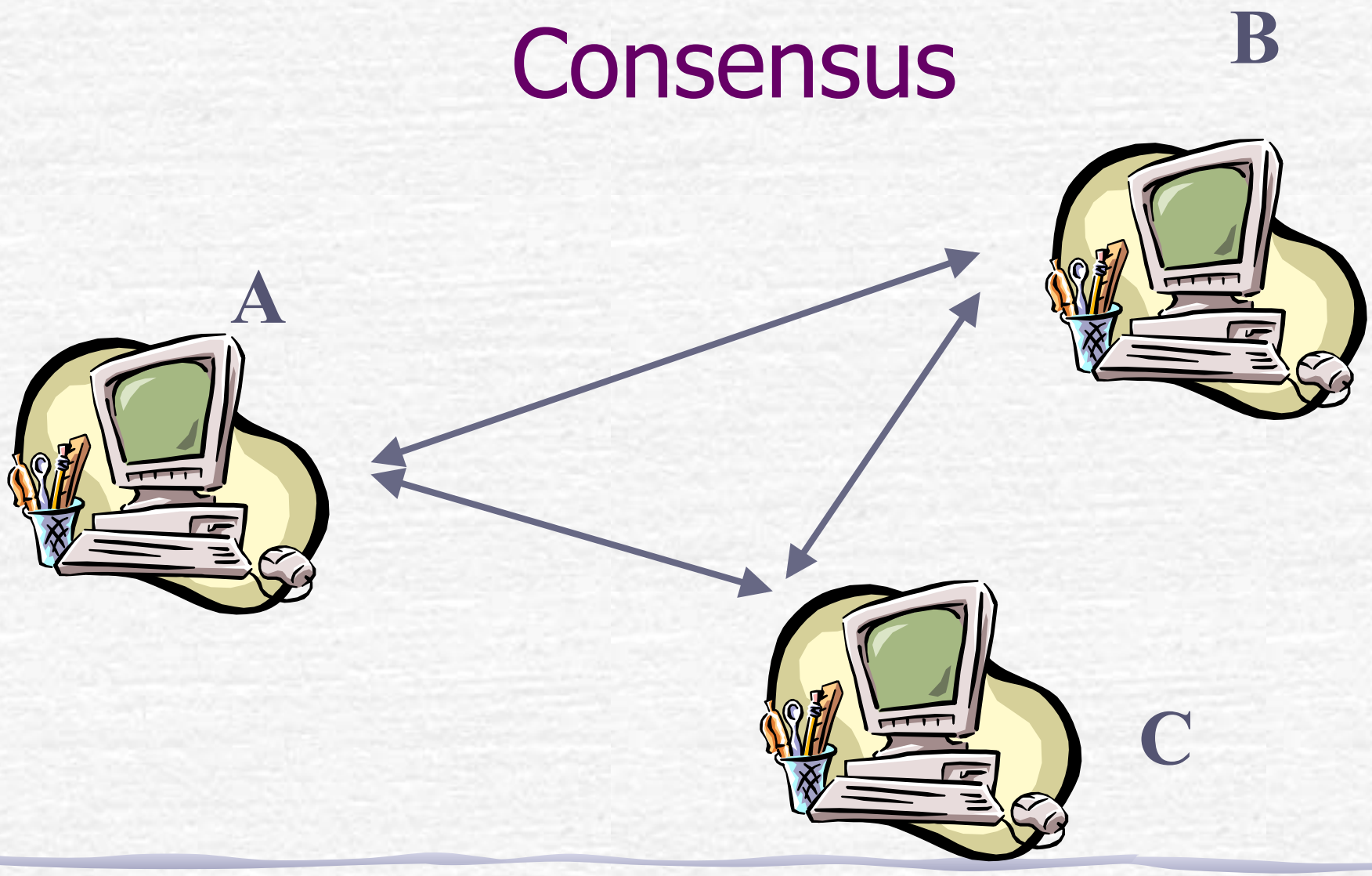
Distributed systems

Consensus

Prof R. Guerraoui
Distributed Programming Laboratory

Consensus



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)

Consensus

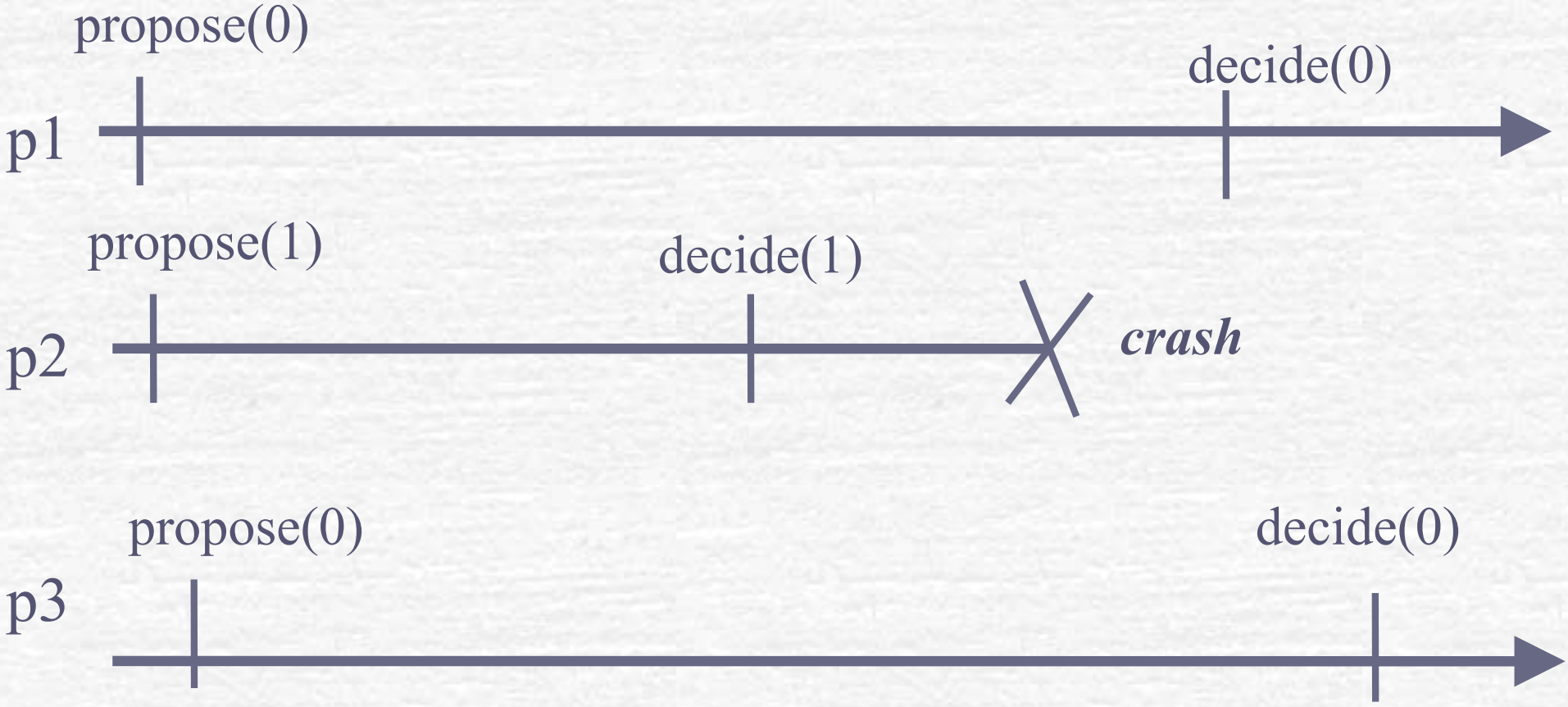
C1. Validity: Any value decided is a value proposed

