Causal Mutual Byzantine Broadcast

Research project

Samuel Pénault, Guillaume Poignant,
Florian Monsion, Mathieu Féry

Supervisors

Vincent Kowalski, Matthieu Perrin,
Achour Mostéfaoui

Byblos Seminar — Mai 19, 2022

Computing model

- Closed system: n processes
- Message passing: processes send and receive messages
- Reliable channels: no message loss
- Asynchronous processes and channels
- Up to $t < \frac{n}{3}$ Byzantine processes

Issue

How to implement a
single-writer/multiple-reader
linearizable read/append register?

Dedicated algorithms

JPDC 2016 D. Imbs, S. Rajsbaum, M. Raynal, J. Stainer.
Read/write shared memory abstraction on top of asynchronous Byzantine message-passing systems.

TCS 2017 A. Mostéfaoui, M. Petrolia, M. Raynal, C. Jard.
Atomic Read/Write Memory in Signature-Free Byzantine Asynchronous Message-Passing Systems.

Dedicated abstraction: SCD-broadcast

TCS 2021 D. Imbs, A. Mostéfaoui, M. Perrin, M. Raynal.
Set-constrained delivery broadcast: A communication abstraction for read/write implementable distributed objects.

OPODIS 2019 A. Auvolat, M. Raynal, F. Taïani.
Byzantine-tolerant set-constrained delivery broadcast

Dedicated algorithms

JPDC 2016 D. Imbs, S. Rajsbaum, M. Raynal, J. Stainer.
Read/write shared memory abstraction on top of asynchronous Byzantine message-passing systems.

TCS 2017 A. Mostéfaoui, M. Petrolia, M. Raynal, C. Jard.
Atomic Read/Write Memory in Signature-Free Byzantine Asynchronous Message-Passing Systems.

Dedicated abstraction: SCD-broadcast

TCS 2021 D. Imbs, A. Mostéfaoui, M. Perrin, M. Raynal.
Set-constrained delivery broadcast: A communication abstraction for read/write implementable distributed objects.

OPODIS 2019 A. Auvolat, M. Raynal, F. Taïani.
Byzantine-tolerant set-constrained delivery broadcast

Complexity issues

- SCD-broadcast requires $\mathcal{O}(n^2)$ messages
  - Not only for reliability
- SCD-broadcast imposes a convoy effect

Problem statement

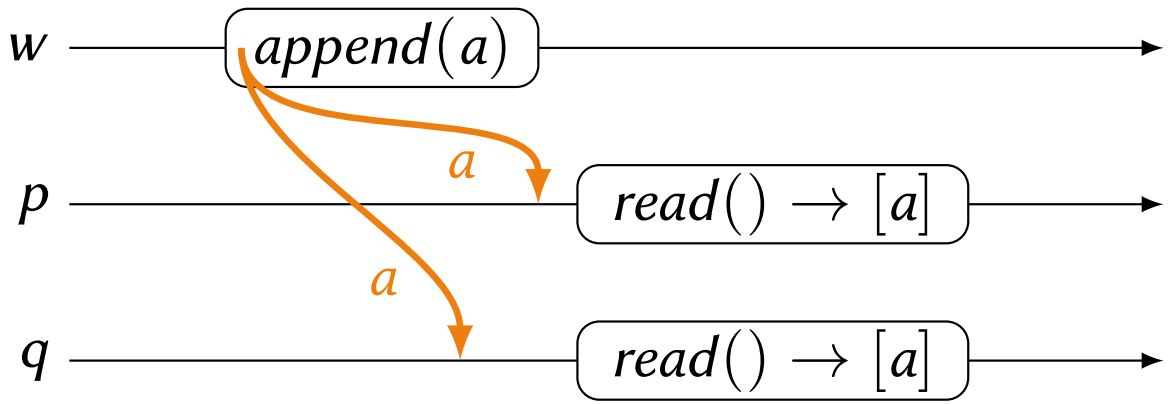
- SCD-broadcast is the abstraction for snapshot objects

  What is the abstraction for Read/Append registers?

Contributions

- CMB-broadcast: a new broadcast abstraction
- An algorithm for a read/append register
- An algorithm for CMB-broadcast

