fault Tolerance

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

Techniques to improve fault tolerance

Redundancy Techniques

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

Reliability

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

Distributed Control

- Distributed control mechanism to avoid single points of failure.

1. Fault Detection and Recovery

Use of H/W and S/W to detect failure and recover from it.

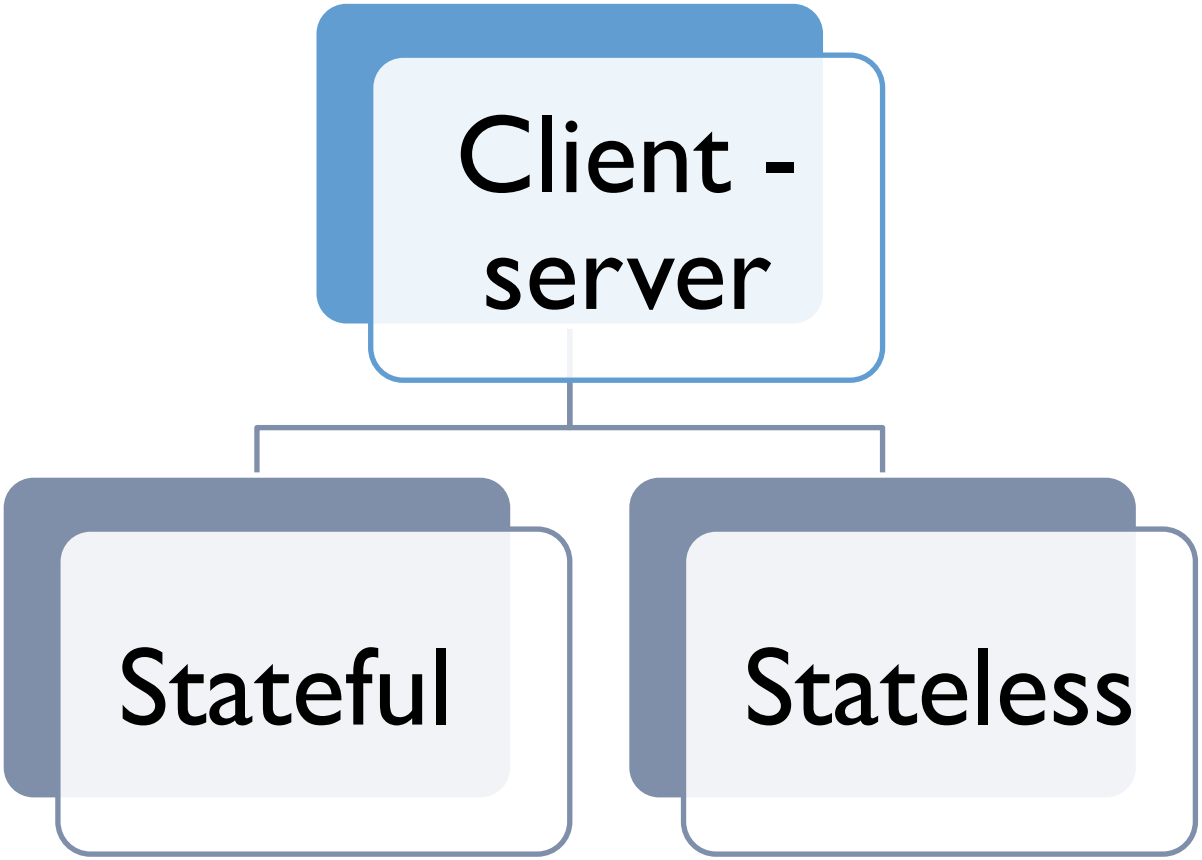
Techniques:

Atomic Transactions

Collection of operations that takes place in a failure

Reliability

2. Stateless Servers



Reliability

3. Acknowledgements and time-outs based retransmissions
   - Duplicate messages are a problem here
   - Detection and handling of Duplicate messages
   - Generating and assigning Sequence Numbers
   - Extra Overhead to detect these
   - Integrate all these things in a cost-effective manner

Flexibility

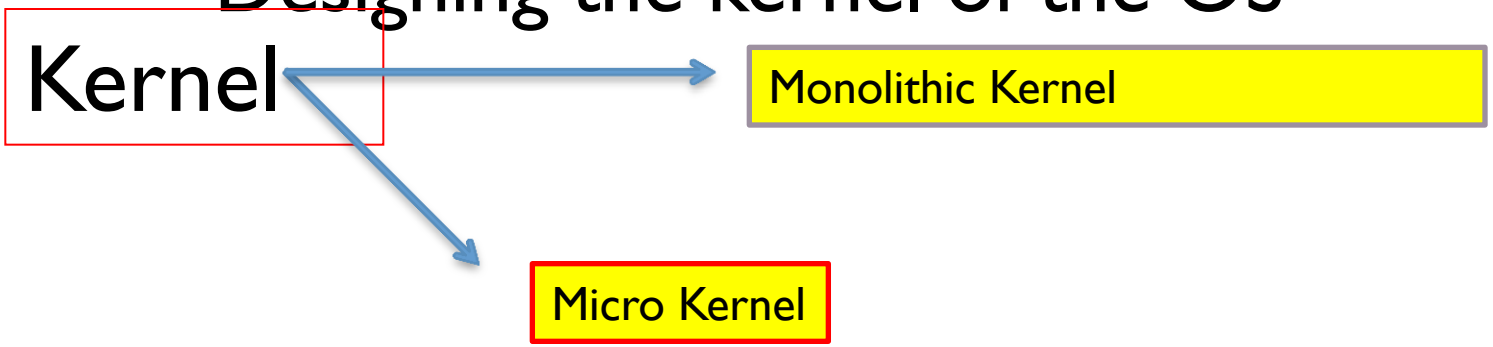
- Why Di s.Os should be flexible?

  Ease of modification

  Ease of enhancement

The important design factor is

Designing the kernel of the OS



Flexibility

- Monolithic Kernel
  - All functions are provided by such kernel
  - Big structure
  - UNIX
- Micro Kernel

To Keep Kernel as small as possible

OS provides minimum facilities

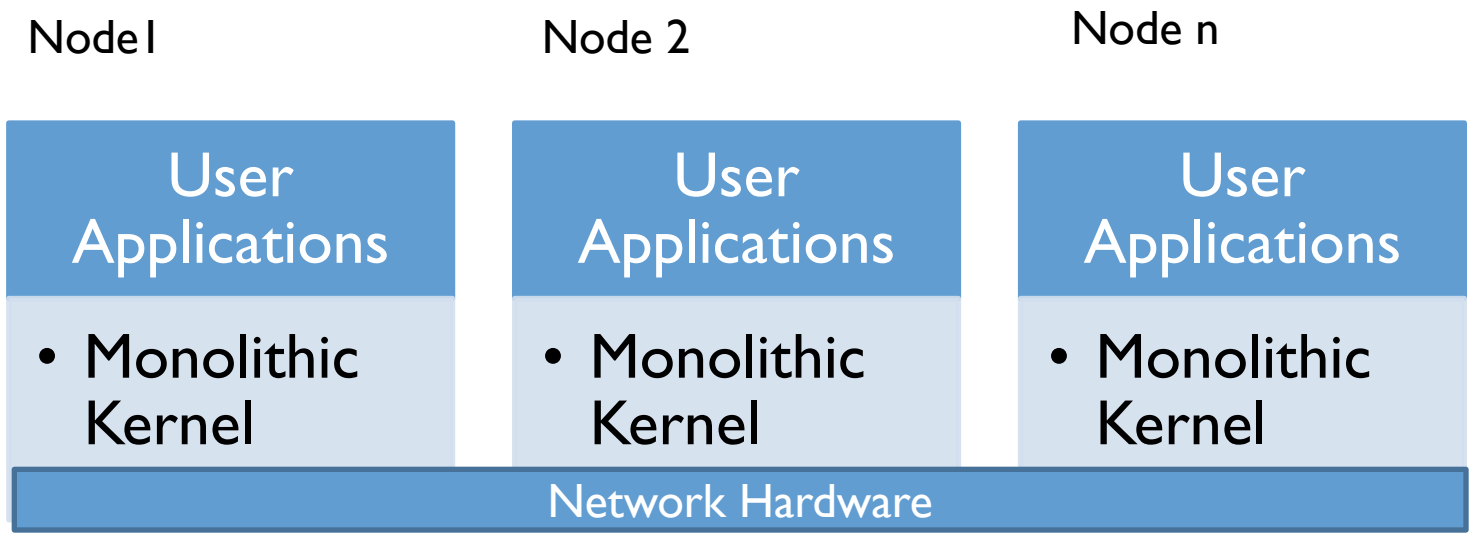
Services provided is inter process communication

Low Device mgmt

Mem. Management

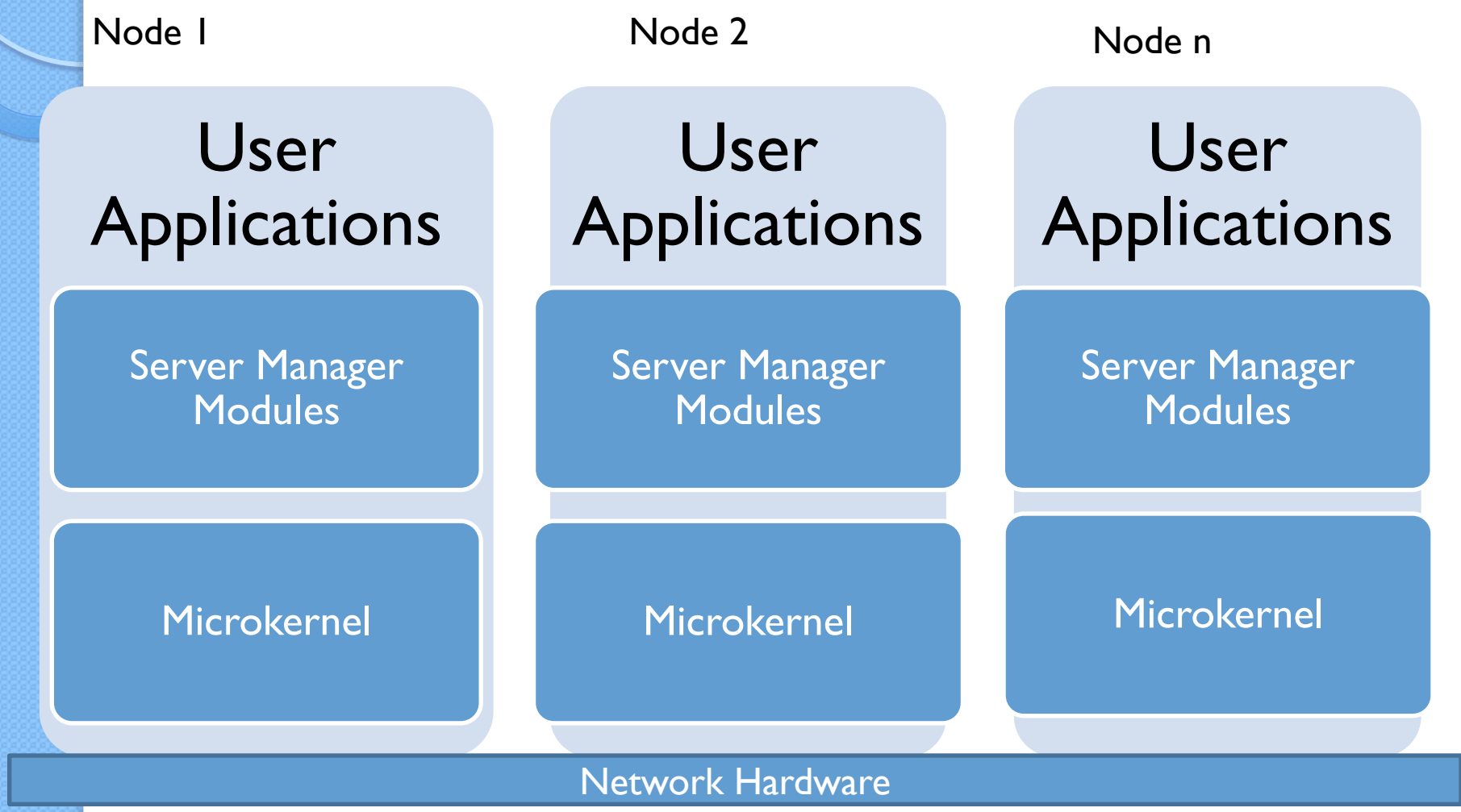
other services as user level server processes.

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 pages transfer instead of small
  - Piggybacking acknowledgement
- 2. Cache whenever possible
- Makes data available
- Saving large amount of Computing time and bandwidth
- 3.Minimize copying data
  - Data path
  - Senders Stack → Message buffer → Kernels
    message buffer
  - Network Interface Board ← ↓

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**
  - **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

I. Avoiding centralized entities

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 increases network traffic.