C2. Agreement: No two correct processes decide differently

C3. Termination: Every correct process eventually decides

C4. Integrity: No process decides twice

Consensus


propose(0)
decide(0)
p1
propose(1)
decide(1)
p2
crash
propose(0)
decide(0)
p3

Uniform consensus

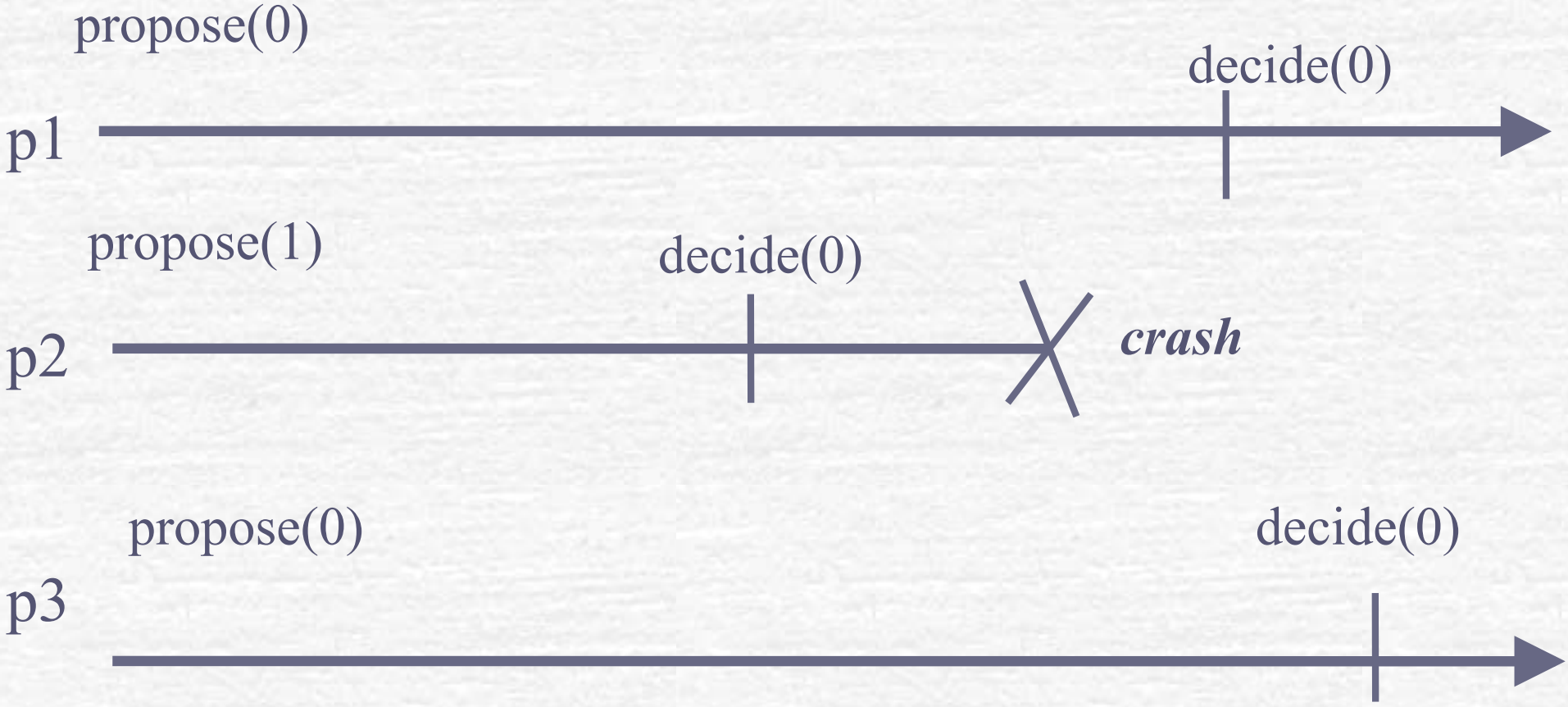
C1. Validity: Any value decided is a value proposed

C2′. Uniform Agreement: No two processes decide differently

C3. Termination: Every correct process eventually decides

C4. Integrity: No process decides twice

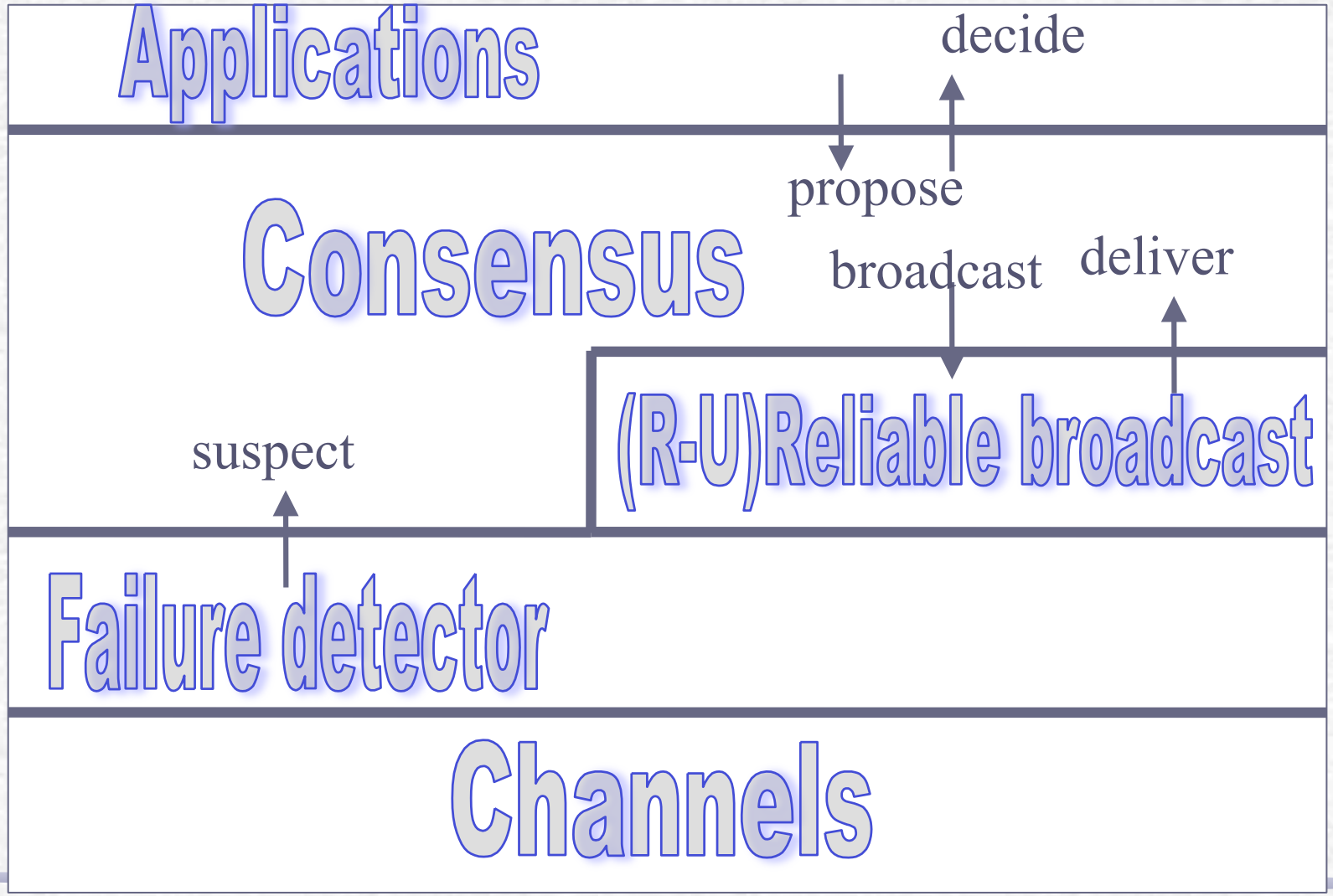
Uniform consensus



Consensus

- ***Events***
  - Request: <Propose, v>
  - Indication: <Decide, v'>
- ***Properties:***
  - ***C1, C2, C3, C4***

Modules of a process



Consensus algorithm I

- A P-based (fail-stop) consensus algorithm

- The processes exchange and update proposals in rounds and decide on the value of the non-suspected process with the smallest id [Gue95]

Consensus algorithm II

- A P-based (i.e., fail-stop) uniform consensus algorithm
- The processes exchange and update proposal in rounds, and after n rounds decide on the current proposal value [Lyn96]

Consensus algorithm III

- A <>P-based uniform consensus algorithm assuming a correct majority
- The processes alternate in the role of a coordinator until one of them succeeds in imposing a decision [DLS,CT,L,LO]

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 (a) to deliver the proposal of the leader in that round to adopt it, or (b) to suspect the leader

Consensus algorithm I

- **Implements:** Consensus (cons).
- **Uses:**
  - BestEffortBroadcast (beb).
  - PerfectFailureDetector (P).
- **upon event** < Init > **do**
  - suspected := ∅;
  - round := 1; currentProposal := nil;
  - broadcast := delivered[] := false;

Consensus algorithm I

- **upon event** < crash, pi > **do**
  - suspected := suspected U {pi};

- **upon event** < Propose, v> **do**
  - **if** currentProposal = nil **then**
    - currentProposal := v;

