Distributed System.

* A DS is a Collection of independent Computer that appears to the users of the system as a single Computer

DS has Two Aspects → The 1st one deal with h/w

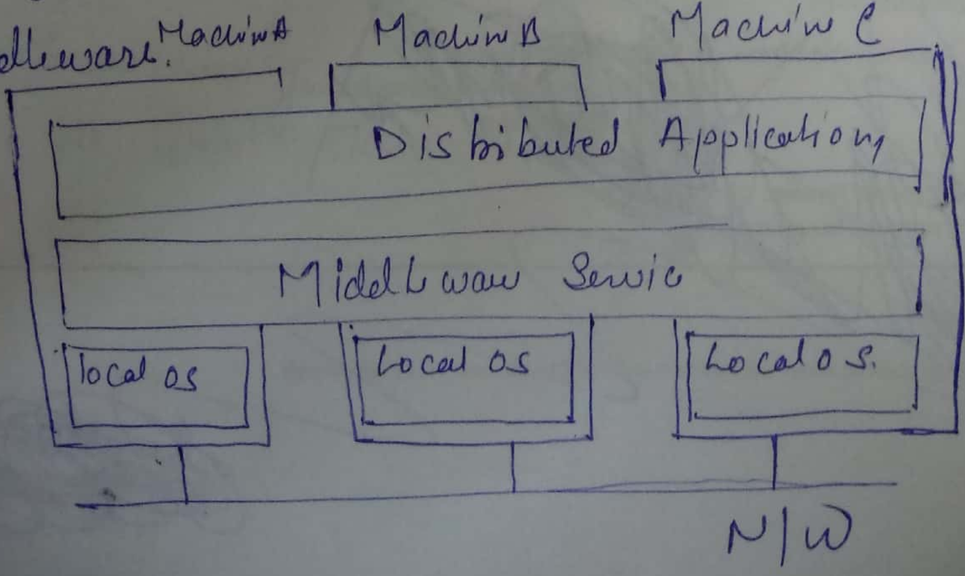
⇒ the machines are autonomous

* The 2nd one deal with S/w : the users think they are dealing with a single System.
  [Both are Essential]

* Users are applications can interact with a DS in a consistent and uniform way.

* It is easy to expand & Scale.

* To support heterogeneous computers & n/w while offering system view, DS are often organized by means of a Layer of S/w that is logically placed b/w a higher level layer consisting of user & Application. and a layer underneath consisting of OS. DS is sometimes called middleware.



Examples of DS

- * The Internet
- * An intranet
- * Mobile Computing.

The Sharing of resources is a main motivation for constructing Distributed Systems.

Resources may be managed by servers and accessed by clients or they mab be encapsulated as object and accessed by other client objects.

Challenges:

- * Heterogeneity of its components
- * Openness. (Which allow components to be added or replaced.
- * Scalability: the ability to work well when the no of Users increases.
- * Failure Handling
- * Concurrency of Components
- * Transparency,
- * No Global Clock

Resource Sharing and Web Challenges.

1) Heterogenity: The internet enables users to access services and run applications over a heterogeneous collection of Computer & n/ws
It applies to all of the following:
1) N/W
2) Computer H/W
3) OS
4) Programing language
5) Implementations by different Developer

* To solving the problems of heterogeneity, middle ware provide a uniform computational model for use by the programmers of servers and distributed application. Ex: Remote object invocation, CORBA.) (Common Object Request Broker) provide remotes Remote object invocation. which allow an object in a program running on 1 computer to invoke a method of an object in a program running on another computer

Openness:

1) Openness of a Computer system is the characteristic that determines whether the system can be extended and reimplemented in various ways.

) The openness of D S is determined by the degree to which new resource sharing services can be added and be made available for use by a variety of client program.

) Openness can not be acheived unless the specification and documentation of the key s/w interface of the components of a system are made available to s/w developer. The key interfaces r published

) Systems that are designed to support resource sharing in this way are termed open D.S. to emphasiz the fact that they are extensible. They may be extended at the h/w level by the addition of computers to the n/w. and at the s/w level. by the introduction of new services and the re implementation of old ones.

RFC- Request for Comment

Security:

1) Many of info. resources that are made available and maintained in DS have a high intrinsic value to their users. Their security is Comportance

2) Security for information resources has 3 Components

•) Confidentiality

•) Integrity

•) Availability

3) In DS Client send request to access data Managed by server which involves sending info. in message over a n/w

* Following two security challenges have not yet been fully met_

*Denial of service Attack

* Security of Mobile Mode

Scalability: DS operate effectively and efficiently at many different scales, ranging from small intranet to Internet

A System is scalable if it remain effective when there is a significant increase in the [illegible] no of resources and the no of user

* The design of scalable DS presents the following challenges.

* Controlling the cost of physical resources

* Controlling the performance loss

Failure Handling:

Computer System sometimes fail. when faults occur in h/w or s/w. programs may produce incorrect resoult or they may stop before thay have compited the intended Computation

Handling of failures is difficult

The techniques for dealing with failures are

1) Detecting failures: Some failures Can be detected. for Example Checksum Can be used to detect Corrupted data in a msg or file.

The Challenge is to manage in the presence of failure that Can not be detected ~~by~~ but may be suspected

2) Masking failure: Some failure that have been detected Can be hidden or made less severe.

