TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Programming Distributed Systems

02 Broadcast

Annette Bieniusa

AG Softech
FB Informatik
TU Kaiserslautern

Summer Term 2019

Overview

- Formalism for specifying distributed algorithms
- Composability of distributed algorithms
- The Broadcast Problem
  - Best-effort broadcast
  - Reliable broadcast
  - Causal broadcast

Motivation

The Broadcast Problem

Informally: A process needs to transmit the same message m to N other processes.

Assumptions

- Complete set of processes in the system is known a-priori
- Perfect-Point-2-Point Link Abstraction
- Asynchronous system (no rounds, no failure detection)

What is the simplest solution that you can think of?

What is the simplest solution that you can think of?

Just go ahead and send the message to everyone, one at a time.

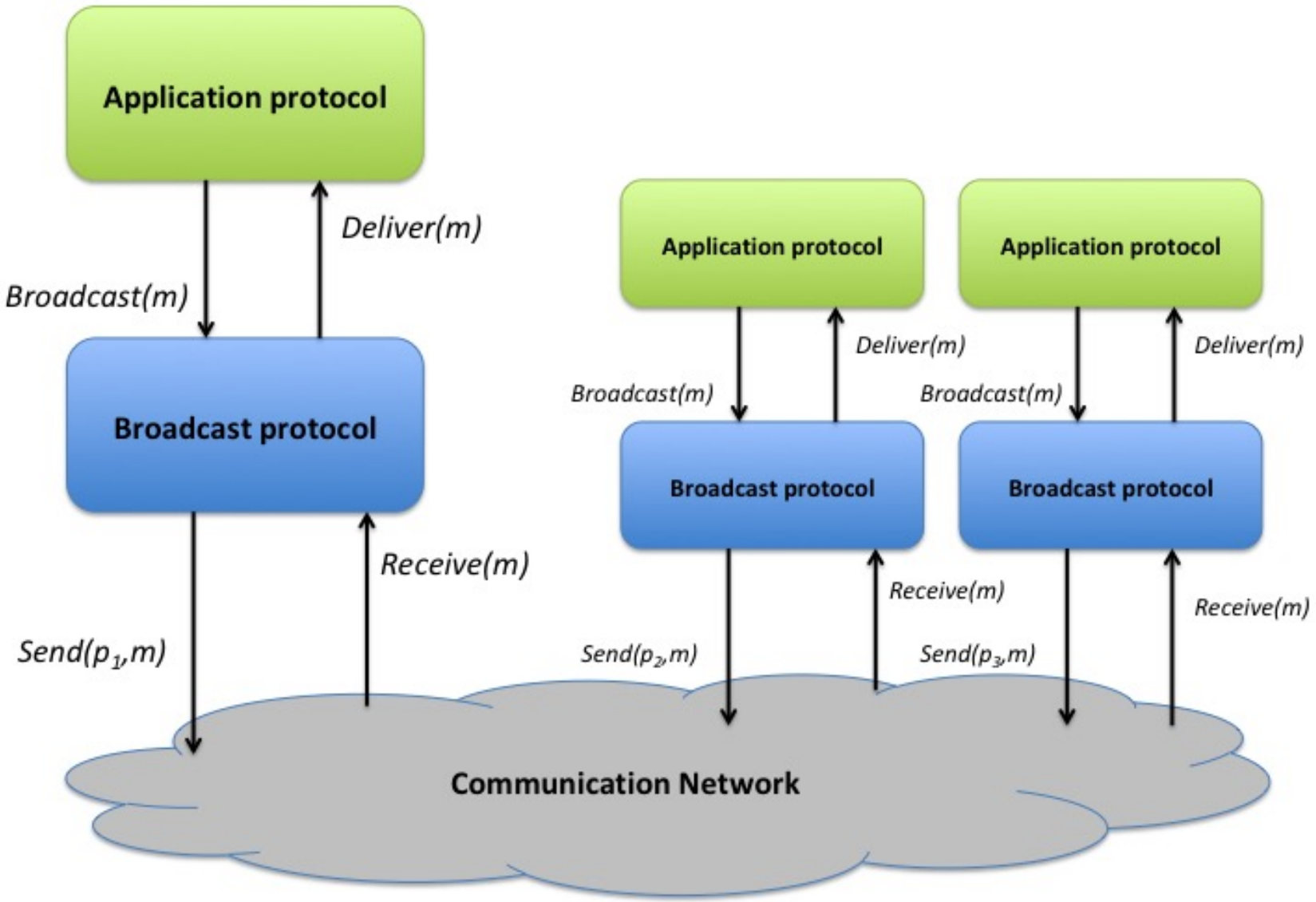
Specifying the broadcast problem

Wait... How do you specify an algorithm for a process again?

Specifying the broadcast problem

Wait... How do you specify an algorithm for a process again?

$\Rightarrow$ Deterministic I/O automaton!


Application protocol
Deliver(m)
Broadcast(m)
Broadcast protocol
Receive(m)
Send(p1,m)
Application protocol
Application protocol
Deliver(m)
Deliver(m)
Broadcast(m)
Broadcast(m)
Broadcast protocol
Broadcast protocol
Receive(m)
Receive(m)
Send(p2,m)
Send(p3,m)
Communication Network

The Anatomy of an Algorithm

- Event driven interface

```
Upon Init           do: ...
Upon Broadcast(m)   do: ...
Upon Receive(p, m)  do: ...
```

- You can trigger an event on another layer:

```
trigger Send(q, m)
trigger Deliver(p, m)
```

- There is a special event called `Init` for initializing the local state.
- `q` denotes the target process when sending a message
- `p` denotes the process where the message originated from

Best-effort Broadcast (BEB): Specification

- *BEB1 (Best-Effort Validity):* For any two correct processes i and j, every message broadcast by i is eventually delivered by j.
- *BEB2 (No Duplication):* No message is delivered more than once.
- *BEB3 (No Creation):* If a correct process j delivers a message m, then m was broadcast to j by some process i.

