Replication and Consensus

CS-438
Decentralized Systems Engineering

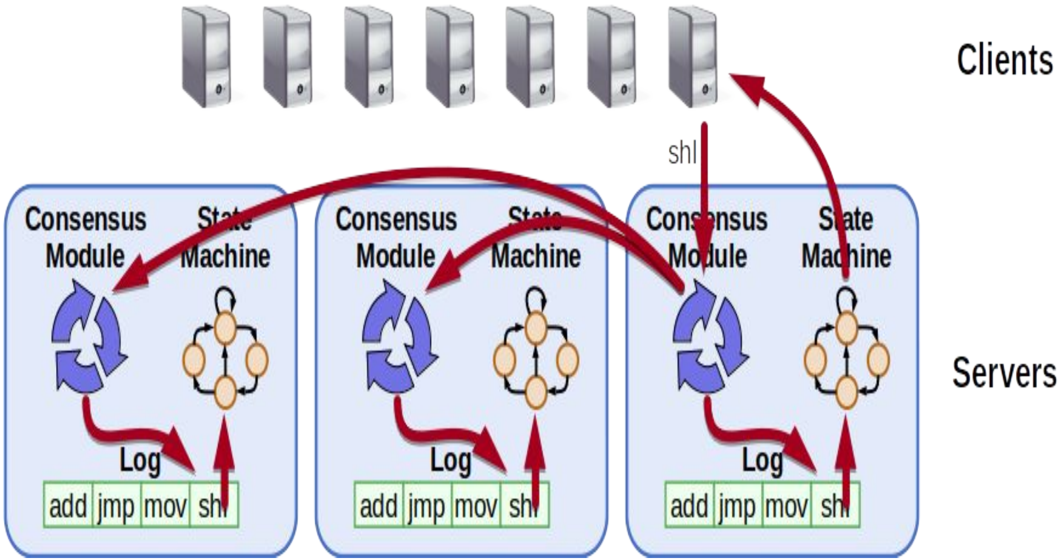
Slide credits: Pasindu Tennage

Replicate Data

- Why replicate?
  - Avoid single point of failure
- What are the challenges of replication?
  - Maintaining strong consistency
- What are the consistency models?
  - Strongly consistent
  - Eventually consistent
  - …
- What applications use strong consistency
  - Boxwood, Zoo-Keeper, Spanner

Replicated Log

- Execute a set of commands in the same order
- Total ordering
  - Consensus
- Majority assumption



Source: Raf user study, John Ousterhout and Diego Ongaro, Stanford University

Failure Modes

- Crash fault: process crashes at time *t* and never recovers after that time
- Omission: process does not send (or receive) a message that it is supposed to send (or receive)
- Crash recovery: A process that crashes and recovers a finite number of times is correct in this model
- Byzantine: may deviate in any conceivable way from the algorithm

Failure Modes

- Crash fault: process crashes at time *t* and never recovers after that time
- Omission: process does not send (or receive) a message that it is supposed to send (or receive)
- Crash recovery: A process that crashes and recovers a finite number of times is correct in this model
- Byzantine: may deviate in any conceivable way from the algorithm

Network Model

△: one way latency of a message from node p to node q

GST: Global stabilization time - time after which each message sent from node p to q is received within bounded △

- Synchronous: for the entire execution △ is bounded and all messages are received within a bounded △
- Partially Synchronous: after GST there exists a bounded △ such that all messages are received within △ (no guarantee before the GST)
- Asynchronous: there is no upper bound on the △

Network Model

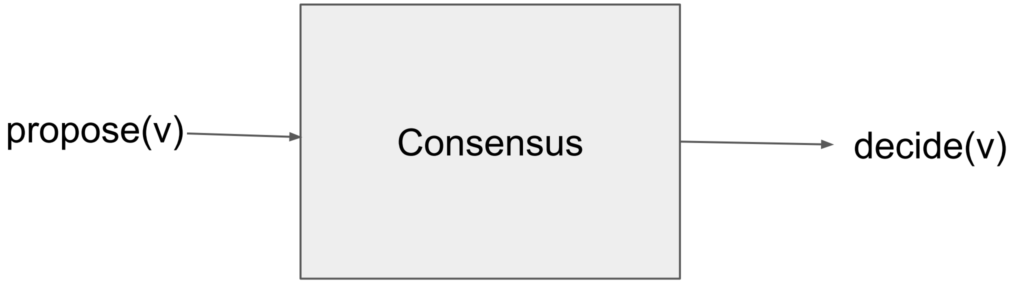
△: one way latency of a message from node p to node q

GST: Global stabilization time - time after which each message send from node p to q is received within bounded △

- Synchronous: for the entire execution △ is bounded and all messages are received within a bounded △
- Partially Synchronous: after GST there exists a bounded △ such that all messages are received within △ (no guarantee about before the GST)
- Asynchronous: there is no upper bound on the △

Consensus Interface

- method Propose(v): propose v
- indication Decide(v): decide on the value v



Consensus: Properties

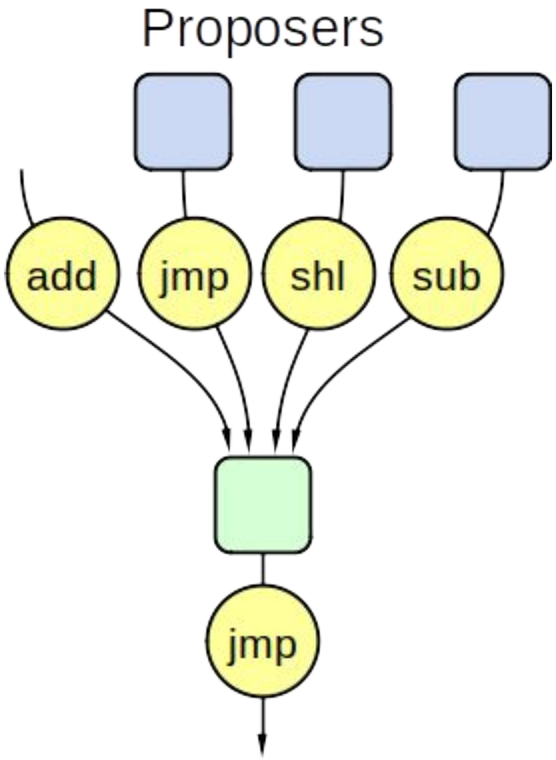
- Validity: Decided value should be one of the values proposed by a replica
- Agreement: No two nodes decide on different values
- Termination: Each node eventually decides

