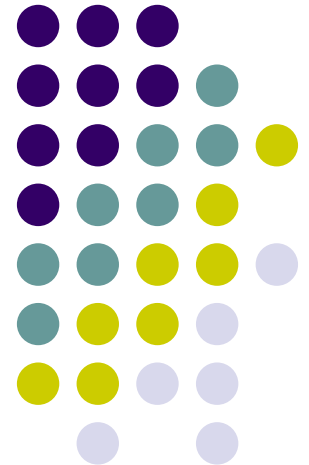


Distributed Algorithms for Building Reliable Systems

Seif Haridi

Total Order Broadcast



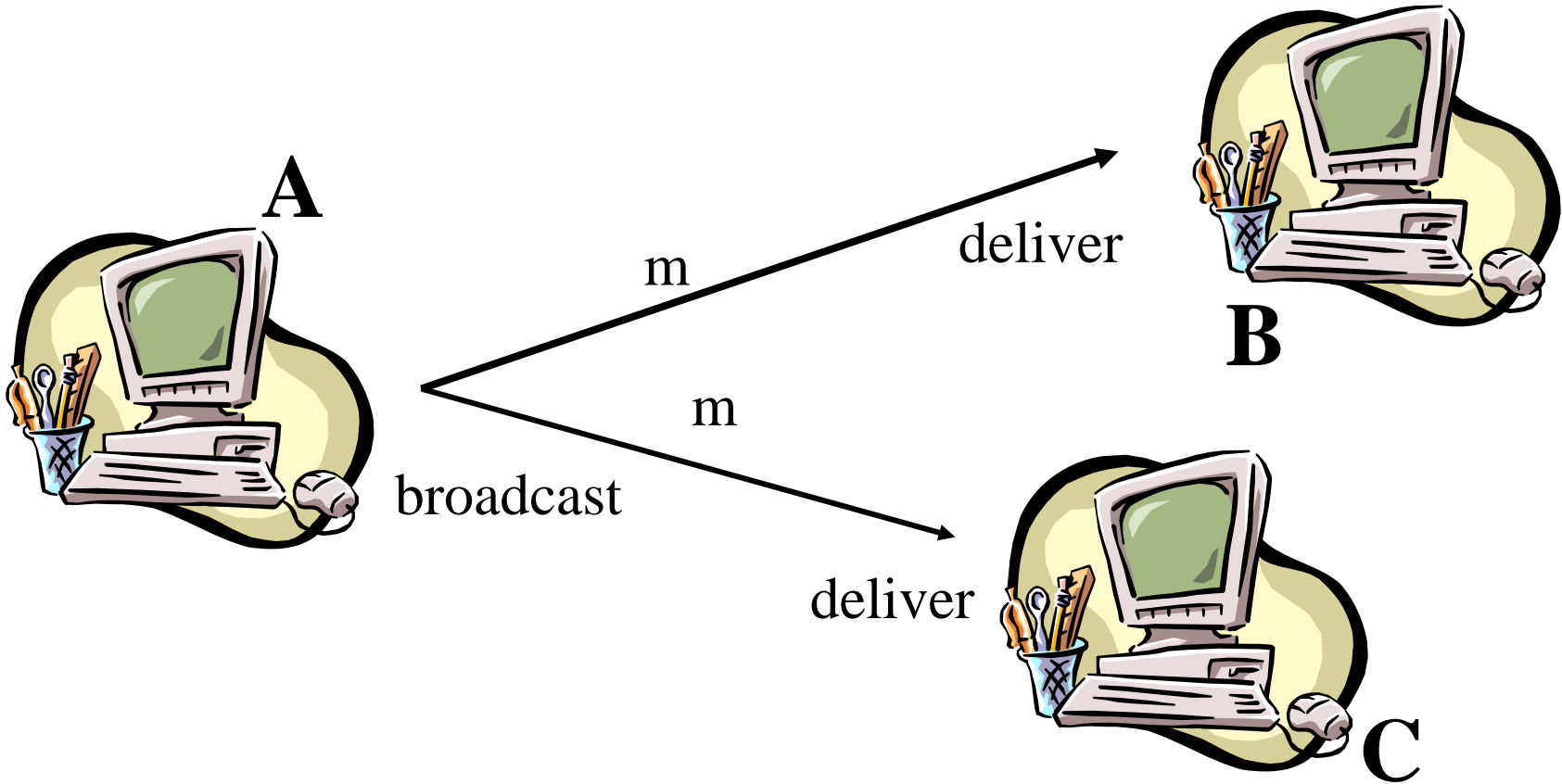


Overview

- ☛ **Intuitions:** what total order broadcast can bring?
- ☛ **Specifications of *total order broadcast***
- ☛ **Consensus-based total order algorithm**



Broadcast



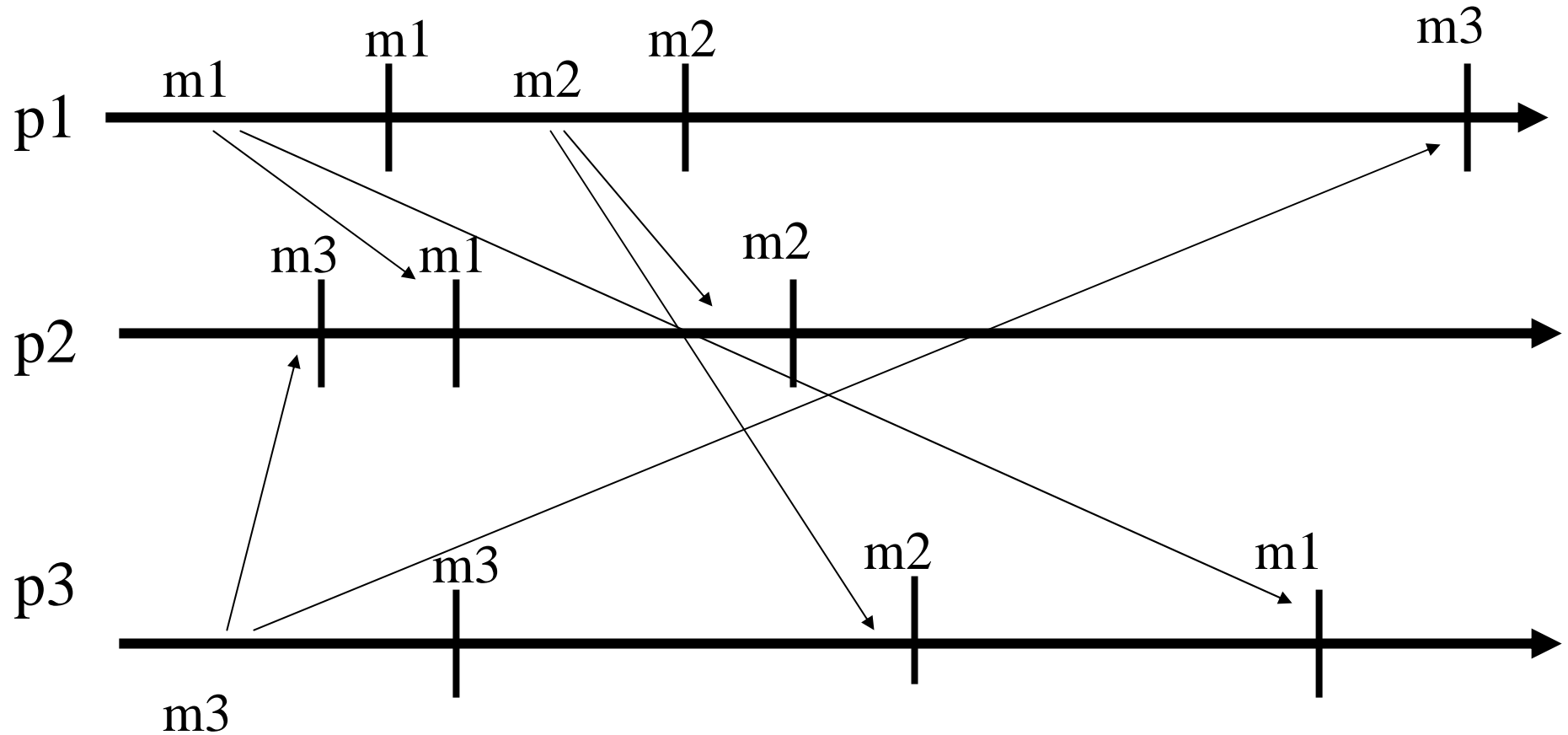


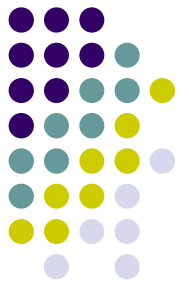
Intuitions (1)