Consensus algorithm I

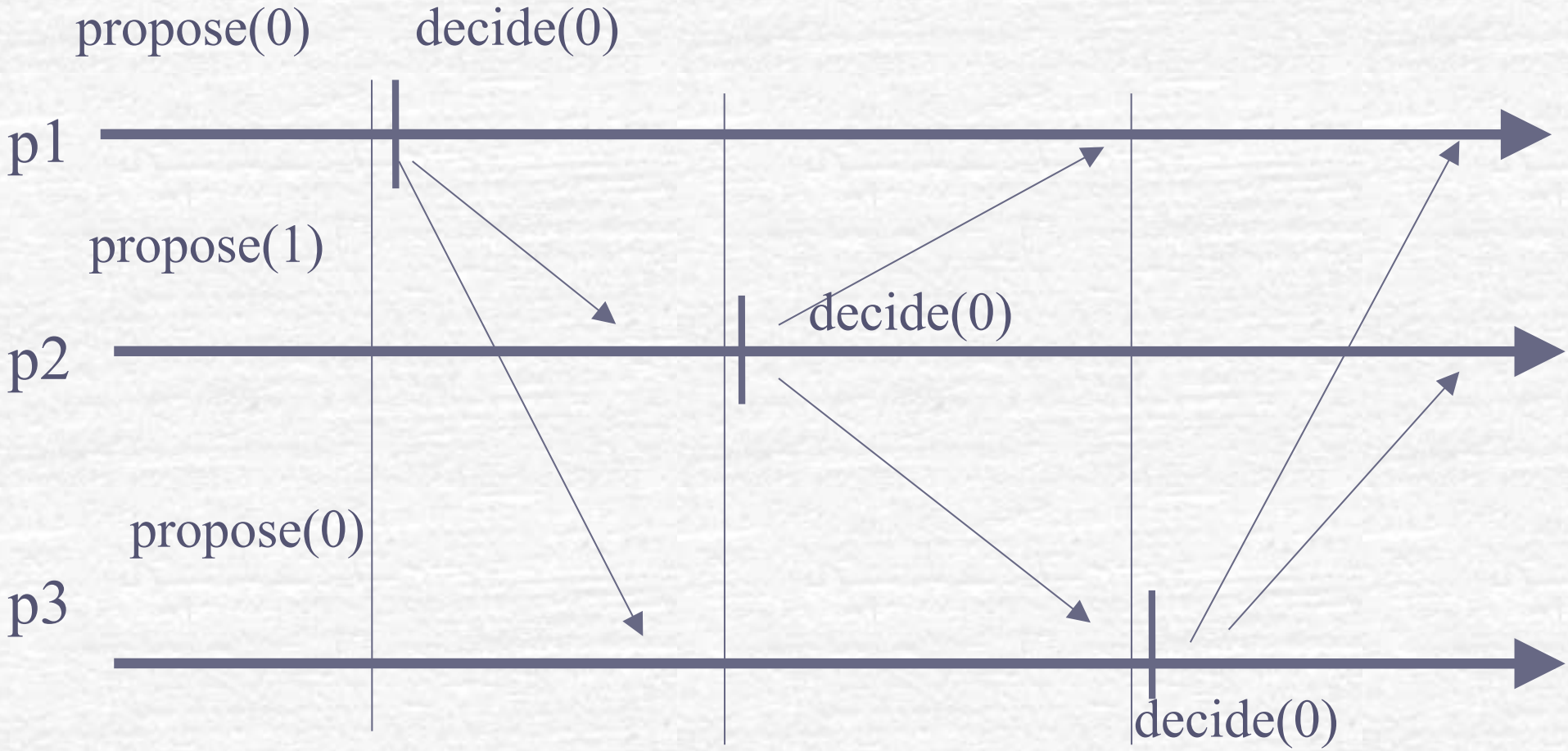
- **upon event** < bebDeliver, p_{round}, value > **do**
  - currentProposal := value;
  - delivered[round] := true;

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

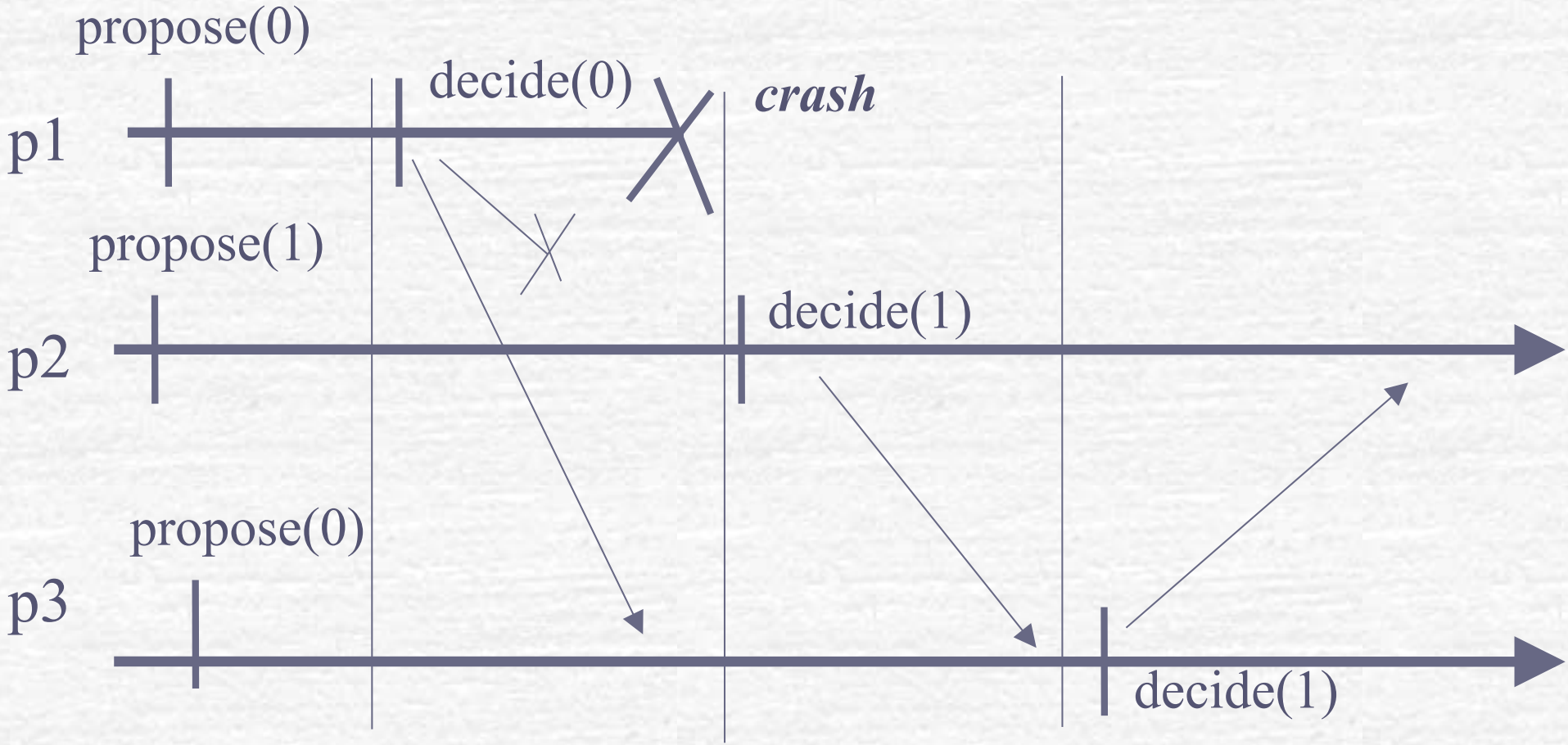
Consensus algorithm I

- **upon event** p_{round}=self **and** broadcast=false **and** currentProposal≠nil **do**
  - **trigger** <Decide, currentProposal>;
  - **trigger** <bebBroadcast, currentProposal>;
  - broadcast := true;

Consensus algorithm I



Consensus algorithm I



Correctness argument

- Let pi be the correct process with the smallest id in a run R.
- Assume pi decides v.
  - If i = n, then pn is the only correct process.
  - Otherwise, in round i, all correct processes receive v and will not decide anything different from v.

Consensus algorithm II

- Algorithm II implements uniform consensus
- The processes go through rounds incrementally (1 to n): in each round I, process pI sends its currentProposal to all.
- A process adopts any currentProposal it receives.
- Processes decide on their currentProposal 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; currentProposal := nil;
  - broadcast := delivered[] := false;
  - decided := false

Consensus algorithm II

- **upon event** < crash, pi > **do**
  - suspected := suspected U {pi};

- **upon event** < Propose, v> **do**

  if currentProposal = nil **then**

  currentProposal := v;

Consensus algorithm II

- **upon event** < bebDeliver, p_{round}, value > **do**
  - currentProposal := value;
  - delivered[round] := true;

