




Distributed systems

Causal Broadcast

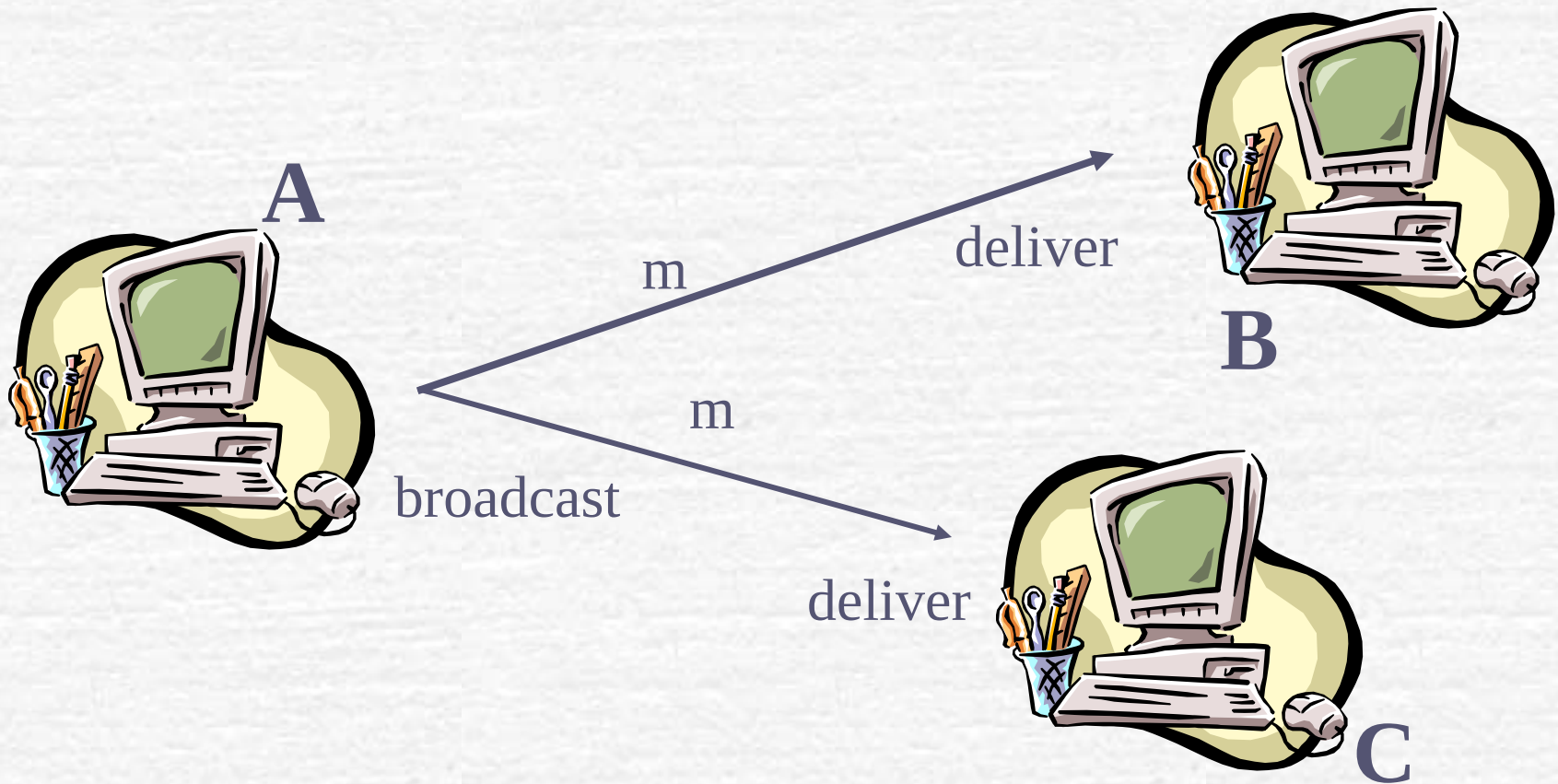
Prof R. Guerraoui
Distributed Programming
Laboratory



Overview

- **Intuitions:** why causal broadcast?
- **Specifications** of ***causal broadcast***
- **Algorithms:**
 - A ***non-blocking*** algorithm using the ***past*** and
 - A ***blocking*** algorithm using ***vector clocks***

