The UNIVERSITY *of* EDINBURGH
SCHOOL *of* INFORMATICS

CS4/MSc

Distributed Systems

Björn Franke

bfranke@inf.ed.ac.uk

Room 2414

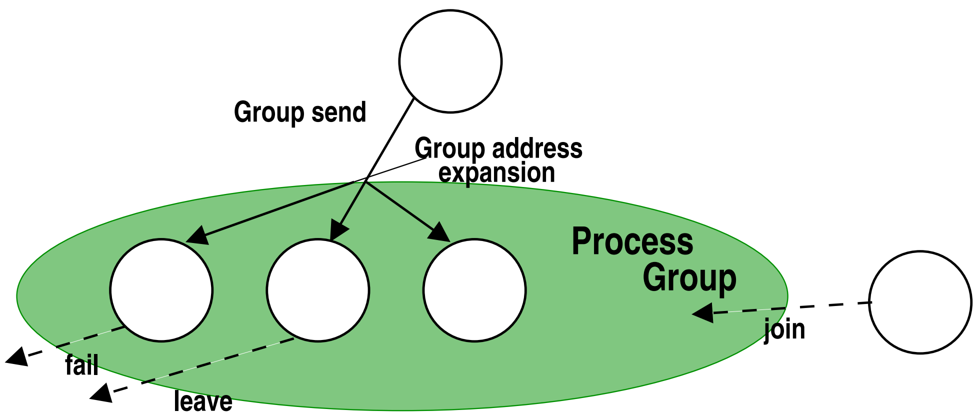
(Lecture 13: Multicast and Group Communication,
21st November 2005)

Group Communication

- **Multicast** is an operation that sends a single message from one process to each of the members of a group of processes.
- In general this is done in such a way that the membership of the group is transparent to the sender.
- A multicast is termed **reliable** if any transmitted message is either received by all members of the group or by none of them.
- A multicast is termed **totally ordered** if all messages transmitted to the group reach all members of the group in the same order.
- Totally ordered reliable multicast is used in active replication systems to send messages from the front end to the replica managers.
- In other applications other weaker forms of ordering are sufficient.
- In order to achieve a required ordering, a message may not be **delivered** (to the application layer) as soon as it is **received** by a process.

Multicast Groups

- Each group has a **group identifier** which is used when messages are addressed to the group.
- Groups can be **static** or **dynamic**.
- An implementation of group communication usually incorporates a **group membership service.**



Group Membership Service

A group membership service has the following roles:

- Providing an interface for group membership changes.
- Implementing a failure detector.
- Notifying members of group membership changes.
- Performing group address expansion.

Using the failure detector the membership service keeps track of implicit changes to the group due to process failures or communication infrastructure failures. If a process is *suspected* it is no longer considered a member of the group.

Since multicast messages are sent to the group (using the group identifier) rather than a list of processes the membership service can expand the identifier in such as way as to reflect the current membership.

Group Views

- Some applications, such as the fault-tolerant systems considered in the previous lecture, require sophisticated failure detection and notification of group membership.
- This may be achieved when the group membership service maintains **group views**, listing current group members, identified by their unique process identifiers.
- The list is ordered; for example, according to the order in which processes joined the group.
- A new group view is generated whenever membership changes.
- Note that this means that a process which wrongly becomes *suspected* may find itself **excluded** and will need to **rejoin** the group explicitly (with a new ID) in order to receive subsequent messages.
- When failures in the communication infrastructure result in a partition of the network the group service management may allow only one subset to continue or may partition the group view into two or more subgroups.

Reliable Multicast

A reliable multicast is one which satisfies the following properties:

Integrity: A correct process P delivers a message m at most once. Furthermore, $P \in group(m)$ and m was supplied to a *multicast* operation by $sender(m)$.

Validity: If a correct process P multicasts a message m then P will eventually deliver m.

Agreement: If a correct process delivers a message m, then all other processes in $group(m)$ will eventually deliver m.

The naive implementation of multicast:

```
B-multicast(g,m): for each process P in group g, send(p,m);
On receive(m) at P: B-deliver(m) at P.
```

is not reliable even if `send` is reliable (consider what happens if the sender fails after sending to a subset of the group) but nevertheless it can be used to implement a reliable multicast.

Reliable Multicast Algorithm

```
On initialization
   Received := {};

For process P to R-multicast message m to group g
   B-multicast(g, m); // P is included as a destination

On B-deliver(m) at process Q with g = group(m)
   if (m not in Received)
   then
      Received := Received + m;
      if (Q not equal to P)
      then B-multicast(g, m); end if
      R-deliver m;
   end if
```

Whilst correct this is inefficient since each message is sent $|g|$ times to each process.

Reliable Multicast based on IP Multicast

The previous algorithm is very pessimistic and a better algorithm, for closed groups, can be developed using IP multicast (which is itself unreliable; see handout 3), piggy-backed acknowledgements and negative acknowledgements.