Best-effort Broadcast: Algorithm

Idea:

- Just go ahead and send the message to every other process.
- When you get one of these messages, you deliver it to the upper layer.

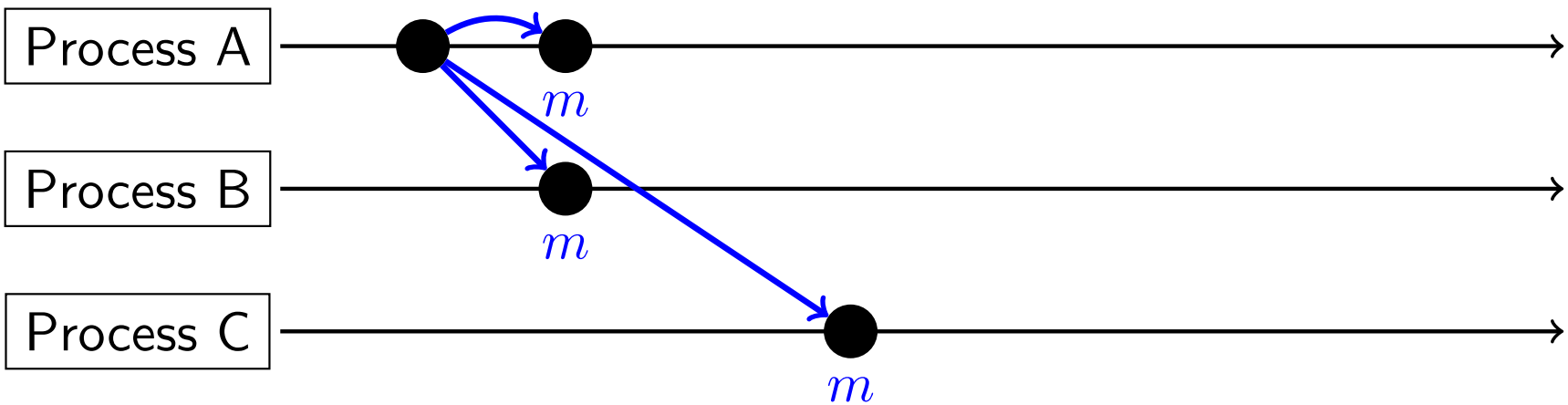
```
State:          -- // could be omitted
Upon Init do:   -- // could be omitted

Upon beb-Broadcast(m) do:
  forall q ∈ Π:
    trigger Send(q, m);

Upon Receive(p, m) do:
  trigger beb-Deliver(p, m);
```

Best-effort Broadcast: Correctness

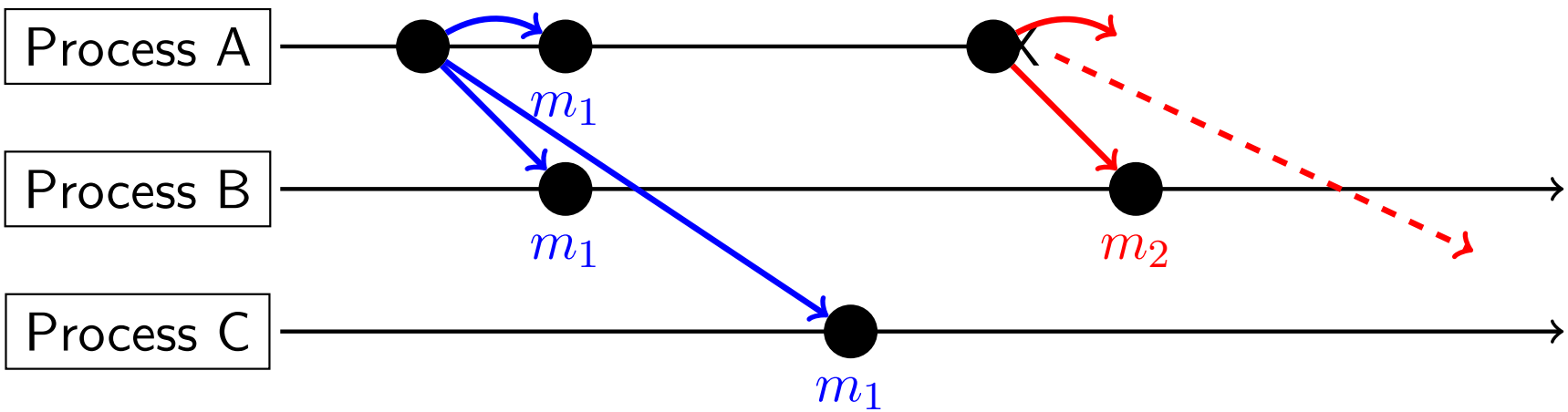
Why does it work?

- BEB1 holds because Perfect-Point-2-Point links guarantee reliable delivery (PL1)
- BEB2 holds due to PL2, BEB3 holds due to PL3

Best-effort Broadcast: Scenario 1



Best-effort Broadcast: Scenario 2



Limitations of Best-effort Broadcast

What happens if a process fails while sending a message?

