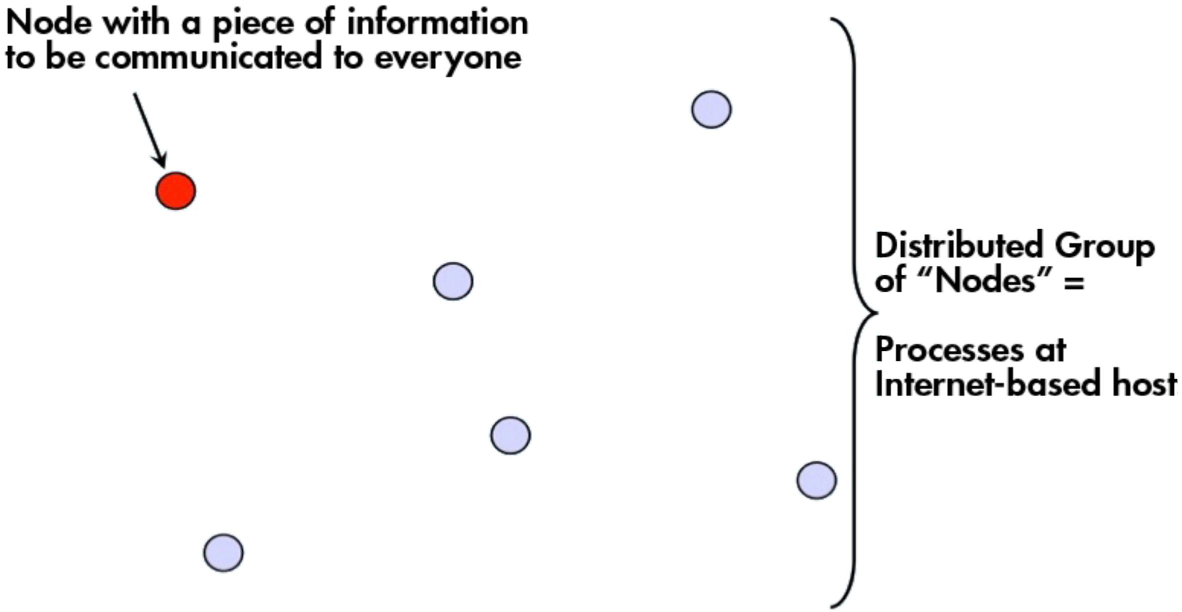
Multicast

ECE 599 / CS 519 – SPRING 2015

Material based on Lectures by Indy Gupta(IG) - All slides © IG

Multicast Problem



Other Communication Forms

- **Multicast** → message sent to a group of processes
- **Broadcast** → message sent to all processes (anywhere)
- **Unicast** → message sent from one sender process to one receiver process

Who Uses Multicast?

- A widely-used abstraction by almost all cloud systems
- Storage systems like Cassandra or a database
  - Replica servers for a key: Writes/reads to the key are multicast within the replica group
  - All servers: membership information (e.g., heartbeats) is multicast across all servers in cluster
- Online scoreboards (ESPN, French Open, FIFA World Cup)
  - Multicast to group of clients interested in the scores
- Stock Exchanges
  - Group is the set of broker computers
  - Groups of computers for High frequency Trading
- Air traffic control system
  - All controllers need to receive the same updates in the same order

Multicast Ordering

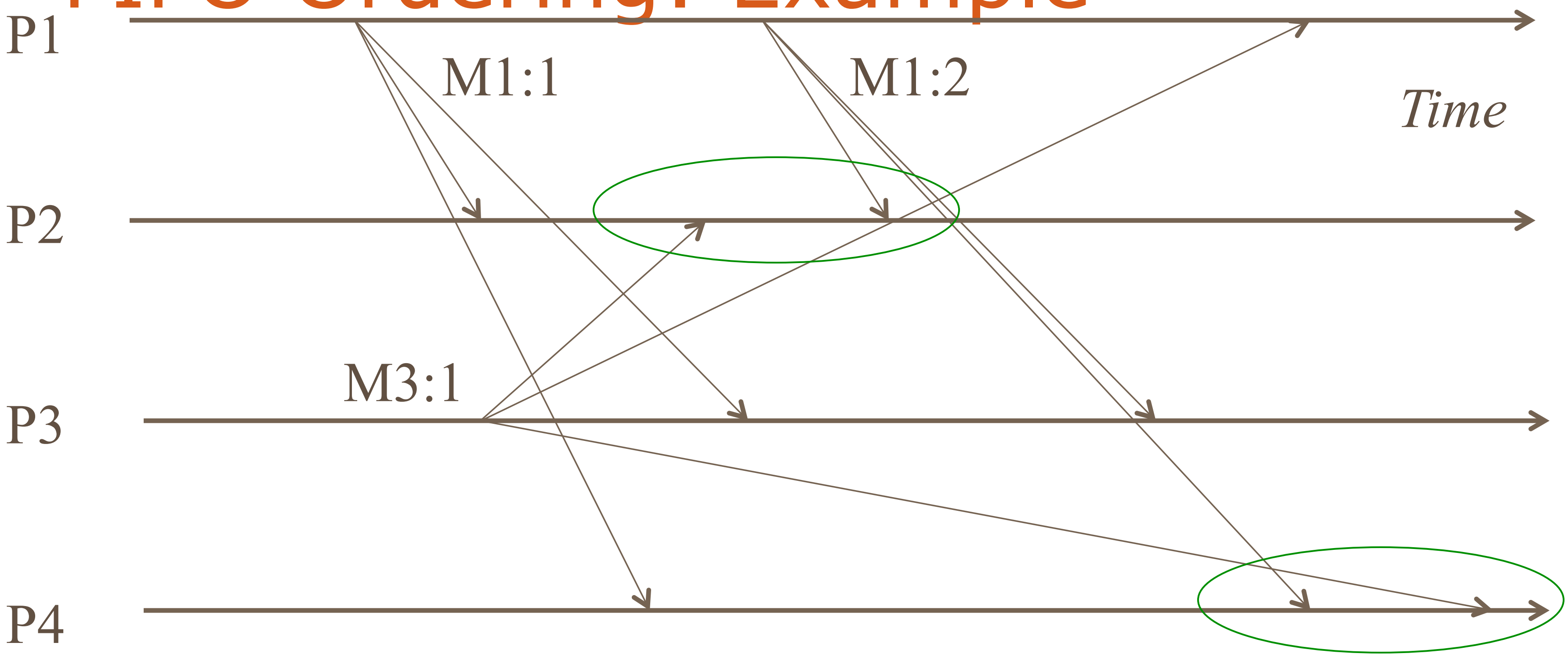
- Determines the meaning of "same order" of multicast delivery at different processes in the group
- Three popular flavors implemented by several multicast protocols
  1. FIFO ordering
  2. Causal ordering
  3. Total ordering

1. FIFO ordering

- Multicasts from each sender are received in the order they are sent, at all receivers
- Don't worry about multicasts from different senders

- More formally
  - *If a correct process issues (sends) multicast(g,m) to group g and then multicast(g,m'), then every correct process that delivers m' would already have delivered m.*

FIFO Ordering: Example



M1:1 and M1:2 should be received in that order at each receiver

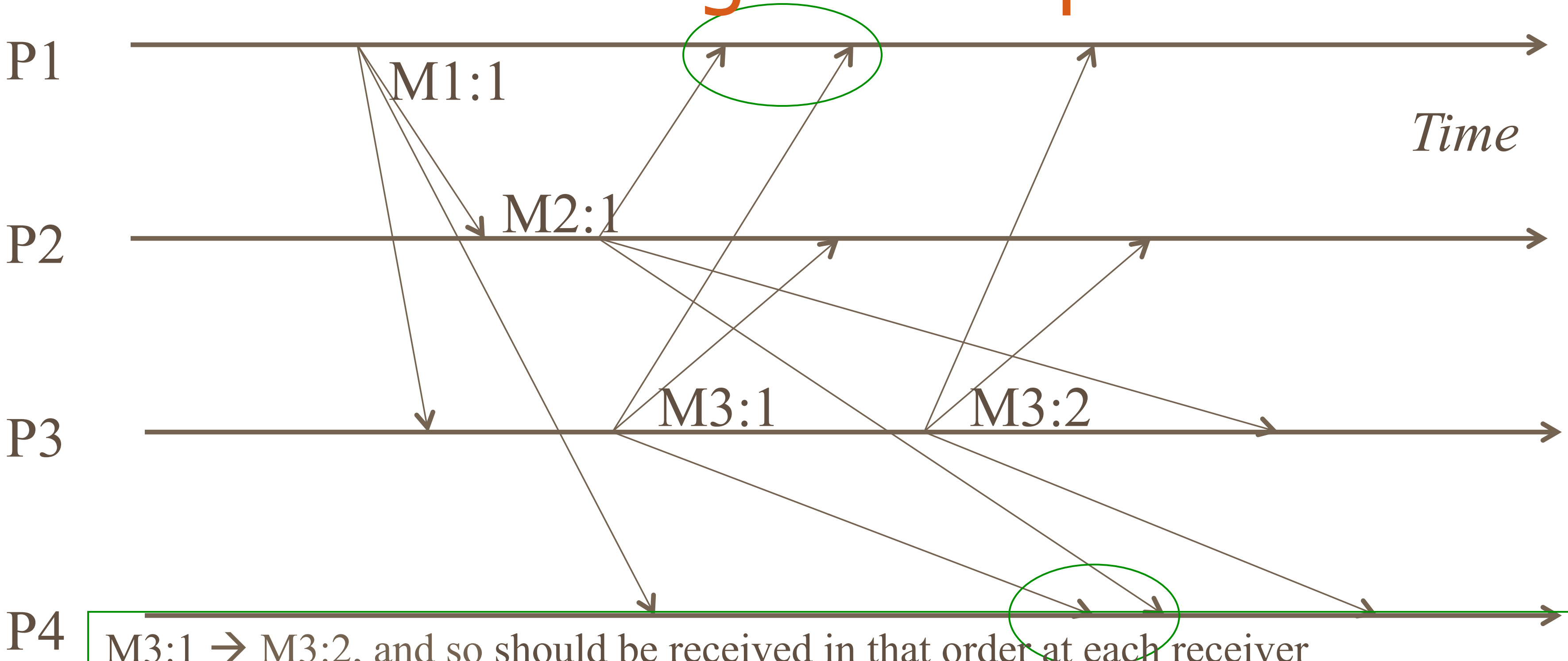
Order of delivery of M3:1 and M1:2 could be different at different receivers

