Randomized Consensus

Mohsen Lesani

FLP Theorem

There is no deterministic consensus protocol for asynchronous distributed systems when even one process crashes.

Common Coin

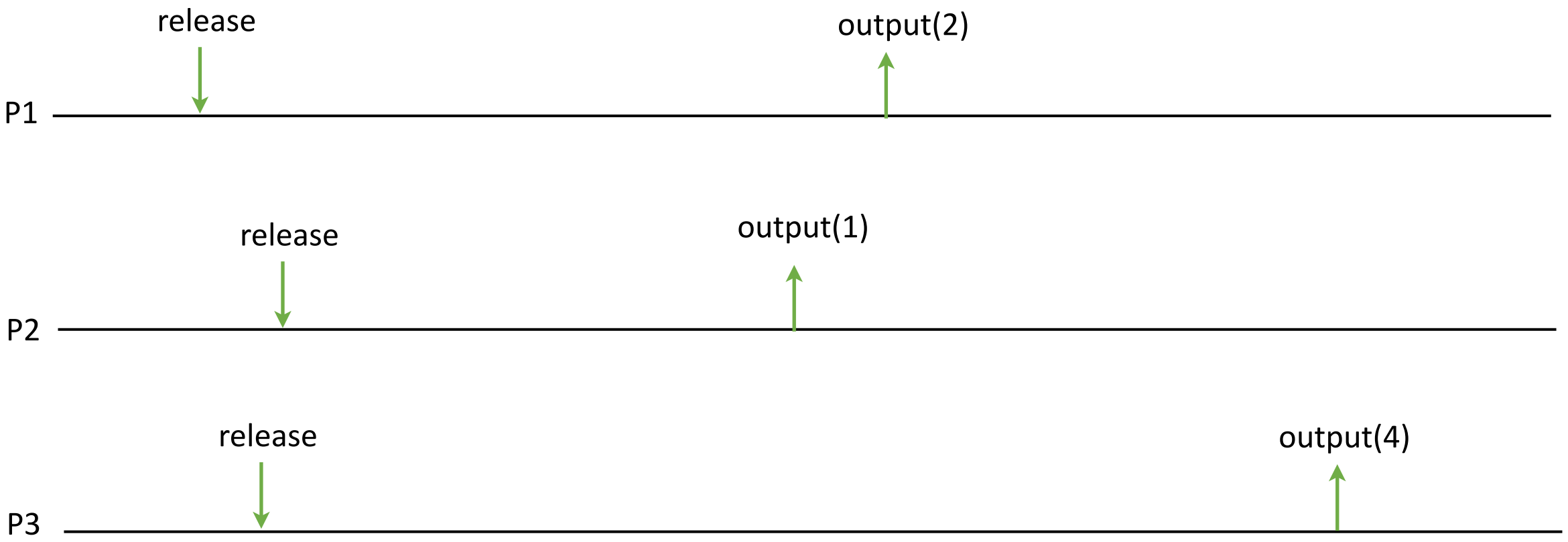
Random values from a domain D

- Request: ⟨ coin, release ⟩:

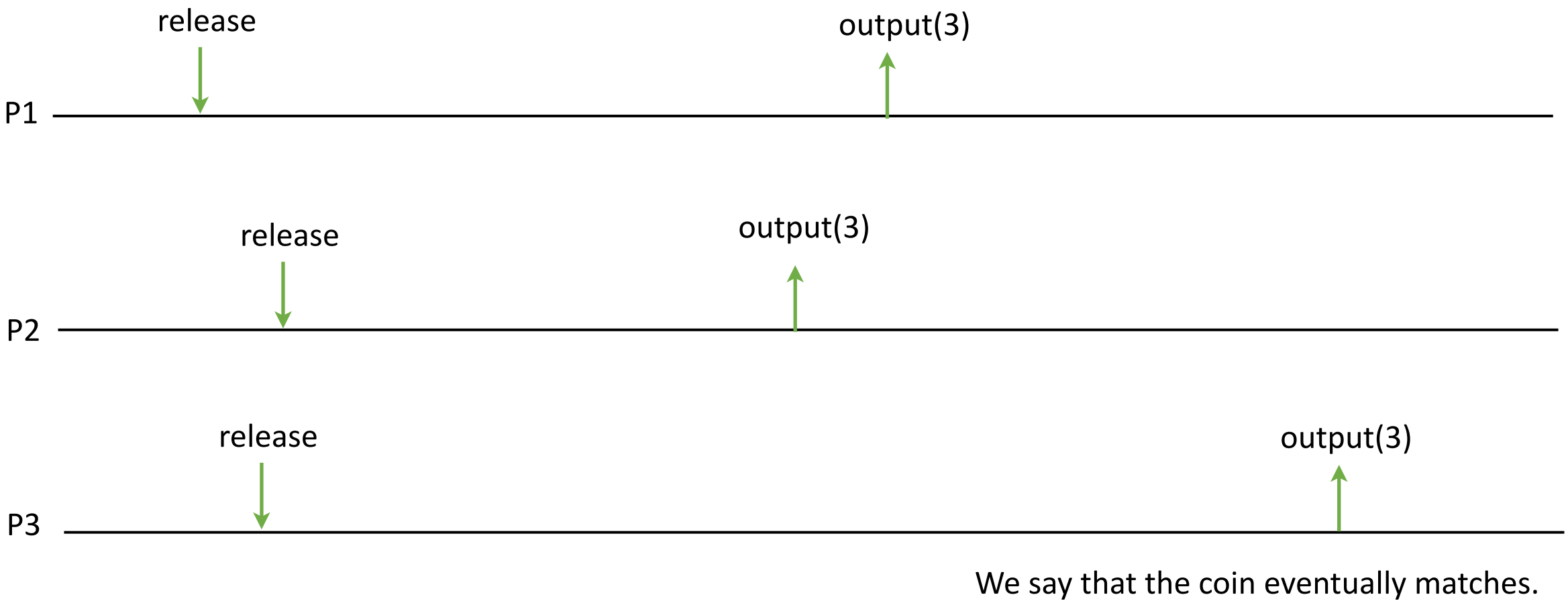
  Releases the coin.
- Indication: ⟨ coin, output(v) ⟩:

  Outputs the coin value v ∈ D.

Common Coin Example



Common Coin Example, Eventually



We say that the coin eventually matches.

Common Coin Specification

- Matching:
  With probability at least δ, every correct process outputs the same coin value.
- No bias:
  In the event that all correct processes output the same coin value, the distribution of the coin is uniform over D (i.e., a matching coin outputs any value in D with probability $1/|D|$).
- Termination:
  Every correct process eventually outputs a coin value.

- Other properties …

Common Coin Implementation

Independent Choice Coin:
Upon releasing the coin, every process simply outputs a value locally at random from D according to the uniform distribution.

If the domain is one bit then this realizes a δ = 2^{-N+1} matching common coin, because the probability that every process selects some b is 2^{-N}, for two bits $b = \{0, 1\}$.

Common Coin Implementation

Beacon Coin:
An external trusted process, called the beacon, periodically chooses an unpredictable random value and broadcasts it at predefined times.
When an algorithm uses a sequence of common coin abstractions, then for the k-th coin, every process outputs the k-th random value received from the beacon.

This coin always matches but it’s difficult to use it in an asynchronous system.

Randomized Consensus

- Agreement
- Integrity
- Validity
- **Probabilistic Termination:**
  Every correct process eventually decides some value with **probability 1**.

Observation

The common coin will eventually output the same value for all processes.

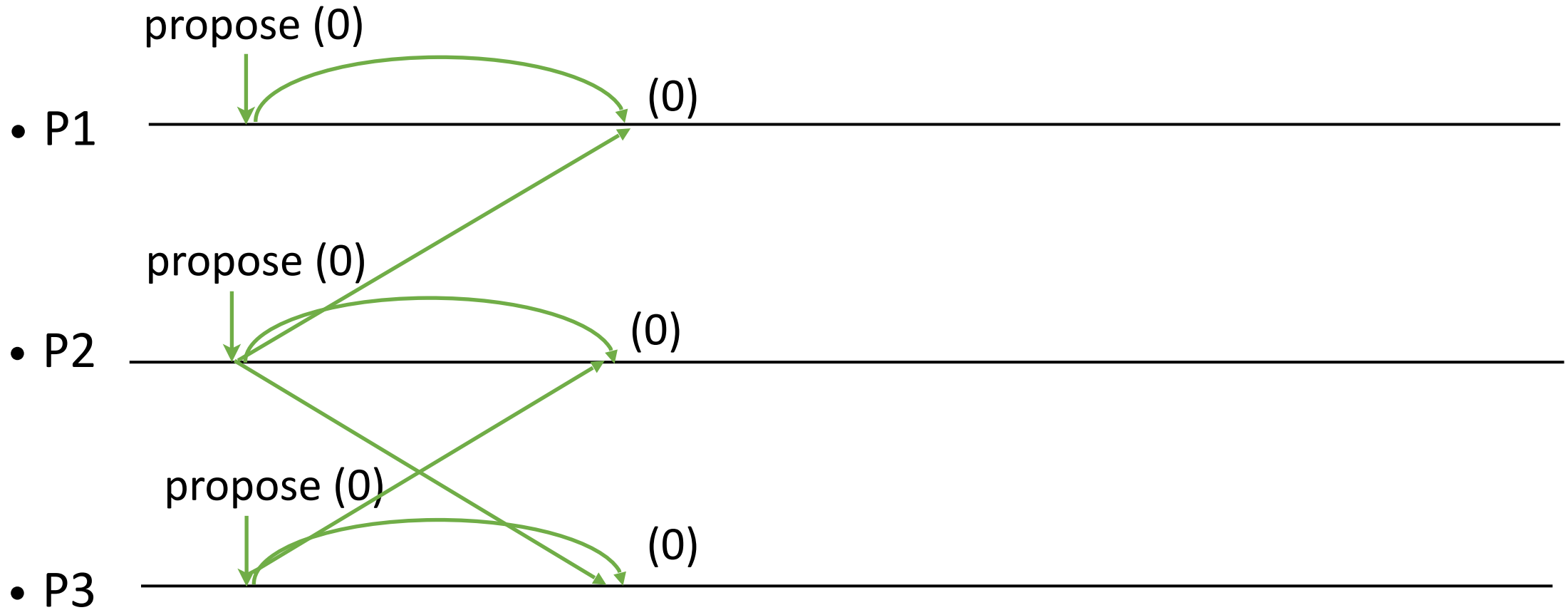
We need to design a consensus protocol that decides when all processes propose the same value.

Broadcast and wait for a quorum

Eventually all the proposals are the same.

