Distributed Systems

CS425/ECE428

Instructor: Radhika Mittal

Acknowledgements for some of materials: Indy Gupta and Nikita Borisov

Logistics

- Note about exams on CampusWire:
  - Midterm 1 (Mar 4-6), Midterm 2 (April 12-14), Finals (May 7-15).
  - Reservation via PrairieTest.
    - You can reserve a slot for Midterm 1 starting Feb 19.
  - If you need DRES accommodations, please upload your Letter of Accommodations on the CBTF website.

- HW1 is due next week Friday.
  - You should be able to solve all questions by now.

Today's agenda

- **Multicast**
  - Chapter 15.4

Ordered Multicast

- Three popular flavors implemented by several multicast protocols:
  1. FIFO ordering
  2. Causal ordering
  3. Total ordering

1. FIFO Order

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

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

2. Causal Order

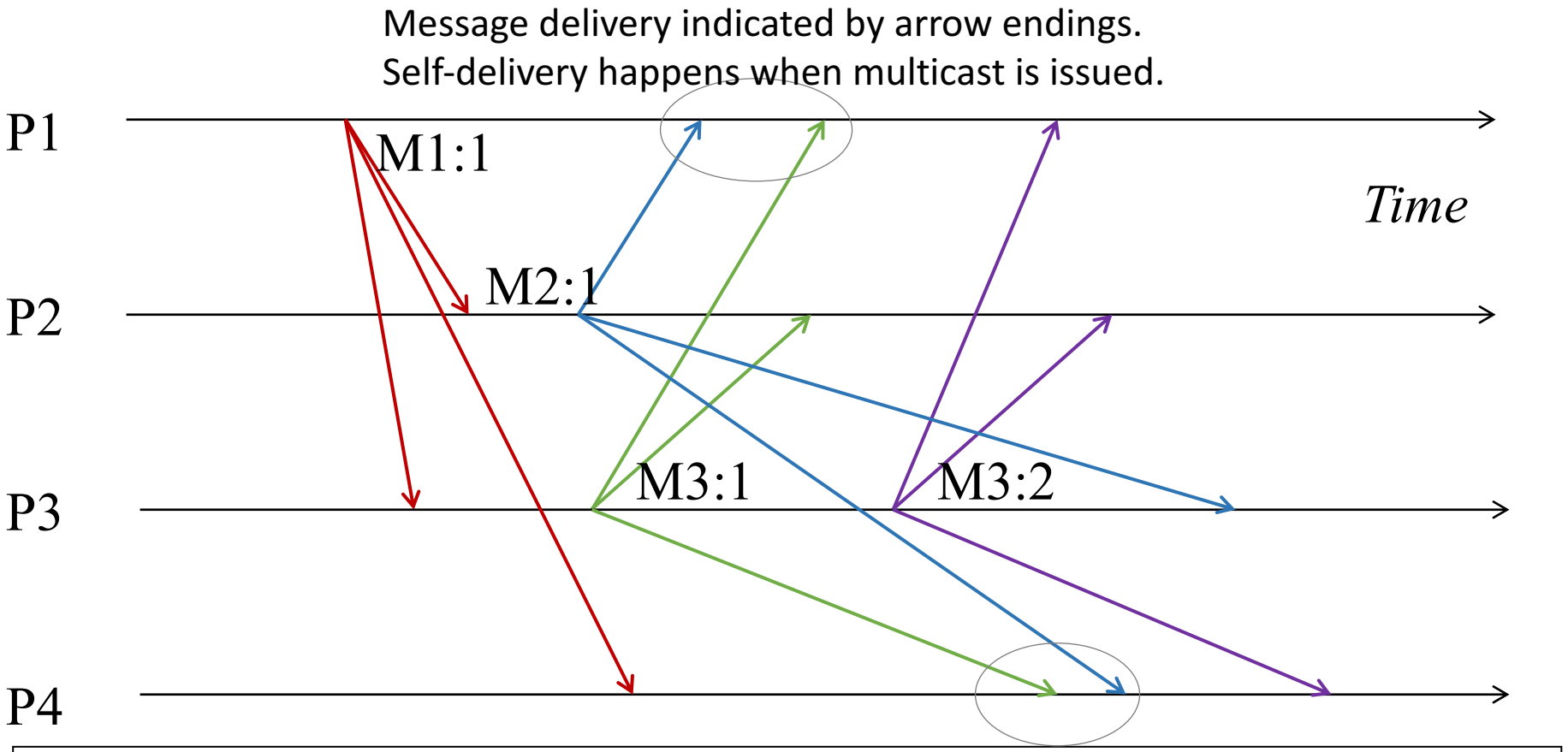
- Multicasts whose send events are causally related, must be delivered in the same causality-obeying order at all receivers.

- More formally
  - *If multicast(g,m) → multicast(g,m') then any correct process that delivers m' will have already delivered m.*
  - → *is Lamport's happens-before*
  - → is induced only by multicast messages in group g, and when they are **delivered** to the application, rather than all network messages.

HB Relationship for Causal Ordering

- HB rules in causal ordered multicast:
  - If ∃ p_i , **e** $\rightarrow_i$ **e'** then **e** $\rightarrow$ **e'**.
    - If ∃ p_i , multicast(*g*,*m*) $\rightarrow_i$ multicast(*g*,*m'*), then multicast(*g*,*m*) $\rightarrow$ multicast(*g*,*m'*)
    - If ∃ p_i , delivery(*m*) $\rightarrow_i$ multicast(*g*,*m'*), then delivery(*m*) $\rightarrow$ multicast(*g*,*m'*)
    - …
  - ~~For any message m, **send(m) → receive(m)**~~
    - For any *multicast* message m, **multicast(*g*,*m*) → delivery(m)**
  - If **e** → **e'** and **e'** → **e"** then **e** → **e"**
    - multicast(*g*,*m*) at p_i → delivery(m) at p_j
    - delivery(*m*) at p_j → multicast(*g*,*m'*) at p_j
    - multicast(*g*,*m*) at p_i → multicast(*g*,*m'*) at p_j
- *Application can only see when messages are "multicast" by the application and "delivered" to the application, and not when they are sent or received by the protocol.*

Causal Order: Example



M3:1 → M3:2, M1:1 → M2:1, M1:1 → M3:1 and so should be delivered in that order at each receiver.

M3:1 and M2:1 are concurrent and thus ok to be delivered in any (and even different) orders at different receivers.

