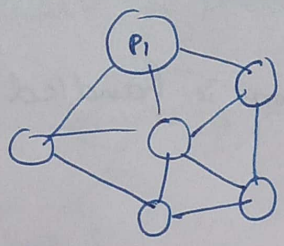
PDC

80M - Theory
20M - Internals.

Distributed Computing
P.K Sinhali

Scalability
Fault Tollerance
Reliability
Random Variable

PEER - PEER MODEL:



Distributed Systems:
→ Data that can be stored at different geographical syste
→ Even if one fails others work.

→ If we use distributed system:
- The fault-tollerence is reduced

⇓

at a time
It acts as a client & server.
Works in the Gossip protocol.
⇓
Information can be passed from one system to another system.

PHYSICAL CLOCK & LOGICAL CLOCK:
when we send a msg and systems are at different time zones, it is handled by logical clock.

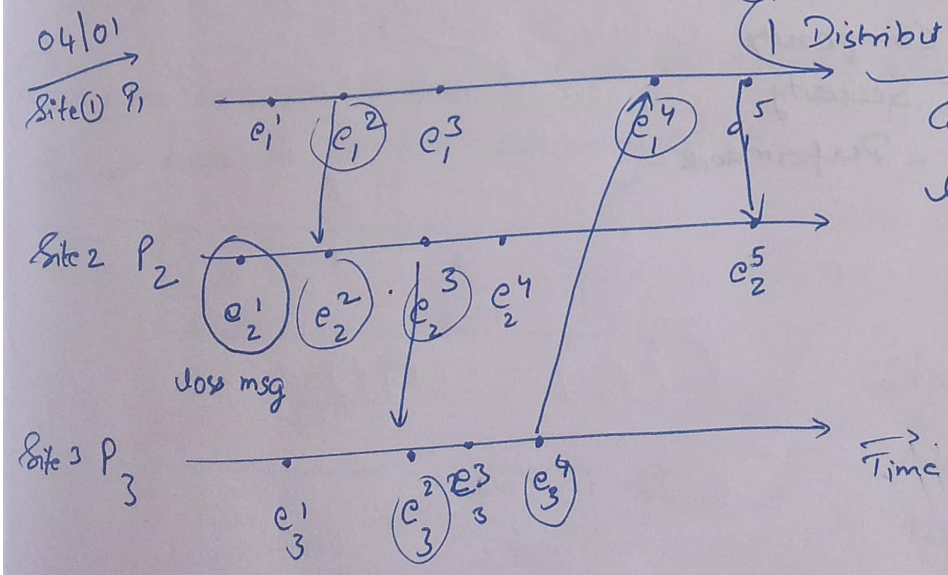
→ Double spending problems are caused in the distributed system. Soft data is used in double spending problem to over come this.

→ Available at anytime to store & retrive data is reliability.
→ There is a centralized servce to access the memory

Drawbacks of DS:
- Syncronyzation problems
- Increase in cost.

Distributed System → George Coulouris, Tim Kindberg
| Parllel Computing → Kai Huang.
| Andrew S. Tanendu
| Distributed Algorithm → Nancy Lynch

Can be handled using logical clock.



SPACETIME

Siken

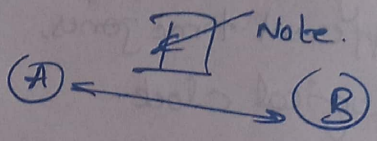
Characteristics of Distributed Systems:

- Message passing from node to node.
  - → distributed calls
  - → web services
- Into Processing Communication is handled via msg processing
-

→ CRITICAL SECTION
→ MUTUAL EXCLUSION

Applications of Distributed System:

→ Cloud computing
→ PEER TO PEER
→ Distributed architecture
→ Distributed laser
→ Web services



Google peers are distributed but server is centralized.

Characteristics

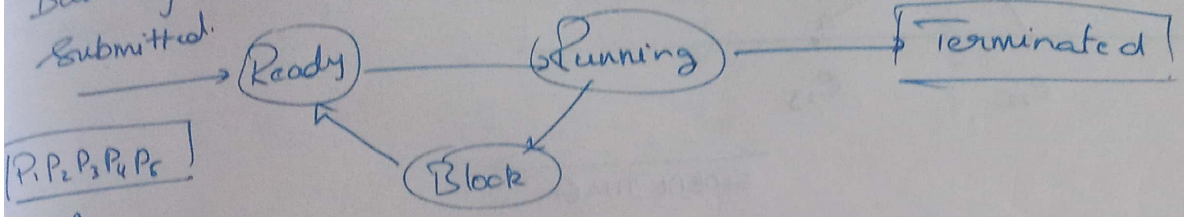
- Message Passing
- Remote Procedural Call
- Distributed Object
- Distributed Agent & active object

Advantage	disAdvantage
- Scalability	- Complexity
- Reliability	- Security
- Flexibility	- Performance

09/01/24

CONCEPT OF PROCESS STATE:

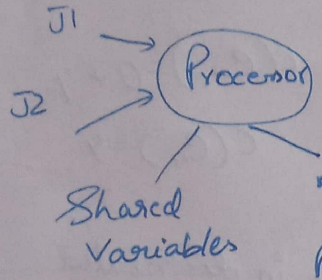
Process: Instance of Program.
Basically there are 4 states of a process.



queue of process in an OS

If the processor is free/in ideal cond^n a particular job can be assigned
If the i/p & o/p are avaible then they are sent to Ready state.

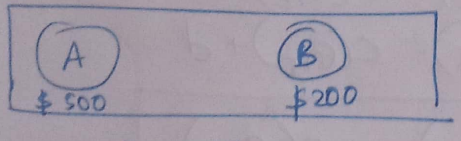
CONCURRENT PROCESS: A process that can run simultaneously without another process.



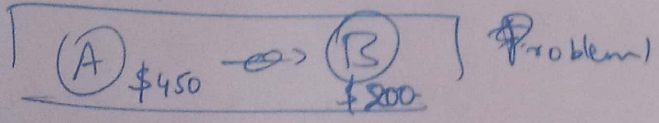
⇒ The way of handling concurrent processes.