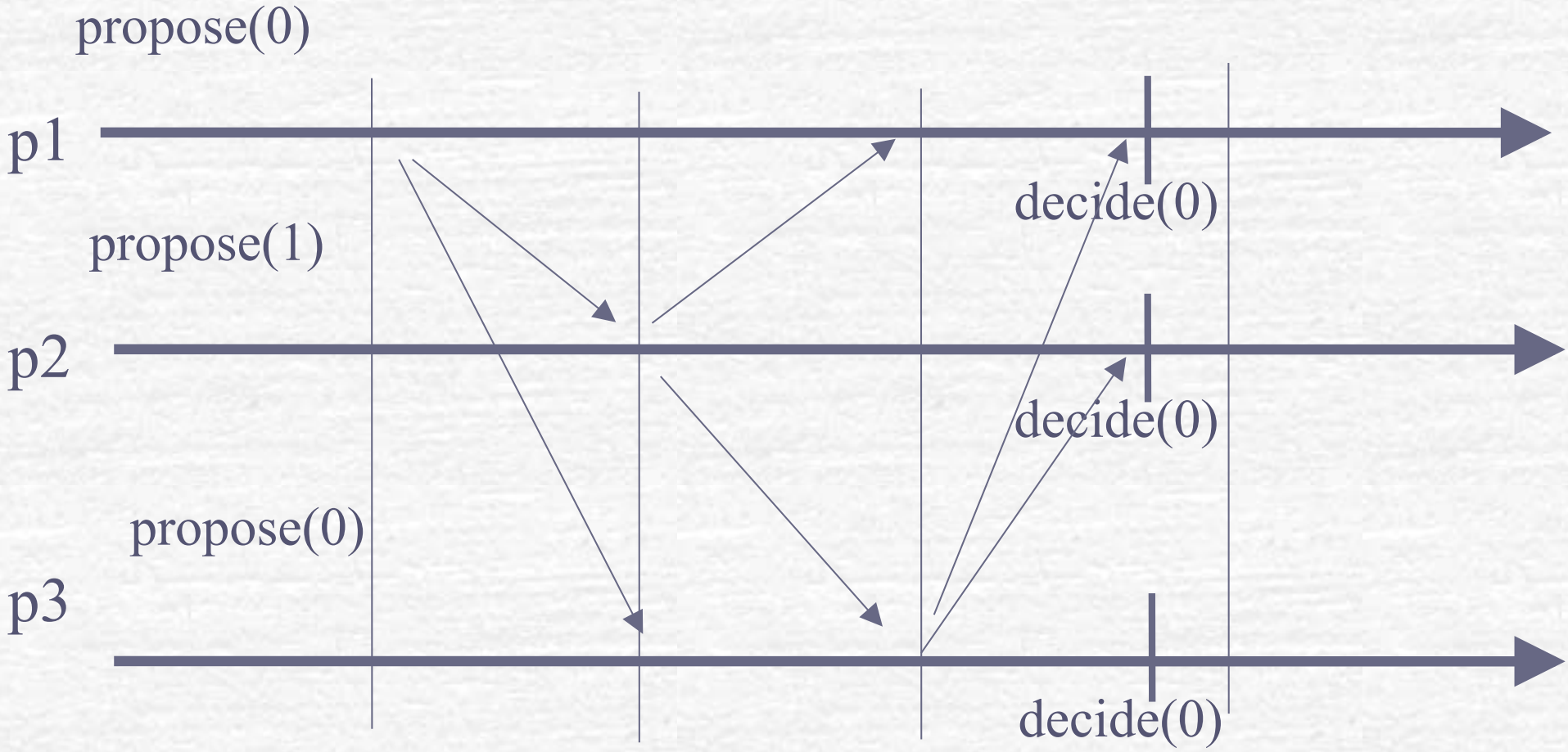
Consensus algorithm II

- **upon event** p_{round} = self **and** broadcast=false **and** currentProposal ≠ nil **do**
  - **trigger** <bebBroadcast, currentProposal>;
  - broadcast := true;

Consensus algorithm II

- **upon event** delivered[round] = true **or** $p_{round} \in$ suspected **do**
  - **if** round=n **and** decided=false **then**
    - **trigger** <Decide, currentProposal>
    - decided=true
  - **else**
    - round := round + 1

Consensus algorithm II



Correctness argument (A)

- **Lemma:** If a process pJ completes round I without receiving any message from pI and J > I, then pI crashes by the end of round J.
- Proof: Suppose pJ completes round I without receiving message from pI, J > I, and pI completes round J. Since pJ suspects pI in round I, pI has crashed before pJ completes round I. In round J either pI suspects pJ (not possible because pI crashes before pJ) or pI receives round J message from pJ (also not possible because pI crashes before pJ completes round I < J).

Correctness argument (B)

- **Uniform agreement:** Consider the process with the lowest id which decides, say pI. Thus, pI completes round n. By our previous lemma, in round I, every pJ with J > I receives the currentProposal of pI and adopts it. Thus, every process which sends a message after round I or decides, has the same currentProposal at the end of round I.

Perfect failure detection

- P ensures:
  - ***Strong completeness:*** eventually every process that crashes is permanently suspected by all correct processes
  - ***Strong accuracy:*** no correct process is suspected by any process

Perfect failure detection

Perfect failure detection

Consensus algorithm

- Without failure detector?

- System is ***asynchronous***: there is no information about whether processes are correct or not

Consensus impossibility (FLP)

- Consensus is impossible in an asynchronous system with at least one crash

Eventual failure detection

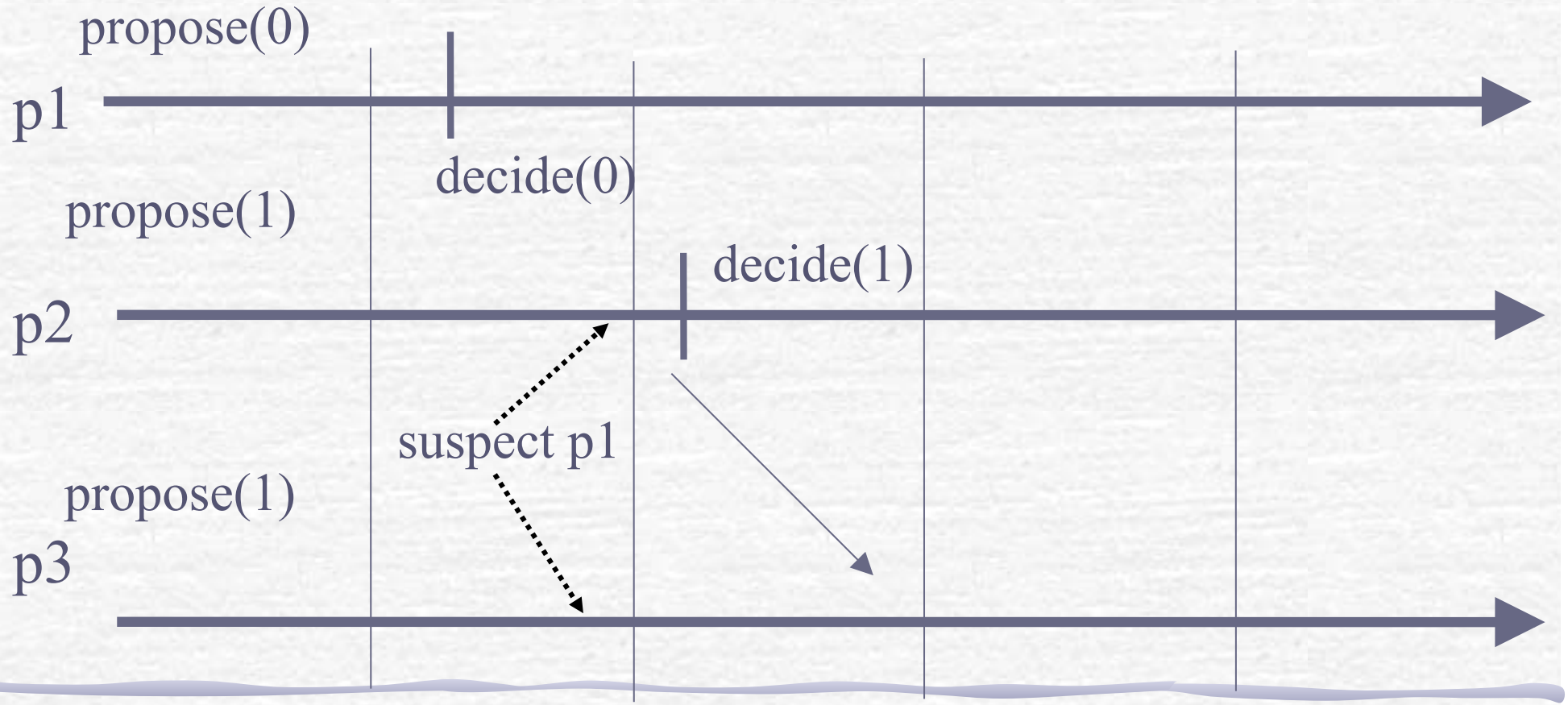
- <>P ensures:
  - ***Strong completeness:*** eventually every process that crashes is permanently suspected by all correct processes
  - ***Eventual strong accuracy:*** eventually no correct process is suspected by any process

"<>" makes a difference

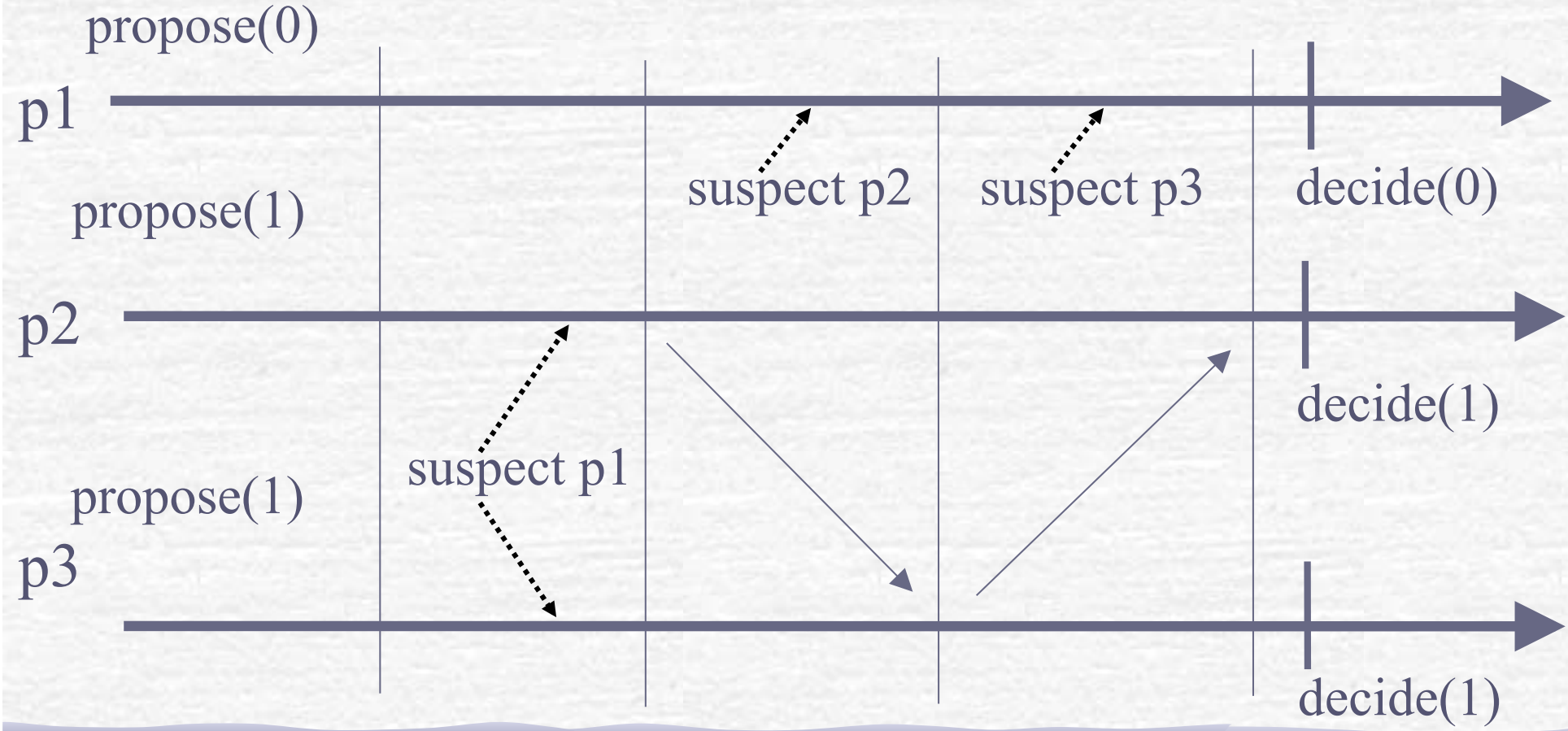
- ***Eventual strong accuracy:*** strong accuracy holds only *after finite time.*
- Correct processes may be *falsely suspected* a finite number of times.
- This breaks consensus algorithms I and II

