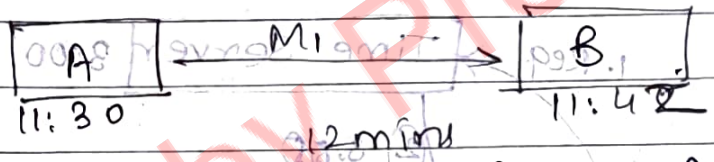
Chp. 3 Synchronisation.

1. Physical clock synchronisation.

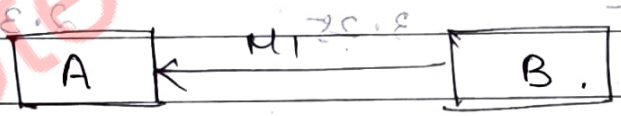
The process of maintaining same time on all the machines in the particular nw is known as clock synchronisation.

(*) Need:

To avoid communication error caused due to difference of time.

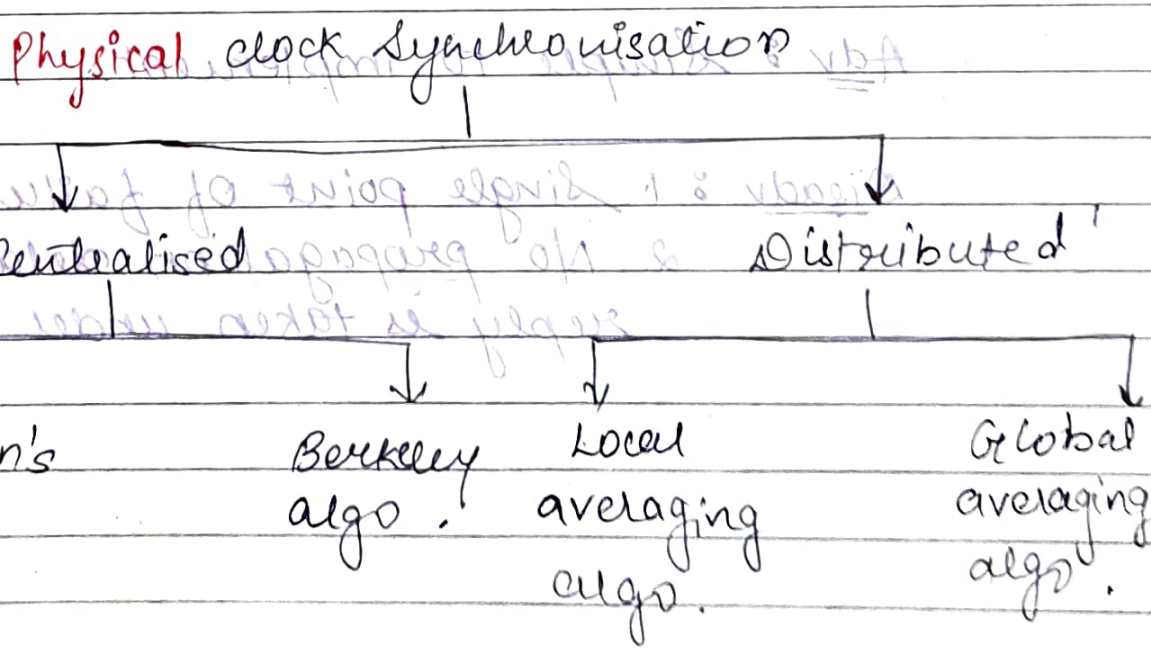


Error: M1 took ~~15 mins~~ from A to B.
(actually less)



Error: M1 is delivered ~~before~~ it is started.