Broadcast



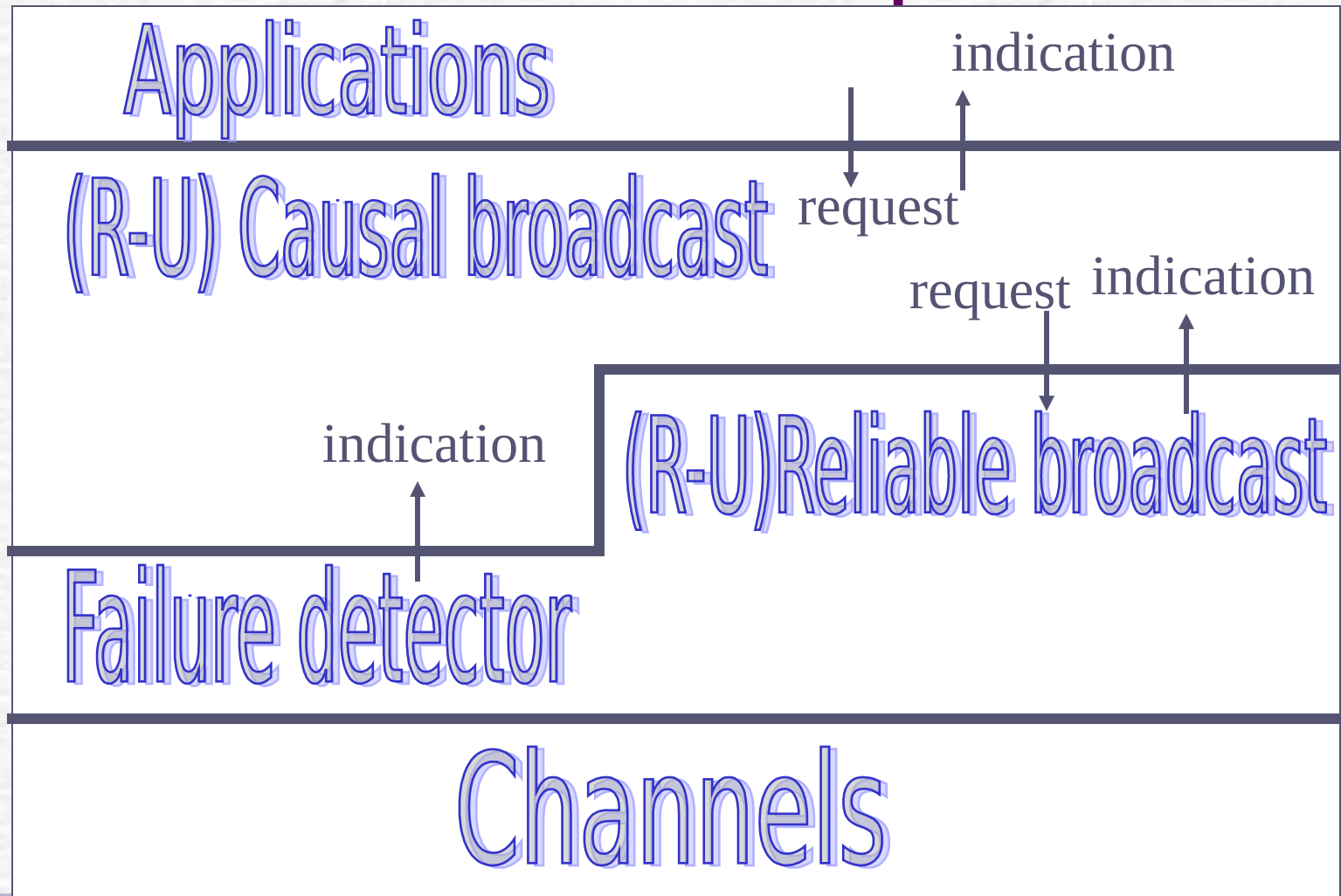
Intuitions (1)

- ✧ So far, we did not consider ordering among messages; In particular, we considered messages to be independent
- ✧ Two messages from the same process might not be delivered in the order they were broadcast
- ✧ A message m_1 that causes a message m_2 might be delivered by some process after m_2

Intuitions (2)

- Consider a system of news where every new event that is displayed in the screen contains a reference to the event that caused it, e.g., a comment on some information includes a reference to the actual information
- Even uniform reliable broadcast does not guarantee such a dependency of delivery
- Causal broadcast alleviates the need for the application to deal with such

Modules of a process



Overview

- ✓ **Intuitions:** why causal broadcast?
- ✓ **Specifications** of ***causal broadcast***
- ✓ **Algorithms:**
 - ✓ A ***non-blocking*** algorithm using the ***past*** and
 - ✓ A ***blocking*** algorithm using ***vector clocks***

