RAFT

Understandable Distributed Consensus

by: Simón Escobar B

Elizabeth & Clarke

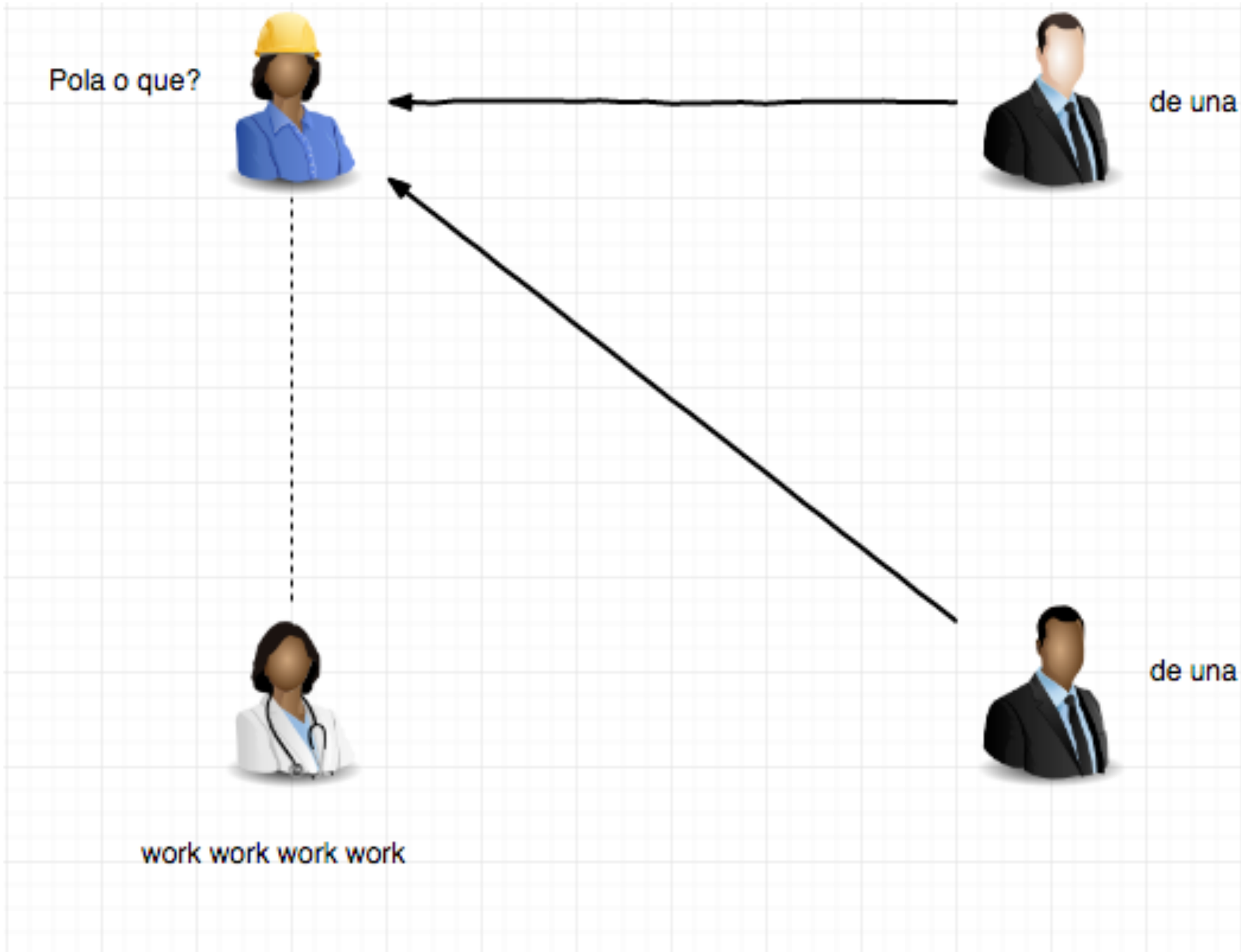
twitter: @sescob27

github: @sescobb27

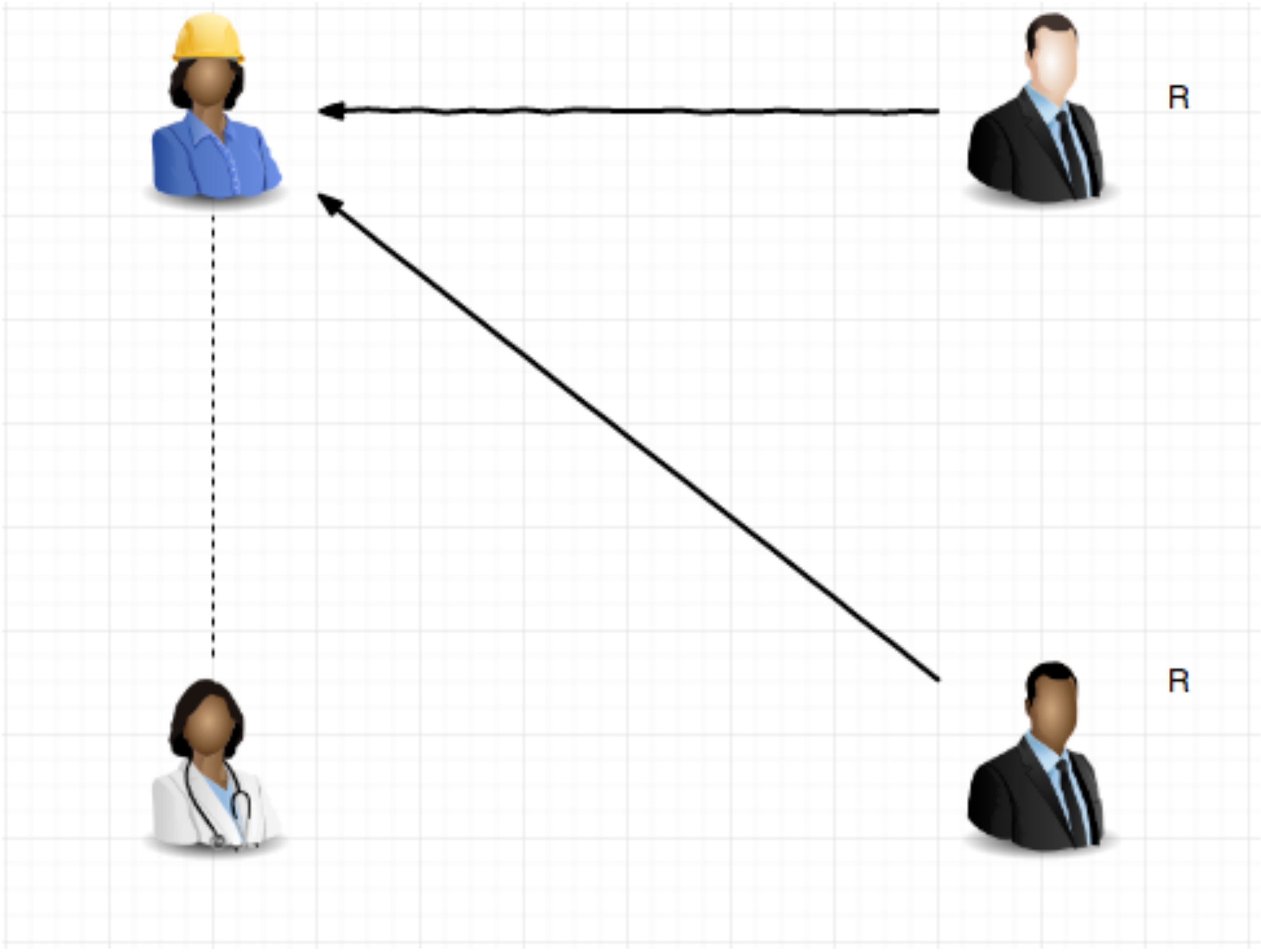
RAFT

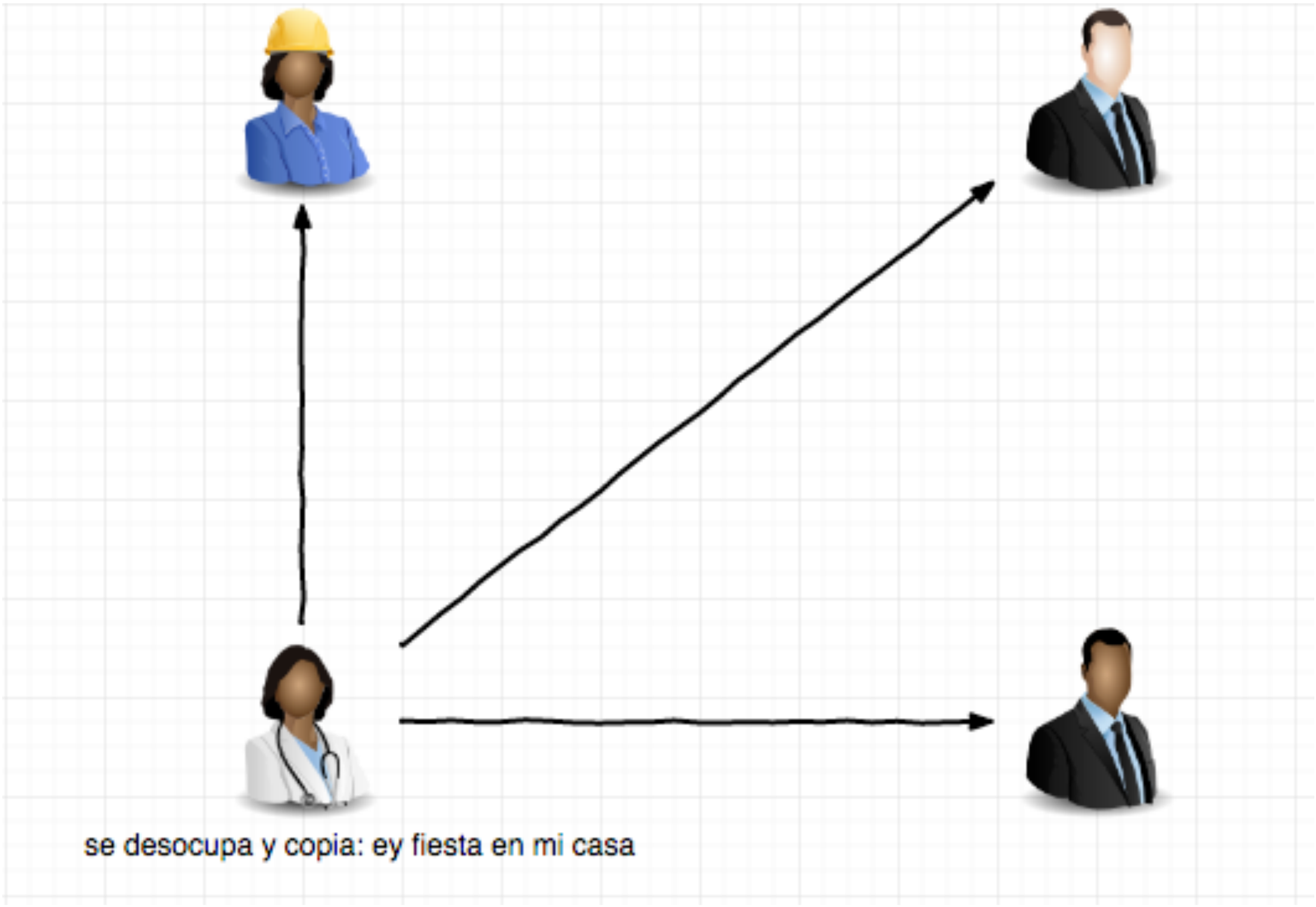
- By:
  - Diego Ongaro
  - John Ousterhout

Pola o que?


Pola o que?
de una
de una
work work work work

viernes pa bbc


R
R


se desocupa y copia: ey fiesta en mi casa

vamos a ir a tomar pola
vamos al lleras

R, de una

RAFT - Consensus

Leader Election

