Untangling Blockchain Consensus Protocols from Blockchain 1.0 to 2.0

Gengrui Zhang

Tencent 腾讯

Content

Motivatioin

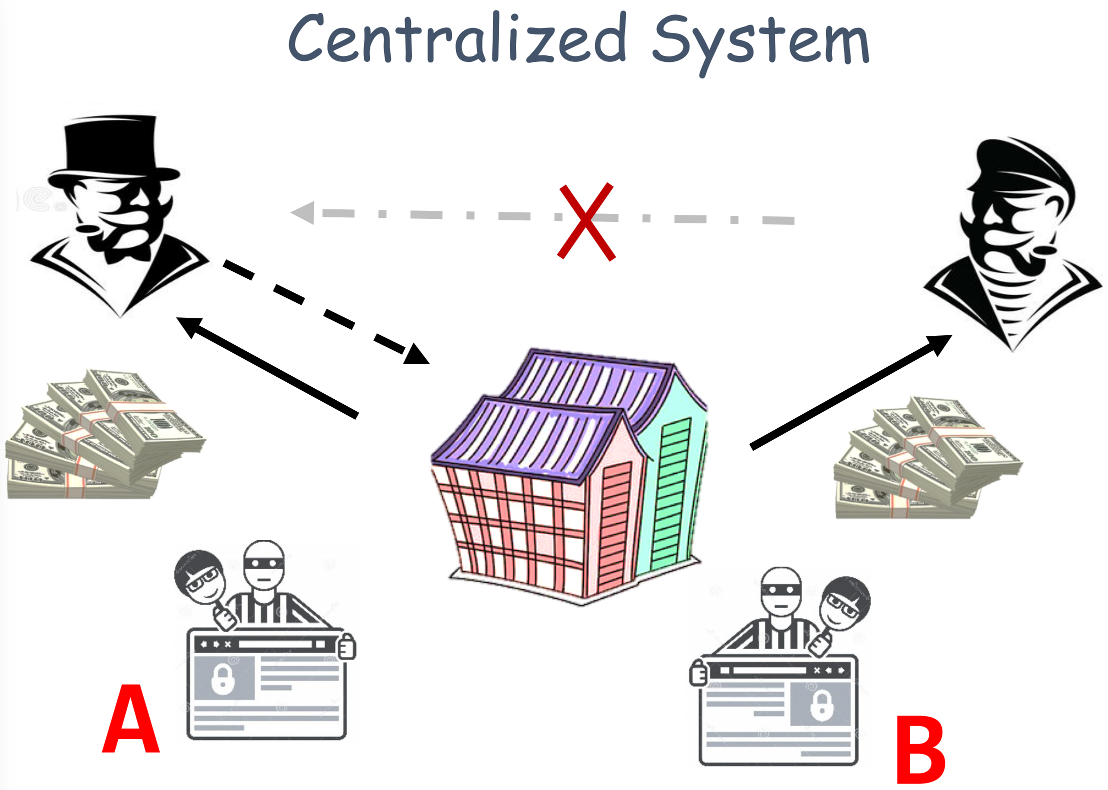
Background

Design & Implementation

Evaluation

- Why do we need a blockchain?
- The consensus protocols: PoW, PoS, Paxos, Raft, PBFT
- The problem with Paxos, Raft
  - Dynasty and D-Chain
- Comparisons and Evaluation

Why do we need a blockchain?



Tencent 腾讯

一处房产多次抵押 兰州夫妇骗贷双双被捕

2014-11-24 08:45:

一处房产多

每日甘肃网
资金，直到债主
检察院以犯罪嫌

2012年11月
介绍信等资料，
快为罗某办理了
套住宅抵押给担
丈夫王某商议出
司看到了罗某夫
终双方以30万元
生约王某夫妇一起

同一房产汽车多次抵押借款

男子涉嫌诈骗罪被批准逮捕

0

来源：中

中山日报7月9日讯
次抵押同一房产、同
据办案检察官介绍
房产已被惠州市惠
法院申请查封，期
2014年9月2日，
万元，并签订抵押
万元。

警方称涉案楼盘有多次抵押现象

-27 15:32:15 来源：羊城晚报

广告:个股明日走势预测

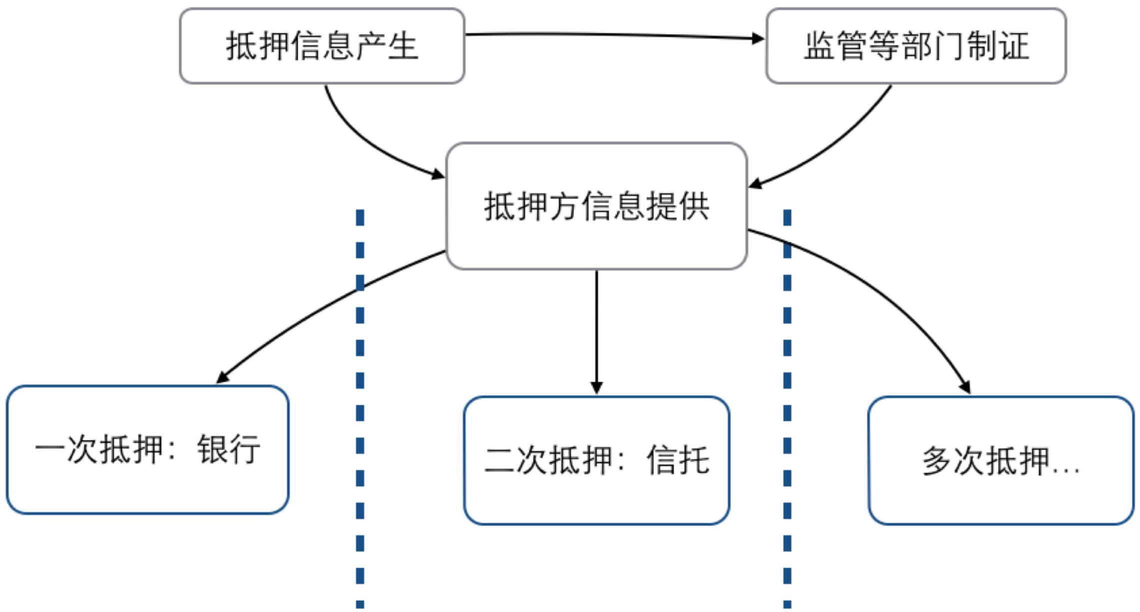
用假房产证作抵押 诈骗他人67万获刑

发布时间：2017-02-18 19:59:37 来源：重庆法制报 大 中 小

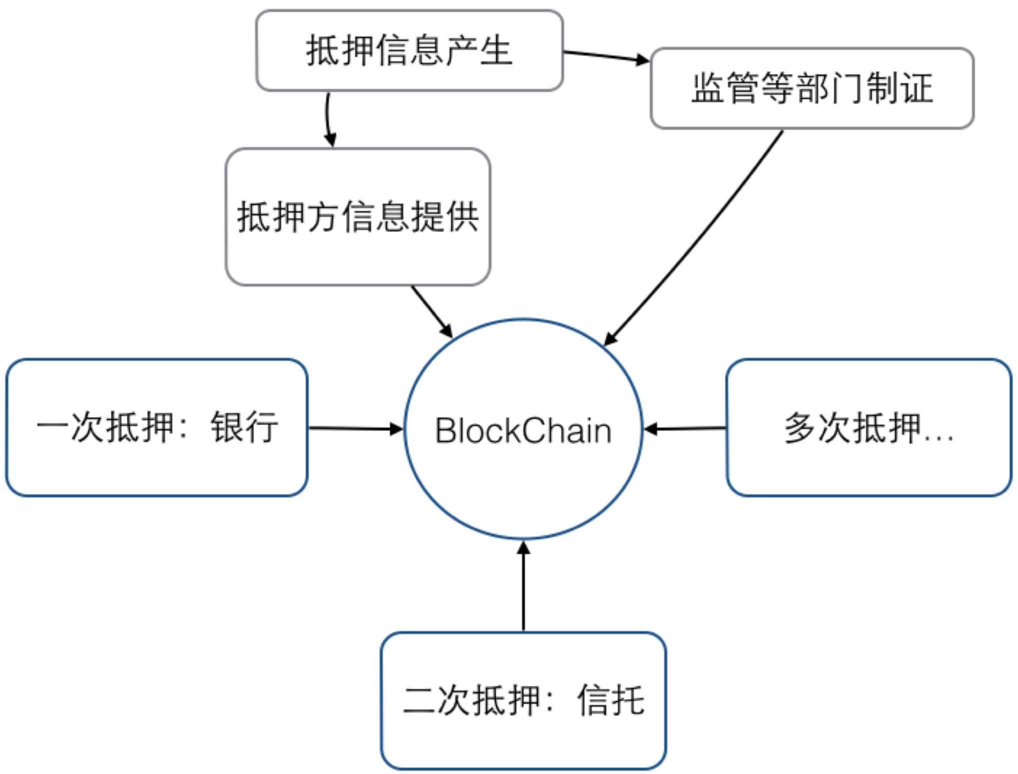
本报讯(通讯员 陈 云)欠下赌债无力归还，办理假房产证作抵押骗取他人财物。近日，綦江法院以诈骗罪判处被告人李某有期徒刑7年，并处罚金，责令被告人李某退赔被害人67万余元。

2013年，被告人李某因赌博欠下债务。因无力归还巨额债务及高额利息，编造做水泥生意需周转资金为由，以假房产证作为抵押，先后骗取杨某某、张某某钱财。2013年10月，李某再次向杨某某提出借款50万元时，因之前的借款未还清，故杨某某不愿

We do need a blockchain that …



Decentralized System



“去中心化的，多方决策，集体维护的可信分布式账本”

区块链是一种以**密码学**算法为基础的点对点**分布式账本**技术，其本质是一种**互联网共享数据库**。

⇩ 非许可类区块链（Permissionless）
“公有链”

向全网络公开，无需节点管理、审核，可任意加入、退出。

⇩ **Proof-of-X**

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

⇩ 许可类区块链（Permissioned）
“联盟链，私有链”

面向合作，需节点管理，各节点需全局地址记录。

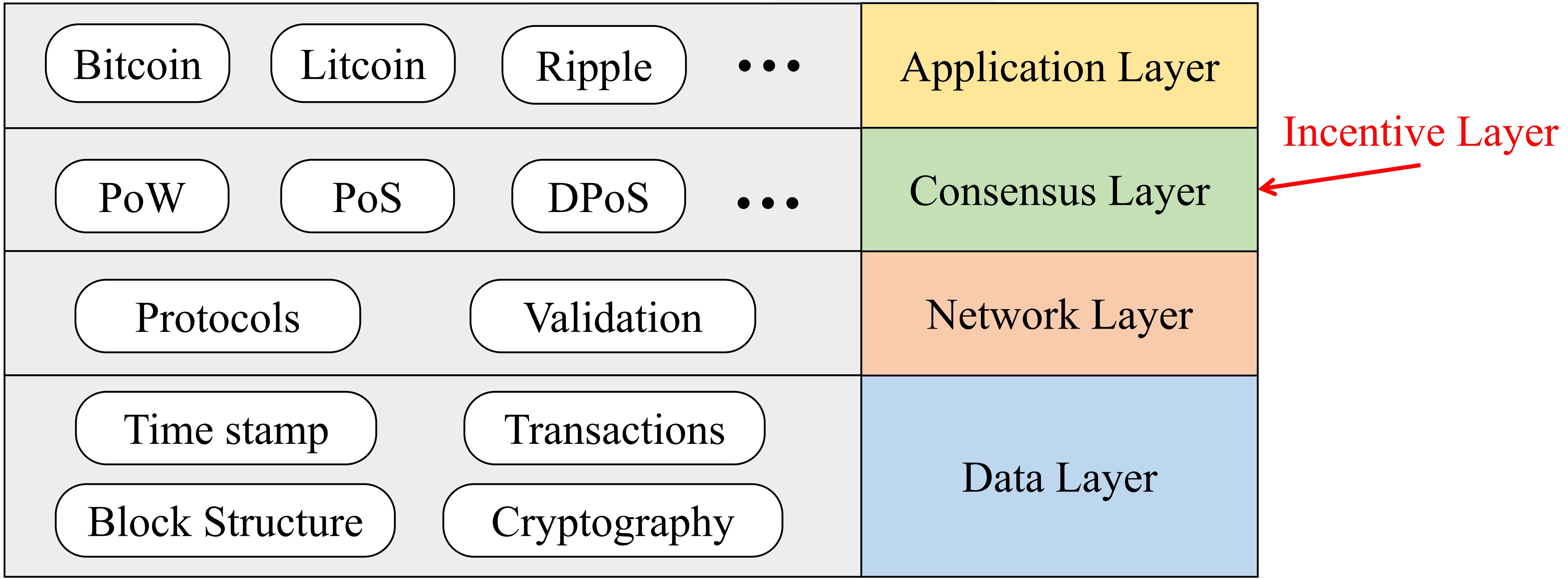
⇩ **Replicated State Machine, Repl.SM**
Byzantine Fault Tolerance, BFT

