

# Distributed systems

## Reliable Broadcast

*Prof R. Guerraoui*  
*Lpdwww.epfl.ch*

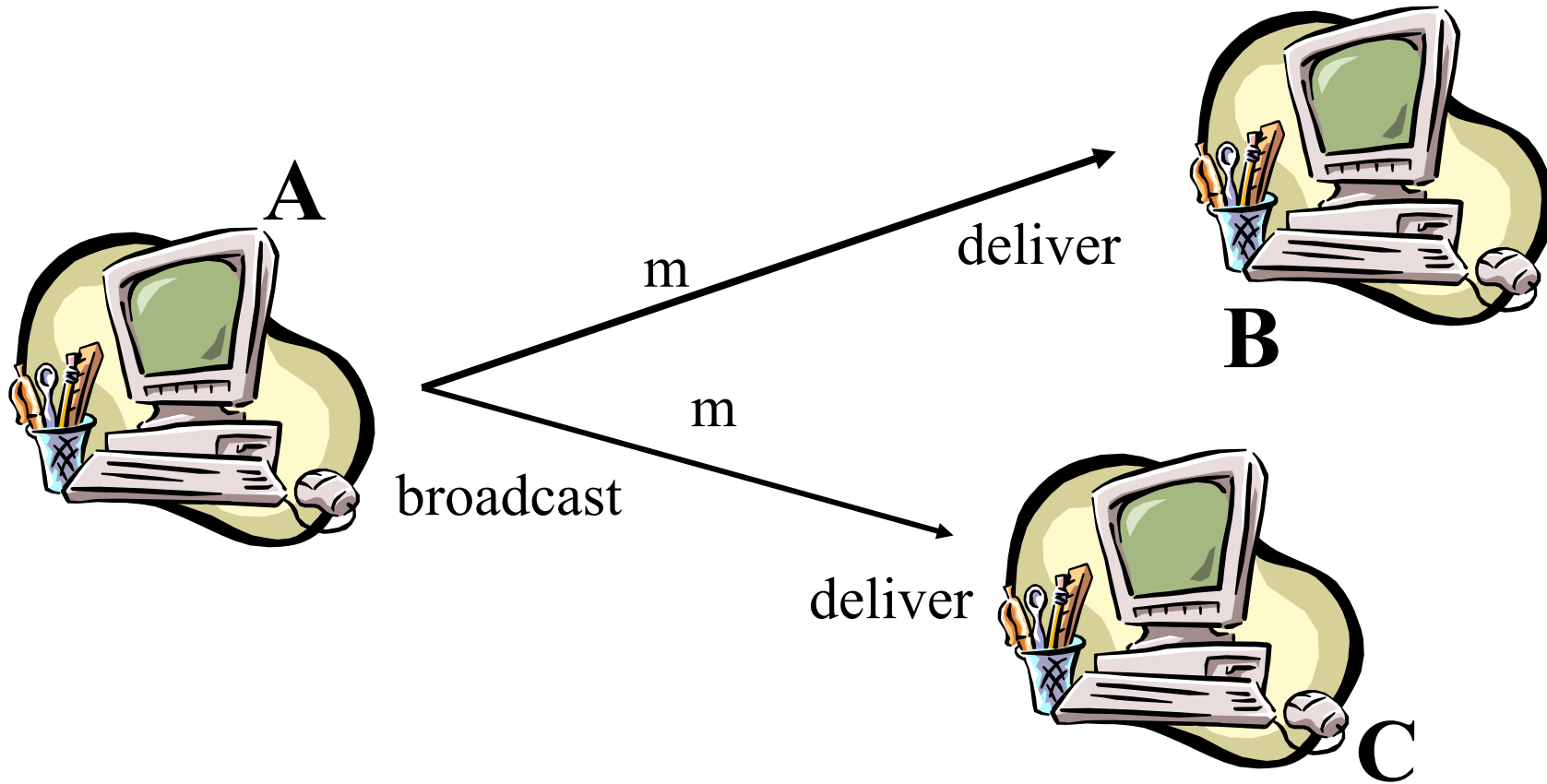


© R. Guerraoui

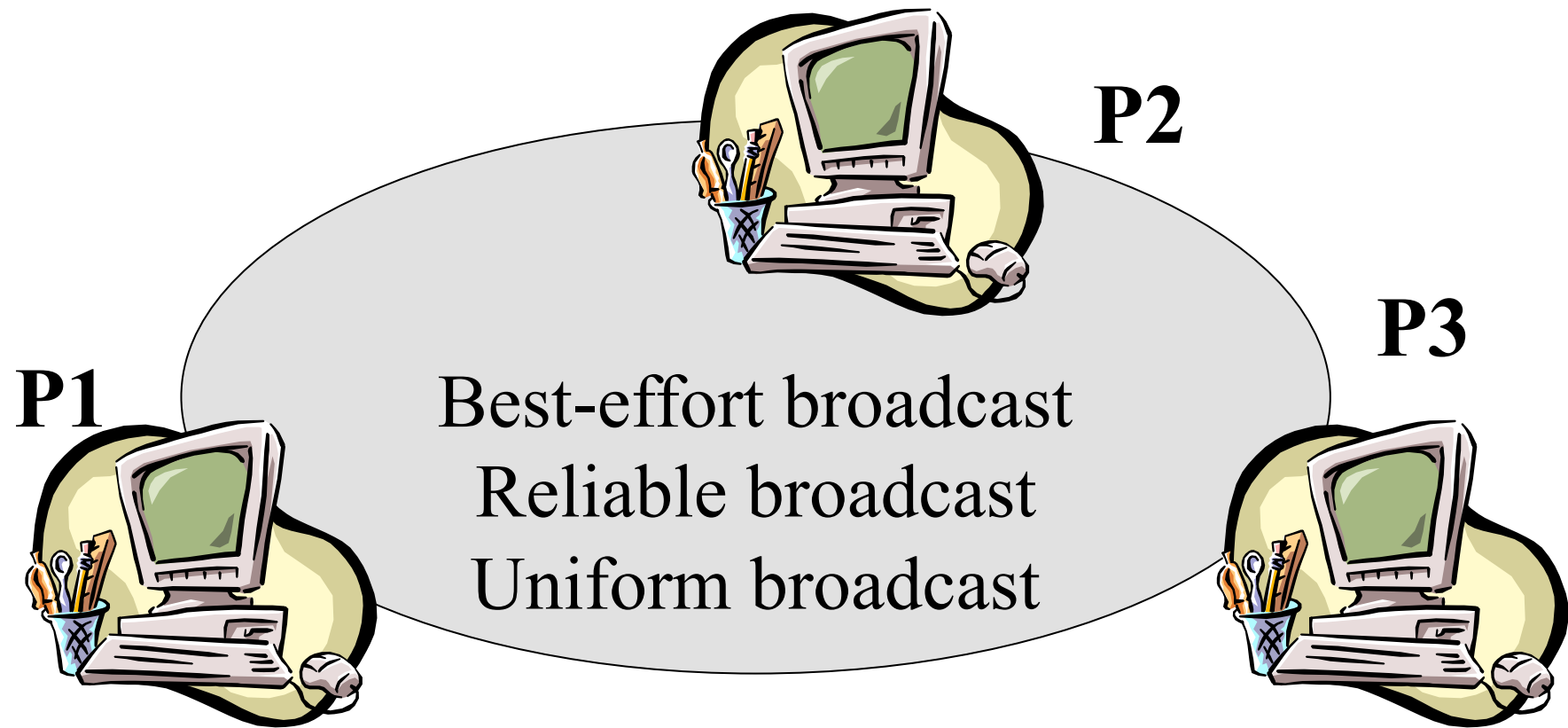
1



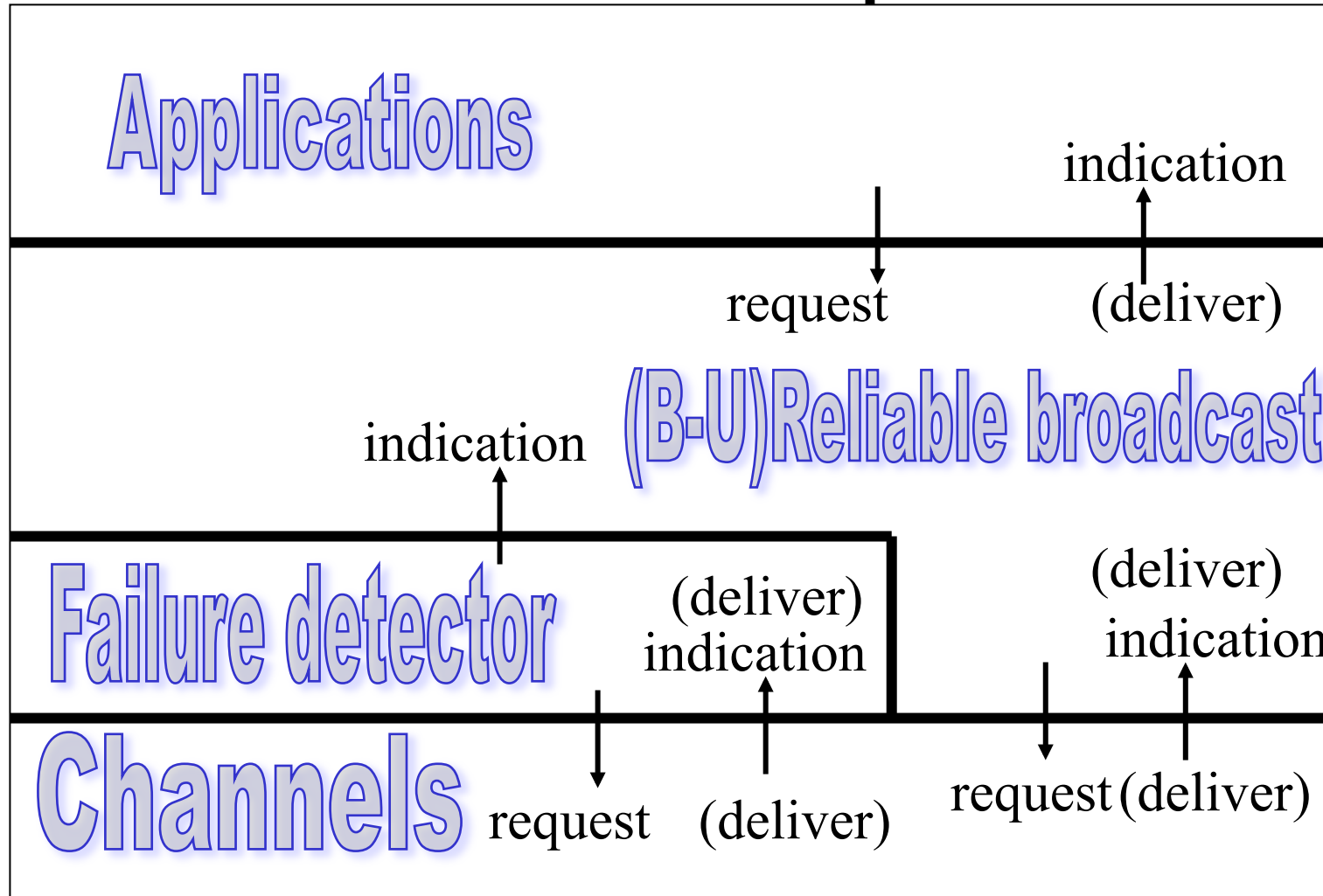
# Broadcast



# Broadcast abstractions



# Modules of a process



# Intuition

- Broadcast is useful for instance in applications where some processes subscribe to events published by other processes (e.g., stocks)
- The subscribers might require some **reliability** *guarantees* from the broadcast service (*quality of service* – *QoS*) that the underlying network does not provide

