Dependable and Distributed Systems

Professor Matthew Leeke
School of Computer Science
University of Birmingham

Topic 10 - Reliable Broadcast Algorithms

The Toolkit

Reliable Links

Ensure every correct nodes receives every messages sent by a correct node

Failure Detectors

Devices that indicate operation states of nodes

Failures modes

Crash failures

Broadcasting Abstractions

Best Efforts Broadcast

Guarantees reliable delivery depending on correctness of sender process

Reliable Broadcast

Guarantees independent of sender process

Uniform Reliable Broadcast

Ensures delivery also for faulty processes (if possible)

Broadcasting Abstractions

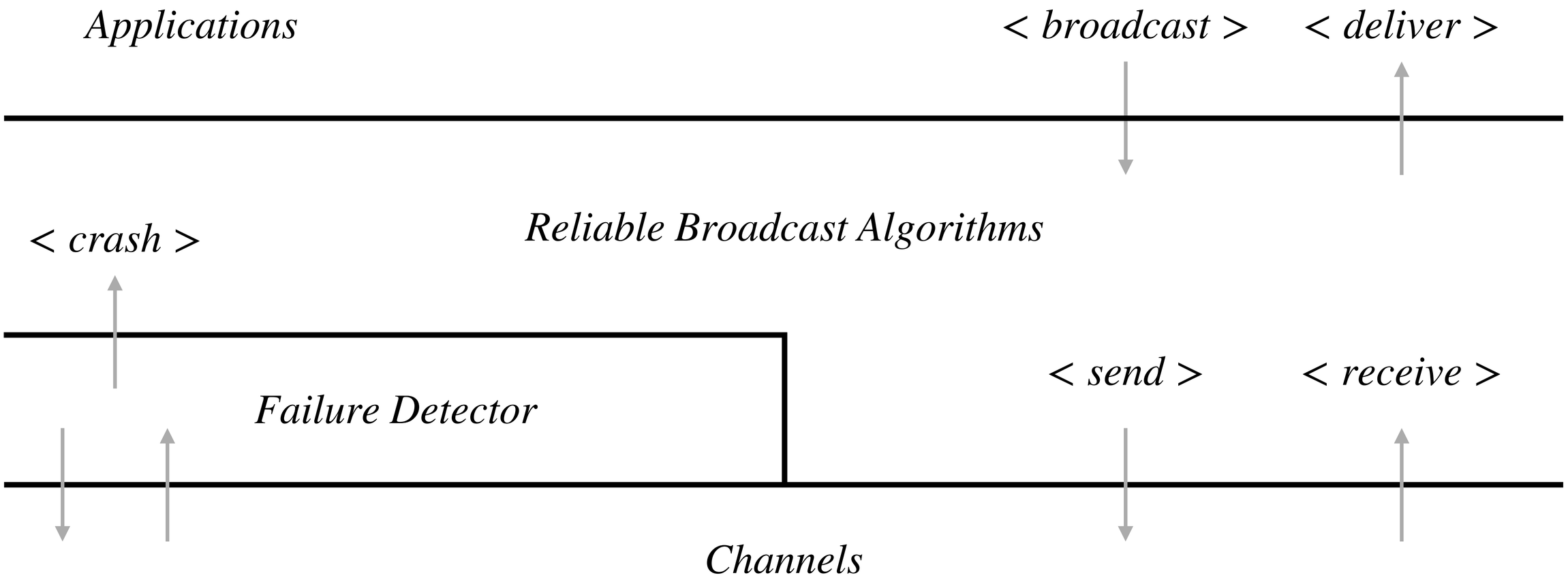
FIFO Reliable Broadcast

Reliable broadcast with FIFO delivery order

Causal Reliable Broadcast

Reliable broadcast with "global FIFO" (causal) delivery order

Broadcasting From A Process Perspective



Intuition

Broadcast is useful, for instance, in applications where some processes subscribe to events published by other processes

Or processes need to know about some important events happening

The subscribers might require some reliability guarantees from the broadcast service (we say quality of service – QoS) that the underlying network does not provide

Message losses

Overview

We first consider three forms of reliability for a broadcast primitive

1. Best-effort broadcast
2. Reliable broadcast
3. Uniform reliable broadcast

First specifications and then algorithms

Later we will consider different delivery orders

Best Efforts Broadcast

Best-Effort Broadcast (BEB)

Events

Request: $< BEBBroadcast, m >$

Indication: $< BEBDeliver, src, m >$

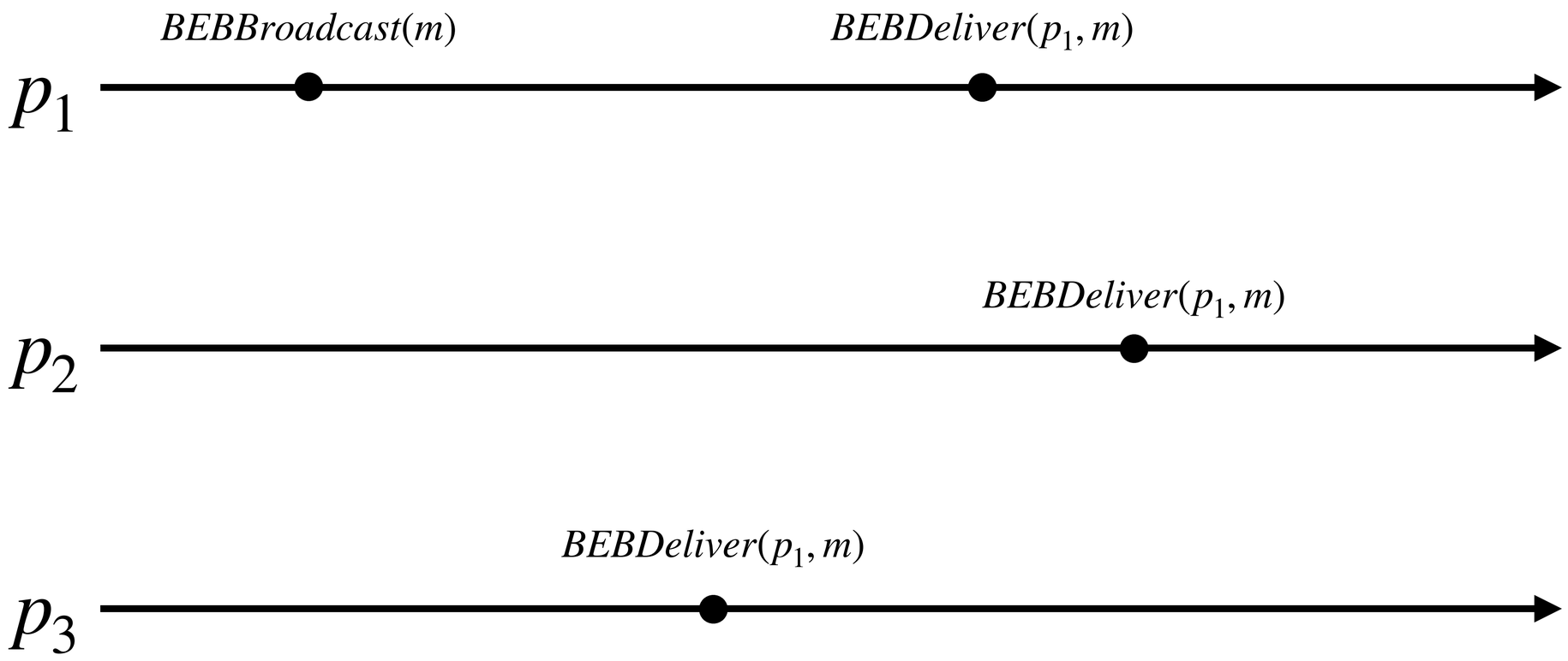
Properties

BEB Property 1 (Validity) - If p_i and p_j are correct then every message broadcast by p_i is eventually delivered by p_j

