Paxos Made Simple

CSE 550
October 27, 2021

Replicated State Machine

- Avoid single point of failure
- Need to agree on what instruction to execute (and in what order)
  - Example: adding 2 and multiplying by 1.1 is different to multiplying by 1.1 and adding 2.
- Assume fail-stop model
  - Nodes can fail and restart, so there has to be some stable storage to remember information.
  - When nodes fail, they simply stop.
    - Don’t overwrite their storage, don’t send adversarial messages, etc.

Terminology

- Proposers
  - Try to get a value chosen by sending proposals
  - Usually, proposing client's requested operations
- Acceptors
  - Respond to proposals (and related messages)
- Learners
  - Determine when a value is chosen
    - in state machine replication example, learn op and execute
- Roles are separate in the paper
- Usually, nodes play all 3 roles

Discussion (shorturl.at/sxEU9)

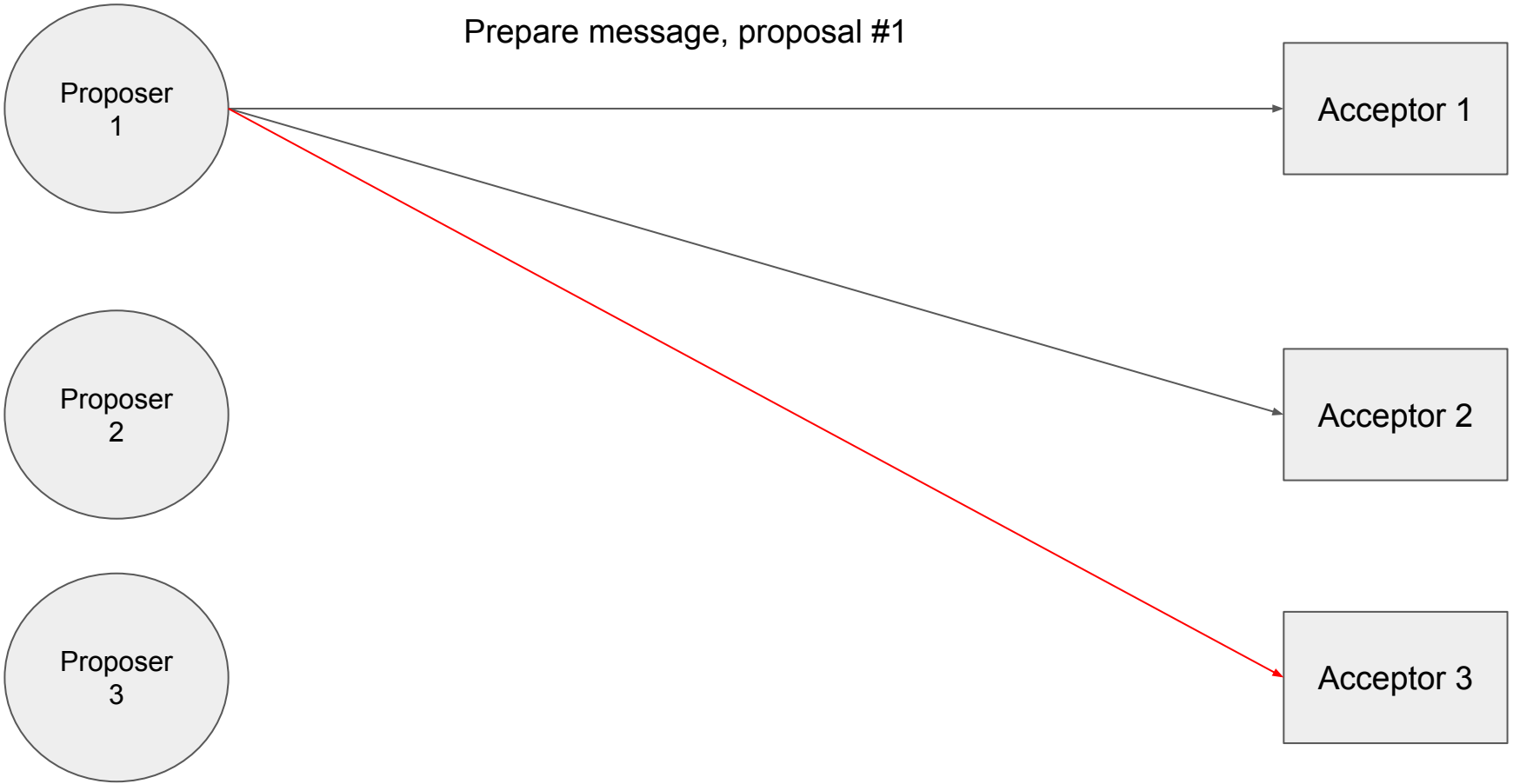
- Should all 3 roles be on the same node? What are the benefits/downsides?
- Should the fail-stop assumption be used in practice?

Terminology: Proposals

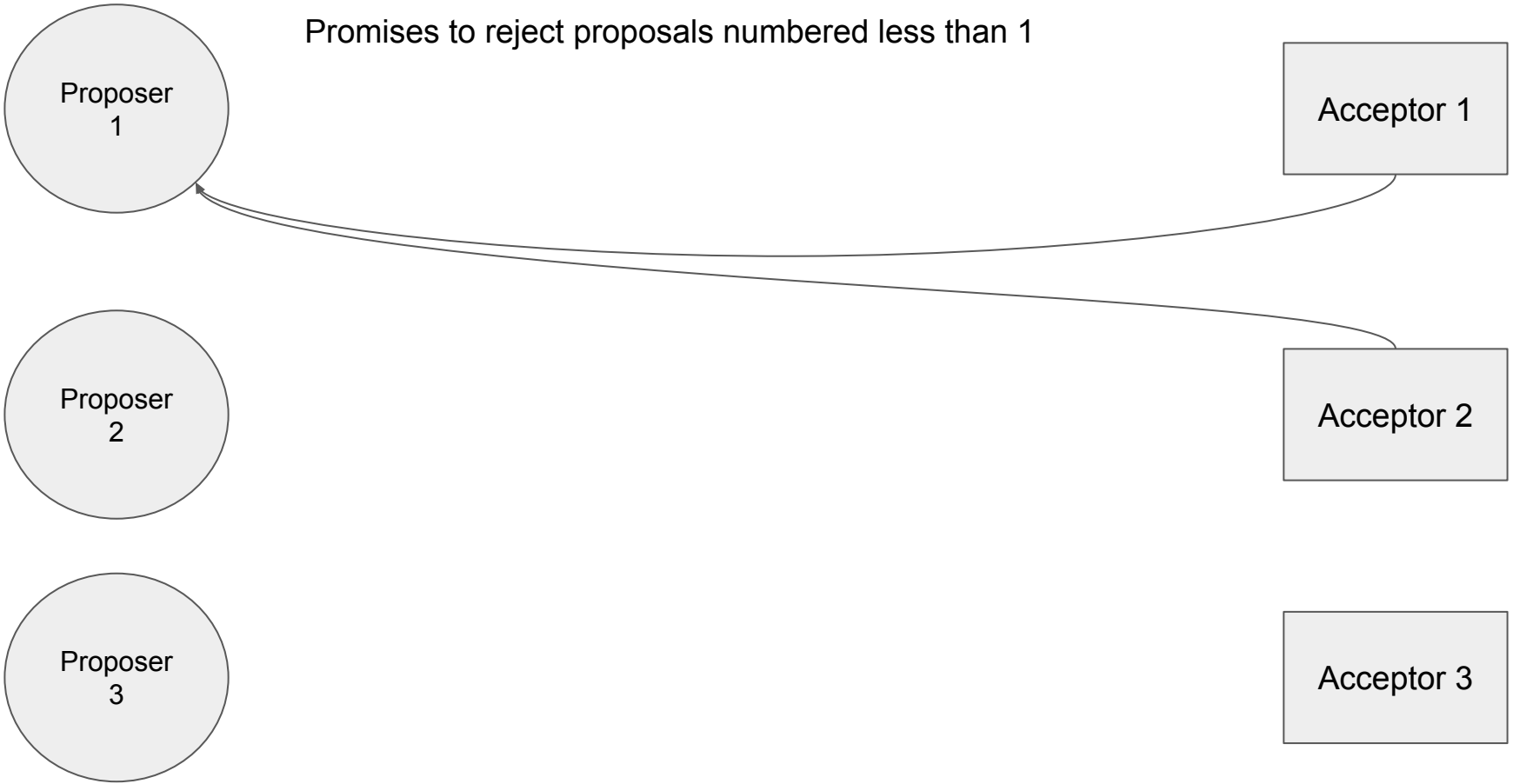
- Proposals have a value (e.g., operation) and a proposal number
- Accepted vs Chosen
  - Accepted: local to a node
  - Chosen: a majority of the acceptors have accepted the proposal
    - Depends on proposal number: maybe (v, 1), (v, 2), (v, 3) accepted, none chosen.
    - Eventually, v will be chosen, but it’s not yet chosen.