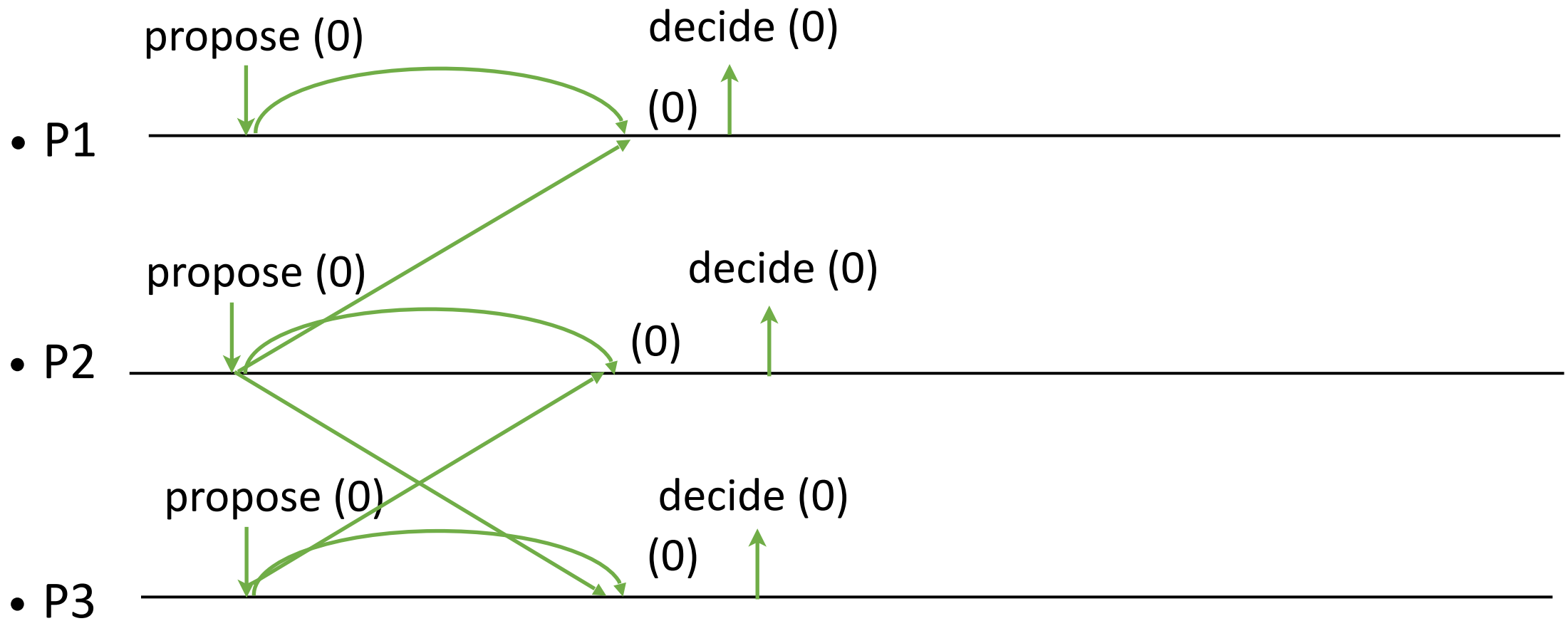
Broadcast and wait for a quorum

Eventually all the proposals are the same.



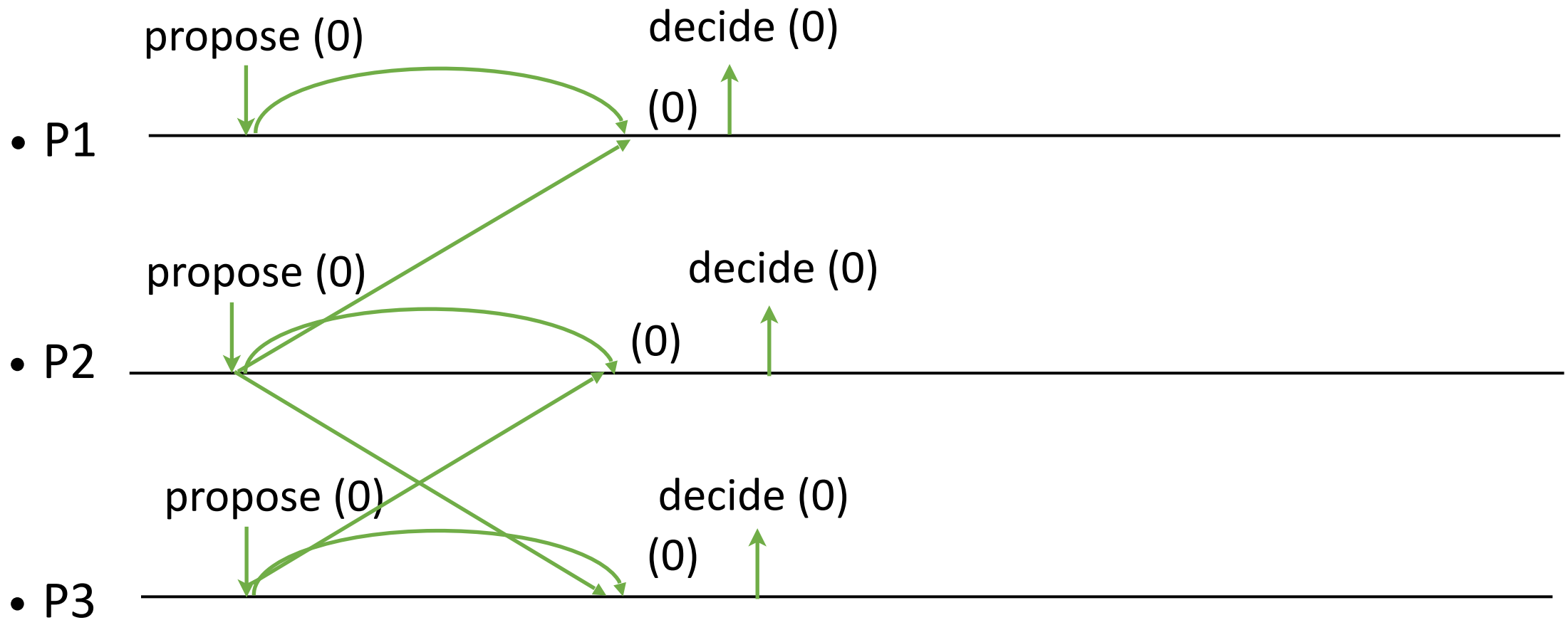
Broadcast and wait for a quorum

Eventually all the proposals are the same.



Broadcast and wait for a quorum

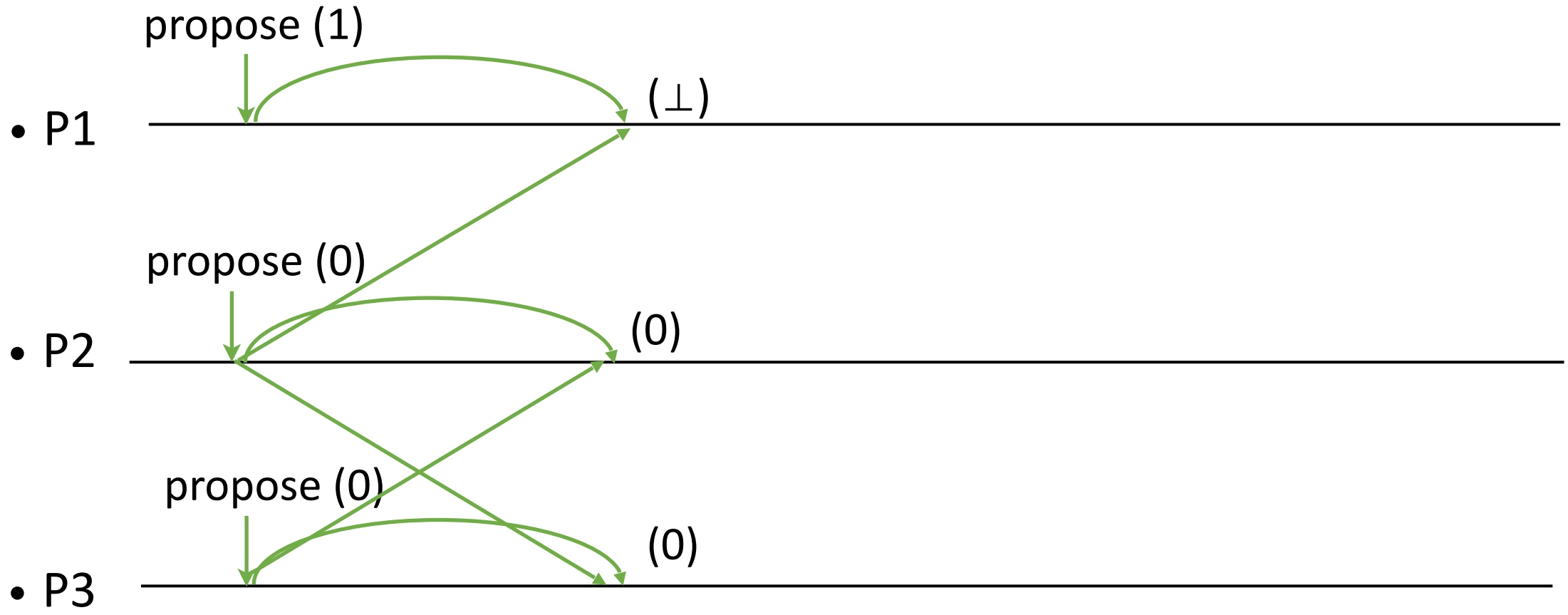
Eventually all the proposals are the same.



Does it have safety and liveness when proposals are different?

Broadcast and wait for a quorum

The proposals are different.

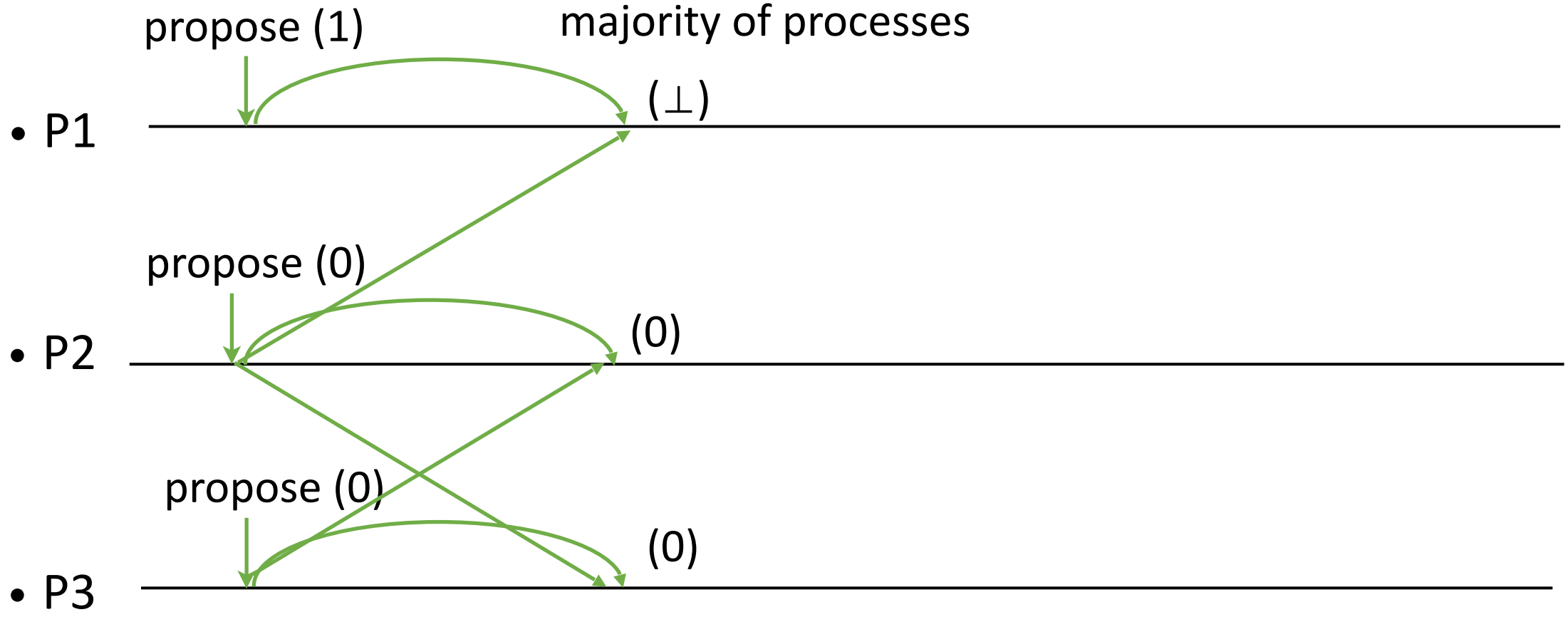


Broadcast and wait for a quorum

The proposals are different.