Acknowledgements are not sent individually to senders but are piggy-backed on to the next message sent to the group. An individual **negative** acknowledgement is sent when a process detects that it has missed a message, by observing the piggybacked acknowledgements.

Each process keeps sequence numbers, recording the messages it has sent to the group and those from other group members that it has delivered. Lost messages are detected when processes observe each other's sequence numbers.

Reliable Multicast based on IP Multicast (2)

- Each process P maintains a sequence number S_g^P for each group g it belongs to.
- Each process also records R_g^P, the sequence number of the latest message it has delivered from process P sent to g.
- When P sends a message to g it piggy-backs the value S_g^P and acknowledgements of the form $\langle Q, R_g^Q \rangle$. P then increments S_g^P by one. Here R_g^Q is the sequence number of the latest multicast message from Q which P has delivered since P last multicast.
- A process delivers a message from P with sequence number S iff $S = R_g^P + 1$; it increments R_g^P by one immediately after delivery.
- If $S \leq R_g^P$ the message has already been delivered and is discarded.
- Later messages are held in a **hold-back queue**.
- If $S > R_g^P + 1$ or $R > R_g^Q$ for an attached acknowledgement $\langle Q, R \rangle$ a message has been lost and is requested using a negative acknowledgement.

Ordered Multicast

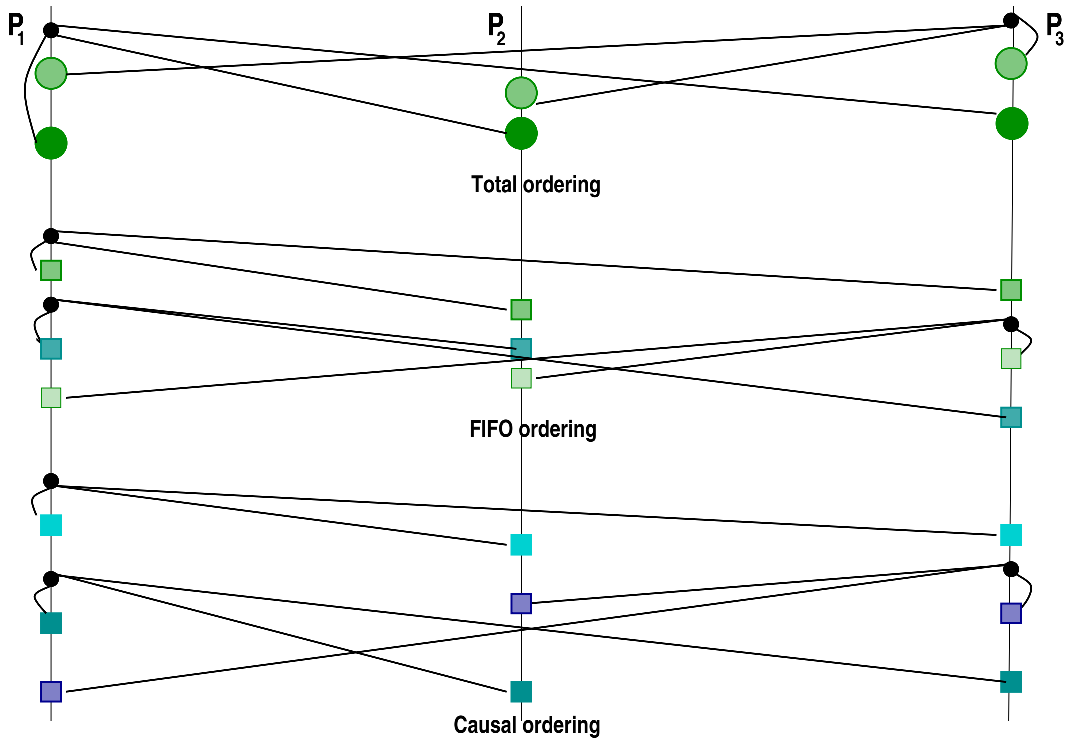
Total ordering: if a correct process delivers message m before it delivers m' then any other correct process that delivers m' will deliver m before it delivers m'.

FIFO ordering: if a correct process issues $multicast(g, m)$ and then $multicast(g, m')$ then every correct process that delivers m' will deliver m before m'.

Causal ordering: if $multicast(g, m) \rightarrow multicast(g, m')$ (where $\rightarrow$ is the happened-before relation induced only by messages sent between the members of g) then any correct process that delivers m' will deliver m before m'.

Note that causal ordering implies FIFO ordering, but both are partial ordering: nothing is stipulated about the relative ordering of messages from different senders. Conversely total ordering does not imply anything about the order in which messages are sent. Consequently hybrid orderings (FIFO-total and causal-total) can be defined.

