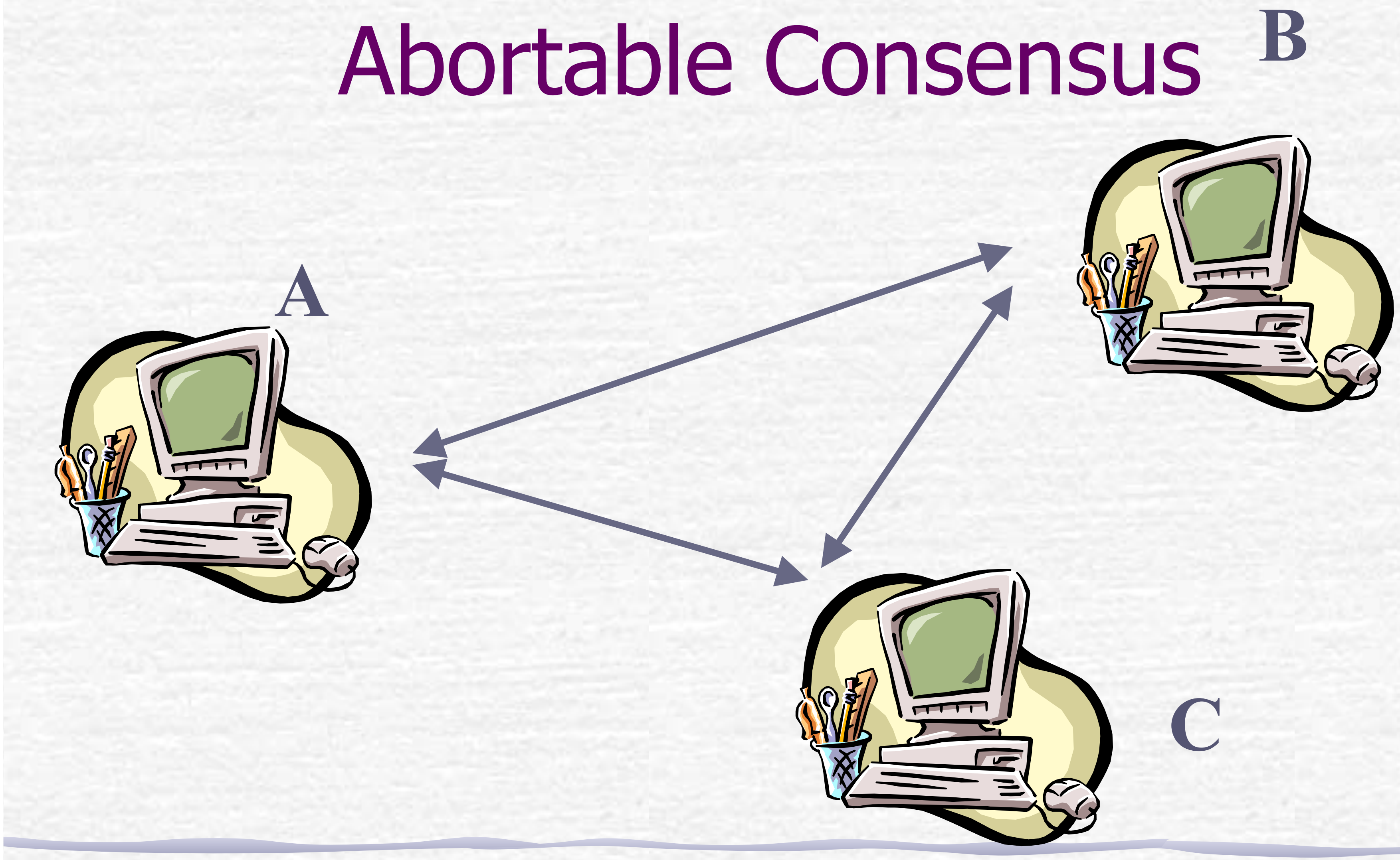
Distributed systems

Abortable Consensus

Prof R. Guerraoui

Distributed Programming Laboratory

Abortable Consensus



Abortable Consensus

- In the consensus problem, the processes propose values and have to agree on one among these values
- In weak consensus processes do not always need to decide: they can abort in case of contention

