



Distributed systems

Total Order Broadcast


Prof R. Guerraoui

Distributed Programming Laboratory

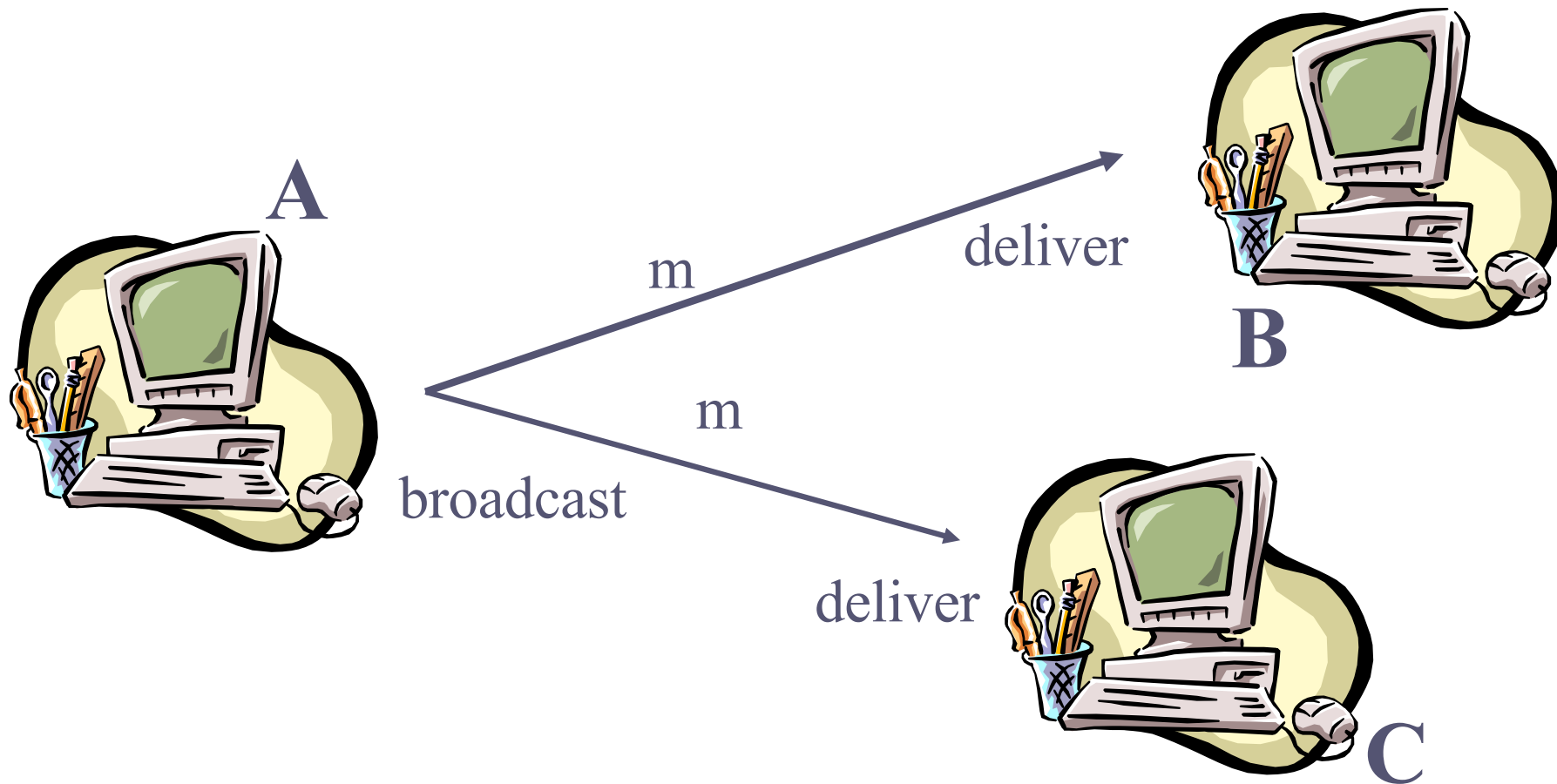




Overview


- **Intuitions:** what is total order broadcast?
 - **Specifications** of *total order broadcast*
 - **Consensus**-based total order algorithm
- 