Two examp's of hiding failures:

1) Messages Can be retransmitted when they fail to arrive

2) file data Can be written to a pair of disk So that if one is corrupted the ~~othey~~ other may still be correct

Hiding failure are not guaranteed to work in the worst cases

Tolerating failures:

Recovery from failure: Recovery involves the design of s/w so that the state of permanent data can be covered or rolled back after a server has crashed

Concurrency: The presence of Multiple users in a DS is a source of Concurrent request to its resources. Each resource must be designed to be Safe in a Concurrent environment

Transparency: The aim is to make certain aspects of Distribution invisible to the application programmer so that they need only be Concerned with the design of their particular application

for Ex: They need not be concerned with its location or the details of how its operations are accessed by other components.

Architectural Model!

- Architectural Model is an abstract view of a D.S.
- Models are constructed to simplify reasoning about the System
- The Goal of A.M is to ensure that the structure will meet present & future demand on it
- Major Concern are to make the system reliable, manageable, adaptable and Cost effective
- A Model of DS is expressed in term of

  Components

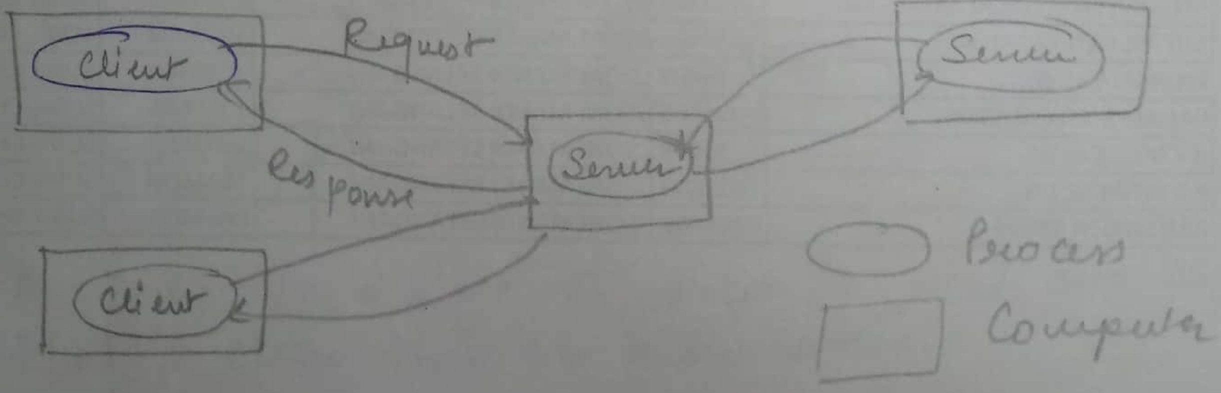
  Placement of Components

  Interaction among Components
- Component of a DS is a process. (running program)

System Architecture Models

1) Client - Server
2) Peer to Peer
3) Variations

Client - Server Architecture



Scanned by CamScanner

•) Client Send Request to server

•) Server Send Responses to Client

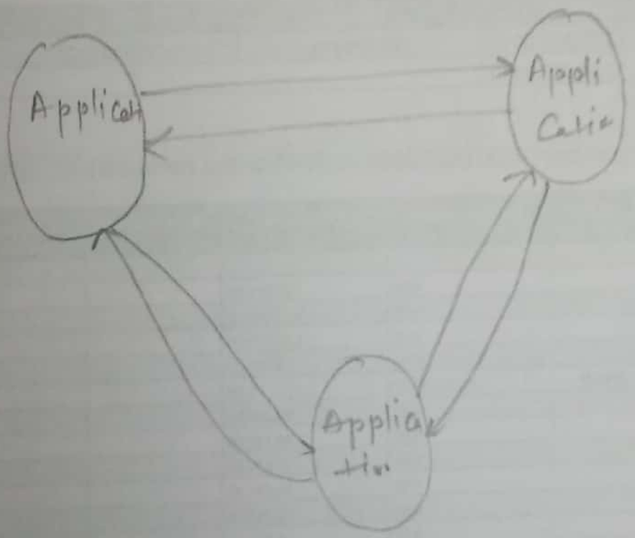
Server may be client of other servers.

→ A Web Server is often a client of a file Server

→ An Internet service is a client of a DNS server

- a server that translates DNS Name to IP address

Peer to Peer Architecture



•) All processes play similar role. i.e. they interact as peer

• No Central Component: Potentially better Scalability and resiliency to failure

•) Use the power of Modern desktop to implement a large Scale D.S

Ex. Skype. Bittorrent. Napster

Architectural Variations:

⇒ Services Provided by Multiple Server

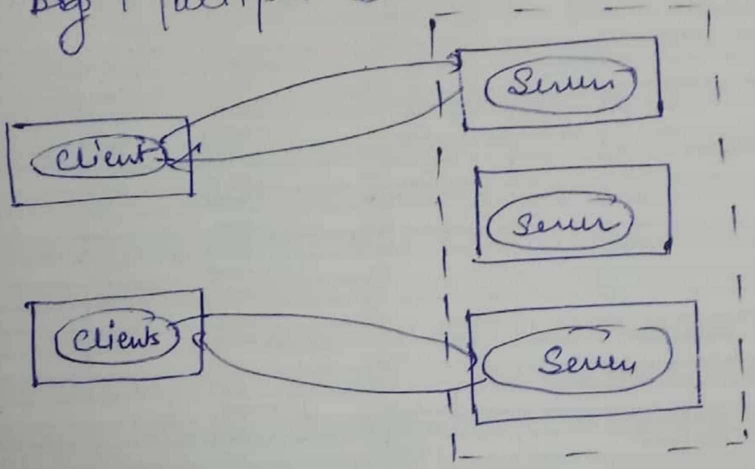
⇒ Proxy Servers and Caches.

