Module 1
Introduction

- A distributed system is one which located at n/w computers communicate and coordinate their actions by only passing msg.

→ Characteristics of D.S.

(i) Concurrency of components
(ii) Lack of a global clock
(iii) Independent failure of components
(iv) multiple autonomous components
(v) Resources may not be accessible to all
(vi) Multiple points of failure
(vii) Multiple points of control
(viii) S/w runs on concurrent processes on diff processes

→ Examples of D.S's

(i) LAN and internet
(ii) DBMS
(iii) ATM N/W
(iv) WWW
(v) Mobile and unique computing.

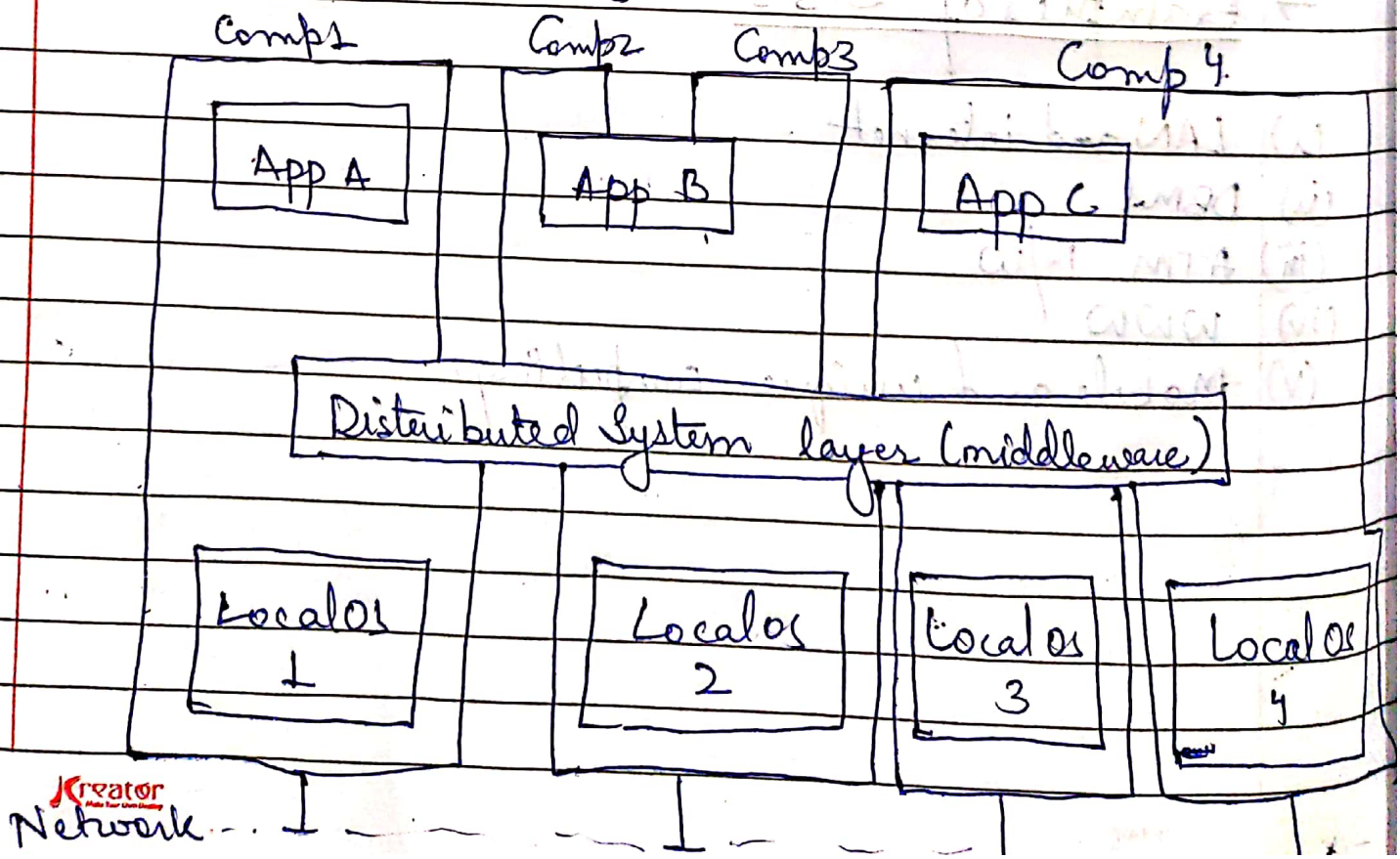
⇒ Distributed System

- A distributed system is a collection of independent systems that appears to all users as a single coherent system. The various important aspect about this defination of D.S are.

(i) D.S. consists of numerous autonomous system or computers. It means that the configuration of the system ~~mak~~ may be different while designing any D.S.

(ii) As the whole D.S. seems like a coherent system, the users are going to work by colaborating with each other.

(iii) The block diagram describing the structure of a simple D.S. can be given as,



- In the above architecture we have 3 applic^{n} and 4 O.s. The app B is distributed across comp 2 and comp 3. The main point about this architecture is that each app is offered the same interface.

⇒ **Goals of D.S (Needs of D.S)**

- **Resource Sharing:-** The main goal of distributed system is to provide the sharing of various resources in a controlled and efficient way. Resources can be printers, computer, webpages, etc.

