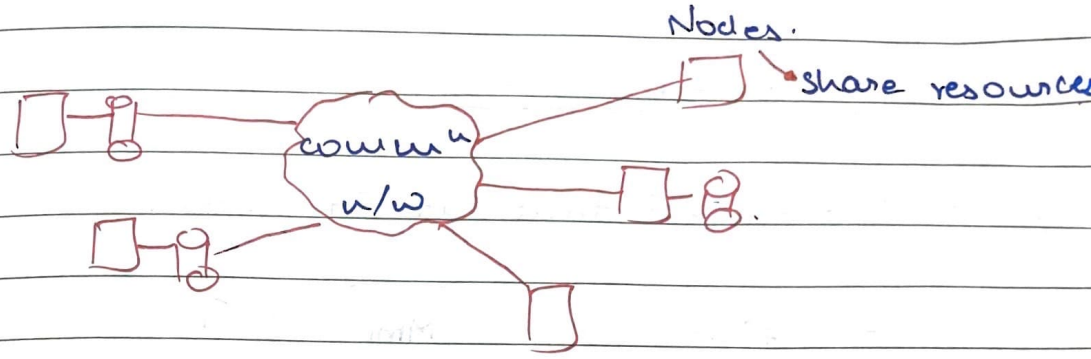
Distributed Systems

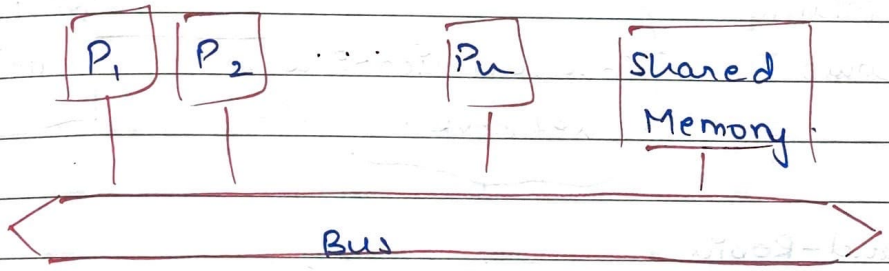
→ It is a collection of independent computers that appears to the users of the system as a single coherent system.

Eg.



Types of architecture.

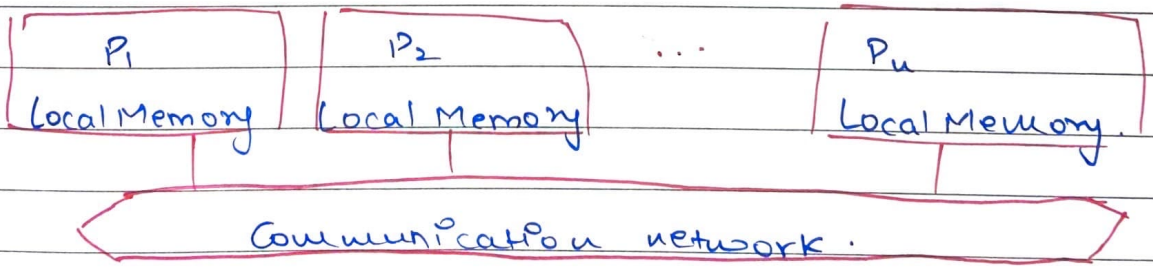
1) Tightly coupled architecture → Shared memory.



$x = p_0$

→ Not scalable due to common bus.

2) Loosely couple architecture → Shared Nothing.



eg. $x = 100$

PAGE No.
DATE / /

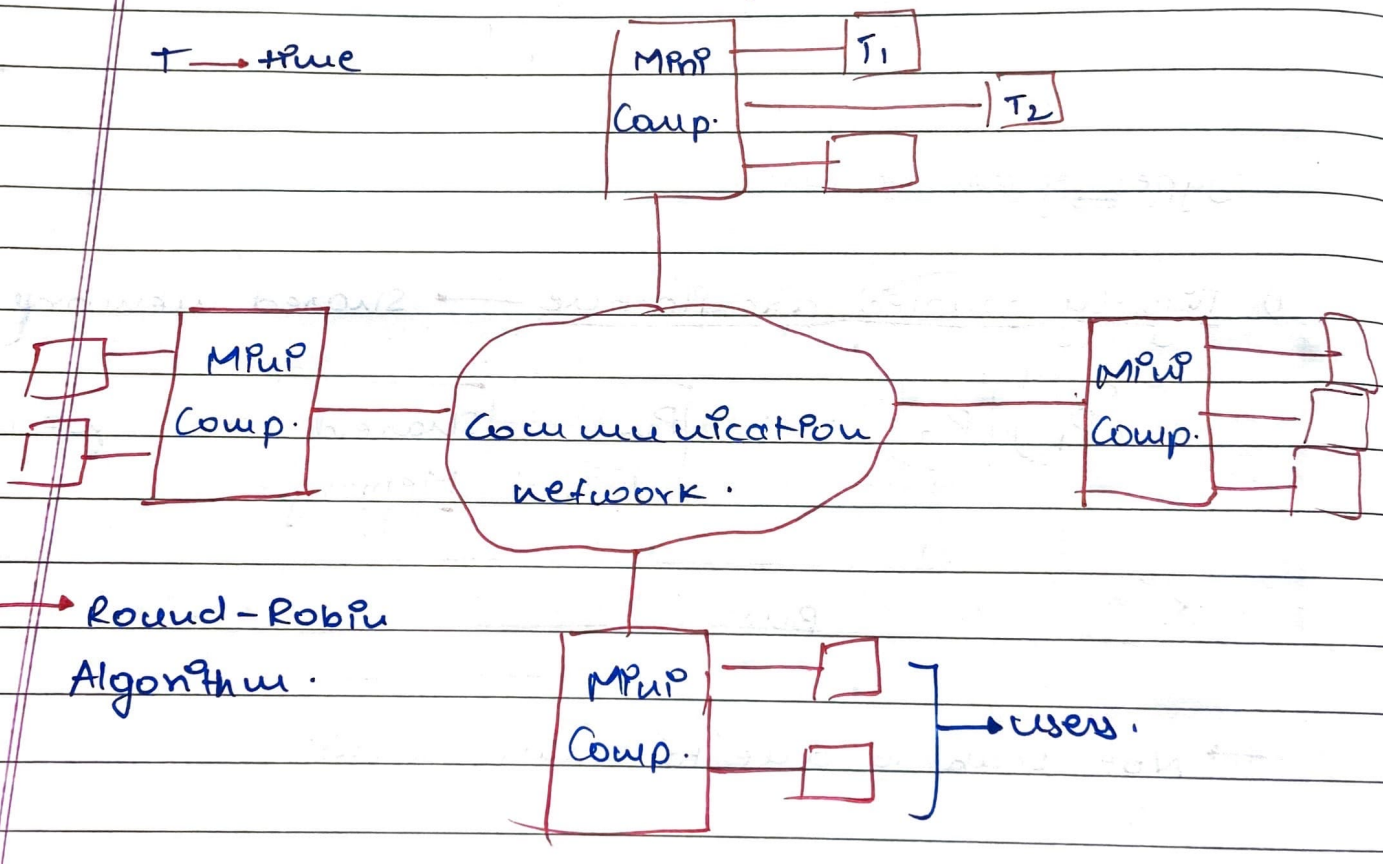
Distributed Computing System models

1) Minicomputer model
2) Workstation model
3) Workstation server model
4) Processor model.
5) Hybrid model.

Minicomputer model

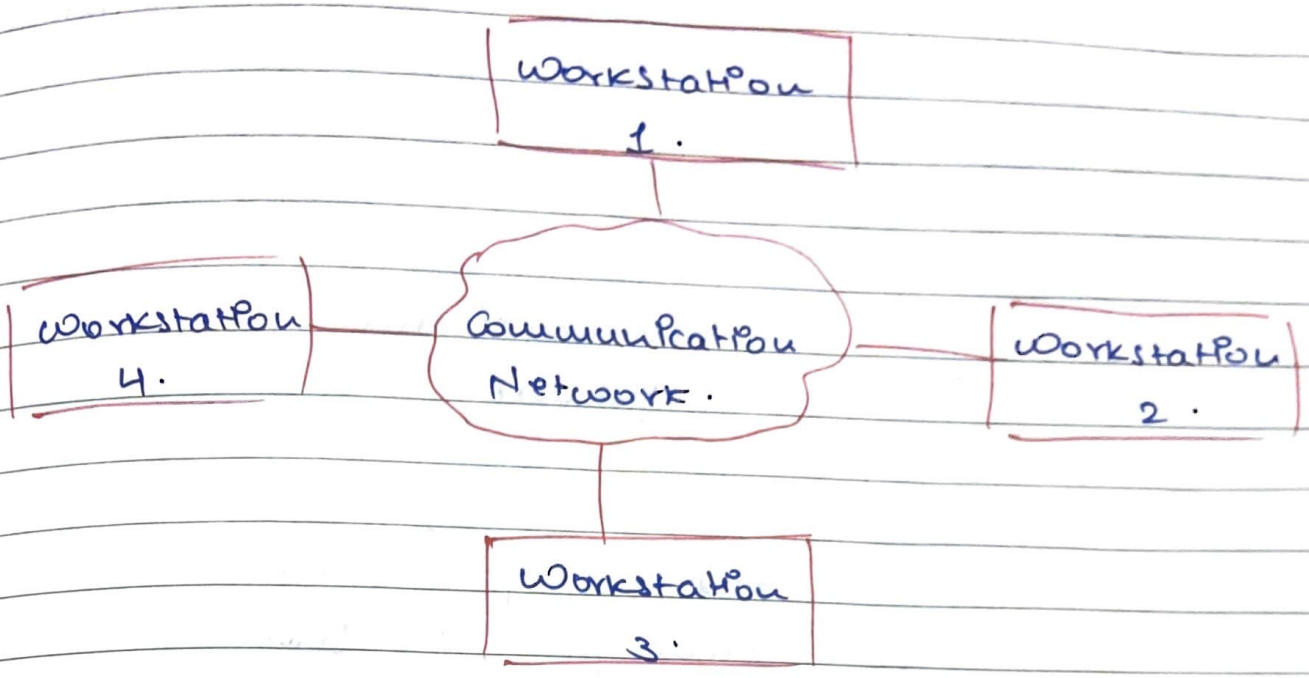
→ Centralized time sharing system.

T → time

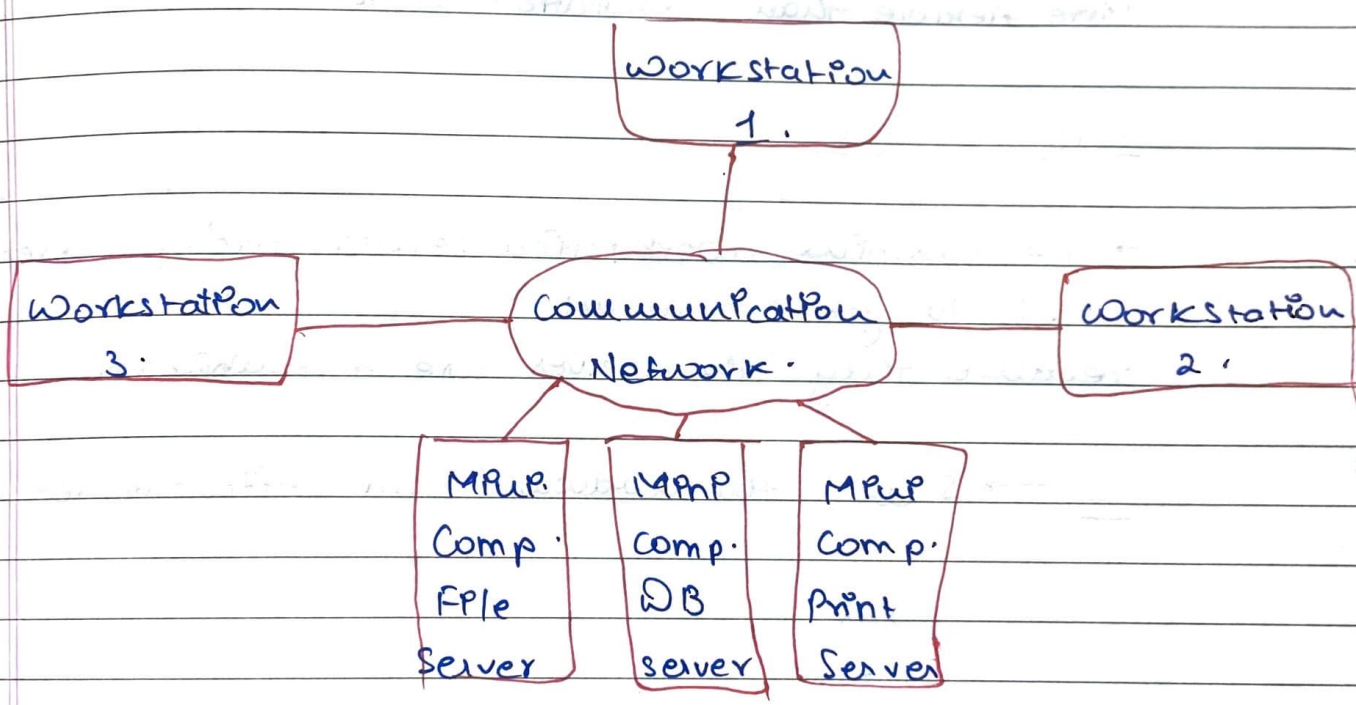


→ Round-Robin Algorithm.