State Replication

Partition Tolerance

Safety

Membership Changes

Why RAFT?

Multiple DataBases?

Multiple APP Servers?

Do you think micro-services are the solution for everything?

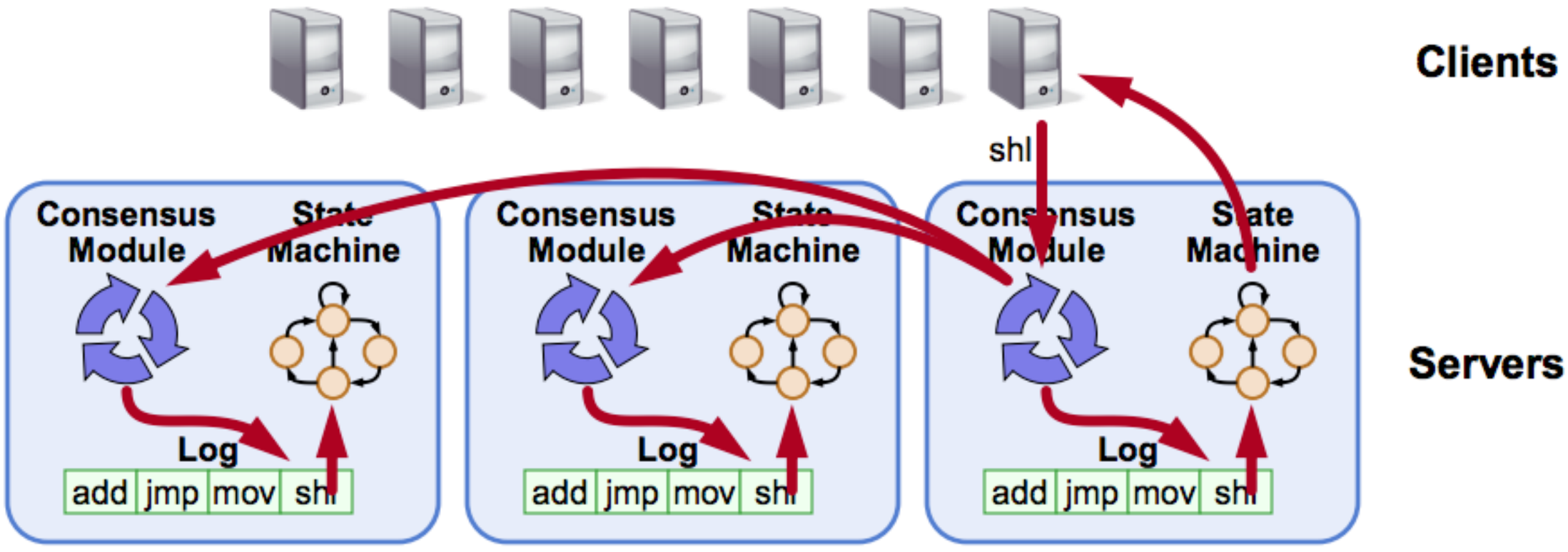
CAP Theorem

Use Cases

- Distributed logs
- Distributed File Systems
- DataBase Transactions
- Distributed Configuration
- etc

Who uses RAFT?

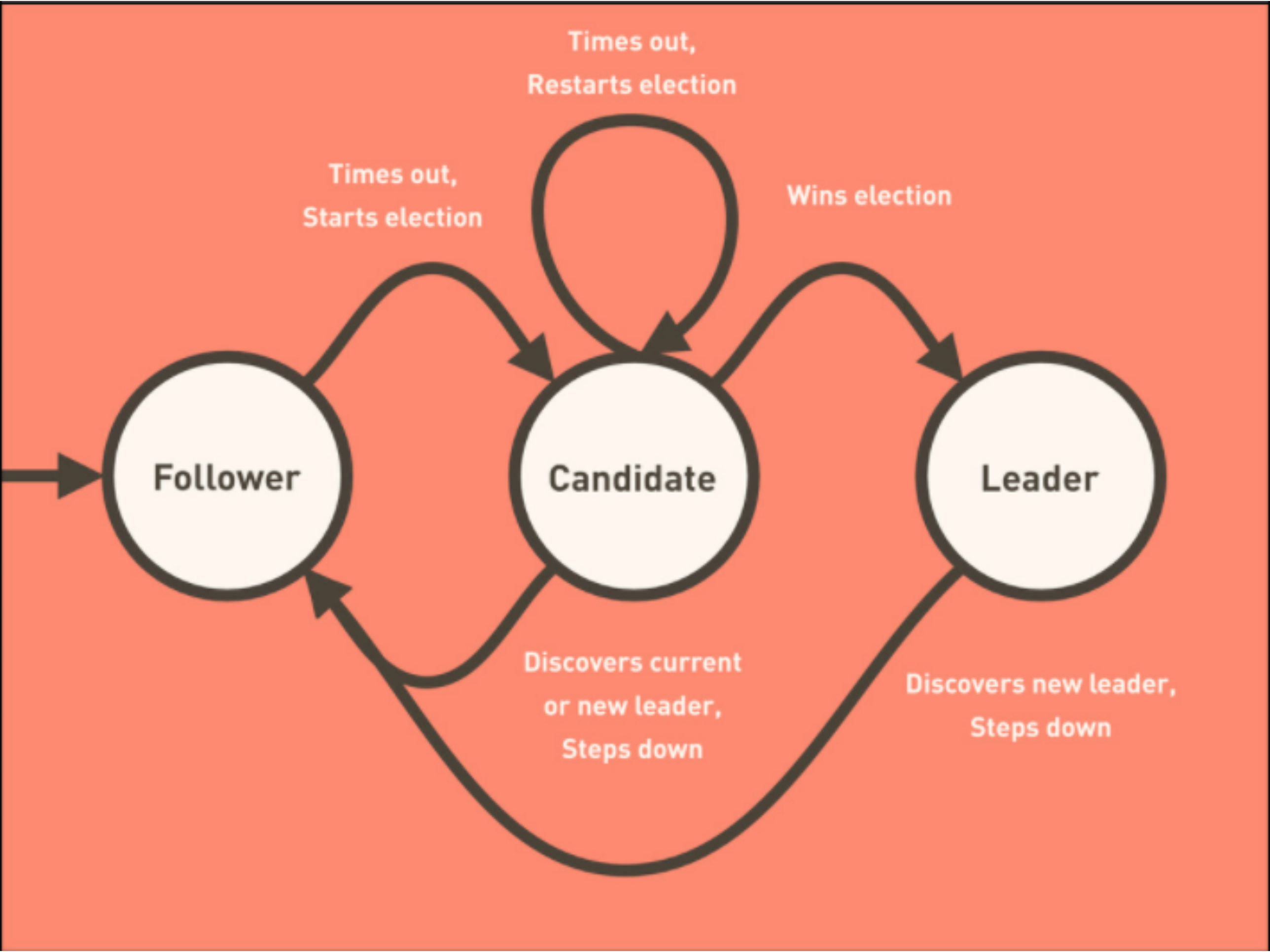
- etcd
- Consul
- Hashicorp
- Kubernetes uses raft in production via etcd
- CockroachDB
- Docker Swarm