- **Distribution (transparency):-** An important goal of a D.S is to hide the fact that it processes and resources are physically distributed across multiple computers. A D.S which is able to present itself to users and application as if it were only a single computer system is said to be transparent. The various types of transparency that can be applied to any distributed system are

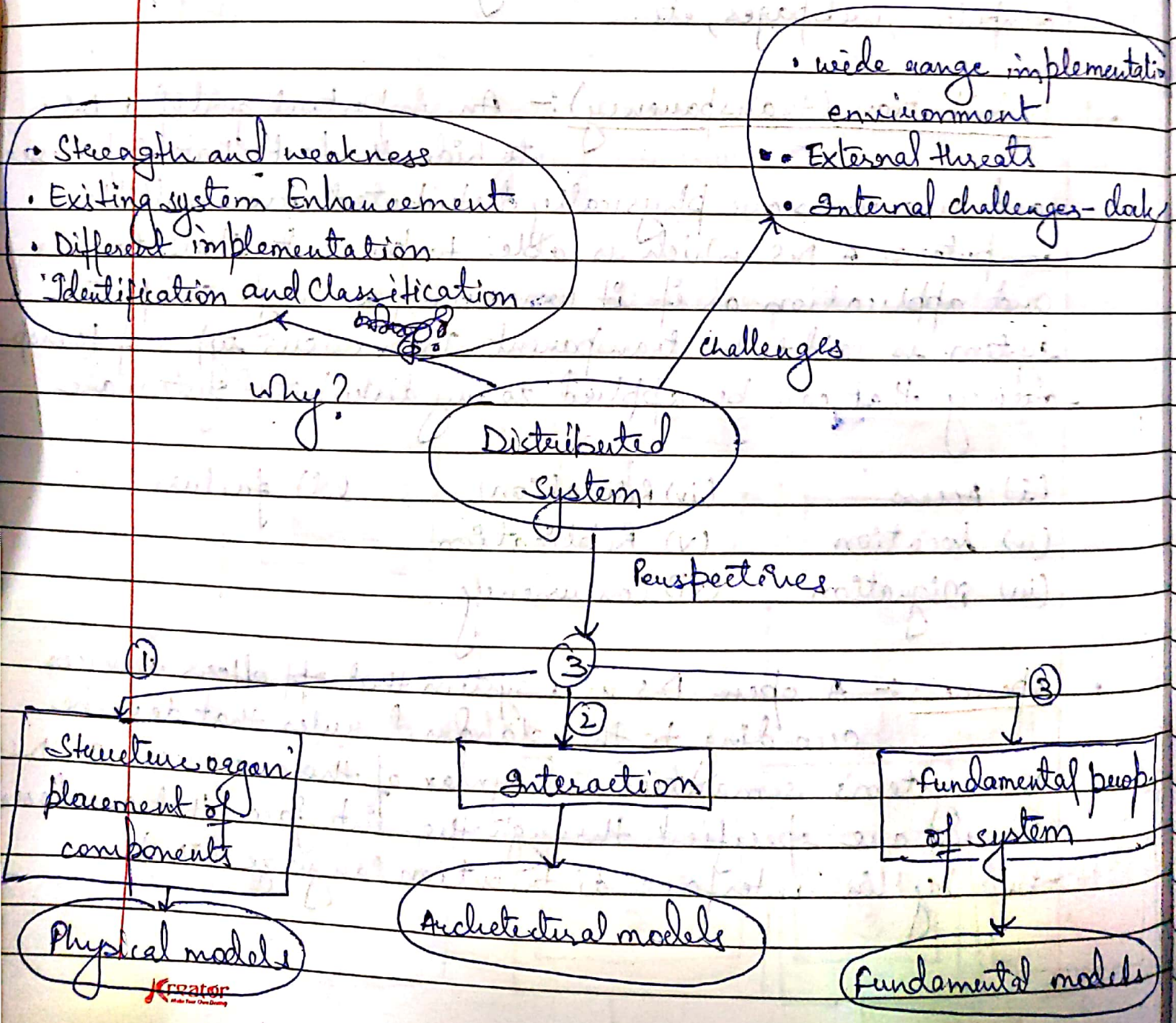
(i) Access
(ii) location
(iii) Migration
(iv) Relocation
(v) Replication
(vi) Concurrency
(vi) failure.

- **Openness:-** A open D.S is a system that offers services according to the standard rules that describe the system's semantics and syntax of those services. In D.S service are specified through the interface that are described by in interface distribution language (IDL)

- **Scalability** :- It refers to the extensions of the existing system for making it more efficient. It means that the design of D.S. must be like that of more application on system are added then in that case also it will work efficiently.

⇒ **D.S. models** (imp)

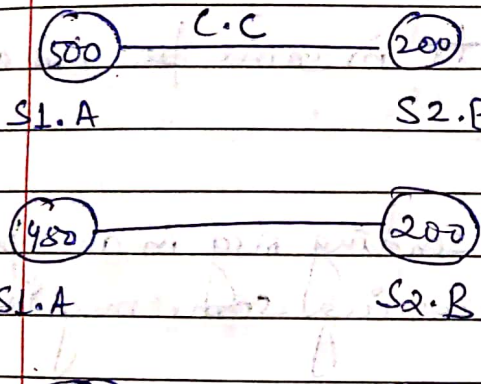
★ Read sir ppt and the diagrams below



⇒ D.s limitations

- Absence of shared Memory (No visibility of systems actual state)
- Absence of Global clock (No common clock)
  - Common Global clock
    - Posibility to have diff value of clock for diff processes due to transmission delay
  - Separate P.C
    - Probs
    - Sync issue

eg



- Here we send 50 from acc A to acc B, but it doesn't get updated hence causing mismatch.

