M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Distributed Systems
Course Code: CSE20/CSE751
Credits: 3:0:0

Term: Oct 2021-Feb 2022

Faculty:
Sini Anna Alex

RAMAIAH
Institute of Technology

Correctness

L1: S_i has received a message with timestamp larger than (ts_i, i) from all other sites.

L2: S_i's request is at the top of $request\_queue_i$.

Theorem: Lamport's algorithm achieves mutual exclusion.

Proof:

- Proof is by contradiction. Suppose two sites S_i and S_j are executing the CS concurrently. For this to happen conditions L1 and L2 must hold at both the sites *concurrently*.
- This implies that at some instant in time, say t, both S_i and S_j have their own requests at the top of their *request_queues* and condition L1 holds at them. Without loss of generality, assume that S_i's request has smaller timestamp than the request of S_j.
- From condition L1 and FIFO property of the communication channels, it is clear that at instant t the request of S_i must be present in $request\_queue_j$ when S_j was executing its CS. This implies that S_j's own request is at the top of its own *request_queue* when a smaller timestamp request, S_i's request, is present in the $request\_queue_j$ – a contradiction!

Correctness

Theorem: Lamport's algorithm is fair.
Proof:

- The proof is by contradiction. Suppose a site S_i's request has a smaller timestamp than the request of another site S_j and S_j is able to execute the CS before S_i.
- For S_j to execute the CS, it has to satisfy the conditions L1 and L2. This implies that at some instant in time say t, S_j has its own request at the top of its queue and it has also received a message with timestamp larger than the timestamp of its request from all other sites.
- But *request_queue* at a site is ordered by timestamp, and according to our assumption S_i has lower timestamp. So S_i's request must be placed ahead of the S_j's request in the *request_queue*$_j$. This is a contradiction!

An optimization

- In Lamport's algorithm,REPLY messages can be omitted in certain situations. For example, if site S_j receives a REQUEST message from site S_i after it has sent its own REQUEST message with timestamp higher than the timestamp of site S_i's request, then site S_j need not send a REPLY message to site S_i.
- This is because when site S_i receives site S_j's request with timestamp higher than its own, it can conclude that site S_j does not have any smaller timestamp request which is still pending.
- With this optimization, Lamport's algorithm requires between $3(N-1)$ and $2(N-1)$ messages per CS execution.

Ricart-Agrawala Algorithm

- The Ricart-Agrawala algorithm assumes the communication channels are FIFO. The algorithm uses two types of messages: REQUEST and REPLY.
- A process sends a REQUEST message to all other processes to request their permission to enter the critical section. A process sends a REPLY message to a process to give its permission to that process.
- Processes use Lamport-style logical clocks to assign a timestamp to critical section requests and timestamps are used to decide the priority of requests.
- Each process p_i maintains the Request-Deferred array, RD_i, the size of which is the same as the number of processes in the system.
- Initially, $\forall i\ \forall j$: $RD_i[j]=0$. Whenever p_i defer the request sent by p_j, it sets $RD_i[j]=1$ and after it has sent a REPLY message to p_j, it sets $RD_i[j]=0$.

Description of the Algorithm

Requesting the critical section:

(a) When a site S_i wants to enter the CS, it broadcasts a timestamped REQUEST message to all other sites.

(b) When site S_j receives a REQUEST message from site S_i, it sends a REPLY message to site S_i if site S_j is neither requesting nor executing the CS, or if the site S_j is requesting and S_i's request's timestamp is smaller than site S_j's own request's timestamp. Otherwise, the reply is deferred and S_j sets RD_j[i]=1

Executing the critical section:

(c) Site S_i enters the CS after it has received a REPLY message from every site it sent a REQUEST message to.

