Consensus

Mohsen Lesani

Consensus

Propose(a)

Propose(b)

Propose(c)

Consensus

Propose(a)
Decide(?)

Propose(b)
Decide(?)
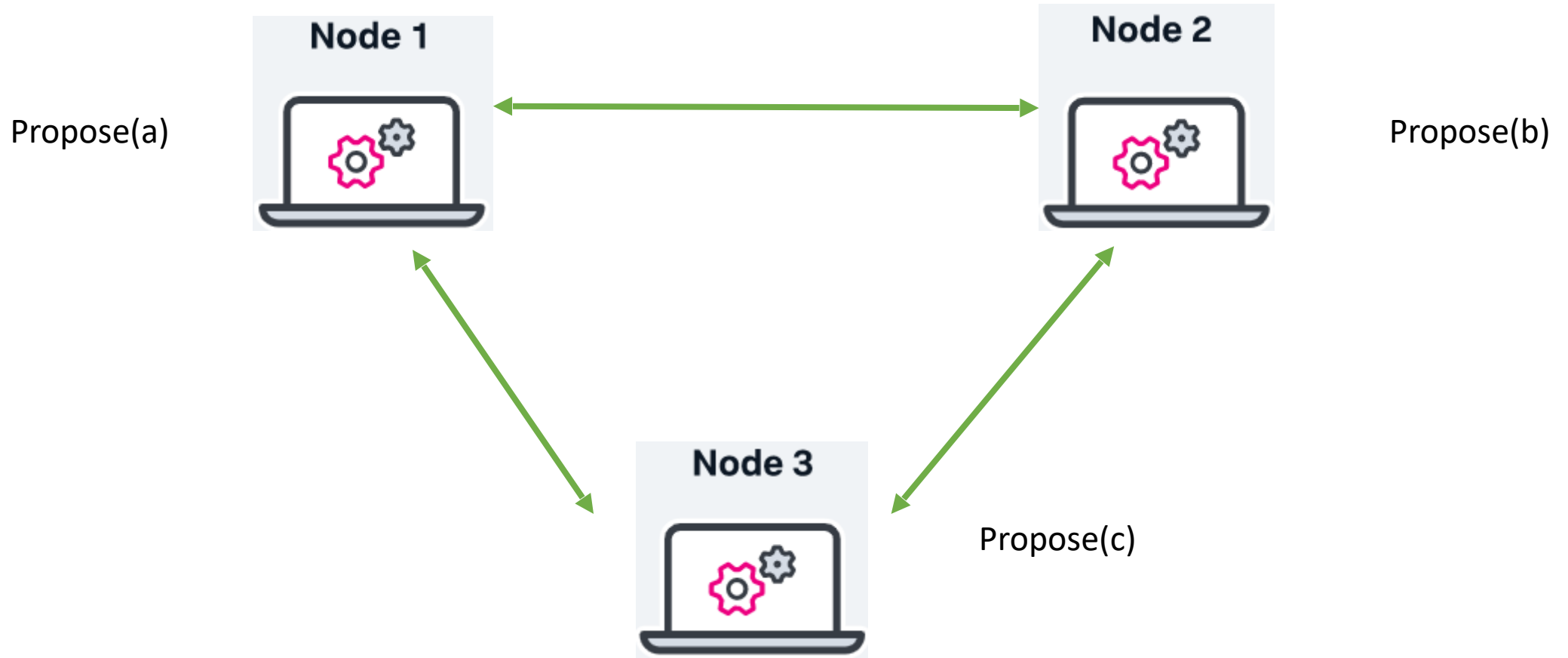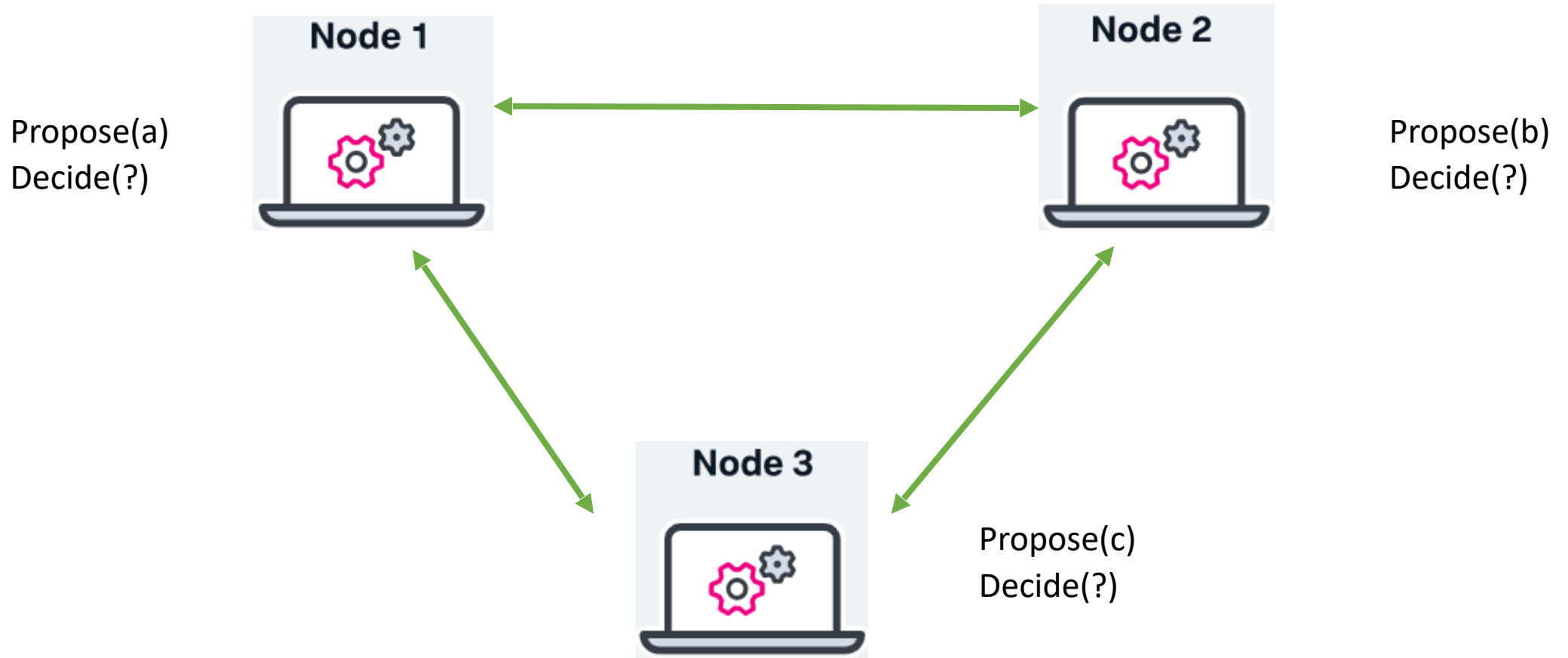
Propose(c)
Decide(?)

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) >
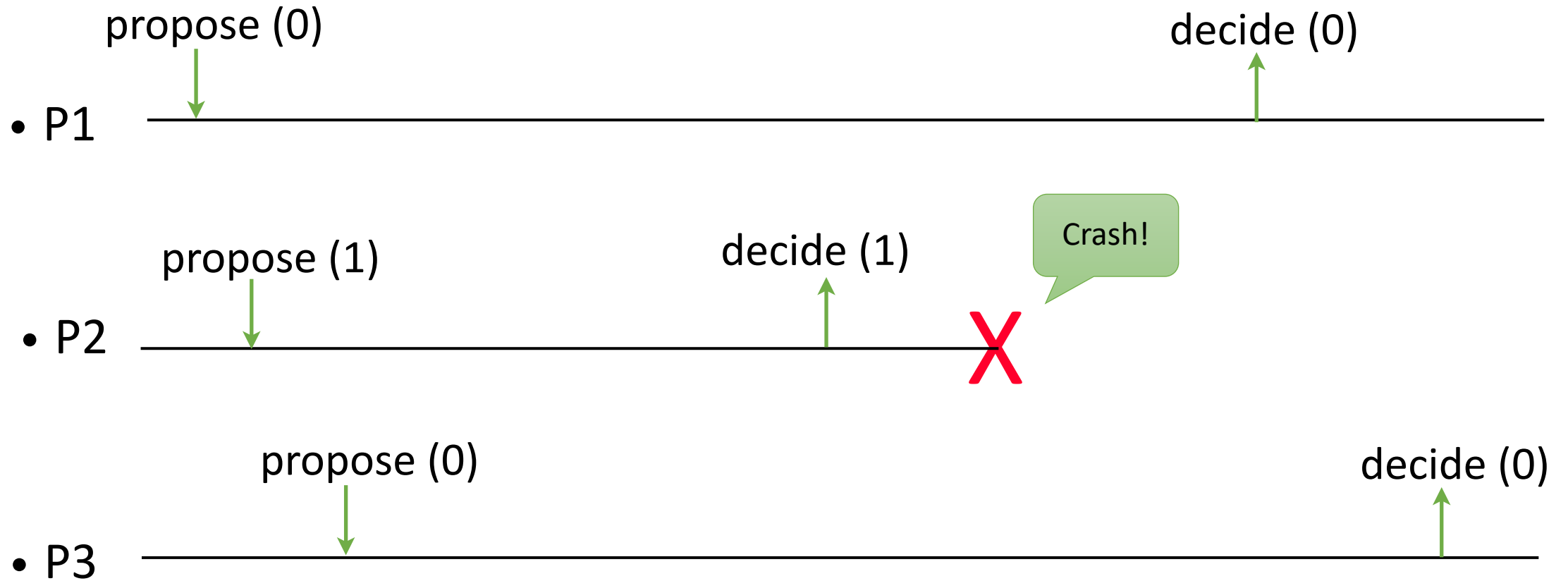 - Indication: < decide(v) >

- **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

P1: propose (0) → decide (0)

P2: propose (1) → decide (1) → Crash! ✗

P3: propose (0) → decide (0)