The increased users increases the complexity

Scalability

- ***2. Avoid centralized algorithms.***
  - ***This algorithm collects information from all nodes***
  - **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 network**

Heterogeneity

- Interconnected set of dissimilar hardware and software
- The following things are different
  - Communication protocols
  - Topologies
  - Servers

We need translation servers

Intermediate standard data format

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
  - Whether message received by the receiver
  - Message sent by a genuine sender
  - The content of the message is not altered

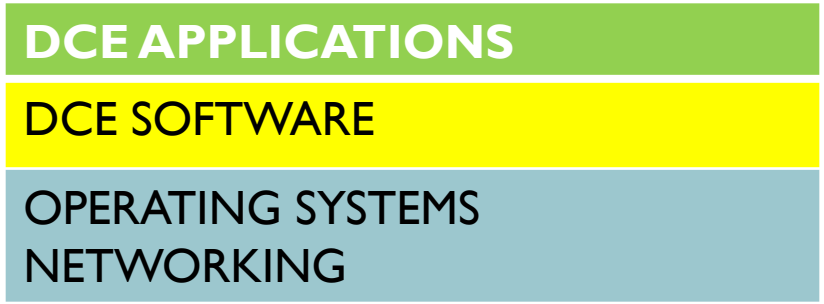
Cryptography and integrity(Trusting smaller servers rather than clients)

Emulation of Existing Operating Systems

- New OS should allow the old features of old OS also

DISTRIBUTED COMPUTING ENVIRONMENT (DCE)

- it is an integrated set of services and tools that can be installed as a coherent environment on top of existing operating systems and serve as a platform for building and running distributed applications.



DCE Creation

- Middleware software layer
- Request for Technology

DCE components

Threads Package

Provides a simple model for concurrent applications

RPC facility

Basis for all communication facility

Easy to use

Network independent

Protocol independent

Provides Secure Communication

DCE Creation

- Distributed Time Service
  - Synchronizes clocks of all systems
    - Permits the use of time values
    - Clocks of one DCE can be synchronized with the other
  - Name Services
    - Cell Directory Service(CDS)
    - Global Directory Service(GDS)
    - Global Directory Agent(GDA)
    - Allow resources such as servers, files and devices files with unique name.
- Security Service
  - Provides tools needed for authentication and authorization
- Distributed File Service(DFS)
  - Provides services such as availability, transparency

DCE Components

Component name	Other components used by it
Threads	None
RPC	Threads, name, security
DTS	Threads, RPC, name, security
Name	Threads, RPC, DTS, security
Security	Threads, RPC, DTS, name
DFS	Threads, RPC, DTS, name, security

Reference:Pradeep K.Sinha, Distributed operating System Concepts and Design

DCE CELLS

- DCE uses the concept of cells.
- Breaks down a large system into smaller, manageable units called cells
- a *cell* is a group of users, machines, or other resources that typically have a common purpose and share common DCE services.
- The minimum cell configuration requires
  - cell directory server,
  - security server,
  - a distributed time server,
  - and one or more client machines.

DCE CELLS

- **Boundaries**

Purpose:

- **Machines of common goal will be put in the same cell.**
- **Product oriented or function oriented cells**

Administration: Each system needs an administrator

All the Machines and administrators are put in same cell

- **For example, all machines**
- **belonging to the same department of a company or a university can belong to a single cell.**
- **From an administration point of view,each cell has a different administrator.**

DCE CELLS

Security

- **users who have trust in each other should be put in the same cell.**
- **In such a design, cell boundaries act like firewalls in the sense that accessing a resource that belongs to another cell requires authentication than accessing a resource that belongs to a user's own cell**

Overhead.

- **machines of users who frequently interact**
- **and the resources frequently accessed by them should be placed in the same cell..**

Distributed Computing System

- A collection of processors inter connected by a communication network in which each processor has its own local memory and other peripherals and communication between any two processors of the system takes place by message passing over the communication network

Why Di-OS?

- (a) Suitability for applications which are distributed in nature
- (b) Sharing of information
- (c) Sharing of resources,
- (d) Better Performance-price ratio,
- (e) Shorter response times
- (f) Higher throughput
- (g) Higher reliability,
- (h) Extensibility and incremental growth
- (i) Flexible in meeting users' needs

MESSAGE PASSING

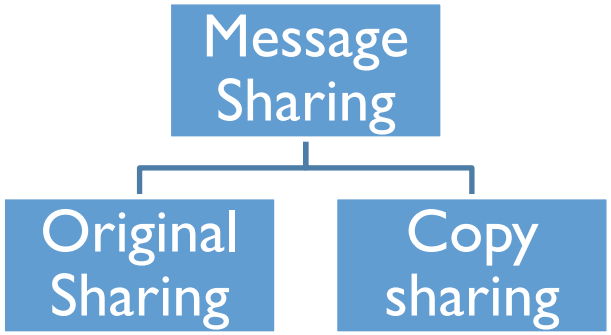
Introduction

Processes communicating with each other

- Distributed operating systems provide IPC

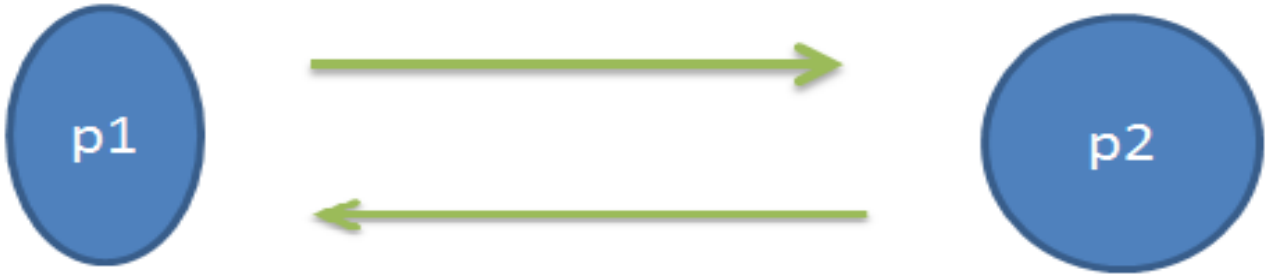
  Methods for sharing information are

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



Basic Methods for sharing data

Original Sharing



Copy sharing

Message Passing System

subsystem of a distributed operating system

- provides a set of message-based IPC protocols.
- enables processes to communicate by exchanging messages
- simple communication primitives, such as *send* and *receive*.

Desirable Features of a Good Message Passing System

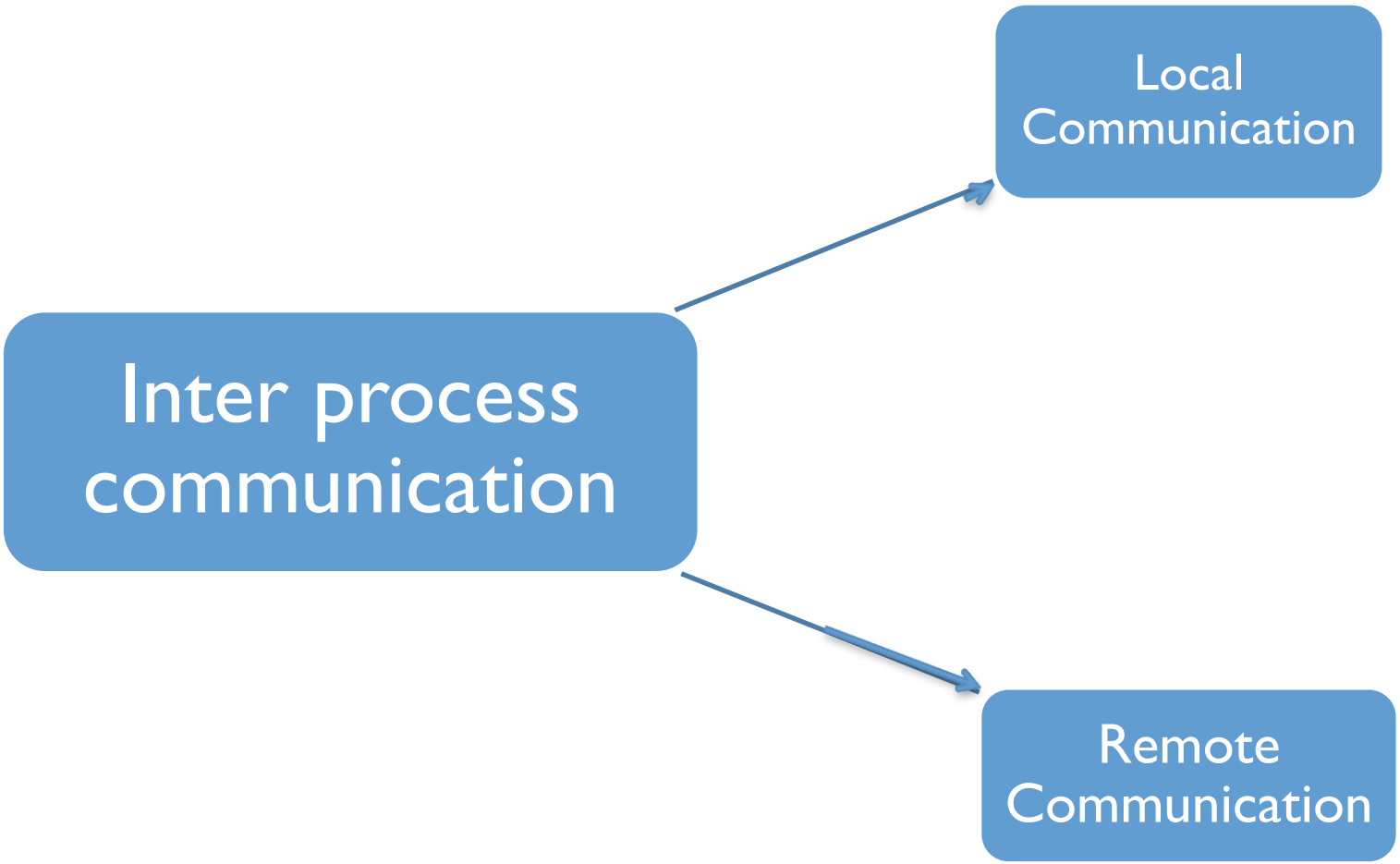
- Simplicity
- Uniform Semantics
- Efficiency
- Reliability
- Correctness
- Flexibility
- Security
- Portability

Simplicity

Message Passing System should be

- Simple
- Easy to Use
- Straight forward to construct new applications
- Clean and Simple Semantics of IPC Protocols

Uniform Communications



Efficiency

If MPS not Efficient is cost increases

- Avoiding the costs of establishing and terminating connections
- Minimizing the costs of maintaining the connections
- Piggybacking of acknowledgment of previous messages with the next message

Reliability

- Acknowledgments and retransmissions on the basis of timeouts
- Detecting and handling duplicates
- Generating and assigning appropriate sequence numbers to messages

Correctness

Issues related to correctness are

- Atomicity → Message sent to every receivers
- Ordered delivery → Messages are in sequence
- Survivability → Messages will be delivered in spite of failures.

Survivability is a difficulty property to achieve.

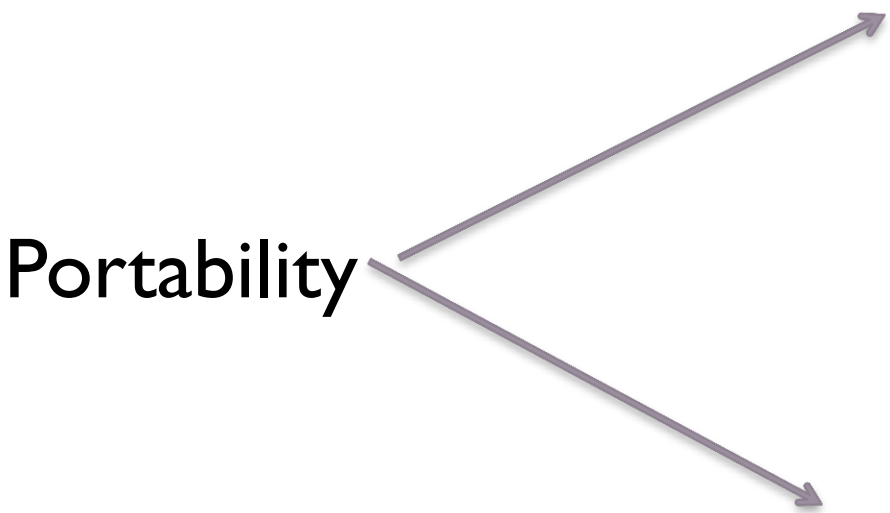
Flexibility

- IPC primitives should be such that the users have the flexibility to choose and specify the types and levels of reliability
- have the flexibility to permit any kind of control flow between the cooperating processes, including synchronous and asynchronous *send/receive*.

