11/10/23

Dependable Distributed Systems Master of Science in Engineering in Computer Science

AA 2023/2024

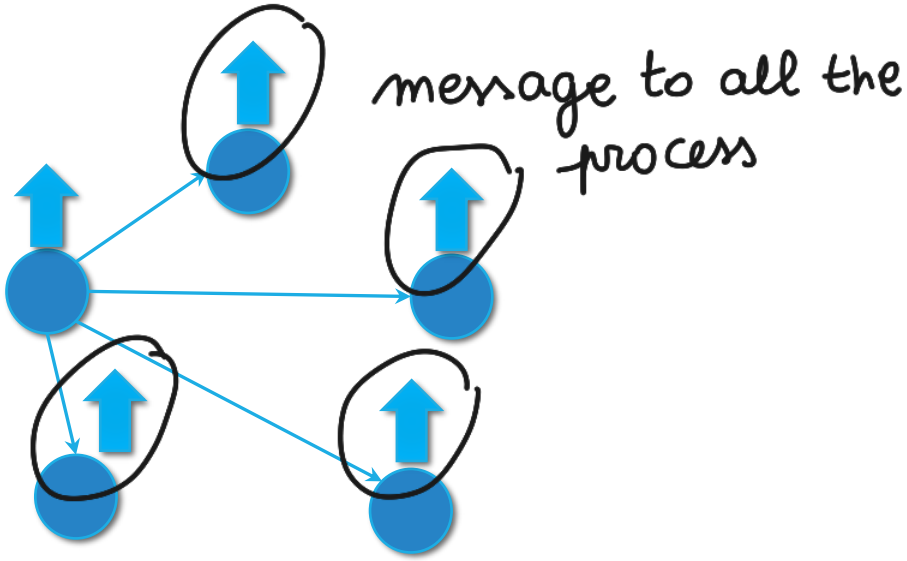
LECTURE 8 : BROADCAST COMMUNICATIONS - BEB & RB

Recap: what we know up to now

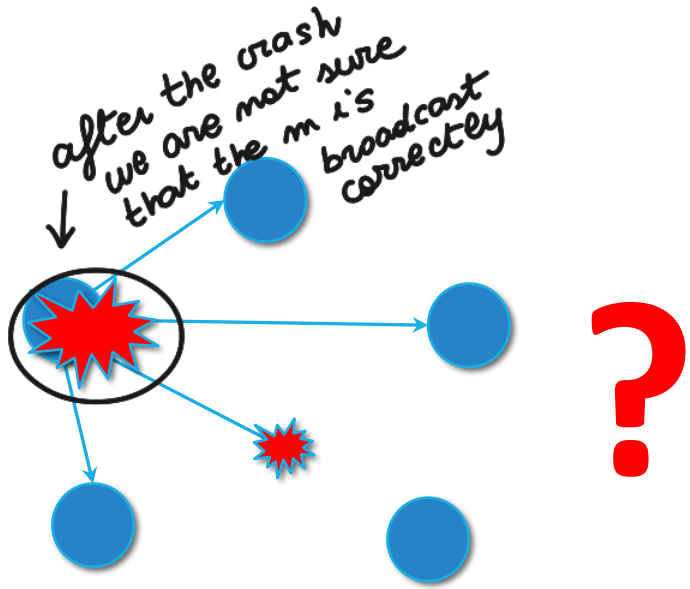
- Define a system model and specify a problem or an abstraction in terms of safety and liveness
- point-to-point communication abstractions
  - fair-loss, stubborn or perfect links
- how to timestamp events
  - physical clocks
  - logical clocks
- handling failures
  - Failure Detector
  - Leader Election

Up to now, the focus has been on the interaction between two processes (like in a client/server environment)

Communication in a group: Broadcast



No Failures



Crash Failures

Best Effort Broadcast (BEB) Specification

Module 3.1: Interface and properties of best-effort broadcast

Module:

Name: BestEffortBroadcast, **instance** *beb*.

Events:

Request: ⟨ *beb*, *Broadcast* | m ⟩: Broadcasts a message m to all processes.

Indication: ⟨ *beb*, *Deliver* | p, m ⟩: Delivers a message m broadcast by process p.

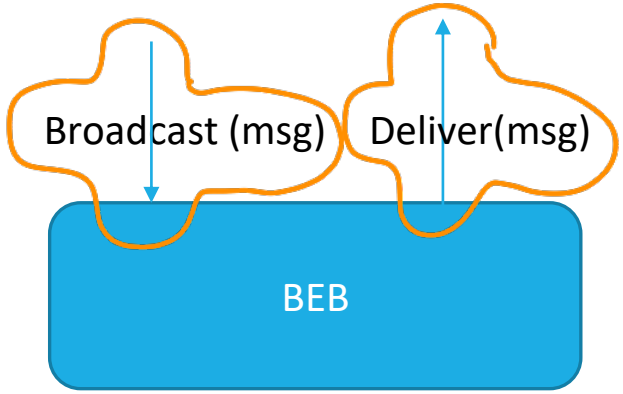
specific source

Properties:

BEB1: *Validity*: If a correct process broadcasts a message m, then every correct process eventually delivers m. → basic

BEB2: *No duplication*: No message is delivered more than once.

BEB3: *No creation*: If a process delivers a message m with sender s, then m was previously broadcast by process s.



Best Effort Broadcast (BEB) Implementation

Algorithm 3.1: Basic Broadcast

Implements:

BestEffortBroadcast, **instance** *beb*.

Uses:

PerfectPointToPointLinks, **instance** *pl*.

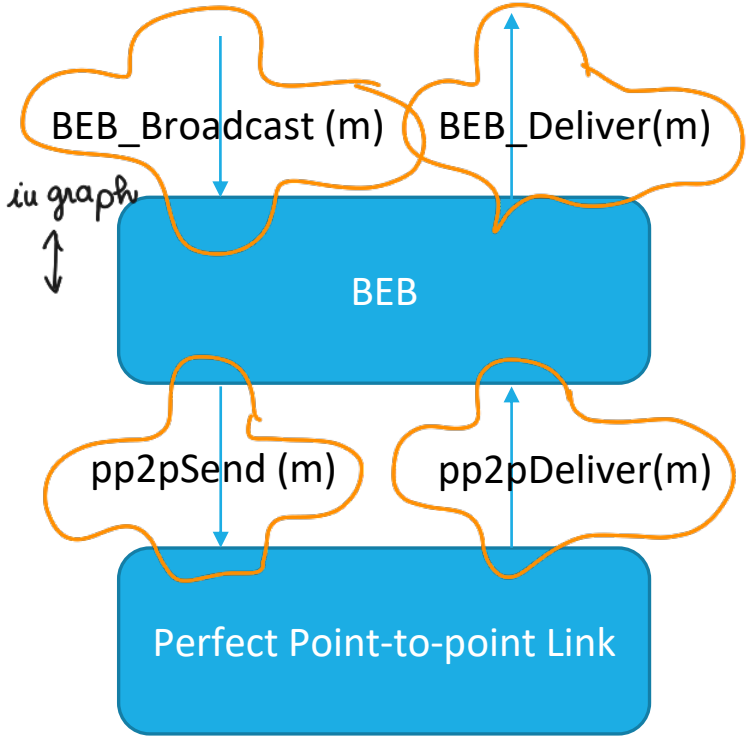
upon event $\langle\ beb, Broadcast \mid m\ \rangle$ **do**
  forall $q \in \Pi$ **do** (process)
    trigger $\langle\ pl, Send \mid q, m\ \rangle$;