Causal broadcast

• **Events**

- Request: $\langle \text{coBroadcast}, m \rangle$
- Indication: $\langle \text{coDeliver}, \text{src}, m \rangle$

- **Property:**

- **Causal Order (CO)**

Causality

- Let $m1$ and $m2$ be any two messages: $m1 \rightarrow m2$ ($m1$ causally precedes $m2$) iff
 - C1 (FIFO order).** Some process p_i broadcasts $m1$ before broadcasting $m2$
 - C2 (Local order).** Some process p_i delivers $m1$ and then broadcasts $m2$
 - C3 (Transitivity).** There is a message $m3$ such that $m1 \rightarrow m3$ and $m3 \rightarrow m2$

Causal broadcast

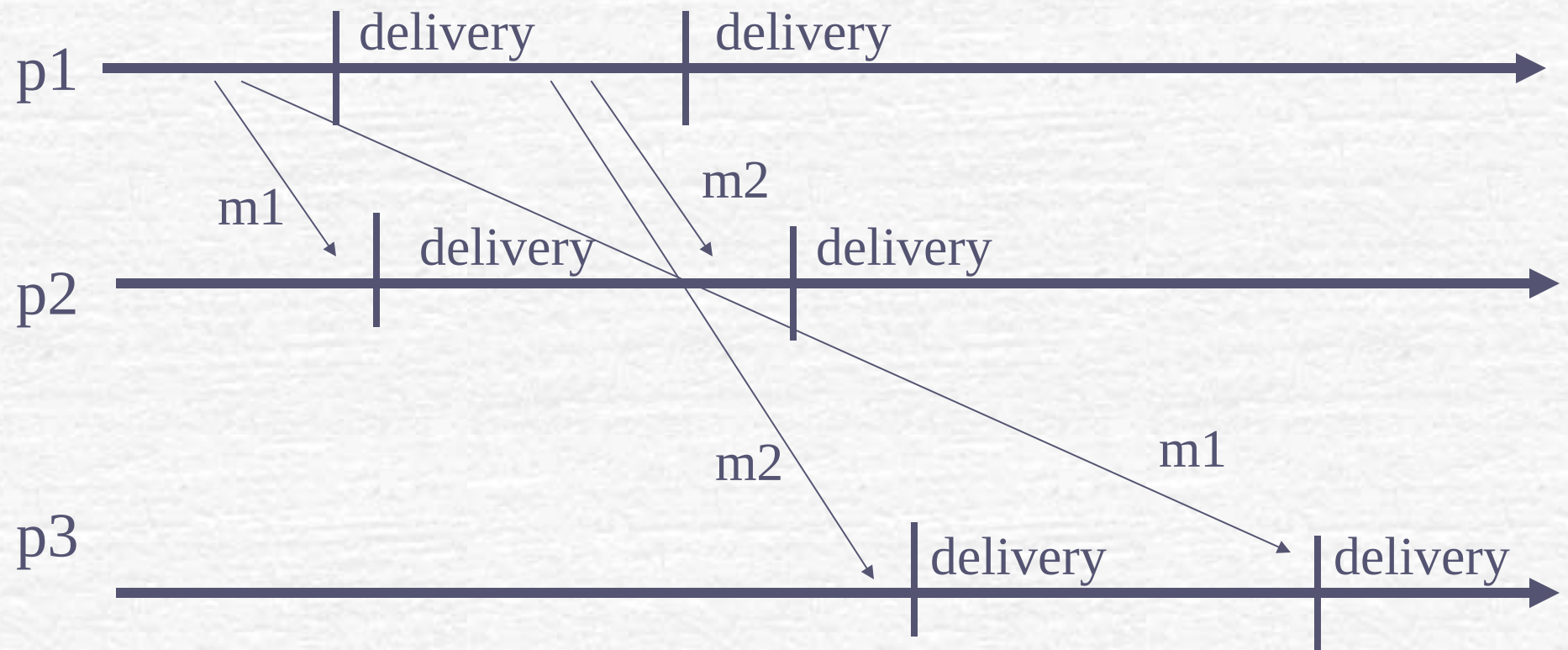
Events

- Request: $\langle \text{coBroadcast}, m \rangle$
- Indication: $\langle \text{coDeliver}, \text{src}, m \rangle$

