Paxos Made Insanely Simple

Hein Meling

Paxos Talk @ UiS, Norway, January 20th 2014

University of Stavanger

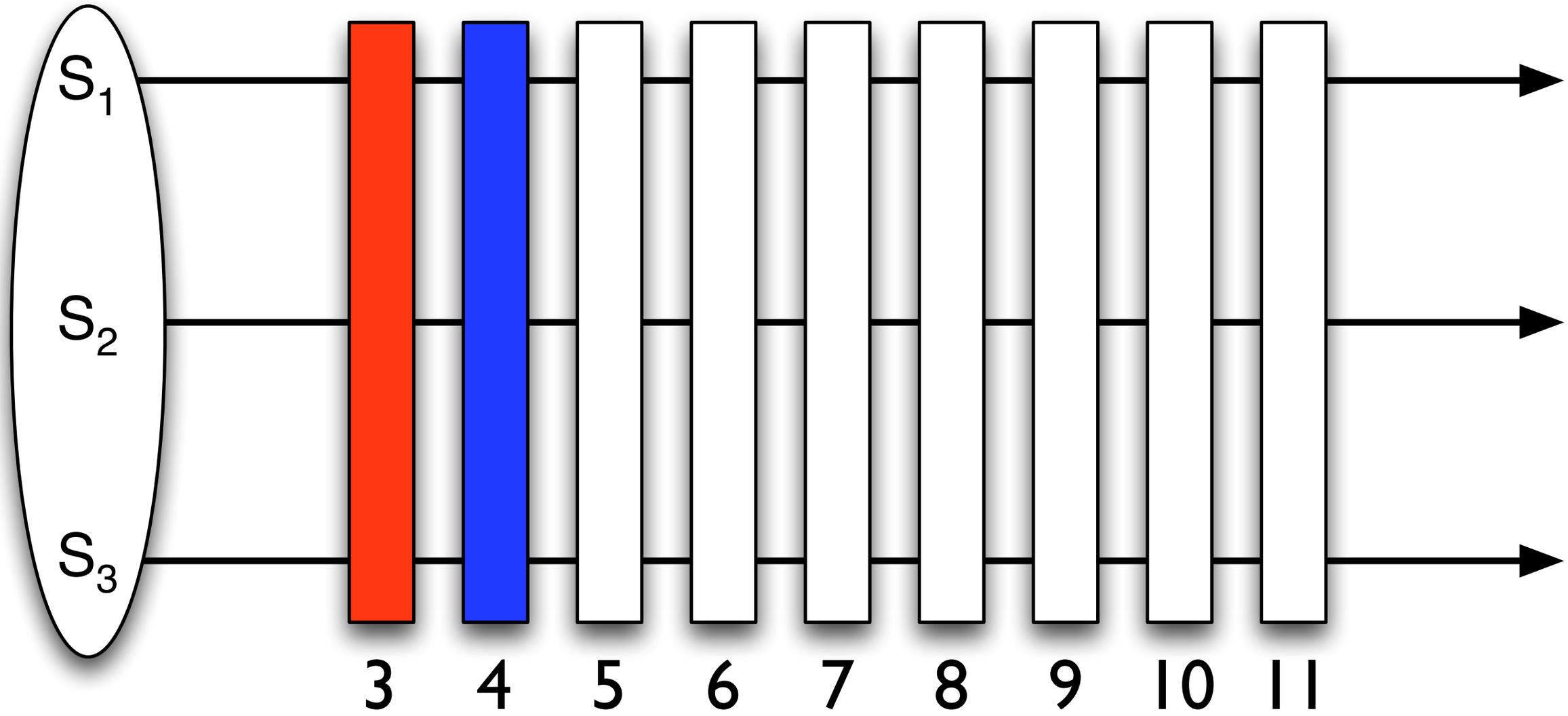
Assumptions

- Unreliable communication
  - May take arbitrarily long to deliver msg
  - Msgs can be duplicated and lost
  - But not corrupted
- Deterministic operations
- Processes may crash, but not behave badly

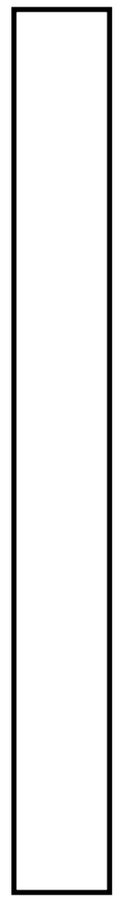
Paxos

(Multi-decree Paxos)

Sequence of Slots

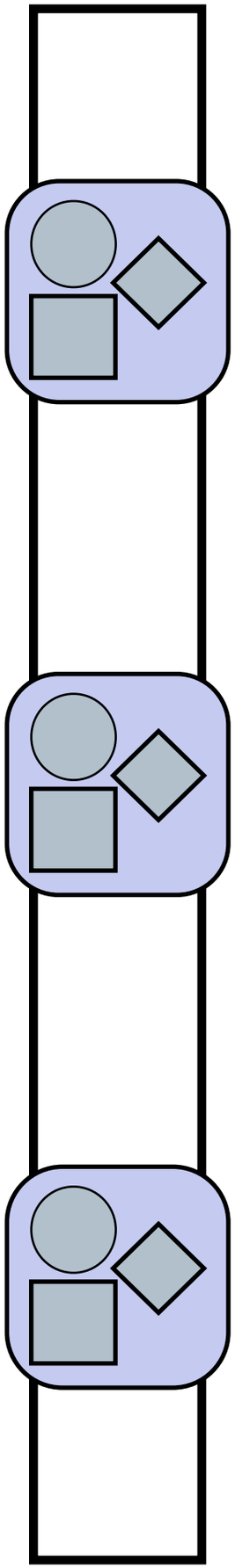


Single-decree Paxos

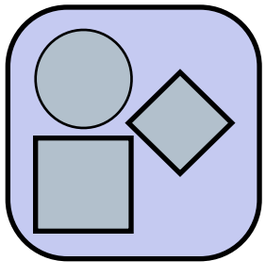


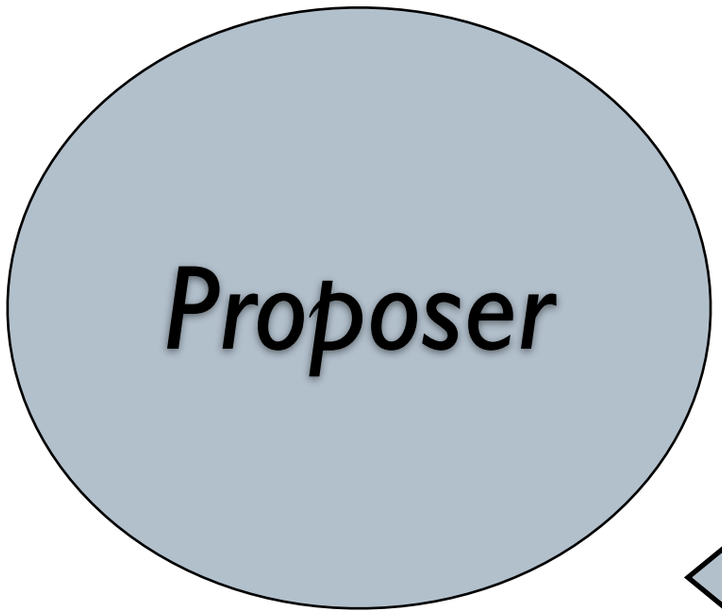
Single-decree Paxos

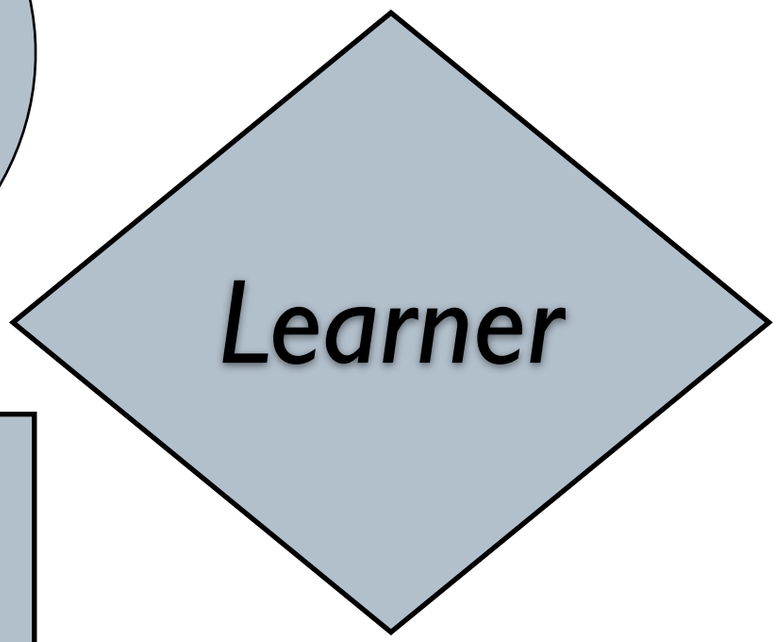
The Server Replicas



Paxos Agents



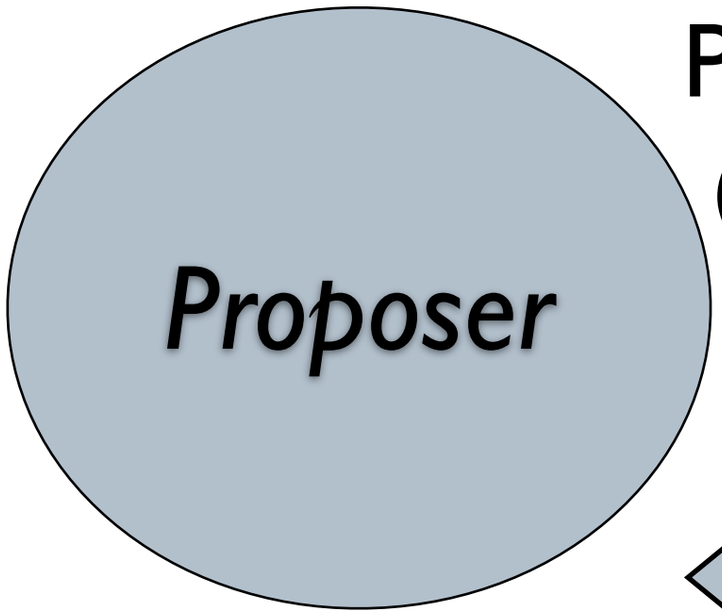


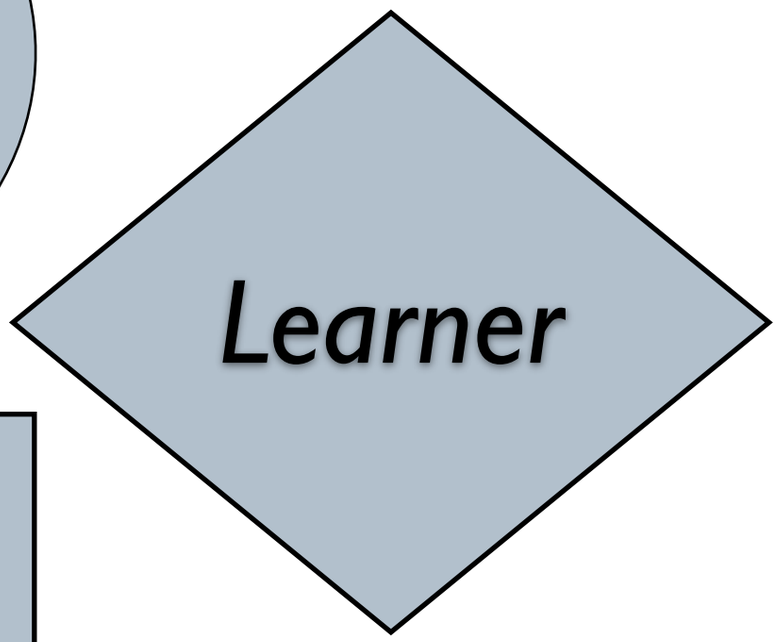


Paxos Agents

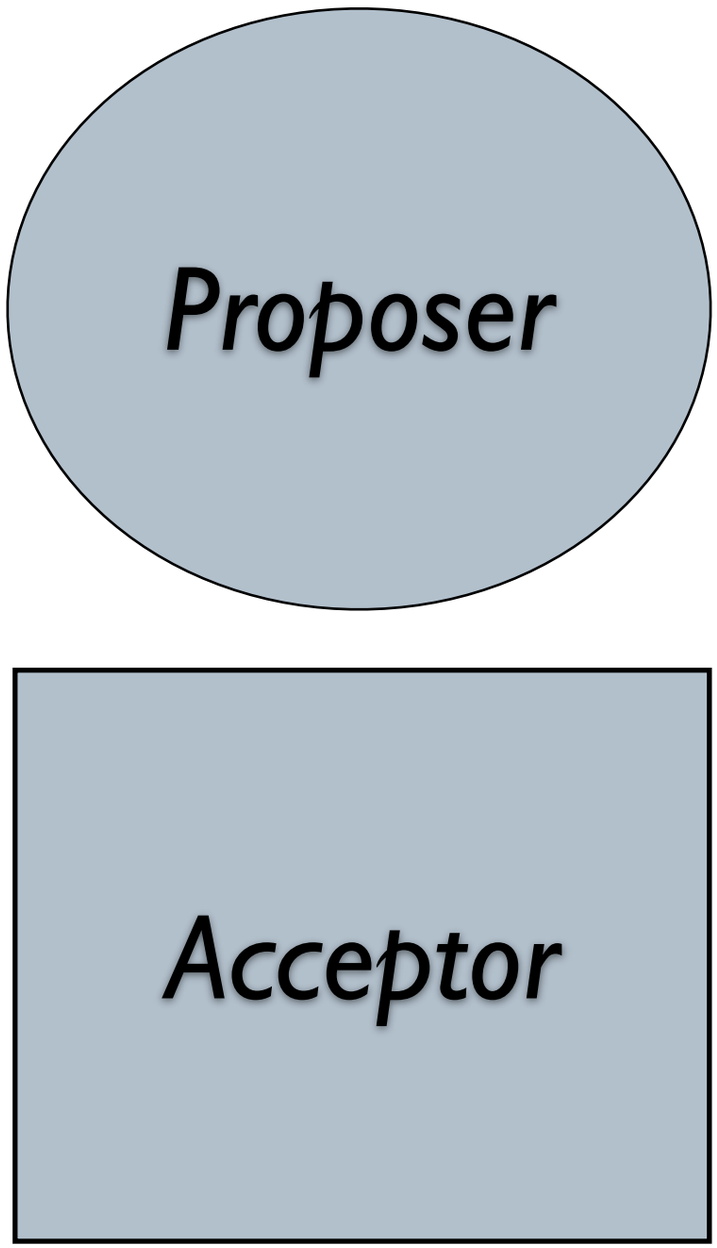
Receive commands from clients

Preliminarily order commands
(propose value for consensus)





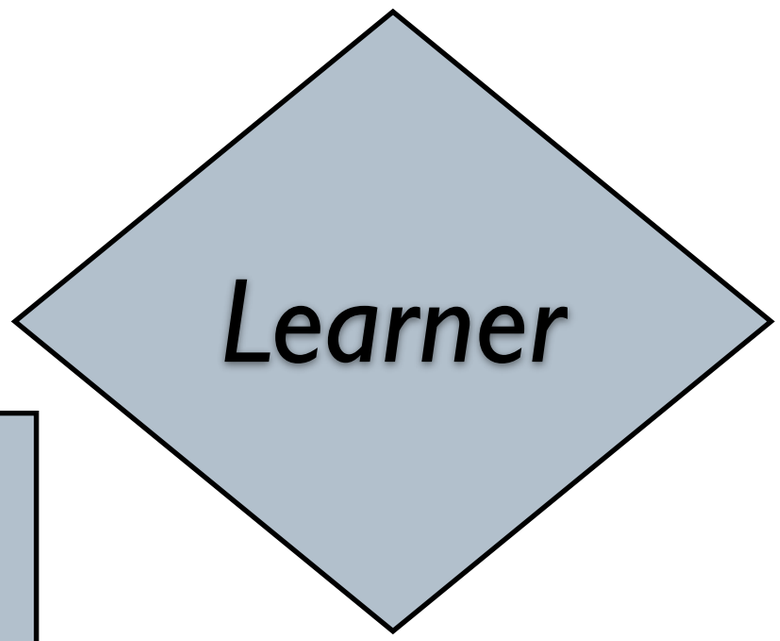
Paxos Agents



Acceptors chooses the consensus value

Paxos Agents

Learners learn the consensus value

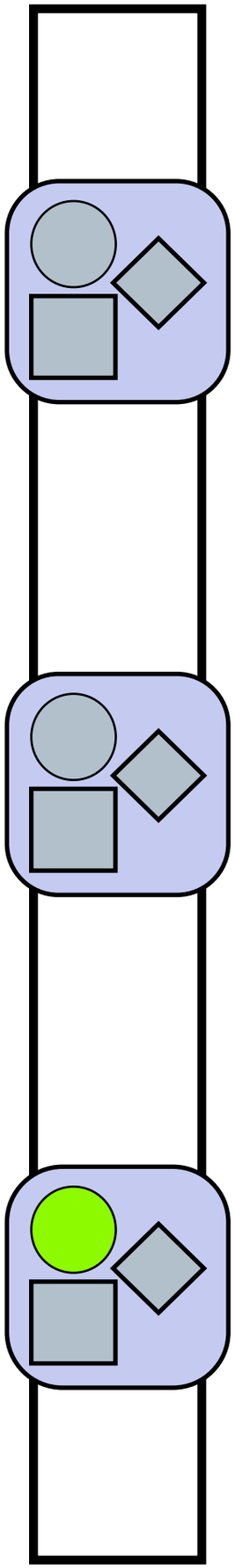


Consensus

- A set of processes tries to choose a common *value*
- The *value* represents a client command

Before we move on!

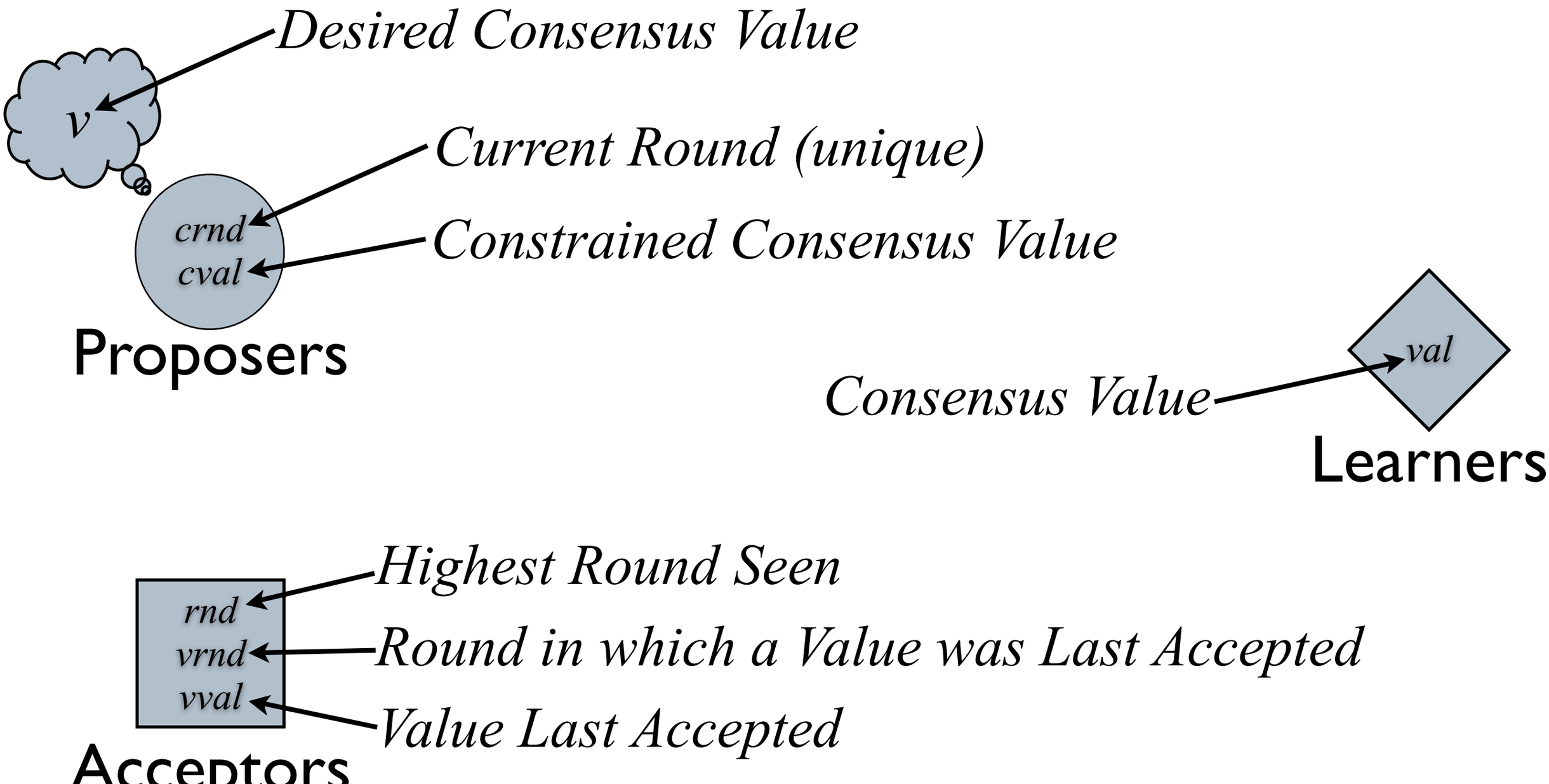
The Proposer



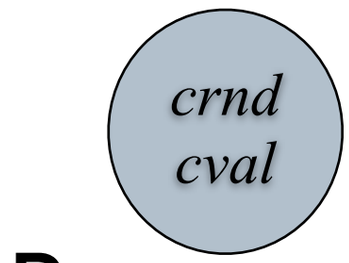
- Common case: there is only one proposer!
- When there is asynchrony we may have
  - Multiple leaders
  - No leader