- If the sender crashes before being able to send the message to all processes, some process will not deliver the message.
- Even in the absence of communication failures!

Limitations of Best-effort Broadcast

What happens if a process fails while sending a message?

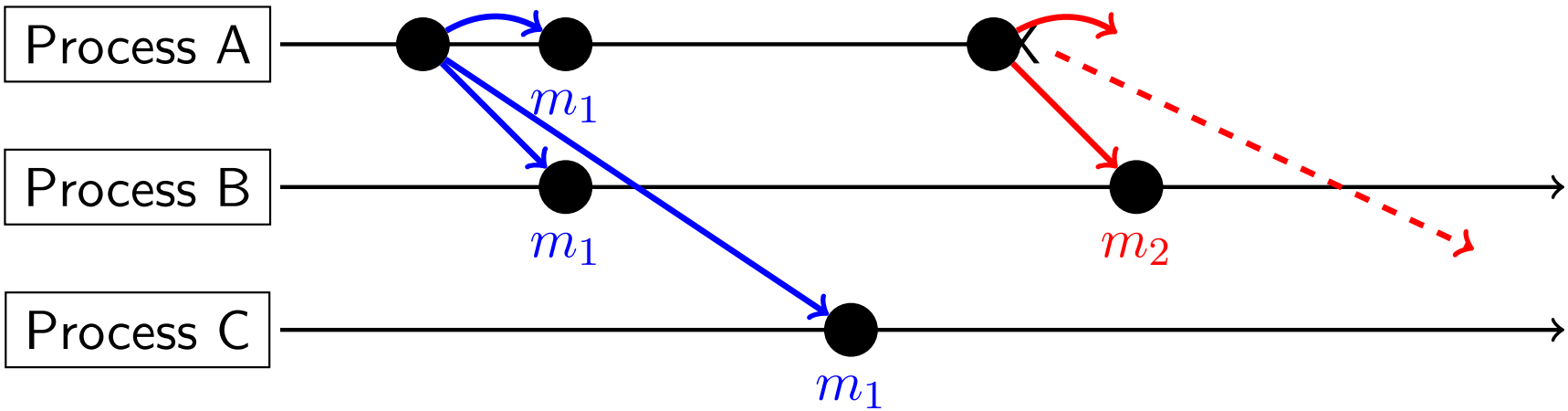
- If the sender crashes before being able to send the message to all processes, some process will not deliver the message.
- Even in the absence of communication failures!

Let's try for a stronger version of broadcast ...

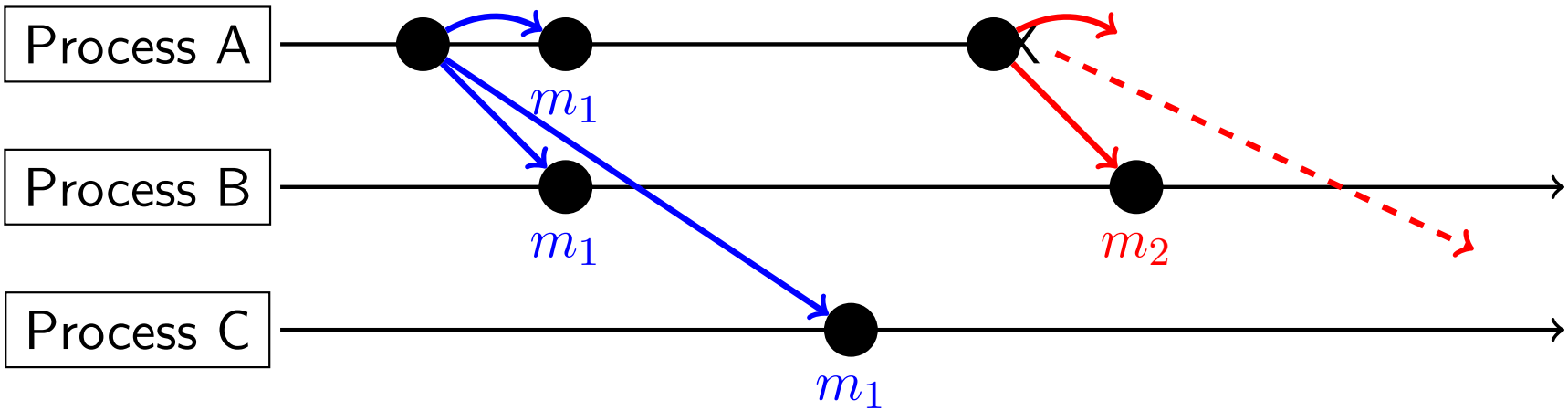
Reliable Broadcast (RB): Specification

- *RB1 (Validity):* If a correct process i broadcasts message m, then i eventually delivers the message.
- *RB2 (No Duplications):* No message is delivered more than once.
- *RB3 (No Creation):* If a correct process j delivers a message m, then m was broadcast to j by some process i.
- *RB4 (Agreement):* If a message m is delivered by some correct process i, then m is eventually delivered by every correct process j.

