Coordination & Agreement in Group Communications

George Coulouris, Jean Dollimore and Tim Kindberg, "Distributed Systems Concepts and Design", Fifth Edition, Pearson Education, 2012

Outline

- Group Communications
- Applications
- Basic Multicast
- Reliable Multicast
- Ordered Multicast
  - FIFO Ordering
  - Casual Ordering
  - Total Ordering

Group Communication (1)

- **Objective:** each of a group of processes must receive copies of the messages sent to the group

- **Group communication requires:**
  - Coordination
  - Agreement: on the set of messages that is received and on the order of delivery

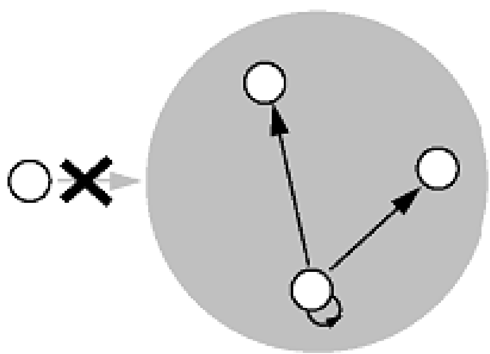
- We study multicast communication of processes whose membership is known (static groups)

Group Communication (2)

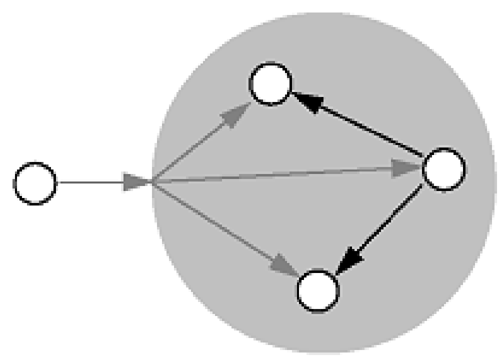
- **System:** contains a collection of processes, which can communicate **reliably** over **one-to-one** channels

- **Processes:** members of groups, may fail only by crashing

- **Groups:**



Closed group



Open group

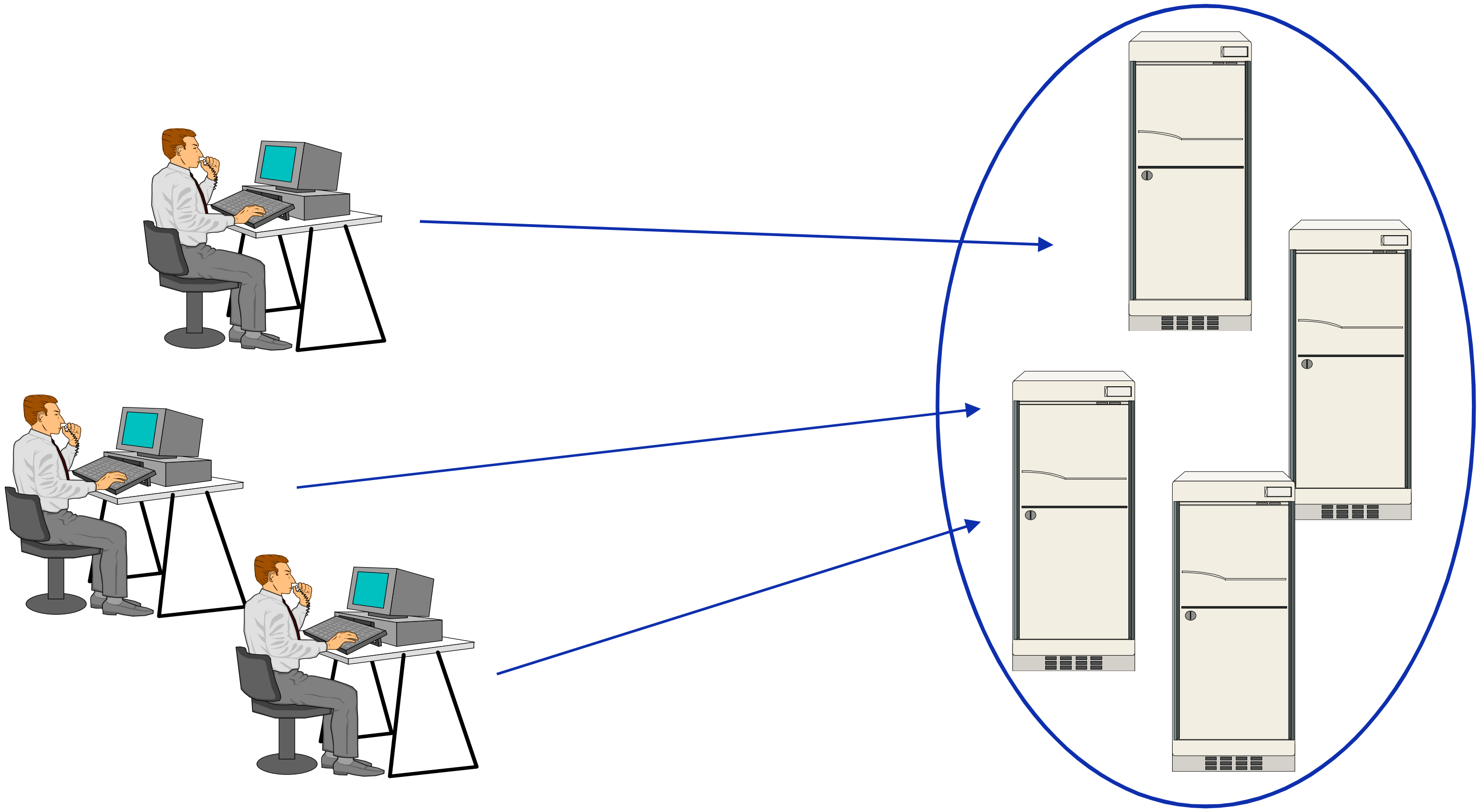
Group Communication (3)

