Distributed Systems

CS425/ECE428

Feb 8 2023

Instructor: Radhika Mittal

Acknowledgements for some of materials: Indy Gupta and Nikita Borisov

Logistics

- MP0 is due today at 11:59pm.
- Please make sure you are on CampusWire
  - Reach out to Manoj (gmk6) if you need access.
- Reminder to share your name when you speak up in class.

Today’s agenda

- **Multicast**
  - Chapter 15.4

- **Goal:** reason about desirable properties for message delivery among a group of processes.

Communication modes

- **Unicast**
  - Messages are sent from exactly one process to one process.
- **Broadcast**
  - Messages are sent from exactly one process to all processes on the network.
- **Multicast**
  - Messages broadcast within a group of processes.
  - A multicast message is sent from any one process to a group of processes on the network.

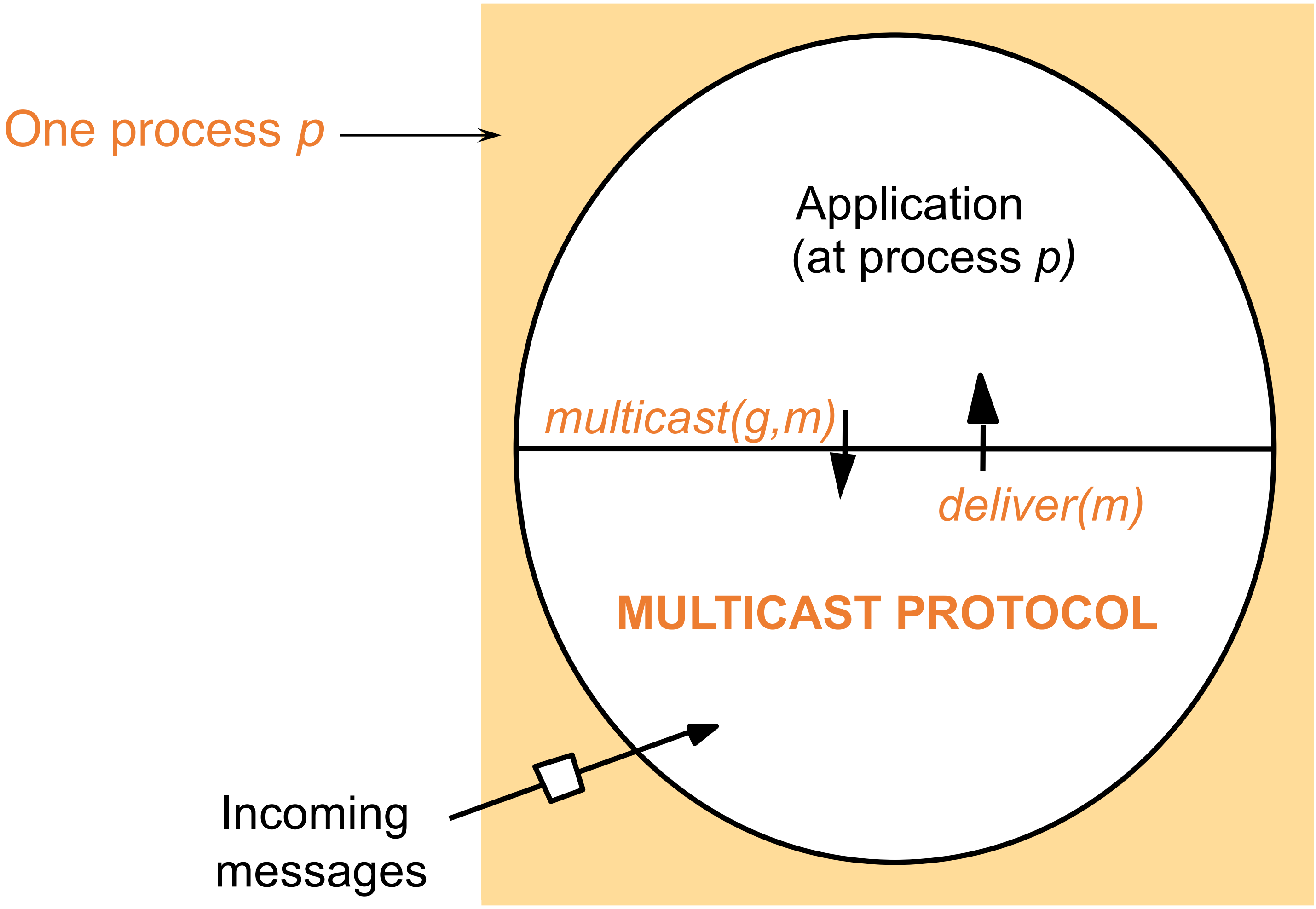
Where is multicast used?

- Distributed storage
  - Write to an object are multicast across replica 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.
- ……

