Computer Science 425
Distributed Systems
(Fall 2009)

Lecture 5

Multicast Communication

Reading: Section 12.4

Klara Nahrstedt

Acknowledgement

- **The slides during this semester are based on ideas and material from the following sources:**
  - **Slides prepared by Professors M. Harandi, J. Hou, I. Gupta, N. Vaidya, Y-Ch. Hu, S. Mitra.**
  - **Slides from Professor S. Gosh's course at University o Iowa.**

Administrative

- **Homework 1 posted**
  - **Deadline, September 17 (Thursday)**
- **MP1 posted today**
  - **Deadline, September 25, Friday**

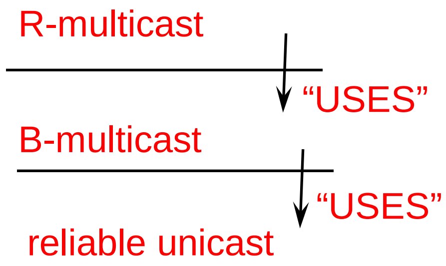
Plan for Today

- Reliable Multicast
- Ordered Multicast
  - Total ordering
  - Causal ordering
  - FIFO ordering

Reliable Multicast

- ***Integrity***: **A *correct* (i.e., non-faulty) process *p* in a *group(m) delivers* a multicast message *m* at most once.**
  - **Safety property: any message delivered is identical to the one that was sent**
- ***Validity***: **If a correct process multicasts (sends) message *m*, then it will eventually deliver *m*.**
  - **Guarantees liveness to the sender.**
  - **Liveness property: any message is eventually delivered to destination**
- ***Agreement***: **If a correct process delivers message *m*, then all the other correct processes in *group(m)* will eventually deliver *m*.**
  - **Property of "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 Algorithm



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

Reliable Multicast Algorithm (R-multicast)

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

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

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

if <u>some</u> correct process B-multicasts a message *m*, then, all correct processes deliver *m* too. If no correct process B-multicasts *m*, then no correct processes deliver *m*.