2. Clock Synchronisation approaches:

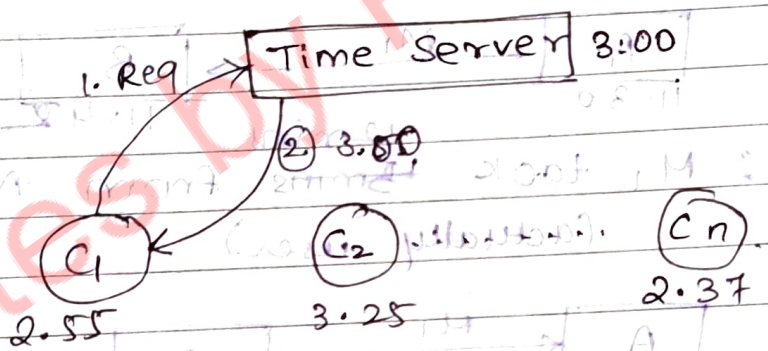


* Centralised clock Synchronisation :

Ek server hoga jo sabka time synchronise karega.

→ Cristian's algo :

There will be a time server which is responsible for maintaining time for all the machines in the network.



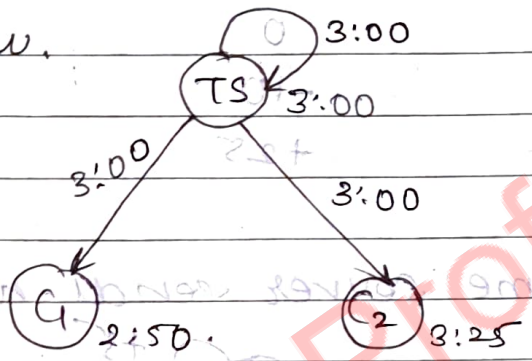
Whenever client requests, server reply with its own time and client sets the time according to the reply.

Adv : Simple to implement.

Disadv : 1. Single point of failure.
2. No propagation delay of the reply is taken under consideration

→ Berkeley algo :

Step1 : Time server will broadcast its own time to all the machines including itself in the n/w.



Step2 : Every machine calculate, clock value. (CV)

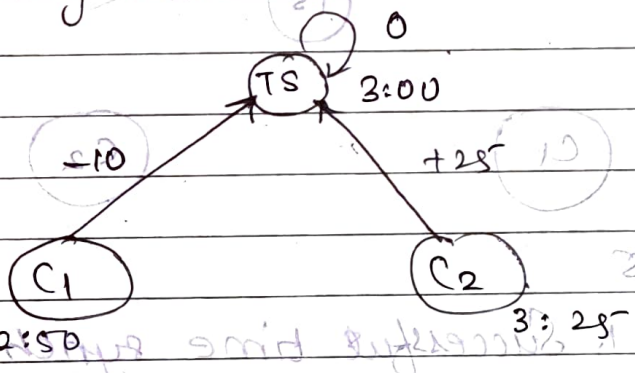
CV = Own Time - Server's Time.

CV (TS) = 0

CV (C1) = -10

CV (C2) = +25

Step3 : Every machine sends CV to time server



Step4 : Time server will now calculate average of all CV's.

$$\text{Avg} = \frac{CV(TS) + CV(C_1) + CV(C_2)}{3}$$

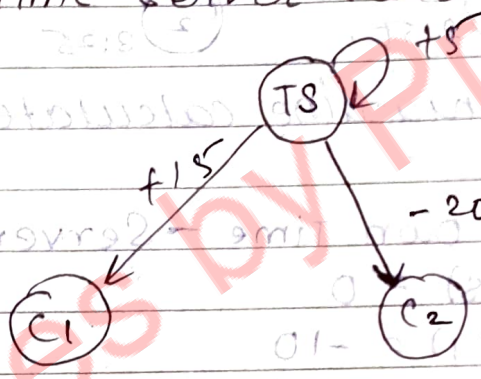
$$= \frac{0 - 10 + 25}{3} = +5$$

Step 5 : Time server now calculates new CV known as clock Adjustment Factor.