2. Causal Ordering

- Multicasts whose send events are causally related, must be received in the same causality-obeying order at all receivers

- Formally
  - *If multicast(g,m) → multicast(g,m')   then any correct process that delivers   m' would already have delivered m.*
  - *(→ is Lamport's happens-before)*

Causal Ordering: Example



M3:1 → M3:2, and so should be received in that order at each receiver

M1:1 → M3:1, and so should be received in that order at each receiver

M3:1 and M2:1 are concurrent and thus ok to be received in different orders at different receivers

Causal vs. FIFO

- Causal Ordering => FIFO Ordering
- Why?
  - If two multicasts M and M’ are sent by the same process P, and M was sent before M’, then M → M’
  - Then a multicast protocol that implements causal ordering will obey FIFO ordering since M → M’
- Reverse is not true! FIFO ordering does not imply causal ordering.

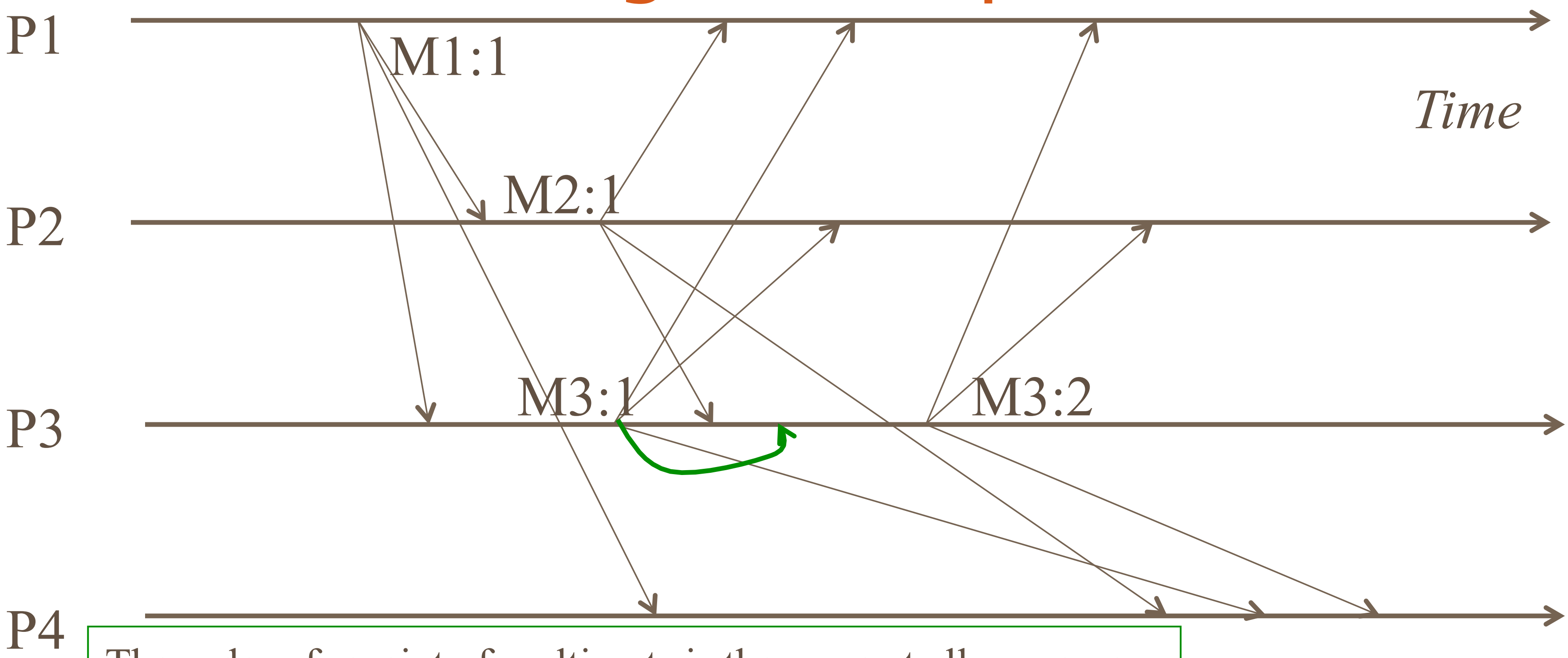
Why Causal at All?

- Group = set of your friends on a social network
- A friend sees your message m, and she posts a response (comment) m' to it
  - If friends receive m' before m, it wouldn't make sense
  - But if two friends post messages m" and n" concurrently, then they can be seen in any order at receivers
- A variety of systems implement causal ordering: Social networks, bulletin boards, comments on websites, etc.

3. Total Ordering

- Also known as "Atomic Broadcast"
- Unlike FIFO and causal, this does not pay attention to order of multicast sending
- Ensures all receivers receive all multicasts in the same order
- Formally
  - *If a correct process P delivers message m before m' (independent of the senders), then any other correct process P' that delivers m' would already have delivered m.*

Total Ordering: Example



The order of receipt of multicasts is the same at all processes.

M1:1, then M2:1, then M3:1, then M3:2

May need to delay delivery of some messages

Hybrid Variants

- Since FIFO/Causal are orthogonal to Total, can have hybrid ordering protocols too
  - FIFO-total hybrid protocol satisfies both FIFO and total orders
  - Causal-total hybrid protocol satisfies both Causal and total orders

Implementation?

- That was *what* ordering is
- But *how* do we implement each of these orderings?