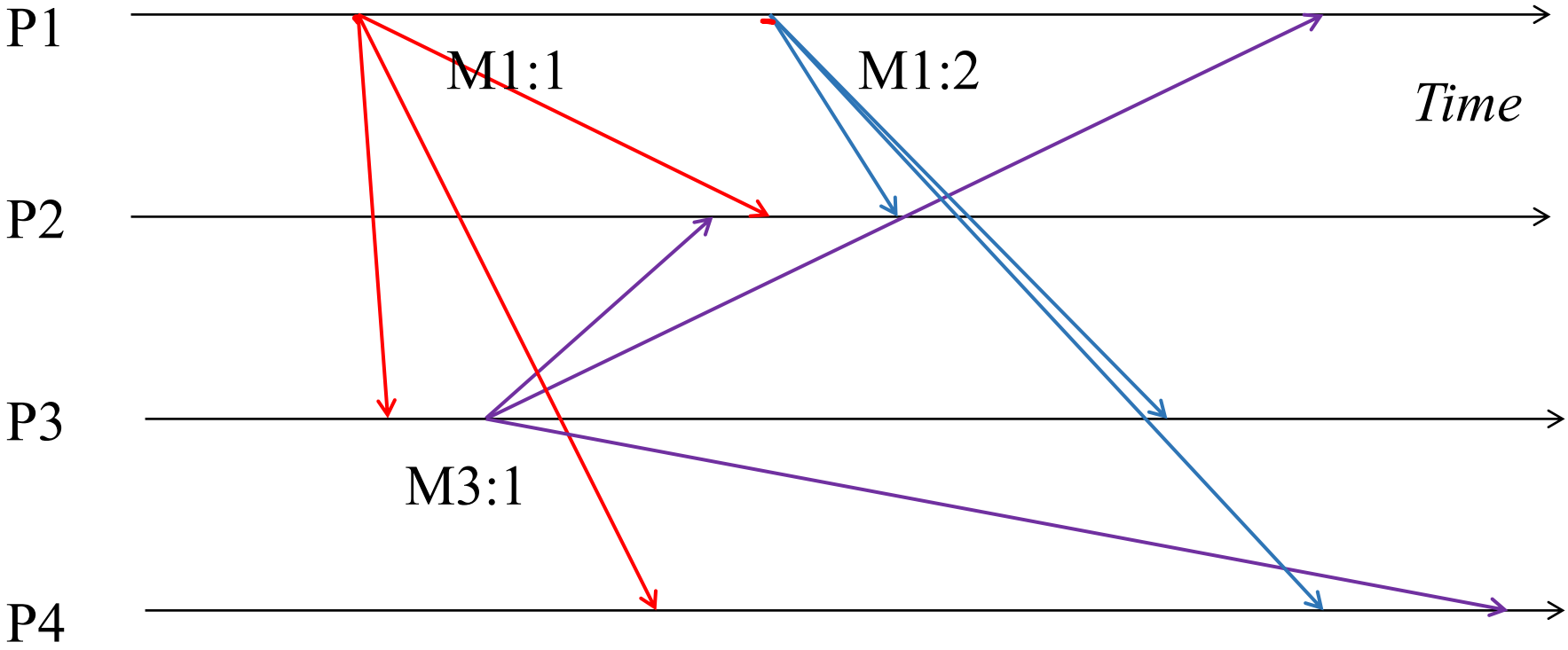
Causal vs FIFO

- Does Causal Ordering imply FIFO Ordering?
  - Yes

- Does FIFO Order imply Causal Order?
  - No

Example

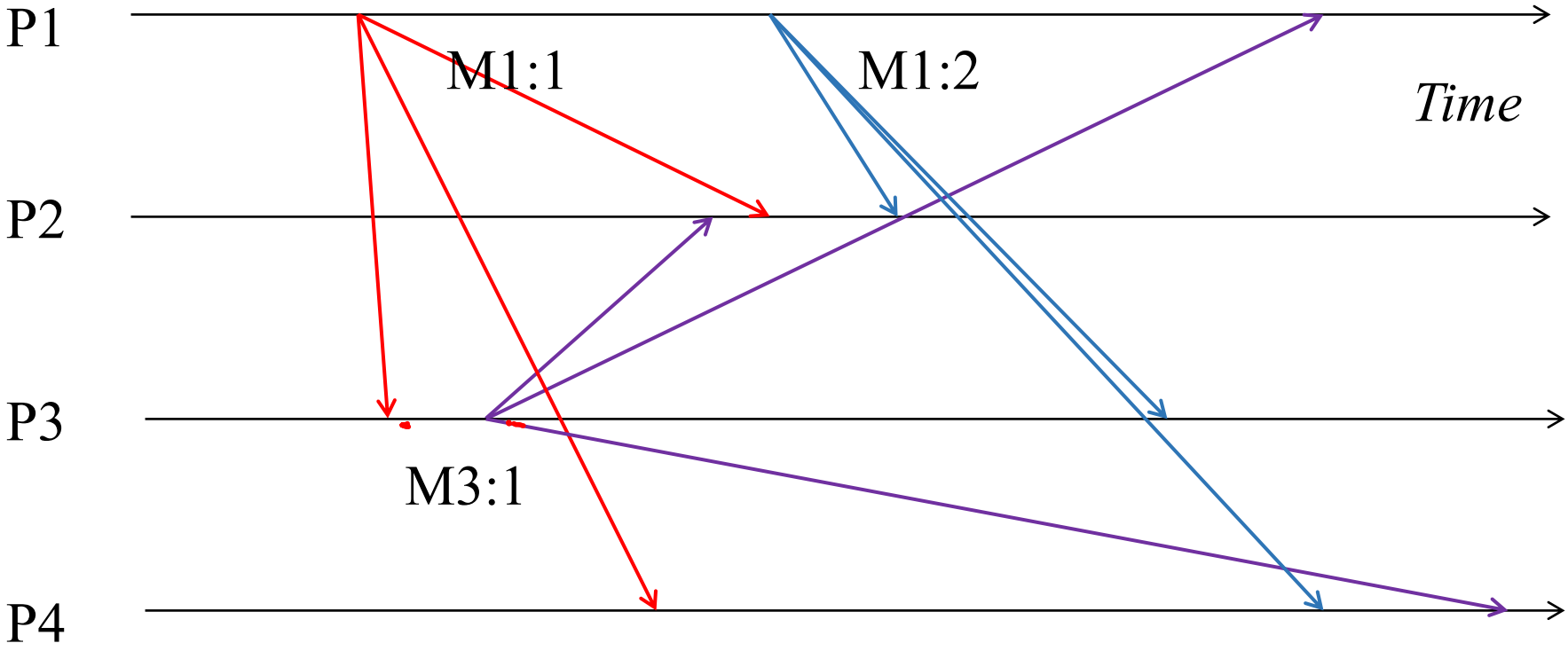
Message delivery indicated by arrow endings.
Self-delivery happens when multicast is issued.



Does this satisfy FIFO order?

Example

Message delivery indicated by arrow endings.
Self-delivery happens when multicast is issued.

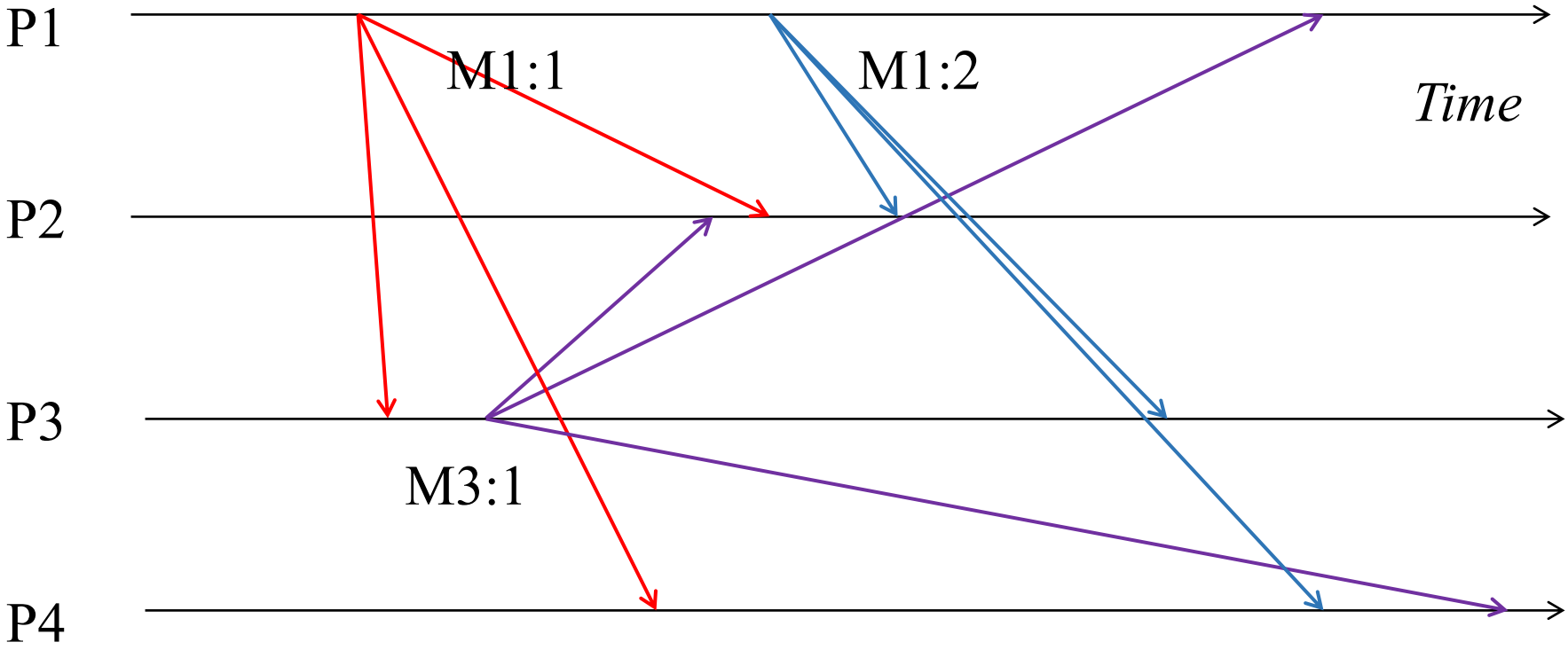


Does this satisfy FIFO order?

Yes

Example

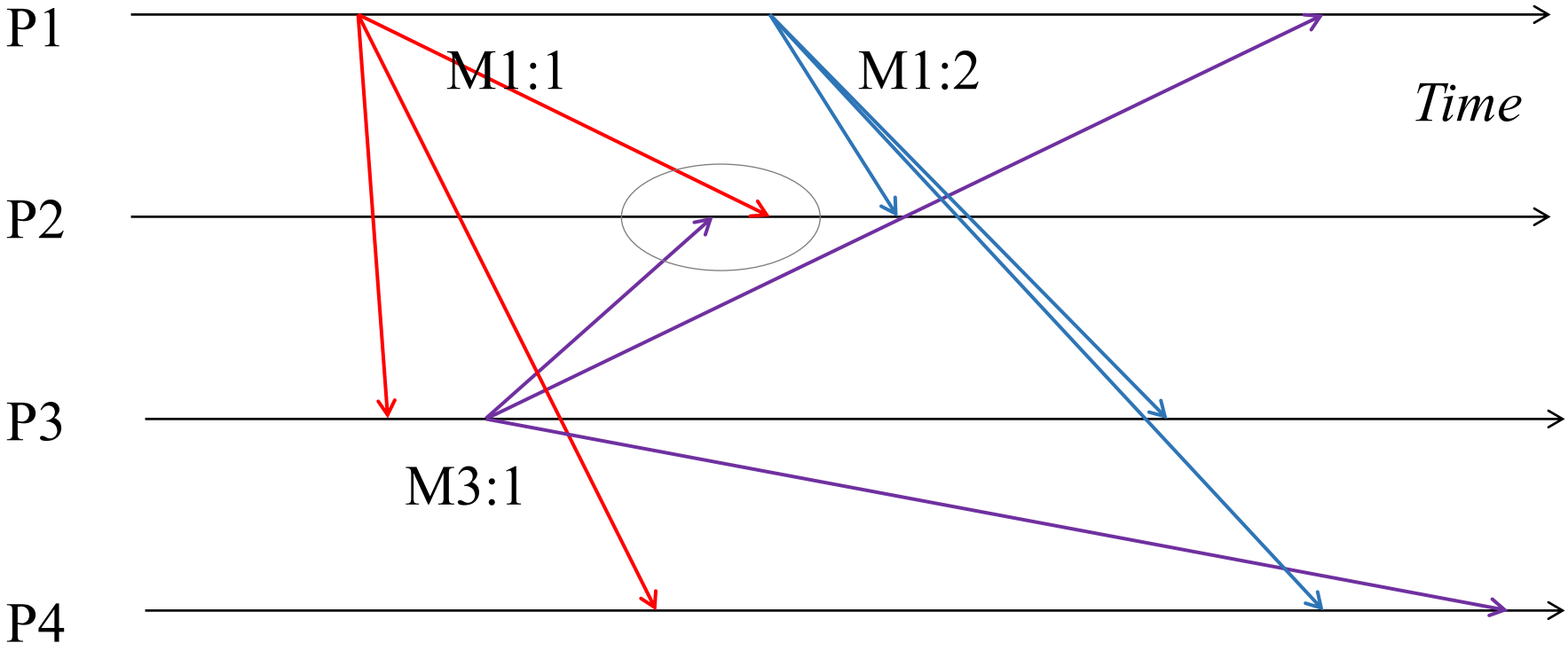
Message delivery indicated by arrow endings.
Self-delivery happens when multicast is issued.



Does this satisfy causal order?

Example

Message delivery indicated by arrow endings.
Self-delivery happens when multicast is issued.

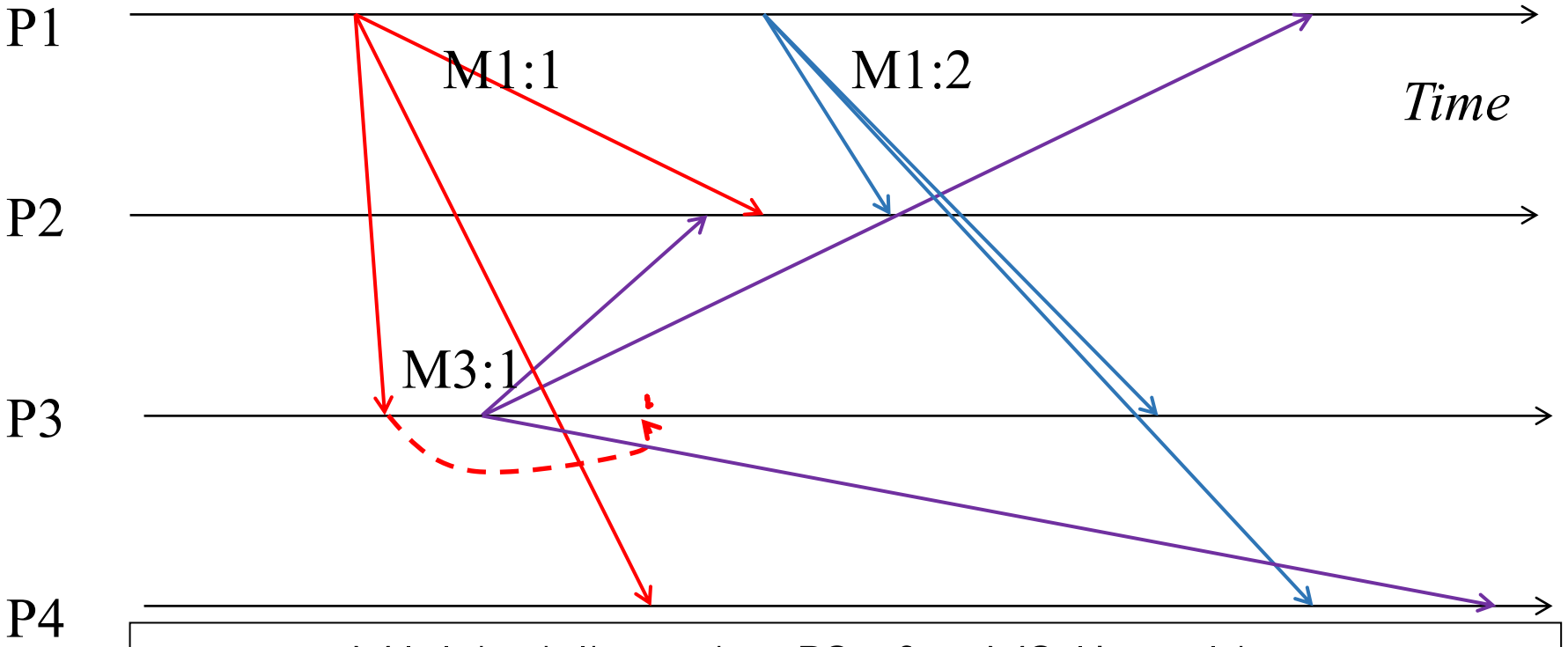


Does this satisfy causal order?

