Distributed Consensus-Part 5: Paxos and Prepare Phase

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)

Monarchy vs. Parliament System

Issue: how to detect and sequence concurrent requests?

Paxos and Associated Algorithm

Paxos Made Simple

Leslie Lamport

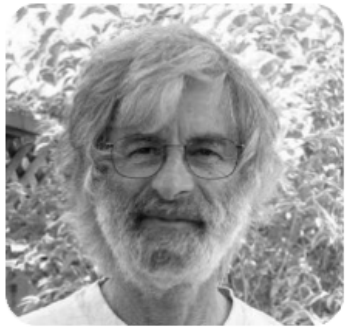
01 Nov 2001

In fact, it is among the simplest and most obvious of distributed algorithms.

Paxoi
Παξοί

The natural port of Gaios, created by the islet *St Nicolas*

Fictional consensus system



PHOTOGRAPHS

BIRTH:

7 February 1941 in New York, New York

LESLIE LAMPORT

United States – 2013

CITATION

For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency.

SHORT ANNOTATED BIBLIOGRAPHY | ACM DL AUTHOR PROFILE | ACM TURING AWARD LECTURE VIDEO | RESEARCH SUBJECTS | ADDITIONAL MATERIALS

If we could travel back in time to 1974, perhaps we would have found Leslie Lamport at his busy local

Feedback Symmetry

Issue: how to detect and sequence concurrent requests?

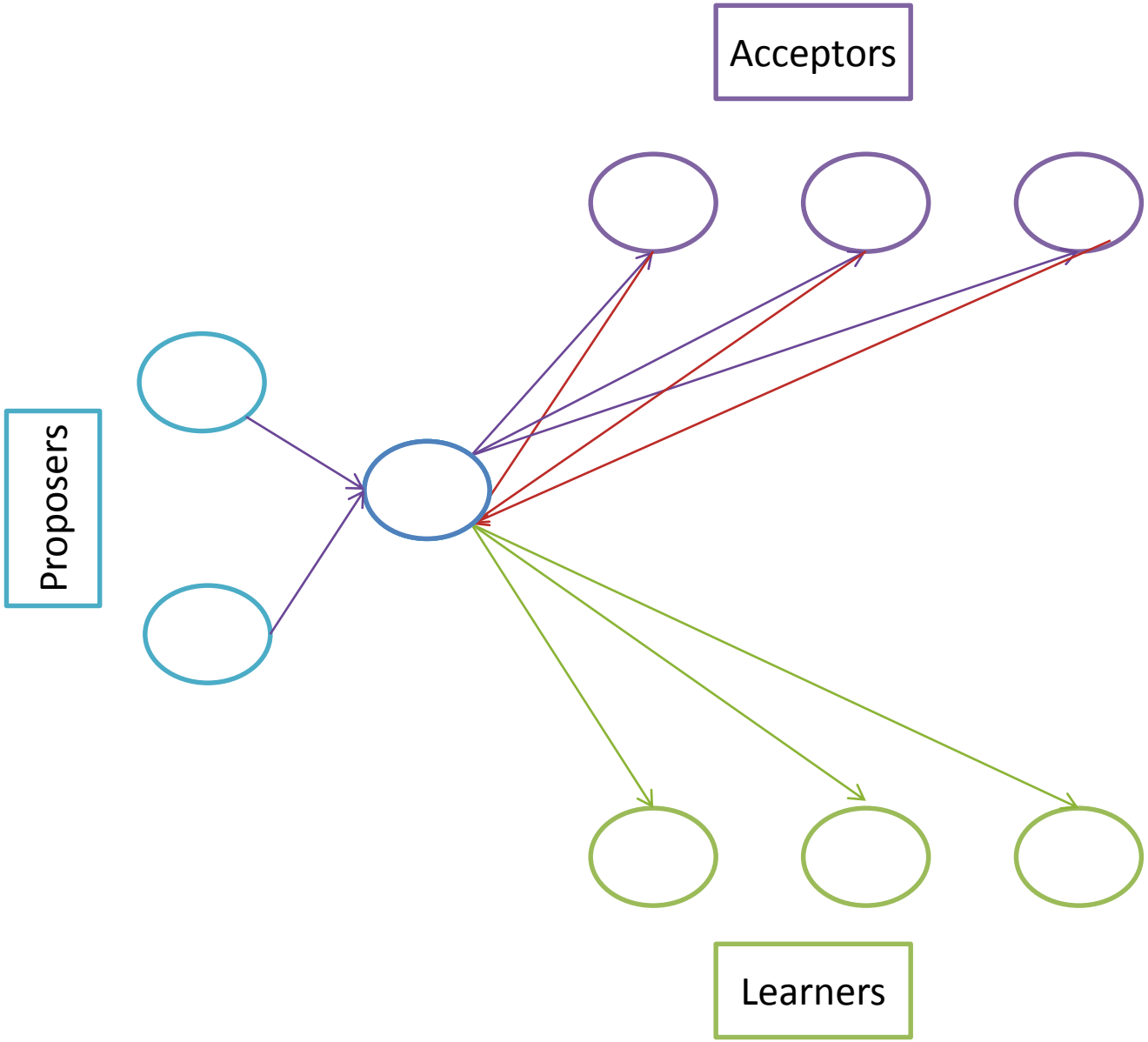
A single replica decides

Multiple processes (acceptors) decide

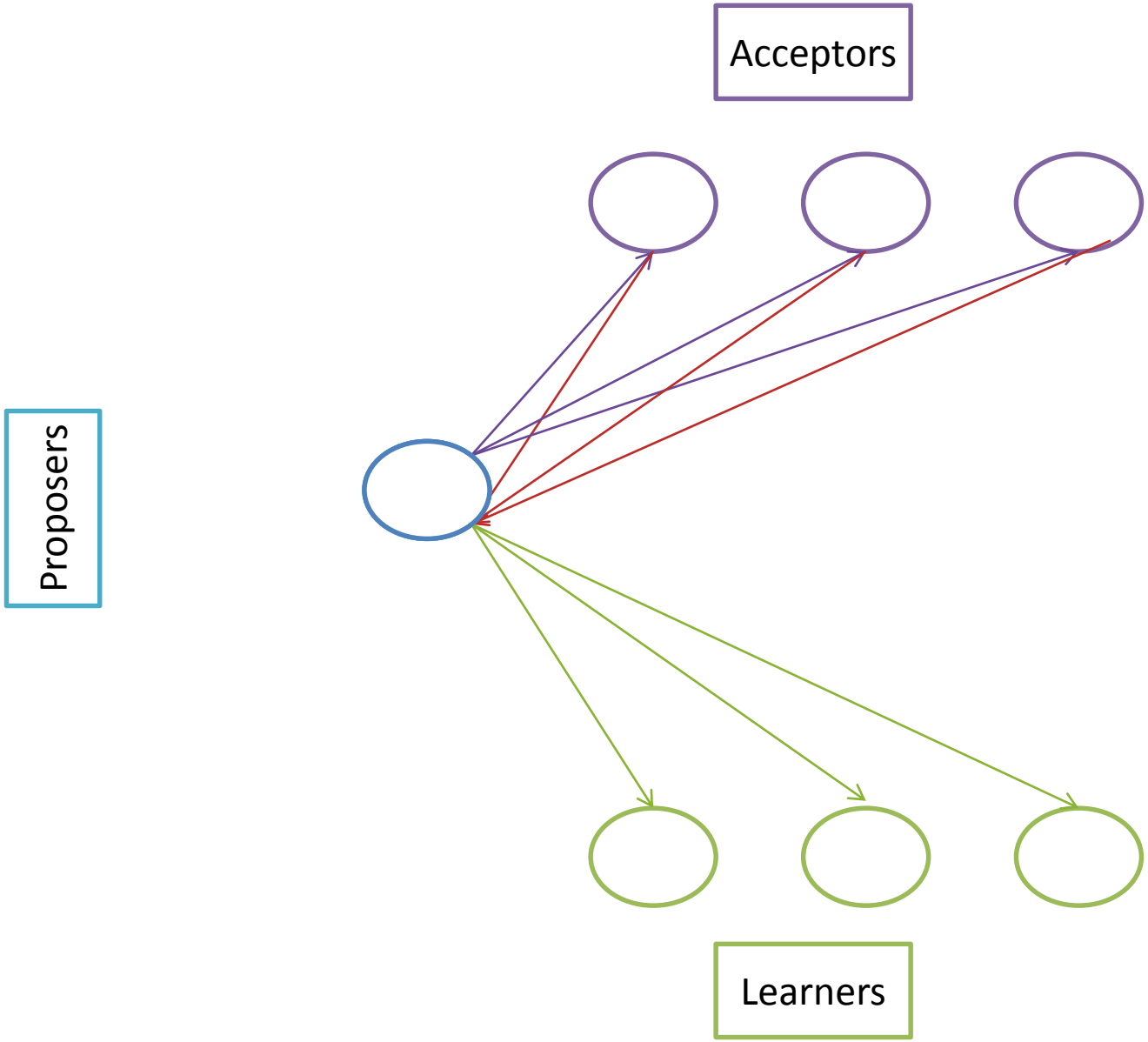
They already decide site-specific aspects of feedback in synchronous case