Specification

AC1. Validity: Any value decided is a value proposed

AC2. Agreement: No two processes decide differently

AC3. Termination: Every process that proposes a value eventually decides or aborts

AC4. Decision: If a single process proposes infinitely often, it eventually decides

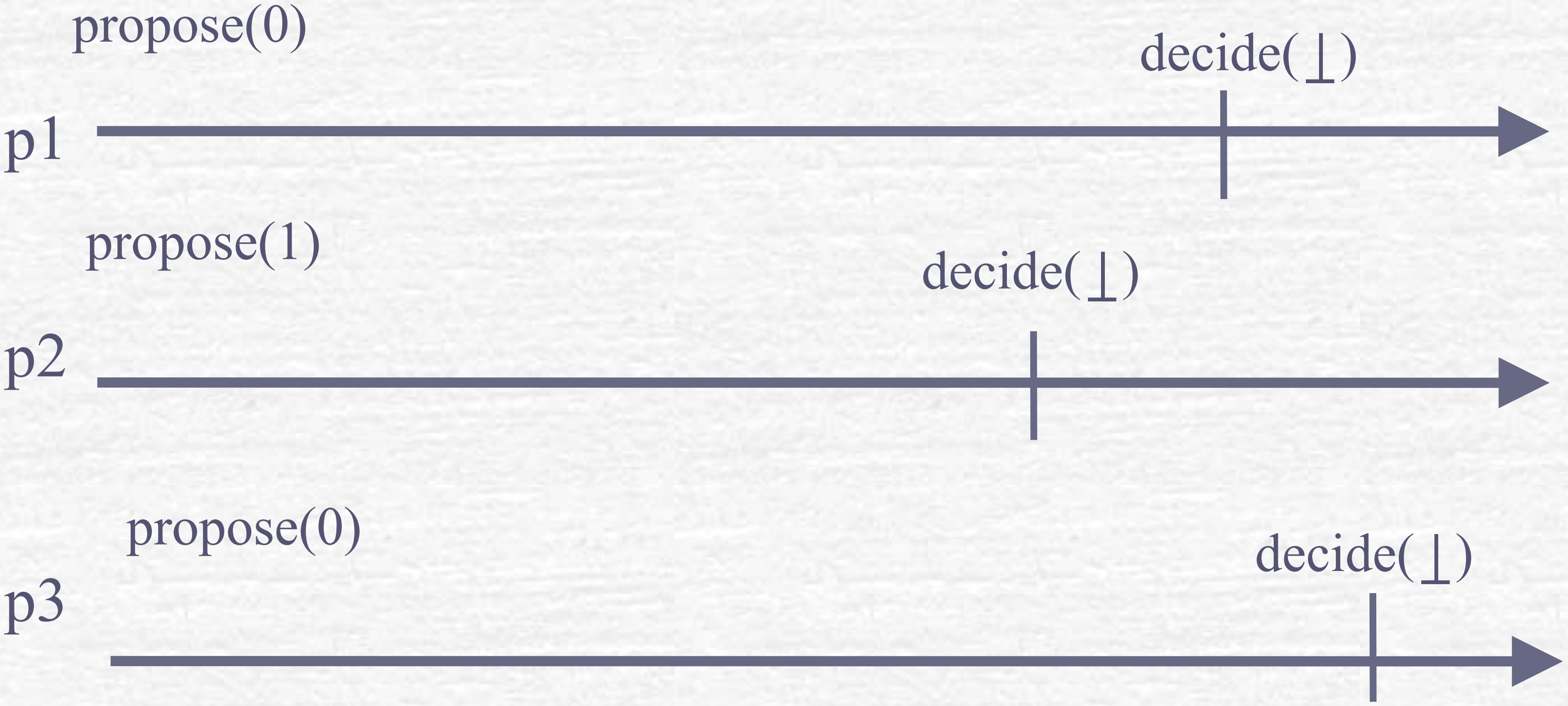
Abort