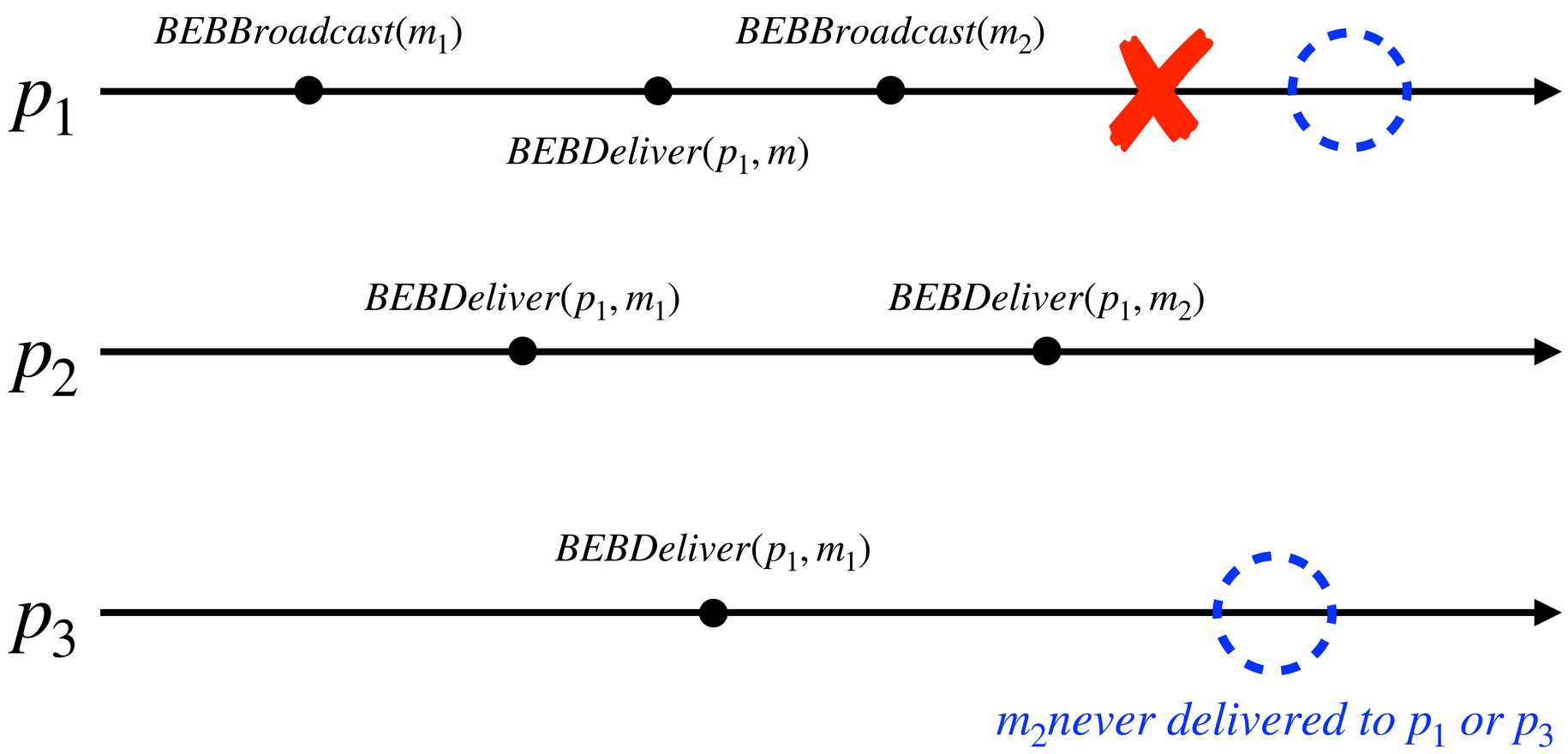
BEB Property 2 (No duplication) - No message is delivered more than once

BEB Property 3 (No creation) - No message is delivered unless it was broadcast

BEB Example 1 - Fault-Free



BEB Example 1 - Faulty Sender



Guarantees

Reliable delivery only if both sender and receiver are correct

No guarantees otherwise

Some may deliver messages and some may not

We would like to guarantee that at least all correct processes deliver the message even if the a sender fails

Now The BEB Algorithm

For the corresponding algorithm we need:

Perfect links - Safety and liveness

Perfect failure detector - Accuracy and completeness

BEB Algorithm

$Implements : BestEffortsBroadcast\ (BEB)$

$Uses : ReliableLinks\ (rp2p)$

$upon\ event\ < BEBBroadcast, m > do$

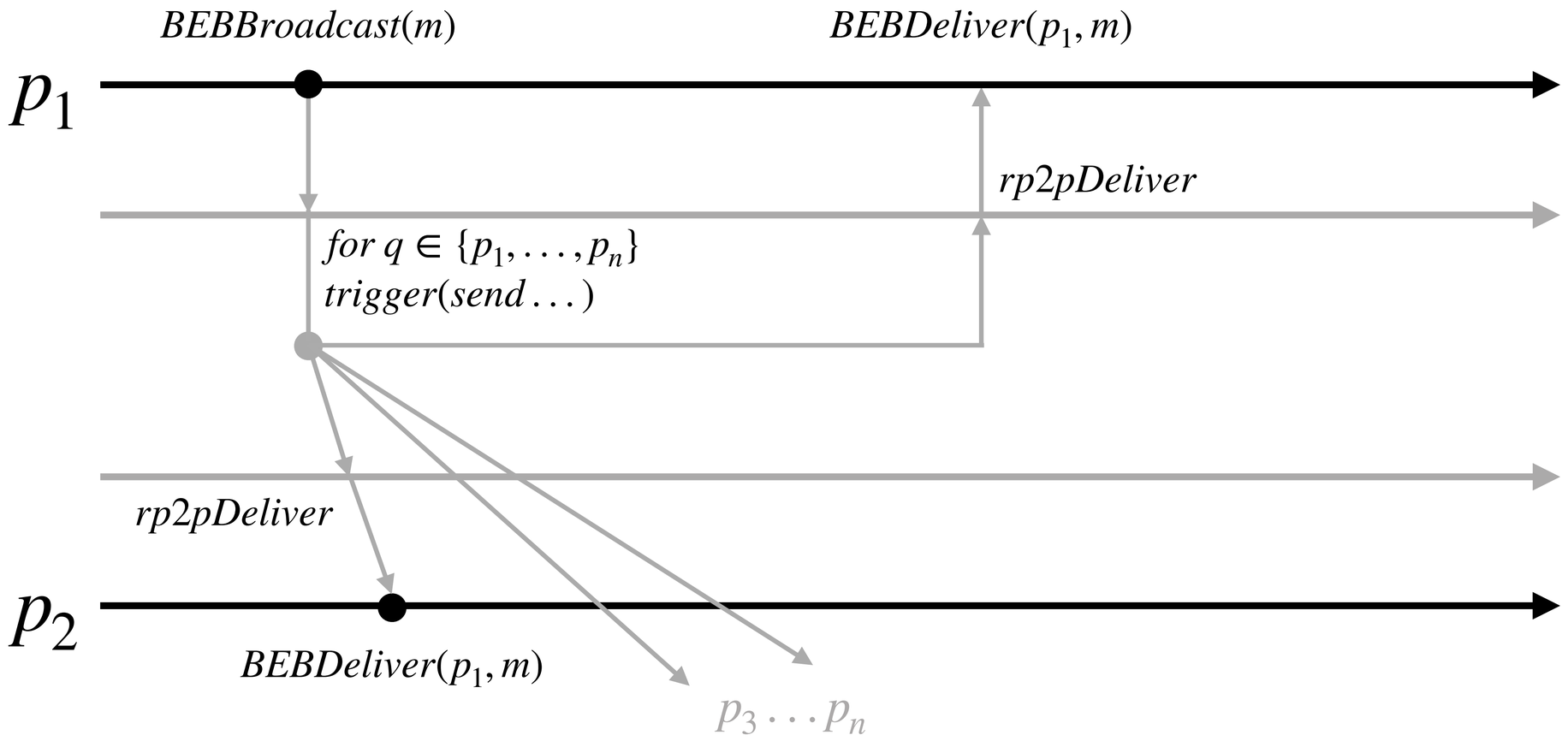
$\quad for\ all\ q \in \{p_1, \ldots, p_n\}\ do$