- In ***reliable*** broadcast, the processes are free to deliver messages in any order they wish
- In ***causal*** broadcast, the processes need to deliver messages according to some order (causal order)
- The order imposed by causal broadcast is however partial: some messages might be delivered in different order by the processes

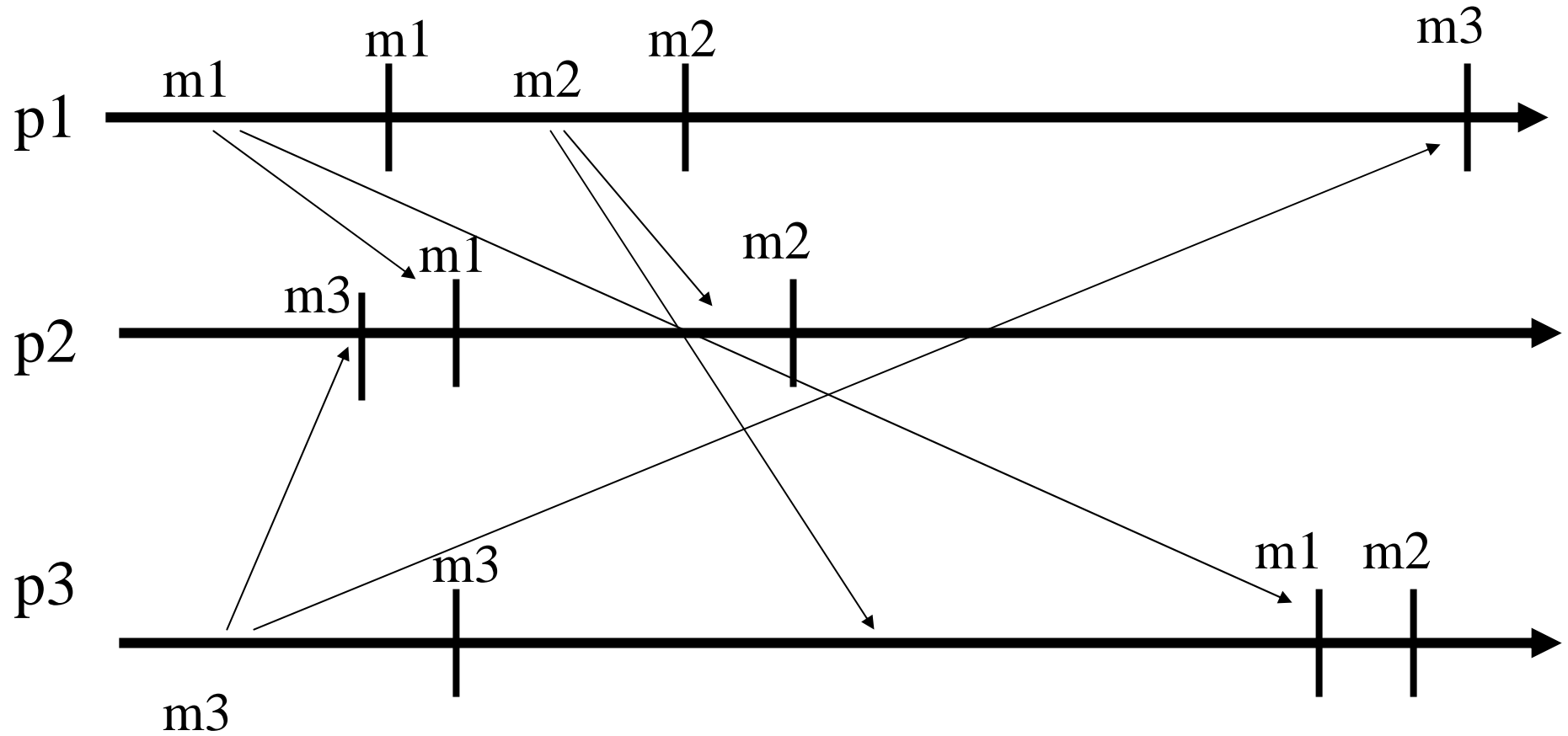


Reliable Broadcast





Causal Broadcast



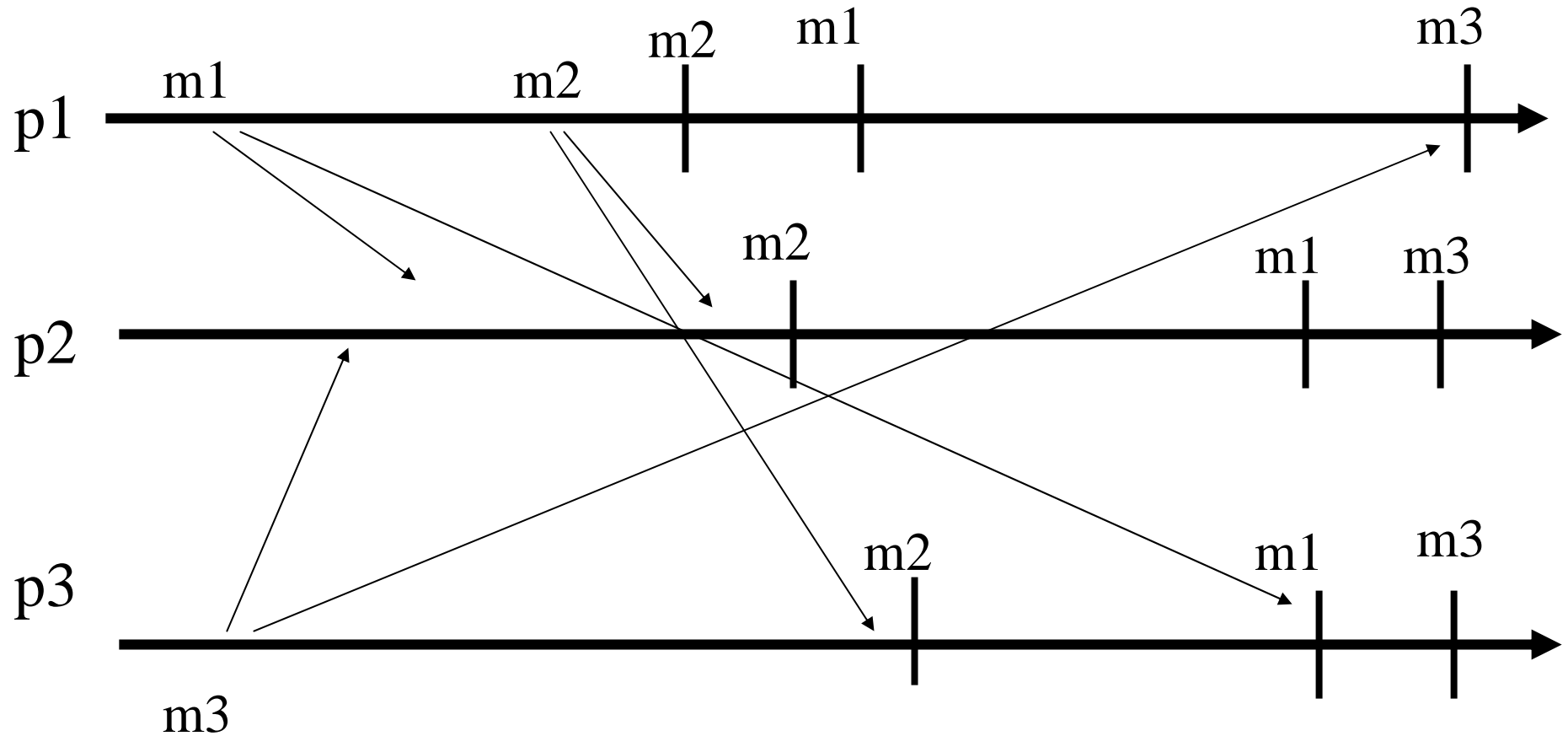


Intuitions (2)

- In *total order* broadcast, the processes must deliver all messages according to the same order (i.e., the order is now total)
- Note that this order does not need to respect causality (or even FIFO ordering)
- Total order broadcast can be made to respect causal (or FIFO) ordering

