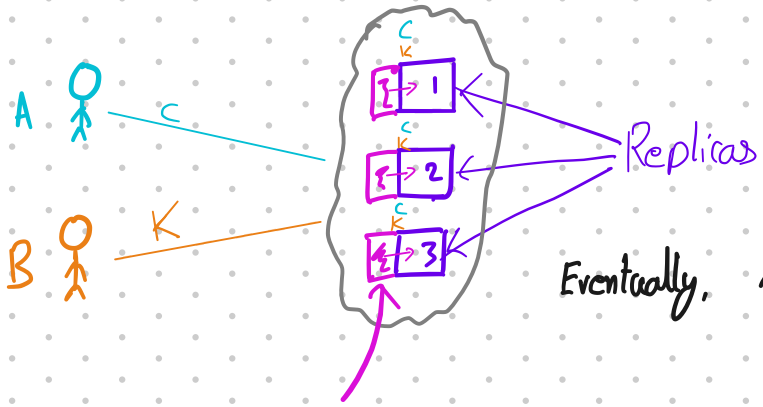
Replicated State Machines } Raft

Last Class:

- RSMs: A recipe for building fault tolerant systems

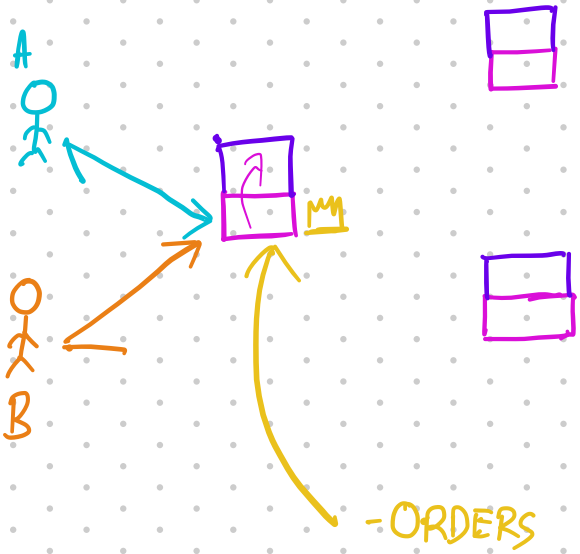


Eventually, state(1) ≡ state(2) ≡ state(3)

Assuming ☐ deterministic

- Correctness requirements
  - Agreement
  - Ordering

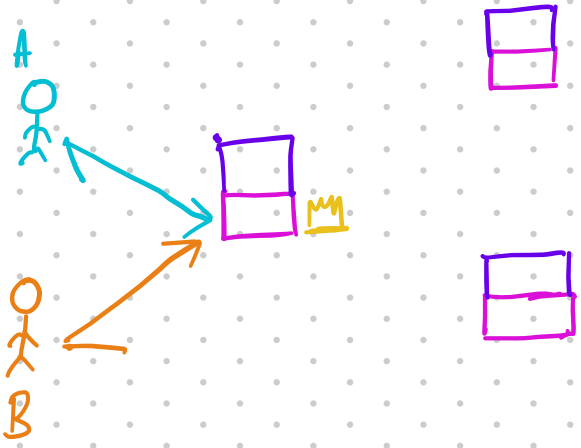
- Leader based RSM



- ORDERS
- REPLICATES
- DECIDES WHEN SAFE TO EXECUTE (DELIVER) COMMAND & RESPOND

CORE CONCERNS

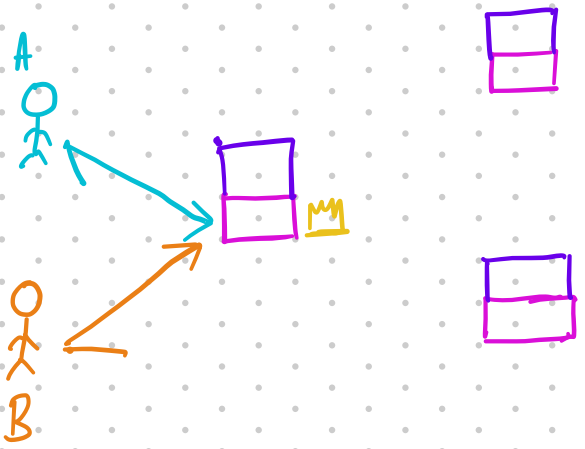
① WHEN IS IT SAFE TO EXECUTE/DELIVER?



REQUIREMENTS:

② HOW TO ORDER & RESPOND TO LEADER

(2) How to Detect & Respond to Failure.



Requirements:

Target Consistency Model:

Practical Concerns We Will Ignore

- Reads Are Expensive

- Crash Recovery

Raft: An RSM Protocol

Failure model: FAIL-STOP ; $< \frac{N}{2}$ PROCESSES CAN FAIL
(Really fail-recover, but needs more)

→ USEFUL FOR FIGURING OUT COMMIT POINT, etc.

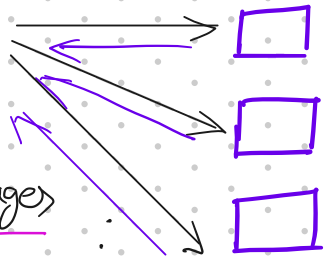
CORE CONCEPT: QUORUM INTERSECTION.

