SFU SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

# CMPT 431 Distributed Systems

Fall 2019

# Group Communication

https://www.cs.sfu.ca/~keval/teaching/cmpt431/fall19/

Instructor: Keval Vora

# Reading

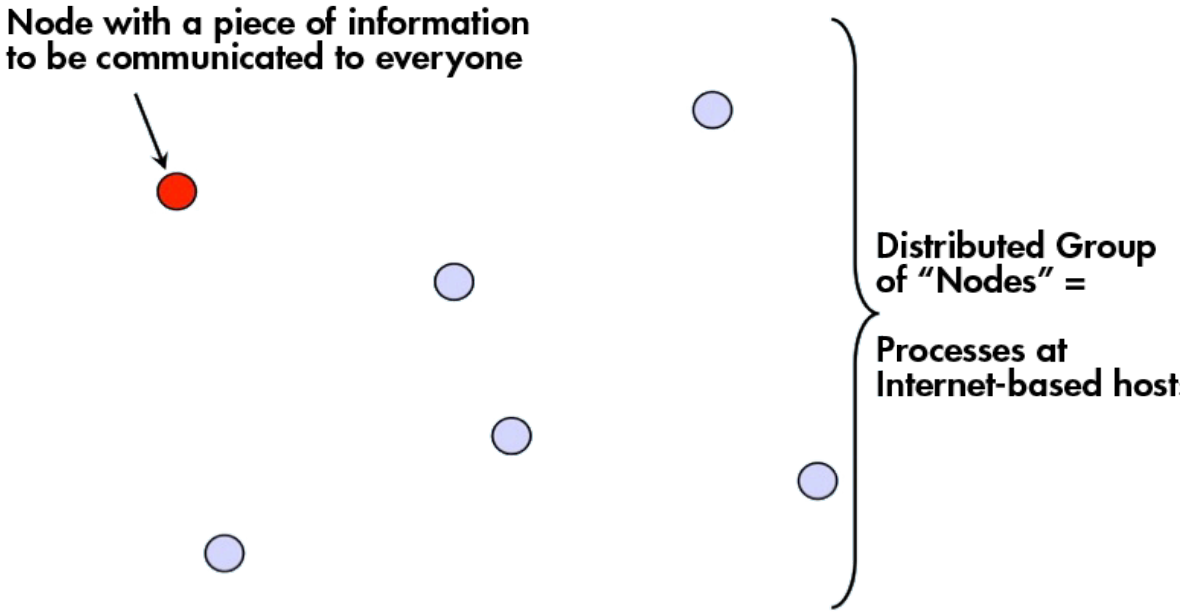

- [DC] Chapter 6
  - Selected topics

- Slides based on Distributed Systems course by Indranil Gupta @ UIUC

# Communication Forms

- Unicast
  - Message sent from one sender process to one receiver process (We relied on this so far in the course)
- Multicast
  - Message sent to a group of processes
- Broadcast
  - Message sent to all processes

# Multicast Problem



# Multicast Uses

- 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 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 multicast protocols
  - FIFO ordering
  - Causal ordering
  - Total ordering

# FIFO Ordering

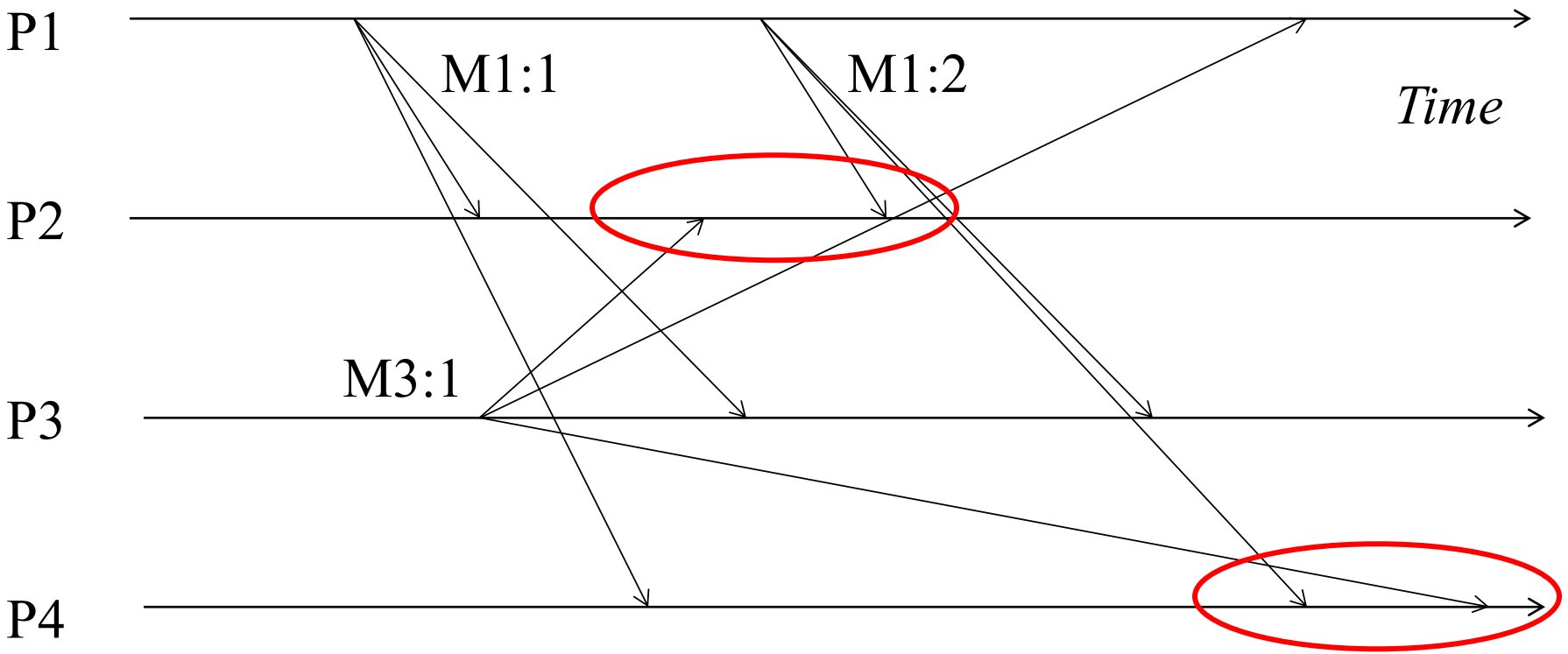
- Multicasts from each sender are received in the order they are sent, at all receivers
- Don't worry about multicasts from different senders

- If a 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
  - (delivers = forwards upwards to application)

