



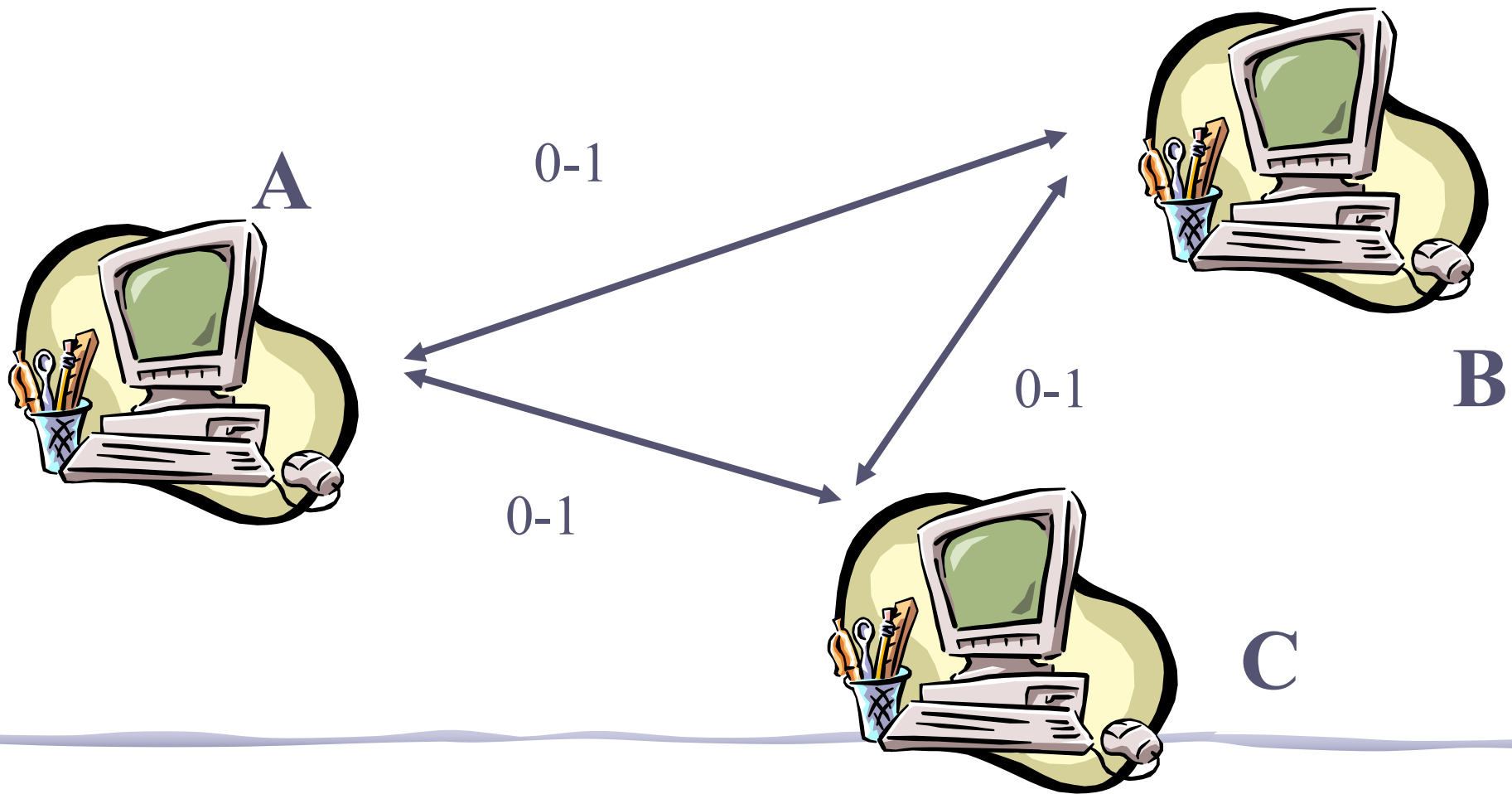
Distributed Systems

Non-Blocking Atomic Commit

Prof R. Guerraoui
Distributed Programming Laboratory



Non-Blocking Atomic Commit: An Agreement Problem





Transactions (Gray)

- A transaction is an atomic program describing a sequence of accesses to shared and distributed information
- A transaction can be terminated either by *committing* or *aborting*

Transactions

- beginTransaction
 - Pierre.credit(1.000.000)
 - Paul.debit(1.000.000)
- outcome := commitTransaction
- if (outcome = abort) then ...



ACID properties

Atomicity: a transaction either performs entirely or none at all

Consistency: a transaction transforms a consistent state into another consistent state

