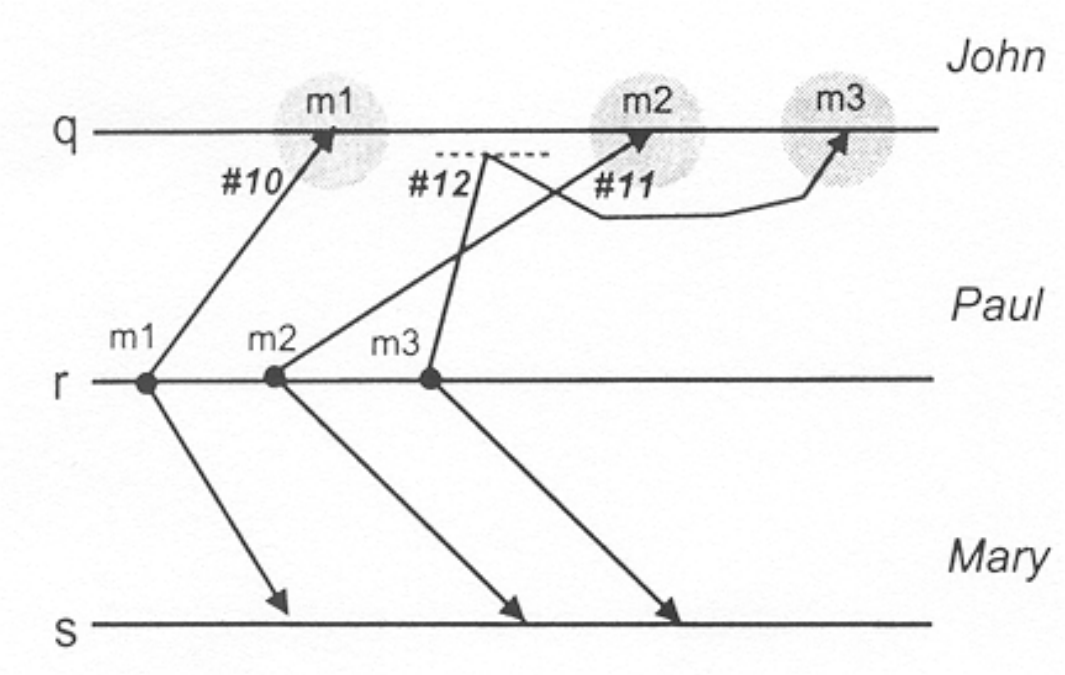
Distributed System Paradigms (15)

FIFO order

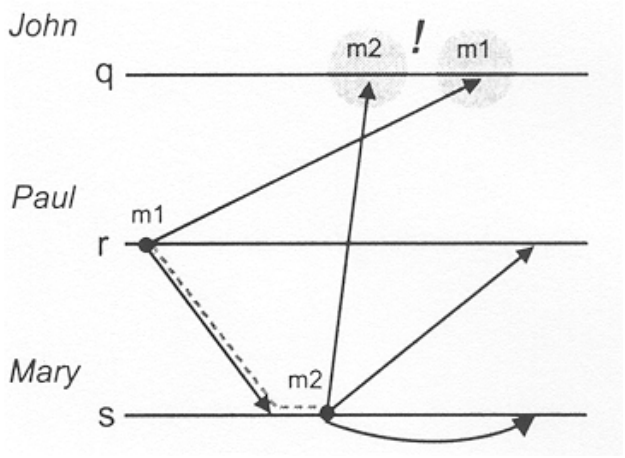
Any two messages sent by the ***same***participant, and delivered to any participant, are delivered in the order sent --> FIFO reflects the *potential causal order* generated by a single process.

Example Ia:



Distributed System Paradigms (16)

Example Ib:

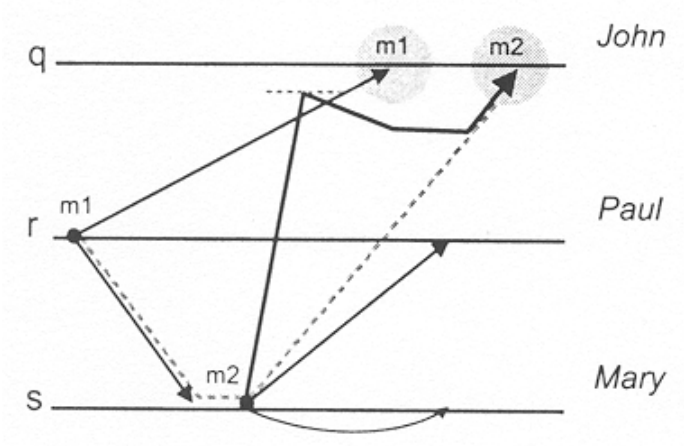


Causal Order

For any two messages m_1 , m_2 sent by *p*, resp. *q*, to the ***same*** destination participant *r:*