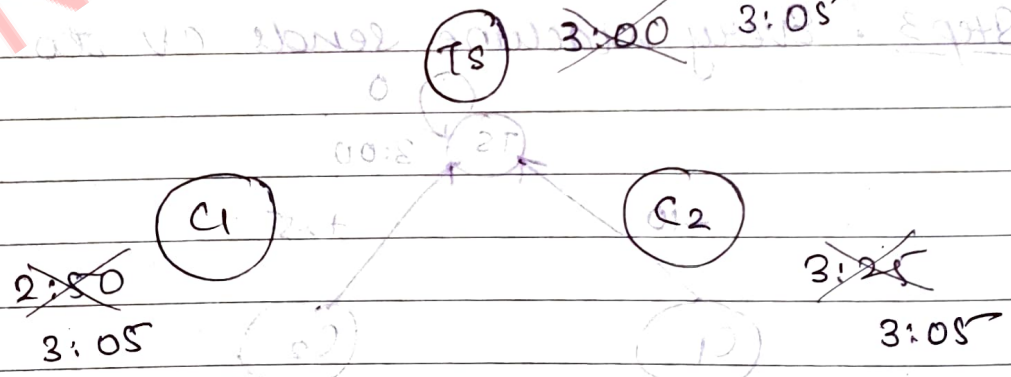
CAF (new CV)

Process	Old CV	New CV
TS	0	+5
C1	-10	+15
C2	+25	-20

Step 6 : Time server sends new CV to all.



Step 7 : Every node adjust the time according to new CV.



Adv : 1. Successful time synchronisation.

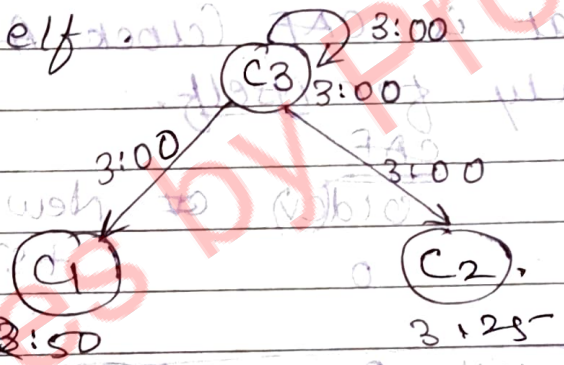
Disadv : 1. Single point of failure.
2. Time consuming.
3. Complex design.

* Distributed ~~clock~~ Clock Synchronisation :

{ koi kisika nahi }.

Global Averaging Algorithm :

Step 1 : The client that needs to adjust the time will broadcast its own time to all the machine including itself.

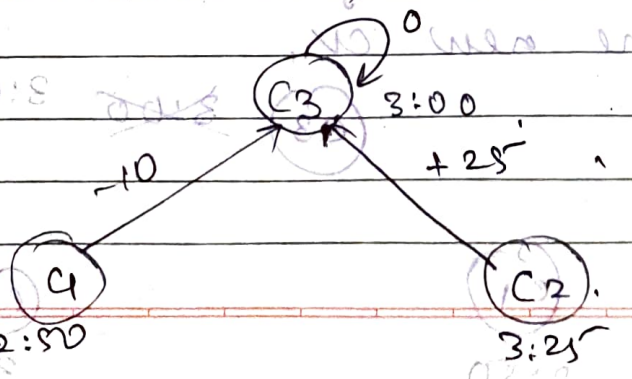


Step 2 : Every machine will not calculate clock value (CV).

CV = own time − Receiving (R^t) time.

CV (C3) = 0.
CV (C1) = −10.
CV (C2) = +25

Step 3 : Every node will now send the calculated CV, to C3.