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

⟷ 混合链

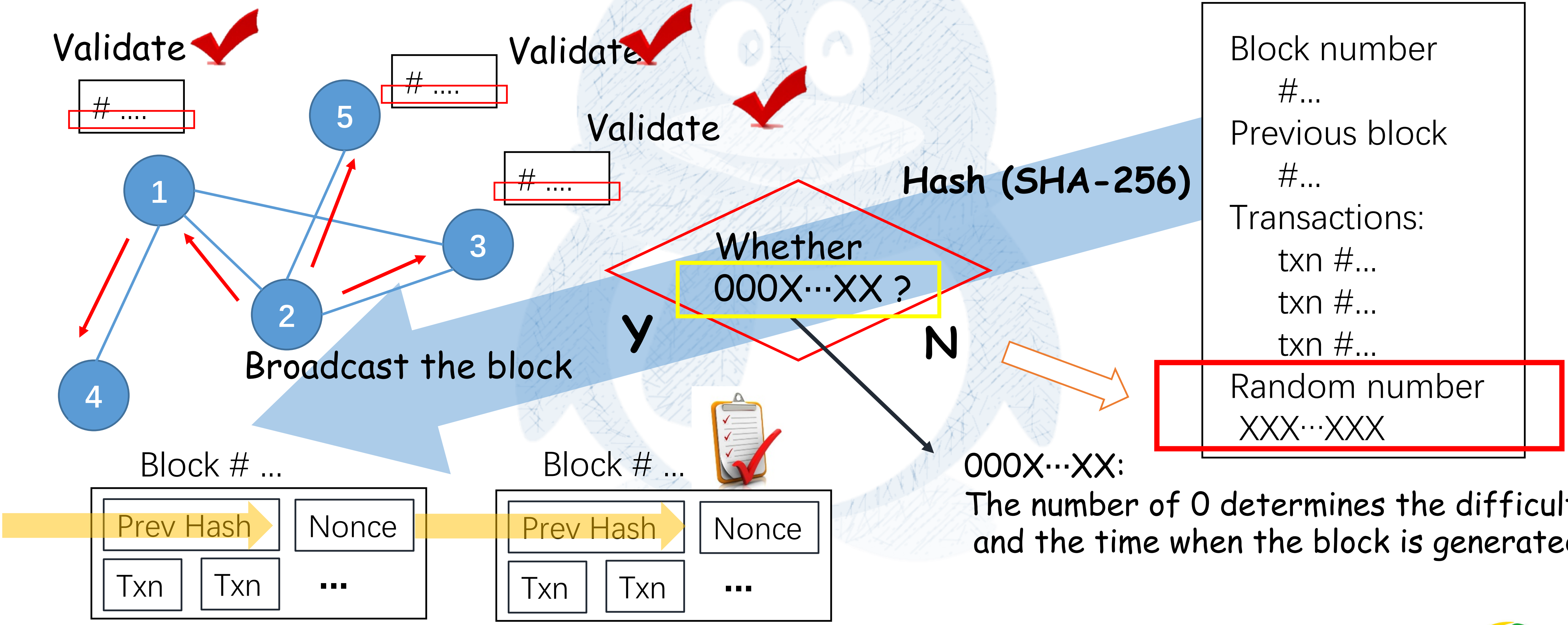
Permissionless Blockchains

—Somehow named as Blockchain 1.0

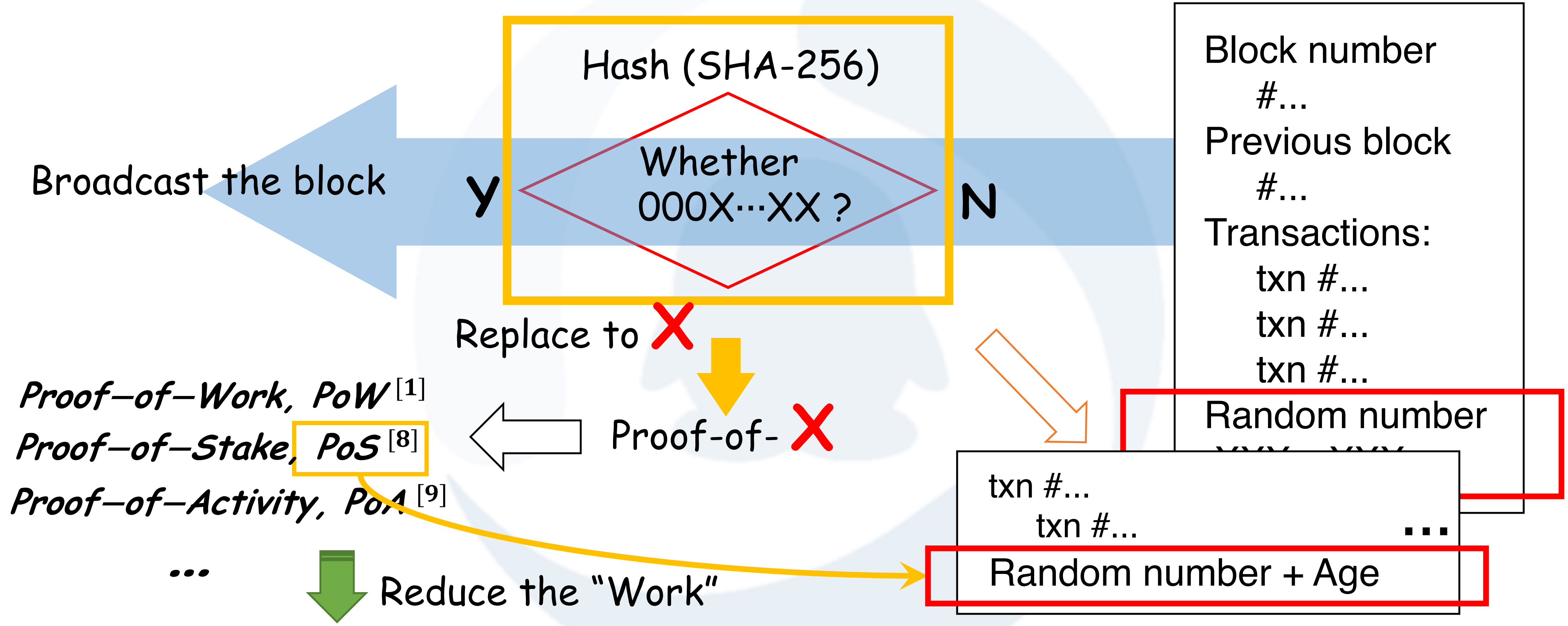


Proof-of-Work, PoW

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



Protocols for Permissionless Blockchains

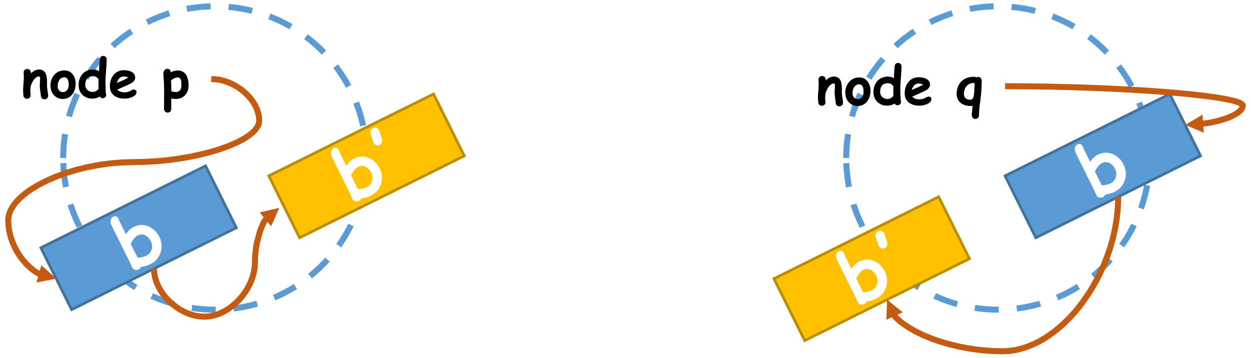


[2] King S, Nadal S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake[J]. self-published paper, August, 2012, 19.

Consensus Finality

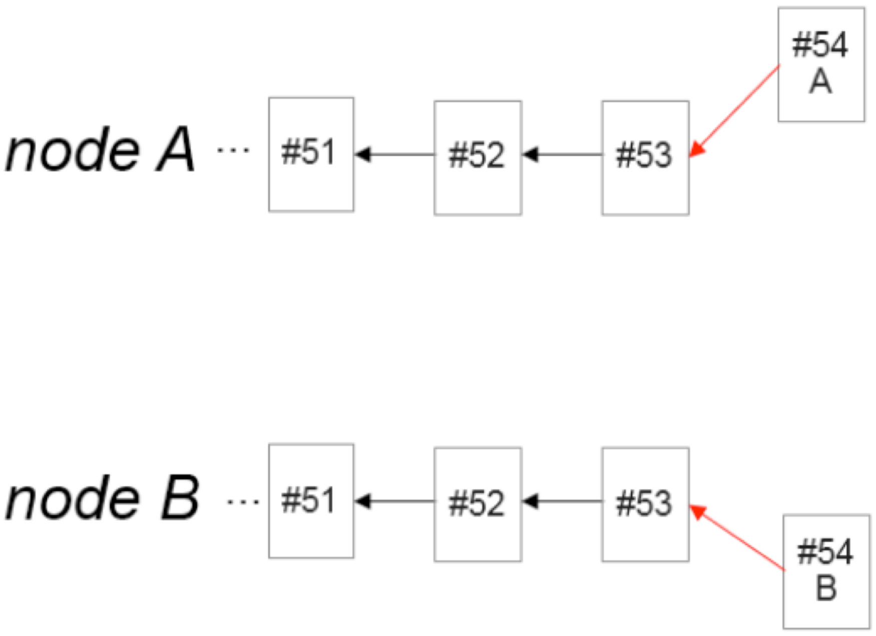
[3] Vukolić M. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication[C]//International Workshop on Open Problems in Network Security. Springer, Cham, 2015: 112-125.

- ✓ If a correct node **p** appends block **b** to its copy of the blockchain before appending block **b'**, then no correct node **q** appends block **b'** before **b** to its copy of the blockchain.

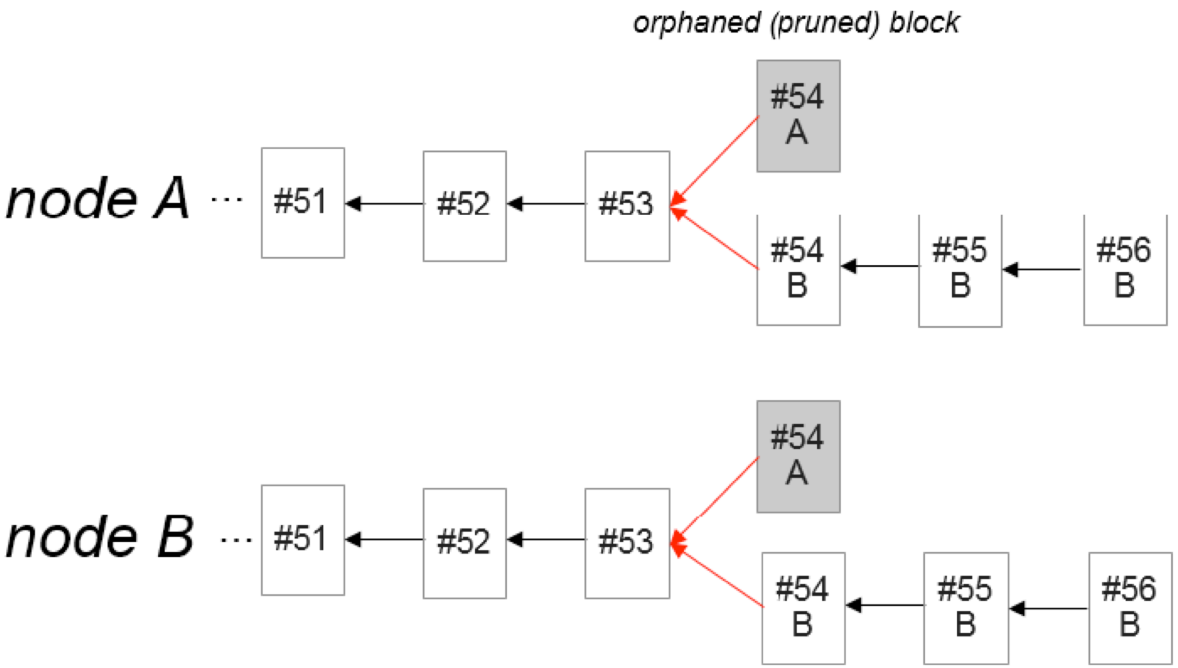


Double-Spending / Chain-forks

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

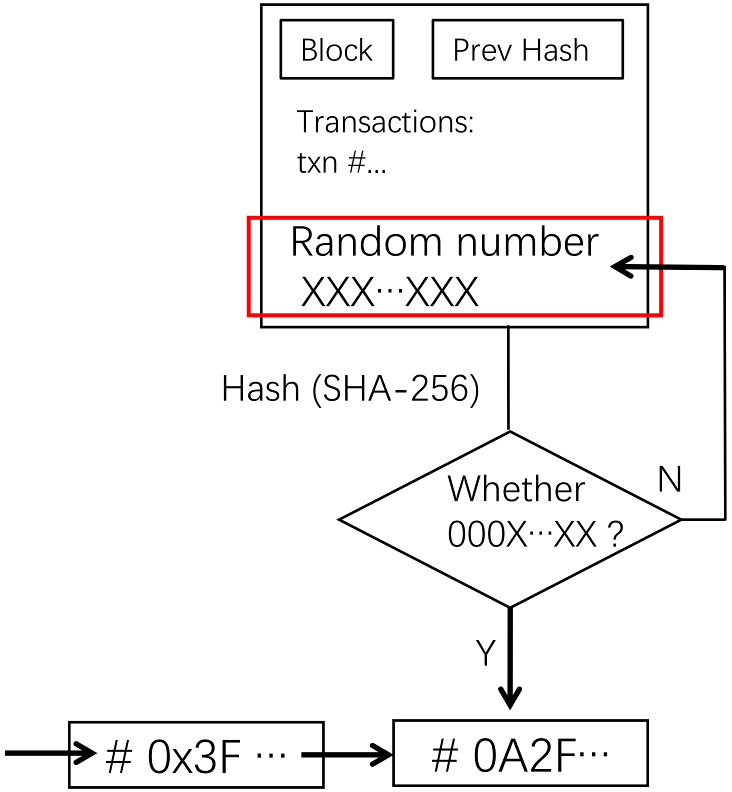


(a) Consensus finality violation resulting in a fork



(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

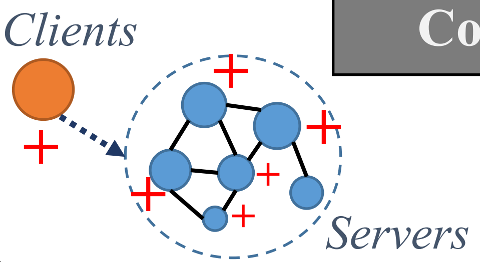
Ripple

Ethereum

Sawtooth Lake

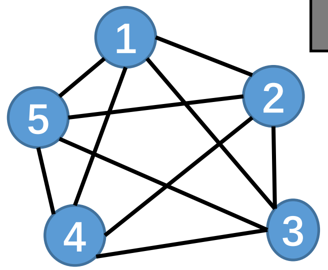
Permissioned Blockchain —Smart Contracts and Blockchain 2.0

Consensus



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

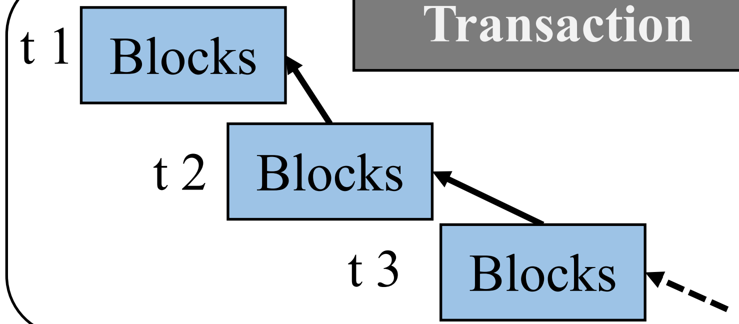
Network



graphs

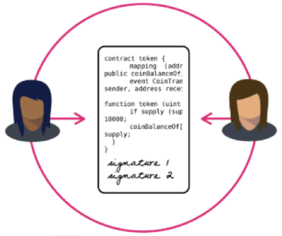
I. Communication protocols
II. Finite number of nodes
III. Node management
IV. Validation

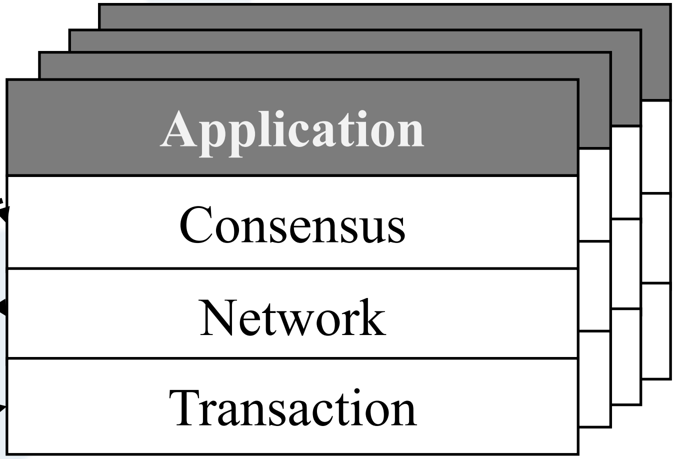
Transaction



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

- Smart Contract
- Cooperation
- …





Permissioned Blockchain

↓

Coordination and Agreement in distributed system

↓

Interactive consistency | Consensus | Byzantine generals

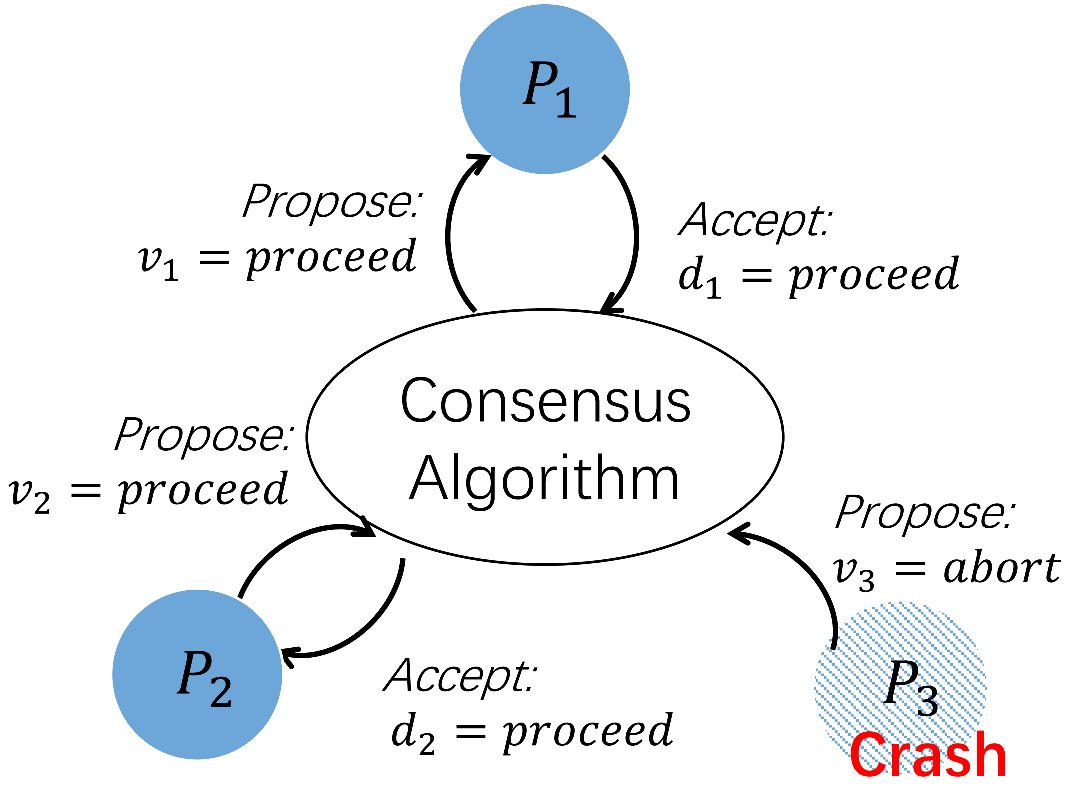
"decision vector" | "crash, omissions" | "arbitrary failures"