Safety

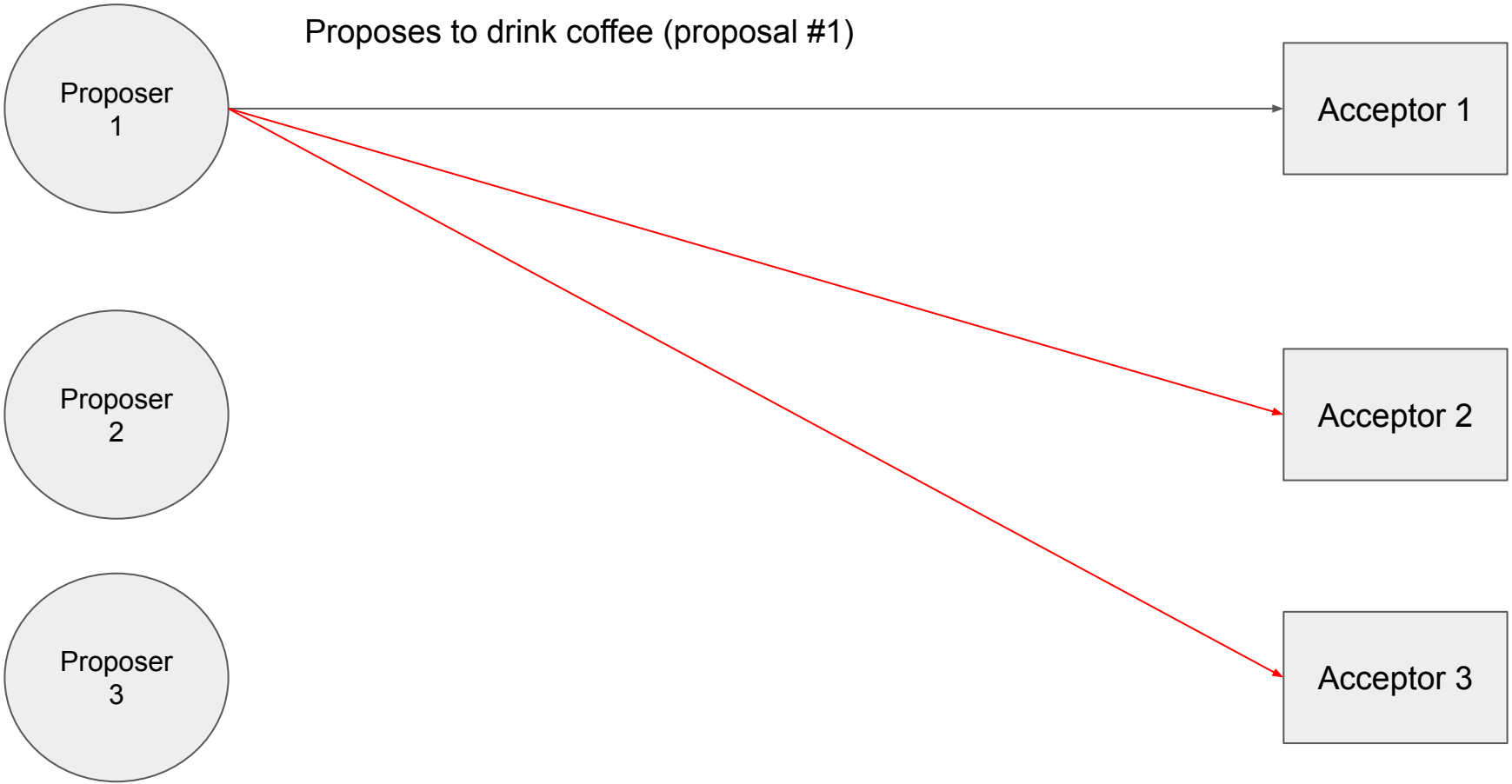
- Paxos guarantees the following (assuming fewer than half of the nodes simultaneously fail).
  - If a value is chosen, then it has been proposed
  - If a value is chosen, no other value is chosen
  - If a learner learns that a value is chosen, then the value is chosen.
- It does NOT guarantee that a value is chosen (liveness)
  - Example in the paper: two proposers preventing acceptors from accepting each other



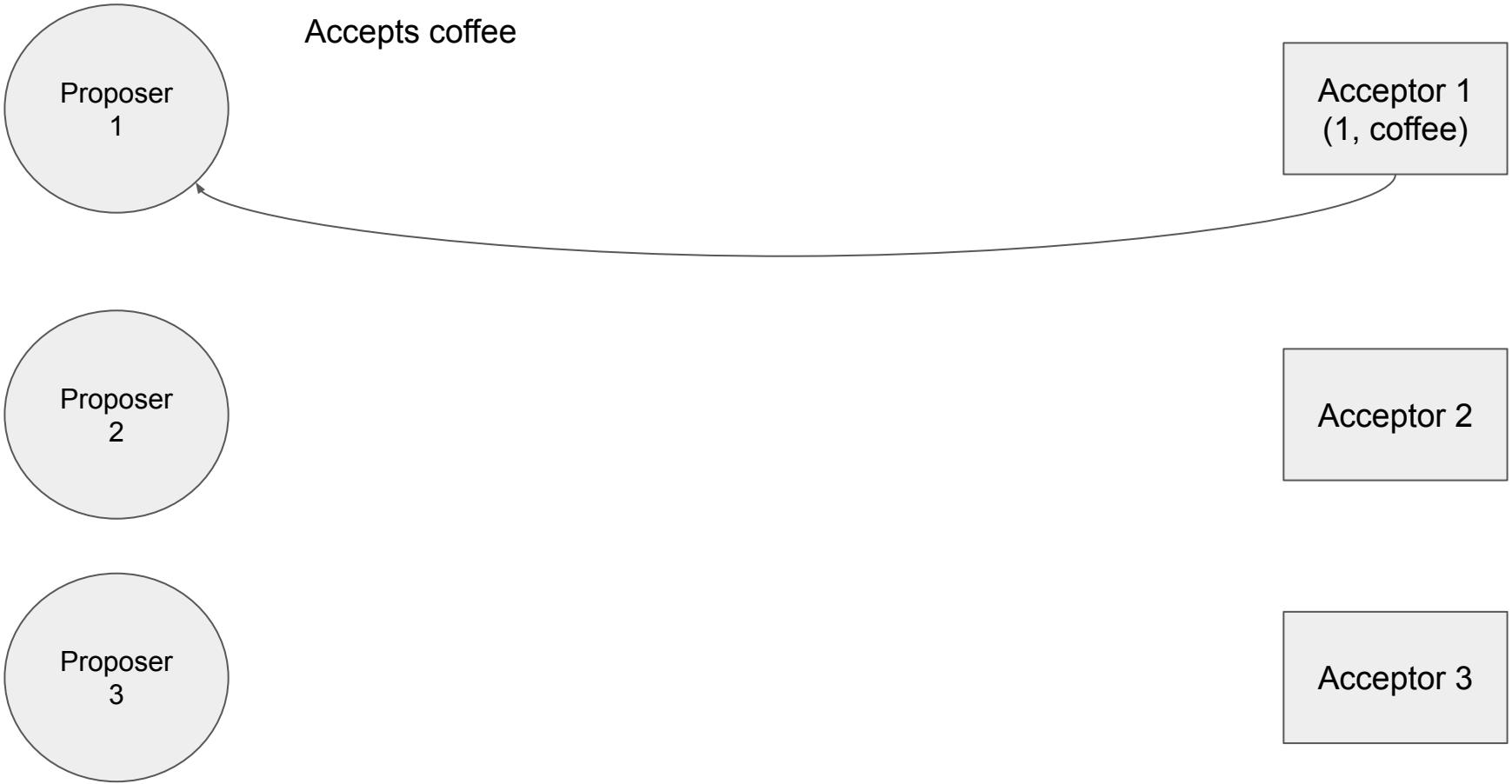
Paxos to decide what drink to have with breakfast



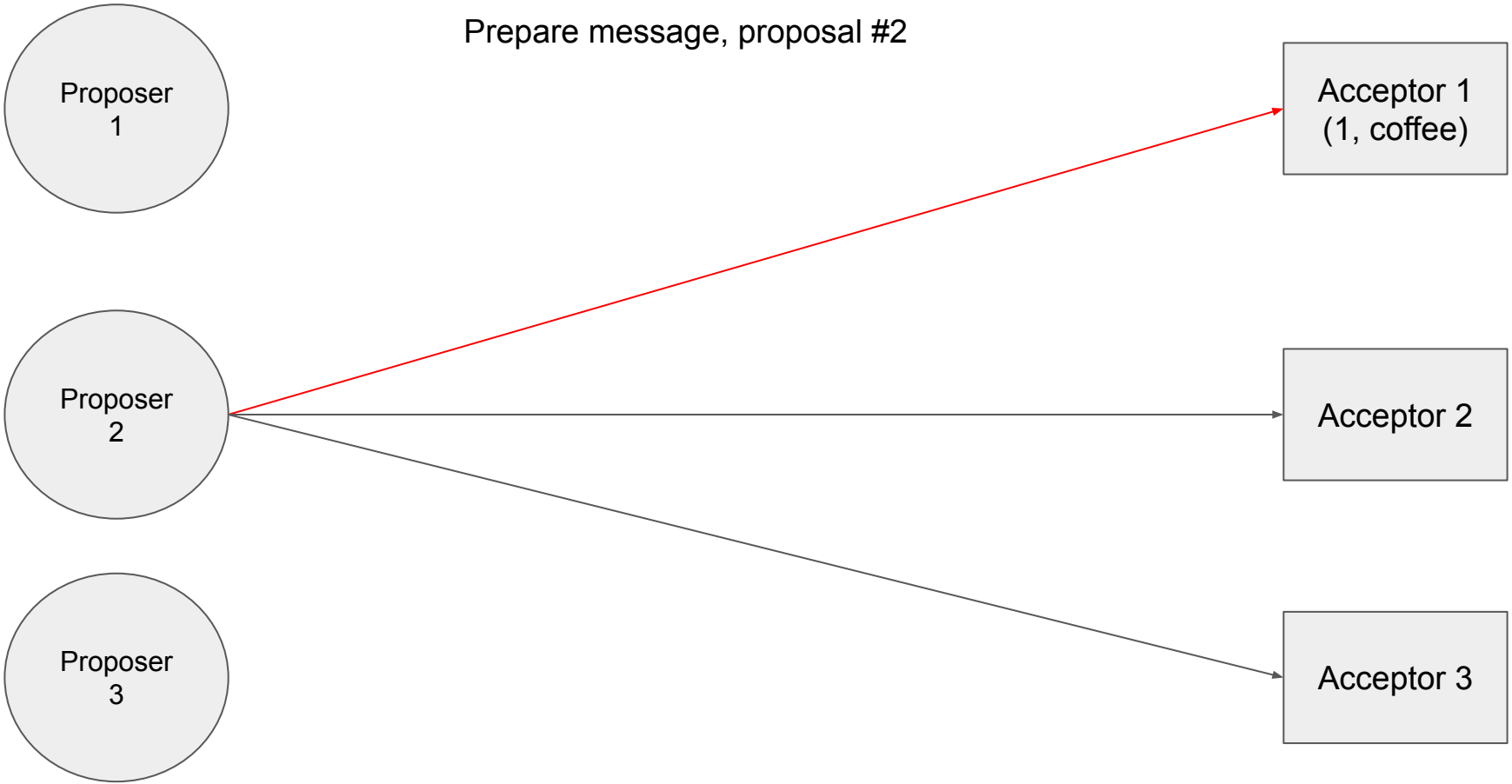
Paxos to decide what drink to have with breakfast



