Blockchains and Consensus Protocols

Gengrui(Edward) Zhang

Ph.D. Student

Dept. of Electrical & Computer Engineering

UNIVERSITY OF TORONTO

Blockchain is a P2P distributed ledger technology based on cryptographic algorithms. Its essence is an Internet shared database.

Permissionless Blockchain

(Open, Entirely Decentralized. Every node can freely join or left)

Permissioned Blockchain

(For cooperation, node management is required, and each node needs a global address record)

Cross chain

Proof-of-X

- ⇒ Proof-of-Work, PoW
- ⇒ Proof-of-Stake, PoS

Replicated State Machine, Repl.SM
Byzantine Fault Tolerance, BFT

⇒ Paxos, ⇒ Raft,
⇒ Practical Byzantine Fault Tolerance, PBFT

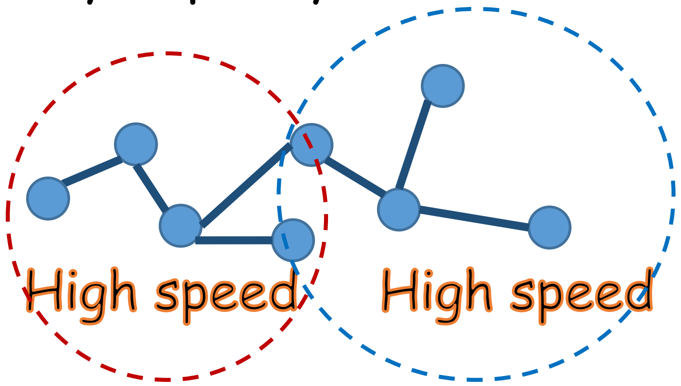
Blockchain is a P2P distributed ledger technology based on cryptographic algorithms. Its essence is an Internet shared database.

Permissionless Blockchain

(Open, Entirely Decentralized. Every node can freely join or left)

No node management …
(don't need to know other nodes' IP)

Maybe partly connected …



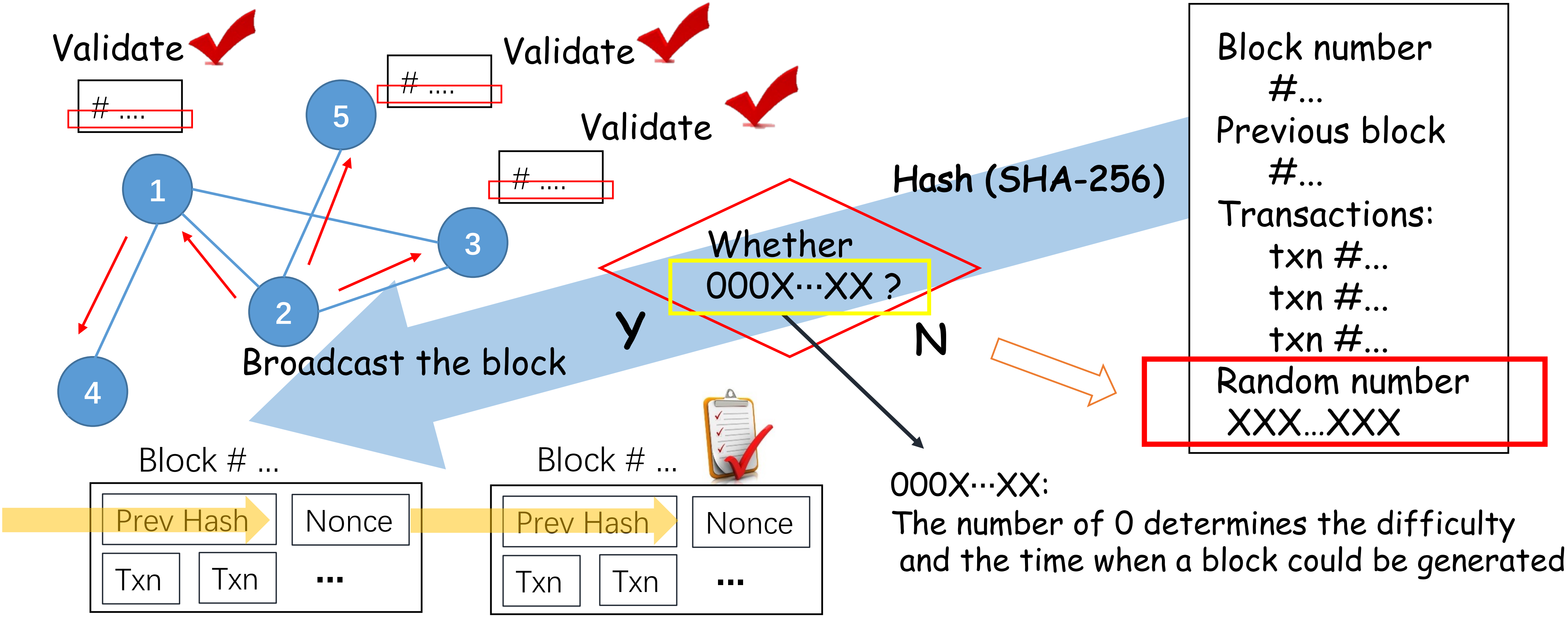
Merge= Longest(Chain A, Chain B)

Proof-of-X

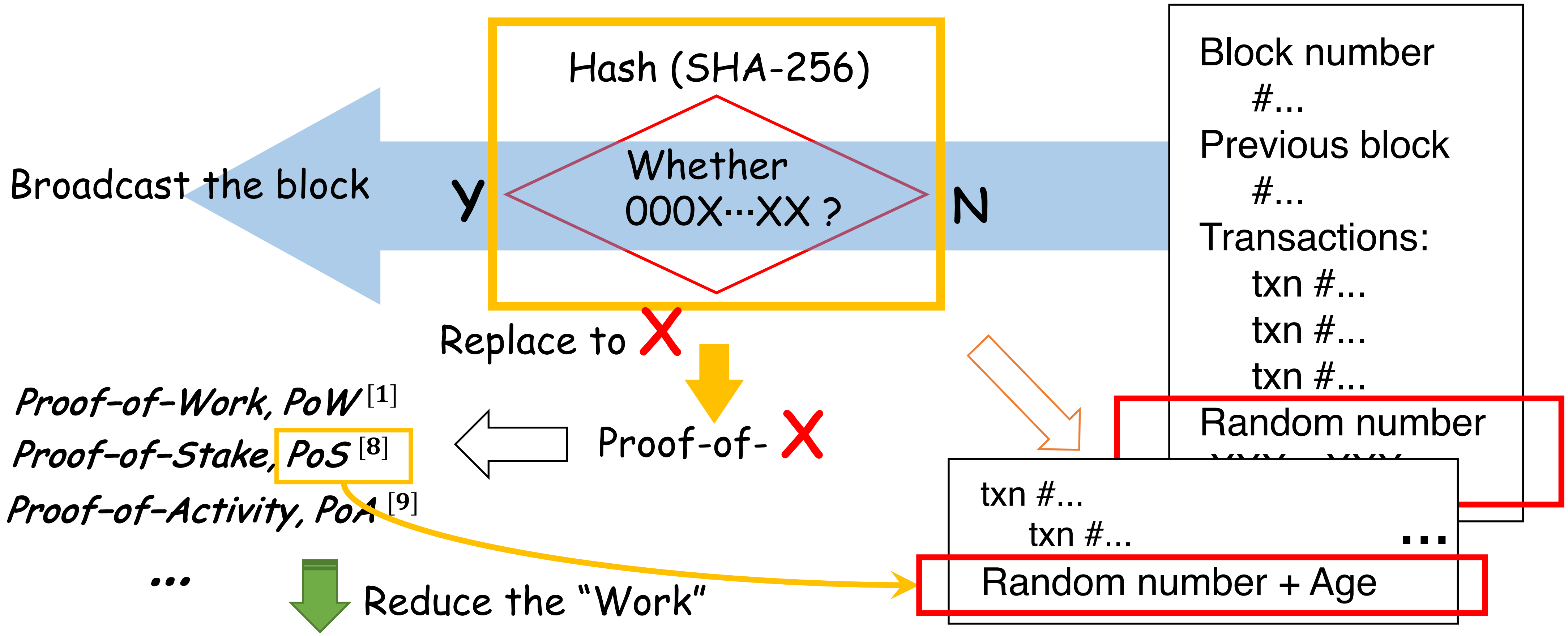
⇒ Proof-of-Work, PoW
⇒ Proof-of-Stake, PoS

Proof-of-Work, PoW

[1] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[J]. 2008.



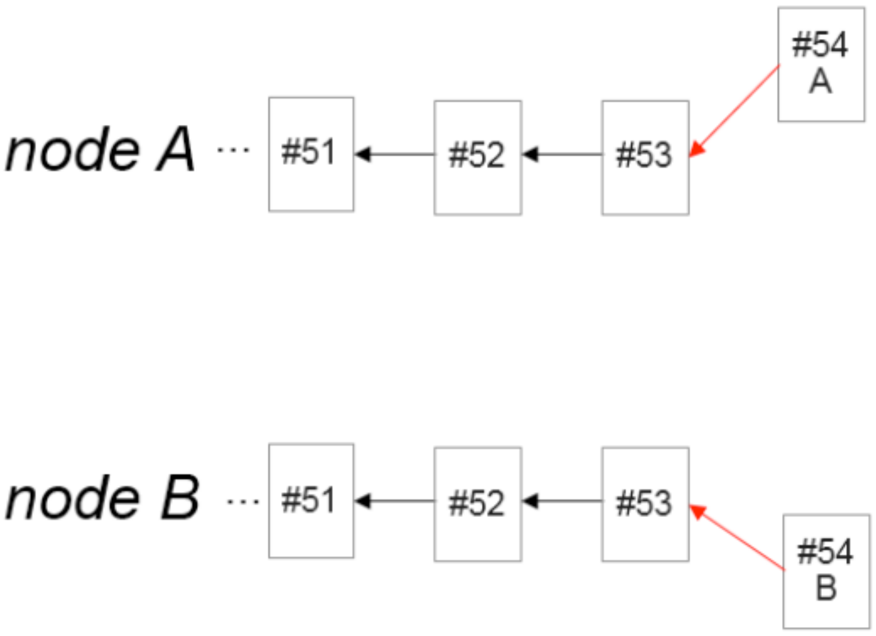
Protocols for Permissionless Blockchains



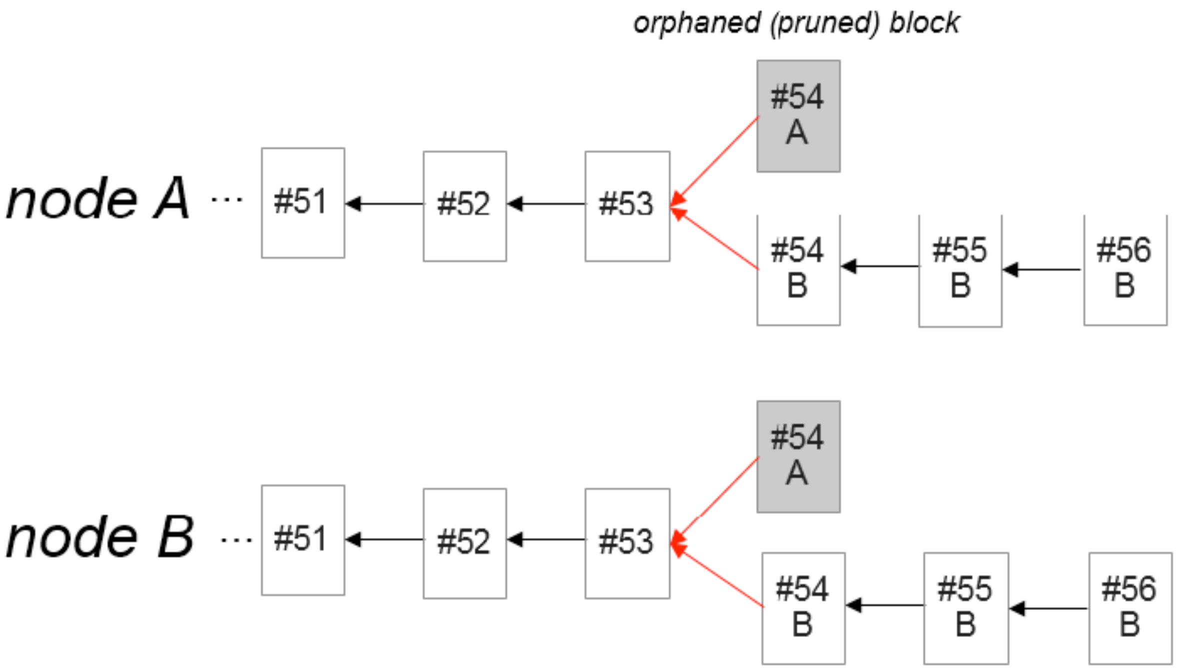
[2] King S, Nadal S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake[J]. self-published paper, August, 2012, 19.