No

Example

Message delivery indicated by (extended) arrow endings.
Self-delivery happens when multicast is issued.



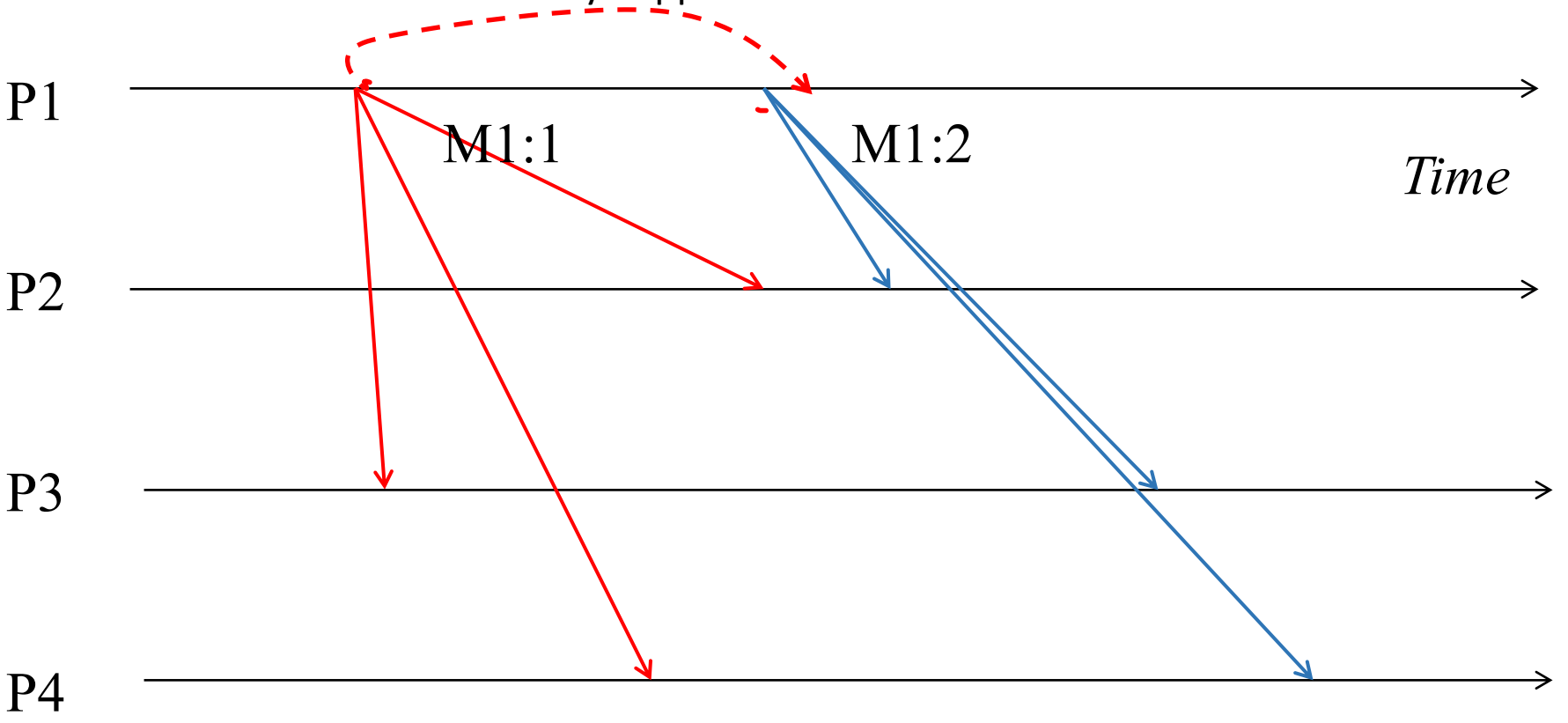
M1:1 is delivered at P3 after M3:1's multicast.

Does this satisfy causal order?

Yes

Example

Message delivery indicated by (extended) arrow endings.
Self-delivery happens when multicast is issued when no extra arrow.

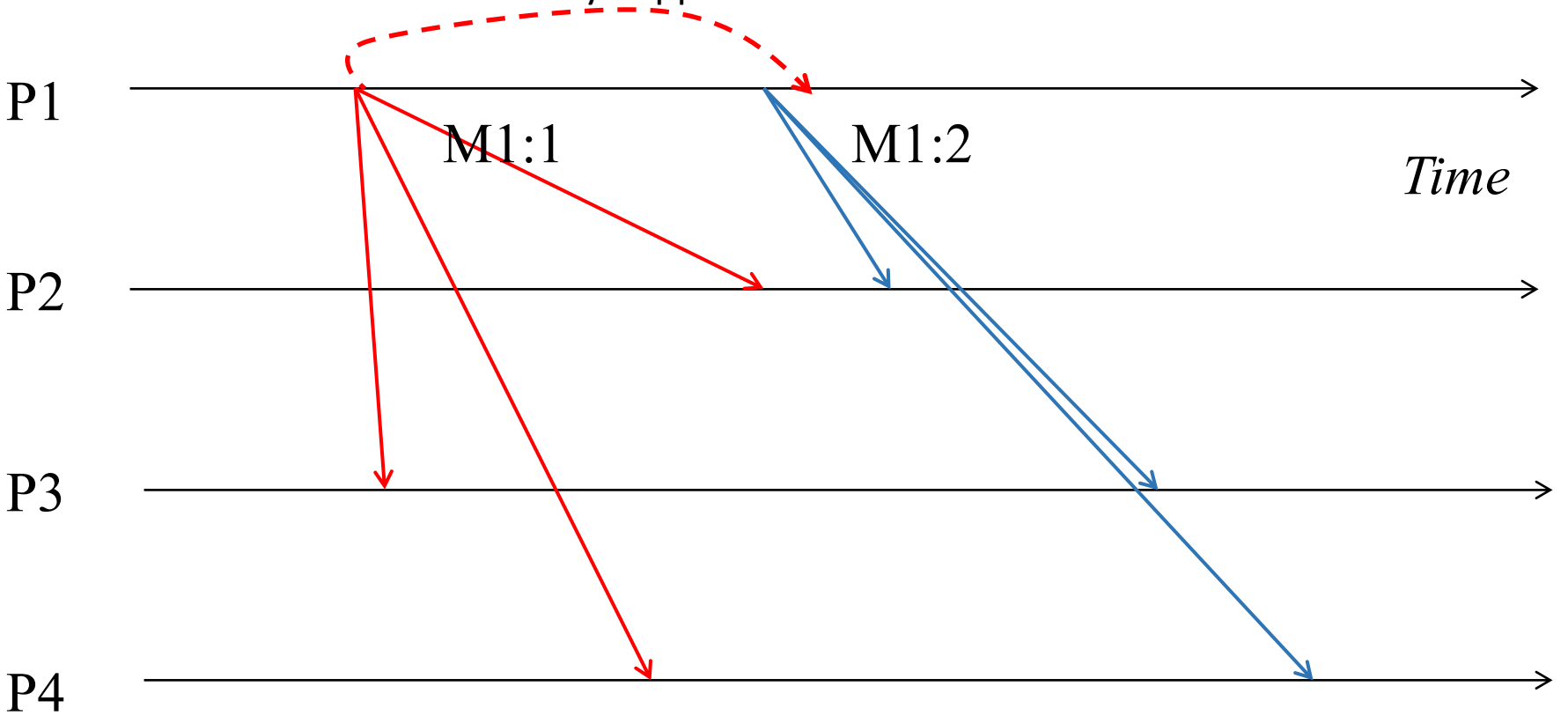


Does this satisfy causal order?

No

Example

Message delivery indicated by (extended) arrow endings.
Self-delivery happens when multicast is issued when no extra arrow.



Does this satisfy FIFO order?

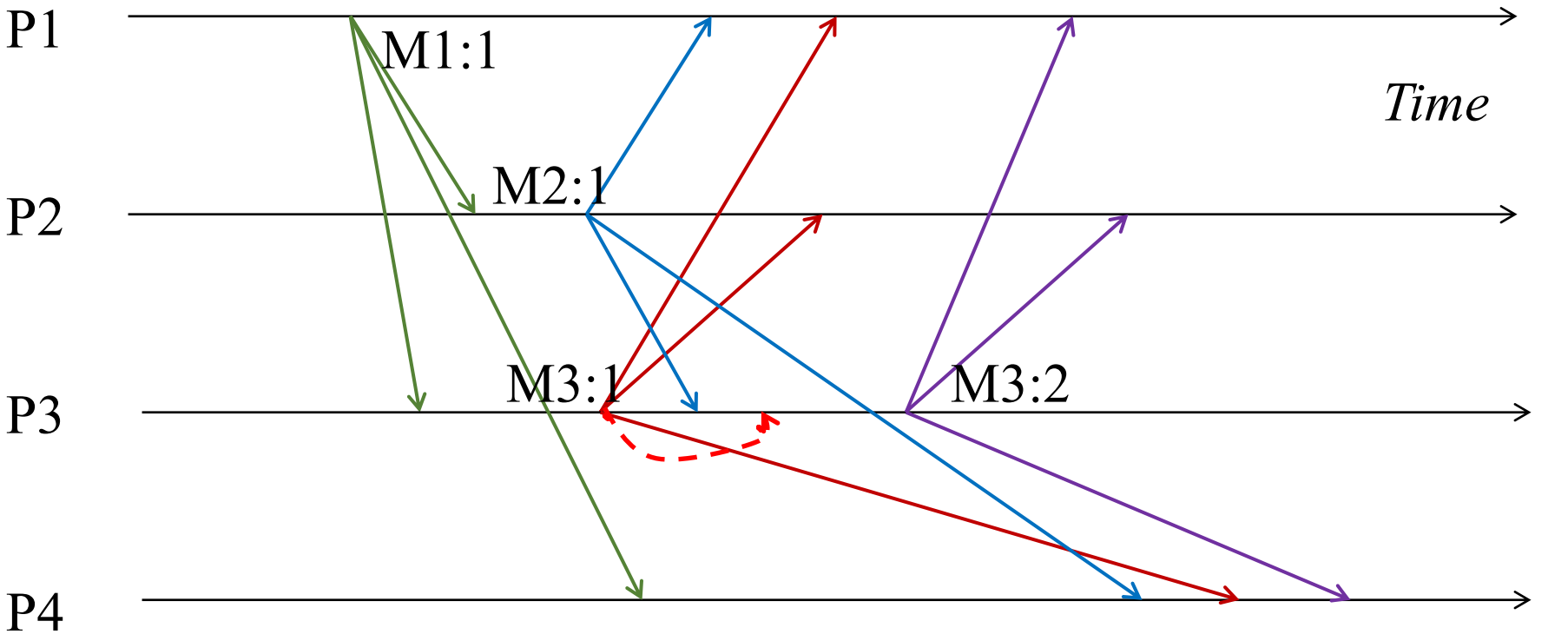
No

3. Total Order

- Ensures all processes deliver all multicasts in the same order.
- Unlike FIFO and causal, this does not pay attention to order of multicast sending.
- Formally
  - If a correct process delivers message *m* before *m'* (independent of the senders), then any other correct process that delivers *m'* will have already delivered *m*.