} message to all processes

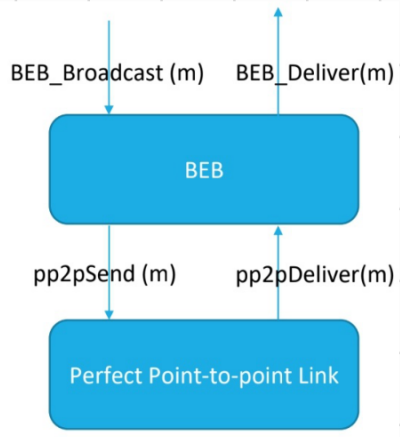
upon event $\langle\ pl, Deliver \mid p, m\ \rangle$ **do**
  trigger $\langle\ beb, Deliver \mid p, m\ \rangle$;

System model

- Asynchronous system NO TIME ASSUMPTION
- perfect links
- crash failures



8. BEB



Properties:

BEB1: *Validity:* If a correct process broadcasts a message m, then every correct process eventually delivers m.

BEB2: *No duplication:* No message is delivered more than once. PP2P

BEB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

Algorithm 3.1: Basic Broadcast

Implements:

BestEffortBroadcast, **instance** *beb*.

Uses:

PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle$ *beb*, *Broadcast* | m $\rangle$ **do**
  forall $q \in \Pi$ **do** (process)
    trigger $\langle$ *pl*, *Send* | q, m $\rangle$;

upon event $\langle$ *pl*, *Deliver* | p, m $\rangle$ **do**
  trigger $\langle$ *beb*, *Deliver* | p, m $\rangle$;

Correctness

Validity

- It comes from the *reliable delivery* property of perfect links + the fact that the sender sends the message to every other process in the system.

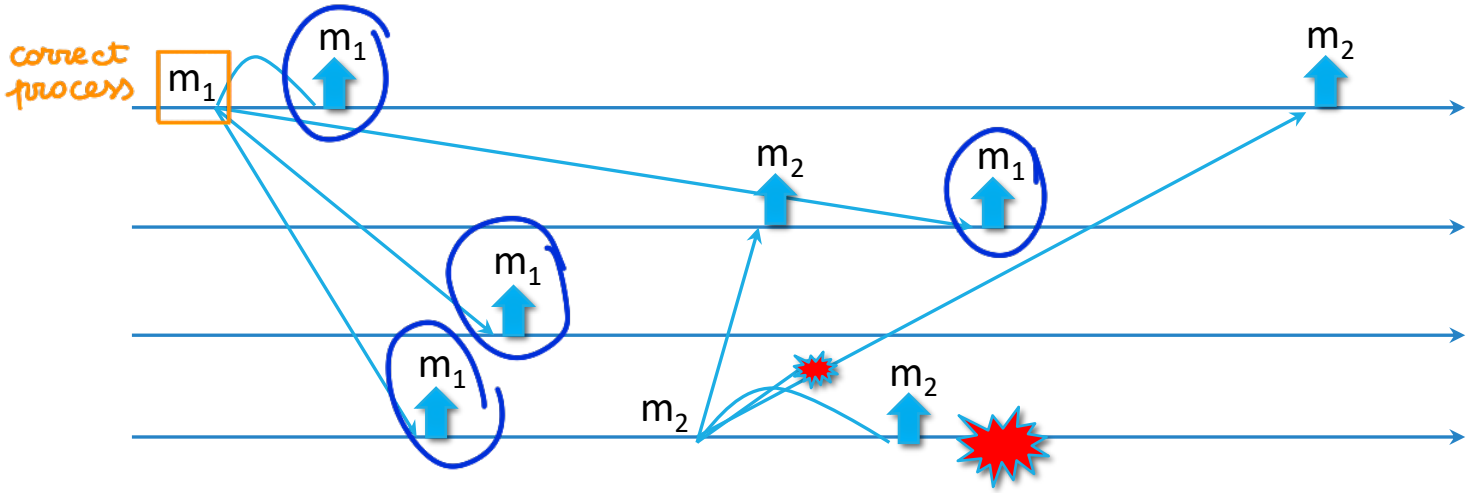
No Duplication

- it directly follows from the No Duplication of perfect links + assumption on the uniqueness of messages (i.e., different messages have different identifiers).

No Creation

- it directly follows from the corresponding property of perfect links.

Observations on Best Effort Broadcast (BEB)



- BEB ensures the delivery of messages as long as the sender does not fail
- If the sender fails processes may disagree on whether or not deliver the message

(Regular) Reliable Broadcast (RB)

Module 3.2: Interface and properties of (regular) reliable broadcast

Module:

Name: ReliableBroadcast, **instance** *rb*.

Events:

Request: ⟨ *rb*, *Broadcast* | *m* ⟩: Broadcasts a message *m* to all processes.

Indication: ⟨ *rb*, *Deliver* | *p*, *m* ⟩: Delivers a message *m* broadcast by process *p*.

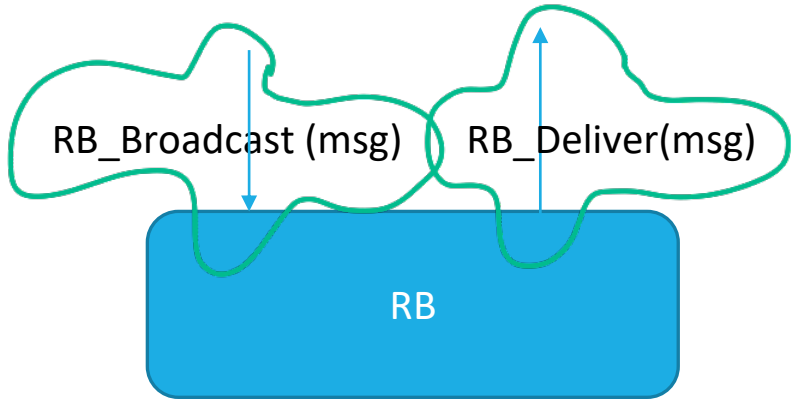
Properties:

RB1: *Validity:* If a correct process *p* broadcasts a message *m*, then *p* eventually delivers *m*.

RB2: *No duplication:* No message is delivered more than once.

RB3: *No creation:* If a process delivers a message *m* with sender *s*, then *m* was previously broadcast by process *s*.

RB4: *Agreement:* If a message *m* is delivered by some correct process, then *m* is eventually delivered by every correct process.