Broadcast

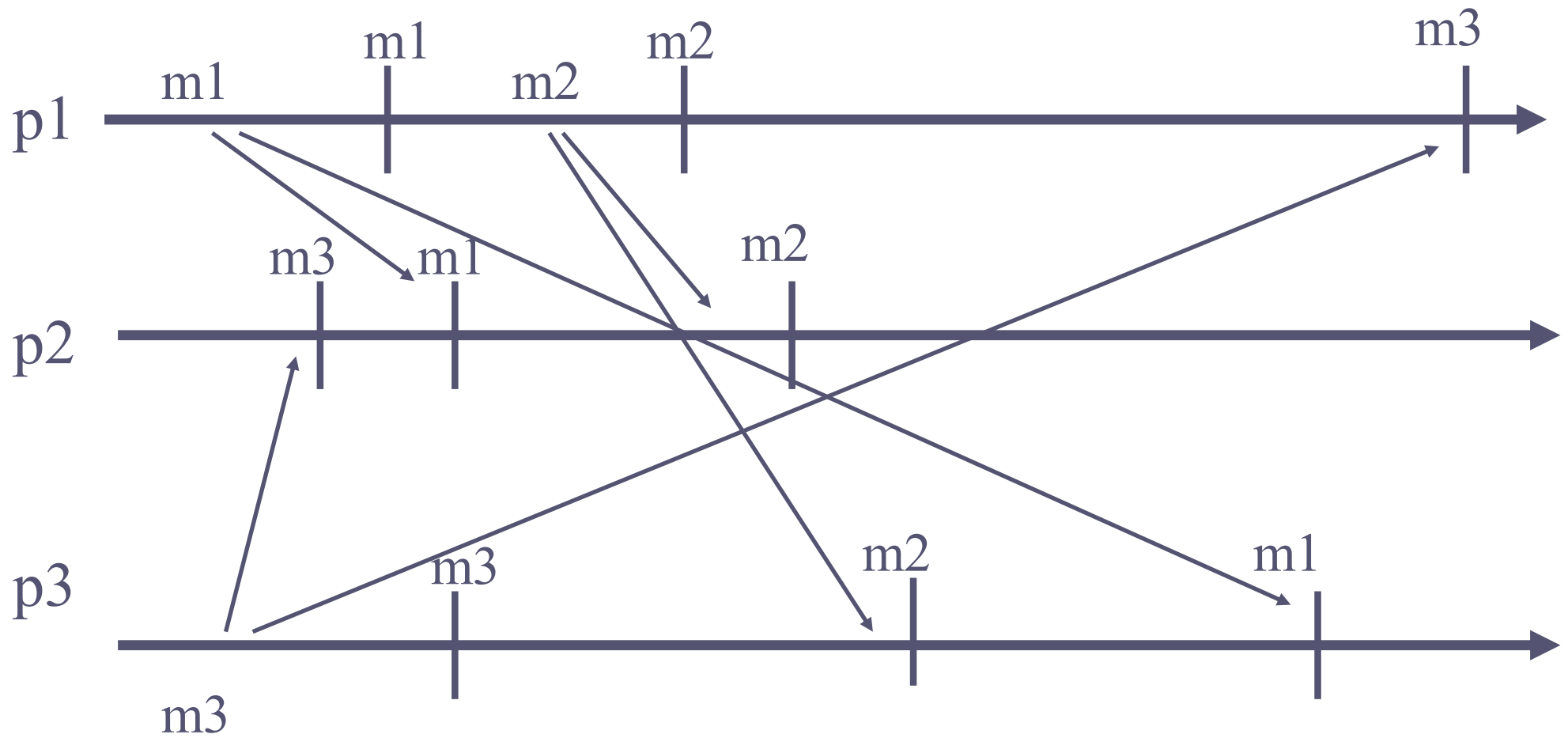




