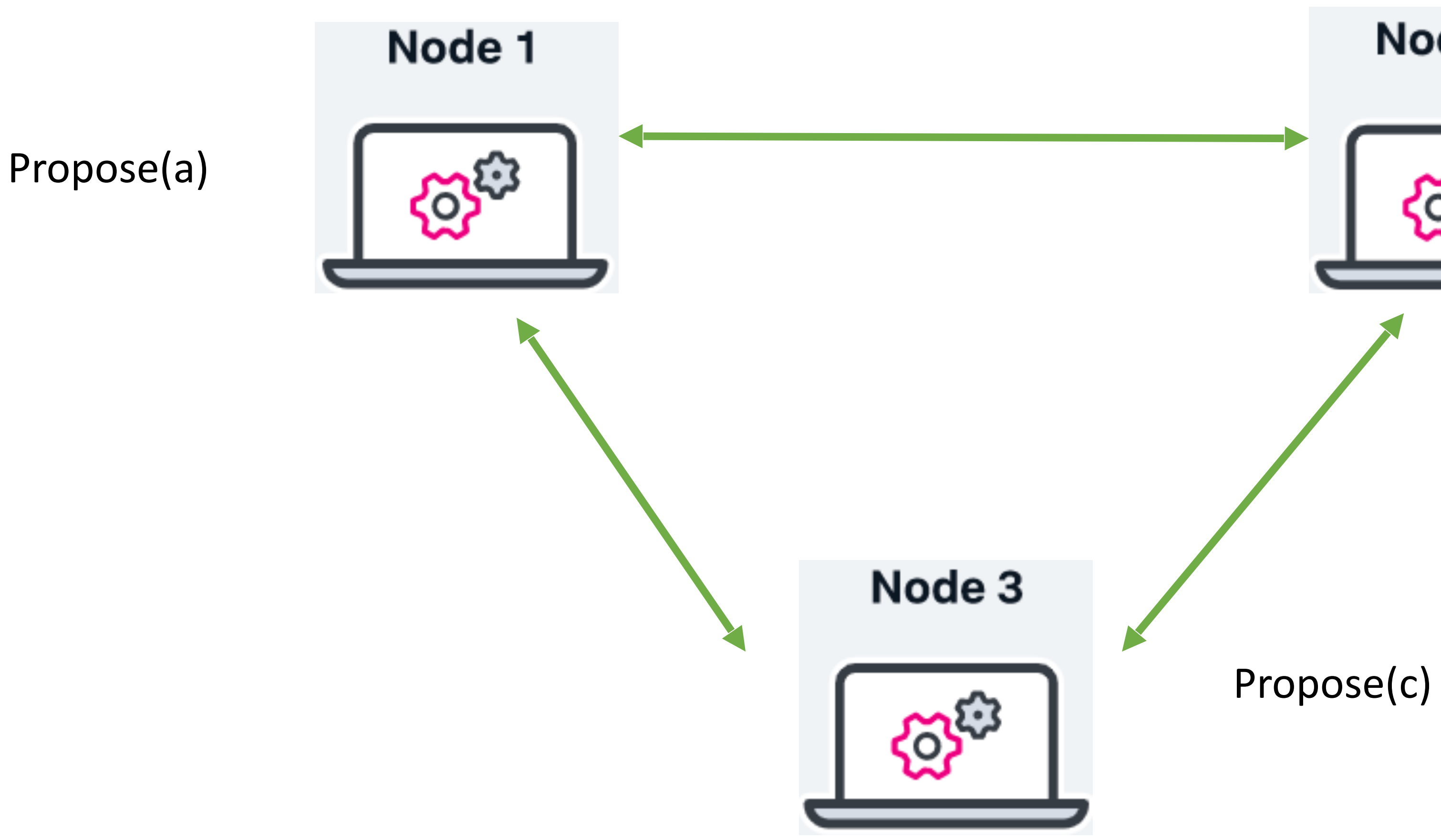
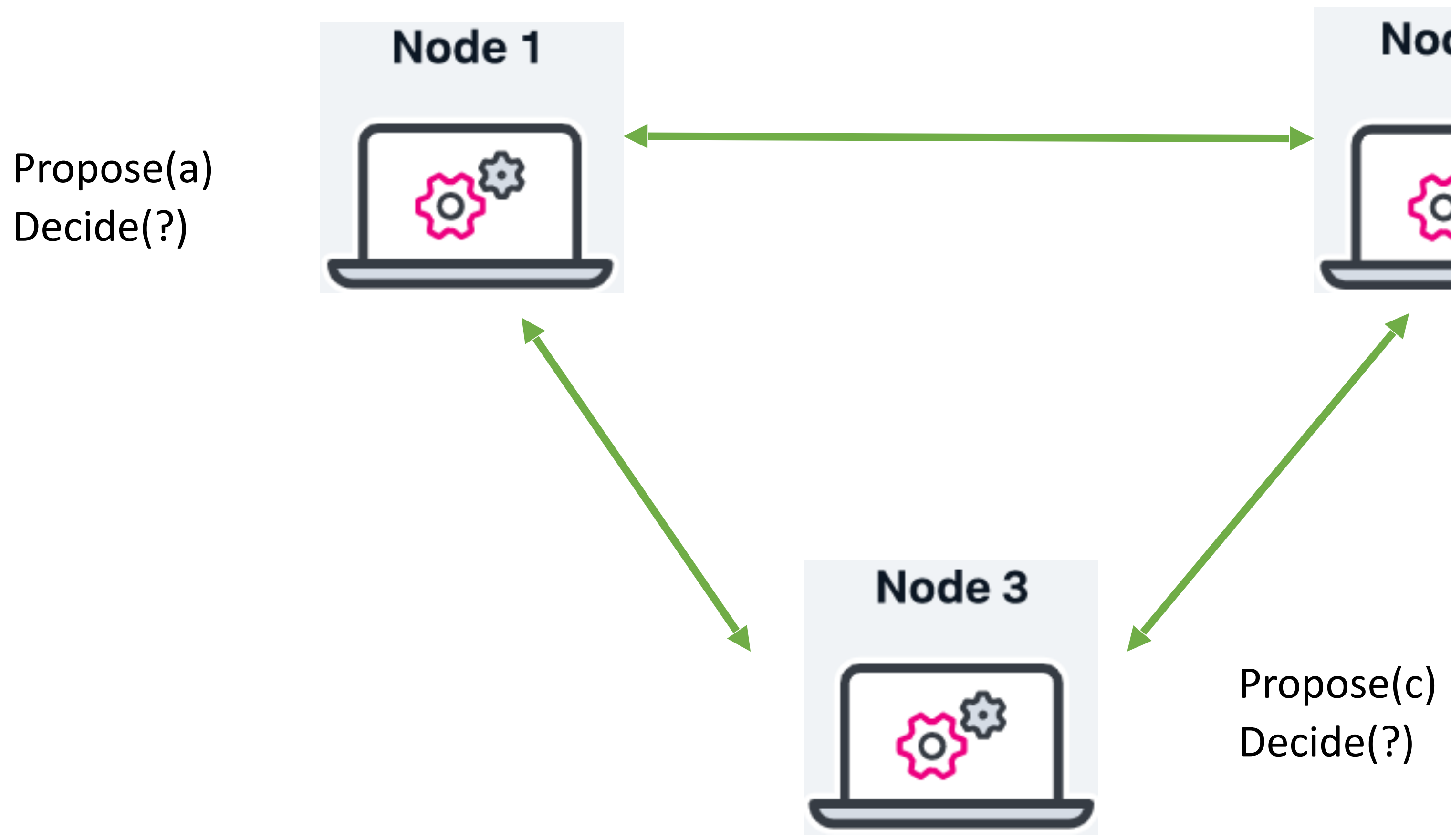
Consensus

Mohsen Lesani

Consensus



Consensus


Node 1
Node 2
Node 3
Propose(a)
Decide(?)
Propose(b)
Decide(?)
Propose(c)
Decide(?)

Consensus

In the consensus problem, the processes propose values and have to agree on one among these values.

Solving consensus is key to solving many problems in distributed computing (e.g., total order broadcast, atomic commit, terminating reliable broadcast and replicated services).

Consensus

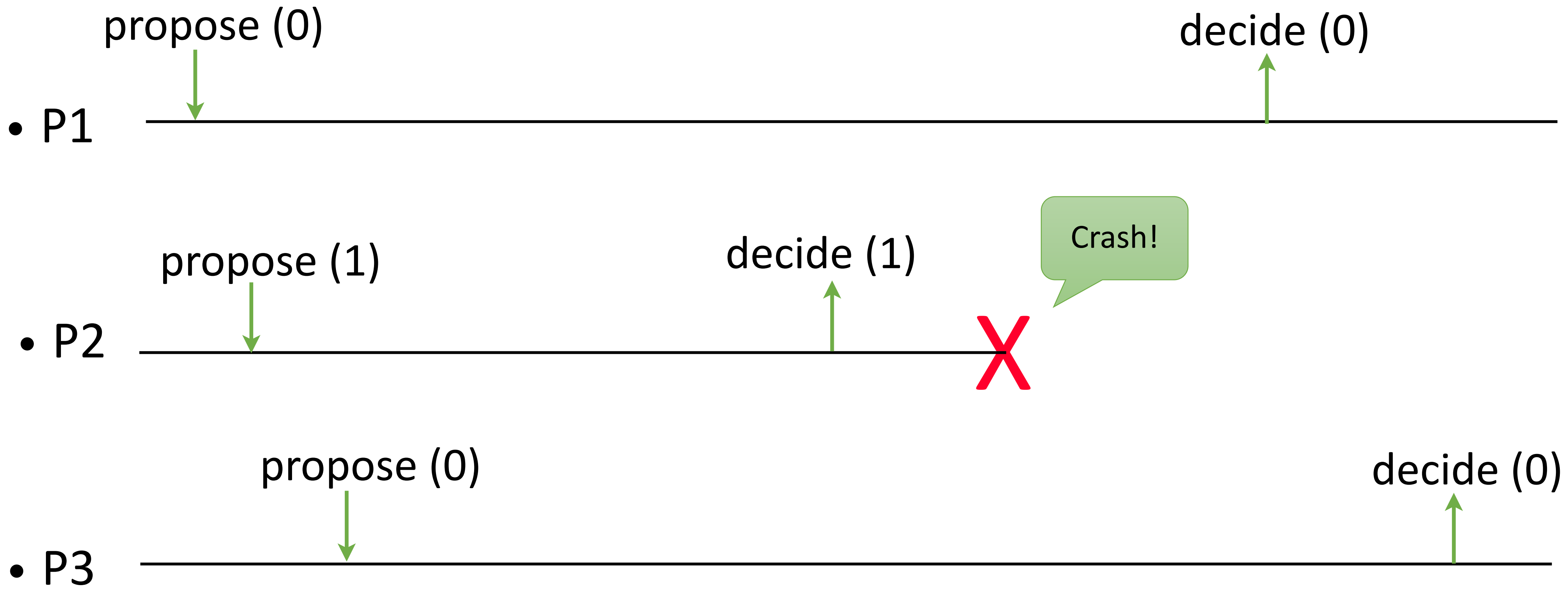
- **Events**
  - Request: < propose(v) >
  - Indication: < decide(v) >

- **Properties:**
  - **C1, C2, C3, C4**

Consensus

- **C1. Validity**: Any value decided is a value proposed.
- **C2. Agreement**: No two **correct** processes decide differently.
- **C3. Termination**: Every correct process eventually decides.
- **C4. Integrity**: No process decides twice.

Consensus



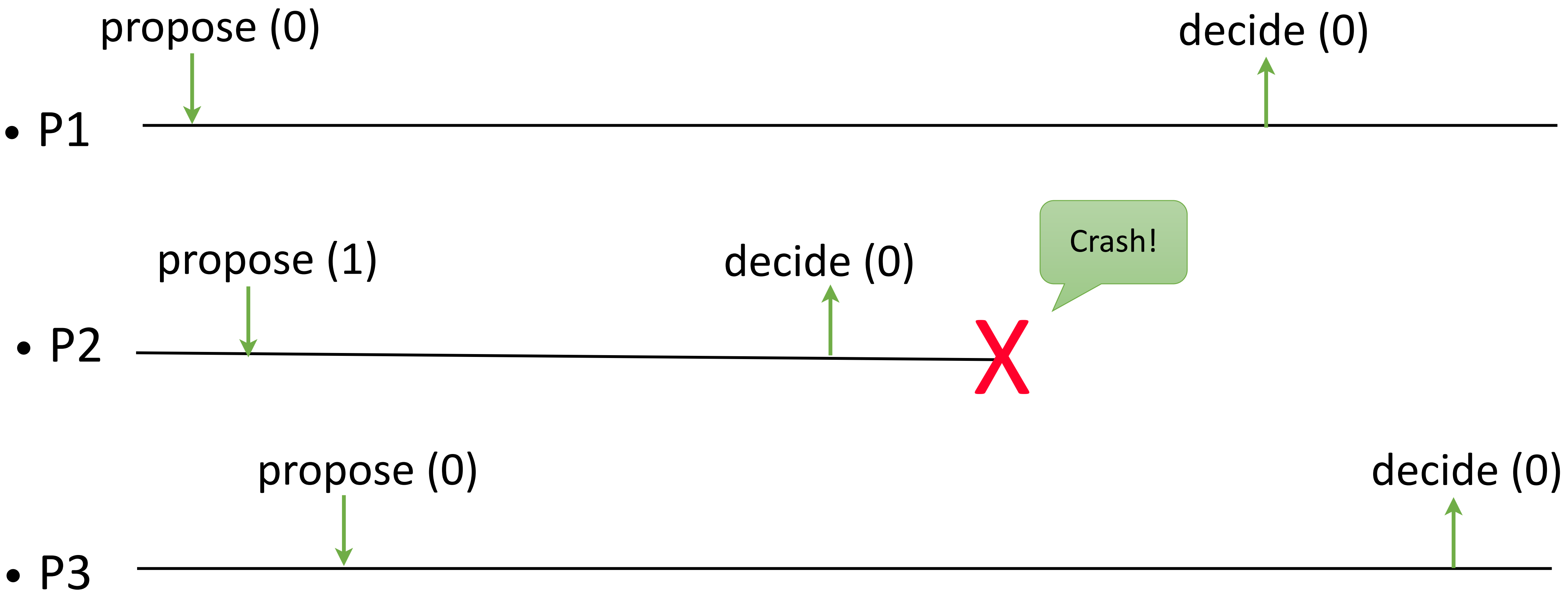
Correct Execution. Correct processes decide the same value.

