DISTRIBUTED SYSTEMS

(1) Clock Synchronisation

1. Clock synchronisation is the process of aligning the clocks of multiple devices or systems to ensure that they maintain a consistent notion of time.

2. In distributed systems, various computers and nodes work together to achieve a common goal. This is where clock synchronisation is crutial to ensure that operations are correct and coordinated

3. In DS, certain processes may run concurrently on multiple computers located geographically apart from each other at larger distances.

   In such situations, to run a proces accurately across multiple node, the clocks of all nodes must be synchronised with each other.

4. To address these challenges, clock synchronisation protocols like NTP (Network time protocol) and PTP (Presision time protocol) are ~~implented~~ employed in DS. These protocols ensure that different nodes have a consistent notion of time.

5. Clock synchronisation is a crutial aspect of DS. Proper synchronised clocks enable accurate event ordering, data consistency and coordinated operations, contributed to overall correctness and efficiency of distributed applications and services

Issues with clock synchronisation

1. In real world, not two clocks are perfectly synchronised.
But some specified time difference blw clocks is tolerated.
↳ This specified time difference is a constant denoted by δ

Difference blw two clocks is called
CLOCK SKEW

and clock skew should not be greater than δ

Two clocks are said to be SYNCHRONIZED if

clock skew $< \delta$

2. In DS, errors occur due to unpredictable communication delay in message exchange to deliver a clock message and clock signals

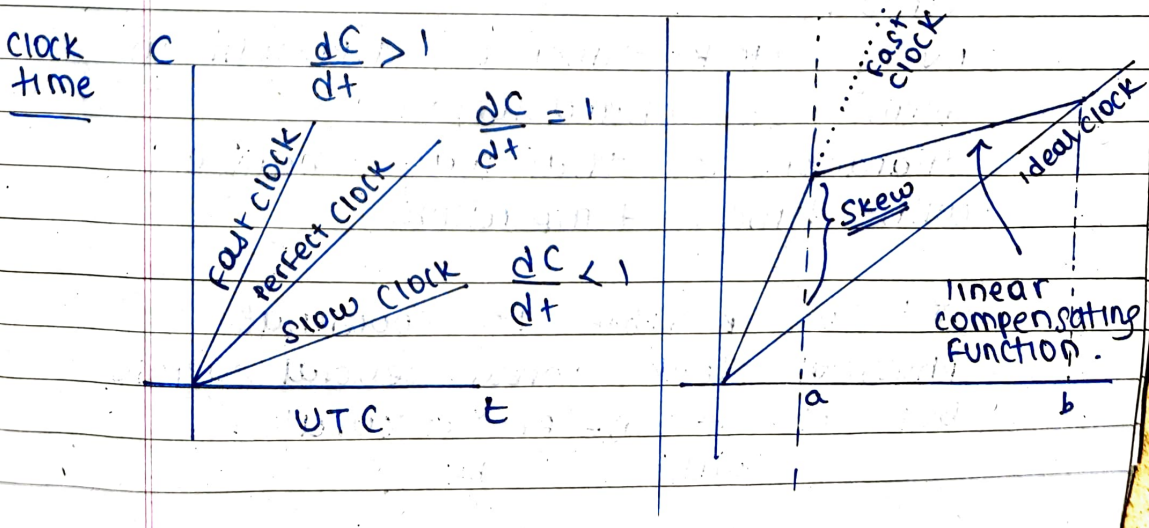
NOTE:

In clock synchronisation, clocks should never be adjusted backward as this can disrupt event ordering and system consistency.

Time should ALWAYS move forward

Instead, if a clock is ahead of the correct time, it should slow down slightly so that it gradually synchronises with correct time, without jumping backward.

clock synchronisation algorithms are designed in such a way that the fast clocks are readjusted slowly instead of readjusting to actual time at once



Clocks

Physical clocks

Logical clocks

In synchronisation, there are mainly two types of clocks.

① Physical clock :

- Time isn't a big issue in traditional centralised systems, where one or more CPU share ~~the~~ a common bus.
The entire system shares the same understanding of time, right or wrong, it is consistent.

- In distributed systems, that is not the case. Every system has its own timer that keeps the clock running.
These clocks are based on the oscillation of piezoelectric crystal or ~~some~~ similar integrated circuit.
Every ~~clock~~ timer is different in terms of characteristics - characteristics that may change with time, temperature

- It is possible to coordinate physical clock across several systems but it will never be accurate.

(2) Logical clock

- Logical clock means creating a protocol on computers in a distributed system so that computers can keep a uniform ordering of happenings inside some virtual time range.

- In a DS, a logical clock is a technique for recording temporal and causative links. Because distributed systems may lack a physically synchronized global clock; a logical clock provides for the global ordering of occurrences from various processes in certain systems.

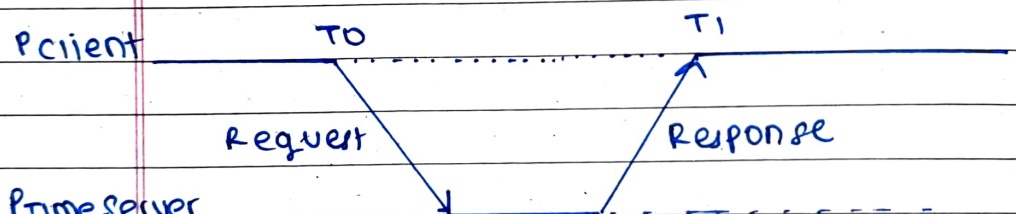
Clock synchronisation algorithms

(I) Christian Algorithm

1. It is a ~~synchronised~~ centralized clock synchronisation algorithm. used to synchronise time with time server by client processes.

