Multicast

Yao Liu

Distributed communication forms

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

Multicast in practice

- Replicated storage systems
  - Writes/reads to the key are multicast within the replica group
- Online scoreboards
  - Multicast to group of clients interested in the scores
- Air traffic control system
  - All controllers need to receive the same updates in the same order

Multicast ordering

- Multicasts have to be received in the same (somewhat) consistent order at all the processes in the group

- Three popular multicast ordering:
  - FIFO ordering
  - Causal ordering
  - Total ordering

Display from bulletin board program

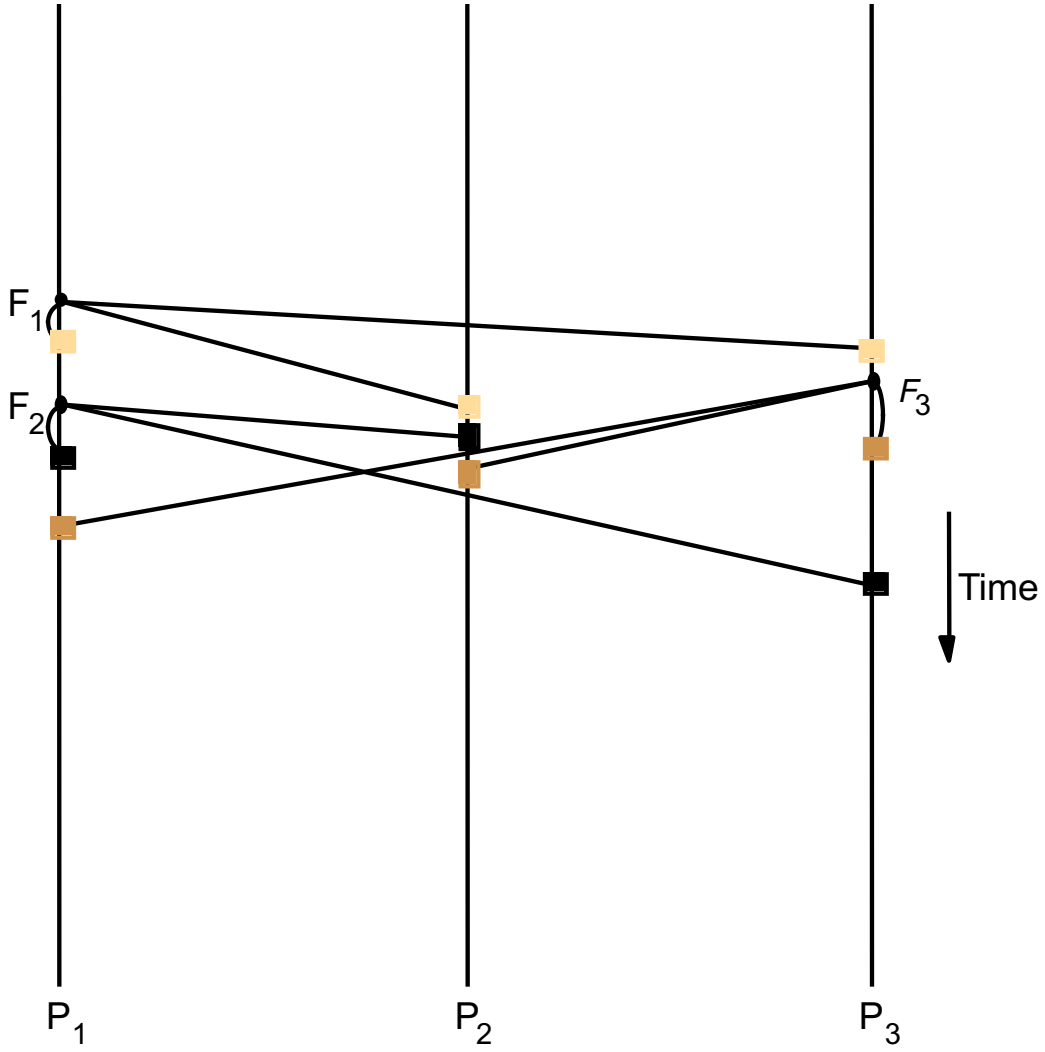
Bulletin board: *os.interesting*		
Item	From	Subject
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L'Heureux	RPC performance
27	M.Walker	Re: Mach
end		

FIFO ordering

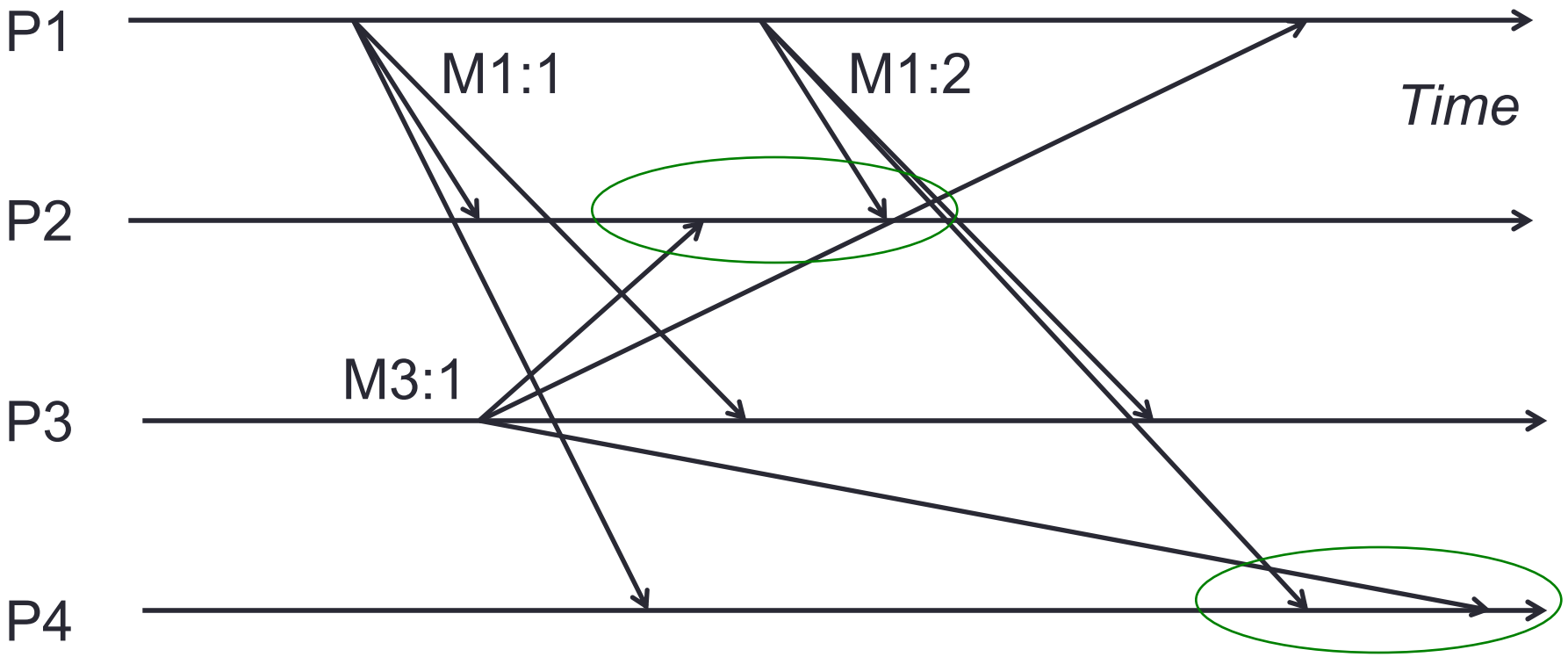
- Multicasts from each sender are received in the order they are sent, at all receivers
- Don't worry about multicasts from different senders
- More formally
  - If a correct process issues (sends) multicast(*g*,*m*) to group *g* and then multicast(*g*,*m'*), then every correct process that delivers *m'* would already have delivered *m*.

FIFO ordering: example

Notice the consistent ordering of FIFO-related messages F_1 and F_2 , – and the otherwise arbitrary delivery ordering of messages.



FIFO ordering: example



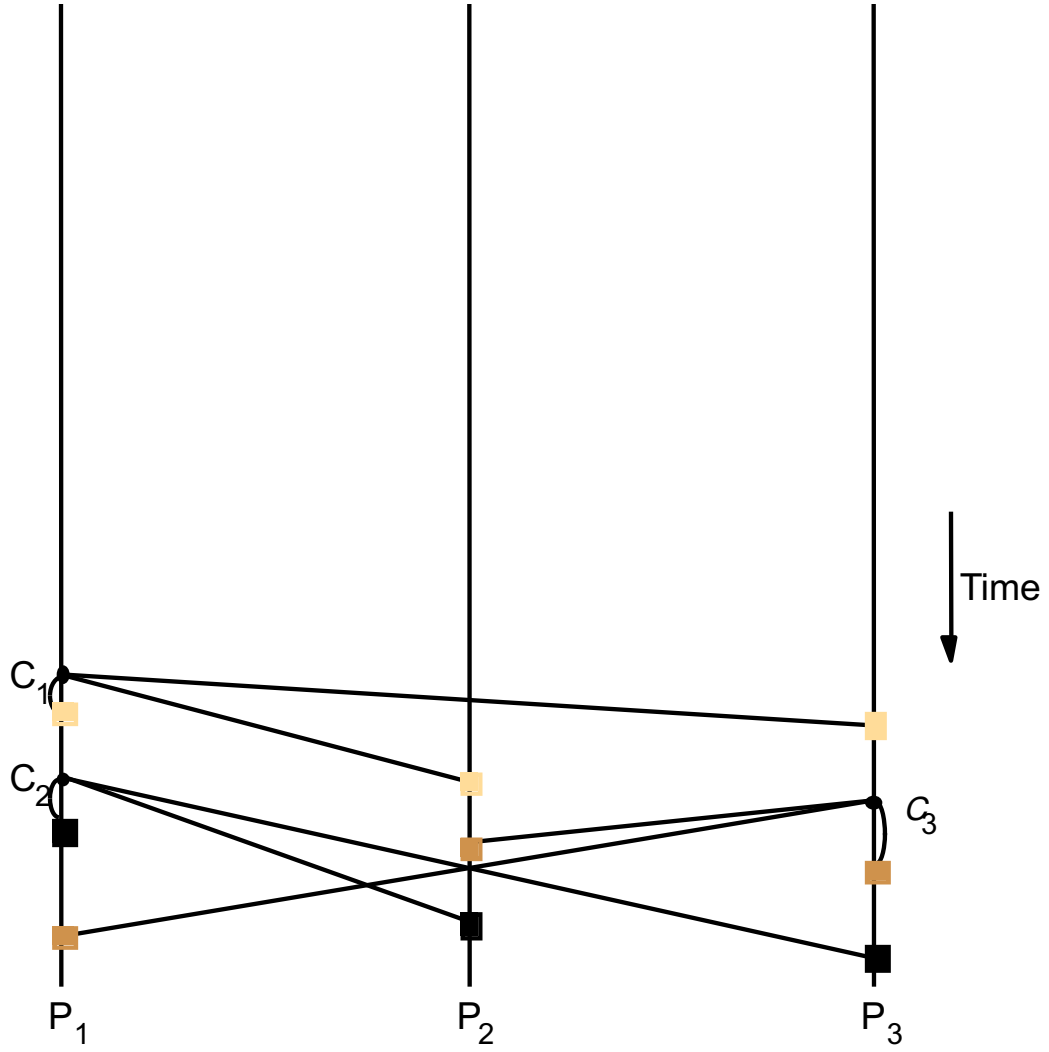
M1:1 and M1:2 should be received in that order at each receiver
Order of delivery of M3:1 and M1:2 could be different at different receivers

Causal ordering

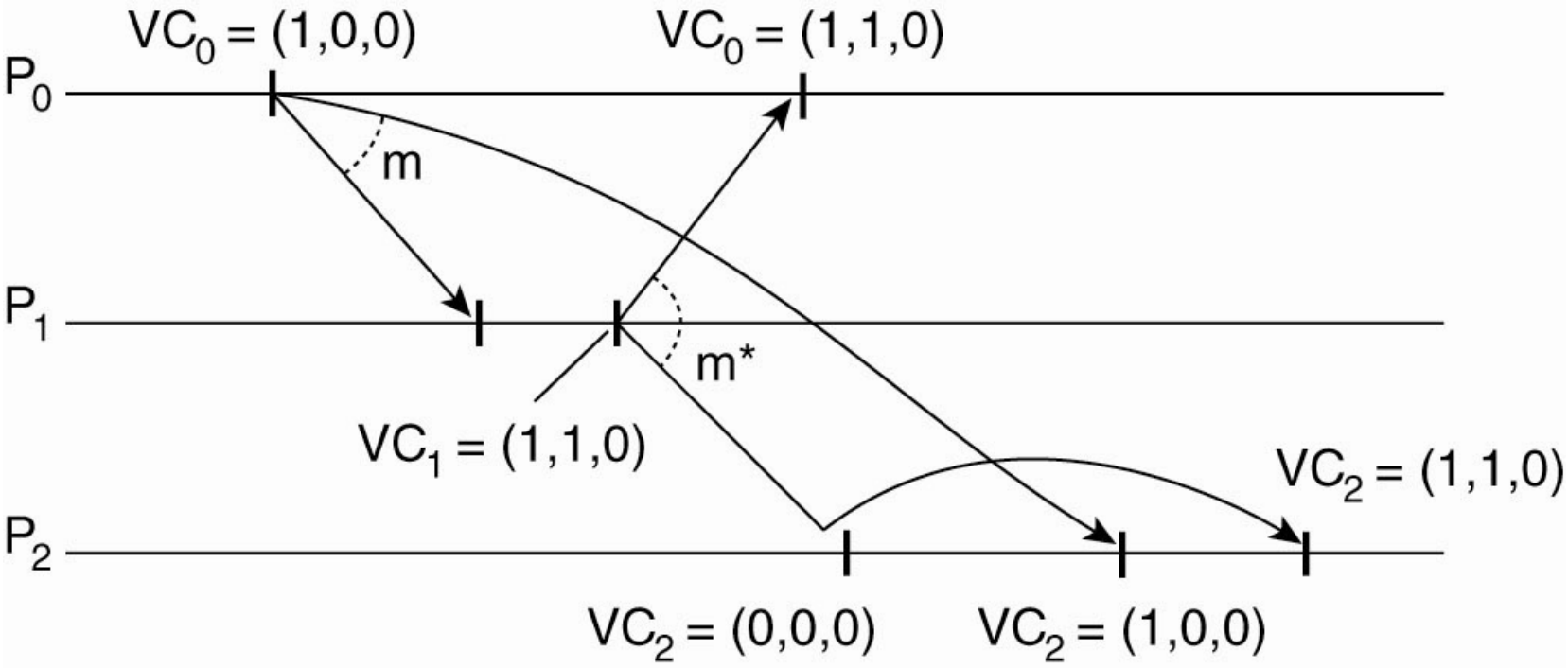
- Multicasts whose send events are causally related, must be received in the same causality-obeying order at all receivers
- Formally
  - If multicast(*g*,*m*) → multicast(*g*,*m'*), then any correct process that delivers *m'* would already have delivered *m*.
  - (→ stands for happened-before)

Causal ordering: example

Notice the consistent ordering of causally related messages C_1 and C_2, C_1 and C_3
– and the otherwise arbitrary delivery ordering of messages.



