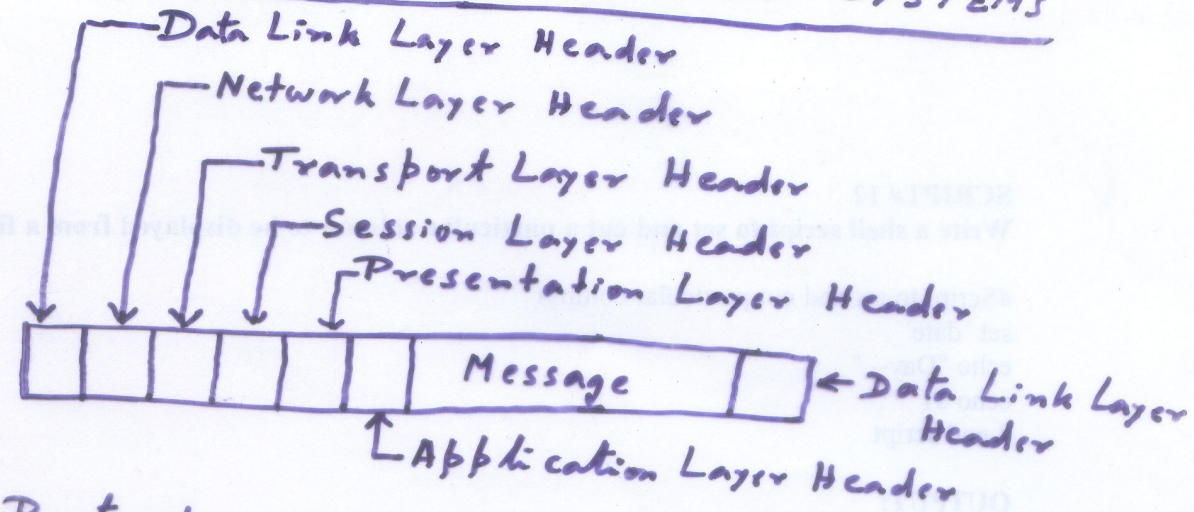
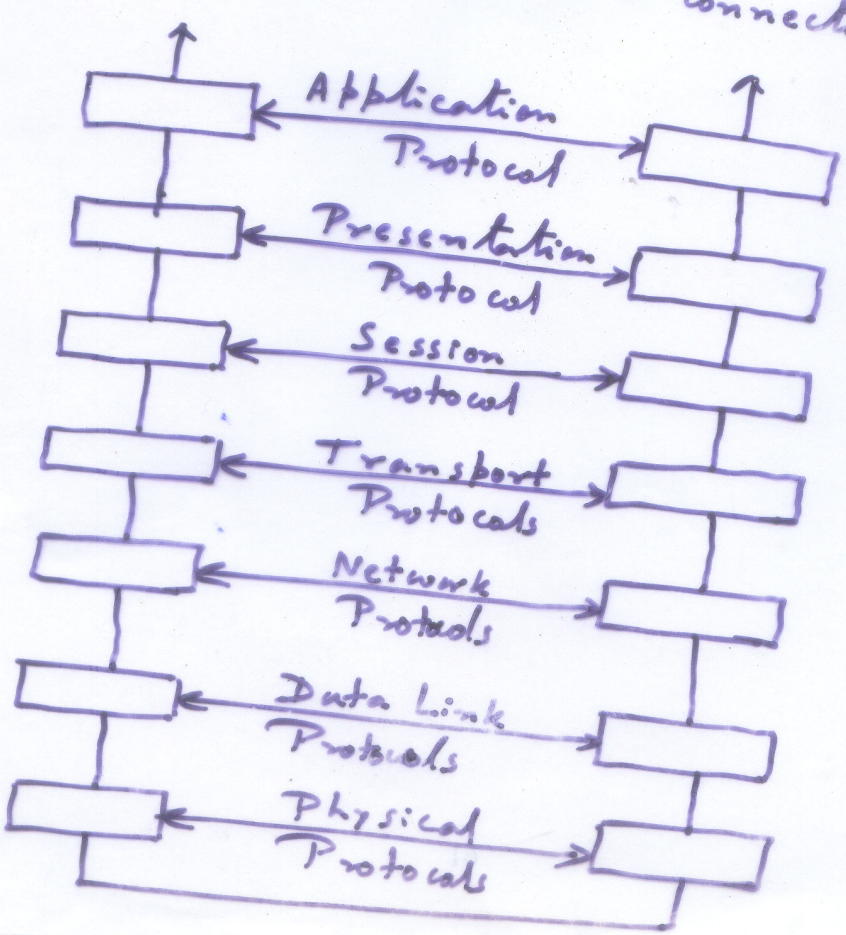
COMMUNICATION IN DISTRIBUTED SYSTEMS



Layered Protocols :-

- Based on ISO - OSI (Open Systems Interconnection) Reference Model
- OSI Model is designed to allow open systems to interconnect or communicate.
- Two general types of protocols - Connection Less
  - Connection Oriented



Lower Level Protocols :-

Physical Layer Protcols. eg: RS-232C standard for Serial lines

Data Link Layer Protocols

Network Layer Protocols eg: IP, Virtual Channel in ATM Networks

Transport Layer Protocols eg: TCP, UDP, (RTP) Real Time Transport Protocol, T/TCP (TCP for Transactions)

Higher Level Protocols :-

Session & Presentation Layer Protocols

Application Protocols eg: FTP, HTTP

MIDDLEWARE PROTOCOLS :-

- Middleware is an application that logically lives in the Application Layer.
- There are protocols to support Middleware Services.
  - Authentication Protocols
  - Authorization Protocols
  - Distributed Commit Protocols
  - Distributed Locking Protocols

