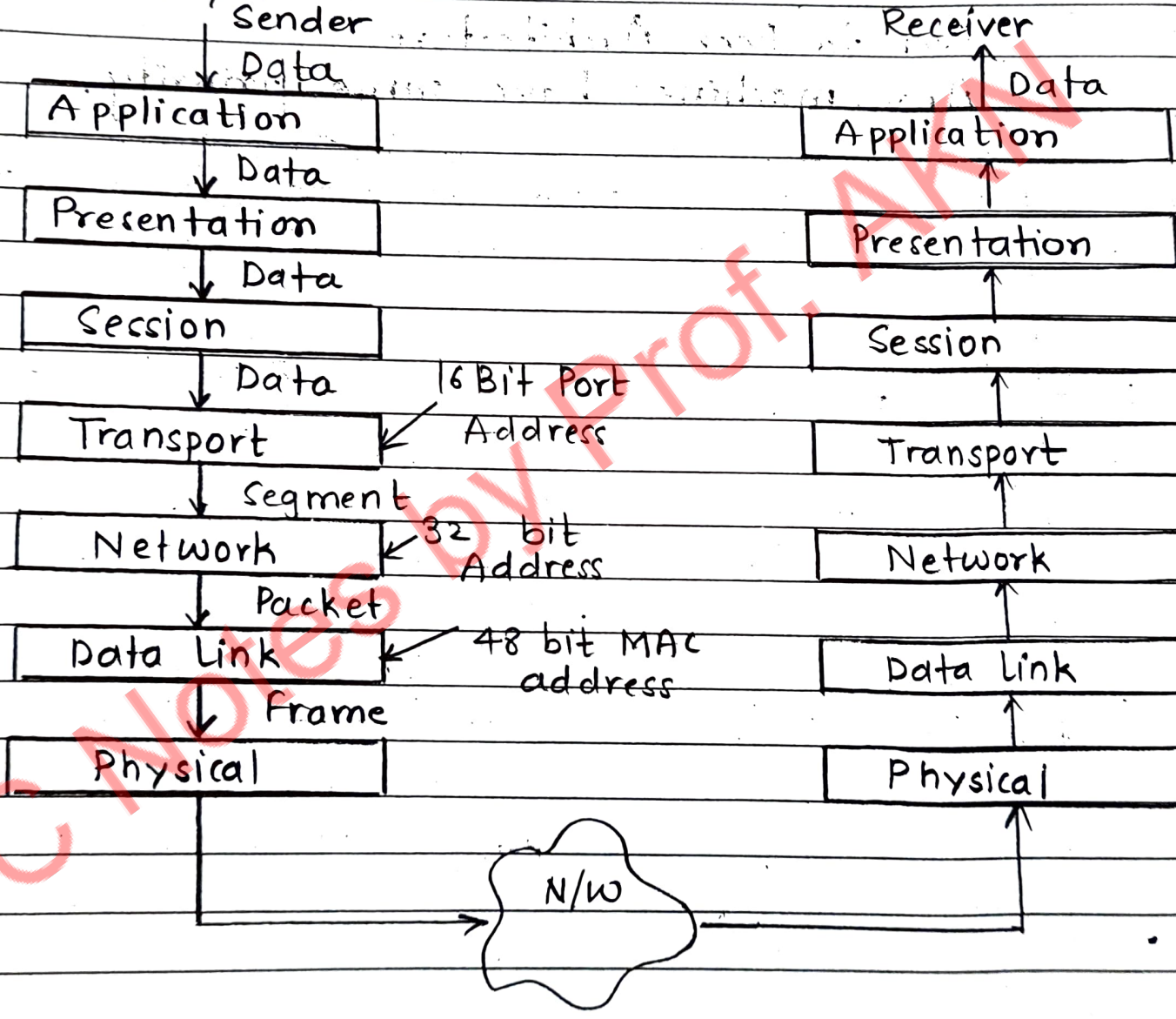
Camlin | Page
Date / /

Open System Interconnection :- (OSI)

It is a 7 layer architecture defined by ISO (International Standard Organisat^{n}) defines how exactly the n/w works i.e. functionality of each layer.