⇒ Lamports Logical Clock

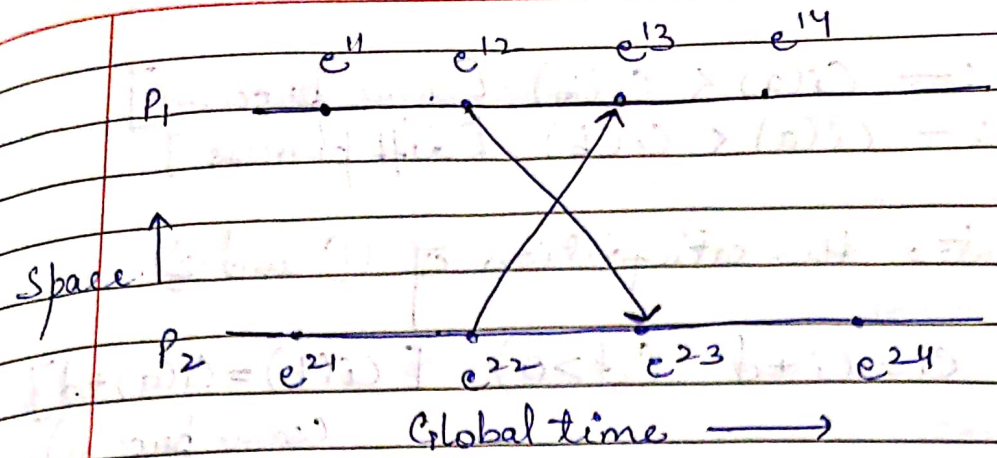
- Process → Execution → Sequence of Events
  - eg 1.) Procedure
    2.) Introduction
    3.) Message exchange
- A happend before relation is introduced (→). It shows casuall dependency b/w events.
- a → b, if a and b are events in same process and 'a' occured before b
- a → b, if a is one event of sending msg in any process and b is event of recieving some msg in any process
- if a → b and b → c then a → c (transitive)

Casually related events → (a → b)

Concurrent events → (if a ↛ b and b ↛ a)

(a || b)

★ Also take notes from sirs book.



- we have 2 events P_1 and P_2 and 4 events for each events process.

$$\begin{bmatrix} e^{12} \longrightarrow e^{23} \\ e^{22} \longrightarrow e^{13} \end{bmatrix} \leftarrow \text{Casually related events}$$

$$[\, e^{11} \,||\, e^{21} \,] \leftarrow \text{concurrent events}$$

- All events before a happen before are concurrent events
- Limitation is that after happen before we cannot predict the concurrency of events.

(P_i) Process

(C_i) clock

$C_i(a) \leftarrow$ timestamp

(a) event

- Conditions specified by systems of clock are:

if $a \rightarrow b$ then $C_i(a) < C_i(b)$

Case (1) :— $C_i(a) < C_i(b)$ [same process]
(2) :— $C_i(a) < C_i(b)$ [diff process]

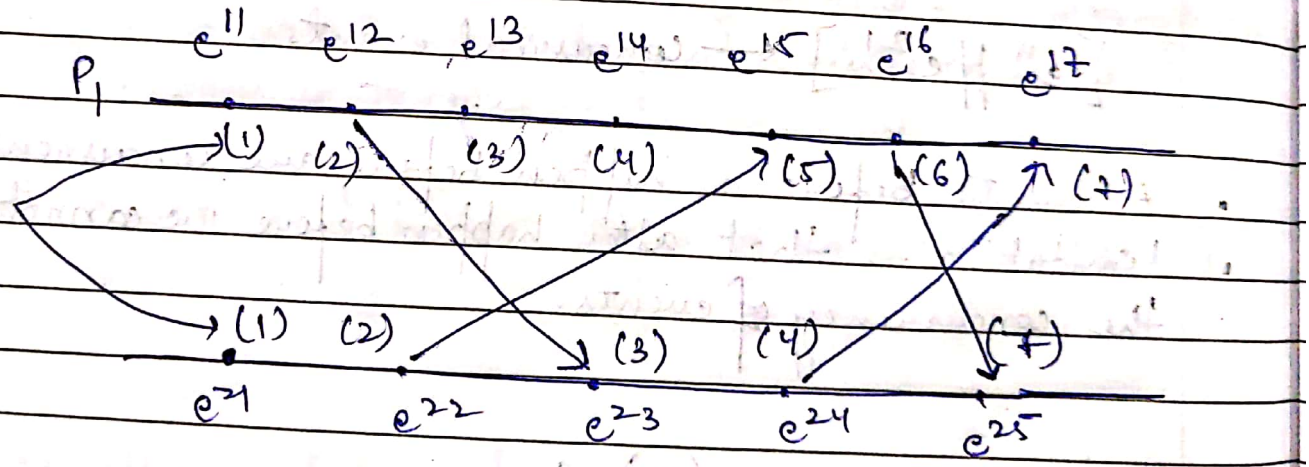
→ How to guarantee the satisfaction of (1) and (2)

Rule (1) :- $C_i = C_i + d$ $(d > 0)$ $[C_i(b) = C_i(a) + d]$ (same process)

(2) :- $C_j = \max(C_j, t_m + d)$ $(d > 0)$ [diff process]

Here,
$m = msg$, $t_m = C_i(a)$ (timestamp of a)

eg :-



- first value is assumed 0 or 1. to be 1 and assuming $d = 1$ for all conditions.
- All cases before happen before are incremented by 1., so e^{12}, e^{22} gets (2)

DATE
PAGE No.

