

Dependable Distributed Systems

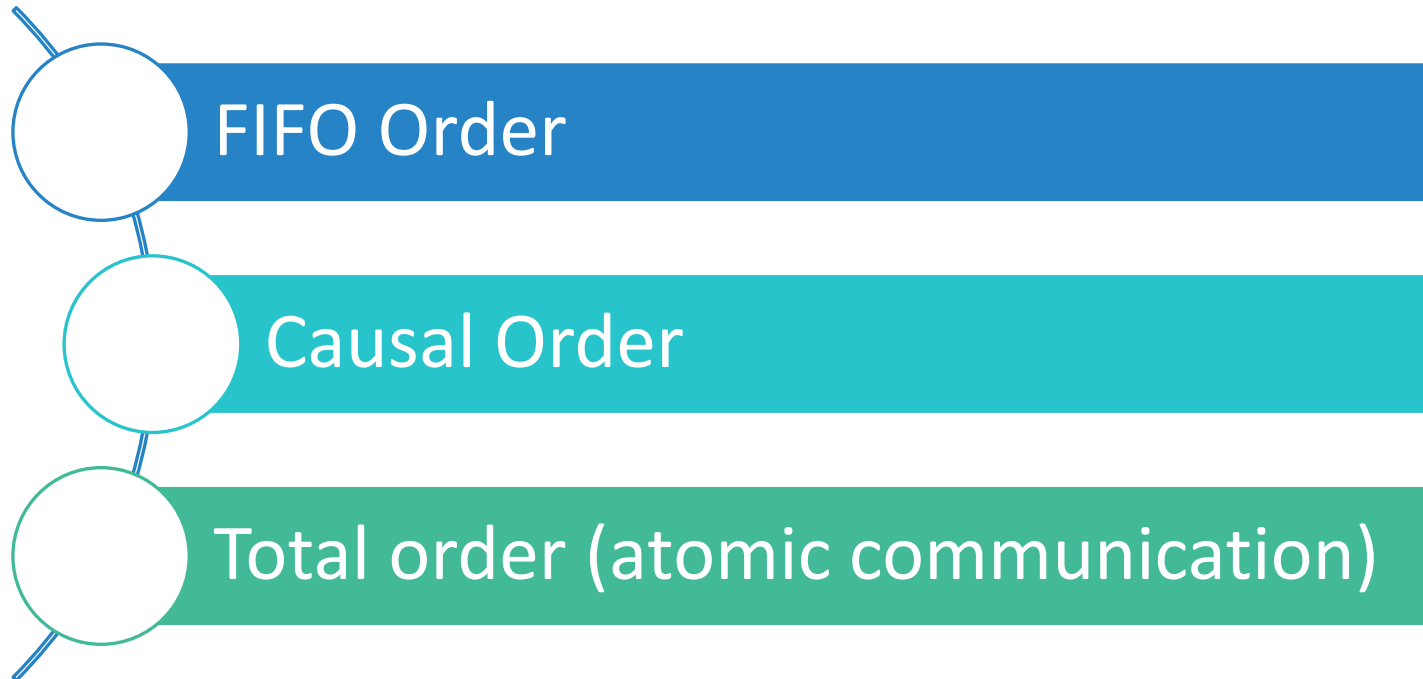
Master of Science in Engineering in Computer Science

AA 2022/2023

LECTURE 12: ORDERED COMMUNICATIONS

Ordered Communication

Define guarantees about the order of deliveries inside group of processes

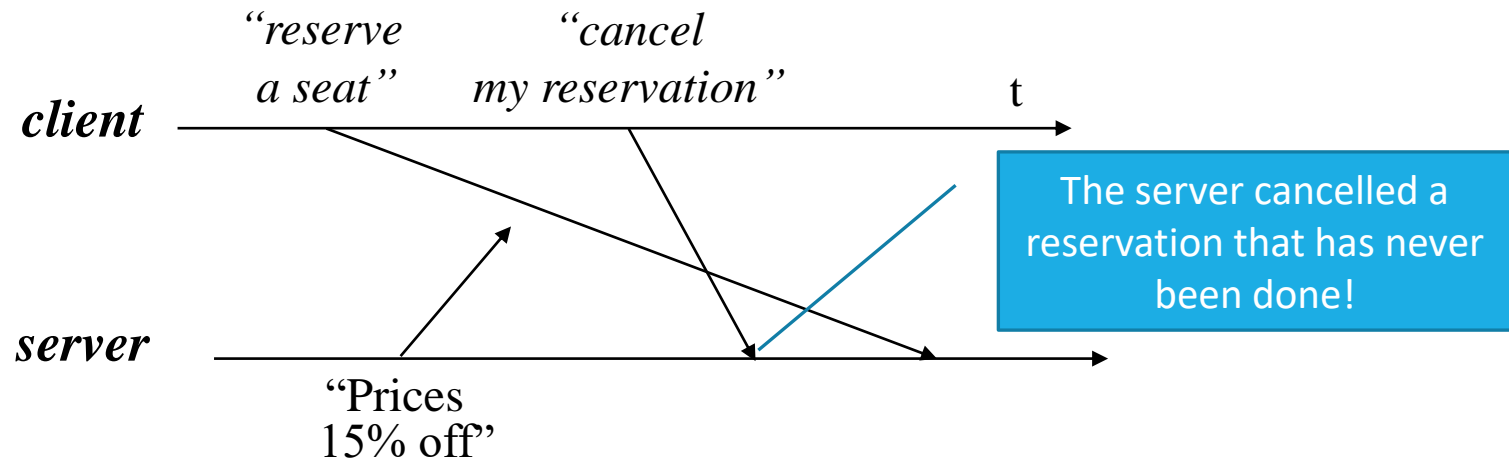


Advantages of ordered communication

Orthogonal wrt reliable communication.

- Reliable broadcast does not have any property on ordering deliveries of messages

This can cause anomalies in many applicative contexts



"Reliable ordered communication" are obtained adding one or more ordering properties to reliable communication

FIFO Broadcast - Specification

FIFO Broadcast can be uniform/non uniform

Module 3.8: Interface and properties of FIFO-order (reliable) broadcast

Module:

Name: FIFOReliableBroadcast, **instance** *frb*.

Events:

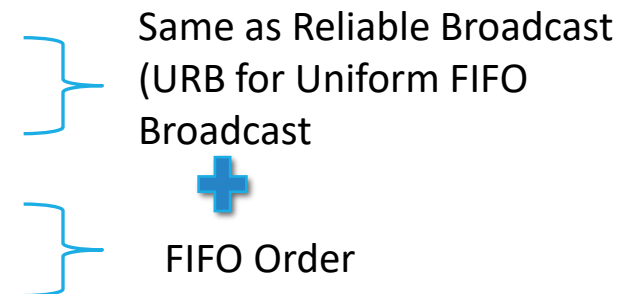
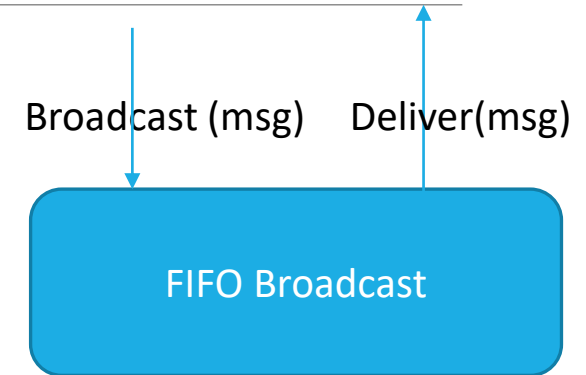
Request: $\langle frb, Broadcast \mid m \rangle$: Broadcasts a message m to all processes.

Indication: $\langle frb, Deliver \mid p, m \rangle$: Delivers a message m broadcast by process p .

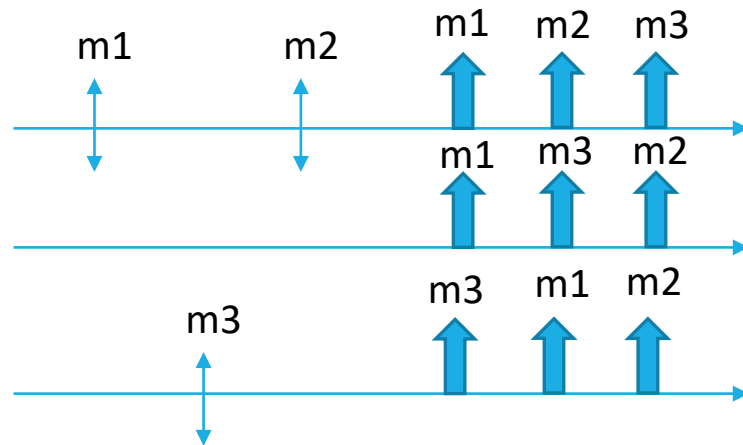
Properties:

FRB1–FRB4: Same as properties RB1–RB4 in (regular) reliable broadcast (Module 3.2).

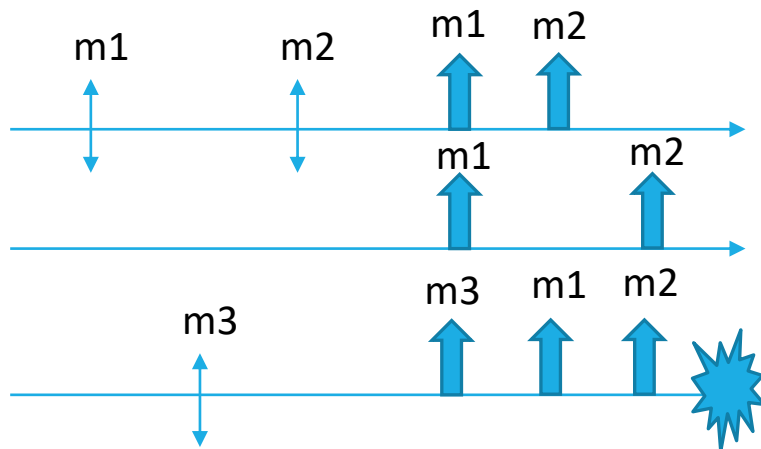
FRB5: FIFO delivery: If some process broadcasts message m_1 before it broadcasts message m_2 , then no correct process delivers m_2 unless it has already delivered m_1 .