Workstation Model



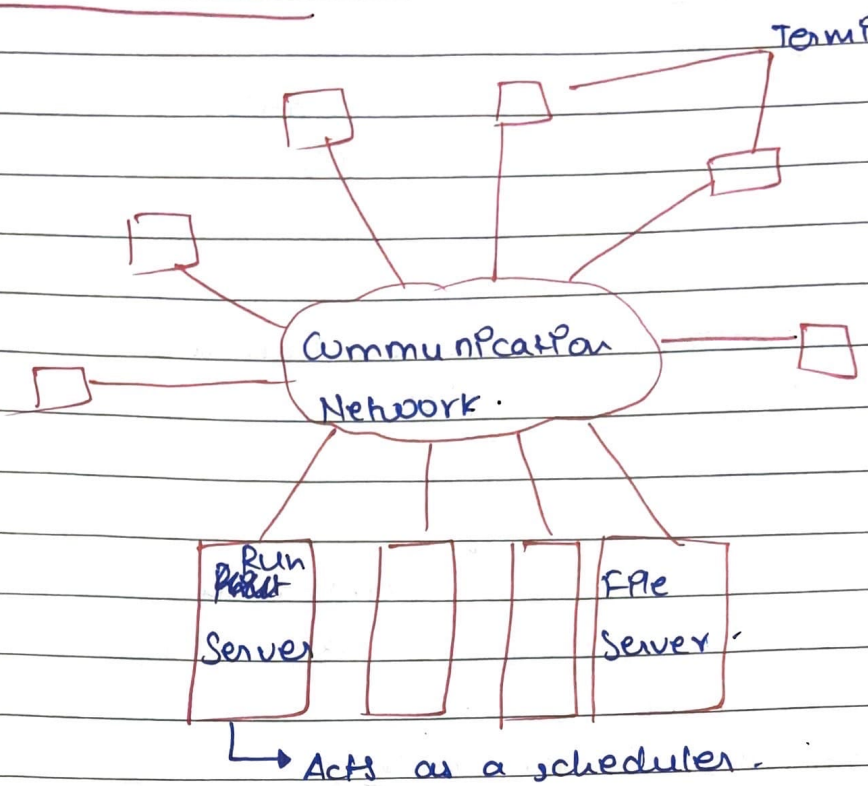
Workstation Server model



- Simple planned server model.
- Works as a client server model.
- Easy to manage resources

PAGE No.
DATE / /

Processor Pool model



* More flexible than workstation model.

Hybrid model.

→ We combine workstation server and processor pool models to get HPS model

→ Because they themselves are a combination.

We → Sharing the hardware and software resources

Advantages of DC

1) Inherently distributed applications → locally as well as globally the data is distributed.

2) Information sharing among distributed users

3) Resource sharing → Eg. Printer.

4) Better price performance ratio.

5) Shorter response time and higher throughput.
→ In workstation model, ideal resources can be shared.
→ Can distribute the tasks to lightly loaded machine according to requirements.

6) Higher reliability and availability.
↳ because of replication

7) Extensibility & Incremental growth.

8) Better flexibility.

Disadvantages of DC

1) Higher maintainence if the systems are not high-end

2) Security breach

PAGE No.
DATE / /

Issues in Designing the distributed Systems

1) Transperancy → Hiding underlying details of the system.
→ Why? → It appears as a single system.

a) Access → Hide details of data coming from various nodes

b) Location → Hide details of location
- Naming → Name given to resource is not known
- Migration → Resources moving from one location to another is not known.

c) Relocation → Hide details of why the resources are relocated from one server to another.

d) Concurrency → Hiding the no. of users using the servers.

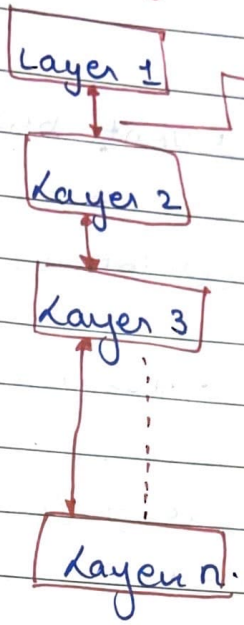
e) Failure → Hide failure & recovery of resources

f) Persistante → Hide the software resource in memory or disk. (Permanent).

Q. Identify and explain transperancy for the given scenario.

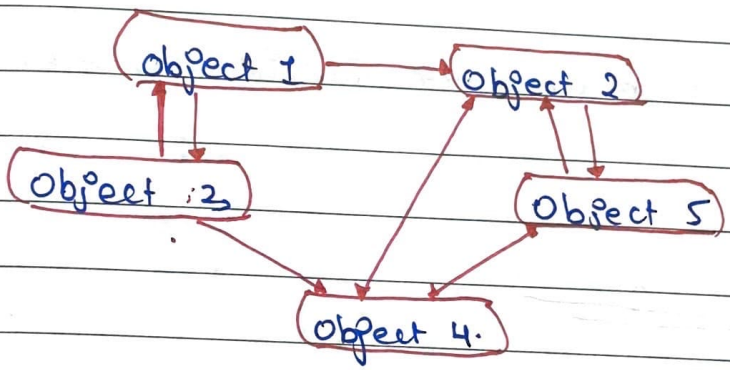
Architecture Styles

1) Layered Architecture → Eg. OSI model.



Disadvantage:
Each layer can communic
with the next layer only

2) Object-based style.



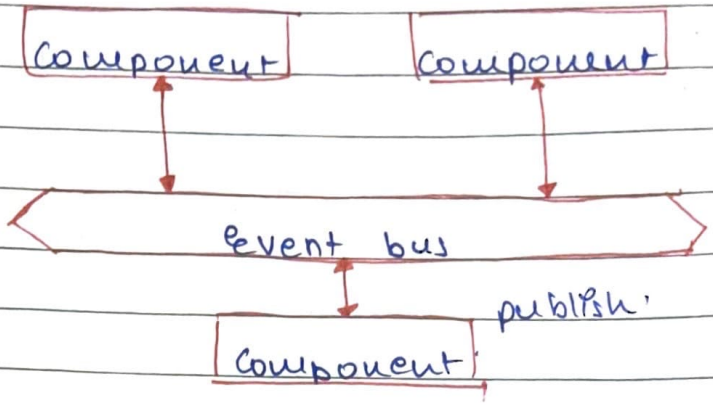
- No restrictions.

Eg. Client server system.

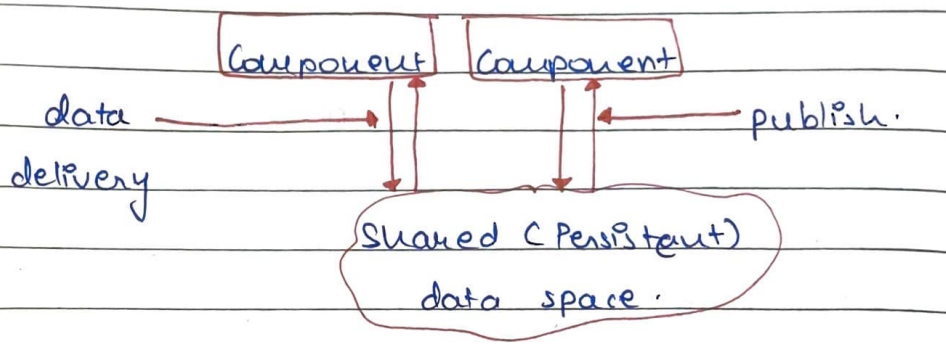
3) ~~Event~~ based.

→ can be used at a particular time only.

3) Event based



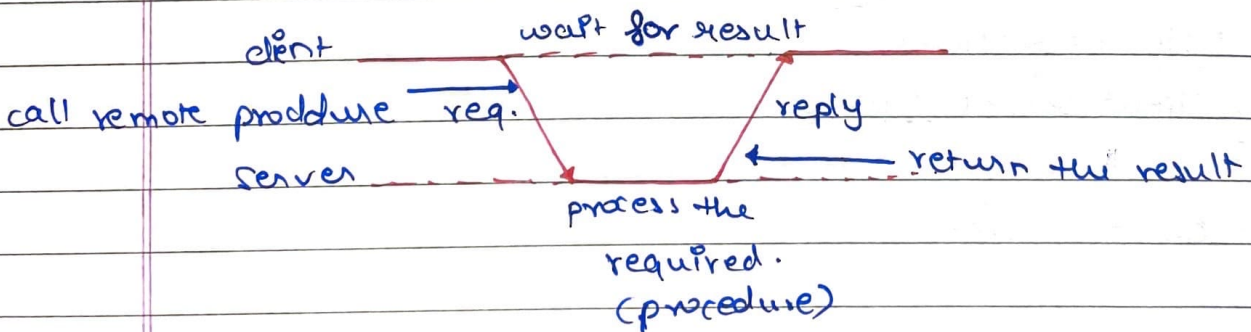
4) Shared data space → Anybody can use anytime.



Bulletin board / Blackboard architecture → another name.

Eg. Blackboard.

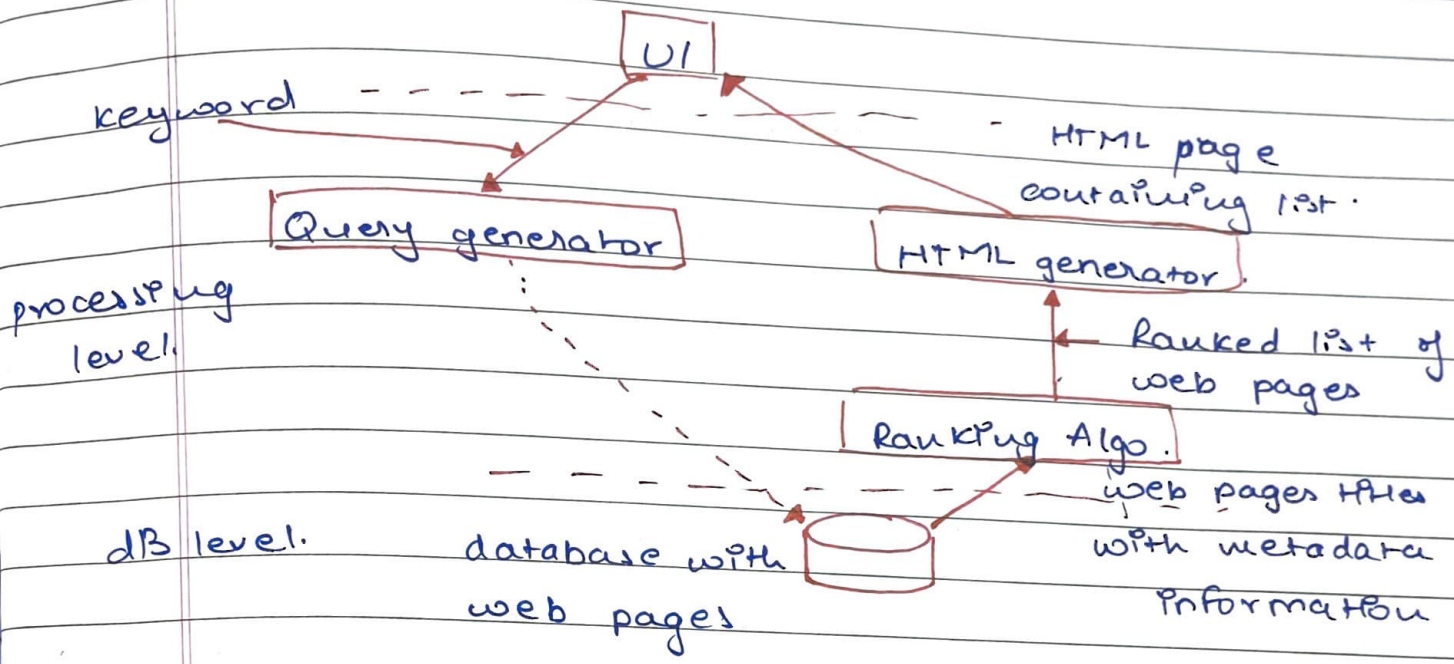
Client Server Architecture

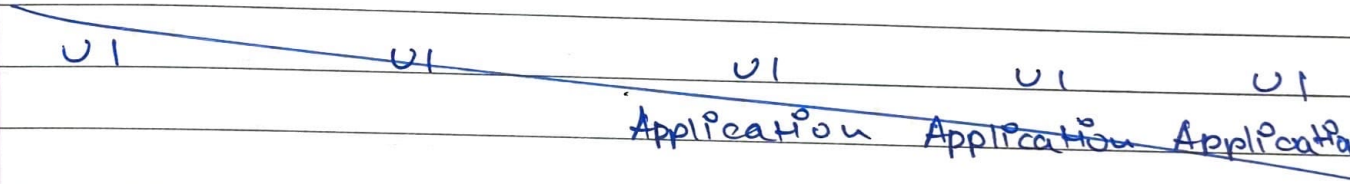


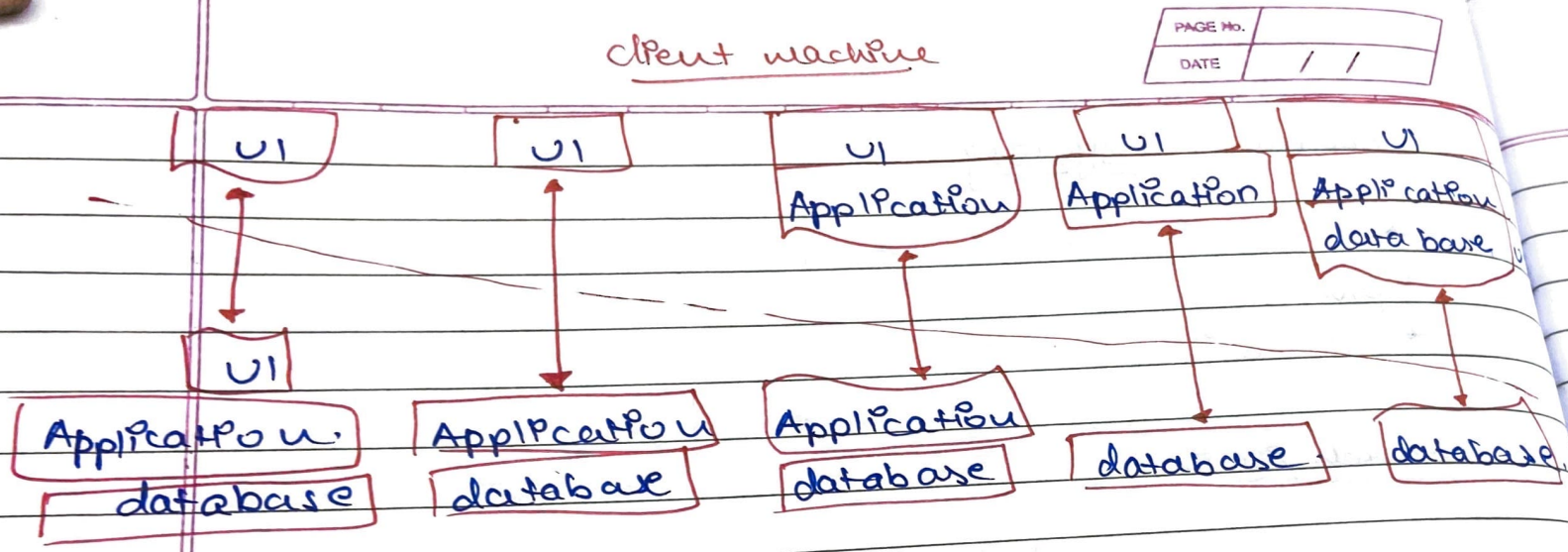
Application Layering

- UI
- processing
- database.

eg. Search Engine







Remote Procedure

→ So it looks like

coherent system

main()

procedure

call

→ when

machi

& exe

Infor

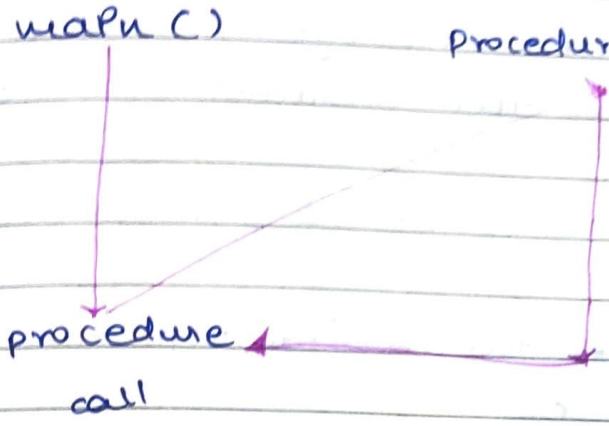
pr

res

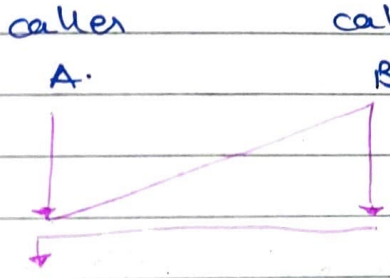
prog

Remote Procedure Call (RPC)

- So it looks like centralized computing (single coherent system).