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

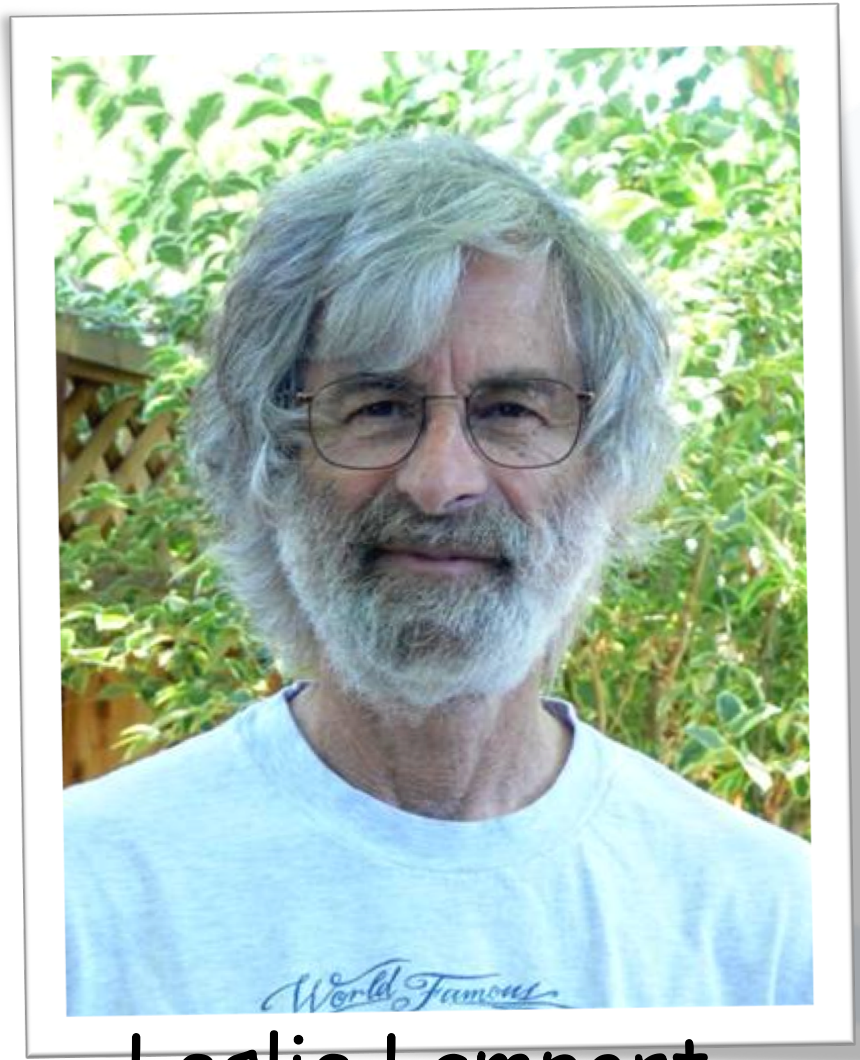
⇒ Paxos:
How to choose a value?

⇒ Raft:
How to replicate a log?

⇒ PBFT:
How to guarantee the correctness under Byzantine conditions?

stronger assumption

stronger assumption



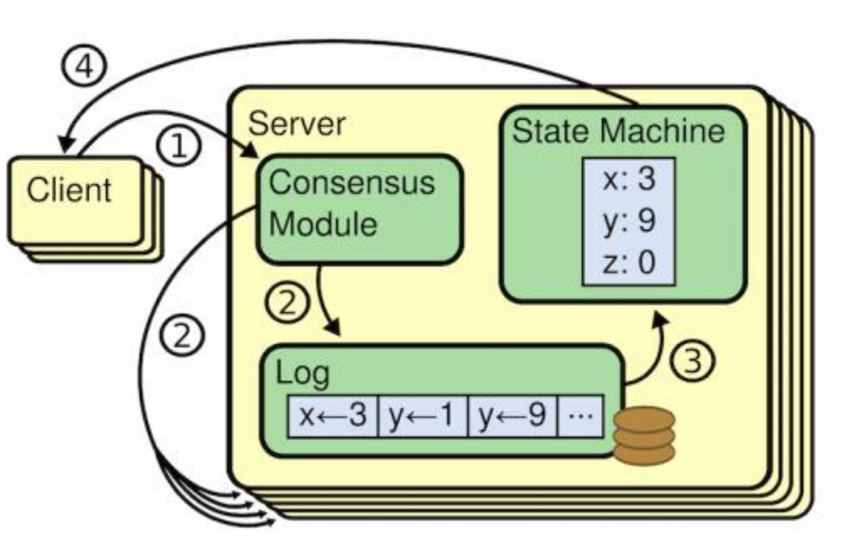
Leslie Lamport

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***

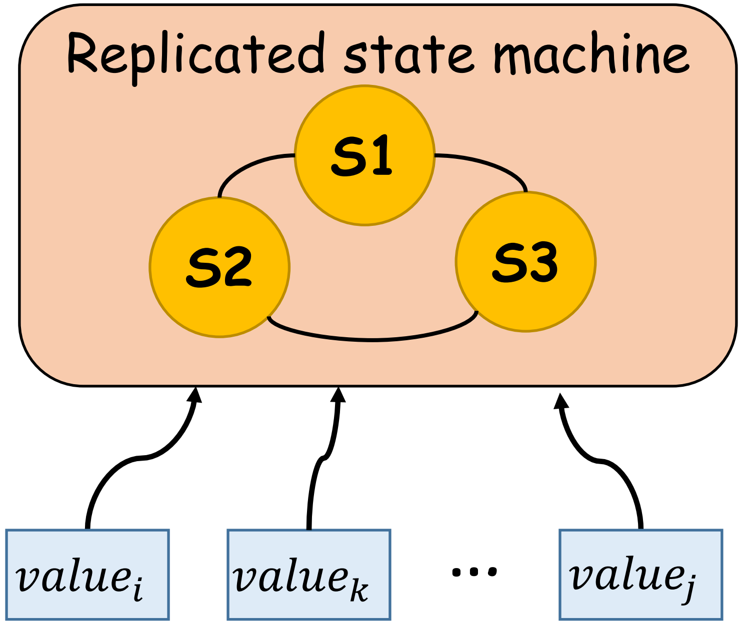
⇩

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



[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.

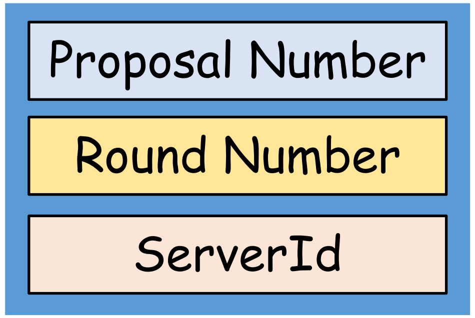
Server
- Proposers
  - -> Active: put forth particular values to be chosen.
  - -> Handle client requests.
- &
- Acceptors
  - -> Passive: respond to messages from proposers.
  - -> Responses represent votes that from consensus.
  - -> Store chosen value, state of the decision process.
  - -> Want to 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.



-> 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.

Putting the actions of the proposer and acceptor together, we see that the algorithm operates in the following two phases.

Phase 1. (Prepare Phase)

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

-> 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.
-> 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*.

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

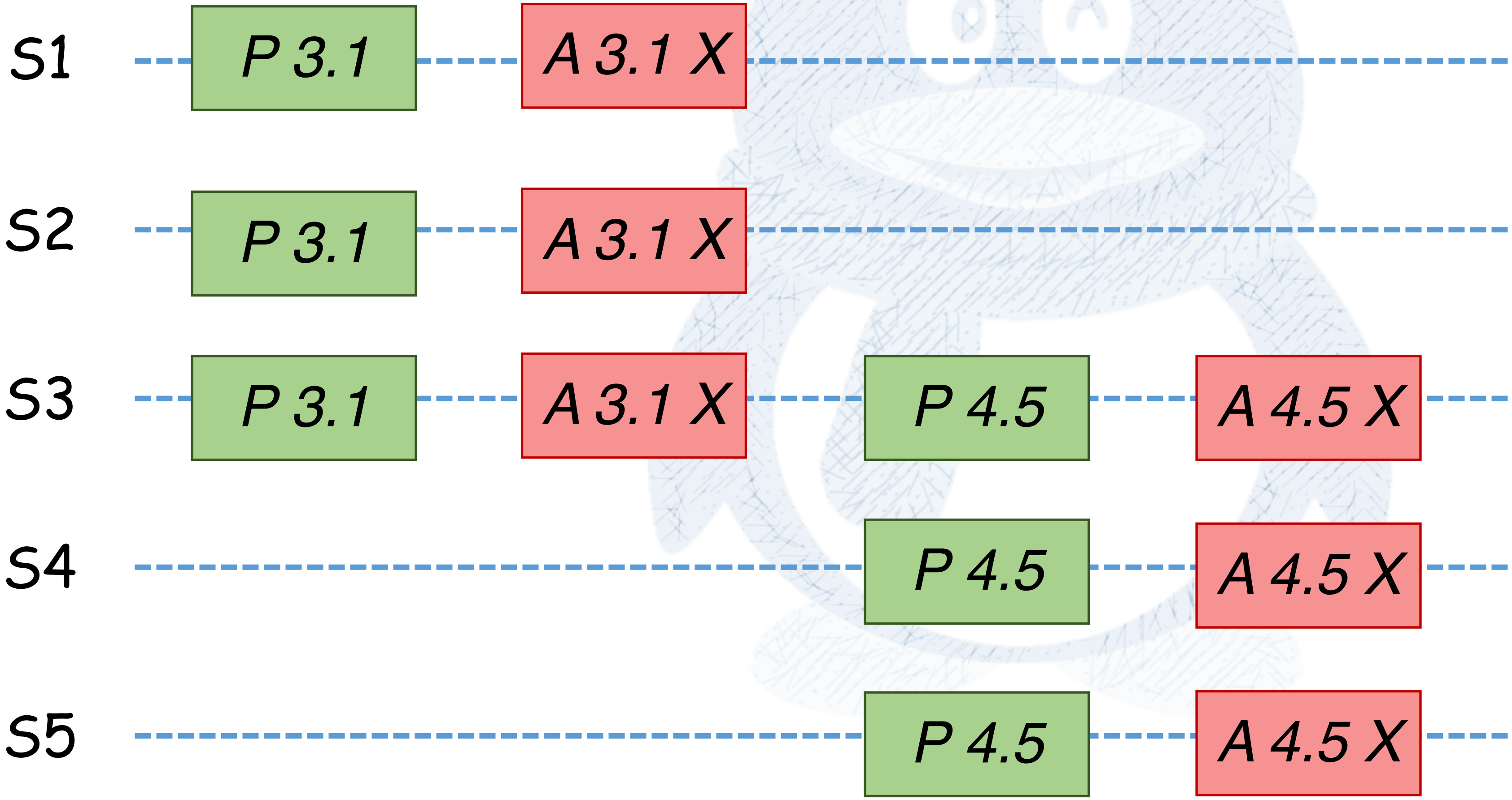
(3) Respond to Prepare(*n*):
-> If n > *minProposal*, then *minProposal* = *n*
-> Return (*acceptedProposal*, *acceptedValue*)

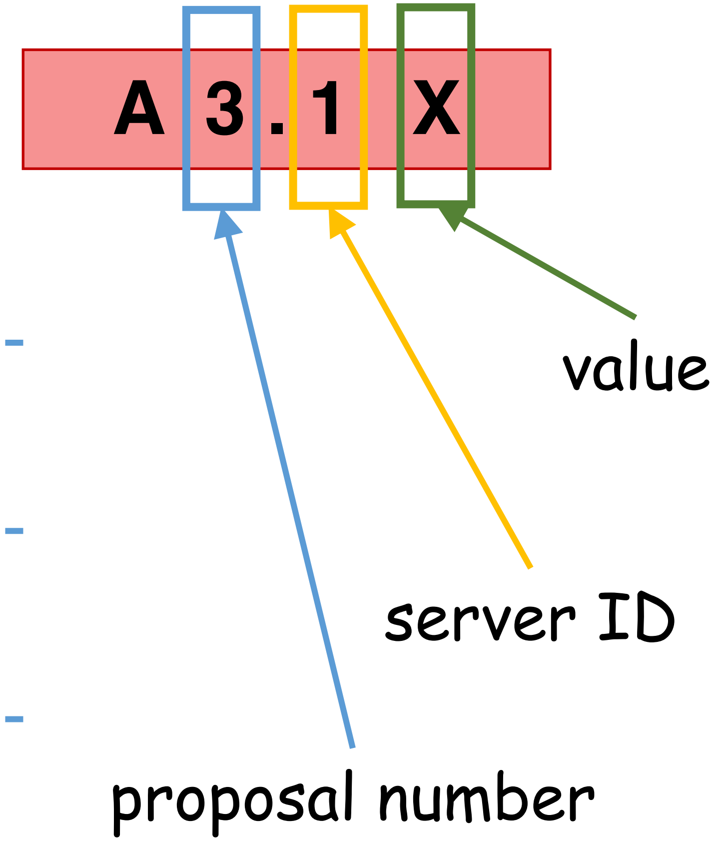
(6) Respond to Accept(*n*, *value*):
-> If *n* >= *minProposal* then
acceptedProposal = *minProposal* = *n*;
acceptedValue = *value*;
-> Return (minProposal)

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

1. Pervious value already chosen

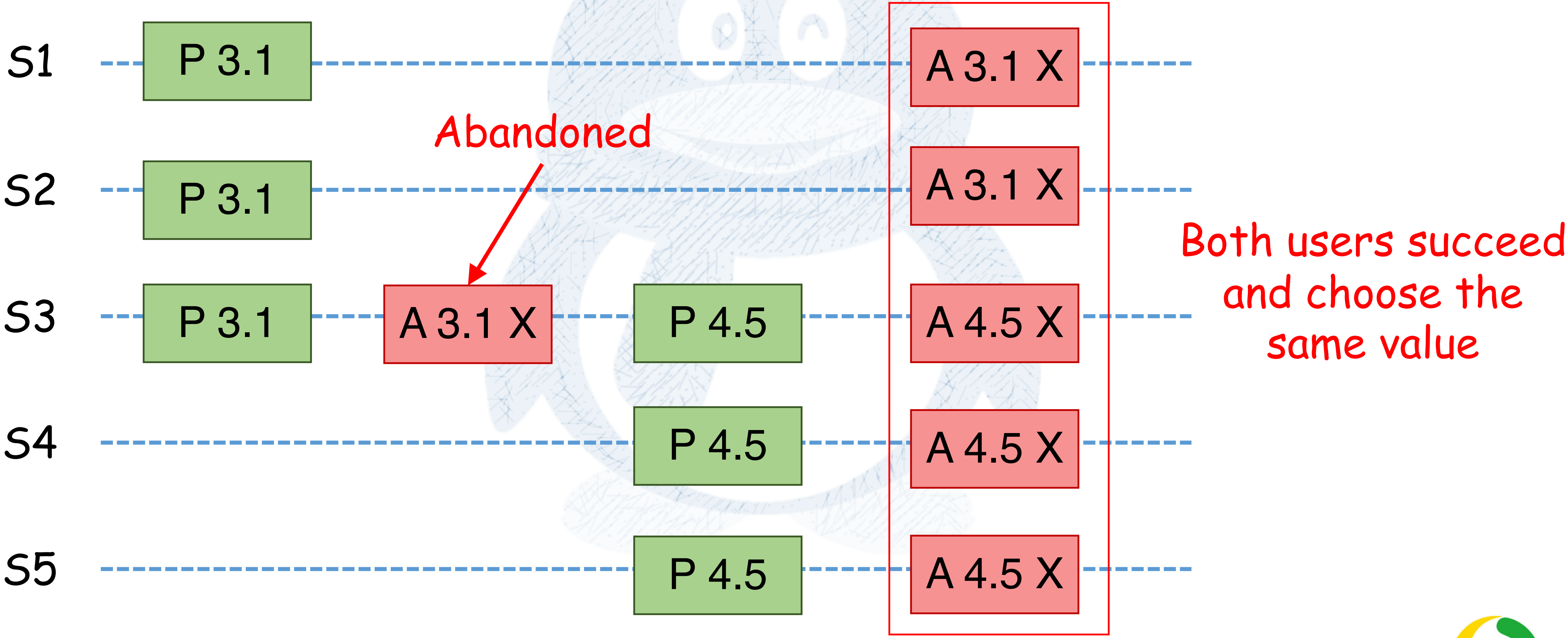
* New proposer will find it and use it





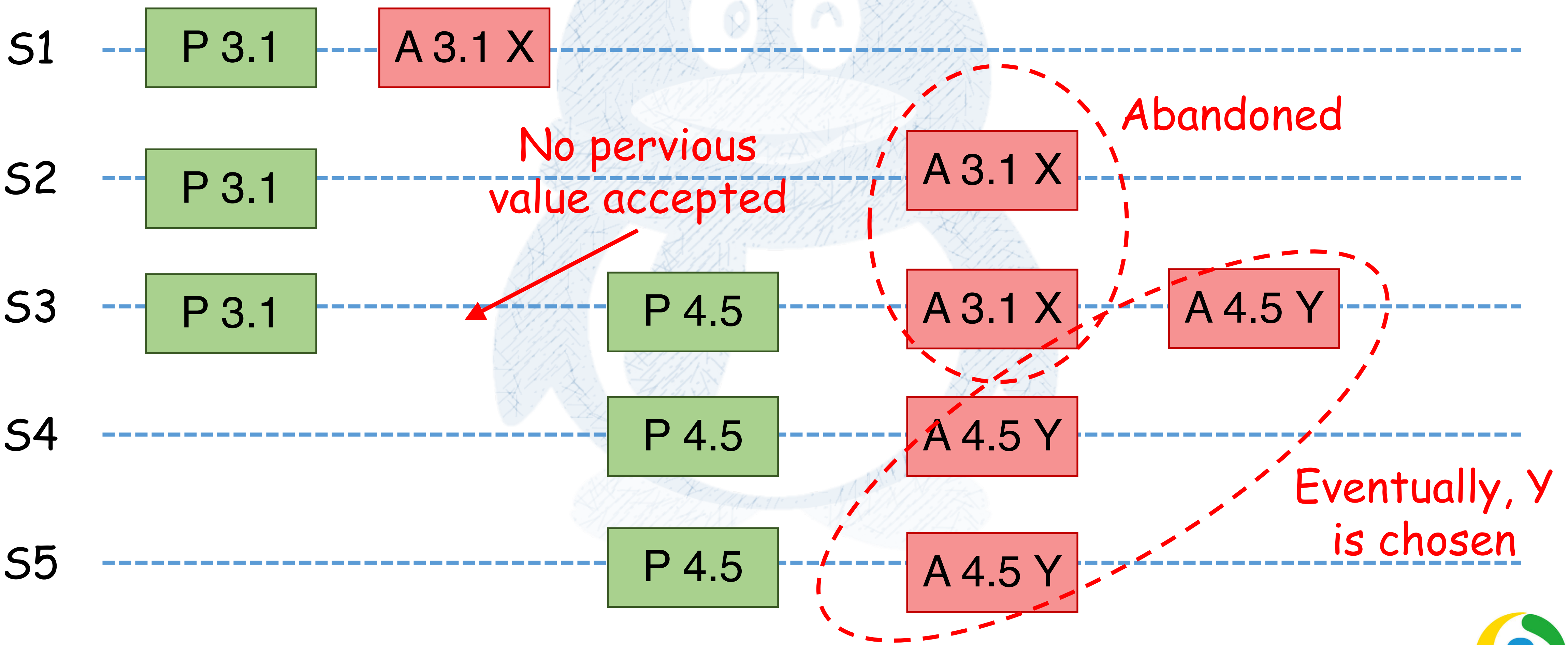
2. Pervious value not chosen, but proposer sees it

