# Distributed systems

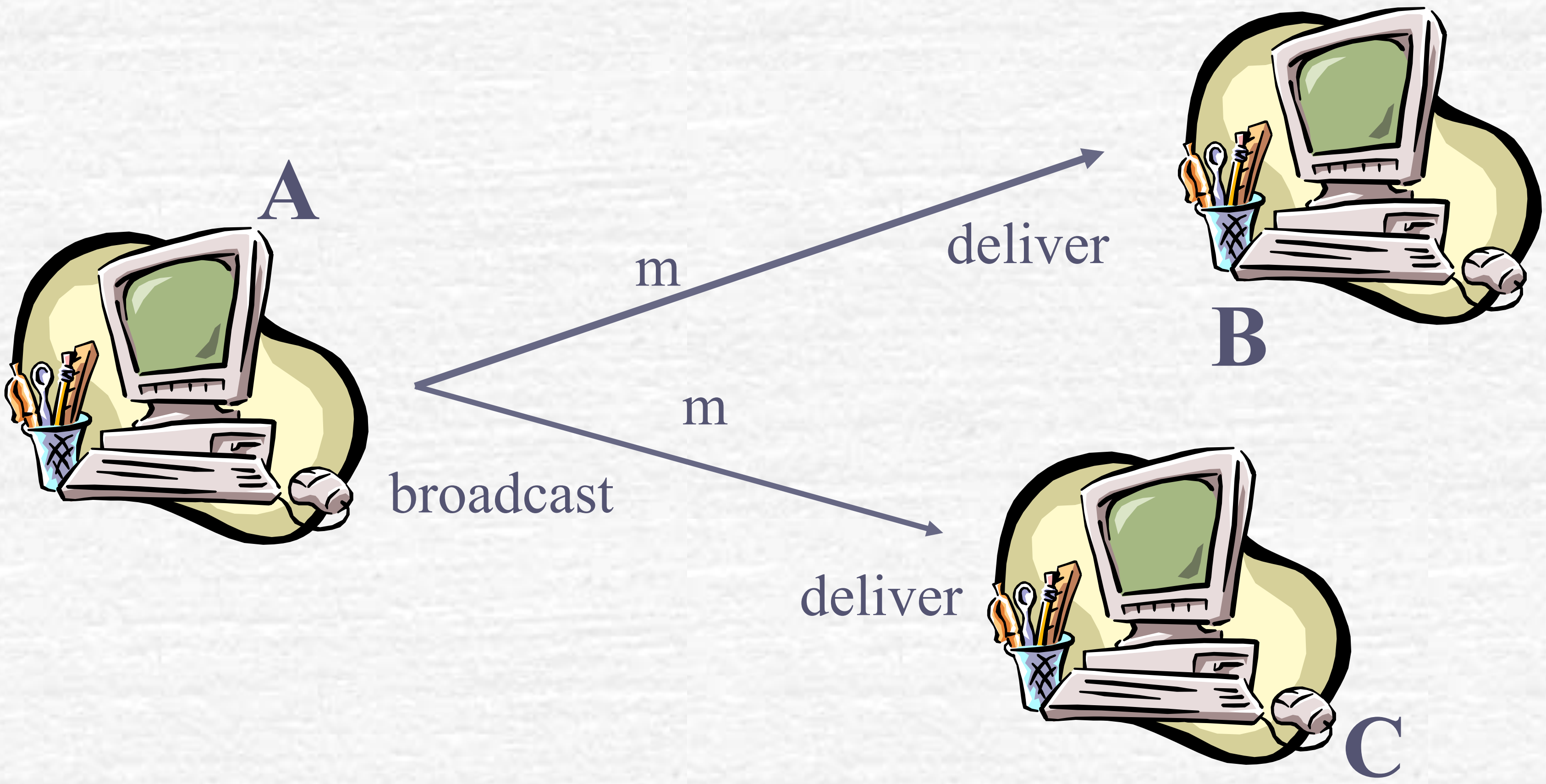
# Total Order Broadcast

Prof R. Guerraoui