ADAPTED REFERENCE MODEL :-

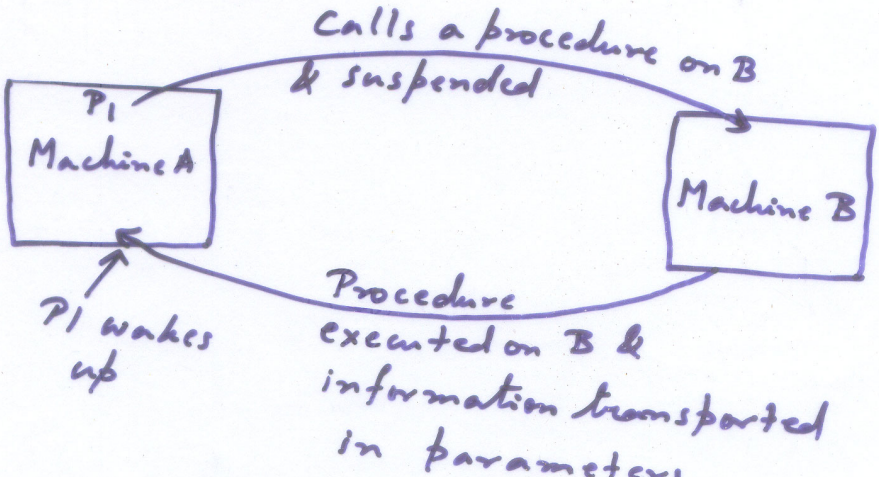
Application
Middleware
Transport
Network
Data Link
Physical

MODELS FOR COMMUNICATION :-

* Remote Procedure Call (RPC)
* Remote Method Invocation (RMI)
* Message Oriented Middleware (MOM)
* Streams

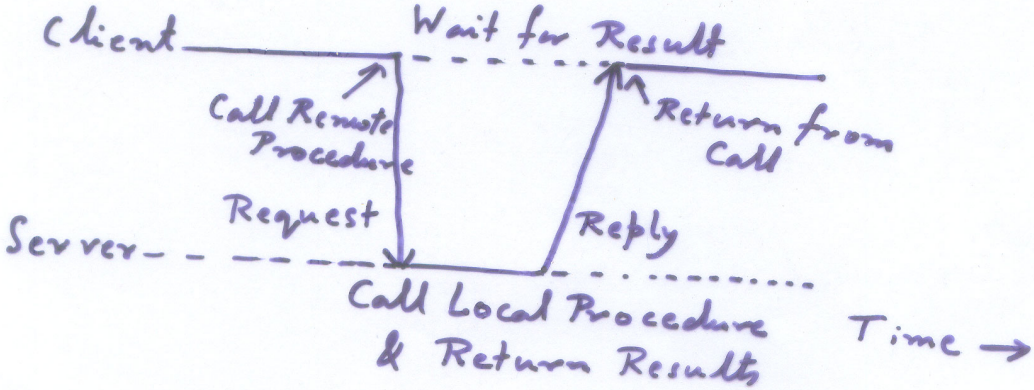
* Remote Procedure Call :-

- Suggested by Birell & Nelson (1984)
- Allows programs to call procedures located on other machines



- Message Passing is not visible to the programmer

Basic RPC Operation :-



- RPC Should be transparent i.e the calling procedure should not be aware that the called procedure is executing on a different machine or vice versa.
- For the purpose Client & Server Stubs are used.
- Client Stub is a piece of code that is equivalent to the calling procedure, which is put into the library.
- Server stub is the server side equivalent of the client stub. It is a piece of code that transforms requests coming in over the network into local procedure calls.
- A RPC occurs in the following steps:

1. The client procedure calls the client stub in the normal way
2. The client stub builds a message and calls the local O.S.
3. The clients O.S sends the message to the remote O.S.
4. The remote O.S gives the message to the server stub
5. The Server stub unpacks the parameters and calls the server.
6. The server does the work and returns the result to the stub.

7. The server stub packs it in a message and calls its local O.S.
8. The server's O.S sends the message to the client's O.S.
9. The client's O.S gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

- The net effect of all these steps is to convert the local call by the client procedure to the client stub, to a local call to the server procedure without either client or server being aware of the intermediate steps.

Parameter Passing :-

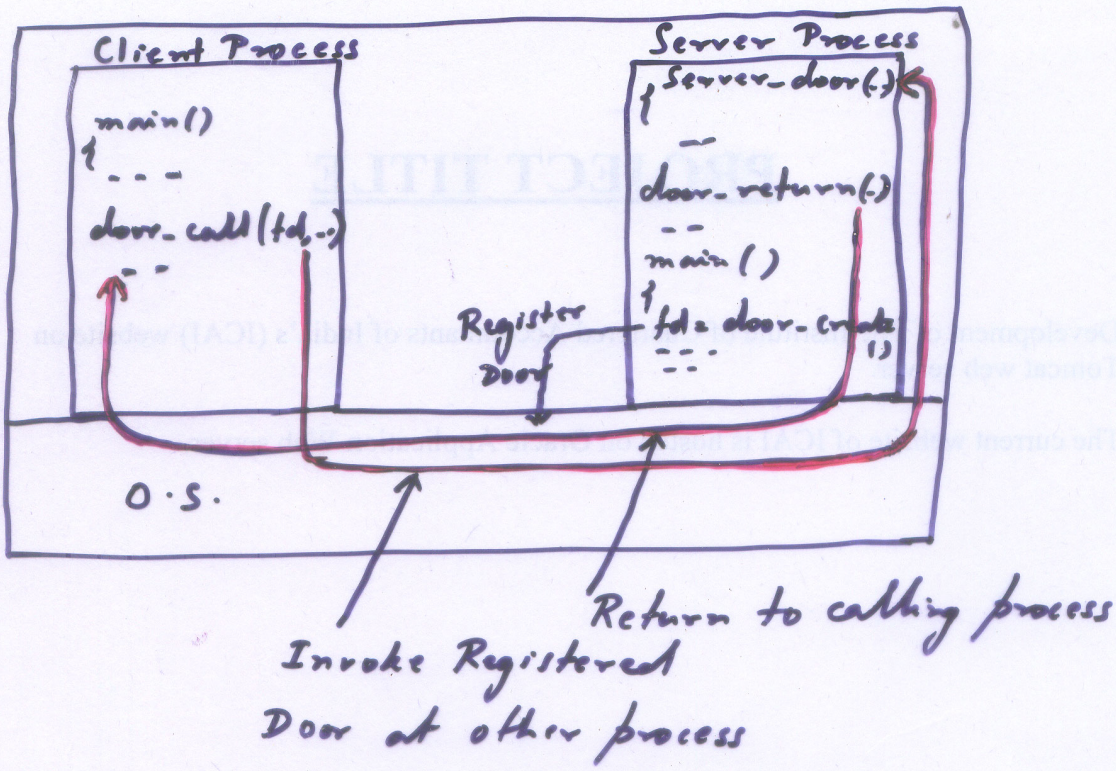
- The function of the client stub is to take its parameters, pack them into a message, and send them to the server stub.
- Packing parameters into a message is called "Parameter Marshaling".
- Passing reference parameters is quite difficult in distributed systems.
  - One solution is just to forbid pointers & reference parameters in general.
  - Some systems deal the case by actually passing the pointer to the server stub & generating special code in the server procedure.

