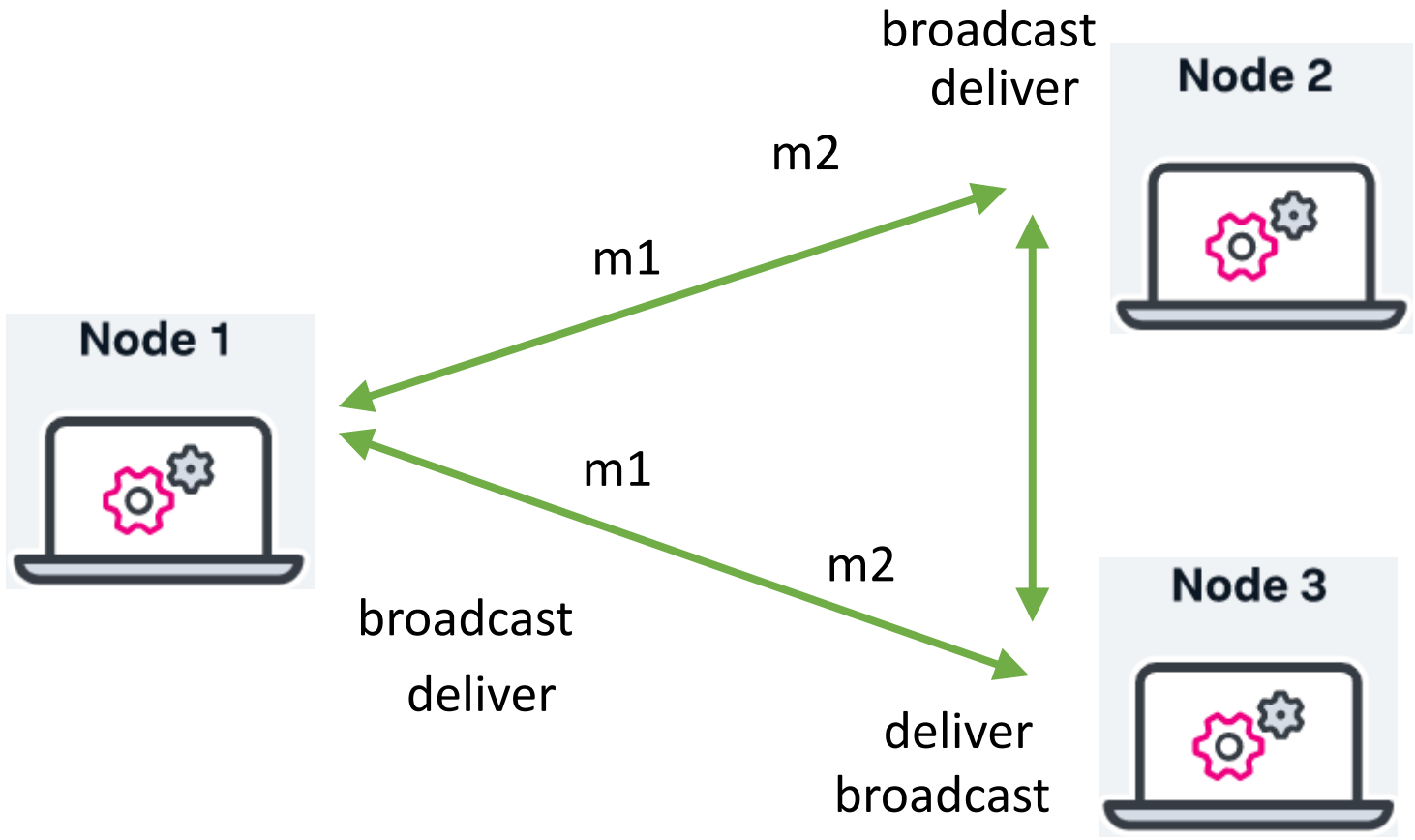
Total Order Broadcast

Mohsen Lesani

Overview

- **Intuitions**: what total order broadcast can bring?
- **Specifications** of **total order broadcast**
- **Consensus**-based total order algorithm