Communication modes

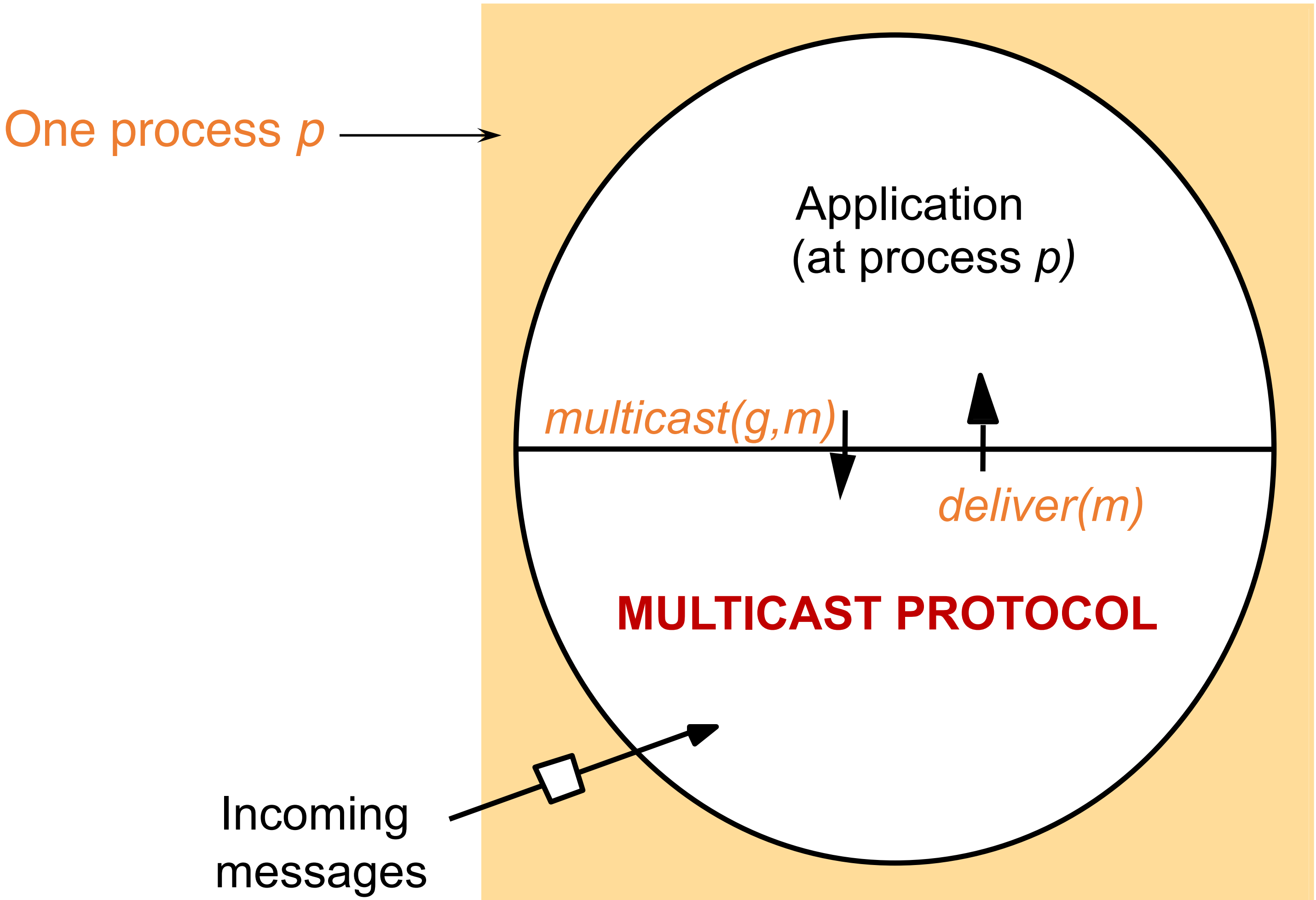
- **Unicast**
  - Messages are sent from exactly <u>one</u> process <u>to</u> <u>one</u> process.
    - *Best effort:* if a message is delivered it would be intact; no reliability guarantees.
    - *Reliable:* guarantees delivery of messages.
    - *In order:* messages will be delivered in the same order that they are sent.
- **Broadcast**
  - Messages are sent from exactly <u>one</u> process <u>to</u> <u>all</u> processes on the network.
- **Multicast**
  - Messages broadcast within a group of processes.
  - A multicast message is sent from any <u>one</u> process <u>to</u> the <u>group</u> of processes on the network.
  - *How do we define (and achieve) reliable or ordered multicast?*

What we are designing in this class?



'g' is a multicast group that also includes the process 'p'.

What we are designing in this class?



'g' is a multicast group that also includes the process 'p'.

Basic Multicast (B-Multicast)

- Straightforward way to implement B-multicast:
  - use a reliable one-to-one send (unicast) operation:

    B-multicast(group g, message m):

    for each process p in g, send (p,m).

    receive(m): B-deliver(m) at p.
- Guarantees: message is eventually delivered to the group if:
  - Processes are non-faulty.
  - The unicast "send" is reliable.
  - *Sender does not crash.*
- *Can we provide reliable delivery even after sender crashes?*
  - *What does this mean?*

Reliable Multicast (R-Multicast)

- **Integrity**: A *correct* (i.e., non-faulty) process *p* delivers a message *m* at most once.
  - *Assumption: no process sends **exactly** the same message twice*
- **Validity**: If a *correct* process multicasts (sends) message *m*, then it will eventually deliver *m* itself.
  - *Liveness for the sender.*
- **Agreement**: If a *correct* process delivers message *m*, then all the other *correct* processes in group(*m*) will eventually deliver *m*.
  - *All or nothing.*
- Validity and agreement together ensure overall liveness: if some correct process multicasts a message *m*, then, all correct processes deliver *m* too.

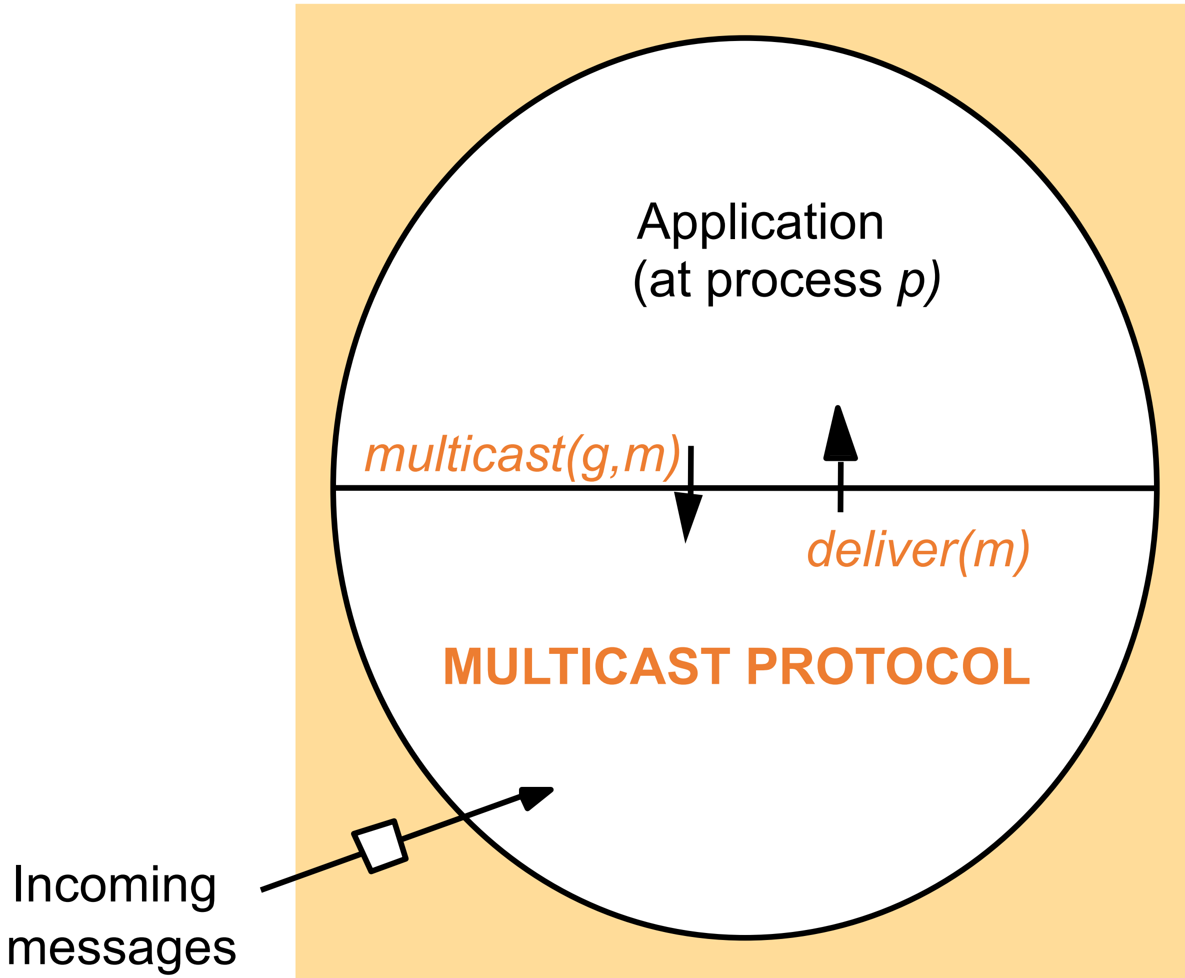
Reliable Multicast (R-Multicast)

- **Integrity**: A *correct* (i.e., non-faulty) process *p* delivers a message *m* at most onc
  - *Assur* *wice*
- **Validity**: I hen it will eventuall
  - *Livene*
- **Agreeme** the other *correct* pr
  - *All or*
- Validity and agreement together ensure overall liveness: if some correct process multicasts a message *m*, then, all correct processes deliver *m* too.