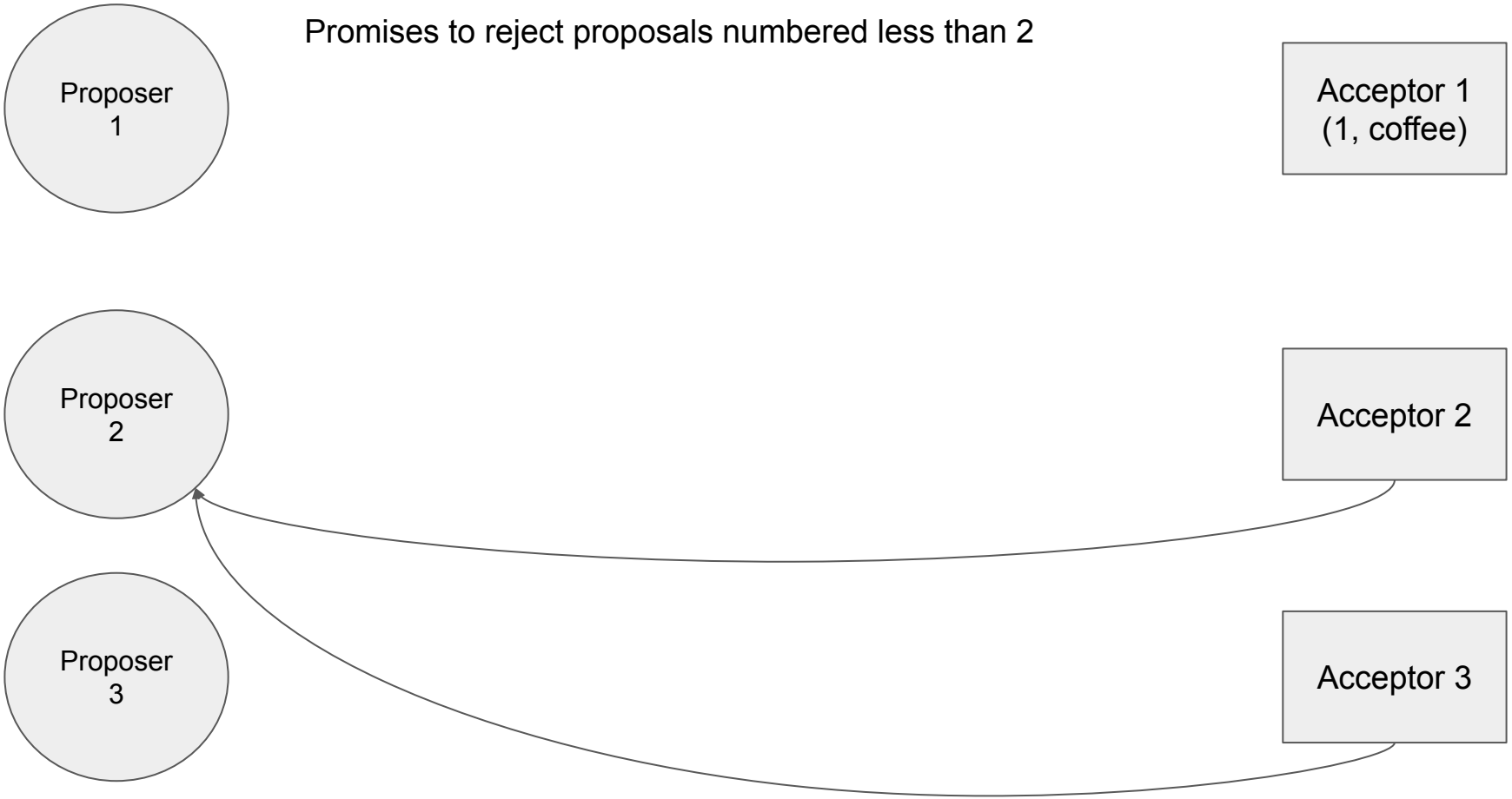
Paxos to decide what drink to have with breakfast



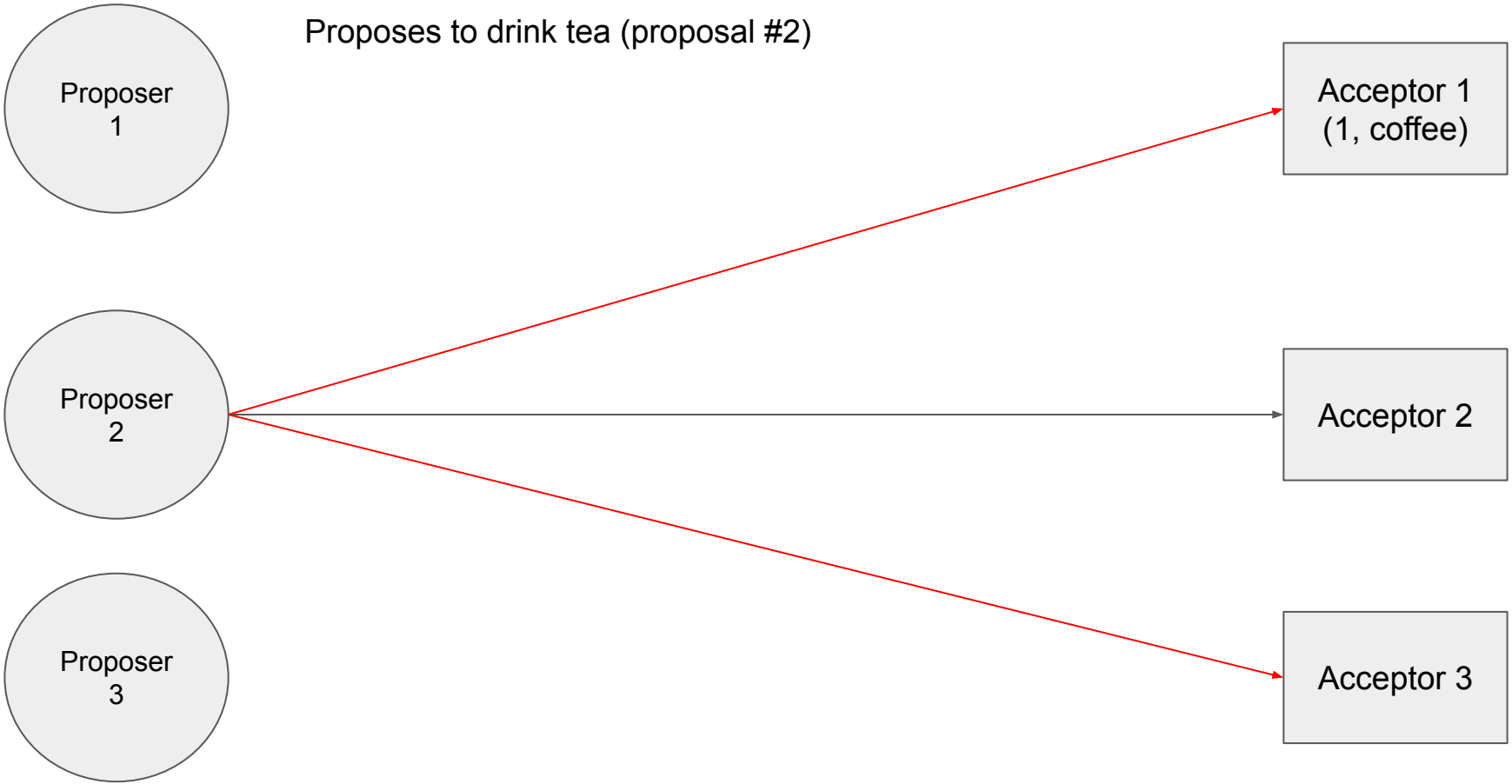
Paxos to decide what drink to have with breakfast

