


# *Adama Science and Technology University*

*"We are dedicated to innovation knowledge"*

## Department of Computer Science and Engineering

## Program : Software Engineering

## Distributed Software System (SEng4305)

**Compiled By: Sufian K (MSc)**

# Chapter 2

# Communication in DS

# Content

- Layered Protocols
- Types of Communication
- Middleware
  - ✓Remote Procedure Call(RPC)
  - ✓Object Oriented Middleware (RIM, Socket)
  - ✓Message Oriented Middleware (MOM)

# Introduction

- The single most important difference between as distributed and uniprocessor system is the Inter process communication(IPC).
- There are two methods of IPC:
  - Shared memory and Message passing.
- In the Uniprocessor system the IPC done via shared memory while in Distributed system IPC done
  - through message passing there is no shared memory.
- This chapter conclude by how a groups of processes can communicate, instead of just two processes.

# Introduction….

- Inter process communication is at the heart of all distributed systems
- Communication in distributed systems is based on
  - message passing as offered by the underlying network
- Modern distributed systems consist of thousands of processes scattered across an unreliable network such as the Internet.
- Unless the primitive communication facilities of the network are replaced by more advanced ones,
  - development of large scale Distributed Systems becomes extremely difficult.

# Layered Protocols

- Why communication in distributed systems?
  - because there is no shared memory.
- Two communicating processes must agree on the syntax and semantics of messages.
- A protocol is a set of rules that governs data communications.
- A protocol defines what is communicated, how it is communicated, and when it is communicated
- For instance, for one computer to send a message to another computer, the first computer must perform the following general steps
  - break the data into small sections called packets
  - add addressing information to the packets identifying the source and destination computers
  - deliver the data to the network interface card for transmission over the network

# Layered Protocols

- The receiving computer must perform the same steps, but in reverse order
  - accept the data from the NIC
  - remove transmitting information that was added by the transmitting computer
  - reassemble the packets of data into the original message
- The key elements of a protocol are syntax, semantics, and timing
  - Syntax: refers to the structure or format of the data
  - Semantics: refers to the meaning of each section of bits
  - Timing: refers to when data should be sent and how fast they can be sent

# Cont….

- **Layers and Services**
  - Within a single machine, each layer uses the services immediately below it and
  - Provides services for the layer immediately above it.
- Two important network models or architectures
  - The OSI (Open Systems Interconnection) Reference Model
  - The TCP/IP Protocol Suite
  - TCP/IP protocol suite is the other; has 4 or 5 layers

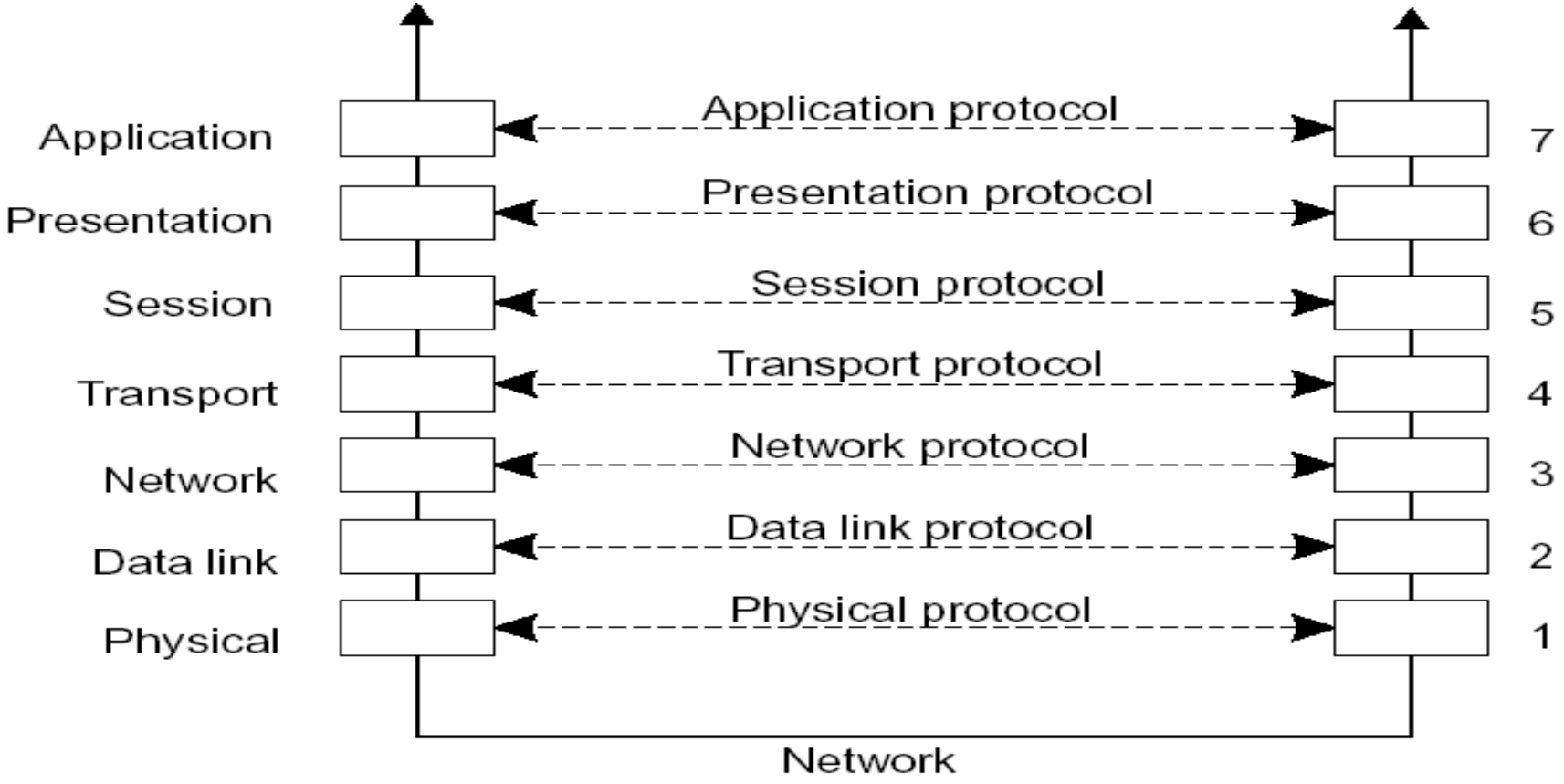
# OSI Layers

- It consists of 7 layers
- Its designed to allow open system to communicate
- **Open system** is one that prepared to communicate with other open systems by
  - using standard rules that govern the format, contents, and meaning of the messages sent and received.
  - These rules are called protocols.
- Protocol is an agreement between the communicating parties on how communication to be processed.

# OSI Layers

- Each layer provides an interface to the one above it.
- The interface consists of a set of operations that together
  - define the service the layer is prepared to offer its users.

# Cont.....




Layers, interfaces, and protocols in the OSI model

# Cont….

- Physical layer :– handles the mechanical/electrical details of the physical transmission of a bit stream **(0's and 1's).**
- Data-link layer :– handles the frames, or fixed-length parts of packets, including any error detection and recovery that occurred in the physical layer.
- Network layer :– provides connections and routes packets in the communication network, including handling the address of outgoing packets, decoding the address of incoming packets, and maintaining routing information for proper response to changing load levels.

# Cont....

- Transport layer :– responsible for low-level network access and for message transfer between clients, including partitioning messages into packets, maintaining packet order, controlling flow, and generating physical addresses.
- Session layer :– implements sessions, or process-to-process communications protocols.
- Presentation layer :– resolves the differences in formats among the various sites in the network, including character conversions, compression, encryption and half duplex/full duplex (echoing).
- Application layer :– interacts directly with the users' deals with file transfer, remote-login protocols and electronic mail, as well as schemas for distributed databases.

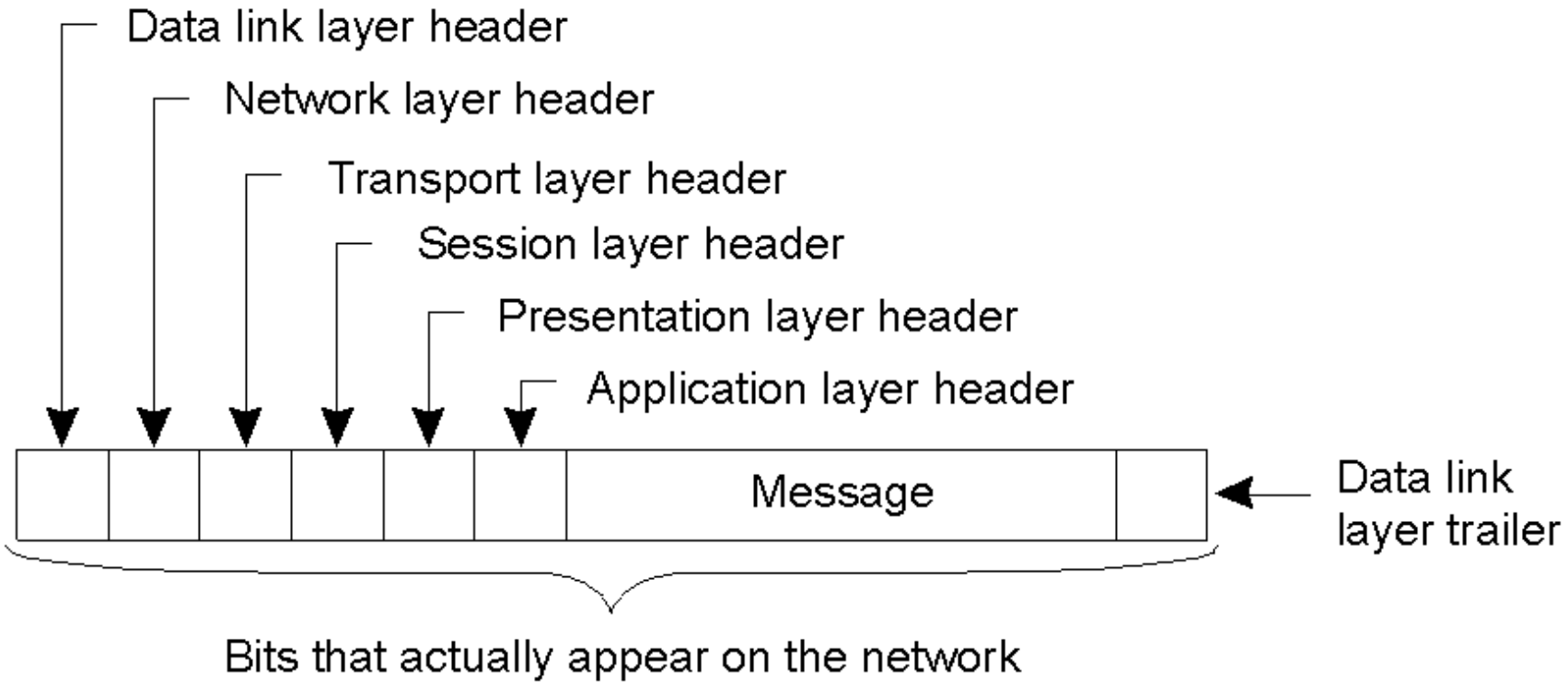
# Cont.....

- Physical: Physical characteristics of the media
- Data Link: Reliable data delivery across the link
- Network: Managing connections across the network or routing
- Transport: End-to-end connection and reliability (handles lost packets); TCP (connection-oriented), UDP (connectionless), etc.
- Session: Managing sessions between applications (dialog control and synchronization); rarely supported
- Presentation: Data presentation to applications; concerned with the syntax and semantics of the information transmitted
- Application: Network services to applications; contains protocols that are commonly needed by users; FTP, HTTP, SMTP, Telnet etc.