Specification – Reliable broadcast



Reliable broadcast

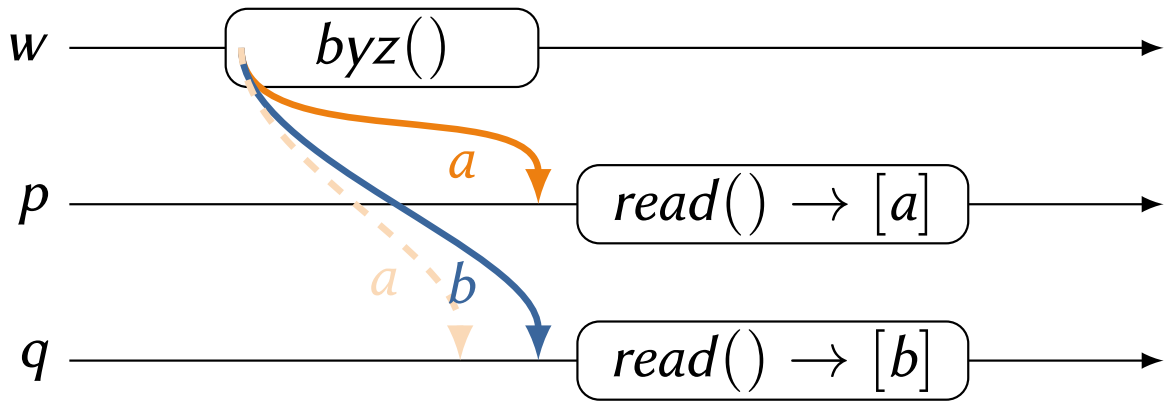
Validity: No message creation

Integrity: No message duplication

Local progress: Correct processes deliver their own messages

Reliability: Correct processes deliver the same set of messages

Specification – Reliable broadcast



Reliable broadcast

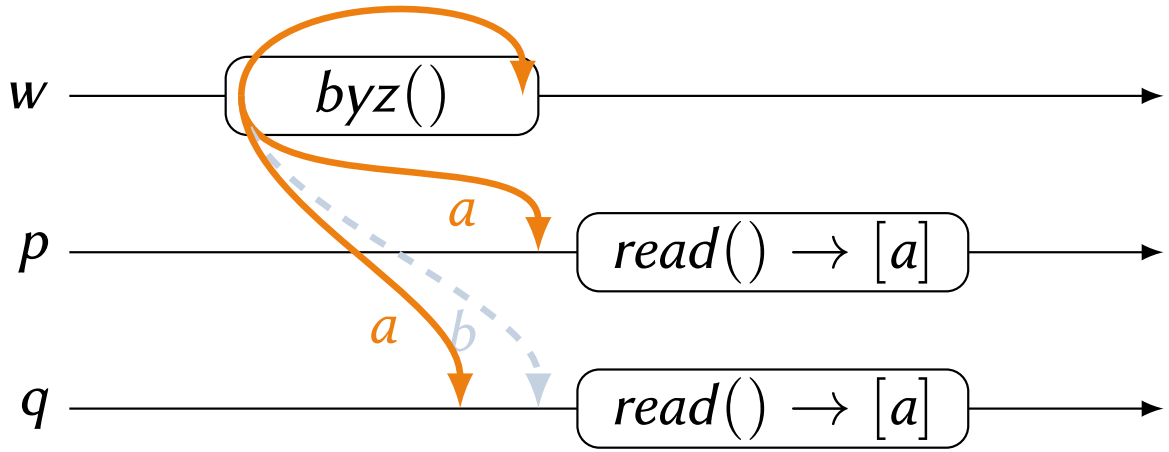
Validity: No message creation

Integrity: No message duplication

Local progress: Correct processes deliver their own messages

Reliability: Correct processes deliver the same set of messages

Specification – Reliable broadcast



Reliable broadcast

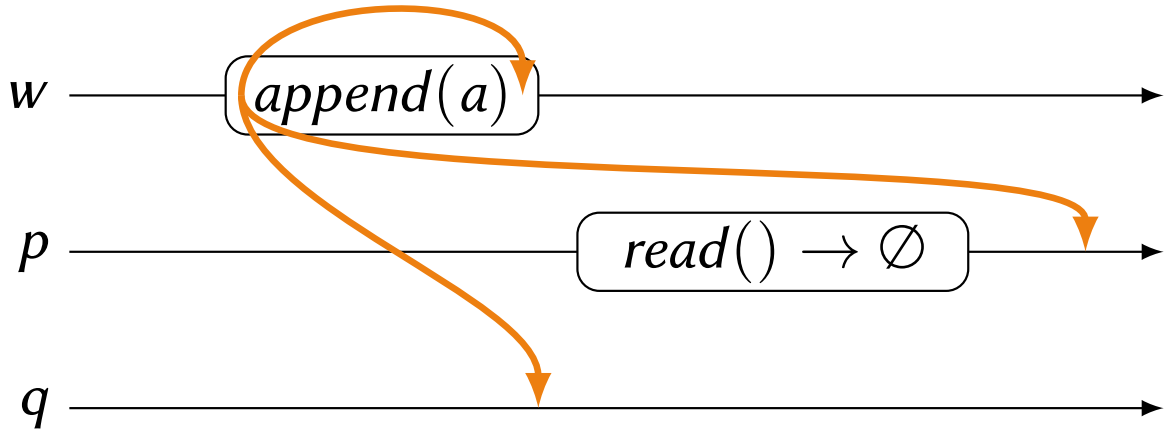
Validity: No message creation

Integrity: No message duplication

Local progress: Correct processes deliver their own messages

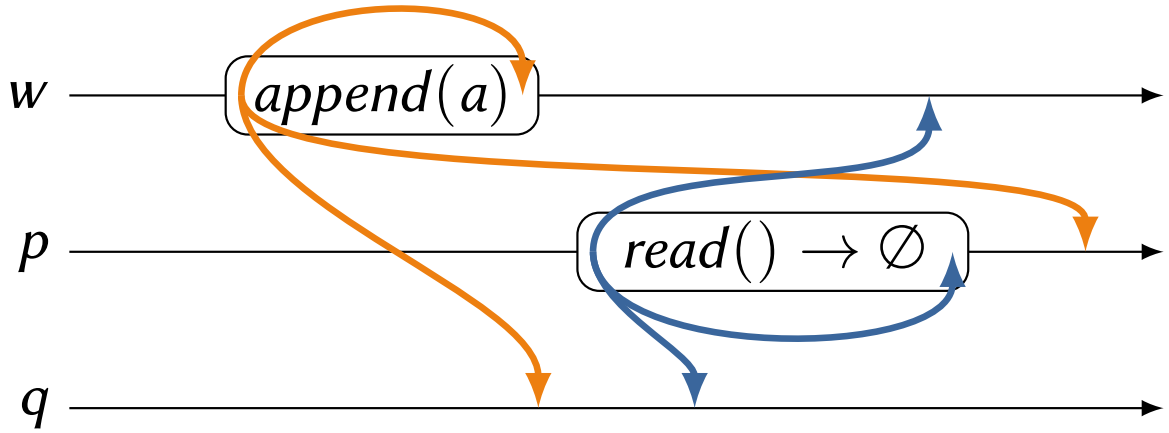
Reliability: Correct processes deliver the same set of messages

Specification – Mutual ordering



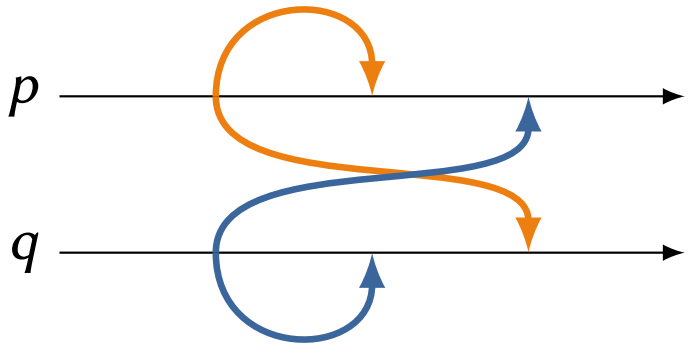
Mutual ordering

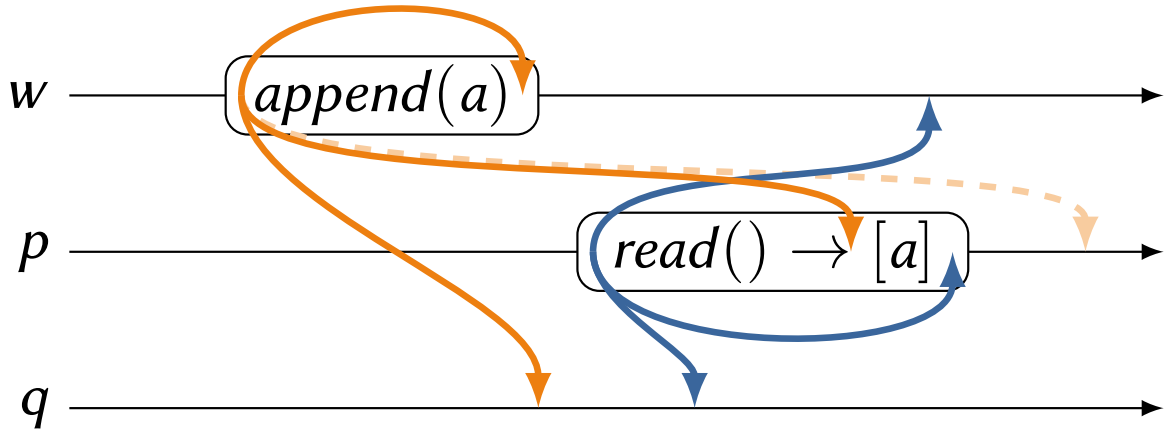
- Correct processes cannot mutually ignore each other
- Forbidden pattern:



Mutual ordering

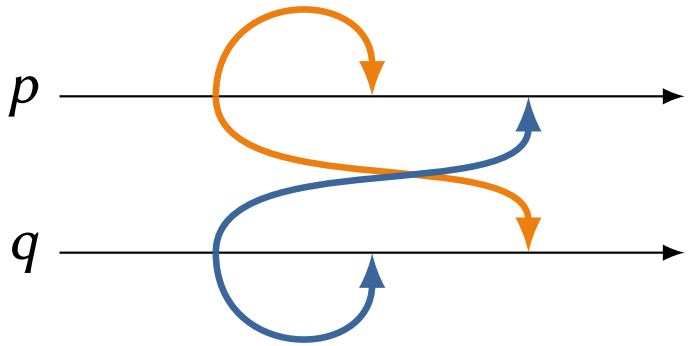
- Correct processes cannot mutually ignore each other
- Forbidden pattern:



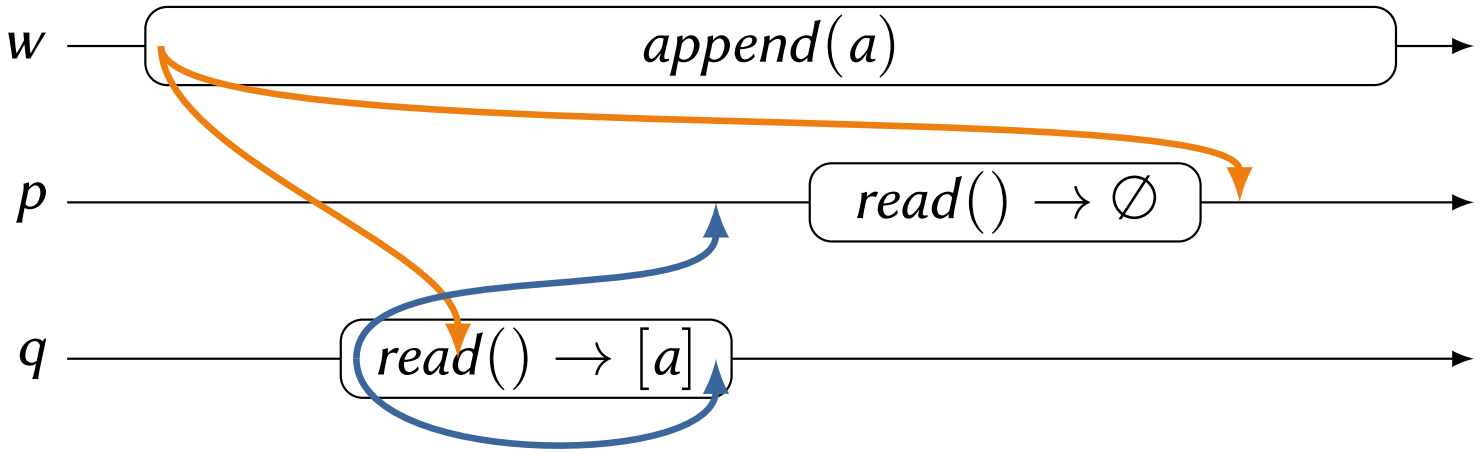


Mutual ordering

- Correct processes cannot mutually ignore each other
- Forbidden pattern:



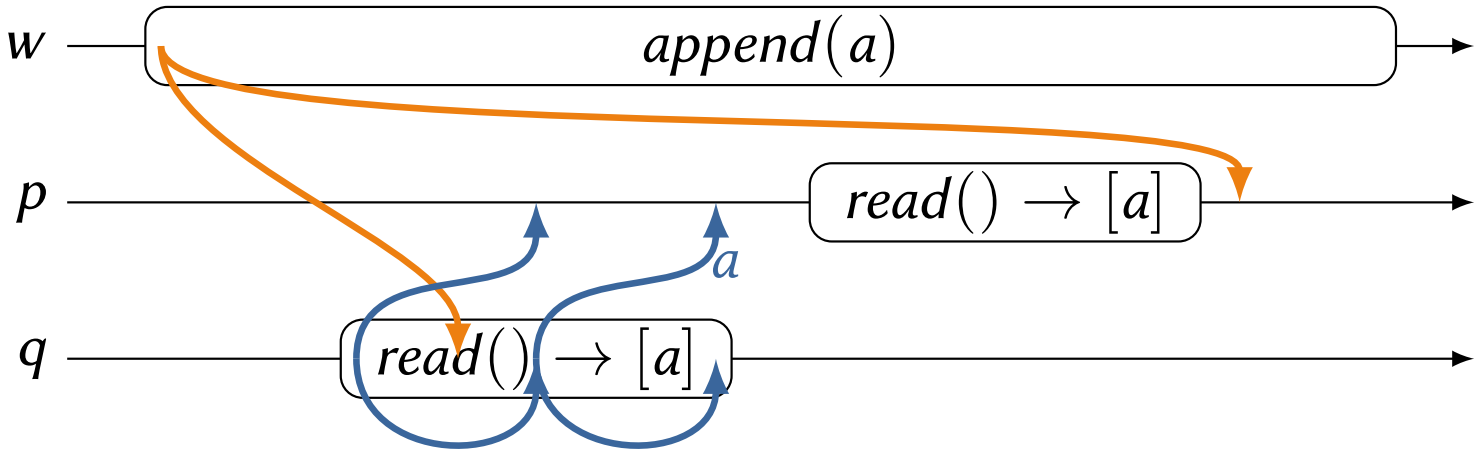
Specification – Causal ordering



Causal ordering

- Causality it transmitted by correct processes

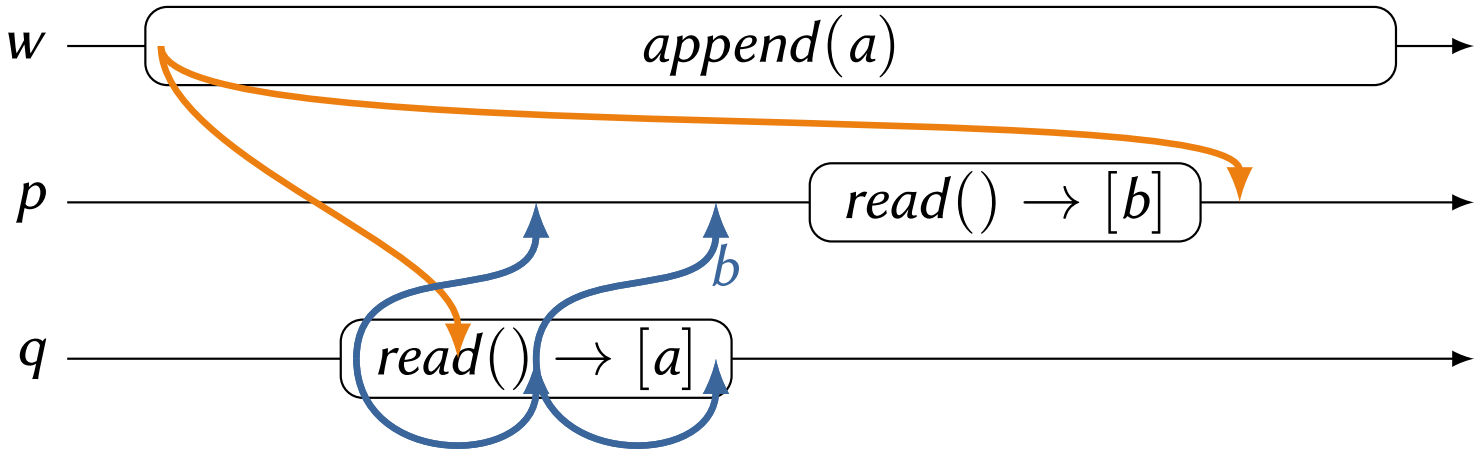
Specification – Causal ordering



Causal ordering

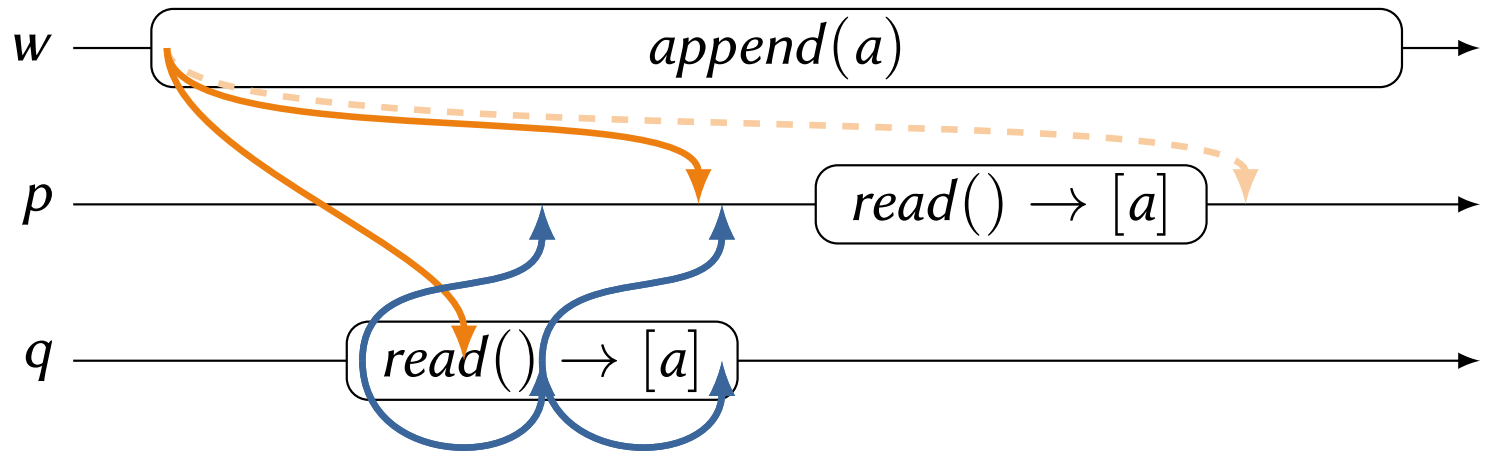
- Causality it transmitted by correct processes

Specification – Causal ordering



Causal ordering

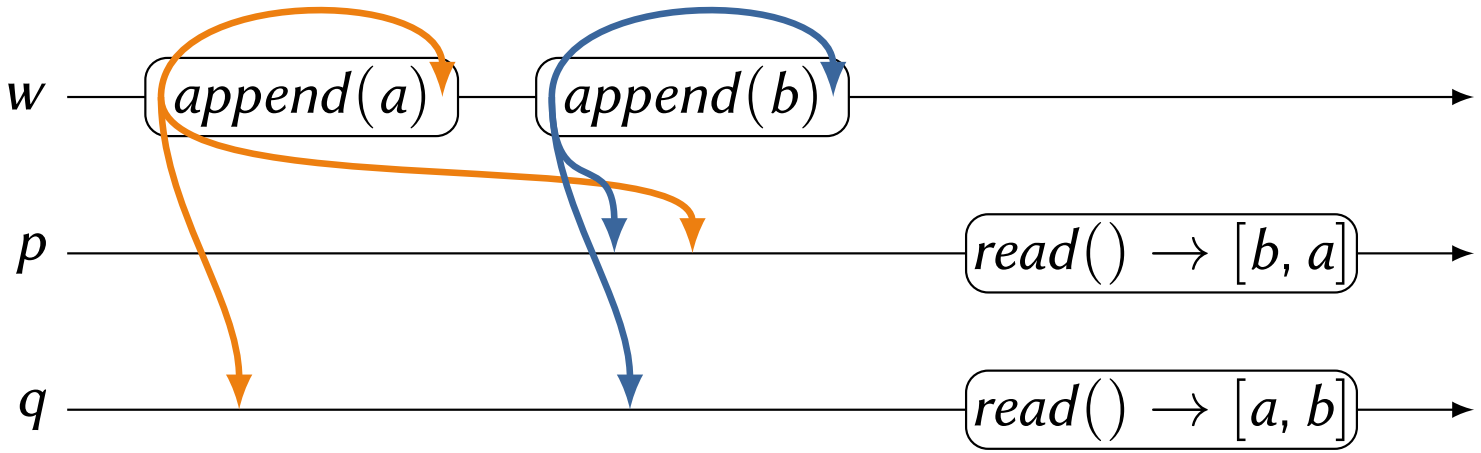
- Causality it transmitted by correct processes



Causal ordering

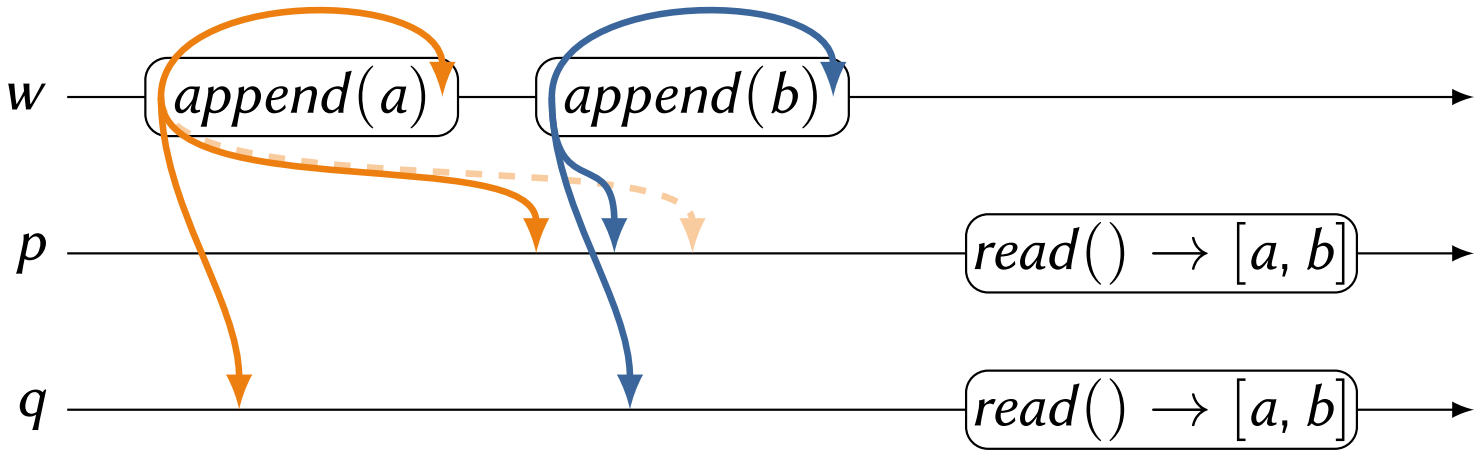
- Causality it transmitted by correct processes