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

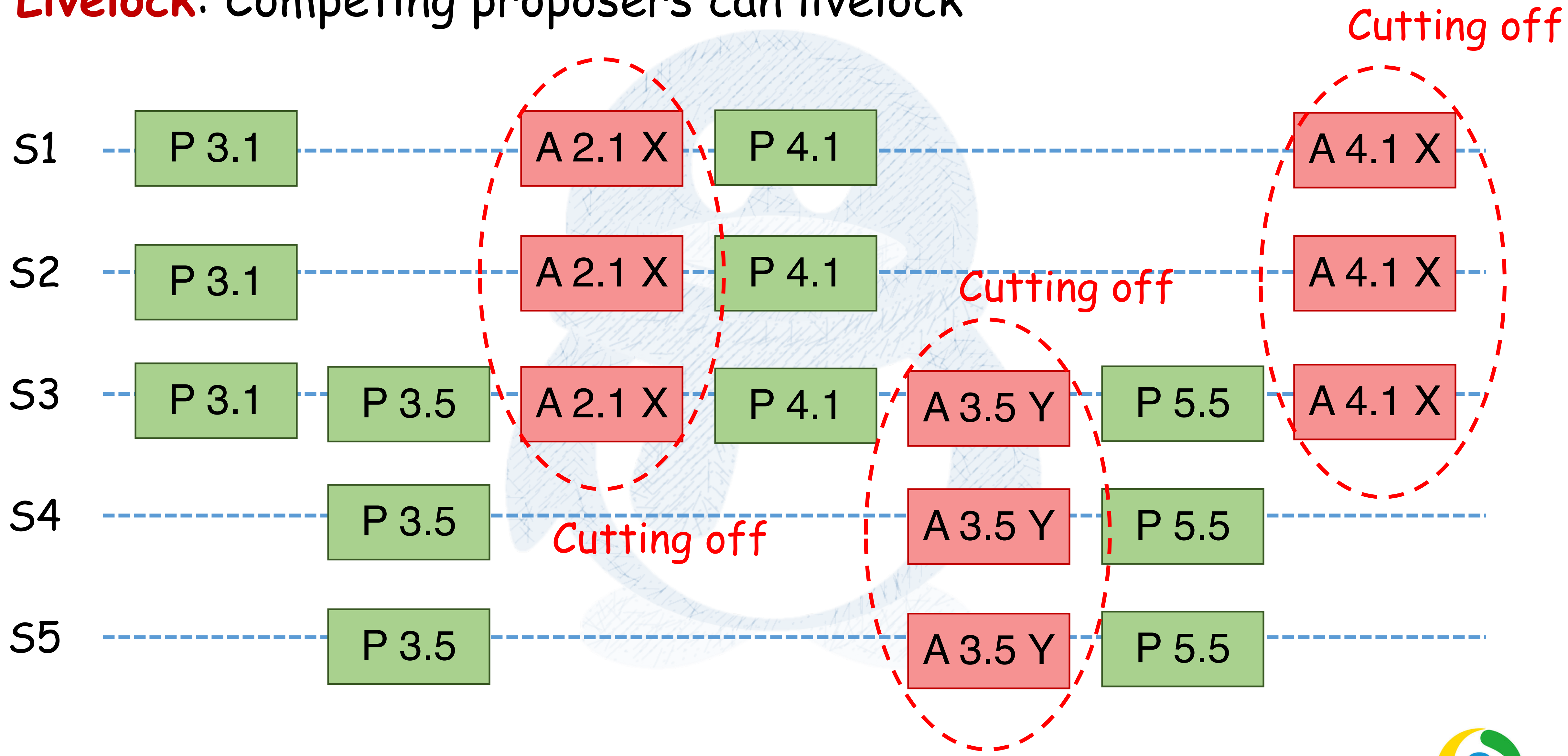


3. Pervious value not chosen, new proposer doesn't see it

- 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.

↓

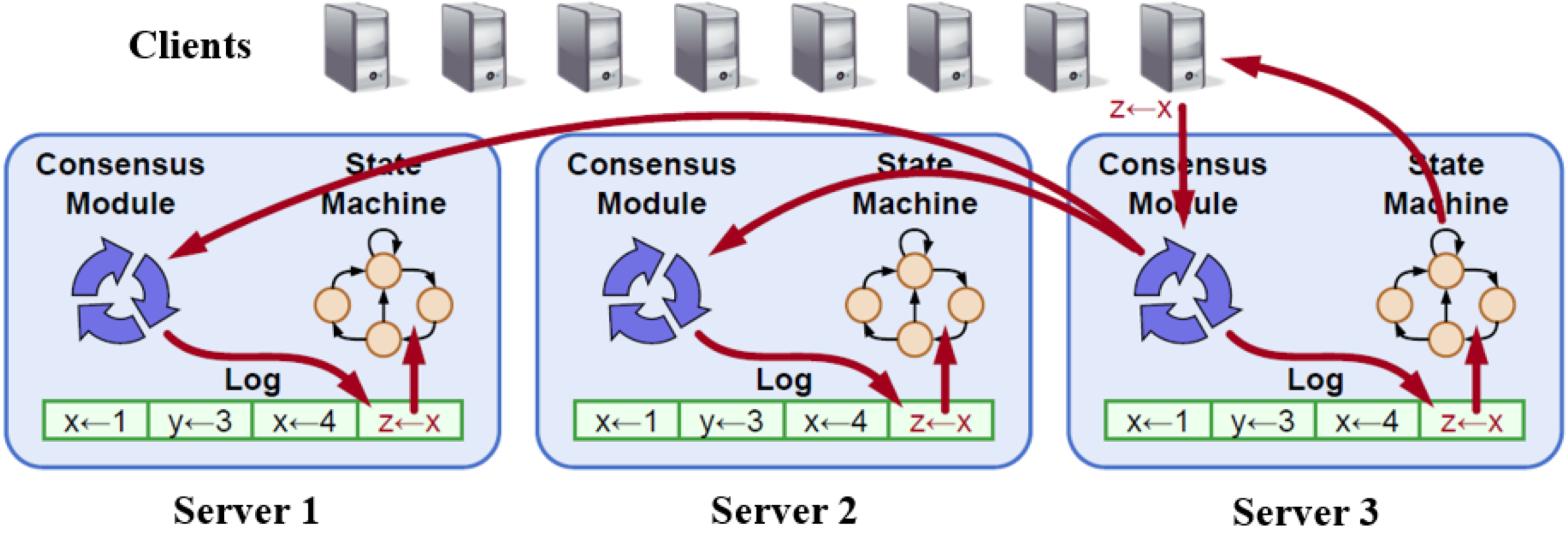
Multi-Paxos, Raft , Zab

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	Candidate	Leader
Followers are passive: they issue no requests on their own but simply respond to requests from leaders and candidates.	The candidate is used to elect a new leader. (using RequestVote RPC)	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

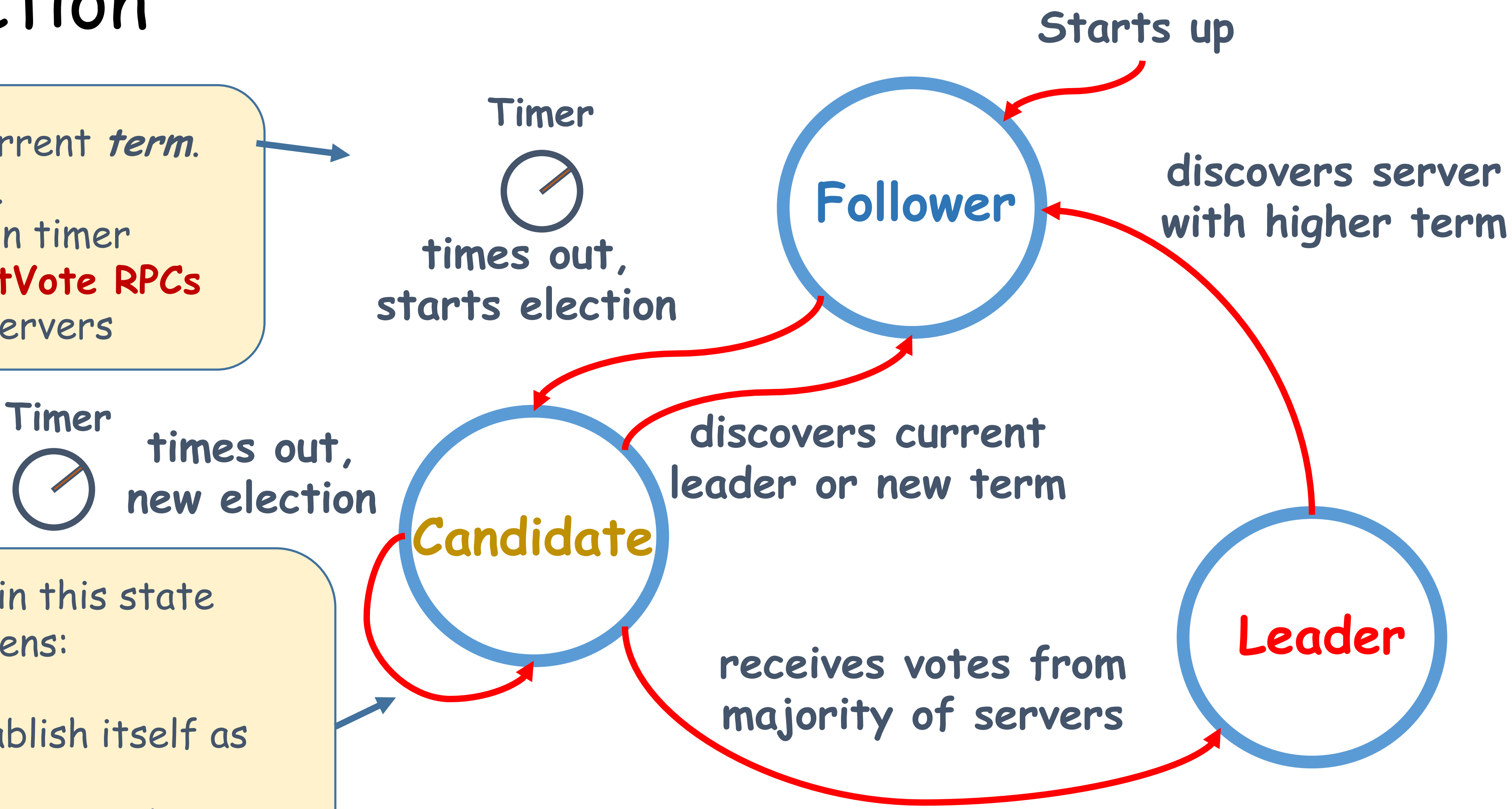
- trigger a timeout
- Reset to the initial time

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

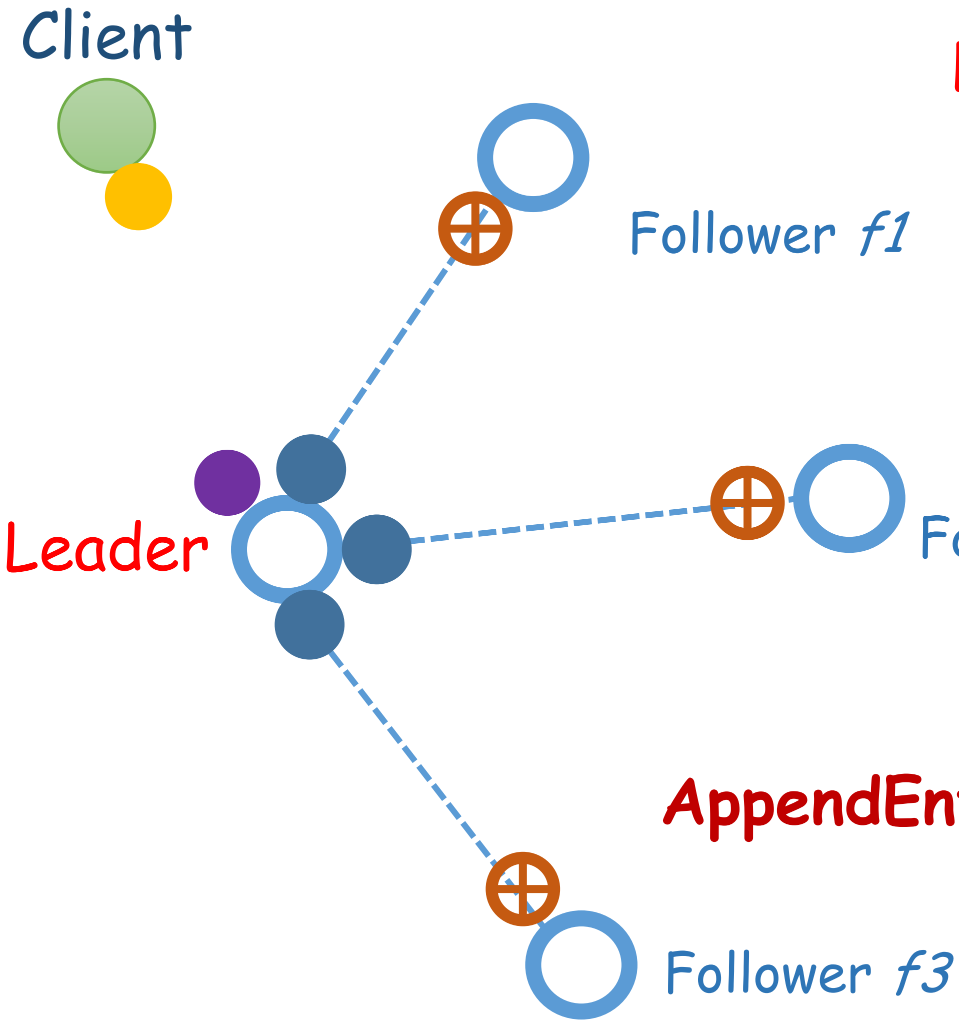
1. increment current *term*.
2. vote for self.
3. Reset election timer
4. Send **RequestVote RPCs** to all other servers

