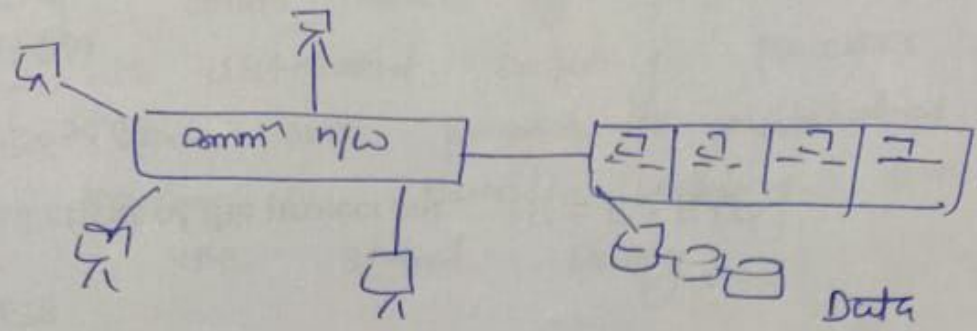
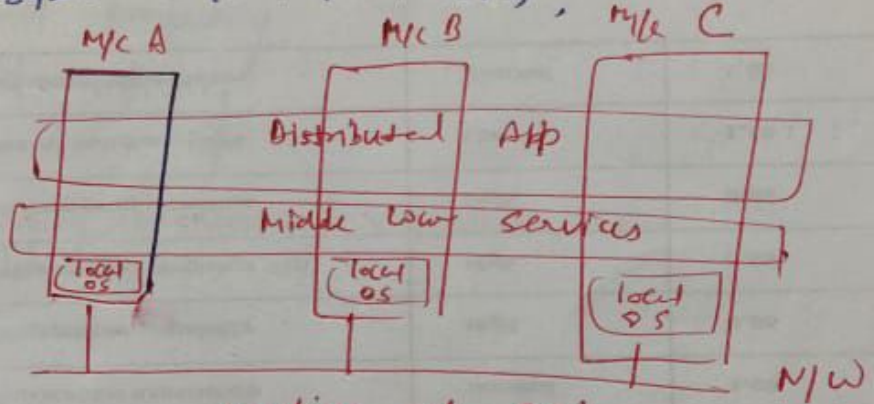
Distributed System – is a collection of autonomous computer linked by a computer N/W and equipped with distributed system software.

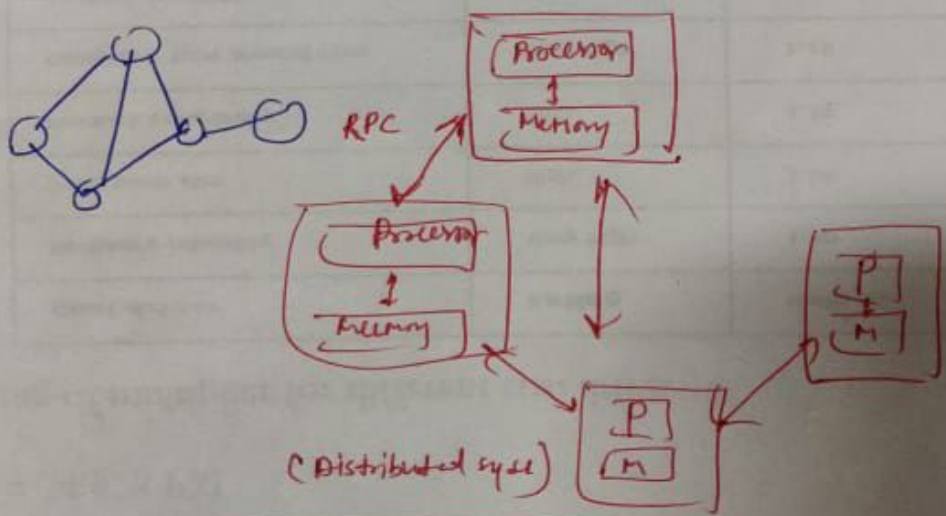


⇒ A distributed system S/W enable the distributed system to coordinate their activity, share the resources of the system (H/W, S/W, data).

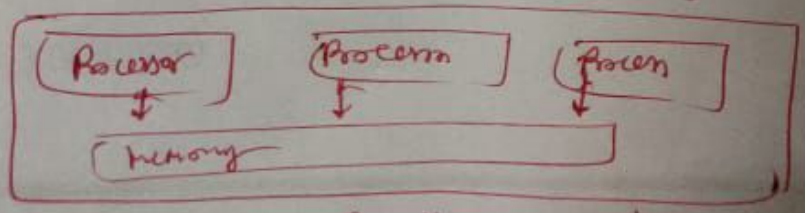


(working of DS)

⇒



(Distributed syst)



(Parallel system)

→ Subprocesses must be synchronized
(variable value must be updated for all.)

Sol^n / Requirement of DS →

1. Comm^n channel
2. Clock synchronized, process synchronization

Deadlock ✗

Advantage →

1. Speed
2. content distribution
3. Scalling
4. Parallelism
5. fault tolerance
6. Redundancy

Problems / Disadvantage →

1. N/w channel failure
2. complexity
3. Security concerns
4. Higher initial cost

local clock	Global clock

Examples —

Telecomm^n n/w
The internet
Airline Reservation^n syst^n
Distributed database
Scientific computing
Distributed rendering

Distributed vs. Parallel computing →

There is no clear distinction b/w the two. Parallel computing is considered as form of distributed computing that's more tightly coupled.

for ex- in distributed computing processors usually have their own private or distributed memory, while processor in parallel computing can have access to the shared memory.

Types of Distributed syste –

1. Clusters computing
2. Grid computing
3. Cloud computing

working of Distributed systems –

Characteristics of Distributed System:-

1) Resource sharing – is the ability to use any h/w, s/w & data anywhere in the system.

2) Openness – is concerned with extensions & improvements of D.S.

3) Concurrency – It arises naturally in D.S, with seperate activities of the user, independancy of the resources & the location of server process in a seperate computer.

4) Scalability – ↑ the scale of system.

5) Fault tolerance – It cares the reliability of system.

6) transparency – hides the complexity of d.s to the users & application programs.

Applications of D.S –

(1) finance & commerce
e.g E-commerce {amazon, e-bay, online banking}

(2) The information Society
↳ Search engines, wikipedia, Social nw'g.

(3) Creative industries & Entertainment
↳ Online gaming, music, youtube.

(4) Health care – Online patient record, health information

(5) Education – E-learning

(6) Transportation & logistic – GPS, google, mups.

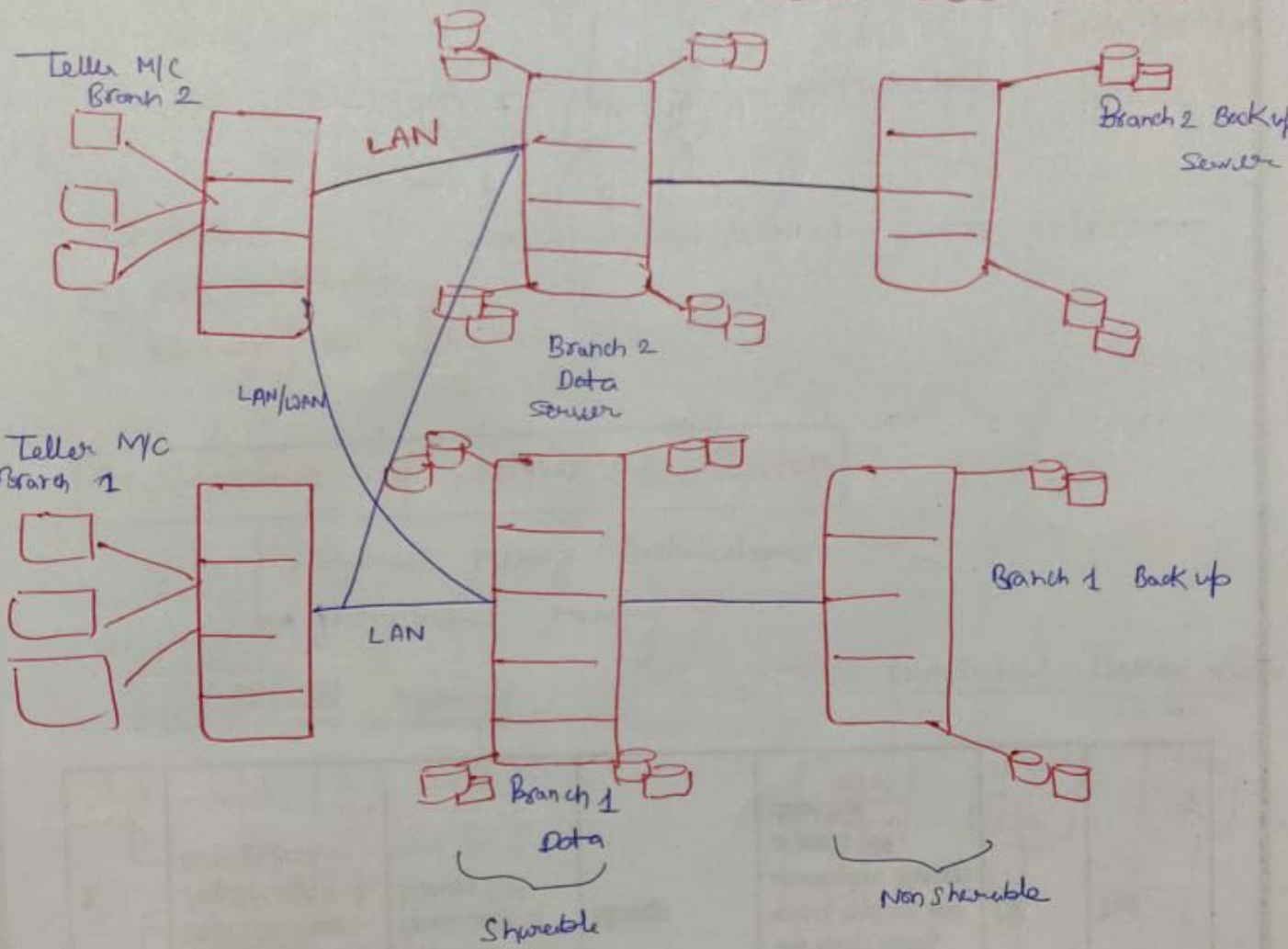
(7) Science

(8) Environment Management – Sensor technology.

Working of Distributed System :-

Distributed System with Example :-

① Normalized Bank with multiple branch offices



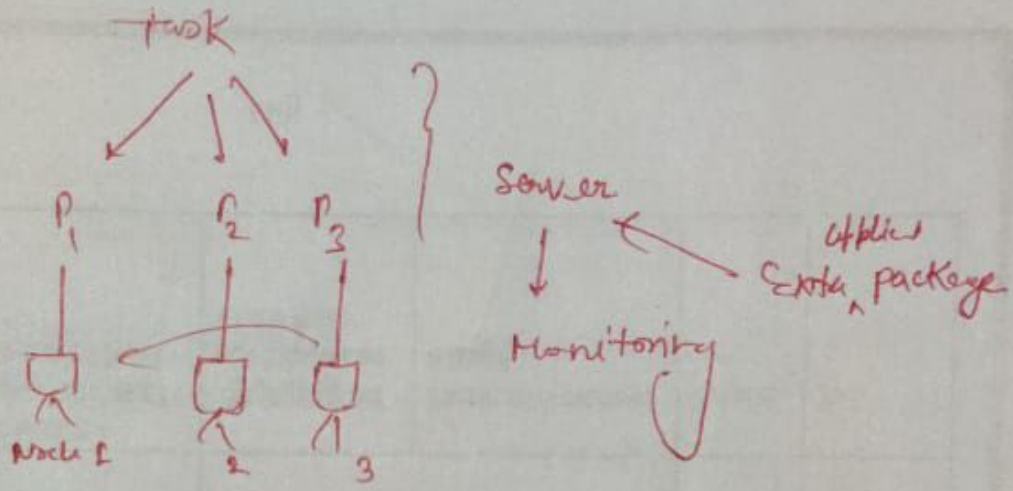
Requirement of Data ~~Structure~~ –

① Security & reliability

② Consistency of replicated data

③ Concurrent transaction

④ fault tolerance

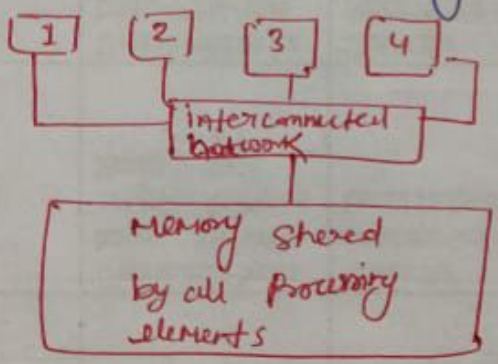


① Status
② Task load balancing
③ Monitoring the nodes

Architecture of Distributed System —

→ Shared Memory Architecture
→ Distributed Memory

Shared Memory

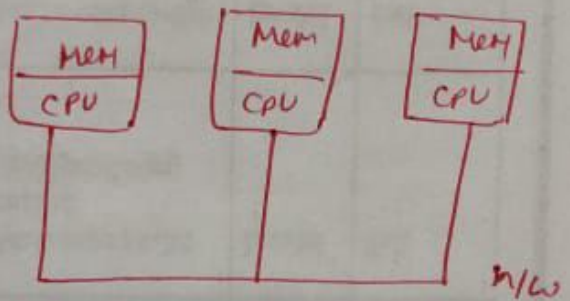


{ Tightly coupled syste }

⇓

Easier to program

Distributed Memory Archi



{ loosly coupled sys }

⇓

Complex

Distributed Computing Models

Fundamental Models

Based on some fundamental properties such as → characteristics
→ failure
→ Security

Example - Interaction Model
failure Model
Security Model

Architectural Models

Based on Architectural Style.
→ client - server Model
→ Peer to Peer Model

ARCHITECTURAL MODEL

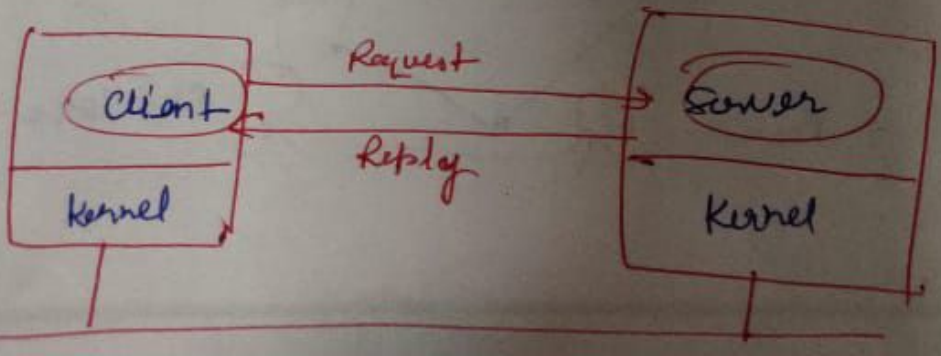
Deals with organization of components across the network of computers and their interrelationship.