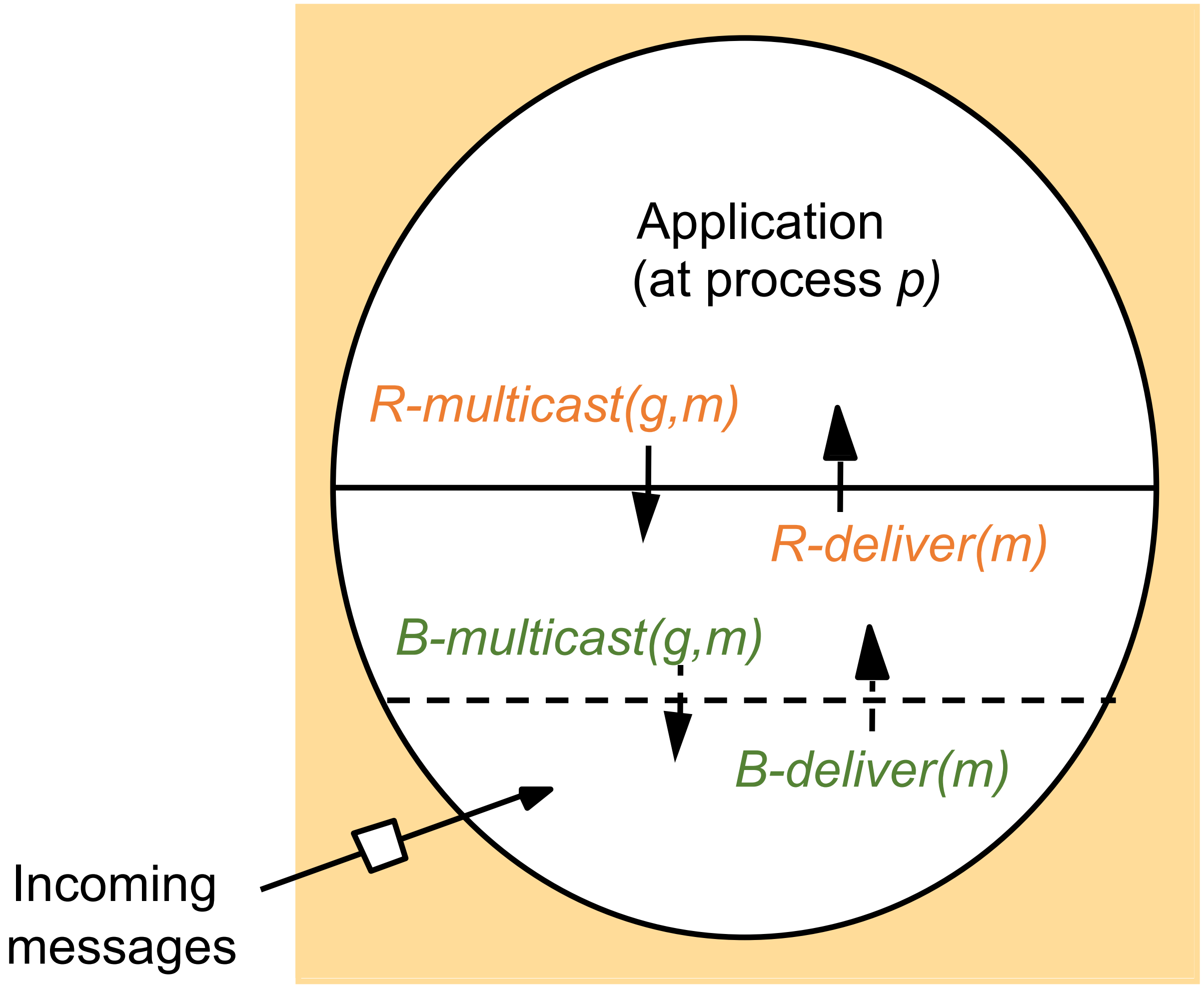
What happens if a process initiates B-multicasts of a message but fails after unicasting to a subset of processes in the group?

Agreement is violated! R-multicast not satisfied.

Implementing R-Multicast



Implementing R-Multicast



Implementing R-Multicast

On initialization

    Received := {};

For process p to R-multicast message m to group g

    B-multicast(g,m); (p ∈ g is included as destination)

On B-deliver(m) at process q in g = group(m)

    if (m ∉ Received):

        Received := Received ∪ {m};

        if (q ≠ p): B-multicast(g,m);

        R-deliver(m)

Reliable Multicast (R-Multicast)

- **Integrity**: A *correct* (i.e., non-faulty) process *p* delivers a message *m* at most once.
  - *Assumption: no process sends **exactly** the same message twice*
- **Validity**: If a *correct* process multicasts (sends) message *m*, then it will eventually deliver *m* itself.
  - *Liveness for the sender.*
- **Agreement**: If a *correct* process delivers message *m*, then all the other *correct* processes in group(*m*) will eventually deliver *m*.
  - *All or nothing.*
- Validity and agreement together ensure overall liveness: if some correct process multicasts a message *m*, then, all correct processes deliver *m* too.