- **Primitives:**
  - ***multicast(g, m)*****:** sends the message *m* to all members of group g
  - ***deliver(m)* :** delivers the message *m* to the calling process
  - ***sender(m)* :** unique identifier of the process that sent the message *m*
  - ***group(m)*****:** unique identifier of the group to which the message *m* was sent

Who Needs Group Communication?

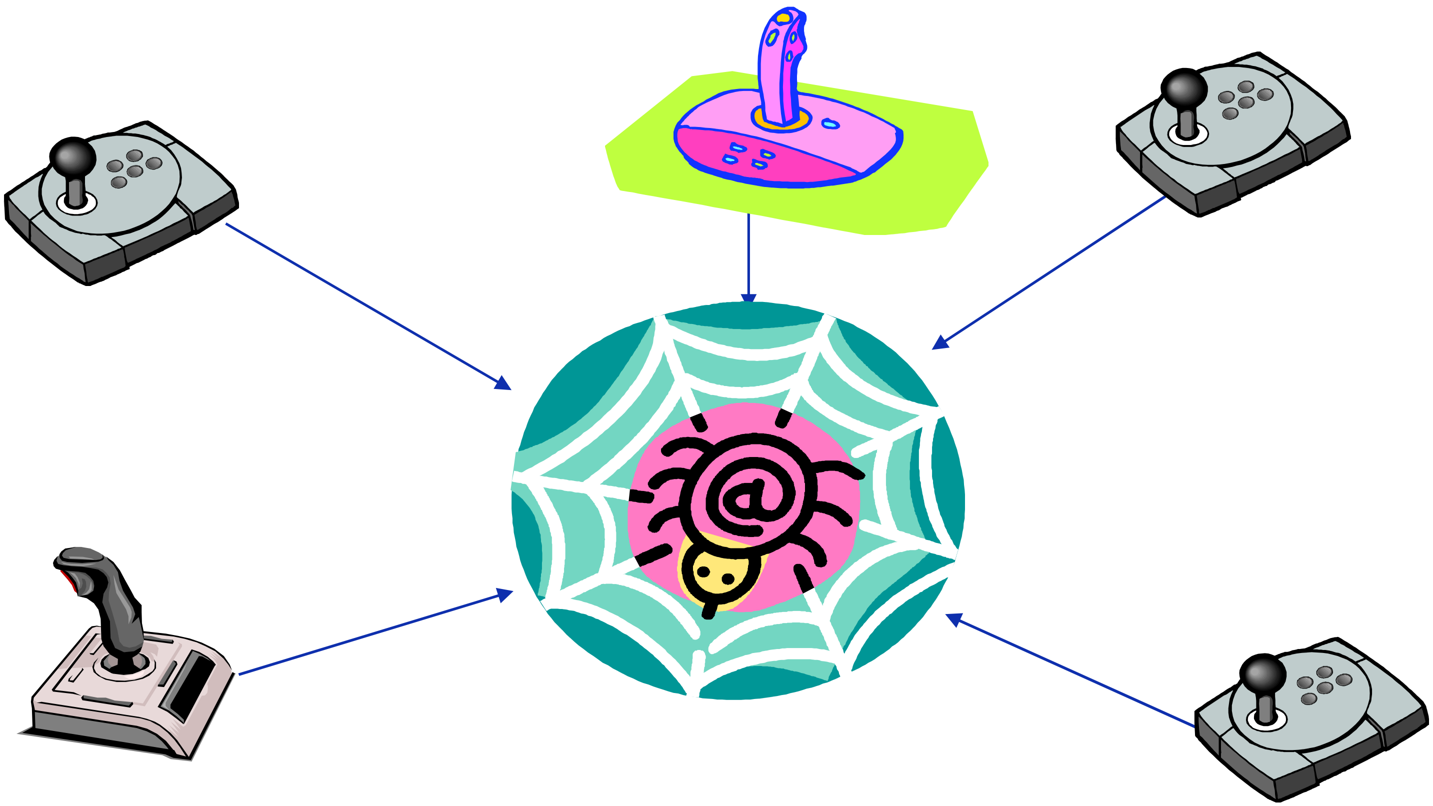
- Highly available servers (client-server)
- Database Replication
- Multimedia Conferencing
- Online Games
- Cluster management
- …

Distributed Web Server



- High availability

Online Game



- Fault-tolerance, Order

Different Comm. Methods

- Unicast
  - Point-to-Point Communication
  - Multiple copies are sent.
- Broadcast
  - One-to-All Communication
  - Abuse of Network Bandwidth
- **Multicast**
  - One-to-multiple Communication

Group Communication (4)

- Basic Multicast
- Reliable Multicast
- Ordered Multicast

Basic Multicast

- **Objective:** Guarantee that a correct process will eventually deliver the message as long as the multicaster does not crash

- **Primitives:** B_multicast, B_deliver

- **Implementation:** Use a reliable one-to-one communication

To B_multicast(g, m)

For each process p $\in$ g, send(p, m);

On receive(m) at p

Use of threads to perform the send operations simultaneously

B_deliver(m) to p

- **Unreliable:** Retransmission & Acknowledgments may be dropped