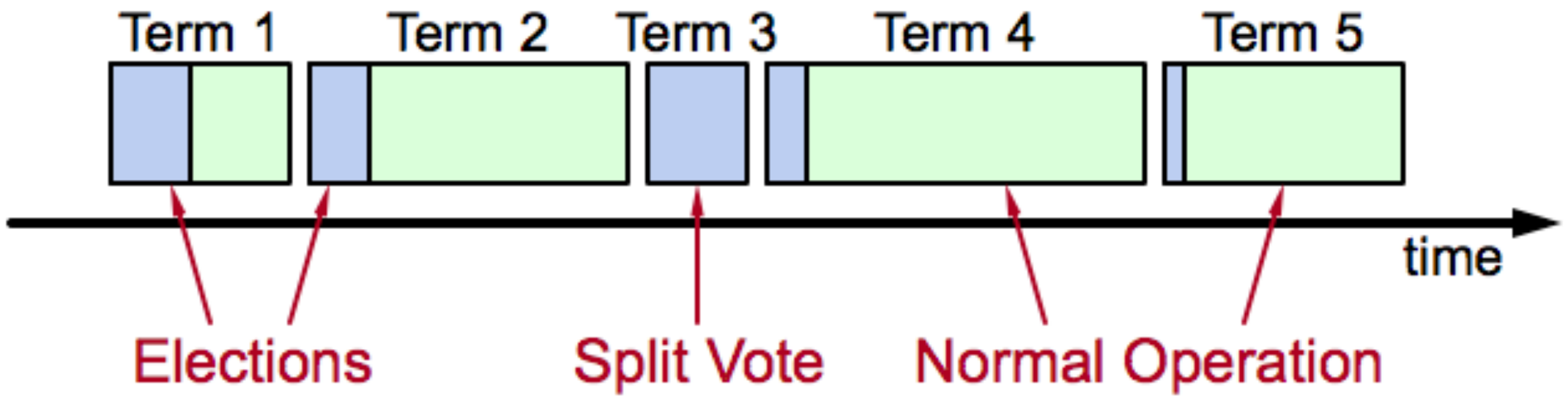

Clients
shl
Consensus Module
State Machine
Log
add
jmp
mov
Servers

RAFT Basics

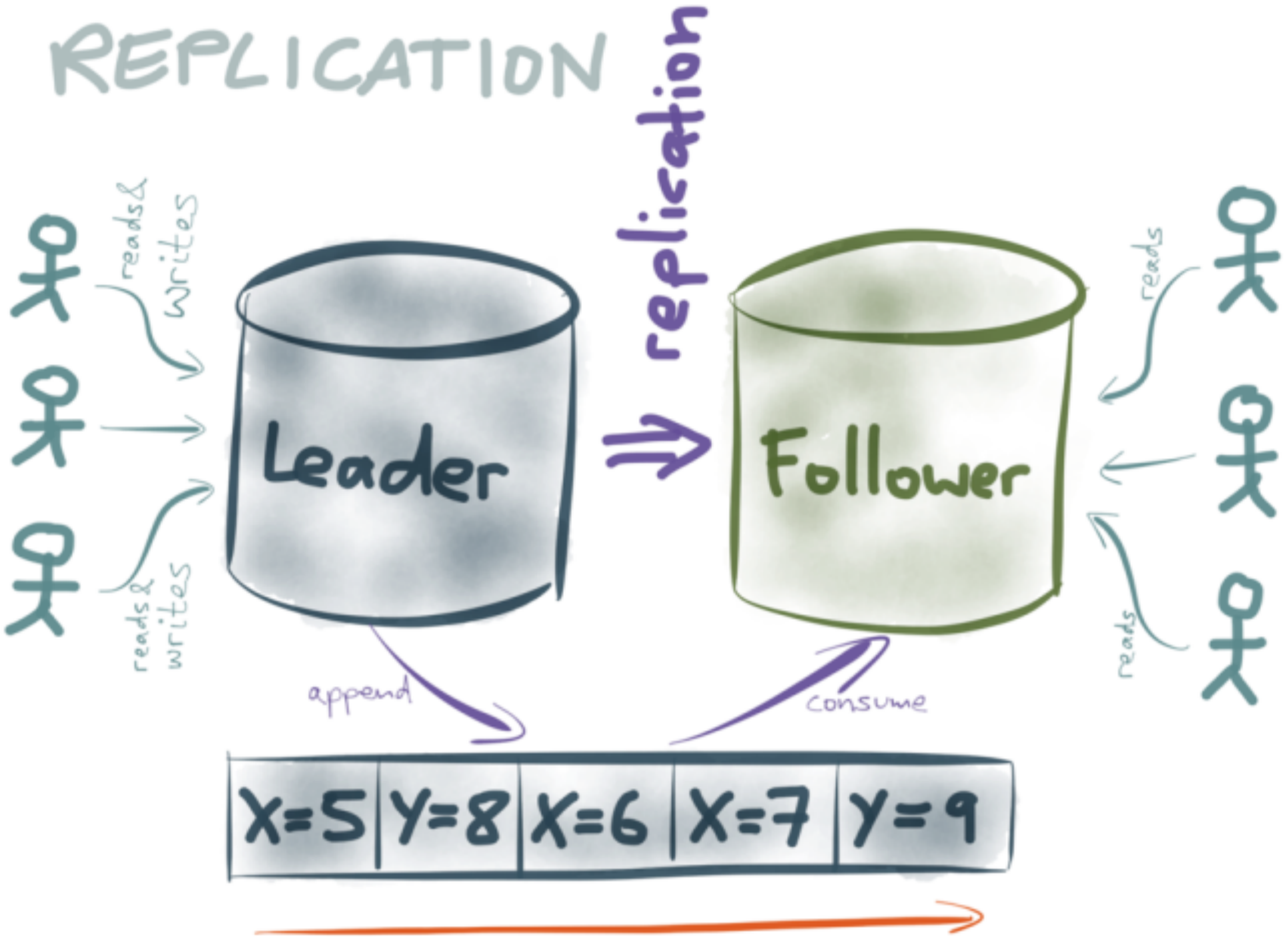
NOTE:

if message exchanges take longer than the typical time between server crashes, candidates will not stay up long enough to win an election; without a steady leader, Raft cannot make progress.