Asynchronous version does not make sense as multiple rounds required

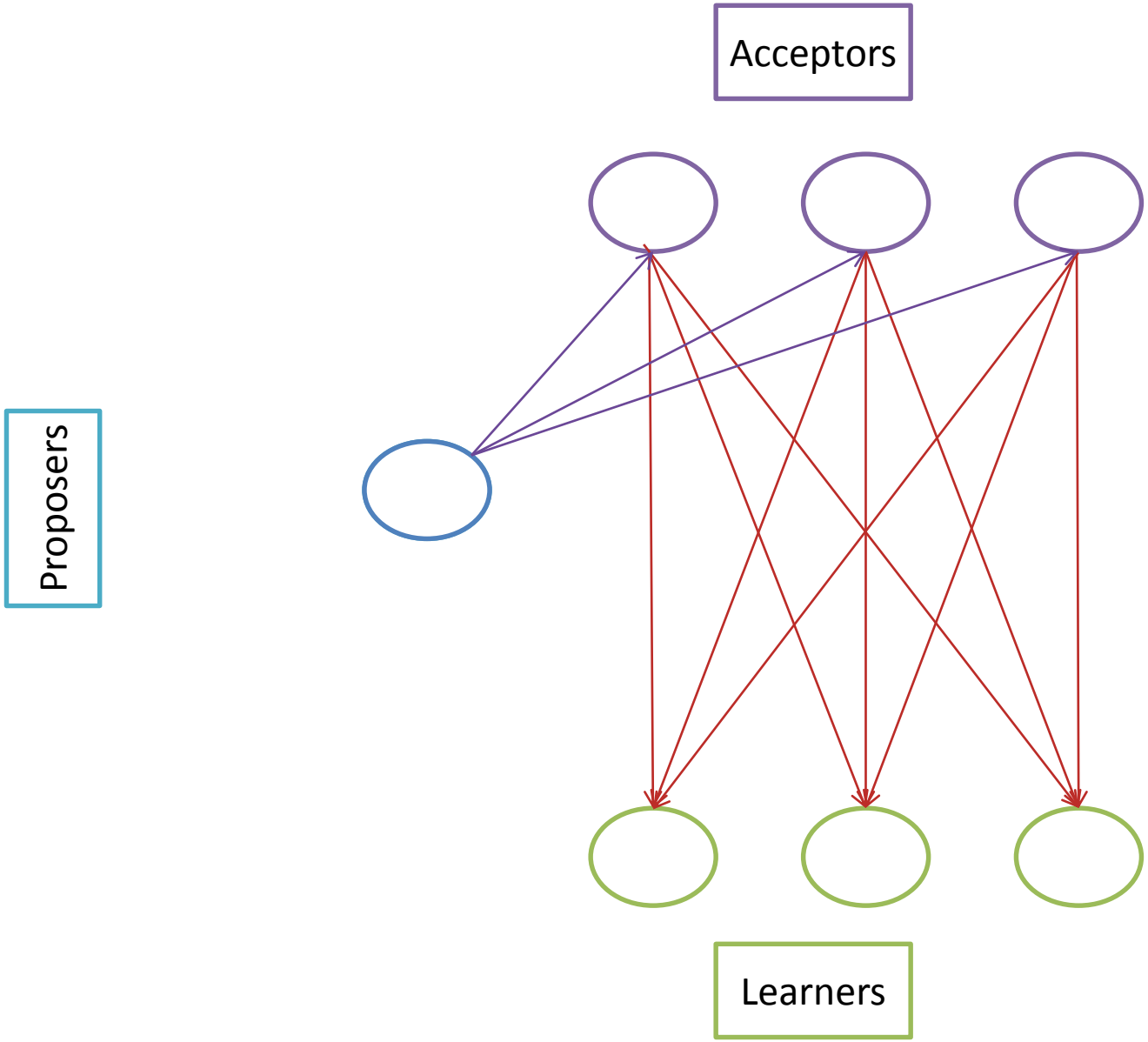
Centralized



Synchronous



(Almost) Basic Paxos



Assumptions and Guarantees

- Messages can be delayed and out of order and get lost and be retransmitted
  - At application level
- The replicated value is written once
  - A single command to a replicated state machine
- All learner commits have the same proposed value.
- A learner site commit to this value
  - When and if it receives its acceptance from a majority of known acceptors
- Nodes can come back up and messages can be retransmitted
  - Eventually all listeners will commit to this value

Majority Rules

Accepter (accept)

Each acceptor accepts only the first proposal it gets

Learner (accepted)

A learner picks the value that has majority acceptances

Different acceptors make get proposals in different order

Acceptances may arrive in learners in different order so different learners may use different majorities

But each pair of majorities will have at least one member in common so will select same proposal

Majority Impossible Based on Proposal Order

Two proposal may get the same number of votes

Choose odd number of acceptors?

Given N acceptors, it is possible to have N concurrent proposals, each of which wins in one acceptor

Some Proposals Are More Equal Than Others

Accepter (accept)

Each acceptor accepts the proposal with the highest proposal number it will see

Learner (accepted)

A learner picks the value that has majority acceptances

Cannot predict future

Vote Early Vote Often

Accepter (accept)

Each acceptor accepts a proposal if its proposal number is higher than what it has seen so far.

Learner (accepted)

A learner picks the value that has majority acceptances

An acceptor can change its mind.

A learner may learn multiple values associated with higher proposal numbers

Individual Vote Can Change But Not the Aggregate Vote

Learner (accepted)

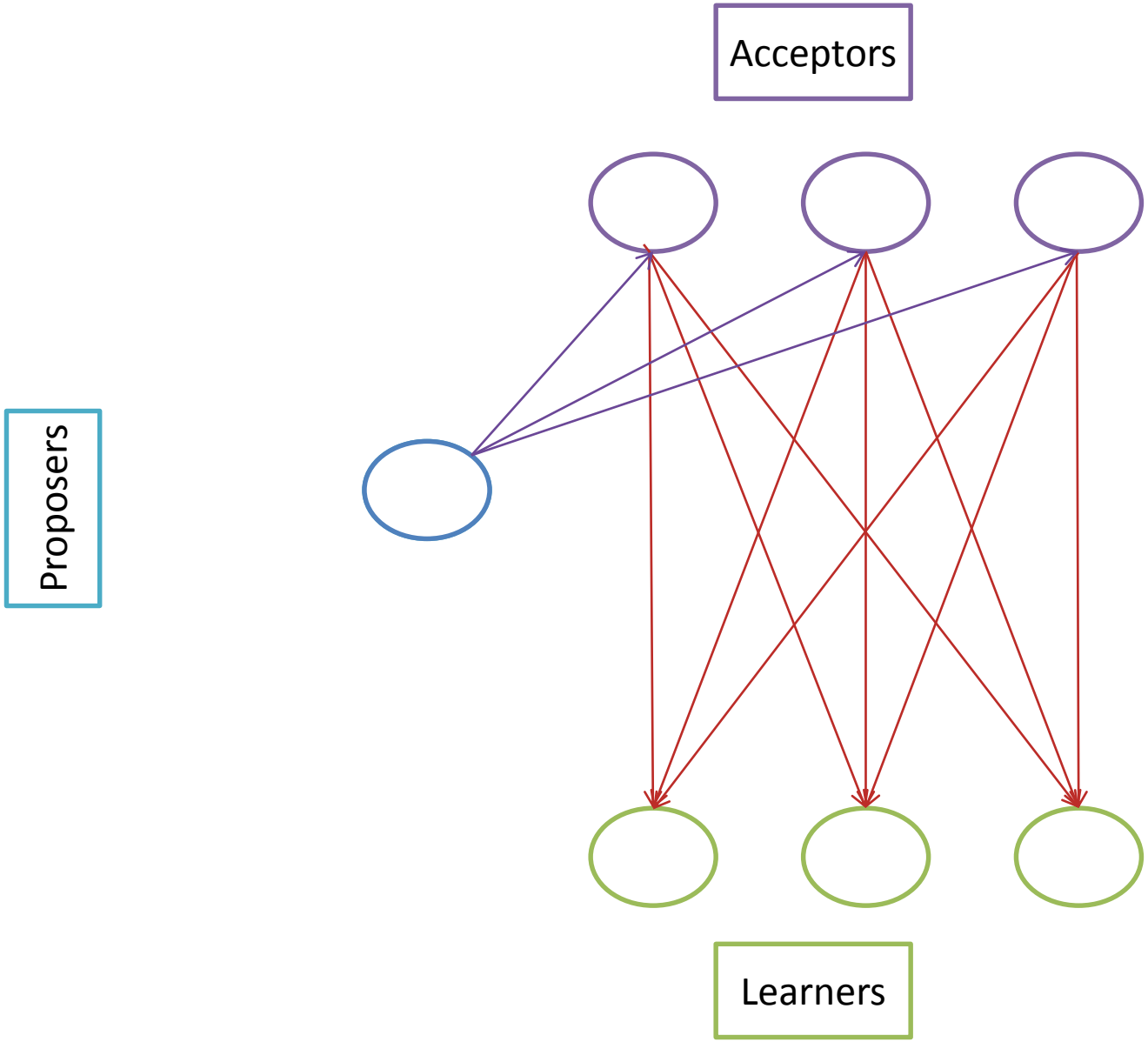
A learner picks the first value that has majority acceptances

Different learners can get different majorities

Common members may have changed their mind

So different learners can pick different values

Control Source Rather Than Sink



Propose Does not Play Spoiler

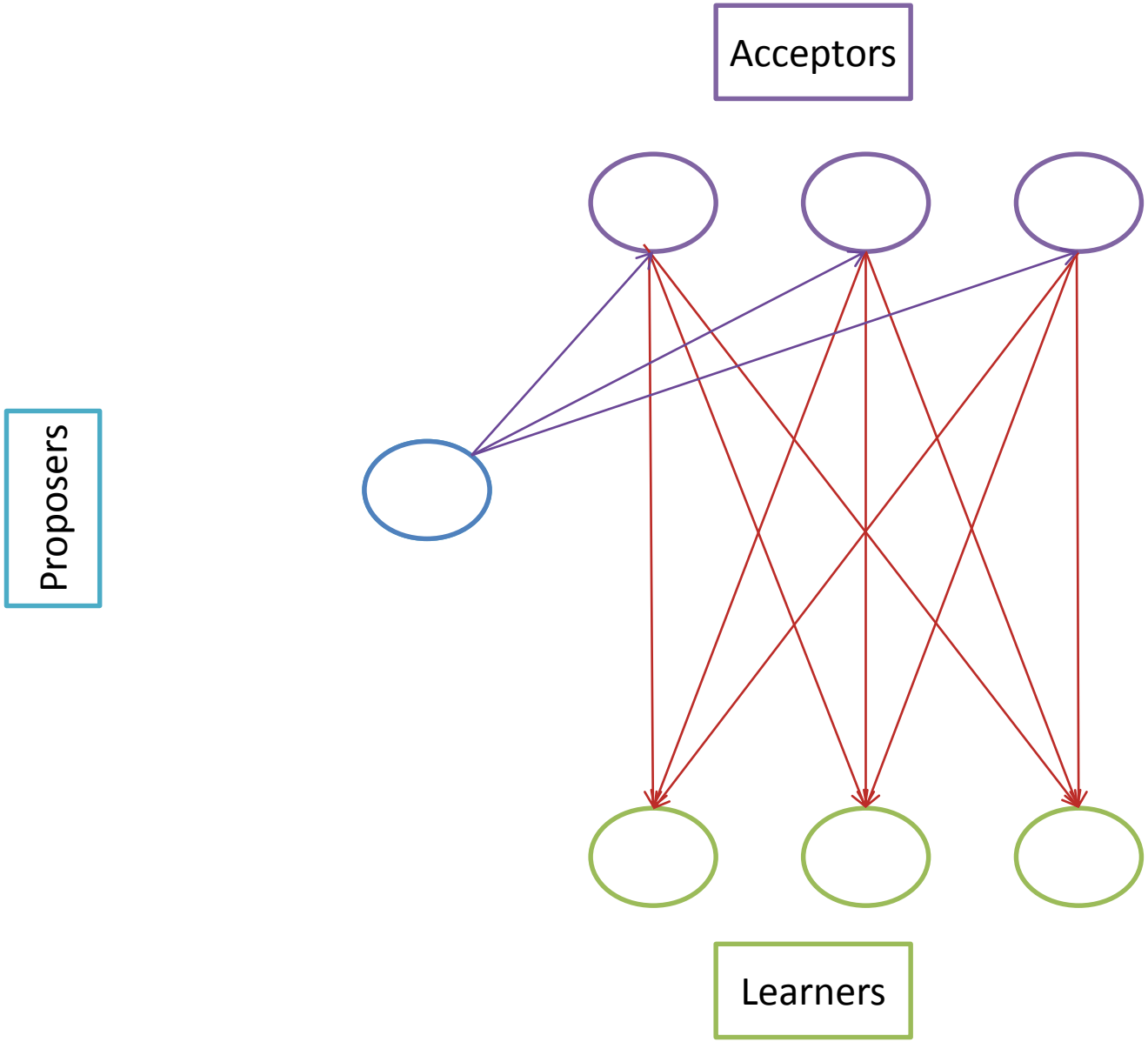
Proposer (propose)