Example

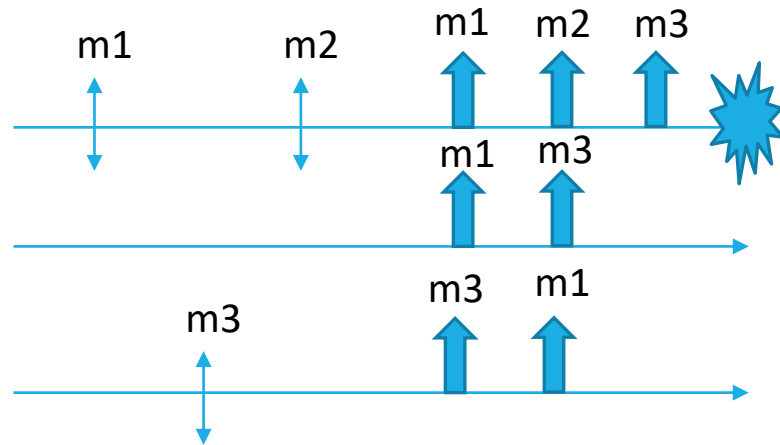


FIFO Uniform Reliable

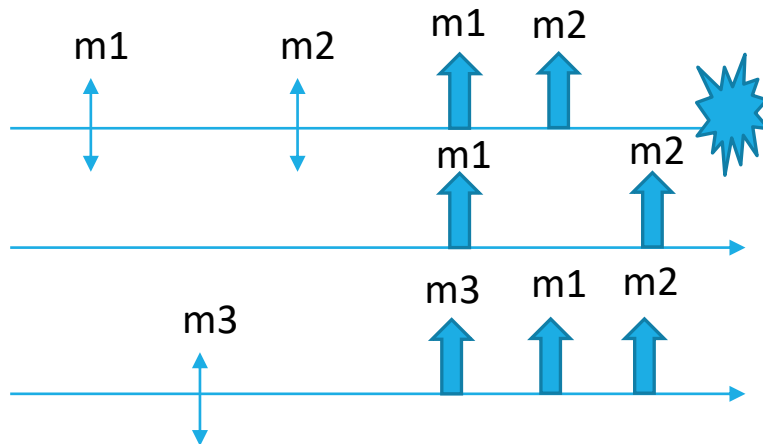


FIFO (Regular) Reliable

Example

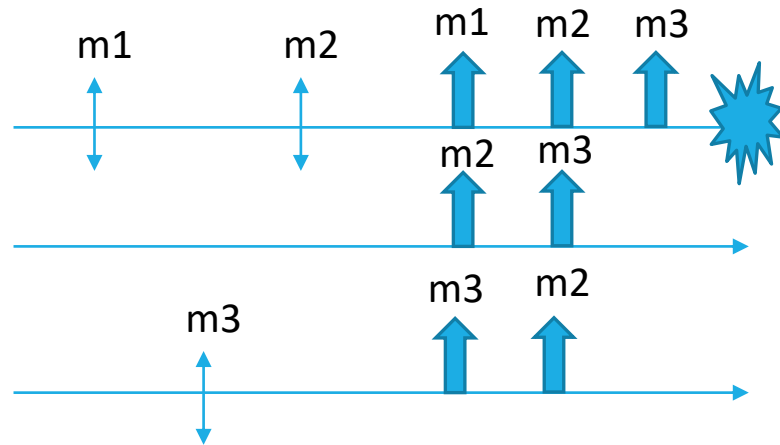


FIFO (Regular) Reliable

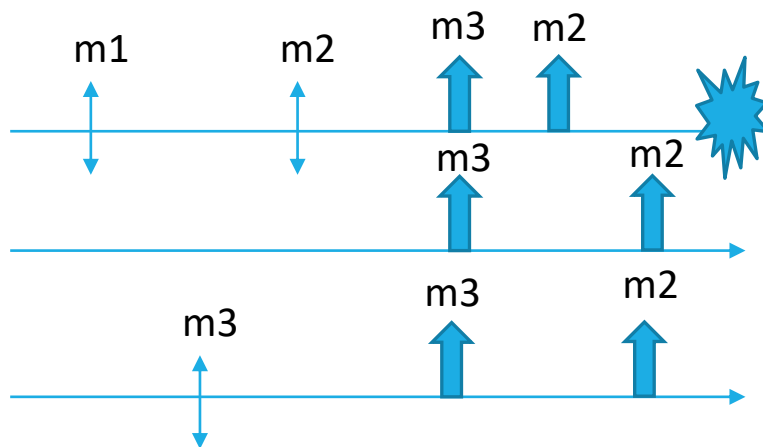


Not Reliable but it satisfies
FIFO Order

Example



Reliable Broadcast but **not**
FIFO Broadcast



Uniform Reliable Broadcast
but not FIFO Broadcast

FIFO Broadcast - Implementation

Algorithm 3.12: Broadcast with Sequence Number

Implements:

FIFOReliableBroadcast, **instance** *frb*.

Uses:

ReliableBroadcast, **instance** *rb*.

upon event $\langle frb, Init \rangle$ **do**

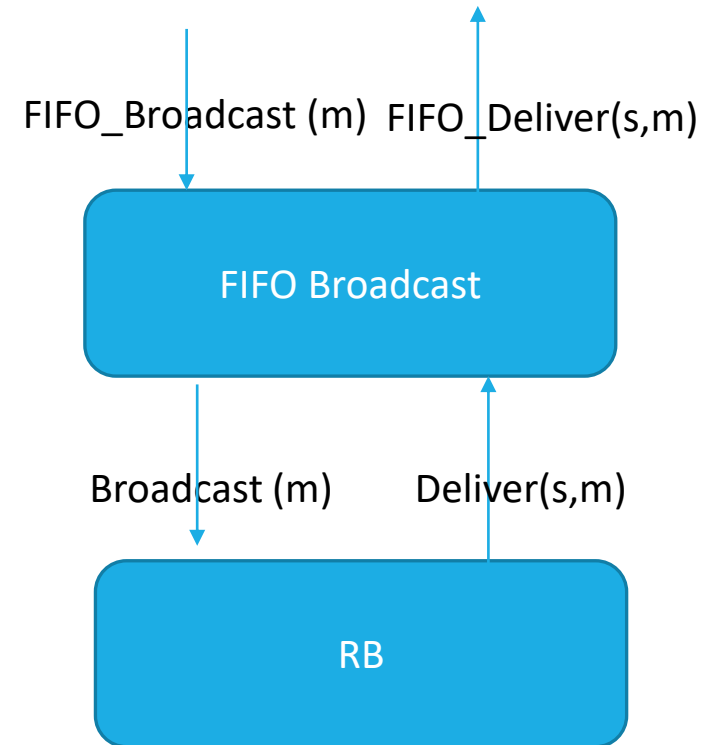
$lsn := 0;$
 $pending := \emptyset;$
 $next := [1]^N;$

upon event $\langle frb, Broadcast \mid m \rangle$ **do**

$lsn := lsn + 1;$
trigger $\langle rb, Broadcast \mid [DATA, self, m, lsn] \rangle;$

upon event $\langle rb, Deliver \mid p, [DATA, s, m, sn] \rangle$ **do**

$pending := pending \cup \{(s, m, sn)\};$
while exists $(s, m', sn') \in pending$ such that $sn' = next[s]$ **do**
 $next[s] := next[s] + 1;$
 $pending := pending \setminus \{(s, m', sn')\};$
trigger $\langle frb, Deliver \mid s, m' \rangle;$