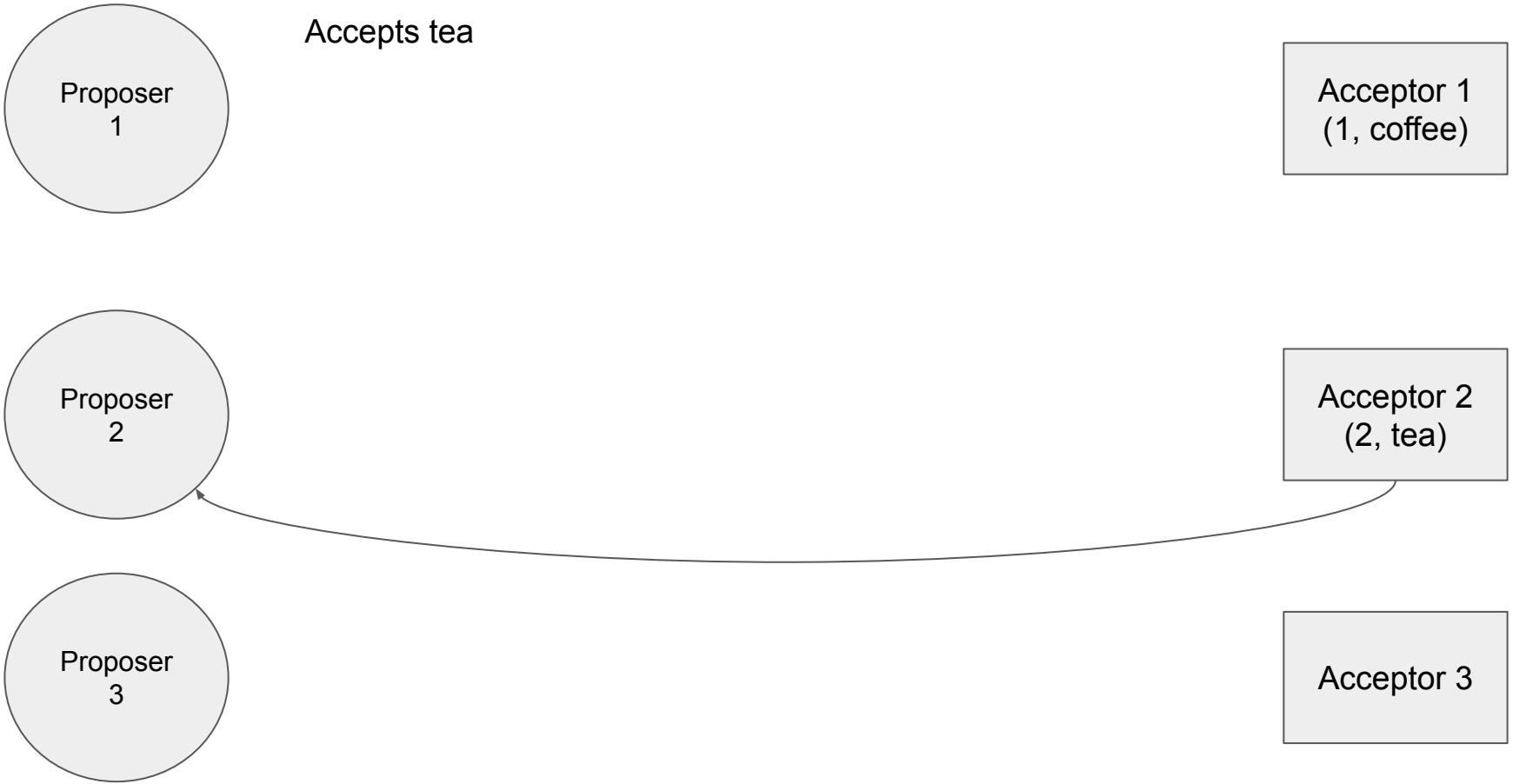


Paxos to decide what drink to have with breakfast

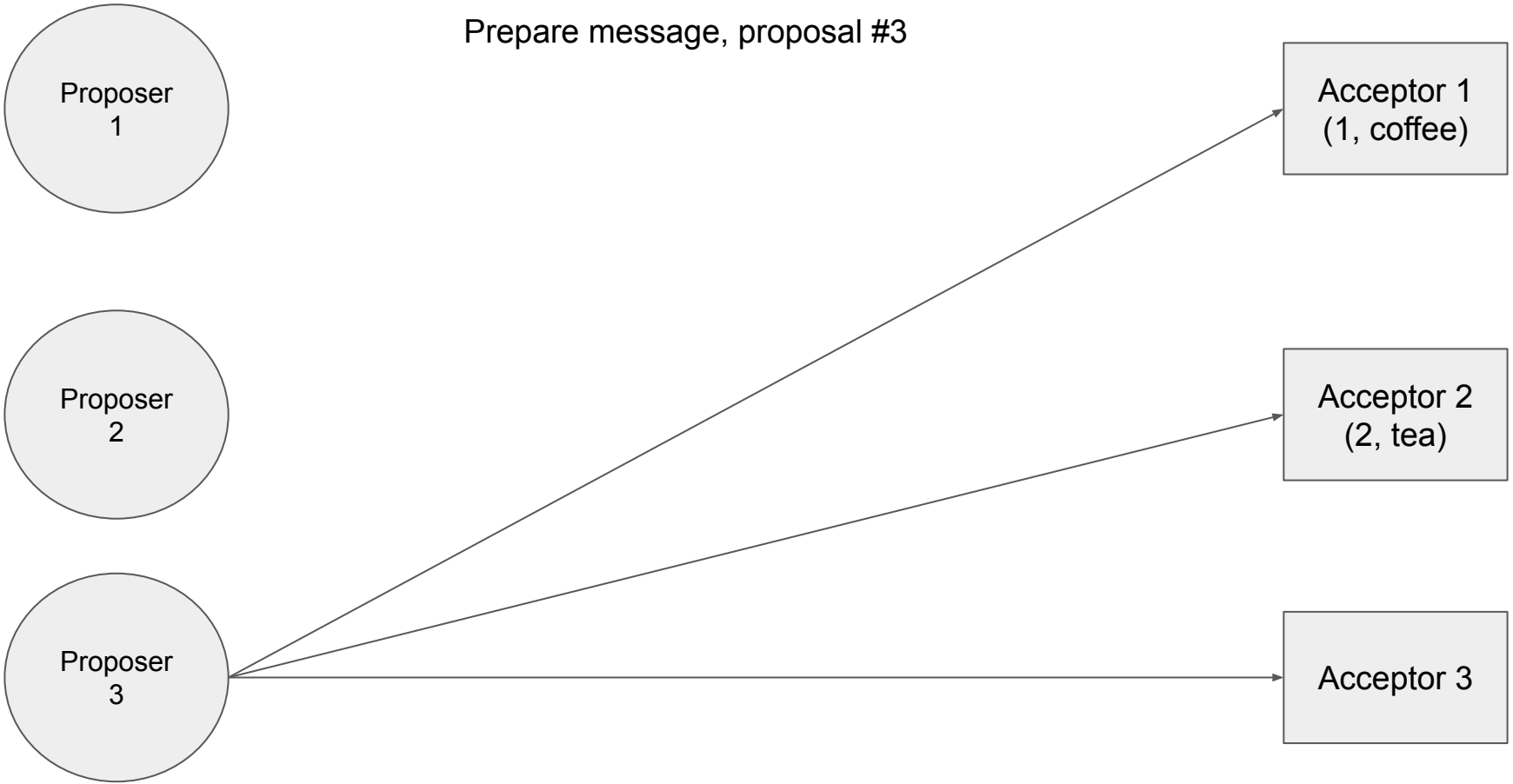


Paxos to decide what drink to have with breakfast

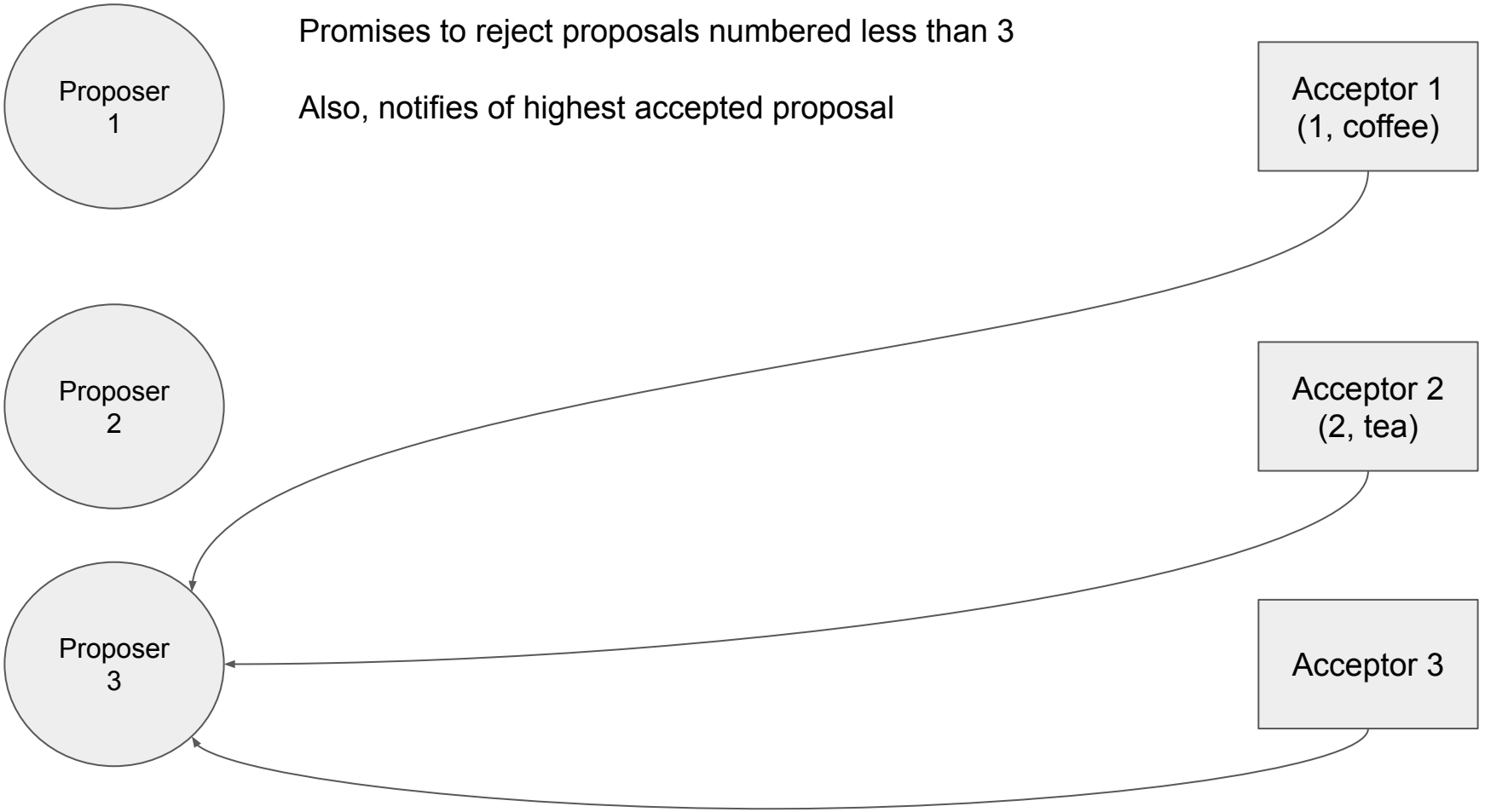

Proposes to drink tea (proposal #2)
Proposer 1
Proposer 2
Proposer 3
Acceptor 1 (1, coffee)
Acceptor 2
Acceptor 3