Reliable Multicast (1)

- **Properties to satisfy:**
  - **Integrity:** A correct process P delivers the message m at most once
  - **Validity:** If a correct process multicasts a message m, then it will eventually deliver m
  - **Agreement:** If a correct process delivers the message m, then all other correct processes in group(m) will eventually deliver m
- **Primitives:** R_multicast, R_deliver

Reliable Multicast (2)

- Implementation using B-multicast:

Initialization

msgReceived := {};

R-multicast(g, m) by p

B-multicast(g, m); **// p $\in$ g**

B-deliver(m) by q with g = group(m)

If (m $\notin$ msgReceived)

Then **msgReceived := msgReceived $\cup$ {m};**
If (q $\neq$ p) Then B-multicast(g, m);
R-deliver(m);

Correct algorithm, but inefficient
(each message is sent |g| times to each process)

Reliable Multicast

- **Reliable Multicast over IP Multicast**
- R-multicast use a combination of IP multicast, **piggybacked acknowledgements** (that is, acknowledgements attached to other messages) and **negative acknowledgements**
- Piggyback acknowledgements on the **messages** that they **send** to the **group**.
- Processes send a **separate response message** only when they detect that they have **missed** a **message**.
- A response indicating the **absence** of an **expected message** is known as a **negative acknowledgement**