- When process on machine A calls a procedure on machine B, the calling process on A is suspended, & execution of called procedure takes place on B. Information can be transported from caller to callee in the parameters & can come back in the procedure result. No message passing is visible to the programmer. This is known as RPC.

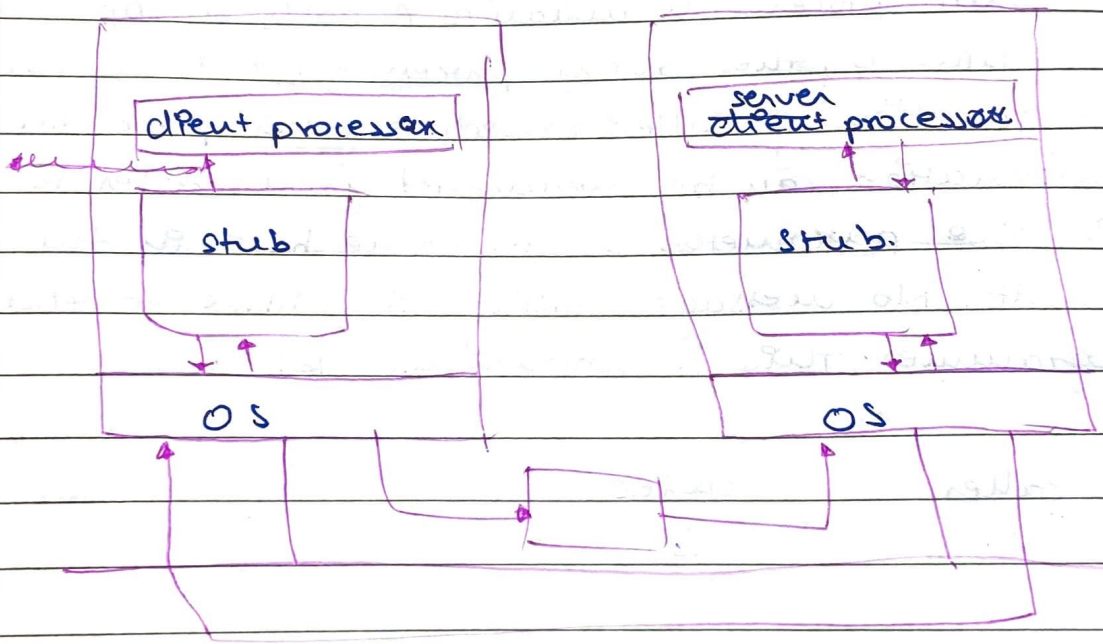


Stubs → creation of packets.
Runtime RPC → Local compiler.
parameter marshalling → packing the parameters.

PAGE No.
DATE / /

Steps of RPC

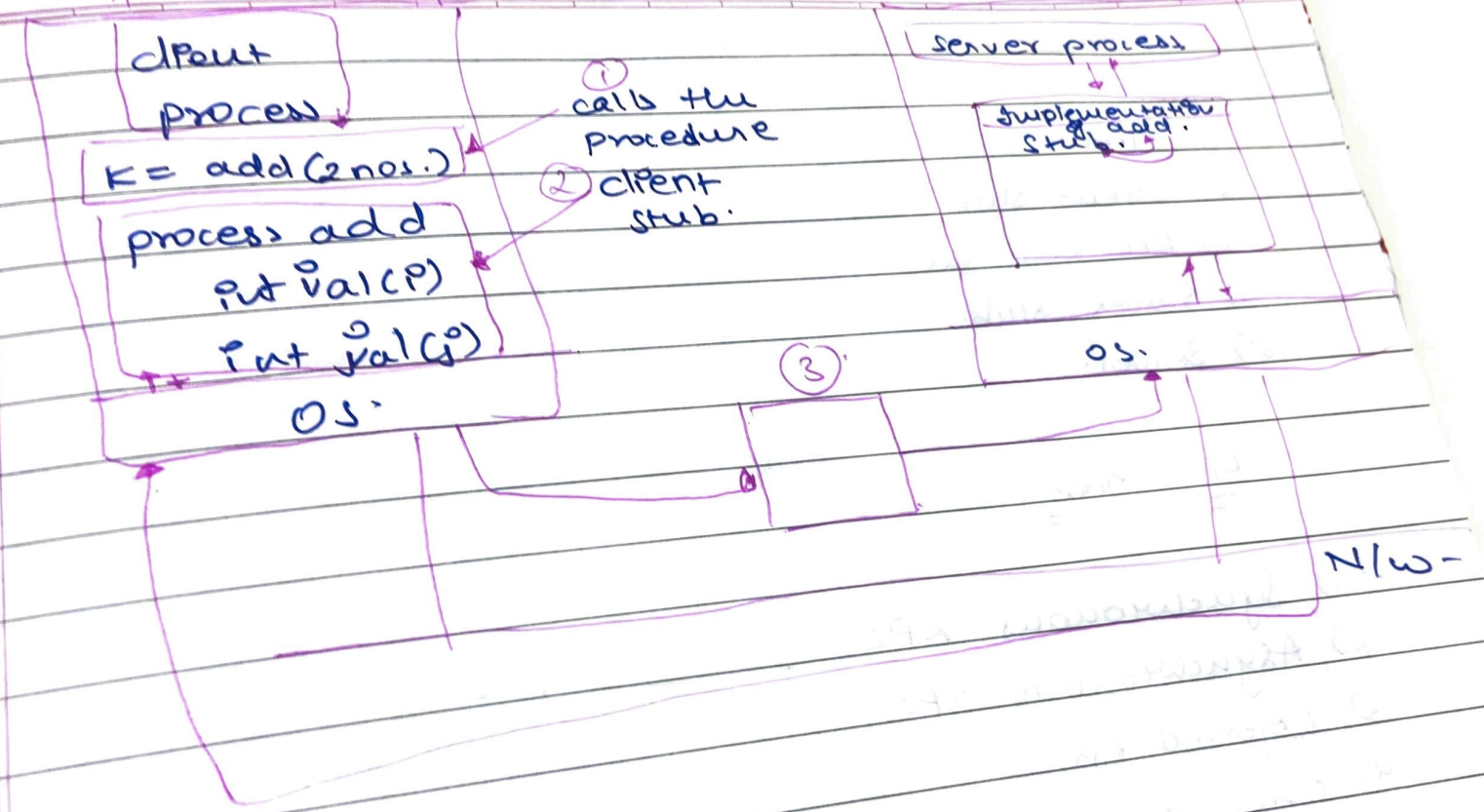
1) Client procedure calls the client stub.
2) The client stub builds the message and calls the local OS.
3) The client OS sends a message to remote OS.
4) The remote OS gives message to server stub.
5) Server stub unpacks the parameter & calls the server.
6) The server does the work & returns the result to the stub.
7) The server stub packs the ~~message~~ result in the message & calls the local OS.
8) Server OS sends the message to client OS.



9) Client OS. gives the message to the client stub.
10) Stub unpacks the result & returns to the client.

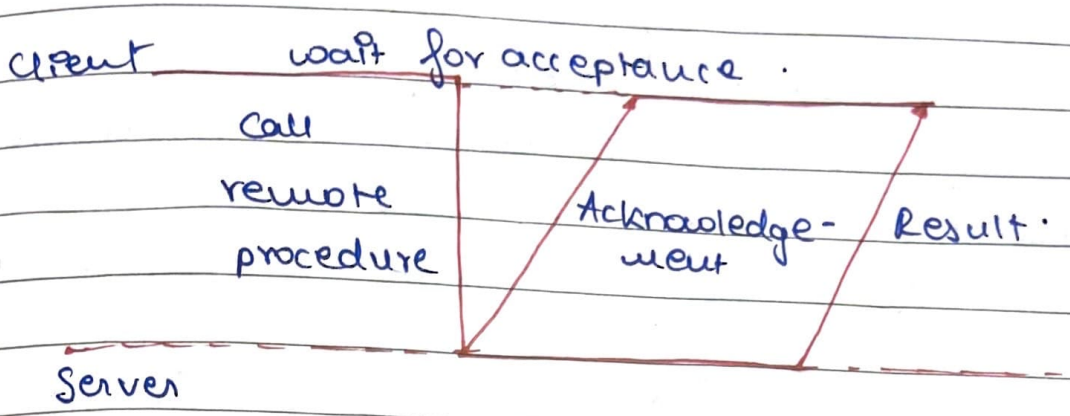
Q. Add 2 numbers.

H.W. add steps.



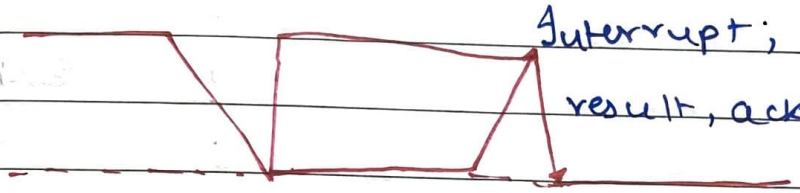
PAGE No.
DATE / /

Asynchronous RPC → waiting for client



Deferred RPC → no wait time

→ Combining two asynchronous RPCs.



Eg. OTP verification.

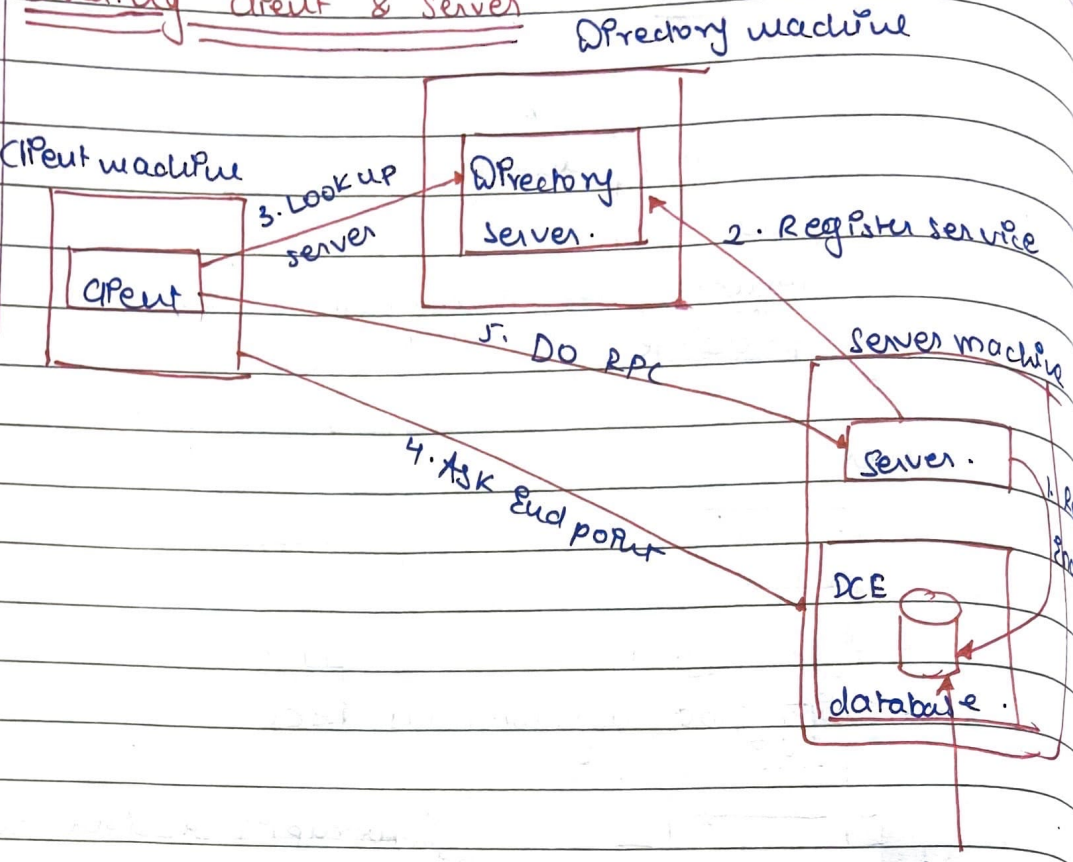
One-way RPC

→ Server does not wait for acknowledgement.

* Port mapping services → to map port to the server.

Q.

Binding client & server



Failure semantics

1) Server failure semantics

→ At least once semantic → If the client received the reply from the server, it means that all has been executed at least once on the server.

→ At most once → If the client gets the reply from the server it means that the RPC call have been executed at most once on the server.

→ Exactly once → Same exactly once.

Eg. Gmail.

PAGE No.
DATE / /

Client failure semantics

→ If server executed RPC call, but meanwhile client crashed, server will discard the response

→ If server was executing long running RPC & in both executions it comes to know that client has crashed then such calls are called Orphaned RPC calls

→ In this, server can do the following:-

1) Extermination → when client gets crashed, server can eliminate the process.
2) Incarnation. → when the server broadcast msgs to know if the client is alive.