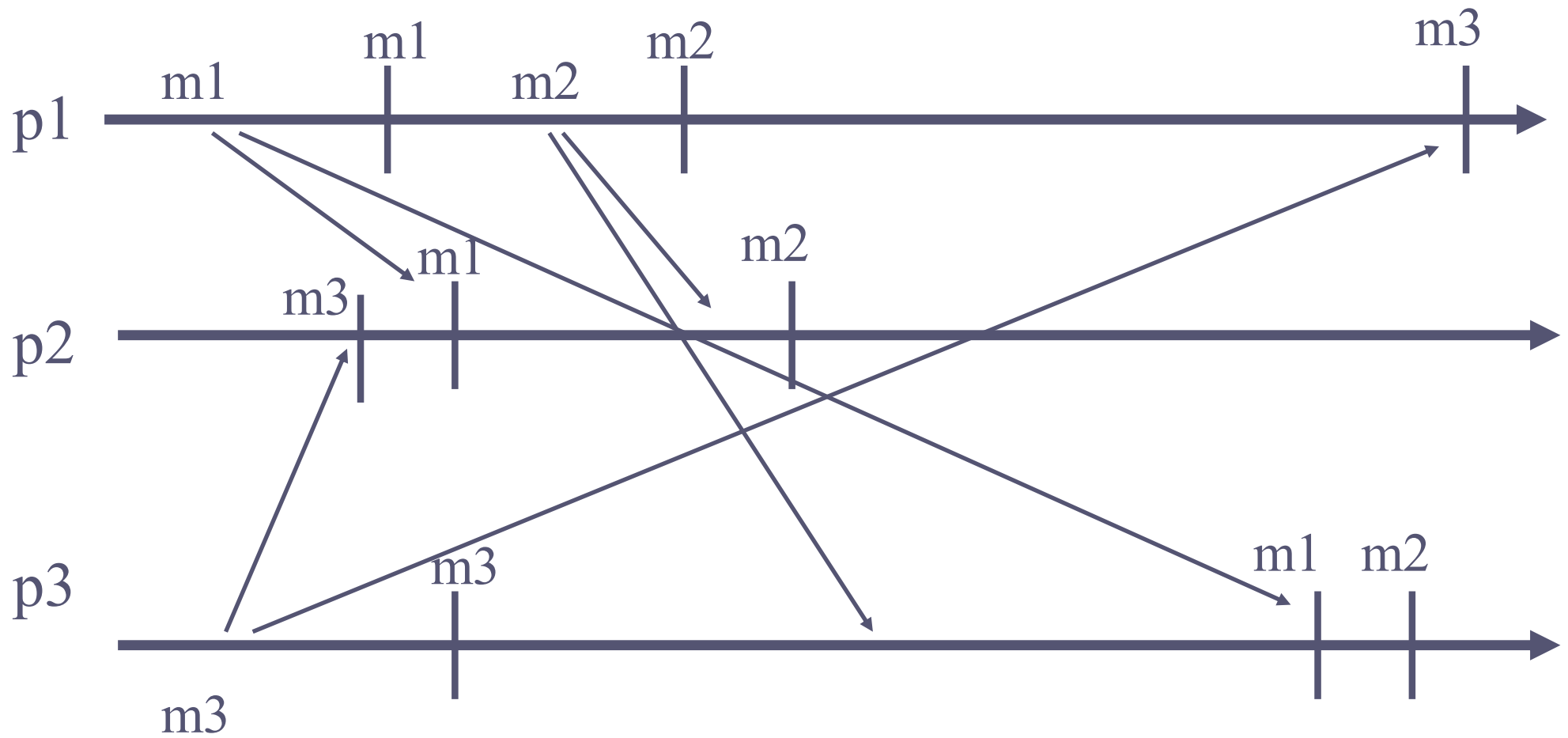
Intuitions (1)