Leader

Paxos Agents

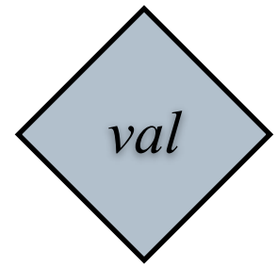


Paxos Agents - The API



Proposers

prepare(*crnd*)
accept(*crnd*,*cval*)



Learners

Acceptors

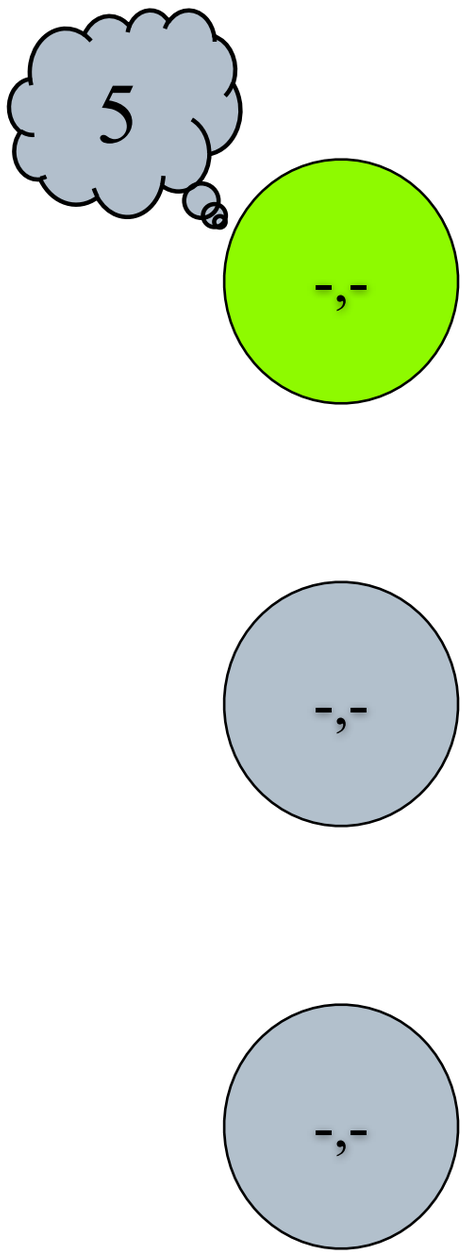
promise(*rnd*,*vrnd*,*vval*)
learn(*rnd*,*vval*)