2. This algorithm works well with a low latency network where the round trip time is short compared to accuracy.

Round trip time is the time start of request and end of corresponding response.



$$T_{new} = T_S + \left[\frac{T_1 - T_0}{2}\right] \quad \text{---(1)}$$

3. The client sends a request to a time server for its current value of UTC time (TS)

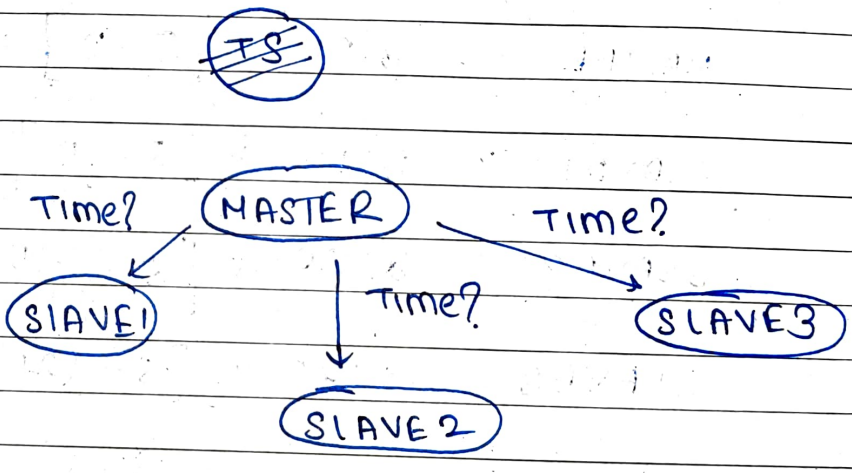
4. The client also records the time at which the request was submitted (T0) and time it got the response (T1)

5. Then client changes the current time at T1 with value received from Ts plus estimate of delay obtaing Ith value.

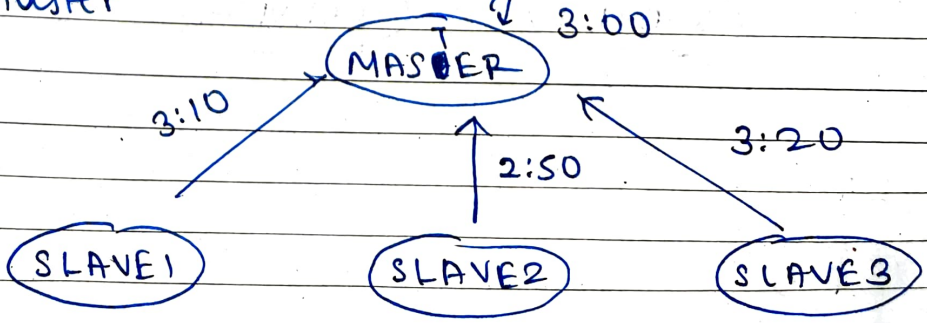
(II) Berkeley Algorithm.

1. It is a centralised clock synchronisation mechanism ~~where~~ in DS that implies that no computer has precise ~~time~~ timing source

2. It is an example of active time server approach. The time server periodically sends a message to all computers in a group.



3. Upon receiving the request each slave node reads its local time and sends back current time to master



4. Master node calculates the time difference blw all clock times received and clock time given by master systems clock ~~and~~

$$\text{Offset} = \frac{0 + (10) + (-10) + (20)}{4}$$

$$\text{Offset} = 5$$

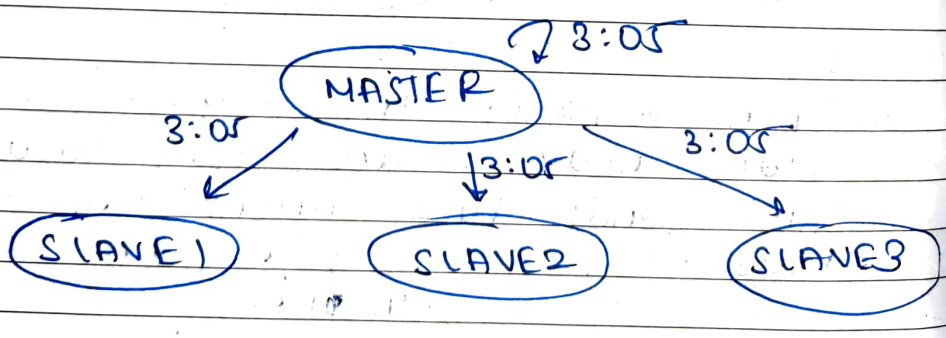
5 This average time difference is added to masters system clock and broadcasted over the network

MASTER : 3:00 + [5 - 0] = 3:05

SLAVE1 : 3:10 + [5 - 10] = 3:05

SLAVE2 : 2:50 + [5 - (-10)] = 3:05

SLAVE3 : 3:20 + [5 - (20)] = 3:05

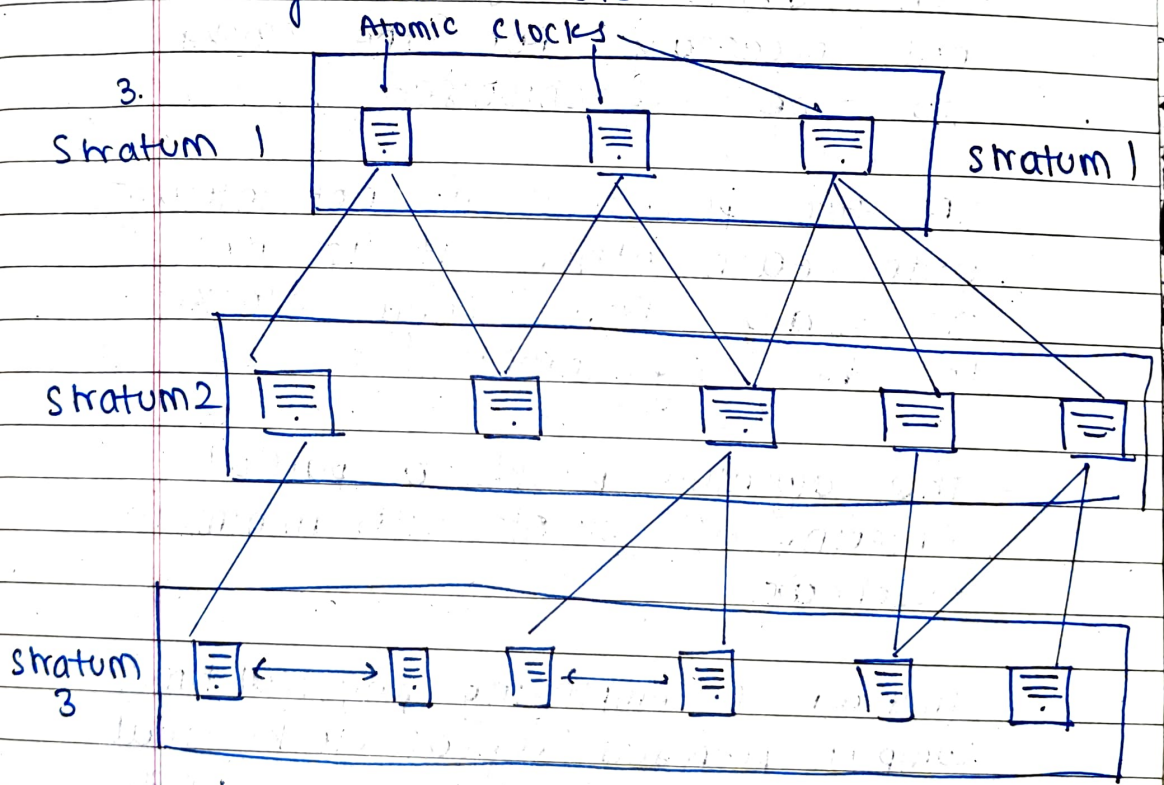


(iii) Network Time protocol (NTP)

1 — It is the standard followed by synchronisation clocks on internet. It is a decentralised algorithm.

