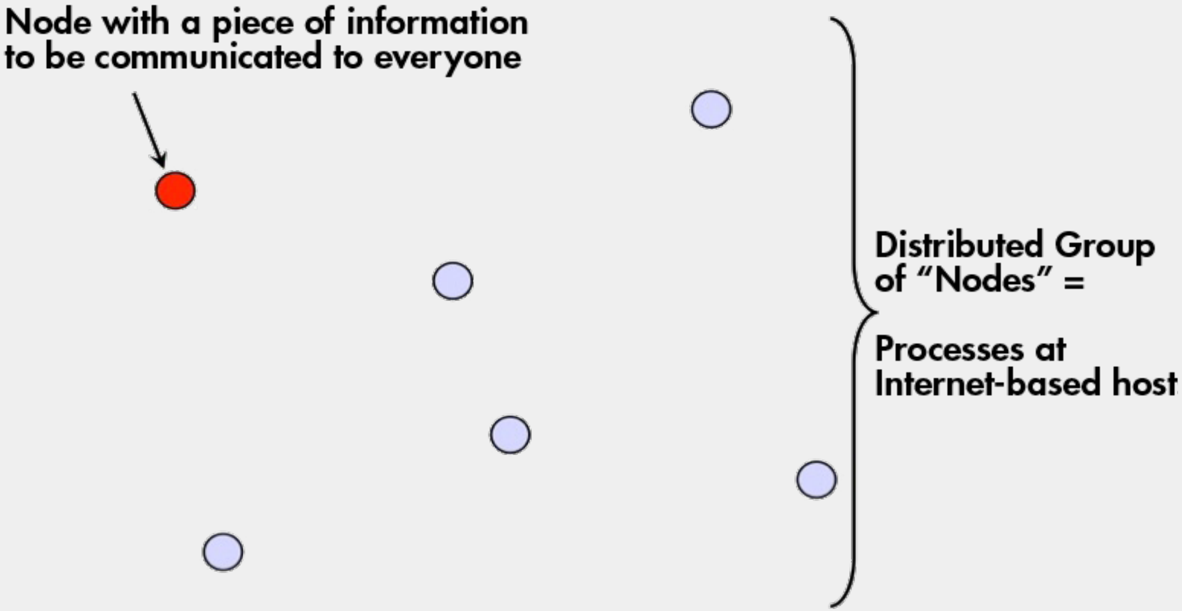
- HW2 is online
- There will be one HW coming out every Wednesday before the midterm

- Synchronous systems vs asynchronous systems
- Safety vs liveness
- Crash/Stop failure vs Byzantine failure
- Why is election hard?

ECEN 757
Multicast

Chapter 15

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?

- Snapshot Algorithm
- Ricart and Agrawala’s Algorithm for mutual exclusion
- Maekawa’s Algorithm for mutual exclusion
- The bully algorithm for election
- …and many more

System Model

- A bunch of clients, separated into several groups
  - Each client belongs to one group
- Reliable one-to-one communication channels
  - No delay guarantees
- Processes can fail only by crashing

- Two functions:
- multicast(g,m): sends the message m to all clients in group g
- deliver(m): after receiving a message, deliver it to application

Basic Multicast

- To B-multicast(g,m): send message m to each process in g
- On receive(m) at p: B-deliver(m) at p

- What are the potential problems?

Basic Multicast

- To B-multicast(g,m): send message m to each process in g
- On receive(m) at p: B-deliver(m) at p

- What are the potential problems?
- Different processes may not receive messages in the same order
- They may not even receive the same messages!
  - The sender may fail before it sends messages to everyone

Reliable Multicast

- How to define “reliable multicast”?
- We require the following properties:
- Integrity: A correct process p delivers a message m at most once
- Validity: If a correct process multicasts message m, then it will eventually deliver m
- Agreement: If a correct process delivers message m, then all other correct processes in the group will eventually deliver m
- Note: Validity + Agreement implies Liveness

R-Multicast over B-Multicast

On initialization
  $Received := \{\}$;

For process p to R-multicast message m to group g
  B-multicast(g, m);   // $p \in g$ is included as a destination

On B-deliver(m) at process q with g = group(m)
  if $(m \notin Received)$
  then
    $Received := Received \cup \{m\}$;
    if $(q \neq p)$ *then B-multicast*(g, m); *end if*
    R-deliver m;
  end if

R-Multicast Properties

- Have all three properties:
- Integrity: A correct process p delivers a message m at most once
- Validity: If a correct process multicasts message m, then it will eventually deliver m
- Agreement: If a correct process delivers message m, then all other correct processes in the group will eventually deliver m

- Still, different processes may receive the messages in different order

Multicast Ordering

- Determines the meaning of “same order” of multicast delivery at different processes in the group
- Three popular flavors implemented

