Traditional distributed systems

- Paxos
- raft

trusted environment

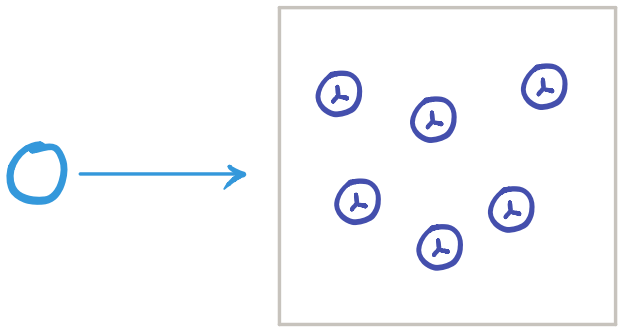
Blockchain

- PoW (Proof of Work):
- PoS (Proof of Stake):
- DPoS (Delegated Proof of Stake):
- PoI (Proof of Importance):
- PoD (Proof of Devotion)
- PBFT (Practical Byzantine Fault Tolerance)
- FPA (Federated Byzantine Agreement)
- Hybrid PoW/PoS
- Proof-of-DDOS

untrusted

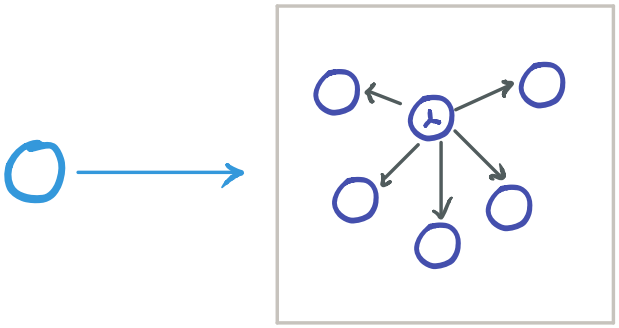
Purpose: decrease the incentive for people to misbehave.

- unfeasible to attack



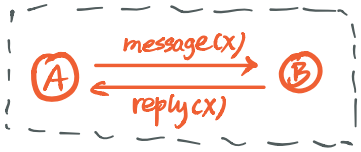
Symmetric consensus

Any server can respond

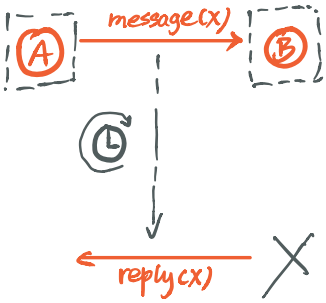


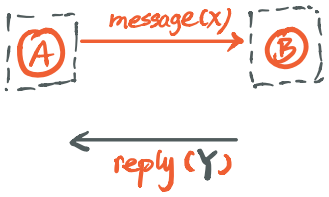
Asymmetric consensus

A leader is elected to issue command



a locality bubble





Bob doesn't respond

Bob reply with wrong message

Someone pretend to be Bob

Blockchain – proof based, slow

Paxos – hard to implement and understand

Paxos

Phase 1. (a) A proposer selects a proposal number n and sends a *prepare* request with number n to a majority of acceptors.