Double-Spending / Chain-forks

[4] Eyal I, Gencer A E, Sirer E G, et al. Bitcoin-NG: A Scalable Blockchain Protocol[C]//NSDI. 2016: 45-59.

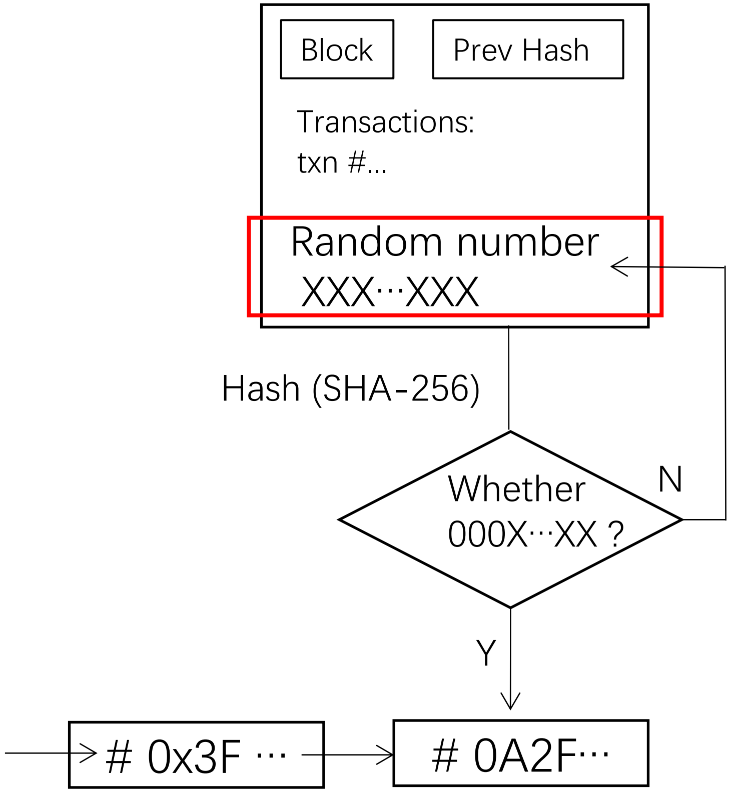


(a) Consensus finality violation resulting in a fork



(b) Eventually, one of the blocks must be pruned by a conflict resolution rule (e.g., Bitcoin's longest chain rule).

Features of Permissionless Blockchains



Features [10] :

- † Open, entirely decentralized
- † No Consensus finality
- † Good Scalability
- † Limited Throughput
- † High Latency
- † Waste Power
- † ~~Fault Tolerance ?~~
- † No correctness proofs

Due to the design of Protocols e.g. block size, difficulty of proof

Due to multi-block confirmations

Useless calculations

Applications:

Bitcoin **Ripple** **Ethereum** **Sawtooth Lake**

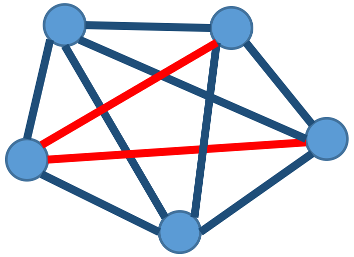
Blockchain is a P2P distributed ledger technology based on cryptographic algorithms. Its essence is an Internet shared database.

MUST need node management …
(Each node needs to know other nodes' IP)

Should be fully connected, freely communicate to each other (complete graph)

<Simple Majority Rules>

May influence throughput, latency



Merge= Majority(Chain A, Chain B)

⇩

Permissioned Blockchain

(For cooperation, node management is required, and each node needs a global address record)

⇩

Replicated State Machine, Repl.SM
Byzantine Fault Tolerance, BFT

⇒ Paxos, ⇒ Raft,
⇒ Practical Byzantine Fault Tolerance, PBFT

Permissioned Blockchain

↓

Coordination and Agreement in distributed system

↓

Interactive consistency	Consensus	Byzantine generals
"decision vector"	"crash, omissions"	"arbitrary failures"

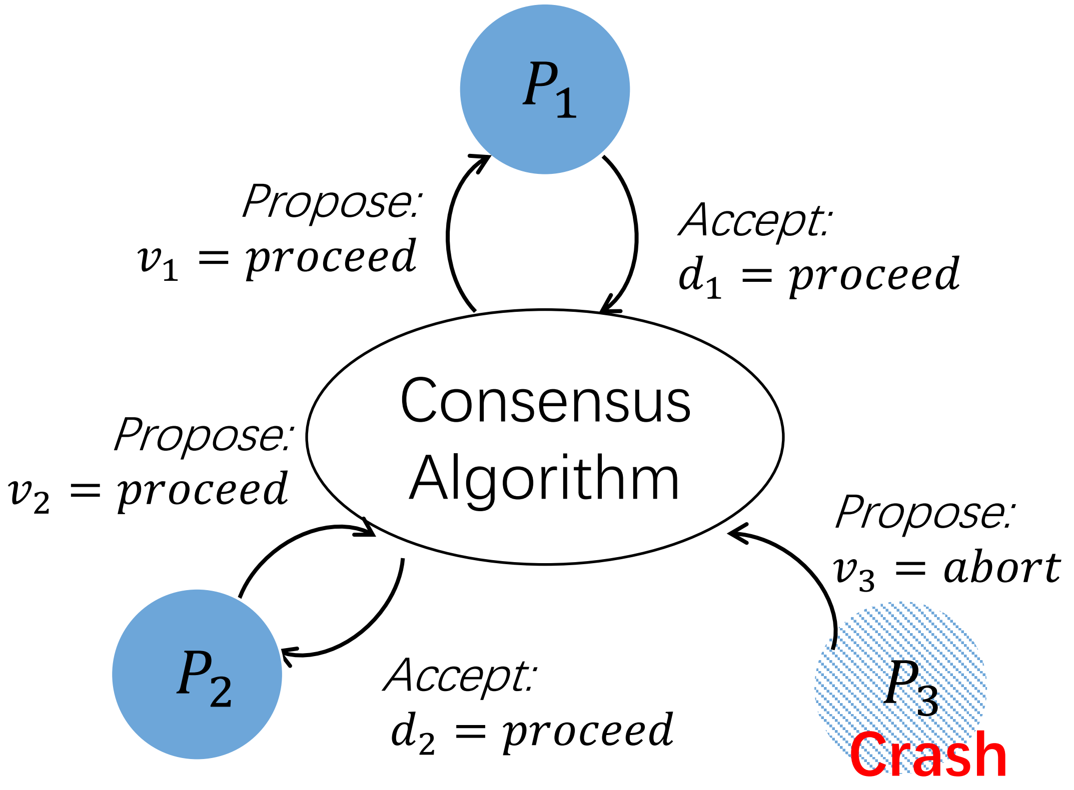
Consensus problem

" To reach consensus, every process p_i begins in the ***undecided*** state and ***proposes*** a single value v_i, drawn from a set D $(i \in N^*)$. The processes communicate with one another, exchanging values. Each process then sets the value of a ***decision variable***, d_i. In doing so it enters the ***decided*** state, in which it may no longer change $d_i (i \in N^*)$"

—— *《Distributed Systems Concepts and Design》*

Replicated State Machine

Byzantine Fault Tolerance, BFT



Consensus for three processes

Fault-tolerance

Crash

Omission

Byzantine

⇒ Paxos:

How to choose a value?

stronger assumption

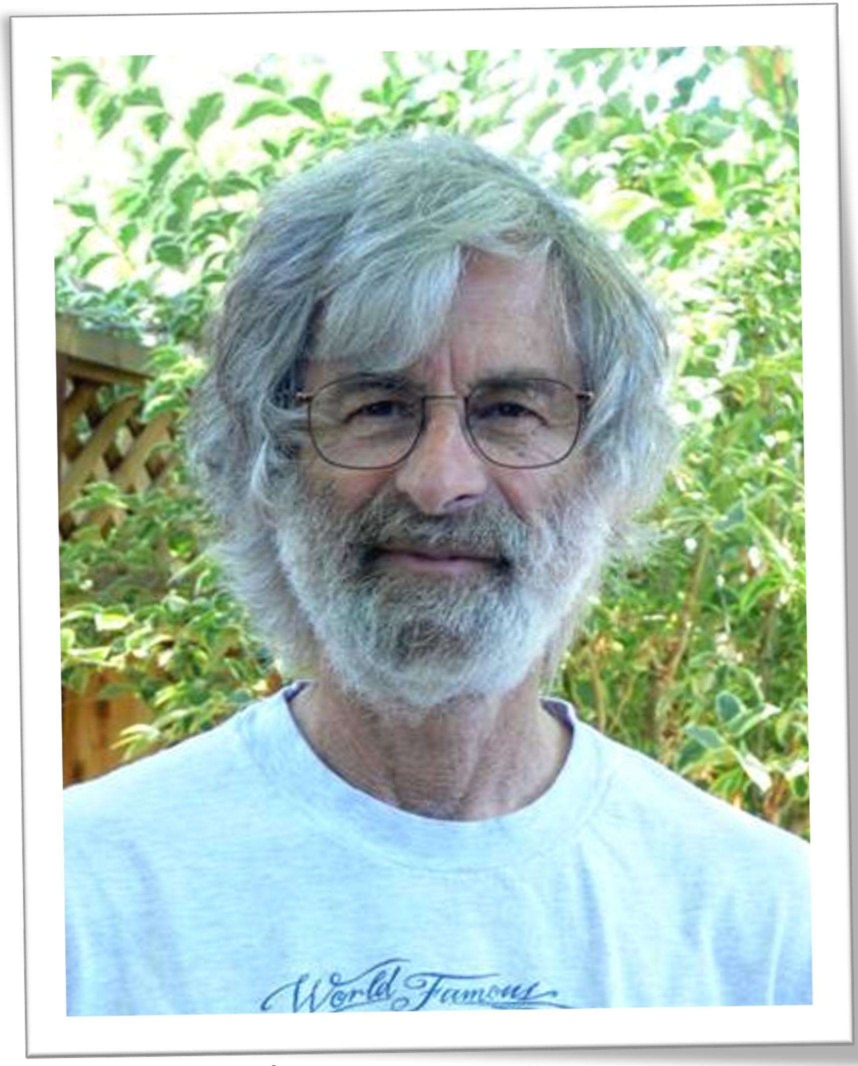
⇒ Raft:

How to replicate a log?

