



Distributed systems

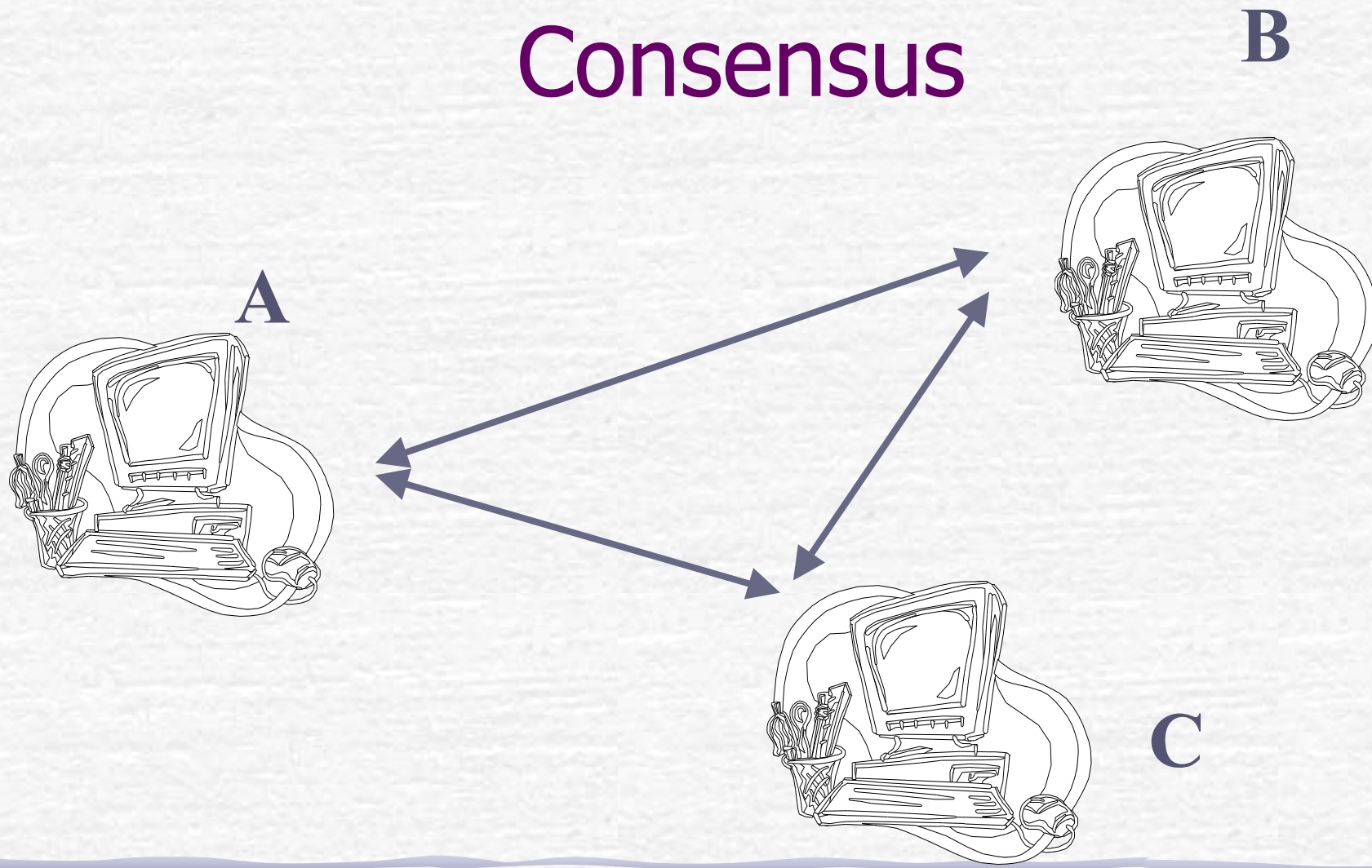
Consensus

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

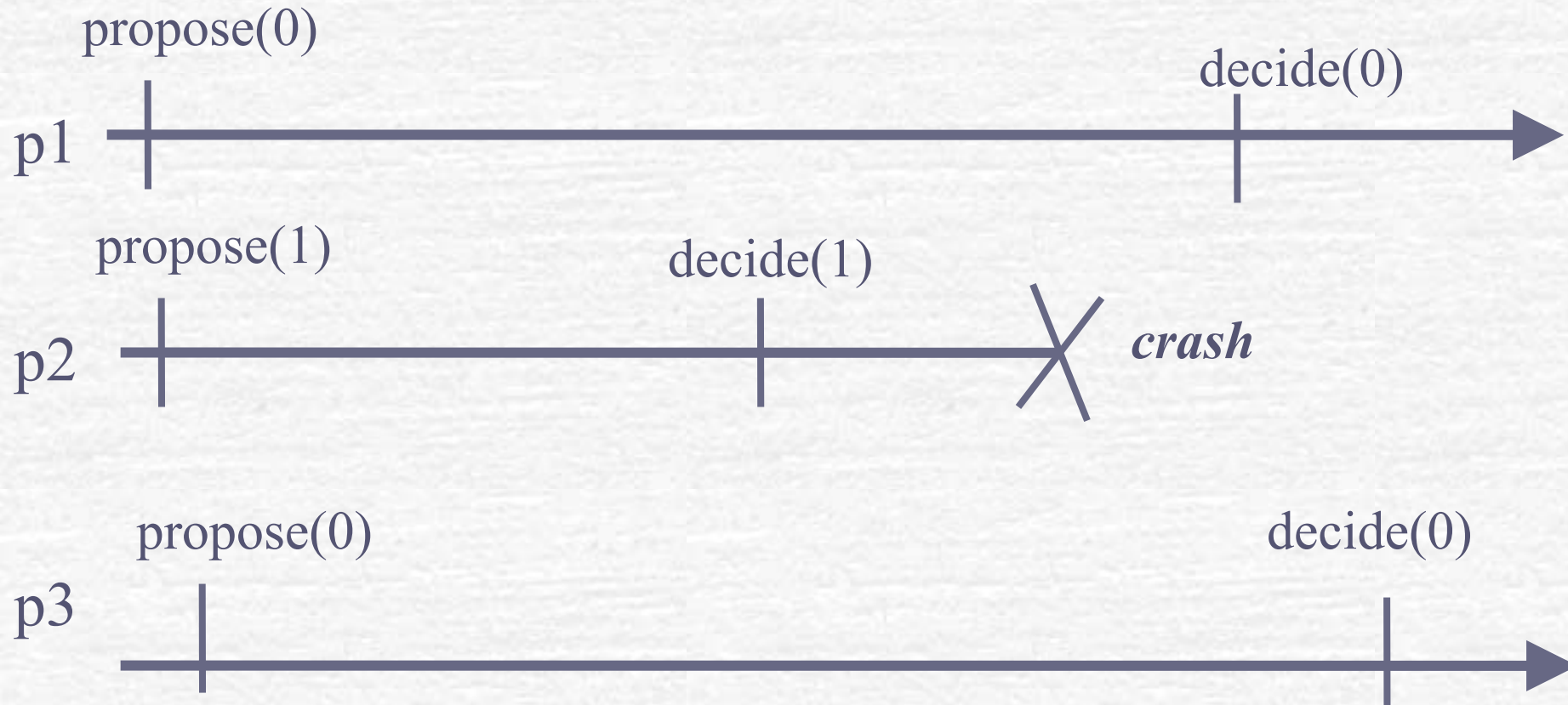
C1. Validity: Any value decided is a value proposed