Description of the Algorithm

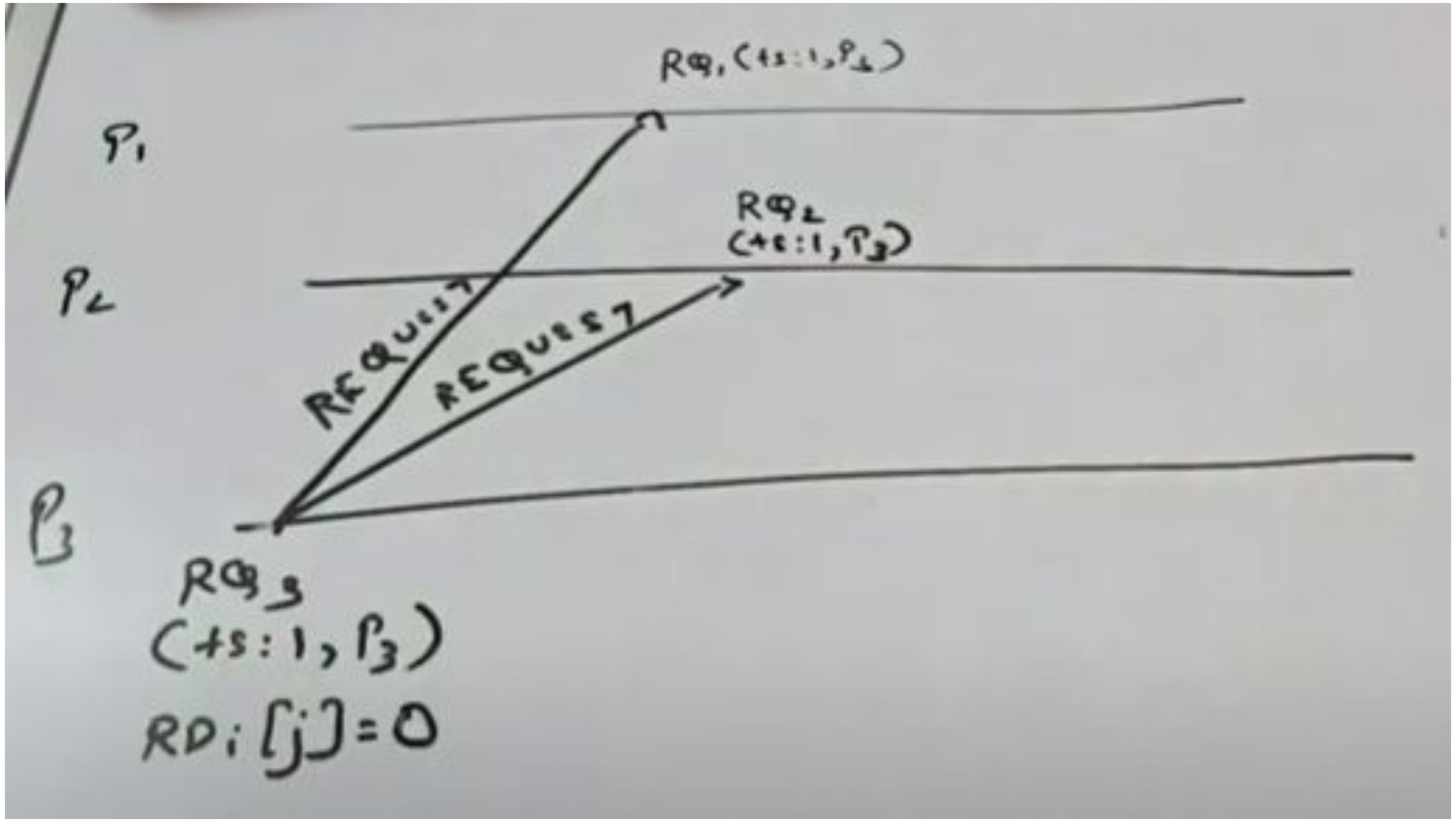
Releasing the critical section:

(d) When site S_i exits the CS, it sends all the deferred REPLY messages: $\forall j$ if RD_i[j]=1, then send a REPLY message to S_j and set RD_i[j]=0.