Specification – FIFO ordering



FIFO ordering

- Correct processes agree on a per-process order



FIFO ordering

- Correct processes agree on a per-process order

Specification – Causal-Mutual-Byzantine-broadcast

Complete specification

Validity No message creation

Integrity No message duplication

Local progress Correct processes deliver their own messages

Reliability Correct processes deliver the same set of messages

Mutual ordering Correct processes cannot ignore each other

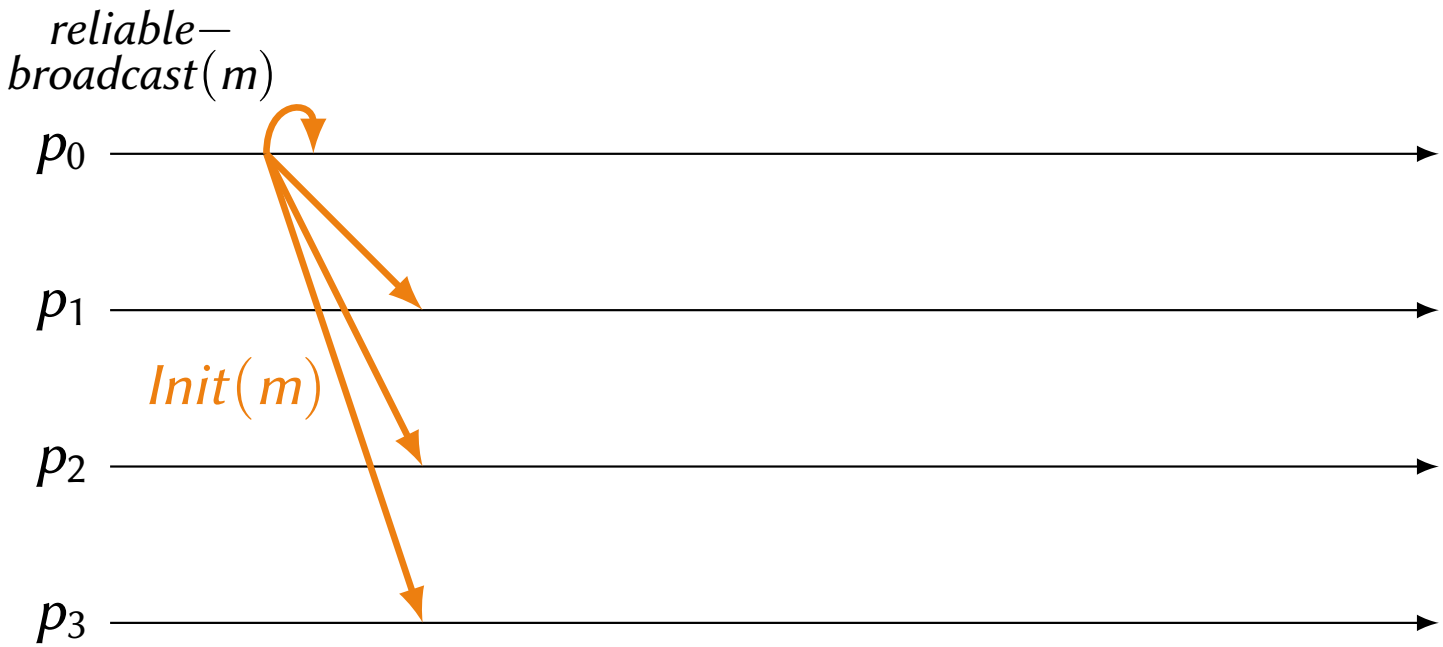
Causal ordering Causality it transmitted by correct processes

Fifo ordering Correct processes agree on a per-process order

Implementation

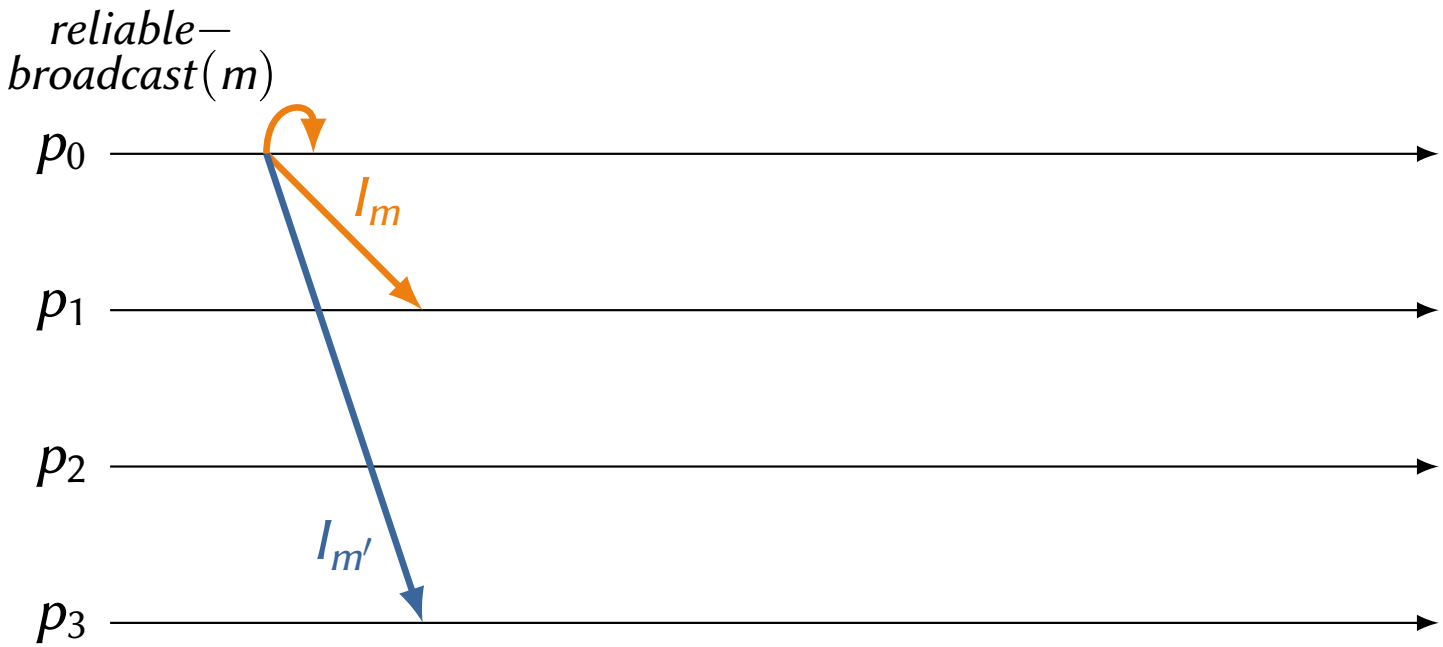
How to implement CMB-broadcast in a closed message-passing system with $t < \frac{n}{3}$ Byzantine processes?