Paxos Examples

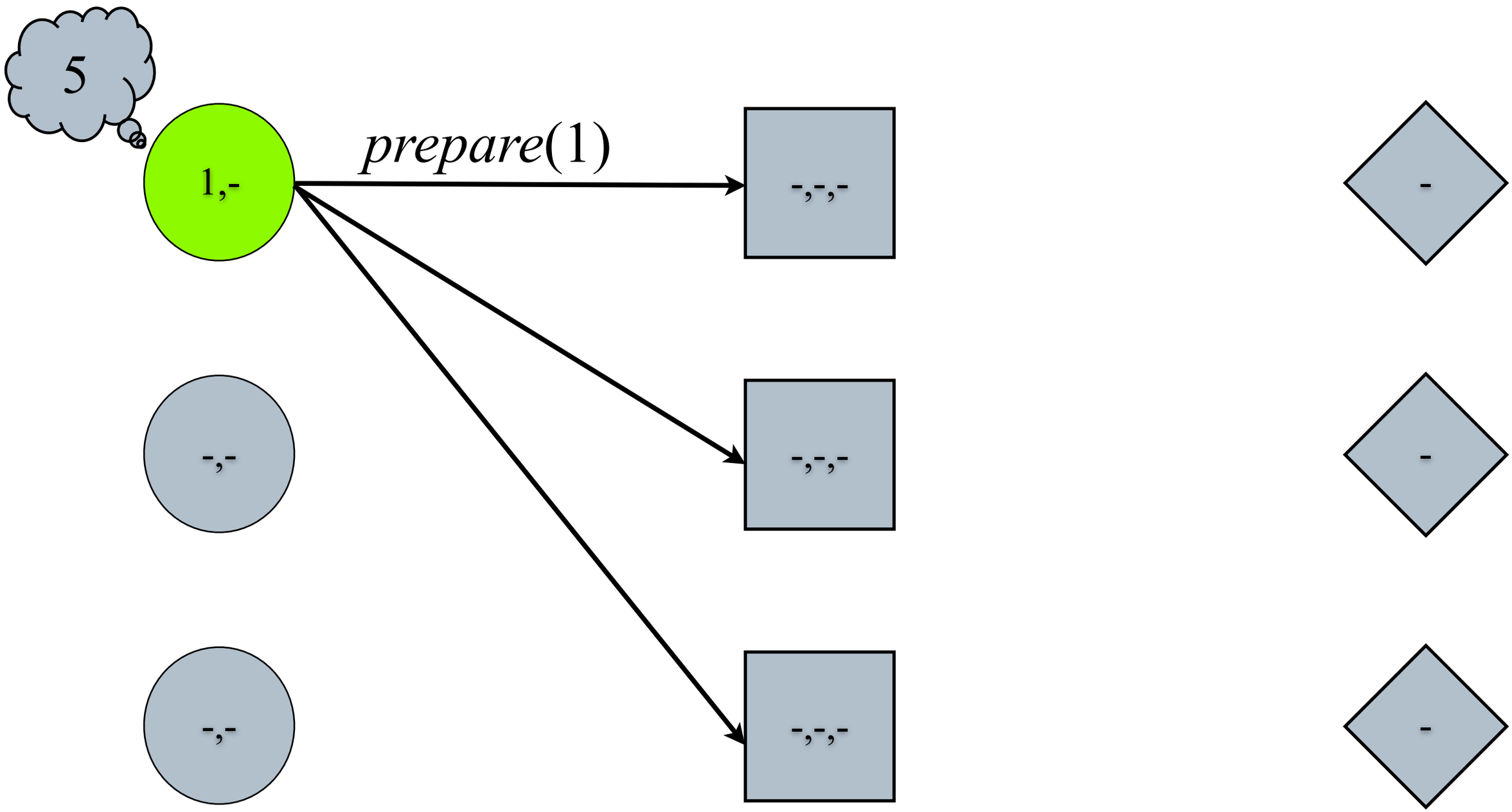
Example 1

A Full Paxos Execution

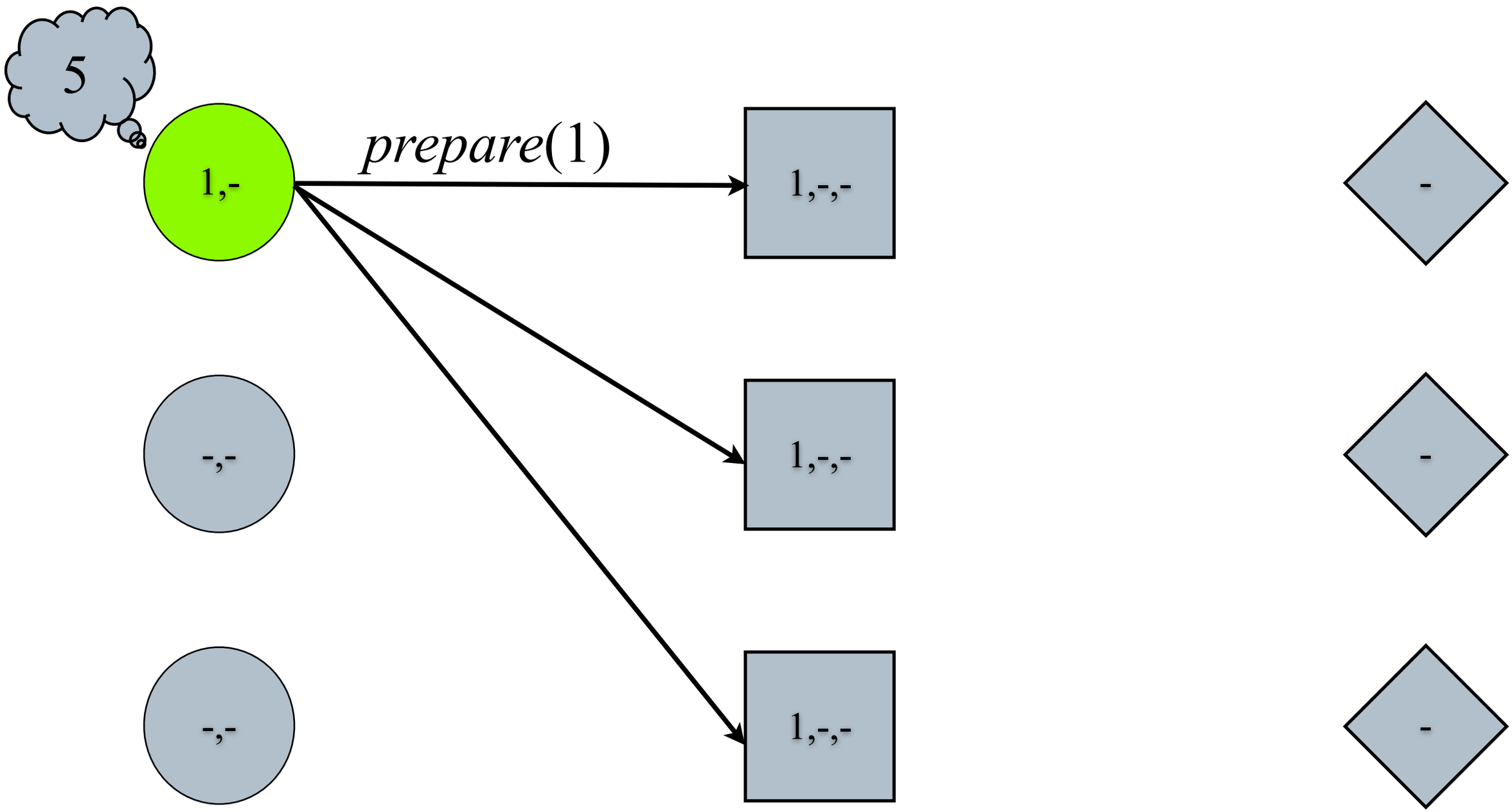
Full Paxos Execution



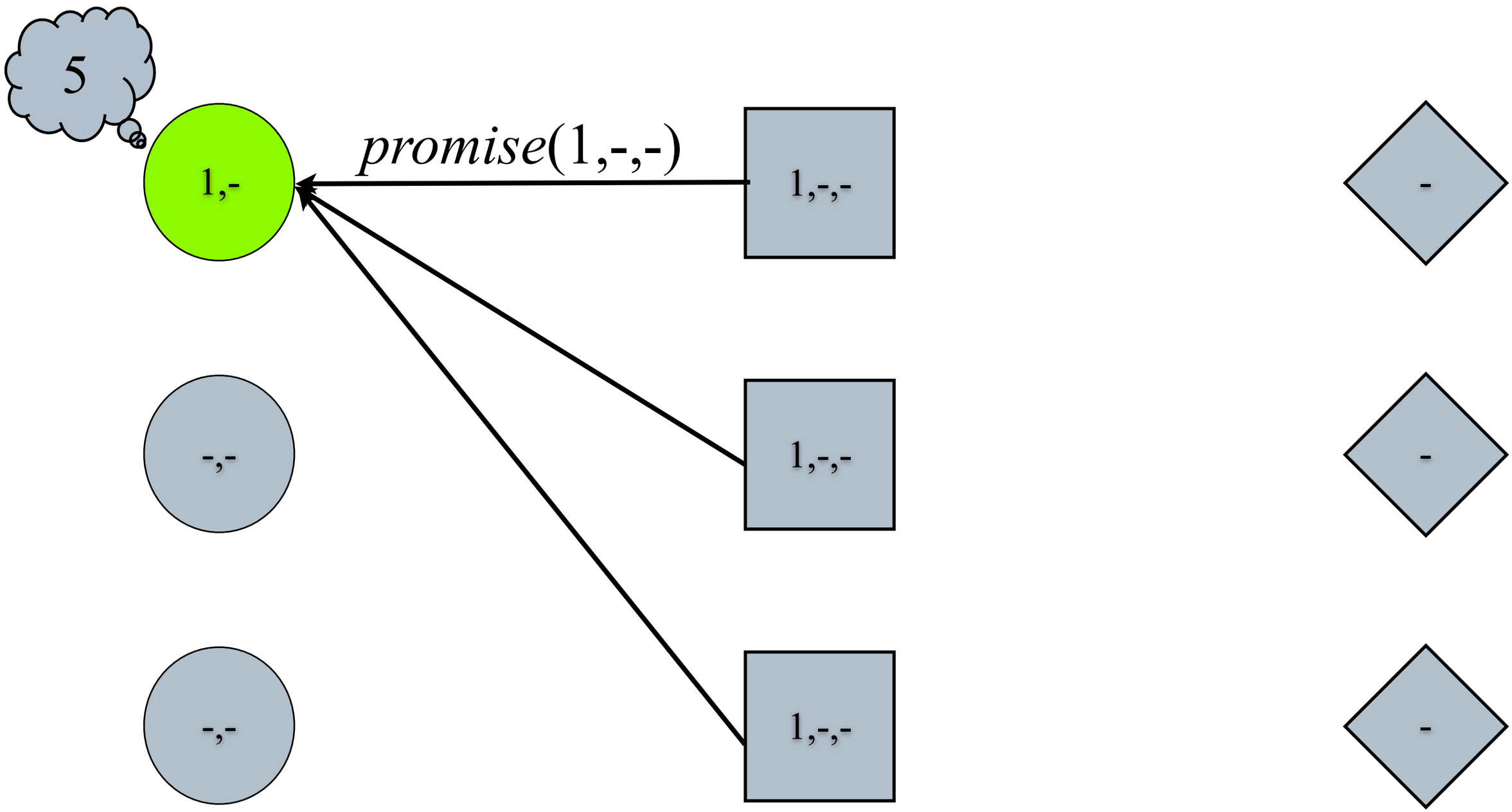
Full Paxos Execution



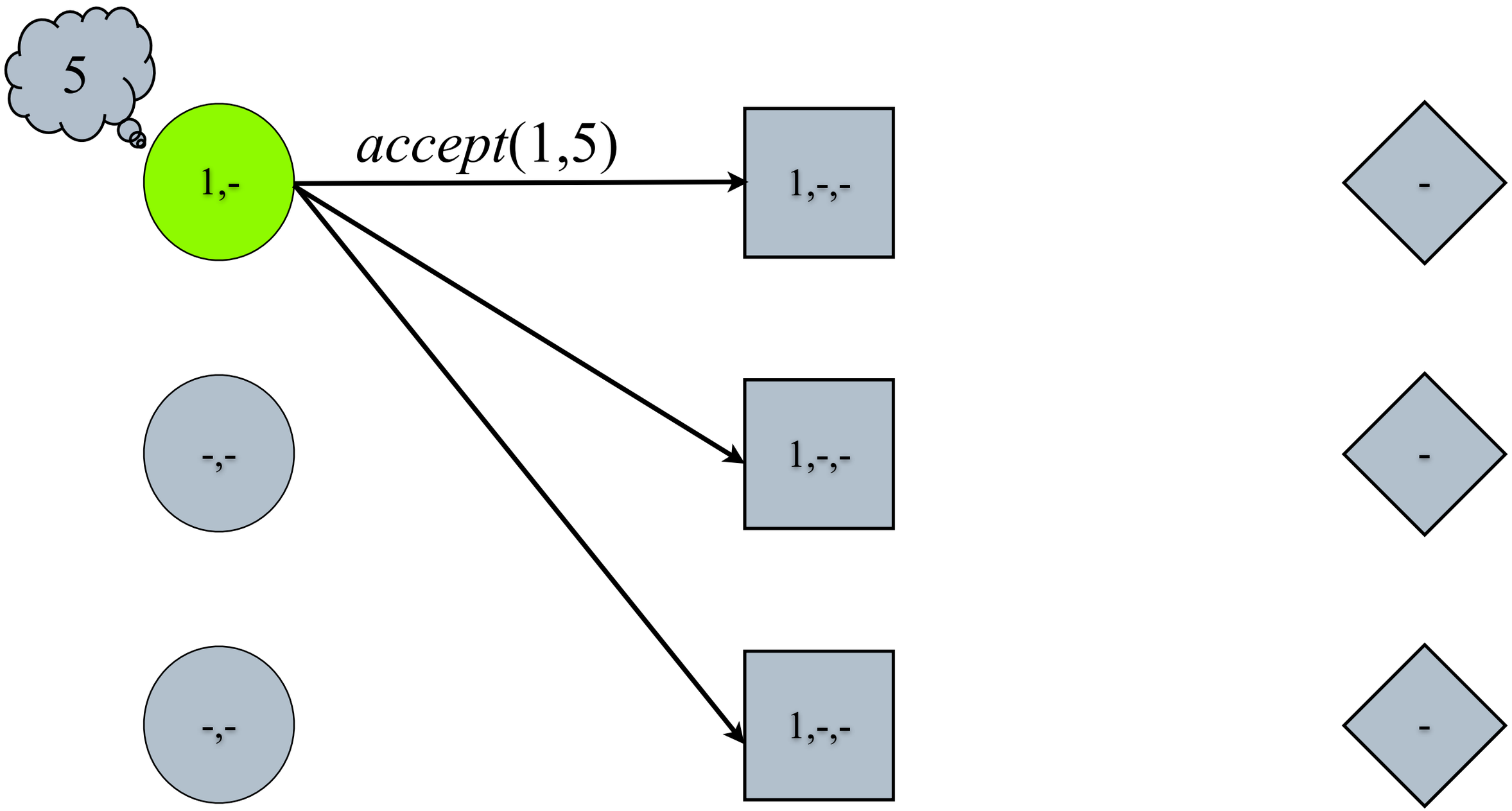
Full Paxos Execution



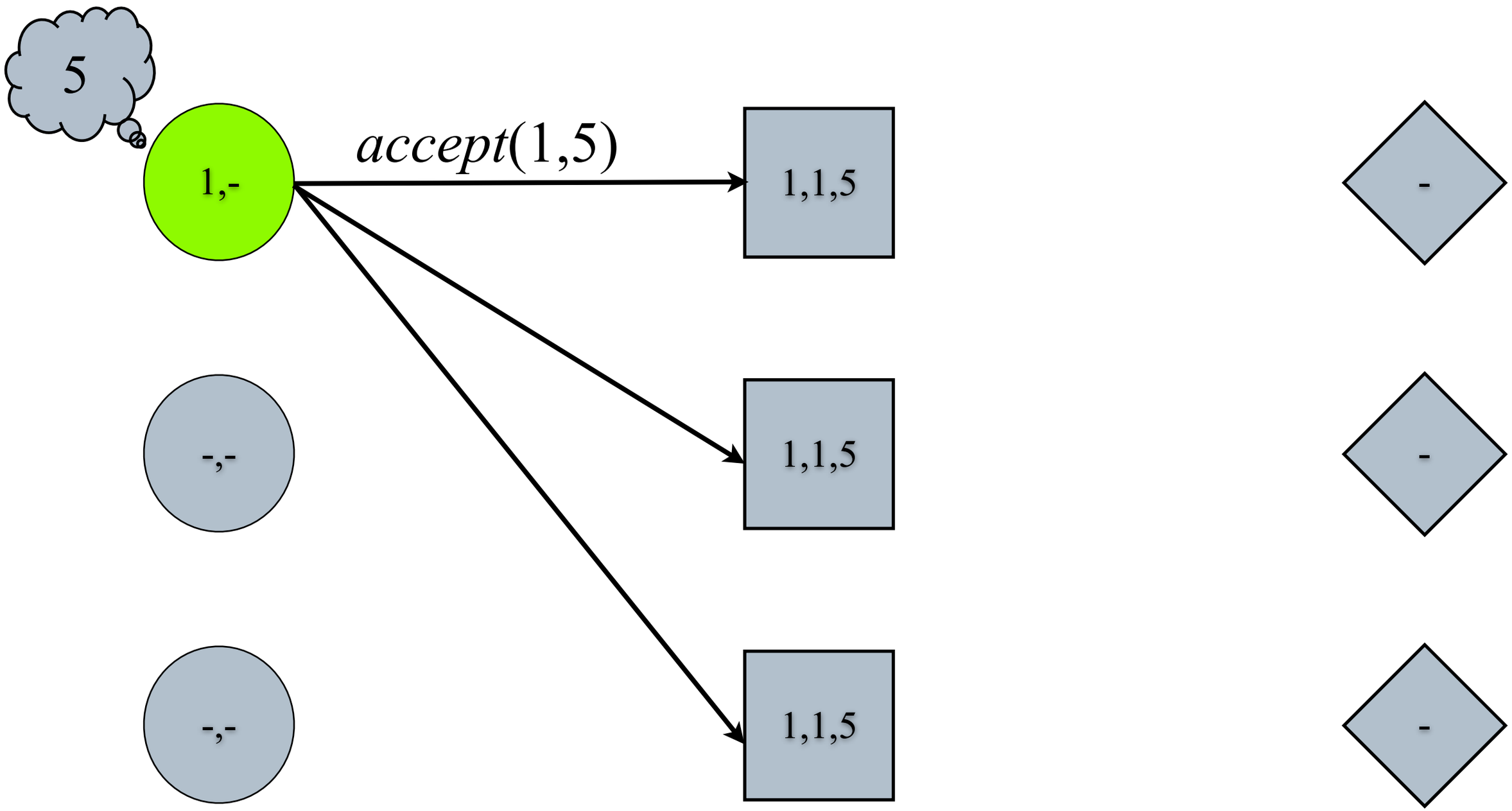
Full Paxos Execution



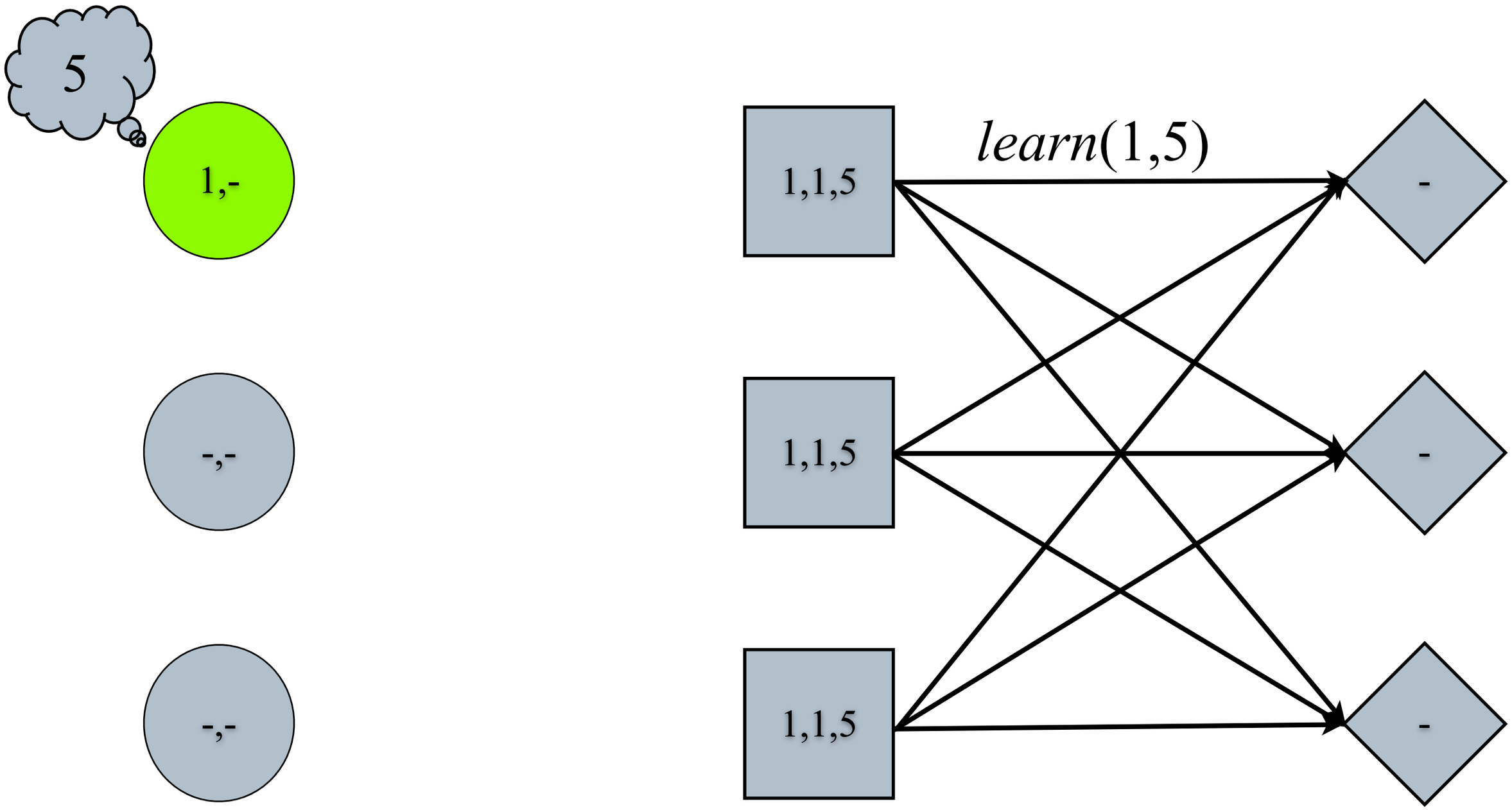
Full Paxos Execution



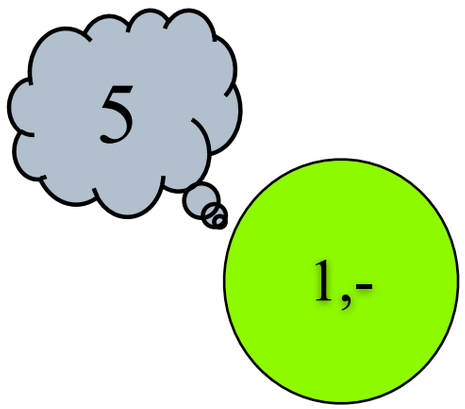
Full Paxos Execution

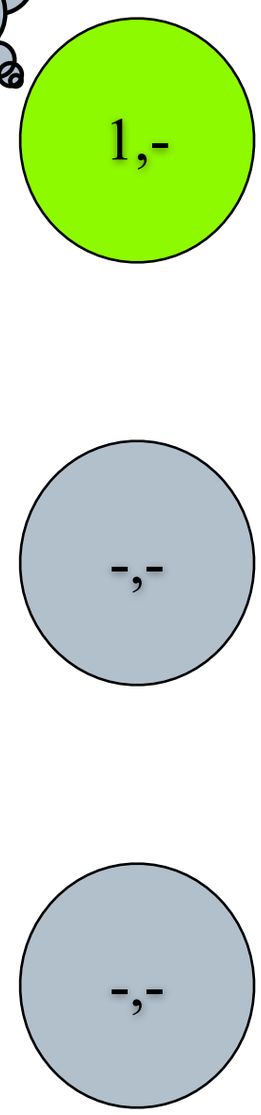


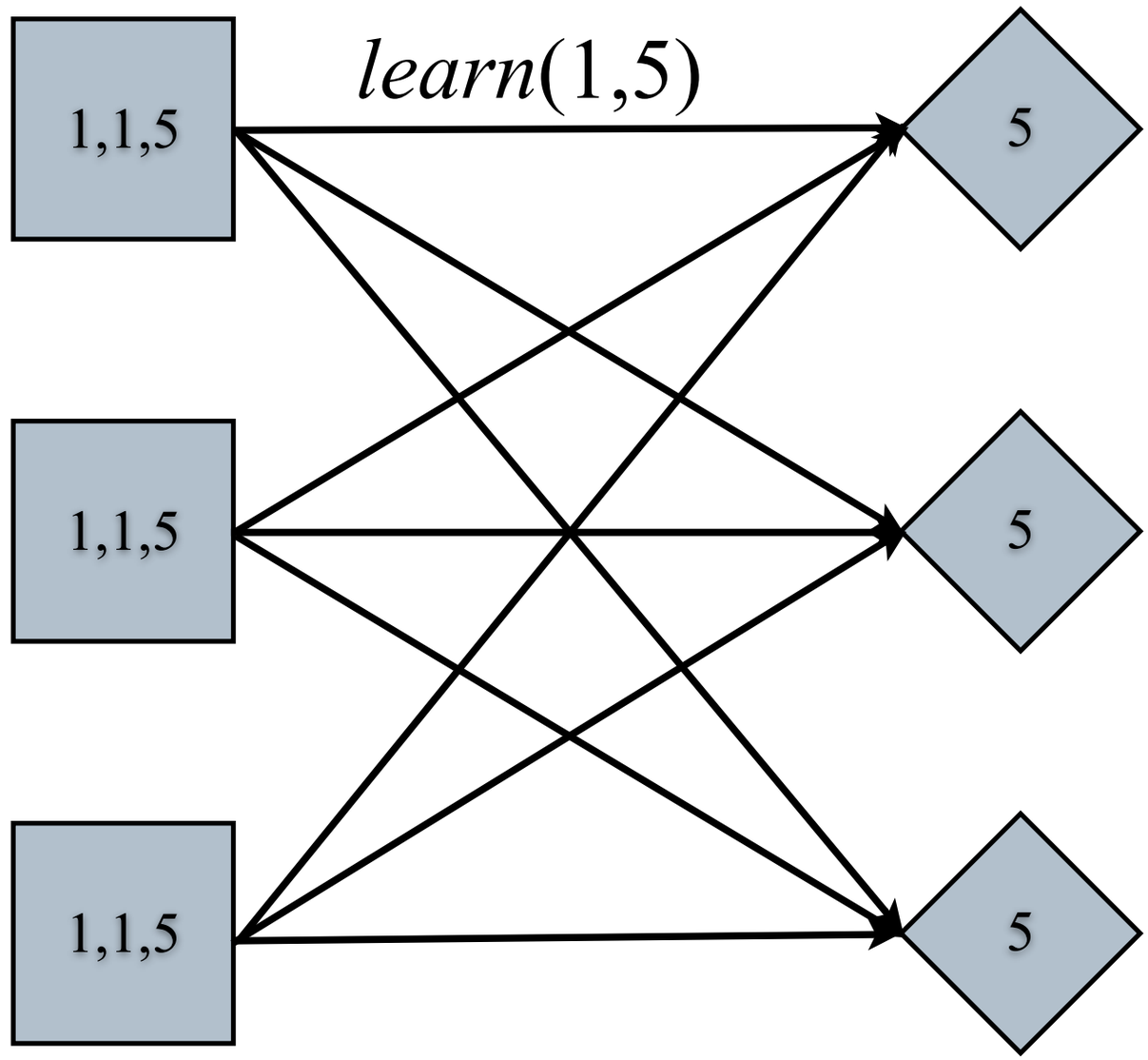
Full Paxos Execution



Full Paxos Execution



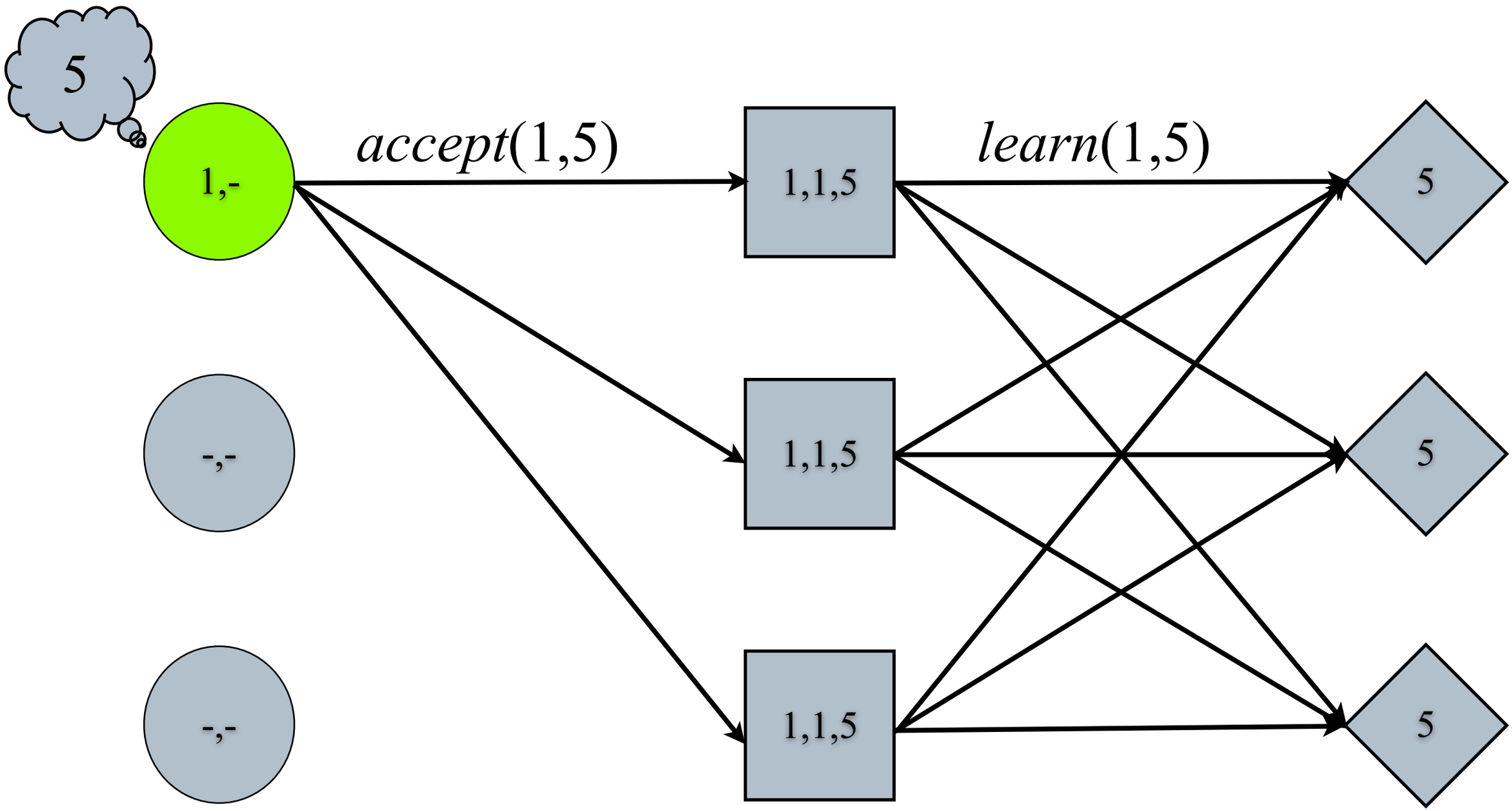




Paxos Optimization

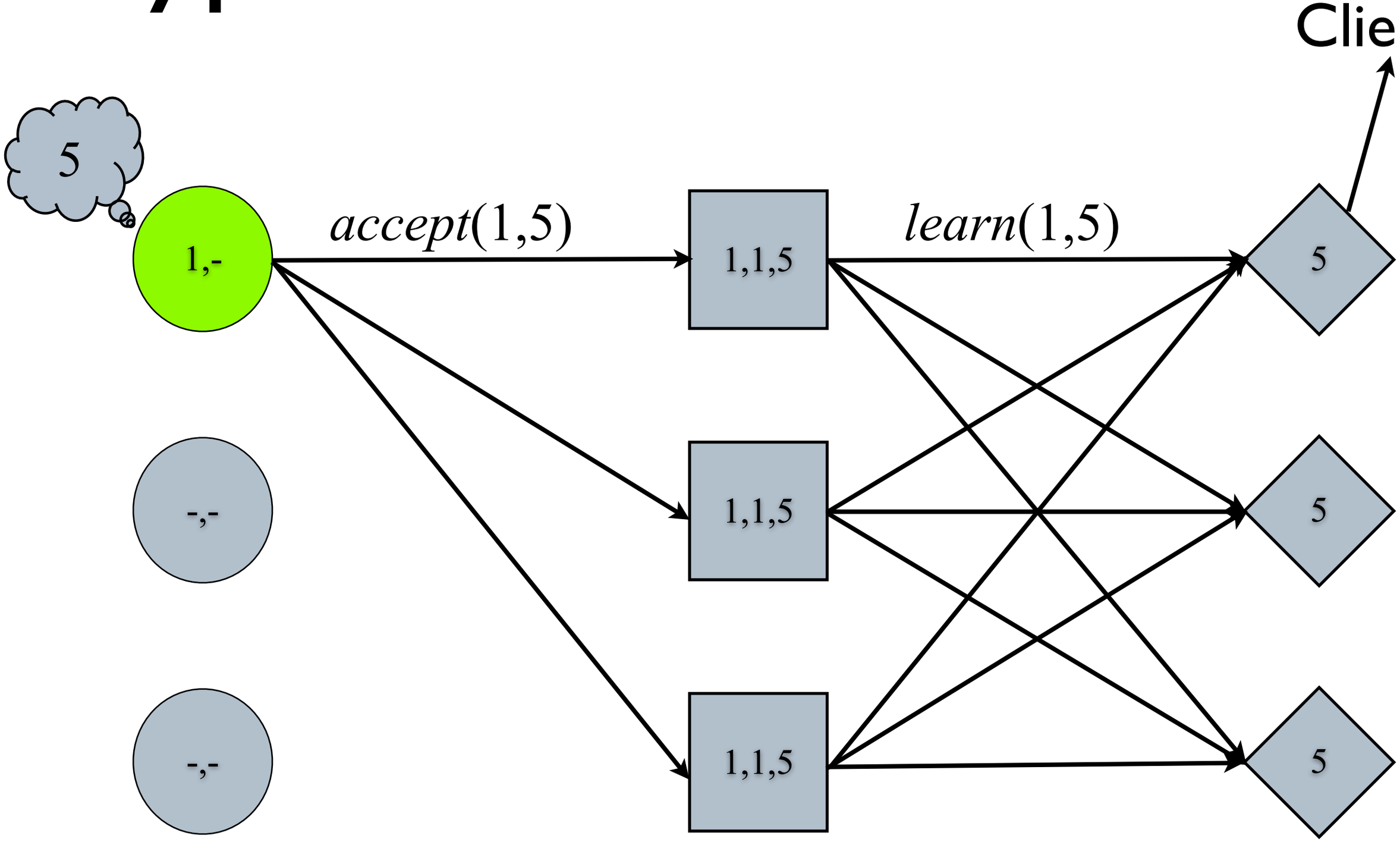
- Leader is stable across multiple Slots
  - Skip the two first message exchanges
- Does not work if multiple proposers think they are leader
  - This may cause multiple rounds of msg exchanges