# FIFO Ordering

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**



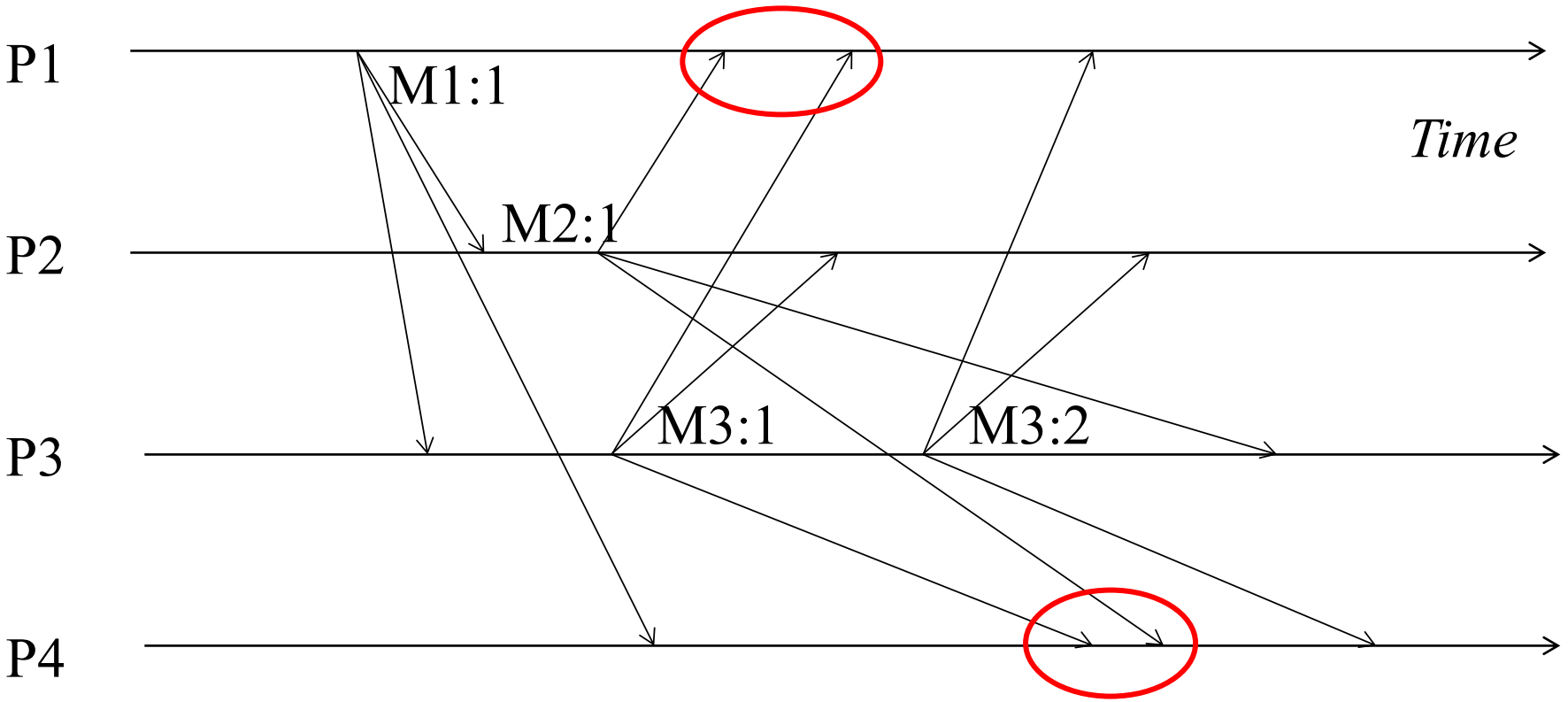
# Causal Ordering

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

- If multicast(g, m) → multicast(g, m') then any correct process that delivers m' would already have delivered m

# Causal Ordering

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 Ordering Uses

- A variety of systems implement causal ordering
  - Social networks, bulletin boards, comments on websites, etc.

- 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 $m_1$ and $m_2$ concurrently, then they can be seen in any order at receivers

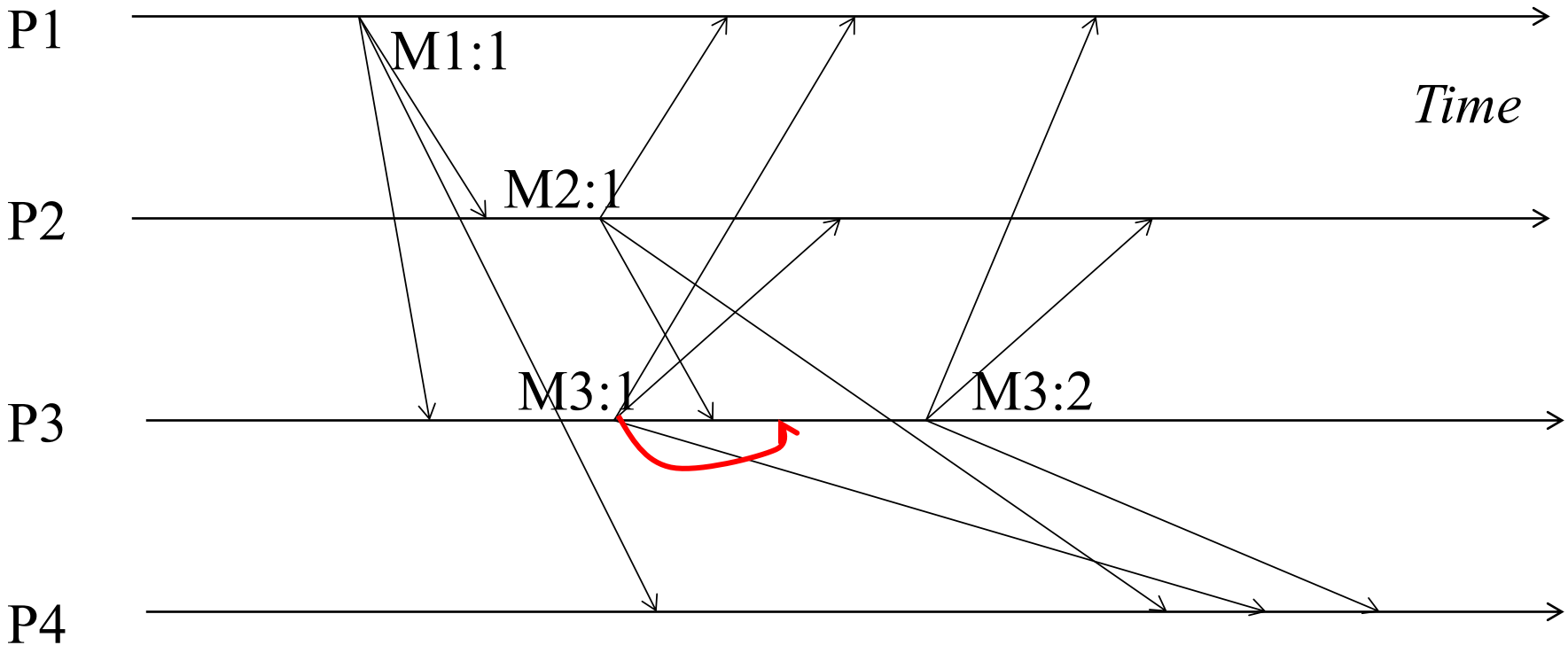
# Total Ordering

- 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

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

# Total Ordering

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 Orderings

- FIFO/Causal ordering are orthogonal to Total ordering

- Can have hybrid ordering protocols
  - FIFO-total hybrid protocol satisfies both FIFO and total orders
  - Causal-total hybrid protocol satisfies both Causal and total orders

- Causal-total hybrid enforces ordering between concurrent events too
  - Any ordering is fine, as long as it is same at all processes

# Implementation

- Next, how to implement these orderings

- FIFO Multicast
- Causal Multicast
- Total Multicast

- Any suggestions on how to implement?

# FIFO Multicast: Data Structures

- Each receiver maintains a per-sender sequence number
- $P_i$ 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 $P_j$

- $P_i[i]$ is the latest sequence number when $P_i$ sent out messages to the group

