Message delivery is an important issue of group communication; two aspects are relevant:

a) who gets the message, and
b) when is the message delivered.

Atomicity

Sequence of message delivery
Ordering for message delivery

It is desired to deliver all messages sent to the group $G$ to all group members of $G$ in the same sequence, because otherwise we might get non-deterministic system behavior.

Example for group reconfiguration

$m_4$ is sent by $C_1$ before the group composition is modified. However, in order to guarantee atomicity, $m_4$ should not be delivered to $S_1$ and $S_2$ (since, due to the crash, it is no longer possible to deliver $m_4$ to $S_3$).
Delivery of messages without delay in the same sequence is not possible in a distributed system ⇒ ordering methods for message delivery.

- Synchronously, i.e., there is a system-wide global time ordering.
- Loosely synchronous, i.e., consistent time ordering, but no system-wide global (absolute) time.

### Total ordering by sequencer

### Virtually synchronous ordering

**sync-ordering**

---

A selected group member serializes all the messages sent to the group.

1. **sender** sends a message.
2. **sequencer** serializes the message and determines a sequence number.
3. **receiver** receives the message.
4. Application processes receive and process the message.

---

**Ordering for message delivery**

Delivery of messages without delay in the same sequence is not possible in a distributed system ⇒ ordering methods for message delivery.

- Synchronously, i.e., there is a system-wide global time ordering.
- Loosely synchronous, i.e., consistent time ordering, but no system-wide global (absolute) time.

### Total ordering by sequencer

### Virtually synchronous ordering

**sync-ordering**

---

**Example**

1. At time $T_1$, sender $N_1$ sends $N_2$ with $N_2$ dependent on $N_1$.
2. At time $T_2$, sender $N_3$ sends $N_2$ with $N_2$ concurrent.
3. At time $T_3$, $N_3$ is received before $N_1$.
4. At time $T_4$, $N_3$ is received after $N_1$.

---

**Diagram**

- Node $N_1$ at time $T_1$.
- Node $N_2$ at time $T_2$.
- Node $N_3$ at time $T_3$.
- Node $N_4$ at time $T_4$. 

---

**Diagram**

- Sender at time $T_1$ sends $N_2$.
- Sequencer serializes $N_2$.
- Receiver receives $N_2$ at time $T_3$.
- Application processes receive and process $N_2$.

---

**Diagram**

- Node $N_1$ at time $T_1$.
- Node $N_2$ at time $T_2$.
- Node $N_3$ at time $T_3$.
- Node $N_4$ at time $T_4$.
Delivery of messages without delay in the same sequence is not possible in a distributed system if ordering methods for message delivery:

- synchronously, i.e., there is a system-wide global time ordering.
- loosely synchronous, i.e., consistent time ordering, but no system-wide global (absolute) time.

**Total ordering by sequencer**

**Virtually synchronous ordering**

**sync-ordering**

Depending on the message delivery guarantee, five classes of multicast services can be distinguished.

1. **unreliable multicast**: an attempt is made to transmit a message to all members without acknowledgement; at-most-once semantics with respect to available members; message ordering is not guaranteed.

2. **reliable multicast**: the system transmits the messages according to “best-effort”, i.e., the “at-least-once” semantics is applied.

   - B-multicast primitive: guarantees that a correct process will eventually deliver the message as long as the multicaster does not crash.

   - B-deliver primitive: corresponding primitive when a message is received.

3. **serialized multicast**: consistent sequence for message delivery; distinction between

   - totally ordered

   - causally ordered (i.e., virtually synchronous)

4. **atomic multicast**: a reliable multicast which guarantees that either all operational group members receive a message, or none of them do.

5. **atomic, serialized multicast**: atomic message delivery with consistent delivery sequence

---

**Taxonomy of multicast**

**Multicast messages** for constructing distributed systems based on group communication;

- different multicast communication semantics

**Multicast classes**

**Relationship between multicast classes**

Multicasting can be realized by using IP multicast which is built on top of the Internet protocol IP.

Java API provides a datagram interface to IP multicast through the class **MulticastSocket**.