# Overview

- We shall consider three forms of reliability for a broadcast primitive
- **(1) *Best-effort broadcast***
- **(2) *(Regular) reliable broadcast***
- **(3) *Uniform (reliable) broadcast***
- We shall give first ***specifications*** and then *algorithms*

# Best-effort broadcast (beb)

## • *Events*

• Request: <bebBroadcast, m>

• Indication: <bebDeliver, src, m>

• *Properties: BEB1, BEB2, BEB3*

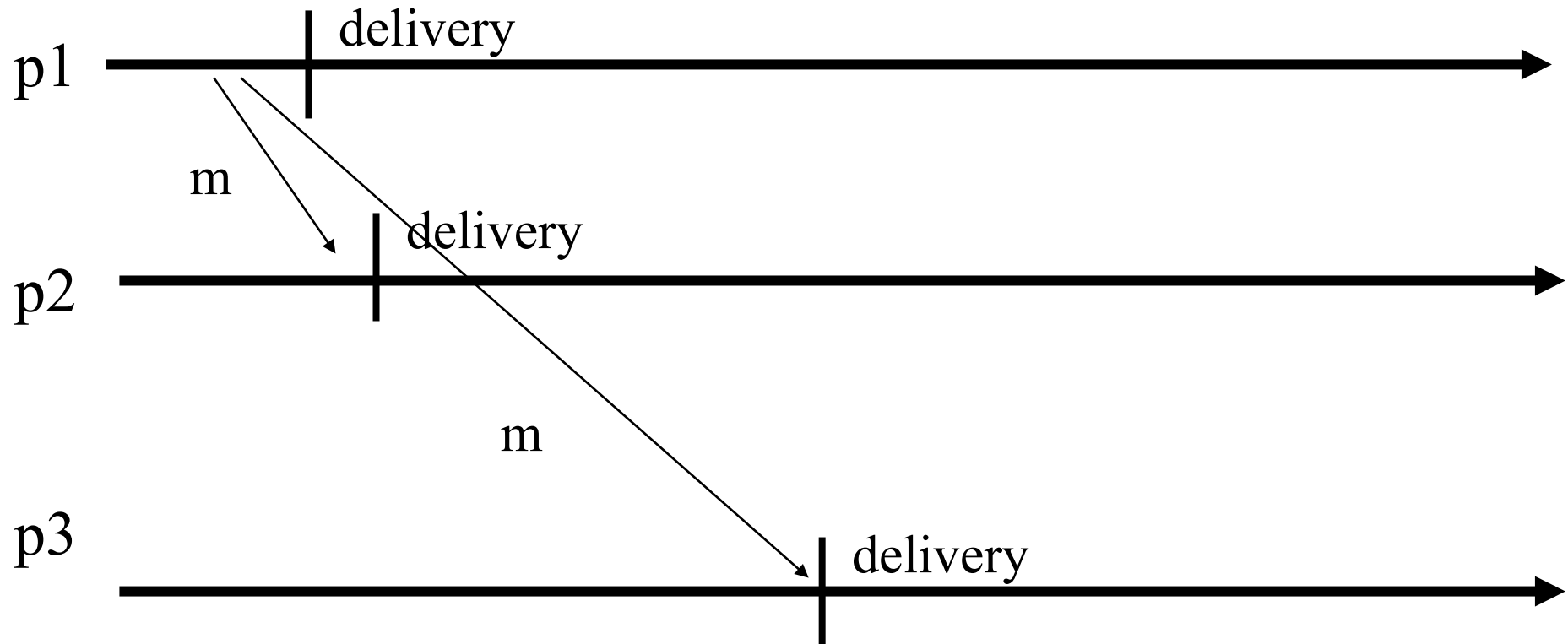
# Best-effort broadcast (beb)