Total Order: Example

Message delivery indicated by (extended) arrow endings.
Self-delivery happens when multicast is issued (when no extra arrow).



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.

Causal vs Total

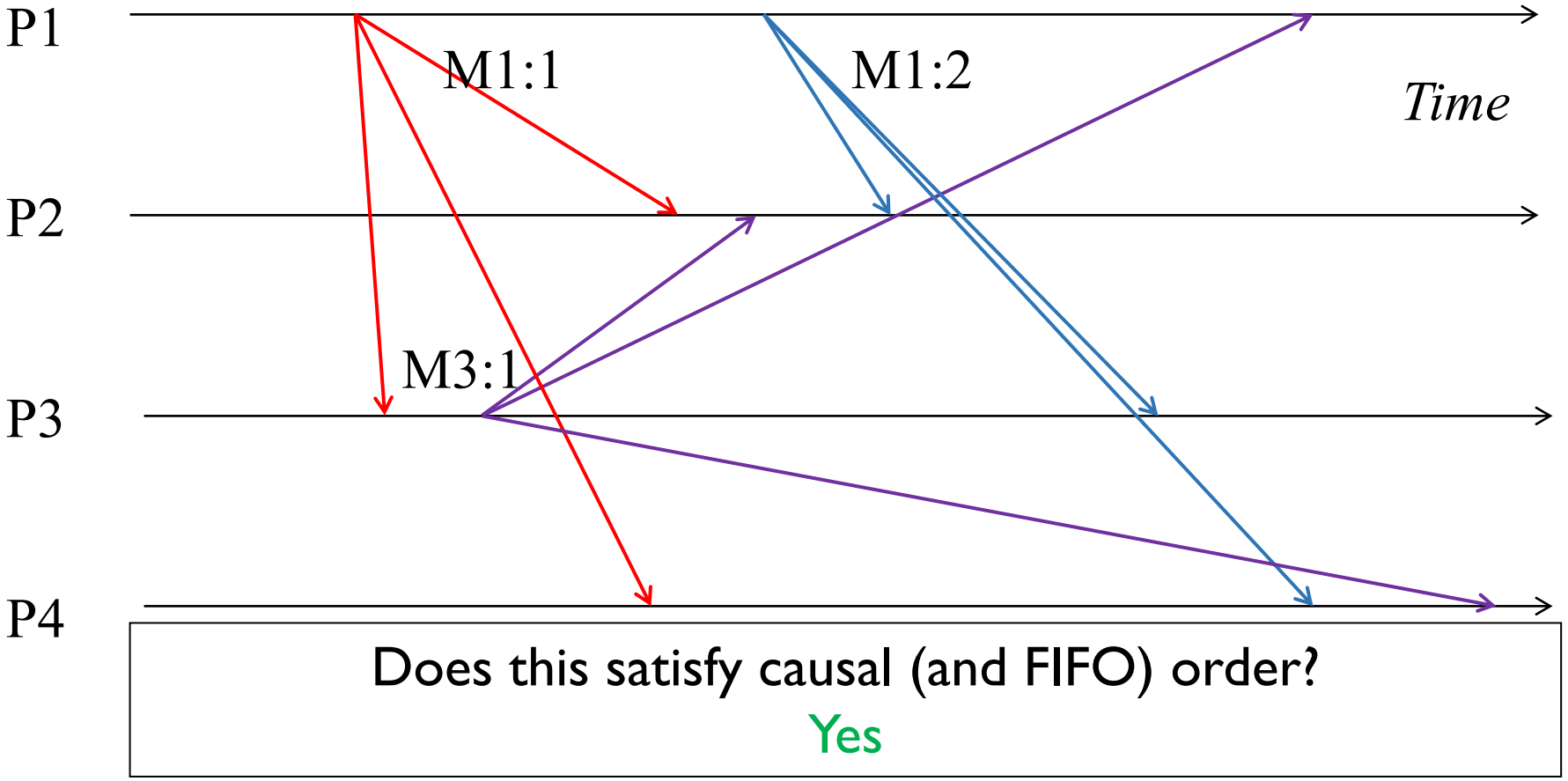
- Total ordering does not imply causal ordering.
- Causal ordering does not imply total ordering.

Hybrid variants

- We can have hybrid ordering protocols:
  - Causal-total hybrid protocol satisfies both Causal and total orders.

Example

Message delivery indicated by (extended) arrow endings.
Self-delivery happens when multicast is issued (when no extra arrow).

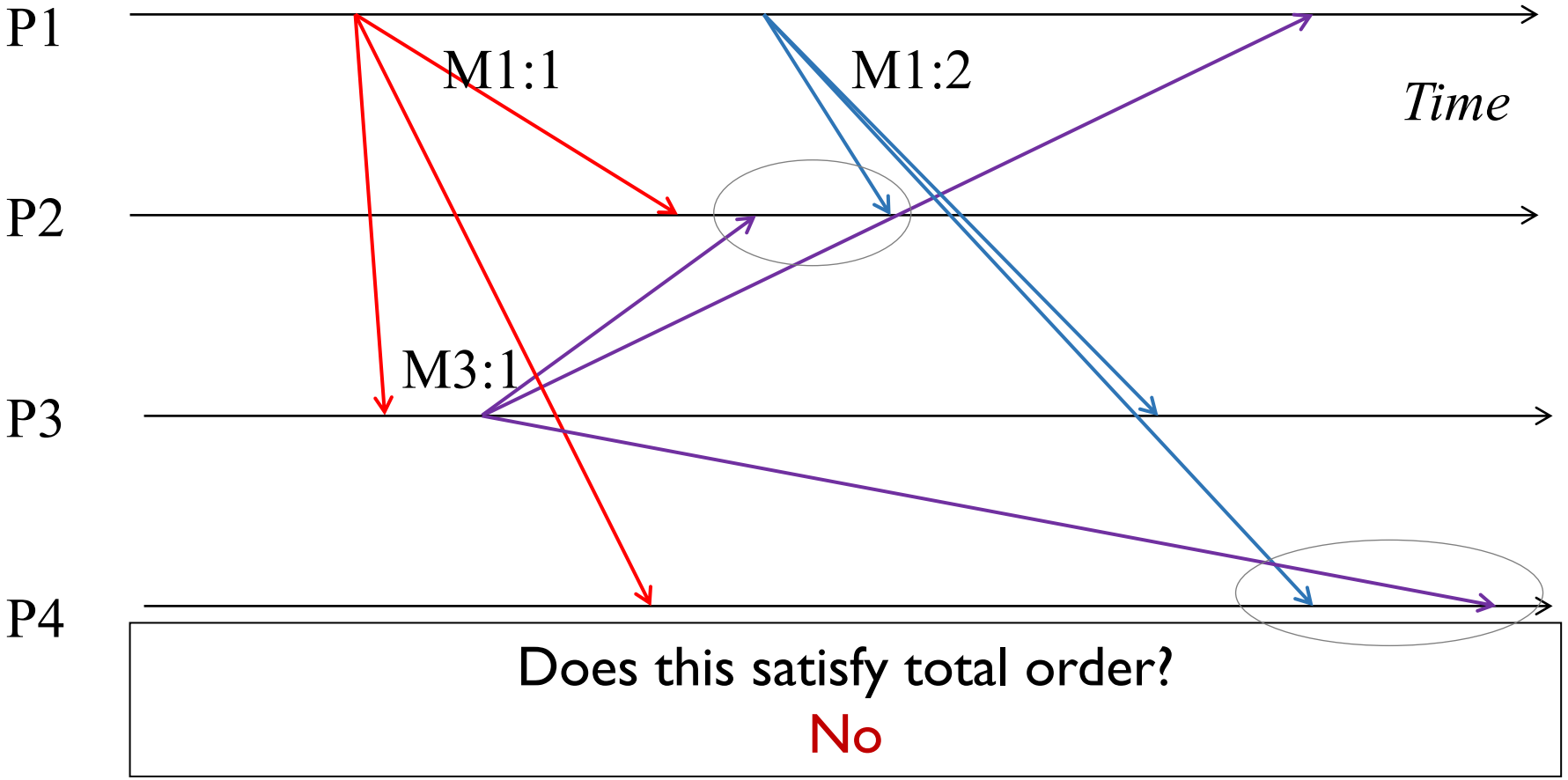


Does this satisfy causal (and FIFO) order?

Yes

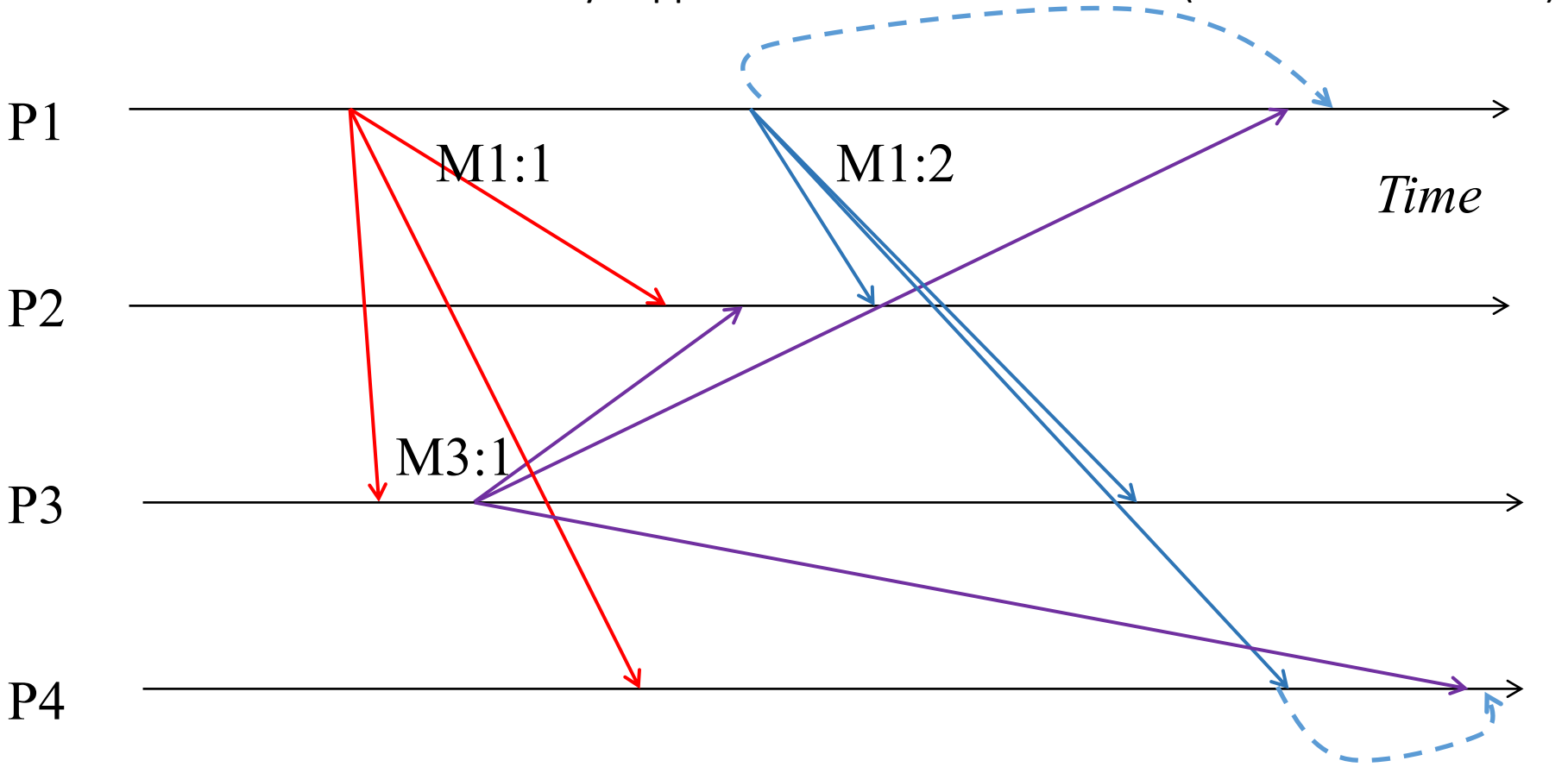
Example

Message delivery indicated by (extended) arrow endings.
Self-delivery happens when multicast is issued (when no extra arrow).