- Special value: ⊥
- Propose(v)
- Decide(v)
- Decide(⊥) → Abort

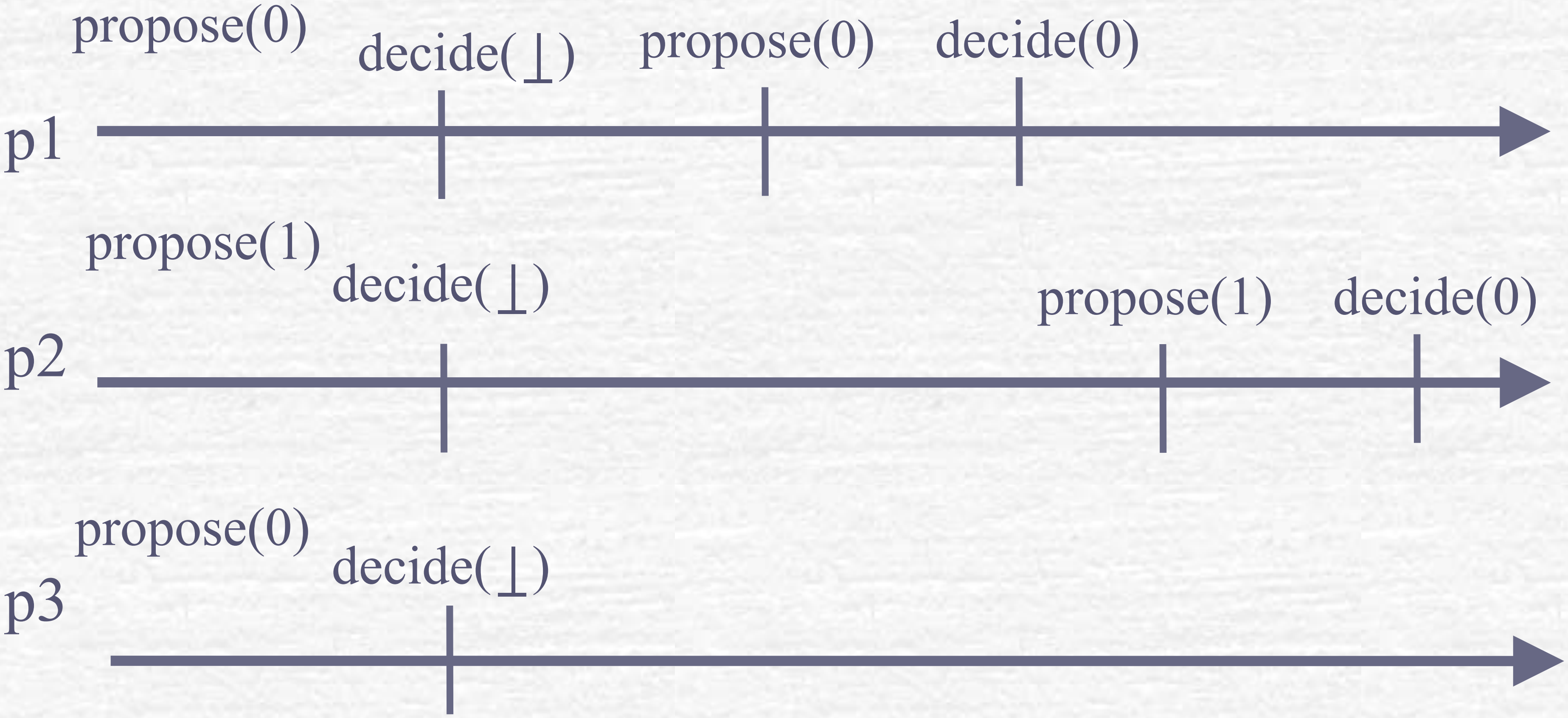
Abort

- Process might abort if another process concurrently tries to propose a value
- If only one process keeps proposing, then this process eventually decides

Run 1 OK



Run 2 OK



p1

propose(0) decide(⊥) propose(0) decide(0)