Security

- Secure end-to-end communication.
- Authentication of the receiver of a message by the sender
- Authentication of the sender of a message by its receiver
- Encryption of a message before sending it over the network

Portability



Message passing should be portable

The applications should be portable

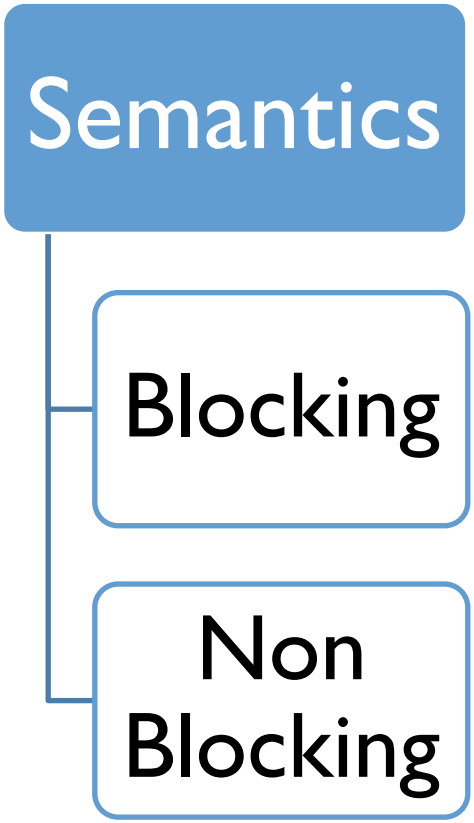
Issues in IPC Message Passing

Header and messages

Actual Data or Pointer to data	Structral information		Sequence No	Addresses	
	Number of bytes	Type		Sending address	Receiving Address

Who is the sender?
Who is the receiver?
One or more receivers?
Message guaranteed?
Sender waits for reply?
Node crash? What to do?
Buffering?
Outstanding Messages?

SYNCHRONIZATION

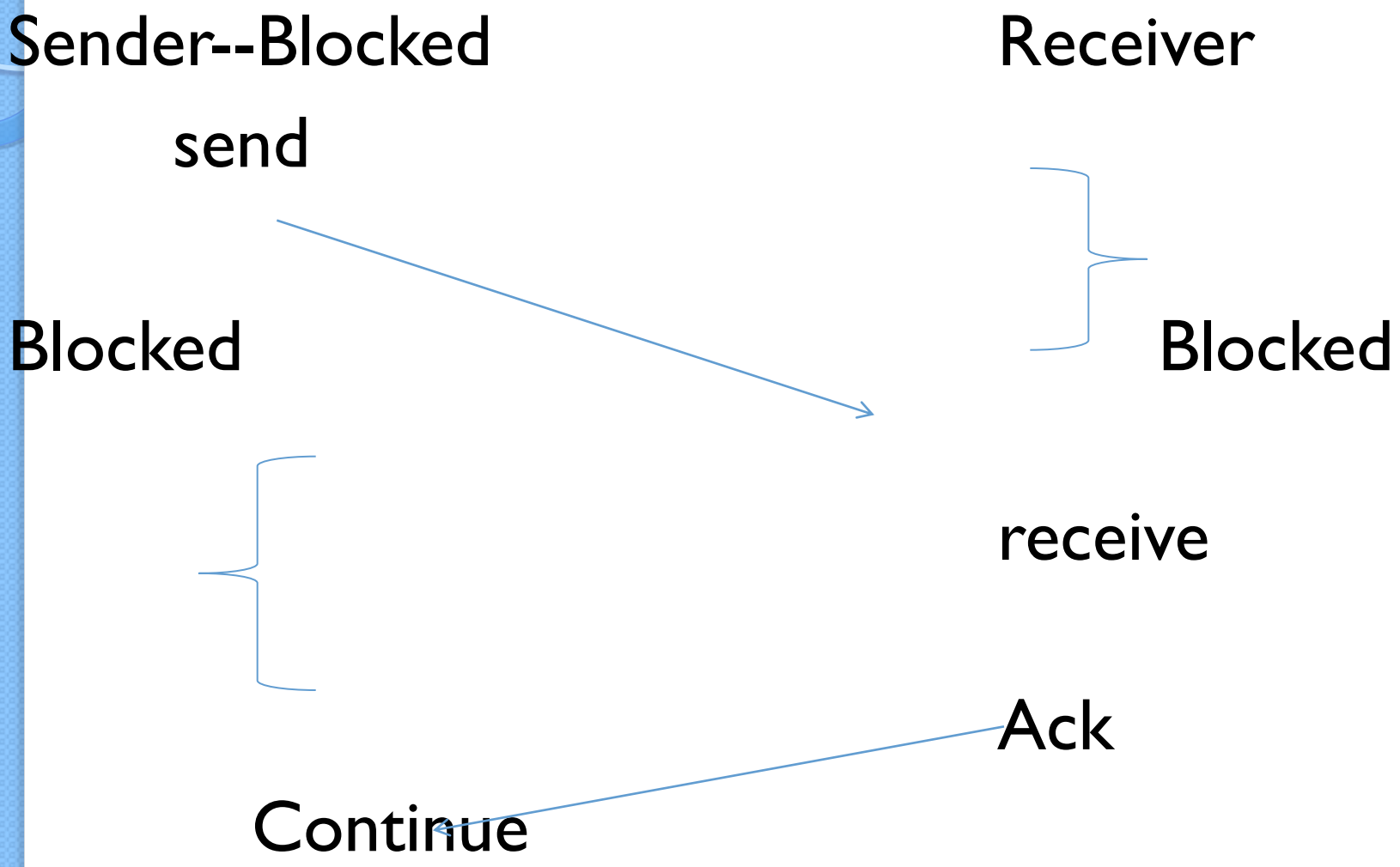


Synchronization → Communication Primitives

SYNCHRONIZATION

- Non blocking
  - Invocation does not does not block the execution of the invoker
- Blocking
  - blocks
- Send primitive

SYNCHRONIZATION



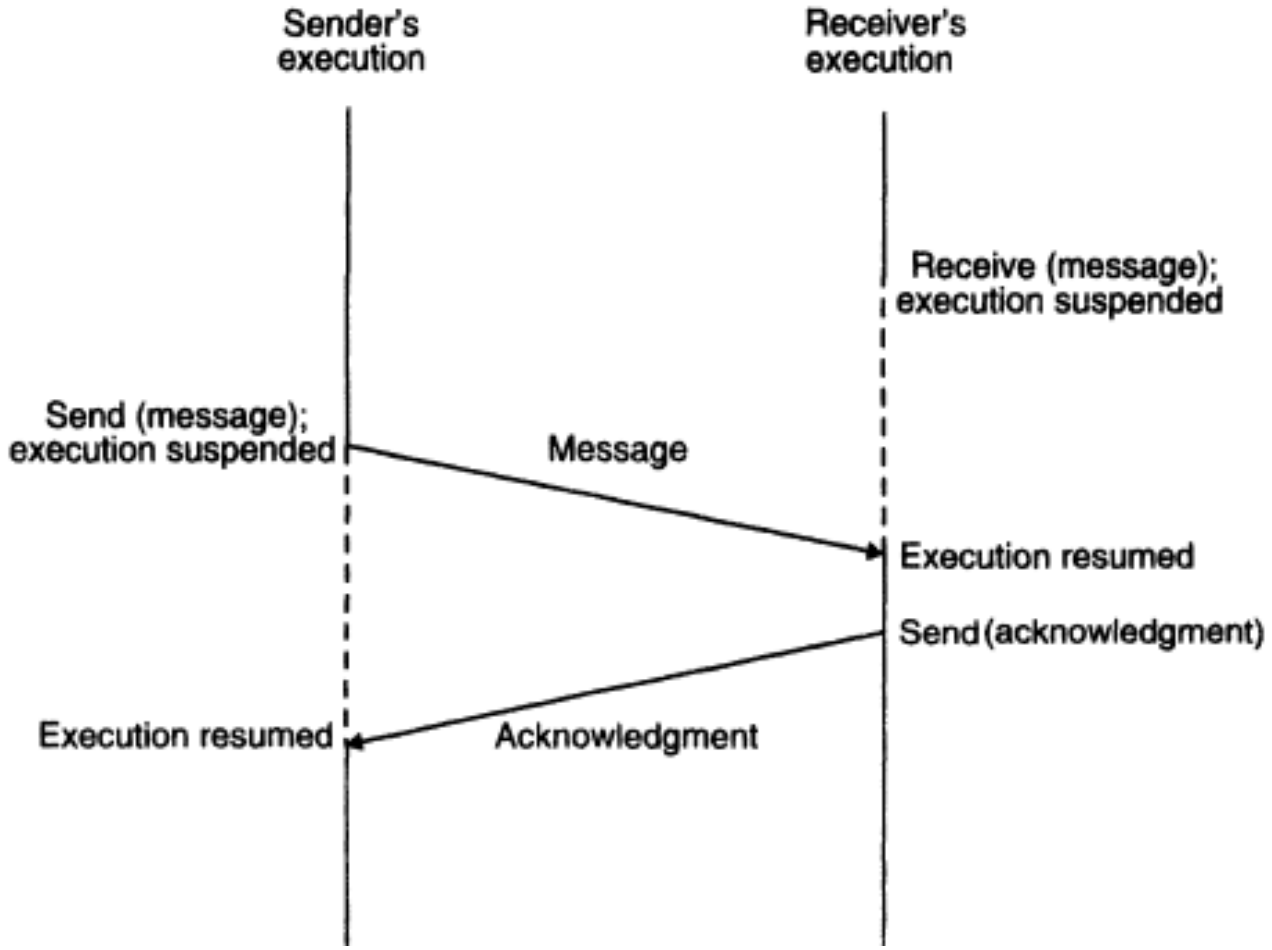
SYNCHRONIZATION

How the receive process know the message has arrived?

- Polling
- Interrupt

- conditional *receive* primitive, returns control to the invoking process almost immediately, either with a message or with an indicator that no message is available.

SYNCHRONIZATION



Ref: P.K.Sinha " Distributed Os Concepts and Design"

DIFFERENCES

SYNCHRONOUS	ASYNCHRONOUS
Simple & Easy to implement	Not a simple method
Reliable	Not reliable
If the message gets lost, no backward error recovery is required	Error recovery is needed
It limits concurrency	No such limitation
Subject to communication Deadlocks	No
Less Flexible	Flexible

Buffering

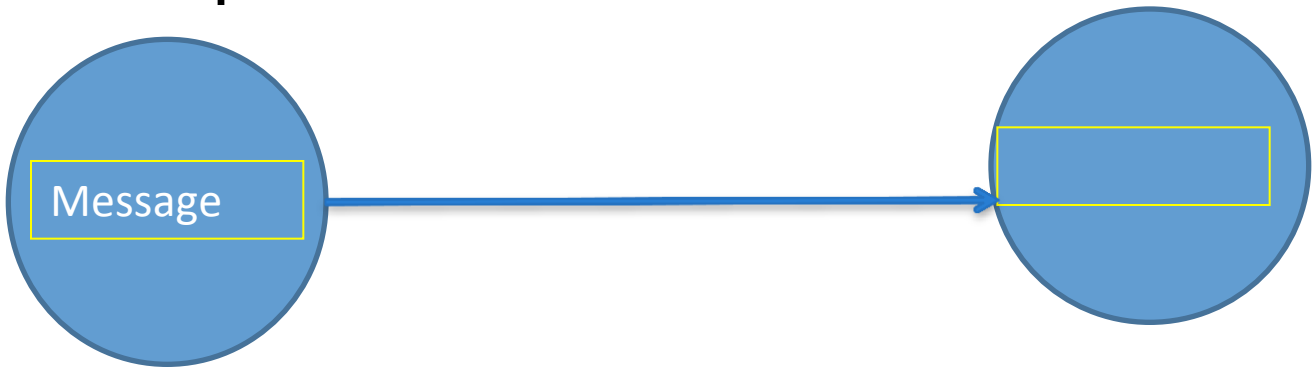
- copying the body of the message from the address space of the sending process to the address space of the receiving process.