Example

Message delivery indicated by (extended) arrow endings.
Self-delivery happens when multicast is issued (when no extra arrow).



Does this satisfy total order?

Yes

Ordered Multicast

- **FIFO ordering:** If a correct process issues multicast(*g*,*m*) and then multicast(*g*,*m'*), then every correct process that delivers *m'* will have already delivered m.

- **Causal ordering:** If multicast(*g*,*m*) → multicast(*g*,*m'*) then any correct process that delivers *m'* will have already delivered *m*.
  - Note that → counts messages **delivered** to the application, rather than all network messages.

- **Total ordering**: If a correct process delivers message *m* before *m'* (independent of the senders), then any other correct process that delivers *m'* will have already delivered *m*.

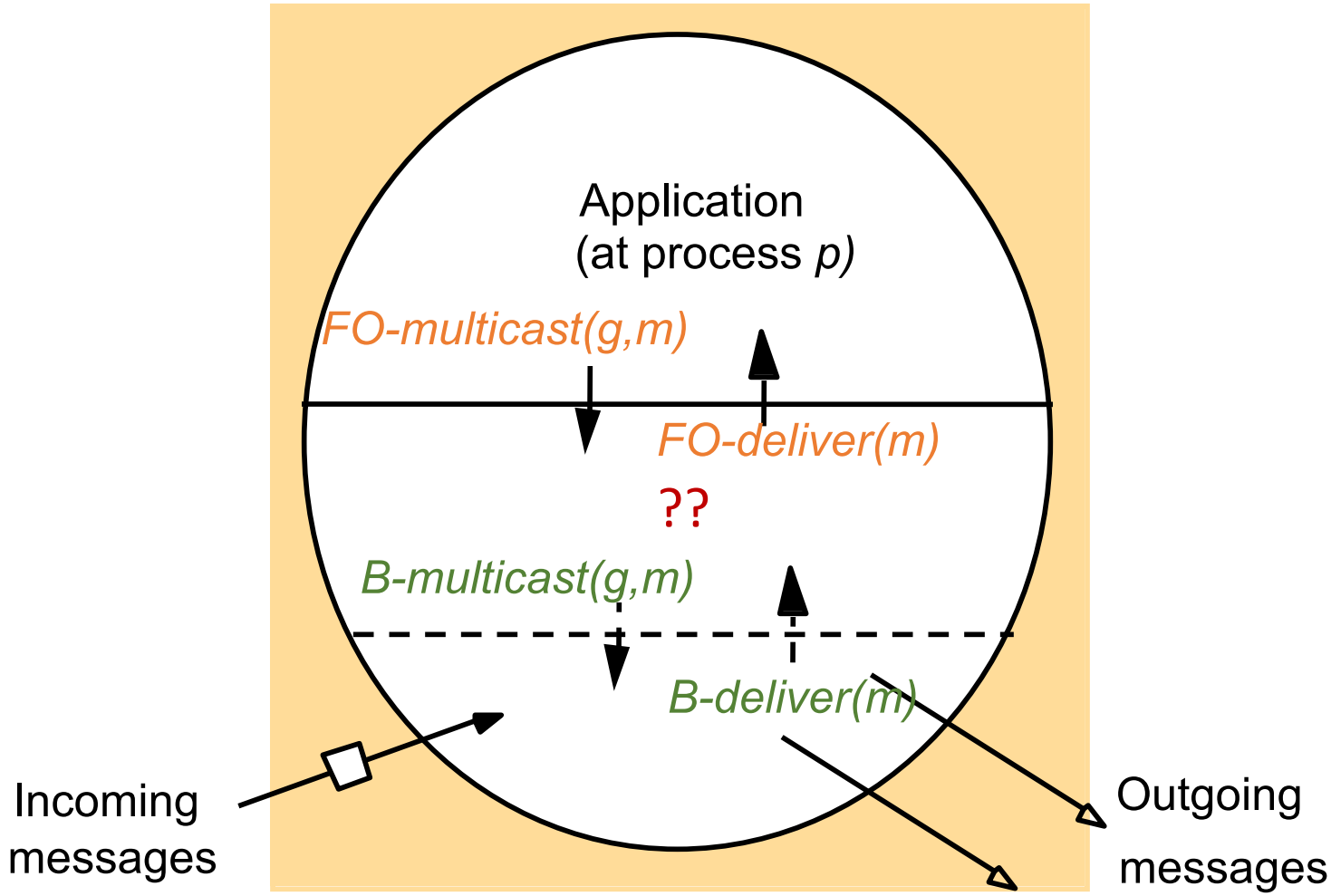
Next Question

How do we implement ordered multicast?

Ordered Multicast

- FIFO ordering
  - If a correct process issues multicast(*g*,*m*) and then multicast(*g*,*m'*), then every correct process that delivers *m'* will have already delivered m.
- Causal ordering
  - If multicast(*g*,*m*) → multicast(*g*,*m'*) then any correct process that delivers *m'* will have already delivered *m*.
  - Note that → counts messages **delivered** to the application, rather than all network messages.
- Total ordering
  - If a correct process delivers message *m* before *m'* (independent of the senders), then any other correct process that delivers *m'* will have already delivered *m*.

Implementing FIFO order multicast



Implementing FIFO order multicast

- Each receiver maintains a per-sender sequence number
  - 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*

Implementing FIFO order multicast

- On FO-multicast(g,m) at process P*j*:

  set P*j*[*j*] = P*j*[*j*] + 1

  piggyback P*j*[*j*] with m as its sequence number.

  B-multicast(g,{m, P*j*[*j*]})

- On B-deliver({m, S}) at Pi from Pj: *If Pi receives a multicast from Pj with sequence number S in message*