2. The main servers are directly linked to a precise and dependable UTC time source. They are the foundations of hierarchical time service with additional servers becoming operational as we go away from roots:

3.



4. The common configuration is:

Stratum 1: UTC servers at big governmental institutions

Stratum 2: Institutional time servers or ISPs time server

Stratum 3: Most users linking to academic time servers

NTP may synchronise computers in three modes

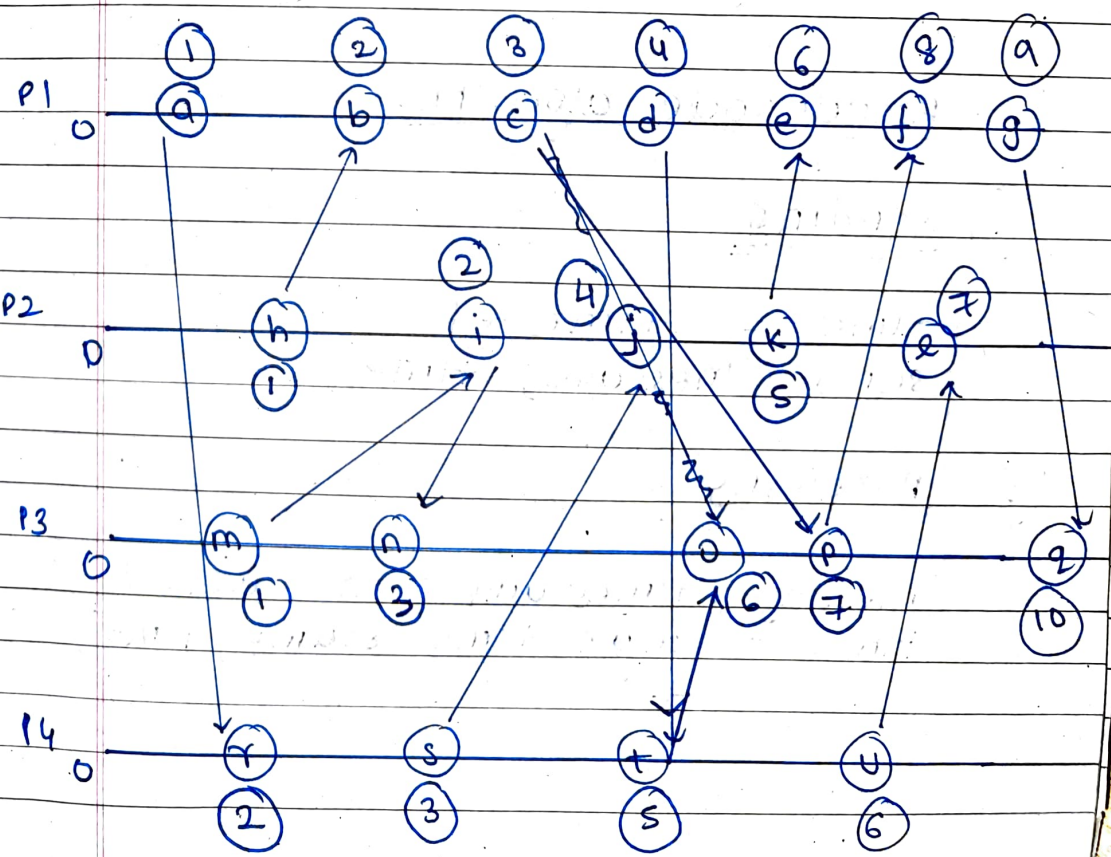
- client-server mode / Unicast
- Multicast mode / Multicast
- symmetrical (peer) mode / Broadcast

(*) **Lamport's logical clock**

- In distributed systems, it is not necessary for the clocks to be synchronised
- If two processes do not interact with each other, it is not necessary for their clocks to be synchronised.
- This algorithm provides a partial ordering of events with minimum overhead.
- Instead of employing physical time, Lamport proposed logical clocks that capture events' orderings through a 'happens-before' relationship

or process

- Each machine has its own clock and the clock is just a 'counter' that increases every time an event happens.
- It is initially set to zero.
- The counter increments by 1 for each event it experiences
- when a process sends a message to another process, it includes its current clock value with message.
- when a process receives message, it updates its clock to be the maximum of its own clock and received clock value and increments by 1.



①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
a	b	c	d		e		f	g	
h	i		j	k		l			
m		n			o	p			q
-	r	s		t	u				

conditions of lamports logical clock

For any two events a and b

If $a \rightarrow b$
then $C(a) < C(b)$

But converse is not true. always

Pseudocode algorithm:

(1) sending

time = time + 1
send (message, time)

(2) Receiving

message, timestamp = receive()
time = max(time stamp, time)
+ 1

(*) Vector Clocks

1.- In Lamports clocks, one cannot directly compare time stamps of two processes and determine their precedence relationship