C2. Agreement: No two correct processes decide differently

C3. Termination: Every correct process eventually decides

C4. Integrity: No process decides twice

Consensus



Uniform consensus

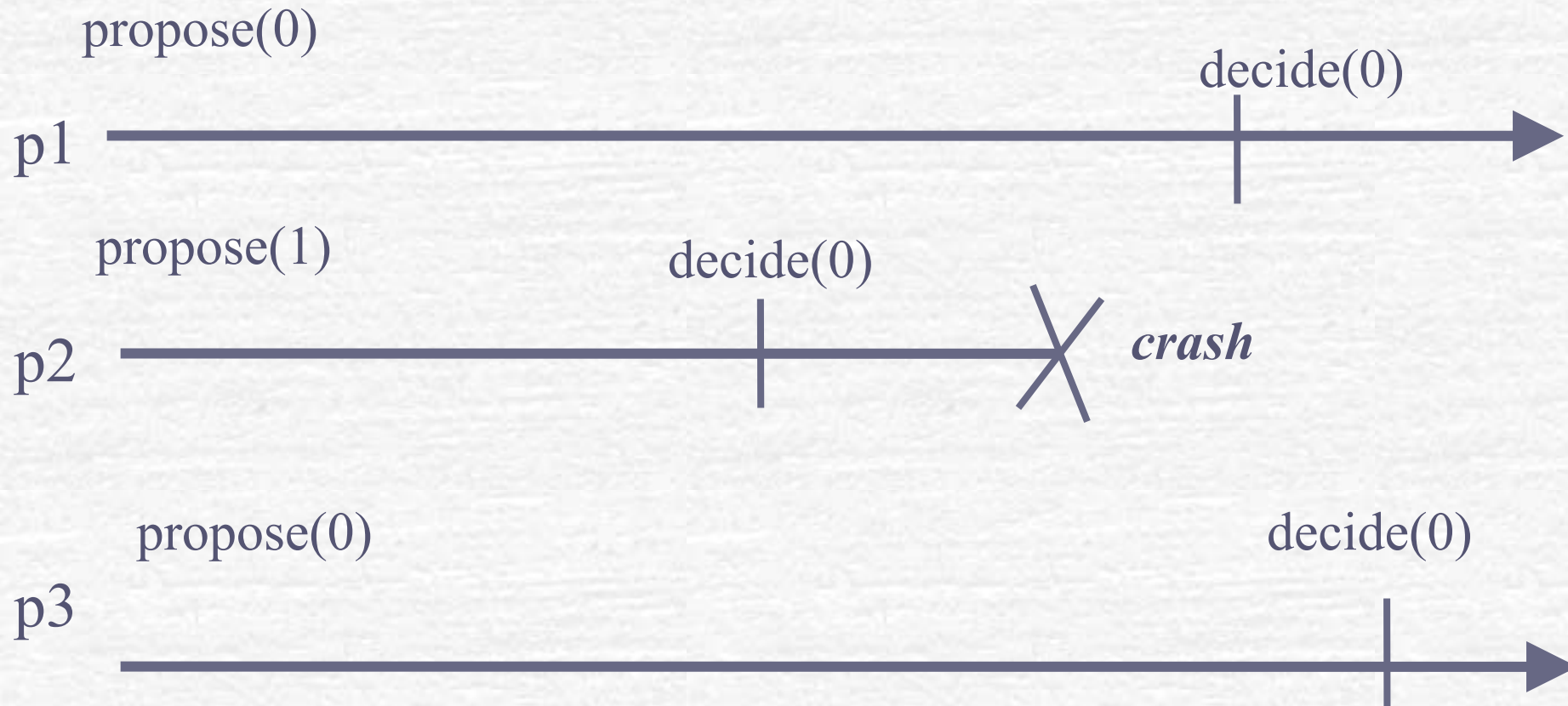
C1. Validity: Any value decided is a value proposed