$\quad\quad trigger < rp2pSend, q, m >$

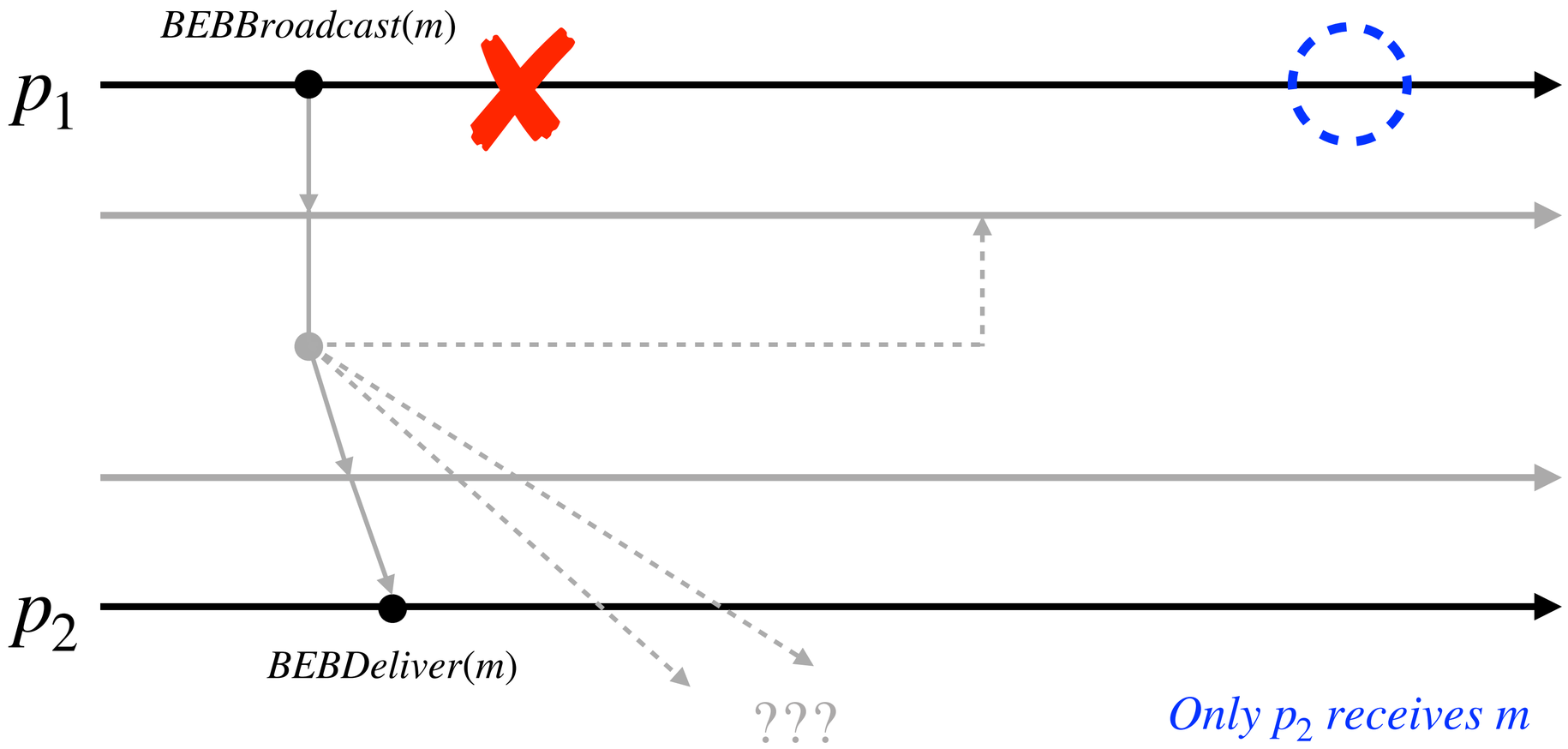
$upon\ event\ < rp2pDeliver, q, m > do$

$\quad trigger\ < BEBDeliver, q, m >$

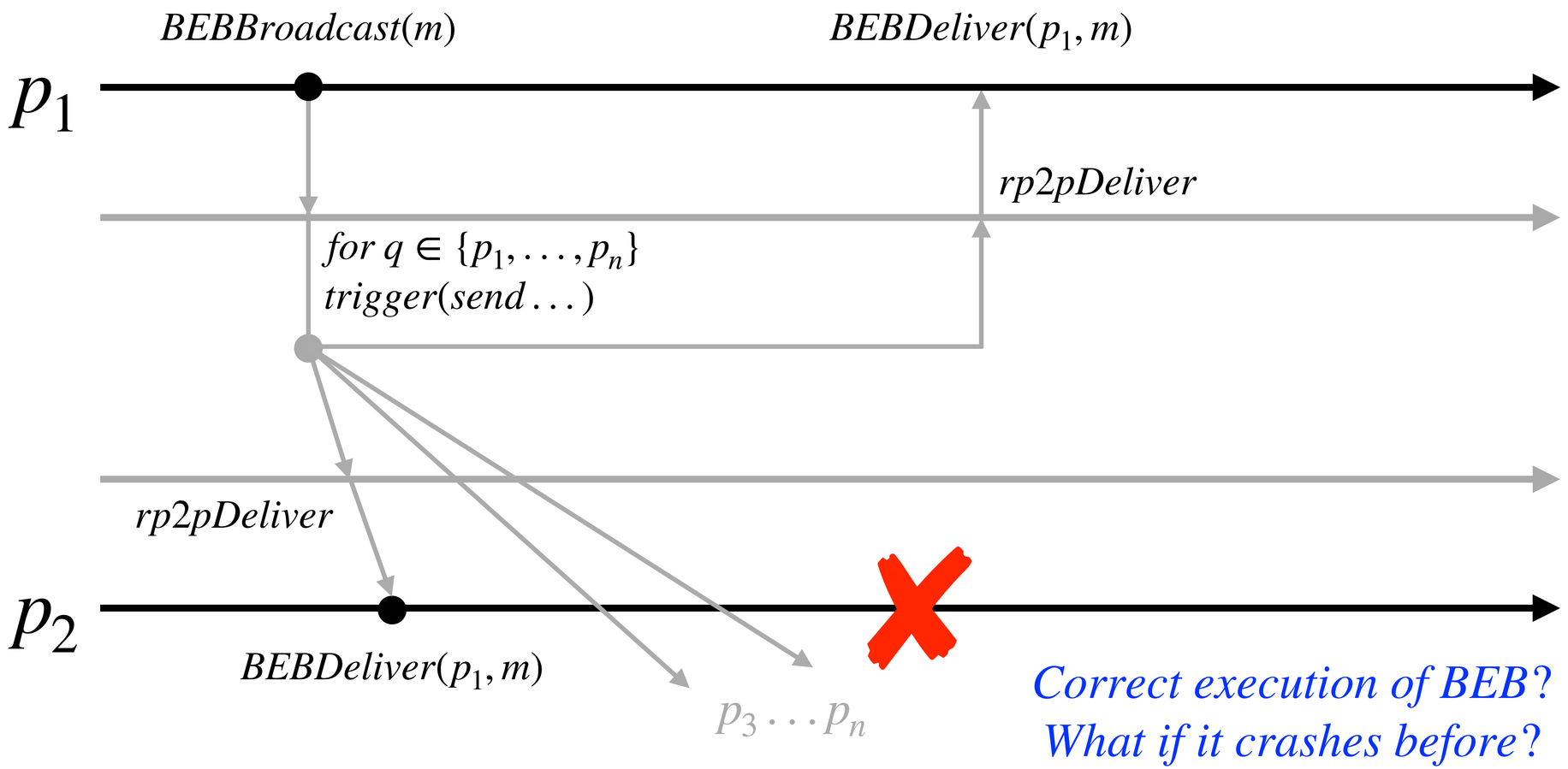
BEB Algorithm 1 - Fault-Free



BEB Algorithm 2 - Sender Crash



BEB Algorithm 3 - Receiver Crash



BEB - Correctness

BEB Property 1 (Validity) - If p_i and p_j are correct then every message broadcast by p_i is eventually delivered by p_j

BEB Property 1 (No duplication) - No message is delivered more than once

BEB Property 1 (No creation) - No message is delivered unless it was broadcast

We also have the properties of reliable links

Reminder - Reliable Link Properties

Reliable Link Property 1 (Validity) - If p_i and p_j are correct then every message sent by p_i to p_j is eventually delivered to p_j

Reliable Link Property 2 (No Duplication) - No message is delivered (to a process) more than once

Reliable Link Property 2 (No Creation) - No message is delivered unless it was sent

Reminder - Reliable Links Algorithm

$$
\begin{aligned}
&\textit{Implements} : \textit{ReliableLinks}\ (rp2p)\\
&\textit{Uses} : \textit{StubbornLinks}\ (sp2p)\\
&\\
&\textit{upon event}\ < \textit{Init} > \textit{do}\\
&\qquad \textit{delivered} = \{\}\\
&\\
&\textit{upon event}\ < rp2pSend, dest, m > \textit{do}\\
&\qquad \textit{trigger}\ < sp2pSend, dest, m >\\
&\\
&\textit{upon event}\ < sp2pDeliver, src, m > \textit{do}\\
&\qquad \textit{if}\ m \notin \textit{delivered}\\
&\qquad\qquad \textit{trigger}\ < rp2pDeliver, src, m >\\
&\qquad\qquad \textit{delivered} = \textit{delivered} \cup \{m\}
\end{aligned}
$$

What Can We Conclude About BEB Correctness?

BEB Property 1 (Validity)

By the validity property of perfect links and the facts that (i) the sender sends the message to all and (ii) every correct process that *rp2pDelivers* a message *bebDelivers* it

BEB Property 2 (No duplication)

By the no duplication property of perfect links

BEB Property 3 (No creation)

By the no creation property of the perfect links

Reliable Broadcast

The Toolkit Grows

Reliable Links

Ensure every correct nodes receives every messages sent by a correct node

Failure Detectors

Devices that indicate operation states of nodes

Failures modes

Crash failures

BEB

Broadcasting Abstractions

Best Efforts Broadcast

Guarantees reliable delivery depending on correctness of sender process

Reliable Broadcast

Guarantees independent of sender process

Uniform Reliable Broadcast

Ensures delivery also for faulty processes (if possible)

Broadcasting Abstractions

FIFO Reliable Broadcast

Reliable broadcast with FIFO delivery order

Causal Reliable Broadcast

Reliable broadcast with "global FIFO" (causal) delivery order

Best Efforts to Reliable Broadcast

We want guarantees even if the sender fails

BEB does not guarantee that much

We also need to relate our algorithm to correct receivers

Can only a subset of correct receivers deliver?

Can they each deliver a different sets of messages?

Assuming they recovered like databases servers, we'd have inconsistency!