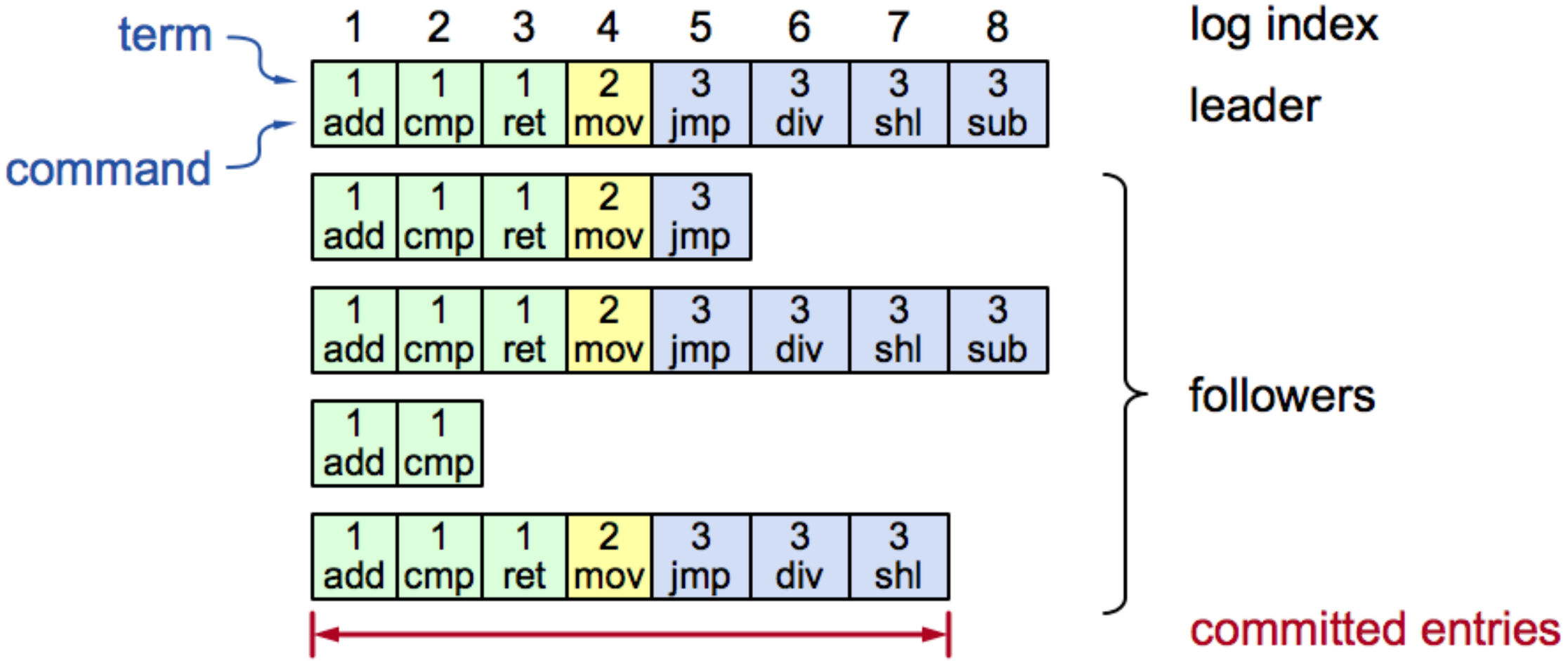

Times out,
Restarts election
Times out,
Starts election
Wins election
Follower
Candidate
Leader
Discovers current
or new leader,
Steps down
Discovers new leader,
Steps down


Term 1
Term 2
Term 3
Term 4
Term 5
time
Elections
Split Vote
Normal Operation

REPLICATION



Log Structure



Append Entries

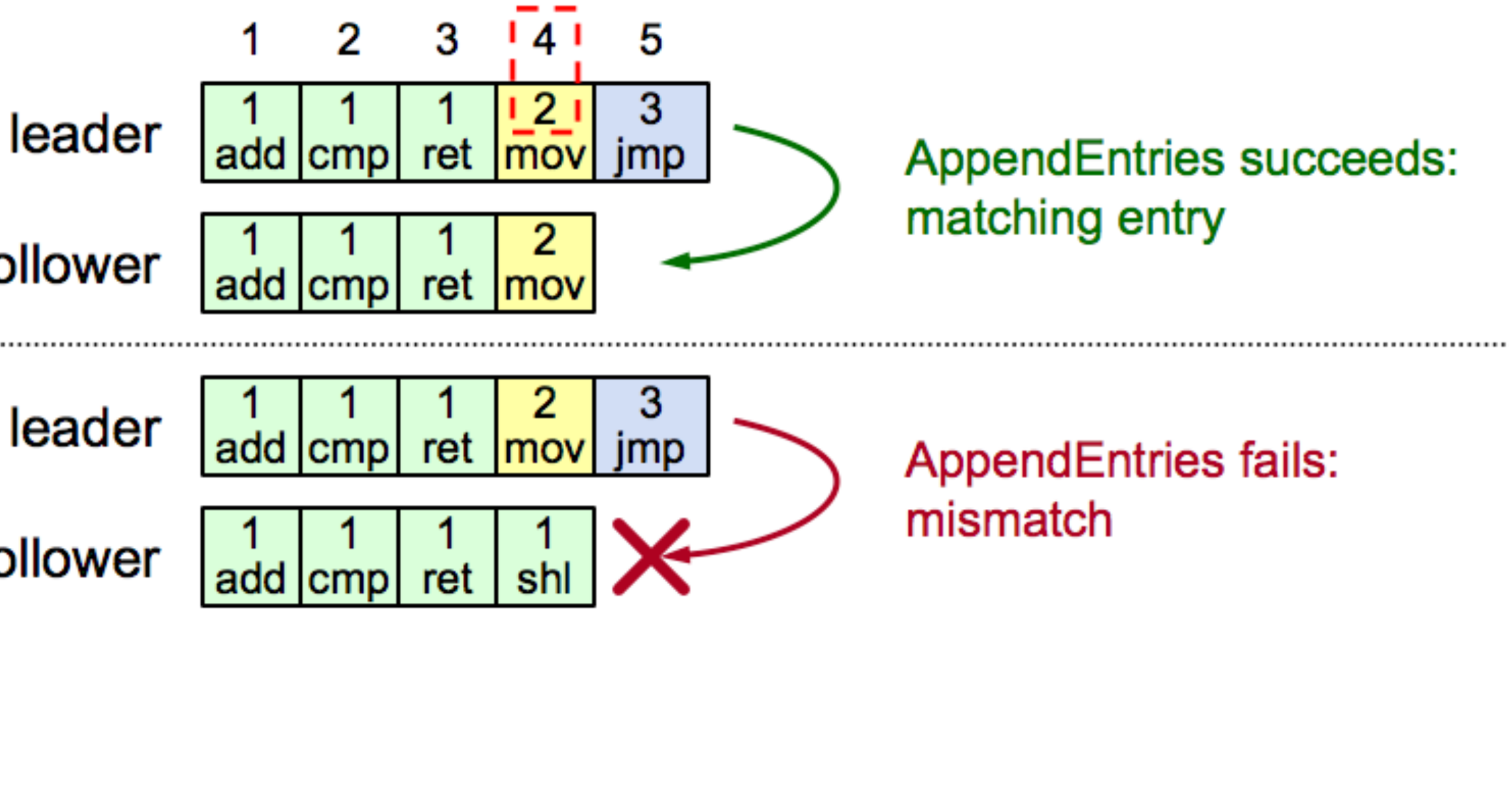
```
{
  term: 1,          // leader's term
  prevLogIndex: 5, // index of log entry immediately preceding new ones
  prevLogTerm: 1,  // term of prevLogIndex entry
  leaderCommit: 5  // leader's commitIndex,
  entries: [{ action: 'add', item: 'pilsen', order_id: '1' }] // log entries to store
}
```