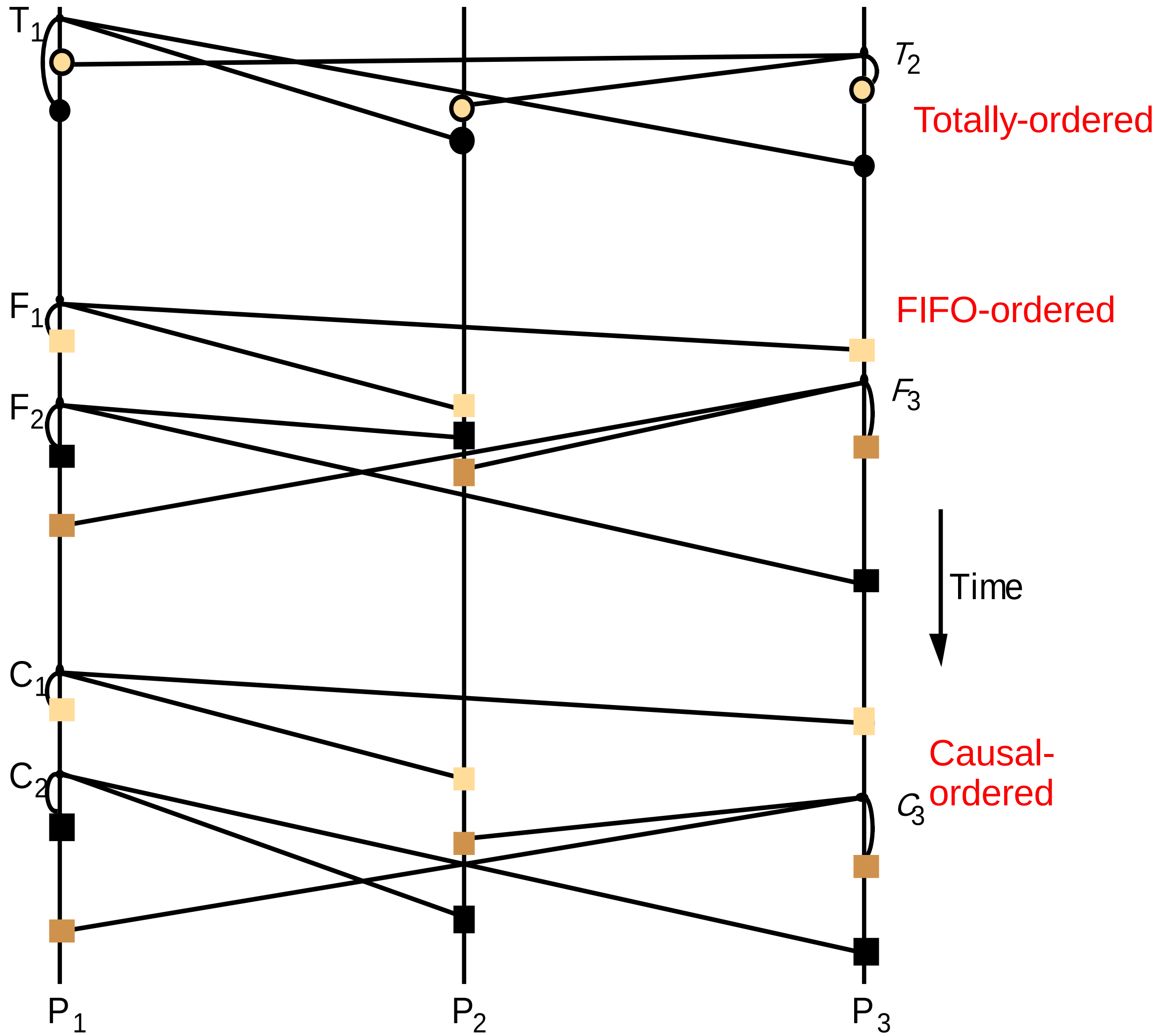
Ordered Multicast

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

Total, FIFO and Causal Ordering

- **Totally ordered** messages T_1 and T_2.
- **FIFO-related** messages F_1 and F_2.
- **Causally-related** messages C_1 and C_3

- Causal ordering implies FIFO ordering
- Total ordering does not imply causal ordering.
- Causal ordering does not imply total ordering.
- Hybrid mode: causal-total ordering, FIFO-total ordering.



Example: Display From Bulletin Board Program

User 1

Post to Bulletin Board

User 2

Post to Bulletin Board

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		

What is the most appropriate ordering for this application?
(a) FIFO (b) causal (c) total