Isolation: a transaction appears to be executed in isolation

Durability: the effects of a transaction that commits are permanent

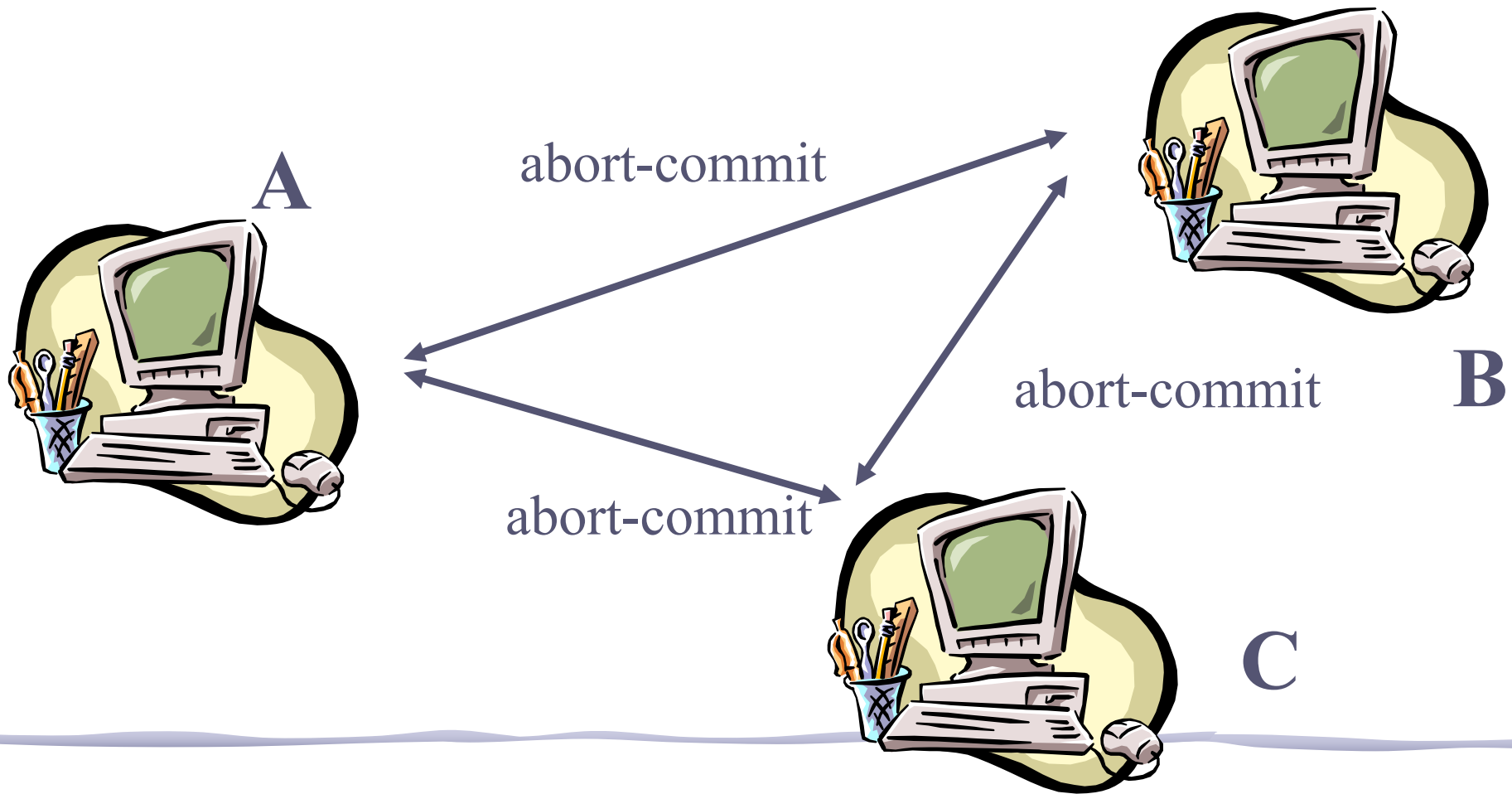
The Consistency Contract

(system)
Atomicity
Isolation
Durability

(programmer)
Consistency (local)

Consistency (global)

Distributed Transaction





Non-Blocking Atomic Commit

- As in consensus, every process has an initial value 0 (*no*) or 1 (*yes*) and must decide on a final value 0 (*abort*) or 1 (*commit*)
- The proposition means the ability to commit the transaction
- The decision reflects the contract with the user
- Unlike consensus, the processes here seek to decide 1 but every process has a veto right

Non-Blocking Atomic Commit

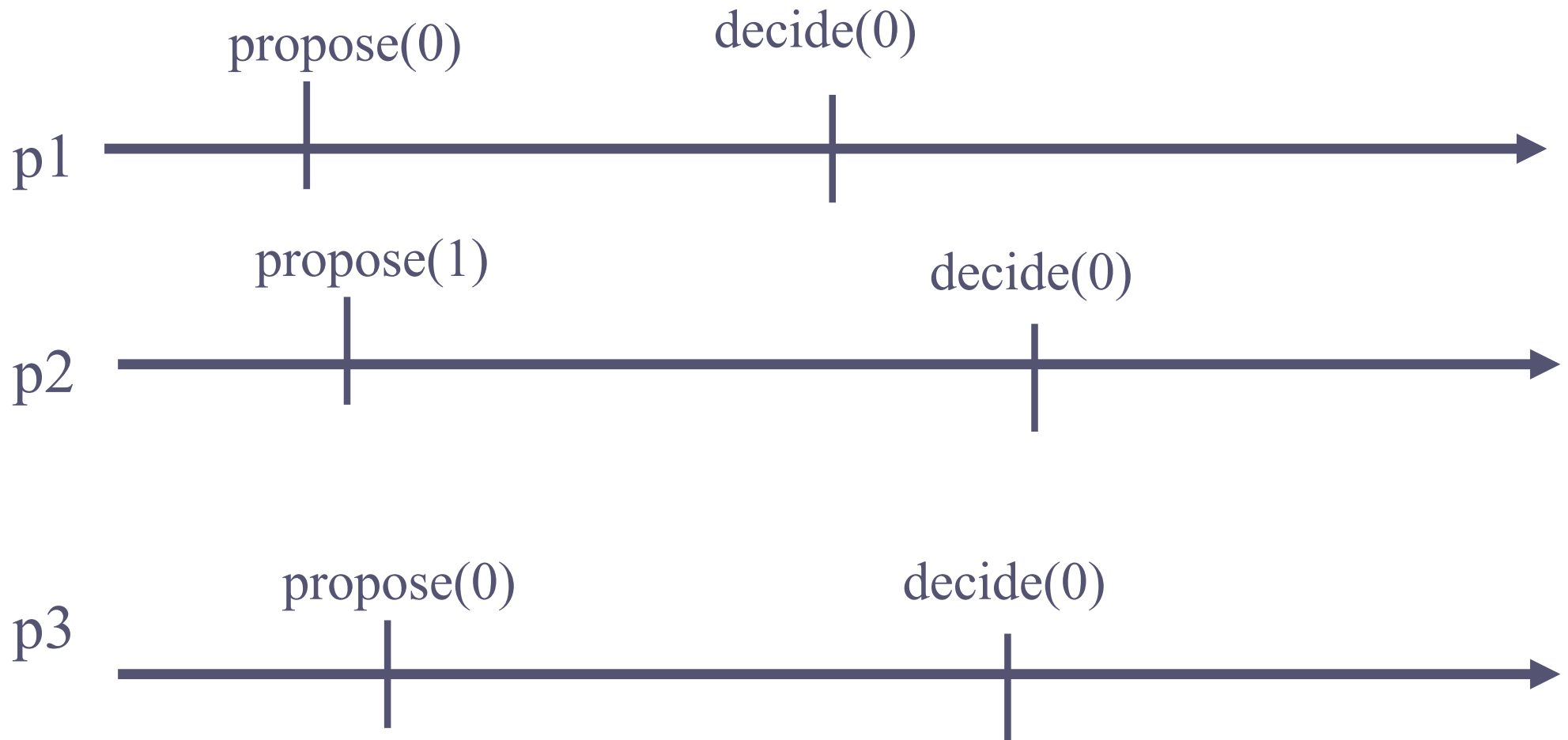
NBAC1. Agreement: No two processes decide differently