" indirating the processor if the next process can be executed based on the avibbility of Resources by passing msgs.

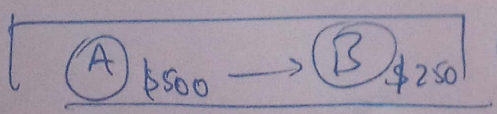
REALTIME PROBLEM IS CAUSED DUE TO TIME ZONES IN DISTRIBUTED SYSTEMS
So we have to go for syncronygation.

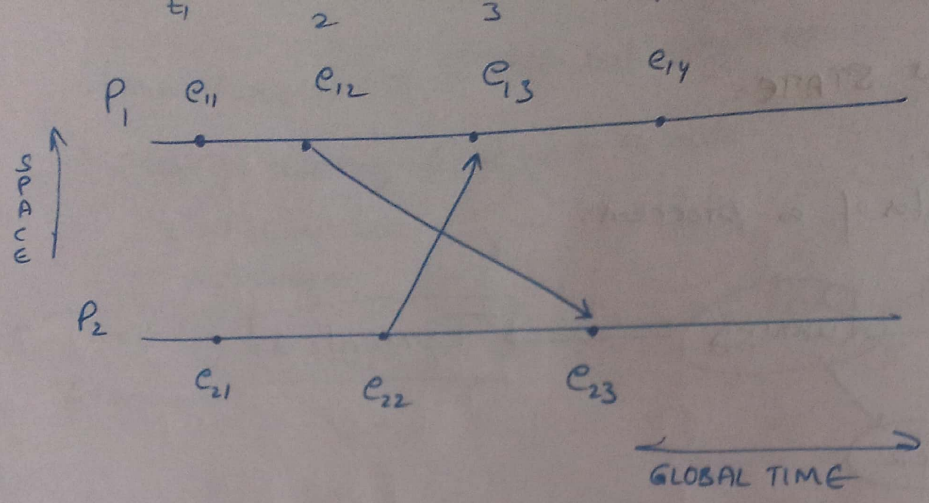


A → B
$50



These can be cleared using Lamport's Solution using logical clock.



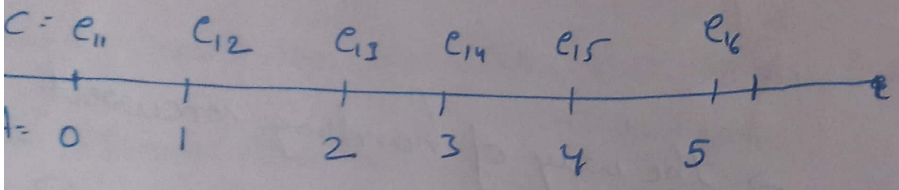


e ⇒ jobs executed by the processes.

In real-time processes are in increasing order. They can't be executed in reverse order i.e 1→2 never 2→1

$e_{11} = e_{21}$ $\quad e_{12} \le e_{23}$ $\quad e_{22} \le e_{13}$ } They are in right order

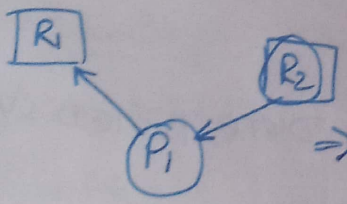
PROCESS P_1 $\qquad$ t = time



$C(e_{11}) = 0 + 4$

$C(e_{11}) = 4$

$C(e_{12}) < C(e_{15})$ ⇒ True always in distributed system.



⇒ Let us assume there is a delay of 4 seconds due to resource allocation for e_{11}

