Towards consensus on distributed consensus

Flavio Junqueira

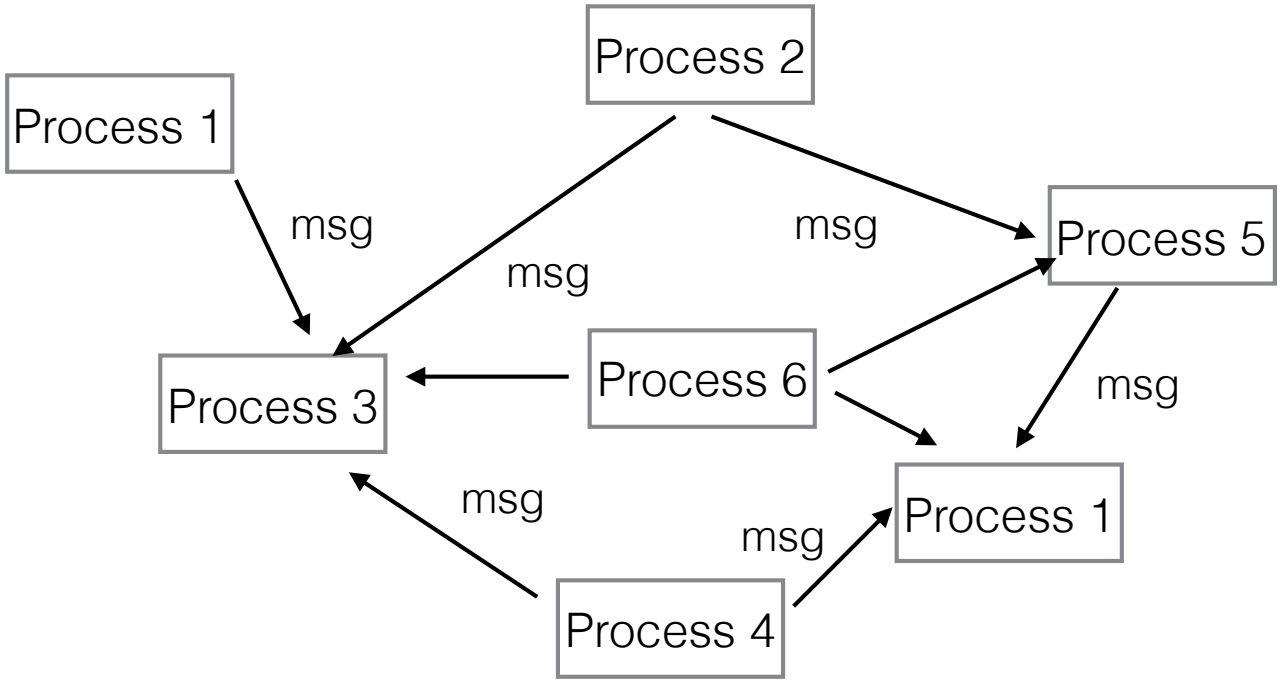
Real-time streams powered by Apache Kafka.

About me

- Core area of expertise: **distributed computing**
- **Confluent**
  - ✓ Infrastructure Engineer
  - ✓ Kafka Core
- **Apache Software Foundation (ASF)**
  - ✓ Apache ZooKeeper, BookKeeper, Kafka
  - ✓ Apache Incubator
- Previously
  - ✓ **Yahoo! Research** and **Microsoft Research**

Distributed Systems

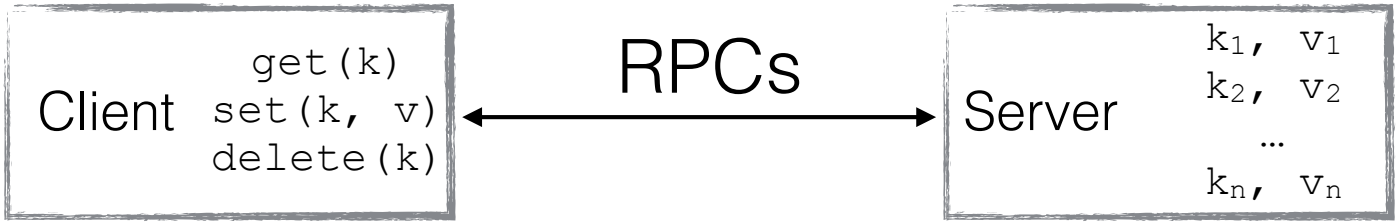
Messages and Processes



System:

- *n* independent processes
- communicate by exchanging messages
- messages follow a protocol

Shared objects and RPCs



Network messages are transparent to the processes

In this talk…

- Distributed consensus
  - ✓ Agreement among processes
- Consensus is a fundamental primitive
  - ✓ … you can get around without it, but not always
- Typically, not in the critical path

Distributed Consensus

Distributed Consensus

- Set of processes, nodes, servers….
- Each process proposes an **initial value**
- Processes **eventually agree** on a value
- Must tolerate crashes

Distributed Consensus

- Set of processes, nodes, servers….
- Each process proposes an **initial value**
- Processes **eventually agree** on a value
- Must tolerate crashes No Byzantine behavior

Distributed Consensus

P2

Color:

P1

Color:

Colors are the proposed values

P3

Color:

Distributed Consensus

P2

Color: ■

P1

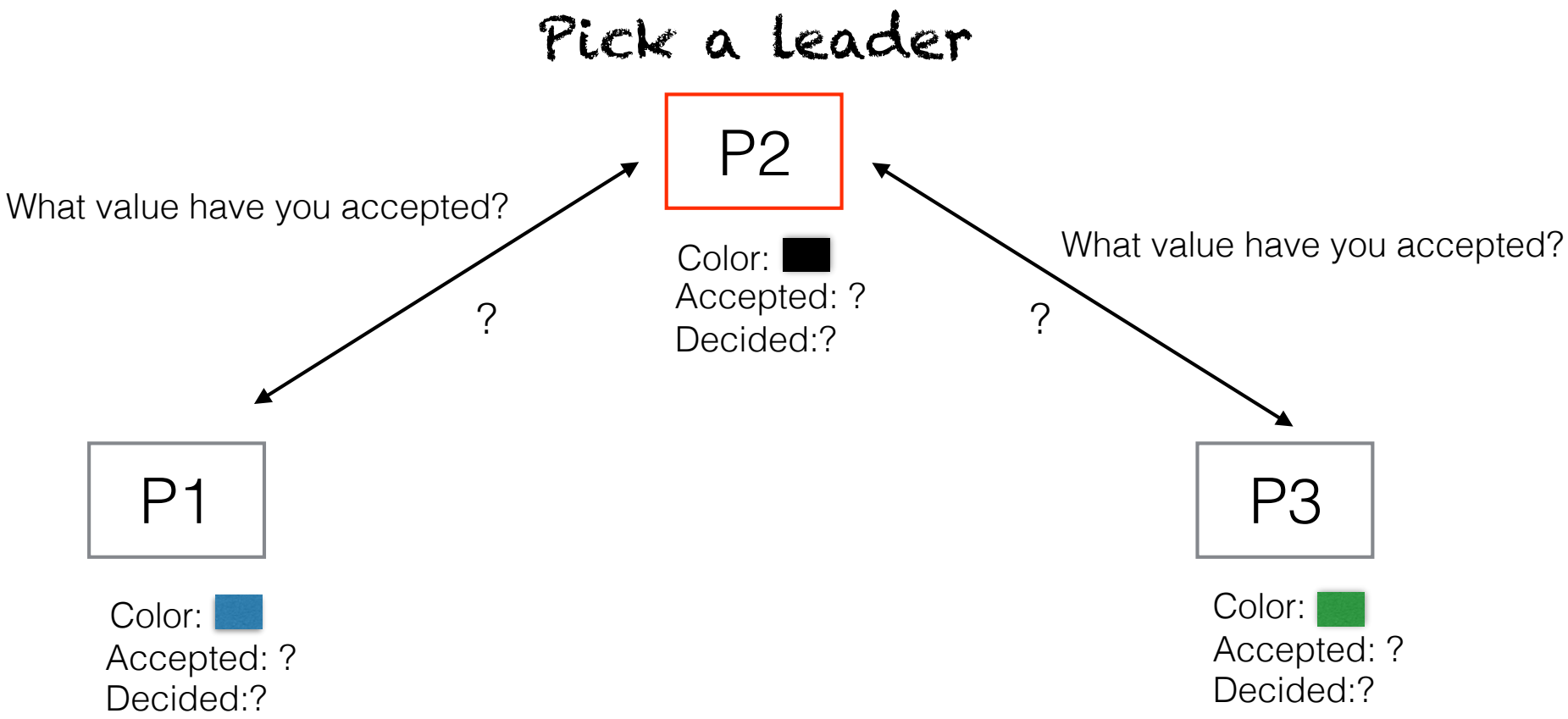
Color: ■

Black is decided.

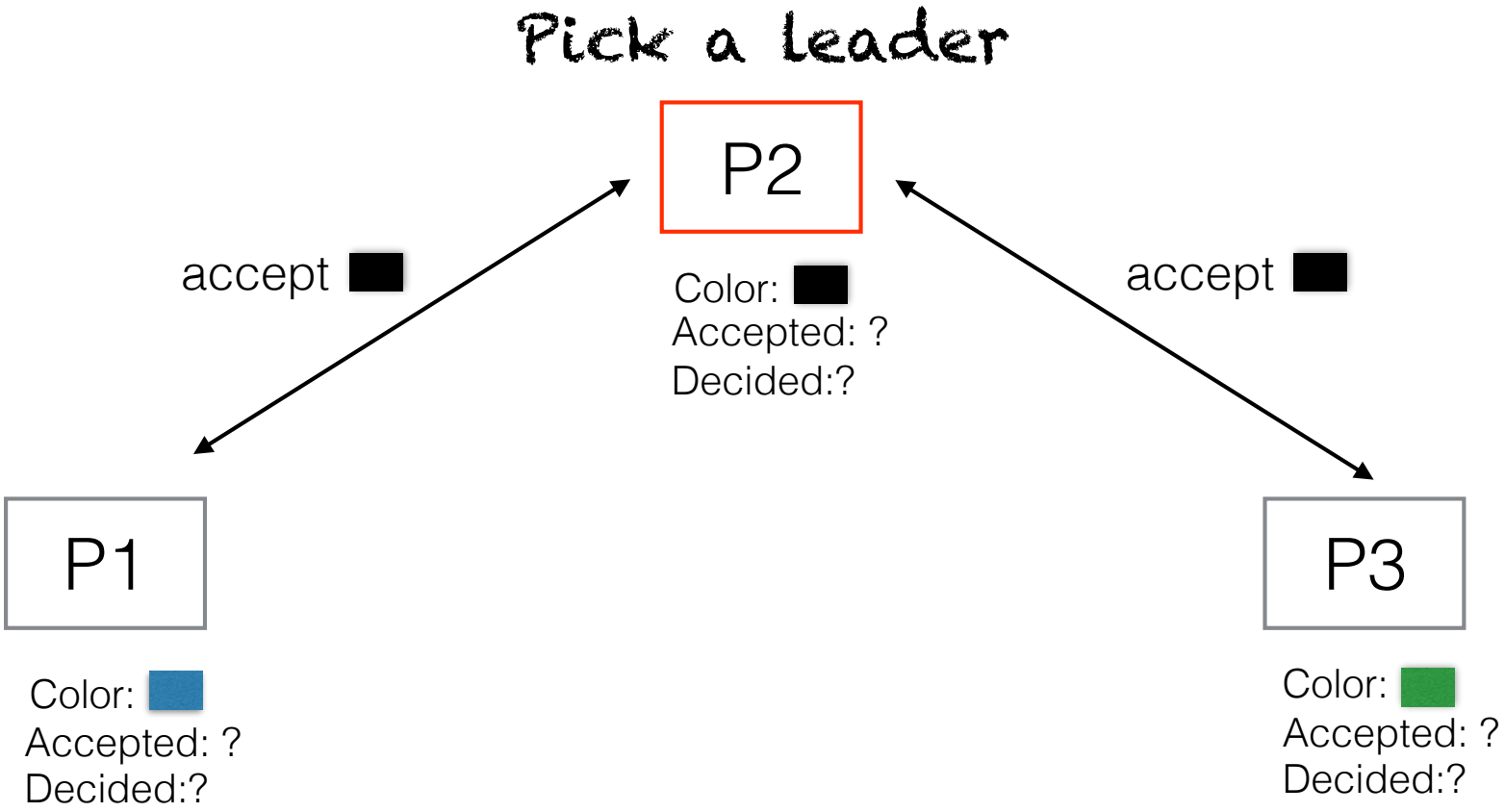
P3

Color: ■

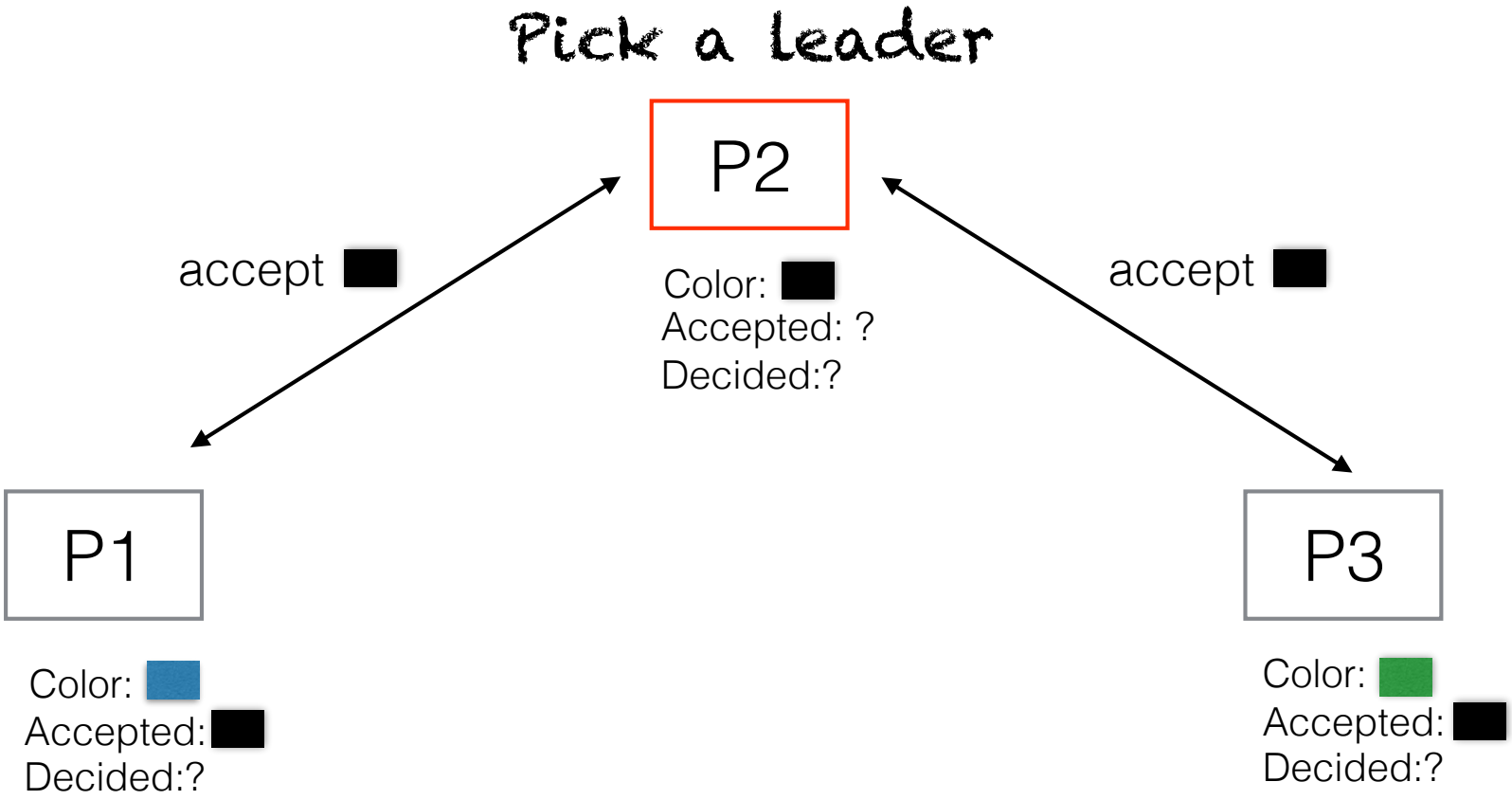
How do we get consensus?



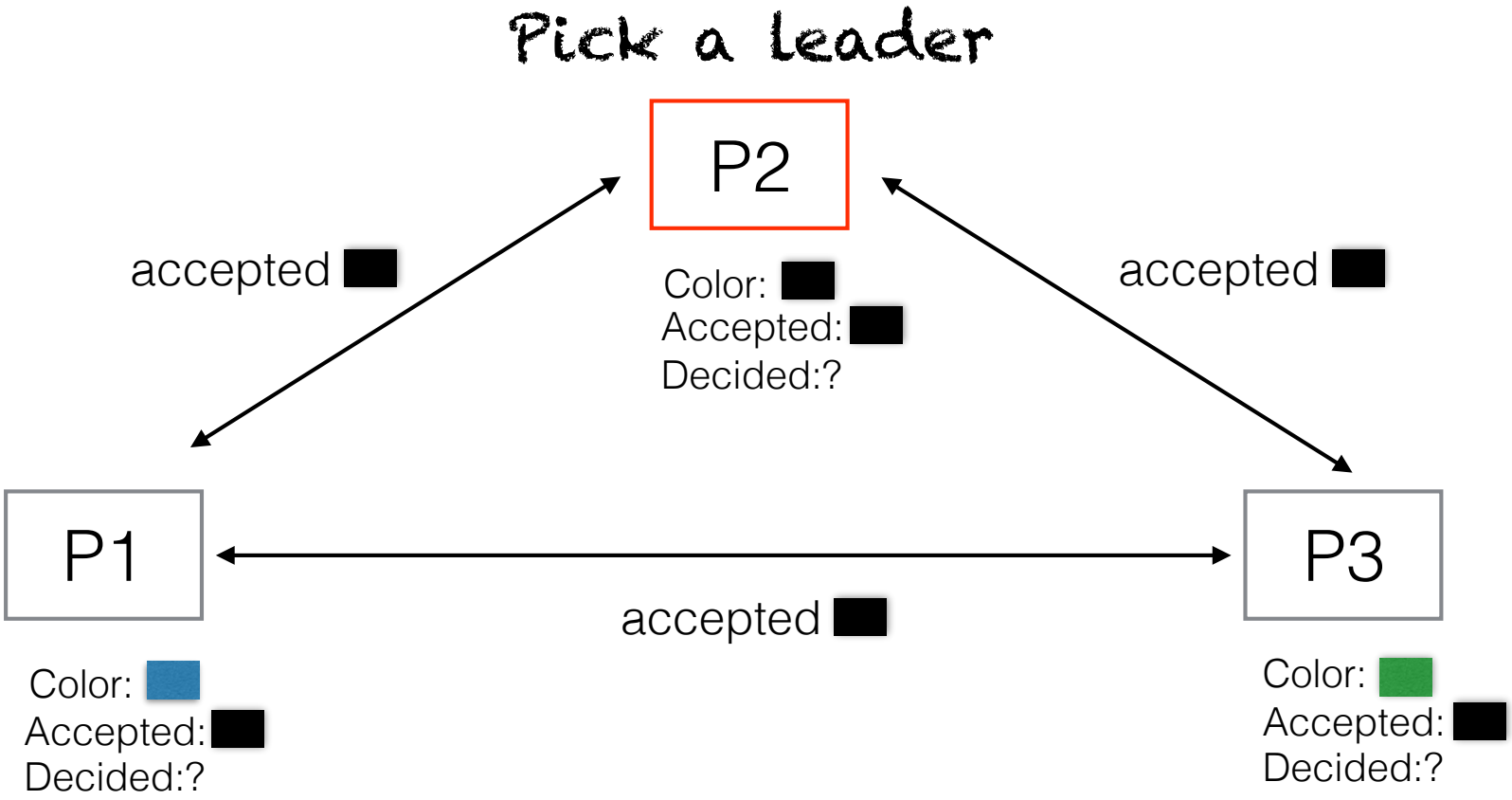
How do we get consensus?



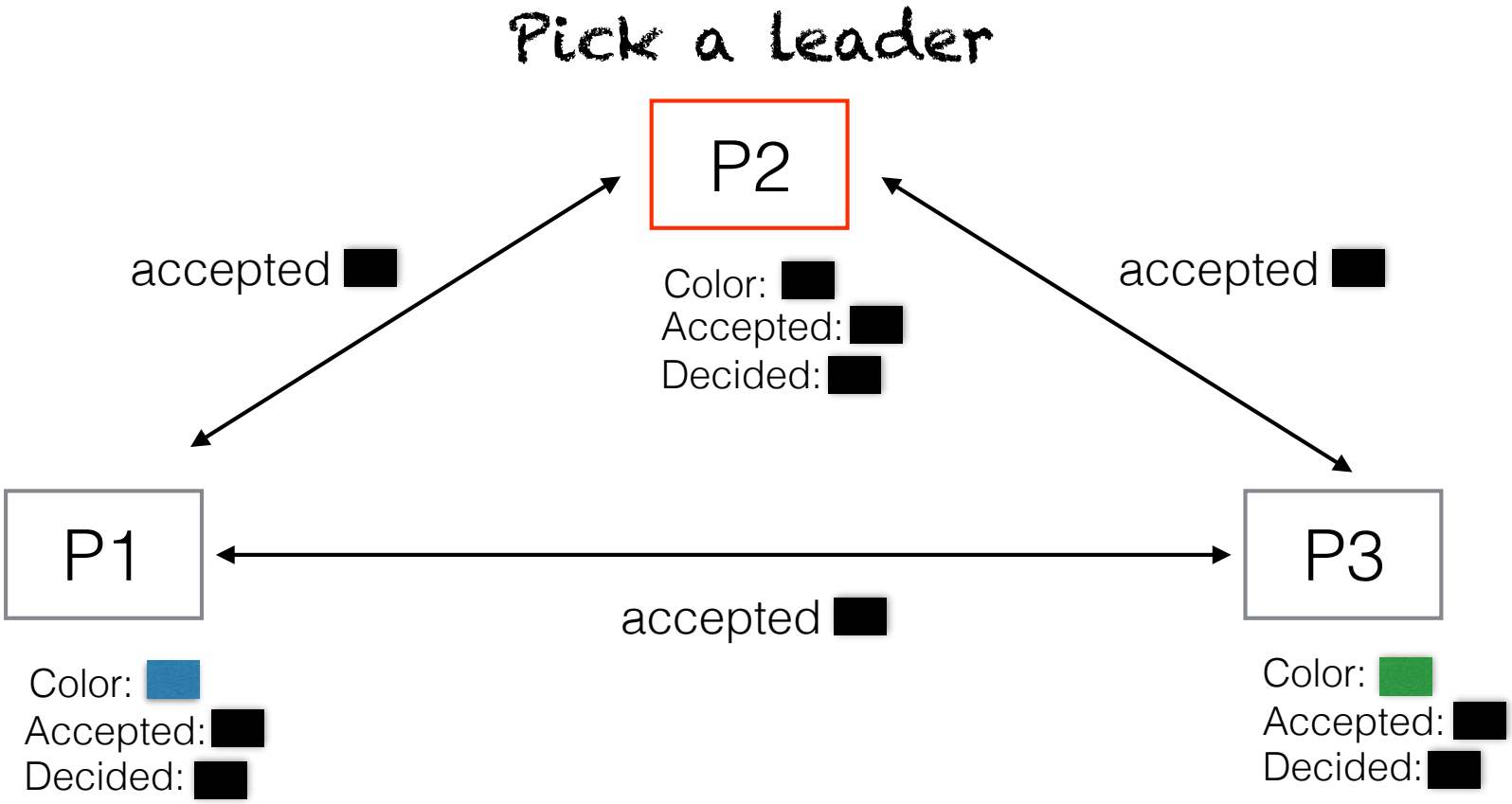
How do we get consensus?