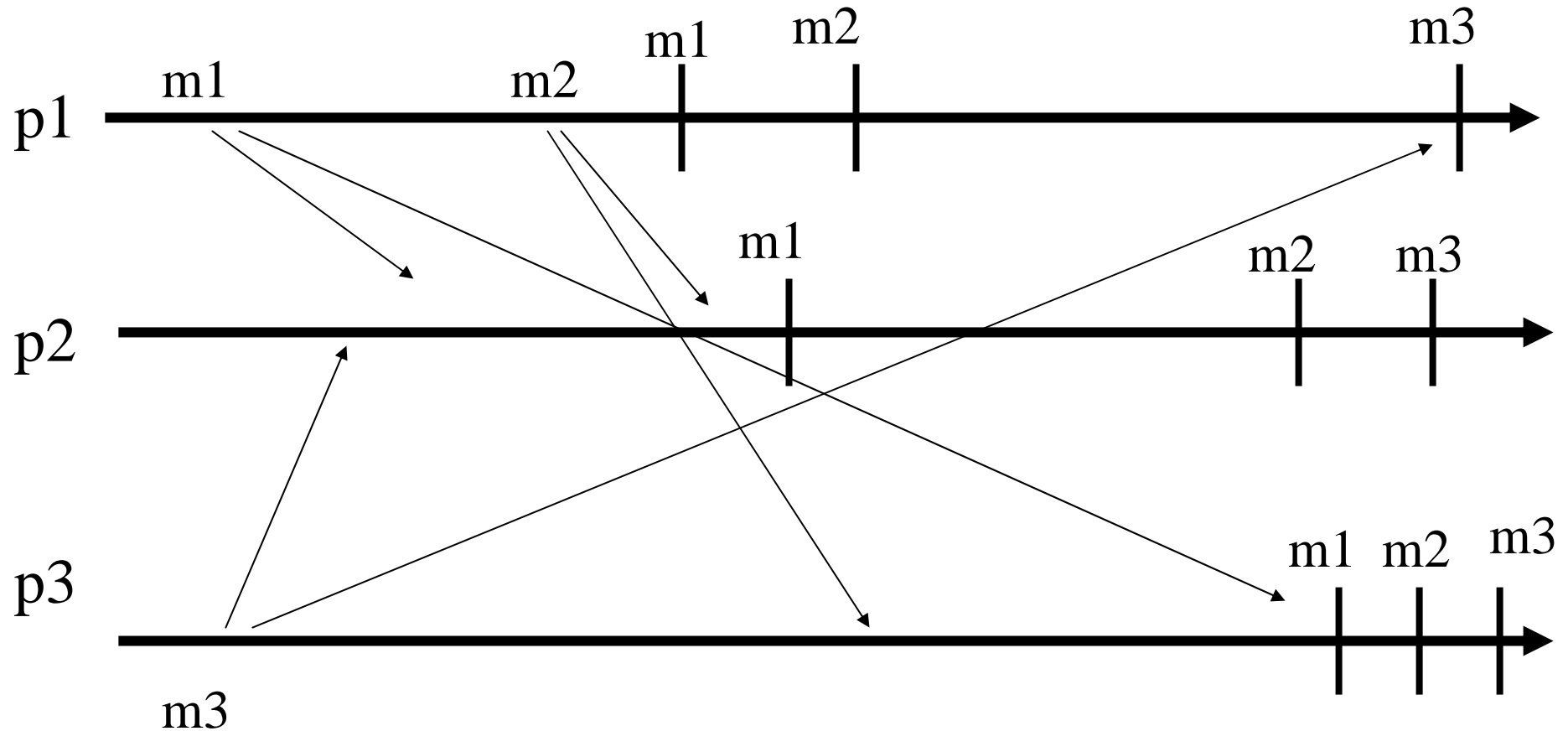


Total Order Broadcast (I)





Total Order Broadcast (II)





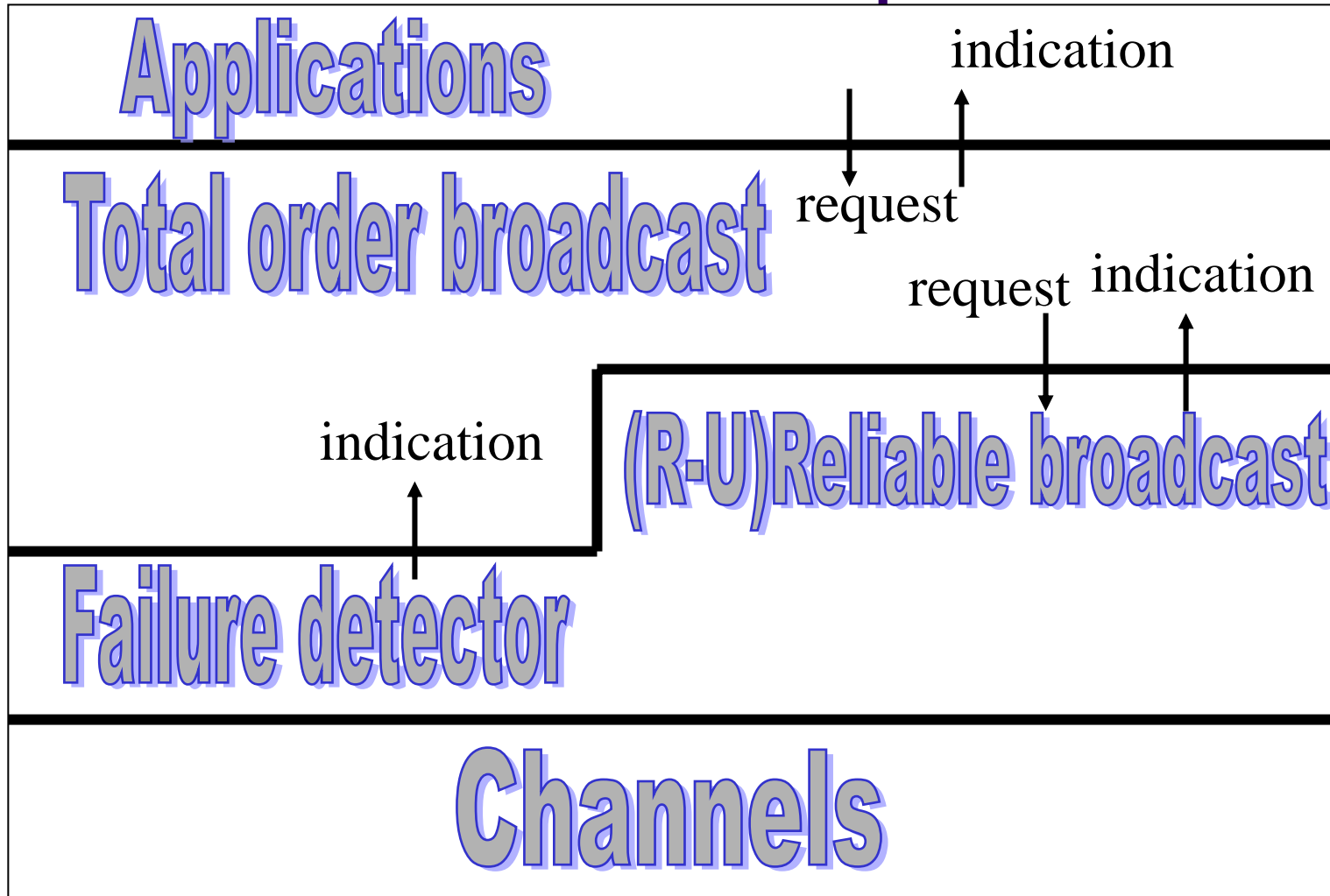
Intuitions (3)

- A replicated service where the replicas need to treat the requests in the ***same order*** to preserve consistency (we talk about state machine replication)

- A notification service where the subscribers need to get notifications in the same order



Modules of a process





Overview

- ☛ **Intuitions:** what total order broadcast can bring?
- ☛ **Specifications of *total order broadcast***
- ☛ **Consensus-based algorithm**



Total order broadcast (tob)

• *Events*

• Request: $\langle \text{toBroadcast}, m \rangle$

• Indication: $\langle \text{toDeliver}, \text{src}, m \rangle$

• *Properties:*

• *RB1, RB2, RB3, RB4*

• *Total order property*



Specification (I)

Validity: If p_i and p_j are correct, then every message broadcast by p_i is eventually delivered by p_j

No duplication: No message is delivered more than once

No creation: No message is delivered unless it was broadcast

(Uniform) Agreement: For any message m . If a correct (any) process delivers m , then every correct process delivers m



Specification (II)

(Uniform) Total order.

Let m and m' be any two messages.