Reliable Broadcast (RB): Scenario 1

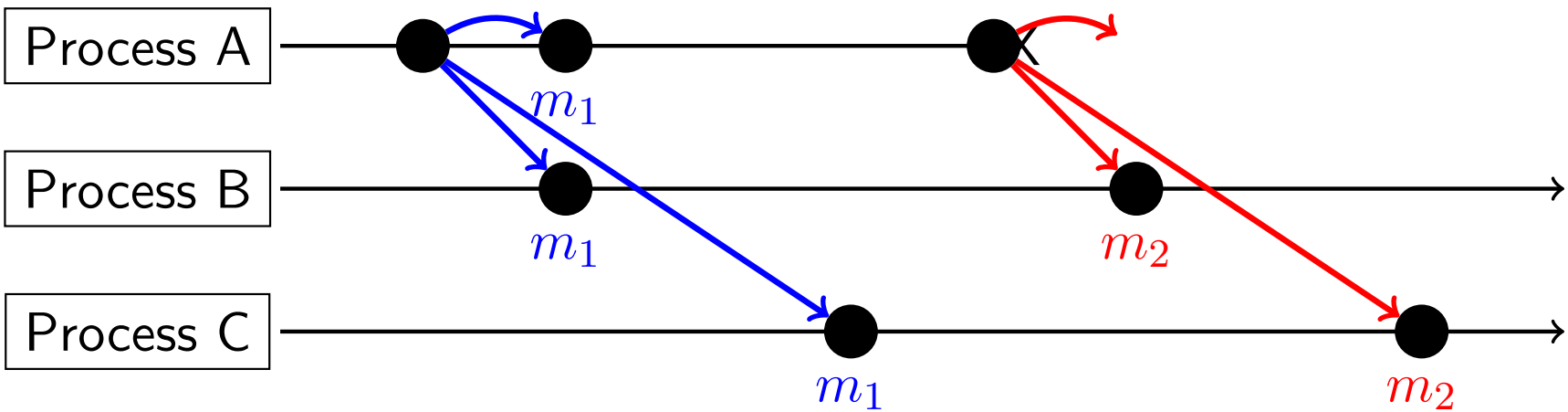


Reliable Broadcast (RB): Scenario 1

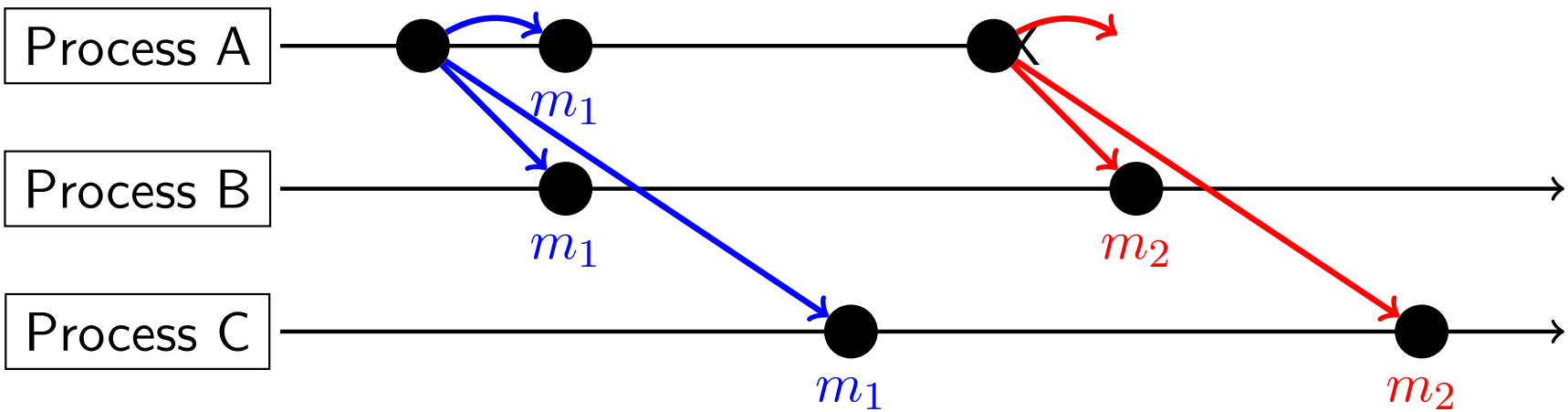


Not possible under Reliable Broadcast: RB4 is violated!