Light-weight RPC

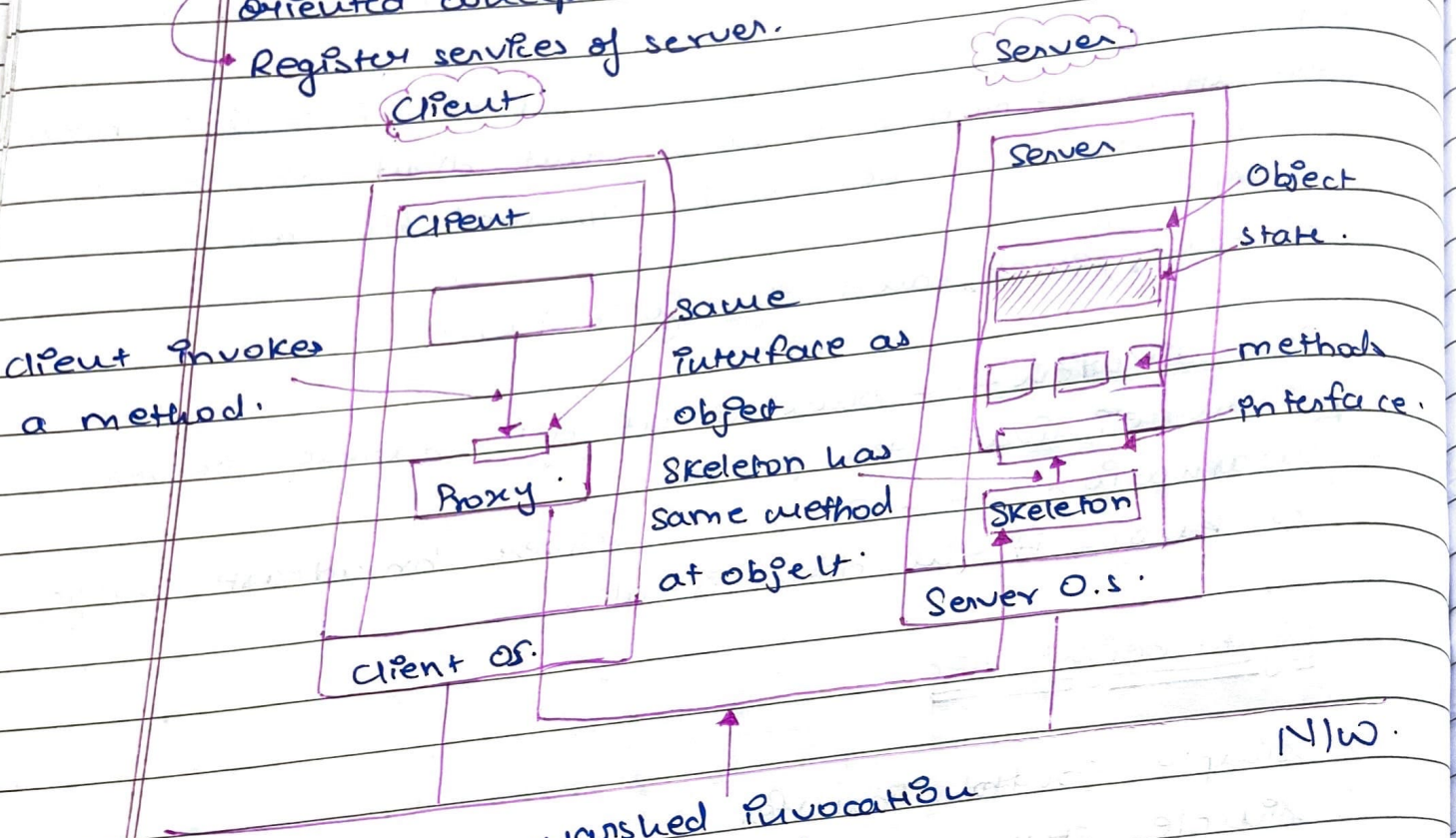
1) Simple control Transfer
2) Simple stub.
3) Simple data transfer
4) Design and concurrency

How to make RPC flexible?
→ Layered approach → by making it reliable → replication.
Problems in RPC → accessing global variables

Remote Object Invocation

- Object oriented concept.
- achieves distribution transparency
  1) Location 2) naming.

(*) RPC → procedure oriented concept
Register services of server.



(*) Skeleton → marshalling
Proxy → unmarshalling.

(*) Persistant object → permanent → requires secondary storage.
Transient object → exists during runtime only → temporary.

(*) How to get the server up-to-date → clock-synchronisation

(*) Binding of object :- 1) Implicit.
2) Explicit

PAGE No.
DATE / /

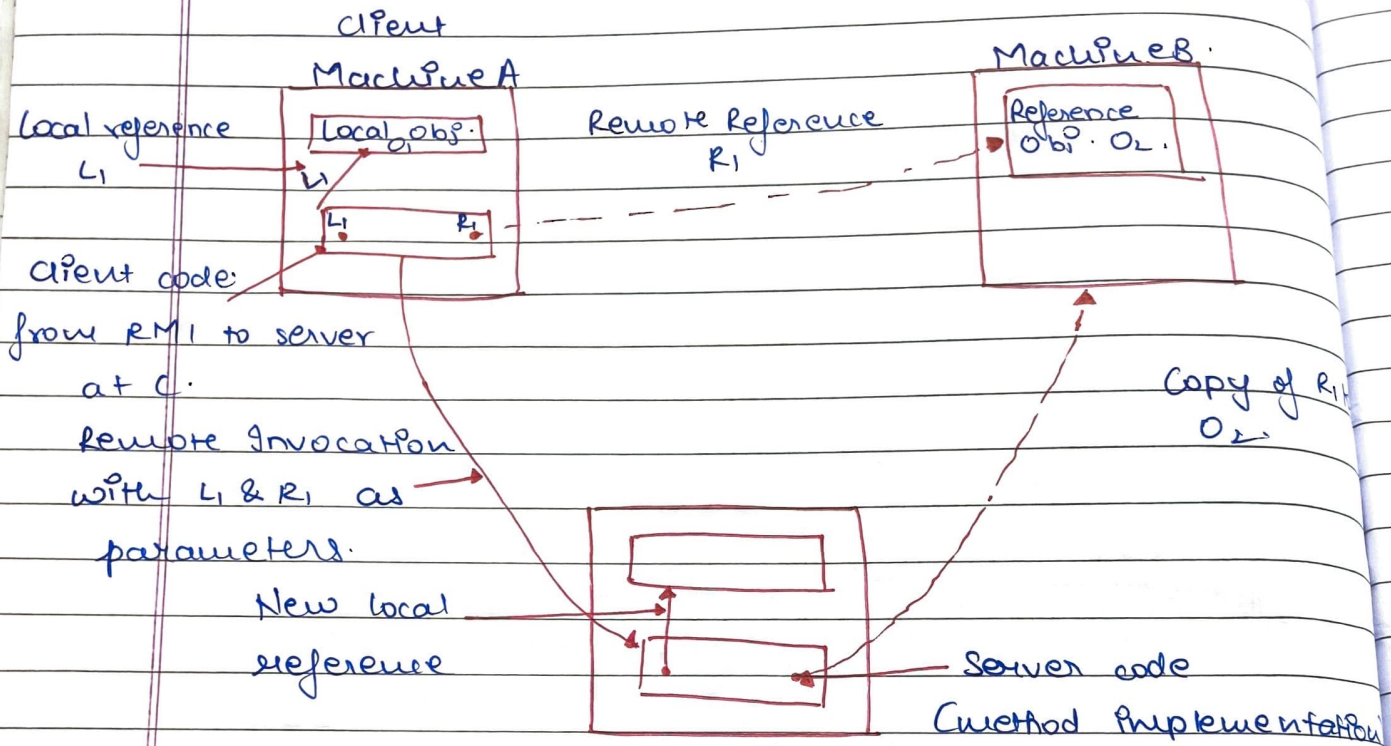
(*) DCE → Distributed Computing Environment → Locating the reference object.

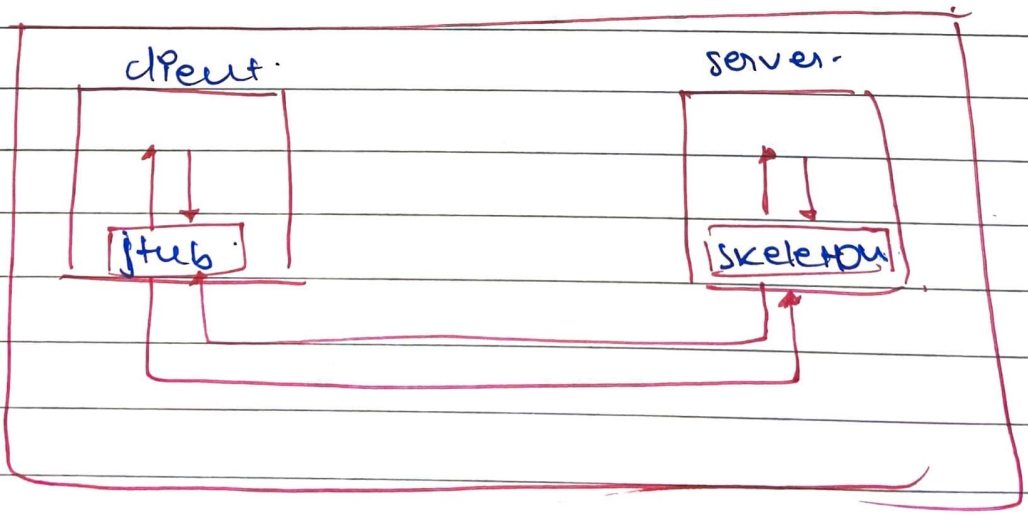
→ Java implementation of RPC.
→ objects in different programs can communicate.

RMI (Remote Method Invocation)

Parameter passing

→ System-wide object reference.





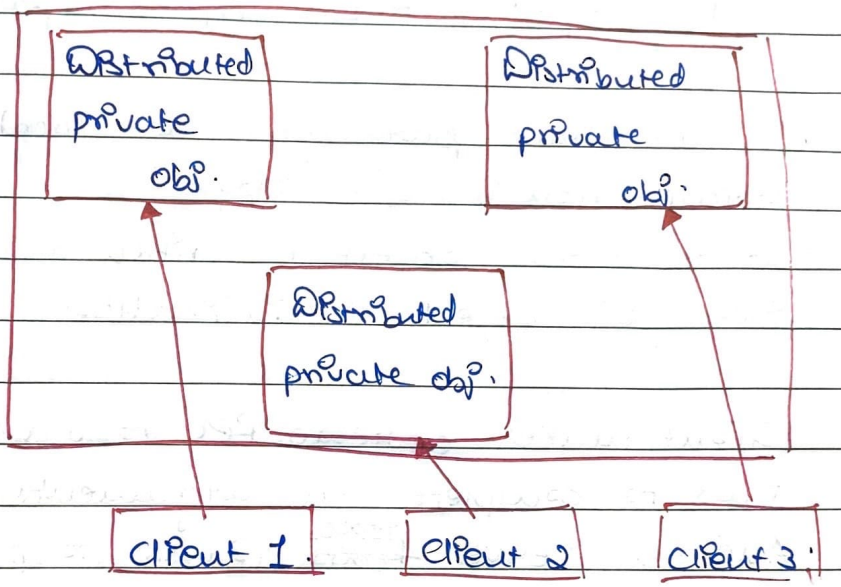
Independent.

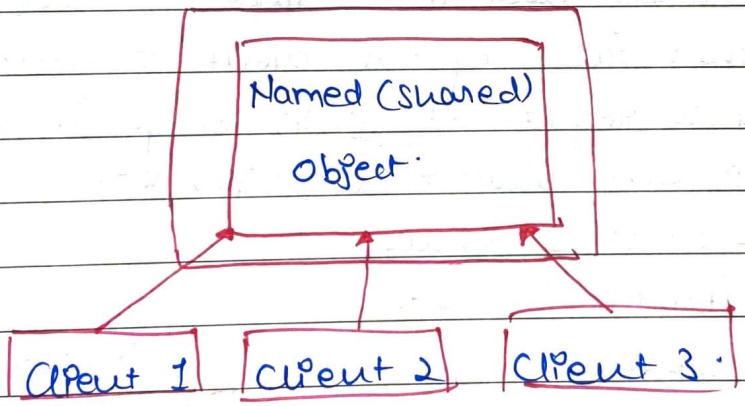
PAGE No.
DATE / /

The WCF Distributed Object Model

→ Distributed & dynamic object → server creates on behalf client

→ Distributed named object.





Method.

Remote Invocation

RPC vs RMI

→ RPC → procedural programming.
RMI → object based.

→ RPC → parameters are ordinary data structures.
RMI → possible to pass objects as parameters

- If marshalled parameters are local → object serialization is used. (Copy).
- State of the object is written to a byte stream through object serialization.

Q. Client makes a ~~remote~~ RPC to a server. Client takes 5 ms to complete the argument for each request. And the server takes 10 ms to process each request. Local OS's processing time for each send or receive operation is 0.5 ms and the network time to transmit each msg or reply each msg is 3 ms. Marshalling and ~~un-~~ unmarshalling takes 0.5 ms per msg. Calculate the time taken by the client to generate a return from 2 request if it is single threaded. You can ignore the context switching time.

→ Request → 5 ms.
M UnM → 0.5
client → 0.5
Network → 3 → 0.5 (skeleton)
0.5 (server)
(u)
server OS → 10 → 0.5 (u)
→ 3 0.5 (u)
→ 0.5
→ 0.5

Steps Steps

1) Create the
2) Provide th
3) Compile
& skeleto
4) Start th
5) Create
6) Create

1)

2)

PAGE No
DATE / /

Steps for writing the RMI program.

1) Create the remote interface.
2) Provide the implementation of remote interface.
3) Compile the implementation class & create the proxy & skeleton objects using rmic tools.
4) Start the registry services by rmiregistry tools.
5) Create & start the remote application.
6) Create & start the client application.