Typical Paxos Execution



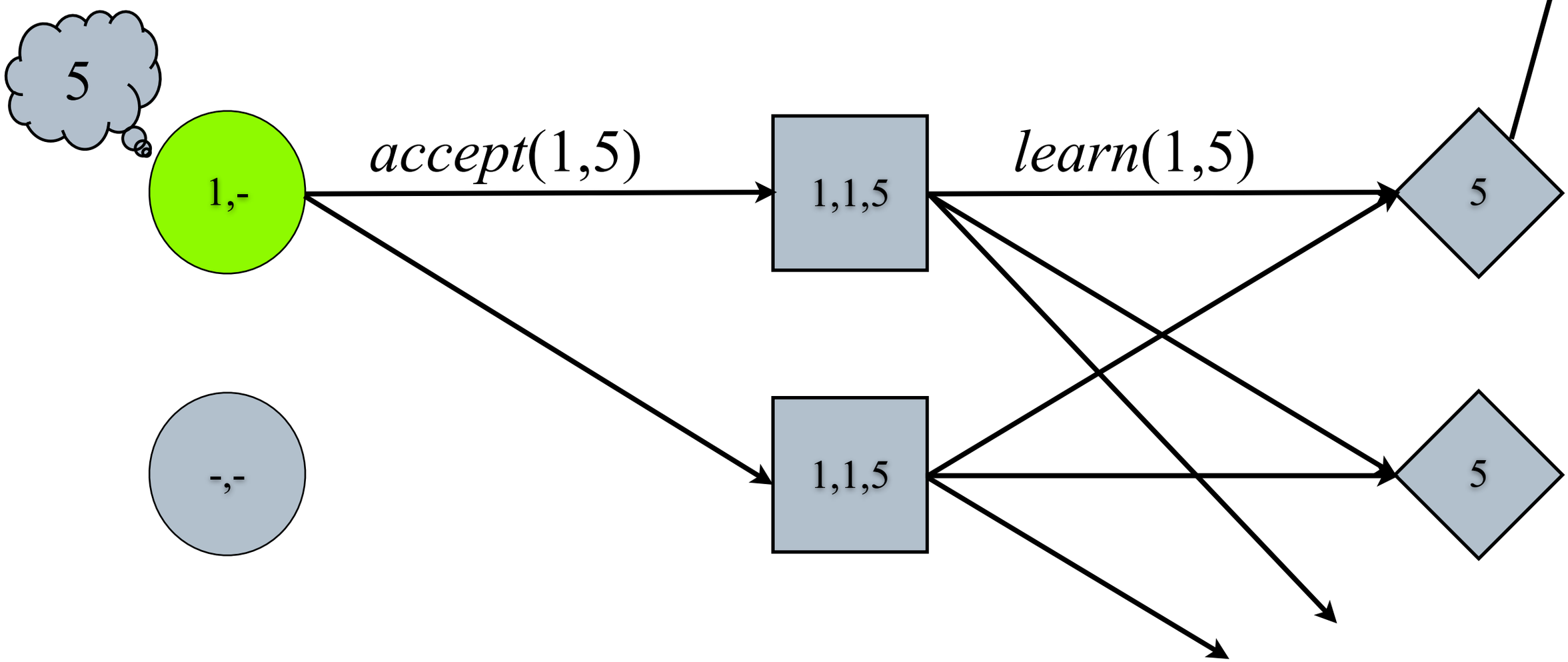
Phase 2

Typical Paxos Execution



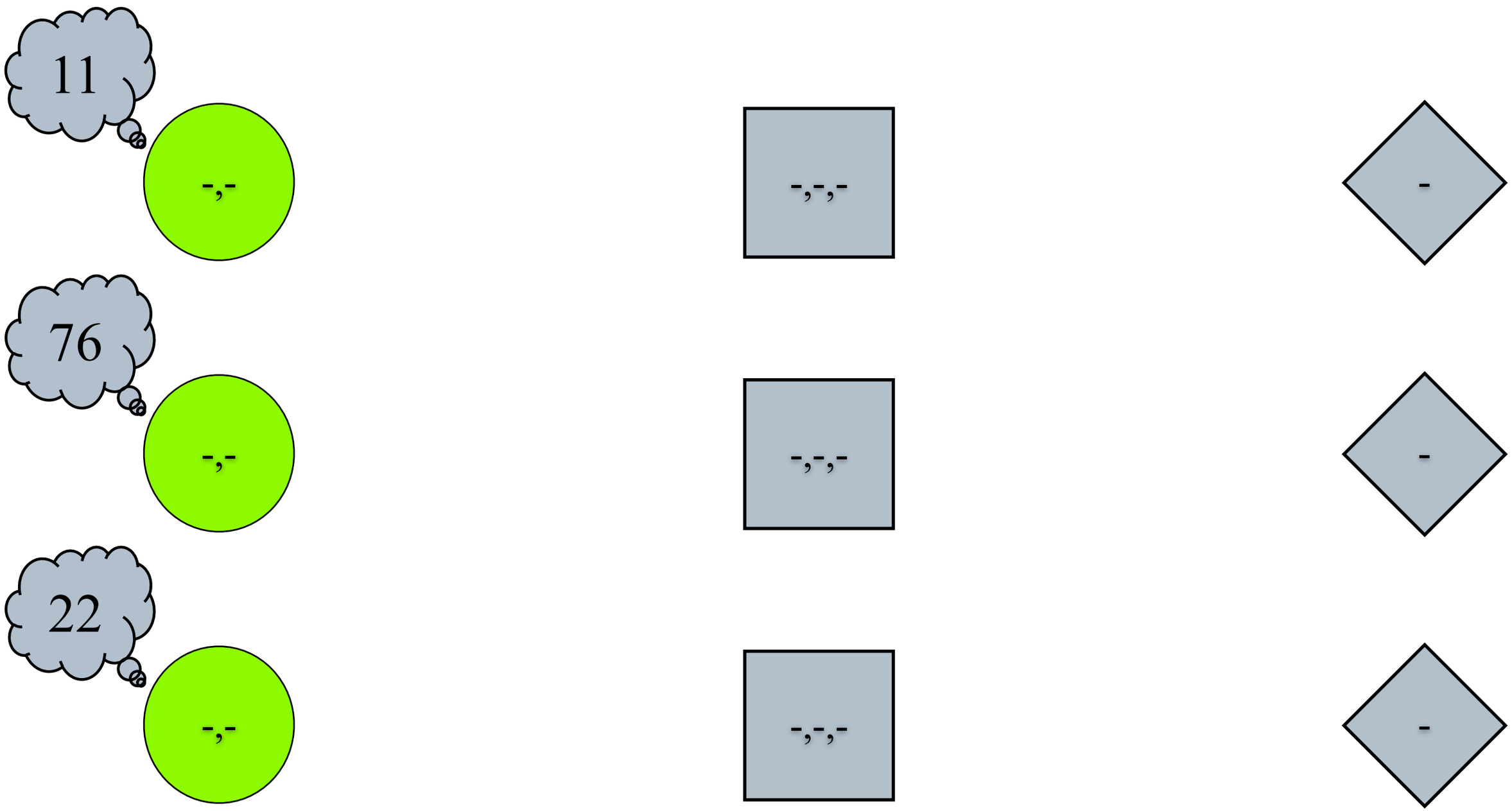
Phase 2

Paxos Execution With a Failed Replica



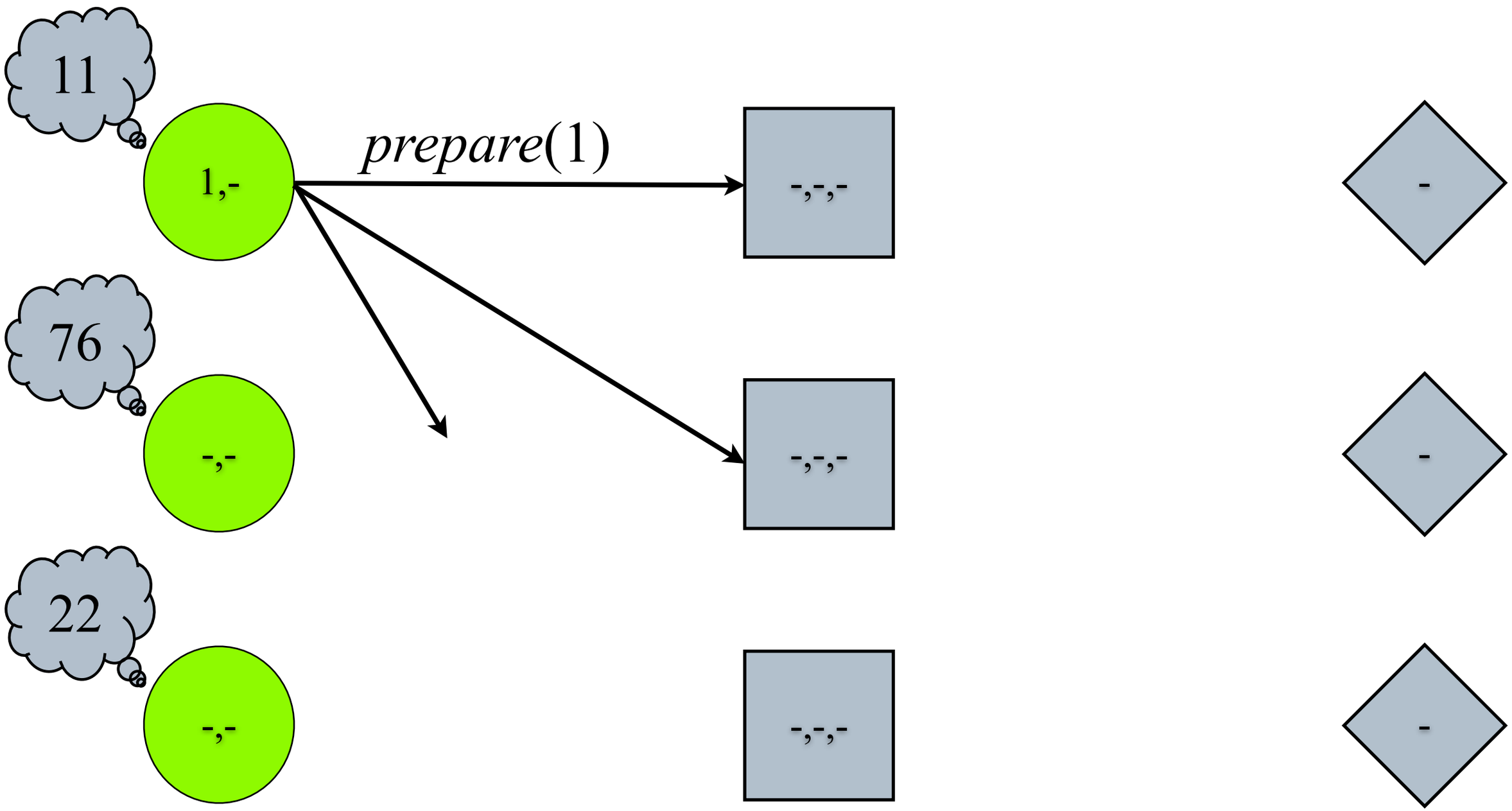
Example II
Problematic Paxos Execution

Problematic Paxos Execution



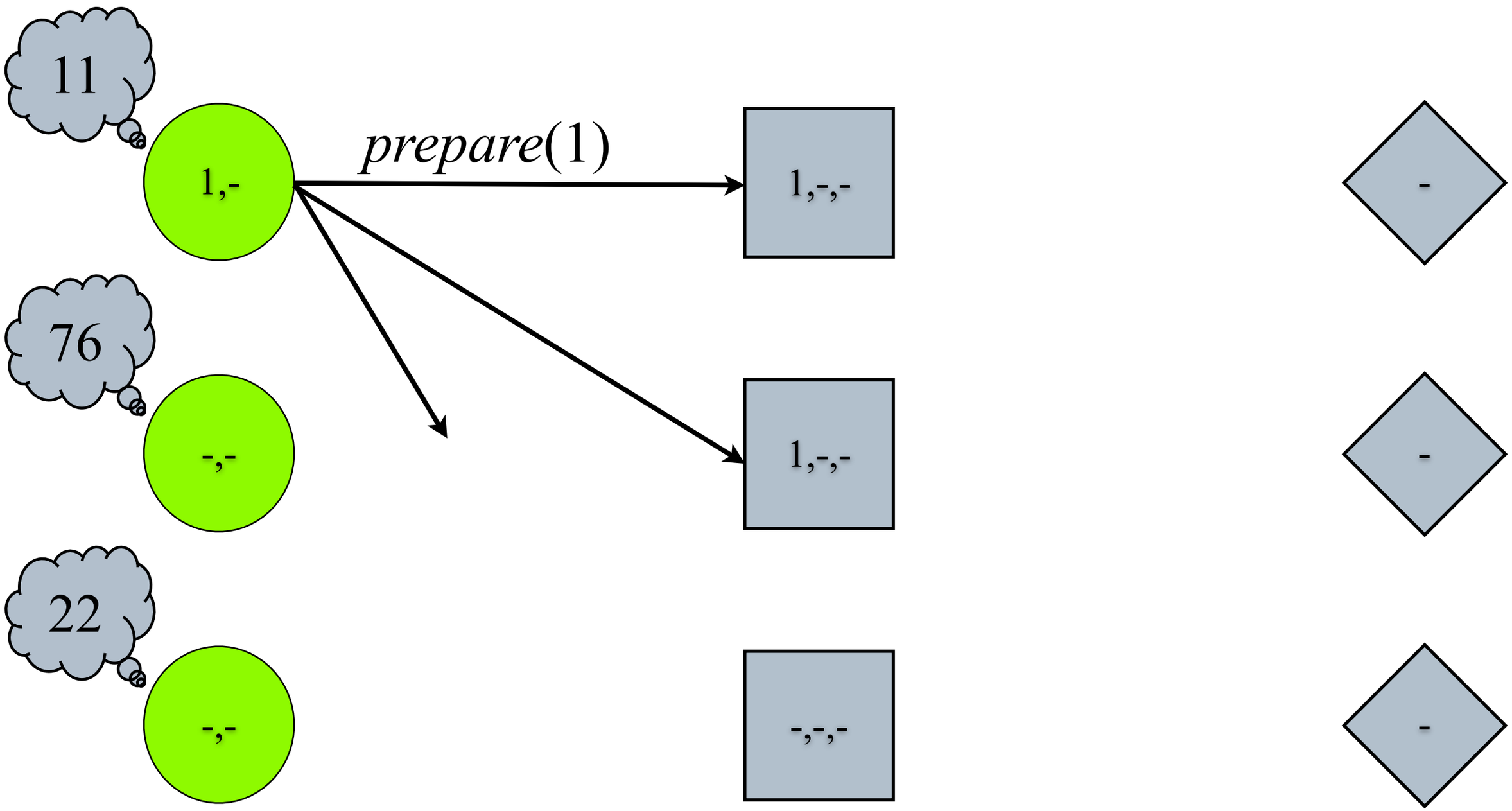
Problematic Paxos Execution

Phase 1a



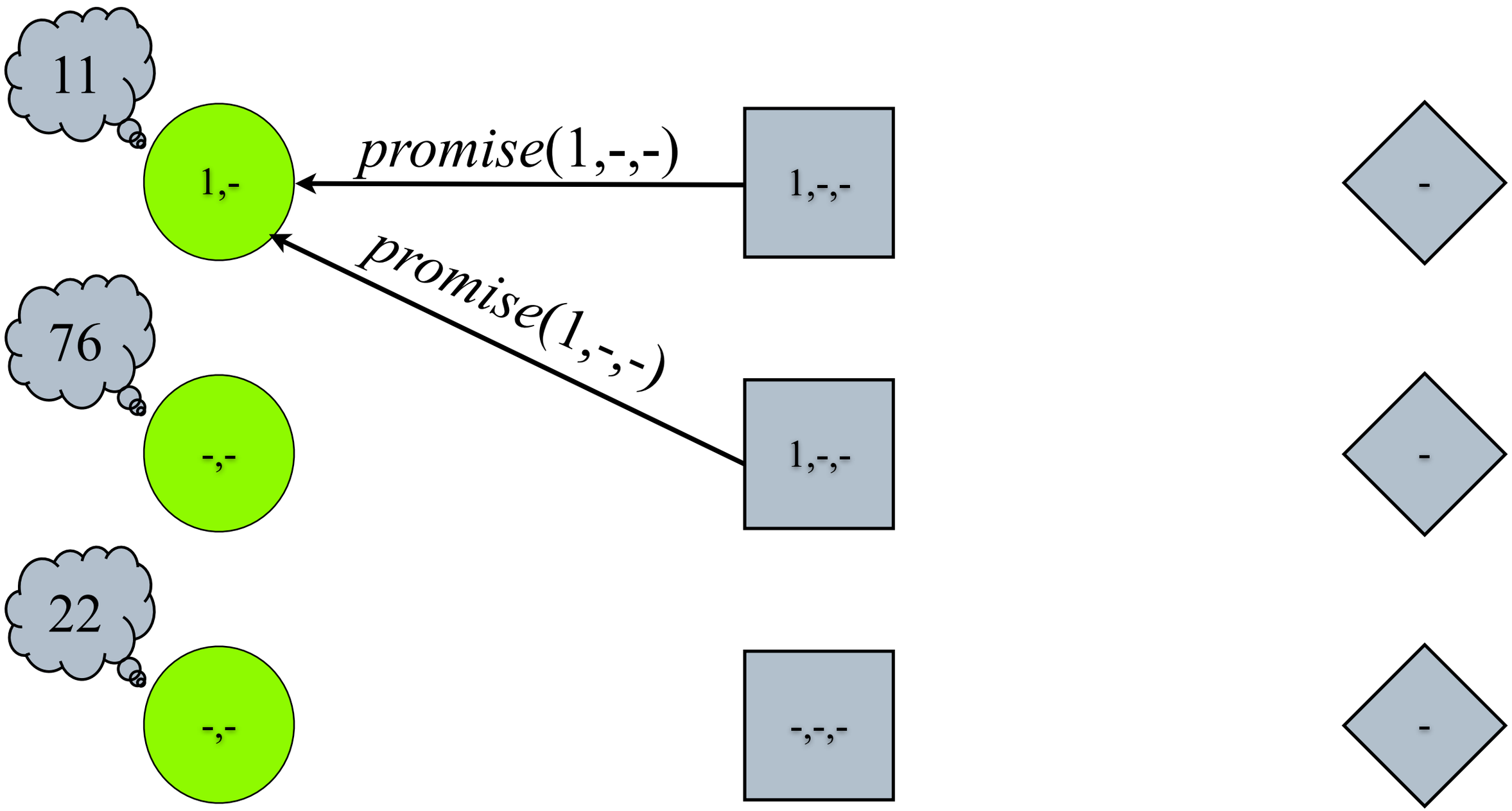
Problematic Paxos Execution

Phase 1a



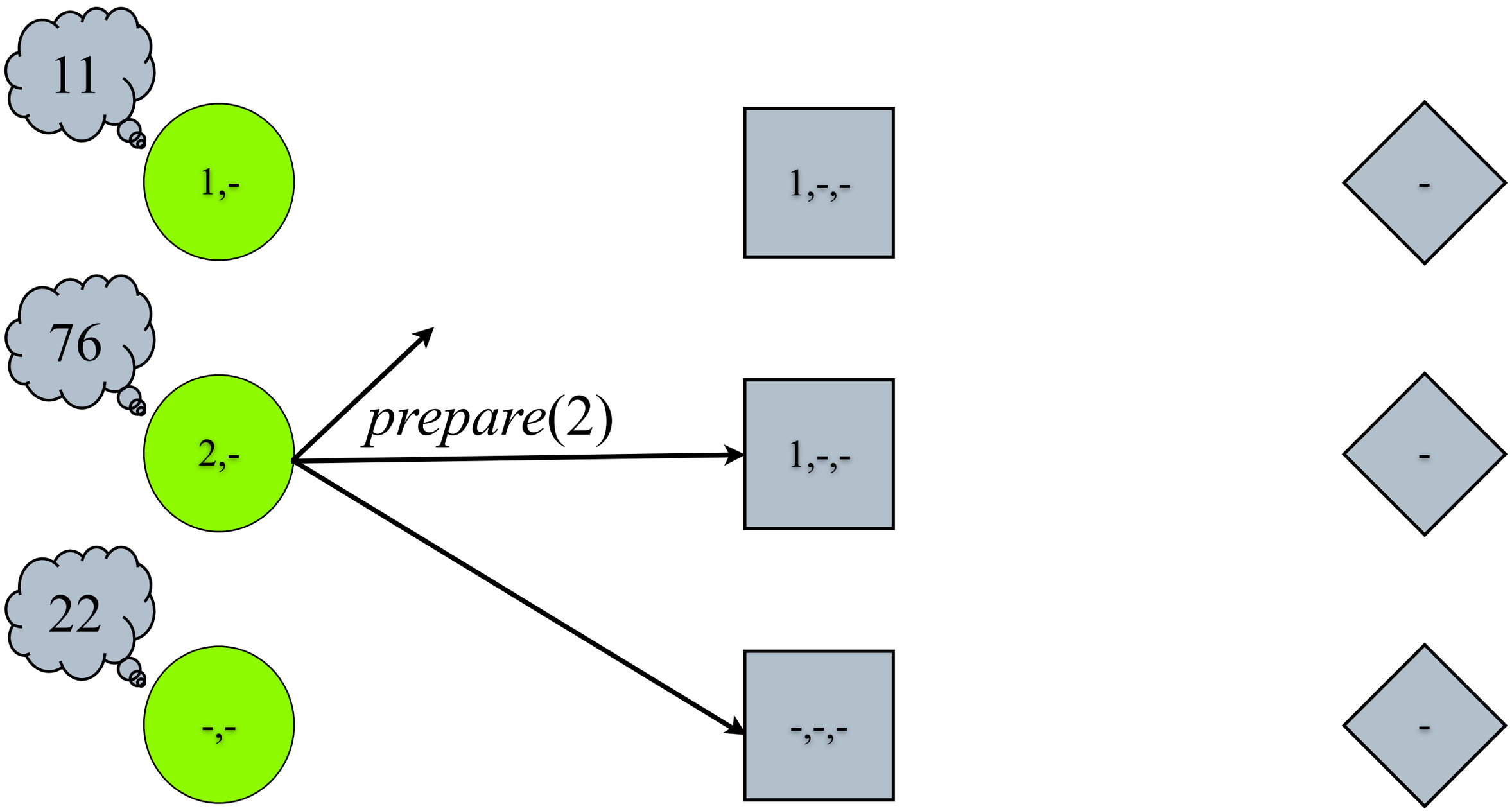
Phase 1b

Problematic Paxos Execution



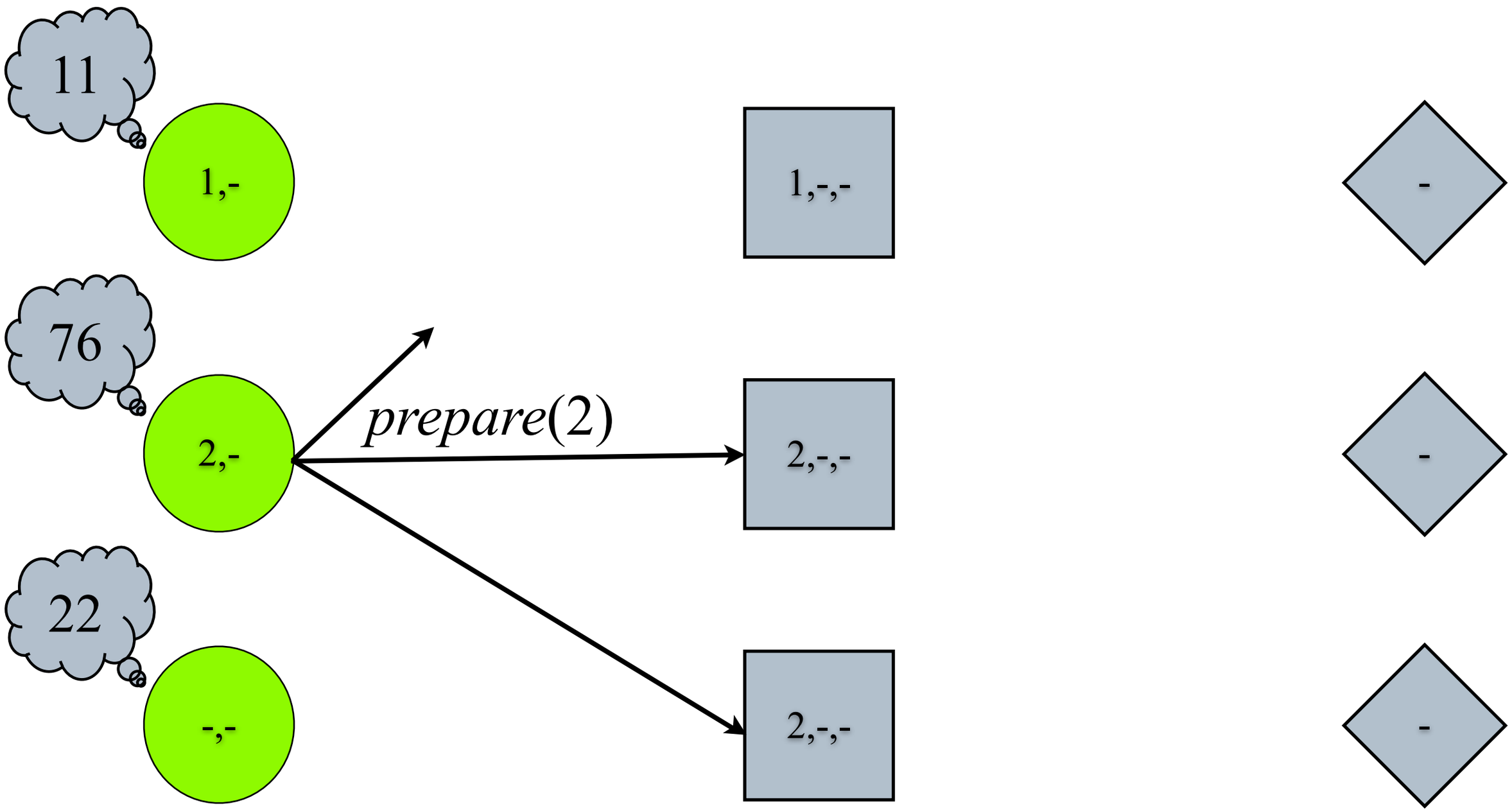
Problematic Paxos Execution

Phase 1a



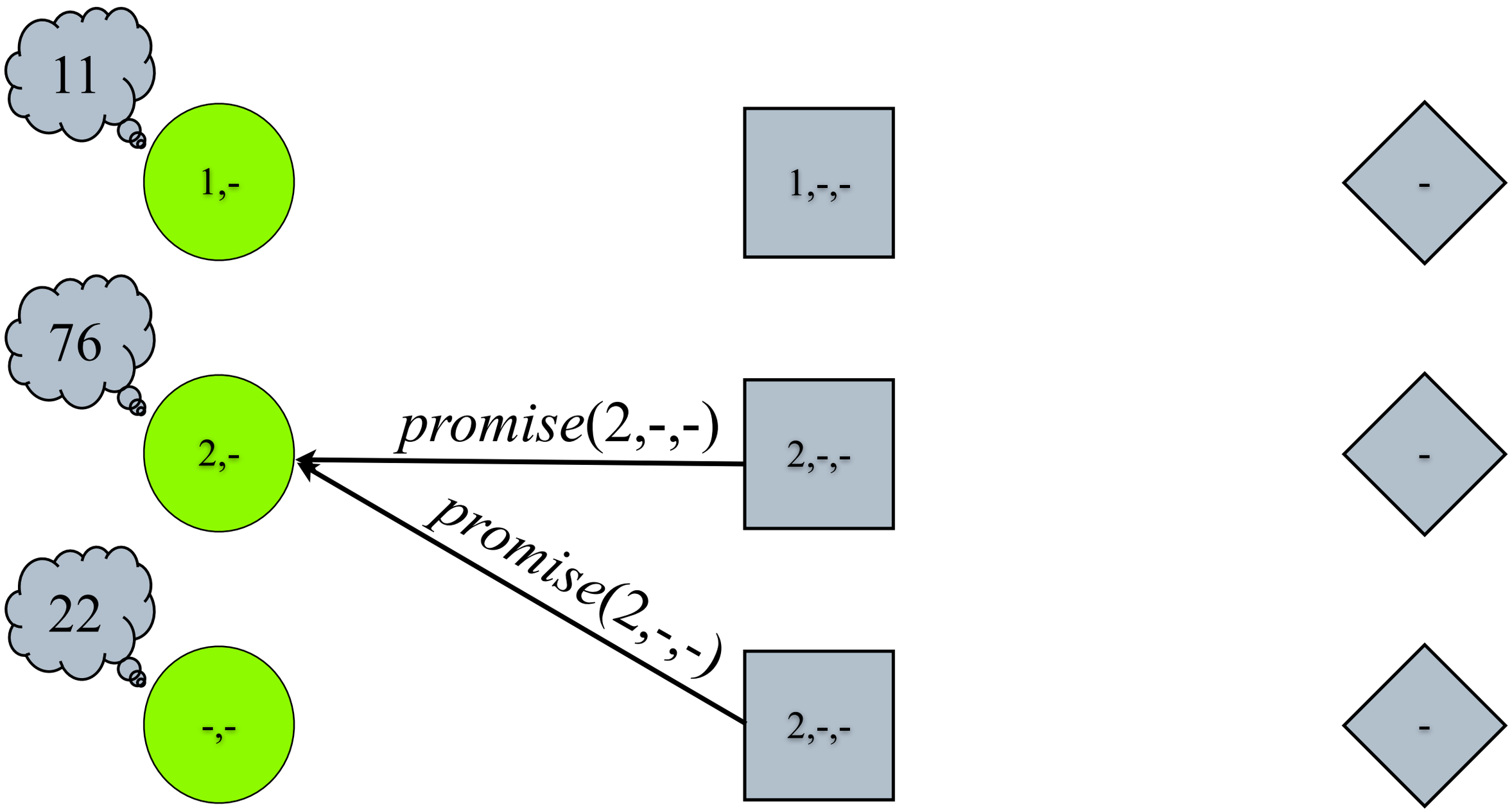
Problematic Paxos Execution

Phase 1a



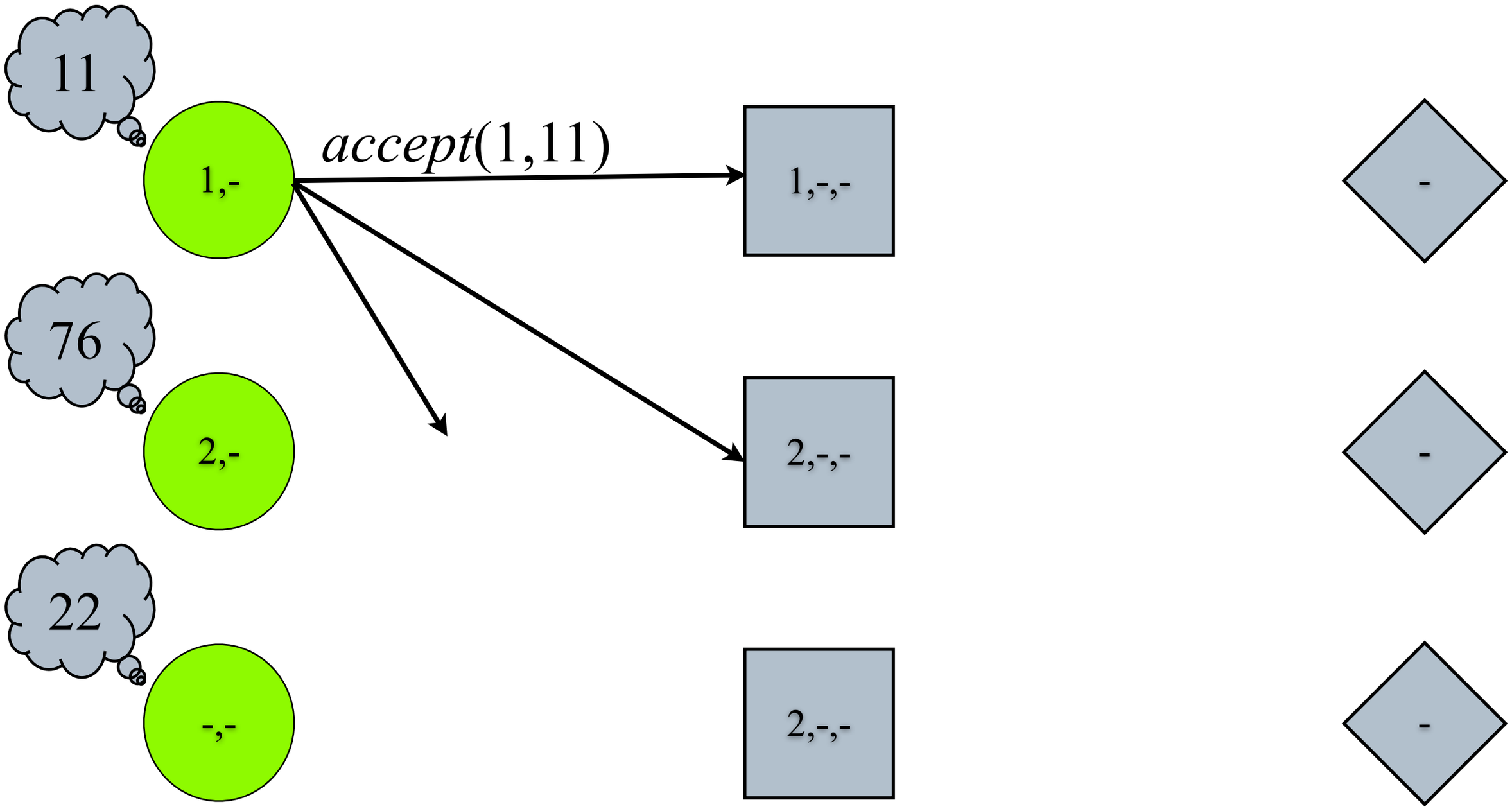
Problematic Paxos Execution

Phase 1b



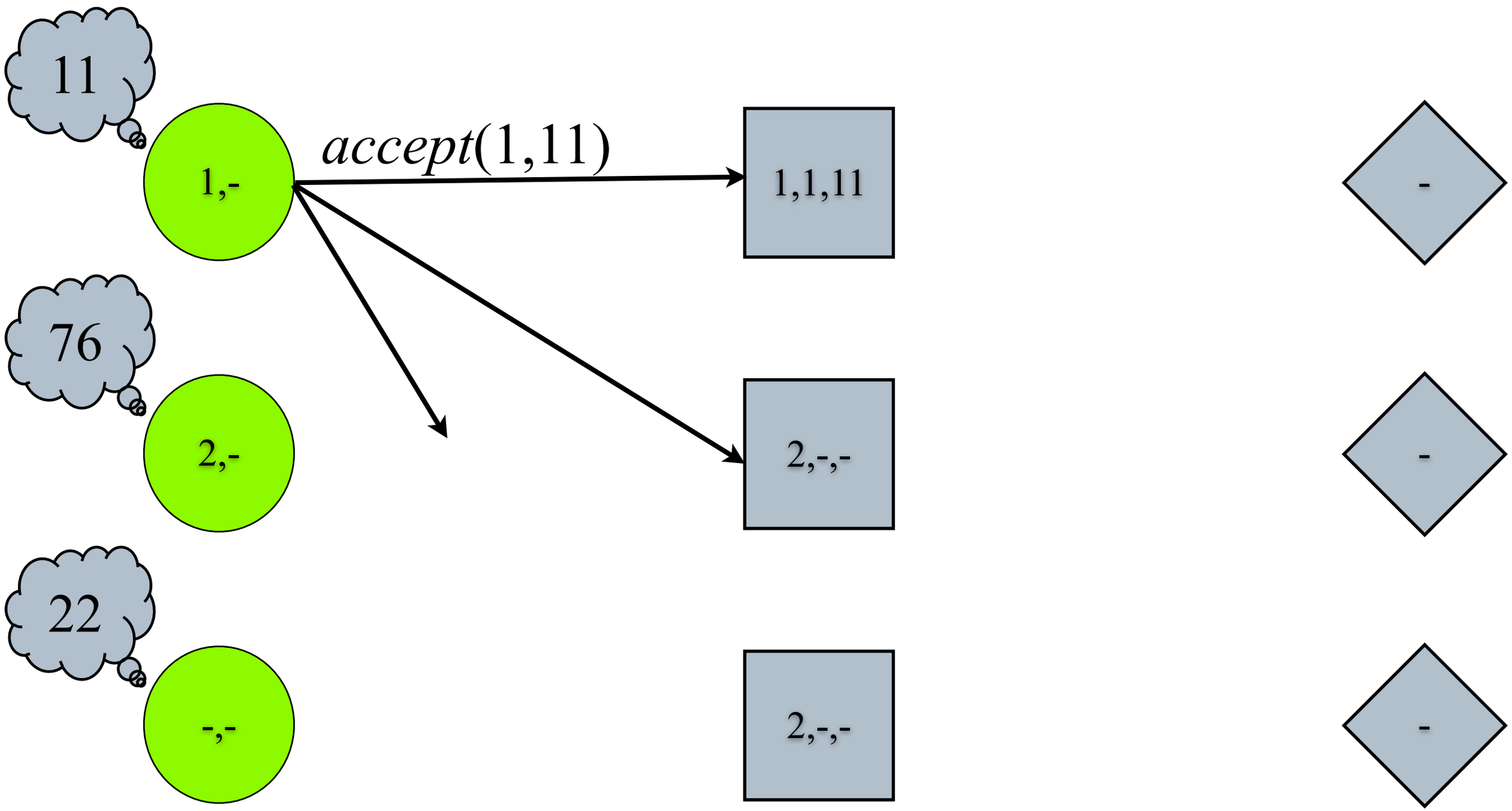
Problematic Paxos Execution

Phase 2a



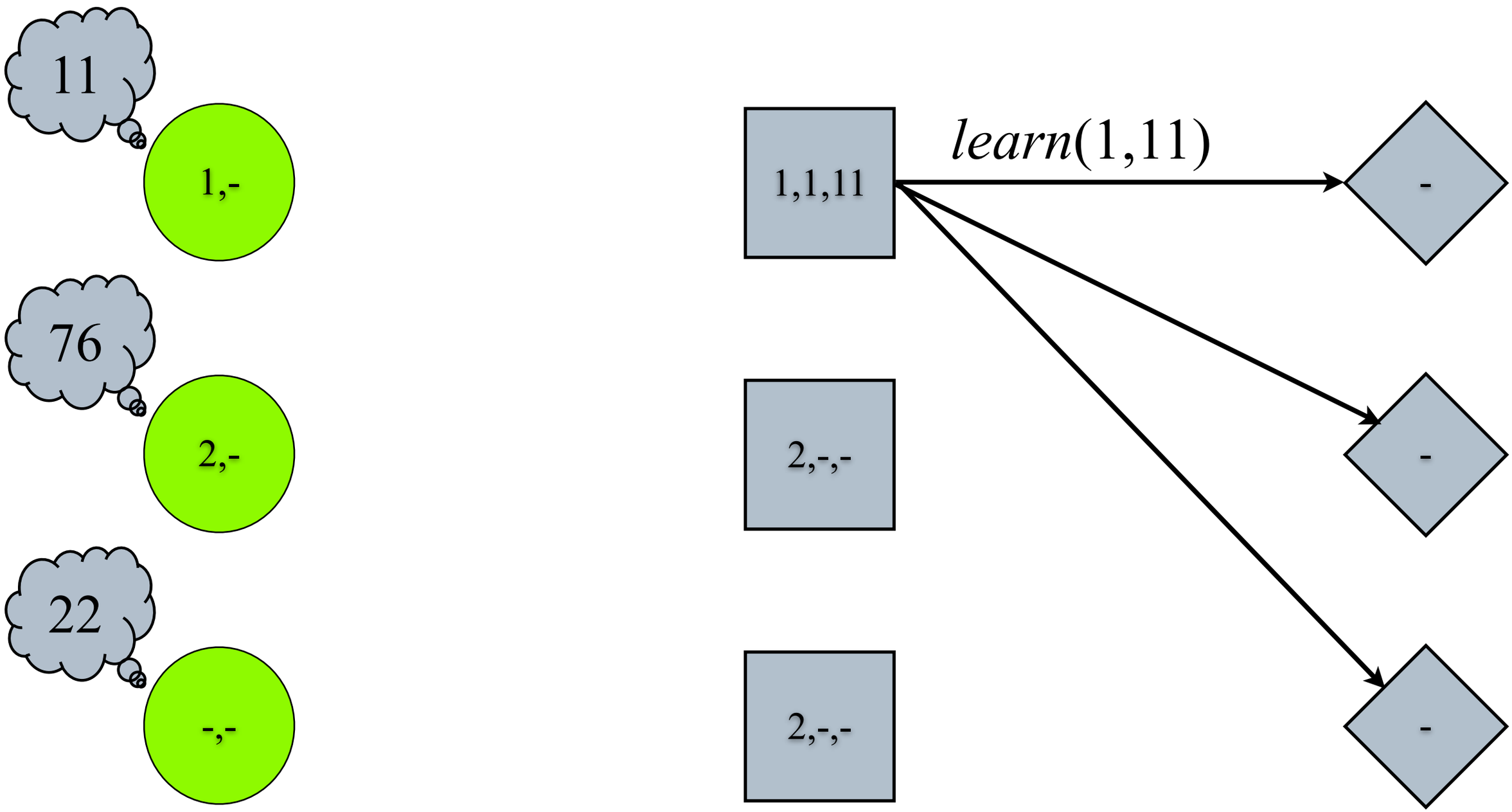
Problematic Paxos Execution

Phase 2a



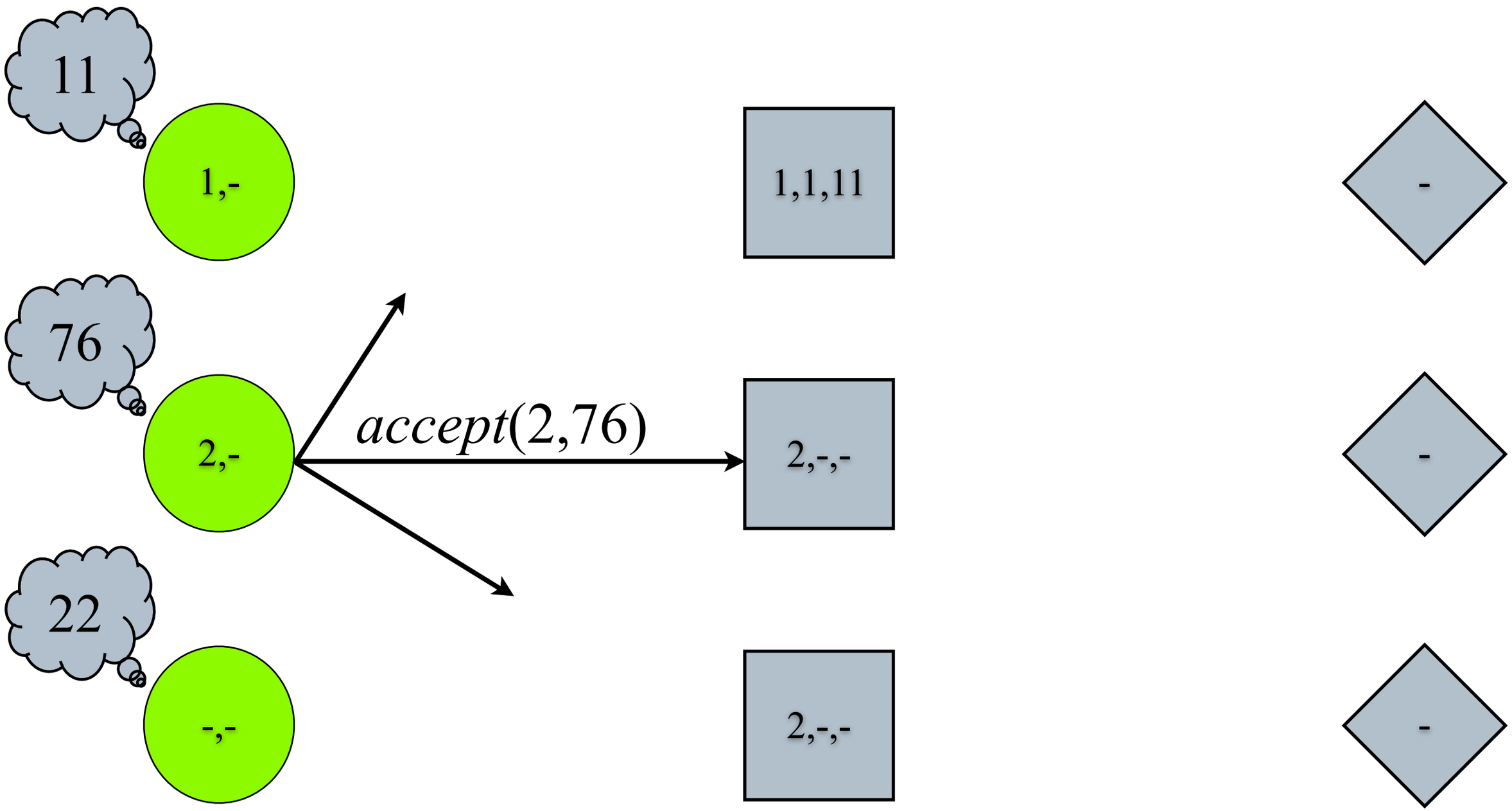