e_{12} = 5 sec $\quad e_{13}$ = 6 sec

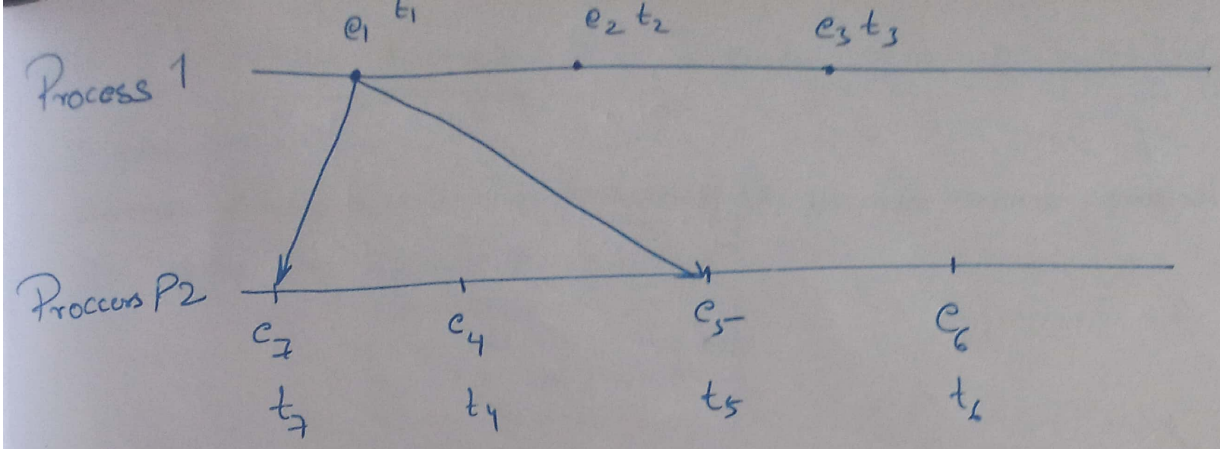
e_{15} = 8 sec $\quad e_{14}$ = 7 sec

e_{16} = 9 sec

$C(e_{12}) = C(e_{12}) + d$

$C(e_{15}) = C(e_{15}) + d$

$C(e_{12}) < C(e_{15})$

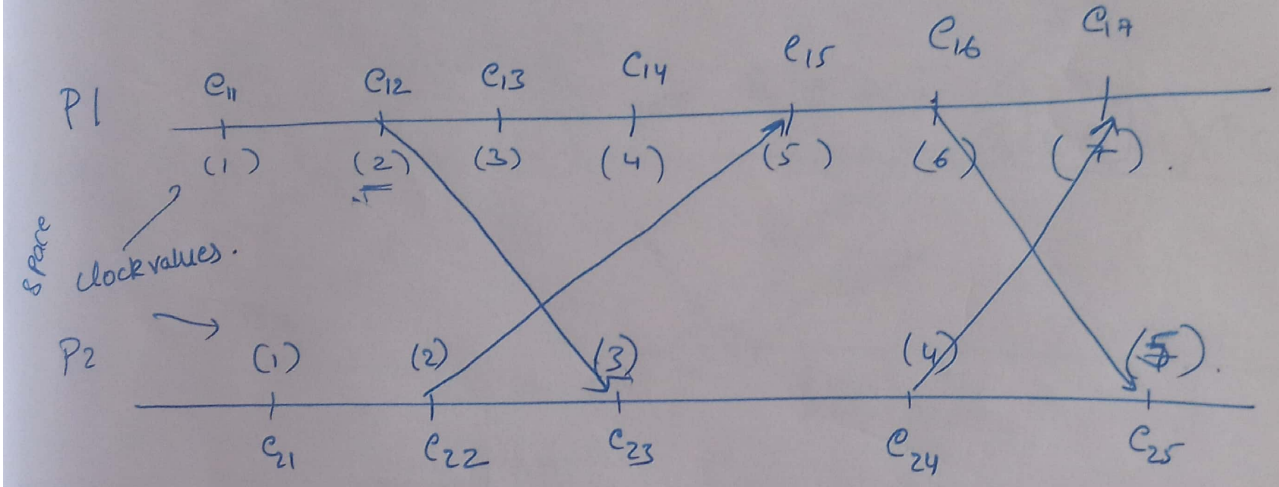


$C(e_1) = t$

~~$C(e_1) \leq C(e_1)$~~

$C(e_1) < C(e_5)$

$C(e_1) < C(e_7)$



$C(e_{12}) < (e_{23})$ $C(e_{24}) < (e_{17})$

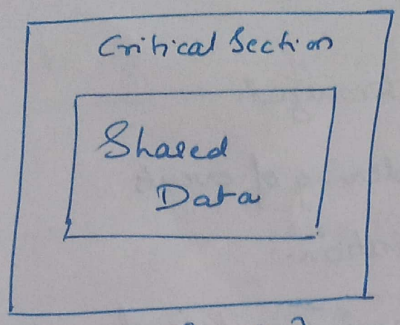
$C(e_{22}) < (e_{15})$

Global Time

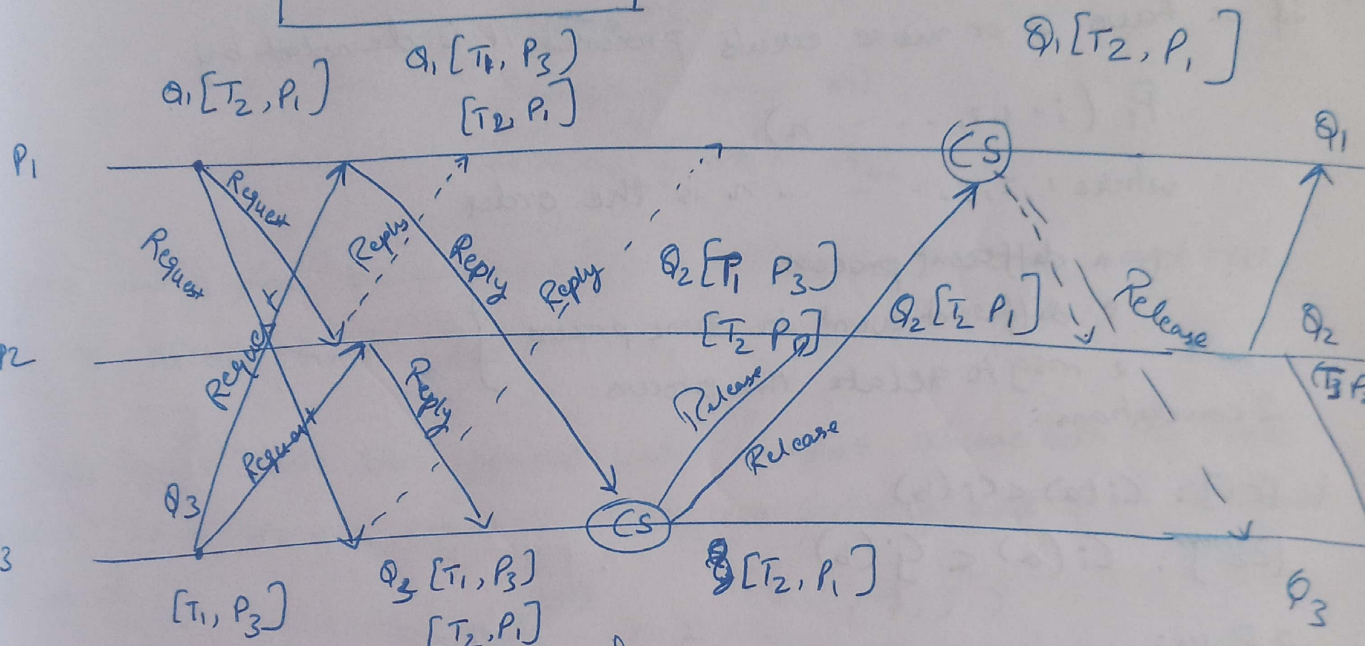
18/01

CRITICAL SECTION: It is a section which is accessible only to one process at a time.

MUTUAL EXCLUSION: It makes sure that process access shared resources or data in ~~serialized~~ serialised way.



$P_1 P_2 P_3 - - - P_n$



There exists a queue for each process known as buffer.

LAMPORT ALGORITHM:

1. P_i sends request to all P_j's
2. P_i make entry in its own queue
3. P_j receives request message
4. P_j makes entry in its own queue & send reply to P_i.
5. Condition for critical section.
   - P_i has received reply from all those P_j's whom he has sent request
   - In queue his request is the top most
6. P_i exits the [illegible] from queue
7. P_i broadcast release msg
8. On recieving msg P_i removes P_i's entry empty.

PERFORMANCE:

- 3(n-1) msg per Critical Section invocation.
- (n-1) reply
- (n-1) release
- (n-1) request.