- 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



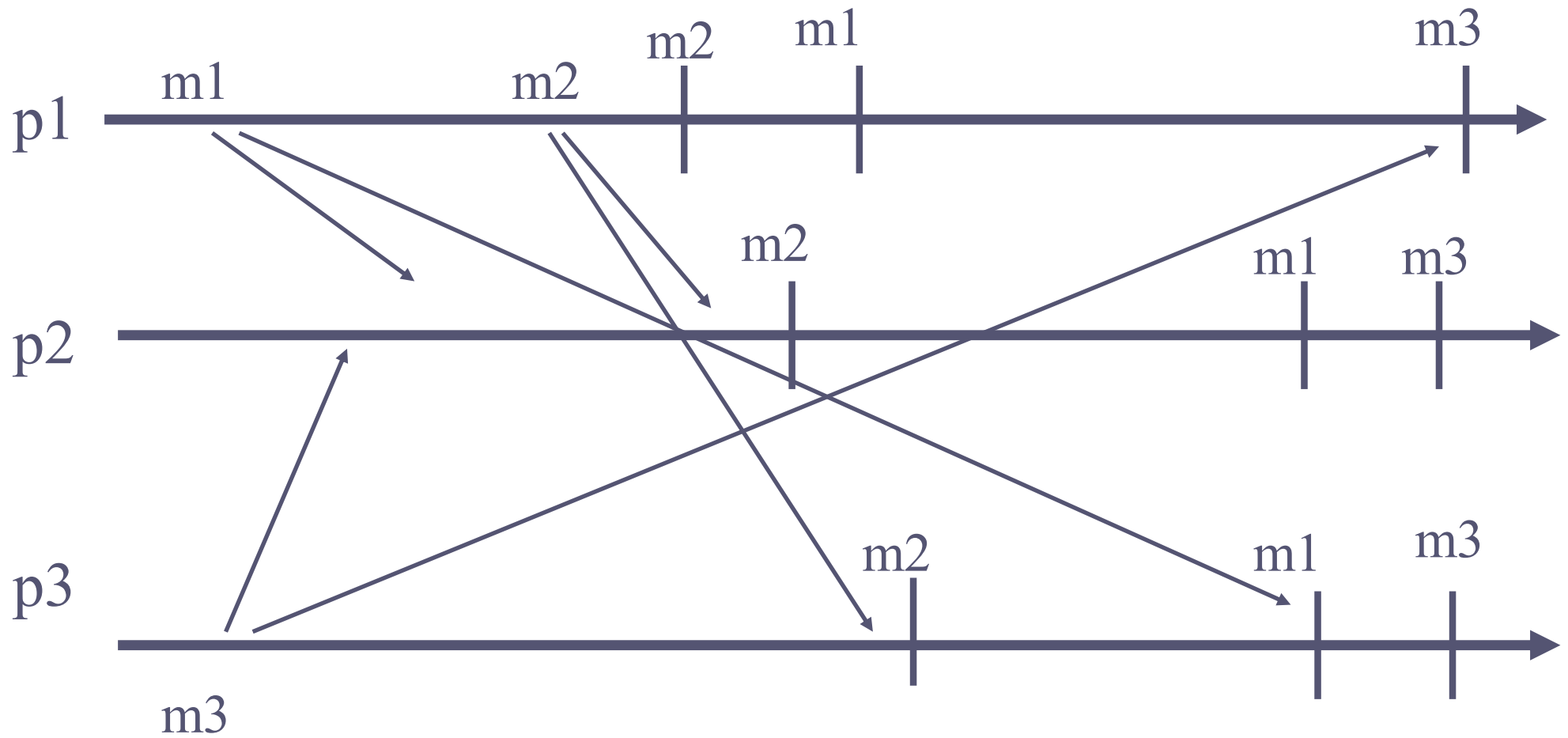
Causal Broadcast



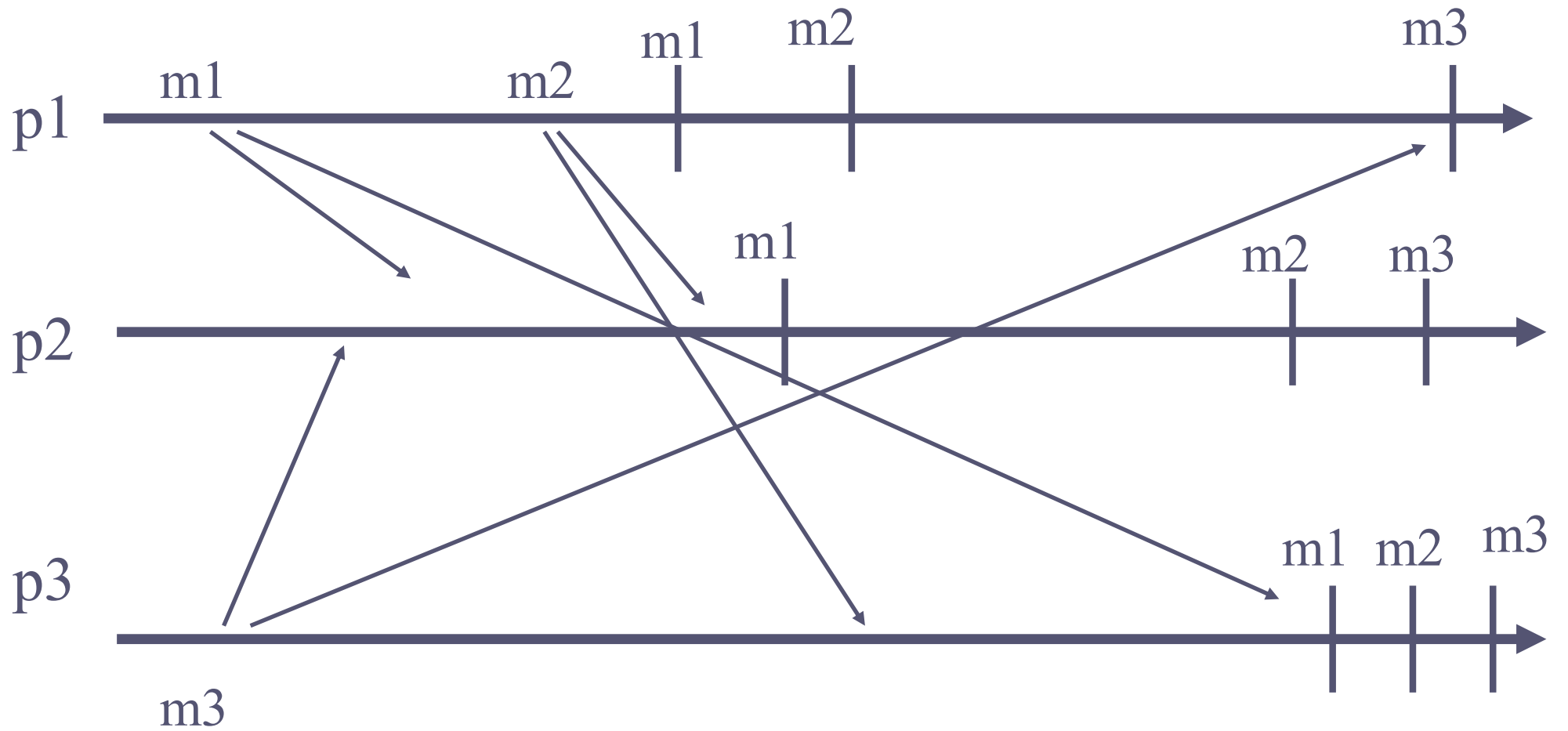
