CSCI 1380 Day 14: Raft

Last Class

Passive V. Active Replication

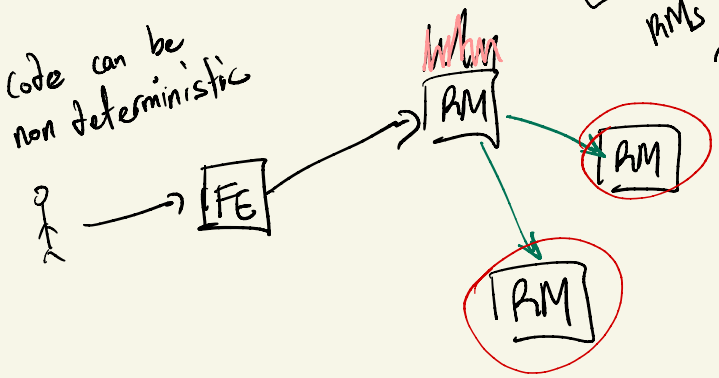
Today

* Picking btw Passive/Active Replication
* Raft (Passive)
  * node information
  * leader election (safety/liveness)
  * client protocol
  * Raft safety (committed data persists)
    * Changes to leader election / log commitment

Passive

Leader/follower — RMs have different roles

code can be non deterministic

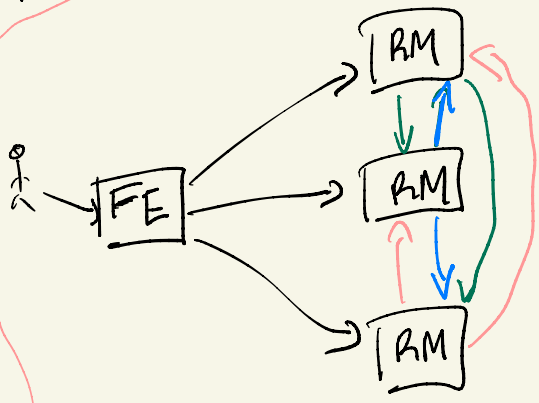


key issues: under failure it takes time to find new leaders & commit

Passive ⟹ data centers/clusters

network latency ⟹ 4ms

Active — all RMs are equal



requires code to be deterministic

key issues: lots of messages & high overhead

Wide area network (Internet)