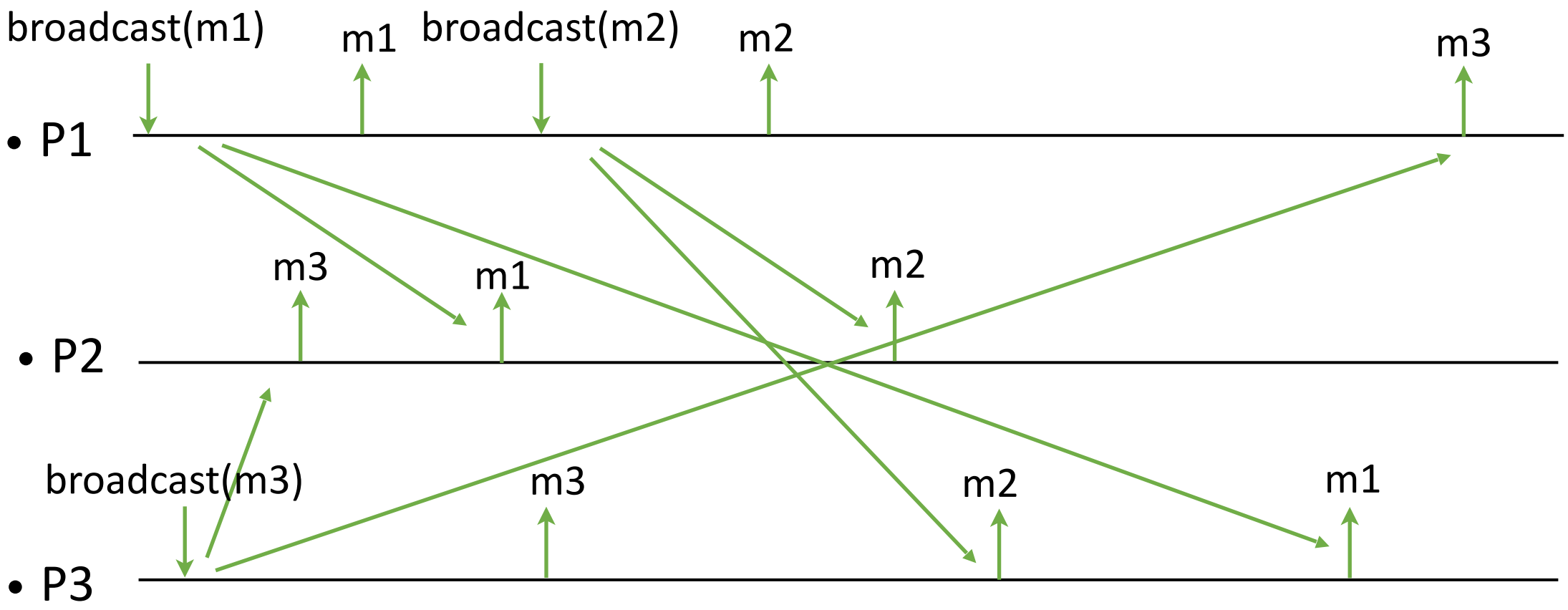
Broadcast


broadcast
deliver
Node 2
m2
m1
Node 1
m1
m2
broadcast
deliver
Node 3
deliver
broadcast