⇒ Mobile Agent

⇒ N/W Computer

→ Mobile devices

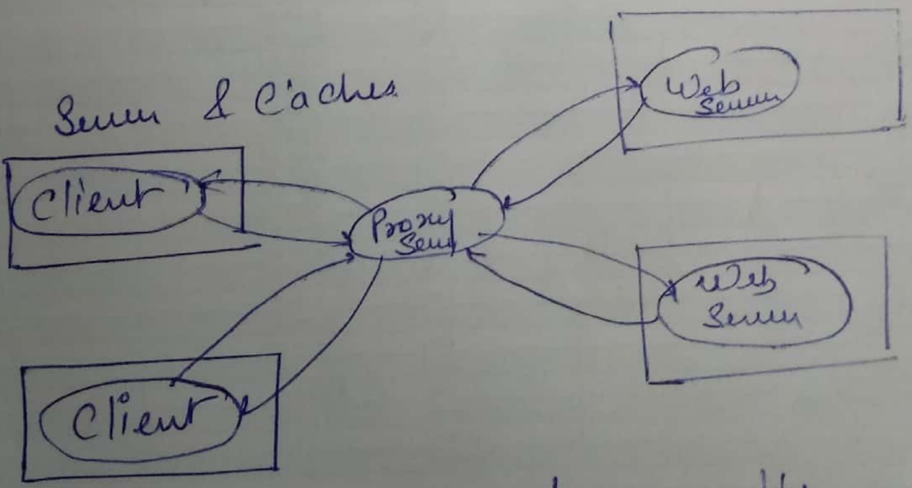
1) Services By Multiple Servers.



1) Multiple Servers provides services to Client

2) Server May Partition the services object or replicate them

2) Pory Server & Caches



1) A Cache is a store of recently used data objects that is closer than the main Store

A newly accessed object is added to the cache.
when that object is accessed again it is fetched from cache, if there is an up to date copy in the cache.

) Proxy servers intercept communication with the real server to provide faster service. better security.

Mobile Agent

•) A running program (both code and data) that travels from 1 computer to another
Ex. A worm
used to attach computer system

Mobile Device
→ Cellular Phone
⇒ Laptops
→ Mobile Sensors

Fundamental Model:

i) Understanding the characteristics that impact distributed System performance and operation

Purpose of Model: To make generalization concerning what is possible or impossible

Interaction Model: Distributed Systems are comprised of entities that Communicate and Coordinate by passing messages. The following characteristics of Communication channels impact the performance of the System.

- Latency - the time b/w the sending of a message at the Source and the receipt of the message at the destination.
- Bandwidth - the total amount of information that can be transmitted over a given time period.
- Jitter - the variation in the time taken to deliver a series of message.

⇒ Coordination of actions of entities in a DS. is impacted by the fact that each entity will have a different clock drift rate

⇒ Synchronous D.S. that relay on certain action happening at the same time can only be built if you can guarantee bounds on System resources. and clock drift Rate

Failure Model

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

- Failstop: A process halts and remains halted. Other processes may not be able to detect this state

- Crash: A process halts and remains halted. Other Processes ~~can~~ may not be able to detect this state.

- Omission: A message inserted in an outgoing message buffer never arrives at the other ends incoming message buffer.

- Send-Omission: A process completes a send but the message is not put in its outgoing message buffer

- Receive-Omission: A message is put in a process's incoming message buffer but that process does not receive it

⇒ **Timing Failure**: Clock drift exeeds allowable bounds.

⇒ **Byzantine Failures**

- Bug
- Message corruption
- Hardest to deal with

Security Model: There are several potential threat a system designer need be aware of.

* Threat of Processes: An attacker may send a request or response using a false identity (spoofing)

* **Threat to communication channel:** An attacker may eavesdrop (listen to message) or inject new message into a communication channel. An attacker can also save messages and reply them later

* Denial of Service: An attacker may overload a server by making excessive requests.

⇒ Cryptography and authentication are often used to provide ~~security~~ security.

* Communication entities can use a shared secret to ensure that they are communicate with one another and to ~~end~~ encrypt their message. So that they can not be read by attackers.

Limitation of Distributed System

1) A DS is a set of Computers that Communicate over a network, and do not share Common memory and Common clock.

2) Absence of a Common clock (Global)
   i) No concept of Global time
   ii) Its difficult to reason about the temporal ordering of events.
      1) Cooperation b/w processes (eg. Producer/Consumer, client/Server)
      2) Arrival of request to the OS (eg for resources)
      3) Collecting up-to-date global state.