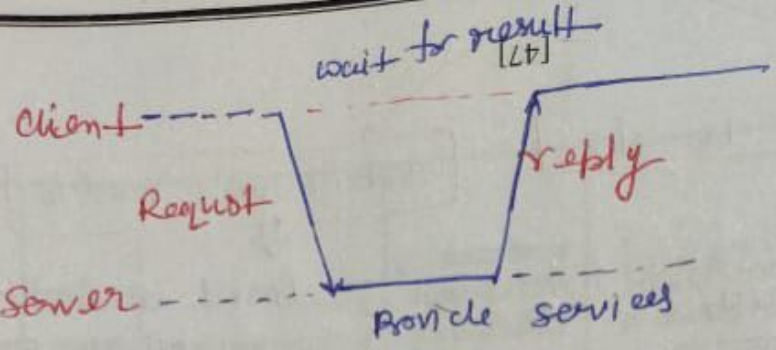
- Client Server
- Peer-to-peer

① Client - Server Model

→ The most important & most widely used distributed system architecture.

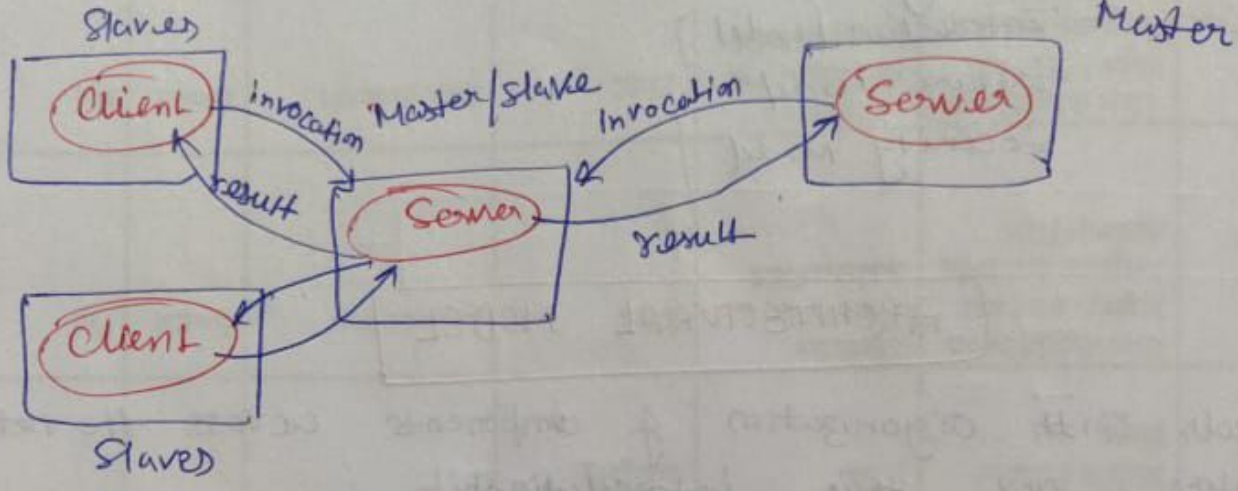
→ Client - Server roles are assigned & changeable.





time-diagram

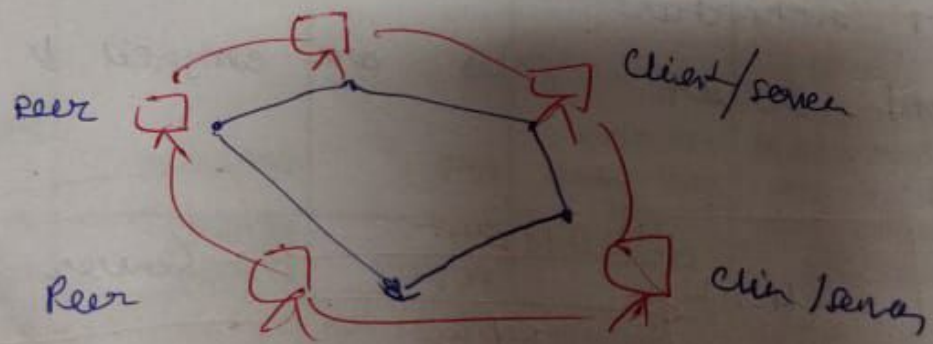
Client invoke individual servers

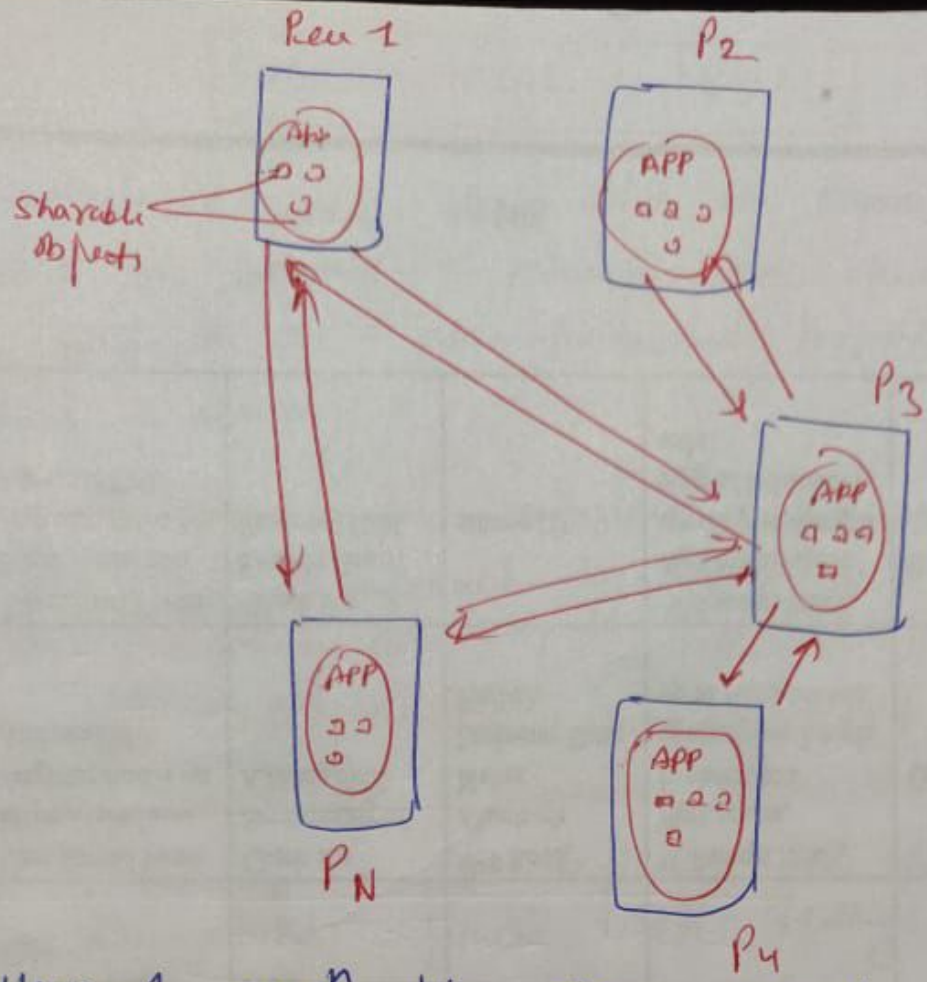


(2) Peer to Peer Model

→ Unlike client-server, P2P model does not distinguish b/w client/server instead each node can either be a client or server depending on whether the node is requesting or providing the services.

→ Each node is considered as a Peer.





→ Pattern of comm^n b/w them depends entirely on application requirements.

→ Each object is replicated in several computers to further distributed the load and to provide resilience in the event of discomm^n of individual computers.

→ Need to have & retrieve individual computers is more complex then in client – server architecture.

FUNDAMENTAL MODEL

Architectural Model deals with the organizational component across the network whereas the fundamental model is based on the some fundamental properties like failures or security.

(1) **Interaction Model** – → computation occurs with in process.

→ The processes interact by passing messages resulting in –
- communication (information flow)
- co-ordination (synchronization + ordering of activities) between processes.

→ Interaction model reflects the facts that comm^n takes place with delays.

→ performance of comm^n channel
- * latency (message, n/w, System) How much time it takes for a packet of data to get from one designated point to another (RTT)
- * Bandwidth
- * Jitter (packet delay variance) variance in time delay in MS b/w data packets over a network.

→ Two variants of the interaction model.

① Synchronous
② Asynchronous

→ Synchronous DS – process is executing in a known lower / upper bounds time.
- Message is received with in a known bounded time.
- known local clock drift rate.

→ Asynchronous DS – It has no bounds on
- Process execution speed
- message trans delay
- Clock drift rate. (clock doesn't run at exactly the same rate as a reference clock.)

(2) FAILURE MODEL

→ Failure Model defines & classifies the faults.

→ It is important to understand the kinds of failures that may occur in a system.

(1) Fail stop – A process halts and remains halted, other process can detect that the process has failed.

(2) Crash – A process halts and remains halted, other process may not be able to detect this state.

(3) Omission – A msg inserted in an outgoing message buffer never arrives at the other ends incoming msg buffer.

(4) Arbitrary – Process / channel exhibits arbitrary behavior: It may send / transmit arbitrary messages at arbitrary times, commit missions; a process may stop or take an incorrect step. (unfair)

(5) Timing failure – clock drift exceeds allowable bounds

(3) Security Model

There are several potential threats a system designer need be aware of:

(1) threats to processes: An attacker send a request & response using false identity.

(2) Threats to comm channels: An attacker may listen to messages & save. (eave drops).

(3) Denial of service: An attacker may overload a server by making excessive request.

→ Cryptography & authentication are often used to protect security.

Theoretical foundation for Distributed computing syste-

Limitation of Distributed System:-

(1) Absence of global clock-

(2) No shared Memory -

Opportunities -

> Performance through parallelism
> Resource sharing (Cost Effectiveness)
> Reliability & availability
  - avoid single points of failure
> Expandability

Limitations

> No Global clock
> No shared memory

→ Data may be scattered on different machines
  — No global memory

→ Decisions based on local information
  — but, information can be exchanged

→ Avoid single point of failure
  — to improve reliability, availability

→ Lack of a precise global time source
  — No common clock.

ABSENCE OF GLOBAL CLOCK

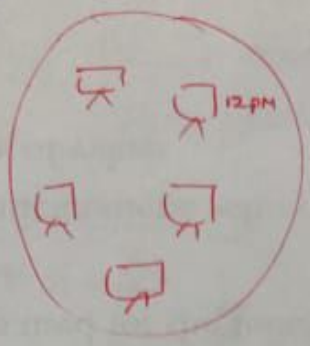
- On a single machine

> A clock can be used to determine ordering of events in concurrent processes.
> In DS, no concept of global time.
> It is difficult to reason about the temporal ordering of events.
  1) Cooperation b/w processes (eg. Producer/Consumer, Client/Se

2) Arrival of request to the OS. (resources)

3) Collecting up-to-date global state.

①



impossible to maintain - global time

(No global clock)

500 → 20 300

300 + 20 = 320 Rs.

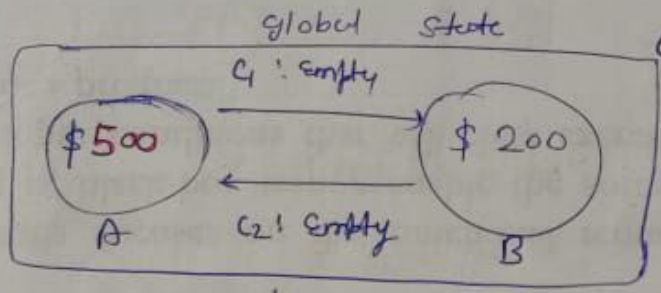
↓

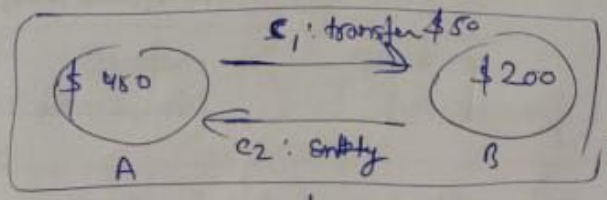
No hierarchy of events in distributed system, we are dependent on local clock of particular device.