A proposer does not override consensus (to be) achieved from earlier proposal

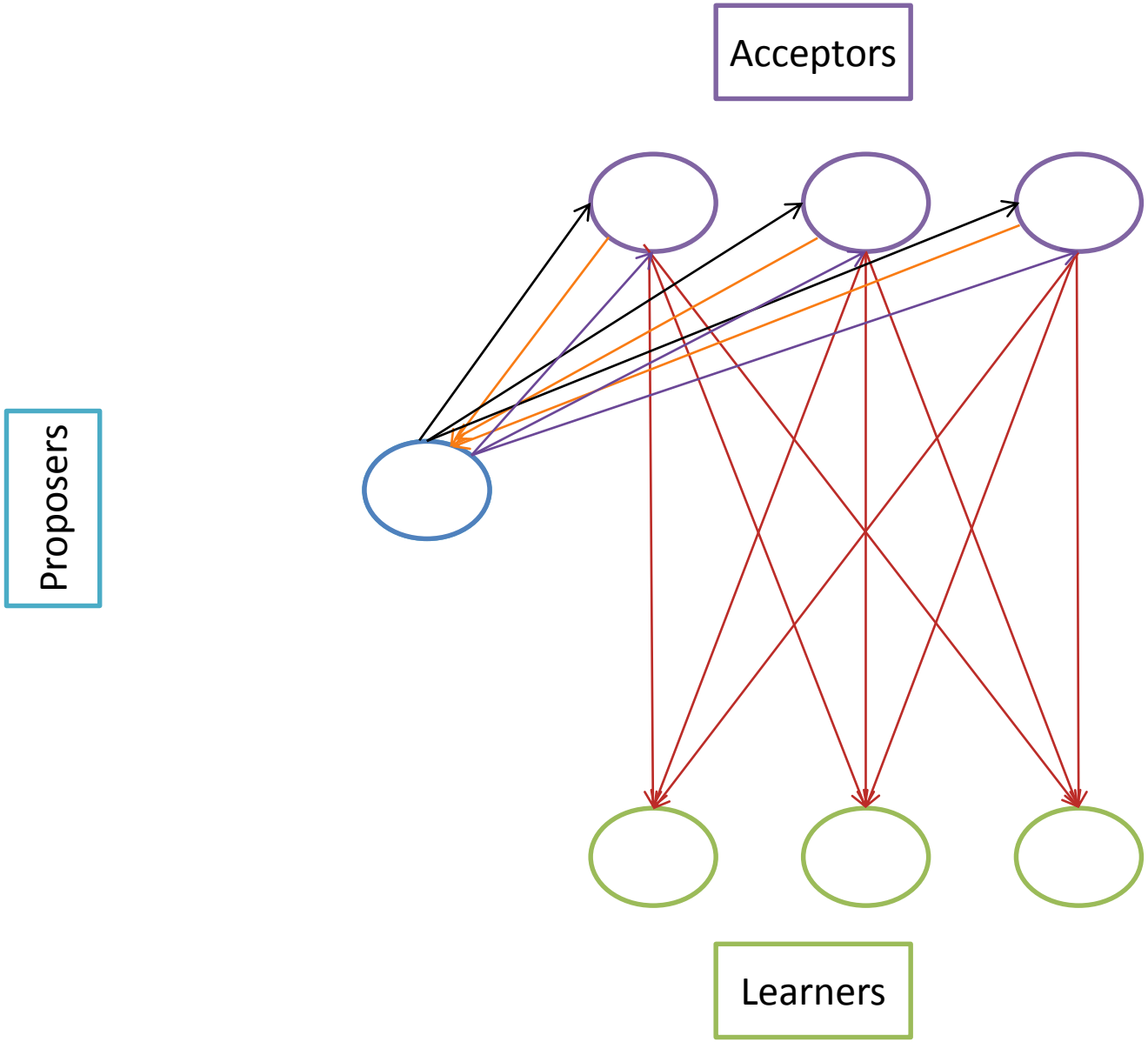
Propose needs to know distributed state before proposing

(Last known proposal, last accepted value)

One-Propose Phase, Two Overall



Two-Propose Phases Three-Overall



Two Phase Propose

Proposer
(pre-propose (p, v^p))

A propose invocation sends a prepare message to query the current state of each acceptor

Proposer
(prepared(p, v^p))

Proposer does not propose new value if it could override a learnt value from previous proposal

Acceptor (prepare)

Each acceptor sends back last accepted value and its proposal number (or optionally last seen proposal number if no acceptance so far) and updates last seen proposal number

Prepared Cases

Proposer
(prepared(p, v^p))

Proposer waits for responses from at least majority of acceptors (a^1, … a^m)

Some acceptor has seen a higher proposal ✓

No responding acceptor has accepted a previous proposal

Majority responding acceptors have accepted some previous proposal

At least one acceptor has accepted some proposal but no majority value in the acceptances majority

Prepared: Higher Proposal Exists

Proposer
(prepared(p, v^p))

Proposer abandons proposal if it learns about a higher number proposal

Progress rather than safety condition

Prepared Cases

Proposer
(prepared(p, v^p))

Proposer waits for responses from at least majority of acceptors (a^1, … a^m):

Some acceptor has seen a higher proposal

No responding acceptor has accepted a previous proposal ✓

Majority responding acceptors have accepted some previous proposal

At least one acceptor has accepted some proposal but no majority value

Prepared: No Known Previous Proposal

Proposer
(prepared(p, v^p))

Proposer sends its proposal and value if it does not know about a previous accepted value

Prepared Cases

Proposer
(prepared(p, v^p))

Proposer waits for responses from at least majority of acceptors (a^1, … a^m):

Some acceptor has seen a higher proposal

No responding acceptor has accepted a previous proposal ✓

Majority responding acceptors have accepted some previous proposal

At least one acceptor has accepted some proposal but no majority value

Prepared: No Known Majority Acceptance

Proposer
(prepared(p, v^p))

Proposer sends its proposal and value if majority acceptors have not yet accepted any value

1 and 2 can both find this condition and propose their values v^{p1} and v^{p2}, p2 > p1

A majority of acceptors could get and accept v^{p1} first then v^{p2}

Non Atomic Get and Set Prepare

Proposer
(pre-propose (p, v^p))

A propose message sends a prepare message to query the current state

Proposer
(prepared(p, v^p))

Proposer sends its proposal and value if majority acceptors have not yet accepted any value

Problem arise because get of acceptor state and set of new proposal number and propose value in acceptor not done atomically

Cannot do all three atomically as result of get affects proposed action (need two proposer phases)

Can do get of acceptor state and set new proposal number atomically

Atomic Get and Set Prepare

Proposer
(pre-propose (p, v^p))

A propose message sends a prepare message *with proposal number* to query the current state and set its proposal number

Accepter (prepare)

Each acceptor sends back last accepted value and its proposal number (or optionally last seen proposal number if no acceptance so far) and updates last seen proposal number

Prepare Fixed

Proposer
(prepared(p, v^p))

Proposer sends its proposal and value if majority acceptors have not yet accepted any value

1 and 2 can both find this condition and propose their values v^{p1} and v^{p2}, p2 > p1

A majority of acceptors could get and accept v^{p1} first then v^{p2}

Prepare Cases

Proposer
(prepared(p, v^p))

A propose message sends a prepare message with proposal number to query the current state and set its proposal number

Some acceptor has seen a higher proposal

No responding acceptor has accepted a previous proposal

Majority responding acceptors have accepted some previous proposal ✓

At least one acceptor has accepted some proposal but no majority value in the acceptances majority

Prepared: Known Majority Value

Proposer
(prepared(p, v^p))

Proposer proposes no value if it knows majority has already accepted a previous value

A prepare phase can prevent some nodes from accepting the previous majority value

If some of the nodes in current majority die, some learner nodes may not get consensus value even though majority of nodes are alive and can converge to a value

Want consensus value to propagate to all acceptors for fault tolerance

Prepared: Known Majority Value Fixed

Proposer
(prepared(p, v^p))

Proposer (re) proposes majority value as its own value

1's prepare phase can prevent some nodes from accepting the previous majority value

If some of the nodes in current majority die, some learner nodes may not get consensus value even though majority of nodes are alive

Want consensus value to propagate to all acceptors for fault tolerance

Prepared Cases

Proposer
(prepared(p, v^p))

A propose message sends a prepare message with proposal number to query the current state and set its proposal number