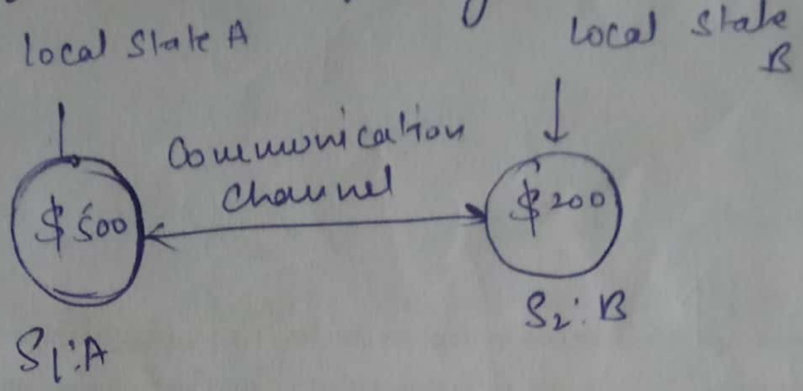
3) It is difficult to design and debug algorithm in a D.S.
   •) Mutual Exclusion
   •) Synchronization
   •) Deadlock

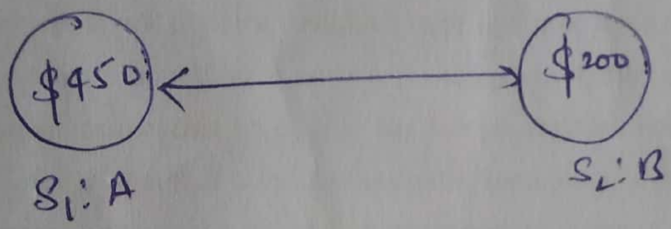
Absence of Shared Memory

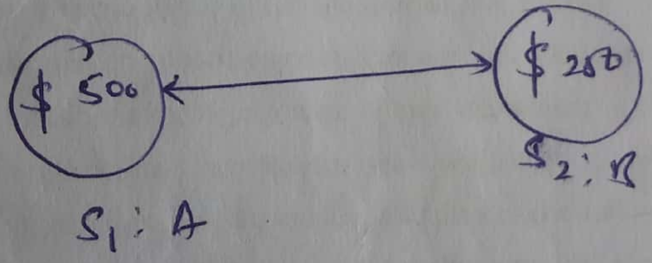
1) No up to date state of the entire system to any individual process as there's no shared Memory.

2) Coherent View: All observations of different processes (Computers) are made at the same physical time.
   •) We can obtain a Coherent but partial view of the system or incoherent view of the system.
   •) Complete view (Global State) — local view (local States) + Message in transit.

Absenca of Shared Memory







Logical Clock: Lamport invented a simple mechanism by which the happened before ordering can be captured numerically called a logical clock

*) A lamport logical clock is a monotonically increasing s/w counter.

Drawback

1) clocks are not strongly consistent: clock lose track of the timestamp of the event on which they are dependent on.

This is because we are using a single integer to store the local and logical time

2) if a→b then C(a) < C(b). the reverse is not necessarily true if the events has occured in different processes.

That is. if a & b are event in different processes and C(a) < C(b) then a→b is not necessarily true.

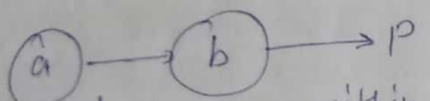
events a & b may be causally related or may not be

Logical clock:

- To order event across process, trying to sync clocks in our approach
- Another Approach = Assign timestamps to events
- As long as these time stamps obey causality, this would work

Happen Before Relationship

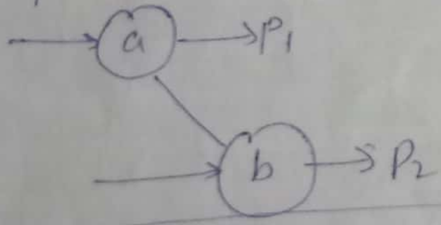
(1)



Ts(a) < Ts(b)

if a and b are within the same process. and A occurred before B.

(2)



Ts(a) < Ts(b)

if a is the event of sending a msg M in our Process & b is the event of receiving M by another process.

Properties Derived from (HBR)

Transitive Relation

if
- $Ts(a) < Ts(b)$
- $Ts(b) < Ts(c)$

Then

$Ts(a) < Ts(c)$

Causally Ordered Relation

$a \rightarrow b$

changes of a, affect b.

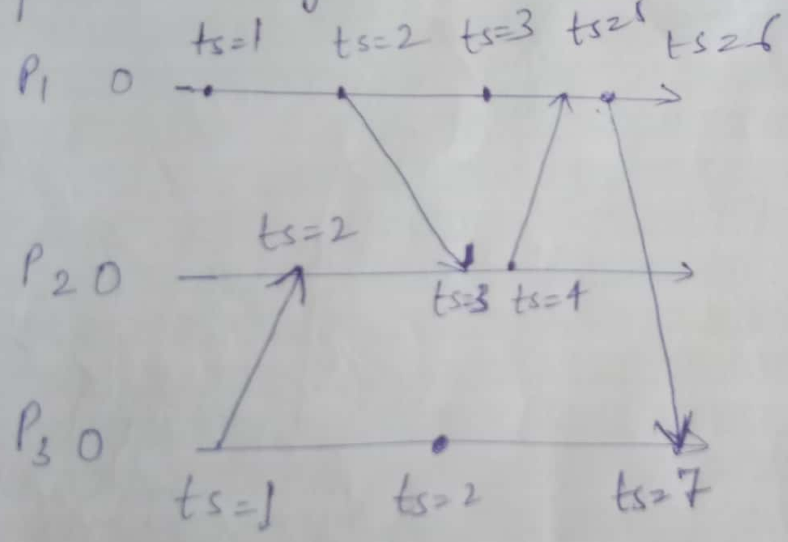
Concurrent Event

A → B ×

B → A ×

A || B

Lamport's Clogical Clock



ts Receive = Max(Own ts, Sende Ts +1

Event occur ts = ts+1

Max (0,1) +1
1+1 =2

Max (2, 2)+1
2 +1 = 3

ts(3, 4) +1
4+1=5

ts(2,6)+1
6+1=7

Algo:

1) A process increment its ~~counter~~ Counter before each event in that process.