Let p_i be any (correct) process that delivers m without having delivered m'

Then no (correct) process delivers m' before m

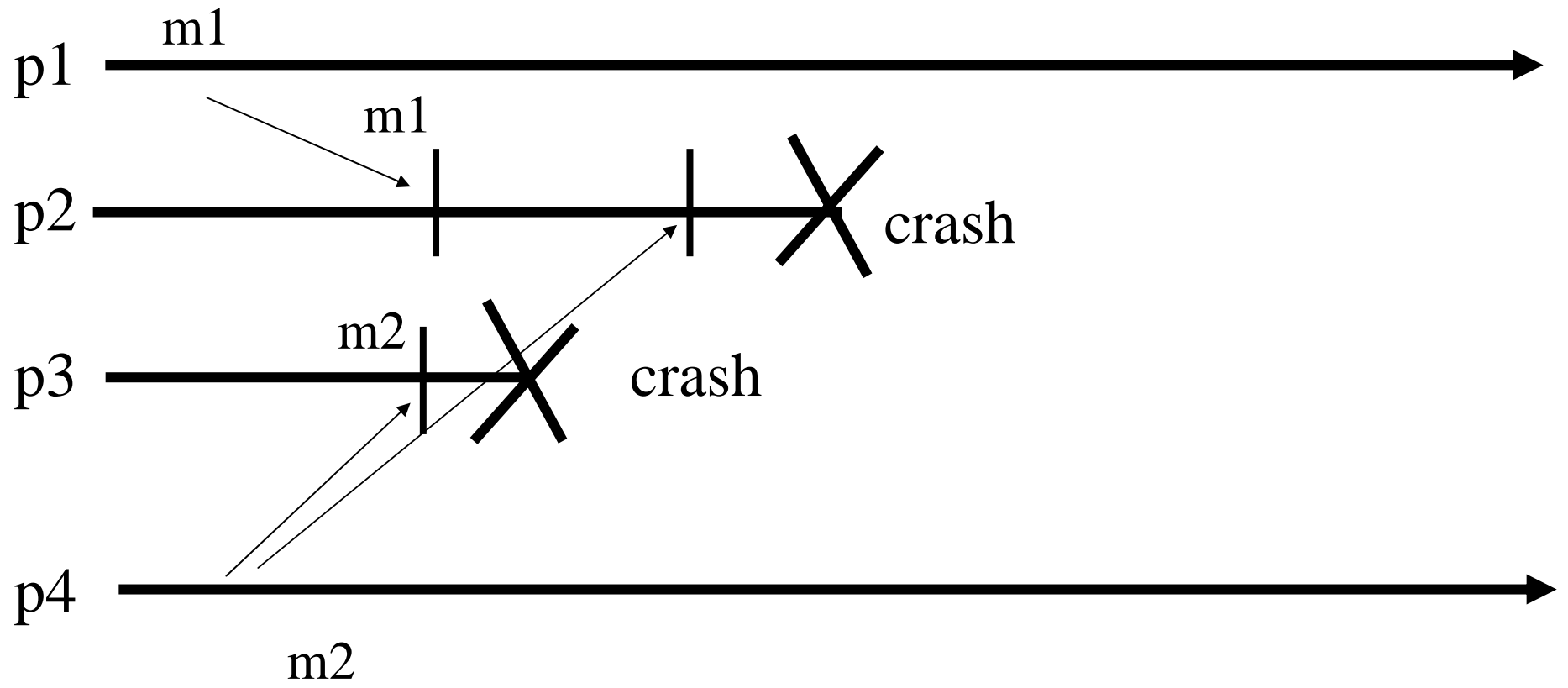
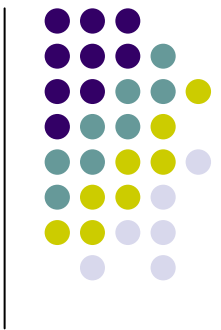


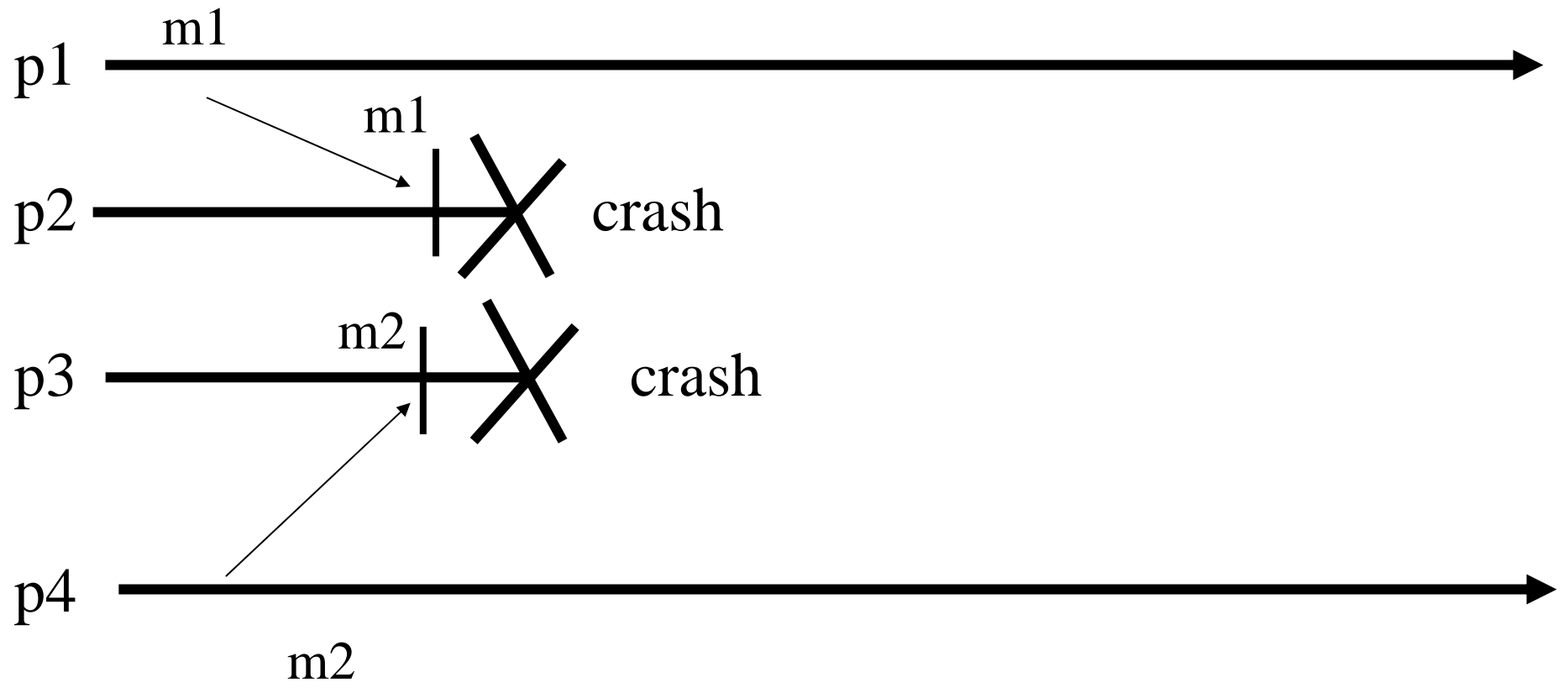
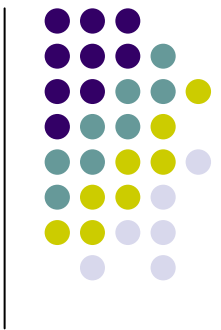
Specifications

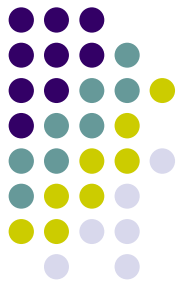
Note the difference with the following properties:

Let p_i and p_j be any two correct (any) processes that deliver two messages m and m' . If p_i delivers m' before m , then p_j delivers m' before m .

Let p_i and p_j be any two (correct) processes that deliver a message m . If p_i delivers a message m' before m , then p_j delivers m' before m .