Step 4: C3 will now calculate average of all CV.

$$Avg = \frac{CV(C3) + CV(C1) + CV(C2)}{3}$$

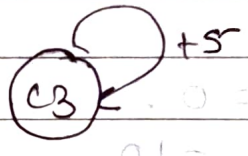
$$= \frac{0 - 10 + 25}{3} = +5 //$$

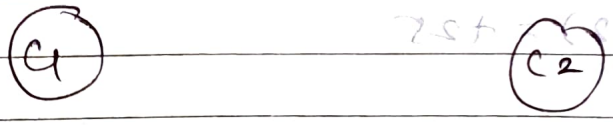
Step 5: C3 will now calculate new CV that is CAF (Clock Adjustment Factor) only for itself.

Process	CAF	Old (CV)	New (CV)
C3		0	+5

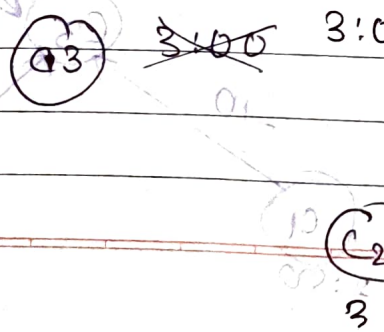
ONLY FOR ITSELF

Step 6: C3 will send new CV to itself.





Step 7: C3 will adjust time according to the new CV.

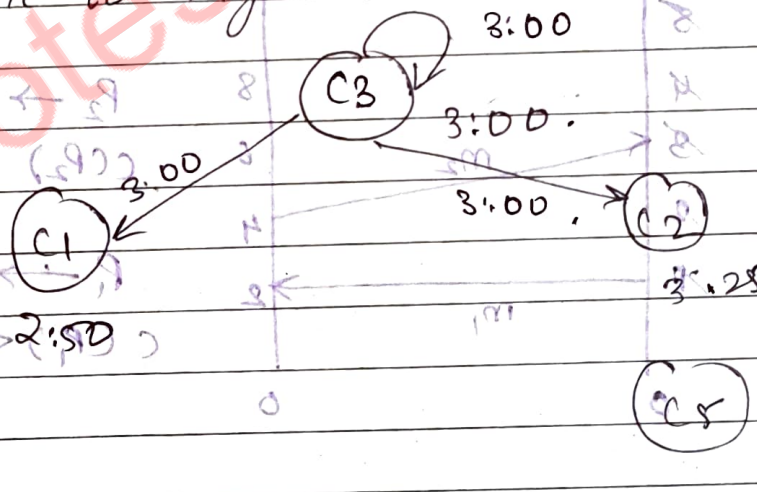


Adv : 1. No single point of failure.

Disadv : 1. No successful synchronization in one go, it will take some iteration
2. Difficult to implement.
3. Time consuming process.
4. Due to broadcasting n/w bandwidth is wasted.

Local averaging algorithm :

Step 1 : The client that needs to adjust the time (C3) will send its own time to neighbours and to itself.



Step 2 to Step 7 (same as global averaging algorithm).

Adv : 1. No single point of failure
2. Since no broadcasting. Bandwidth utilisation is not waited.

Disadv : ① No successful synchronisation in one go, it will take some iteration.
② Difficult to implement ③ Time consuming process.