Types

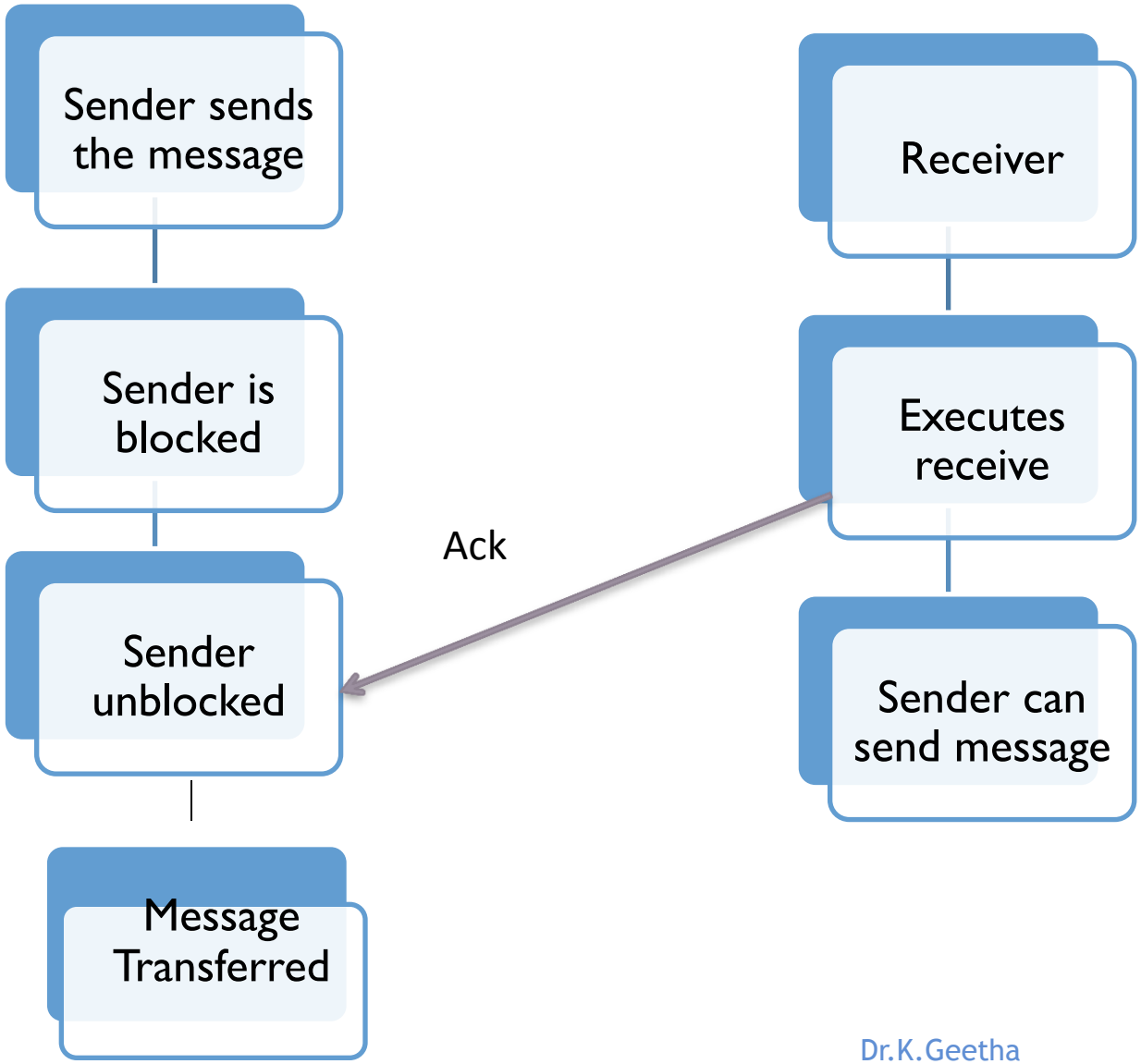
- a *null buffer*, or *no buffering*,
- and a *buffer with unbounded capacity*.
- *single-message* and *finite-bound*,
- *multiple-message*, buffers.

Buffering

- Null Buffer or No Buffering
  - No place for the temporary storage of message
  - Message remains in the senders address space until receiver executes the receive

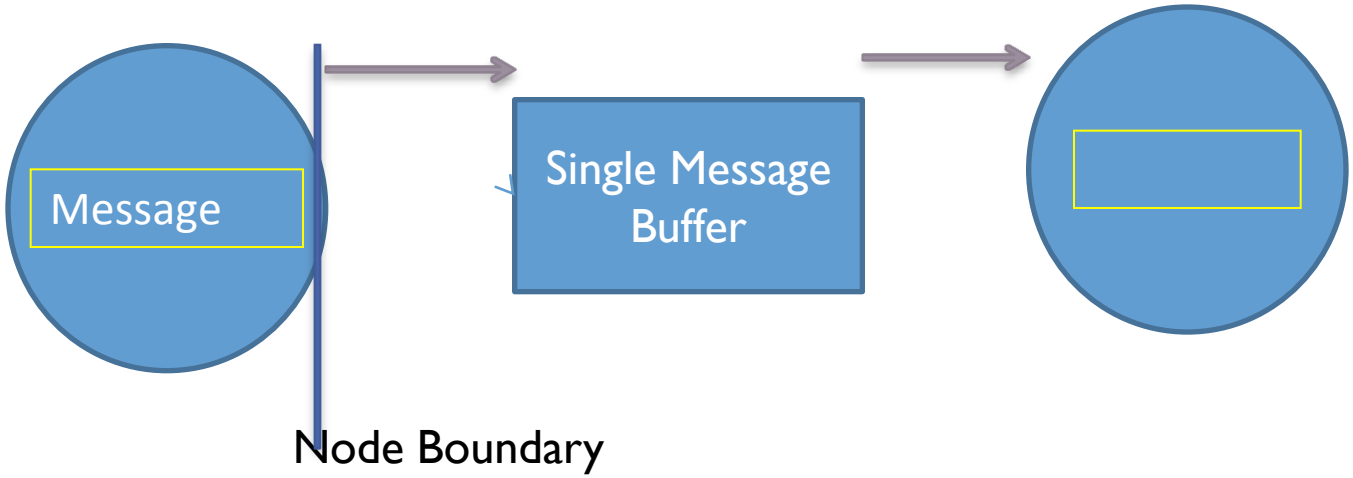


Buffering


Sender sends the message
Sender is blocked
Sender unblocked
Message Transferred
Ack
Receiver
Executes receive
Sender can send message

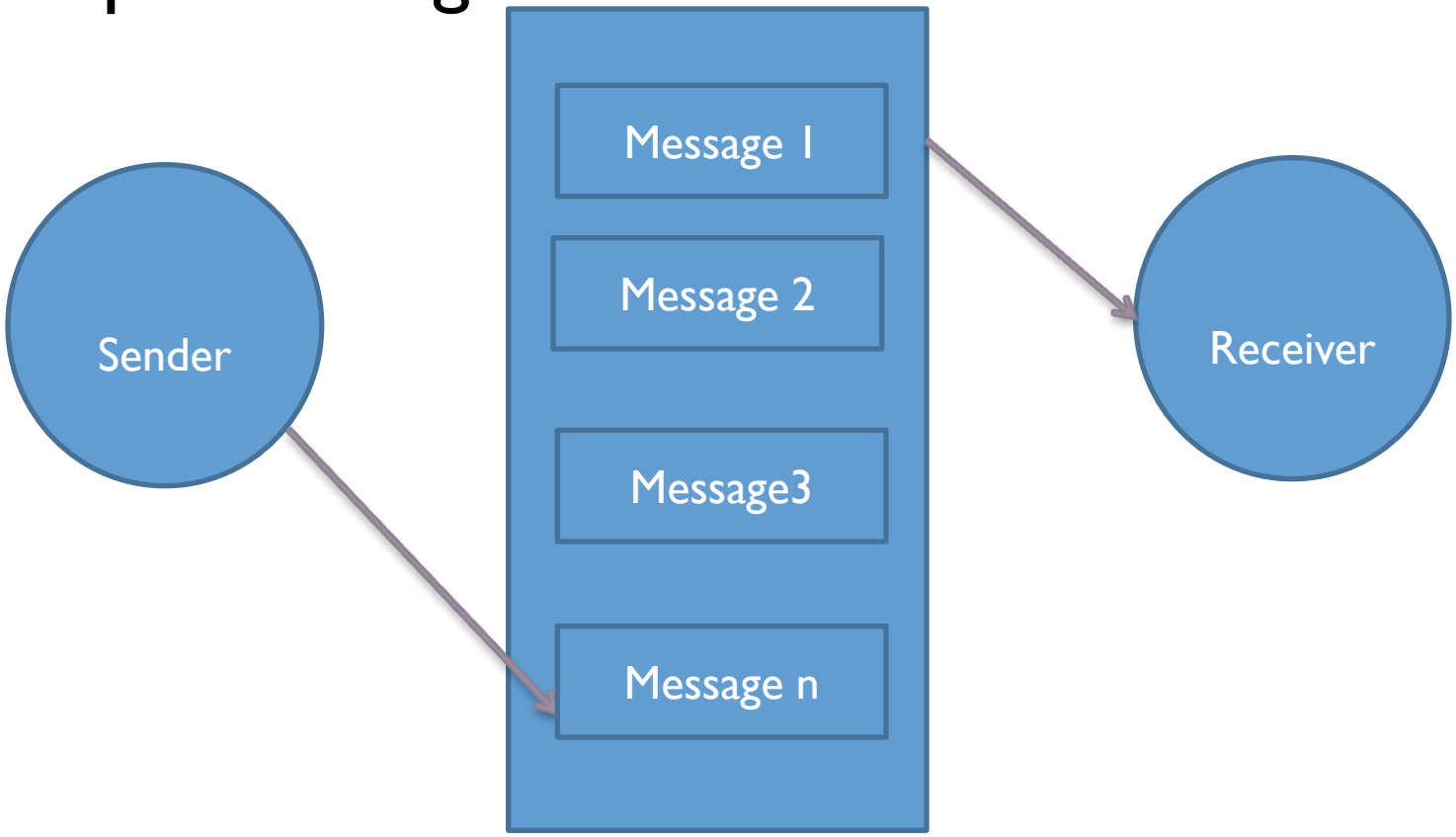
Buffering

- Single Message Buffer- suitable for synchronous

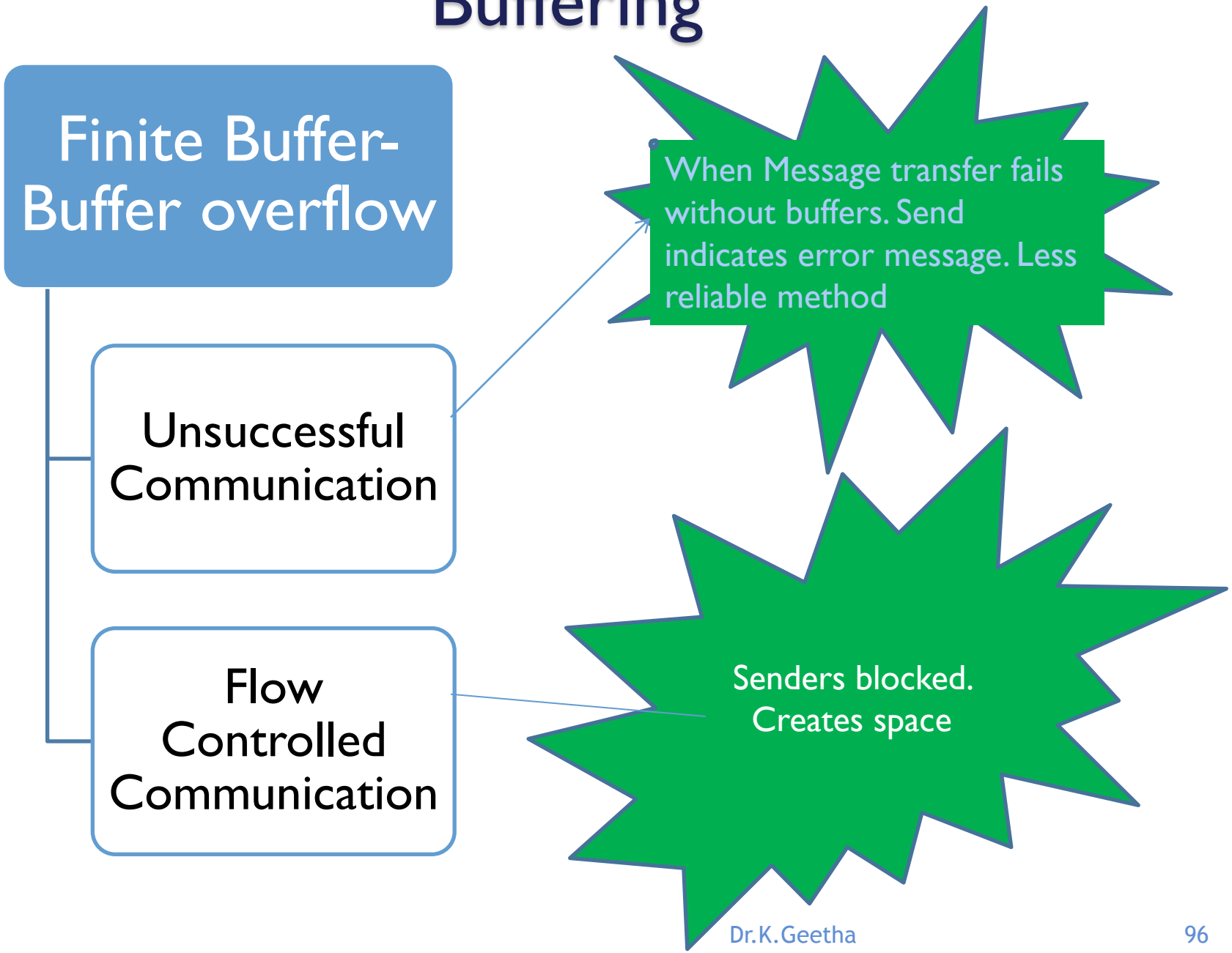


Buffering

- Multiple Message Buffer



Buffering

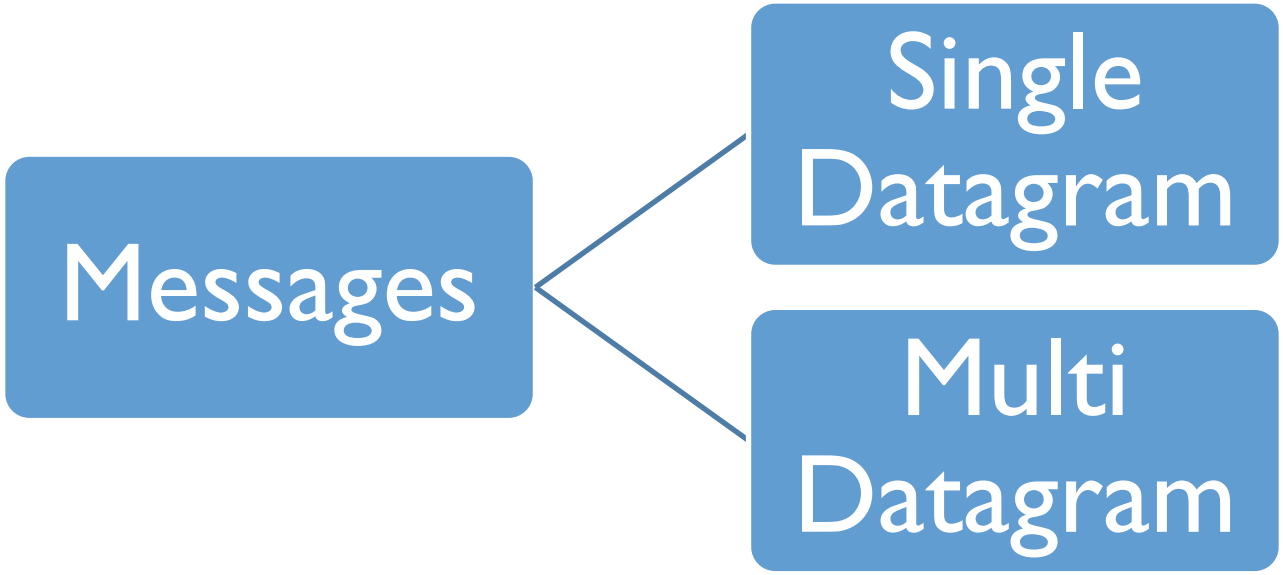

Finite Buffer-
Buffer overflow
Unsuccessful
Communication
Flow
Controlled
Communication
When Message transfer fails without buffers. Send indicates error message. Less reliable method
Senders blocked.
Creates space