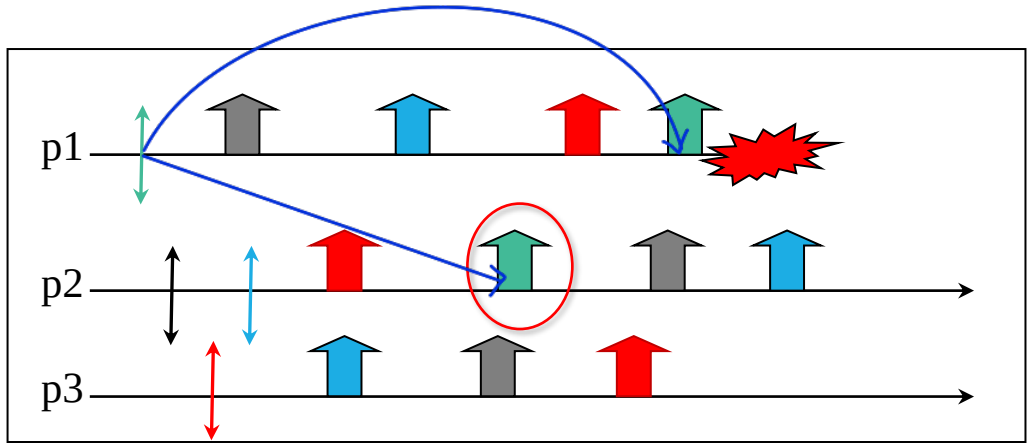


Same as BEB

+

Liveness: agreement

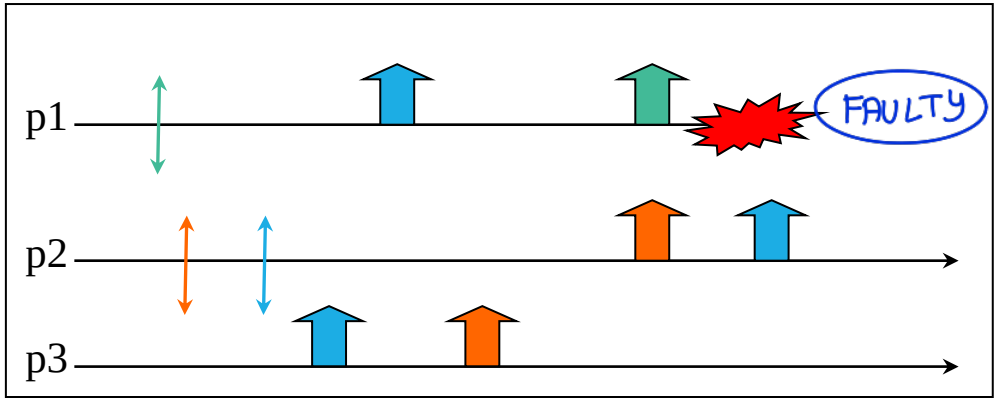
BEB vs RB



every correct process have the same arrows

Satisfies BEB but not RB (violation of the Agreement Property)

Satisfies RB



(Regular) Reliable Broadcast (RB) Implementation in Synchronous Systems

Algorithm 3.2: Lazy Reliable Broadcast

Implements:
ReliableBroadcast, **instance** *rb*.

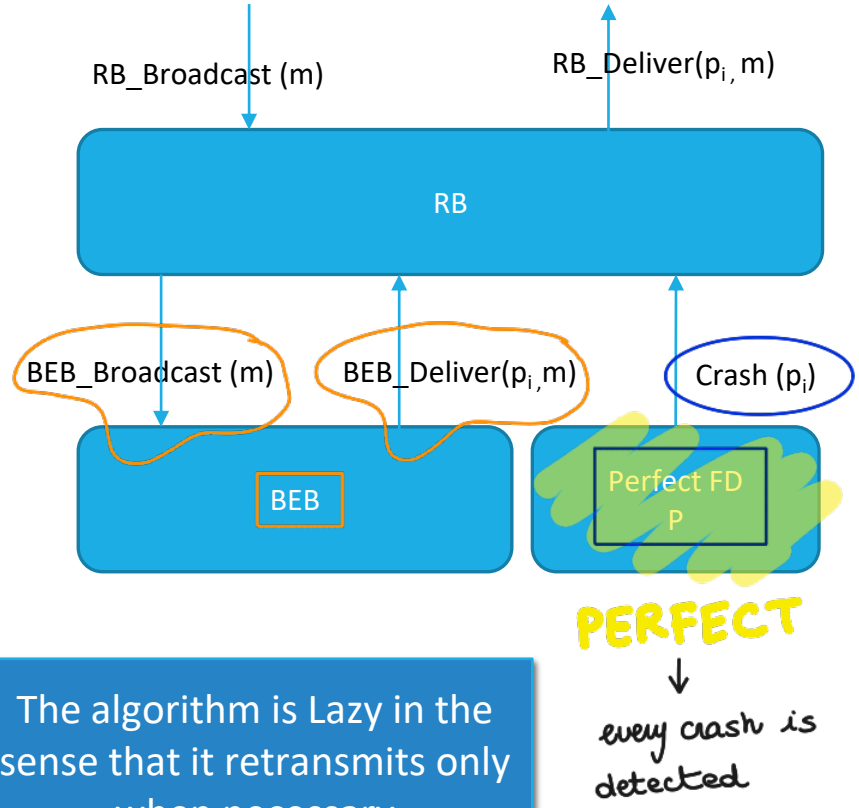
Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *rb*, *Init* $\rangle$ **do**
  $correct := \Pi$;
  $from[p] := [\emptyset]^N$;

upon event $\langle$ *rb*, *Broadcast* | m $\rangle$ **do**
  trigger $\langle$ *beb*, *Broadcast* | [DATA, *self*, m] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [DATA, s, m] $\rangle$ **do**
  if $m \notin from[s]$ **then**
    trigger $\langle$ *rb*, *Deliver* | s, m $\rangle$;
    $from[s] := from[s] \cup \{m\}$;
    if $s \notin correct$ **then**
      trigger $\langle$ *beb*, *Broadcast* | [DATA, s, m] $\rangle$;

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
  $correct := correct \setminus \{p\}$;
  forall $m \in from[p]$ **do**
    trigger $\langle$ *beb*, *Broadcast* | [DATA, p, m] $\rangle$;



The algorithm is Lazy in the sense that it retransmits only when necessary

8. RB synchronous system

Properties:

RB1: *Validity:* If a correct process p broadcasts a message m, then p eventually delivers m.

RB2: *No duplication:* No message is delivered more than once.

RB3: *No creation:* If a process delivers a message m with sender s, then m was previously broadcast by process s.

RB4: *Agreement:* If a message m is delivered by some correct process, then m is eventually delivered by every correct process.

Algorithm 3.2: Lazy Reliable Broadcast

Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle$ *rb*, *Init* $\rangle$ **do**
  $correct := \Pi$; all processes
  $from[p] := [\emptyset]^N$; array; store msgs that you receive from every p

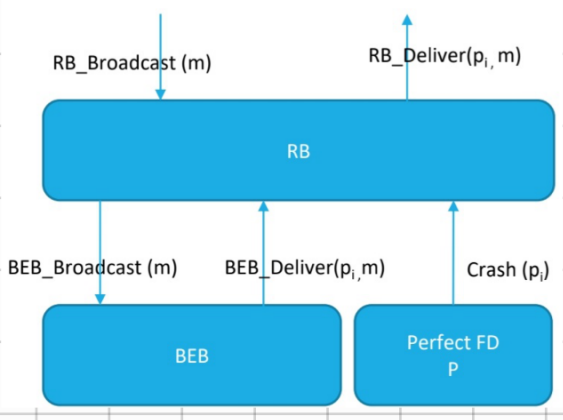
upon event $\langle$ *rb*, *Broadcast* | m $\rangle$ **do**
  trigger $\langle$ *beb*, *Broadcast* | [DATA, *self*, m] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [DATA, s, m] $\rangle$ **do**
  if $m \notin from[s]$ **then** check if it is the 1° time that you are receiving the m from the source s
    trigger $\langle$ *rb*, *Deliver* | s, m $\rangle$; ← TRUE then
    $from[s] := from[s] \cup \{m\}$;
    if $s \notin correct$ **then** check if the source is already correct
      trigger $\langle$ *beb*, *Broadcast* | [DATA, s, m] $\rangle$;