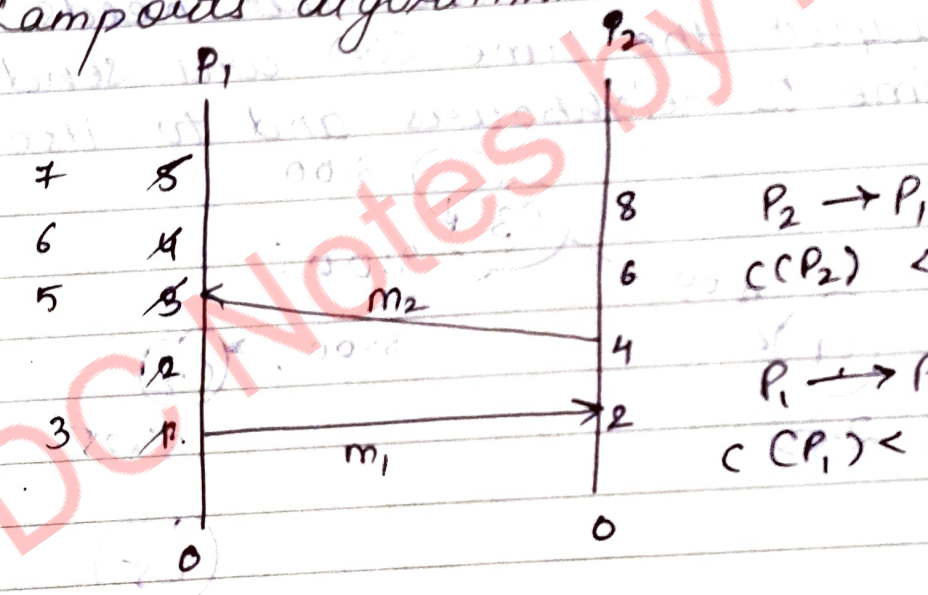
Page No.
Date

Logical clock synchronization / Event ordering

It works on the principle of happened before a relationship which is denoted as '$\rightarrow$'. Eg: $a \rightarrow b$.

$C(a) < C(b)$ this indicates event 'a' happened before event 'b' and hence clock (a) should be less than clock (b).

If a happened before b i.e. $a \rightarrow b$ $C(a) < C(b)$ is not true then use Lamport's algorithm.



$P_2 \rightarrow P_1$

$C(P_2) < C(P_1)$

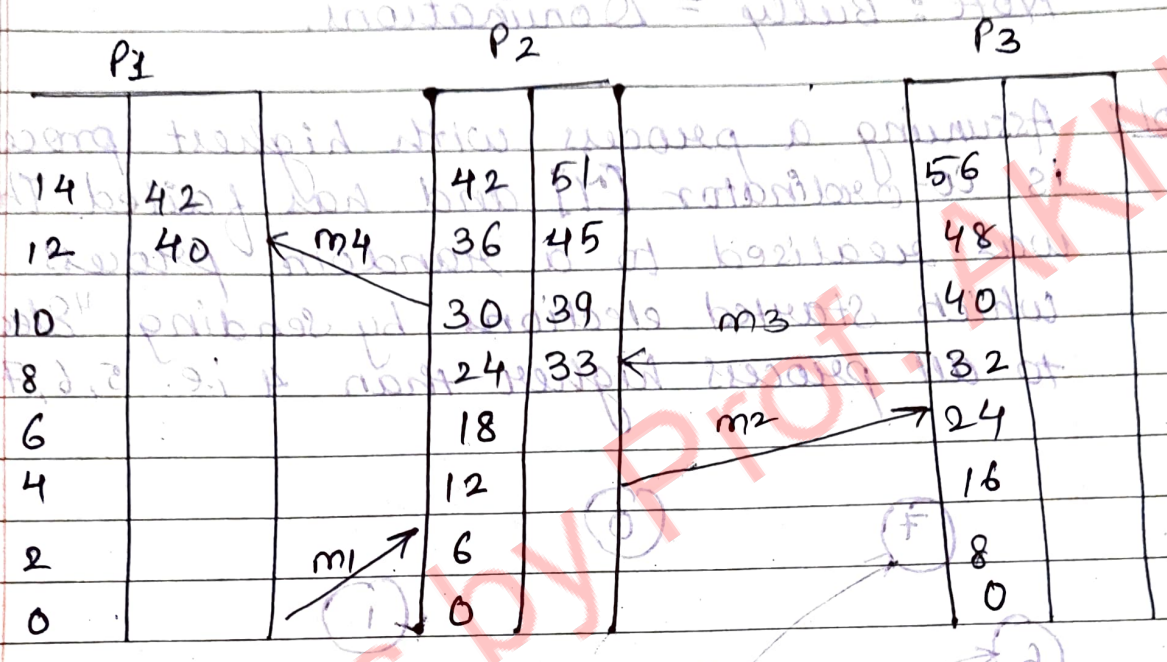
$P_1 \rightarrow P_2$

$C(P_1) < C(P_2)$.