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
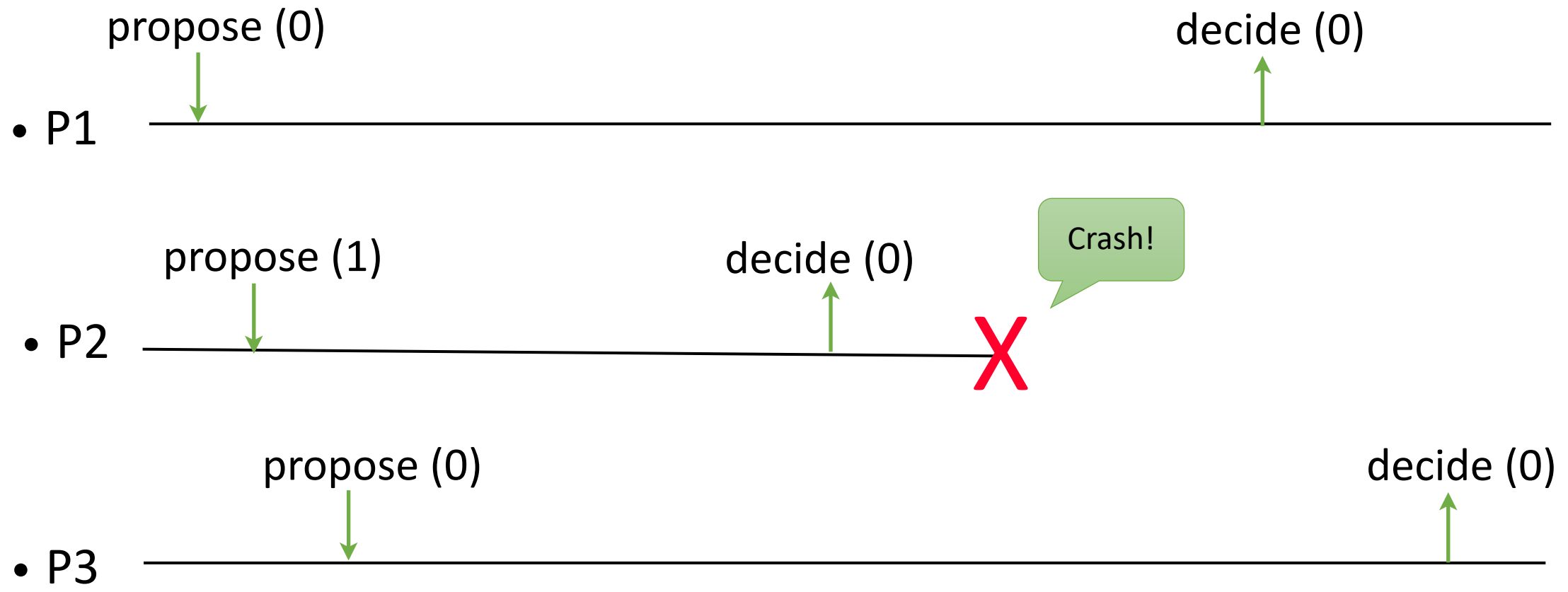
Modules of a process
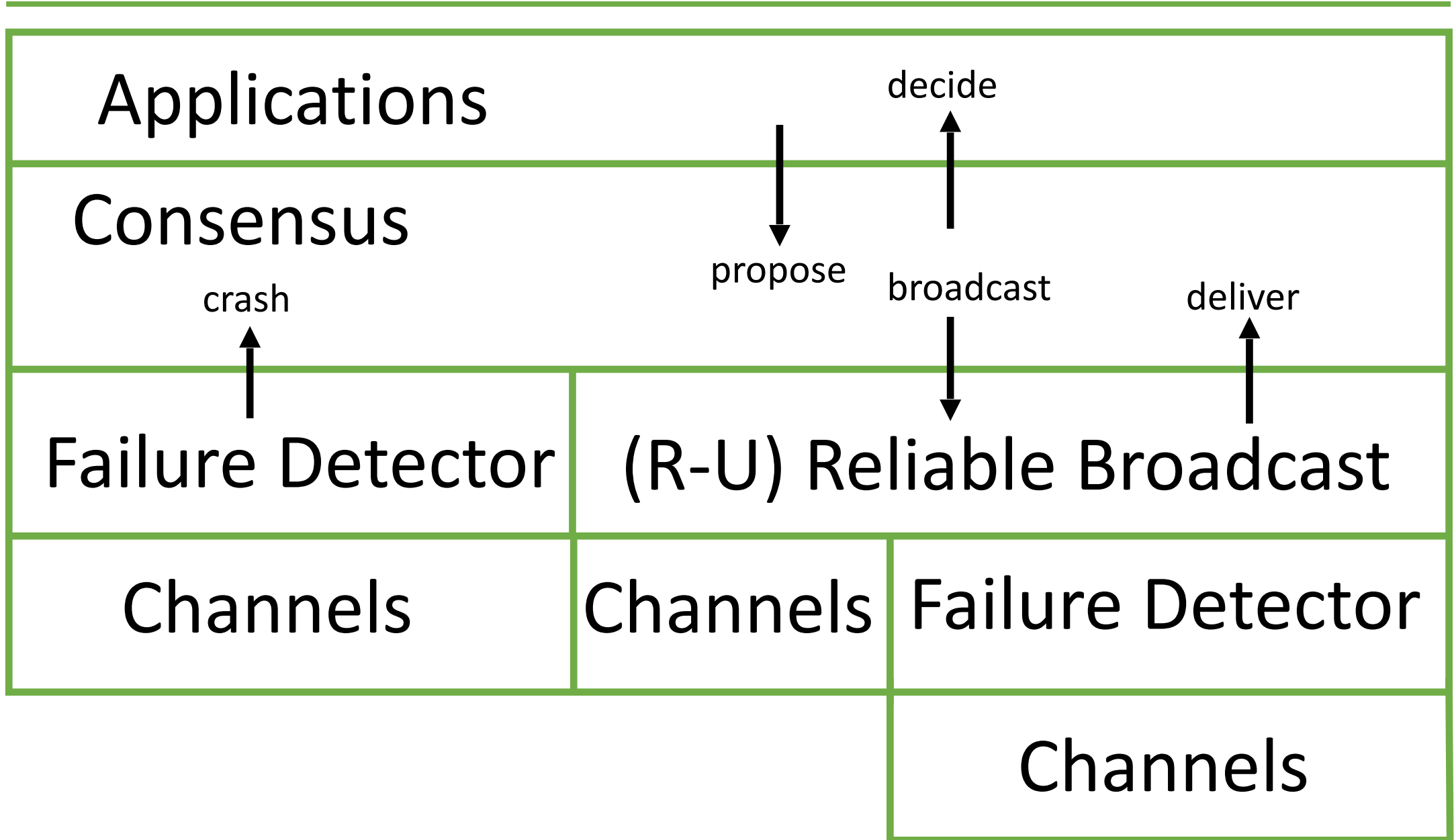
Three algorithms

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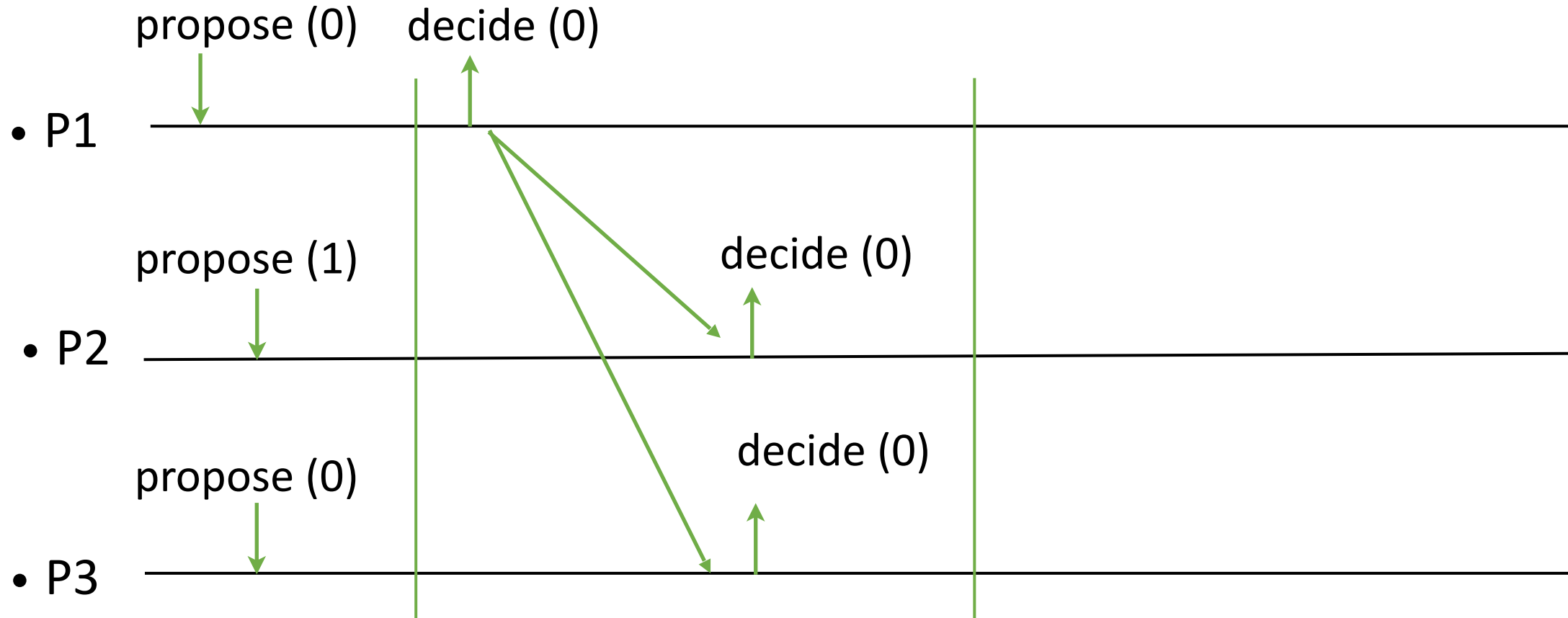
P is the perfect failure detector.
<>P is the eventually perfect failure detector.

Fail-stop Consensus

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

P is the perfect failure detector.

Fail-stop Consensus

propose (0)   decide (0)

• P1

propose (1)   decide (0)

• P2

propose (0)   decide (0)

• P3

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

Fail-stop Consensus

propose (0)   decide (0)

• P1

propose (1)

• P2

propose (0)

decide (0)

• P3
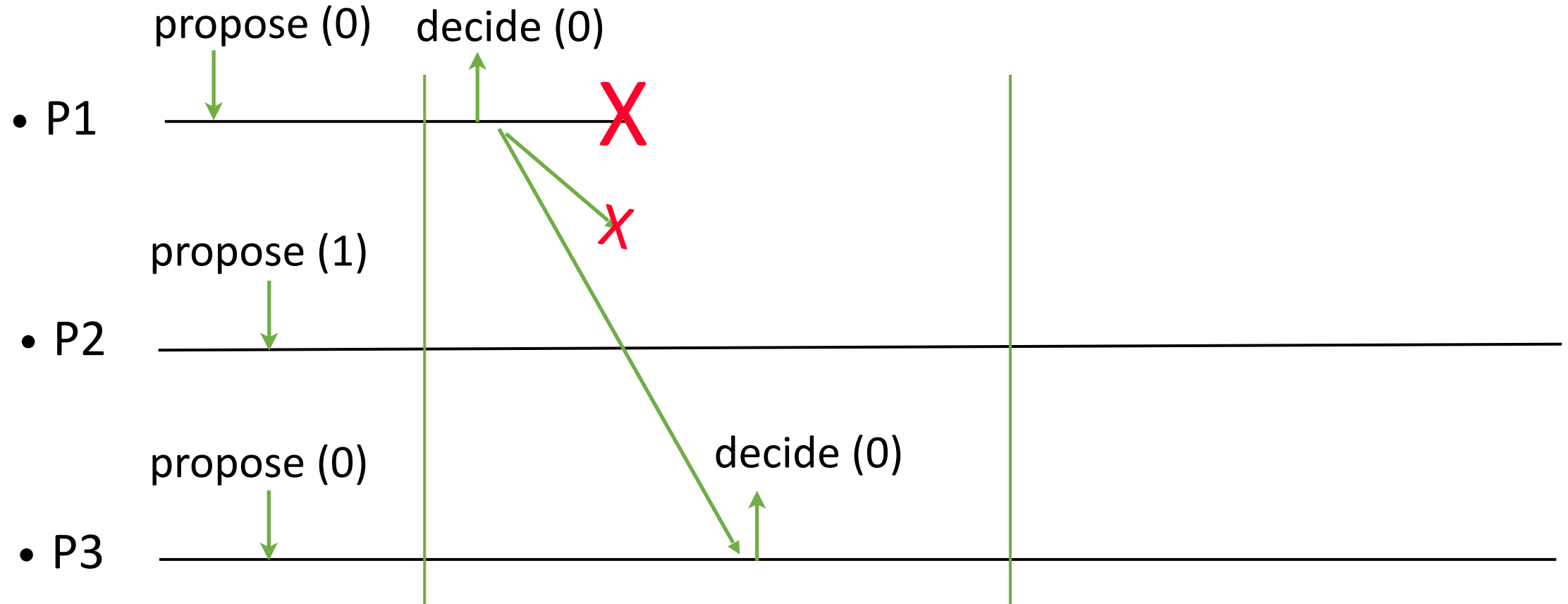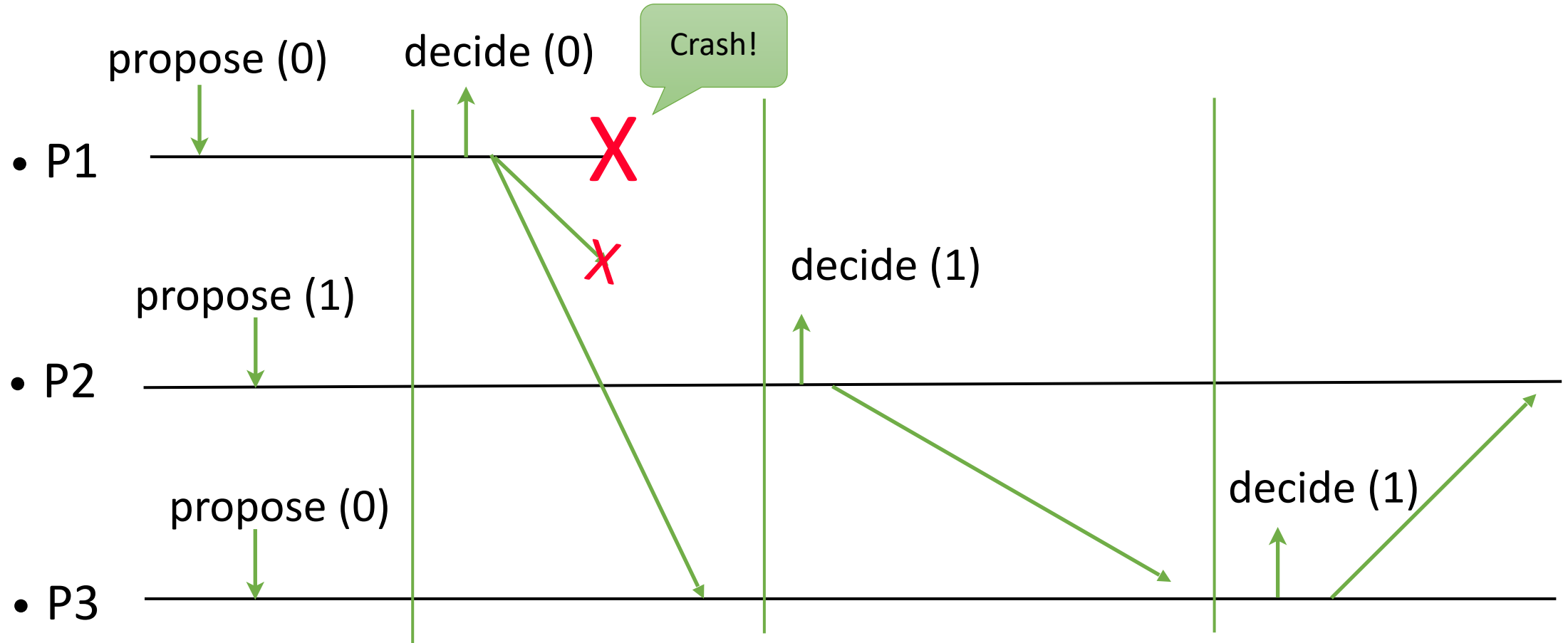
What if he crashes? P2 never decides.

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

Fail-stop Consensus

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.