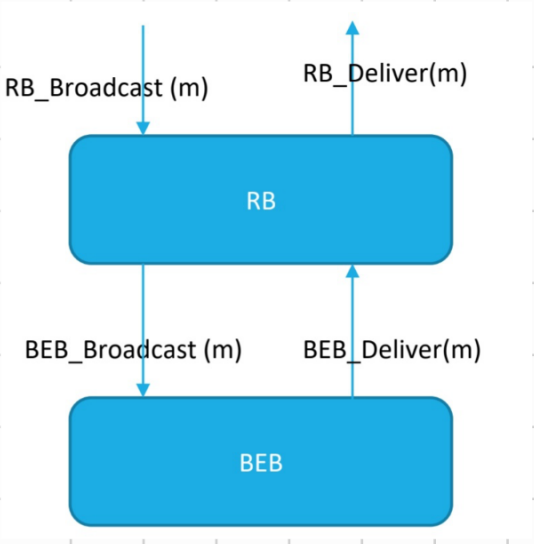
ensure NO DUPLICATION

upon event $\langle$ $\mathcal{P}$, *Crash* | p $\rangle$ **do**
  $correct := correct \setminus \{p\}$;
  forall $m \in from[p]$ **do** check all the msgs received from faulty
    trigger $\langle$ *beb*, *Broadcast* | [DATA, p, m] $\rangle$; retransmitted

→ LAZY



8. RB asynchronous system



Algorithm 3.3: Eager Reliable Broadcast

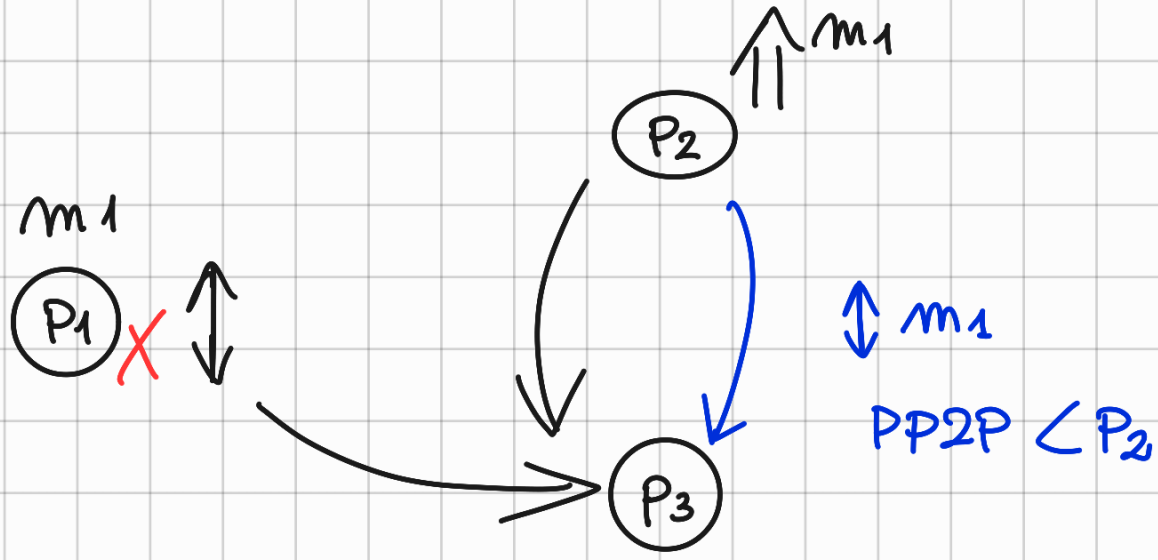
Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*.

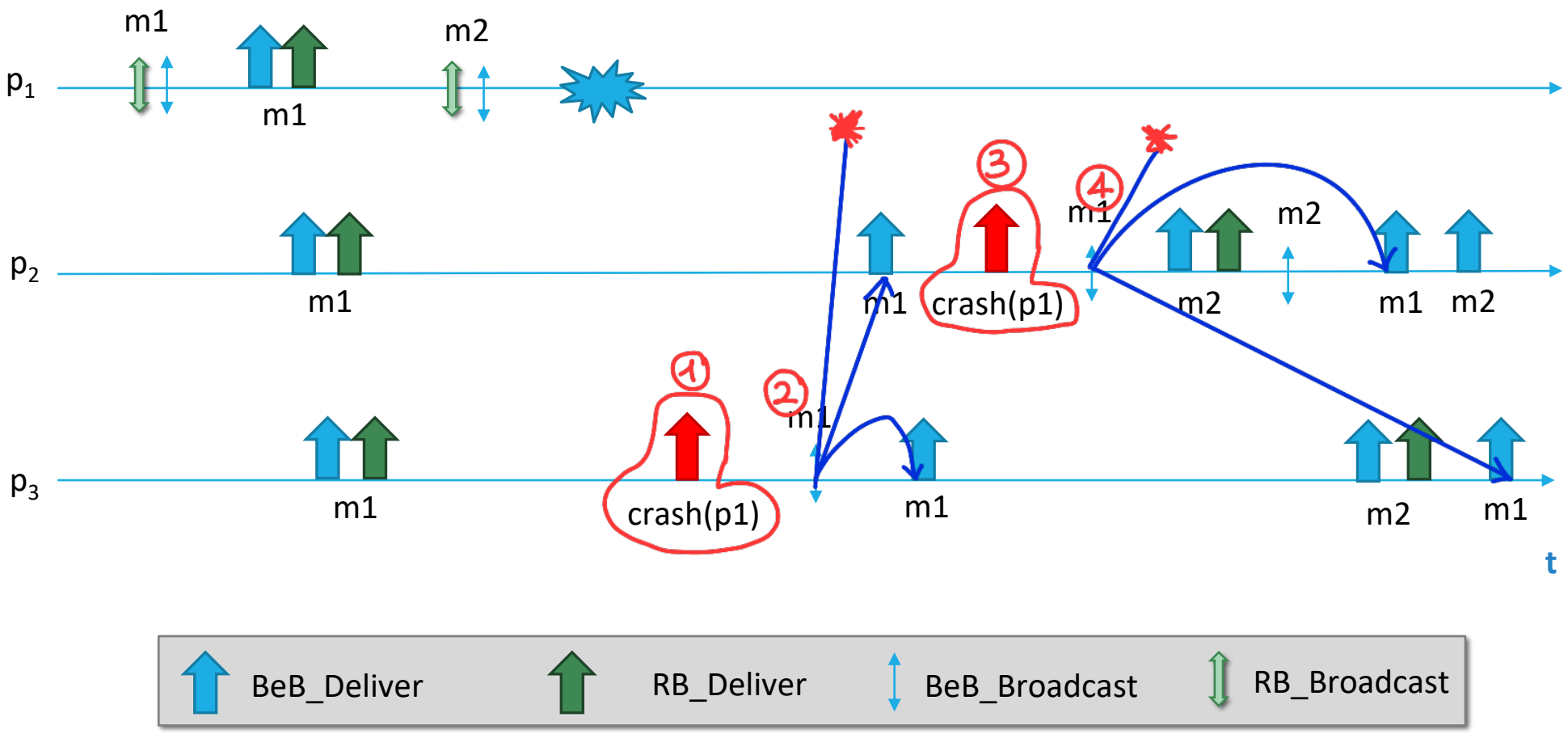
upon event ⟨ *rb*, *Init* ⟩ **do**
delivered := ∅;

upon event ⟨ *rb*, *Broadcast* | *m* ⟩ **do**
trigger ⟨ *beb*, *Broadcast* | [DATA, *self*, *m*] ⟩;