Distributed Programming Laboratory

# Overview

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

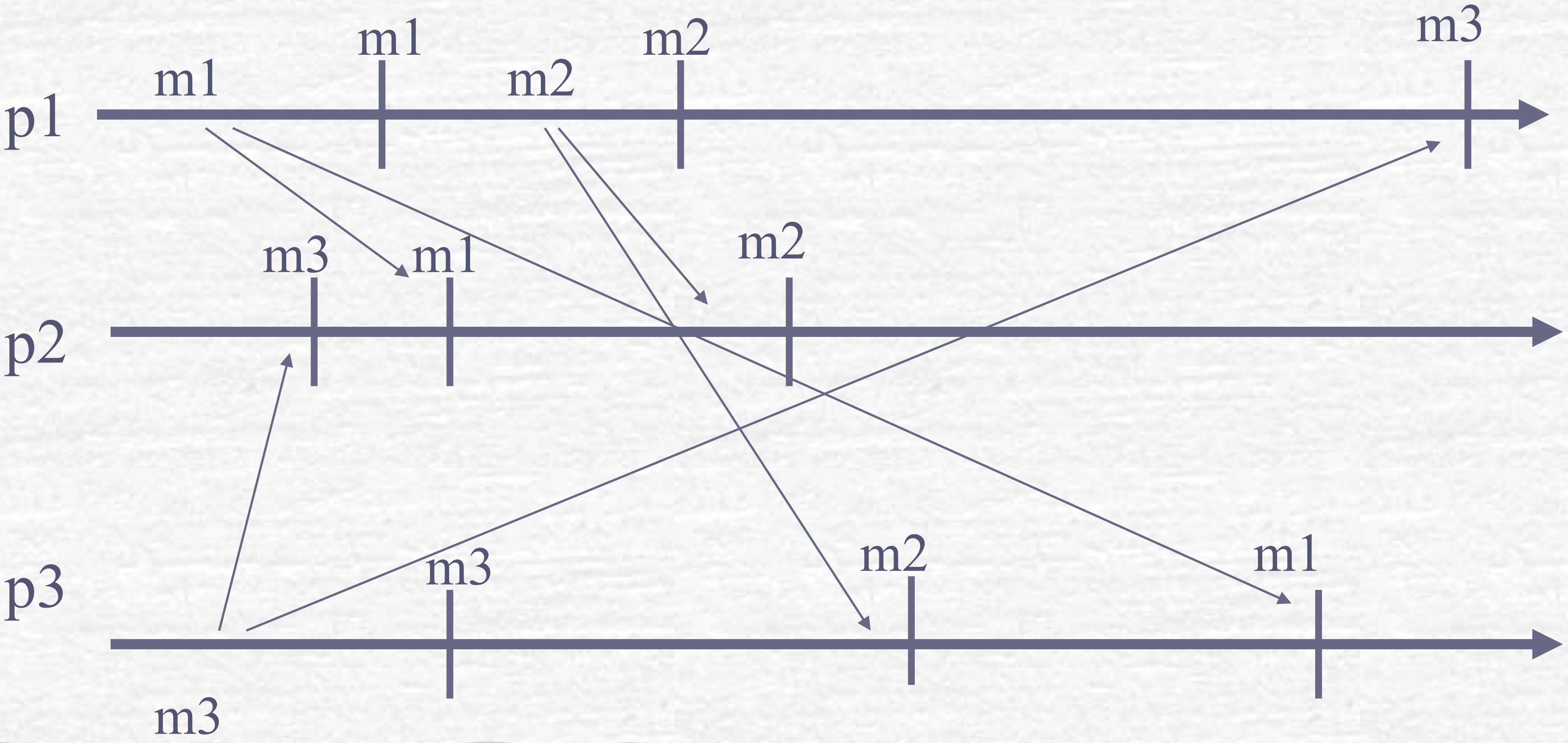
# Broadcast



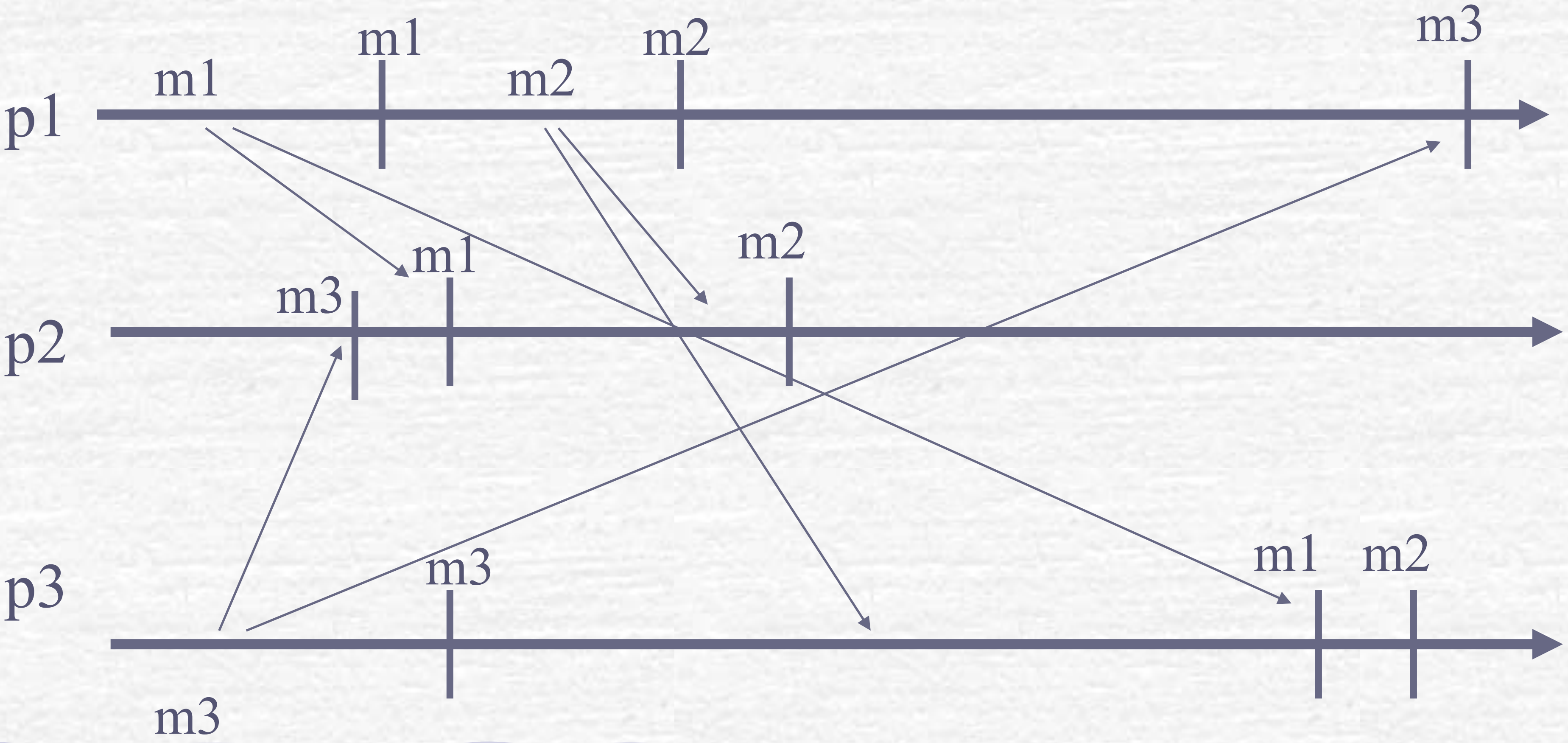
# Intuitions (1)