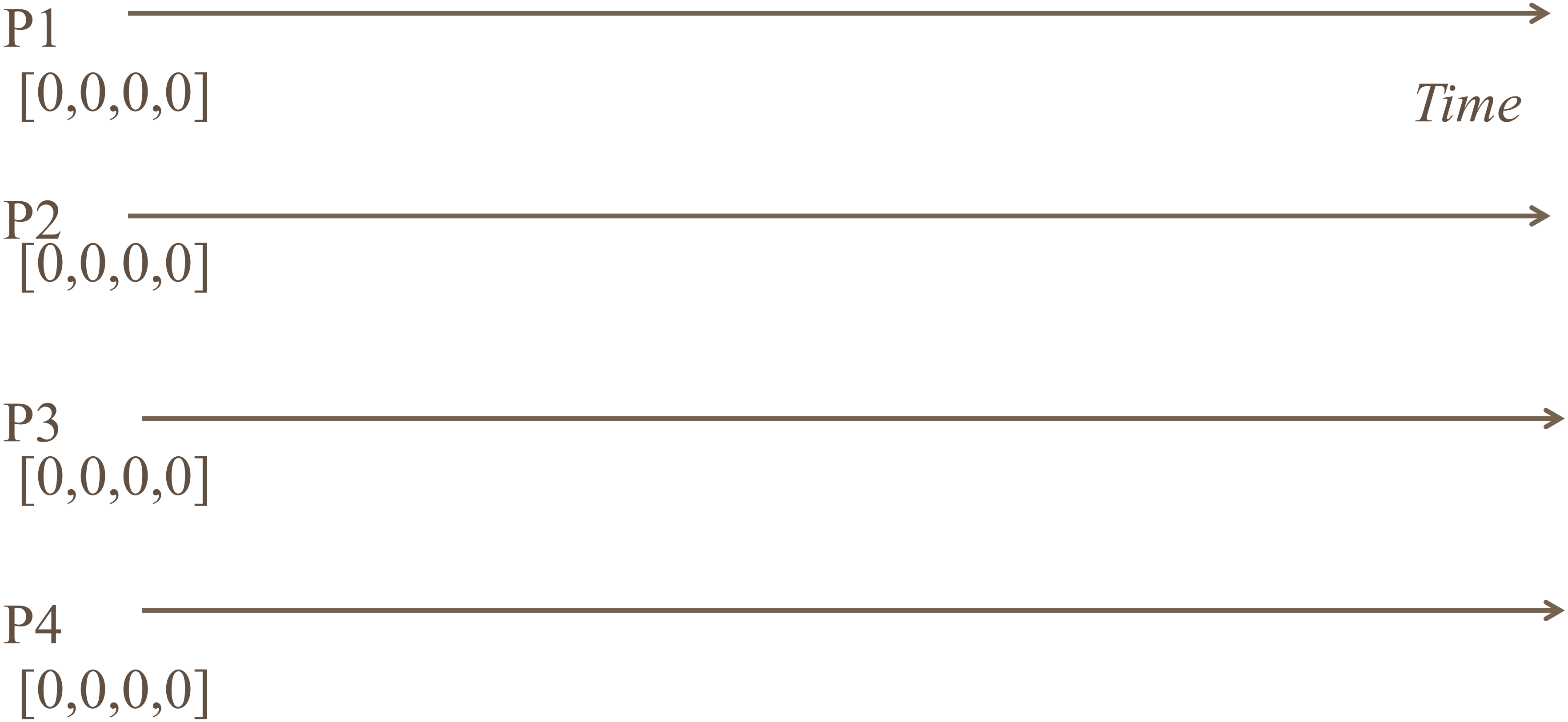
FIFO Multicast: Data Structures

- Each receiver maintains a per-sender sequence number (integers)
  - Processes P*1* through P*N*
  - P*i* maintains a vector of sequence numbers P*i*[1…N] (initially all zeroes)
  - P*i*[*j*] is the latest sequence number P*i* has received from P*j*

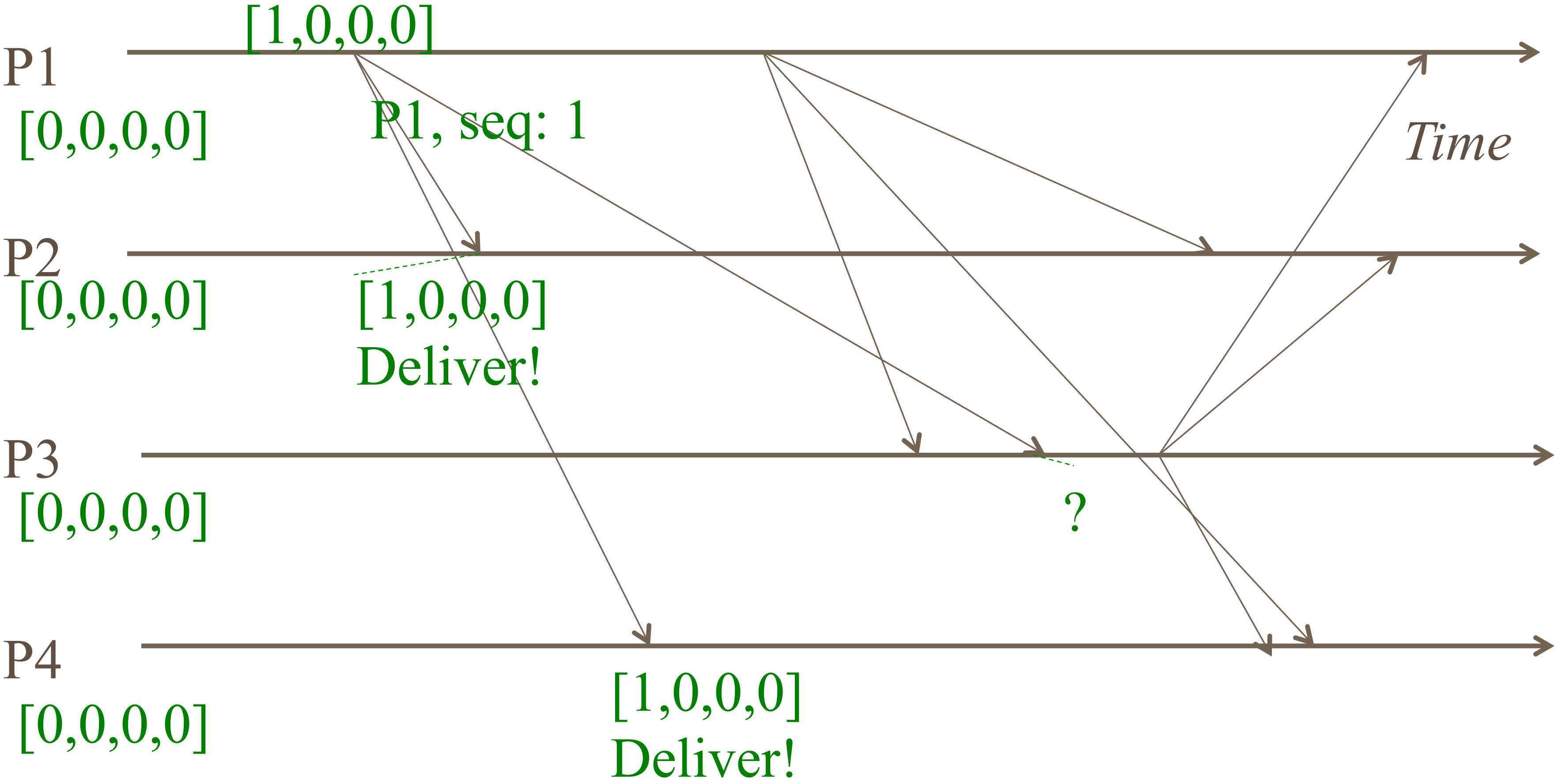
FIFO Multicast: Updating Rules

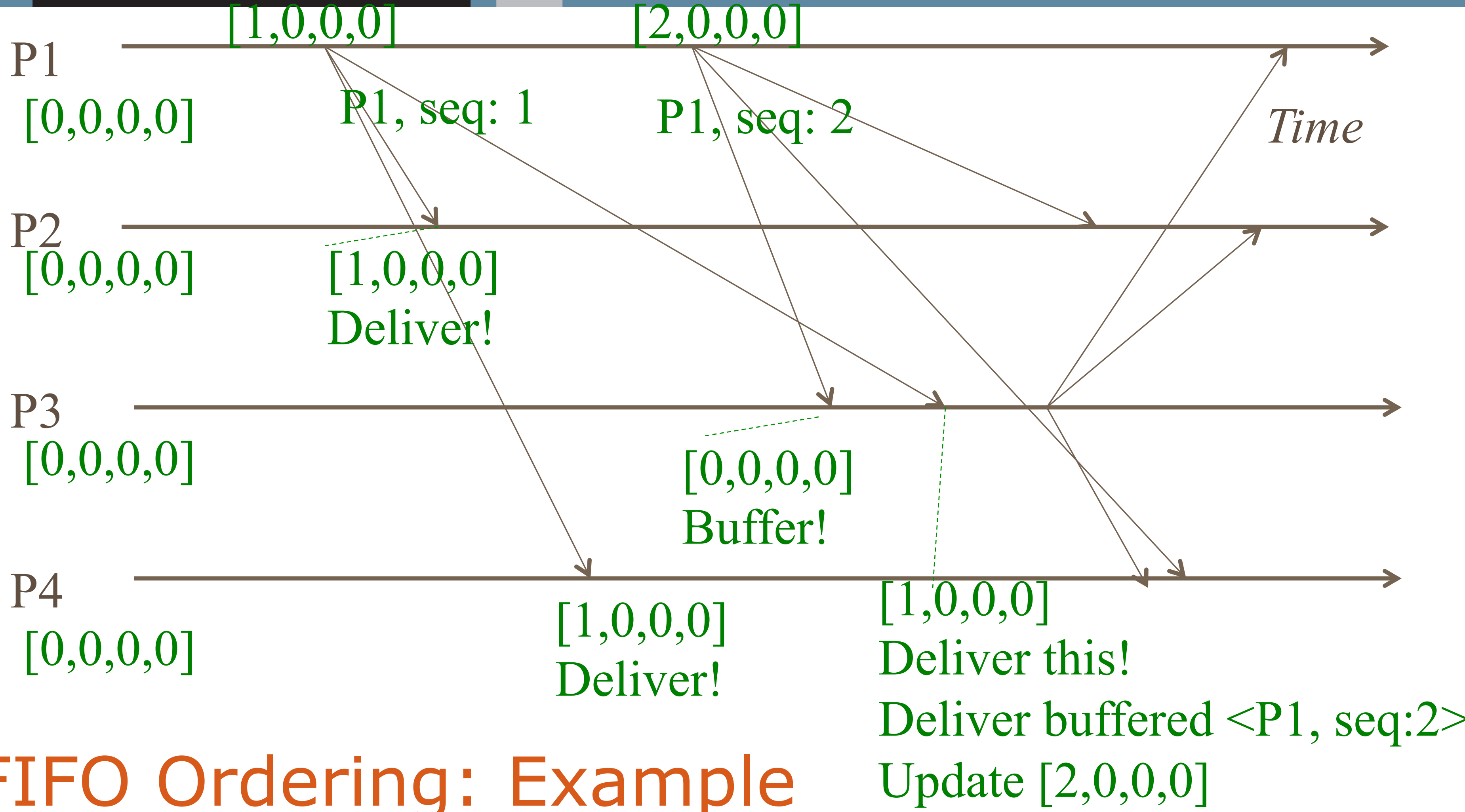
- Send multicast at process P*j*:
  - Set P*j*[*j*] = P*j*[*j*] + 1
  - Include new P*j*[*j*] in multicast message as its sequence number
- Receive multicast: If P*i* receives a multicast from P*j* with sequence number *S* in message
  - if (S == P*i*[*j*] + 1) then
    - deliver message to application
    - Set P*i*[*j*] = P*i*[*j*] + 1
  - else buffer this multicast until above condition is true