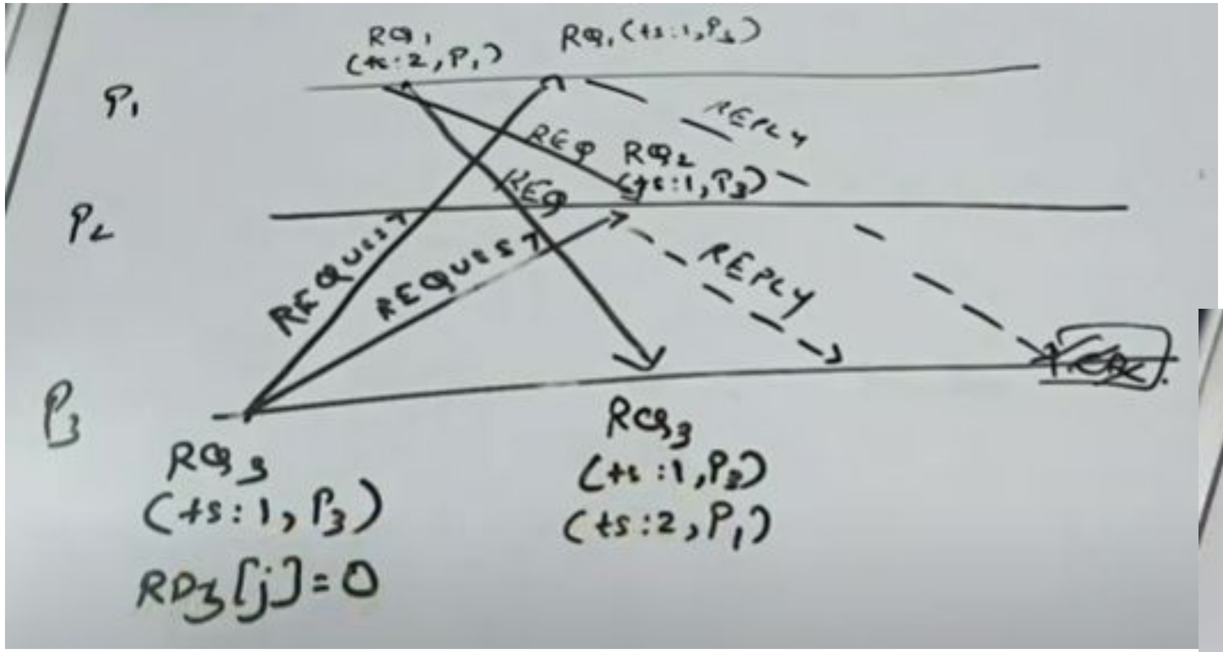
Notes:

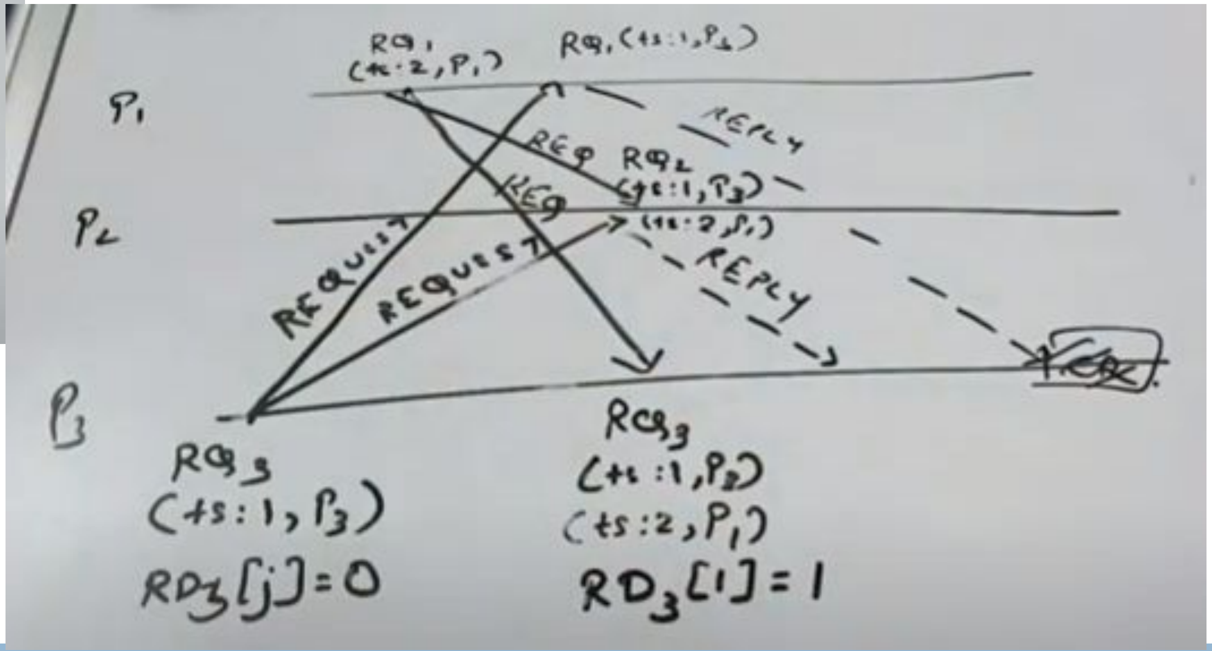
- When a site receives a message, it updates its clock using the timestamp in the message.
- When a site takes up a request for the CS for processing, it updates its local clock and assigns a timestamp to the request.