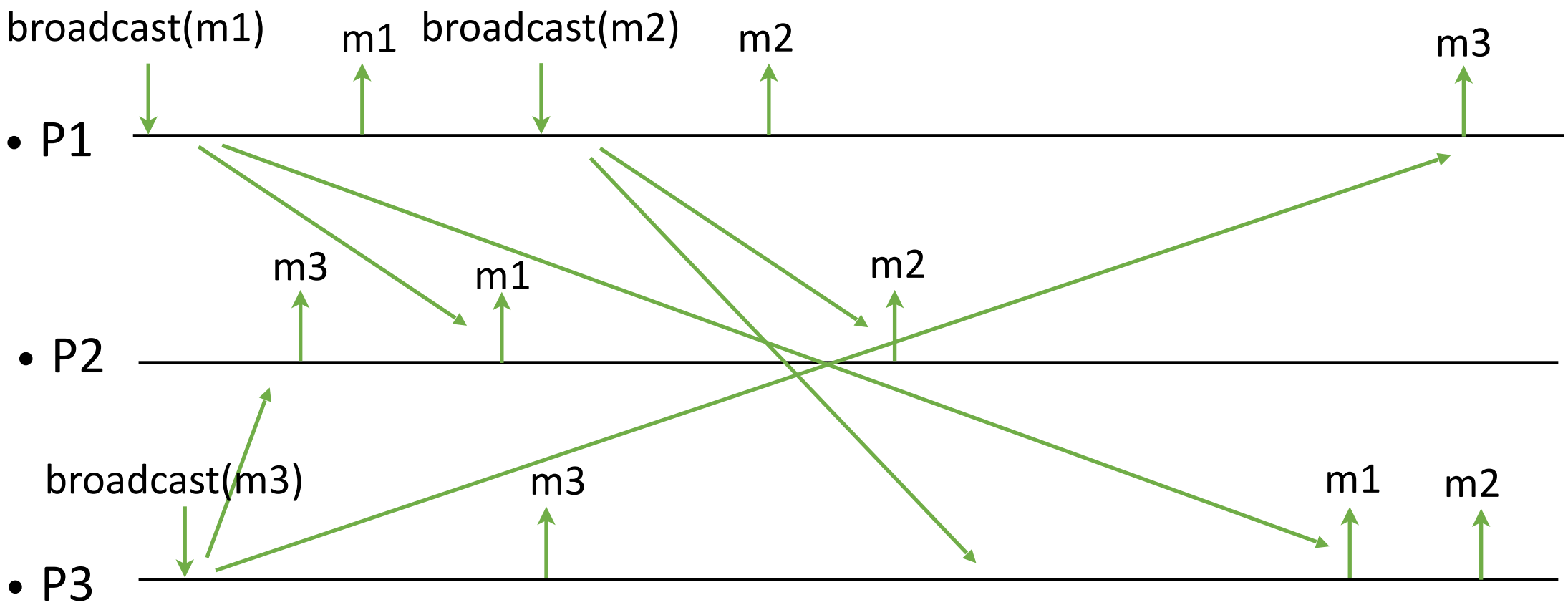
Intuitions (1)

- In **reliable** broadcast, the processes are free to deliver messages in any order they wish
- In **causal** broadcast, the processes need to deliver messages according to some order (causal order)
- The order imposed by causal broadcast is however partial: some messages might be delivered in different orders by different processes

Casual Broadcast



Reliable Broadcast



There is not causality between m3 and (m1 and m2).

Intuitions (3)