FIFO Ordering: Example



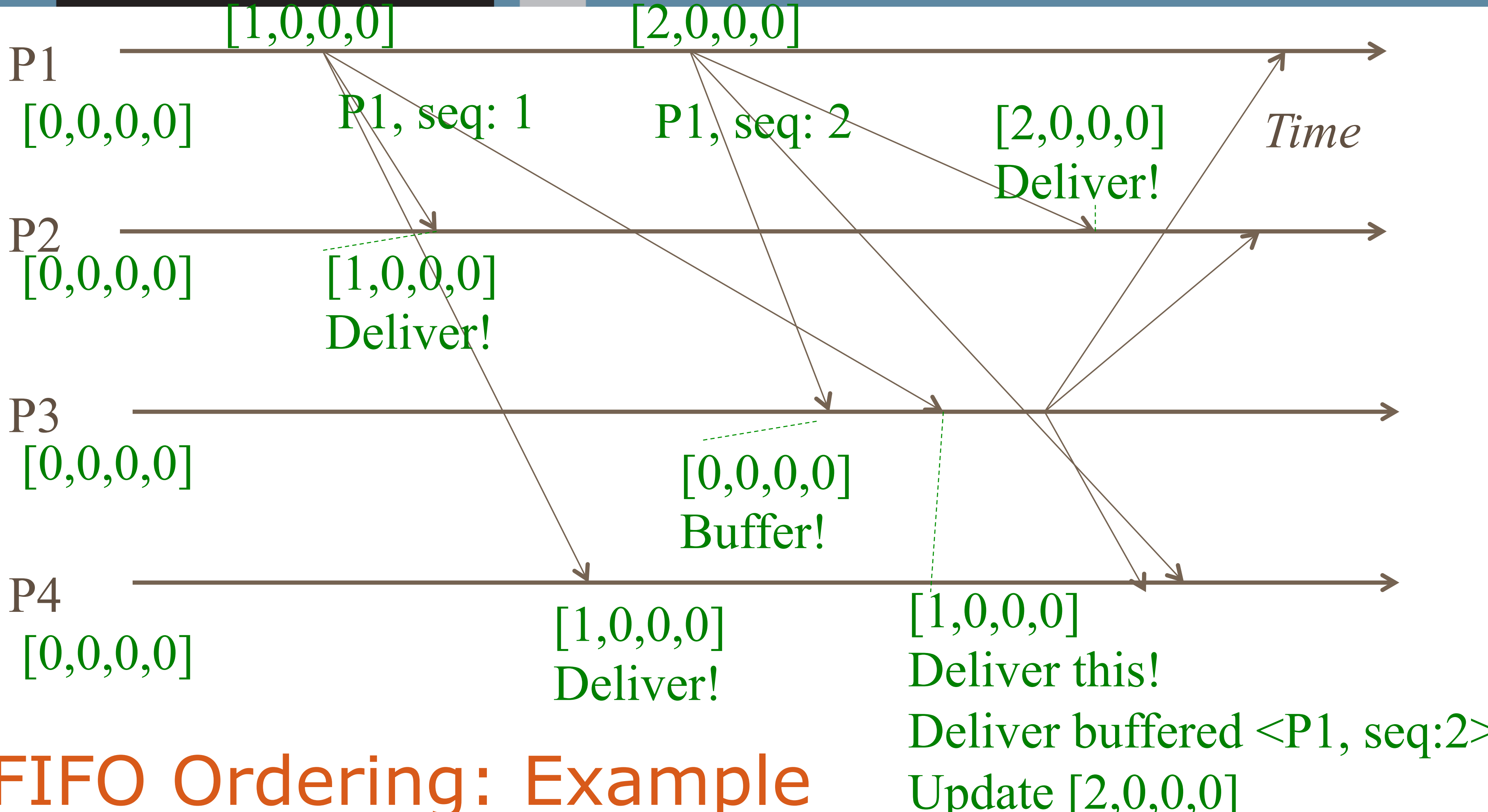
FIFO Ordering: Example

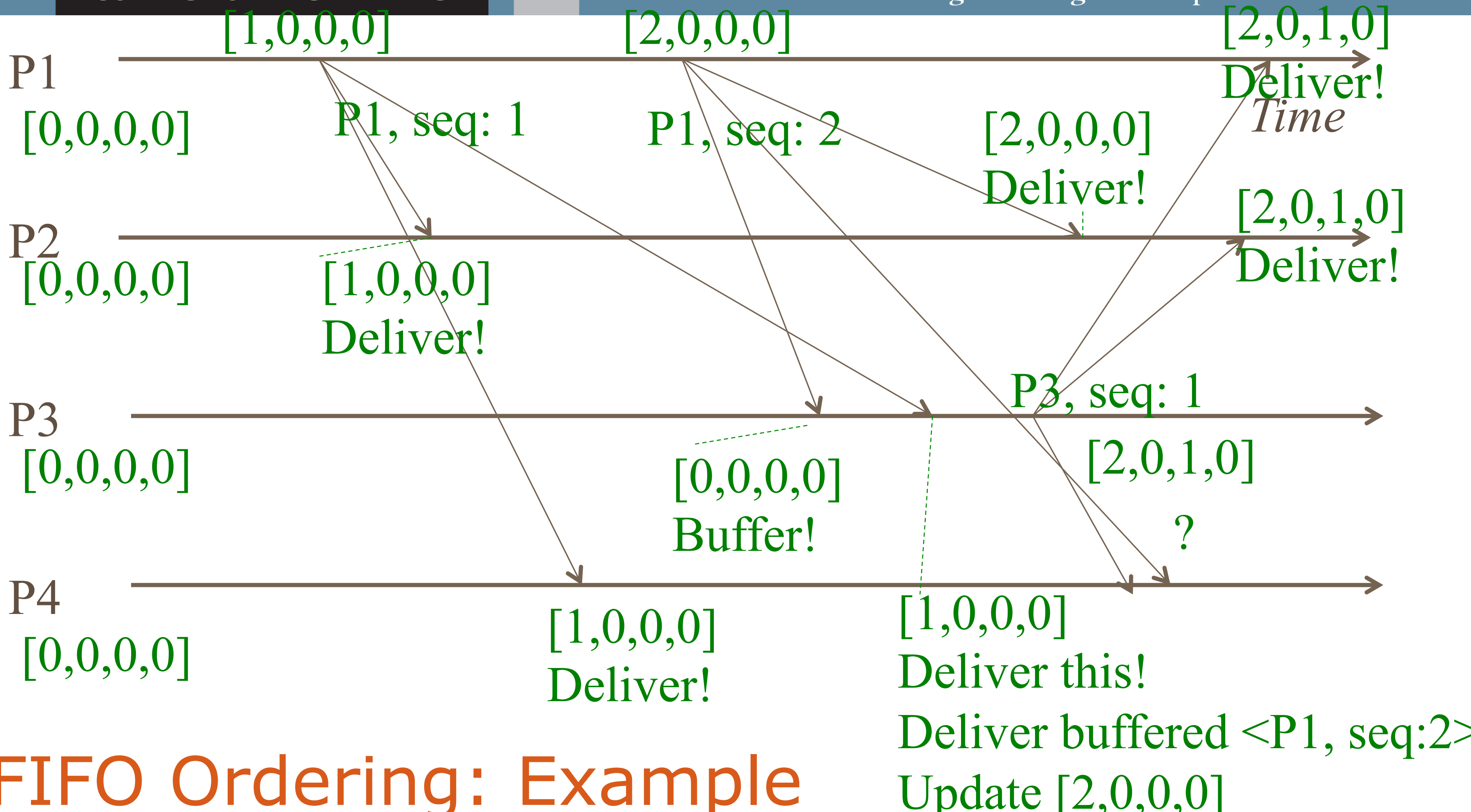




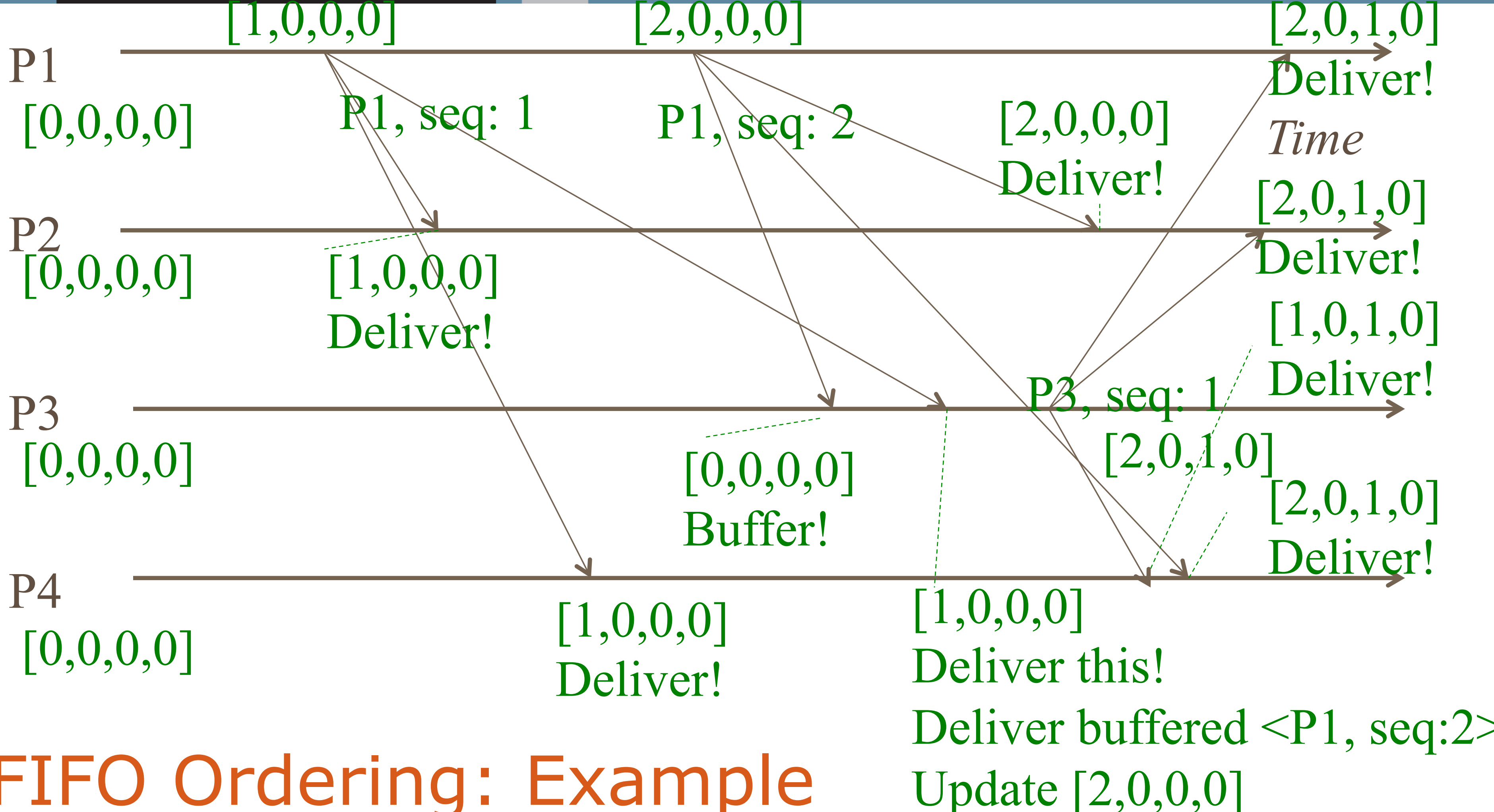
FIFO Ordering: Example

FIFO Ordering: Example





FIFO Ordering: Example



FIFO Ordering: Example

Total Ordering

- Ensures all receivers receive all multicasts in the same order

- Formally
  - *If a correct process P delivers message m before m' (independent of the senders), then any other correct process P' that delivers m' would already have delivered m.*

Sequencer-based Approach

- Special process elected as leader or sequencer
- Send multicast at process P*i*:
  - Send multicast message M to group and sequencer
- Sequencer:
  - Maintains a global sequence number S (initially 0)
  - When it receives a multicast message M, it sets S = S + 1, and multicasts <M, S>

Sequencer-based Approach (2)

- Receive multicast at process P*i*:
  - P*i* maintains a local received global sequence number S*i* (initially 0)
  - If P*i* receives a multicast M from P*j*, it buffers it until it both
    1. P*i* receives <M, S(M)> from sequencer, and
    2. S*i* + 1 = S(M)
    - Then deliver it message to application and set S*i* = S*i* + 1

Causal Ordering

- Multicasts whose send events are causally related, must be received in the same causality-obeying order at all receivers
- Formally
  - *If multicast(g,m) → multicast(g,m') then any correct process that delivers m' would already have delivered m.*
  - *(→ is Lamport's happens-before)*

Causal Multicast: Data structures

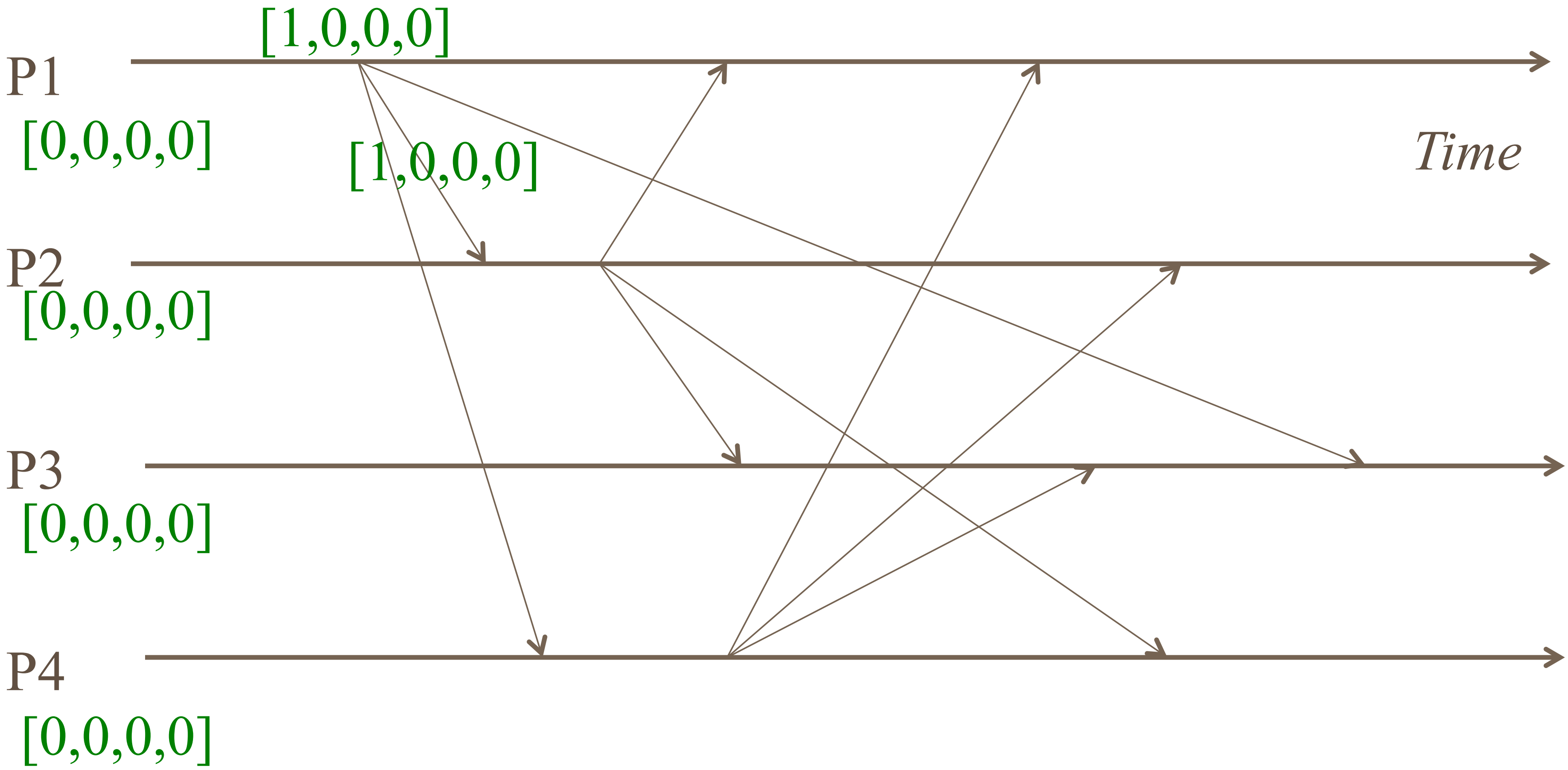
- Each receiver maintains a vector of per-sender sequence numbers (integers)
  - Similar to FIFO Multicast, but updating rules are different
  - Processes P*1* through P*N*
  - P*i* maintains a vector P*i*[1…N] (initially all zeroes)
  - P*i*[*j*] is the latest sequence number P*i* has received from P*j*