## ☛ *Properties*

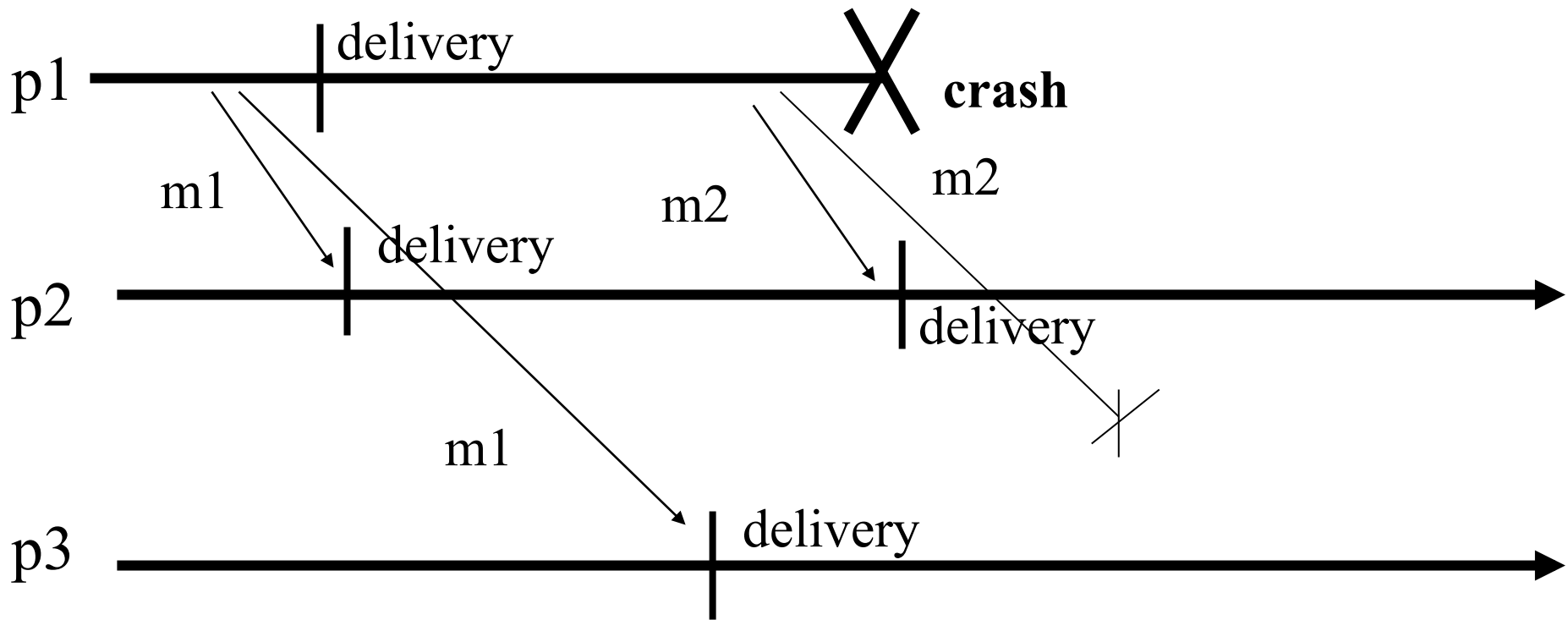
- ☛ ***BEB1. Validity:*** If  $p_i$  and  $p_j$  are correct, then every message broadcast by  $p_i$  is eventually delivered by  $p_j$
- ☛ ***BEB2. No duplication:*** No message is delivered more than once
- ☛ ***BEB3. No creation:*** No message is delivered unless it was broadcast



# Best-effort broadcast



# Best-effort broadcast



# Reliable broadcast (rb)

## • ***Events***

- Request:  $\langle \text{rbBroadcast}, m \rangle$
- Indication:  $\langle \text{rbDeliver}, \text{src}, m \rangle$