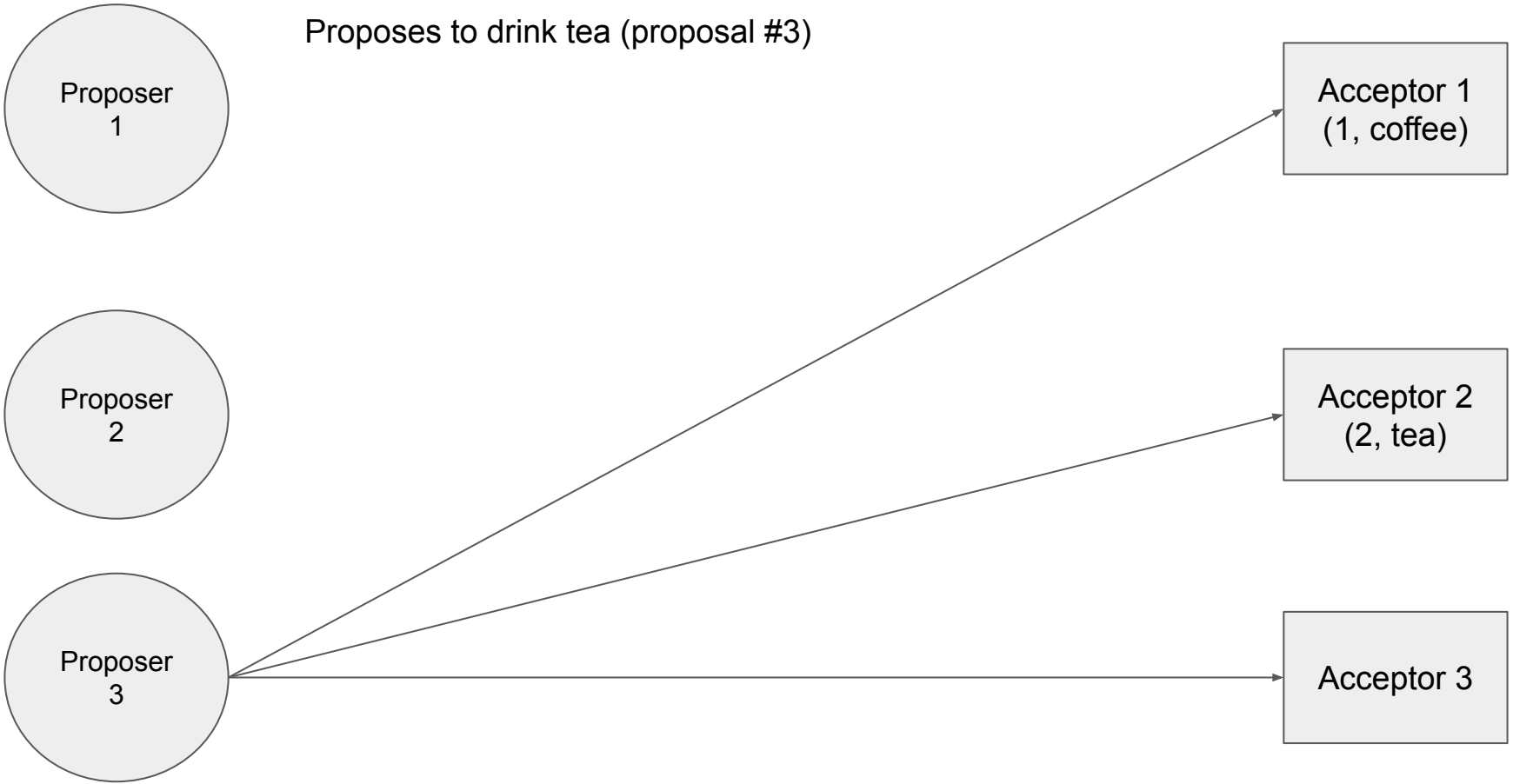
Paxos to decide what drink to have with breakfast



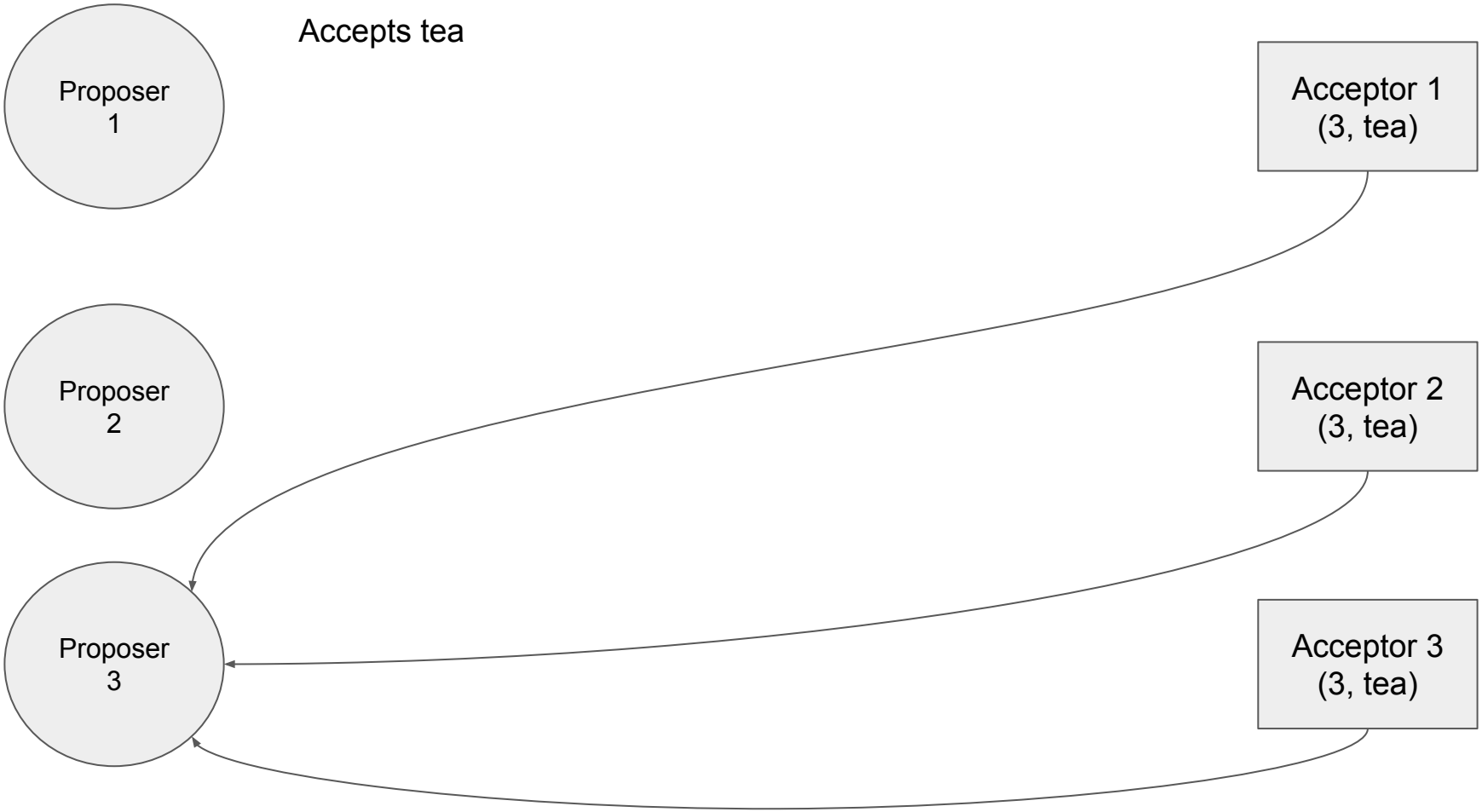
Paxos to decide what drink to have with breakfast



Paxos to decide what drink to have with breakfast



Paxos to decide what drink to have with breakfast



Paxos to decide what drink to have with breakfast

Liveness

- Elect a leader, aka a “distinguished proposer”
  - Leader should be stable - save election cost
  - Paper suggests node that finishes phase 1 is implicitly made leader
- Can’t really assume there is a single leader; multiple nodes may think they are leader.
  - Randomized timeouts to prevent fighting in new elections

Networking/Congestion Control Again!

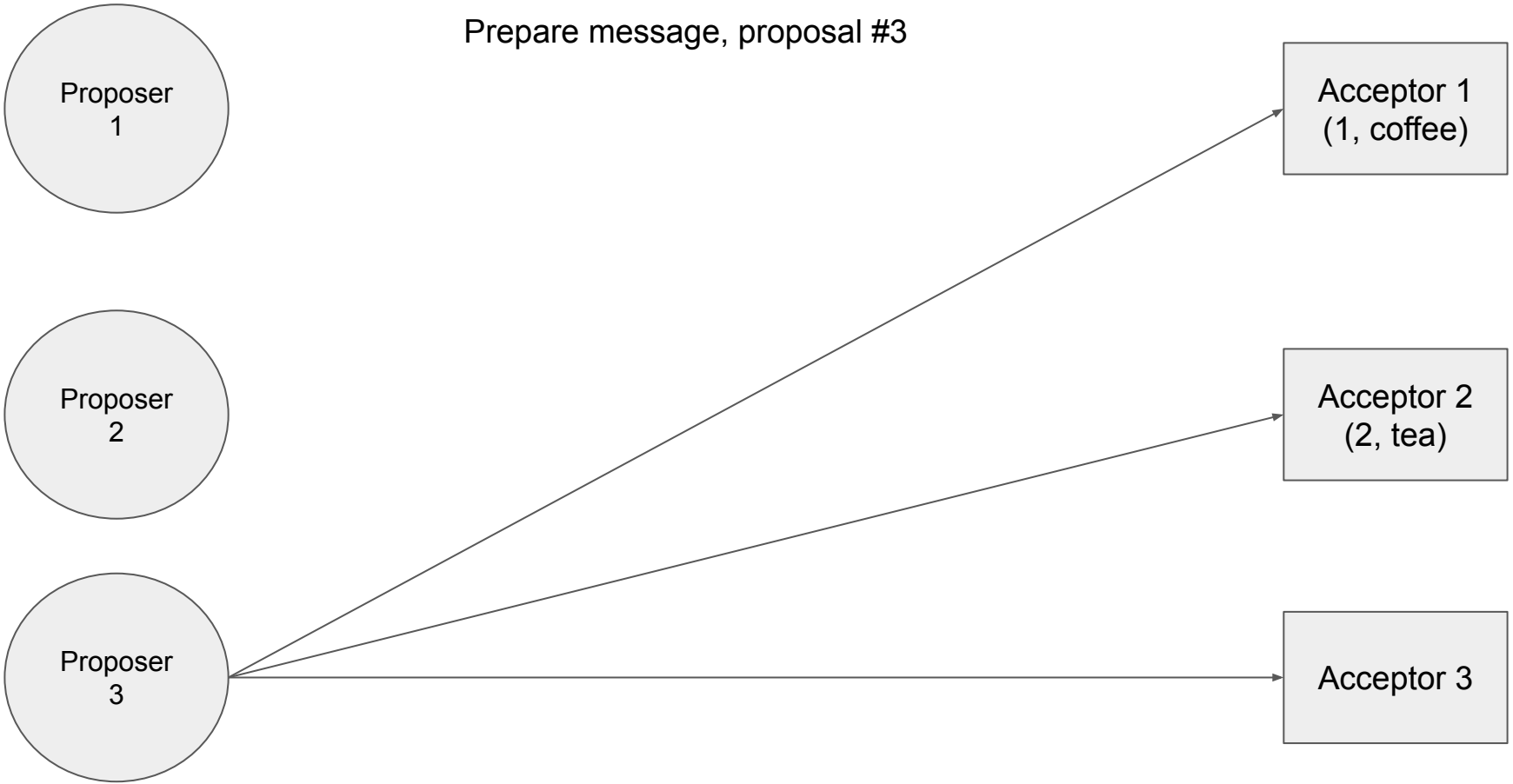
- Paxos: collision is when you get responses for prepare messages from a majority of servers but don't get accepts from a majority.
- Exponential backoff
  - After n collisions, randomly wait {0, 1, 2, …, 2^n-1} time units before retrying.
- Additive Decrease, Multiplicative Increase (PMMC paper)
  - On collision, double the retry time
    - Give the other proposer a chance to finish
  - On success, decrement the retry time
    - Don't want to wait too long if you can go faster