Reliable Broadcast (RB): Scenario 2

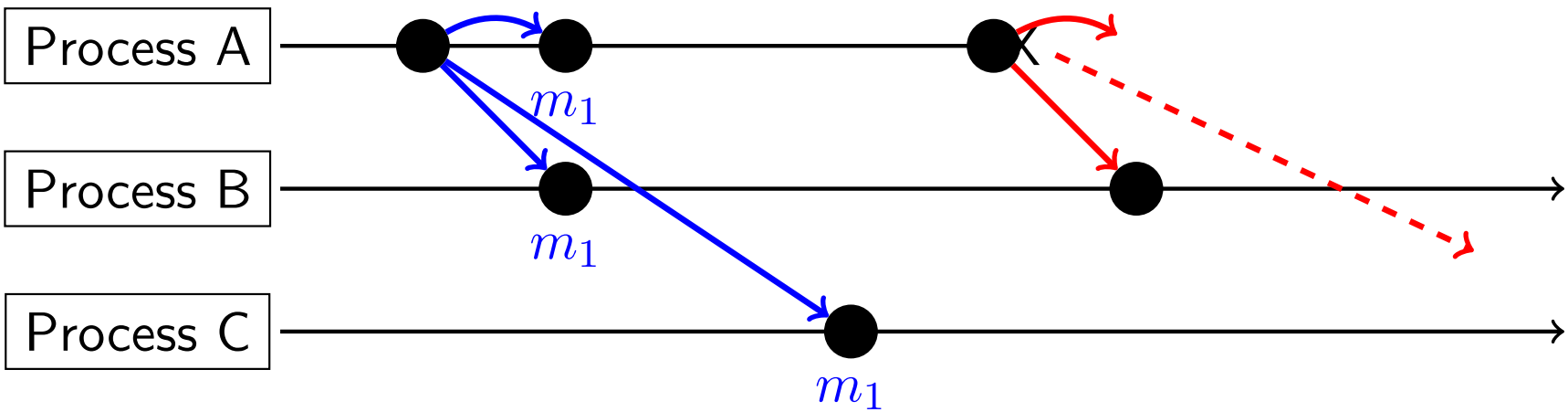


Reliable Broadcast (RB): Scenario 2

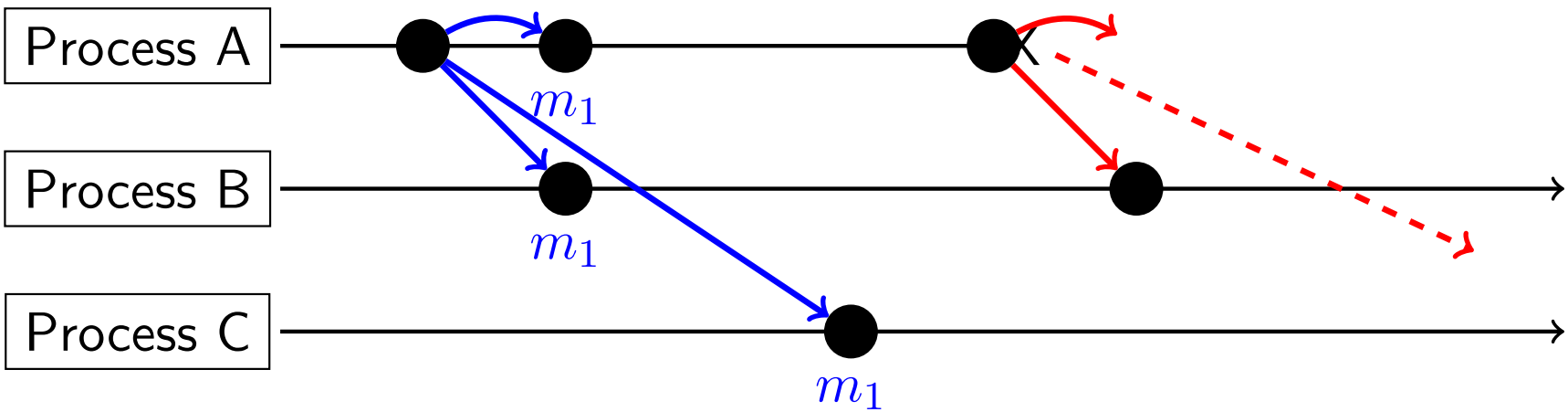


The fact that process A does not deliver m_2 is not a problem, because only correct processes are required to deliver their own messages (RB1).

Reliable Broadcast (RB): Scenario 3

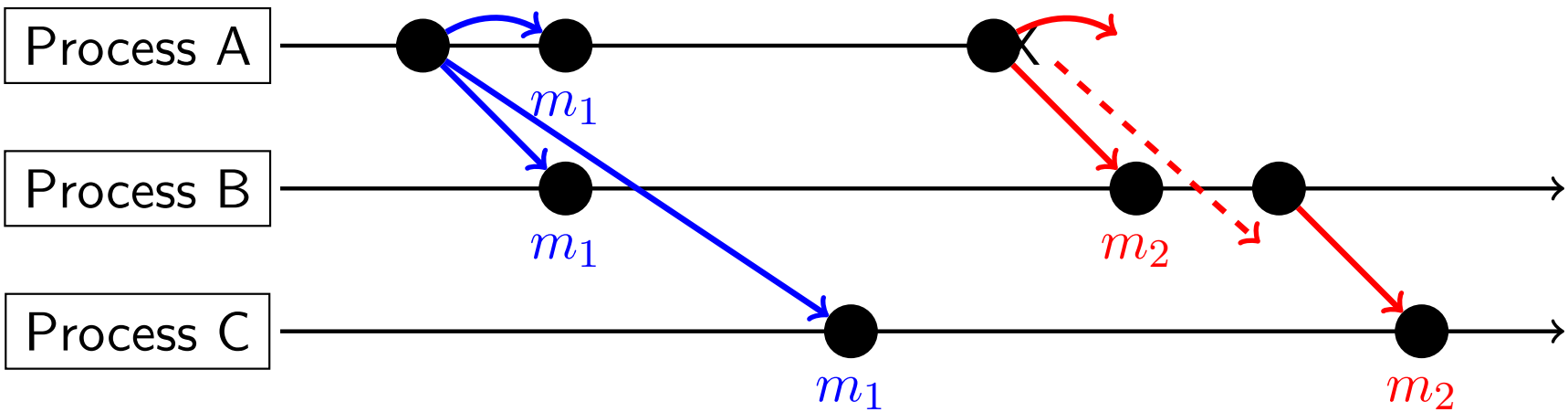


Reliable Broadcast (RB): Scenario 3



The fact that no process delivers m_2 is not a problem, because process A is faulty (RB1) and no process delivers m_2 (RB4).