Extended RPC Models :-

* Doors
* Asynchronous RPC

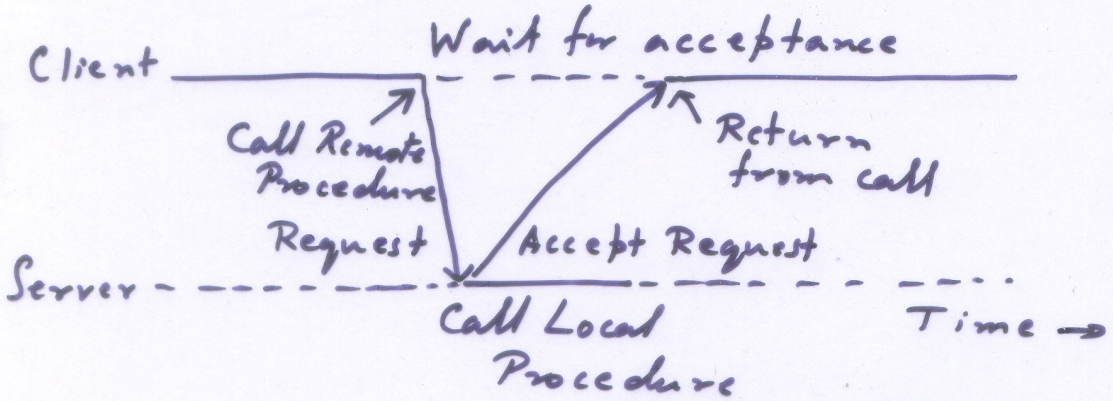
Doors :-

- The original RPC Model assumes that the caller and the callee can communicate only by means of passing messages over a network.
- For the processes running on same machine, Local IPC is used.
- Local IPC is much more efficient than networking facilities.
- A few O.S offers an equivalent of RPC for processes that are colocated on the same machine, called 'Doors'.
- A door is a generic name for a procedure in the address space of a server process that can be called by processes colocated with the server.
- Doors are originally designed for Spring O.S (Mitchell et al)
- A similar mechanism, called Lightweight RPC was developed by Bershad et al.
- Calling doors require support from Local O.S.
- The main benefit of doors is that they allow the use of a single mechanism, namely procedure calls for communication in a distributed system.

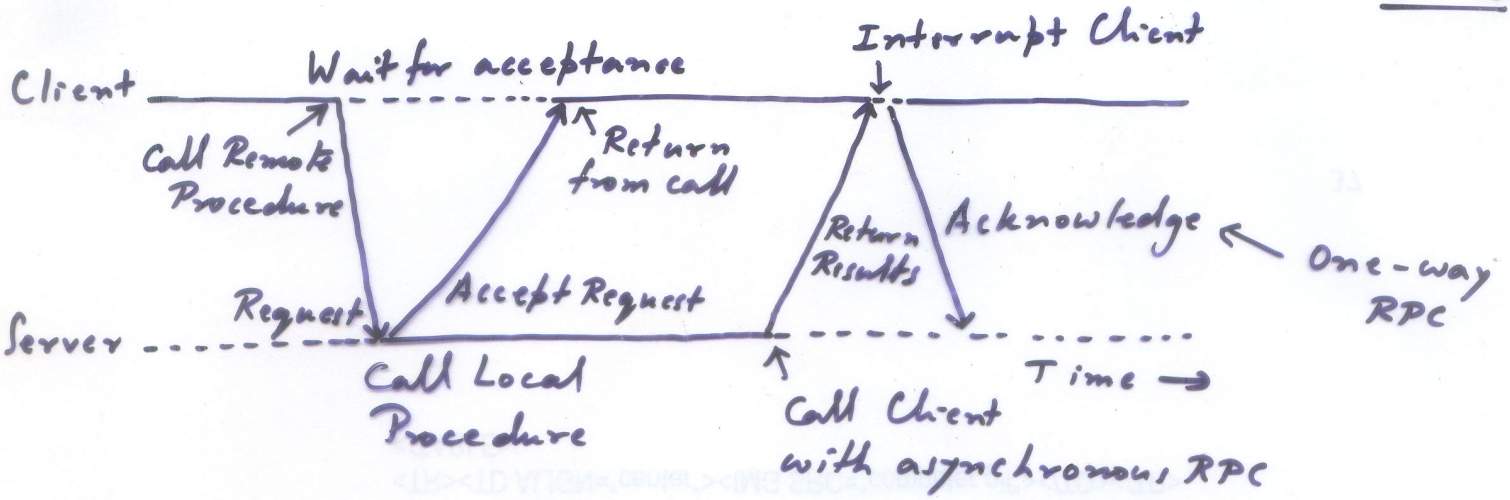


Asynchronous RPC :-

- In conventional procedure calls, when a client calls a remote procedure, the client will block until a reply is returned. This behaviour in sometimes unnecessary.
- In asynchronous RPC, a client immediately continues after issuing the RPC request.