- ***Properties: RB1, RB2, RB3, RB4***

# Reliable broadcast (rb)

## ☛ *Properties*

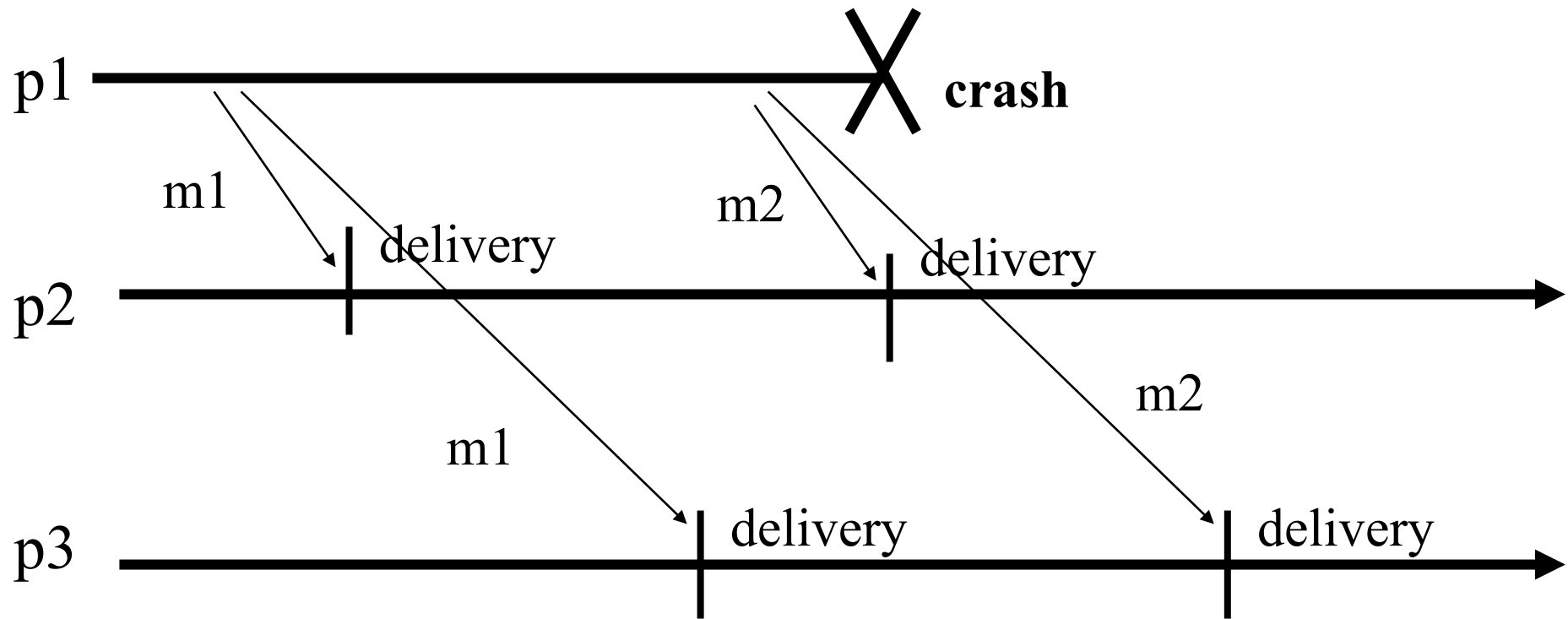
☛ ***RB1 = BEB1.***

☛ ***RB2 = BEB2.***

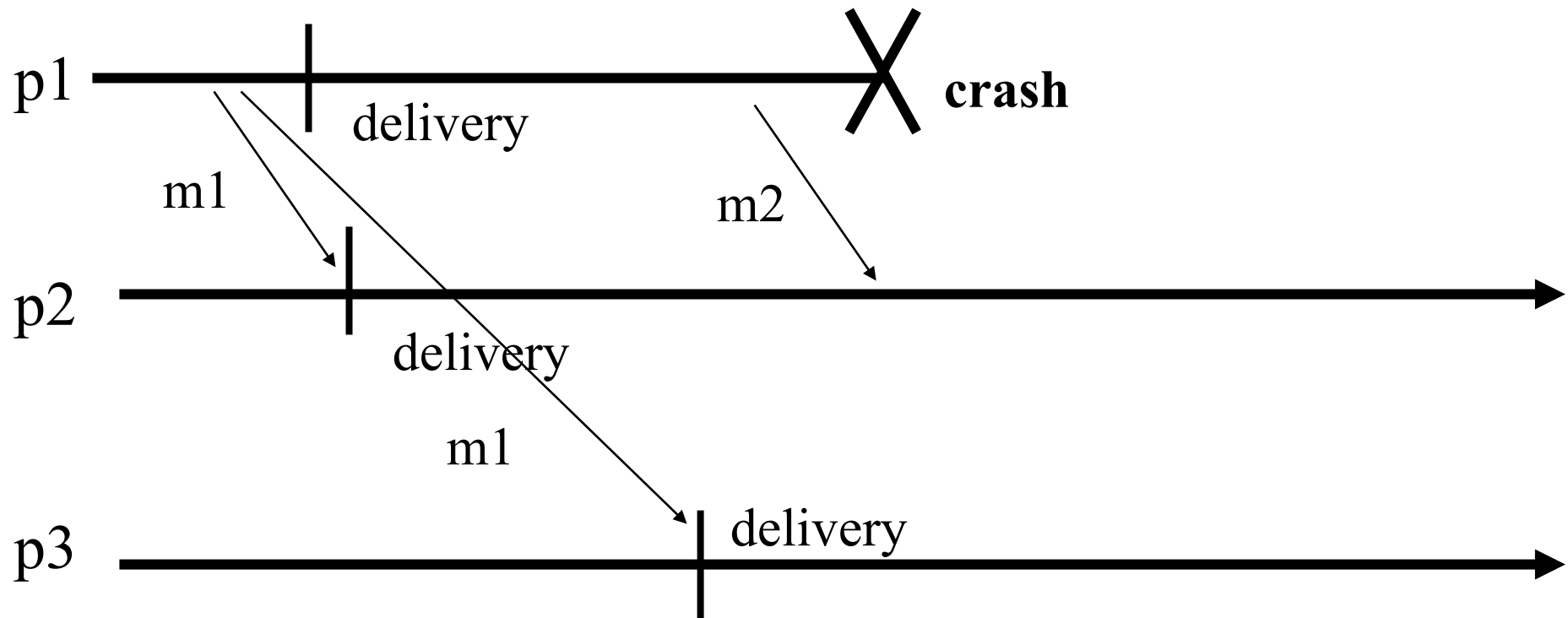
☛ ***RB3 = BEB3.***

☛ ***RB4. Agreement:*** For any message  $m$ , if a correct process delivers  $m$ , then every correct process delivers  $m$