Buffering

- Create_buffer system call
- Creates a buffer of size specified by receiver.
- Receivers mail box or kernels address space located
- More complex to design
- Overhead involved for creation, deletion and maintenance of buffers

MUITIDATAGRAM MESSAGES

- MTU-Maximum Transfer Unit
- If Message > MTU segmented and fragmented messages



MUITIDATAGRAM MESSAGES

- Single Datagram

Messages <MTU of the network can be sent in a single packet

Multi Datagram

Messages> MTU sent in multiple packets.

- The disassembling into multiple packets on the sender side and the reassembling on the receiver side is the responsibility of the message-passing system.

ENCODING AND DECODING OF MESSAGE DATA

- structure of program objects should be preserved while they are being transmitted
- not possible in a heterogeneous
- in homogeneous systems, *it* is very difficult to achieve this goal mainly because of two reasons

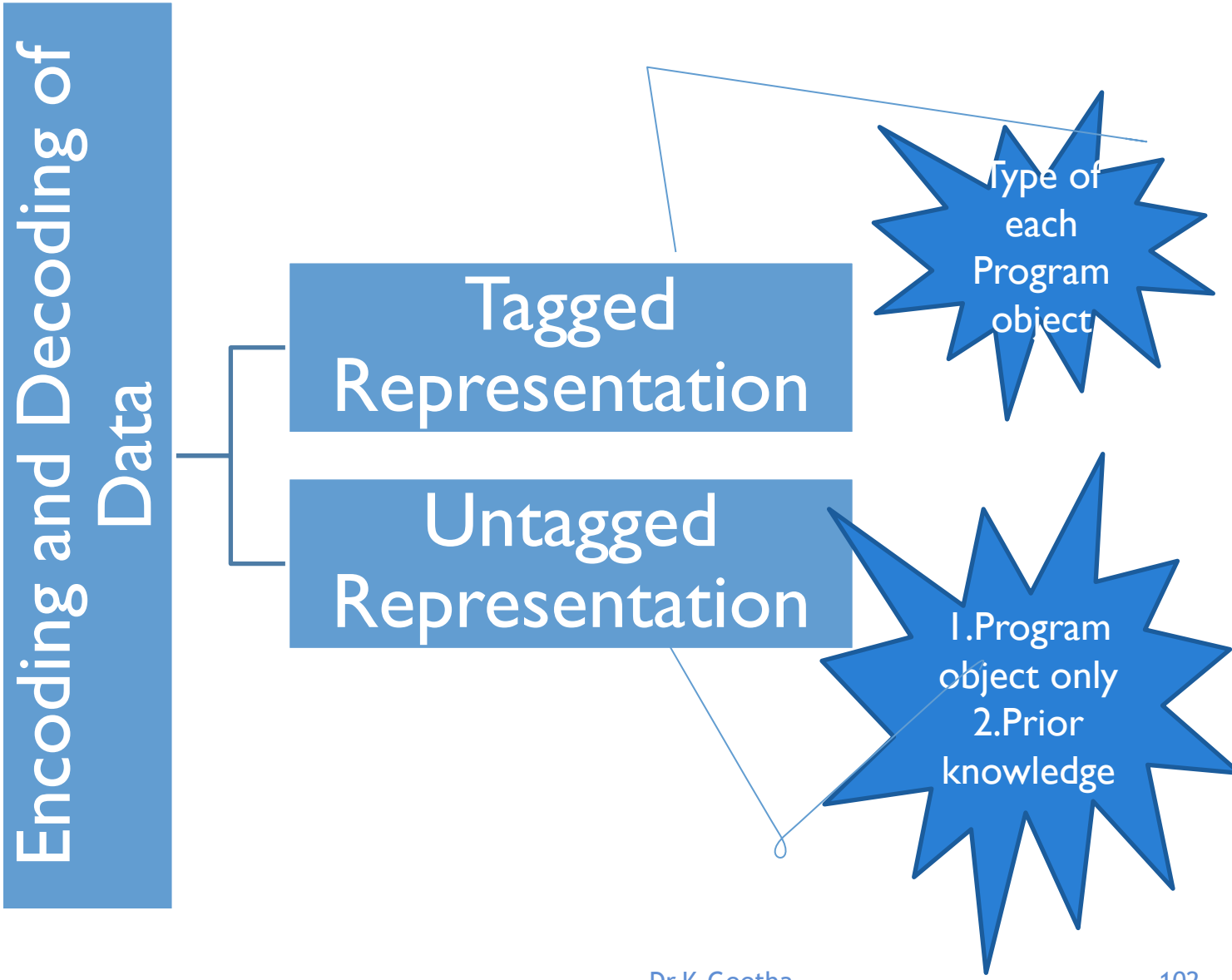
-An absolute pointer value loses its meaning during transfer. Example tree object

-Varying amount of storage space.

ENCODING AND DECODING OF MESSAGE DATA

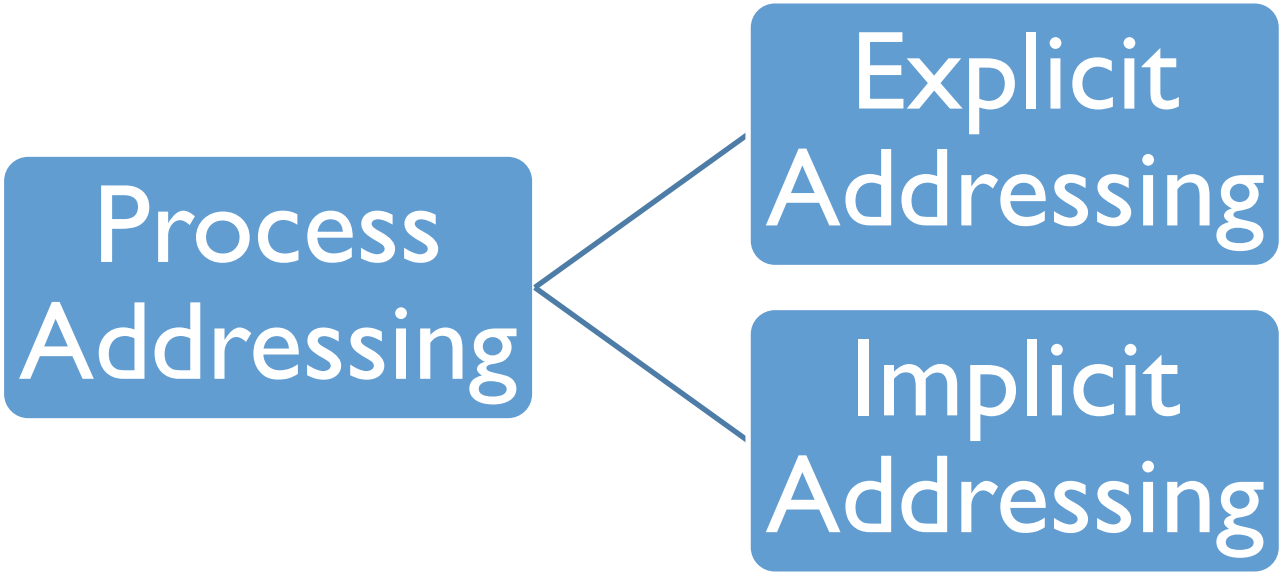
- problems are there in transferring program objects in their original form,
- they are first converted to a stream form that is suitable for transmission
- This conversion process takes place on the sender side and is known as ***encoding* of a message data.**
- This conversion process takes place on the receiver side and is known as ***decoding* of a message data.**

ENCODING AND DECODING OF MESSAGE DATA



PROCESS ADDRESSING

Naming of the Parties involved in communication



PROCESS ADDRESSING

- *Explicit addressing*. The process with which communication is desired is explicitly given as a parameter in the communication primitive used.

send (process_id, message)

receive (process_id, message)

PROCESS ADDRESSING

- *Implicit addressing. D*oes not explicitly name

  a process for communication.

Also known as functional addressing

e.g Client server communication

- send_any (service_id, message)
- receive_any (process_id, message)

PROCESS ADDRESSING

- identify a process is by a combination of *machine_id* and *local_*

id, such as *machine_id@local_id*.

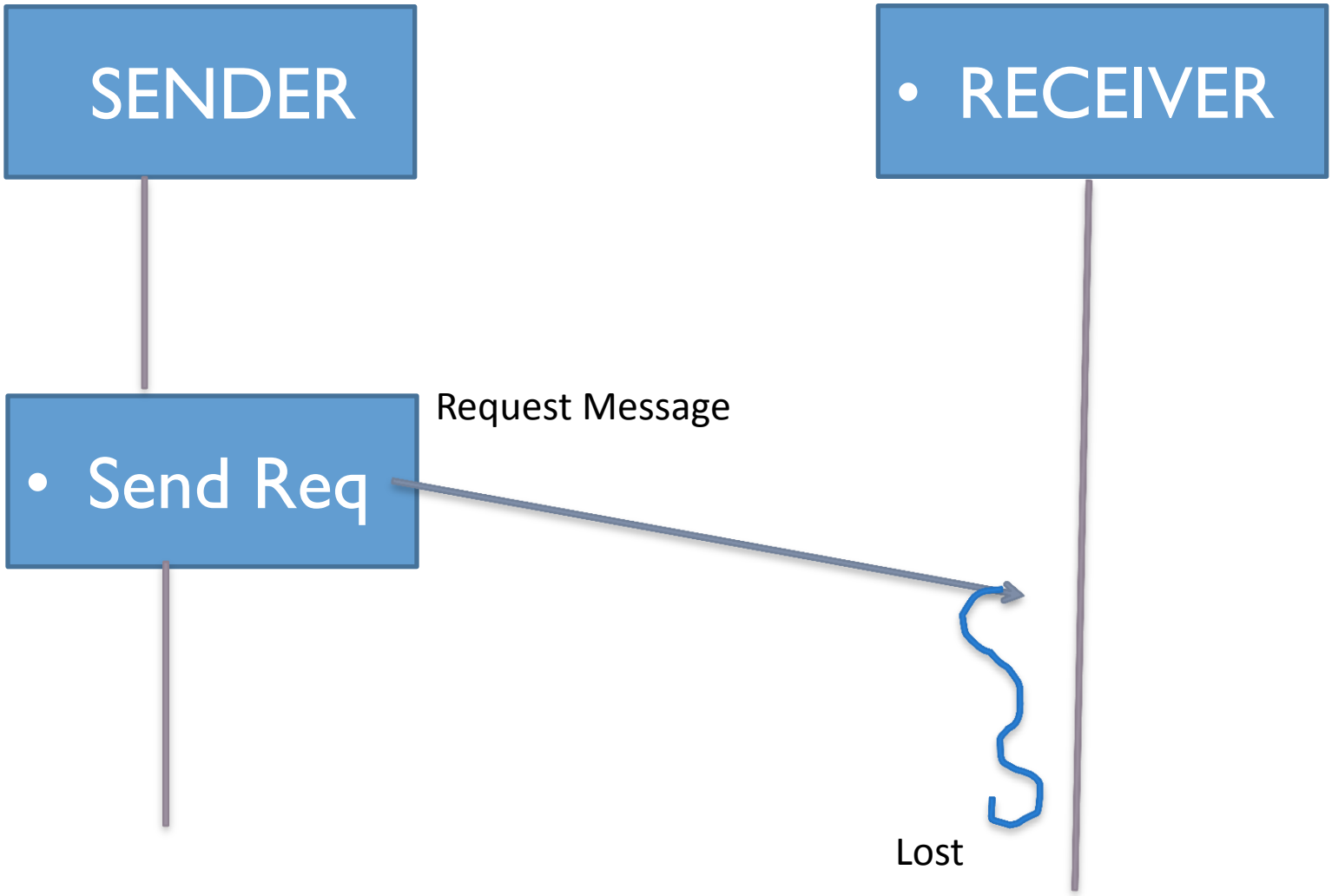
- processes can be identified by a
- combination of the following three fields: *machineld, local_id, and machineid*.