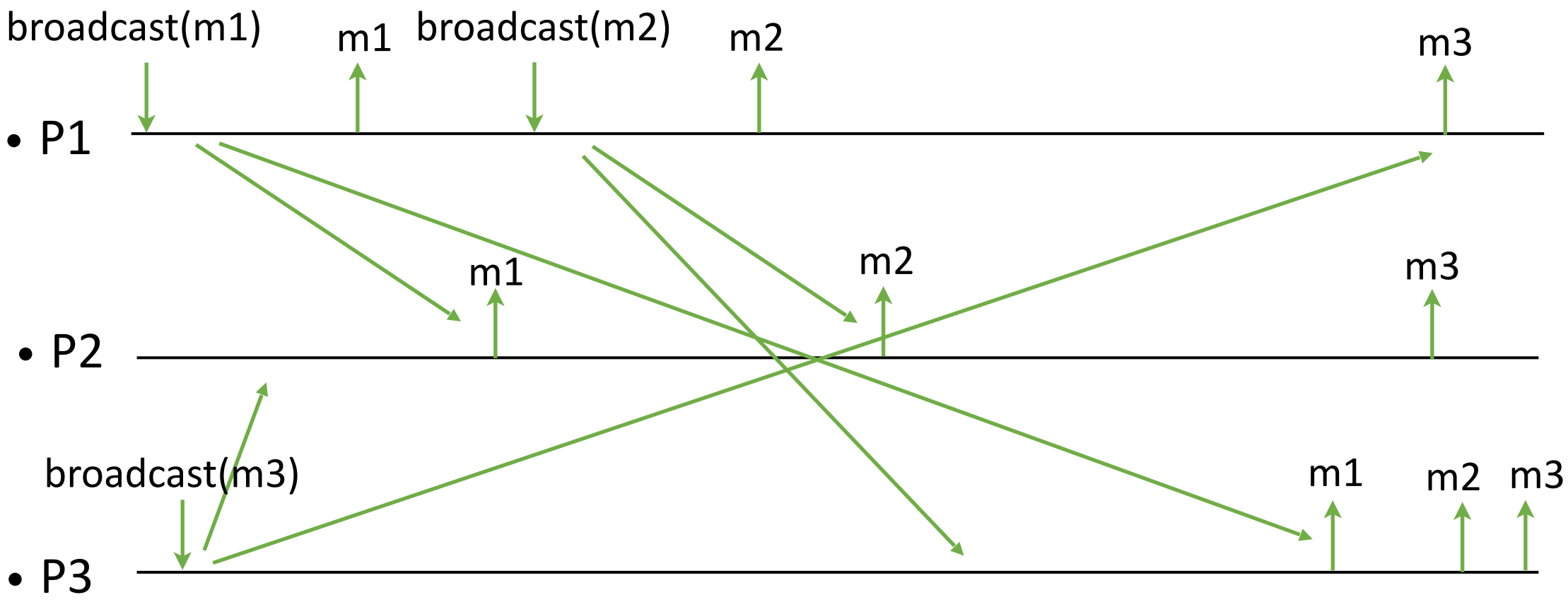
- A replicated service where the replicas need to treat the requests (or transactions) in the **same order** to preserve consistency (state machine replication)

- A notification service where the subscribers need to get notifications in the same order

Intuitions (2)

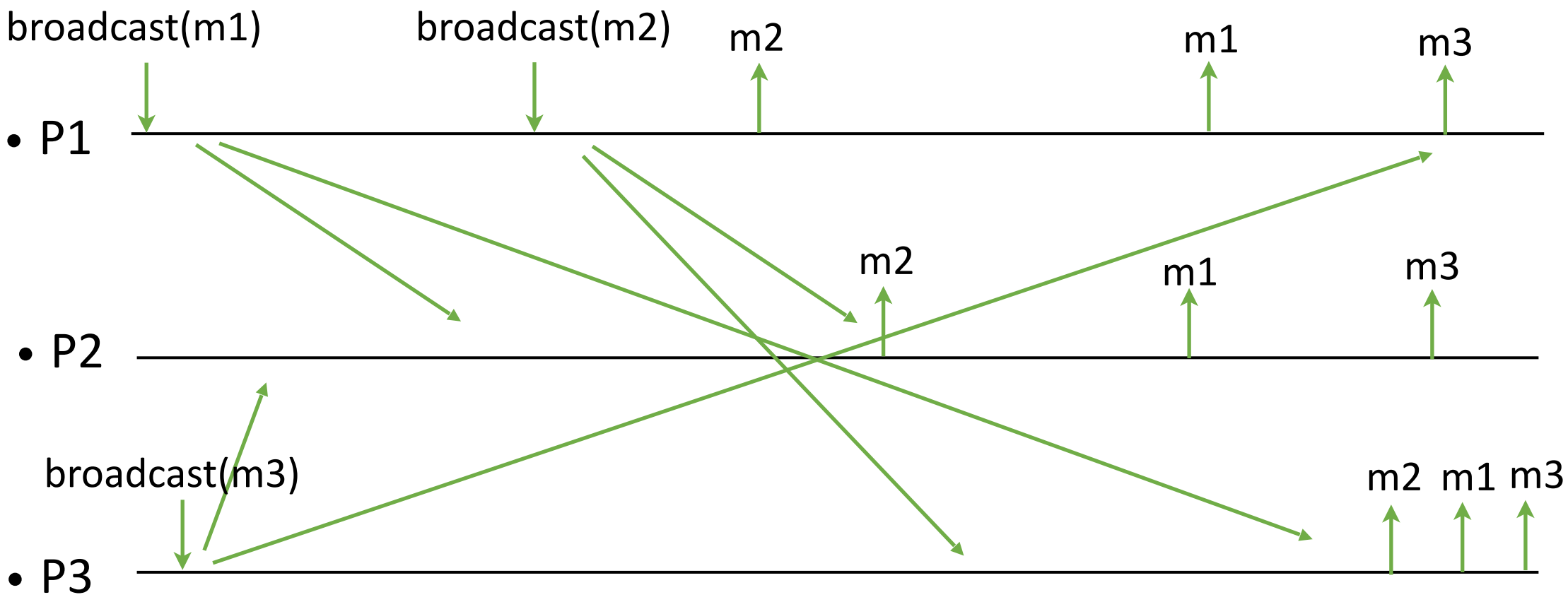
- In **total order** broadcast, the processes must deliver all messages according to the same order (i.e., the order is now total)
- Note that this order does not need to respect causality (or even FIFO ordering)
- Total order broadcast can be made to respect causal (or FIFO) ordering