Date / /

Application Layer :-

This layer is responsible for interacting with the user

Eg: Imagine a web browser (http) to access the web page.

Presentation Layer :-

This layer is responsible for taking data from application layer and perform following 3 things

(1) Converting data from user understandable form to system understandable form

(2) Compression if required

(3) Encryption for security.

Session Layer :-

This layer is responsible for dialouge control and token management to establish connectⁿ with server.

Transport layer :-

This layer performs process to process communicatⁿ for which it uses 16 bit port address

It also performs :-

(1) Error control

(2) Flow control

(3) Congestion control

Network layer :-

This layer performs host to host communicatⁿ for which it per uses 32 bit IP address

It also performs fragmentatⁿ if required.

Camlin | Page
Date / /

Data Layer :-

This layer maps a packet and converts it into a frame, this is known as framing. It also performs error control and has a sub layer known as MAC (Medium Access Control) that takes care of collision avoidance and prevention technique.

Physical layer :-

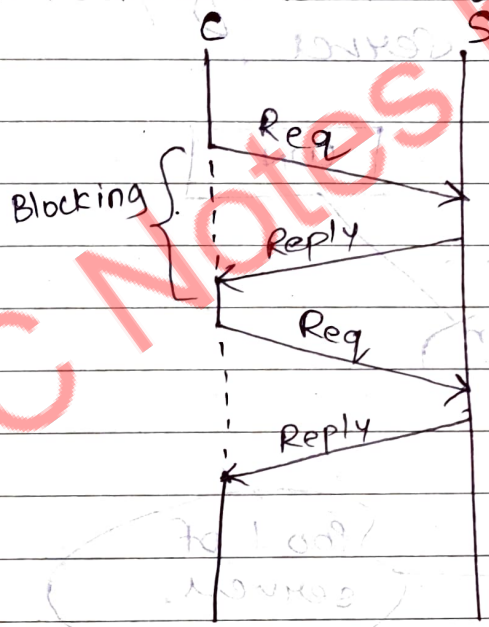
This layer performs transmission of raw bits of 0's and 1's i.e. electrical s/g to receiver's physical layer through network.

⊛ Message oriented communication. [Types of communication]

1. Synchronous (Blocking)
2. Asynchronous (Non blocking).
3. Persistent.
4. Transient.

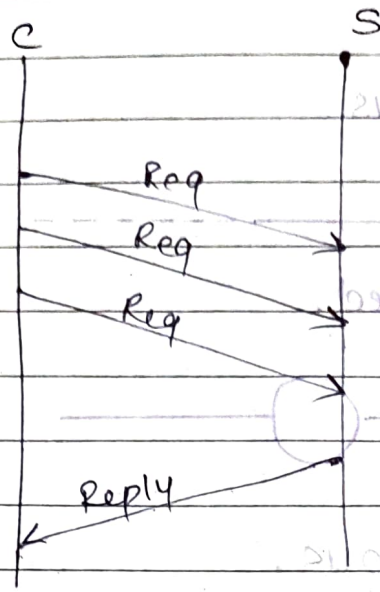
1. Synchronous / blocking.

The process where the request is sent and then blocked until reply is received is known as synchronous/blocking.



2. Asynchronous (Non-blocking.

The process where the request is continuously sent without getting blocked the reply is received is known as asynchronous/non-blocking



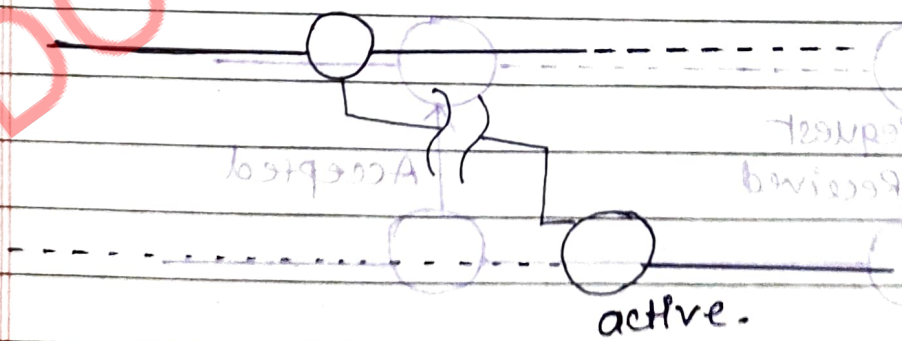
3. Persistent :

The fact where the client and server both may or may not be online.