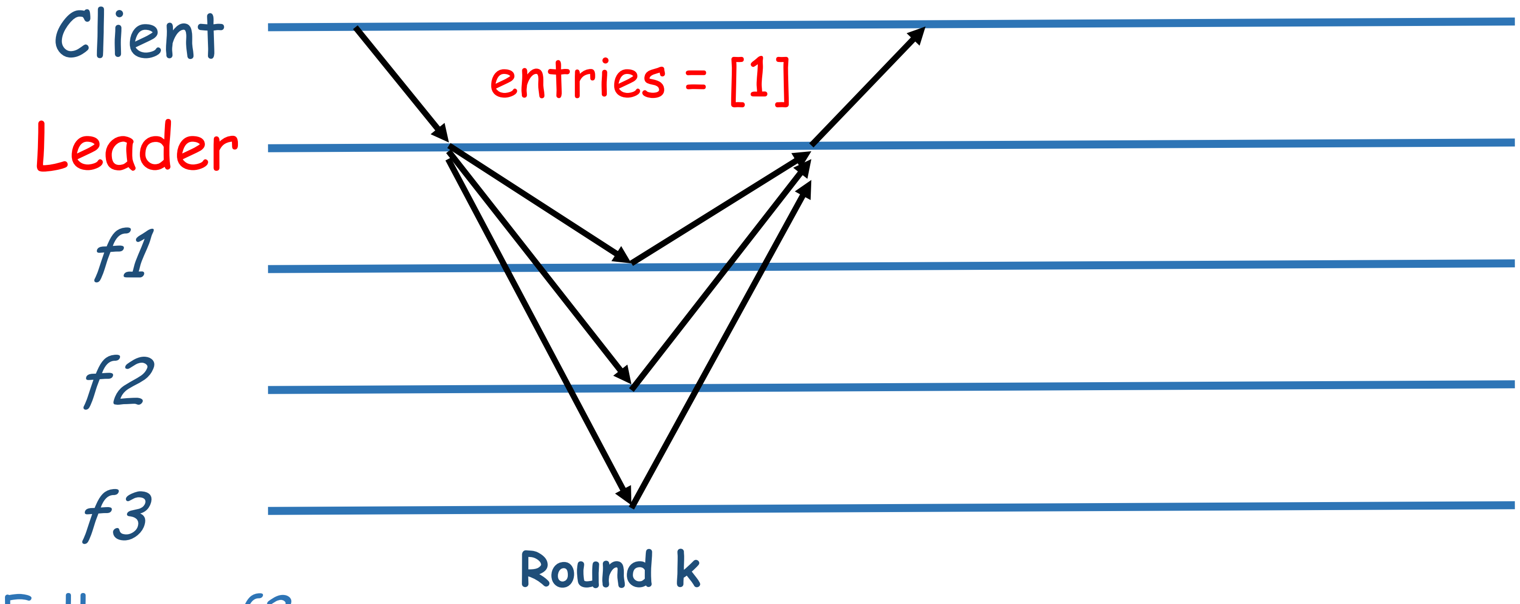


A candidate continues in this state until three things happens:

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

Log Replication



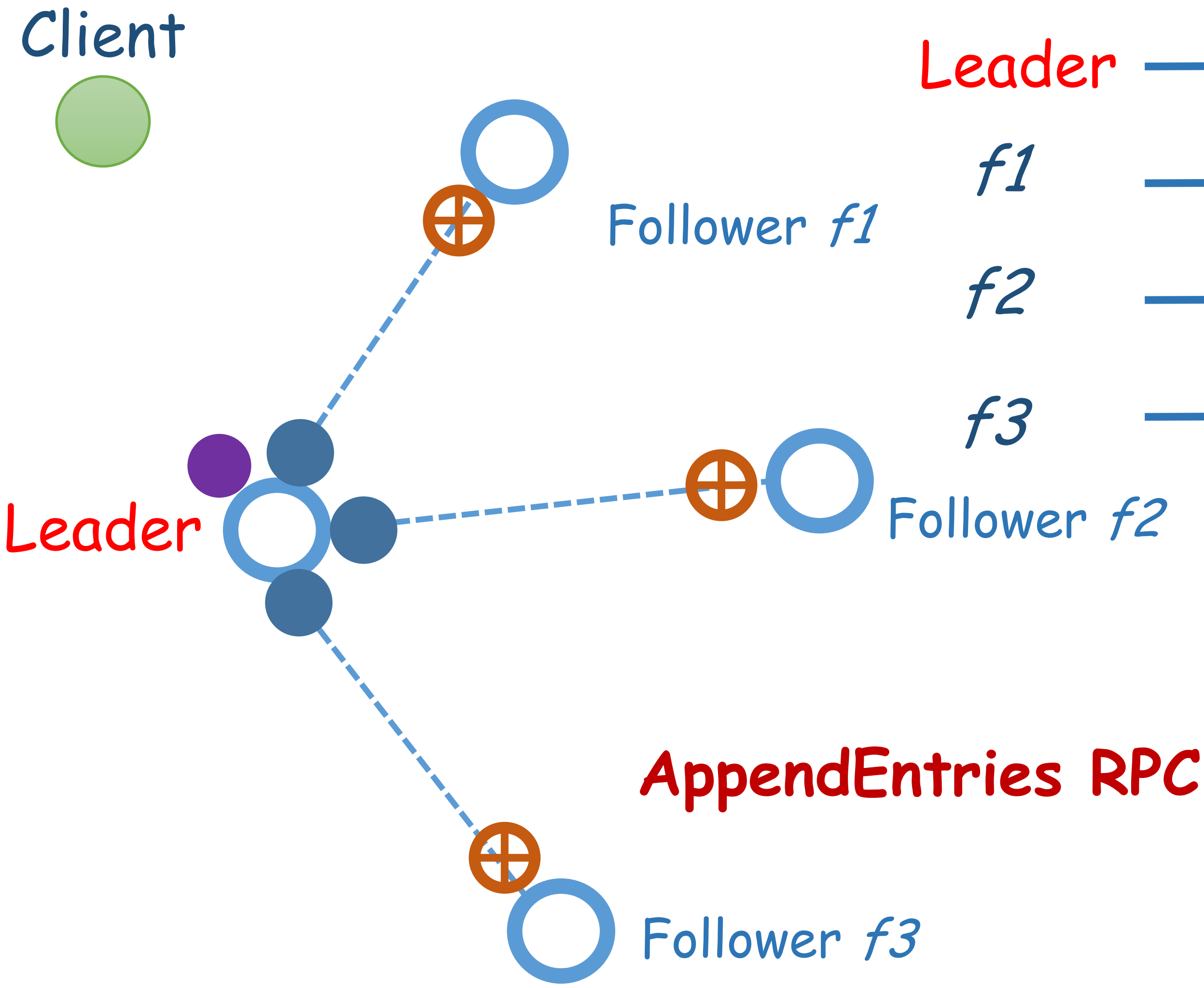


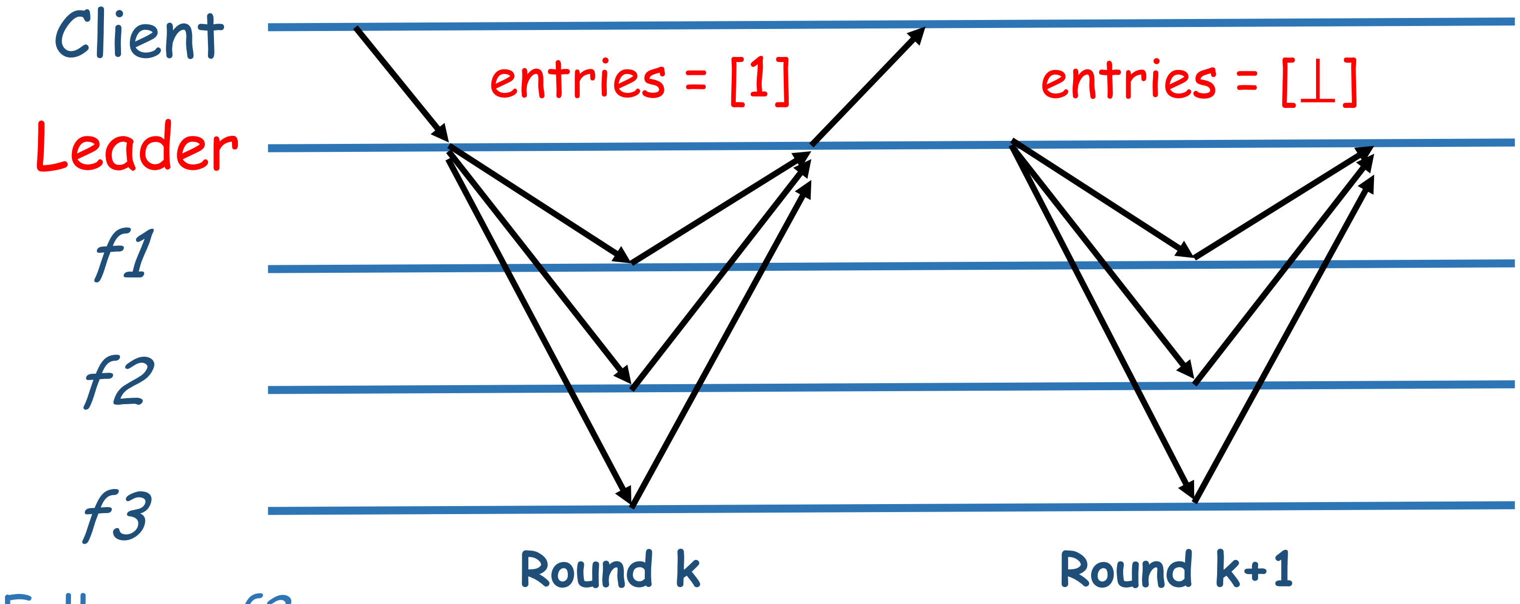
AppendEntries RPC

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

received majority replies

Log Replication





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

Term

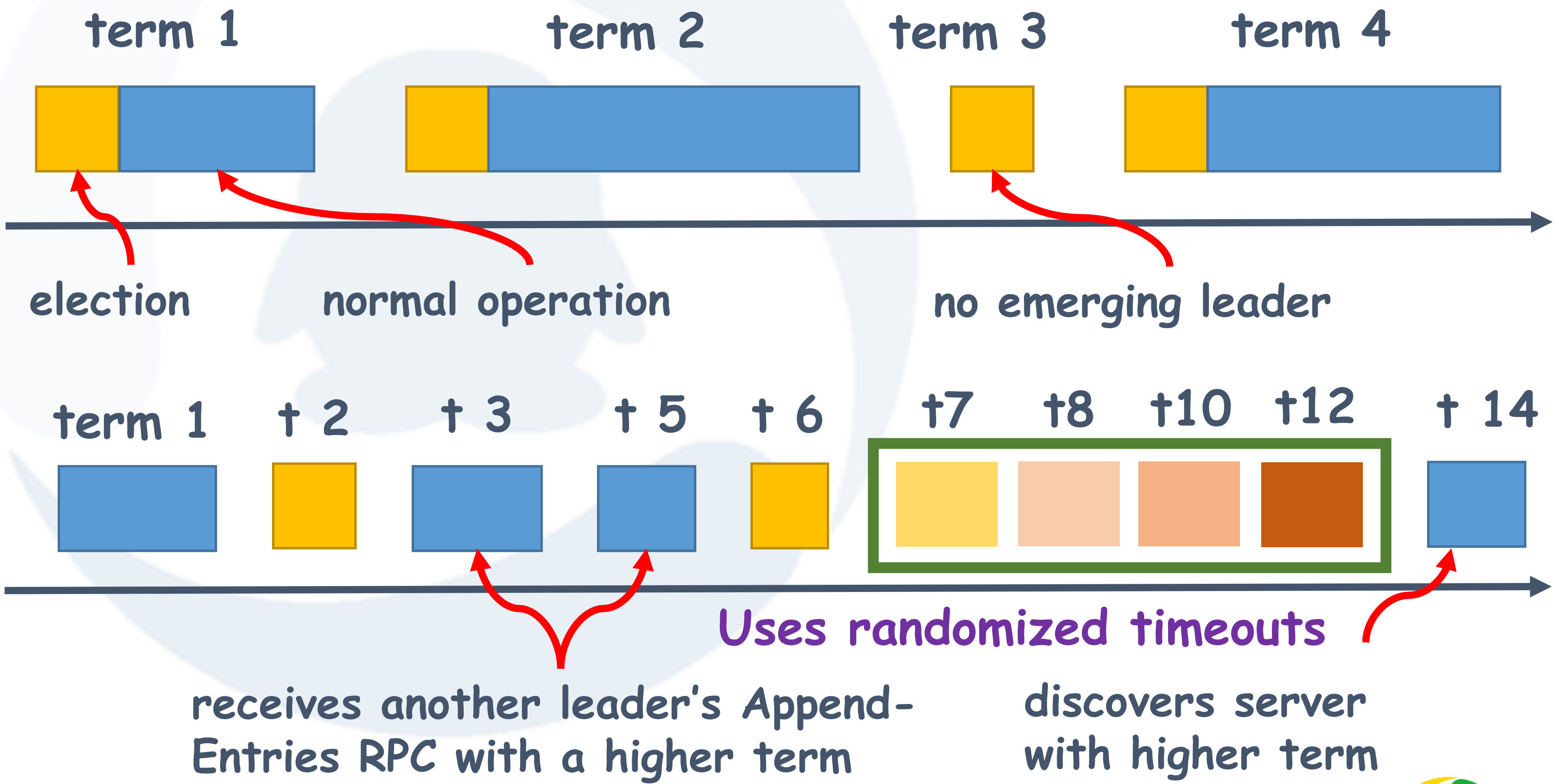
Time is divided into terms, and each term begins with an election. After a successful election, a single leader manages the cluster until the end of the term. Some elections fail, in which case the term ends without choosing a leader. The transitions between terms may be observed at different times on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



Learn more on …

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

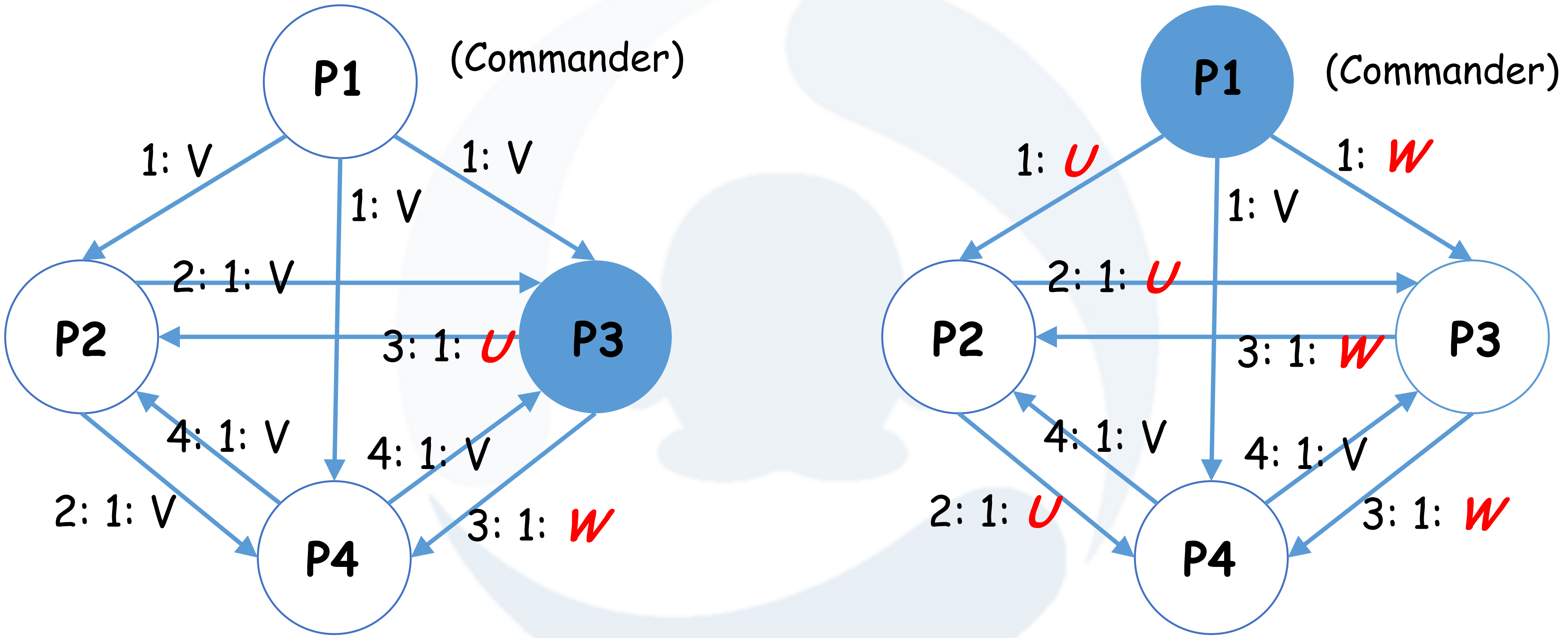
[11] Howard H, Schwarzkopf M, Madhavapeddy A, et al. Raft refloated: do we have consensus?[J]. ACM SIGOPS Operating Systems Review, 2015, 49(1): 12-21.