Uniform Consensus

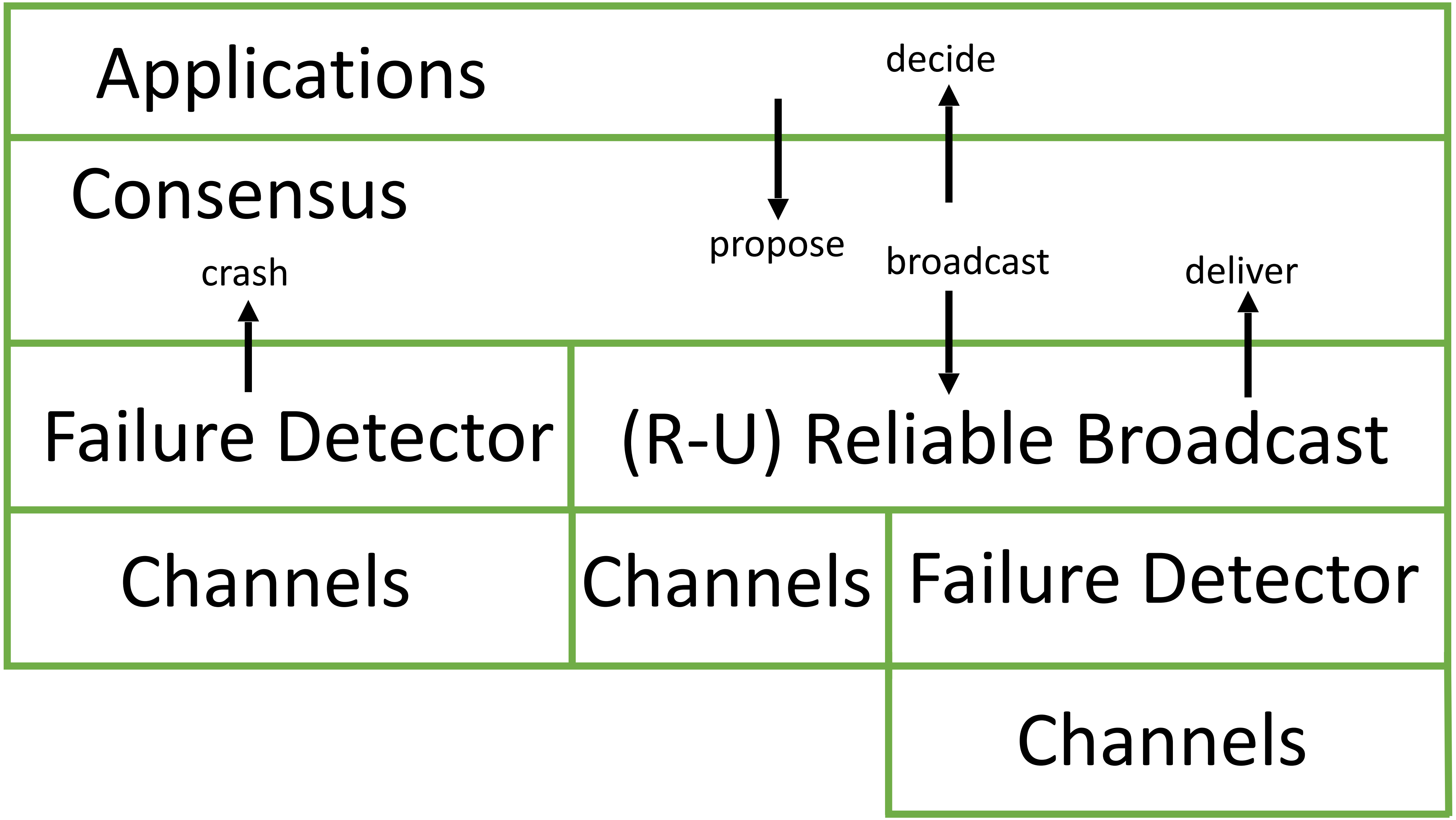
- **C1. Validity**: Any value decided is a value proposed.
- **C2’. Uniform Agreement**: No two **processes** decide differently.
- **C3. Termination**: Every correct process eventually decides.
- **C4. Integrity**: No process decides twice.

Even incorrect processes should not decide differently.

Uniform Consensus



Modules of a process



Three algorithms

- A P-based (i.e., fail-stop) consensus algorithm
- A P-based (i.e., fail-stop) uniform consensus algorithm
- A <>P-based uniform consensus algorithm assuming a correct majority

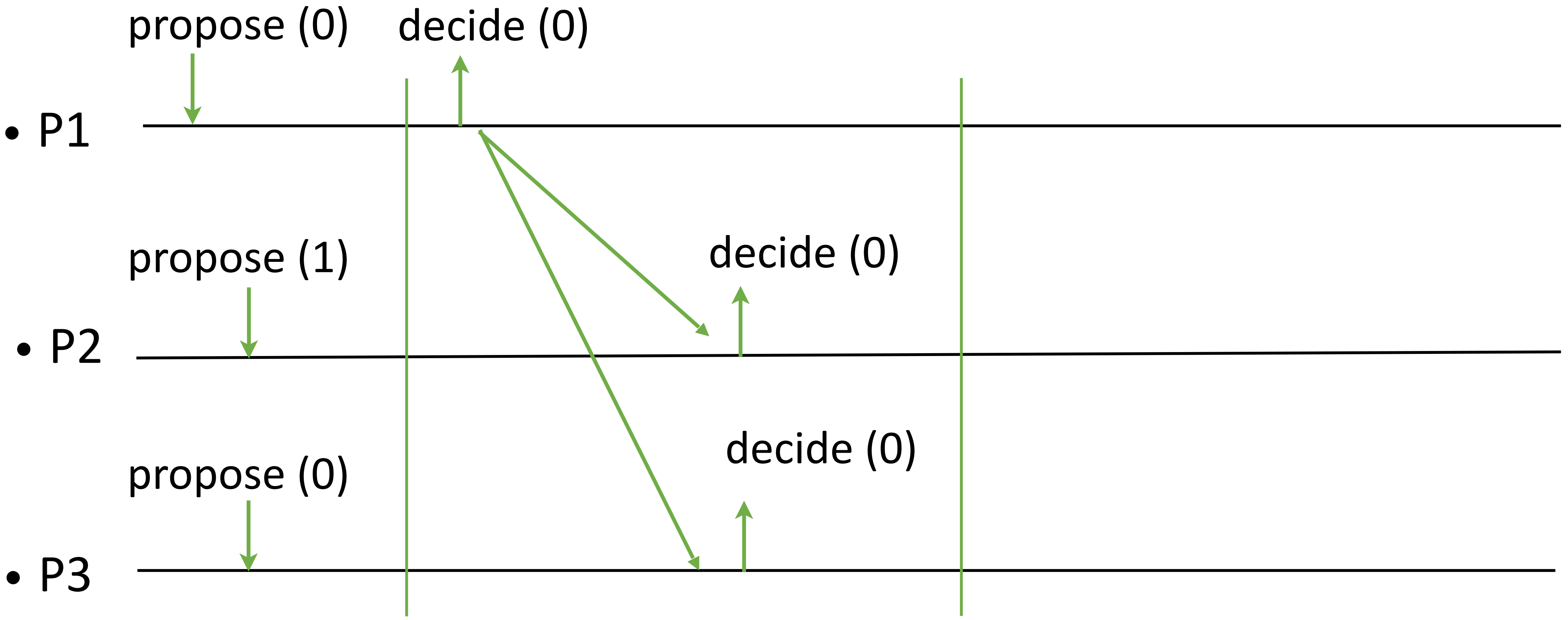
P is the perfect failure detector.
<>P is the eventually perfect failure detector.

Consensus algorithm I

- A P-based (i.e., fail-stop) consensus algorithm

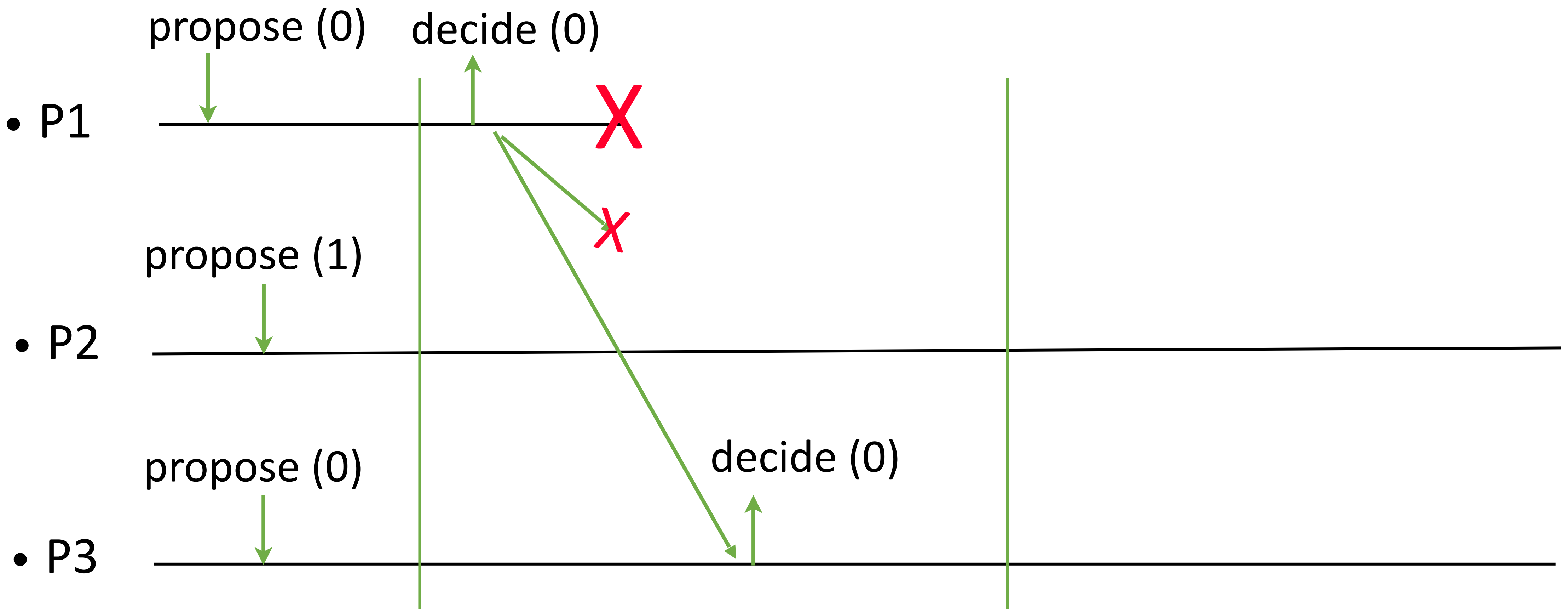
P is the perfect failure detector.

Consensus algorithm I



Can the first process decide its own value and impose it on others?

Consensus algorithm I



What if he crashes? P2 never decides.