A majority (quorum) of processes are correct.

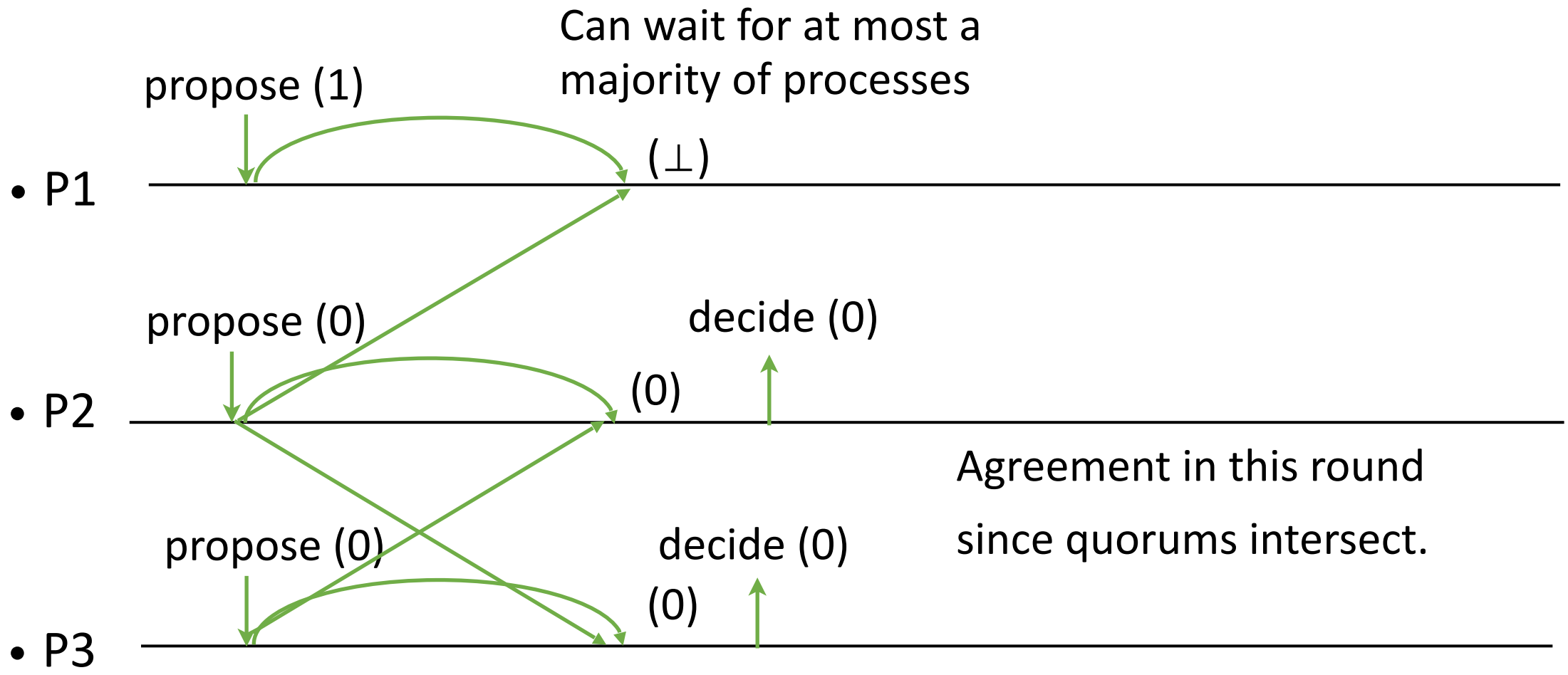
Can wait for at most a majority of processes



Broadcast and wait for a quorum

The proposals are different.

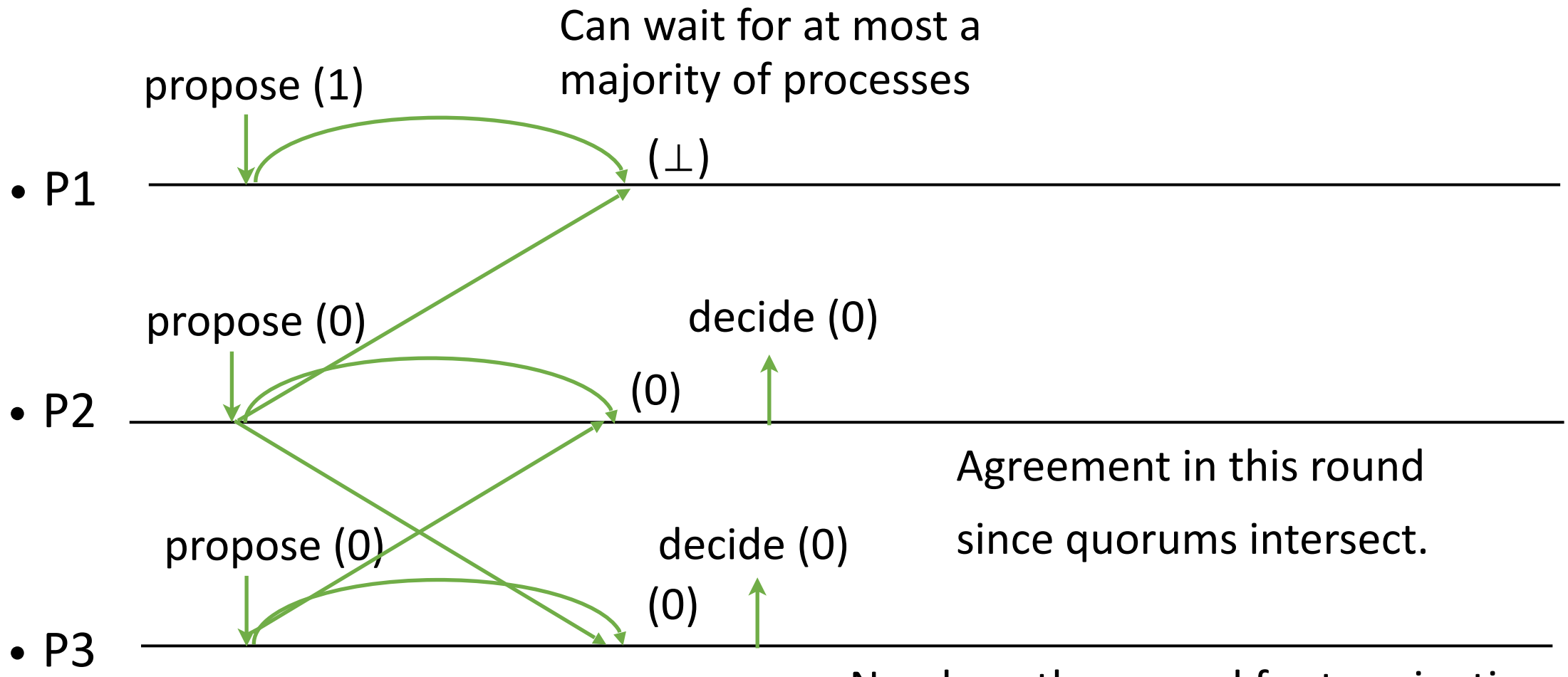
A majority (quorum) of processes are correct.



Broadcast and wait for a quorum

The proposals are different.

A majority (quorum) of processes are correct.

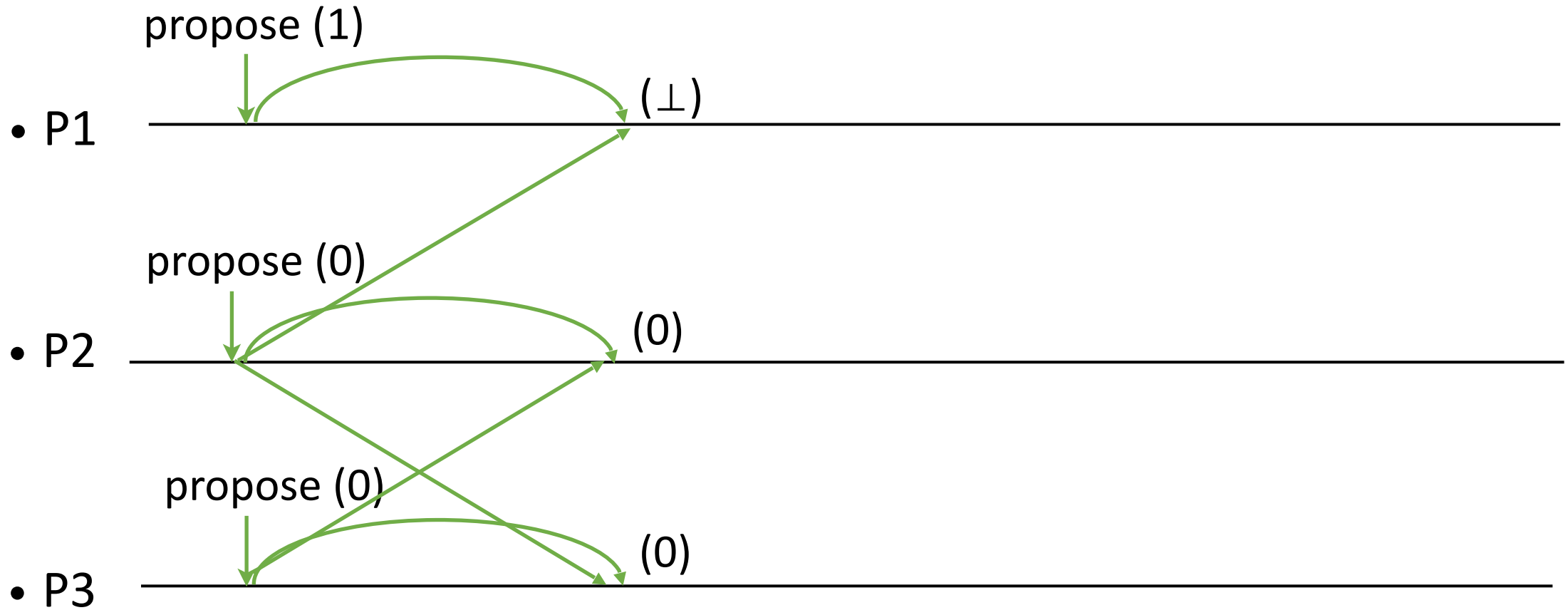


Need another round for termination

What about agreement across rounds?