LAMPORT

- Physical clock - Needs to be synchronized.
- Instead of it, we can go for ordering of events
- '$\longrightarrow$' happened before relation.
- If we have 2 or more events processes, it is denoted by

P_i $(i = 1, 2, - - - n)$

where 1, 2, 3, - - - - n is the order

{ * different process
* different event in same process
* msg to relate the process } 2 conditions:

[C1]: $C_i(a) < C_i(b)$

[C2]: $C_i(a) < C_j(b)$

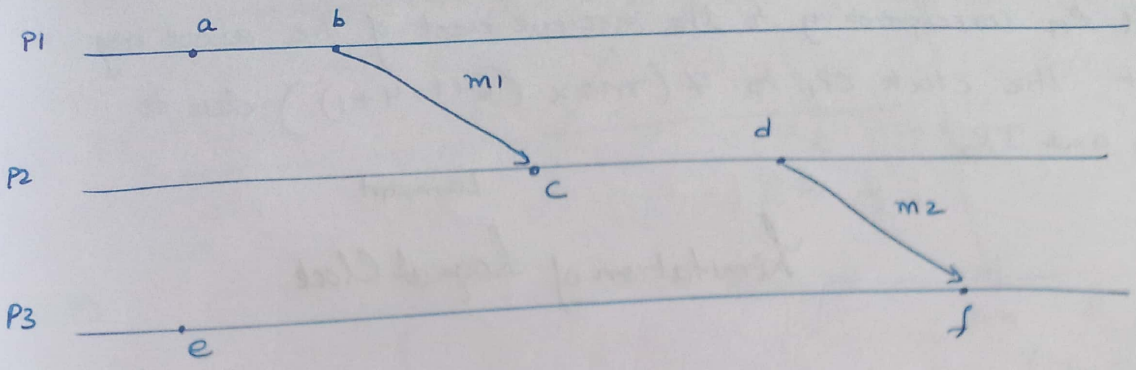
2 Rules:

[IR_1]: if $a \longrightarrow b$

$C_i(b) = C_i(a) + d$ $d > 0$

[IR_2]: $C_j = \max(C_j, t_m + d)$ $d > 0$

Example:



$a \longrightarrow b$ Process P1

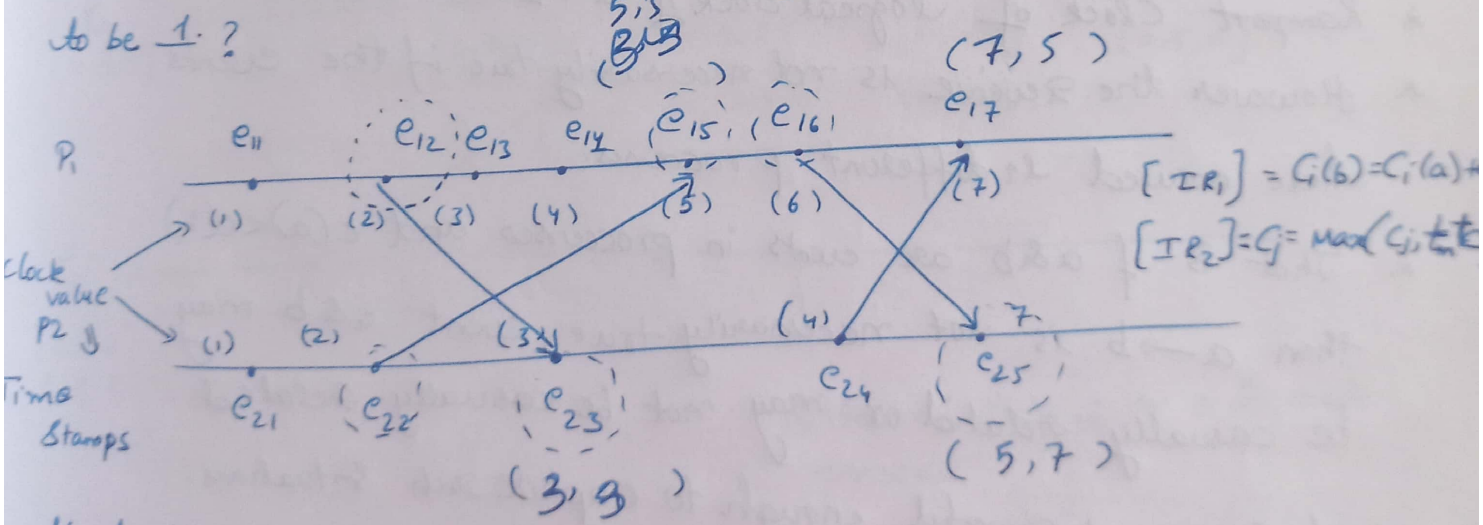
$c \longrightarrow d$ Process P2

$b \longrightarrow c$ because of msg m1

$d \longrightarrow f$ because of msg m2

$a \not\rightarrow e$ and $e \not\rightarrow a$ because a&e are different process and no msg to relate them alle

Ex: How logical clocks are updated under Lamport scheme both the clock value CP_1 & CP_2 are assumed to be zero initially and d is assumed to be 1. ?