Reliable Broadcast (RB): Scenario 4



Reliable Broadcast (RB): Algorithm

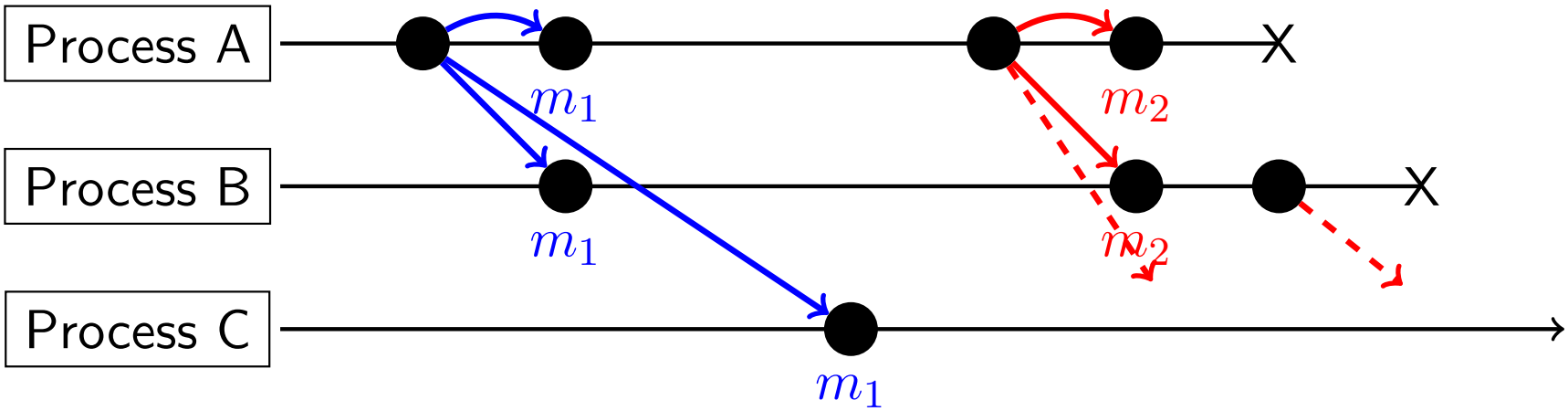
```
State:
  delivered //set of message ids that were already delivered

Upon Init do:
  delivered <- ∅;

Upon rb-Broadcast(m) do
  trigger rb-Deliver(self, m);
  m_id <- generateUniqueID(m);
  delivered <- delivered ∪ {m_id};
  trigger beb-Broadcast([m_id, m]);

Upon beb-Deliver(p, [m_id, m]) do
  if ( m_id ∉ delivered ) then
    delivered <- delivered ∪ {m_id};
    trigger rb-Deliver(p, m);
    trigger beb-Broadcast([m_id, m]);
```

Why is this algorithm correct?

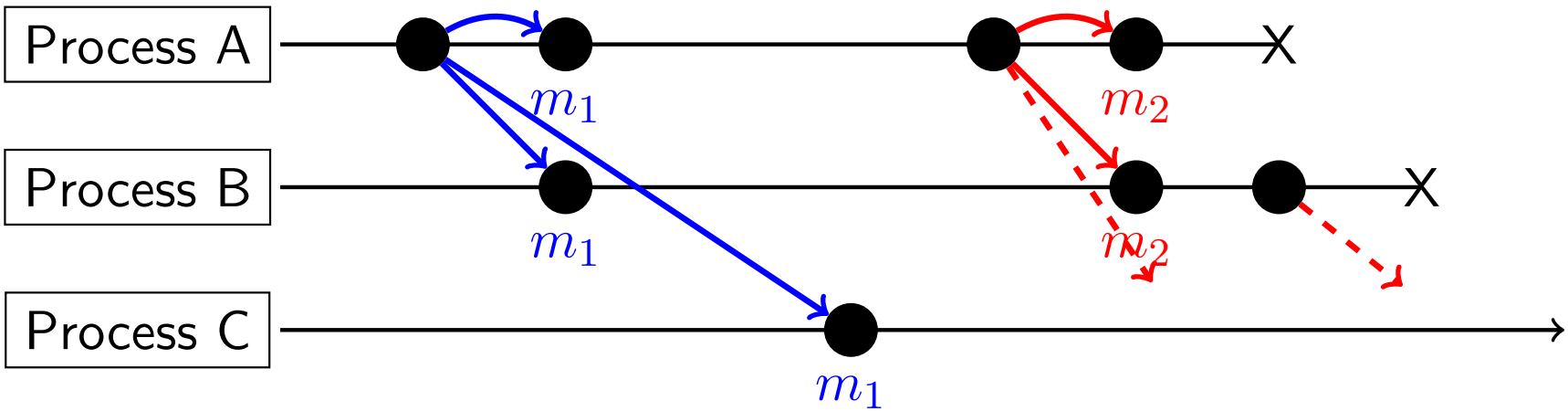
Reliable Broadcast (RB): Correctness

- *RB1 (Validity):* If a correct process i broadcasts message m, then i eventually delivers the message.
  - Delivering the message is the first step when handling rb-Broadcast.
- *RB2 (No Duplications):* No message is delivered more than once.
  - By handling the set of delivered messages.
- *RB3 (No Creation):* If a correct process j delivers a message m, then m was broadcast to j by some process i.
  - By BEB3.
- *RB4 (Agreement):* If a message m is delivered by some correct process i, then m is eventually delivered by every correct process j.
  - Before rb-Delivering m, a correct process forwards m to all processes. By BEB1 and p being correct, all correct processes will eventually receive m and rb-Deliver it.