We need BED plus something else

Best Efforts Broadcast (BEB)

Events

Request: $< BEBBroadcast, m >$

Indication: $< BEBDeliver, src, m >$

Properties

BEB Property 1 (Validity) - If p_i and p_j are correct, then every message broadcast by p_i is eventually delivered by p_j

BEB Property 2 (No duplication) - No message is delivered more than once

BEB Property 3 (No creation) - No message is delivered unless it was broadcast

Reliable Broadcast (RB)

Events

Request: $< RBroadcast, m >$

Indication: $< RDeliver, src, m >$

Properties

RB Property 1 (Validity) - BEB Property 1

RB Property 2 (No duplication) - BEB Property 2

RB Property 3 (No creation) - BEB Property 3

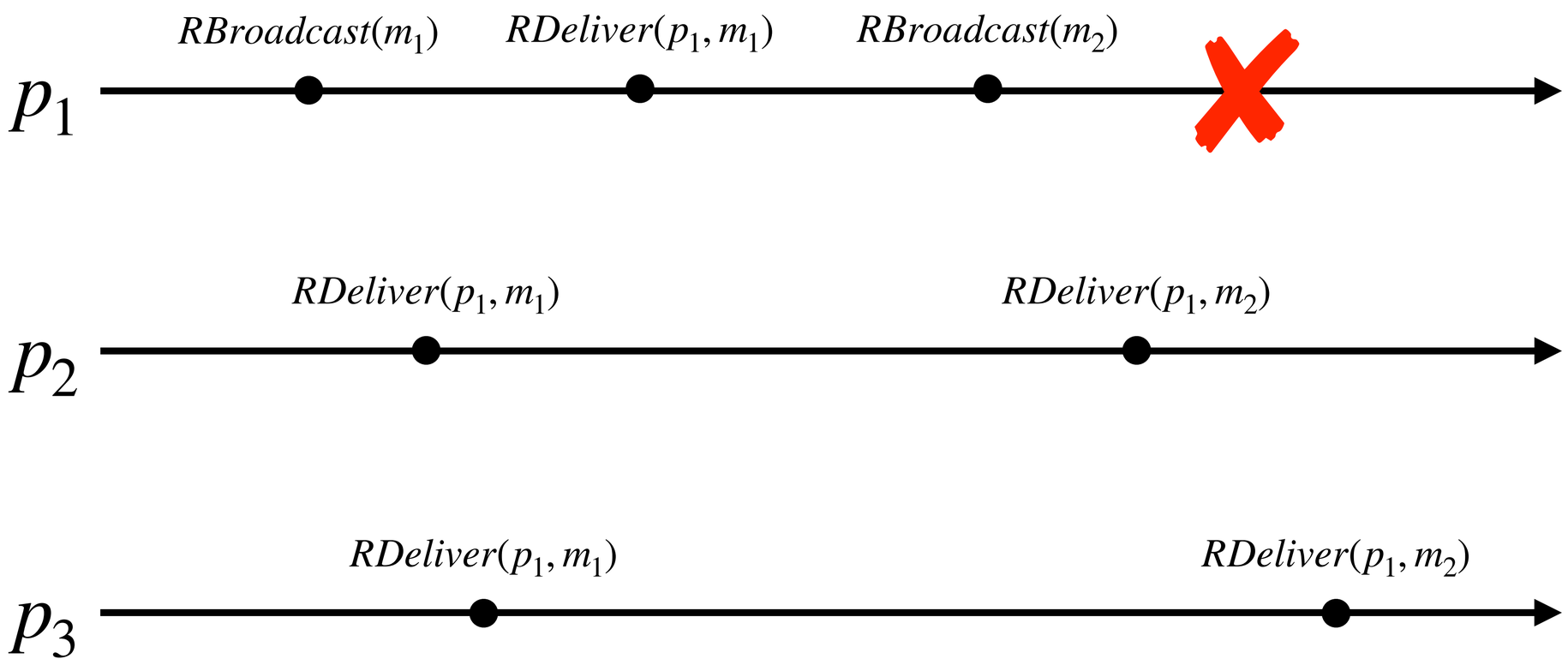
RB Property 4 (Agreement) - For any message m, if a correct process delivers m, every correct process delivers m

RB Intuition

Reliable broadcast gives "all or nothing" guarantee

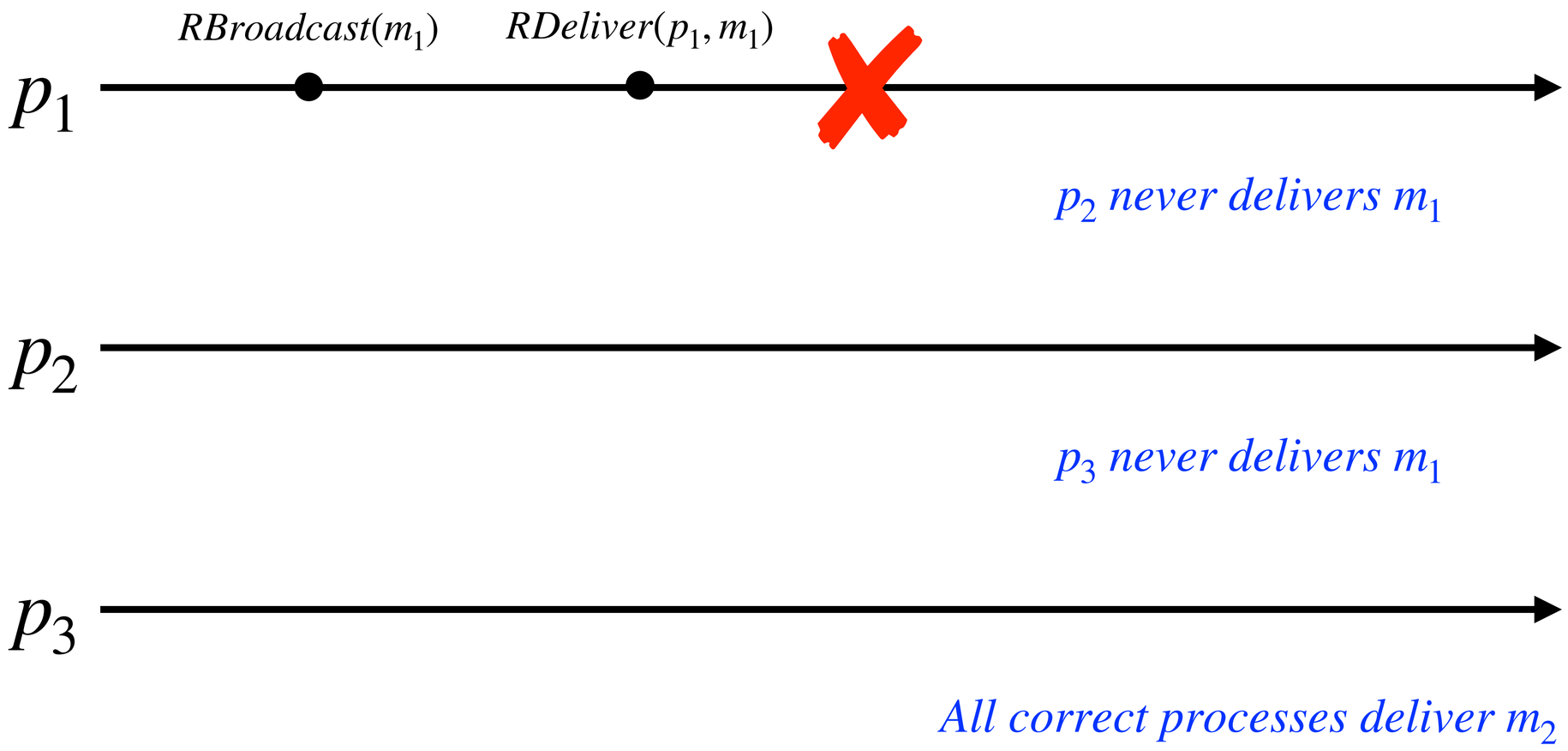
Is the Agreement property a safety or liveness property?

