Lamport's Mutual Exclusion algorithm:

→ It is a non token based algorithm It executed using timestamp.

→ When a process need to enter critical section, we use lamport algorithm and ricard agrawala algorithm.

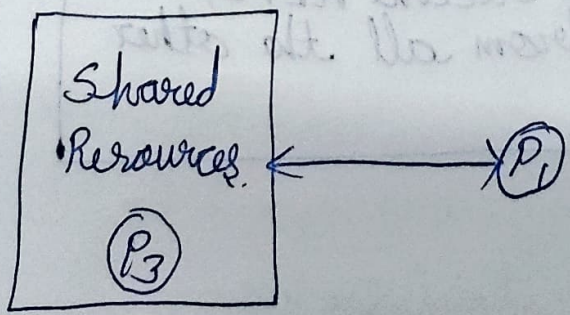
→ Lamport algorithm will be based on three messages.

* Request
* Reply
* Release.

Request :- When a process need to enter critical section it sends request message.

Reply ⇒ When the site allows a process request it sends the reply message.

Release ⇒ When process exited from critical section it sends the release message.



ameter in the non-blocking call also gets set with the handle of a location that user process can later check for the completion of the synchronous send ation.

buffer_i
kernel_i

→ Queue is maintained for each process to store critical section requests ordered by their timestamps.

→ Message are delivered in FIFO order.

F	I	F	O

→ Timestamp → Critical section request → Lamport's logical clock.

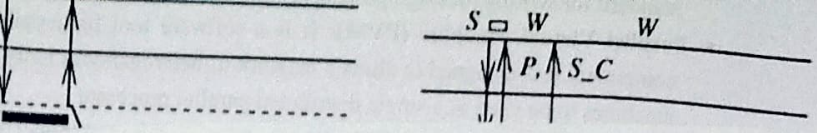
→ Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp.

→ The execution of critical section request is always in the order of their timestamp.

2 Requirements to enter into critical section:-

1:- The T_i is on top of the Queue.

2:- It should receive REPLY message from all the other processes.



Algorithm:

i) To enter critical section:-

→ Request message "Request (ts$_i$, i)" is sent to all other sites and places the request on queue$_i$.

→ After receiving the request message, a timestamped REPLY message is sent and places the request on queue$_j$.

ii) To execute the critical section:-

→ If it has received the message from all the other sites.

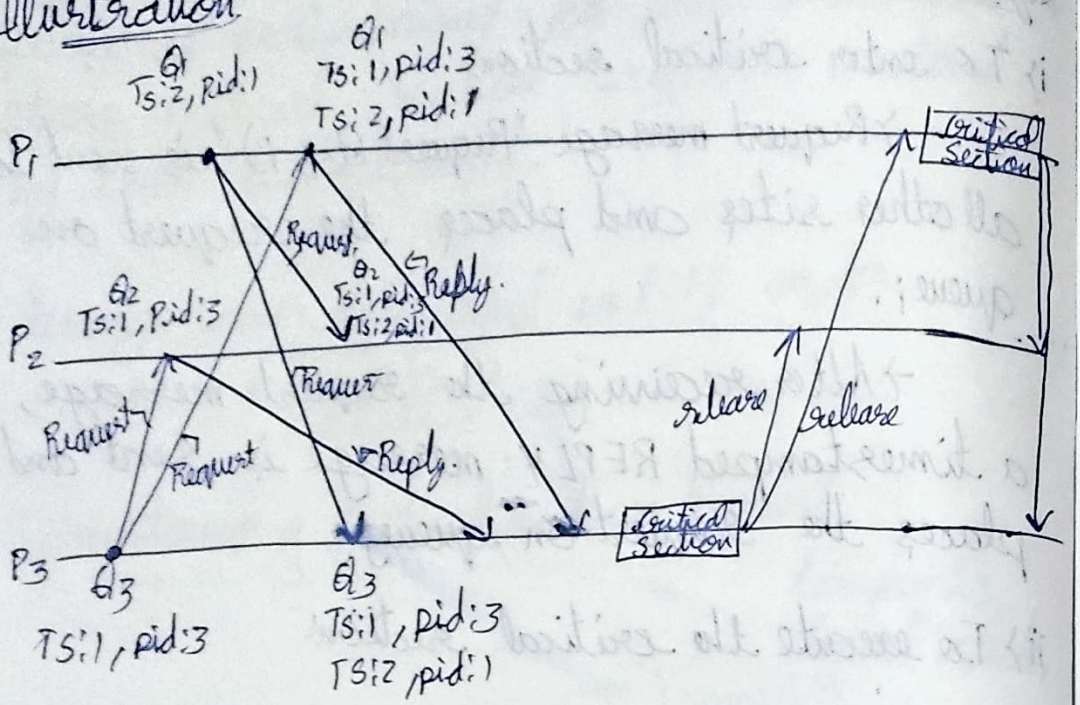
→ Its own request is at top of the queue.

iii) To release the critical section:-

→ When a P$_i$ exits the critical section, it removes its own request from top of its queue and sends a timestamped RELEASE message to all other sites.