C2'. Uniform Agreement: No two processes decide differently

C3. Termination: Every correct process eventually decides

C4. Integrity: No process decides twice

Uniform consensus



Consensus

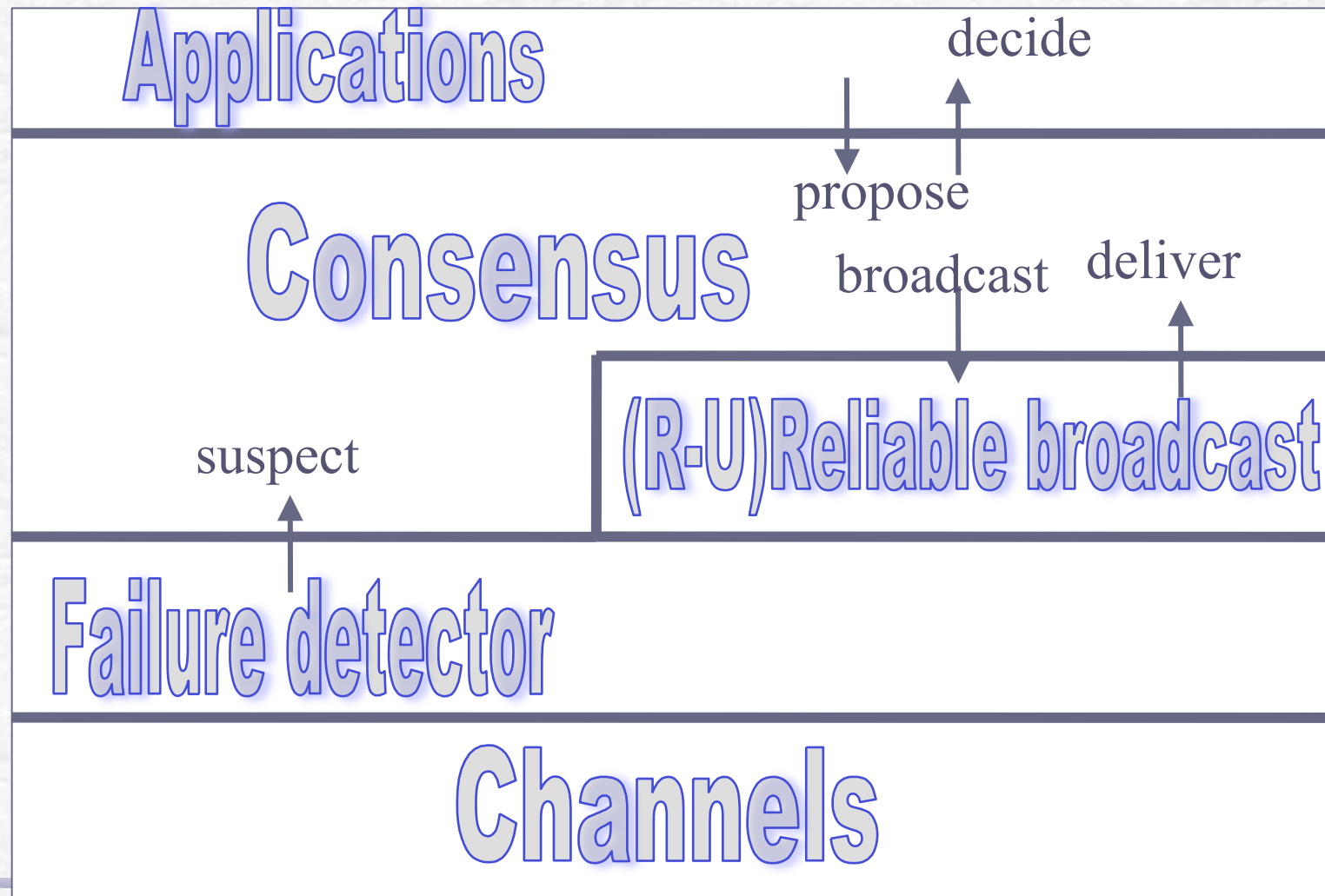
• ***Events***

- Request: $\langle \text{Propose}, v \rangle$
- Indication: $\langle \text{Decide}, v' \rangle$

• ***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

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 [DLS88,CT91]

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 := $\emptyset$;
 - round := 1; currentProposal := nil;
 - broadcast := delivered[] := false;

Consensus algorithm I

- **upon event** $\langle \text{crash}, p_i \rangle$ **do**
 - $\text{suspected} := \text{suspected} \cup \{p_i\};$
- **upon event** $\langle \text{Propose}, v \rangle$ **do**
 - **if** $\text{currentProposal} = \text{nil}$ **then**
 - $\text{currentProposal} := v;$

Consensus algorithm I

☛ **upon event** $\langle \text{bebDeliver}, p_{\text{round}}, \text{value} \rangle$ **do**

☛ $\text{currentProposal} := \text{value};$

☛ $\text{delivered}[\text{round}] := \text{true};$

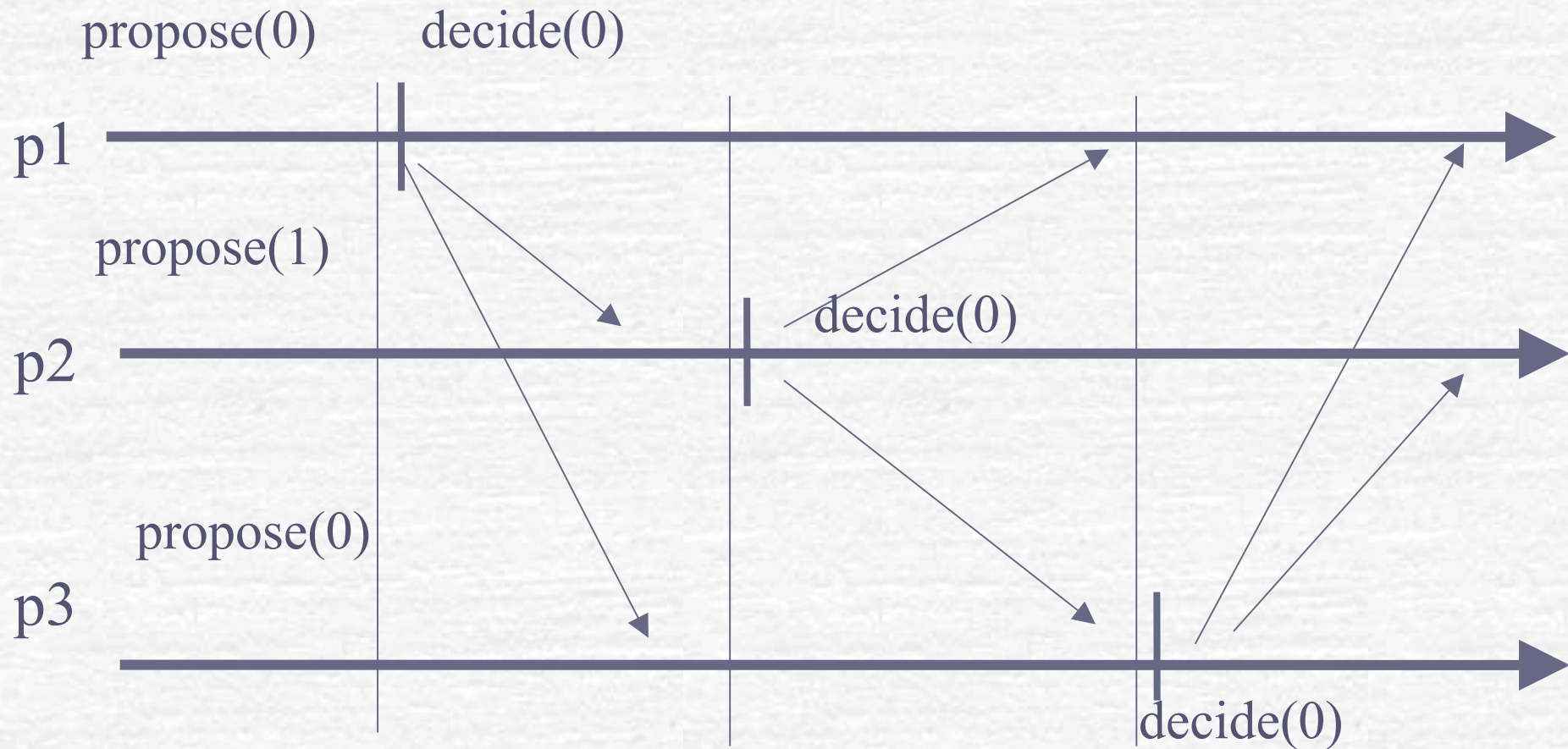
☛ **upon event** $\text{delivered}[\text{round}] = \text{true}$ **or**
 $p_{\text{round}} \in \text{suspected}$ **do**

