LogDevice: the Consensus Story

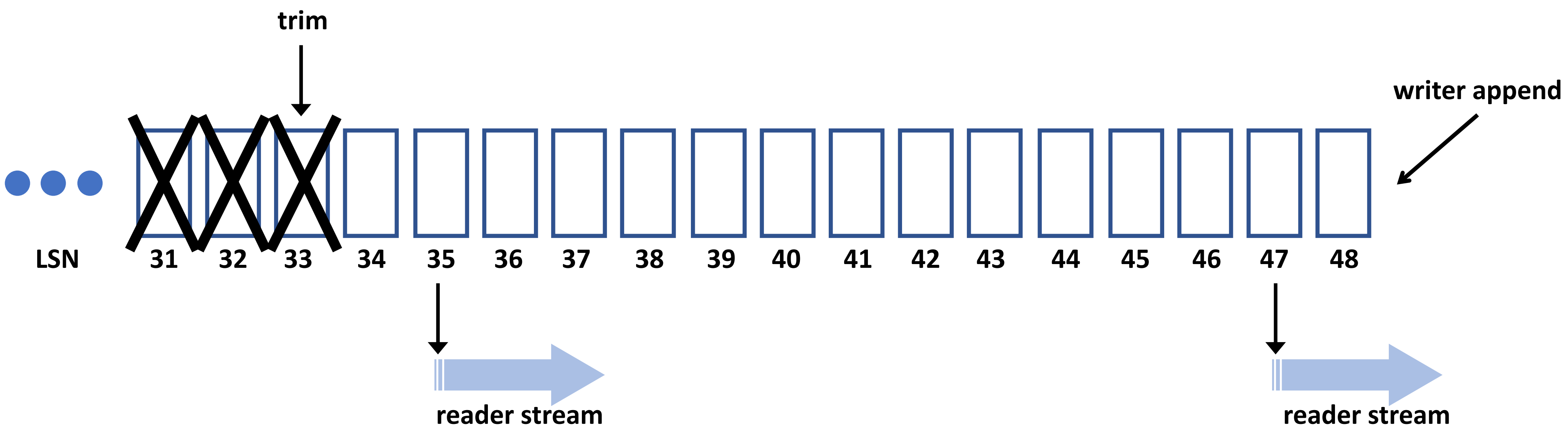
Xi Xiong, LogDevice SWE

LogDevice

- Log data model built on a strongly consistent Paxos consensus engine
- Carefully chosen variants of Paxos to achieve:
  - fault tolerance with fewer copies
  - flexible quorums for highly available, high throughput and low latency steady state replication
  - zero-copy quorum reconfiguration with high availability

Log abstraction

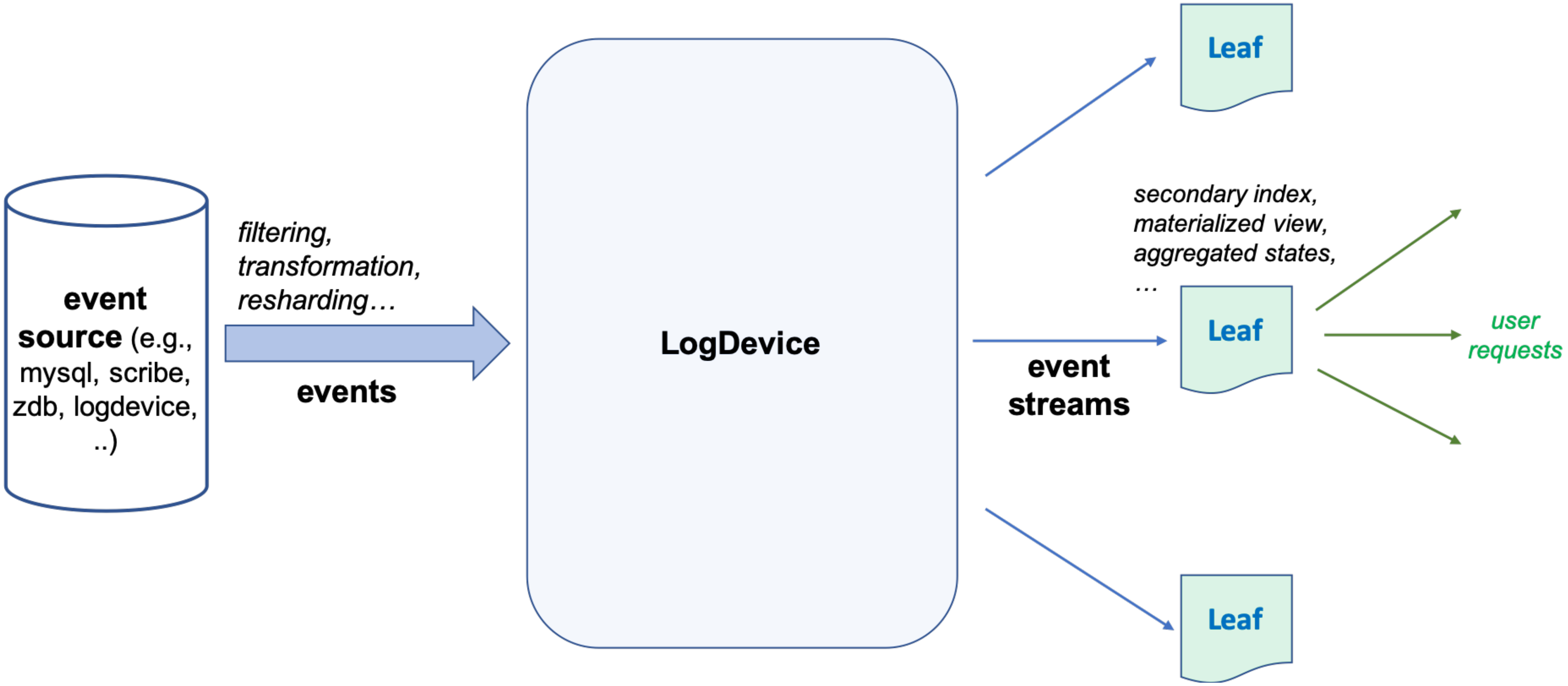
Log data model



Log is the abstraction for reliable communication

- RPC: thrift, etc…
  - require strongest inter-service dependencies (availability, rpc format, etc)
- Log as communication primitive
  - supports fan-out and streaming subscription
  - messages durably replicated and persisted as **ordered** log records
  - messages can be independently replayed again and again by consumers
  - minimal inter-service dependencies
    - consumers can be down for hours or days, can still catch up once up via backfills
    - load isolation: consumer won't overwhelm producer service
    - easier to handle data format changes

Log is the abstraction for distributed state replication and distribution

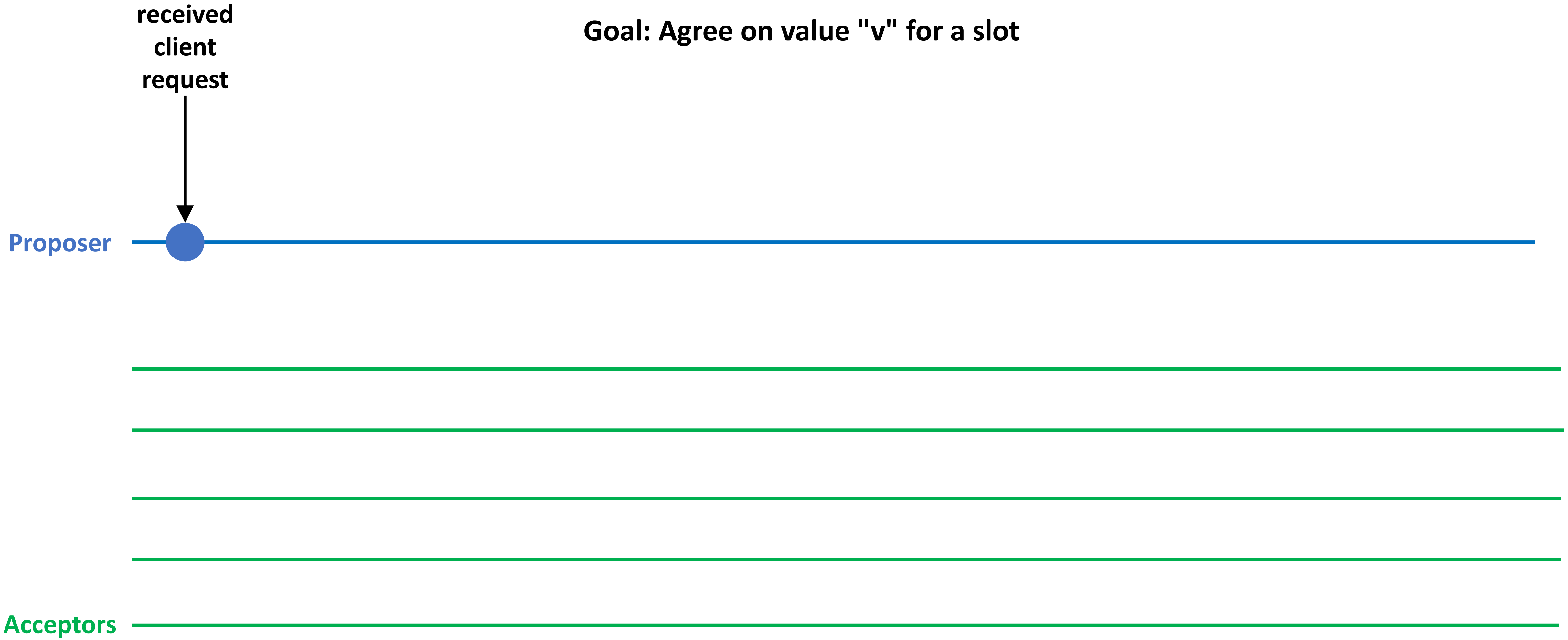


Let's talk Paxos

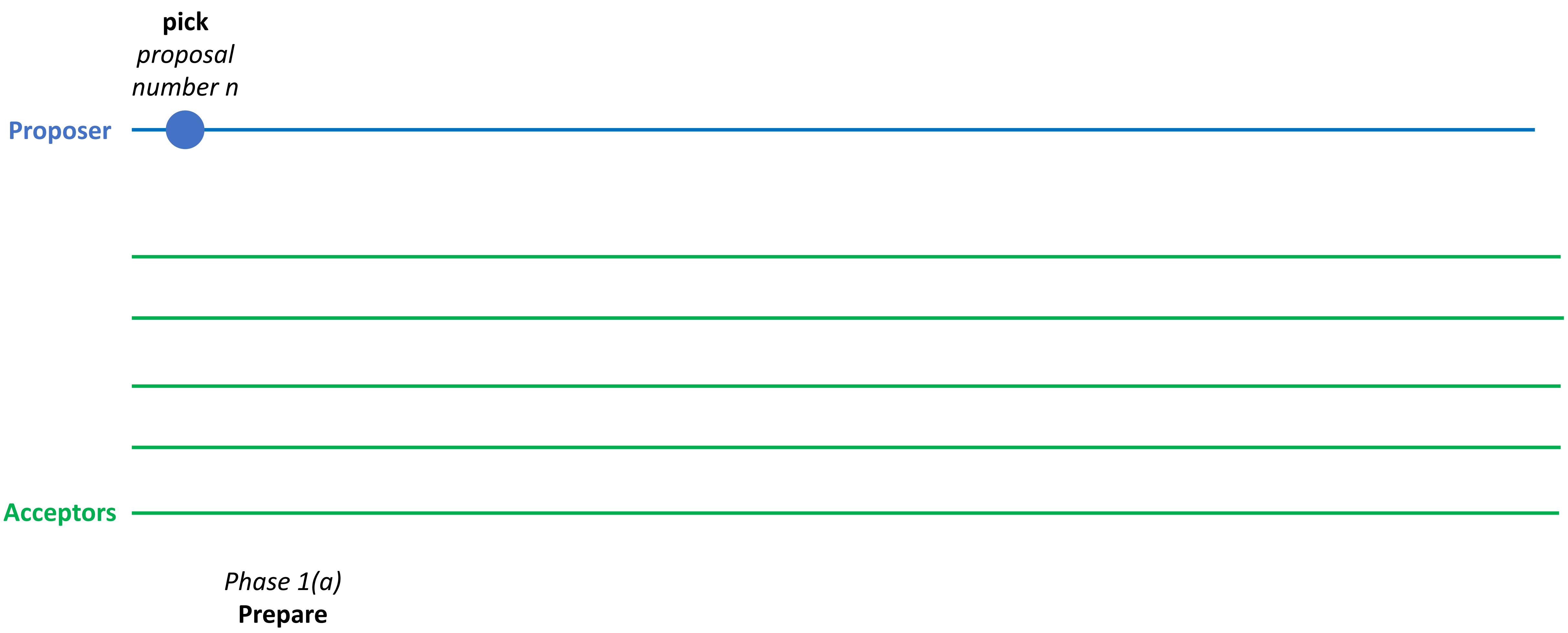
Concepts & Roles

- Proposers: propose value to be chosen
  - value proposed usually on behalf of clients
- Acceptors: agrees and persists decided values
- Learner: a process wish to learn the chosen value