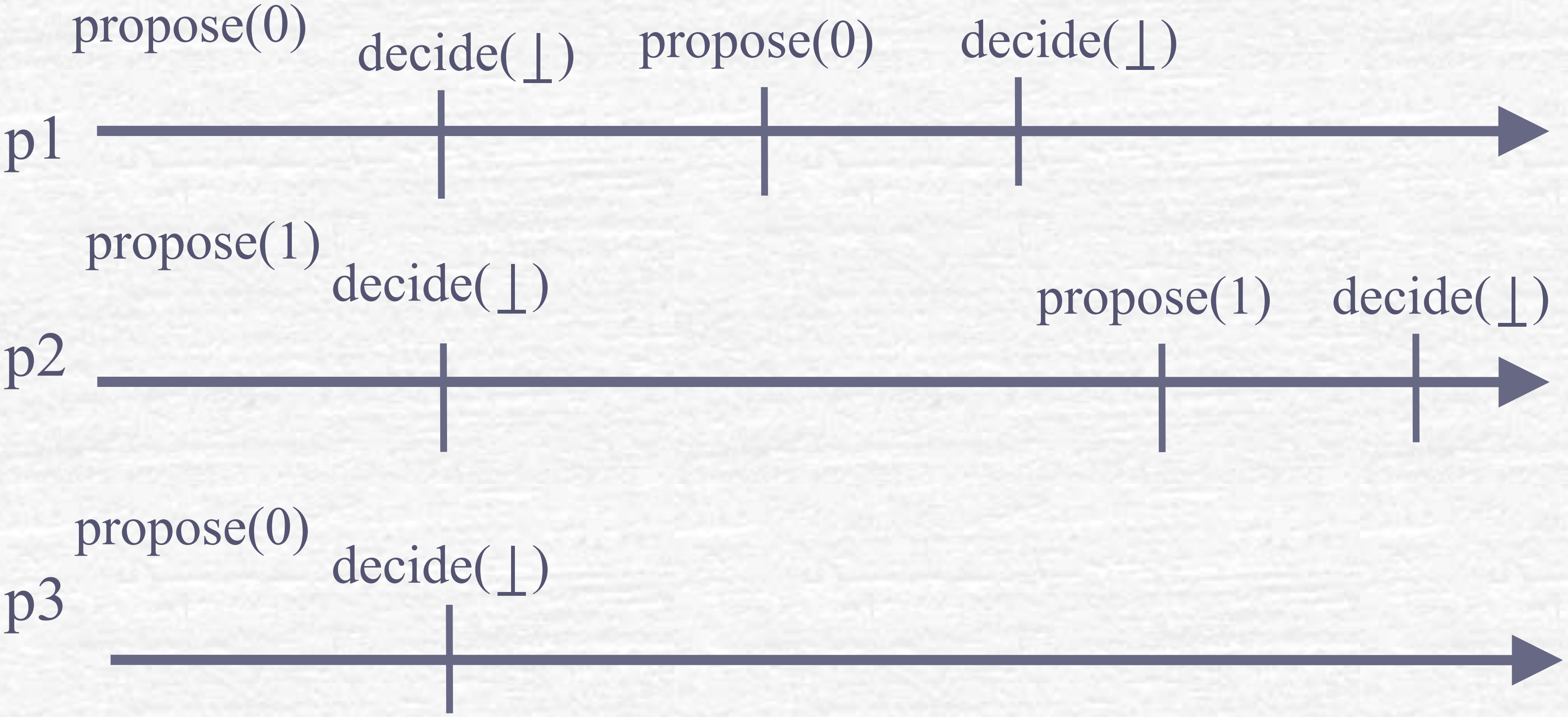
p2

propose(1) decide(⊥) propose(1) decide(0)