Two operations: store(val) → bool
load() → v

} Goal: If store succeeds any subsequent load returns val [Safety]

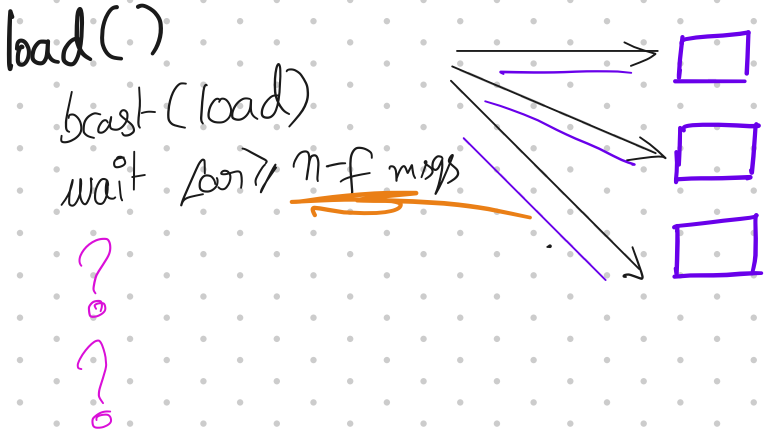
[Note: Not guaranteeing termination
Failure model: at most f fail
≤f fail: Term.
>f fail:
]

Consider n=3 f=1



store(val)
| bcast (store, val)
| wait for ≥ f+1 messages
| if |:done| ≥ f+1
| return true
else
REVISIT N. WEEK

if (cur = ⊥)
cur = val
send(–, :done)
else
send(–, :no)
(cur val)



Basic Idea [n=5, f=2]

store(val)

load()

[n=7, f=1]

store(val)

load()

Observe: $f < \frac{n}{2}$ & n-odd $\Rightarrow$ $f+1 = n-f$

$f = \frac{n-1}{2}$

$f+1 = \frac{n+1}{2}$

$n-f = \frac{n+1}{2}$

How does this all relate to Raft?

Normal operation
- Leader working as expected, etc.

} Append Entry to add command:
↳ Leader: store(log idx, cmd)

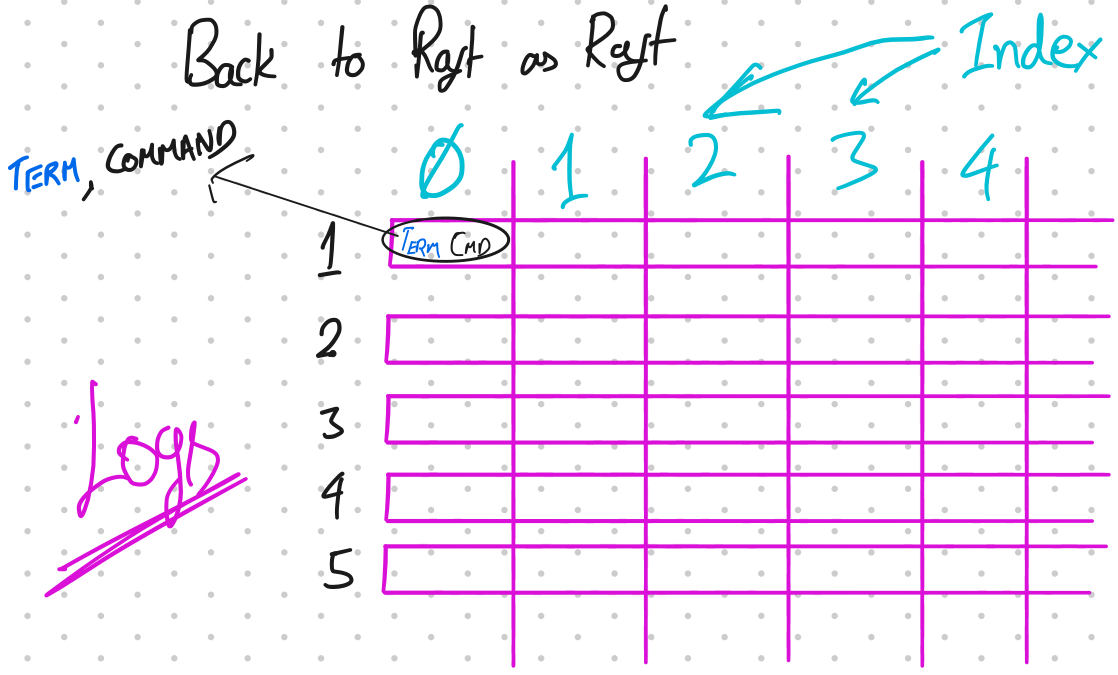
Leader Election (Request-Vote to become leader

Leader [illegible]

↳ Candidate: store (term, node ID)

↳ Leader is value in last stored term

Back to Raft as Raft



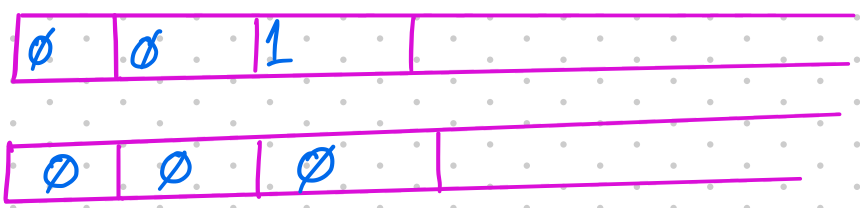
State @ each node

- Log
- currentTerm
- voted For
- type
  - Follower
  - Candidate
  - Leader
- commit index
- last applied