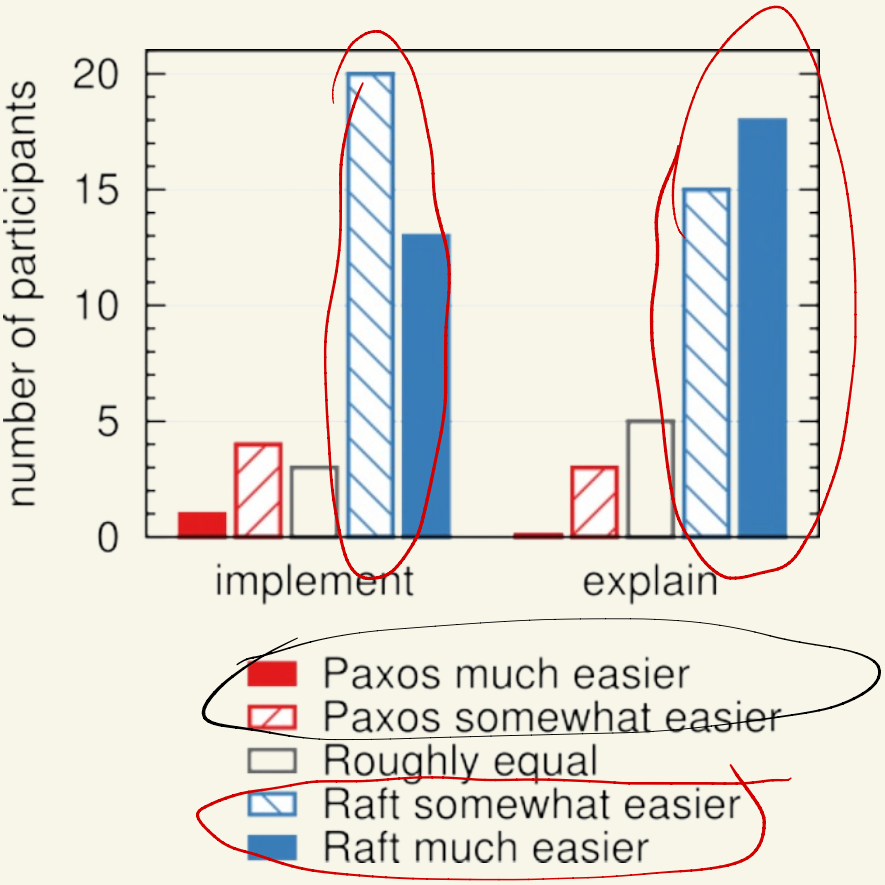
network latency ⟹ 40 – 100ms

Geo distributed database/storage will use active replication

Can you tolerate the latency overhead of leader failures / leader election?

"The dirty little secret of the NSDI community is that at most five people really, truly understand every part of Paxos ;-)." – NSDI reviewer

"There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system…the final system will be based on an unproven protocol." – Chubby authors



Raft

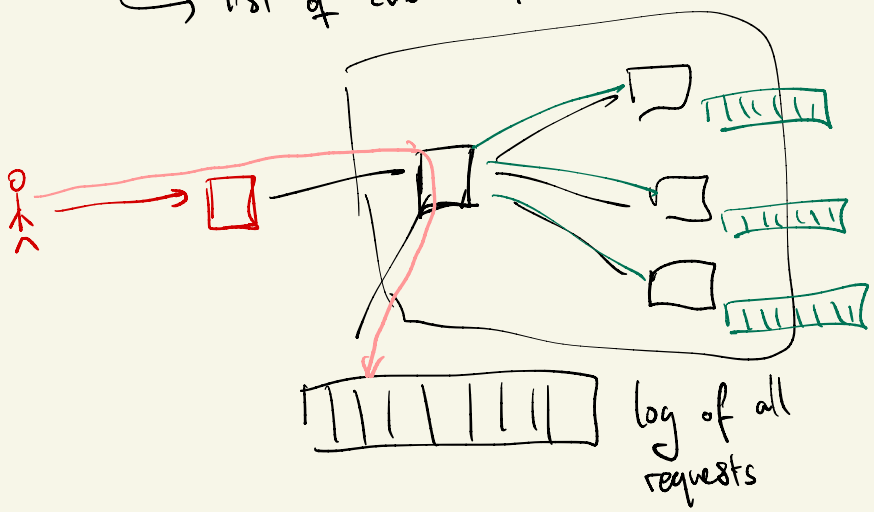
Consensus = all nodes in the algorithm agree on some value

- all nodes → quorum (majority)
- agree on some value → Chandy Lamport "agreement on what is in a snapshot"
- some value → "log"