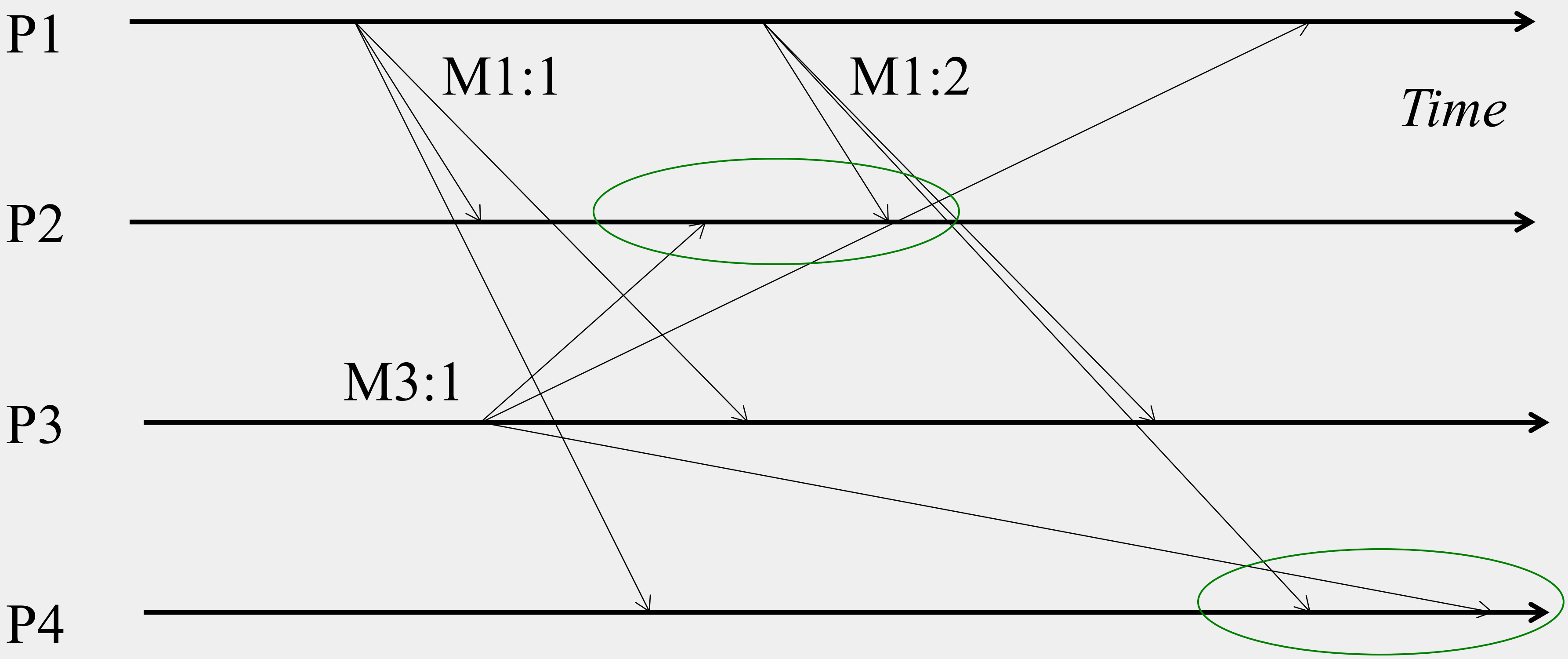
  by several multicast protocols

  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

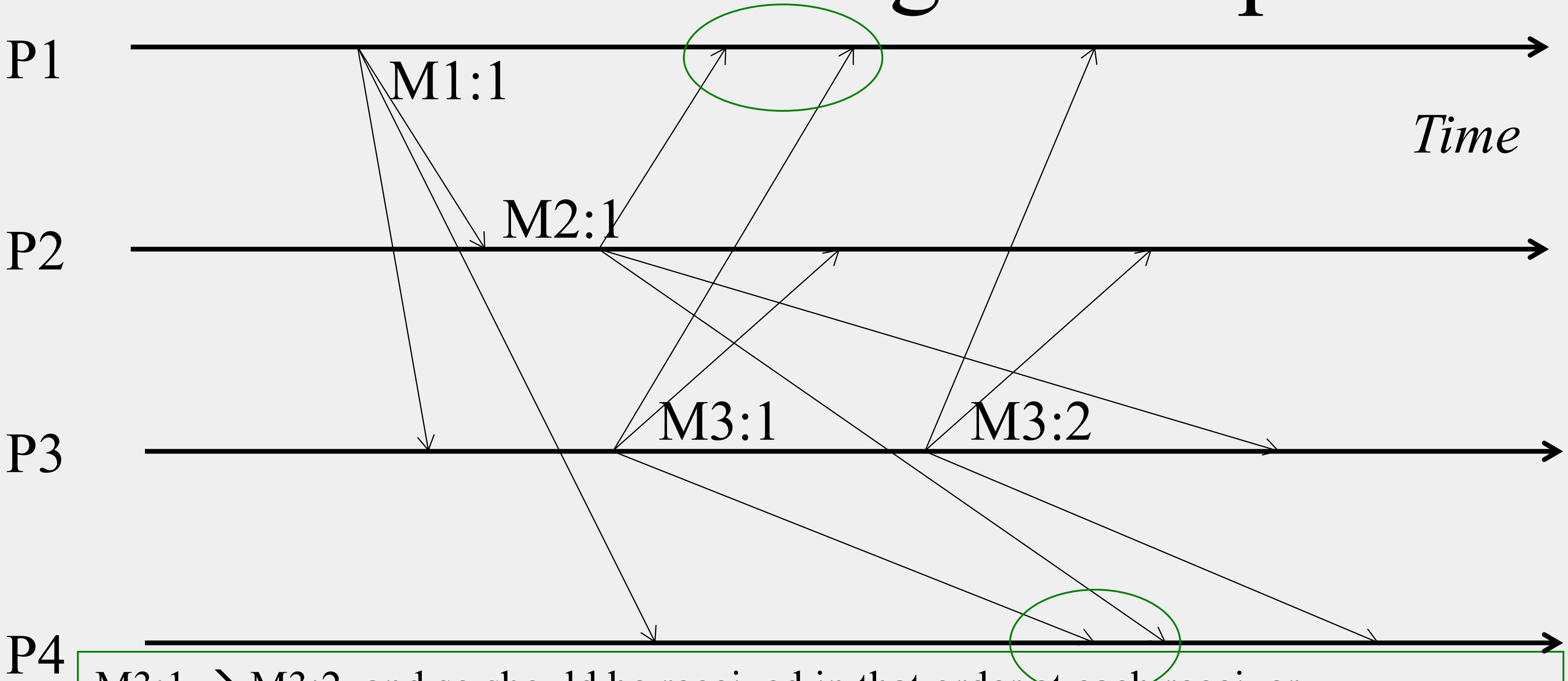


M1:1 and M1:2 should be received in that order at each receiver
Order of delivery of M3:1 and M1:2 could be different at different receivers

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.

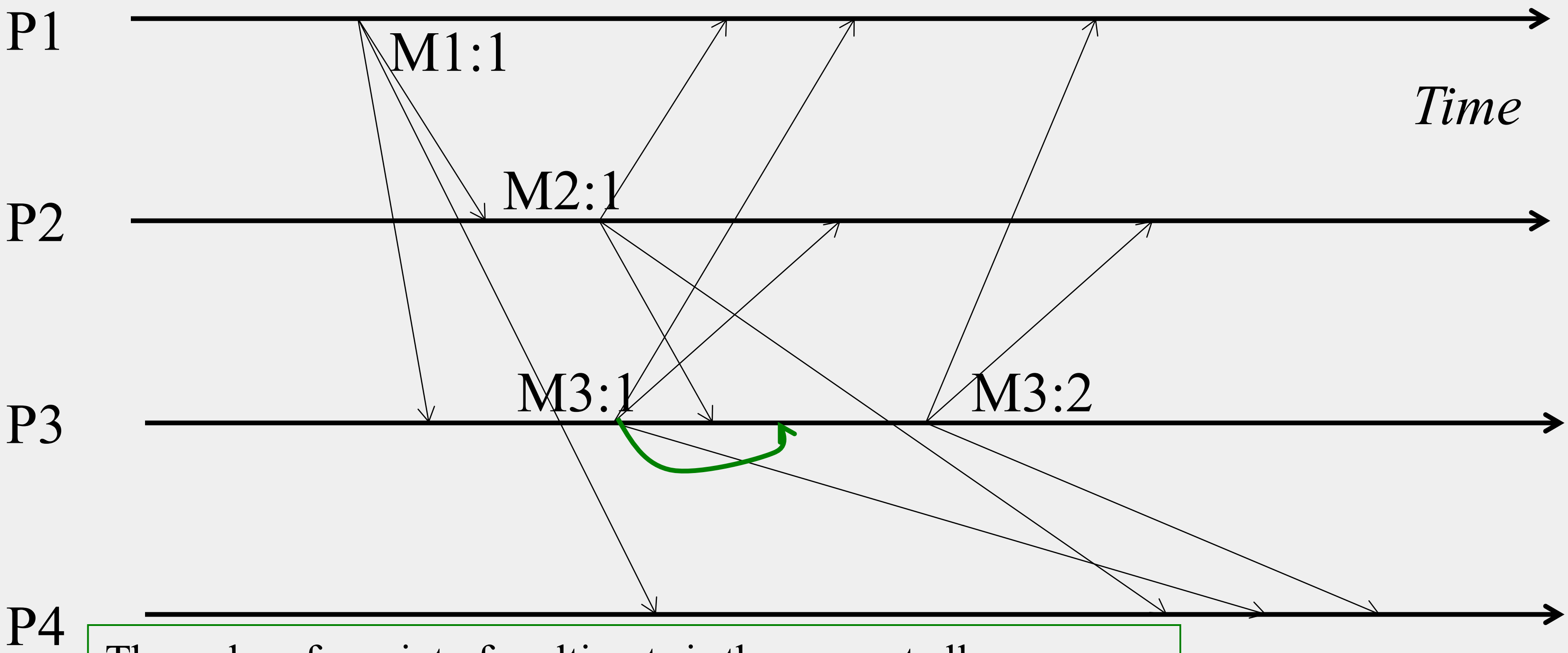
Why Causal at All?

- 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



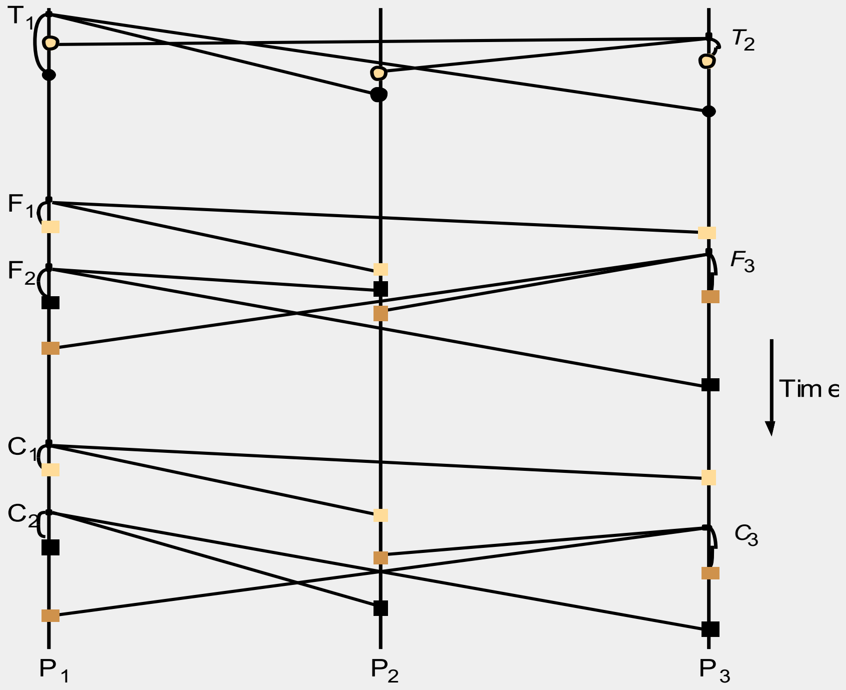
The order of receipt of multicasts is the same at all processes.
M1:1, then M2:1, then M3:1, then M3:2
May need to delay delivery of some messages

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

Example

Notice the consistent ordering of totally ordered messages T_1 and T_2, the FIFO-related messages F_1 and F_2 and the causally related messages C_1 and C_3 – and the otherwise arbitrary delivery ordering of messages.



Implementation?