(Figure 2 in the paper is your friend)

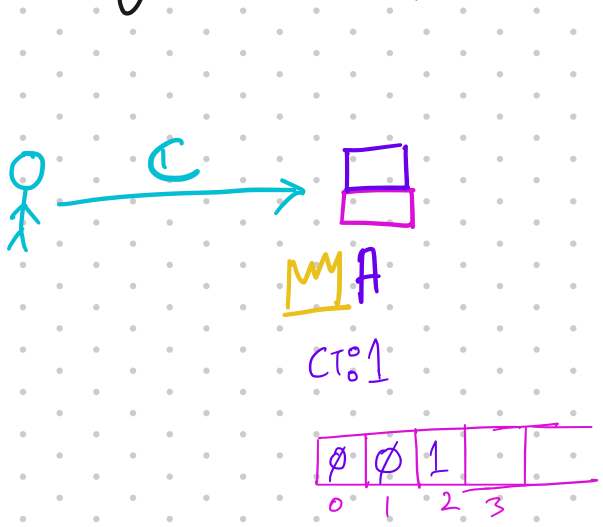
Invariants

- currentTerm is increasing on each node.
- For any term at most 1 process/node is LEADER.
- Leader (for the current term*) has THE authoritative log.
- term, log index maps to a unique command

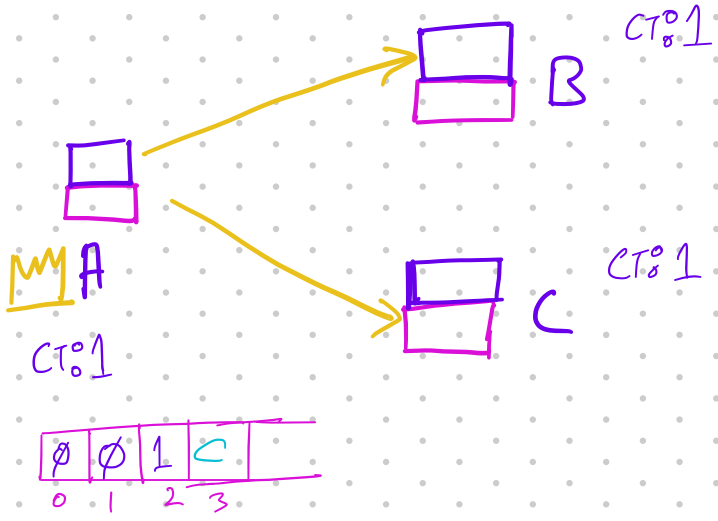


- COMMIT POINT:

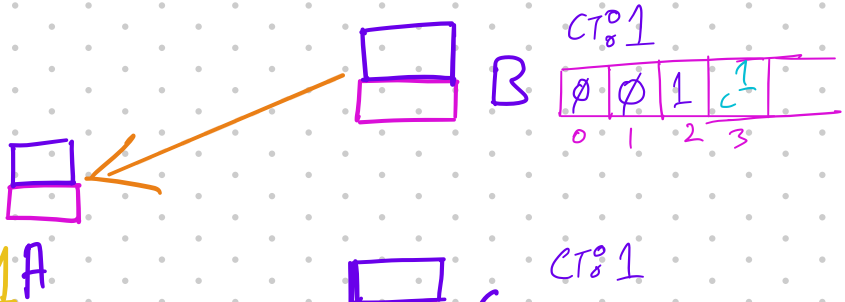
Walk through the basic protocol — see how invariants are preserved.



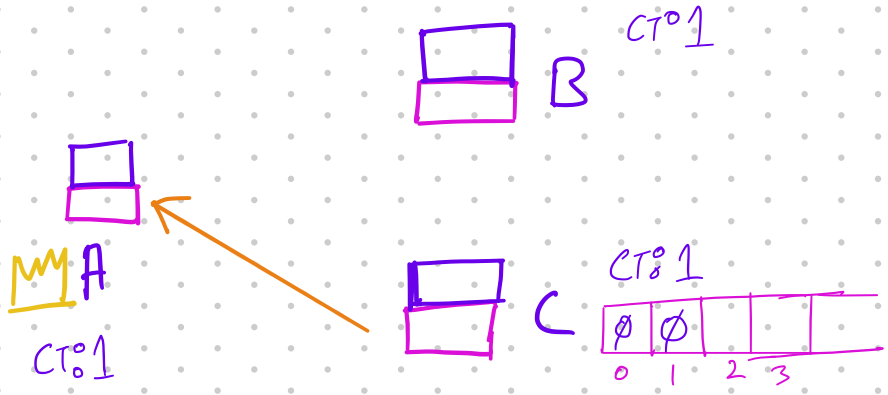
AppendEntry (index=3, c, term=1, lastLogIndex=2, lastLogTerm=1)