FIFO-ORDERED MULTICAST

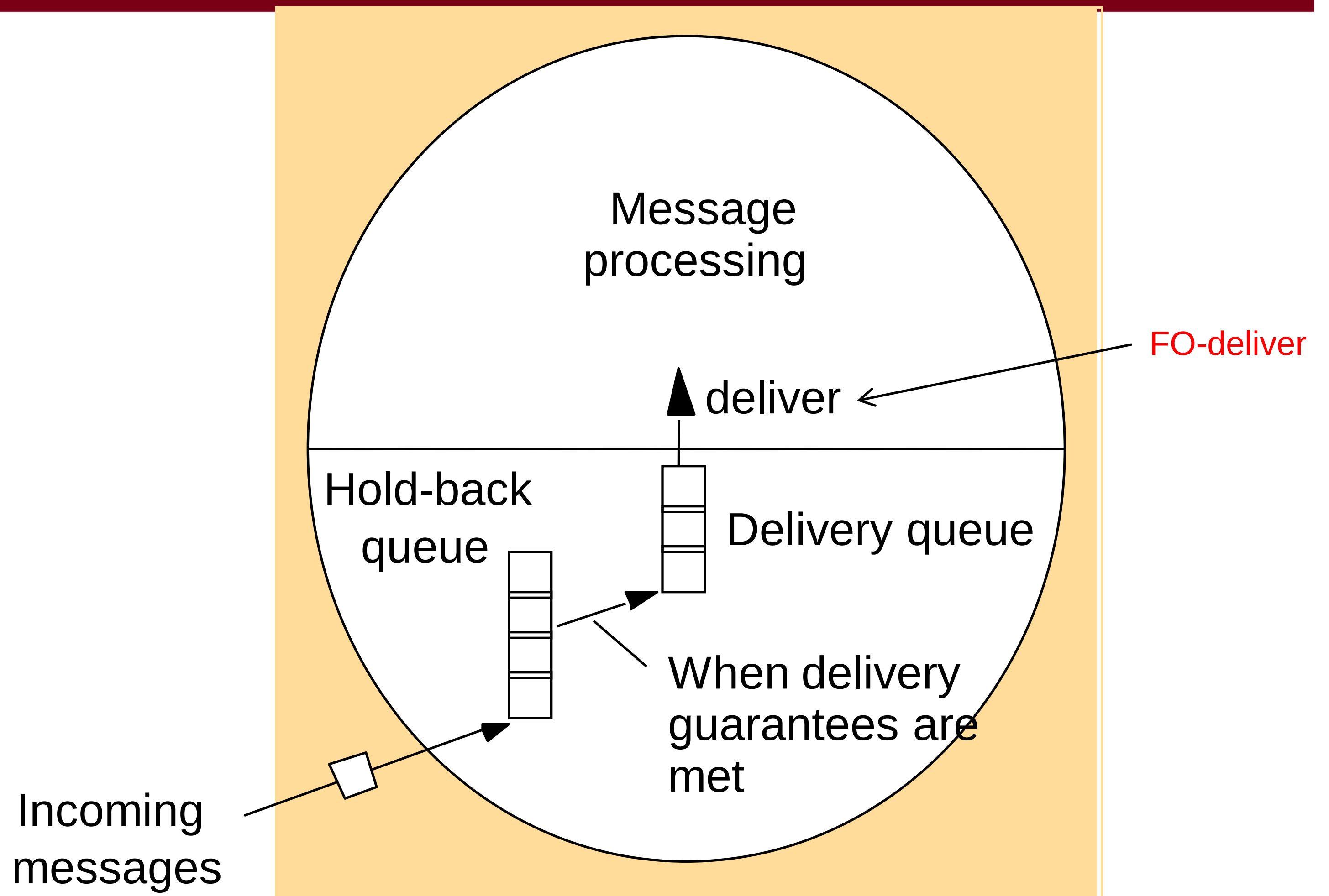
Providing Ordering Guarantees (FIFO)

- **Process messages from each process in the order they were sent:**
  - **Each process keeps a sequence number for each other process.**
  - **Messages are sent with local sequence number**
  - **When a message is received,**

If Message# is

- **as expected (next sequence), accept**
- **higher than expected, buffer in a queue**
- **lower than expected, reject**

Hold-back Queue for Arrived Multicast Messages: received yet undelivered messages