Intuitions (2)

- 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)





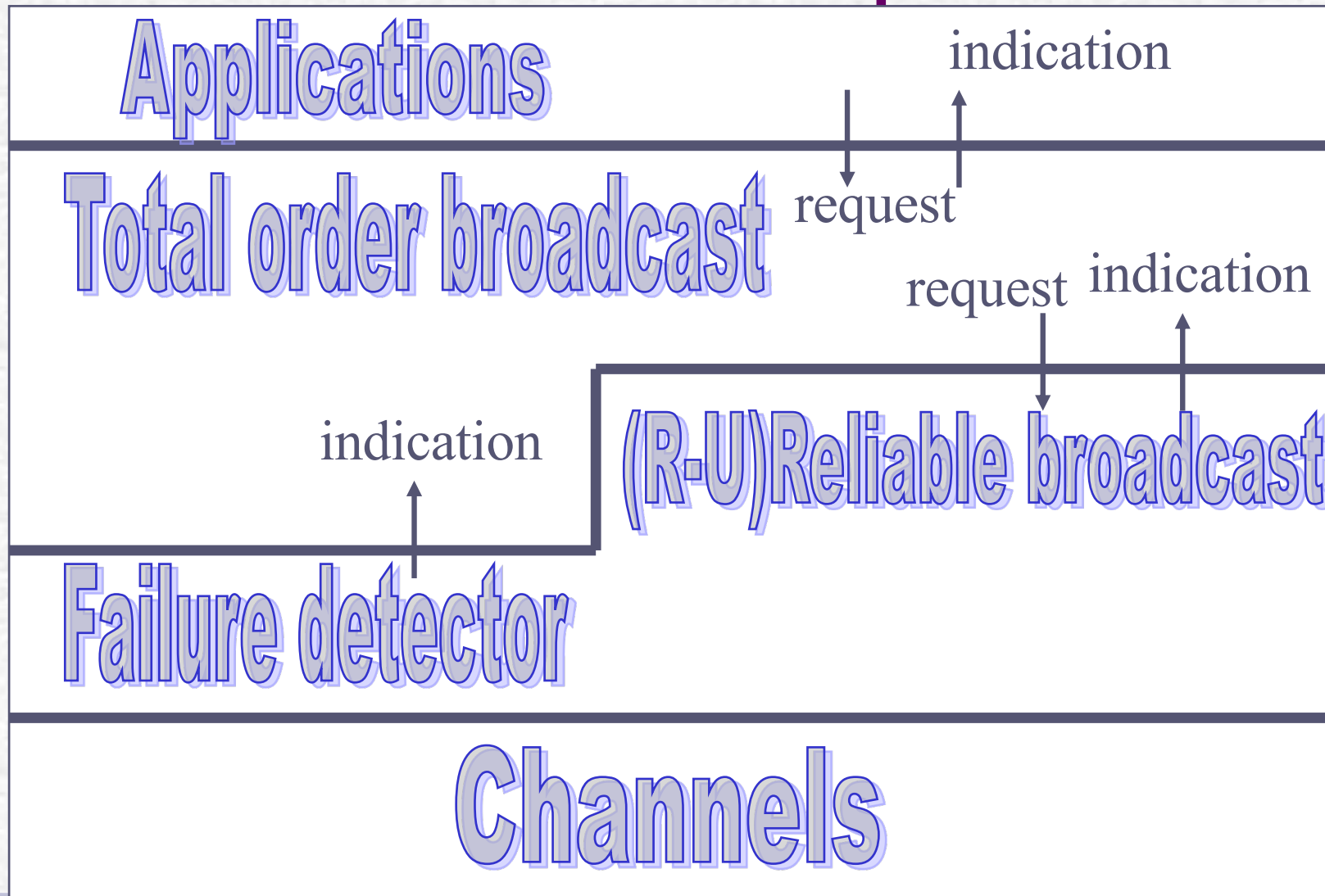
Intuitions (3)