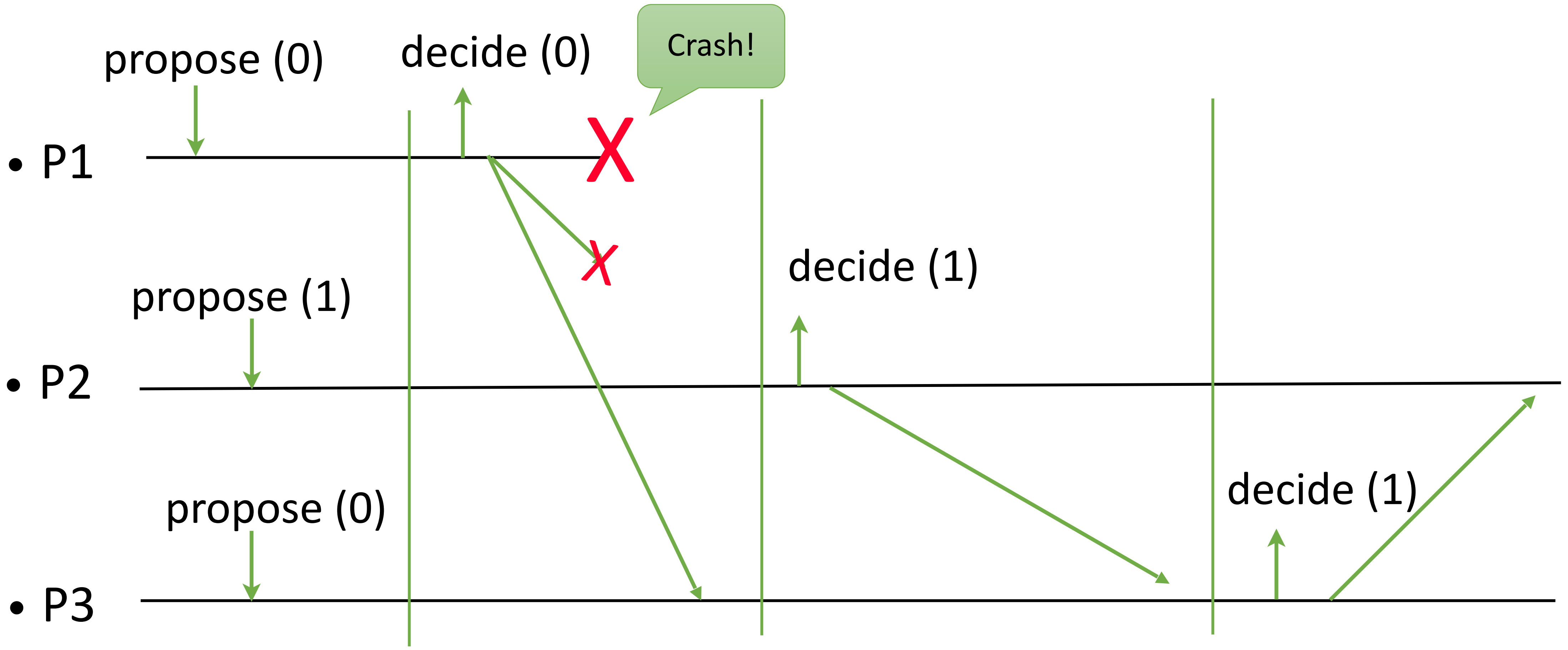
Consensus algorithm I

Idea:
Given an order for the processes, the first **correct** process broadcasts its proposal and imposes it on others.

Consensus algorithm I

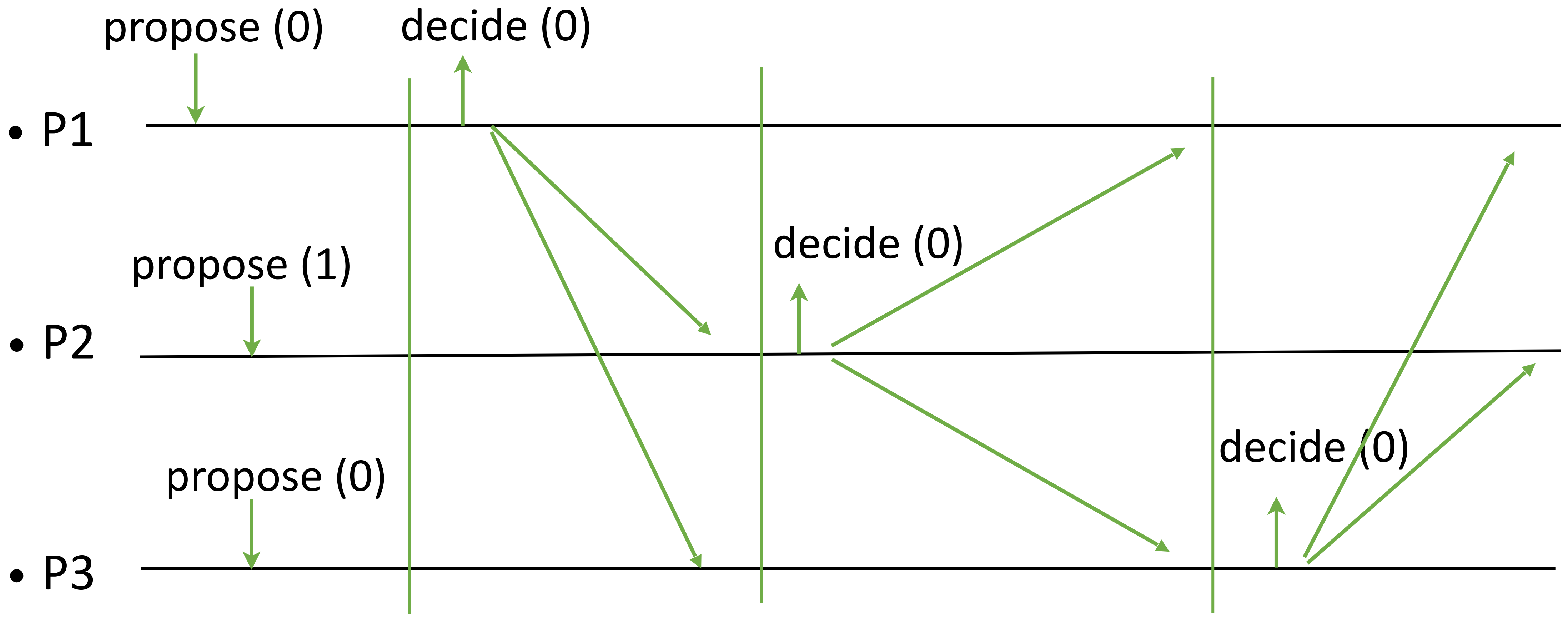
- The processes go through rounds incrementally (1 to n): in each round, the process with the id corresponding to that round is the leader of the round.
- The leader of a round decides its current proposal and broadcasts it to all.
- A process that is not leader in a round waits to
  (a) deliver the proposal of the leader in that round and adopts it, or
  (b) suspect the leader

Consensus algorithm I



We note that p3 cannot decide in round 1 because p1 might crash and not impose the value on p2. Then p2 can decide differently.

Consensus algorithm I



Consensus algorithm I

```
Implements: Consensus (cons).
Uses:
    BestEffortBroadcast (beb).
    PerfectFailureDetector (P).
upon event < Init > do
    suspected := ∅
    round := 1; prop := ⊥
    broadcast := delivered[] := false
upon event < propose(v) > do
    if prop = ⊥ then
        prop := v
upon event < P, crash(pi) > do
    suspected := suspected U {pi}
```

delivered map is used to remember if a message is delivered in a round.

broadcast is used to not broadcast twice.

Consensus algorithm I

```
upon event $p_{round}$=self and prop $\neq \perp$ and broadcast=false do
    trigger <decide(prop)>
    trigger <beb, broadcast(prop)>
    broadcast := true
```

When it is my turn, I decide my current proposal and broadcast it.

Consensus algorithm I

```
upon event < beb, deliver($p_{round}$, value) > do
    prop := value
    delivered[round] := true

upon event delivered[round] = true or $p_{round} \in$ suspected do
    round := round + 1
```

Correctness argument

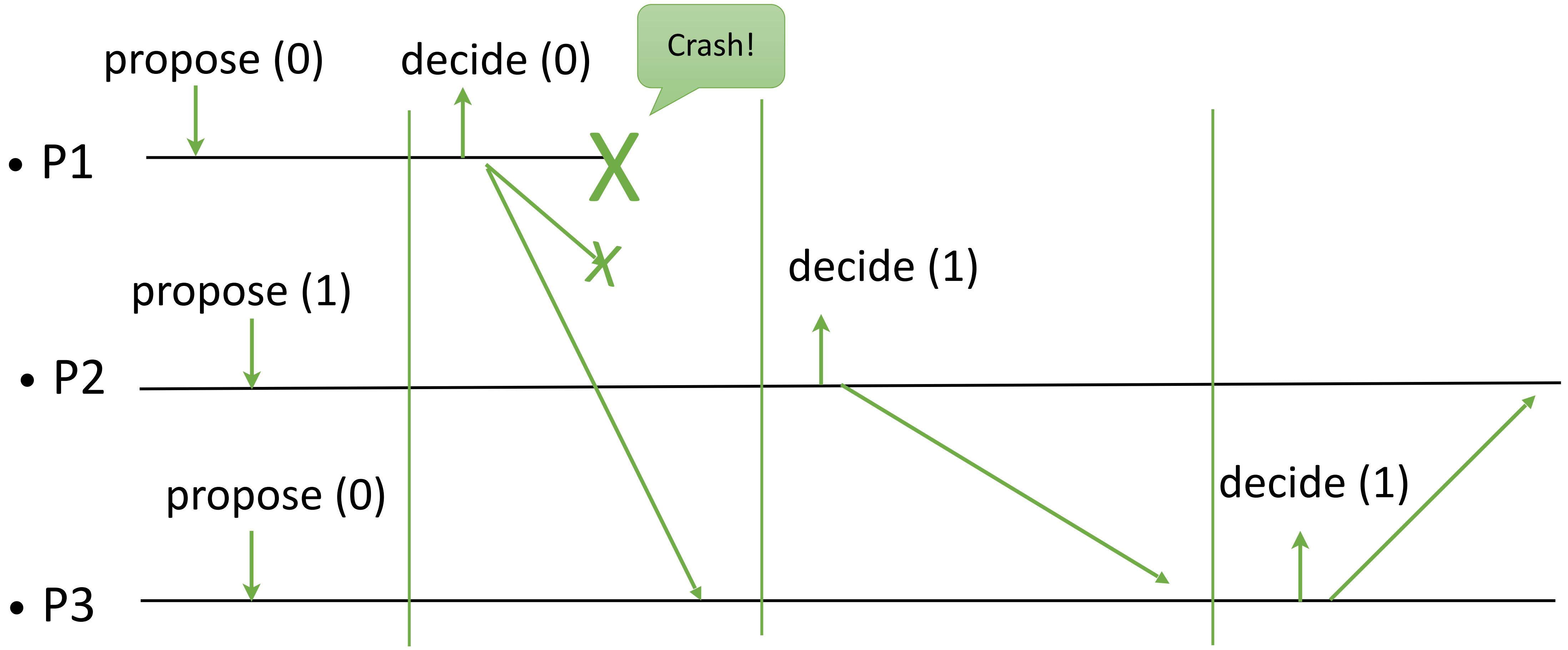
Agreement:

- Let pi be the correct process with the smallest id.
- Assume pi decides v.
- In round i, all correct processes receive and adopt v. From that round on, no other value exists in the system, and every correct process eventually decides v.