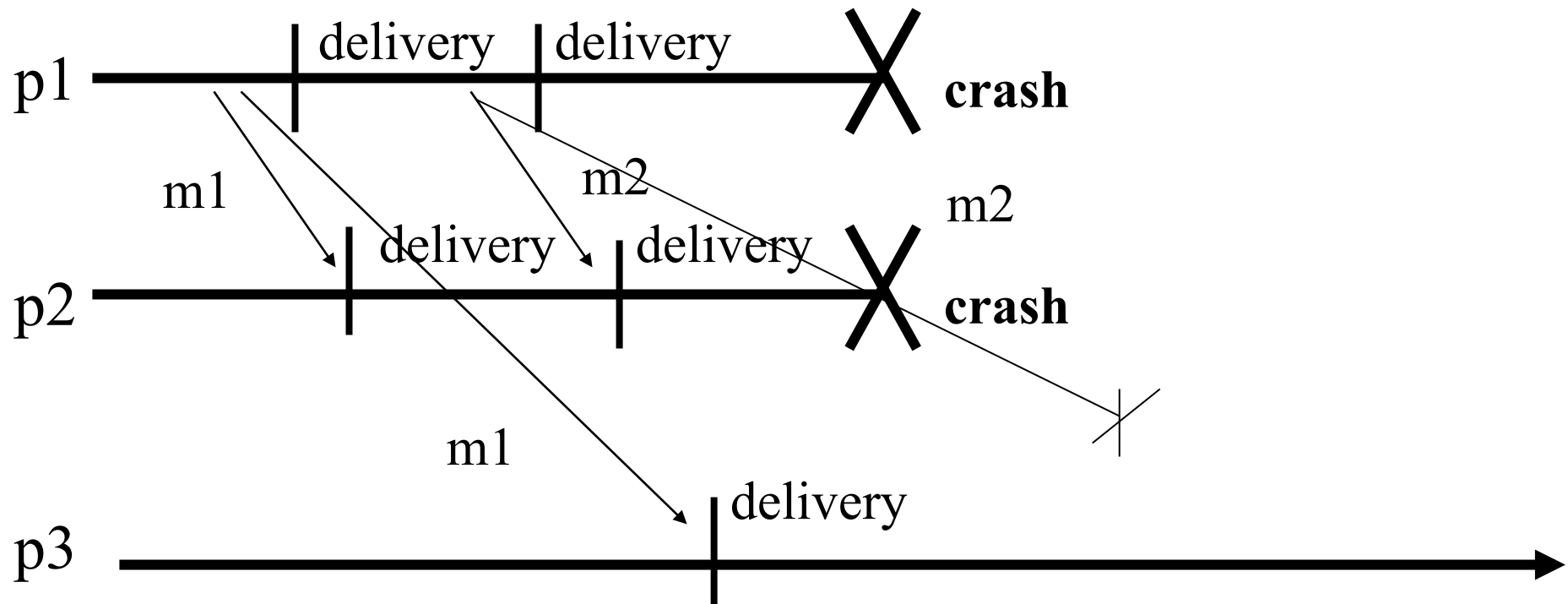
# Reliable broadcast



# Reliable broadcast



# Reliable broadcast



# Uniform broadcast (urb)

## • **Events**

• Request: <urbBroadcast, m>

• Indication: <urbDeliver, src, m>

• **Properties: URB1, URB2, URB3, URB4**



# Uniform broadcast (urb)

## • *Properties*

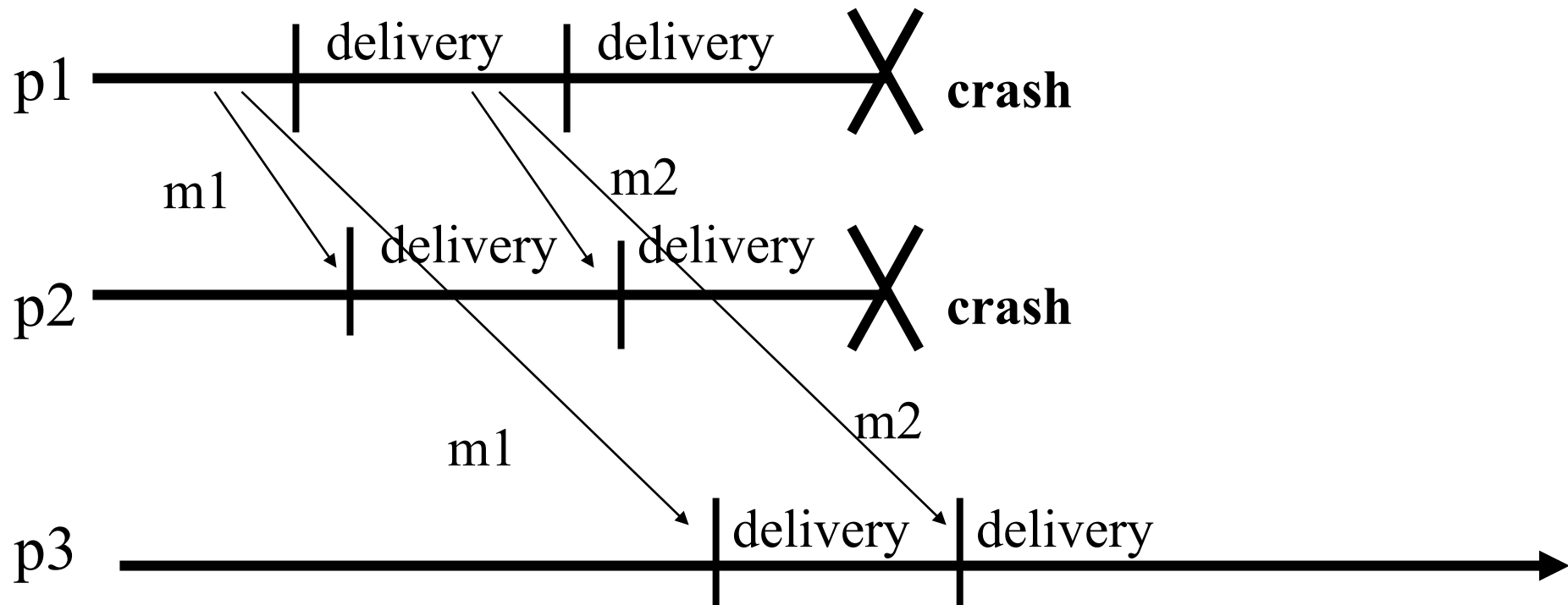
• ***URB1 = BEB1.***

• ***URB2 = BEB2.***

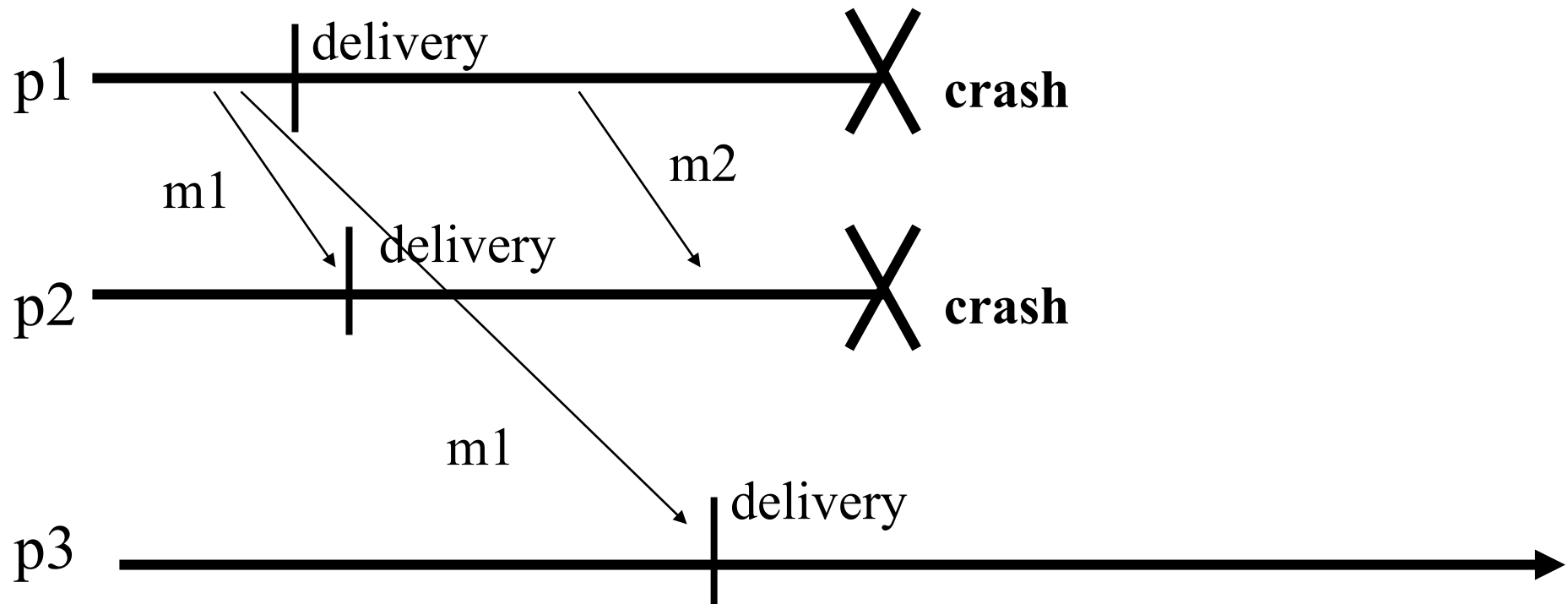
• ***URB3 = BEB3.***

• ***URB4. Uniform Agreement:*** For any message  $m$ , if a process delivers  $m$ , then every correct process delivers  $m$

# Uniform reliable broadcast



# Uniform reliable broadcast



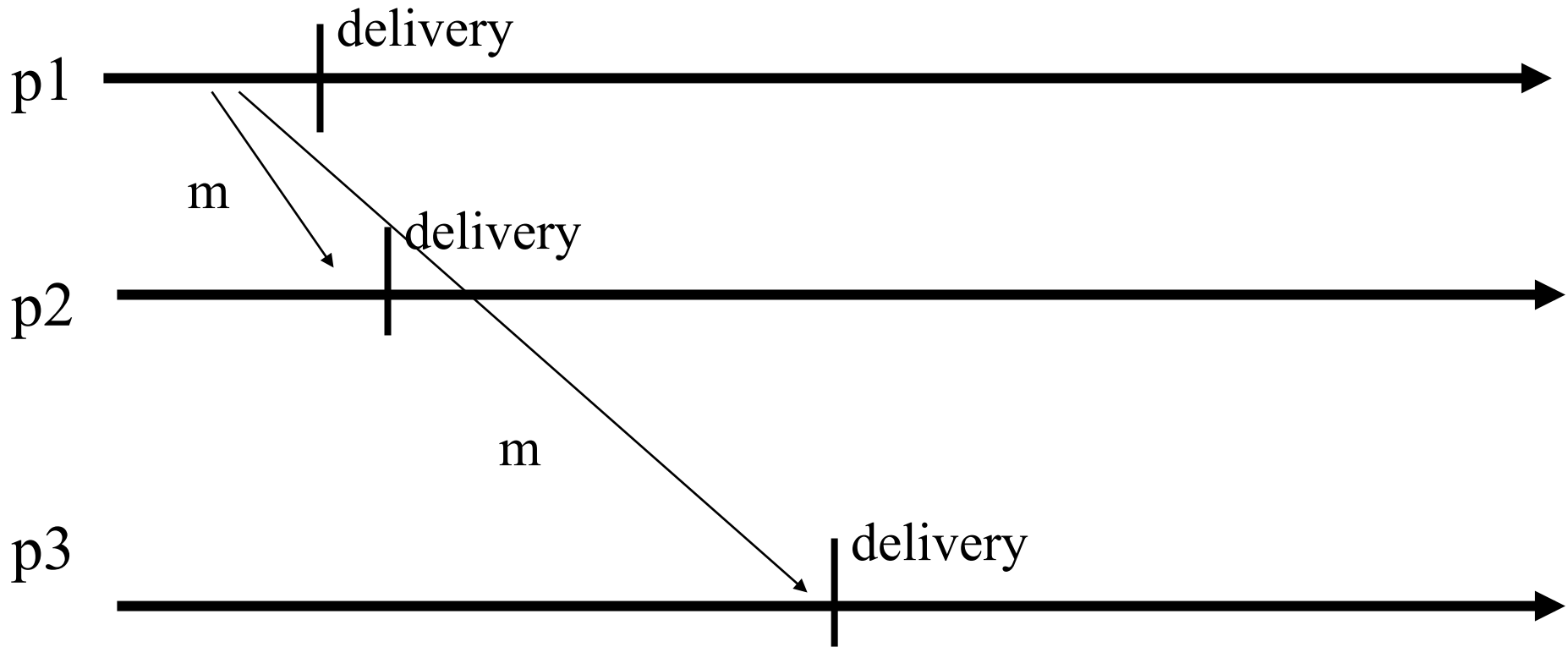
# Overview

- We consider three forms of reliability for a broadcast primitive
- **(1) *Best-effort broadcast***
- **(2) *(Regular) reliable broadcast***
- **(3) *Uniform (reliable) broadcast***
- We give first *specifications* and then ***algorithms***