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

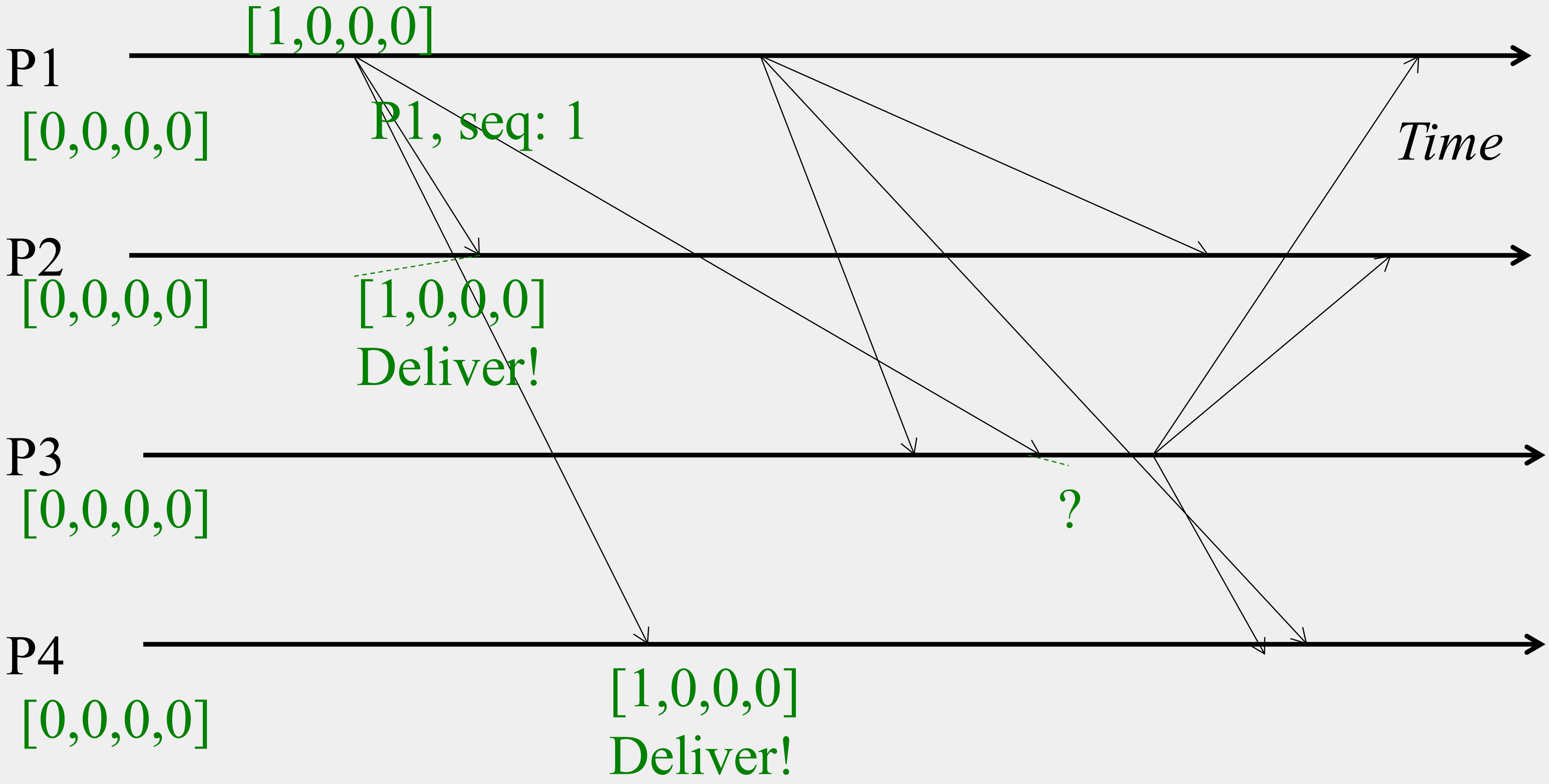
FIFO Multicast: Data Structures

- 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

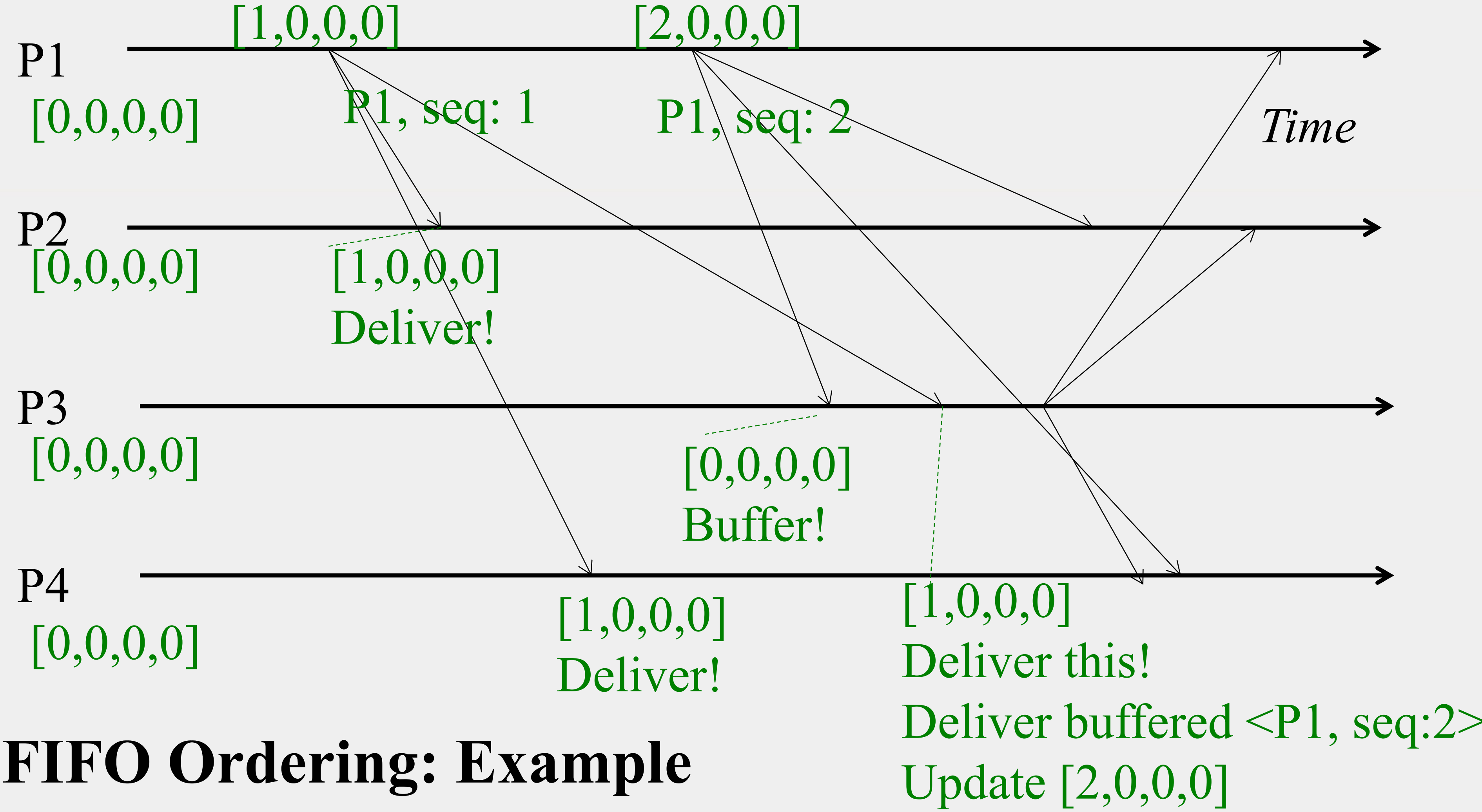
- Send multicast at process Pj:
  - Set Pj[j] = Pj[j] + 1
  - Include new Pj[j] in multicast message as its sequence number
- Receive multicast: If Pi receives a multicast from Pj with sequence number S in message
  - if (S == Pi[j] + 1) then
    - deliver message to application
    - Set Pi[j] = Pi[j]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