*Media (lower) Layers*

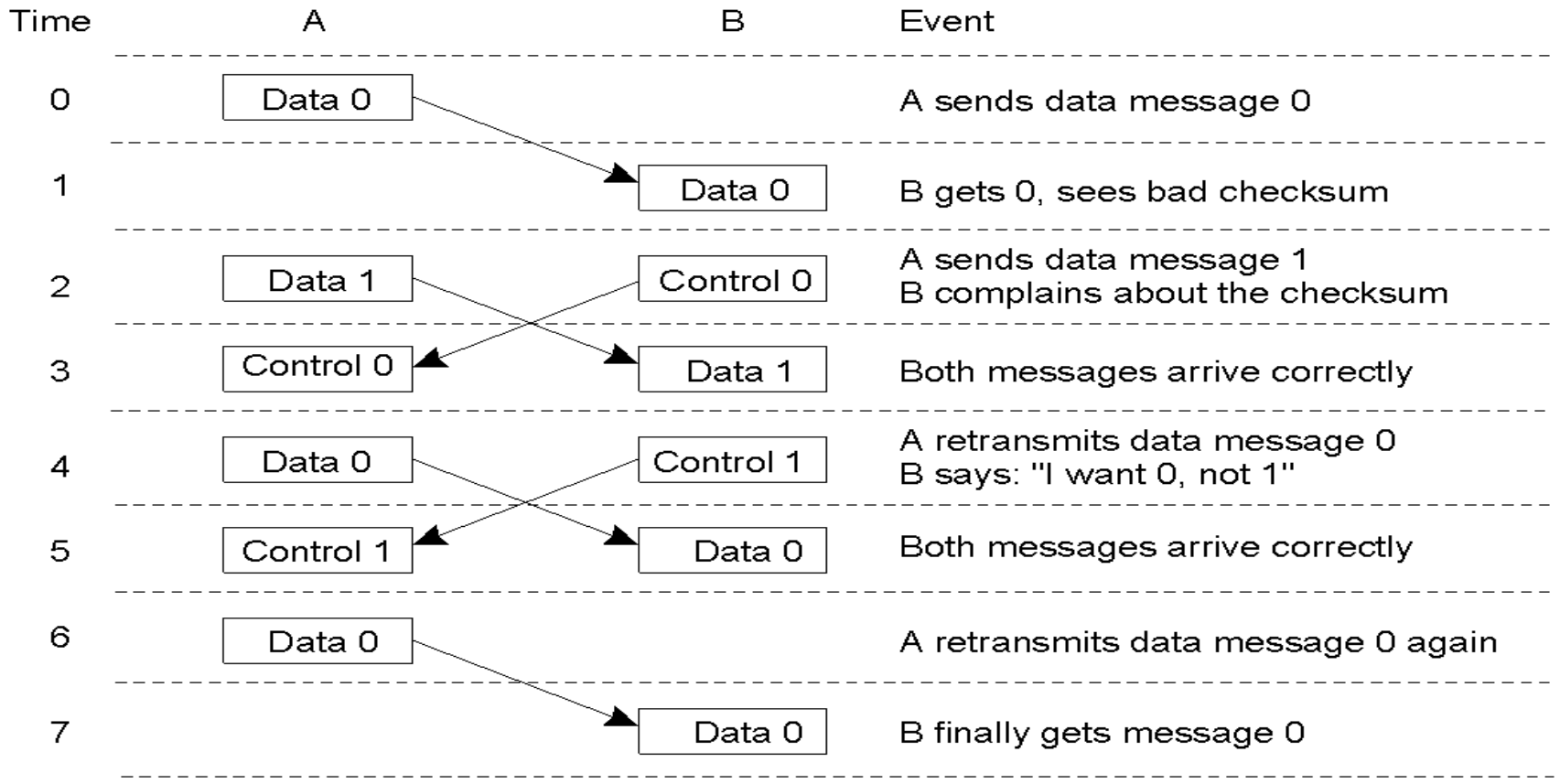
*Host (upper) Layers*

# Cont.....




Typical message as it appears on the network.

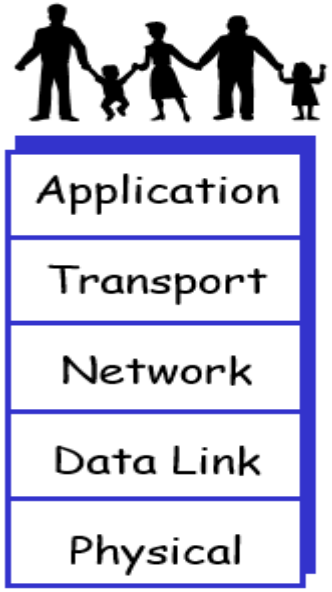
# Cont....



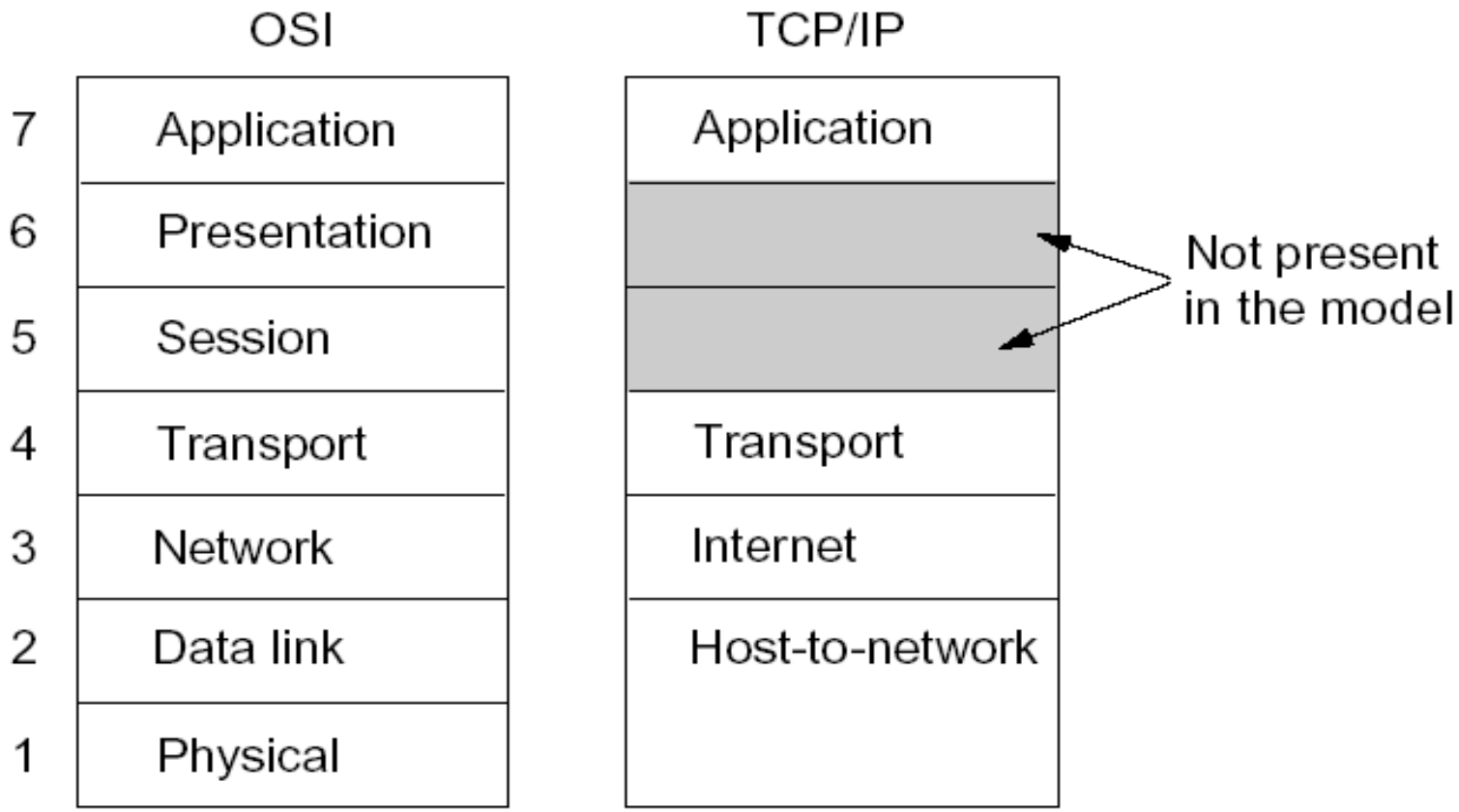

- Conversation occurs between a receiver and a sender at each layer (Discussion in the data link layer)

# TCP/IP Reference Model

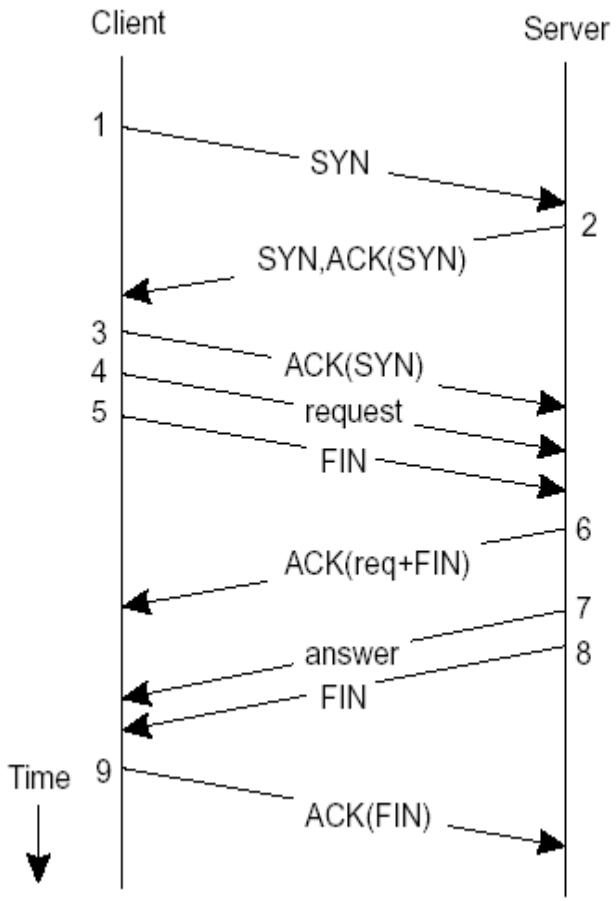
- TCP/IP (Transmission Control Protocol/Internet Protocol)
- It has 4 (or 5 depending on how you see it) layers:
  - Application, Transport, Internet (Internetwork), and Host-to-network (some split it into Physical and Data Link)



# OSI and TCP/IP Layers Correspondence



# Transport Protocols: Client-Server TCP



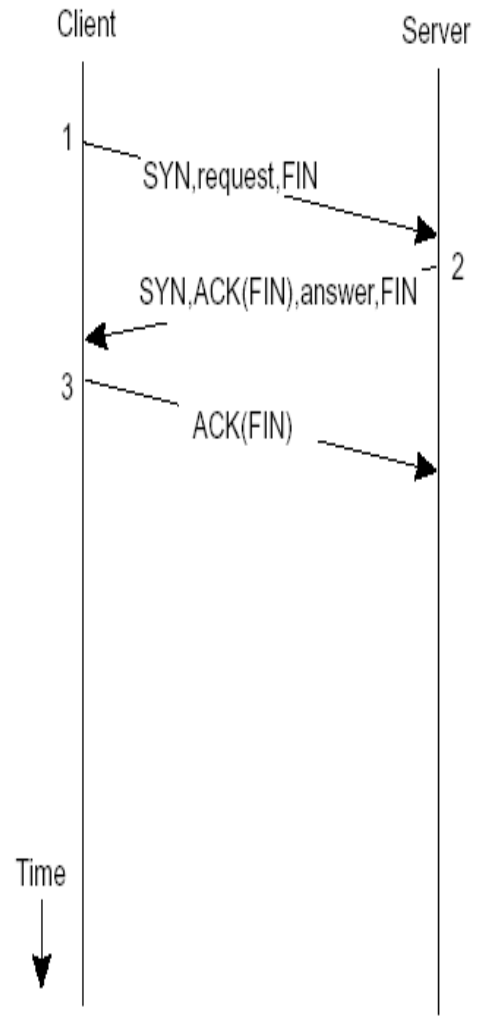

Assuming no messages are lost,

- Client initiates a setup connection using a **3-way handshake** (1-3)
- Client sends its request (4)
- It then sends a message to close the connection (5)
- Server acknowledges receipt and informs the client that the connection will be closed down (6)
- Then sends the answer (7) followed by a request to close the connection (8)
- Client responds with an Ack to finish conversation (9)

Normal Operation of TCP

# Cont.....

- Combine connection setup with request and closing connection with answer. Such protocol is called TCP for Transactions (T/TCP)
  - ☞ Client sends a single message consisting of a setup request, service request, and information to the server that the connection will be closed down immediately after receiving the answer (1)
  - ☞ Server sends acceptance of connection request, the answer, and a connection release (2)
  - ☞ Client acknowledges tear down of the connection (3)




**Transactional TCP**

# Types of Communication

- Communication can be
  - persistent or transient
  - asynchronous or synchronous
- **Persistent**: a message that has been submitted for transmission is stored by the communication system as long as it takes to deliver it to the receiver e.g., email delivery.
- **Transient**: a message that has been submitted for transmission is stored by the communication system only as long as the sending and receiving applications are executing.
- **Asynchronous**: a sender continues immediately after it has submitted its message for transmission.
  - the message is (temporarily) stored immediately by the middleware upon submission.