1)

```
import java.rmi.*;
public interface Adder extends Remote
{
    public int add (int x, int y).
    throws RemoteException;
3.
```

2)

```
import java.rmi.*
import java.rmi.server;
public class AdderRemote
extends UnicastRemoteObject
implements Adder{
AdderRemote () throws
 RemoteException {.
      super ();
 3.
 public int add (int x, int y).
 {  {return (x+y);}.
  }.
```

3) rmic AdderRemote

4) rmiregistry 5000

```
import java.rmi.*;
import java.rmi.registry.*;
public class Myserver {
  public static void main(String args[]){
    try{
      Adder stub = new AdderRemote();
      Naming.rebind("rmi://localhost:
                    5000/abc", stub);
    }
    catch(Exception e){ }
  }
}
```

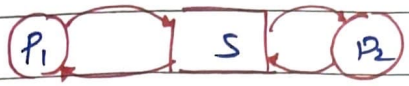
```
import java.rmi.*;
public class MyClient {
  public static void main(String args[]) {
  try{
    Adder stub = (Adder)Naming.lookup
    ("rmi://localhost:5000/abc");
    System.out.println(stub.add(3,4));
  }
  catch(Exception e){ }
}
}
```

PAGE No.
DATE / /

Message Passing

→ Interprocess communication (IPC)

→ Shared memory



→ Disadv. → Blind write.

→ Message format is same

Features of good message passing

1) Simplicity.
→ Send & recieve.

2) Uniform semantics. → Local → same machine.
→ remote → different machine

sending data and ack. packed. ← → Reliable data transfer.

3) Efficiency → Reducing no. of messages exchanged.
→ faster communication.
→ minimising cost of communication.
→ Reducing no. of acknowledgement
→ sending one for multiple.

4) Reliability → no packet loss
→ in-time delivery
→ no errors
→ no duplicity.

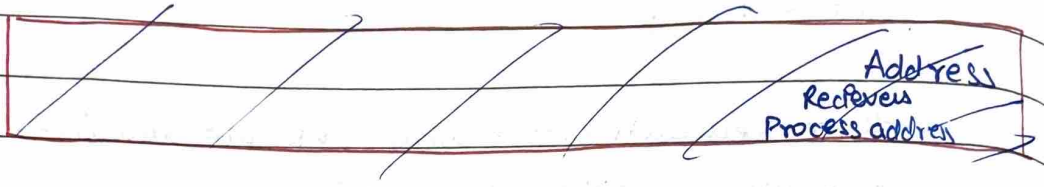
5) Correctness → can be used for group communication
→ atomicity → all or nothing.
→ ordered delivery → sequence no.
→ survivability

6) Flexibility → broadcast protocol
multicast protocol

7) Security → Encryption
↳ Authenticate user.

8) Portability → Transferrable to any system.
↳ Should be visible to all users

Issues in IPC by Message Passing



Actual data or pointer to data.	Structural Information		Sequence no. or message id	Address.	
	no. of bytes	Type		Receiver's Process Address	Sending Process Address.

← variable size → ← Fixed length header →

Synchronisation.

↳ Blocking → Synchronous
↳ Non-blocking → Asynchronous
↳ Receiver Knows? → through notification.

PAGE No.
DATE / /

Buffering

1) Null buffer → normal phone calls.
~~Advantage~~ → Sender & receiver should be active.

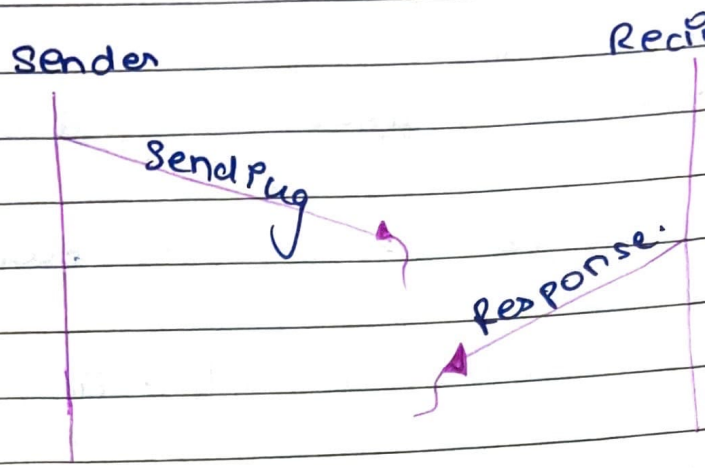
2) Single message buffer → Synchronous communication
→ OTP
→ Message will be in sender and reciever address space.

3) Unbounded capacity buffer → unlimited usage.

4) Bounded capacity buffer → limited

PAGE No.
DATE / /

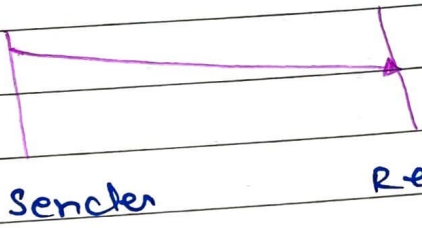
Failure handling techniques in message passing.



1) Loss of request → reciever is crashed.
   ↳ lack of buffer.

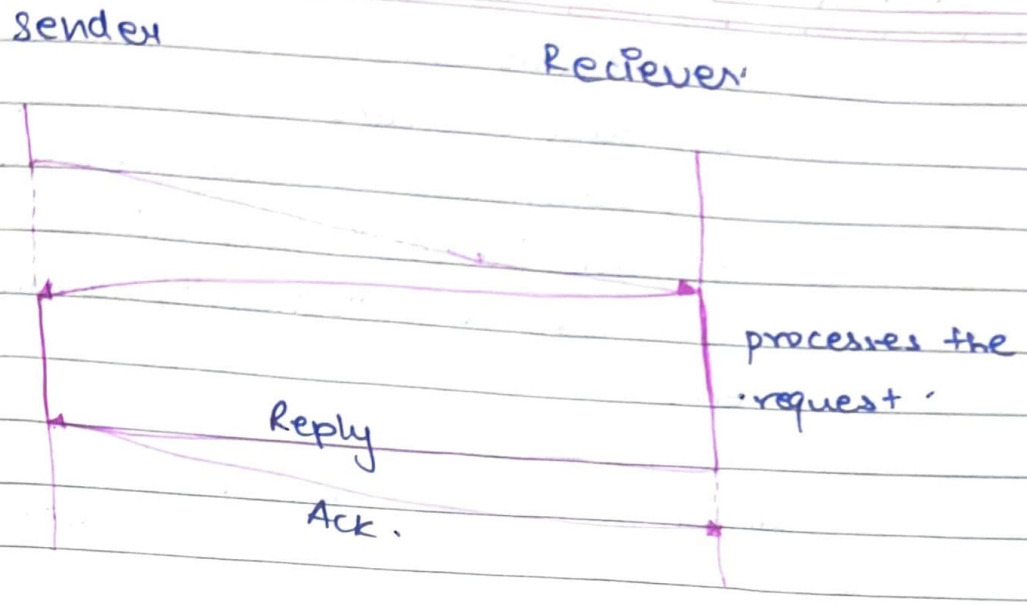
2) Loss of response.

3) Unsuccessful admission of request.



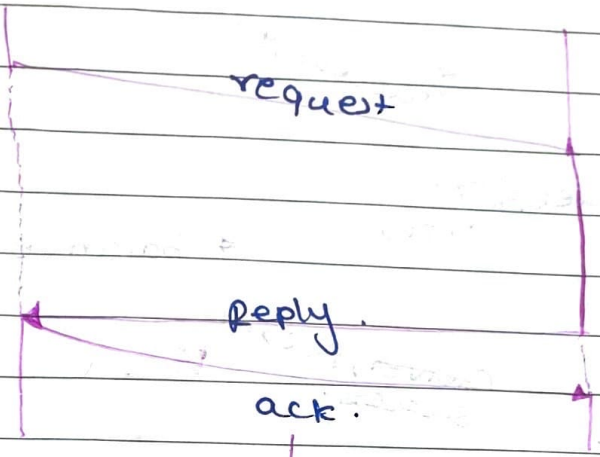
→ Multiple execution, server crashed.

Protocol



server

useless because process is in blocked state for too long.



→ If acknowledgement is removed it does not remain reliable.

→ Idempotency means repeatibility.

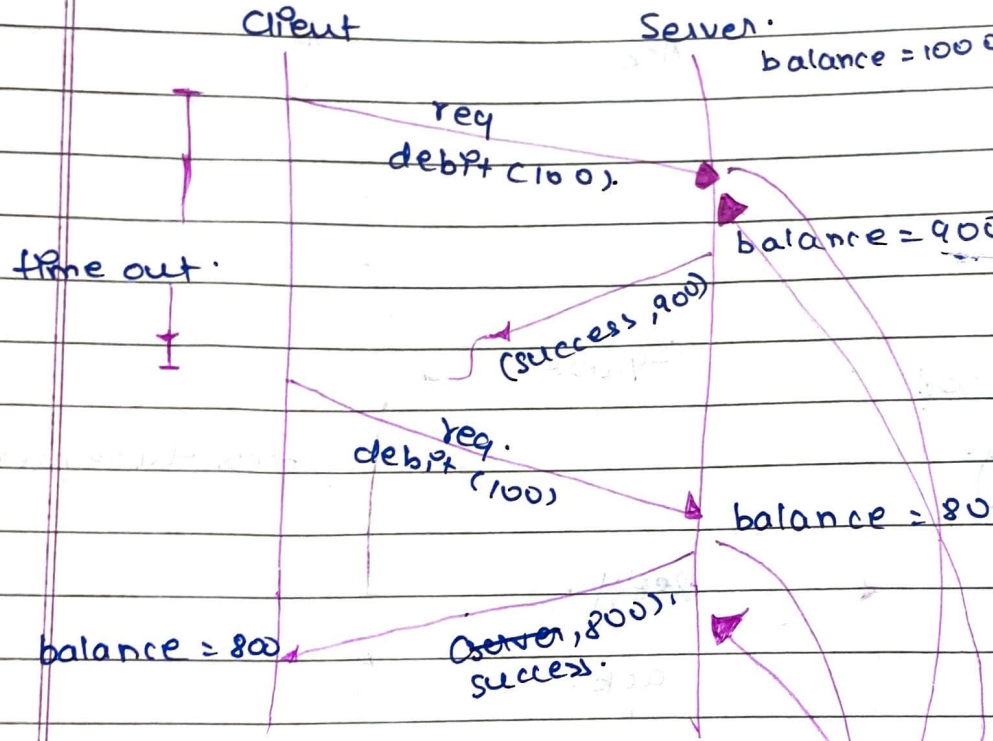
→ Idempotent operation produces same result without any side effects no matter how many times it is performed with the same arguments. (consistency)

→ Operations that do not necessarily produces same results when executed repeatedly with the same arguments are said to be non-Idempotent.

Eg. Loading the website.
Authentication while login.

max, sqrt functions used.

debit (amount).
if (balance > amount)
    balance = balance - amount;
return ("success", balance);
else return ("failure", balance);
end.



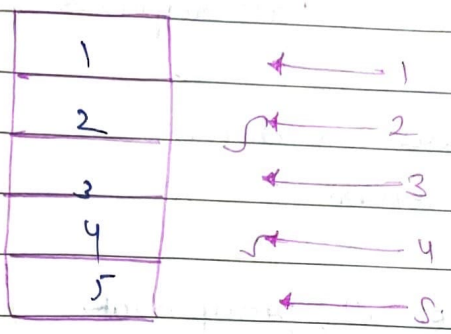
* To prevent this from happening, cache table is maintained

req. id.	reply to be sent
1.	900.

Multidatagram → Single message with multiple ~~no~~ packets.
Issues → would not be in sequence, packet loss, waiting increases when stop and wait is used.
Reducing ack. → using blast protocol → ~~ack~~ only 1 ack for all packets

Stop and wait → synchronous communication

→ Conditions required → packets received in sequence
→ Receiver has cache to store the sequence id.



what to do in this case?
→ The reciever already has a count of the no. of packets.
→ We have to use selective repeat after timeout.

Group Communication

1) 1 - M
2) M - 1
3) M - M
4) 1 - 1

* Broadcast → maximum utilization of server, time saving

Group management

→ open group
→ close group

Group addressing

→ multicast and broadcast ~~messaging~~ addressing.
* multicast → asynchronous communication → since & broadcast sender does not wait for acknowledgement.
→ both buffered and unbuffered
→ shared-space or bulletin board architecture is used.

PAGE No.
DATE / /

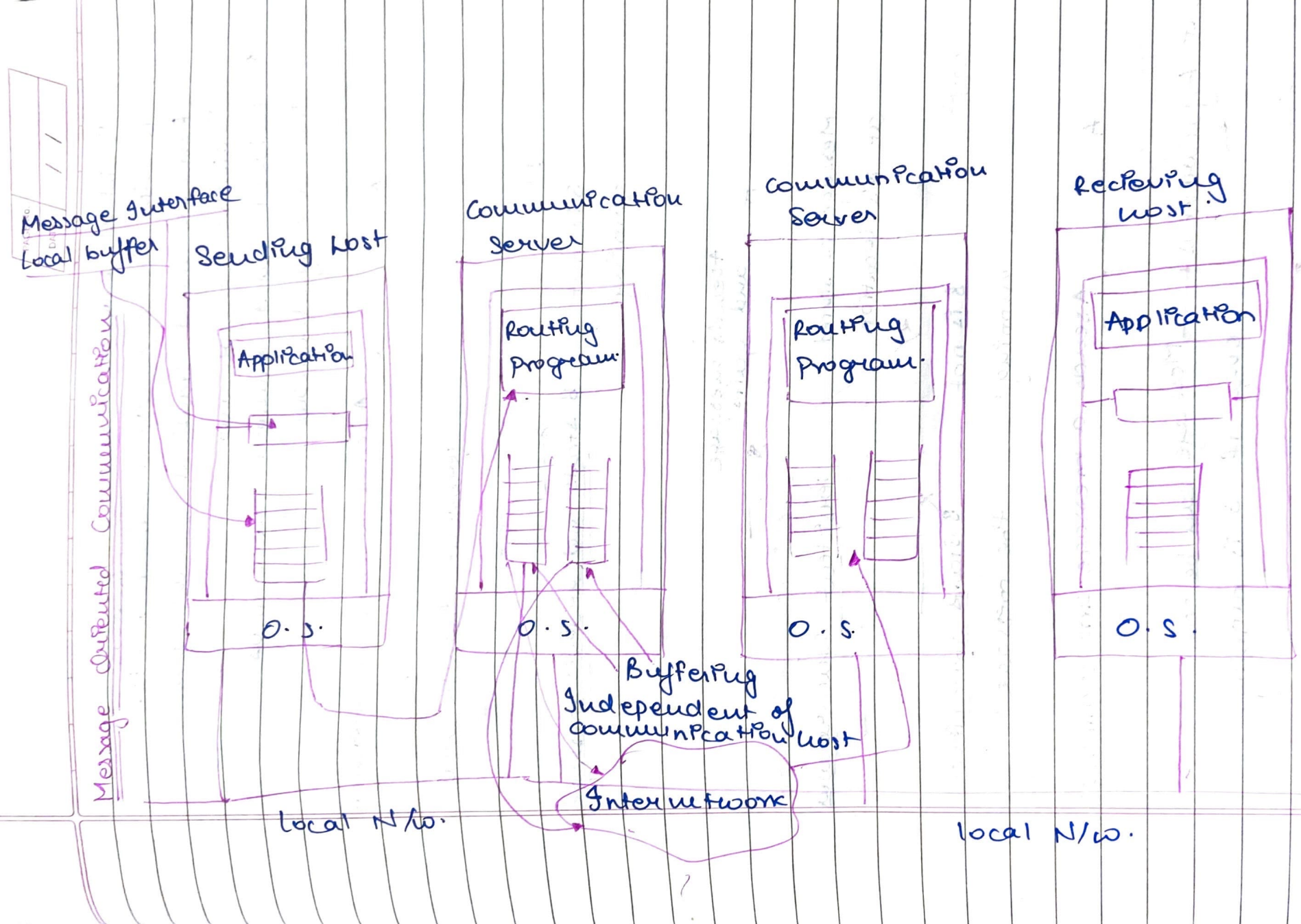
(*) Flexibility and reliability in multicast communication

- Zero reliable → no response.
- 1 reliable → one response.
- w/m reliable → n out of m responses.
- all reliable → all responses

2) m to 1 → many servers one client.
→ Eg. moodle server.