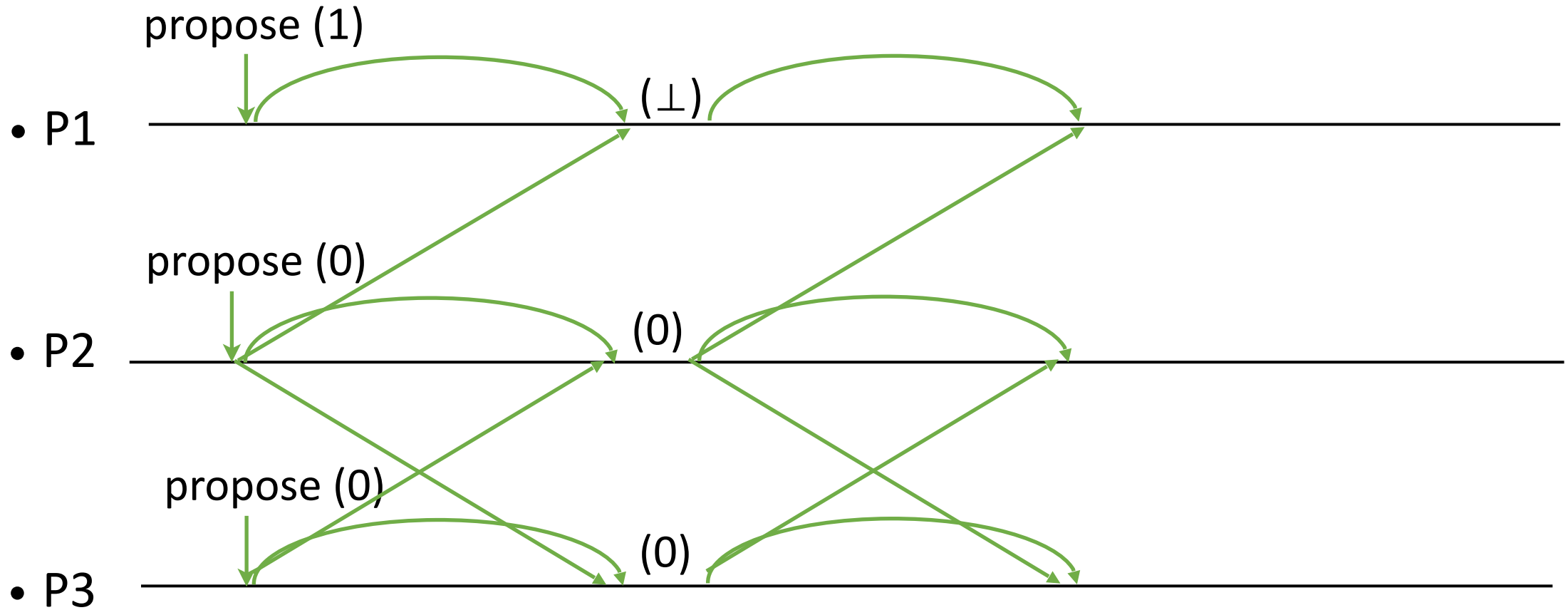
Adopt the decided value for next round

Second round to make others adopt



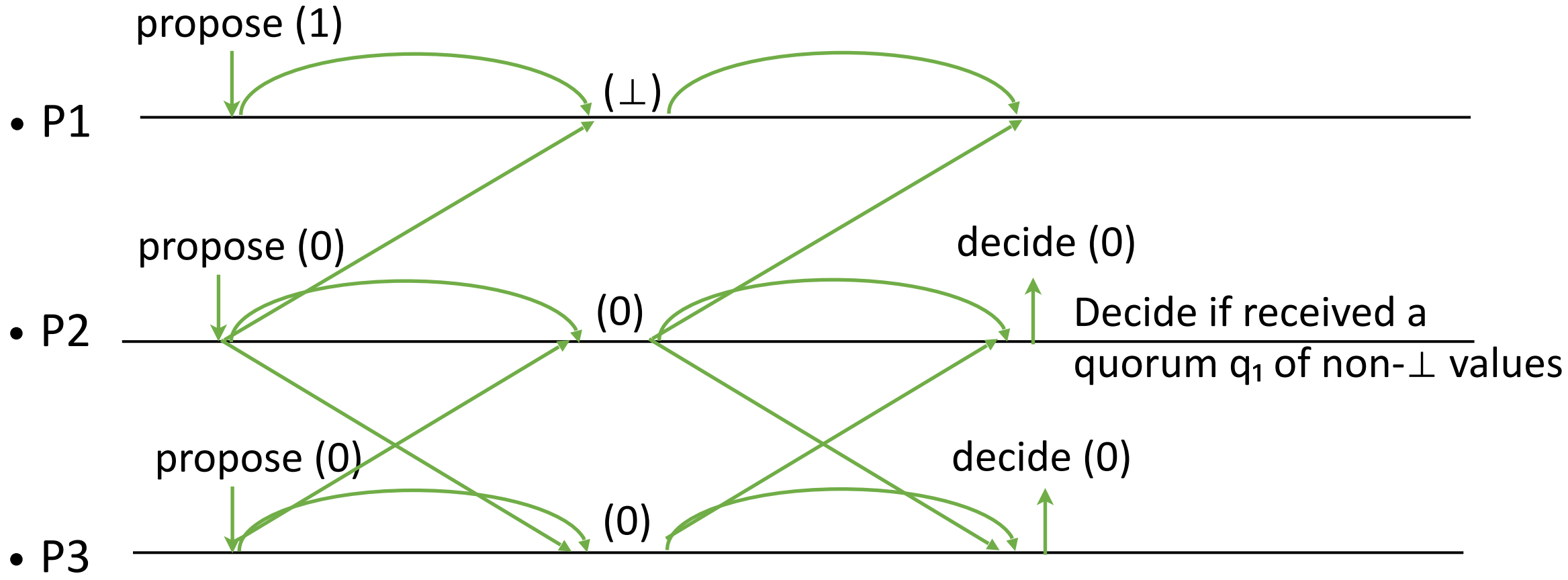
Adopt the decided value for next round

Second round to make others adopt



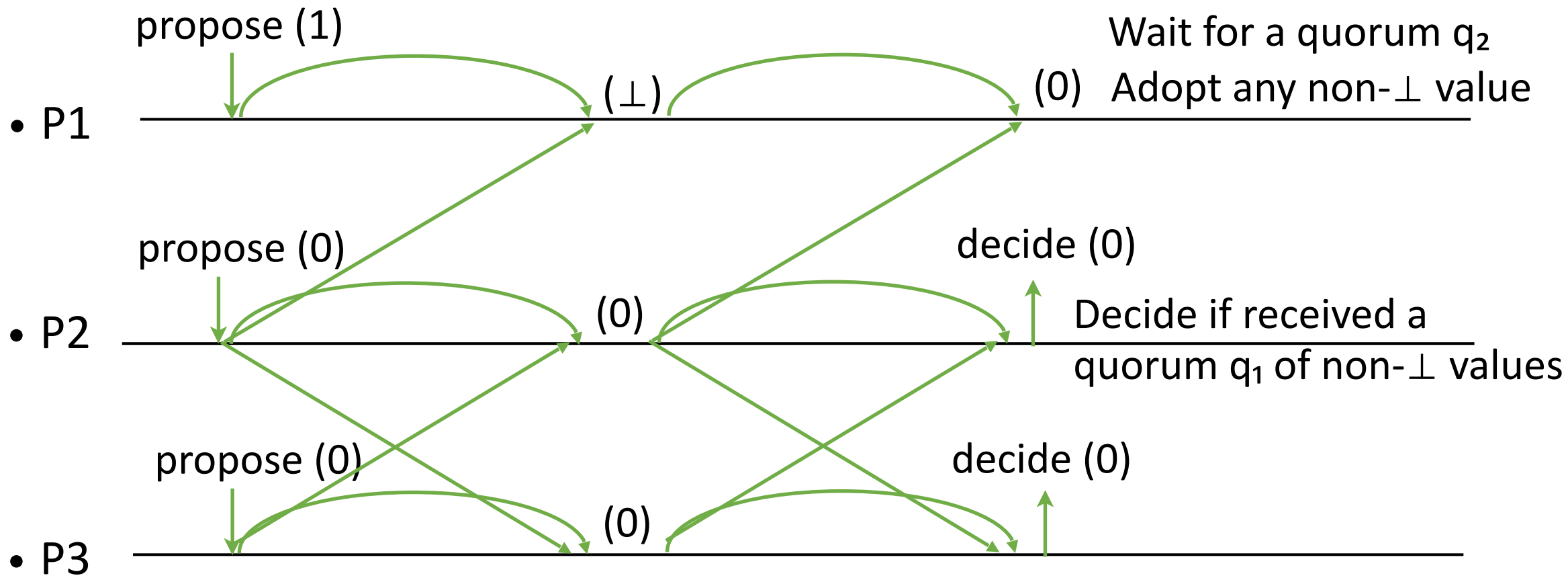
Adopt the decided value for next round

Second round to make others adopt



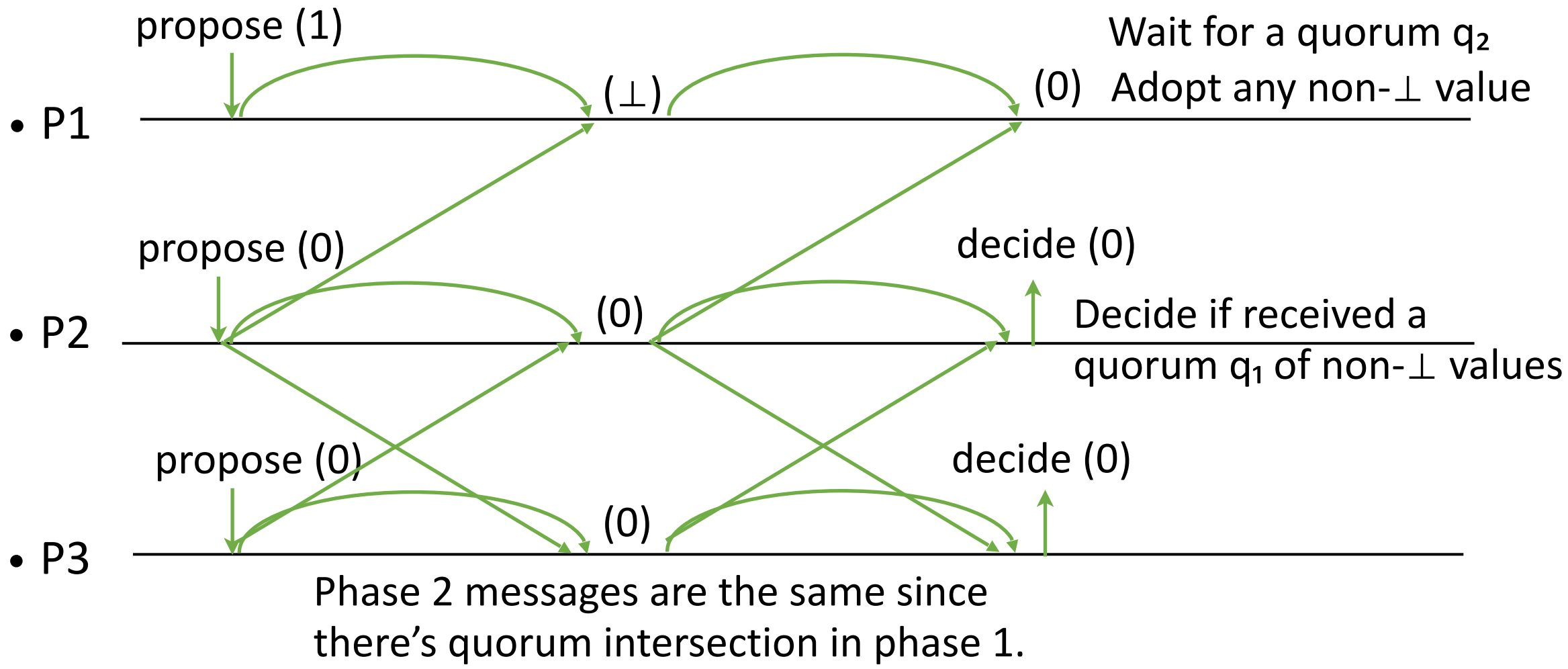
Adopt the decided value for next round

Second round to make others adopt



Adopt the decided value for next round