Strawman 1: Single acceptor

- [Strawman: a solution attempt that partially solves the problem] – not a full solution and has drawbacks

- Problem: Single point of failure

- Does not solve consensus



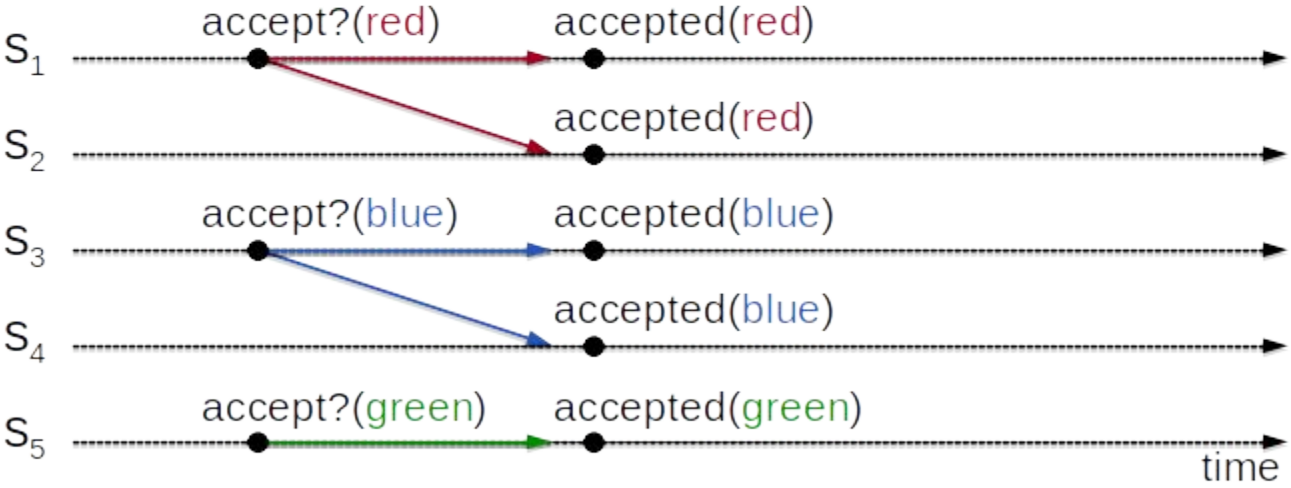
Source: Raf user study, John Ousterhout and Diego Ongaro, Stanford University

Strawman 2: Quorum of acceptors

- Have a quorum of acceptors:
  - Each acceptor accepts the first value
  - Value chosen by a majority of acceptors

- Split votes

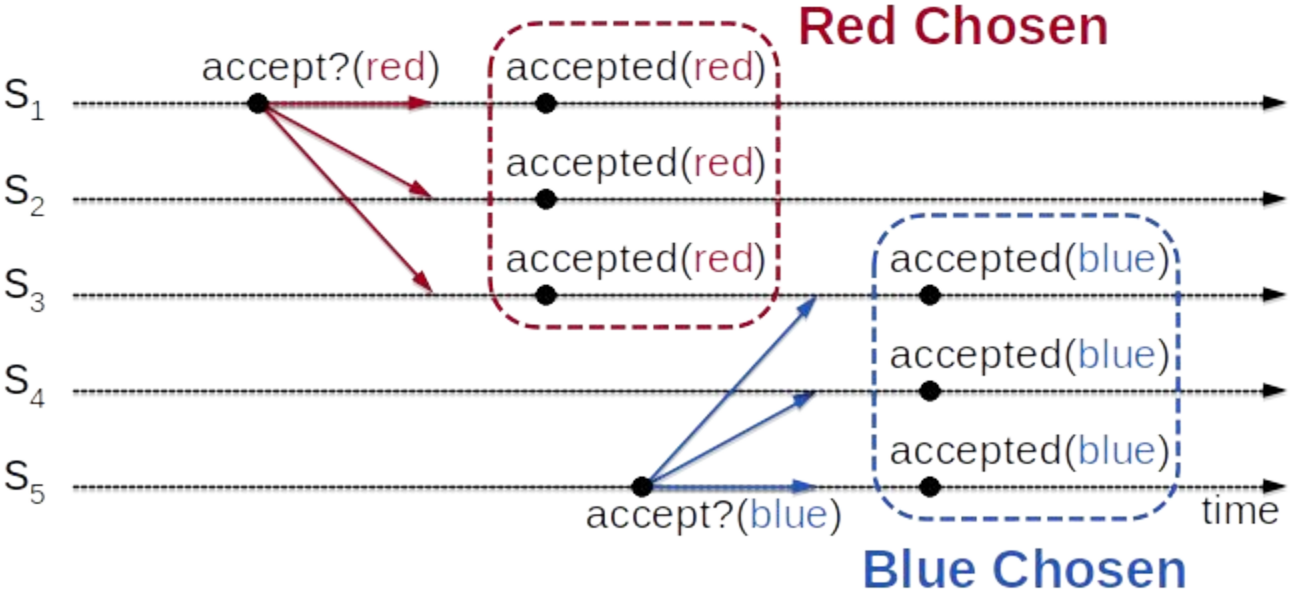
- Does not solve consensus



Source: Raf user study, John Ousterhout and Diego Ongaro, Stanford University

Strawman 3: Accept multiple values

- How to determine the safe values to accept?
- Does not solve consensus
- We need a two phase protocol



Source: Raf user study, John Ousterhout and Diego Ongaro, Stanford University

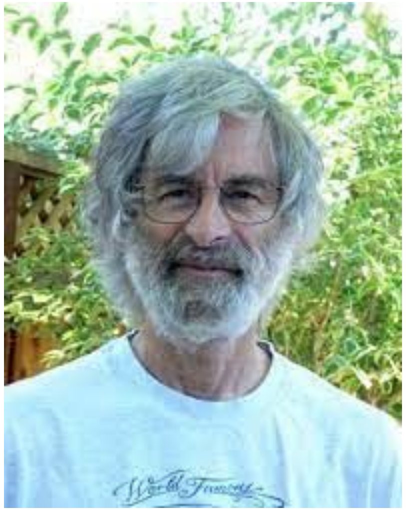
Paxos

- The Part-Time Parliament
  Lamport, Leslie. "The part-time parliament." *Concurrency: the Works of Leslie Lamport*. 277-317.

- Paxos made simple
  Lamport, Leslie. "Paxos made simple." *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (2001): 51-58.

