Distributed System :-

A system in which h/w or s/w components located at network computers communicate & coordinate their actions only by message passing.

A distributed system is collection of independent computers that appear to the users of the system as a single computer linked by a network with s/w design to produce an integrated computing facility.

Reasons :- (for studying DS)

① Functional distribution
- client/server
- Host/terminal
- data gathering/processing
- Sharing of resources

② Application requirement
③ Load distribution/balancing
④ Replication of processing power
⑤ Physical separation
⑥ Economy

Challenges or Problems connected with Distributed Systems :-

① Heterogeneity → (difference in OS, pgming language, h/w)
- Middleware (e.g. CORBA, JAVA RMI) → It hides all the internal details & acts as a s/w layer.
- Mobile Code (e.g. Java Applets)

② Openness (able to be extended & reimplemented)
③ Security ← Confidentiality (authentication), Integrity (data shouldn't be lost), Availability
④ Scalability
⑤ Failure handling → (detecting, recovering, masking/prevention/avoidance of failure).

Characteristics of Distributed Systems :-

(Parallelism (particular time interval) / (particular time instance))

1. Concurrency
2. Absence of global clock
3. Absence of global state
4. Independent failure of components

Concurrency < send, receive } primitives
(Message passing)

global state → (concept of shared memory (common memory))

Examples of Distributed Systems :- (One can have its own OS)

1. Internet
2. Intranet
3. Distributed Multimedia System
   (e.g. VideoConferencing, video & audio on demand)
4. Mobile computing
5. Telephony System
   e.g. ISDN

challenges

6. Concurrency
7. Transparency

22/2/05 Imp.

System Models :-

- H/W — Architectural models → take care of Physical design of components
- S/W — Fundamental models → properties of the design of components
  - interaction model
  - failure
  - security

Interaction deal with ← Synchronization, Concurrency / how transaction take place, Coordination

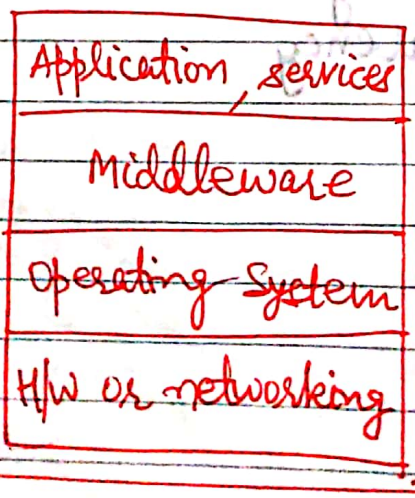
Failure — Failure handling

Security deals with — openness, scalability, Transparency (N/w transparency, local transparency, database transparency)

Architectural model :- we talk in terms of 3 things

1. Software layers
2. System Architecture
3. Variations

Software layers :-



Software layers

Examples of middleware are CORBA, Java RMI, Microsoft DCOM (Distributed Component Object Model),-

middleware is a s/w which is intelligent enough to take care of operating system (requests of) & h/w.

(CORBA – Common Object Request Broker Architecture
Java RMI (Remote Method Invocation).
middleware

Middleware :- It is layer of s/w whose purpose is to hide heterogeneity & to provide convenient programming model for applications. It provide communication with the help of remote method invocation, notification of events, replication (shadowing) of shared data.

It also provides services for use by application programs which include naming (DNS domain Name System), security, transactions & persistent storage.

Application → Internet, intranet

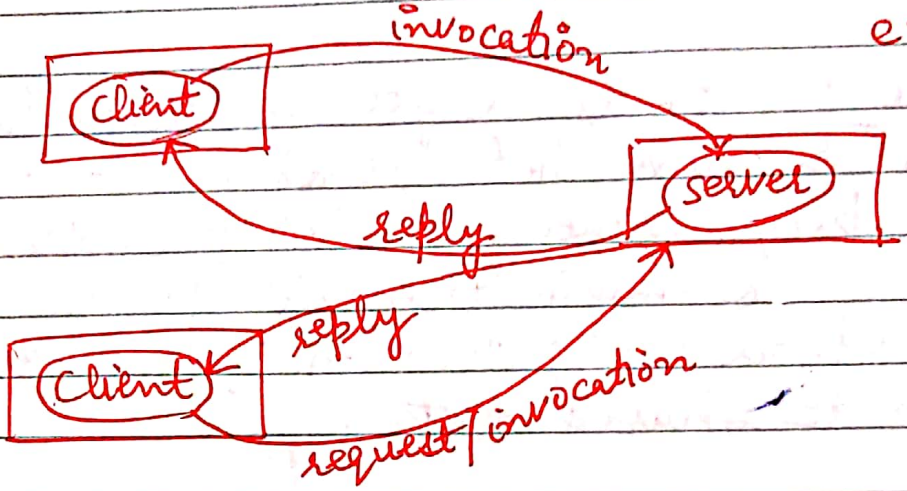
ODP (Open Distributed Processing).

23/2/05

System Architecture

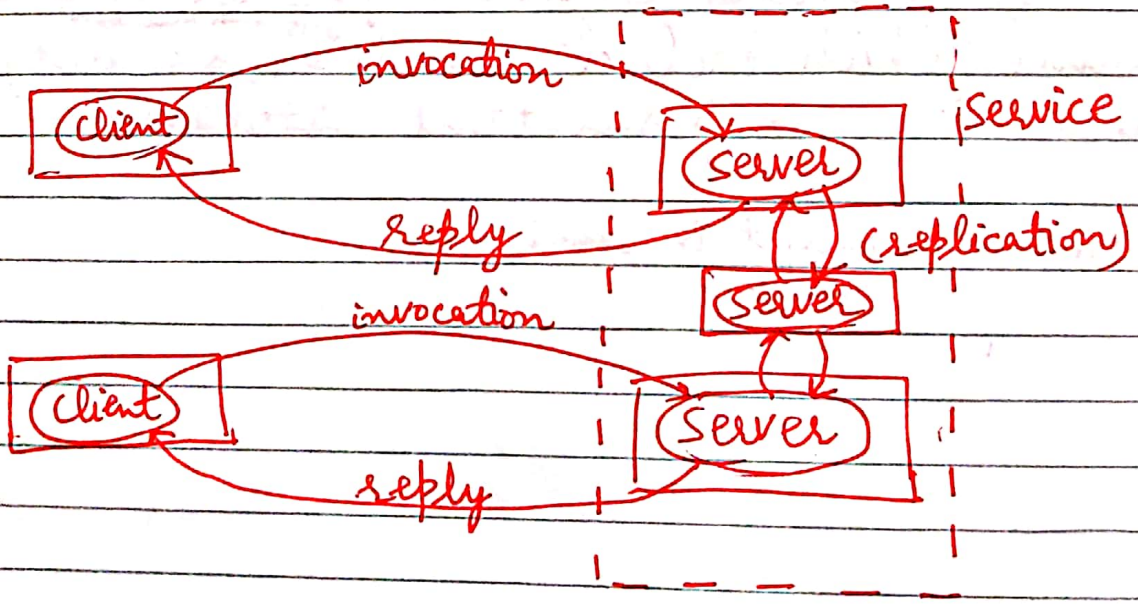
1) → Client / server model
2) → Model with multiple servers
3) → Proxy servers and caches
4) → Peer processes