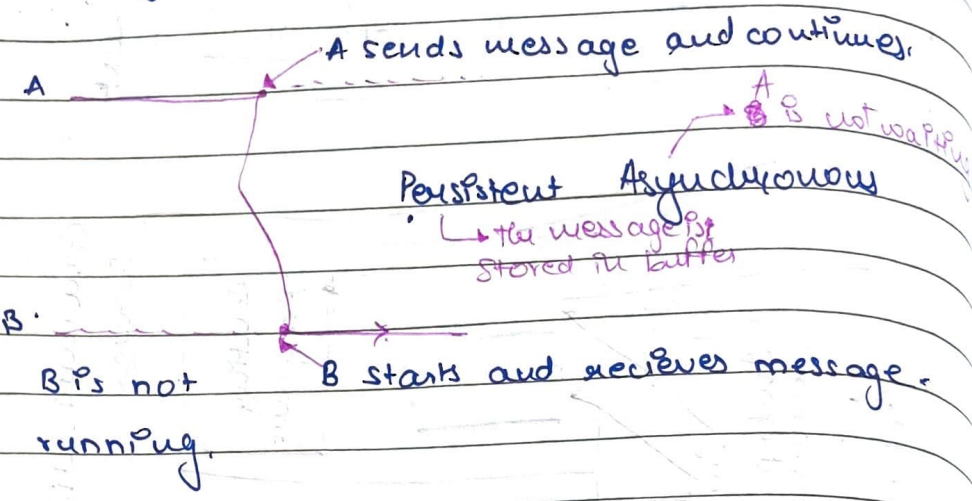
3) m to m. → Eg. Conference call, group chat.
→ Order should be maintained.

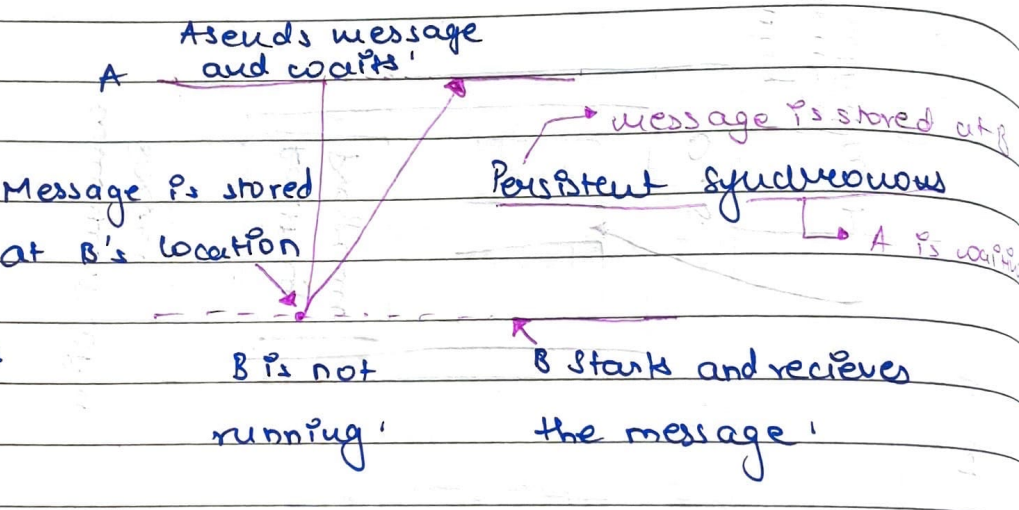
Assignment → Failure handling, reliability, buffering of IPC, group communication for diagrams, use cases, protocols, any application (social media) → study and write in your own words.
Any feature required to add. (Eg. last seen, status should not be seen). 5-6 pages. (10 marks). 5th October.
First week of October.


Message Interface
Local buffer
Sending host
Application
O.S.
Communication Server
Routing Program
O.S.
Communication Server
Routing Program
O.S.
Receiving host
Application
O.S.
Message Oriented Communication
Buffering Independent of communication host
Internetwork
local N/w.
local N/w.

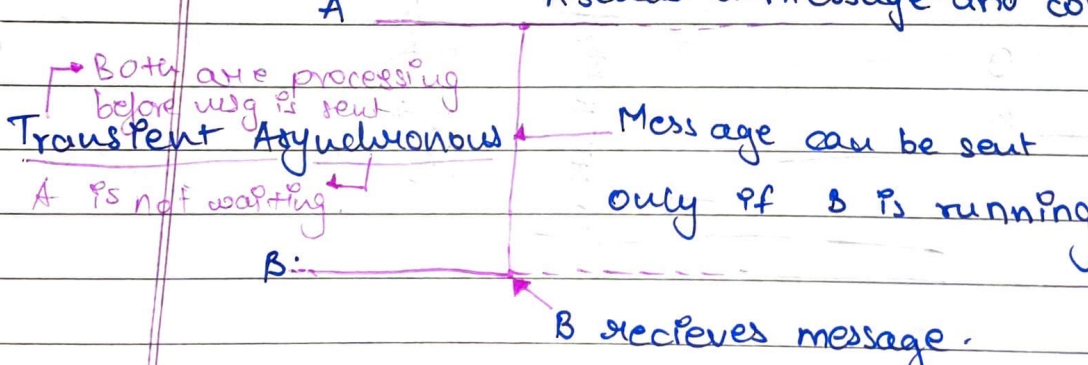
1) Persistent Synchronous
2) Receipt based transient ~~Synchronization~~ Synchronous.
3) Response based transient synchronous.
4) Persistent synchronous
5) Delivery based transient synchronous
6) Transient synchronous.

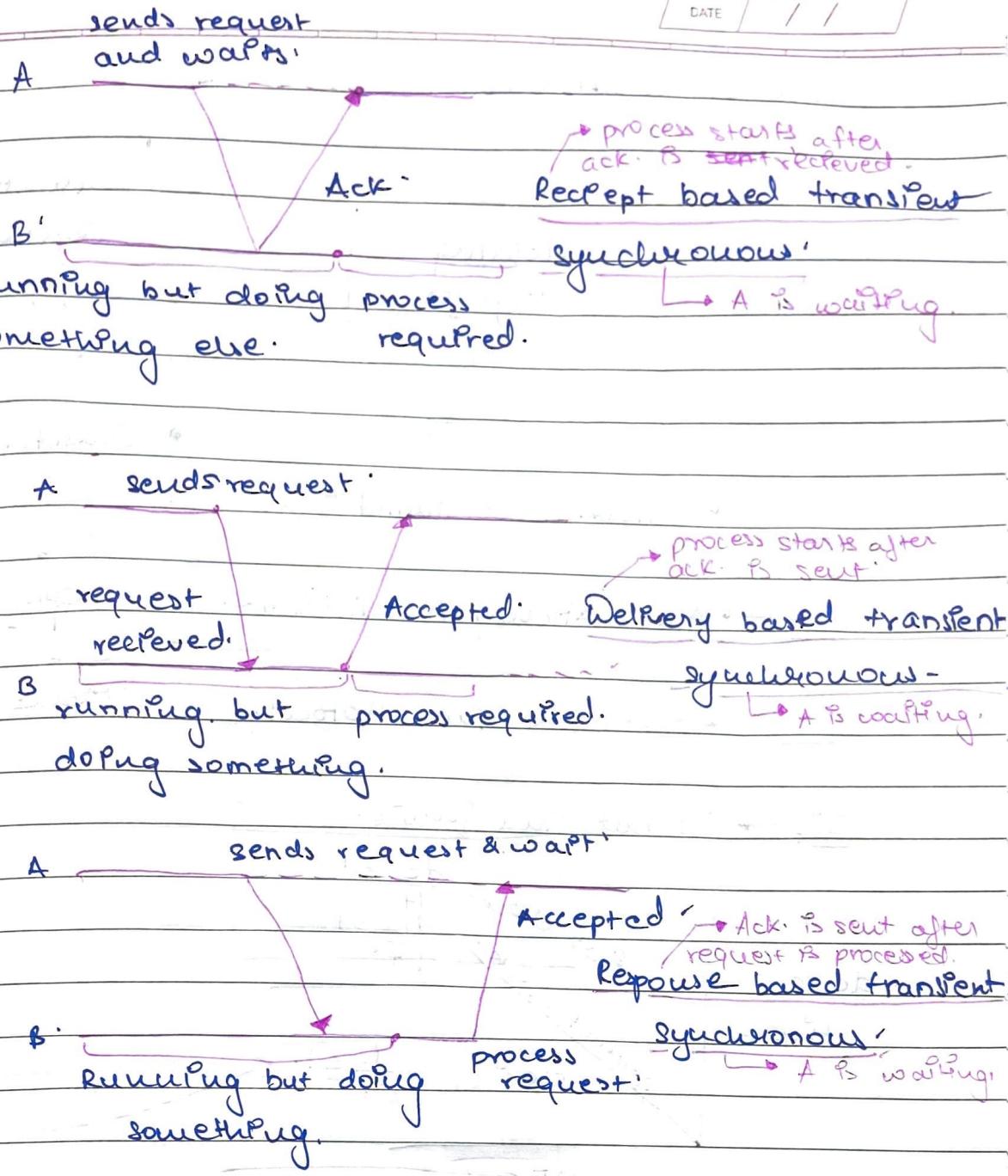
same





Eg. OTP

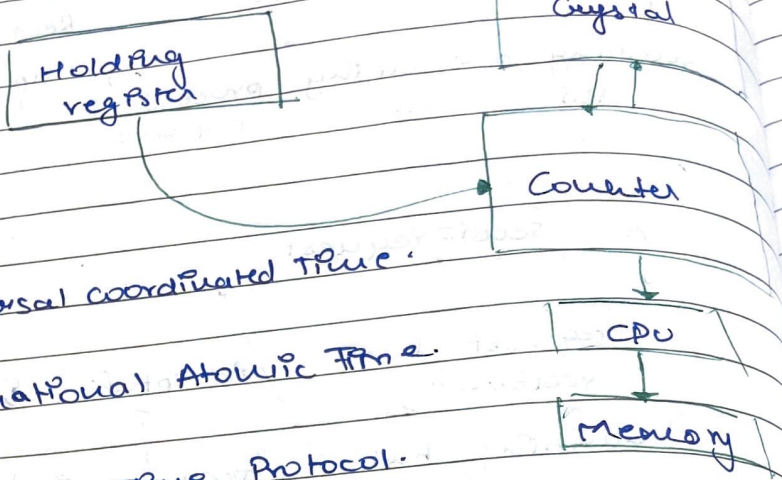




Q. Explain the different forms of communication with example.

* Persistent → message is stored.
transient → Both sides are active
Synchronous → waits
Asynchronous → does not wait.

Clock Synchronisation

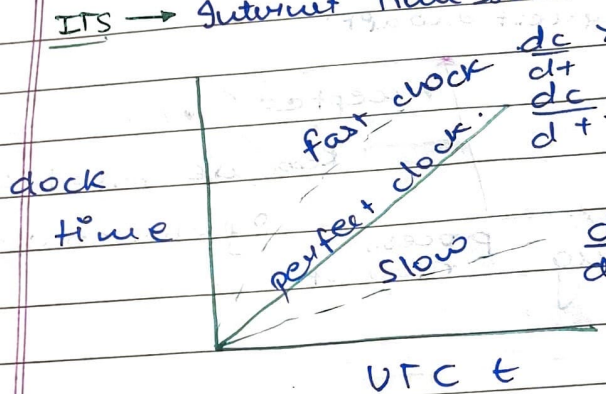


UTC → Universal Coordinated Time.

TAI → International Atomic Time.

NTP → Network Time Protocol.

ITS → Internet Time Service.



* Cannot adjust manually → the transaction would not stay valid.
* OS provides with the date & time.

$$1 - \delta \leq \frac{dc}{dt} \leq 1 + \rho$$

PAGE No.
DATE / /

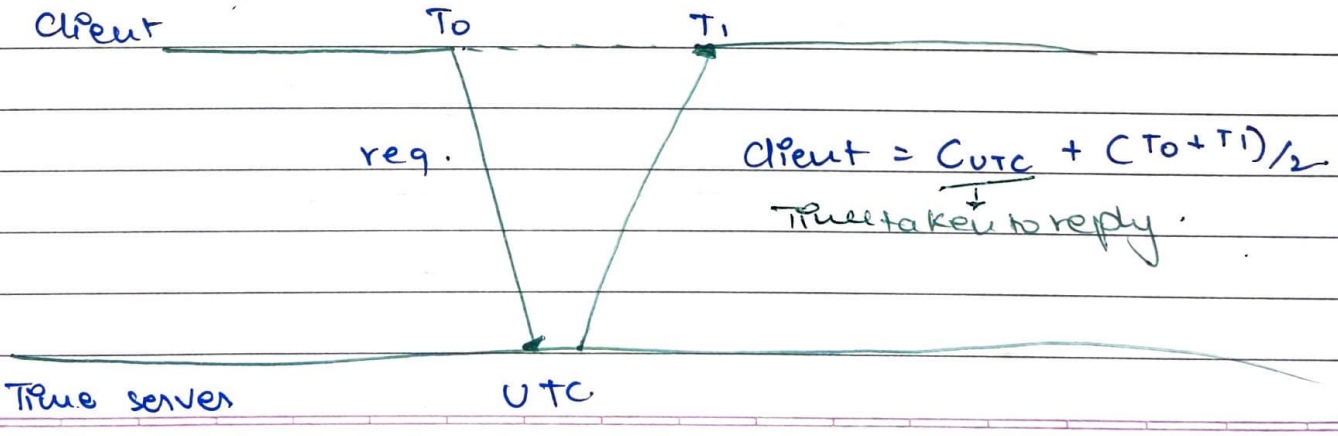
Centralized → where process can share clock & memory. It relies on shared ~~mem~~ memory.