Causal ordering: example



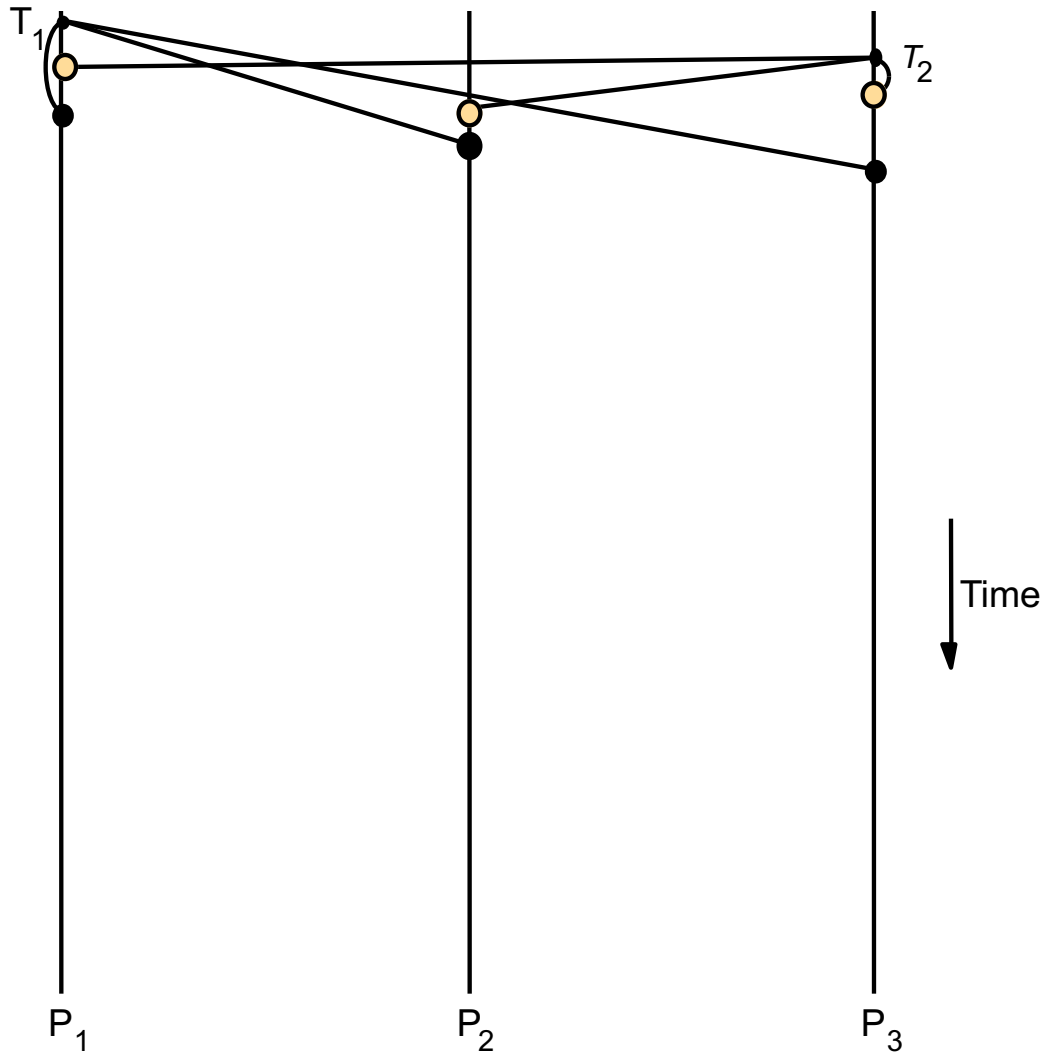
m → m*, and therefore should be received in that order at each receiver

Total ordering

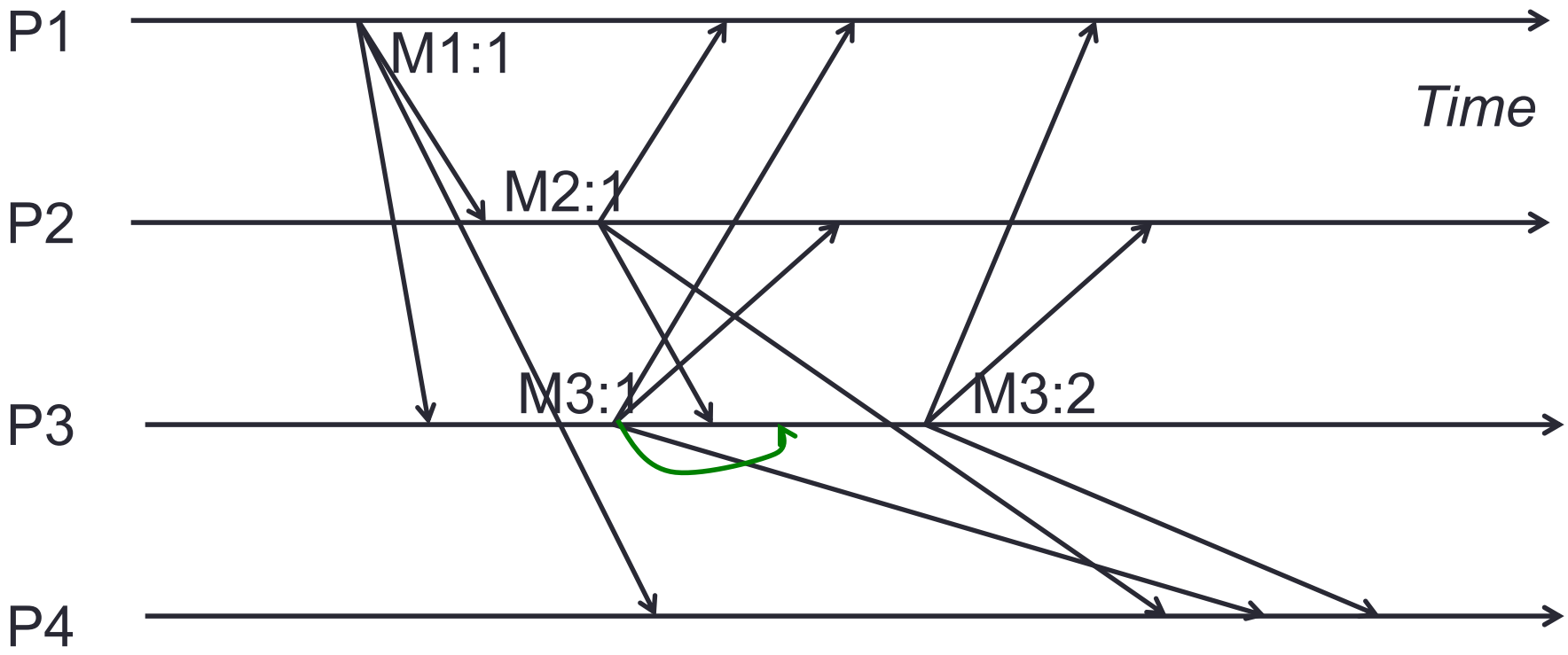
- Ensures all receivers receive all multicasts in the same order
- However, no guarantee is made receiving multicasts in the order of sending
- Formally
  - If a correct process P delivers message *m* before *m'* (independent of the senders), then any other correct process P' that delivers *m'* would already have delivered *m*.

Total ordering: example

Notice the consistent ordering of totally ordered messages T_1 and T_2.



Total ordering: example

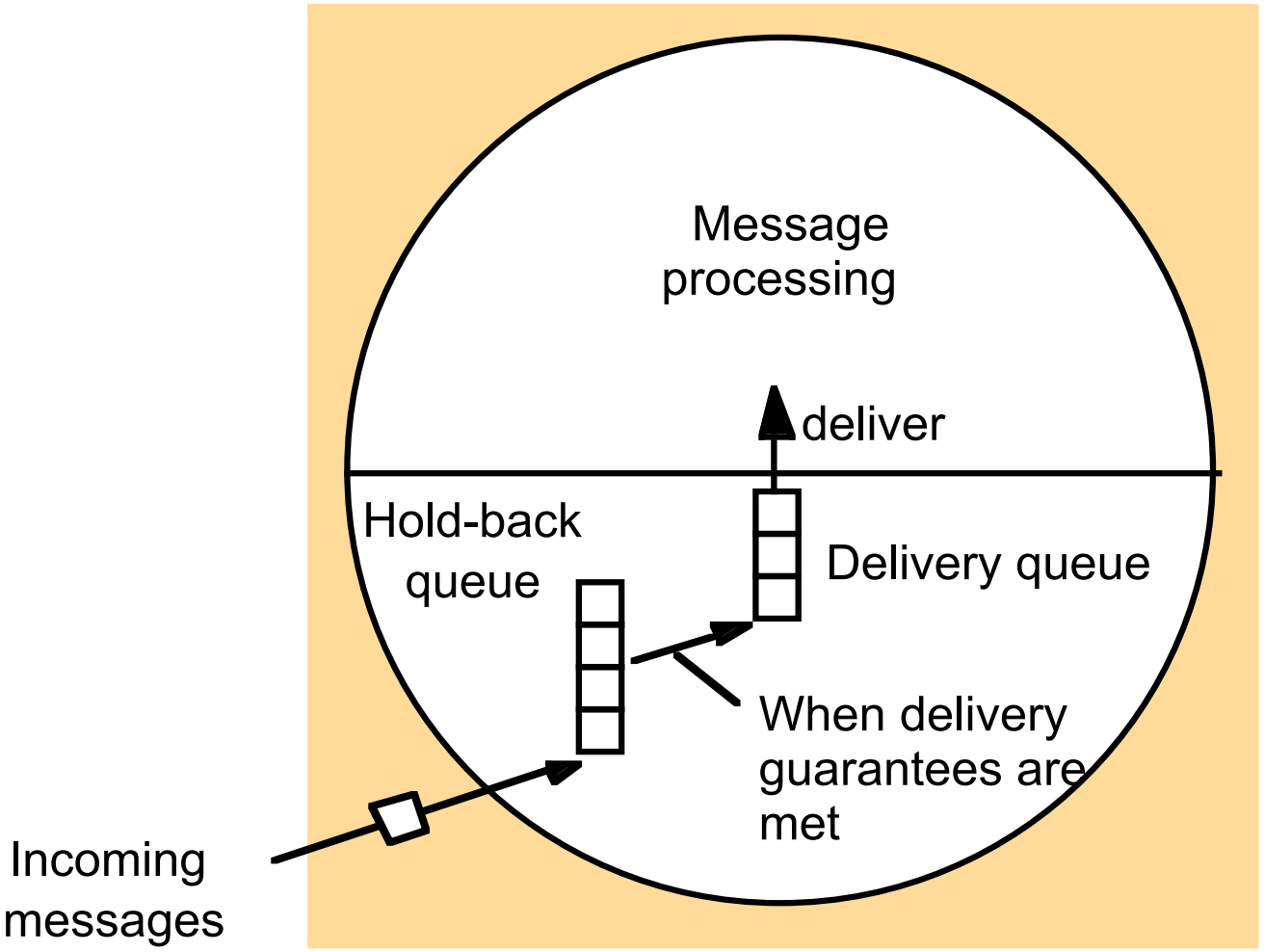


The order of receipt of multicasts is the same at all processes.
M1:1, then M2:1, then M3:1, then M3:2
May need to delay delivery of some messages

Implementing ordered multicast

- Incoming messages are held back in a queue until delivery guarantees can be met
- Coordination among all machines needed to determine delivery order
- Implementation of three popular flavors:
  - FIFO-ordering
    - easy, use a separate sequence number for each process
  - Causal ordering
    - use vector timestamps
  - Total ordering
    - use a sequencer