1)

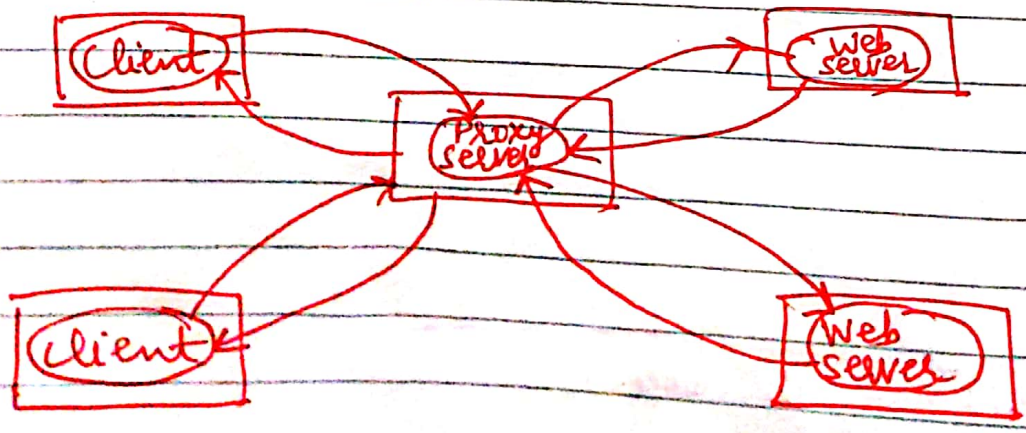


e.g. - web crawlers (used in search engine)

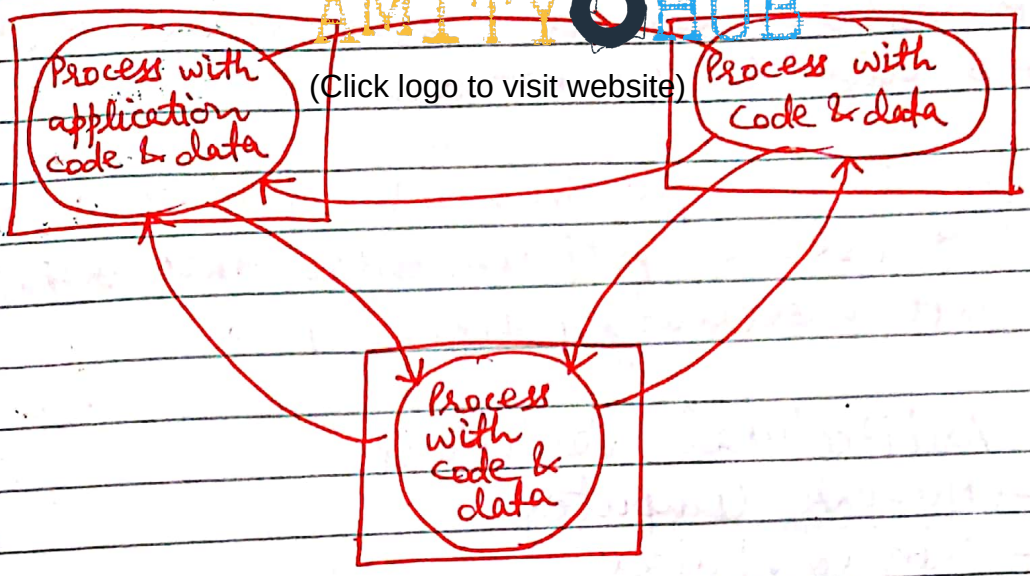
2)



3)



4)



Proxy servers (i) act as a cache memory, (ii) act as a firewall, (iii) used to do traffic control (iv) monitoring

Only 2% of the n/w topology is used as peer processes. Peer processes are used for coordination & synchronization.
Internet is a combination of client/server model with multiple servers & proxy servers and caches.

Variations :- Variation is required b'coz of the following reasons:
(i) The use of mobile code.
(ii) Need for low cost computers with limited h/w resources.
(iii) The requirement to add & remove mobile devices.

Variations include :
→ Mobile code - Java applets
→ Mobile Agents
→ Network computer
→ Thin computers
→ Spontaneous networking

(till Deadlock detection in sessionals)

Adv. of mobile code is better interactivity & follows client/server model.

Mobile agent is a s/w in which machines are most of the time free to perform their other tasks (adv.) and time, cost & security (disadv.)

28/2/05 Architectural Variations
- Network Computer
- Thin computer
- Spontaneous n/w

Thin Computer:
Computer servers and dummy systems.
It is vice-versa of network computer.
E.g. ATM
Disadv. Take time to load OS & application pgms
Adv. distribution of work.

(1) In DS idle processors can be utilized by other OS in the system. Some processes can run on idle processor & time can be consumed. 2) User can't move processes on idle m/c manually.

Network Computer

Spontaneous n/w: Used to carry out -
- WAP (wireless Application Protocol)
- mobile devices

Fundamental Model :-
Make generalization of various topologies making suitable assumptions.
- Interaction (handles mutual Exclusion, deadlock, data & msg comm., synchron, coordination.)
- Failure
- security

We can communicate data & message and b'coz of this we achieve synchronization and coordination in distributed systems.

Interaction Performance depends on
- Communication channel
- Global clock

Communication channel
- bandwidth
- time latency
  - transmission time
  - waiting time
  - time taken by operating system
- jitter

Time taken by OS to take care boot keeping task.

Jitter is a effect which affects the cause of variation in transmission time when a series of messages are sent together. (due to congestion, etc.)

Ques. What is the diff. b/w or Why do we need global clock & global memory? What is the impact of absence of global clock?

