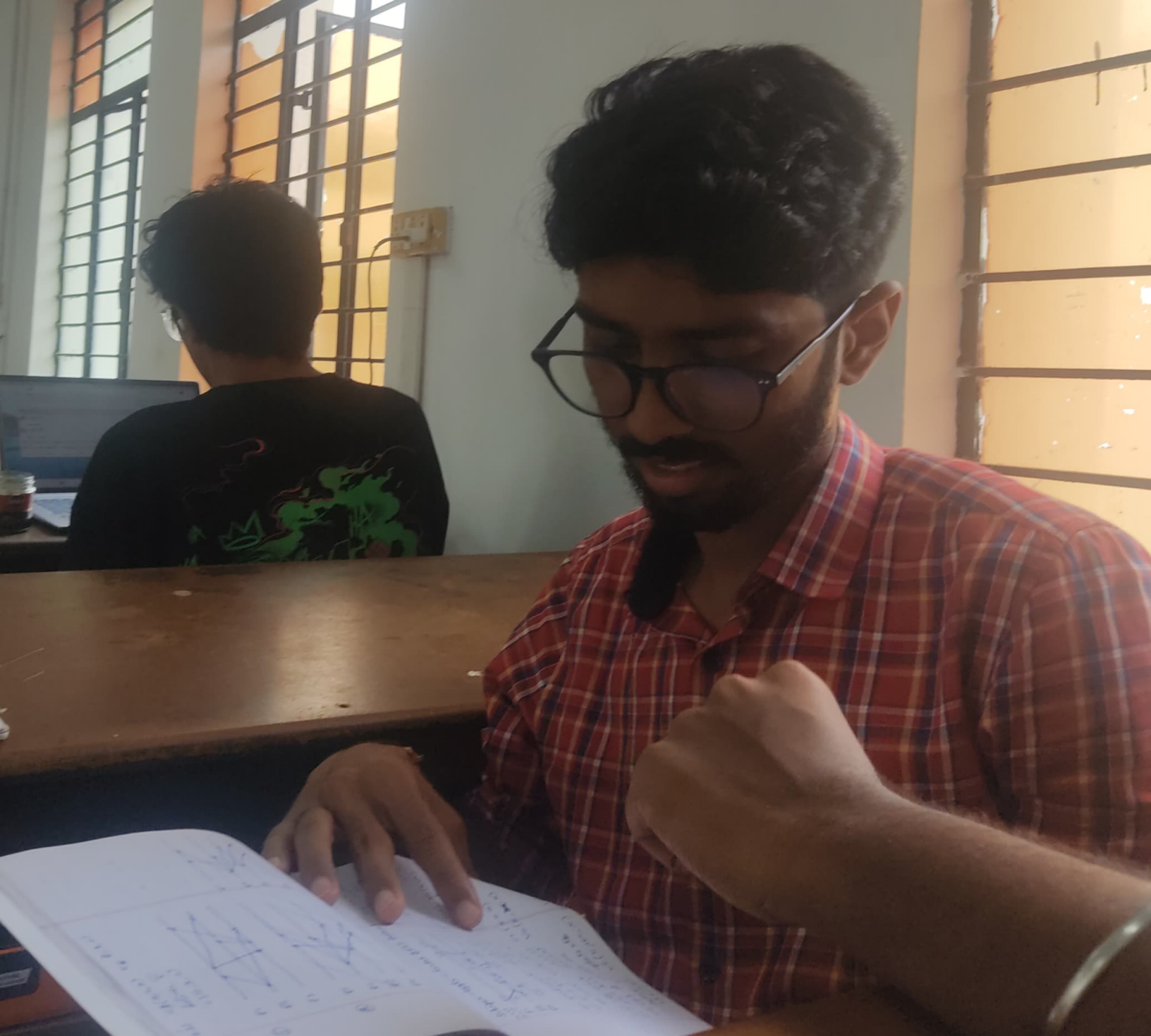
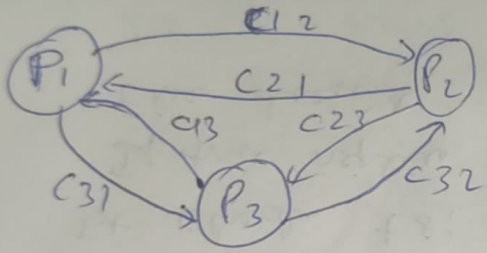


8-9-23 Chandy Lamport's Global Snapshot Recording Protocol



Marker msg : sent to separate msgs - yet to be recorded from already recorded msgs.