- Crash fault tolerant

- Partial synchronous (agreement holds under asynchrony, but termination holds only under partial synchrony)



Paxos

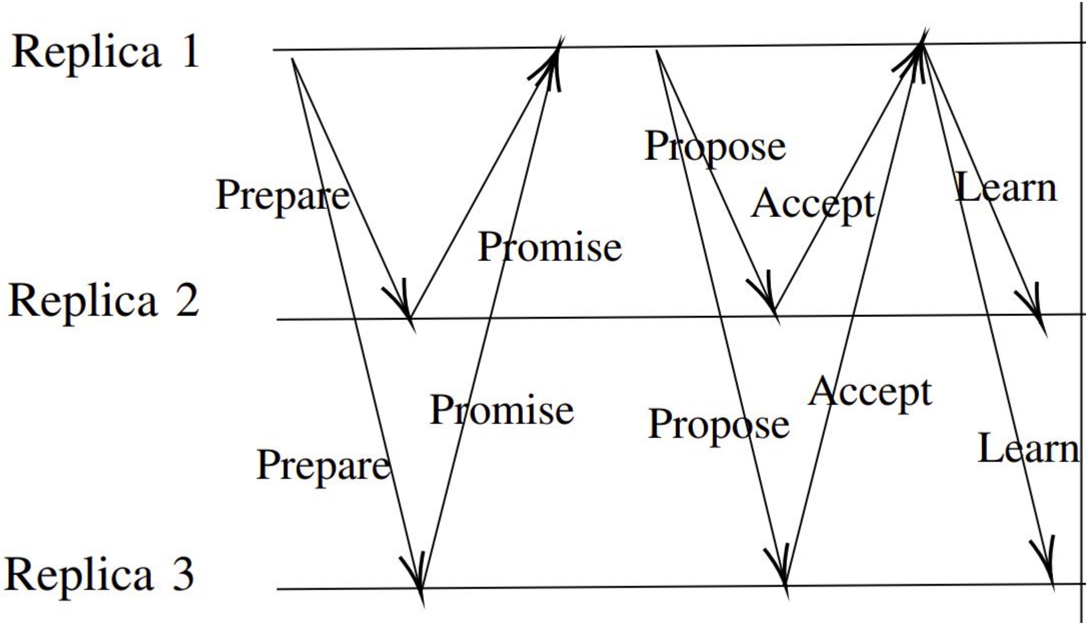
- Proposers:
  - Handle client requests
  - Propose(v)
- Acceptors
  - Respond to proposer messages
  - Store chosen value
- In our setup, each replica can act as a proposer and acceptor

Ballot Number

- A unique number (ballot)

Paxos: a two phase protocol

- Prepare-Promise
  - Find the safe value
  - Block older proposals that are not yet completed
- Propose-Accept
  - Propose the safe value
- [Learn]



Source: Baxos

Paxos

- Proposer: upon Propose(value):
  - Chose n > minBallot; broadcast Prepare(n)

- Acceptor: upon receiving Prepare(n) from p:
  - if n>minBallot
    - minBallot = n
    - send p Promise(n, acceptedBallot, acceptedValue)

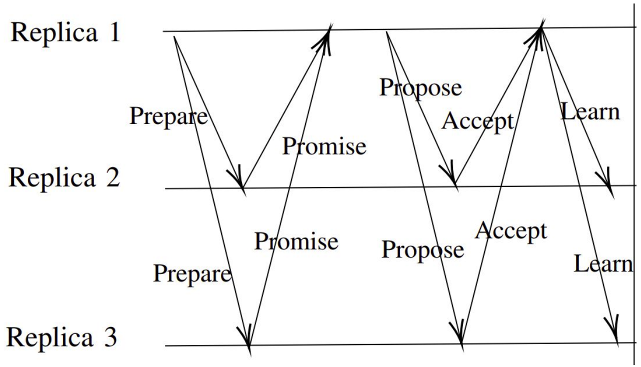
- Proposer: upon receiving a majority Promise(n, b, v):
  - if b==-1 for all Promise messages: value = value
  - else: value = v where b is the highest from the {(b,v)}
  - Broadcast Propose(n, value)

- Acceptor: upon receiving Propose(n,v) from p:
  - if n>= minBallot
    - acceptedBallot = n; acceptedValue=v;
    - send p Accept(n, v)

- Proposer: upon receiving a majority Accept(n, v):
  - decide (v); broadcast (Learn, v)

Replica State

- minBallot = 0
- acceptedBallot =-1
- acceptedValue = nil



Source: Baxos

Example Executions

- Case 1: No previously accepted value
- Case 2: Previously decided value
- Case 3: Previously accepted, but not decided value

Paxos

- Proposer: upon Propose(value):
  - Chose n > minBallot; broadcast Prepare(n)

- Acceptor: upon receiving Prepare(n) from p:
  - if n>minBallot
    - minBallot = n
    - send p Promise(n, acceptedBallot, acceptedValue)

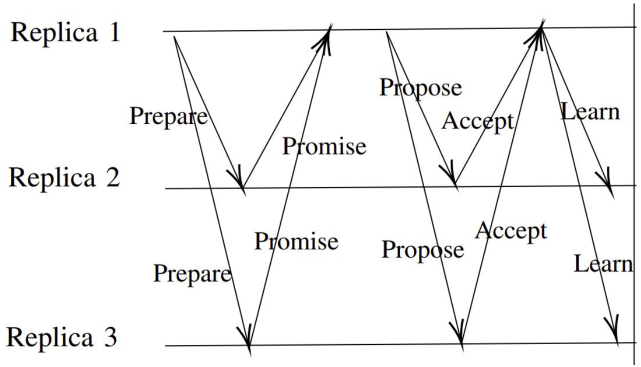
- Proposer: upon receiving a majority Promise(n, b, v):
  - if b==-1 for all Promise messages: value = value
  - else: value = v where b is the highest from the {(b,v)}
  - Broadcast Propose(n, value)

- Acceptor: upon receiving Propose(n,v) from p:
  - if n>= minBallot
    - acceptedBallot = n; acceptedValue=v;
    - send p Accept(n, v)

- Proposer: upon receiving a majority Accept(n, v):
  - decide (v); broadcast (Learn, v)