Problematic Paxos Execution

Phase 2b



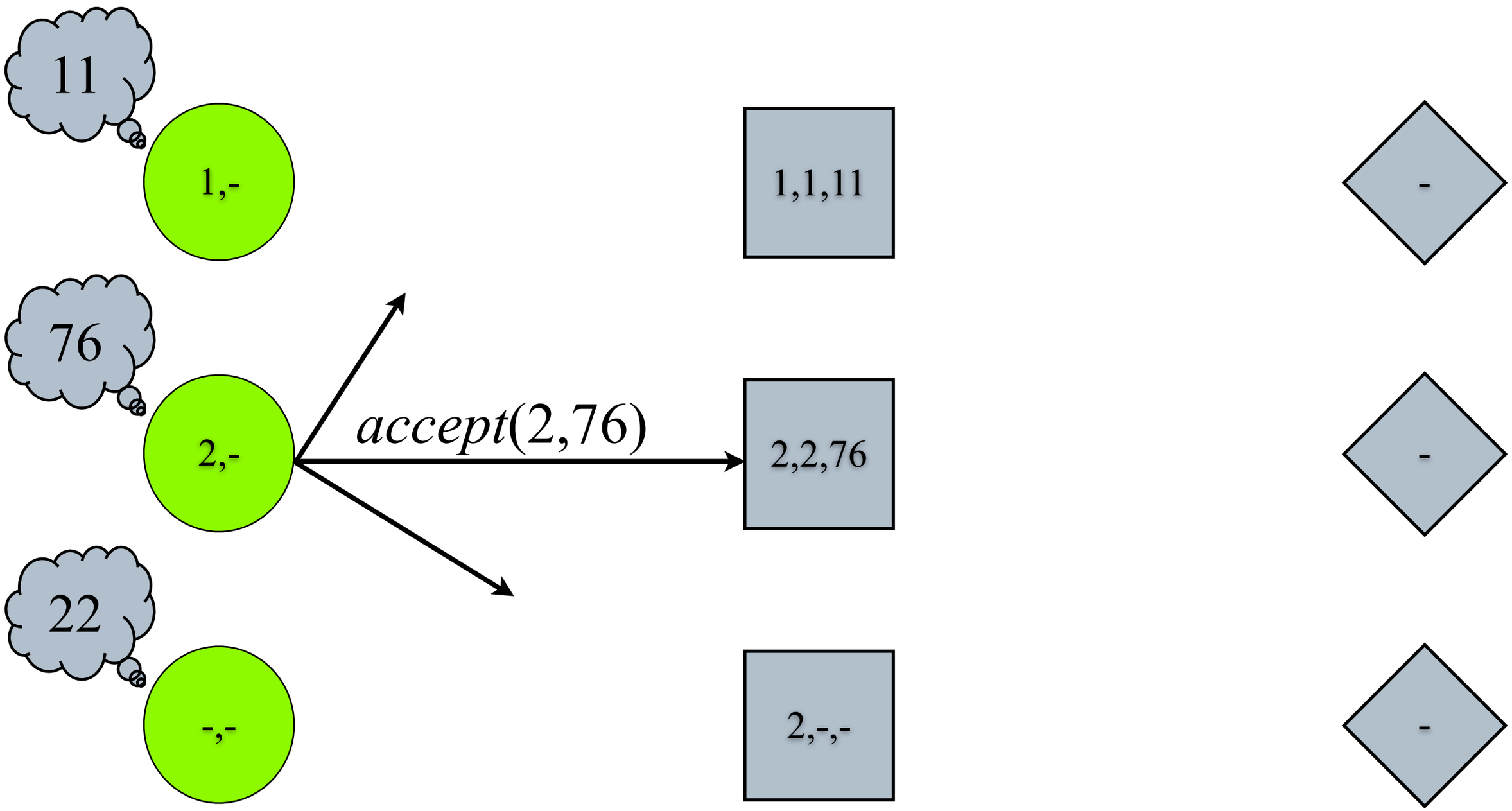
Problematic Paxos Execution

Phase 2a



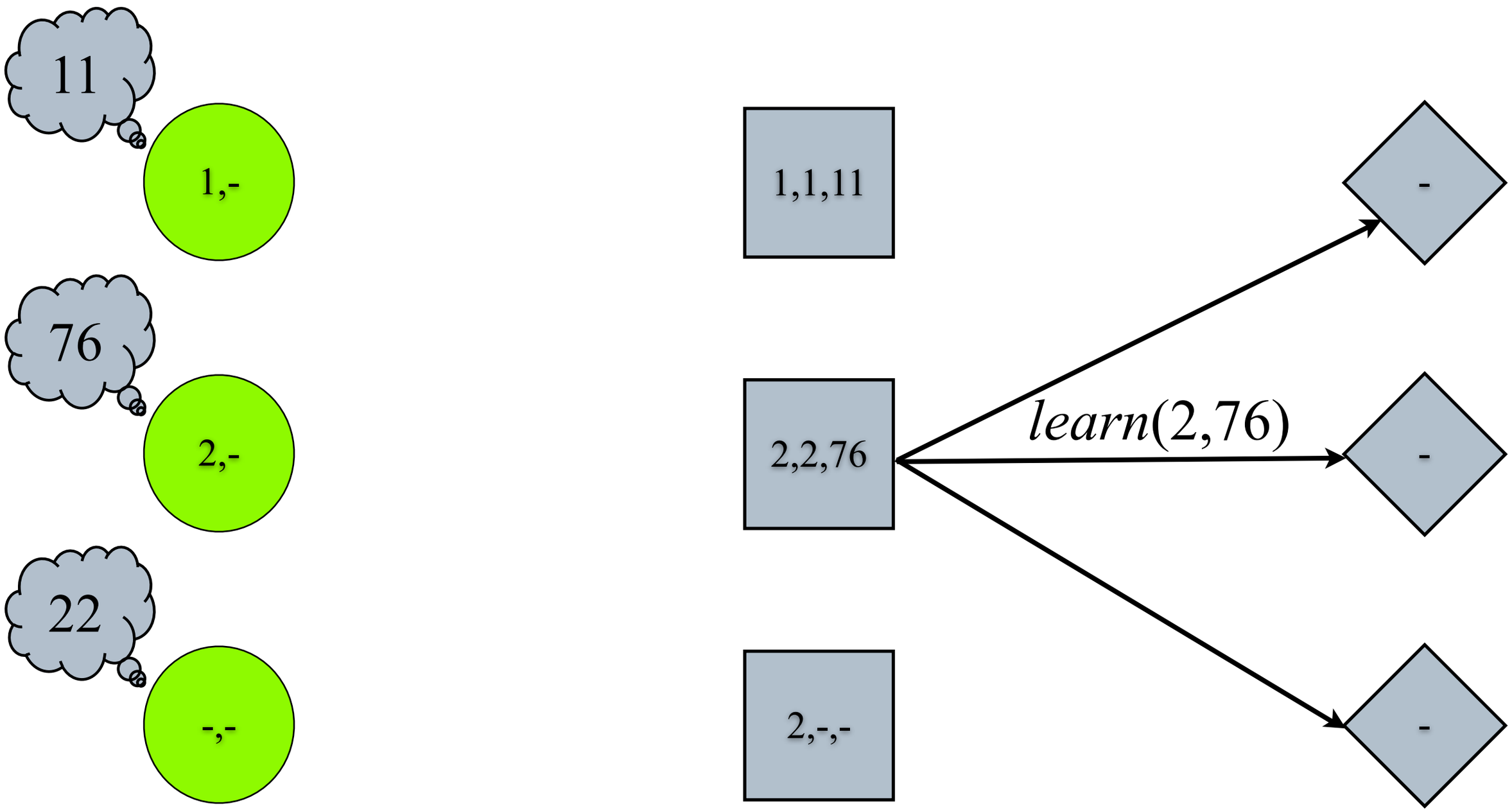
Problematic Paxos Execution

Phase 2a



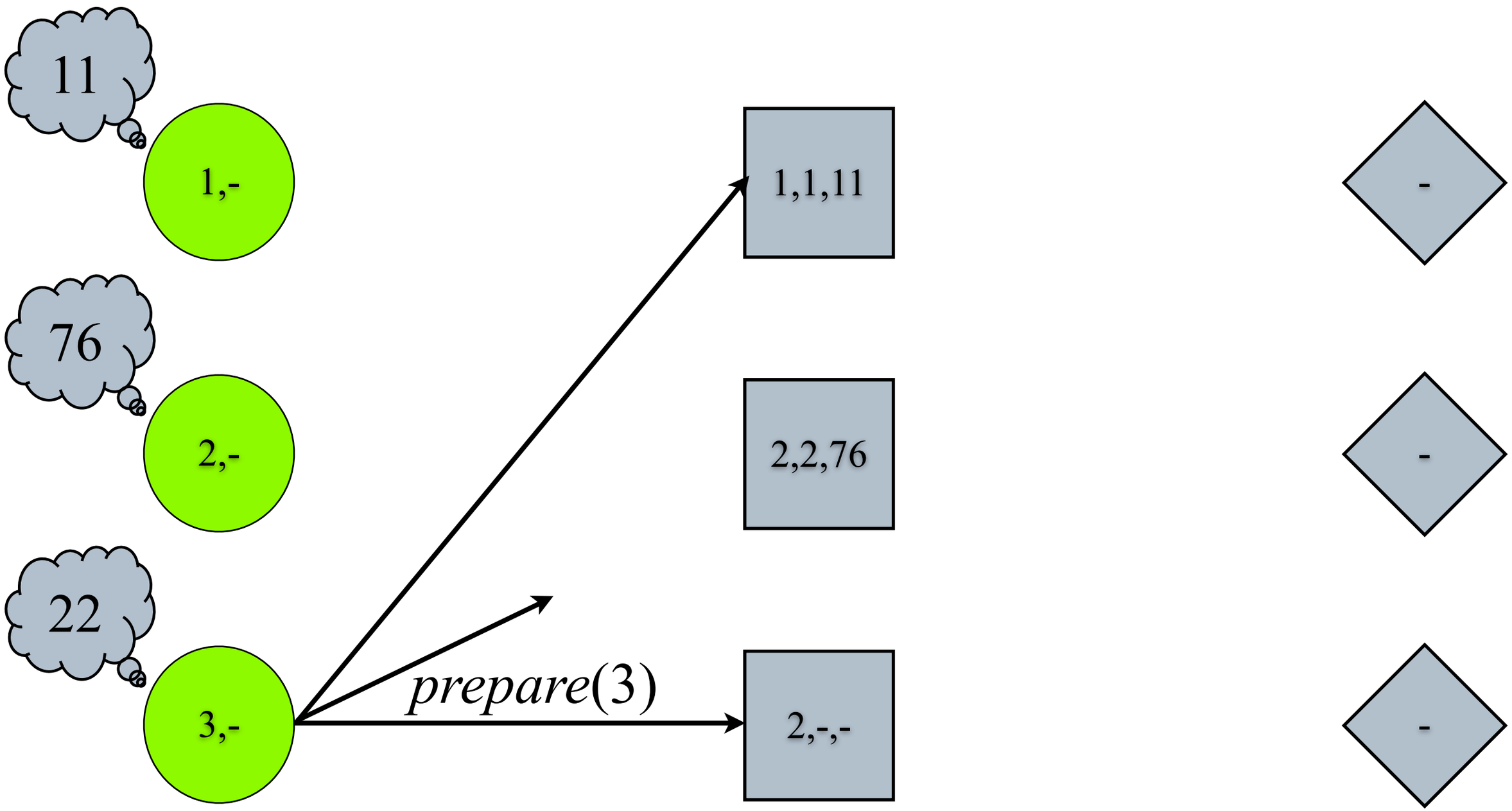
Problematic Paxos Execution

Phase 2b



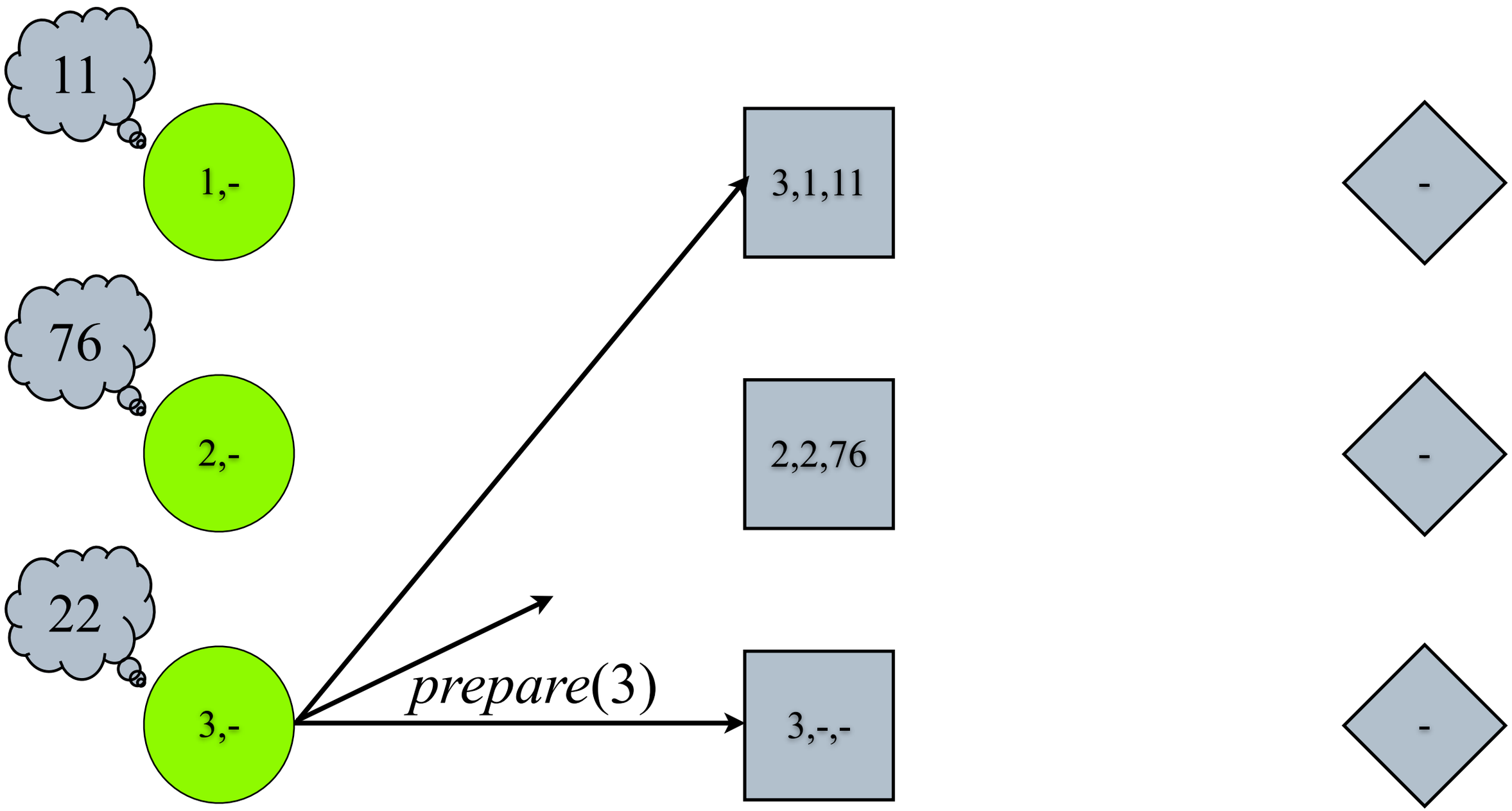
Problematic Paxos Execution

Phase 1a



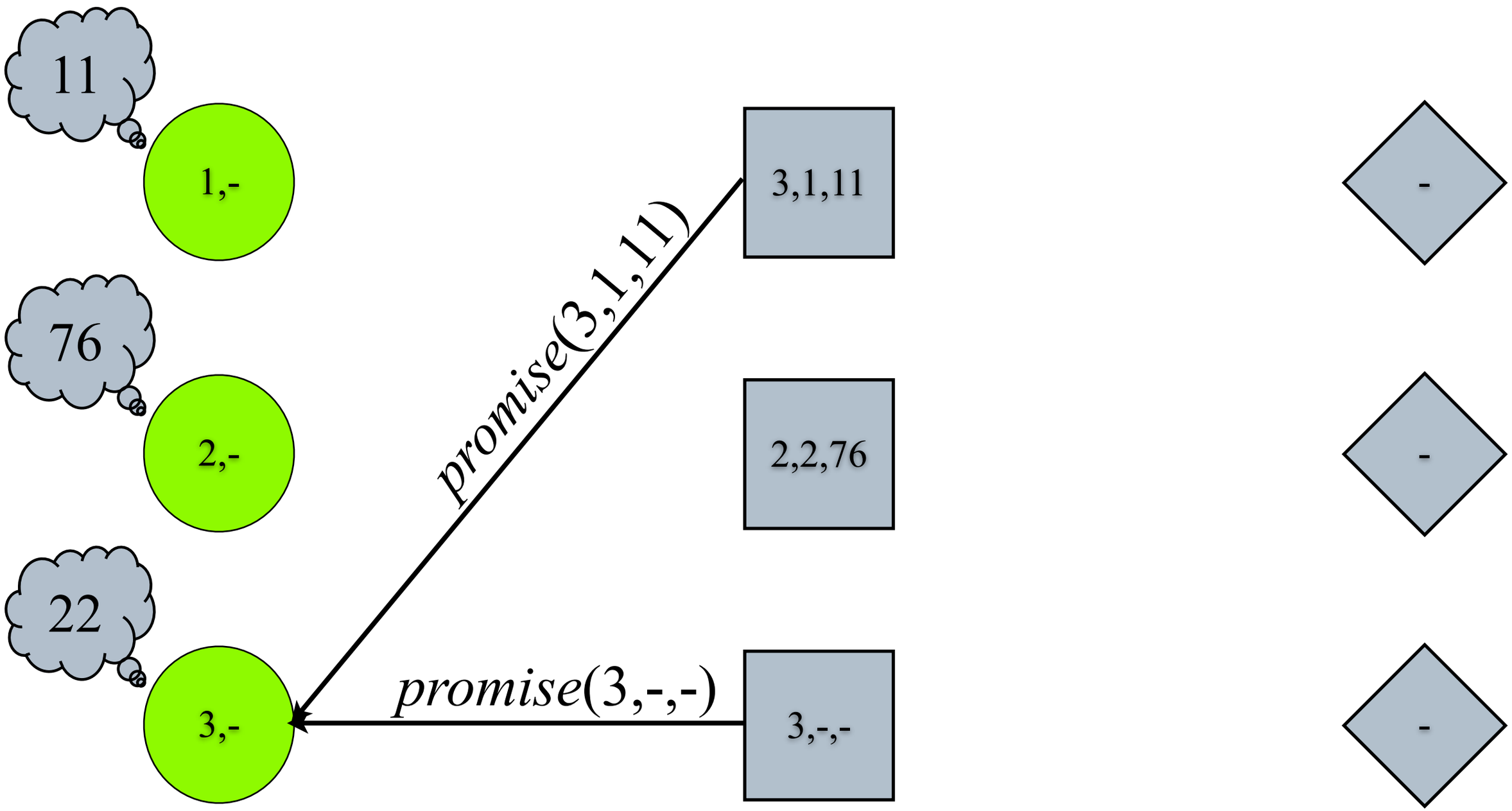
Problematic Paxos Execution

Phase 1a



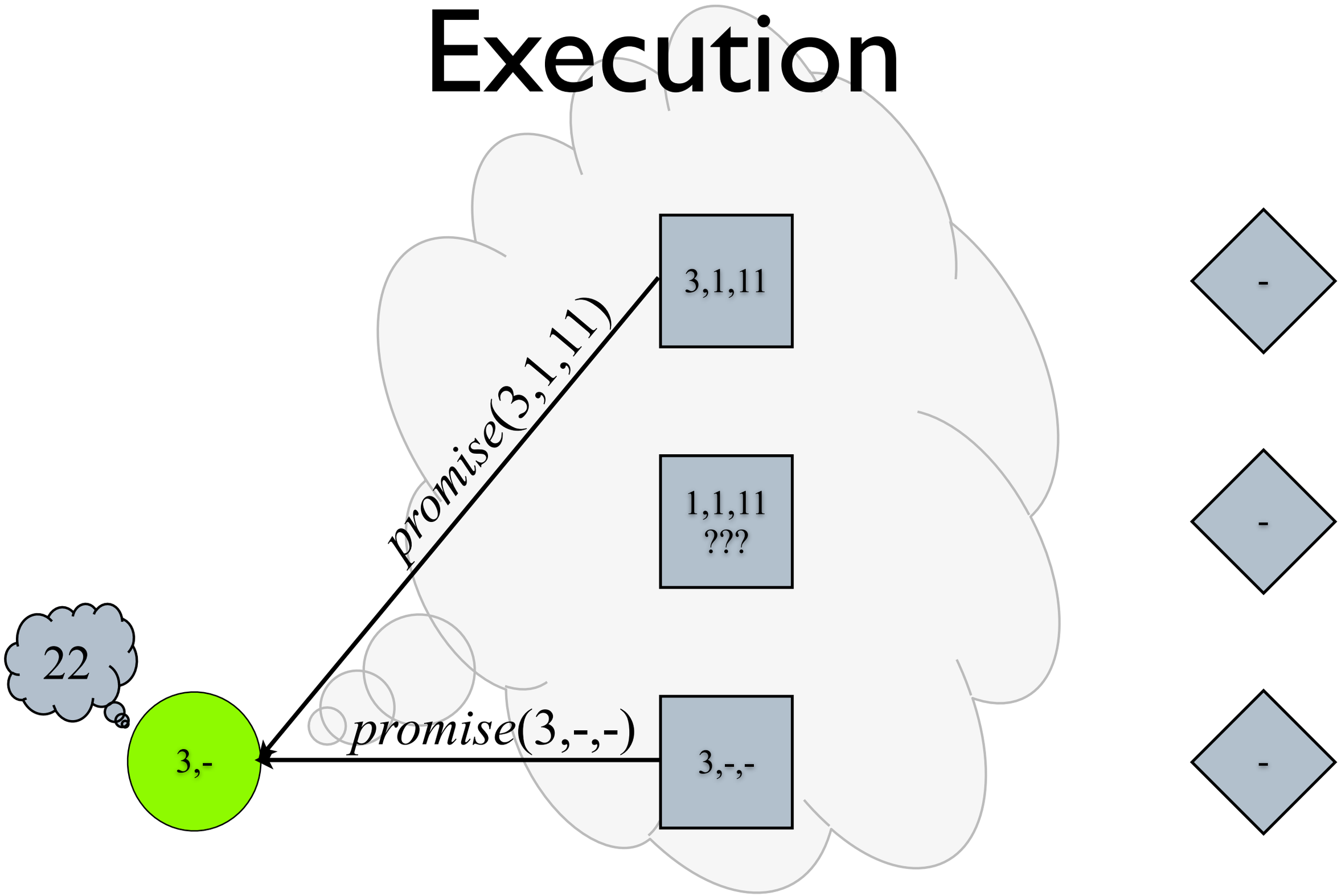
Phase 1b

Problematic Paxos Execution



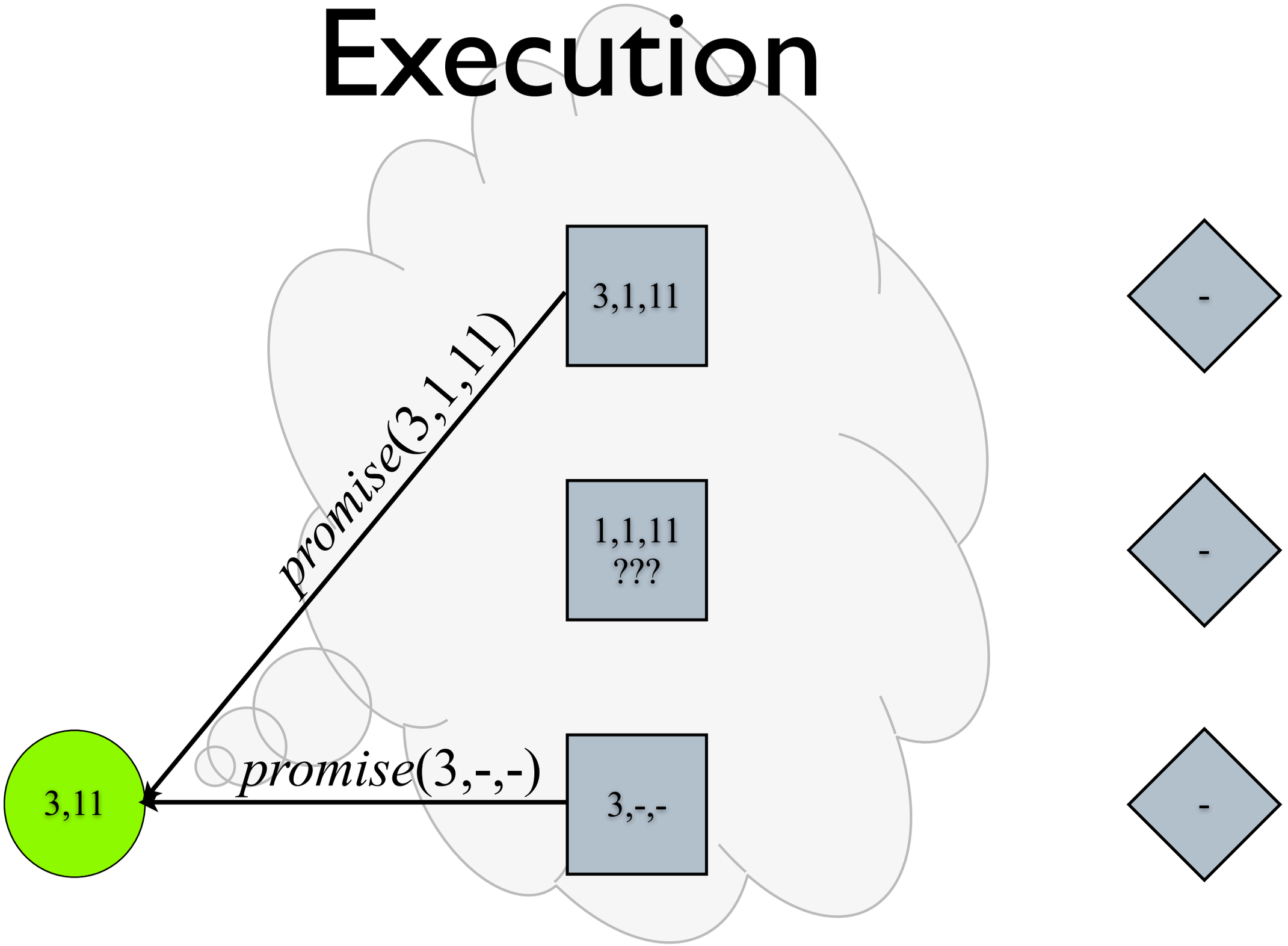
Problematic Paxos Execution

Phase 1b



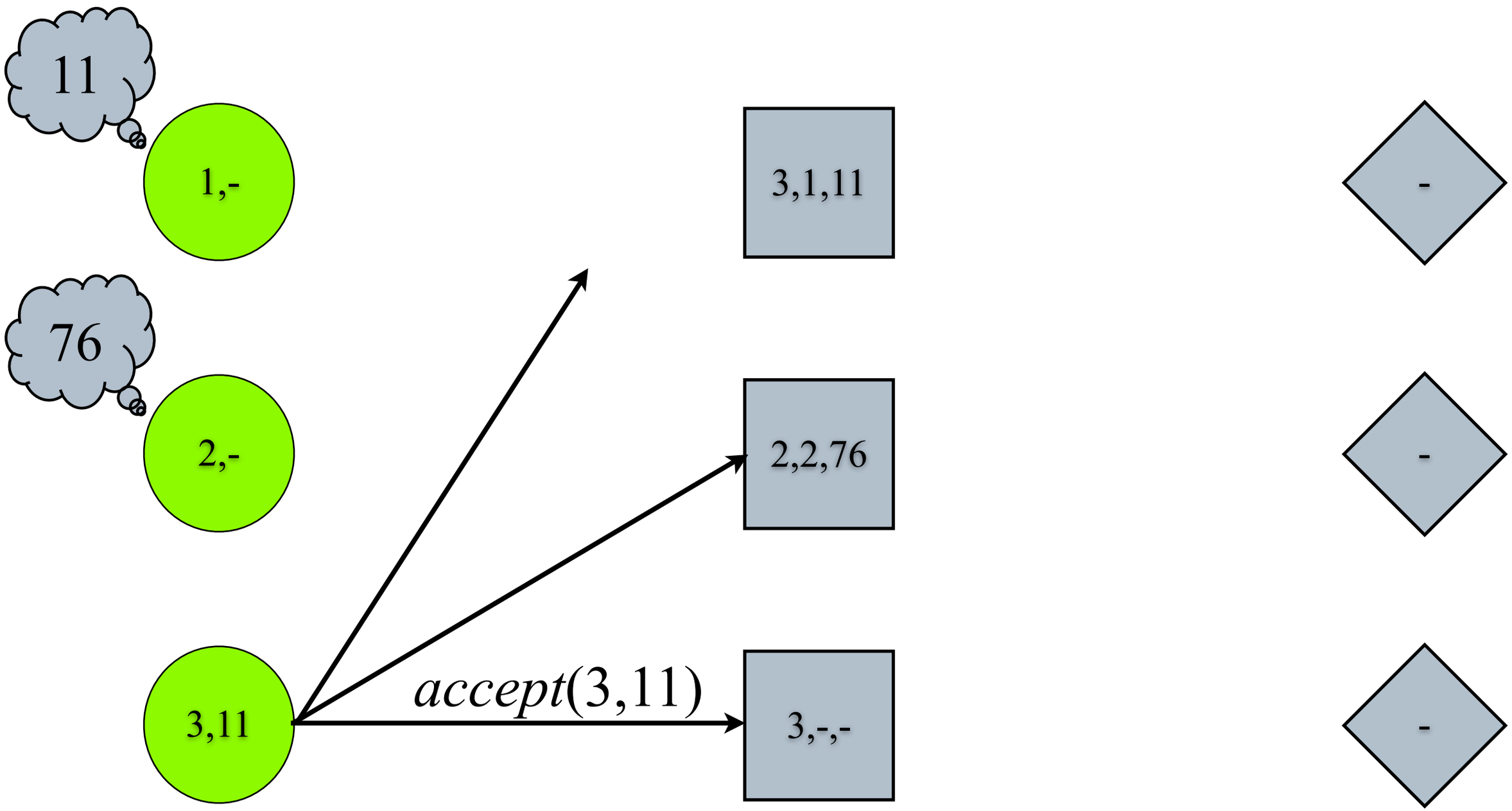
Problematic Paxos Execution

Phase 1b



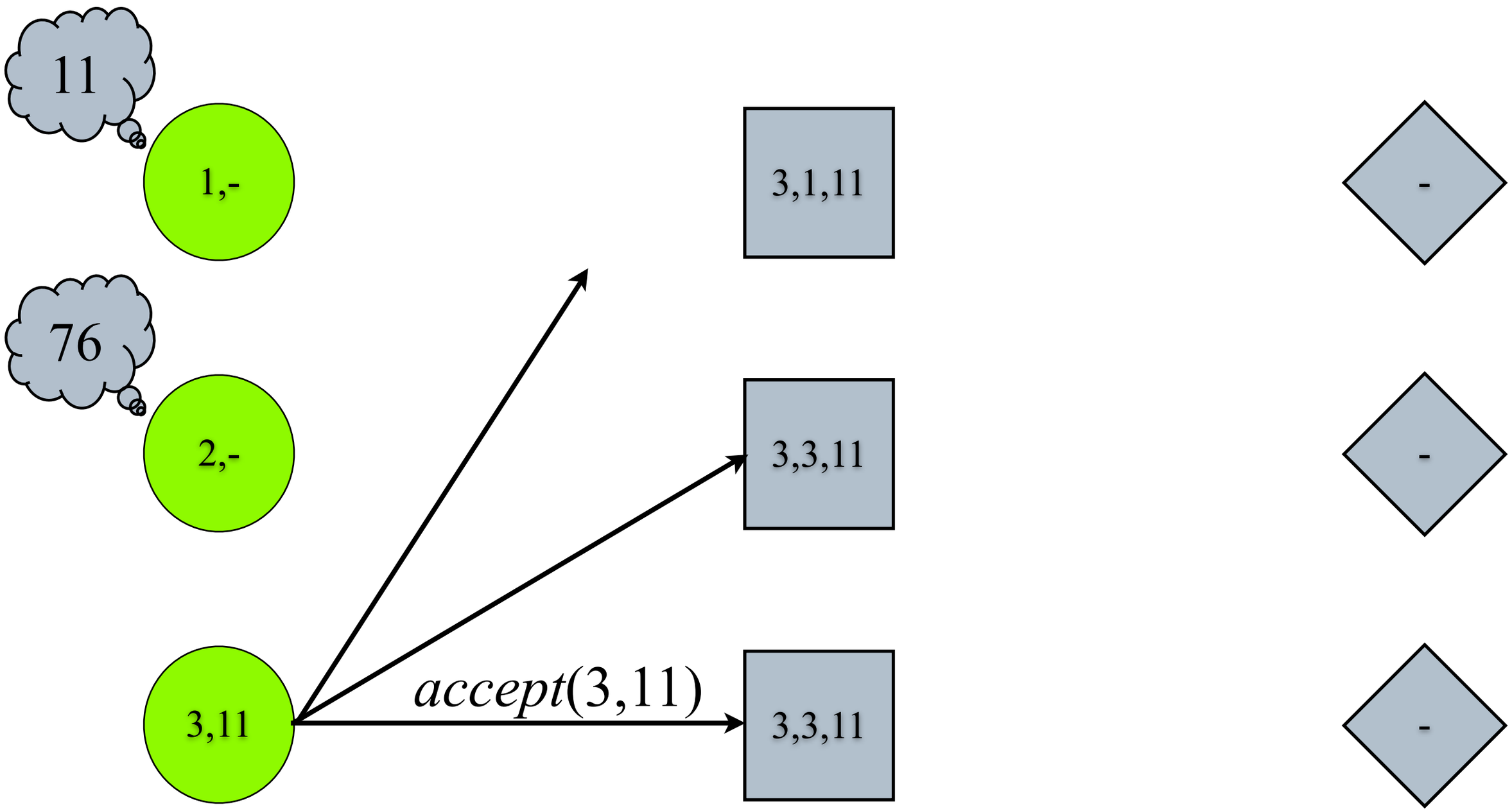
Problematic Paxos Execution

Phase 2a



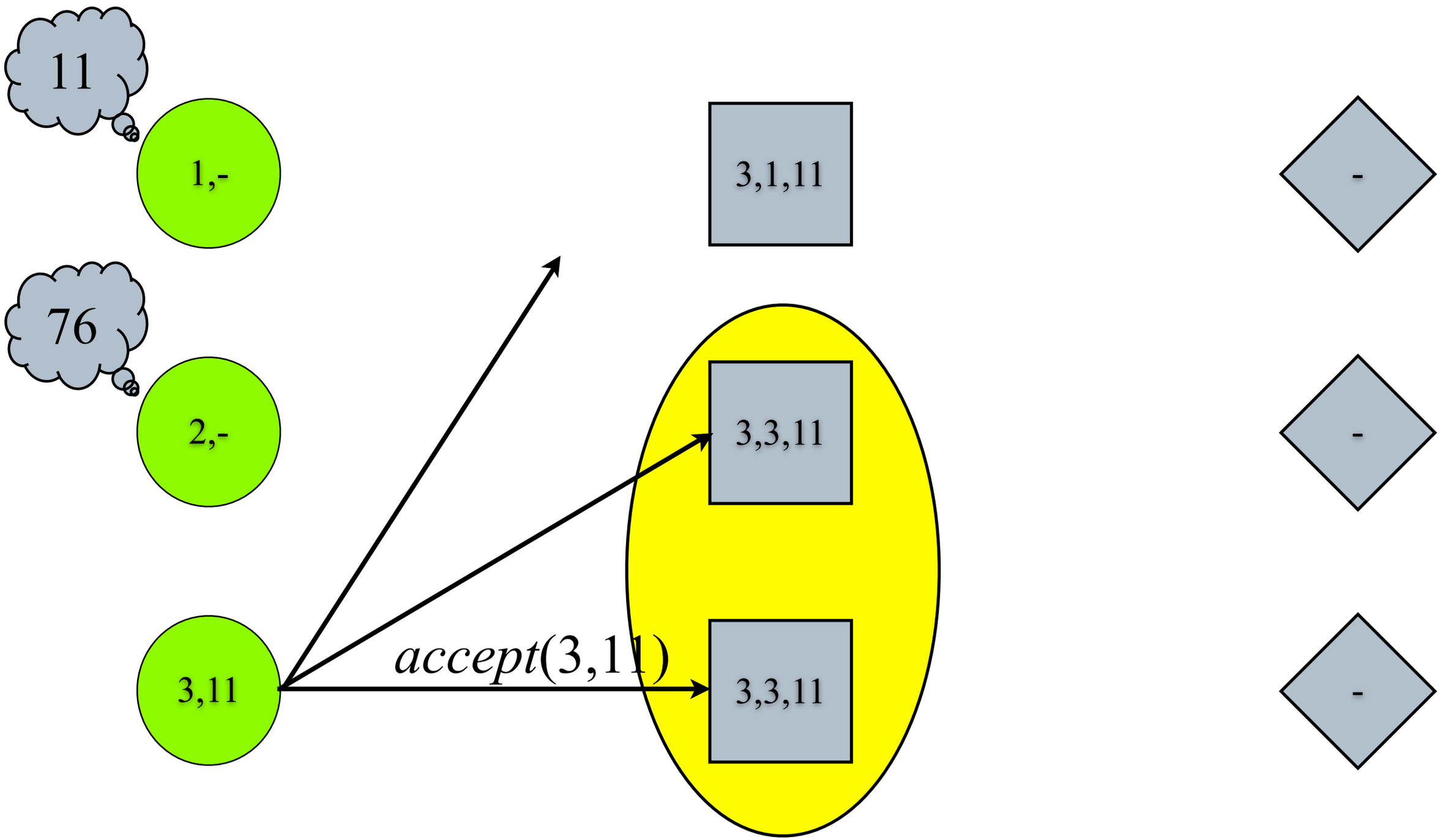
Problematic Paxos Execution

Phase 2a



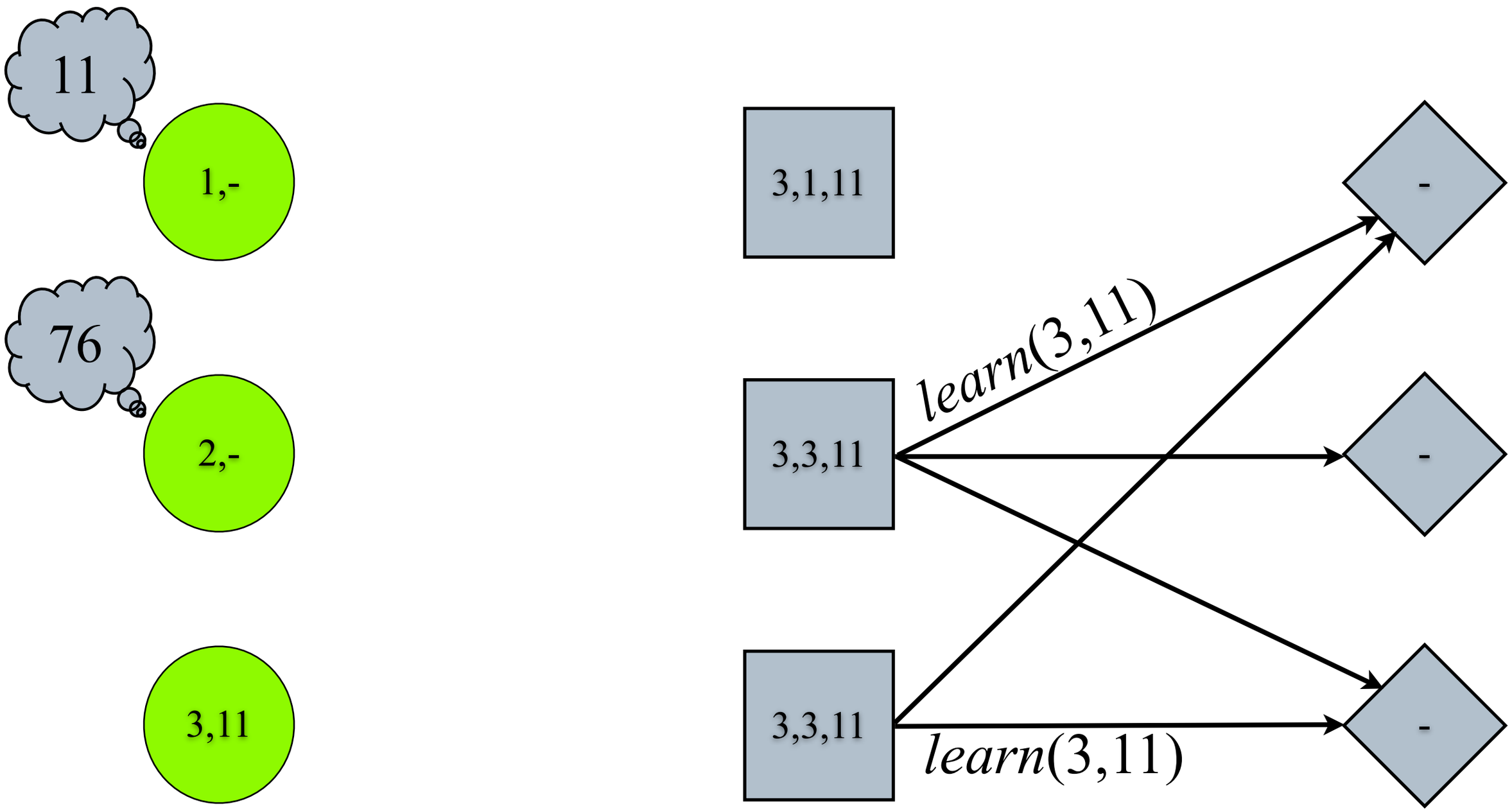
Problematic Paxos Execution

Phase 2a



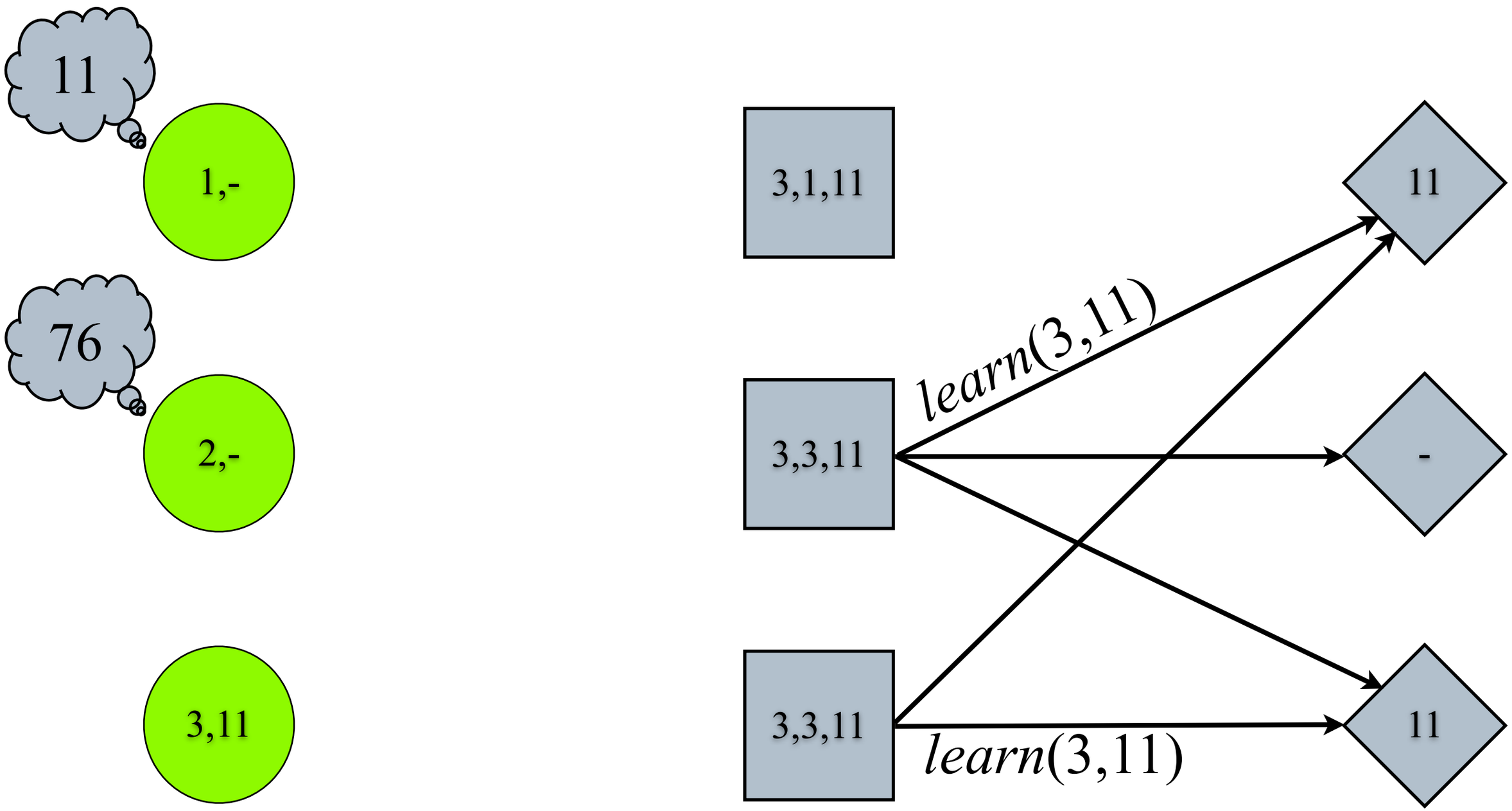