4. Transient.

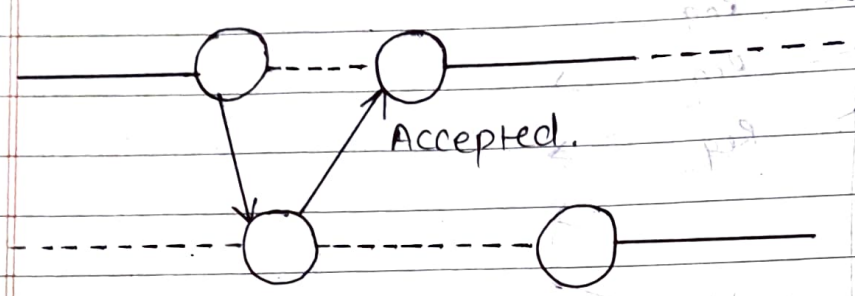
The fact where the client and server both has to be online.

(*) POSSIBLE COMBINATION OF SYNCHRONOUS, ASYNCHRONOUS, PERSISTENT, AND TRANSIENT.

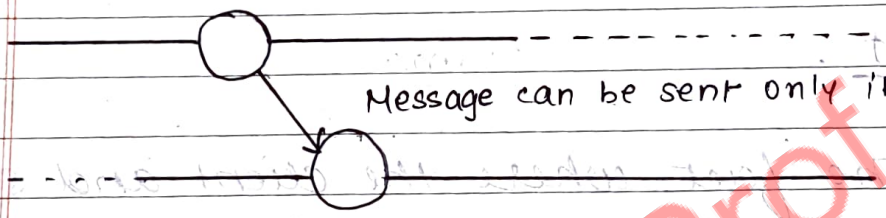


i) Persistent Asynchronous.

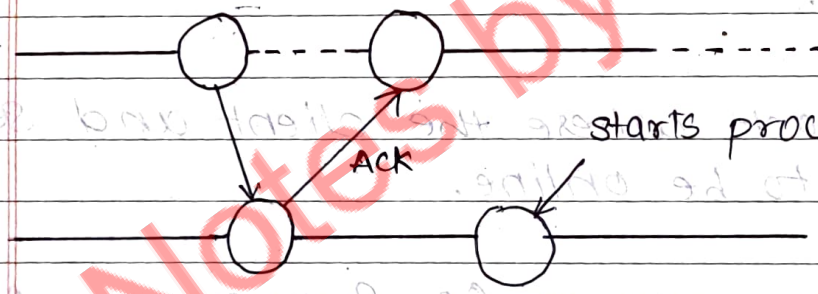
ii) Persistent Synchronous.



iii) Transient Asynchronous.

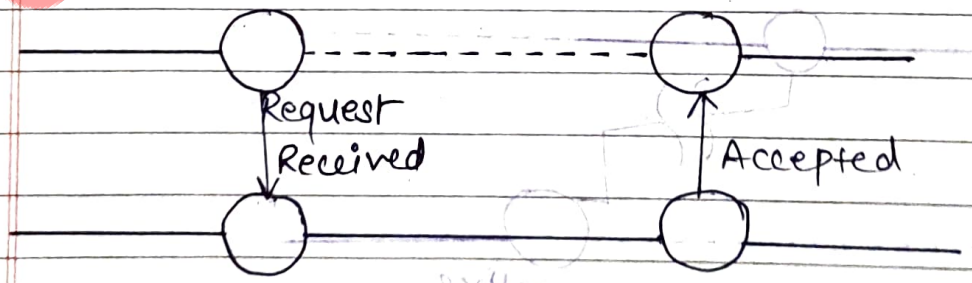


iv) Transient Synchronous.