FIFO Ordering: Example

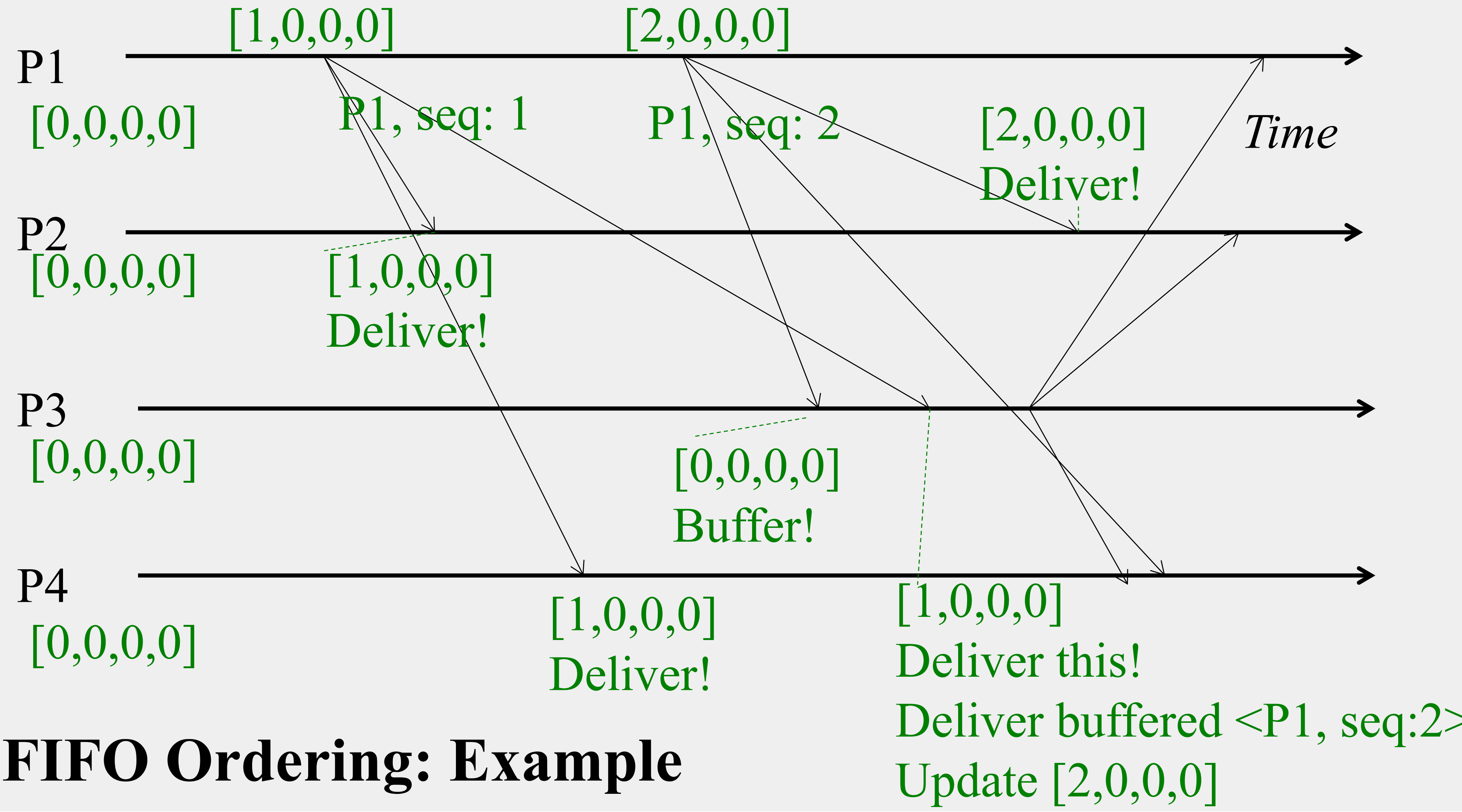


FIFO Ordering: Example

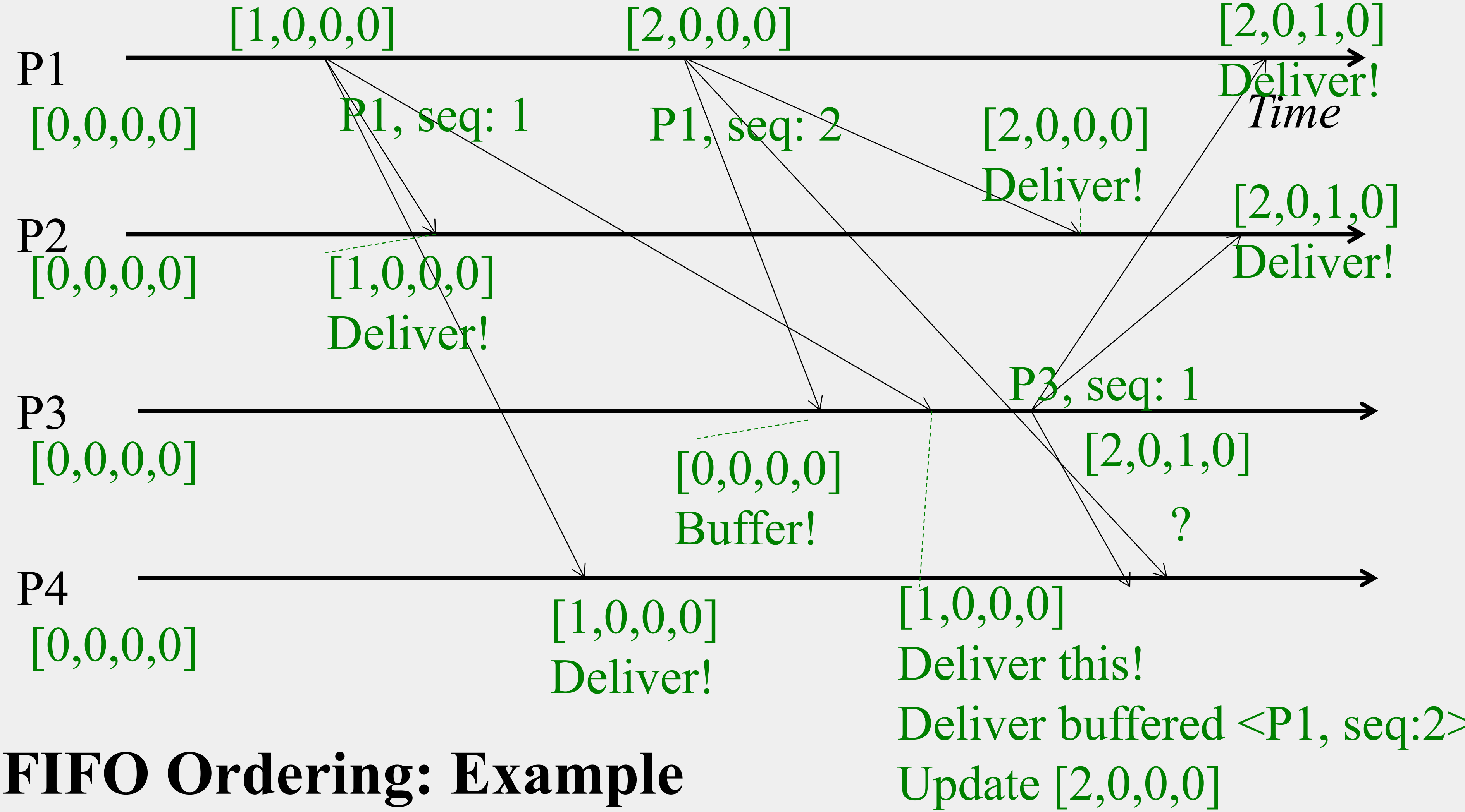
FIFO Ordering: Example



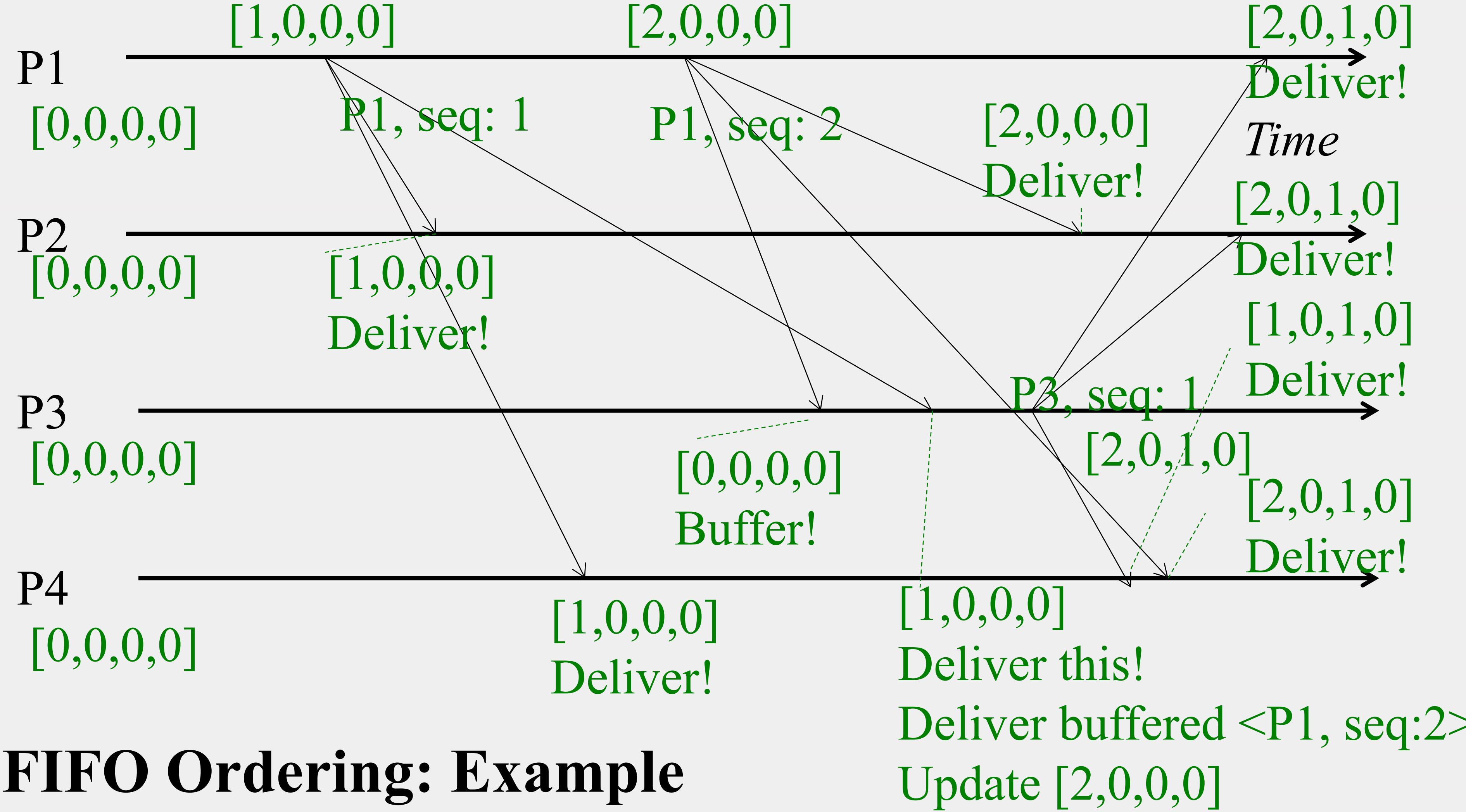
FIFO Ordering: Example



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.*

Basic Idea

- Maintain a “global sequence number”
- Whenever a process needs to multicast, it needs to:
- 1. Get the sequence number
- 2. Multicast the packet
- 3. Increase the sequence number for others

- What mechanism can we use?

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>
- 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

Sequencer-based Approach

1. Algorithm for group member p

On initialization: $r_g := 0$;

To TO-multicast message m to group g
    B-multicast($g \cup \{sequencer(g)\}$, $<m, i>$);

On B-deliver($<m, i>$) *with* $g = group(m)$
    Place $<m, i>$ in hold-back queue;

On B-deliver($m_{order} = <$"order", i, $S>$) *with* $g = group(m_{order})$
    wait until $<m, i>$ in hold-back queue and $S = r_g$;
    TO-deliver m;   // (after deleting it from the hold-back queue)
    $r_g = S + 1$;

2. Algorithm for sequencer of g

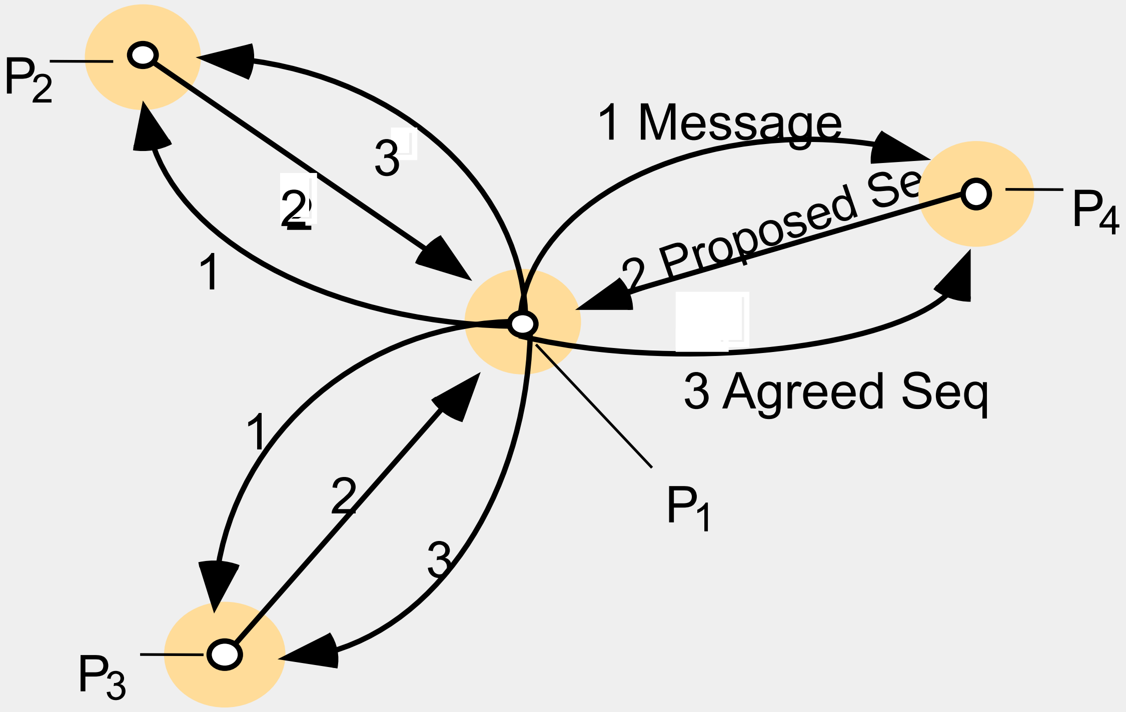
On initialization: $s_g := 0$;

On B-deliver($<m, i>$) *with* $g = group(m)$