- Combining two asynchronous RPC's is sometimes also referred as "Deferred Synchronous RPC".

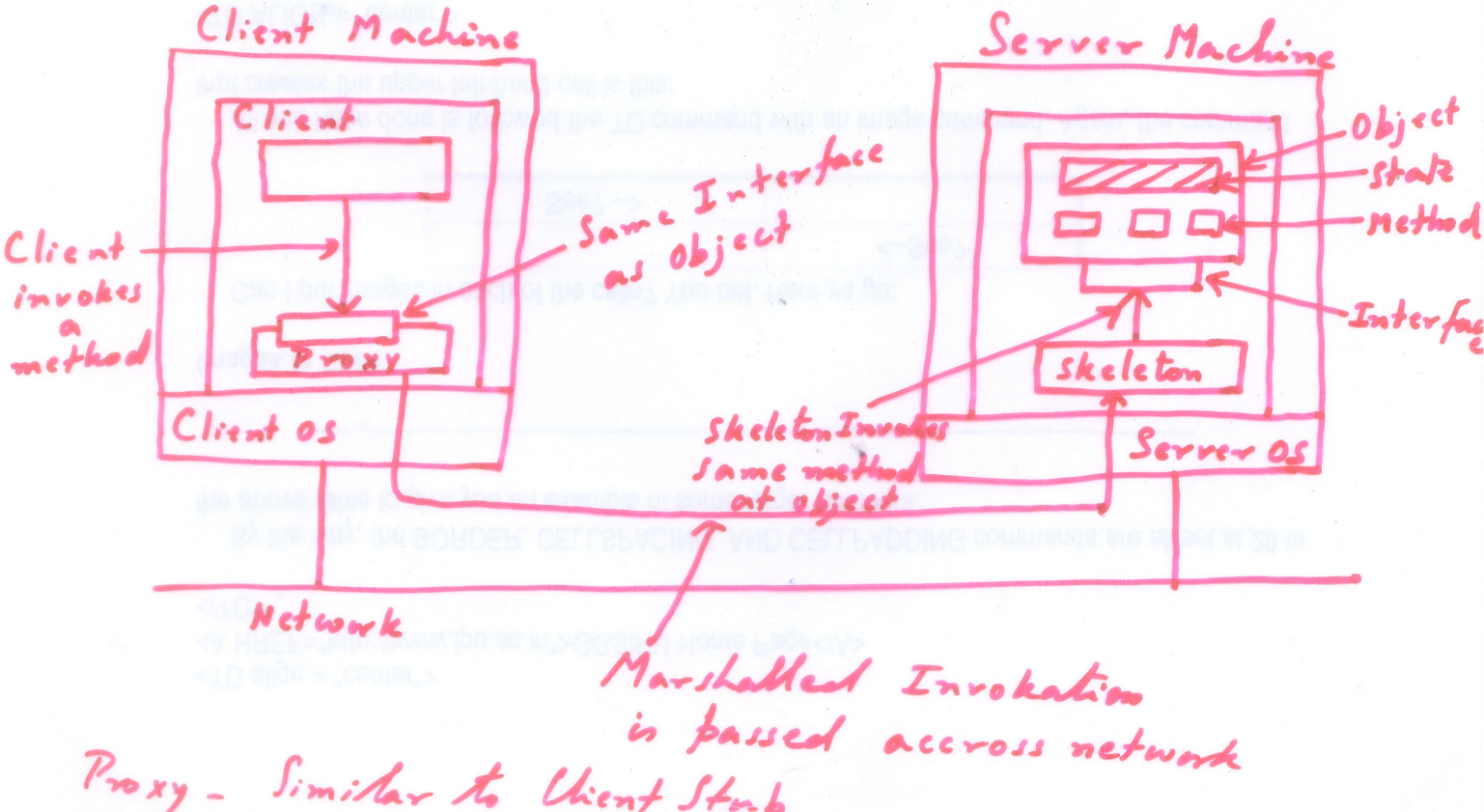


REMOTE OBJECT INVOCATION :-

- Using RPC's Remote Objects are invoked.
- eg: CORBA, DCOM

Distributed Objects :-

- Interfaces (thro' which the methods are made available) are placed in one system while the objects reside on another machine.



Proxy - Similar to Client Stub
Skeleton - Similar to Server Stub

- When a client binds to a distributed object, an implementation of the object's interface, called a proxy is loaded into the clients address space.
- A proxy is analogous to a client stub in RPC Systems.
- The only thing it does is marshal method invokations into messages and unmarchal reply messages to return the result of the method invokation to the client.
- The actual object resides at the server machine, where it offers the same interface as it does on the client machine.
- Incoming invocation requests are first passed to a server stub, often referred to as a skeleton, which unmarshals them to proper method invokations at the object's interface at the server.
- The server stub is also responsible for the marshalling replies and forwarding reply messages to the client.

* Most distributed objects are not distributed. They reside at a single machine. Only the interfaces implemented by the object are made available on other machines.
* Such objects are also referred to as <u>remote objects</u>.

Compile time vs Runtime Objects:-

- Compile time objects are defined as the instance of a class. eg: Java, C++.
- Runtime objects are generally used for constructing distributed objects, as they are independent of programming language.
- When dealing with runtime objects, how objects are actually implemented is basically left open.
- An object whose methods can be invoked from a remote machine, is implemented by object adapter.
- Object adapter acts as a wrapper around the implementation with the sole purpose to give it the appearance of an object

Persistent & Transient objects:-

- A persistent object is one that continues to exist even if it is currently not contained in the address space of a server process.
- A transient object is an object that exists only as long as the server that manages it.