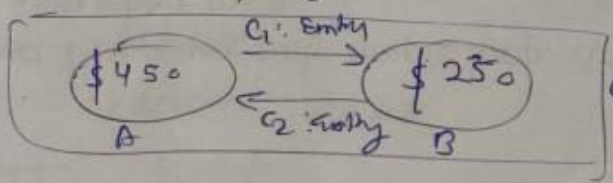
→ ~~Inconsistent~~

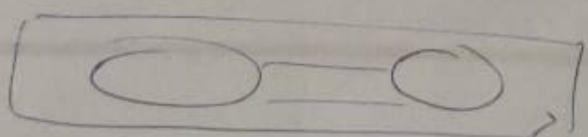
② It is better to have no shared memory, then is less possibility that we will lose all data in once.

But It is a limitation too. One process can not access all the data. (only partially they can access)





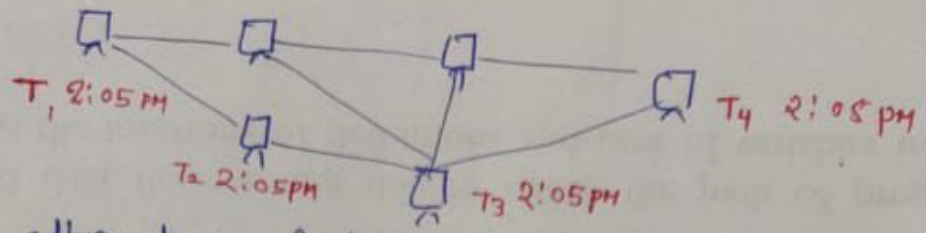




LOGICAL CLOCK

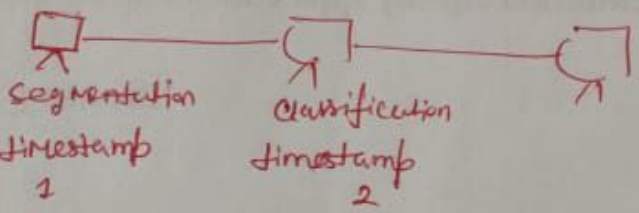
Global state
Local state

(1) To order event across process, trying to sync clocks is one approach.

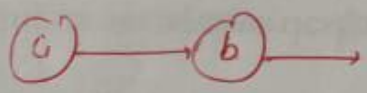


(2) Another approach : Assign timestamps to events.

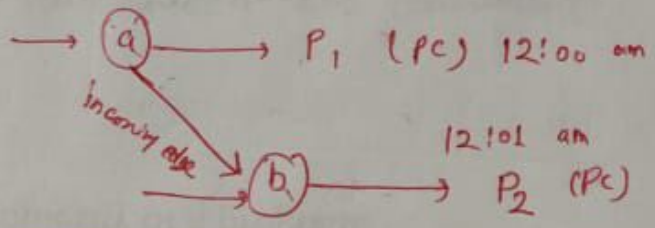
(3) As long as these timestamps obey causality, this would work.



Causality concept is based on **Happen Before Relationship.**



P (Pc) $TS(a) < TS(b)$



$TS(a) < TS(b)$

sender (timestamp) < receiver (timestamp)

Properties derived from Happen Before Relationship

1) Transitive Relation
2) Causality Ordered Relation
3) Concurrent Event

if

$Ts(a) < Ts(b)$

$Ts(b) < Ts(c)$

Then

$Ts(a) < Ts(c)$

} transitive Relation

$a \rightarrow b$ (dependency)

if 'a' is happening before 'b' then if there is any changes in 'a' then it will effect 'b'.

} causally ordered relation

$A \rightarrow B$

A is not before B,

$B \rightarrow A$

B is not before A,

$A \parallel B$

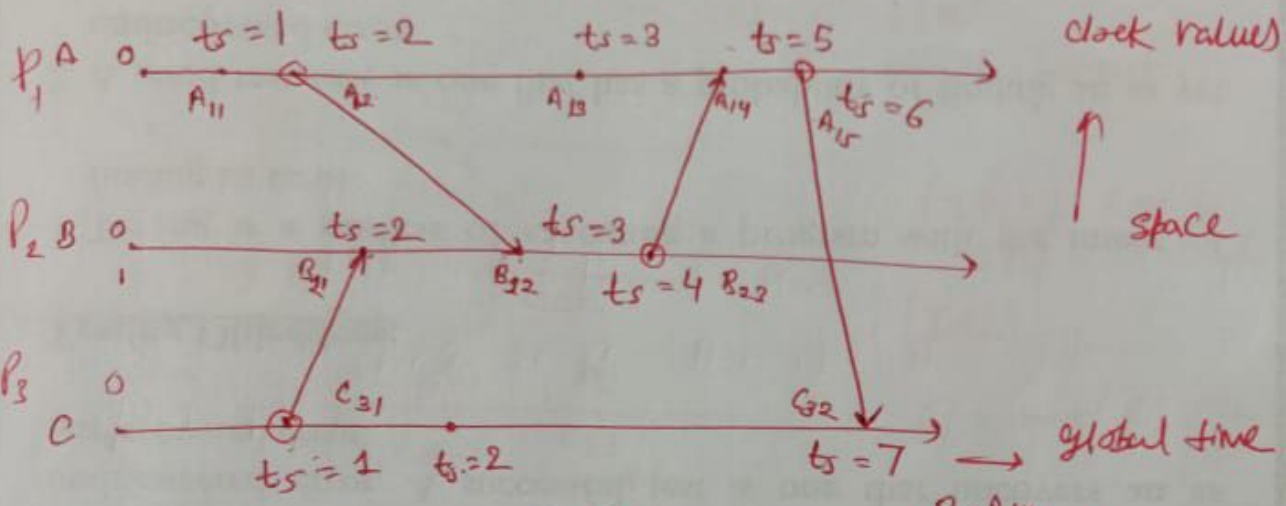
A and B are concurrent.

} concurrent event

Lamport Logical Clock

$ts = \max(\text{own timestamp}, \text{sender } ts) + 1$

$ts = \text{timestamp} + 1$



$P_2 \to B_{21}$ Max $(0,1)+1$
$= 1+1$
$= 2$

$P_2 \to B_{22}$ Max $(2,2)+1$
$= 3$

$P_1 \to A_{14}$ Max $(3,4)+1$
$= 5$

$P_3 \to C_{32}$
Max $(ts\ 2, 6)+1$
$= 7$

Algorithm → Each process uses a local counter which is an integer. the initial value of counter is '0'.

→ Process increments its counter when A send or receive instruction takes place.

→ The counter is assigned to the event as its timestamp.

→ The send event carries its time stamp.

→ For receive event counter is updated as

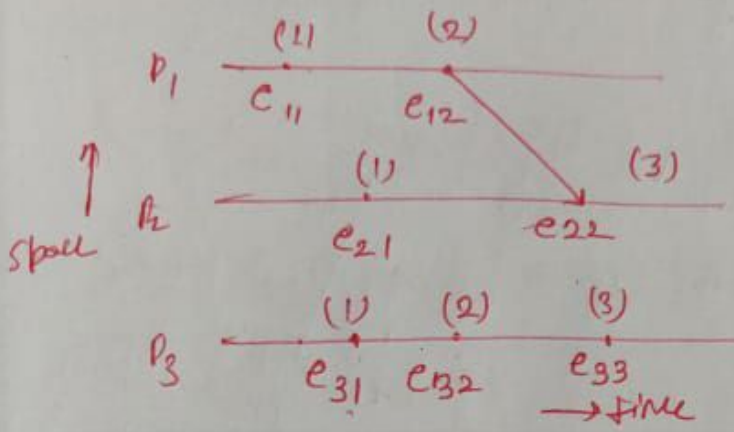
Max (Local clock, Msg timestamp (sender ts)) $+1$

Vector Clock

Limitation of Lamport clock:

if $a \rightarrow b$ $\quad C(a) < C(b)$ true

if $a \rightarrow b$ $\quad C(a) < C(b)$ may be or not.

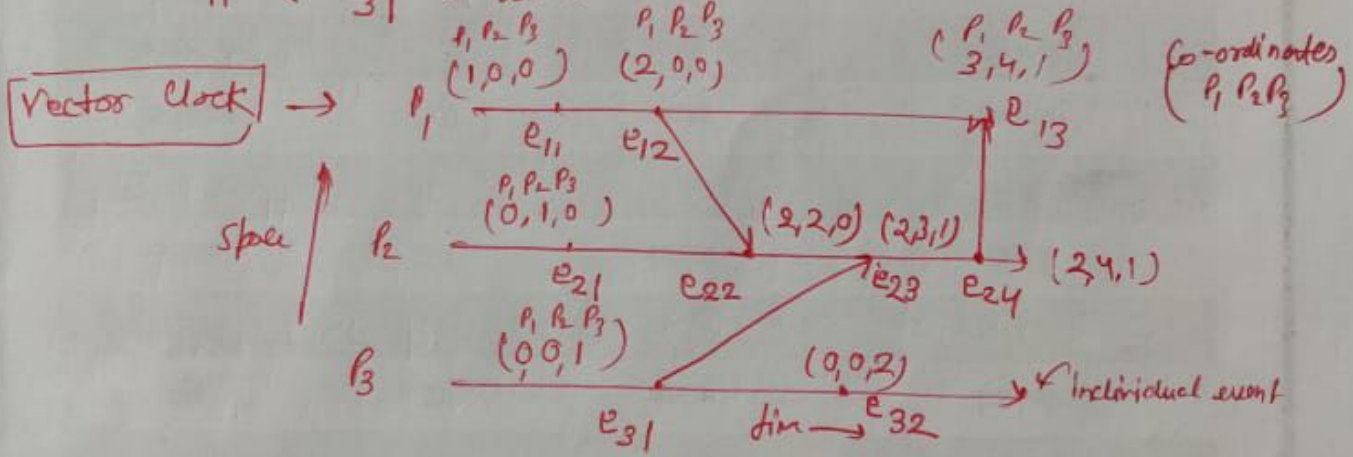


[IR1] if $a \rightarrow b$
$C(b) = C(a) + \alpha$

[IR2] if
$C_j = \max(C_j, t_m + \alpha)$

$e_{11} < e_{31}$ ← limit

Vector Clock →



$P_2 \rightarrow e_{22}$ (incoming edge) $\{2,0,0\}$ $\quad$ $P_2 \rightarrow e_{23}$ $\{2,3,0\}$

$\{0,2,0\}$ $\quad$ $\{0,0,1\}$

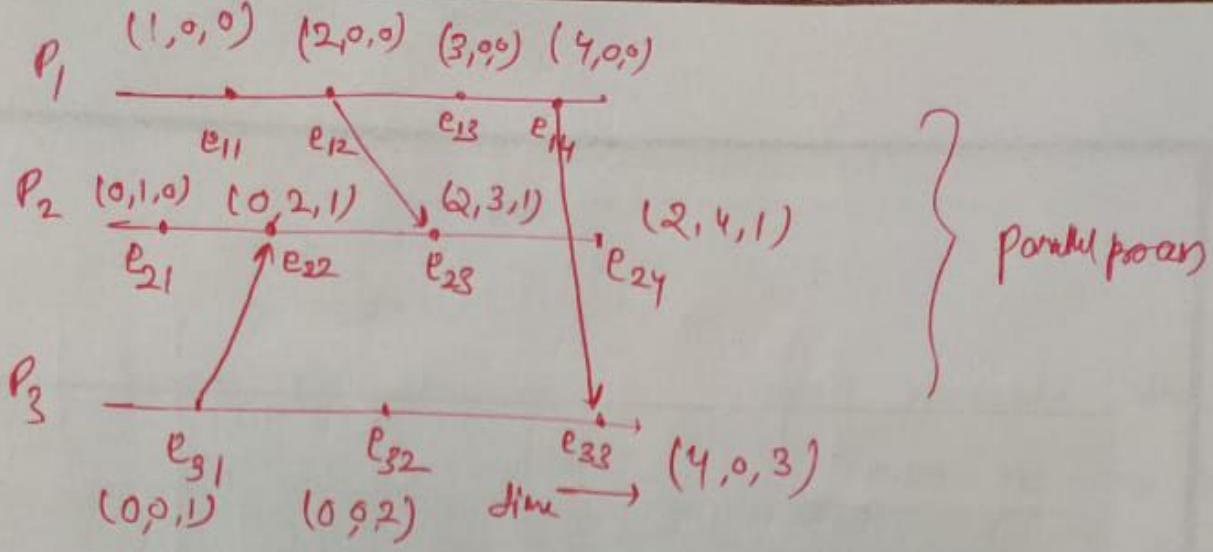
$\Downarrow$ $\quad$ $\Downarrow$

max $\{2,2,0\}$ $\quad$ max $\{2,3,1\}$

$P_2 \rightarrow e_{24}$ $\{2,4,1\}$

$P_1 \rightarrow e_{13}$ $\{3,0,0\}$
$\{2,4,1\}$ > max $\{3,4,1\}$

Q→



$P_2 \rightarrow e_{22}$ Max $\{(0,2,0)\ (0,0,1)\} = \{0,2,1\}$

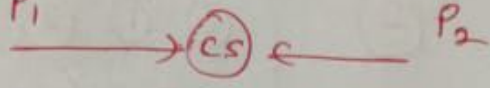
$P_2 \rightarrow e_{23}$ Max $\{(0,3,1)\ (2,0,0)\} = \{2,3,1\}$

$P_2 \rightarrow e_{24}$ Max $\{2, 4, 1\}$

$P_3 \rightarrow e_{33}$ Max $\{(4,0,0)\ (0,0,3)\} = (4,0,3)$

[UNIT 2]

DISTRIBUTED MUTUAL EXCLUSION



When a process is accessing a shared variable, the process is said to be in a CS, so no two process can be in the same CS at the same time. this is called Mutual Exclusion.