Problematic Paxos Execution

Phase 2b



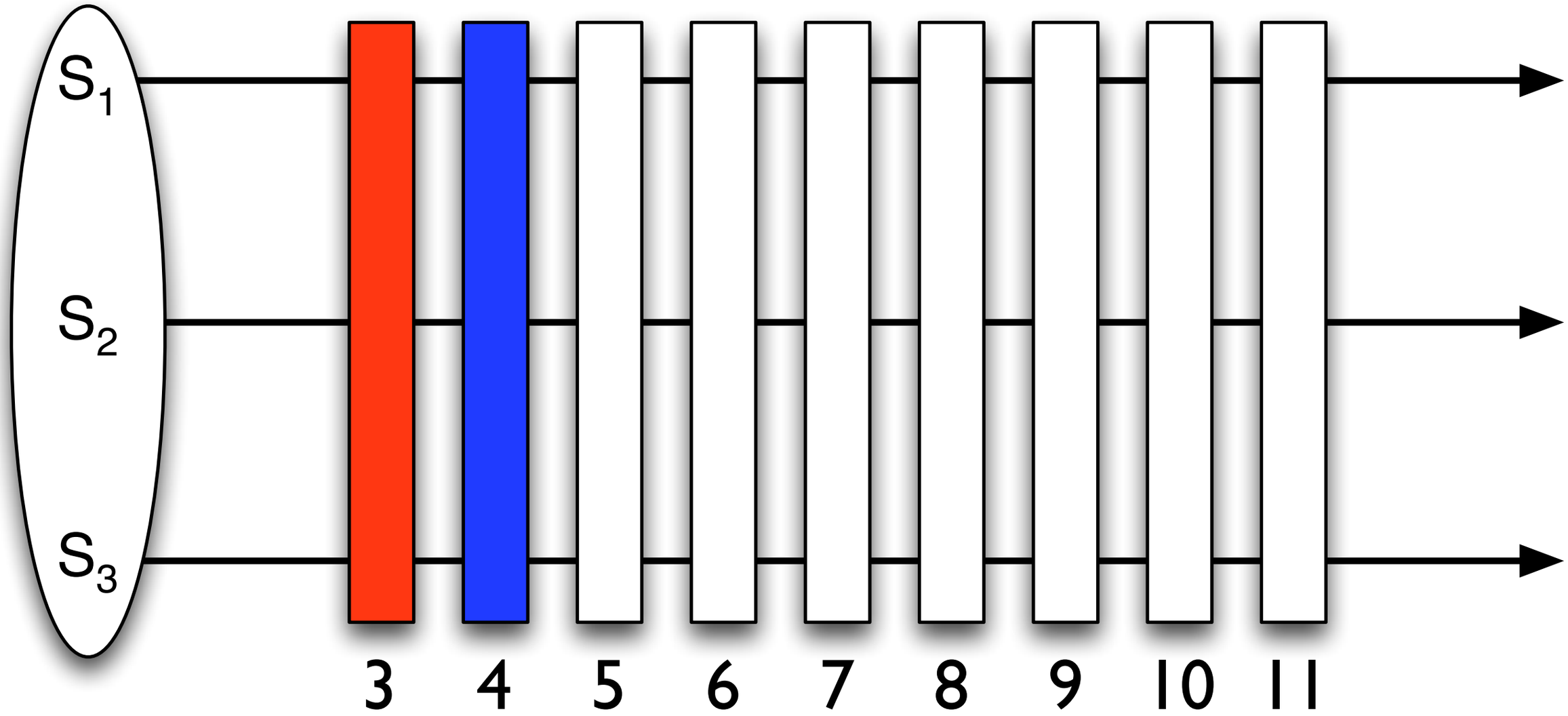
Problematic Paxos Execution

Phase 2b

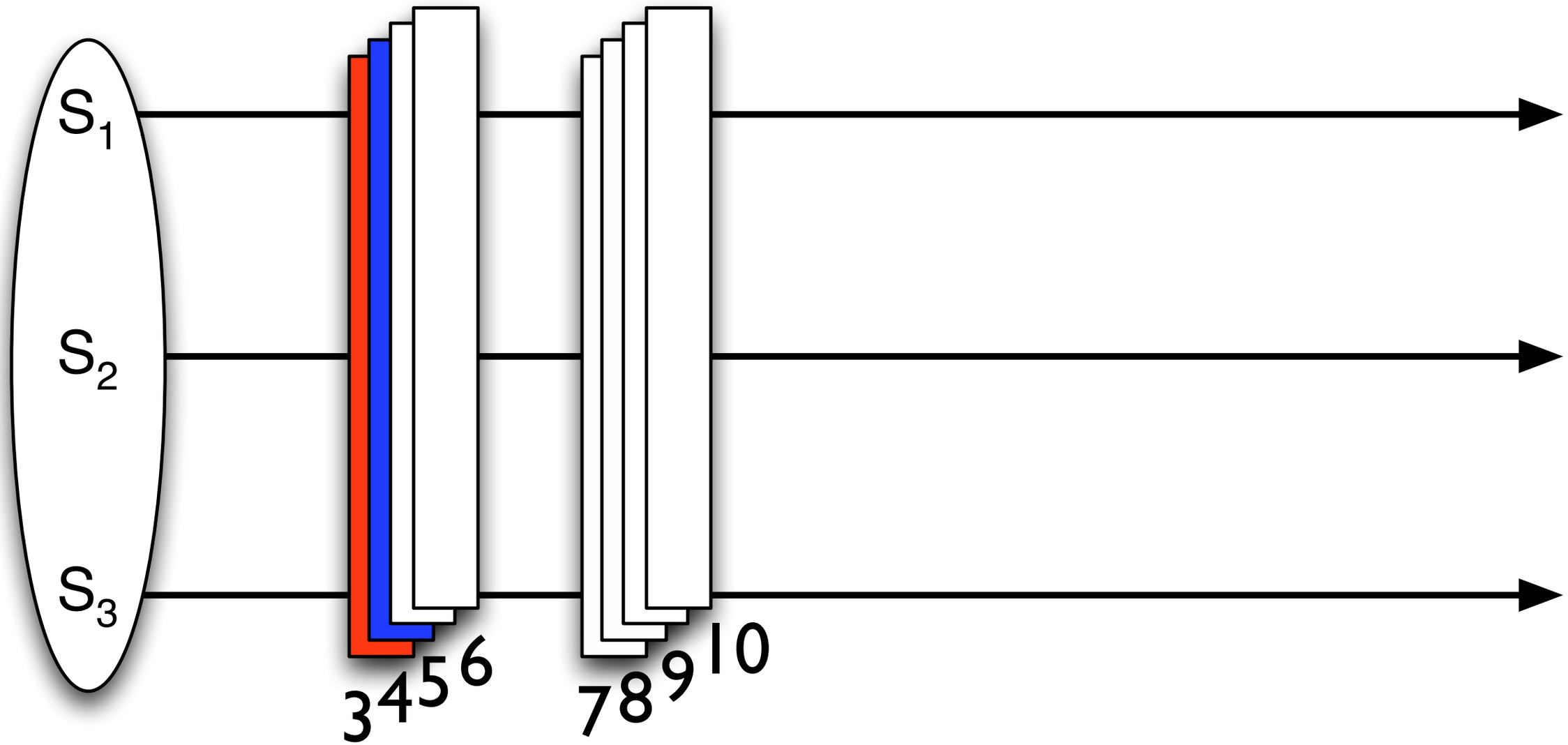


Slots and Concurrency (Pipelining)

Sequence of Slots

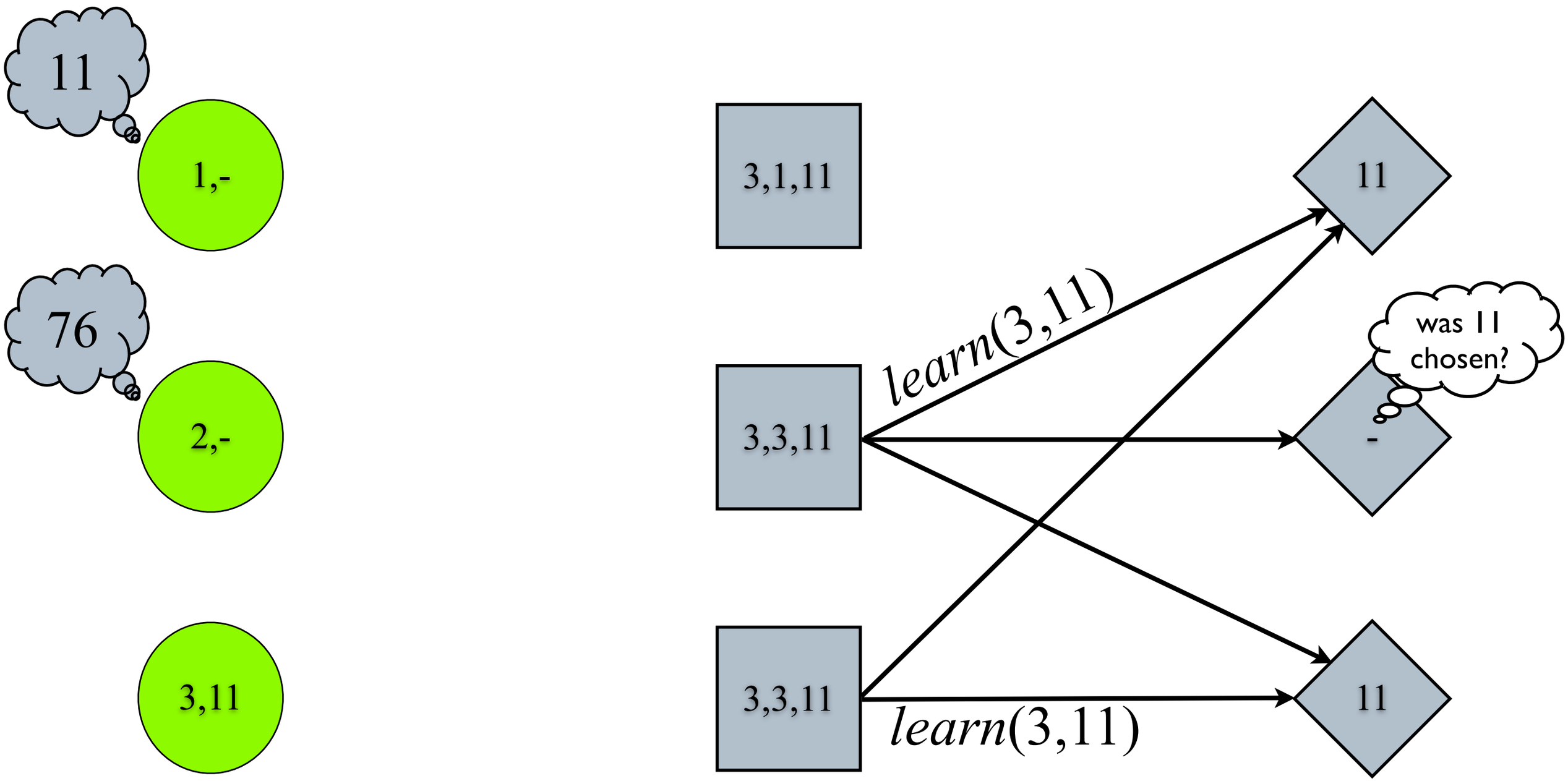


Multiple Slots Concurrently

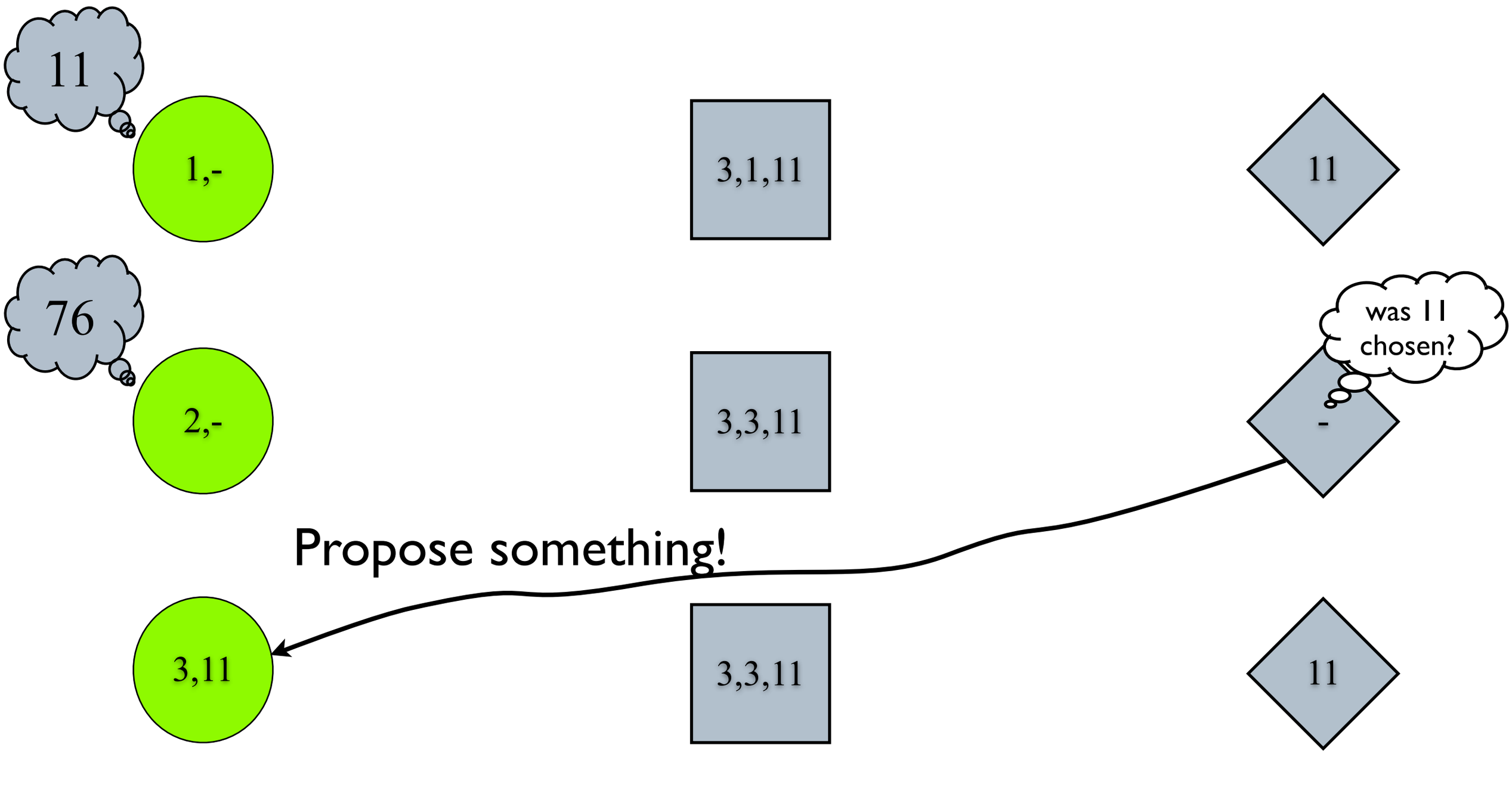


Missing Learns

Learner may not know that a value has been chosen

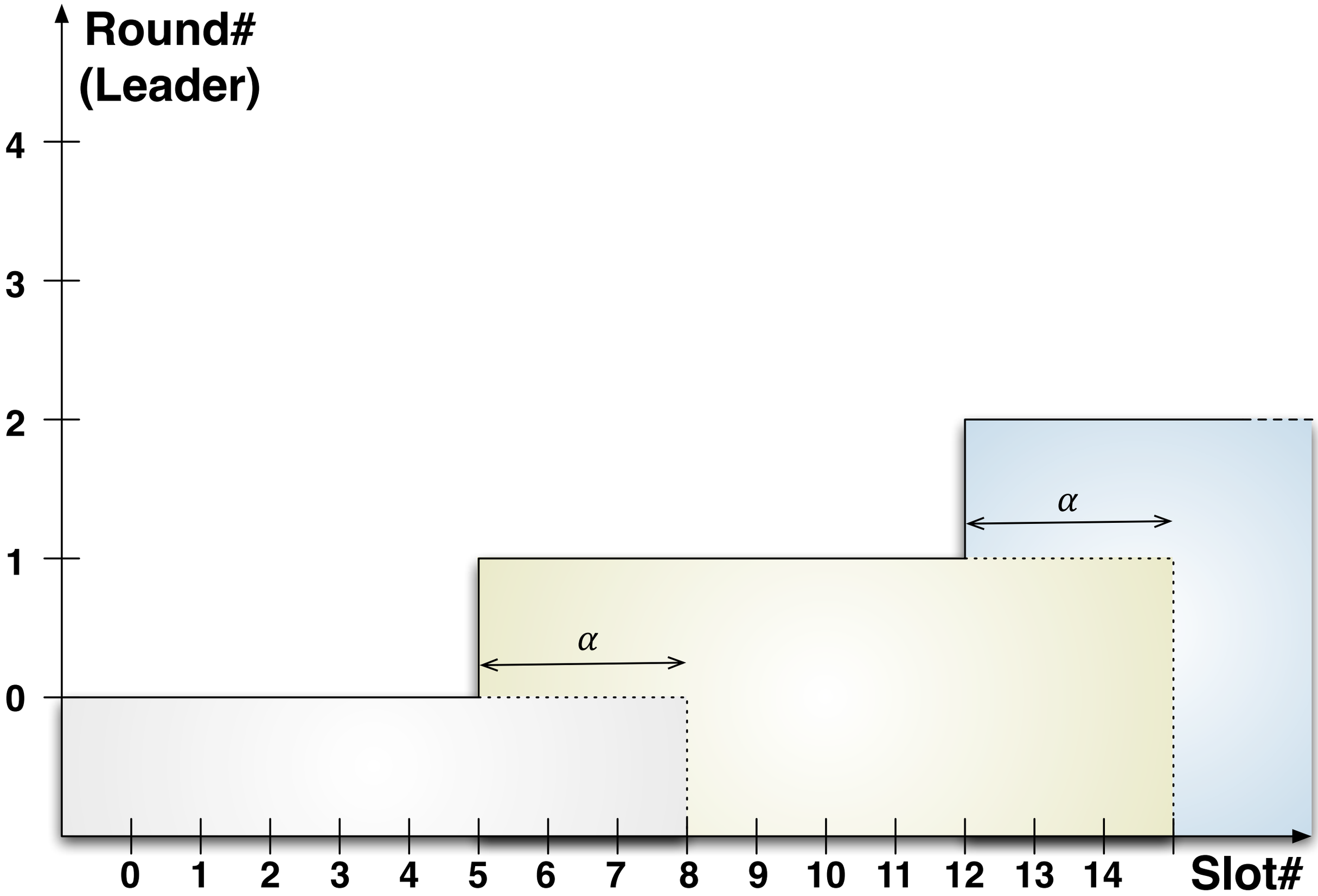


Learner may not know that a value has been chosen



Understanding Paxos

Rounds and Slots


Round#
(Leader)
4
3
2
1
0
α
α
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Slot#

Paxos Algorithm

Proposer

Algorithm 3 Classic Crash Paxos — Proposer c

Initialization:
A {Set of acceptors}
$crnd \leftarrow 0$ {Current round number}

PHASE 1a: Proposer c (Leader):
on $\langle \text{TRUST}, c \rangle$ from Ω_c {Ω_c indicates proposer c as the leader}
    $crnd \leftarrow \text{pickNext}(crnd)$ {Select proposal number larger than $crnd$}