Overview

- ☛ **Intuitions:** what total order broadcast can bring?
- ☛ **Specifications of *total order broadcast***
- ☛ **Consensus-based algorithm**

(Uniform) Consensus



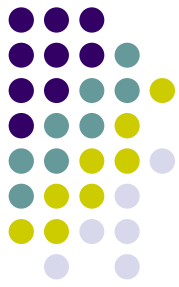
In the (uniform) consensus problem, the processes propose values and need to agree on one among these values

C1. Validity: Any value decided is a value proposed

C2. (Uniform) Agreement: No two correct (any) processes decide differently

C3. Termination: Every correct process eventually decides

C4. Integrity: Every process decides at most once



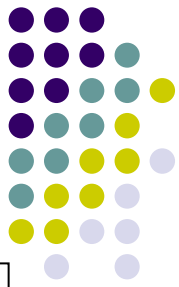
Consensus

• *Events*

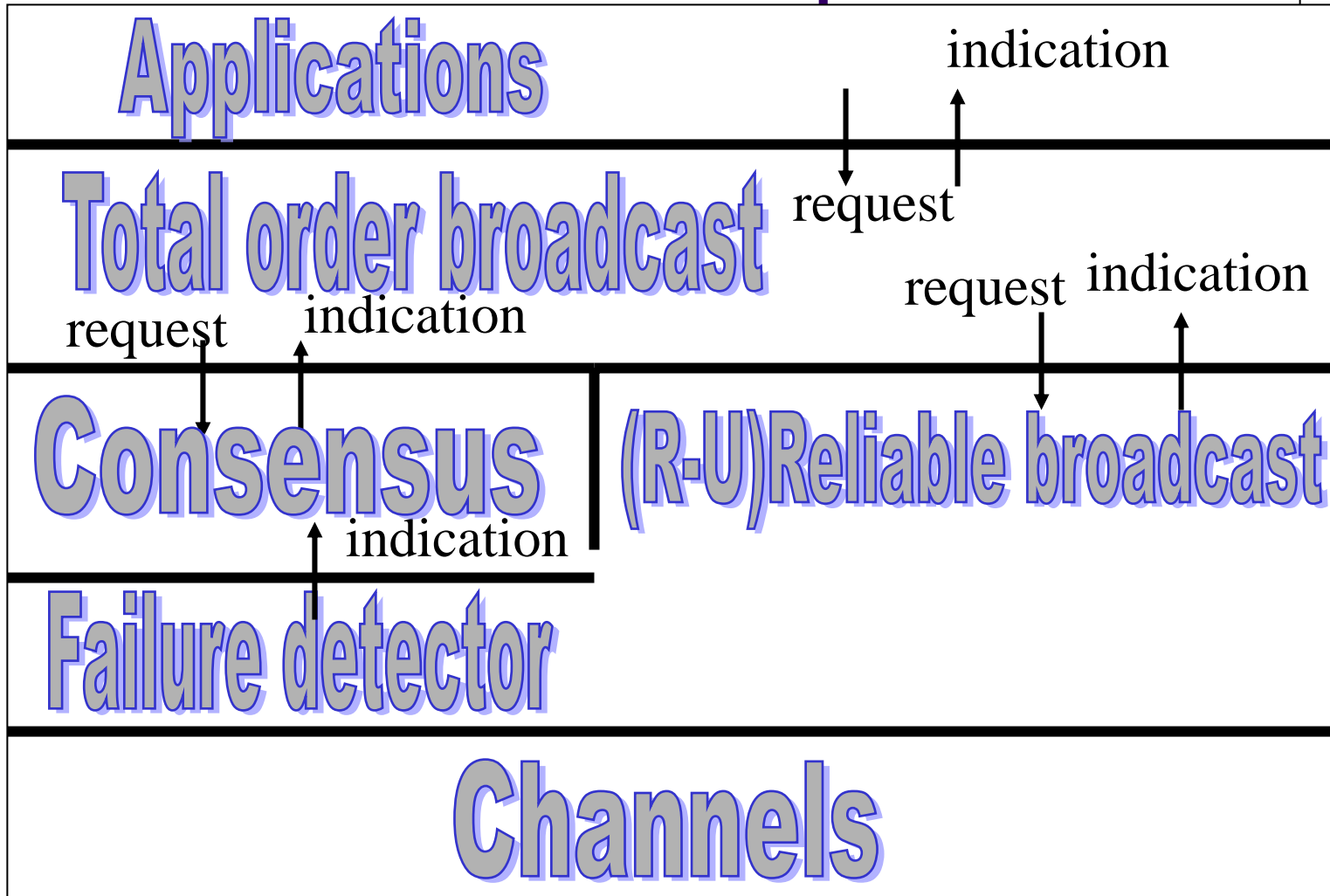
- Request: $\langle \text{Propose}, v \rangle$
- Indication: $\langle \text{Decide}, v' \rangle$

• *Properties:*

- *C1, C2, C3, C4*



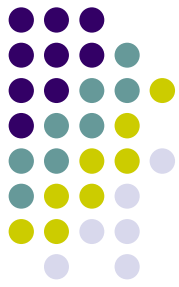
Modules of a process





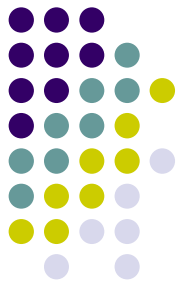
Algorithm

- ☞ **Implements:** TotalOrder (to).
- ☞ **Uses:**
 - ☞ ReliableBroadcast (rb).
 - ☞ Consensus (cons);
- ☞ **upon event** < Init > **do**
 - ☞ unordered = delivered = empty;
 - ☞ wait := false;
 - ☞ sn := 1;



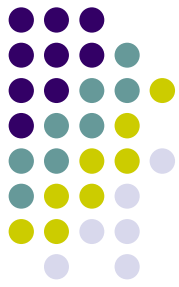
Algorithm (cont'd)

- ☛ **upon event** $\langle \text{toBroadcast}, m \rangle$ **do**
 - ☛ **trigger** $\langle \text{rbBroadcast}, m \rangle$;
- ☛ **upon event** $\langle \text{rbDeliver}, sm, m \rangle$ and (m not in delivered) **do**
 - ☛ $\text{unordered} := \text{unordered} \cup \{(sm, m)\}$;
- ☛ **upon** (unordered not empty) and not(wait) **do**
 - ☛ $\text{wait} := \text{true}$;
 - ☛ **trigger** $\langle \text{Propose}, \text{unordered} \rangle_{sn}$;



Algorithm (cont'd)

- ☛ **upon event** $\langle \text{Decide}, \text{decided} \rangle_{sn}$ **do**
 - ☛ $\text{unordered} := \text{unordered} \setminus \text{decided};$
 - ☛ $\text{ordered} := \text{deterministicSort}(\text{decided});$
 - ☛ for all (sm, m) in ordered:
 - ☛ **trigger** $\langle \text{toDeliver}, sm, m \rangle;$
 - ☛ $\text{delivered} := \text{delivered} \cup \{m\};$
 - ☛ $sn := sn + 1;$
 - ☛ $\text{wait} := \text{false};$