Total Broadcast (I)



The total order m1, m2, m3.

Total Broadcast (II)



The total order m2, m1, m3.

Overview

- Intuitions: what total order broadcast can bring?
- **Specifications of total order broadcast**
- Consensus-based algorithm

Total order broadcast (tob)

- **Events**
  - Request: <broadcast (m)>
  - Indication: <deliver (src, m)>

- **Properties:**
  - **RB1, RB2, RB3, RB4**
  - **Total order property**

Specification (I)

- **Validity**: If pi and pj are correct, then every message broadcast by pi is eventually delivered by pj.
- **(Uniform) Agreement**: For any message m. If a correct (any) process delivers m, then every correct process delivers m.
- **No duplication**: No message is delivered more than once.
- **No creation**: No message is delivered unless it was broadcast.

Specification (II)

- **(Uniform) Total Order**:
  - Let m and m' be any two messages.
  - Let pi be a correct (any) process that delivers m without having delivered m’ before m.
  - Then there is no correct (any) process that delivers m' before m or only delivers m'.

Specifications

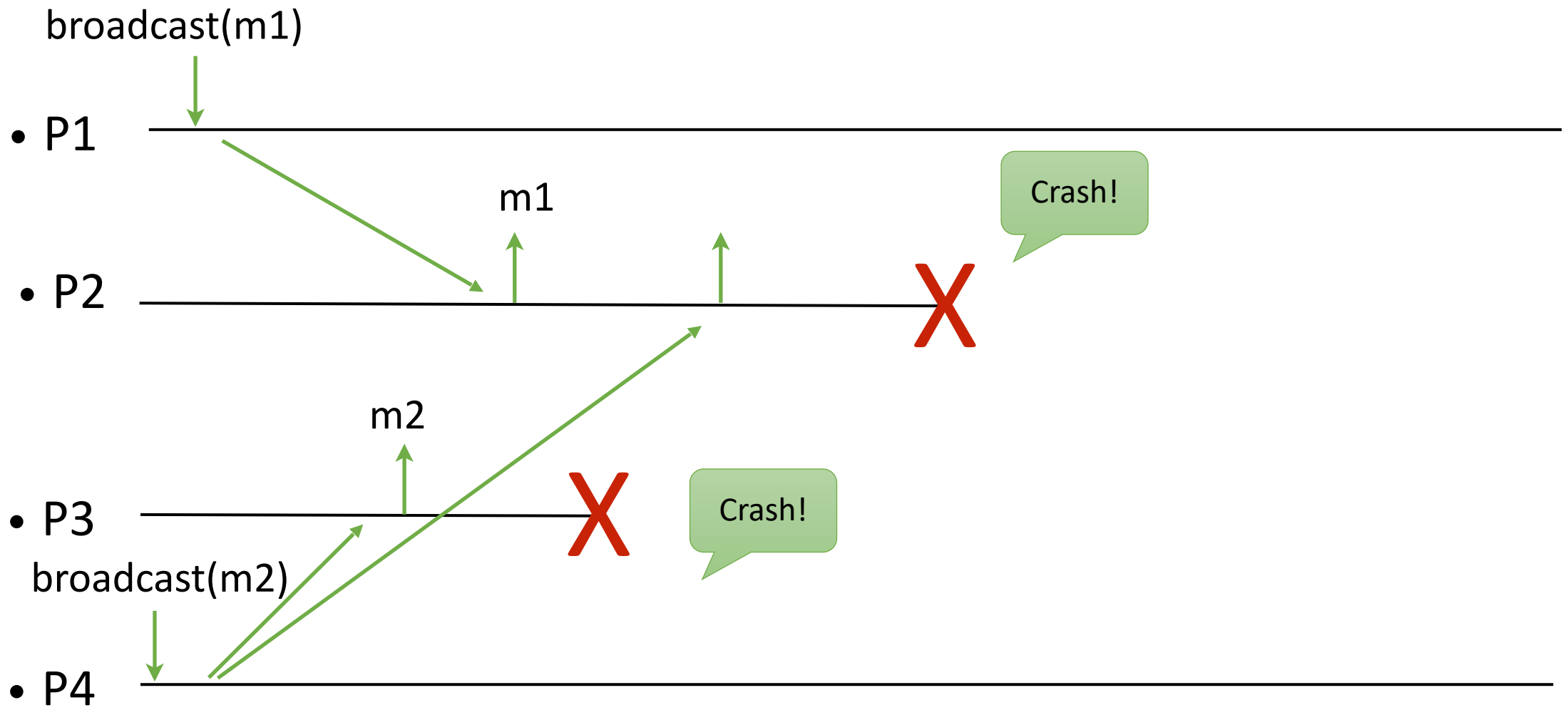
Note the following incorrect statements:

- Let pi and pj be any two correct (any) processes that deliver two messages m and m'. If pi delivers m before m', then pj delivers m before m'.
- Let pi and pj be any two (correct) processes that deliver a message m. If pi delivers a message m' before m, then pj delivers m' before m.

The first definition allows m and m' to be delivered in pi, and m' and not m be delivered in pj.

The second definition allows m to be delivered only in pi, and m’ to be delivered only in pj.

Example

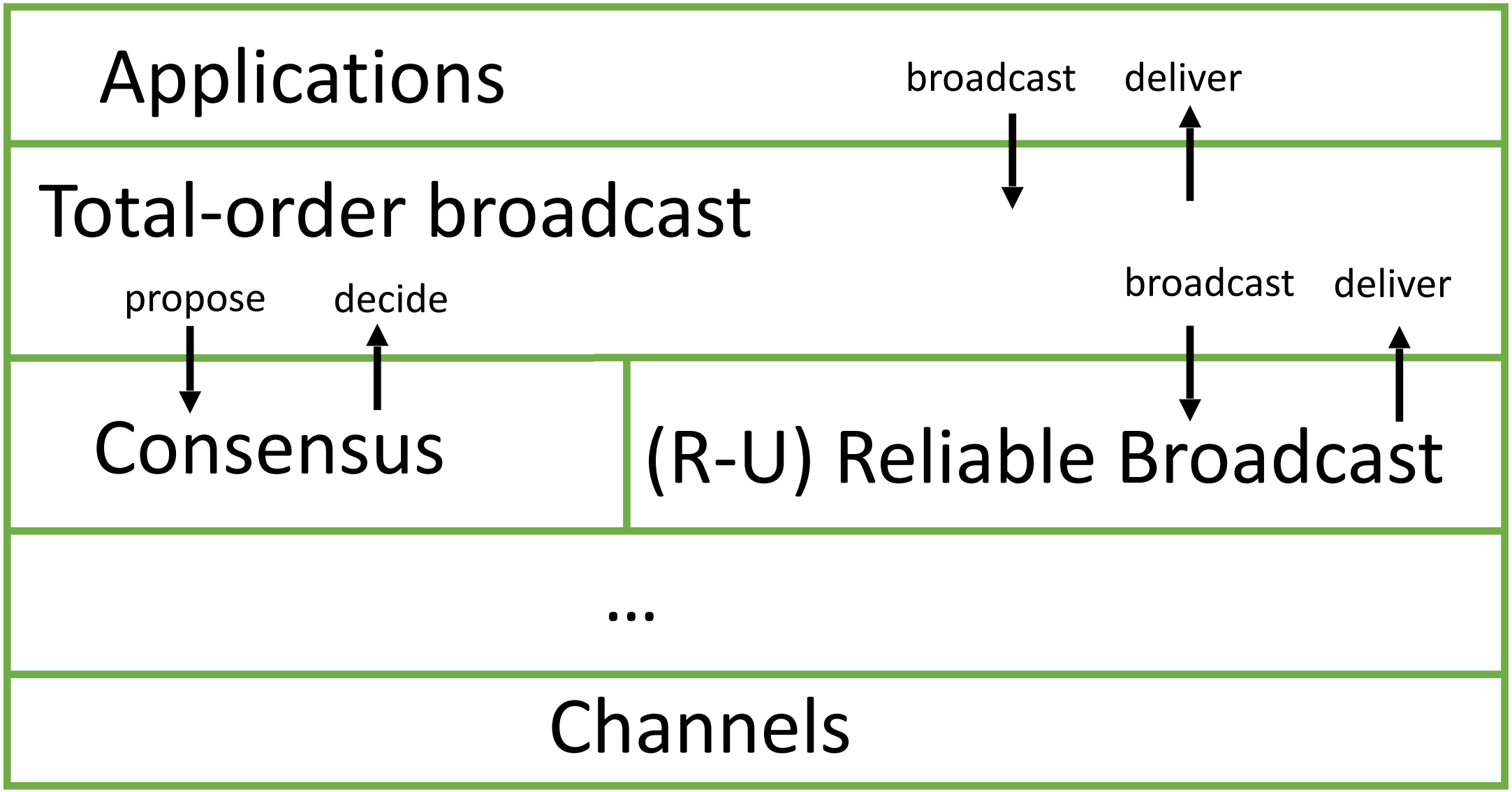


Incorrect execution. The process p2 cannot deliver m2, and the process p3 cannot deliver m1.

Overview

- Intuitions: what total order broadcast can bring?
- Specifications of total order broadcast
- **Consensus-based algorithm**

Modules of a process



Consensus

- In the (uniform) consensus, the processes propose values and need to agree on one among these values.

- **Events**
  - Request: <propose(v)>
  - Indication: <decide(v)>
- **Properties:**
  - **C1, C2, C3, C4**

Uniform Consensus

- **C1. Validity**: Any value decided is a value proposed.
- **C2. Uniform Agreement**: No two correct (any) processes decide differently.
- **C3. Termination**: Every correct process eventually decides.
- **C4. Integrity**: Every process decides at most once.

Total-order broadcast

- Idea: Which message to deliver next is decided by consensus.

- Use rounds of consensus. In each, processes propose their set of messages, and one set is decided, deterministically sorted and delivered.