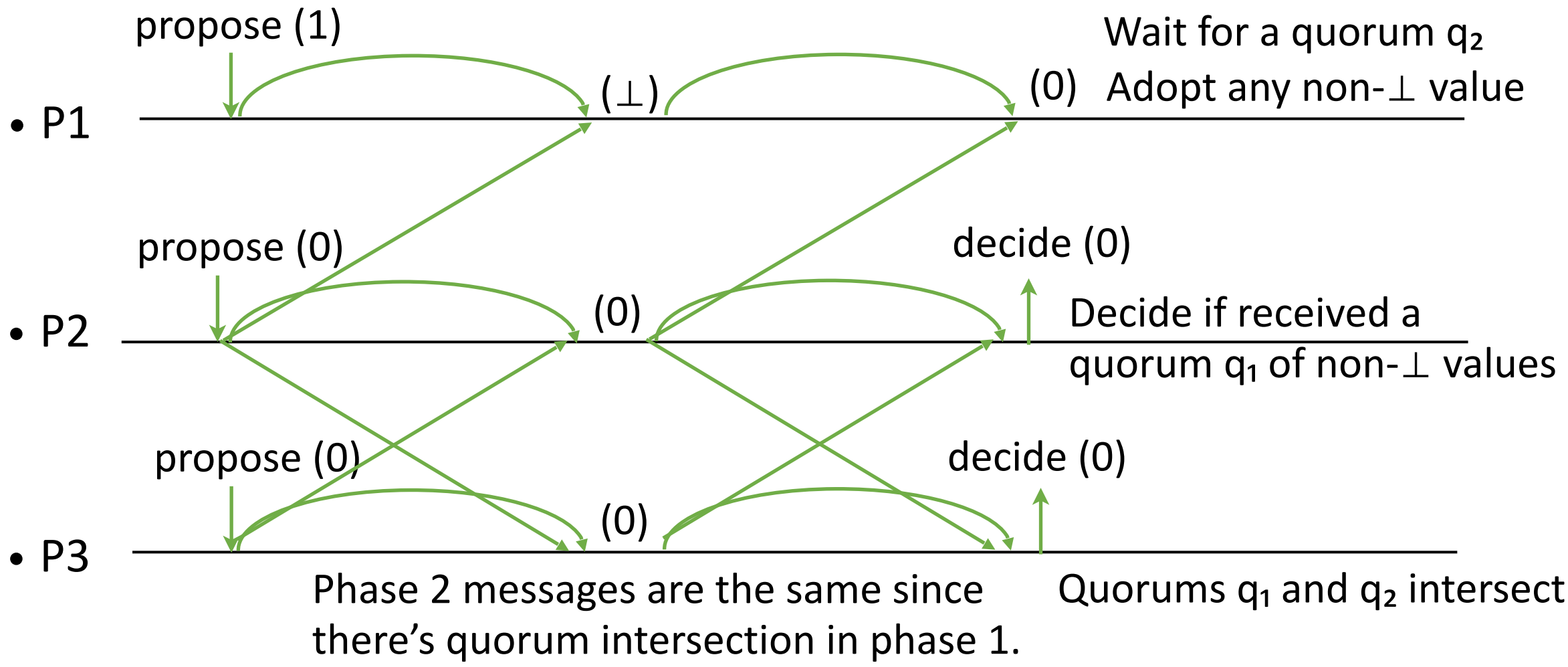
Second round to make others adopt



Phase 2 messages are the same since there's quorum intersection in phase 1.

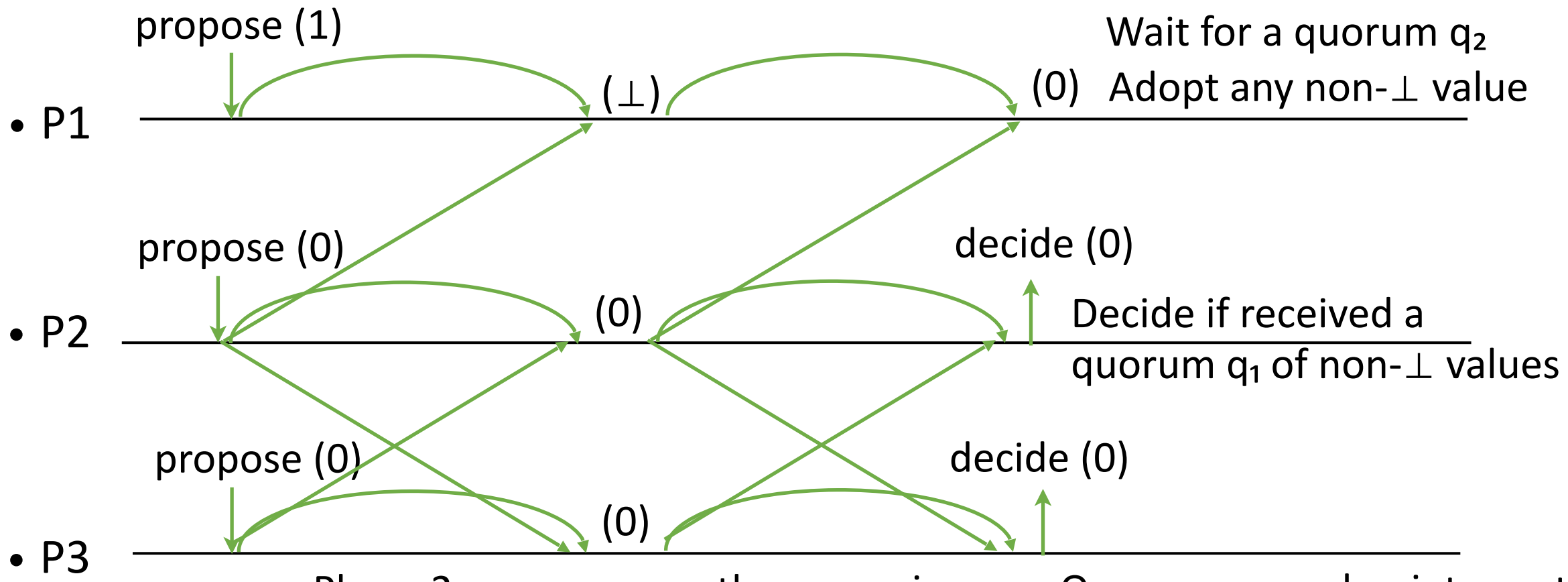
Adopt the decided value for next round

Second round to make others adopt



Adopt the decided value for next round

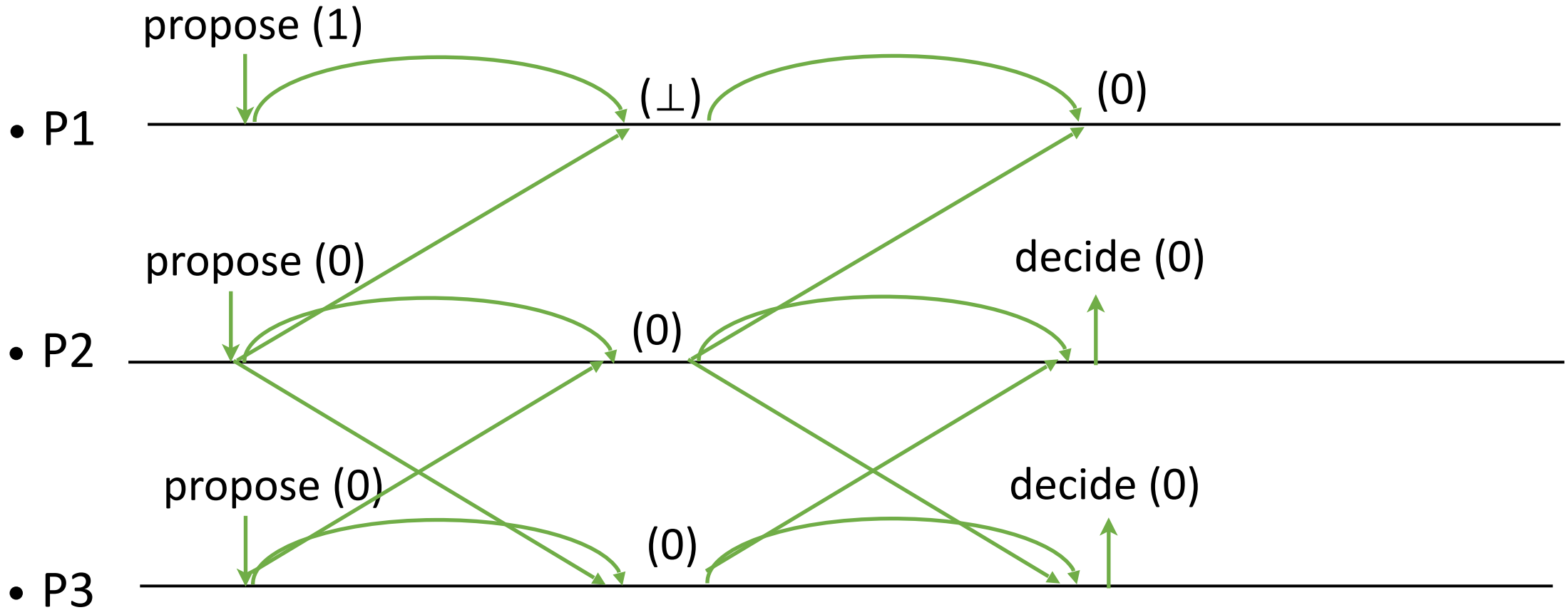
Second round to make others adopt



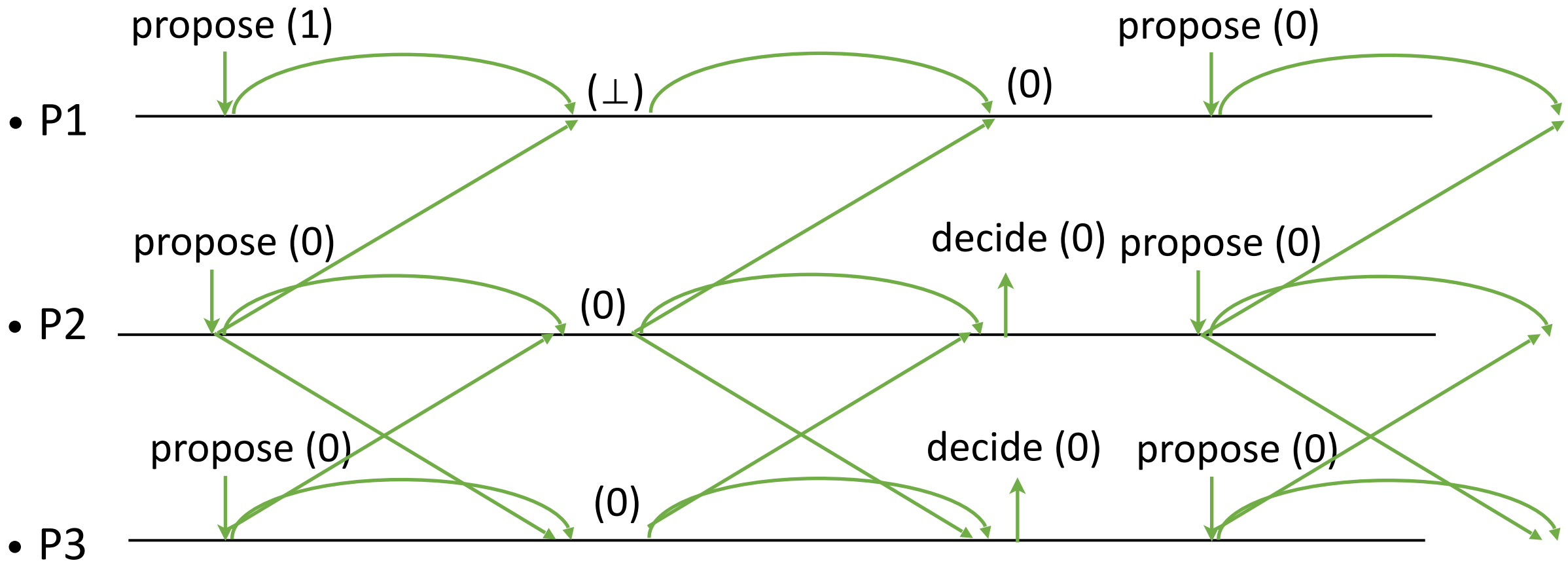
Phase 2 messages are the same since there's quorum intersection in phase 1.