Fail-stop Consensus

propose (0)   decide (0)

P1

propose (1)

decide (0)

P2

propose (0)

decide (0)

P3
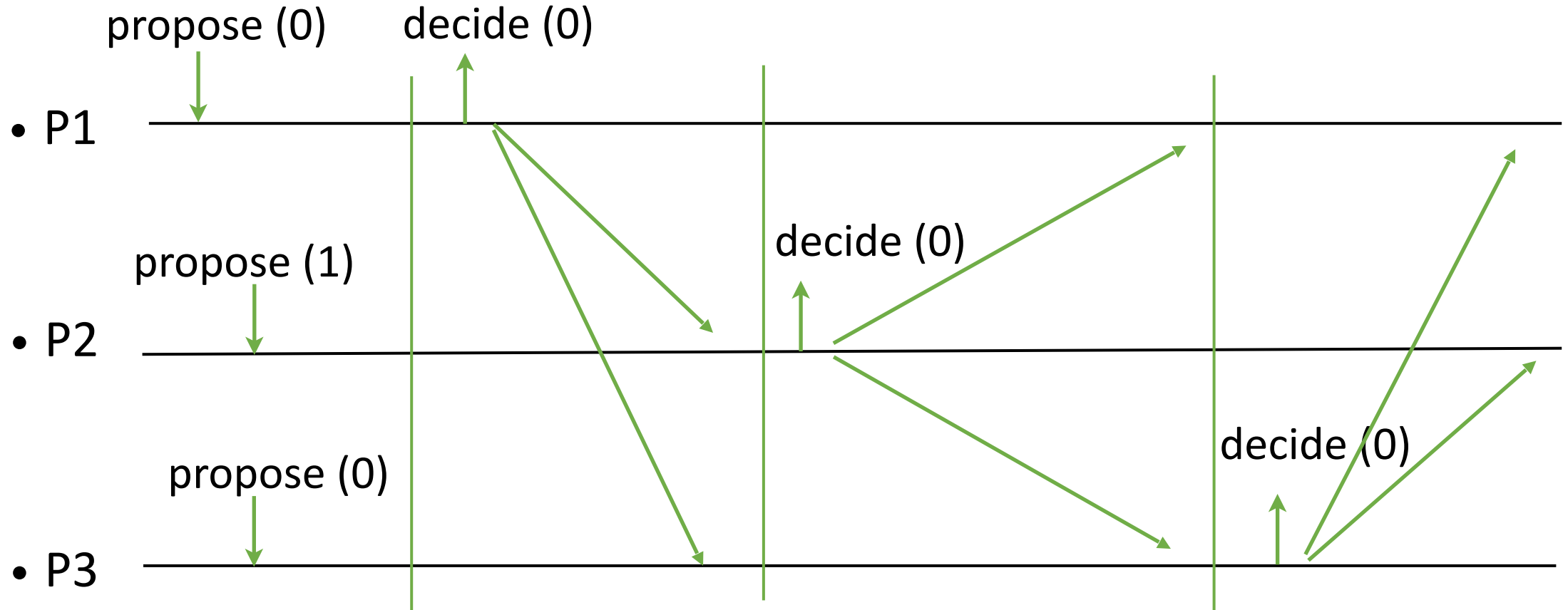
The first correct leader propagates its value throughout the system. Later leaders go through the later rounds because they don't know whether there has been a correct leader before them. The leader of a round could be the first correct leader.

Fail-stop Consensus

- 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

Fail-stop Consensus

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.

Fail-stop Consensus

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.

Fail-stop Consensus

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.

Fail-stop Uniform Consensus

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

Non-uniformity in the previous protocol

propose (0)   decide (0)   Crash!

propose (1)   decide (1)

propose (0)   decide (1)

P1

P2

P3
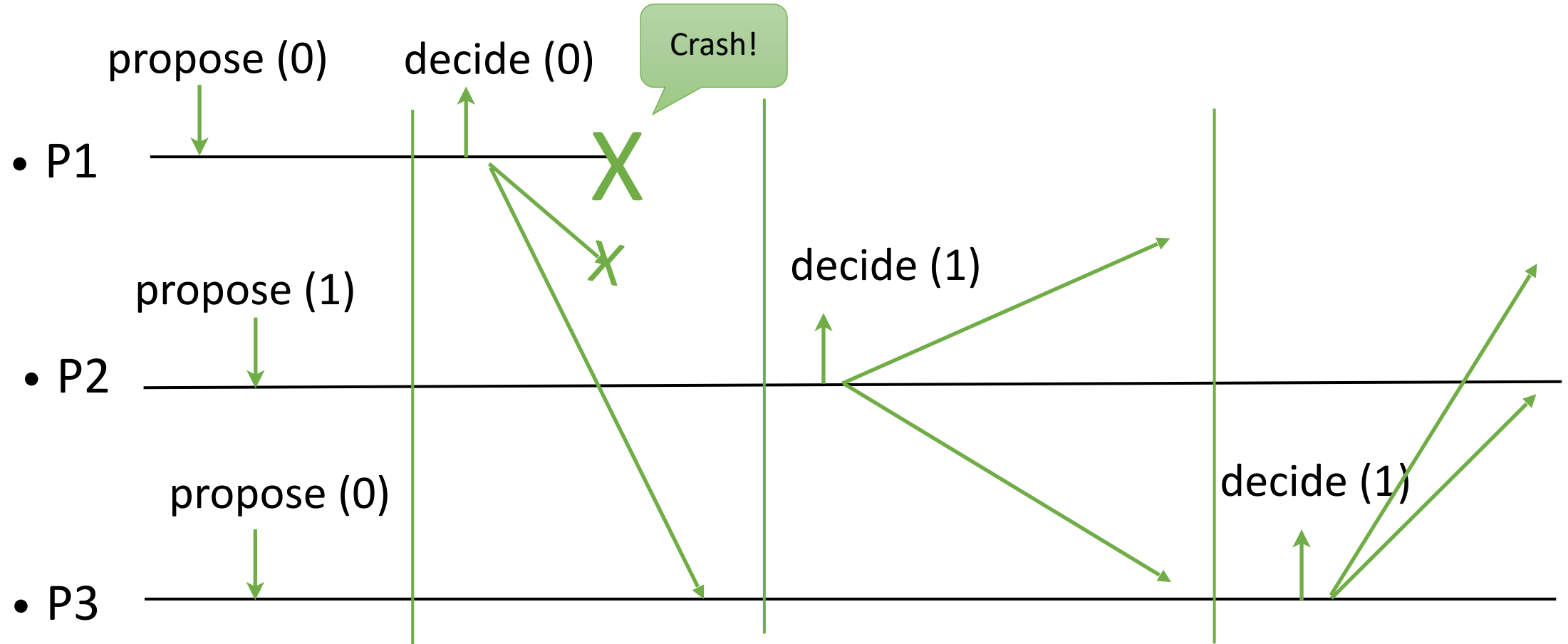
Fail-stop Uniform Consensus

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.
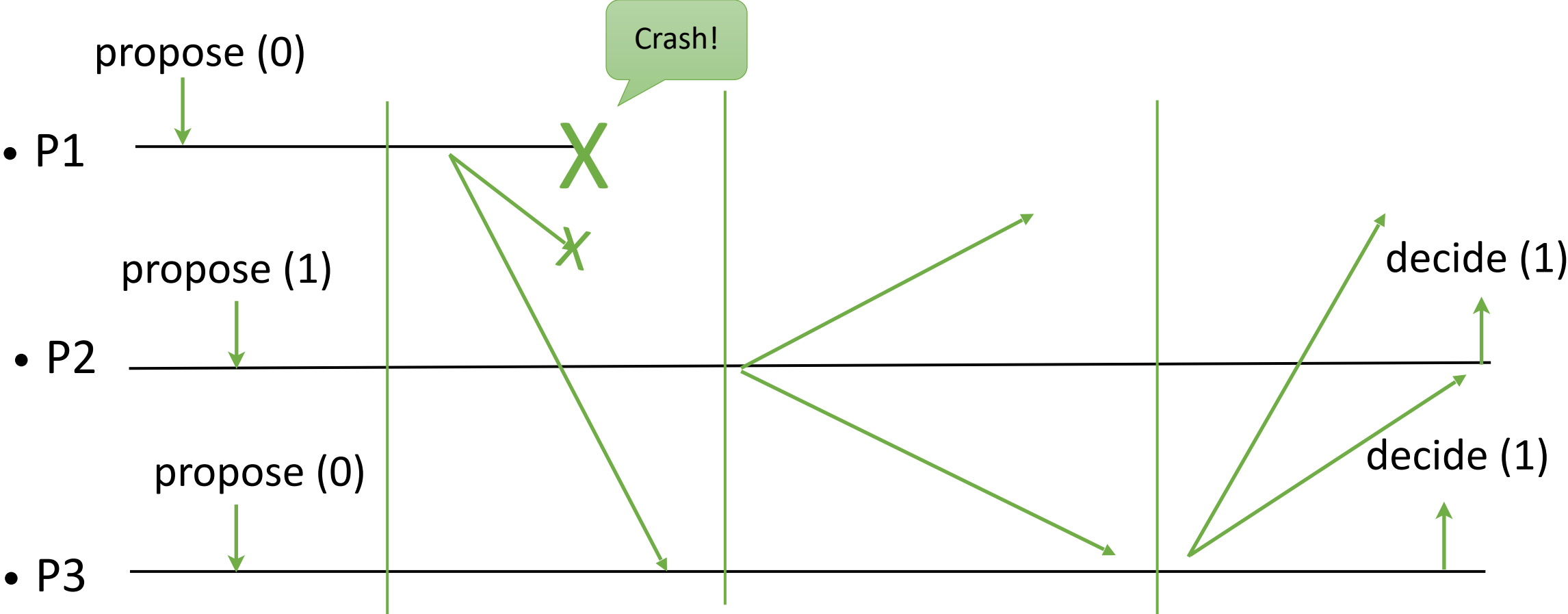
Fail-stop Uniform Consensus

Fail-stop Uniform Consensus

propose (0)

decide (0)

• P1

propose (1)

decide (0)

• P2

propose (0)

decide (0)

• P3
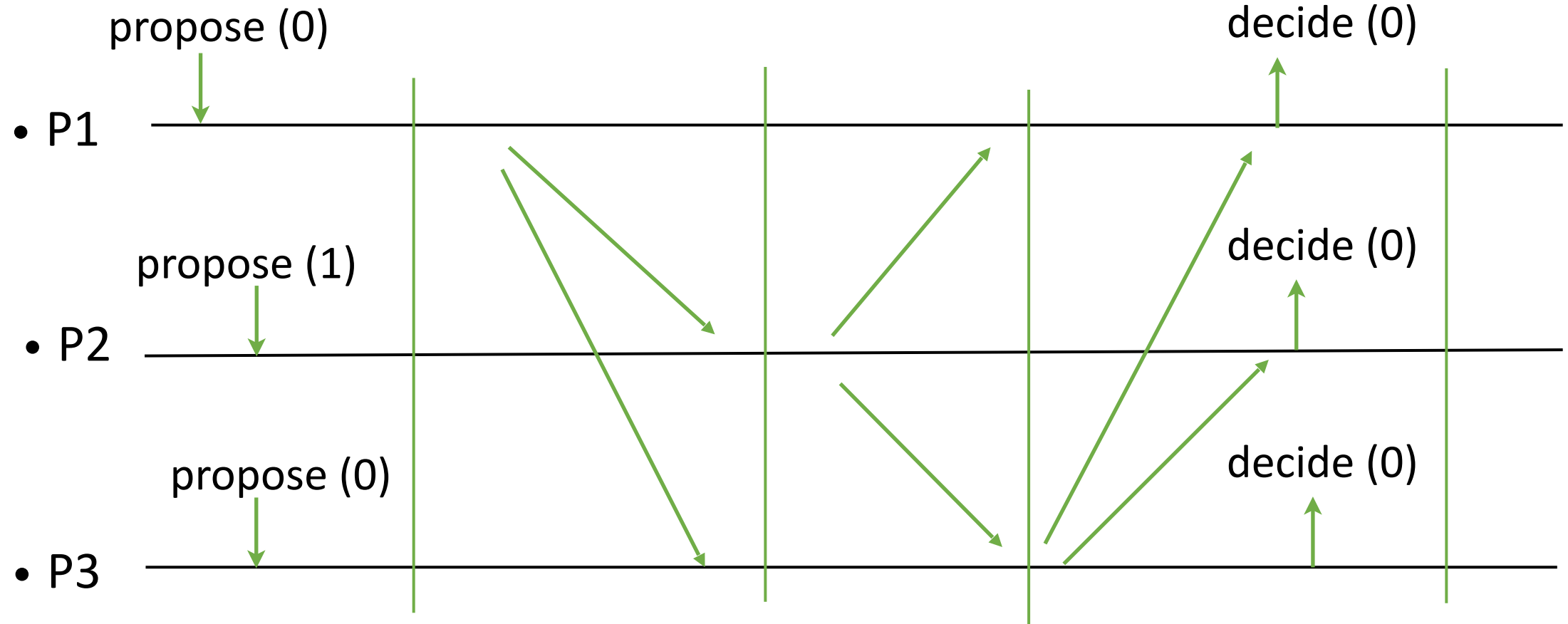
Fail-stop Uniform Consensus

- 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.

Fail-stop Uniform Consensus

Implements: Uniform Consensus (ucons).