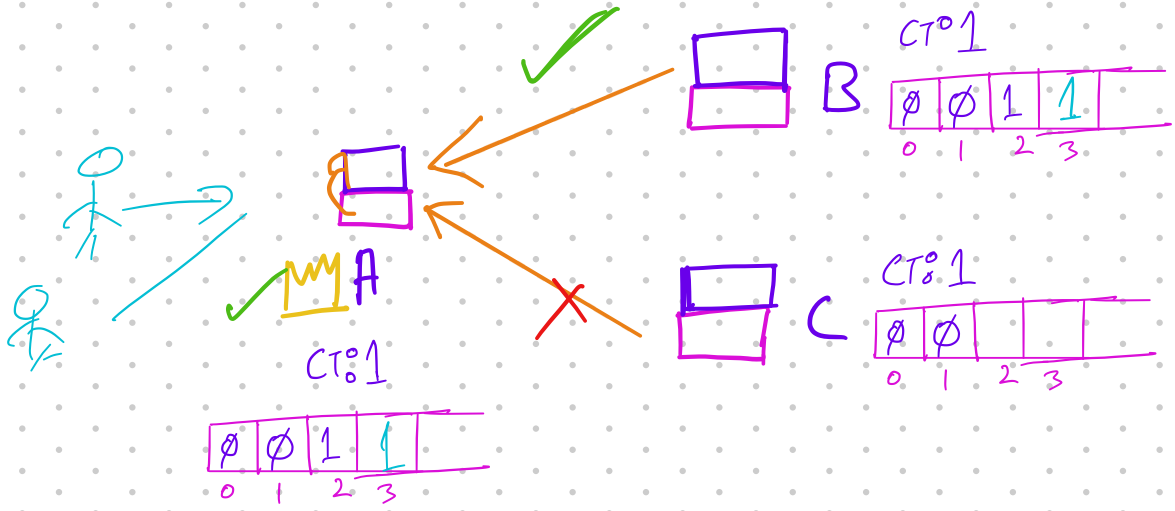
AppendEntry (index=3, c, term=1, lastLogIndex=2, lastLogTerm=1)



AppendEntry (index=3, c, term=1, lastLogIndex=2, lastLogTerm=1)

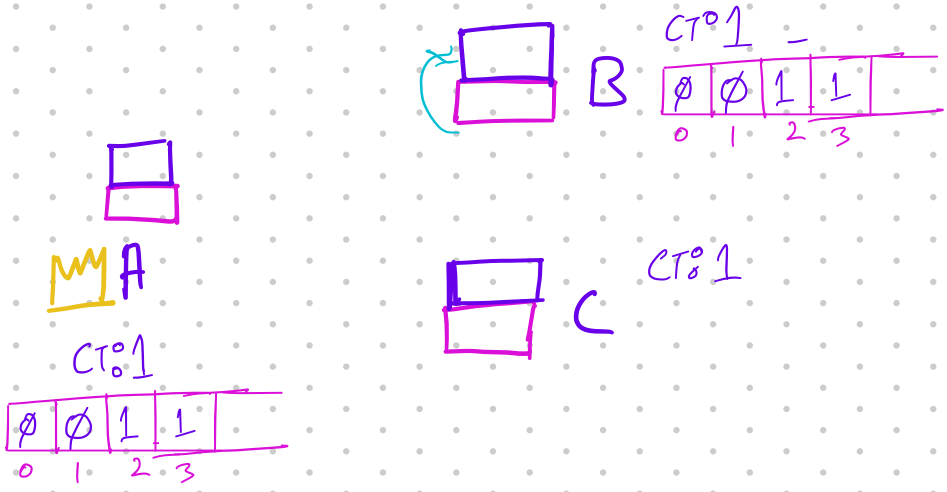


AppendEntry (index=3, c, term=1, lastLogIndex=2, lastLogTerm=1)