How do we get consensus?

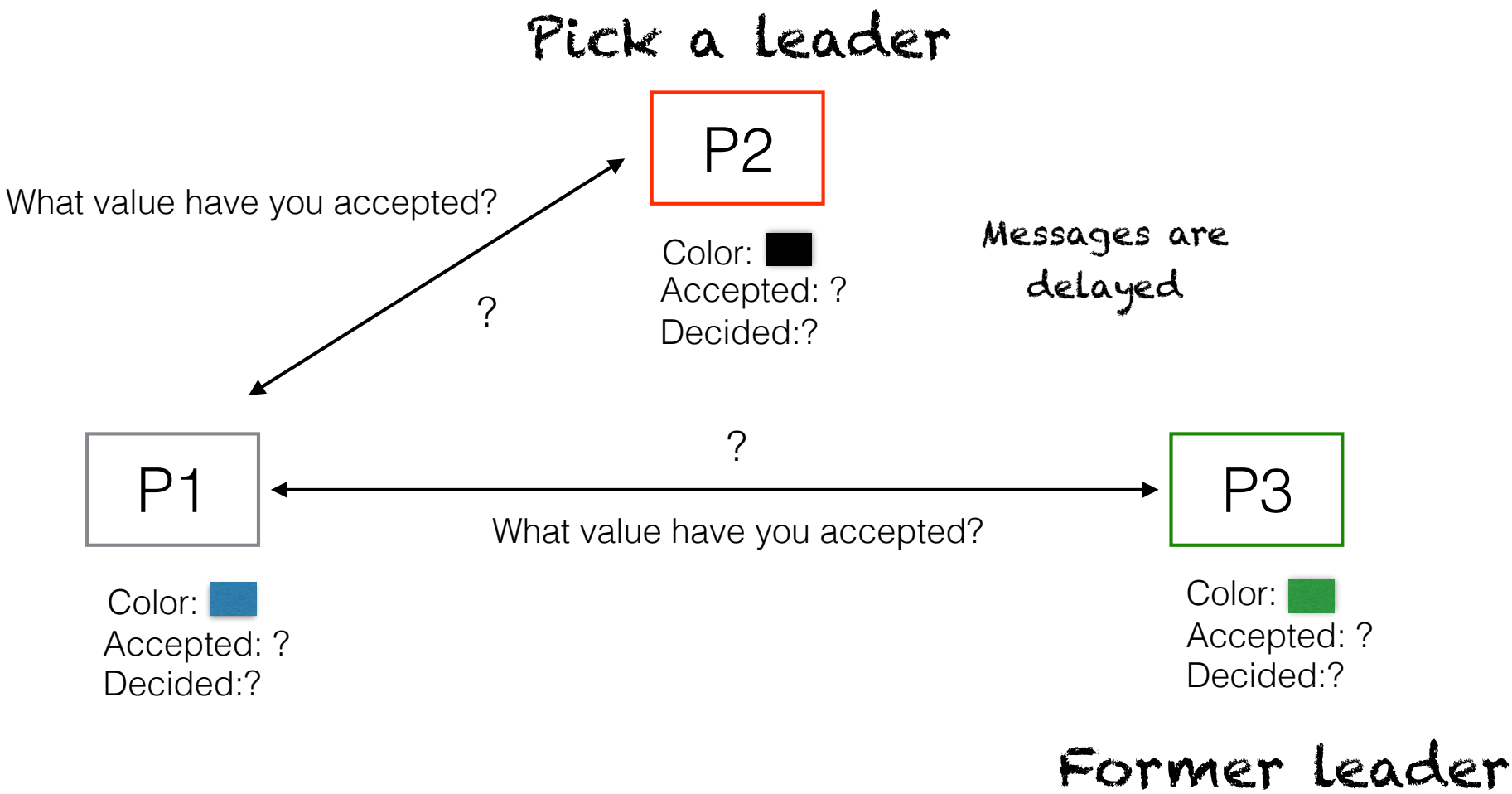


How do we get consensus?

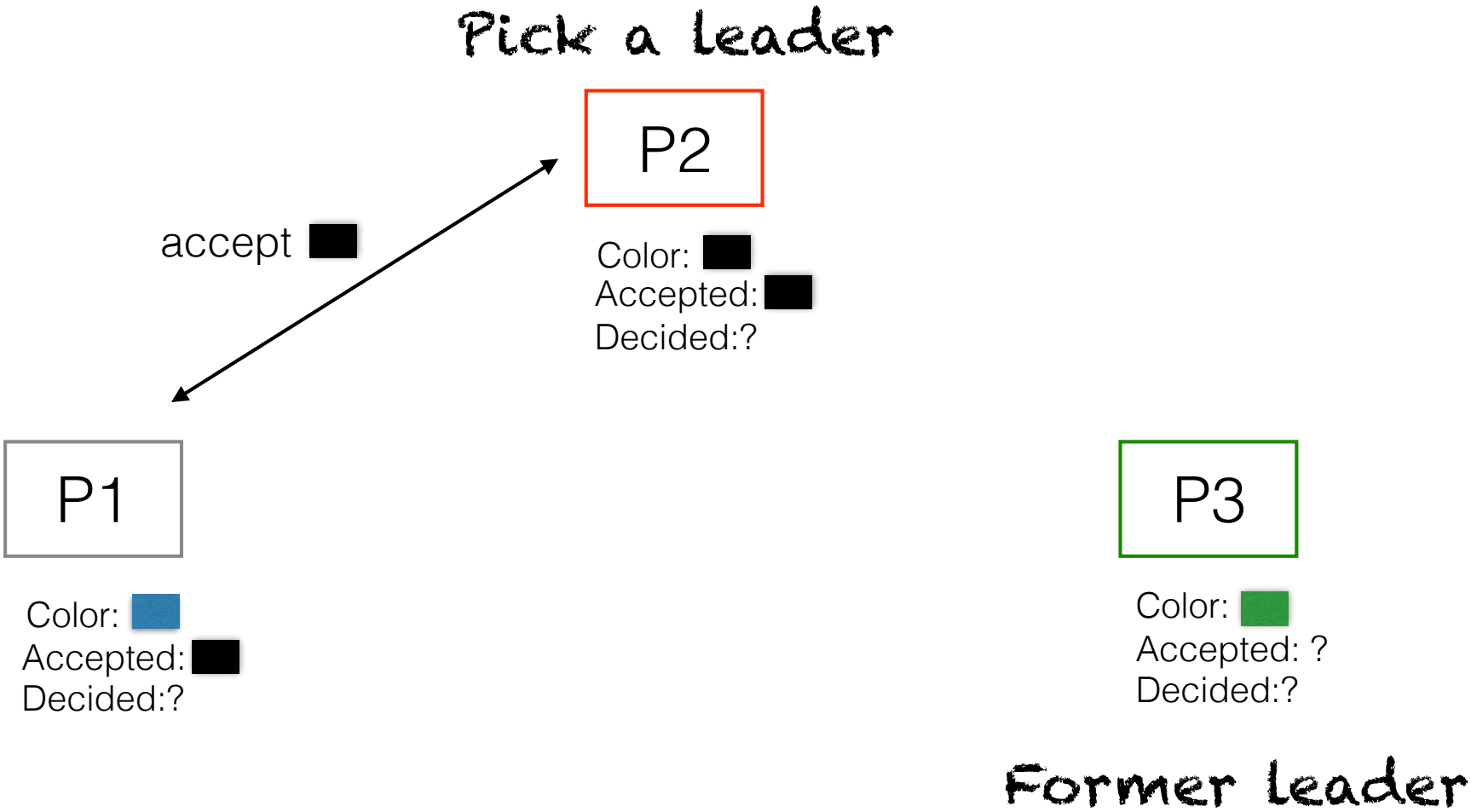


Not entirely safe…

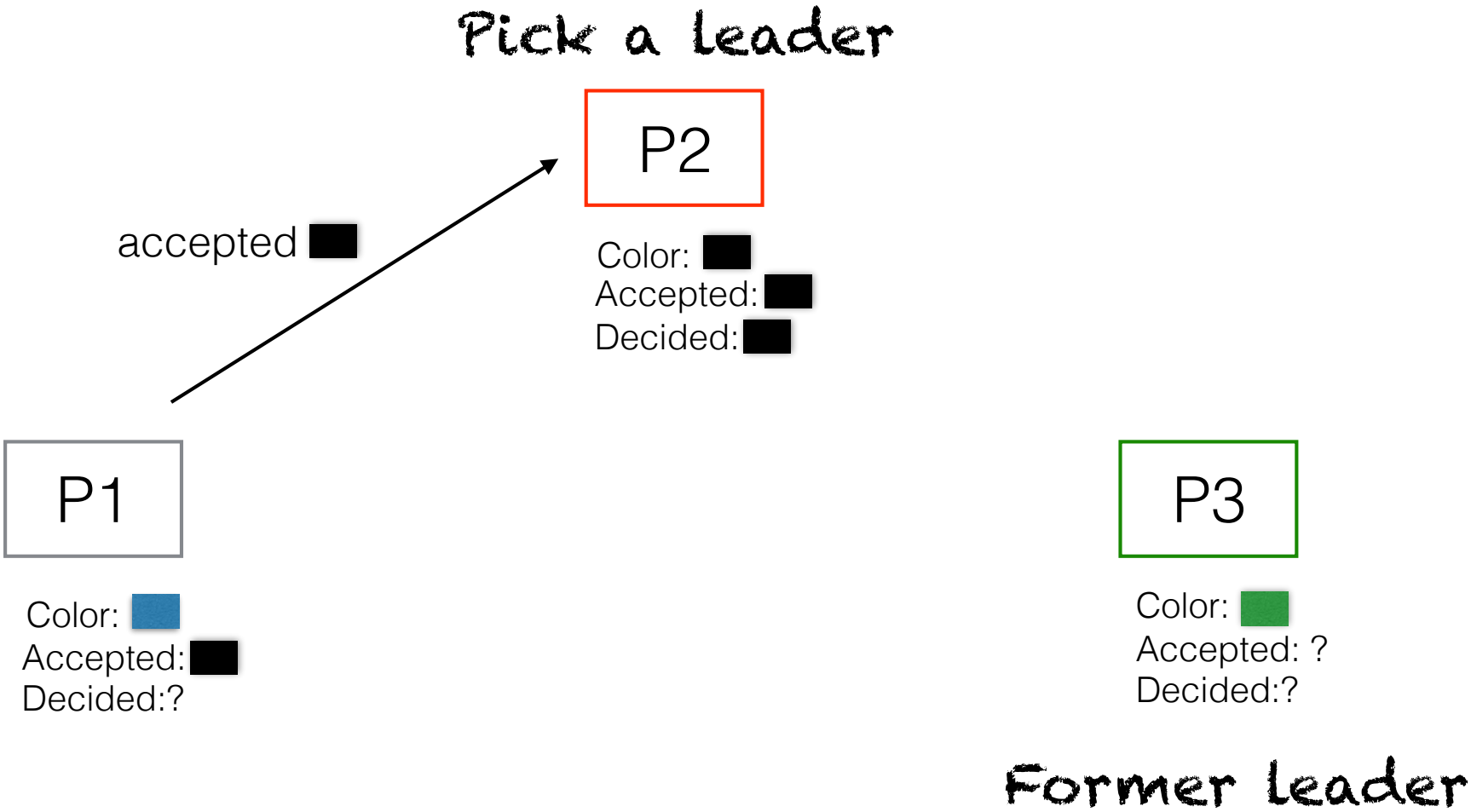
How do we get consensus?



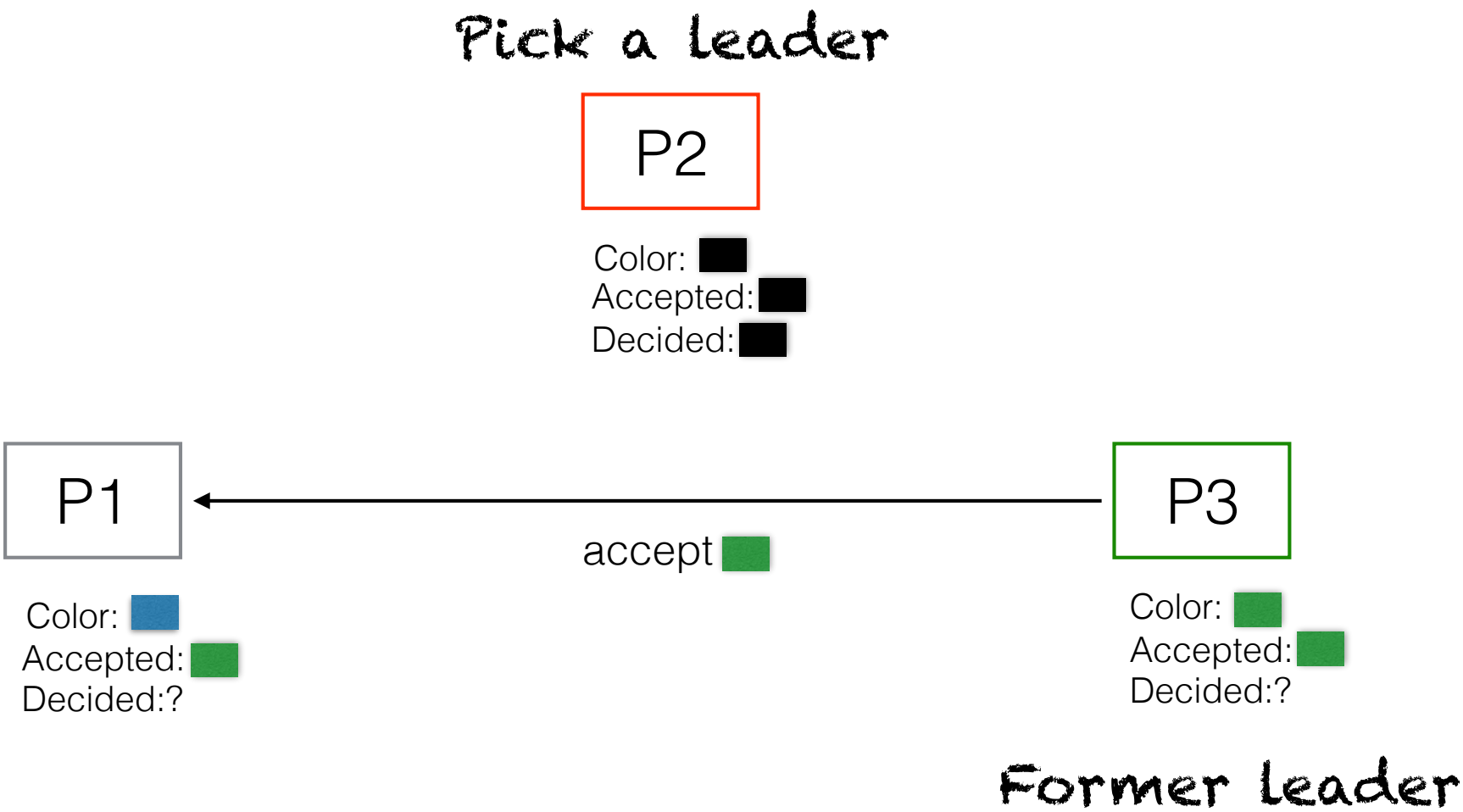
How do we get consensus?



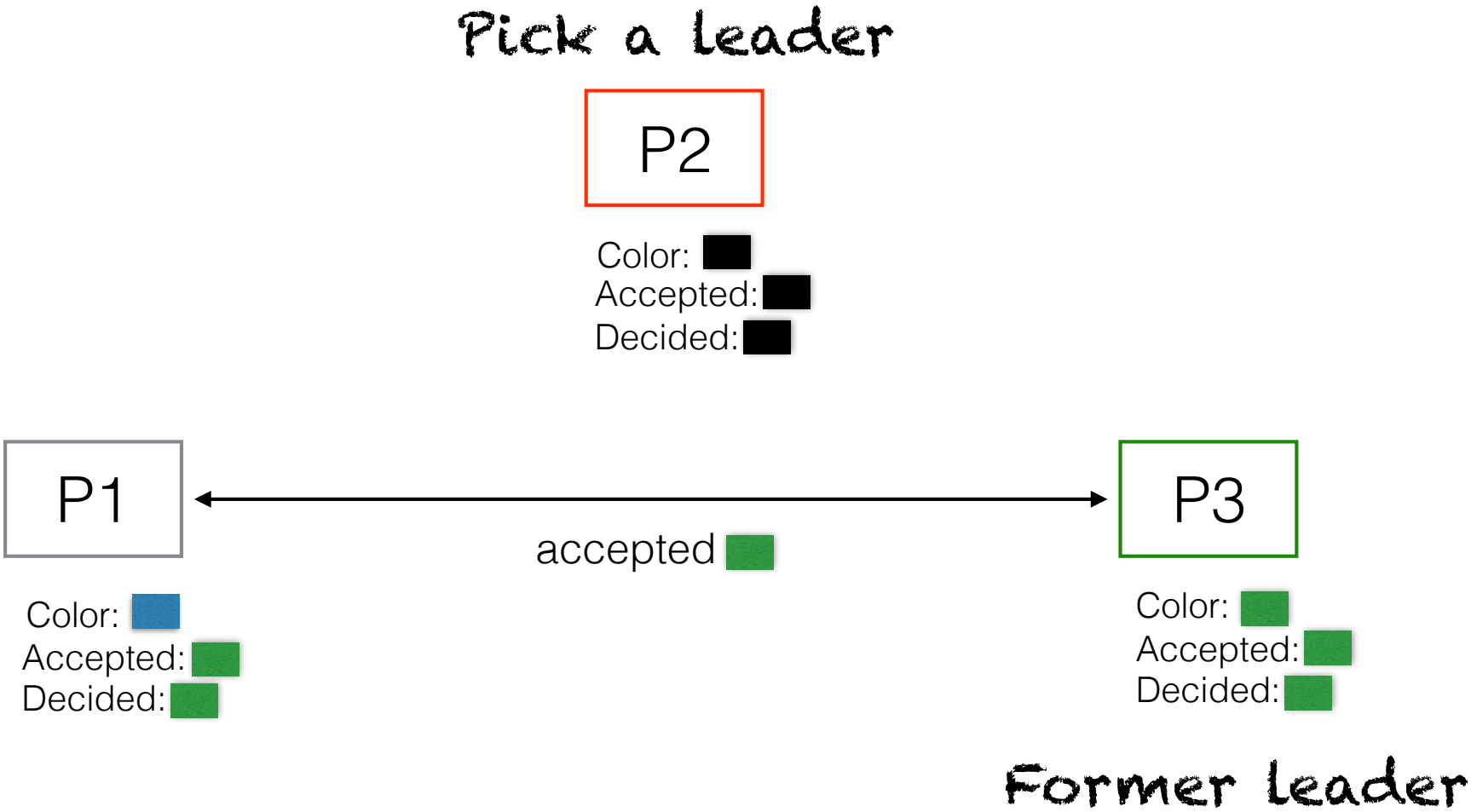
How do we get consensus?



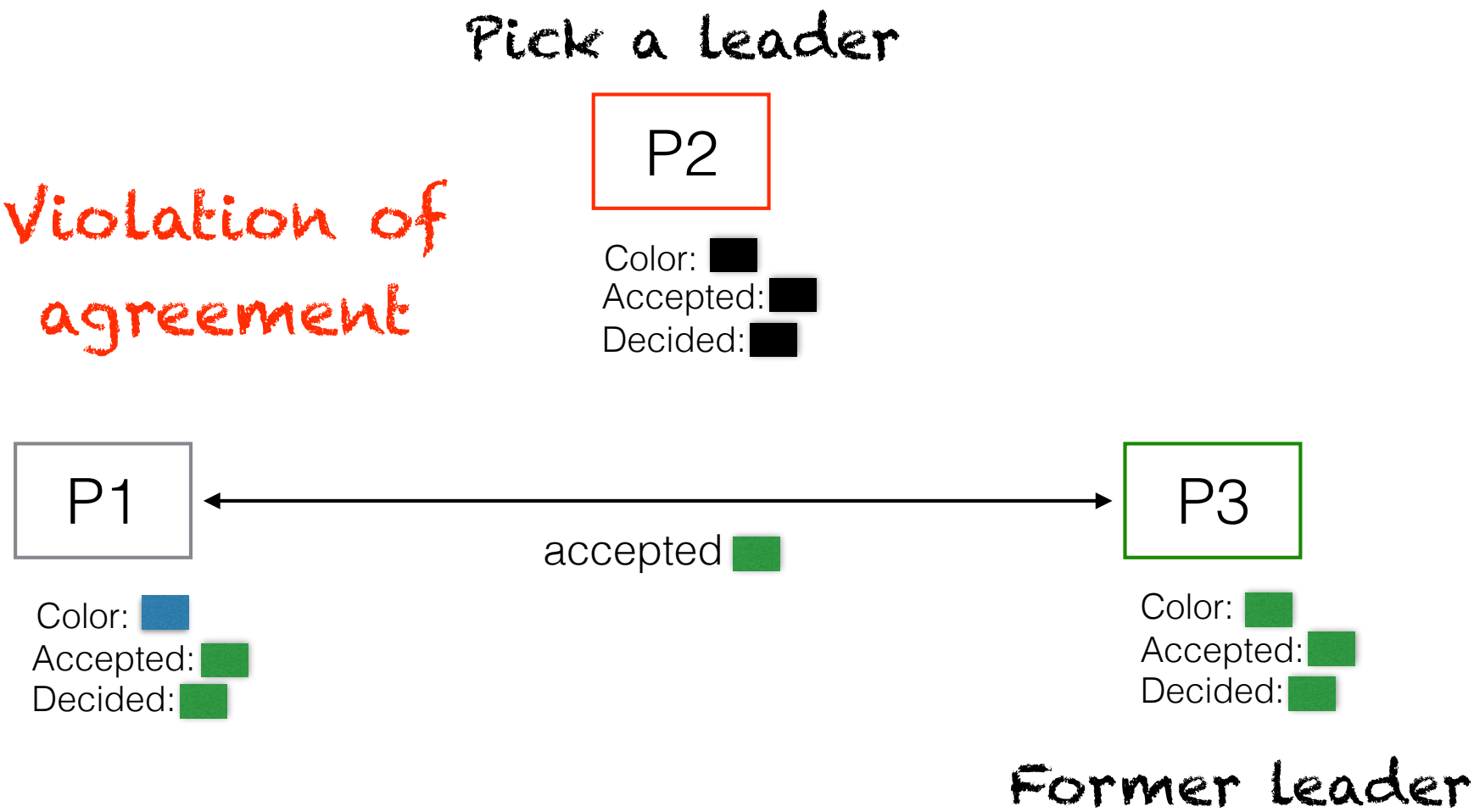
How do we get consensus?



How do we get consensus?