Uses:

    BestEffortBroadcast (beb).

    PerfectFailureDetector (P).

upon event < Init > **do**

    suspected := $\varnothing$

    round := 1; prop :=$\perp$

    broadcast := delivered[] := false

    decided := false

upon event < propose(v) > **do**

    if prop = $\perp$ **then**

        prop := v

upon event < P, crash(pi) > **do**

    suspected := suspected U {pi}

> decided is used to not decide twice.

upon event p_{round}=self **and** prop $\neq\perp$ **and** broadcast=false **do**

    trigger <beb, broadcast(prop)>
    broadcast := true

Fail-stop Uniform Consensus

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.

<>P-based Consensus

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

<>P is the eventually perfect failure detector.

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

- We assume a partially synchronous systems where we have an eventually perfect failure detector ◇P.

<>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 the previous two protocols.

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.

Agreement violated with <>P in protocol 1

propose (0)   decide (0)

P1

propose (1)

decide (1)

P2

propose (0)   crash(p1)

P3

crash(p1)
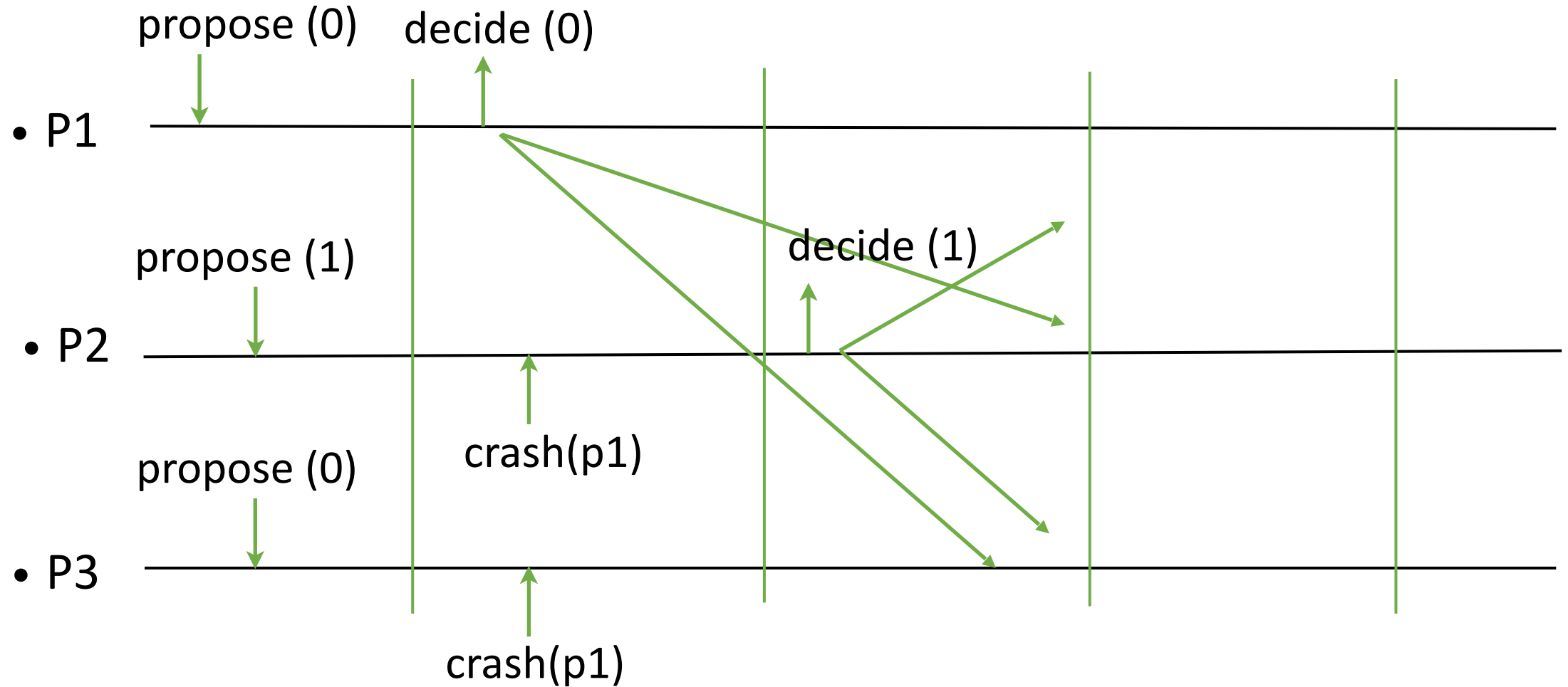
The processes p2 and p3 move to the next round too early. Process P2 decides 1. Agreement is violated.

Agreement violated with <>P in protocol 2

propose (0)

decide (0)

P1

propose (1)

crash (p2)

crash(p3)

decide (1)

P2

crash(p1)

decide (1)

propose (0)

P3

crash(p1)

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

<>P-based Consensus

The idea:

A correct leader tells processes what to decide.

A leader can fail or be 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.

Leader election

- Processes move through rounds.

- Simplifying assumption: Process pi is leader in round k if k mod N = i.

- The processes alternate in the role of a **leader** until one of them succeeds in imposing a decision.

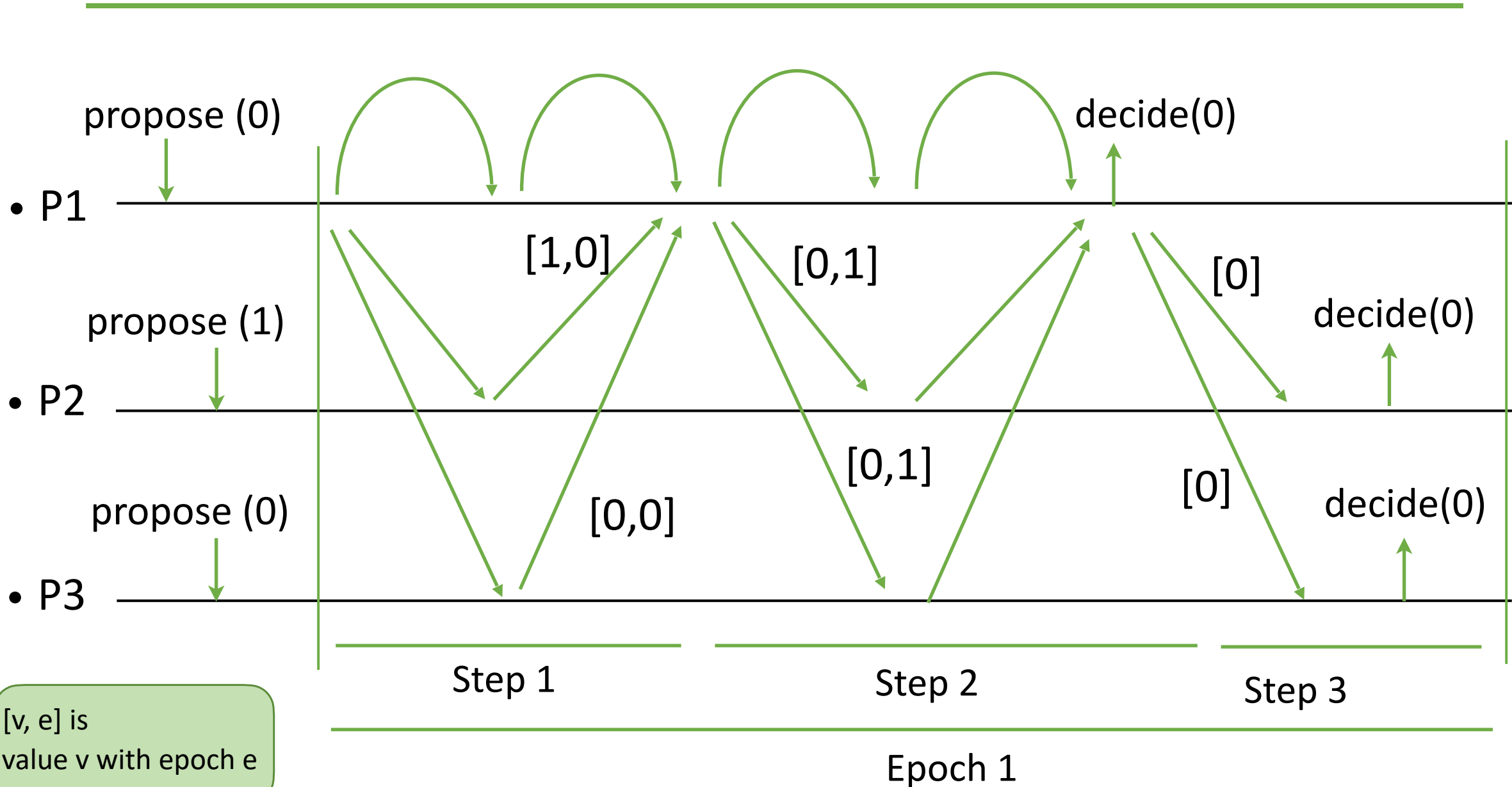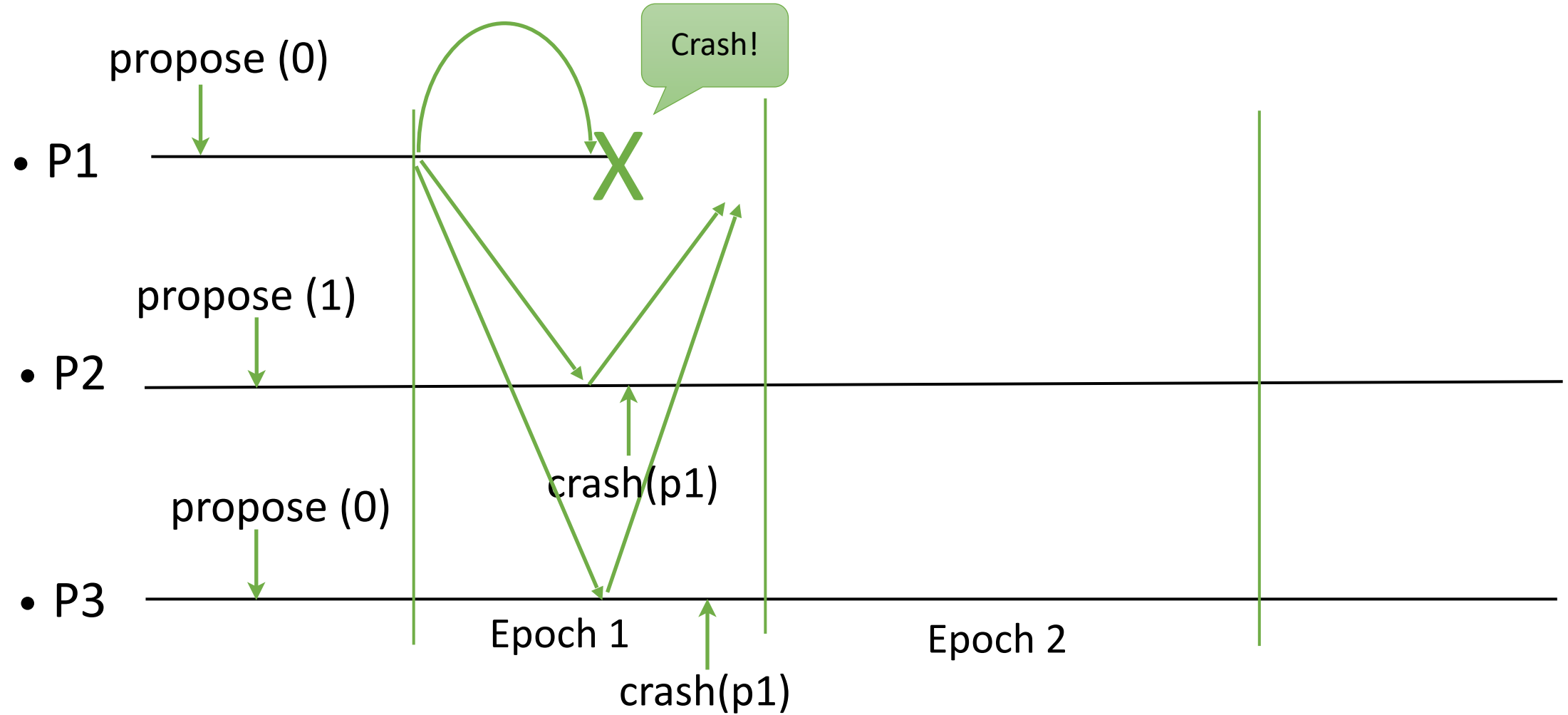# <>P-based Consensus
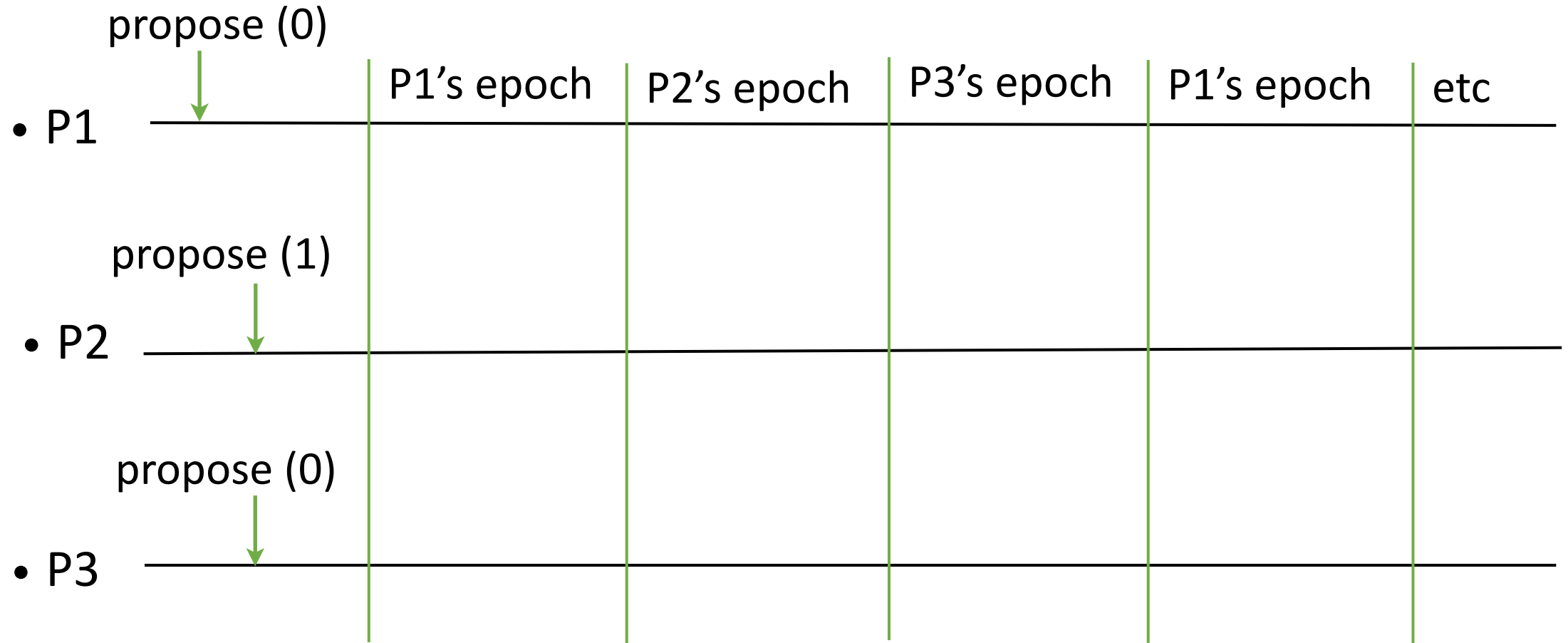
<>P-based Consensus