NBAC2. Termination: Every correct process eventually decides

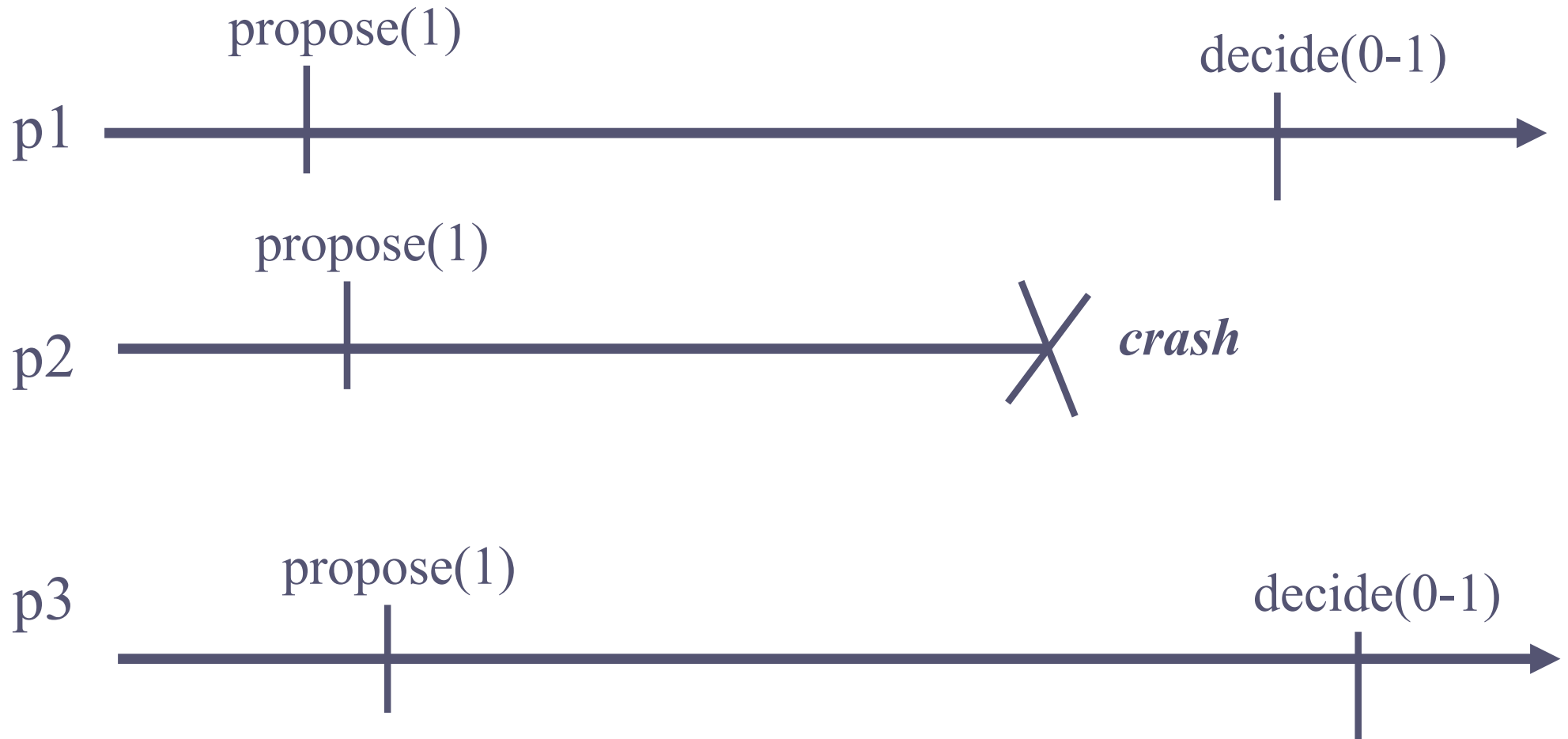
NBAC3. Commit-Validity: 1 can only be decided if all processes propose 1