Consensus algorithm II

A P-based (i.e., fail-stop) **uniform** consensus algorithm

Non-uniformity in algorithm I

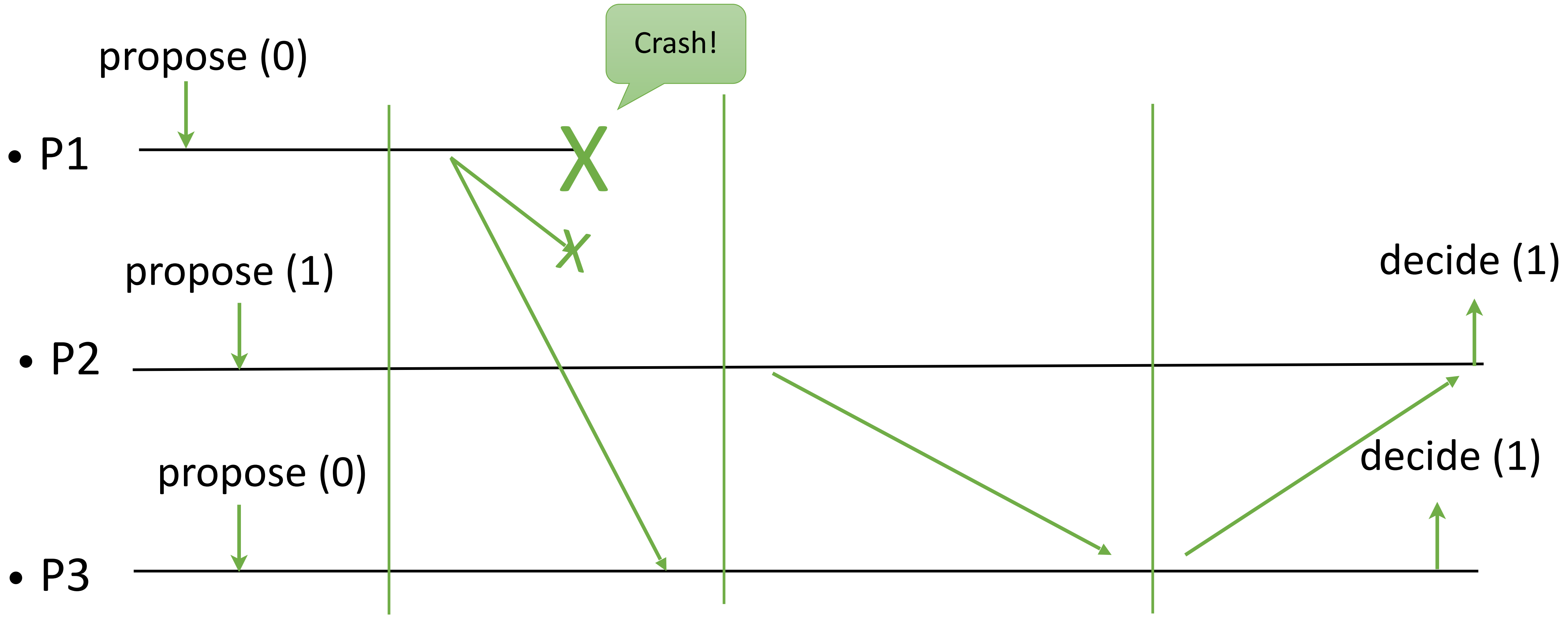


Consensus algorithm II

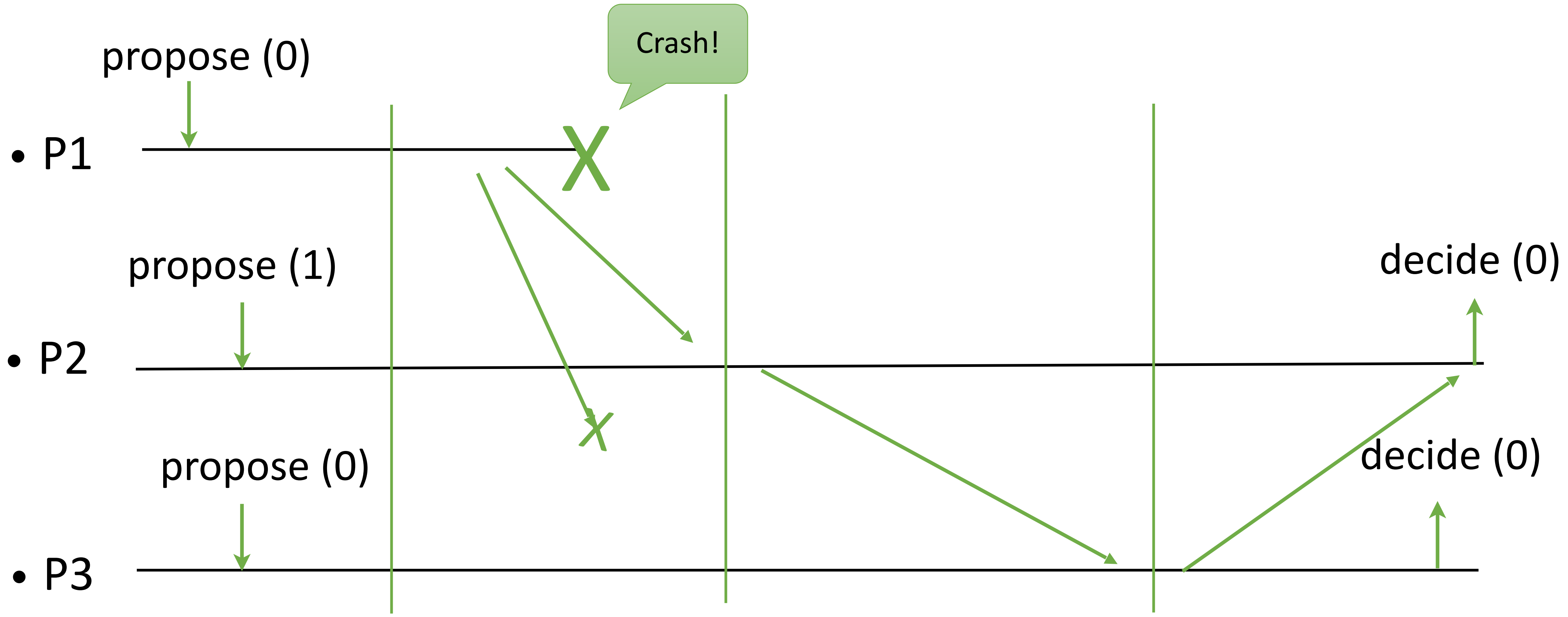
The idea:

- Not letting a process decide and then crash before imposing its value on everyone.
- So we delay the decision.
- The first correct process that succeeds imposing its value may be the last one. Therefore, we delay the decision to the last round.
- The processes exchange and update their proposals in rounds, and after n rounds decide on the current proposal value.

Uniformity in algorithm II

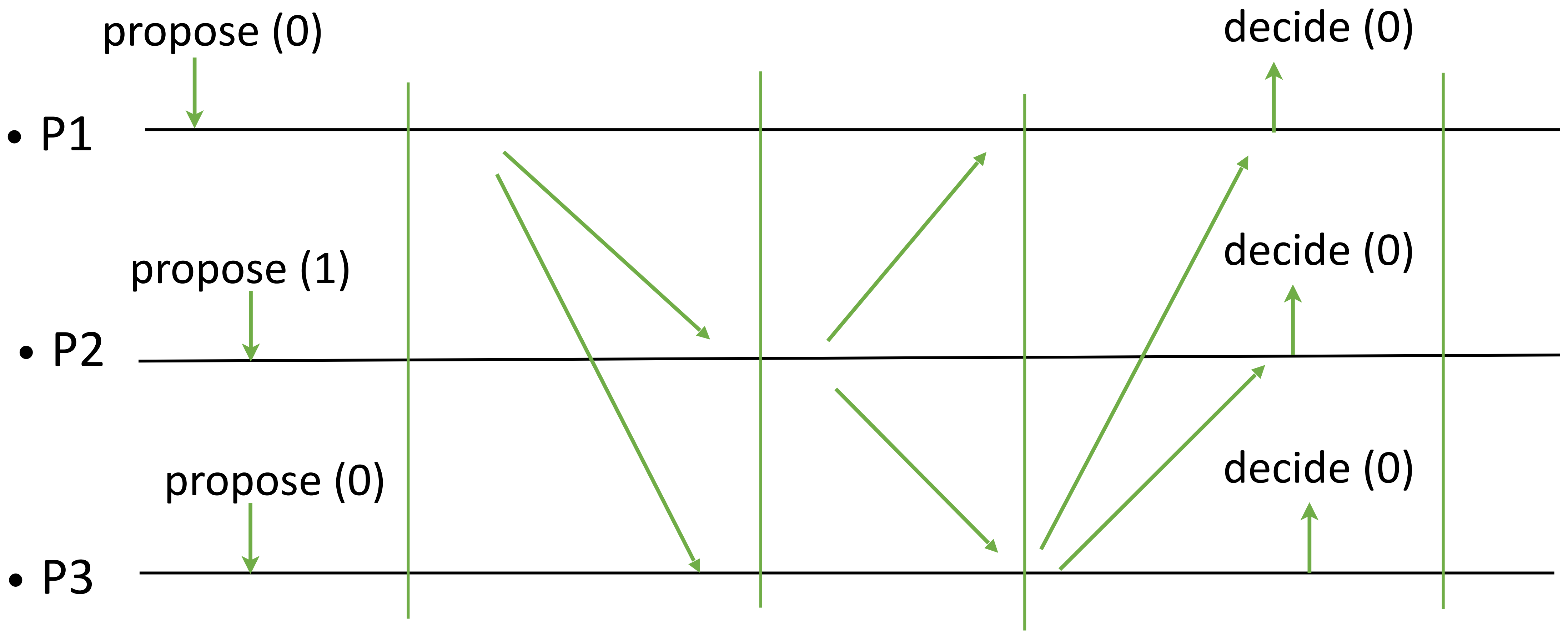


Uniformity in algorithm II



The proposal of the crashed process is decided.

Consensus algorithm II



Consensus algorithm II

- The processes go through rounds incrementally (1 to n).
- In each round i, process pi sends its current proposal to all.
- A process adopts any proposal it receives.
- Processes decide on their current proposal values at the end of round n.

Consensus algorithm II

```
Implements: Uniform Consensus (ucons).
Uses:
    BestEffortBroadcast (beb).
    PerfectFailureDetector (P).
upon event < Init > do
    suspected := ∅
    round := 1; prop :=⊥
    broadcast := delivered[] := false
    decided := false
upon event < propose(v) > do
    if prop = ⊥ then
        prop := v
upon event < P, crash(pi) > do
    suspected := suspected U {pi}
```

decided is used to not decide twice.

Consensus algorithm II

```
upon event $p_{round}$=self and prop $\neq \perp$ and broadcast=false do
    trigger <beb, broadcast(prop)>
    broadcast := true
```

Consensus algorithm II

```
upon event < beb, deliver($p_{round}$, value) > do
    prop := value
    delivered[round] := true

upon event delivered[round] = true or $p_{round} \in$ suspected do
    if round=n and decided=false then
        trigger <decide(prop)>
        decided := true
    else
        round := round + 1
```

Correctness argument

Uniform agreement:

- Consider the *decided process* p_i *with the lowest id*.
- Consider the processes p_j that have not crashed by round i. At round i, p_j must have adopted the proposal of pi or suspected pi.
- By the accuracy property of P, p_i could not be suspected at round i. Thus, p_j has adopted the proposal of p_i at round i.
- Since then, that value has been the only value in the system.
- This includes the values of the processes in round n which they decide.

Consensus Algorithm III

A <>P-based uniform consensus algorithm assuming a correct majority.

<>P is the eventually perfect failure detector.

Leader-driven consensus

- The most important paradigm.

- Introduced (as total-order broadcast) in
  - Viewstamped replication
  - Paxos
  - PBFT

- It is used in many cloud service platforms.

<>P

<>P ensures:

- **Strong completeness**: Eventually, every process that crashes is permanently suspected by all correct processes.

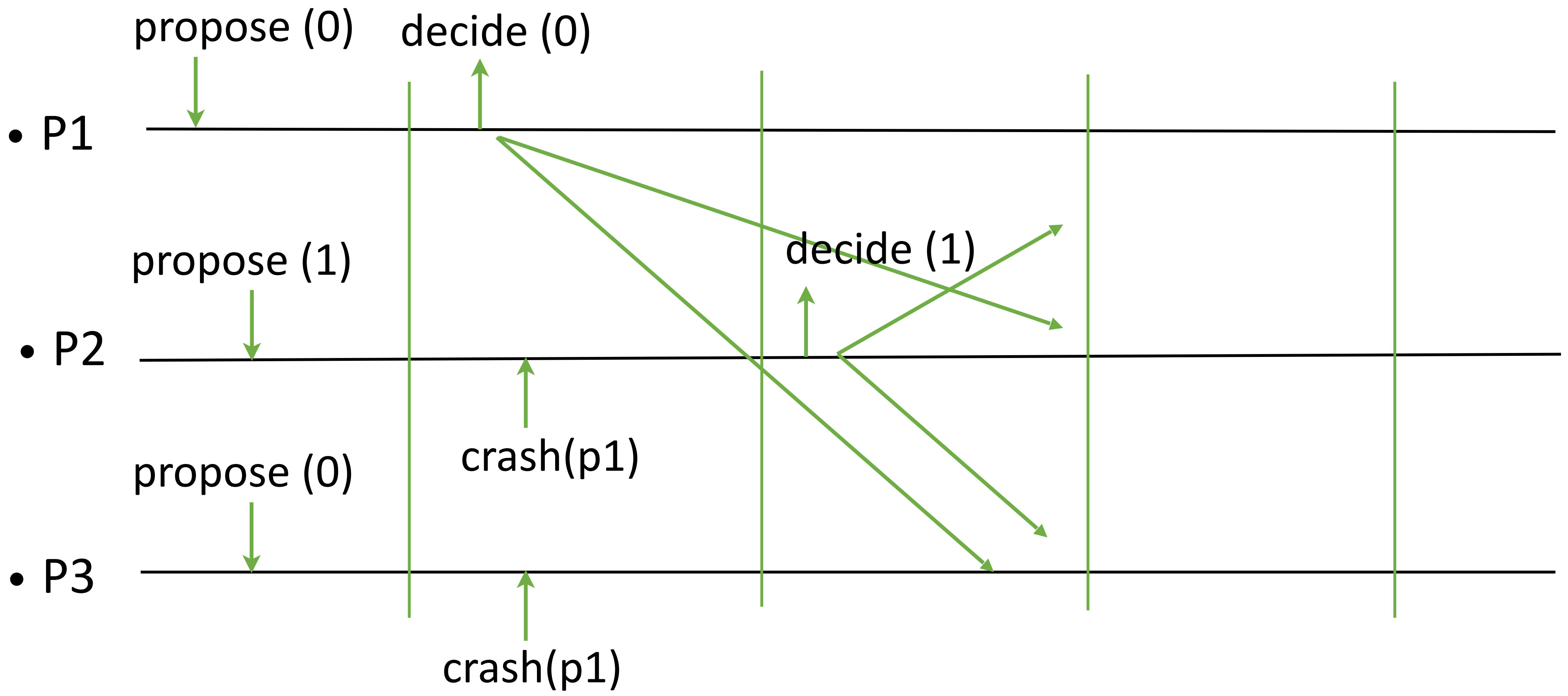
- **Eventual strong accuracy**: Eventually, no correct process is suspected by any process.