Some acceptor has seen a higher proposal

No responding acceptor has accepted a previous proposal

Majority responding acceptors have accepted some previous proposal

At least one acceptor has accepted some proposal but no majority value in the acceptances majority ✓

Prepared: Known Non Majority Value

Proposer
(prepared(v^p))

Proposer (re) proposes value of highest accept proposal number as its own value

The value with highest proposal number may or may not be or become majority value (depending on whether there is some other unknown proposal in the system)

If it does become majority value then prepare phase may have locked some nodes from accepting it

Prepared Cases

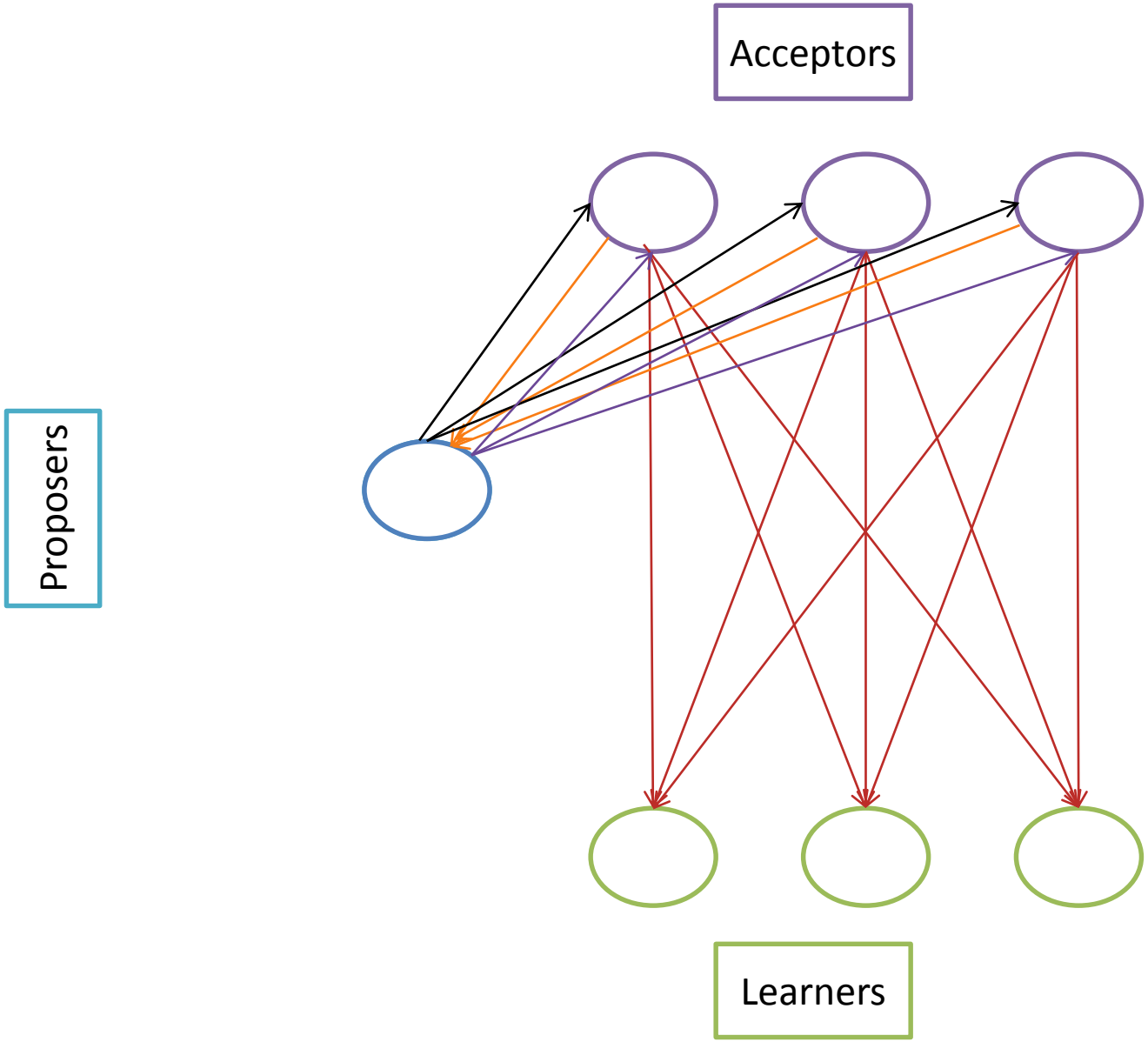
Proposer
(prepared(v^p))

Proposer abandons proposal if it learns about a higher number proposal

Proposer sends its proposal and value if majority acceptors have not yet accepted any value

Proposer (re) proposes with highest accept proposal number as its own value (which may also be a majority value)

Three-Phase Summary



Phase 1 Summary

Proposer
(pre-propose (p, v^p))

A propose message sends a prepare message with proposal number to query the current state and set its proposal number

Accepter (prepare)

Each acceptor sends back last accepted value and its proposal number (or optionally last seen proposal number if no acceptance so far)
and updates last seen proposal number

Phase 2 Summary

Proposer
(prepared(v^p))

Proposer abandons proposal if it learns about a higher number proposal

Proposer sends its proposal and value if majority acceptors have not yet accepted any value

Proposer (re) proposes with highest accept proposal number as its own value (which may also be a majority value in majority acceptors)

Accepter (accept)

Each acceptor accepts a proposal if its proposal number is higher than what it has seen so far.

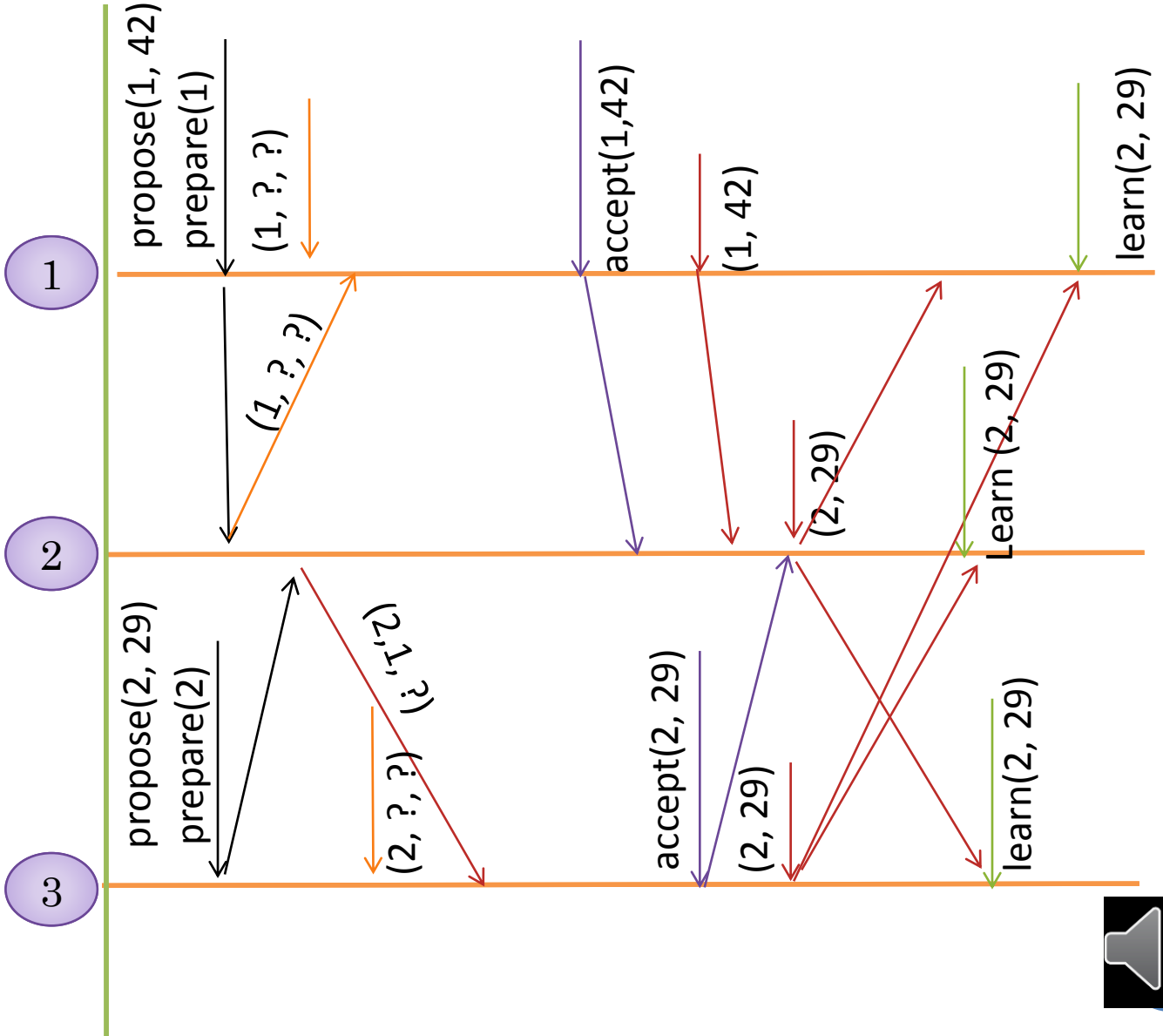
Phase 3

Learner (accepted)