Distributed Mutual Exclusion -

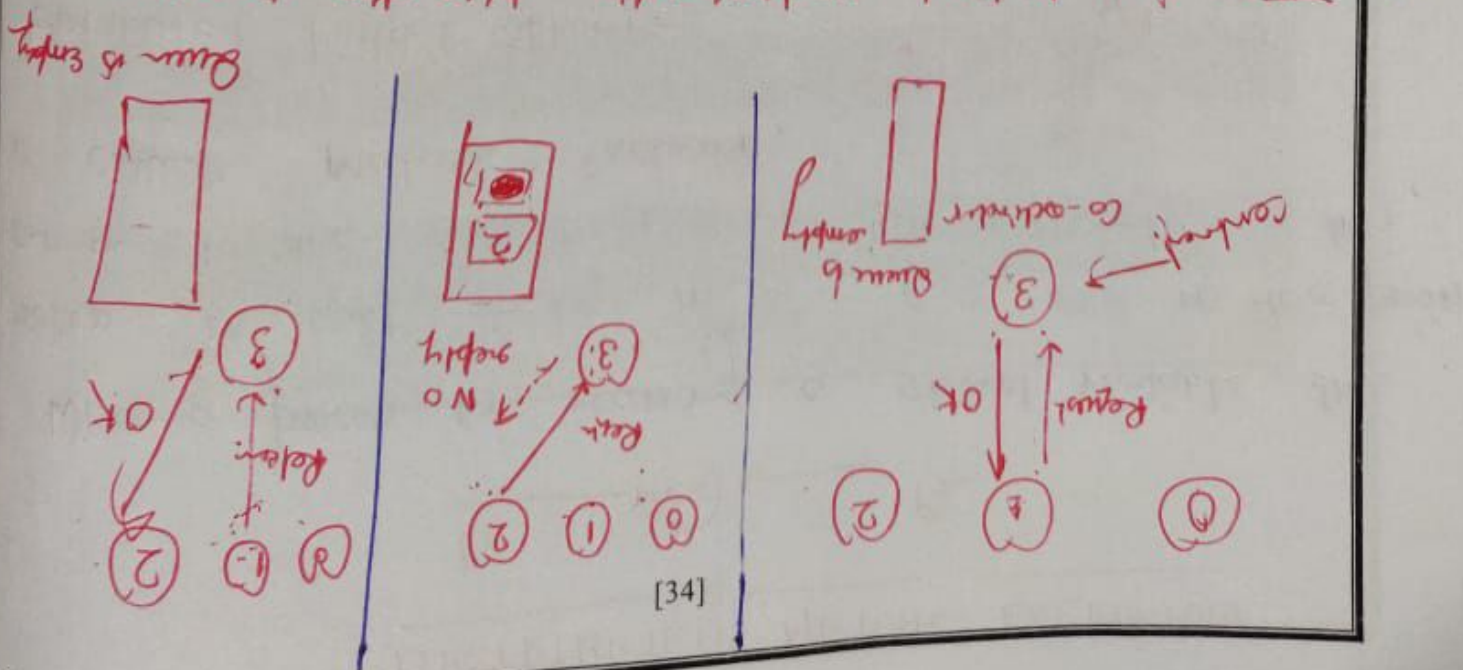
- Assume there is agreement on how a resource is identified.
  → pass identifier with request
- Create an algo. to allow a process to obtain exclusive acces to a resource.
- The Diff. algo. based on Message passing to implement Mutual exclusion in distributed system are
  1. Centralized Algo.
  2. Token Ring algo,
  3. Distributed Algo.
  4. Decentralized Algo.

CENTERALIZED ALGORITH

→ One process is elected as the co-ordinator.
→ Whenever a process wants to acces a shared-resource, it sends request to the coordinator to ask for permission.
→ Co-ordinator may queue request.

Column Name	Size	Data type	Constraint
Name	50	varchar	Primary Key
Emailid	30	varchar	Not Null
Mobileno	10	Bigint	Not Null
Message	500	varchar	Not Null
Date		Timestamp	Not Null

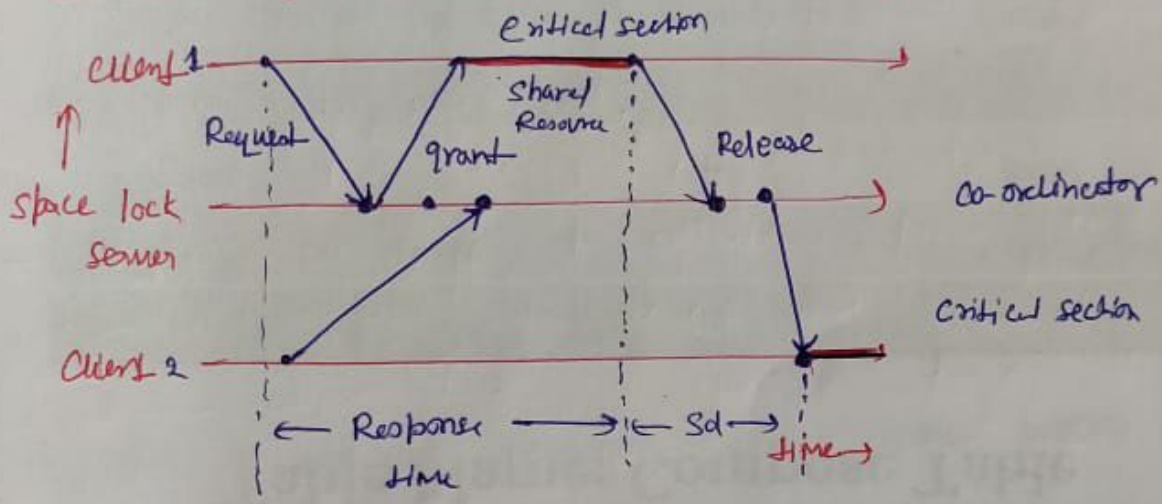
Table Name: Review Table



Requirement of Mutual Exclusion Algo-

1) Only one request accessen the CS at a time.
2) Freeelom from deadlocks
3) " " Starvation
4) Fairness
5) Fault tolerance.

Performance of a Mutual Exclusion Algo-

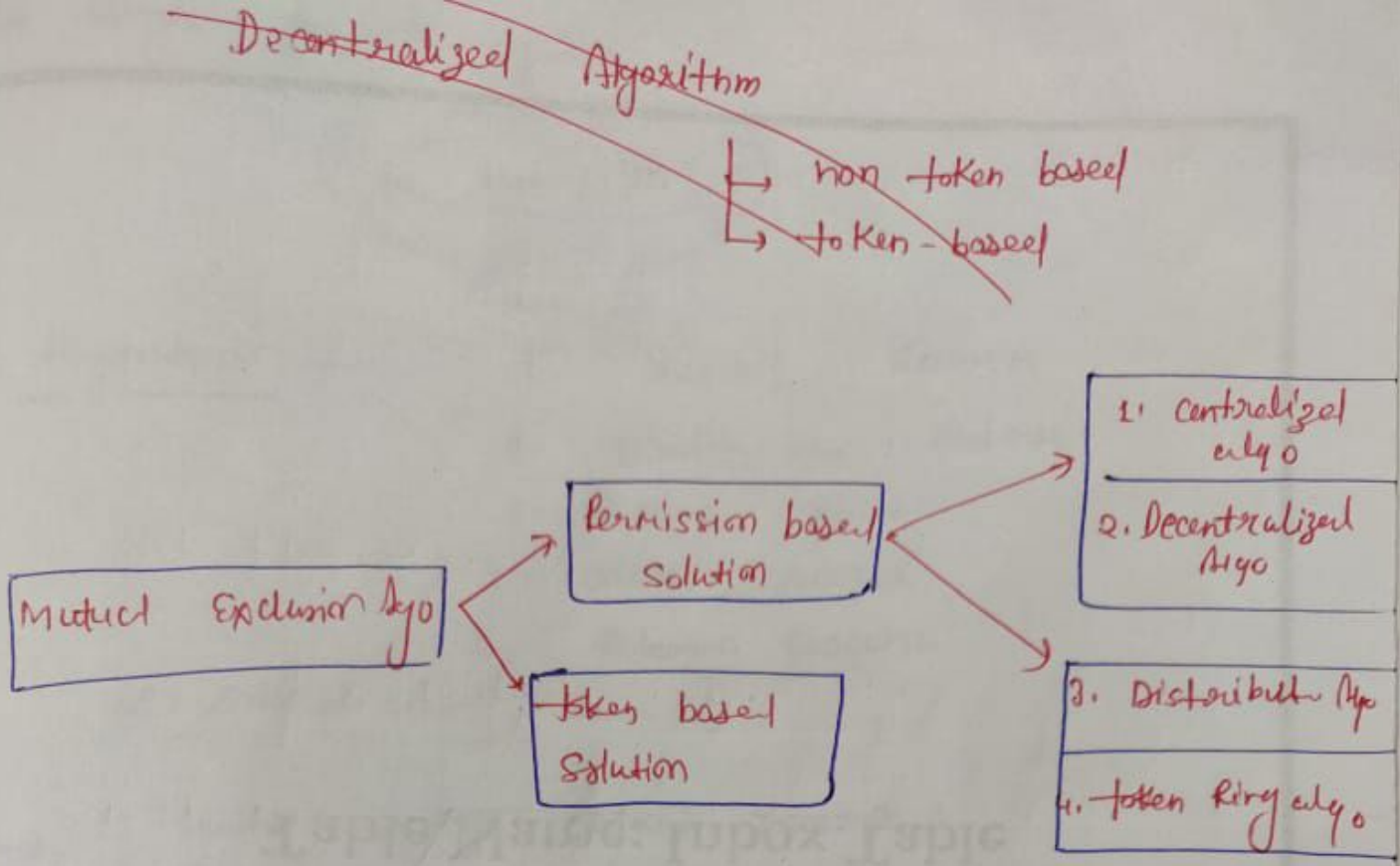


System throughput S (rate at which the system execute request for the CS)

$$S = \frac{1}{Sd + E}$$

Sd = synchronization delay
E = avg execution time

→ low load & high load performance can be calculated.
→ best & worst case perform can be calculated, if fluctuate statistical is there, take the average.



Example- one Printer 10 systems,

Token Based Solution

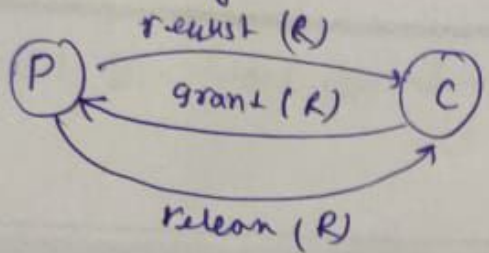
→ Pass a special message to all other process known as "token".
→ Only one token is available.
→ Whoever has that token is allowed to access the shared resource.
→ Drawback:- if token is lost (it will take time to create new token).

Permission Based solution

1. Centralized Algorithm

→ One process is elected as a coordinator and it will grant or reject to access the resources.

→ Single processor system.



Algorithm — 1. Request Resource
2. Wait for Response
3. Receive grant
4. access resource
5. release resource

C: Critical sect / Resource (R)

→ if another process claimed resource:
Maintain Queue. (FIFO) service order.

Benefits: Its fair, and all requests processed in order.
Easy to implement, understand, verify.

Drawback: If coordinator failed / crashed, system will be stopped.

2. De-Centralized Algorithm

→ It is solution of Centralized algorithm.
→ To access the shared resource, process will require the majority to vote Co-ordinator.
→ All process will give their concern/respond to access resource.
→ Winning / recommended process will access resource firstly.

Distributed Algo (Ricart + Agrawala Algorithm)

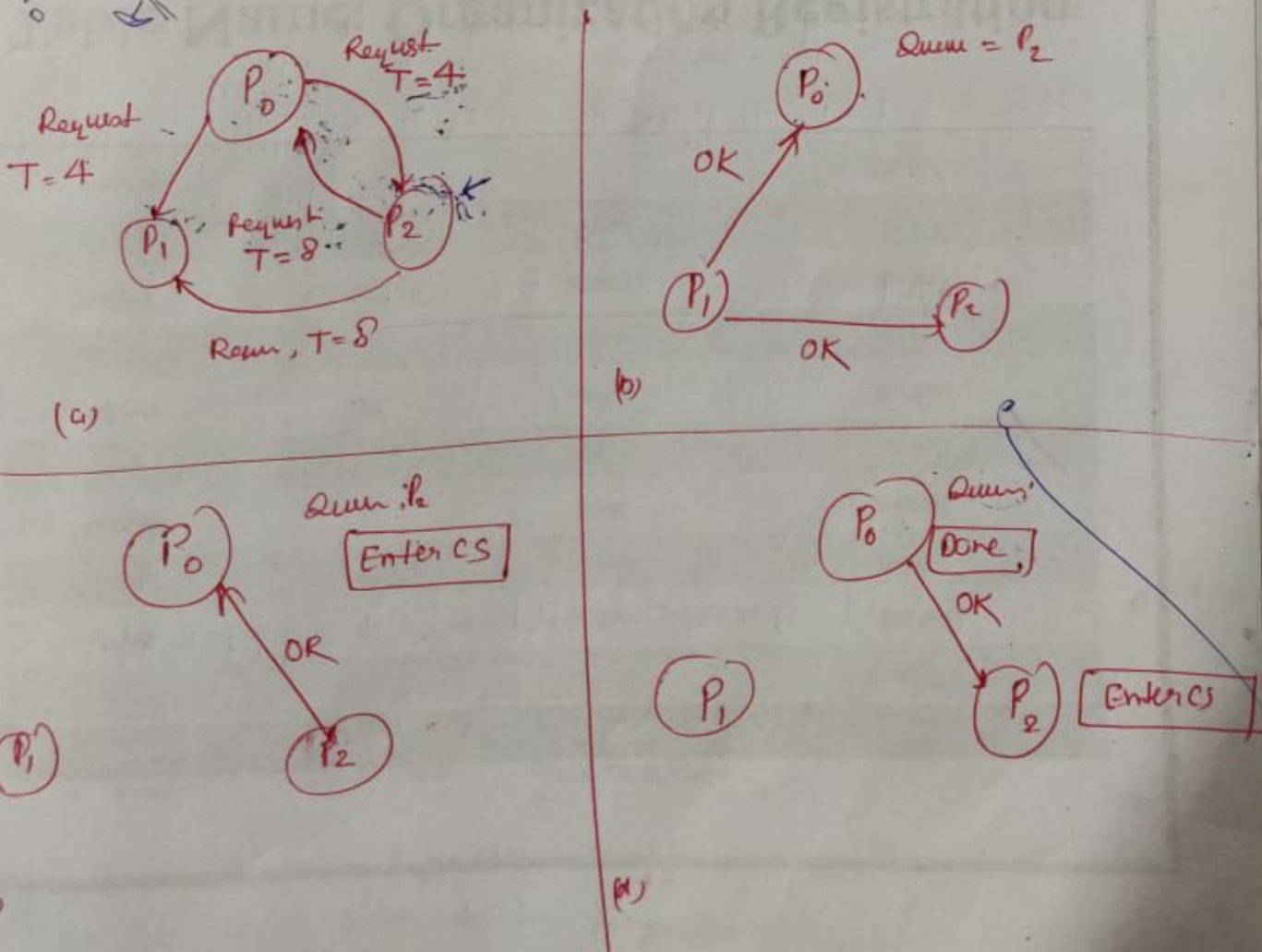
→ There should be ~~total~~ ordering of all events in the system.