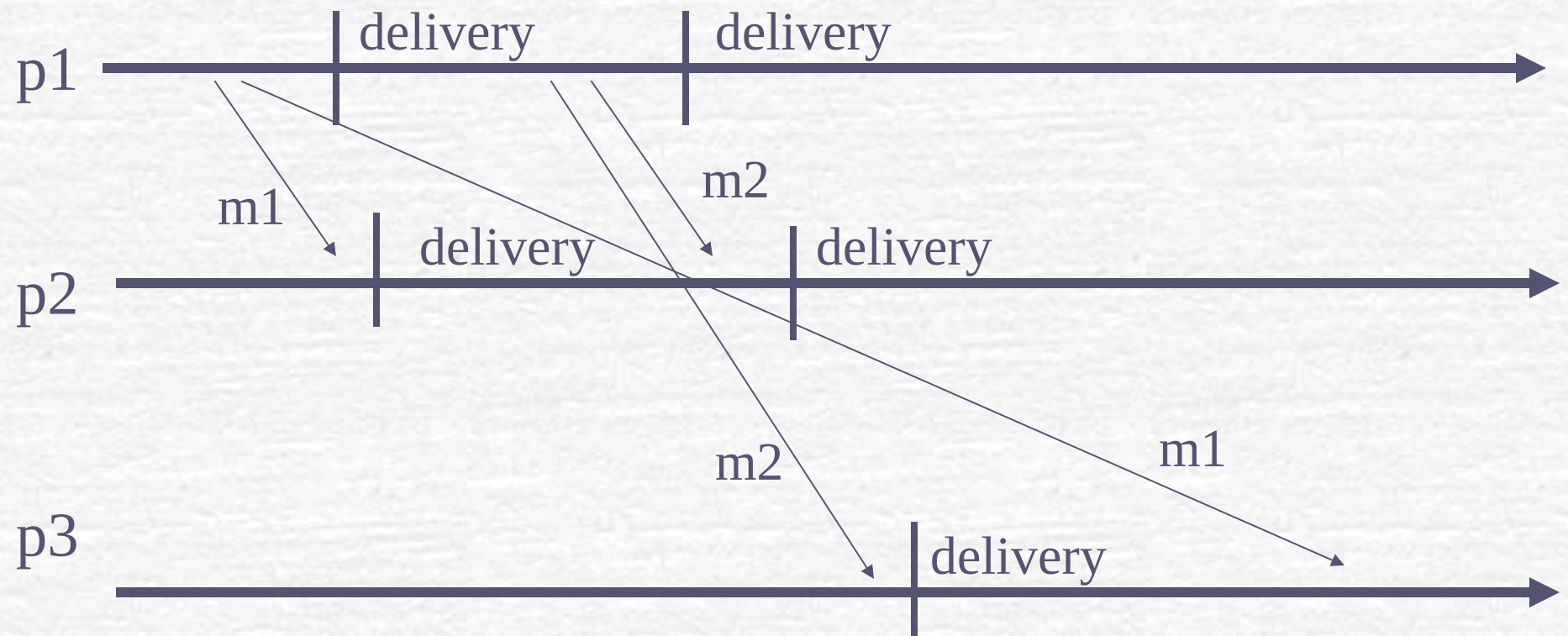
- **Property:**

- **CO:** If any process p_i delivers a message m_2 , then p_i must have delivered every message m_1 such that $m_1 \rightarrow m_2$