Implementation – Reminder on Bracha's algorithm



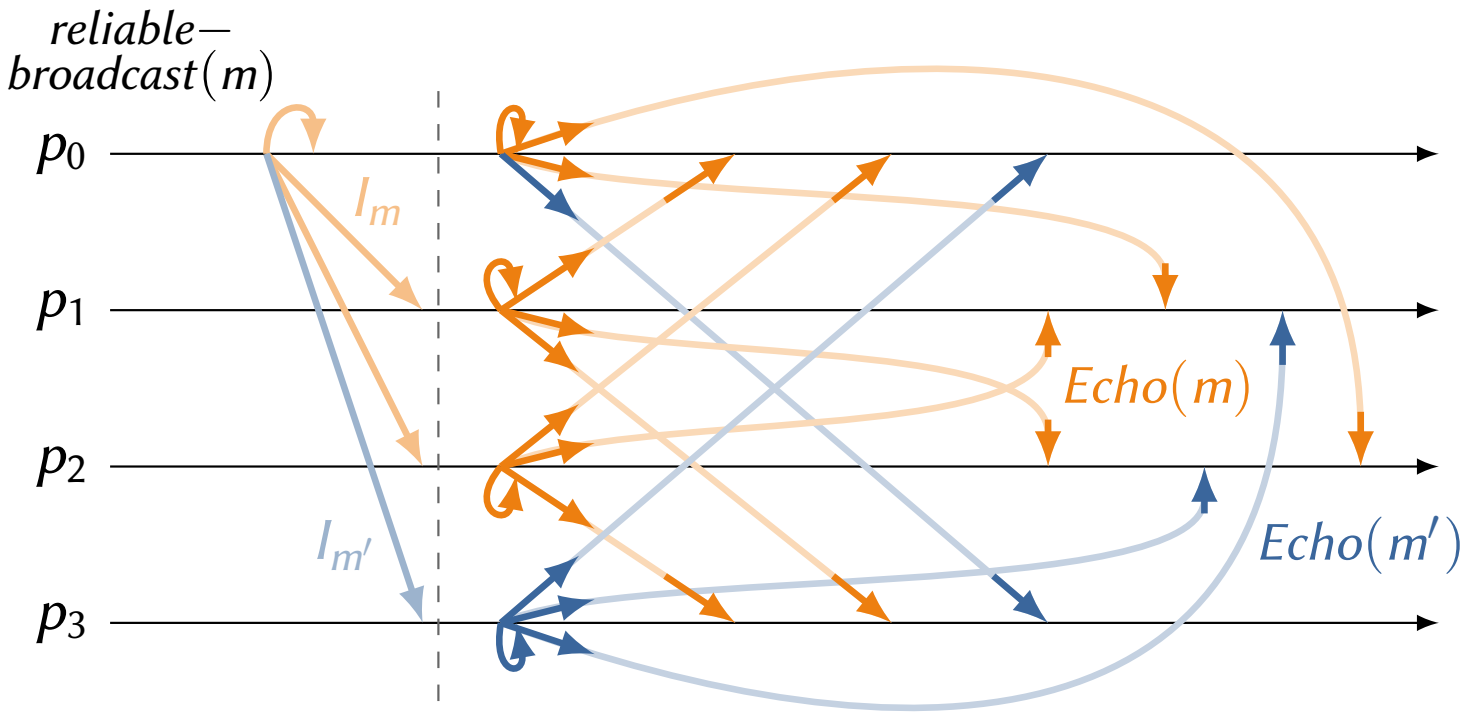
Properties

- After *Echo*: at most one message supported by correct processes
- After *Ready*: all correct processes support the same messages



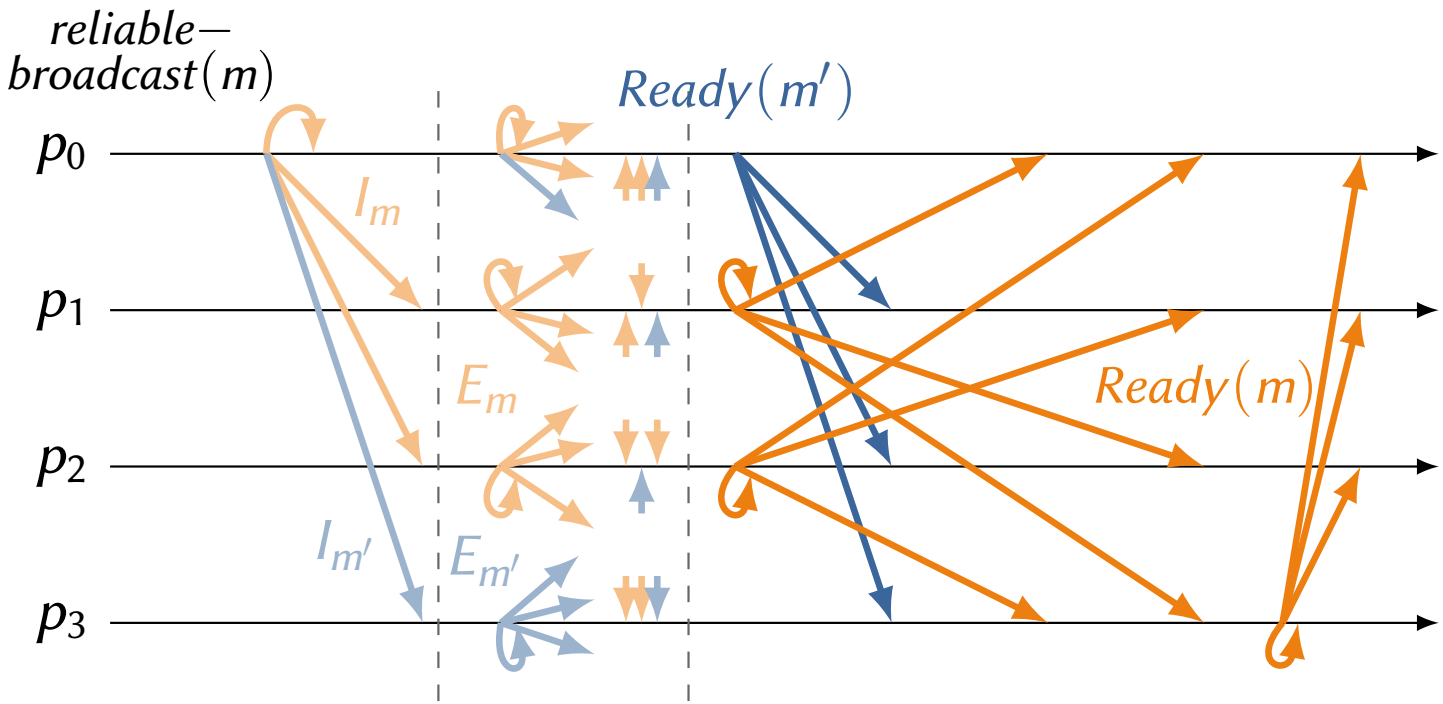
Properties

- After *Echo*: at most one message supported by correct processes
- After *Ready*: all correct processes support the same messages