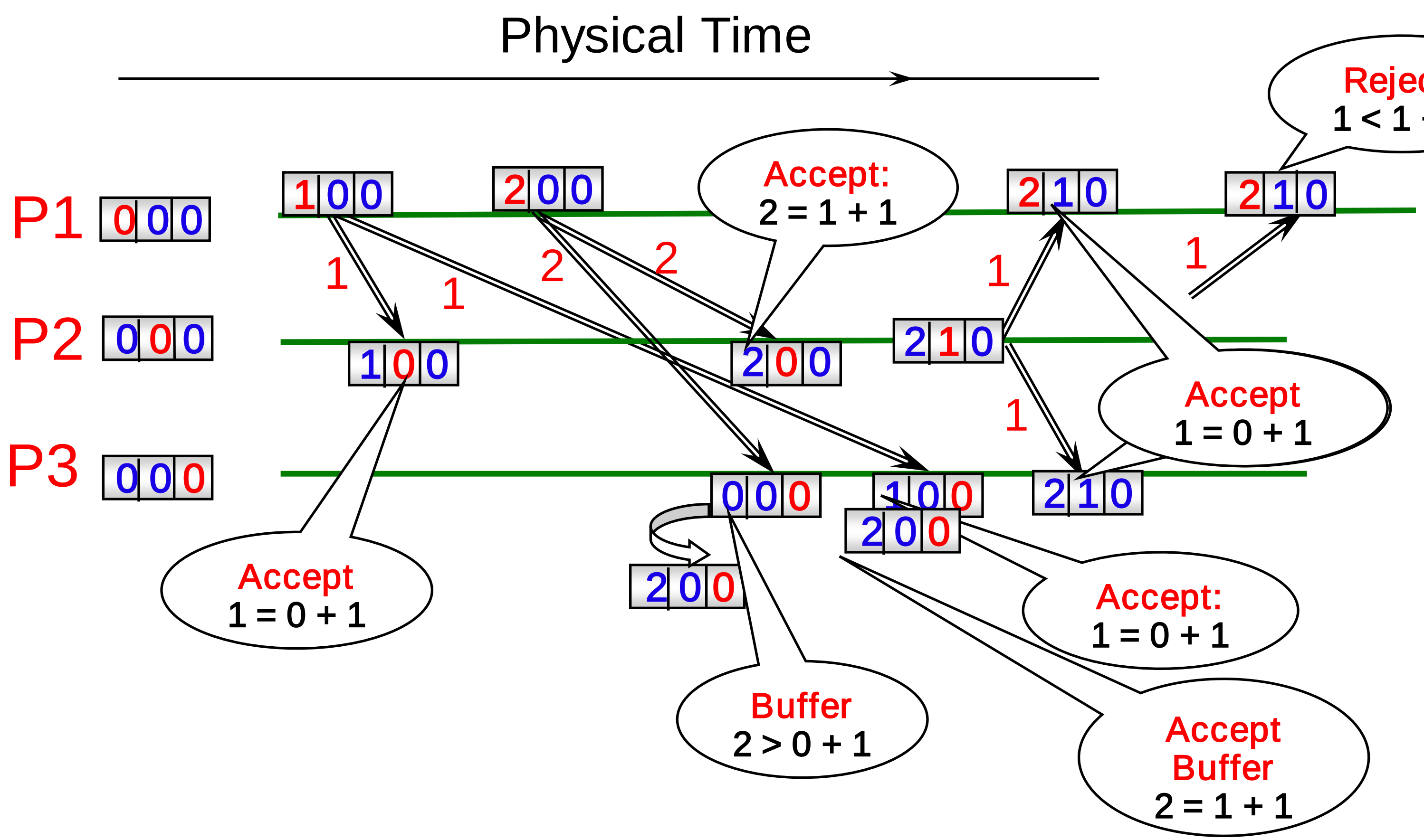


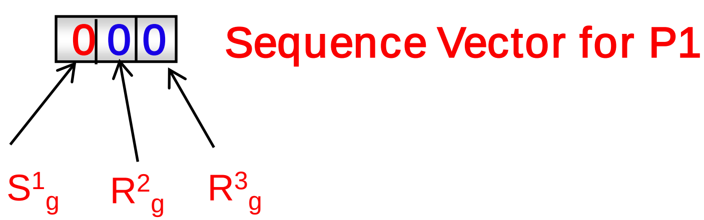
Implementing FIFO Ordering (FIFO-ordered multicast)

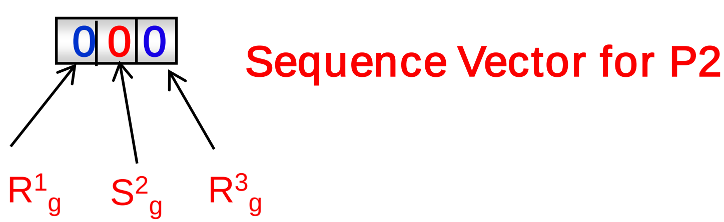
- **S^p_g: count of messages *p* has sent to *g*.**
- **R^q_g: the recorded sequence number of the latest message that *p* has delivered from *q* to the group *g*.**
- **For *p* to FO-multicast *m* to *g***
  - ***p* increments S^p_g by 1**
  - **p "piggy-backs" the value S^p_g *onto the message.***
  - ***p B-multicasts m to g.***
- **At process *p*, upon receipt of *m* from *q* with sequence number *S*:**
  - ***p* checks whether $S = R^q_g+1$. If so, p FO-delivers *m* and increments R^q_g**
  - **If $S > R^q_g+1$, *p* places the message in the hold-back queue until the intervening messages have been delivered and $S = R^q_g+1$.**
  - ***If* $S < R^q_g+1$, then drop the message (we have already seen the message)**