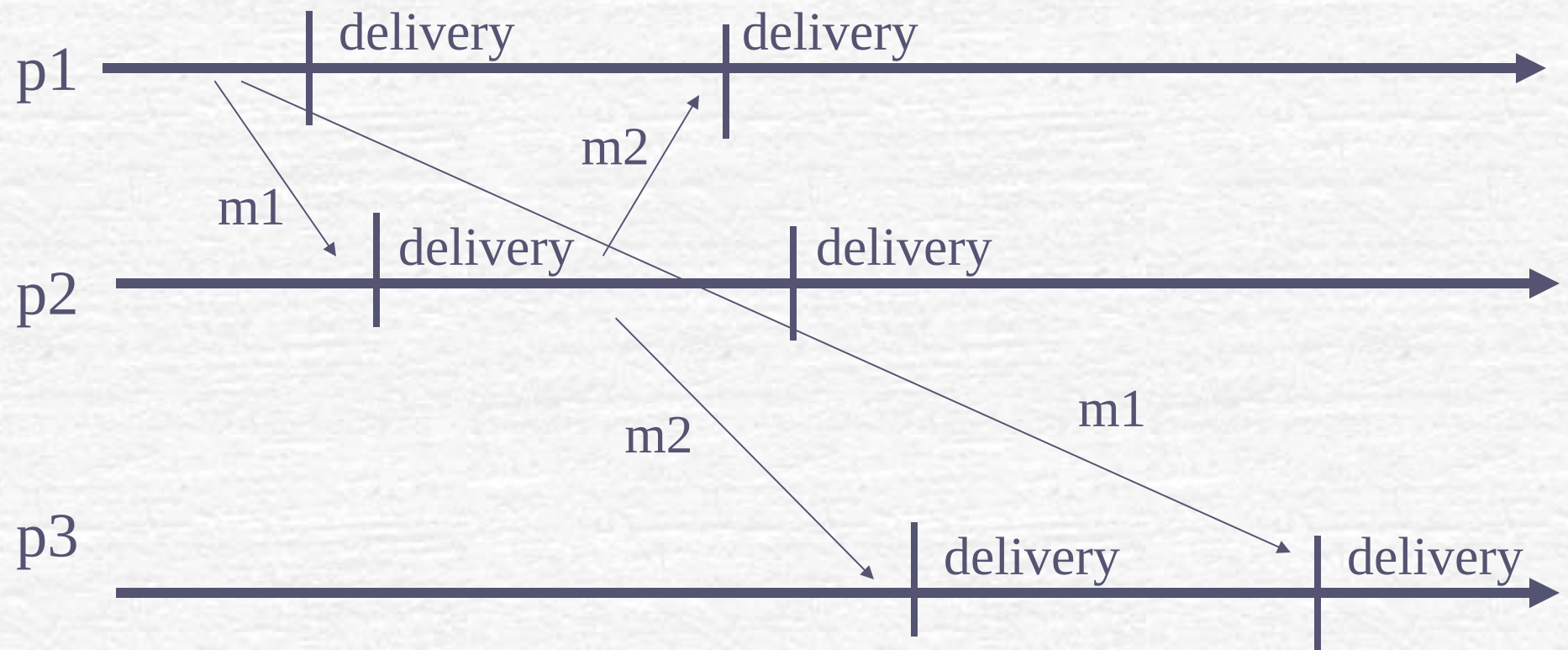
Causality ?



Causality ?



Causality ?



Reliable causal broadcast (rcb)

• **Events**

- Request: $\langle \text{rcoBroadcast}, m \rangle$
- Indication: $\langle \text{rcoDeliver}, \text{src}, m \rangle$

• **Properties:**

- ***RB1, RB2, RB3, RB4 +***
- ***CO***

Uniform causal broadcast (ucb)

• **Events**

- Request: $\langle \text{ucoBroadcast}, m \rangle$
- Indication: $\langle \text{ucoDeliver}, \text{src}, m \rangle$

• **Properties:**

- **URB1, URB2, URB3, URB4 +**
- **CO**

Overview

- ✓ **Intuitions:** why causal broadcast?
- ✓ **Specifications** of ***causal broadcast***
- ✓ **Algorithms:**
 - ✓ A ***non-blocking*** algorithm using the ***past*** and
 - ✓ A ***blocking*** algorithm using ***vector clocks***

Algorithms

- We present **reliable causal broadcast** algorithms using **reliable broadcast**
- We obtain **uniform causal broadcast** algorithms by using instead an underlying **uniform reliable broadcast**

Algorithm 1

- ✎ **Implements:** ReliableCausalOrderBroadcast (rco).
- ✎ **Uses:** ReliableBroadcast (rb).
- ✎ **upon event** < Init > **do**
 - ✎ delivered := past := \emptyset ;
- ✎ **upon event** < rcoBroadcast, m > **do**
 - ✎ **trigger** < rbBroadcast, [Data,past,m]>;
 - ✎ past := past \cup {[self,m]};