NBAC4. Abort-Validity: 0 can only be decided if some process crashes or votes 0

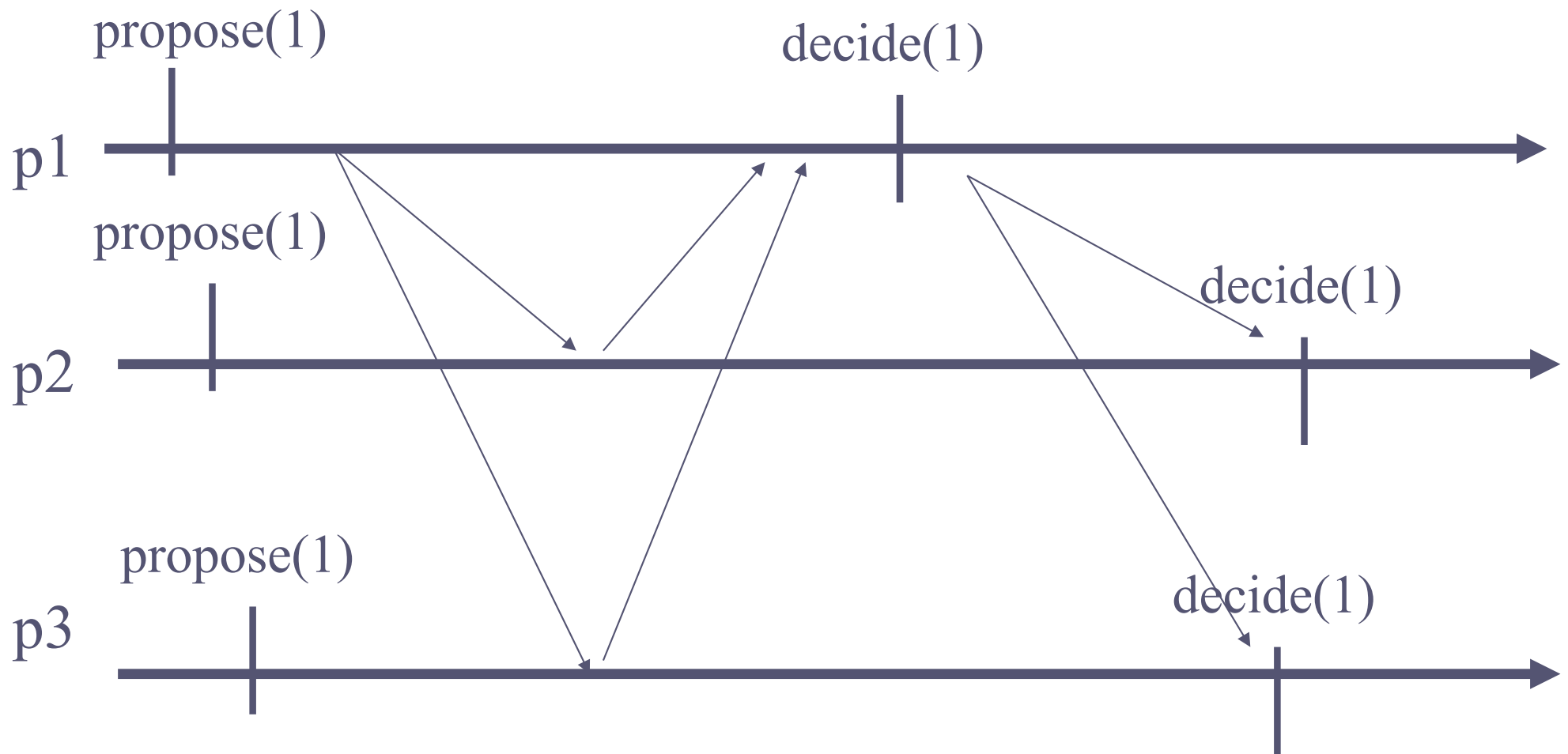
Non-Blocking Atomic Commit



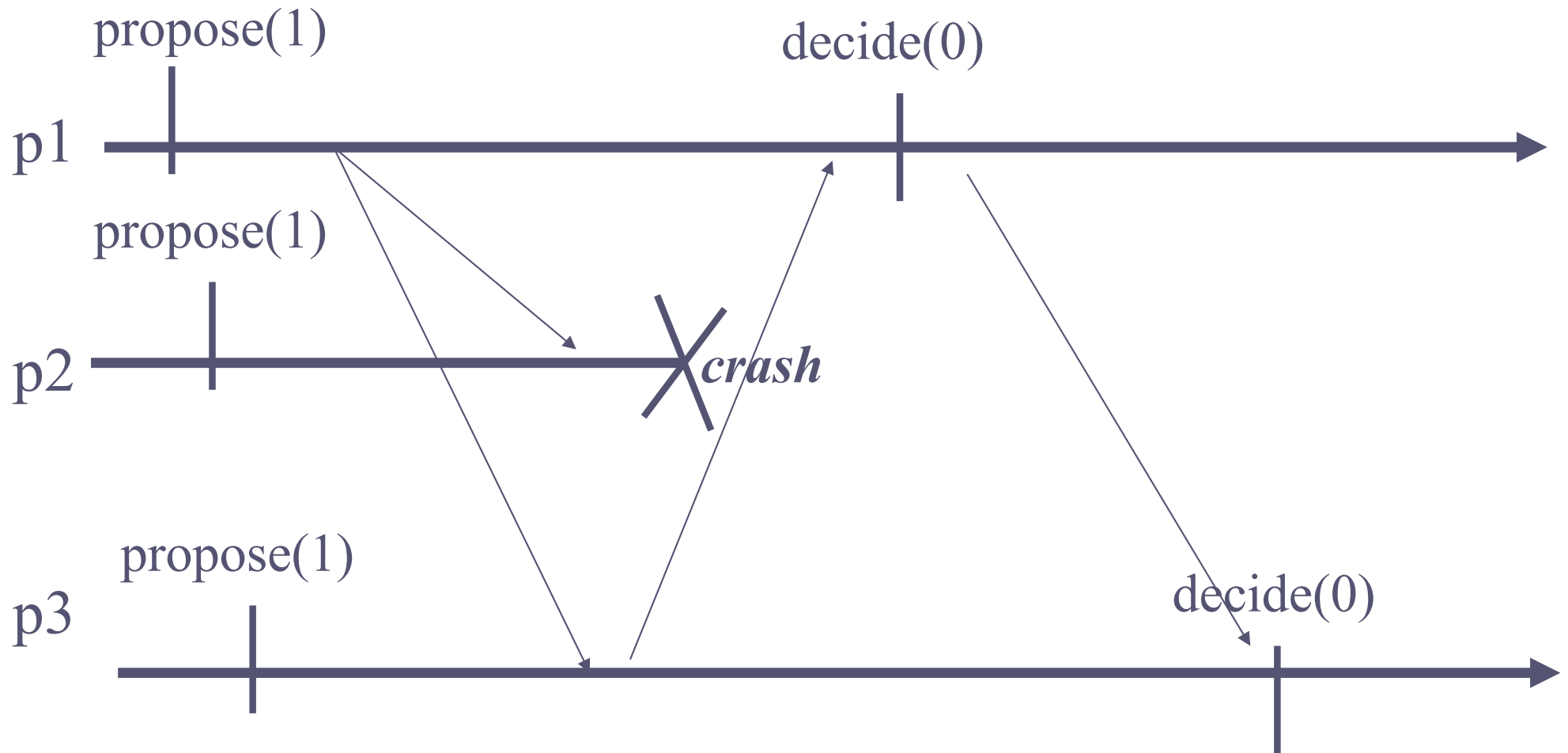
Non-Blocking Atomic Commit



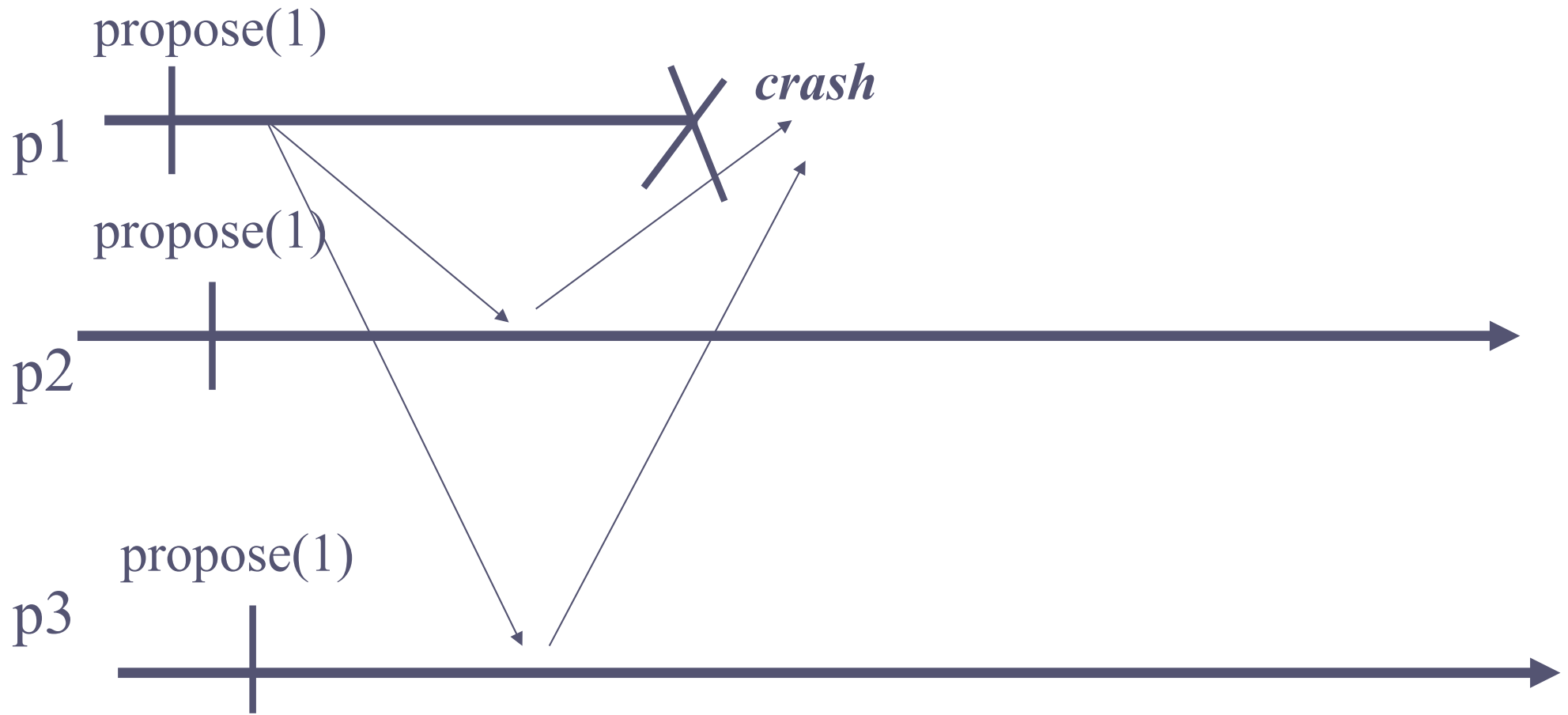
2-Phase Commit



2-Phase Commit



2-Phase Commit



Non-Blocking Atomic Commit

• *Events*

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

• *Properties:*

- ***NBAC1, NBAC2, NBAC3, NBAC4***

Algorithm (nbac)