2) When a process sends a message it includes its Counter value with the msg

3) On receiving a msg. the counter of recipient is updated, if necessary. to the greater of its Current Counter & the timestamp. in the received msg. The counter is then incremented by 1 before the msg is considered received

algo for sending is

```
        time = time+1
  time stamp = time
  Send (Msg, time_stamp);
```

Algo for receiving a msg

```
  (msg, timestamp) = receive();
  time = max (time_stamp, time) +1;
```

---

* A logical Clock is a mechanism for Capturing Chronological and Causal relationships in a D.S.

* D.S. may have no physically synchronous global clock so a logical clock allow Global ordering on events from different processes in such system.

Vector logical Clocks:

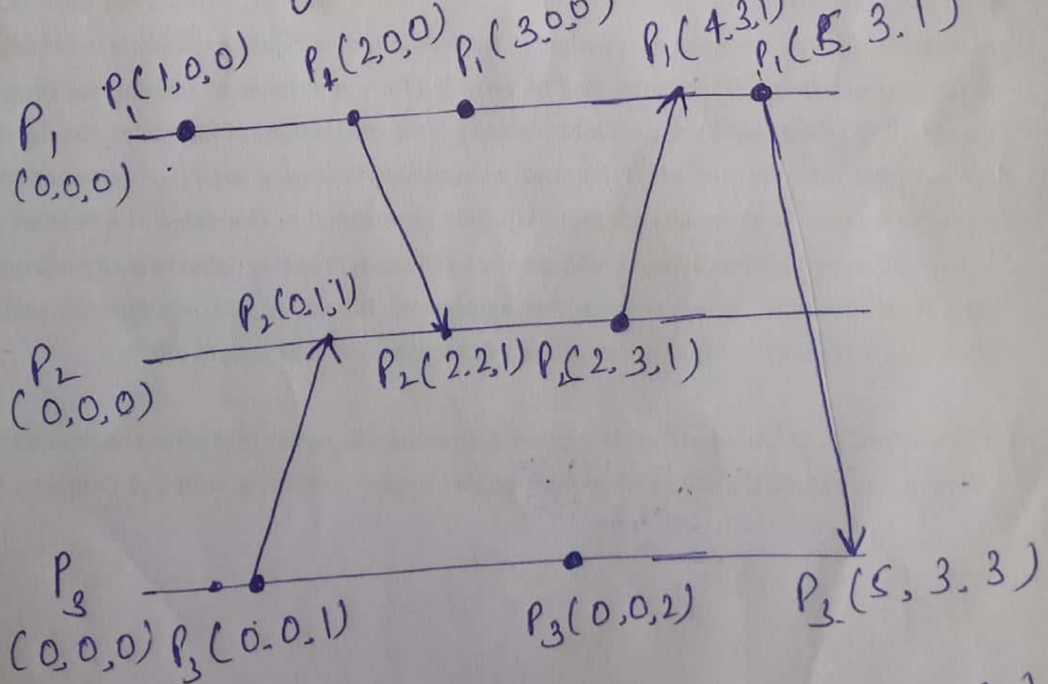
1) Vector clocks overcomes the shortcoming of Lamports clock - The fact that $L(e) < L(e')$" we can not conclude the $e \rightarrow e'$

2) A vector clock for a system of N processes is an array of N integers.

3) Each process keeps its own vector clock V_i, which it uses to timestamps ~~on the message~~ local event.

4) Like Lamport timestamps, processes piggyback vector timestamps on the message they send to ~~one~~ one another

5) A vector clock is an algorithm for generating a partial ordering of events in a distributed system and detecting causality violations.



$P_2(0,1,0)$
$P_3(0,0,1)$
$Max(P_2, P_3)$
$P_2(0,1,1)$

$P_2(0,2,1)$
$P_1(2,0,0)$
$Max(P_2\ P_1)$
$P_2(2,2,1)$

$P_1(4,0,0)$
$P_2(2,3,1) \Rightarrow P_1(4,3,1)$

$P_3(0,0,3)$
$P_1(5,3,1)$
$P_3(5,3,3)$
=

Algorithm :

Step1 = Vector initialize to 0 for each process.

$V_i[j] = 0$ for i. j = 1.2.---M.

Step 2 = Increment Vector before time stemp event

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

Step 3 = Message is sent from P_i with V_i attached to it

Step 4 :- When P_j receive message

$V_j[i] = \max(V_i[i], V_j[i])$ for $i = 1 --- M$.

Vector Timestamps have the disadvantage, compared with Lamport timestamps, of taking up an amount of storage and message payload that is proportional to M, the no of processes.