# Types of Communication

- **Synchronous:** the sender is blocked until its message is stored in a local buffer at the receiving host or delivered to the receiver.
- There are essentially three points where synchronization can take place.
  - First, the sender may be blocked until the middleware notifies that it will take over transmission of the request.
  - Second, the sender may synchronize until its request has been delivered to the intended recipient.
  - Third, synchronization may take place by letting the sender wait until its request has been fully processed, that is, up the time that the recipient returns a response.

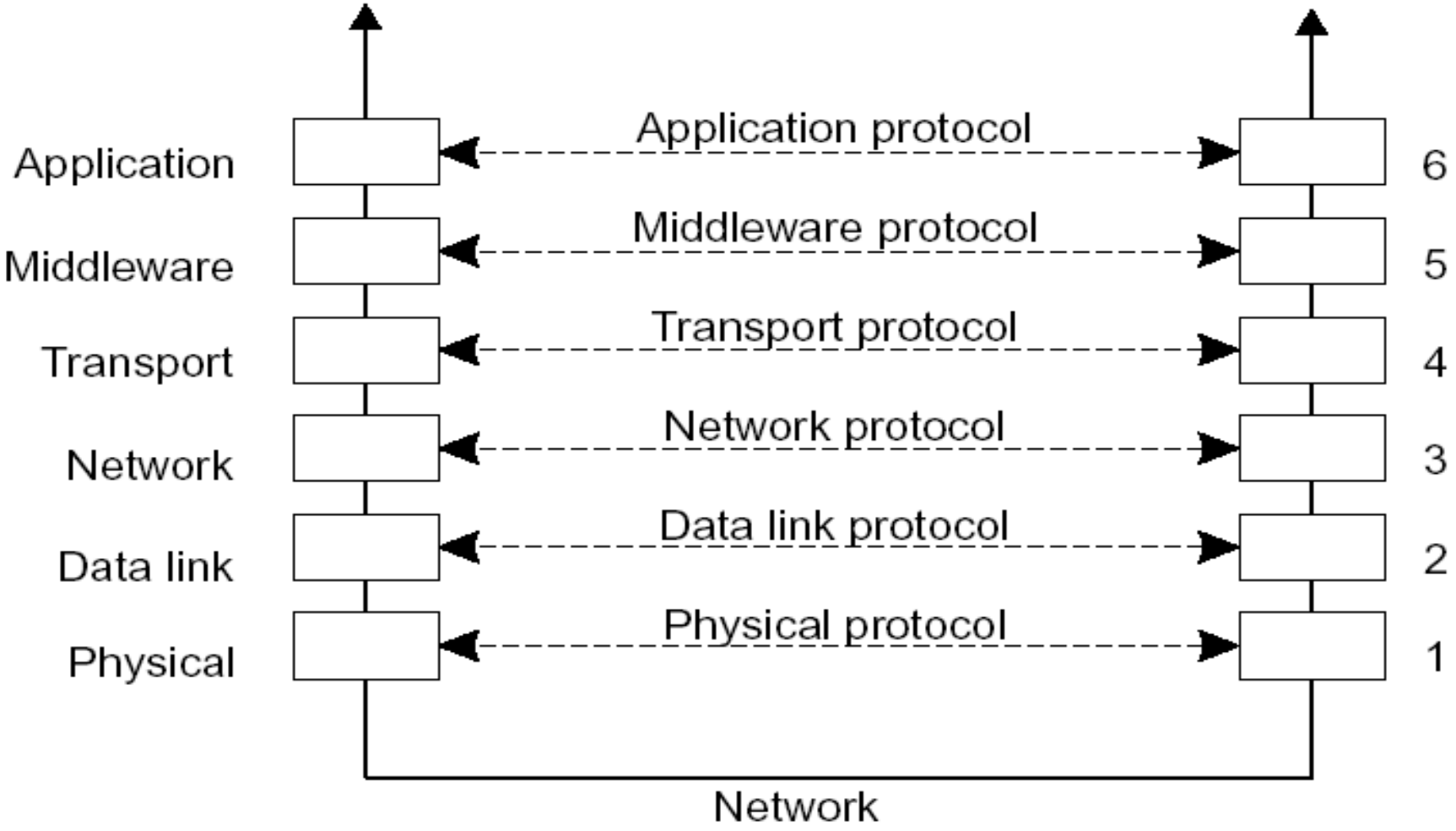
# Middleware protocols

- A middleware is an application that
  - contains general-purpose protocols to provide services.
  - Maintains information integrity across multiple systems
- It is an intermediate layer of software that sits between the application and the network.
- In DS middleware used to enhance the distribution transparency that is missing in network OS.
  - Distribution transparency is the phenomena by w/c distribution aspects in the system are hidden from user and application.

# Cont.....

- Middleware is used in distributed systems to provide common services, such as
  - authentication, authorization, compilation for best performance on particular architectures, input/output translation, and error handling.
  - distributed transactions (commit protocols; locking mechanisms)
  - middleware communication protocols (calling a procedure or invoking an object remotely, synchronizing streams for real-time data, multicast services)

# Cont.....




*An adapted reference model for networked communication*

# Cont.....

- It's simple and sophisticated, but there are implementation problems
  - The calling and called procedures run in different address spaces.
  - Parameters and results have to be exchanged; what if the machines are not identical?
  - What happens if both machines crash?

# Remote Procedure Call (RPC)

- The first distributed systems were based on explicit message exchange between processes through the use of
  - explicit **send** and **receive** procedures;
  - but do not allow access transparency (do not conceal communication at all).
- In 1984, Birrel and Nelson introduced a different way of handling communication: RPC
  - RPC allows a program to call a procedure located on another machine.

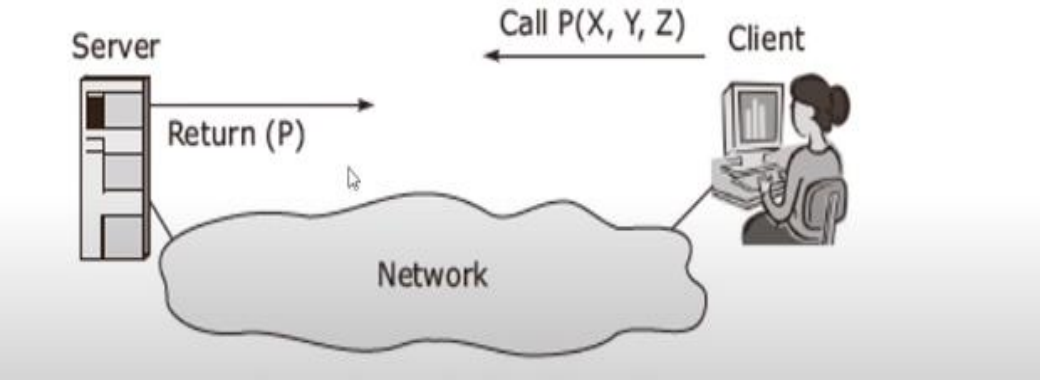
# Remote Procedure Call (RPC)

- RPC Concept: Allowing programs to call procedures located on other machines.
- When a process on machine **A** calls a procedure on machine **B**, the calling process on A is suspended, and execution of the called procedure takes place on **B**.
- Information can be transported from the **caller** to the **callee** in the parameters and can come back in the procedure result.
  - No message passing at all is visible to the programmer.
- This method is known as Remote Procedure Call(RPC).

# Cont….

- **RPC** is a software communication protocol that one program can use, to request a service from a program located in another computer on a network without having to understand the network's details.
- It's programs to call procedures located on other machines.
  - It is also known as a function call or a subroutine call.
  - It Use the client server model

- Basic RPC operation



# Cont….

- It's a
  - powerful technique for constructing distributed, client-server-based applications.
- RPC is used to call other processes (program) on the remote systems like a local system.
  - For example Email.

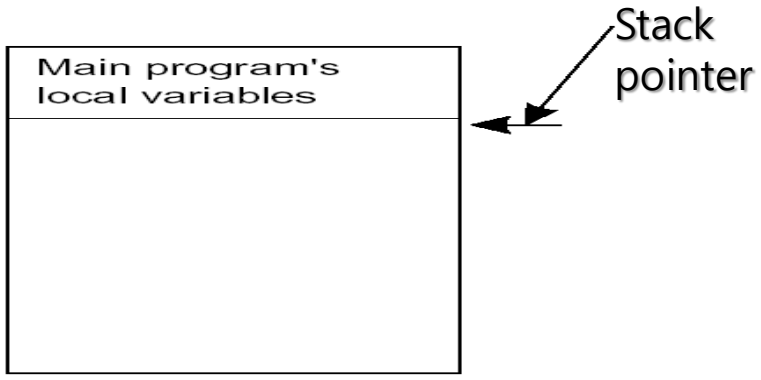
# Cont.....

- Conventional Procedure Call,( i.e., on a single machine)
  - e.g. count = read (fd, buf, bytes); where

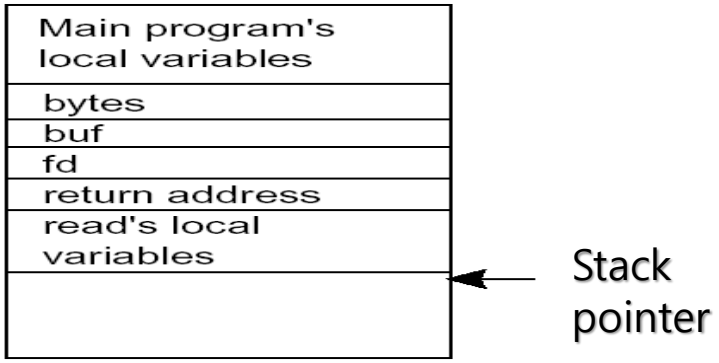
    **fd** is an integer indicating a file

    **buf** is an array of characters into which data are read

    **bytes** is the number of bytes to be read




Parameter passing in a local procedure call: the stack before the call to read

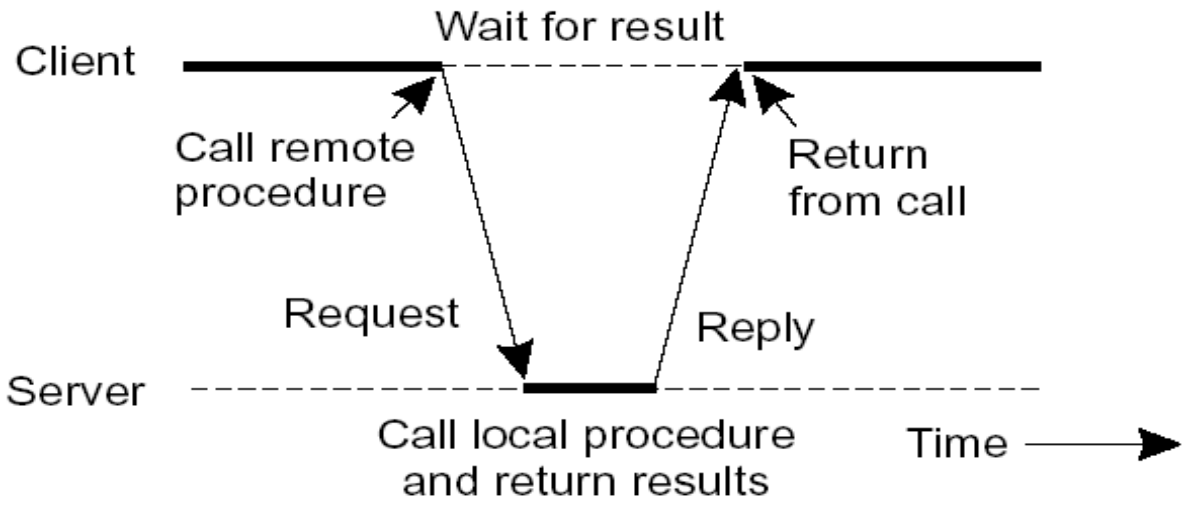



Stack while the called procedure is active

- Parameters can be call-by-value (fd and bytes) or call-by reference (buf) or in some languages call-by-copy/restore.