Reliable Multicast

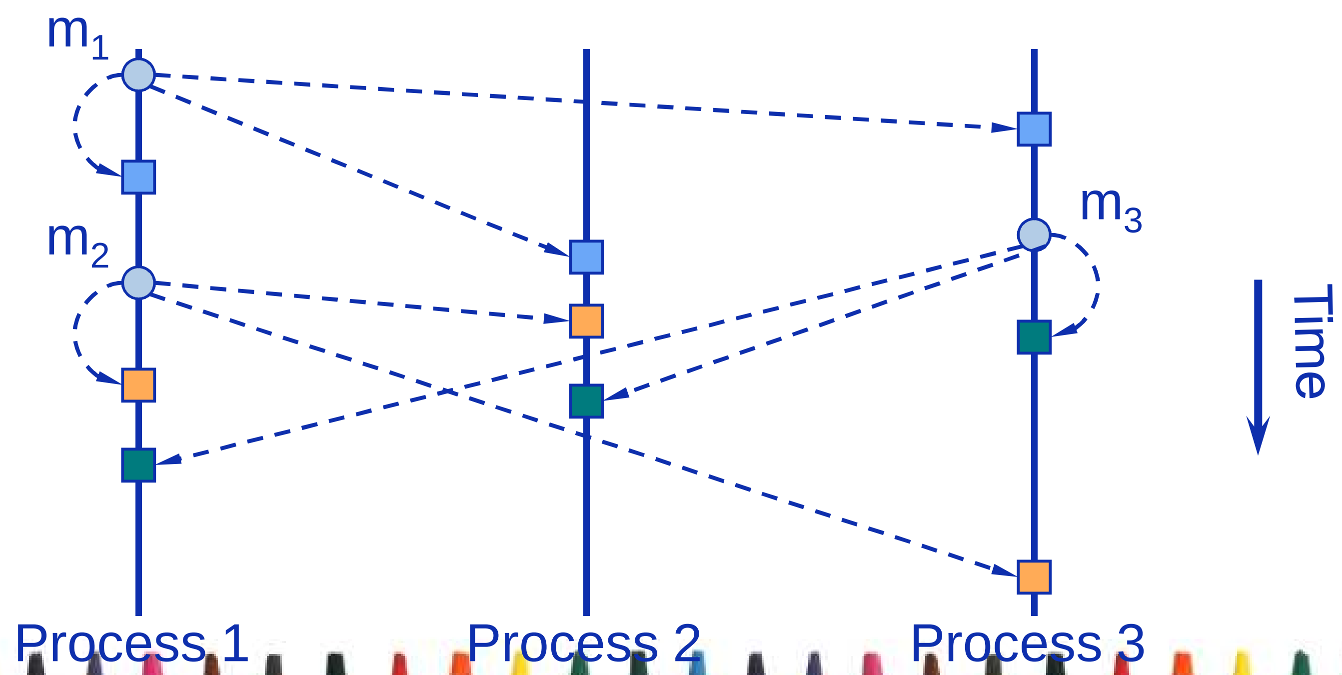
- **Uniform Agreement**
- If a process, whether it is correct or fails, delivers message m, then all correct processes in group(m) will eventually deliver m.

Ordered Multicast

- **Ordering categories:**
  - FIFO Ordering
  - Total Ordering
  - Causal Ordering
  - Hybrid Ordering: Total-Causal,
    Total-FIFO

FIFO Ordering (1)

- If a correct process issues multicast(g, m_1) and then multicast(g, m_2), then every correct process that delivers m_2 will deliver m_1 before m_2