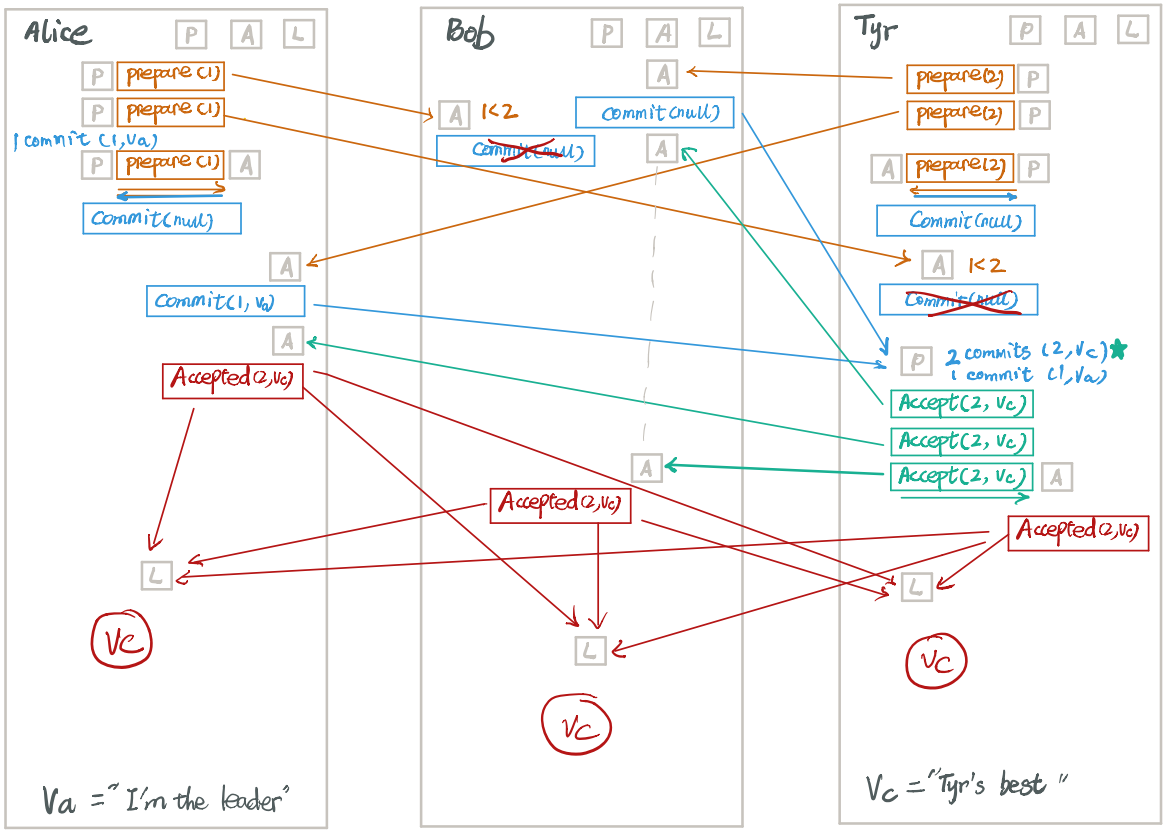
(b) If an acceptor receives a *prepare* request with number n greater than that of any *prepare* request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.

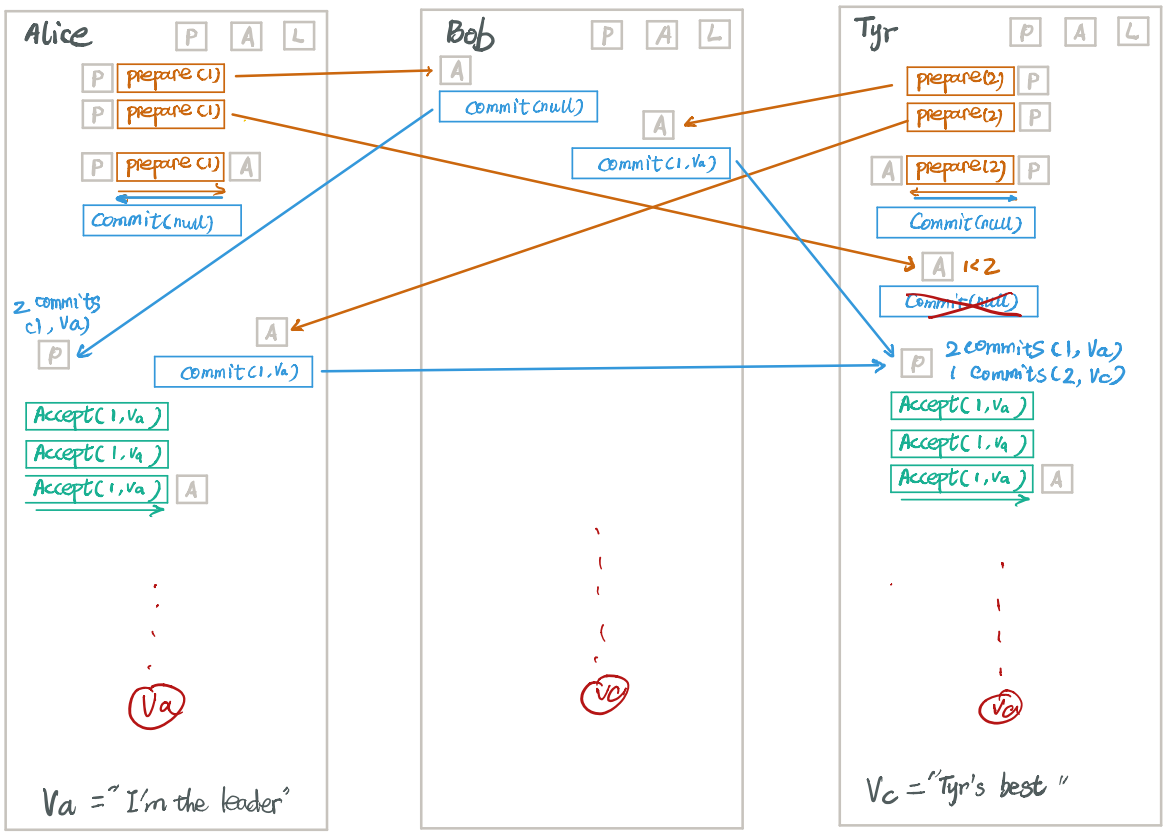
Phase 2. (a) If the proposer receives a response to its *prepare* requests (numbered n) from a majority of acceptors, then it sends an *accept* request to each of those acceptors for a proposal numbered n with a value v, where v is the value of the highest-numbered proposal among the responses, or is any value if the responses reported no proposals.