- New for relation,

  $e^{12} \longrightarrow e^{23}$, we can have (3, 3) so acc. to rule 2 we add max(a, b)'s values so (3)

  $e^{22} \longrightarrow e^{15}$, we can have (5 or 3) so (5) is added to e^{15}

  $e^{16} \longrightarrow e^{25}$, we have (5, 7) so (7) is added.

  $e^{24} \longrightarrow e^{17}$, we have (7, 5) so, (7) is added.

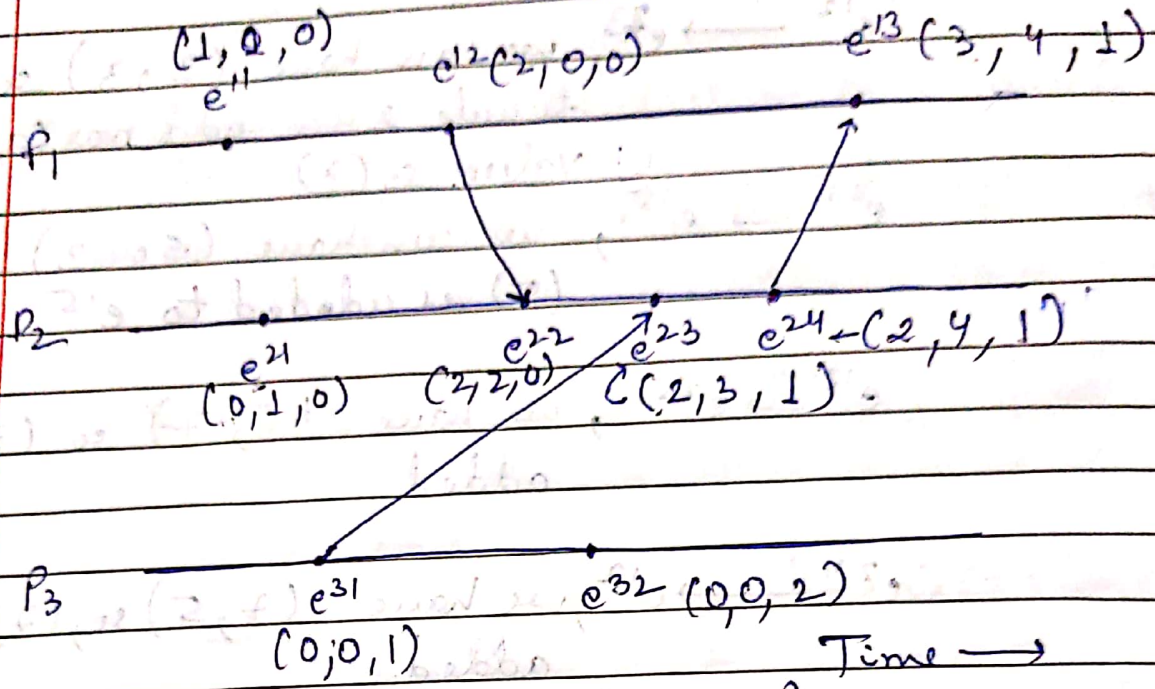
→ <u>limitations of LLC</u>: Concurrency of events after happen before relation cannot be checked

eg $a \rightarrow b$ then

$C_i(a) < C_i(b)$ but $C_i(b) < C_i(a)$: Not necessary.

* do example from book.