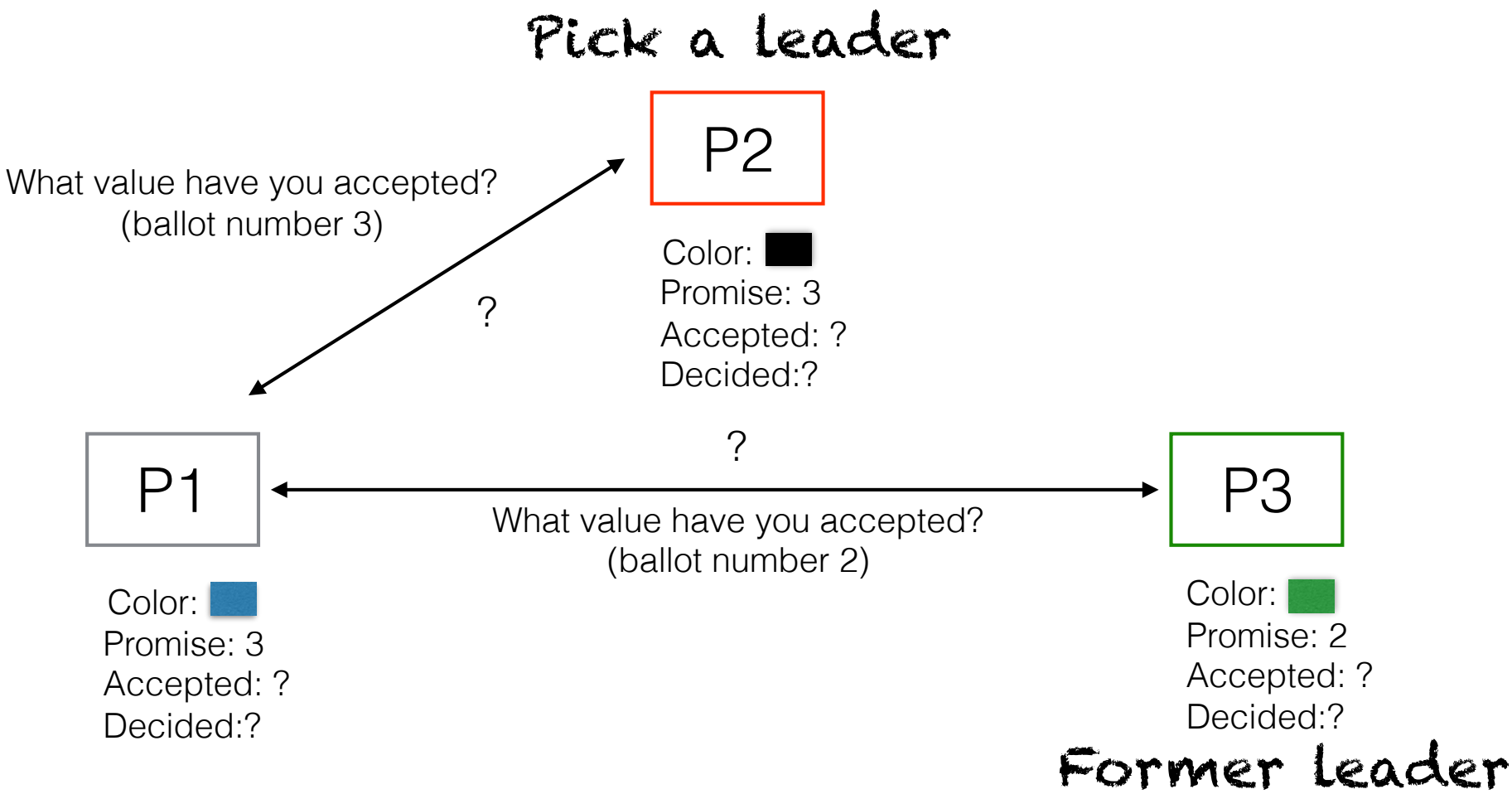


How do we get consensus?

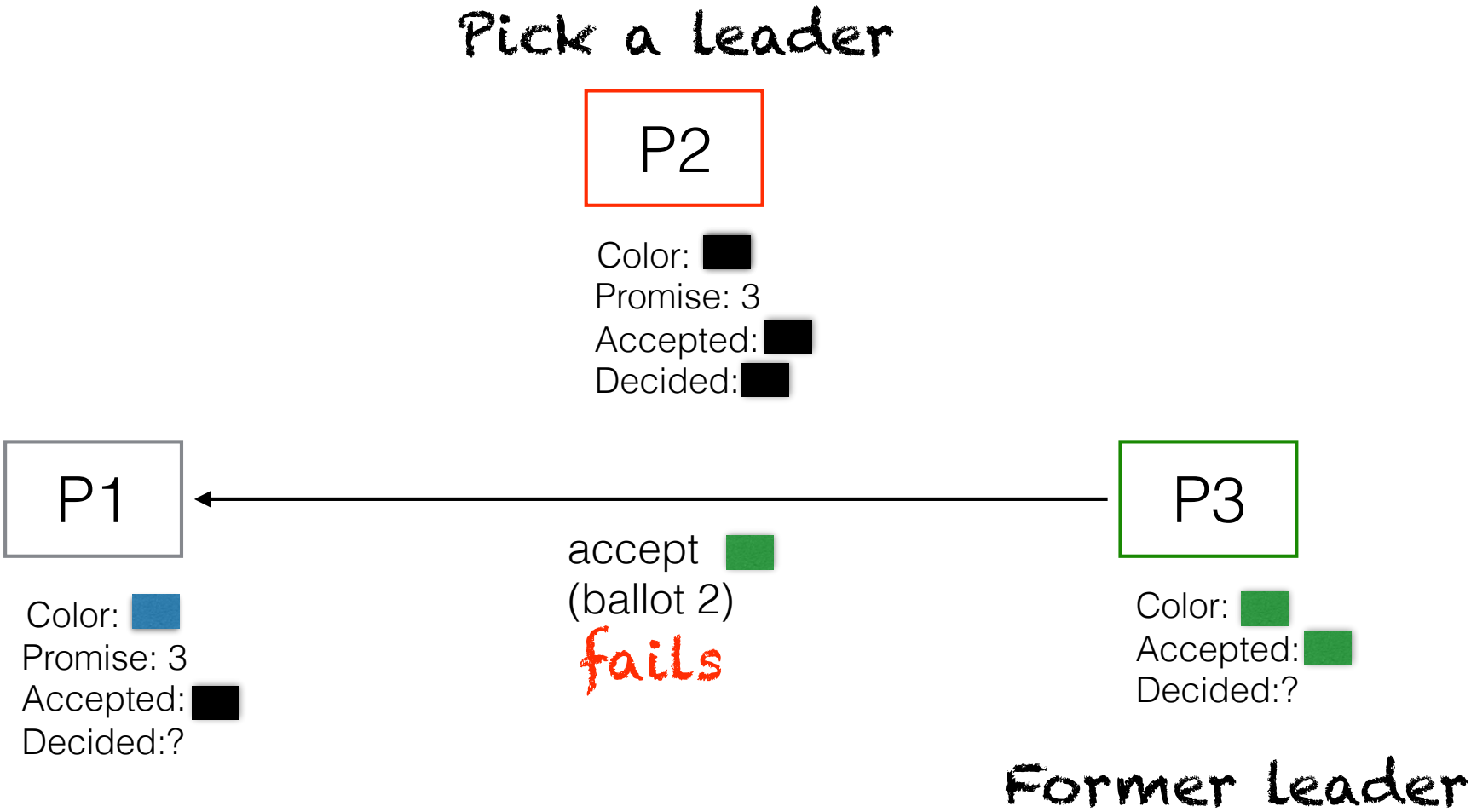


Fence (ballot number)

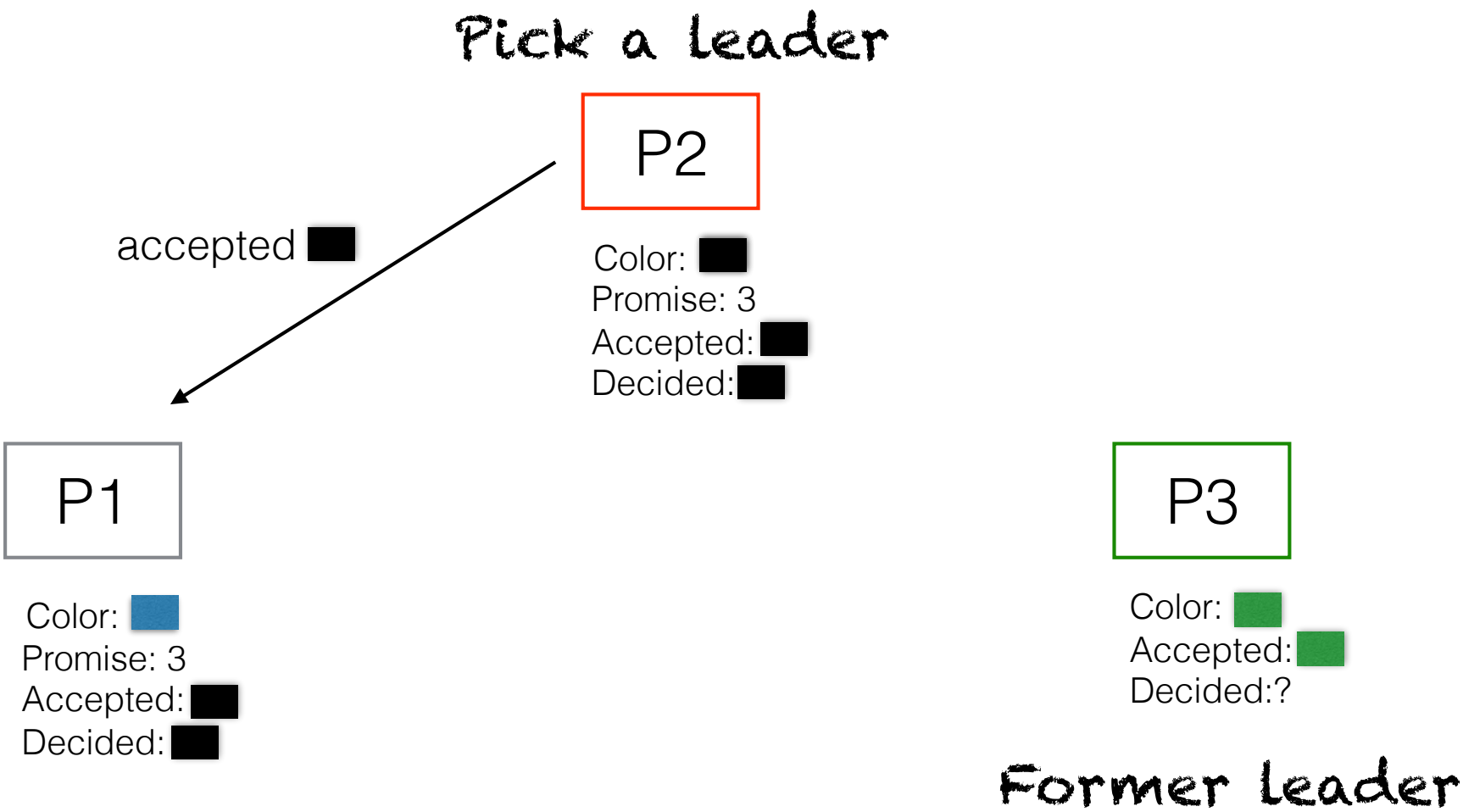
How do we get consensus?



How do we get consensus?



How do we get consensus?



Impossibility

- Message delays, network partitions, slow processes
  - ✓ Asynchronous systems
- Consensus revisited
  - ✓ Non-faulty processes agree on a value
  - ✓ The decision value must have been proposed
  - ✓ Decide eventually

[Fischer, Lynch, Patterson, Impossibility of distributed consensus with one faulty process, JACM, April 1985]

Impossibility

- Message delays, network partitions, slow processes
  - ✓ Asynchronous systems
- Consensus revisited
  - ✓ Non-faulty processes agree on a value
  - ✓ The decision value must have been proposed
  - ✓ Decide eventually ⬅ Termination

[Fischer, Lynch, Patterson, Impossibility of distributed consensus with one faulty process, JACM, April 1985]

Impossibility

P1

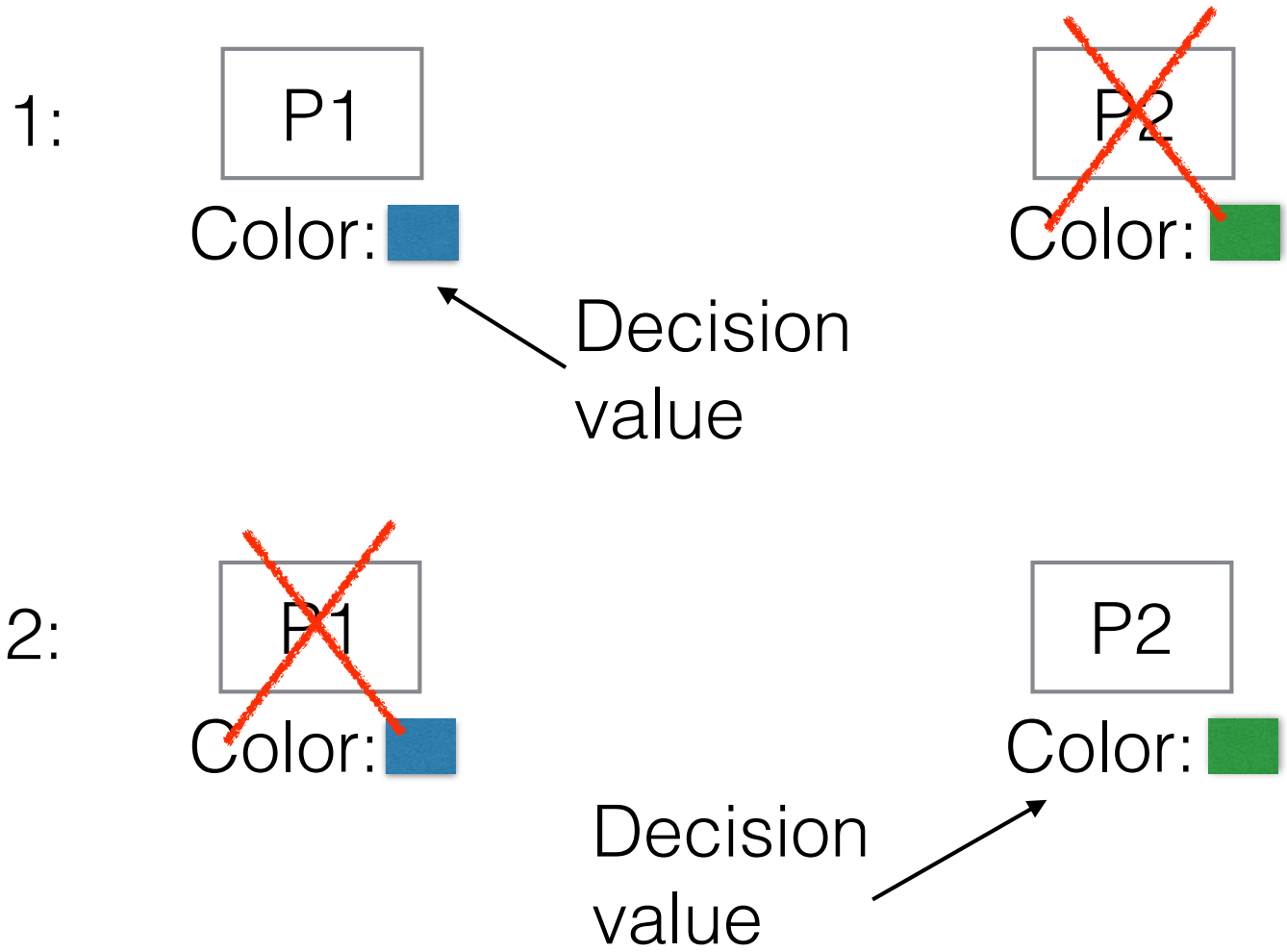
Color:

P2

Color:

Have two processes running a consensus protocol

Impossibility

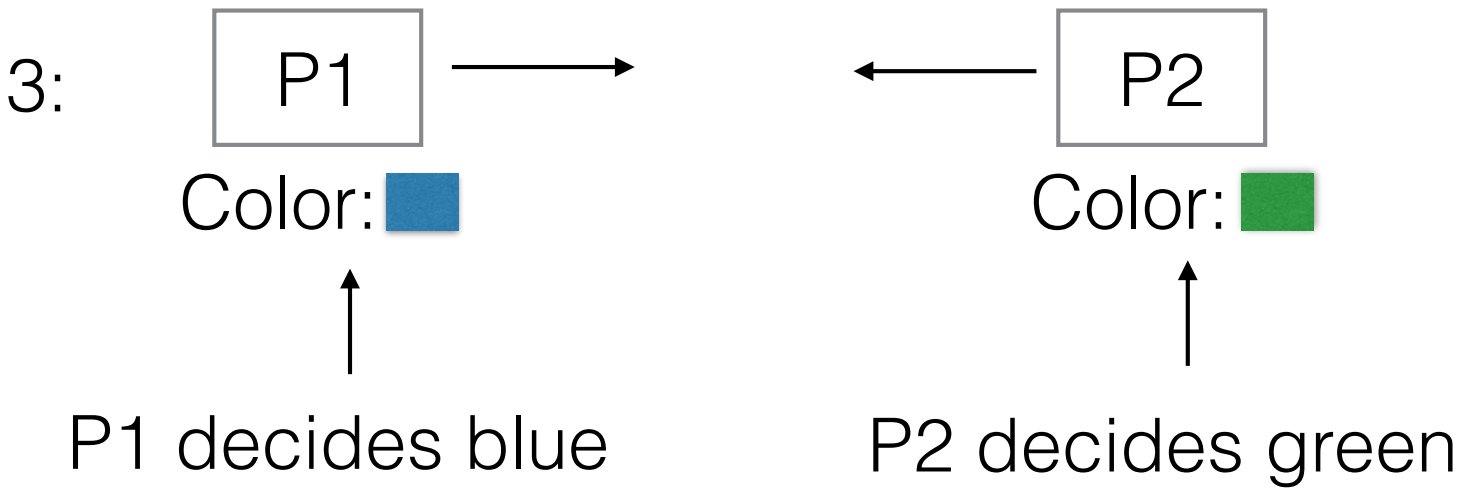


Impossibility

3:

Impossibility

Messages are delayed
or processes are slow.



Bottom line: can't distinguish slow
from crashed.

Possible

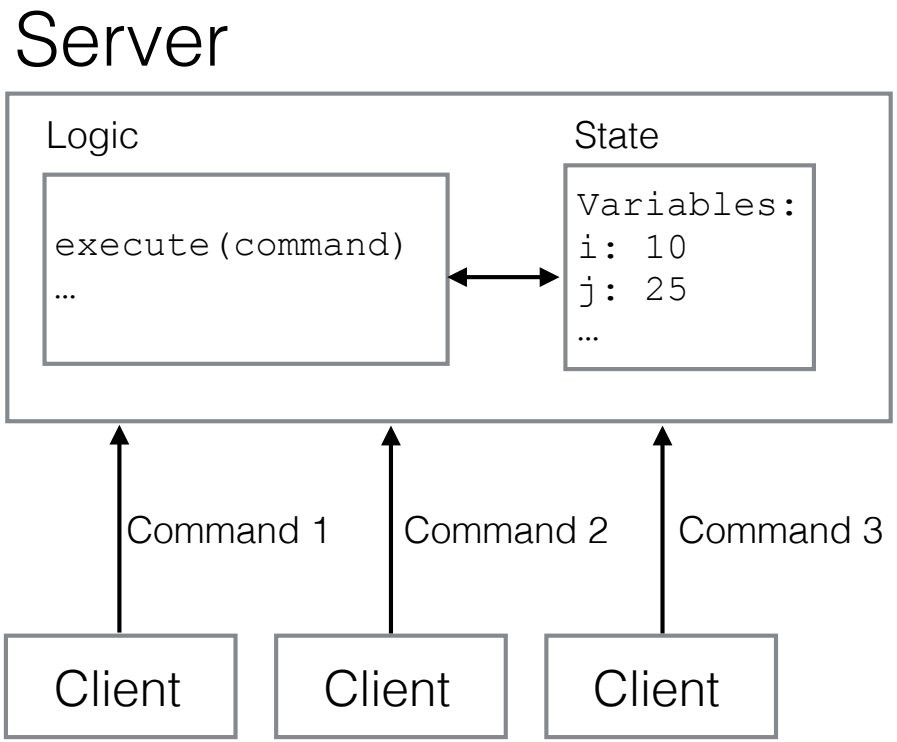
- Partially synchronous system
  - ✓ Asynchronous at times
  - ✓ Eventually stabilizes
- Able to elect a stable leader
- Have $2f + 1$ processes
  - ✓ f is the number of tolerated crashes
  - ✓ no Byzantine behavior in this presentation

State machine replication and Atomic broadcast

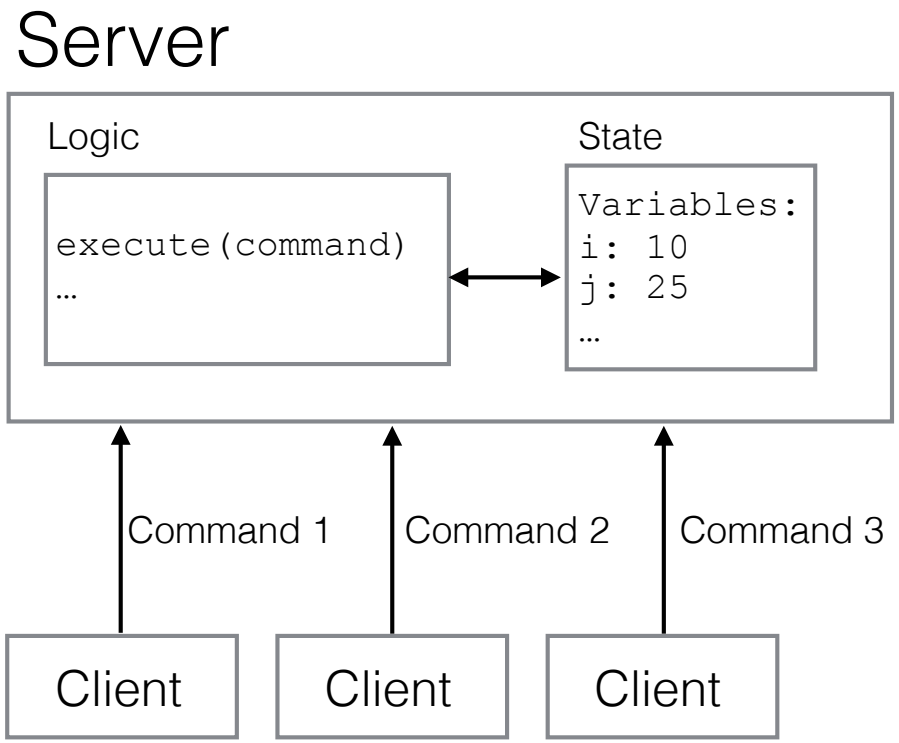
Consensus, Broadcast, and Replication

- Important application of distributed consensus
  - ✓ Replication
- State-machine replication
  - ✓ Agreement -> atomic broadcast
  - ✓ Execution -> deliver and execute requests

