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

Lodha and Kshemkalyani's fair mutual exclusion algorithm

Decreases the message complexity of the Ricart–Agrawala algorithm by using the following interesting observation: when a site is waiting to execute the CS, it need not receive REPLY messages from every other site. To enter the CS, a site only needs to receive a REPLY message from the site whose request just precedes its request in priority.

Lodha and Kshemkalyani’s fair mutual exclusion algorithm

System model

Each request is assigned a priority *ReqID* and requests for CS access are granted in the order of decreasing priority. We will defer the details of what *ReqID* is composed of to later sections. The underlying communication network is assumed to be error free.

Lodha and Kshemkalyani's fair mutual exclusion algorithm

Uses three types of messages (REQUEST, REPLY, and FLUSH) and obtains savings on the number of messages exchanged per CS access by assigning multiple purposes to each. For the purpose of blocking a mutual exclusion request, every site Si has a data structure called *local_request_queue* (denoted as LRQi), which contains all concurrent requests made with respect to Si's request, and these requests are ordered with respect to their priority.

Multiple uses of a REPLY message

1. A REPLY message acts as a reply from a process that is not requesting.
2. A REPLY message acts as a collective reply from processes that have higher priority requests.

A REPLY(R_j) from a process P_j indicates that R_j is the request made by P_j for which it has executed the CS. It also indicates that all the requests with priority $\geq$ priority of R_j have finished executing CS and are no longer in contention.

RAMAIAH
Institute of Technology

Uses of a FLUSH message

Similar to a REPLY message, a FLUSH message is a logical reply and denotes a collective reply from all processes that had made higher priority requests. After a process has exited the CS, it sends a FLUSH message to a process requesting with the next highest priority, which is determined by looking up the process's local request queue. When a process P_i finishes executing the CS, it may find a process P_j in one of the following states:

1. R_j is in the local queue of P_i and located in some position after R_i, which implies that R_j is concurrent with R_i.
2. P_j had replied to R_i and P_j is now requesting with a lower priority. (Note that in this case R_i and R_j are not concurrent.)
3. P_j's requst had higher priority than P_i's (implying that it had finished the execution of the CS) and is now requesting with a lower priority. (Note that in this case R_i and R_j are not concurrent.)

Multiple uses of a REQUEST message

Considering two processes P_i and P_j, there can be two cases:

Case 1 P_i and P_j are not concurrently requesting. In this case, the process which requests first will get a REPLY message from the other process.

Case 2 P_i and P_j are concurrently requesting. In this case, there can be two subcases:

1. P_i is requesting with a higher priority than P_j. In this case, P_j's REQUEST message serves as an implicit REPLY message to P_i's request. Also, P_j should wait for REPLY/FLUSH message from some process to enter the CS.
2. P_i is requesting with a lower priority than P_j. In this case, P_i's REQUEST message serves as an implicit REPLY message to P_j's request. Also, P_i should wait for REPLY/FLUSH message from some process to enter the CS.

Maekawa's algorithm

Requesting the critical section:

(a) A site S_i requests access to the CS by sending REQUEST(i) messages to all sites in its request set R_i.

(b) When a site S_j receives the REQUEST(i) message, it sends a REPLY(j) message to S_i provided it hasn't sent a REPLY message to a site since its receipt of the last RELEASE message. Otherwise, it queues up the REQUEST(i) for later consideration.