RB Example 1 - Sender Crash

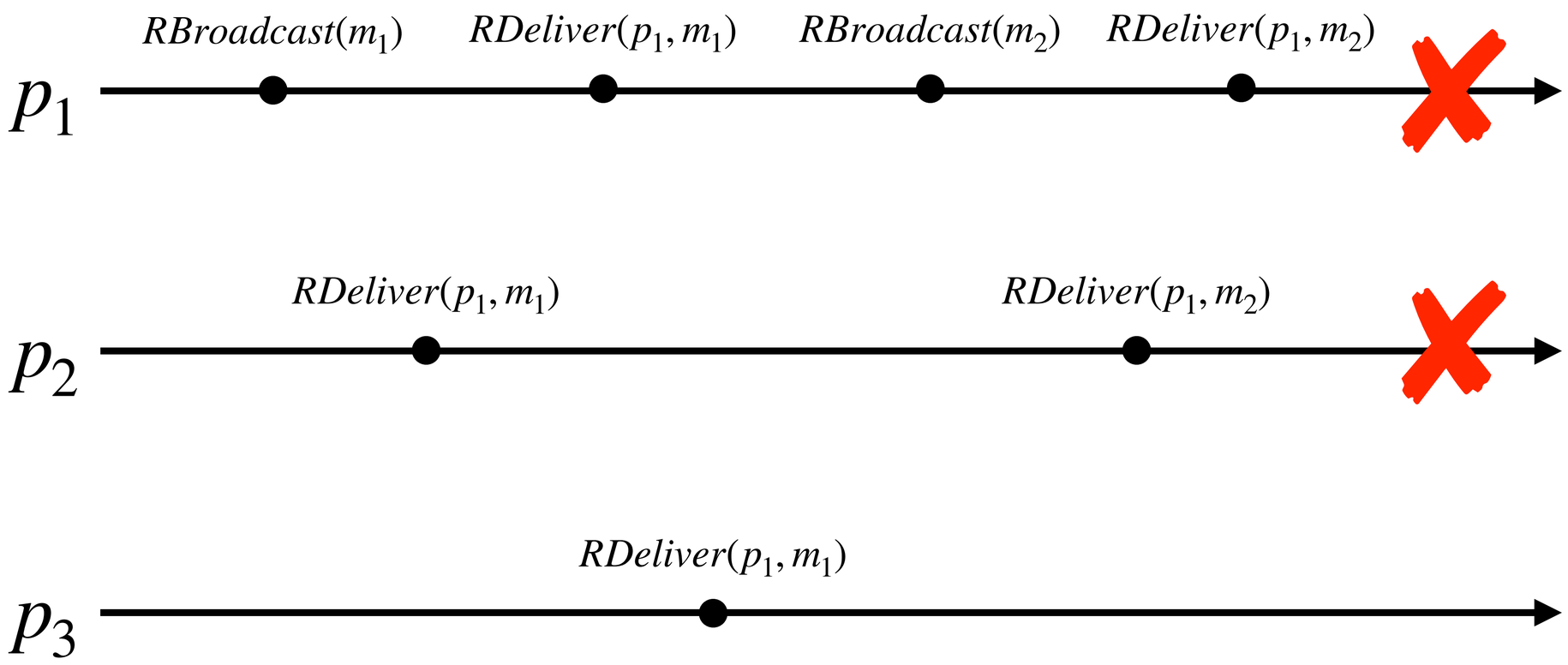


All correct processes deliver m_2

RB Example 2 - Sender Crash



RB Example 3 - Multiple Crashes



No correct process delivers m_2

All faulty processes deliver m_2 (we don't care)

RB Notes

Only correct processes are required to agree

Faulty processes may deliver or not deliver

Can we do better?

Deliver message to all processes which are able to deliver it?

We can and we will (with Uniform Reliable Broadcast)

Now The RB Algorithm

For the corresponding algorithm we need:

BEB

Perfect links - Safety and liveness

Perfect failure detector - Accuracy and completeness

Translate call to Broadcast to BEBBroadcast, which is great if the sender is correct