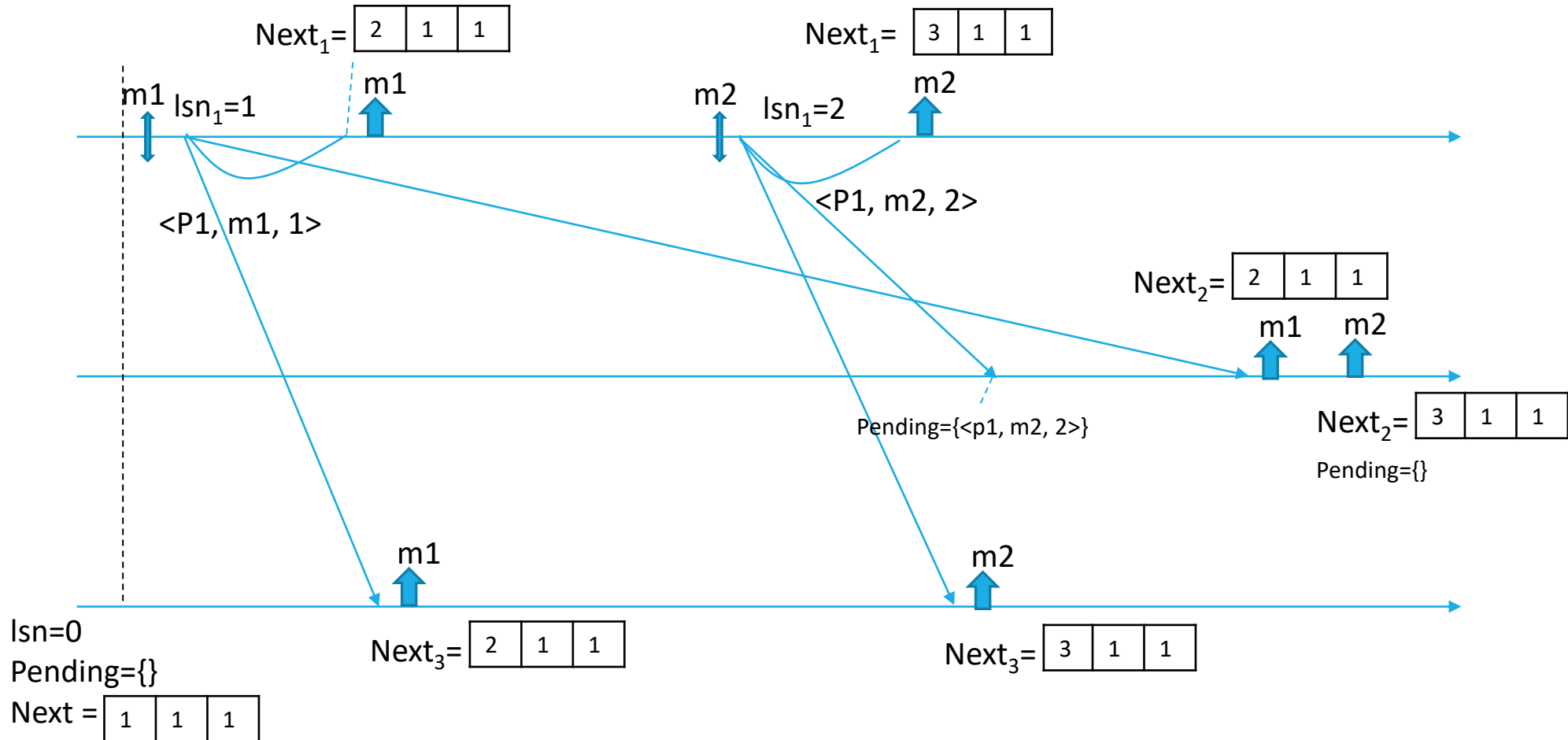



```

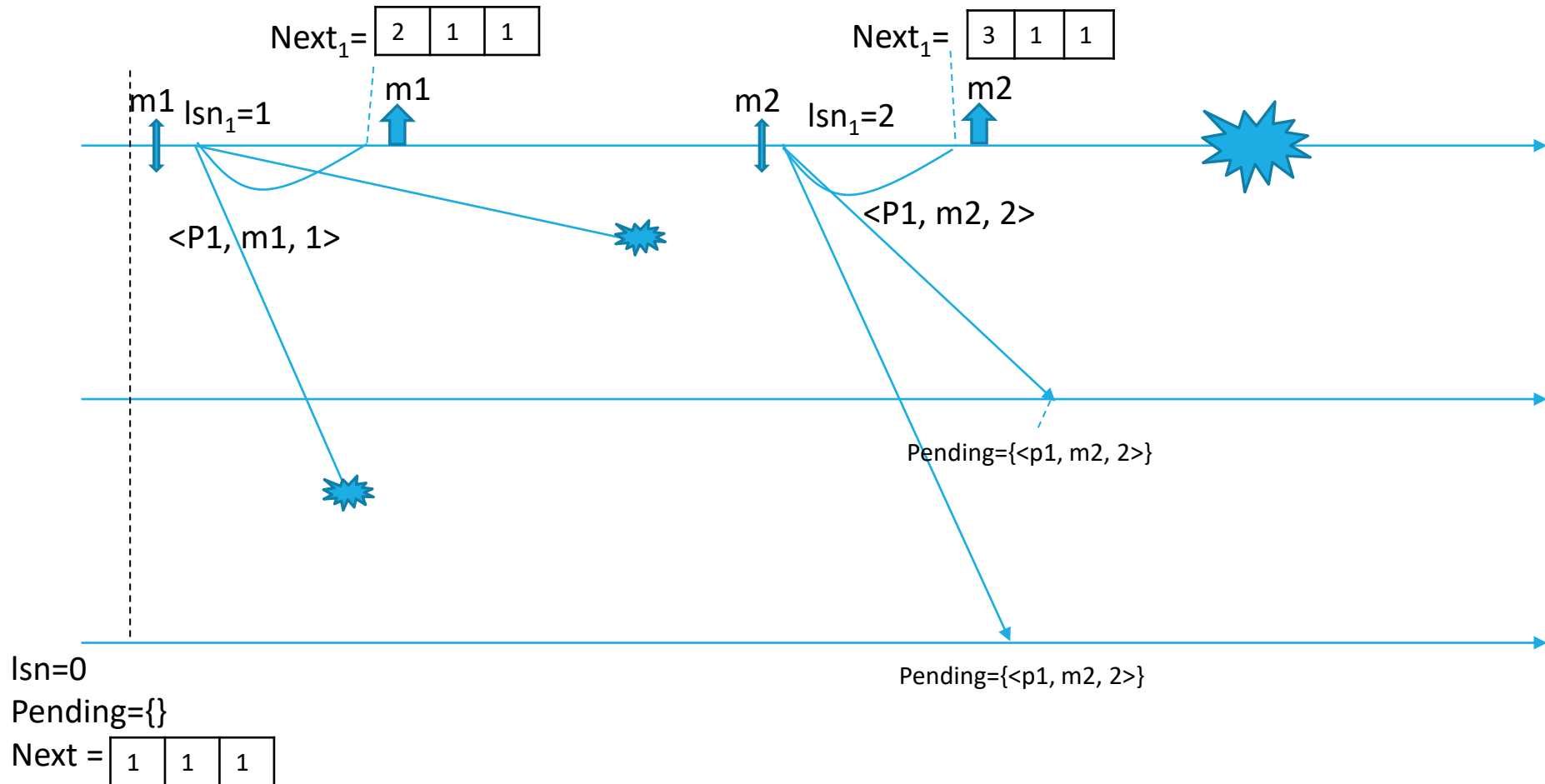
upon event  $\langle rb, Deliver \mid p, [DATA, s, m, sn] \rangle$  do
  pending := pending  $\cup \{(s, m, sn)\}$ ;
  while exists  $(s, m', sn') \in pending$  such that  $sn' = next[s]$  do
    next[s] := next[s] + 1;
    pending := pending  $\setminus \{(s, m', sn')\}$ ;
    trigger  $\langle frb, Deliver \mid s, m' \rangle$ ;

```

Example 1



Example 2



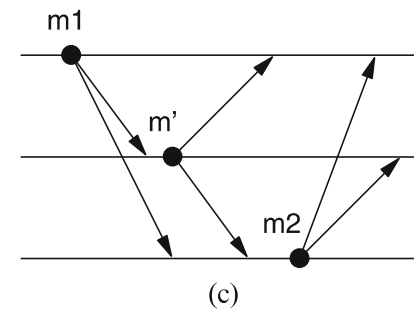
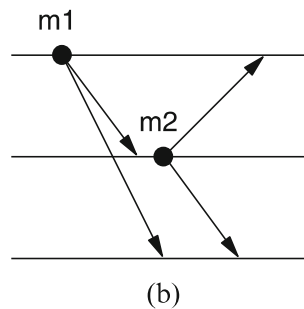
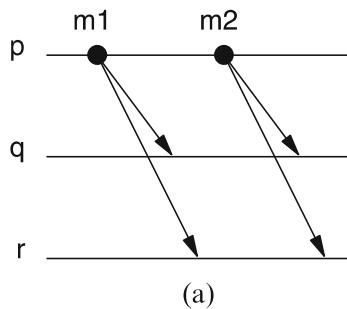
Causal Order Broadcast

Ensures that messages are delivered according to the cause–effect relationships

- Causal order is an extension of the happened-before relation

A message m_1 may have *potentially caused* another message m_2 (denoted as $m_1 \rightarrow m_2$) if any of the following holds:

- some process p broadcasts m_1 before it broadcasts m_2
- some process p delivers m_1 and subsequently broadcasts m_2
- there exists some message m' such that $m_1 \rightarrow m'$ and $m' \rightarrow m_2$



Causal Order Broadcast Specification

Causal Order Broadcast can be uniform/non uniform

Module 3.9: Interface and properties of causal-order (reliable) broadcast

Module:

Name: CausalOrderReliableBroadcast, **instance** *crb*.

Events:

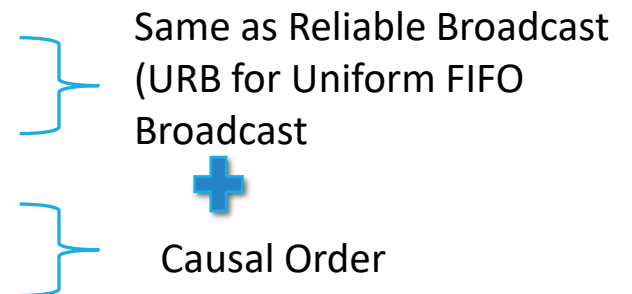
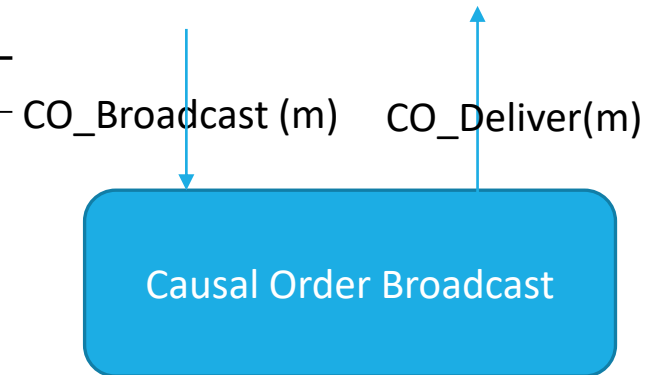
Request: $\langle crb, Broadcast \mid m \rangle$: Broadcasts a message m to all processes.

Indication: $\langle crb, Deliver \mid p, m \rangle$: Delivers a message m broadcast by process p .