Replication

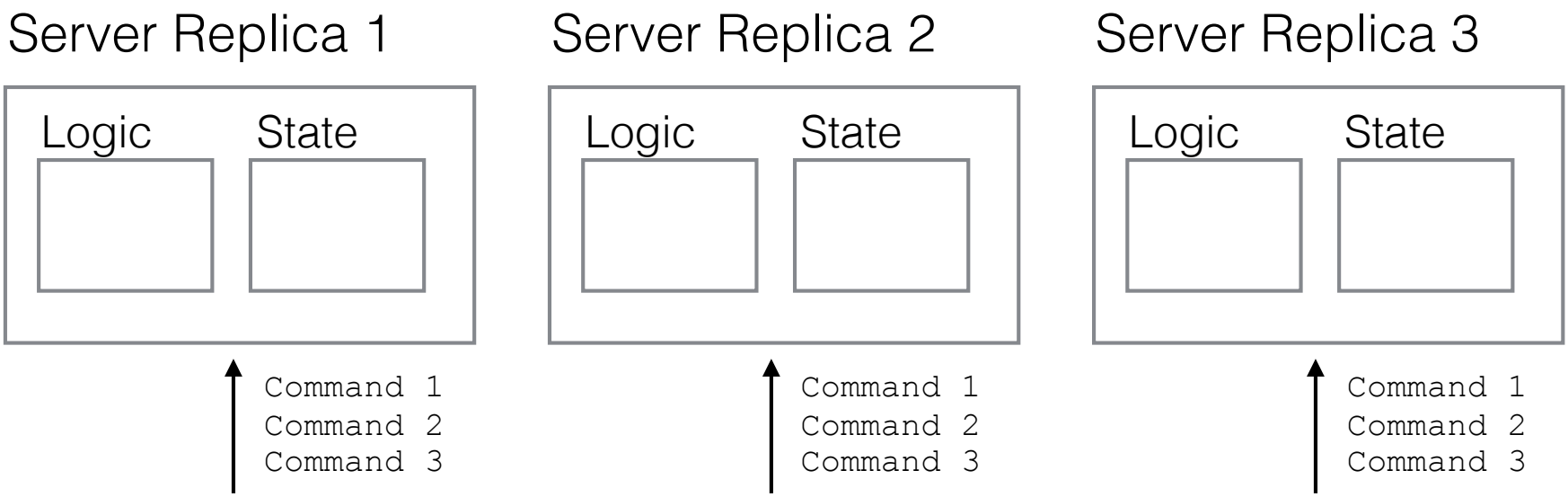


Replication

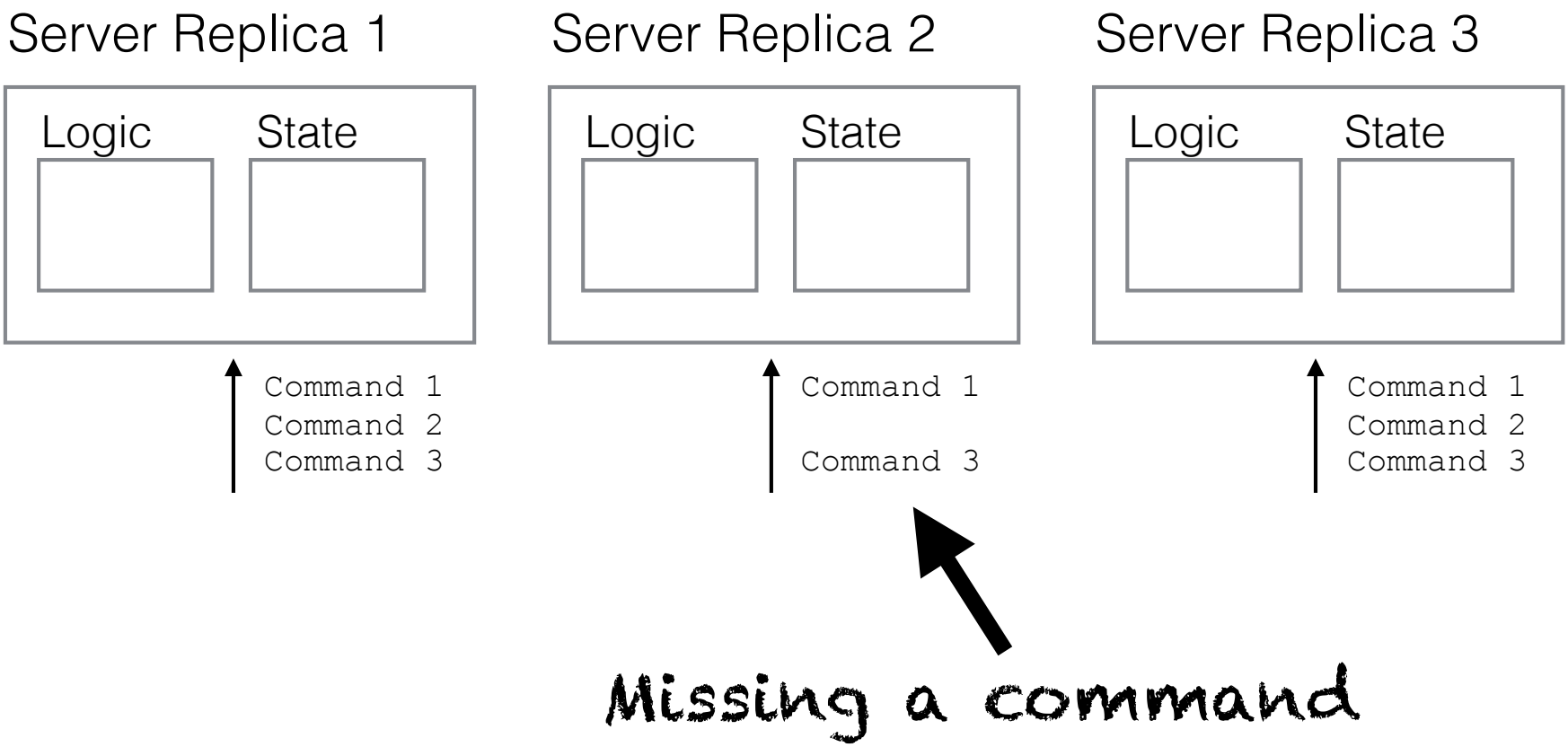


Logic and state captured as a deterministic state machine

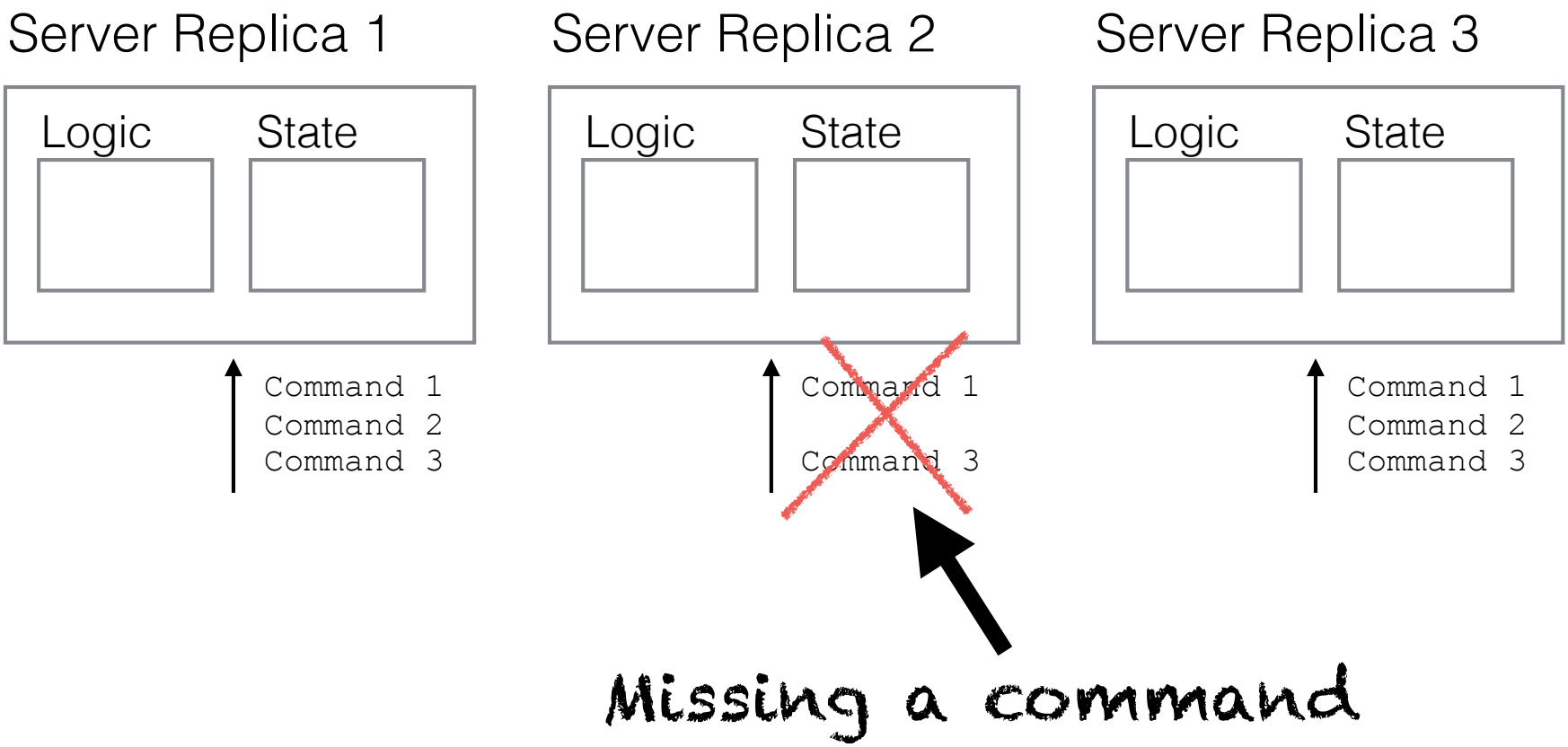
Replication - Consistency



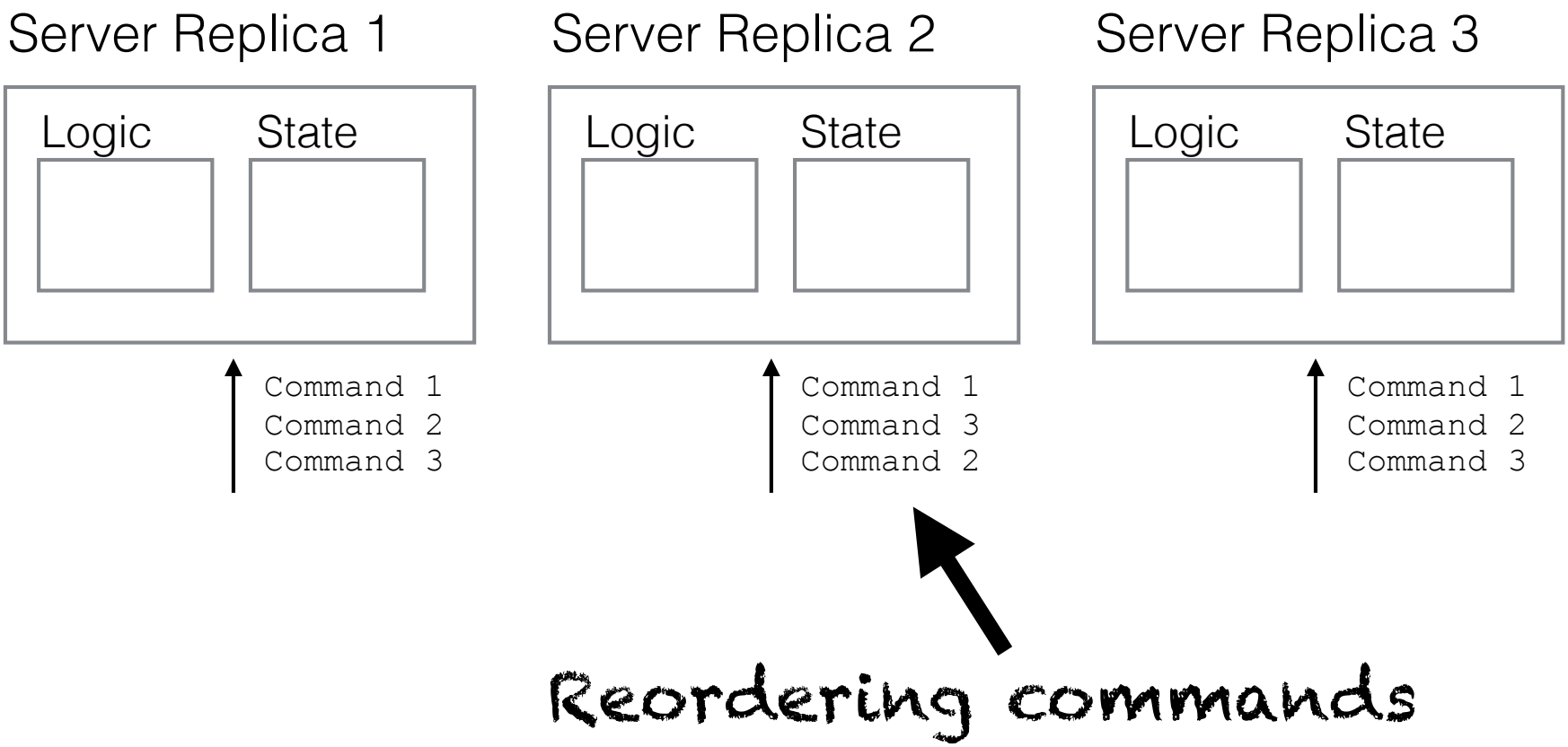
Replication - Consistency



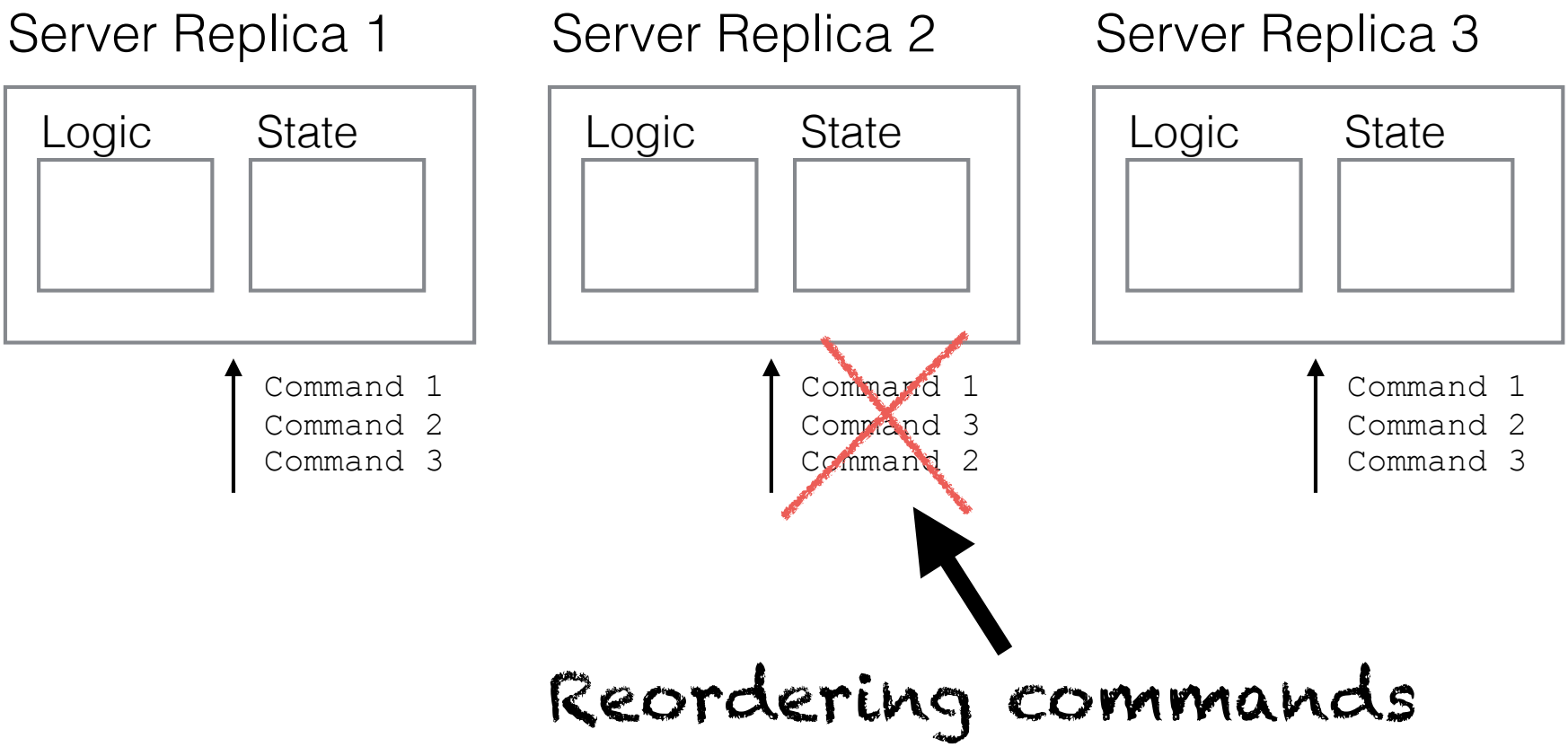
Replication - Consistency



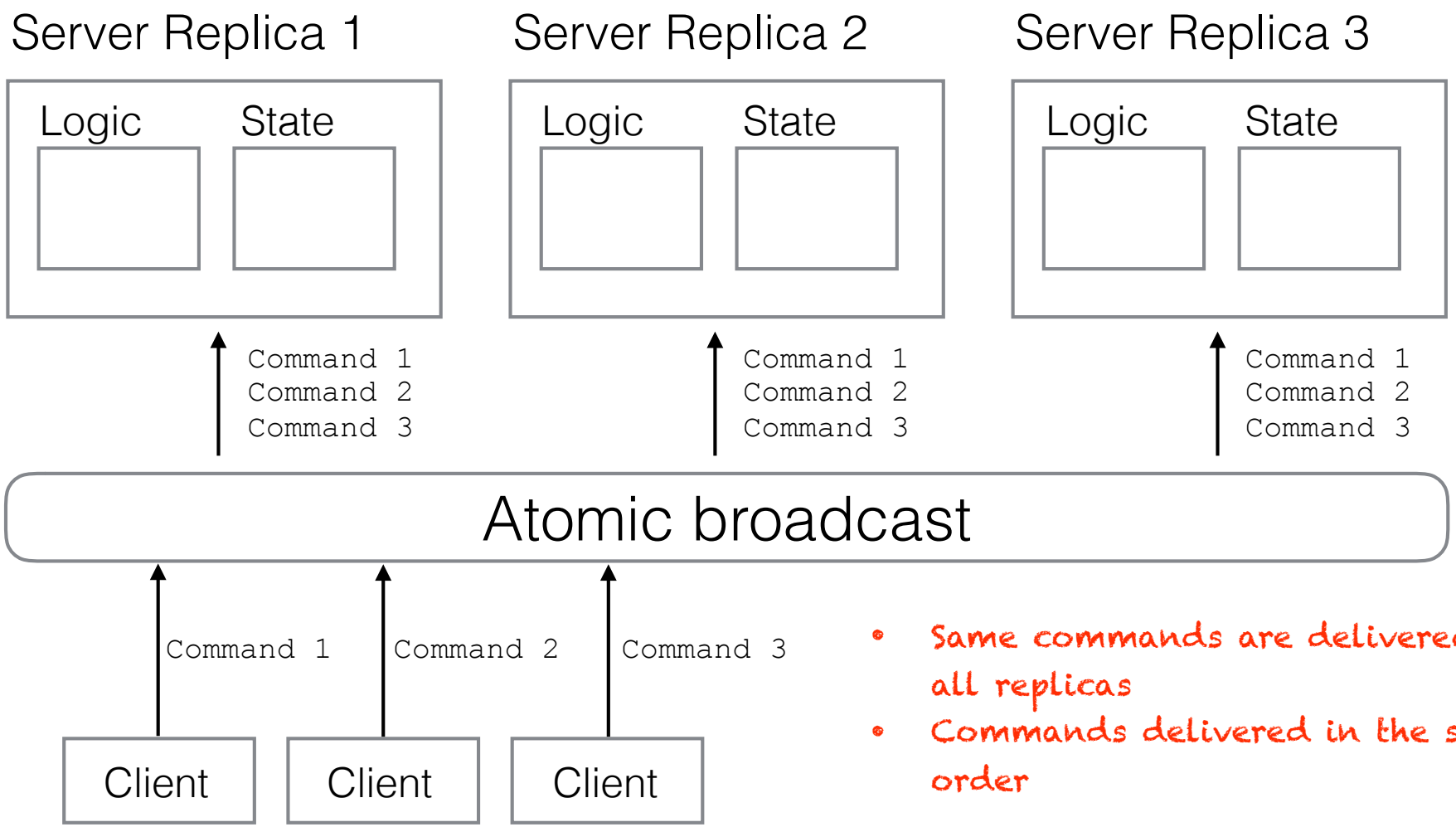
Replication - Consistency



Replication - Consistency

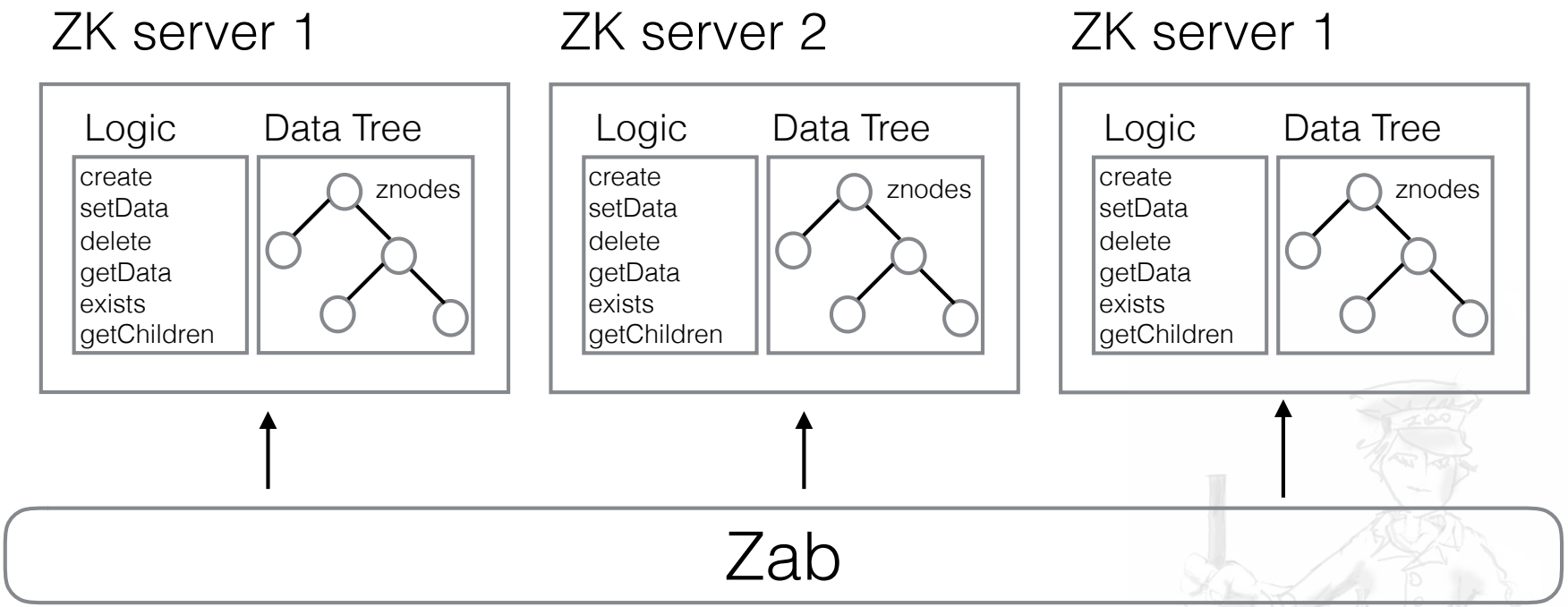


Replication - Consistency

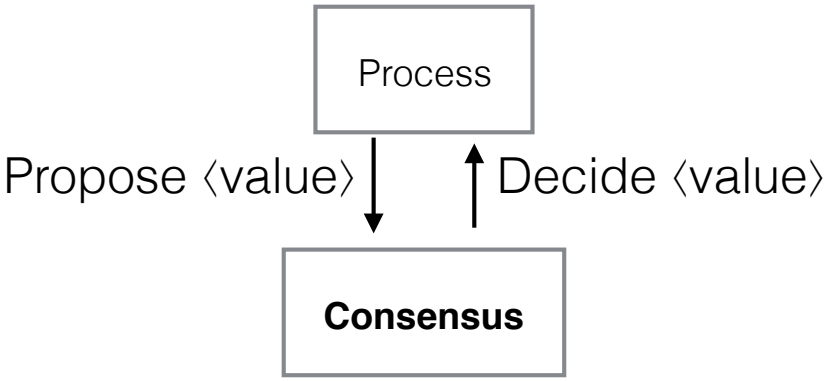


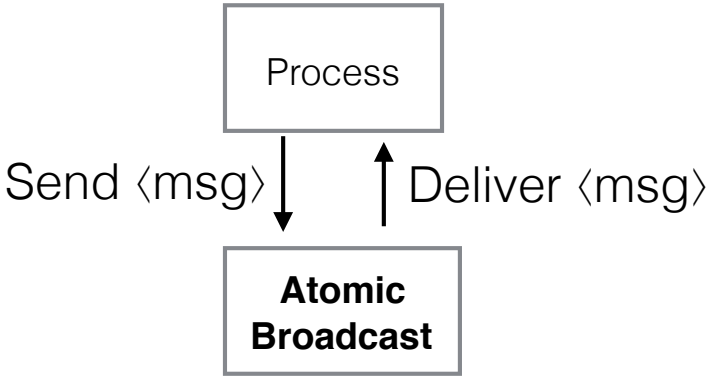
- Same commands are delivered to all replicas
- Commands delivered in the same order