Log Consistency

1	2	3	4	5	6
1 add	1 cmp	1 ret	2 mov	3 jmp	3 div

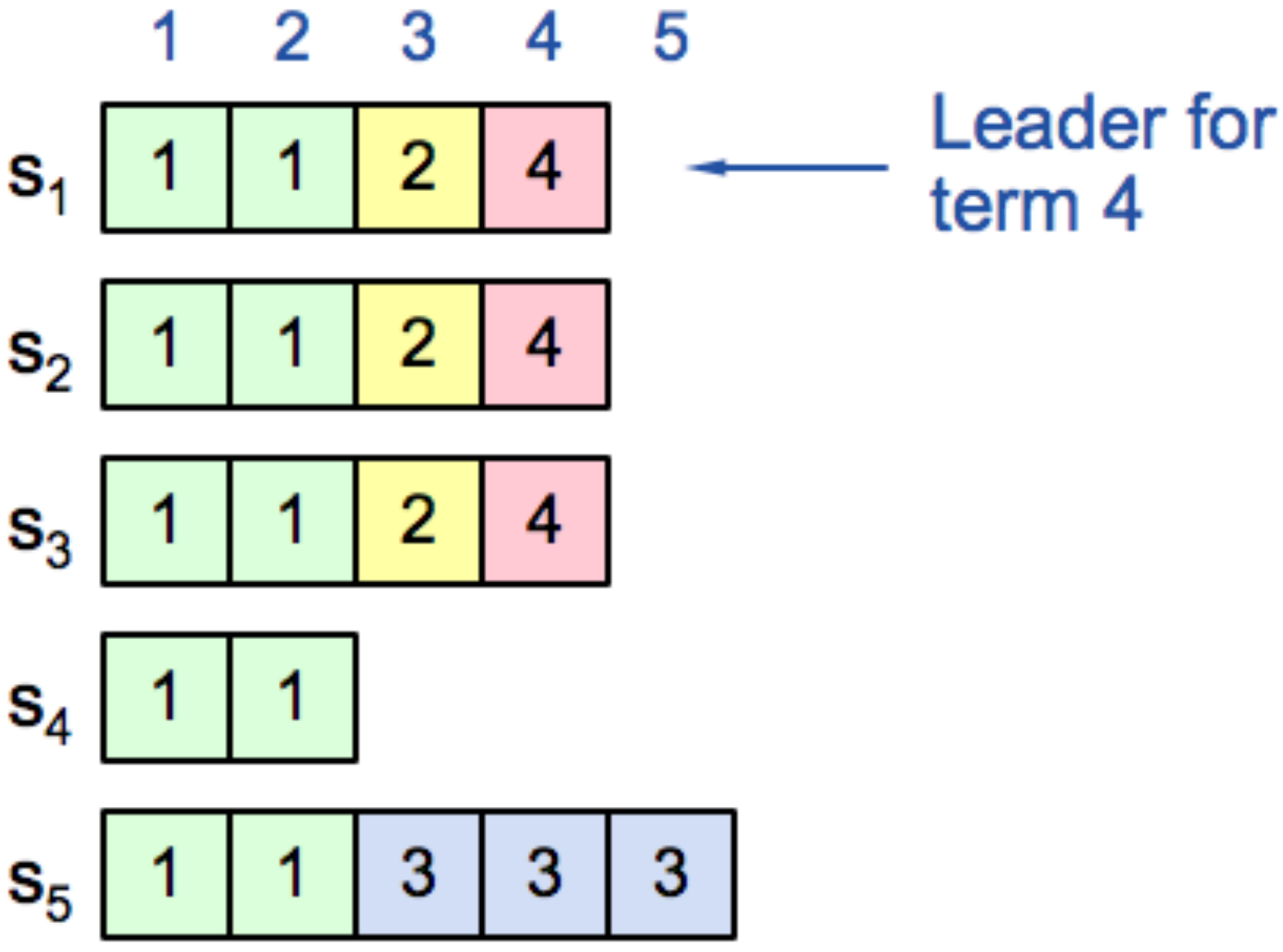
1 add	1 cmp	1 ret	2 mov	3 jmp	4 sub

Consistency Check



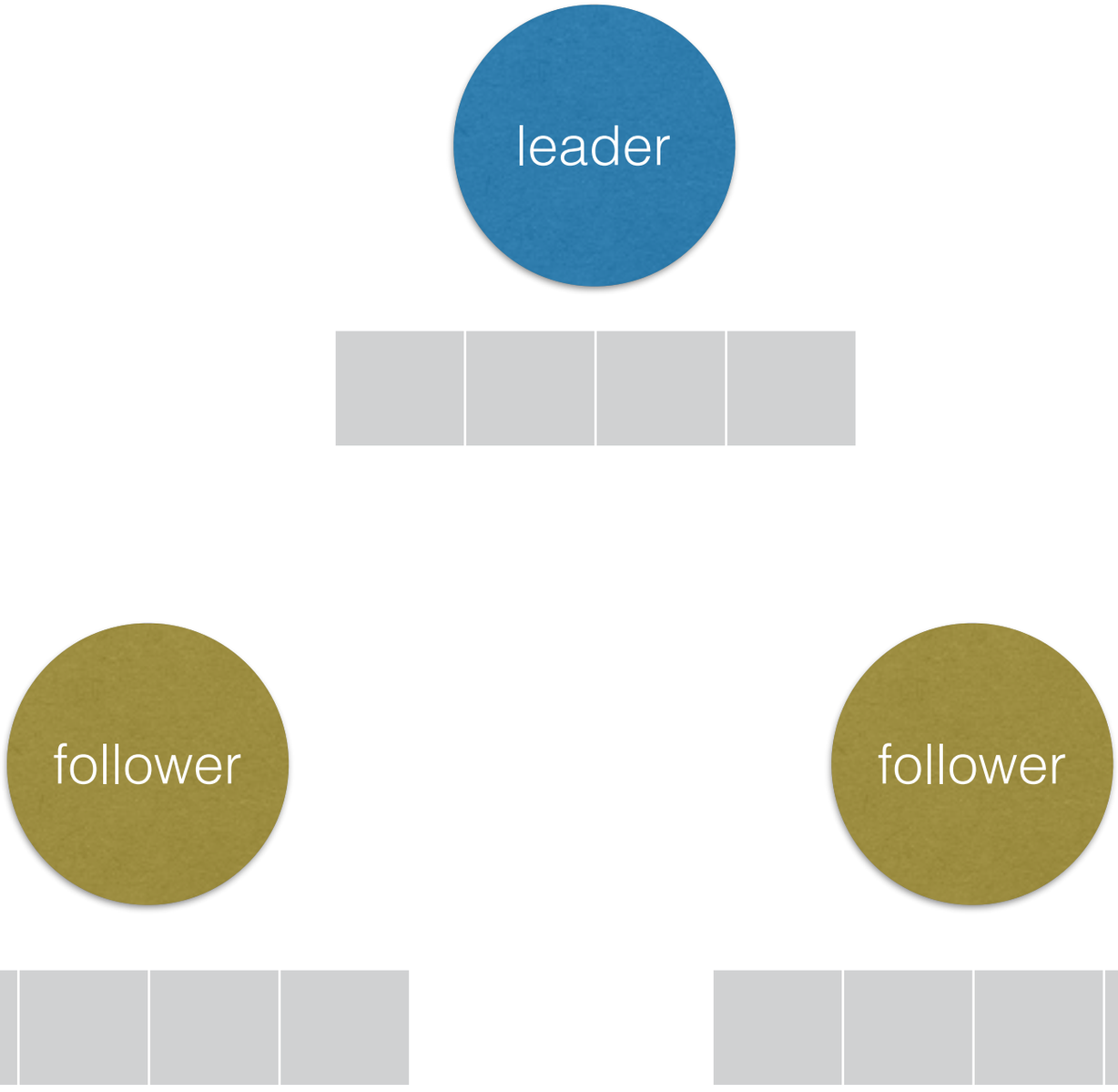
Commitment Rules

- **For a leader to decide an entry is committed**
  - Must be stored on a majority of servers
  - At least one new entry from leader's term must also be stored on majority of servers

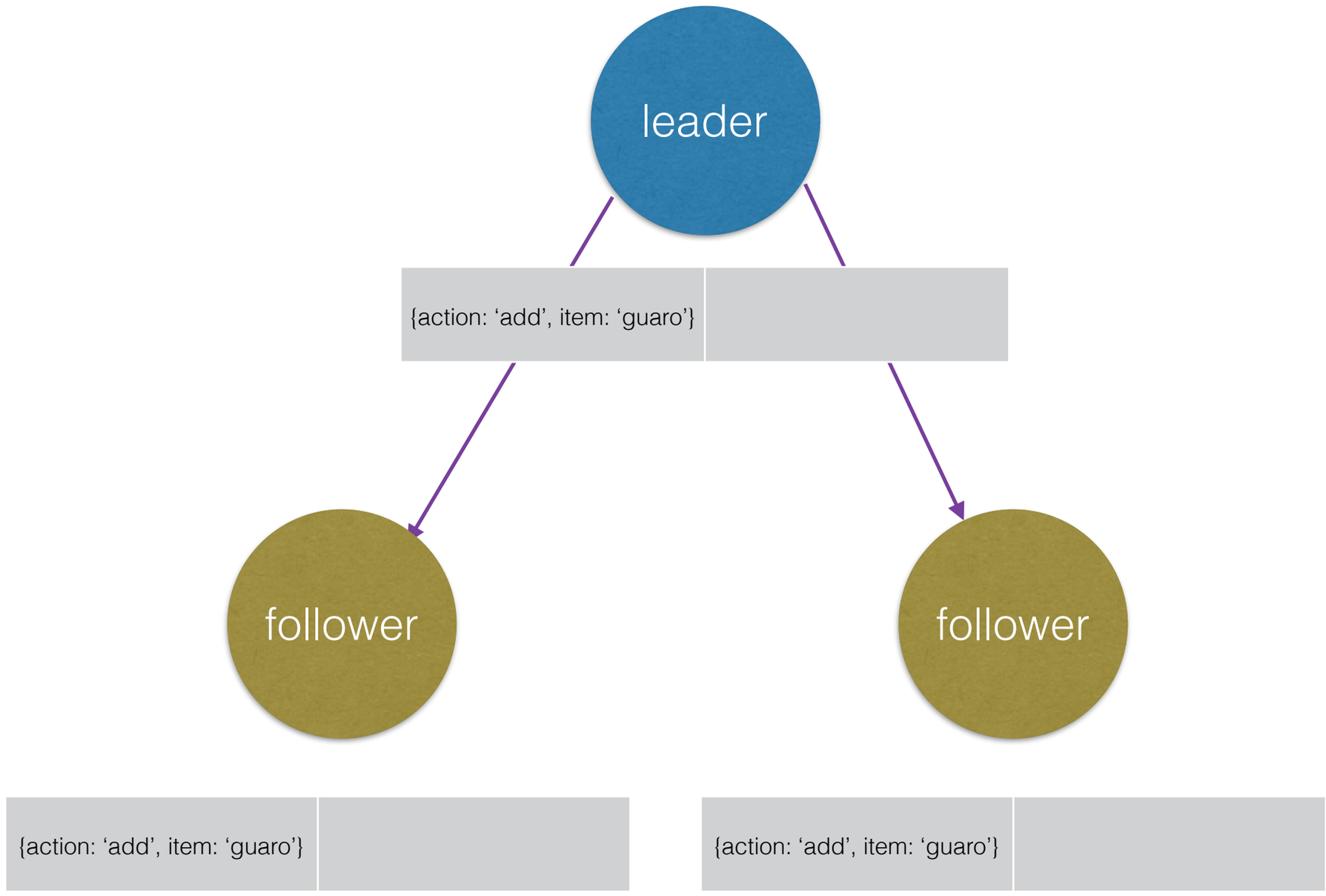

1 2 3 4 5
s_1 1 1 2 4
Leader for term 4
s_2 1 1 2 4
s_3 1 1 2 4
s_4 1 1
s_5 1 1 3 3 3