# Algorithm (beb)

- **Implements:** BestEffortBroadcast (beb).
- **Uses:** PerfectLinks (pp2p).
- **upon event**  $\langle \text{bebBroadcast}, m \rangle$  **do**
  - **forall**  $p_i \in S$  **do**
    - **trigger**  $\langle \text{pp2pSend}, p_i, m \rangle$ ;
- **upon event**  $\langle \text{pp2pDeliver}, p_i, m \rangle$  **do**
  - **trigger**  $\langle \text{bebDeliver}, p_i, m \rangle$ ;

# Algorithm (beb)

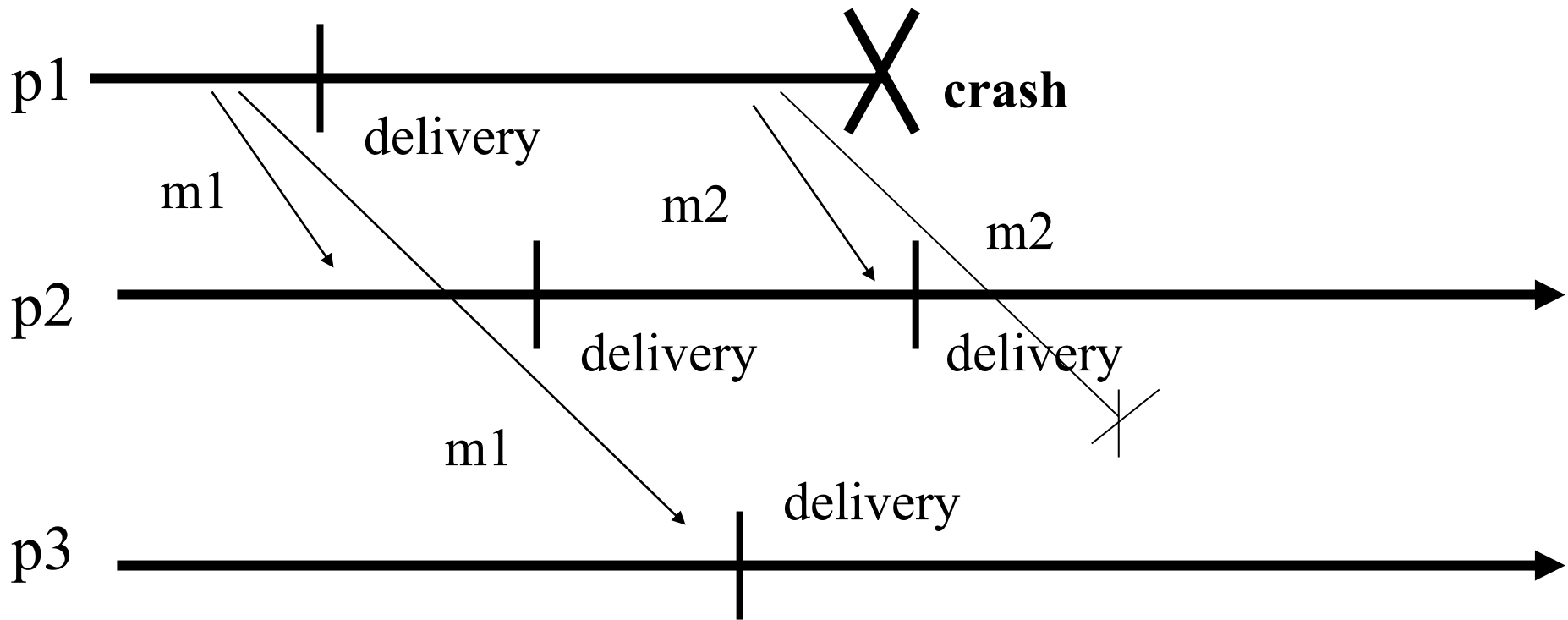


# Algorithm (beb)

## • *Proof (sketch)*

- ***BEB1. Validity:*** By the validity property of perfect links and the very facts that (1) the sender sends the message to all and (2) every correct process that pp2pDelivers a message bebDelivers it
- ***BEB2. No duplication:*** By the no duplication property of perfect links
- ***BEB3. No creation:*** By the no creation property of the perfect links

# Algorithm (beb)





# Algorithm (rb)

- **Implements:** ReliableBroadcast (rb).
- **Uses:**
  - BestEffortBroadcast (beb).
  - PerfectFailureDetector (P).
- **upon event** < Init > **do**
  - delivered :=  $\emptyset$ ;
  - correct := S;
  - **forall**  $p_i \in S$  **do** from[ $p_i$ ] := empty;

## Algorithm (rb – cont'd)

- **upon event** < rbBroadcast, m > **do**
  - delivered := delivered U {m};
  - **trigger** < rbDeliver, self, m >;
  - **trigger** < bebBroadcast, [Data,self,m] >;

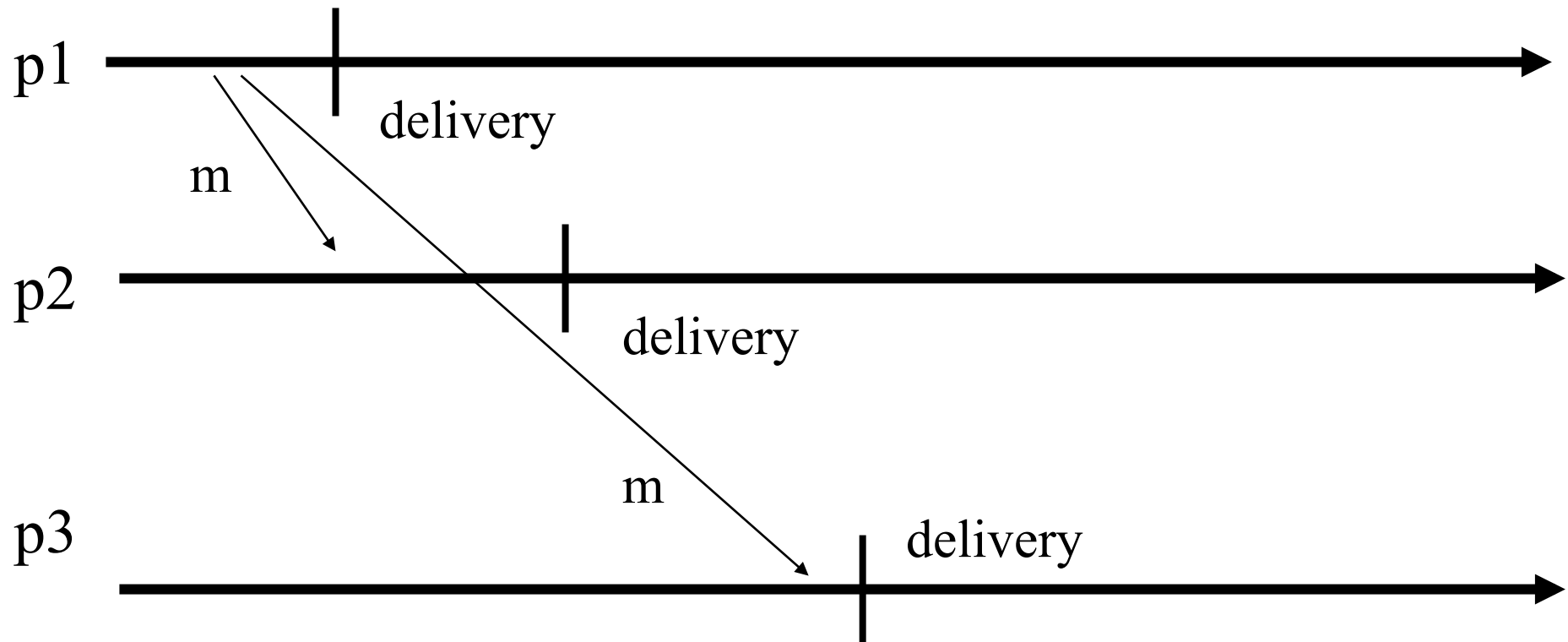
# Algorithm (rb – cont'd)