Binding a client to an object :-

- Implicit binding : the client is offered a simple mechanism that allows it to directly invoke methods using only a reference to an object.
- Explicit binding : the client should first call a special function to bind to the object before it can actually invoke its methods. Explicit binding generally returns a pointer to a proxy that is then become locally available.

Static & Dynamic RMI :-

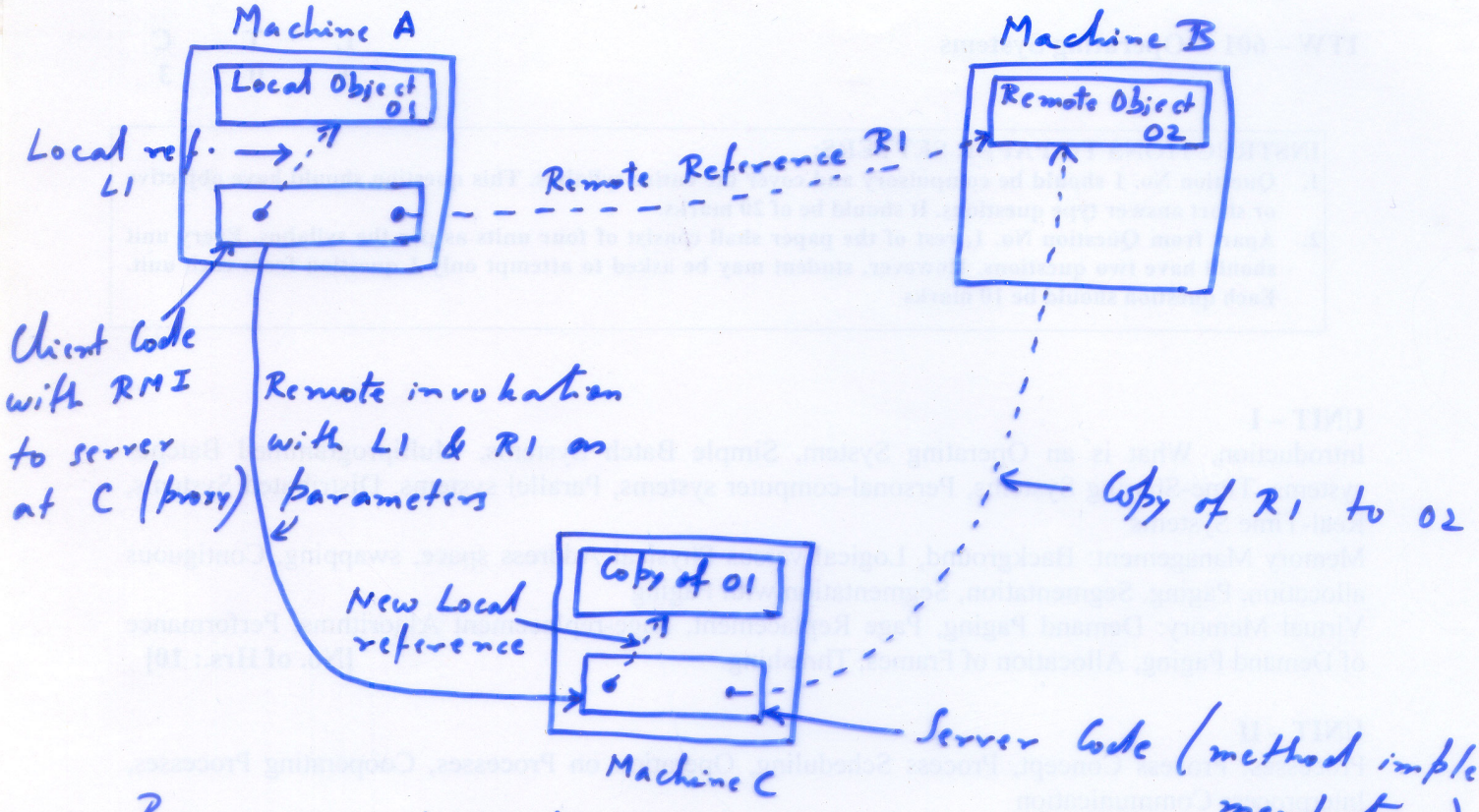
- Using predefined interface definitions to provide RMI support is referred to as Static invokation.
- Static invokation requires that the interfaces of an object are known when the client application is being developed.
- To compose a method invokation at runtime, also referred to as dynamic invokation.
- The essential difference with static invokation is that an application selects at runtime which method it will invoke at a remote object.

* invoke (object, method, input parameters, output parameters)

Eg: invoke (fobject, id(append), int) (dynamic)

fobject.append(int) (static)

Parameter Passing:-

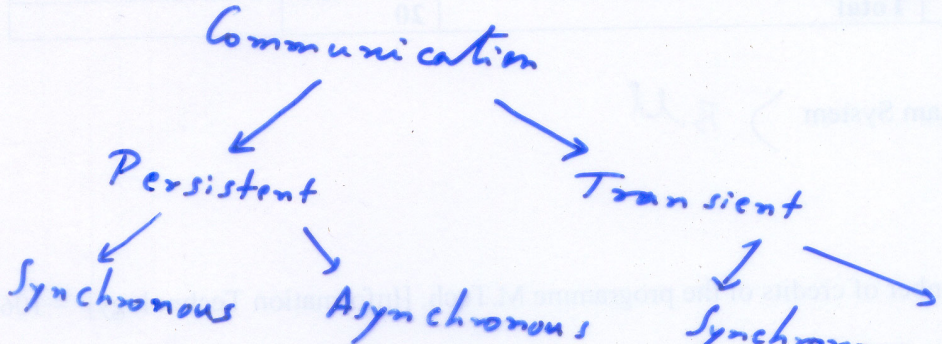


Passing an object by reference or by value

- Because most RMI systems support systemwide object references, passing parameters in method invokations is generally less restricted than in the case of RPC's.
- Figure shows a client program running on machine A and a server program on machine C. The client has a reference to a local object O1 that it uses as a parameter when calling the server program on machine C.
- It also holds a reference to a remote object O2 residing at machine B, which is also used as a parameter.
- When calling the server, a copy of O1 is passed along with only a copy of the reference to O2 to the server on machine C.