Marker send

- Checks its state
- Record its state & send marker msg to all outgoing channels

Marker Rcv.

- Second msg comes in all the rcv channels then stop
- Until then keep recording

Distributed Mutex Algos → Non Token • Lamport • Ricart
→ Token - based • Suzuki
→ Quorum - based • Maekawa

Lamport's Algo

- Every process has a req-queue
- When S_i wants to enter critical section (C-S), it broadcasts REQUEST(ts_i, i) to all other sites and places the request on req-queue$_i$ # (ts_i, i) is the timestamp
- When S_j receives REQUEST(ts_i, i), it is placed in req-queue$_j$ & returns REPLY (with timestamp) to S_i

If S_j also gonna send req. & it rcv. S_i req.
It checks $ts_i < ts_j$
If true replies to S_i
Else doesn't

- S_i enters C-S if both:
  1) S_i rcvd. msgs with ts. larger than (ts_i, i) from all other sites
  2) S_i's req. is at top of req. queue$_i$
- After exiting C-S, S_i removes req. at top of its req-queue, and broadcasts timestamped RELEASE to all other sites
- When S_j rcvs. the RELEASE, it removes S_i's req. at front of queue

① $P_3 \rightarrow P_1 \;||\; P_3 \rightarrow P_2$

P_1 | P_2 | P_3

- Req(1, P3, P1)
- Rep(3, P1, P3)
- Req(1, P3, P2)
- Rep(3, P2, P3)
- Req(1, P3, {P1, P2})
- Rep(3, P1, P3)
- Rep(3, P2, P3)

CS

- Rel(7, P3, P2)
- Rel(7, P3, P1)
- Rel(7, P3, {P1, P2})

- Req(10, P1, {P2, P3})
- Req(10, P3, P1)
- Req(10, P1, P2)
- Req(10, P3, P2)
- Rep(11, P2, {P1, P3})
- Rep(11, P2, P1)
- Rep(12, P3, P1)
- Req(10, P3, {P2, P3})
- Req(10, P1, P3)
- Rep(11, P2, P3)
- Rep(12, P3, P1)

CS

- Rel(15, P1, P3)
- Rep(16, P1, P3)
- Rel(15, P1, P2)
- Rel(15, P1, P3)
- Rep(16, P1, P3)

CS

- Rel(17, P3, {P1, P2})

CSEB-82

① P1 ⟶ P3 || P2 || P1 ⟶ P2 || P1

P1	P2	P3
• Req(1, P1, {P2, P3})		
	• Req (1, P1, P2)	
		• Req (1, P1, P3)
	• Rep (3, P2, P1)	
		• Rep(3, P3, P1)
• Rep(3, P2, P1)		
• Rep(3, P3, P1)		
CS		
• Rel (4, P1, {P3, P3})		
	• Rel (4, P1, P2)	
		• Rel(4, P1, P3)
		• Req (7, P3, {P1, P2})
	• Req (7, P2, {P1, P3})	
• Req(7, P1, {P2, P3})		
	• Req(7, P3, P2)	
• Req(7, P3, P1)		
• Req(7, P2, P1)		
		• Req (7, P2, P3)
		• Req (7, P1, P3)
	• Req (7, P1, P2)	
	• Rep (8, P2, P3)	
• Rep(8, P1, P3)		
		• Rep(8, P2, P3)
		• Rep(8, P1, P3)
		CS
		• Rel (10, P3, {P1, P2})
	• Rel (10, P3, P2)	
• Rel (10, P3, P1)		

- Rep(11, P1, P2)

- Rep(11, P3, P2)