- 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 order by the processes

# Reliable Broadcast



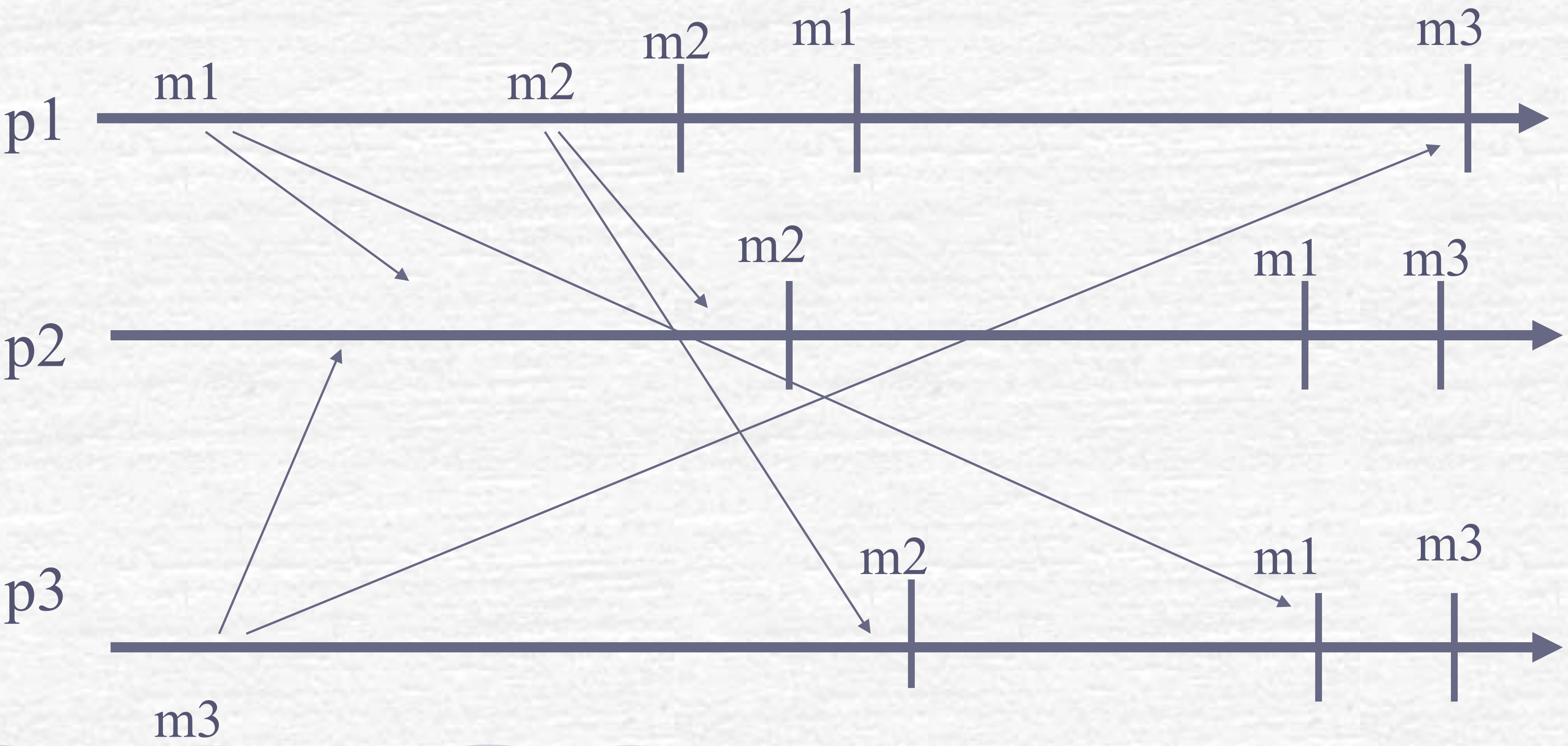