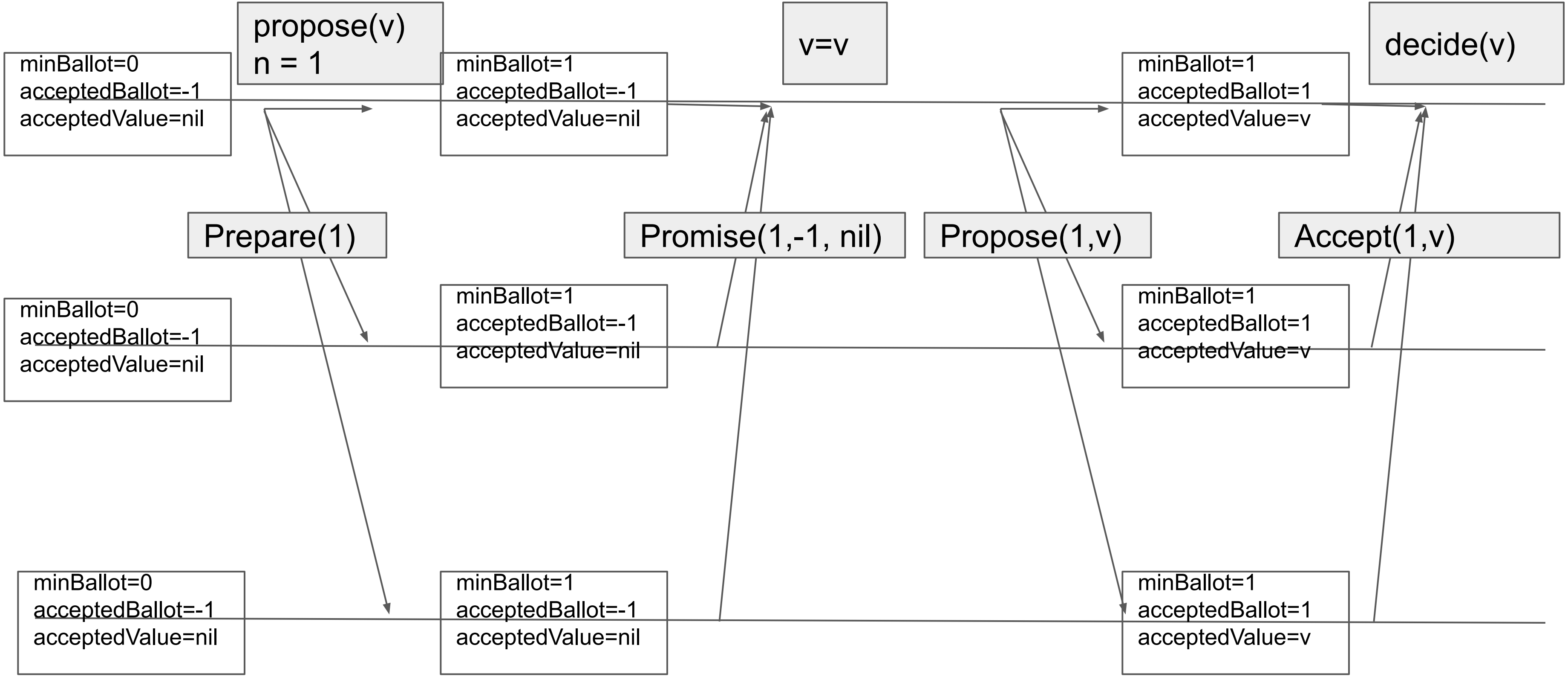
Replica State

- minBallot = 0
- acceptedBallot =-1
- acceptedValue = nil

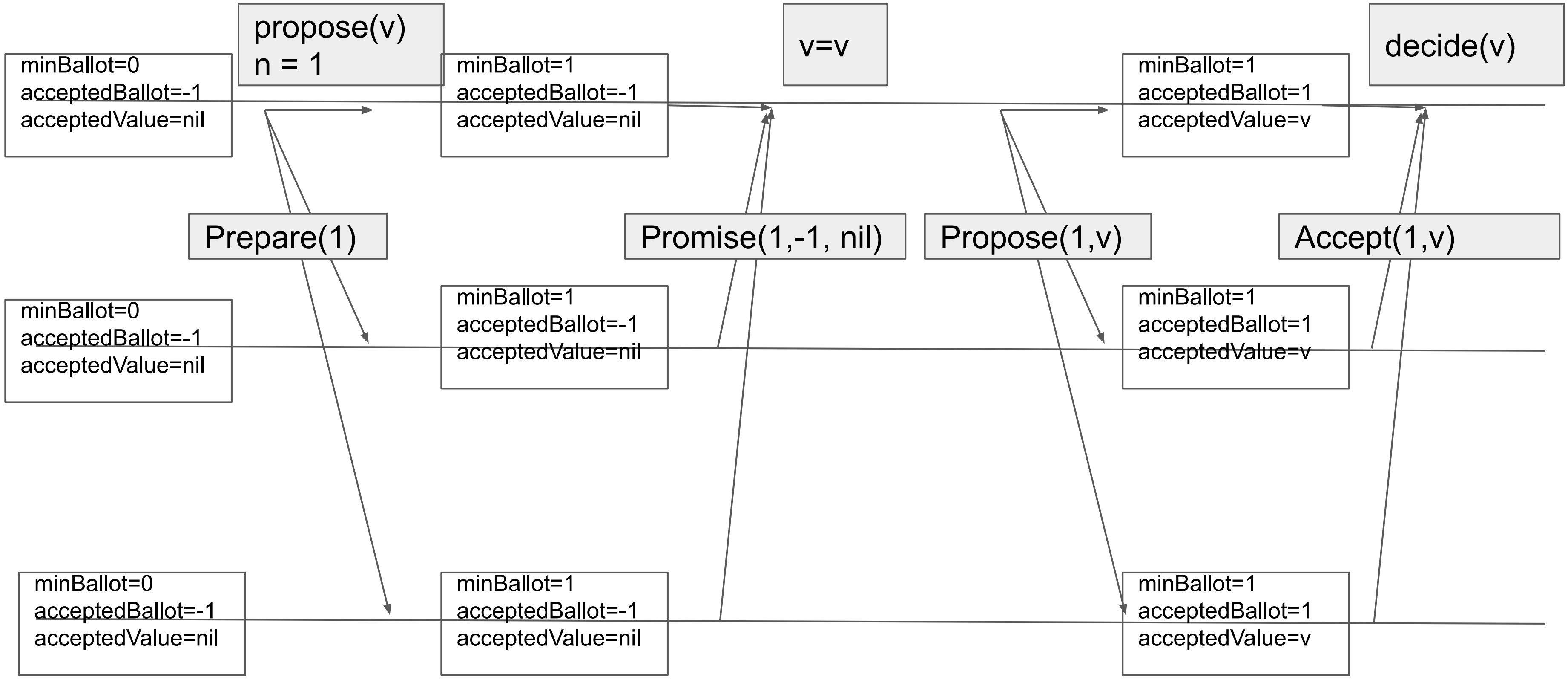


Source: Baxos

Case 1: No previously accepted value



Case 2: Previously decided value: initial state



Paxos

- Proposer: upon Propose(value):
  - Chose n > minBallot; broadcast Prepare(n)