⇒ Vector clock

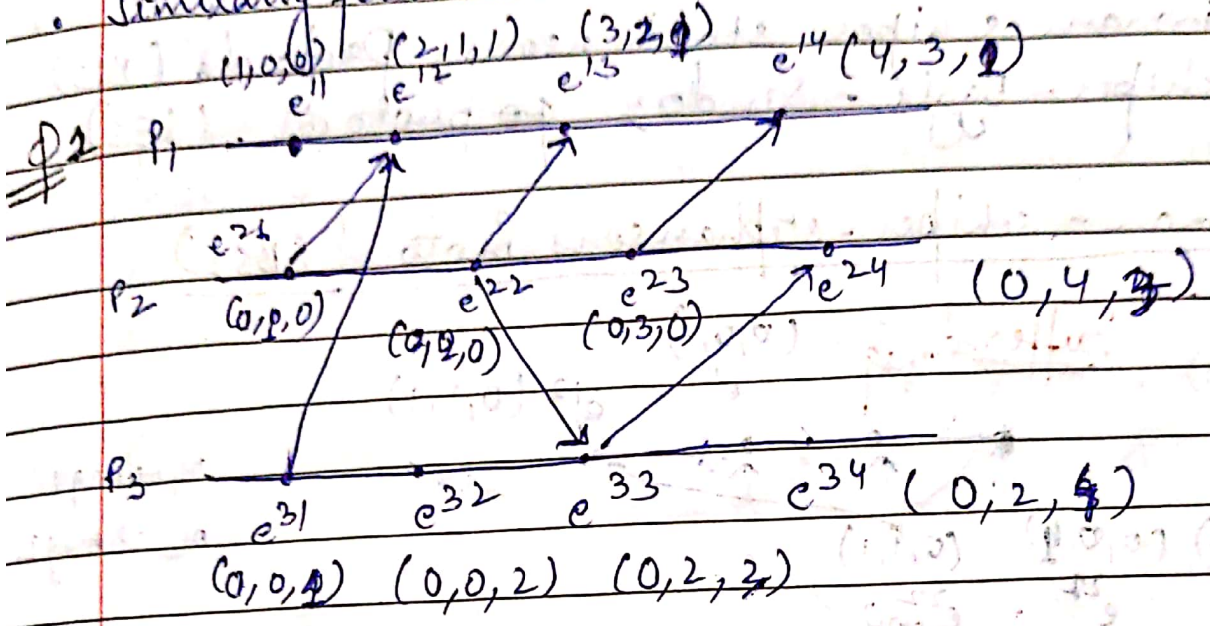


- Intial clock value is (0, 0, 0)
- In vector clock the clock values can be more than on for an even depending on processes while this is not so in lamports clock
- So 3 process in above eg so three values of clock at each processor.
- Updation occurs same as lamports clock
- e^{11} has (1, 0, 0) as no connect^{n} to P_2 and P_3 same for e^{21} and e^{31}
- for e^{12}, 1 clock val is increased so (2, 0, 0) same for e^{32} (0, 0, 2)

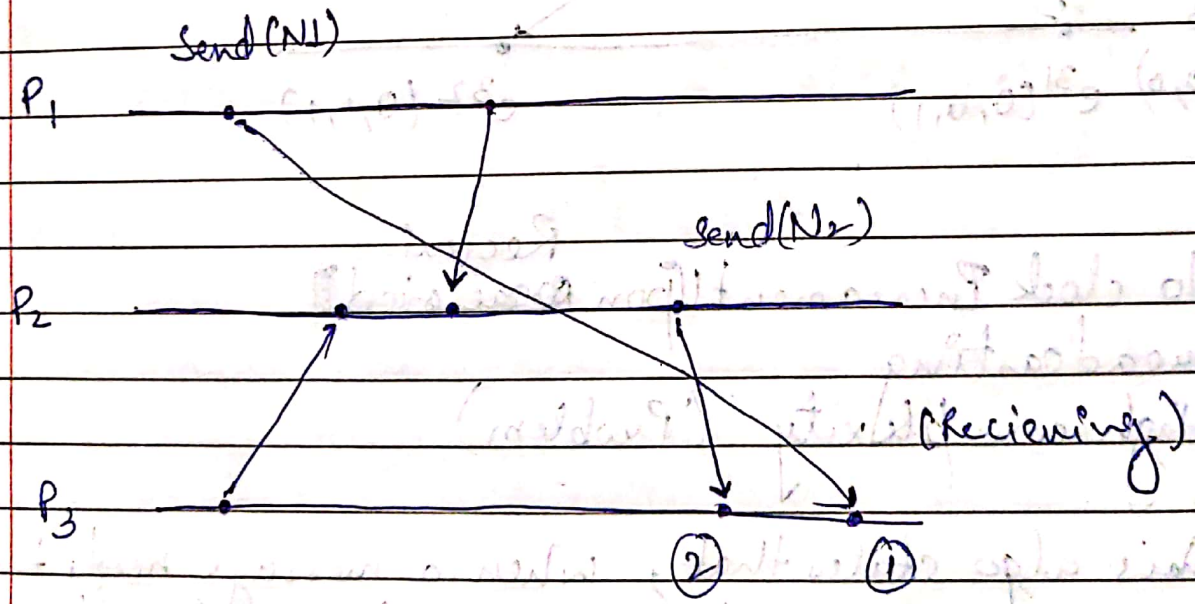
$e^{12} \longrightarrow e^{22}$ have a happened befor relation, now at e^{22} we have (2, 0, 0) and (0, 2, 0), we compare these and

set bigger value and set the individual values so (2,2,0) is set.

- Similarly for all other nodes.



⇒ Casual Odering of messages



- According to this diagram N1 is sent first and N2 later but according to diagram N2 is recieved first at P3, but this cannot happen in distributed sys.
- N2 is kept is buffer till N1 is recieved. Then N2 is