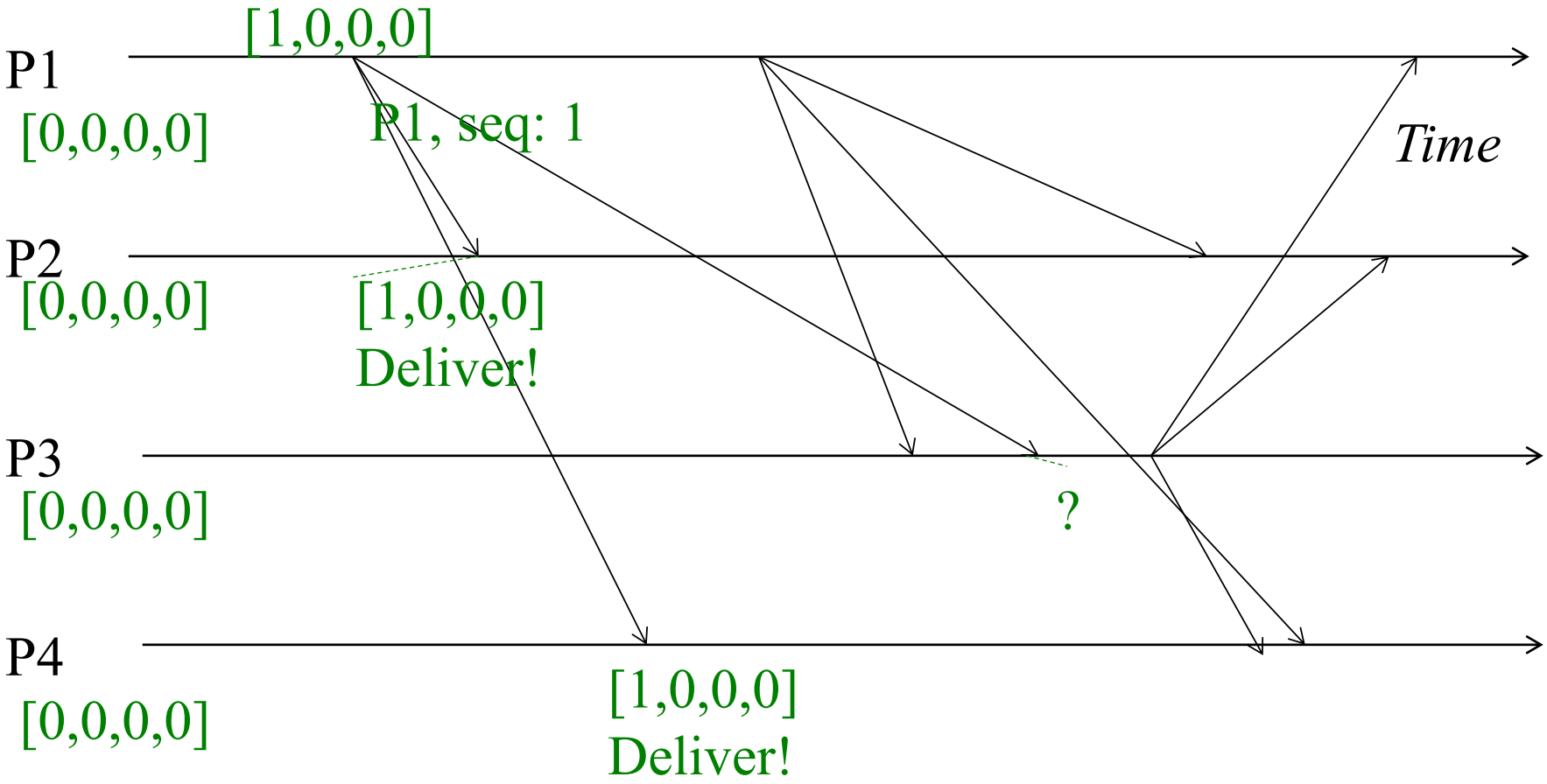
# FIFO Multicast: Updating Rules

- 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

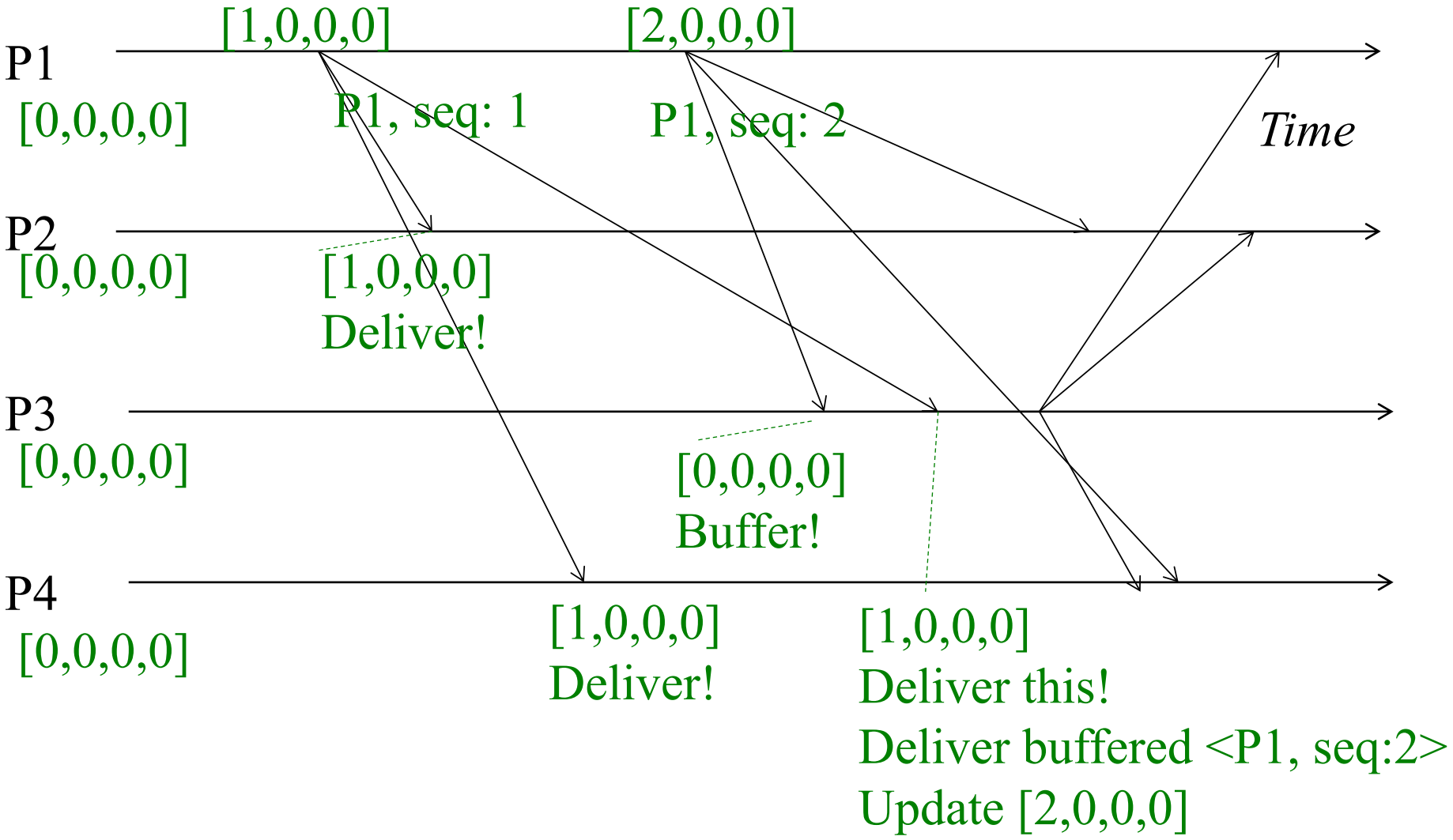
- Receive multicast: $P_i$ receives a multicast from $P_j$ with sequence number S in message
  - If ($S == P_i[j] + 1$) then
    - Deliver message to application
    - Set $P_i[j] = P_i[j] + 1$
  - Else buffer this multicast until above condition is true