  if (S == P*i*[*j*] + 1) then

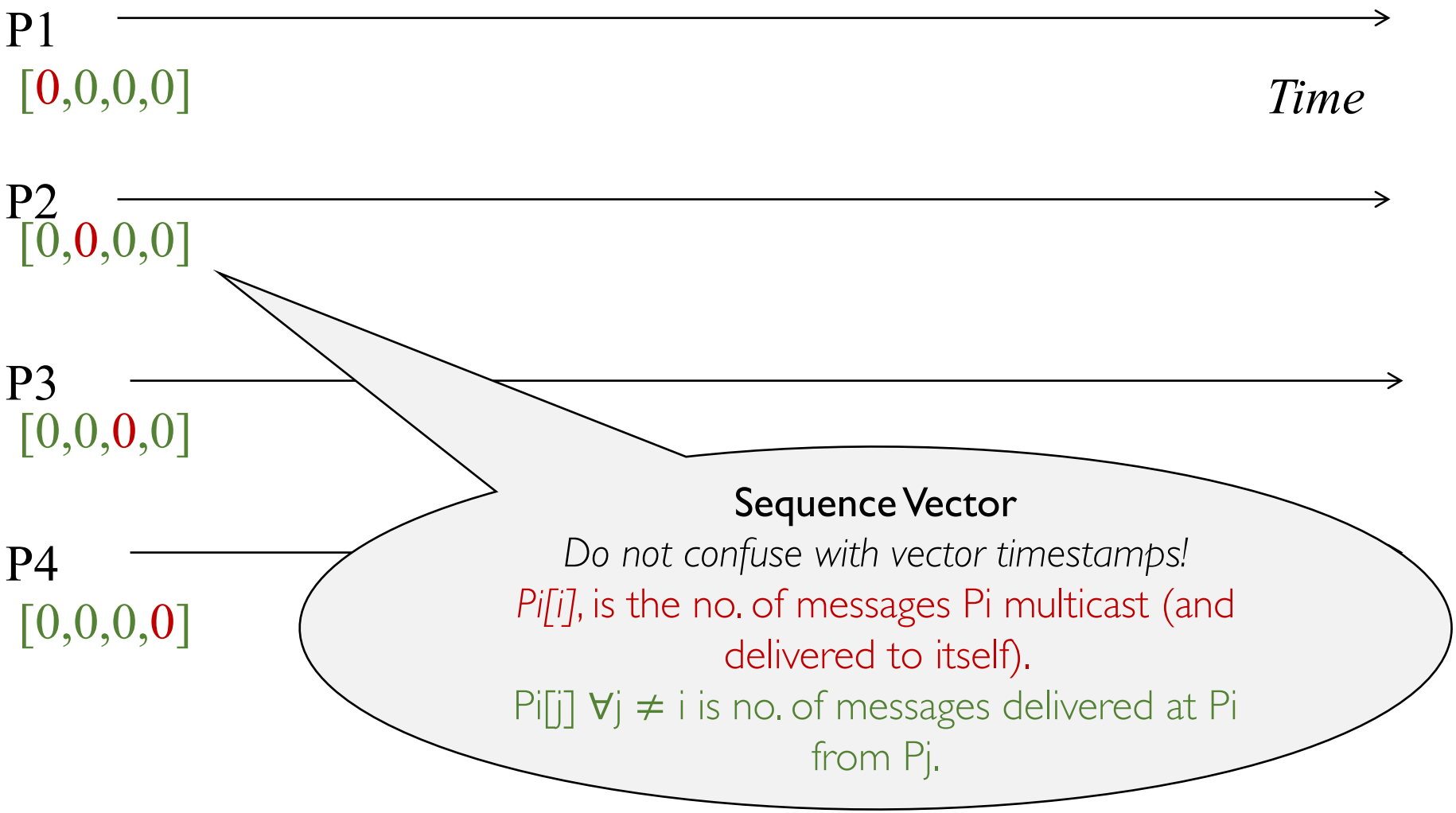
  FO-deliver(m) to application

  set P*i*[*j*] = P*i*[*j*] + 1

  else buffer this multicast until above condition is true

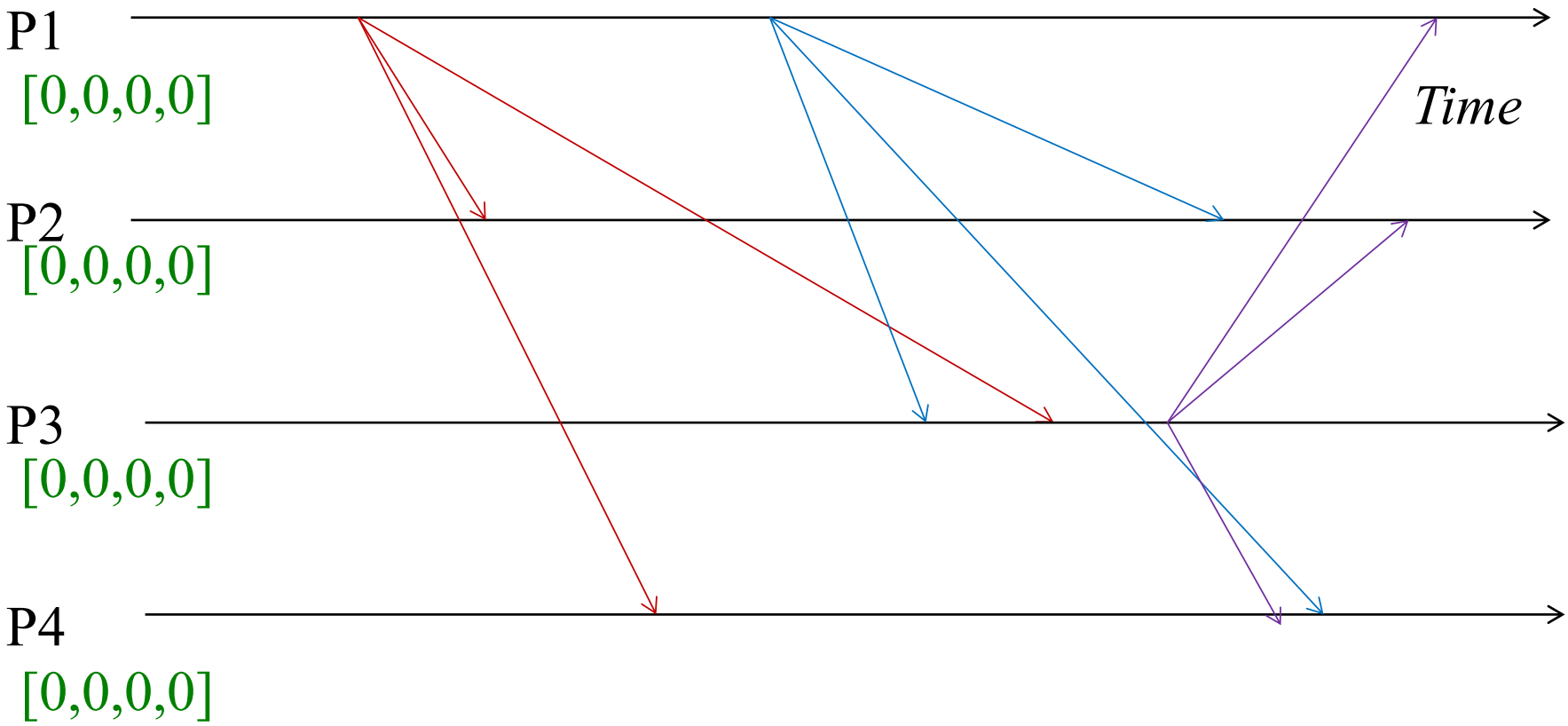
FIFO order multicast execution

FIFO order multicast execution



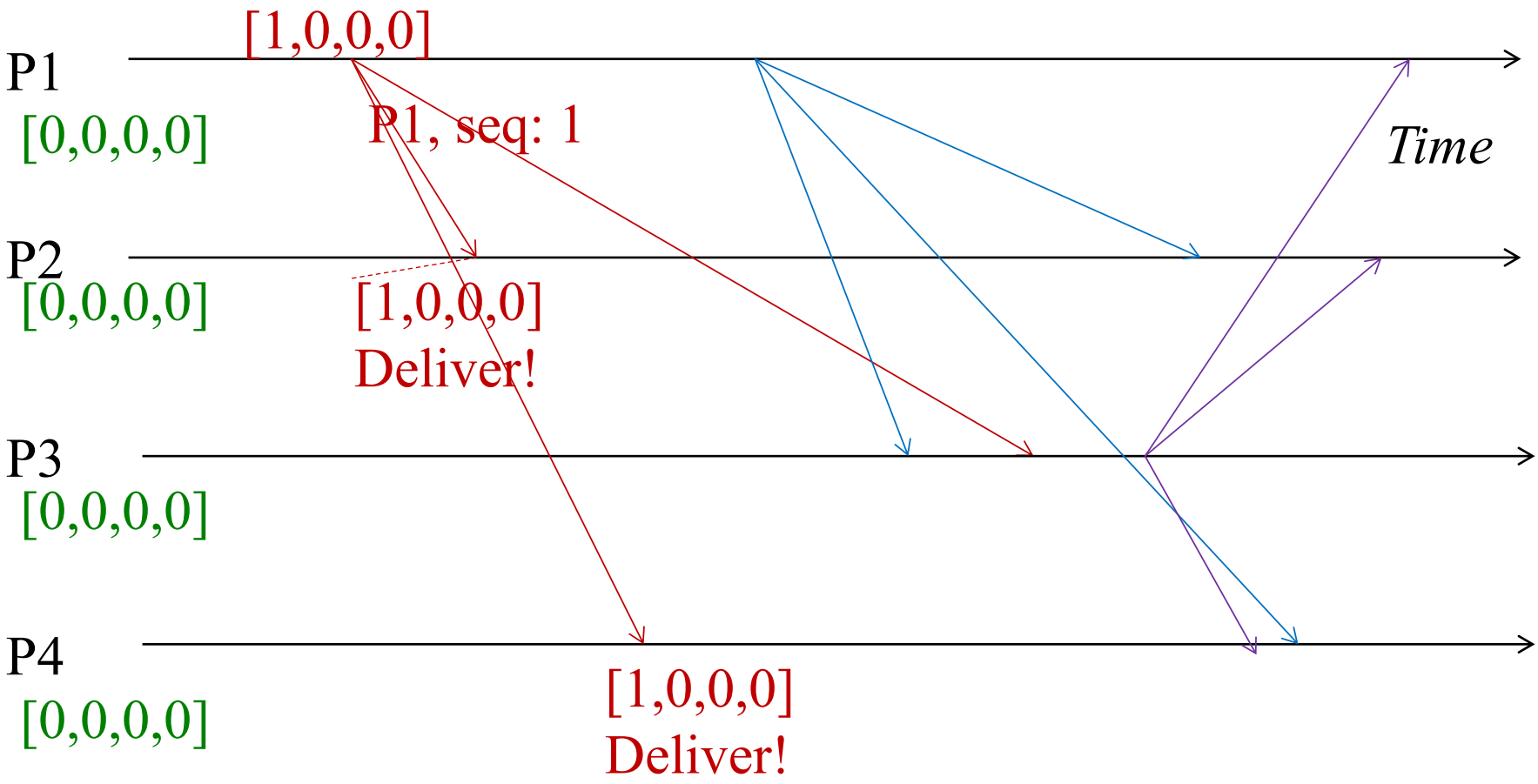
FIFO order multicast execution

FIFO order multicast execution

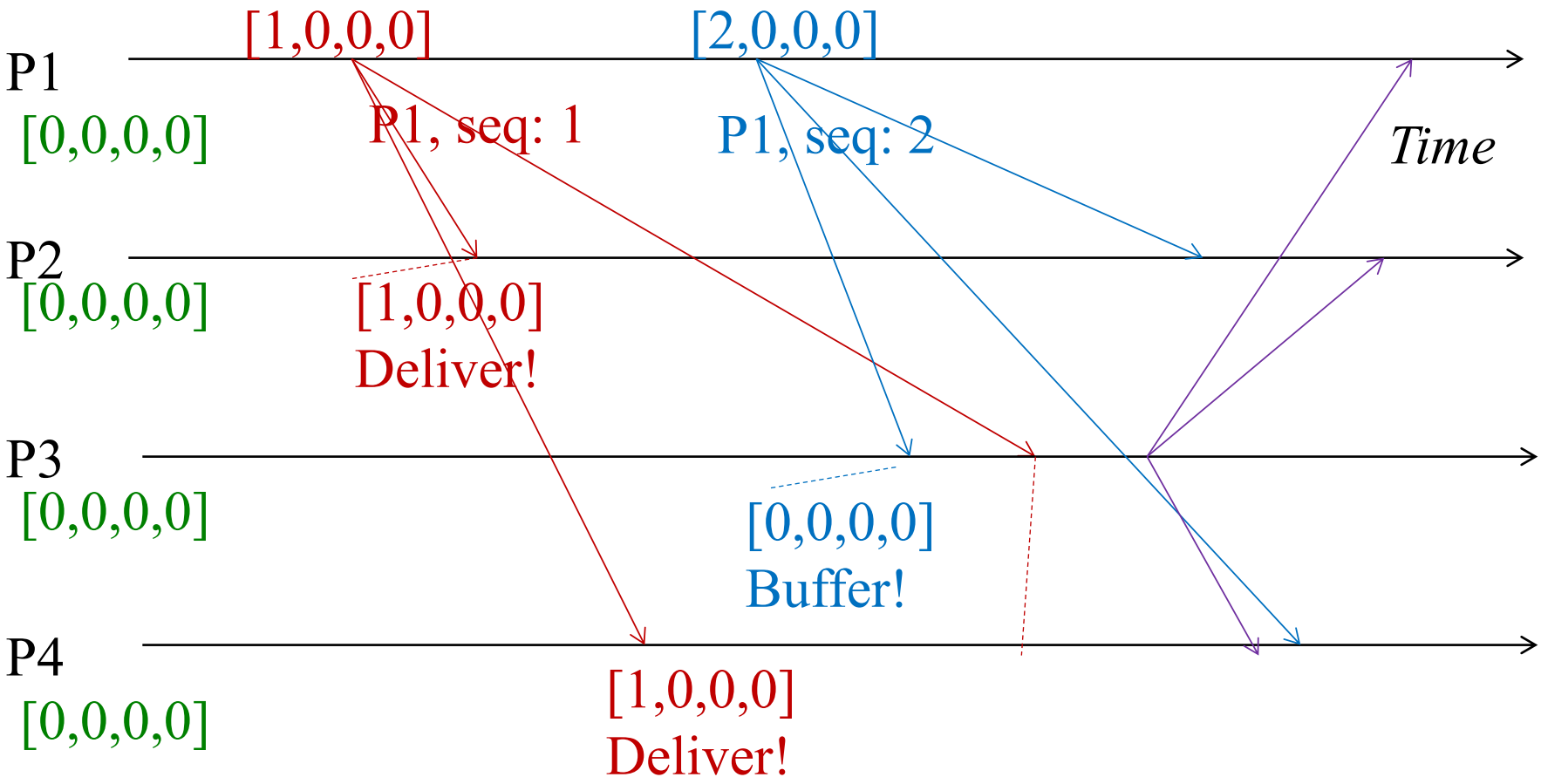


Self-deliveries omitted for simplicity.