  (see next slide)

Agreement violated with <>P in algorithm I

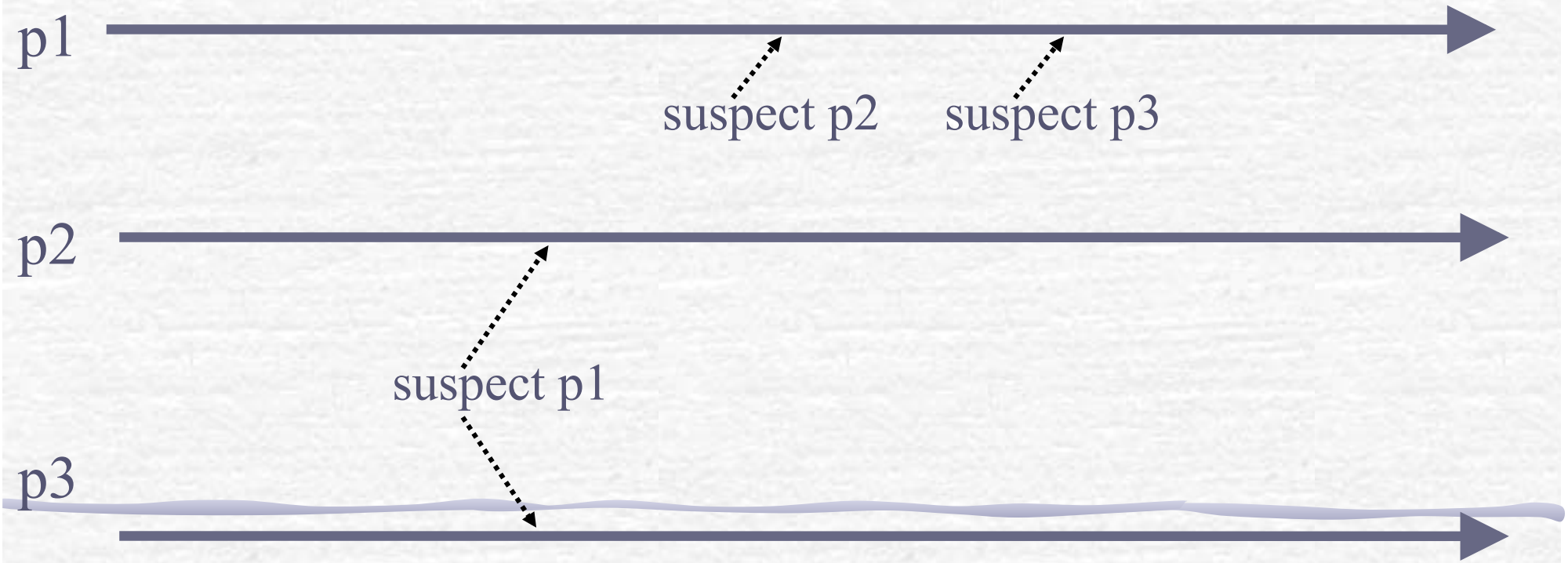


Agreement violated with <>P in algorithm II

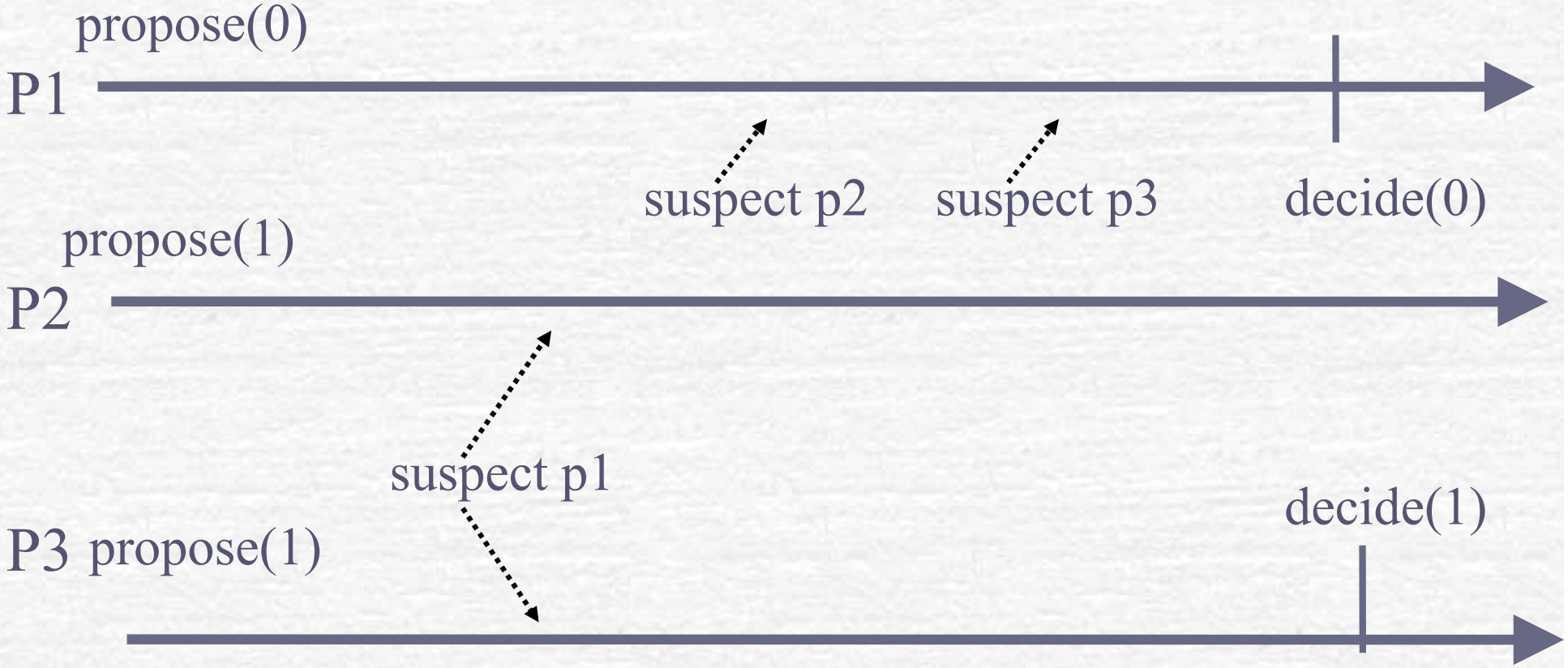


Eventual perfect failure detector

- <>P is an unreliable failure detector: no process p "knows" at any time, whether another process q has crashed or not



Assume 2 computers can crash



Consensus impossibility 2

- Consensus is impossible with an unreliable failure detector, e.g., <>P, if half of the processes can crash