Apache ZooKeeper



Atomic broadcast vs. Consensus



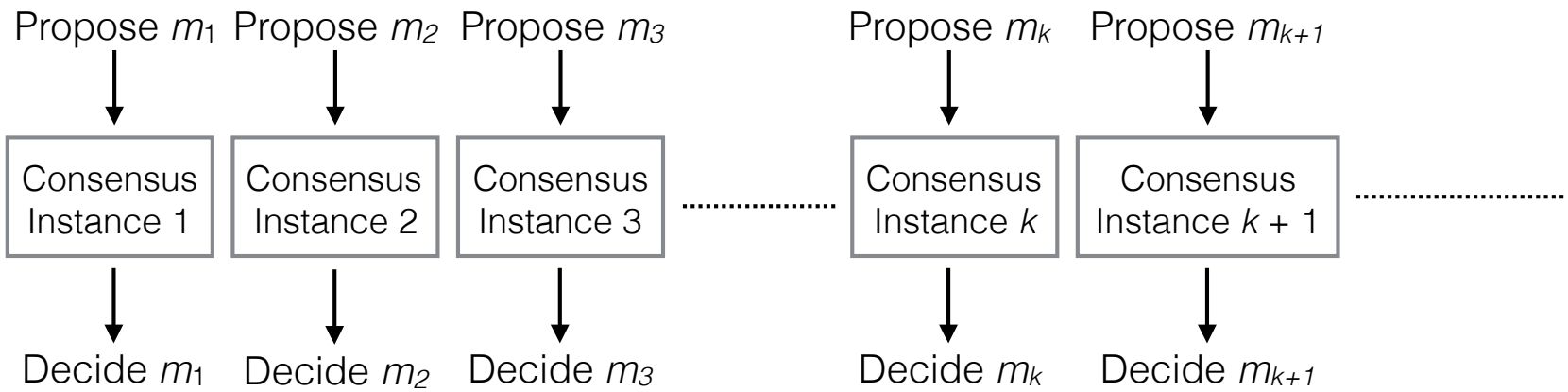


Properties:
1. Decide upon a single value
2. Decide upon a value proposed by some process
3. Eventually decide

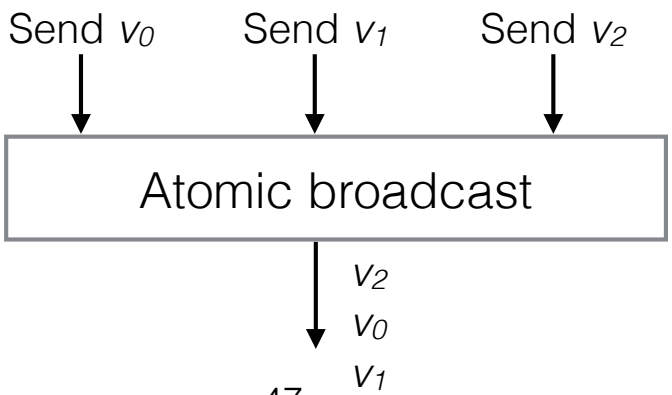
Properties:
1. Deliver the same messages to all processes
2. Deliver all messages in the same order

Atomic broadcast vs. Consensus

1- AB →Consensus

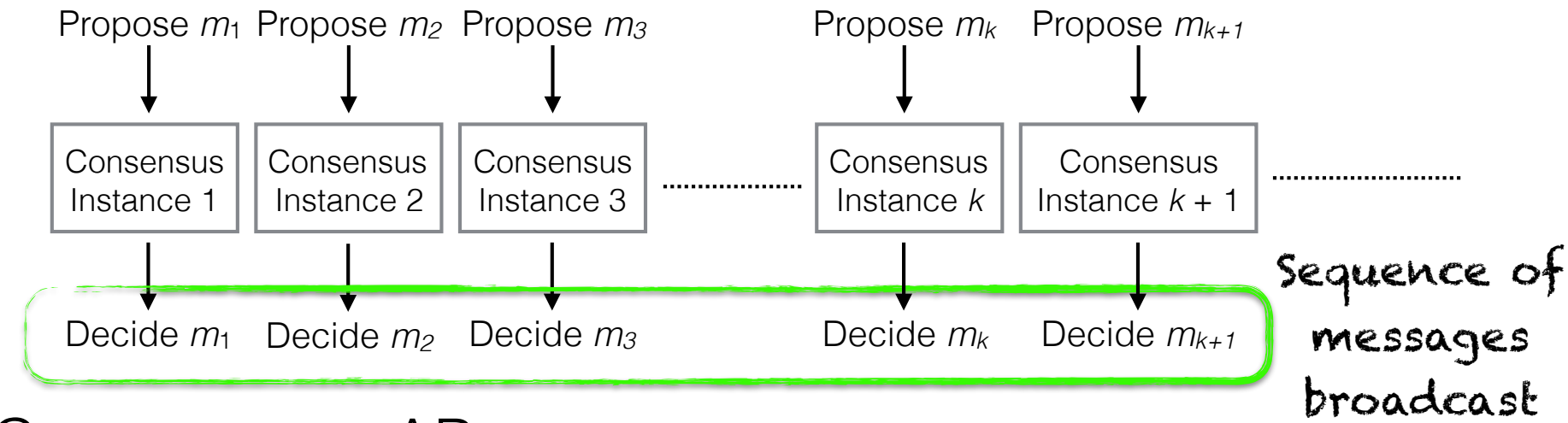


2- Consensus →AB

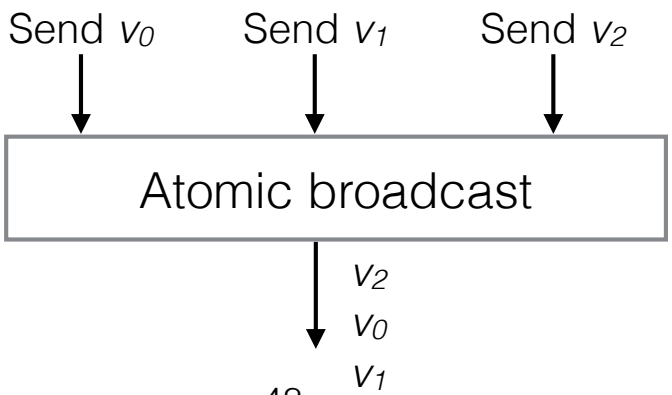


Atomic broadcast vs. Consensus

1- AB →Consensus

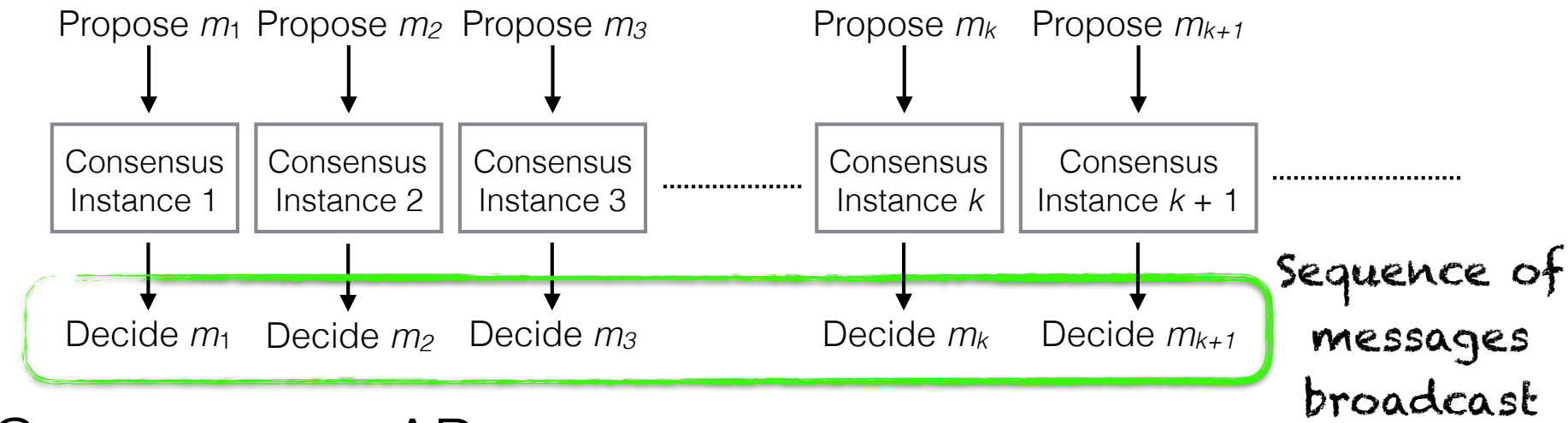


2- Consensus →AB

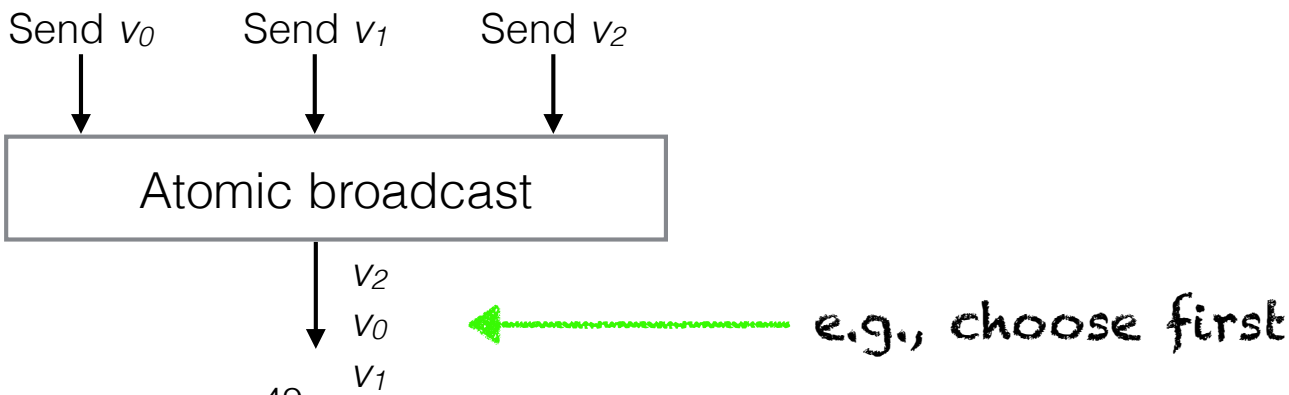


Atomic broadcast vs. Consensus

1- AB →Consensus

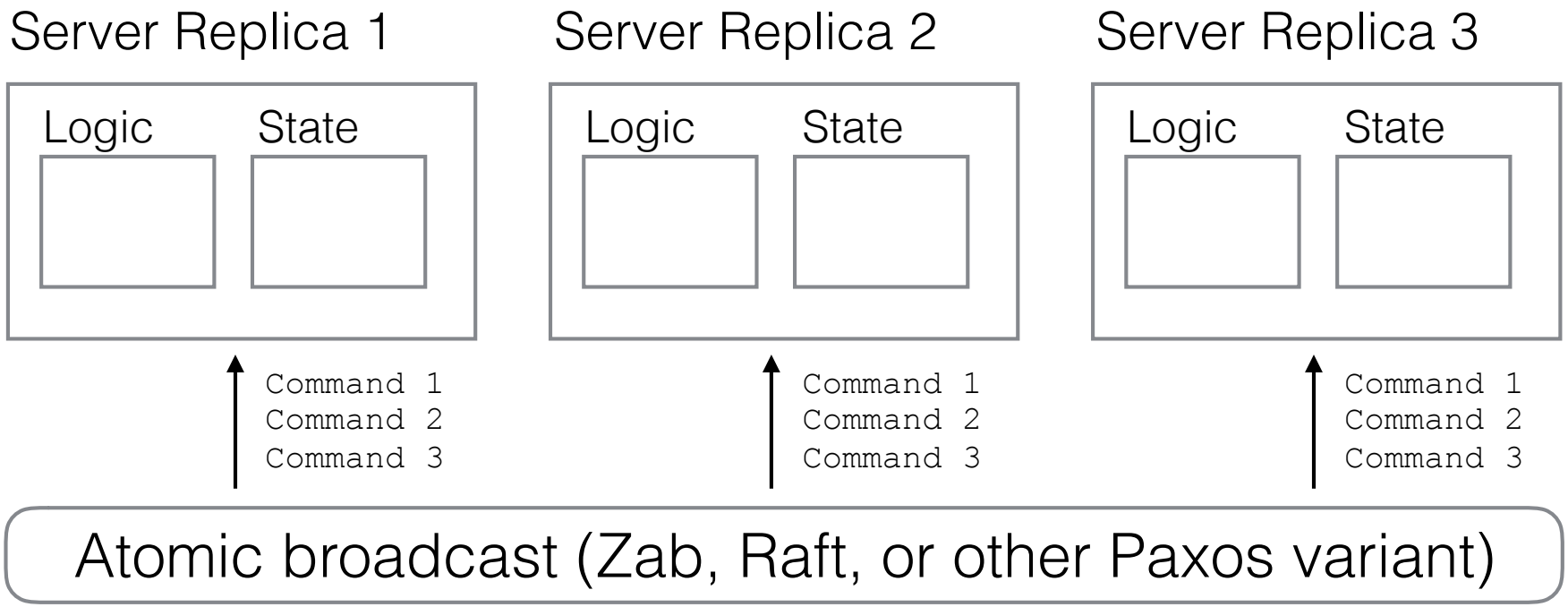


2- Consensus →AB



How do I replicate my own system?

From scratch…

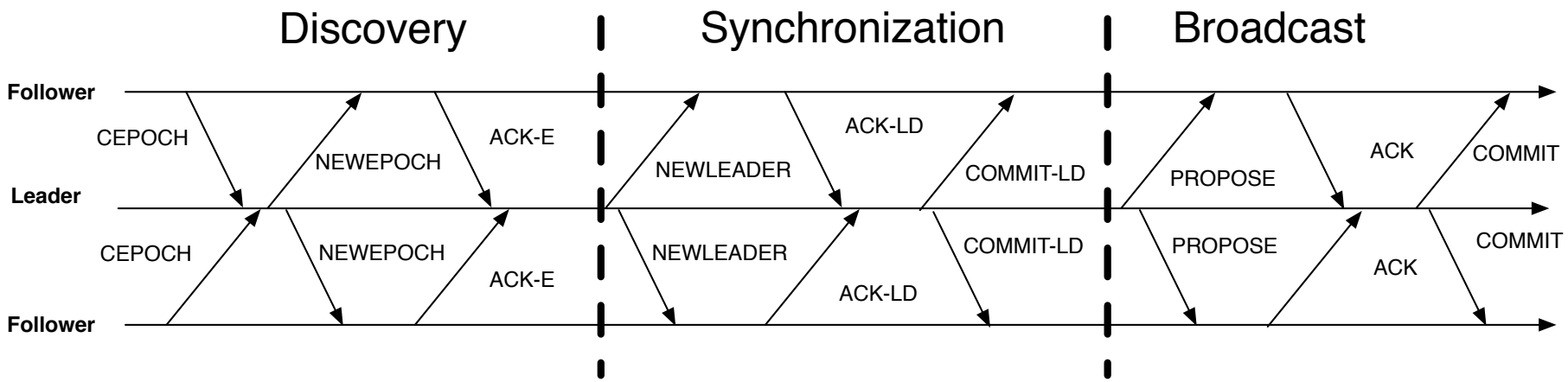


- Quorum-based replication, typically majority
- Reconfiguration internal to the protocol

Zab Flow

CEPOCH = Follower sends its last promise to the prospective leader
NEWEPOCH = Leader proposes a new epoch *e′*
ACK-E = Follower acknowledges the new epoch proposal
NEWLEADER = Prospective leader proposes itself as the new leader of epoch *e′*
ACK-LD = Follower acknowledges the new leader proposal
COMMIT-LD = Commit new leader proposal

PROPOSE = Leader proposes a new transaction
ACK = Follower acknowledges leader proosal
COMMIT = Leader commits proposal



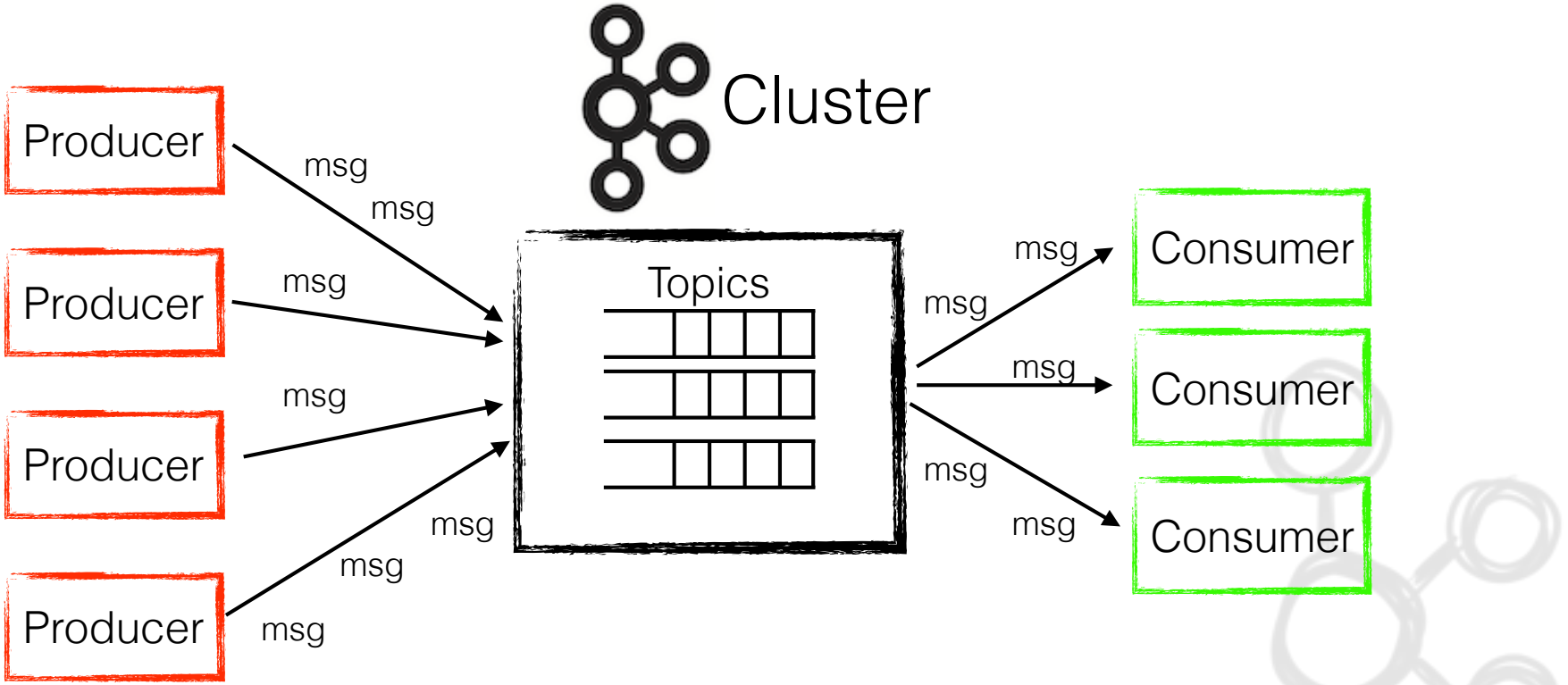
Using a configuration master

Apache Kafka

Apache BookKeeper

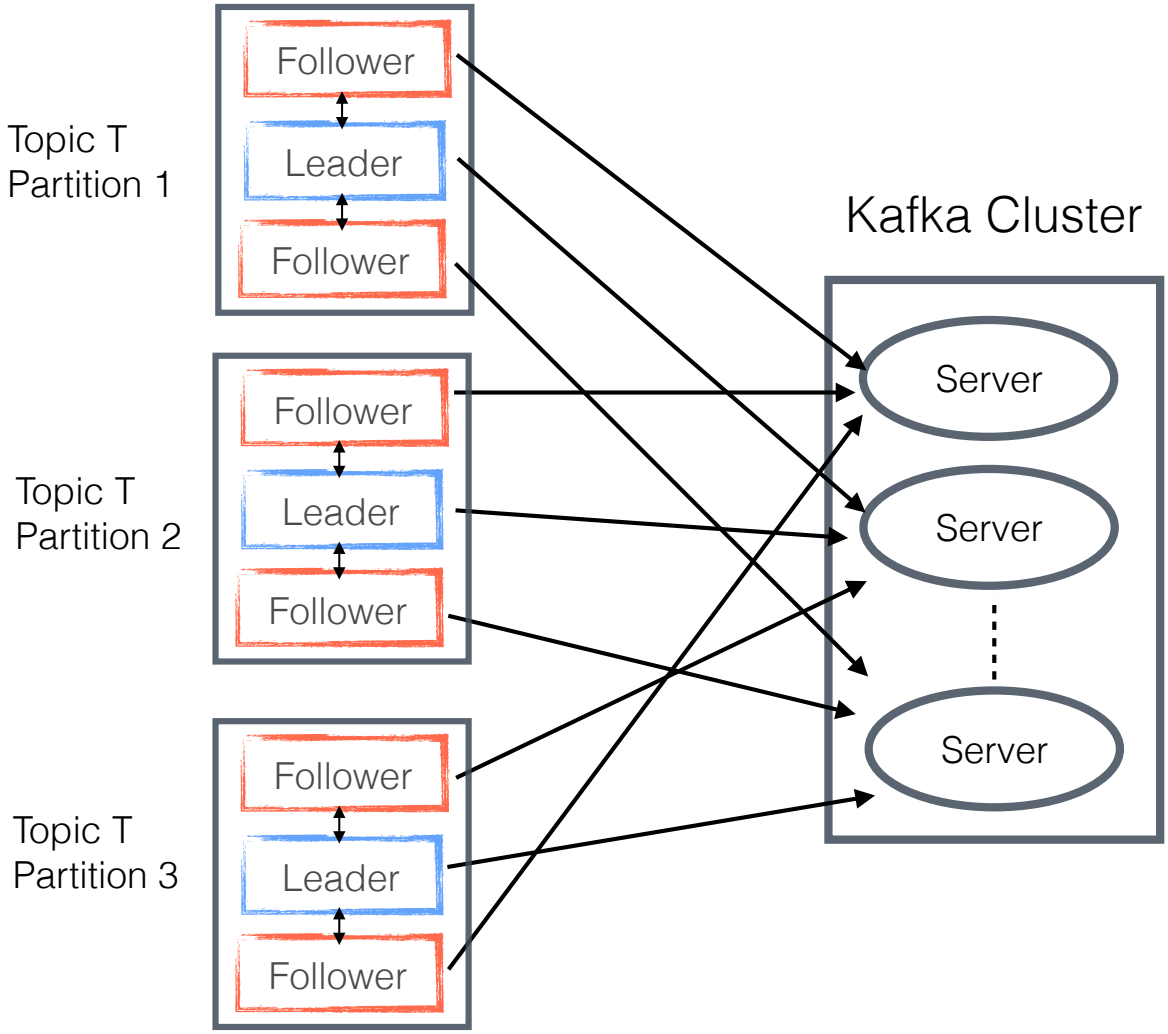
[L. Lamport, D. Malkhi, L. Zhou, Vertical paxos and primary-backup replication,PODC 2009]