Paxos Assumptions - Revisited

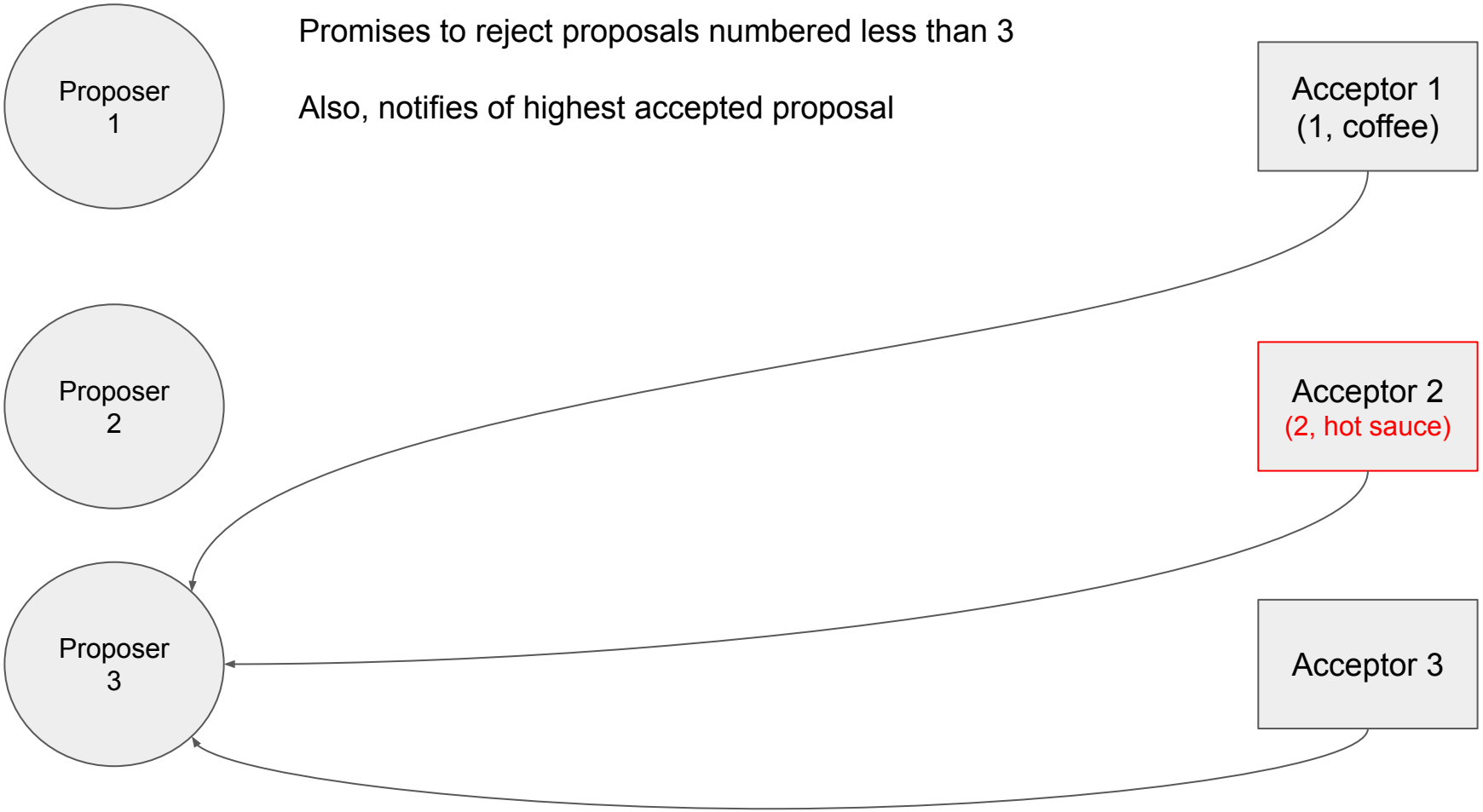
- Non-Byzantine
- Agents remember information to help restore state on failure
- Messages can be delayed, dropped, and duplicated, but cannot be corrupted

Byzantine Faults

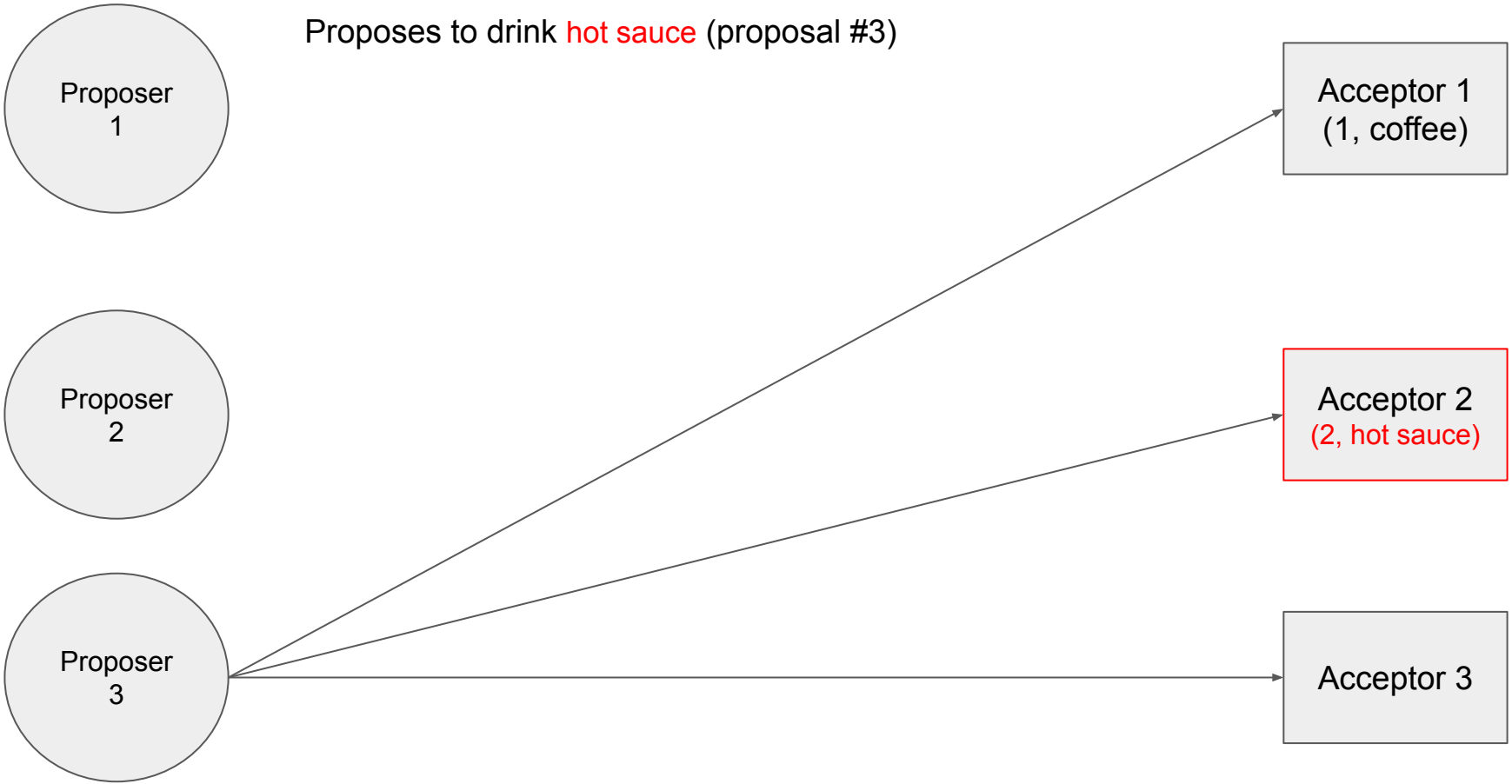
- Unreliable instead of unresponsive/inactive
  - Indistinguishable from functioning agent
  - Can send bad/incorrectly process messages
- Paxos is not Byzantine fault tolerant
  - Takes maximum from node responses
  - Nodes promise not to accept lower numbered proposals