- Acceptor: upon receiving Prepare(n) from p:
  - if n>minBallot
    - minBallot = n
    - send p Promise(n, acceptedBallot, acceptedValue)

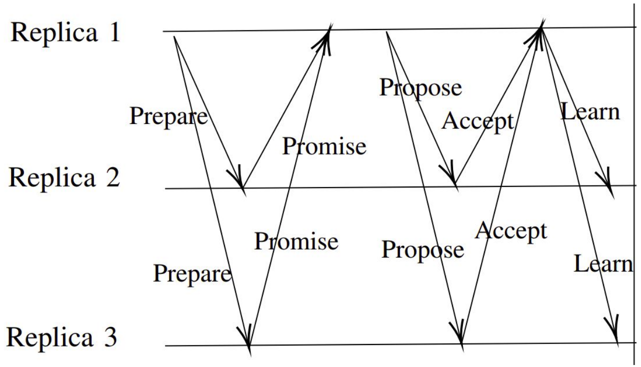
- Proposer: upon receiving a majority Promise(n, b, v):
  - if b==-1 for all Promise messages: value = value
  - else: value = v where b is the highest from the {(b,v)}
  - Broadcast Propose(n, value)

- Acceptor: upon receiving Propose(n,v) from p:
  - if n>= minBallot
    - acceptedBallot = n; acceptedValue=v;
    - send p Accept(n, v)

- Proposer: upon receiving a majority Accept(n, v):
  - decide (v); broadcast (Learn, v)