Example: FIFO Multicast

*(do **NOT** confuse with vector timestamps)*







CAUSAL-ORDERED MULTICAST

Causal Multicast

- **Let us focus on multicast group** *g*
- **Each process** *iєg* **maintains a vec*tor*** V^g_i **of length** |*g*| **where**
  - $V^g_i[j]$ **counts the number of group** *g* **messages from** ***j to i***
- **Messages multicast by process** *i* **are tagged with the vector timestamp** V^g_i
- *Recall rule for* ***receiving vector timestamps***

$$V_{receiver}[j] = \begin{cases} Max(V_{receiver}[j], V_{message}[j]), & \text{if } j \text{ is not self} \\ V_{receiver}[j] + 1 & \text{otherwise} \end{cases}$$

- **i.e. when process** *i* **receives a** $<m, V^g_j>$ **from** *j*, **then**
  - $V^g_i[k] = max(V^g_i[k], V^g_j[k])$ *if* $k \neq i$
  - $V^g_i[k] = V^g_i[k] + 1$ *if* $k = i$
- *Remember V(a) < V(b) iff a happens before b*

Causal Ordering using vector timestamps

Algorithm for group member p_i $(i = 1, 2 \ldots, N)$

On initialization

$V_i^g[j] := 0 \; (j = 1, 2 \ldots, N);$

The number of group-g messages from process j that have been seen at process i so far

To CO-multicast message m to group g

$V_i^g[i] := V_i^g[i] + 1;$

B-multicast$(g, <V_i^g, m>);$

On B-deliver$(<V_j^g, m>)$ *from* p_j, *with* $g = group(m)$

place $<V_j^g, m>$ in hold-back queue;