Message Oriented Communication :-

- RPC's and Remote object invokations contribute to hiding communication in distributed systems. i.e they enhance access transparency.
- But neither method is always appropriate as due to their synchronous nature.
- In such cases message oriented communication is used
-

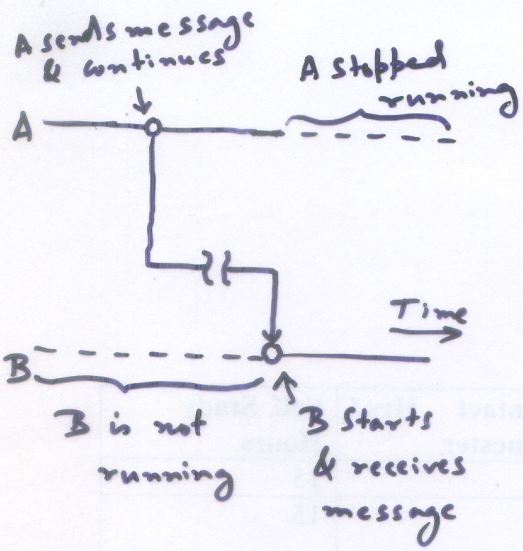


Persistent → A message that has been submitted for transmission is stored by the communication system as long as it takes to deliver it to the receiver.

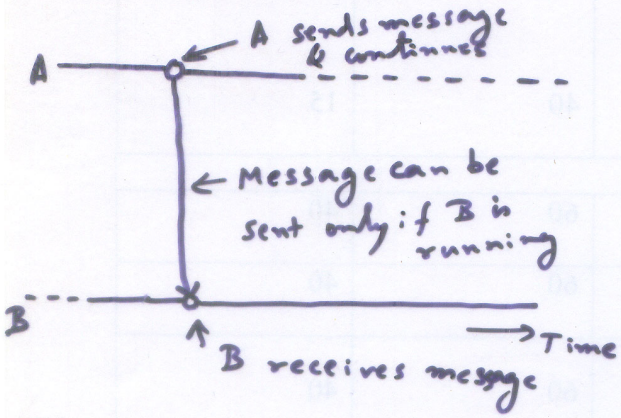
Transient → A message is stored by the communication system only as long as the sending and receiving application are executing.

Synchronous → The sender is blocked until its message is stored in a local buffer at the receiving host, or actually delivered to the receiver.

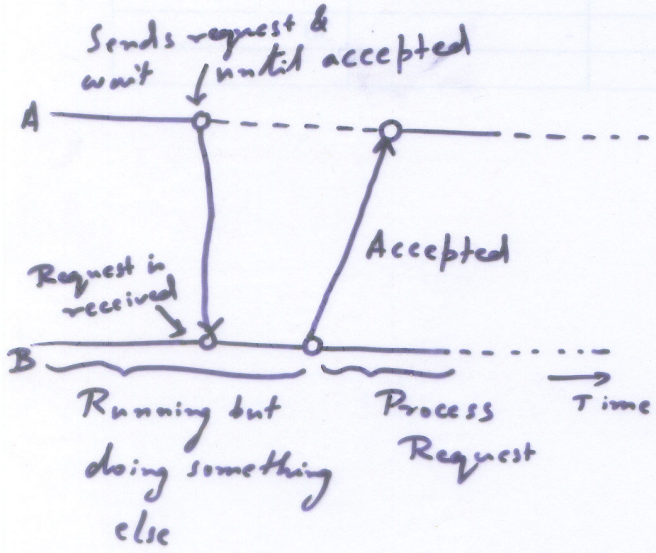
Asynchronous → Sender continues immediately after it has submitted its message for transmission.