To overcome global clock
1. global notion of time
2. local clock synchronized
   - clock drift rate (due to this we can't maintain global notion time)

(A common global clock is communicating with, due to transmission delay time we can't have global clock notion time)

Interaction model Algorithms — Synchronous (time bound) / Asynchronous (no time bound)

Synchronous is the model which has time bound on
① Each process knows lower bounds & upper bounds
② transmitted message takes specific amt of time.
③ local clock with known drift rate is maintained
(deadlock avoidance)

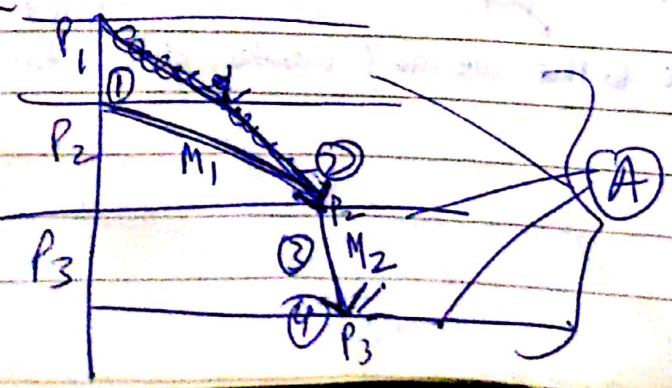
Asynchronous model does not specify time bound
① Execution time/speed of a process (could differ from one to another).
(deadlock detection & recovery)
② There could be any transmission time.
③ local clock with unknown drift rate

Work on the principle of "Event ordering"

Internet is the best example of asynchronous model.

Only highly secured models are synchronous models whereas most of the models are based on asynchronous in DS.

Event ordering is based on concept of logical clock — Timestamping
(vector clock)



Detect Recover Mask Redundancy (backup server (copy of data on diff. servers))

Failure Model :

We take care of causes of failure & how to handle them.

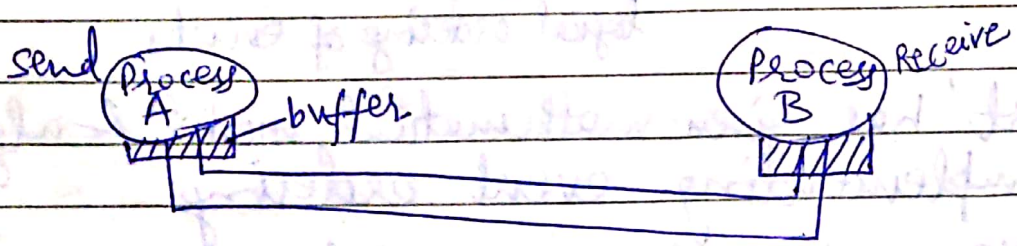
Failures based on severity
- Benign (low severity)
  - Omission (Computer System / Process / Channel)
  - Timing
- Byzantine (high severity)
  - Arbitrary

Omission failures are those failures which do not let process & communication channel to do what they are supposed to do.

Process (computer system) — crash (H/W failure)
crashing of a system is detected on the basis of synchronous whereas it is undetected " " " asynchronous.

Process — crash (fail-stop)
Fail-stop is based on synchronous & it is detectable.
Crash is non-detectable by other systems.

Channel Omission failure
- send omission
- Receive Omission
- channel Omission



buffer is managed & created by OS

"dropping of message"

Byzantine — Arbitrary (Process / Channel)

Arbitrary failures are the one in which there is some omission in the processing steps which were intended to be carried out or some unintended processing steps are executed.

Timming failures when interaction model is synchronous.

Clock (difference in drift rate of local clock)

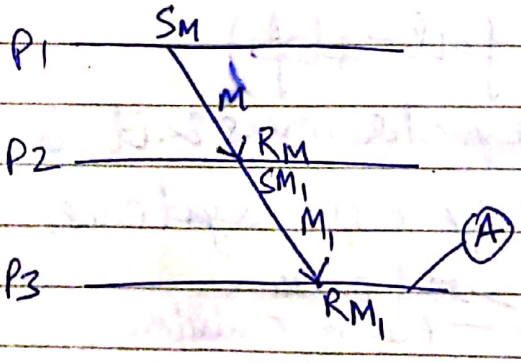
Performance — Process, channel

SECURITY — security of processes, channels, objects against unauthorized access

Imp.

1/3/05

Event Ordering :-



$P_1(S_M)$
$P_2(R_M)$
$P_2(S_{M_1})$
$P_3(R_{M_1})$
A

↓ logical ordering of events

'Lamport' has given mathematical model & algo for implementing event ordering.
He gave Happened Before Relation '→'

1. Existence of a process is an event.
2. Any msg sent by a process is an event.
3. Any msg received by another process is an event.

$P_1(S_M) \rightarrow P_2(R_M)$
$P_2(R_M) \rightarrow P_2(S_{M_1})$
$P_2(S_{M_1}) \rightarrow P_3(R_{M_1})$

} Casual Effect
Casually Related Events
Concurrent Event

The events which have occurred in past are going to effect events in future to occur is called Casual Effect. These events are called casually Related Events.

$P_1(S_M) \rightarrow P_2(S_{M_1})$

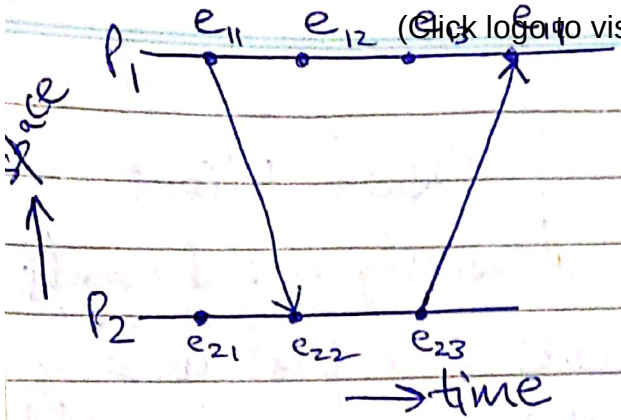
The ~~relation~~ relation ——— is defined as $a \rightarrow b$ if happened before,
1. a and b are events in the ~~same~~ process & a has occurred before b.
2. $a \rightarrow b$ if a is the event of sending a message M in a process & b is the event of receiving of the same message M.

a & b are concurrent events iff

$a \not\rightarrow b$
$b \not\rightarrow a$
} $a \| b$

Two events a & b in a DS should have either of these relationships -
(i) $a \rightarrow b$
(ii) $b \rightarrow a$
(iii) $a \| b$

P_1 & P_2 are no. of processes or systems in the DS



$e_{11} < e_{13}$
$e_{11} \rightarrow e_{13}$
$e_{11} \rightarrow e_{22}$
$e_{11} \rightarrow e_{23}$
$e_{22} \rightarrow e_{14}$

Space-time diagram

Lamport has also given the concept of Timestamping ie. Lamport logical clock C_i for each process P_i

For each process there is a logical clock C_i.

It runs on the following basis: If there are 2 events a & b in the system then $C(a) < C(b)$ when the following 2 conditions are met.

Condition 1 if a has occured before b & then $C_i(a) < C_i(b)$ a & b are events of process P_i.