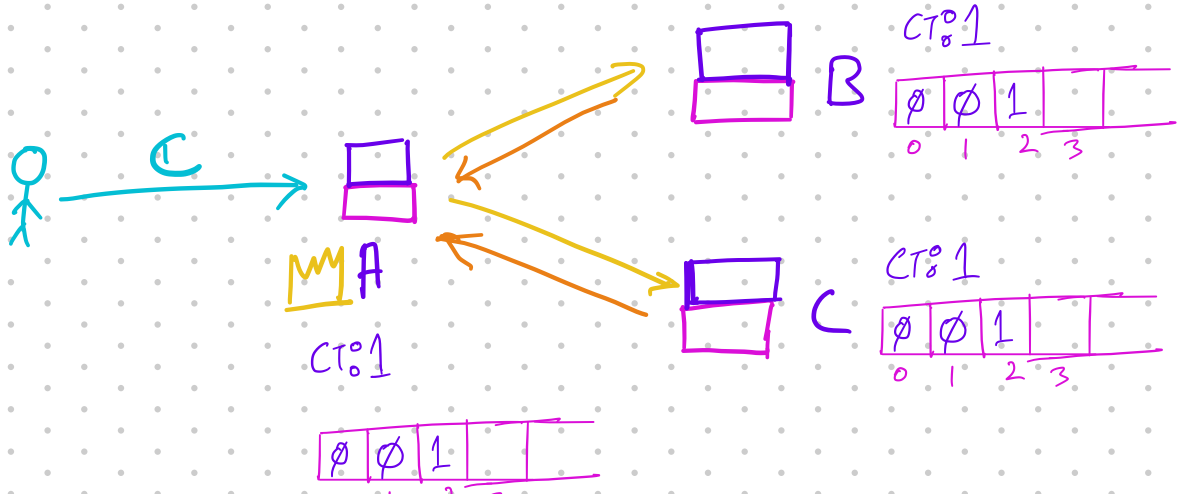


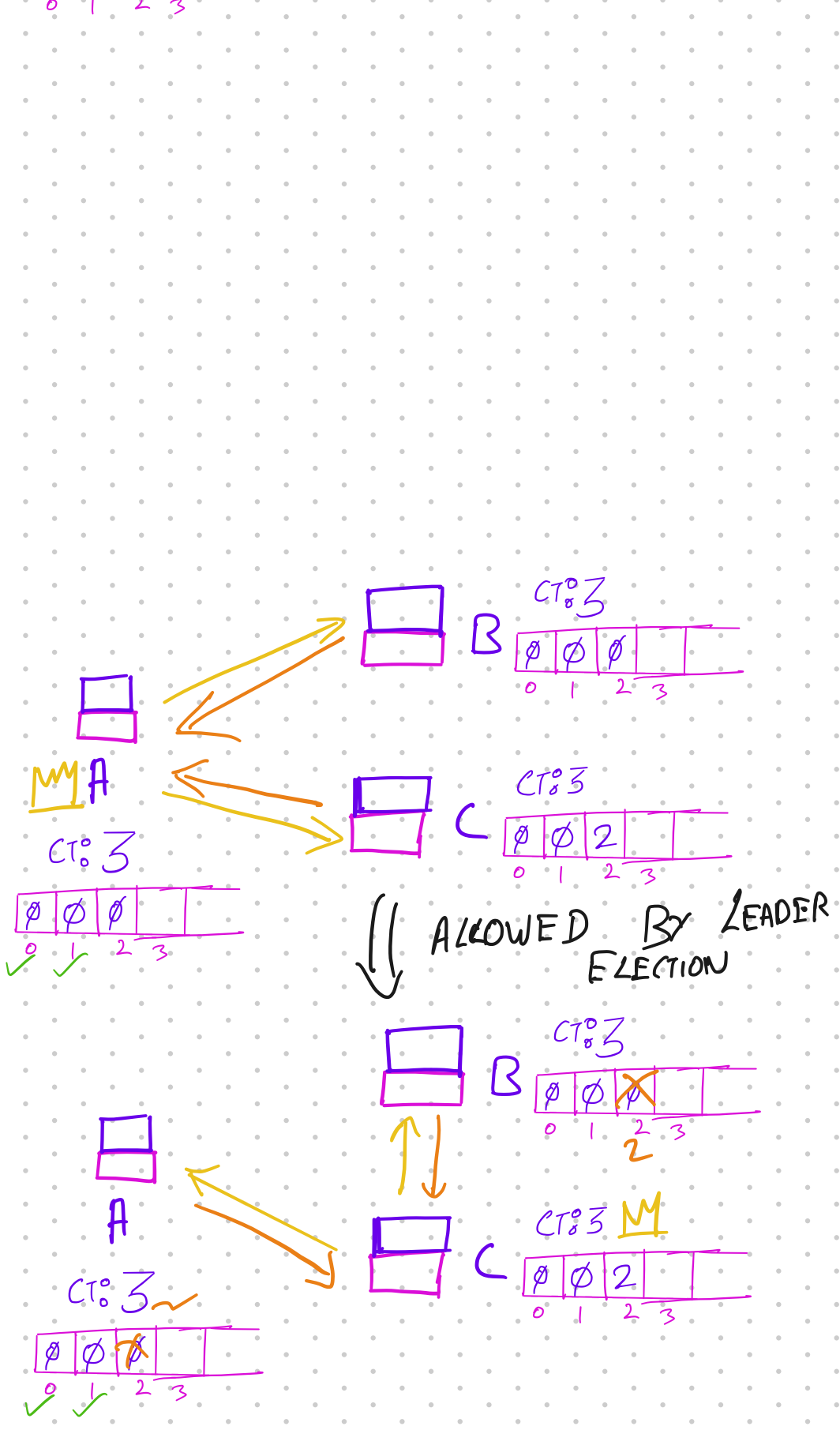
What happens with C?

When does B deliver the command?



Commit Points





- Leader (for the current term*) has THE authoritative log.

When Is An Entry Committed?

Leader Election

Invariant - For any term at most 1 process/node is Leader.

① Trigger election.

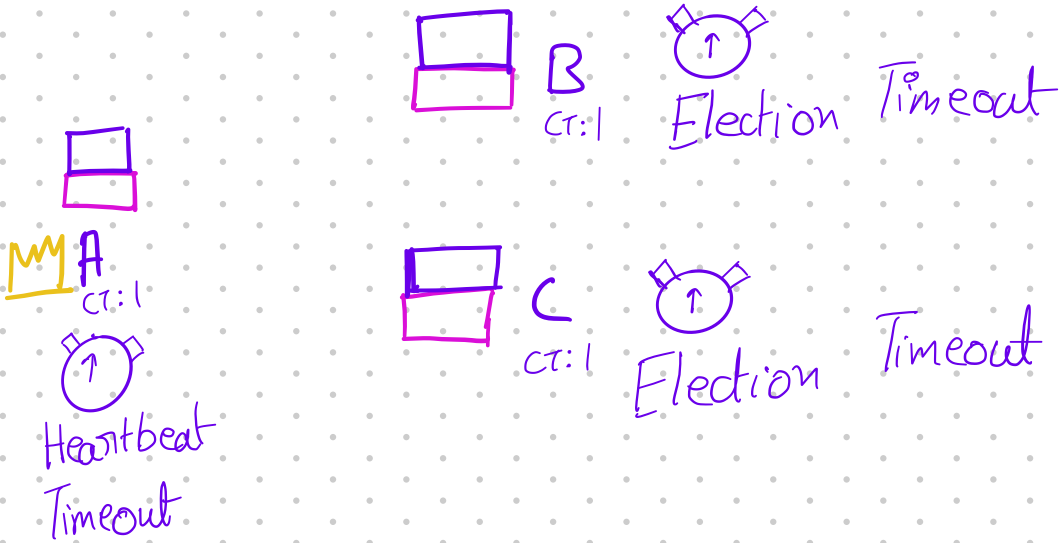
Desirable Avoid leader election when possible (until leader fails) } Async model ⇒ Impossible