⇒ PBFT:

How to guarantee the correctness under Byzantine conditions?

stronger assumption



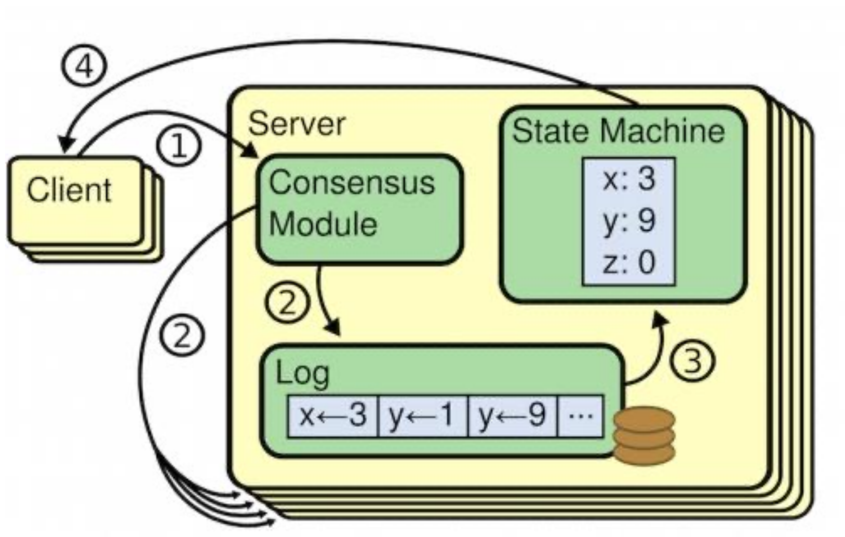
Leslie Lamport

Lamport's research contributions have laid the foundations of the theory of distributed systems.

- "**Time, Clocks, and the Ordering of Events in a Distributed System**", which received the PODC Influential Paper Award in 2000,
- "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs", which **defined the notion of Sequential consistency**,
- "**The Byzantine Generals' Problem**",
- "Distributed Snapshots: **Determining Global States of a Distributed System**" and
- "**The Part-Time Parliament**".

http://www.lamport.org

Replicated State Machine



[5] Schneider F B. Implementing fault-tolerant services using the state machine approach: A tutorial[J]. ACM Computing Surveys (CSUR), 1990, 22(4): 299-319.

- The consensus algorithm manages a replicated log containing state machine commands from clients.
- The state machine process identical sequences of commands from the logs, so they produce the same outputs.

Paxos ***Raft*** ***ViewStamp*** ***Zab***

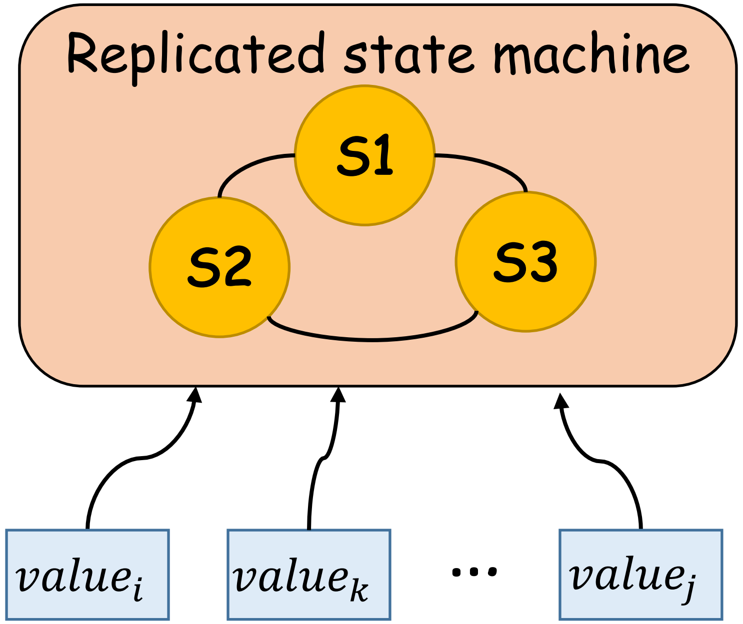
⇩

Ensure Safety under non-Byzantine Conditions, including network delays, partitions, and packet loss, duplication, and reordering

Paxos

System model: Asynchronous, non-Byzantine.

Servers: Proposers, Acceptors, Learners



[6] Lamport L. Time, clocks, and the ordering of events in a distributed system[J]. Communications of the ACM, 1978, 21(7): 558-565.

[7] Lamport L. The part-time parliament[J]. ACM Transactions on Computer Systems (TOCS), 1998, 16(2): 133-169.

[8] Lamport L. Paxos made simple[J]. ACM Sigact News, 2001, 32(4): 18-25.

[9] Lampson B. The ABCD's of Paxos[C]//PODC. 2001, 1: 13.

Safety & Liveness

The Safety requirements for consensus are:

- Only a value that has been proposed may be chosen.
- Only a single value is chosen, and
- A process never learns that a value has been chosen unless it actually has been.

The Liveness requirements for consensus are:

- Some proposed value is eventually chosen.
- If a value is chosen, servers eventually learn about it.

Server

- **Proposers**
  - -> Active: put forth particular values to be chosen.
  - -> Handle client requests.

&

- **Acceptors**
  - -> Passive: respond to messages from proposers.
  - -> Responses represent votes that from consensus.
  - -> Store chosen value, state of the decision process.
  - -> Want to know which value was chosen.

Proposal

Each proposal has a unique number (proposal number)

-> Higher number take a priority over lower numbers.

-> It must be possible for a proposer to chose a new proposal number higher than anything it has seen/used before.

Proposal Number
Round Number
ServerId

-> Each server stores maxRound: the Largest Round Number it has been so far.

-> To generate a new proposal number:

(1) Increment maxRound. (2) Concatenate with ServerId.

-> Proposers must persist maxRound on disk: must not reuse proposal numbers after crash /restart.

Proposers

(1) Choose new proposal number *n*.
(2) Broadcast Prepare(*n*) to all servers.

(4) When responses received from majority, if any *acceptedValue* returned, replace value with *acceptedValue* for highest *acceptedProposal*.

(5) Broadcast Accept(*n*, *value*) to all servers

(7) When responses received from majority:
-> Any rejections (result > n) : go to (1)
-> Otherwise, value is chosen

Acceptors

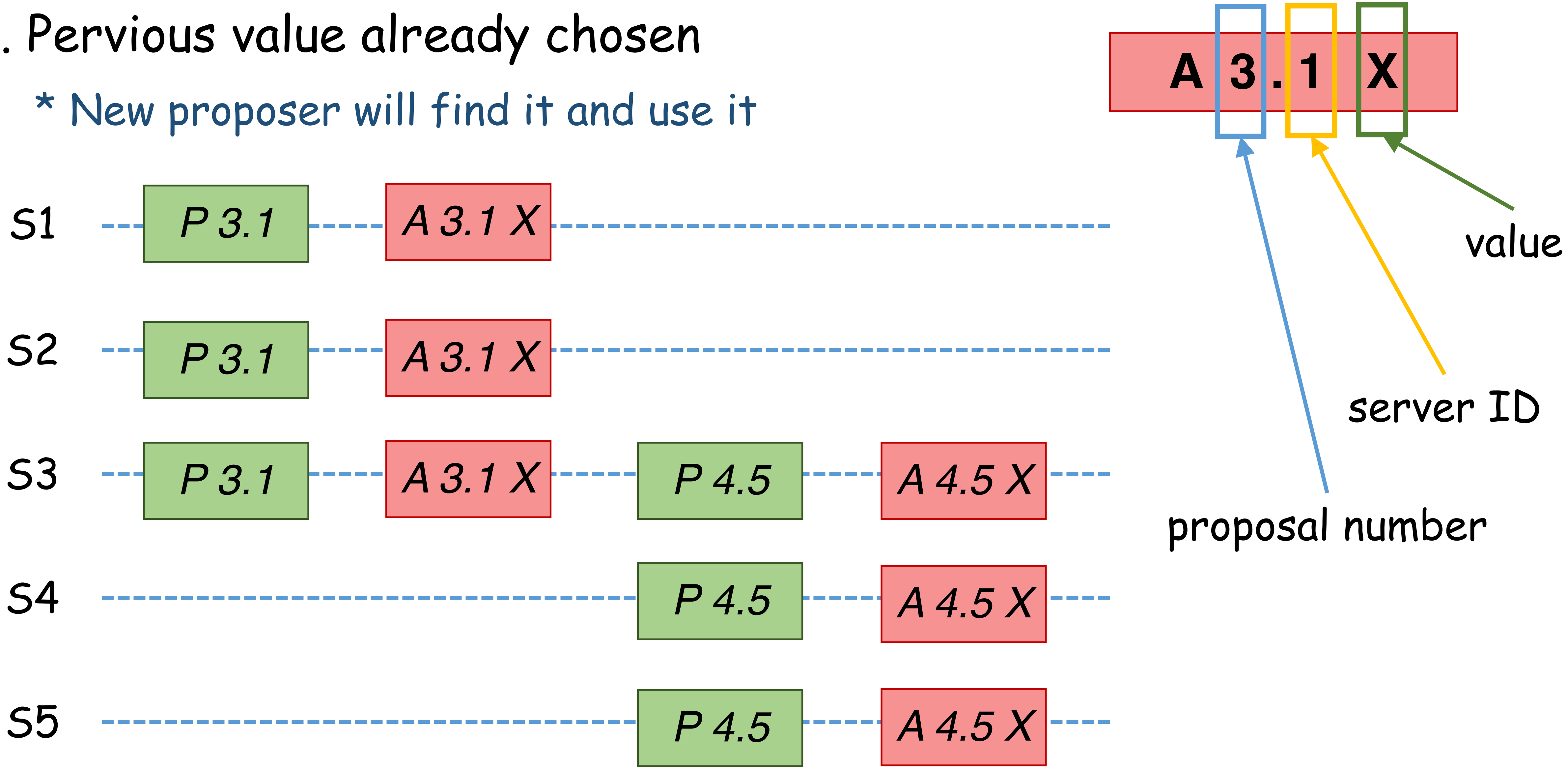
(3) Respond to Prepare(*n*):
-> If n > *minProposal*, then *minProposal* = *n*
-> Return (*acceptedProposal*, *acceptedValue*)

(6) Respond to Accept(*n*, *value*):
-> If *n* >= *minProposal* then *acceptedProposal* = *minProposal* = *n*; *acceptedValue* = *value*;
-> Return (minProposal)

Acceptors must record *minProposal*, *acceptedProposal*, and *acceptedValue* on stable storage (disk).