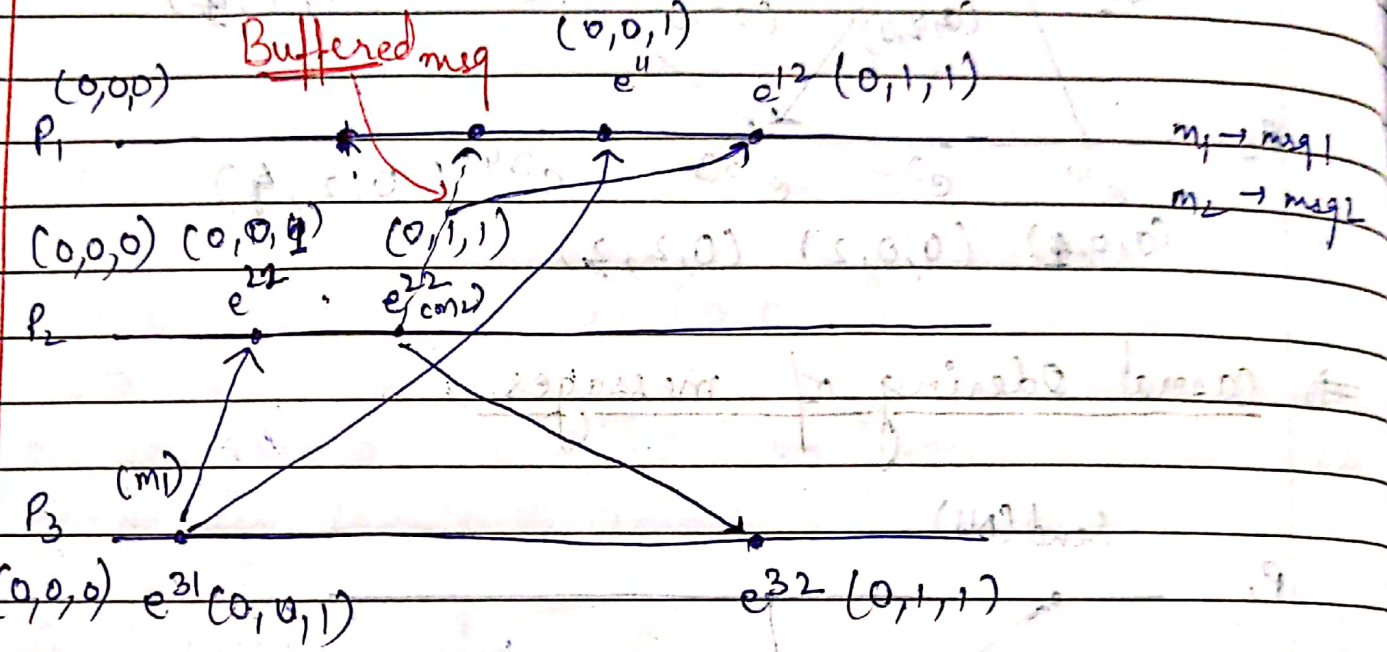
recieved.

- To maintain this we use.

(1) Birman schiper stephensons protocols (BSS)
(2) schiper - Eggli - sandoz protocol (SES)

⇒ Birman - schiper - stephensons Protocol (BSS)

eg



- No clock Increament upon Recieving
- Broadcasting
- High complexity. (Problem).

- This algo states that, when a message needs to be sent, it is not sent to another process directly, rather it is broadcasted to all process.

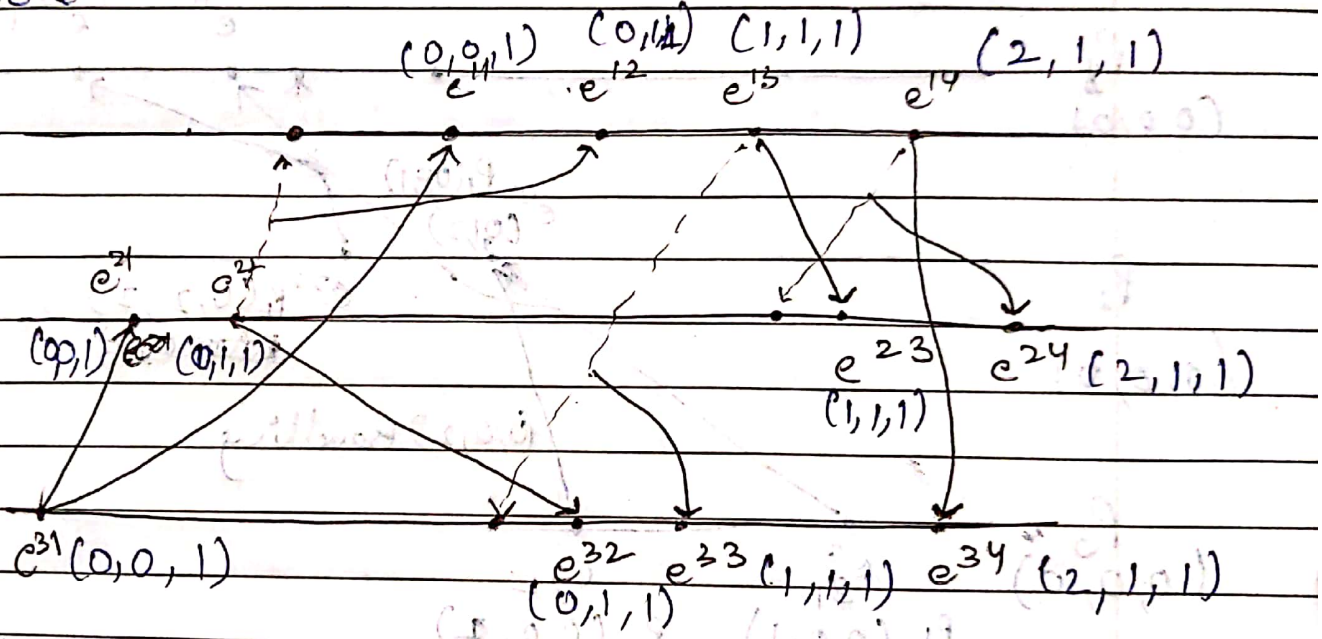
eg. $P_1 \xrightarrow[\text{message}]{\text{sends}} P_2$, so P_3 also recieves message.