Why? Better performance

Instead tradeoff

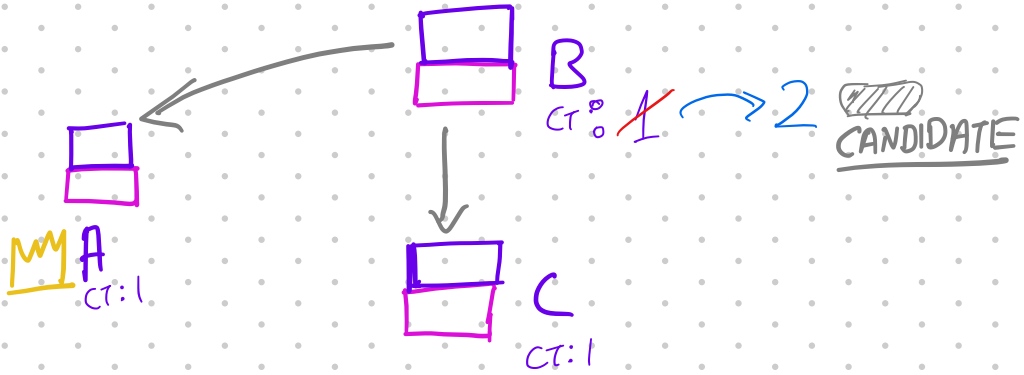
Freq. of unnecessary election

Time to recover from leader failure.



HB timeout < Election timeout ?

② When election timeout expires at node B



RequestVote (ID = B, term = 2,)

INV - Leader (for the current term*) has THE authoritative log.

⇒ Any node elected in term 2 must have all committed entries.

Two paths
→ Only elect leaders with all committed entries [Raft]
→ Fix up log after election [Next week, etc.]

PROBLEM: ONLY THE LEADER KNOWS WHAT ENTRIES ARE COMMITTED

Solution: Quorum Intersection

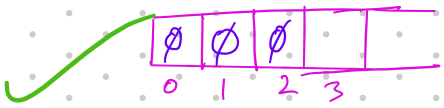
↳ Any committed entry was replicated to a quorum. [n/2 + 1 nodes]

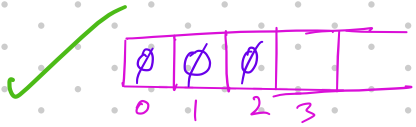
⇒ Known to at least one process in any other quorum.

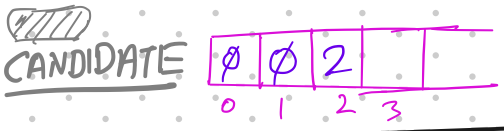
- Candidate needs vote from quorum to become leader.

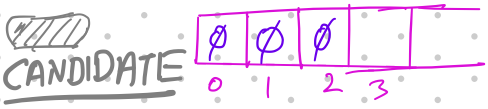
- Nodes vote for a candidate only if
  - (a) Candidate has more entries in its log
  - OR (b) Candidate has the same number of entries but candidate's last entry term ≥ node's last entry term.