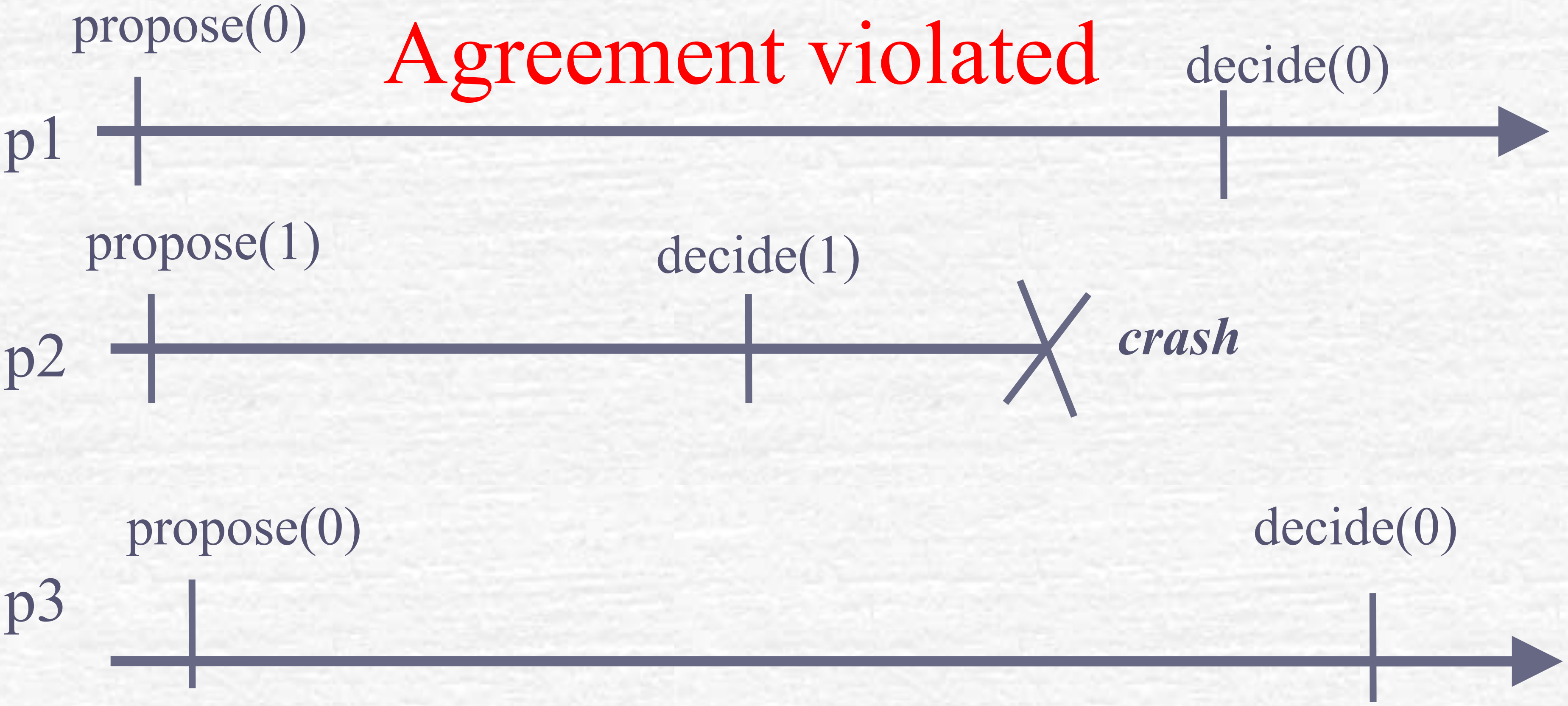
p3

propose(0) decide(⊥)