Ordered Multicast

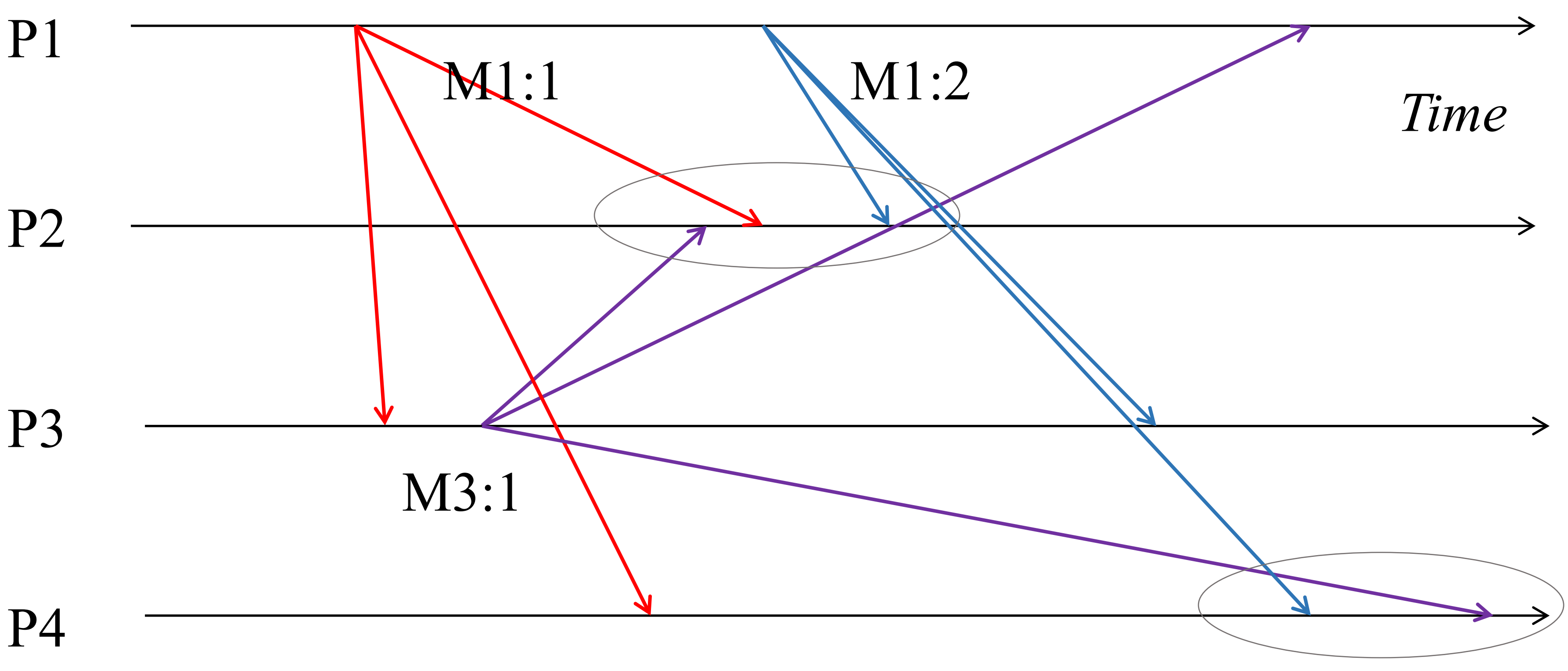
- Three popular flavors implemented by several multicast protocols:
  1. FIFO ordering
  2. Causal ordering
  3. Total ordering

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.*

FIFO Order: Example



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

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.

Where is causal ordering useful?

- 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.

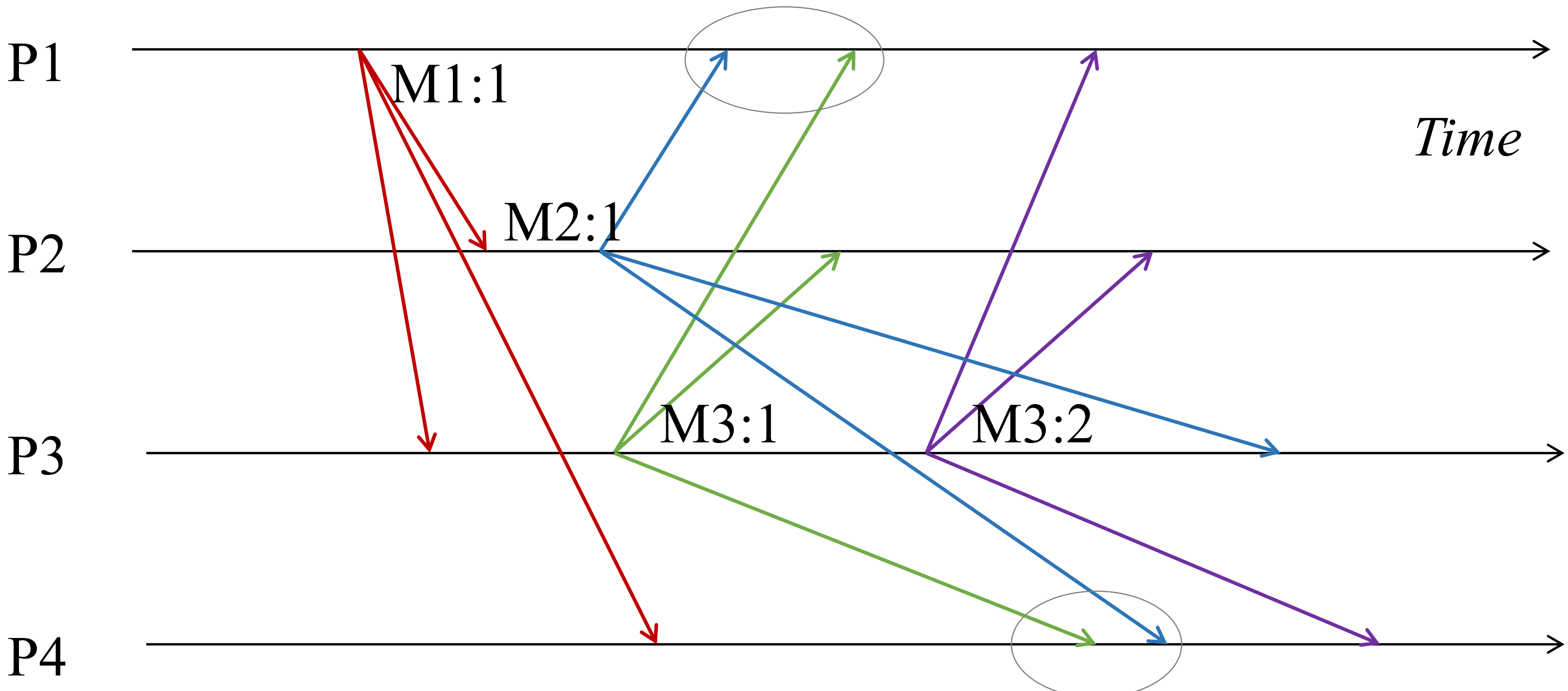
HB Relationship for Causal Ordering

- 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)** $\rightarrow$ **receive(m)**

HB Relationship for Causal Ordering

- HB rules in causal ordered multicast:
  - If $\exists$ $\mathbf{p_i}$, **e** $\rightarrow_i$ **e'** then **e** $\rightarrow$ **e'**.
    - If $\exists$ $\mathbf{p_i}$, multicast(*g*,*m*) $\rightarrow_i$ multicast(*g*,*m'*), then multicast(*g*,*m*) $\rightarrow$ multicast(*g*,*m'*)
    - If $\exists$ $\mathbf{p_i}$, delivery(*m*) $\rightarrow_i$ multicast(*g*,*m'*), then delivery(*m*) $\rightarrow$ multicast(*g*,*m'*)
    - …
  - ~~For any message m, **send(m)** $\rightarrow$ **receive(m)**~~
    - For any *multicast* message m, multicast(*g*,*m*) $\rightarrow$ delivery(m)
  - If **e** $\rightarrow$ **e'** and **e'** $\rightarrow$ **e''** then **e** $\rightarrow$ **e''**
    - multicast(*g*,*m*) at p_i $\rightarrow$ delivery(m) at p_j
    - delivery(*m*) at p_j $\rightarrow$ multicast(*g*,*m'*) at p_j
    - multicast(*g*,*m*) at p_i $\rightarrow$ 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