    $MV \leftarrow \emptyset$ {Initialize set of (round, vote value) pairs}
    send $\langle \text{PREPARE}, crnd \rangle$ to A

PHASE 2a: Proposer c (Leader):
on $\langle \text{PROMISE}, rnd, vrnd, vval \rangle$ with $rnd = crnd$ from acceptor a
    $MV \leftarrow MV \cup (vrnd, vval)$ {Add value of acceptor a}
    if $|MV| \geq n_a - t_a$ **then** {Got promises from all correct acceptors?}
        if $(vrnd = \perp)\ \forall (vrnd, vval) \in MV$ **then** {No promises with a value?}
            $cval \leftarrow \text{pickAny}()$ {Propose any value}
        else
            $cval \leftarrow \text{pickLargest}(MV)$ {Pick proposed value $vval$ with largest $vrnd$}
        send $\langle \text{ACCEPT}, crnd, cval \rangle$ to A

Acceptor

Algorithm 4 Classic Crash Paxos — Acceptor

Initialization:
P {Set of proposers}
L {Set of learners}
$rnd \leftarrow 0$ {Current round number}
$vrnd \leftarrow \perp$ {Last voted round number}
$vval \leftarrow \perp$ {Value of last voted round}

PHASE 1b: Acceptor a:
on $\langle$PREPARE, $n\rangle$ with $n > rnd$ from proposer c
  $rnd \leftarrow n$ {The next round number}
  send $\langle$PROMISE, $rnd, vrnd, vval\rangle$ to c

PHASE 2b: Acceptor a:
on $\langle$ACCEPT, $n, v\rangle$ with $n \geq rnd \wedge n \neq vrnd$ from proposer c
  $rnd \leftarrow n, \quad vrnd \leftarrow n, \quad vval \leftarrow v$
  send $\langle$LEARN, $n, v\rangle$ to L

Glossary

- Proposers = Leaders
- Learners = Commanders
- Round = Ballot
- Slot = Consensus Instance
- Prepare = p1a msg
- Promise = p1b msg
- Accept = p2a msg
- Learn = p2b msg

Paxos Properties

Safety

Replicas always remain consistent with each other, no matter how many crashes occur.

Liveness

If a majority of replicas can communicate with each other, Paxos can make progress.

Summary

- Paxos needs
  - $2f+1$ replicas to tolerate f failures
    - Two communication steps
- It may not terminate, but it is always safe
- Can also be used for Atomic Commit

Questions?