Apache Kafka - 10,000 ft



Apache Kafka - Replication

- Topics
- Partitions
- Replication



Partition replication

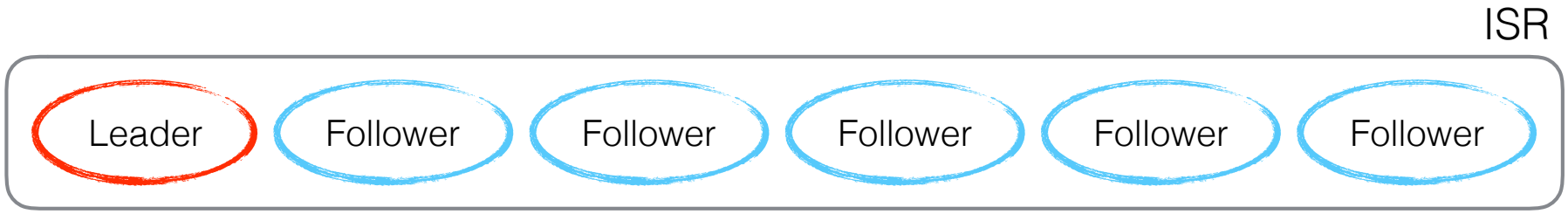
6 replicas

Partition replication

6 replicas
ISR - In-sync replicas

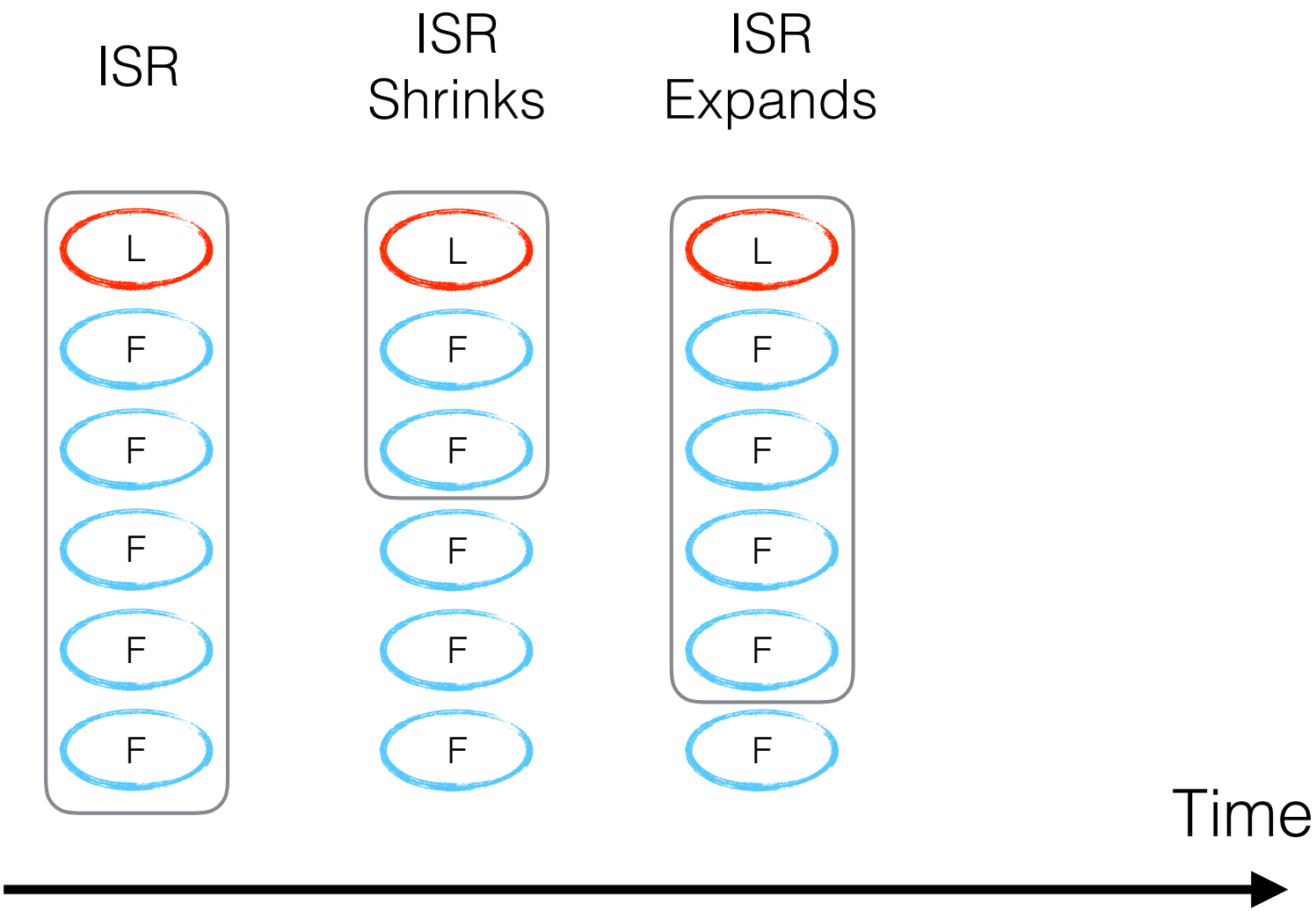
Partition replication

6 replicas
ISR - In-sync replicas

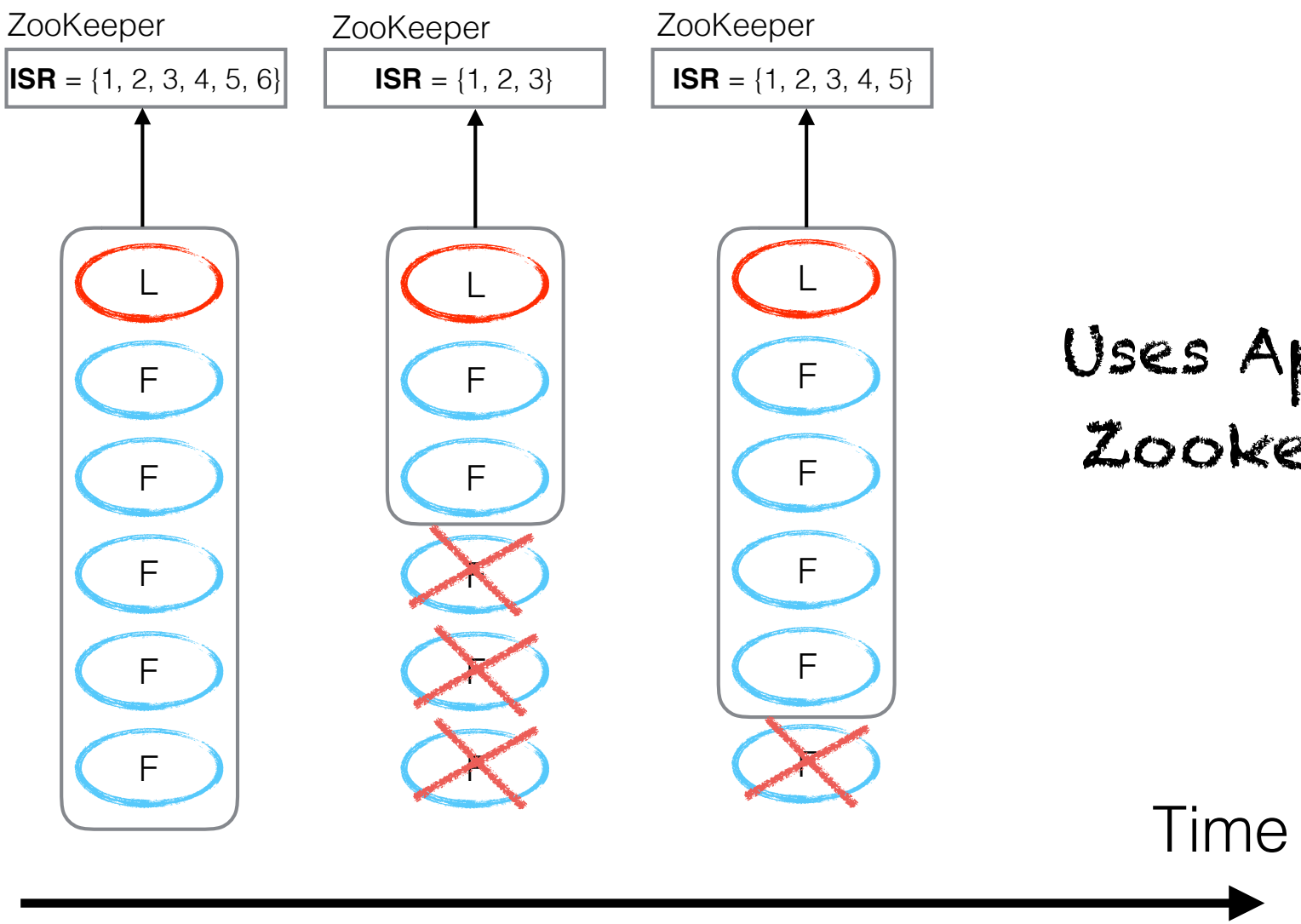


Set of replicas that must reply before commit

Partition replication

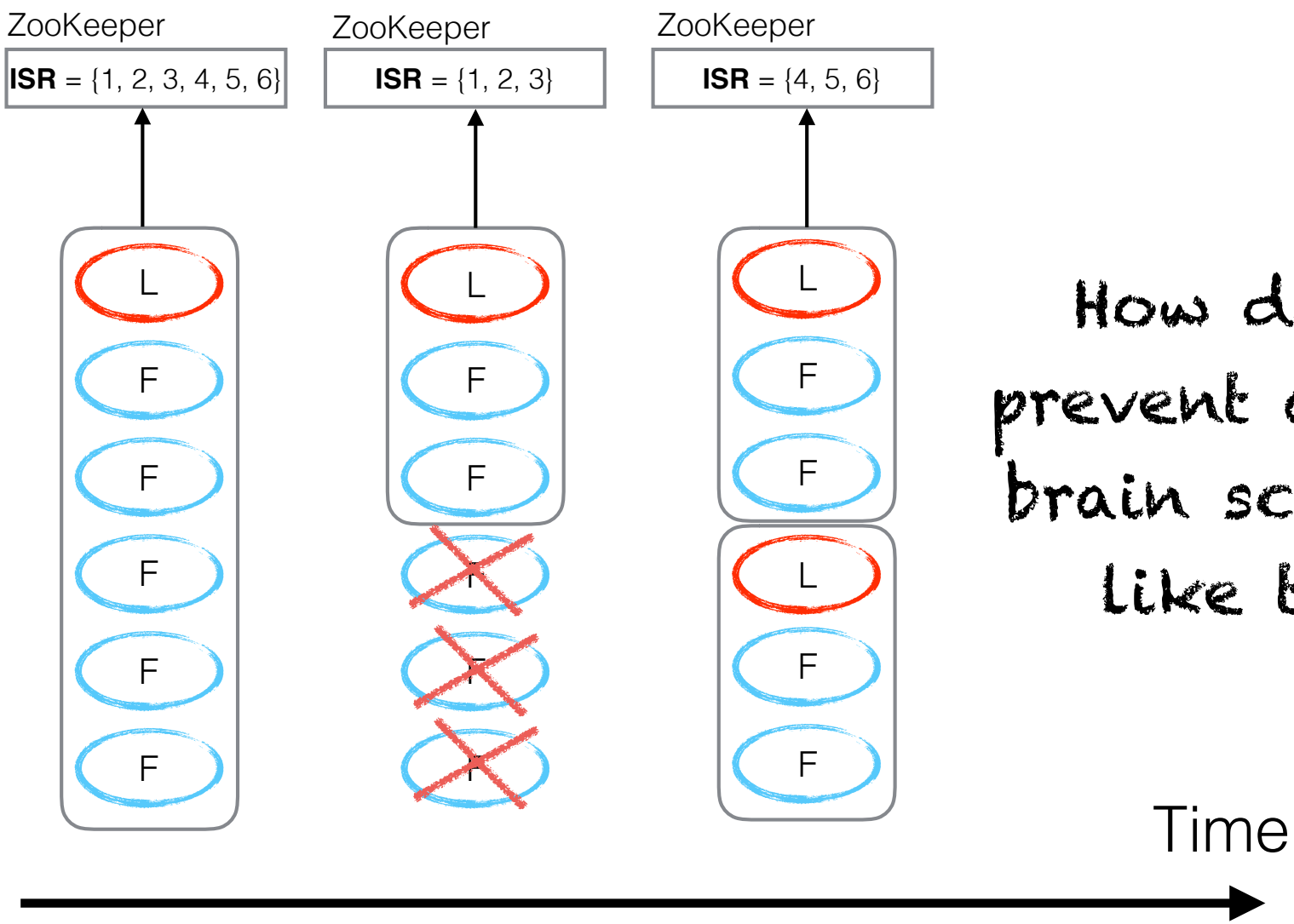


Partition replication



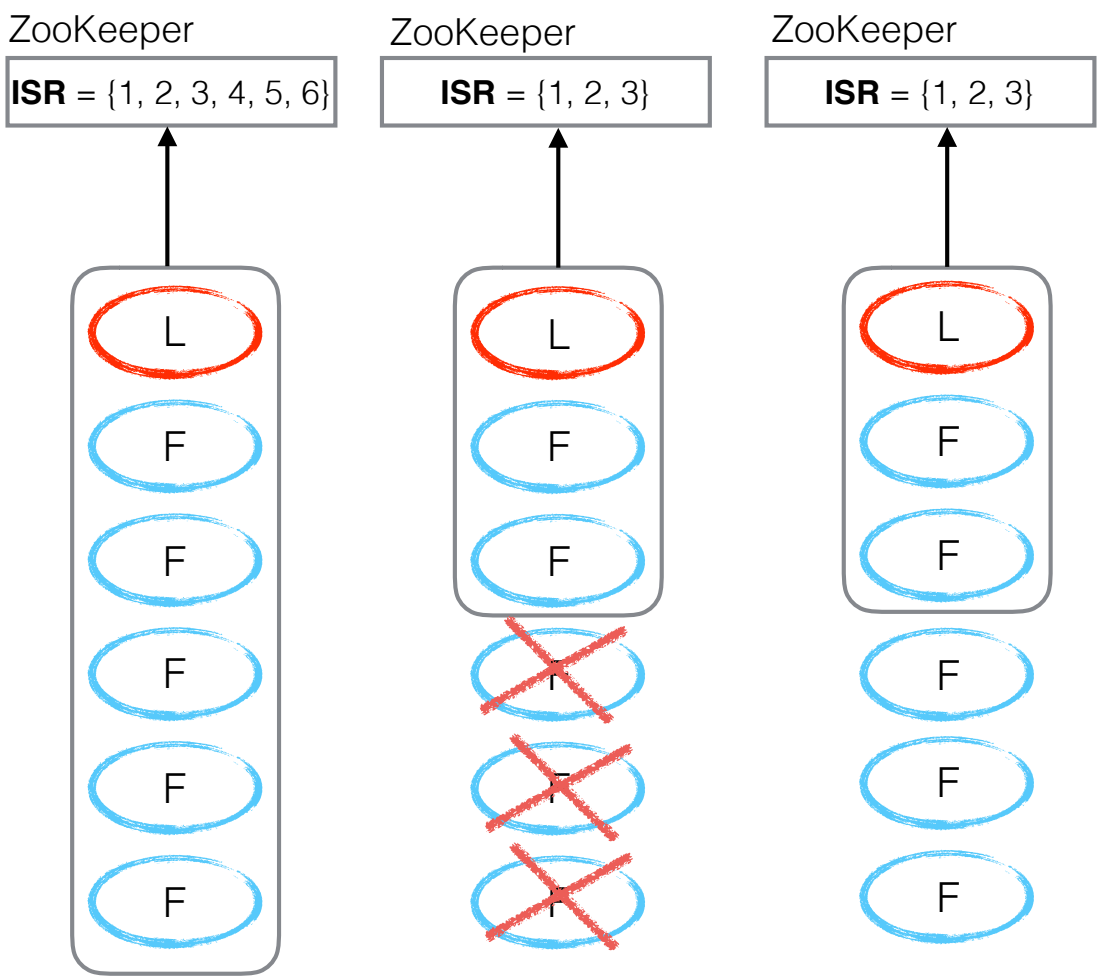
Uses Apache Zookeeper

Partition replication



How do we prevent a split-brain scenario like this?