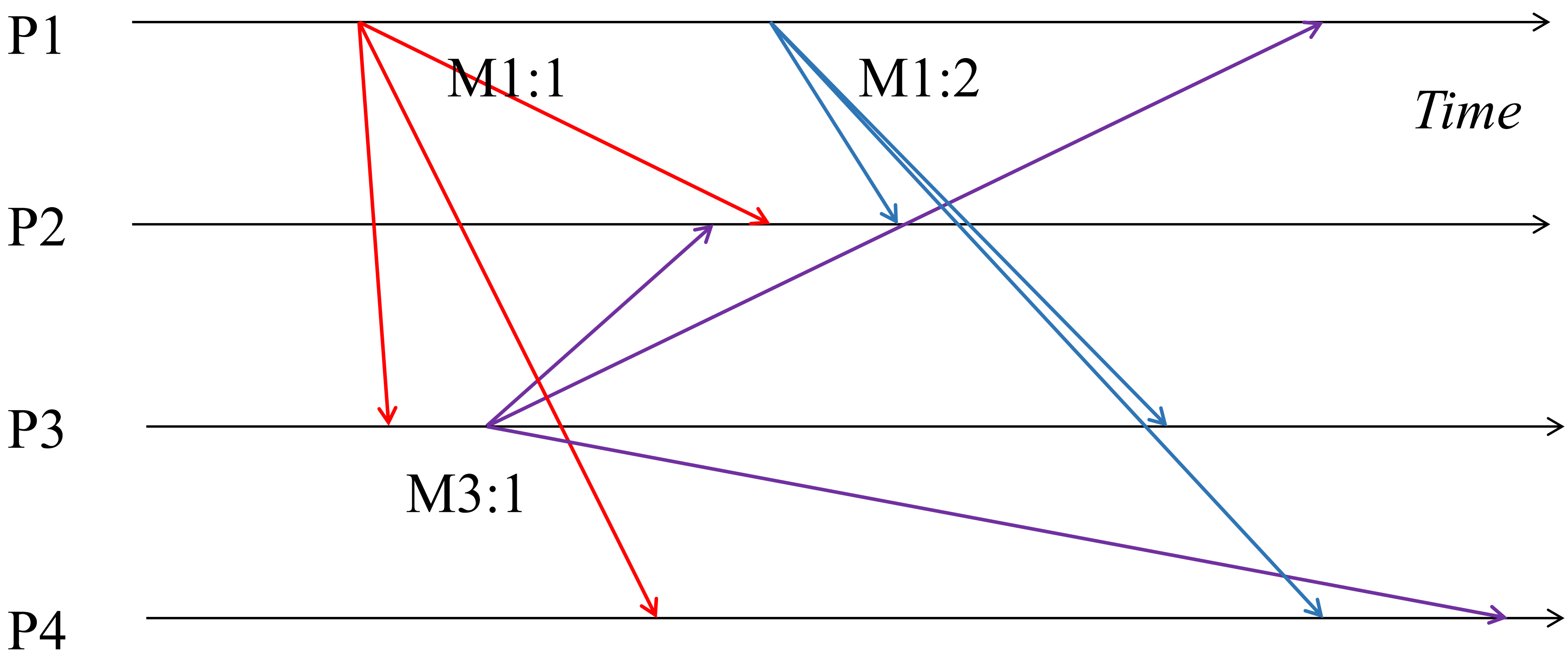
M3:1 → M3:2, M1:1 → M2:1, M1:1 → M3:1 and so should be delivered in that order at each receiver.

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

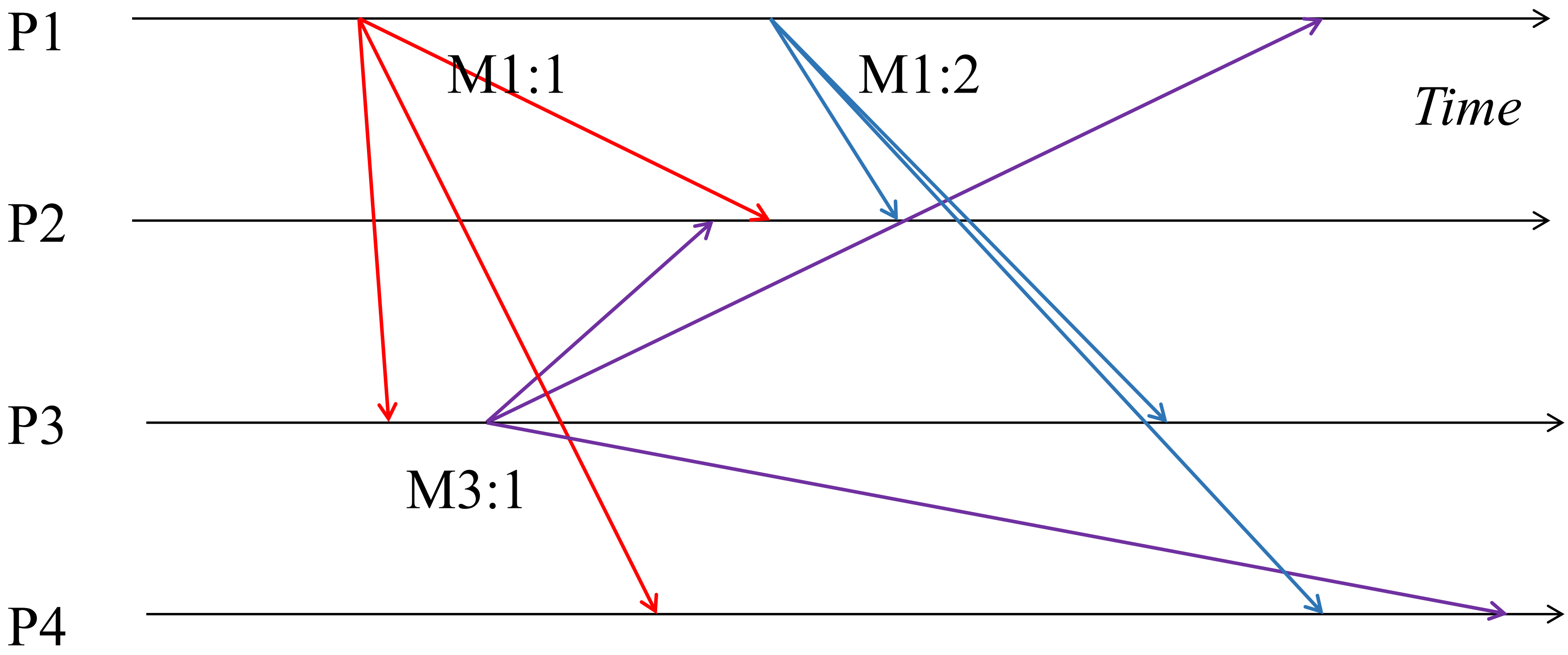
- Causal Ordering => FIFO Ordering
- Why?
  - If two multicasts M and M' are sent by the same process P, and M was sent before M', then M → M'.
  - Then a multicast protocol that implements causal ordering will obey FIFO ordering since M → M'.
- Reverse is not true! FIFO ordering does not imply causal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

Example



Does this satisfy FIFO order?