PROCESS ADDRESSING

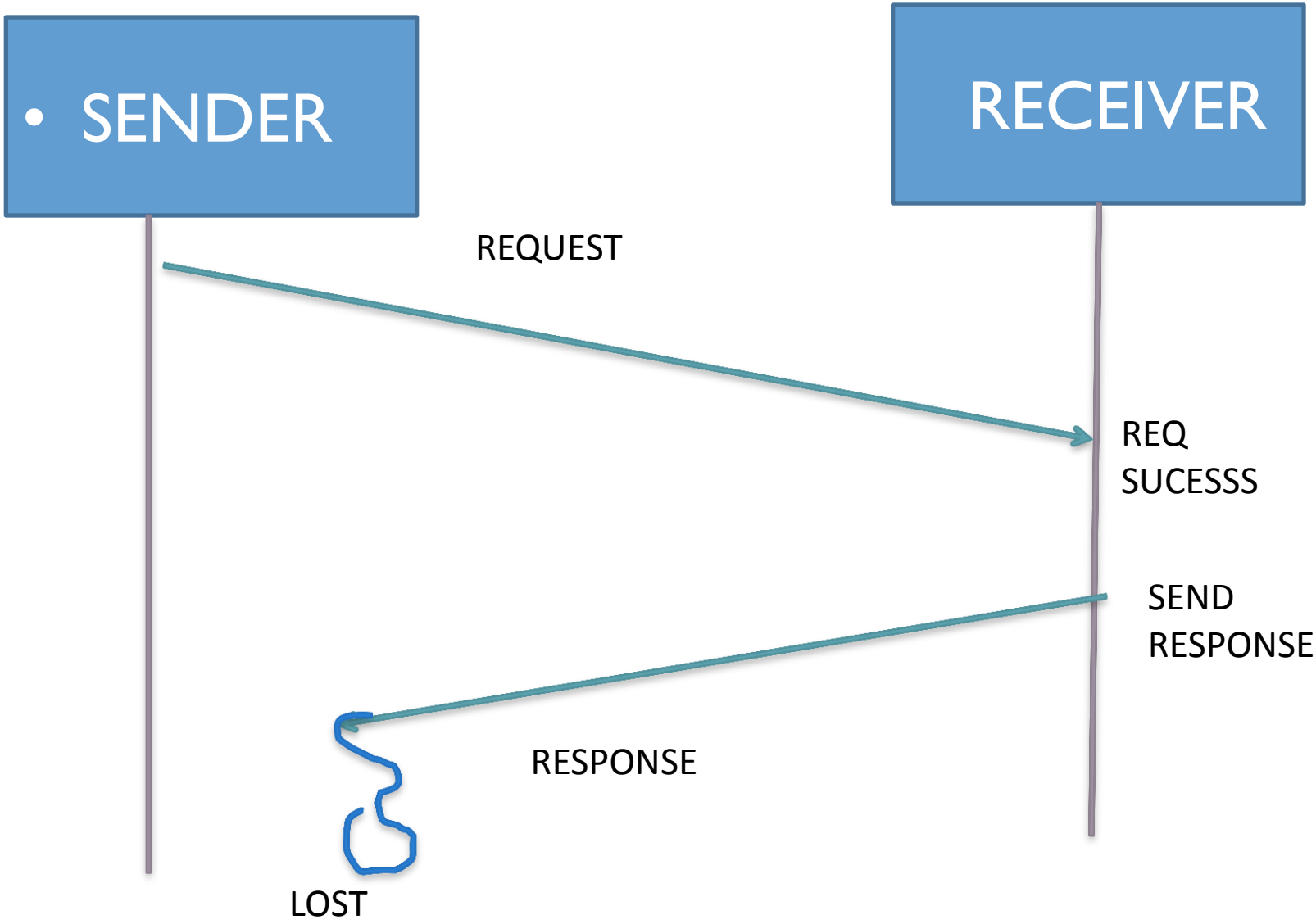
- The first field identifies the node on which the process is created
- The second field is a local indentifier generated by the node on which the process is created
- The third field identifies the last known location (node) of the process

FAILURE HANDLING

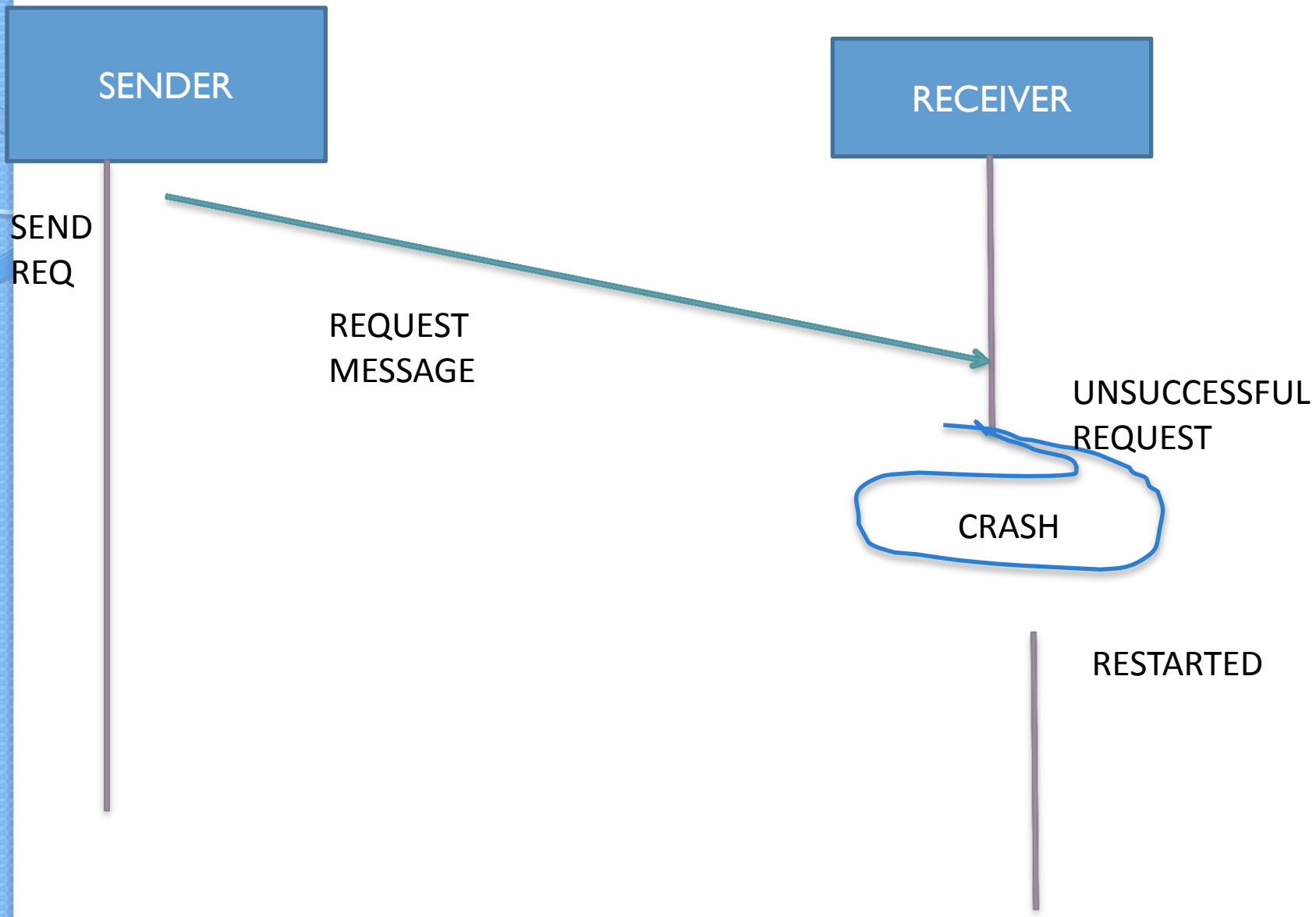
- Partial failures such as a node crash
- *Loss of request message.*
  - due to the failure of communication link between the sender and receiver
  - receiver's node is down at the time the request message reaches
- *Loss of response message.*
  - due to the failure of communication link
  - sender's node is down at the time the response message reaches there.
- *Unsuccessful execution of the request.*
  - crashing while the request is being processed.