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 to a majority.

3. It decides and broadcasts the decision to all.

<>P-based Consensus

propose (0)

decide(0)

• P1

[1,0]   [0,1]   [0]

propose (1)

decide(0)

• P2

[0,1]

propose (0)   [0,0]   [0]

decide(0)

• P3

Step 1   Step 2   Step 3

[v, e] is
value v with epoch e

Epoch 1

<>P-based Consensus

<>P-based Consensus

propose (0)

propose (1)

propose (0)

decide (0)

P1

P2

P3

[1,0]

[0,1]

[0,1]

[1,0]

[0,0]

[0,1]

[0,1]

Step 1

Step 2

Step 3
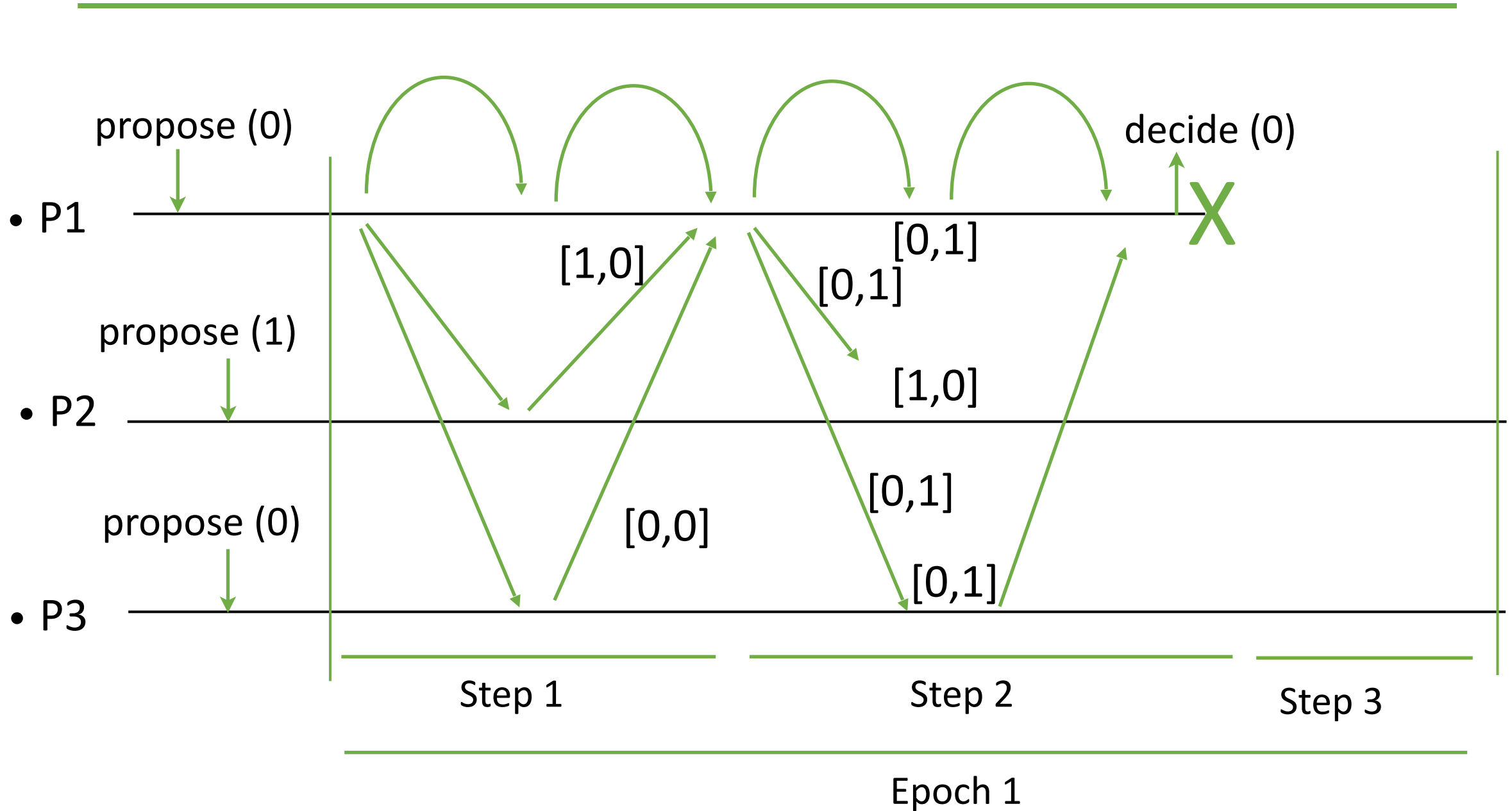
Epoch 1
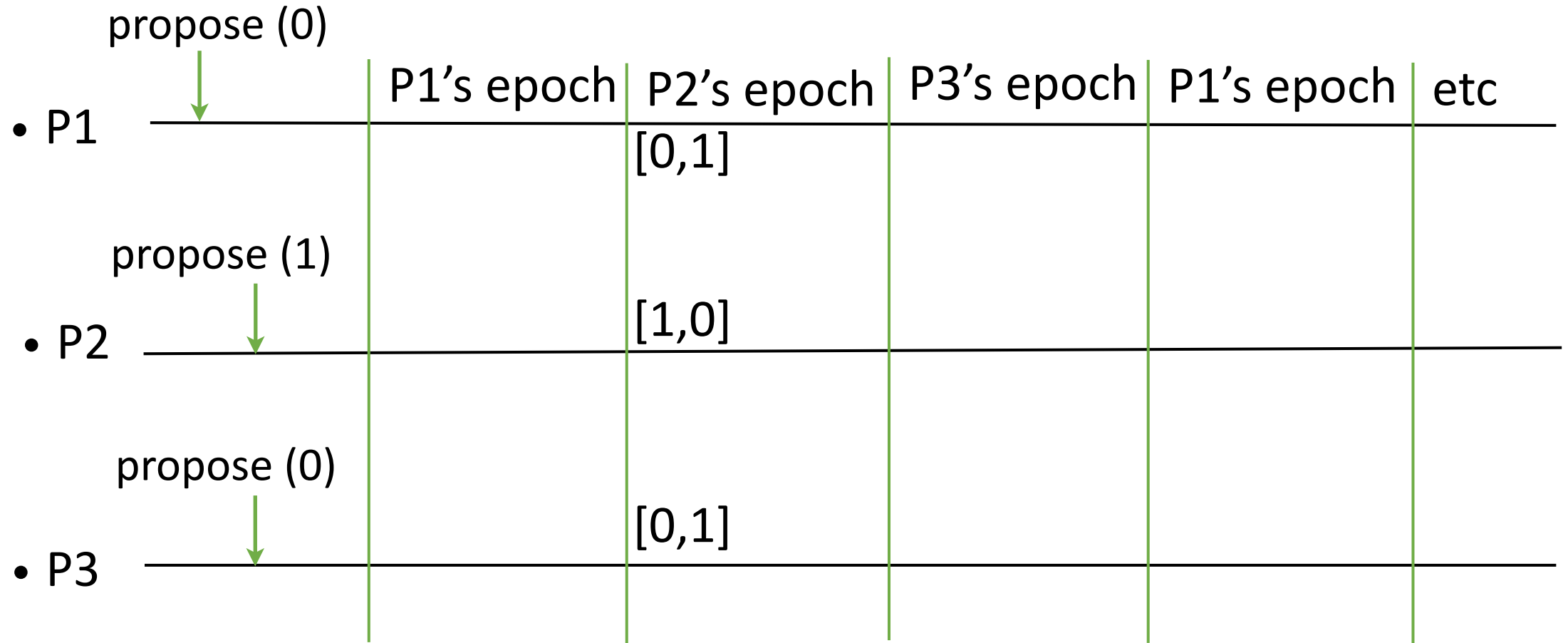
<>P-based Consensus