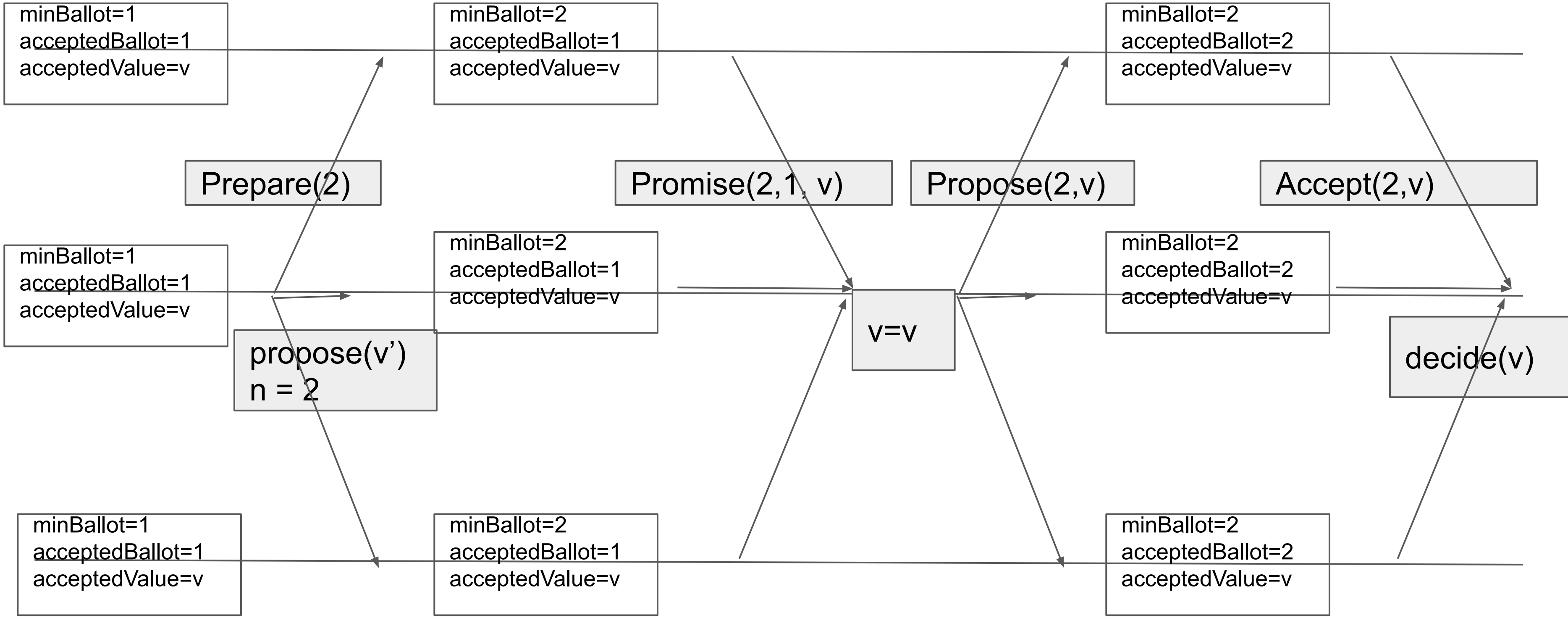
Replica State

- minBallot = 0
- acceptedBallot =-1
- acceptedValue = nil

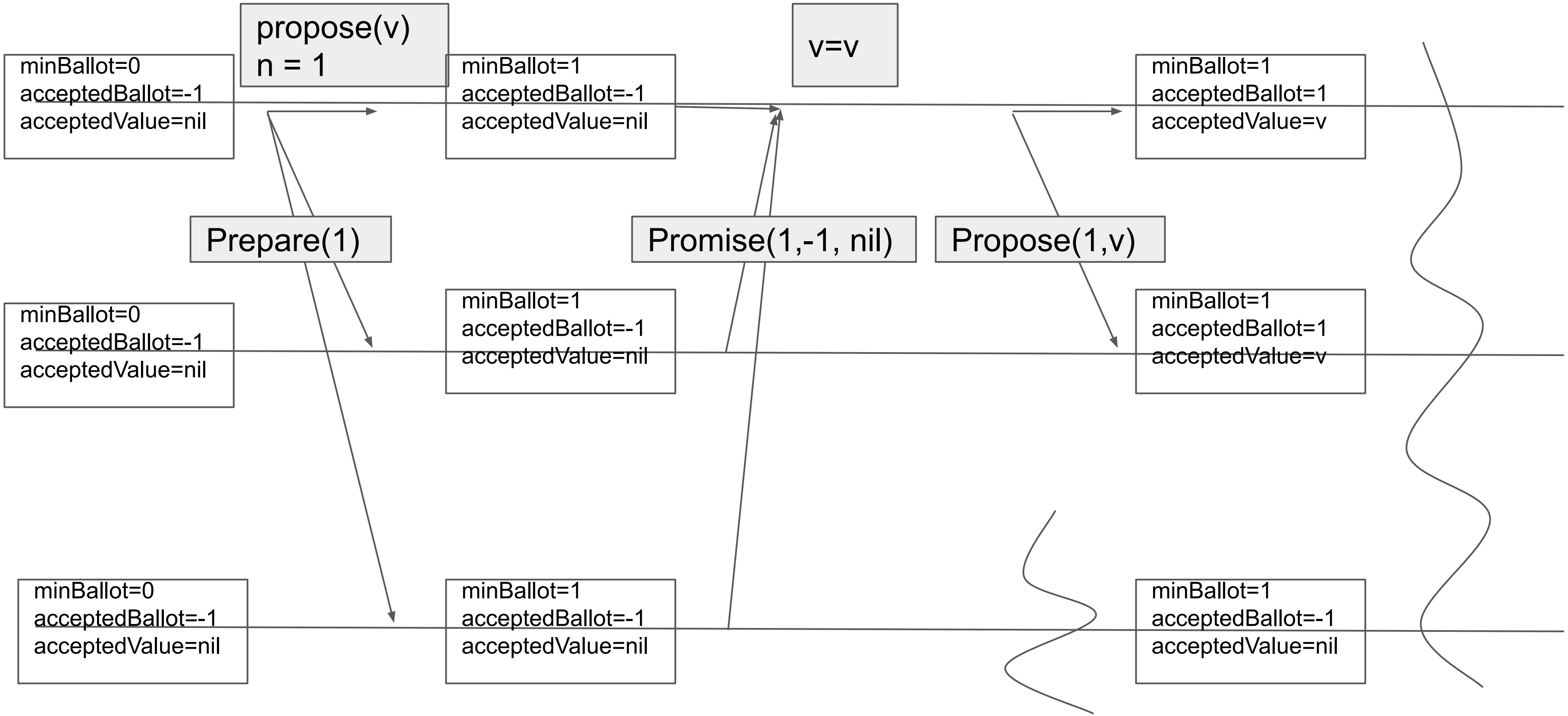


Source: Baxos

Case 2: Previously decided value: cntd



Case 3: Previously accepted, but not decided value



Paxos

- Proposer: upon Propose(value):
  - Chose n > minBallot; broadcast Prepare(n)

- Acceptor: upon receiving Prepare(n) from p:
  - if n>minBallot
    - minBallot = n
    - send p Promise(n, acceptedBallot, acceptedValue)

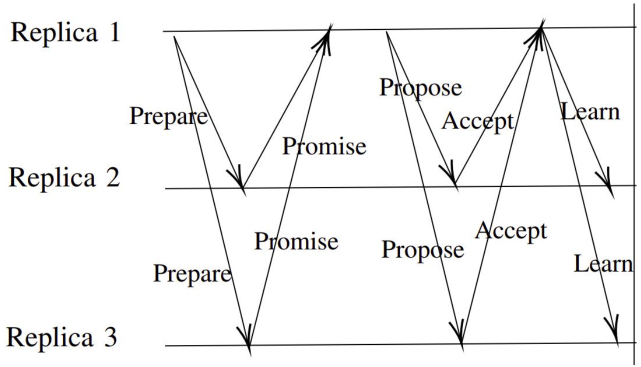
- Proposer: upon receiving a majority Promise(n, b, v):
  - if b==-1 for all Promise messages: value = value
  - else: value = v where b is the highest from the {(b,v)}
  - Broadcast Propose(n, value)

- Acceptor: upon receiving Propose(n,v) from p:
  - if n>= minBallot
    - acceptedBallot = n; acceptedValue=v;
    - send p Accept(n, v)

- Proposer: upon receiving a majority Accept(n, v):
  - decide (v); broadcast (Learn, v)