It states, consider sender's time, add 1 with the assumption of propagation delay and set that as the new time of the receiver.

~~LAMPORT's~~

Q.] For the following architecture apply Lamport's rule:



m1 : $P_1 \rightarrow P_2$

$C(P_1) < C(P_2) \rightarrow$ True.

m2 : $P_2 \rightarrow P_3$.

$C(P_2) < C(P_3) \rightarrow$ True.

m3 : $P_3 \rightarrow P_2$. $P_2 \rightarrow P_3$.

$C(P_3) < C(P_2) \rightarrow$ False

m4 : $P_2 \rightarrow P_1$. $P_1 \rightarrow P_2$.

$C(P_2) < C(P_1) \rightarrow$ False.

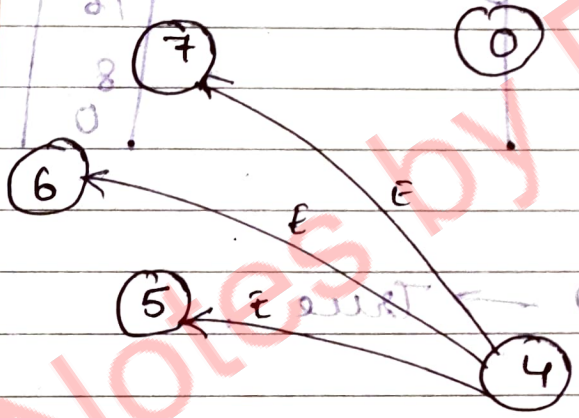
(*) Election Algorithm.

It is used for selecting co-ordinates in distributed environment.

1.] Bully Algorithm.

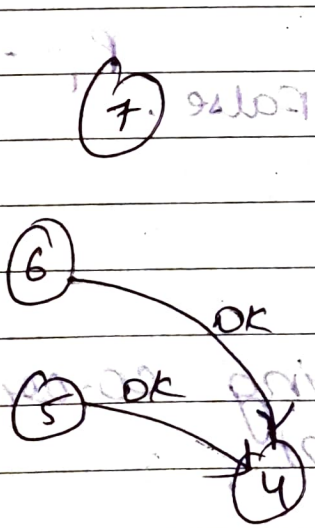
Note : Bully = Dominations.

Step1: Assuming a process with highest process id is co-ordinator [7] and has failed. This was realised to a random process 4 which started elections by sending "Election" to all process higher than 4 i.e. 5, 6, 7.

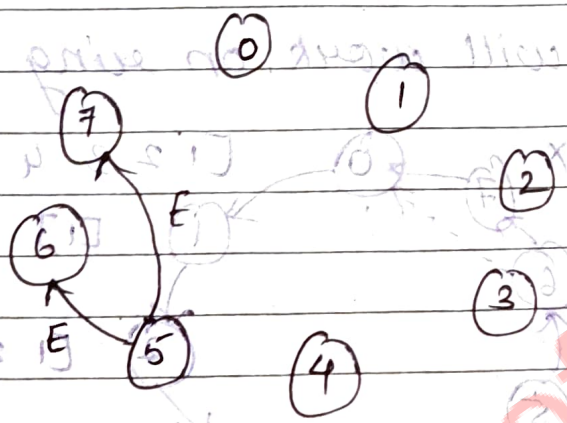


E : Election

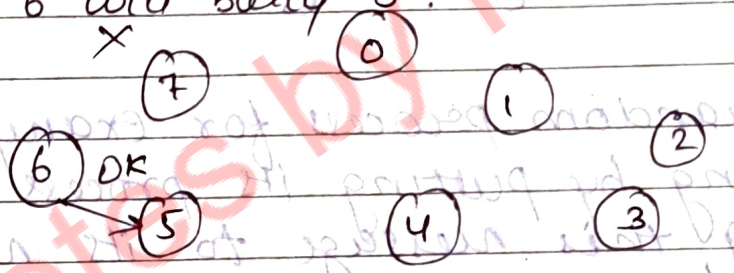
Step2: 5 and 6 realises process lower than them has started elections and so both with bully 4 by sending "OK".