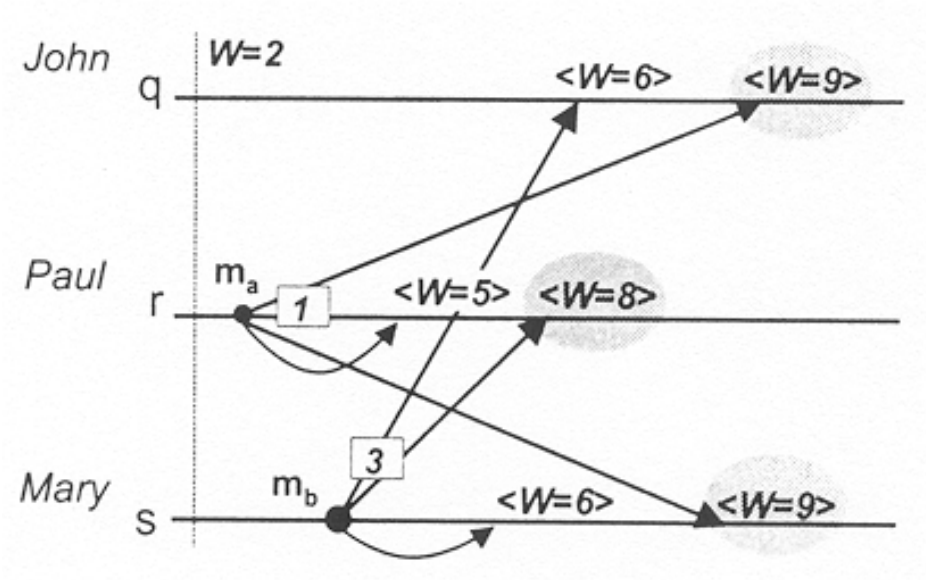
If $send_p(m_1) \longrightarrow send_q(m_2)$ then $deliver_r(m_1) \longrightarrow deliver_r(m_2)$, i.e. m_1 is delivered to *r* before m_2

Example IIa:



Distributed System Paradigms (17)

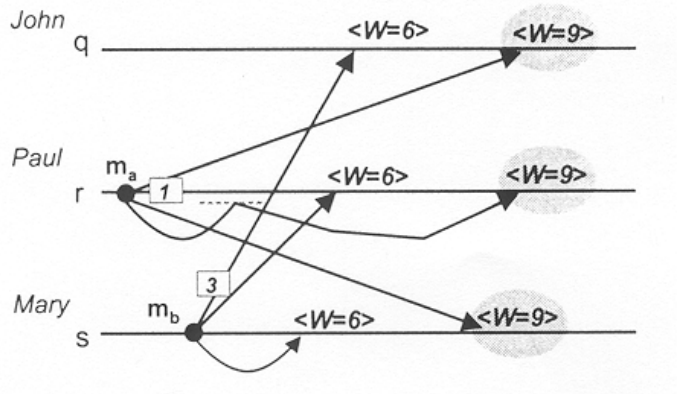
Example IIb:



Total Order

Any two messages delivered to any pair of participants are delivered in the same order to both participants

Example III:



Distributed System Paradigms (18)

Ordering Algorithms

causal order algorithm I:

(*past*$_p$*:=*list of all messages sent and received before by a sender process *p*)

- When a message is sent, it carries the *past* of its sender in a control field.
- After sending the message, it is added to the sender´s *past*.
- When a message is received, its *past* is checked. Messages in *past* not yet delivered, are delivered to the application and added to the *past* of the recipient. Then, the received message itself is delivered and, also, added to the *past* of the recipient.

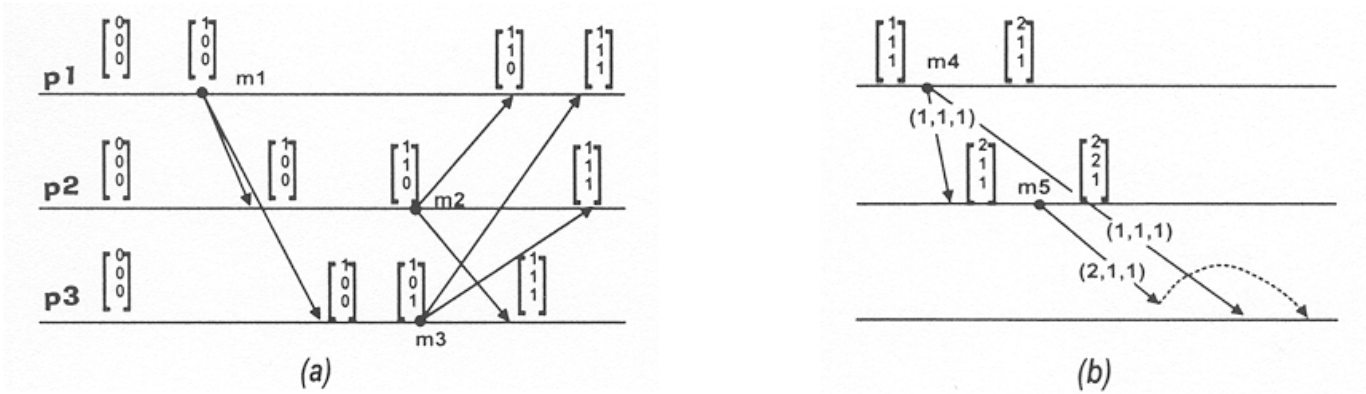
causal order algorithm II:

(*past*$_p$*:=*list of all message identifiers sent and received before by a process *p*)

- When a message is sent, it carries the *past* of its sender in a control field.
- After sending the message, its identifier is added to the sender´s past.
- When a message is received, its past is checked. If messages in past are not yet delivered, the message is put on hold, until these messages will arrive and will be delivered to the application and, subsequently, their identifiers have been added to the past of the recipient.
- Then, the received message itself is delivered and, also, its identifier is added to the past of the recipient.

Distributed System Paradigms (19)

Example for multicast messages (past is reduced from a *matrix clock* to a *vector clock*):



causal order algorithm III (Lamport clock)*:*

(*past:=*each process keeps a single integer called *logical clock (lclock))*

- When a message is sent, it carries the *lclock* of its sender in a control field.The *lclock* is incremented.
- Messages are exchanged using FIFO channels (Two messages from the same sender to the same destination are received in the order they were sent.).
- When a message is received, it is placed in a waiting queue, ordered according to its *lclock* (those with identical *lclock* are ordered according to their sender´s identifier). The message is kept in waiting state until a message with equal or greater *lclock* is received from every sender in the system.(Because of FIFO channels, then, all messages with smaller *lclock* (timestamp) have also been received). The message becomes *deliverable*.
- A *deliverable* message *m* at the head of a waiting queue is delivered. When *m* is delivered, the $lclock_p$ $(past_p)$ of the recipient is updated according to this rule: $lclock_p = \max(lclock_p , lclock_m)$

Distributed System Paradigms (20)

Approach 1 (symmetric algorithms) for designing total order algorithms
Assumptions:
- all messages are sent to all participants (always met by multicast)
- a deterministic rule is used to order messages having the same *lclock*

Apply algorithm III
---> messages are delivered in the same order at every process
---> all processes execute the same steps (therefore symmetric algorithms)

Pro´s:
- simple to implement, relying on logical clocks. Even these can be abandoned. if synchronized clocks are available to timestamp messages.
- no additional exchange of control messages

Con´s:
- Latency of message delivery is determined by the rate of the slowest process
- Symmetry assumption may be too strong (not all processes are sending processes --> add. control mess.)

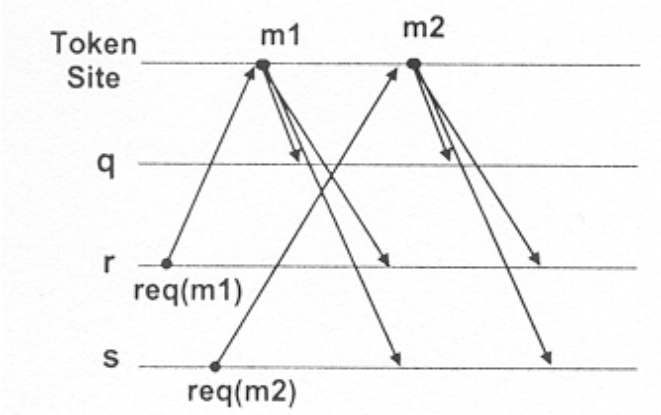
Approach II (token site)

Idea: select one process (*sequencer, token site)* to do the job of ordering
- all messages are sent to the sequencer which assigns a unique sequence number to them
- It retransmits them back to all intended recipients

Distributed System Paradigms (21)

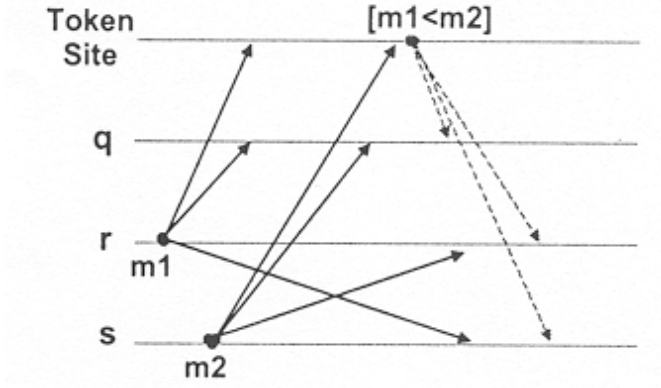
Example I:

(sequencer sends the messages)



Example II:

(sequencer only sends the sequence numbers)



- This scheme performs best when messages are sent by the sequencer itself ---> Some systems dynamically move the token to the most busiest node, or rotate the token among all nodes.
- In order to preserve the ordering information in case of a sequencer crash, the Chang/Maxemchuk protocol variant requires the sequence numbers to be known by a quorum of nodes before delivering the messages