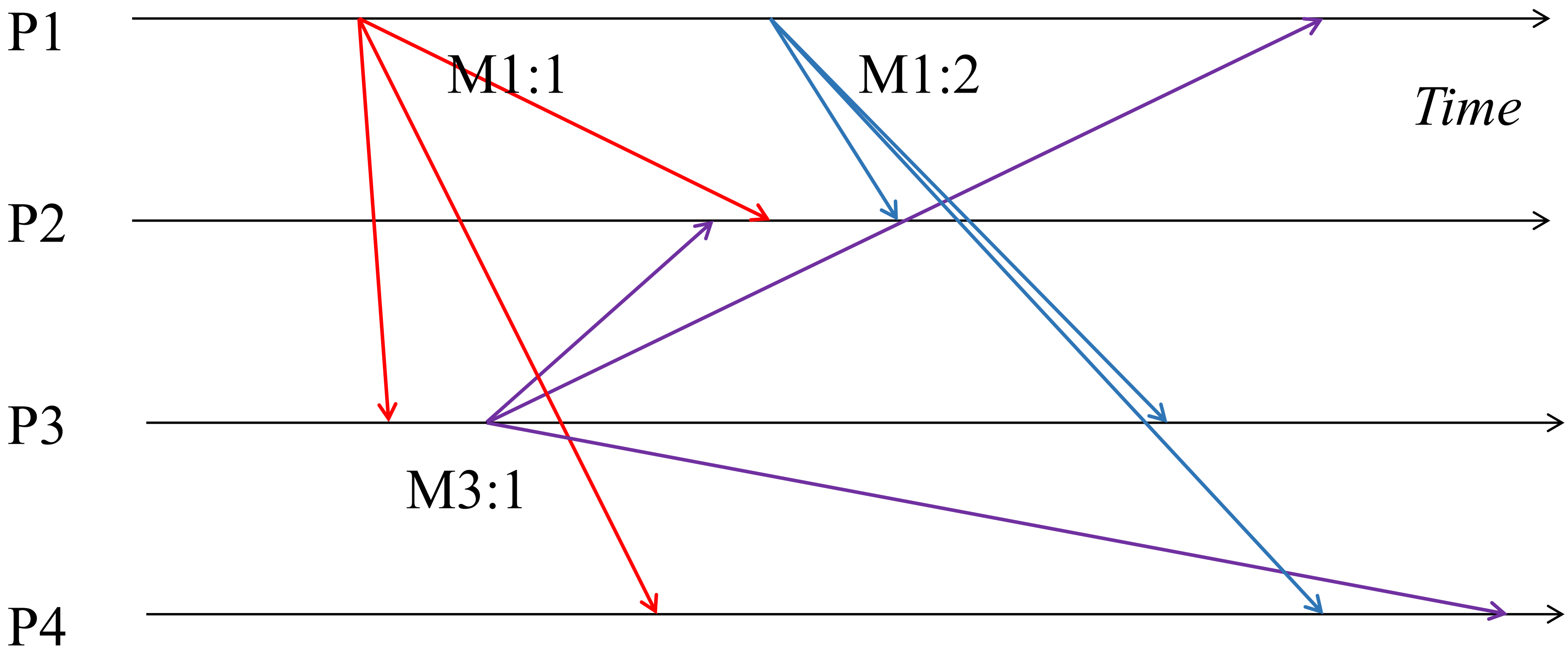
Example



Does this satisfy FIFO order?