A learner picks the first proposal that has majority acceptances

Prepare-Prepare--Accept-Accept-Override

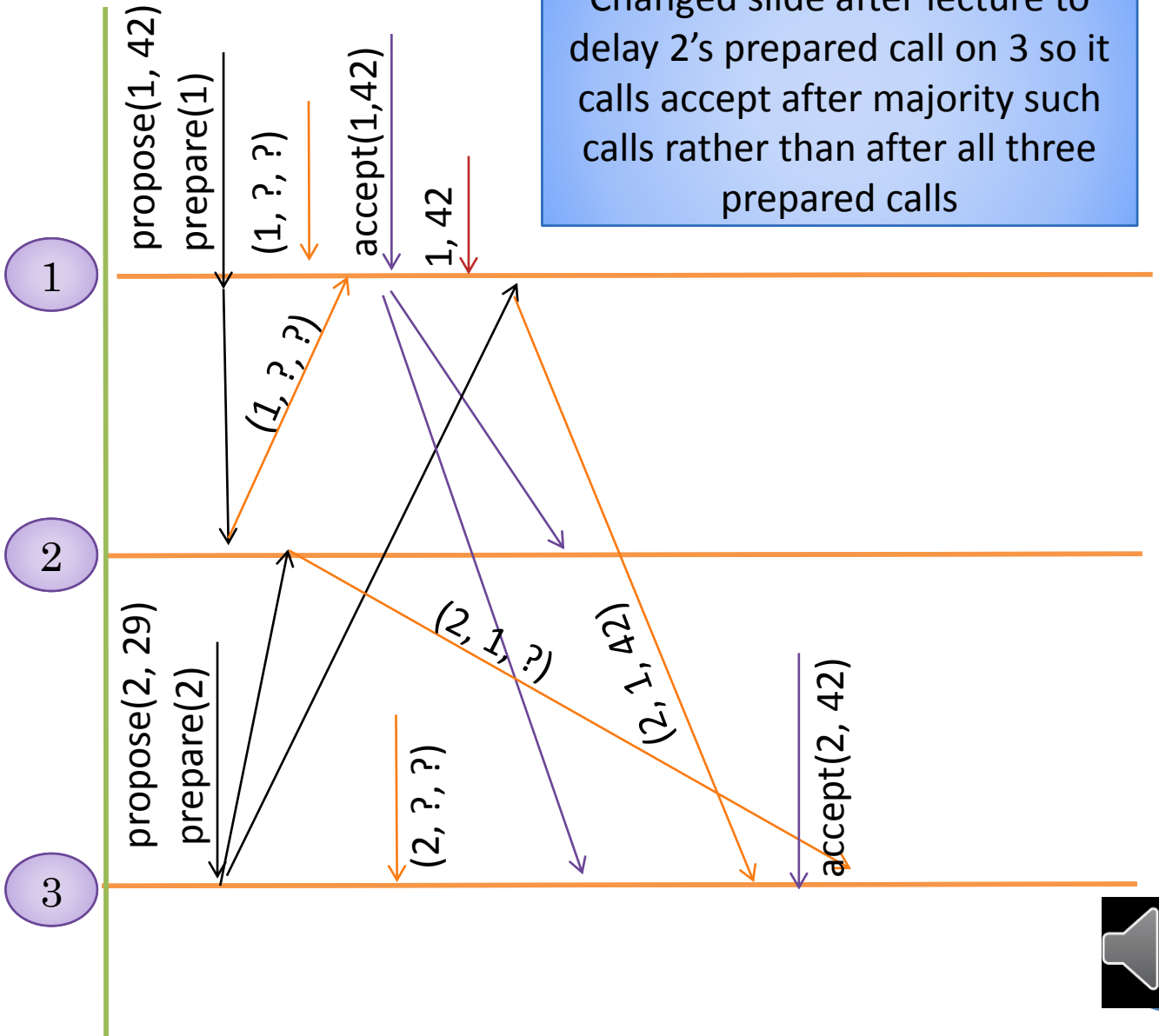
Value	Proposal
42	1
29	2
29	2



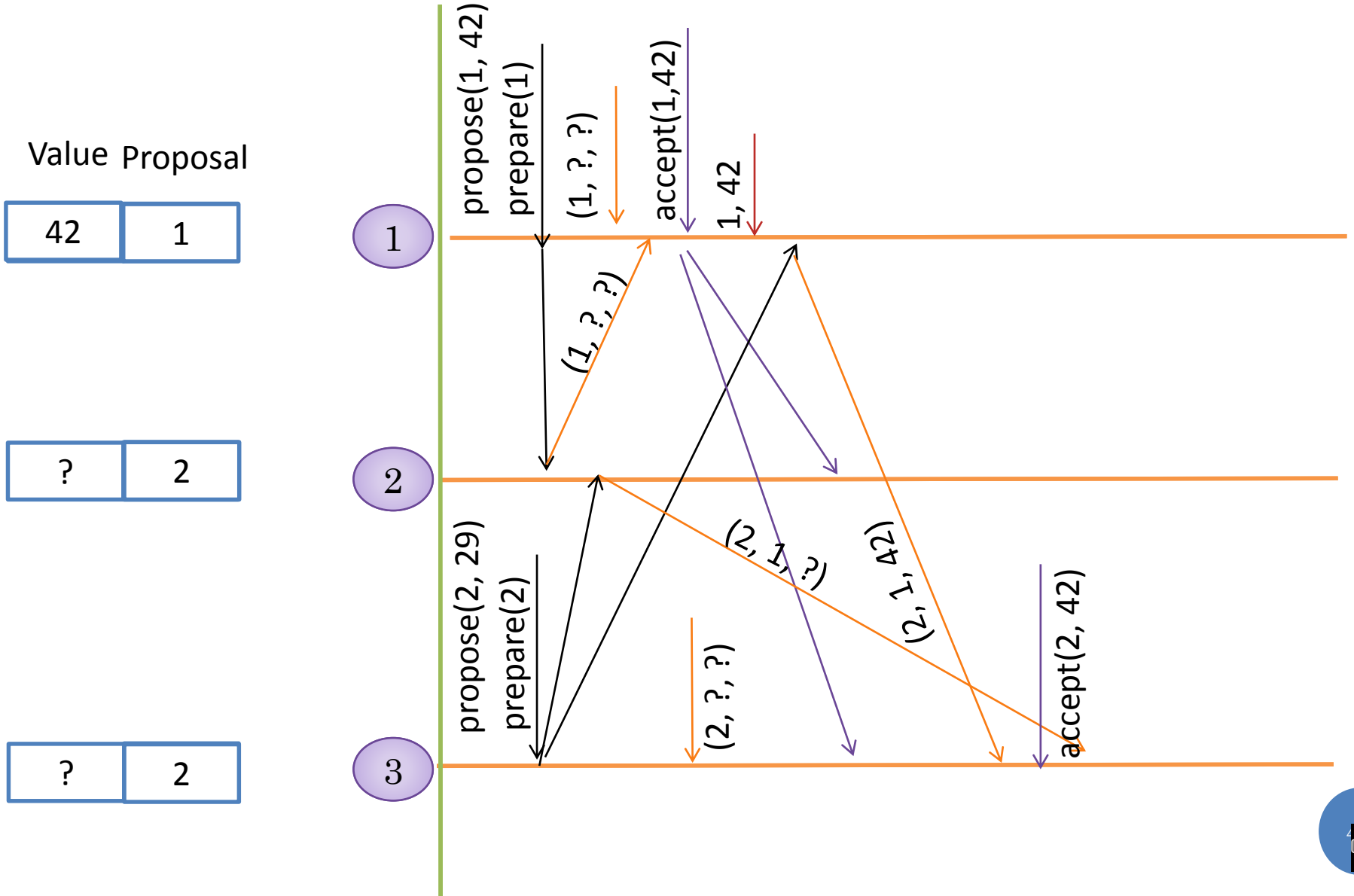
Prepare-Accept-Prepare-Accept- Re-propose

Changed slide after lecture to delay 2's prepared call on 3 so it calls accept after majority such calls rather than after all three prepared calls