i.e If a and b are two events and if $C(a) < C(b)$

$\not\Rightarrow a \rightarrow b$

does not imply

To solve this issue, vector clocks were designed

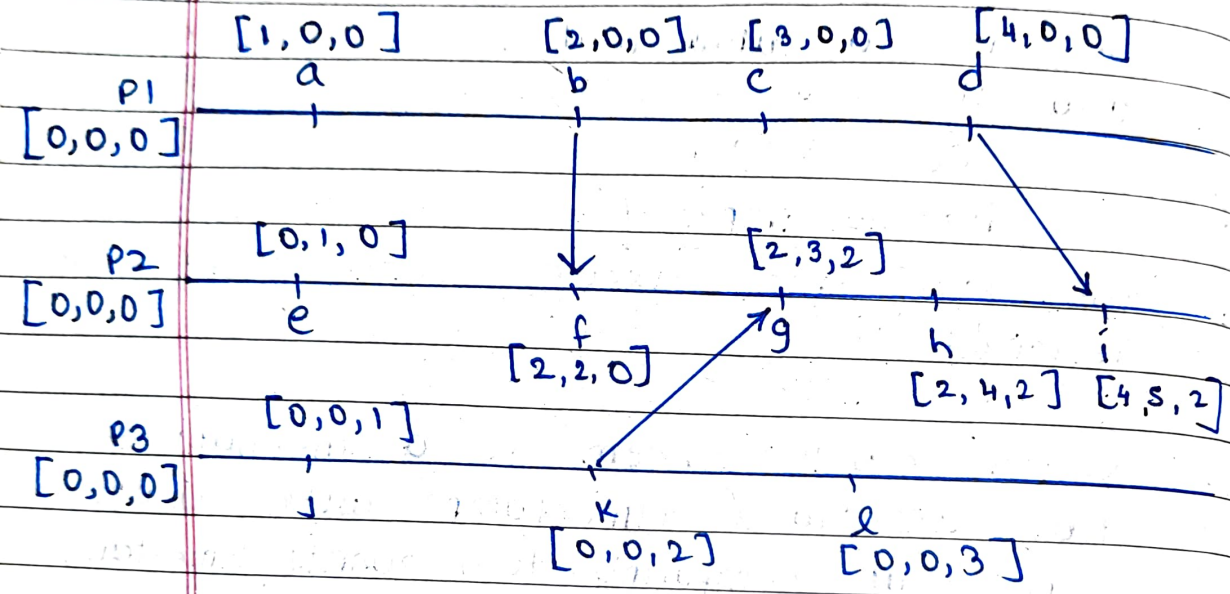
2- vector clock system is a mechanism that associates timestamps with events such that comparing two events timestamps indicate whether those two events are causally related.

3- vector clock is represented as a vector $[c_1, c_2, c_3, \ldots, c_n]$ with an integer clock value of each process. [ci contains clock value of process i]

4- when a process generates an event or sends a message, it increments its own entry in the vector clock and includes its current vector clock with an event or message

5- Upon receiving an event or message, the receipient process updates its own vector clock by taking maximum value from each corresponding entry of its own vector clock and received vector clock.

Additionally the recipient increments its own entry in the vector clock to account for the received event.



MUTUAL EXCLUSION

- Mutual exclusion is an important concept in concurrent programming which ensures that simultaneous operations or node locations do not use common assets or crucial areas concurrently.

Thus it prevents conflicting and inconsistent results.

- **ON a SINGLE COMPUTER,**

memory and other resources are shared blw different processes. Thus status of shared resources and status of users is easily available. So with help of shared variable, mutual exclusion can be solved easily.

- **IN DISTRIBUTED ENVIRONMENT**

We have neither shared memory nor common physical clock and therefore mutual exclusion problem is difficult in DS.

- To eliminate the mutual exclusion problem in DS, approach based on message passing is used.

Requirements of Mutual exclusion Algorithm.

(1)- No Deadlocks : Two or more site should not wait endlessly for any message that will never arrive.

(2)- No Starvation :

Each site who wants to execute critical section should get an opportunity to execute it in finite time

Any site should not execute critical section indefinitely

Any site should not WAIT indefinitely to execute critical section while other sites are repeadly executing critical sections.

(3)- Fairness :

Each site should get a fair chance to execute critical section

Any request made to critical section must be executed in the order they are made.

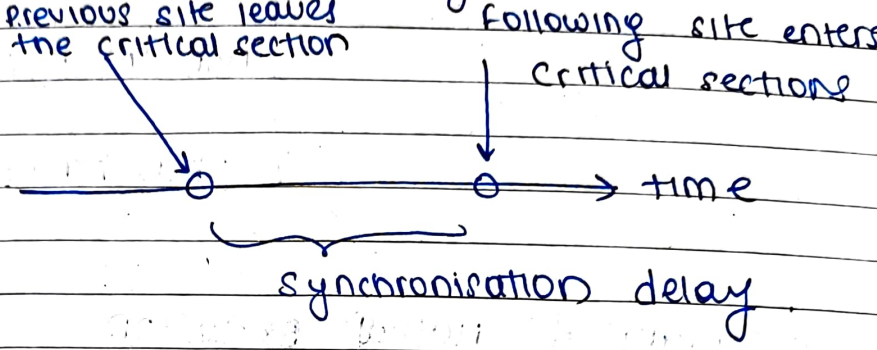
(4)- Fault tolerant

In case of failure, it should be able to recognise it by itself in

order to function correctly without disruption.

Peformance metrics

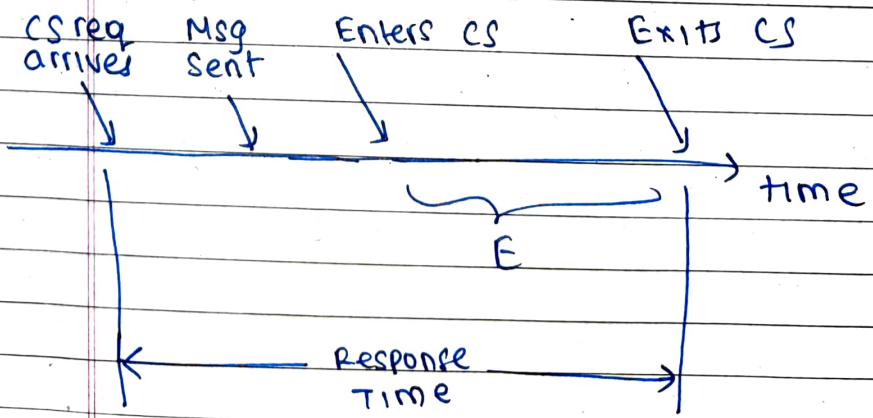
① - Synchronisation delay



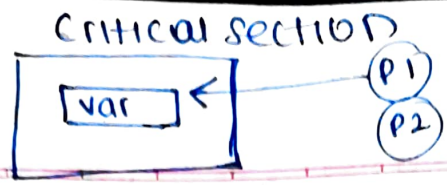
② - System Throughput: Rate at which request for critical section gets executed.

③ - Message complexity: Number of proceses that where required by CS execution by a proces.