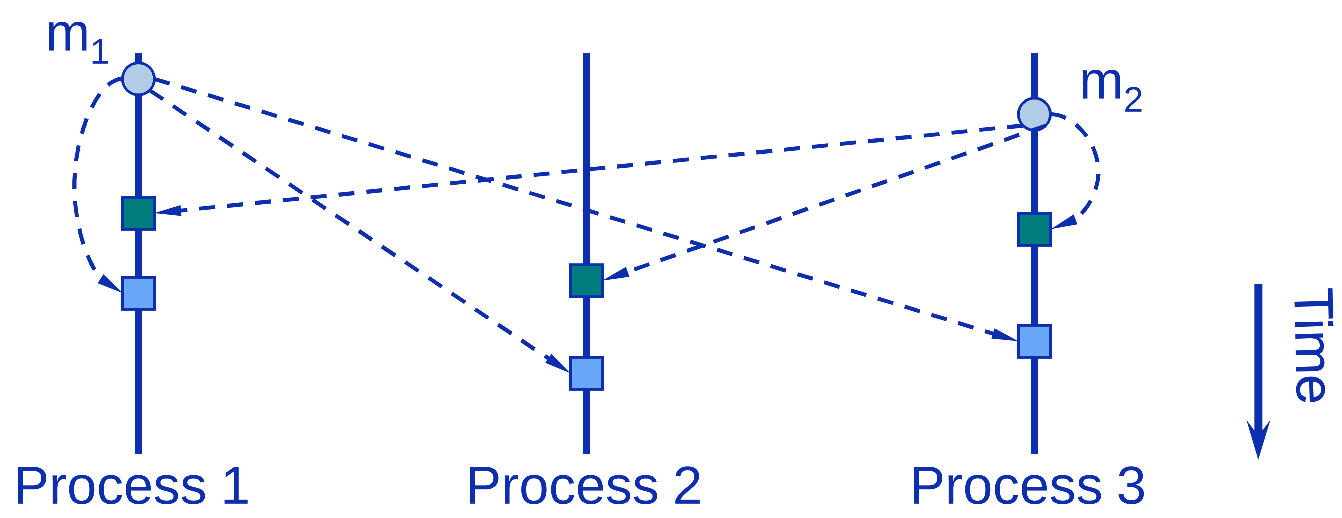


FIFO Ordering (2)

- **Primitives:** FO_multicast, FO_deliver
- **Implementation:** Use of sequence numbers
- Variables maintained by each process p:
  - $\mathbf{S_g^p}$**:** Number of messages sent by p to group g
  - $\mathbf{R_g^q}$**:** sequence number of the latest message p has delivered from process q that was sent to the group
- Algorithm
- FIFO Ordering is reached only under the assumption that groups are non-overlapping

Total Ordering (1)

- If a correct process delivers message m_2 before it delivers m_1, then any correct process that delivers m_1 will deliver m_2 before m_1



- **Primitives:** TO_multicast, TO_deliver

Total Ordering (2)

- **Implementation:** Assign totally ordered identifiers to multicast messages
- Each process makes the same ordering decision based upon these identifiers
- **Methods for assigning identifiers to messages:**
  - Sequencer process
  - Processes collectively agree on the assignment of sequence numbers to messages in a distributed fashion

Total Ordering (3)

- **Sequencer process:** Maintains a group-specific sequence number S_g

Initialization

$S_g := 0;$

B-deliver(<m, Ident.>) with g = group(m)

B-multicast(g, <“order”, Ident., S_g>);

$S_g = S_g + 1;$

- Algorithm for group member $p \in g$

Initialization

$R_g := 0;$

Total Ordering (4)

Unique identifier of m

TO-multicast(g, m) by p

B-multicast(g ∪ Sequencer(g), <m, Ident.>);

B-deliver(<m, Ident.>) by p, with g = group(m)

Place <m, Ident.> in hold-back queue;

B-deliver(m_{order}= <“order”, Ident., S>) by p, with g = group(m_{order})

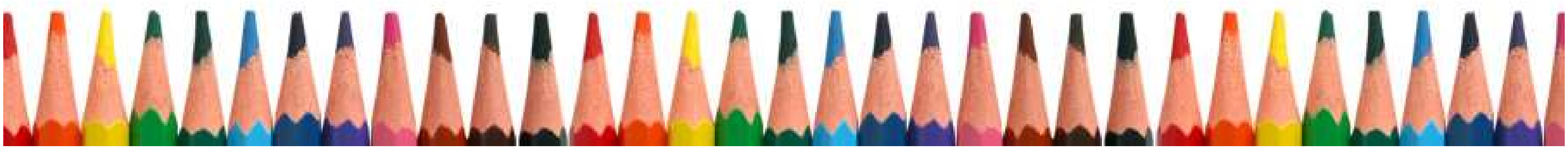
Wait until (<m, Ident.> in hold-back queue AND S = R_g);