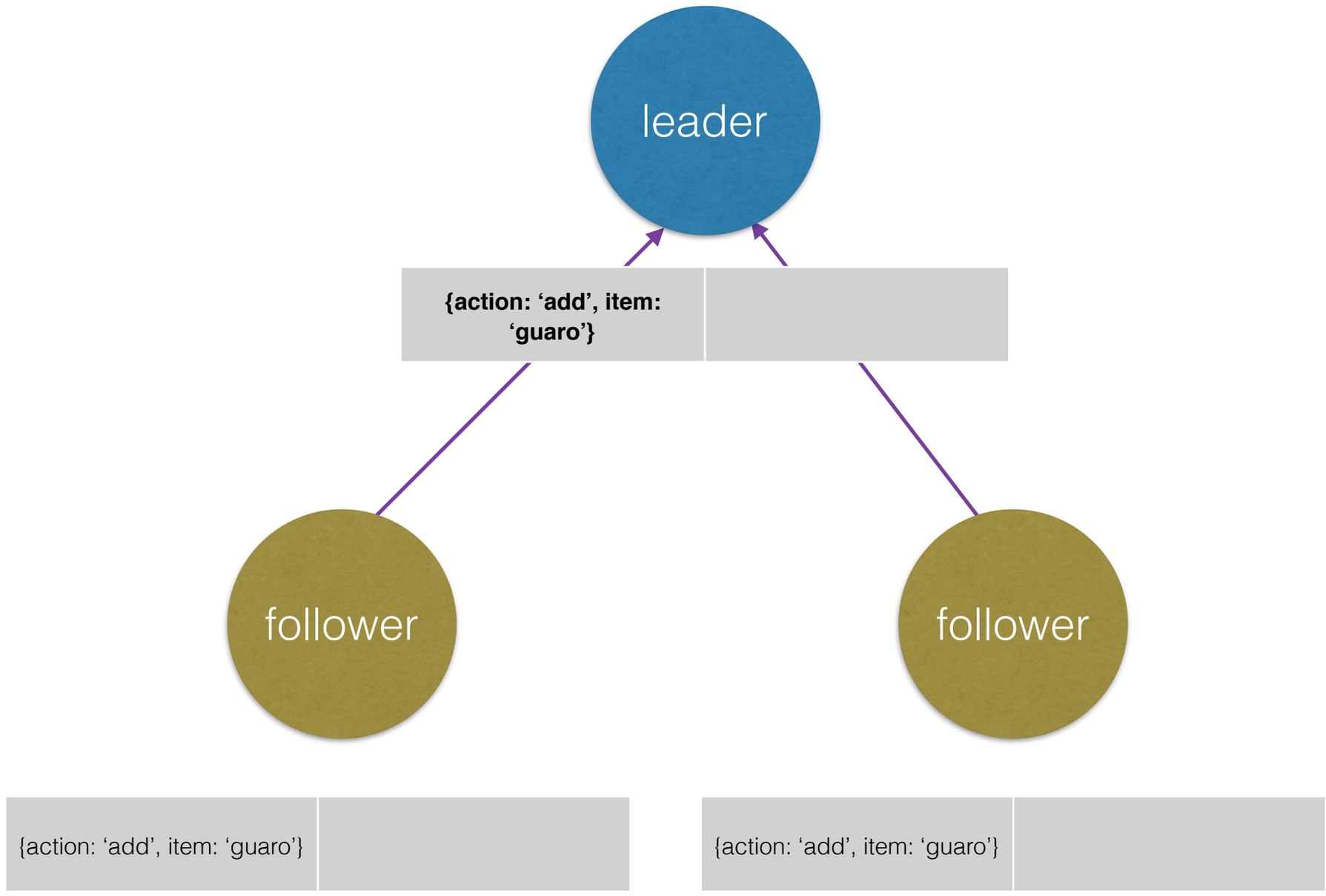
How it works?

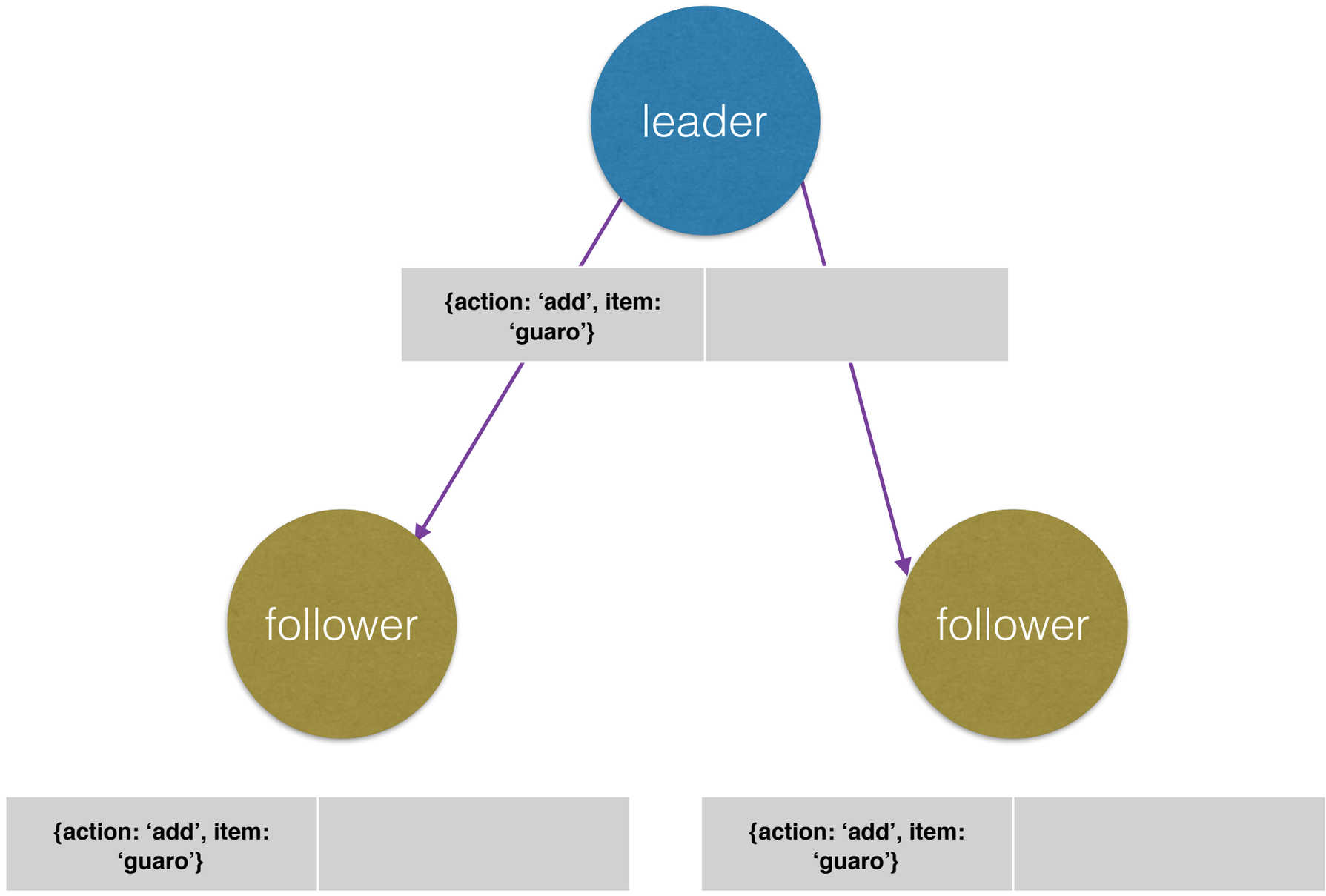
Happy Consensus


leader
follower
follower

request


leader
{action: ‘add’, item: ‘guaro’}
follower
follower
{action: ‘add’, item: ‘guaro’}
{action: ‘add’, item: ‘guaro’}


leader
{action: ‘add’, item: ‘guaro’}
follower
follower
{action: ‘add’, item: ‘guaro’}
{action: ‘add’, item: ‘guaro’}