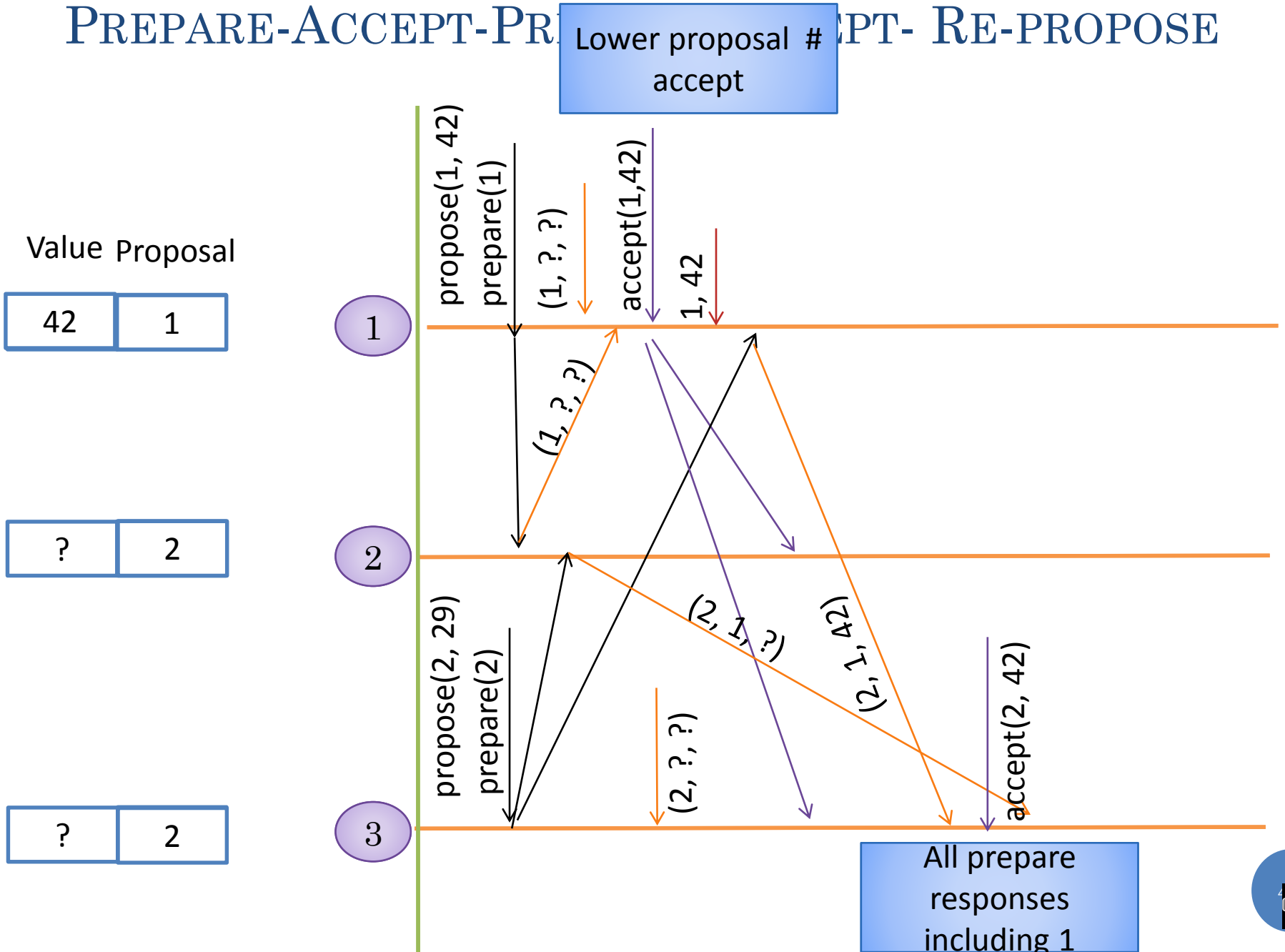
Value	Proposal
42	1
?	2
?	2



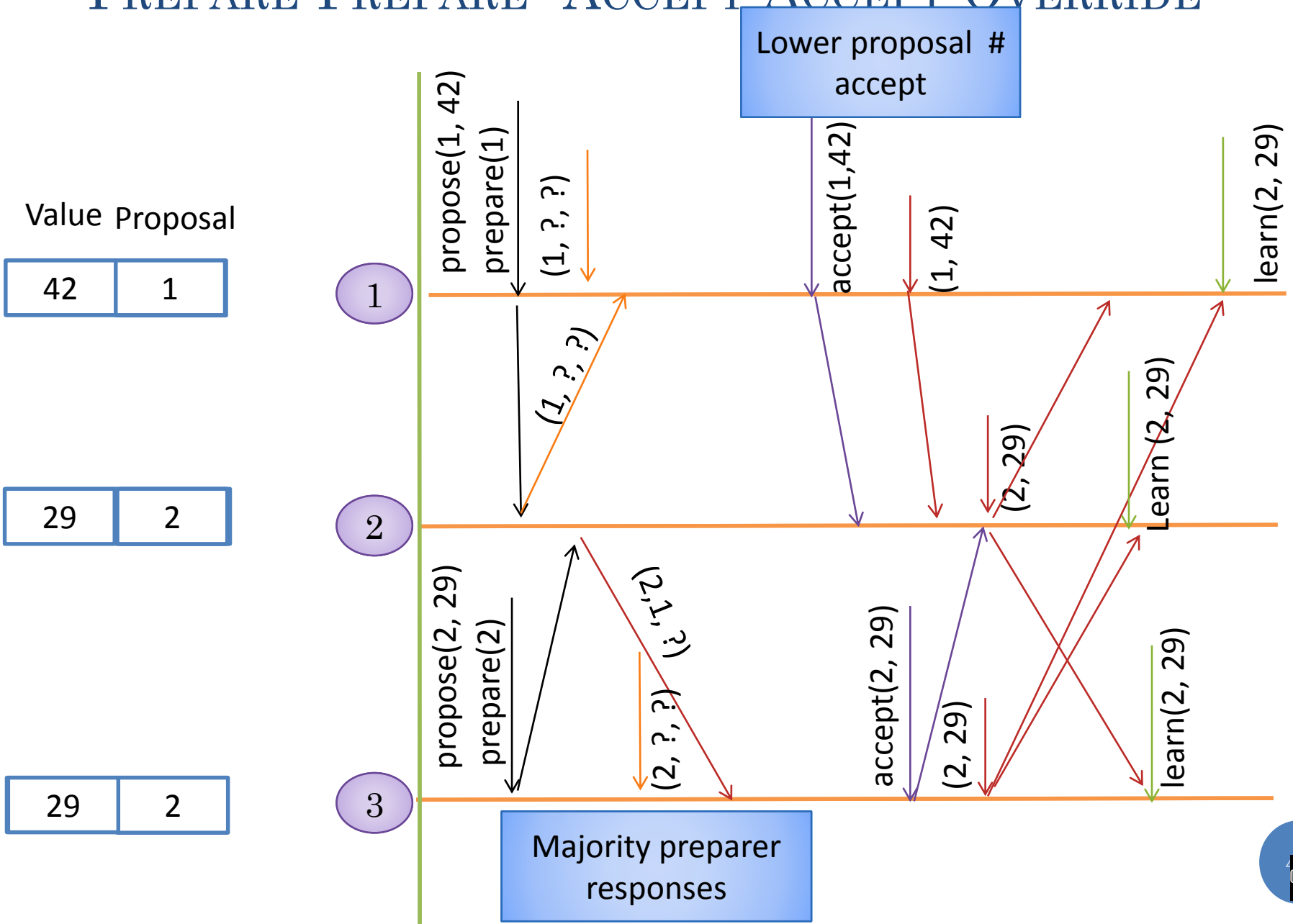
Prepare-Accept-Prepare-Accept- Re-propose



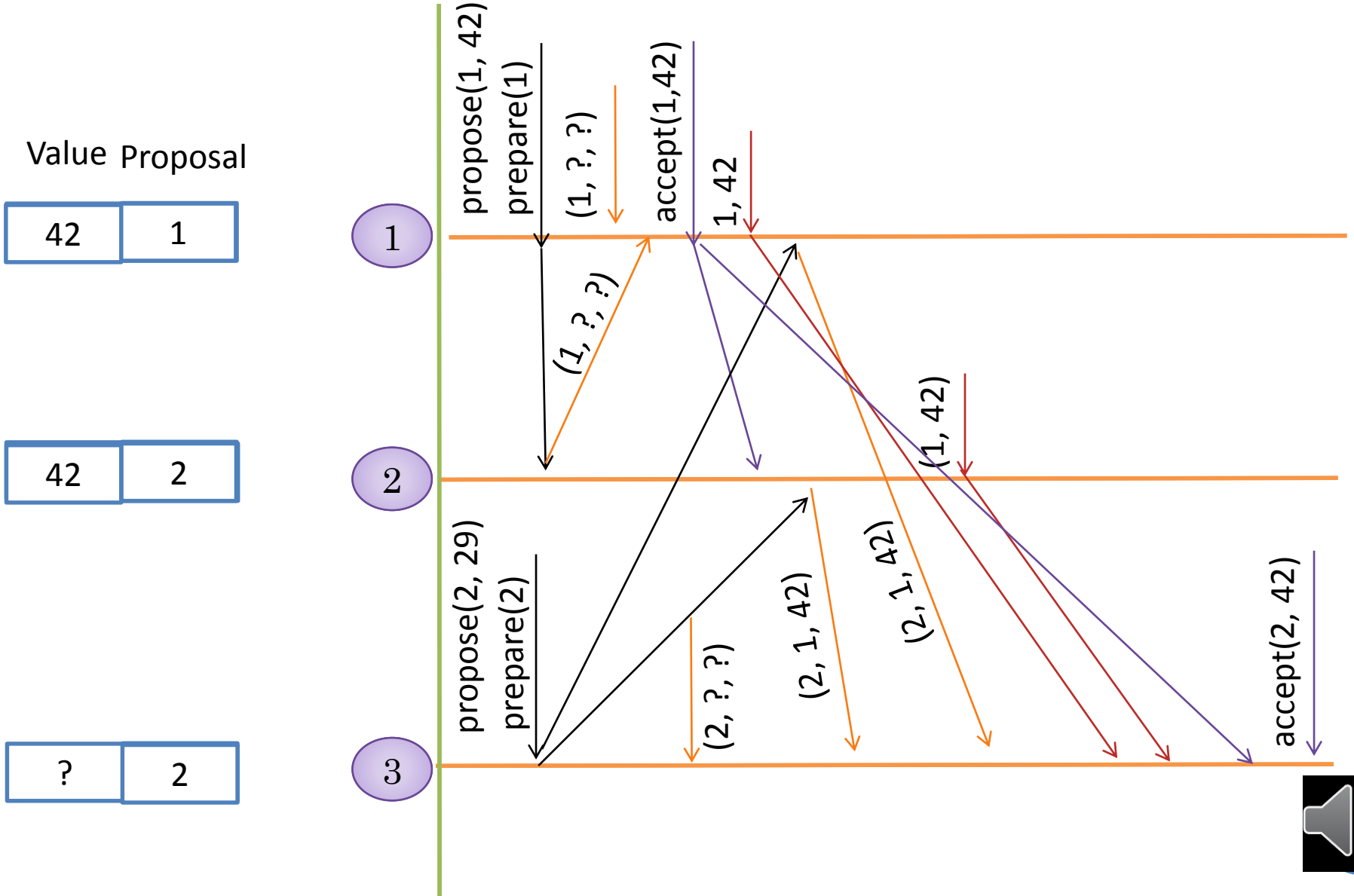
Prepare-Accept-Pr[illegible]ept- Re-propose