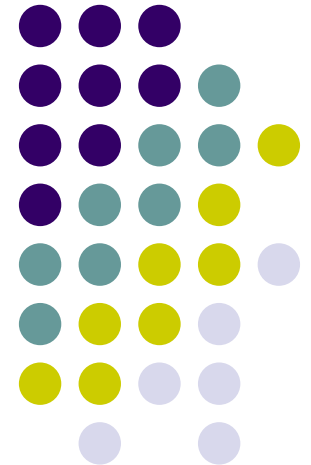


Equivalences

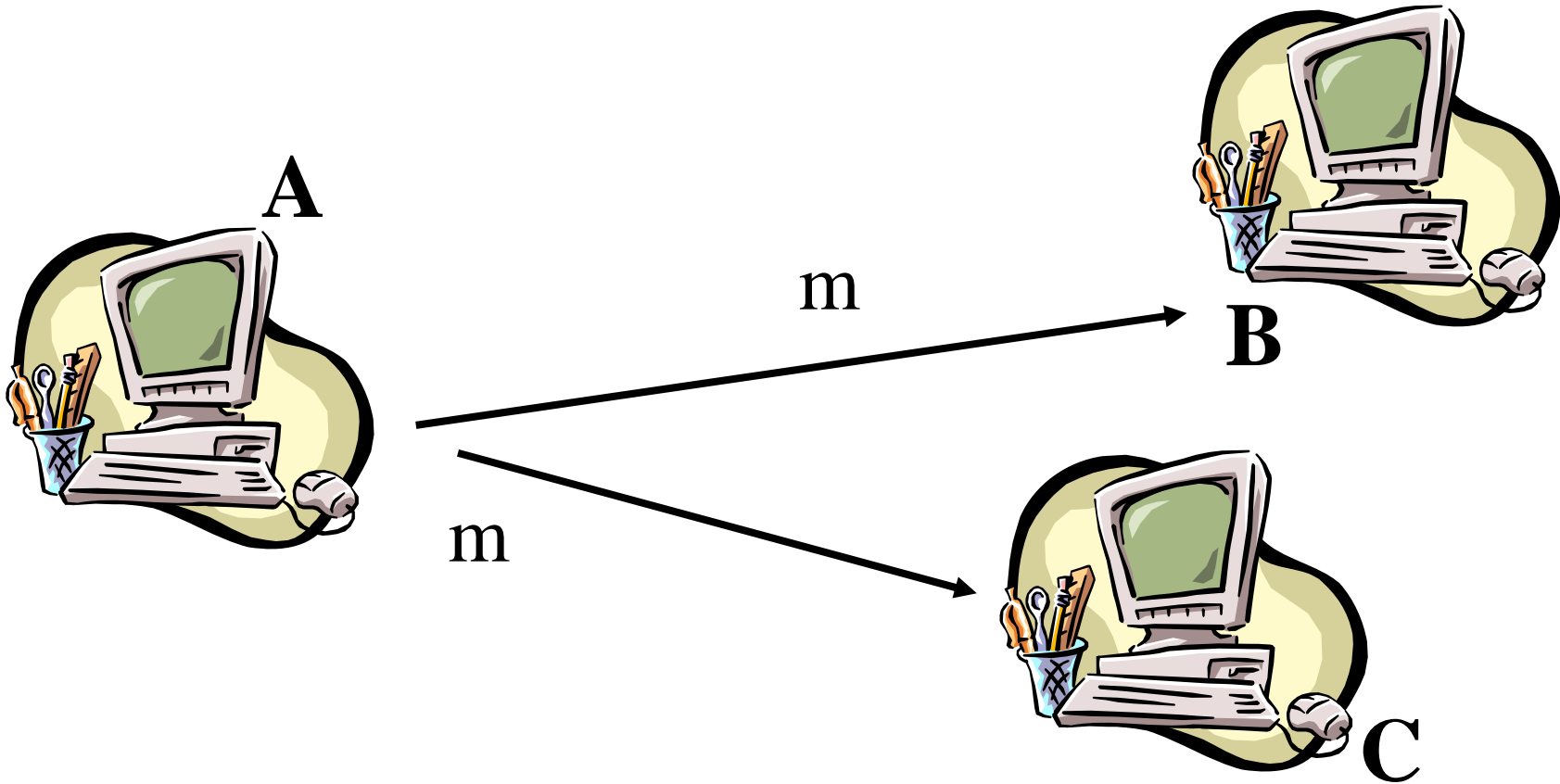
1. One can build consensus with total order broadcast
2. One can build total order broadcast with consensus and reliable broadcast

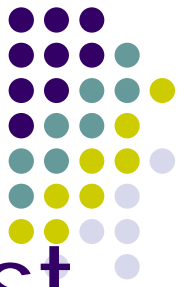
Therefore, consensus and total order broadcast are equivalent problems in a system with reliable channels

Terminating Reliable Broadcast



Terminating Reliable Broadcast



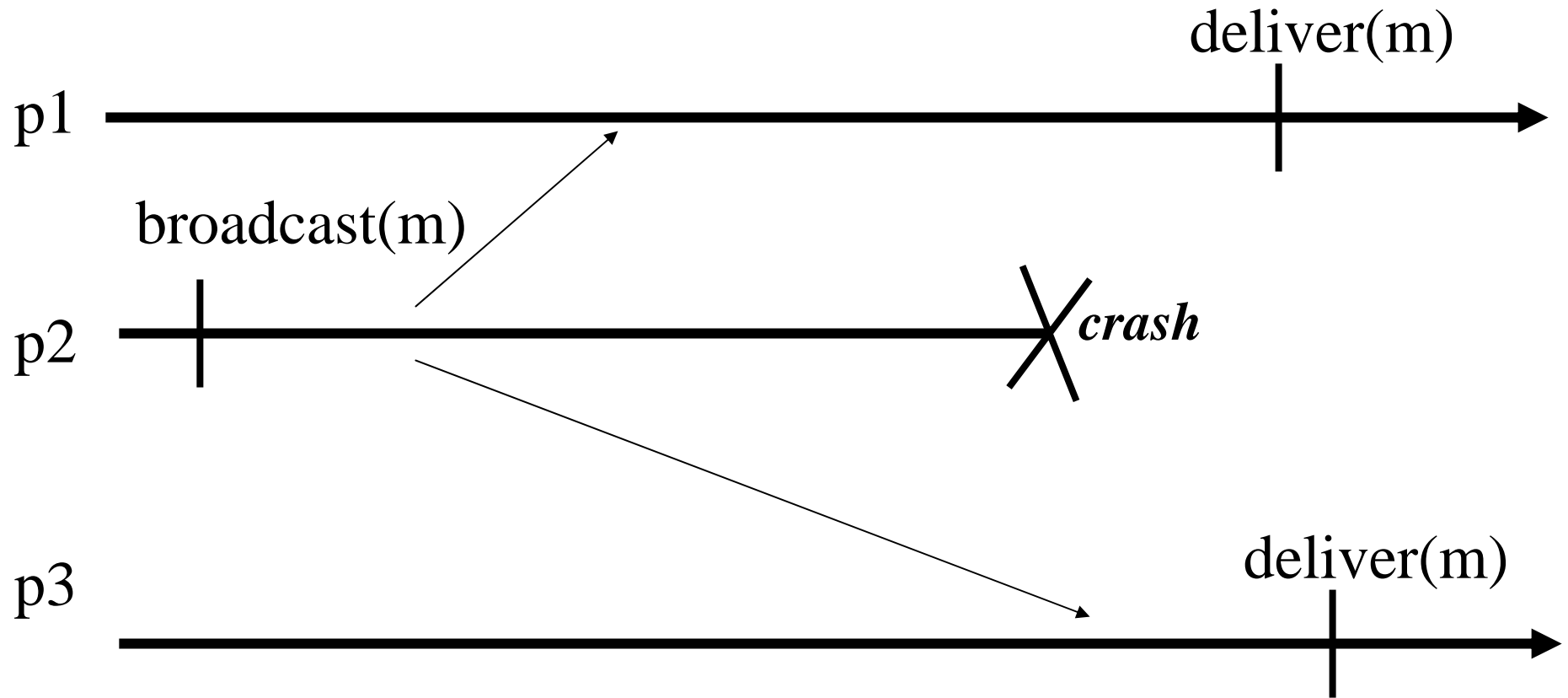


Terminating Reliable Broadcast

- Like reliable broadcast, terminating reliable broadcast (TRB) is a communication primitive used to disseminate a message among a set of processes in a reliable way
- TRB is however strictly stronger than (uniform) reliable broadcast