- To prevent starvation of messages sent by a process, each process first broadcasts its messages.

TOB

Implements: TotalOrder (to)

Uses:

rb: ReliableBroadcast

c: Consensus (cons) a sequence indexed by sn

upon event < Init > **do**

proposals = delivered = ∅

wait := false

sn := 1

wait is used to take consensus rounds in turn.
sn is the sequence number of the current consensus round.

TOB

upon event <broadcast (m)> **do**

    trigger <rb, Broadcast, m>

upon event <rb, deliver (sm, m)> **do**

    if (m ∉ delivered)

        proposals := proposals U {(sm,m)}

upon proposals ≠ ∅ and ¬wait **do**

    wait := true:

    trigger <c[sn], propose(proposals)>

After a process delivers a message in a consensus round, the process may receive it late from the broadcast. Such a message is ignored.

TOB

```
upon event <c[sn], decide(decided)> do
    proposals := proposals \ decided
    ordered := deterministic-sort (decided)
    foreach (sm,m) in ordered:
        trigger <deliver (sm, m)>
        delivered := delivered U {m}
    sn := sn + 1
    wait := false
```

Equivalences

- One can build total order broadcast with consensus and reliable broadcast
- One can build consensus with total order broadcast (deciding the first delivered message)

Therefore, consensus and total order broadcast are equivalent problems in a system with reliable channels

Parts of slides adopted from R. Guerraoui