<>P

“<>” makes a difference:

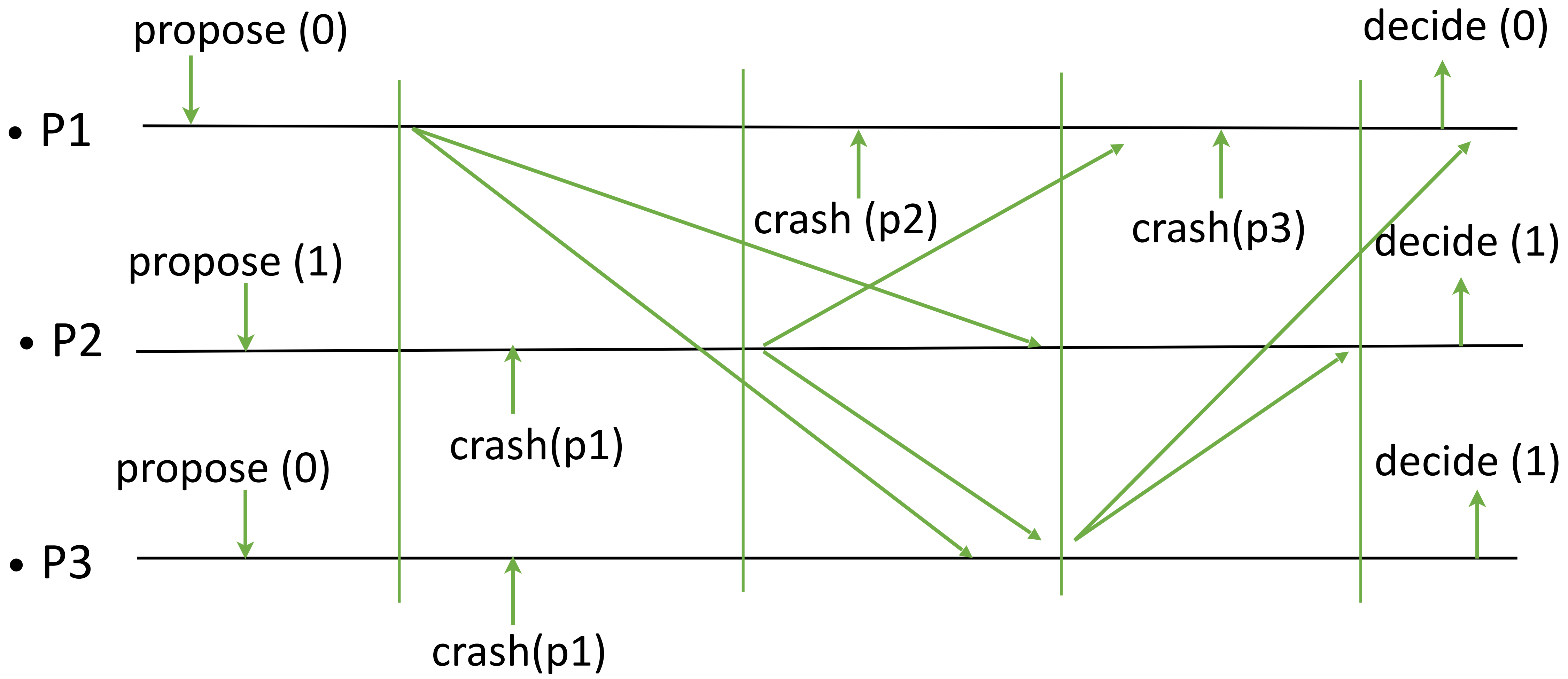
- **Eventual strong accuracy:** strong accuracy holds only after finite time.
- Correct processes may be falsely suspected a finite number of times.
- This breaks consensus algorithms I and II.

Agreement violated with <>P in algorithm I



The processes p2 and p3 move to the next round too early. Process P2 decides 1 early. Agreement is violated.

Agreement violated with <>P in algorithm II



Inaccurate crash messages make P2 and P3 miss the proposal value from P1, and similarly P1 misses proposal values from P2 and P3.

Consensus algorithm III

The idea:

A correct leader tells processes what to decide.

A leaders can fail or falsely suspected to have failed. It should change. To preserve agreement, it leaves a trace for the next.

Before deciding, the leader first stores the value in a quorum.

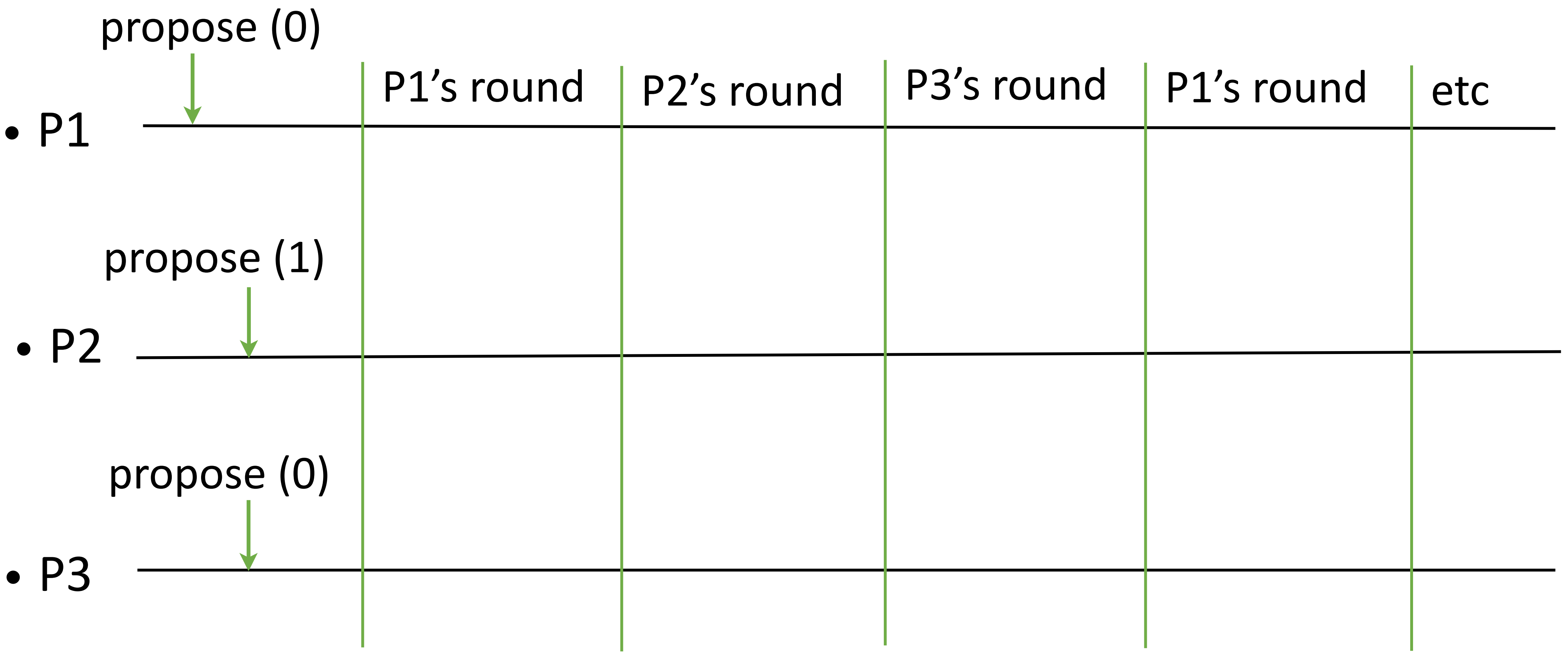
If the leader fails, the next leader retrieves the value from a quorum, and completes the decision of that value in the system.

Consensus algorithm III

- The processes alternate in the role of a **leader** until one of them succeeds in imposing a decision.
- Processes move incrementally from one round to the other.
- Simplifying assumption: Process pi is leader in every round k if k mod N = i.

  Failure detector is not perfect. The correct sets might differ. We will consider leader detector later.

Consensus algorithm III

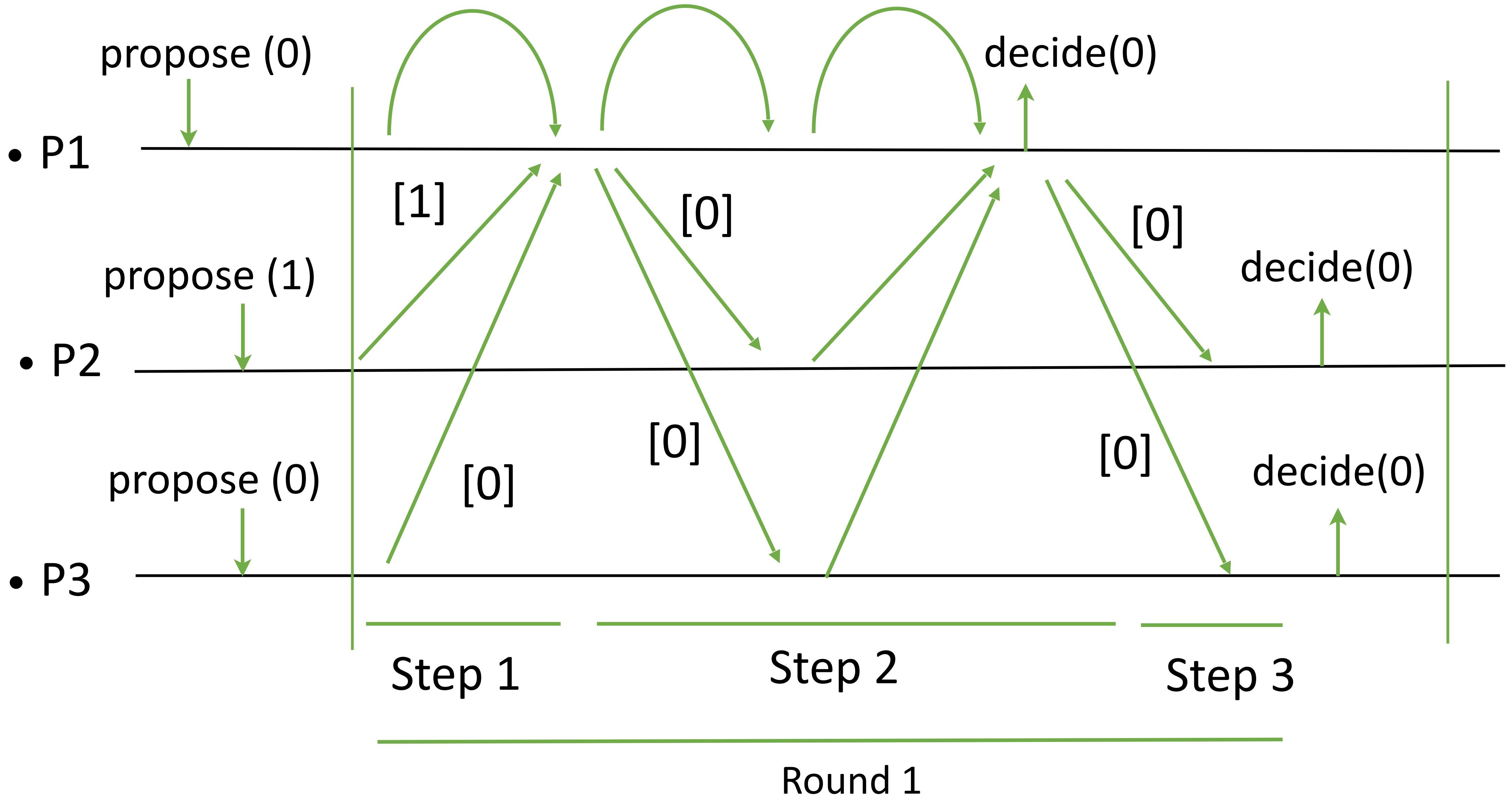


Consensus algorithm III

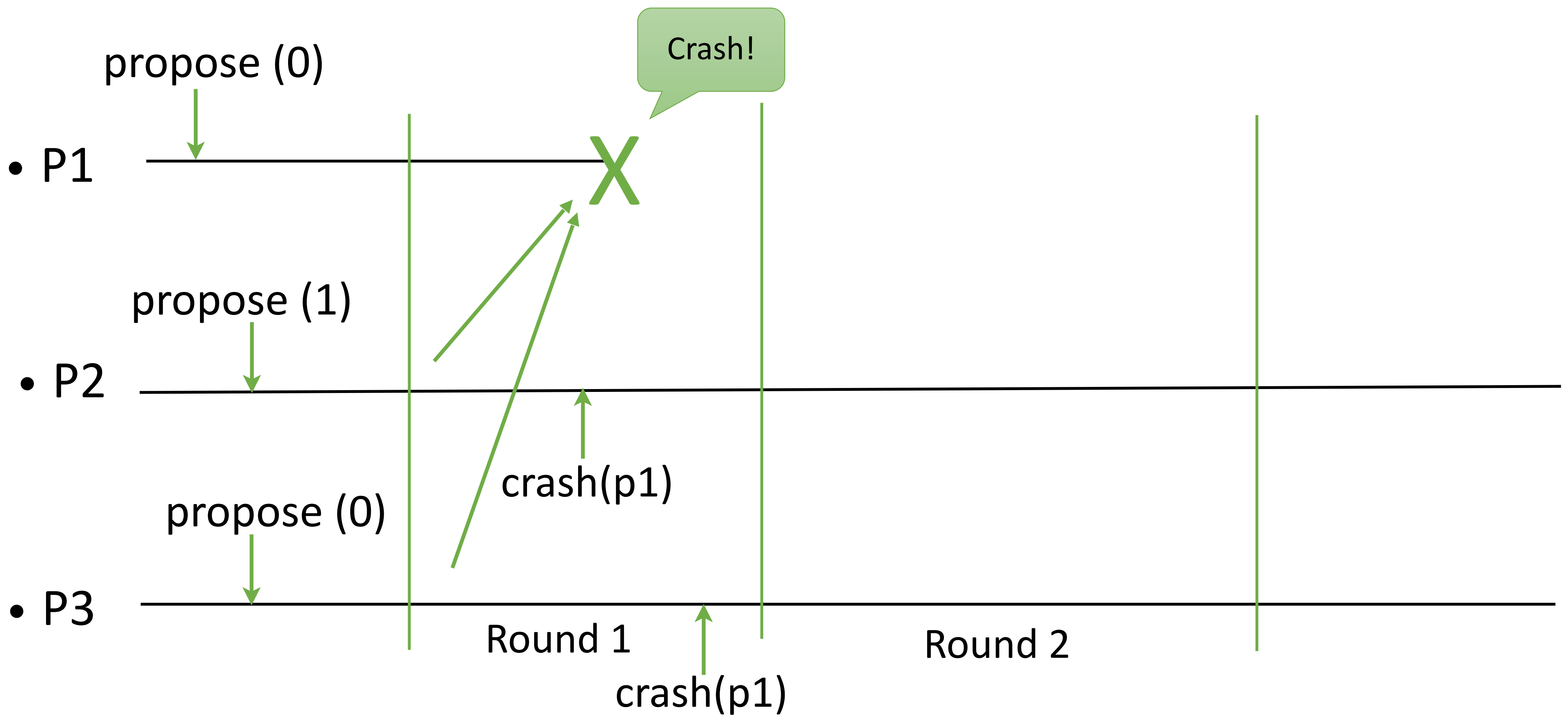
To decide, a leader executes:

1. It reads the latest adopted value from a majority.
   The latest value according to the round number when the value has been adopted.
2. It imposes that value at a majority.
3. It decides and broadcasts the decision to all.