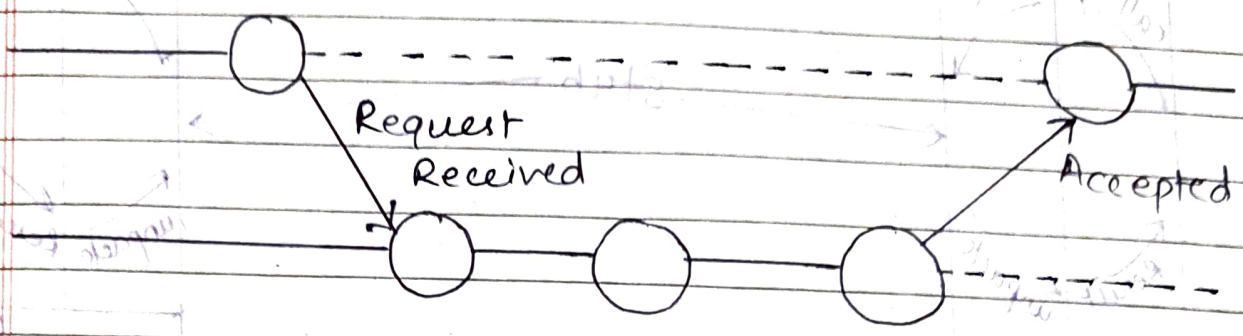
(Receipt Based)

v)



Transient Synchronous (Delivery Based).

vi) Transient Synchronous



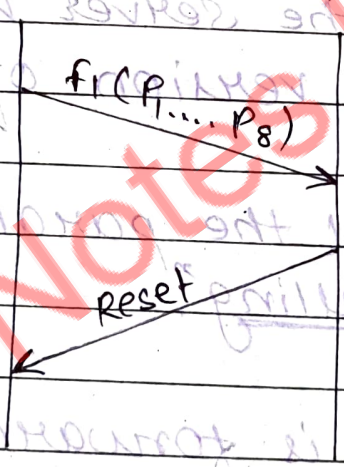
(Response Based)

Imp

* Remote Procedure Column. (RPC) [concept]

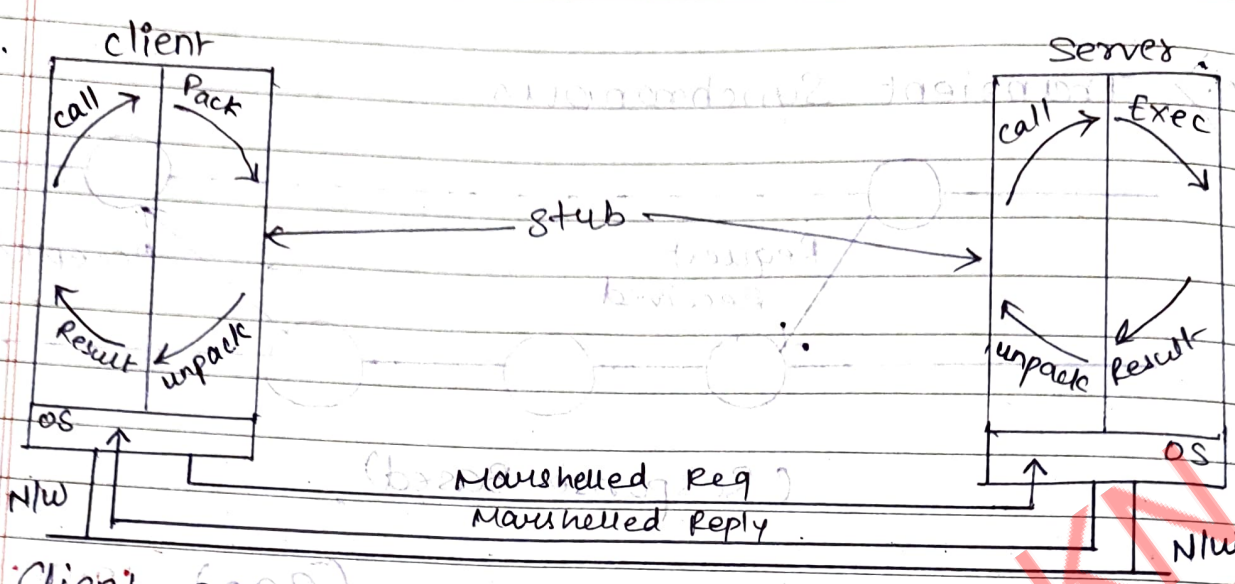
The process of calling a function which is not on the same machine, but present on some other machine in the network, this is known as remote procedure call.