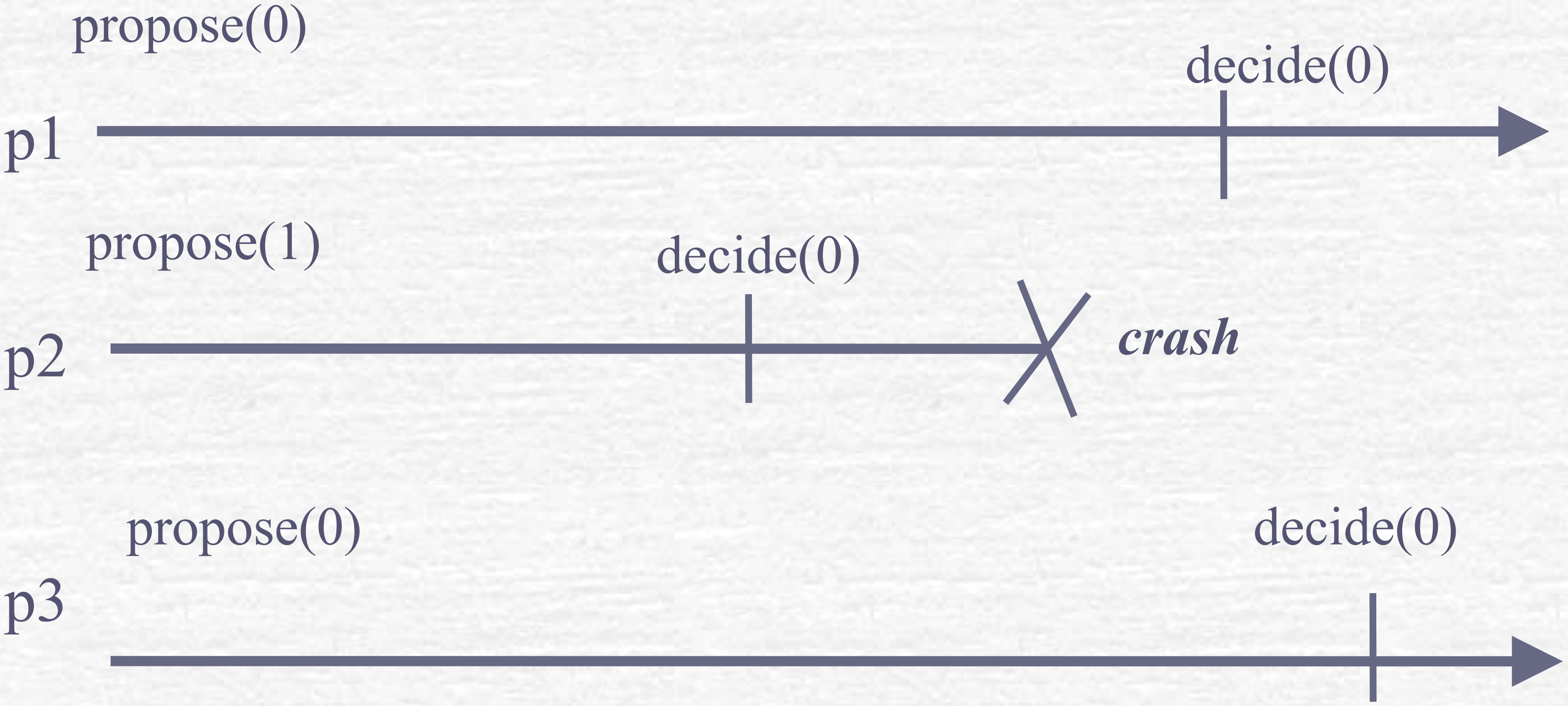
Run 3 OK


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

Run 4


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

Run 5 OK


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

RW Abortable Consensus Alg.

- Majority of correct processes
- Fail-silent
- No failure detector

RW Abortable Consensus Alg.

- Each processes keeps estimate of proposal and timestamp
- Two phases
- Read phase: check if estimate of the decision in system
- Write phase: reach a decision
- Any phase can abort → decide(⊥)

Read Phase

- **Implements:** Abortable Consensus (ac).
- **Uses:**
    - BestEffortBroadcast (beb).
    - PerfectPointToPointLinks (pp2p).
- **upon event** < Init > **do**
    - tstamp := rank(self)

Read Phase

```
upon event < acPropose, v> do
    tstamp := tstamp + N
    tempvalue := v
    trigger <bebBroadcast | [R, tstamp]>
upon event <bebDeliver|pj, [R,ts]>
    if rts ≥ ts or wts ≥ ts then
            trigger <Send | pj,[Nack]>
    else
            rts := ts
            trigger <Send | pj,[ReadAck,wts,val]>
```

Read Phase

```
upon event <Receive | pj,[Nack]> do
  trigger <acReturn | ⊥>
upon event <Receive |pj,[ReadAck,ts,v]>
  readSet := readSet U {(ts,v)}

upon (|readSet|>N/2) do
  (ts,v):=highest(readSet)
  if v != ⊥ then tempValue := v
  trigger <bebB | [W,tstamp, tempValue]>
```