Partition replication

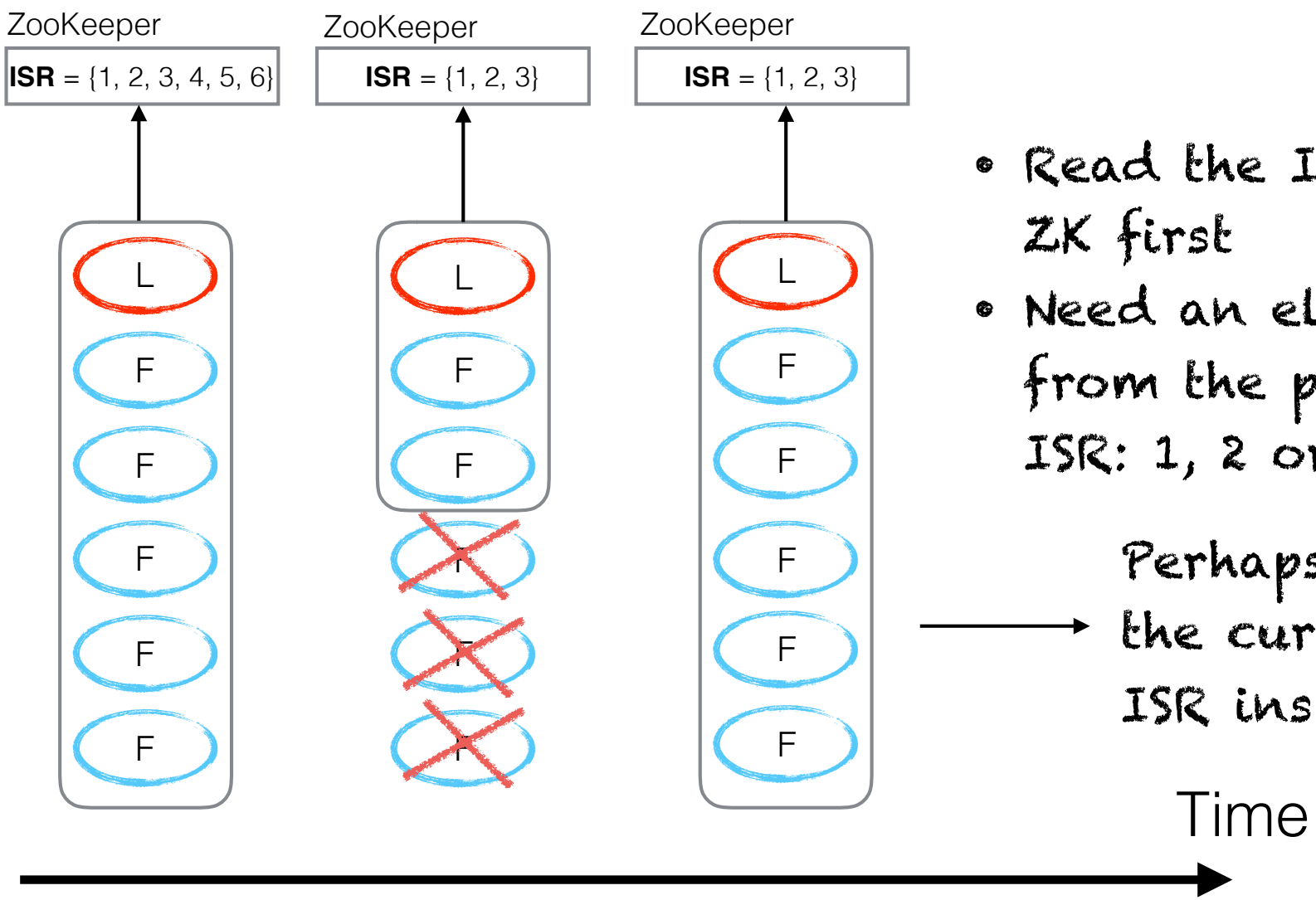


- Read the ISR from ZK first
- Need an element from the previous ISR: 1, 2 or 3

Cannot become the new ISR

Time

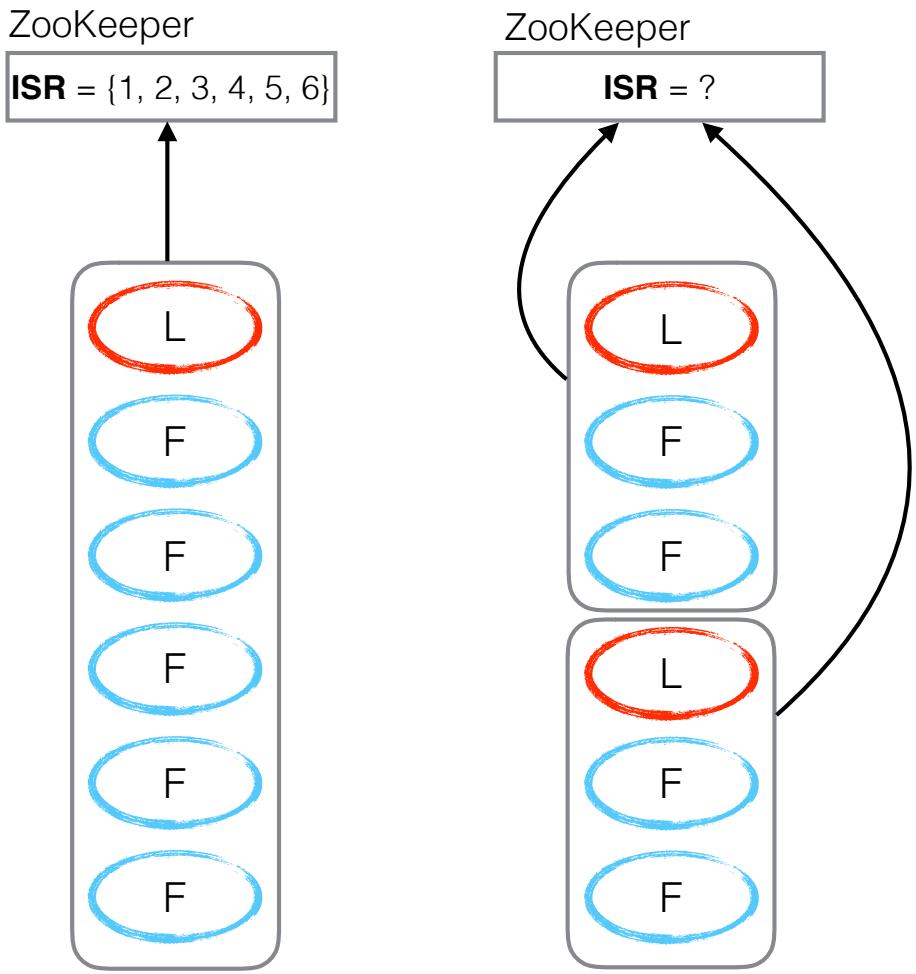
Partition replication



- Read the ISR from ZK first
- Need an element from the previous ISR: 1, 2 or 3

Perhaps join the current ISR instead

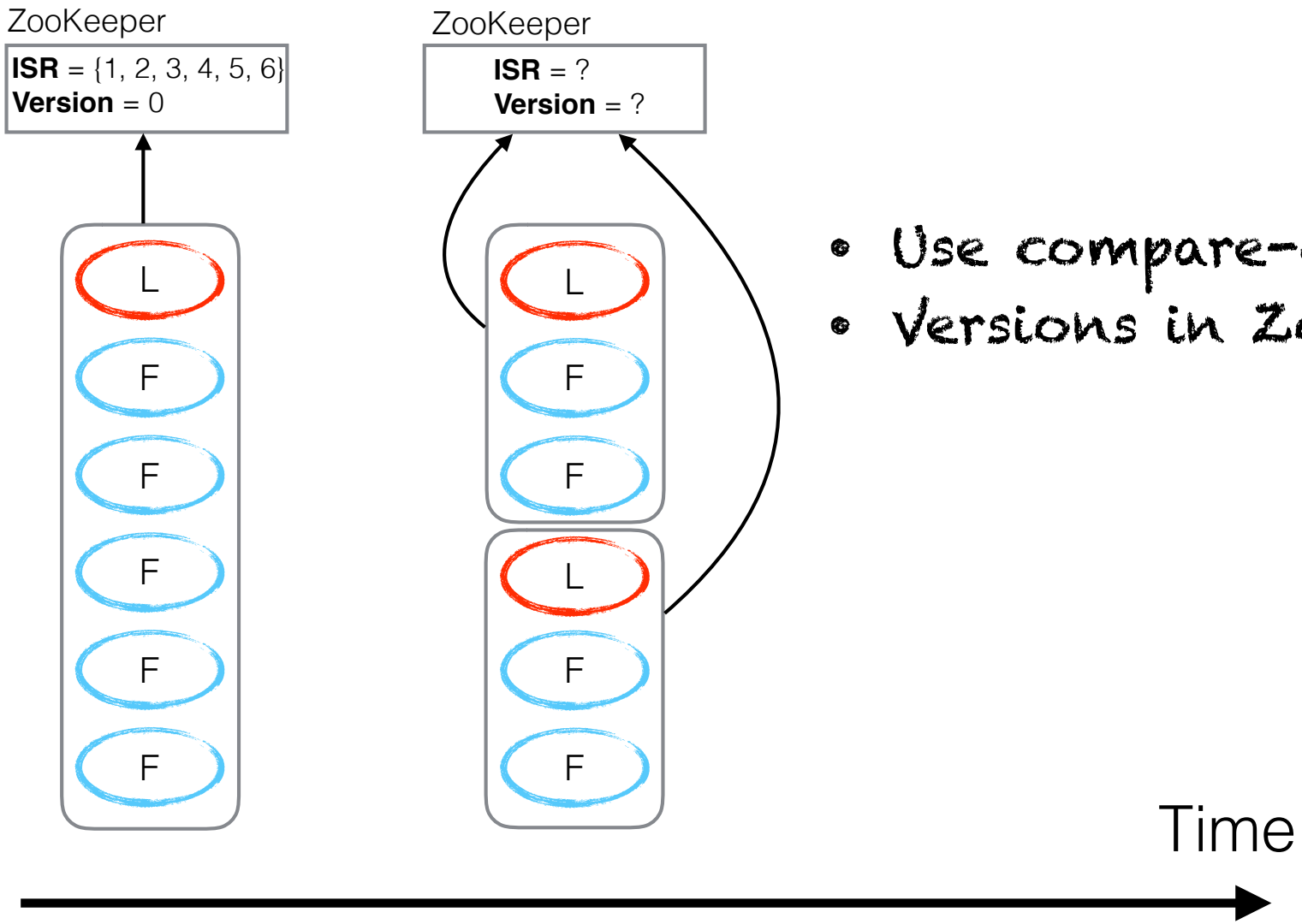
Partition replication



- How do we avoid this split brain?
- Both subsets were part of the previous ISR

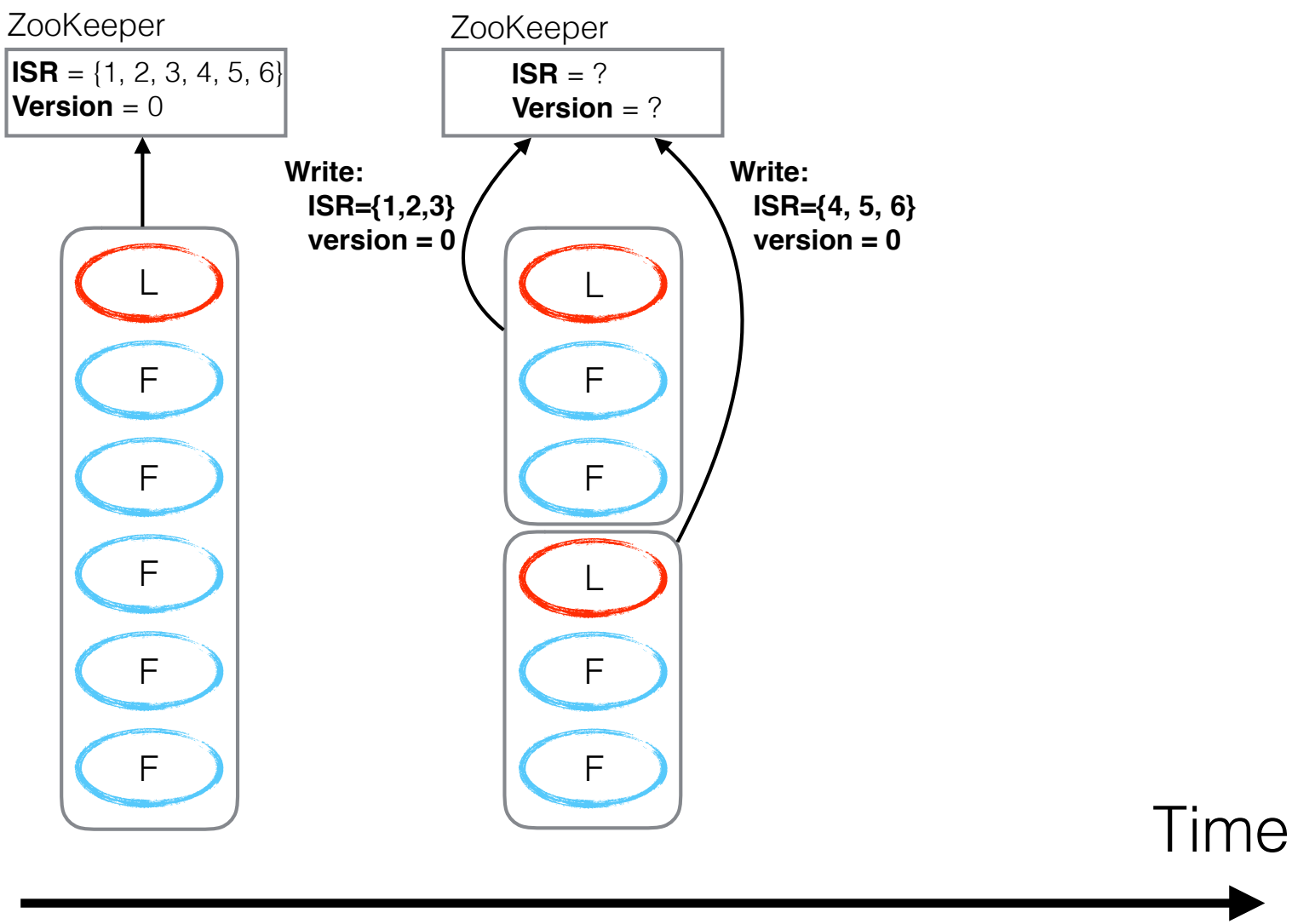
Time

Partition replication

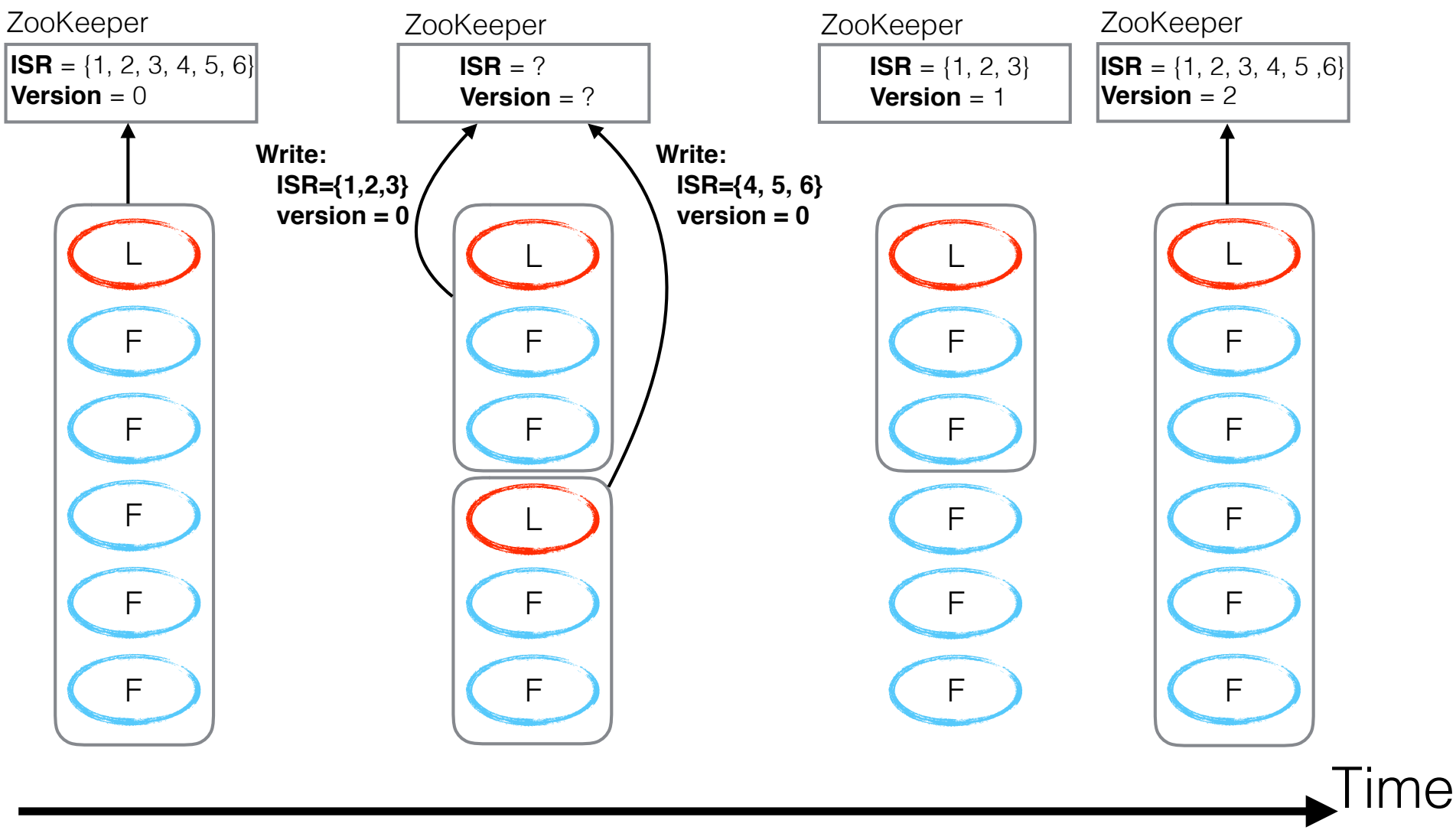


- Use compare-and-swap
- Versions in ZooKeeper

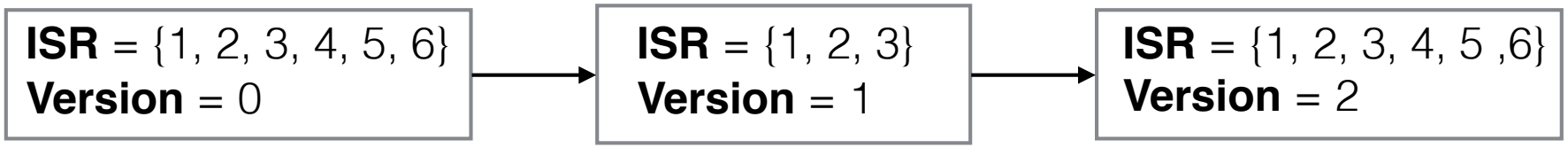
Partition replication



Partition replication



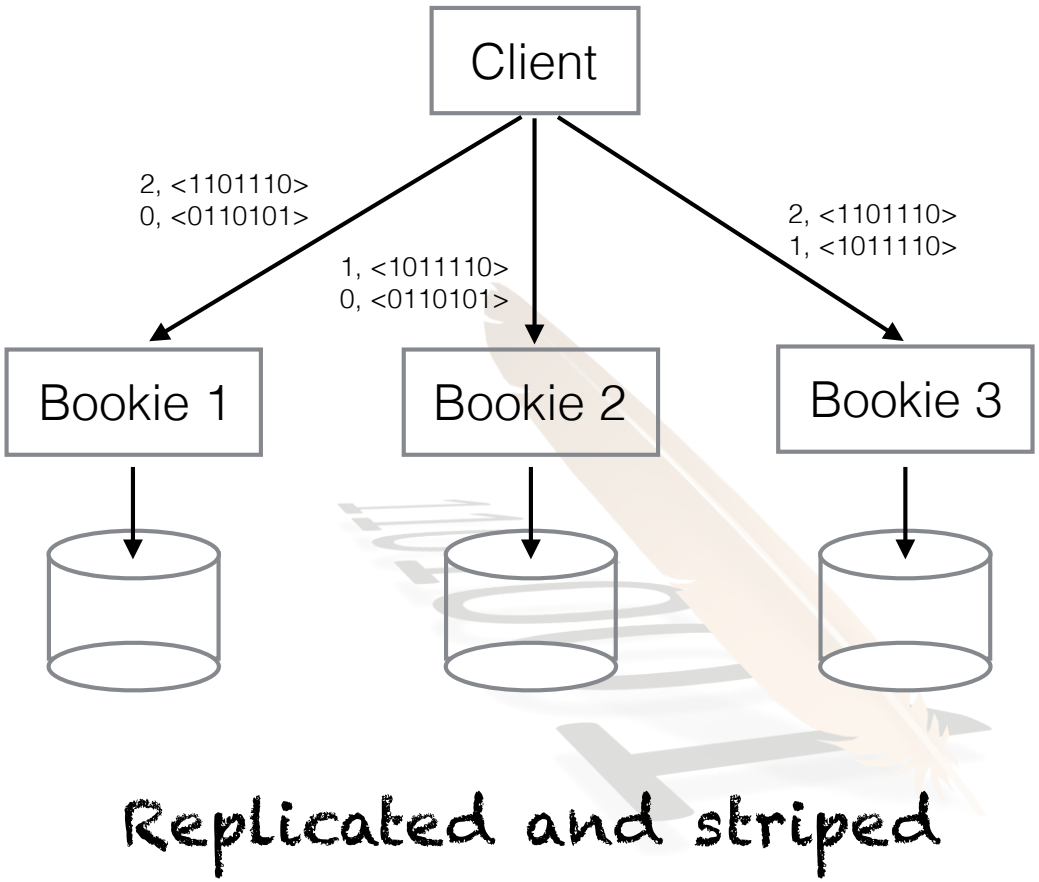
Total order of ISR changes



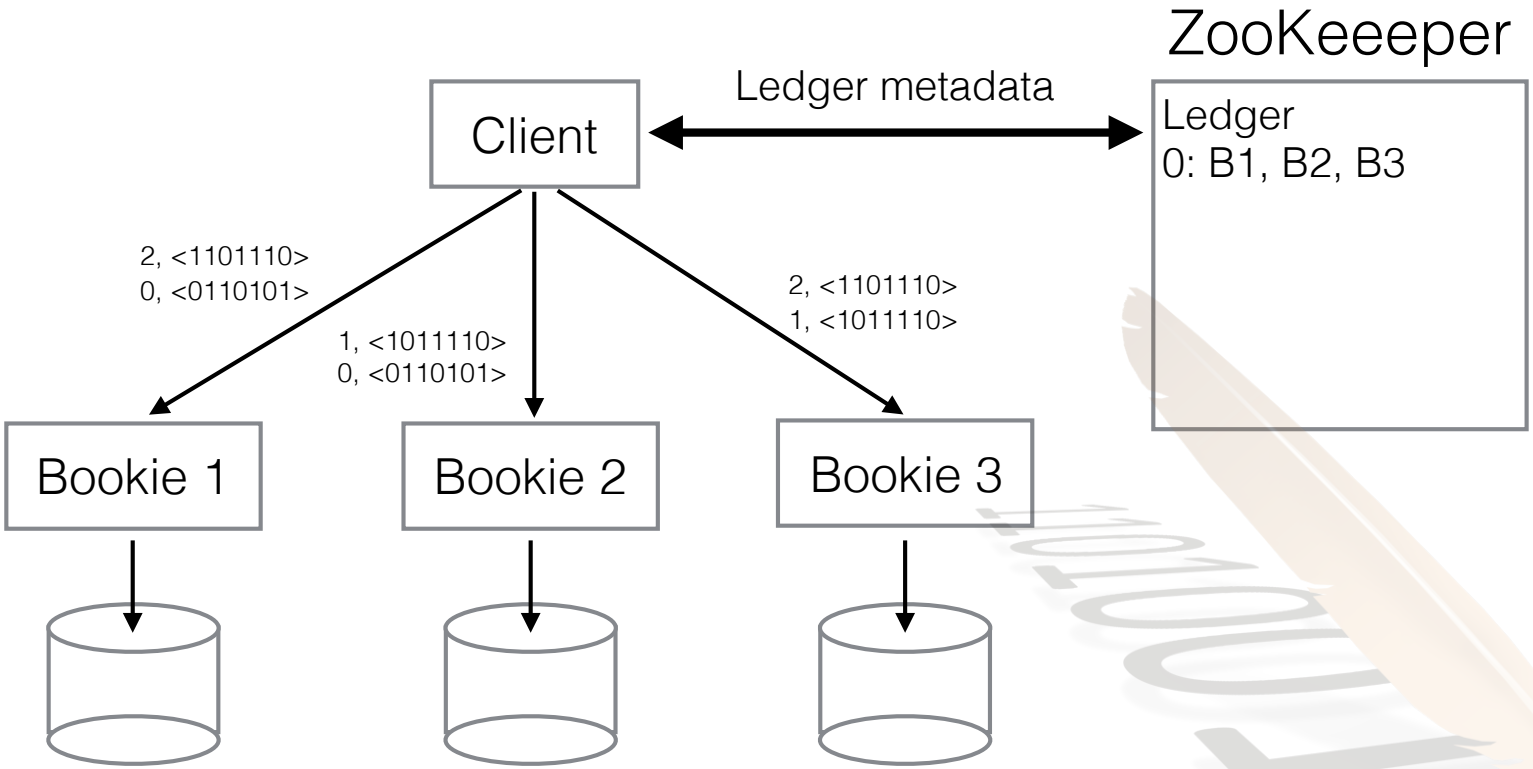
Need agreement on the order of ISR changes

Apache BookKeeper

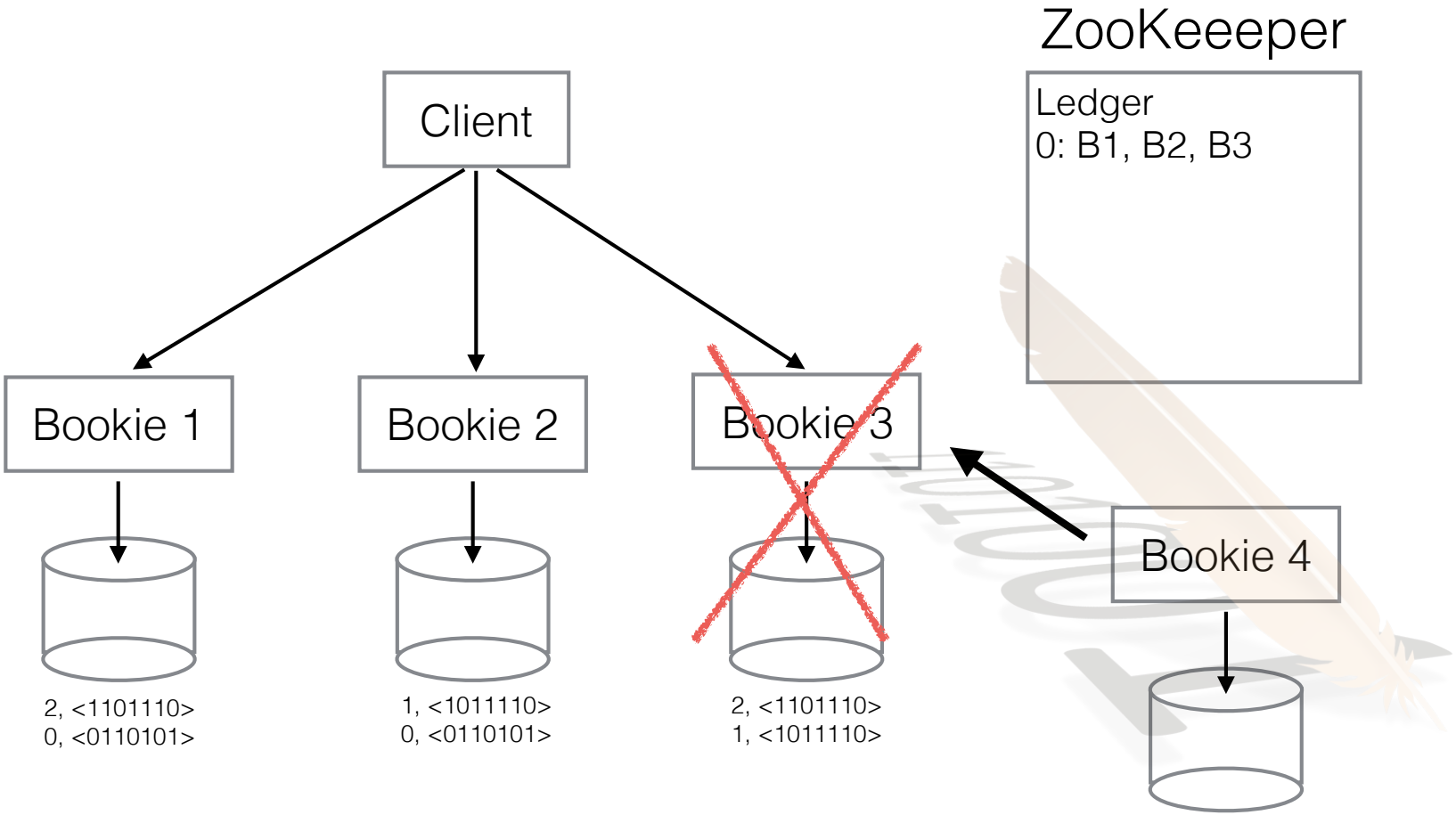
- Ledgers
  - ✓ Like a log segment
- Ensemble
- Single-writer
- Only writer changes the ensemble composition