The hold-back queue for arriving multicast messages

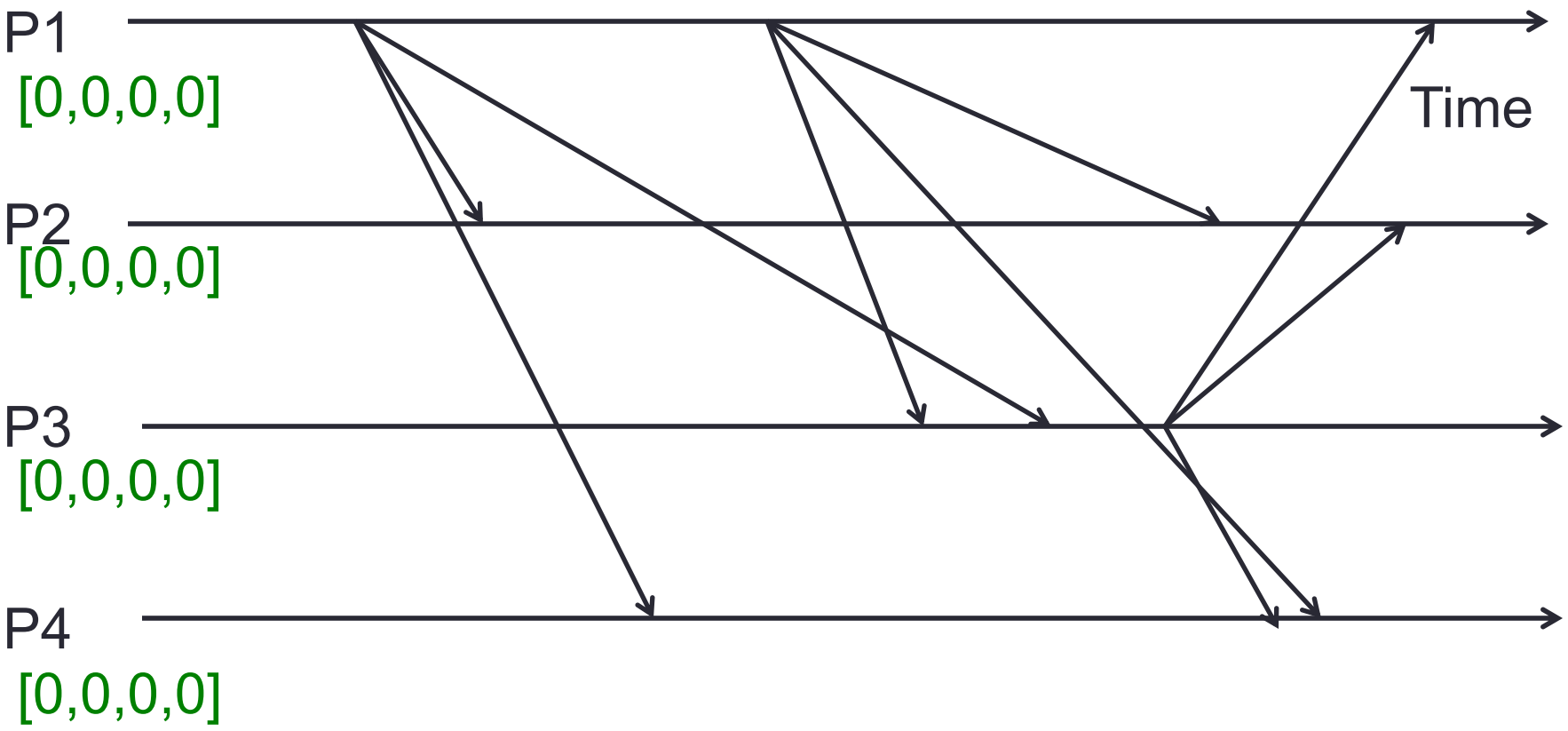


Implementing FIFO order

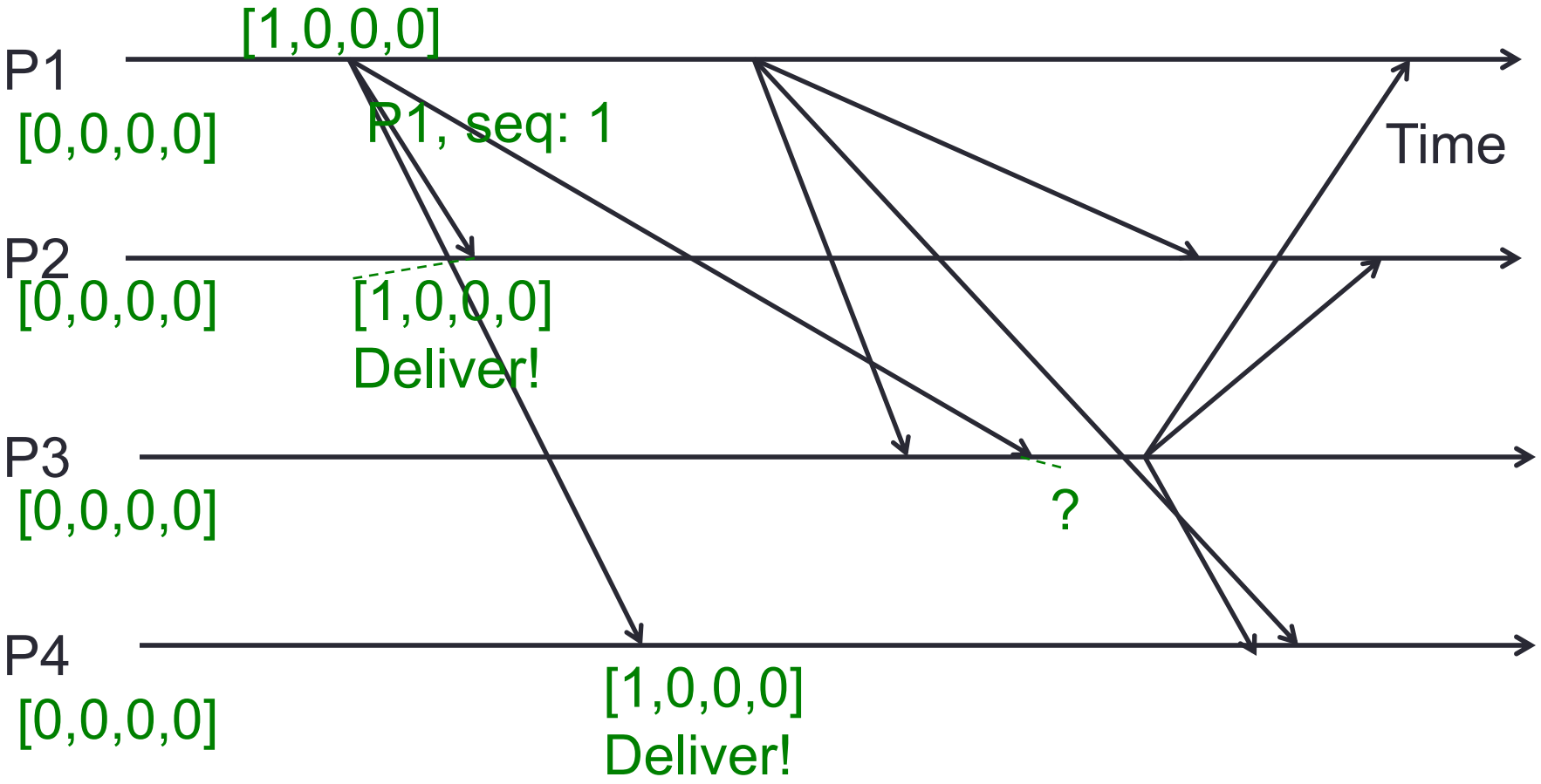
- Each receiver maintains per-sender sequence numbers (integers)
  - Processes P_1 through P_N
  - P*i* maintains a vector P*i*[1…N] (initially all zeroes)
  - P*i*[*j*] is the latest sequence number P*i* has received from P*j*

Implementing FIFO order

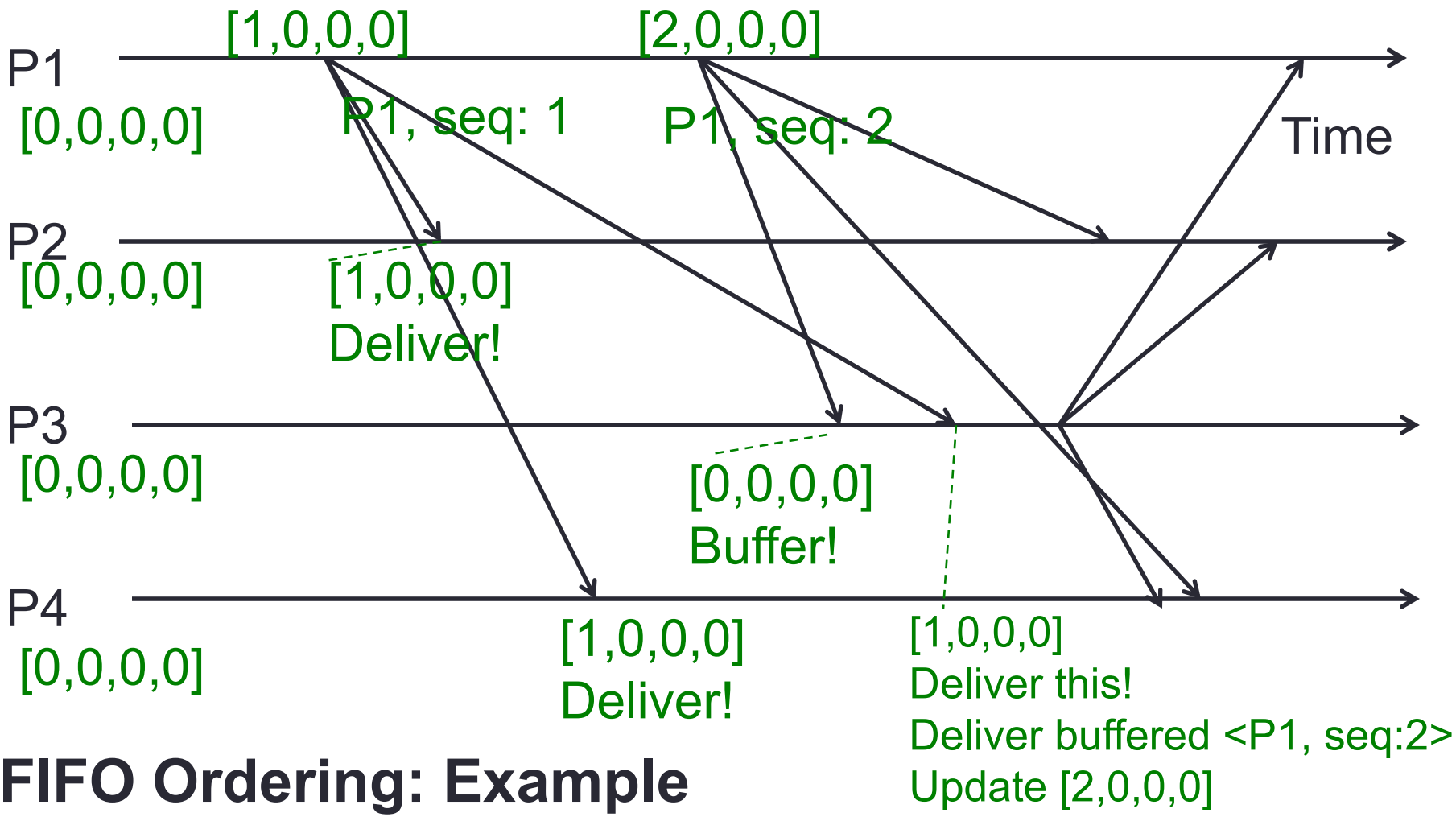
- Send multicast at process P*j*:
  - Set P*j*[*j*] = P*j*[*j*] + 1
  - Include new P*j*[*j*] in multicast message as its sequence number
- Receive multicast: If P*i* receives a multicast from P*j* with sequence number *S* in message
  - if (S == P*i*[*j*] + 1) then
    - deliver message to application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