# FIFO Ordering

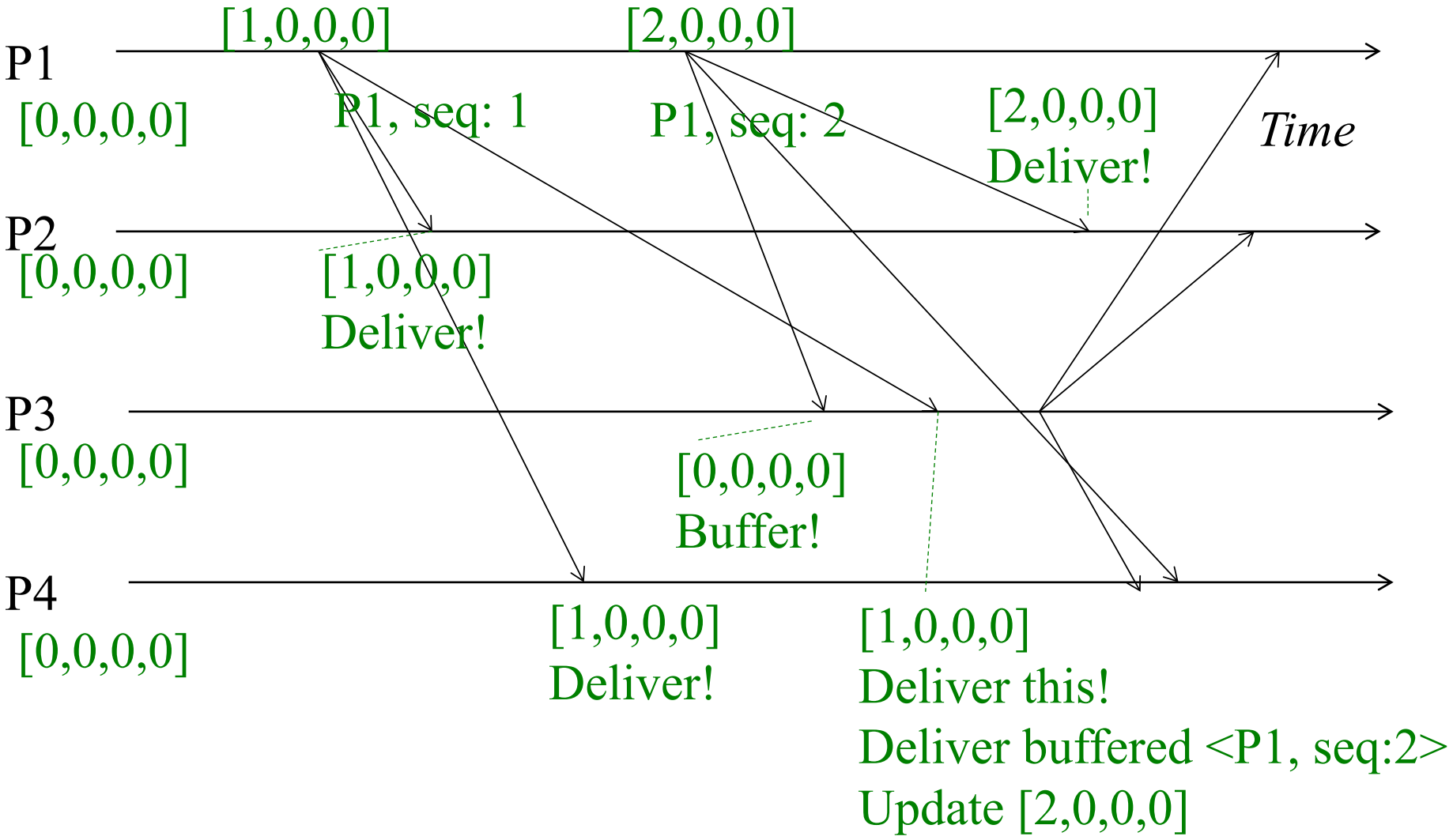
# FIFO Ordering



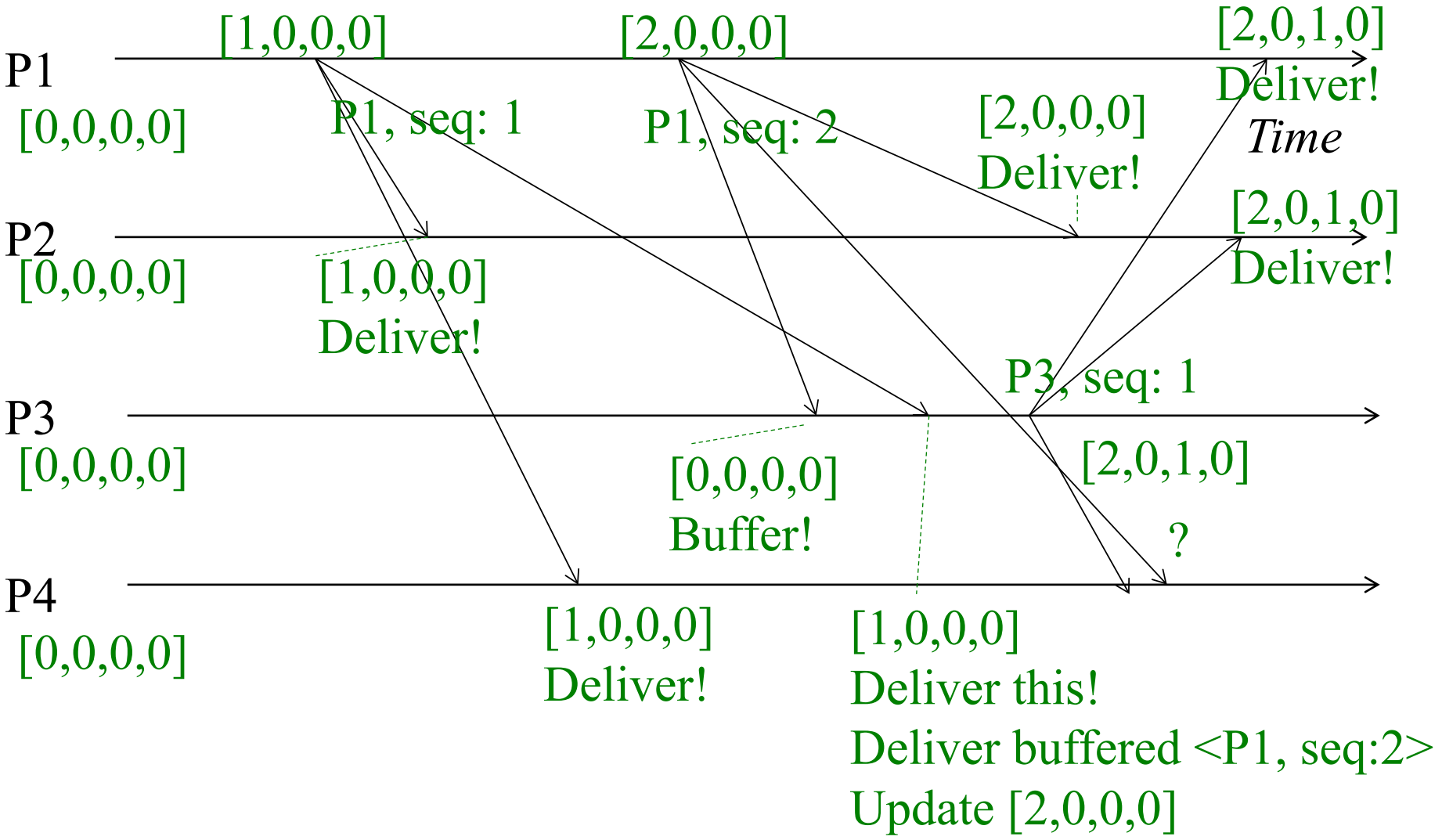
# FIFO Ordering



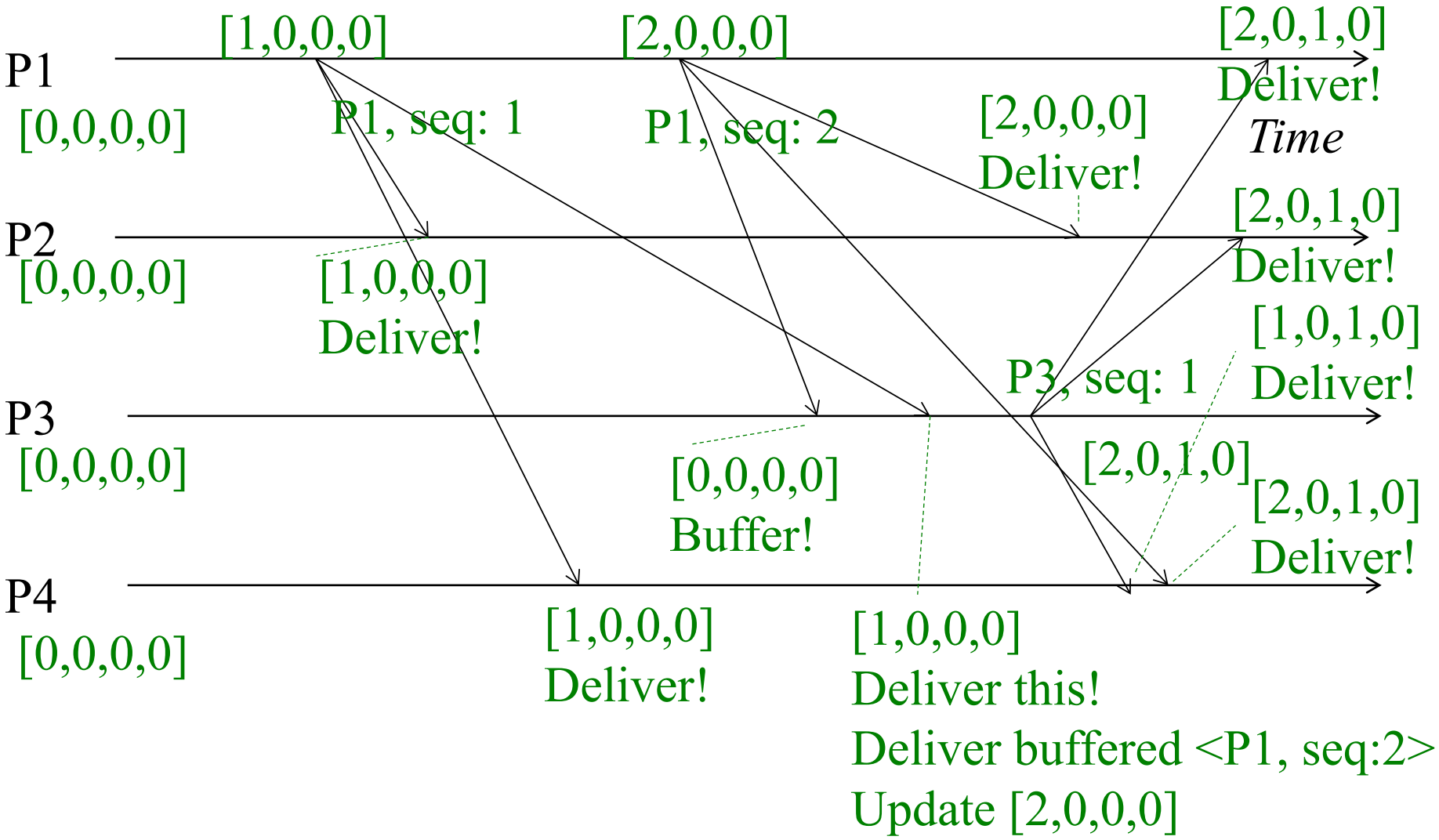
# FIFO Ordering



# FIFO Ordering



# FIFO Ordering



# Causal Ordering

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