- Rep(11, P1, P2)
- Rep(11, P3, P2)

CS

- Rel (13, P2, {P1, P3})

- Rel (13, P2, P3)

- Rel (13, P2, P1)

- Rep (14, P2, P1)

- Rep (14, P3 P1)

- Rep (14, P2, P1)
- Rep (14, P3, P1)

CS

- Rel (16, P1, {P2, P3})

- Rel (16, P1, P2)

- Rel (16, P1, P3)

- Req (17, P2, {P1, P3})

- Req (17, P1, {P2, P3})

- Req (17, P2, P3)

- Req (17, P1, P2)

- Req (17, P2, P1)

- Req (17, P1, P3)

- Rep (18, P1, P2)

- Rep (18, P3, P2)

- Rep(18, P1, P2)
- Rep(18, P3, P2)

CS

- Rel (20, P2, {P1, P3})

- Rel (20, P2, P1)

- Rel (20, P2, P3)
- Rep (21, P3, P1)

- Rep (21, P2, P1)

- Rep(21, P2, P1)
- Rep(21, P3, P1)

CS

- Rel(23, P1, {P2, P3})
  - Rel(23, P1, P2)
    - Rel(23, P1, P3)

② P1 ⟶ P3 || P2 || P1 ⟶ P2 || P1

P1 (000) P2 (000) P3 (000)

- Req(1, P1, {P2, P3})
  - Req(1, P1, P2)
    - Req(1, P1, P3)
  - Rep(3, P2, P1)
    - Rep(3, P3, P1)

- Rep(3, P2, P1)
- Rep(3, P3, P1)

CS

- Req(7, P1, {P2, P3})
  - Req(7, P2, {P1, P3})
    - Req(7, P3, {P2, P1})
  - Req(7, P1, P2)
    - Req(7, P2, P3)
- Req(7, P3, P1)
  - Req(7, P3, P2)
    - Req(7, P1, P3)
- Req(7, P2, P1)
  - Rep(10, P2, P1) (001)
    - Rep(10, P3, {P1, P2}) (000)
  - Rep(10, P3, P2)
- Rep(10, P2, P1)
- Rep(10, P3, P1)

CS

- Rep(14, P1, {P2, P3})
  - Rep(14, P1, P2)
    - CS
    - Rep(18, P2, P3)
  - Rep(14, P1, P3)
    - Rep(18, P2, P3)
    - CS

- Req(22, P1, {P2, P3})
  - Req(22, P2, {P1, P3})
    - Req(22, P1, P2) (0 1 0)
  - Req(22, P1, P3)
  - Req(22, P2, P3)
  - Rep(24, P3, {P1, P2})
    - Rep(24, P2, P1)
      - Rep(24, P2, P1)
      - Rep(24, P3, P1)
        - CS
        - Rep(28, P1, P2) (∞∞)
  - Rep(22, P1, P2)
    - CS

21-9-23 Ricart Agrawala's Mutex Algo

① $P_1 \rightarrow P_1 \| P_2 \| P_3 \rightarrow P_2 \rightarrow P_3$

- Only Req., Rep., no Release message used
- Req. sent to all other process
- Only if rep. rcvd. from all access CS
- Rep. sent if i) no need to access cs
  ii) TS of req. is lower than its own T

P1 (000)

- Req(1, P1, {P2, P3})
- Rep(2, P2, P1)
- Rep(2, P3, P1)

P2 (000)

- Req(1, P1, P2)
- Rep(2, P2, P1)

P3 (000)

- Req(1, P1, P3)
- Rep(2, P3, P1)

CS

- Req(3, P1, {P2, P3})
- Req(3, P2, P1) (010)
- Req(3, P3, P1) (011)
- Rep(4, P2, P1)
- Rep(4, P3, P1)

CS

- Rep(6, P1, P2) (001)
- Rep(6, P1, P3) (000)

- Req(3, P2, {P1, P3})
- Req(3, P1, P2)
- Req(3, P3, P2)
- Rep(4, P2, P1) (001)