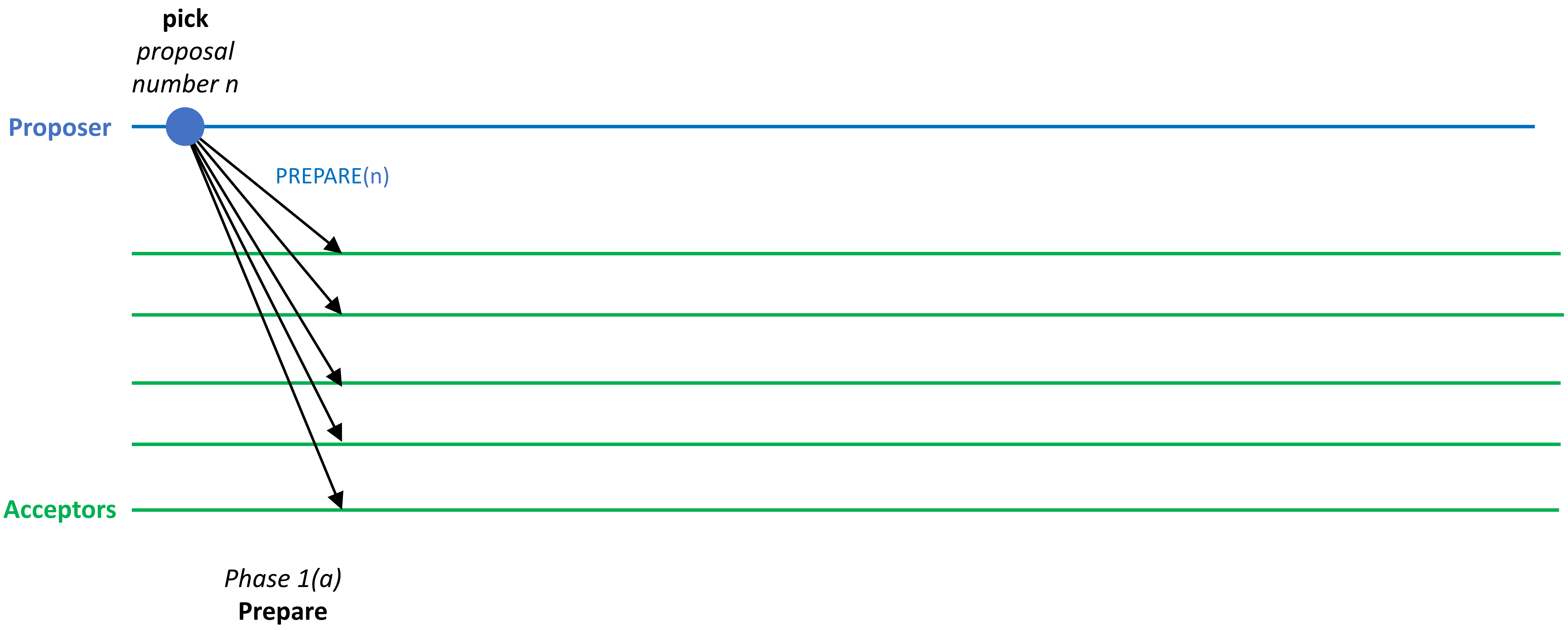
Goal: Agree on value "v" for a slot



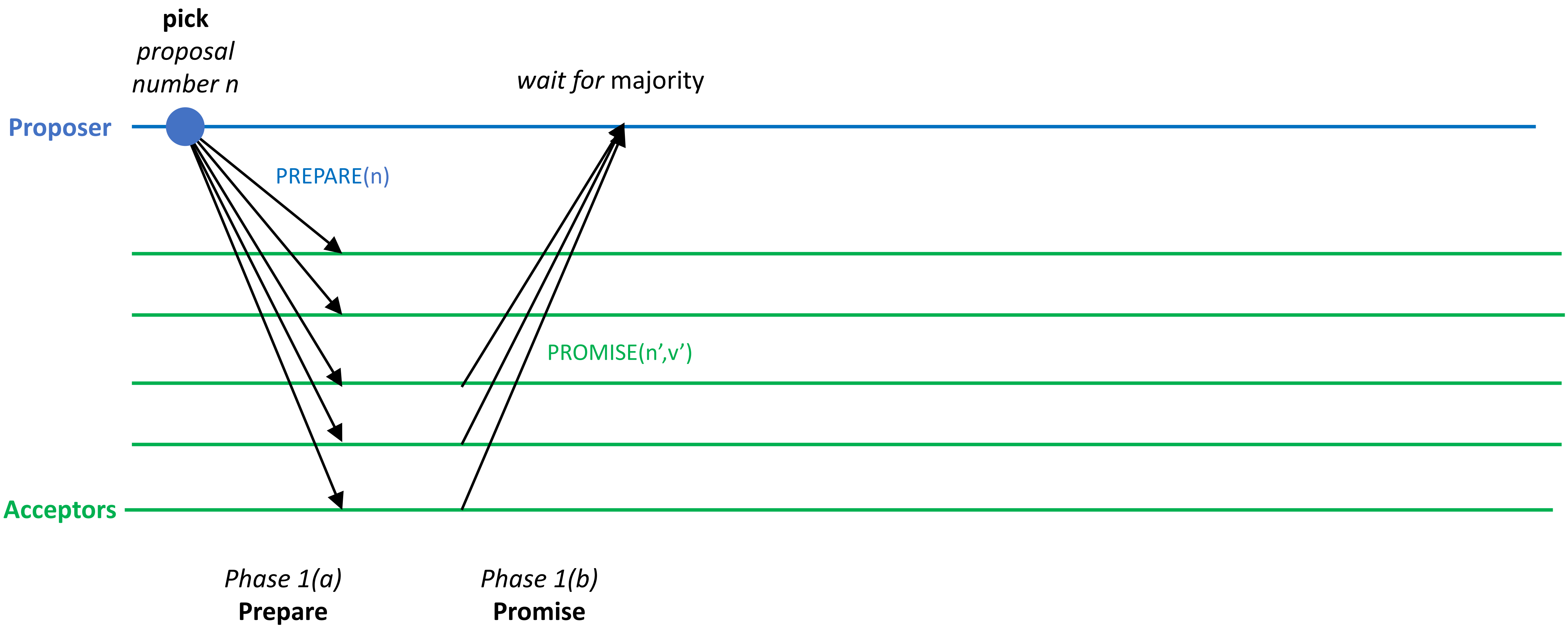
Goal: Agree on value "v" for a slot



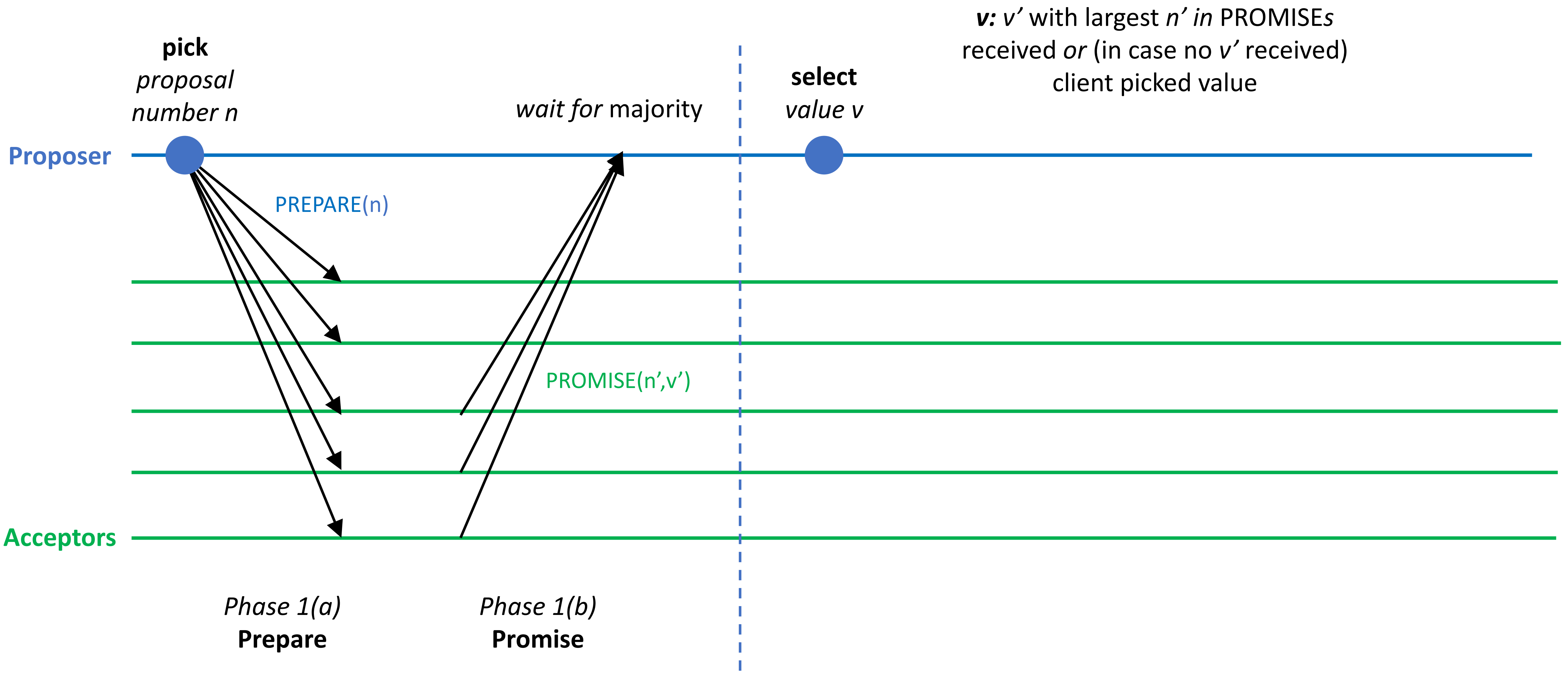
Goal: Agree on value "v" for a slot



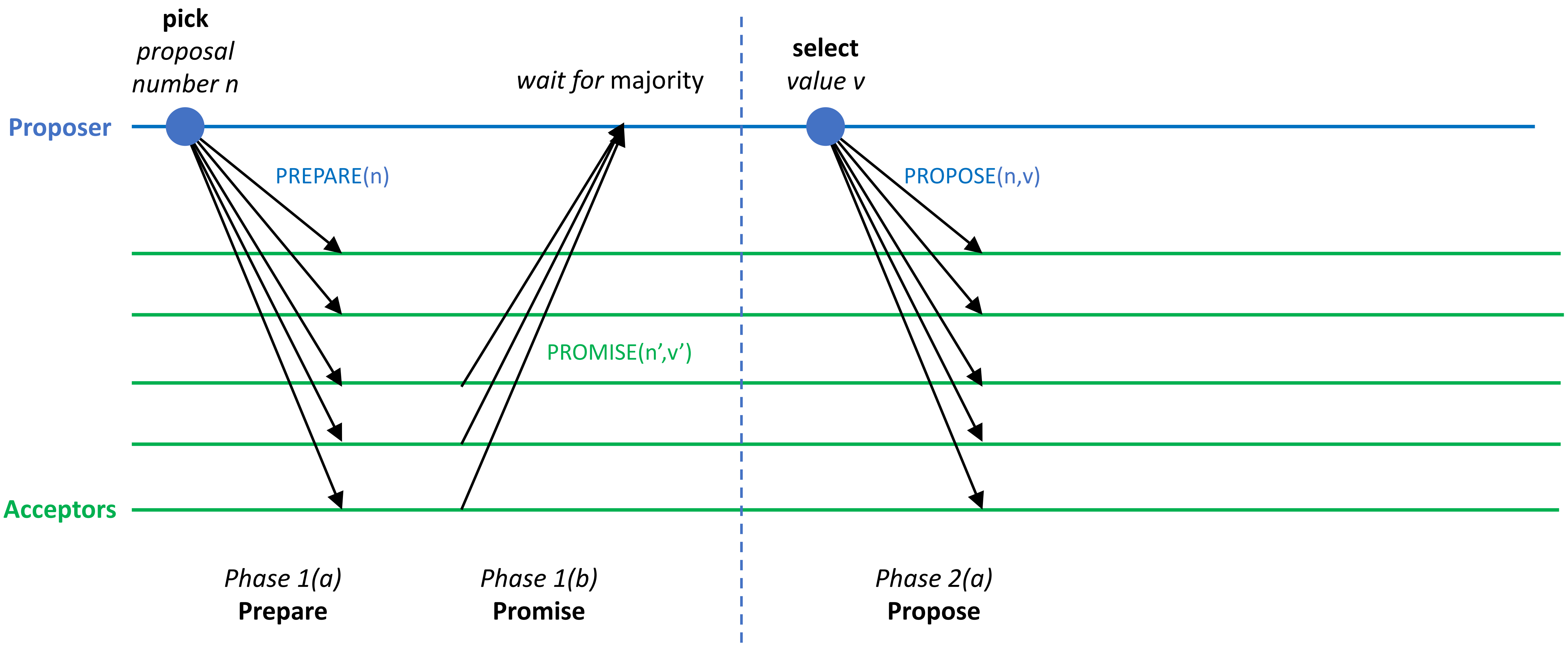
Goal: Agree on value "v" for a slot



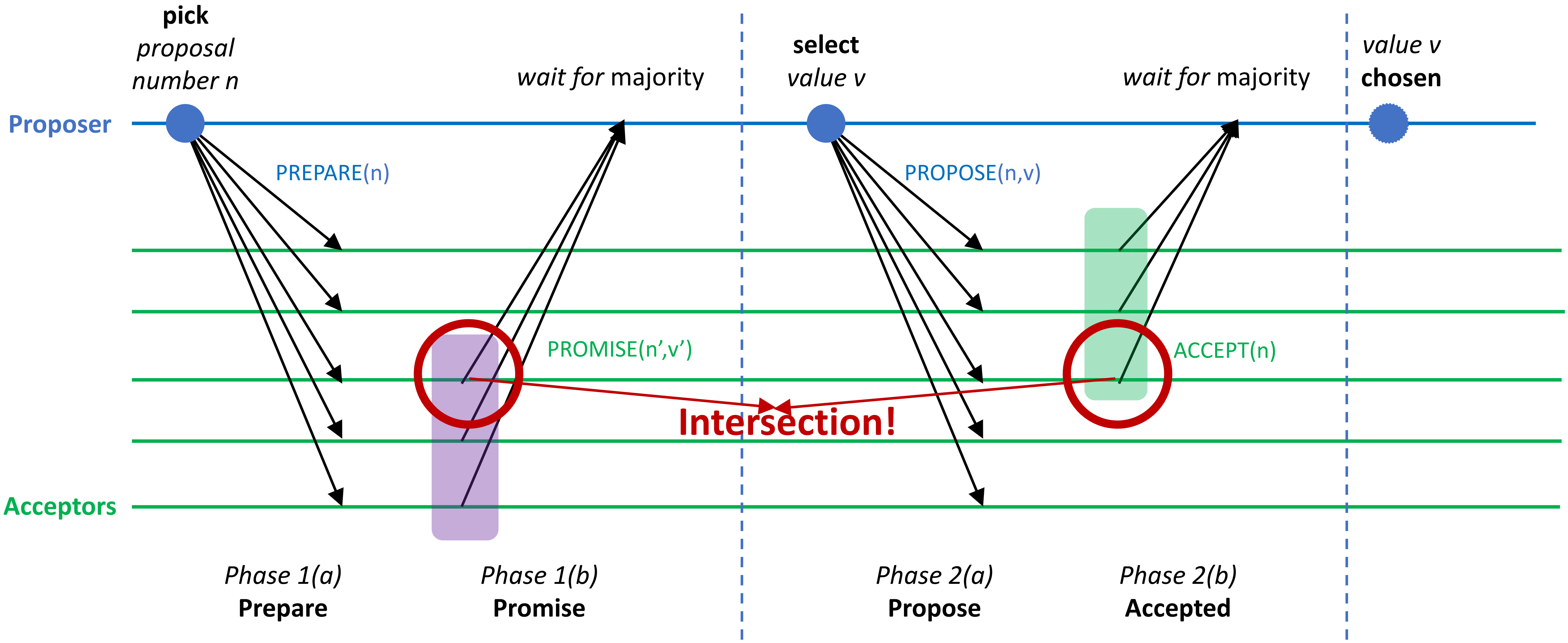
Goal: Agree on value "v" for a slot



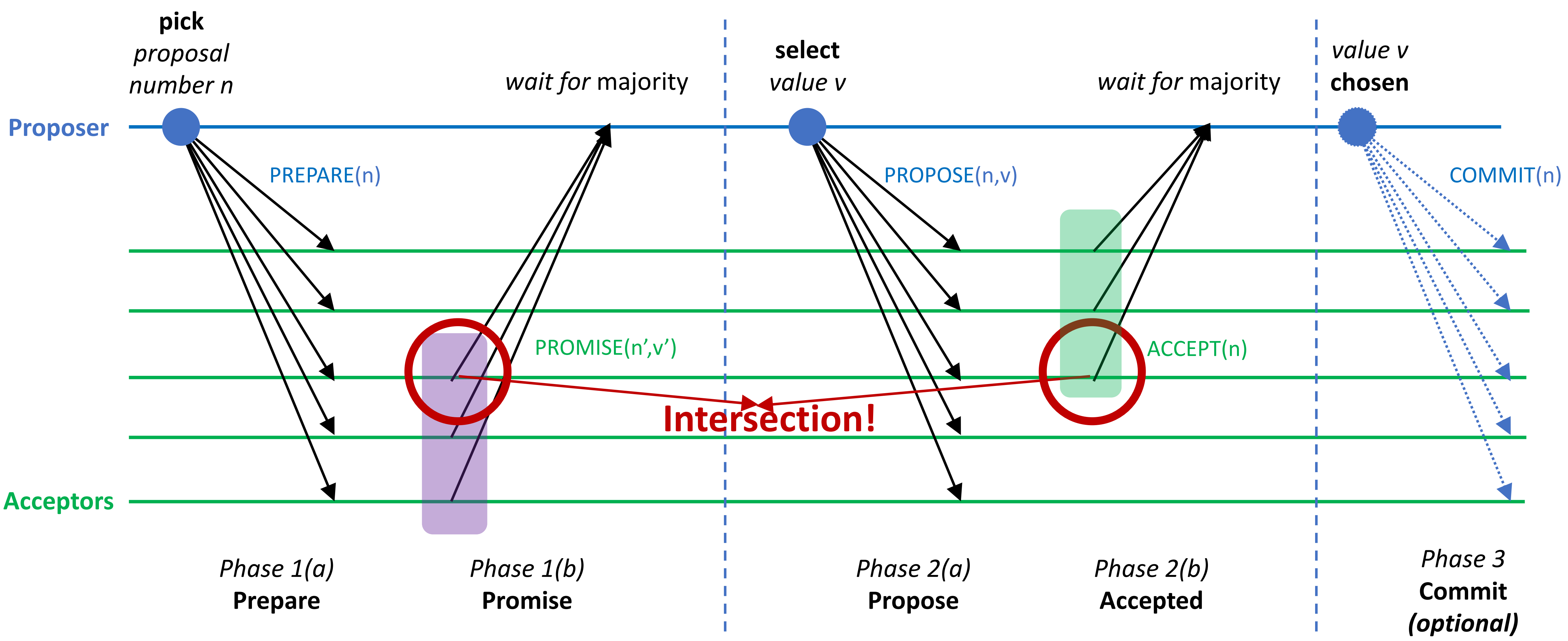
Goal: Agree on value "v" for a slot



Goal: Agree on value "v" for a slot



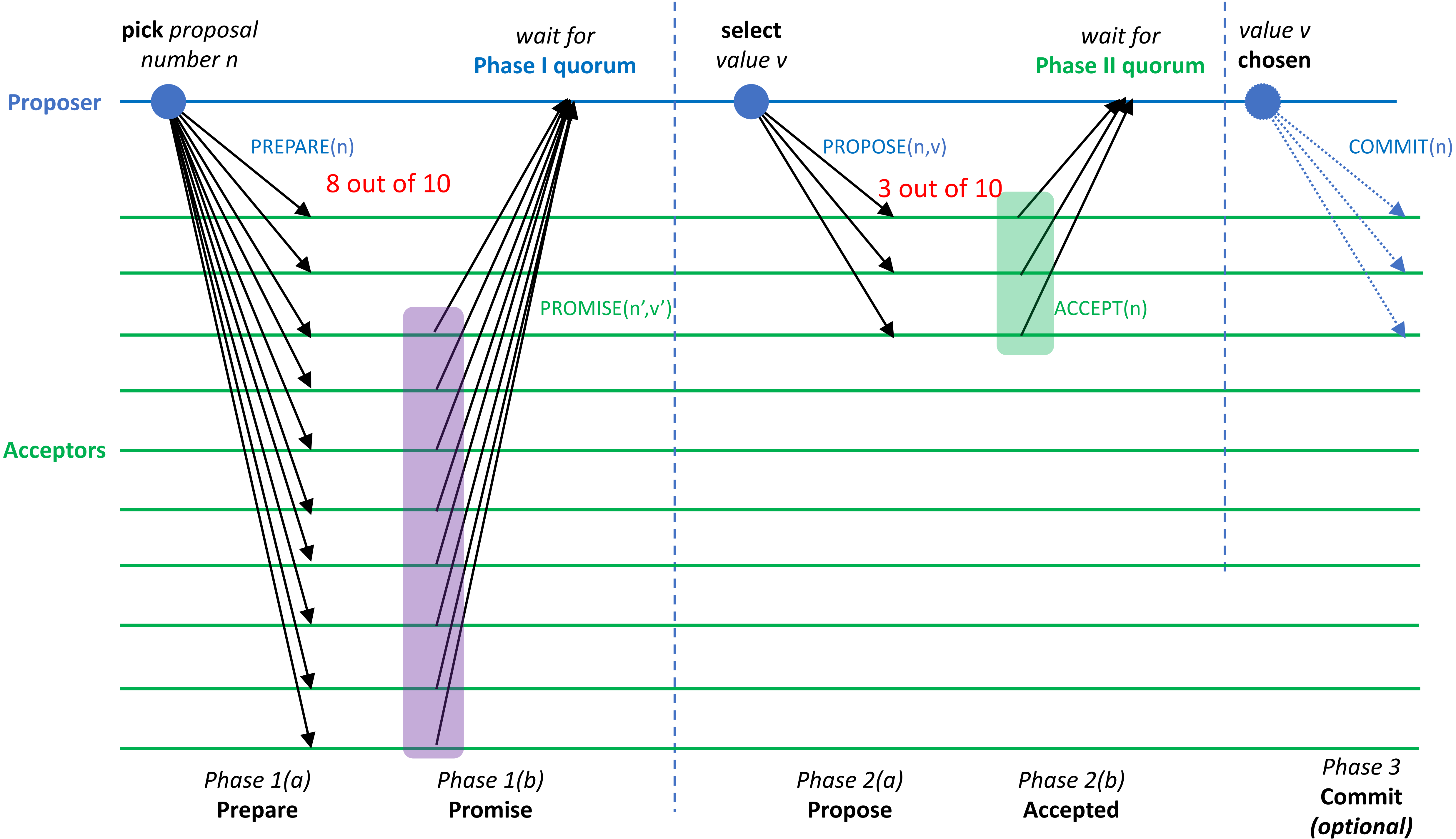
Goal: Agree on value "v" for a slot



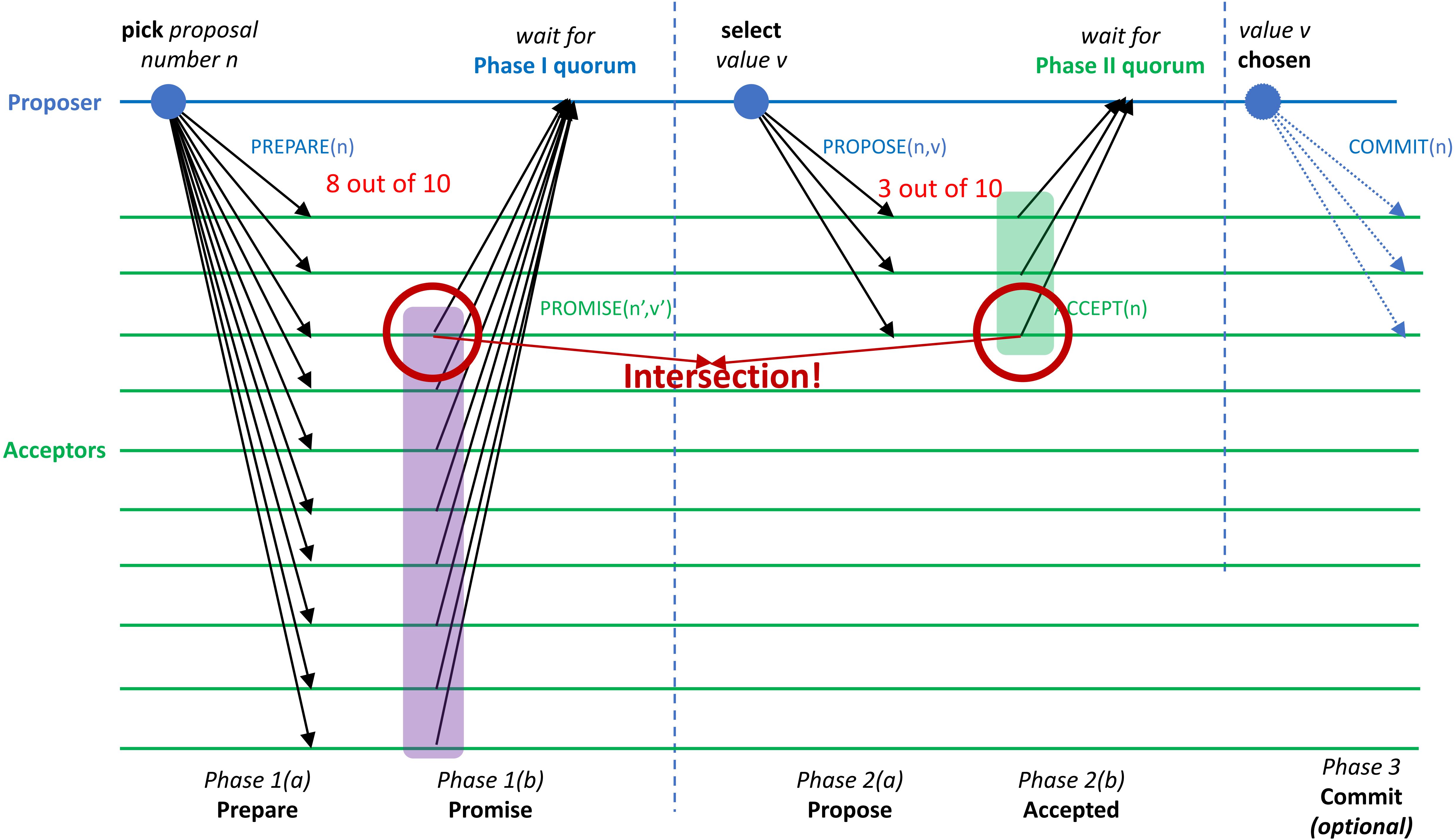
Flexible Paxos

- Single decree Paxos [1] *restriction*: Phase 1 and 2 must use a majority quorum of servers and that any two quorums must intersect