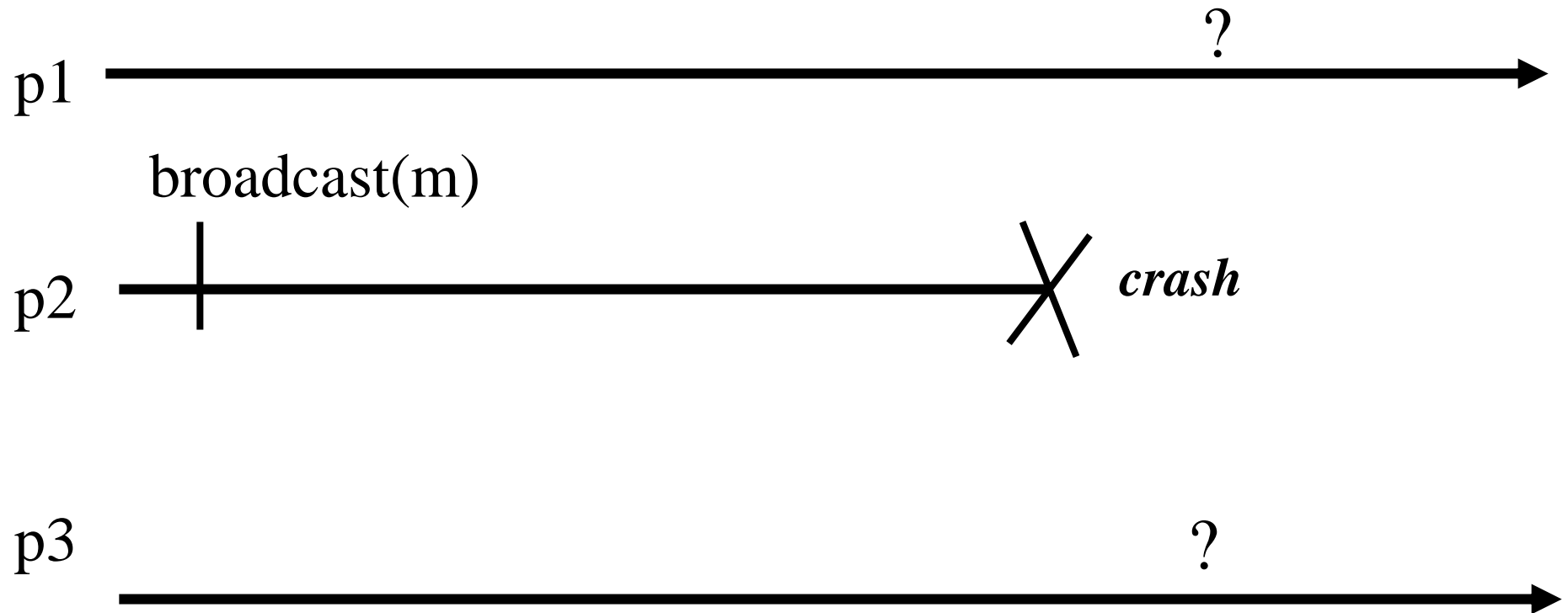


(Uniform) Reliable Broadcast



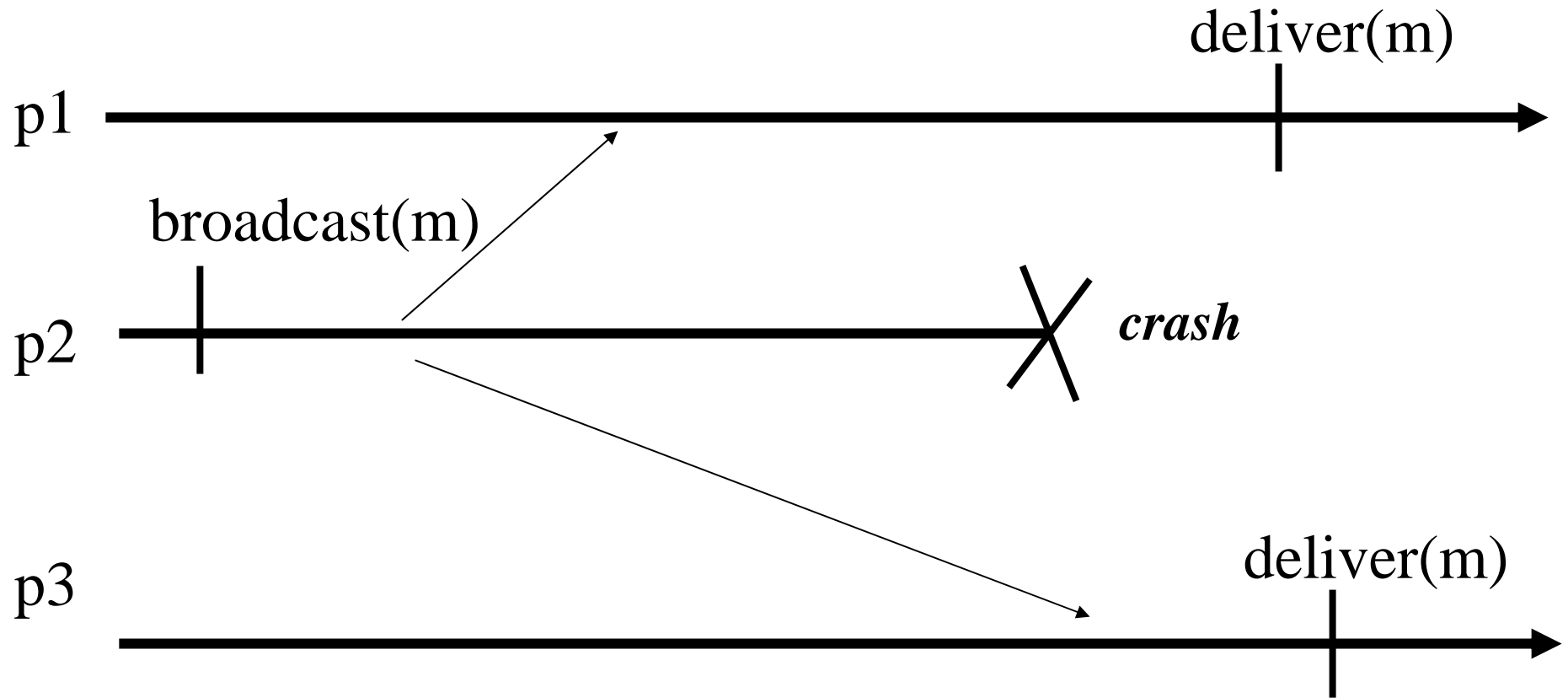


(Uniform) Reliable Broadcast



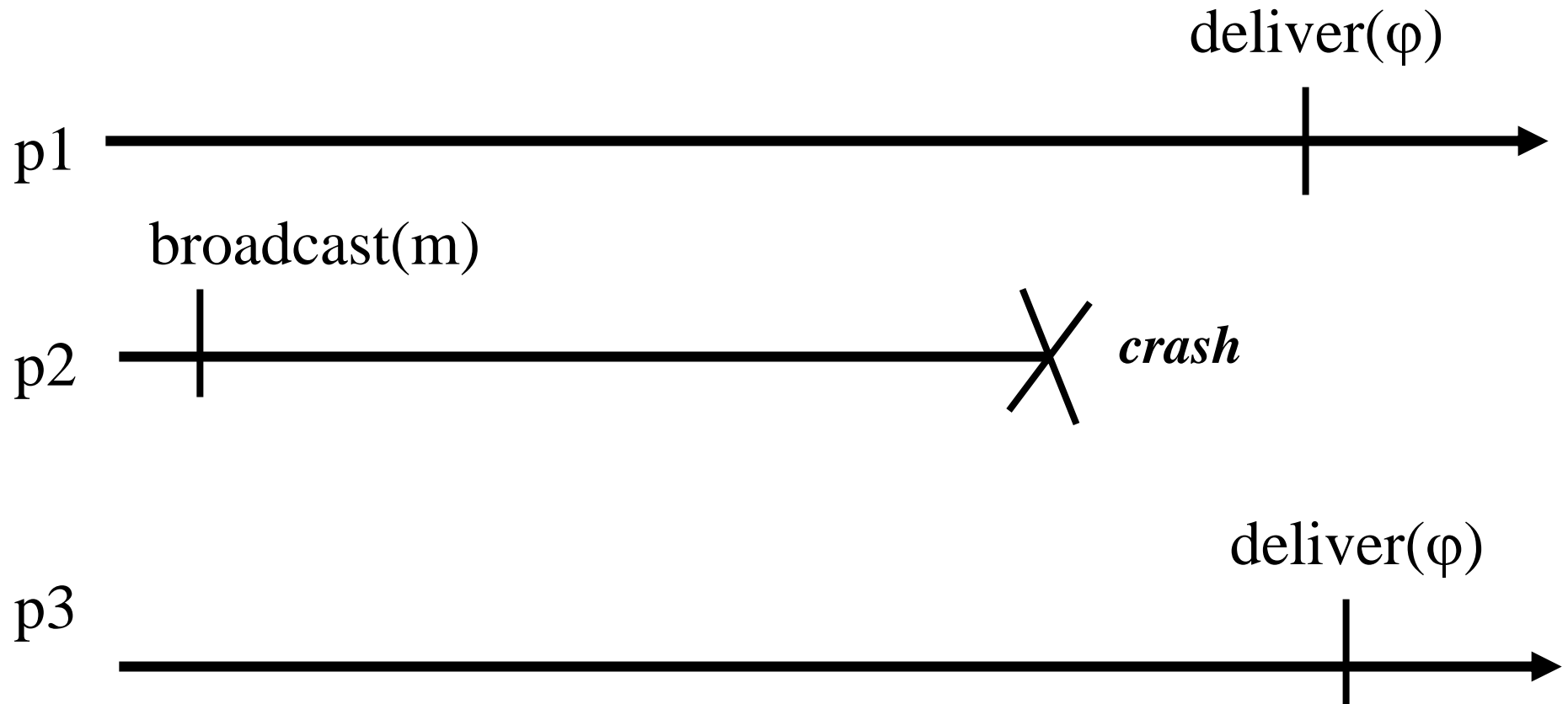


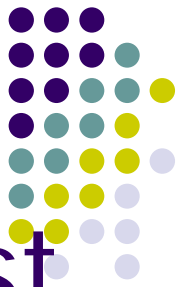
Terminating Reliable Broadcast





Terminating Reliable Broadcast





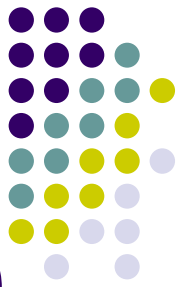
Terminating Reliable Broadcast

- ***Like*** reliable broadcast, correct processes in TRB agree on the set of messages they deliver
- ***Like*** (uniform) reliable broadcast, every correct process in TRB delivers every message delivered by any process
- ***Unlike*** reliable broadcast, every correct process delivers a message, even if the broadcaster crashes