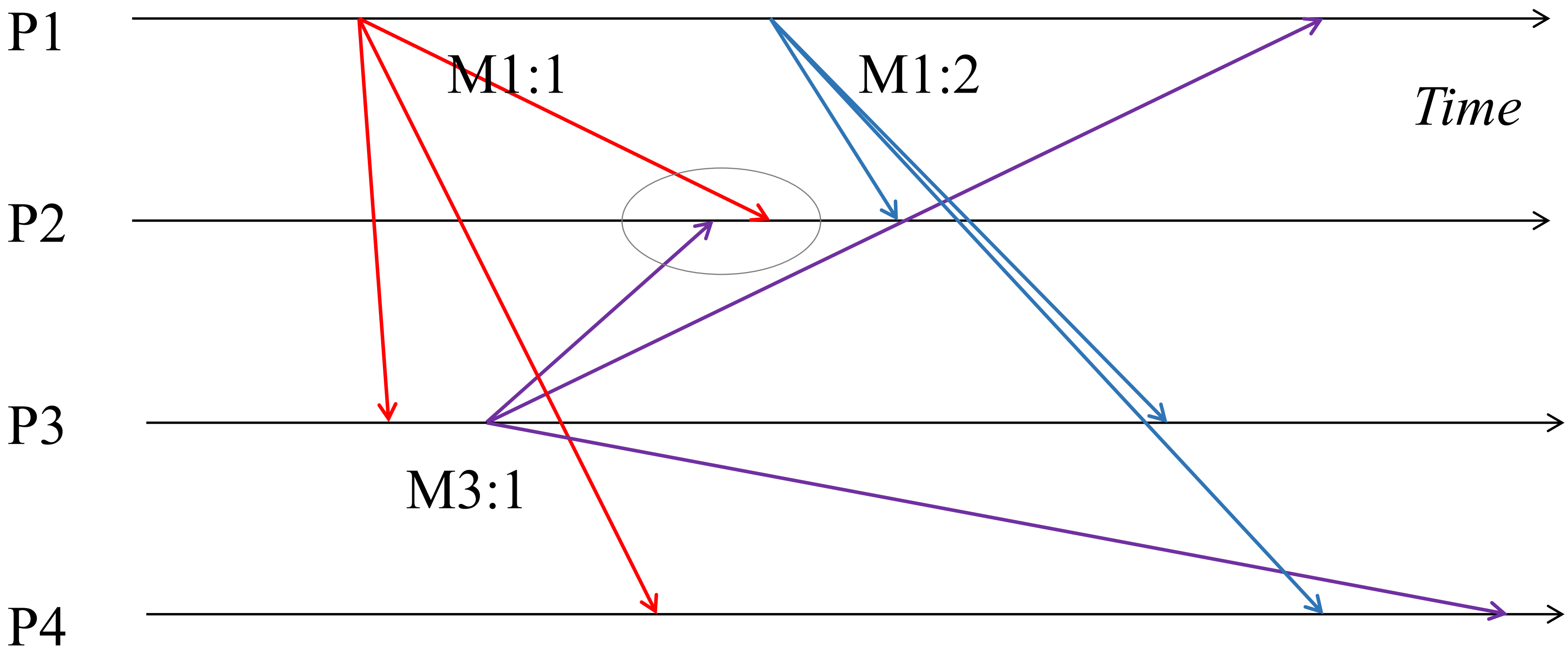
Yes

Example



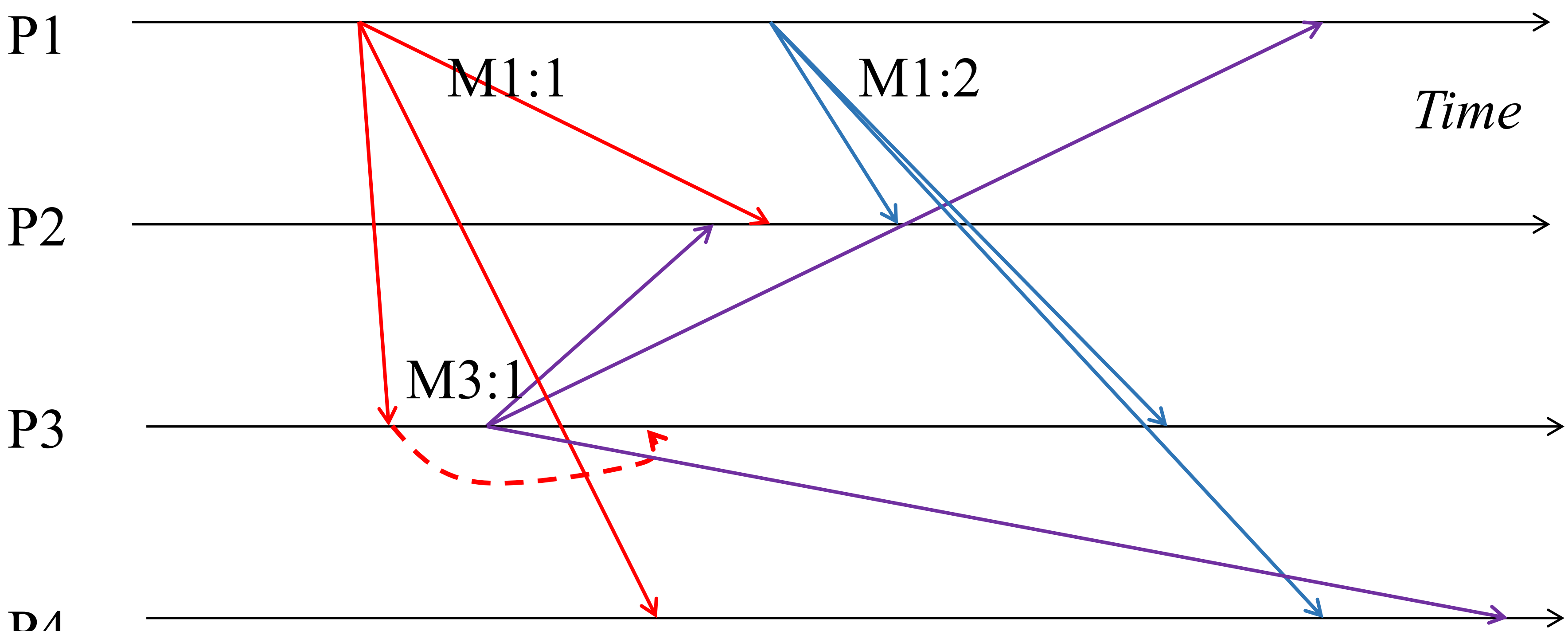
Does this satisfy causal order?

Example



Does this satisfy causal order?
No

Example

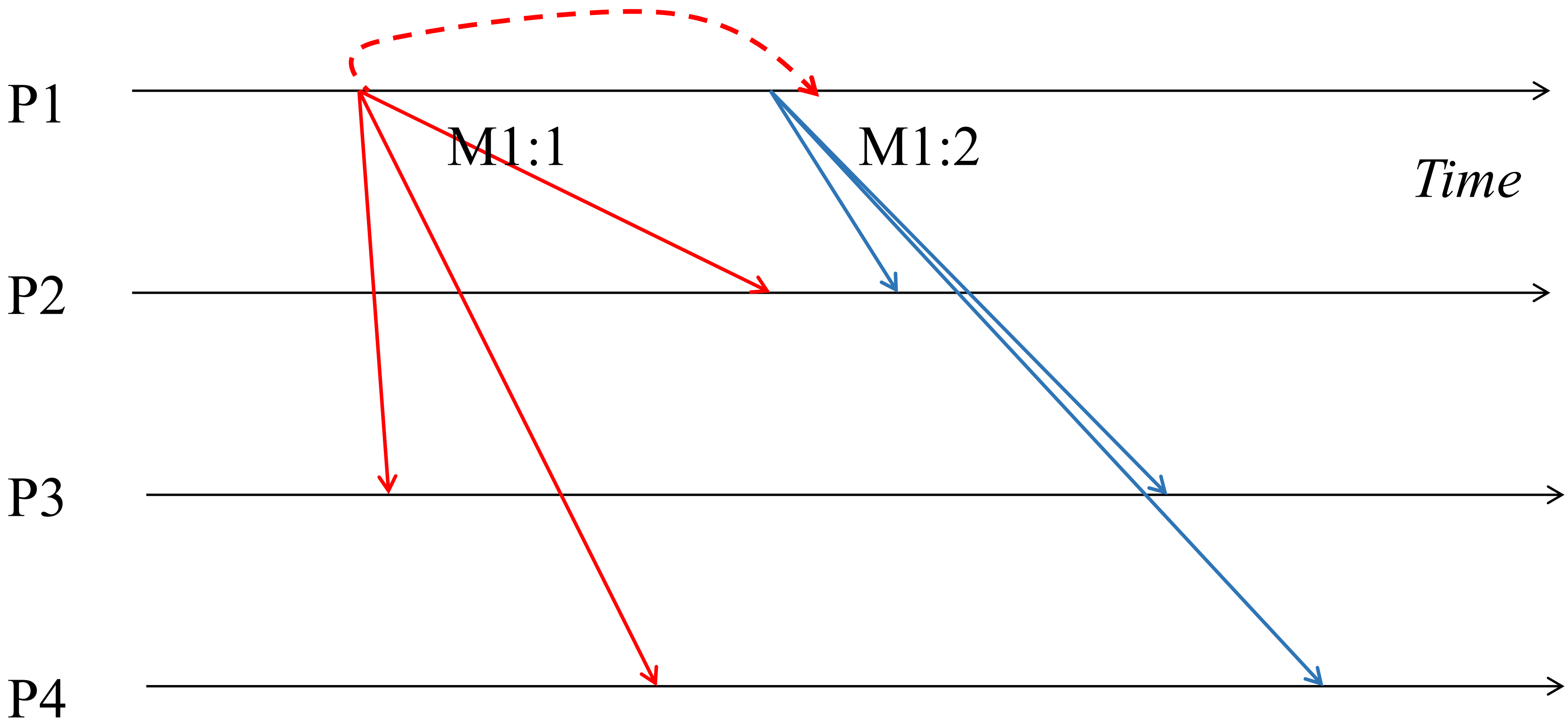


M1:1 is delivered at P3 after M3:1's multicast.

Does this satisfy causal order?