Eg:



P1.... P8 : Parametric
fi () : function.

(*) PROCESS



1. Client process makes the call and realises that the function to be executed is not present on client, and hence this request is handed over to ~~server~~ an interface known as "stub".

2. Client stub locates the server having required function and same version of stub.

3. Client stub packs all the parameters, this is known as "Marshelling".

4. Marshelled request is forwarded to server. Server stub unpacks this request, which is known as unmarshelling.

5. Server stub calls the required function, server process executes the function and returns the function to server stub.

7. Server stub packs the result and gives marshelled reply to client.

8. Client stub unpacks the packet, and returns the result to the calling process.

Transparancies to be maintained by RPC.

1. Location.
2. Migration (w.r.t program).
3. Relocation (w.r.t function).

Parameter passing semantics.

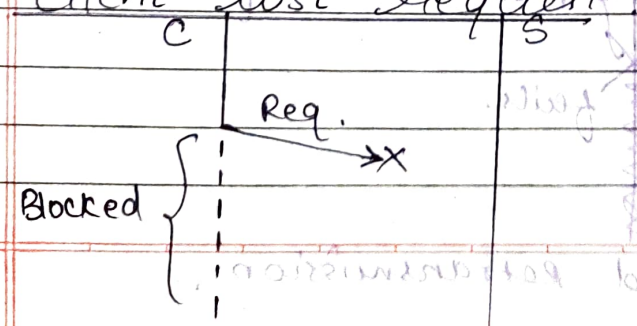
1. Call by value.
2. Call by Reference.

Imp. ⊛ RPC failures (Process resilience).

i) Client cannot locate the server.

→ ~~Client~~ function is'nt available.
→ Server ~~the~~ crashes.
→ Stub mismatch.
→ client nw failure.
→ server nw failure.

ii) Client lost request.



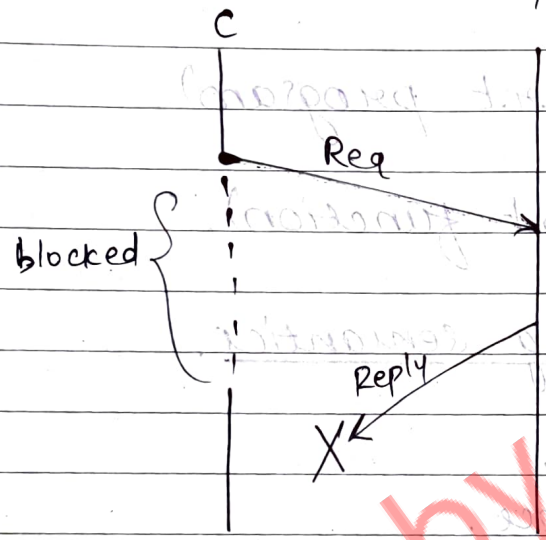
The request is being sent to server by client but the request is lost

Page No.
Date

Soln : Timeout and retransmission.

3.) Server lost reply.

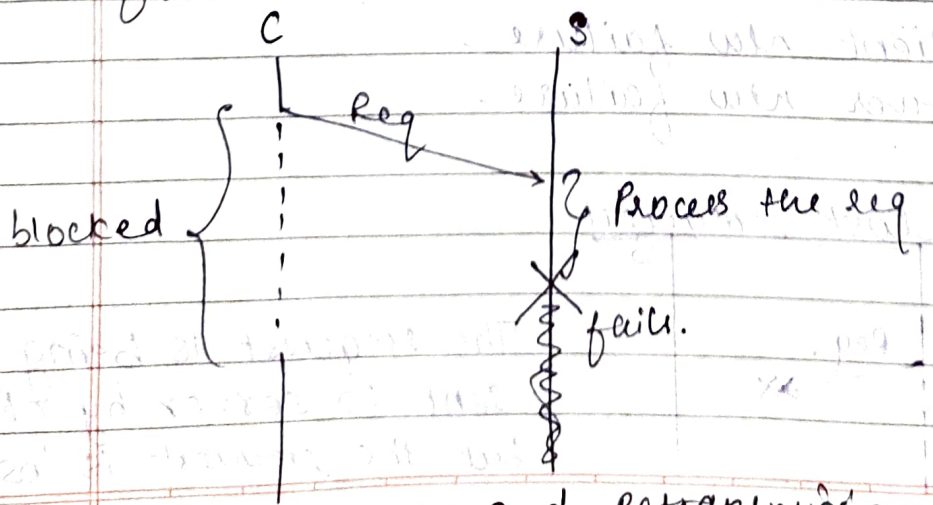
Client sends the request and gets blocked, server sends the reply, but it gets lost.