Properties:

CRB1–CRB4: Same as properties RB1–RB4 in (regular) reliable broadcast (Module 3.2).

CRB5: *Causal delivery*: For any message m_1 that potentially caused a message m_2 , i.e., $m_1 \rightarrow m_2$, no process delivers m_2 unless it has already delivered m_1 .

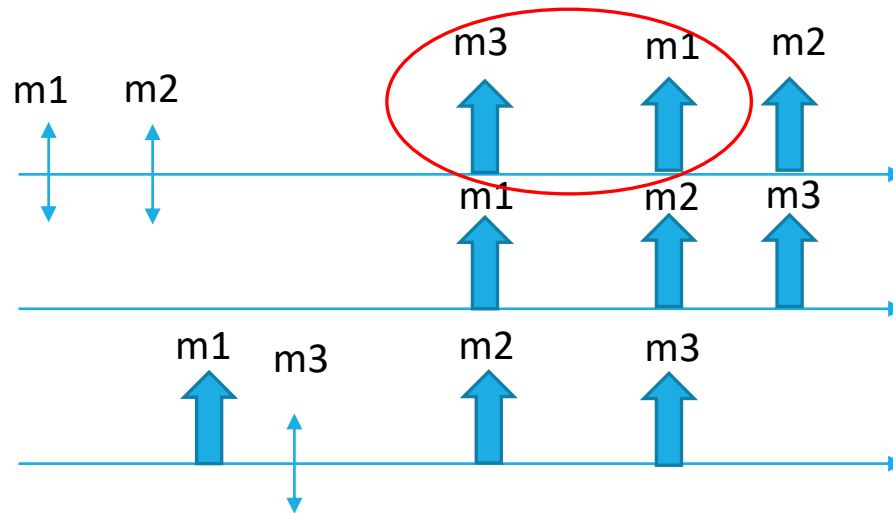


Causal Order Broadcast

Observation

- Causal Broadcast = Reliable Broadcast + Causal Order
 - Causal Order \Rightarrow FIFO Order
 - But FIFO Order \nRightarrow Causal Order
- Causal Order = FIFO Order + Local Order
 - **Local Order:** if a process delivers a message m_a before sending a message m_b , then no correct process deliver m_b if it has not already delivered m_a

Example 1

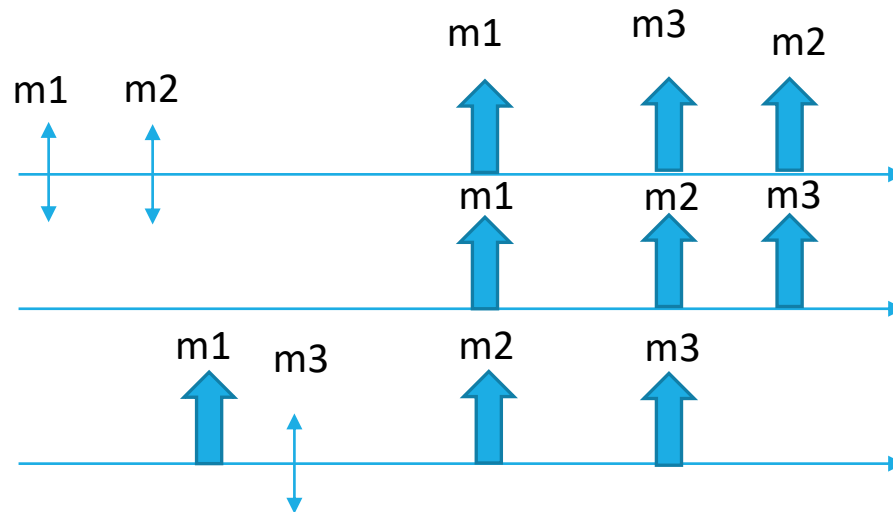


FIFO Reliable but Not Causal

To have causal we need

- m1 -> m2 (FIFO)
- m1 -> m3 (local Order)

Example 2



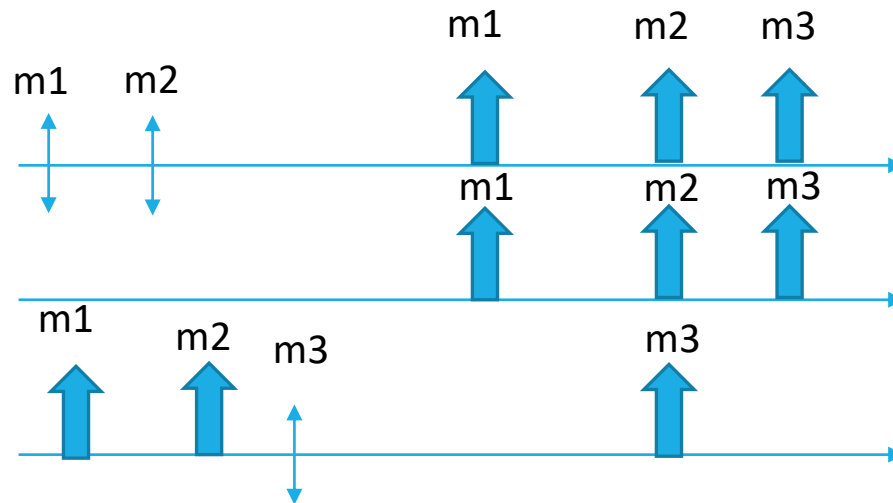
Causal Reliable

- To have causal we need
- *m1* -> *m2* (FIFO)
 - *m1* -> *m3* (local Order)



m1, *m2*, *m3*
m1, *m3*, *m2*

Example 3



Causal Reliable

- To have causal we need
- m1 -> m2 (FIFO)
 - m2 -> m3 (local Order)



m1, m2, m3

Causal Order Broadcast Implementation

Algorithm 3.15: Waiting Causal Broadcast

Implements:

CausalOrderReliableBroadcast, **instance** *crb*.

Uses:

ReliableBroadcast, **instance** *rb*.

upon event $\langle crb, Init \rangle$ **do**

$V := [0]^N$;

$lsn := 0$;

$pending := \emptyset$;

upon event $\langle crb, Broadcast \mid m \rangle$ **do**

$W := V$;

$W[rank(self)] := lsn$;

$lsn := lsn + 1$;

trigger $\langle rb, Broadcast \mid [DATA, W, m] \rangle$;

upon event $\langle rb, Deliver \mid p, [DATA, W, m] \rangle$ **do**

$pending := pending \cup \{(p, W, m)\}$;

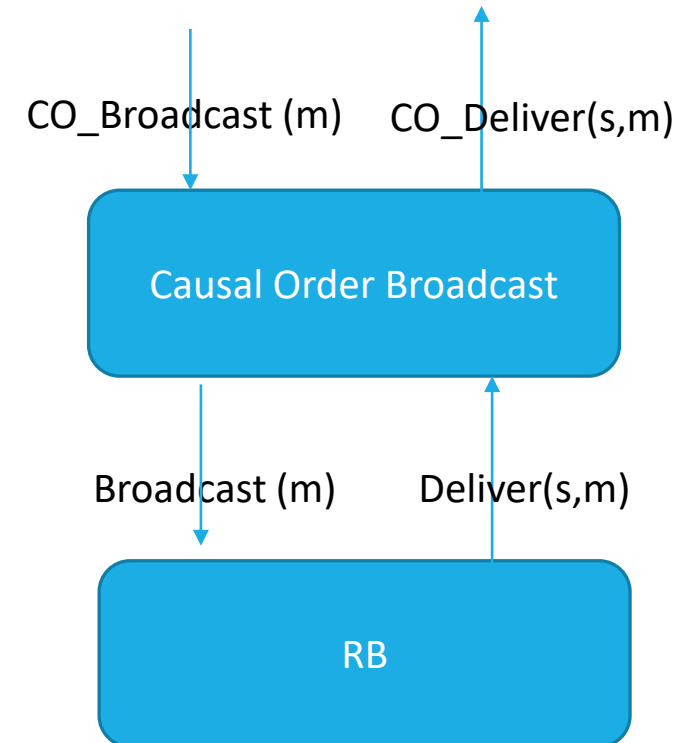
while exists $(p', W', m') \in pending$ such that $W' \leq V$ **do**

$pending := pending \setminus \{(p', W', m')\}$;