- Flexible Paxos [2]: not all quorums need to intersect. Only need that any Phase 1 quorum and any Phase 2 quorum must intersect.



pick *proposal number n*
wait for
Phase I quorum
select
value v
wait for
Phase II quorum
value v
chosen
Proposer
PREPARE(n)
8 out of 10
PROPOSE(n,v)
3 out of 10
COMMIT(n)
PROMISE(n',v')
ACCEPT(n)
Acceptors
Phase 1(a)
Prepare
Phase 1(b)
Promise
Phase 2(a)
Propose
Phase 2(b)
Accepted
Phase 3
Commit
(optional)

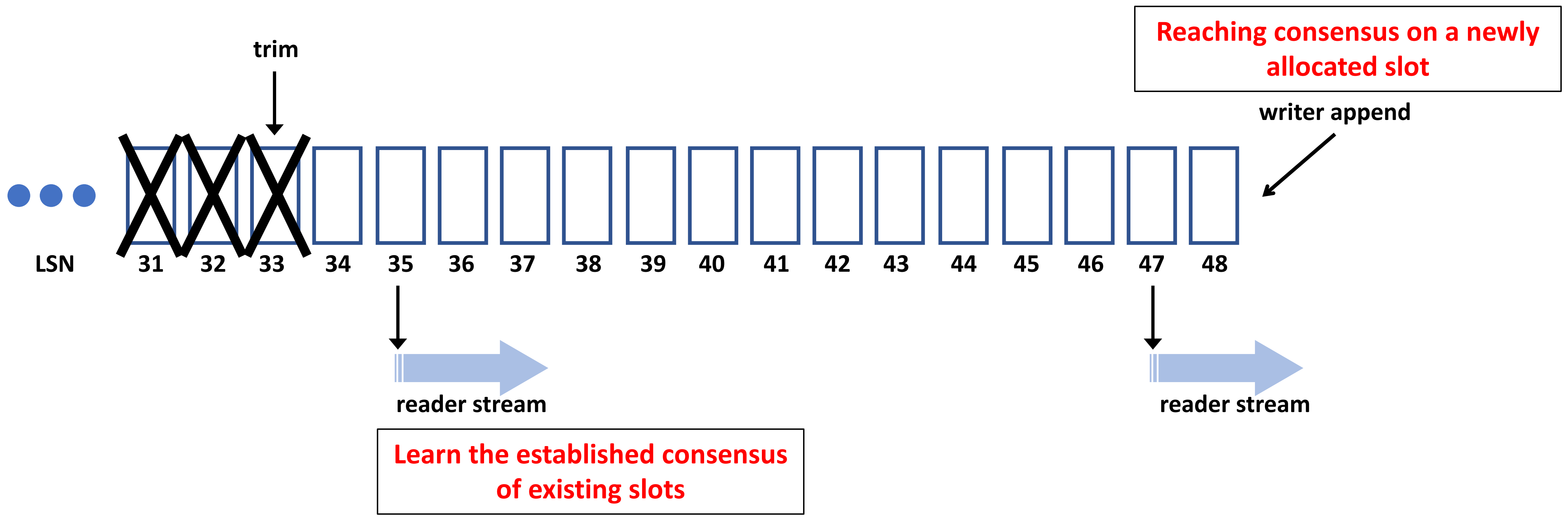

pick proposal number n
wait for
Phase I quorum
select value v
wait for
Phase II quorum
value v chosen
Proposer
PREPARE(n)
8 out of 10
PROPOSE(n,v)
3 out of 10
COMMIT(n)
PROMISE(n',v')
ACCEPT(n)
Intersection!
Acceptors
Phase 1(a)
Prepare
Phase 1(b)
Promise
Phase 2(a)
Propose
Phase 2(b)
Accepted
Phase 3
Commit
(optional)