Consensus algorithm III

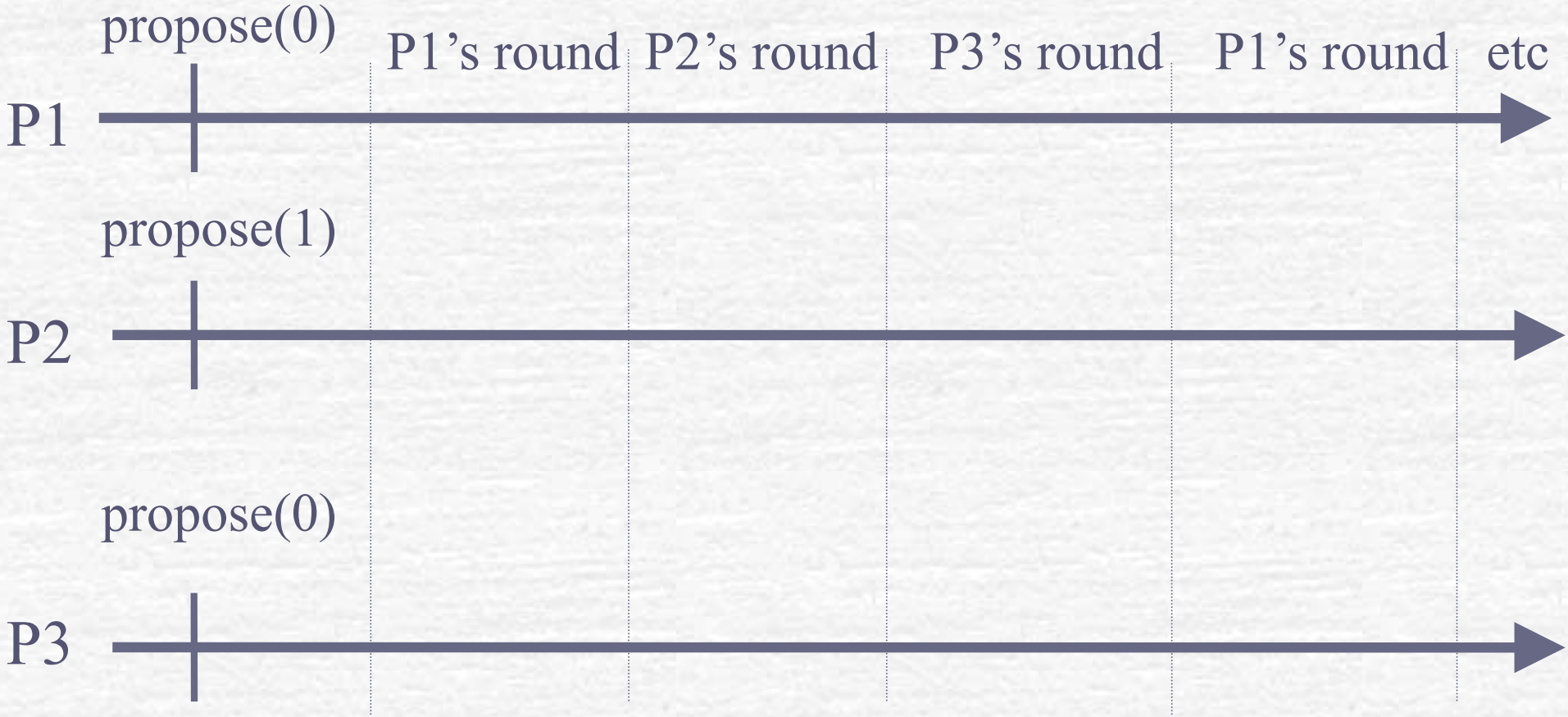
- A uniform consensus algorithm assuming:
  - a correct majority
  - a <>P failure detector

- Basic idea: the processes alternate in the role of a phase coordinator until one of them succeeds in imposing a decision

Consensus algorithm III

- The algorithm is also round-based: processes move incrementally from one round to the other
- Process pi is ***leader*** in every round k such that k mod n = I
- In such a round, pi ***tries to decide*** (next 2 slides)

Rotating leadership



When does a leader decide?

- Pi decides if it is not suspected

(those that suspect Pi inform Pi and move on)

- If Pi decides, Pi broadcasts the decision

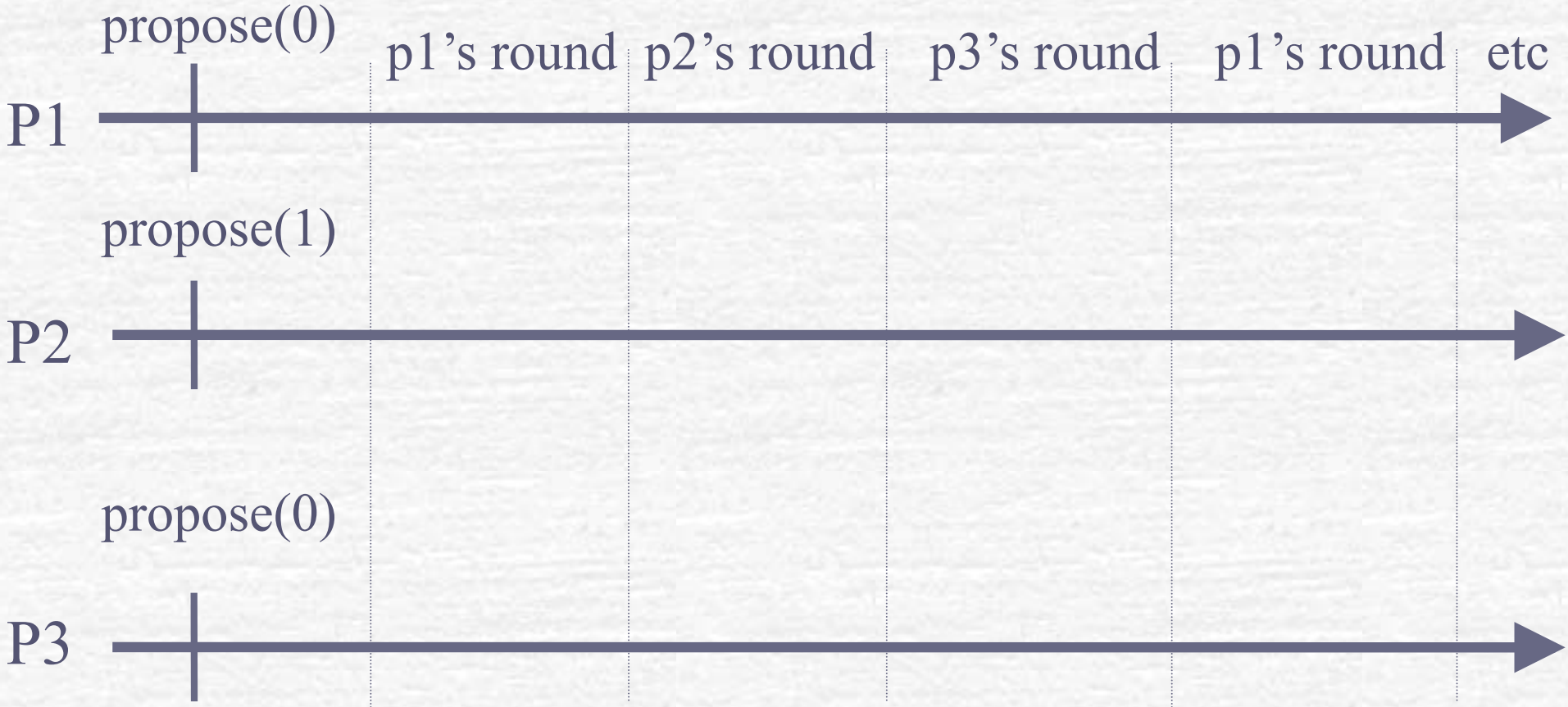
Consensus algorithm III

- pi succeeds if it is not suspected (processes that suspect pi inform pi and move to the next round; pi does so as well)
- If pi succeeds, pi uses a reliable broadcast to send the decision to all