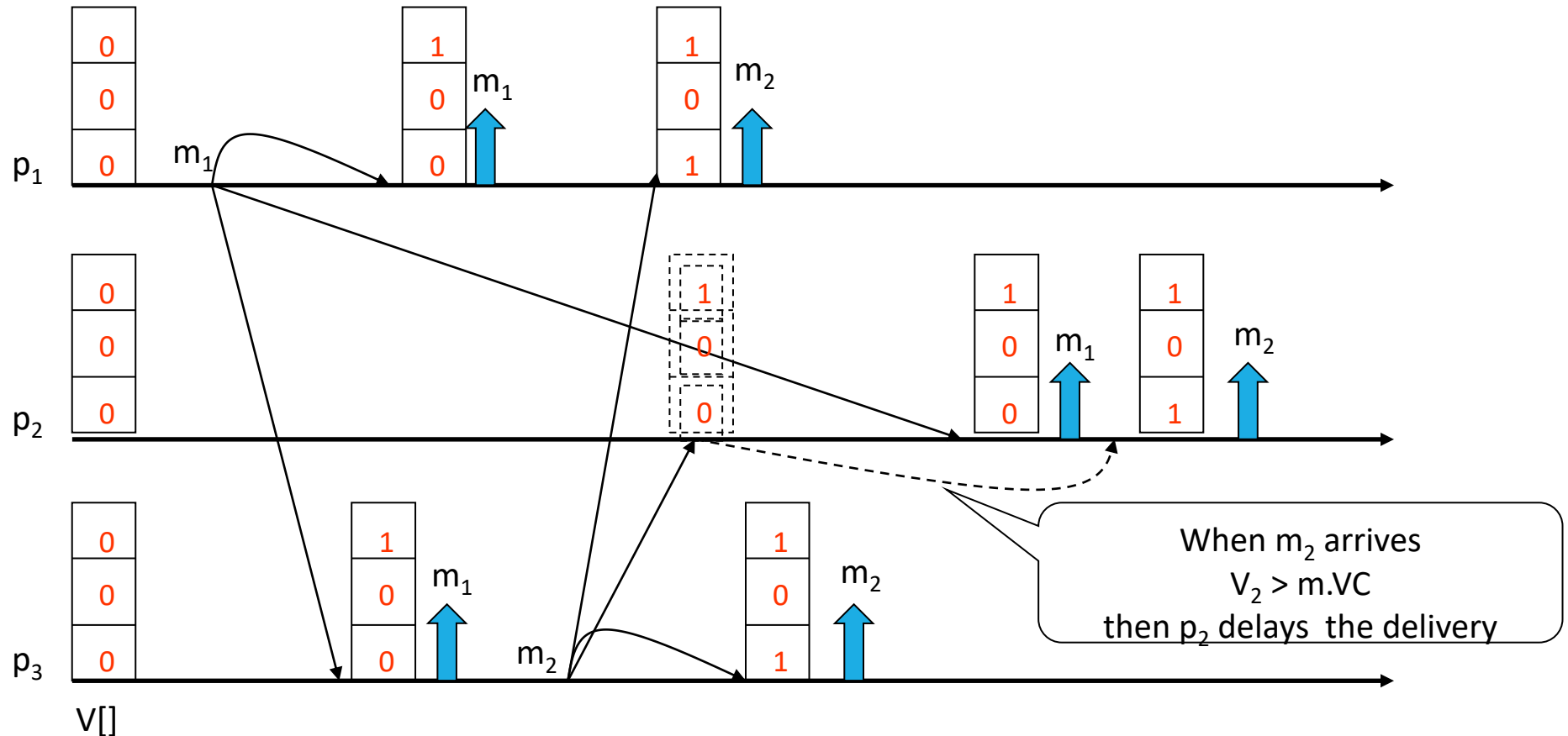
$V[rank(p')] := V[rank(p')] + 1$;

trigger $\langle crb, Deliver \mid p', m' \rangle$;

The function $rank()$
associates an entry of the
vector to each process



Waiting Causal Broadcast example



Causal Order Broadcast: Safety

Property:

- Let two broadcast messages m and m' such that $\text{broadcast}(m) \rightarrow \text{broadcast}(m')$ then each process have to deliver m before m'

Observation:

- if m is the k -th message sent by p_i then $m.Vc[i] = k-1$

Safety property can be proved by induction using the causal ordering relation among broadcast messages

Definition:

- Let two broadcast events b and b' with $b \rightarrow b'$. These events have a **causal distance** k if \exists a sequence of k broadcast events $b_1 \dots b_k$ such that
 - $\forall i \in \{1 \dots k\} \ b_i \rightarrow b_{i+1} \wedge (\neg \exists) \ m^* \mid b_i \rightarrow m^* \rightarrow b_{i+1}$
 - $b \rightarrow b_1$
 - $b_k \rightarrow b'$

Proof – basic case ($K=0$)

Given two messages m, m' such that

- $\text{broadcast}(m) \rightarrow \text{broadcast}(m')$
- There does not exist $\text{broadcast}(m'')$ such that $\text{broadcast}(m) \rightarrow \text{broadcast}(m'') \wedge \text{broadcast}(m'') \rightarrow \text{broadcast}(m')$.

We can have two distinct cases

1. m and m' have been issued by the same process
2. m and m' have been issued by distinct processes

Case 1 – broadcast produced by the same process

- p_j is the receiver
- For line 3 in broadcast procedure
 - $m'.VC[i] := m.VC[i] + 1$.
if m is the h -th message sent by p_i , $m.VC[i] = h - 1$ and $m'.VC[i] = h$.
- A process p_j that receives m' verifies the following delivery condition:
 - $\forall x \in \{1, \dots, n\} \ m'.VC[x] \leq V_j[x] \text{ and } m'.VC[i] \leq V_j[i]$
- $V_i[x]$ is equals to h if and only if the h -th message sent by p_x was delivered by p_i .
(line 3 receive thread).
- Consequently from 2,3,4, m' can be delivered only after the deliver of m .

Case 2 – broadcast produced by distinct processes

m and m' was been sent by distinct processes, respectively p_i e p_j . p_k is the receiver.

$\text{broadcast}(m) \rightarrow \text{broadcast}(m')$, m' was broadcasted by p_j after the deliver of m .
Without loss of generality $m.VC[i]=h-1$

- For line 3 of reception thread e for assumption of $k=0$ we have $m'.VC[i]=h$.

The receiver process p_k respects the following delivery condition:

- $\forall x \in \{1, \dots, n\} m'.VC[x] \leq V_k[x]$ and $m'.VC[i] \leq V_k[i]$

To deliver the message, $m'.VC[i] \leq V_k[i]$, that is $V_k[i] \geq h$

$V_k[i]$ is equals to h if and only if the h -th message sent by p_i has been delivered by p_k .
(line 3 of reception thread thread).

For 2,3,4, p_k can deliver m' only after the deliver of m

Proof – Inductive step($k > 0$)

\exists a sequence of k broadcast events $b_1, b_2 \dots b_k$ such that

$$b \rightarrow b_1 \rightarrow b_2 \rightarrow \dots \rightarrow b_k \rightarrow b'$$

Inductive hypothesis : m has been delivered before m_k

We have to prove that m_k has been delivered before m' .

- It follows from the basic case.

m has been delivered before m' .

Causal Order Broadcast: Liveness

Property:

- Eventually each message will be delivered

Liveness is guaranteed by the following assumptions:

- The number of broadcast events that precedes a certain event is finite
- Channels are reliable

Causal Order Broadcast Implementation

Algorithm 3.13: No-Waiting Causal Broadcast

Implements:

CausalOrderReliableBroadcast, **instance** *crb*.

Uses:

ReliableBroadcast, **instance** *rb*.

upon event $\langle crb, Init \rangle$ **do**

$delivered := \emptyset$;

$past := []$;

upon event $\langle crb, Broadcast \mid m \rangle$ **do**

trigger $\langle rb, Broadcast \mid [DATA, past, m] \rangle$;

$append(past, (self, m))$;

upon event $\langle rb, Deliver \mid p, [DATA, mpast, m] \rangle$ **do**

if $m \notin delivered$ **then**

forall $(s, n) \in mpast$ **do**

if $n \notin delivered$ **then**

trigger $\langle crb, Deliver \mid s, n \rangle$;

$delivered := delivered \cup \{n\}$;

if $(s, n) \notin past$ **then**

$append(past, (s, n))$;

trigger $\langle crb, Deliver \mid p, m \rangle$;

$delivered := delivered \cup \{m\}$;

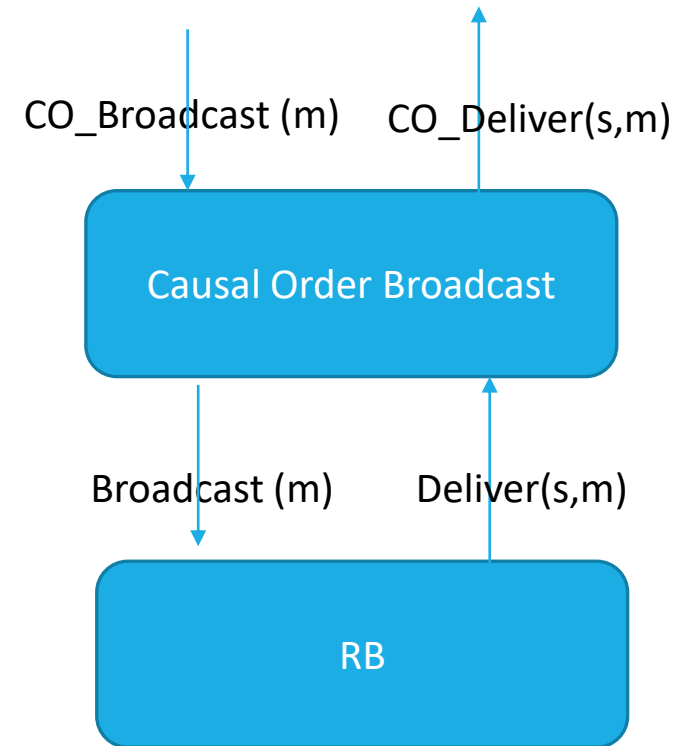
if $(p, m) \notin past$ **then**

$append(past, (p, m))$;