Data Structure:

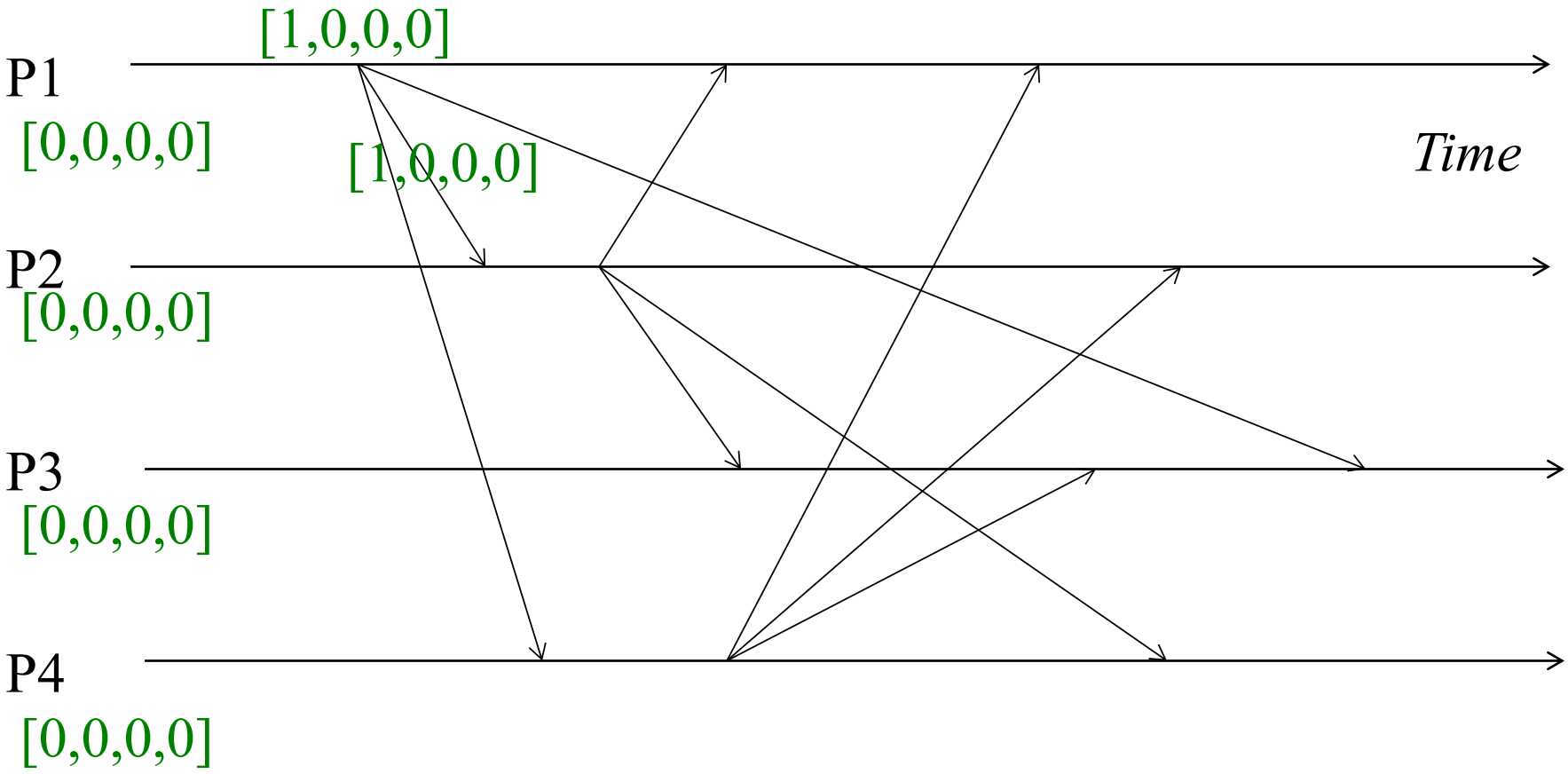
- Each receiver maintains a vector of per-sender sequence numbers
  - Similar to FIFO multicast, but updating rules are different
  - $P_i$ maintains a vector $P_i[1…N]$ (initially all zeroes)
  - $P_i[j]$ is the latest sequence number $P_i$ has received from $P_j$