→ To access Resource, process will create a message that contain:
- identifier (machine ID, process ID)
- Name of resource
- Timestamp (~~totally~~ ordered Lamport) / current time.

→ Send request to all processes in the group including itself.

→ wait until response of all process / everyone gives permission.

→ The process will have lowest time will be allowed to access critical section / shared resources.



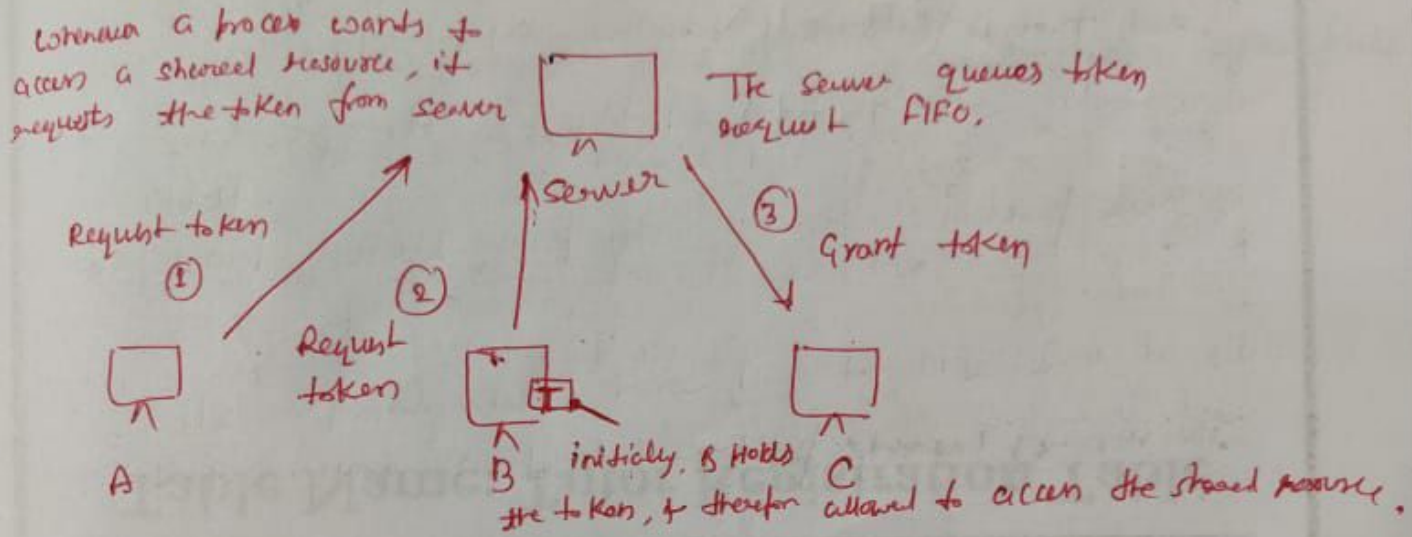
4. Token Ring Algorithm

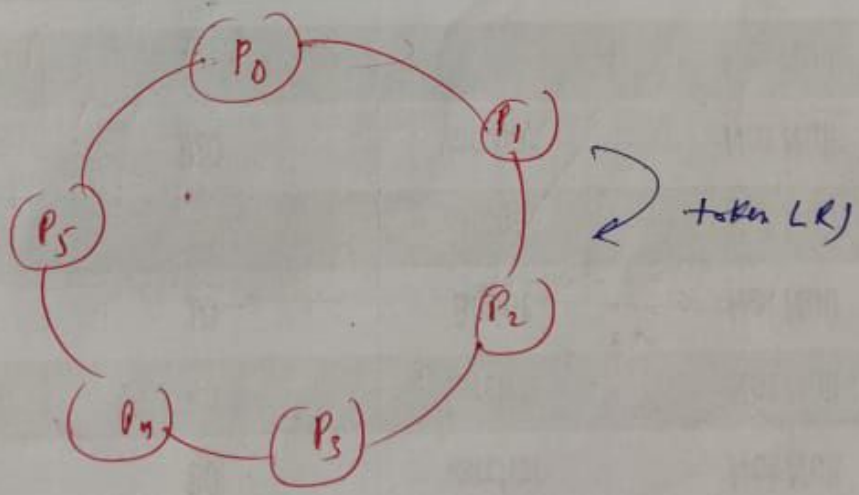
"Whenever a process want to access a Resource, it request for token (appointment) from server."

→ server maintain queue of request & execute each request in FIFO manner.

→ construct logical ring in software.

→ Process communicates with neighbor.





Algorithm —

Initialization

- Process gets token for resource (R).

Only one process will have token at a time.

- Mutual exclusion guaranteed.

If token is lost (process died)

- It will have to be regenerated.

Token circulates around ring

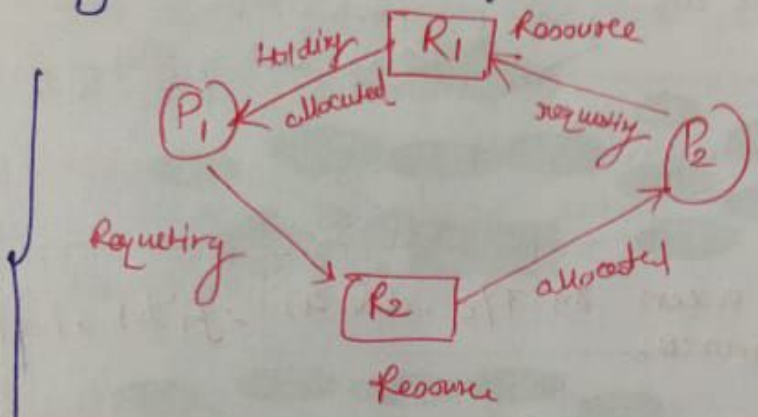
- From P_i to P_{i+1} Mod N

When process acquires token

- Check to see, if it needs to utilize the resource /enter CS.
- if no, send token to neighbor.
- if yes, access R, resource
- Hold token until done + release token after finish

DEAD LOCK

It is a situation when 2 processes sharing the same resource are effectively preventing each other from accessing the resources.

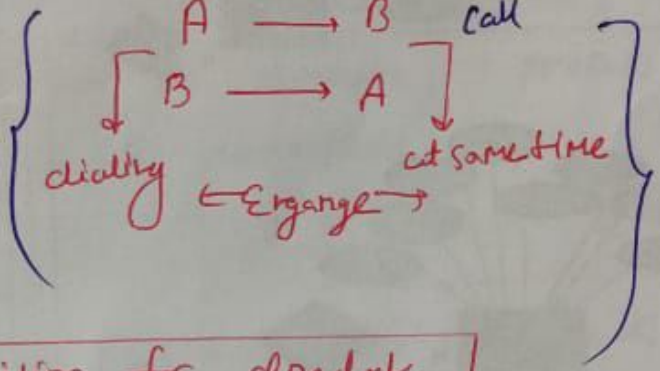


$P_1 \longrightarrow R_1$

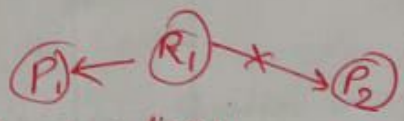
$P_2 \longrightarrow R_2$

P_1 is holding R_1, until P_1 releases R_1, P_2 can't access.

P_2 " " R_2 " P_2 " " R_2, P_1 " "



Condition for deadlock :-

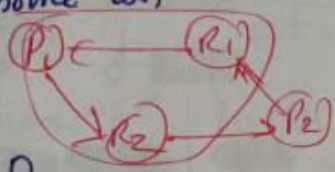


(1) – Mutual exclusion → Not Shareable (only one P_0 can use R_1 at a time)

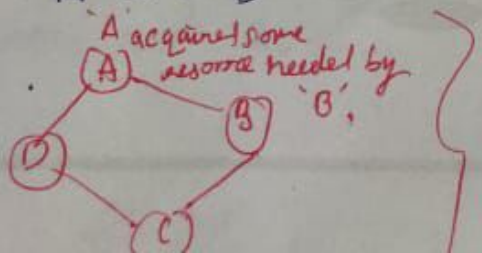
(2) – Hold + wait → Holding on resource + waiting for another to come.

(3) – No preemption → Resource can't be released voluntarily by itself.

(4) – circular wait



↳ A B C D



Types of Deadlock in Distributed system

① **Resource Deadlock** : It uses 'AND' condition.

AND condition - A Process that requires resources for execution, can proceed when it has acquired All those resources".

Example - If a process require 3 resources, it will execute when it will have all 3 resources.

② **Communication Deadlock** : It uses 'OR' condition.

OR condition - A process that requires resources for execution can proceed when it has acquired - At-least one resource.

Example - To execute, process must have at least 1 resource.

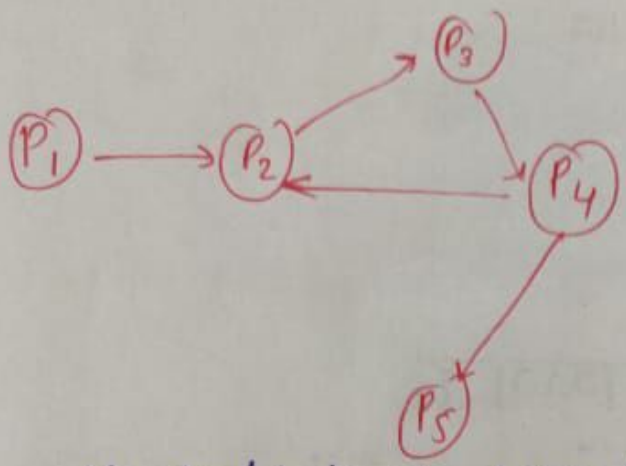
Deadlock Conditions:

→ The condition for deadlock in a system using the AND condition is the "existence of a cycle".

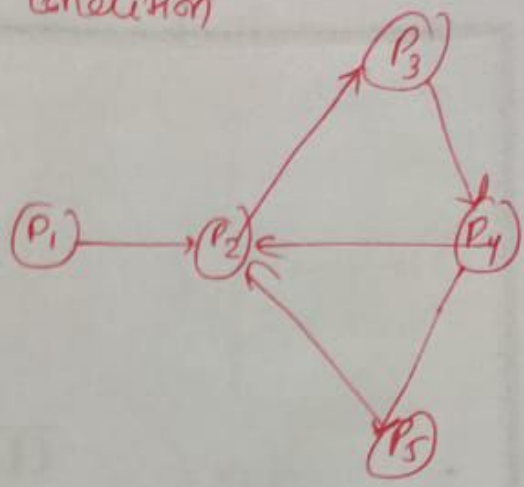
→ The condition for deadlock in a System using the OR condition is the "existence of a knot".

→ Knot (K) : It consists of a set of nodes such that for every "node a" in K, all nodes are reachable from node a.