Condition 2 If a is the sending event of process P_i & b is the receiving event of process P_j then

$$C_i(a) < C_j(b)$$

P_i P_j

Algorithm

1. Clock C_i is incremented between two events in process P_i.

i.e. $\boxed{C_i = C_i + d}$

2- If event a is the sending msg of process P_i then we assign a timestamp t_m equals to $C_i(a)$

$$t_m = C_i(a) \quad [\text{from rule 1}].$$

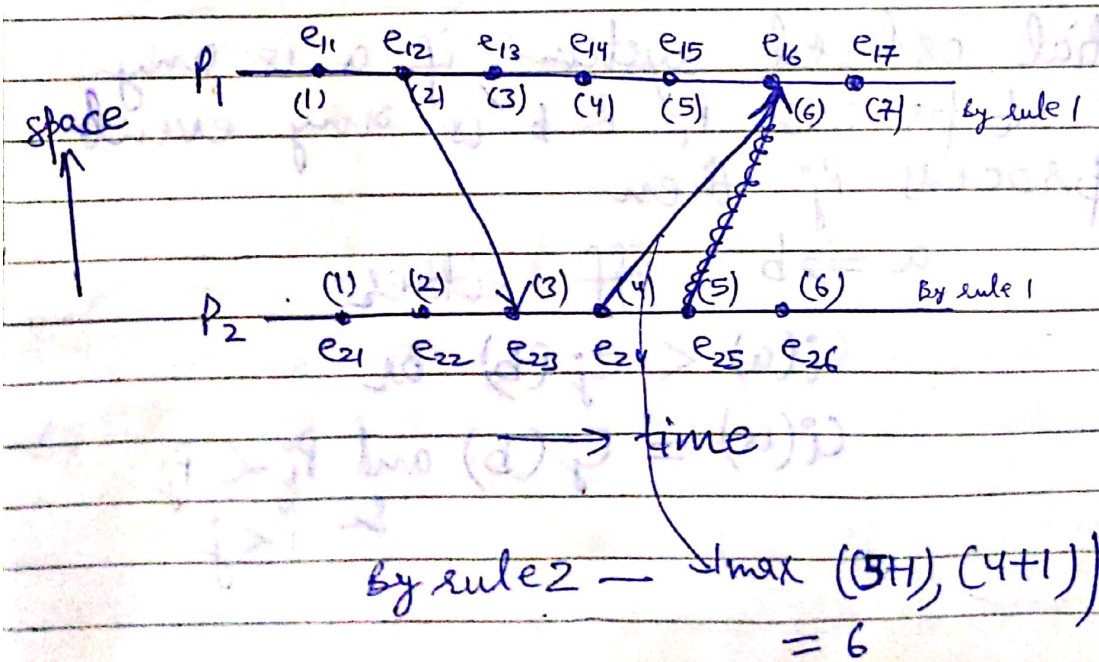
& C_j is set to a value greater than or equal to its present value and greater than t_m

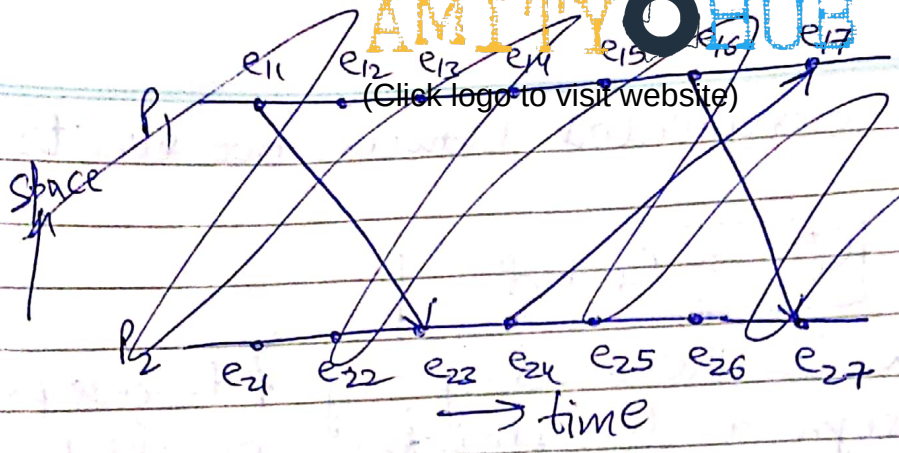
i.e. $\boxed{C_j := \max(C_j, t_m + d)}$

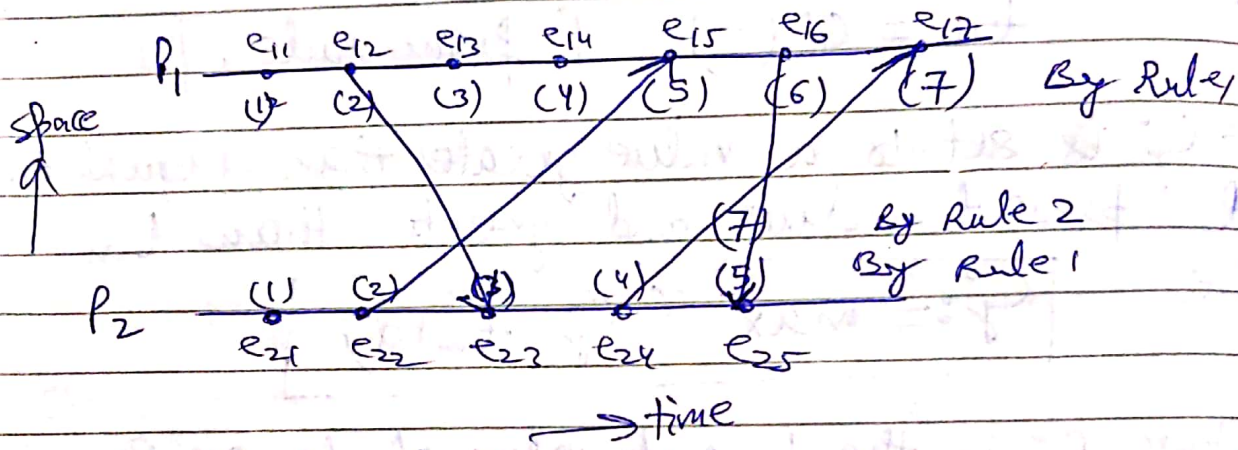
where C_j is the present value of process P_j ~~implemented~~ incremented as per ~~step~~ 1.

Note Usually d is taken to be 1.

Ques. Perform timestamping for the following time-space diagram.