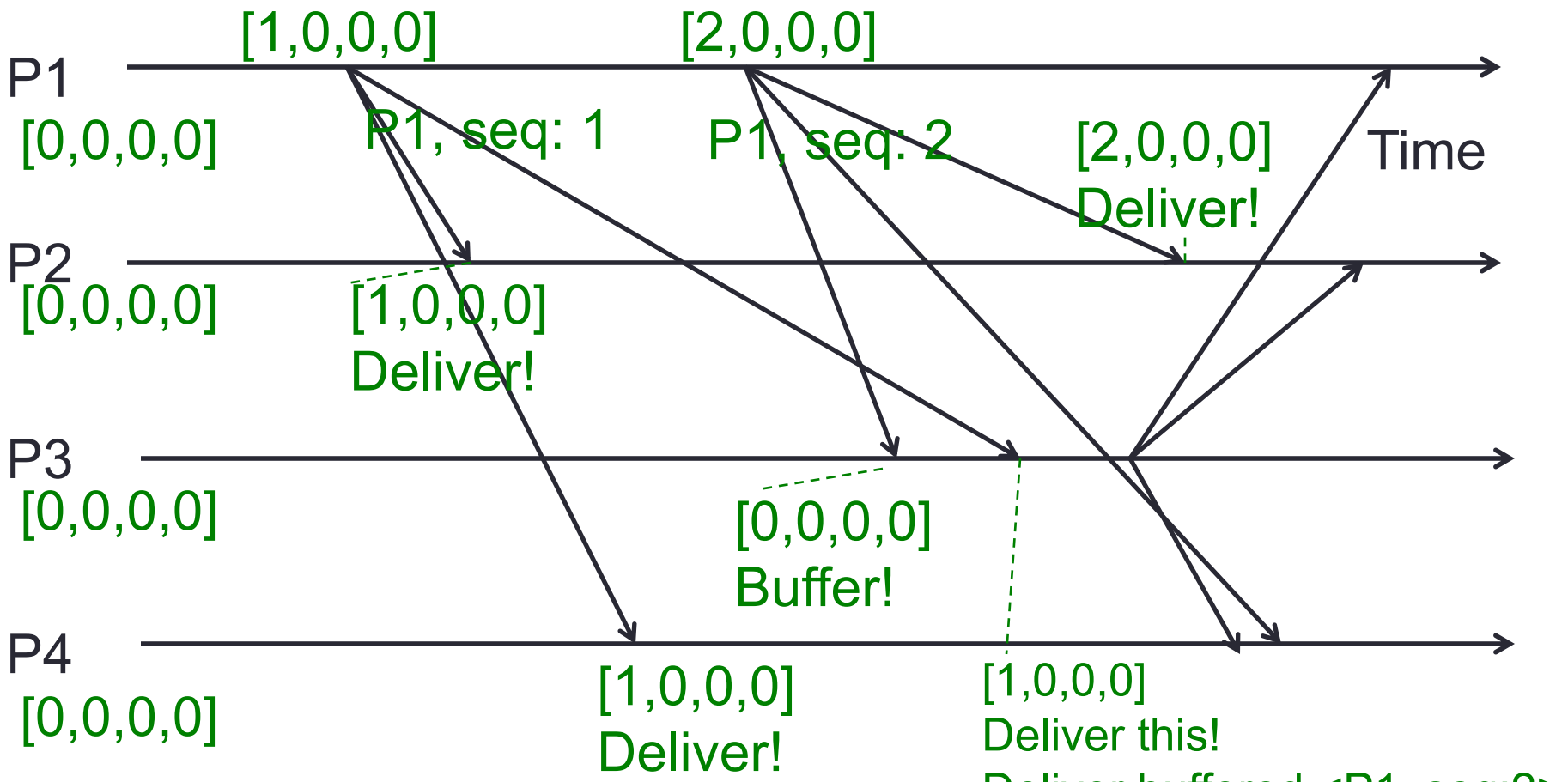
FIFO Ordering: Example



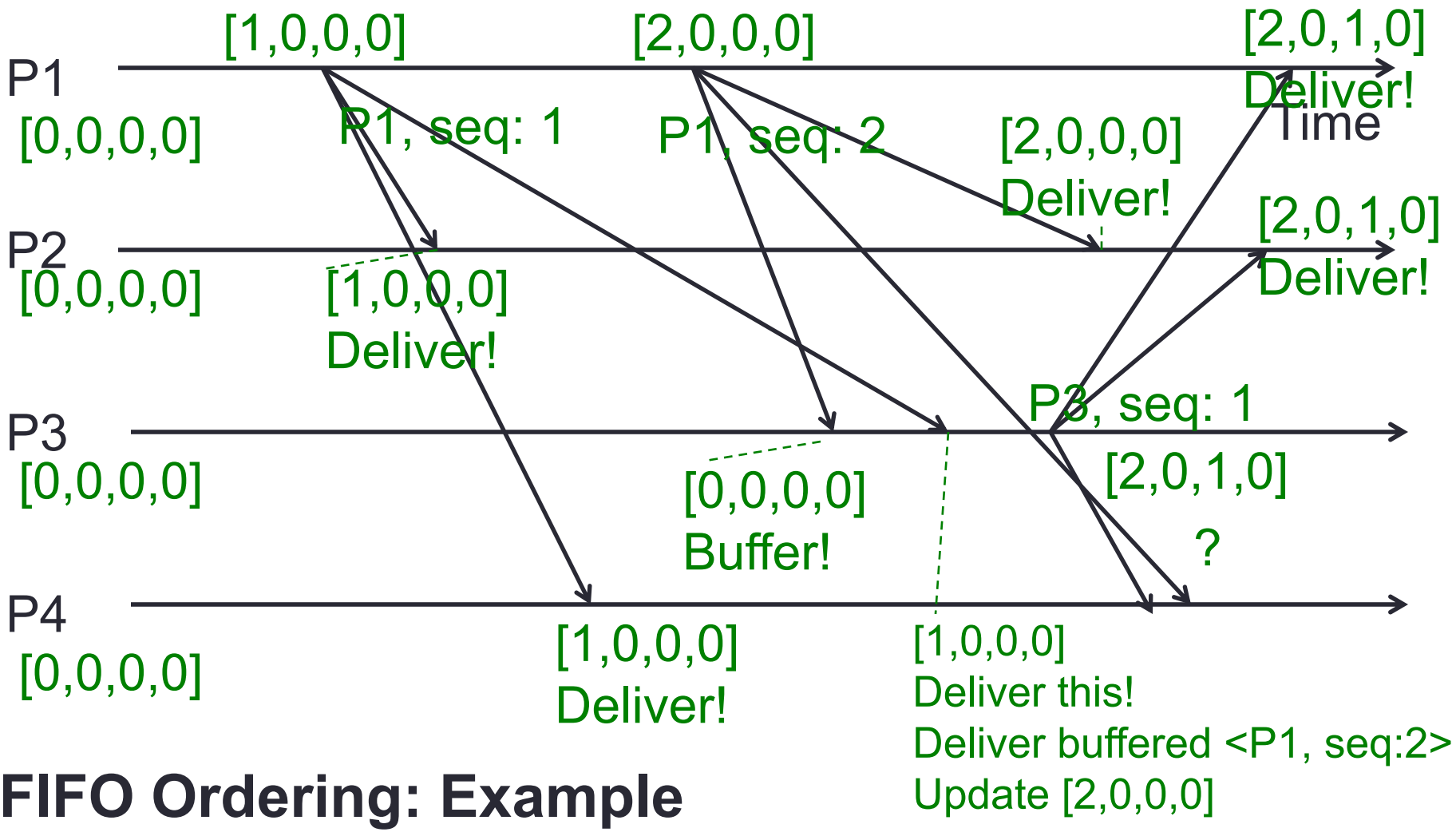
FIFO Ordering: Example



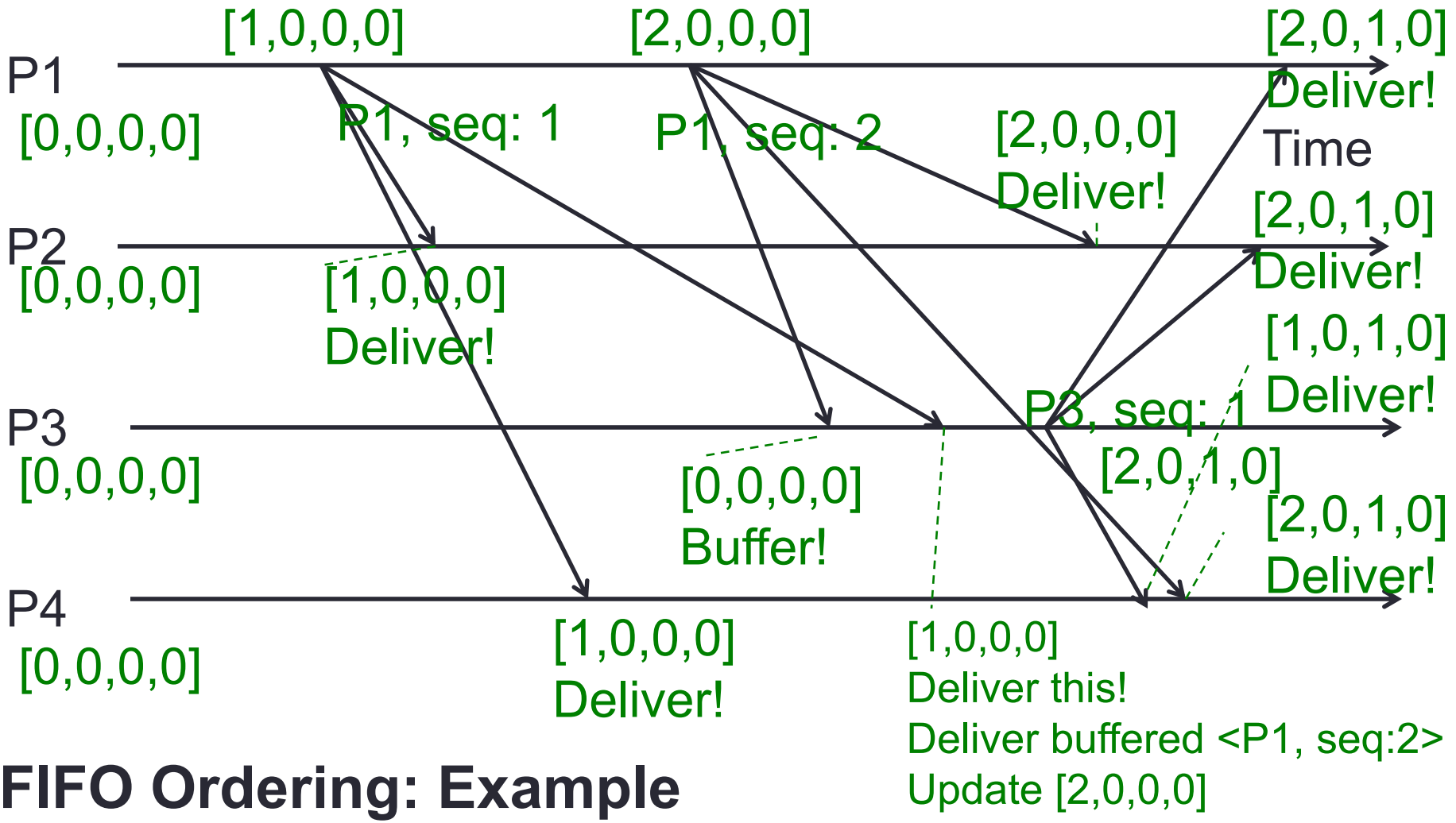
FIFO Ordering: Example



FIFO Ordering: Example



FIFO Ordering: Example



FIFO Ordering: Example

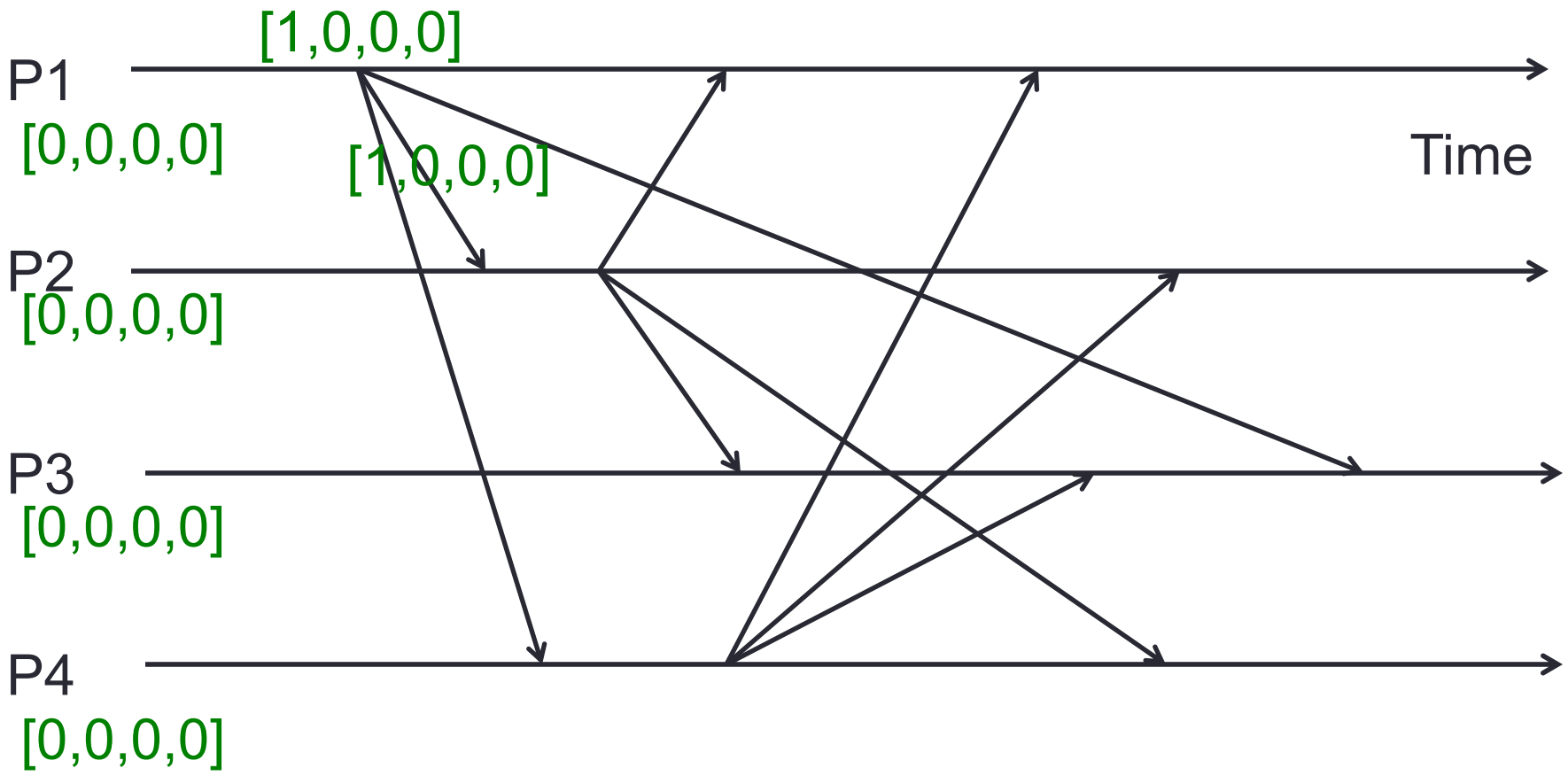
Causal ordering using vector timestamps

- 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*

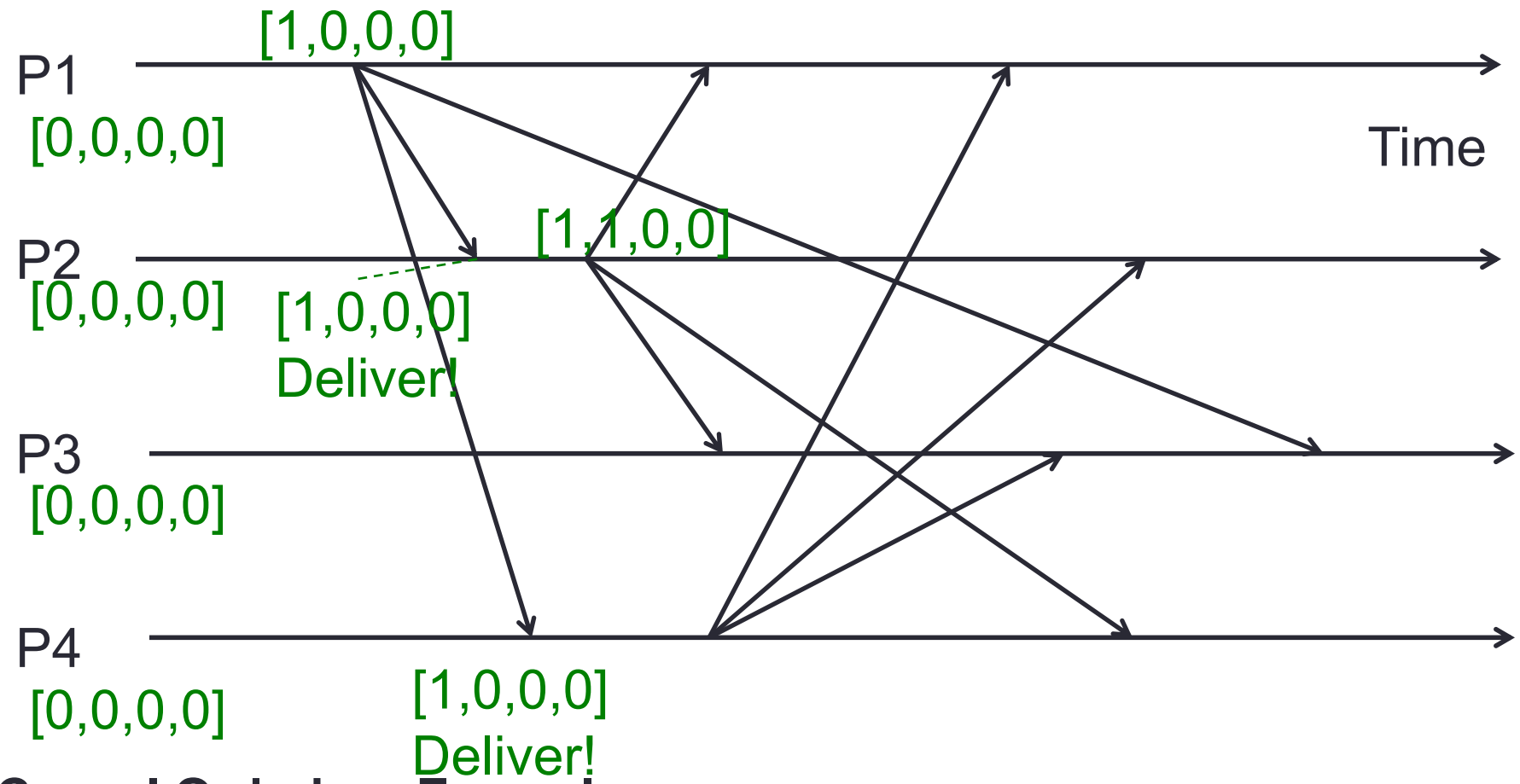
Causal ordering using vector timestamps

- Send multicast at process P*j*:
  - Set P*j*[*j*] = P*j*[*j*] + 1
  - Include new 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] (= P*j*[1…N]) in message, buffer it until:

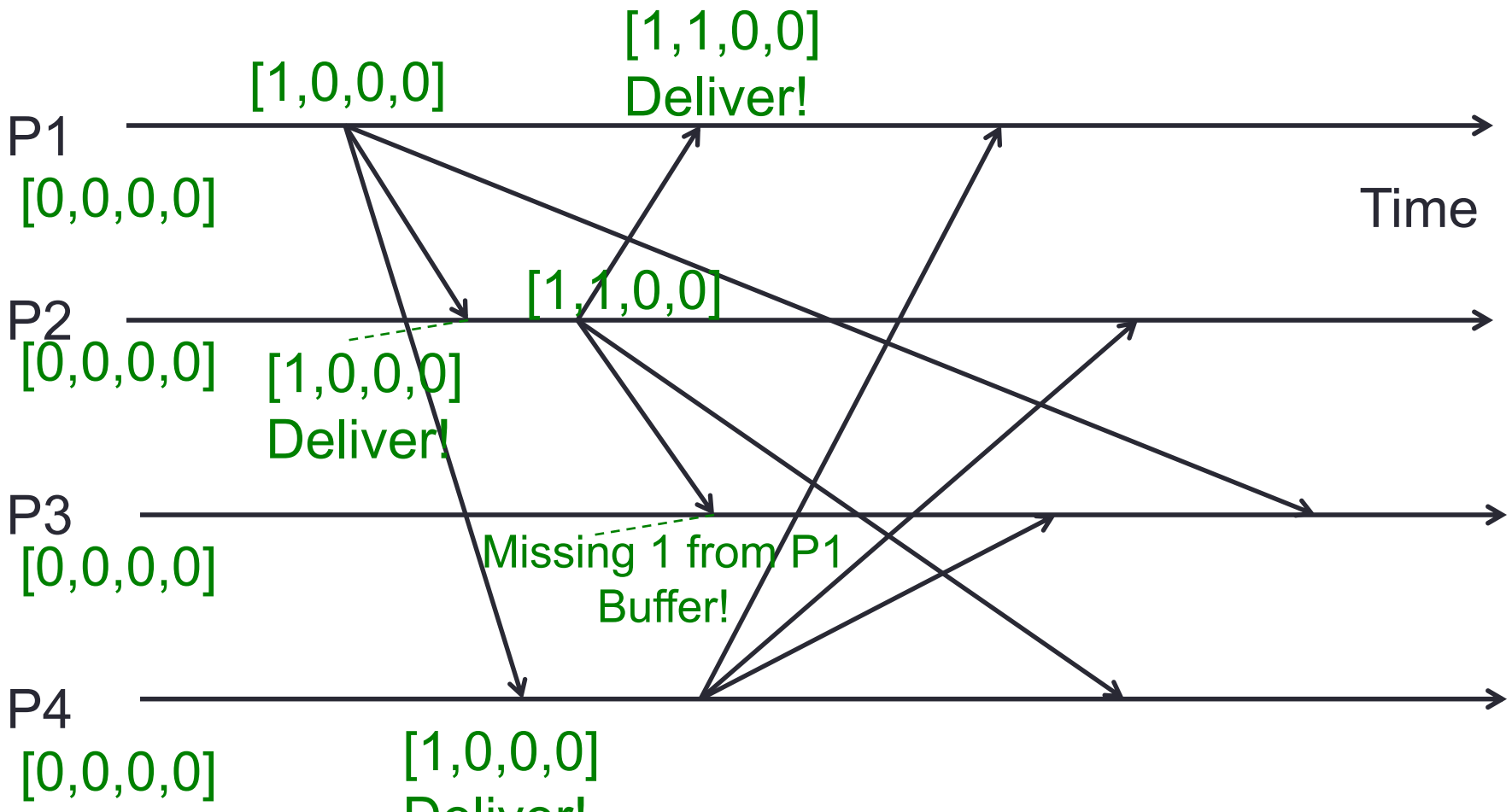
1. This message is the next one P*i* is expecting from P*j*, i.e., M[*j*] = P*i*[*j*] + 1
2. All multicasts, anywhere in the group, which happened-before M have been received at P*i*, i.e.,
   - For all $k \neq j$: M[*k*] ≤ P*i*[*k*]
   - i.e., ***Receiver satisfies causality***
3. When above two conditions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