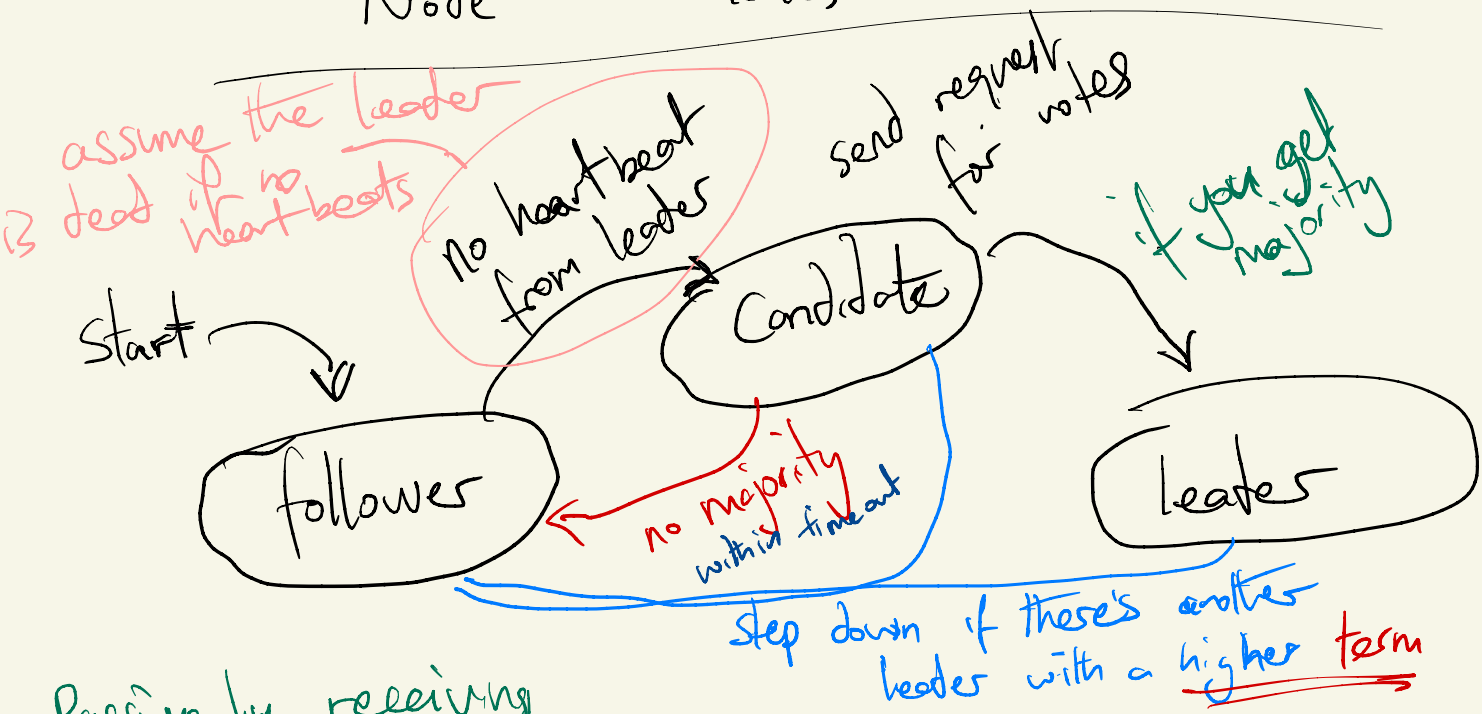
"Raft cluster" ⇒ N nodes that participate in protocole
- → 1 leader
- → N-1 followers