Reliable Broadcast (RB): Scenario 5



Reliable Broadcast (RB): Scenario 5



The fact that m_2 has been delivered by faulty A and B does not imply that m_2 has to be delivered by C as well. Yet, this situation is not desirable, because two processes deliver something and another one does not.

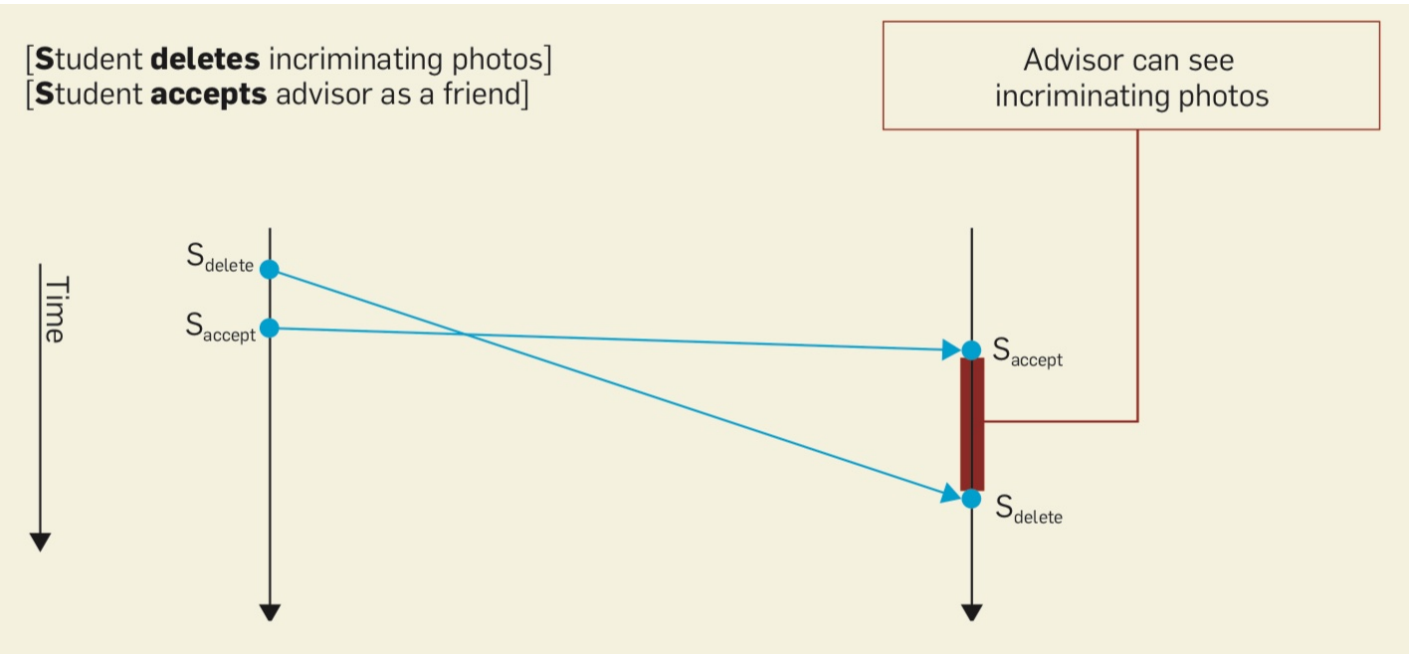
$\Rightarrow$ Interaction with external world!

Uniform Reliable Broadcast - Specification

- *URB1 (Validity):* If a correct process i broadcasts message m, then i eventually delivers the message.
- *URB2 (No Duplications):* No message is delivered more than once.
- *URB3 (No Creation):* If a correct process j delivers a message m, then m was broadcast to j by some process i.
- *URB4 (Uniform Agreement):* If a message m is delivered by some ~~correct~~ process i, then m is eventually delivered by every correct process j.

Problem: Message ordering

- Given the asynchronous nature of distributed systems, messages may be delivered in *any* order.
- Some services, such as replication, need messages to be delivered in a consistent manner, otherwise replicas may diverge.



FIFO Order

If a process p broadcasts a message m before the same process broadcasts another message m', then no correct process q delivers m' unless it has previously delivered m.

$$broadcast_p(m) \rightarrow broadcast_p(m') \Rightarrow deliver_q(m) \rightarrow deliver_q(m')$$

Causal Order

If the broadcast of a message m happens-before the broadcast of some message m', then no correct process delivers m' unless it has previously delivered m.