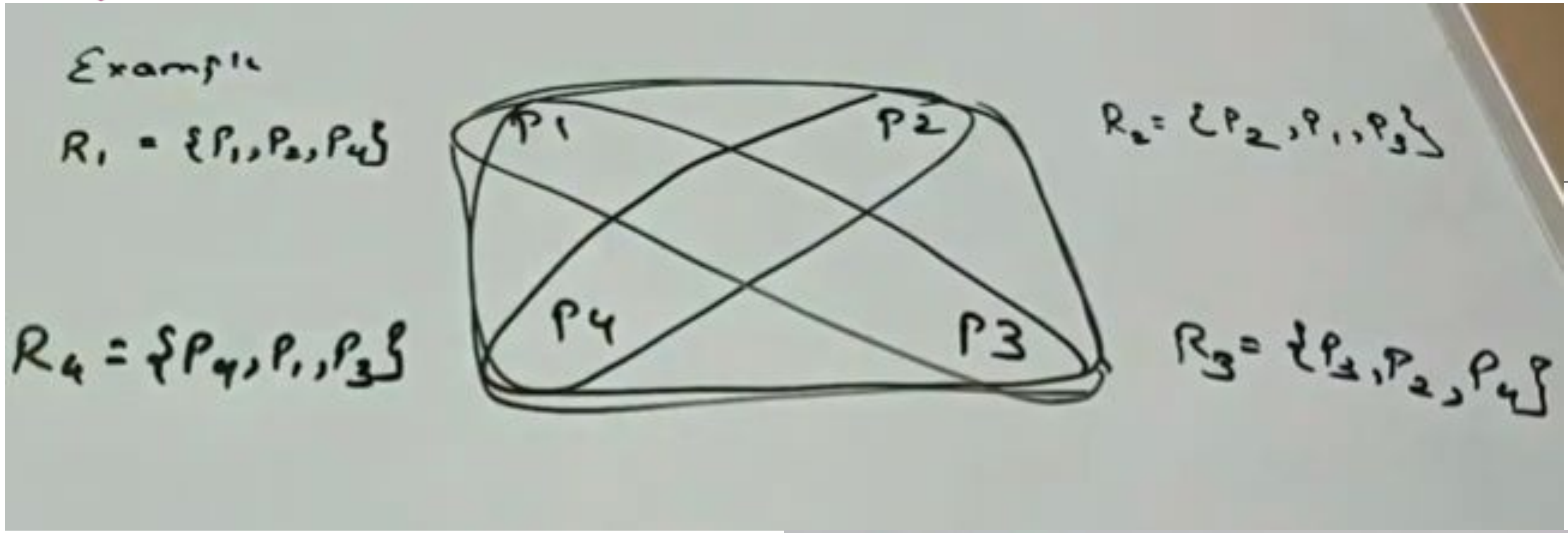
Executing the critical section:

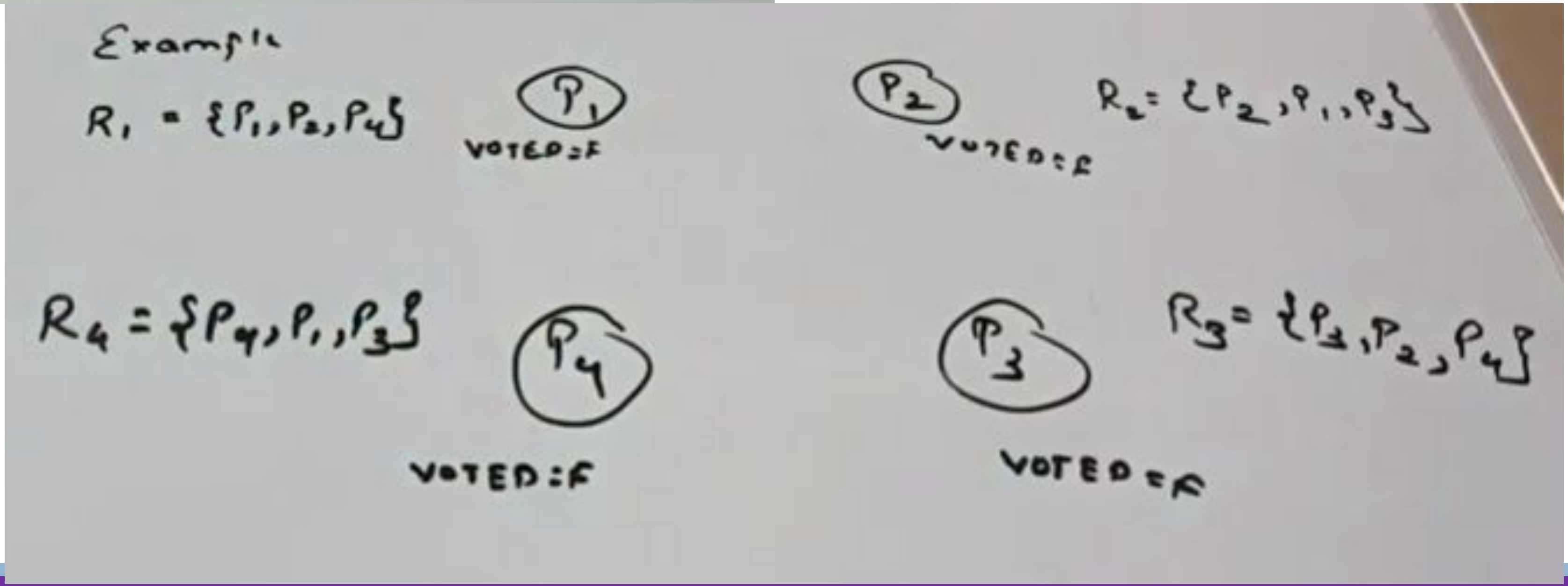
(c) Site S_i executes the CS only after it has received a REPLY message from every site in R_i.