Partial ordered

$a \Rightarrow b$ $\quad P_i < P_j$

$e_{11}, e_{21}, e_{12}, e_{22}\ e_{13}\ e_{23}\ e_{14}\ e_{24}\ e_{15}\ e_{16}\ e_{17}\ e_{25}$

A partial ordered system if a is any event at process P_i & b is any event at process P_j then

$a \Rightarrow b$ iff either

$C_i(a) < C_j(b)$ or

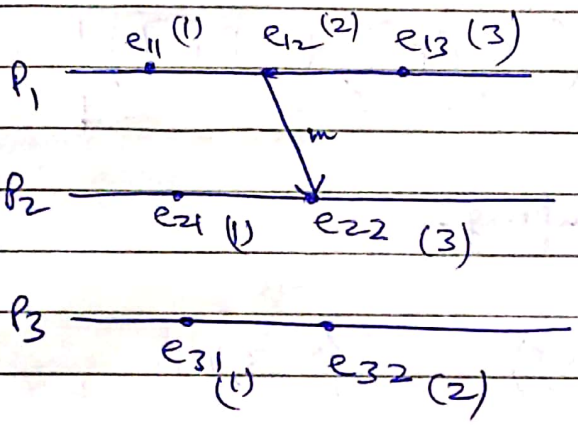
$C_i(a) = C_j(b)$ and $P_i \propto P_j$ & $i < j$

The logical clock whatever timestamping performs it is called virtual time.

Diff. b/w virtual time & Physical time:

1. Physical time can never be zero, but virtual time can be zero.
2. It can happen that virtual time do not increment at all & physical time may increment.
3. Virtual time is set to zero initially.

2/3/05 $C(a) < C(b) \Rightarrow a \rightarrow b$



$e_{11} \rightarrow e_{22}$
(1) < (3)
$e_{11} \rightarrow e_{32}$
(1) < (2)

(From Shivaratri Book)

Vector Clock :- It is a set of nos. which depends on the no. of processes.

If there are 3 processes, set will contain 3 nos.

$\{\downarrow, \downarrow, \downarrow\}$

$[i] \rightarrow P_i$, $j \rightarrow P_j$, $K \rightarrow P_K$ ..... $n \rightarrow P_n$

If n is the no. of processes in a distributed system then each process P_i will have a clock C_i with length n.

$C_i[i]$ is the ith entry of C_i which corresponds to P_i's own logical time &

$C_i[j]$, $j \neq i$ is the j^{th} entry of C_i at P_i indicating the last event occurred at P_j affecting P_i. This concept is implemented in 2 steps:

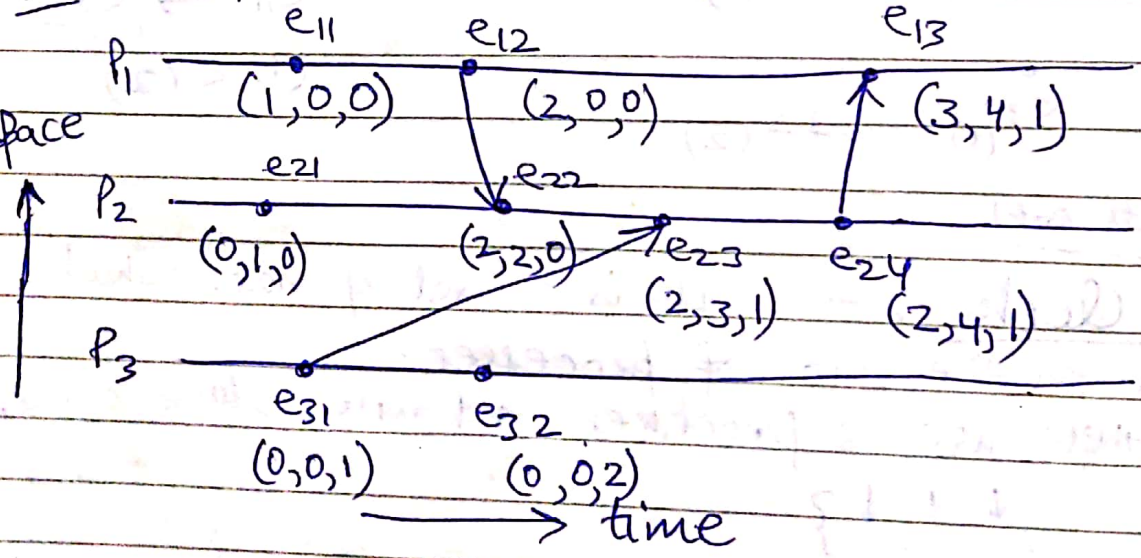
(1) Clock C_i is implemented b/w any two events in process P_i.

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

(2) If event 'a' is sending of msg by process P_i then on receiving the same msg by process P_j will update C_j as following

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

Ques. Perform Timestamping.



$$\text{or} \begin{cases} t^{e_{11}} < t^{e_{32}} \\ t^{e_{32}} < t^{e_{11}} \end{cases}$$ for casually related events

but e_{11} & e_{32} are not casually related events.

(1,0,0) $\not<$ (0,0,2)
(0,0,2) $\not<$ (1,0,0)

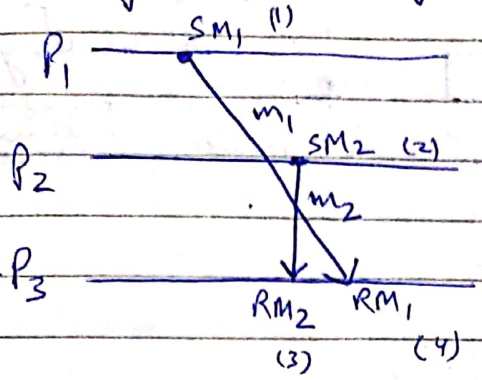
for e_{22}

(1, 2,)
max (0,2) = 2
∴ (2, 2,)

Diff b/w ordering of messages & ordering of events.

Application

→ Ordering of Messages



① Birman-Schiper-Stephenson } Two protocols
② Schiper-Eggli-Sandoz

It is based on broadcasting.

It may be based on broadcasting or point-to-point.

Protocols
Global state
Termination Detection
} 3 topics Do Yourself (Chiranjeevi)

7/3/05 **Mutual Exclusion :-**

→ Shared resource must be Atomic (at any instance — one process)

→ Mutual Exclusion is a soln to avoid concurrency

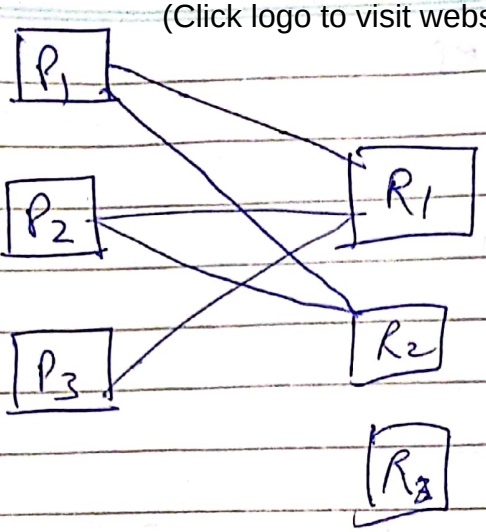
~~It states that~~

→ Process — site

shareable → Resource — critical section

site (a process running on a single system)

→ Semaphore is an integer (binary) value controlling the ~~access~~ entry to critical section; shared variable gives soln to the mutual exclusion problem.