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

<>P-based Consensus

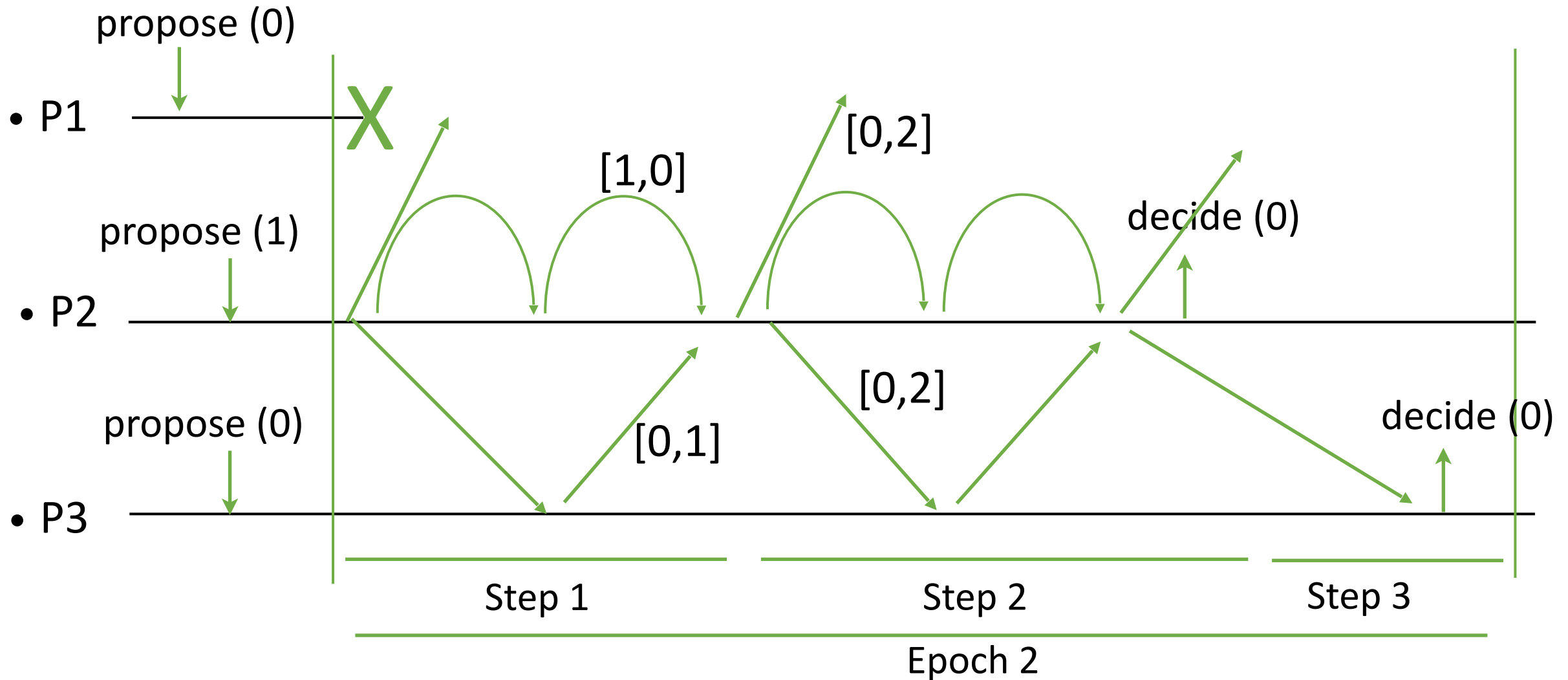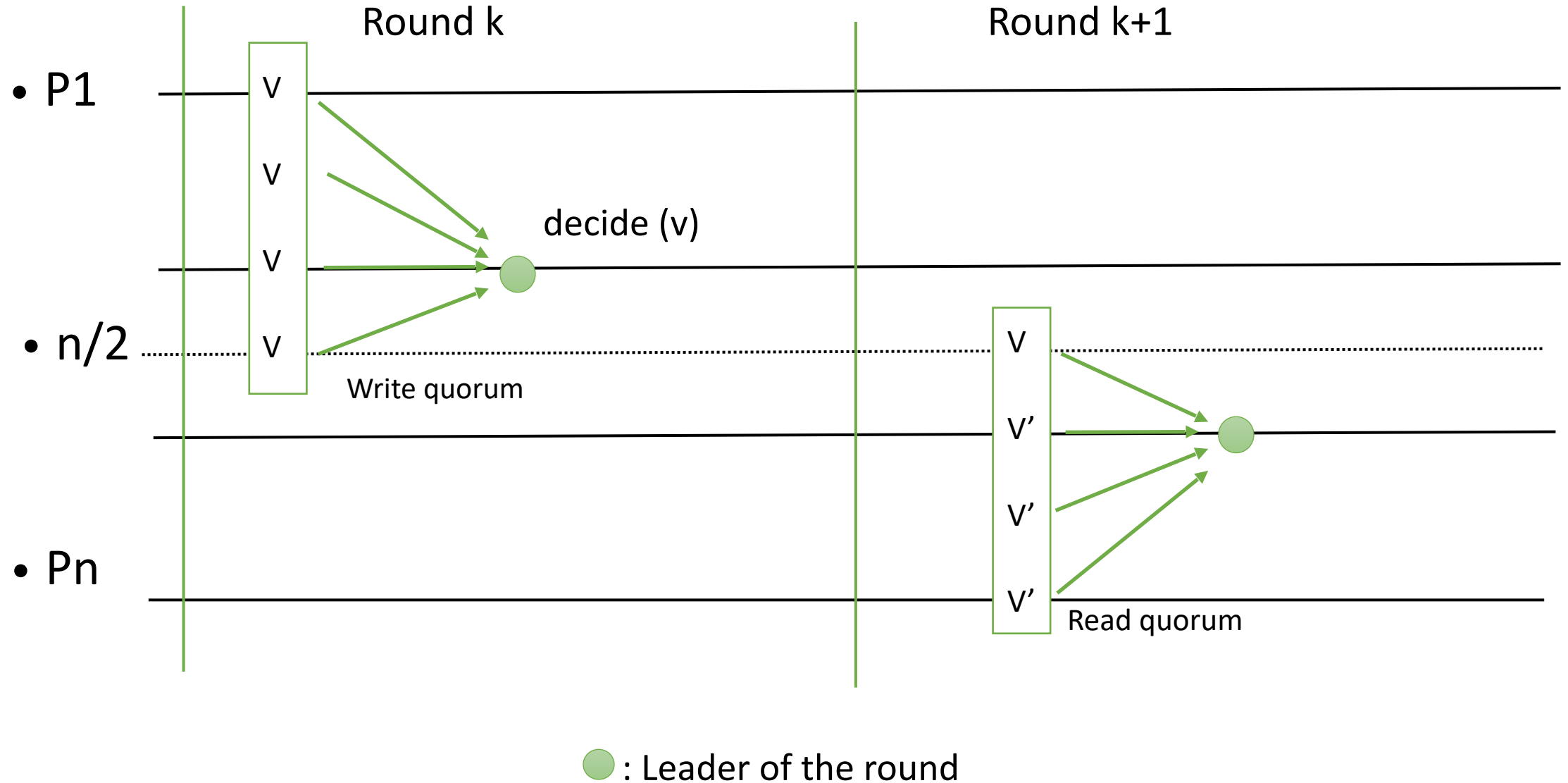The process p2 adopts the already decided value 0 from a majority and imposes it.

Quorums intersect

Leader-driven consensus
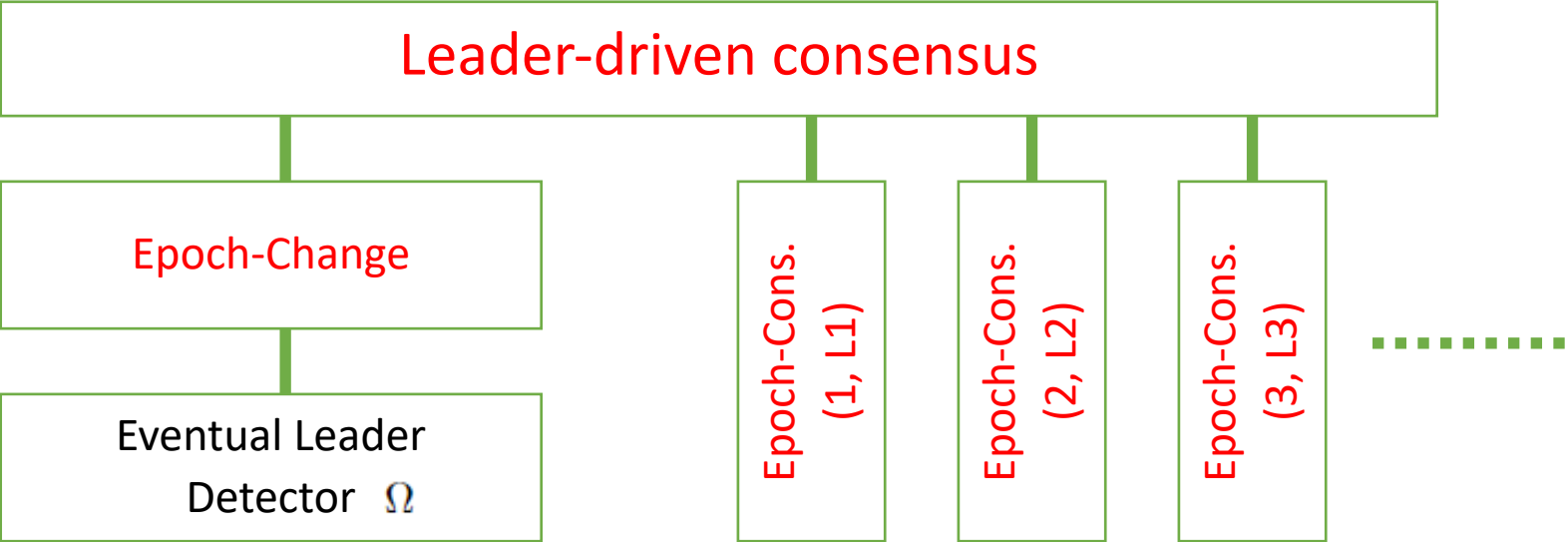
Epoch Consensus Steps

quorum   quorum

read   state   write   accept   decided
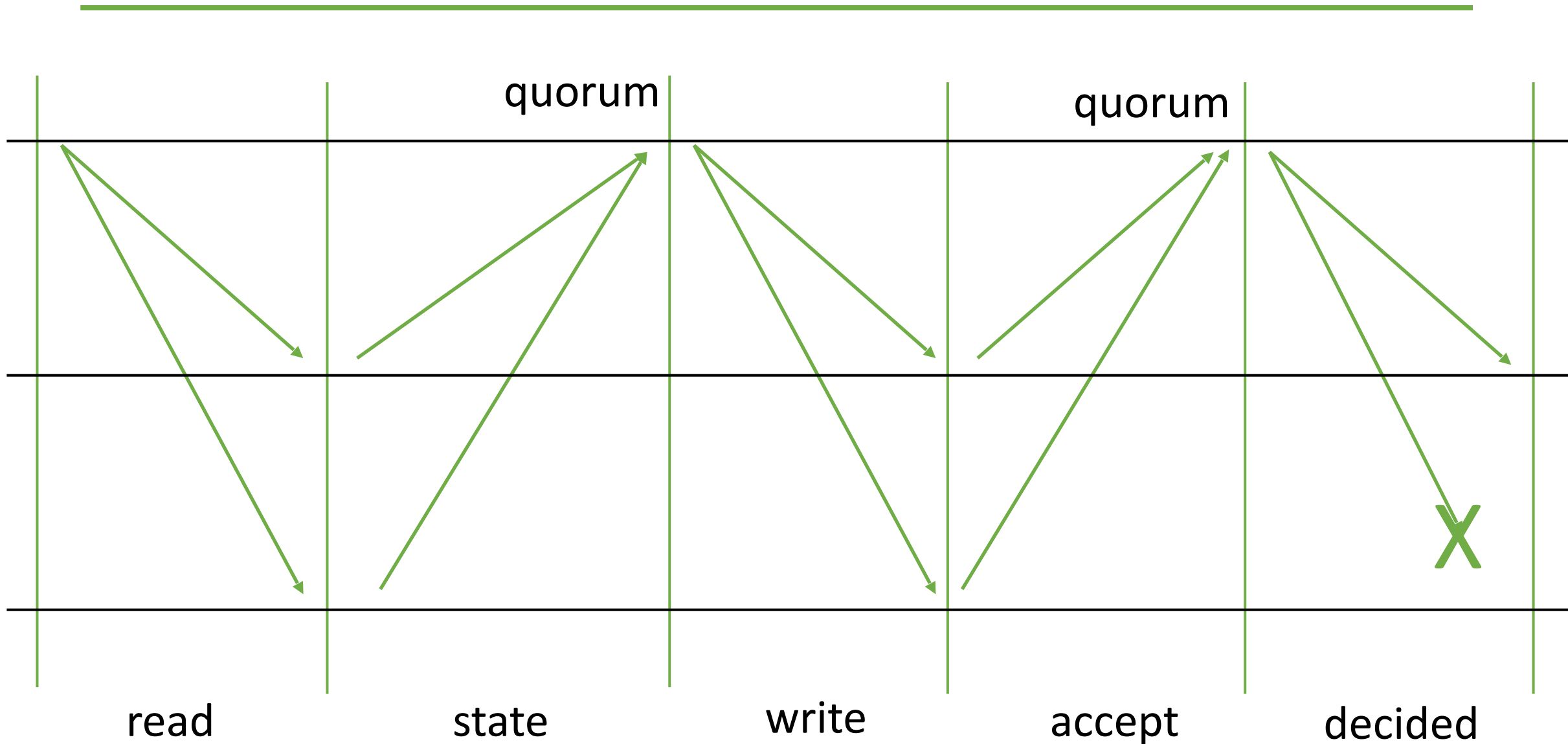
Epoch-Consensus (ep)

Associated with timestamp ts and leader L

Events:

- Request <ep, propose(v)>

  Proposes v for epoch consensus (executed by leader only)

- Indication <ep, decide(v)>

  Outputs decided value v for epoch consensus

- Request <ep, abort>

  Aborts this epoch consensus

- Indication <ep, aborted(s)>

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

Implementing epoch consensus

- 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 write the value and acknowledge
  - Broadcast the decision.

- Analogous to the shared single-writer register protocol

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(p, State(ts, v))> **do**
    states[p] := (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)

- 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.

  - 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 eventual correct leader can complete all the steps.

- Abort behavior (EP6)
  - Immediate from the algorithm.

Leaders

How are leaders elected and changed in rounds?

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 of a process may change over time.
  The sequence of leaders might be different across processes.
  Eventually processes follow the same correct leader.

Implements:
    EventualLeaderDetector, **instance** Ω.