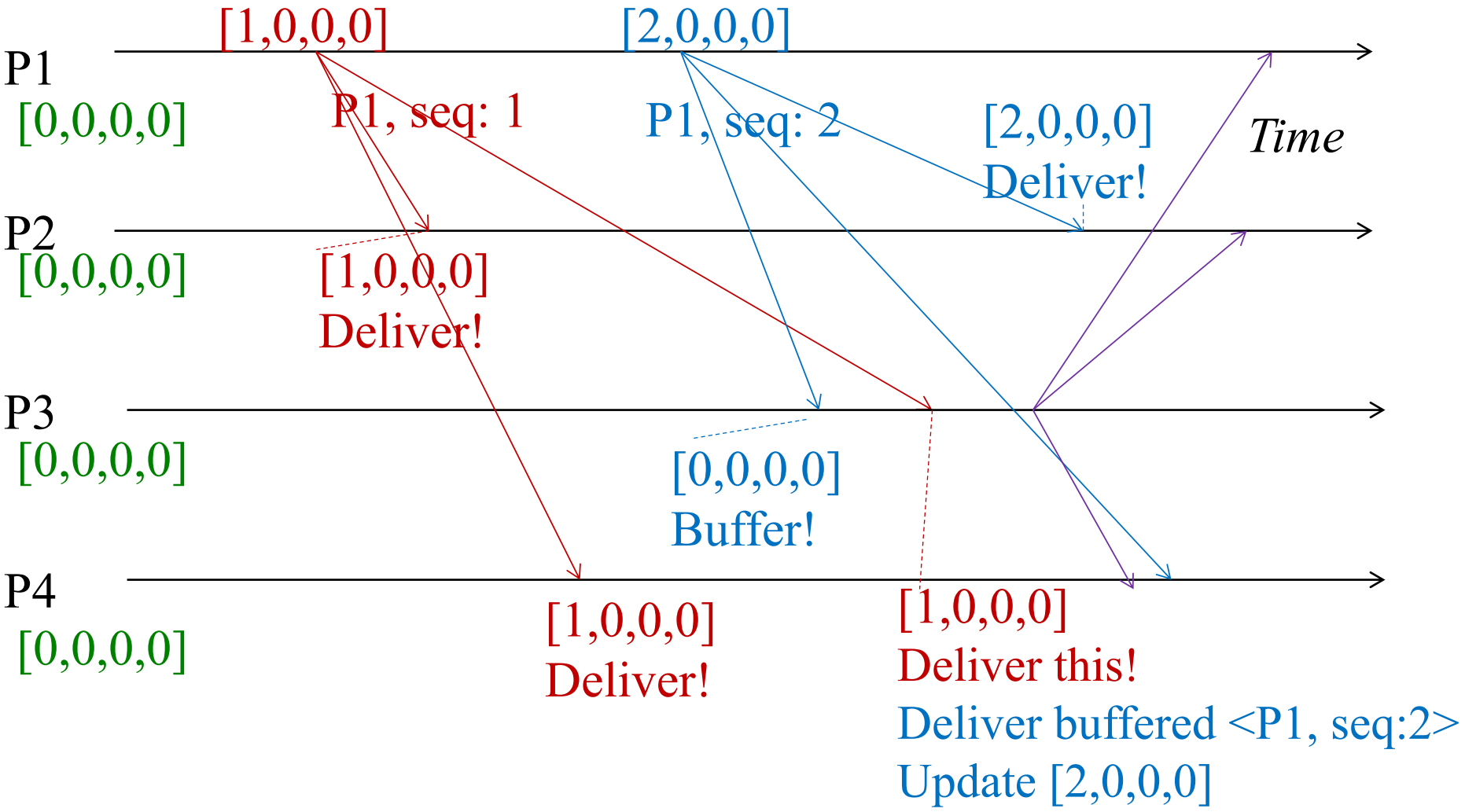
FIFO order multicast execution



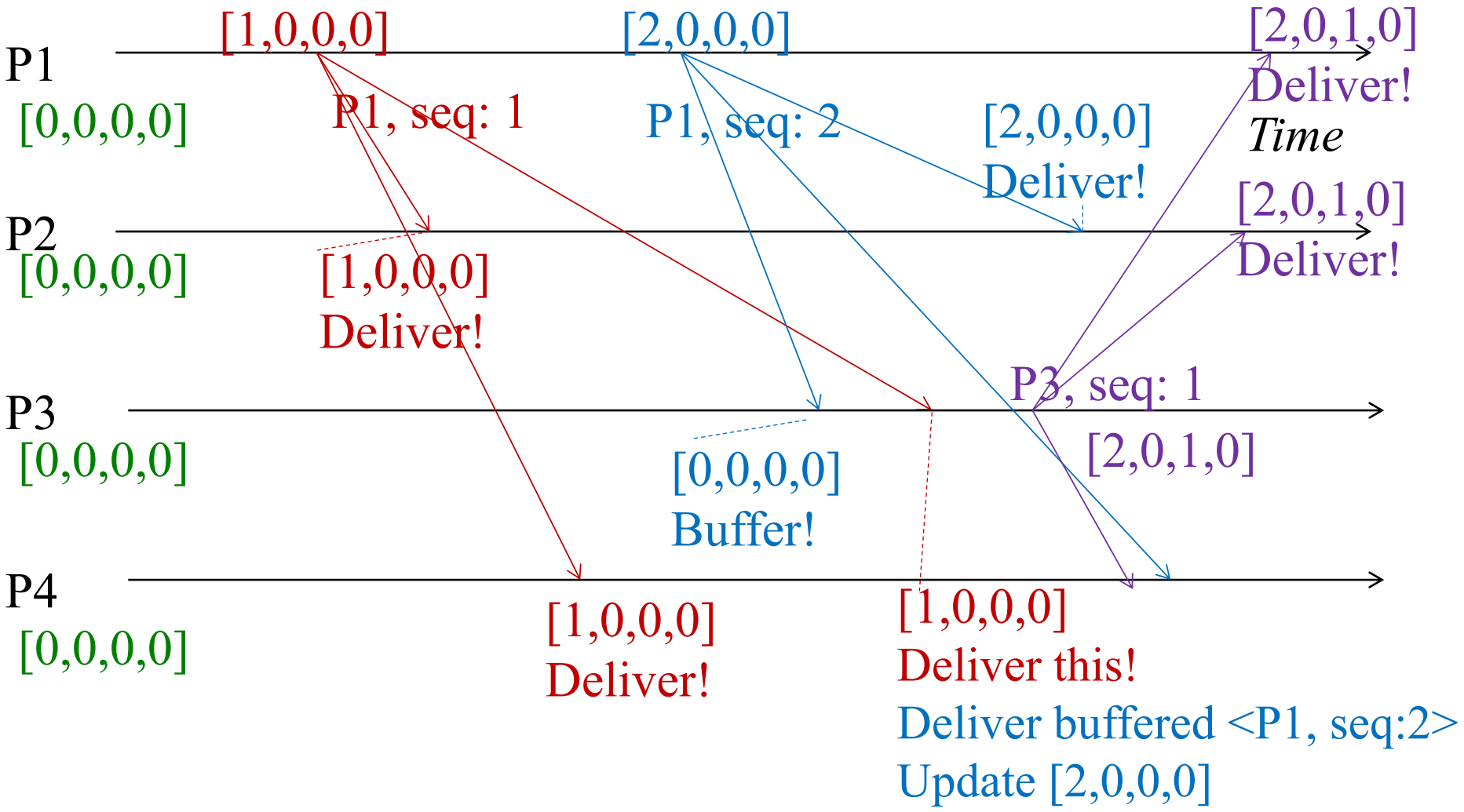
FIFO order multicast execution



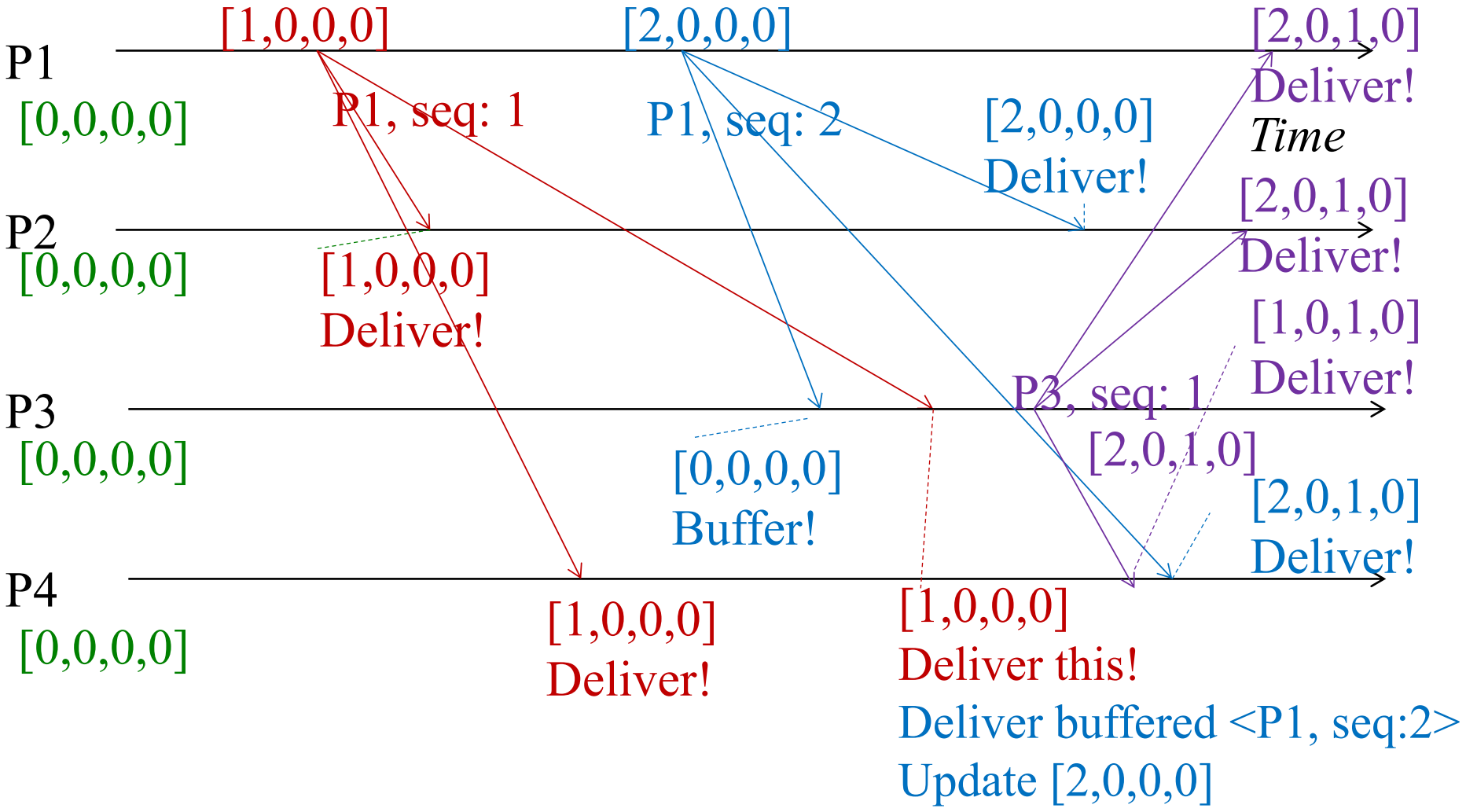
FIFO order multicast execution



FIFO order multicast execution



FIFO order multicast execution



Implementing FIFO order multicast

- On FO-multicast(g,m) at process P*j*:

  set P*j*[*j*] = P*j*[*j*] + 1

  piggyback P*j*[*j*] with m as its sequence number.