1. Pervious value already chosen

* New proposer will find it and use it

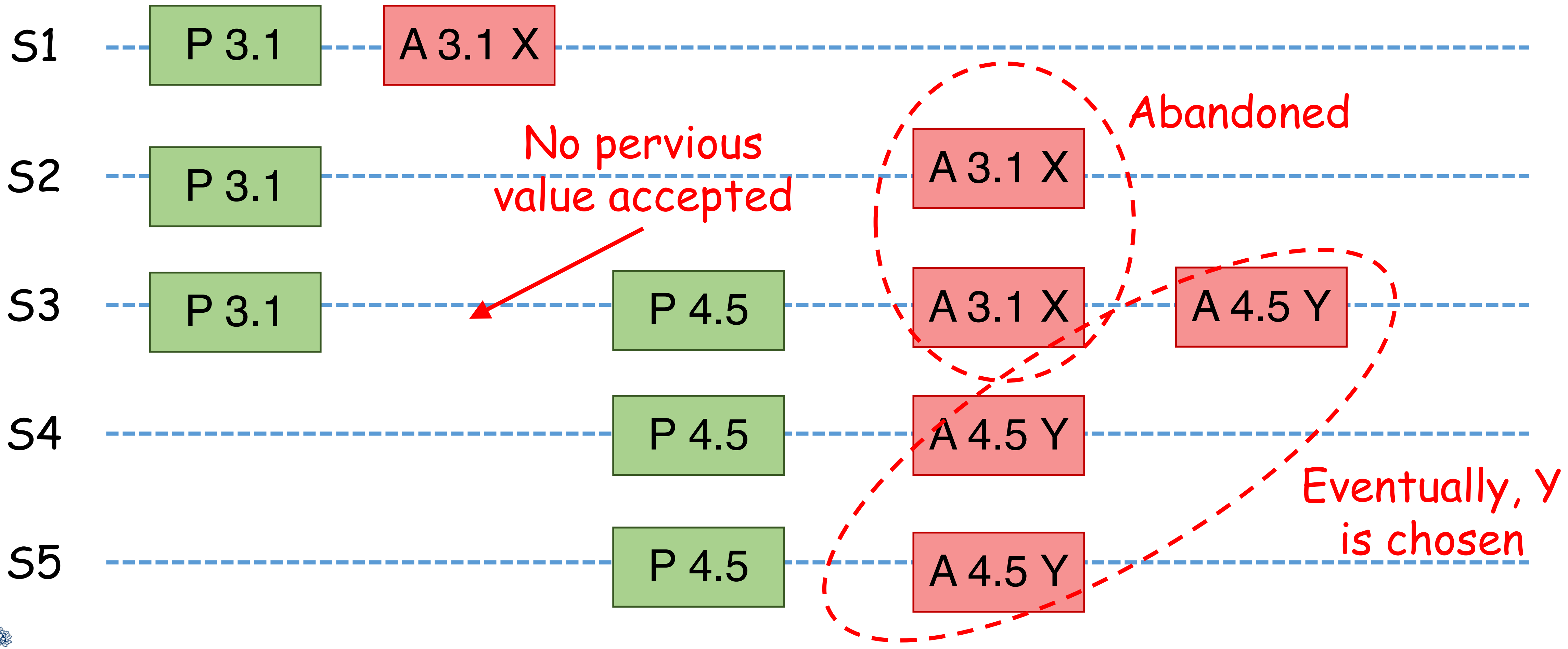


2. Pervious value not chosen, but proposer sees it

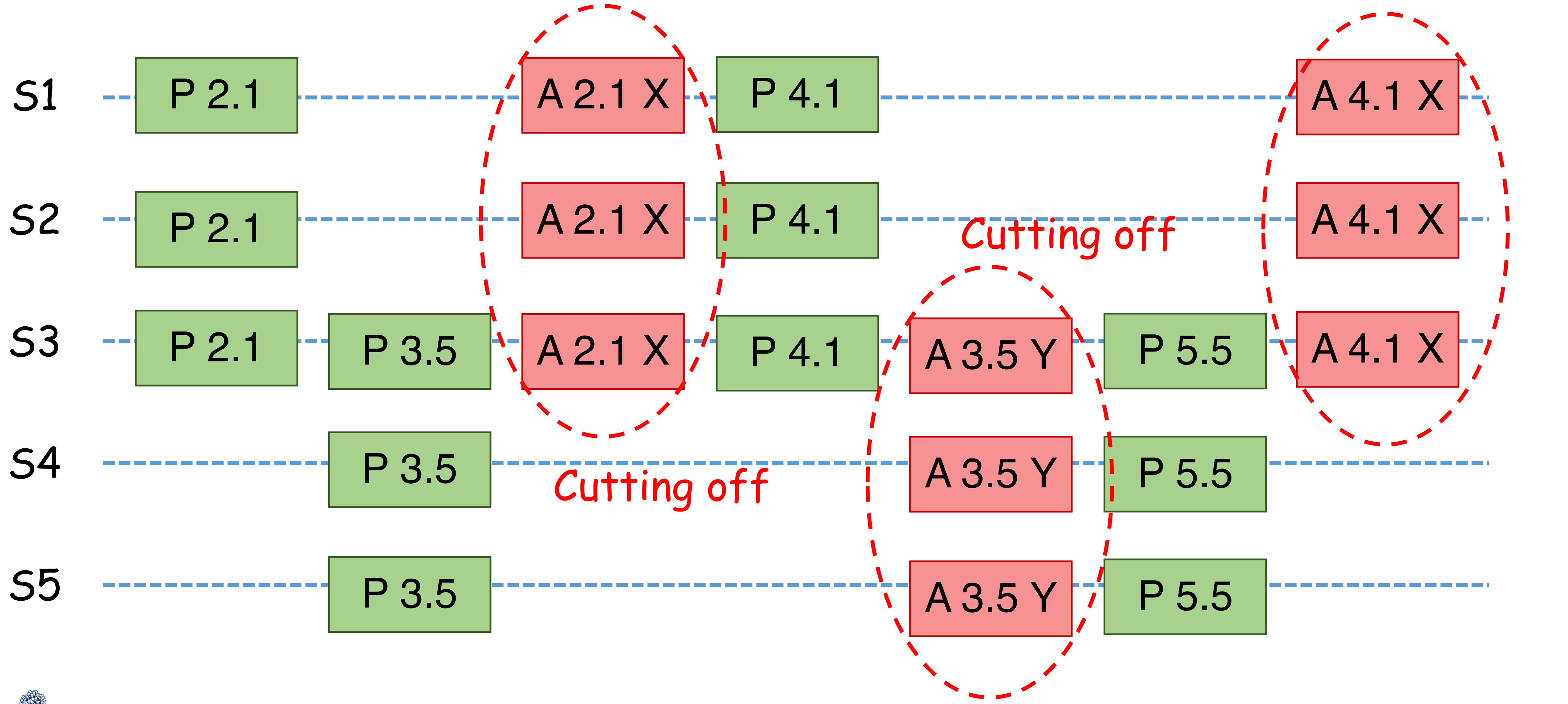
- New proposer will use exiting value
- Both proposers can succeed

3. Pervious value not chosen, new proposer doesn't see it

- New proposer chooses its own value
- Older proposal blocked



Livelock: Competing proposers can livelock



Disadvantages in Basic Paxos

-> Competing proposers can *Livelock*.

-> Only proposer knows which value has been chosen.

-> If other servers want to know, must execute Paxos with their own proposal.

Hint:
=> one solution:
Randomized delay before restarting. Give other proposers a chance to finish choosing.

Anyone can be a proposer.
(Advantages/Disadvantages)

↓

Handle the request with a leader.

↓

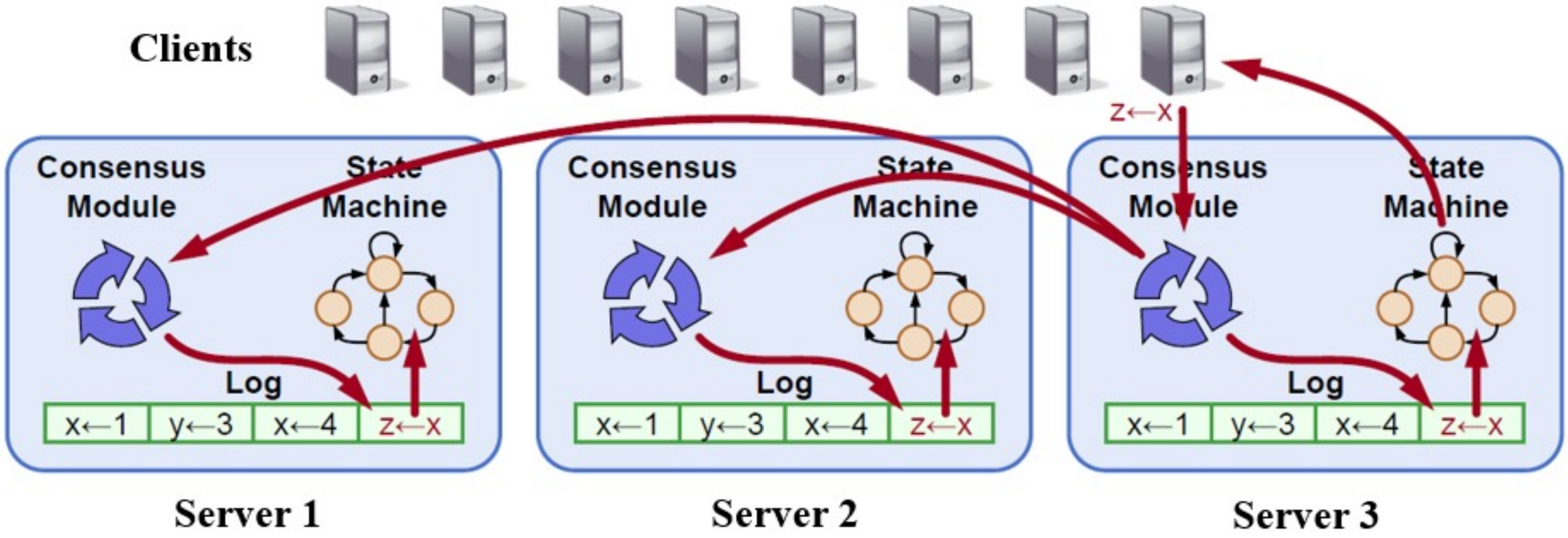
Multi-Paxos, Raft , Zab

Raft

[10] Ongaro D, Ousterhout J K. In search of an understandable consensus algorithm[C]//USENIX Annual Technical Conference. 2014: 305-319.

Strong leader

Raft uses a stronger form of leadership than other consensus algorithm. For example, log entries only flow from the leader to other servers. This simplifies the management of the replicated log an makes Raft easier to understand.



Server states: **Follower** **Candidate** **Leader**

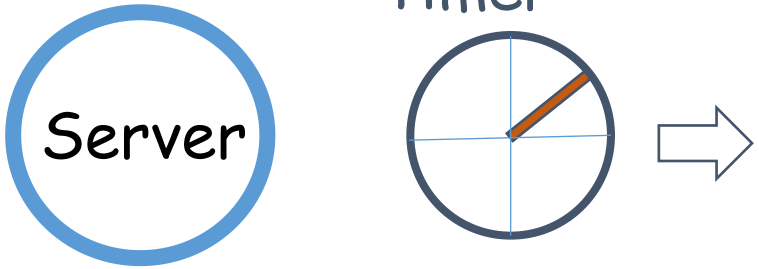
Follower: Followers are passive: they issue no requests on their own but simply respond to requests from leaders and candidates.

Candidate: The candidate is used to elect a new leader. (using RequestVote RPC)

Leader: The leader handles all client requests (using AppendEntries RPC).

! => In normal operation there is exactly one leader and all of the other servers are followers.