∴ Clock incrementing only on msg sending and not msg recieving.

- msg 1 from e^{31} is sent to both P_2 and P_3 giving value (0,0,1) here 1 shows that 1 msg is passed.

- Now e^{22} schds msg 2 to P_1 and P_3 giving value (0,1,1) to e^{12} and e^{32}, but P_1 cannot recieve as 2 msg are being passed at once this is not possible. So first e^{11} recieves msg then e^{12} recieves msg 2, till then it is buffered. e^{32} can recieve msg 2 as only 1 value changes from e^{31} to e^{32}

Q1



Find Clock values for each event.

- Broadcasting based, even when a process P_i has to send a message only to another process P_j, the message has to be sent to all processes in the system.

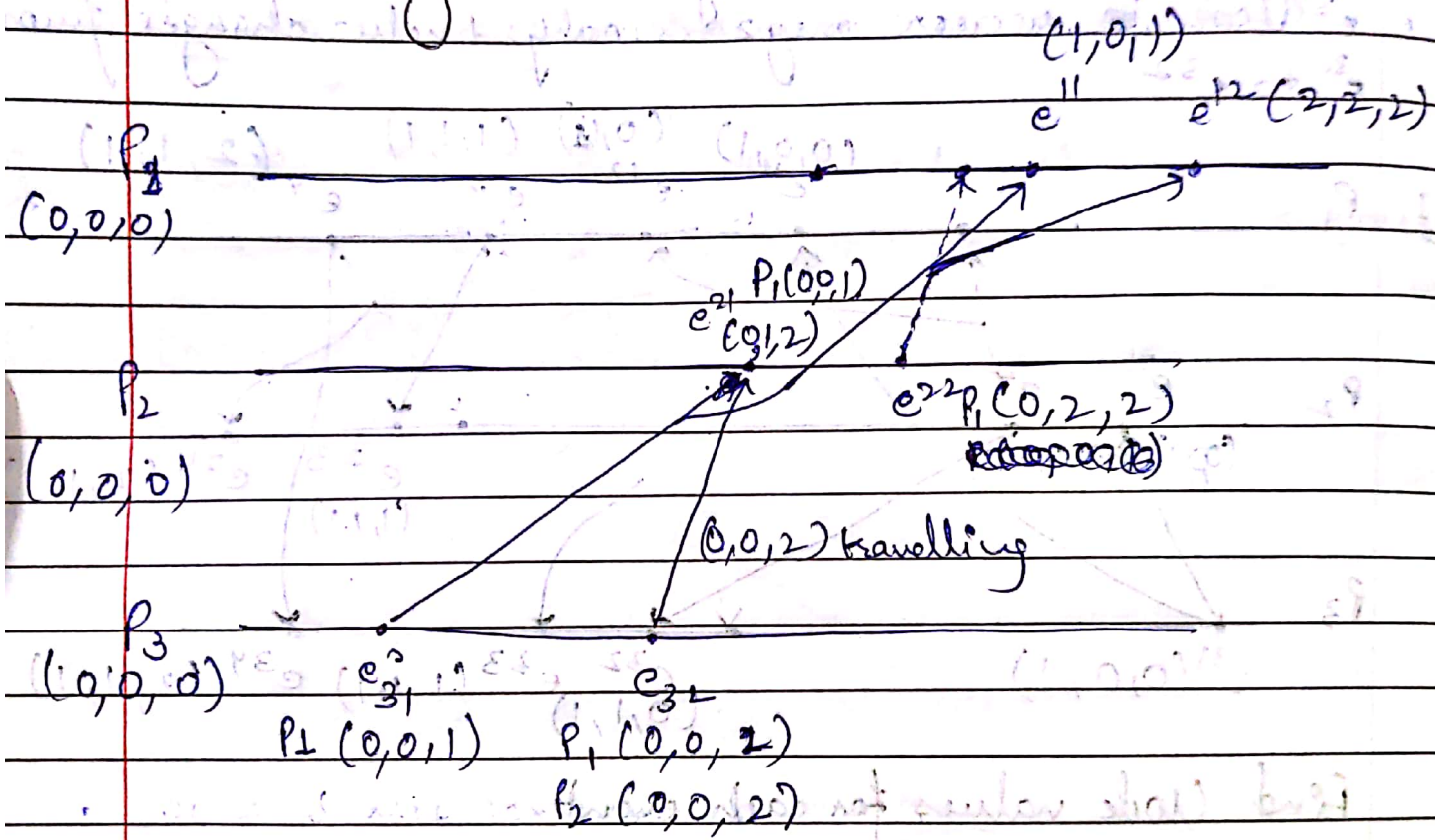
- More the messages, smaller size for the message, limited state information.

- In SES, No need for broadcasting. Few messages, moderate to large size. lots of state information.

- In matrix algo, unicast, multicast and broadcasting. It has few msgs, lots of state info and Laige size msgs.

★ Do algorithm from Book. (Imp.)

⇒ SES Algorithm



- In this clock is incremented upon recieving.
- The event holds info of all the events and info travells.