Causal Multicast: Updating Rules

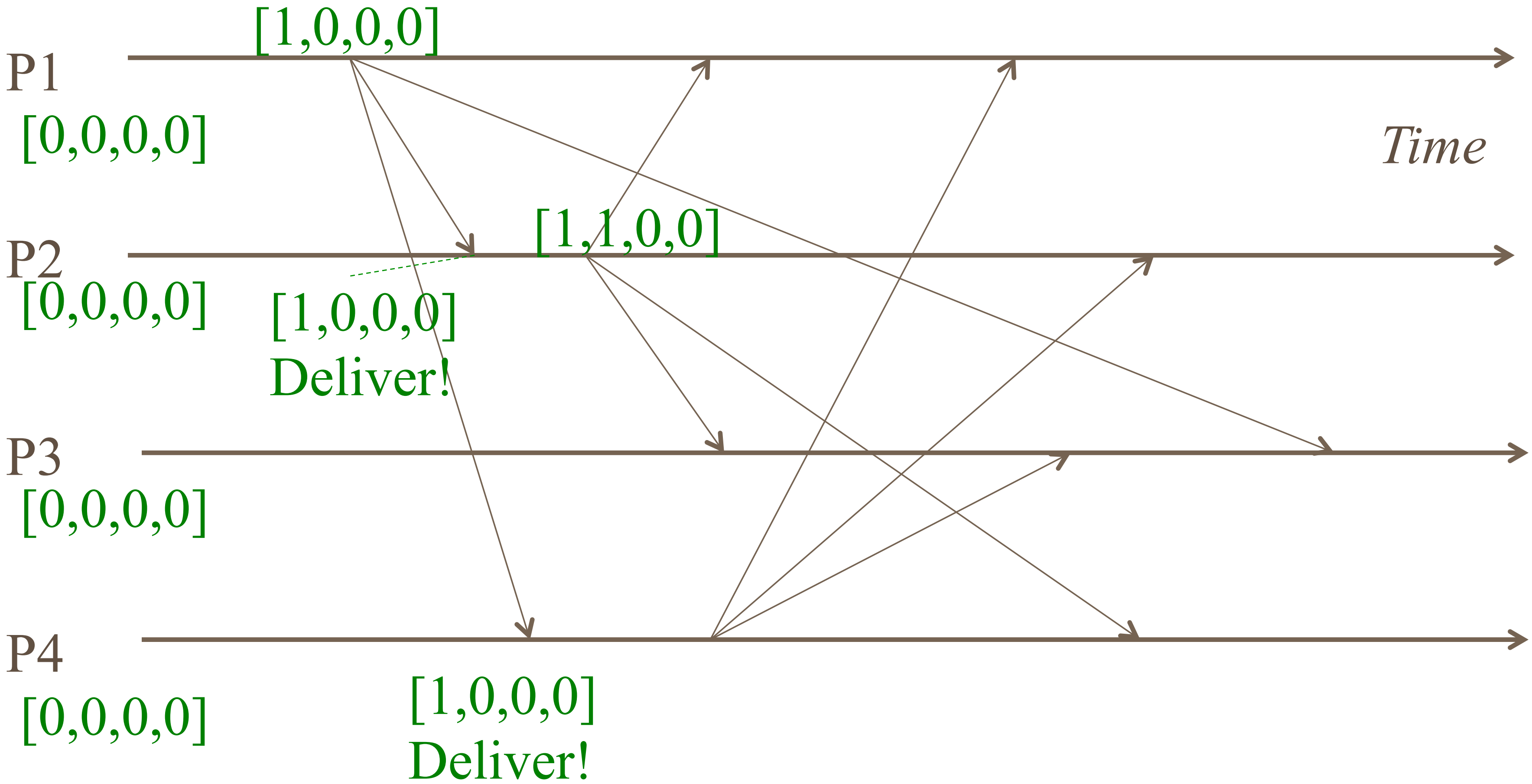
- Send multicast at process Pj:
  - Set Pj[j] = Pj[j] + 1
  - Include new entire vector Pj[1…N] in multicast message as its sequence number
- Receive multicast: If Pi receives a multicast from Pj with vector M[1…N] (= Pj[1…N]) in message, buffer it until both:

1. This message is the next one Pi is expecting from Pj, i.e.,
   - M[j] = Pi[j] + 1
2. All multicasts, anywhere in the group, which happened-before M have been received at Pi, i.e.,
   - For all $k \neq j$: M[k] $\leq$ Pi[k]
   - i.e., ***Receiver satisfies causality***
3. When above two conditions satisfied, deliver M to application and set Pi[j] = M[j]