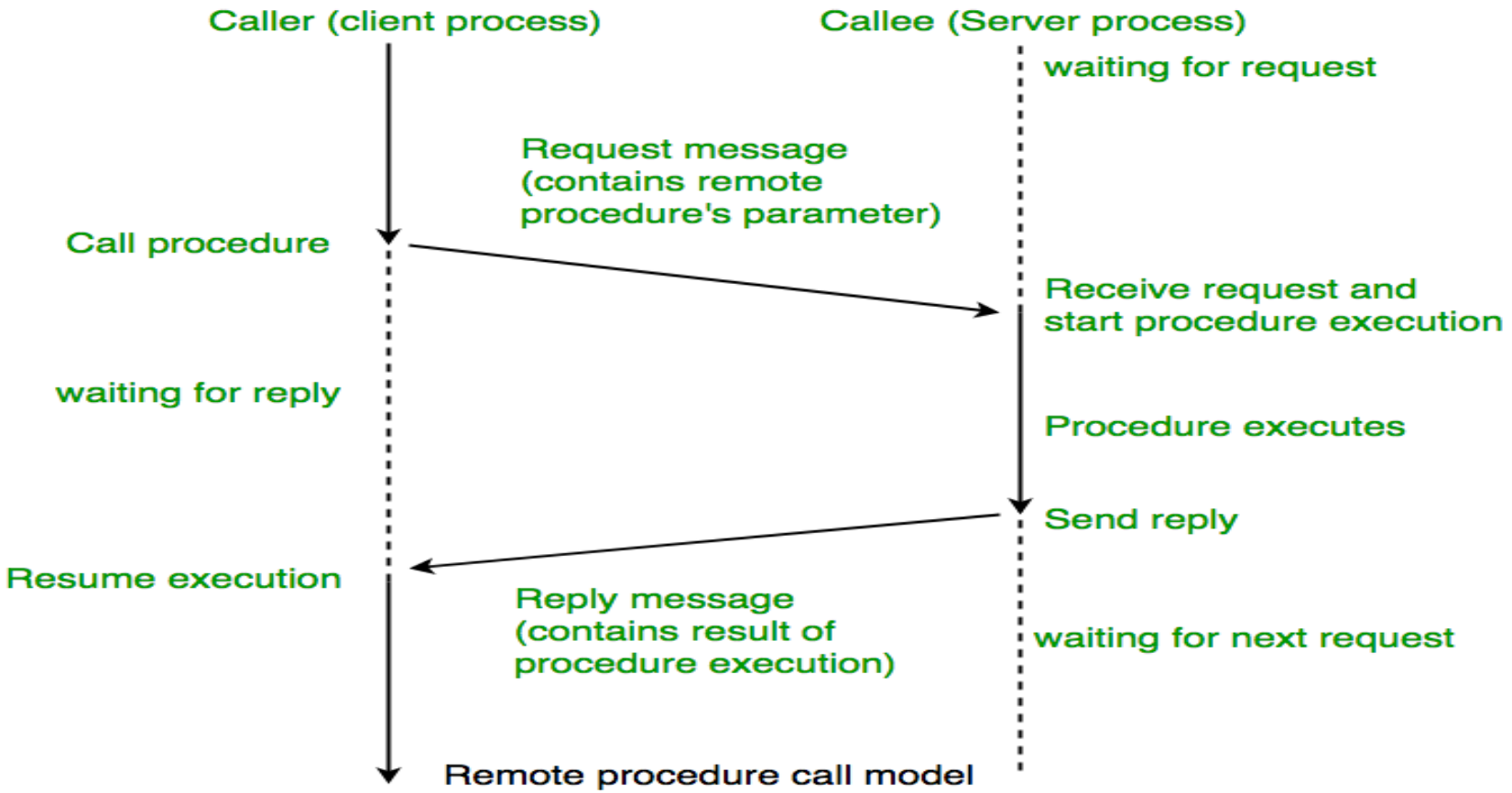
# Cont....

- RPC would like to make a remote procedure call look the same as a local one; it **should be transparent**,
  - i.e., the calling procedure should not know that the called procedure is executing on a different machine or vice versa.




Principle of RPC between a client and server program

# Cont.....


Caller (client process)
Callee (Server process)
waiting for request
Request message
(contains remote
procedure's parameter)
Call procedure
Receive request and
start procedure execution
waiting for reply
Procedure executes
Send reply
Resume execution
Reply message
(contains result of
procedure execution)
waiting for next request
Remote procedure call model

# Cont..

- RPC includes mainly five elements:
  - The Client
  - The client stub (stub:- is a piece code used for converting parameters(for pack/unpack)).
  - RPC runtime (RPC communication package )
  - The server Stub
  - The server

# Cont..

- The Client:-
  - ✓ it user process w/c initiate a RPC.
  - ✓ The client makes a perfectly normal call that invokes a corresponding procedure in the client stub.
- Client Stub:-
  - ✓ on receipt of a request it packs requirements into the message and asks to RPC runtime to send.
  - ✓ On receipt of the results it unpacks the result and passed it into a client.
- RPC runtime:- it handles the transmission message between the client and server.

# Cont..

- Server Stub :-
  - ✓ it unpacks a call request and make a perfectly normal call to invoke the appropriate procedure in the server.
  - ✓ On receipt of a result a procedure execution it packs the result and asks on RPC runtime to send.
- Sever :- it execute an appropriate procedure and return the result from a server stub.

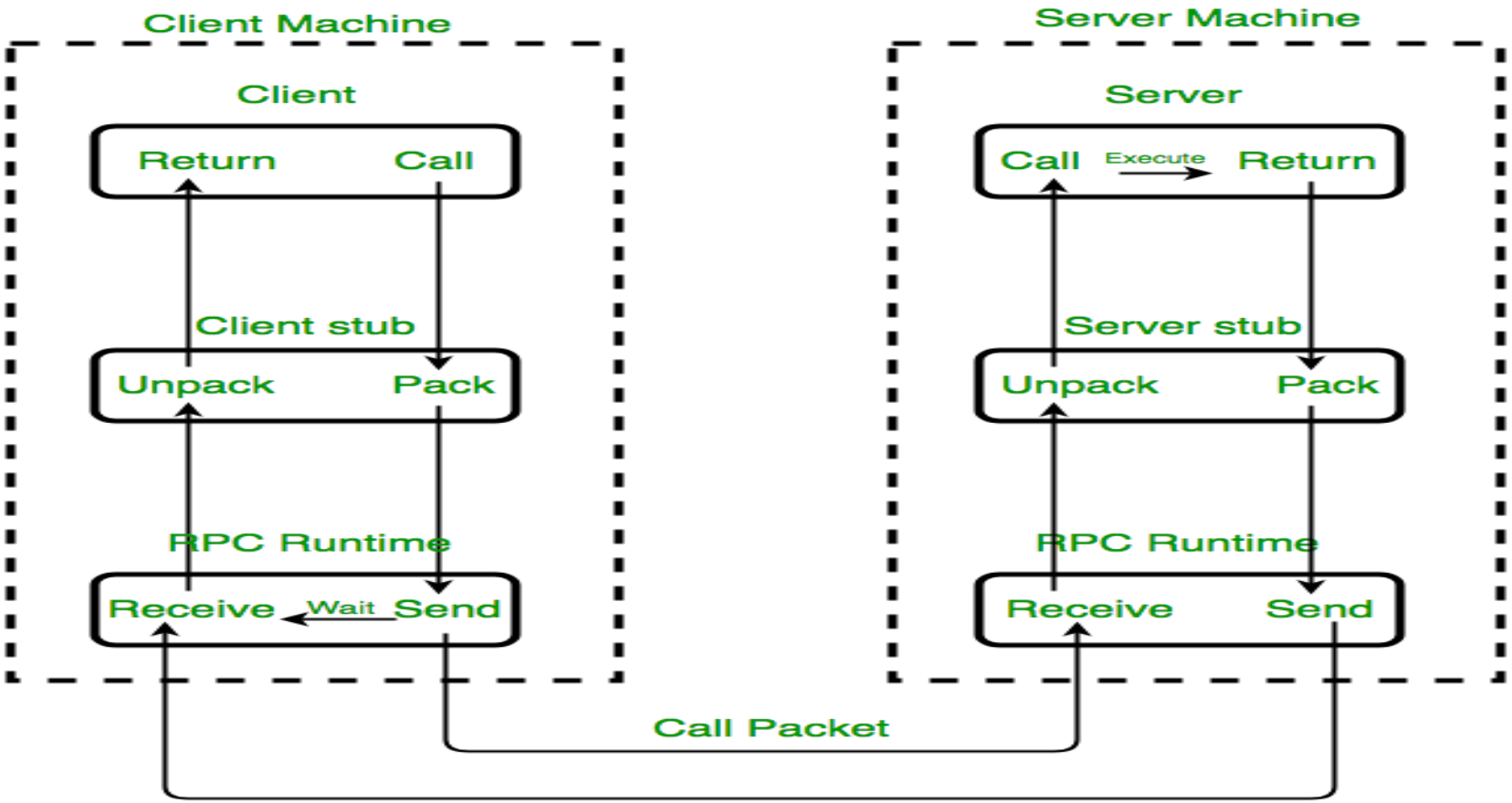
# Cont....

## Steps of a Remote Procedure Call

1. Client procedure calls client stub in the normal way
2. Client stub builds a message and calls the local OS (packing parameters into a message is called parameter marshaling)
3. Client's OS sends the message to the remote OS
4. Remote OS gives the message to the server stub
5. Server stub unpacks the parameters and calls the server
6. Server does the work and returns the result to the server stub
7. Server stub packs it in a message and calls the local OS
8. Server's OS sends the message to the client's OS
9. Client's OS gives the message to the client stub
10. Stub unpacks the result and returns to **client**

- Hence, for the client remote services are accessed by making ordinary (local) procedure calls; not by calling send and receive.