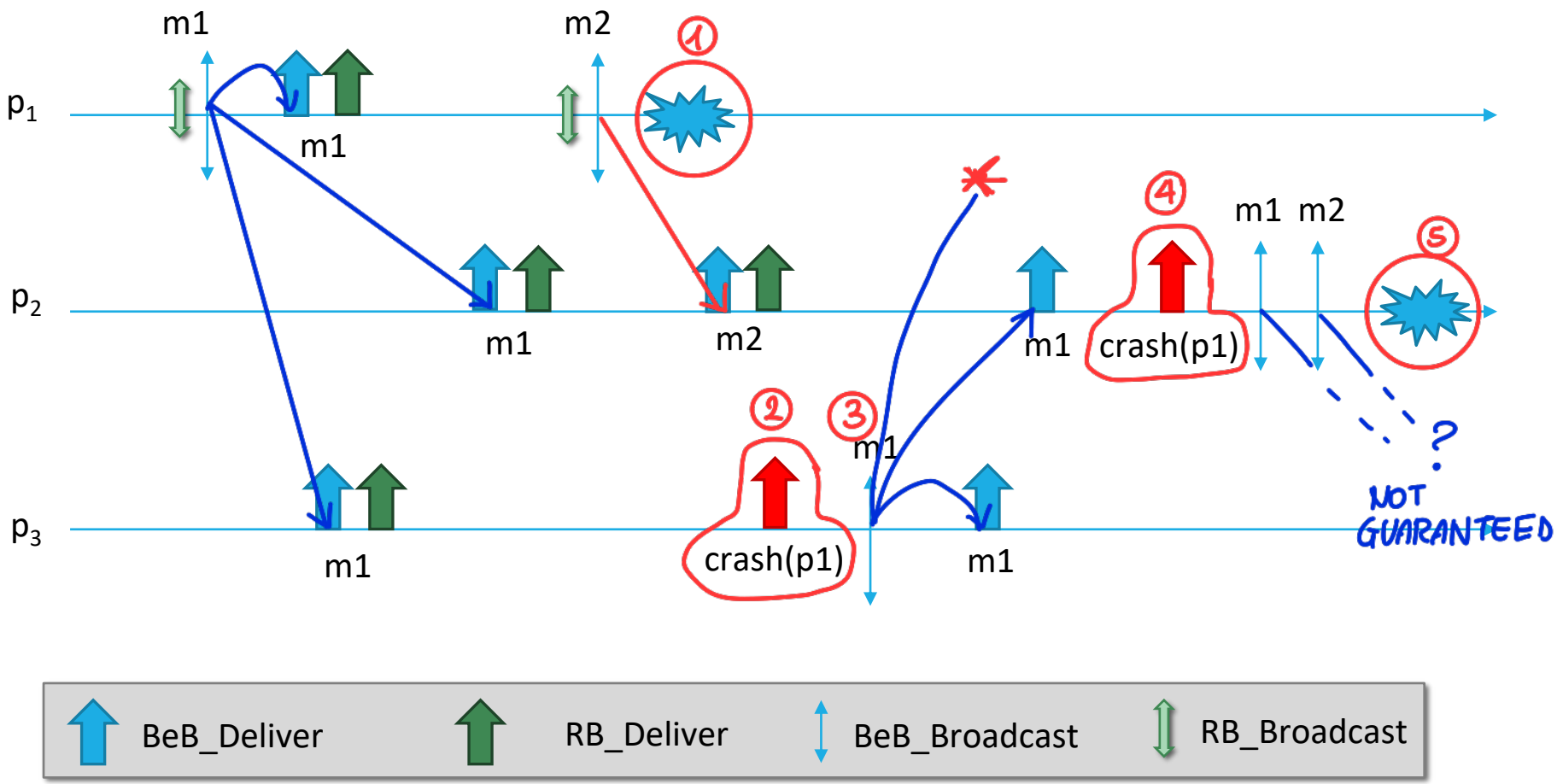
upon event ⟨ *beb*, *Deliver* | *p*, [DATA, *s*, *m*] ⟩ **do**
if *m* ∉ *delivered* **then**
delivered := *delivered* ∪ {*m*};
trigger ⟨ *rb*, *Deliver* | *s*, *m* ⟩;
trigger ⟨ *beb*, *Broadcast* | [DATA, *s*, *m*] ⟩;