- If d is not give assume d to be 1

$C_{e_{11}} = 1+1 = 2$

$C_{e_{12}} = 2+1 = 3$

$C_{e_{13}} = 3+1 = 4$

$C_{e_{14}} = 4+1 = 5$

$C_{e_{15}} = 5+1 = 6$

$C_{e_{16}} = 6+1 = 7$

$C_{e_{17}} = 7+1 = 8$

$C_{e_{21}} = 1+1 = 2$

$C_{e_{22}} = 2+1 = 3$

$C_{e_{23}} = 3+1 = 4$

$C_{e_{24}} = 4+1 = 5$

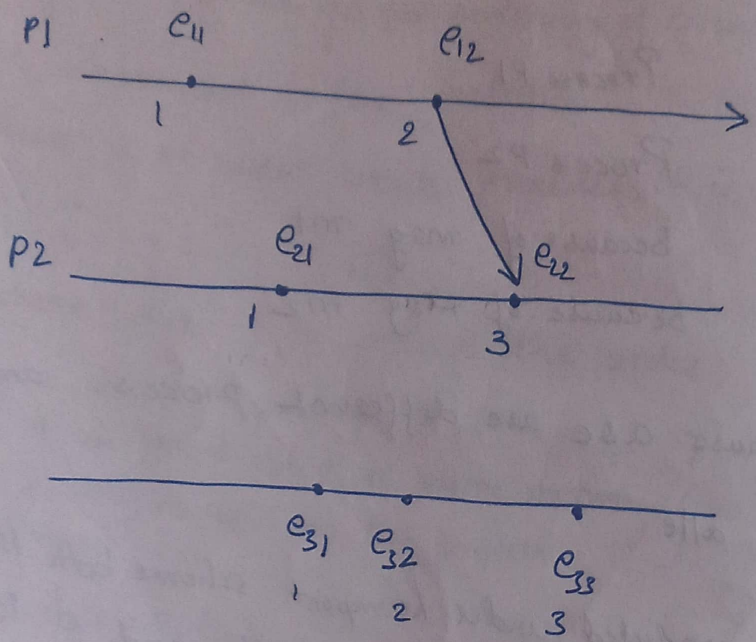
$e_{e_{25}} = \max(7, 5+1) = 7$

The msg is assigned a time stamp = 4
The event e_{17} corresponding to the recieve event of the above msg increment the clock CP_1 to 7 (max (6+1, 4+1)) due to rule IR_1 and IR_2.

20/01

Lamport

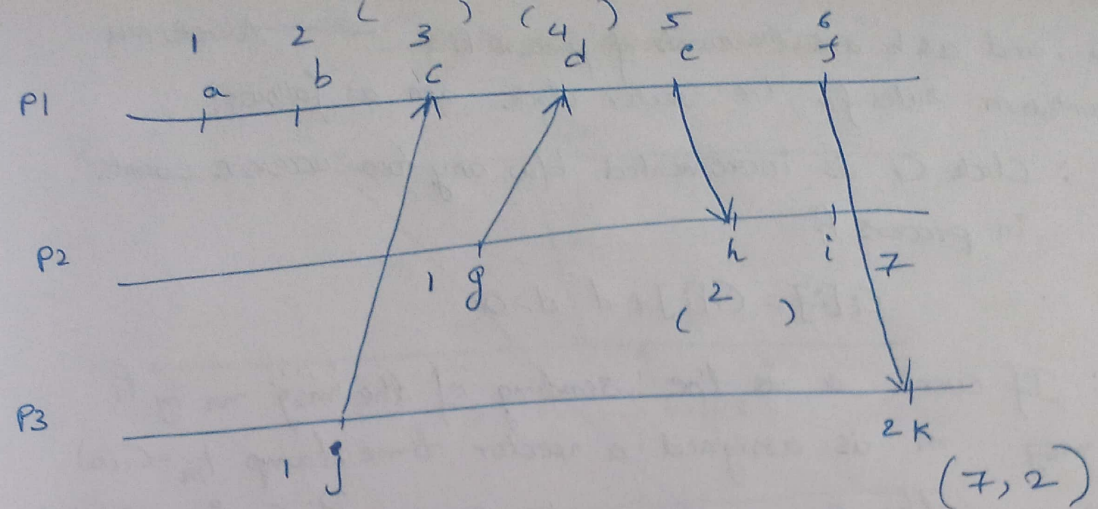
Limitation of ~~Logical~~ Clock



* Lamport Clock of logical clock if $a \rightarrow b$ then $C(a) < C(b)$
* However the reverse is not necessarily true if the events here occured in different proccesses.
* That is if a & b are events in proccesses and $C(a) < C(b)$ then $a \rightarrow b$ is not necessarily true; event a & b may be casually related or may not be casually related.
* L-C is not powerful enough to capture such situations.

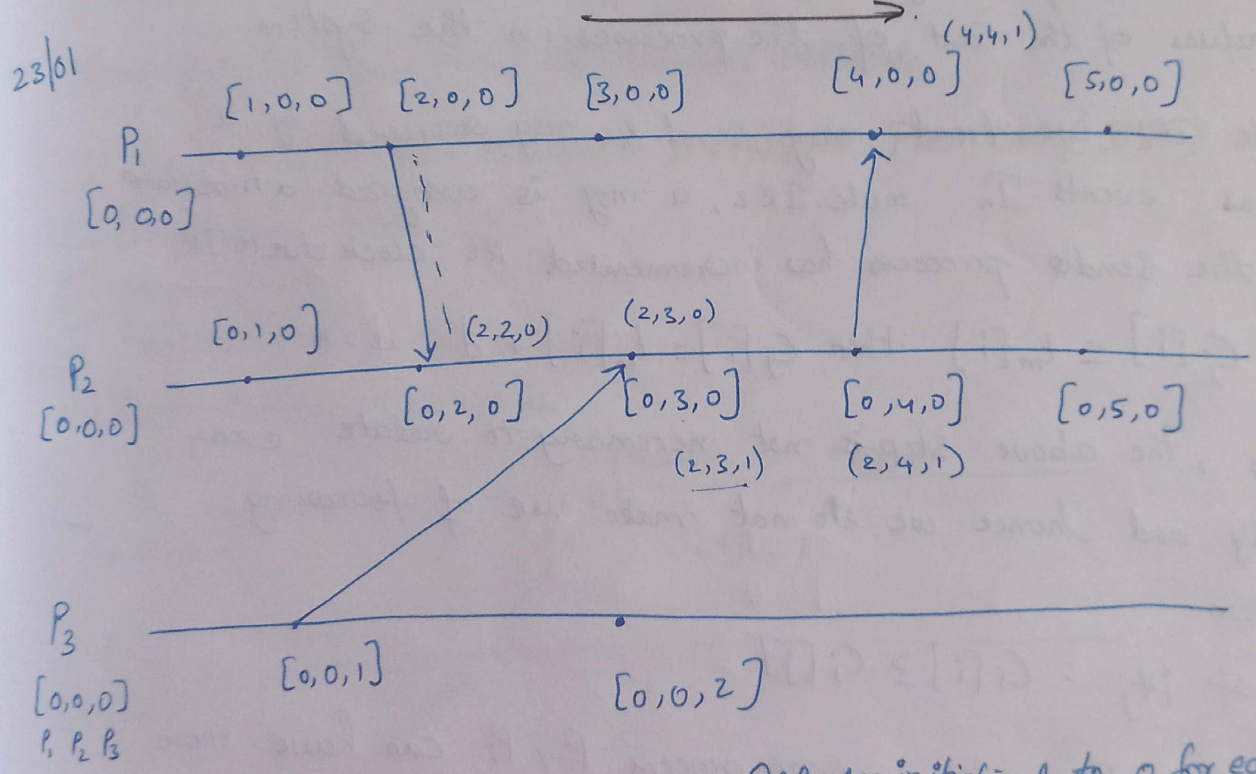
[IR1] if $a \rightarrow b$ $C(a) < C(b)$ True