Quorums q_1 and q_2 intersect
So P1 will adopt **the** value.

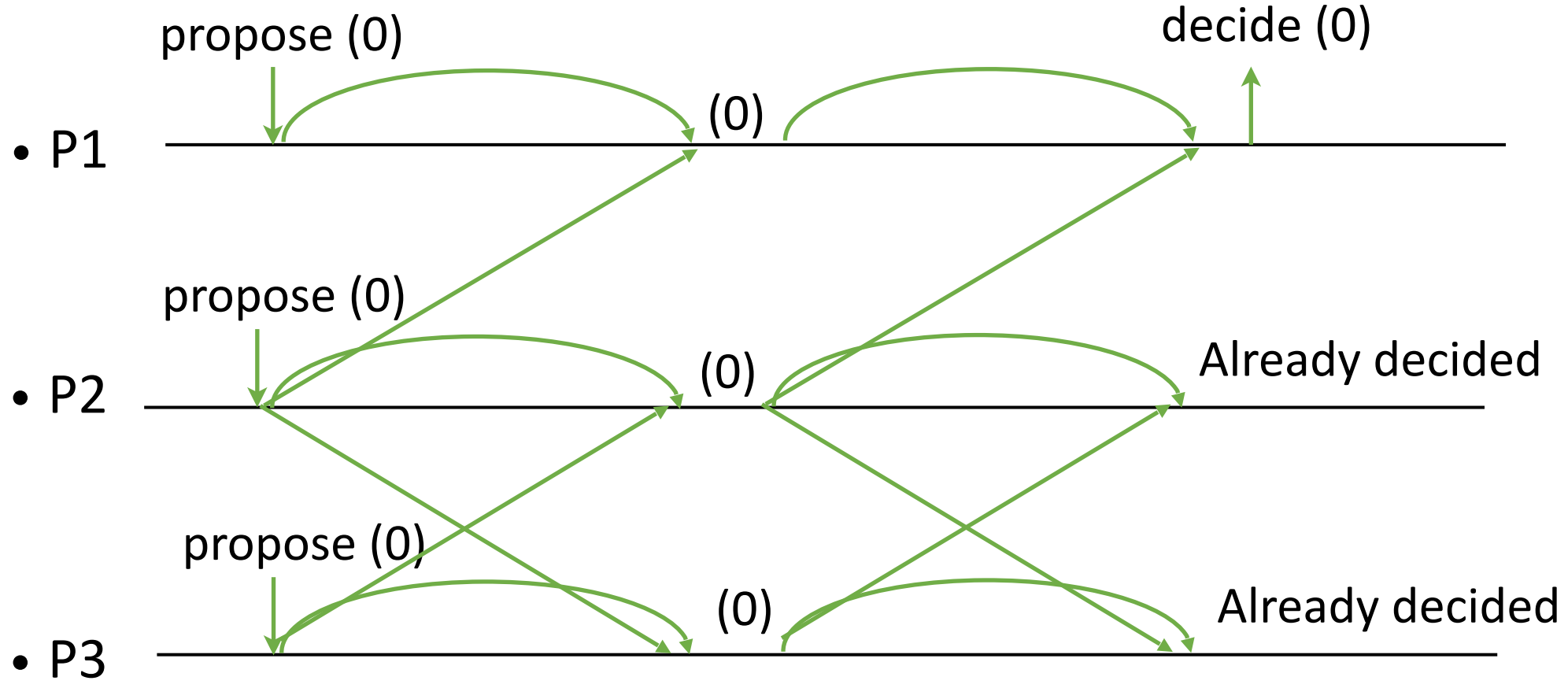
Next round



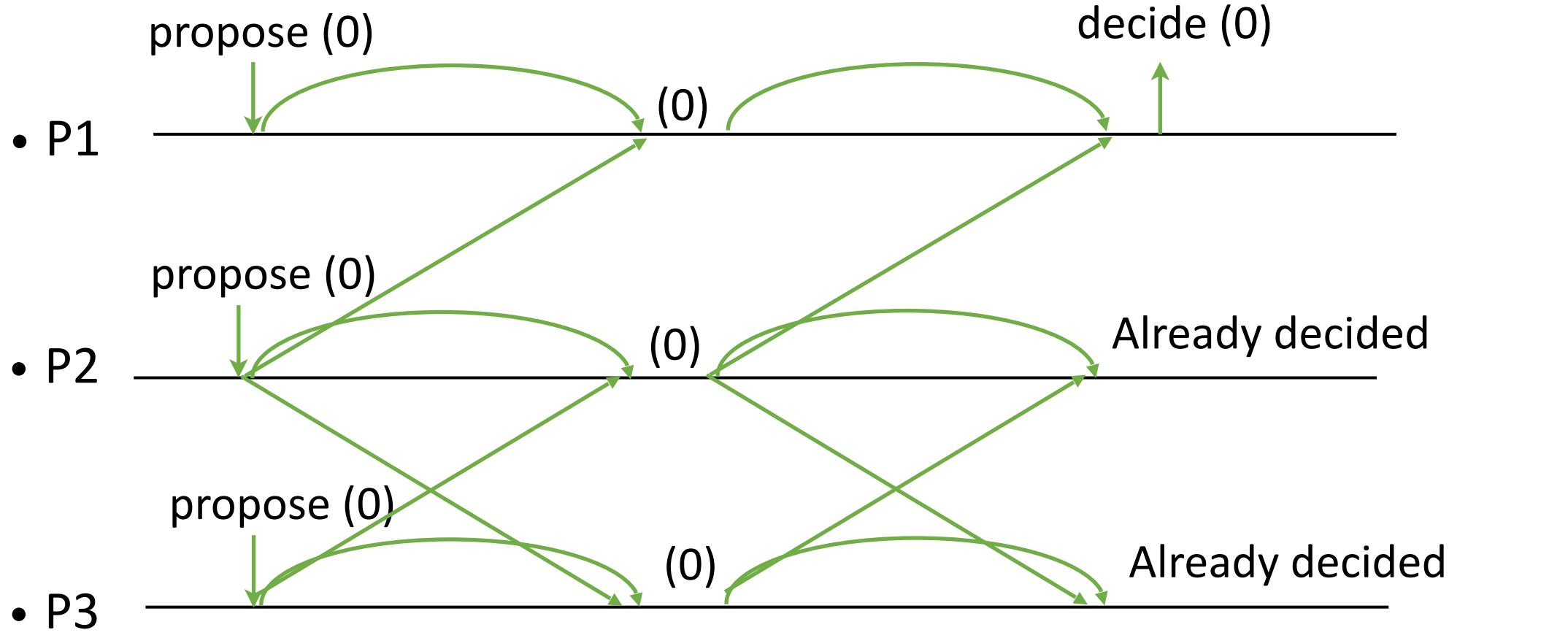
Next round


propose (1)
propose (0)
(⊥)
(0)
• P1
propose (0)
decide (0)
propose (0)
(0)
• P2
propose (0)
decide (0)
propose (0)
(0)
• P3

Next round

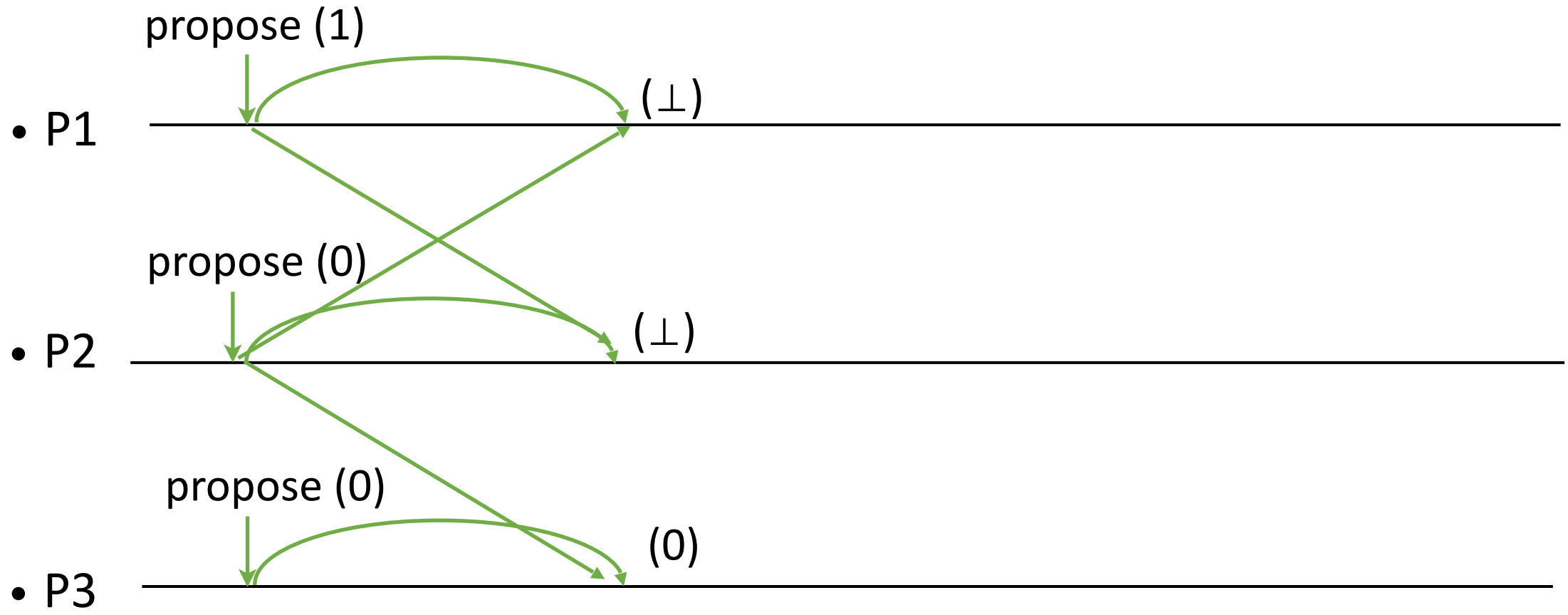

propose (0)
decide (0)
(0)
P1
propose (0)
Already decided
(0)
P2
propose (0)
(0)
Already decided
P3