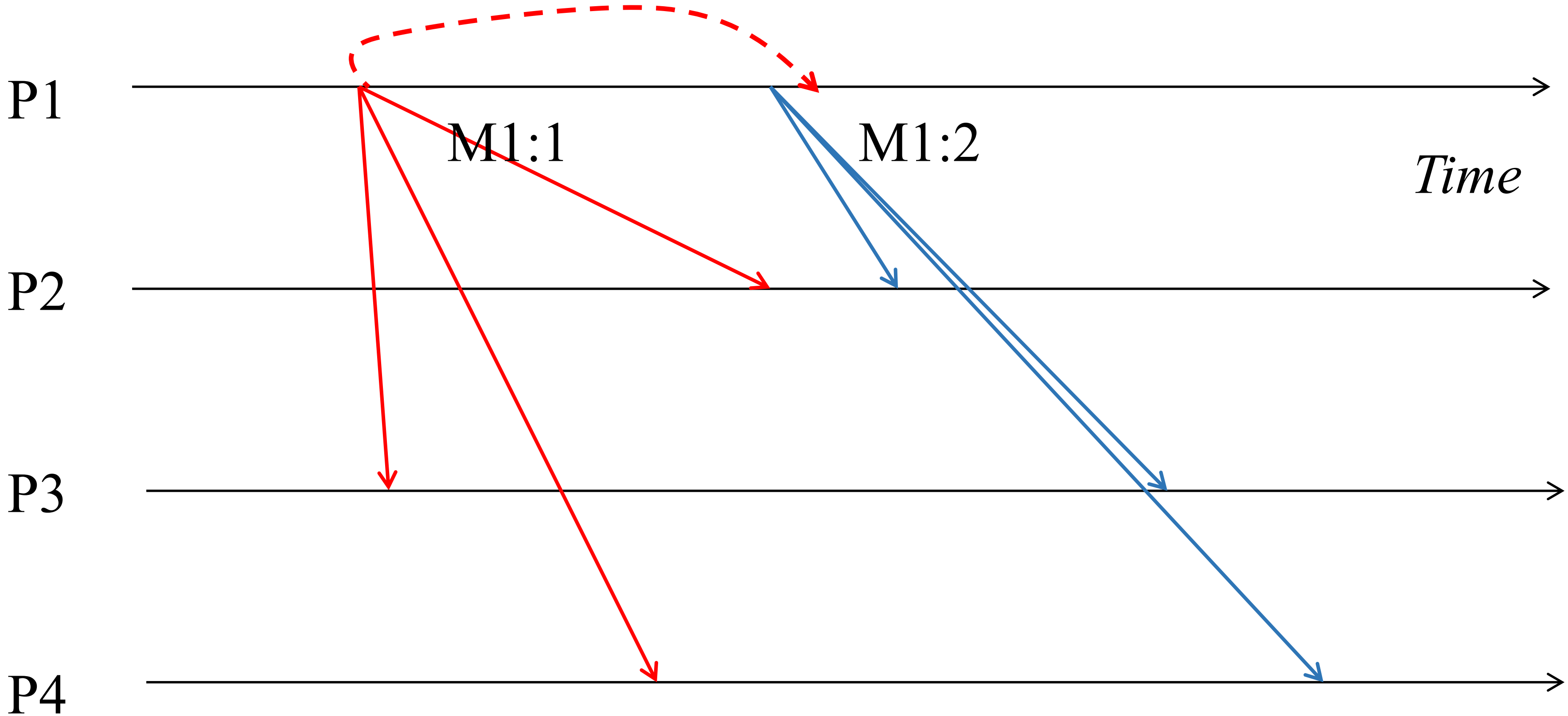
Yes

Example



Does this satisfy causal order?
No

Example



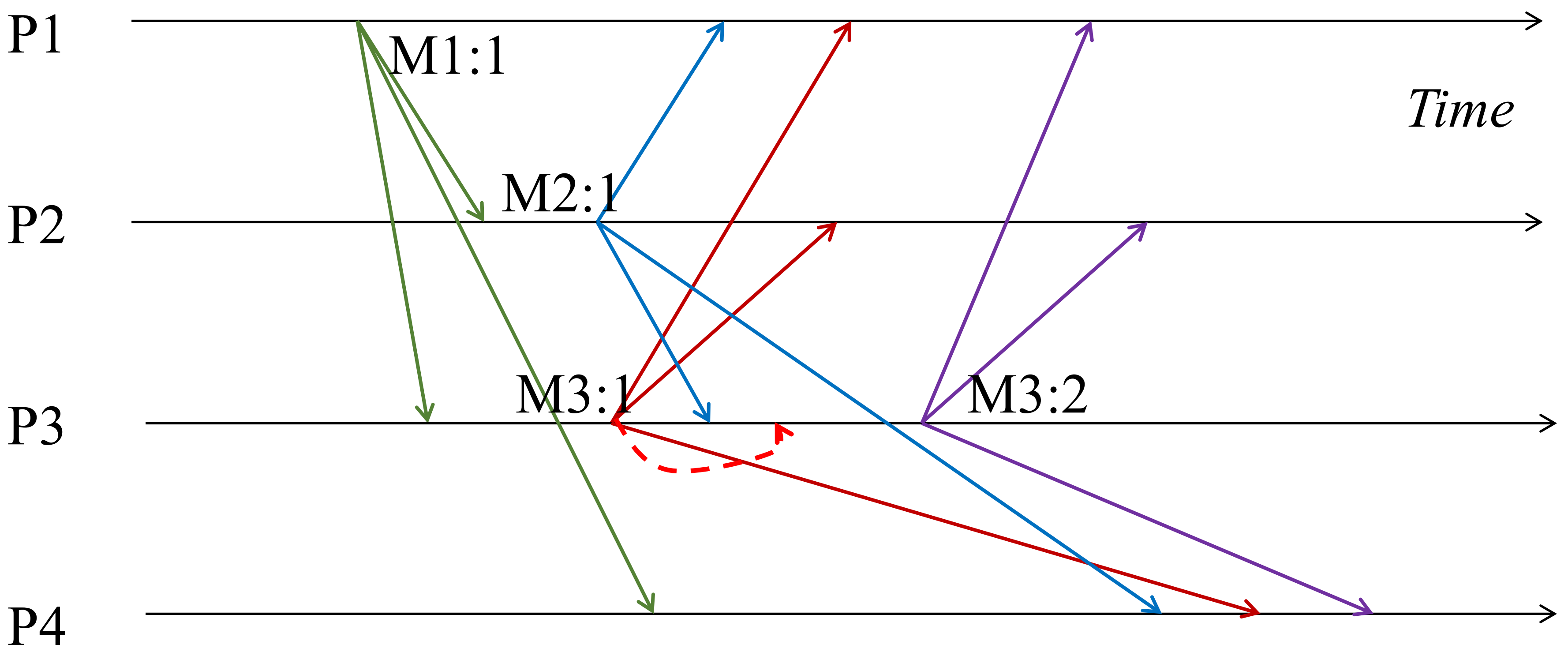
Does this satisfy FIFO order?

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



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.