leader
{action: ‘add’, item: ‘guaro’}
follower
follower
{action: ‘add’, item: ‘guaro’}
{action: ‘add’, item: ‘guaro’}

response

leader

{action: ‘add’, item: ‘guaro’}	

follower

{action: ‘add’, item: ‘guaro’}	

follower

{action: ‘add’, item: ‘guaro’}	

Tricky Consensus

request

leader

Network Partition

{action: ‘add’, item: ‘guaro’}

follower

follower

Network Partition

leader

{action: ‘add’, item: ‘guaro’}

follower

leader

request

{action: ‘add’, item: ‘ron’}

Network Partition

leader

{action: ‘add’, item: ‘guaro’}

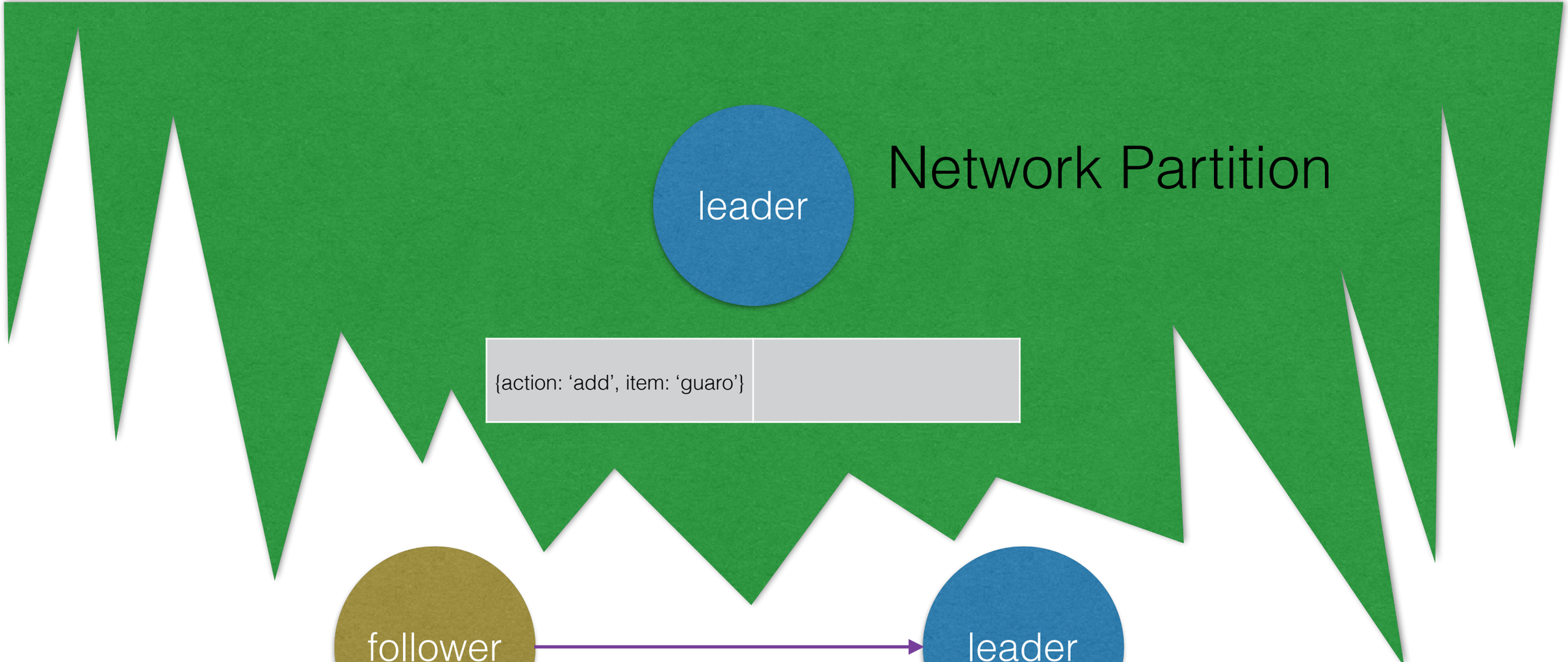
follower

leader

{action: ‘add’, item: ‘ron’}

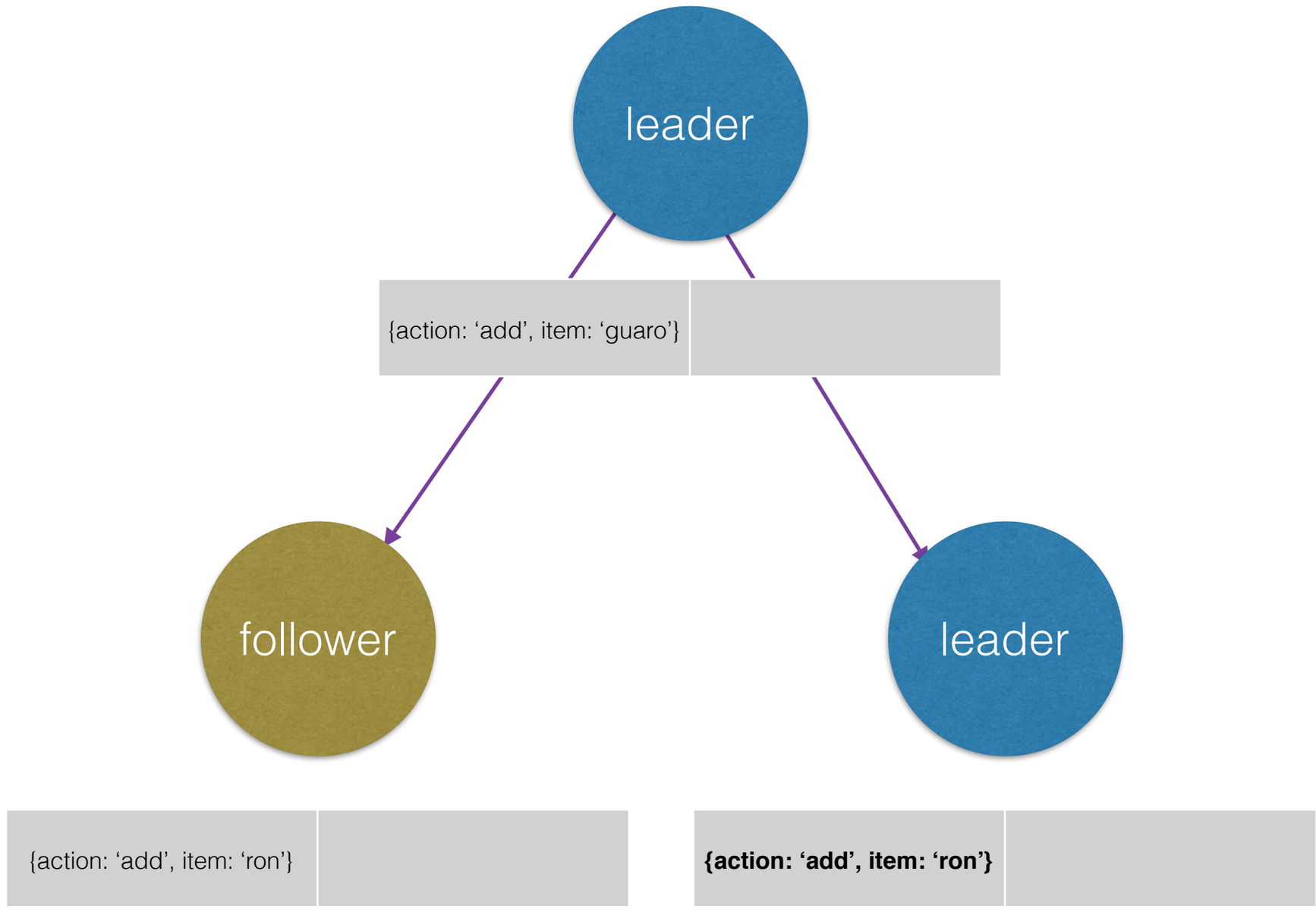
{action: ‘add’, item: ‘ron’}

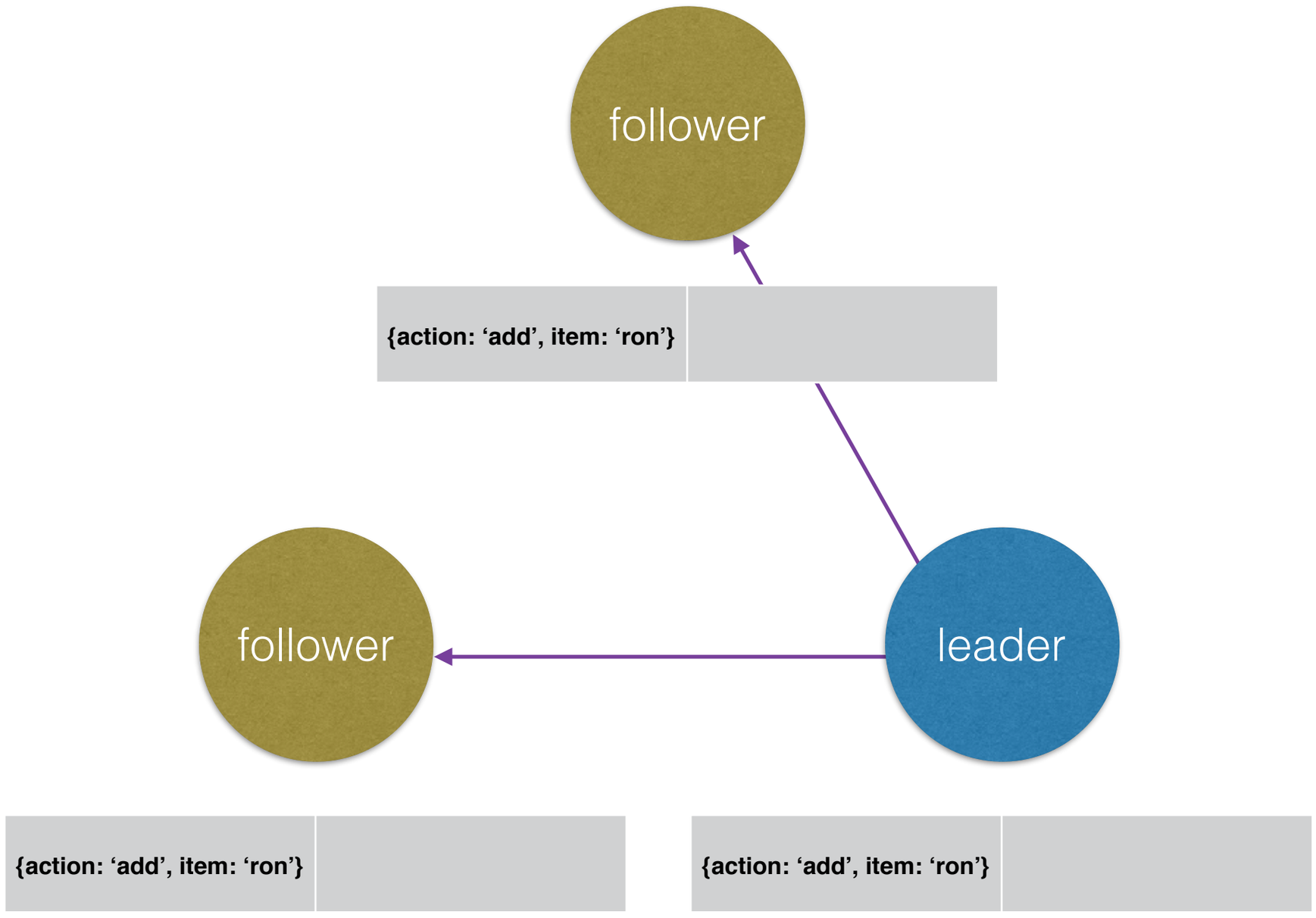
Network Partition



{action: ‘add’, item: ‘ron’}	

{action: ‘add’, item: ‘ron’}	


leader
{action: ‘add’, item: ‘guaro’}
follower
leader
{action: ‘add’, item: ‘ron’}
{action: ‘add’, item: ‘ron’}


follower
{action: ‘add’, item: ‘ron’}
follower
leader
{action: ‘add’, item: ‘ron’}
{action: ‘add’, item: ‘ron’}

WTF Consensus

request

leader

{action: ‘remove’, item: ‘tip’}

follower

follower

Network Partition

leader

{action: ‘remove’, item: ‘tip’}

follower

{action: ‘remove’, item: ‘tip’}

follower

Network Partition

leader

{action: ‘remove’, item: ‘tip’}

follower

{action: ‘remove’, item: ‘tip’}

follower

Network Partition

Network Partition

leader

{action: ‘remove’, item: ‘tip’}	

follower

{action: ‘remove’, item: ‘tip’}	

follower

Network Partition

leader