m1
P1
m1
P2
P3
m1
PP2P < P2

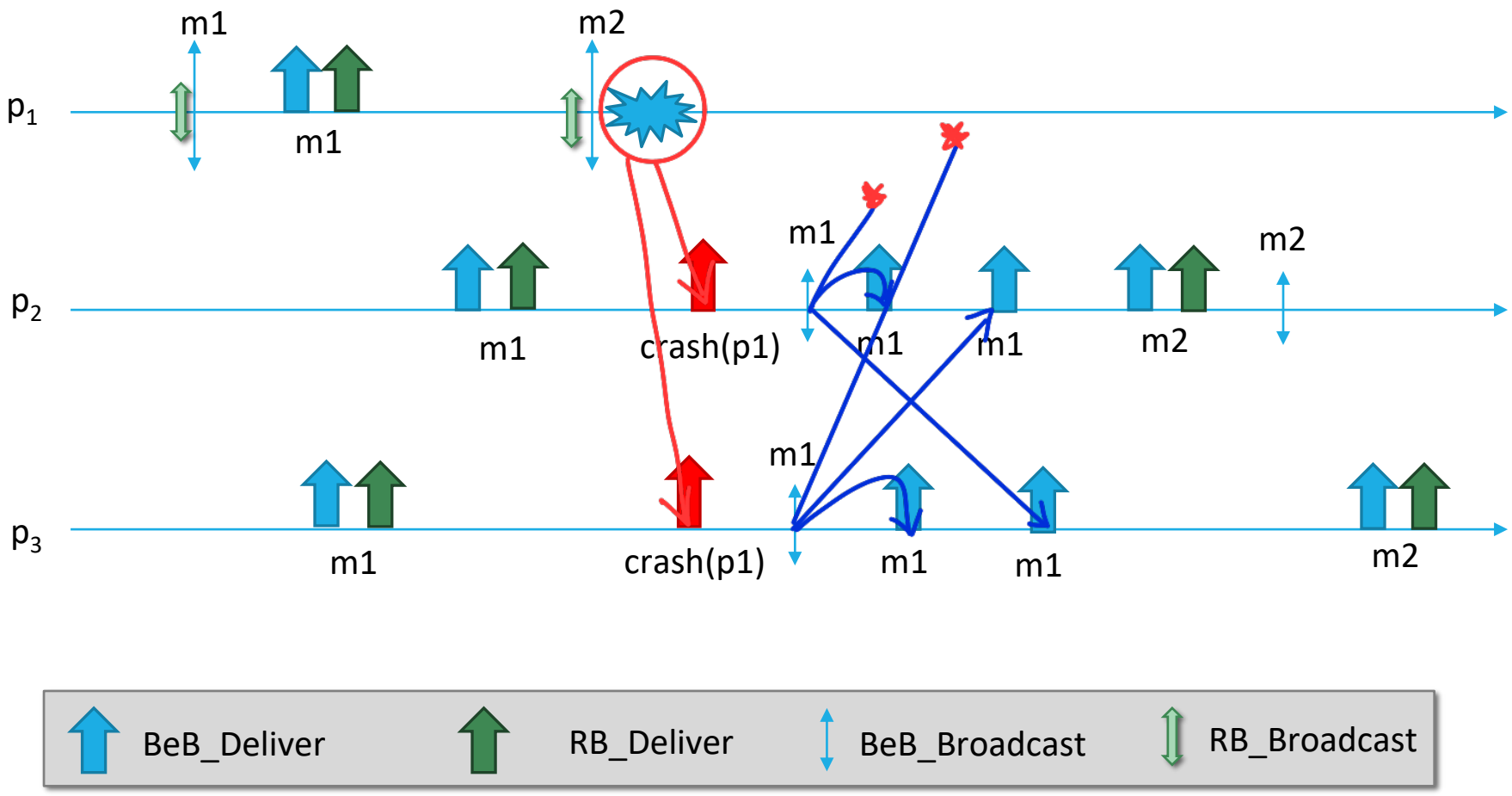
Example


m1
m2
p1
m1
p2
m1
m1
crash(p1)
m1
m2
m1
m2
p3
m1
crash(p1)
m1
m2
m1
t
1
2
3
4
BeB_Deliver
RB_Deliver
BeB_Broadcast
RB_Broadcast

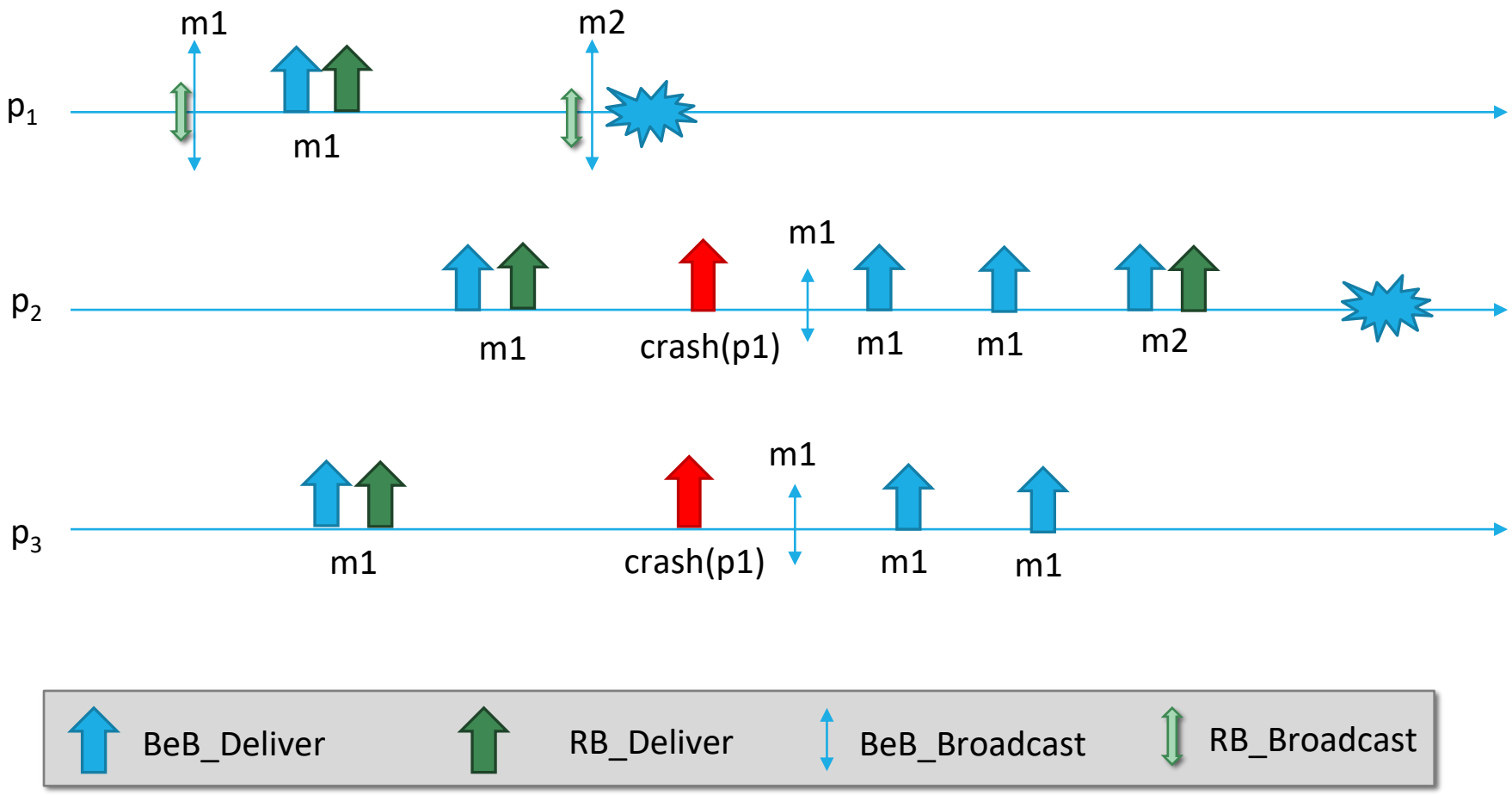
Example


m1
m2
1
p1
m1
m1
m2
4
m1 m2
5
p2
m1
crash(p1)
2
3
m1
?
NOT GUARANTEED
p3
m1
crash(p1)
m1
BeB_Deliver
RB_Deliver
BeB_Broadcast
RB_Broadcast

Example


m1
m2
p1
m1
m1
p2
crash(p1)
m1
m1
m1
m2
m2
m1
p3
m1
crash(p1)
m1
m1
m2
BeB_Deliver
RB_Deliver
BeB_Broadcast
RB_Broadcast

Example


m1
m2
p1
m1
m1
p2
m1
crash(p1)
m1
m1
m2
m1
p3
m1
crash(p1)
m1
m1
BeB_Deliver
RB_Deliver
BeB_Broadcast
RB_Broadcast

Performance of Lazy RB Algorithm

- Best case

  1 BEB message per one RB message

- Worst case

  n-1 BEB messages per one RB (this is the case with n-1 failures)

- What if the FD is not perfect?

(Regular) Reliable Broadcast (RB) Implementation in Asynchronous Systems

Algorithm 3.3: Eager Reliable Broadcast

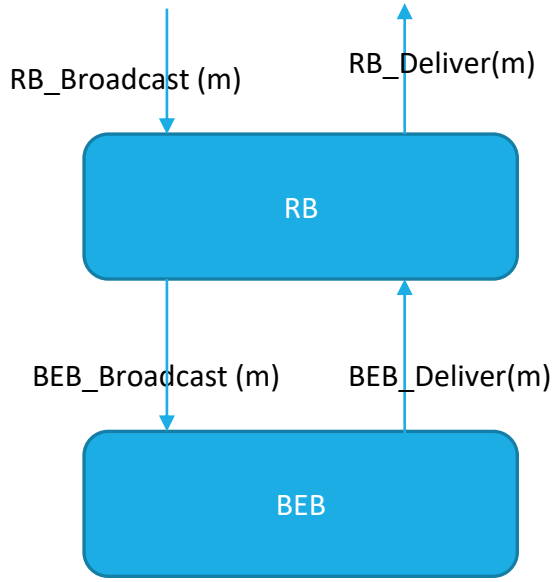
Implements:
ReliableBroadcast, **instance** *rb*.