Soln : Timeout and Retransmission.

4.) Server fails

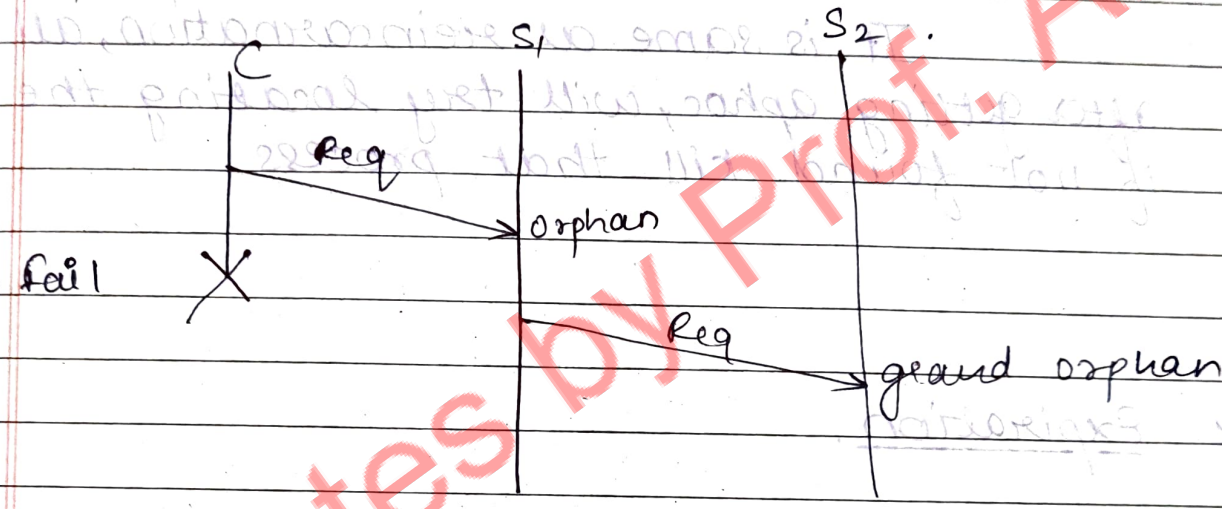
Client sends the function, server starts executing and ~~in middle of~~ during execution it fails.



Soln : Timeout and Retransmission.

5) Soln Client failure.

Client sends the request, and crashes server processes the request, ~~it~~ ~~their~~ and hence that function becomes orphan as there is no existence of calling parent. and if ~~their~~ ~~fun~~ occurs a situation where their is function call within function then upcoming all functions would be grand orphans.



Soln :

1. Ex-termination. } State for IA }

Client will enter details of server (S_1) in log, when request is made. If client fails and reboot, it will check the log and inform that server (S_1) to kill all previous transaction of client, this will kill only orphans.

2. Reincarnation

When client after failure reboots, it will broadcast epoch (time) and inform all by broadcasting epoch to kill all processes which has started before epoch, even though it will kill orphans & grand orphans it will kill all processes.

3. Gentle Reincarnation.

It is same as reincarnation, all servers after getting aphoc, will try locating the owner, if not found kill that process.