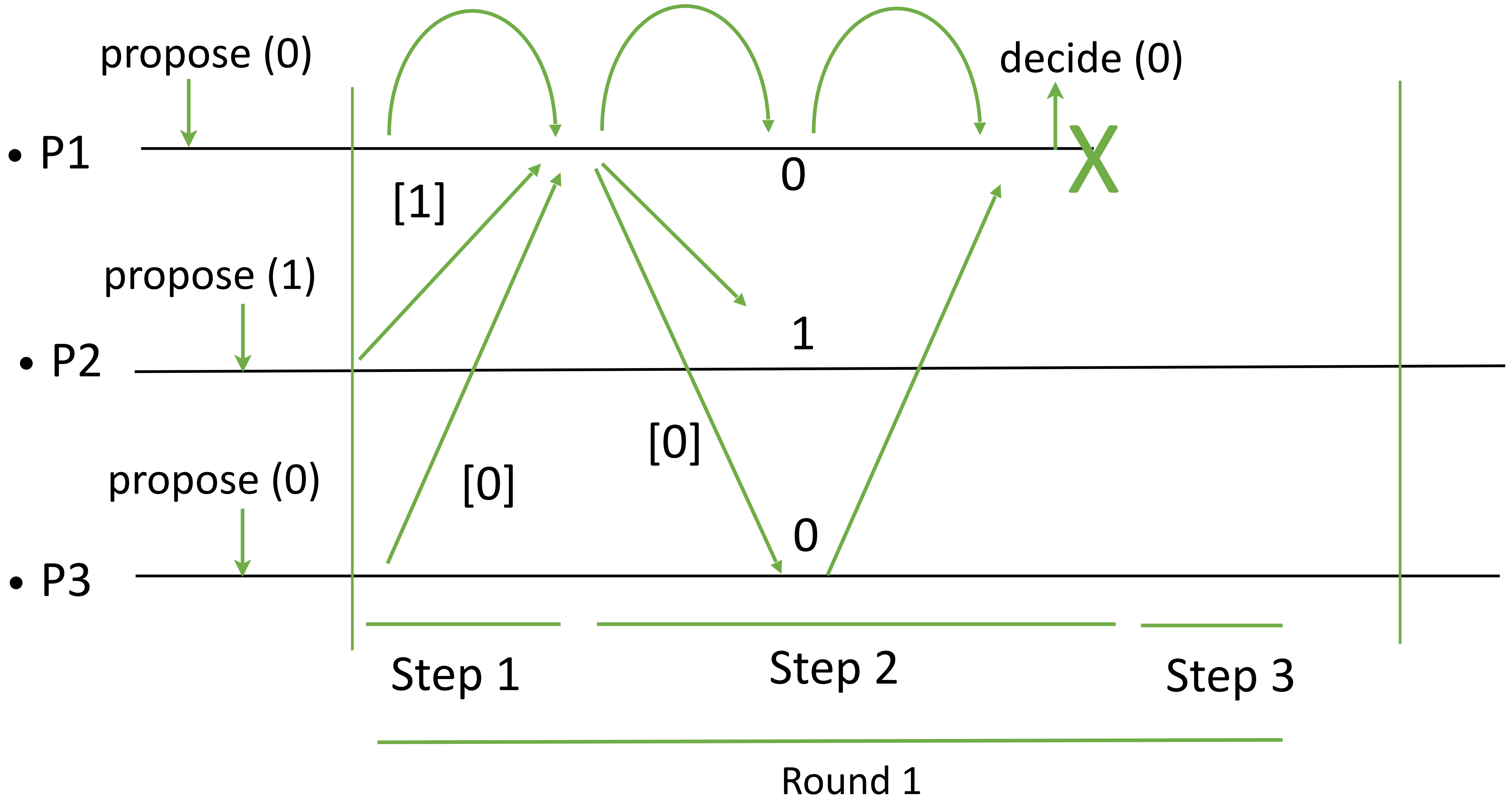
Consensus algorithm III



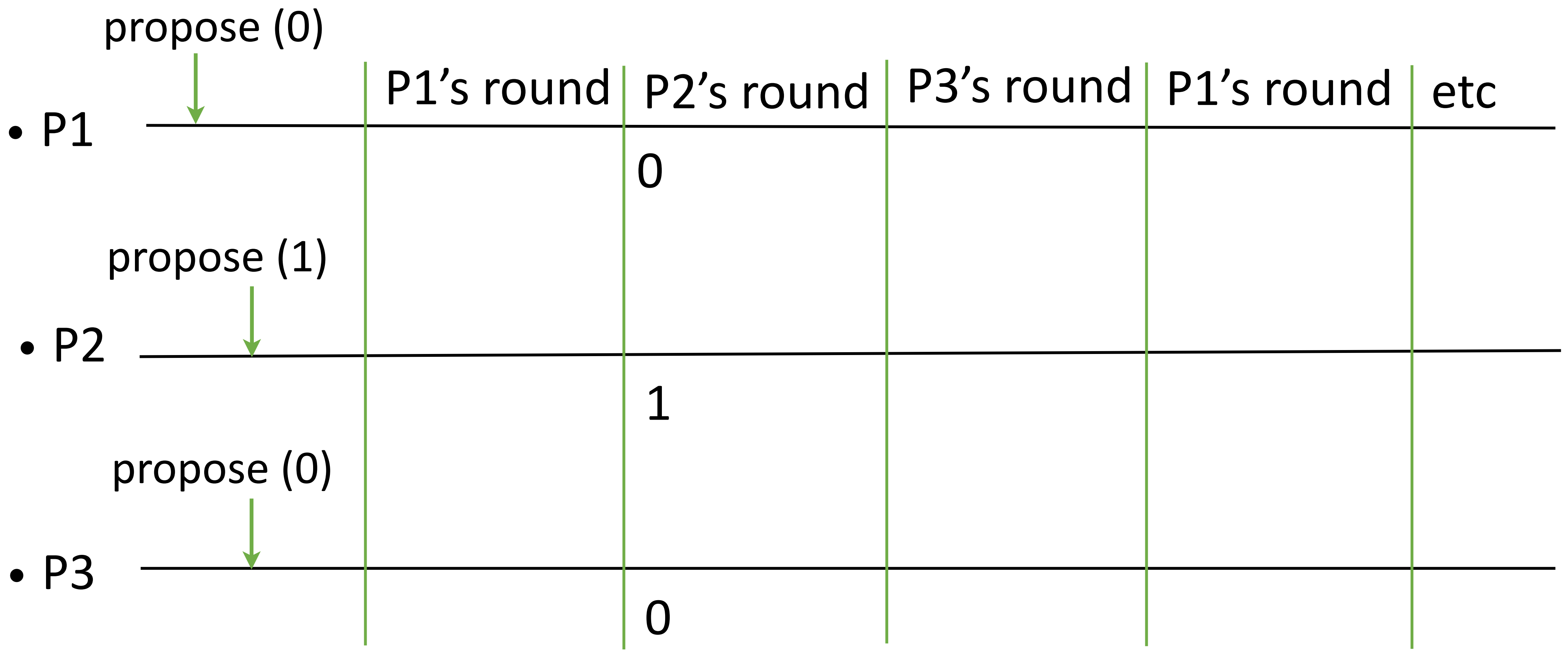
Consensus algorithm III



Consensus algorithm III

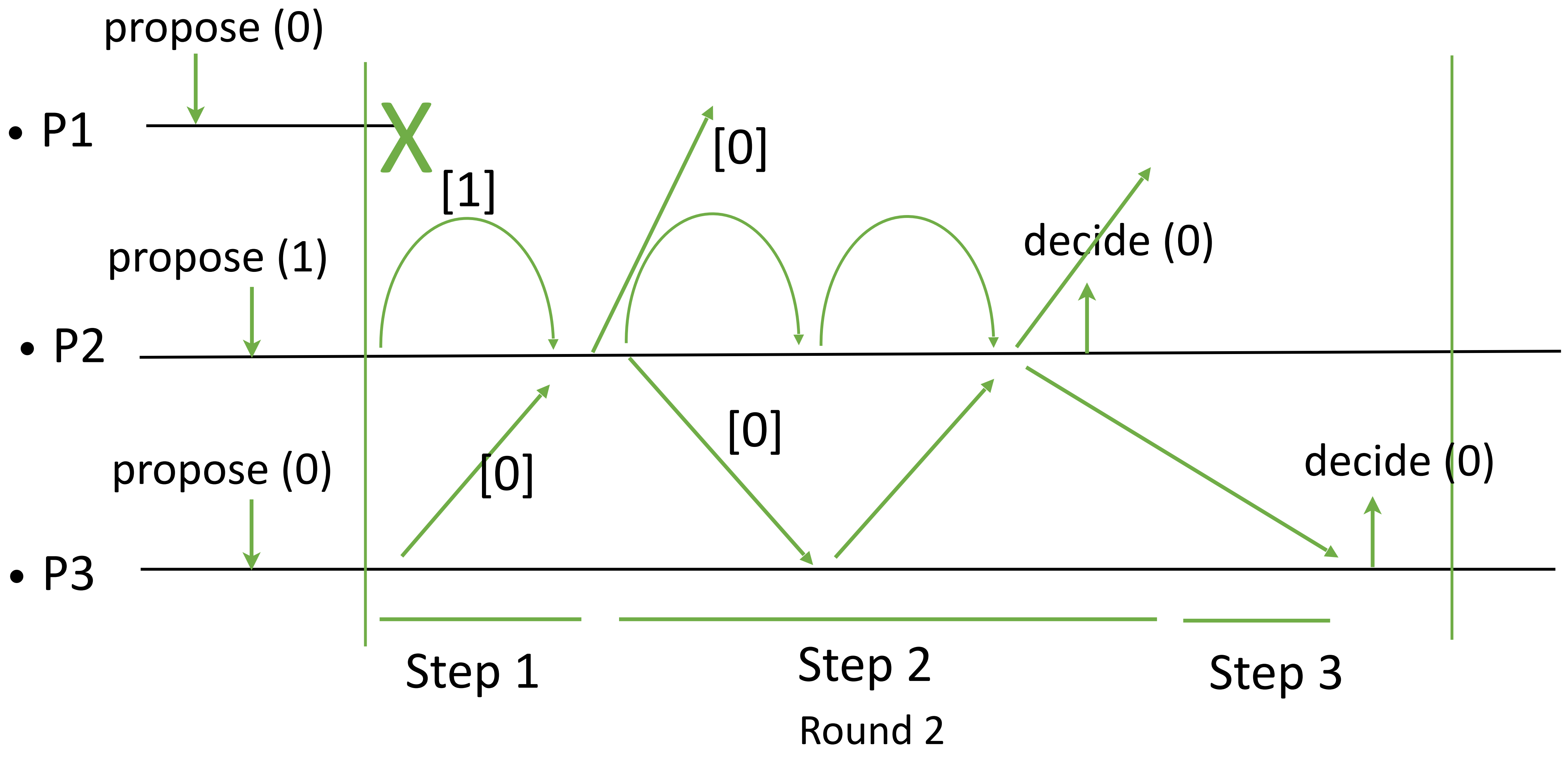


Consensus algorithm III



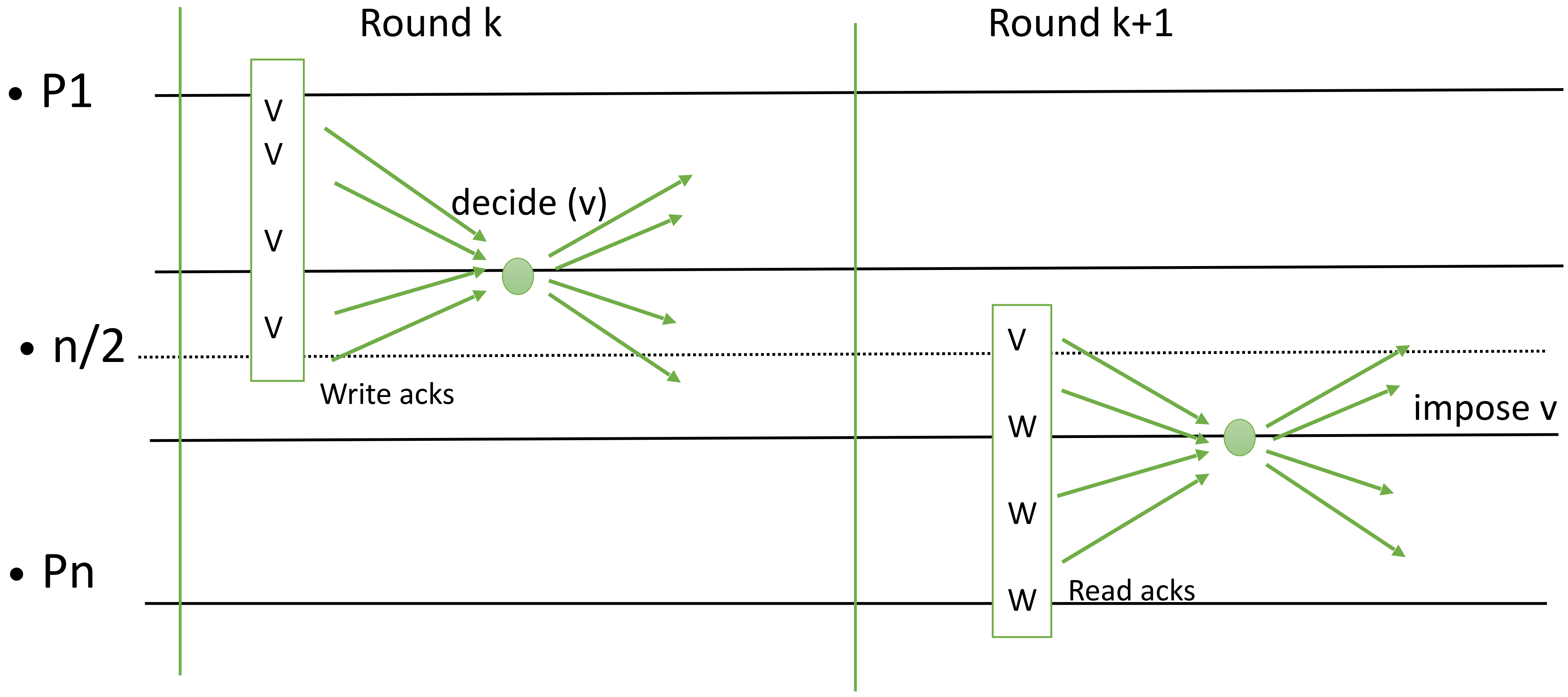
Consensus failed in round 1. It will continue to round 2.

Consensus algorithm III



The process p2 adopts the already decided value 0 from a majority and imposes it.

Quorums intersect



● : Leader of the round

Eventually Perfect Failure Detector

Agreement is guaranteed:

- Before a decision, the value is written to a quorum. The next decision is either in the same round, or reads the value from a quorum.

Liveness is guaranteed:

- Eventually the leader is correct. There is a correct majority that will not leave the leader blocked.

Totally Unreliable Failure Detector

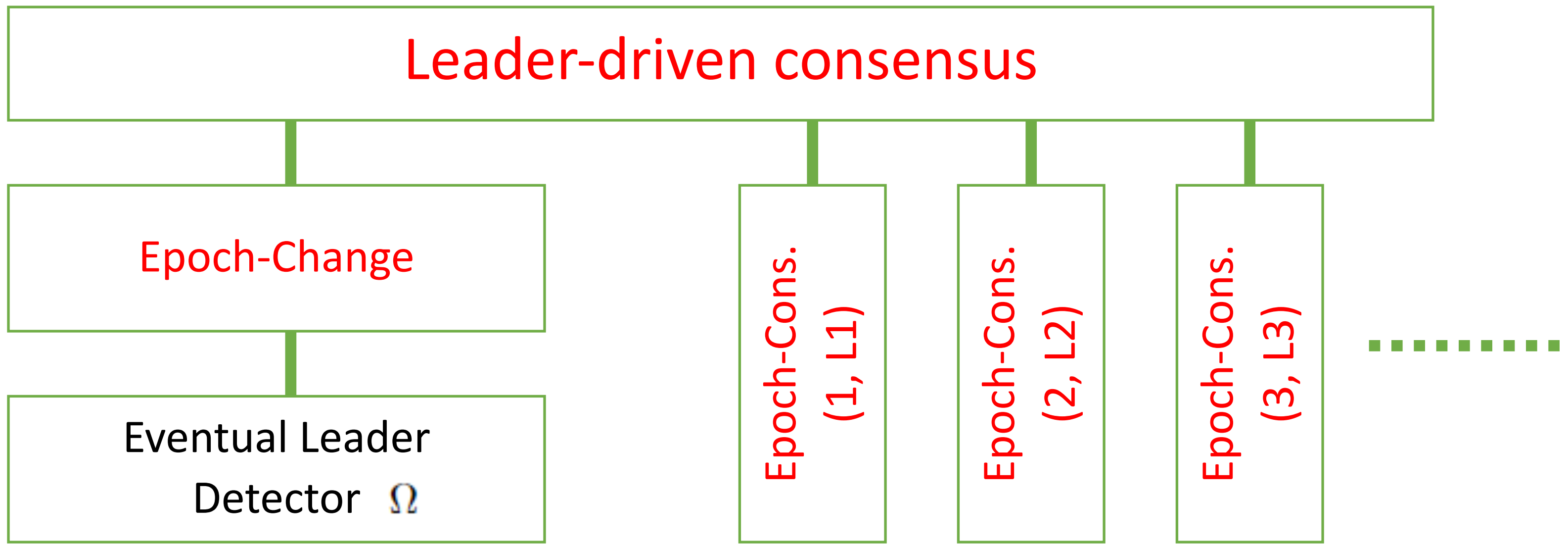
Agreement is guaranteed:

- It may suspect a correct process. May lead to leader change.
- It may not suspect an incorrect process. The protocol does not need it.

Termination not guaranteed:

- Everybody may be suspected infinitely often.

Leader-driven consensus



- Leader-driven consensus invokes
  - One instance of Epoch-Change (invokes Omega)
  - Multiple instances of Epoch Consensus
    - Identified by the epoch number and a designated leader

Uniform Consensus (uc)

- Events :
  - Request <uc, propose(v)>
    Proposes value v for consensus
  - Indication <uc, decide(v)>
    Outputs a decided value v of consensus

- Properties :
  - UC1 (Termination): Every correct process eventually decides.
  - UC2 (Validity): Any decided value has been proposed by some process.
  - UC3 (Integrity): No process decides twice.
  - UC4 (Uniform Agreement): No two processes* decide differently.

* both correct or faulty

Epoch-Change (ec)

- Events:
  - Indication <ec, start-epoch(ts, L)>
    - Starts epoch (ts,L), timestamp ts and leader L
- Properties:
  - EC1 (Monotonicity):
    If a correct process starts epoch (ts,L) and later starts epoch (ts',L'), then ts' > ts.
  - EC2 (Consistency):
    If a correct process starts epoch (ts,L) and another correct process starts epoch (ts,L'), then L = L'.
  - EC3 (Eventual Leadership):
    Eventually every correct process starts no further epoch; moreover, every correct process starts the same last epoch (ts,L), where L is a correct process.

Epoch-Consensus (ep)

Associated with timestamp ts and leader L

Events:

- Request <ep, propose(v)>