- Req(3, P3, {P1, P2})
- Req(3, P1, P3)
- Req(3, P2, P3)
- Rep(4, P3, P1) (010)

~~Rep(5, P3, P2)~~ (000)

- Rep(7, P3, P2) (000)

21-9-23 Ricart Agrawala's Dmutex Algo

(1) $P_1 \rightarrow P_1 \| P_2 \| P_3 \rightarrow P_2 \rightarrow P_3$

- Only Req., Rep., no Release message used
- Req. sent to all other process
- Only if rep. rcvd. from all access CS
- Rep. sent if i) no need to access CS
  ii) TS of req. is lower than its own TS

P1 (000)

- Req(1, P1, P2, P3)
- Rep(2, P2, P1)
- Rep(2, P3, P1)

P2 (000)

- Req(1, P1, P2)
- Rep(2, P2, P1)

P3 (000)

- Req(1, P1, P3)
- Rep(2, P3, P1)

CS

- Req(3, P1, {P2, P3})
- Req(3, P2, P1) (010)
- Req(3, P3, P1) (011)
- Rep(4, P2, P1)
- Rep(4, P3, P1)

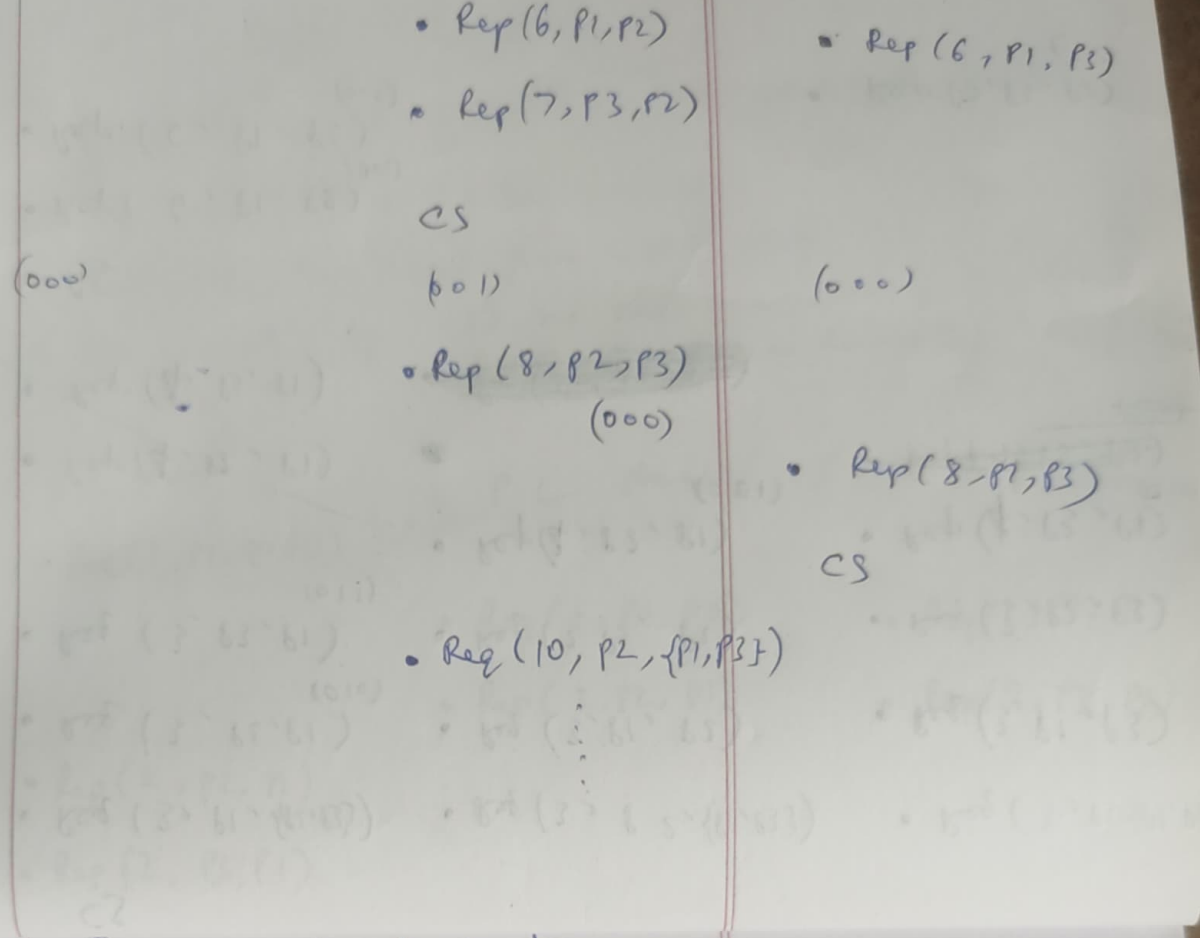
CS

- Rep(6, P1, P2) (001)
- Rep(6, P1, P3) (000)

- Req(3, P2, {P1, P3})
- Req(3, P1, P2)
- Req(3, P3, P2)
- Rep(4, P2, P1) (001)

- Req(3, P3, {P1, P2})
- Req(3, P1, P3)
- Req(3, P2, P3)
- Rep(4, P3, P1) (010)
- ~~Rep(5, P3, P2)~~ (000)

- Rep(7, P3, P2) (000)



Quorum - based Mutex

- Diff from other algos:
  1) Site req. only subset of sites
     Req. sites are chosen such that $\forall i, j : R_i \cap R_j \neq \phi$
  2) Site can send only one reply at a time. The next one can only be sent after a release msg. for prev. reply

- Coterie C is a set of sets where each set $g \in C$ is called Quorum
- Quorums in a coterie satisfy:
  1) Intersection: Every pair of quorums $g, h \in C$ must satisfy $g \cap h \neq \phi$
  2) Minimality: Every pair of quorums $g, h \in C$ must satisfy $g \not\subseteq h$

Algo : Maekawa Algo

- Request
  - Site S_i req. access to CS by sending Req(i) to all sites in R_i
  - Site S_i sends Reply(j) to S_i if no Reply sent since last Release. Else, queue Req(i)
- Execute