☛ $\text{round} := \text{round} + 1;$

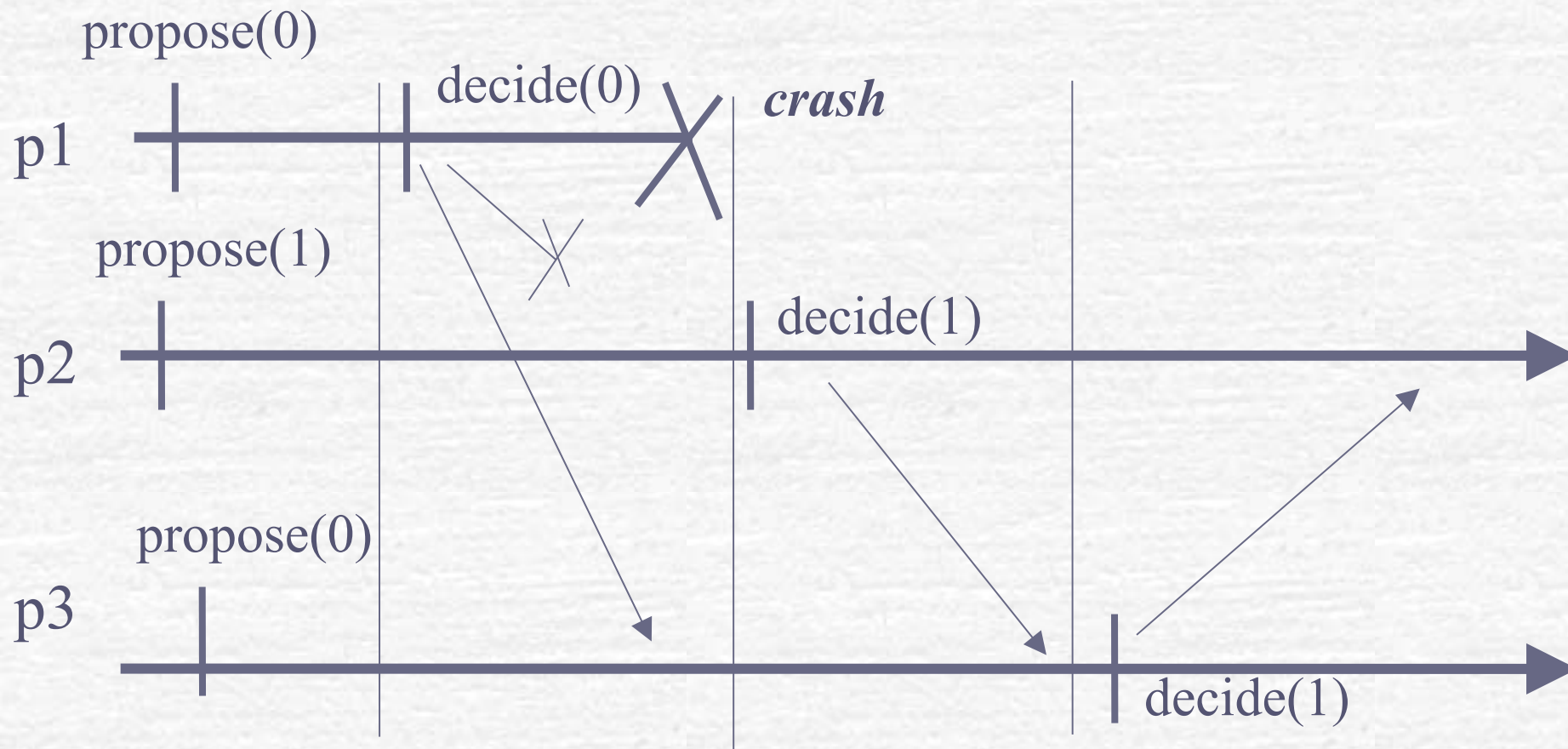
Consensus algorithm I

- **upon event** $p_{\text{round}} = \text{self}$ **and** $\text{broadcast} = \text{false}$ **and** $\text{currentProposal} \neq \text{nil}$ **do**
 - **trigger** $\langle \text{Decide}, \text{currentProposal} \rangle$;
 - **trigger** $\langle \text{bebBroadcast}, \text{currentProposal} \rangle$;
 - $\text{broadcast} := \text{true}$;

Consensus algorithm I



Consensus algorithm I



Correctness argument

- Let p_i be the correct process with the smallest id in a run R .
- Assume p_i decides v .
 - If $i = n$, then p_n 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 p_I 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 := $\emptyset$;
 - round := 1; currentProposal := nil;
 - broadcast := delivered[] := false;
 - decided := false

Consensus algorithm II

• **upon event** $\langle \text{crash}, p_i \rangle$ **do**

 suspected $:=$ suspected $\cup \{p_i\};$

• **upon event** $\langle \text{Propose}, v \rangle$ **do**

if currentProposal = nil **then**

 currentProposal $:= v;$

Consensus algorithm II

```
upon event < bebDeliver,  $p_{\text{round}}$ , value > do  
  currentProposal := value;  
  delivered[round] := true;
```

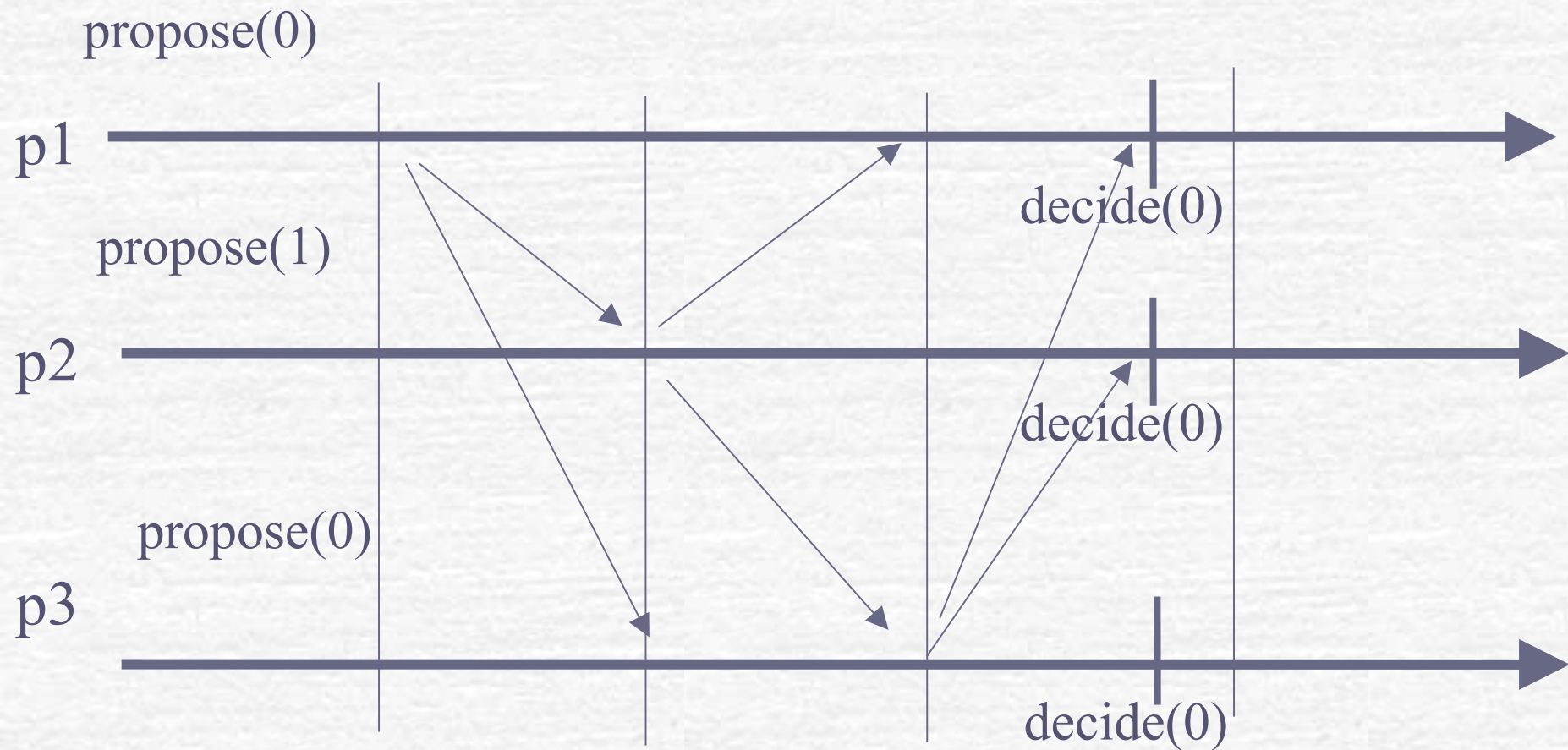

Consensus algorithm II

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

Consensus algorithm II

```
upon event  $p_{\text{round}} = \text{self}$  and  
    broadcast = false and  
    currentProposal  $\neq$  nil do  
    trigger <bebBroadcast, currentProposal>;  
    broadcast := true;
```

Consensus algorithm II



Correctness argument (A)

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

Correctness argument (B)

- **Uniform agreement:** Consider the process with the lowest id which decides, say p_I . Thus, p_I completes round n . By our previous lemma, in round I , every p_J with $J > I$ receives the currentProposal of p_I 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 .