Concept of Message Passing System

Causal Ordering of Message

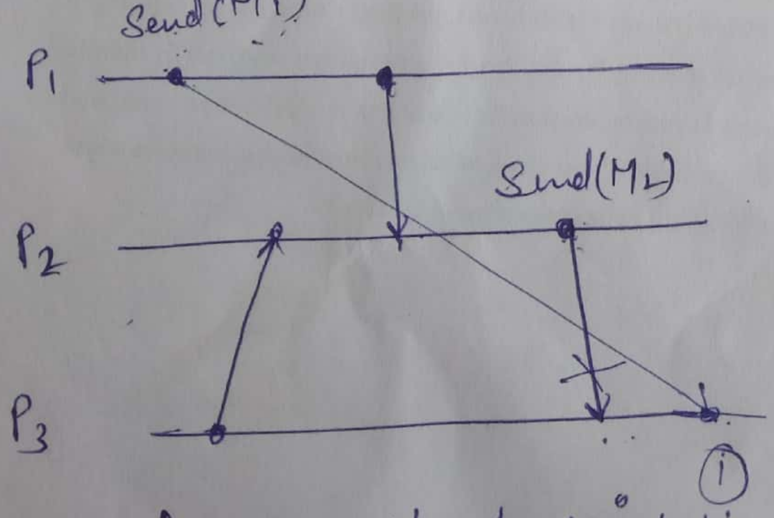
The purpose of Causal ordering of message is to insure," that the same Causal relationship for the "message send event correspond with message receive event. i.e.

All the messages are processed in order that they were created.

In other words : if Send$(M_1) \rightarrow$ Send (M_2) (when Send (M)) is event sending message M). then every recipient of both messages M_1 & M_2 must receive M_1 before M_2.

~~Violation of~~

There are Two protocols that make use of Vector clocks for the Causal ordering of messages in D.S.



An example of violation of Causal ordering of message

BSS Algorithm: Birman - Schiper - Stephenson Protocol

(1) Broadcast based: A message sent is received by all other processes.

2) Deliver a message to a process only if the message preceding it immediately has been delivered to the process.

⇒ otherwise. buffer the message

3) Accomplished by using a vector accompanying the message

Algorithm:

1) Process Pi increments the vector time VTpi[i], time stamps, and broadcasts the message m. VTPi[i]-1 denotes the no of messages preceding m.

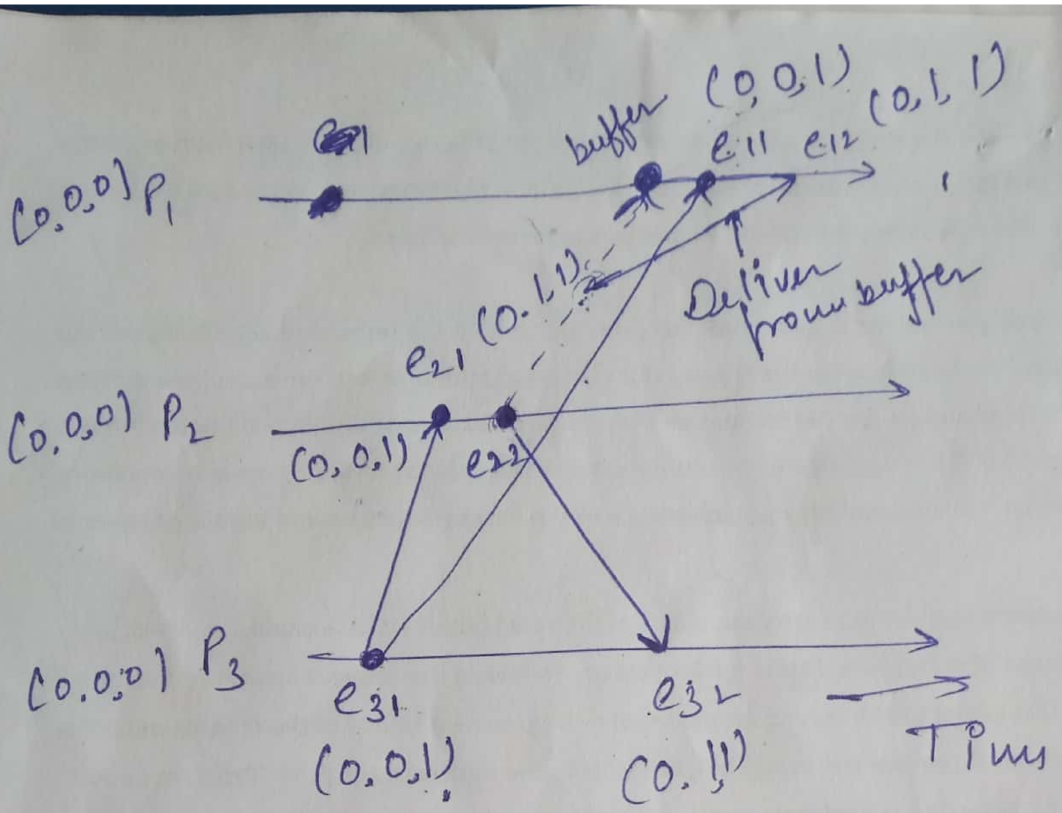
2) Pj ≠ Pi receives m, m is delivered when:

a) VT Pj[i] = VTm[i]-1

b) VT Pj[k] >= VTm[k] for all k in {1,2,--n} - [i], n is the total no of processes.
Delayed message are queued in a sorted manner.

c) Concurrent message are ordered by time of receipt.

3) When m is delivered at Pj, VT pj Updated according Rule 2 of vector clocks.



There is no clock increment upon receiving.

Send (M_1) → Send (M_2) → Receive (M_2) → Send (M_3)

Schiper-Eggli-Sandoz Protocol

- * No need for broad cast Message
- * Each process maintain a vector V_P of size N-1 (N the no of processes in the system)
- * V_P is a vector of tuple (P', t):

  P' = Destination Process ID

  t = Vector timestamp.

- * Tm: logical time of sending message m
- * Tpi: Present logical time at Pi
- * Initially, V_P is empty.