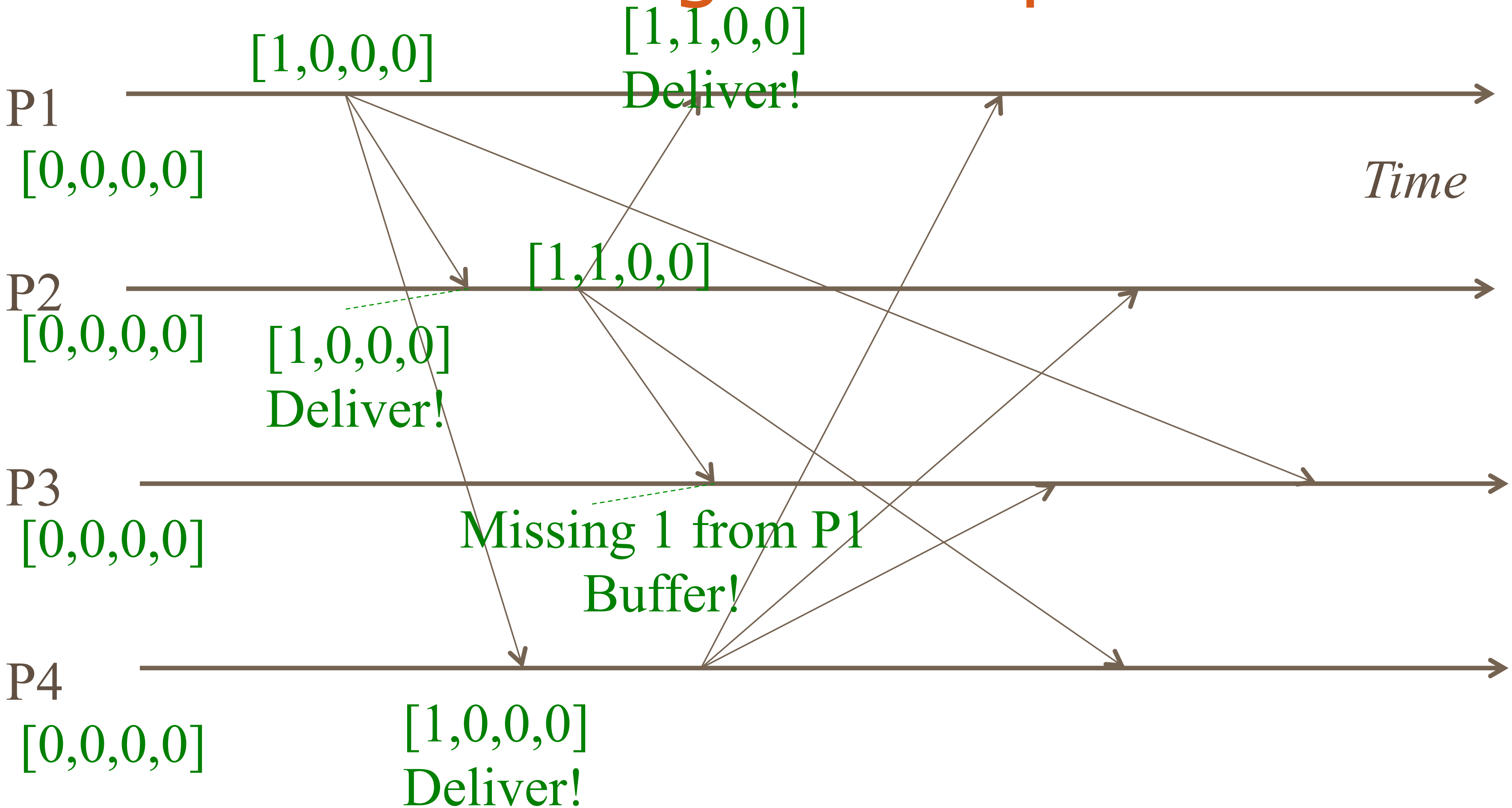
Causal Ordering: Example



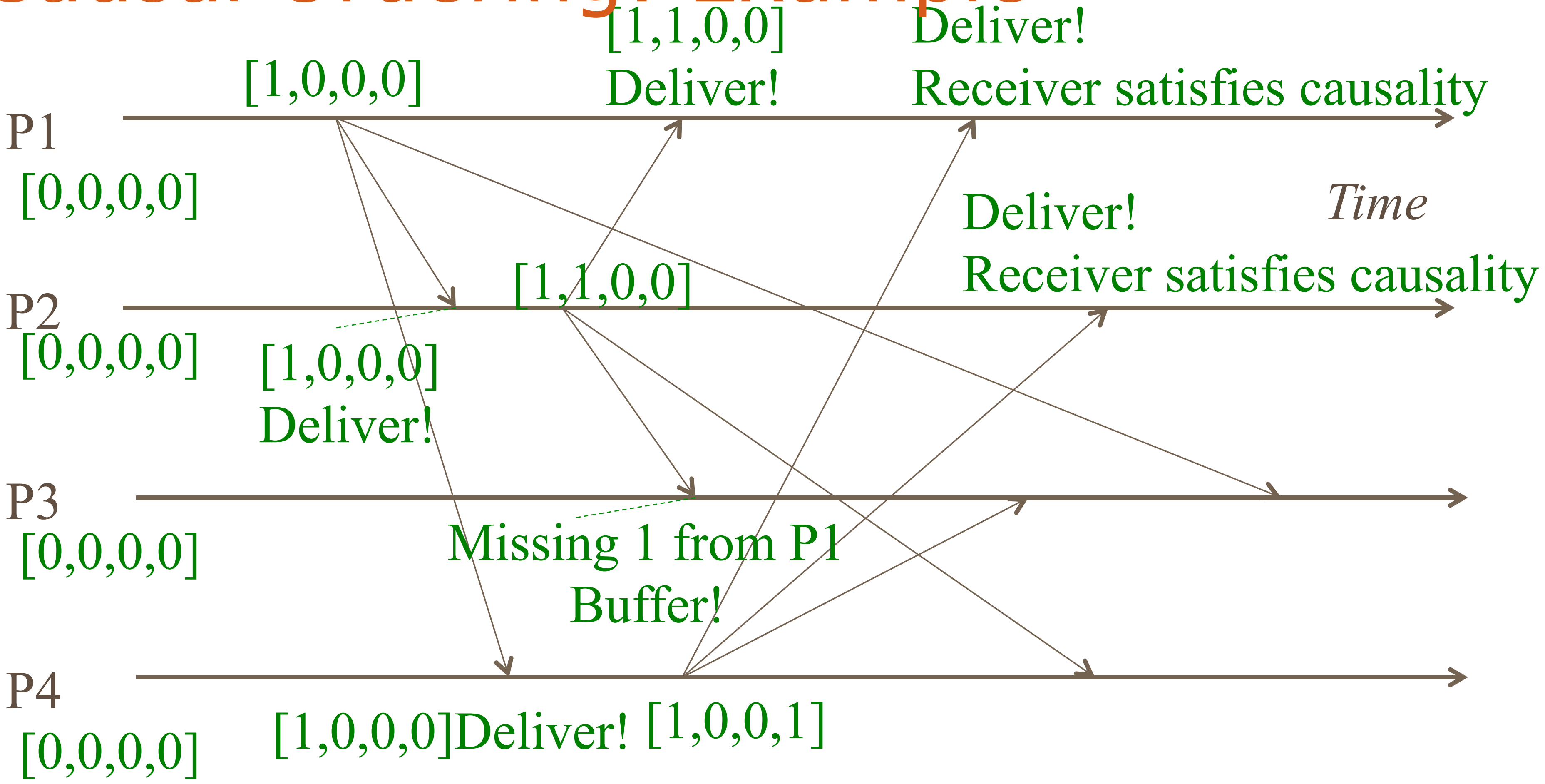
Causal Ordering: Example



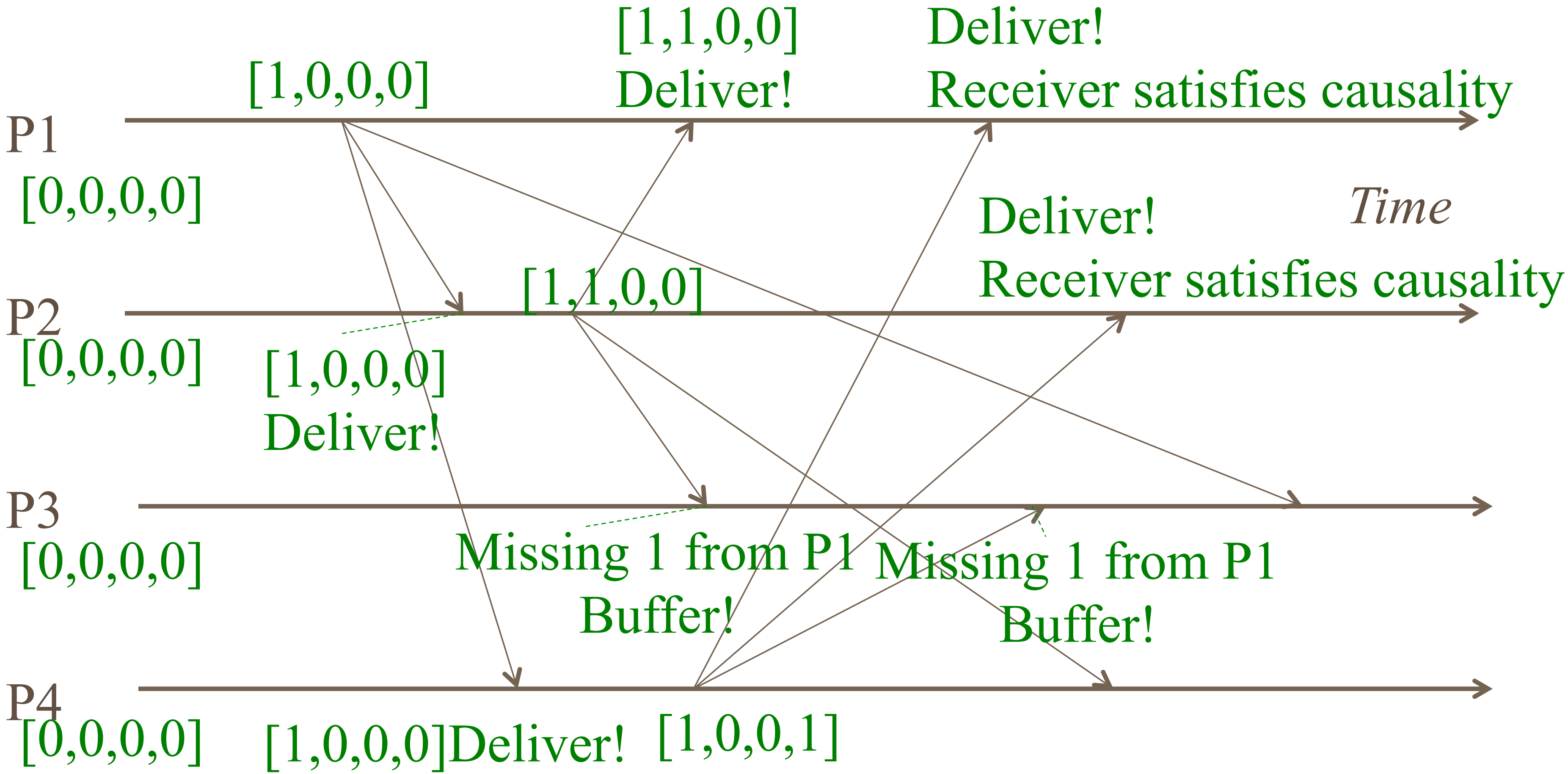
Causal Ordering: Example



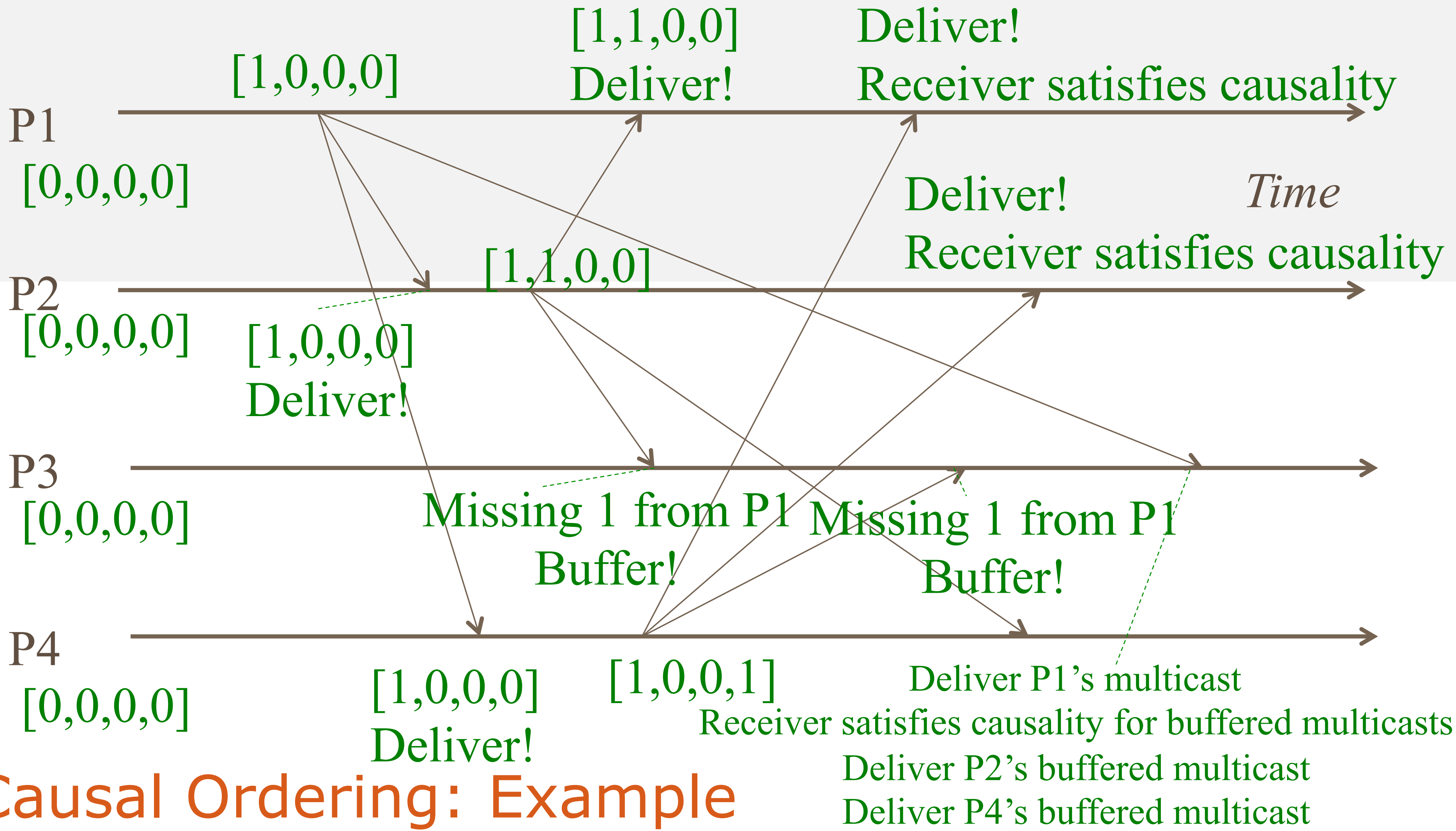
Causal Ordering: Example



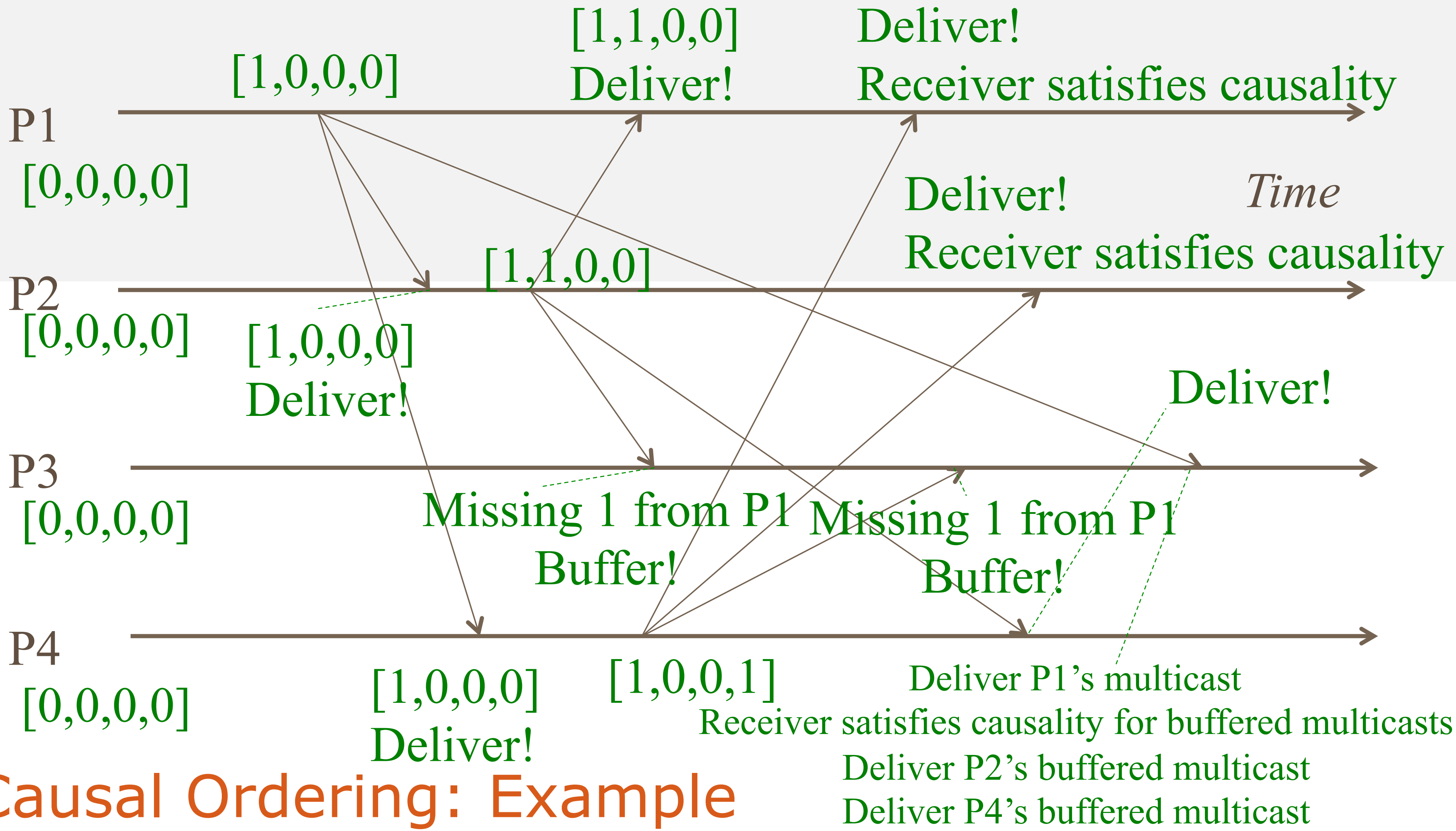
Causal Ordering: Example



Causal Ordering: Example



Causal Ordering: Example



Summary: Multicast Ordering

- Ordering of multicasts affects correctness of distributed systems using multicasts
- Three popular ordering semantics
  - FIFO, Causal, Total
- And their implementations
- What about reliability of multicasts?
- What about failures?

Reliable Multicast

- Reliable multicast loosely says that every process in the group receives all multicasts
  - Reliability is orthogonal to ordering
  - Can implement Reliable-FIFO, or Reliable-Causal, or Reliable-Total, or Reliable-Hybrid protocols
- What about process failures?
- Definition becomes vague

Reliable Multicast (under failures)

- Need all *correct* (i.e., non-faulty) processes to receive the same set of multicasts as all other correct processes
  - Faulty processes stop anyway, so we won't worry about them

Implementing Reliable Multicast

- Let's assume we have reliable unicast (e.g., TCP) available to us
- First-cut: Sender process (of each multicast M) sequentially sends a reliable unicast message to all group recipients
- First-cut protocol does not satisfy reliability
  - If sender fails, some correct processes might receive multicast M, while other correct processes might not receive M

REALLY Implementing Reliable Multicast

- Trick: Have receivers help the sender
  - Sender process (of each multicast M) sequentially sends a reliable unicast message to all group recipients
  - When a receiver receives multicast M, it also sequentially sends M to all the group's processes

Analysis

- Not the most efficient multicast protocol, but reliable
- Proof is by contradiction
- Assume two correct processes P*i* and P*j* are so that P*i* received a multicast M and P*j* did not receive that multicast M
  - Then P*i* would have sequentially sent the multicast M to all group members, including P*j*, and P*j* would have received M
  - A contradiction
  - Hence our initial assumption must be false
  - Hence protocol preserves reliability