"Node" ⇒ log + statemachine + raft protocol code

log → list of client requests



Node Roles

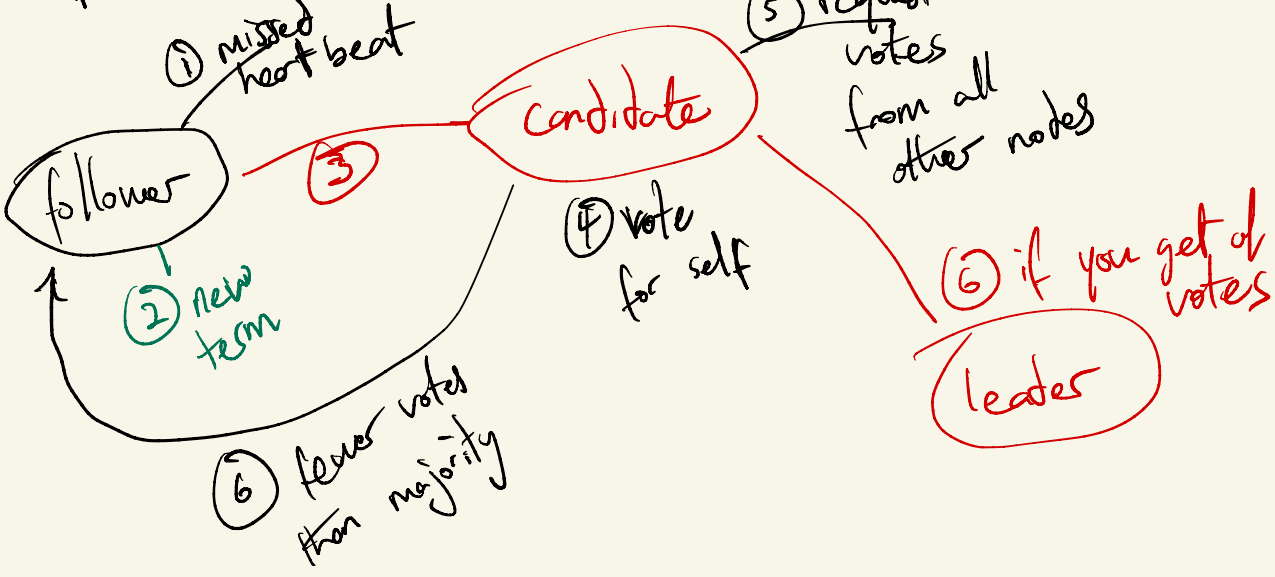


Passively receiving request from leader & putting in logs

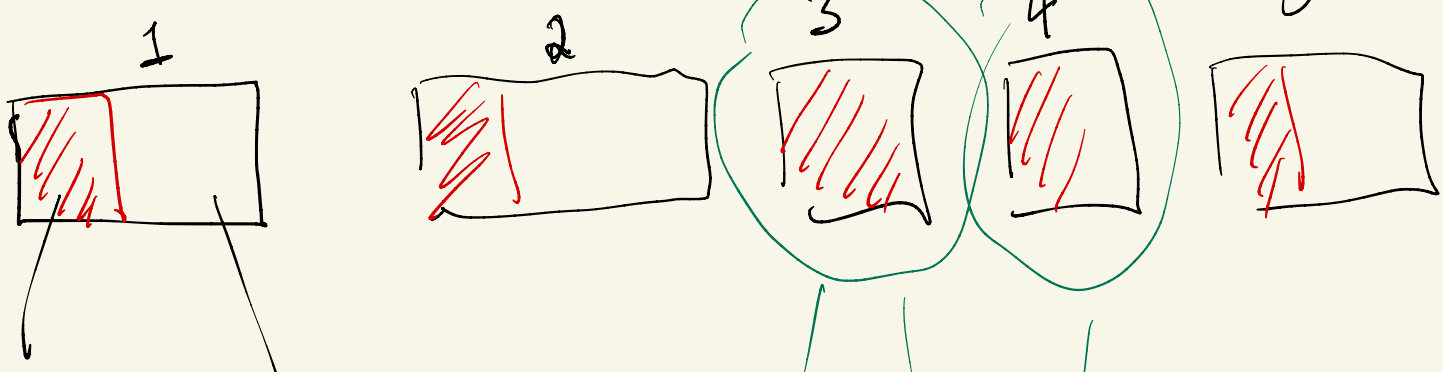
Leader Elections

① N → # of nodes is a predefined value

Because N is well defined every node can calculate majority



Terms



elections:
candidate is waiting

normal operations:
candidate has become leader & can service request from clients

Split vote so no leader elected

Term with no leader elected

---

cluster (N=3)

node A was leader & failed

candidate


candidate


if Node B & C detect failure at same time then they both become candidates

Majority = 2 votes

How to ensure a leader (liveness)

(1) random delay into state transition (follower → candidate)!

(2) different weight (How does this impact partitions)

→ ordering to break ties

Raft

(3) make each follower use different T/O

Add randomness to avoid multiple nodes becoming candidates at same time!!

each node randomly set Heartbeat TimeOut ⟹ [T ↔ 2T]

T: time to send msgs to everyone (broadcast)

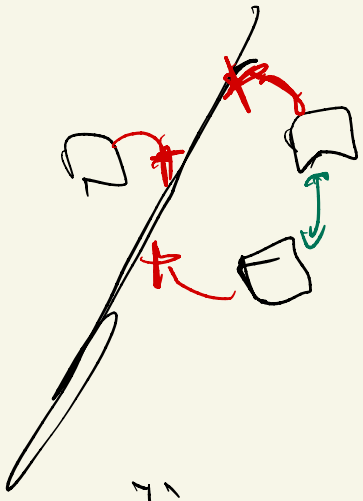
Raft Election Properties

(1) liveness ⇒ eventually you will have a leader
↳ random timeouts

(2) Safety ⇒ at most one viable leader
↳ each node only votes once
+ only the candidate with majority votes wins