From Single-Decree Paxos to Multi-Paxos

What is Multi-Paxos

- Scaling Paxos from single value to a **growing chain** of single-value consensus **slots**
  - Practically, we need consensus on multiple values in distributed systems
  - High throughput/Low latency – one phase 1 (leader election) + multiple phase 2 (replication)
  - directly maps to log abstraction: append-only, immutable after consensus

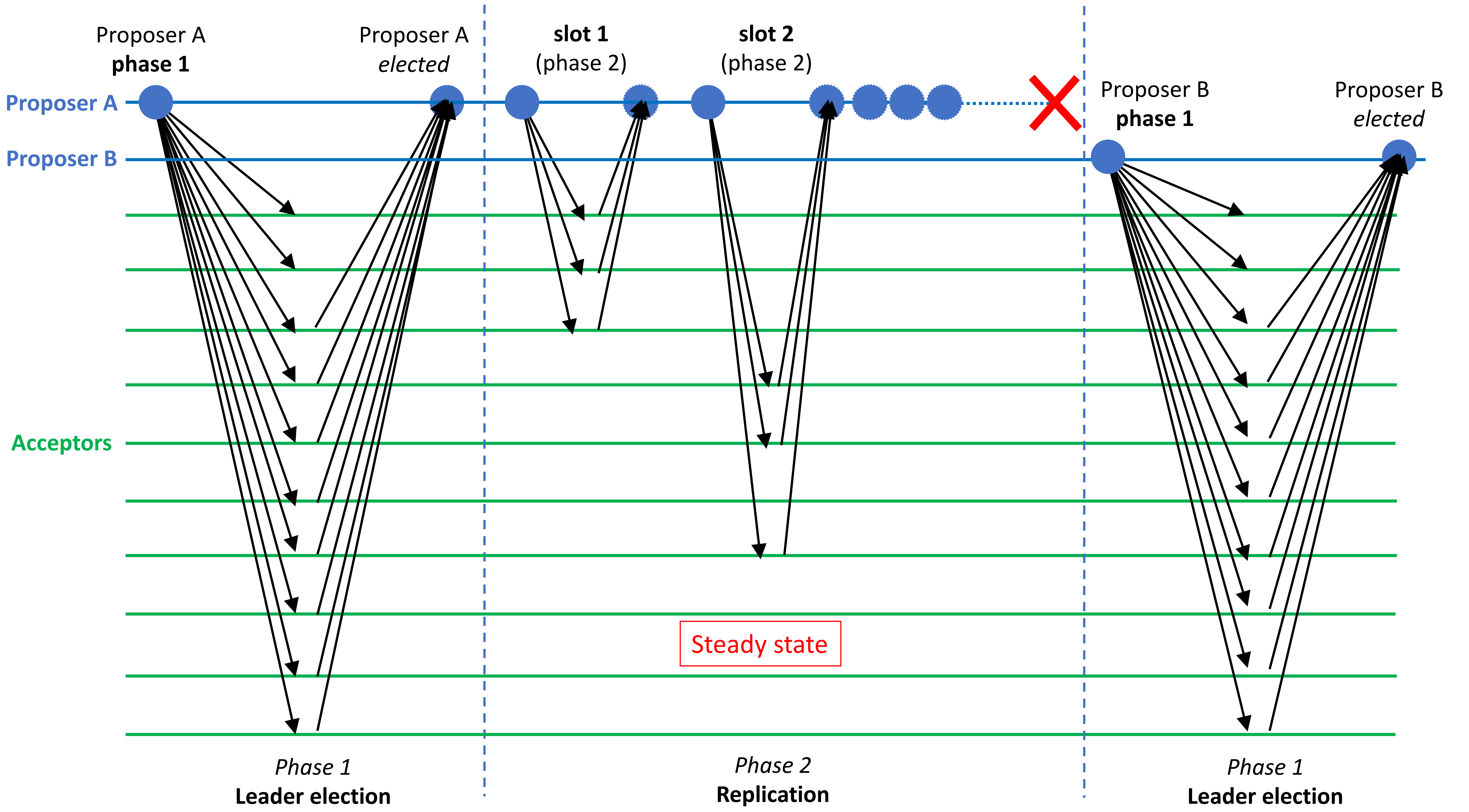
Log is **THE** abstraction for multi-Paxos



Multi-Paxos + Flexible Quorums **is a game changer**

- Highly performant steady state via **larger acceptor membership and smaller replication quorums**
  - Higher **Throughput**: pipelined Phase 2 replication with small quorums (e.g., 3 out of 20)
  - Lower **Latency**: leader picks best 3 out of 20
  - Higher **write availability**: with larger acceptor membership, leader can keep writing as long as any 3/20 acceptors are up

- Leader election – less common
  - Phase 1 – Leader election with larger quorum (e.g., 18 out of 20) only during leader failover



Proposer A
phase 1
Proposer A
elected
slot 1
(phase 2)
slot 2
(phase 2)
Proposer B
phase 1
Proposer B
elected
Proposer A
Proposer B
Acceptors
Steady state
Phase 1
Leader election
Phase 2
Replication
Phase 1
Leader election

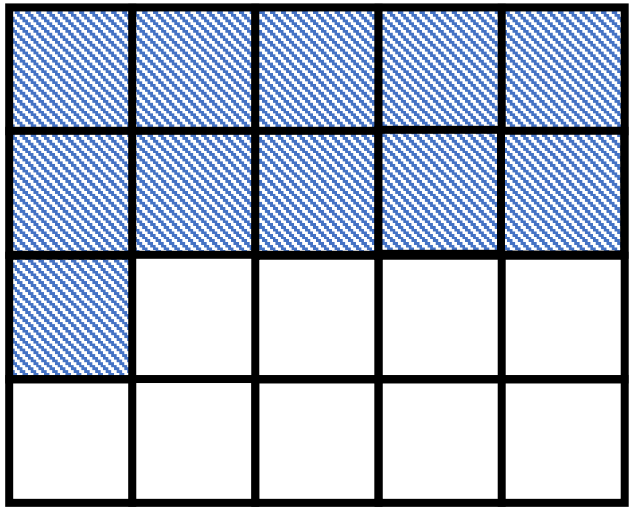
Question: with a larger Phase 1 quorum (e.g., 18 out of 20) , is it more difficult (i.e., less available) to elect a leader?

Failure domain aware Placement

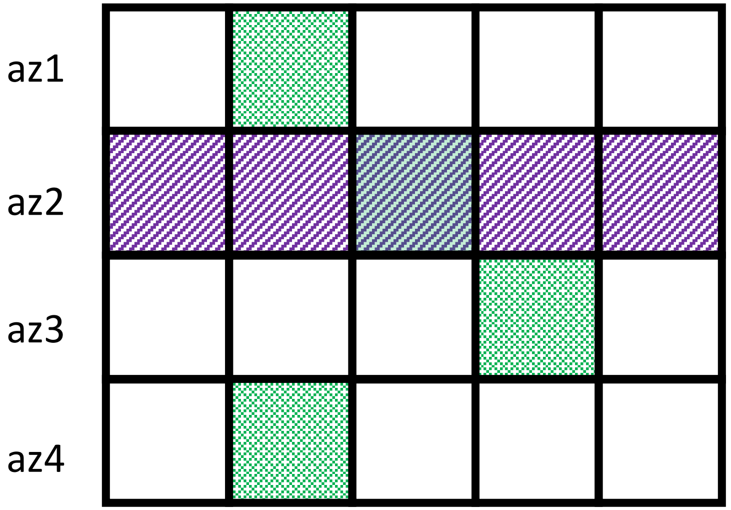
- Goal: Improving **availability** and fault tolerance for Phase 1 (leader election)

- Solution: Failure domain aware placement
  - Reducing size requirement of Phase 1 quorum by enforcing **topology constrains** on Phase 2 quorums during replication
  - result: Phase 1 quorum require much smaller number of acceptors during correlated failures

Flexible Quorums example: Grid Quorums



(a) Basic Paxos:
Phase 1 and 2 quorum: simple majority



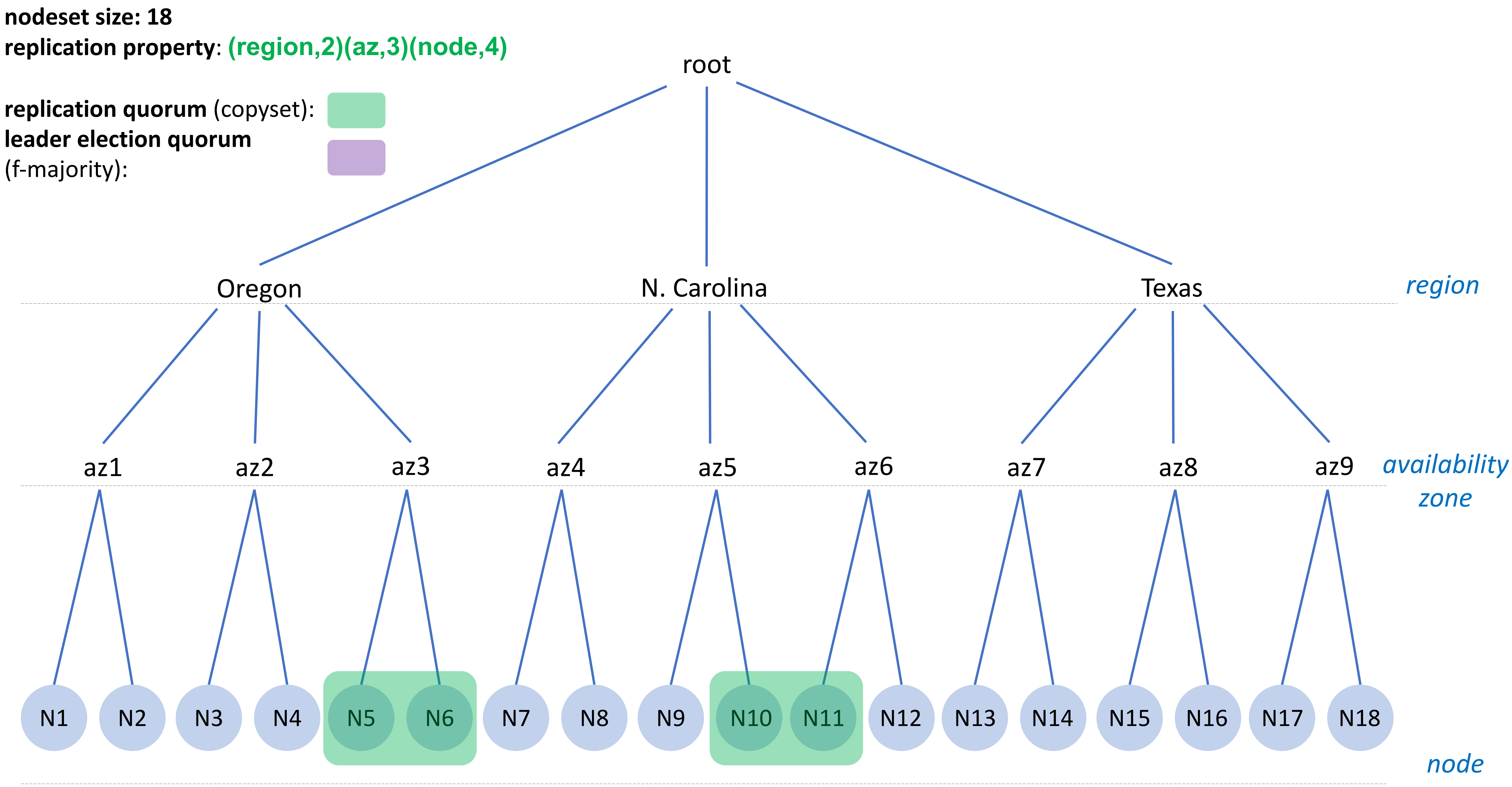
(b) Flexible Paxos:
Phase 1 quorum: one full Availability Zone (AZ)
Phase 2 quorum: a node in each AZ

Neither Phase 1 nor Phase 2 quorum need simple majority!