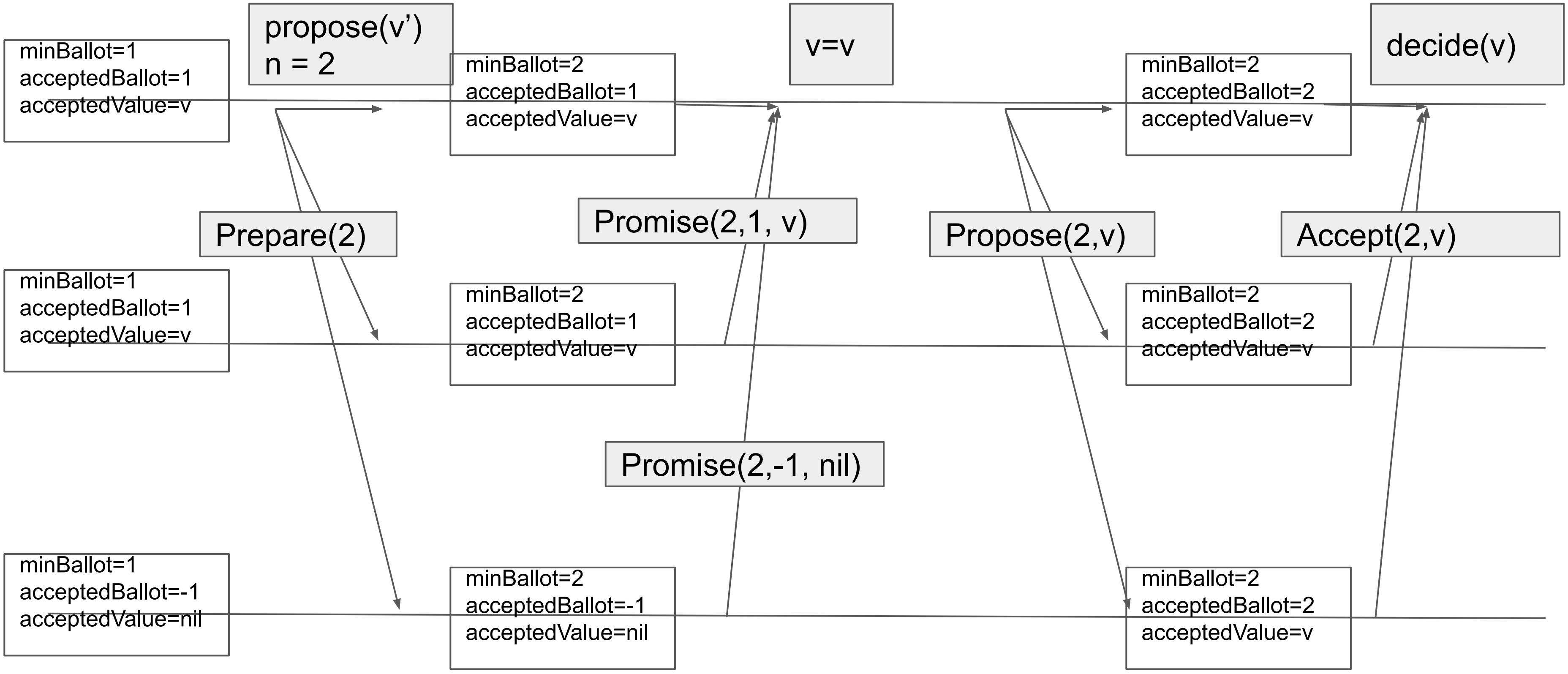
Replica State

- minBallot = 0
- acceptedBallot =-1
- acceptedValue = nil



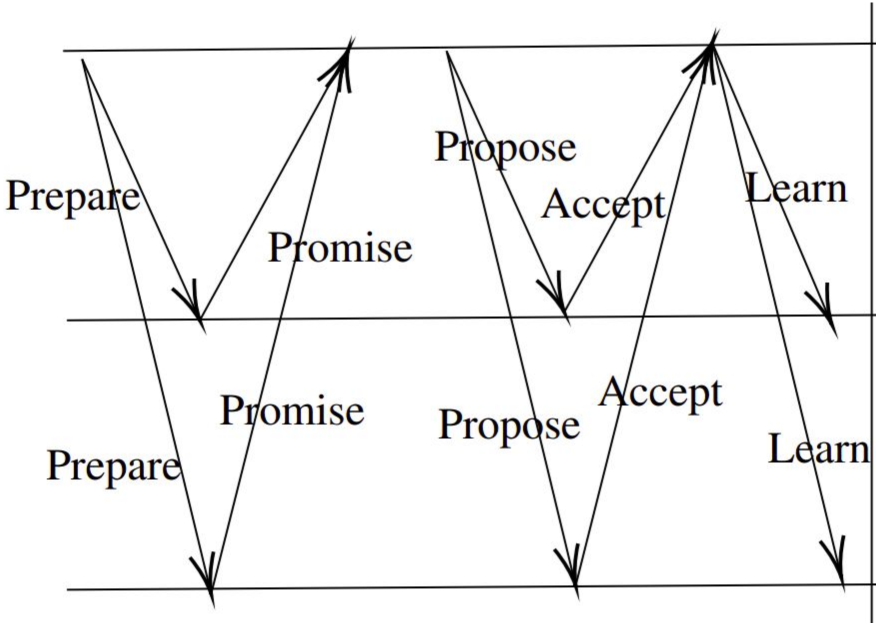
Source: Baxos

Case 3: Previously accepted, but not decided value

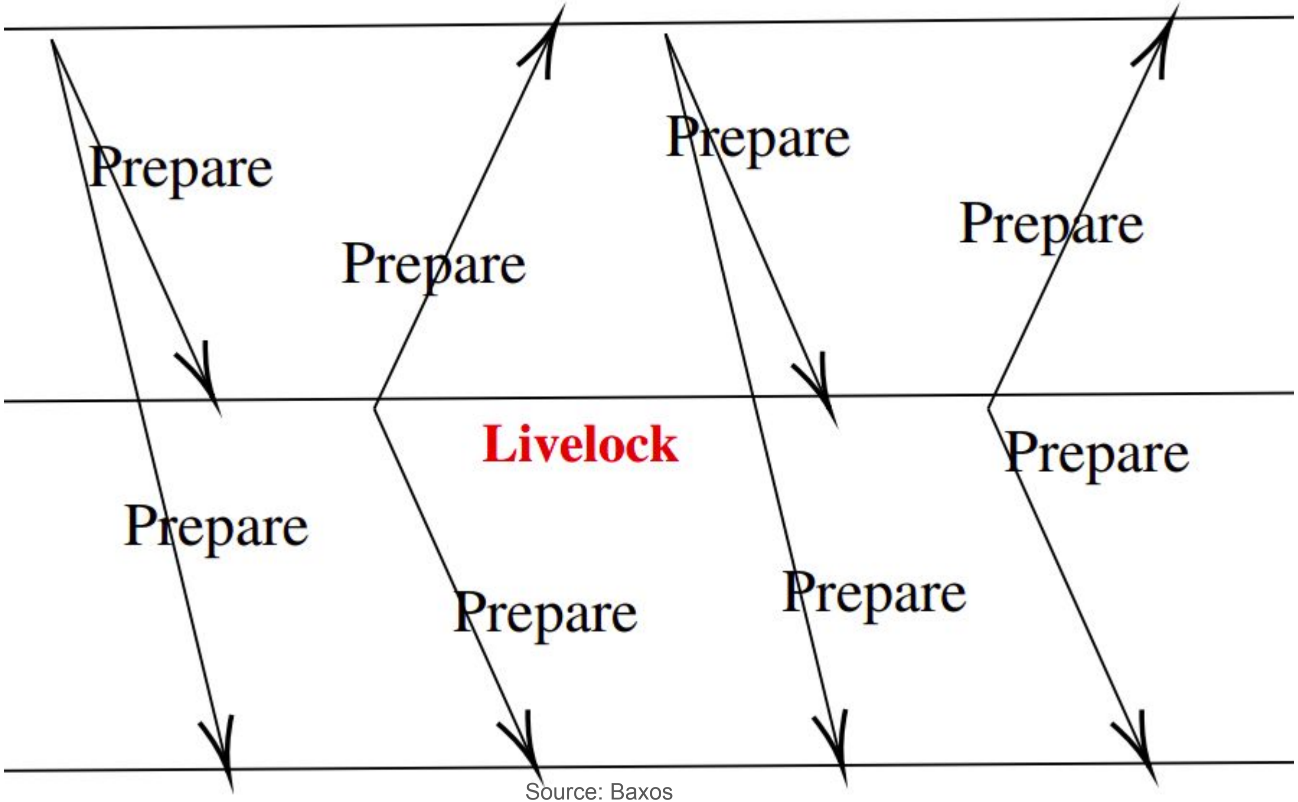


Proofs

- Validity: trivially satisfied
- Agreement: if a value is decided by at least one node, then all the future proposers will learn that value in the prepare-promise phase (quorum intersection)
- Termination: If the network is synchronous for 4△ with only a single proposer