viable = process client requests

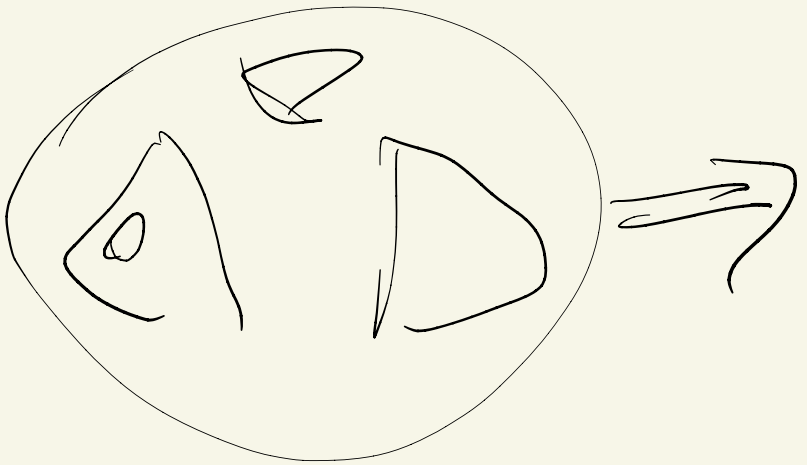
when there are partitions ⇒



these nodes can communicate with each other but not across partitions

n/w partition

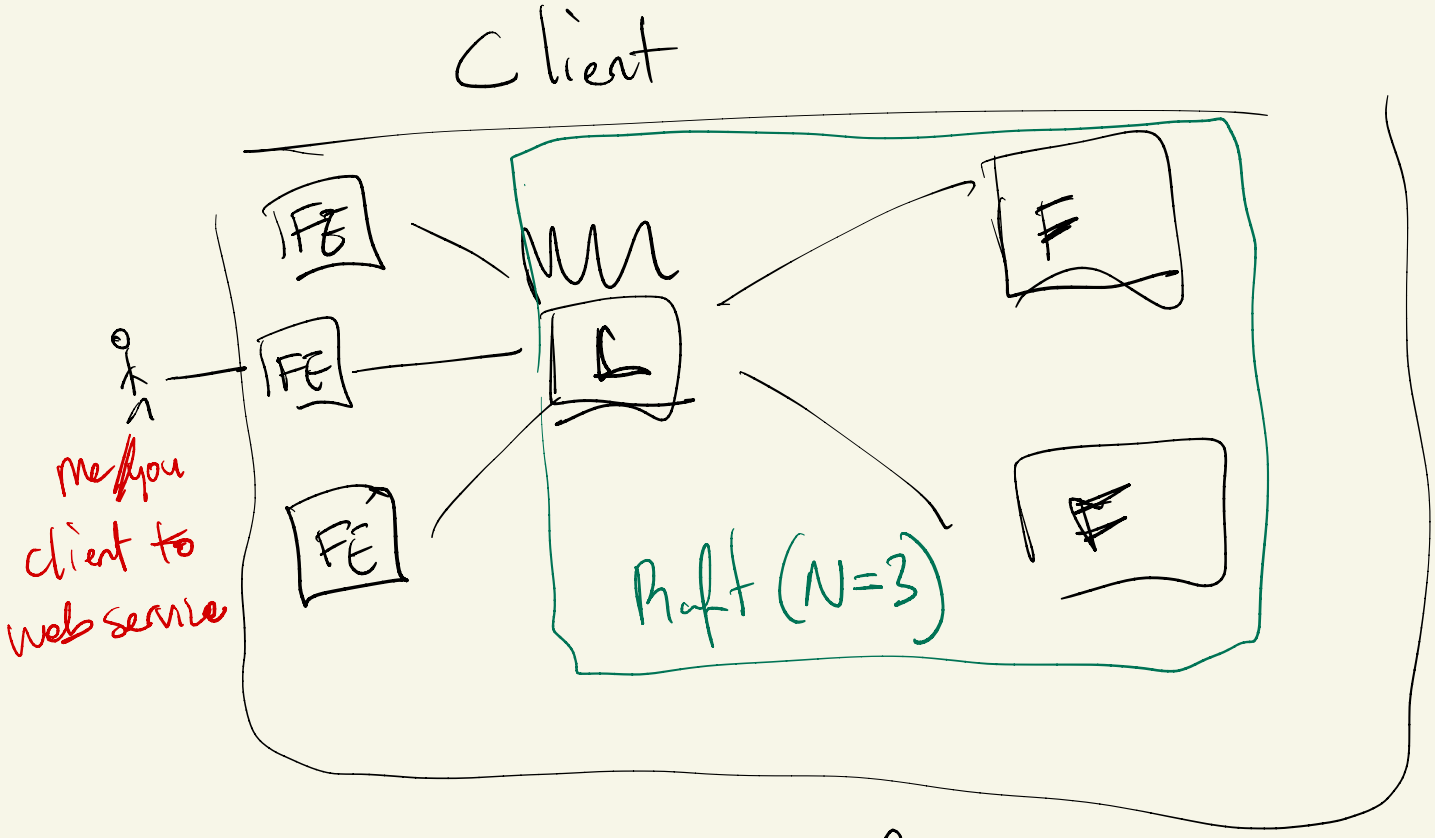
How many failure can happen in your cluster ??