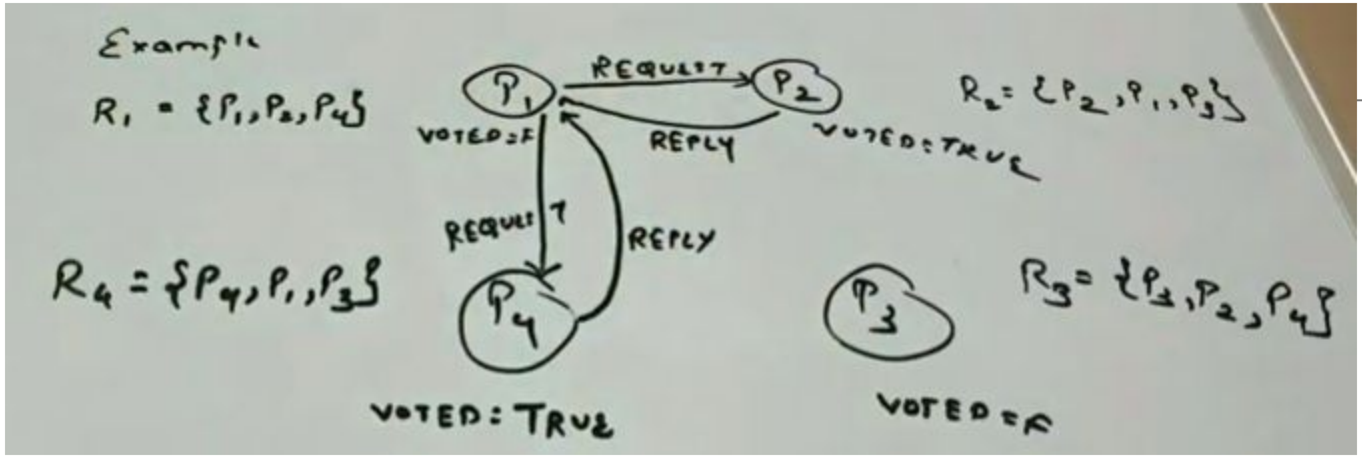
Releasing the critical section:

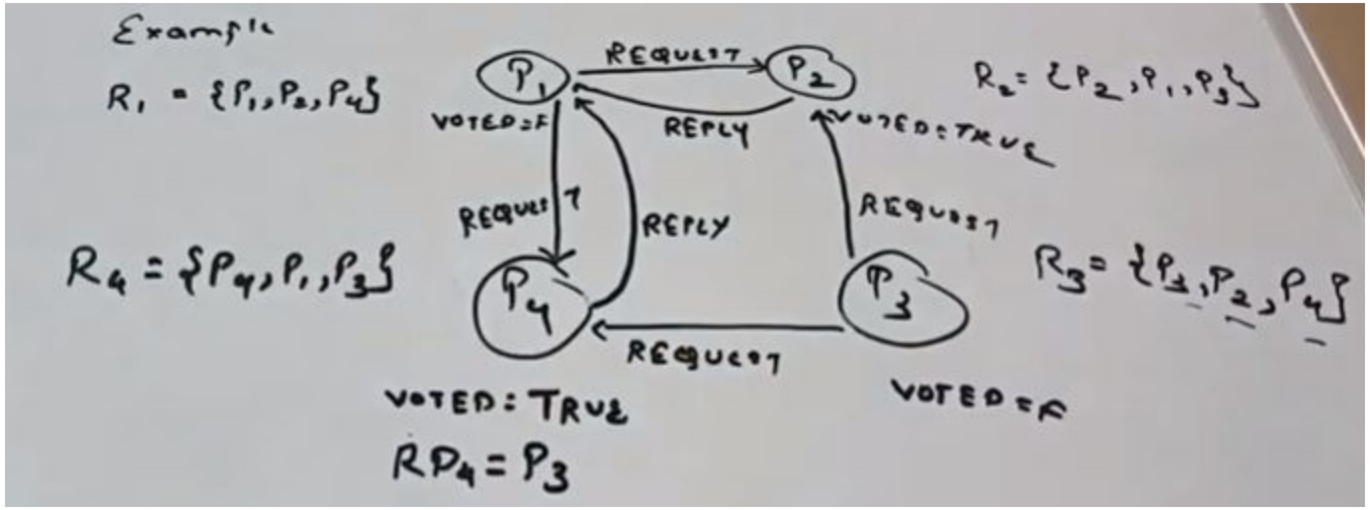
(d) After the execution of the CS is over, site S_i sends a RELEASE(i) message to every site in R_i.