  B-multicast(g, {m, P*j*[*j*]})

- On B-deliver({m, S}) at Pi from Pj: *If Pi receives a multicast from Pj with sequence number S in message*

  if (S == P*i*[*j*] + 1) then

  FO-deliver(m) to application

  set P*i*[*j*] = P*i*[*j*] + 1

  else buffer this multicast until above condition is true

Implementing FIFO reliable multicast

- On FO-multicast(g,m) at process P*j*:

  set P*j*[*j*] = P*j*[*j*] + 1

  piggyback P*j*[*j*] with m as its sequence number.

  R-multicast(g,{m, P*j*[*j*]})

- On **R-deliver({m, S})** at Pi from Pj: *If Pi receives a multicast from Pj with sequence number S in message*

  if (S == P*i*[*j*] + 1) then

  FO-deliver(m) to application

  set P*i*[*j*] = P*i*[*j*] + 1

  else buffer this multicast until above condition is true

Ordered Multicast

- **FIFO ordering:** If a correct process issues multicast(*g*,*m*) and then multicast(*g*,*m'*), then every correct process that delivers *m'* will have already delivered m.

- **Causal ordering:** If multicast(*g*,*m*) → multicast(*g*,*m'*) then any correct process that delivers *m'* will have already delivered *m*.
  - Note that → counts messages **delivered** to the application, rather than all network messages.

- **Total ordering**: If a correct process delivers message *m* before *m'* (independent of the senders), then any other correct process that delivers *m'* will have already delivered *m*.

Implementing total order multicast

- Basic idea:
  - Same sequence number counter across different processes.
  - Instead of different sequence number counter for each process.

- Two types of approach
  - Using a centralized sequencer
  - A decentralized mechanism (ISIS)

Implementing total order multicast

- Basic idea:
  - Same sequence number counter across different processes.
  - Instead of different sequence number counter for each process.

- Two types of approach
  - **Using a centralized sequencer**
  - A decentralized mechanism (ISIS)

Sequencer based total ordering

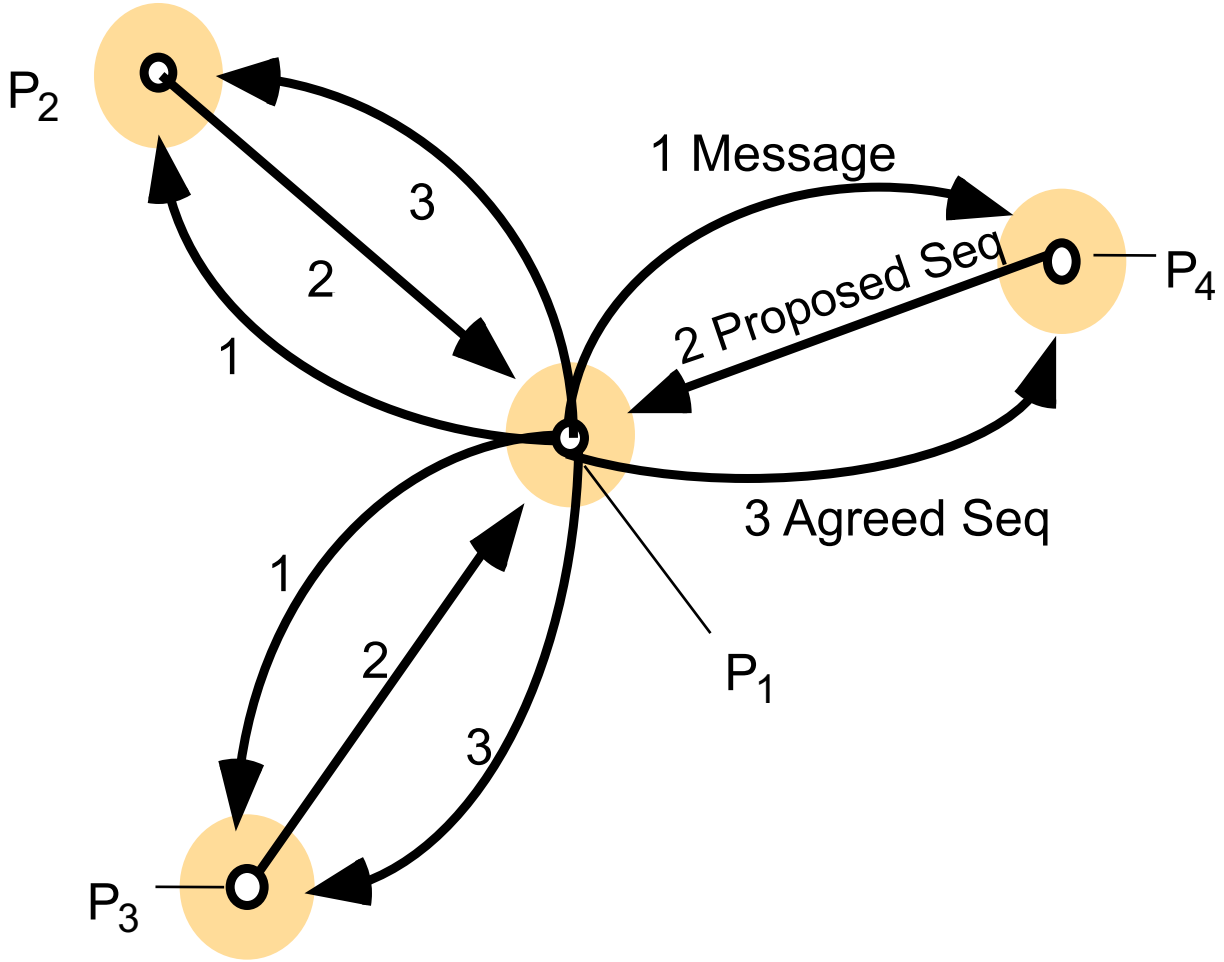
- Special process elected as leader or sequencer.
- TO-multicast(g,m) at P*i*:
  - Send multicast message m to group g *and the sequencer*
- Sequencer:
  - Maintains a global sequence number S (initially 0)
  - When a multicast message m is B-delivered to it:
    - sets S = S + 1, and B-multicast(g,{"order", m, S})
- Receive multicast at process P*i*:
  - P*i* maintains a local received global sequence number S*i* (initially 0)
  - On B-deliver(m) at P*i* from P*j*, it buffers it until both conditions satisfied
    1. B-deliver({"order", m, S}) at P*i* from sequencer, and
    2. S*i* + 1 = S
    - Then TO-deliver(m) to application and set S*i* = S*i* + 1

Implementing total order multicast

- Basic idea:
  - Same sequence number counter across different processes.
  - Instead of different sequence number counter for each process.

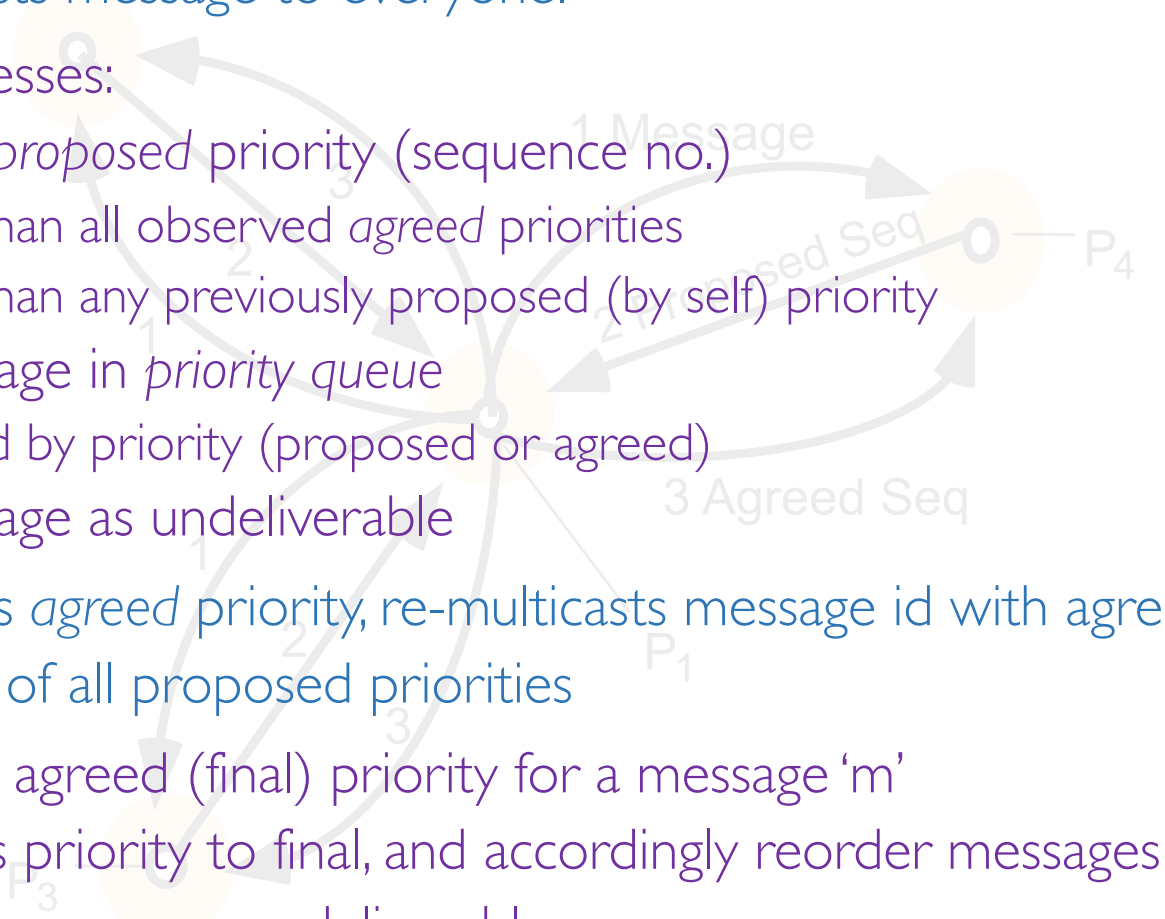
- Two types of approach
  - Using a centralized sequencer
  - **A decentralized mechanism (ISIS)**

ISIS algorithm for total ordering



ISIS algorithm for total ordering

- Sender multicasts message to everyone.
- Receiving processes:
  - reply with *proposed* priority (sequence no.)
    - larger than all observed *agreed* priorities
    - larger than any previously proposed (by self) priority
  - store message in *priority queue*
    - ordered by priority (proposed or agreed)
  - mark message as undeliverable
- Sender chooses *agreed* priority, re-multicasts message id with agreed priority
  - maximum of all proposed priorities
- Upon receiving agreed (final) priority for a message 'm'
  - Update m's priority to final, and accordingly reorder messages in queue.
  - mark the message m as deliverable.
  - deliver any deliverable messages at front of priority queue.



ISIS algorithm for total ordering

- We'll discuss an example in the next class.