Liveness with multiple leaders



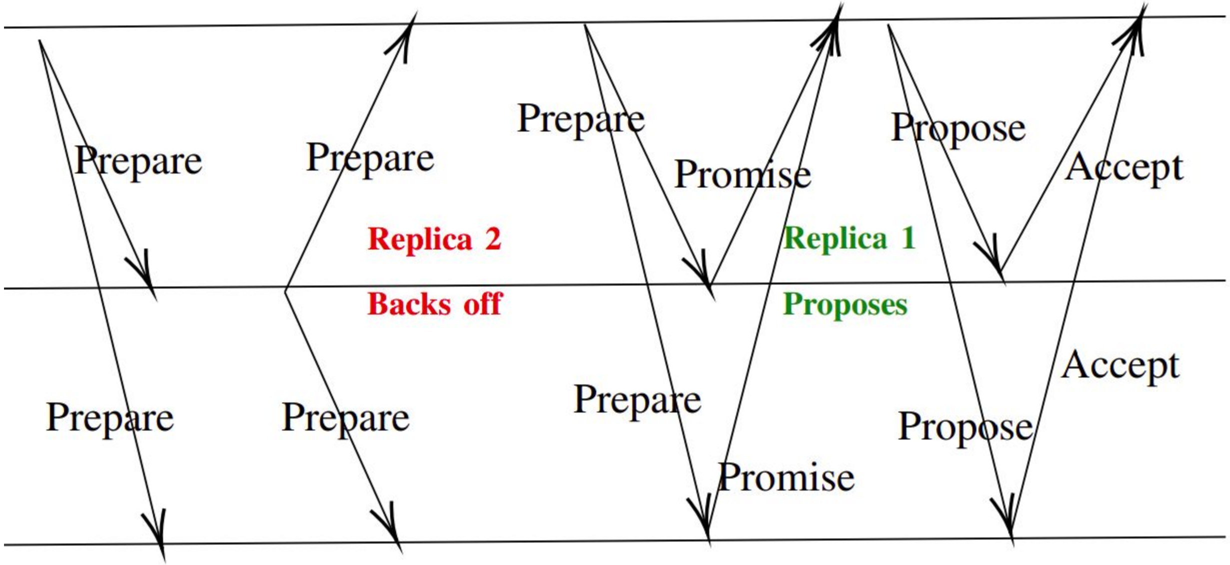
Source: Baxos

Ensuring Liveness

- Leader based Paxos - Multi-Paxos
  - Eliminate Prepare-Promise phase

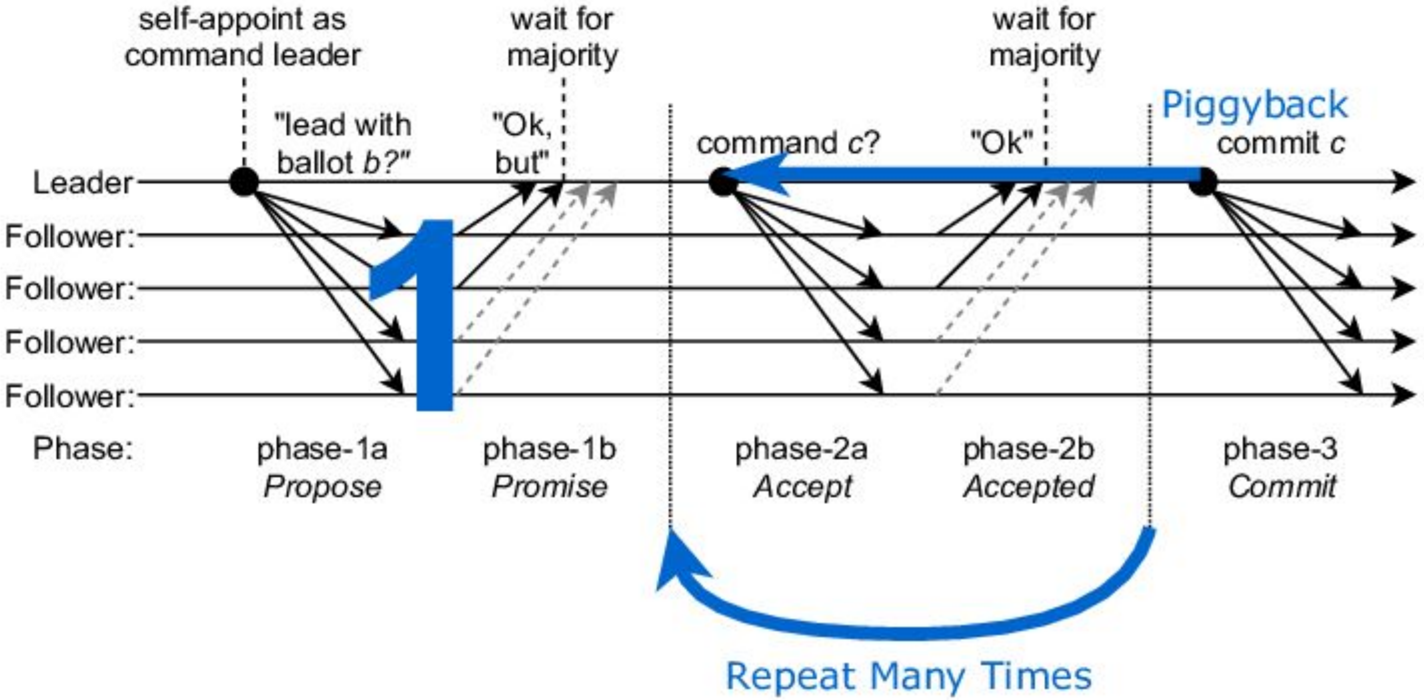
- Random back off - Baxos
  - Optimistic contention handling

Random back off



Source: Baxos

Multi-Paxos



Source: https://www.researchgate.net/figure/Multi-Paxos-optimization_fig1_339998273

Questions?