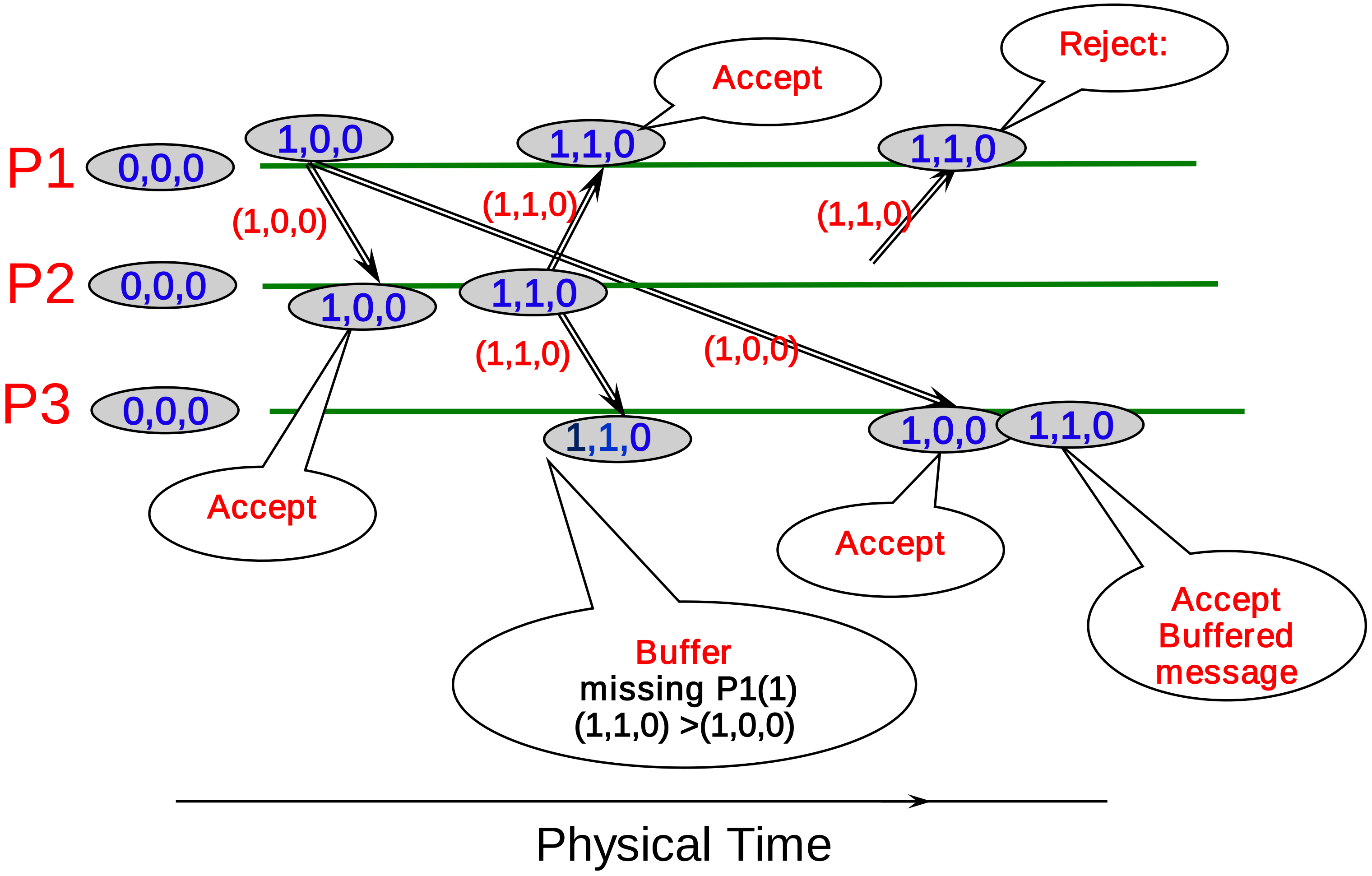
wait until $V_j^g[j] = V_i^g[j] + 1$ and $V_j^g[k] \le V_i^g[k] \; (k \ne j)$;

Guarantees Causal ordering

CO-deliver m; // after removing it from the hold-back queue

$V_i^g[j] := V_i^g[j] + 1;$

Example: Causal Ordering Multicast



TOTAL-ORDERED MULTICAST

1st Method - Using Sequencer

- **Delivery algorithm similar to FIFO**
- **Except that processes maintain group specific sequence number (as opposed to process specific sequence number)**

- **Sender attaches** *unique id 'i'* **to each message** *m* **and sends** *<m,i>* **to the** *sequencer(g)* **as well as to group** *g*

- **Sequencer maintains group specific sequence number S_g (consecutive and increasing) and** *B-multicasts order messages* **to** *g*

Total Ordering Using a Sequencer (Method 1)

1. Algorithm for group member *p*

On initialization: $r_g := 0;$

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

unique msg id

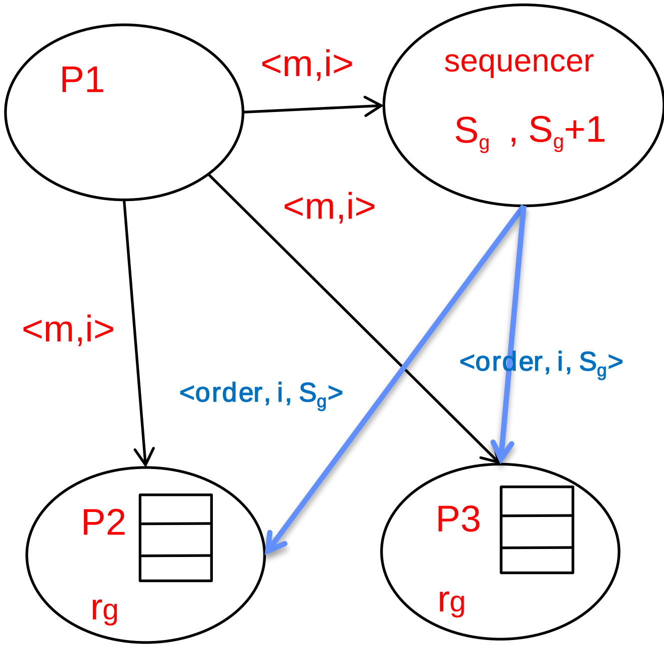
On B-deliver$(\langle m, i \rangle)$ *with* $g = group(m)$
    Place $\langle m, i \rangle$ in hold-back queue;