(b) If an acceptor receives an *accept* request for a proposal numbered n, it accepts the proposal unless it has already responded to a *prepare* request having a number greater than n.

Phase 3.

To learn that a value has been chosen, a learner must find out that a proposal has been accepted by a majority of acceptors. The obvious algorithm is to have each acceptor, whenever it accepts a proposal, respond to all learners, sending them the proposal.




Alice
P A L
P prepare(1)
P prepare(1)
P prepare(1) A
commit(null)
2 commits (1, Va)
P
A
commit(1, Va)
Accept(1, Va)
Accept(1, Va)
Accept(1, Va) A
Va
Va = "I'm the leader"
Bob
P A L
A
commit(null)
A
commit(1, Va)
Vc
Tyr
P A L
prepare(2) P
prepare(2) P
A prepare(2) P
Commit(null)
A 1<2
Commit(null)
P 2 commits (1, Va)
1 commits (2, Vc)
Accept(1, Va)
Accept(1, Va)
Accept(1, Va) A
Vc
Vc = "Tyr's best"

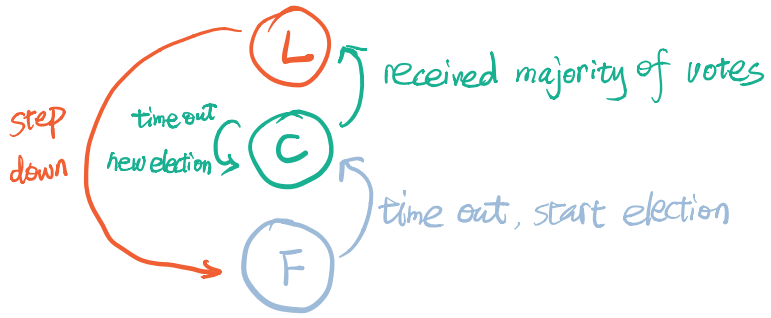
Raft

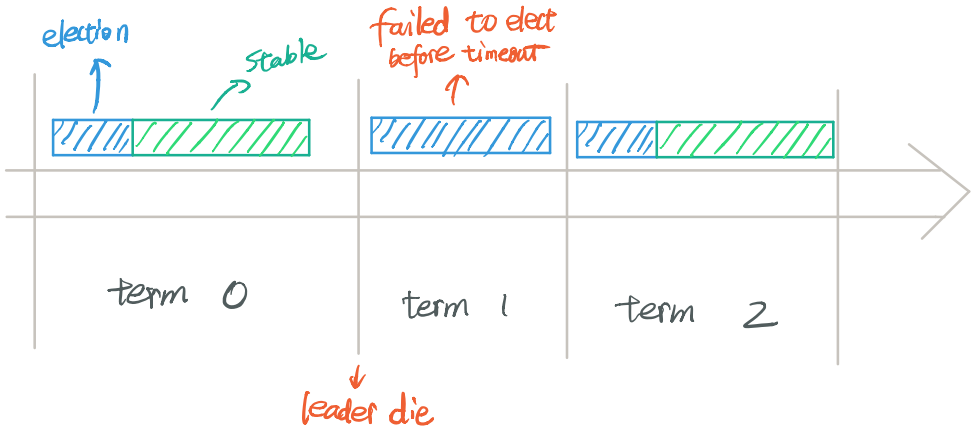
Goal: Have and available replicated state machine.

- available: provide meaningful respone for a log
- replicated: makes other nodes aware of its log
- state machine: follow the log to arrive at the same state

Assymetric

- leader :
- Candidate:
- Follower





Stable >> election

Election

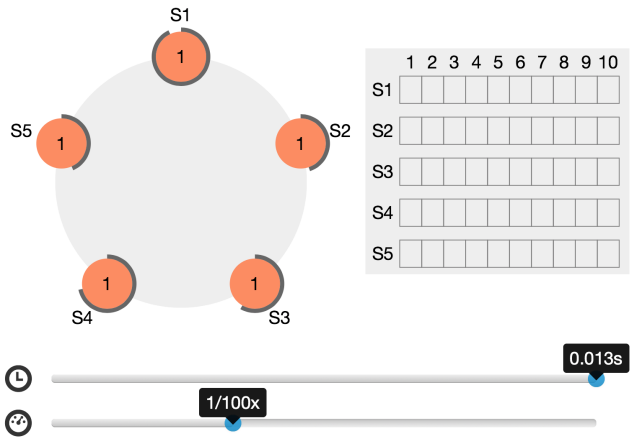
1. Increment current term
2. Vote for self
3. request votes

properties {
- Safety: at most one winner per term.
- Liveness: Someone eventually wins.

outcome {
- I'm leader: receive majority votes
- I'm not: receive valid heart beat
- no leader: election timeout

Raft Visualization

Here's a Raft cluster running in your browser. You can interact with it to see Raft in action. Five servers are shown on the left, and their logs are shown on the right. We hope to create a screencast soon to explain what's going on. This visualization (RaftScope) is still pretty rough around the edges; pull requests would be very welcome.