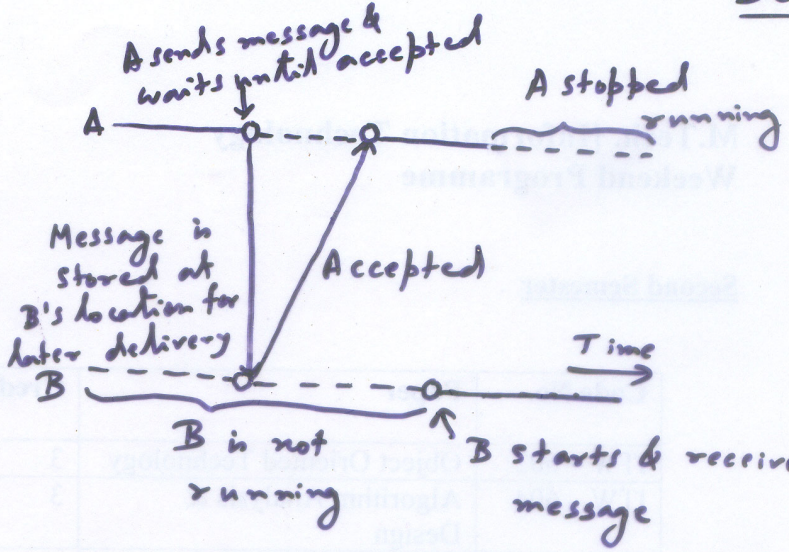
Persistent Asynchronous Communication



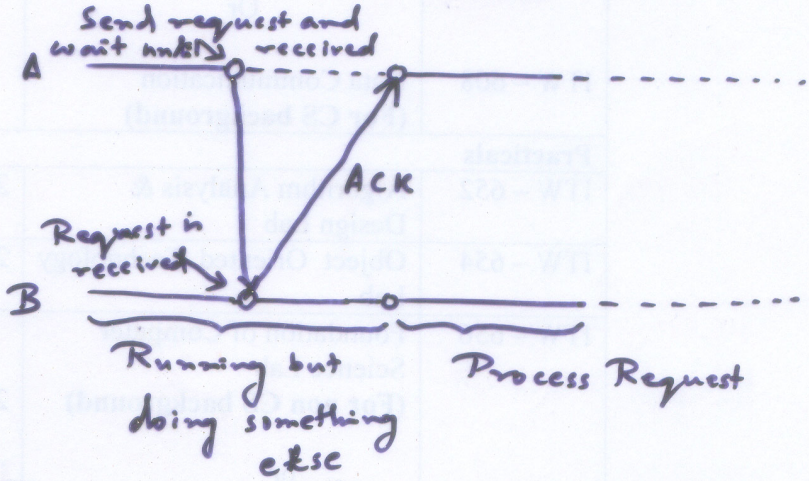
Transient asynchronous communication



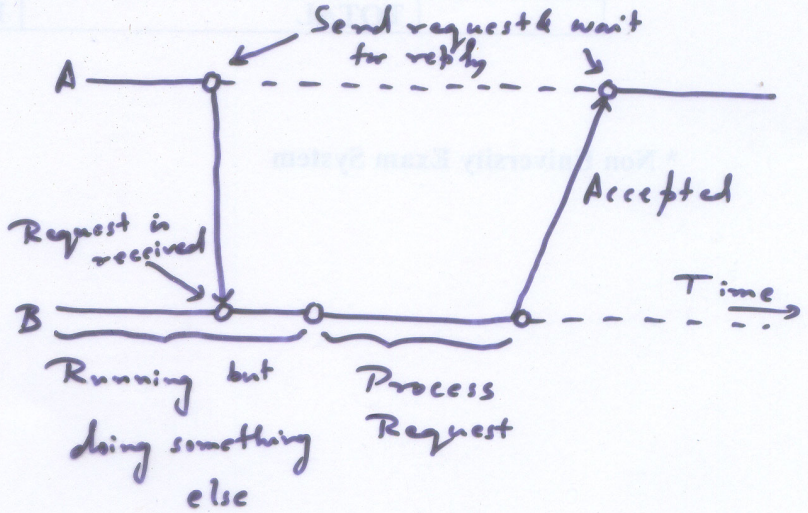
Delivery based transient synchronous communication



Persistent synchronous Communication



Receipt based transient synchronous communication



Response based transient synchronous communication

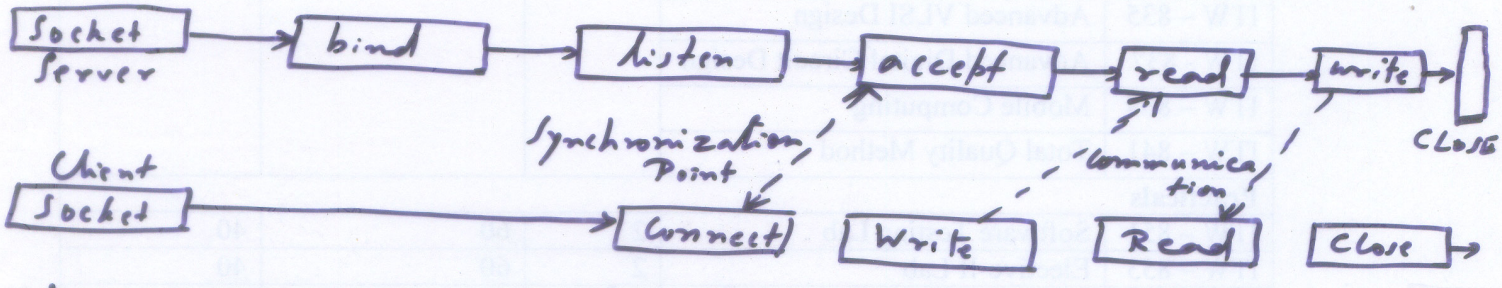
Message Oriented Transient Communication :-

- Many distributed systems & applications are built directly on top of the simple message oriented model offered by transport layer.

Berkeley Sockets :-

- Transport Level
- Socket is a communication end point to which an application can write data that are to be sent out over the underlying network, and from which incoming data can be read.

Primitive	Meaning
Socket	Create a new communication end point.
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection



Message Passing Interface (MPI) :-

- Designed for parallel applications & tailored to transient communication.
- Communication takes place within a known group of processes. Each group is assigned an identifier. A (group-id, process-id) is used instead of transport level address for communication

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_Send	Send a message & wait until copied to local or remote buffer
MPI_ssend	Send a message & wait until receipt starts
MPI_Sendrecv	Send a message & wait for reply
MPI_isend	Pass reference to outgoing message and continue
MPI_issend	Pass reference to outgoing message and wait until receipt starts
MPI_recv	Receive a message, block if there is none
MPI_irecv	Check if there is an incoming message, but do not block.