# Cont..



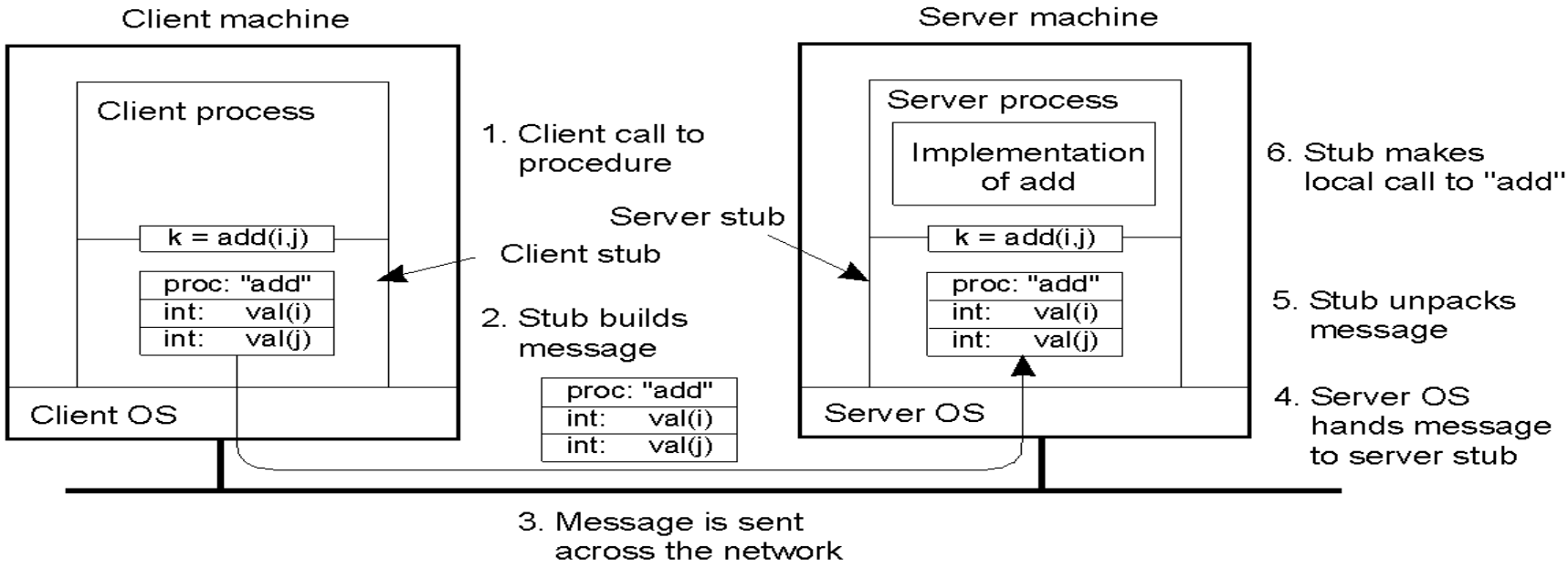

Implementation of RPC mechanism

# Parameter Passing

- **Passing Value (Call-by value copy)** is copy all parameters into a message before transmission.
  - when we pass the value of the parameter during the calling of the function, it copies them to the function's actual local argument
- **Passing Reference(call by reference)** is passes pointers to the parameters that are passed from the client to the server.
  - In this case we pass the parameter's location reference/address, it copies and assigns them to the function's local argument.

# Cont....

- **Example Passing Value Parameters**
  - e.g., consider a remote procedure add(i, j), where i and j are integer parameters




Steps involved in doing remote computation through RPC

# Cont....

## 2. Passing Reference Parameters

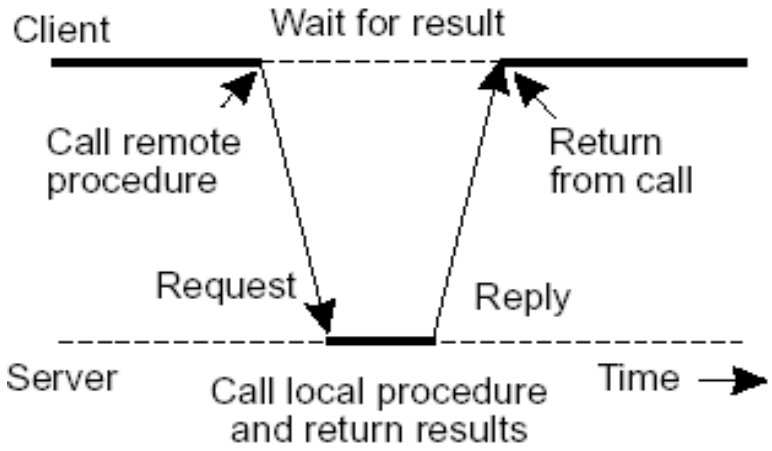
- Assume the parameter is a pointer to an array
- Copy the array into the message and send it to the server
- The server stub can then call the server with a pointer to this array
- The server then makes any changes to the array and sends it back to the client stub which copies it to the client
- This is in effect call-by-copy/restore
- Optimization of the method
  - ✓ one of the copy operations can be eliminated if the stub knows whether the parameter is input or output to the server
  - ✓ if it is an input to the server (e.g., in a call to write), it need not be copied back
  - ✓ if it is an output, it need not be sent over in the first place; only send the size
- The above procedure can handle pointers to simple arrays and structures, but difficult to generalize it to an arbitrary data structure

## Asynchronous RPC

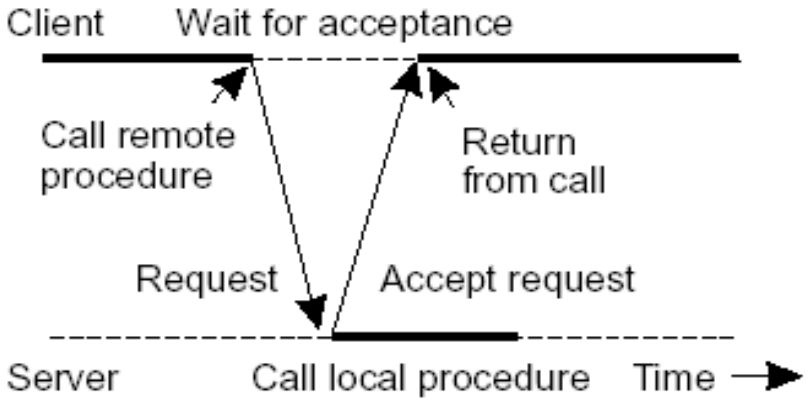
- If there is no need to block the client until it gets a reply.
- Two cases

**1. if there is no result to be returned**

  - e.g., adding entries in a database, ...
  - the server immediately sends an Ack promising that it will carryout the request.
  - the client can now proceed without blocking




(a)



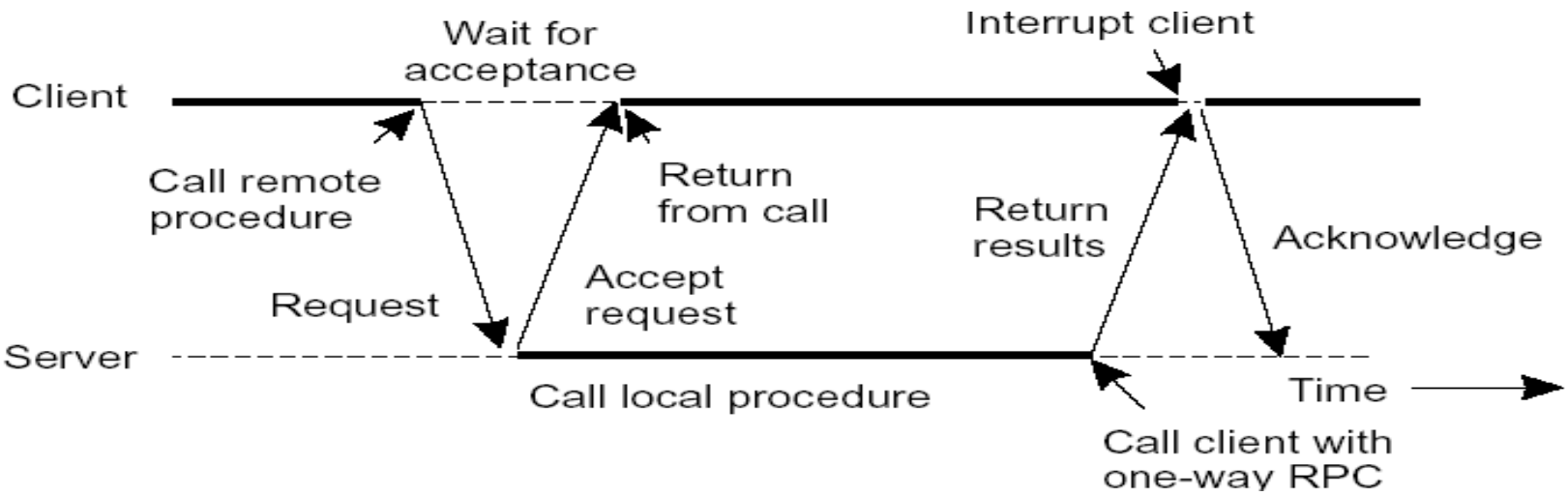

(b)

a) the interconnection between client and server in a traditional RPC

b) the interaction using asynchronous RPC

2. If the result can be collected later

- e.g., prefetching network addresses of a set of hosts,
- the server immediately sends an Ack promising that it will carryout the request.
- the client can now proceed without blocking
- the server later sends the result.




A client and server interacting through two asynchronous RPCs

# Cont....

- The above method combines two asynchronous RPCs and is sometimes called deferred synchronous RPC
- Variants of asynchronous RPC
  - ✓ let the client continue without waiting even for an Ack, called one-way RPC
  - ✓ problem: if reliability of communication is not guaranteed.

# Cont....

- With asynchronous RPCs, the server immediately sends a reply back to the client the moment the RPC request is received, after which it calls the requested procedure.
- The reply acts as an acknowledgment to the client that the server is going to process the RPC.
- The client will continue without further blocking as soon as it has received the server's acknowledgment.
- Example of Asynchronous RPC
  - ✓ Transferring money from one account to another,
  - ✓ Adding entries into a database,

# Remote Object (Method) Invocation (RMI)

- RMI is an API used to build distributed application
  - ✓ It is an expansion of the RPC mechanisms
- It's allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine.
- Through RMI, an object running in a JVM present on a computer (Client-side) can invoke methods on an object present in another JVM (Server-side).
  - The RMI allows an object to invoke methods on an object running in another JVM.

# Remote Object (Method) Invocation (RMI)

- RMI enhances **distribution transparency** as a consequence of an object that hides its internal from the outside world by means of a well-defined interface.
- Distributed Objects
  - An object encapsulates data, called the state, and the operations on those data, called methods
  - Methods are made available through interfaces.
  - State of an object can be manipulated only by invoking methods
  - This allows an interface to be placed on one machine while the object itself resides on another machine;
    - ✓ such an organization is referred to as a distributed object

# Cont....

- The state of an object is not distributed, only the interfaces are; such objects are also referred to as remote objects.
- The RMI provides remote communication between the applications using two objects stub and skeleton.
- It uses this objects for communication with the remote object.
- A remote object is
  - an object whose method can be invoked from another JVM.
- RMI is a pure java solution to Remote Procedure Calls (RPC) and is used to create a distributed application in java.
- The Stub and Skeleton objects are used for communication between the client and server side.