# Causal Broadcast



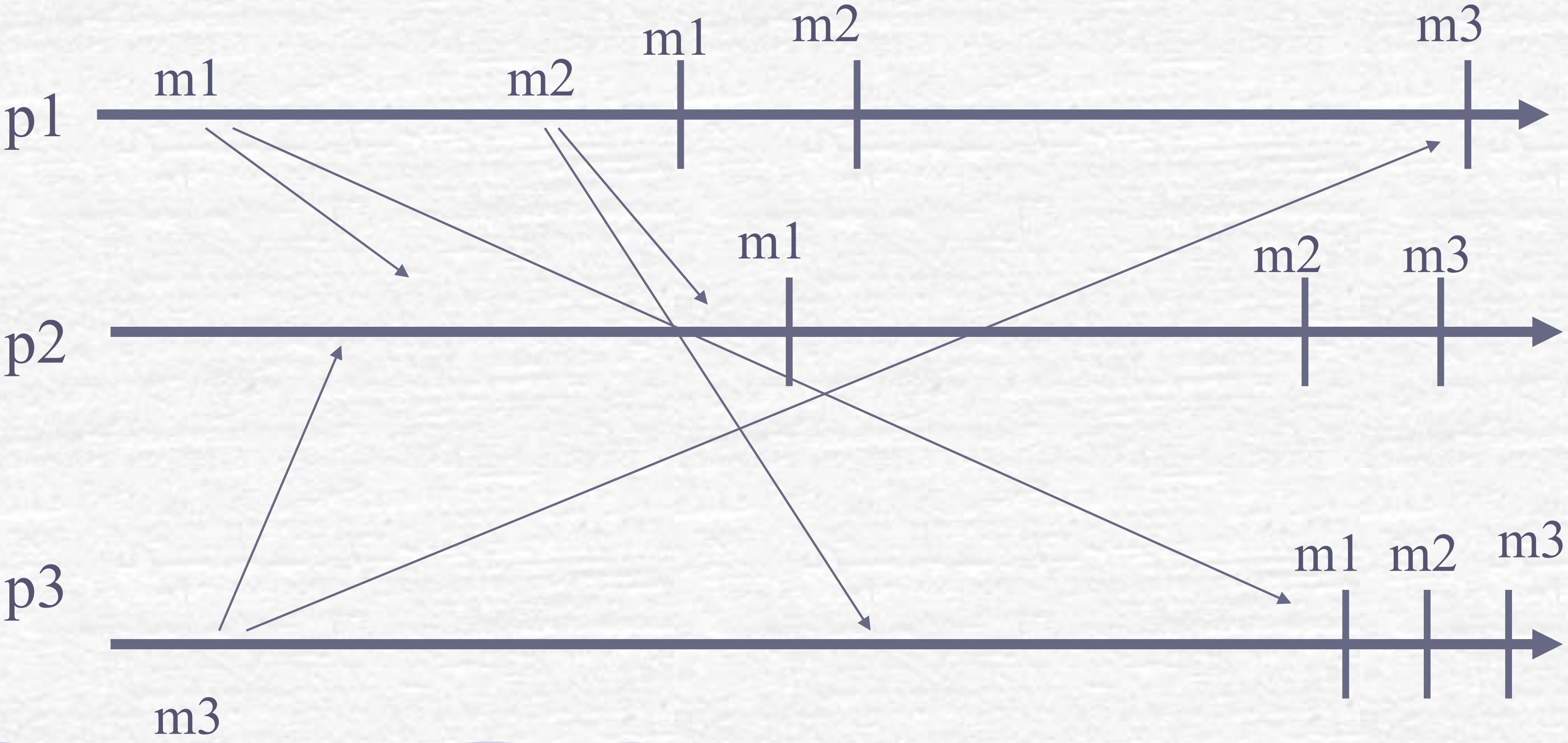
# Intuitions (2)

- 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 Order Broadcast (I)