Example

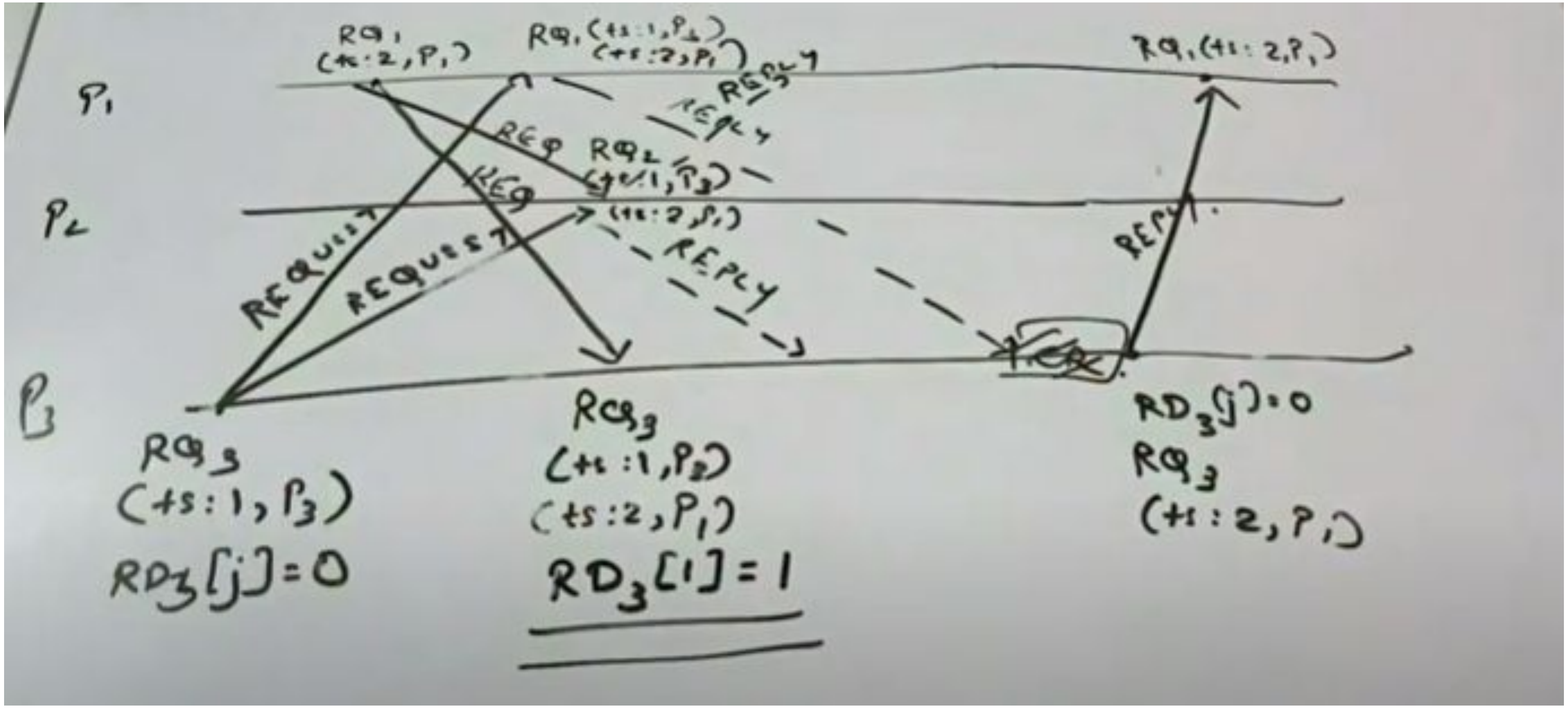


Example


P1
P2
P3
REPLY
REQUEST
REQUEST
REPLY
RQ3
(ts:1,P3)
RD3[j]=0
RQ3
(ts:1,P3)
(ts:2,P1)


P1
P2
P3
REPLY
REQUEST
REQUEST
REPLY
RQ3
(ts:1,P3)
RD3[j]=0
RQ3
(ts:1,P3)
(ts:2,P1)
RD3[1]=1

Example


P1
P2
P3
REQUEST
REQUEST
REPLY
REPLY
REPLY
REPLY
RQ3
(ts:1, P3)
RD3[j]=0
RQ3
(ts:1, P2)
(ts:2, P1)
RD3[1]=1
RD3[j]=0
RQ3
(ts:2, P1)

RAMAIAH
Institute of Technology

Thank you