Start write phase

Write Phase

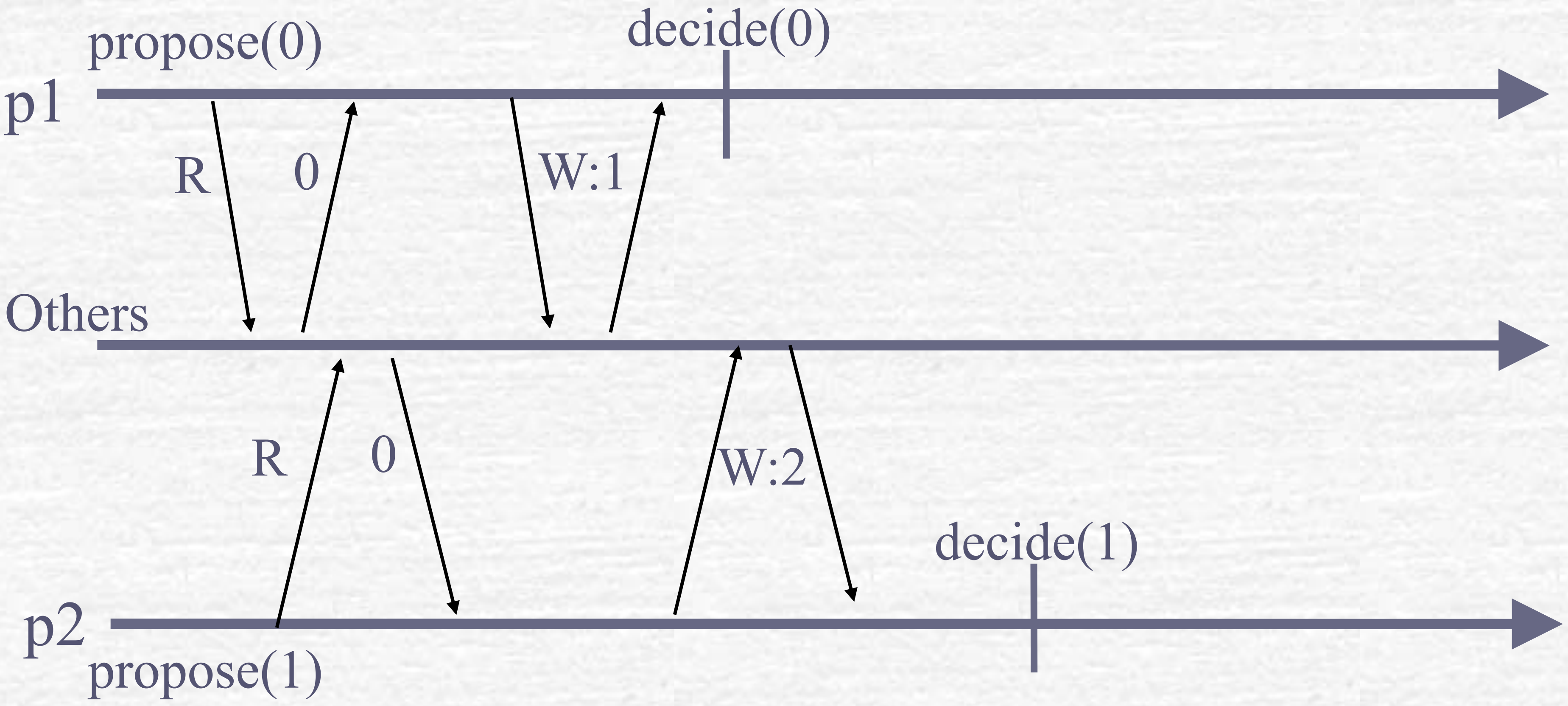
```
upon event <bebDeliver|pj, [W,ts,v]>
  if rts > ts or wts > ts then
      trigger <Send | pj,[Nack]>
  else
      val := v
      wts := ts
      trigger <Send | pj,[WriteAck]>
```