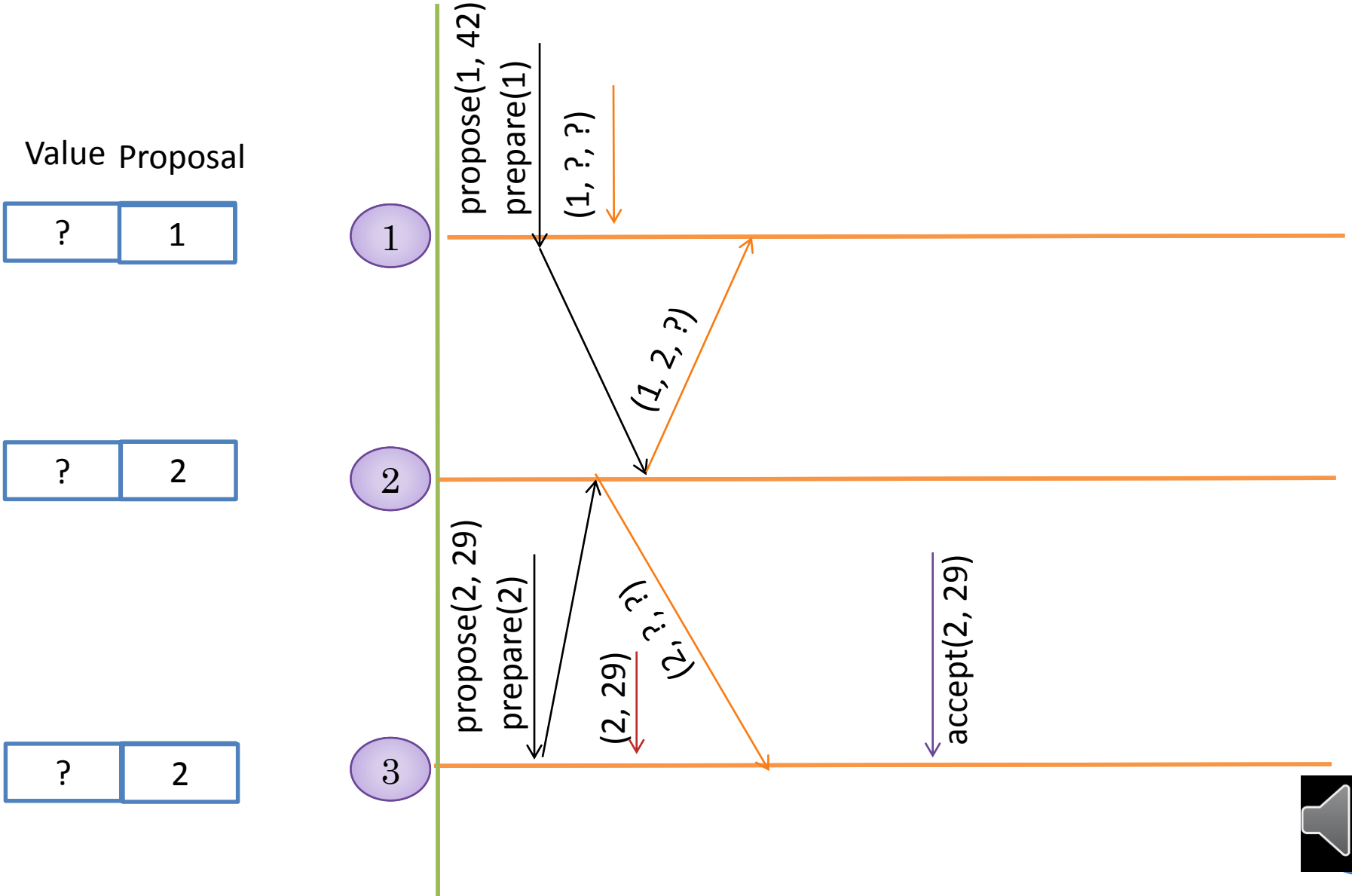
Prepare-Prepare--Accept-Accept-Override



Prepare-Accept-Prepare-Accept- Fault

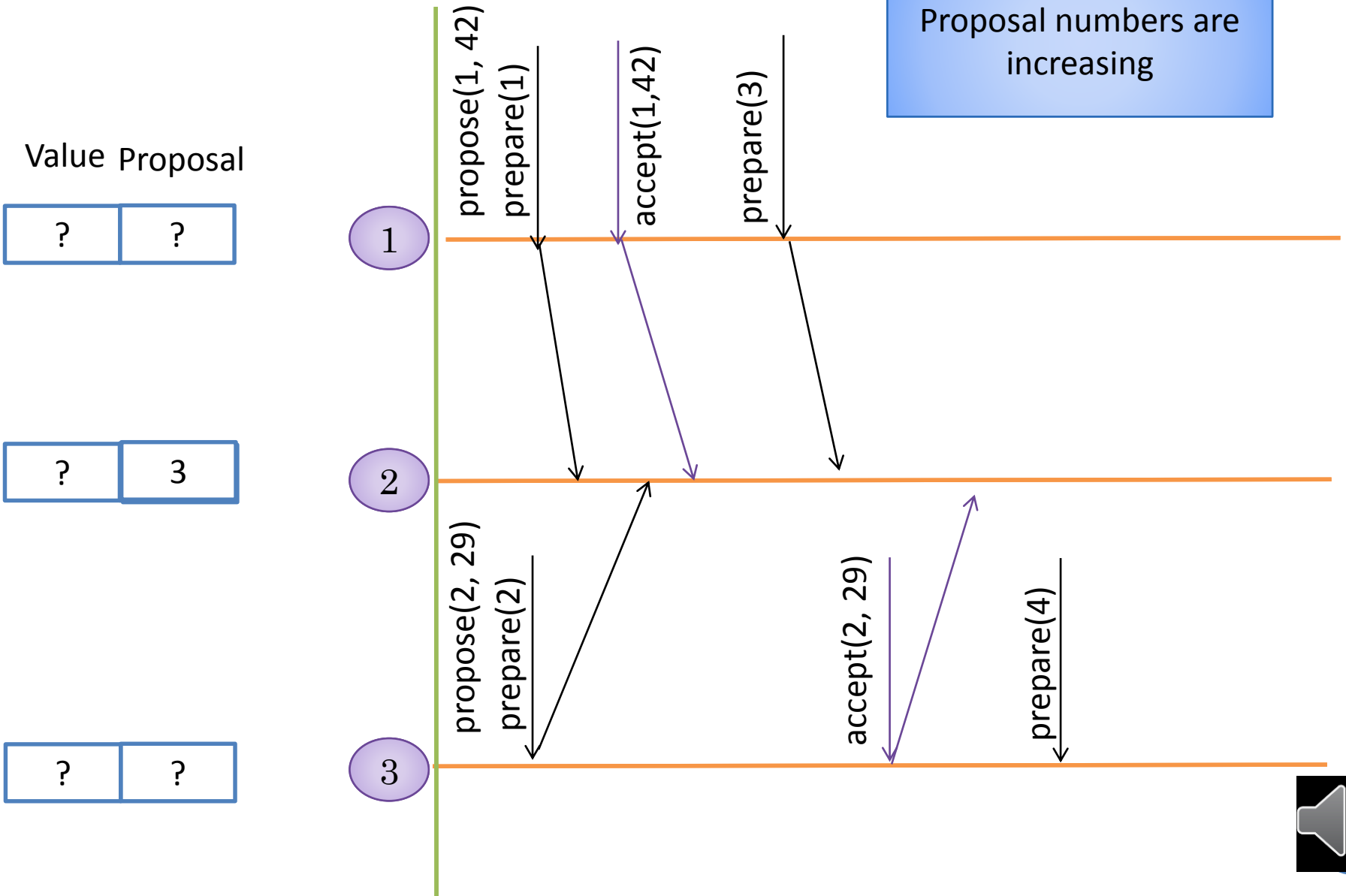


Prepare-No-Accept



Livelock

Proposal numbers are increasing



Disjoint Proposal Ids

Each replica could be initialized with different sets of ids at initialization time

Id of proposal made by a replica might be smaller than the ids of proposals it has already seen

Re-proposals may not be accepted

Increment #Proposals

Each replica keeps track of # known proposals.

Id of new proposal is one more than # known proposals

Two proposals made by different replicas can have same id

Id. #Proposals+1

Each replica keeps track of # known proposals.

Id of new proposal is (siteId.#Proposals + 1)

All proposals made by a site will be smaller or larger than all proposals of another site

#Proposals+1.ID

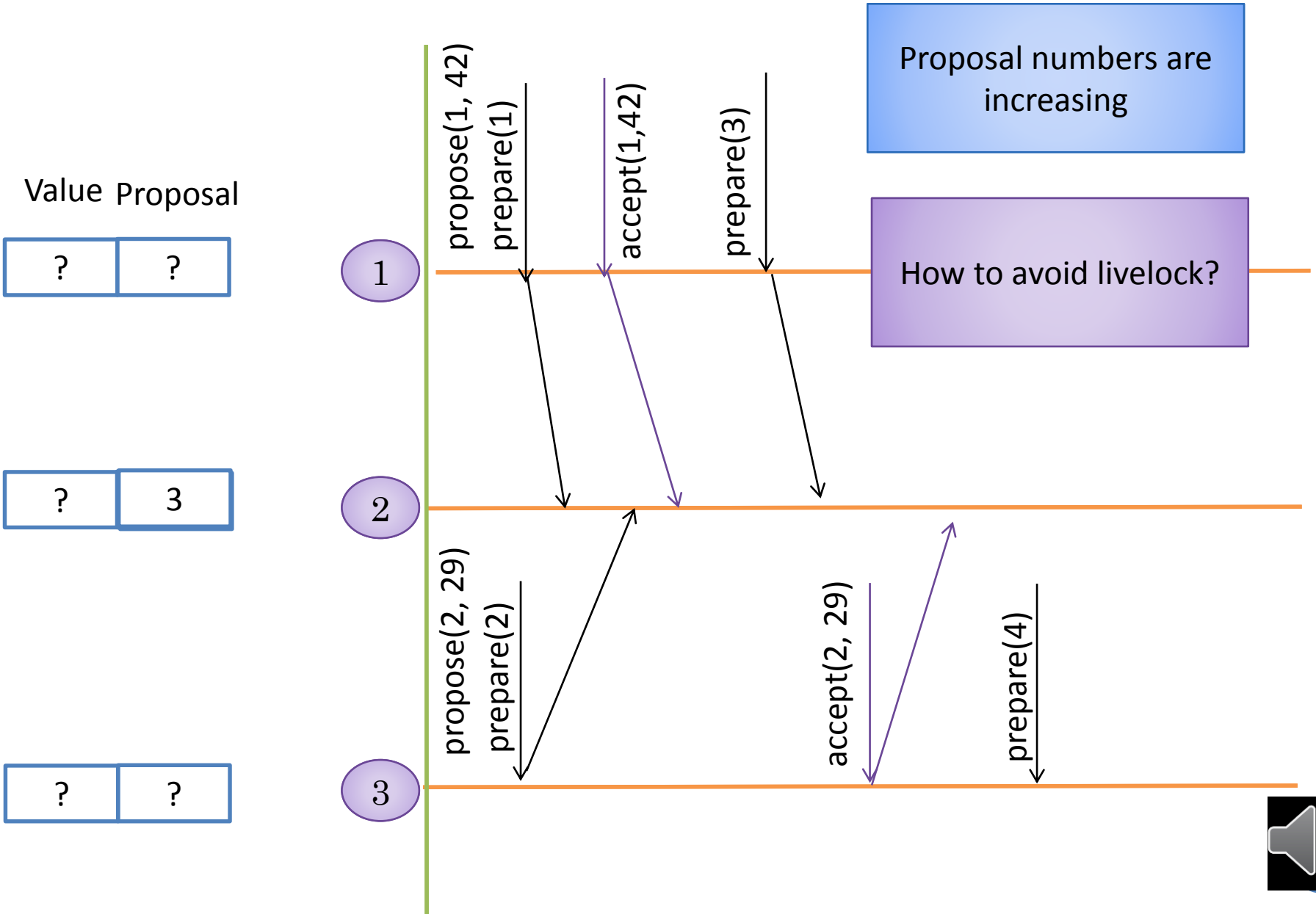
Each replica keeps track of # known proposals.