TO-deliver(m);

R_g = S + 1;

Total Ordering (4)

- The sequence numbers S_g^p attached to each multicast message, allows the recipients to learn about messages which they have missed
- A process q can Rdeliver (m) only if the sequence number
- $S_g^p = R_g^q + 1$
- Immediately following Rdeliver (m) the value is R_g^q incremented
- If an arriving message has a number **$S \le R_g^q$** then process q knows that it has **already performed** Rdeliver on that message and can **safely discard it.**
- If **$S > R_g^q$** then the receiving process q knows that it has **missed** some message from p destined for the group g.
- In this case the **receiving process q** puts the message in a **hold-back queue** and sends a negative acknowledgement to **the sending process p requesting** the **missing message(s)**

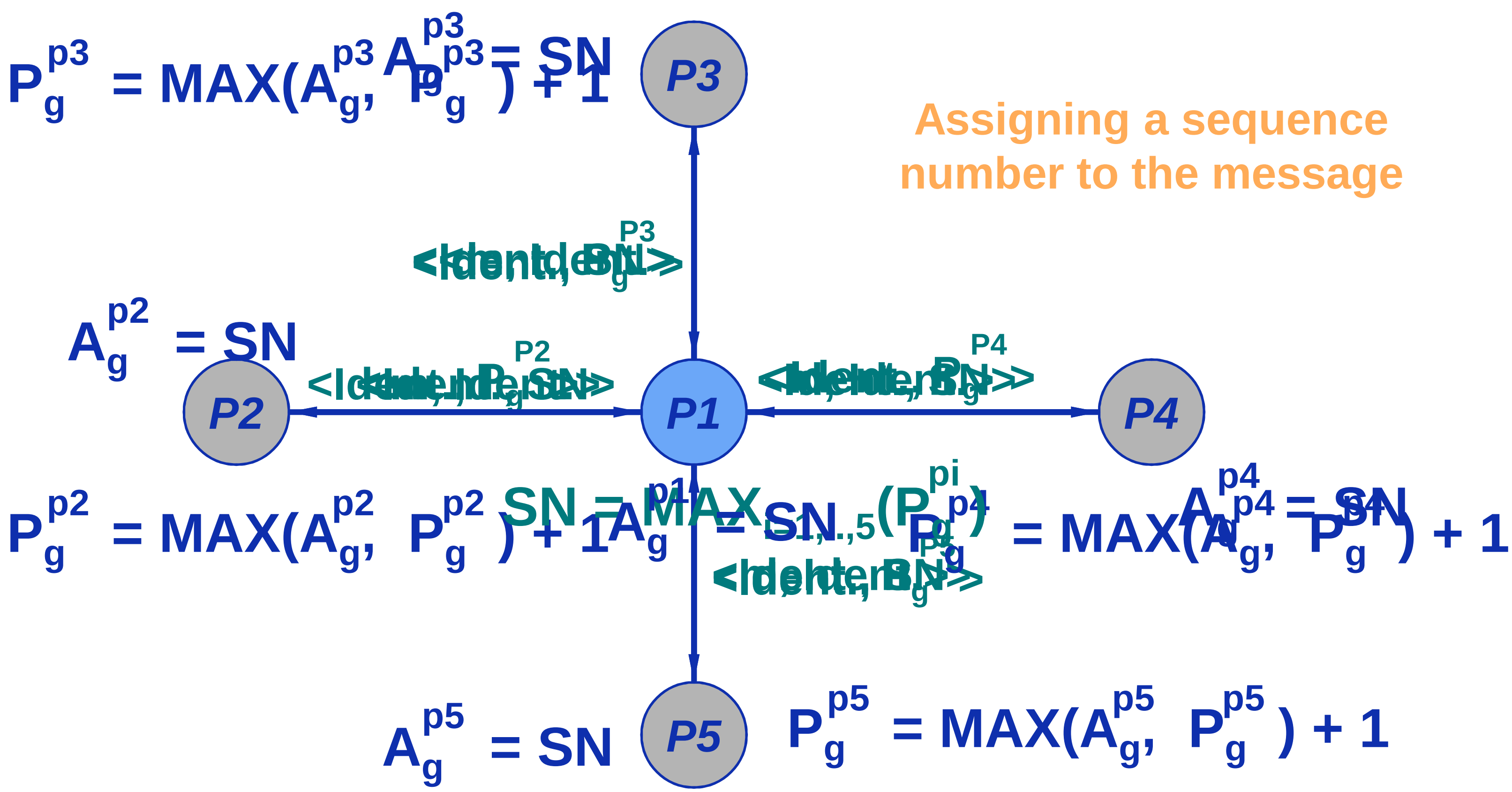


Total Ordering (5)

- Processes collectively agree on the assignment of sequence numbers to messages in a distributed fashion

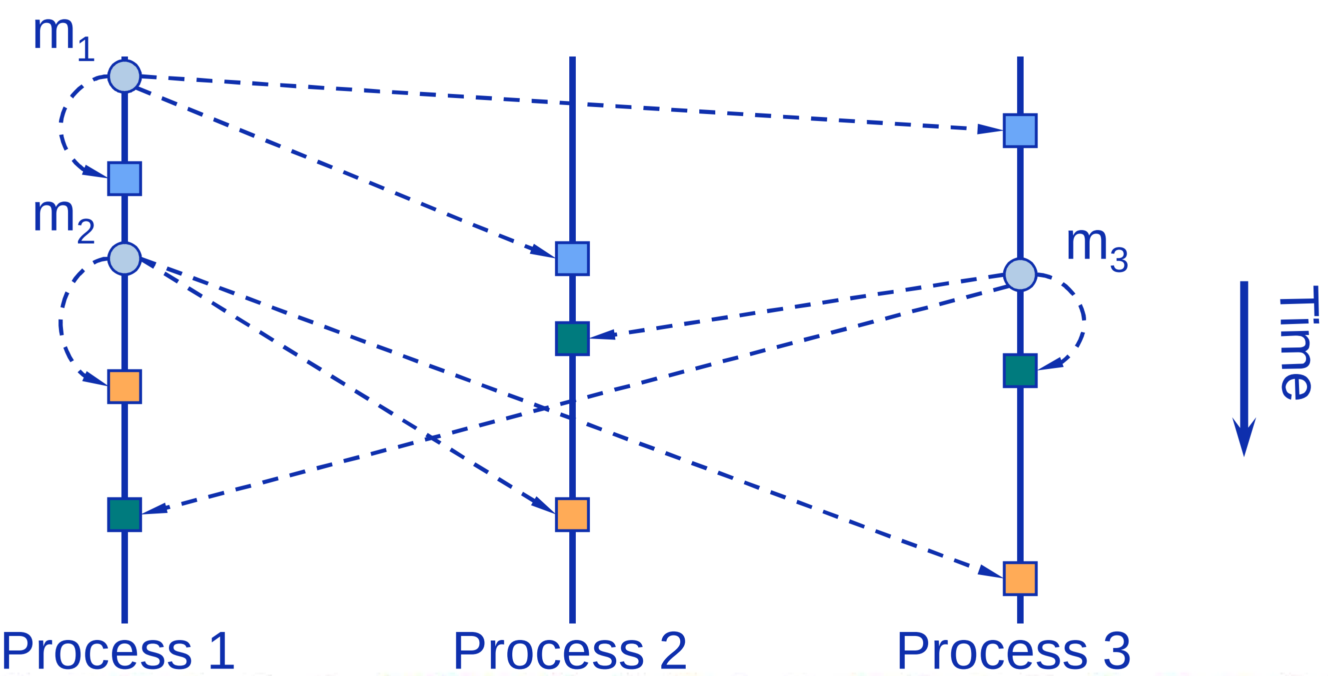
- Variables maintained by each process p:
  - $\mathbf{P_g^q}$**:** largest sequence number proposed by q to group g
  - $\mathbf{A_g^q}$**:** largest agreed sequence number q has observed so far for group g