Write Phase

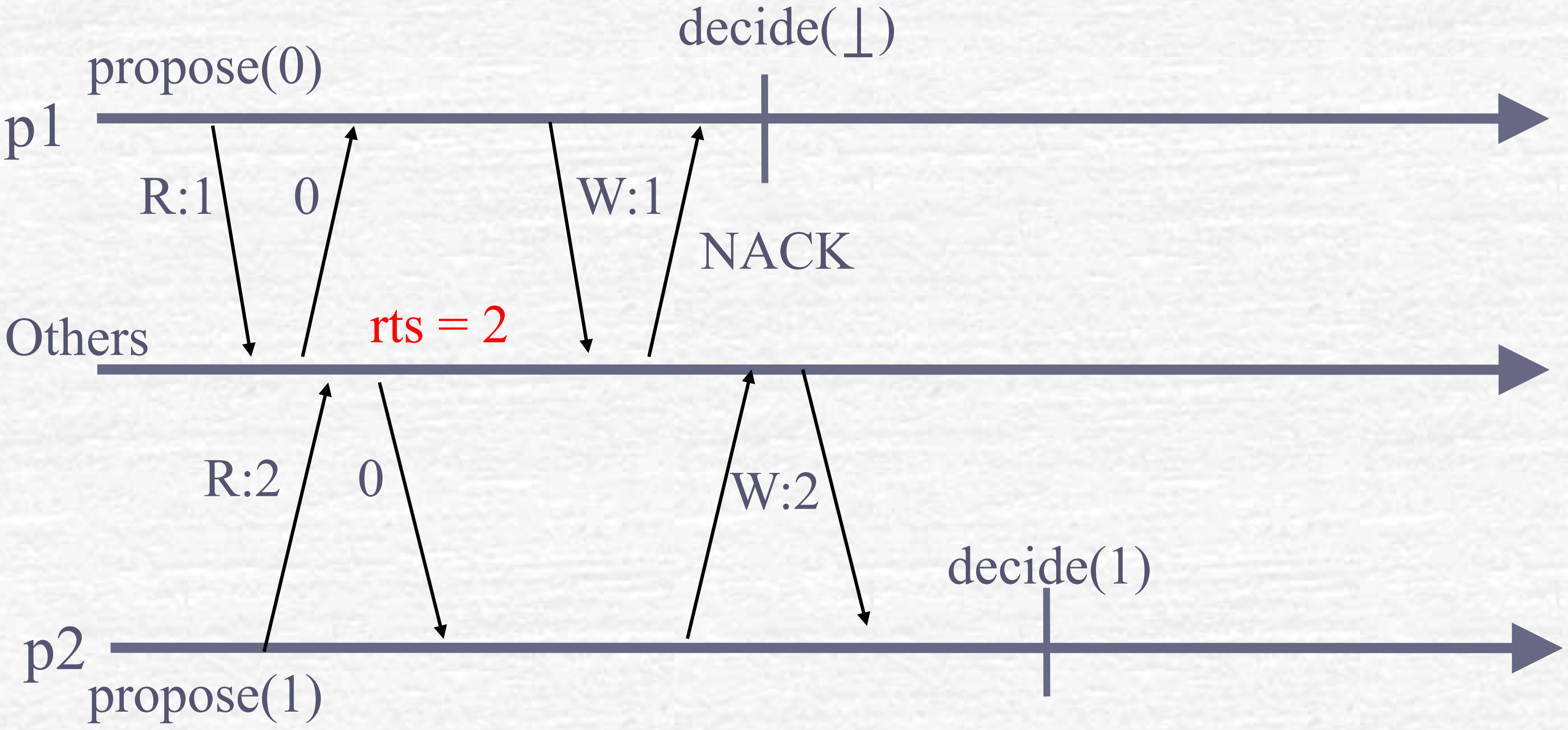
```
upon event <Receive | pj,[Nack]> do
  trigger <acReturn | _l>
upon event <Receive |pj,[WriteAck]>
  wAcks++

upon (wAcks > N/2) do
  readSet := empty
  wAcks := 0
  trigger <acReturn | tempValue>
```

Do we need **rts** ? Example with only one ts



With rts


decide(⊥)
propose(0)
p1
R:1
0
W:1
NACK
Others
rts = 2
R:2
0
W:2
decide(1)
p2
propose(1)