Hence we detect faulty senders using perfect failure detector

RB Algorithm

$Implements: ReliableBroadcast\ (RB)$

$Uses: BestEffortsBroadcast\ (BEB), PerfectFailureDetector\ (P)$

$upon\ event\ < Init > do$

$delivered = \{\}$

$correct = \{p_1, \ldots, p_n\}$

$for\ all\ p \in S\ do$

$from[p_i] = \{\}$

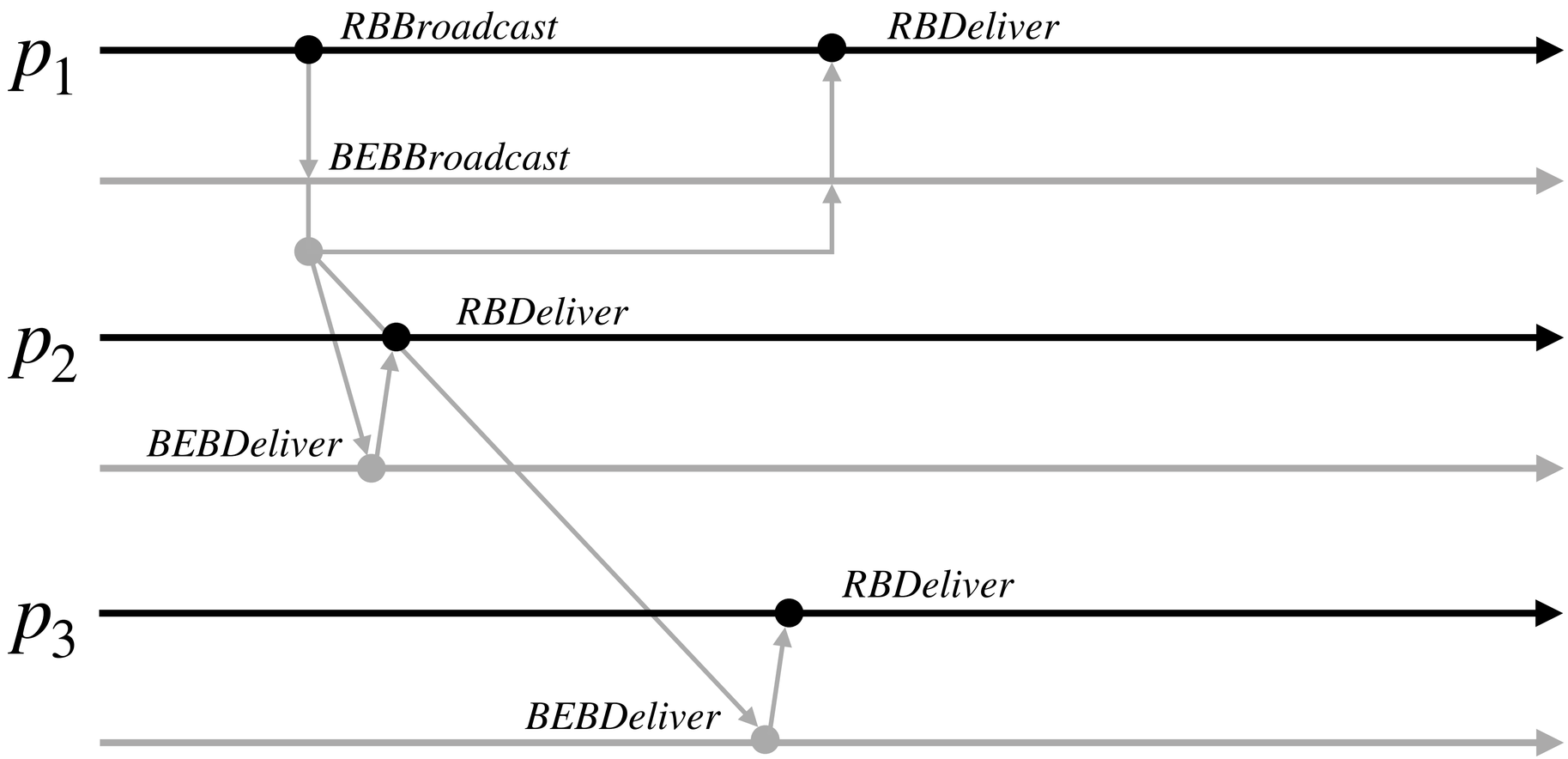
$upon\ event\ < RBBroadcast, m > do$

$trigger < BEBBroadcast, [Data, self, m] >$

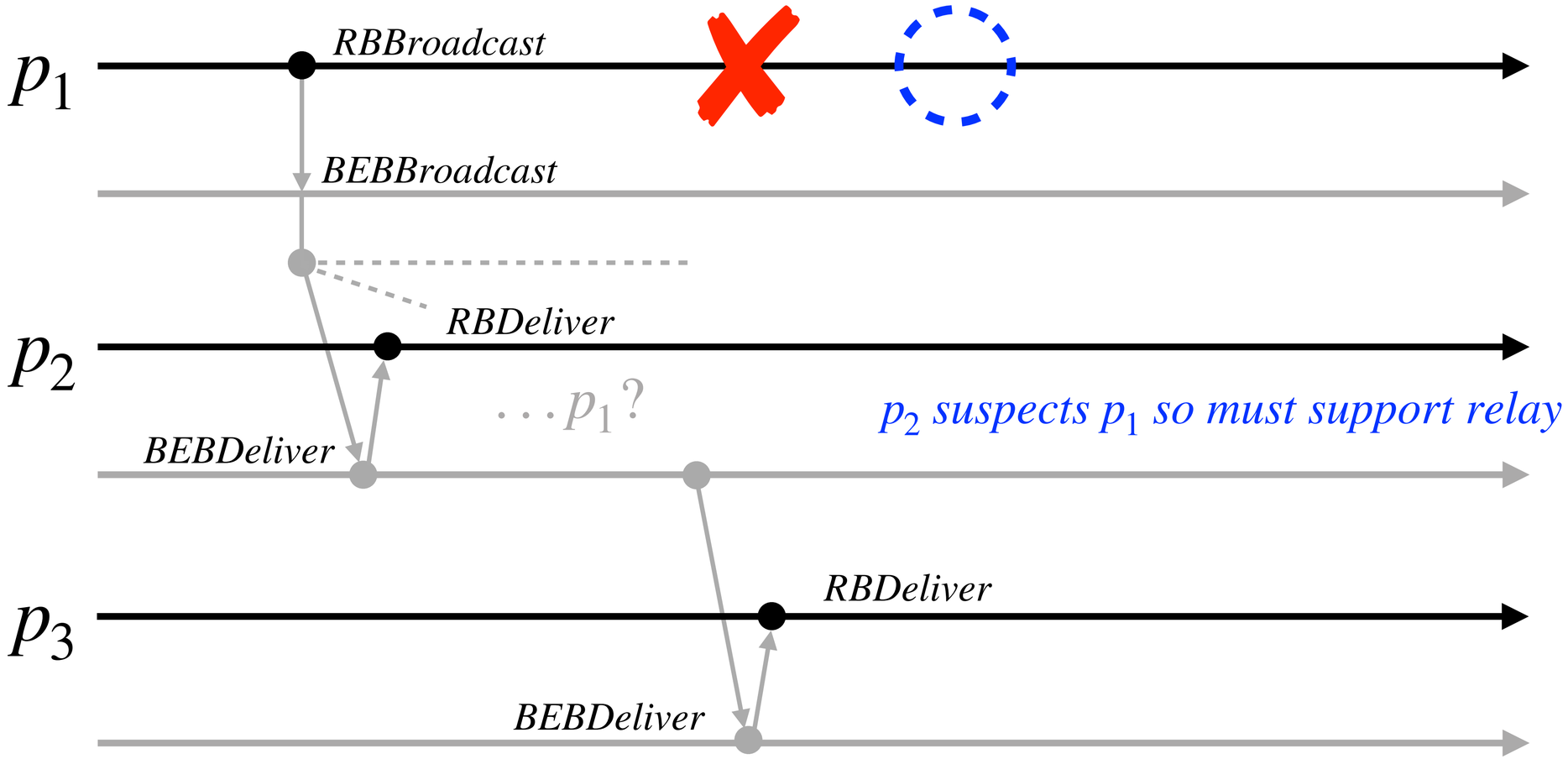
RB Algorithm

$$
\begin{aligned}
&\textit{upon event } < crash, p_i > \textit{ do} \\
&\quad correct = correct \setminus \{p_i\} \\
&\quad \textit{for all } [p_j, m] \in [p_i] \textit{ do} \\
&\quad\quad trigger < BEBBroadcast, [Data, p_j, m] > \\
&\\
&\textit{upon event } < BEBDeliver, p_i, [Data, p_j, m] > \textit{ do} \\
&\quad \textit{if } m \notin delivered \textit{ then} \\
&\quad\quad delivered = delivered \cup \{m\} \\
&\quad\quad trigger < RBDeliver, p_j, m > \\
&\quad\quad \textit{if } p_i \notin correct \textit{ then} \\
&\quad\quad\quad trigger < BEBBroadcast, [Data, p_j, m] > \\
&\quad\quad \textit{if} \\
&\quad\quad\quad from[p_i] = from[p_i] \cup \{[p_j, m]\}
\end{aligned}
$$

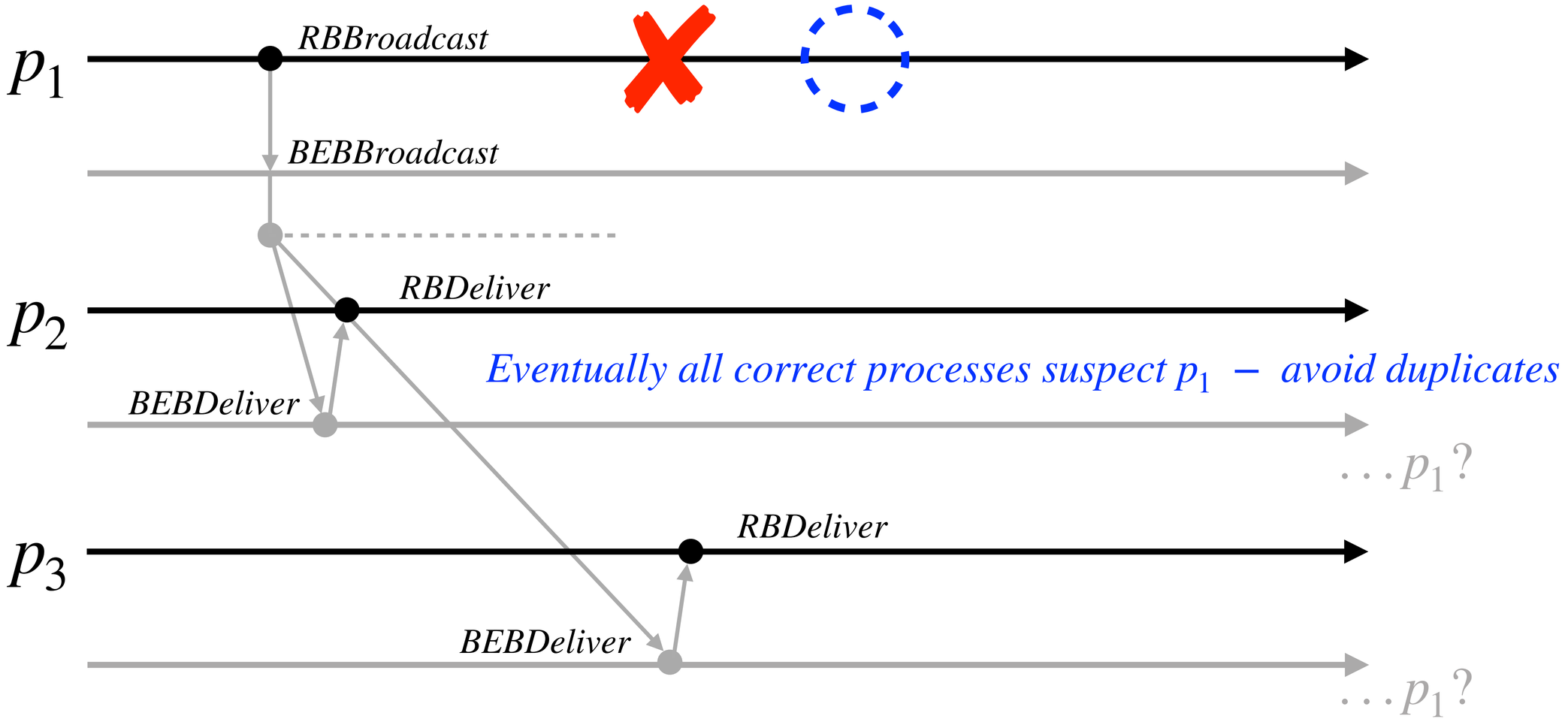
RB Algorithm 1 - Fault Free



RB Algorithm 2 - Faulty Sender



Multiple Deliveries? Check for Duplicates?



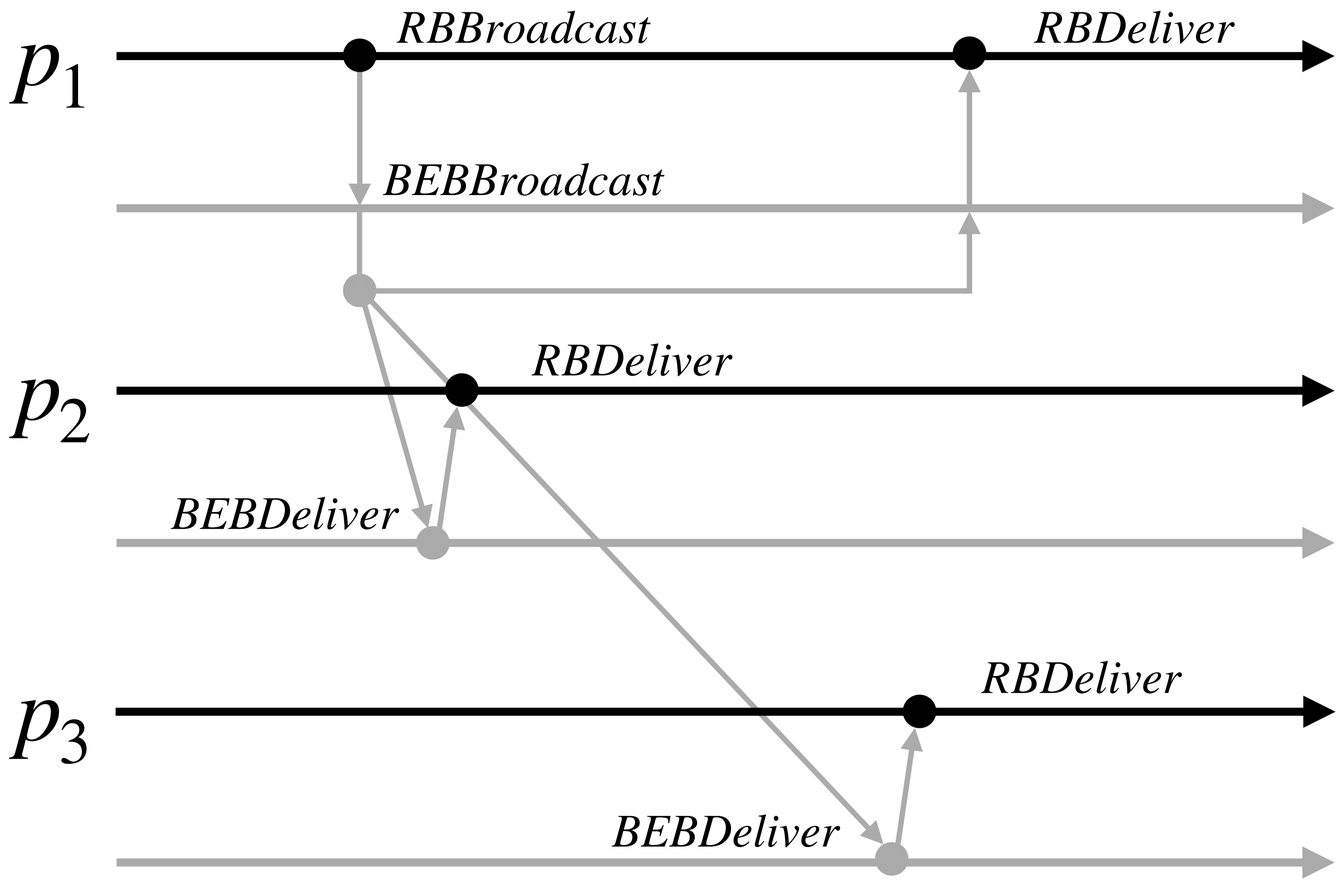
RB Algorithm - Complexity Analysis

Best case: Fault free execution

Exactly N messages

Worst case: Every process is required to relay messages to intermittently failing processes across the network