- from $e_{21} \longrightarrow e_{32}$ info of $P_1(0,0,1)$ is travelled. Similarly when $e_{32} \longrightarrow e_{21}$ then clock is updated acc to msg from e_{32} (0,0,2) and clock incremented on receiving so (0,1,0) so adding (0,1,2) and $P_1(0,0,1)$ is travelled with info to e_{21}

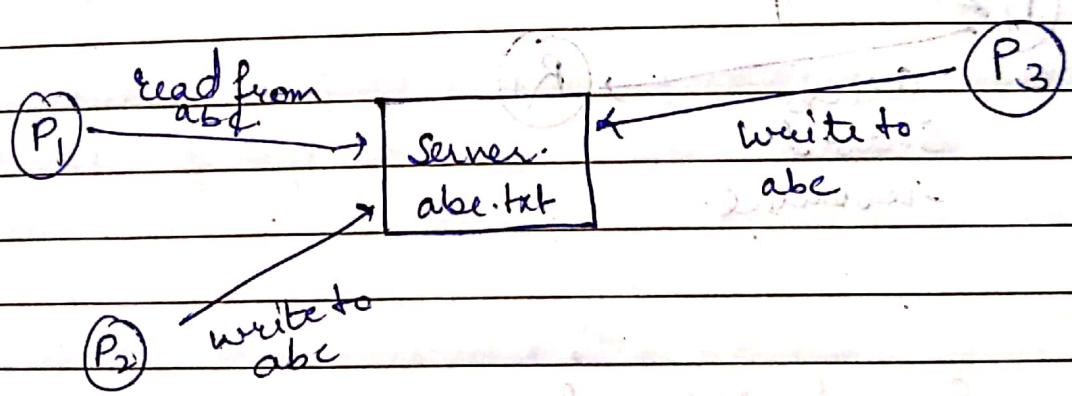
⇒ Mutual Exclusion

- 1 resource can be used by only one process and at that time cannot be used by any other process.

→ Need for mutual exclusion.

- Distributed processes need to coordinate to access shared resources

eg Writing a file in Distribution system



- In Uniprocessor systems, mutual exclusion to a shared memory resource is provided through shared variables or

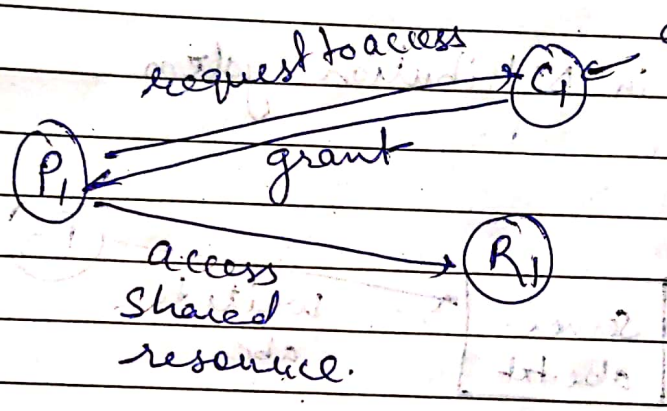
→ Types of Mutual Exclusion

- Classified into two category

1.) Permission based approach

- A process which wants to access a shared resource, requests the permission from one or more coordinator

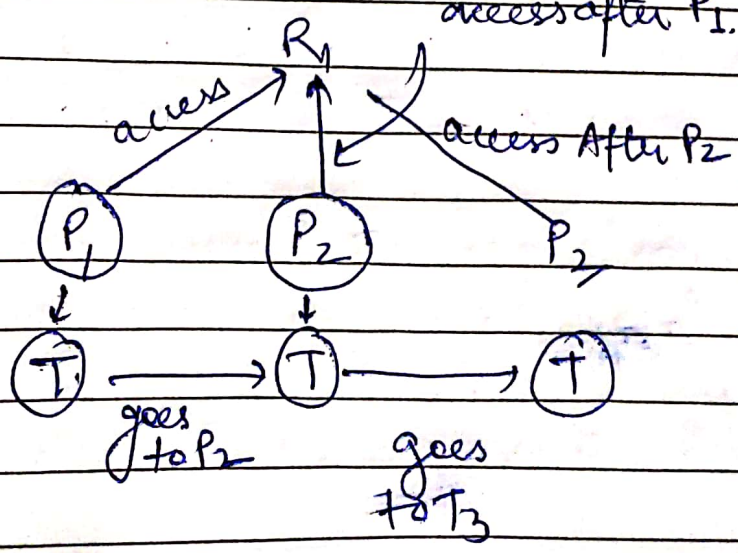
eg



2.) Token Based Approach

- Each shared resource has token.
- Token is circulated among all the processes
- A process can access the resource if it has the token

eg



→ Permission based Approach

- It has 2 types of algorithm used.

(i) A Centralized Algorithm

- One process is selected as coordinator (C) for a shared resource
- Coordinator maintains a Queue of access requests as multiple processes may request at same time. A stack is not used as it is LIFO type.
- If only 1 request for a Resource then direct access granted.
- Whenever a process wants to access the resource, it sends a request msg to the coordinator to access the request
- When request recieved :-
  - (i) If no other process is currently the resource it grants the access permission to process by sending 'grant' msg
  - (ii) If another process is accessing the request resource, the coordinator queues the request, and does not reply to the request.
- The process release the exclusive access after accessing the resource.