Example : 'OR' condition



No deadlock (One + No cycle)



Deadlock (One to cycle)

Distributed System Deadlock Detection

Handling Deadlock in D.S —

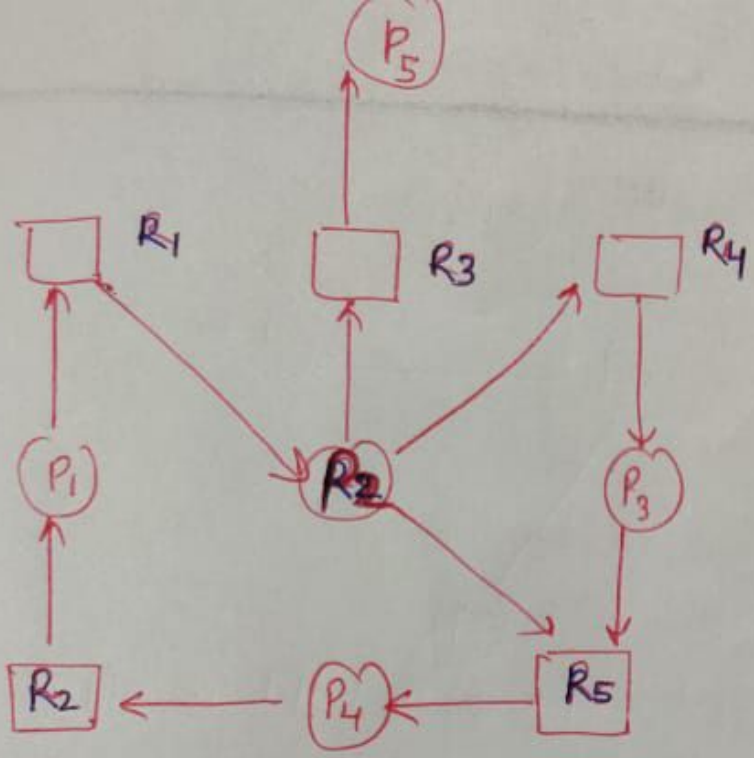
- Ignorance
- Avoidance
- Prevention
- Detection & recovery

(1) Use wait for Graph (WFG)

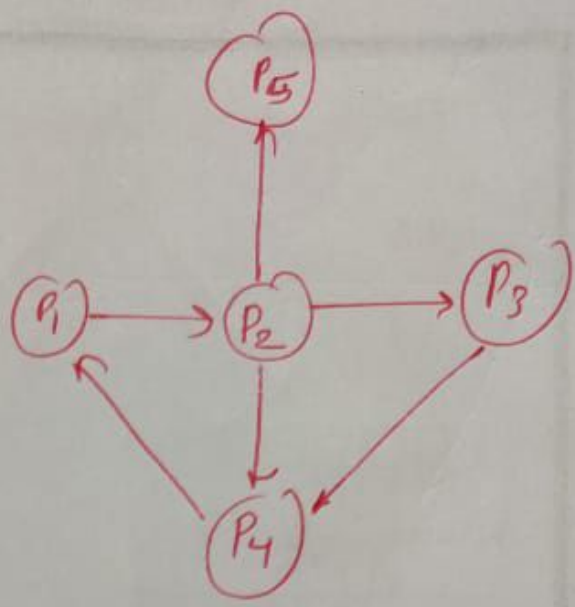
→ All nodes are processes (threads).

→ Resource allocation is done by a process (thread) Sending a request message to another process (thread) which manages the resource (client-server) communication Model, RPC Paradigm).

→ A system is deadlocked, IF there is a directed cycle (or Knot) in global WFG.



(Resource allocation graph)



(corresponding Wait for graph)

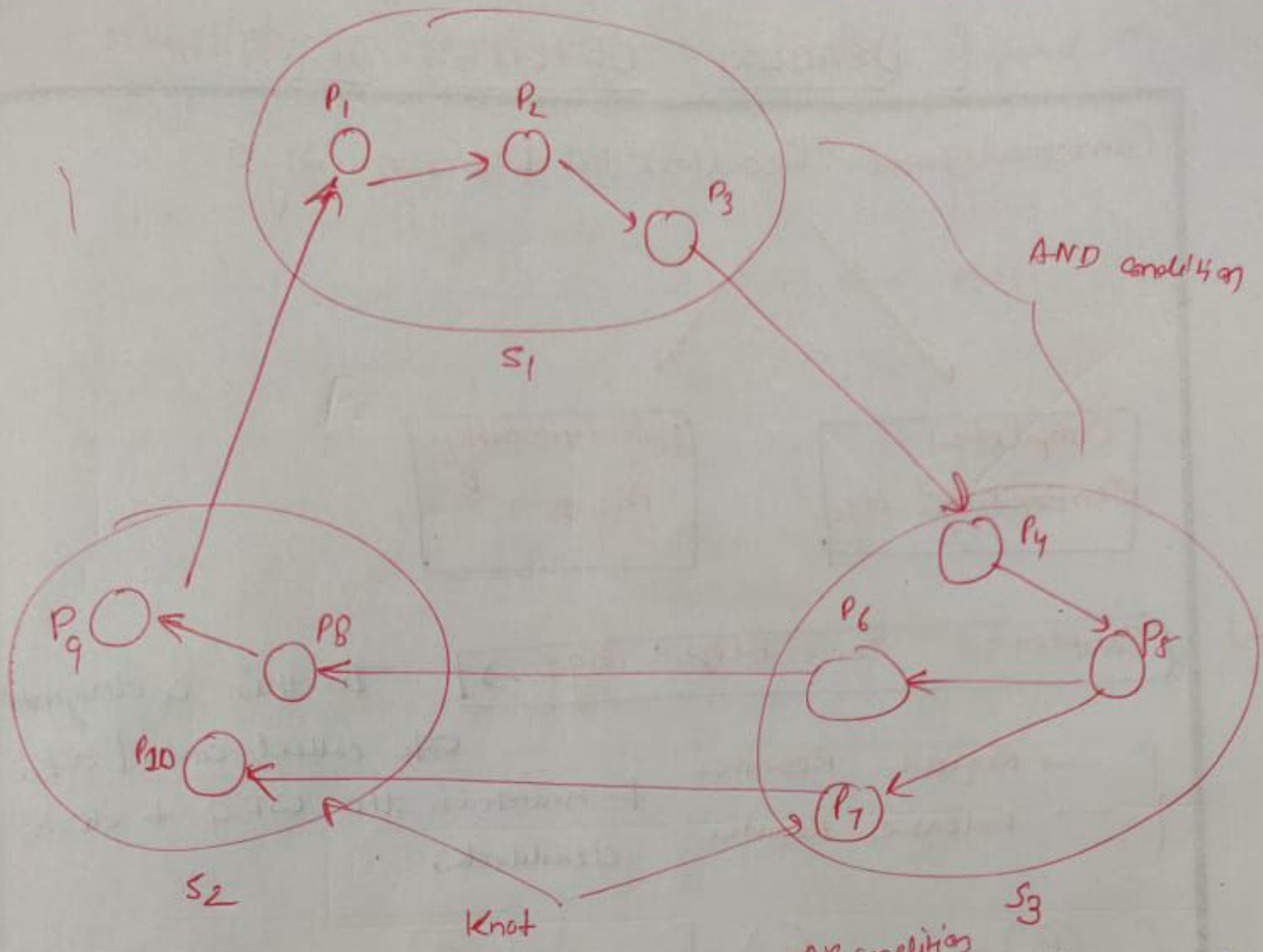
DS Deadlock Detection, Cycle vs. Knot

The AND Model requires all resources currently being requested to be granted to un-block a computation

> A cycle is sufficient to declare a deadlock with this Model.

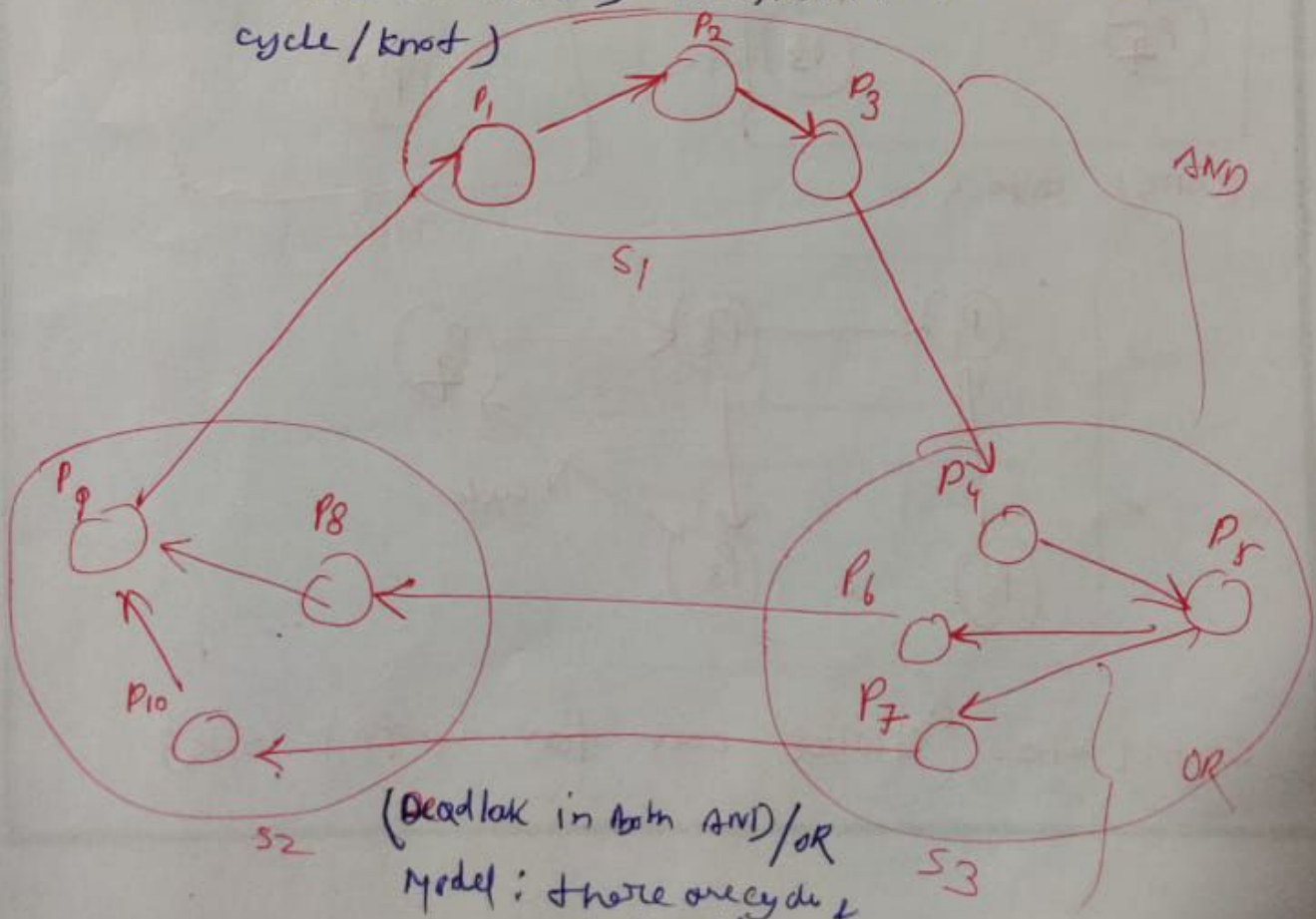
The OR Model allows a computation making Multiple different resource requests to unblock as soon as any are granted.

- A cycle is a necessary condition
- A knot is a sufficient "



OR condition

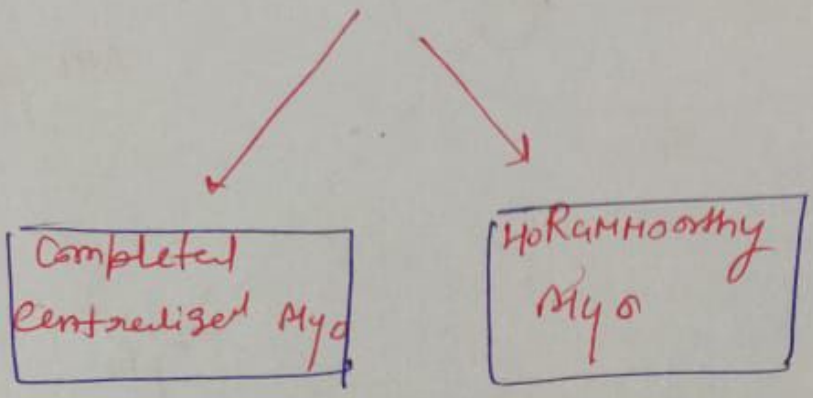
(Deadlock in the AND) Model : there is a cycle but no knot, OR, Model : No deadlock (Due to no cycle / knot)



(Deadlock in both AND)/OR Model : there are cycle +

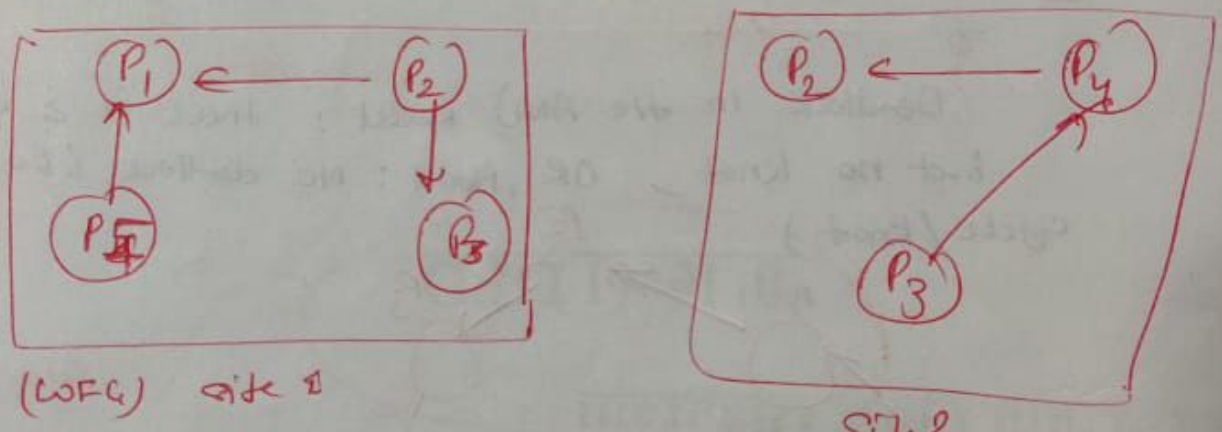
[DEADLOCK DETECTION ALGORITHM]

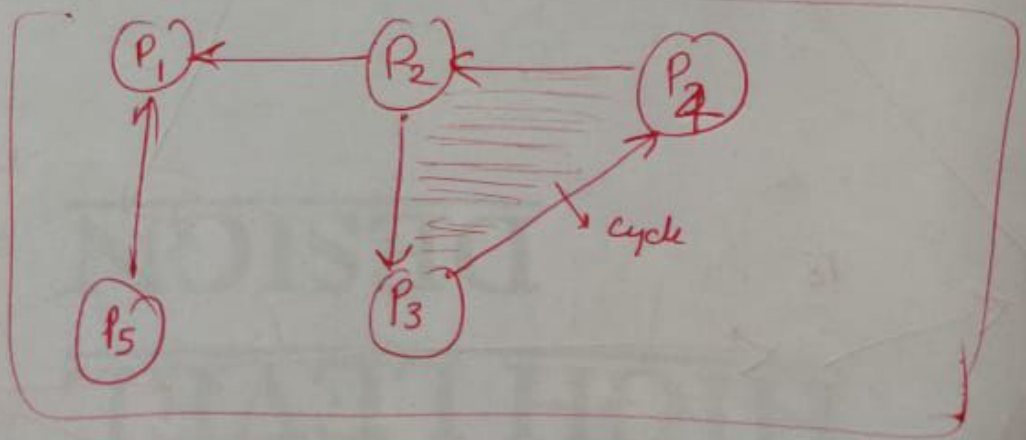
Centralized Deadlock [23] Detection Algo



(1) **Completed centralized Algo →** In this a designated site called control site. It maintain the WFG to check deadlock.