[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*

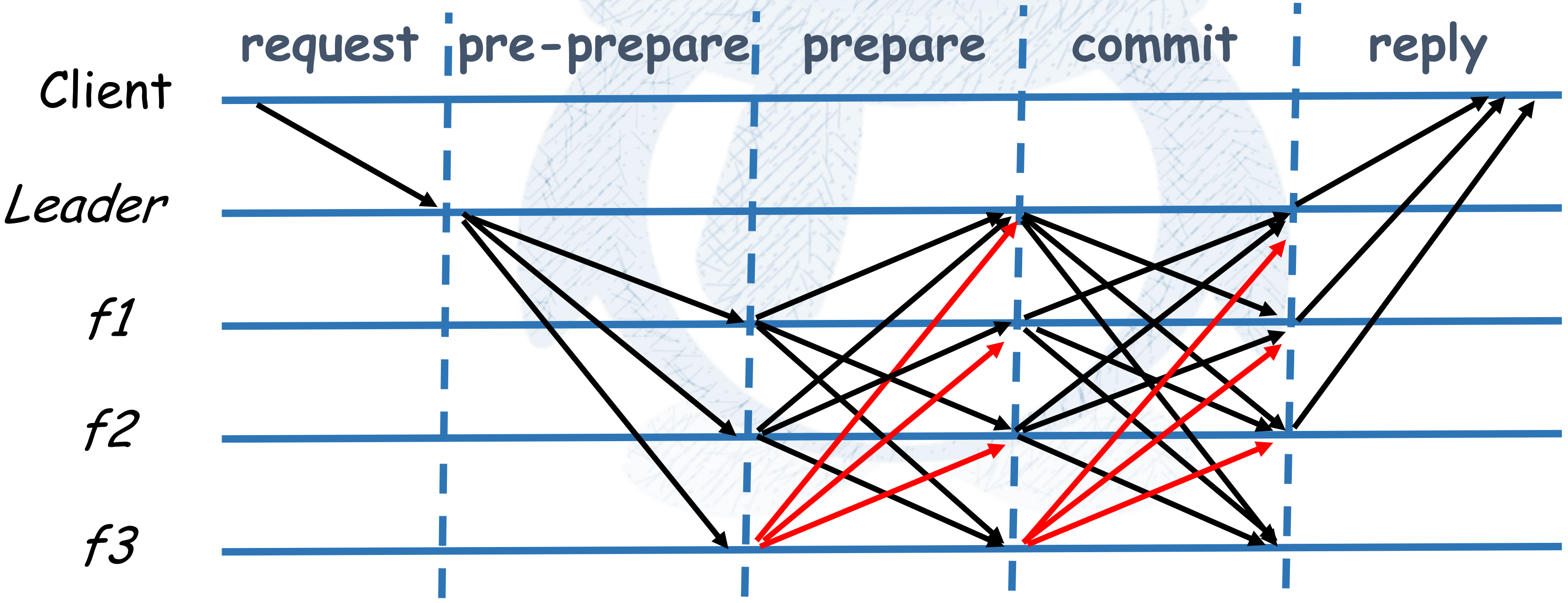
P4 decides on majority(*V*, *V*, *W*) = *V*

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.

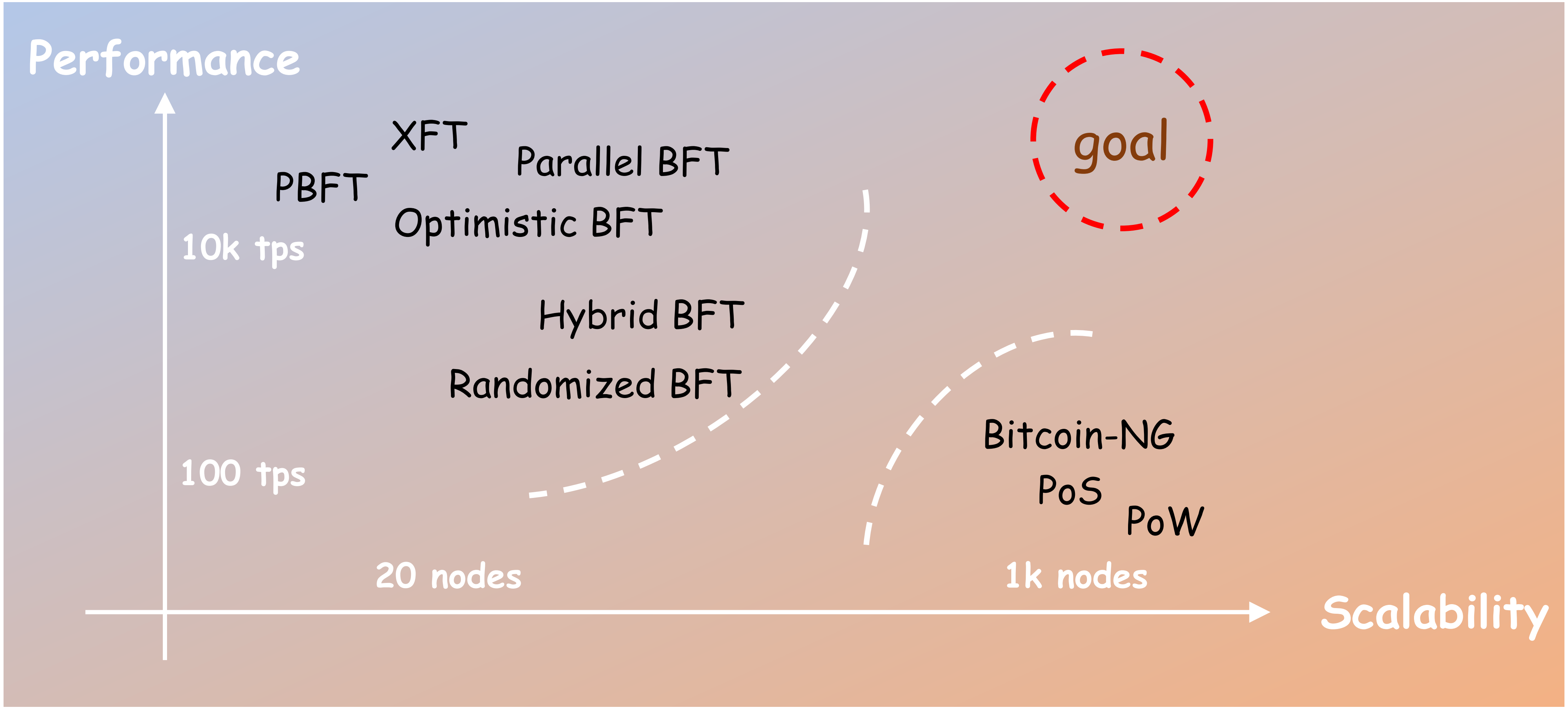
[19] Liu S, Viotti P, Cachin C, et al. XFT: Practical Fault Tolerance beyond Crashes[C]//OSDI. 2016: 485-500.

[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

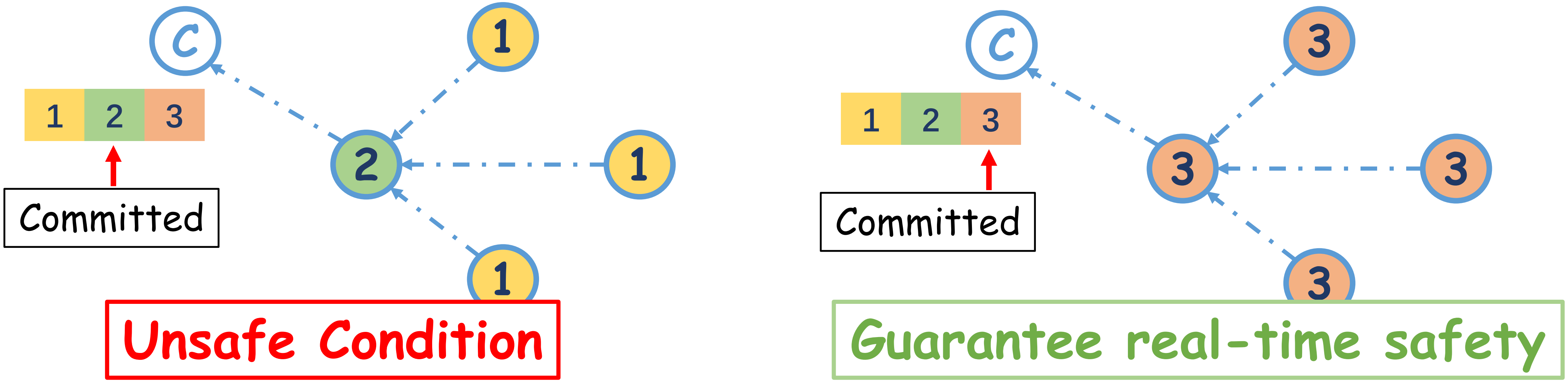
	Proof-Of-Work	Repli. StateM. / BFT based protocols
Node identity management	Open, entirely decentralized	Permissioned, nodes need to know IDs of all other nodes
Consensus finality	no	yes
Throughput	Limited (due to possible chain forks)	Good (tens of thousands tps)
Scalability	Excellent (like Bitcoin)	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



The safety limitations in Raft

The leader can not guarantee that a majority cluster has committed the entry before the $CommitIndex_l$ increases.



Design of Dynasty consensus protocol

- ✓ Two-Phase Commit
- ✓ View-Change

- ○ Guarantee real-time safety and Liveness
- ○ Increase throughput, decrease latency

[21] Zhang G, Xu C. An Efficient Consensus Protocol for Real-time Permissioned Blockchains under non-Byzantine Condititons [C]//International Conference on Green, Pervasive, and Cloud Computing. Springer, Cham, 2018

RPCs in Dynasty:

Propose RPC , *RequestVote* RPC, *LogPhase* RPC, *CommitPhase* RPC, *Heartbeat* RPC

(Notify) (Ticket) (LP Reply) (CP Reply) (HB Reply)

$\boldsymbol{Leader_{id}}$	redirects clients to a new leader when the elder leader crashes.
$\boldsymbol{term}$	leader's term
$\boldsymbol{Index_{log}^{prev}}$	index of log entries immediately proceeding new ones
$\boldsymbol{Entries(\Omega)}$	entries need to commit
$\boldsymbol{term_{log}^{prev}}$	term of the log entries with $Index_{log}^{prev}$

$\boldsymbol{Leader_{id}}$	redirects clients to a new leader when the elder leader crashes.
$\boldsymbol{term}$	leader's term
$\boldsymbol{Index_{log}^{prev}}$	index of log entries immediately proceeding new ones
$\boldsymbol{Index_{cmt}^{l}}$	commit index of leader

=> Followers passively reply: {*success*, *term*}

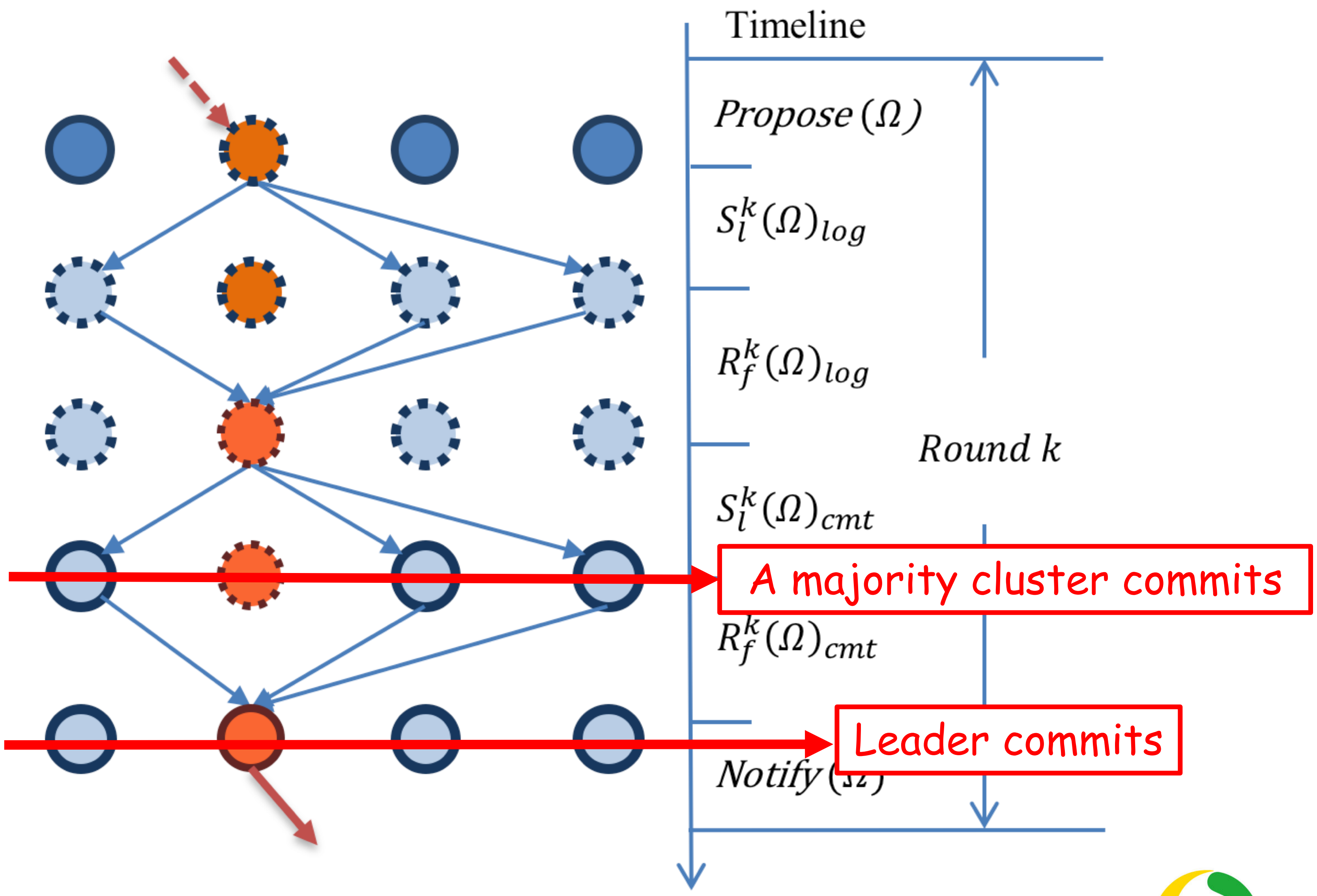
Two-Phase Commit

Propose
A client propose a record Ω

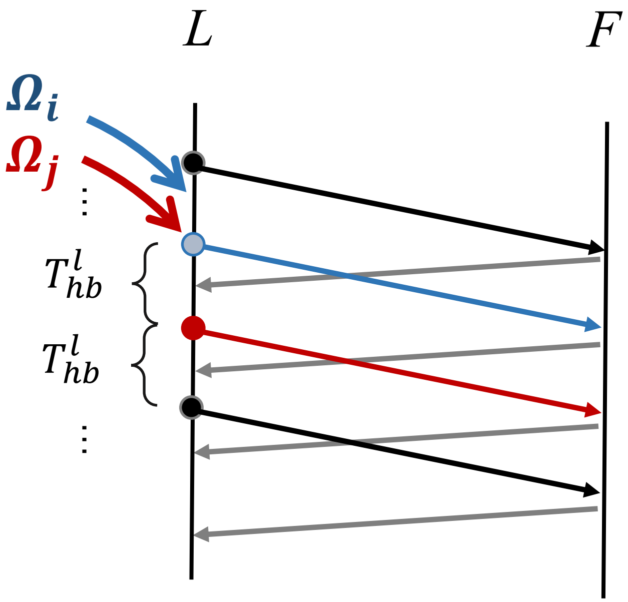
Log Phase
1. L broadcasts $S_l^k(\Omega)_{log}$
2. if (S_l^k passes LC)
   → f logs Ω
   → and returns $R_f^k(\Omega)_{log}$

Commit Phase
1. Broadcasts $S_l^k(\Omega)_{cmt}$ if L receives $n/2$ $R_f^k(\Omega)_{log}$
2. if (S_l^k passes LC)
   → f commits Ω
   → and gives $R_f^k(\Omega)_{cmt}$
3. When L receives $n/2$ replies, L commits Ω