{action: ‘remove’, item: ‘tip’}	

follower

{action: ‘remove’, item: ‘tip’}	

candidate

Network Partition

leader

{action: ‘remove’, item: ‘tip’}	

leader

{action: ‘remove’, item: ‘tip’}	

follower

Network Partition

leader

{action: ‘remove’, item: ‘tip’}	

leader

{action: ‘remove’, item: ‘tip’}	

follower

leader

Network Partition

{action: ‘remove’, item: ‘tip’}	

leader

follower

{action: ‘remove’, item: ‘tip’}	

{action: ‘remove’, item: ‘tip’}	

{action: ‘remove’, item: ‘tip’}	

{action: ‘remove’, item: ‘tip’}	

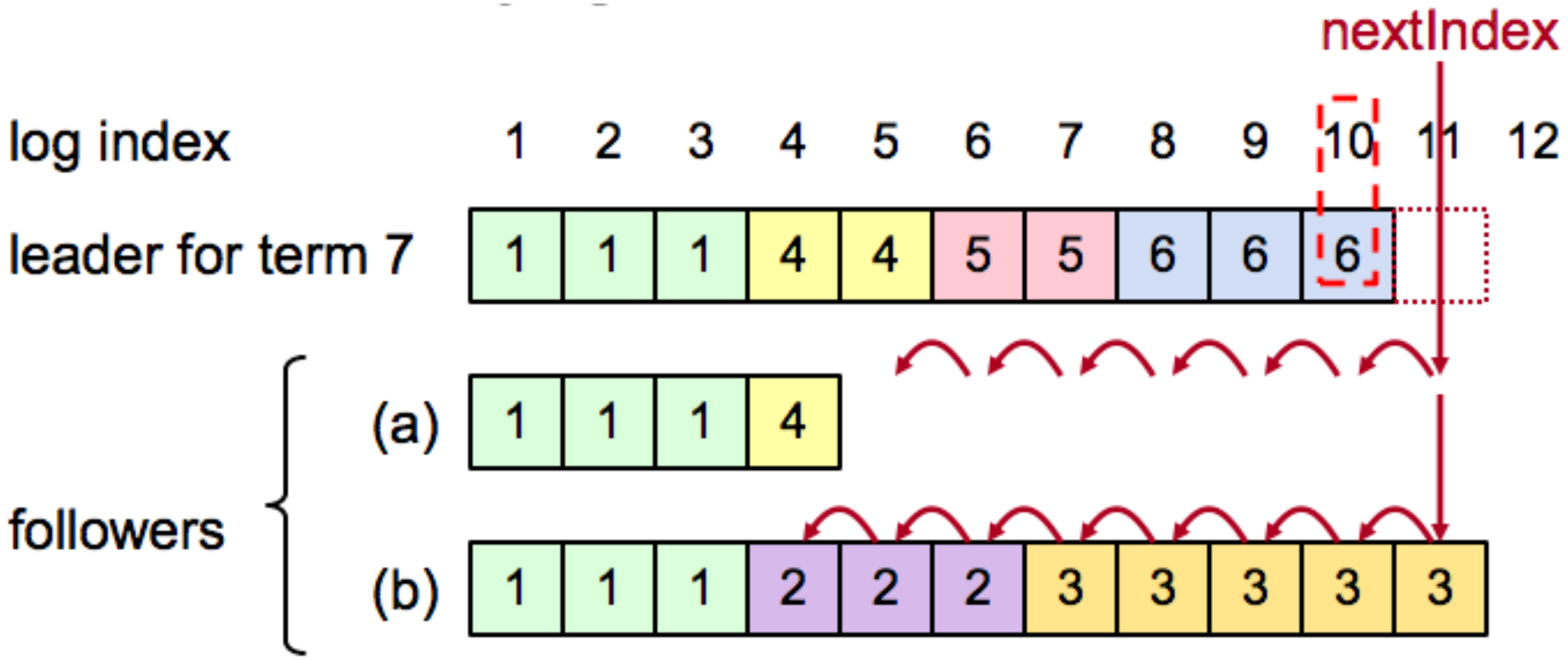
{action: ‘remove’, item: ‘tip’}	

Exactly-Once semantics

- **What if leader crashes after executing command, but before responding?**
  - client embeds a unique id in each command
  - Server includes id in log entry
  - Before accepting command, leader's state machine checks its log for entry with that id
  - If id found in log, ignore new command, return response from old command

Leader changes can result in log inconsistencies

Repairing Follower Logs



Thanks

Referencias

- In Search of an Understandable Consensus Algorithm - https://raft.github.io/raft.pdf
- Raft: Consensus for Rubyists by Patrick Van Stee - https://www.youtube.com/watch?v=IsPxhZ2IsWw
- Raft lecture (Raft user study) - https://www.youtube.com/watch?v=YbZ3zDzDnrw
- Toggle navigation
- The Secret Lives of Data - http://thesecretlivesofdata.com/raft/
- https://www.confluent.io/blog/using-logs-to-build-a-solid-data-infrastructure-or-why-dual-writes-are-a-bad-idea/