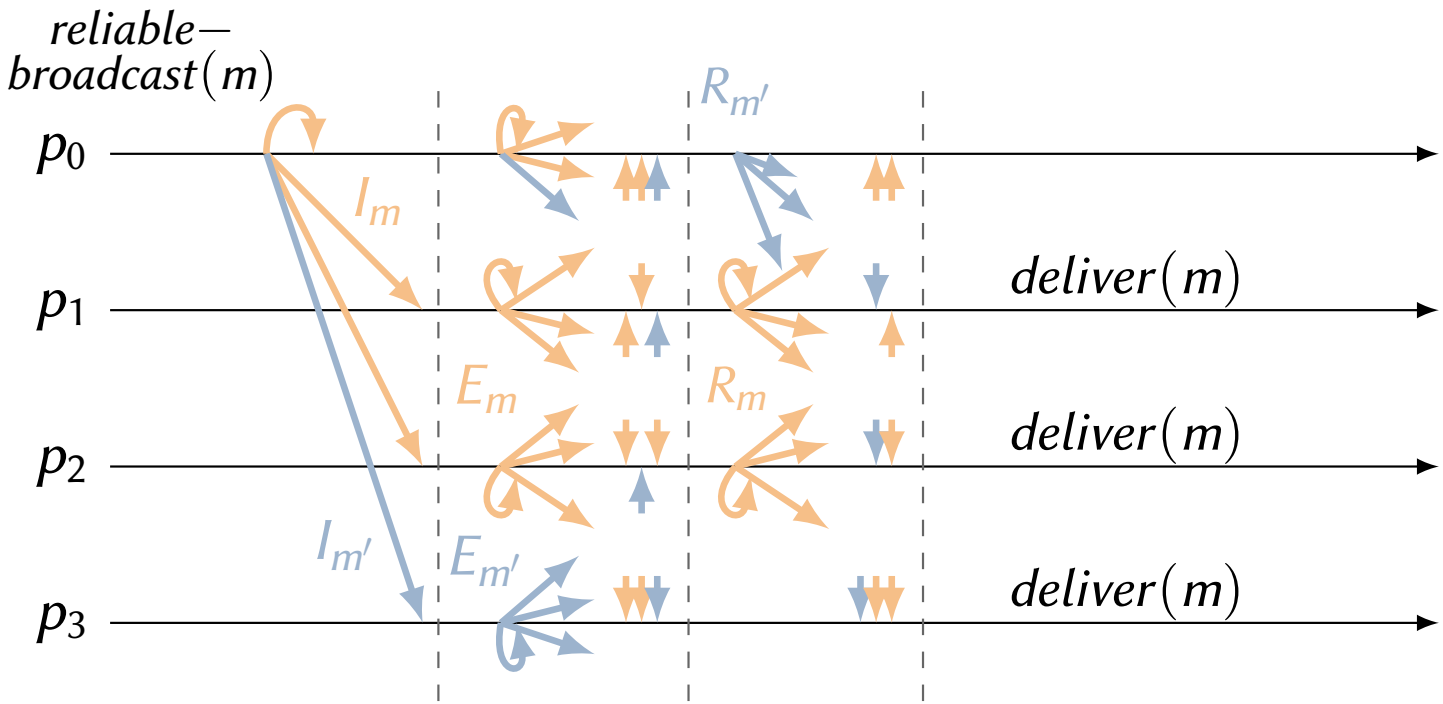
Properties

- After *Echo*: at most one message supported by correct processes
- After *Ready*: all correct processes support the same messages



Properties

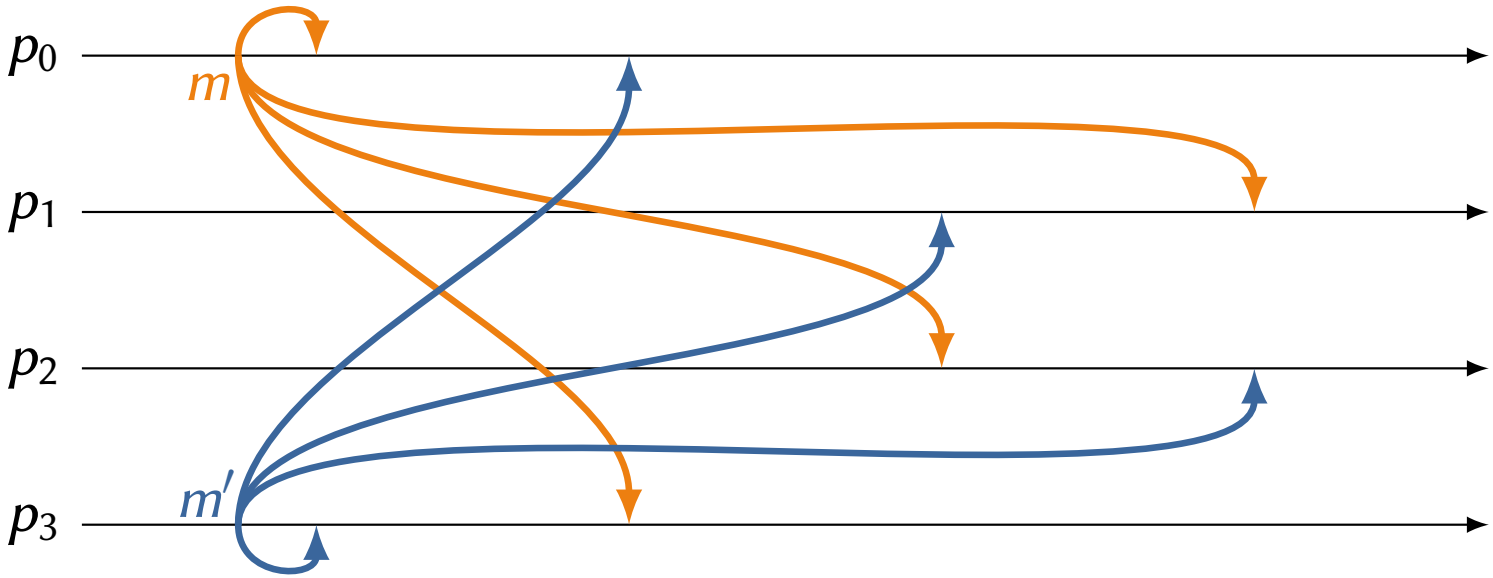
- After *Echo*: at most one message supported by correct processes
- After *Ready*: all correct processes support the same messages



Properties

- After *Echo*: at most one message supported by correct processes
- After *Ready*: all correct processes support the same messages

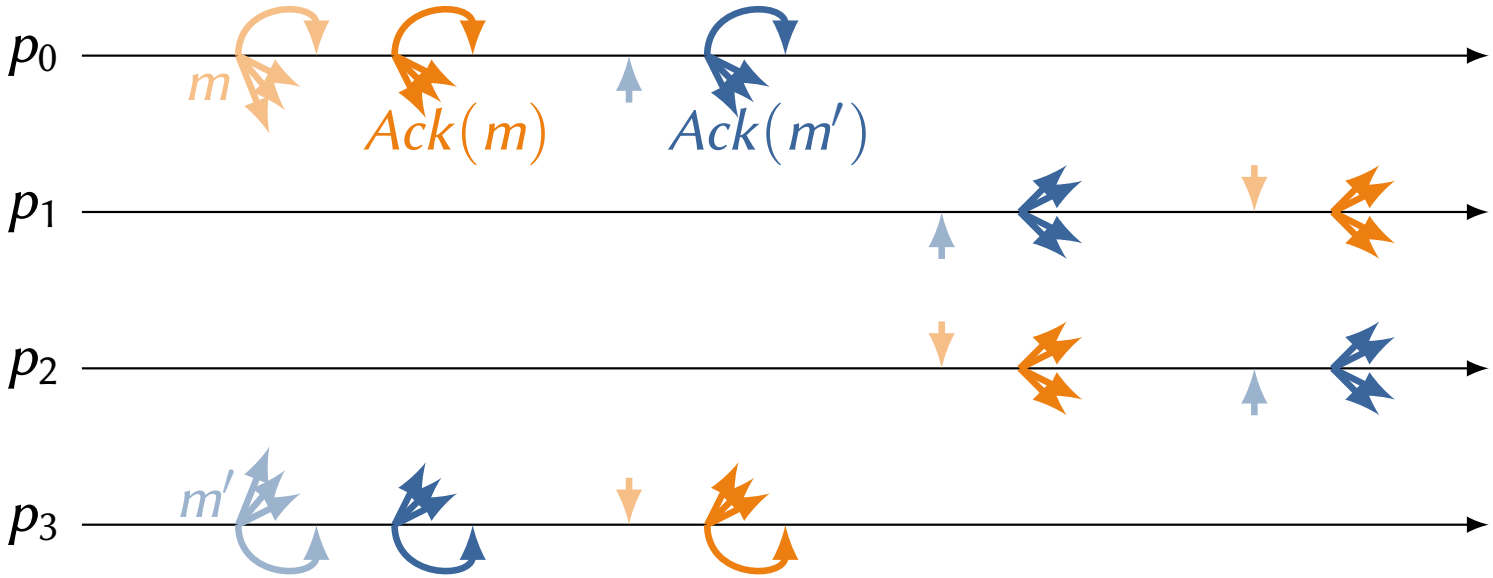
Implementation – Mutual Ordering



$\nexists$ two disjoint majorities of correct processes $\left(> \frac{n+t}{2}\right)$