[IR2] if $a \rightarrow b$ $C(a) < C(b)$ may not be true.



(3, 2) (4, 2) (6, 2)

VECTOR CLOCK



① Vector initialized to 0 for each process

$V_i[j] = 0$ for 1, 2, 3 - - -

② Increment vector before time stamp

$V_i[i] = V_i[i] + 1$

③ msg is sent from P_i with V_i do it

when P_j receive msg

$V_j[i] = \max(V_j[i], V_j[i])$

* If $a \rightarrow b$, and a & b are the events of process P_i ⟵ ALGORITHM
* Implementation rules for the vector clock are as follows:

$[IR_1]$: Clock C_i is incremented b/w any two successive events in process P_i

$$C_i[i] = C_i[i] + d \quad d > 0$$

$[IR_2]$: If event a is the sending of the msg m by P_i then msg m is assigned a vector time stamp $t_m = C_i(a)$ on accieving the same msg m by process P_i, C_j is updated as follows:

$$\forall K : C_j[K] = \max(C_j[k], t_m[K])$$

on the receipt of msg, a process learns about the more recent clock values of the rest of the processes in the system.

In rule IR1, we treat msg send & msg accieved by a process as events. In rule IR 2, a msg is assigned a timestamp after the sender process has incremented its clock due to IR_1

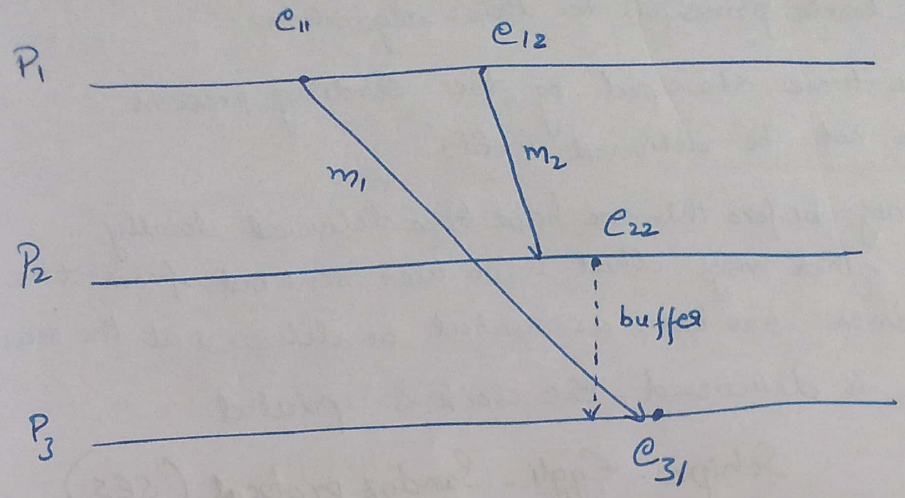
$$\text{if } C_j[i] \leq t_m[i] \text{ then } C_j[i] = t_m[i] + d \quad d > 0$$

However, the above step is not necessary to relate events casually and hence we do not make use of following discussion.

$$\forall i \neq j : C_i[i] \geq C_j[j]$$

e proof is obvious because no process $P_j \neq P_i$ can have more to date knowledge about the clock, value of process i'd clocks e monotonically non-decreasing.

CASUAL ORDERING of MSG:

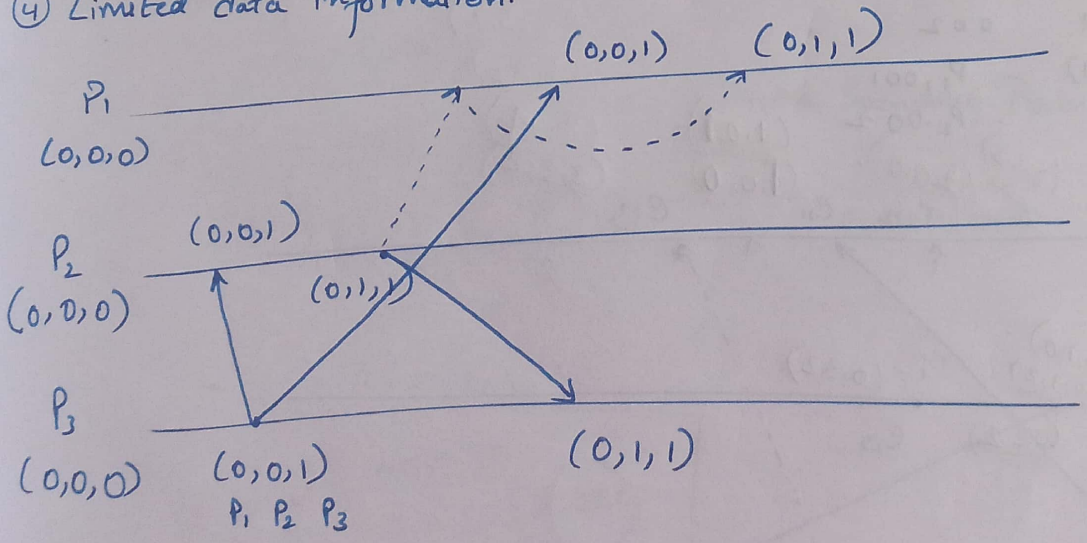


It deals with the notation of maintaining the casual relationship that holds among "message send" events with the corresponding "message recieves" event.