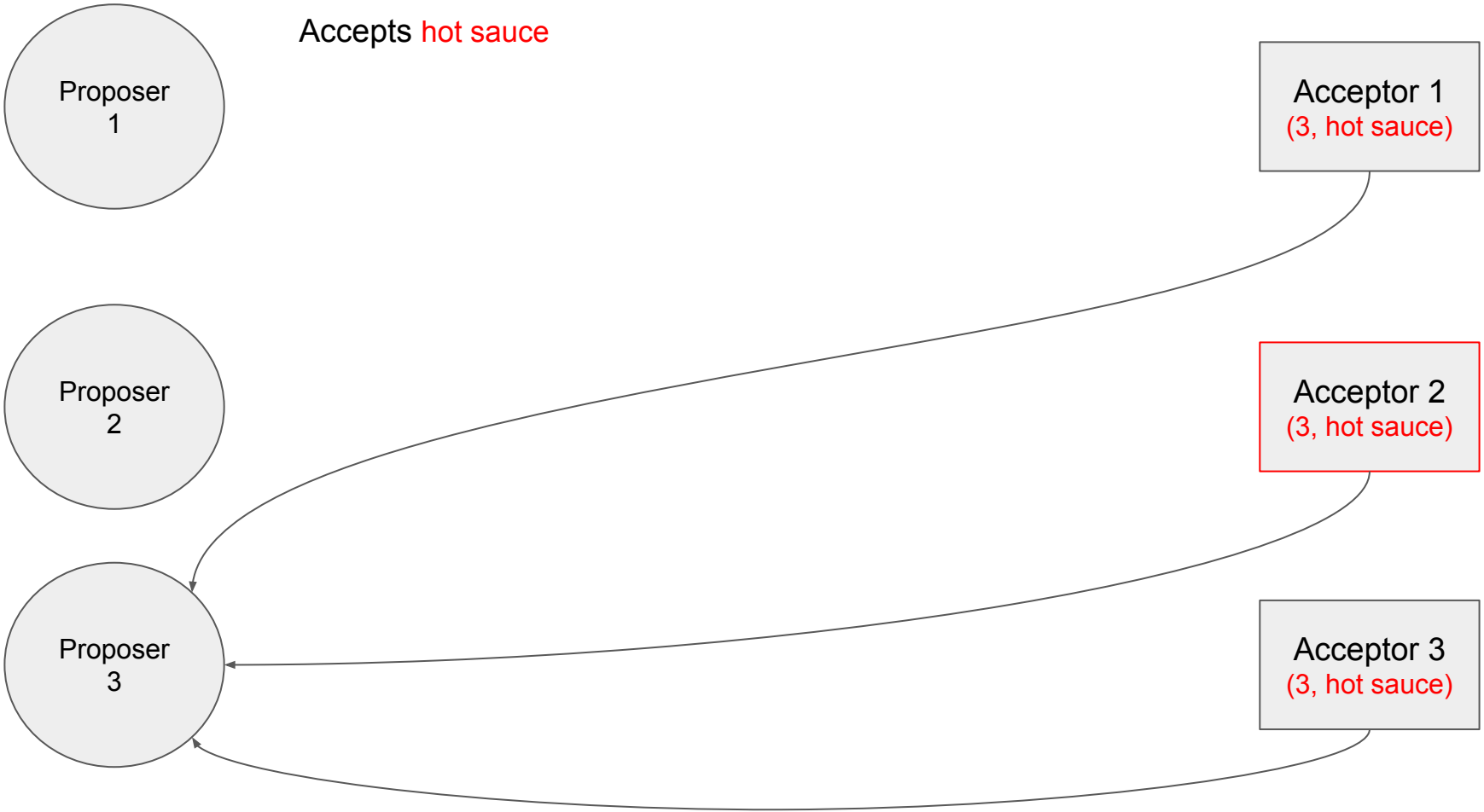
Paxos to decide what drink to have with breakfast



Paxos to decide what drink to have with breakfast



Paxos to decide what drink to have with breakfast

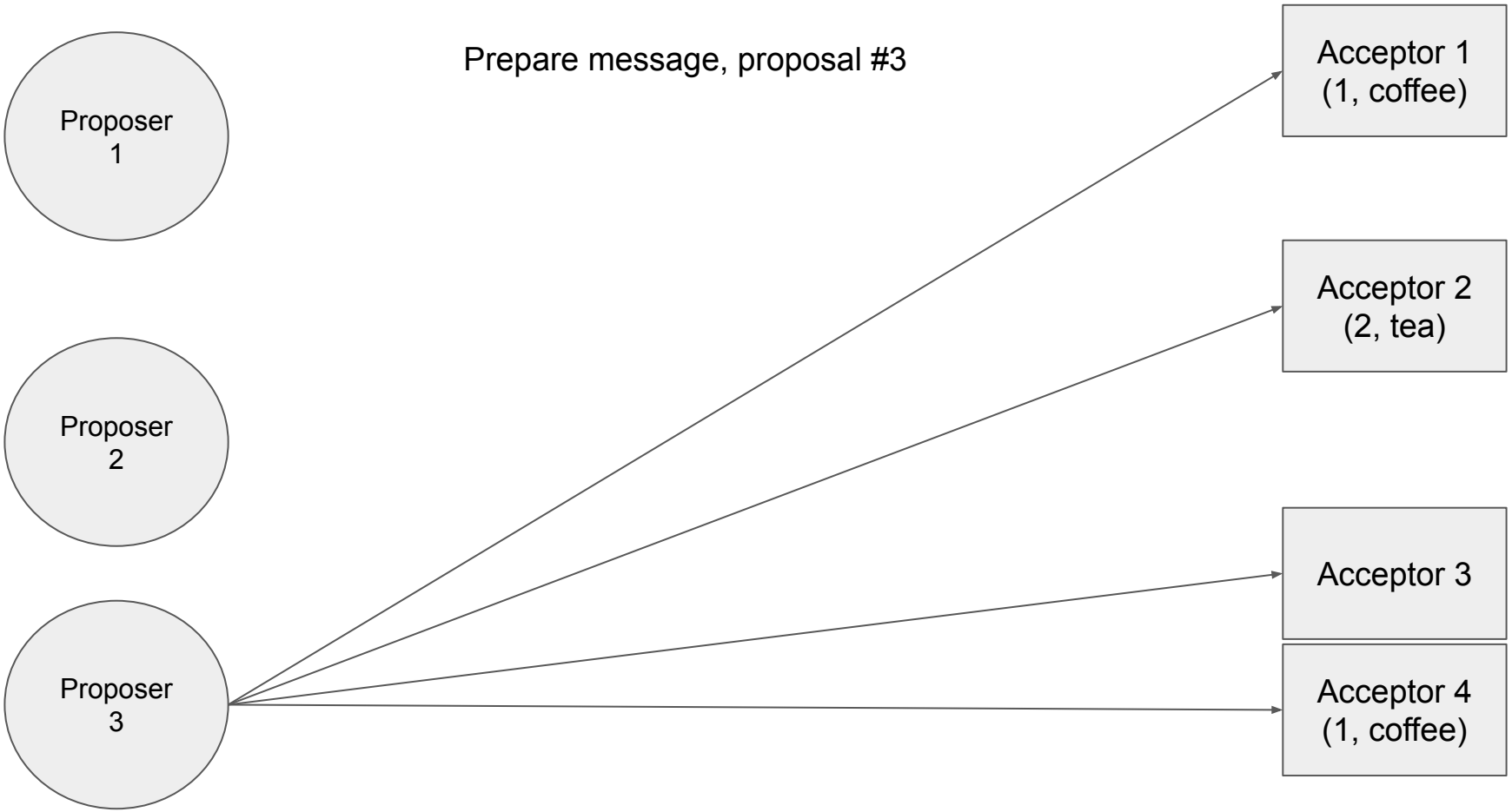


Paxos to decide what drink to have with breakfast

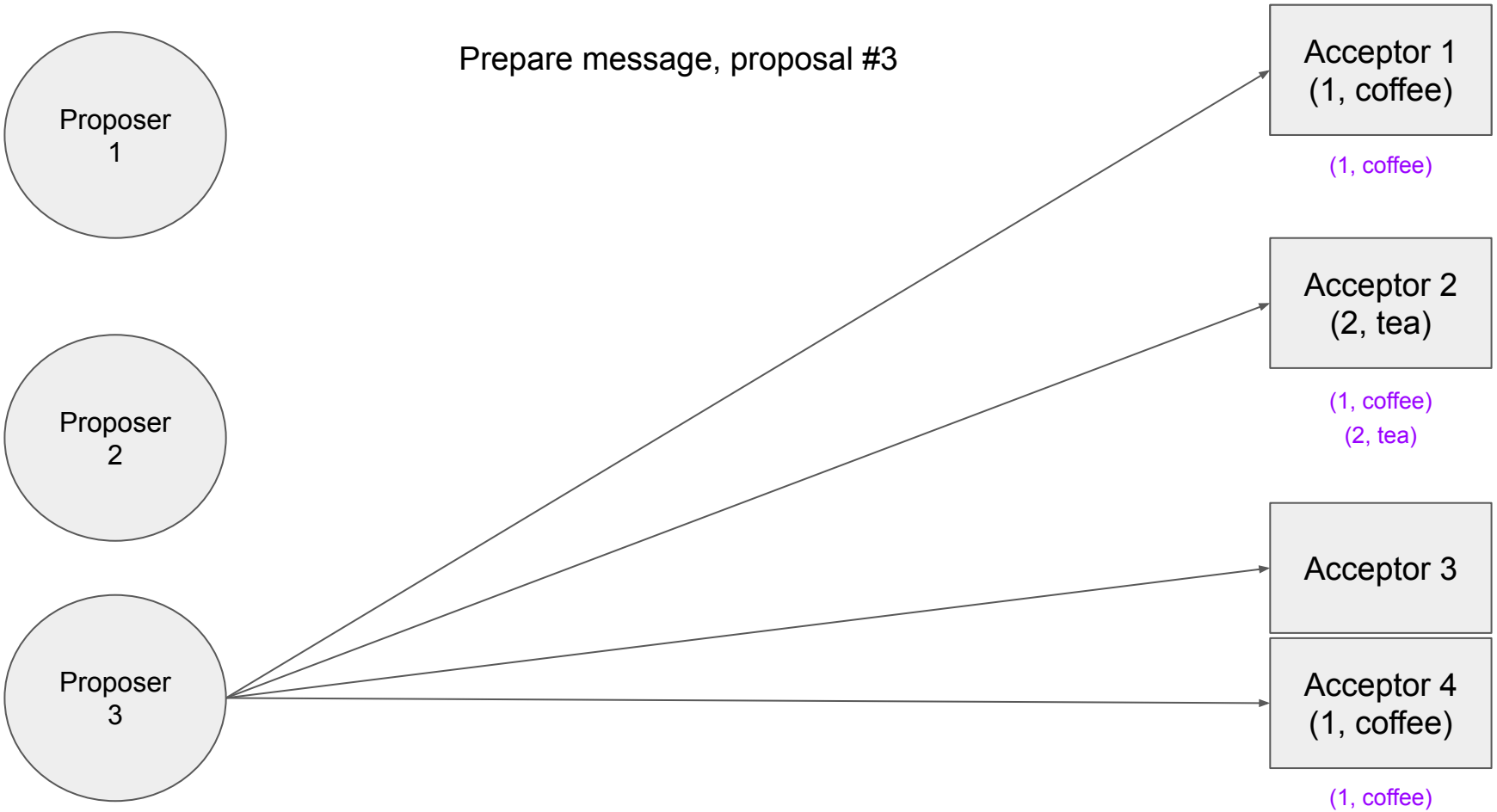
Byzantine Paxos (Lamport, 2010)

- Can dominate f byzantine agents by deploying >4f total agents
  - But we can do better! (>3f)