- **upon event**  $\langle \text{crash}, p_i \rangle$  **do**
  - $\text{correct} := \text{correct} \setminus \{p_i\};$
  - **forall**  $[p_j, m] \in \text{from}[p_i]$  **do**
    - **trigger**  $\langle \text{bebBroadcast}, [\text{Data}, p_j, m] \rangle;$

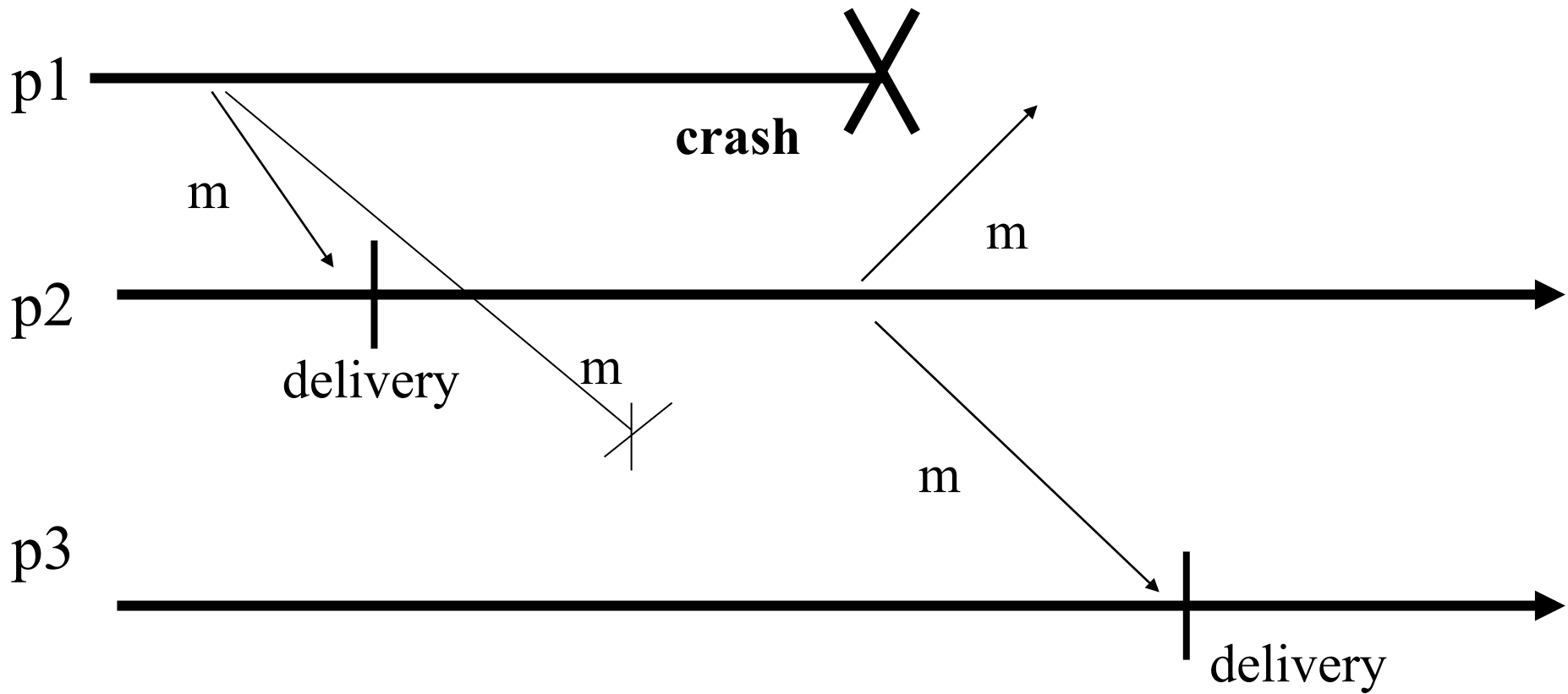
# Algorithm (rb – cont'd)

- **upon event**  $\langle \text{bebDeliver}, p_i, [\text{Data}, p_j, m] \rangle$  **do**
  - **if**  $m \notin \text{delivered}$  **then**
    - $\text{delivered} := \text{delivered} \cup \{m\};$
    - **trigger**  $\langle \text{rbDeliver}, p_j, m \rangle;$
    - **if**  $p_i \notin \text{correct}$  **then**
      - **trigger**  $\langle \text{bebBroadcast}, [\text{Data}, p_j, m] \rangle;$
    - **else**
      - $\text{from}[p_i] := \text{from}[p_i] \cup \{[p_j, m]\};$

# Algorithm (rb)



# Algorithm (rb)



# Algorithm (rb)

## • *Proof (sketch)*

- **RB1. RB2. RB3:** as for the 1st algorithm
- **RB4. Agreement:** Assume some correct process  $p_i$  rbDelivers a message  $m$  rbBroadcast by some process  $p_k$ . If  $p_k$  is correct, then by property BEB1, all correct processes bebDeliver and then rebDeliver  $m$ . If  $p_k$  crashes, then by the completeness property of  $P$ ,  $p_i$  detects the crash and bebBroadcasts  $m$  to all. Since  $p_i$  is correct, then by property BEB1, all correct processes bebDeliver and then rebDeliver  $m$ .

# Algorithm (urb)

- **Implements:** uniformBroadcast (urb).
- **Uses:**
  - BestEffortBroadcast (beb).
  - PerfectFailureDetector (P).
- **upon event** < Init > **do**
  - correct := S;
  - delivered := forward := empty;
  - ack[Message] :=  $\emptyset$ ;



# Algorithm (urb – cont'd)

- **upon event**  $\langle \text{crash}, \text{pi} \rangle$  **do**
  - $\text{correct} := \text{correct} \setminus \{\text{pi}\};$
- **upon event**  $\langle \text{urbBroadcast}, \text{m} \rangle$  **do**
  - $\text{forward} := \text{forward} \cup \{[\text{self}, \text{m}]\};$
  - **trigger**  $\langle \text{bebBroadcast}, [\text{Data}, \text{self}, \text{m}] \rangle;$