- → Request Resource
- → Release Resource





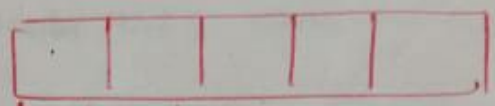
control site (Global wait for Graph) site 3

→ Drawback is that everything is depend on control site.

(2) Ho-Ramnoorthy Algo → two-phase
→ One-phase

(i) Two-phase algo –

$R_1 \{S_1 - - S_n\}$

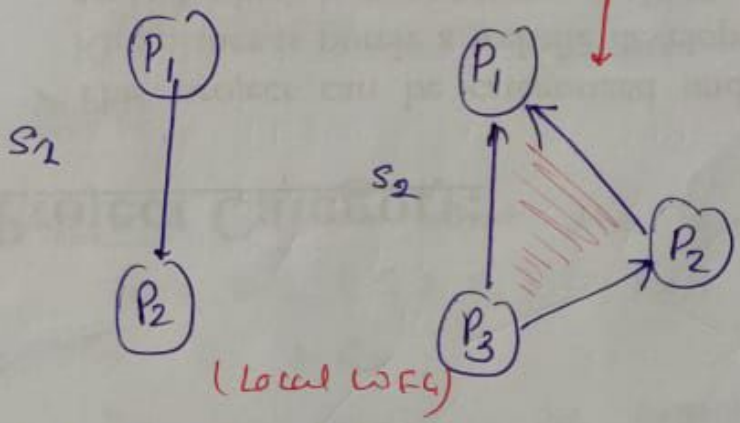
Site → 

(Status table)

↓

WFG

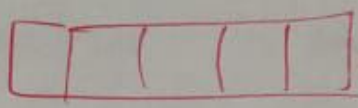
↓

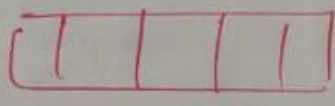


(Local WFG)

X No Global WFG

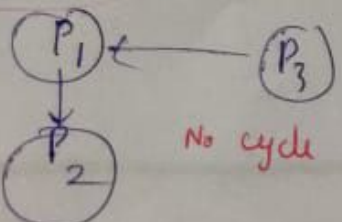
(ii) One Phase algo

Site →  ← Resource Status

 ← Process Status

$R_1 \leftarrow P_1$ (RS)

$P_2 \leftarrow P_1$ (PS)

Ex- 

No cycle

Fully- Centralized approach -

In this, the local co-ordinators send local state information to the central coordinator in the form of messages. One of the following Methods is used to transfer info. from local co-ordinators to the central coordinator.

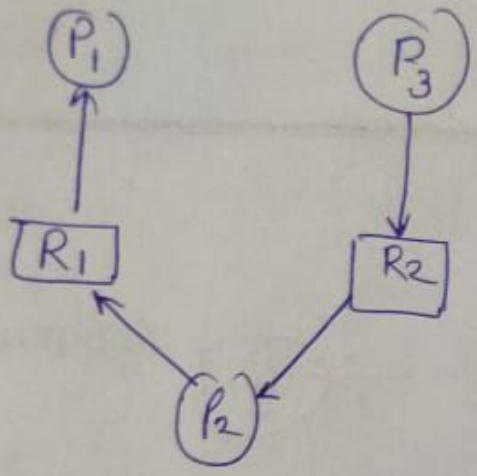
(1) Continuous transfer - A local co-ordinator sends a message promptly the update done in the local WFG whenever a new edge is added to or deleted from it.

(2) Periodic transfer - To reduce the no. of msgs, a local co-ordinator periodically (when a number of changes have occurred in its local WFG) sends a list of edges added to or deleted from its WFG since the previous msgs was sent.

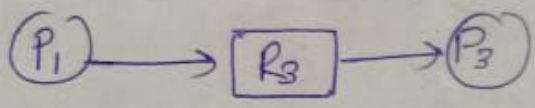
(3) Transfer-on-request - send msg only when the central co-ordinator makes a request for it.

Example - Suppose that the system is comprised of only two site (S_1 & S_2) with S_1 having two resources R_1 & R_2, S_2 is having one resource R_3. Also suppose that there are three processes (P_1, P_2, P_3) that competing for the three resources in the following manner:

- P_1 is holding R_1 and requesting for R_3.
- P_2 " " R_2 and " " R_1.
- P_3 " " R_3 " " " R_2.

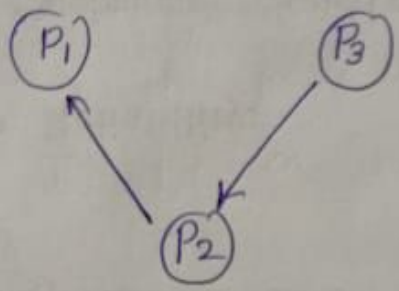


site S_1



site S_2

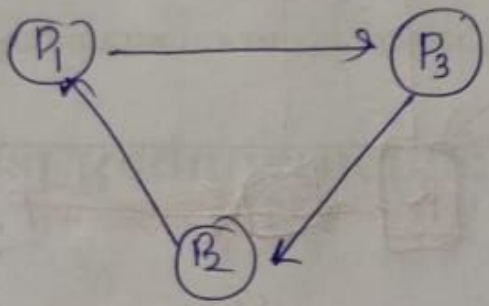
(Resource Allocation Graph of each site)



site S_1

site S_2

(wait for graph of each site)



global WFG by taking the union of the local WFGs.

Let us consider the same Example, suppose that the three processes (P_1, P_2, P_3) compete for the three resources (R_1, R_2, R_3) in the following manner;

Step 1 : P_1 requests for R_1 and R_1 is allocated to it

Step 2 : P_2 " " R_2 " R_2 " "

Step 3 : P_3 " " R_3 " R_3 " "

Step 4 : P_2 request for R_1 and waits for it.

Step 5 : P_3 " " R_2 " " " "

Step 6 : P_1 releases R_1 and R_1 is allocated to P_2.

Step 7 : P_1 requests for R_3 and waits for it.

Assuming that the method of continuous transfer is employed by the algo., the following sequence of messages will be sent to the central co-ordinator:

m_1 : from site S_1 to add the edge (R_1, P_1)

m_2 : from site S_1 to add " " (R_2, P_2)

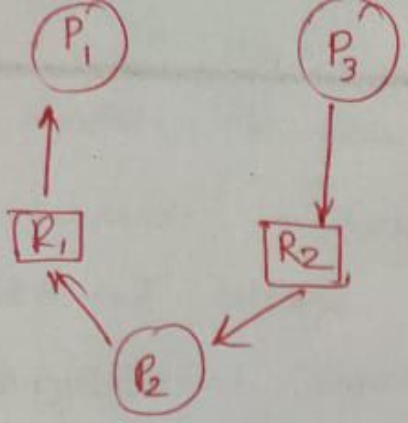
m_3 : " " S_2 " " " (R_3, P_3)

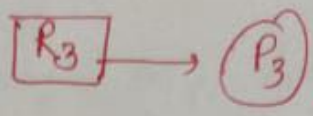
m_4 : " " S_1 " " (P_2, R_1)

m_5 : " " S_1 " " (P_3, R_2)

m_6 : from site S_1 to delete edges (R_1, P_1) and (P_2, R_1), and add edge (R_1, P_2)

m_7 : from site S_2 to add edge (P_1, R_3)





S1 (RAG)

fig. 1

S2 (RAG)

fig. 2

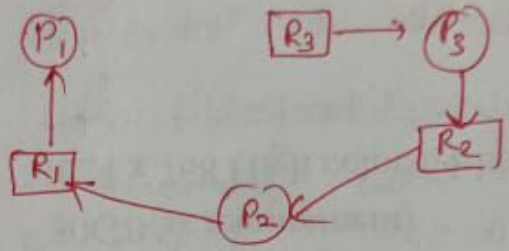
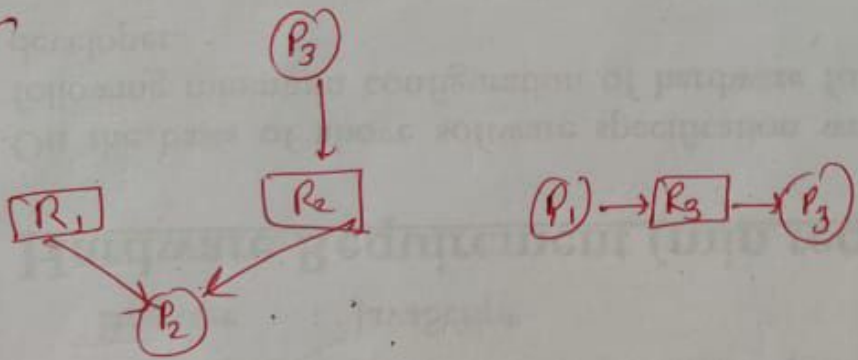


fig. 3 (Global RAG Maintained by central co-ordinator)

(Resource allocation graphs after step 5:)



S1

fig. 4

S2

fig. 5

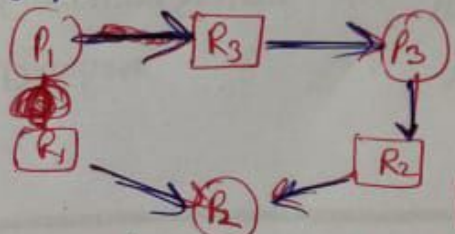


fig 6 (central coordinator)

(resource allocation graph after step 7)

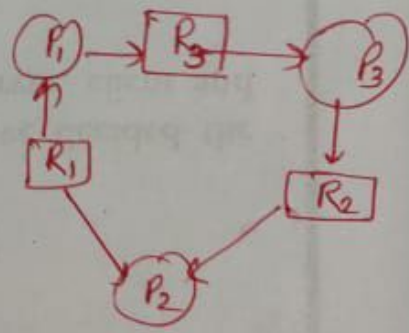


fig 7 (central coordinator)

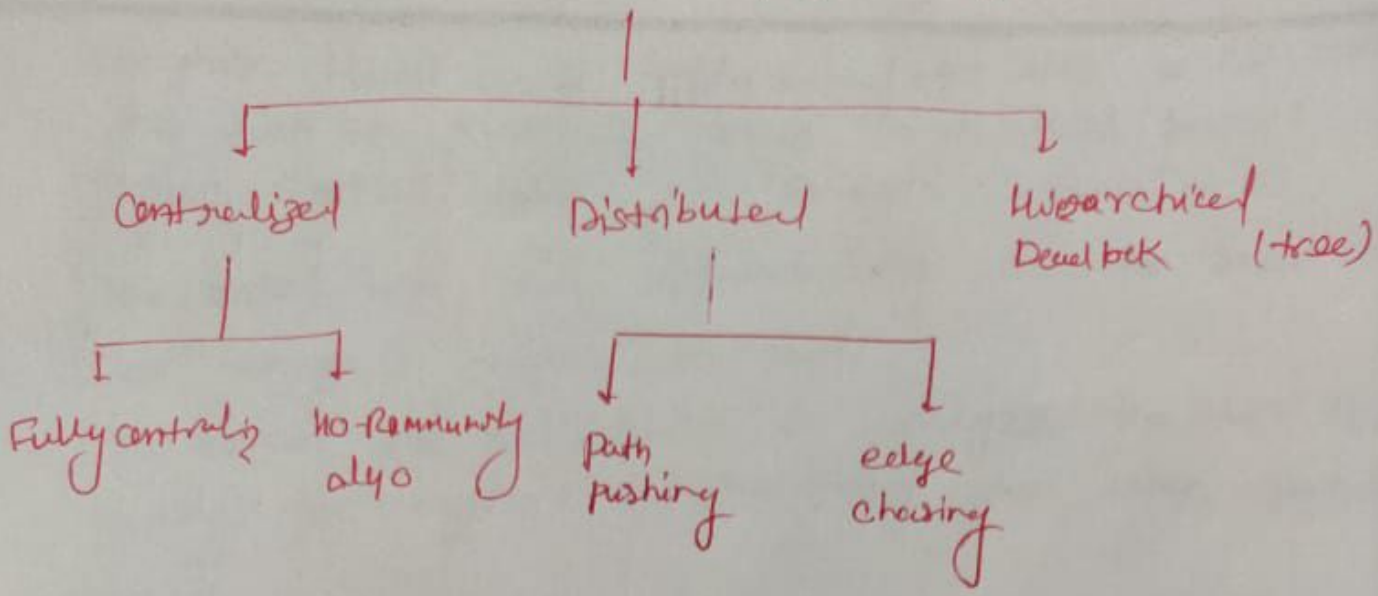
(RAG of the central coordinator showing false deadlock if message m_7 is received before m_6 by the central coordinator.)

→ In Fig 6, thus no cycles, indicating that the system is free from deadlock.