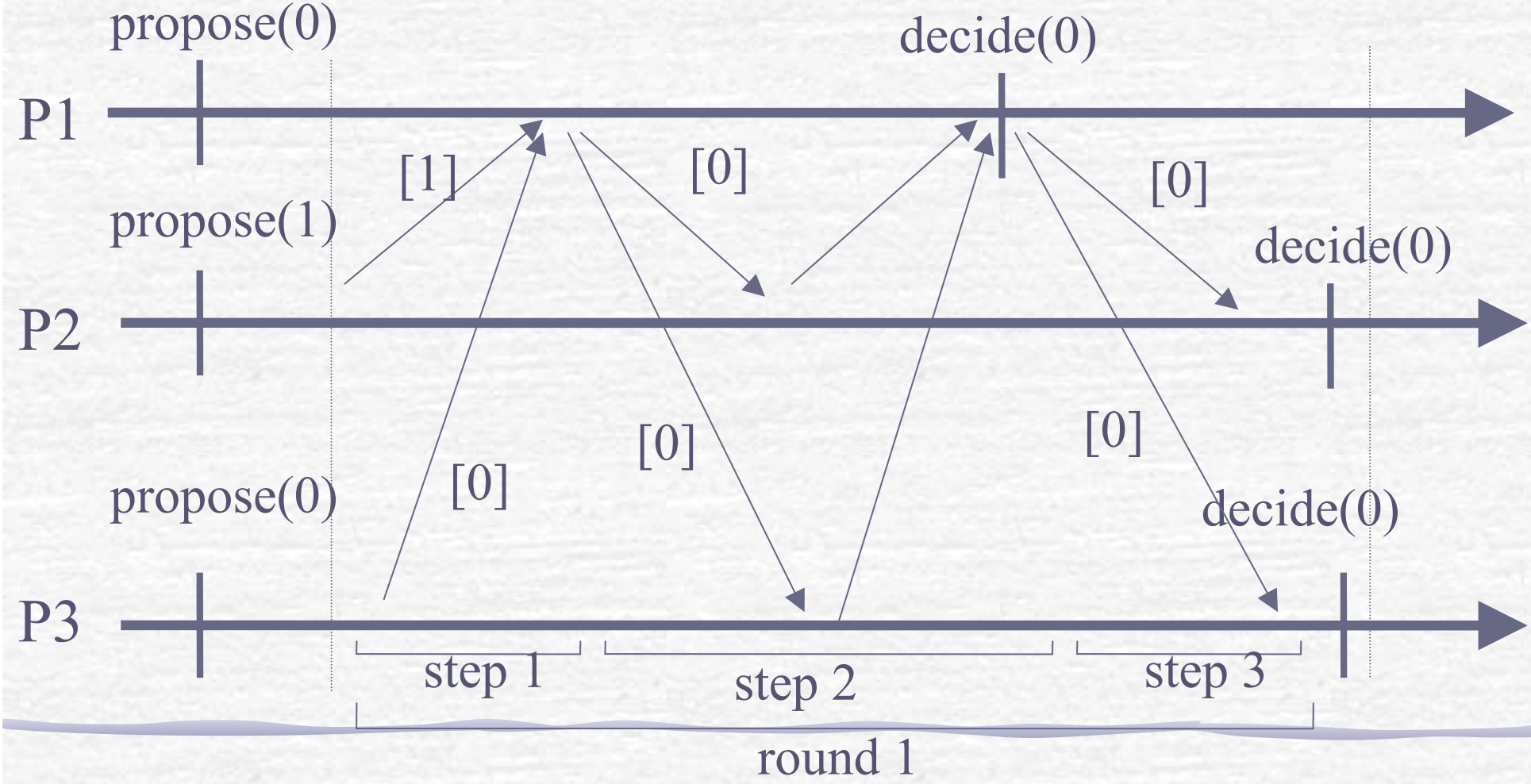
Consensus algorithm III

- To decide, pi executes steps 1-2-3
  - 1. pi selects among a majority the latest adopted value (latest with respect to the round in which the value is adopted – see step 2)
  - 2. pi imposes that value to a majority: any process in that majority adopts that value – pi fails if it is suspected
  - 3. pi decides and broadcasts the decision