Send (m_1) & Send (m_3) are casually related.

BIRMAN - SCHIPER - STEPNSON PROTOCAL (BSS)

① Broadcast based
② more msg
③ Smaller size for the usage
④ Limited data information.



① Each procers increment its vector clock (by 1)
② Delivery of same msg by P_j until:
P_j recieved all msg from P_i that proceed by m
P_j recieved all msg by P_i before sending m
③ P_i update vector clock

24/01

BSS (uses broadcasting)

There are three basic principals to this algorithm.

① All msgs are time stamped by the sending process

② A msg can not be delivered until:

→ All the msgs before this one have been delivered locally

→ All the other msgs that have been sent out from the original processes has been accounted as delivered at the receiving proc

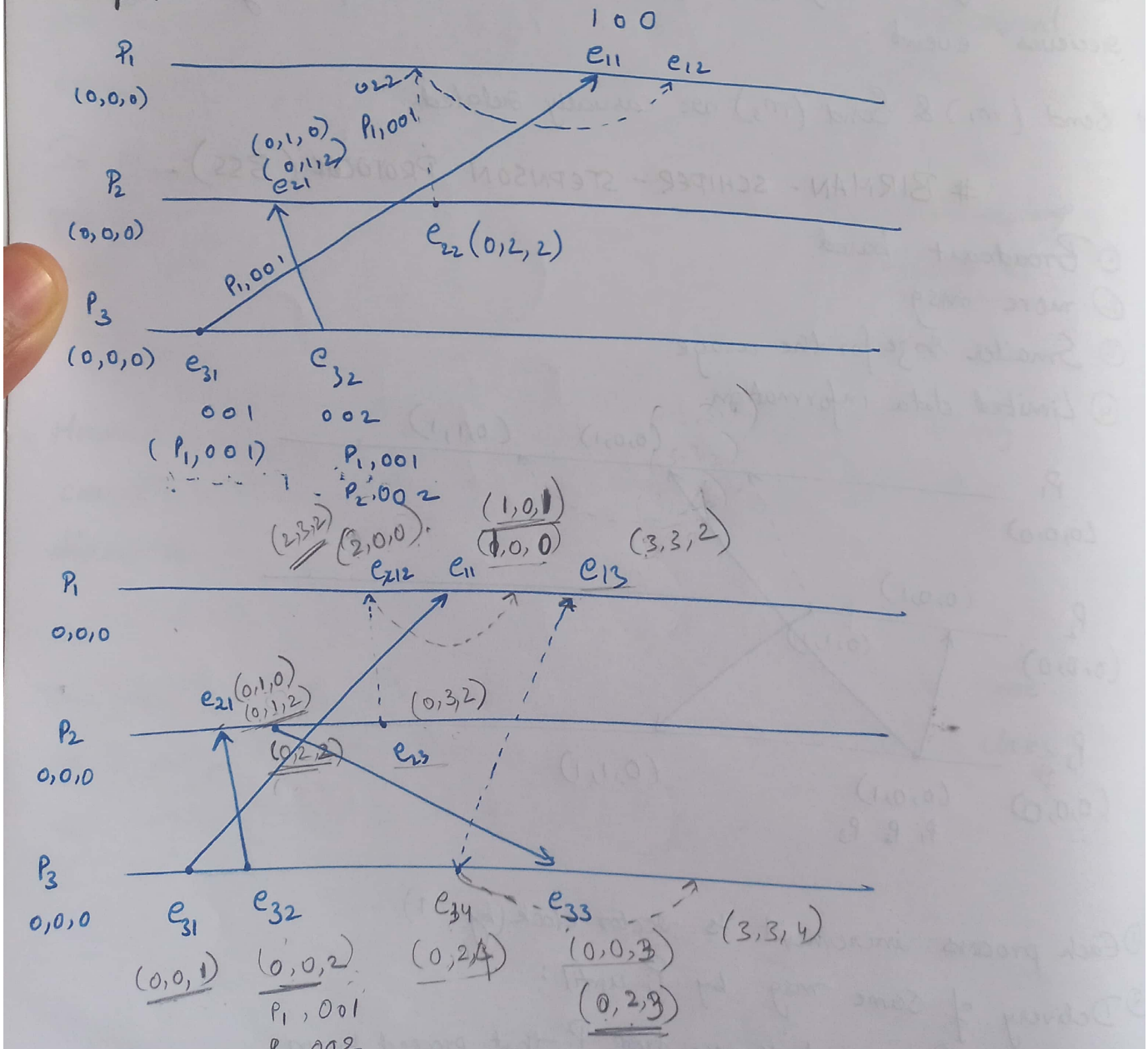
③ When a msg is delivered, the clock is updated.

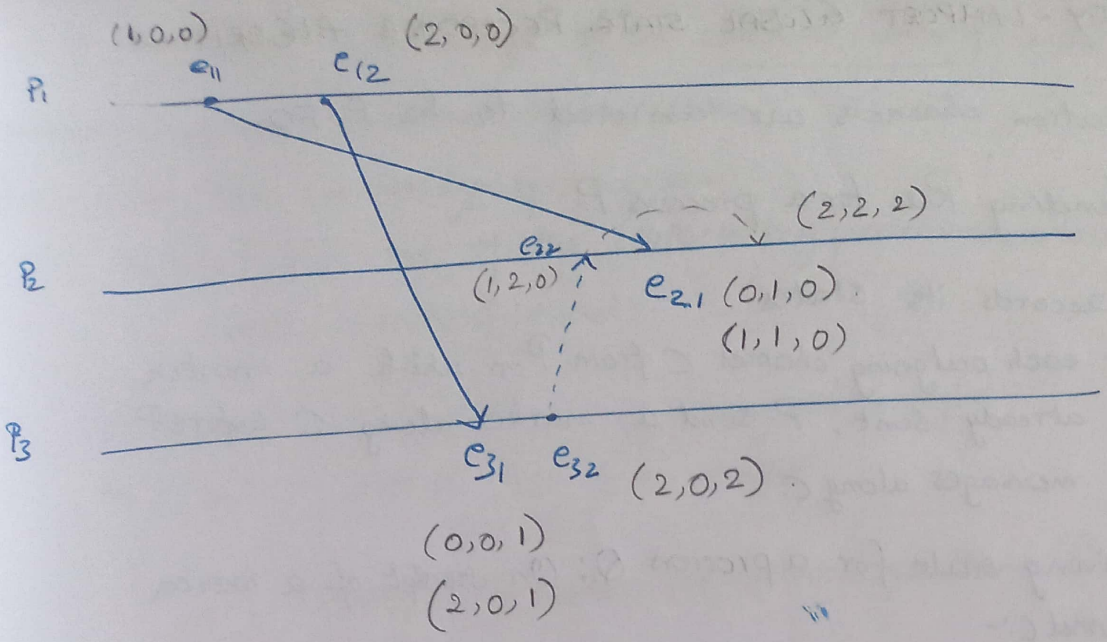
Schiper - Eggli - Sandoz protocal (SES)