Timer



initial time t_i

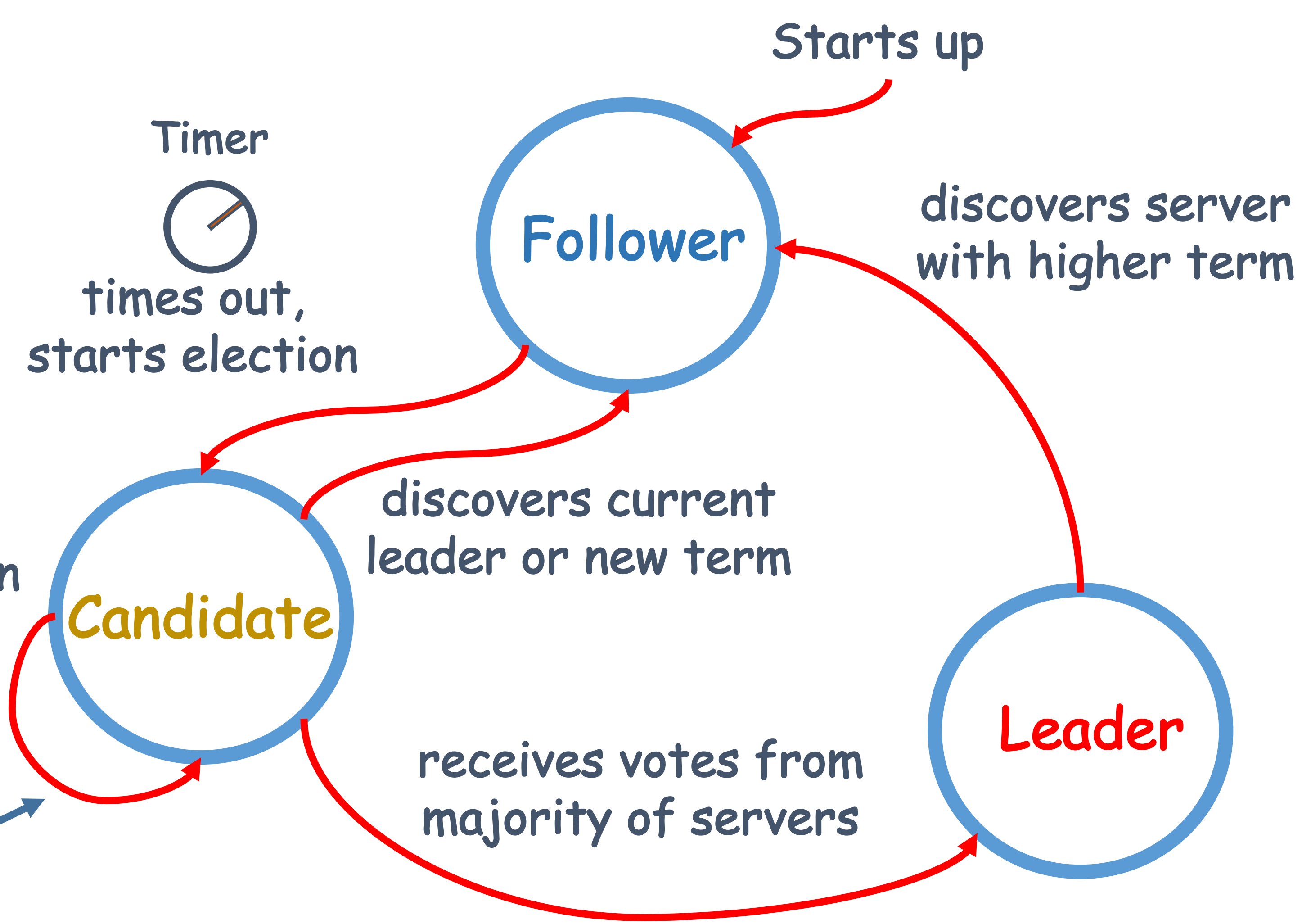
- trigger a timeout
- Reset to the initial time

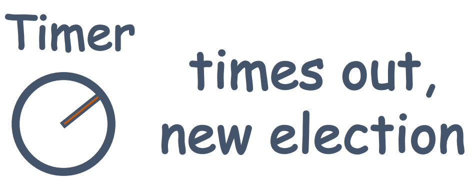
RequestVote RPC → Considers there is no alive leader and begins an election to choose a new leader.

AppendEntries RPC → A server remains in follower state as long as it receives valid RPCs from a leader or candidate.

Leader election

1. increment current *term*.
2. vote for self.
3. Reset election timer
4. Send **RequestVote RPCs** to all other servers

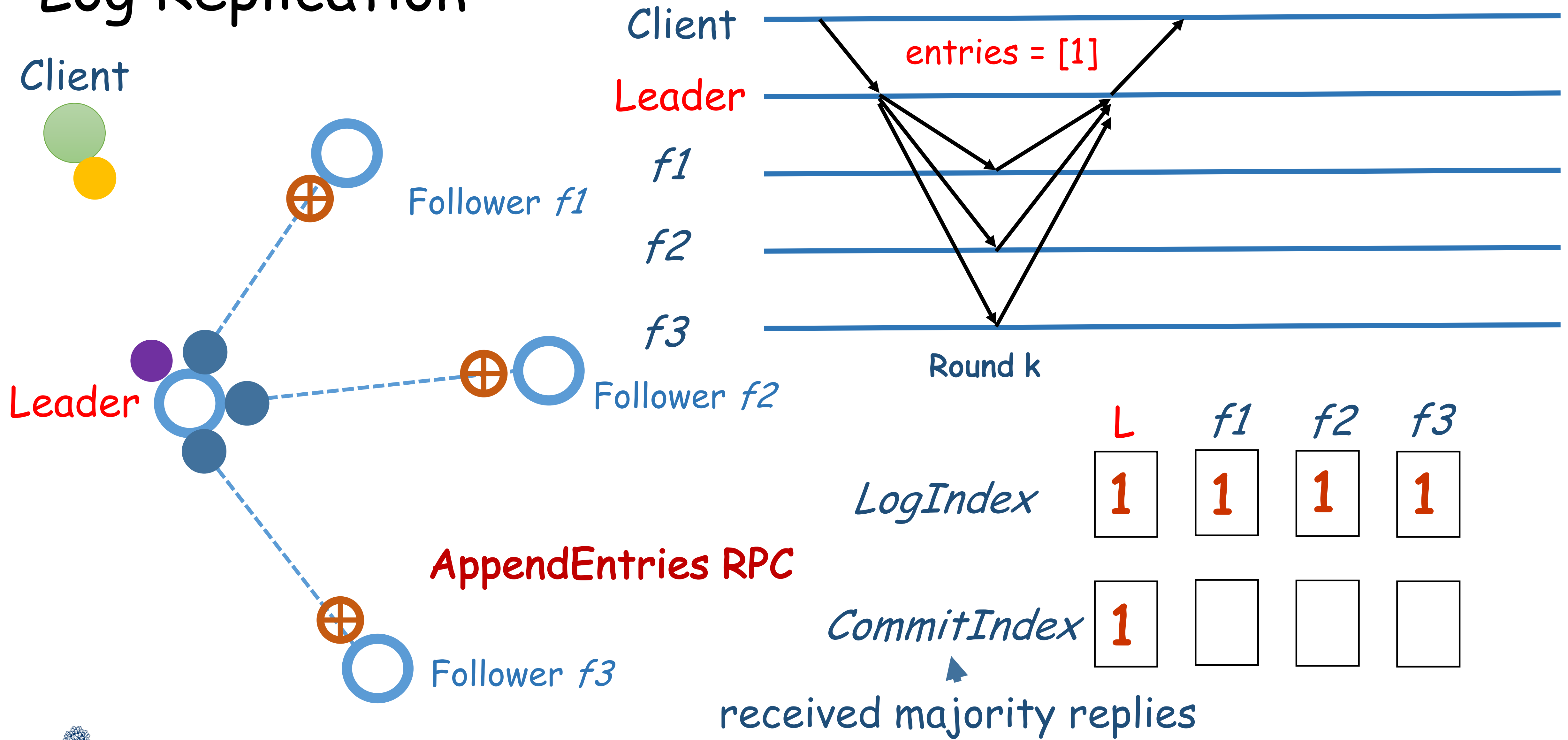




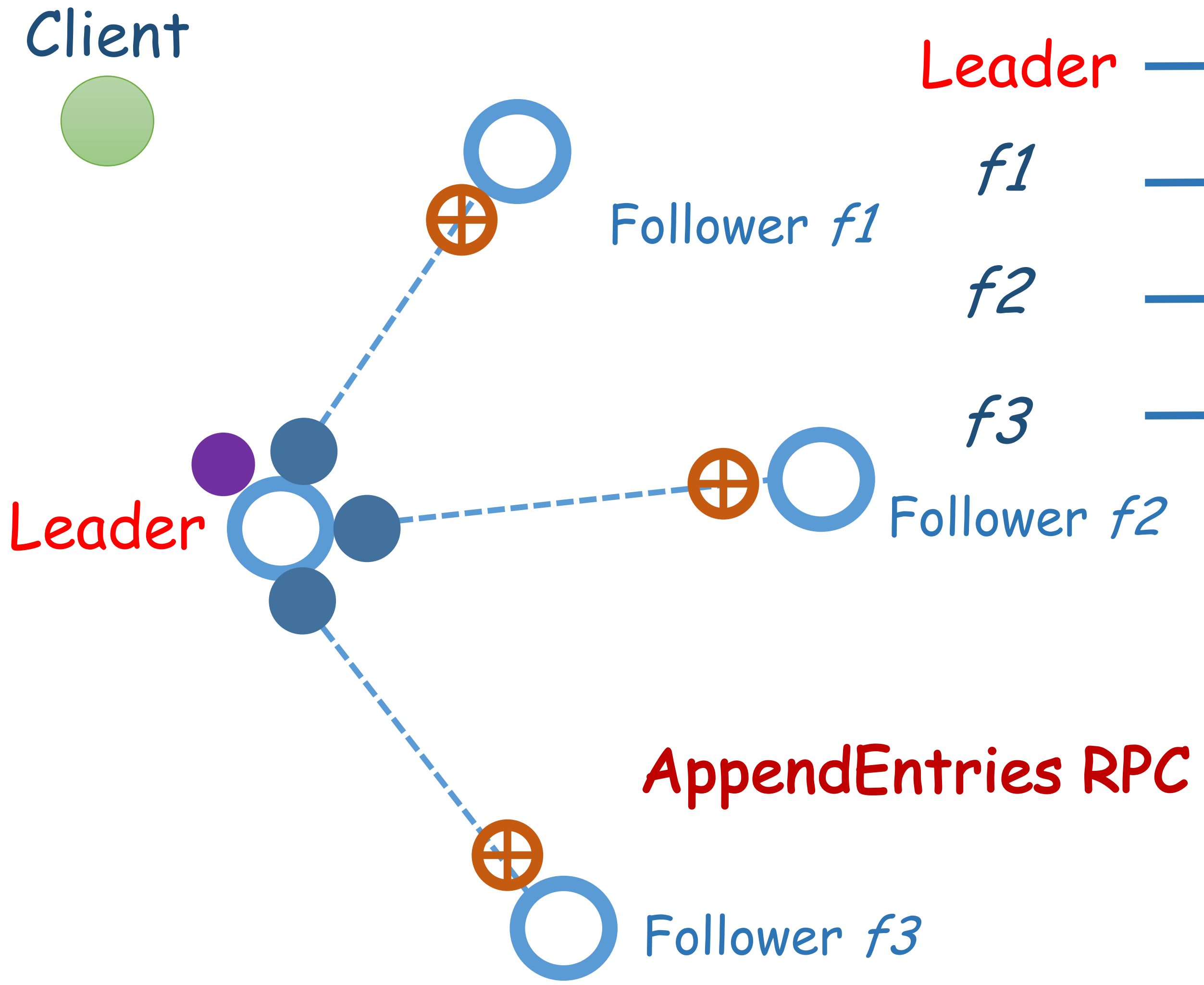
A candidate continues in this state until three things happens:

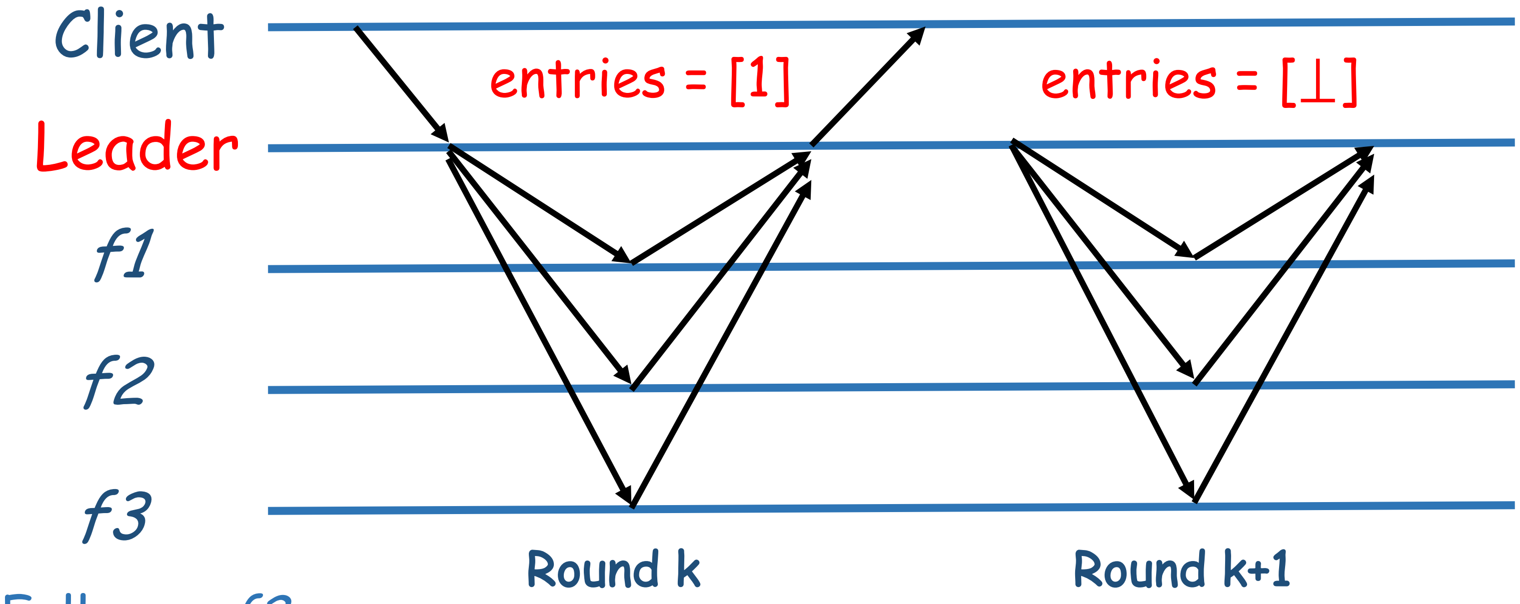
1. it wins the election.
2. another server establish itself as leader,
3. a period of time goes by with no winner.

Log Replication



Log Replication





	L	f1	f2	f3
LogIndex	1	1	1	1
CommitIndex	1	1	1	1

Term

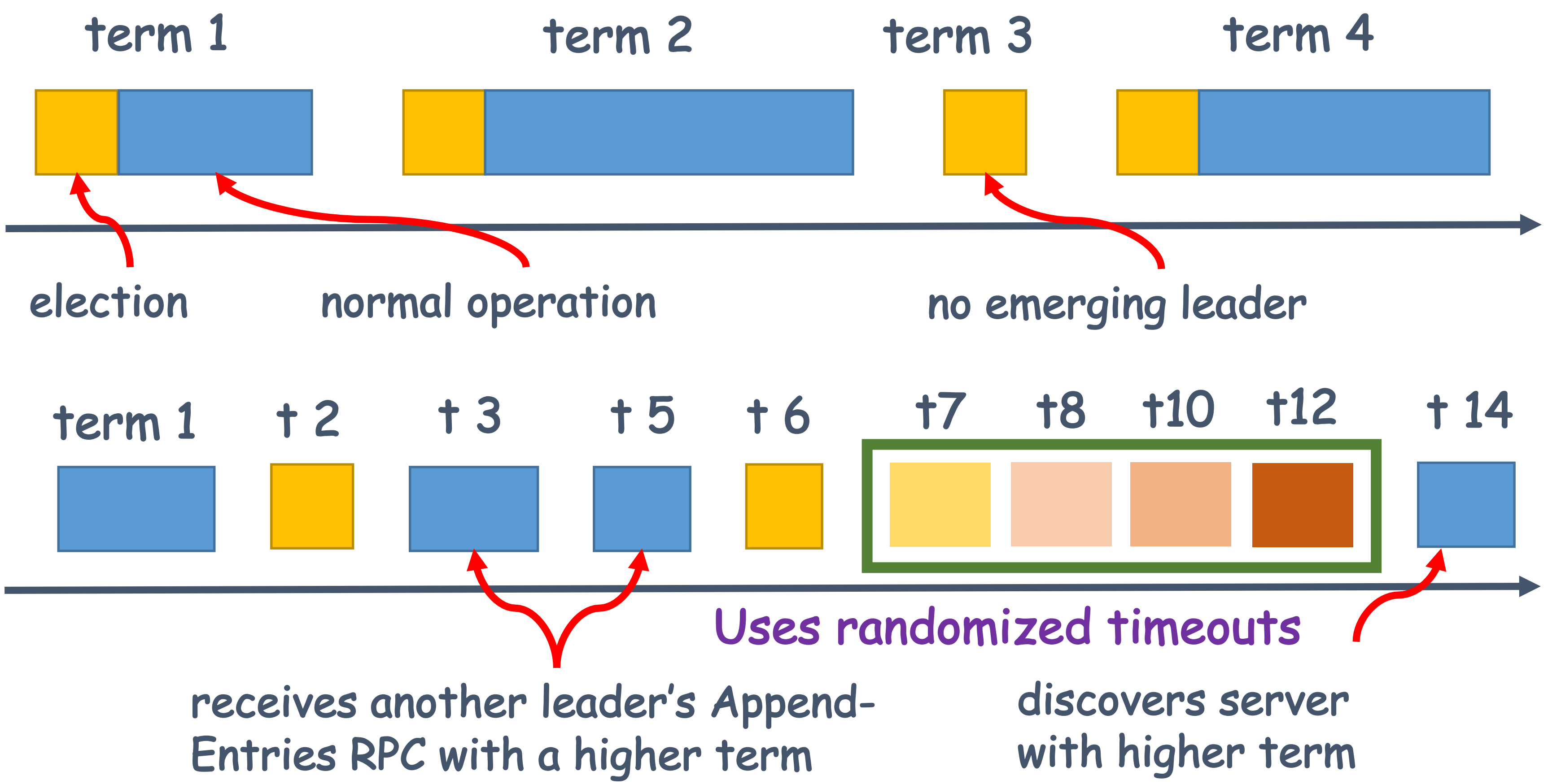
Time is divided into terms, and each term begins with an election. After a successful election, a single leader manages the cluster until the end of the term. Some elections fail, in which case the term ends without choosing a leader. The transitions between terms may be observed at different times on different servers.

In a system's dimension

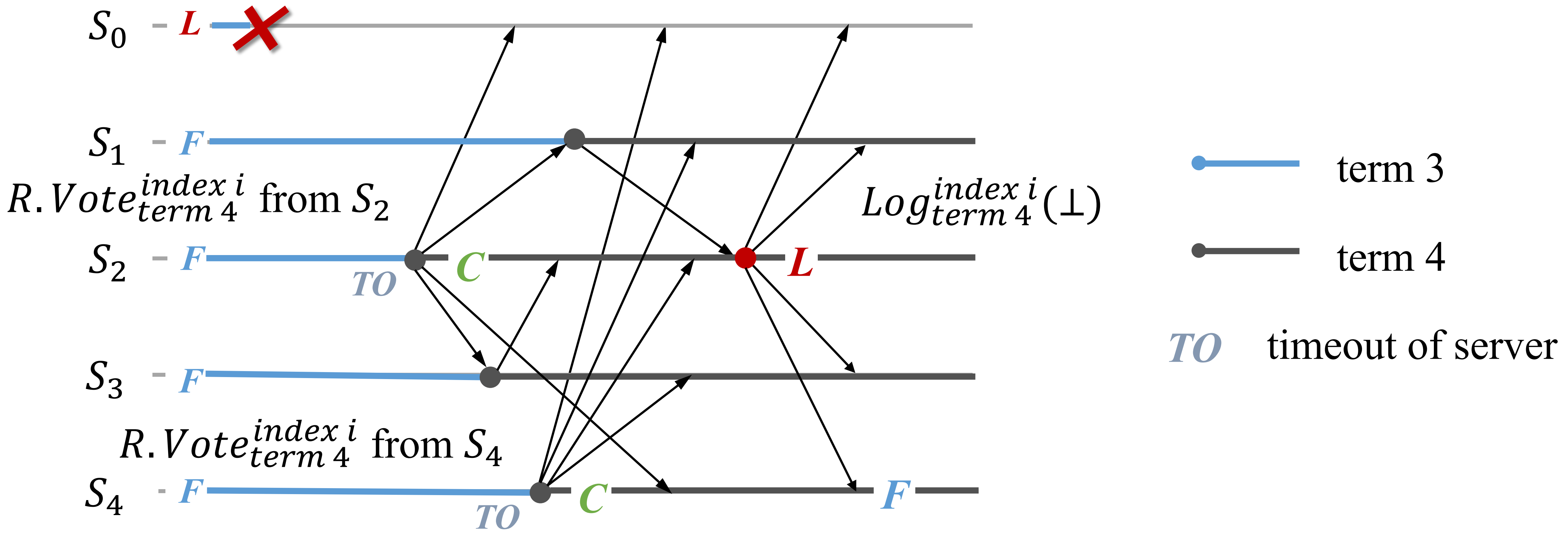
Terms are numbered with consecutive integers.

Raft ensures that there is at most one leader in a given term.