Next round

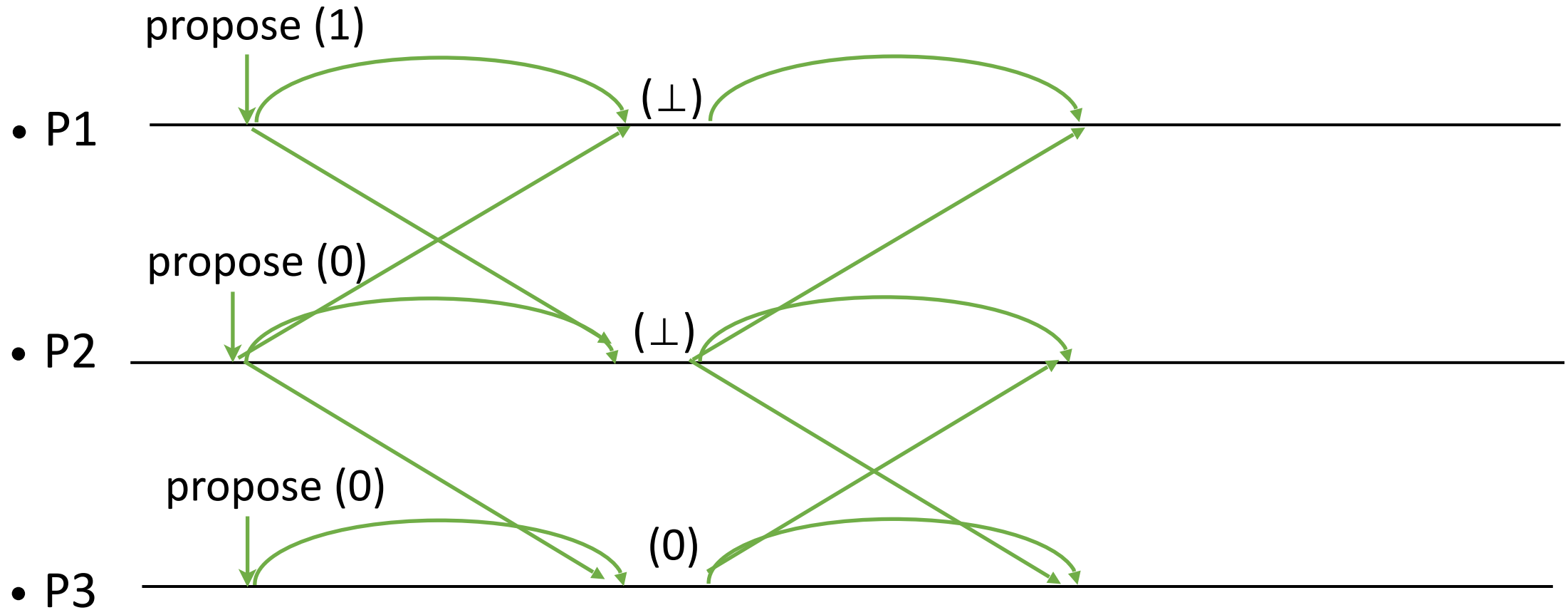


In the next round, all the remaining correct processes decide.

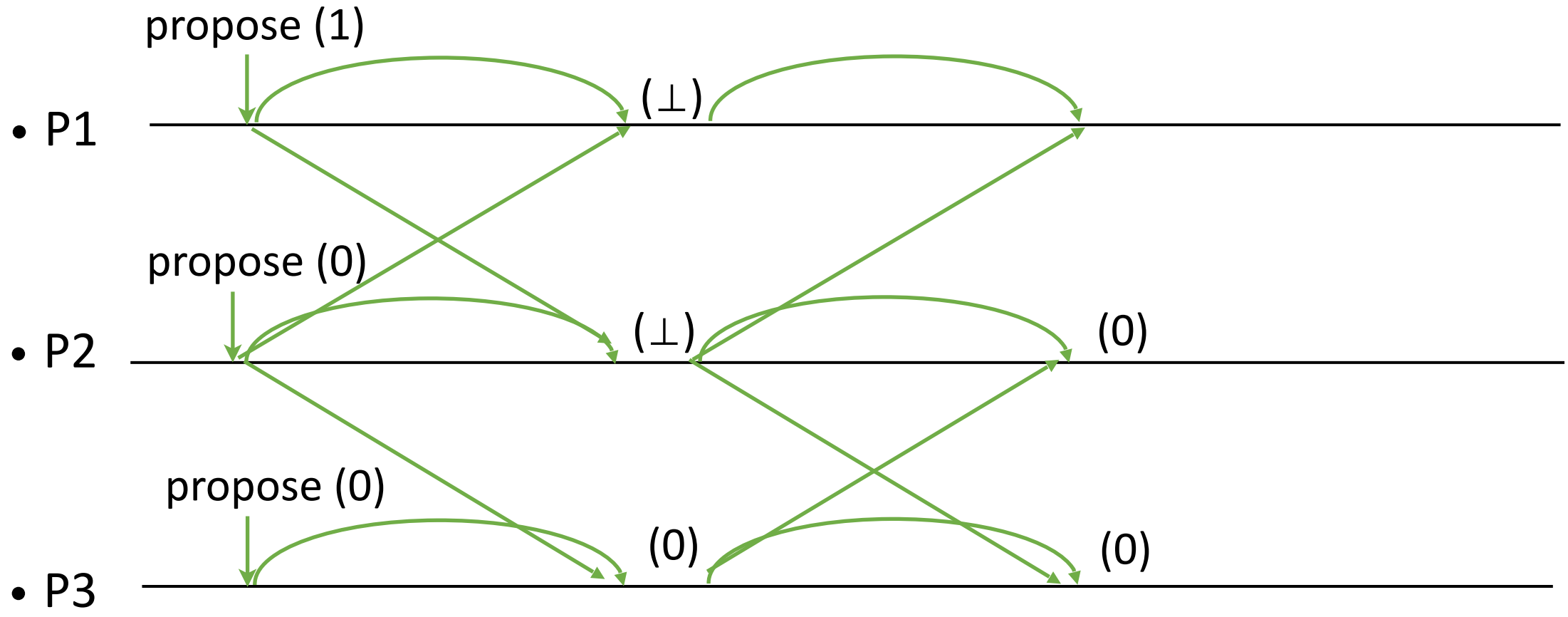
If not received non-⊥ value, get value from the coin for the next round



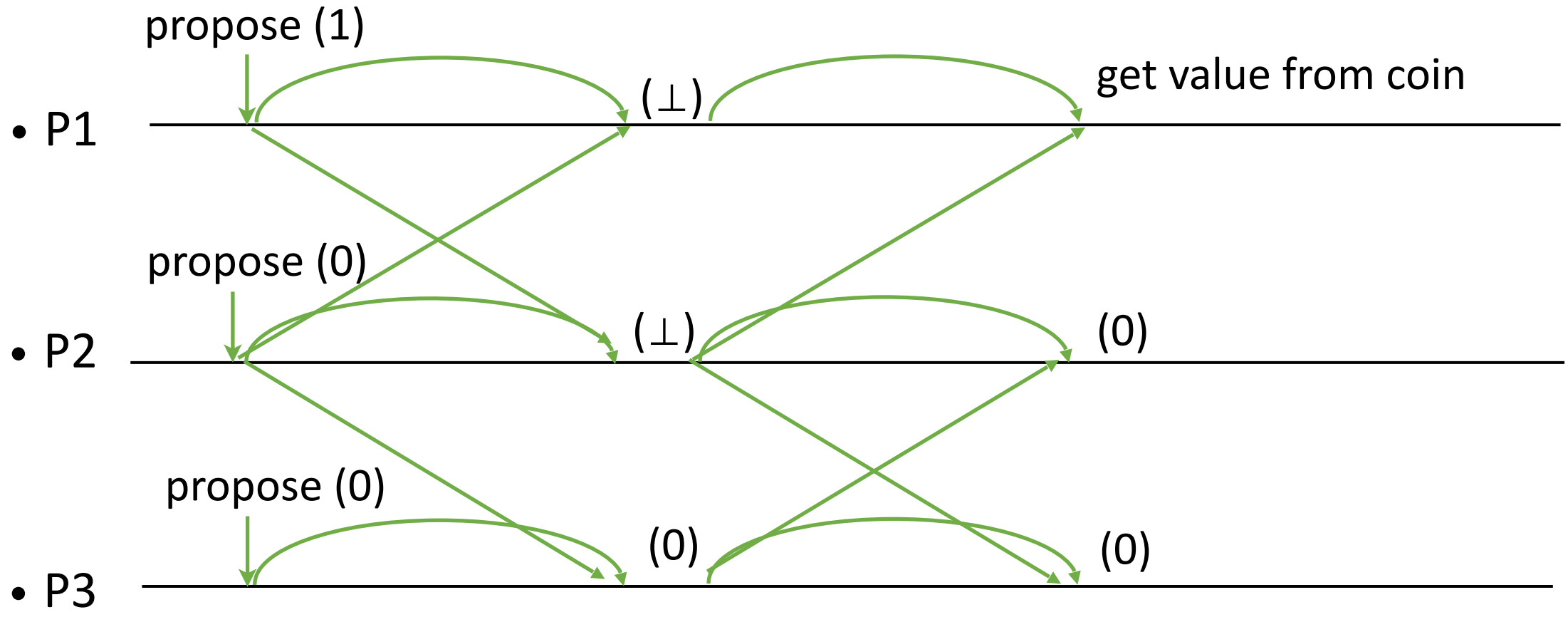
If not received non-⊥ value, get value from the coin for the next round



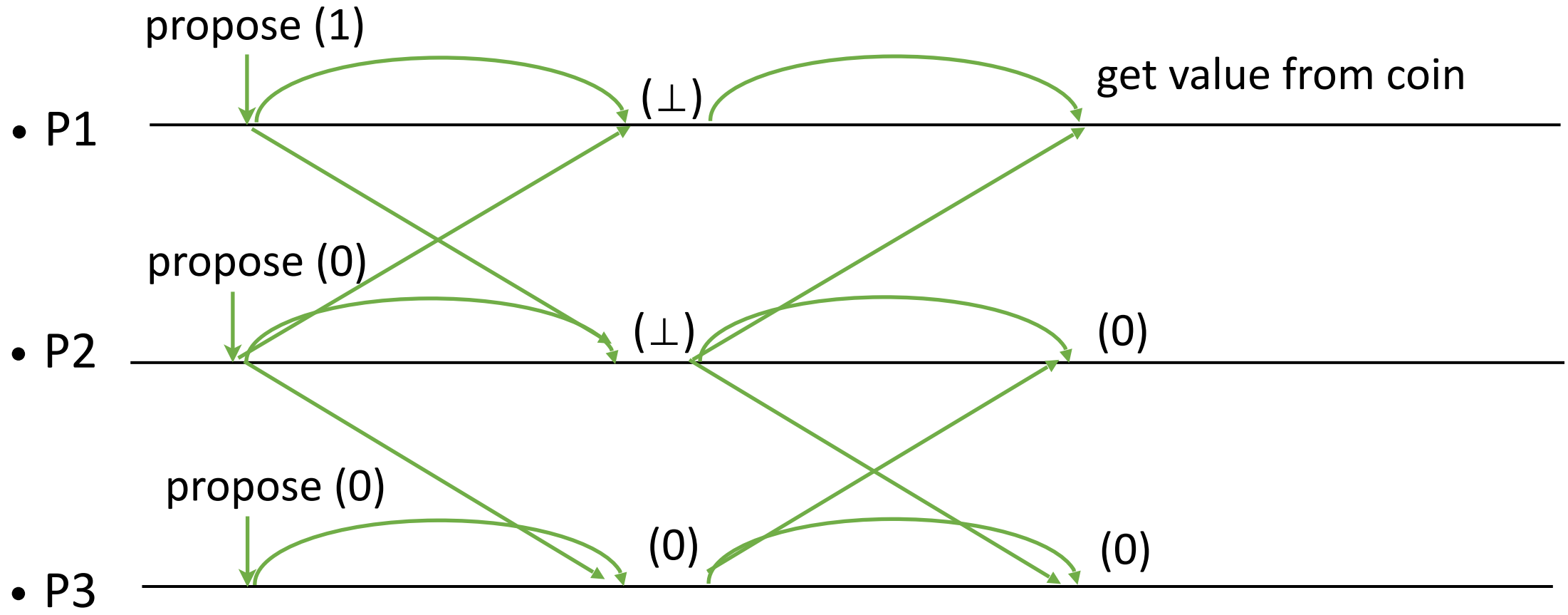
If not received non-⊥ value, get value from the coin for the next round