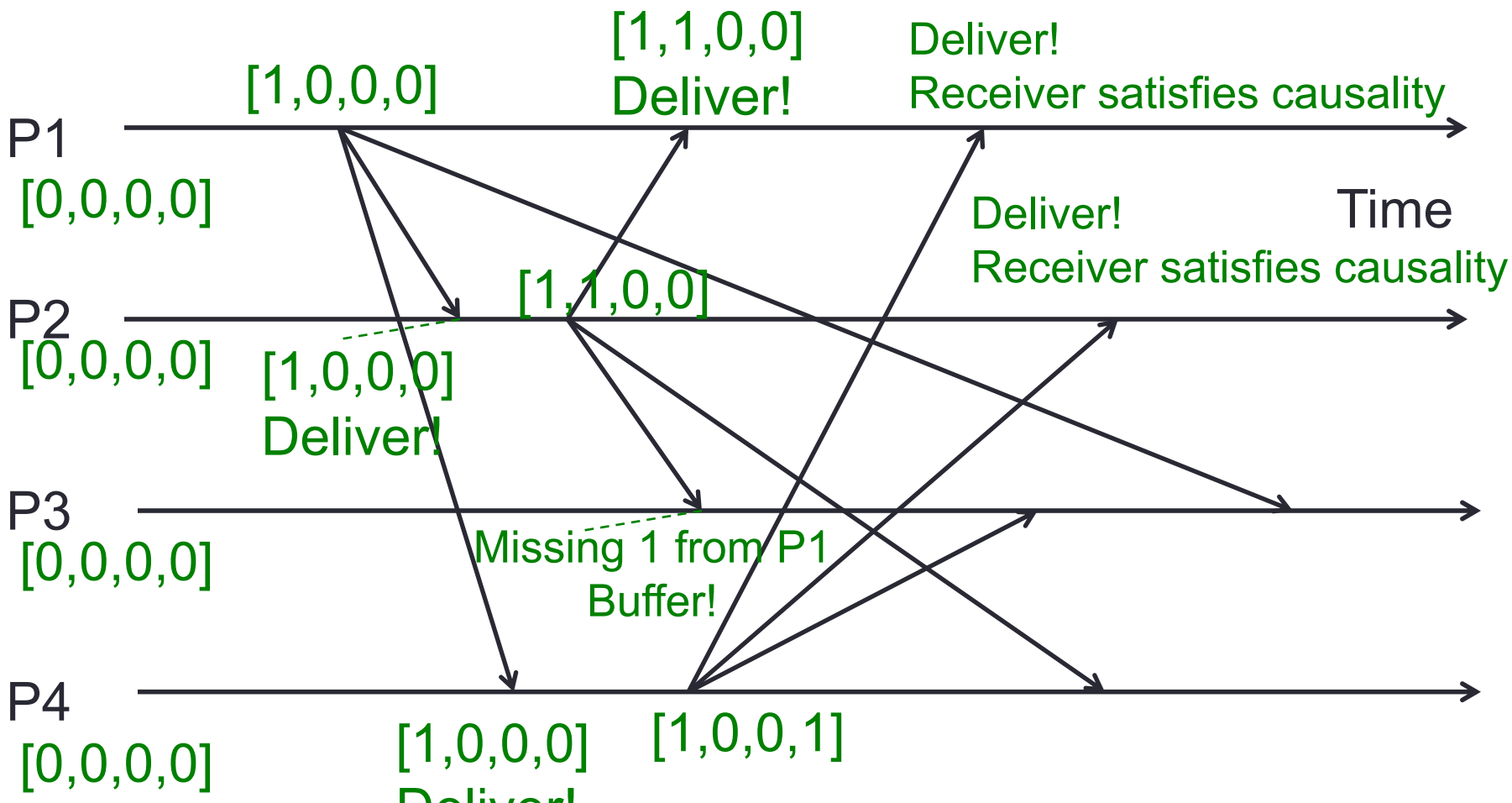
Causal Ordering: Example



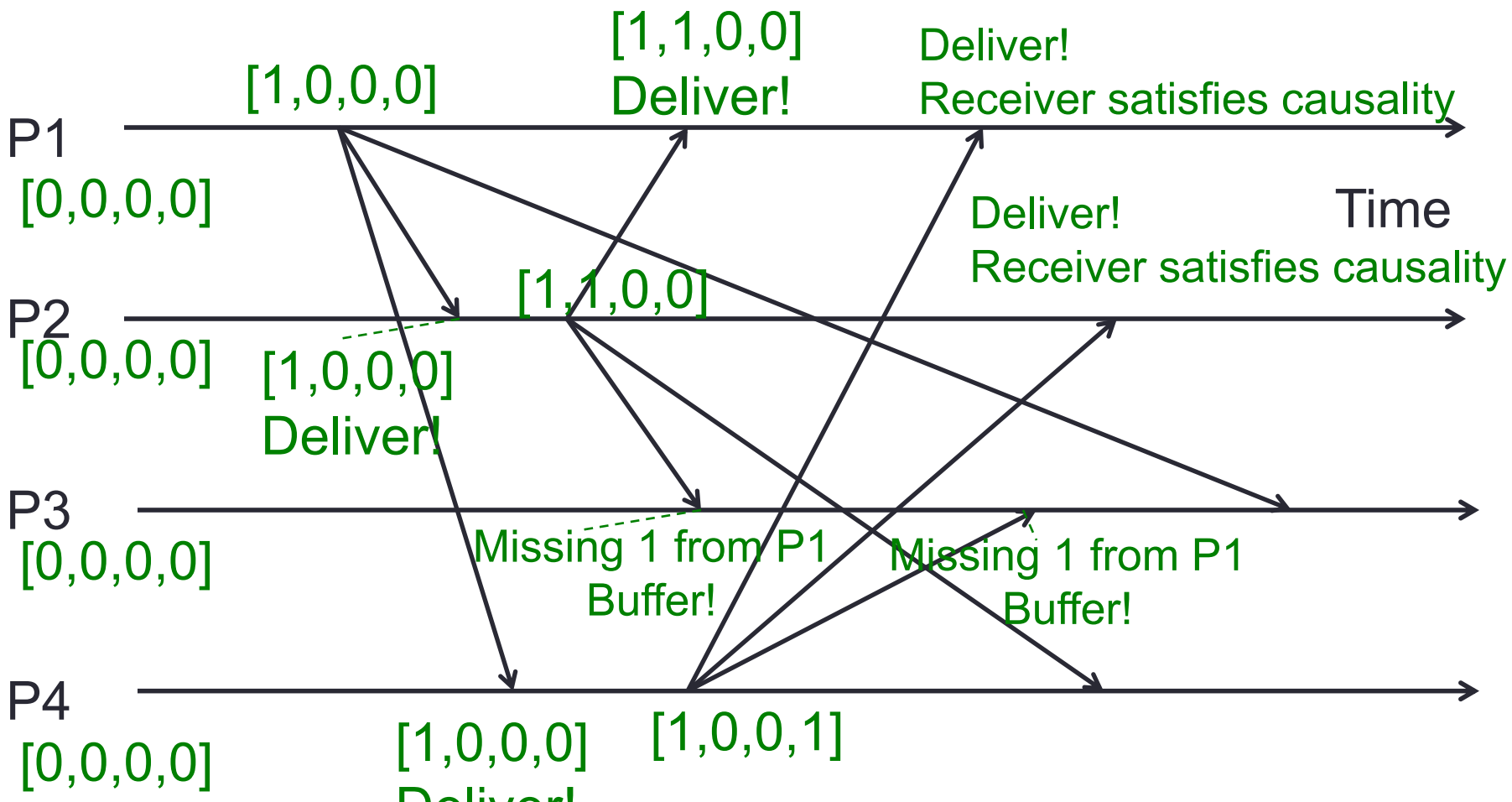
Causal Ordering: Example



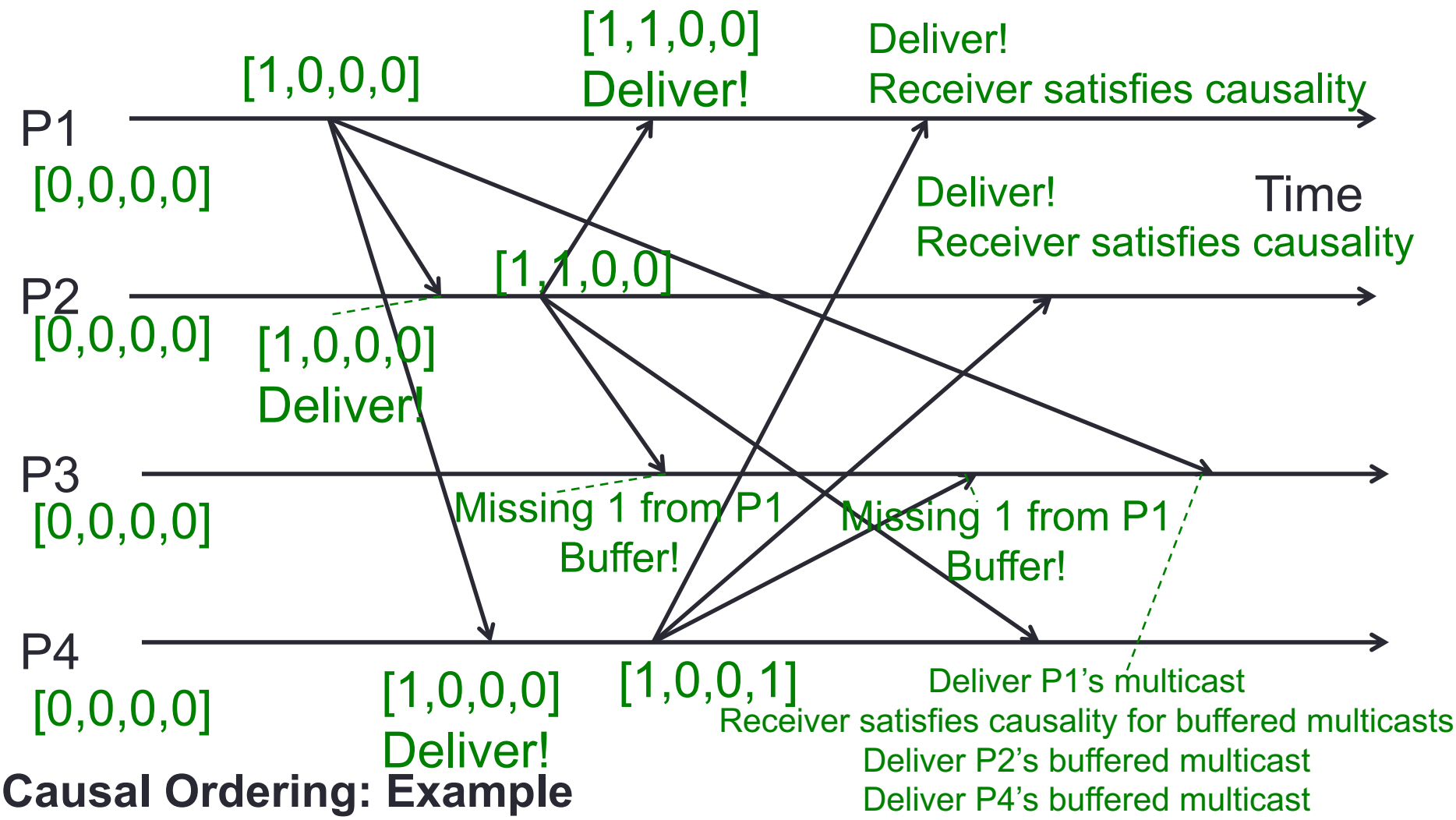
Causal Ordering: Example



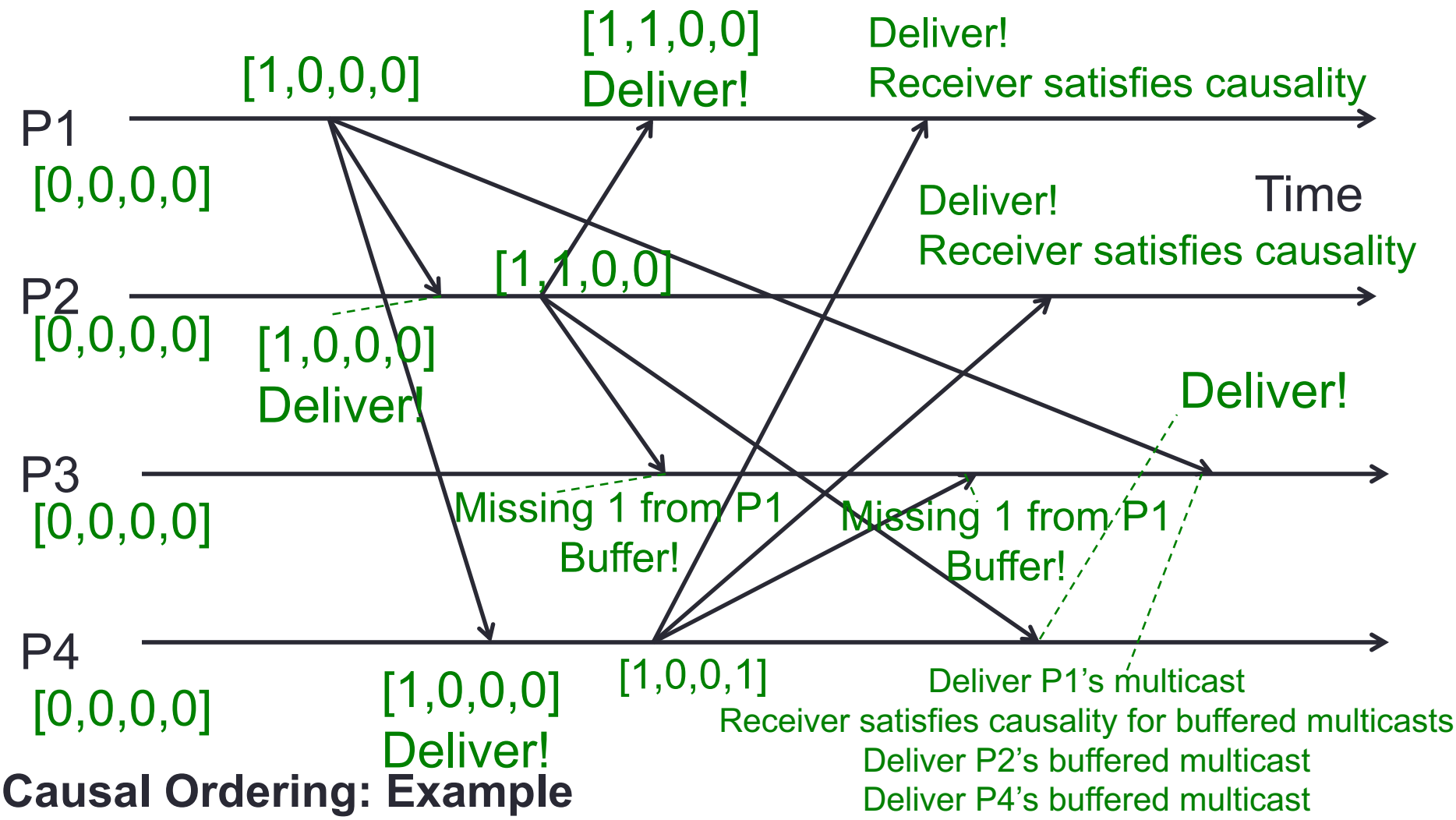
Causal Ordering: Example



Causal Ordering: Example



Causal Ordering: Example

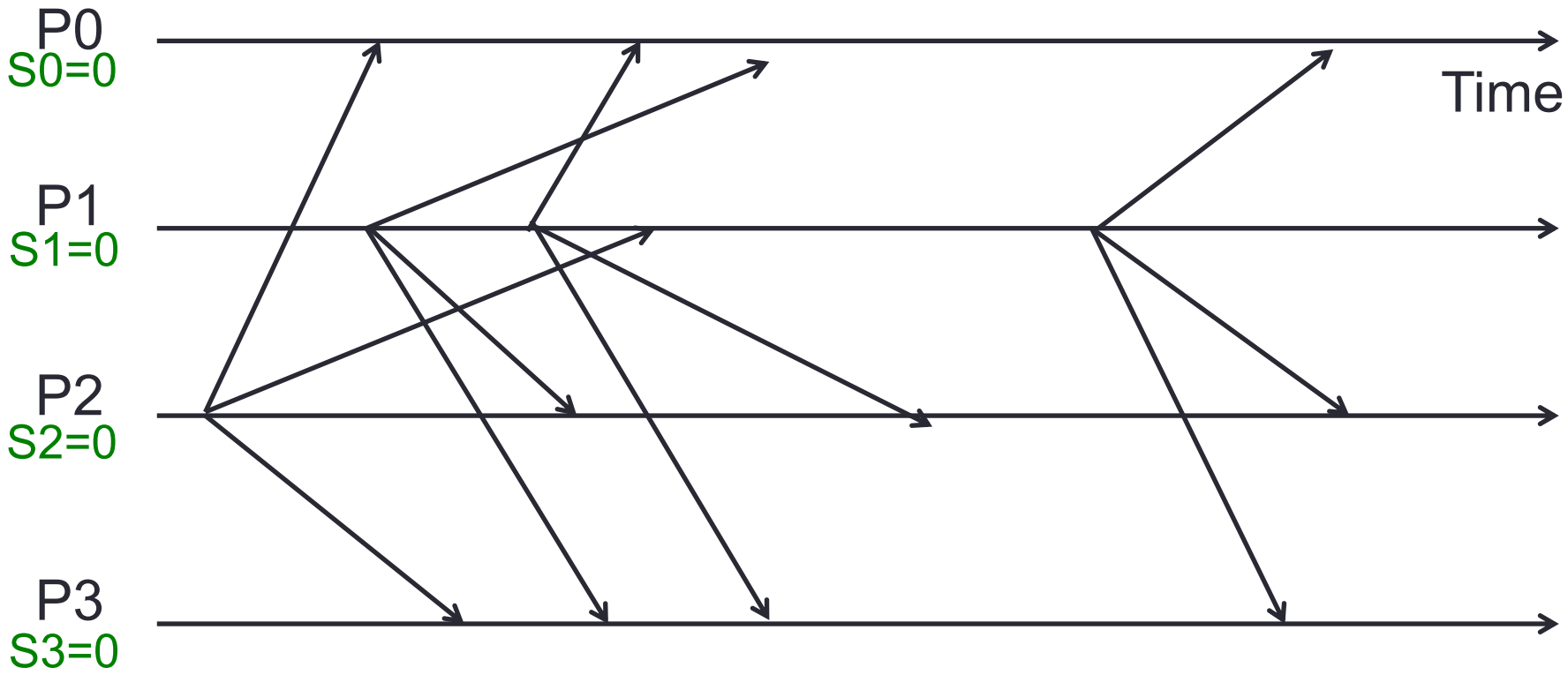


Causal Ordering: Example

Total ordering using a sequencer

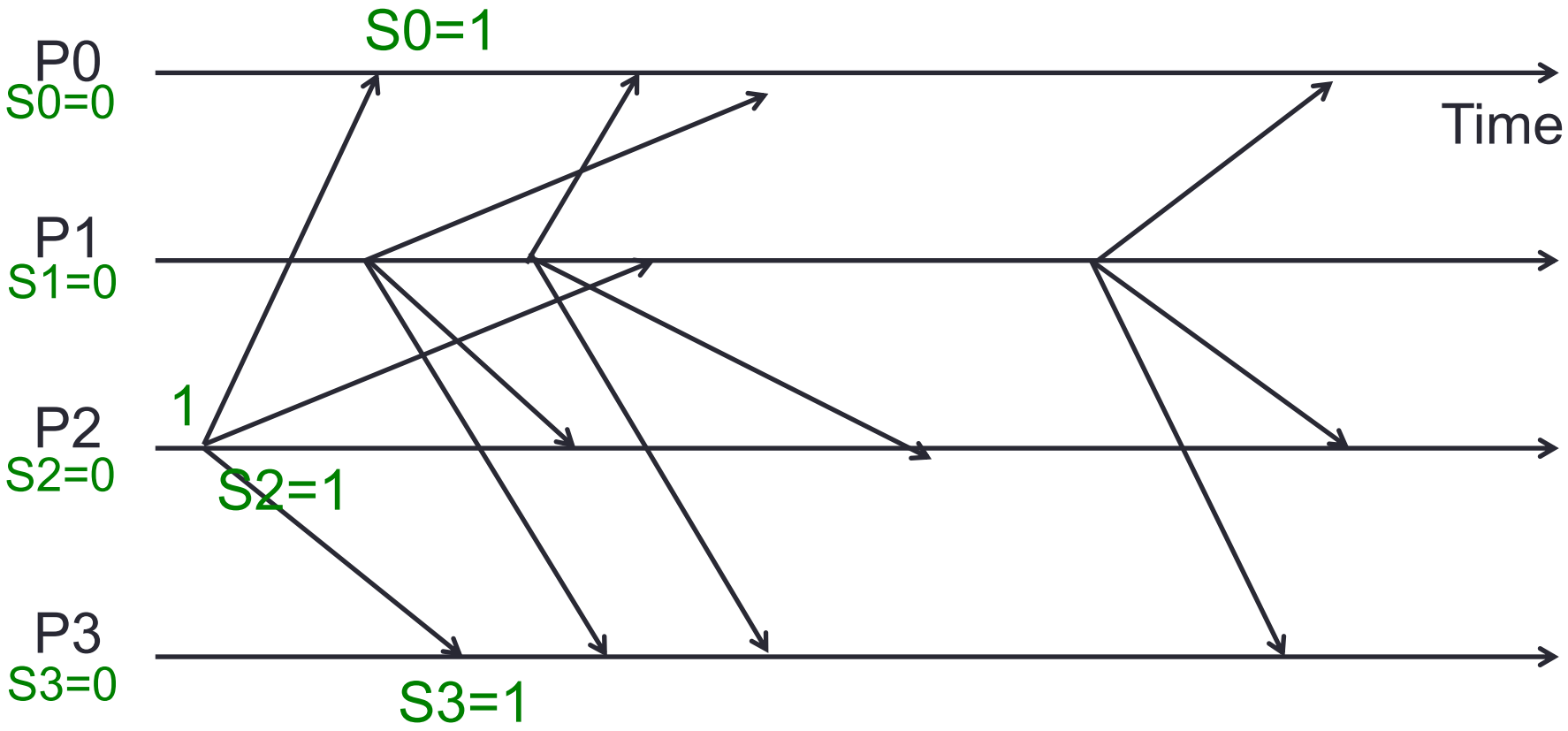
- Special process elected as leader or sequencer
- Send multicast at process P*i*:
  - Send multicast message M to group and sequencer
- Sequencer:
  - Maintains a global sequence number S (initially 0)
  - When it receives a multicast message M, it sets S = S + 1, and multicasts <M, S>
- Receive multicast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

Totally ordered multicast using a sequencer. Assuming that the sequencer receives the multicast instantaneously after the multicast is sent.