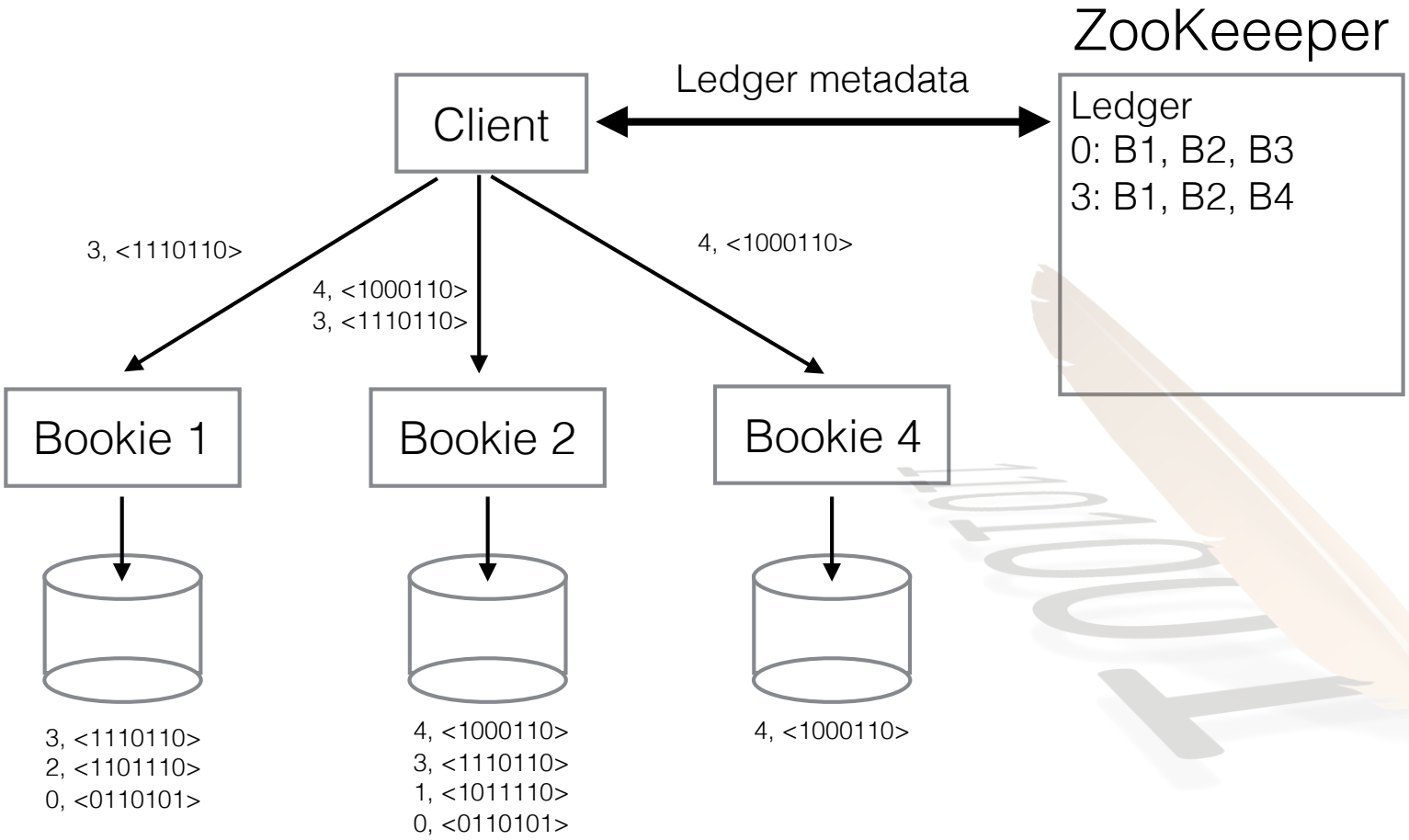
Apache BookKeeper



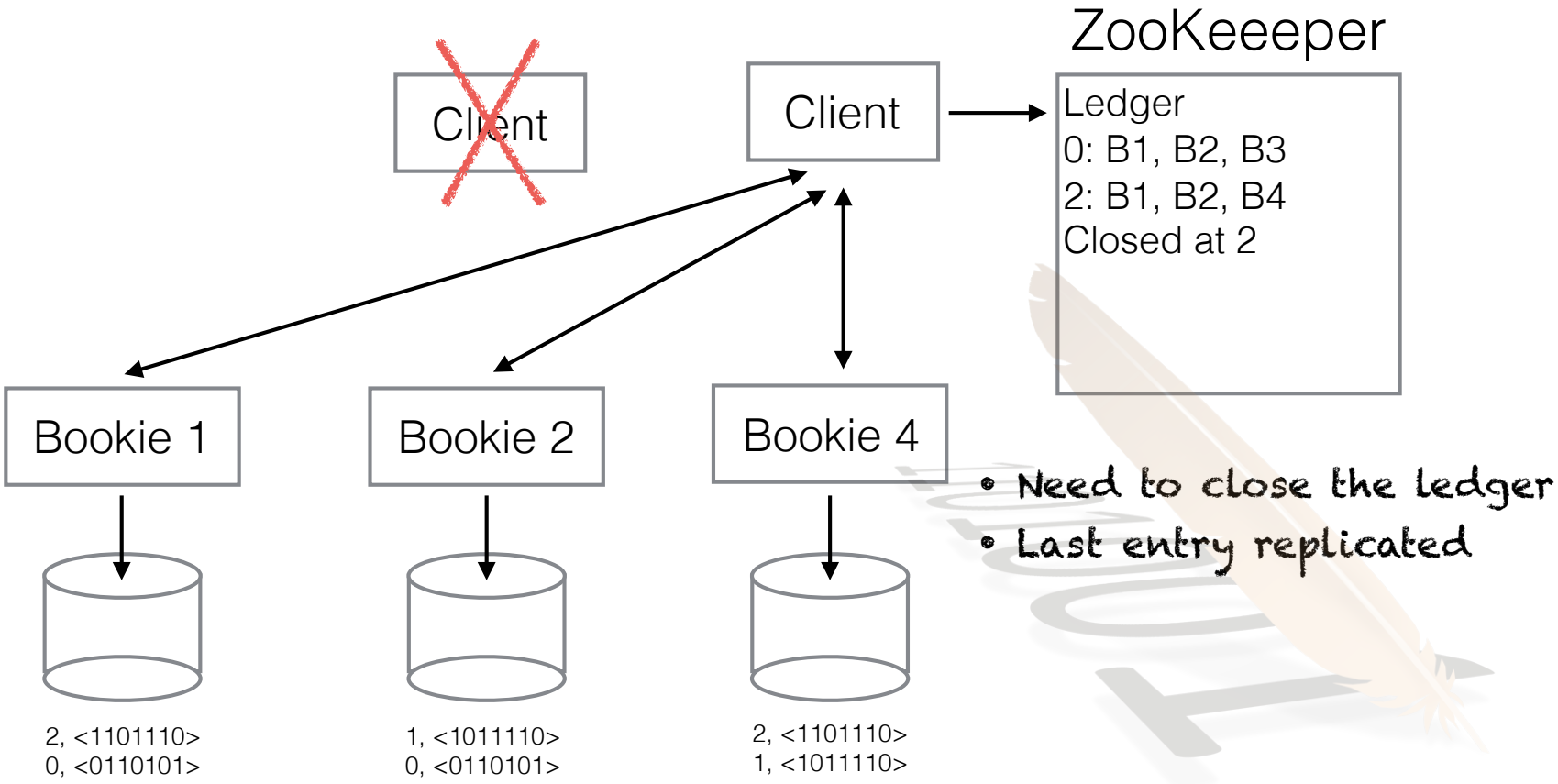
Apache BookKeeper



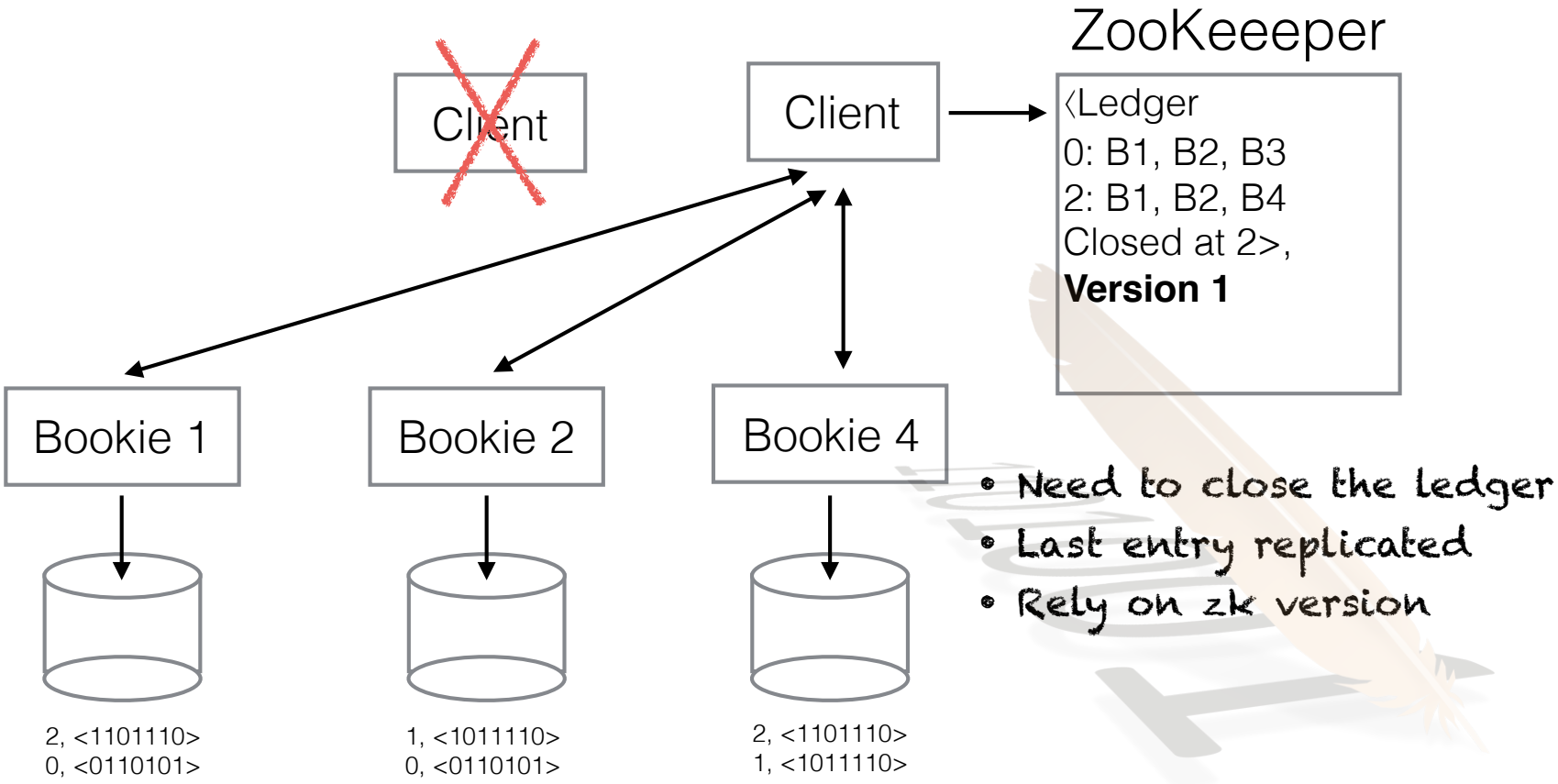
Apache BookKeeper



Apache BookKeeper



Apache BookKeeper



Use Compare-and-Swap

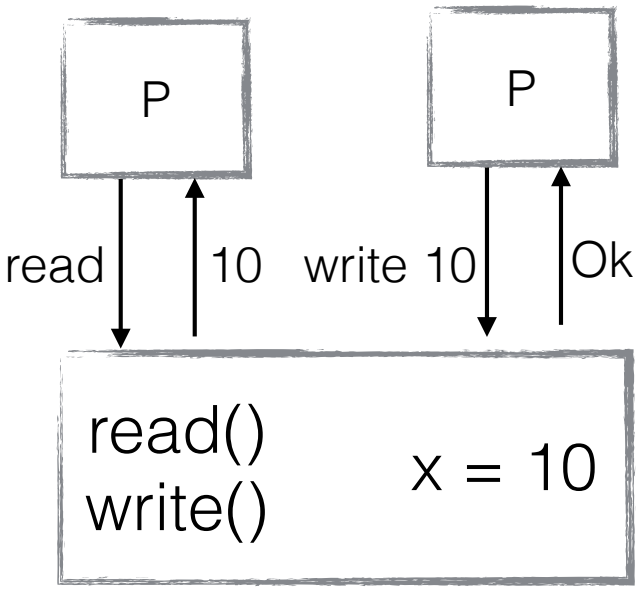
- CAS is consensus number *n*
- *… which means that the CAS object must implement consensus for n processes*

[Herlihy, *Wait-free Synchronization*]

Where is consensus not needed?

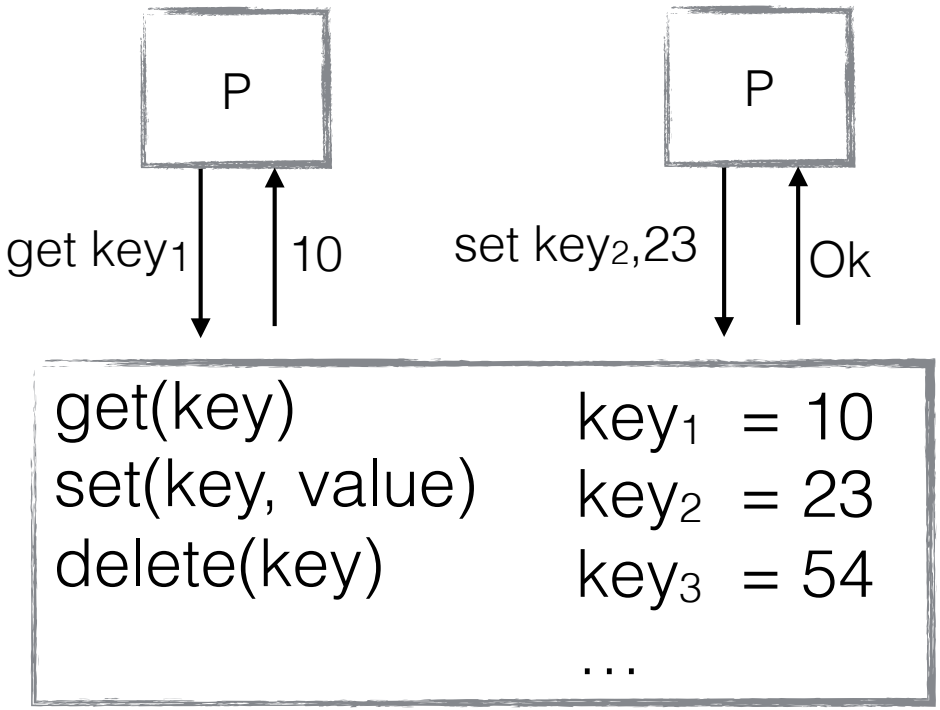
(*also from Herlihy's Hierarchy*)

Read/Write Registers

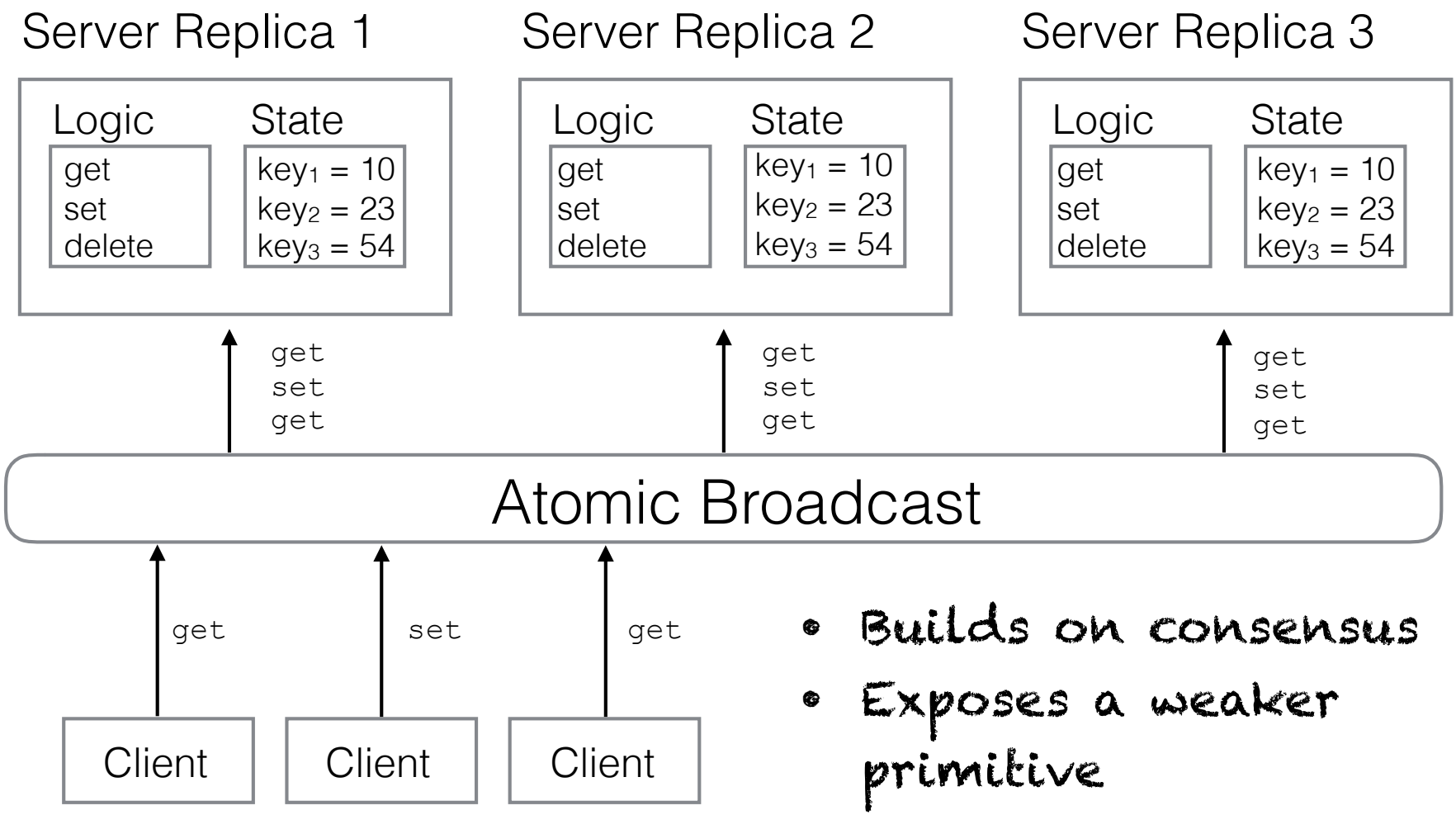


Consensus number 1
(can't have consensus even
for 2 processes only)

Key-Value Store



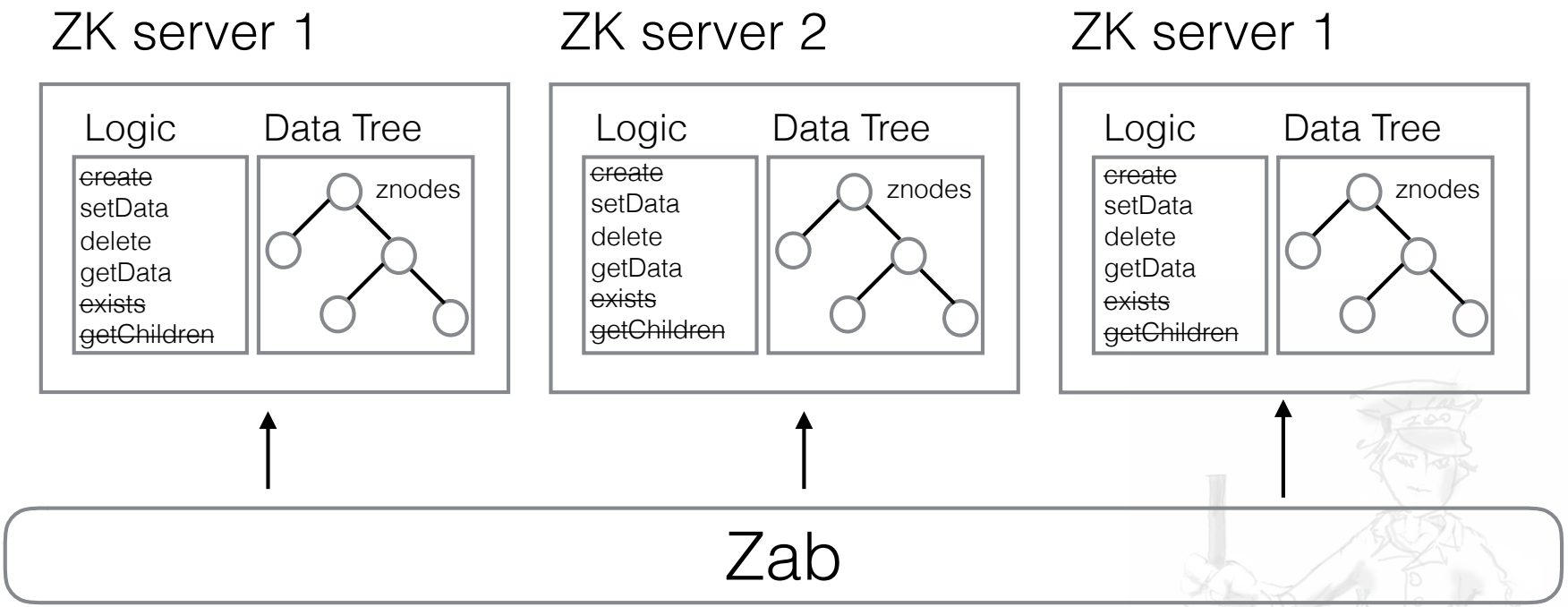
Implementation



- Builds on consensus
- Exposes a weaker primitive

Stripped-down ZooKeeper

Becomes a key-value store

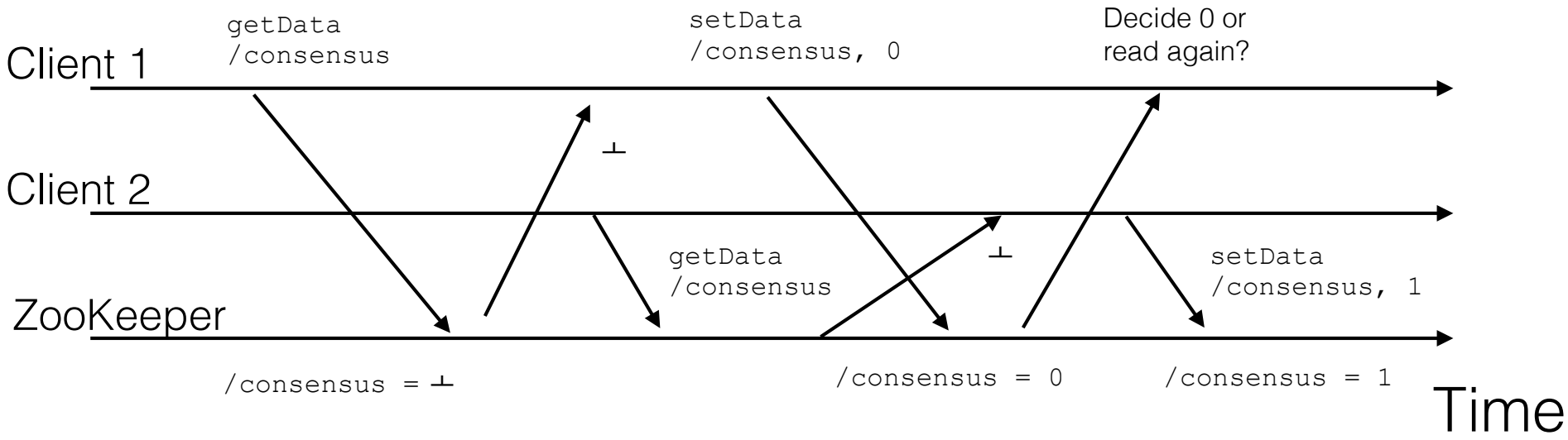


Additionally: setData and delete are unconditional

Why doesn’t ZooKeeper KV-store solve consensus?

No-consensus argument

- Clients 1 and 2 are trying to get consensus
- Client 1 initial value is 0
- Client 2 initial value is 1



Wrap up

State-machine replication

Atomic broadcast

Distributed Consensus

consensus number 1

r/w registers
– e.g., key-value stores

consensus number *n*

Compare-and-swap
– used to coordinate replica sets

confluent

Real-time data streaming platform based on Kafka.

http://confluent.io

Questions?

e-mail: fpj@apache.org
twitter: @fpjunqueira
web site: http://fpj.me