4. Expiration.

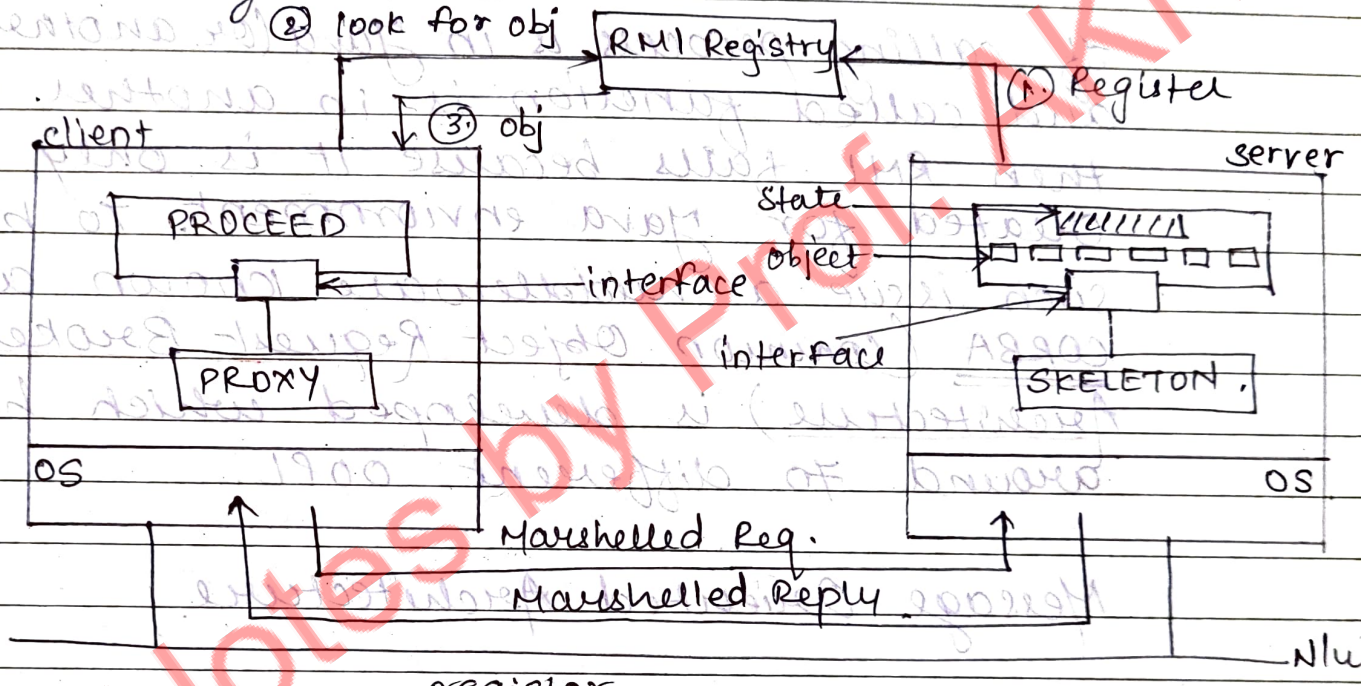
Server keeps a timeout it waits for a client request again during that time period. If client does not request then server checks if client alive. If it is then compute the execution and sends back result.

Imp

(*) Remote Method Invocation (RMI) / Remote Object Invocation. [implementation of RPC in Java].

RMI is implementation of RPC in Java. i.e. both calling and called program should be in Java.

(*) Block Diagram.



Server initially ~~registor~~ register, interface and object on RMI registory.

Client process makes the call and realises that the required function is not on the same machine and hence forwards this request to proxy. Proxy which is client side stub, look up for server and object in RMI registory, gets the required object, implements the same interface as that of server.

Proxy now packs the parameters (marshelling) and sends this marshelled request to server.

Server skeleton (server stub) unpacks the request (unmarshelling) and with required object calls the function. Server process executes and returns the result to skeleton, which packs this and gives it to client. Client proxy unpacks the result and handover the result to calling process.

If calling process is in Java (or another OOPL) and called function is in another OOPL then RMI fails because it is only created for Java enviornment. To handle such issue a middleware known as CORBA (Common Object Request Broker Architecture) is developed which handles around 70 different OOPL.

Message Oriented Architecture.

* Types of Communication.
(Same as message oriented communication)

* Message Broker Architecture (Refer notes)
* Message queuing architecture (Refer notes).
* Stream oriented communication. (Refer notes)
* Group communication. (Refer notes)