Consensus algorithm III

- A uniform consensus algorithm assuming:
 - a correct majority
 - a $\langle \rangle P$ failure detector
- Basic idea: the processes alternate in the role of a phase coordinator until one of them succeeds in imposing a decision

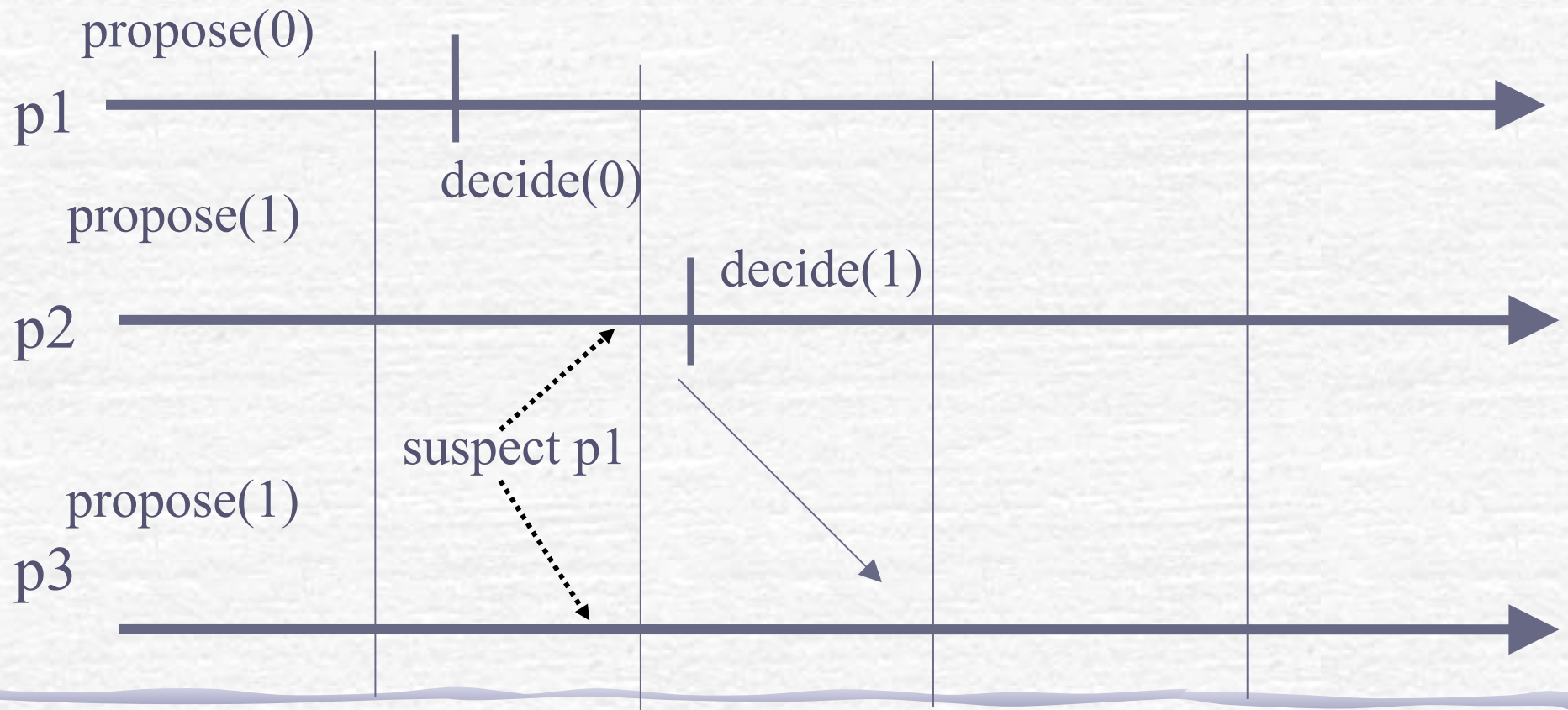
Consensus algorithm III

- $\langle \rangle P$ 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

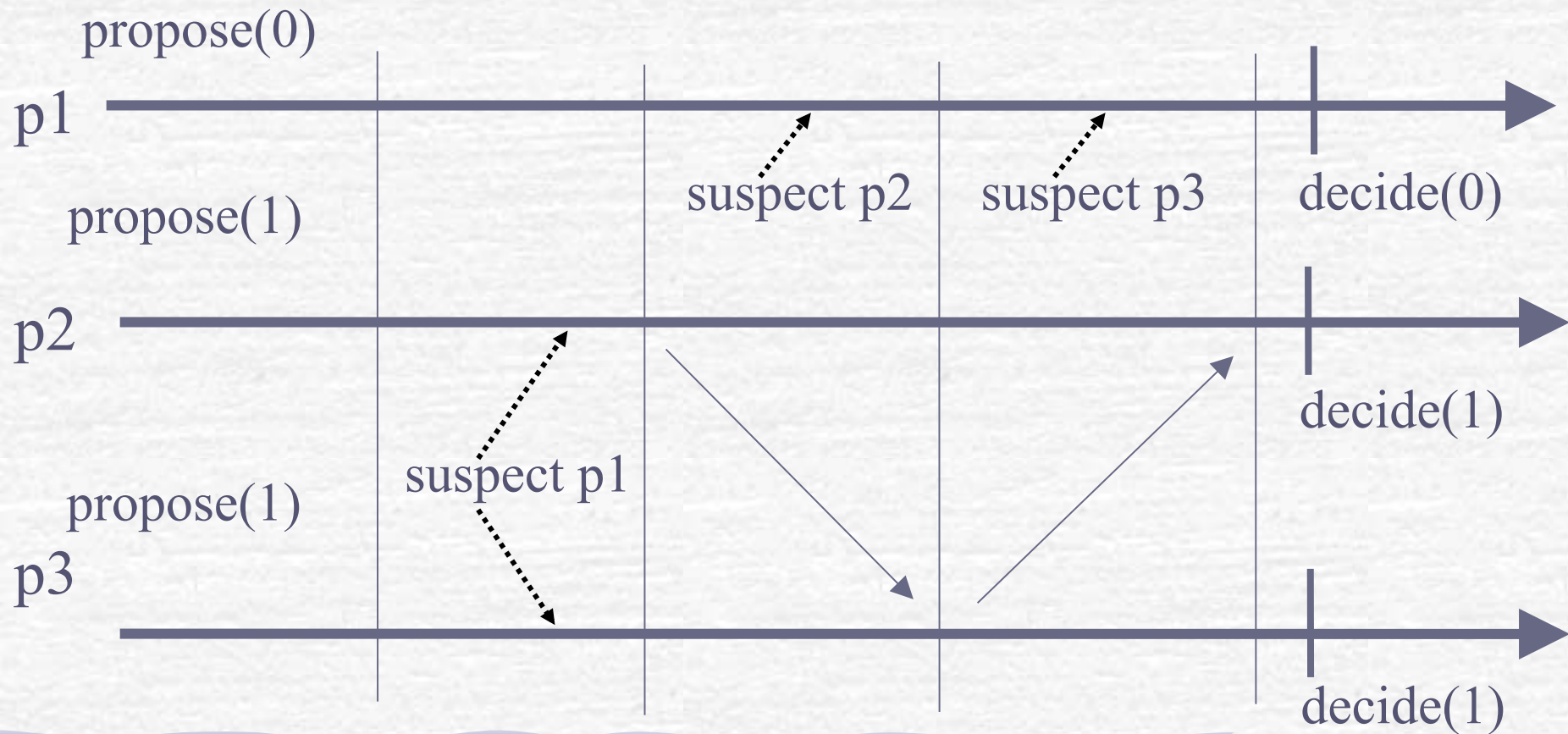
"<>" makes a difference

- ***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
 - Counter examples for algorithm I and II (see next slide)

Agreement violated with $\langle \rangle P$ in algorithm I