- **Implements:** nonBlockingAtomicCommit (nbac).
- **Uses:**
 - BestEffortBroadcast (beb).
 - PerfectFailureDetector (P).
 - UniformConsensus (uniCons).
- **upon event** < Init > **do**
 - `prop := 1;`
 - `delivered := \emptyset ; correct := Π ;`

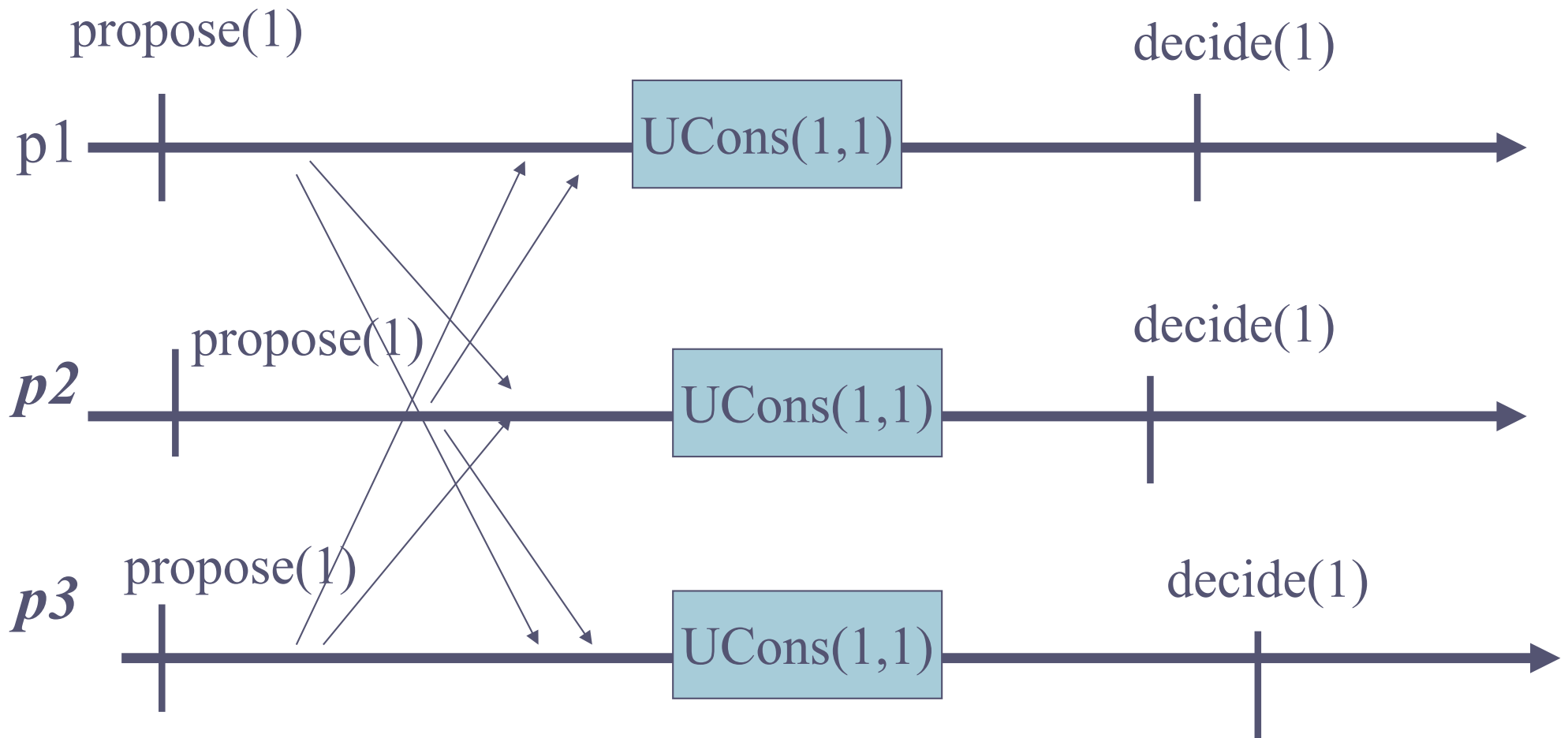
Algorithm (nbac – cont'd)

- **upon event** $\langle \text{crash}, p_i \rangle$ **do**
 - $\text{correct} := \text{correct} \setminus \{p_i\}$
- **upon event** $\langle \text{Propose}, v \rangle$ **do**
 - **trigger** $\langle \text{bebBroadcast}, v \rangle$;
- **upon event** $\langle \text{bebDeliver}, p_i, v \rangle$ **do**
 - $\text{delivered} := \text{delivered} \cup \{p_i\}$;
 - $\text{prop} := \text{prop} * v$;

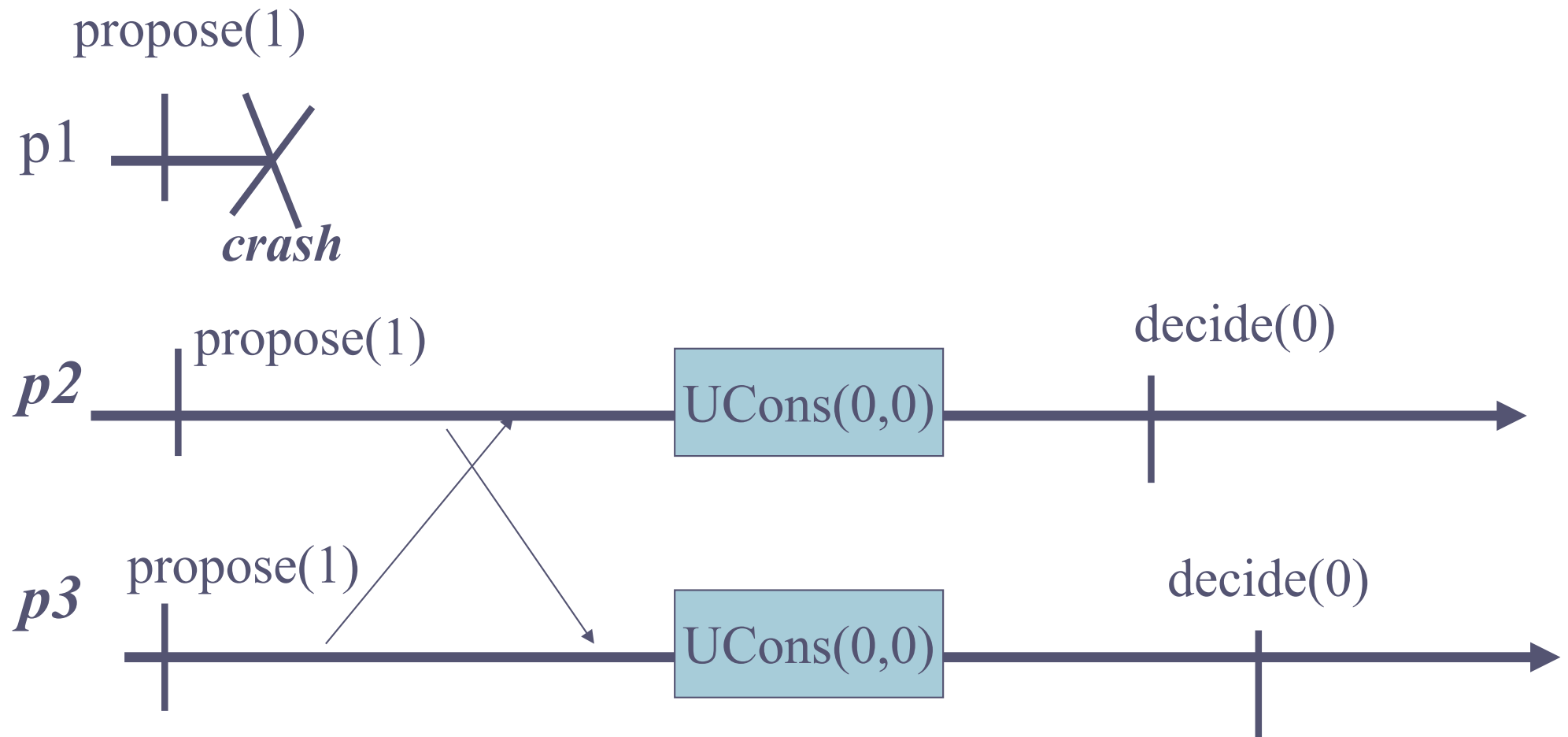
Algorithm (nbac – cont'd)