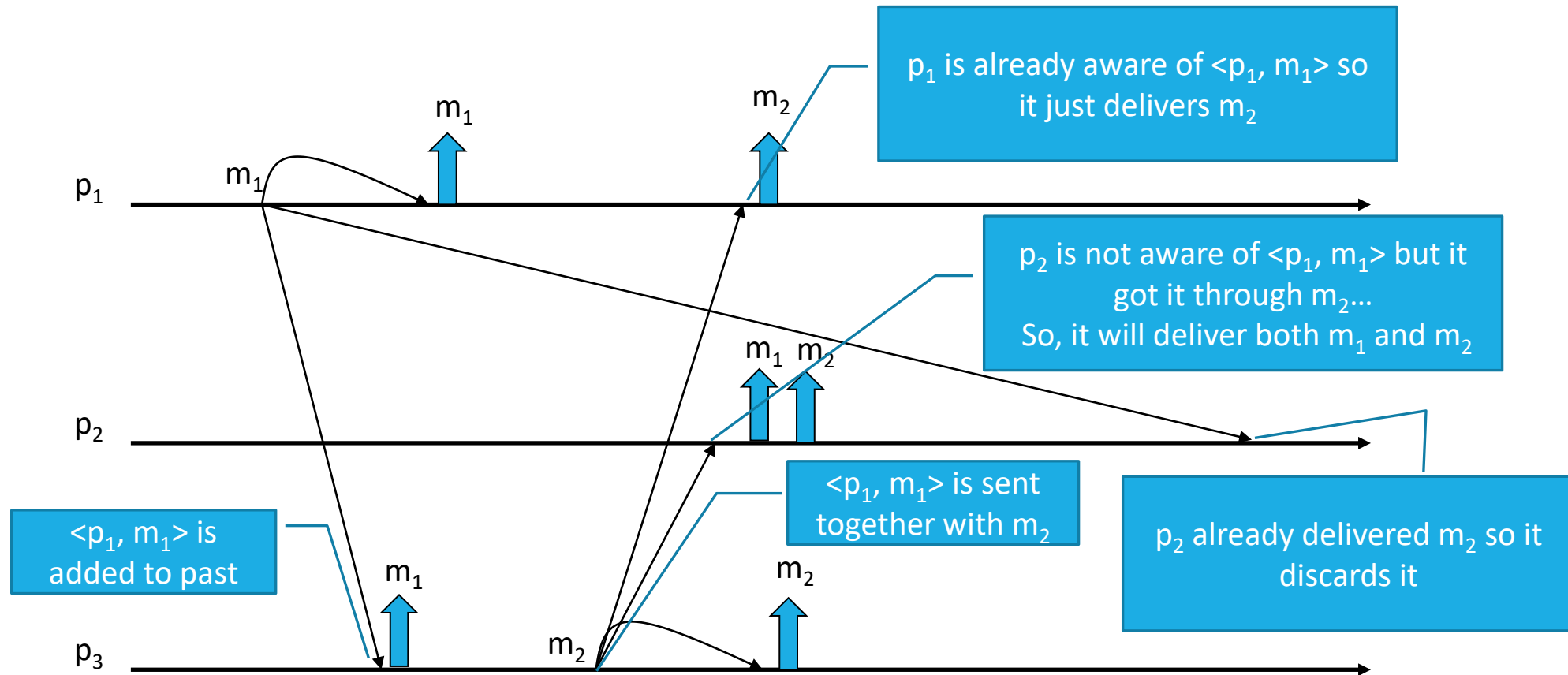
$append(L, x)$ adds an element x at the end of list L

(implicit assumption) No two processes broadcast the same m

by the order in the list

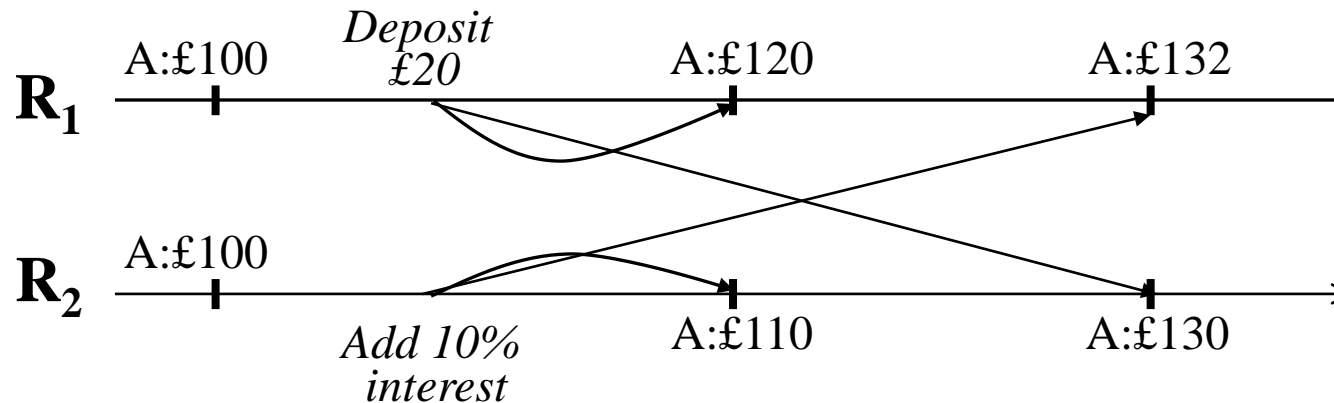


Non-Waiting Causal Broadcast example



Advantages of Ordered Communication

- Causal Order is not enough strong to avoid anomalies
- E.g. Bank account replicated on two sites

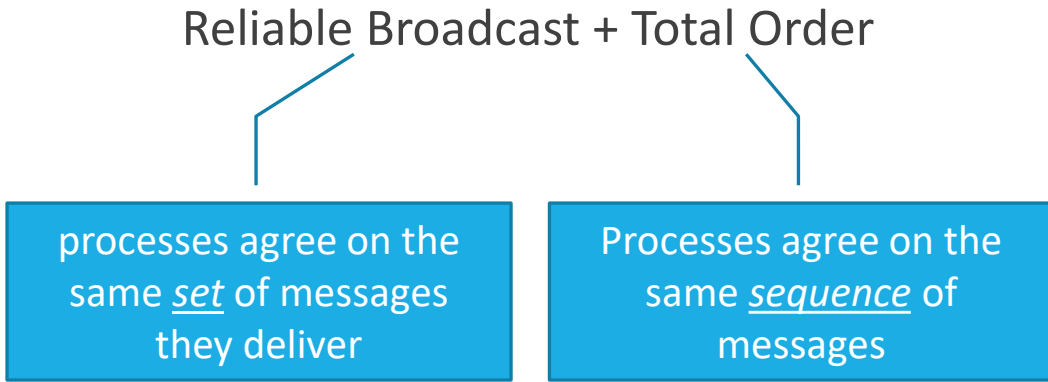


- **same initial state, different final state at the two sites**
- To have the same final state we need to ensure that the order of deliveries is the same at each process.
- Note that ensuring the same delivery order at each replicas does not consider the sending order of messages

Total Order Broadcast

A *total-order (reliable) broadcast* abstraction orders all messages, even those from different senders and those that are not causally related

Reliable Broadcast + Total Order



processes agree on the same set of messages they deliver

Processes agree on the same sequence of messages

The total-order broadcast abstraction is sometimes also called atomic broadcast

message delivery occurs as if the broadcast were an indivisible “atomic” action

the message is delivered to all or to none of the processes and, if the message is delivered, every other message is ordered either before or after this message.

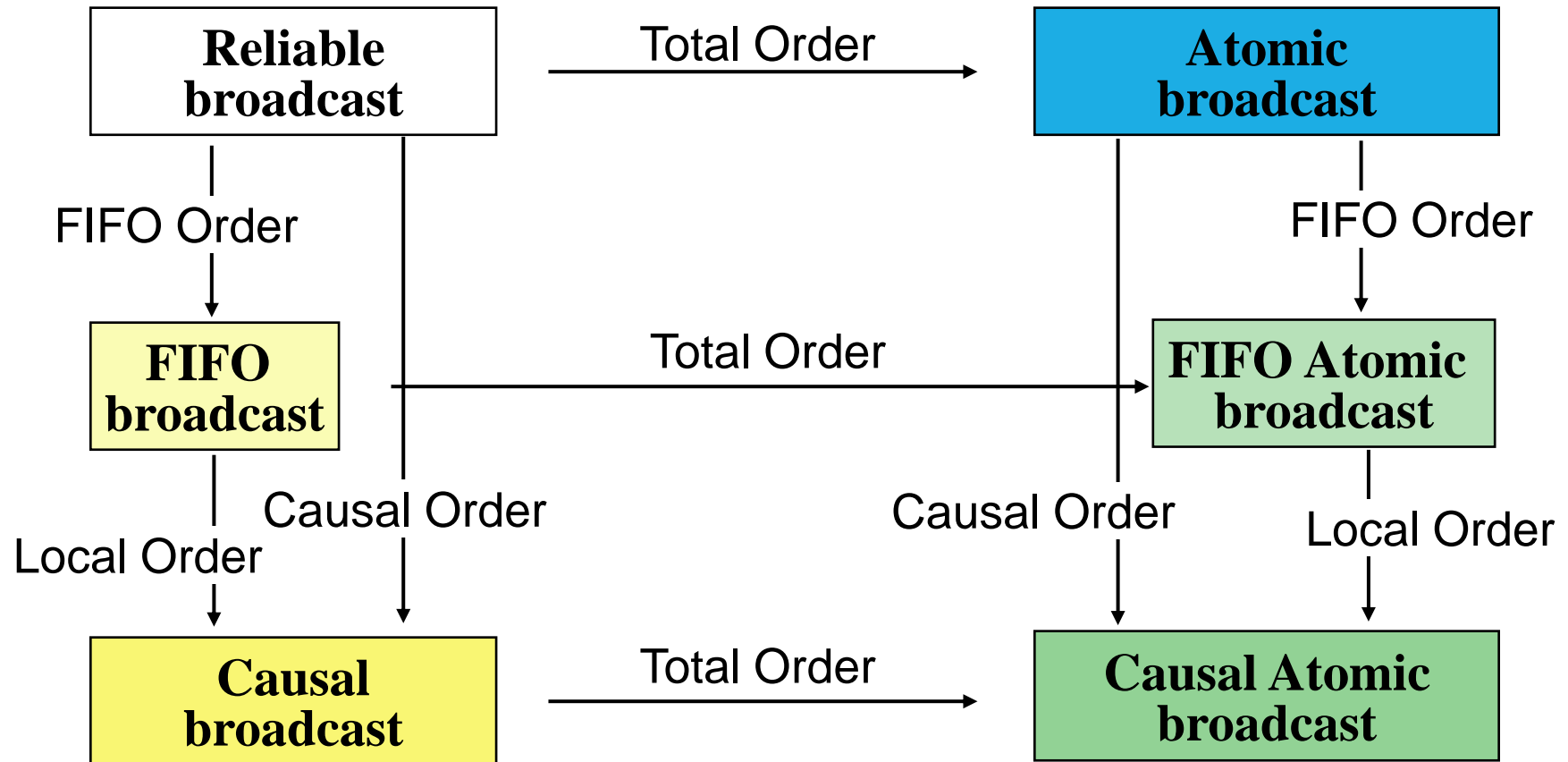
Total Order Broadcast

Total order is orthogonal with respect to FIFO and Causal Order.