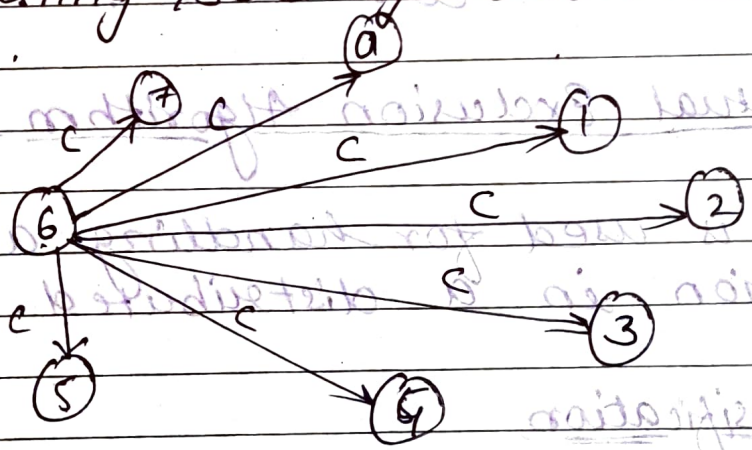
Step 3: Now 5 and 6 starts the election.



Step 4: Now 6 will bully 5.

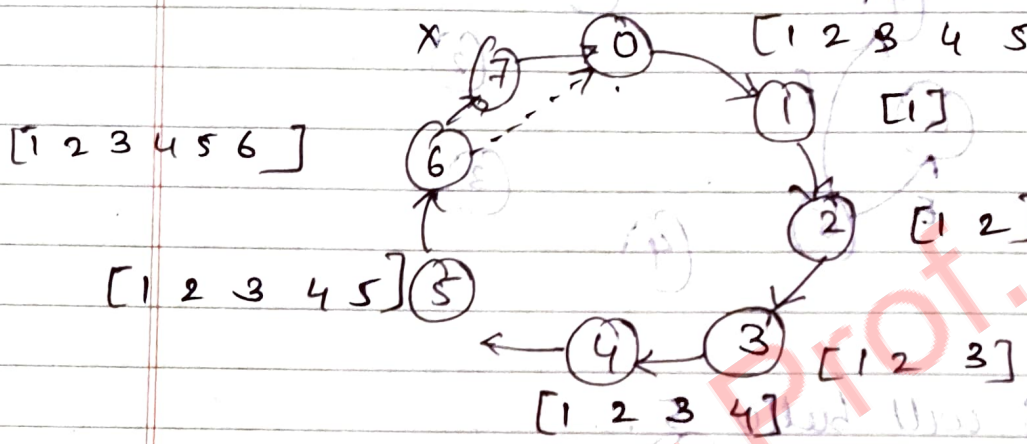


Step 5: Since 7 is down, 6 is winner and it informs all the nodes about the wining by broadcasting / sending "co-ordinator" message.



*2.] Ring Algorithm.

It will work on ring topology.



→ Any random process for example 1 starts probing by putting its process ID, and giving this message to its neighbour i.e. '2' and this will continue till the message is received by the initiator i.e. 1.

→ Looking at the prob one will understand who is the "coordinator".

* Mutual Exclusion Algorithm.