# Total Order Broadcast (II)



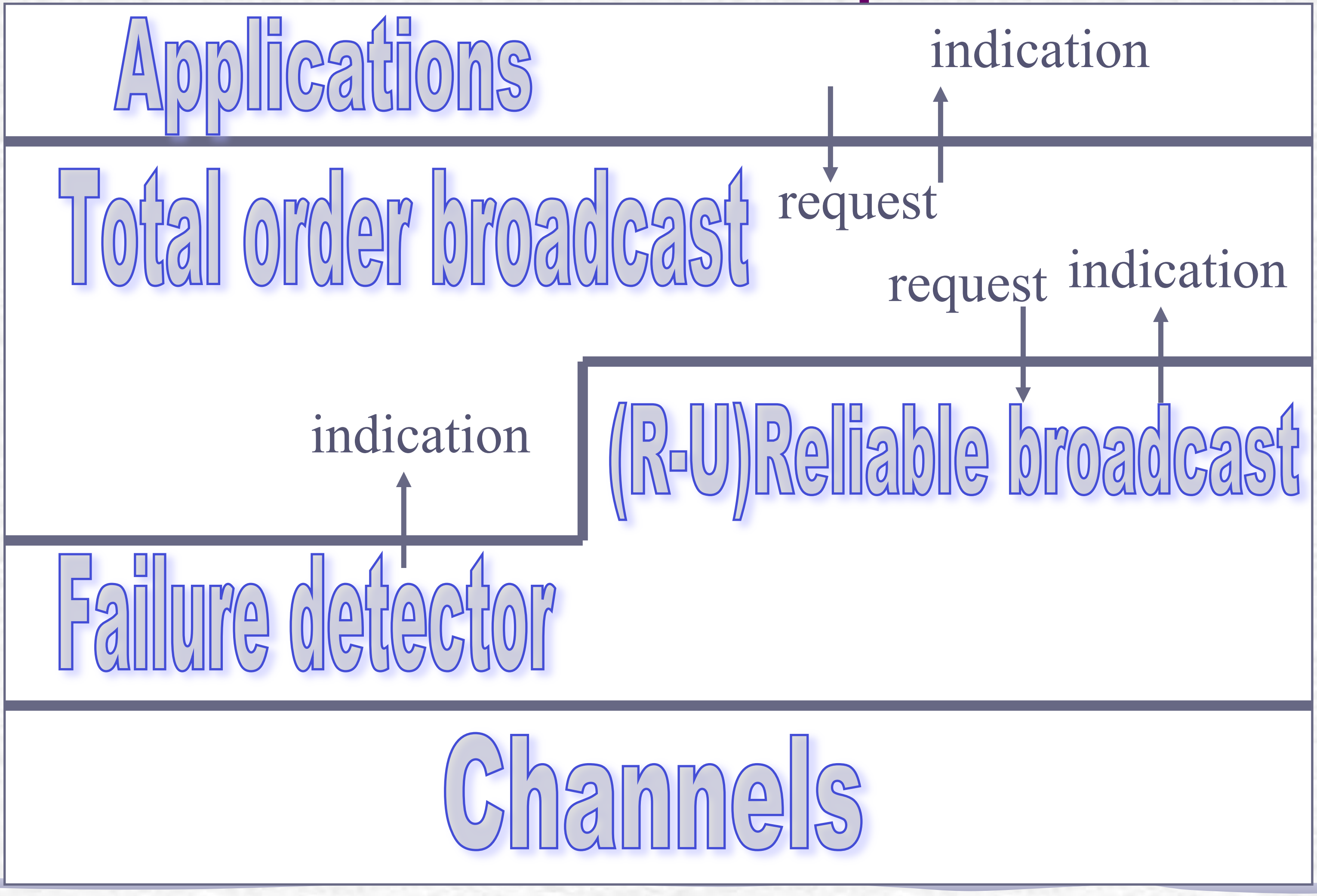
# Application

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

# Application

- A replicated service where the replicas need to treat the requests in the ***same order*** to preserve consistency
- State machine replication

# Modules of a process



# Overview

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

# Total order broadcast (tob)

- ***Events***
  - Request: <toBroadcast, m>
  - Indication: <toDeliver, 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

***No duplication:*** No message is delivered more than once

***No creation:*** No message is delivered unless it was broadcast

***(Uniform) Agreement:*** For any message m. If a correct (any) process delivers m, then every correct process delivers m

# Specification (II)

***(Uniform) Total order***:

Let m and m’ be any two messages.

Let pi be any (correct) process that delivers m without having delivered m’

Then no (correct) process delivers m’ before m

# Specifications

***Note the difference with the following properties:***

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.