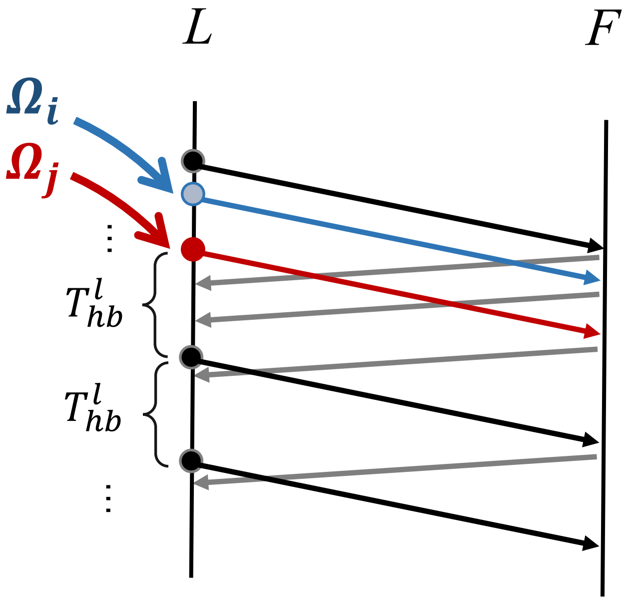
Notify
L notifies proposed clients



- Decrease Latency
- Increase throughput

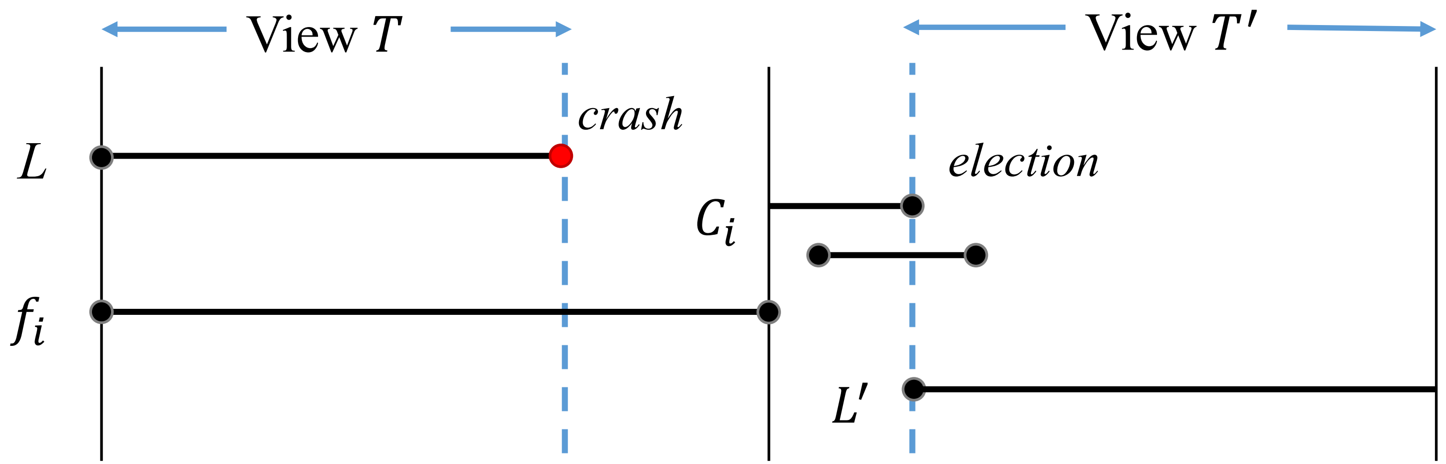


Raft



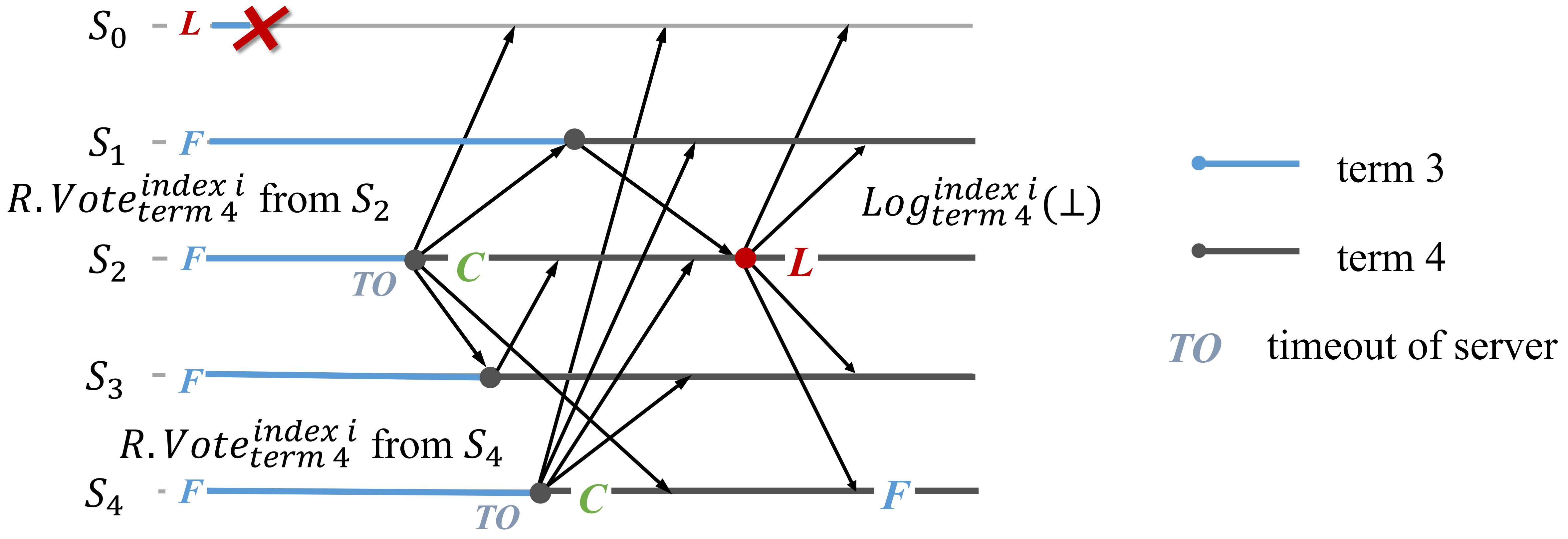
Dynasty

- View-Change

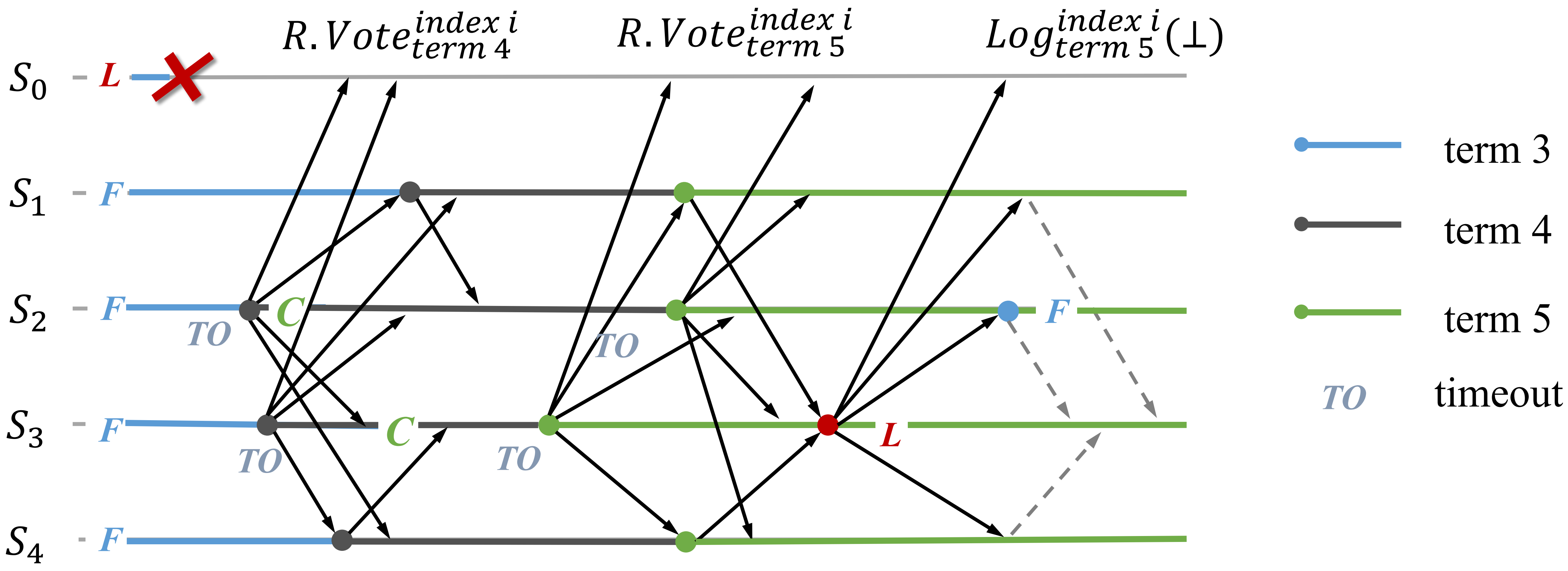


Guarantee Liveness:
A new leader must be chosen after a view T

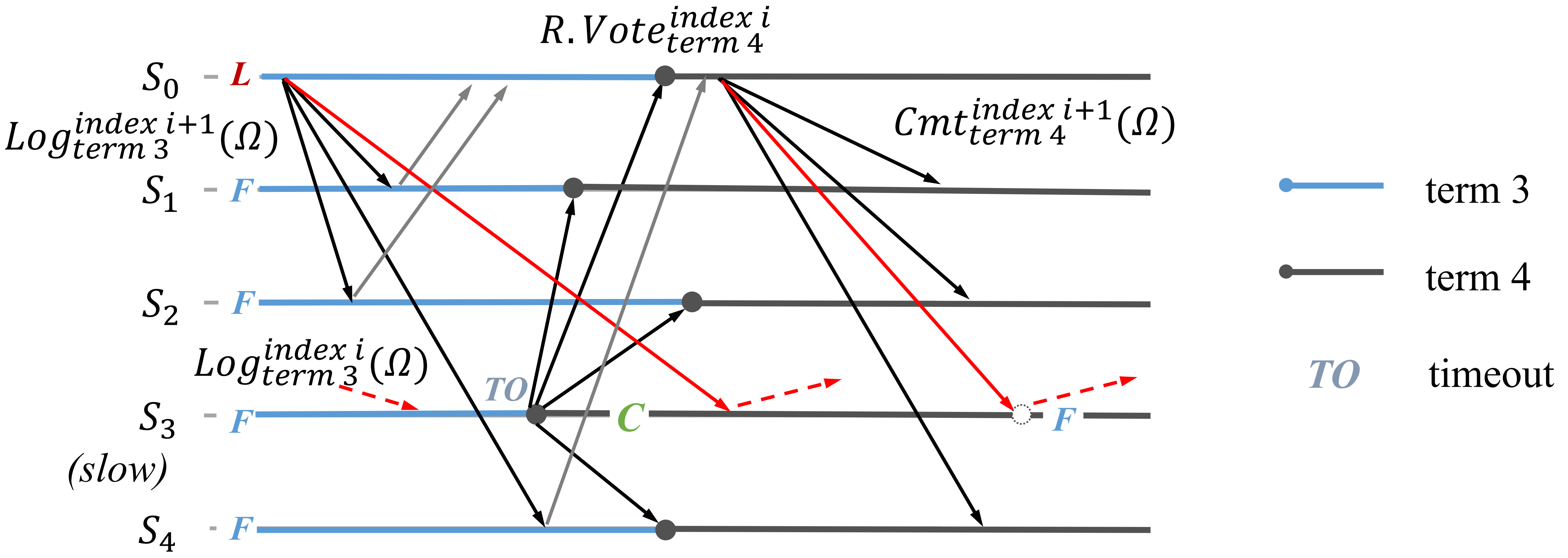
Case 1 => Two candidates with the same term



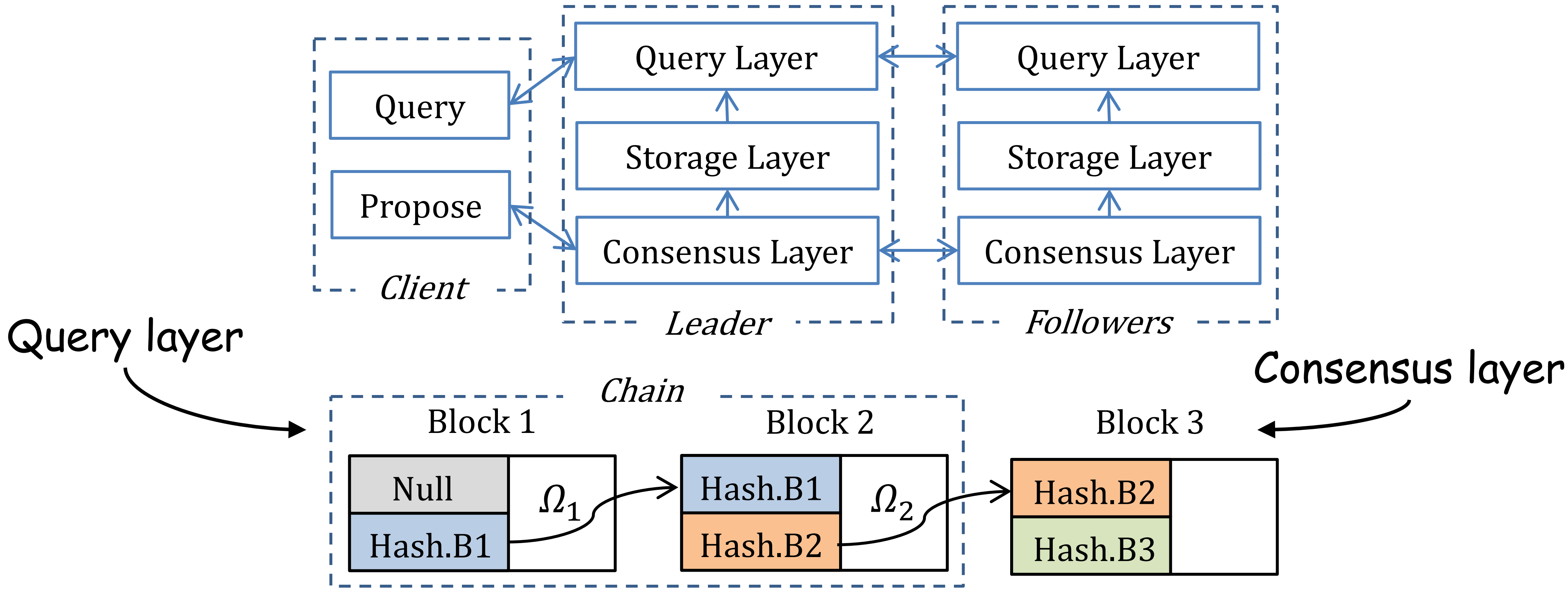
Case 2 => Two candidates with split votes



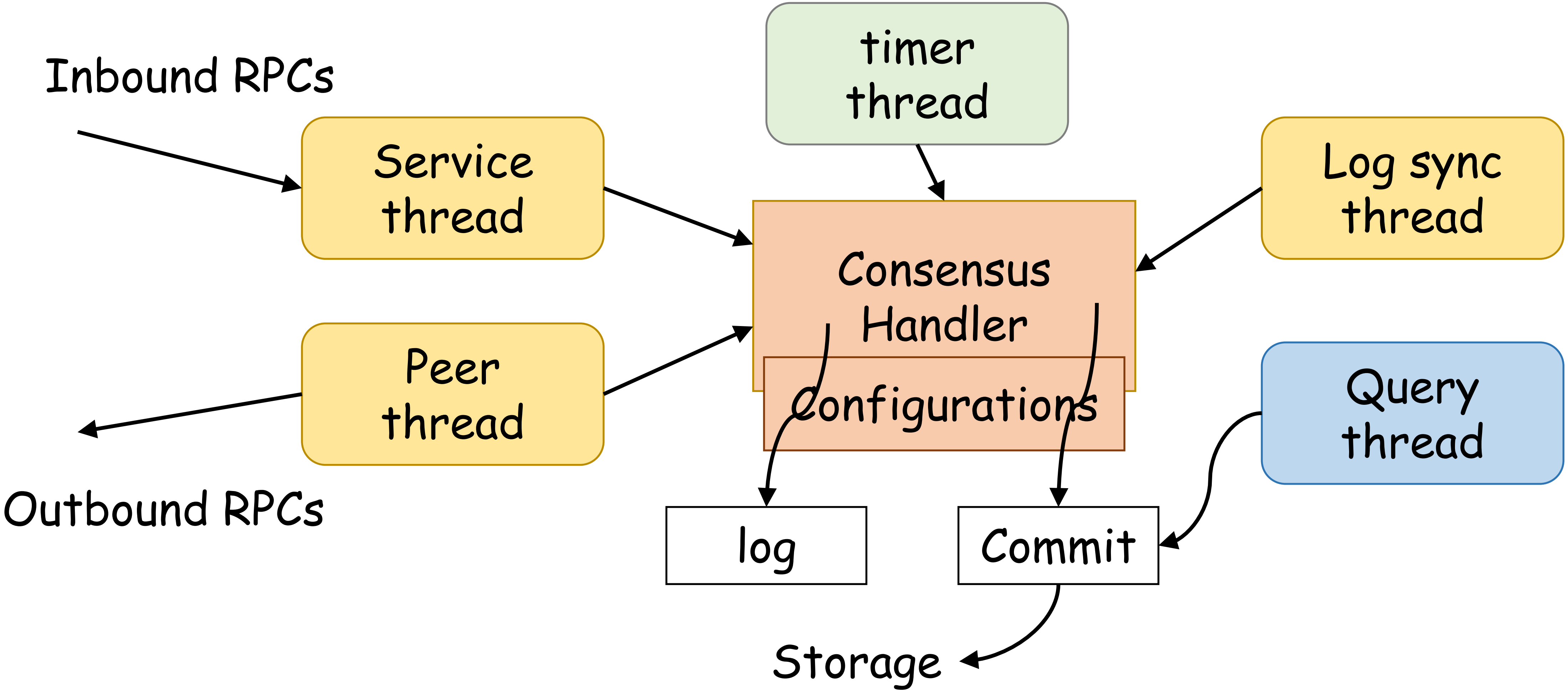
Case 3 => An election started by a slow node