P.T.O →

Illustration:



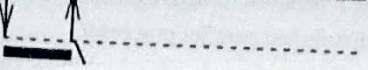
Steps: Scenario: 3 processes (P_1, P_2, P_3).

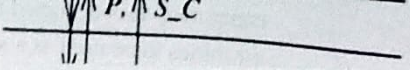
1) Request phase:-
- P_3 sends a Request (timestamp = 1) to P_2 & P_1
- P_1 sends a Request (timestamp = 2) to P_2 & P_3.

2) Reply phase:
- P_1 and P_2 sends Reply to P_3.
- P_2 and P_3 sends reply to P_1

3) Critical section:-
- P_3's request (timestamp = 1) is processed first.
- P_1 waits until P_3 release the critical section

4) Release phase:-
- P_3 sends a release message to P_2 & P_1
- P_1 enters the critical section.





performance

→ Request = (n-1)

→ Reply = (n-1)

→ Release = (n-1)

→ Critical section = 3(n-1).

Ricart Agrawala's Algorithm:

→ It is a non token based algorithm. It executed using timestamp.

→ When a process need to enter critical section, we use lamport algorithm and ricart algorithm.

→ Ricart agrawala's algorithm is an optimization on lamport algorithm.

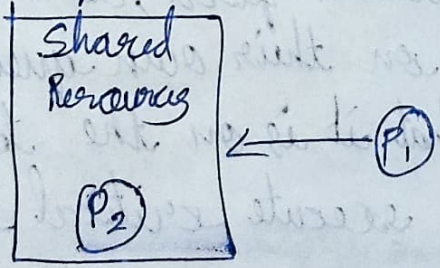
→ Ricart-Agrawala's algorithm uses only types of messages:-

* Request
* Reply.

→ It ensures mutual exclusion in a distributed system through message passing.

Request ⇒ When a process need to enter critical section it sends request message.

Reply ⇒ When the site allows a process request it sends the reply message to all the deferred requests send by processes.



ter in the non-blocking call also gets set with the handle of a location that process can later check for the completion of the synchronous send
.
or

buffer_i
kernel_i

→ Queue is maintained for each process to store critical section requests ordered by their timestamps.

→ Messages are delivered in FIFO order.

F	IF	O

→ Timestamp → Critical section request → Lamports logical clock.

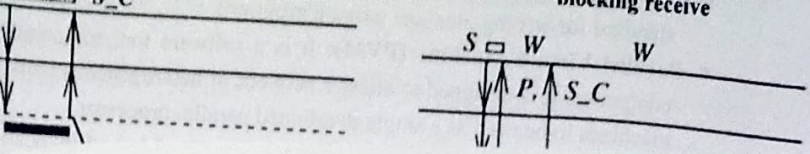
→ Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp.

→ The execution of critical section requests is always in the order of their timestamp.

Requirements to enter critical section

1. It should receive REPLY message from all the other processes

→ The process that requested first, don't put their own timestamp on their own queue because, it is clear that it is on the top & its already in edge to execute critical section.



Algorithm :-

i) Requesting the critical section :

→ When a process P_i wants to enter the critical section it sends a request message to all the other processes containing.
"Request (ts_i, i),
→ Wait for Reply messages from all other processes.

ii) Receiving a Request :-

→ When a process receives a Request from another process, first it checks/compares the T_i (timestamp) with T_j (current clock) and decide weather to grant Permission (send) Reply. or to Delay reply.

- Grand permission :- If P_j is not in the critical section or has a lower-priority request
- Delay Reply :- If P_j is in the critical section or has a higher-priority request.