Exactly N^2 messages



Uniform Reliable Broadcast

Broadcasting Abstractions

Best Efforts Broadcast

Guarantees reliable delivery depending on correctness of sender process

Reliable Broadcast

Guarantees independent of sender process

Uniform Reliable Broadcast

Ensures delivery also for faulty processes (where possible)

Broadcasting Abstractions

FIFO Reliable Broadcast

Reliable broadcast with FIFO delivery order

Causal Reliable Broadcast

Reliable broadcast with "global FIFO" (causal) delivery order

Best Efforts Broadcast (BEB)

Events

Request: $< BEBBroadcast, m >$

Indication: $< BEBDeliver, src, m >$

Properties

BEB Property 1 (Validity) - If p_i and p_j are correct, then every message broadcast by p_i is eventually delivered by p_j

BEB Property 2 (No duplication) - No message is delivered more than once

BEB Property 3 (No creation) - No message is delivered unless it was broadcast

Reliable Broadcast (RB)

Events

Request: $< RBroadcast, m >$

Indication: $< RDeliver, src, m >$

Properties

RB Property 1 (Validity) - BEB Property 1

RB Property 2 (No duplication) - BEB Property 2

RB Property 3 (No creation) - BEB Property 3

RB Property 4 (Agreement) - For any message m, if a correct process delivers m, every correct process delivers m

Reliable Broadcast to Uniform Reliable Broadcast

Only correct processes are required to agree

Fault processes may deliver or not deliver

Can we do better?

Deliver messages to all process that are able to deliver

Uniform Reliable Broadcast (URB)

Events

Request: $< URBroadcast, m >$

Indication: $< URDeliver, src, m >$

Properties

RB Property 1 (Validity) - BEB Property 1

RB Property 2 (No duplication) - BEB Property 2

RB Property 3 (No creation) - BEB Property 3

RB Property 4 (Uniform Agreement) - For any message m, if a process delivers m, every correct process delivers m

Eager Implementation for Reliable Broadcast

Translate **all** instances of *RBroadcast* to *BEBBroadcast*

Relay all messages (no matter whether they come from a correct or fault process)

Be careful to eliminate duplicates

URB Algorithm Idea

Adapt an eager implementation of reliable broadcast

Deliver message m only if all currently alive process have acknowledged it

If all alive processes have $BEBDeliver(m)$ then all correct processes $BEBDeliver(m)$

We only have to ensure that they eventually also $URBDeliver(m)$

URB Algorithm

Implements : *UniformReliableBroadcast* (*URB*)

Uses : *BestEffortsBroadcast* (*BEB*), *PerfectFailureDetector* (*P*)

upon event $< Init >$ *do*

$delivered = \{\}$

$pending = \{\}$

$correct = \{p_1, \ldots, p_n\}$

for all messages m do

$ack[m] = \{\} = \{\}$

upon event $< crash, p_i >$ *do*

$correct = correct \setminus \{p_i\}$

URB Algorithm

$$
\begin{aligned}
&\textit{upon event } < \textit{URBBroadcast}, m > \textit{ do}\\
&\qquad \textit{pending} = \textit{pending} \cup \{[\textit{self}, m]\}\\
&\qquad \textit{trigger} < \textit{BEBBroadcast}, [\textit{Data}, \textit{self}, m] >\\
&\\
&\textit{upon event } < \textit{BEBDeliver}, p_i, [\textit{Data}, p_j, m] > \textit{ do}\\
&\qquad \textit{ack} = \textit{ack}[m] \cup \{p_i\}\\
&\qquad \textit{if } [p_j, m] \notin \textit{pending then}\\
&\qquad\qquad \textit{pending} = \textit{pending} \cup \{[p_j, m]\}\\
&\qquad\qquad \textit{trigger} < \textit{BEBBroadcast}, [\textit{Data}, p_j, m] >\\
&\\
&\textit{upon event } < [p_j, m] \in \textit{pending} : \textit{correct} \subseteq \textit{ack}[m] \wedge m \notin \textit{delivered} > \textit{ do}\\
&\qquad \textit{delivered} = \textit{delivered} \cup \{m\}\\
&\qquad \textit{trigger} < \textit{URBBroadcast}, p_j, m >
\end{aligned}
$$

URB Algorithm - Correctness

If some process delivers m, why must all other processes also eventually deliver m?

We can provide a lemma relating to a simpler and more specific statement

Lemma: If a correct process p_i performs $BEBDeliver(m)$ then eventually $URBDelivers(m)$

Proof: Any correct process that performs $BEBDeliver(m)$ also performs $BEBBroadcast(m)$

By BEB1 there is a time at which p_i performs $BEBDeliver(m)$ for every correct process

For the completeness of our failure detector we know any fault process will eventually be excluded from the set of correct processes in the URB algorithm

Eventually the condition $correct \subseteq ack[m]$ will hold, hence p_i eventually performs $URBDeliver(m)$

URB Algorithm - Correctness

URB Property 1 (Validity)

If a correct process p_i performs $URBBroadcast(m)$ then eventually p_i performs $BEBBroadcast(m)$

From BEB1, every correct process p_j eventually performs $BEBDeliver(m)$

By our lemma, p_j eventually performs $URBDeliver(m)$

URB Property 2 (No duplication)

Follows from BEB2 and BEB3

URB Algorithm - Correctness

URB Property 3 (No creation)

Follows from BEB2 and BEB3

RB Property 4 (Uniform Agreement)

Assume some process p_i performs $URBDeliver(m)$

By the URB algorithm and the (completeness and accuracy) properties of the failure detector, every correct process has already or will eventually perform $BEBDeliver(m)$

By our lemma, every correct process will $URBDeliver(m)$

Broadcasting Abstractions

Best Efforts Broadcast

Guarantees reliable delivery depending on correctness of sender process

Reliable Broadcast

Guarantees independent of sender process

Uniform Reliable Broadcast

Ensures delivery also for faulty processes (if possible)

Broadcasting Abstractions

FIFO Reliable Broadcast

Reliable broadcast with FIFO delivery order

Causal Reliable Broadcast

Reliable broadcast with "global FIFO" (causal) delivery order

FIFO Broadcast

Best Efforts Broadcast (BEB)

Events

Request: $< BEBBroadcast, m >$

Indication: $< BEBDeliver, src, m >$

Properties

BEB Property 1 (Validity) - If p_i and p_j are correct, then every message broadcast by p_i is eventually delivered by p_j

BEB Property 2 (No duplication) - No message is delivered more than once

BEB Property 3 (No creation) - No message is delivered unless it was broadcast

Reliable Broadcast (RB)

Events

Request: $< RBroadcast, m >$

Indication: $< RDeliver, src, m >$

Properties

RB Property 1 (Validity) - BEB Property 1

RB Property 2 (No duplication) - BEB Property 2

RB Property 3 (No creation) - BEB Property 3

RB Property 4 (Agreement) - For any message m, if a correct process delivers m, every correct process delivers m

Uniform Reliable Broadcast (URB)

Events

Request: $< URBroadcast, m >$

Indication: $< URDeliver, src, m >$

Properties

RB Property 1 (Validity) - BEB Property 1

RB Property 2 (No duplication) - BEB Property 2

RB Property 3 (No creation) - BEB Property 3

URB Property 4 (Uniform Agreement) - For any message m, if a process delivers m, then every correct process delivers m

Ordered Reliable Broadcasts

Defines a “per message” guarantee of all-or-nothing

Processes can see different broadcasts in different orders

Two approaches:

- First-In First-out (FIFO) broadcast
- Causal broadcast (we think of this as global FIFO)

FIFO Broadcast

Idea is to combine reliable broadcast with sequence numbers for ordering

Let m_1 and m_2 be any two messages

If some process p_i broadcasts m_1 before m_2 then any other process p_j should deliver m_1 before m_2

FIFO Ordering: If some process p_i broadcasts m_1 before m_2 then, for any other process p_j, it is the case that p_j does not deliver m_2 unless it previously delivered m_1

We can apply this to reliable broadcast and uniform reliable broadcast

Reliable FIFO Broadcast (RFB)

Events

Request: $< RFBroadcast, m >$

Indication: $< RFDeliver, src, m >$

Properties

RFB Property 1 (Validity) - RB Property 1 (Validity) - BEB Property 1

RFB Property 1 (Validity) - RB Property 2 (No duplication) - BEB Property 2

RFB Property 3 (No creation) - RB Property 3 (No creation) - BEB Property 3

RFB Property 4 (Agreement) - RB Property 4 (Agreement)