    B-multicast(g, $<$"order", i, $s_g>$);
    $s_g := s_g + 1$;

Distributed Approach

- Each process q maintains two numbers:
- A_g^q: the largest agreed sequence number it has observed so far
- P_g^q: its own largest proposed sequence number

Distributed Approach



Distributed Approach

- 1. p multicasts a message
- 2. Upon receiving message m, process q puts it in the buffer, and proposes a sequence number = $\max\{A_g^q, P_g^q\} + 1$
- 3. When p receives all proposed sequence number, it determines the final sequence number as the maximum among all proposed numbers. It multicasts the final sequence number

How to Deliver Messages

- Sort all messages in the buffer by their sequence numbers, either the “proposed” sequence number or the “final” sequence number

- If the message with the smallest number has a “final” sequence number, the message can be delivered

- Why?

Causal Ordering

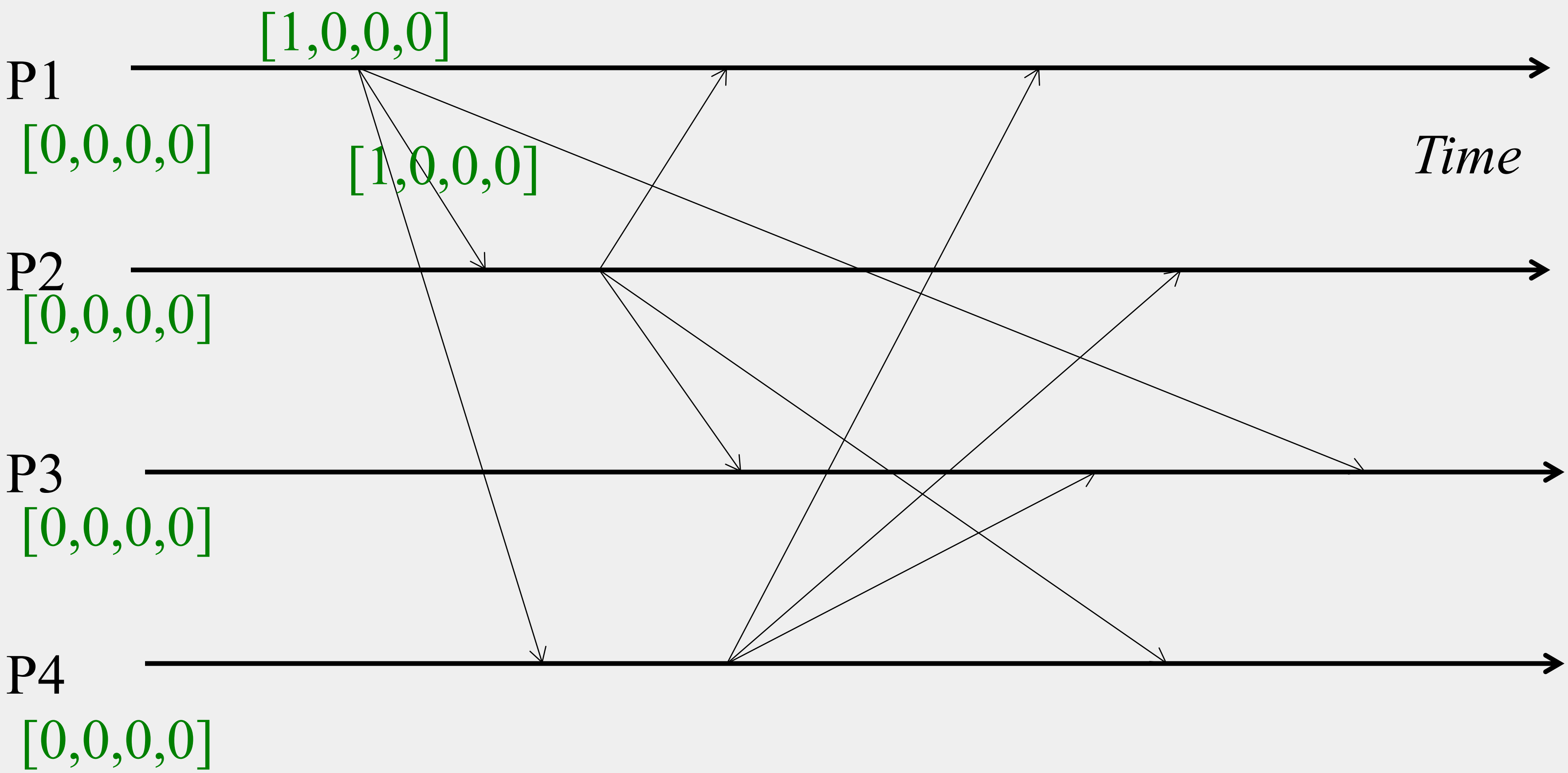
- Multicasts whose send events are causally related, must be received in the same causality-obeying order at all receivers
- Formally
  - *If multicast(g,m) → multicast(g,m′) then any correct process that delivers m′ would already have delivered m.*
  - *(→ is Lamport's happens-before)*