# Causal Multicast: Updating Rules

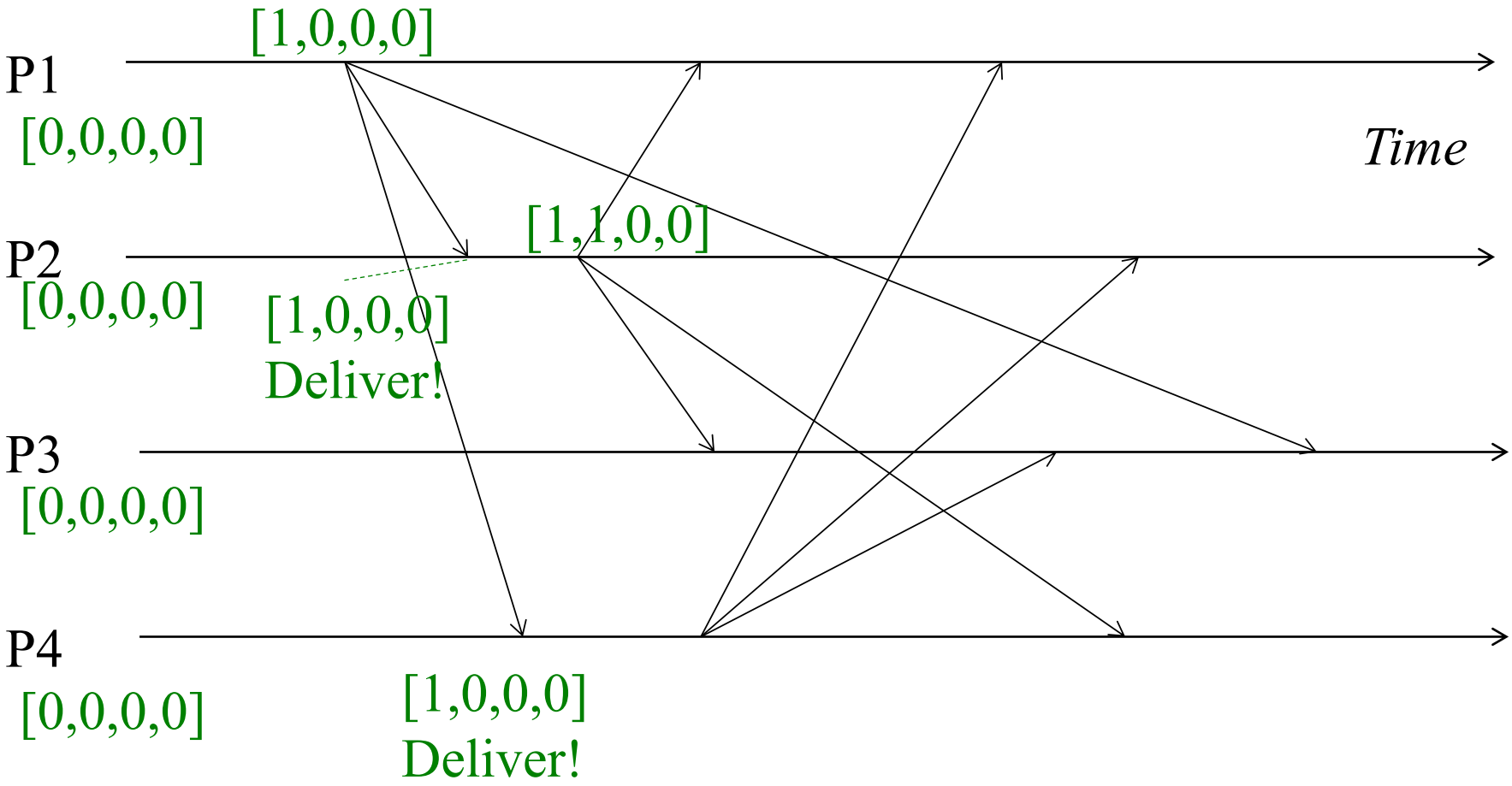
- Send multicast at process $P_j$:
  - Set $P_j[j] = P_j[j] + 1$
  - Include entire vector $P_j[1…N]$ in multicast message as its sequence number

- Receive multicast: if $P_i$ receives a multicast from $P_j$ with vector M[1…N] in message, buffer it until:
  1. This message is the next one $P_i$ is expecting from $P_j$, i.e., $M[j] = Pi[j] + 1$
  2. And, all multicasts, anywhere in the group, which happened before M have been received at $P_i$, i.e., $\forall k \neq j: M[k] \leq P_i[k]$
- When above two conditions satisfied, deliver M to application and set $P_i[j] = M[j]$