Agreement violated with $<>P$ in algorithm II



Consensus algorithm III

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

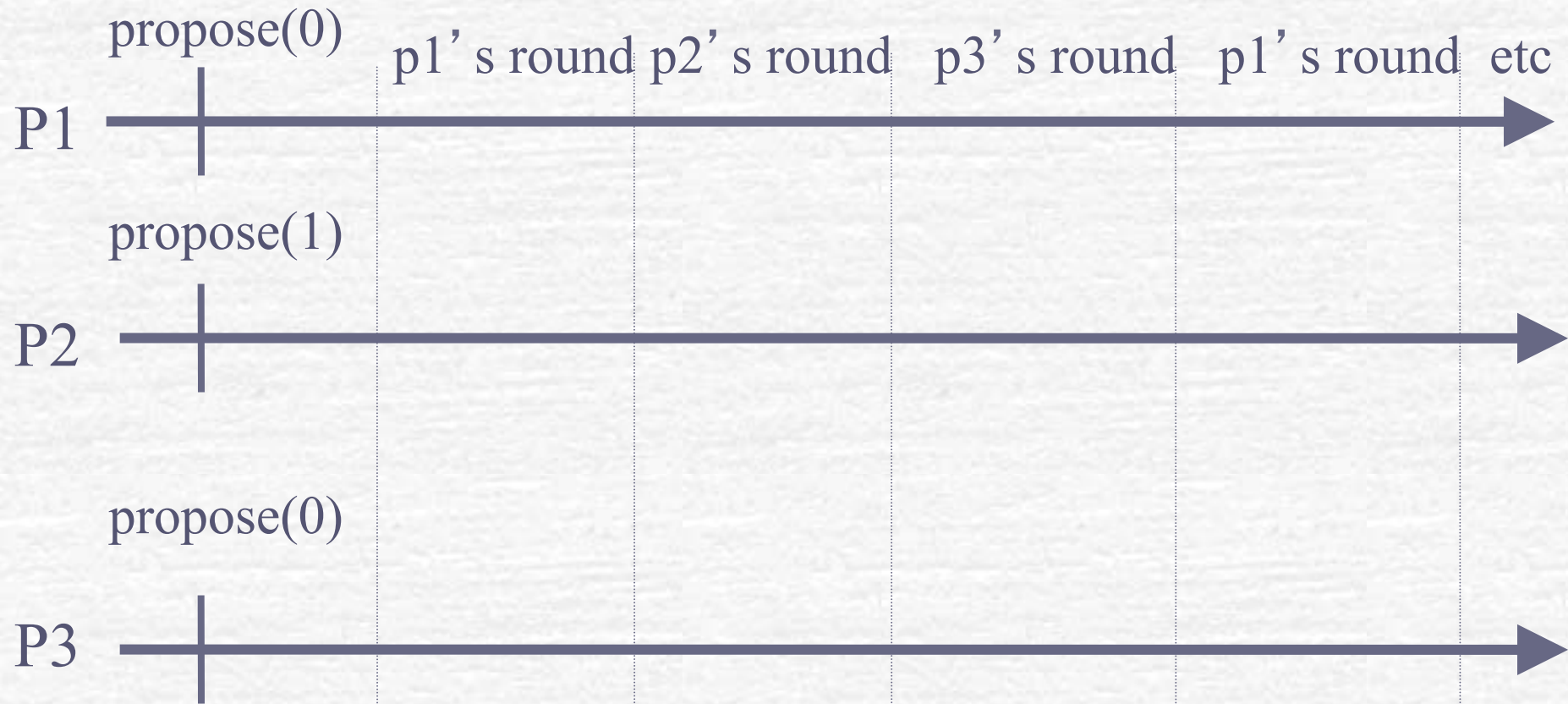
Consensus algorithm III

- p_i succeeds if it is not suspected (processes that suspect p_i inform p_i and move to the next round; p_i does so as well)
- If p_i succeeds, p_i uses a reliable broadcast to send the decision to all (the reliability of the broadcast is important here to preclude the case where p_i crashes, some other processes delivers the message and stop while the rest keeps going without the majority)