How recently was log updated





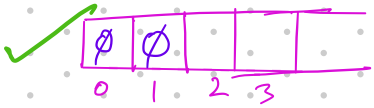




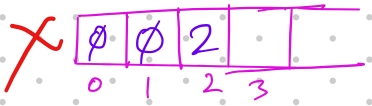
000

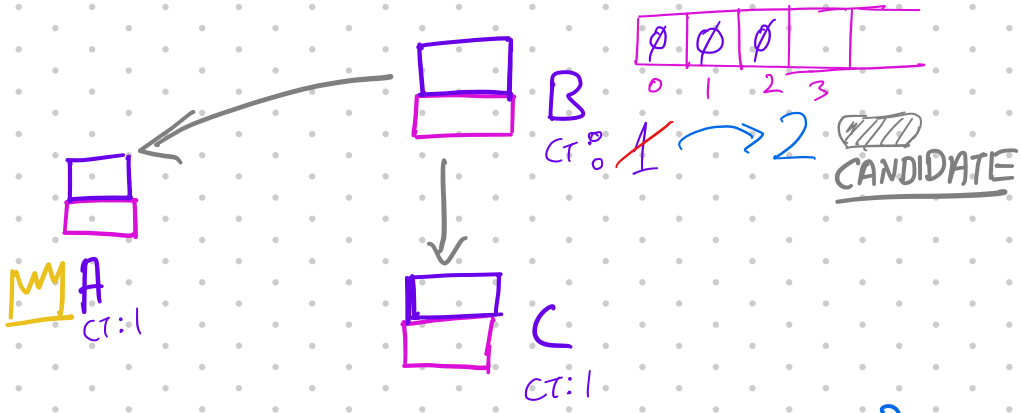
0 0

0 0 0



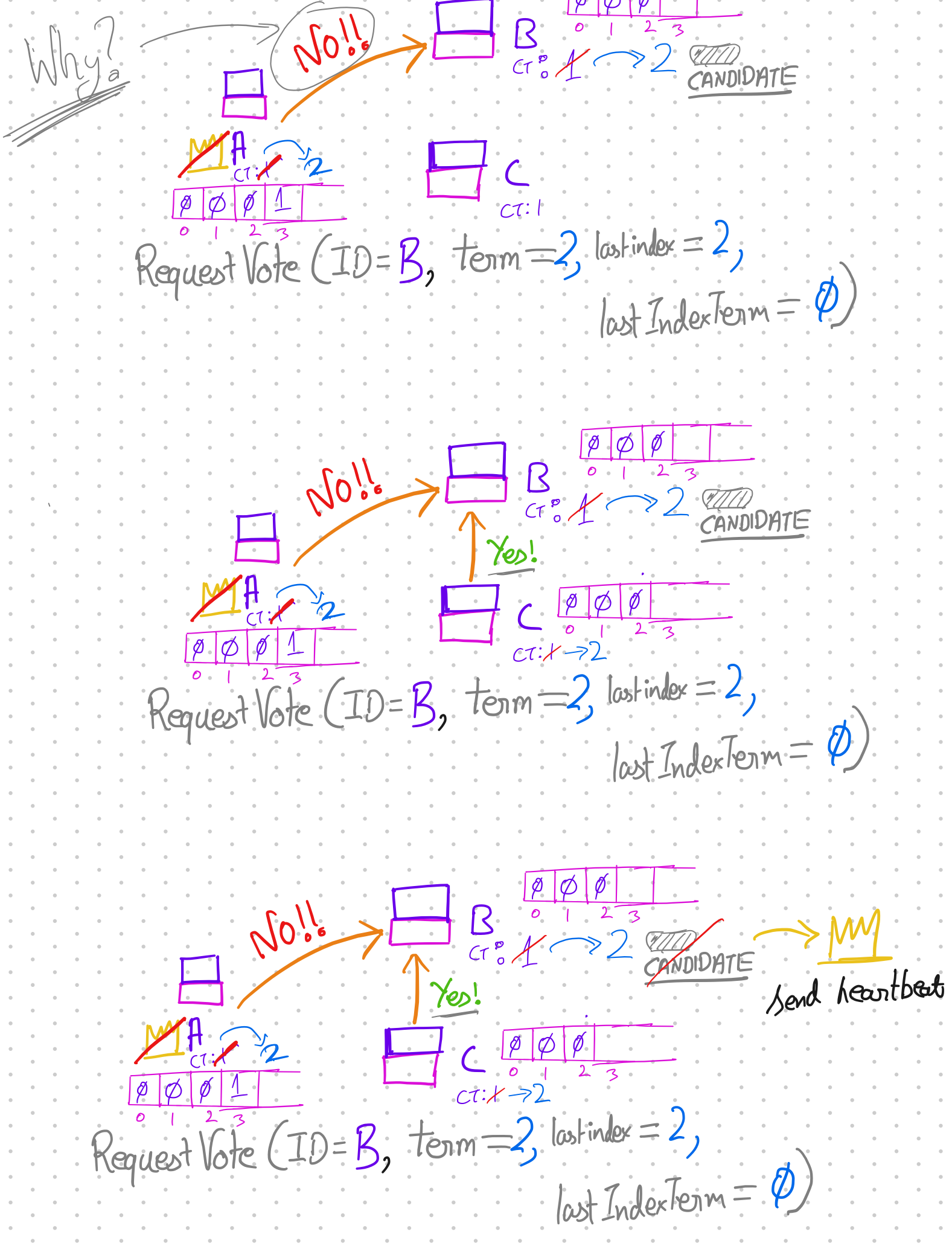
02



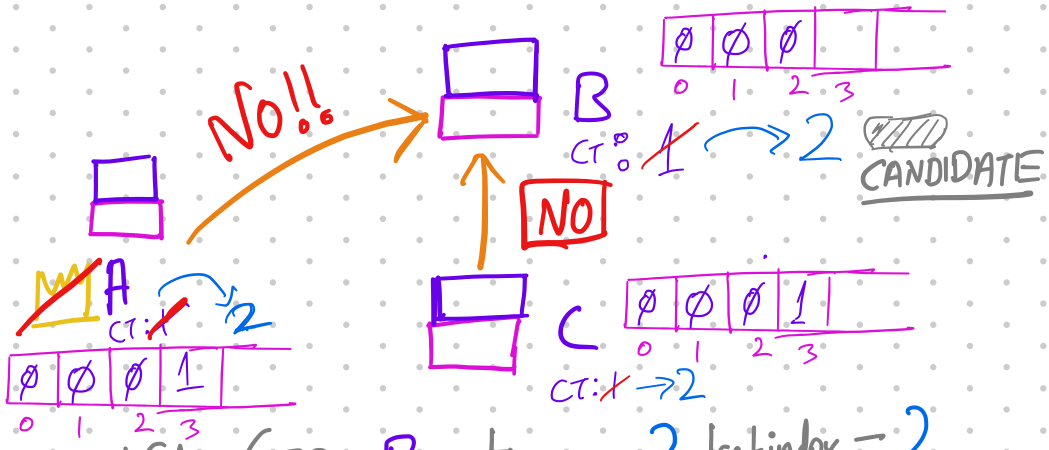


RequestVote (ID=B, term=2, lastIndex=2,

lastIndexTerm=ϕ)