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

upon event $\langle\, \Omega,\, \textit{Init}\, \rangle$ **do**
    $\textit{suspected} := \emptyset$;
    $\textit{leader} := \bot$;

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

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

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

Leader election
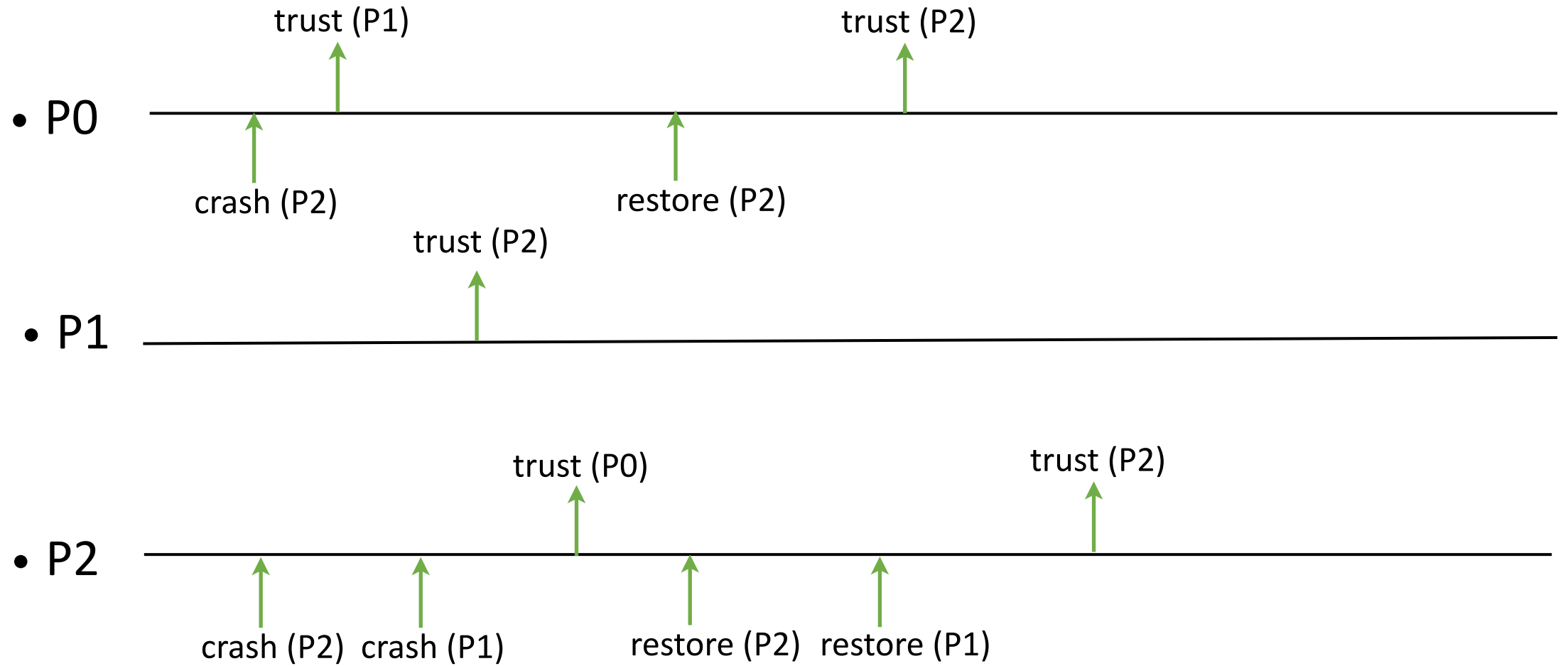
Leaders

How do we associate leaders to rounds?

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, Monotonicity

- Use eventual leader detector (Ω)

- A locally increasing timestamp can provide monotonicity.

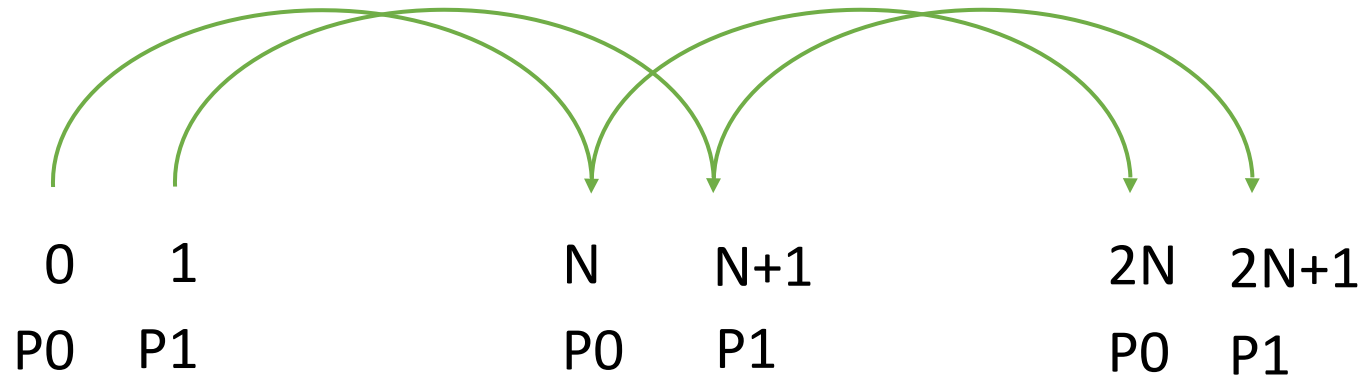- Maintain current trusted leader and an increasing timestamp.

Epoch-change, Consistency

- However, "before eventually", Ω does not guarantee the same leader or order of leaders at different processes. This can violate consistency.
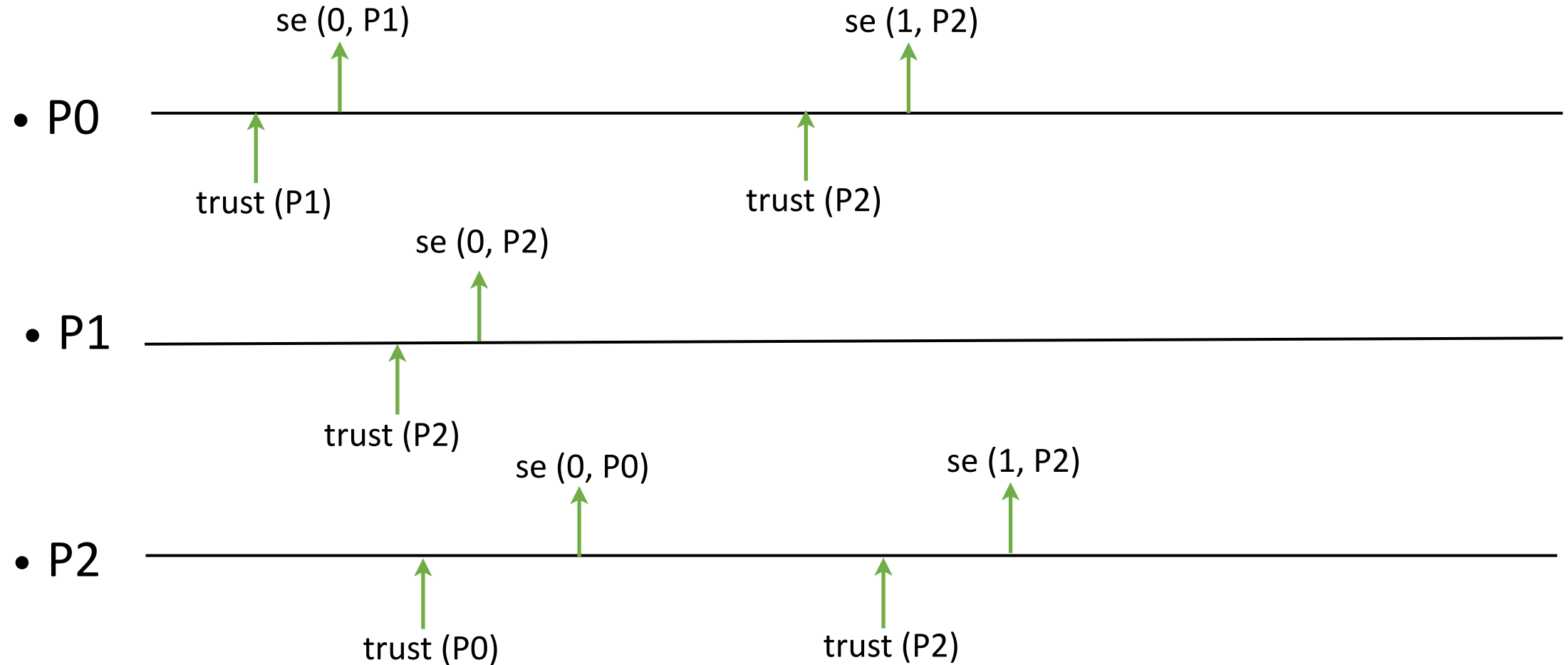
Epoch-change, Consistency

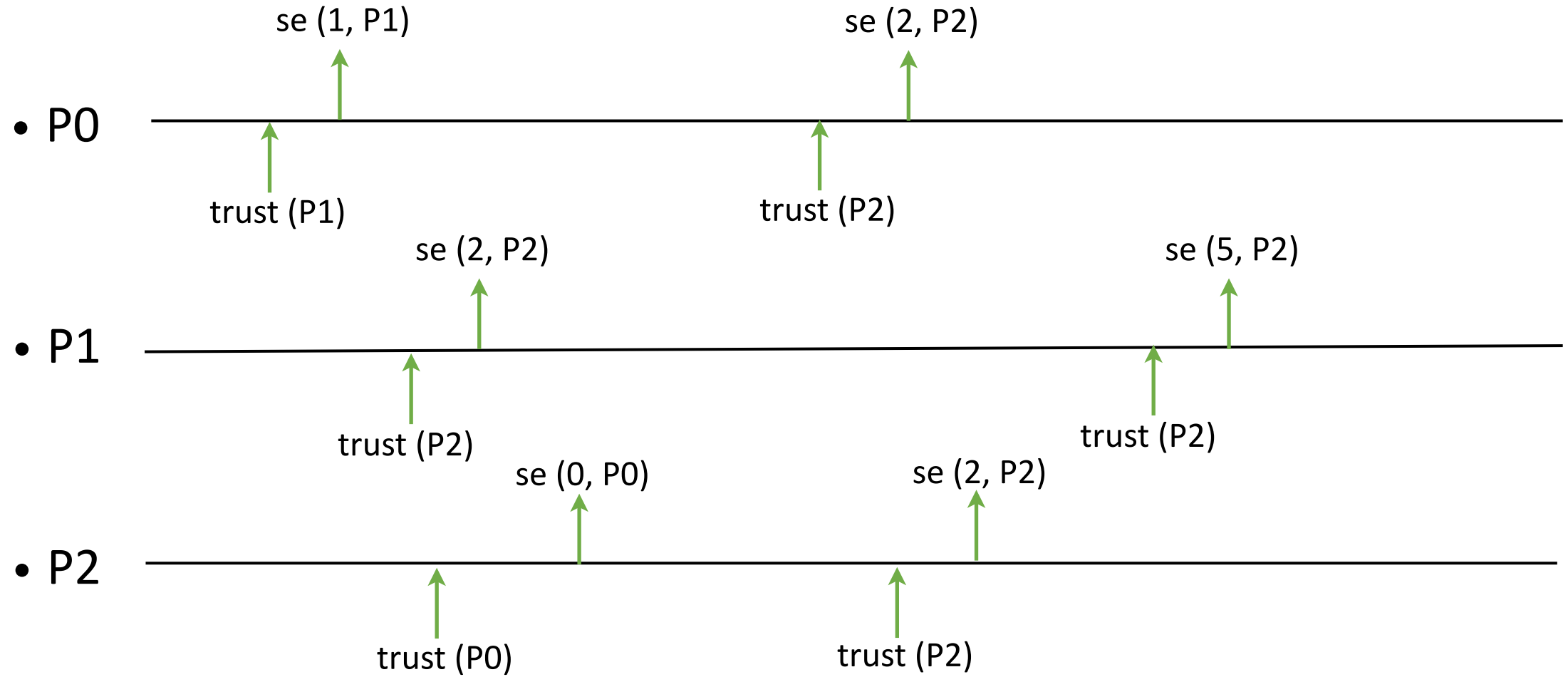- Therefore, the timestamp domain is disjointly divided between processes.
- For a new leader, we jump to the next timestamp for that process.


```
   0     1              N     N+1            2N    2N+1

  P0    P1             P0     P1             P0    P1
```

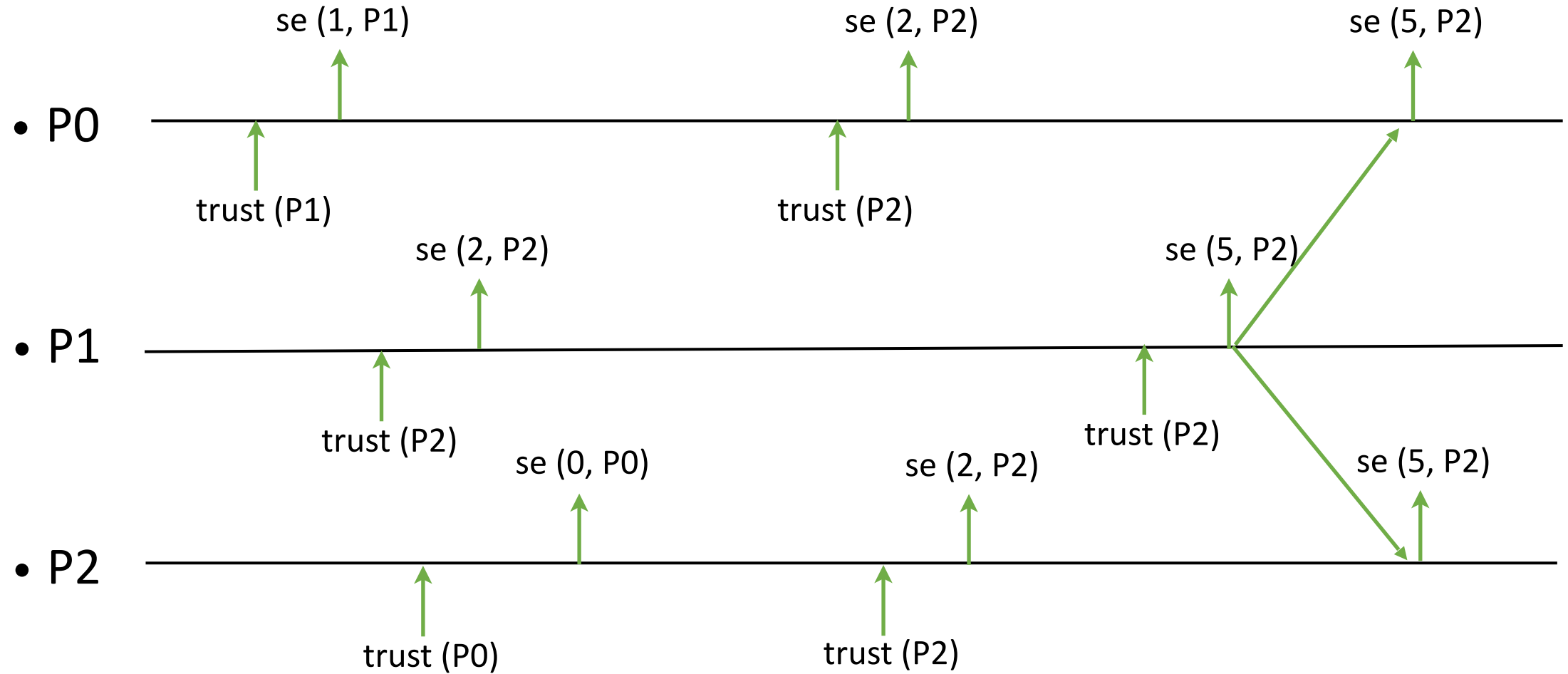# Epoch-change, Consistency

To have eventual leadership, the last leader should be installed with the *same timestamp* at all processes.