Algorithm

1) Sending a Message

⇒ Send Message M, timestamp tm, along with V_P_1 to P_2

⇒ Insert (P_2, tm) into V_P_1.. overwrite the previous value of (P_2, t) if any.

⇒ (P_2, tm) is not sent. Any further message ~~carrgins~~ carrying (P_2, tm) in V_P_1 can not be delivered to P_2

until $tm < t\ P_2$

Delivering a Message:

1) if V-M (in the message) does not contain any pair (P_2, t) it can be delivered.

2) /* (P_2, t) exist */ if ~~t~~ $t \nless tP_2$ buffer the message. (Don't deliver)

3) else $(t < tp_2)$ deliver it

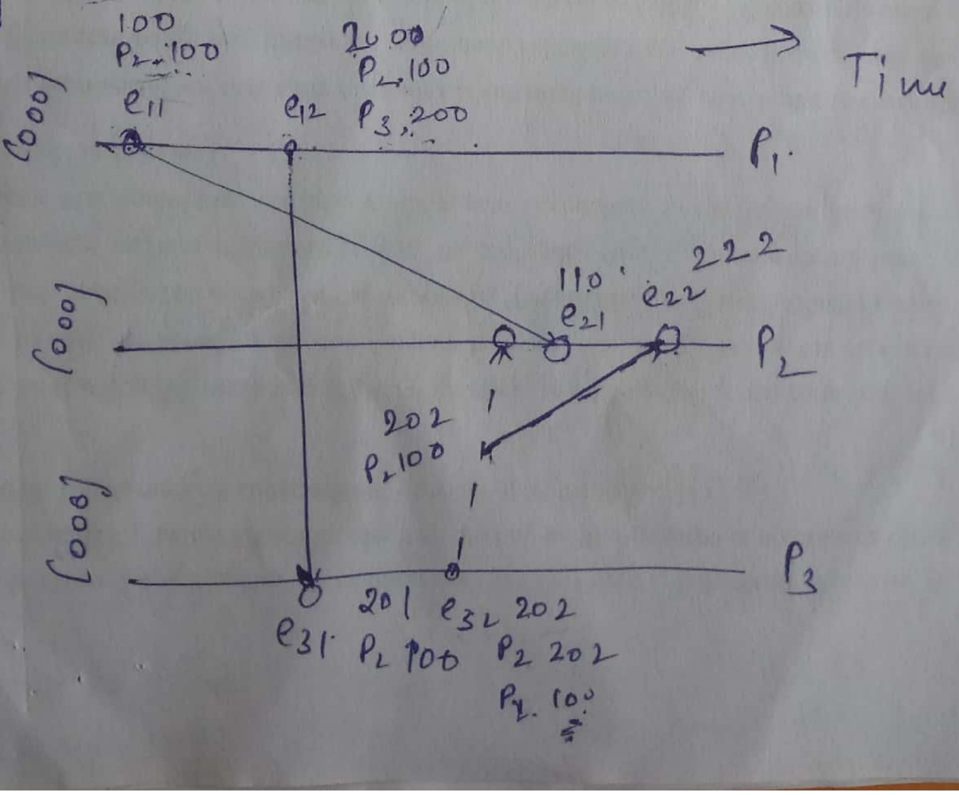
What does the condition $t \geq TP_2$ imply?

t is message vector time stamp.

$t > TP_2 \Rightarrow$ for all j's $t[j] > TP_2[j]$

This implies some events occurred without P_2's knowledge in other processes. So P_2 decides to buffer the message

$\Rightarrow$ when $t < TP_2$ message is delivered & TP_2 is updated with help of V_P_2



Global State of Distributed System.

⇒ Global state of DS is a collection of the local states of all the processes and the states of all the communication channels.

⇒ The state of Process P_i is a collection of all events that happend at the process.

- Internal message send and message receive

⇒ The state of channel is a collection of ~~the local~~ all the messages that have been sent and not yet received

Consistent Global State:

⇒ A message ~~m~~ sent by a process P_i to a process P_j and recorded in the local state of P_i should be either recorded in the local state of receiver P_j ~~and~~ or the ~~state~~ channel C_{ij}

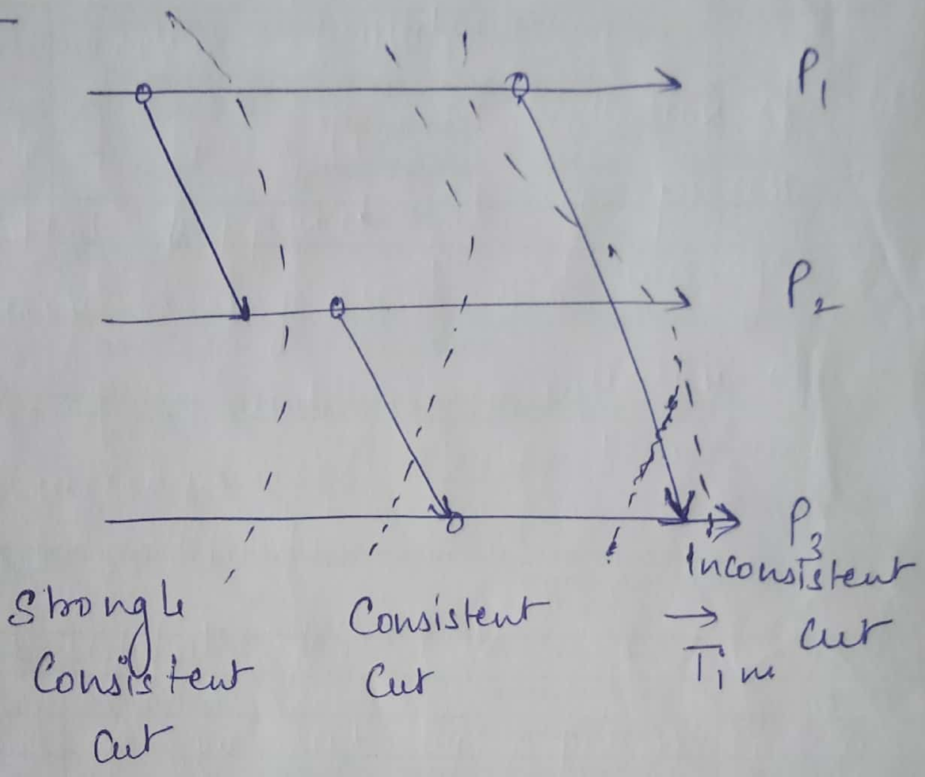
→ A message m sent by ~~the~~ a process P_i after recording its local state should not be recorded in the local state of receiver P_j and also not in the state of channel C_{ij}.