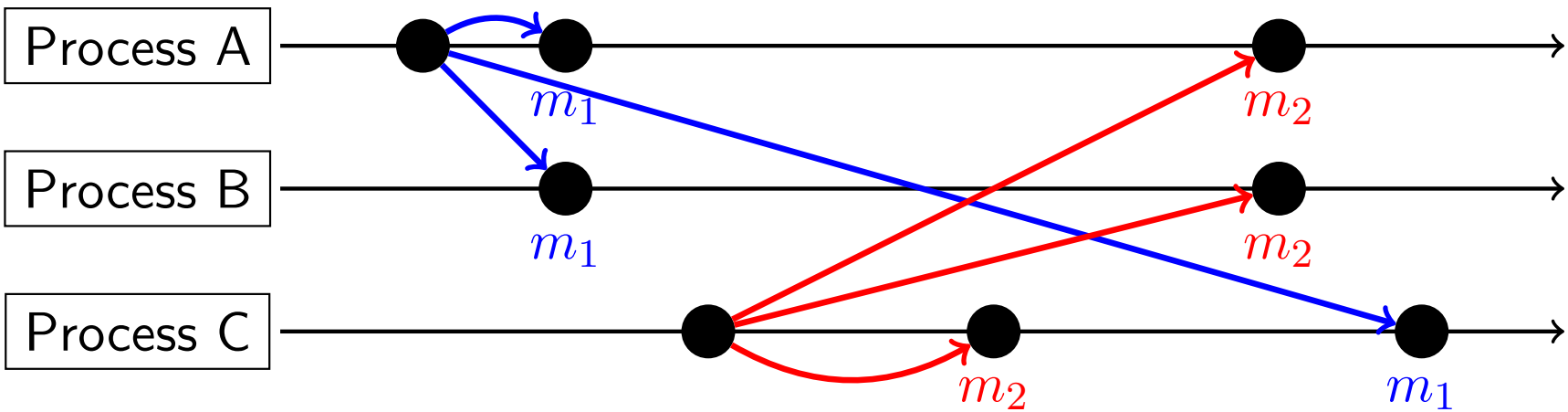
$$broadcast_p(m) \rightarrow broadcast_q(m') \Rightarrow deliver_r(m) \rightarrow deliver_r(m')$$

Total Order

If correct processes p and q both deliver messages m, m', then p delivers m before m' if and only if q delivers m before m'.

$$deliver_p(m) \rightarrow deliver_p(m') \Rightarrow deliver_q(m) \rightarrow deliver_q(m')$$

Message ordering: Quizzzzz

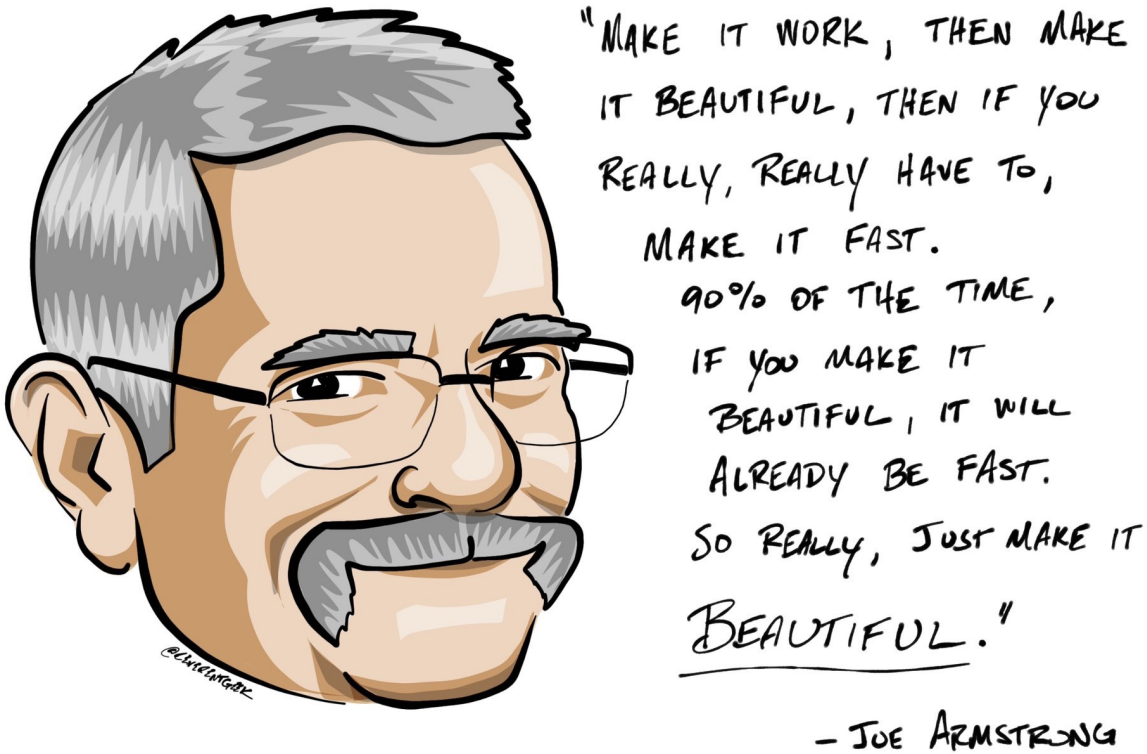


Is this allowed under FIFO Order, Causal Order, Total Order?

Summary

- Composability of distributed algorithms by stacking algorithms
- Correctness proofs based on properties of underlying level + algorithmic properties
- Different variants of solution to the Broadcast Problem
  - Best-effort broadcast
  - Reliable broadcast
  - Uniform reliable broadcast
  - Causal broadcast ($\Rightarrow$ next lecture)
  - [Uniform causal broadcast]

Joe Armstrong († 20 April 2019)



Checkout Joe's thesis[1] for lots of wisdom on building distributed systems!

Sketch by David Neal (http://reverentgeek.com/)

Further reading I

[1] Joe Armstrong. “Making reliable distributed systems in the presence of software errors”. Diss. Royal Institute of Technology, Stockholm, Sweden, 2003. URL: http://erlang.org/download/armstrong_thesis_2003.pdf.