'n' is a variable which is shared which can be accessed by any process or any resource & 'n' is present in shared common RAM (memory).

Semaphore can't be utilized in mutual Exclusion in distributed system.

→ In DS, using message passing we can achieve Mutual Exclusion.

→ Classification of Distributed Mutual Exclusion:
- Token-based
- Non-token based

Non-token based algorithms are the algos which decide whether a site is to get into critical section after successive rounds of message passing. A site satisfying 'assertion' is allowed to get into critical section.

Token-based are based on 'privilege message', a unique token is shared among the sites. A site is allowed to enter its C.S if it possesses the token & it continues to hold the token until the execution of the CS is over.

Diff. b/w idle C.S & idle token state -

Any site getting into mutual exclusion can be
→ Executing critical section
→ Requesting C.S
→ neither of above (idle C.S.)

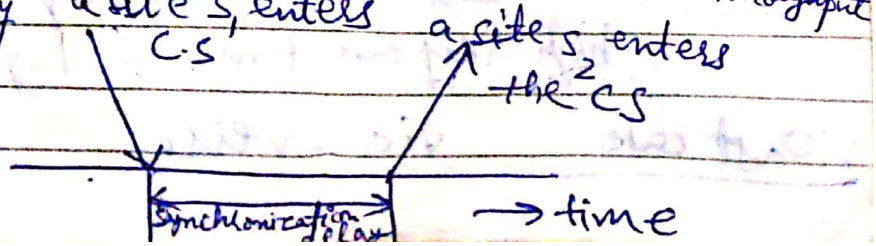
Idle token state → when a particular site is executing non critical section area still holding the token.

Requirements which any mutual exclusion algorithm should meet out :-
(i) Avoid concurrency
(ii) Avoid deadlock
(iii) Avoid starvation
(iv) Fault tolerance → Any mutual exclusion algo should meet out failures.
(v) Algo. should be fair enough.

Performance metrics :
① No. of messages required
② Synchronization delay
③ Response time (C.S execution time for first response)
④ System throughput

There should be less no. of messages.
More the no. of messages, more transmission delay, more waiting time, more response time & less system throughput, the algo will be poor/worst.

Synchronization delay
There should be less synch. delay for less CS time



Response time is the time interval a request waits for its C.S execution to be over after its request messages have been sent out. Response time should be high.

System throughput (It should be high)
It is the rate at which the system execute request for the C.S.

$$\text{throughput} = 1/(Sd + E)$$

where Sd = Synchronization delay
E = Average C.S execution time

Ques. What is Mutual Exclusion? What are the requirements to be meet out? How do you classify Mutual Exclusion? ← Token based, Non-token based
What are the performance metrics for it?
what is the best case & worst case of an algo?

High Load : Pending request for mutual exclusion at a site.
Low Load → When the no. of request messages which can be dealt by a site without making queue.
No load → No msg being transmitted. (more than one request simultaneously)
or Null load

Least load → If only one msg is there.

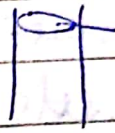
Best case — Less no. of messages, Less synch. delay, high response time & high system throughput.
Worst case — vice-versa.

Ques. What is the simple solution to Mutual Exclusion principle? Explain its drawbacks.

→ Simple solution to Mutual Exclusion is Maintain single control site, grants permission for the CS execution.

Control site will maintain a queue, acc. to priority it will give C.S to the one which is at top of the queue.

(FIFO) (Shortest Job First)



Drawbacks

(i) It is not able to meet out failure. (in case when control site fails)
(ii) It can lead to starvation.
(iii) It is highly prone to congestion.
(iv) Transmission delays are high.

Non-token based algorithms :—

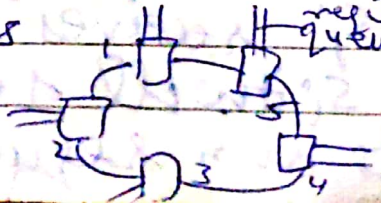
→ Lamport's
→ Richart - Agarwala
→ Maekawa
→ Generalized

Acc. to Lamport's & Richart - Agarwala algos, Each site in a DS maintain a request queue.

$$R_i = \{S_1, S_2 \text{---} S_N\}$$

$$1 \le i \le N$$

All the sites have been sequenced in some manner or ordered & all processes have been timestamped in Lamport's.



$P_i \propto P_j$

Algorithm is divided into 3 stages →

1) Requesting for C.S
2) Executing C.S
3) Releasing C.S

1) Requesting for C.S

(a) A site S_i want to get C.S will send a timestamp request msg to all the sites interested in getting C.S

(b) A site S_j when receives a request msg it will return a timestamp reply and places the requests of others in their own queue.

2) Executing C.S

(a) site S_i will get into/enter C.S depending on either of conditions is satisfied

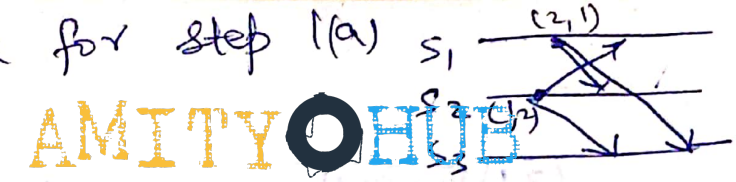
(i) S_i has received a msg with a timestamp larger than its own from all other sites. as compared to all other sites.

(ii) S_i request is at the top of the request queue R_i

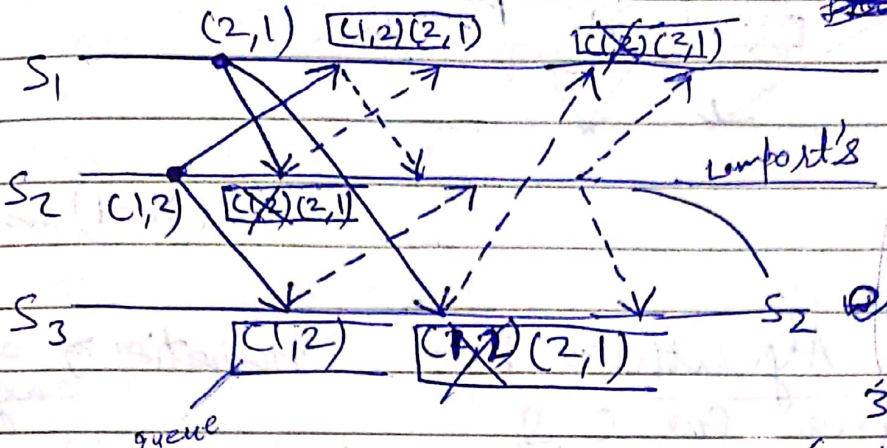
3) Releasing C.S

(a) Site S_i upon ~~executing~~ exiting the C.S will remove its request from the top of its request queue & send a timestamp release msg to all the sites in the request set.