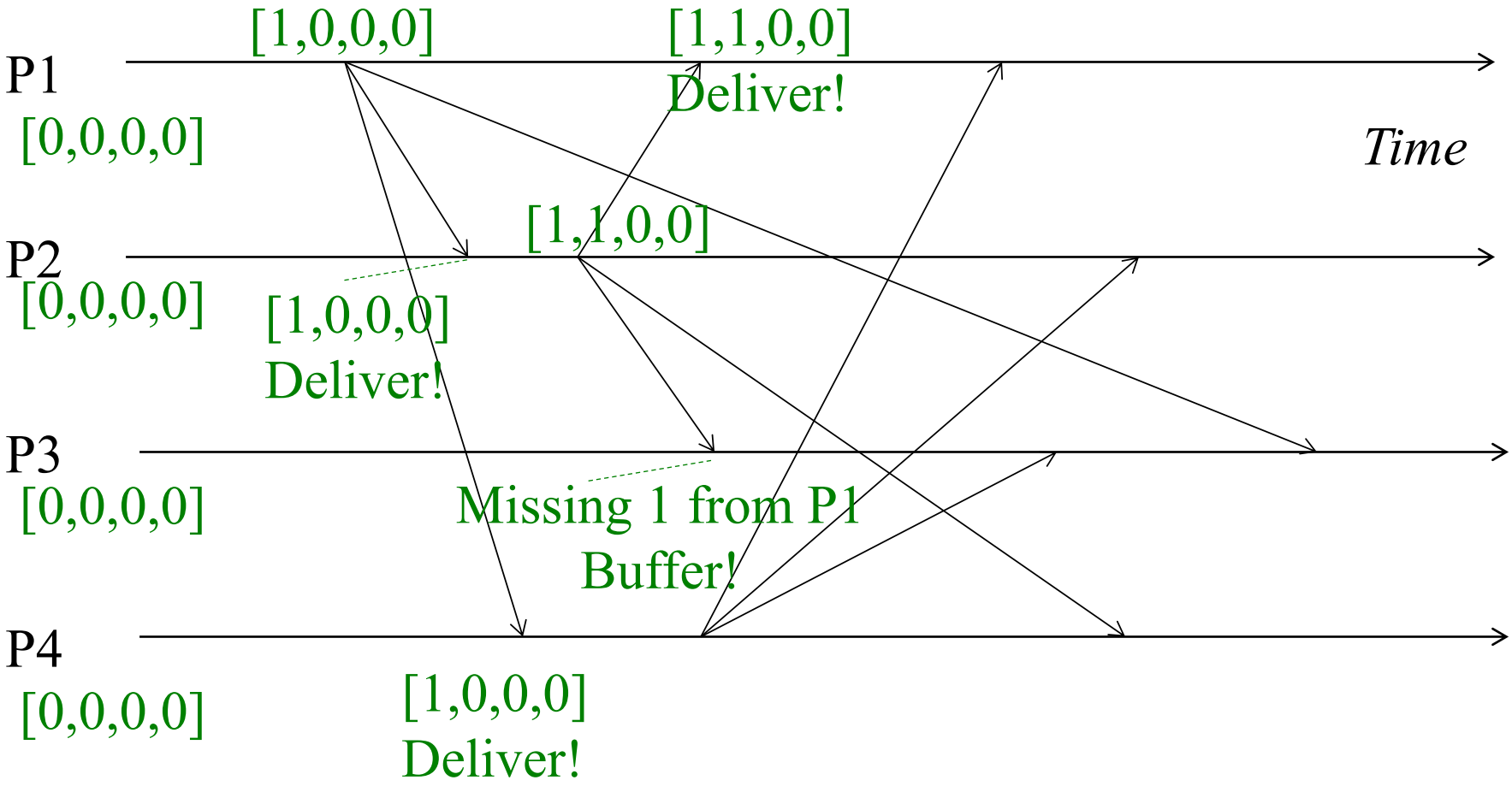
# Causal Ordering



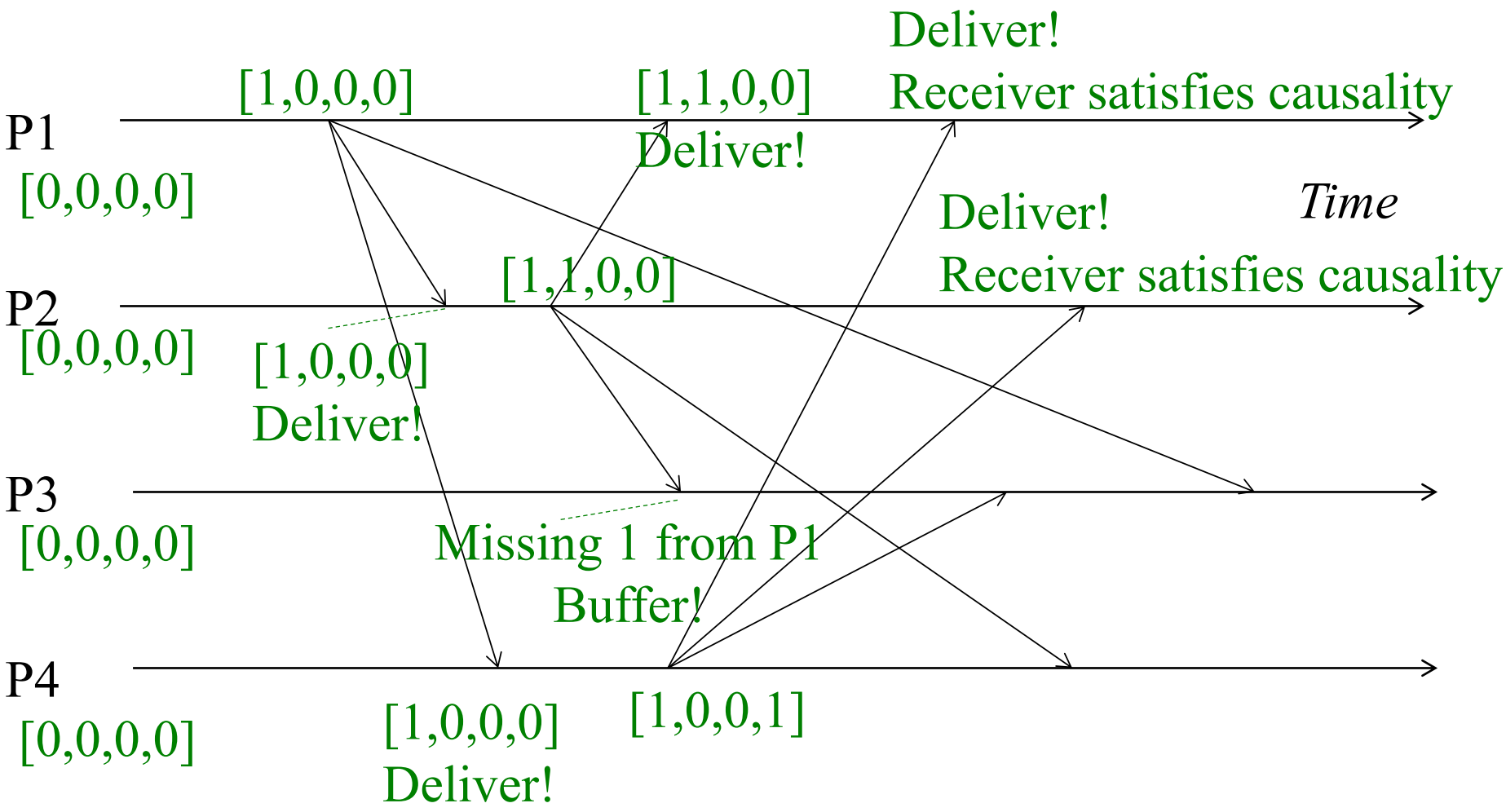
# Causal Ordering



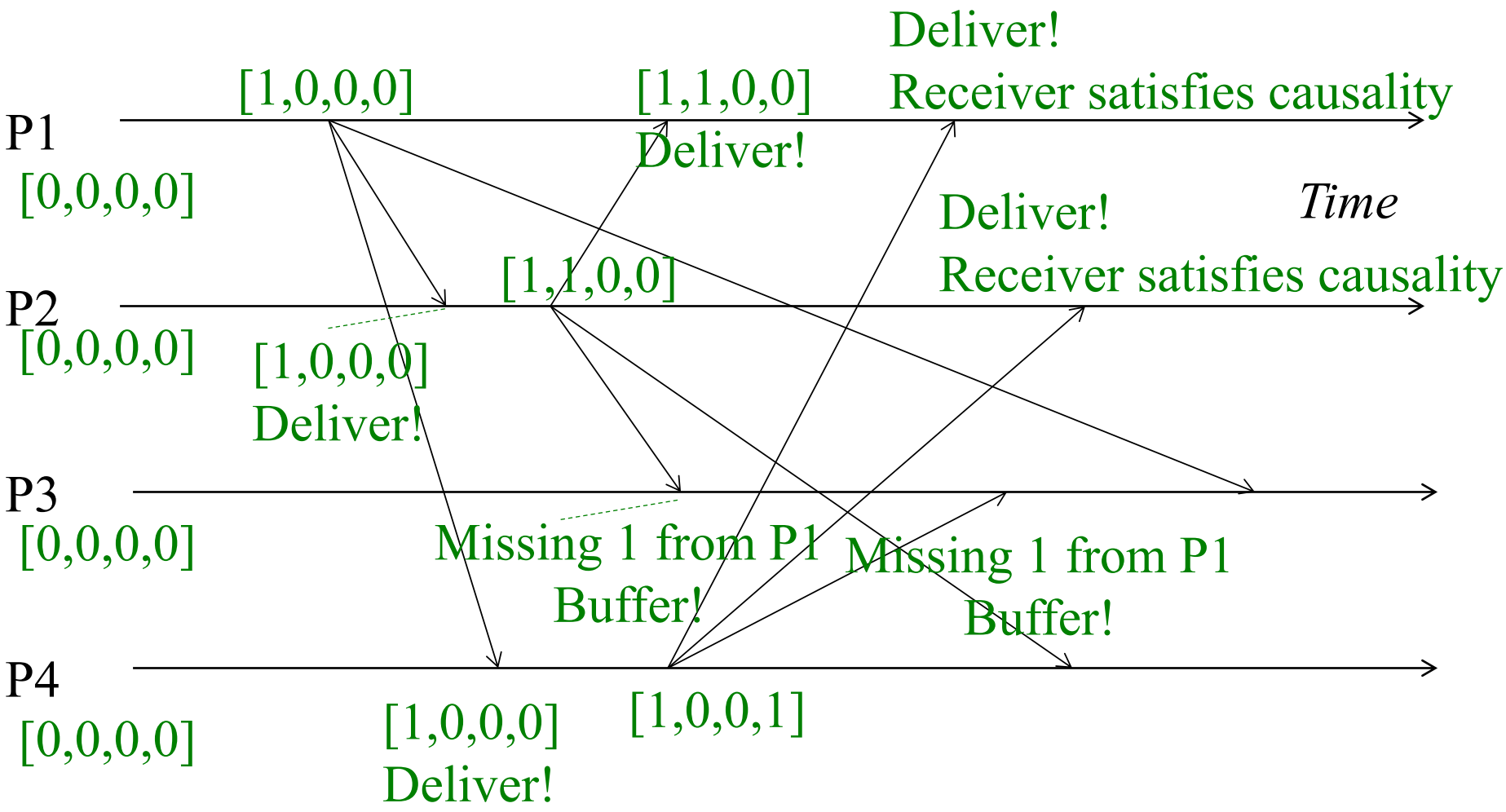
# Causal Ordering



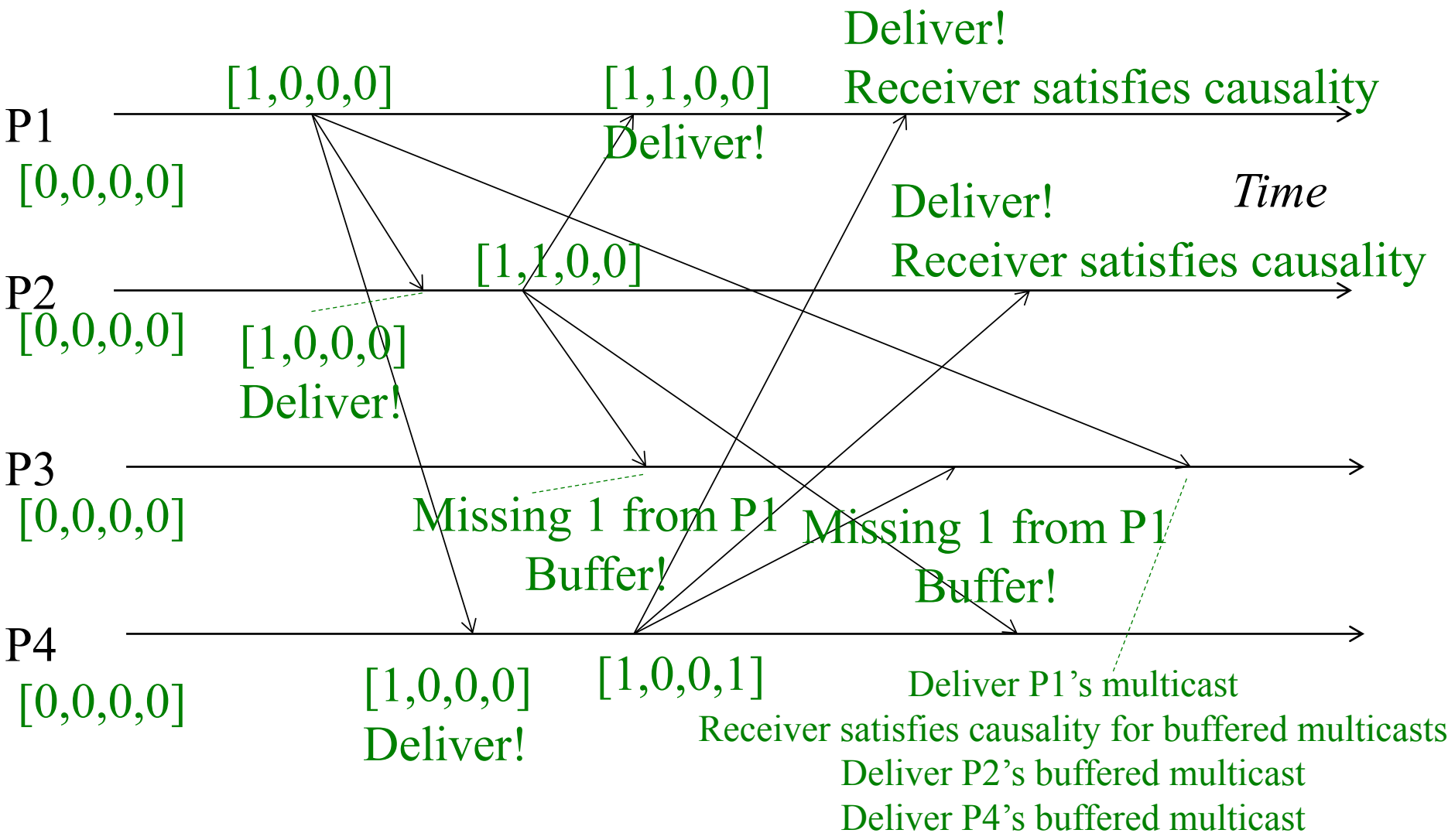
# Causal Ordering



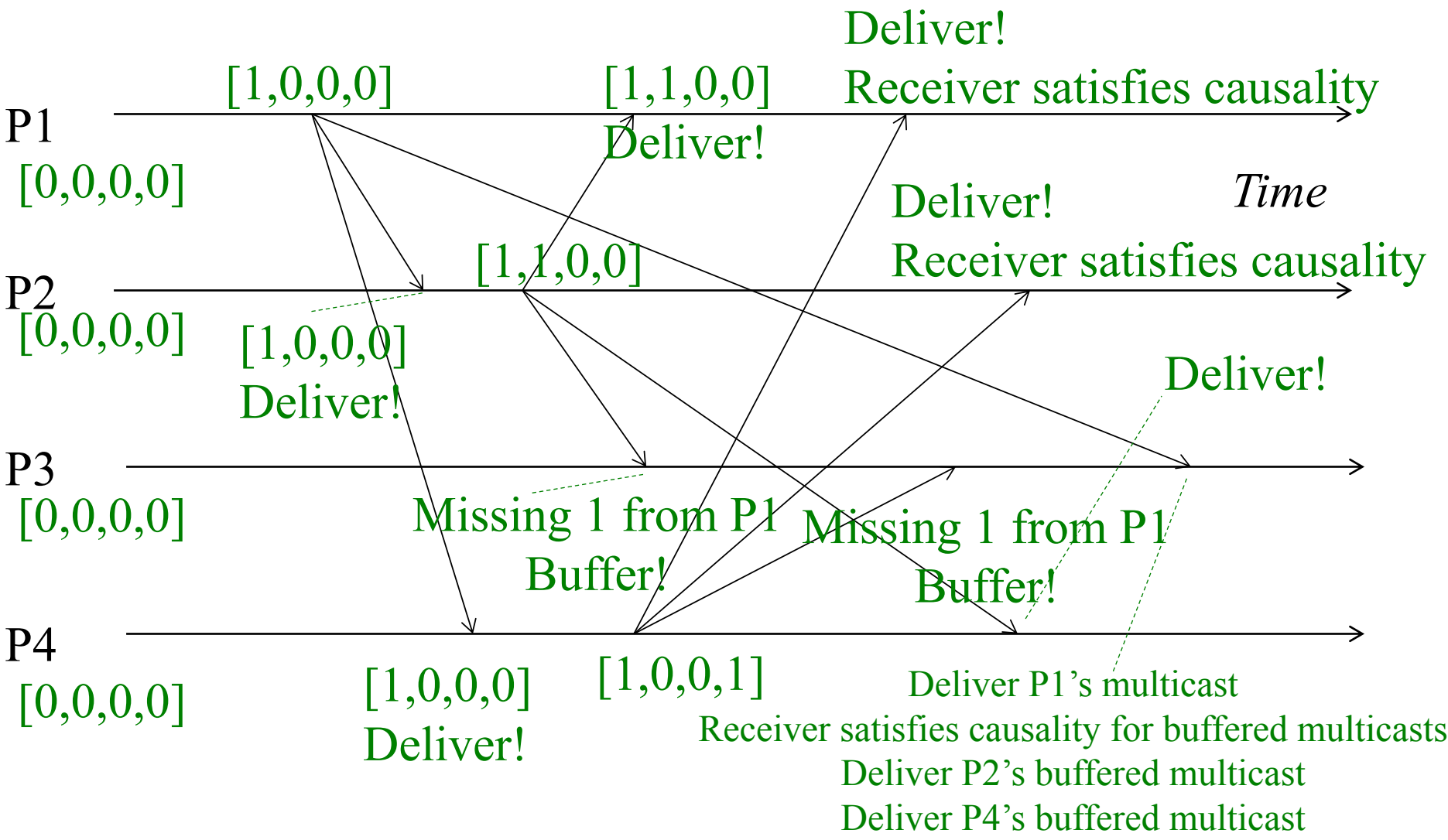
# Causal Ordering



# Causal Ordering



# Causal Ordering



# Total Ordering

- Ensures all receivers receive all multicasts in the same order

- Relies on a special process elected as leader or sequencer
  - Interested students: Check out Leader Election!
  - We will assume leader has been established and known to all

# Sequencer-based Approach

- Send multicast at process $P_i$:
  - Send multicast message M to group and sequencer

- Sequencer:
  - Maintains a global sequence number S (initially 0)
  - Upon receiving a multicast message M, it sets S = S + 1, and multicasts <M, S>

- Receive multicast at process $P_i$:
  - $P_i$ maintains a local received sequence number $S_i$ (initially 0)
  - If $P_i$ receives a multicast M from $P_j$, it buffers M until:
    - $P_i$ receives <M, S(M)> from sequencer, and
    - $S_i + 1 = S(M)$
  - Then $P_i$ delivers its message to application and set $S_i = S_i + 1$

# Summary: Multicast Ordering

- Ordering of multicasts strongly tied to correctness of distributed systems using multicasts
- Three popular ways of implementing ordering
  - 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?
- 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

- Assume we have reliable unicast (e.g., TCP) available
- First try: Sender process (of each multicast M) sequentially sends a reliable unicast message to all group recipients
- Does this satisfy reliability?
  - No: If sender fails, some correct processes might receive multicast M, while other correct processes might not receive M

# 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

# Reliable Multicast

- 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

# Virtual Synchrony or View Synchrony

- Attempts to preserve multicast ordering and reliability in spite of failures

- Combines a membership protocol with a multicast protocol
  - Interested students: Check out membership protocols!