- **upon event** correct \ delivered = empty **do**
 - **if** correct $\neq \Pi$
 - prop := 0;
 - **trigger** < uncPropose, prop >;
- **upon event** < uncDecide, decision > **do**
 - **trigger** < Decide, decision >;

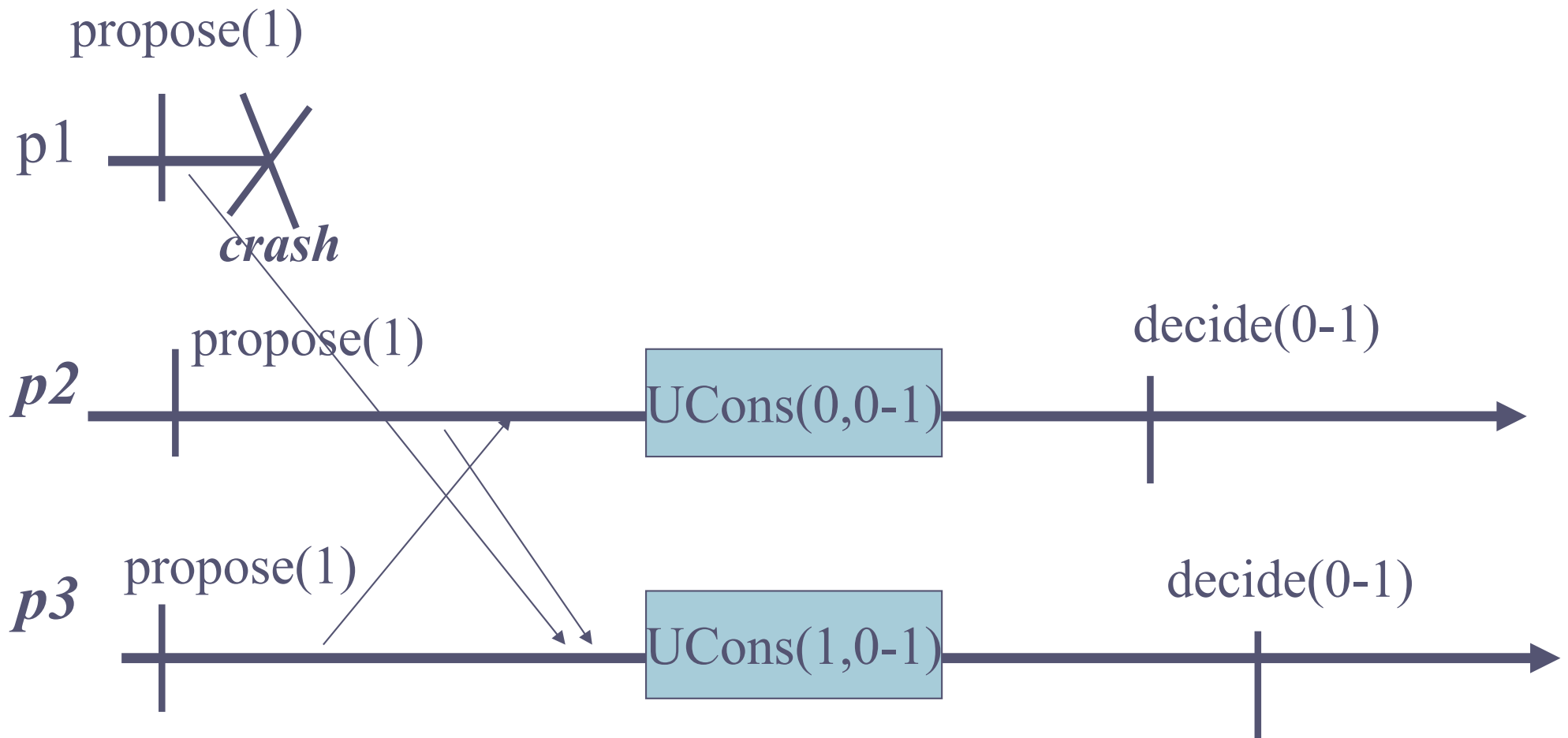
nbac with ucons



nbac with ucons



nbac with ucons





Non-Blocking Atomic Commit