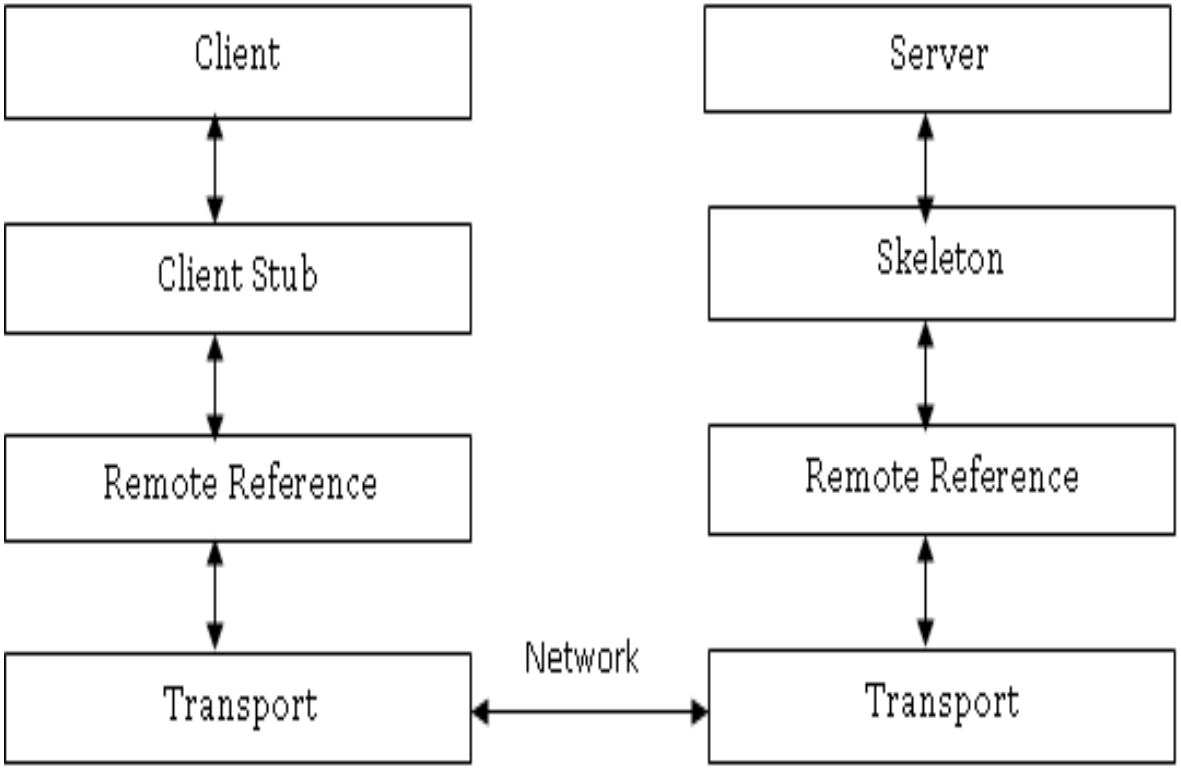
# Cont….

- **Stub**
  - It is an object that acts as a gateway for the client side.
  - All the outgoing requests are routed through it.
  - It resides at the client side and represents the remote object.
- When the caller(client) invokes a method on the stub object, it does the following tasks:
  - It initiates a connection with a remote JVM,
  - It writes and transmits (marshals) the parameters to the remote JVM.
  - It waits for the result
  - It reads (un marshals) the return value or exception, and
  - It finally, returns the value to the caller.

# Cont..

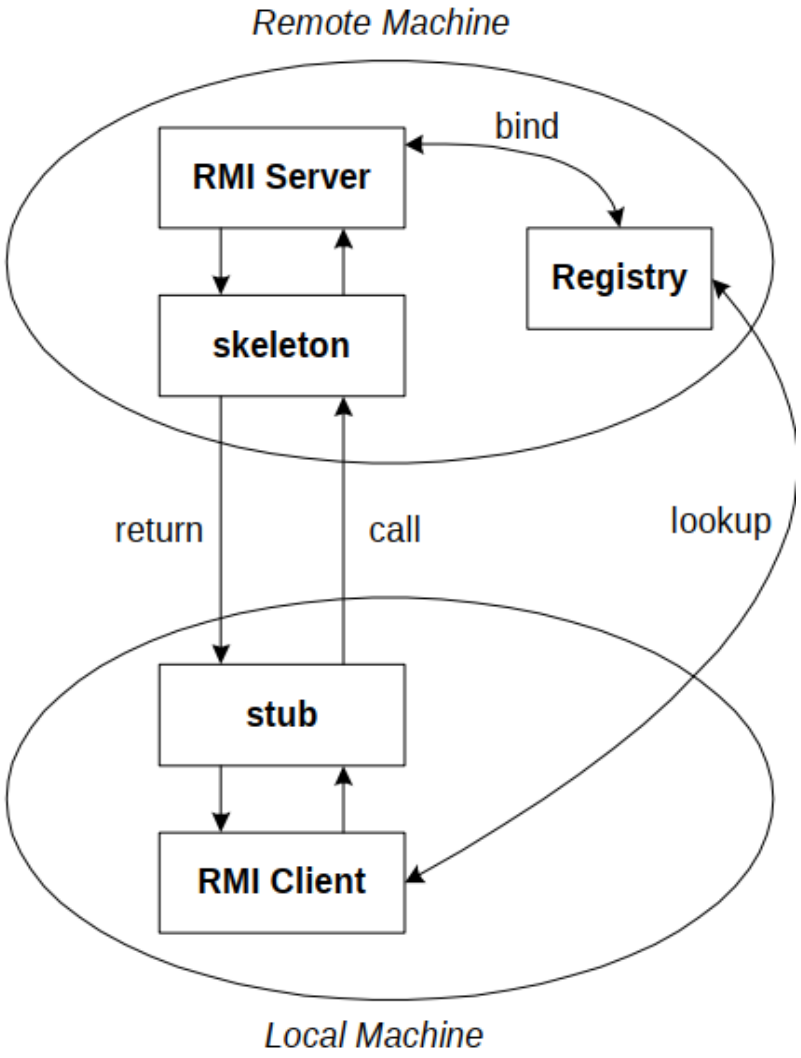
- Skeleton
  - It is an object, that acts as a gateway for the server-side object.
  - All incoming requests are routed through it.
- When the skeleton receives the incoming request, it does the following tasks:
  - It reads the parameter for the remote method.
  - It invokes the method on the actual remote object, and
  - It writes and transmits (marshals) the result to the caller.

# RMI Architecture and Components

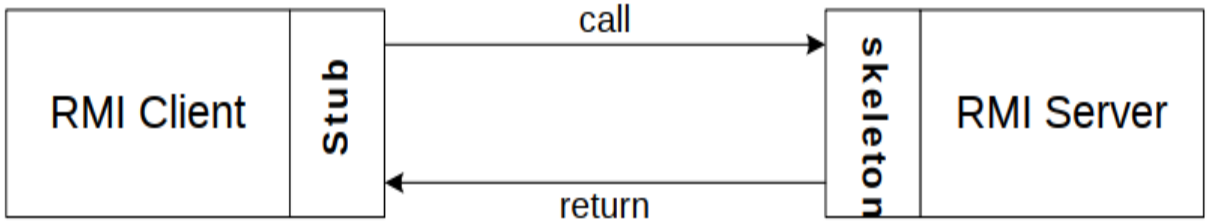


# Cont..

- General RMI architecture
  - The server must first **bind** its name to the registry
  - The client **lookup** the server name in the registry to establish remote references.
  - The **stub** serializes the parameters to the skeleton, the **skeleton** invoking the remote method and serializing the result back to the stub.



# Cont….




- A client invokes a remote method, and the call is first forwarded to the stub.
- The **stub** is responsible for sending the remote call over to the server-side skeleton
  - The stub opens a socket to the remote server, marshaling the object parameters, and forwarding the data stream to the skeleton.
- A **skeleton** contains a method that receives the remote calls, unmarshals the parameters, and invokes the actual **remote object implementation**.

# Java RMI Application-Example

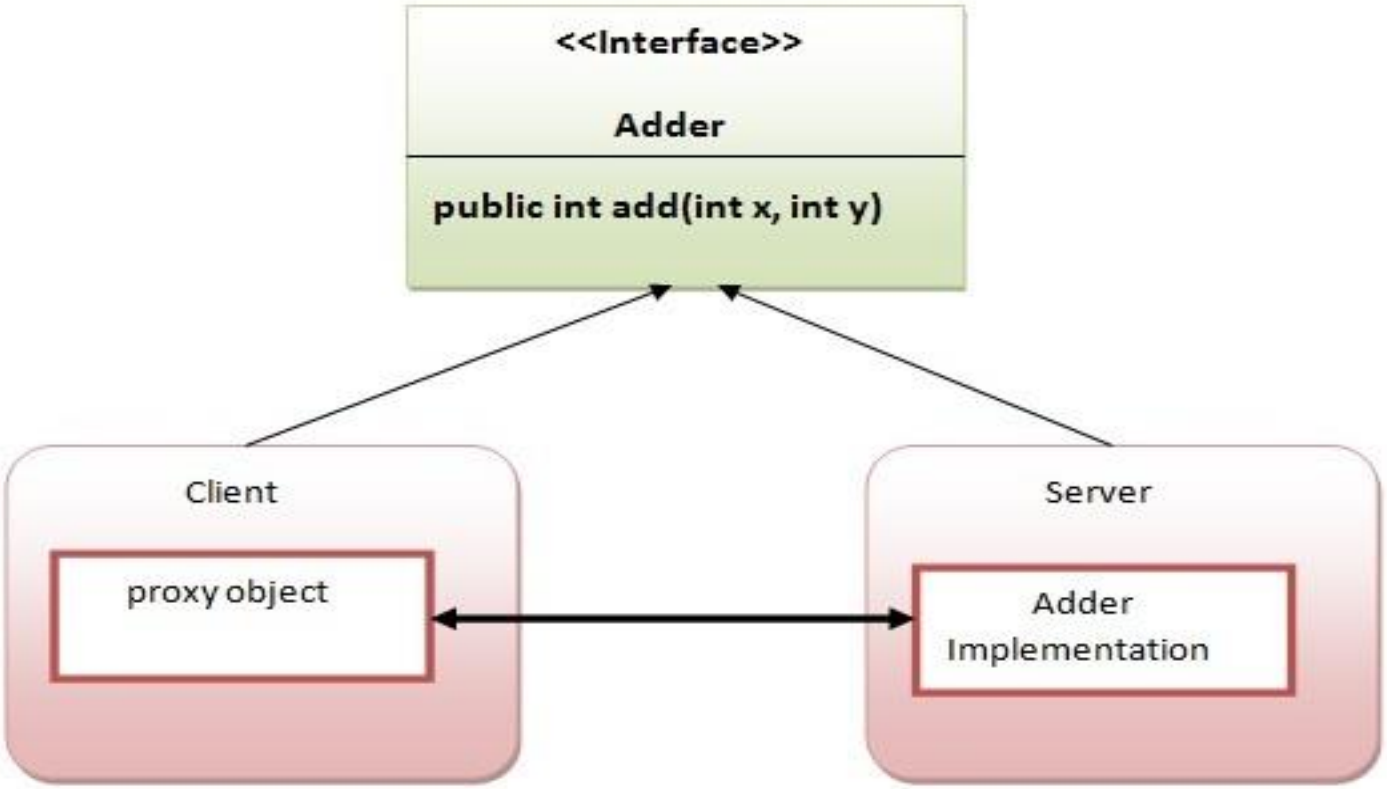
- **Steps for developing an RMI Application**
  1. Create a remote interface that extends the **Remote**
  2. Create a class that implements the remote interface
  3. Create and run the server application
  4. Create and run the client application

# Java RMI Application-Example

- In this example, we have followed the all steps to create and run the RMI application.
  - The client application needs only two files, **the remote interface**, and the **client application**.
  - In the RMI application, both client and server interact with the **remote interface**.
  - The **client application** invokes methods on **the proxy object**, and RMI sends the request to **the remote JVM**.
  - The return value is sent back to **the proxy object** and then to the client application.

# Java RMI Application- Example

- Create a simple **Java RMI application** that **adds two integers** numbers as shown by the following diagram.



# 1. Create remote interface that extends Remote

- A **remote interface** provides the description of all the methods of a particular remote object.
- The **client** communicates with this **remote interface.**
- **To create a remote interface:-**
  - Create an interface that extends the predefined interface **Remote** which belongs to the package.
  - Declare all the business **methods** that can be **invoked** by the client in this interface.
  - Since there is a chance of **network issues** during remote calls, an exception named **RemoteException** may occur; throw it.

# 1. Create remote interface that extends Remote

```
import java.rmi.*;

public interface Adder extends Remote{

public int add(int x,int y)throws RemoteException;

}
```

# 2. Create a class implementing the remote interface

- We need to implement the remote interface created in the earlier step. (We can write an implementation class separately or we can directly make the server program implement this interface.)
- **To develop an implementation class:-**
  - Implement the interface created in the previous step.
  - Provide implementation to all the abstract methods of the remote interface.
- **Note:** Interface methods by default are public and abstract.
  - Abstract methods do not have a body (the body is provided by the implementing class).