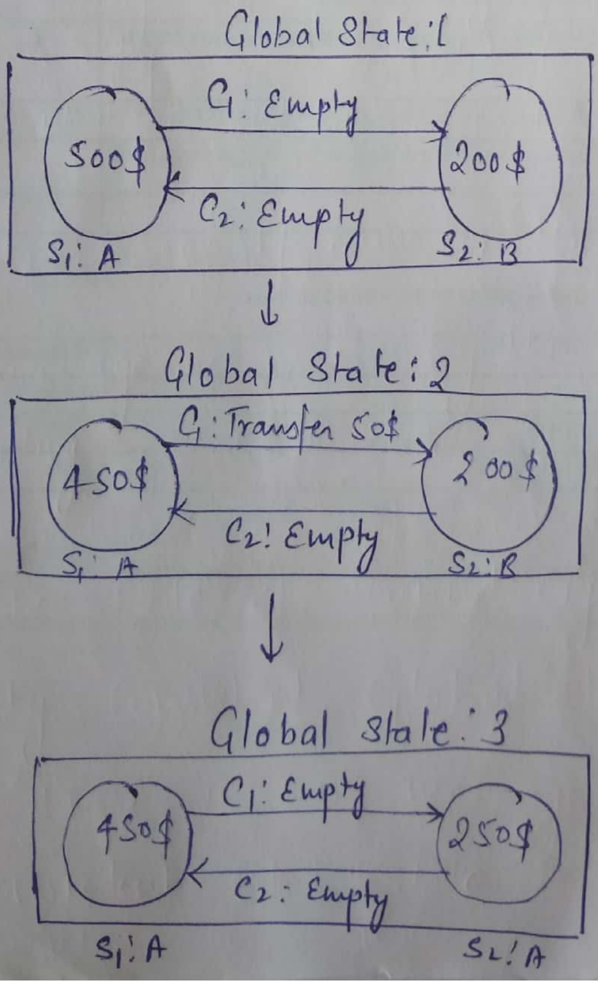
Strongly Consistent Global State: A consistent Global State in which there is no message in transit.

An Inconsistent Global State is the one that records the receiving of a message ~~and the sending of messa~~

in the receiving process. or as a transit message in the channel. but the sending of message is not recorded

Example





Chandy Lamport's Algorithm

1) A distributed Algorithm to collect Global State

2) Assumption: All communication channel are FIFO (no overtaking of messages)

3) An initiator processor records its local state and send a marker to all its communication channels (before sending any message on the channels)

4) When a process receives the marker
→ if it has already recorded its local state
→ Records the messages received after recording the local state (but before the marker was received) as the status of the channel.

Else:
→ Records the local state, record the state of the channel as empty and send out the marker on all of its communication channels

5) All processes report their local state and the status of their associated incoming channels to the ~~interi~~ initiator process

Termination Detection

⇒ A Distributed System generally consist of a set of cooperating processes which communicate with each other by exchanging message. In this case, it is important to know when the computation has terminated.

⇒ A process may either be in an active state or idle state.

⇒ Only active processes can send messages.

⇒ An active process may become idle at any time.

⇒ An idle process can become active on receiving a computation message

⇒ Computation messages are those that are related to the underlying computation being performed by the cooperating process

⇒ A computation is said to have terminated if and only if all the processes are idle and there are no message in transit.

⇒ The message sent by the termination detection algo. are referred to as control message

Notations

- $B(DW)$ = Computation message sent as a part of the computation and DW is the weight assigned to it
- $C(DW)$ = Control message sent from the processes to the controlling agent and DW is the weight assigned to it.

Huang's Termination Detection Algorithm

Rule 1: The controlling agent or an active process having weight w may send a computation message to a process P by doing:

Derive W_1 and w_2 such that

$w_1 + w_2 = w$, $w_1 > 0$, $w_2 > 0$

$w = w_1$

Send $B(w_2)$ to P

Rule 2: On receipt of $B(DW)$, a process P having weight w does.

$w = w + DW$

if P is idle, P become active

Rule 3: An active process having weight w may become idle at any time by doing

Send $C(w)$ to the controlling agent

$w = 0$

(The process becomes idle)

Rule 4. On receiving $C(\Delta W)$, the controlling agent having weight w takes the following actions:

$W = W + \Delta W$

if $w = 1$ conclude that the computation has terminated