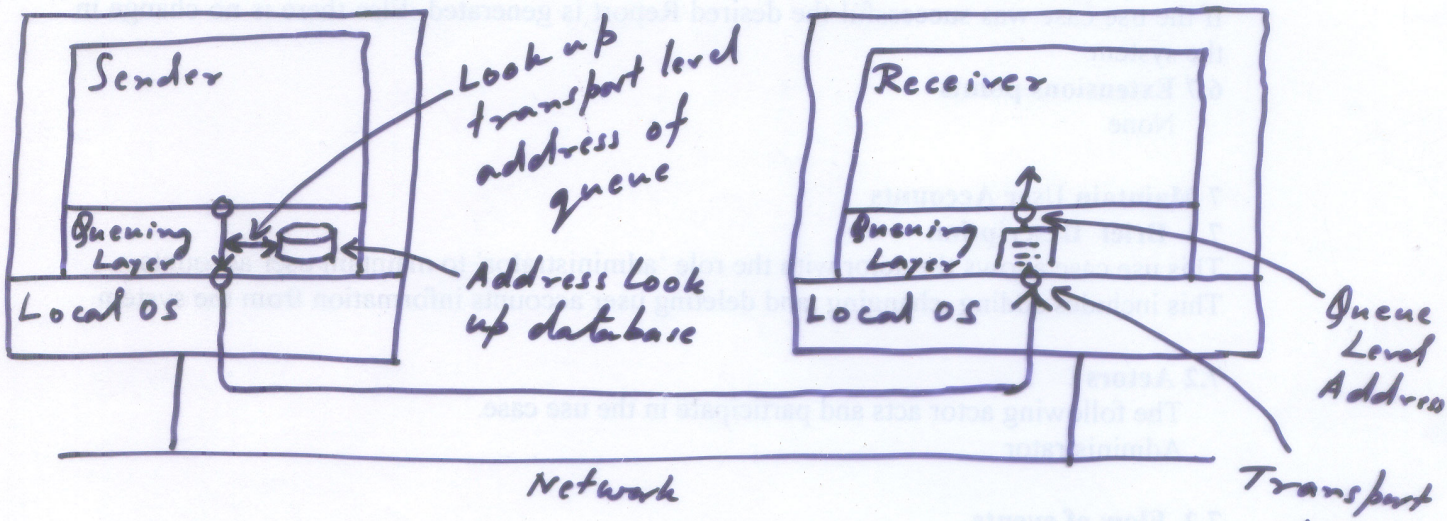
Message Oriented Persistent Communication :-

- Message-queuing Systems or Message Oriented Middleware (MOM)
- Provide extensive support for Persistent Asynchronous communication
- These systems offer intermediate term storage capacity for messages, without requiring either the sender or receiver to be active during message transmission.

- An important difference with Sockets & MPI is that message queuing systems are typically targeted to support message transfers that are allowed to take minutes instead of seconds or milliseconds.
- The basic idea behind a message queuing system is that applications communicate by inserting messages in specific queues.
- These messages are forwarded over a series of communication servers and are eventually delivered to the destination.
- Different primitives used in this system are-

Primitive	Meaning
Put	Append a message to a specific queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first, never block
Notify	Install a handler to be called when a message is put into the specified queue

General Architecture of a Message Queuing System :-



* Source Queue & Destination Queue
* Database - contains queue names to network locations
* Queue Manager - Manages queues
* Special queue managers operate as routers
* Relays - forward incoming messages to other queue managers

Stream Oriented Communication :-

- Used in Communication of Continuous Media (eg: Multimedia files)
- Eg: To reproduce the original sound, it is essential that the samples in the audio stream are played out in the order they appear in the stream.
- Data Stream - Sequence of data units.

Transmission Modes:-

Asynchronous Transmission Mode - The data items in a stream are transmitted one after another, but there are no further timing constraints on when transmission of items should take place.

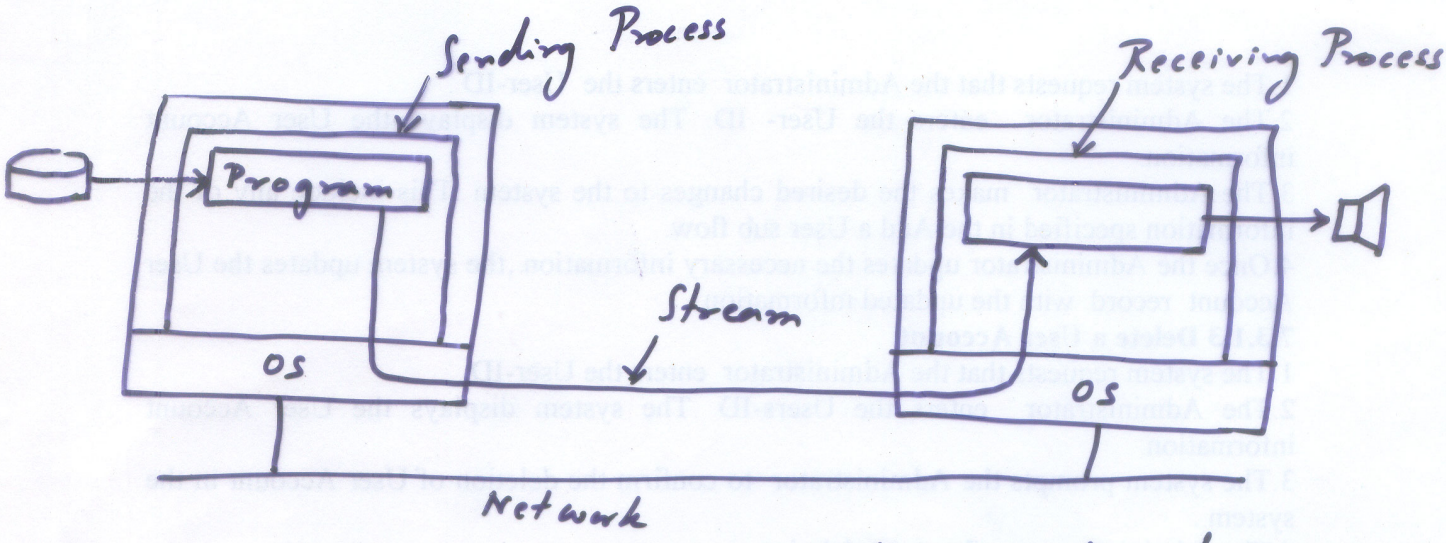
Synchronous Transmission Mode:- There is a maximum end-to-end delay defined for each unit in a data stream. Whether a data unit is transferred much faster than the maximum tolerated delay is not important.

Isochronous Transmission Mode:- It is necessary that data units are transferred on time. This means that data transfer is subject to a maxm & minim end-to-end delay, also referred to as bounded (delay) jitter.

* Streams:-

- Simple Stream - Consists of only a single sequence of data.
- Complex Stream - Consists of several related simple streams, called substreams.

* Synchronization between streams is an important issue in Distributed communication.



Setting up a stream between two processes across a network