(b) When a site S_j receives a release msg from site S_i it will remove S_i request from its own request queue.

Ex for step 1(a) 

Ex for step 1 (b) S_1, S_2 & S_3 are sites. C.S S_1 & S_2 request for ~~same~~ ~~should be released~~.

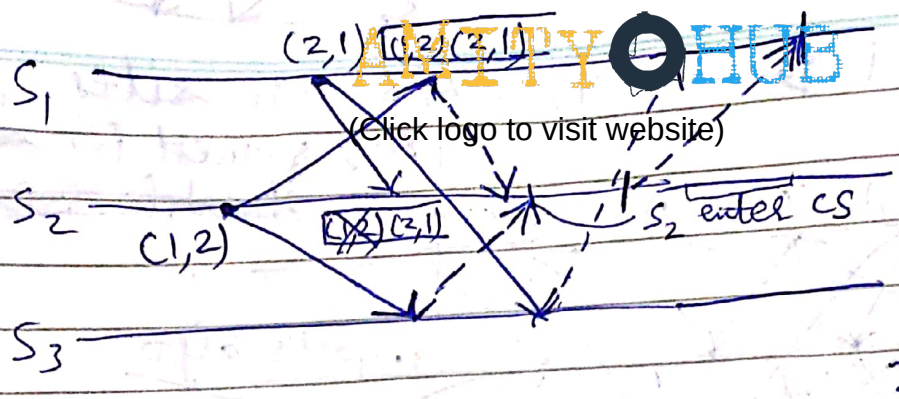


$3(N-1) = 3 \times 2$
(N-1) — request
(N-1) — reply msgs
(N-1) — release

Richart – Agarwala algo
It is the optimized form of Lamport algo.
In this reply & release messages are combined.
No. of messages have been reduced to $2(N-1)$.
(Do). (Proof also.)

9/3/05 Richart – Agarwala :-

1. Requesting
   (a) same as lamport
   (b) Reply should come from those who are not interested in entering C.S or those who have larger timestamp than S_i.
2. Executing
3. Releasing / Exiting
   It will send reply messages to all other sites which have been deferred or avoided in step 1 earlier.



Richart-Agrawala algo

2(N-1) messages

Maekawa's Algorithm :- (optimization of above 2 algos.)

① Requesting for C.S

These are certain conditions which should be met

(a) Every site contains a request set. Request should be send to only those sites which are present in its request set.
(a) update set.

(b) Execute the C.S when you get reply messages from all sites which are there in request set.

For two sites S_i & S_j $\forall i \forall j$

$$R_i \cap R_j \neq \phi$$

Request sets R_i, R_j

③ Every set should have atleast his own entry.

3. Release messages
1. Request "
2. Reply "
} 3 type of messages

No. of sites in any request set $= \sqrt{N}$

Messages conveyed $= 3\sqrt{N}$

~~Conditions~~ Construction of RS :-

1. $\forall i \forall j$, $i \neq j$, $1 \leq i, j \leq N$
   then $R_i \cap R_j \neq \phi$

2. $\forall i$, $1 \leq i \leq N$, $S_i \in R_i$

3. $\forall i$, $1 \leq i \leq N$, $|R_i| = K$
   where K is any integer no.

(All request sets should be equal in number or size)

4. Any site S_j is contained in K no. of R_i's; where $1 \leq i, j \leq N$

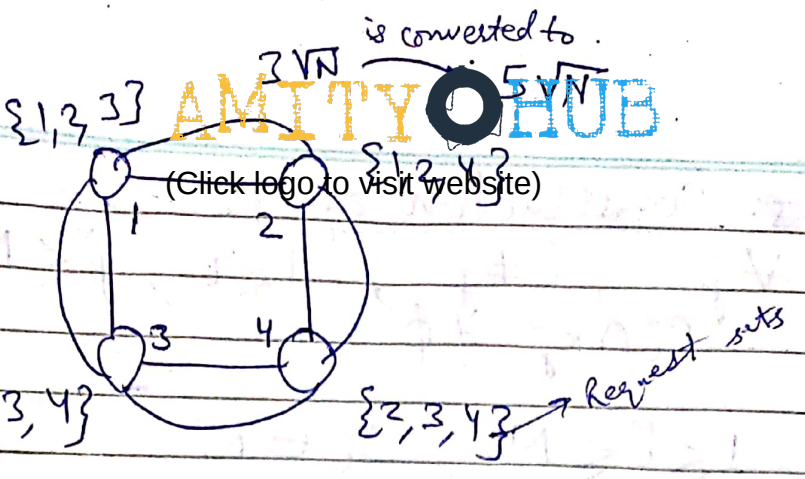
* Relation b/w N & K
  where N = no. of sites & K = ~~size of request set~~

$$N = K(K-1) + 1 \qquad K = \sqrt{N}$$

Maekawa's algo leads to deadlock.

To avoid deadlock, 3 more messages are to be transmitted → yield messages.
- → Failed
- → Inquire

10/3/05



Generalized :-

Based on information structure for site S_i consist of

(i) request set, R_i
(ii) inform^n set, I_i
(iii) status set, S_i
and
CSSTAT variable

(Site's knowledge of status of CS)

status set contains the sites whose information is there in inform^n set.

Executing, Requesting, Idle

$\{E, R, I, I\}$ I_i

$\{7, 5, 9, 2\}$ S_i

Information set contains the status of the site.

Critical section status variable → denotes whether CS is free or not.

This algo is to avoid deadlock.

Two more conditions : alongwith 4 conditions of Maekawa algo

1. $\forall i, 1 \leq i \leq N$ & $S_i \in I_i$

$I_i \subset R_i$

subset

belong to

2. $\forall i \; \forall j ; 1 \le i, j \le N$

$(I_i \cap I_j \neq \phi) \vee (S_i \in R_j \wedge S_j \in R_i)$

Phases

1(a) same as Maekawa

1(b) Once you have sent the request messages to all they'll send grant messages by taking care of few actions

(i) check CSSTAT variable whether it is free or not.

(ii) check its own status in the inform^n set

(iii) Maintain a queue which has sites in increasing timestamp order.

if S_i is not at top of queue, it will not send grant msg.

2. Receives grant msgs from all sites then goto execution stage.

3. Releasing C-S : After execution

(i) free itself in CSSTAT variable

(ii) send grant msgs to those sites which are present in its inform^n set & make the status idle.

(iii) Delete the entry of S_i from top of queue.

It does not lead to deadlock b'coz request sets are not compared but inform^n sets are compared. Inform^n set & CSSTAT variable does not have to wait for reply msg for longer time.