→ However, suppose the message m_7 from site S_2 is received before message m_6 from S_1 by the central coordinator. In this case, The central co-ordinator will incorrectly conclude that a deadlock has occured, & may initiate deadlock recovery actions.

→ one method to avoid the detection of false deadlock is to use Lamport's algo. & attach a unique global timestamp with each message.

Deadlock Detection Control



~~Distsit~~

The Ho-Ramamoorthy Algorithms

(1) **Two-phase Model** – It can be for AND or OR model.

→ In this, each site has a status table of locked & waited resources.

→ The control site will periodically ask for this table from each node.

→ The control node will search for cycles & if found, will request the table again from each node.

→ Only common information in both reports will be analyzed for confirmation of a cycle.

(2) One-phase Model —

→ One-phase Model can be implemented for AND or OR model. In this each site keeps 2 tables for all local process:
→ 'Process status' table, 'Resource status' table.
→ The control site will periodically ask for these tables (via message) from each node.
→ The control site will build & analyze the WFG, looking for cycles & resolving them when found.

Deadlock detection Algo. in Distributed System

(1) Path-Pushing Algorithm — (WFG-Based Distributed Algorithm)

→ As in centralized & hierachical approaches, here each site maintains its own local WFG.

→ In this, Path information sent from waiting node to blocking node.
→ This approach deals with the 'transactions'.
→ In this all transactions are totally ordered or sequentially.
→ Each transaction may have sub transactions but they executes in sequentially manner.
→ Path information sent from waiting node to blocking node.
→ In this, WFG constructed by disseminating dependency sequences:
(i) Each node builds a WFG based on local info & info from other sites.
(ii) Detect and resolves local deadlocks.

(iii) Transmits to other site deadlock information in form of waiting path.

imp

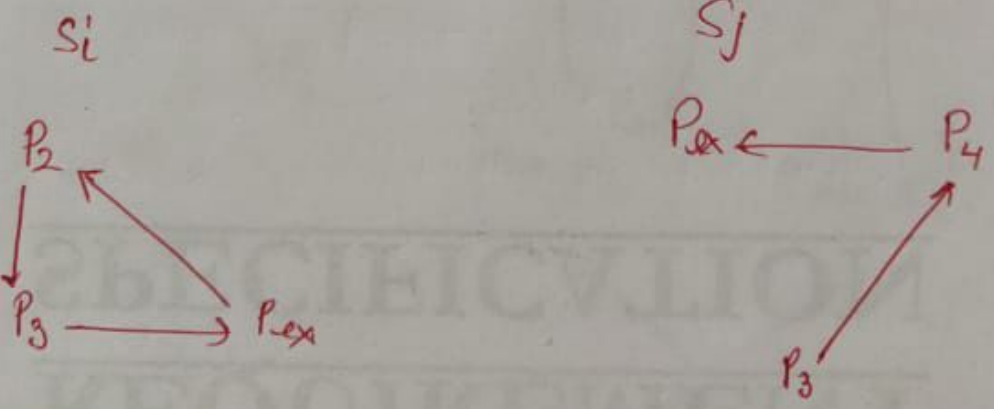
→ In this algorithm information about the wait-for-dependencies is propogated in the form of paths.

→ In this, each site maintains its local WFG. It include node 'Pex' (external node of perticular process).

$P_i \longrightarrow P_{ex}$

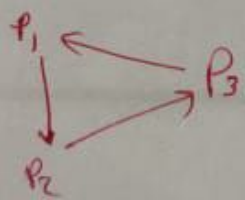
is waiting for resource held by P_{ex} (another site process)

$P_{ex} \longrightarrow P_j$



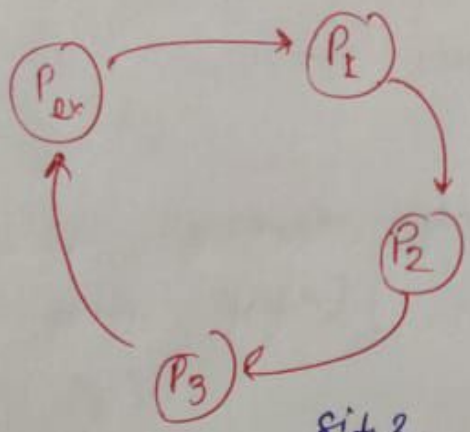
Algorithm –

(1) if local WFG contains cycles that does't involves P_{ex} thus syster is in deadlock.

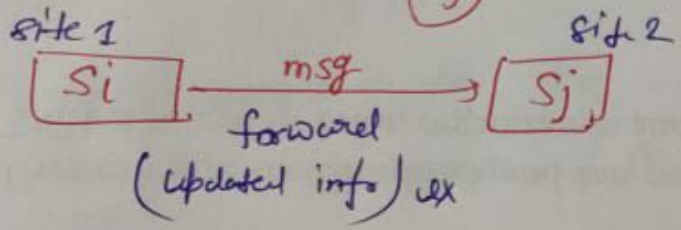


(2) If P_{ex} exist suppose

$$P_{ex} \longrightarrow P_1 \longrightarrow P_2 \longrightarrow P_3 \longrightarrow P_{ex}$$

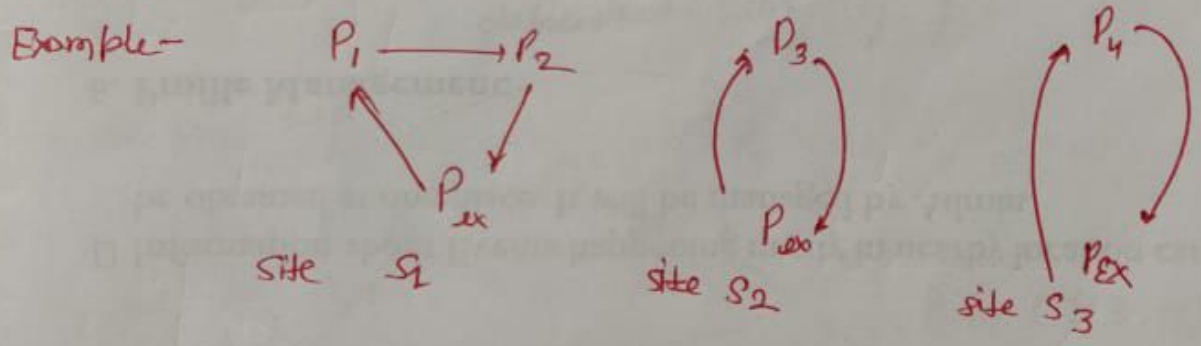


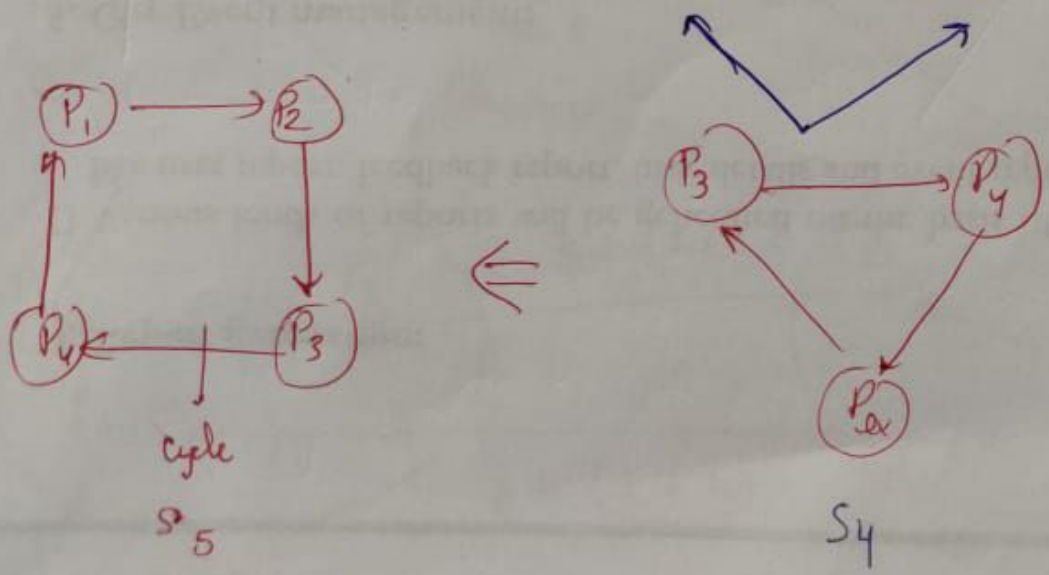
(3)



(4) S_j form WFG with new infor. to detect whether a system is in deadlock or not.

Example-





(2) Edge chasing Algorithm — (Chandy-Misra-Haas Algo)

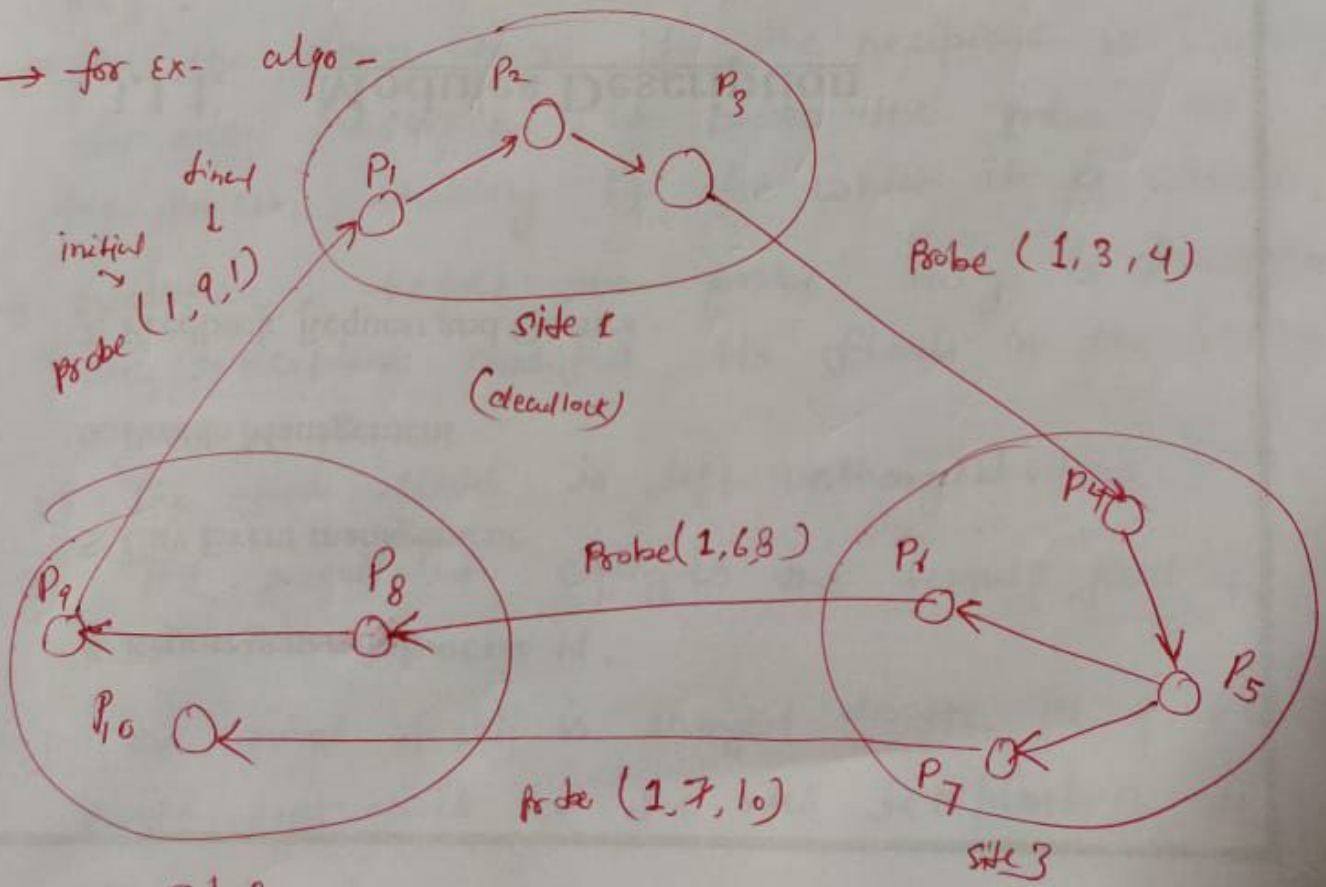
→ In this algo. uses a special msg called 'Probe'. A Probe is a triplet (i, j, k) denoting that it belongs to a deadlock detection initiated for process P_i, P_j, P_k.

→ In this we construct global TWF graph. (transaction wait for graph).

→ Process (P_i, P_j, P_k)
P_i is dependent on P_k, if there exists a sequence of processes $\{P_i, P_{j1}, P_{j2}, P_{j3}, P_k\}$

→ The system maintain a boolean array like dependent j { if dependent j (i) means P_i is dependent on P_j }

→ for ex- algo -



→ Probe mssg contains :

Probe (i, j, k)

i → The identifier of the process just blocked.

j → The " " " " sending this msg.

k → " " " " to whom this msg is being sent.

→ On receiving a probe msg, the recipient checks to see if it itself is waiting for any resource. if not, this means that the recipient is using the resource requested by the process that sent the probe msg to it.

In this case, the recipient simply ignore the msg.

→ On the other hand, if the recipient is waiting for any resource, it passes the probe msg to the process holding it for which it is waiting.

→ However, before the probe msg is forwarded, the recipient modifies its fields in the following manner :

1) The first field is left unchanged.

2) The recipient changes the second field to its own process id.

3) The third field is changed to the id of the process that will be the new recipient of this message.

Every new recipient of the probe msg repeats this procedure.

→ If the 'probe' msg return back to the original sender (the process whose Id is in the first field) a cycle exists & system is deadlocked.