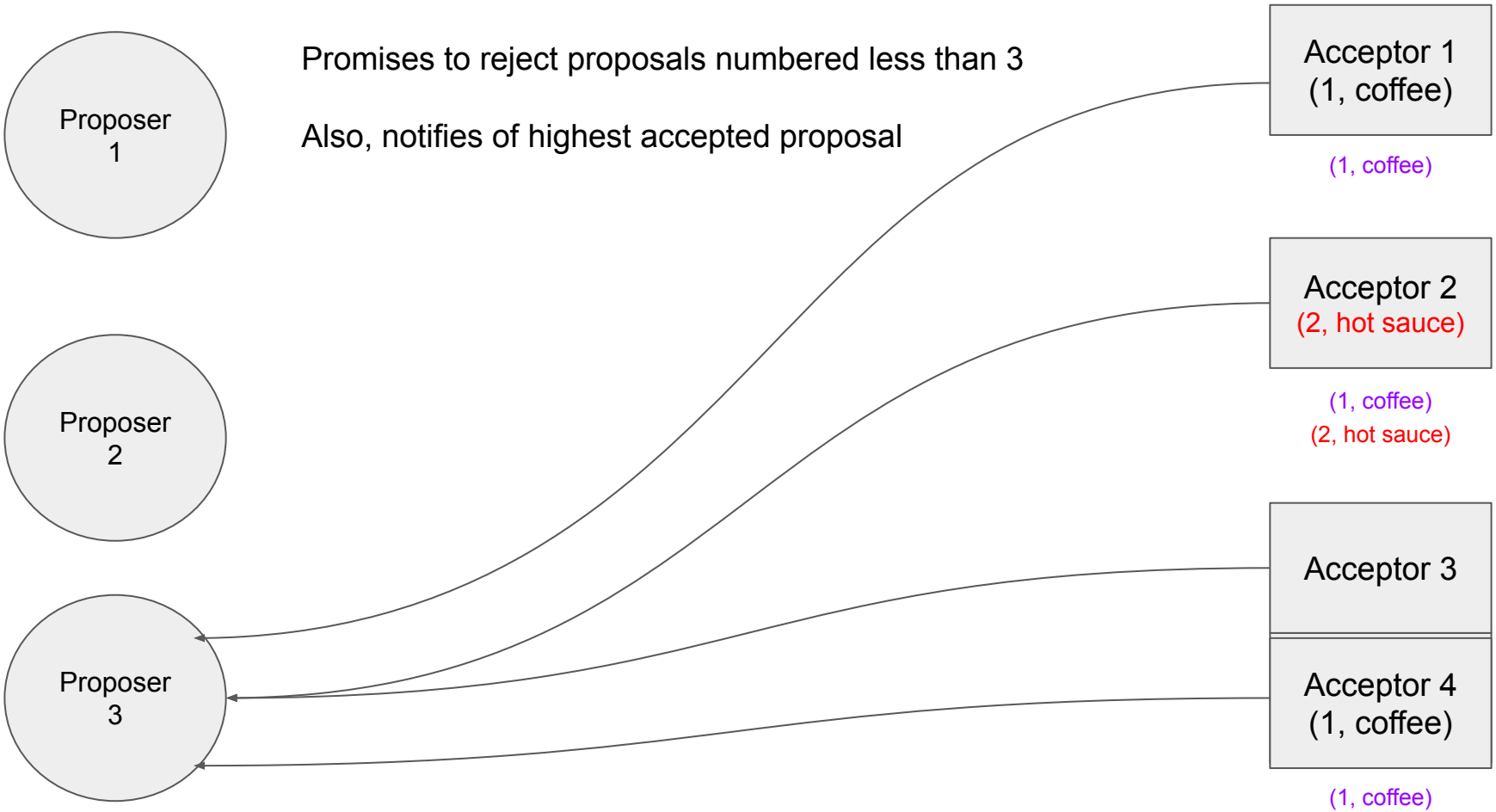
- Introduce “verification” phase and additional conditions on proposed values
  - Require stronger conditions for guarantees
  - Double check suggested values from acceptors - make sure they were proposed before



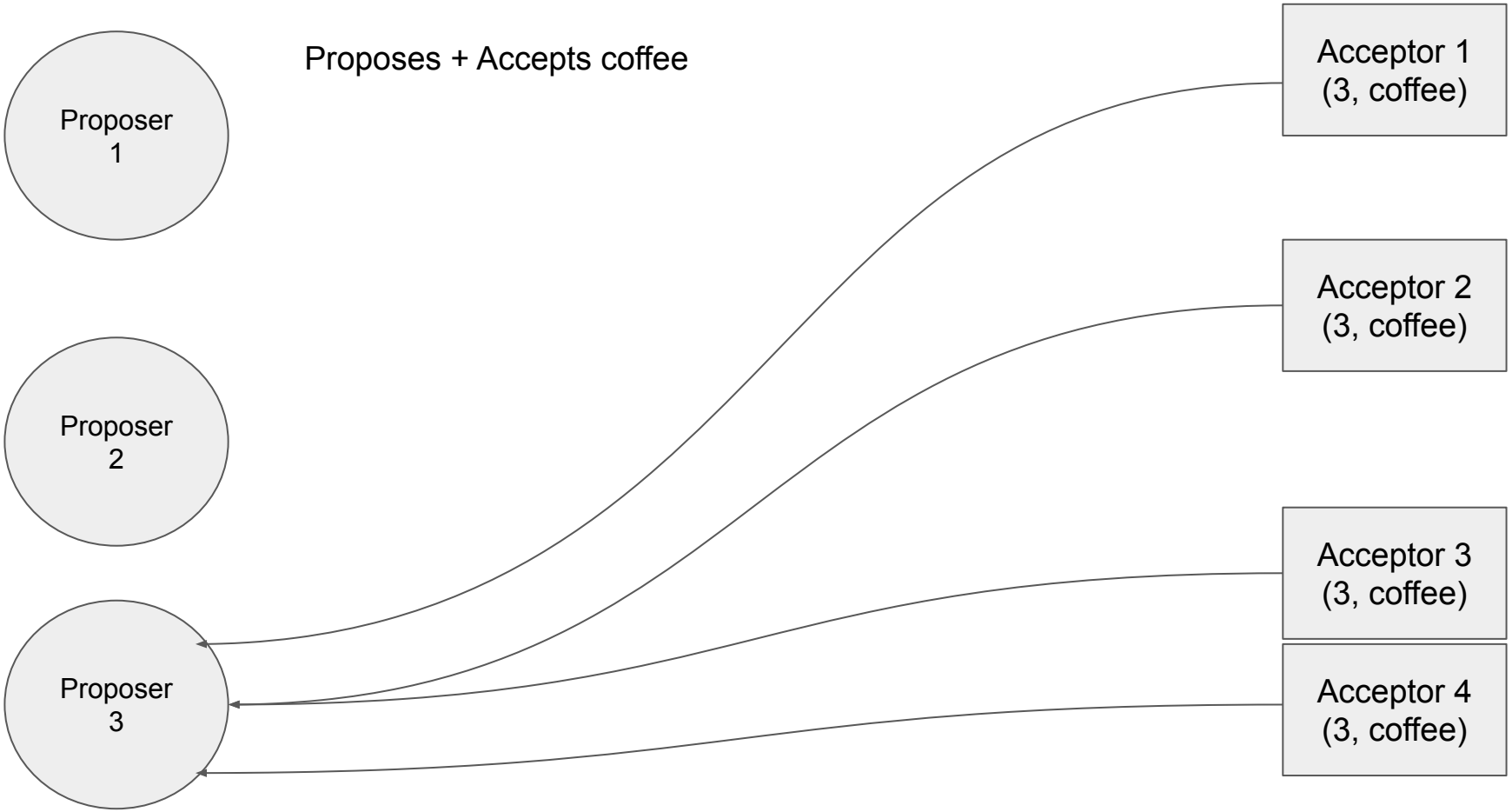
Paxos to decide what drink to have with breakfast



Paxos to decide what drink to have with breakfast



Paxos to decide what drink to have with breakfast



Paxos to decide what drink to have with breakfast

Byzantine Paxos

- Harder to account for byzantine leaders
  - Liveness issues
- Expensive
  - More agents
  - More state needs to be saved
  - Higher bandwidth usage

Paxos Assumptions - Revisited

- ~~Non-Byzantine~~
- Agents remember information to help restore state on failure
  - Log paxos state (current proposal, accepted proposals, etc.)
- Messages can be delayed, dropped, and duplicated, but cannot be corrupted
  - This sounds familiar...

Discussion

- Given the costs of implementing a byzantine consensus algorithm, in what cases would it be useful?
- What are some of the examples of issues that might manifest in Paxos during disk failures?
- Paxos accounts for what we previously considered network layer issues (message dropping, duplication, delayed). Should some of this responsibility be shifted back onto the network layer? What would distributed algorithms look like with stronger network guarantees?
    - Tbt end-to-end argument

Paxos in the Wild

- MultiPaxos
  - Running a fresh Paxos instance for every command in a stream would be expensive
  - By keeping a stable leader, may be able to completely skip phase 1
- Cheap Paxos
- Fast Paxos
- Vertical Paxos

Paxos in the Wild

- Google Chubby (lock service) and Bigtable
- Google Spanner
- Amazon Elastic Container Services
- Apache Cassandra NoSQL

- Paxos clusters are usually composed of 3, 5, or 7 nodes

- Raft is considered main alternative to Paxos
  - Advertises itself as easier to understand and implement
- Overview
  - Nodes are either a “leader” or a “follower”
    - There can only be one leader
  - Leader is responsible for log
    - Client sends request to leader
    - Leader tentatively executes request, forwards to followers
    - If majority of followers reply, result is committed
    - On leader change, followers forced to copy new log
  - Election based on timeouts and “terms”
- In perfect no-fault case, Paxos is “more optimal”

FLP Impossibility Result and Safety vs Liveness

- In an asynchronous system, “no [deterministic] consensus protocol is totally correct in spite of one fault”
  - N generals problem is impossible if they don’t share a clock
- Three characteristics of asynchronous consensus algorithms
  - Termination
  - Agreement
  - Validity
- In practice
  - Safety: Agreement, Validity
  - Liveness: Termination
- Paxos: safe at the cost of liveness
  - Termination provided by randomness (no longer deterministic)

Discussion

- Are there scenarios where liveness may be preferable to safety?