Uses:
BestEffortBroadcast, **instance** *beb*.

upon event ⟨ *rb*, *Init* ⟩ **do**
delivered := ∅;

upon event ⟨ *rb*, *Broadcast* | m ⟩ **do**
trigger ⟨ *beb*, *Broadcast* | [DATA, *self*, m] ⟩;

upon event ⟨ *beb*, *Deliver* | p, [DATA, s, m] ⟩ **do**
if $m \notin$ *delivered* **then**
delivered := *delivered* ∪ {m};
trigger ⟨ *rb*, *Deliver* | s, m ⟩;
trigger ⟨ *beb*, *Broadcast* | [DATA, s, m] ⟩;



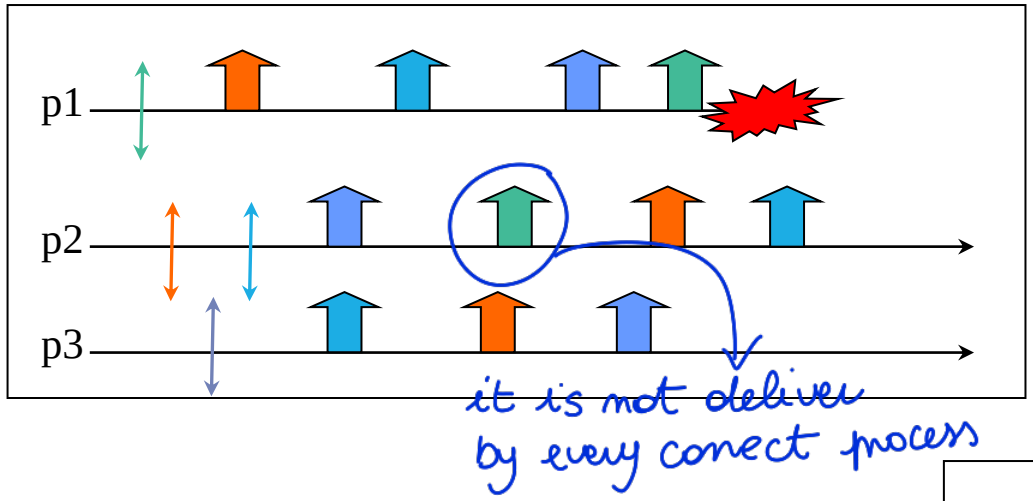
The algorithm is Eager in the sense that it retransmits every message

Performance of Eager RB Algorithm

- Best case = Worst case

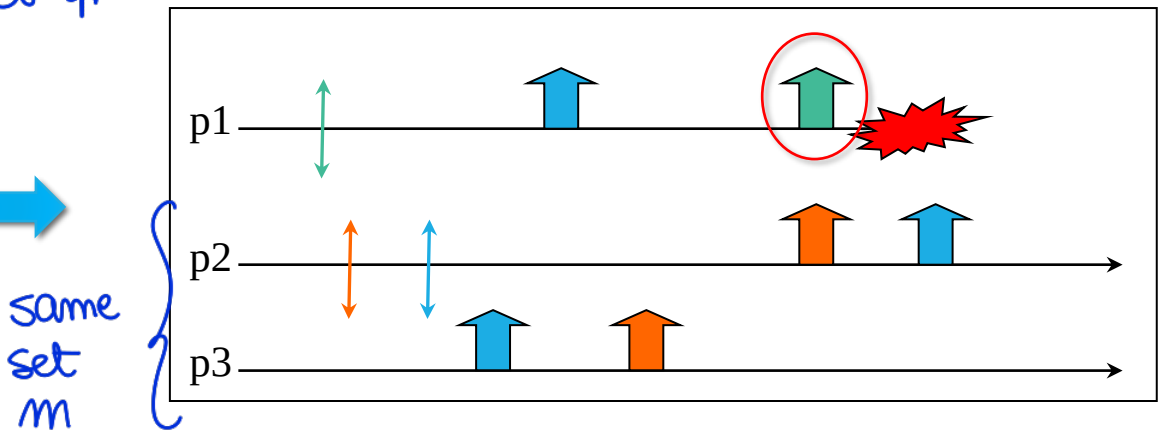
  n BEB messages per one RB

BEB vs RB

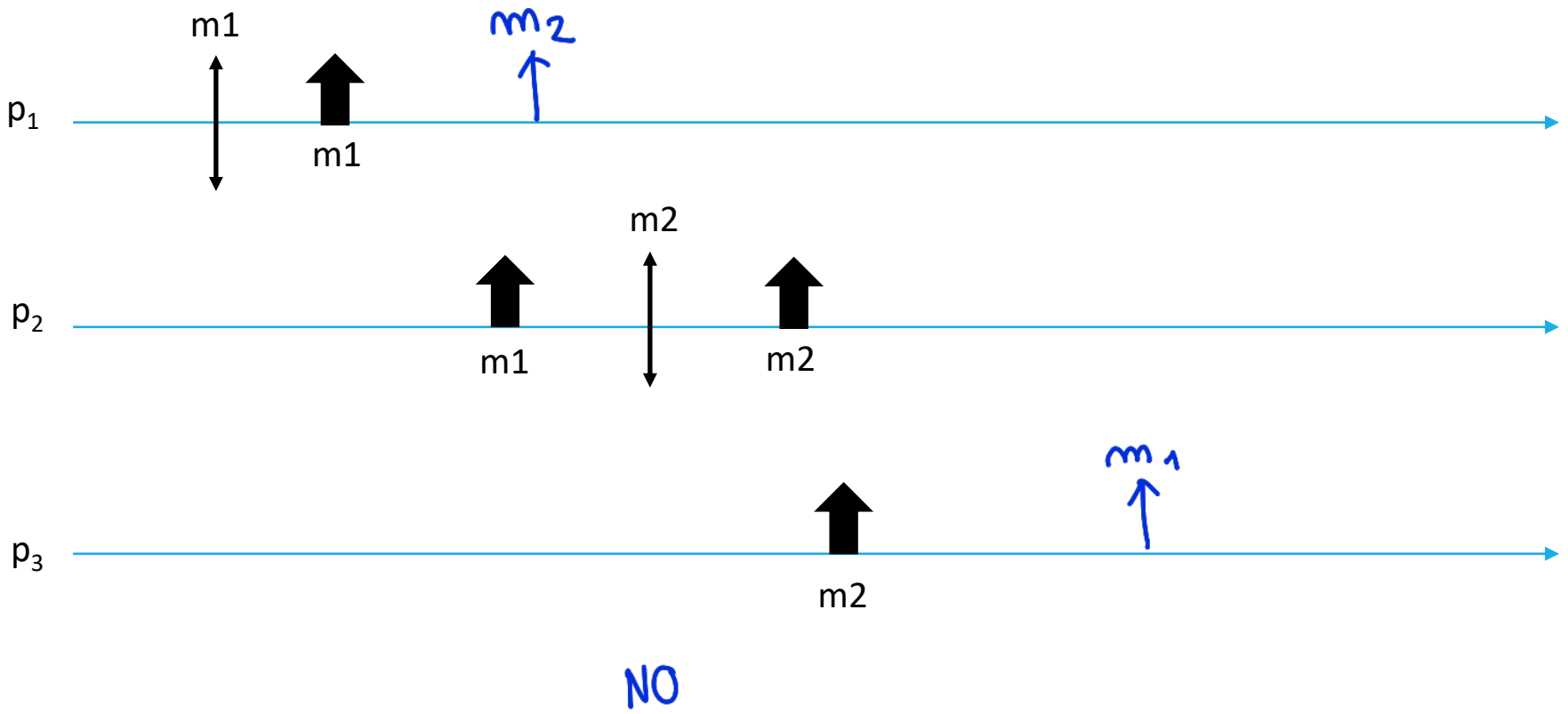


Satisfies BEB but not RB
(violation of the Agreement
Property)