④ - Response Time: - Time interval from a request send to CS and execution completion.



one process at a time



Mutual Exclusion

- Centralised Mutual Exclusion
- Distributed Mutual Exclusion
  - Non-token based
  - Token-based

(1) Centralised Mutual Exclusion Algorithm.

1. In centralised algorithm, one process is elected as a coordinator which maybe the machine with highest network address.

2. The coordinator coordinates all processes and grant the permission to a process to enter in the critical section (CS) to use the shared resources.

3. when a process wants to enter the critical region, it sends a request message to the coordinator stating which critical region it wants to enter and asking for permission.

4. If no other process is currently in that critical region, the coordinator sends back a reply ~~grand~~ granting permission.

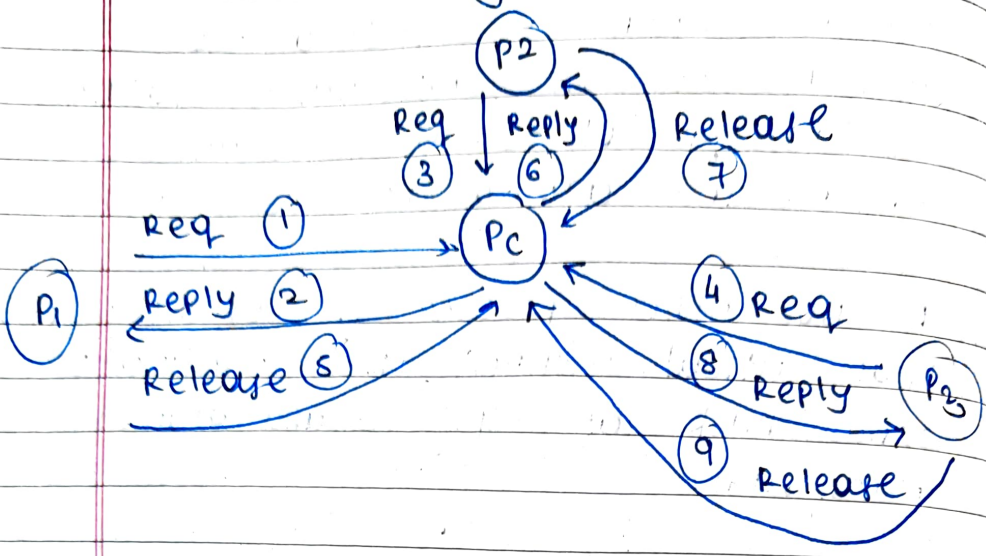
When the reply arrives, the requesting process enters critical region.

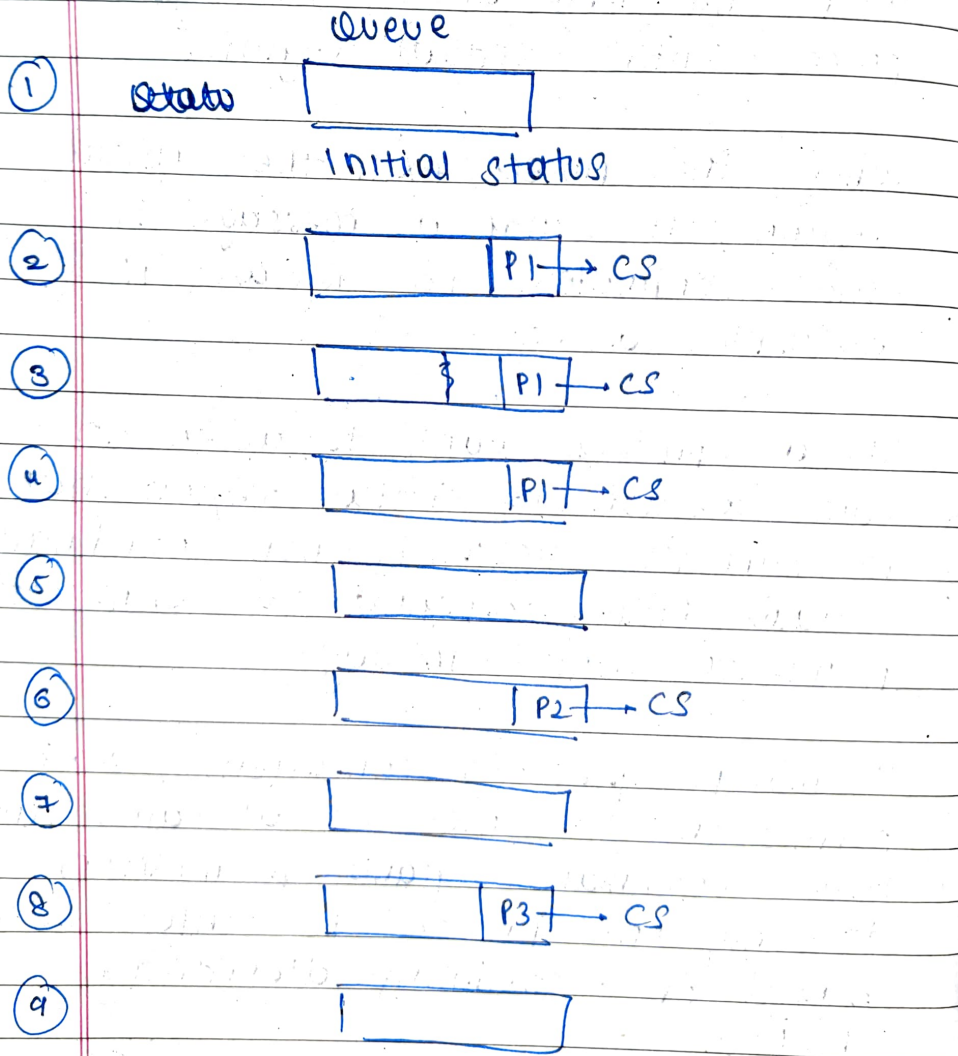
When the process exits the critical region, it sends a message to the coordinator to release its exclusive access.

If a process wants to access the same section of critical region that is currently occupied, then coordinator refrains from replying or replies with permission denied message

If multiple processes concurrently seek permission to enter CS at same time, then coordinator grants permission to only one process to enter CS based on scheduling algorithms (FCFS).

Given following Senario





[Suzuki-Kasami]

Distributed Algorithm
- Token based
- Quorum based [E Maekawa]
- Non token based [Lamports, Ricart-Agarwala, Maekawas]

(1) Lamports Algorithm for mutual exclusion in distributed systems.

- Lamports Algorithm is a permission based algorithm proposed by Lamport as an illustration for his synchronisation scheme for distributed systems.