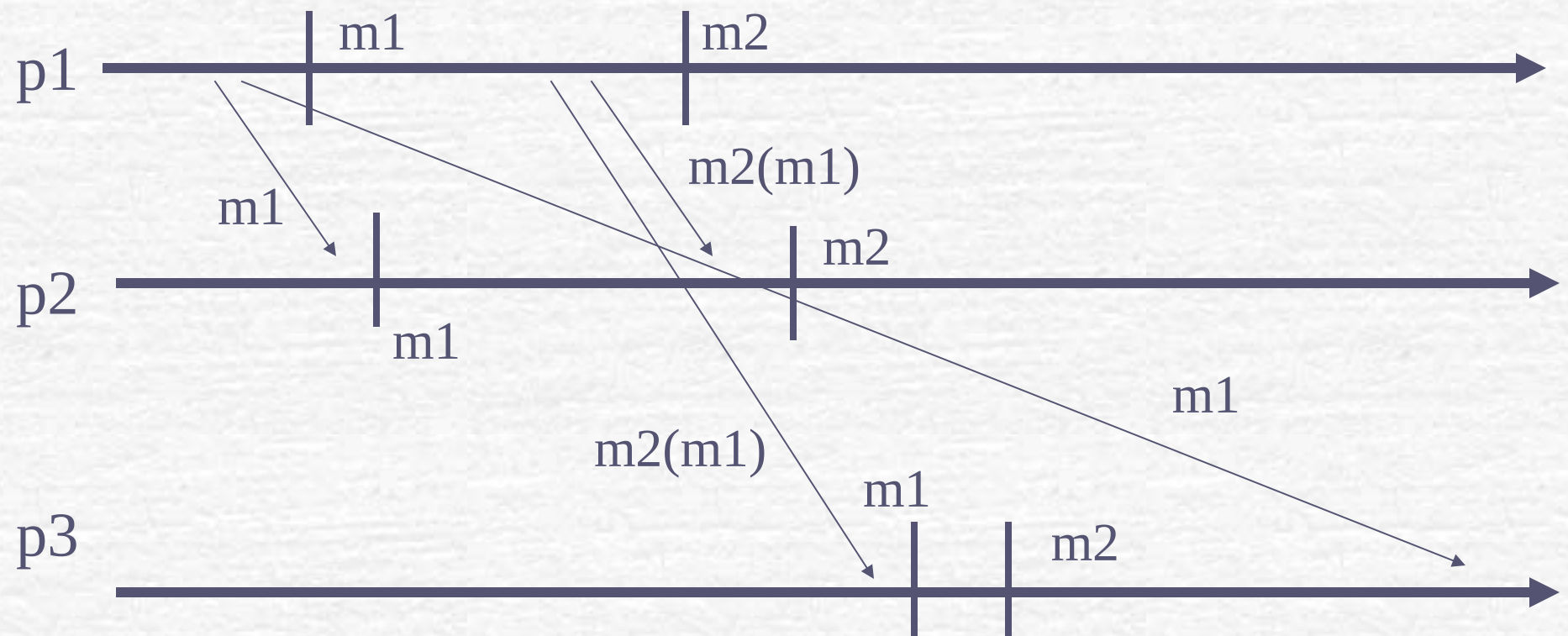
Algorithm 1 (cont'd)

- ☛ **upon event** $\langle \text{rbDeliver}, \text{pi}, [\text{Data}, \text{pastm}, \text{m}] \rangle$
do
 - ☛ **if** $\text{m} \notin \text{delivered}$ **then**
 - ☛ (*) **forall** $[\text{sn}, \text{n}]$ in pastm **do**
 - ☛ **if** $\text{n} \notin \text{delivered}$ **then**
 - ☛ **trigger** $\langle \text{rcoDeliver}, \text{sn}, \text{n} \rangle$;
 - ☛ $\text{delivered} := \text{delivered} \cup \{\text{n}\}$;
 - ☛ $\text{past} := \text{past} \cup \{[\text{sn}, \text{n}]\}$;