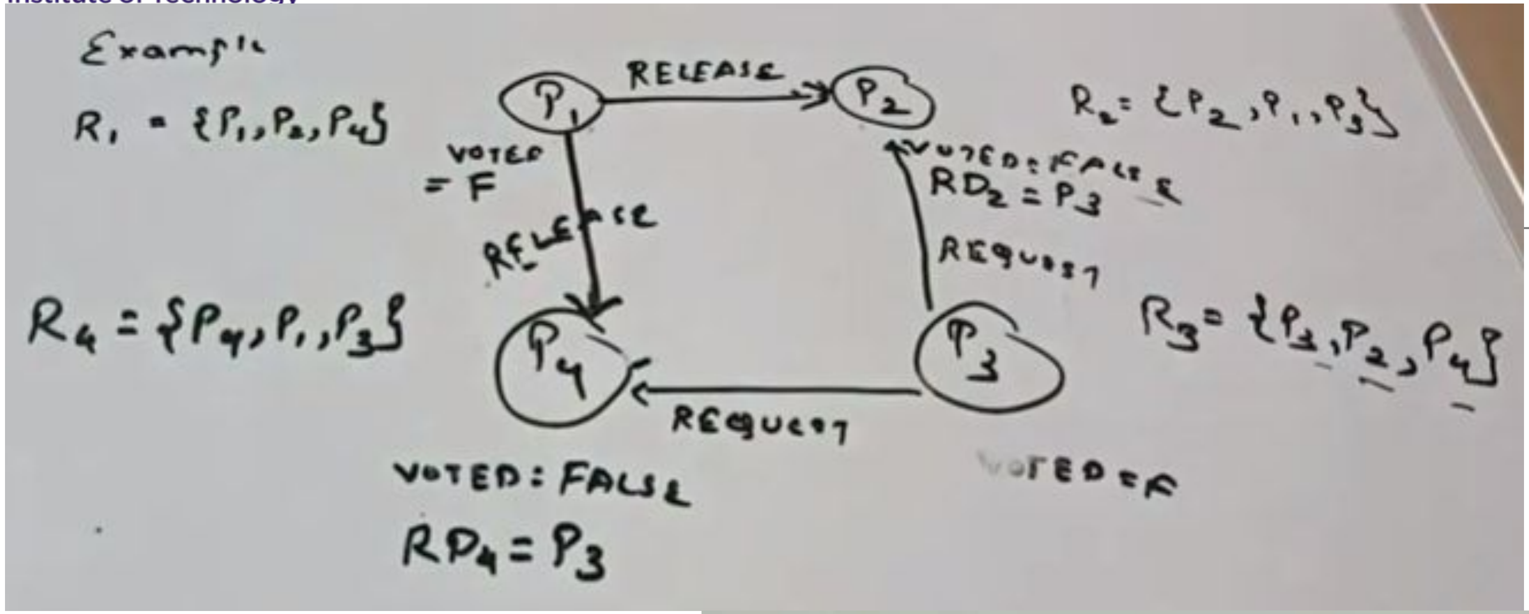
(e) When a site S_j receives a RELEASE(i) message from site S_i, it sends a REPLY message to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty, then the site updates its state to reflect that it has not sent out any REPLY message since the receipt of the last RELEASE message.