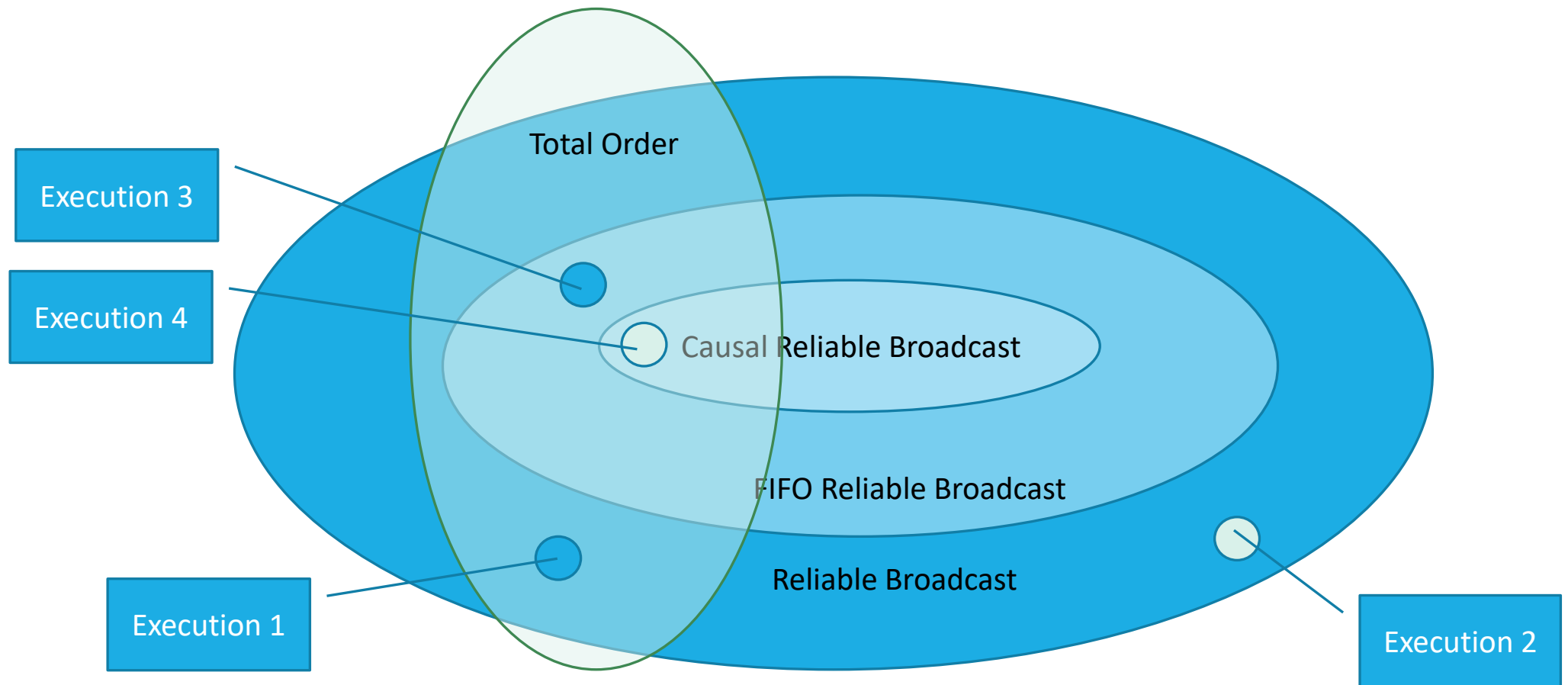
Total order would accept indeed a computation in which a process p_i sends n messages to a group, and each of the processes of the group delivers such messages in the reverse order of their sending.

_> The computation is totally ordered but it is not FIFO.

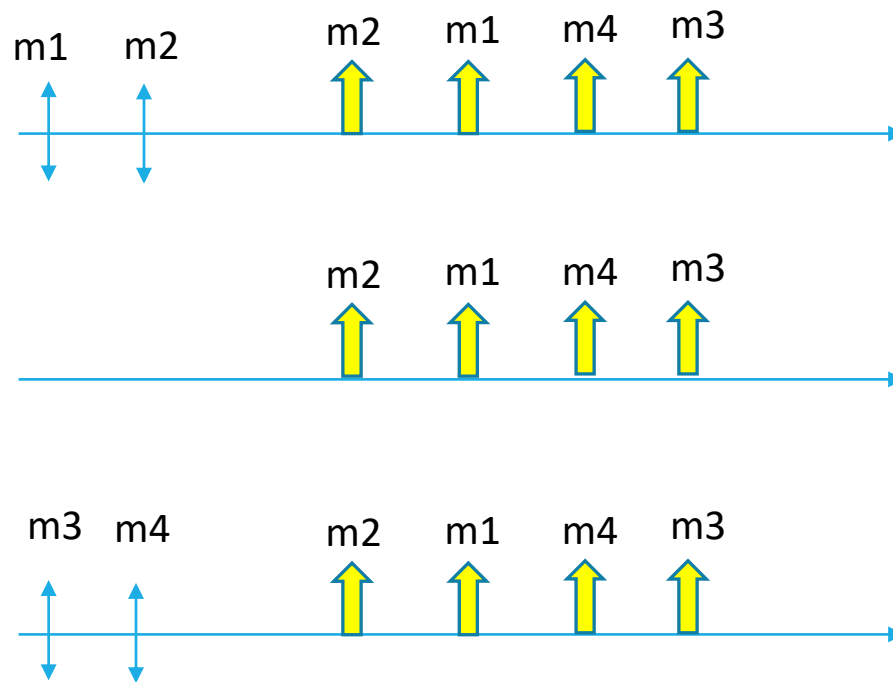
Relationship between Broadcast Specifications