Epoch-change, Eventual Leadership

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.
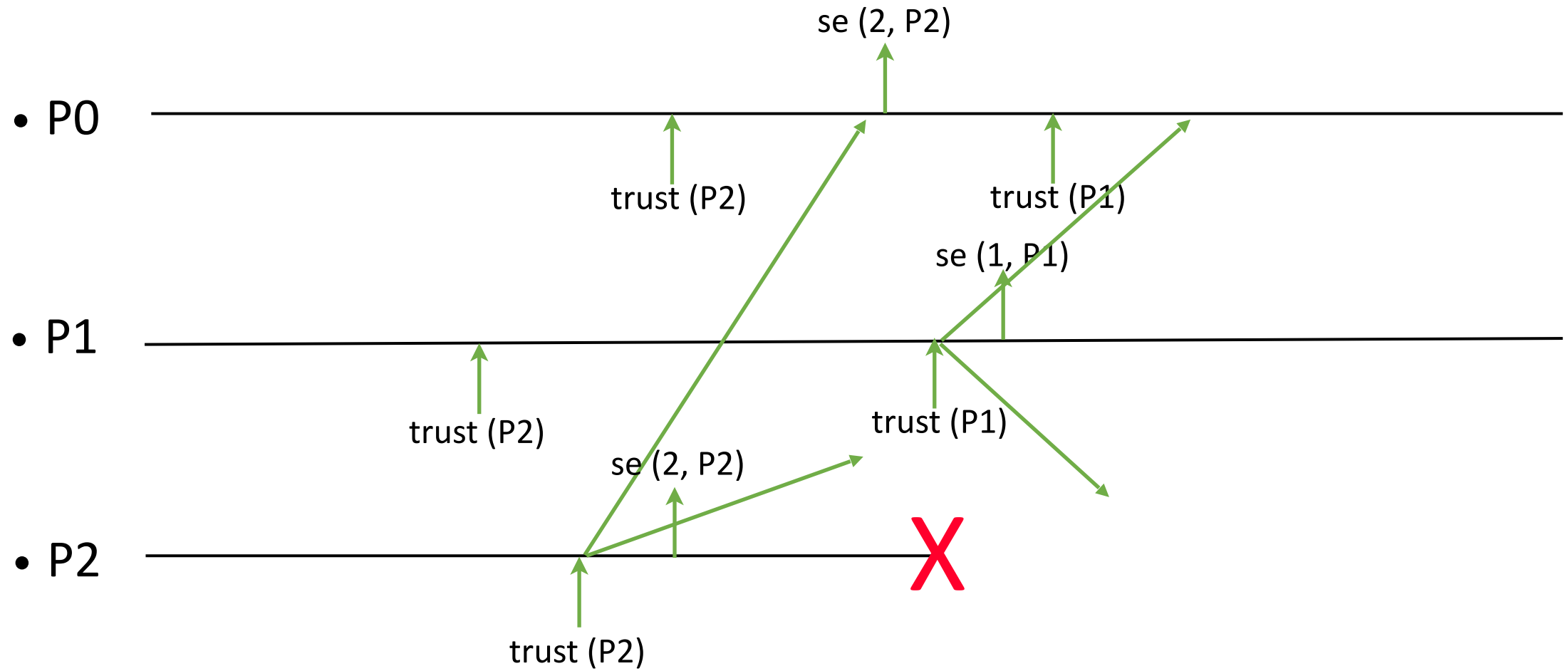
Epoch-change, Eventual Leadership

- 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.

Epoch-change, Eventual Leadership

Eventual leadership

Without eventual accuracy, eventual leadership is not guaranteed:
Every leader may be suspected infinitely often.

Implements:
    EpochChange, **instance** *ec*.
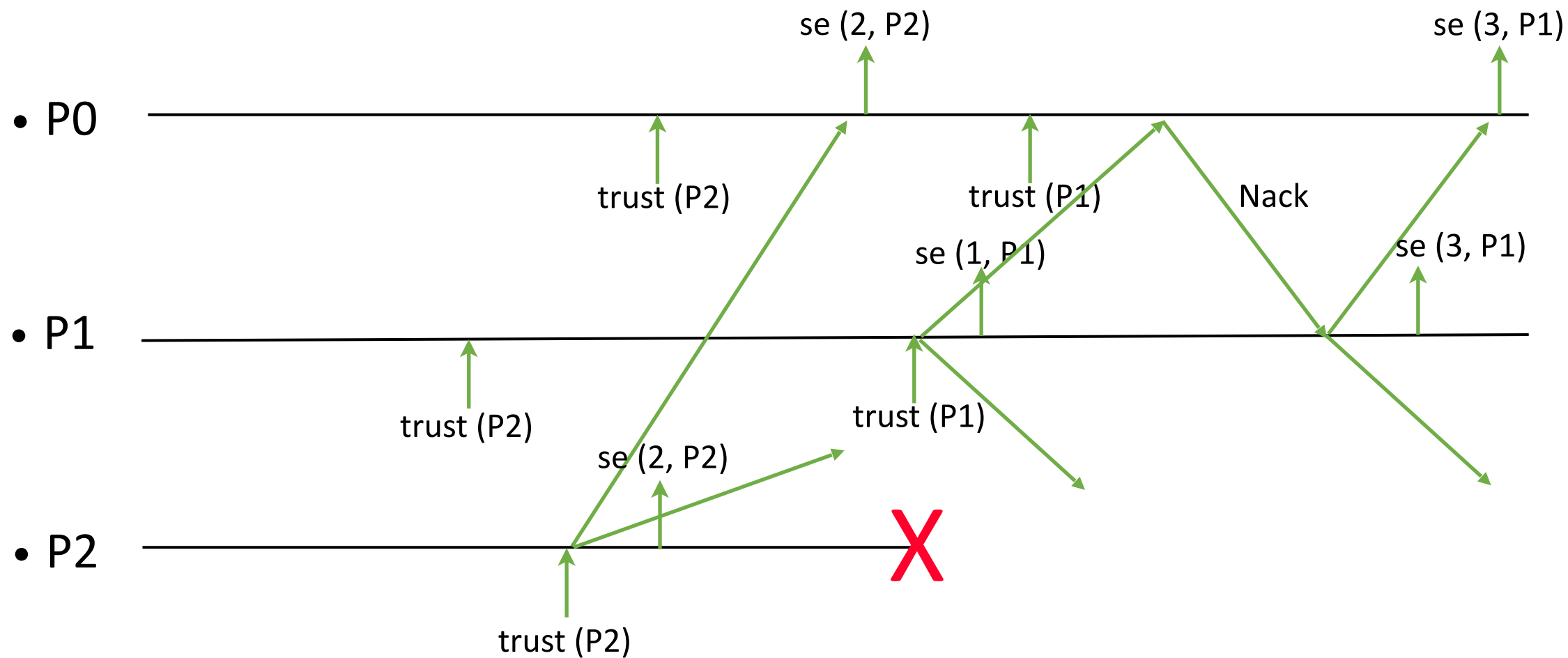
Uses:
    PerfectPointToPointLinks, **instance** *pl*;
    BestEffortBroadcast, **instance** *beb*;
    EventualLeaderDetector, **instance** Ω.

upon event $\langle$ *ec, Init* $\rangle$ **do**
    trusted := ℓ_0;
    lastts := 0;
    ts := *rank(self)*;

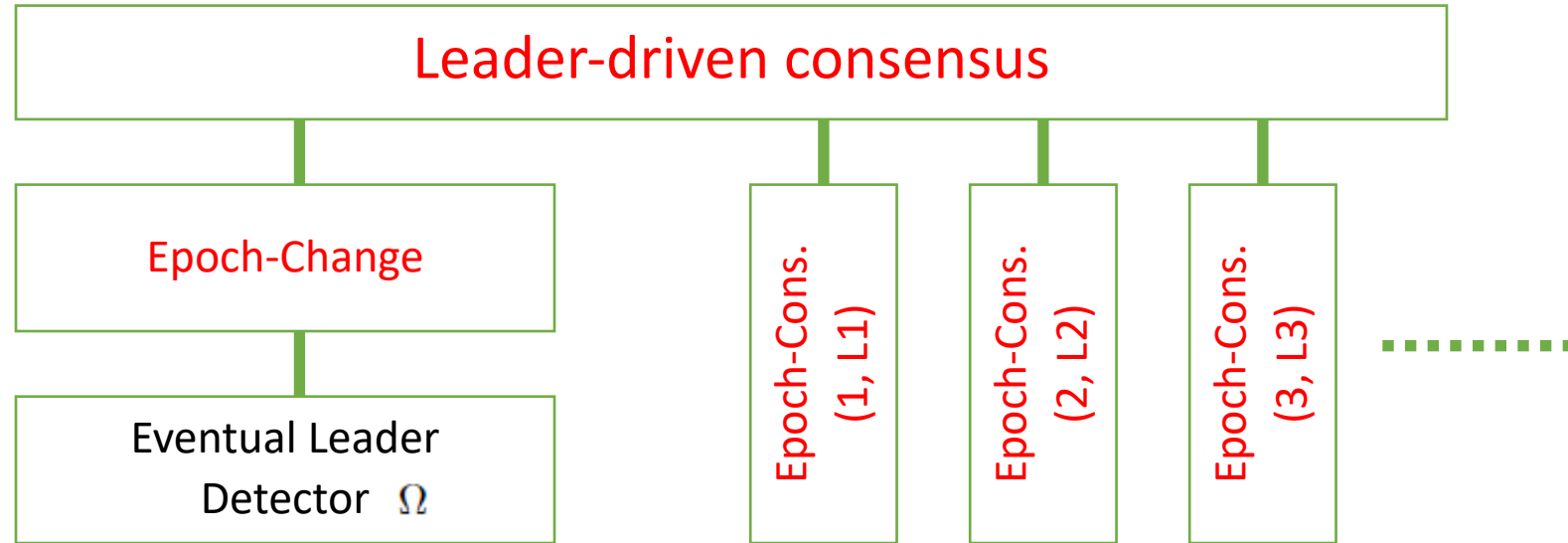trusted is the latest leader that Ω suggested.
lastts is the last timestamp that was accepted for the leader.
ts is the timestamp that this process wants to impose on others (when Ω suggests her as the leader.)

upon event $\langle\ \Omega,\ Trust\ |\ p\ \rangle$ **do**
    $trusted := p;$
    if $p = self$ **then**
        $ts := (lastts/N + 1)*N + rank(self)$
        trigger $\langle\ beb,\ Broadcast\ |\ [\text{NEWEPOCH}, ts]\ \rangle;$

upon event $\langle\ beb,\ Deliver\ |\ \ell,\ [\text{NEWEPOCH}, newts]\ \rangle$ **do** **where** $newts \neq lastts$
    if $\ell = trusted \land newts > lastts$ **then**
        $lastts := newts;$
        trigger $\langle\ ec,\ StartEpoch\ |\ newts, \ell\ \rangle;$
    else
        trigger $\langle\ pl,\ Send\ |\ \ell,\ [\text{NACK}]\ \rangle;$

upon event $\langle\ pl,\ Deliver\ |\ p,\ [\text{NACK}]\ \rangle$ **do**
    if $trusted = self$ **then**
        $ts := ts + N;$
        trigger $\langle\ beb,\ Broadcast\ |\ [\text{NEWEPOCH}, ts]\ \rangle;$

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,
  - It retrieves the value z and and ep-decides(z)

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 incorrect

Leader-driven consensus

Implements uc
uses ec, ep (multiple instances)

upon <init> **do**
       val := $\perp$; proposed := false; decided := false
       (ets, L) := (0, P0); (newts, newL) := (0, $\perp$)
       Initialize Epoch Consensus instance 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 instance 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 $\land$ val $\neq\bot$ $\land$ ¬proposed **do**

    proposed := true

    trigger <ep[ets], propose(val)>

upon <ep[ets], decide(v)> **do**

    if ¬decided **then**

        decided := true

        trigger <decide(v)>

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)
  - From EC3 (eventual leadership), EP5 (termination) and the algorithm (no redundant aborts)

- Validity (UC2)
  - From EP1 (validity) and the algorithm (leaders store and propose user proposals).

- Integrity (UC3)
  - The decided variable in the algorithm.

- Uniform Agreement (UC4)
  - From the algorithm (decide only after decide from ep), EP2 (agreement), and EP4 (lock-in)

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

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