- 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 is total order broadcast?
 - **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






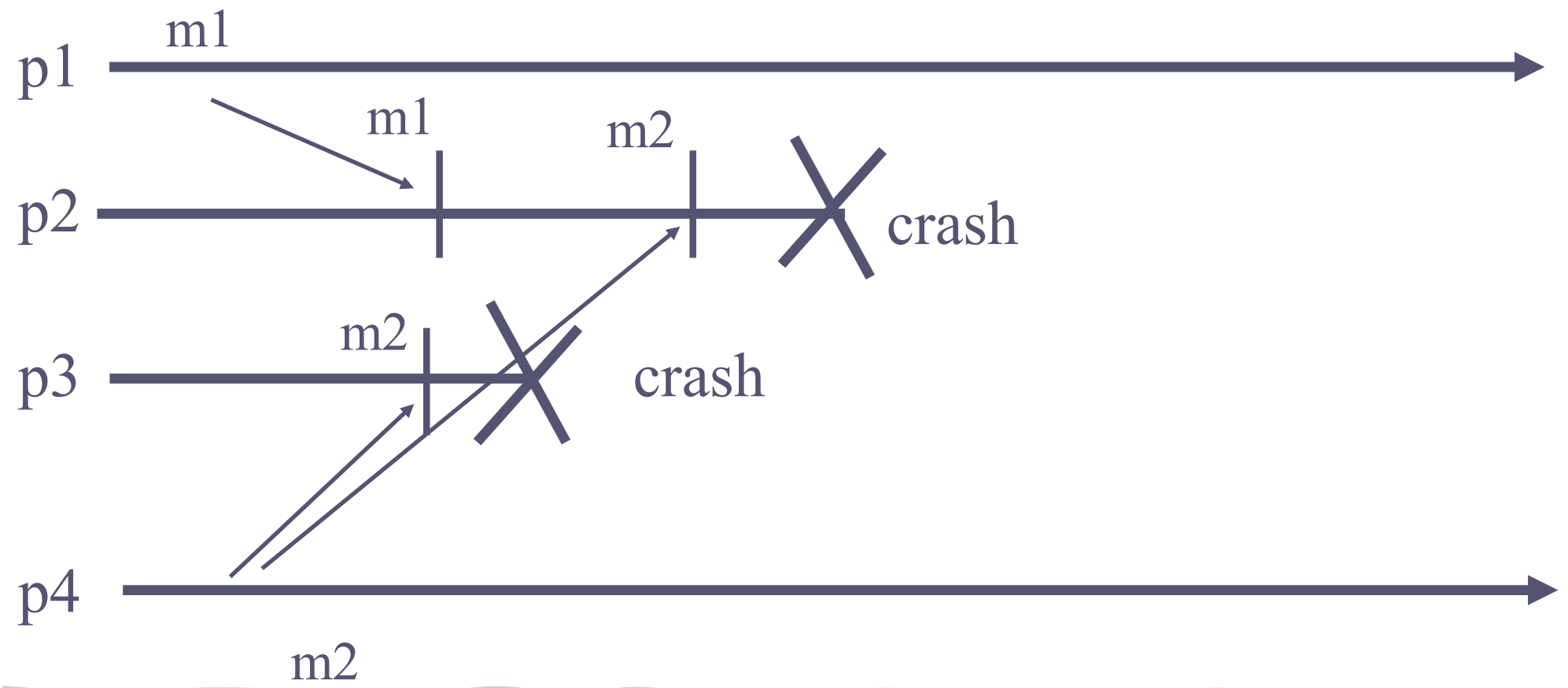
Specifications

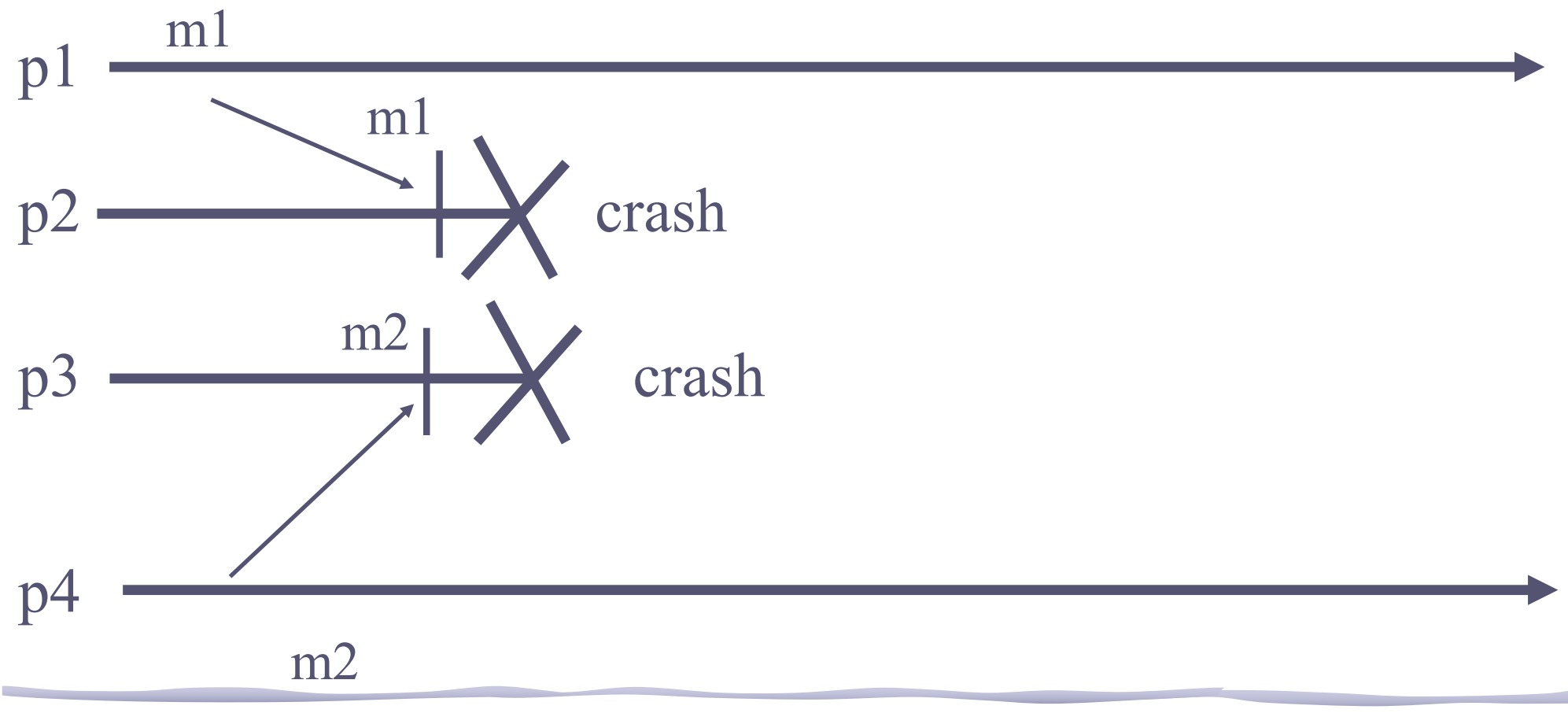
Note the difference with the following properties:

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

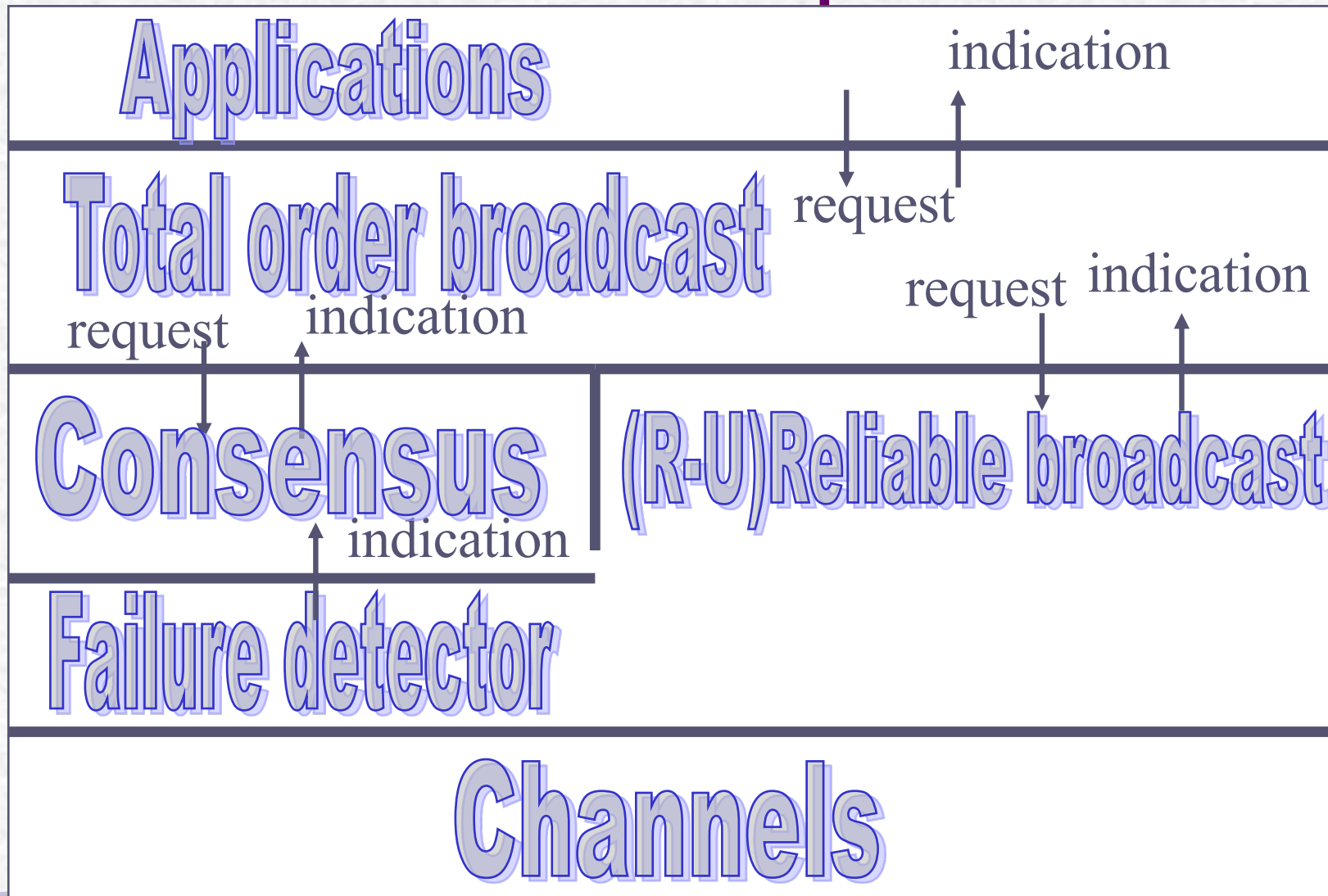
Events

- 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: = \emptyset ;
 - ☞ 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 $(m \notin \text{delivered})$ **do**
 - $\text{unordered} := \text{unordered} \cup \{(sm, m)\}$;
- ☞ **upon** $(\text{unordered} \neq \emptyset)$ and $\text{not}(\text{wait})$ **do**
 - ☞ $\text{wait} := \text{true}$;
 - ☞ **trigger** $\langle \text{Propose}, \text{unordered} \rangle_{sn_i}$;

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