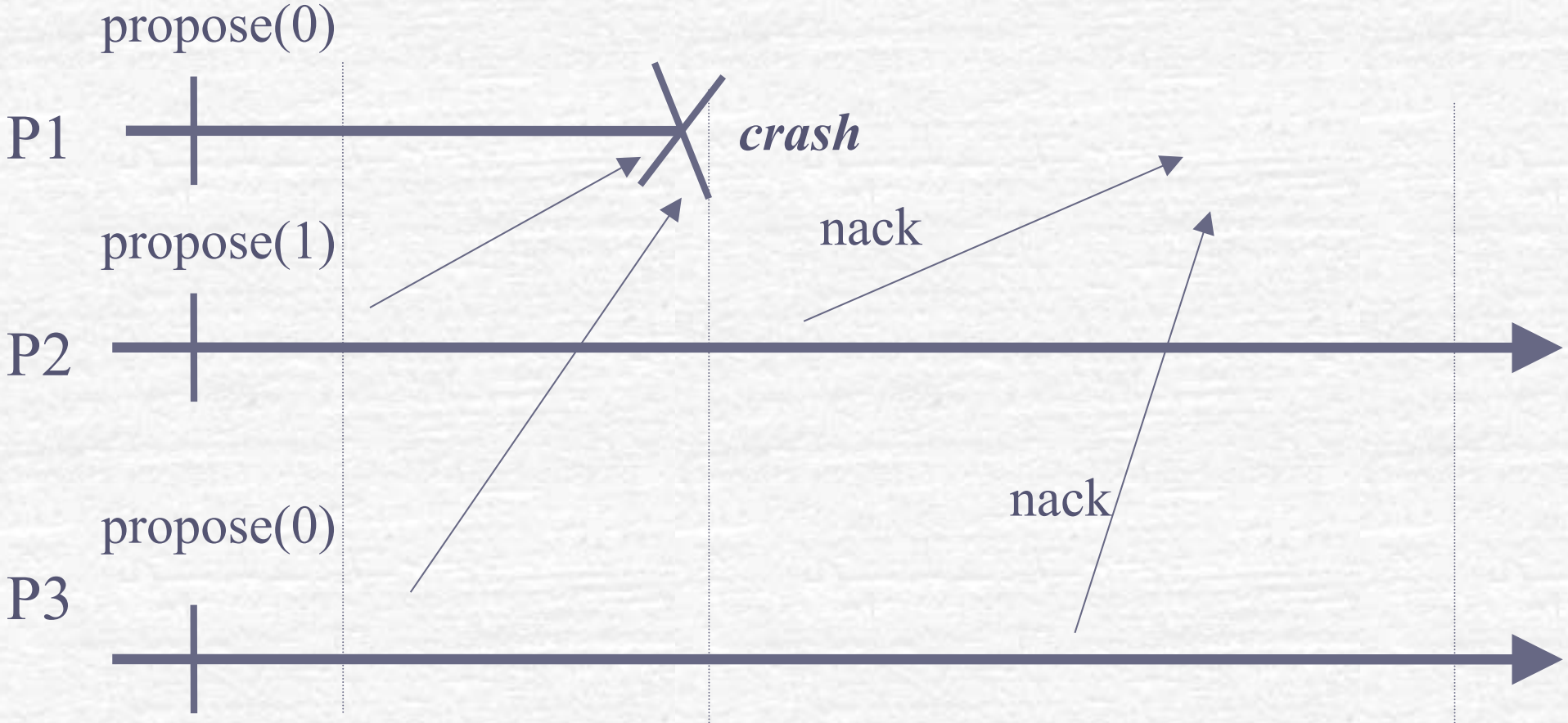
Consensus algorithm III



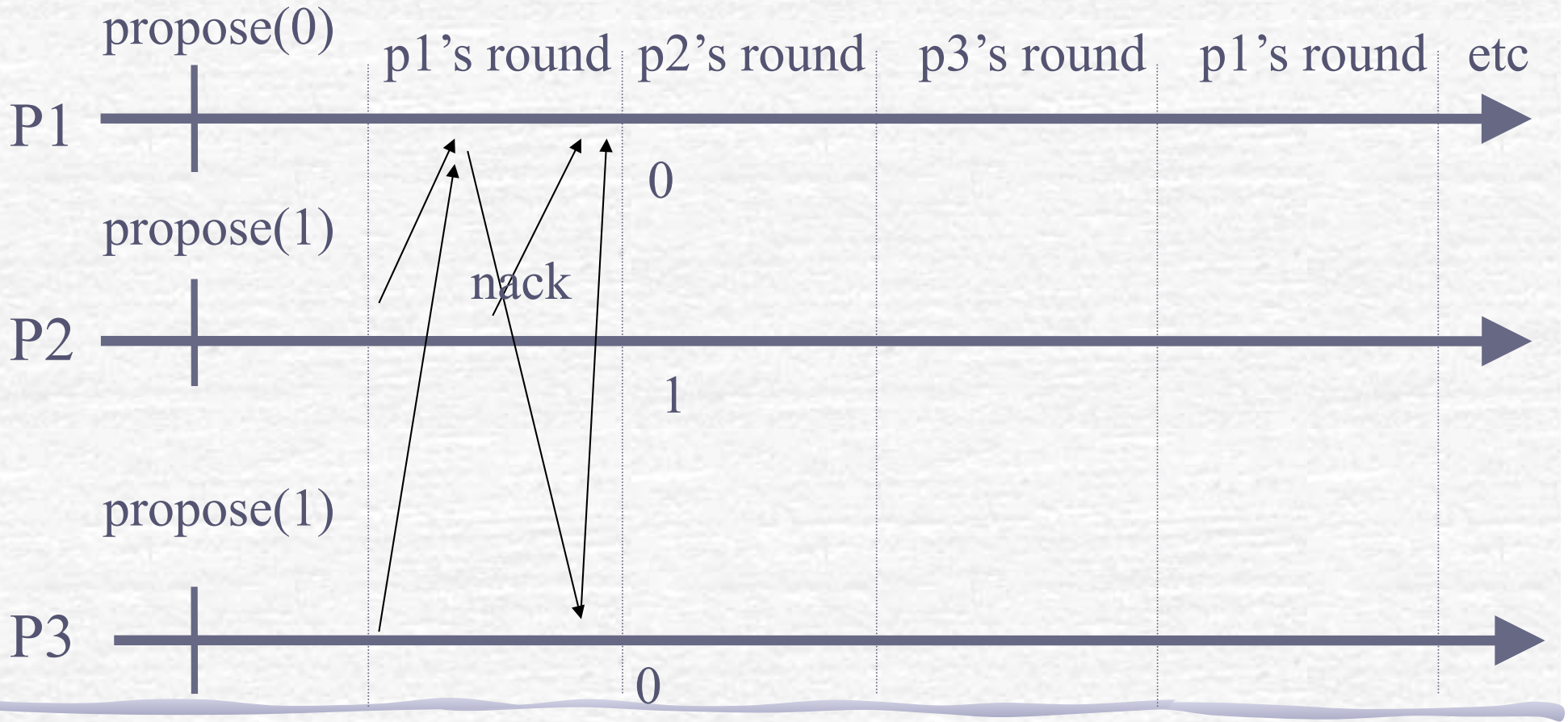
Consensus algorithm III



Consensus algorithm III



Consensus algorithm III



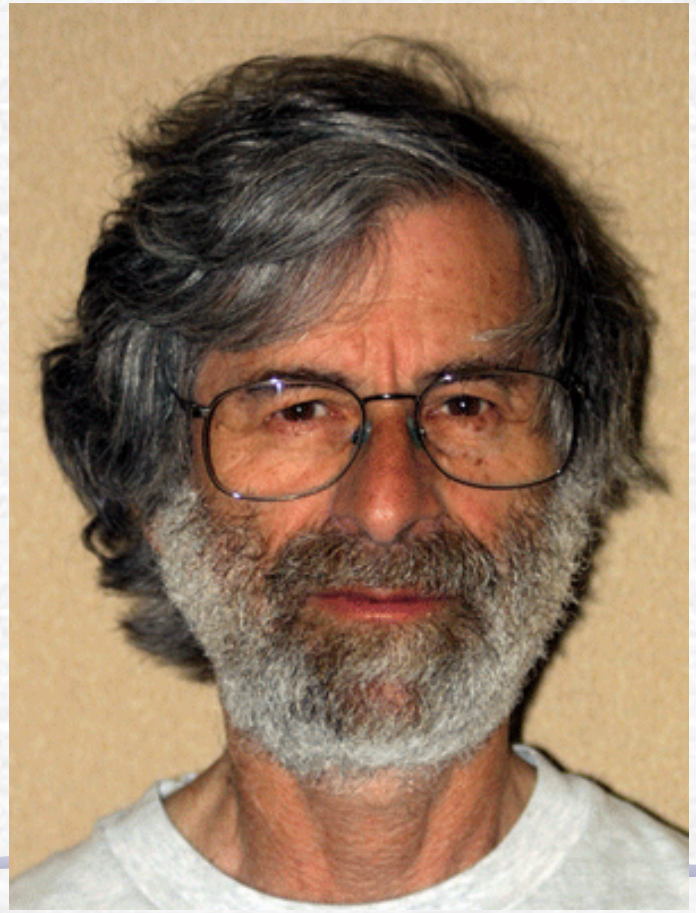
Consensus algorithm III

- Why reliable broadcast?

- Why we need to adopt a value from previous round?

DLS-L-CT-LO

- Consensus can be solved with eventual synchrony and a majority of correct computers



Correctness argument A

- ***Validity*** and ***integrity*** are trivial
- Consider ***termination***: if a correct process decides, it uses reliable broadcast to send the decision to all: every correct process decides
- Assume by contradiction that some process is correct and no correct process decides. We argue that this is impossible.

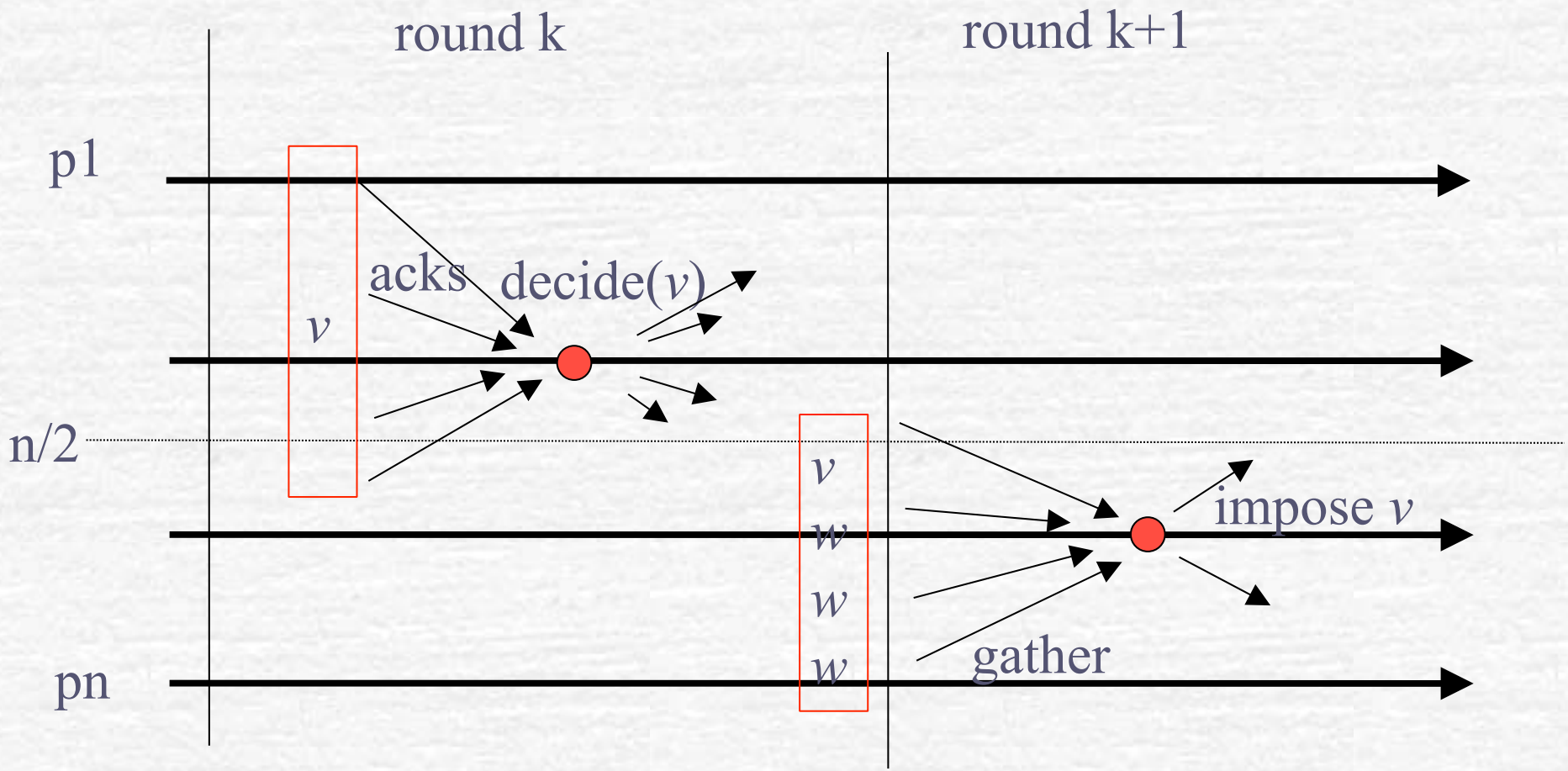
Correctness argument A’

- By the correct ***majority*** assumption and the ***completeness*** property of the failure detector, no correct process remains blocked forever in some phase.
- By the ***accuracy*** property of the failure detector, some correct process reaches a phase where it is leader and it is not suspected and reaches a decision in that phase: a contradiction

Correctness argument B

- Consider now ***agreement***
- Let k be the first round in which some process pi decides some value v, i.e., pi is the leader of round k and pi decides v in k
- This means that, in round k, a majority of processes have adopted v
- By the algorithm, no value else than v will be proposed (and hence decided) by any process in a round higher than k

Correctness argument B



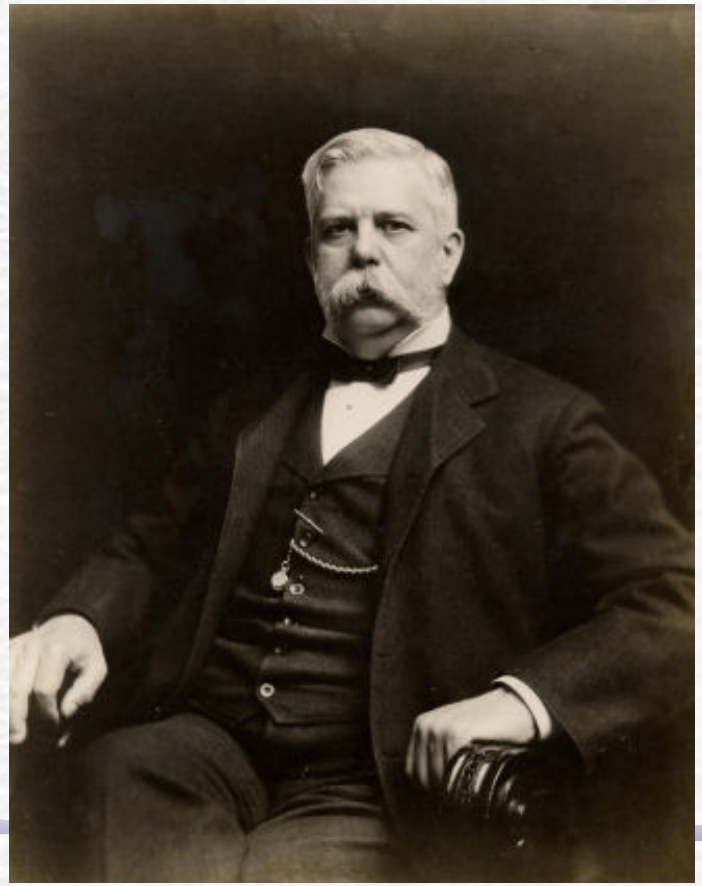
● = leader of that round

Agreement is never violated

- Look at "totally unreliable" failure detector (provides no guarantees)
  - may always suspect everybody
  - may never suspect anybody
- Agreement is never violated
  - Can use the same correctness argument as before
  - Termination not ensured (everybody may be suspected infinitely often)

Westinghouse 1800

Direct v Reverse air-brake system



Summary

- (Uniform) Consensus problem is an important problem to maintain consistency

- Three algorithms:
  - I: consensus using P
  - II: uniform consensus using P
  - III: uniform consensus using <>P and a correct majority