Implementation of D-Chain



Threads



Inbound RPCs
Service thread
Outbound RPCs
Peer thread
timer thread
Consensus Handler
Configurations
Log sync thread
Query thread
log
Commit
Storage

Applications

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

Blockchain as a Service (BaaS)

Blockchain framework
"Consensus Algorithm",
"Data Structure"

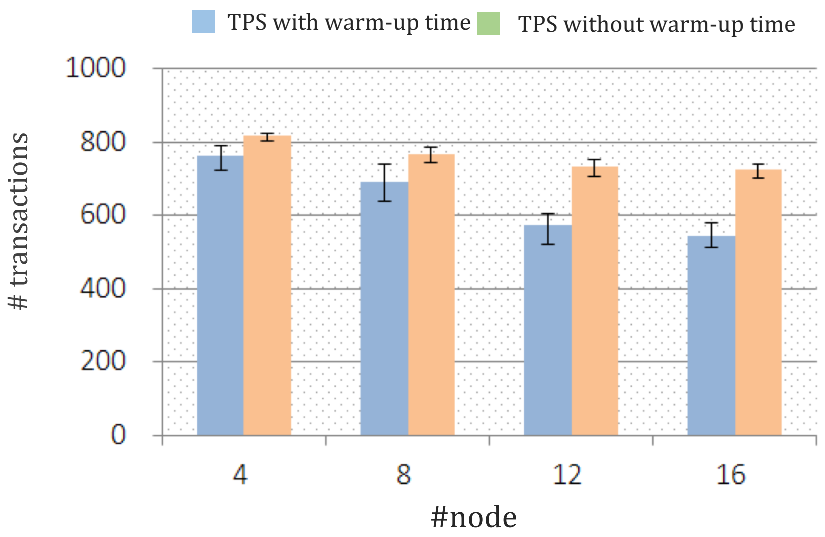
Digital Content Protection
"Blockchain based"

Evaluation of D-Chain framework

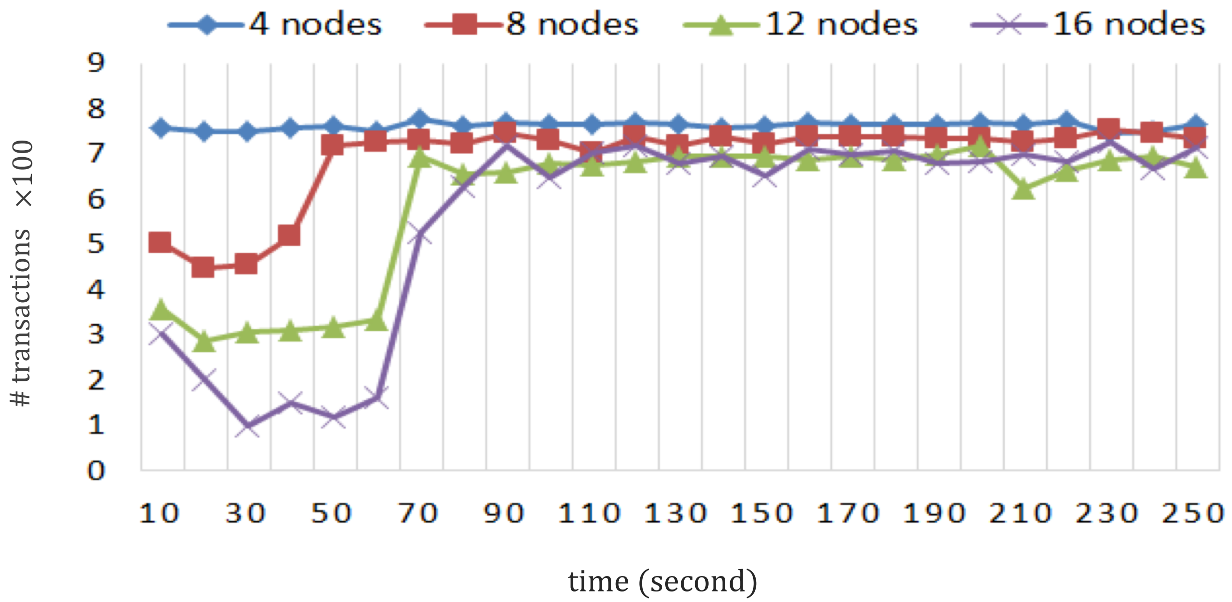
- ❖ Measured throughput and latency on clusters of 4, 8, 12 and 16 nodes.
- ❖ Not considering any case of node failure.
- ❖ All the results are averaged over 10 independent runs.

Each server has an E5-2630 2.40GHz CPU, 64 GB RAM, 2 TB hard drive, running on Ubuntu 14.04.1, and connected to the other servers via 1GB switch.

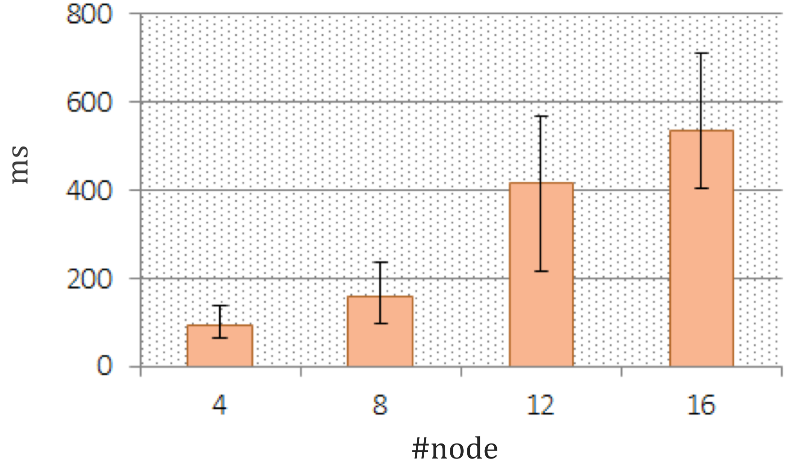
(a) Transactions per second



(b) Transactions Committed



(c) Latency



Measured tests on 4, 8, 12, 16 nodes.

While the latency in different scales of the system increases as expected, the number of committed transactions per second stabilizes at a point within less than 8% difference after a warming-up period

Future work

Design a strong-leader based consensus protocol that tolerates Byzantine fault.

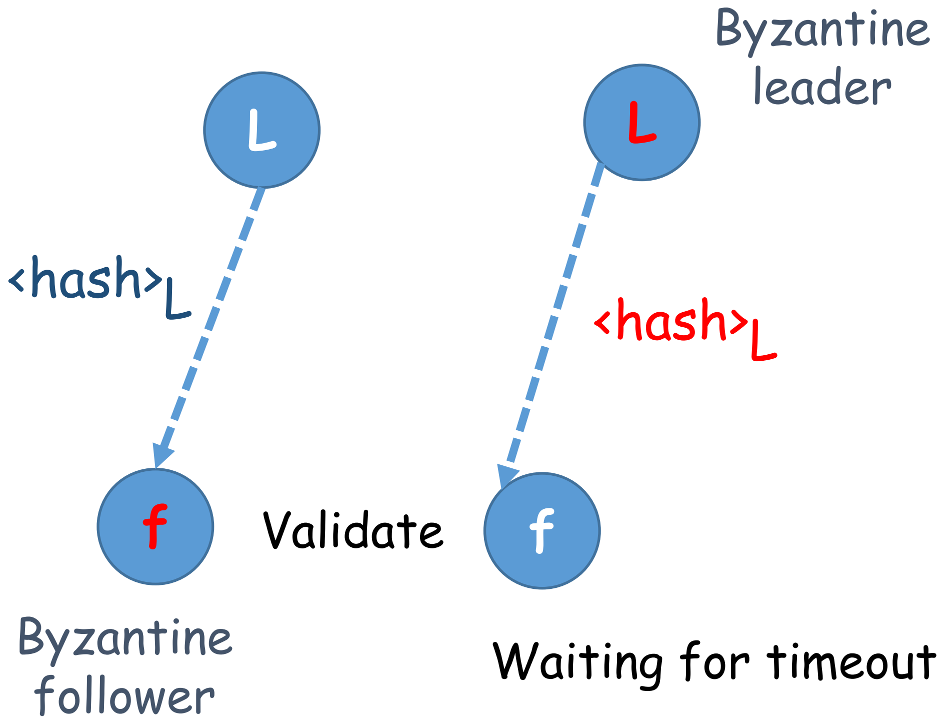
- Reduce the additional costs of Byzantine broadcast.
- Improve the performance of throughput and latency in normal case.

Raft → Log replication + incorrect message from other servers. (Safety)

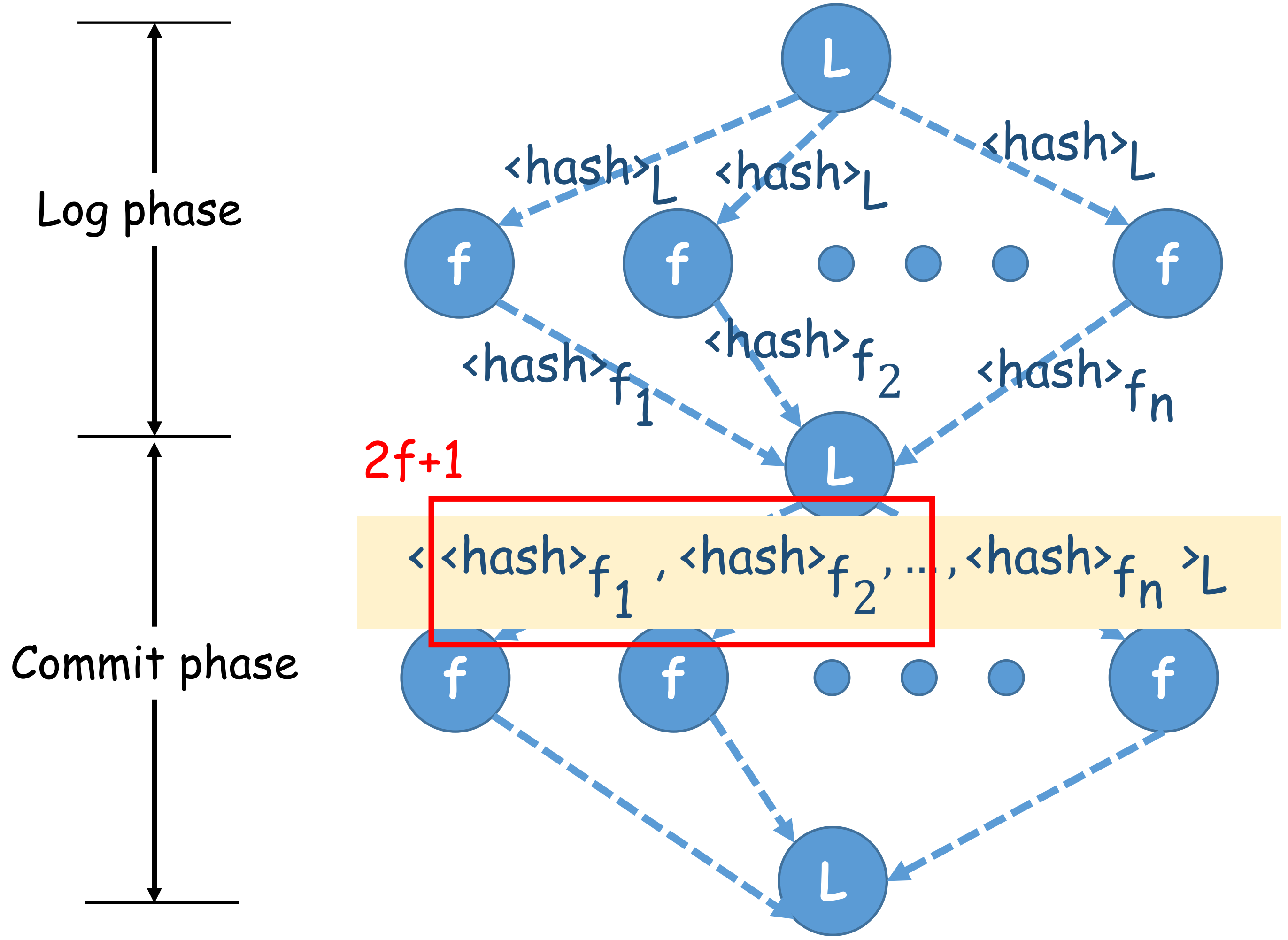
Raft → Leader election + continuously starting new elections. (Liveness)

Log replication

+ using signature and hash

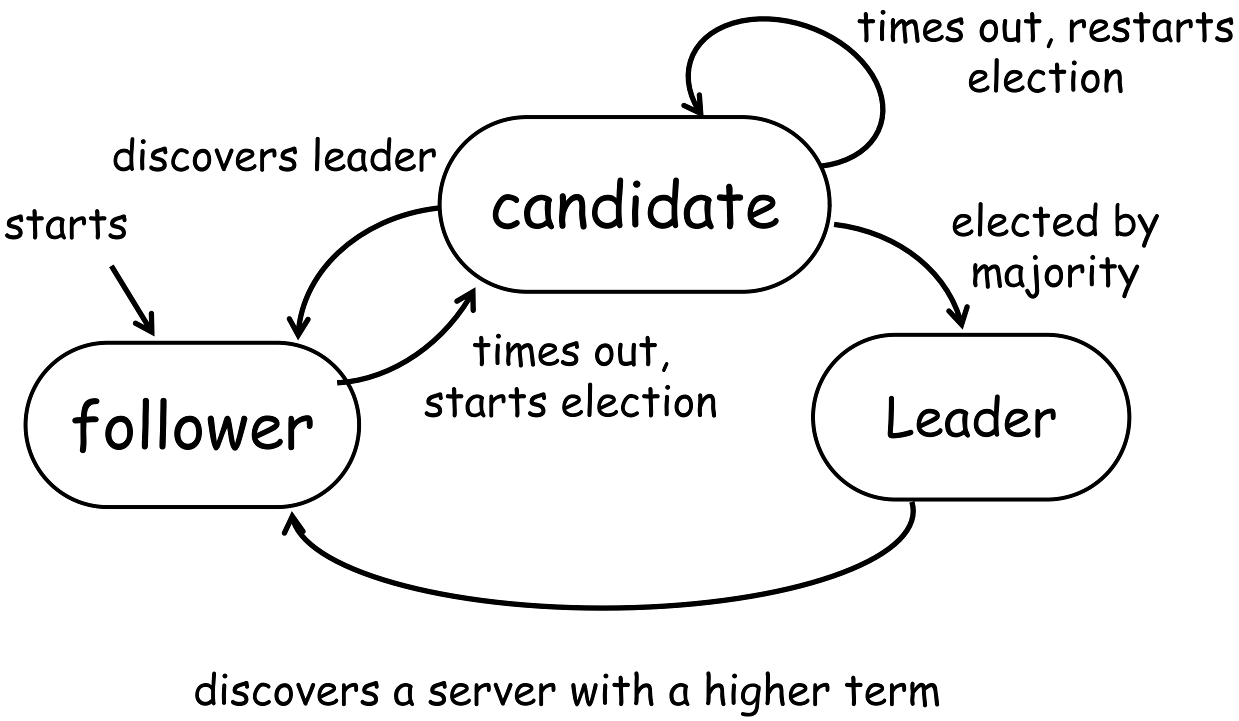


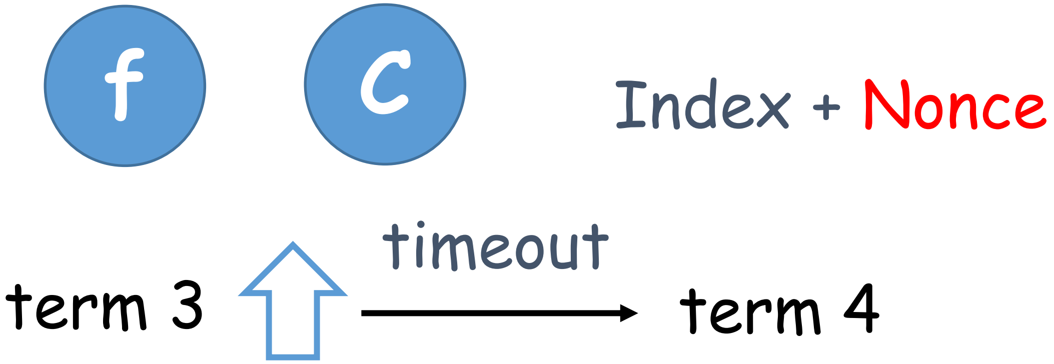
During normal case



Leader election + using Proof-of-CommitIndex (PoCI)

Prevent the node from continuously increasing the term value ⇨ Guarantee the liveness





Proof of current commit index. Hash code:

000klx49f….

Continuously increasing: term 5, 6 ..

0000g8xk3r….

00000p5cgx4kl….

Thank you for listening!

Questions?

Tencent 腾讯