- In permission based, timestamp is used to order critical section requests and to resolve any conflict b/w requests.

- In this algorithm, critical section requests are executed in increasing order of timestamps. → FCFS.

(1) - When a process wants to enter critical section, it sends a time stamped request to all other processes in system.

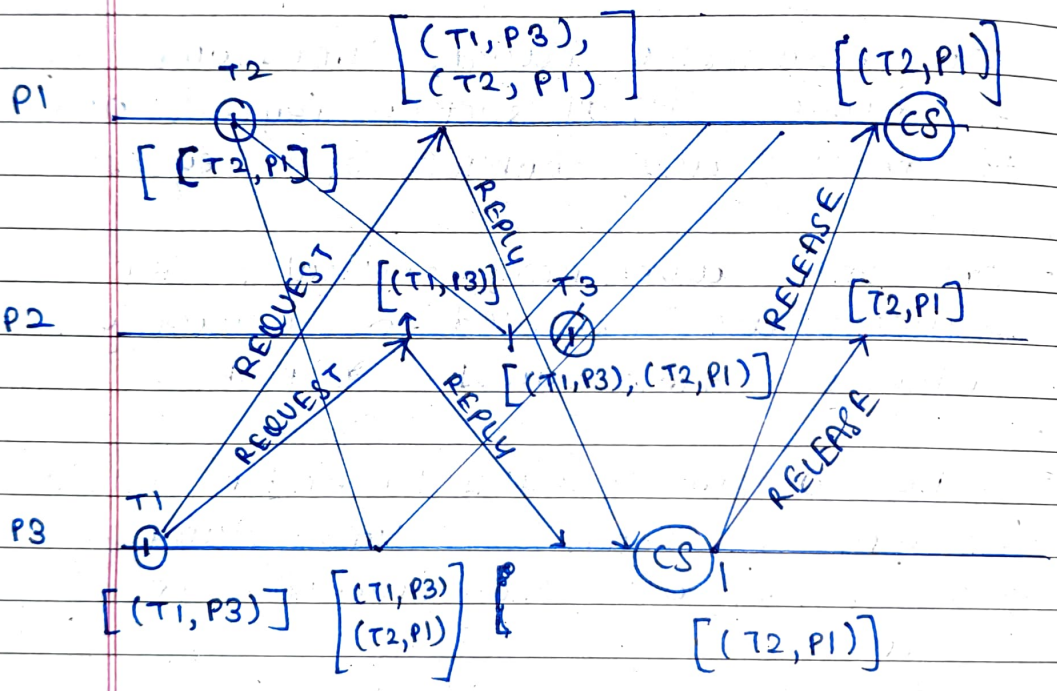
(2) - Upon receiving the request, each process sends a reply ~~with~~ message ~~with~~ to requesting process.

(3) - The requesting process can only enter critical section only after it has received messages from all processes and its own request has earliest time stamp among all outstanding request

(4) — After executing critical section, the process sends a release message to all ~~mess~~ other processes informing them that the shared resource is now available.

— Types of messages in lamport's Algo

{ REQUEST
REPLY
RELEASE }



(Timestamp, process-ID)

(*) — Performance

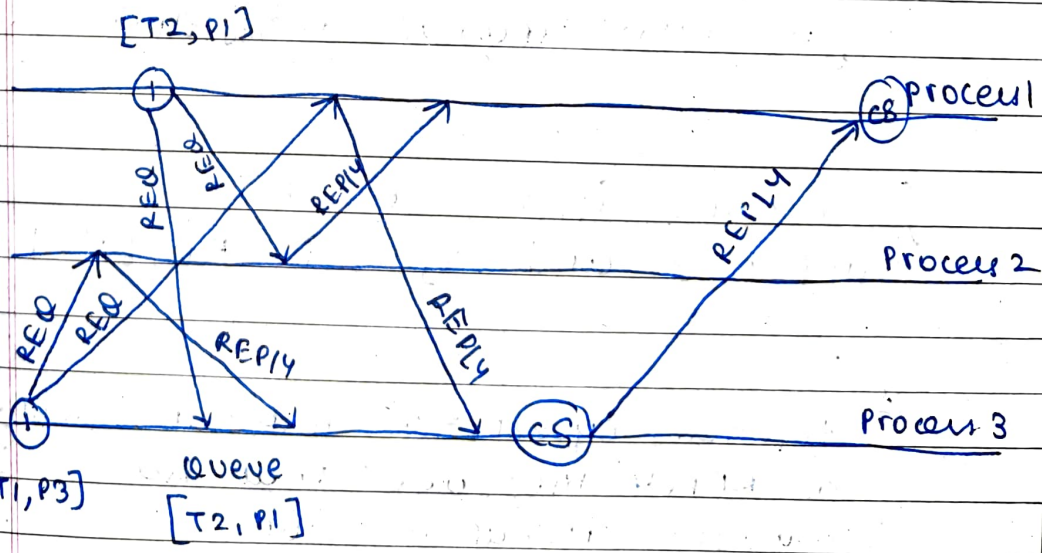
For each CS execution, lamport's algo requires (N-1) Req, (N-1) reply and (N-1) release messages

⟹ lamport's algorithm requires 3(N-1) msgs per CS invocation.

(2) Ricart - Agarwala Algorithm.

- Ricart - Agarwala algorithm is an optimization on lamport's algorithm.
- Ricart - Agarwala algorithm uses only two types of message: REQUEST and REPLY
- It is assumed that all processes keep a logical clock which is updated according to the clock rules
- The algorithm requires a total ordering of requests. Requests are ordered according to their global timestamps.

If timestamps are equal, process identifiers are compared to their global timestamps If time stamps are equal, process identifiers are compared to order them.



- A process sends a REQUEST message to all other processes to request their permission to enter critical section.

- A process sends a REPLY message to a process to give its permission to enter CS.

Note:

when process S_i requests process S_j

S_j sends a reply message IF and only if:

S_j is neither requesting nor executing CS

OR

IF S_j is requesting and S_i's request timestamp is smaller than S_j's own request timestamp.

Otherwise the request is deferred and S_j sets $RD_j[i] = 1$

deferred request ⟹ delay in sending a REPLY message that has request critical section.

site Si enters the CS after it has received a REPLY message from every ~~mess.~~ site it sent a REQUEST message to.

when site Si ~~exist~~ exits the CS, it sends all the deferred REPLY messages

i.e $\forall j$ IF $RD_i[j] = 1$ then:

send Reply message to Sj and set $RD_i[j] = 0$

Performance

For each CS execution, Ricart Agarwala algorithm requires (N-1) request messages and (N-1) Reply messages.