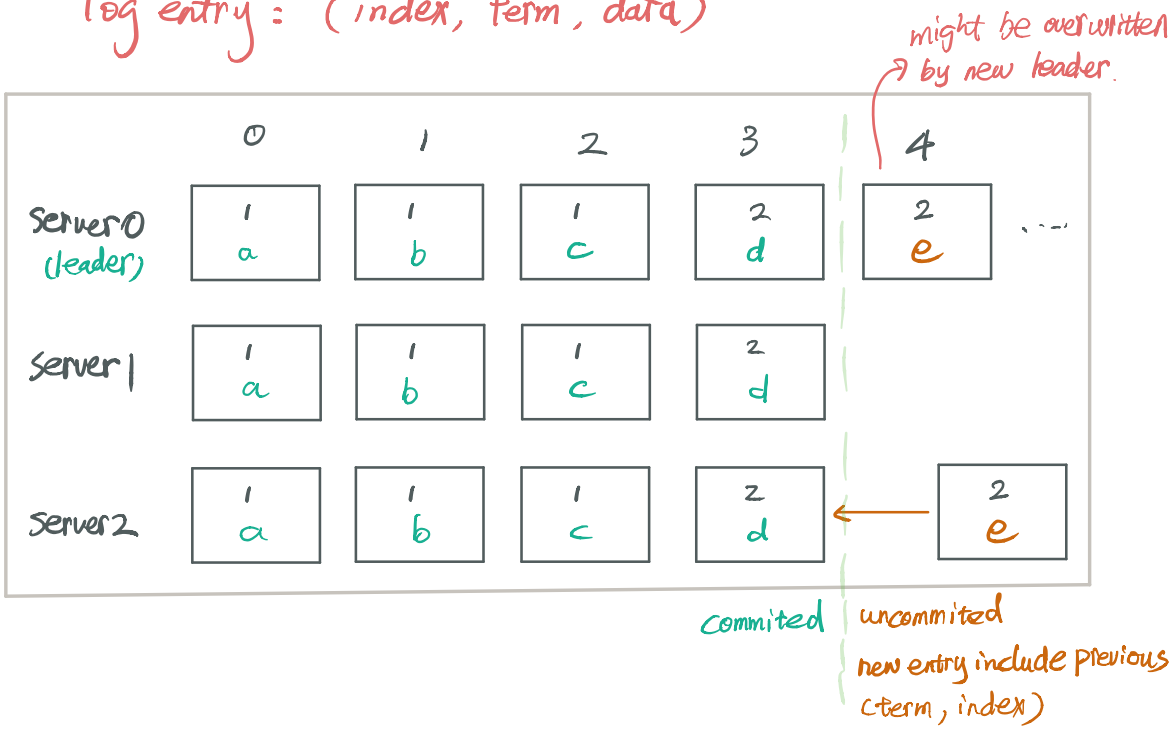


The Secret Lives of Data is a different visualization of Raft. It's more guided and less interactive, so it may be a gentler starting point.

Log Replication

└ exchange information

log entry : (index, term, data)



log replication
- log safety : committed entries in the logs of all future leaders.
- log coherency : deny votes if candidate has "less complete" log
- log integrity : an entry is committed if 1) it's majority stored;
  2) at least one entry from leader's term is also majority-stored.

who's using : Cloud Foudary, Kubernetes, etcd, docker swam, Consul

not vote
- if your term is larger
- if your index is better

$(t_v > t_c)$ or $((t_v = t_c)$ and $(i_v > i_c))$

PoW

Pros:

- attack is hard, unfeasible
- the coin is precious given that so much has bee spent.

Cons:

- amateur can't compete with professional miners. (CPU vs ASIC/GPU)
- Matthew Effect.

PoS

Pros:

- Attackers have no incentive to attack
- Inexpensive. low cost.

Cons:

- Matthew Law

PoI (Proof of Importance)

Not only reward those with large balance, but also take into account how much they transact to others and who they transact with.

Pros:

- Good incentive system. The more you use, the better you importance (trust score) is.