RFB Property 5 (FIFO Order) - If some process p_i broadcasts m_1 before m_2 then, for any other process p_j, it is the case that p_j does not deliver m_2 unless it previously delivered m_1

Uniform FIFO Broadcast (UFB)

Events

Request: $< UFBroadcast, m >$

Indication: $< UFDeliver, src, m >$

Properties

UFB Property 1 (Validity) - URB Property 1 (Validity) - BEB Property 1

UFB Property 1 (Validity) - URB Property 2 (No duplication) - BEB Property 2

UFB Property 3 (No creation) - URB Property 3 (No creation) - BEB Property 3

UFB Property 4 (Uniform Agreement) - URB Property 3 (Uniform Agreement)

UFB Property 5 (FIFO Order) - If some process p_i broadcasts m_1 before m_2 then, for any other process p_j, it is the case that p_j does not deliver m_2 unless it previously delivered m_1

RFB Algorithm

$Implements : ReliableFifoBroadcast\ (RFB)$
$Uses : ReliableBroadcast\ (RB)$

$upon\ event\ < Init > do$
$\quad buffer = \{\}$
$\quad next = [1...n] = (1,...,1)$
$\quad seq = 1$

$upon\ event\ < RFBBroadcast, m > do$
$\quad trigger < RBroadcast, (seq, m) >$
$\quad seq = seq + 1$

RFB Algorithm

$$
\begin{aligned}
&\textit{upon event } < RDeliver, src, (s_n, m) > \textit{ do} \\
&\quad buffer = buffer \cup (src, s_n, m) \\
&\quad while(\exists (src', s'_n, m') \in buffer : src' = src \wedge s_n = next[src])\ do \\
&\quad\quad trigger < RFDeliver, src, m' > \\
&\quad\quad next[src] = next[src] + 1 \\
&\quad\quad buffer = buffer \setminus (src', s'_n, m')
\end{aligned}
$$

Causal Broadcast

Broadcasting Abstractions

Best Efforts Broadcast

Guarantees reliable delivery depending on correctness of sender process

Reliable Broadcast

Guarantees independent of sender process

Uniform Reliable Broadcast

Ensures delivery also for faulty processes (if possible)

Broadcasting Abstractions

FIFO Reliable Broadcast

Reliable broadcast with FIFO delivery order

Causal Reliable Broadcast

Reliable broadcast with "global FIFO" (causal) delivery order

Global FIFO Implies Causal?

Think about the news feeds or tickers you have seen

Every new event that is displayed contains a reference to the event that caused it

A common on some information contains some reference to the source information

There is a global causal relationships that accounts for local actions

Even URB does not guarantee such a dependency of delivery, hence it must be dealt with by individual applications when URB is used

Causal broadcast alleviates the need for the application to deal with such dependency

Capturing Causal Order

Let m_1 and m_2 be any two messages

We use $m_1 \rightarrow m_2$ to denote that m_1 causally precedes m_2 if and only if we have:

Causal Property 1 (FIFO Order) - Some process p_i broadcasts m_1 before broadcasting m_2

Causal Property 2 (Local Order) - Some process p_i delivers m_1 and then broadcasts m_2

Causal Property 3 (Transitivity) - There is a message m_3 such that $m_1 \rightarrow m_3$ and $m_3 \rightarrow m_2$

Where an algorithm satisfies these in general, we say it meets the causal order property

Causal Order Broadcast (COB)

Events

Request: $< COBroadcast, m >$

Indication: $< CODeliver, src, m >$

Properties

COB Property 1 (Validity) - RB Property 1 (Validity) - BEB Property 1

COB Property 1 (Validity) - RB Property 2 (No duplication) - BEB Property 2

COB Property 3 (No creation) - RB Property 3 (No creation) - BEB Property 3

COB Property 4 (Agreement) - RB Property 4 (Agreement)

COB Property 5 (Causal Order) - Meets the causal order property

Two Approaches To Causal Order Broadcast

1. A non-blocking algorithm that uses some knowledge of the past

2. A blocking algorithm that uses vector clocks

COB Algorithm 1 - Using The Past

$$
\begin{aligned}
&Implements: COBroadcast\ (COB)\\
&Uses: ReliableBroadcast\ (RB)\\
&\\
&upon\ event\ \ <Init>\ do\\
&\qquad delivered = \{\}\\
&\qquad past = \{\}\\
&\\
&upon\ event\ \ <COBroadcast, m>\ do\\
&\qquad trigger < RBroadcast, [Data, past, m]) >\\
&\qquad past = past \cup \{[self, m]\}
\end{aligned}
$$

COB Algorithm 1 - Using The Past

$$
\begin{aligned}
&\textit{upon event } < \textit{RDeliver}, p_i, [\textit{Data}, \textit{past}_m, m] > \textit{do} \\
&\quad \textit{if } m \notin \textit{delivered then} \\
&\quad\quad \textit{for all } [s_n, m'] \in \textit{past do} \\
&\quad\quad\quad \textit{if } m' \notin \textit{delivered then} \\
&\quad\quad\quad\quad \textit{trigger} < \textit{CODeliver}, s_n, m' > \\
&\quad\quad\quad\quad \textit{delivered} = \textit{delivered} \cup \{m'\} \\
&\quad\quad\quad\quad \textit{past} = \textit{past} \cup \{[p_i, m']\} \\
&\quad\quad \textit{trigger} < \textit{CODeliver}, p_i, m > \\
&\quad\quad \textit{delivered} = \textit{delivered} \cup \{m\} \\
&\quad\quad \textit{past} = \textit{past} \cup \{[p_i, m]\}
\end{aligned}
$$

COB Algorithm 2 - Using Vector Clocks

Instead of sending our complete knowledge of the past, we could send a smaller representation of the information that is actually necessary

Every process maintains a vector clock VC

Only broadcast events are counted in VC

Vector clock expresses the causal past of every process

Send the vector timestamp of the past with every broadcast

Compare the timestamp of incoming message with current vector time to decide of delivery ordering

COB Algorithm 2 - Using Vector Clocks

Implementation that guarantees logical clocks will satisfy the correctness conditions:

Implementation Correctness 1 (IC1) - Clock C_K must be incremented between many two successive events on process $P_K : C_K[K] = C_K[K] + d$

Implementation Correctness 2 (IC2) - If event A is the sending of message M on P_K with vector timestamp $C_K(A)$ and event B is receive message M on P_L with vector timestamp $C_L(B)$ we can calculate $C_L(B) = max(C_L[p], C_K[p] + d)$ for all p

We know that satisfying these means we can guarantee $\forall A, B : C_K(A) < C_L(B) \leftrightarrow A \rightarrow B$

COB Algorithm 2 - Using Vector Clocks

$$
\begin{aligned}
&\textit{Implements} : \textit{COBroadcast}\ (\textit{COB})\\
&\textit{Uses} : \textit{ReliableBroadcast}\ (\textit{RB})\\
&\\
&\textit{upon event}\ < \textit{Init} > \textit{do}\\
&\qquad \textit{pending} = \{\}\\
&\qquad \textit{for all}\ p_i \in S\\
&\qquad\qquad V[p_i] = 0\\
&\\
&\textit{upon event}\ < \textit{COBroadcast}, m > \textit{do}\\
&\qquad \textit{trigger} < \textit{CODeliver}, \textit{self}, m >\\
&\qquad \textit{trigger} < \textit{RBroadcast}, [\textit{Data}, \textit{VC}, m]) >\\
&\qquad \textit{VC}[\textit{self}] = \textit{VC}[\textit{self}] + 1
\end{aligned}
$$

COB Algorithm 2 - Using Vector Clocks

$$
\begin{aligned}
&\textit{upon event } < \textit{RDeliver}, p_j, [\textit{Data}, VC_m, m] > \textit{do} \\
&\quad \textit{if } p_j \neq \textit{self then} \\
&\quad\quad pending = pending \cup \{(p_j, [\textit{Date}, VC_m, m])\} \\
&\quad\quad \textit{trigger} < \textit{DeliverPending} > \\
&\\
&\textit{upon } < \textit{DeliverPending} > \textit{do} \\
&\quad \textit{if } (s_x, [\textit{Data}, VC_x, x] \in pending \textit{ where } \forall p_j : VC[p_j] \geq VC_x[p_j]) \textit{ do} \\
&\quad\quad pending = pending \setminus (s_x, [\textit{Data}, VC_x, x]) \\
&\quad\quad \textit{trigger} < \textit{CODeliver}, s_x, x > \\
&\quad\quad VC[s_x] = VC[s_x] + 1
\end{aligned}
$$

Broadcasting Abstractions

Best Efforts Broadcast

Guarantees reliable delivery depending on correctness of sender process

Reliable Broadcast

Guarantees independent of sender process

Uniform Reliable Broadcast

Ensures delivery also for faulty processes (if possible)

Broadcasting Abstractions

FIFO Reliable Broadcast

Reliable broadcast with FIFO delivery order

Causal Reliable Broadcast

Reliable broadcast with "global FIFO" (causal) delivery order