  - S_i executes CS only after every site in R_i sends Reply(j)
- Release
  - After CS, S_i sends Release(i) to every site in R_i
  - S_j rcvs. Release(i), then it pops queue & sends Reply to that Req. If empty queue, it waits for next Req.

Deadlock possible

Lowest timestamp ⇒ higher priority

Low priority process rolled back as victim

Inquire , Yield , Grant

for rollback — by send release — release used to resolve deadlock

Deadlock Detection

- Single Resource Model — Process req. only 1 resource at a time
- AND Model — Process req. ≥ 1 resource at a time. Needs all to satisfy
- OR Model — " " " " " Need only one to satisfy
- AND-OR Model — Need 2 A's & 2 B's OR 2 C's & 2 D's

Knapp's classification

Obermarck's Algo (Pathpushing, AND/OR) → • Every site builds local wait-for graph
• This is pushed to adjacent site
• They append changes & push & so on
• Cycle means deadlock
• Memory intensive & Time intensive
• Might detect false tve

Chandy Misra Haas (Edgechasing)

↳ Probe (i, j, k)
initiator, sender, receiver

- Sender sends probe to its dependents (aka edge in wait-for graph)
- Only blocked processes send probe msgs
- Deadlock when k == i
- Msgs sent only to processes that are dependent across sites

(OR model exists CMH Algo)

3-10-23

Knot in OR model

Cycle ⇒ Deadlock in AND model
only
Knot ⇒ Deadlock in OR model

A knot is a node from where all other nodes can be reached

Consensus & Agreement

CAT-2

Unit 1 - Global States, Cuts & types
(Strongly consistent
Consistent
inconsistent)

Unit 2 - Chandy-Lamport Global State recording
how it works, why it is not inconsistent

Unit 3 - Mutex & Deadlock Algos
properties, phantom deadlocks etc.

Non-token based
Token based
Quorum based

avoidance, prevention,
detection
deadlock models
Knapp's classes.

quorum X
Maekawa X