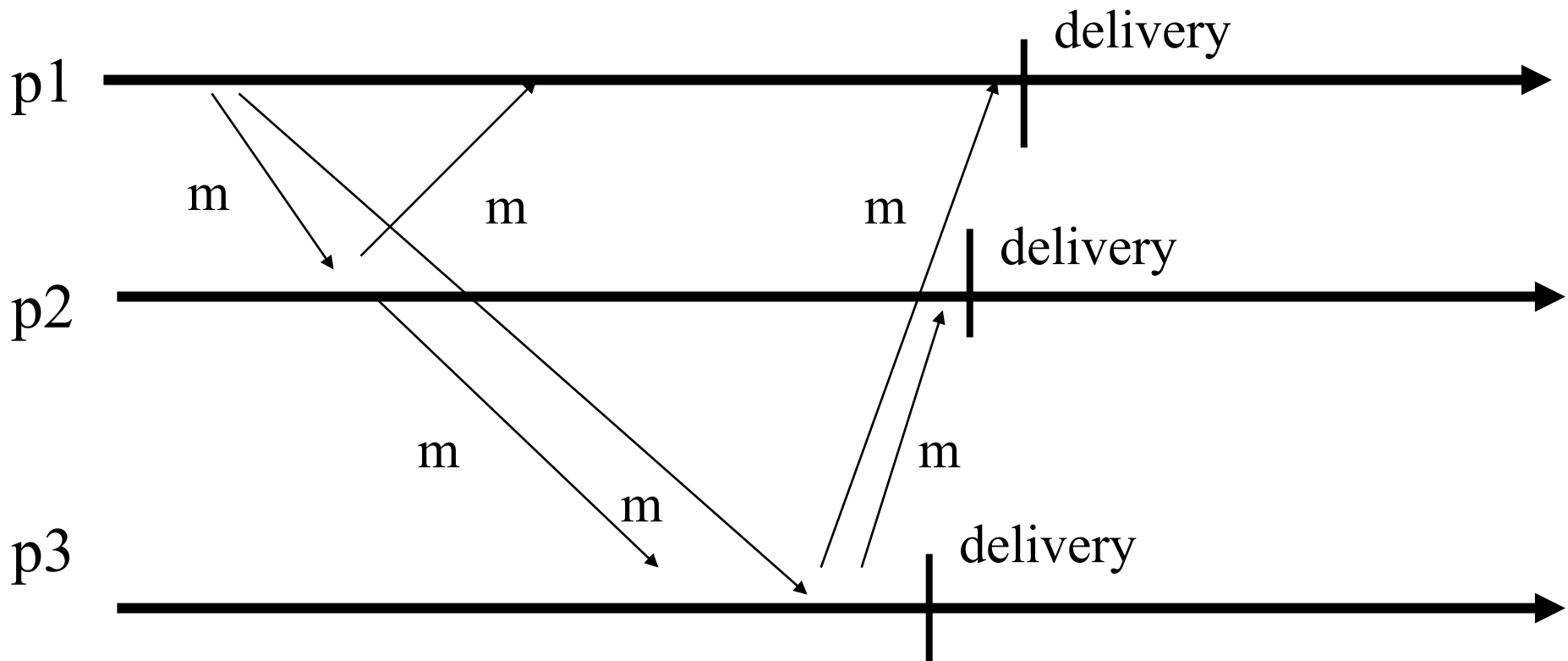
# Algorithm (urb – cont'd)

- **upon event**  $\langle \text{bebDeliver}, p_i, [\text{Data}, p_j, m] \rangle$  **do**
  - $\text{ack}[m] := \text{ack}[m] \cup \{p_i\};$
  - **if**  $[p_j, m] \notin \text{forward}$  **then**
    - $\text{forward} := \text{forward} \cup \{[p_j, m]\};$
    - **trigger**  $\langle \text{bebBroadcast}, [\text{Data}, p_j, m] \rangle;$

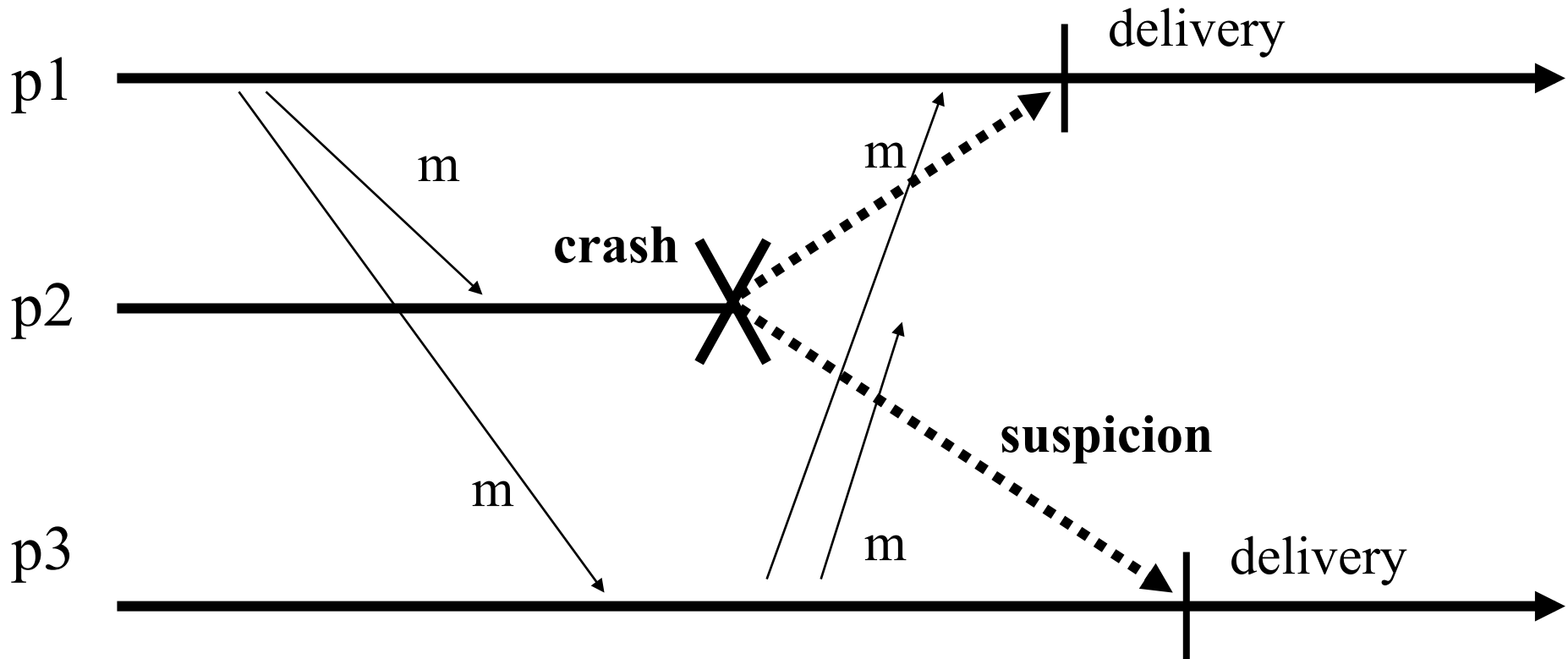
# Algorithm (urb – cont'd)

- **upon event** (for any  $[pj, m] \in \text{forward}$ )  
 $\langle \text{correct} \subseteq \text{ack}[m] \rangle$  **and**  $\langle m \notin \text{delivered} \rangle$  **do**
  - $\text{delivered} := \text{delivered} \cup \{m\};$
  - **trigger**  $\langle \text{urbDeliver}, pj, m \rangle;$

# Algorithm (urb)



# Algorithm (urb)



# Algorithm (urb)

## • ***Proof (sketch)***

- ***URB2. URB3:*** follow from BEB2 and BEB3
- ***A simple lemma:*** *If a correct process  $p_i$  bebDelivers a message  $m$ , then  $p_i$  eventually urbDelivers  $m$ .*
- Any process that bebDelivers  $m$  bebBroadcasts  $m$ . By the completeness property of the failure detector and property BEB1, there is a time at which  $p_i$  bebDelivers  $m$  from every correct process and hence urbDelivers  $m$ .

# Algorithm (urb)

## • ***Proof (sketch)***

- ***URB1. Validity:*** If a correct process  $p_i$  urbBroadcasts a message  $m$ , then  $p_i$  eventually bebBroadcasts and bebDelivers  $m$ : by our lemma,  $p_i$  urbDelivers  $m$ .
- ***URB4. Agreement:*** Assume some process  $p_i$  urbDelivers a message  $m$ . By the algorithm and the completeness and accuracy properties of the failure detector, every correct process bebDelivers  $m$ . By our lemma, every correct process will urbDeliver  $m$ .