On B-deliver$(m_{order} = \langle \text{“order”}, i, S \rangle)$ *with* $g = group(m_{order})$
    wait until $\langle m, i \rangle$ 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$(\langle m, i \rangle)$ *with* $g = group(m)$
    $B\text{-}multicast(g, \langle \text{“order”}, i, s_g \rangle);$
    $s_g := s_g + 1;$



Group *g*: *P1, P2, P3*

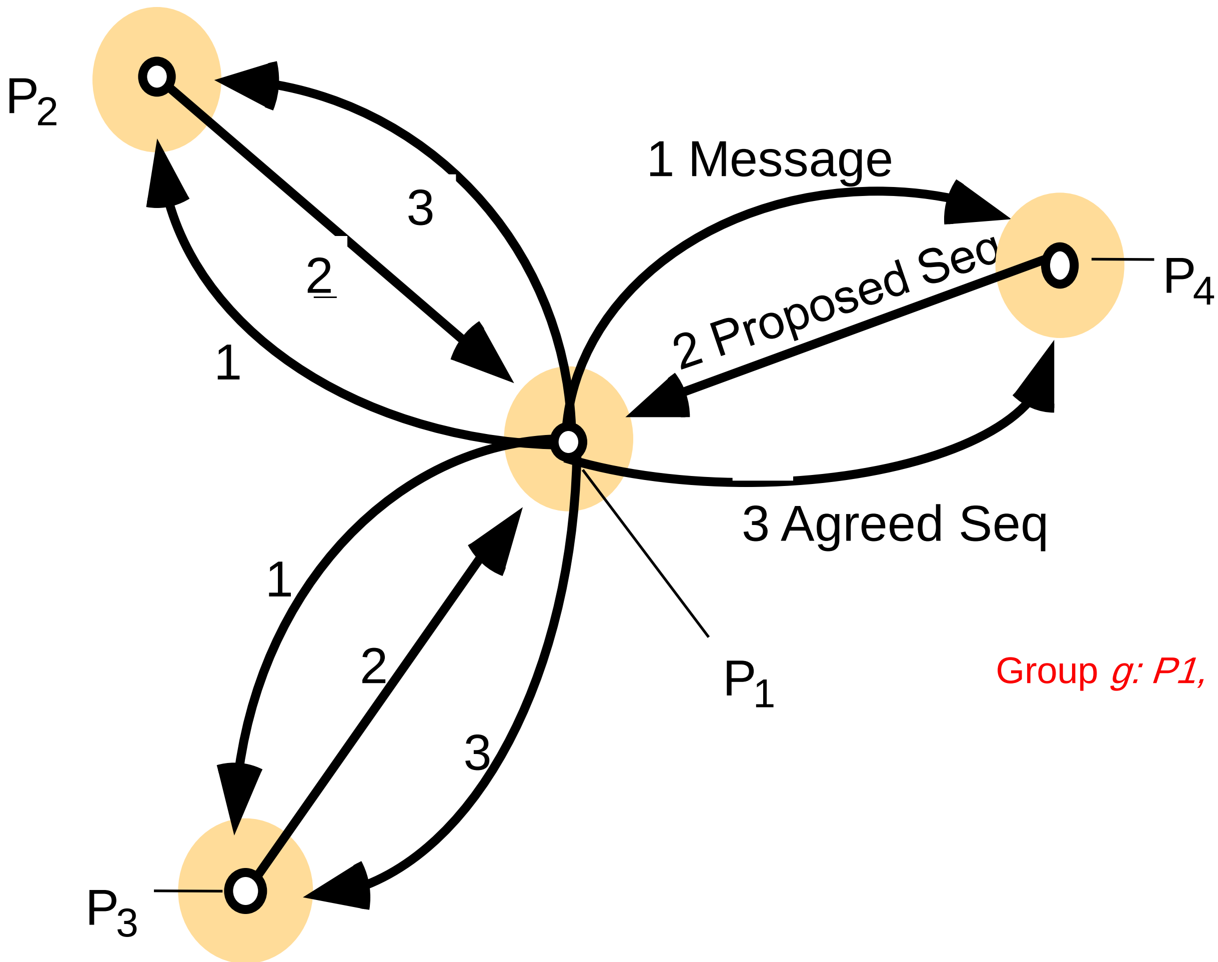
Sequencer (g) :