Causal Multicast: Data Structures

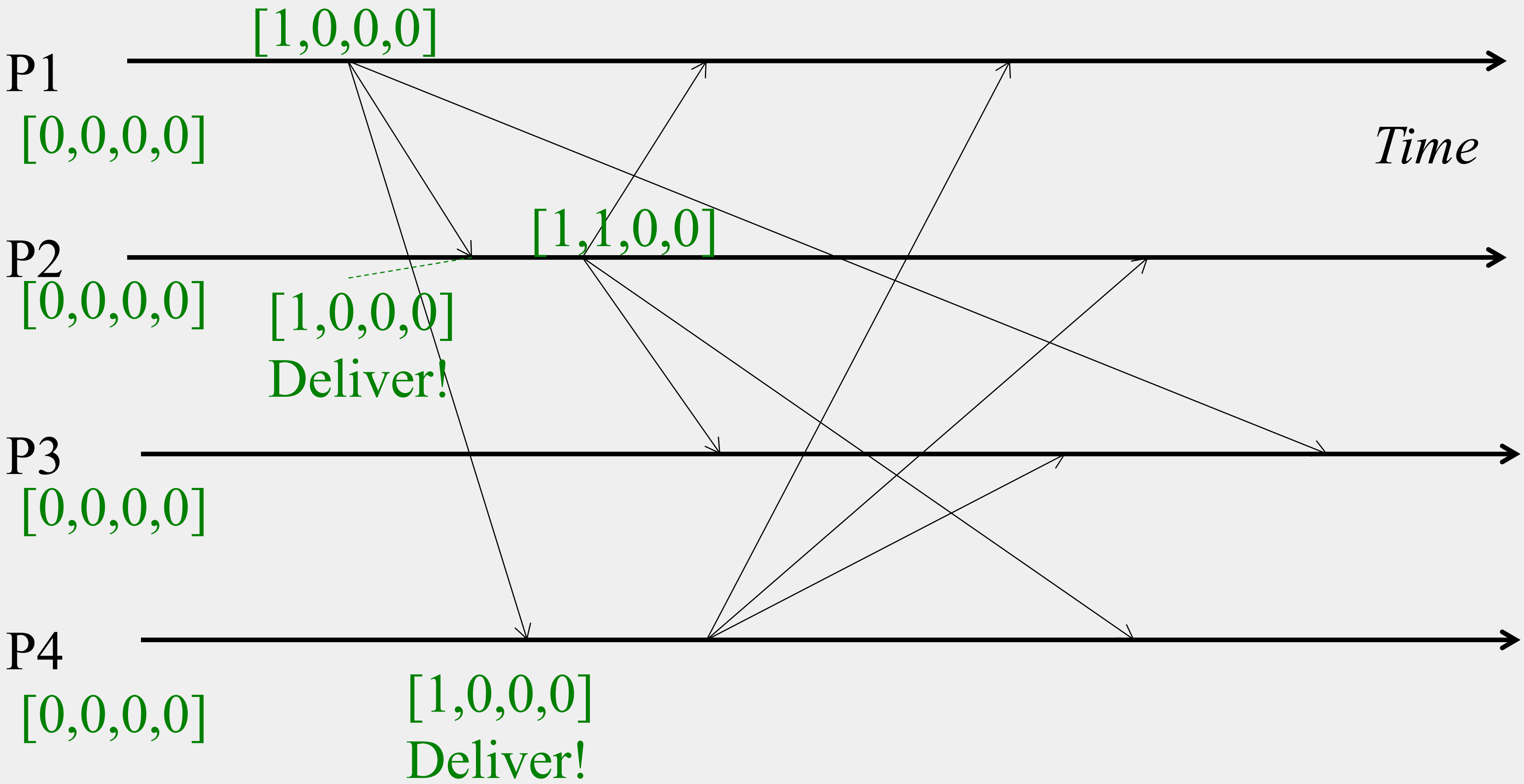
- Each receiver maintains a vector of per-sender sequence numbers (integers)
  - Similar to FIFO Multicast,
    but updating rules are different
  - Processes P*1* through P*N*
  - P*i* maintains a vector P*i*[1…N]
    (initially all zeroes)
  - P*i*[*j*] is the latest sequence number P*i*
    has received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

Causal Multicast: Updating Rules

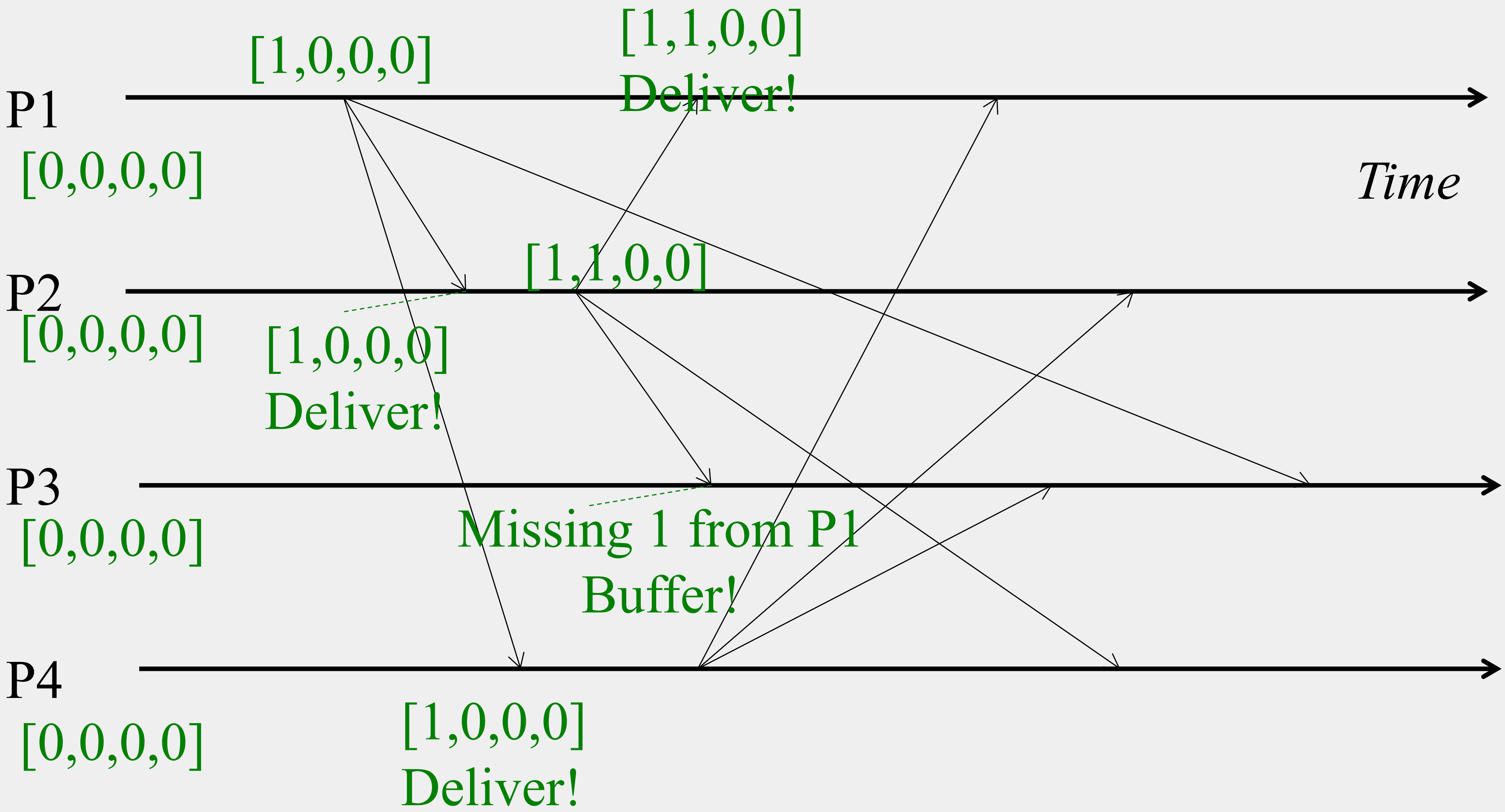
- Send multicast at process Pj:
  - Set Pj[j] = Pj[j] + 1
  - Include new entire vector Pj[1…N] in multicast message as its sequence number
- Receive multicast: If Pi receives a multicast from Pj with vector M[1…N] (= Pj[1…N]) in message, buffer it until both:
  1. This message is the next one Pi is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