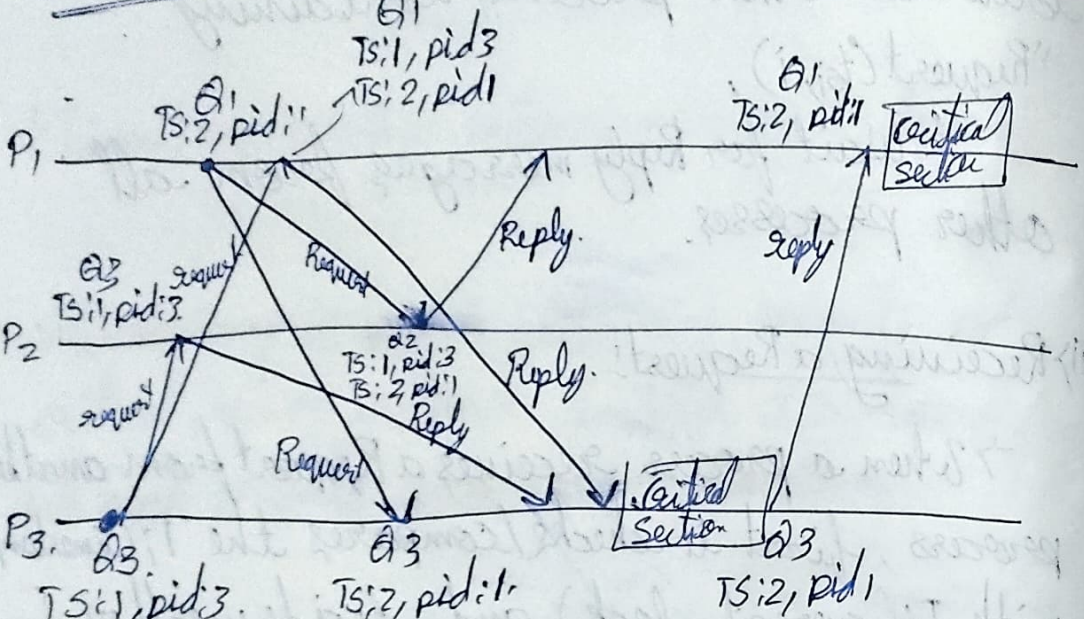
iii) Entering the critical section :-

→ Processes P_i enters the critical section after receiving Reply message from all other processes.

iv) Exiting the critical section:

→ Upon exiting the critical section, P_i processes any deferred requests by sending Reply messages to the respective processes.

Illustration:



Steps & scenario; 3 processes (P_1, P_2, P_3).

i) Timestamps:

→ P_3 requests CS with $T_1 = 1$

→ P_2 requests CS with $T_2 = 2$.

ii) Request phase:

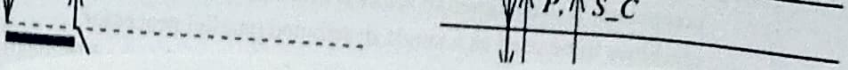
→ P_3's request is prioritized because $T_3 = 1$ is the smallest timestamp.

→ P_1 & P_2 defer their replies to P_3.

iii) Critical section Entry:

→ P_3 enters the CS first.

→ After P_3 exits, P_1's request is processed next,



Performance :-

-> Request = (N-1)

-> Reply = (N-1).

-> Critical section execution = 2(N-1).

Comparison with other Algorithms

Algorithm.	Message complexity.	Dependency.	Fault Tolerance
Lamport Mutual Exclusion	3(N-1)	Logical clocks + queues.	Limited
Ricart - Agrawala's.	2(N-1)	Responsiveness of processes.	Moderate
Token-Based.	≈1	Token availability	High

Snapshot Algorithm for FIFO channels:-

Each distributed application has number of processes running on different physical servers. These processes communicate with each other through messaging channels.

> A snapshot captures the local state of each process along with the state of each communication channel.

Snapshots are required to:-

- checkpointing
- Collecting garbage
- Detecting deadlocks
- Debugging.

Chandy-Lamport algorithm:-

→ This algorithm always captures the consistent global state of a distributed system.

→ Any process in the distributed system can initiate this global state recording algorithm using a special message called "Marker".

process can later check for the completion of the synchronous send

buffer_i
kernel_i

-> This marker traverse the distributed system across all communication channel and each process to record its own state.

-> In the end, the state of entire system (Global state) is recorded.

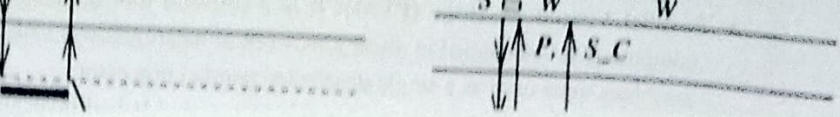
-> This algorithm does not interfere with normal execution of processes.

How it works:

-> This algorithm can be initiated by any process by executing the "Marker Sending Rule", records its local state and sends a marker on each outgoing channel.

-> The receiving process executes the "Marker Receiving Rule" once received the Marker. If in case, the process has not yet recorded its local state, it sets the state of the incoming channel on which the marker has been received as "Empty" or "Null".

-> The incoming channel on which the marker has been received is set as empty in order to block any other messages coming on this channel while recording is being done.



→ Then the receiving process executes the "Marker Sending Rule" to record its own local state + record the state of other incoming channels i.e expect the channel on which the MARKER is received.

→ The collection of these two local snapshots determine the global state of the distributed system.

Chandy Lamport Algorithm:-

Marker sending Rule:- for process i

1) Process i records its state.
2) For each outgoing channel C on which a marker has not been sent, i sends a marker along C before i sends further message along C.

Marker receiving Rule:- for process j

On receiving a marker along channel C;

If j has not recorded its state then.

Record the state of C as the empty set follow the "Marker Sending Rule".

Else:

Record the state of c as the set of messages received along C after j's state was recorded and before j received the marker along C.

Complexity:

→ The recording part of a single instance of the algorithm requires $O(e)$ messages & $O(d)$ time, where e is the number of edges in the network & d is the diameter of the network.