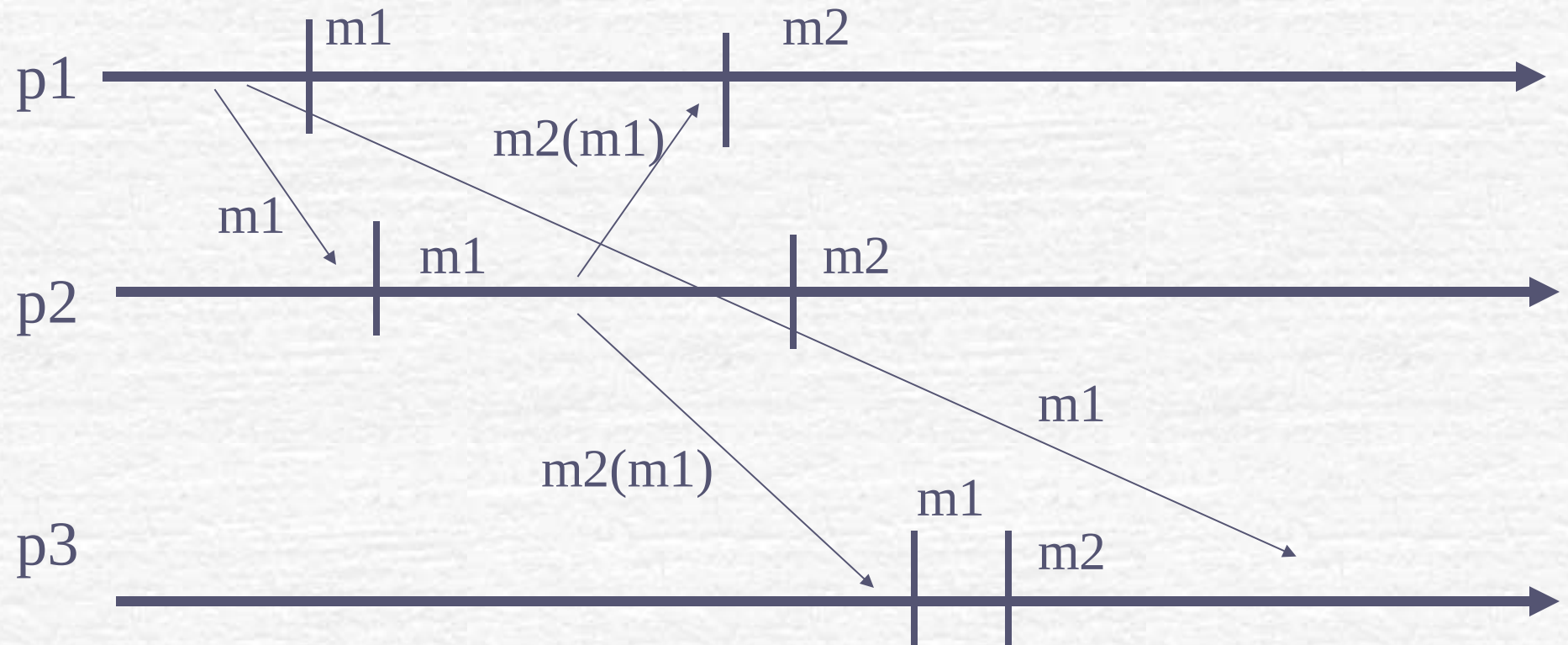
Algorithm 1 (cont'd)

- (*)
- ...
- ...
- ...
- **trigger** <rcoDeliver,pi,m>;
- delivered := delivered U {m};
- past := past U {[pi,m]};

Algorithm 1



Algorithm 1



Uniformity

- Algorithm 1 ensures causal reliable broadcast
- If we replace reliable broadcast with uniform reliable broadcast, Algorithm 1 would ensure uniform causal broadcast

Algorithm 1' (gc)

- ✎ **Implements:** GarbageCollection (+ Algo 1).
- ✎ **Uses:**
 - ✎ ReliableBroadcast (rb).
 - ✎ PerfectFailureDetector(P).
- ✎ **upon event** < Init > **do**
 - ✎ delivered := past := \emptyset ;
 - ✎ correct := S;
 - ✎ ackm := \emptyset (for all m);

Algorithm 1' (gc – cont'd)

- ☛ **upon event** $\langle \text{crash}, p_i \rangle$ **do**
 - ☛ $\text{correct} := \text{correct} \setminus \{p_i\}$
- ☛ **upon** for some $m \in \text{delivered}$: $\text{self} \notin \text{ackm}$ **do**
 - $\text{ackm} := \text{ackm} \cup \{\text{self}\};$
 - **trigger** $\langle \text{rbBroadcast}, [\text{ACK}, m] \rangle;$

Algorithm 1' (gc – cont'd)

- ☛ **upon event** $\langle \text{rbDeliver}, p_i, [\text{ACK}, m] \rangle$ **do**
 - ☛ $\text{ack}_m := \text{ack}_m \cup \{p_i\};$
 - ☛ **if forall** $p_j \in \text{correct}: p_j \in \text{ack}_m$ **do**
 - ☛ $\text{past} := \text{past} \setminus \{[s_m, m]\};$

Algorithm 2

- **Implements:** ReliableCausalOrderBroadcast (rco).
- **Uses:** ReliableBroadcast (rb).
- **upon event** < Init > **do**
 - **for all** $p_i \in S$: $VC[p_i] := 0$;
 - $pending := \emptyset$

Algorithm 2 (cont'd)

- **upon event** < rcoBroadcast, m > **do**
 - **trigger** < rcoDeliver, self, m >;
 - **trigger** < rbBroadcast, [Data, VC, m] >;
 - $VC[self] := VC[self] + 1;$