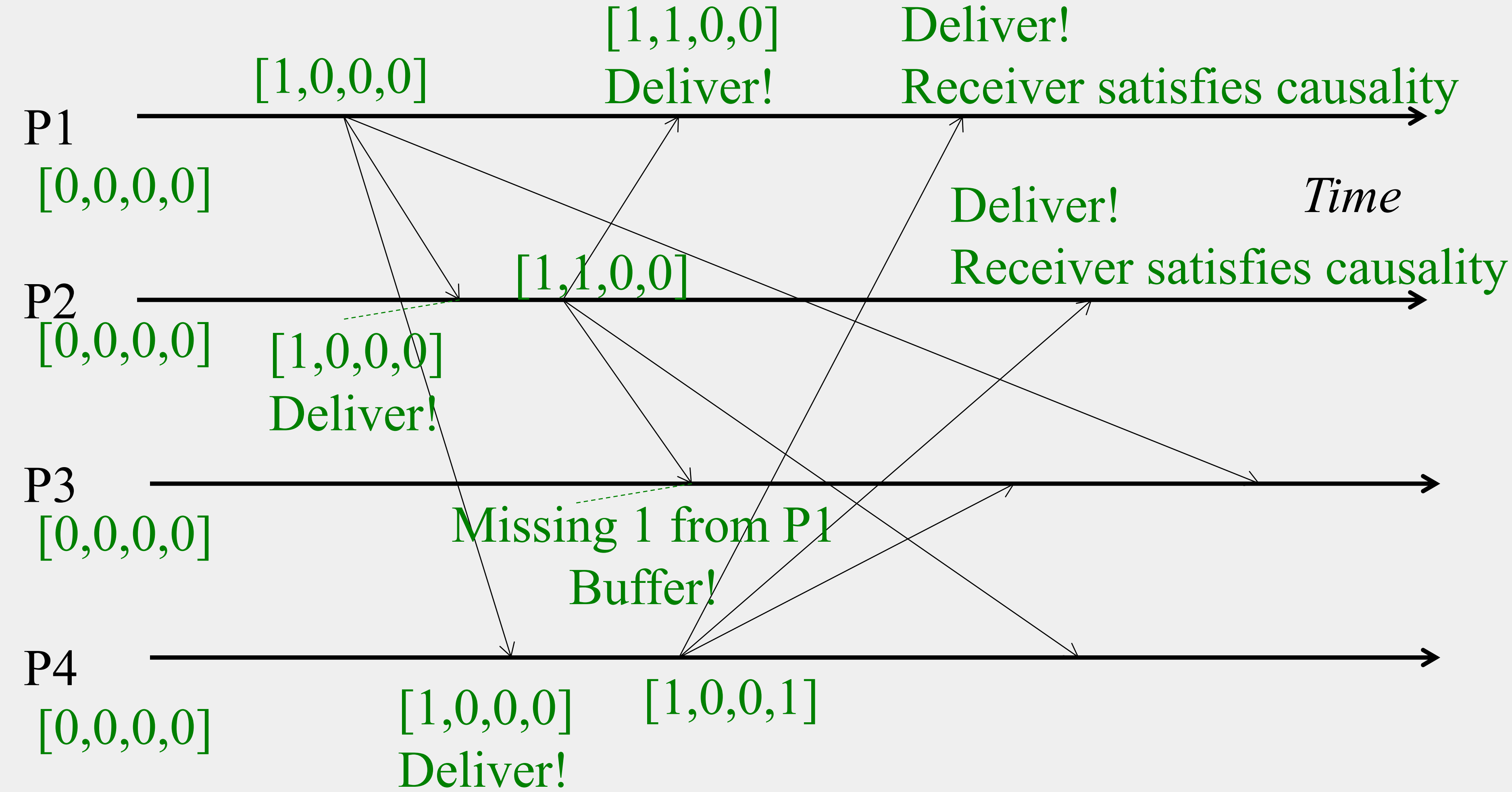
Causal Ordering: Example



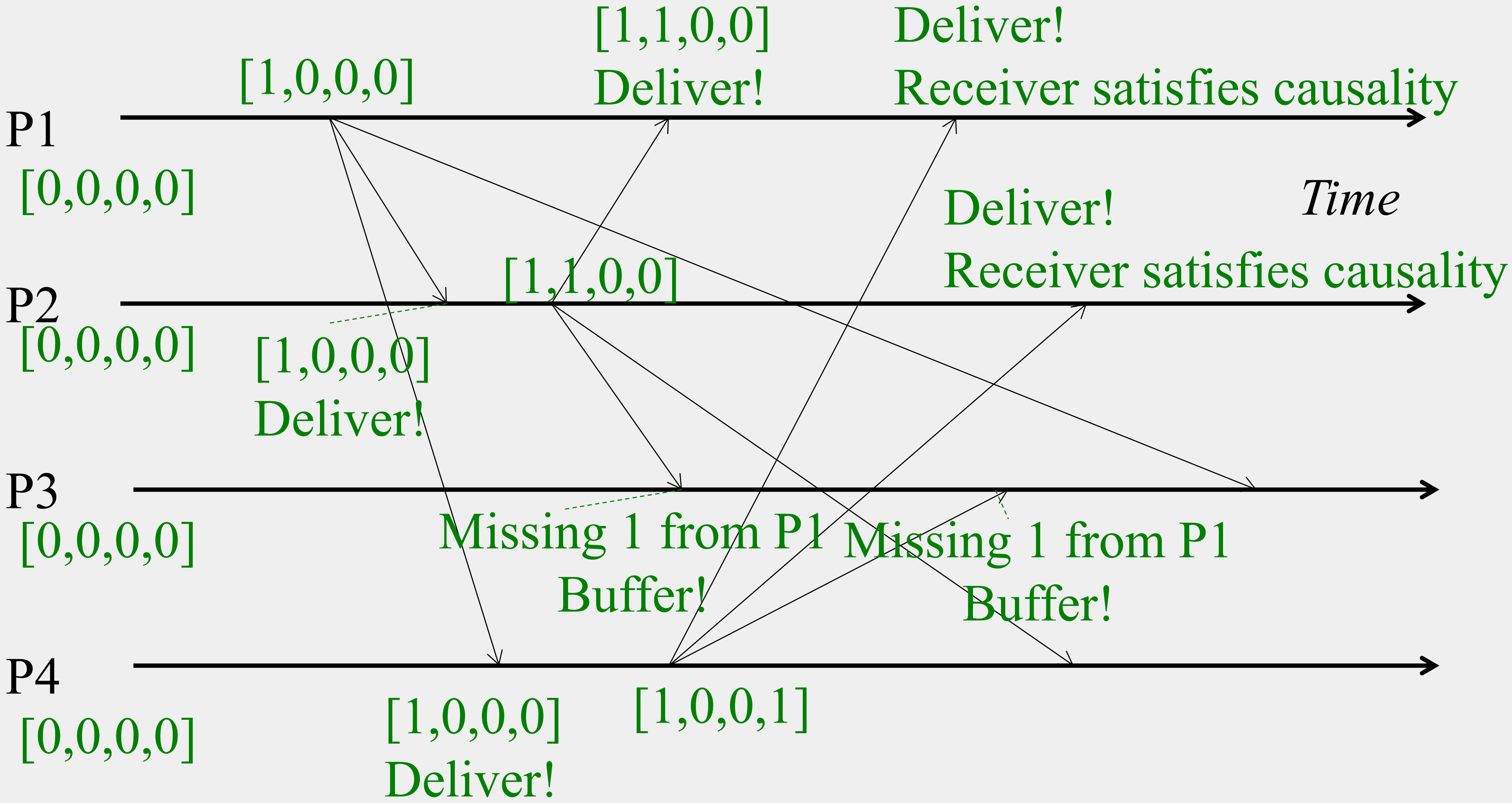
Causal Ordering: Example



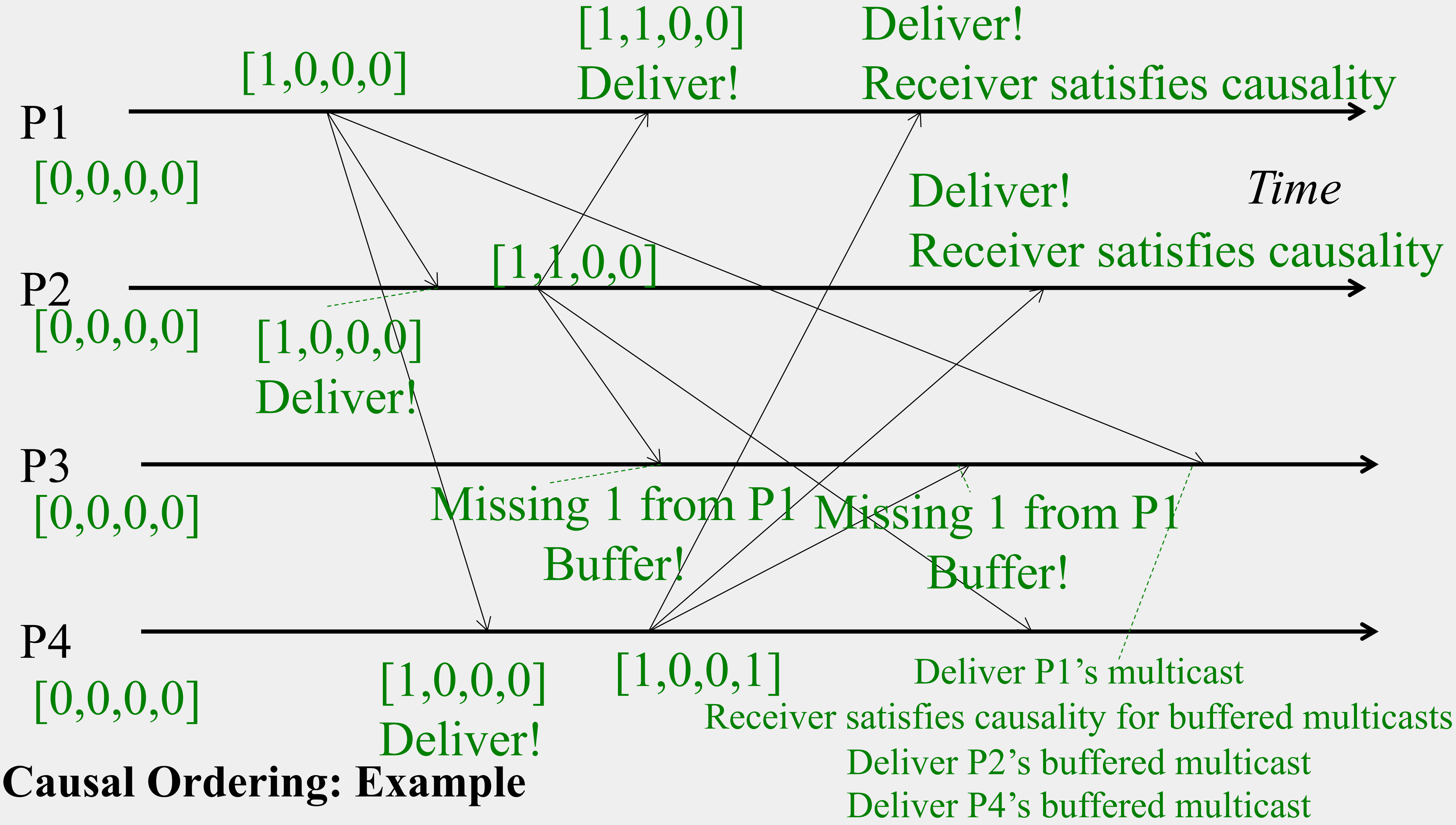