# Overview

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

# (Uniform) Consensus

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

***C1. Validity***: Any value decided is a value proposed

***C2. (Uniform) Agreement:*** No two correct (any) processes decide differently

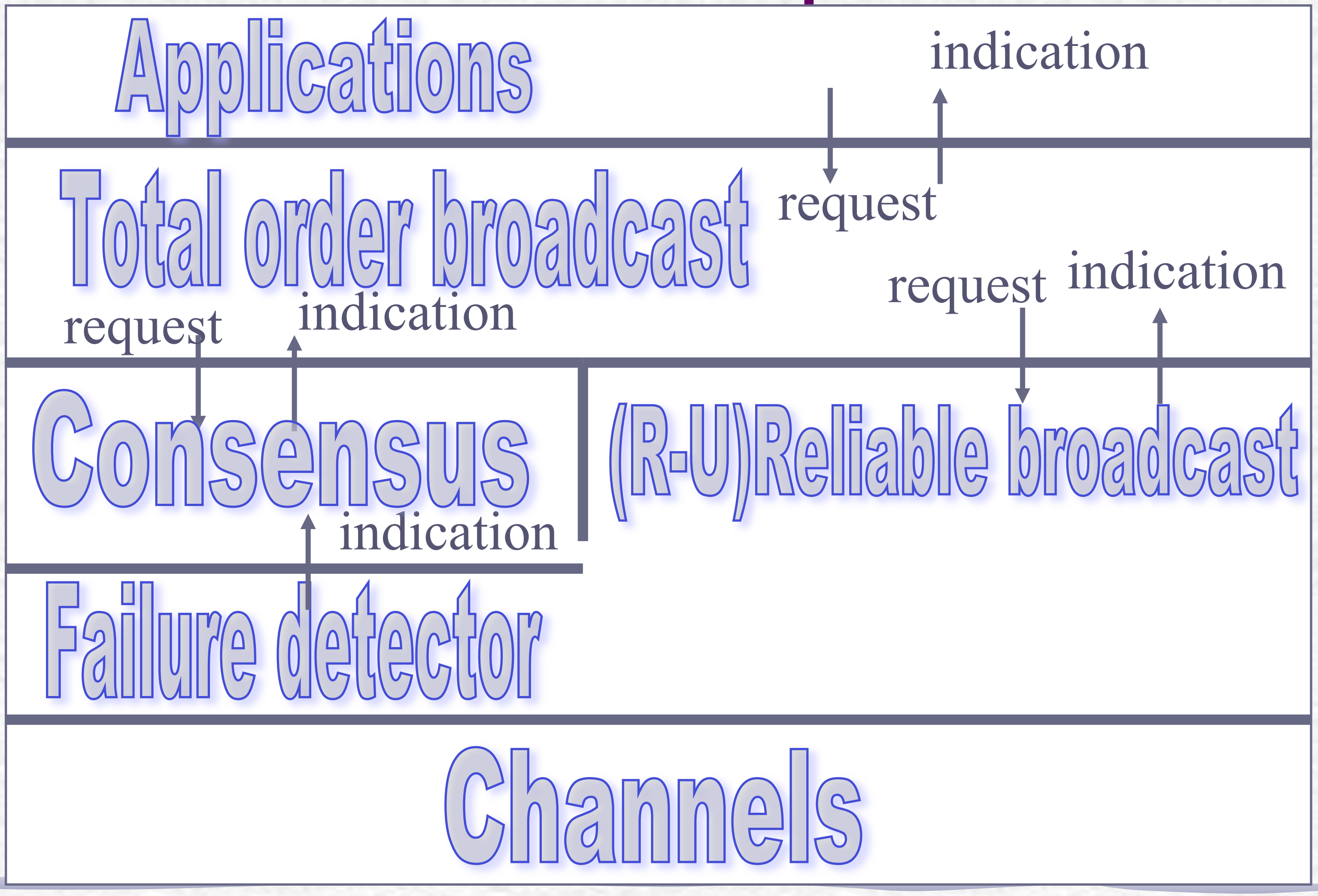
***C3. Termination:*** Every correct process eventually decides

***C4. Integrity***: Every process decides at most once

# Consensus

- ***Events***
  - Request: <Propose, v>
  - Indication: <Decide, v’>
- ***Properties:***
  - ***C1, C2, C3, C4***

# Modules of a process



Applications

indication

request

Total order broadcast

request

indication

request

indication

Consensus

(R-U)Reliable broadcast

indication

Failure detector

Channels

# Algorithm

- **Implements:** TotalOrder (to).
- **Uses:**
    - ReliableBroadcast (rb).
    - Consensus (cons);
- **upon event** < Init > **do**
    - unordered = delivered = empty;
    - wait := false;
    - sn := 1;

# Algorithm (cont’d)

```
upon event < toBroadcast, m> do
    trigger < rbBroadcast, m>;
upon event <rbDeliver,sm,m> and  (m not in delivered) do
    unordered := unordered U {(sm,m)};
upon (unordered not empty) and not(wait) do
    wait := true:
    trigger < Propose, unordered>$_{sn}$;
```

# Algorithm (cont’d)

```
upon event <Decide,decided>$_{sn}$ do
    unordered := unordered \ decided;
    ordered := deterministicSort(decided);
    for all (sm,m) in ordered:
        trigger < toDeliver,sm,m>;
        delivered := delivered U {m};
    sn : = sn + 1;
    wait := false;
```

# Equivalences

1. One can build consensus with total order broadcast
2. One can build total order broadcast with consensus and reliable broadcast

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