Why?
No!!
B
CT: 1 → 2
CANDIDATE
A
CT: X → 2
C
CT: 1
Request Vote (ID=B, term=2, lastIndex=2, lastIndexTerm=∅)
No!!
Yes!
Request Vote (ID=B, term=2, lastIndex=2, lastIndexTerm=∅)
Send heartbeat
Request Vote (ID=B, term=2, lastIndex=2, lastIndexTerm=∅)

Alt History

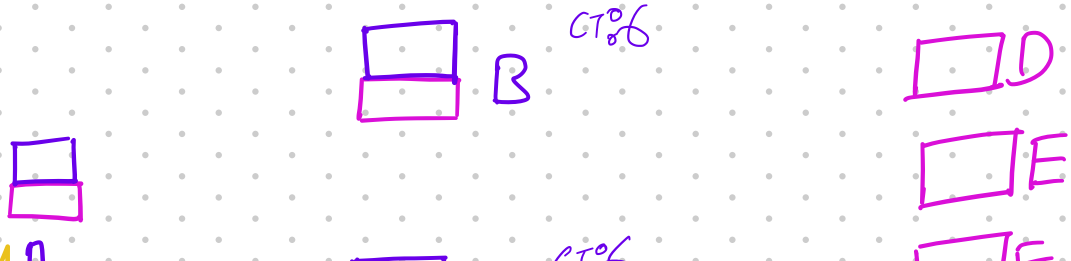


RequestVote (ID=B, term=2, lastIndex=2, lastIndexTerm=∅)

What now???

Joint quorums & reconfig.

Problem

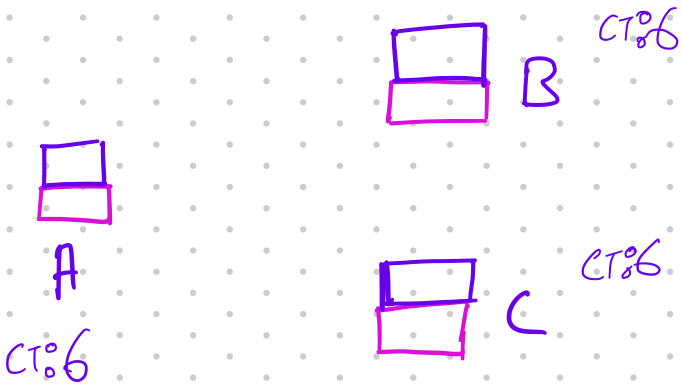


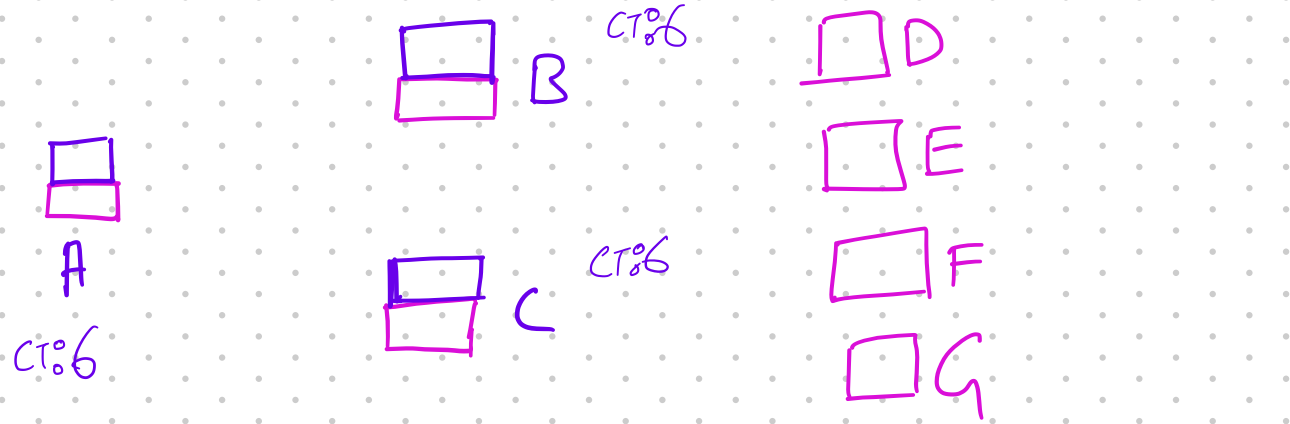
$N/2 + 1 = 4$: D, E, F & G form a quorum

But - do not have current log
(or any other state)

How to have them safely join

OLD + NEW : Operations on two Raft instances!






M002
[]
AER(fd:= 4,
lli =3,
llt= 2)