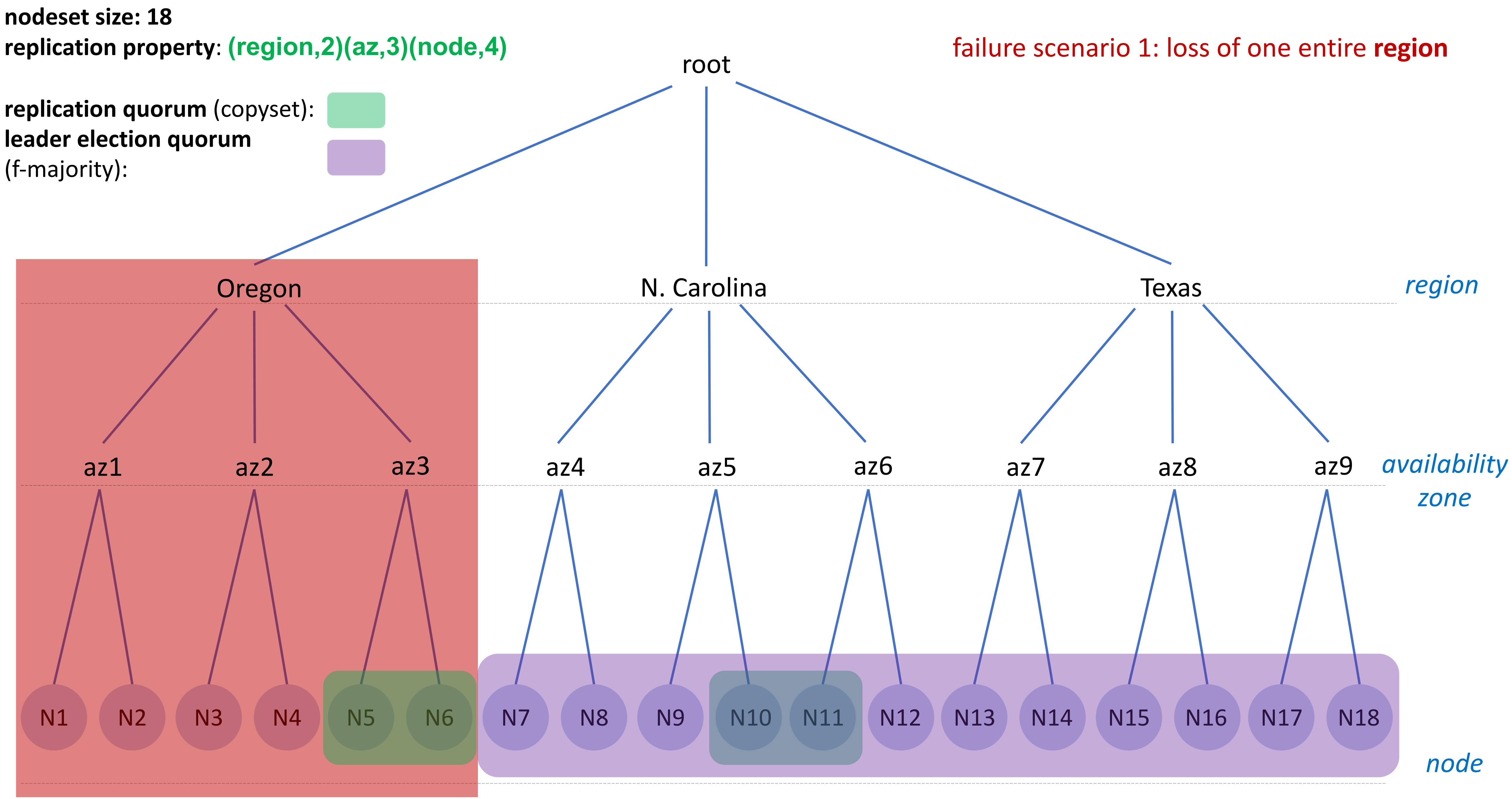
→ Significantly reduced minimal number of acceptors required: floor(M*N/2)+1 -> M+N-1

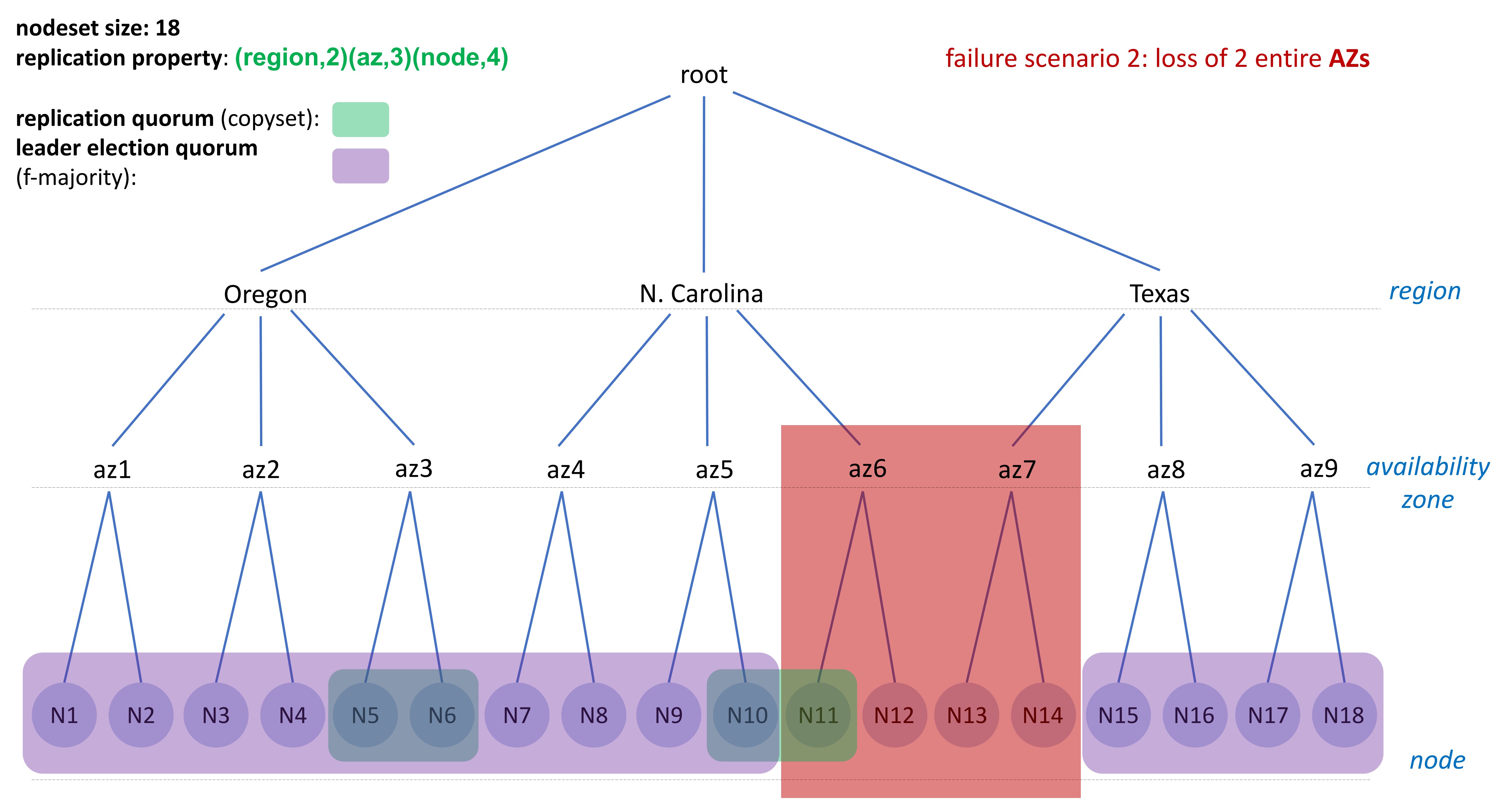
→ Higher **availability** and better **latency**.

→ Better data availability and durability in correlated failures



nodeset size: 18
replication property: (region,2)(az,3)(node,4)
replication quorum (copyset):
leader election quorum (f-majority):
root
Oregon
N. Carolina
Texas
region
az1
az2
az3
az4
az5
az6
az7
az8
az9
availability zone
N1
N2
N3
N4
N5
N6
N7
N8
N9
N10
N11
N12
N13
N14
N15
N16
N17
N18
node

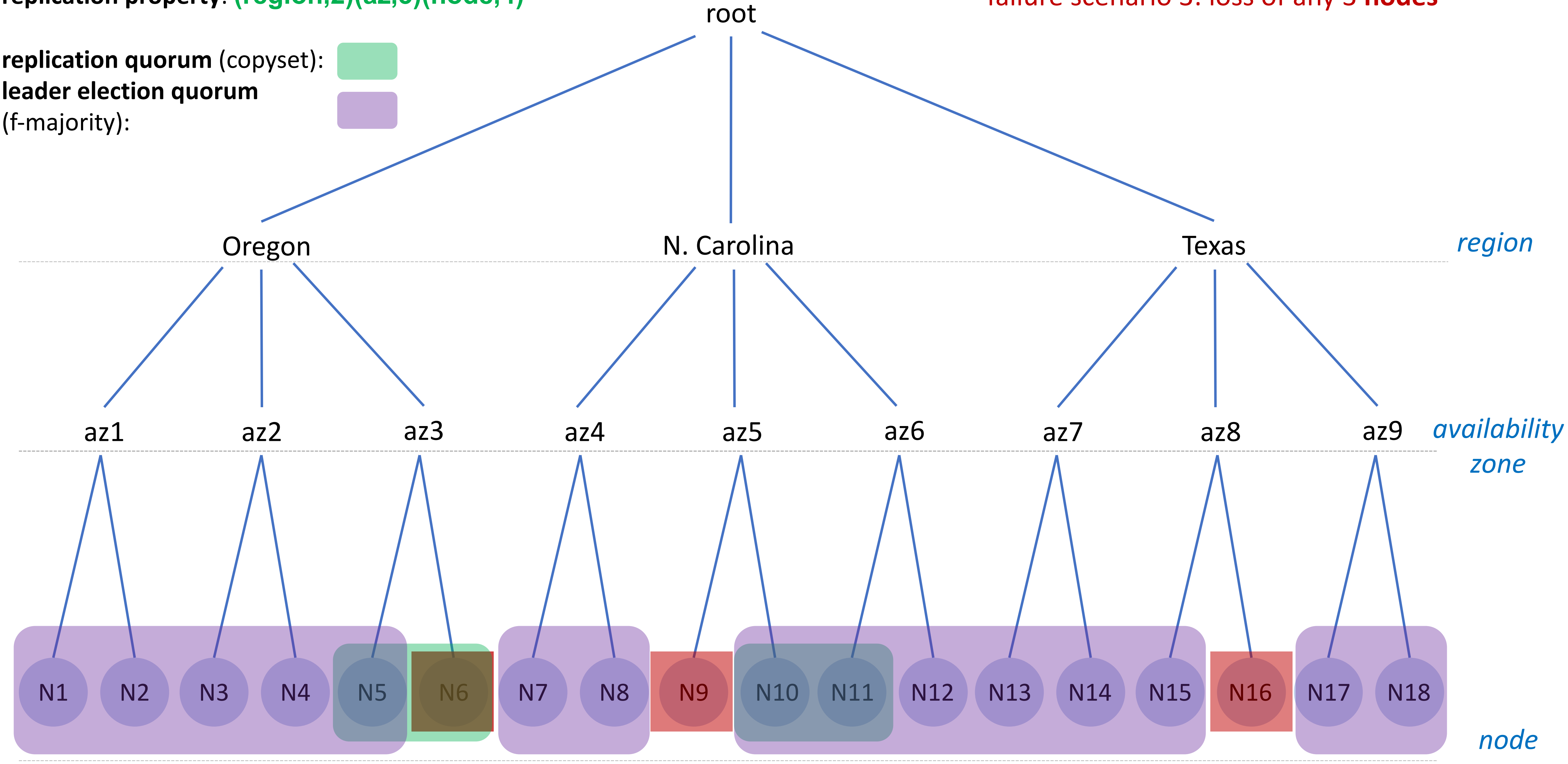

nodeset size: 18
replication property: (region,2)(az,3)(node,4)
replication quorum (copyset):
leader election quorum (f-majority):
failure scenario 1: loss of one entire region
root
Oregon
N. Carolina
Texas
region
az1
az2
az3
az4
az5
az6
az7
az8
az9
availability zone
N1
N2
N3
N4
N5
N6
N7
N8
N9
N10
N11
N12
N13
N14
N15
N16
N17
N18
node


nodeset size: 18
replication property: (region,2)(az,3)(node,4)
failure scenario 2: loss of 2 entire AZs
replication quorum (copyset):
leader election quorum
(f-majority):
root
Oregon
N. Carolina
Texas
region
az1
az2
az3
az4
az5
az6
az7
az8
az9
availability
zone
N1
N2
N3
N4
N5
N6
N7
N8
N9
N10
N11
N12
N13
N14
N15
N16
N17
N18
node

nodeset size: 18
replication property: **(region,2)(az,3)(node,4)**

failure scenario 3: loss of any 3 nodes

replication quorum (copyset):
leader election quorum (f-majority):