# 2. Create a class implementing the remote interface

- To provide the implementation of **the remote interface** we need to either extend the **UnicastRemoteObject**s class or use (export) the **exportObjects()** method of the UnicastRemoteObjects calss.

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class AdderRemoteImpl extends UnicastRemoteObject
implements Adder{
// Default constructor to throw RemoteException  from its parent constructor
AdderRemoteImpl()throws RemoteException{
super();     //to inherit feature of superclass
}
public int add(int x, int y) throws RemoteException
{
   return x+y;
}
}
```

# 3. Create and run the server application

- Next step is to create **stub and skeleton objects** using the RMI compiler.
  - The **rmic tool** invokes the RMI compiler and creates stub and skeleton objects from the **implementation class**.
  - Syntax: `rmic classname //command prompt`
- The **Naming class** provides
  - methods to get and store the remote object.
- **Start the rmiregistry:** next is to start the registry service by using the rmiregistry tool.
  - Syntax: `start rmiregistry //command prompt`

# 3. Create and run the server application

- Here, we should create **a remote object** and **bind** it to the **RMIregistry**.
- To develop **a server program** :-
  - Create a client class from where you want to invoke **the remote object**.
  - Create **a remote object** by instantiating the **implementation class** as shown below.
  - Export the **remote object** using the method **exportObject()** of the class named **UnicastRemoteObject** which belongs to the package **java.rmi.server**.
  - Get the RMI registry using the **getRegistry()** method of the **LocateRegistry** class which belongs to the package **java.rmi.registry**.

# 3. Create and run the server application

- Bind the **remote object created** to the registry using the **bind() method** of the class named **Registry(Naming)**.
- The **bind method** is used to bind the remote object to the **new name.**
  - ✓To this method, pass a string representing the bind **new name** and **the object exported**, as **parameters**.

# 3. Create and run the server application

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class MyServer {
public static void main(String[] args) {
  try {
//Create a local instance of the object/sekeleton
AdderRemoteImpl obj = new AdderRemoteImpl();
//put the local instance in the registry with new name
Naming.rebind("rmi://localhost:1099/lab2_rmi",obj);
System.out.println("Server is connected and ready for
   operation.");
  } catch (Exception e) {
  System.out.println("Server not connected: " + e);
  }}}
```

# 3. Create and run the client application

- **To develop a client program:-**
  - Create a client class from where you are intended to invoke the remote object.
  - Get the RMI registry using the **getRegistry()** method of the **LocateRegistry** class which belongs to the package **java.rmi.registry**.
  - Fetch the object from the registry using the method **lookup()** of the class **Registry** which belongs to the package **java.rmi.registry**.
    - ✓ To this method, you need to **pass** a string value representing the bind name as a parameter.
    - ✓ This used will **return** you the reference of **the remote object**.
  - The **lookup()** returns an object of type remote, and downcast it to the type Adder. Finally, invoke the required method using the obtained **remote object**.

# 4. Create and run the client application

```
import java.rmi;
public class MyClient {
public static void main(String[] args) {
//Get or find a remote object from the registry
 try {
Adder stub =
   (Adder)Naming.lookup("rmi://localhost:1099/lab2_rmi");
 System.out.println("remote object accessed \n"+
"The sum of the two numbers is "+ stub.add(5,7));
 } catch (Exception e) {
  System. out.println("Client exception: " + e);
  }}}
```



# Cont....

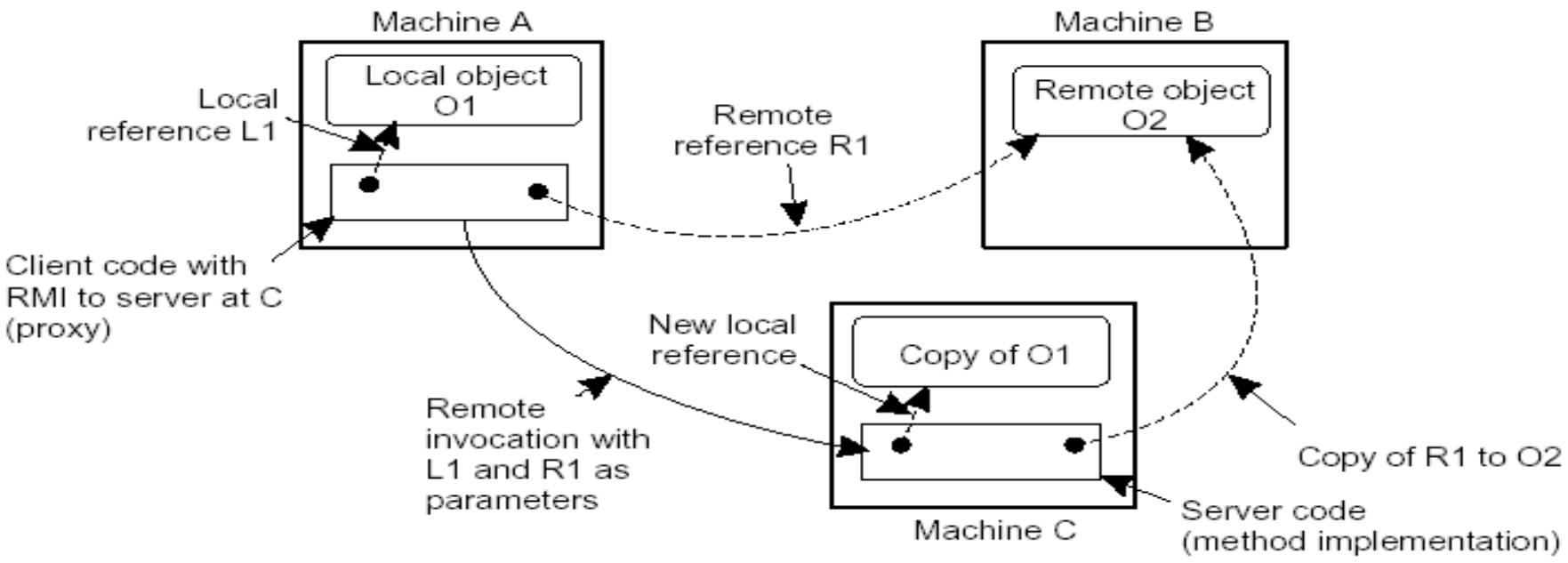
## Binding a Client to an Object

- A process must first bind to an object before invoking its methods, which results in a proxy being placed in the process's address space
- Binding can be implicit (directly invoke methods using only a reference to an object) or explicit (by calling a special function)
- An object reference could contain
  - network address of the machine where the object resides
  - endpoint of the server
  - an identification of which object
  - the protocol used
  - ...

# Cont….

- Parameter Passing
  - There are two situations when invoking a method with object reference as a parameter: the object can be local or remote to the client
  - Local object: a copy of the object is passed; this means the object is passed by value.
  - Remote object: copy and pass the reference of the object as a value parameter; this means the object is passed by reference.

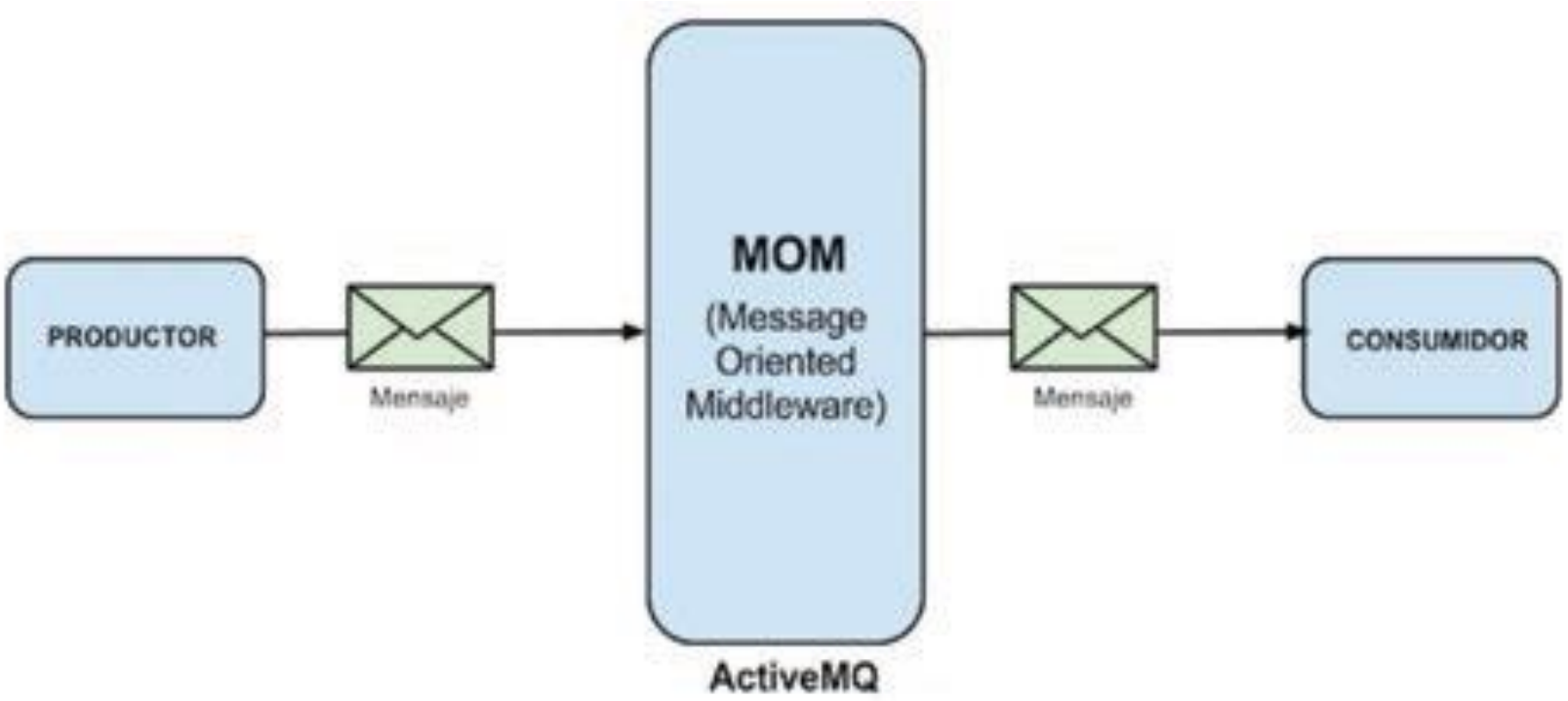
# Cont….