Causal Ordering: Example



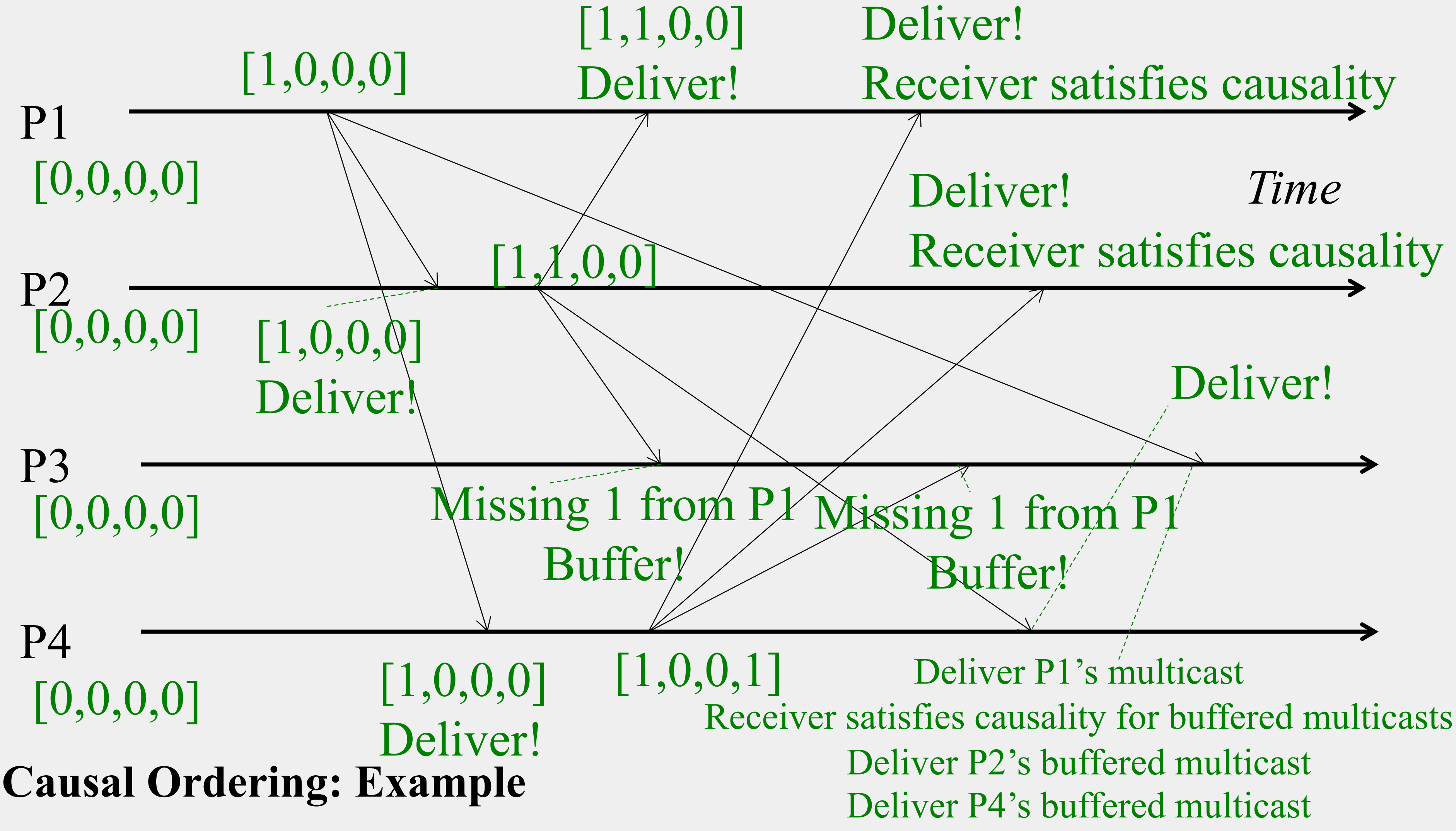
Causal Ordering: Example



Causal Ordering: Example



Causal Ordering: Example



Causal Ordering: Example