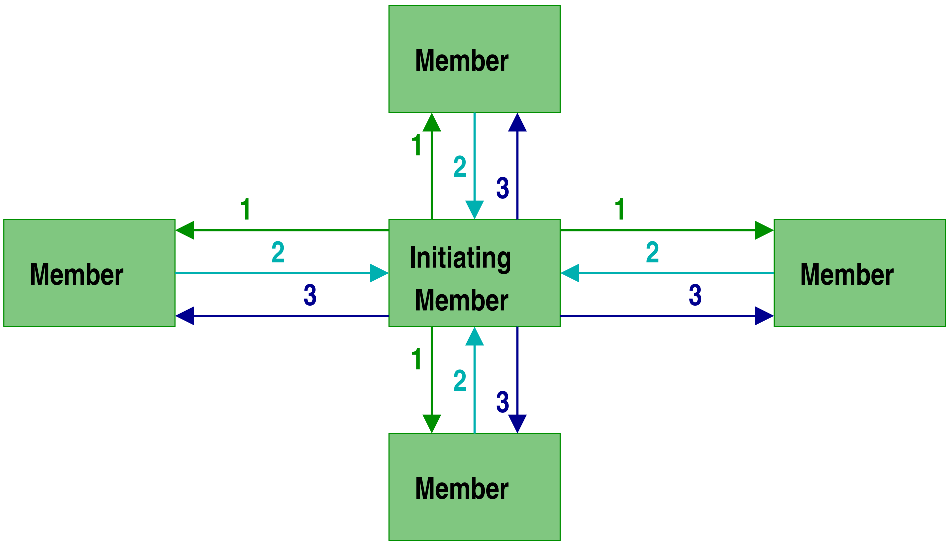
Orderings: Total, FIFO and Causal



The Isis Algorithm for Total Ordering

- Totally ordered identifiers are associated with all messages and each process makes ordering decisions based on these identifiers.
- Each process Q in a group g keeps A_g^Q, the largest agreed **sequence number** it has seen in g and P_g^Q its own largest proposed sequence number.
- When a process P wishes to multicast a message m to group g it *B-multicasts* $\langle m, i \rangle$ to g, where i is a unique identifier for m.
- Each process Q replies to P with a proposal for the message's agreed sequence number $P_g^Q = \max(A_g^Q, P_g^Q) + 1$. Q provisionally assigns the proposed sequence number to the message and places it in its hold-back queue.
- P collects all the proposed sequence numbers and selects the largest a; it then *B-multicasts* $\langle i, a \rangle$ to g.
- Each process Q in g sets $A_g^Q = \max(A_g^Q, a)$ and attaches a to the message (identified by i), reordering the hold-back queue if necessary.

Isis algorithm (2)



The initiating member sends a proposed number (message 1) to the other members. Each member sends its own proposed number at 2. The initiator makes a selection and informs all members in message 3.

View Delivery

As the membership of a group changes the group membership service delivers a **view** of the current membership to each process in the group. Although group membership changes may occur concurrently an order is imposed on the sequence of views delivered to each process.

As with multicast message delivery, **view delivery** is distinct from receiving a notification of membership change. Group membership protocols keep proposed views on a hold-back queue until all current members agree that they should be delivered.

A view delivery system should satisfy the following properties:

Order: If process P delivers view $v(g)$ and then view $v'(g)$, then no other process $Q \neq P$ delivers $v'(g)$ before $v(g)$.

Integrity: If process P delivers view $v(g)$ then $P \in v(g)$.

Non-triviality: If process Q joins g and becomes indefinitely reachable from process $P \neq Q$ then eventually Q is always in the views that P delivers.

View-synchronous Group Communication (1)

View-synchronous communication extends reliable multicast to take account of changing group views. It provides the following guarantees:

Agreement: Correct processes deliver the same set of messages in any given view.

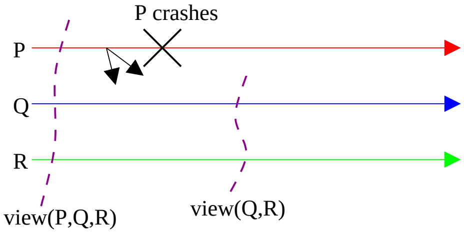
Integrity: If a process P delivers message m, then it will not deliver m again. Also, $P \in group(m)$ and m was supplied to a *multicast* operation by $sender(m)$.

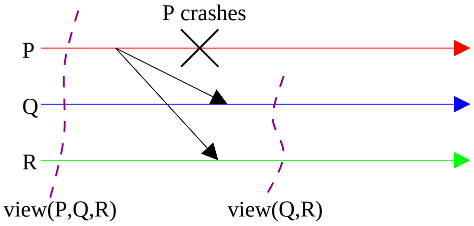
Validity (closed groups): Correct processes always deliver the messages that they send. If the system fails to deliver a message to any process Q, then it notifies the surviving processes by delivering a new view with Q excluded, immediately after the view in which any of them delivered the message.

The delivery of a new view conceptually cuts the history of each process and every message that is delivered at all is either delivered before the cut for all processes, or after it.

View-synchronous Group Communication (2)

Acceptable





Unacceptable