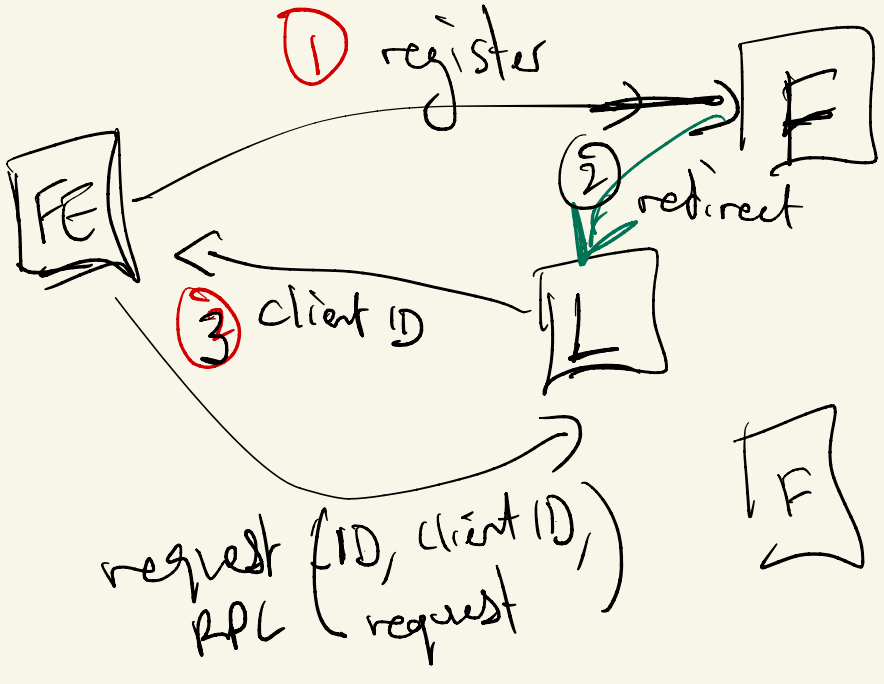
at most 3 servers can fail in cluster

$$N = 2F + 1$$

max # of failures in your cluster



Raft's clients are the front end servers



(1) FE picks random Raft node

(2) if random Node is not leader then redirect

(3) register with leader to get client ID

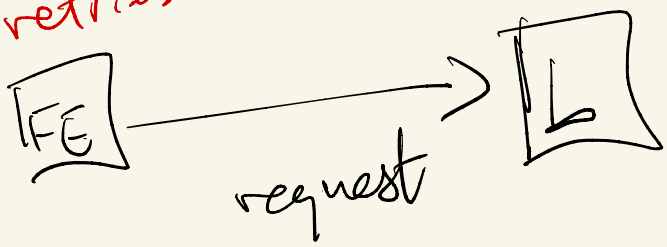
(4) all requests include client ID + request ID

client side

request retries

leader side

duplicate suppression & response replay



(1) duplicate suppression
↳ checking log for request ID

(2) responds with a cached response (response replay)

which semantics requires the three techniques?