Thus, it requires 2(N-1) msgs per CS execution.

* Quorum Based Mutual Exclusion Algorithm

- Quorum based algorithm are different in two ways

① A site does not request permission from all sites, but only from a subset of the sites.

The request set of sites are chosen such that
$\forall i \forall j : 1 \leq i, j \leq N$

$R_i \cap R_j \neq \emptyset$

consequently each pair of sites has a site that mediates conflicts b/w that pair.

② A site can send out only one REPLY message at any time. A site can only send a reply message only after it has received a RELEASE message for previous REPLY message

Maekawa's Algorithm

- Maekawa's Algorithm was the first quorum based mutual exclusion algorithm
- In this algorithm, a site does not request from every other site but from a subset of sites which are called quorum
- The request sets for sites i.e quorum in this algorithm are constructed as follows:

① $\forall i \; \forall j : 1 \leq i, j \leq N, i \neq j :: R_i \cap R_j \neq \emptyset$

i.e R_i and R_j must overlap.
i.e there is atleast one common site blw the request set of any two sites.

② $\forall i : 1 \leq i \leq N :: S_i \in R_i$

③ $\forall i : 1 \leq i \leq N :: |R_i| = K$

→ Every process S_i must be a member of its own quorum set R_i

→ This states that the size of each quorum R_i is exactly K where K is a constant.

④ $N = K(K-1) + 1$ and $|R_i| = \sqrt{N}$

where : $N \rightarrow$ total no. of processes in system
K is size of each quorum ($|R_i|$)

Tirthraj Mahajan

$N = K(K-1)+1$ ensures that each quorum R_i overlaps with all other quorum

suppose : $K = 3$

$N = 3(3-1)+1 = 7$

∴ For 7 processes, quorum size = 3

$|R_i| = \sqrt{N}$

This ensures that size of each quorum is proportional to the total no. of processes

IF N is perfect square, then $K = \sqrt{N}$ making quorum construction symmetric and balanced.

Algorithm

A site S_i executes the following steps to execute the C.S.

① To enter critical section

- when a site S_i wants to enter the critical section, it sends a REQUEST message to all other sites in request set R_i

- when a site S_j receives a REQUEST (i) message —

condition1: If Sj has not already sent a reply message since the last RELEASE message, it immediately responds to Si with a reply message.

condition 2: If Sj has already already sent a reply message since the last release message

Condition 2 : If Sj has already sent a reply message (eg another site Sk), Sj cannot send a Reply msg immediately.

Instead it queues up the request from Si.

(2) To execute the critical section:

A site Si can enter the critical section if it has received the REPLY message from all the site in request set Ri

(3) To release the critical section:

- When a site Si exits the CS, it sends a RELEASE(i) message to all the sites in request set Ri

- When a site Sj receives the RELEASE(i) message from site Si, it sends a REPLY message to the next site waiting in queue and deletes the entry from queue

- In case queue is empty, site S_j update its status to show that it has not sent any REPLY message since the receipt of the last RELEASE message.

Performance :

Maekawa's Algorithm requires invokation of $3|R_i|$ i.e $3\sqrt{N}$ messages per critical section execution as the size of a request set is $\sqrt{N}$.

$\sqrt{N}$ → REQUEST
$\sqrt{N}$ → REPLY
$\sqrt{N}$ → RELEASE.

NOTE

Maekawa's Algorithm can deadlock because site is exclusively locked by other sites and requests are not prioritized by their timestamps.