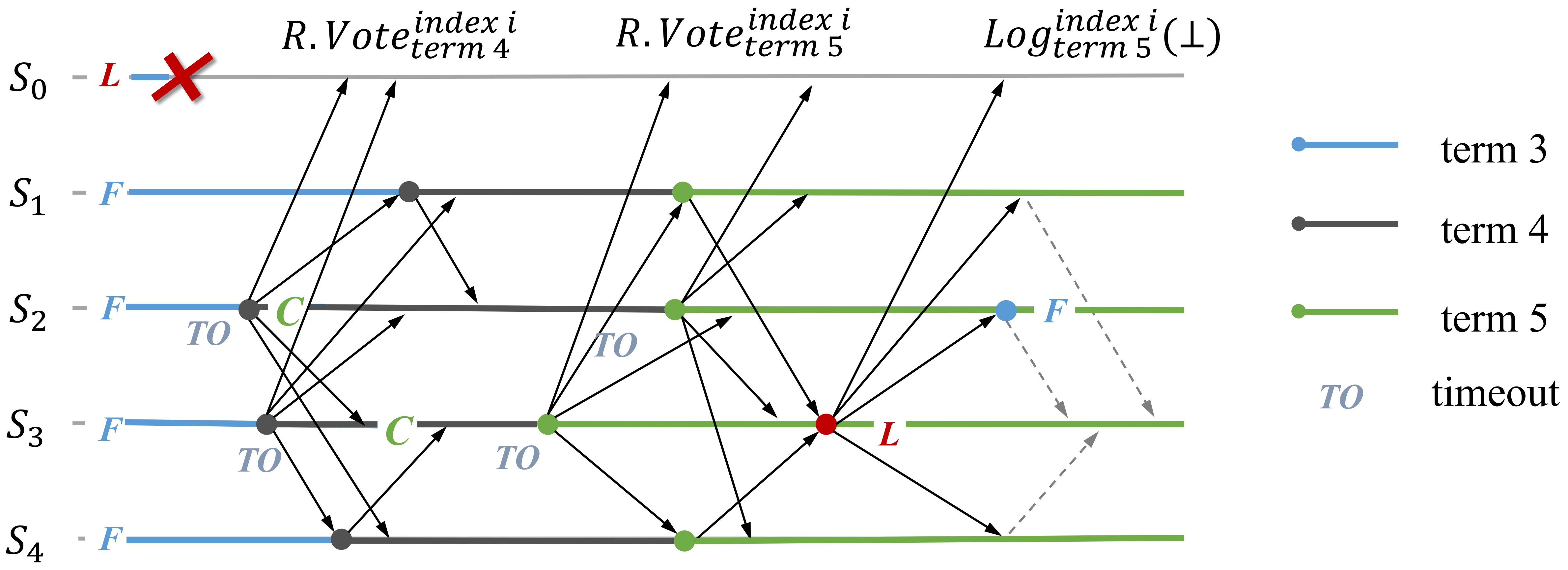
In a server's dimension



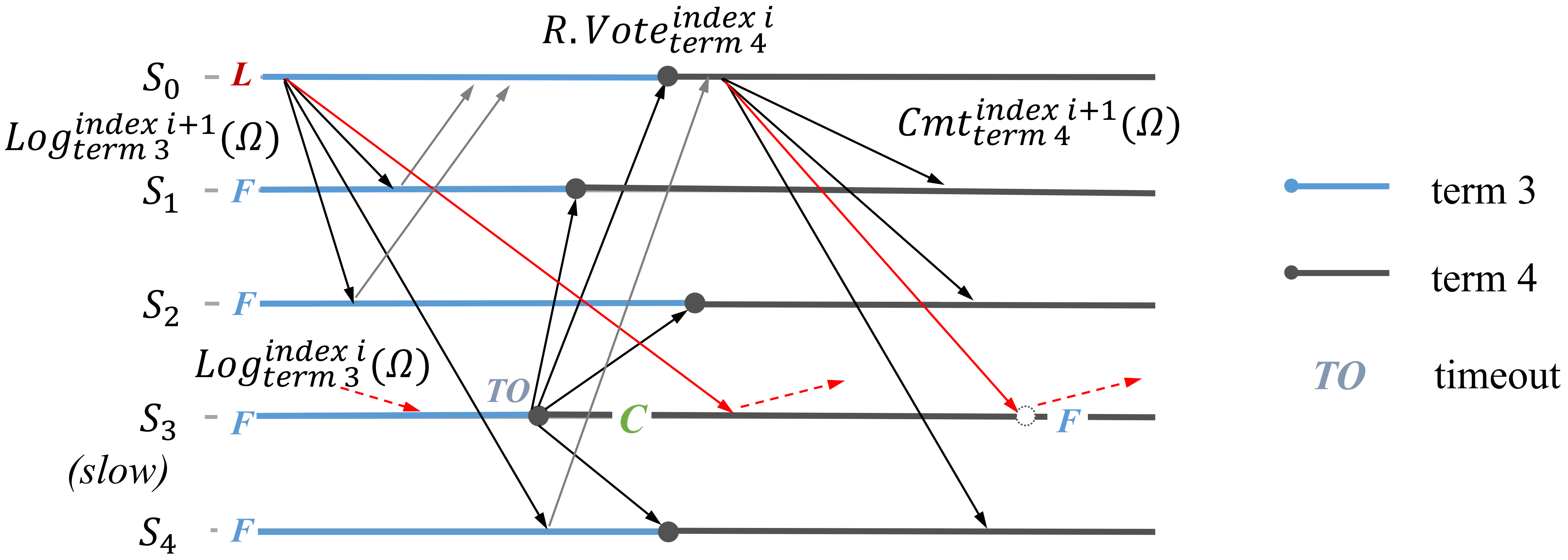
Case 1 => Two candidates with the same term



Case 2 => Two candidates with split votes



Case 3 => An election started by a slow node



Learn more on …

[10] Howard H. ARC: analysis of Raft consensus[R]. University of Cambridge, Computer Laboratory, 2014.

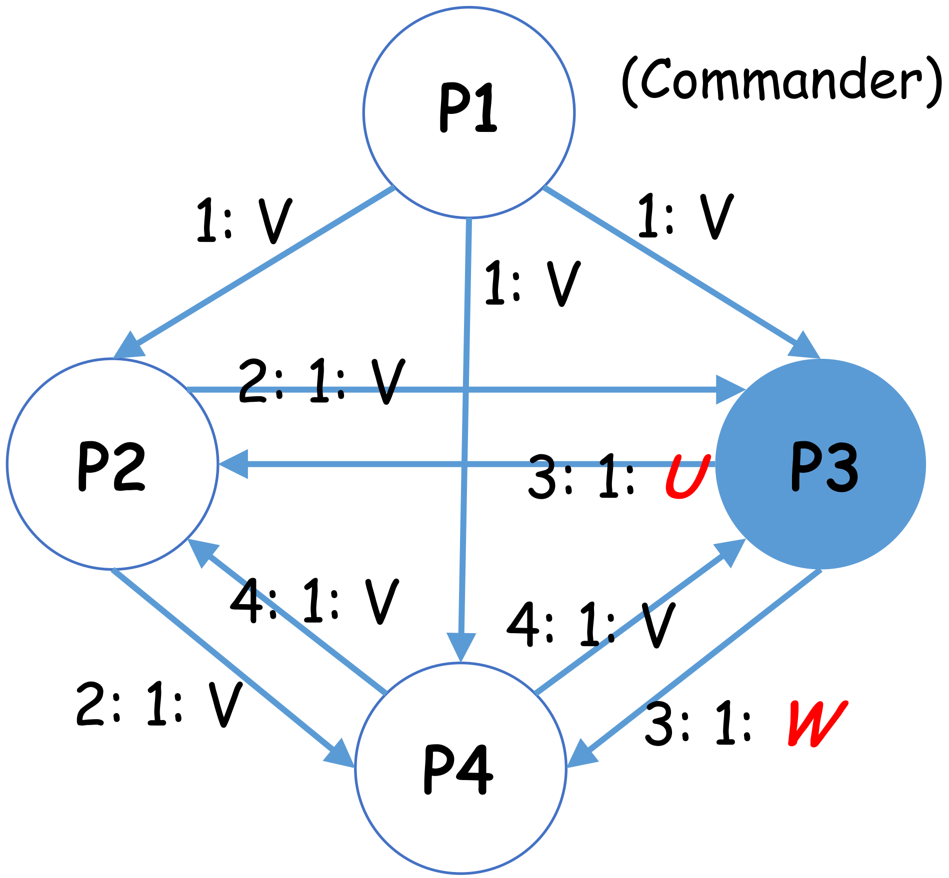
[11] Howard H, Schwarzkopf M, Madhavapeddy A, et al. Raft refloated: do we have consensus?[J]. ACM SIGOPS Operating Systems Review, 2015, 49(1): 12-21.

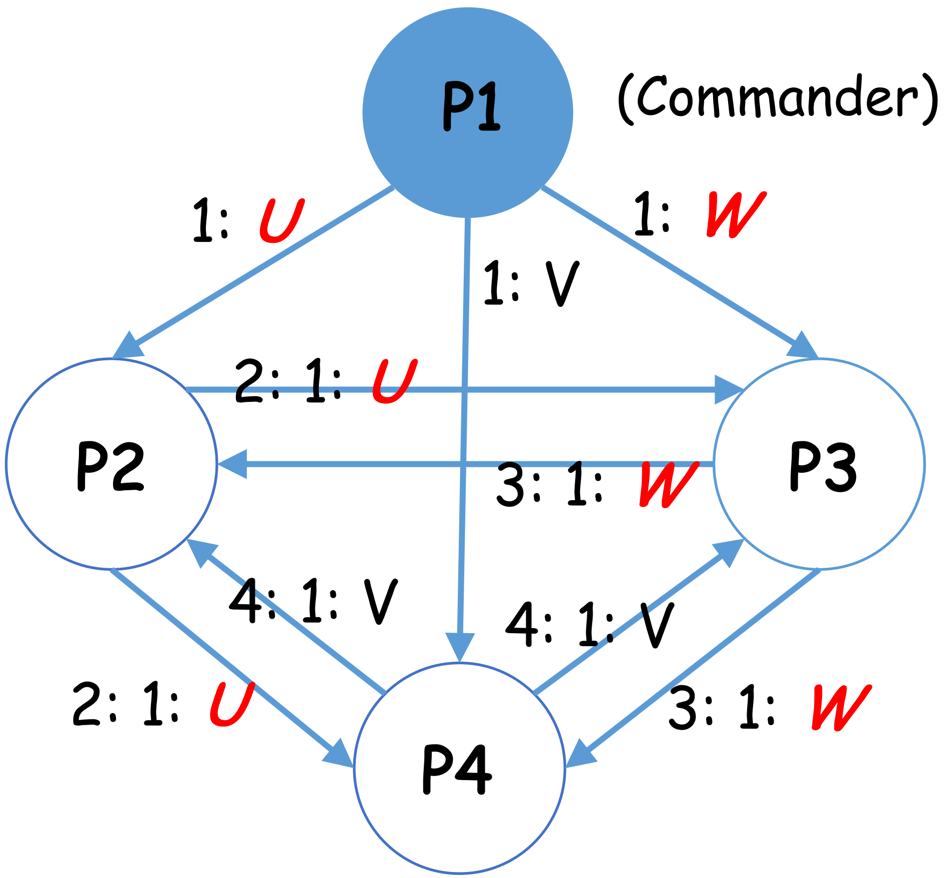
[12] Woos D, Wilcox J R, Anton S, et al. Planning for change in a formal verification of the Raft consensus protocol[C]//Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs. ACM, 2016: 154-165.

[13] Wilcox J R, Woos D, Panchekha P, et al. Verdi: a framework for implementing and formally verifying distributed systems[C]//ACM SIGPLAN Notices. ACM, 2015, 50(6): 357-368.

[14] Evrard H, Lang F. Automatic distributed code generation from formal models of asynchronous concurrent processes[C]//Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on. IEEE, 2015: 459-466.

Byzantine Condition => Assume that processes can exhibit arbitrary failures.