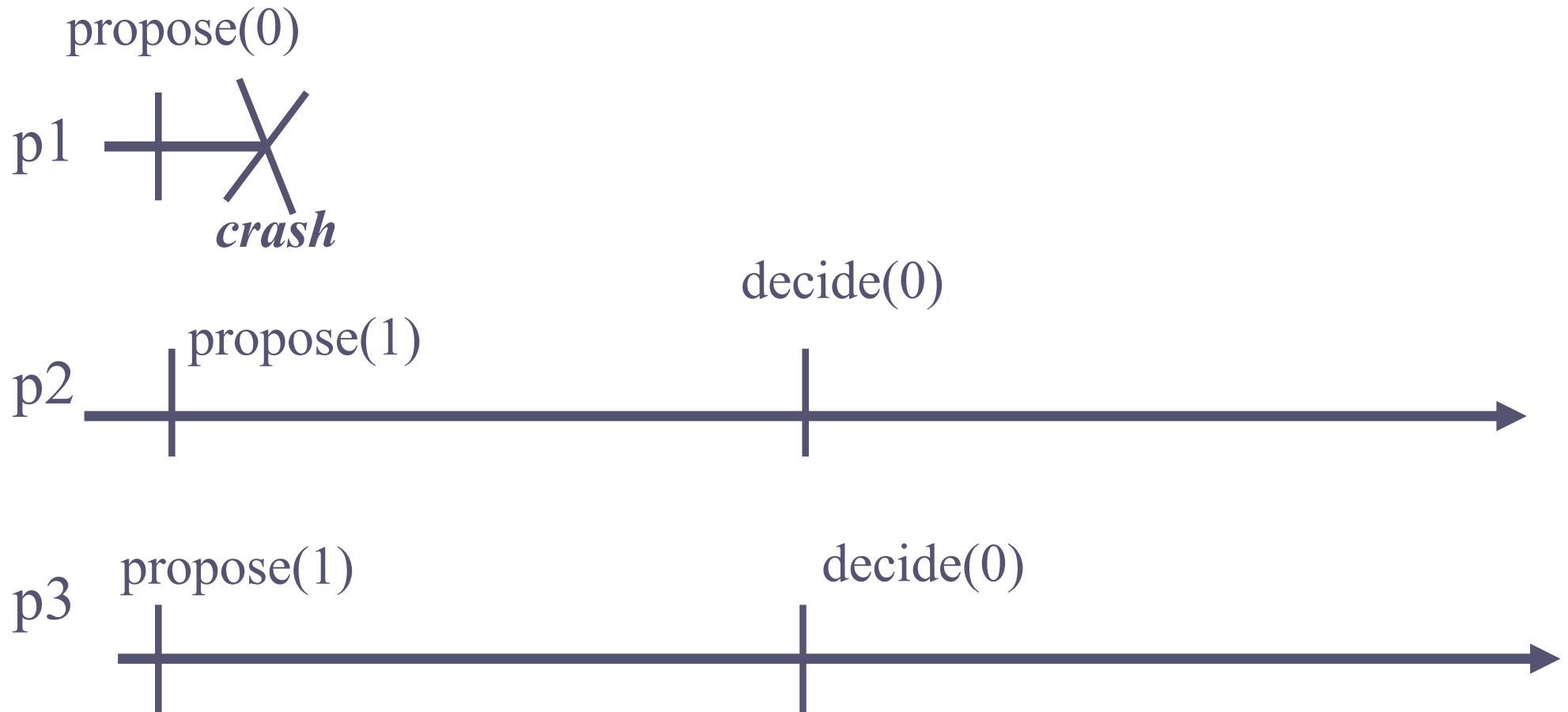
- Do we need perfect failure detector P?
 - 1. $\langle \rangle P$ is not enough
 - 2. P is needed if one process can crash



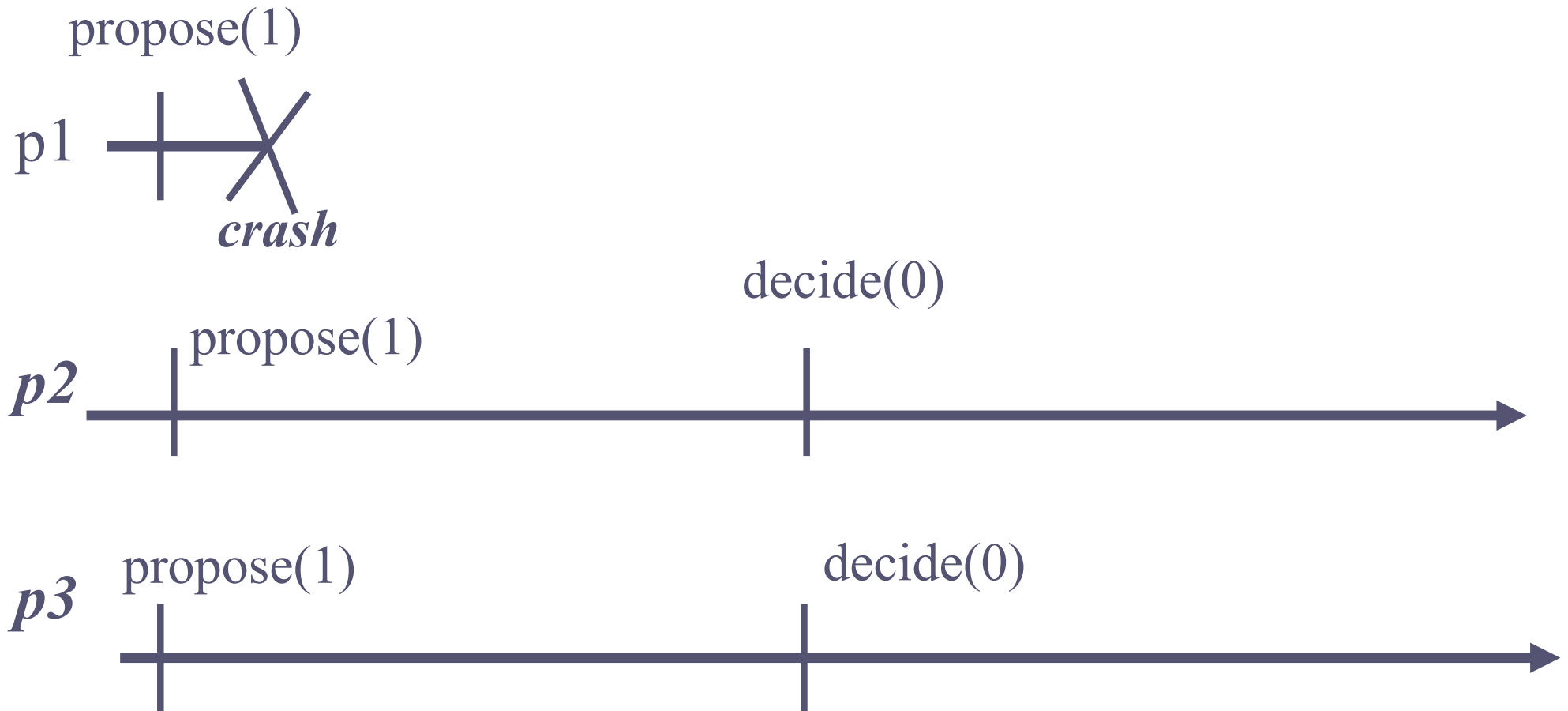
Non-Blocking Atomic Commit

- Do we need perfect failure detector P?
 - **1. $\langle \rangle P$ is not enough**
 - 2. P is needed if one process can crash