Terminating Reliable Broadcast

- The problem is defined for a specific broadcaster process $p_i = \text{src}$ (known by all processes)
- Process src is supposed to broadcast a message m (distinct from φ)
- The other processes need to deliver m if src is correct but will deliver φ if p_i crashes



Terminating Reliable Broadcast (pi)

TRB1. Integrity: If a process delivers a message m , then either m is ϕ or m was broadcast by src

TRB2. Validity: If the sender src is correct and broadcasts a message m , then src eventually delivers m

TRB3. (Uniform) Agreement: For any message m , if a correct (any) process delivers m , then every correct process delivers m

TRB4. Termination: Every correct process eventually delivers exactly one message



Terminating Reliable Broadcast

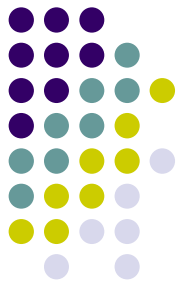
• *Events*

• Request: $\langle \text{trbBroadcast}, m \rangle$

• Indication: $\langle \text{trbDeliver}, p, m \rangle$

• *Properties:*

• ***TRB1, TRB2, TRB3, TRB4***



Algorithm (trb)

- ☛ **Implements:** trbBroadcast (trb).

- ☛ **Uses:**

 - ☛ BestEffortBroadcast (beb).

 - ☛ PerfectFailureDetector (P).

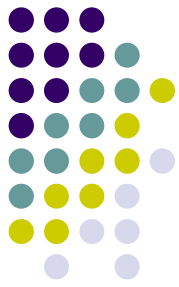
 - ☛ Consensus(cons).

- ☛ **upon event** < Init > **do**

 - ☛ prop := \perp ;

 - ☛ correct := S;

 - ☛ W := false



Algorithm (trb – cont'd)

- **upon event** $\langle \text{trbBroadcast}, m \rangle$ **do**
 - **trigger** $\langle \text{bebBroadcast}, m \rangle$;

- **upon event** $\langle \text{crash}, \text{src} \rangle$ and $(\text{prop} = \perp)$ **do**
 - $\text{prop} := \varphi$;



Algorithm (trb – cont'd)

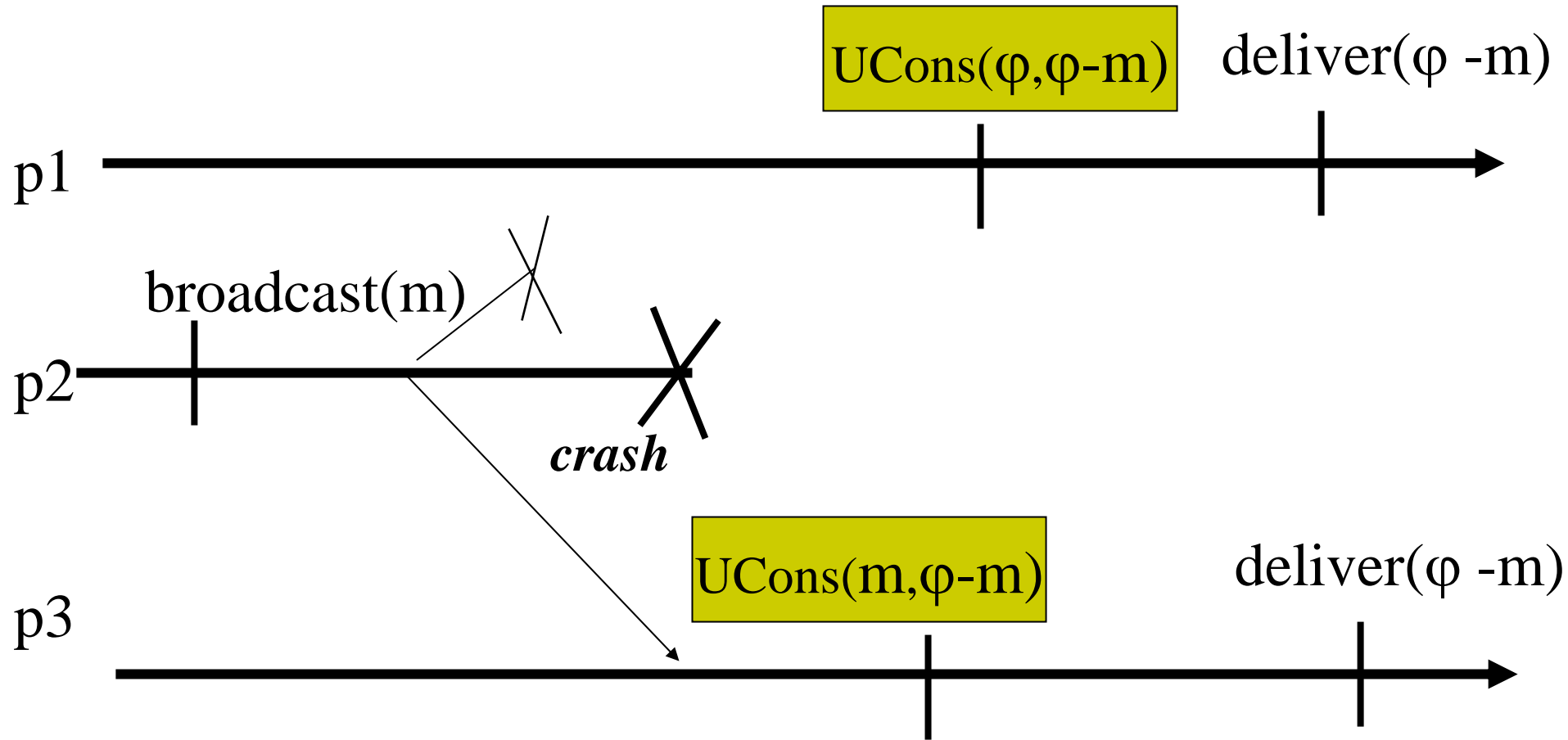
- **upon event** $\langle \text{bebDeliver}, \text{src}, \text{m} \rangle$ and $(\text{prop} = \perp)$ **do**
 - $\text{prop} := \text{m};$

- **upon** $(\text{prop} \neq \perp)$ and $\text{not}(w)$ **do**
 - $w := \text{true}$
 - **trigger** $\langle \text{Propose}, \text{prop} \rangle;$

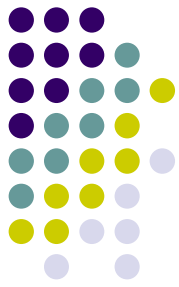
- **upon event** $\langle \text{Decide}, \text{decision} \rangle$ **do**
 - **trigger** $\langle \text{trbDeliver}, \text{src}, \text{decision} \rangle;$



Algorithm (trb); src = p2

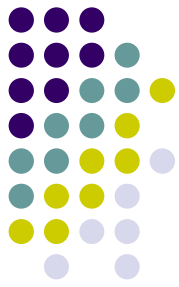


Terminating Reliable Broadcast



- Our TRB algorithm uses the perfect failure detector P (i.e., P is sufficient)
- Is P also necessary?
 - Is there an algorithm that implements TRB with a failure detector that is strictly weaker than P ? (this would mean that P is not necessary)
 - Is there an algorithm that uses TRB to implement P (this would mean that P is necessary)

Terminating Reliable Broadcast



- We give an algorithm that implements **P** using **TRB**; more precisely, we assume that every process p_i can use an infinite number of instances of TRB where p_i is the sender
 - 1. Every process p_i keeps on trbBroadcasting messages $m_{i1}, m_{i2}, \text{ etc}$
 - 2. If a process p_k delivers ϕ_i , p_k suspects p_i
 - NB. The algorithm uses (non-uniform) TRB