P2 decides on majority(*V*, *U*, *V*) = *V*

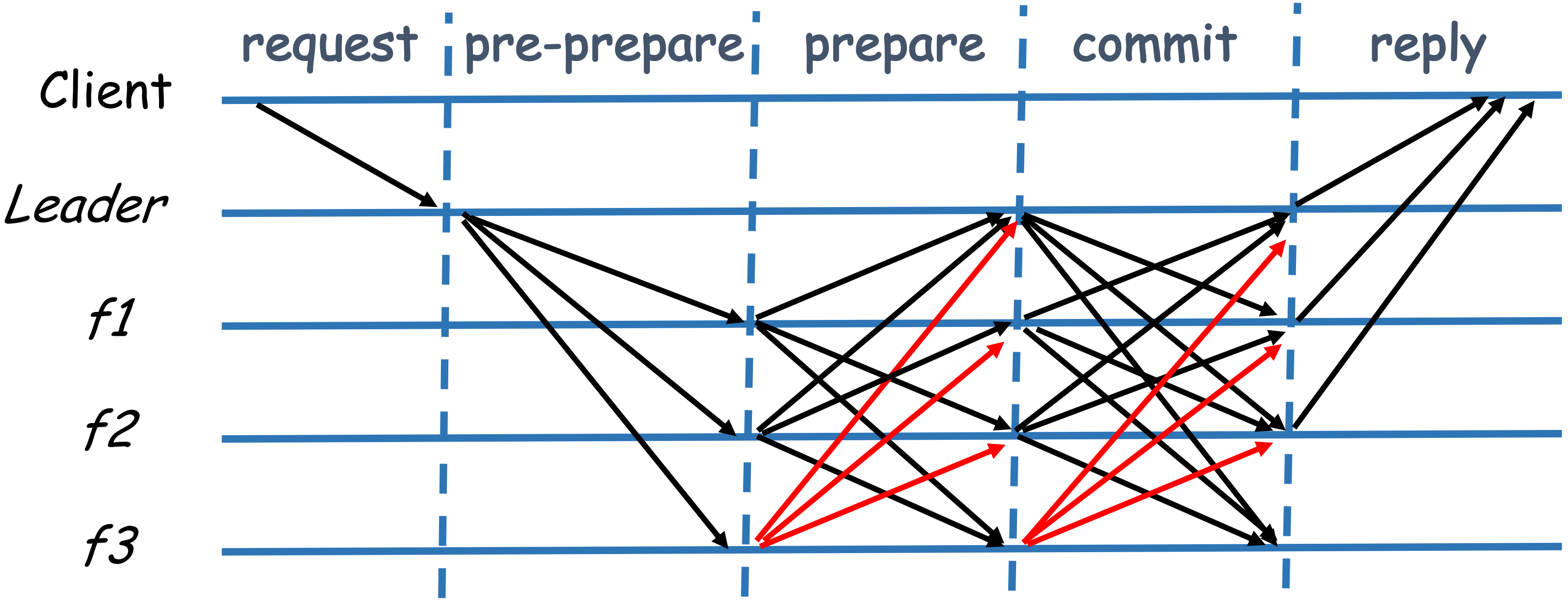
P4 decides on majority(*V*, *V*, *W*) = *V*

P2, P4 decides on majority(*V*, *U*, *W*) = ∅

(no majority values exists)

PBFT: tolerant Byzantine failures with 3f+1 nodes

- A client sends a request to invoke a service operation to the primary.
- The primary multicasts the request to the backups.
- Replicas execute the request and send a reply to the client.
- The client waits for *f+1* replies from different replicas with the same results; this is the result of the operation.



Learn more on …

[15] Lamport L, Shostak R, Pease M. The Byzantine generals problem[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1982, 4(3): 382-401.

[16] Schneider F B. Byzantine generals in action: Implementing fail-stop processors[J]. ACM Transactions on Computer Systems (TOCS), 1984, 2(2): 145-154.

[17] Veronese G S, Correia M, Bessani A N, et al. Efficient byzantine fault-tolerance[J]. IEEE Transactions on Computers, 2013, 62(1): 16-30.

[18] Castro M, Liskov B. Practical Byzantine fault tolerance[C]//OSDI. 1999, 99: 173-186.

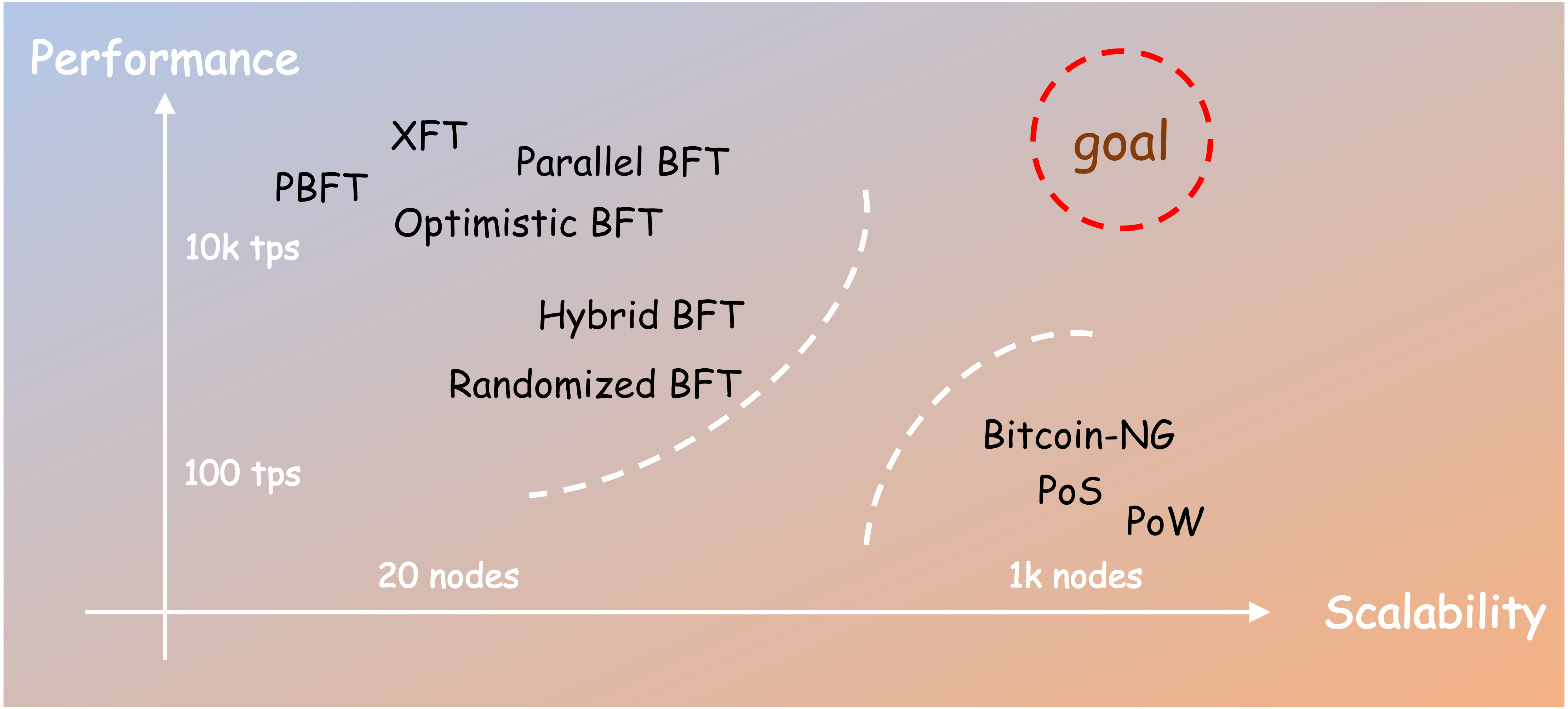
[19] Liu S, Viotti P, Cachin C, et al. XFT: Practical Fault Tolerance beyond Crashes[C]//OSDI. 2016: 485-500.

[20] Miller A, Xia Y, Croman K, et al. The honey badger of BFT protocols[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2016: 31-42.

Some High-level Comparisons

	Proof-Of-Work	Repli. StateM. / BFT based protocols
Node identity management	Open, entirely decentralized	Permissioned, nodes need to know IDs of all other nodes
Consensus finality	no	yes
Throughput	Limited (due to possible chain forks)	Good (tens of thousands tps)
Scalability	Excellent (like Bitcoin)	Limited (not well explored)
Latency	High latency (due to multi-block confirmations)	Excellent (effected by network latency)
Power consumption	Poor (useless hash calculations)	good
Network synchrony assumptions	Physical clock timestamps	None for consensus safety
Correctness proofs	no	yes

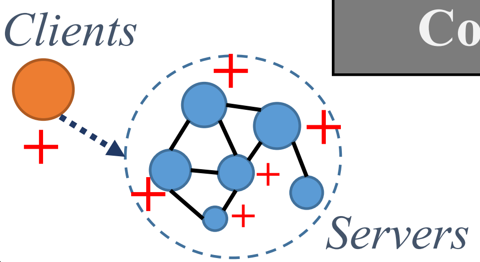
Performance and Scalability



Blockchain as a Service (BAAS)

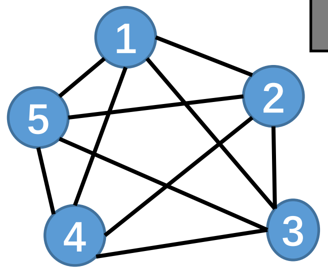
—Smart Contracts and Blockchain 2.0

Consensus



I. State machine based protocols
II. Proof-Of-X based protocols
III. BFT based protocols

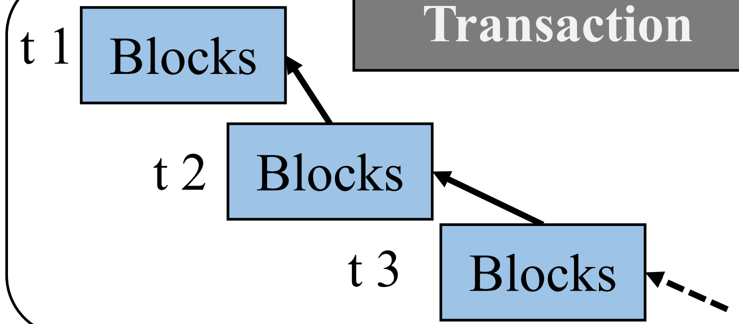
Network



graphs

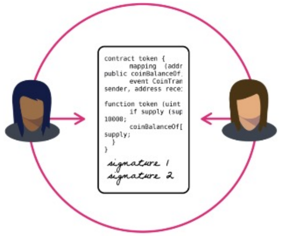
I. Communication protocols
II. Finite number of nodes
III. Node management
IV. Validation

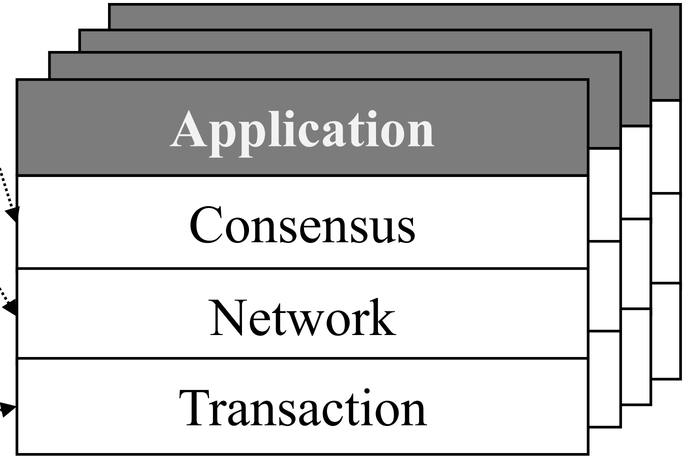
Transaction



I. Chain structure
II. Encrypted data
III. Block header
IV. Logic time stamp
V. Signatures

- Smart Contract
- Cooperation
- …





Applications

Blockchain based Applications
(Used car trading model, Real estate registration)

Blockchain as a Service (BaaS)

Blockchain framework
"Consensus Algorithm",
"Data Structure"

Digital Content Protection
"Blockchain based"

Thanks for listening!

UNIVERSITY OF TORONTO

Gengrui (Edward) Zhang
Email: gengrui.zhang@mail.utoronto.ca
Web: https://gengruizhang.github.io