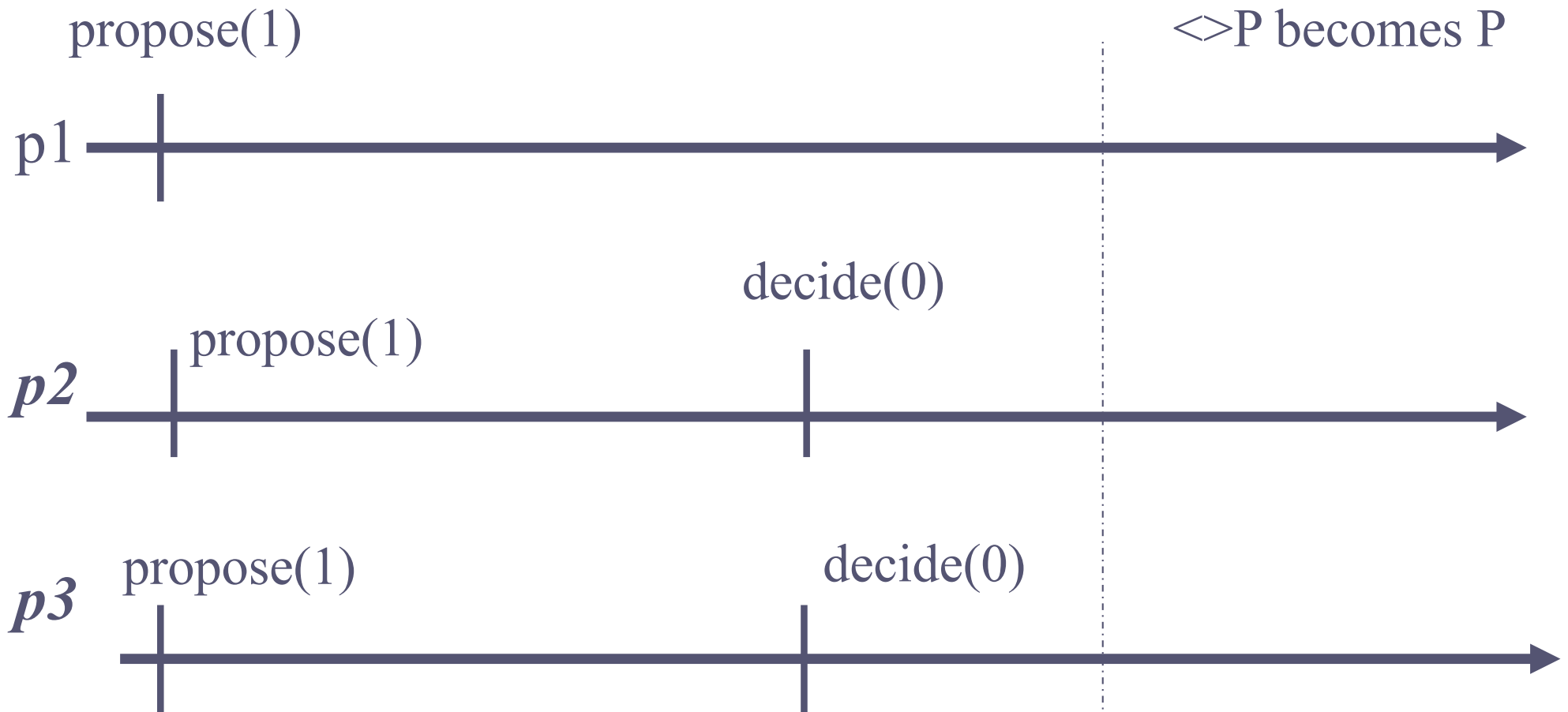
1. Run 1



1. Run 2



1. Run 3

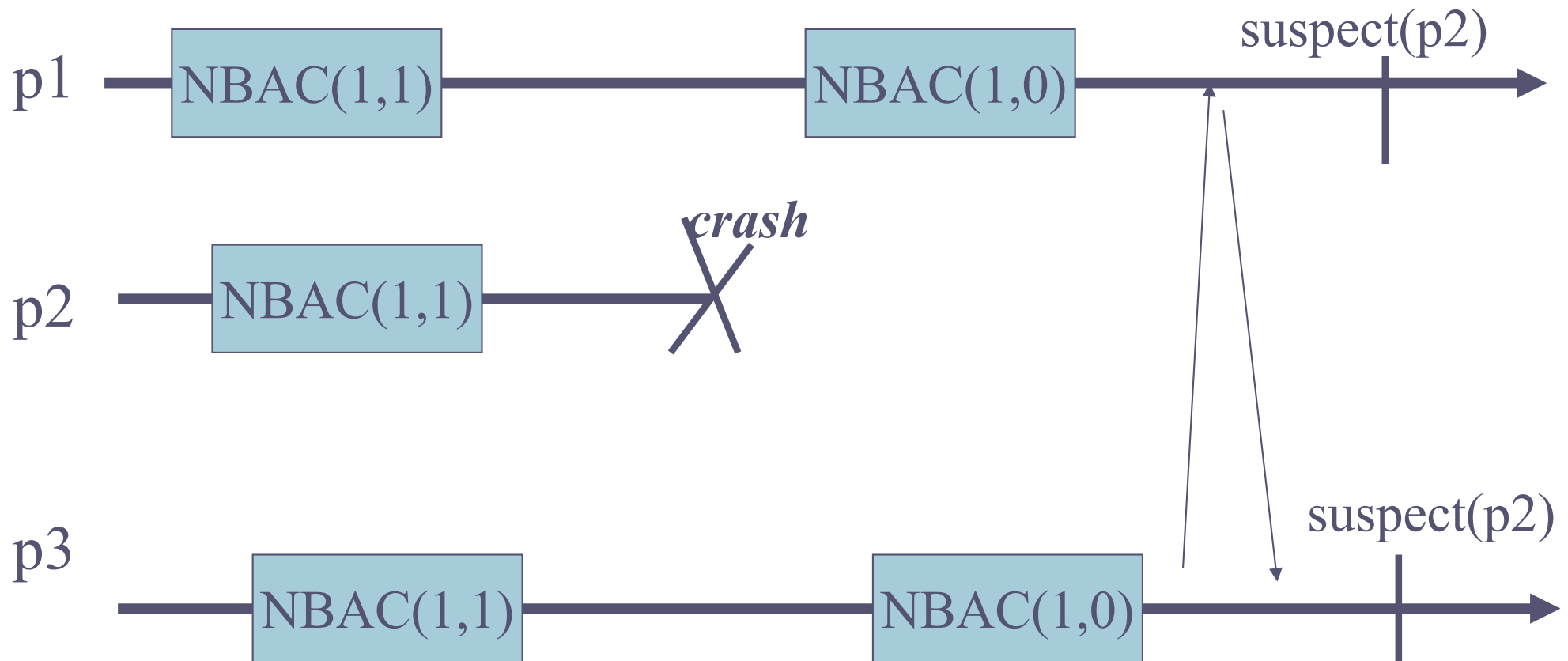




Non-Blocking Atomic Commit

- Do we need perfect failure detector P ?
 - 1. $\langle \rangle P$ is not enough
 - **2. P is needed if one process can crash**

2. P is needed with one crash



History

- Atomic Commit (Eswaran/Gray 76 – Gray 78)
- NBAC (Skeen 81)
 - Complexity of Sync NBAC (DS 83)
- Async NBAC (Had 90 – Gue 95)
 - Fast Async NBAC (KD95, GLS95, GL06)
- FD NBAC (DFGHTK 04)
 - Optimal NBAC (GW17)