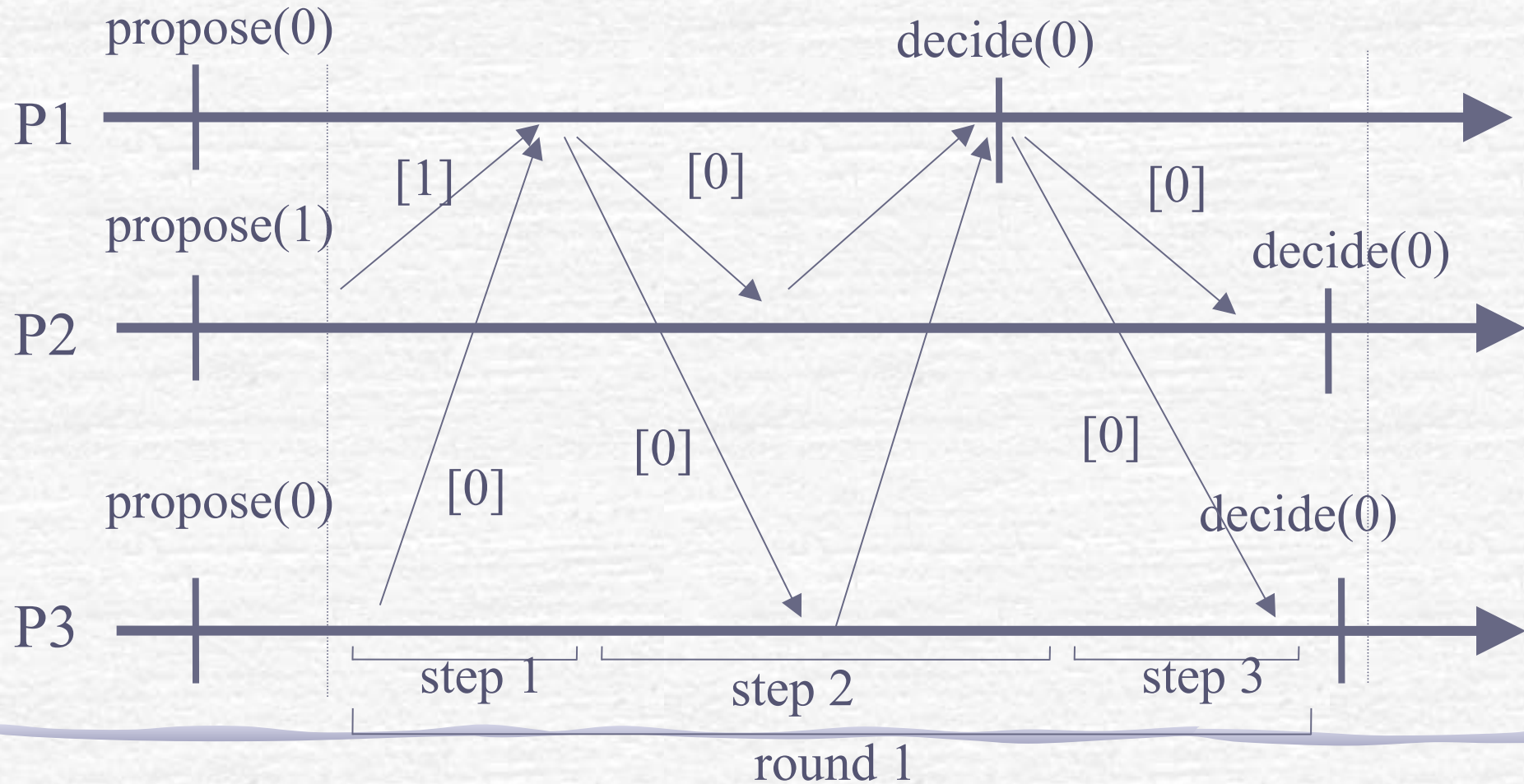
Consensus algorithm III

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

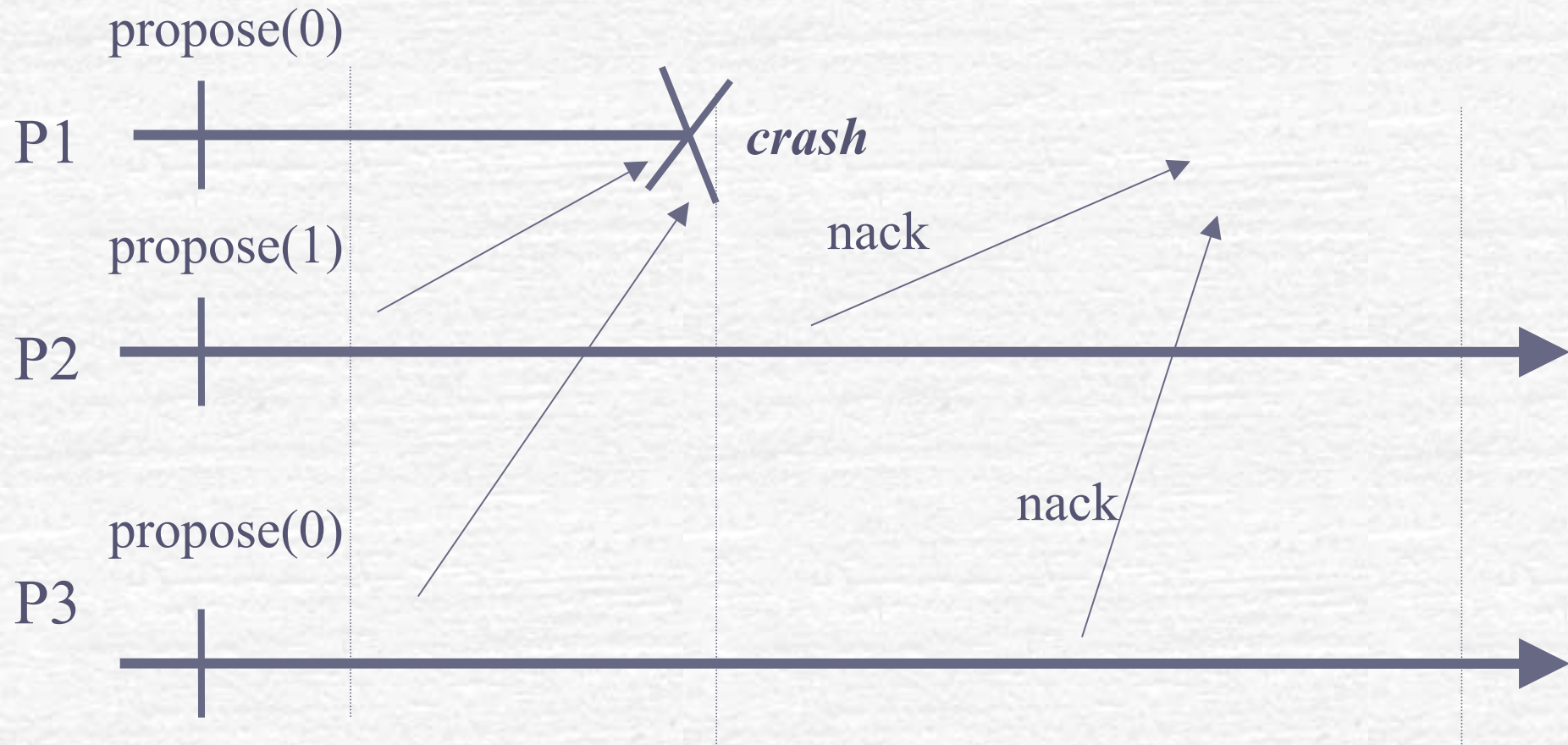
Consensus algorithm III



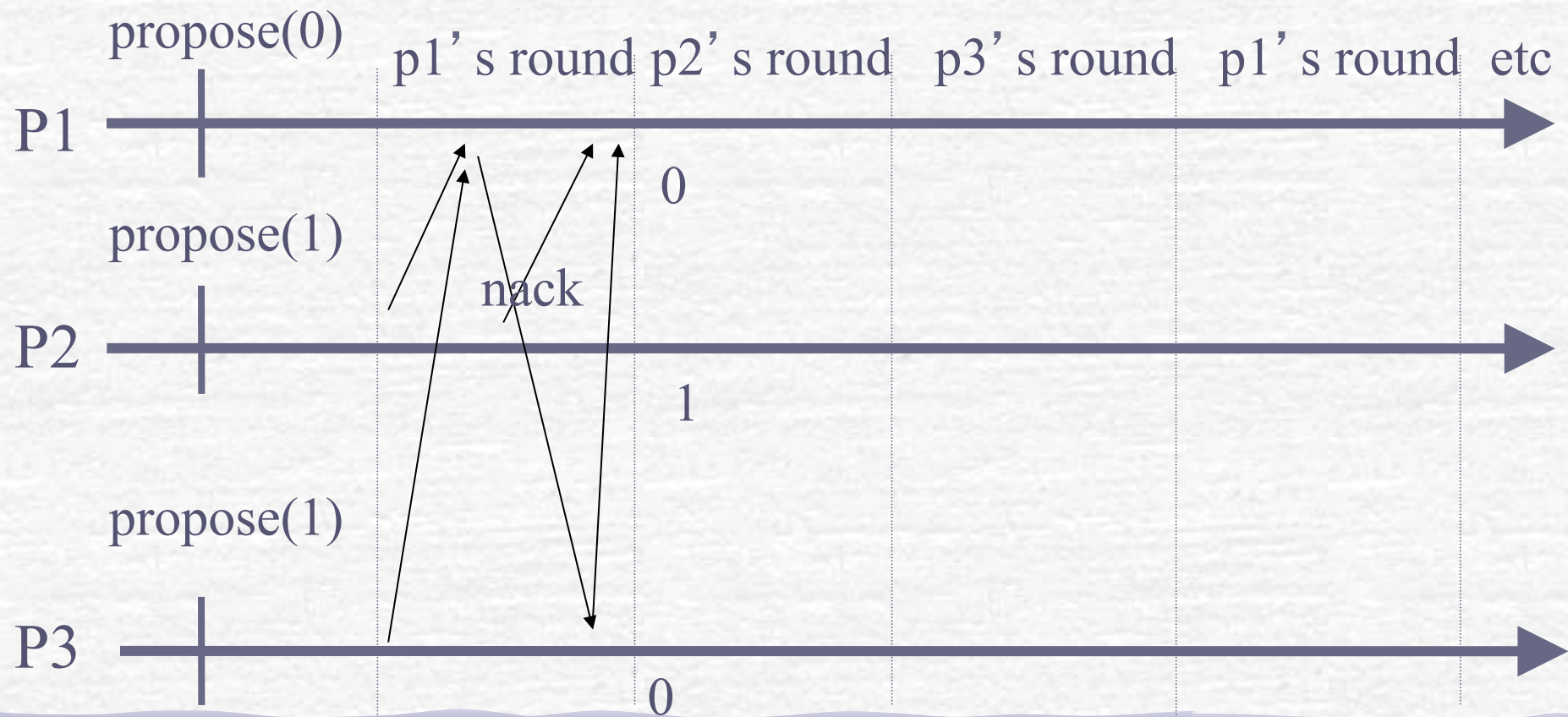
Consensus algorithm III



Consensus algorithm III



Consensus algorithm III



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'

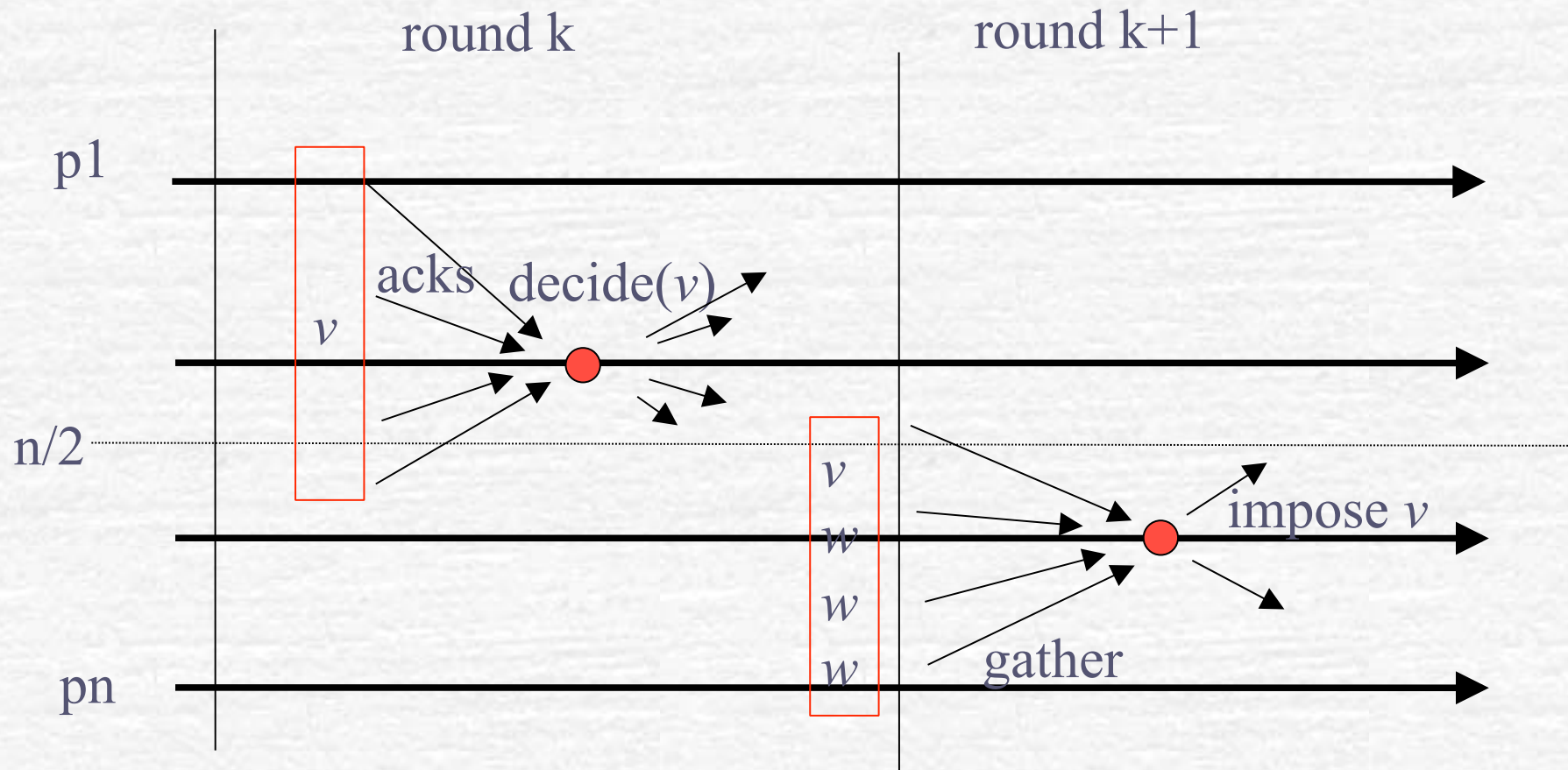
- By the correct **majority** assumption and the **completeness** property of the failure detector, no correct process remains blocked forever in some phase.
- By the **accuracy** property of the failure detector, some correct process reaches a round where it is leader and it is not suspected and reaches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

Correctness argument B

- Consider now *agreement*
- Let k be the first round in which some process p_i decides some value v , i.e., p_i is the leader of round k and p_i 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

new



● = leader of that round

Agreement is never violated

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

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 $\langle \rangle P$ and a correct majority