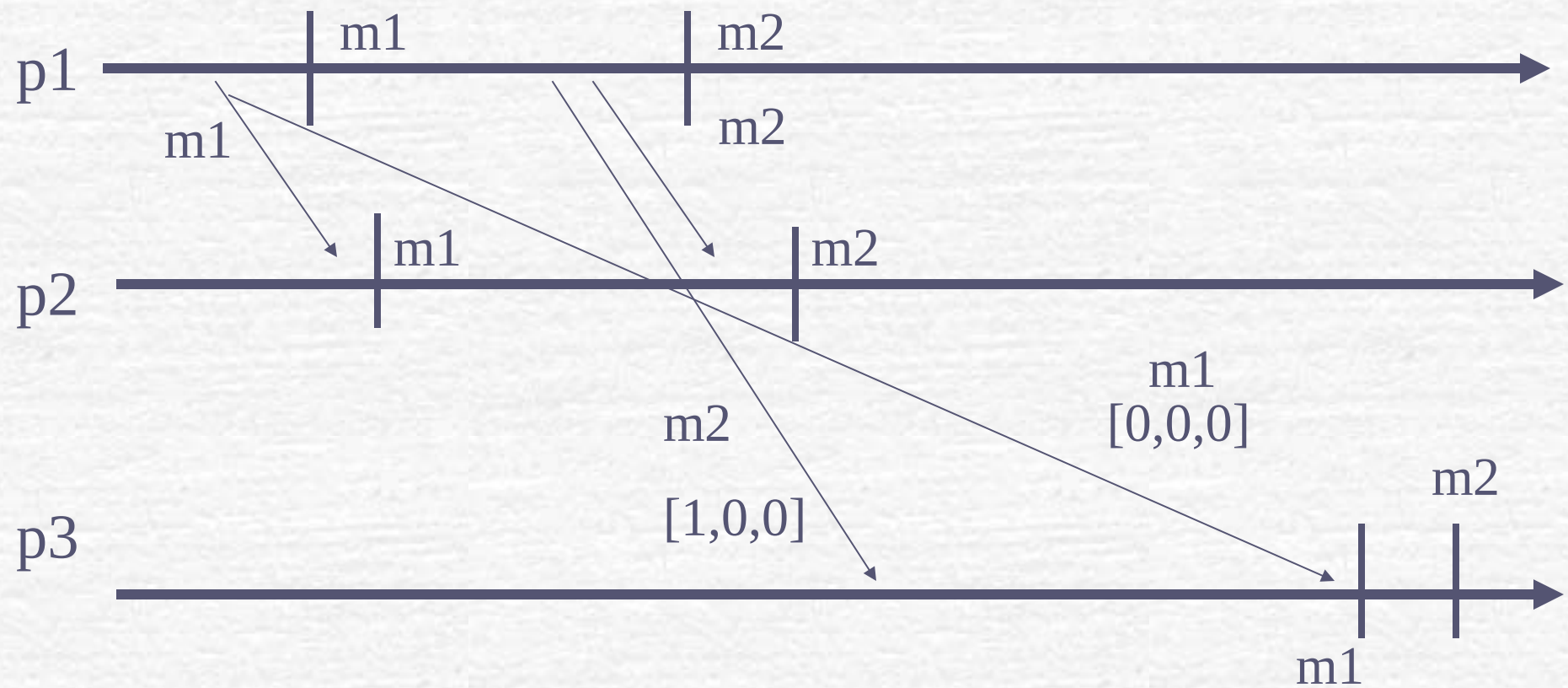
Algorithm 2 (cont'd)

- ☛ **upon event** <rbDeliver, pj, [Data,VCm,m]>
do
 - ☛ **if** pj \neq self **then**
 - ☛ pending := pending \cup (pj, [Data,VCm,m]);
 - ☛ deliver-pending.

Algorithm 2 (cont'd)

- **procedure deliver-pending is**
 - **While** $(s, [\text{Data}, \text{VCm}, m]) \in \text{pending}$ **s.t.**
 - **for all** $pk: (\text{VC}[pk] \geq \text{VCm}[pk])$ **do**
 - $\text{pending} := \text{pending} - (s, [\text{Data}, \text{VCm}, m]);$
 - **trigger** $\langle \text{rcoDeliver}, \text{self}, m \rangle;$
 - $\text{VC}[s] := \text{VC}[s] + 1.$

Algorithm 2



Algorithm 2