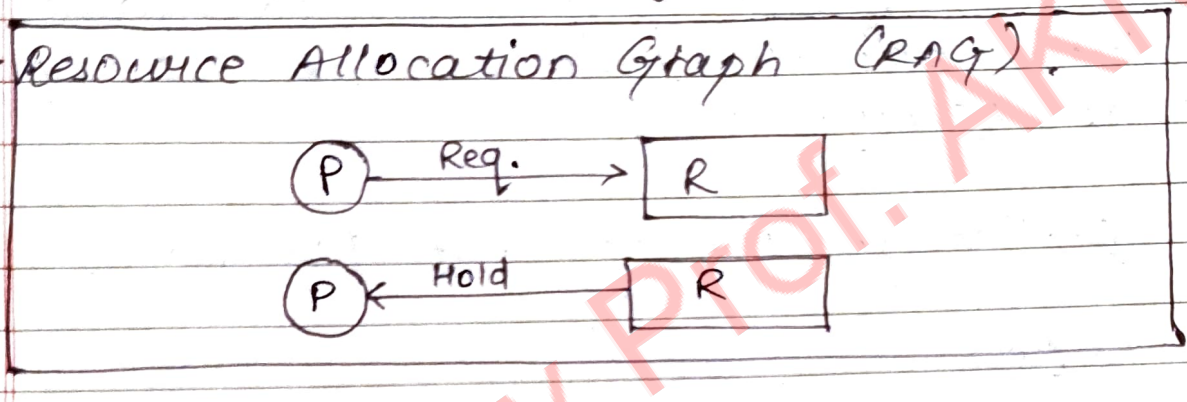
It is used for handling a critical section in a distributed environment.

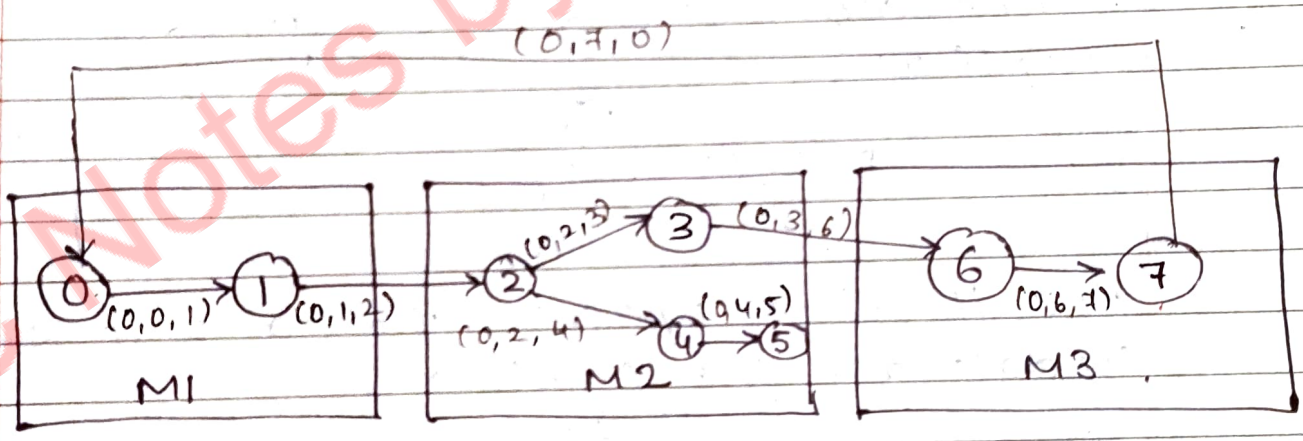
Classification.

* Deadlock in distributed System.

⊛ Chandy Hass Misra (CHM) OR Edge Chasing Algorithm

→ Resource Allocation Graph (RAG).





Any random process for eg. 0 start probing which contains (initiator, sender, receiver) this probe would pass to every node and if Sender gets back the probe message, there is a cycle in the network that is deadlock, so the initiator will kill itself.

Problem : If multiple node start probing they will kill itself, if encountered with deadlock.