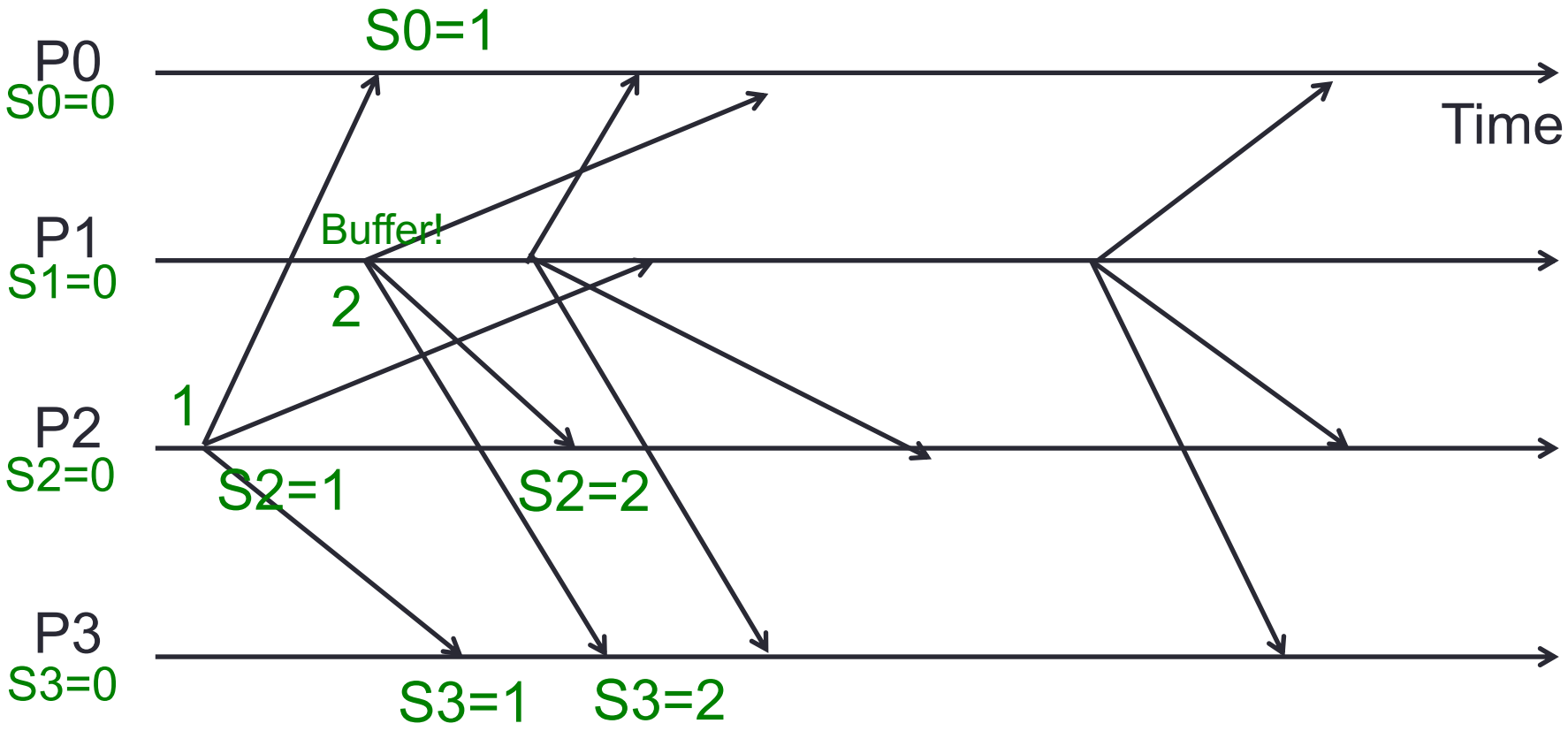
Total Ordering: Example

Totally ordered multicast using a sequencer. Assuming that the sequencer receives the multicast instantaneously after the multicast is sent.



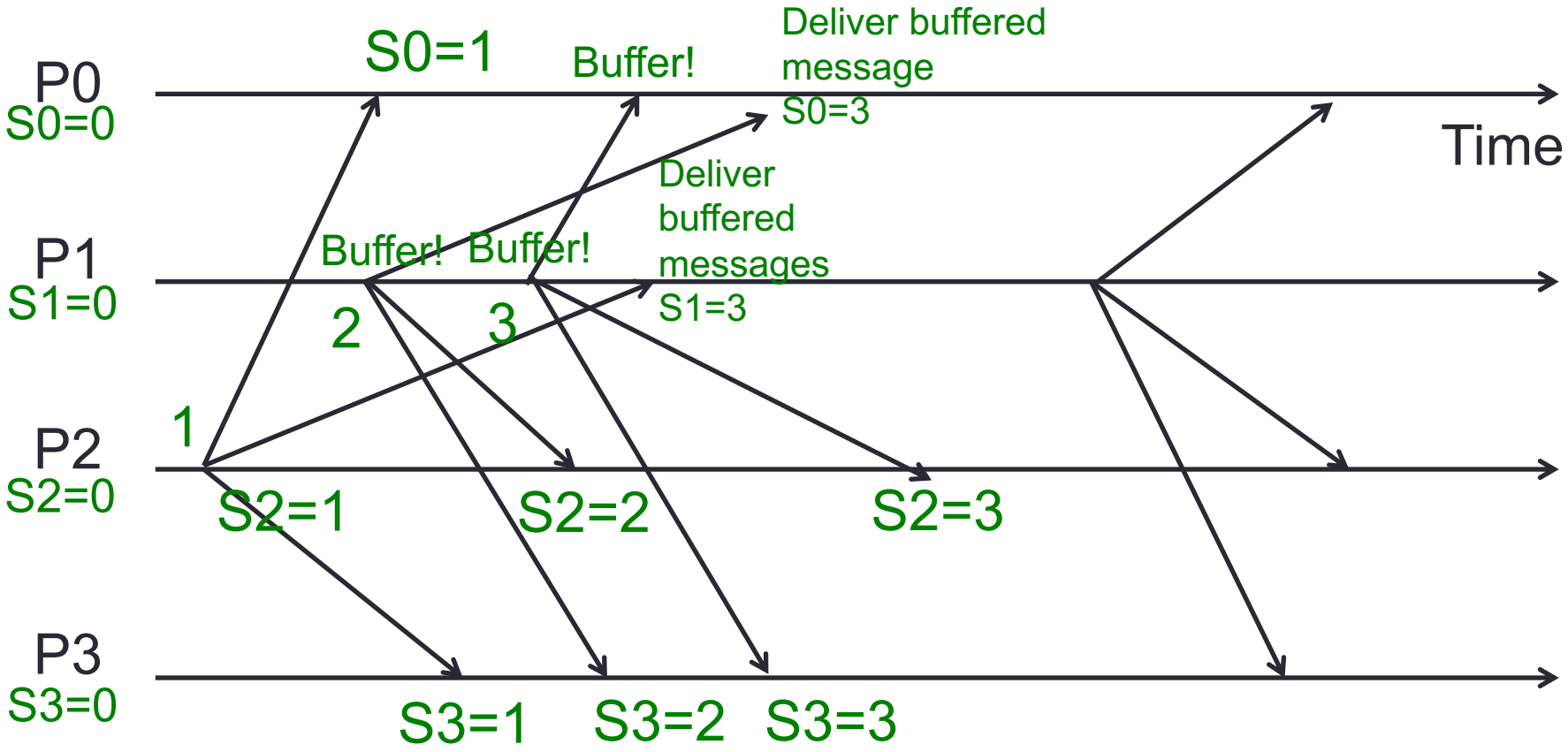
Total Ordering: Example

Totally ordered multicast using a sequencer. Assuming that the sequencer receives the multicast instantaneously after the multicast is sent.



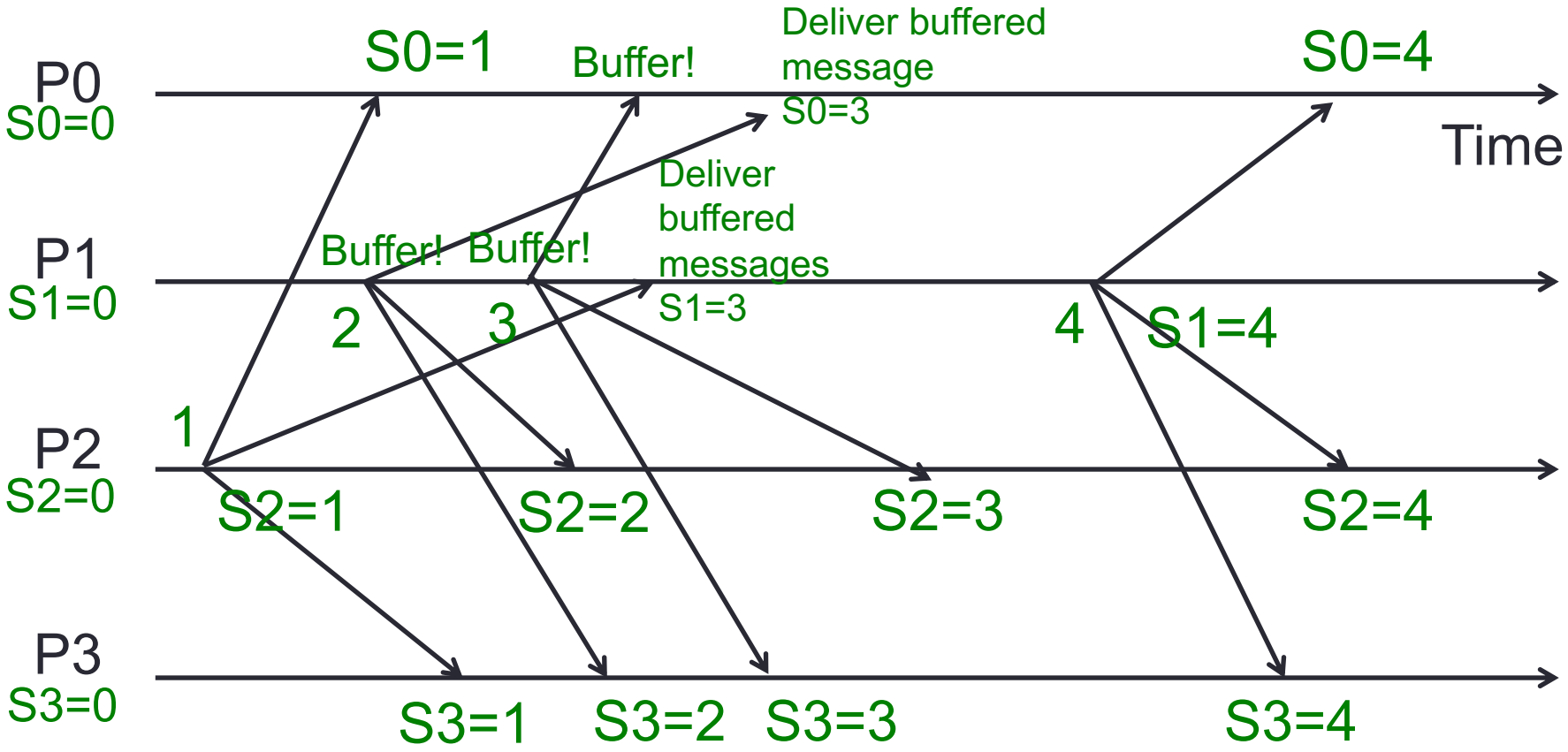
Total Ordering: Example

Totally ordered multicast using a sequencer. Assuming that the sequencer receives the multicast instantaneously after the multicast is sent.



Total Ordering: Example

Totally ordered multicast using a sequencer. Assuming that the sequencer receives the multicast instantaneously after the multicast is sent.