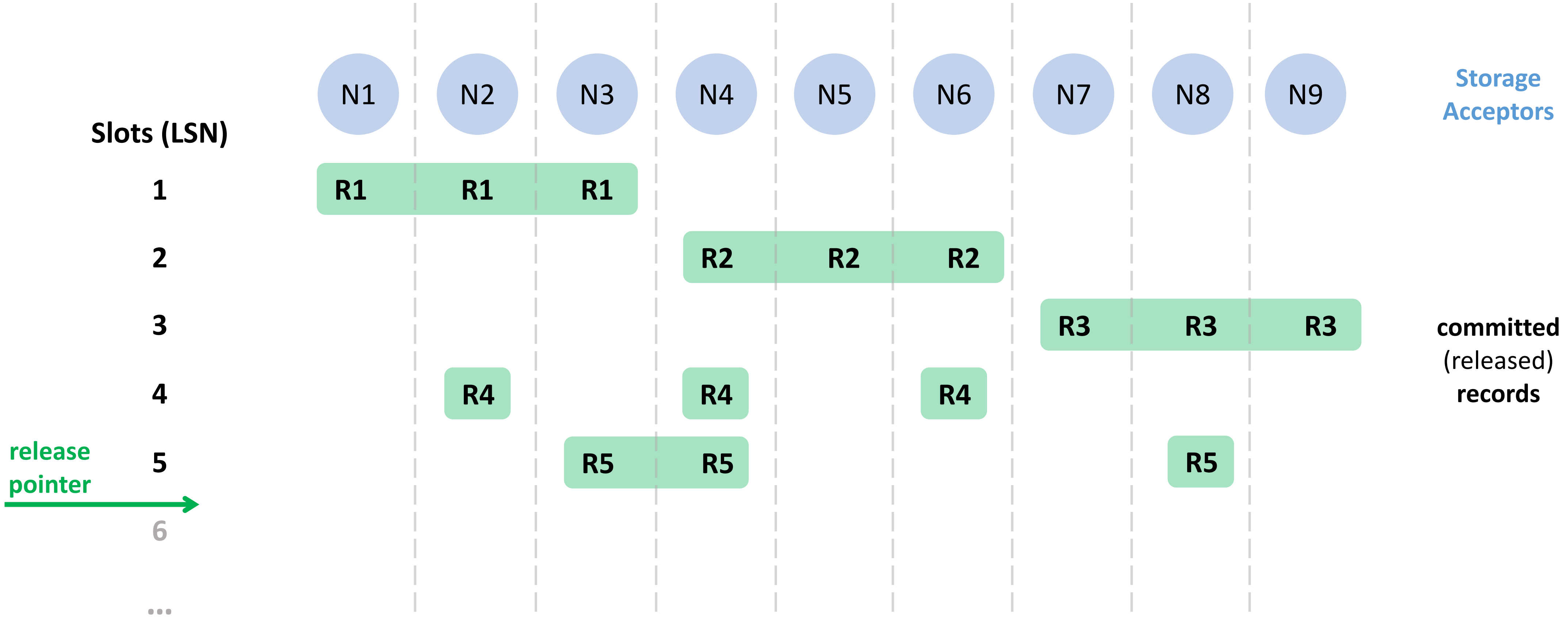


Storing and learning consensus results

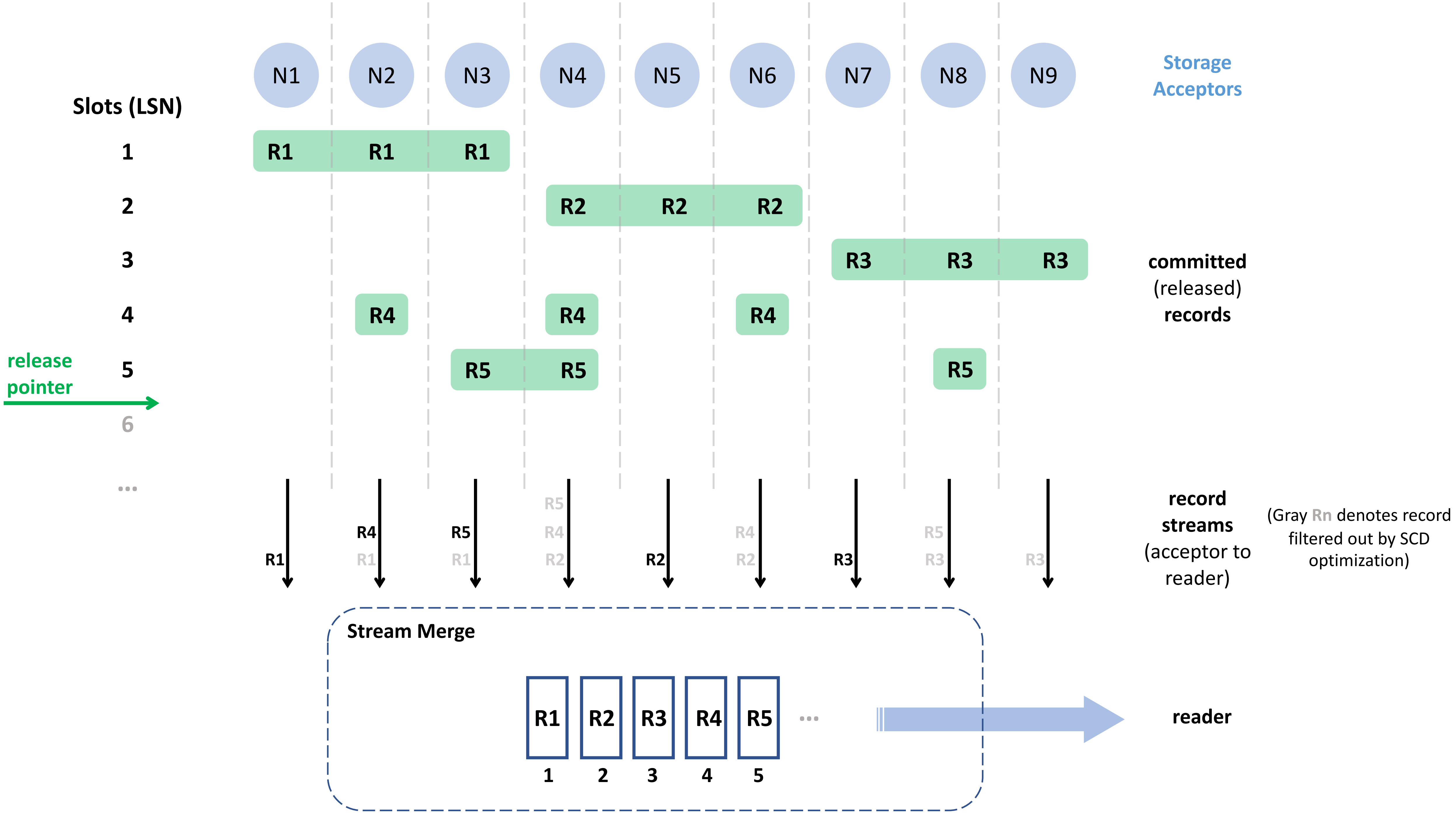
- Consensus log records are stored among acceptors in a **data striping** fashion
  - storage acceptors do **not** store the full copy of the log
  - flexible Paxos enables disjoint small replication quorums over large acceptor membership - perfect for striping

- Advantages:
  - Only ***f+1*** record copies are needed for tolerating ***f*** acceptor failures
  - Log throughput and capacity **not** bounded by a single storage acceptor

- Learning the result of consensus: reading the log via streaming
  - Acceptors stream their local copy of **committed** records
  - Client reader **merges** all acceptor record streams using slot (LSN) order
  - **Single Copy Delivery** (SCD) optimization achieves 1X read amplification



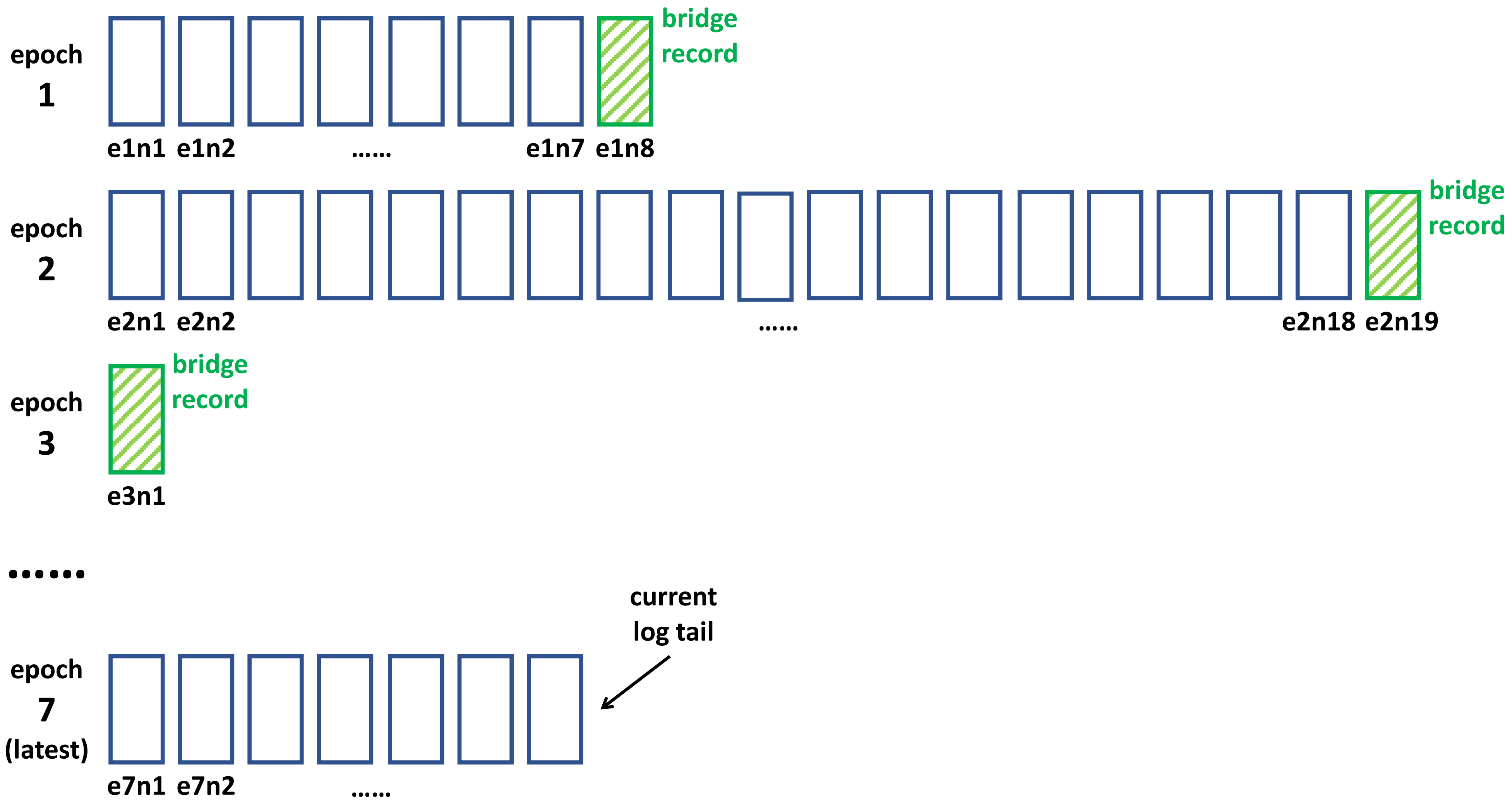
Storage Acceptors
N1
N2
N3
N4
N5
N6
N7
N8
N9
Slots (LSN)
1
R1 R1 R1
2
R2 R2 R2
3
R3 R3 R3
committed (released) records
4
R4 R4 R4
5
R5 R5 R5
release pointer
6
...


Storage Acceptors
N1
N2
N3
N4
N5
N6
N7
N8
N9
Slots (LSN)
1
2
3
4
5
6
...
release pointer
R1 R1 R1
R2 R2 R2
R3 R3 R3
R4 R4 R4
R5 R5 R5
committed (released) records
record streams (acceptor to reader)
(Gray Rn denotes record filtered out by SCD optimization)
Stream Merge
R1 R2 R3 R4 R5 ...
1 2 3 4 5
reader

Log Segments and configuration management

- A Log in LogDevice -> A sequence of **log segments** indexed by monotonically increasing **epoch**
  - each segment has its **fixed** configuration: idea inspired by *Stoppable Paxos* [5]
- Reconfiguration can happen **out-of-band** of replication via an auxiliary metadata store
  - epoch store: stores log segment configuration. back by Zeus (Zookeeper).
  - auxiliary metadata store inspired by *Vertical Paxos* [3]
- Starting a new log segment when:
  - Leader (sequencer) fail-over
  - reconfiguring replication property and storage acceptor membership
- Similar design also adopted by Delos [4]

Log Segments



configuration of a log epoch segment

epoch: 1 | **SEQ: N0** | [(region, 2), (node, 3)] | { N1, N2, N3, N4, N5 }

epoch: 1	**SEQ: N0**	[(region, 2), (node, 3)]	{ N1, N2, N3, N4, N5 }
epoch	**sequencer (leader)**	**replication property**	**storage node set (acceptors)**

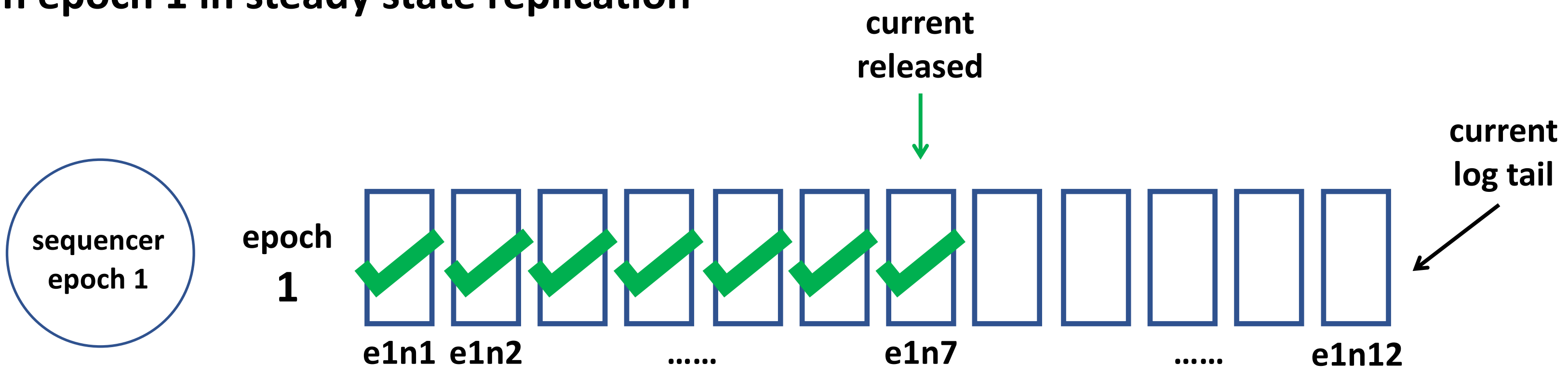
epoch transition: Sealing and Bridge record

- Starting a new log segment requires first **“Sealing”** the previous log segment.
  - A procedure similar to executing Phase 1 Paxos on a **leader election quorum** of the **previous segment**
  - Once sealing is done, no append request can be successfully ACKed to the sealed log epoch segment

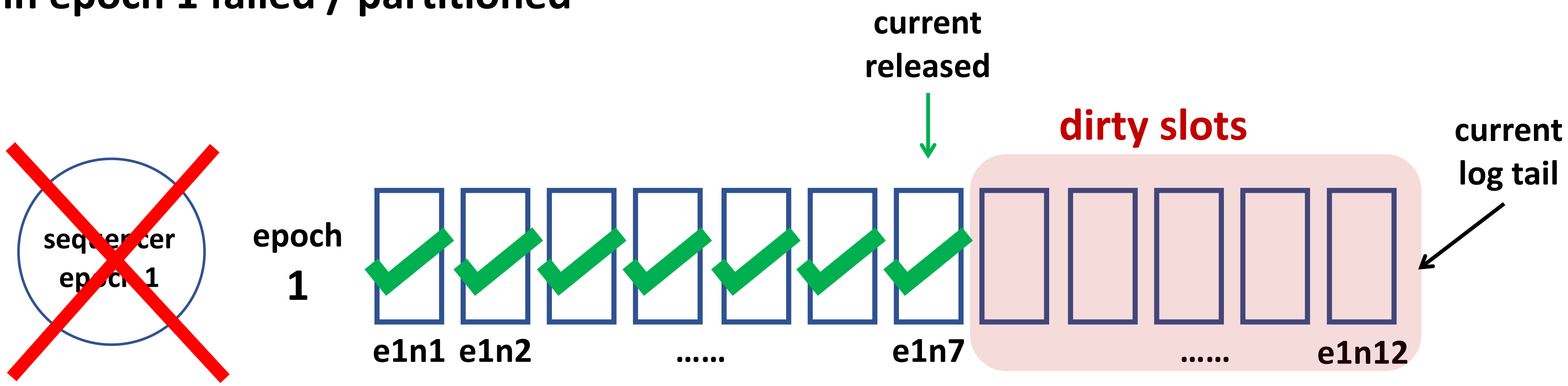
- After Sealing, an **epoch recovery** procedure is performed to:
  - learn the last appended record slot in the sealed epoch segment
  - place a **bridge record** immediately after the last record, marking the end of the log segment
  - once bridge record is place the log epoch segment becomes **immutable** (until trimmed)