- If a majority sends $Ack(m)$ before $Ack(m')$
  - p_3 delivers m before m'
- If a majority sends $Ack(m')$ before $Ack(m)$
  - p_0 delivers m' before m
- If there is no clear majority
  - p_0 delivers m' before m and p_3 delivers m before m'

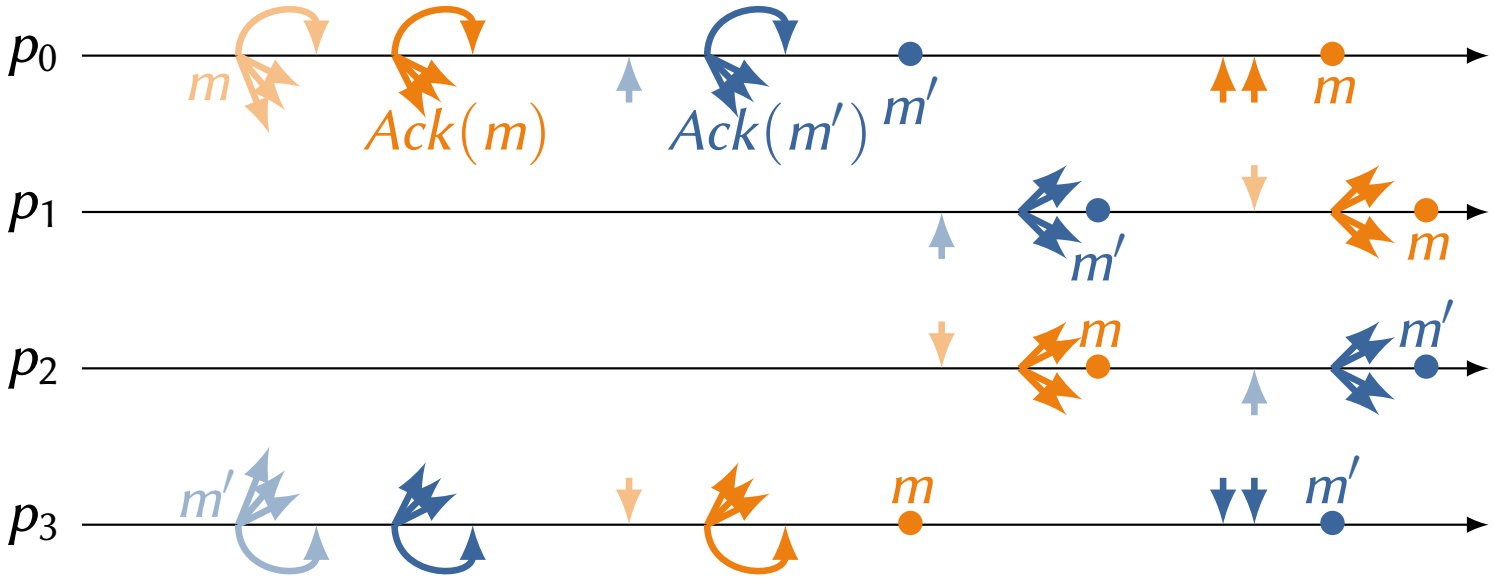
Implementation – Mutual Ordering



$\nexists$ two disjoint majorities of correct processes $\left(> \frac{n+t}{2}\right)$

- If a majority sends $Ack(m)$ before $Ack(m')$
  - p_3 delivers m before m'
- If a majority sends $Ack(m')$ before $Ack(m)$
  - p_0 delivers m' before m
- If there is no clear majority
  - p_0 delivers m' before m and p_3 delivers m before m'

Implementation – Mutual Ordering



$\nexists$ two disjoint majorities of correct processes $\left(> \frac{n+t}{2}\right)$

- If a majority sends $Ack(m)$ before $Ack(m')$
  - p_3 delivers m before m'
- If a majority sends $Ack(m')$ before $Ack(m)$
  - p_0 delivers m' before m
- If there is no clear majority
  - p_0 delivers m' before m and p_3 delivers m before m'

FIFO Ordering

- It suffises to have FIFO point-to-point channels

Causal ordering

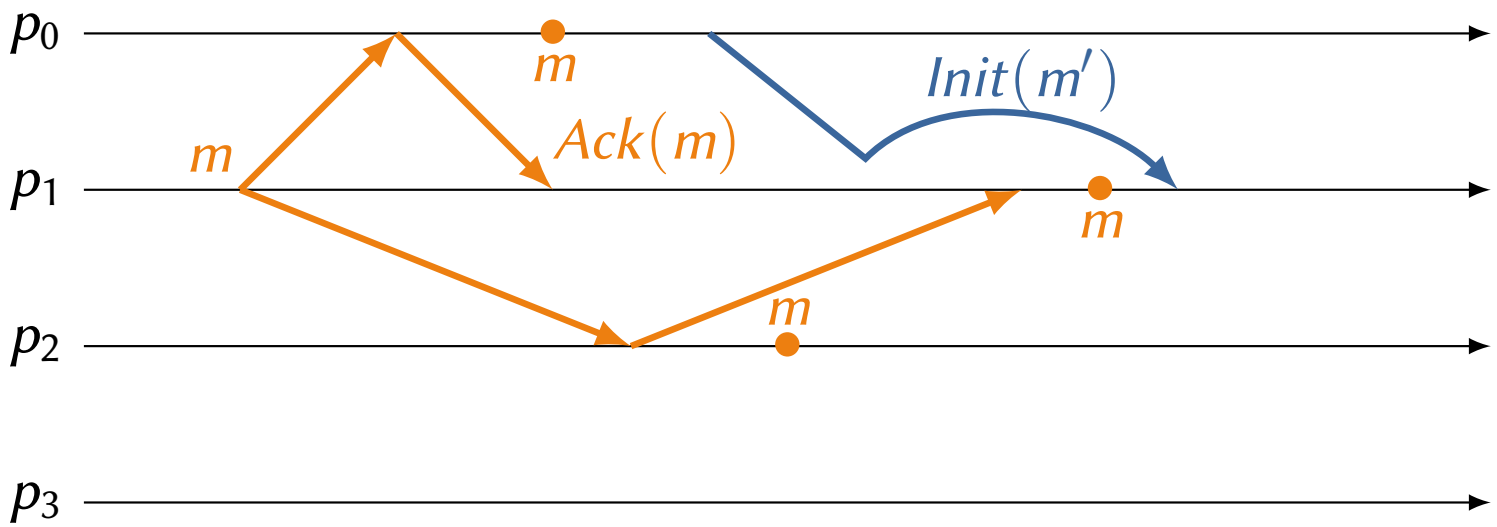
- We can reuse the Ack messages for a FIFO+forward strategy

FIFO Ordering

- It suffises to have FIFO point-to-point channels

Causal ordering

- We can reuse the Ack messages for a FIFO+forward strategy



Contributions

- Specification of CMB-broadcast
- From CMB-broadcast to read/append register
- Implementation for CMB-broadcast
- Proof of the algorithm

Perspectives

- Implementation of CMB-broadcast from read/append registers
- Implementation of CMB-broadcast in open systems?
- Relationship between SCD and CMB?