Total Ordering: Example

Hybrid variants

- FIFO and causal orderings are orthogonal to total ordering
- We 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

- We have multicast that can deliver messages to all processes in a group in (somewhat) consistent order
- Still need to ensure all processes receive the message
- And handle group membership changes, e.g., join, leave, failure

Reliable multicast

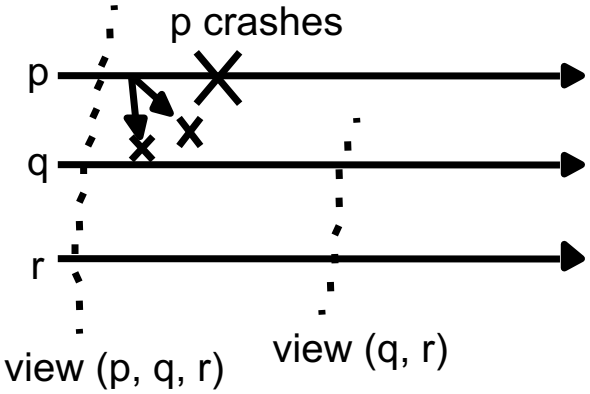
- All non-faulty processes who do not join/leave during communication receive the message
  - Multicast group membership can change
  - View-synchronous multicast
- Need to handle sender failure after sending to a subset of the group
- Need to retransmit lost messages
- Can be built upon reliable unicast
  - e.g., using reliable transport layer protocol, TCP

View-synchronous multicast

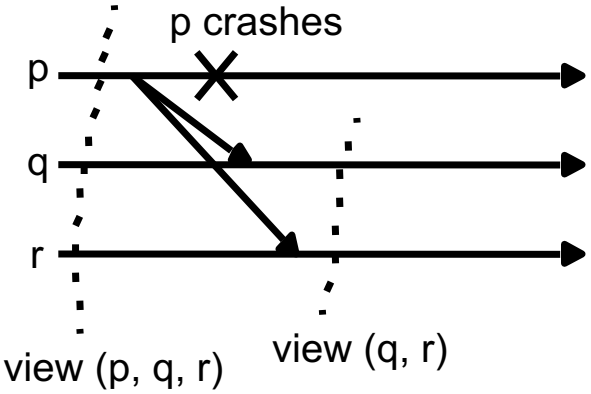
- A view reflects ***current membership of group***
- A view is **delivered** when a membership change occurs and the application is notified of the change
  - Receiving a view is different from delivering a view
    - All members have to agree to the delivery of a view
- View-synchronous multicast
  - The delivery of a new view draws a conceptual line across the system, and every message is either delivered on one side or the other of that line

View-synchronous multicast

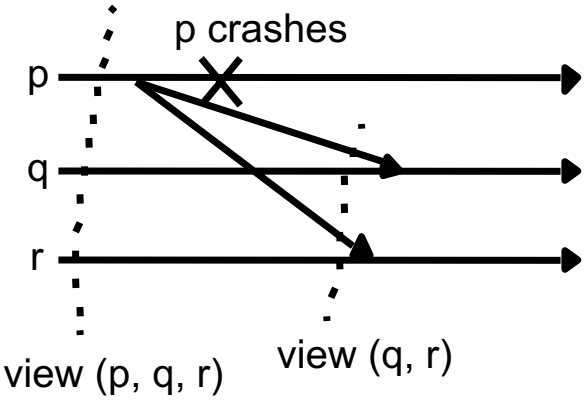
a (allowed).



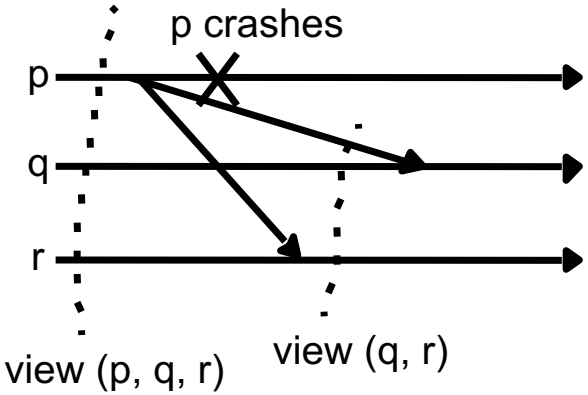
b (allowed).



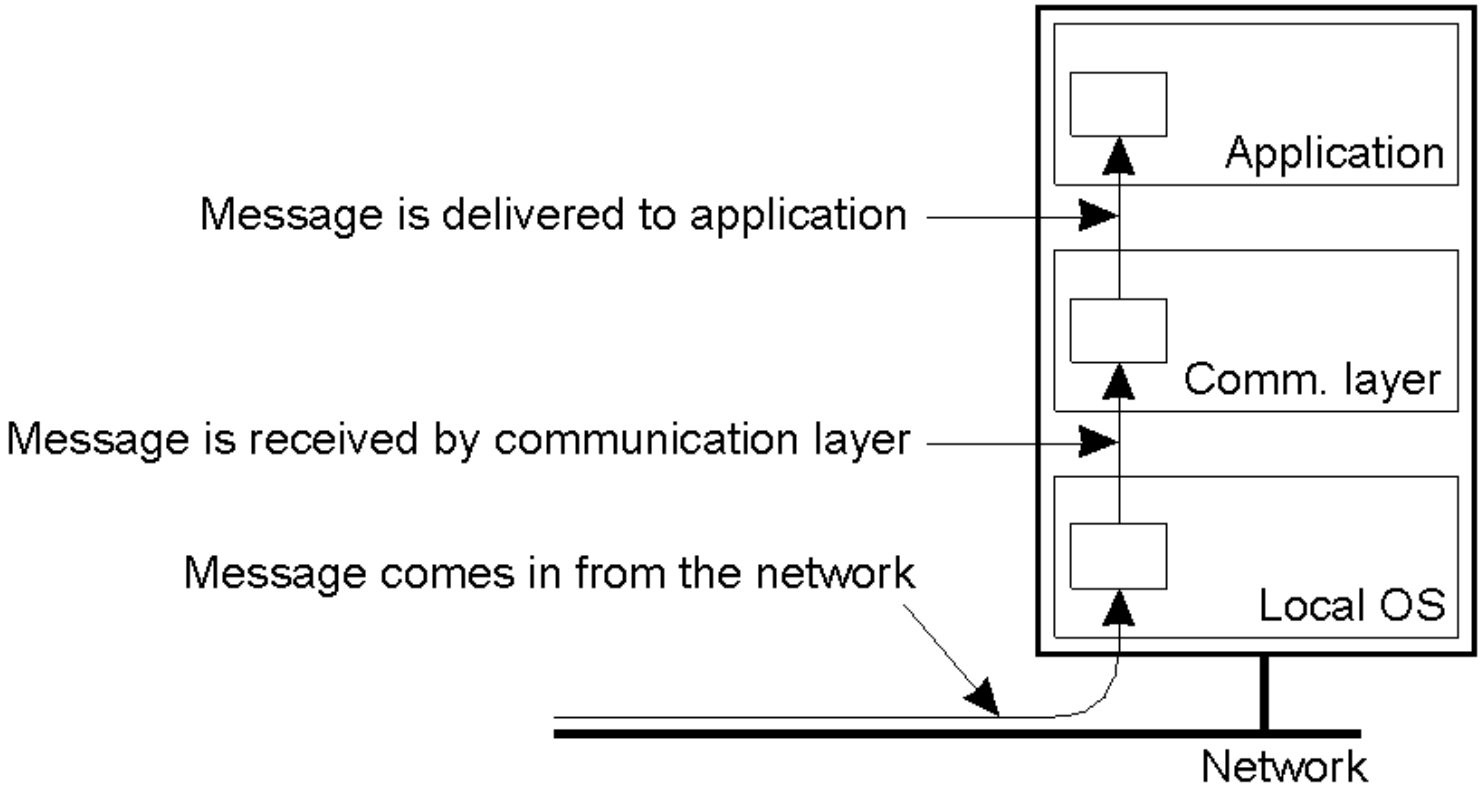
c (disallowed).



d (disallowed).



View-synchronous multicast implementation



View-synchronous multicast implementation

- Only stable messages are delivered
- **Stable message**: a message received by all processes in the message's group view

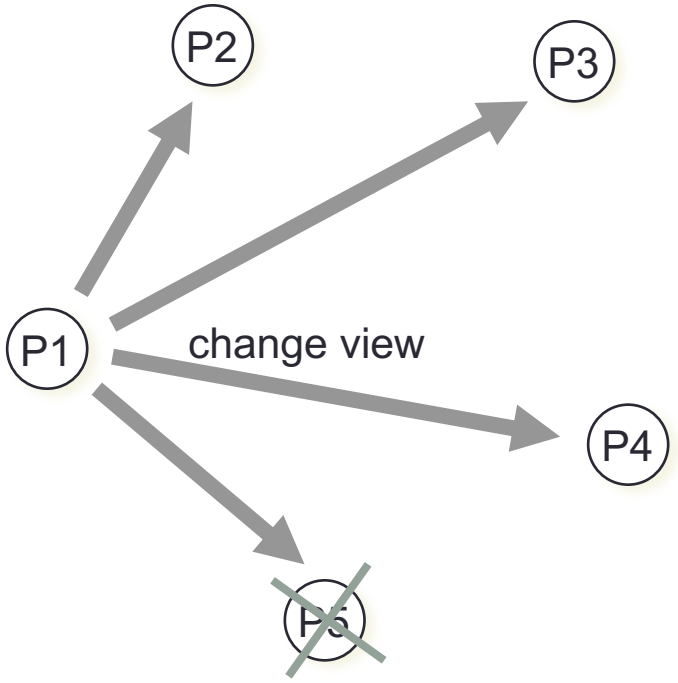
- Assumptions (can be ensured by using TCP):
  - Point-to-point communication is reliable
  - Point-to-point communication ensures FIFO-ordering

Implementation: receiving all messages

- Make sure each process in G_i has received all messages that were sent to G_i
  - A sender may have failed → there may be processes that will not receive a message *m*
  - These processes should get *m* from somewhere else
- Let every process hold *m* until it knows that all members of G_i received it
  - Once all members received it, m is stable
  - Only stable messages are delivered
  - Select an arbitrary process in G_i and request it to send *m* to all other processes
    - Delivery within the group is reliable, so this ensures that the message is stable

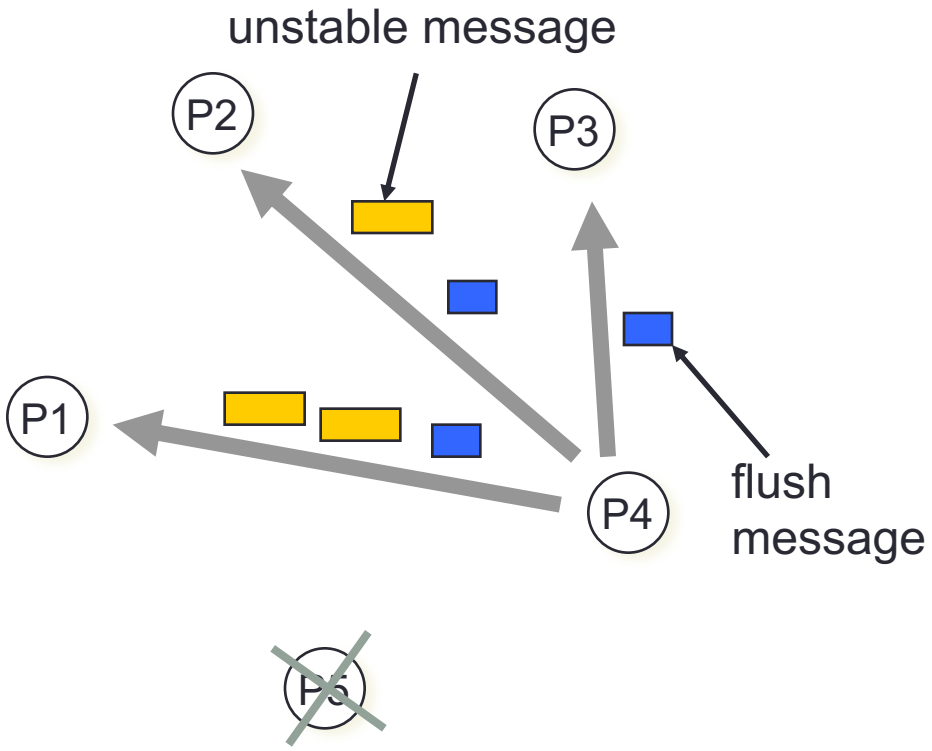
Implementation: view change

- G_i = {P1, P2, P3, P4, P5}
- P5 fails
- P1 detects that P5 has failed
- P1 send a "view change" message to every process in G_{i+1} = {P1, P2, P3, P4}



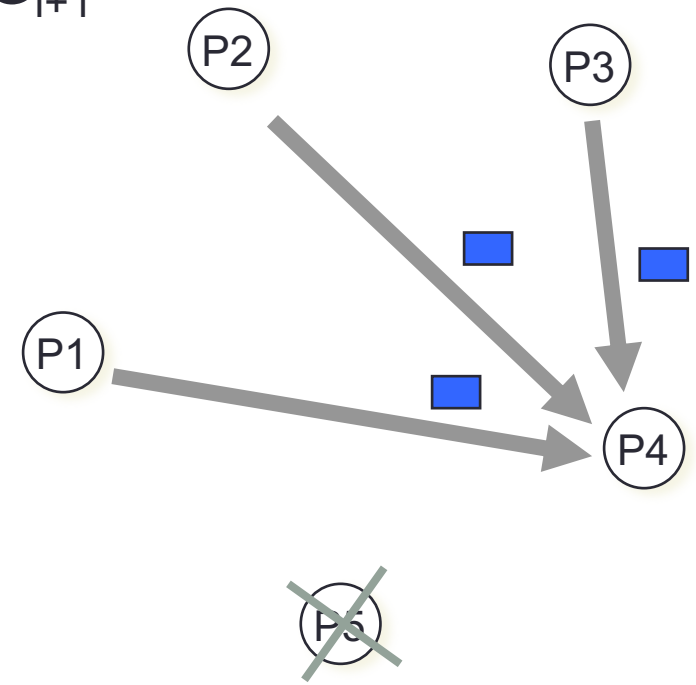
Implementation: view change

- Every process
  - Send each unstable message *m* from G_i to members in G_{i+1}
  - Marks *m* as being stable
  - Send a flush message to mark that all unstable messages have been sent



Implementation: view change

- Every process
  - After receiving a flush message from **every** process in $G_{i+1,}$ it installs G_{i+1}



Reading

- Sections 15.4 and 18.2 of CBook
- Section 8.4 of TBook

Acknowledgement

- These slides contain material developed and copyrighted by Professor Indranil Gupta (UIUC).