Animation: Leader (sequencer) failure scenario

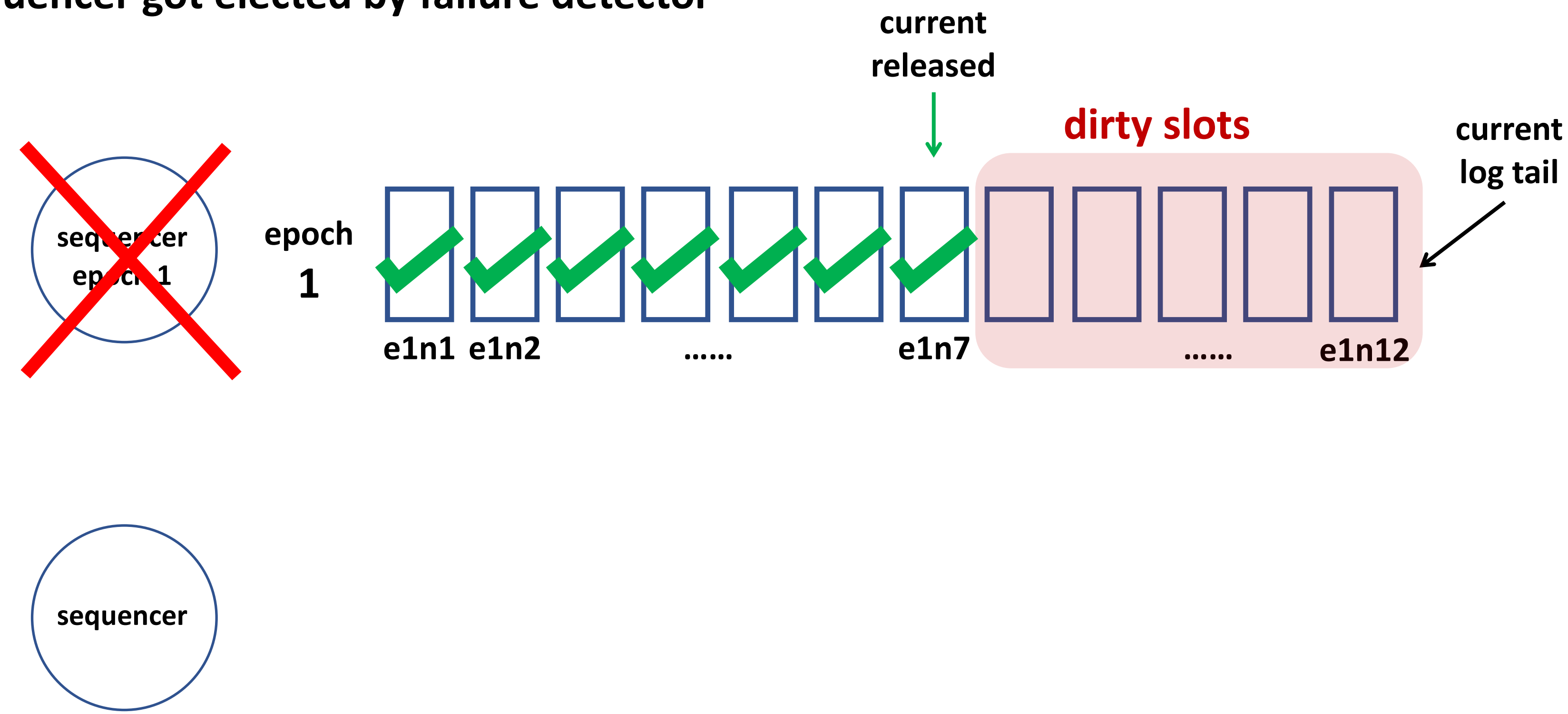
0. Sequencer in epoch 1 in steady state replication



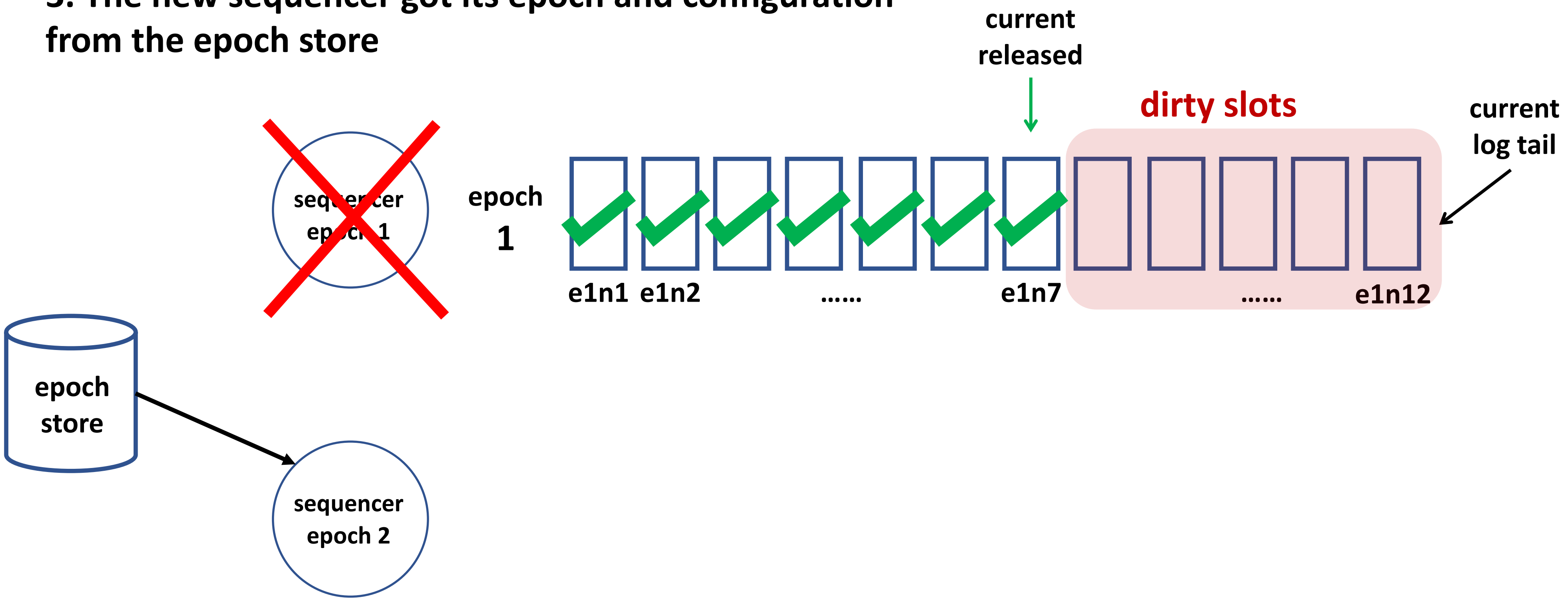
1. Sequencer in epoch 1 failed / partitioned



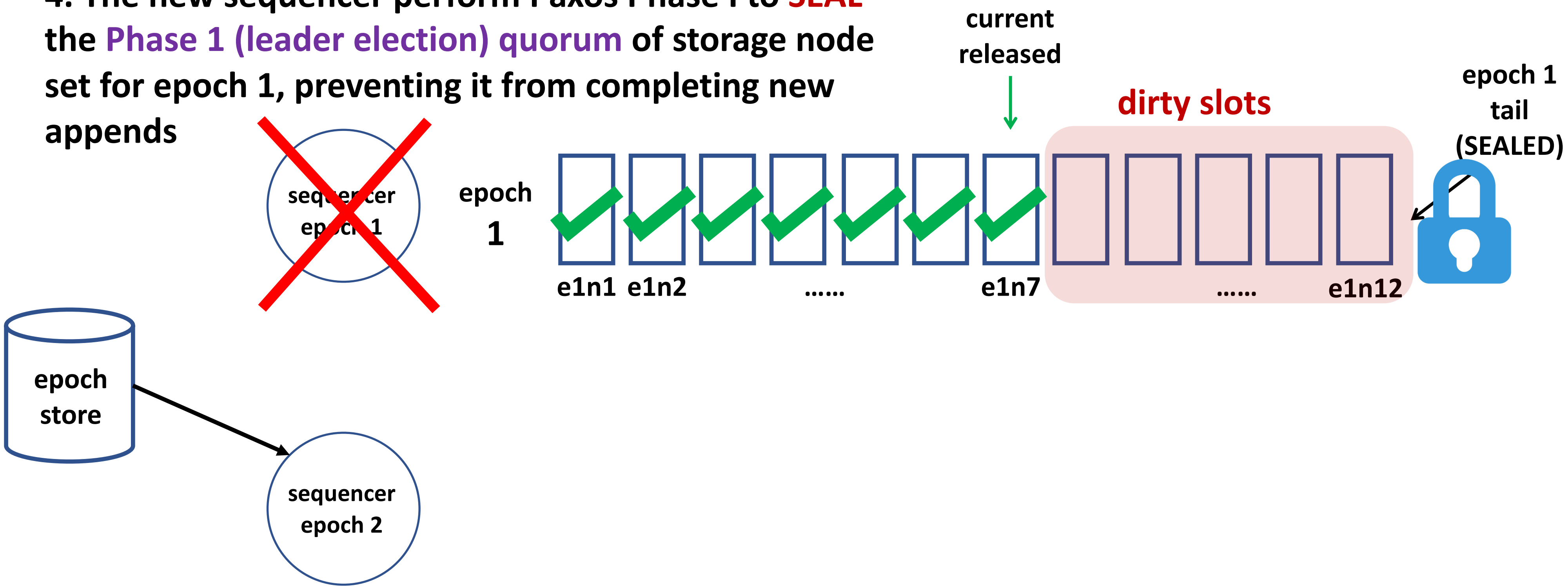
2. A new sequencer got elected by failure detector



3. The new sequencer got its epoch and configuration from the epoch store



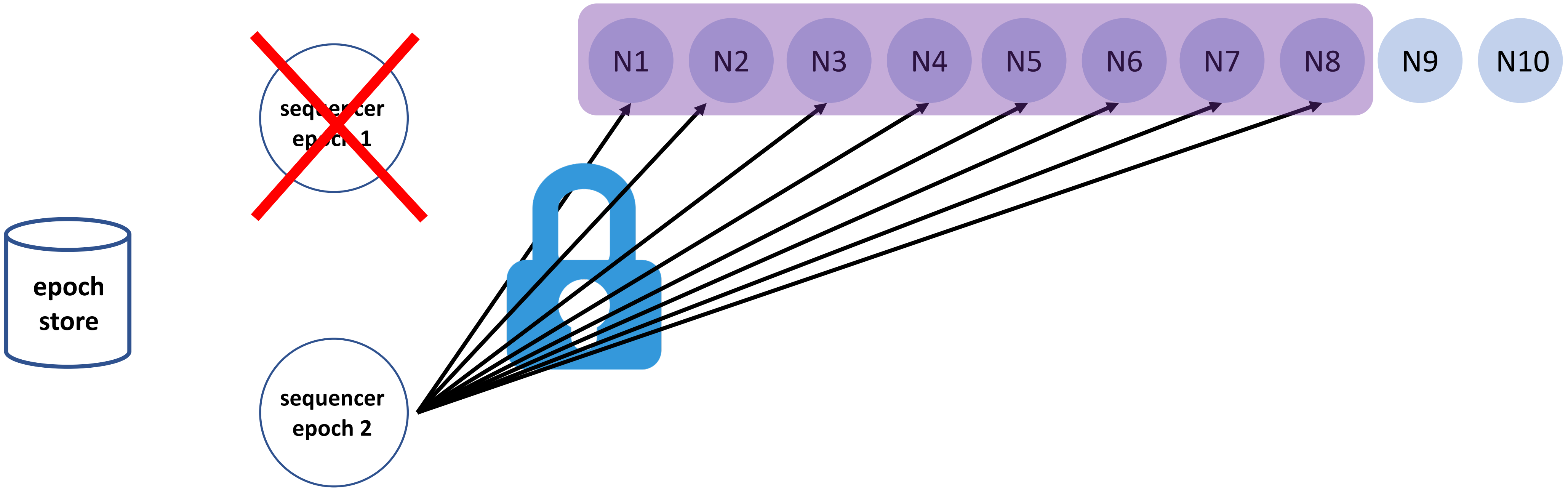
4. The new sequencer perform Paxos Phase I to SEAL the Phase 1 (leader election) quorum of storage node set for epoch 1, preventing it from completing new appends



Seal Zoom in:

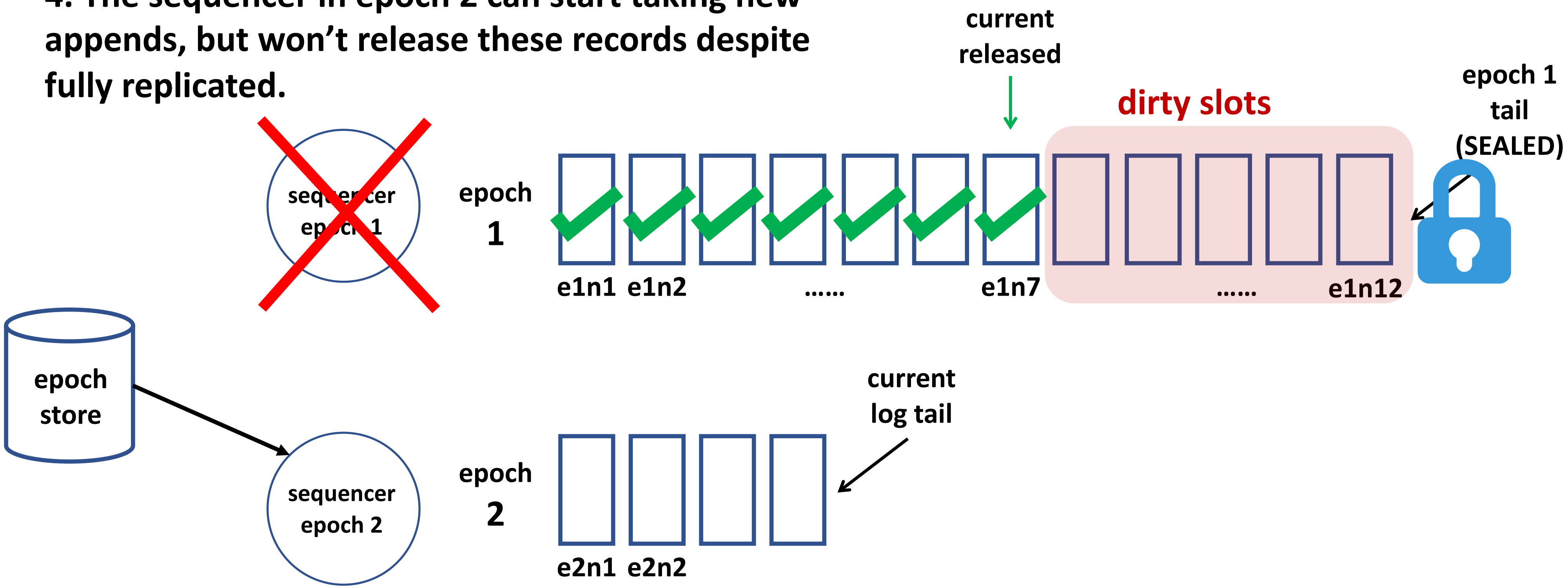
Sequencer in epoch 2 Seals phase 1 quorum of the configuration of the previous epoch segment (epoch). 2 (epoch) is used as the proposal/ballot number.