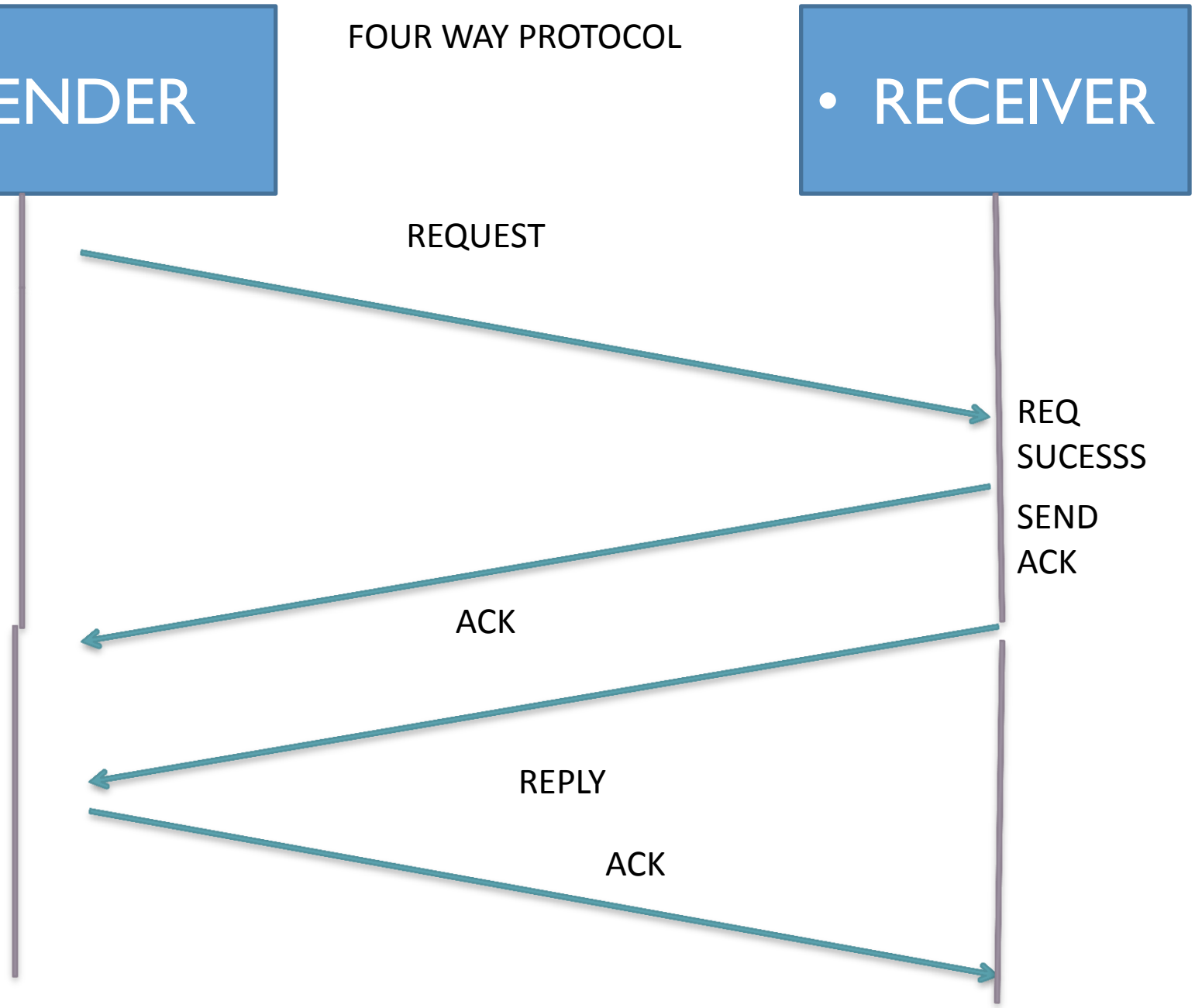
LOSS OF REQUEST MESSAGE



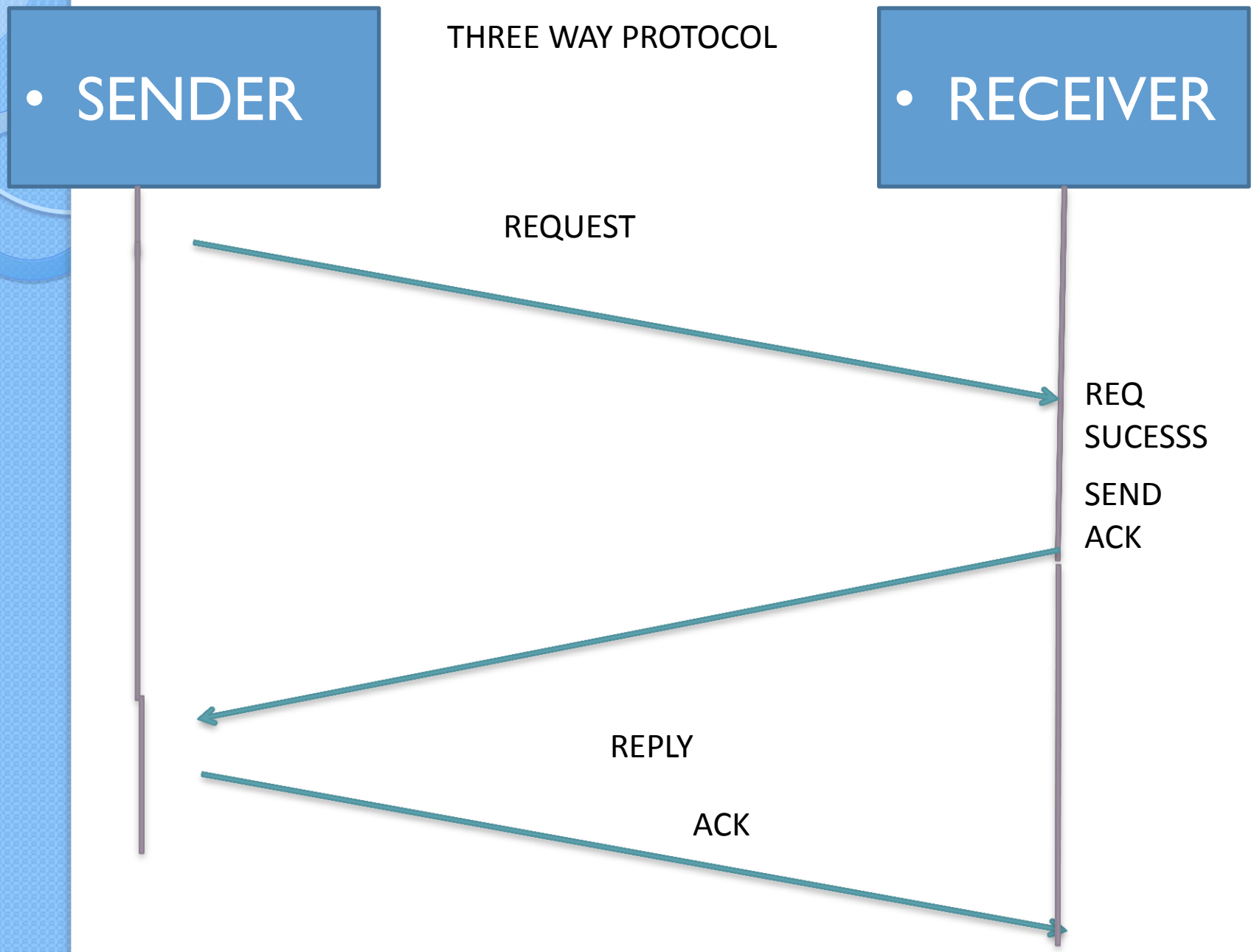
LOSS OF RESPONSE


SENDER
RECEIVER
SEND
REQ
REQUEST
MESSAGE
UNSUCCESSFUL
REQUEST
CRASH
RESTARTED

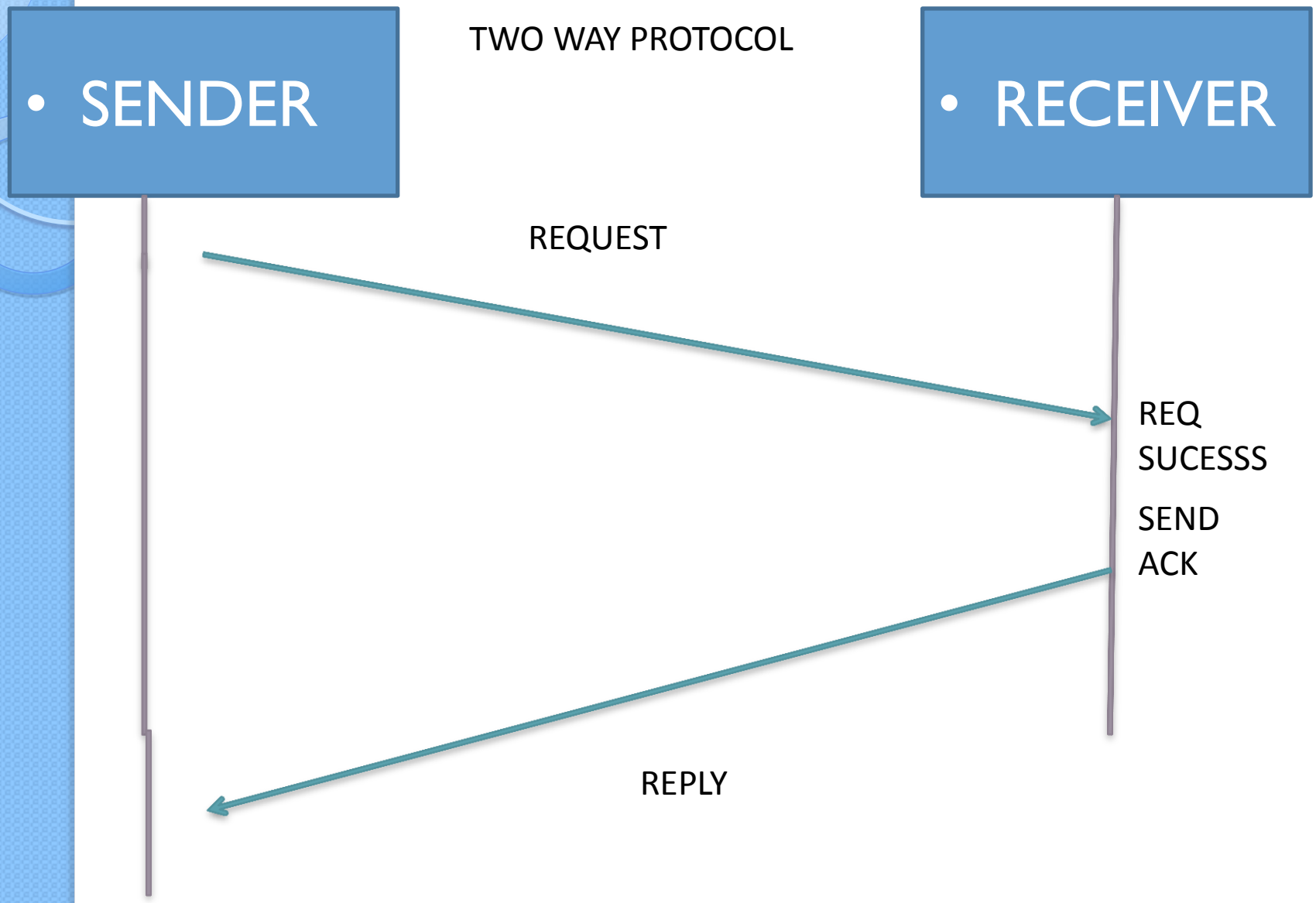
FOUR WAY PROTOCOL

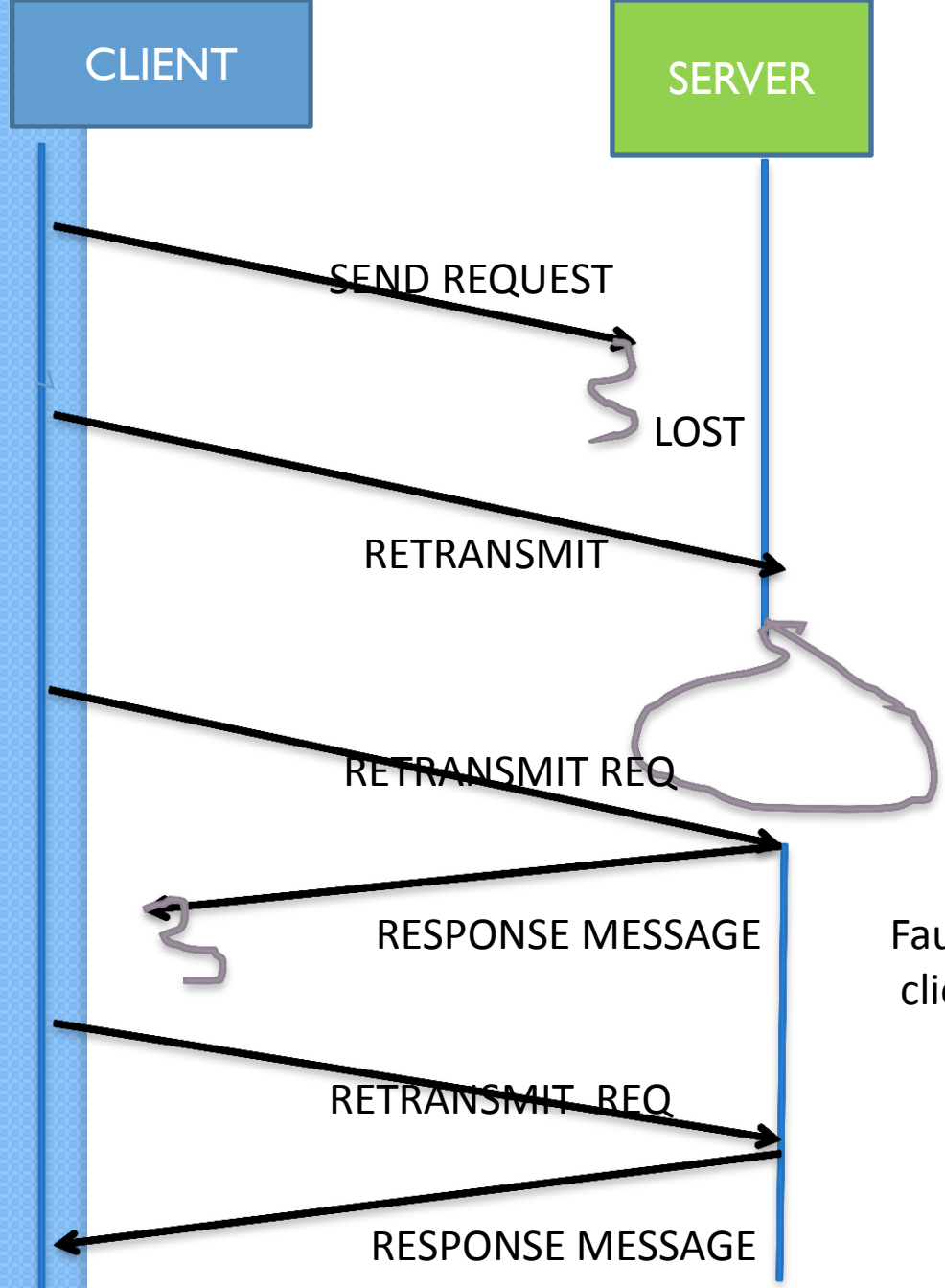


THREE WAY PROTOCOL



TWO WAY PROTOCOL





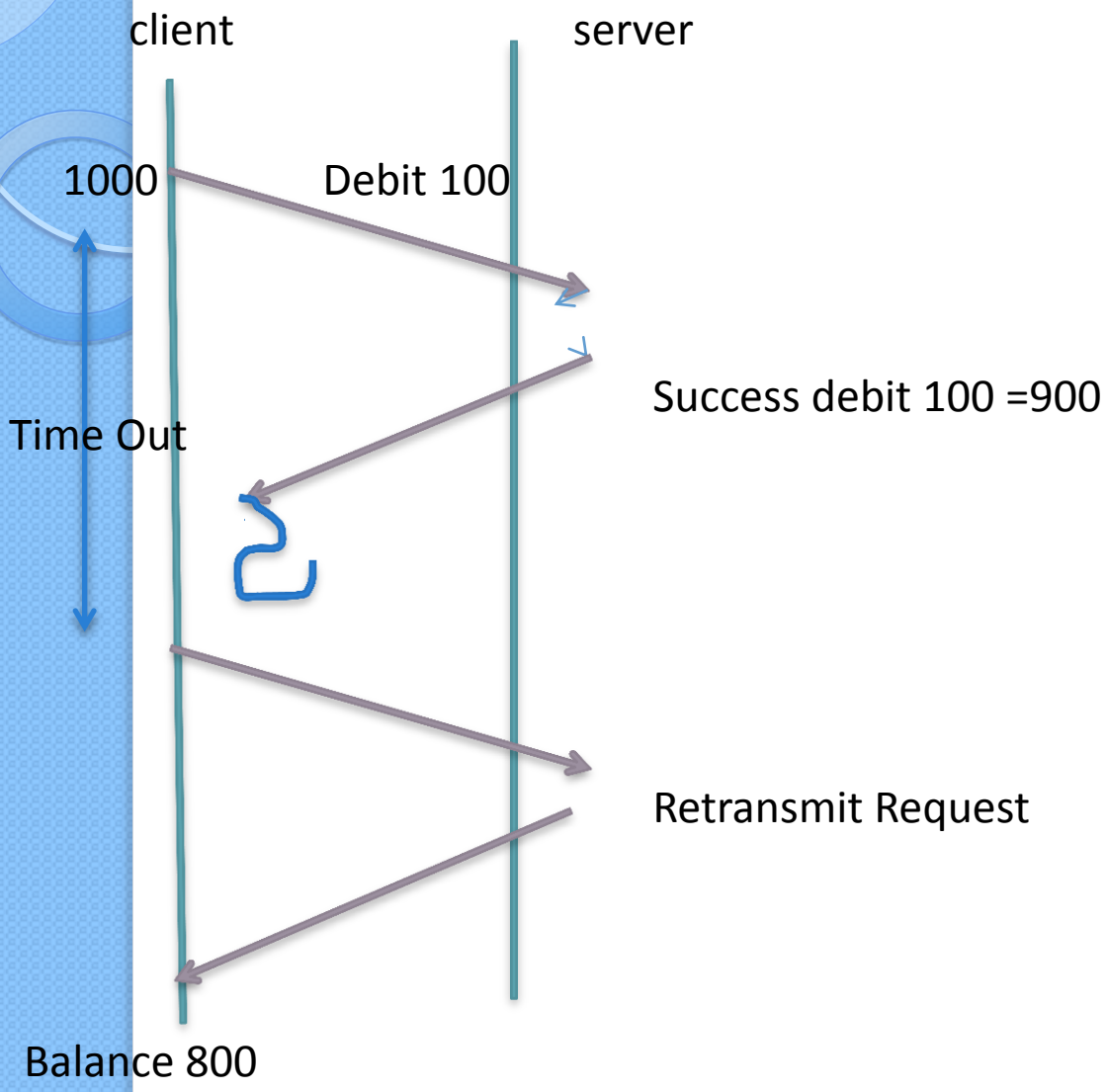
Fault tolerant Communication Between client- Server