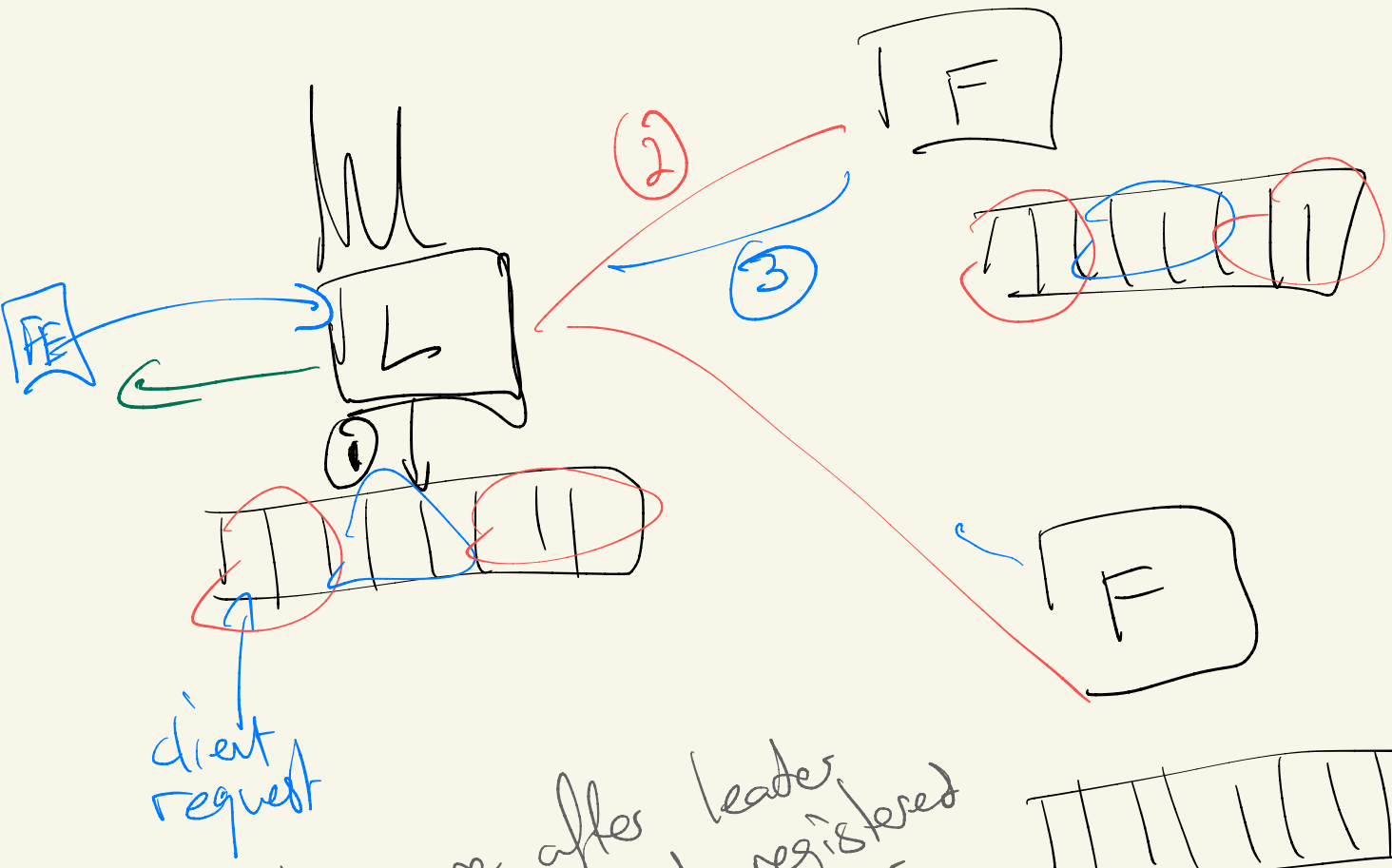
at least once	at most once	exactly once
	✓	✓
	✓	

not practical

Raft claims:

exactly once as long as client doesn't fail

Log Commitment



These steps are after leader is found & you already registered with the leader

1. get request (make sure new)
2. put in local log
3. replicate followers
4. after a majority of nodes have update logs
5. then entry is considered committed
6. respond to client

Raft

1. leader election (liveness / safety)
2. client interactions
3. log (log commit)
4. Node (states / transition / data structures)