If not received non-⊥ value, get value from the coin for the next round



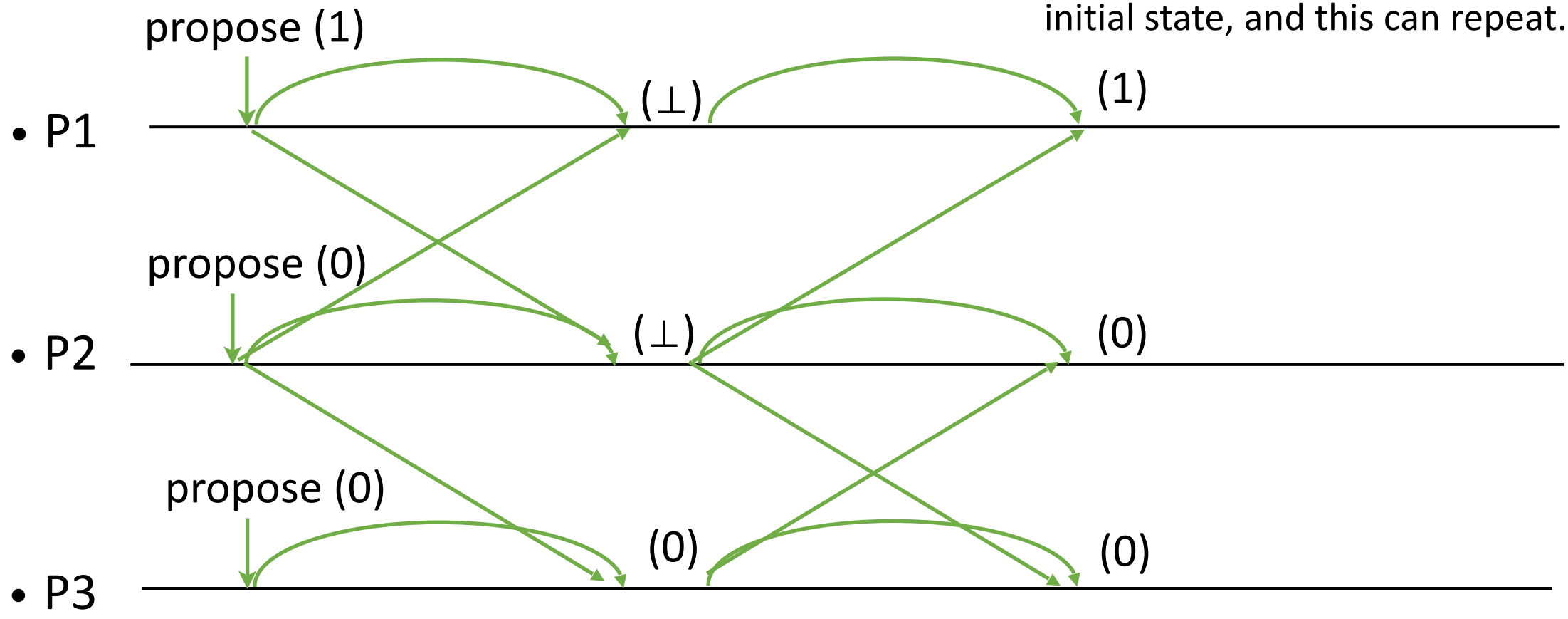
If not received non-⊥ value, get value from the coin for the next round



In Phase 2, ⊥ is sent only when there are different proposals. So any binary value from the coin is valid.

Deterministic choice would lead to non-termination

Deterministic choice of 1 for P1 takes the configuration to the initial state, and this can repeat.



Randomized Binary Consensus (1/3)

Implements: RandomizedConsensus, with domain {0, 1}.

Uses: BestEffortBroadcast, instance beb;

CommonCoin (multiple instances).

upon event ⟨ Init ⟩ **do**

round := 0; phase := 0; proposal := ⊥; decided := ⊥;

val := $[\perp]^n$

upon event ⟨ propose(v) ⟩ **do**

proposal := v; round := 1; phase := 1

trigger ⟨ beb, broadcast (Phase1(round, proposal)) ⟩

Randomized Binary Consensus (2/3)

upon event ⟨ beb, deliver(p, Phase1(r, v) ⟩ such that phase = 1 ∧ r = round **do**
  val[p] := v
  if decided ≠⊥ **then**
    trigger ⟨ beb, broadcast(Phase2(round, decided)) ⟩
  else if #(val) > n/2 **do**
    if ∃v such that val[p] for all p **then**
      proposal := v
    else
      proposal := ⊥
    val := $[\perp]^n$; phase := 2
    trigger ⟨ beb, broadcast(Phase2(round, proposal)) ⟩

Randomized Binary Consensus (3/3)

upon event ⟨ beb, deliver(p, Phase2(r, v) ⟩ such that phase = 2 ∧ r = round **do**
  val[p] := v
  if #(val) ≥ n/2 ∧ coin[round] is not initialized **then**
    Initialize a new instance coin[round] of CommonCoin with domain {0, 1}
    trigger ⟨ coin[round], release ⟩
upon event ⟨ coin[round], output(c) ⟩ **do**
  if decided ≠⊥ **then**
    proposal := decided
  else if ∃v ≠⊥ such that #{p∈Π | val[p]=v} > n/2 **then**
    decided := proposal := v
    trigger ⟨ decide(v) ⟩
  else if ∃p ∈ Π, v ≠⊥ such that val[p] = v **then**
    proposal := v
  else
    proposal := c
  val := $[\perp]^n$; round := round + 1; phase := 1
  trigger ⟨ beb, broadcast(Phase1(round, proposal)) ⟩

Add code to stop one round after decision.
Optimization: After decision, send the value to others by a reliable broadcast and stop.

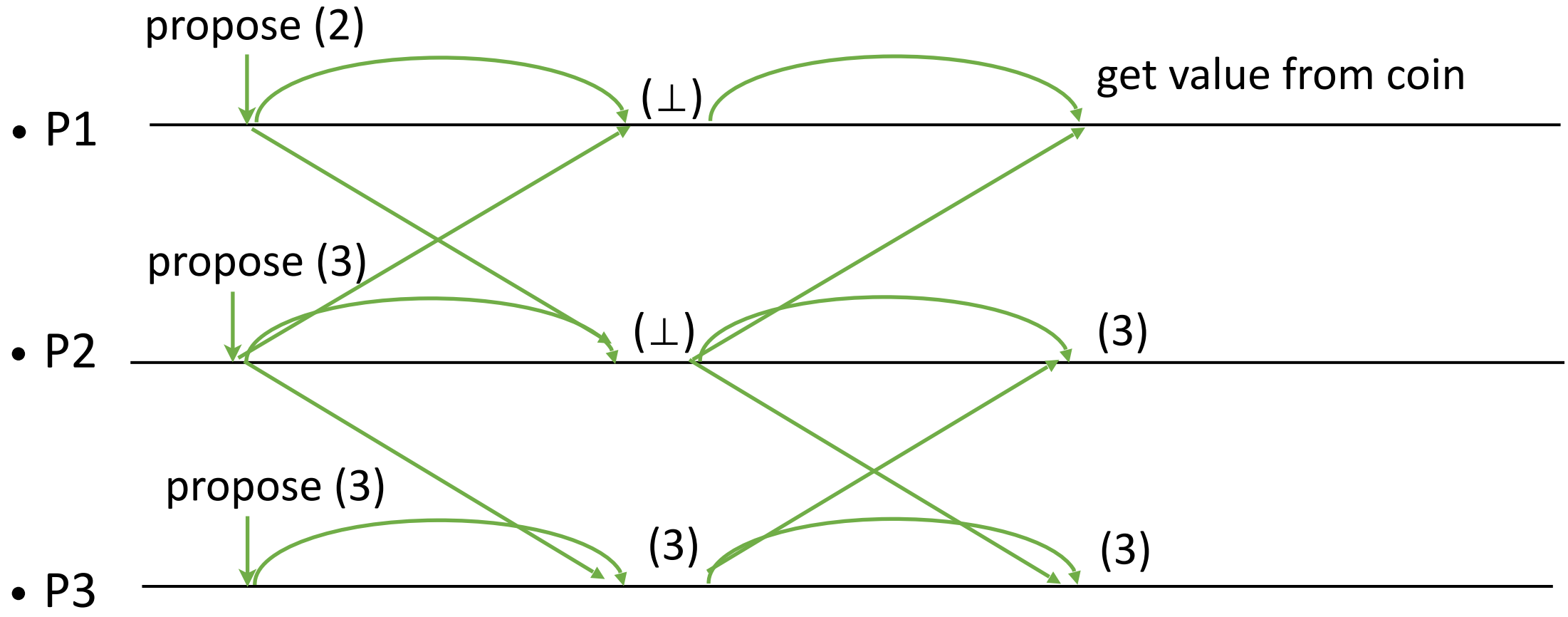
Proof of Properties

- **Agreement:**
  Case 1. Decisions in the same round: because of quorum intersection in phase 1.
  Case 2. Decisions in two rounds: The value decided in the first round is adopted by all correct processes for the second round.
- **Integrity:**
  The handler issues decision only if no decision is already made.
- **Validity:**
  A decided value in a round was proposed by a majority in that round. We show by induction on the rounds that a proposal in every round is a proposal from the first round. A process adopts its proposal for the next round either from this round which is proposed by a majority in this round, or from the coin. A value is taken from the coin only when the process receives $\perp$ from a majority in phase 2. A process sends $\perp$ for phase 2 after receiving different binary proposals in phase 1. So any binary value from the coin is a proposed value.
- **Probabilistic Termination:**
  The proposals adopted from the previous round are the same because of quorum intersection in phase 1. The coin takes values uniformly at random. It will eventually match the adopted value if any, and all processes decide in that round.

Randomized Consensus for General Domains

- Common coin with a domain
- In order to satisfy **validity**, the coin should output only proposed values. The domain of the coin should be a subset of **proposed values**.
- In each round, each process broadcasts its proposal.
- Processes collect the set of proposals, and instantiate the coin with it.
- This set grows and processes eventually converge.
- Correct processes eventually invoke the coin with the same set.
- In the meanwhile, the coin provides termination so that the protocol makes progress.

Get value from the coin for the next round



The coin should return either 2 or 3.

Randomized Consensus for General Domains

Implements: RandomizedConsensus.

Uses: …; ReliableBroadcast, instance rb

upon event ⟨ Init ⟩ **do**

...

values := ∅

upon event ⟨ propose(v) ⟩ **do**

...

values := values ∪ {v}

trigger ⟨ rb, broadcast(Proposal(v) ⟩

upon event ⟨ rb, deliver(p, Proposal(v) ⟩ **do**

values := values ∪ {v};

upon #(val) > N/2 ∧ phase = 2 ∧ decision = ⊥ **do**

Initialize a new instance coin[round] of CommonCoin with domain values;

…