P1

P1 → P1 P2 P4
P2 → P1 P2 P3
P3 → P2 P3 P4
P4 → P1 P3 P4

REQ

Flag=F P1 —REQ→ P2 Flag=F

1

REQ ↓

Flag=F P4 P3

GRANT

2 Flag=T P1 ←GRANT— P2 Flag=T

GRANT ↑

Flag=T P4 P3

CS

3 Flg=T P1 P2 Flag=T

REQ ↑

Flag=T P4 ←REQ— P3 Flag=F

REQ

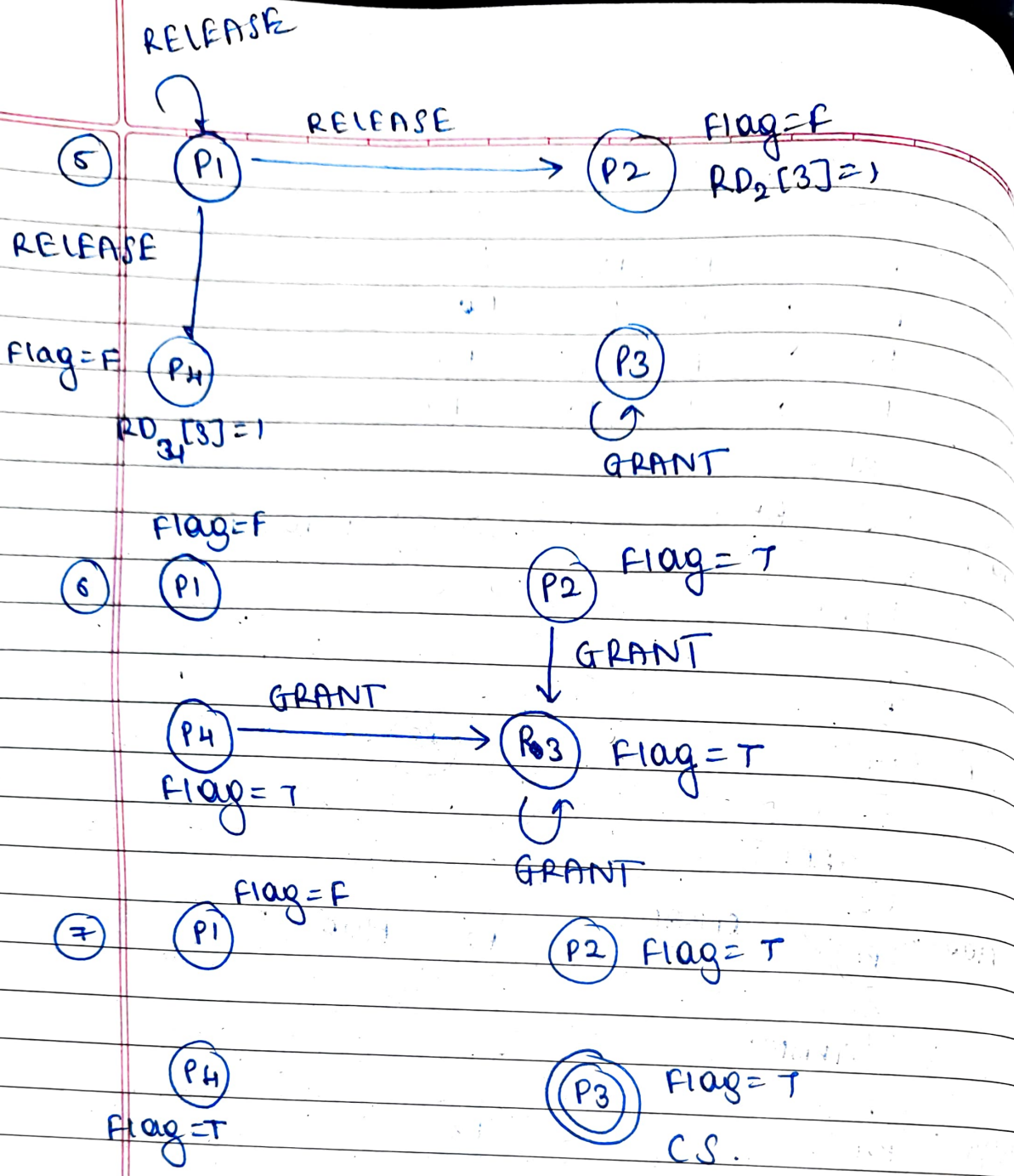
CS

4 P1 P2 Flag=T $RD_2[3]=1$

Flag=T P4 P3 Flag=F

$RD_4[3]=1$

GRANT


RELEASE
RELEASE
5
P1
P2
Flag=F
RD2[3]=1
RELEASE
Flag=F
P4
RD3,4[3]=1
P3
GRANT
Flag=F
6
P1
P2
Flag=T
GRANT
GRANT
P4
Flag=T
P3
Flag=T
GRANT
Flag=F
7
P1
P2
Flag=T
P4
Flag=T
P3
Flag=T
CS.

Token based algorithms:

- In token based algorithms, a token is shared among the sites.
- A site is allowed to enter its CS if it possesses the token.
- Token based algorithms use sequence nos instead of timestamps to distinguish blw old and current req.
- A Every site requesting to get a token contains a sequence no.
- On every token request, the sequence number is incremented by 1.

Token based Algorithms
- Token-Ring Algo
- Suzuki Kasami Algorithm.

(1) Token Ring algorithm

- ~~Token based~~
- Token-Ring algorithm is an example of token-based distributed mutual exclusion algorithm.
- It is based on concept of a circulating ring in a logical ring topology.

P.T.O

* - Token creation and initial Dismibution.

1. - Initially, there is only one token in the system.
Each node is allowed to hold the token for a fixed amount of time. Once the node is done with the operation or does not need the token, it passes on to the neighbour ring

2. Suppose N processes, token is passed from P_K to $P_{(K+1) MOD N}$ in point to point message.

* - Critical section execution

1 - When a process aquires the token from its neighbour, it checks if it needs to access the share resources.
2 - If so, the process goes ahead and enters CS.
3 - After it is finished, it exits the CS and paues the token along the ring.
4 - It is not permitted to immediately enter CS again for different event using the same token.

* - Token passing

1.- Depending on the order and distribution of token, certain process might get more opportunity to access CS than others. faster.
To ensure fairness, various strategies can be employed such as

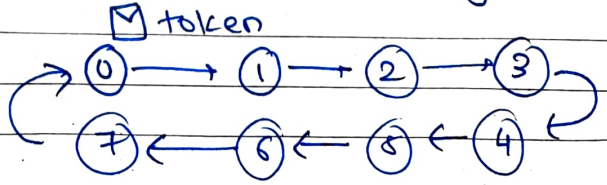
2- However, as the token circulates throughout the ring, it ensures each process gets a chance to enter CS.

3- If the node holding the token crashes, we can detect the token disappearance by calculating the maximum time to complete one round.
Suppose there are N nodes, each node can hold token for maximum 't' time, total completion time = Nt (max)

4- Thus if the token does not return to a node by that time limit, we can consider token as lost.

5- Recreation of token has to be done carefully so that only one machine ends up creating new token and we don't end up creating multiple tokens in ring.

6- One strategy is to elect a leader who is responsible for detecting token loss and regenerating token.



diagram

Election Algorithms

1- Election algorithms are used to elect a leader or coordinator among a group of distributed nodes.

2- The leader is responsible for ~~creating~~ cooperating and managing the activities of the distributed system to ensure proper functioning and efficient communication.

3- Distributed election algorithms are essential for maintaining fault tolerance, load balancing and overall system reliability.

4- The choice of algorithms depends on the specific requirements of system and desired tradeoffs b/w simplicity, performance and fault tolerance.

common algos

- RING
- BULLY.

learn From GitHub

GOSSIP BASED COORDINATION

1 - Gossip based coordination is a decentralized approach used in DS for achieving coordination and information dissemination among nodes.

2 - Inspired from real world gossip spreads where information is passed from one person to another in random and probablistic manner.

3 - In gossip based coordination system, each node communicates with few randomly selected ~~neighbouring~~ neighbour nodes at regular intervals of time.

4 - During these interactions, nodes exchage information they possess and this information spreads throughout n/w gradually.

5 - The protocol continues till desired level of coordination or consensus is achieved among mayority of nodes.

6 - It is important to note that while gossip protocols are scalable and fault tolerant, it may take time for ~~to~~ message to propogate through the n/w.
↳ eventual consistency rather than immediate consistency.

(I) Aggregation

1- Aggregation is a gossip based coordination process in which we collect and combine information from multiple nodes to generate a summary or a consolidated view of data.

2- Each node communicates only with small, randomly chosen subset of nodes in each round and they exchange local state information and converge them toward desired aggregate value.

3- Each node maintains a ~~partial result~~ local state representing partial result

4- Each node maintains two different threads - active and passive thread

5- Active thread periodically sends the local state with random neighbour

6- Passive thread waits for messages by other nodes, and as soon as it receives a message, it will send back a reply containing its own local state.

7- Both nodes update their local state on the aggregation function

8- over successive rounds, the states accro

nodes converge to a global aggregate.