Id of new proposal is Proposals + 1.siteid

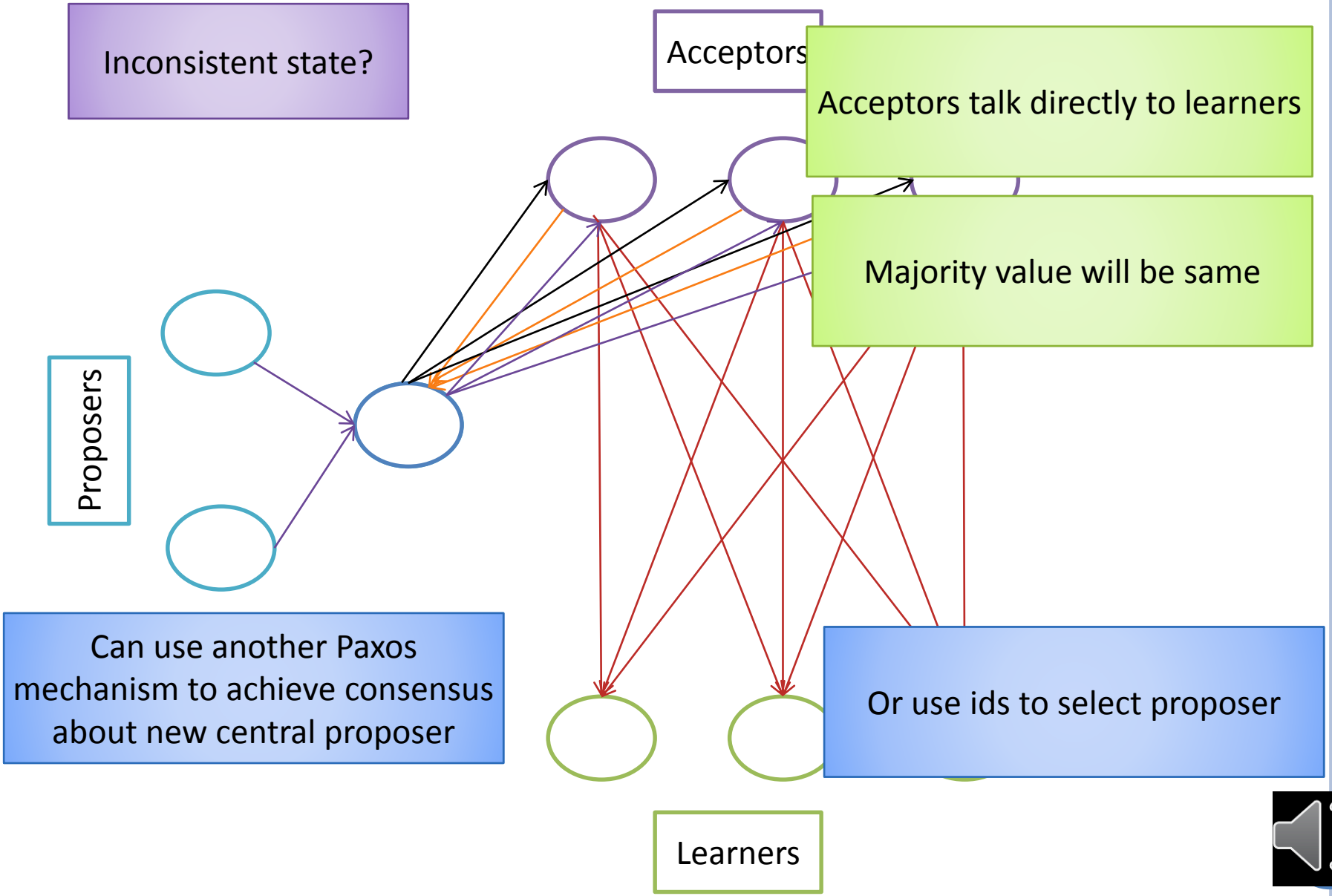
Id of proposal made by a replica will be unique and larger than the ids of proposals it has already seen

```
protected float getAndSetNextProposalNumber(StateType aProposal) {
  lastProposalNumber = wholePart(lastProposalNumber) + 1 + myPrefix;
}
```

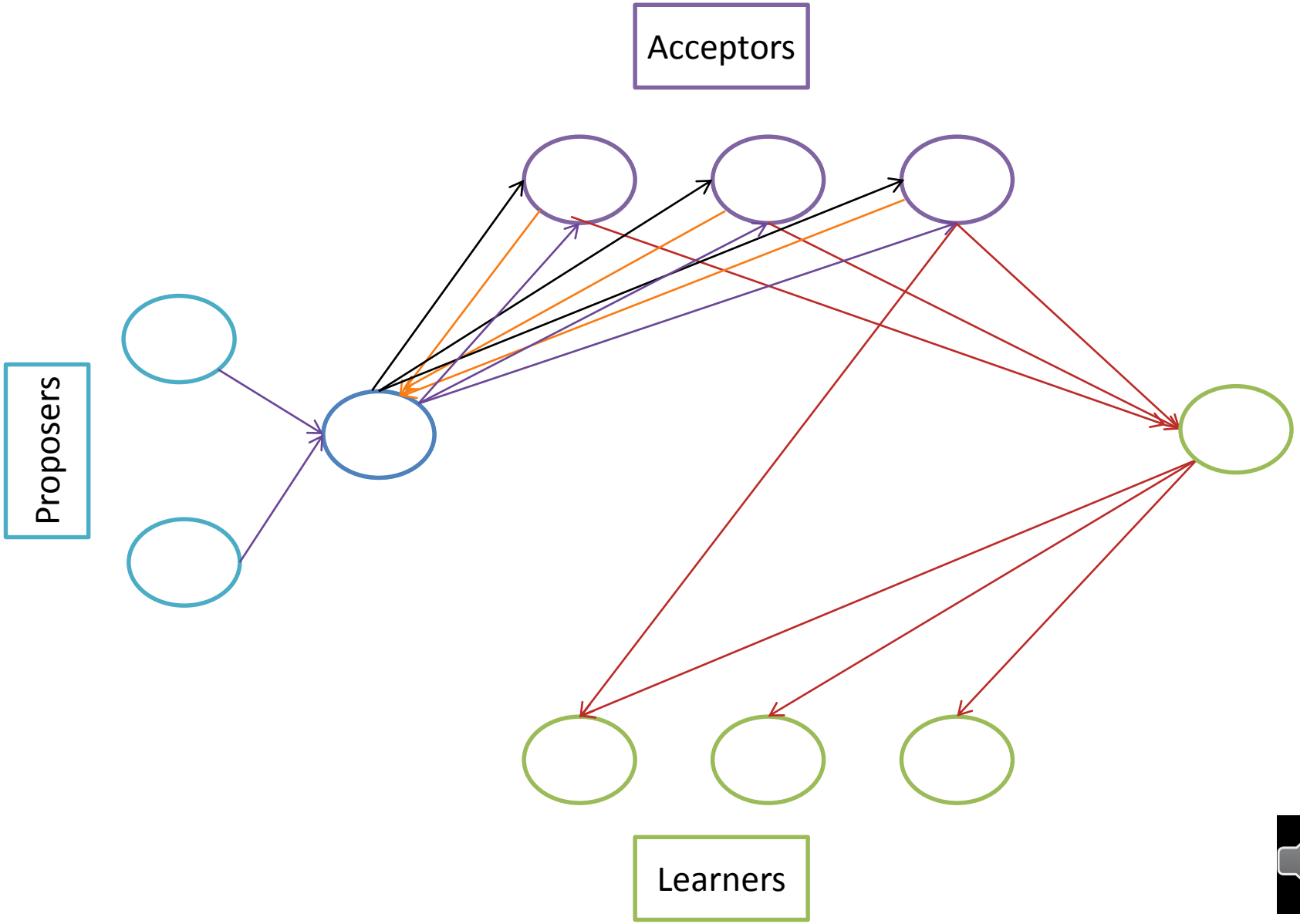
Livelock


Value
Proposal
?
?
?
3
?
?
1
2
3
propose(1, 42)
prepare(1)
accept(1,42)
prepare(3)
propose(2, 29)
prepare(2)
accept(2, 29)
prepare(4)
Proposal numbers are increasing
How to avoid livelock?

Central Proposer to Avoid Livelock



Central Listener



Lamport's Requirements

"P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v. "

"P2[a]. If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v. "

"P2[c]. For any v and n, if a proposal with value v and number n is issued, then there is a set S consisting of a majority of acceptors such that either (a) no acceptor in S has accepted any proposal numbered less than n, or (b) v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in S." ("Any set S consisting of a majority of acceptors contains at least one member of C , we can conclude that a proposal numbered n has value v by ensuring that the following invariant is maintained.")

Lamport's Requirements

" P1a. An acceptor can accept a proposal numbered n iff it has not responded to a prepare request having a number greater than n"

Lamport's Algorithm (Verbatim)

- "Phase 1.
  - (a) A proposer selects a proposal number n and sends a prepare request with number n to a majority of acceptors.
  - (b) If an acceptor receives a prepare request with number n greater than that of any prepare request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.
- Phase 2.
  - (a) If the proposer receives a response to its prepare requests (numbered n) from a majority of acceptors, then it sends an accept request to each of those acceptors for a proposal numbered n with a value v, where v is the value of the highest-numbered proposal among the responses, or is any value if the responses reported no proposals.
  - (b) If an acceptor receives an accept request for a proposal numbered n, it accepts the proposal unless it has already responded to a prepare request having a number greater than n"

Lamport's Algorithm (Verbatim)

- "Phase 1.
  - (a) A proposer selects a proposal number n and sends a prepare request with number n to a majority of acceptors.
  - (b) If an acceptor receives a prepare request with number n greater than that of any prepare request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.
- Phase 2.
  - (a) If the proposer receives a response to its prepare requests (numbered n) from a majority of acceptors, then it sends an accept request to each of those acceptors for a proposal numbered n with a value v, where v is the value of the highest-numbered proposal among the responses, or is any value if the responses reported no proposals.
  - (b) If an acceptor receives an accept request for a proposal numbered n, it accepts the proposal unless it has already responded to a prepare request having a number greater than n"