epoch: 1 | **SEQ: N0** | [(region, 2), (node, 3)] | { N1, N2, N3, N4, N5, N6, N7, N8, N9, N10 }

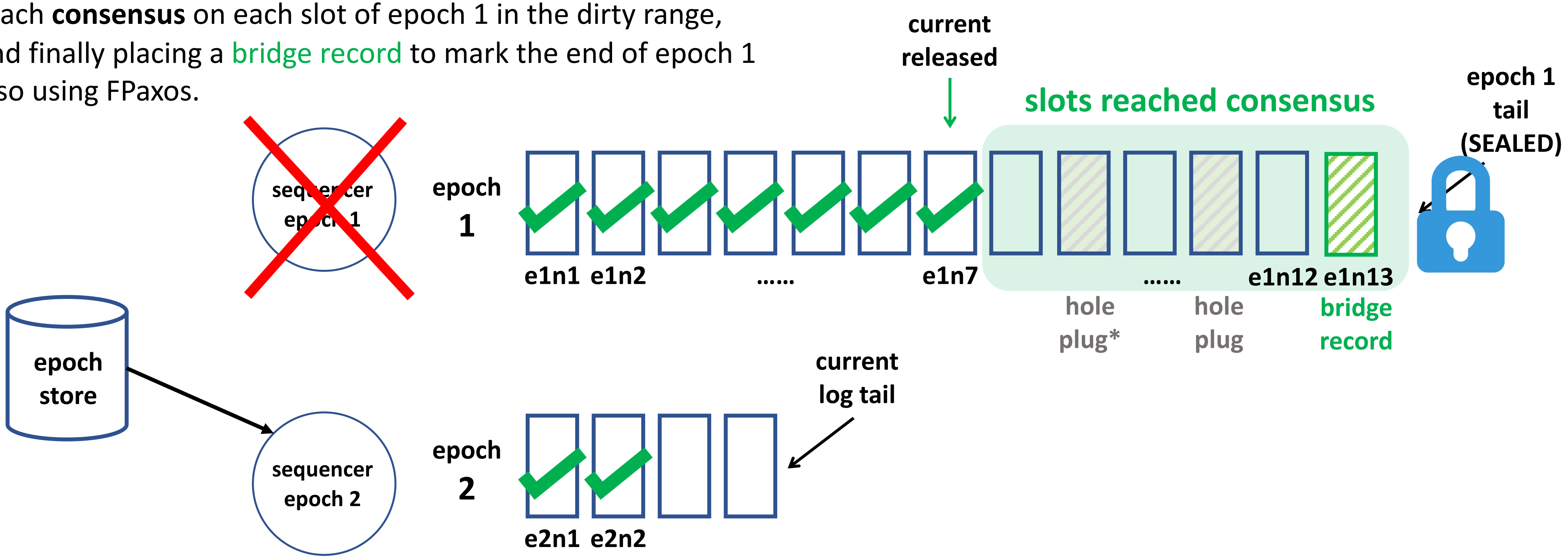


epoch: 2 | **SEQ: N20** | [(region, 2), (node, 3)] | { N4, N5, N6, N7, N8, N9, N10, N11 }

4. The sequencer in epoch 2 can start taking new appends, but won't release these records despite fully replicated.

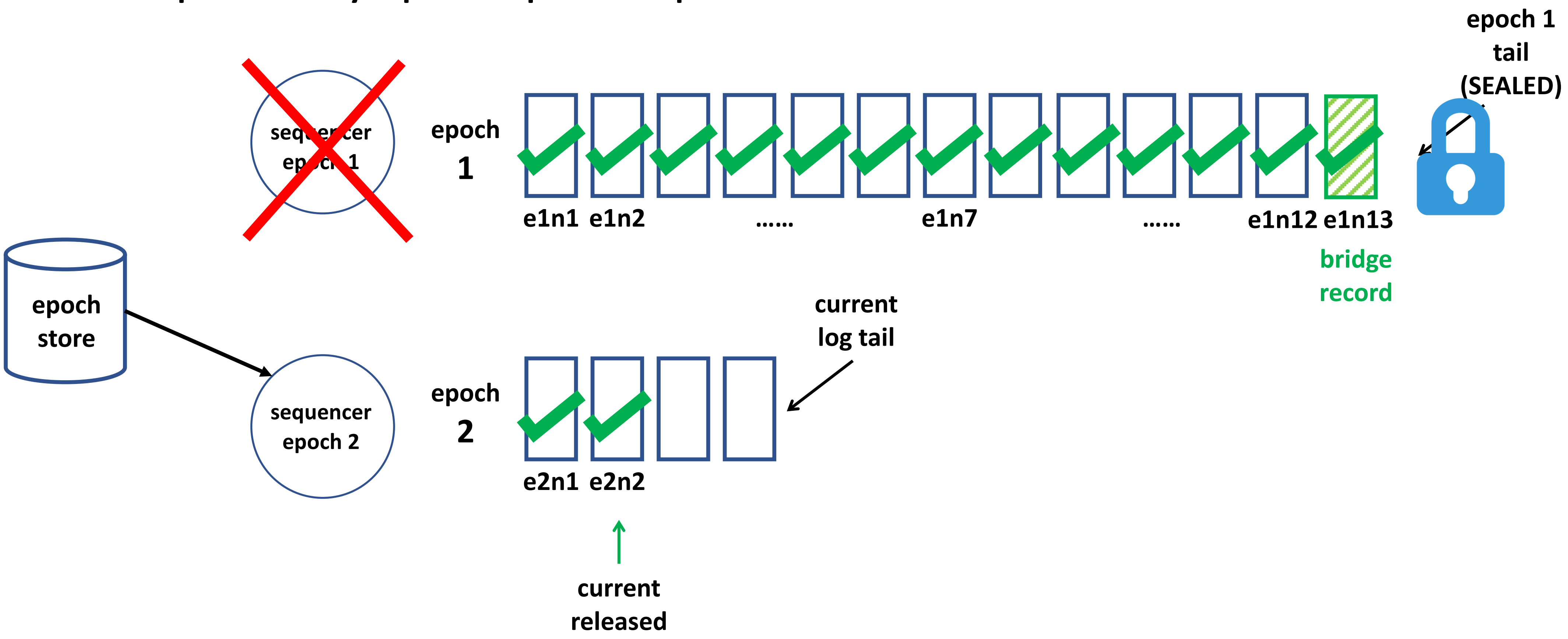


5. At the same time, sequencer in epoch 2, with potential other successors, keep running FPaxos (Phase I and II) to reach **consensus** on each slot of epoch 1 in the dirty range, and finally placing a bridge record to mark the end of epoch 1 also using FPaxos.



* hole plug is inserted for the LSN slots that are were NOT ACKed originally, indicating a benign (non-dataloss) gap in the LSN sequence.

6. The sequencer in epoch 2 can finally release all records up to the fully replicated prefix of epoch 2.



Zero-move, out-of-band reconfiguration

- **No data movement** with reconfiguration
  - start a new log segment only requires a transaction in metadata store

- Out-of-band reconfiguration benefits:
  - Allowing different requirements and design choices in **data plane** vs. **metadata plane**
    - trade-off on durability, availability, throughput, …
  - Higher **availability** in reconfiguration
    - Scenario: steady state log replication is stuck (e.g., quorum loss)
      - In-band: cannot reconfigure, require manual intervention!
      - Out-of-band: reconfiguration by starting a new log segment with a new health acceptor membership

- Low reconfiguration latency
  - Reconfiguration latency: TX in metadata store + Sealing the previous segment.
  - No joint consensus. No intermediary transition. Not blocked by data replication.

Highlights

- Superior steady state replication performance
- Smart placement for IaaS compliant failure modelling
- Only requires ***f+1*** copies for tolerating ***f*** failures
  - 40% less space compared with raft when ***f*** = 2
- Low latency, Zero-move reconfiguration
- Log capacity and throughput **not** bounded by a single node
- High write availability from out-of-band reconfiguration

Takeaways

- Log is THE abstraction for modeling multi-paxos
- LogDevice is a managed service with strong consistency of multi-paxos and highly performant and efficient with flexible quorums
  - Designed to be a reliable, scalable and flexible service

LogDevice: Paxos at Facebook Scale

- LogDevice powering Scribe use case: https://engineering.fb.com/data-infrastructure/scribe/
  - "the total size of these logs is several petabytes every hour."
  - 2.5 TB/s writes; 7 TB/s reads globally

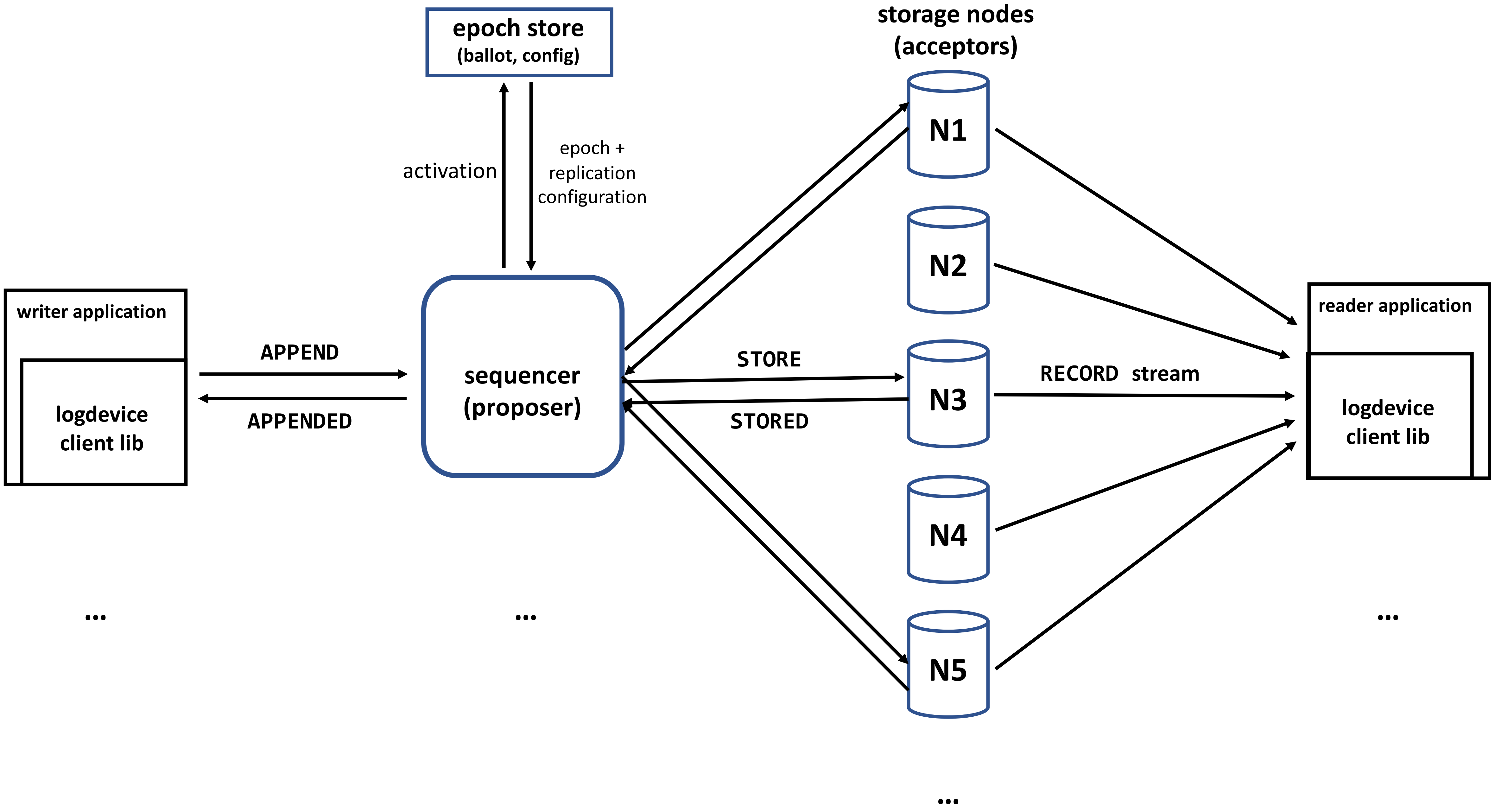
References

- [1] **Paxos Made Simple.** Leslie Lamport. 2001
- [2] **Flexible Paxos: Quorum intersection revisited.** Heidi Howard, Dahlia Malkhi, Alexander Spiegelman. 2016
- [3] **Vertical Paxos and Primary-Backup Replication.** Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. 2009
- [4] **Delos: Simple, flexible control plane storage.** Mahesh Balakrishnan and Jason Flinn. 2019
- [5] **Stoppable Paxos.** Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. 2008
- [6] **Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters**. Sridharan Ranganathan, et al. 2001

Appendix

Leader election and APPEND routing

- Leader election is a plug-in in Multi-Paxos context
- Gossip-based failure detector [6]
  - cluster nodes exchange gossips periodically (e.g., every 100ms)
  - nodes maintain local cluster state; clients poll cluster state from server;
- placement and routing: weighted consistent hashing
  - input: logid, sequencer configuration (map of *node* -> *weights*), cluster state
  - output: sequencer node id for the log
- "Soft consensus"
  - best effort for local views to converge quickly (i.e., 1-3 seconds)
  - failing to achieve that won't affect correctness, but may affect liveness (i.e., availability / latency)
    - sequencer ping-pong issue



epoch store
(ballot, config)
activation
epoch +
replication
configuration
storage nodes
(acceptors)
writer application
logdevice
client lib
APPEND
APPENDED
sequencer
(proposer)
STORE
STORED
N1
N2
N3
N4
N5
RECORD stream
reader application
logdevice
client lib
...
...
...
...