Information structure can be < static (Array)
Dynamic (Linked list)

Token-Based Algorithms :-

- Suzuki – Kasami
- Singhal's Heuristic
- Raymond's Tree

Request Messages } Type of messages
Token Messages }

Suzuki – Kasami : It is based on broadcasting concept.

① Requesting Token state
② Executing Token state
③ releasing / Idle CS
④ Idle token state → Holding the token but not executing CS.

It sends requesting msg to all the sites in Distributed system. The site which receives the request msg is in one of these 4 states.

① requesting token state
② Executing " "
③ Idle CS
④ Idle token state

✓ It sends token back to the site on the following 2 constraints

① To decide among all request site
② To distinguish b/w outdated & current request

To handle above 2 constraints, it maintains two arrays.

① Each site maintains an array S [1, ...N]

1, ... N are sequence no. of request msg

Ordered set (S_i°, S_n) S_i° = site number, S_n = Sequence no.

For first request sequence S_1 (1,1) → sequence no. for II nd S_2 (1,2), for III rd S_3 (1,3)

Any j^{th} entry in the array is the most recent request being handled i.e. it has received token. $S[i, \ldots j \ldots N]$

A token itself will have an array which contains the sequence no. of each sites.

$T[\quad]$

A token has queue for itself based on FIFO. and not based on timestamping.

FIFO leads to starvation.

Queue is maintained for n no. of requests.

Priority to give token is for needy i.e. which has largest sequence no. then the request at the top of queue. This algo is ultimately leading to starvation. Performance is 0 when nobody send request for CS.

or Performance is N (worst case). (N messages)

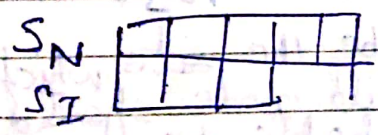
Singhal's Heuristic : It is based on send request to sites either they have something in common or directly like maekawa's.

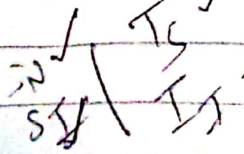
Request set

Information set → Status of those sites who are interested for token in present future.

Send request only to subset of sites.

Each site maintain 2 arrays





14/3/05

→ Singhal's Heuristic

Sequence no. | 7 | 9 | 3 | 4 | SN
Information | R | E | N | H | SI
} 2 arrays maintained by each site.

→ Requesting R
→ Executing E
→ Idle N (Not interested in CS)
→ Hold H

TS				
TI				

steps:
1(a) The request is sent to requesting/executing/holding sites.

(b) Array should get updated and sequence no. increased by 1.

2. Allocation by all the sites in queue alongwith the token & compare the sequence no. update TS.

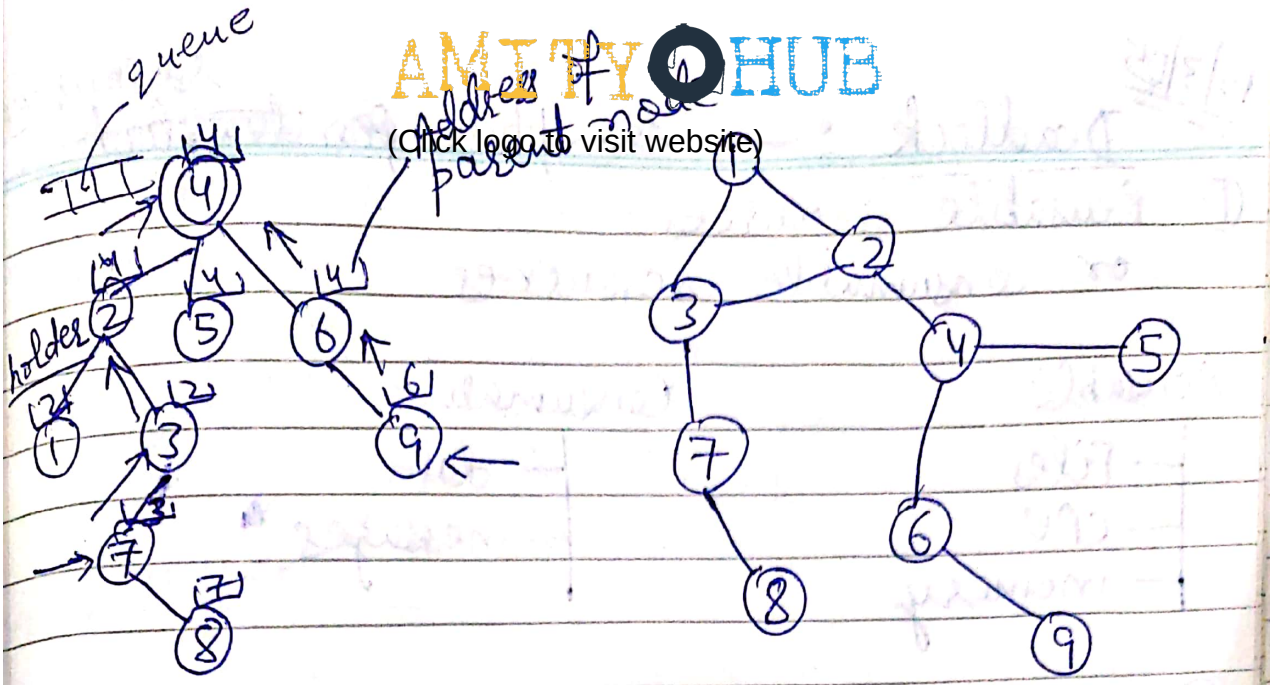
In releasing phase, make your own entry N in the TI array.

→ Raymond's Tree : Prefer to make binary tree.

① The root node should have the token.

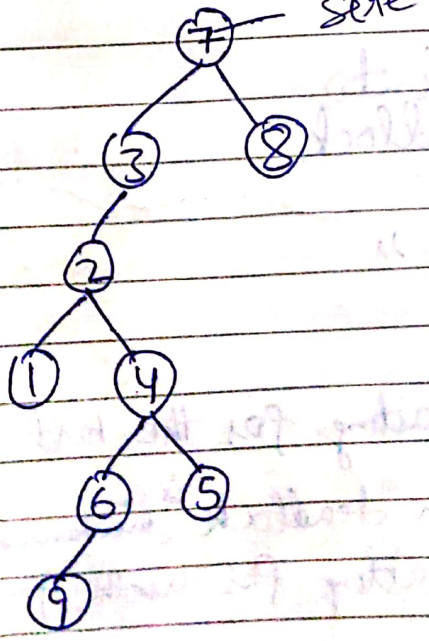
② The child node should be directly reachable to the parent node. will be the site which

Directly or indirectly reachable.



We use binary trees b'coz no. of messages would be reduced by half.

Requesting
follow the path to root node through its parent.
On this basis we send the request.
Token will be allocated to the request.
Token is released.



Implementation Issues
Maintaining queue.