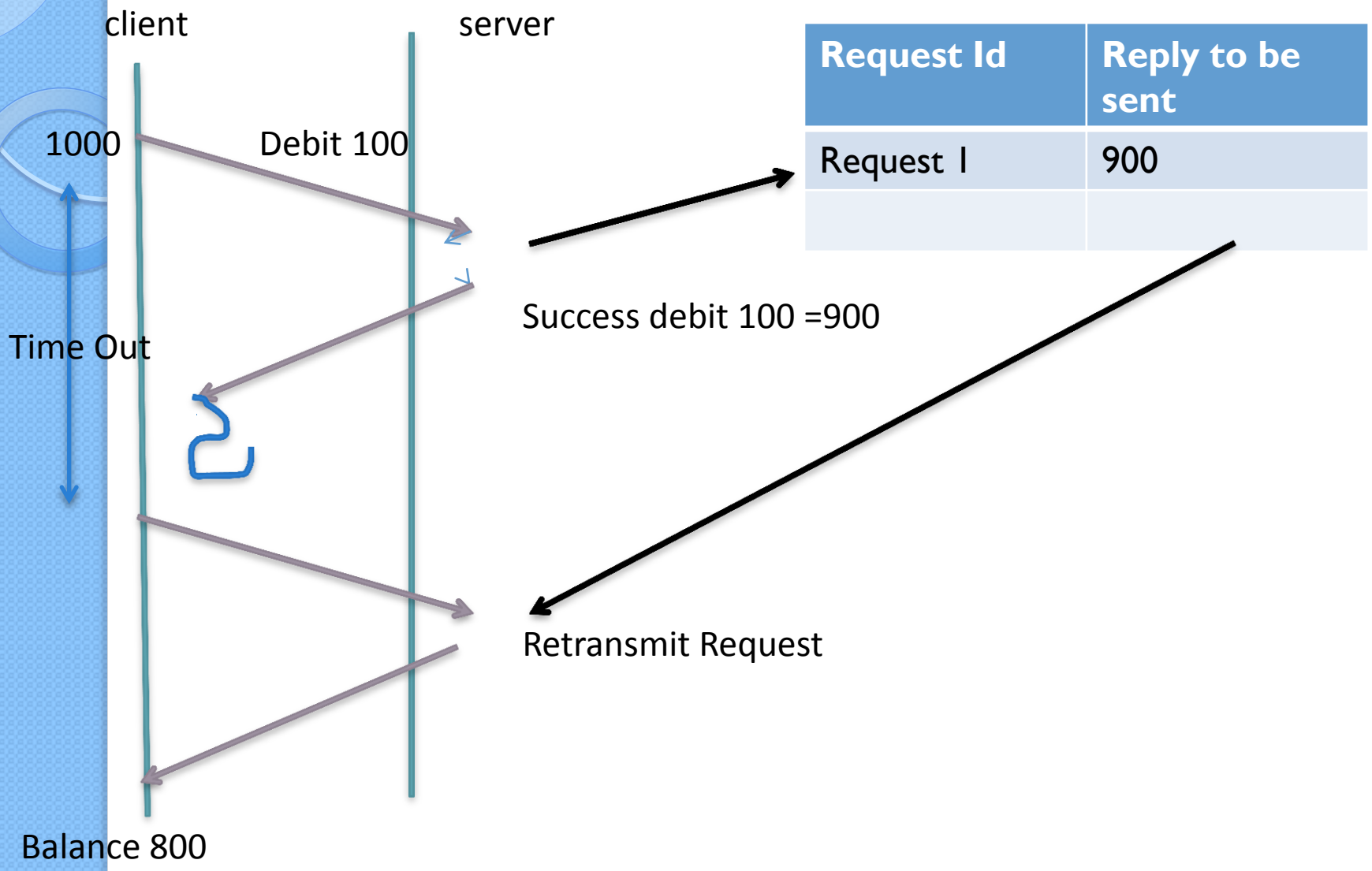
Idempotency and Handling or Duplicate Request Messages

- Repeatability
- produces the same results with same arguments.
- Eg. Sqrt finding

Not the same results

Eg. Debit(amount)


client
server
1000
Debit 100
Success debit 100 =900
Time Out
Retransmit Request
Balance 800



Request Id	Reply to be sent
Request I	900

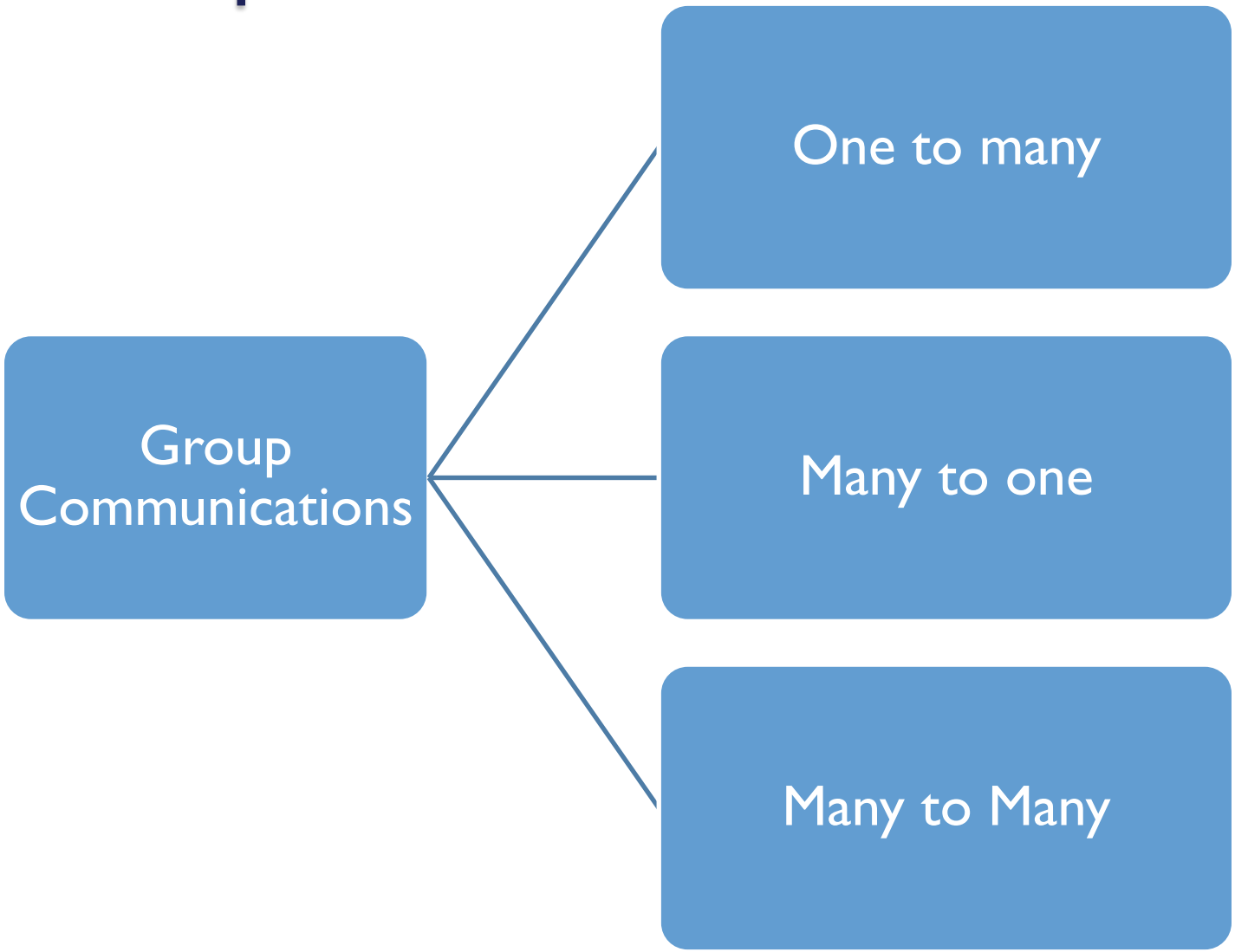
lost and Out-of-Sequence Packets

- For successful completion of a multidatagram message transfer, reliable delivery of every packet is important.
- Reliability — Ack for each pkt
  - Stop and Wait Protocol
  - Called Blast Protocol
- link failure leads to — Ack for all
  - Message Loss
  - Out of sequence

lost and Out-of-Sequence Packets

- Selective Repeat
- Header Part two extra fields
  - First field buffer size
  - Second Position of packet
  - After timeout not received pakets will be sent using bitmap

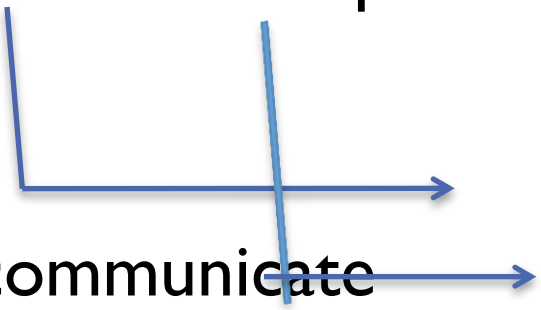
Group Communications



Group Communications

- **One-to-Many Communication**
  - **Also known as multicasting**
  - **Broadcasting is a special case**
- **Group Management**

closed and open.



only members can communicate

Any member can communicate

Group Addressing

- A special network address to which multiple machines can listen. Such a network address is called a *Multicast Address.*
- A packet sent to a broadcast address is automatically delivered to all machines on the network.
- Otherwise one-one communication.

Group Addressing

- **Buffered and Un-Buffered Multicast**

1. A sending process cannot wait until all the receiving processes that belong to the multicast group are ready to receive the multicast message.

2. The sending process may not be aware of all the receiving processes that belong to the multicast group.

Group Addressing

Semantics for one-to many communications:

1. *Send-to-all semantics*. A copy of the message is sent to each process of the multicast group and the message is buffered until it is accepted by the process.

2. *Bulletin-board semantics*. A message to be multicast is addressed to a channel instead of being sent to every individual process of the multicast group. From a logical

Flexible Reliability in Multicast Communication

- Degree of Reliability depends on
- The *O-Reliable*. asynchronous multicast in which the sender does not wait for any response after multicasting the message.
- The *1-Reliable*. The sender expects a response from any of the receivers. The first server that responds is an example of l-reliable multicast communication.

Flexible Reliability in Multicast Communication

3. *M-out-of-N-Reliable*. The multicast group consists of *n* receivers and the sender expects a response from *m* (1<*m*<*n)* of the *n* receivers.

4. All-reliable. The sender expects a response message from all the receivers of the multicast group

GROUP COMMUNICATION PRIMITIVES

- Send
- Send-group

Many-One Communication

- Non-Determinism. The receiver may want to wait for information from any of a group of senders

Many-Many Communication

Issues

Ordered Delivery

Message sequencing

GROUP COMMUNICATION PRIMITIVES

- Absolute Ordering
- Consistent Ordering
- Casual Ordering

Absolute ordering

Messages are delivered in exact order

Global time stamps are used

Consistent order

- This order may be different from the order in which messages were sent

GROUP COMMUNICATION PRIMITIVES

- Casual Order
- If two message sending events are not causally related, the two messages may be delivered to the receivers in any order.
- Happened before relation-One event should happen before the another in the time domain.

REFERENCE

Pradeep K. Sinha, "Distributed Operating System Concepts and Design", PHI, New Delhi,2007.