- Single point of failure
- Bottleneck

2^{nd} Method - ISIS Algorithm

- **Processes collectively agree on sequence numbers (priority) in three rounds**
- **Sender sends message *m* with its *id* to all receivers;**
- **Receivers suggest priority (sequence number) and reply to sender with proposed priority;**
- **Sender collects all proposed priorities; decides on final priority (breaking ties with process ids), and resends the agreed final priority for message m**
- **Receivers deliver message *m* according to decided final priority**

ISIS algorithm for total ordering (Method 2)



Group *g: P1, P2, P3, P4*

ISIS algorithm for total ordering

1. **sender** *p* ***B*-multicasts** *<m,i>* **with message** *m* **and unique id** *i* **to everyone.**
2. **On receiving** *m* **(***first time***)**
   1. *m* **is added to a** ***priority queue*** **and tagged as** ***undeliverable***
   2. **reply to sender with** ***proposed priority*****, i.e., a sequence number**
      - » *seq number = 1 + largest seq number heard so far, suffixed with the recipient's process ID*
   3. *priority queue is always sorted by priority*
3. **Sender**
   1. **collects all responses from the recipients,**
   2. **calculates their** ***maximum*****, and**
   3. **re-multicasts (B-multicast) original message with this as the** ***final priority*** **for** *m*
4. **On receiving** *m* **(***with final priority***)**
   1. **mark the message as** ***deliverable*****,**
   2. **reorder the priority queue, and**
   3. **deliver the set of lowest priority messages that are marked as** ***deliverable*****.**

Proof of Total Order (By Contradition)

- **For m_1, consider the first process** *p* **that delivers m_1**
  - **At** *p*, **let m_1 have the agreed sequence number (***finalpriority(m_1)***) and marked** *deliverable (at the front of the hold-back priority queue)*
  - **Let m_2 be another message that has not yet been delivered**
    - » **i.e., m_2 is on the same queue (it has not been assigned its sequence number) or has not been seen yet by** *p*
  - **Then**
    - » *finalpriority(m_2) ≥ proposedpriority(m_2)* **due to: "max" operation at sender &**
    - » *proposedpriority(m_2) ≥ final priority (m_1)* **due to: proposed priorities by** *p* **only increase (***m_1* **is ahead of the queue)**
- **Suppose there is some other process** *q* **that delivers** m_2 **before it delivers** m_1*. Then at q*
  - *Finalpriority(m1) ≥ proposedpriority(m1) ≥ finalpriority (m2)*
- *Contradiction !*

Summary

- Multicast is operation of sending one message to multiple processes
  - Basic multicast
    - » Uses reliable unicast
    - » Guarantees integrity, validity but not agreement
  - Reliable multicast
    - » Uses basic multicast
    - » Guarantees agreement (no ordering)
- Ordering – FIFO, total, causal
  - FIFO-multicast uses sequence number for each process and a queue
  - Causal-multicast uses vector time stamps
  - Total order- multicast uses a sequencer or agreement on sequence numbers