- The situation when passing an object by reference or by value
- Examples:
  - Java RMI

| No | RPC | RMI |
|---|---|---|
| 1. | RPC is a library and OS dependent platform. | Whereas it is a java platform. |
| 2. | RPC supports procedural programming. | RMI supports object-oriented programming. |
| 3. | RPC is less efficient in comparison of RMI. | While RMI is more efficient than RPC. |
| 4. | RPC creates more overhead. | While it creates less overhead than RPC. |
| 5. | The parameters which are passed in RPC are ordinary or normal data. | While in RMI, objects are passed as parameter. |
| 6. | RPC is the older version of RMI. | While it is the successor version of RPC. |
| 7. | There is high Provision of ease of programming in RPC. | While there is low Provision of ease of programming in RMI. |
| 8. | RPC does not provide **any security.** | While it provides **client level security.** |
| 9. | It's development cost is **huge.** | While it's development **cost is fair or reasonable.** |
| 10 | Can be **complex** due to low-level implementation | Generally **simpler** to use and implement |
| | | |
| | | |

# Message-Oriented Middleware (MOM)

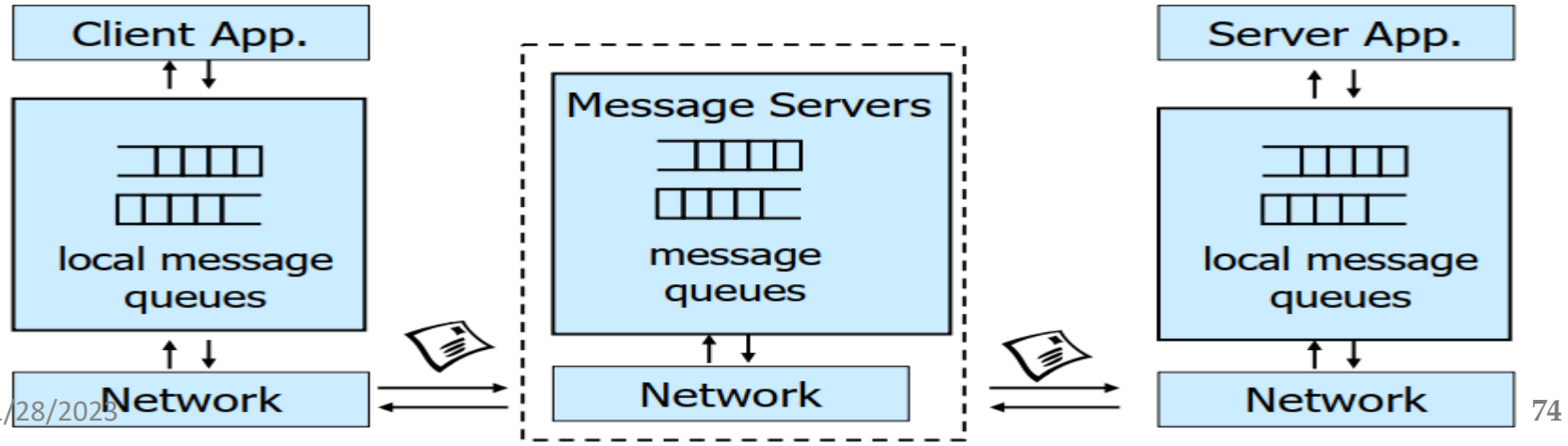


# Message-Oriented Middleware (MOM)

- It is primarily middleware that **facilitates communication** between distributed applications.
- While MOM supports both **synchronous and asynchronous messaging,** it is most closely identified with **asynchronous** messaging using **queuing**.
- MOM sends messages from one application to another using **a queue** as an **interim** step.
  - Client messages are sent to a queue and remain there until they are retrieved by the server application.
- The **advantage** of this system is that the server application does not need to be available when the message is sent, instead, the server can retrieve the message at any time.

# Message-Oriented Middleware (MOM)

- In addition, since messages can be **retrieved off** the queue in any order,
- MOM can also facilitate the retrieval of messages using priority or load-balancing schemes.
  - So MOM, is the Communication using the messages.
- Messages stored in **message queues**.
  - ✓ message servers decouple client and server.
- Various assumptions about message content



# Properties of MOM

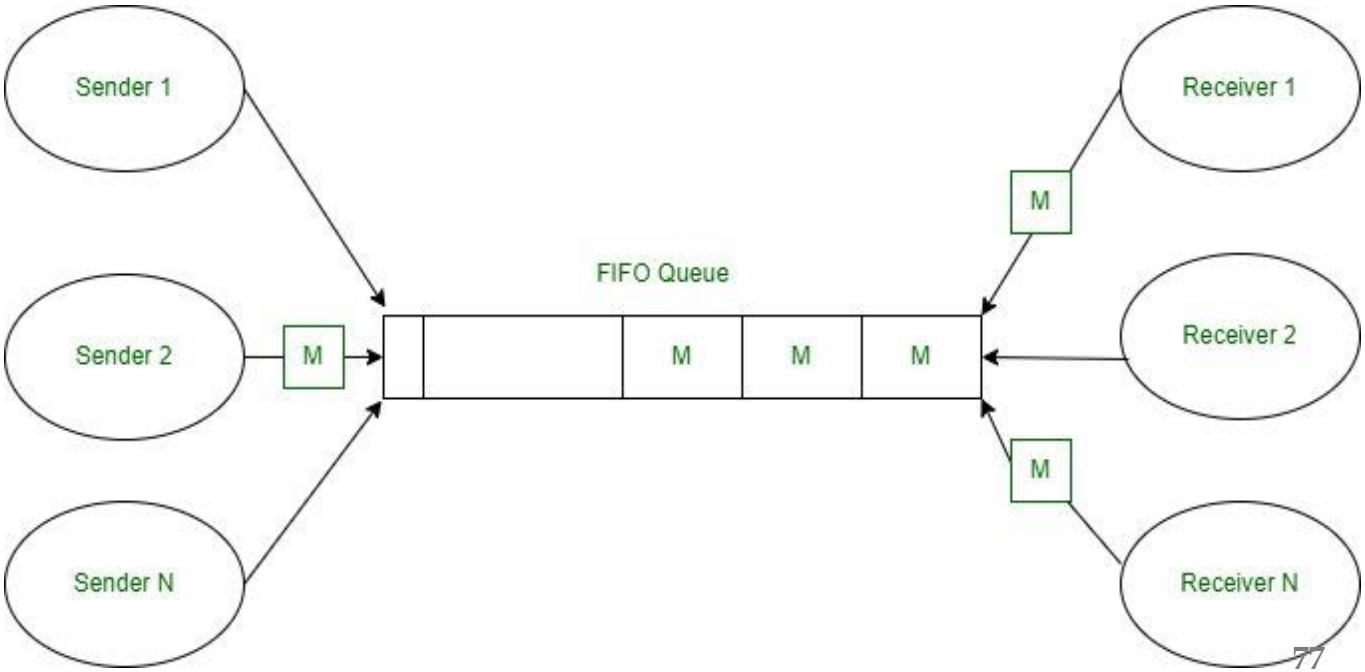
- **Asynchronous interaction**
  - Client and server are only loosely coupled
  - Messages are queued
  - Good for application integration
- **Support for reliable delivery service**
  - It ensures message delivery by using **reliable queues** and by providing the directory, security, and administrative services required to support messaging.
  - Keep **queues** in persistent storage
- **Processing of messages by an intermediate message server(s)**
  - May do filtering, transforming, logging, …
  - Networks of message servers
- **Natural for database integration**

# Disadvantages of MOM

- Poor programming abstraction (but has evolved)
  - ✓ Rather low-level
  - ✓ Request/reply difficult to achieve, but can be **done**.
- Message formats originally unknown to middleware.
  - ✓ No type checking (JMS addresses this – implementation?)
- Queue abstraction only gives one-to-one communication
  - ✓ Limits scalability (JMS pub/sub – implementation?)

# Structure and working of the Message Queue in MOM

- **Message Queue** helps in storing the message on a MOM platform.
- MOM clients can send and receive messages through the queue.
- Queues act as a central component for implementing **asynchronous interaction** within MOM.



# Types of MOM

- Message-Oriented Middleware (MOM) includes:
  - Java Message Service(JMS)
  - IBM WebSphere MQ
  - Web Services

# Stream Oriented Communication

- RPC and message-oriented communication are based on the exchange of discrete messages
  - ✓ Timing might affect performance, but not correctness.
- In stream-oriented communication the message content (multimedia streams)
  - ✓ must be delivered at a certain rate, as well as correctly.
- It provides facilities for the exchange of time-dependent information (continuous media) such as audio and video streams

# Stream Oriented Communication

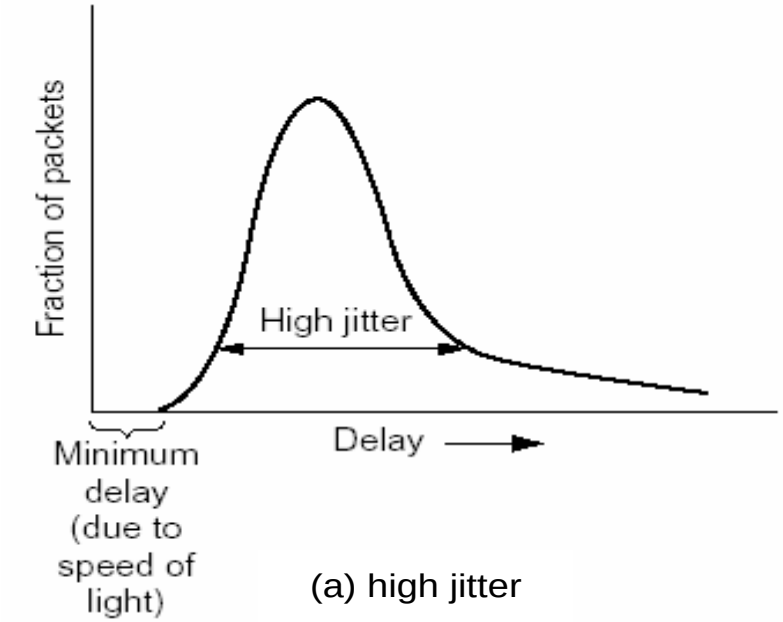
- Media: means by which information is conveyed
- Types of media
  - Discrete media
    - ✓ No temporal dependence between data items
    - ✓ Example text, still images, object code or executable files
  - Continuous media
    - ✓ Temporal dependence between data items
    - ✓ Example audio, video, animation etc.

# Quality of Services (QoS)

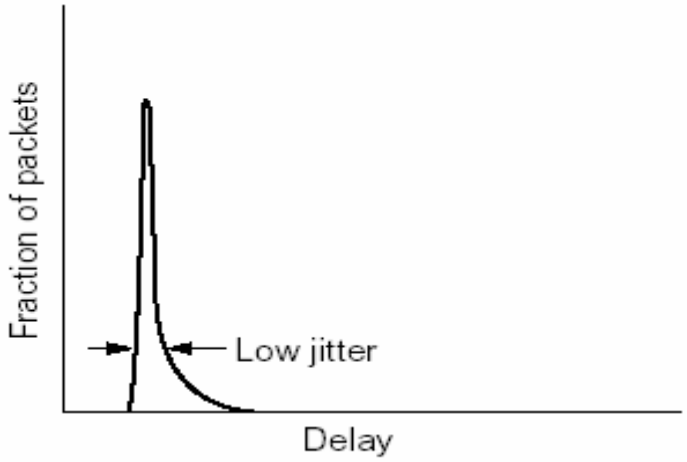
- QoS requirements describe what is needed from the underlying distributed system and network to ensure acceptable delivery;
- It measure communication performance
- QoS refers to flow characteristics
  - Reliability:- lack of reliability means losing a packet or acknowledgement, which entails retransmission for some media types
  - Delay(latency) :- source-to-destination delay
  - Bandwidth :- requirements of applications
  - Jitter :- the variation in the packet (message) arrival times belonging to the same flow.

# Quality of Services (QoS)

- The variation in the time taken to deliver a series of messages.
- It is very relevant to multimedia data




(a) high jitter




(b) low jitter

# Cont....

- Different applications have different flow characteristic requirements

| Application | Reliability | Delay | Jitter | Bandwidth |
|---|---|---|---|---|
| E-mail | High | Low | Low | Low |
| File transfer | High | Low | Low | Medium |
| Web access | High | Medium | Low | Medium |
| Remote login | High | Medium | Medium | Low |
| Audio on demand | Low | Low | High | Medium |
| Video on demand | Low | Low | High | High |
| Telephony | Low | High | High | Low |
| Videoconferencing | Low | High | High | High |

# Cant.....

- Techniques to improve Network QoS.
- There are five common methods:
  - ✓ buffering,
  - ✓ traffic shaping,
  - ✓ scheduling,
  - ✓ admission control, and
  - ✓ resource reservation
- Some are useful for multimedia)

# Individual Assignment I(10%)

- Compile Short Notes on the following topics
  - ✓ Types MOM:
    - ☞ Java Message Service(JMS)
    - ☞ IBM WebSphere MQ
    - ☞ Web Services
  - ✓ Stream Oriented Communication
  - ✓ Discuss the techniques to improve network QoS briefly.