→ Centralized clock synchronization algorithms are passive time server & active time server.

① Christian's Algorithm
② Berkeley's Algorithm.

① Christian's Algorithm.

→ Passive time server algorithm.
→ users RPC.
→ There is a machine with UTC reciever which recieves precise UTC.
→ Machine sends a request to time server at least $\delta/2\delta$, where δ is a maximum difference allowed between clock & UTC.
→ The time server sends reply message with current UTC where ~~req~~ recieves ~~required~~ request.
→ The machine measures the delay between the server's sending message & machines recieving message. Then it ~~used~~ uses the measure to adjust the clock.



clock drift → clocks running at different rates.

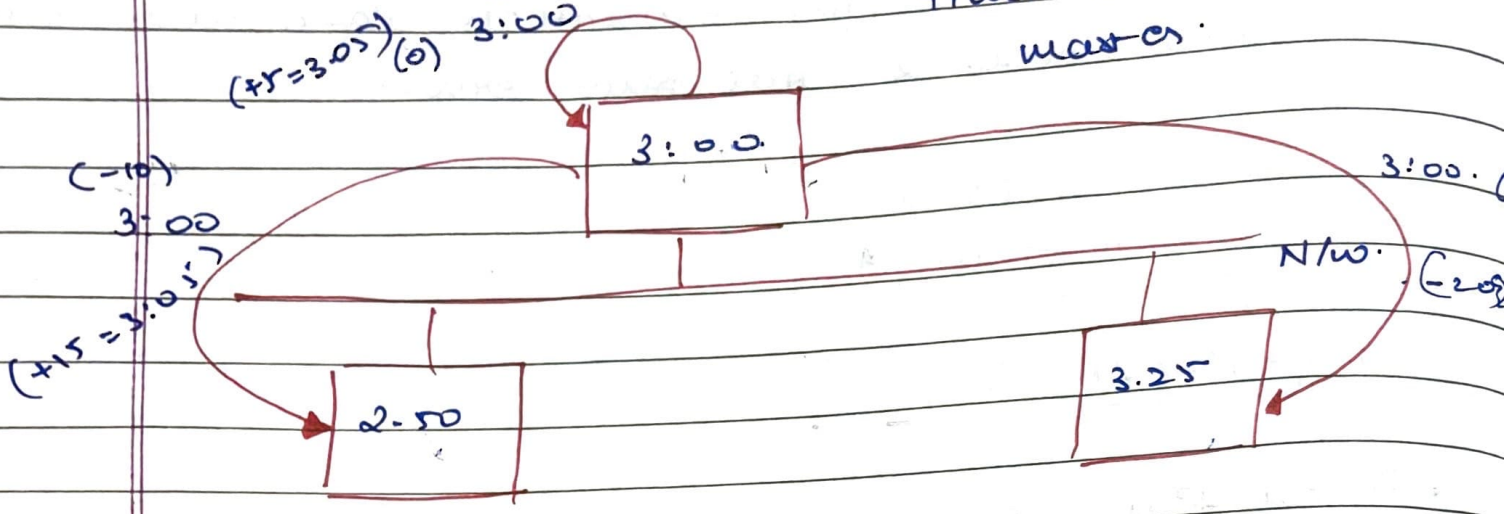
PAGE No.
DATE / /

② Berkeley's Algorithm

→ used in systems without UTC server.

gives time.

time deamon master.



25 - 10 + 0 = 15 $\frac{15}{3} = 5$.

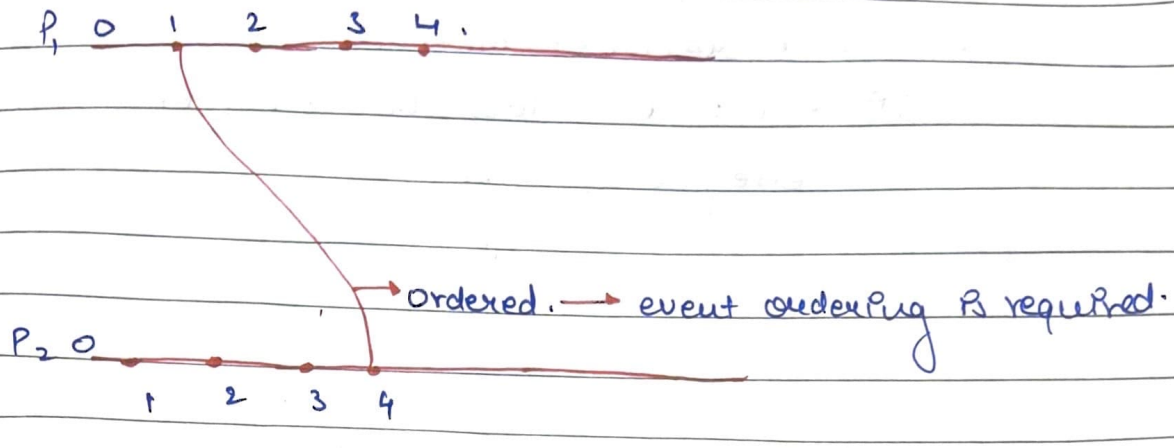
* If master crashes, new one is appointed with elec algorithm.

→ Discard the higher and lower values to take avera

* Clock synchronisation is done to maintain consiste which is required in any system.

→ real time is not required in clock synchronization

Logical clock.



* Lamport clock is used.
* Concurrent events → same timing events.

Happened before Relation

→ If A & B are the events in the same process, and A is executed before B, then A → B.

→ If A represents sending of message & B is a reciept of message, then A → B.

→ Relation is transitive

A → B, B → C then A → C.

→ Relation is undefined across the processes that donot exchange messages.

Event ordering using Happened Before

→ Define the notion of time:

If A → B, then C(A) < C(B).

→ If A & B are concurrent,

then C(A) <, >, =, C(B).

→ Each process maintain the logical clock (cp)

→ whenever event occurs locally at i,

$LC_i = LC_i + 1$

PAGE No.
DATE / /

→ When I sends message to J, piggyback LC_i.

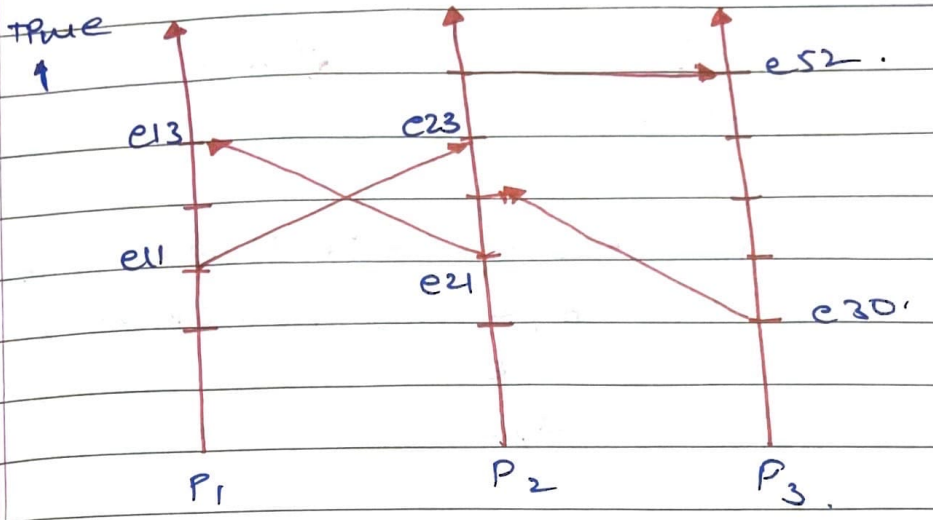
→ When J receives message from i, if $LC_j < LC_i$, then $LC_j = LC_i + 1$.

else, do nothing.

PAGE No.
DATE / /

Lamport Clock

→ If absolute time is not important, this can be used
↓
Global time.



Eg. $e_{11} \rightarrow e_{20}$, $e_{21} \rightarrow e_{30}$ → concurrent events.

→ Used to store in the database and fetch it.

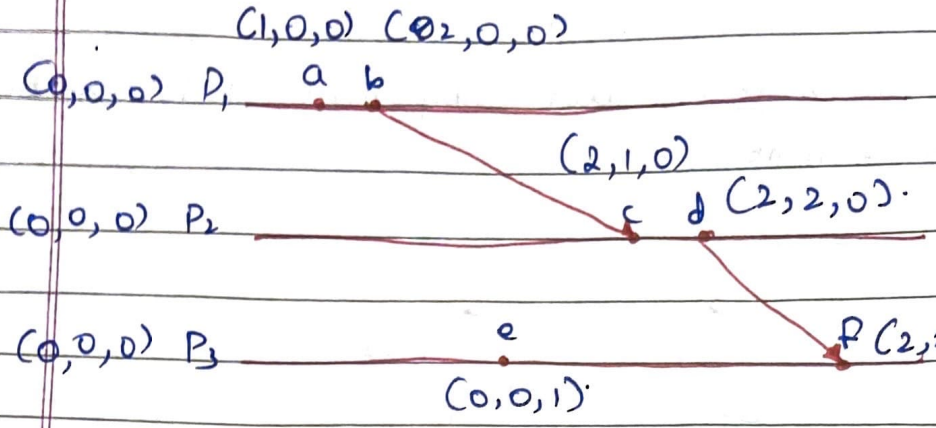
Vector clock

→ Each process is maintains a vector V_i
- $V_i[i]$: number of events that have occured at i.
- $V_i[j]$: number of events we know that have occured at j.

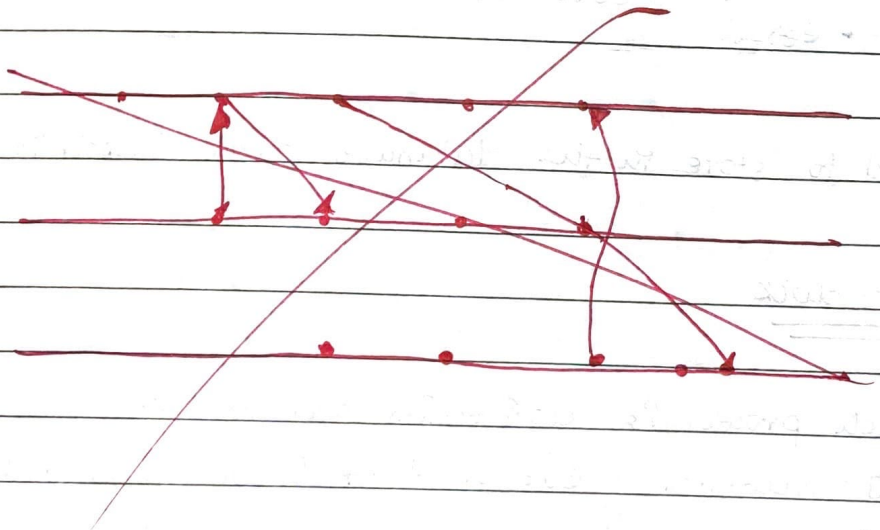
→ Update the vector clock as follows
- Local event : increment $V_i[i]$.
- Send message: piggyback entire vector V.
- Receipt of message $V_j[k] = \max(V_j[k], V_i[k])$

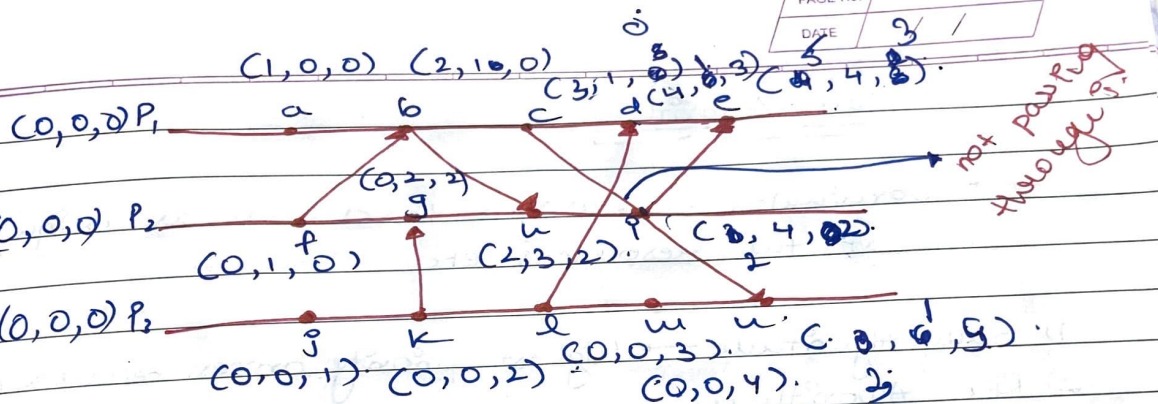
a) Receiver is told about how many events the sender knows occured at another process k.

Also, $V_j[i] = V_j[i] + 1$.



Concurrent events → a, e → If there is no communication.
b, e
c, e
d, e.