  Proposes v for epoch consensus (executed by leader only)
- Request <ep, abort>

  Aborts this epoch consensus
- Indication <ep, decide(v)>

  Outputs decided value v for epoch consensus
- Indication <ep, aborted(s)>

  Signals that this epoch consensus has completed the abort and returns state s

Leader-driven consensus

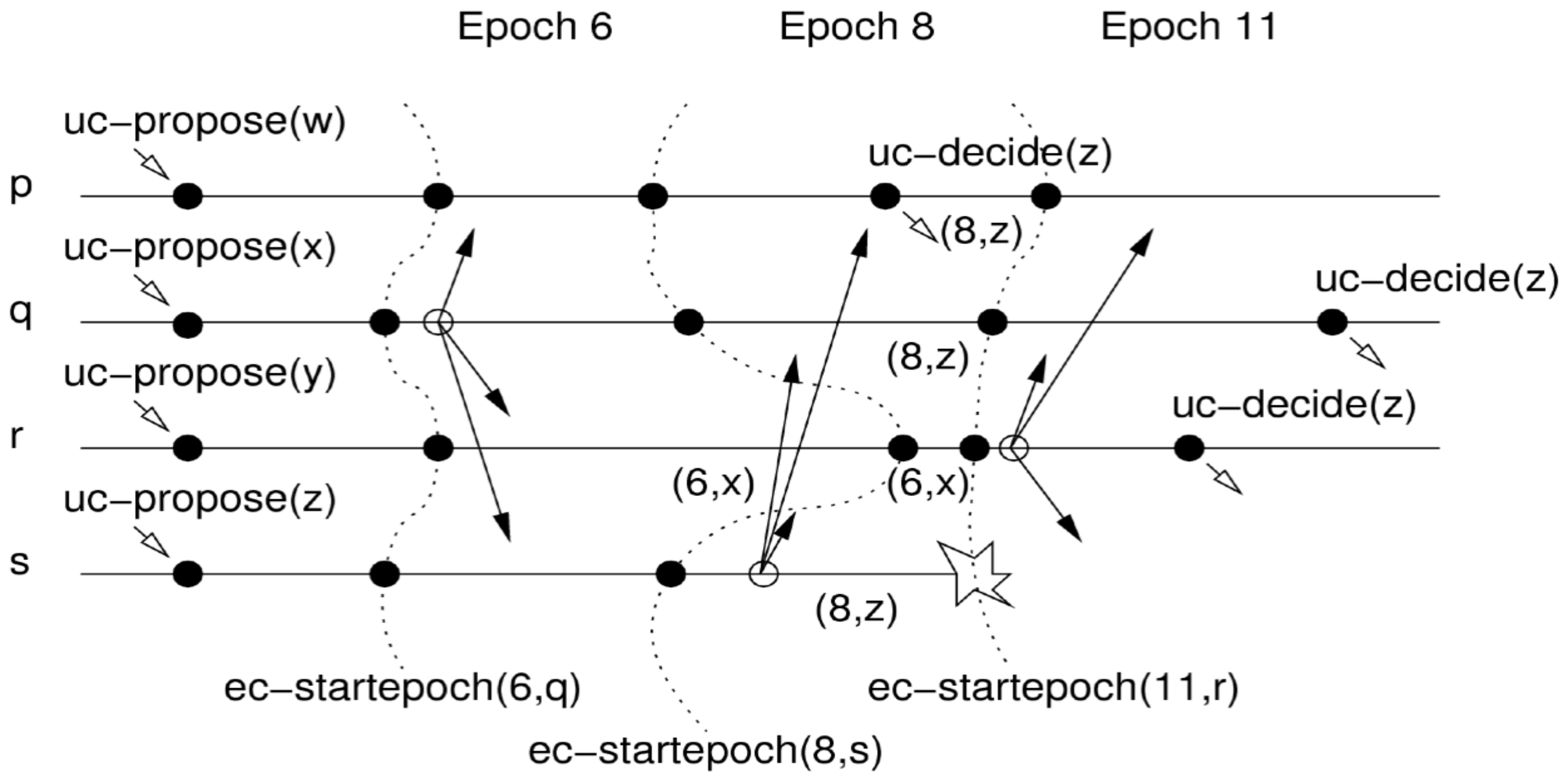
```
Implements uc
uses ec, ep (multiple instances)

upon <init> do
        val := ⊥;  proposed := false;  decided := false
        (ets, L) := (0, L0);  (newts, newL) := (0, ⊥)
        Initialize Epoch Consensus inst. ep[0] with timestamp 0 and leader L0

upon <uc, propose(v)> do
        val := v
upon <ec, start-epoch(newts', newL')> do
        (newts,newL) := (newts',newL')
        trigger <ep[ets], abort>
upon <ep[ets], aborted(s)> do
        (ets,L) := (newts,newL)
        Initialize Epoch Consensus inst. ep[ets] with timestamp ets, leader L, and state s
        proposed := false
```

Switching from an epoch to the next

Leader-driven consensus

```
upon L = self ∧ val ≠⊥ ∧ ¬proposed do
        proposed := true
        trigger <ep[ets], propose(val)>

upon <ep[ets], decide(v)> do
        if ¬decided then
                decided := true
                trigger <uc, decide(v)>
```

Change of epochs



- Every process (p, q, r, s) uc-proposes a value
- Epoch 6 has leader q
    - q ep-proposes x, but only r receives it before epoch aborts
    - r now has state (6,x)
- Epoch 8 has leader s
    - s ep-proposes z, processes p, q, s receive it
    - only p ep-decides(z); then s crashes
- Epoch 11 has leader r, and ep-decides(z)

Implementing consensus

- In asynchronous system with processes prone to crash or Byzantine failures, deterministic algorithms cannot implement consensus [FLP].

- We use a timing assumption, we use ◇P, encapsulated as a leader detection oracle Ω
  - Ω periodically designates a trusted leader
  - Ω is not perfect, may make mistakes

- Variations of Ω can be implemented in partially synchronous systems
  - With crash or Byzantine failures

Eventual Leader Detector (Ω)

- Events :
  - Indication <Ω, trust(p)>

    Indicates that process p is trusted to be leader

- Properties :
  - ELD1 (Eventual accuracy): Eventually every correct process trusts some correct process.
  - ELD2 (Eventual agreement): Eventually no two correct processes trust different processes.

- The trusted leader may change over time, different leaders may be elected, only eventually every process follows the same “good” leader.