Let's Identify



Execution 1

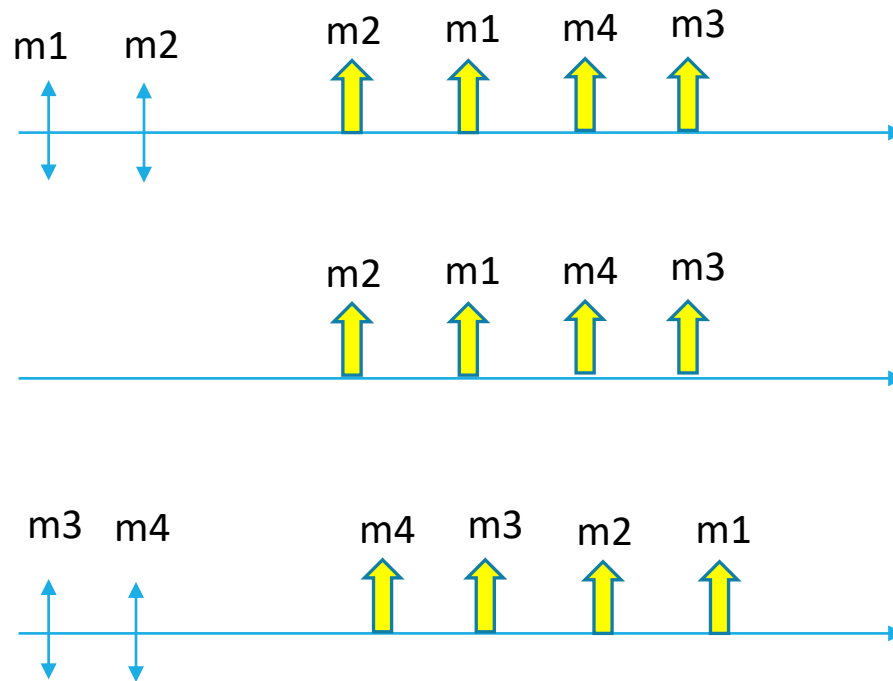


Reliable, Total, not FIFO

FIFO Order

- m1 -> m2
- m3 -> m4

Execution 2

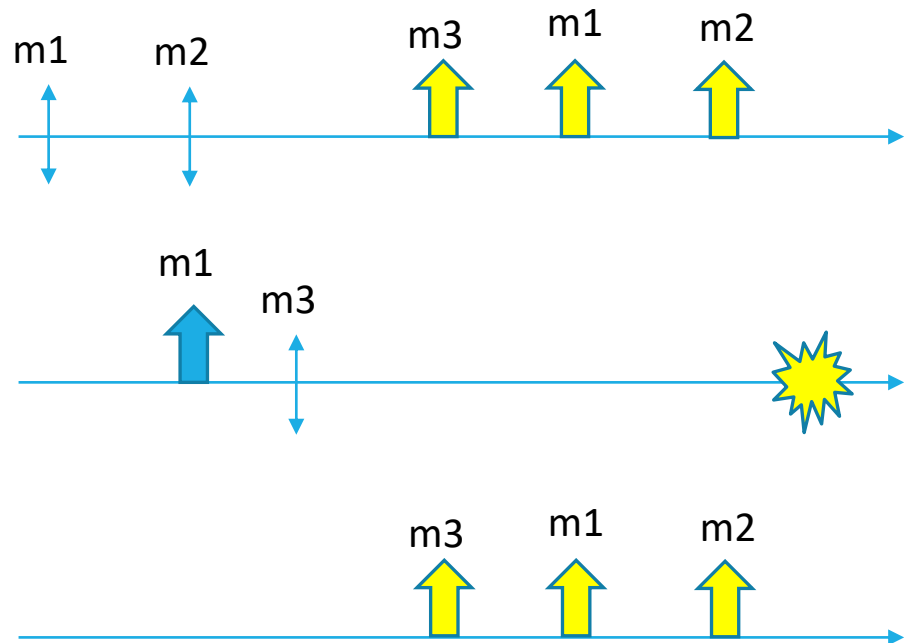


Reliable, not total, not FIFO

FIFO Order

- $m1 \rightarrow m2$
- $m3 \rightarrow m4$

Execution 3

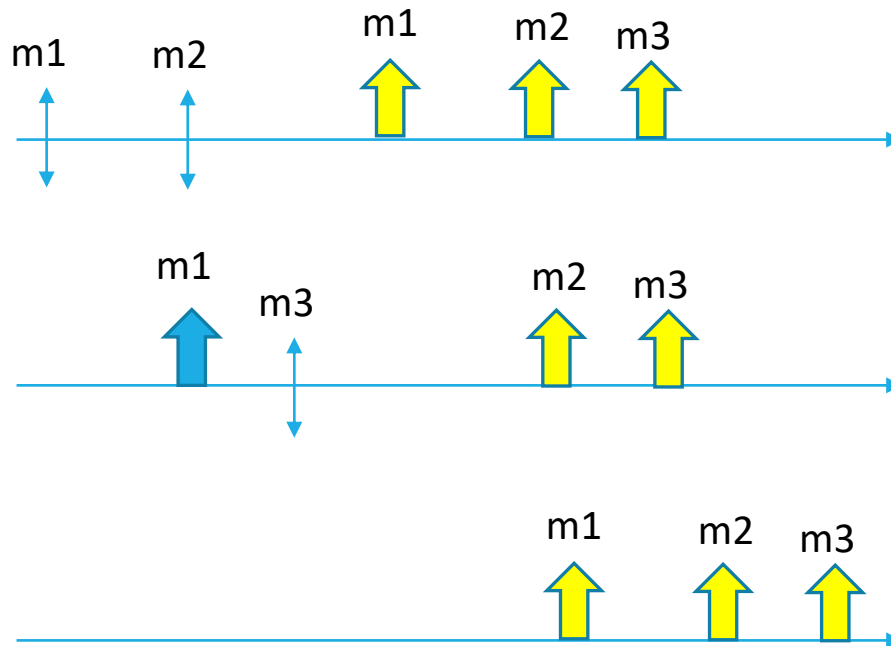


Total, FIFO, Not Causal

Ordering Relationships

- $m1 \rightarrow m2$ (FIFO order)
- $m1 \rightarrow m3$ (local order)

Execution 4



Total, FIFO, Causal

Ordering Relationships

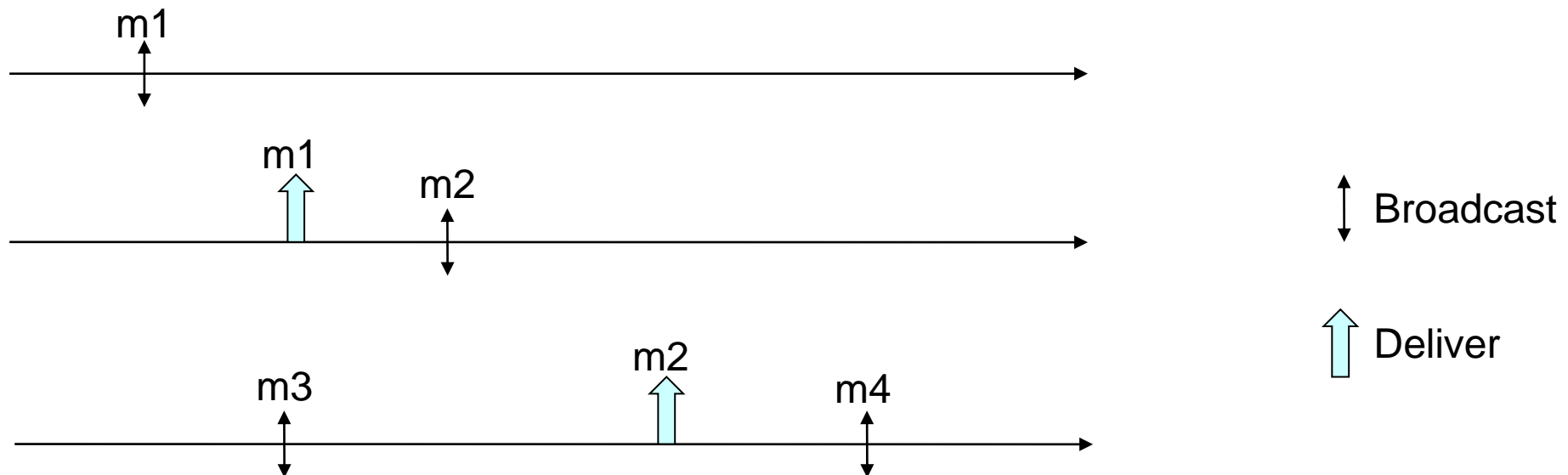
- m1 -> m2 (FIFO order)
- m1 -> m3 (local order)



- m1, m2, m3
- m1, m3, m2

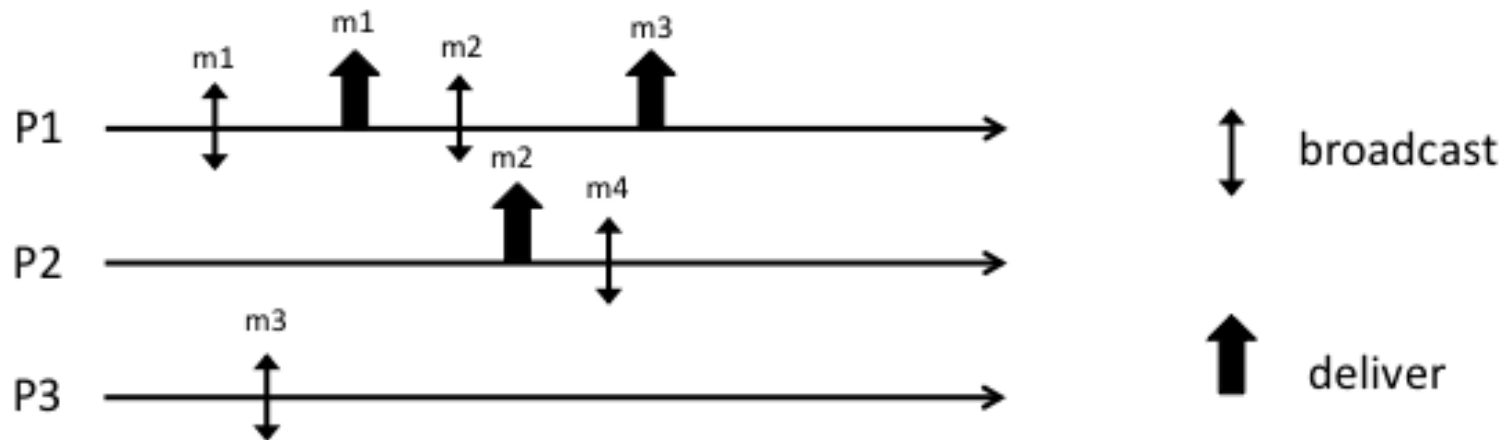
Exercise

Provide all the delivery sequences such that both total order and causal order are satisfied.



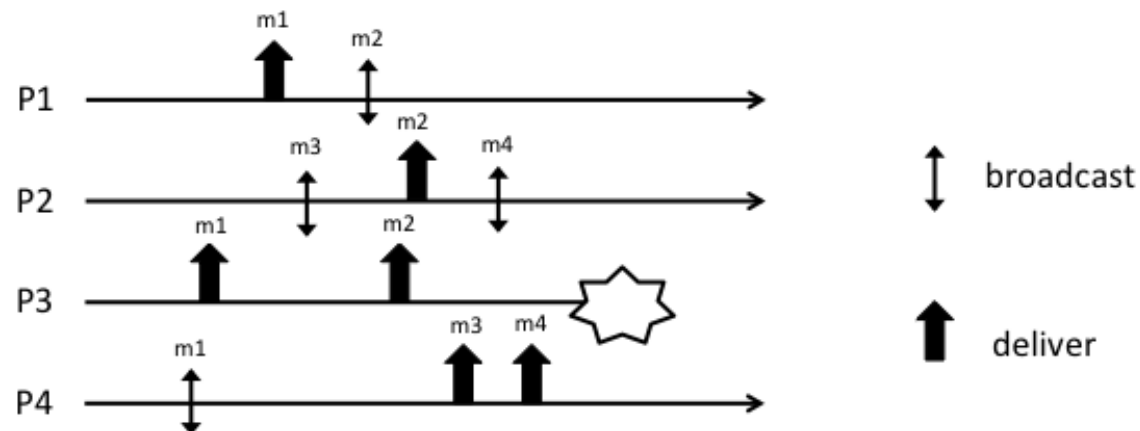
Exercise

1. Provide all the possible sequences satisfying Causal Order
2. Complete the execution in order to have a run satisfying FIFO order but not causal order



Exercise

1. Provide the list of all the possible delivery sequences that satisfy both Total Order and Causal Order
2. Complete the history (by adding the missing delivery events) in order to satisfy Total Order but not Causal Order
3. Complete the history (by adding the missing delivery events) in order to satisfy FIFO Order but not Causal Order nor Total Order



References

C. Cachin, R. Guerraoui and L. Rodrigues. Introduction to Reliable and Secure Distributed Programming, Springer, 2011

- Chapter 3 - from Section 3.9 (except 3.9.6)