Total Ordering (6)



Causal Ordering (1)

- If multicast(g, m_1) $\rightarrow$ multicast(g, m_3), then any correct process that delivers m_3 will deliver m_1 before m_3



Causal Ordering (2)

- **Primitives:** CO_multicast, CO_deliver
- Each process p_i of group g maintains a timestamp vector

  V_i^g

$V_i^g[j]$ = Number of multicast messages received from p_j that happened-before the next message to be sent

- **Algorithm for group member p_i:**

  Initialization

  - **Example**

    $V_i^g[j]$:= 0 (j = 1, …, N);

Causal Ordering (3)

CO-multicast(g, m)

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

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

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

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

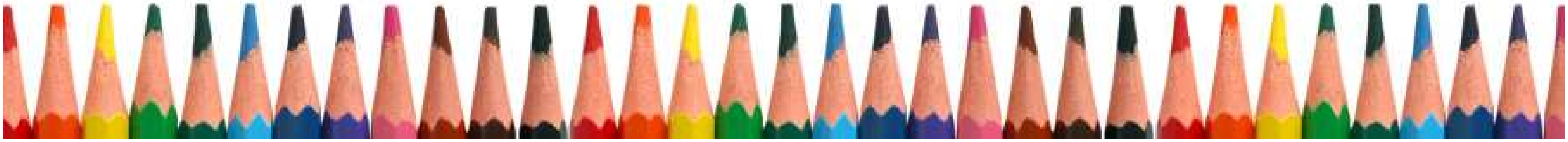
Wait until $(V_j^g[j] = V_i^g[j] + 1)$ AND $(V_j^g[k] \le V_i^g[k])$;
$(k \neq j)$

CO-deliver(m);

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

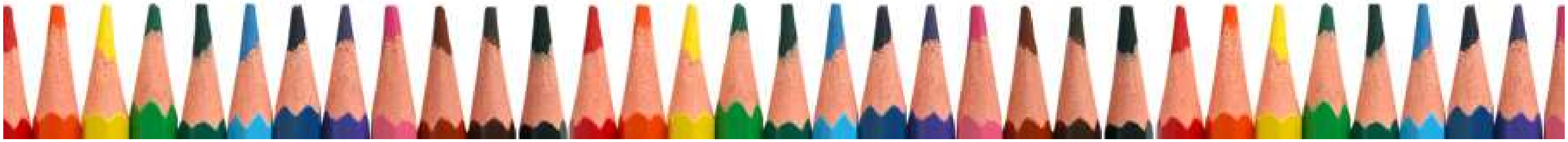
Causal Ordering (1)

- To implement Causal ordering on top of Basic Multicast (bmulticast)
- Each process maintains a vector clock
- To send a Causal Ordered multicast a process first uses a bmulticast
- When a process pi performs a bdeliver(m) that was multicast by a process pj it places it in the holding queue until:
  - It has delivered any earlier message sent by pj
- **and**
  - It has delivered any message that had been delivered at pj before pj multicast m
- Both of these conditions can be determined by examining the vector timestamps



Overlapping Groups

- A process in more than one group.
- **Global FIFO Ordering** If a correct process issues multicast(g,m) and then multicast(g’,m’) then every correct process in $g \cap g'$ that delivers m’ delivers m before m’
- **Global Causal Ordering** If multicast(g,m) $\rightarrow$ multicast(g’, m’) then every correct process in $g \cap g'$ that delivers m’ delivers m before m’
- **Pairwise Total Ordering** If a correct process delivers message m sent to g before it delivers m’ sent to g’ then every correct process in $g \cap g'$ which delivers m’ delivers m before m’



Summary

- Group Communications
- Applications
- Basic Multicast
- Reliable Multicast
- Ordered Multicast
  - FIFO Ordering
  - Casual Ordering
  - Total Ordering