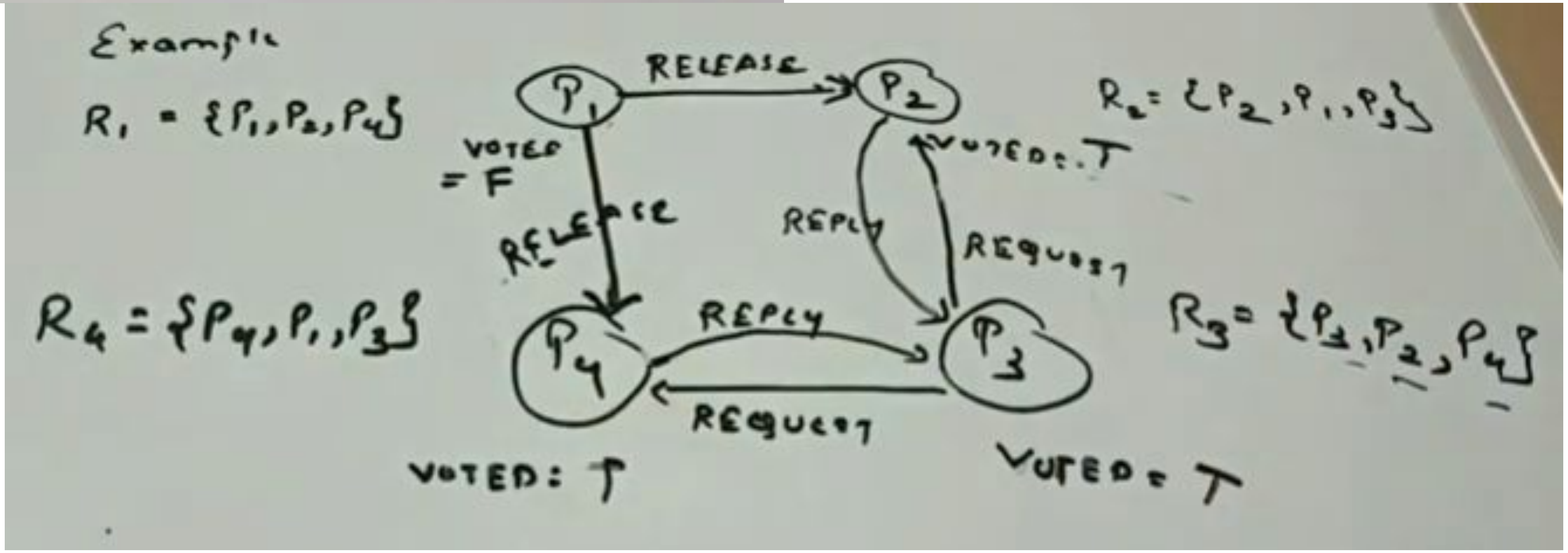

Example
R1 = {P1,P2,P4}
P1
P2
R2= {P2,P1,P3}
P4
P3
R4 = {P4,P1,P3}
R3= {P3,P2,P4}


Example
R1 = {P1,P2,P4}
P1
VOTED:F
P2
VOTED:F
R2= {P2,P1,P3}
R4 = {P4,P1,P3}
P4
P3
R3= {P3,P2,P4}
VOTED:F
VOTED:F


Example
R1 = {P1, P2, P4}
P1
REQUEST
P2
R2 = {P2, P1, P3}
VOTED=F
REPLY
VOTED=TRUE
REQUEST
REPLY
R4 = {P4, P1, P3}
P4
P3
R3 = {P3, P2, P4}
VOTED: TRUE
VOTED=F


Example
R1 = {P1, P2, P4}
P1
REQUEST
P2
R2 = {P2, P1, P3}
VOTED=F
REPLY
VOTED=TRUE
REQUEST
REPLY
REQUEST
R4 = {P4, P1, P3}
P4
P3
R3 = {P3, P2, P4}
REQUEST
VOTED: TRUE
VOTED=F
RP4 = P3


Example
R1 = {P1, P2, P4}
P1
RELEASE
P2
R2 = {P2, P1, P3}
VOTED = F
VOTED = FALSE
RD2 = P3
RELEASE
REQUEST
R4 = {P4, P1, P3}
P4
P3
R3 = {P3, P2, P4}
REQUEST
VOTED : FALSE
VOTED = F
RD4 = P3


Example
R1 = {P1, P2, P4}
P1
RELEASE
P2
R2 = {P2, P1, P3}
VOTED = F
VOTED = T
RELEASE
REPLY
REQUEST
R4 = {P4, P1, P3}
P4
REPLY
P3
R3 = {P3, P2, P4}
REQUEST
VOTED : T
VOTED = T

Thank you