Monarchical Eventual Leader Detection

```
Implements:
    EventualLeaderDetector, instance $\Omega$.

Uses:
    EventuallyPerfectFailureDetector, instance $\Diamond\mathcal{P}$.

upon event $\langle\ \Omega, Init\ \rangle$ do
    $suspected := \emptyset$;
    $leader := \perp$;

upon event $\langle\ \Diamond\mathcal{P}, Suspect \mid p\ \rangle$ do
    $suspected := suspected \cup \{p\}$;

upon event $\langle\ \Diamond\mathcal{P}, Restore \mid p\ \rangle$ do
    $suspected := suspected \setminus \{p\}$;

upon $leader \neq \text{maxrank}(\Pi \setminus suspected)$ do
    $leader := \text{maxrank}(\Pi \setminus suspected)$;
    trigger $\langle\ \Omega, Trust \mid leader\ \rangle$;
```

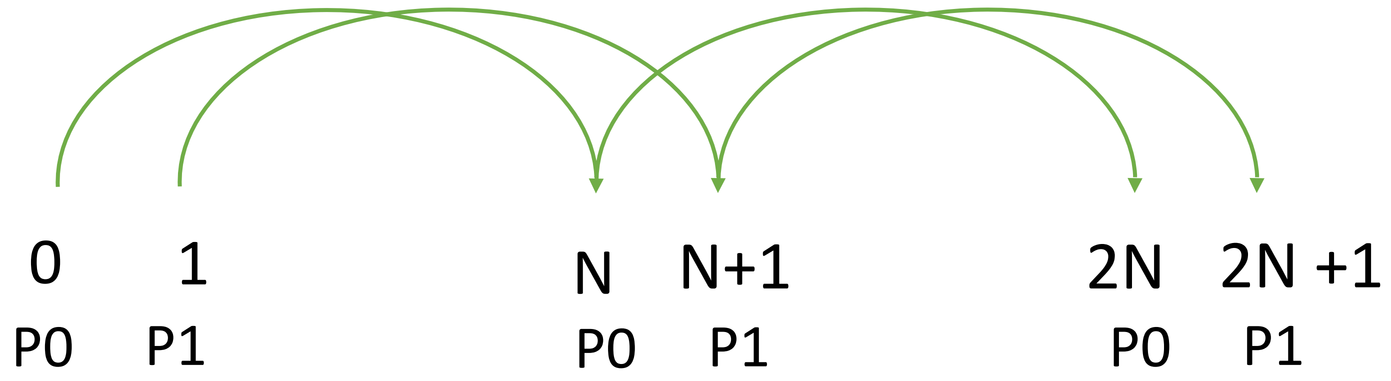
Epoch-Change (ec)

- Events:
  - Indication <ec, start-epoch(ts, L)>

    Starts epoch (ts,L), timestamp ts and leader L
- Properties:
  - EC1 (Monotonicity):
    If a correct process starts epoch (ts,L) and later starts epoch (ts',L'), then ts' > ts.
  - EC2 (Consistency):
    If a correct process starts epoch (ts,L) and another correct process starts epoch (ts,L'), then L = L'.
  - EC3 (Eventual Leadership):
    Eventually every correct process starts no further epoch; moreover, every correct process starts the same last epoch (ts,L), where L is a correct process.

epoch-change

- Use eventual leader detector (Ω)
- A locally increasing timestamp can provide monotonically.
- Maintain current trusted leader and an increasing timestamp.
- However, “before eventually”, Ω does not guarantee the same leader or order of leaders at different processes. This can violate consistency. Therefore, the timestamp domain is disjointly divided between processes.
- For a new leader, we jump to the next timestamp for that process.



epoch-change

- To have eventual leadership, the last leader should be installed with the *same timestamp* at all processes.
- If a process finds himself to be the leader, he broadcasts himself as the leader with a timestamp.
- He might be rejected by a process that has a leader with a larger timestamp.
- The new leader repeats broadcasting with larger timestamps until he does not hear any rejections.

Leader-Based Epoch-Change

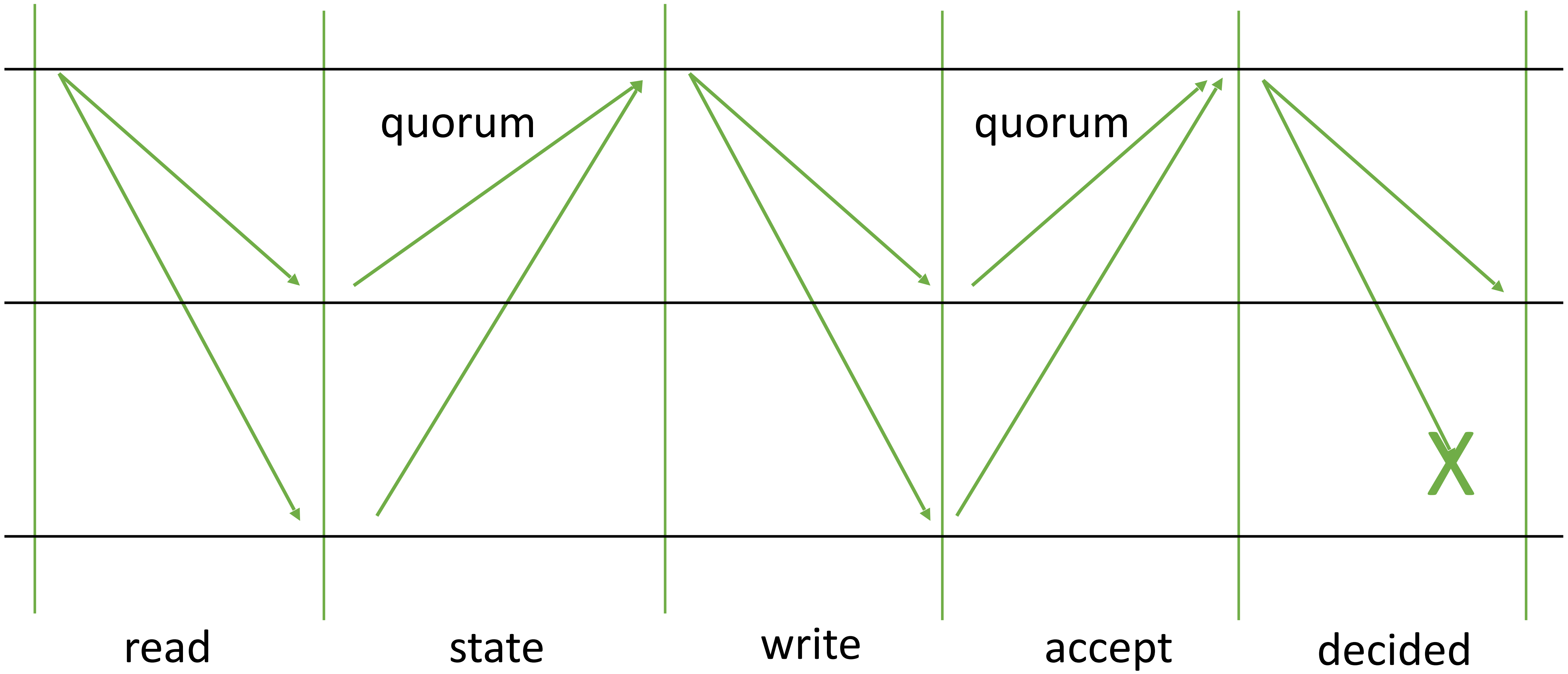
```
Implements:
    EpochChange, instance $ec$.

Uses:
    PerfectPointToPointLinks, instance $pl$;
    BestEffortBroadcast, instance $beb$;
    EventualLeaderDetector, instance $\Omega$.

upon event ⟨ $ec$, Init ⟩ do
    $trusted := \ell_0$;
    $lastts := 0$;
    $ts := rank(self)$;

upon event ⟨ $\Omega$, Trust | $p$ ⟩ do
    $trusted := p$;
    if $p = self$ then
        $ts := ts + N$;
        trigger ⟨ $beb$, Broadcast | [NEWEPOCH, $ts$] ⟩;

upon event ⟨ $beb$, Deliver | $\ell$, [NEWEPOCH, $newts$] ⟩ do
    if $\ell = trusted \wedge newts > lastts$ then
        $lastts := newts$;
        trigger ⟨ $ec$, StartEpoch | $newts, \ell$ ⟩;
    else
        trigger ⟨ $pl$, Send | $\ell$, [NACK] ⟩;

upon event ⟨ $pl$, Deliver | $p$, [NACK] ⟩ do
    if $trusted = self$ then
        $ts := ts + N$;
        trigger ⟨ $beb$, Broadcast | [NEWEPOCH, $ts$] ⟩;
```

trusted is the latest leader that Ω suggested.
lastts is the last timestamp that was accepted for the leader.
ts is the timestamp that this process wants to impose on others (when Ω suggests her as the leader.)

Implementing epoch consensus

- Read/write epoch consensus algorithm
  - Analogous to replicated implementation of a shared single-writer register
- State consists of a timestamp/value pair
- Leader reads state and looks for a value
  - Chooses value with highest timestamp
  - If no value found, takes its own proposal
  - Writes the chosen value
- Decide once a quorum of processes (> N/2) write the value and acknowledge

Epoch Consensus Steps



Read/write epoch consensus

```
**Implements** ep, uses pl, beb when N > 2f with timestamp ets and leader L

upon <ep, init(valts',val')> do
    proposal := ⊥;  states := [⊥]^N;  accepted := 0
    (valts, val) := (valts', val')

upon <ep, propose(v)> do
    proposal := v
    trigger <beb, broadcast(Read)>

upon <beb, deliver(L, Read)> do
    trigger <pl, Send(L, State(valts, val))>

upon <pl, deliver(q, State(ts, v))> do
    states[q] := (ts,v)

upon #(states) > N/2 do
    (ts,v) := highest(states);  states := [⊥]^N
    if v ≠⊥ then proposal := v
    trigger <beb, broadcast(Write(proposal))>
```

Read/write epoch consensus

```
upon <beb, deliver(L, Write(v))> do
    (valts,val) := (ets,v)
    trigger <pl, Send(L, Accept)>

upon <pl, deliver(q, Accept)> do
    accepted := accepted + 1

upon accepted > N/2 do
    accepted := 0
    trigger <beb, broadcast(Decided(proposal))>

upon <pl, deliver(L, Decided(v))> do
    trigger <ep, decide(v)>

upon <ep, abort> do
    trigger <ep, aborted(valts,val)>
```

Epoch-Consensus (ep)

Associated with timestamp ts and leader L

Events:

- Request <ep, propose(v)>

  Proposes v for epoch consensus (executed by leader only)
- Request <ep, abort>

  Aborts this epoch consensus
- Indication <ep, decide(v)>

  Outputs decided value v for epoch consensus
- Indication <ep, aborted(s)>

  Signals that this epoch consensus has completed the abort and returns state s

Epoch-Consensus (ep)

- Properties
  - EP1 (Validity):
    If a correct process decides v in an epoch ts, then v was proposed by the leader of some epoch consensus with ts' ≤ ts.
  - EP2 (Uniform Agreement):
    No two [correct*] processes decide differently. *for Byzantine epoch consensus
  - EP3 (Integrity):
    A correct process decides at most once.
  - EP4 (Lock-in):
    If a process decides v in epoch ts’ < ts, no process decides a value different from v in epoch ts.
  - EP5 (Termination):
    If the leader L is correct, has proposed a value and no process aborts, then every correct process eventually decides.
  - EP6 (Abort behavior):
    When a correct process aborts, then it eventually completes the abort; plus, a correct process completes an aborts only if it has been aborted before.
- Every process must run a well-formed sequence of epoch consensus instances, only one instance of epoch consensus at a time. Give state from previous (aborted) instance to next instance. Associated timestamps monotonically increasing.

Correctness

- Validity (EP1)
  - The decided value either comes from a State message or from the leader L itself.
  - The value of a State message has been (inductively) written by some leader.
- Uniform Agreement (EP2)
  - Immediate from the fact that every process decides only the value of the Decided message. The leader sends the Decided message only once.
- Integrity (EP3)
  - The Decided message is sent at most once: when it is sent, the accepted variable is set to zero and cannot get more than N/2 again.
- Lock-in (EP4)
  - A write-quorum (> N/2) stored v before sending the Accept message in previous epoch ts'.
  - Processes passed it in the state to subsequent epochs to ts.
  - Then, L reads v in at least one State message from a read-quorum (> N/2).
- Termination (EP5)
  - A majority is correct, and the correct leader can complete all the steps.
- Abort behavior (EP6)
  - Immediate from the algorithm.

Uniform Consensus (uc)

- Events :
  - Request <uc, Propose(v)>

    Proposes value v for consensus
  - Indication <uc, decide(v)>

    Outputs a decided value v of consensus

- Properties :
  - UC1 (Termination): Every correct process eventually decides.
  - UC2 (Validity): Any decided value has been proposed by some process.
  - UC3 (Integrity): No process decides twice.
  - UC4 (Uniform Agreement): No two processes* decide differently.

* both correct or faulty

Correctness

- Termination (UC1 / WBC1)
  - From EC3 (eventual leadership), EP5 (termination) and the algorithm (no redundant aborts)
- Validity (UC2) / Weak Validity (WBC2)
  - From EP1 (validity) and the algorithm (leaders store and propose user proposals).
- Integrity (UC3)
  - The decided variable in the algorithm.
- Uniform Agreement (UC4 / WBC4)
  - From the algorithm (decide only after decide from ep), EP2 (agreement), and EP4 (lock-in)

We will talk about WBC (Weak Byzantine Consensus) in future lectures.

Parts of slides adopted from C. Cachin, R. Guerraoui, L. Rodrigues