Satisfies RB

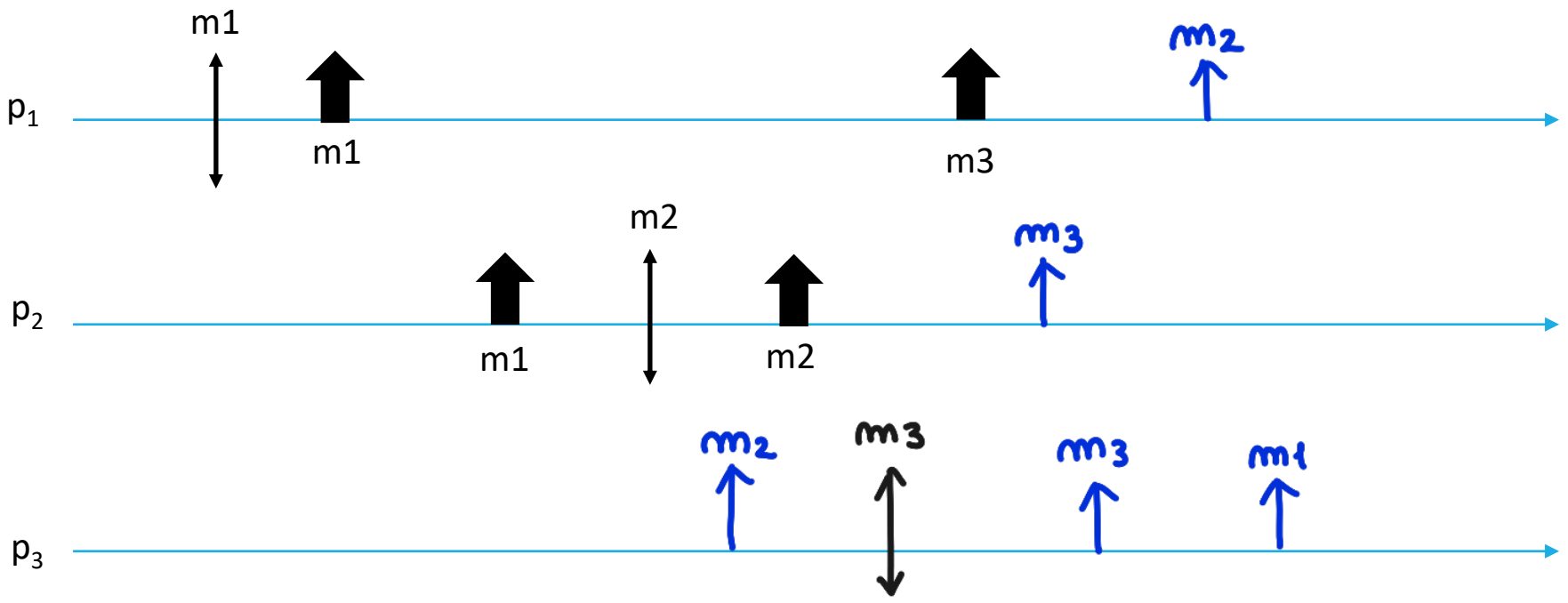


Exercise 1



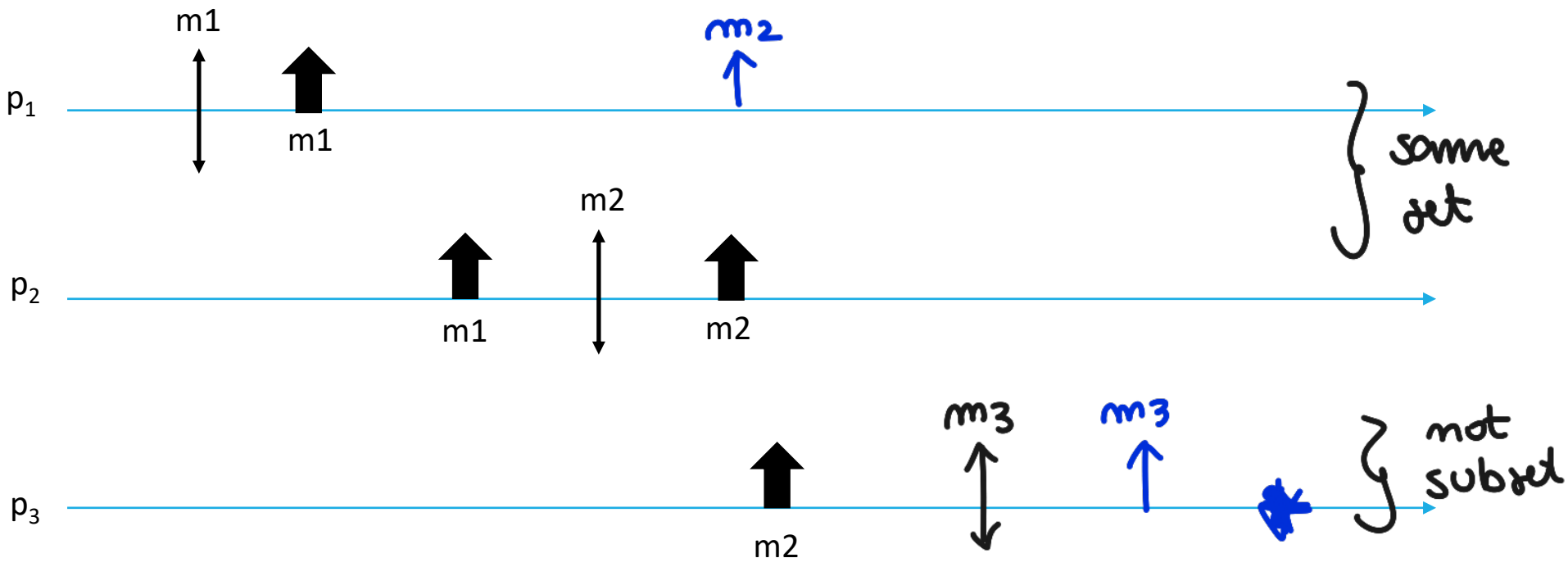
Does the execution satisfy the BEB specification? If not, complete the execution by adding BEB-broadcast, beb-deliveries and/or crash faults

Exercise 2



Does the execution satisfy the BEB specification? If not, complete the execution by adding BEB-broadcast, BEB-deliveries and/or crash faults

Exercise 3



Does the execution satisfy the regular RB specification? If not, complete the execution by adding RB-broadcast, RB-deliveries and/or crash faults.
If not, is it possible complete the execution by adding one single crash fault to satisfy the RB specification?