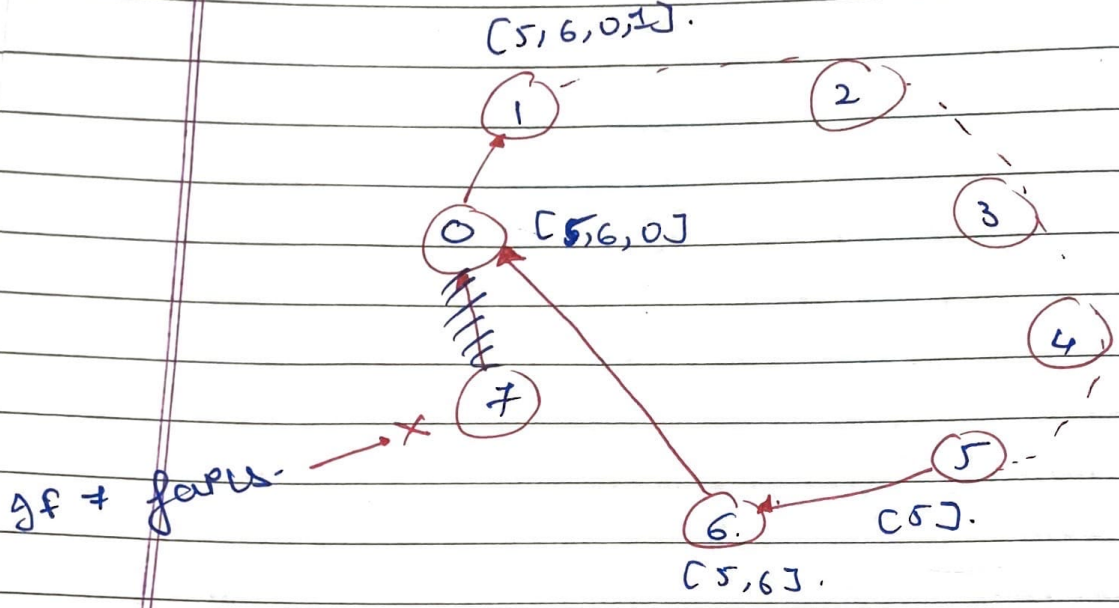



(1,0,0) (2,1,0)
(3,1,3)
(4,4,5).
(0,0,0) P1
a
b
c
d
e
not passing through
(0,2,2)
g
(0,0,0) P2
f
(0,1,0)
(2,3,2).
h
p
(3,4,5).
q
(0,0,0) P3
j
k
l
m
n
(0,0,1).
(0,0,2)
(0,0,3).
(0,0,4).
(3,4,5).

Election Algorithm.

→ Coordinator → assigns tasks such as assign or request the resources etc.

1) Bully Algorithm → highest priority process becomes coord
2) Ring Algorithm.



Q. Which Algorithm is better? Why??

* It sends (Idt message).
* When it again reaches (5) then it looks for higher priority in the array and chooses that as coordinator
* How will we know (7) has failed → ping comm...

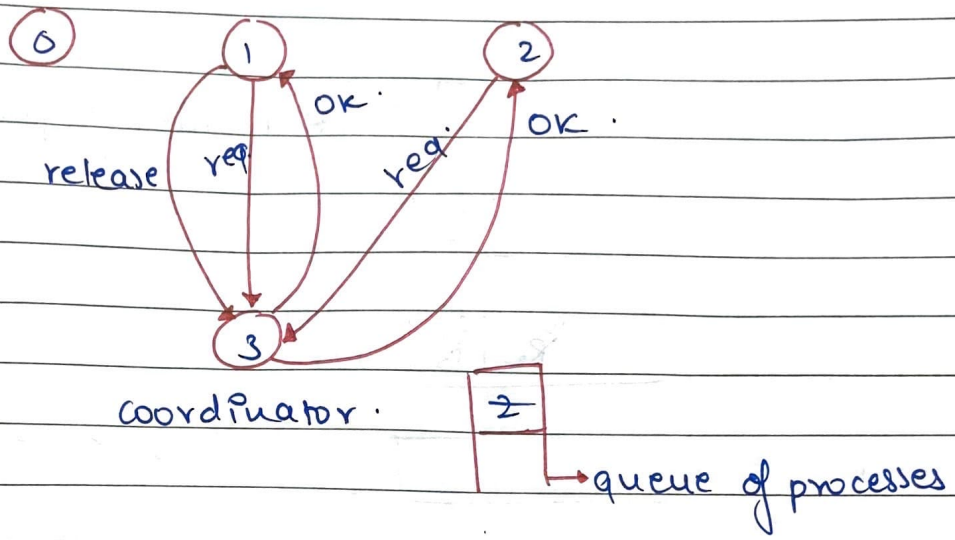
28 21 34

PAGE No.
DATE / /

Mutual Exclusion Algorithm.

→ Simultaneously 2 processes cannot be executed.
→ only 1 process can use the resources at a time.

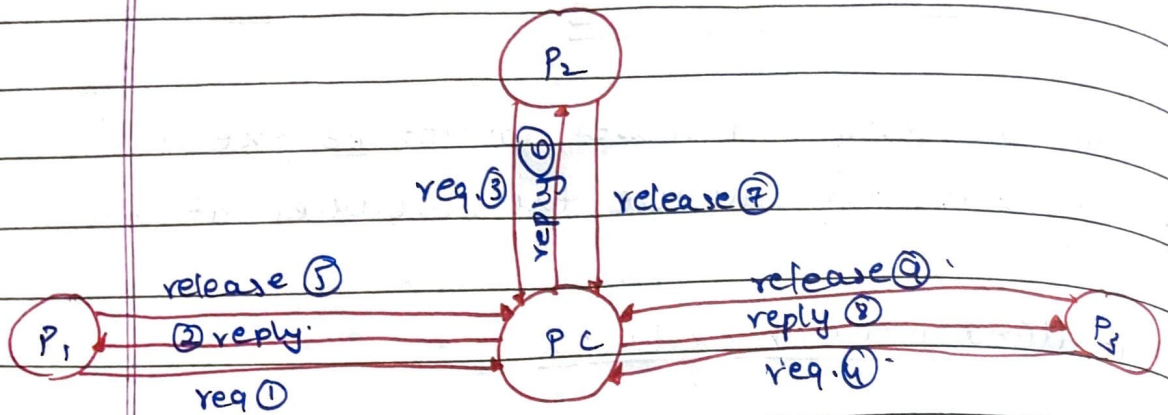
① Centralised Mutual Exclusion

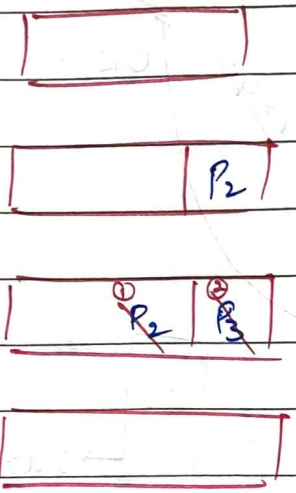


① request
② use
③ release.

Disadvantage

1) Centralised coordinator crashed
2) Starvation.


P2
req ③
reply ⑥
release ⑦
release ⑤
② reply
req ①
P1
PC
release ⑨
reply ⑧
req. ④
P3


Initial states of queue.
P2
① P2
② P3

For the arrival of new events, timestamp has to increase by 1.

PAGE No. / DATE / /

(2) Distributed Mutual Exclusion

→ Based on event ordering & time stamps.

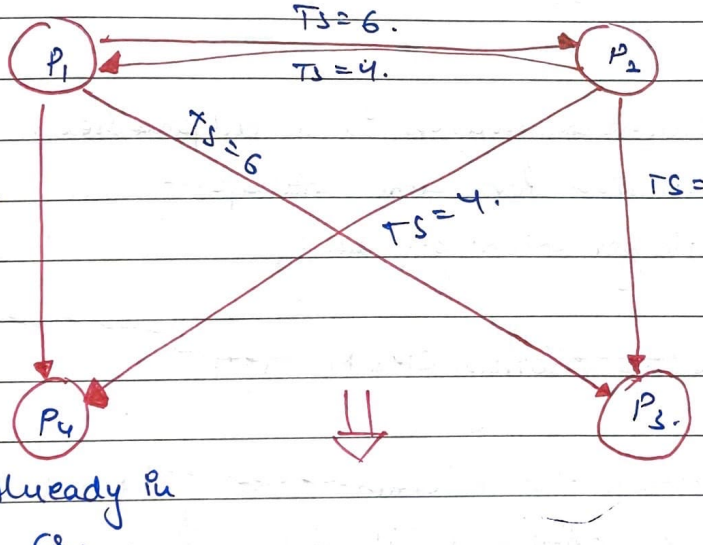
→ Process K enters critical section as follows:

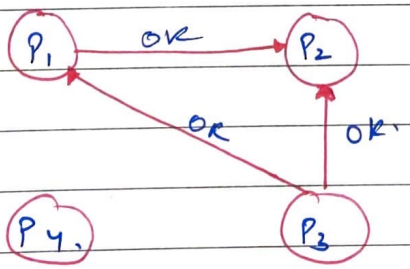
- Generate new time stamp $T_{SK} = T_{SK} + 1$
- Send request (K, T_{SK}) all other n-1 processes
- Enter critical section
- Upon receiving a request message, process P_j:

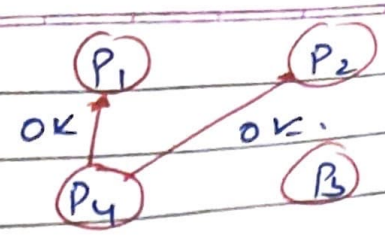
a) Sends reply if no contension.

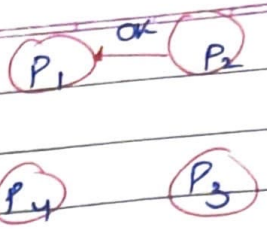
b) If already in critical section, does not ~~be~~ reply, queue required.

c) If wants to enter, compare T_{Sj} with T_{SK} & sends reply if $T_{SK} < T_{SK}$, else queue.









Difference

→ No coordinator in distributed.

→ Single point of failure & 'n' point of failure in centralized and distributed respectfully.

(*) Timeout mechanism used to solve ~~single~~ 'n' point of failure.

DP → Advantage

→ All processes know id of all processes.

→ Permission for time stamp (waits for other processes)

(*) Solution → Voting (Any Majority)

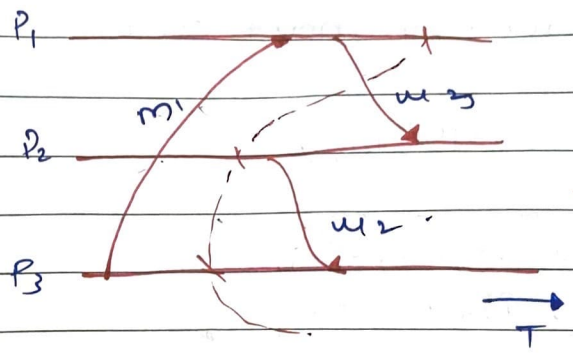
Q. Either draw the diagram through the time stamps given or write the algorithm through to the diagram given.

PAGE No
DATE / /

Global State

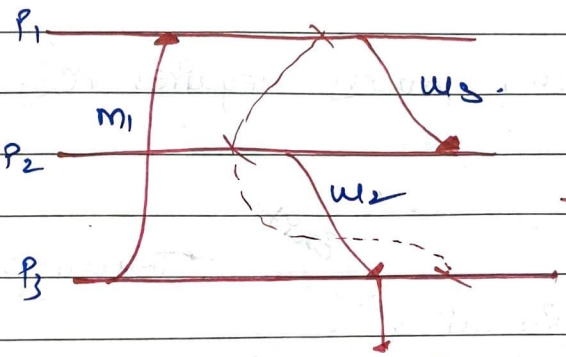
→ Global State of D.S.
- Local state of each process
- Messages sent but not received.

→ Many appl^{n} need to know the state of the system.
- failure recovery, distributed deadlock detection.



→ consistent.

→ consistent.



→ Inconsistent.

Because it cannot be received before checkpoint.
It is received without being sent.

PAGE No.
DATE / /

Processes

1) Multiprogramming → Single Processor

2) Multiprocessing → CPU bound
→ I/O bound

Q. Why all processes are not either CPU or I/O bound?

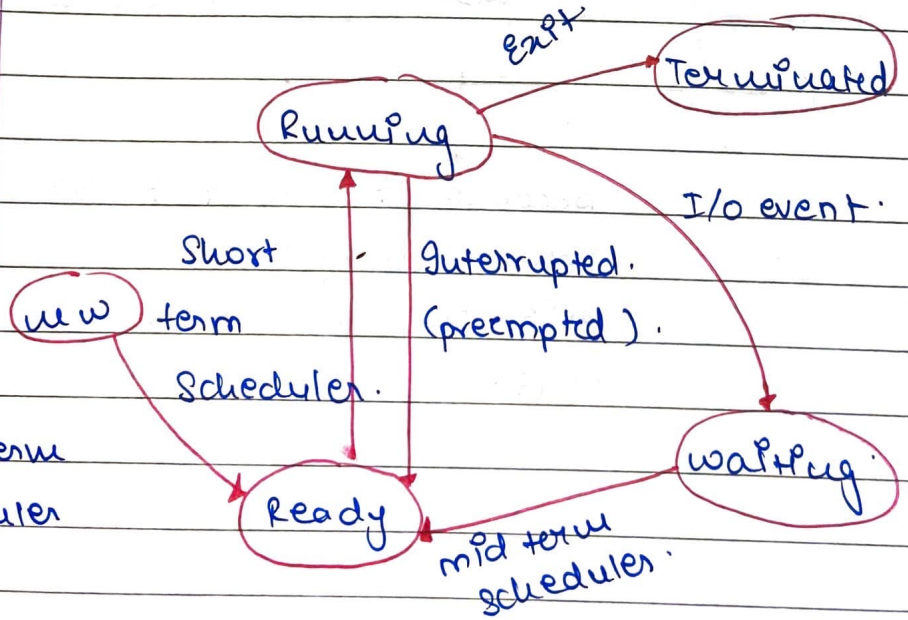
→ Multiprogramming won't occur.

prog 1 | P_{11} CPU | I/O | CPU | I/O | CPU |

prog 2. | CPU | I/O | CPU | I/O | CPU (P_{23}) |

P_{11} P_{21} P_{13} P_{24} P_{13} P_{23}

* Each an every process requires PCB.



PAGE No.
DATE / /

(*) System call → interrupt generated by the system.

(*) Ready queue and long term scheduler is required because we need to balance I/O and CPU bounds.

Objectives of scheduling

1) Maximum CPU utilization
2) Throughput becomes maximum.
3) minimum waiting time
4) response time.
5) minimum turn around time.

CPU scheduling Algorithms

Non-preemptive	Preemptive.
1) FCFS	1) SJF (SRTF)
2) SJF	2) Priority
3) Priority	3) RR.
	4) EDF

Q. How to know if it is pre-emptive or not?
→ Starting time of all processes is same → non-preemptive
Hence, why RR is ~~no~~ preemptive.