⇓

(do not use broadcast)

Example:





Global State:

It is a set of local states of all individual processes

$$R = \{P_1 - - - P_n\}$$

Need for global clock:

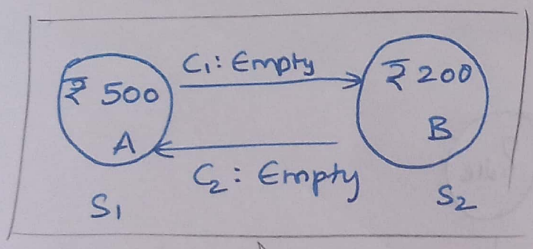
1. Distributed dead lock detection
2. Termination detection
3. Check point.

25/01

① Distributed

Ex:

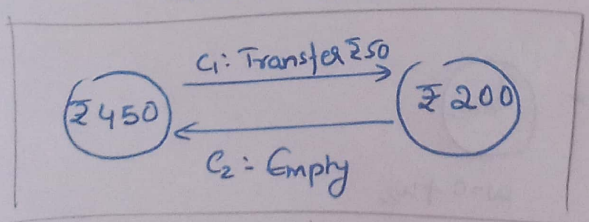
Global State 1



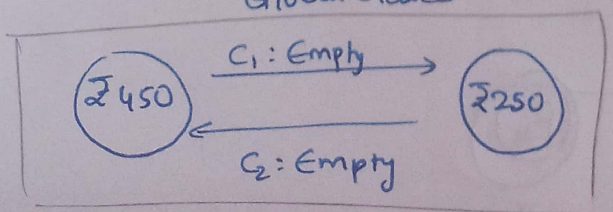
Double Spending Problem ?

FLP Problems.

Global State 2



Global State 3



CHANDY-LAMPORT GLOBAL STATE RECORDING ALGORITHM:

The communication channels are assured to be FIFO

① Marker Sending Rule for a proccess P

- P records its states
- for each outgoing channel C from P on which a marker has not been already sent, P send a marker along C before P sends further messages along C.

② Marker receiving rule for a proccess Q: On receipt of a marker along a channel C:-

if Q has not recorded its state then

begin

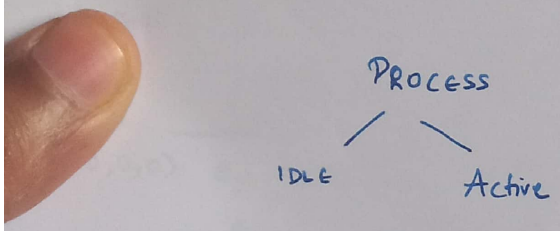
Record the state of C as empty sequence follow the "marker sending rule"

end

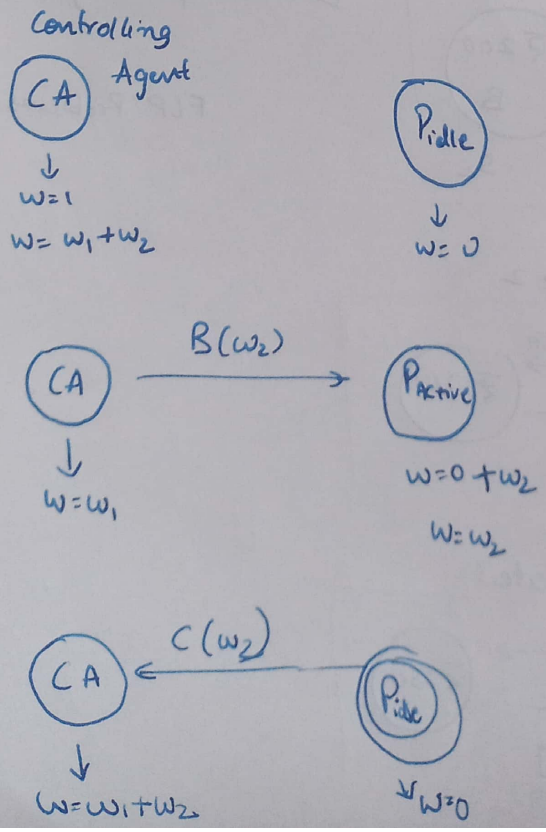
else

Record the state of C as the sequence of msg's received along C, after Q's state was recorded and before Q receives the marker along C

TERMINATION DETECTION ALGORITHM:



Ex:



* It is an example of the usage of the "coherent view" of a distributed system.

coherent view: global consistent system/states.

BASIC IDEA: There are two type of processes.

active or idle one of the cooperating process monitors the computation is called controlling agent.

(i) $B(w_i)$ → Computation msg Along with weight.

(ii) $C(w_i)$ → Control msg with weight.

Huang Termination Detection Algorithm. Based on 4 Rules:

RULE 1: The controlling agent having weight W may send a computation msg to a process P.

$W = W_1 + W_2$, $W_1 > 0 : W_2 > 0$

$W = W_1$

Send $B(W_2)$ to P_1

RULE 2: On Recipt of $B(w_2)$ a process P having weight, $W = W + W_2$ if P idle, P become active

RULE 3: An active process having weight w may become idle send $C(w_2)$ to the controlling agent , $W = 0$

RULE 4: On Recieving $C(w_2)$ the controlling will have weight $W = W_1 + W_2 = 1$

↓

Computation Terminated.