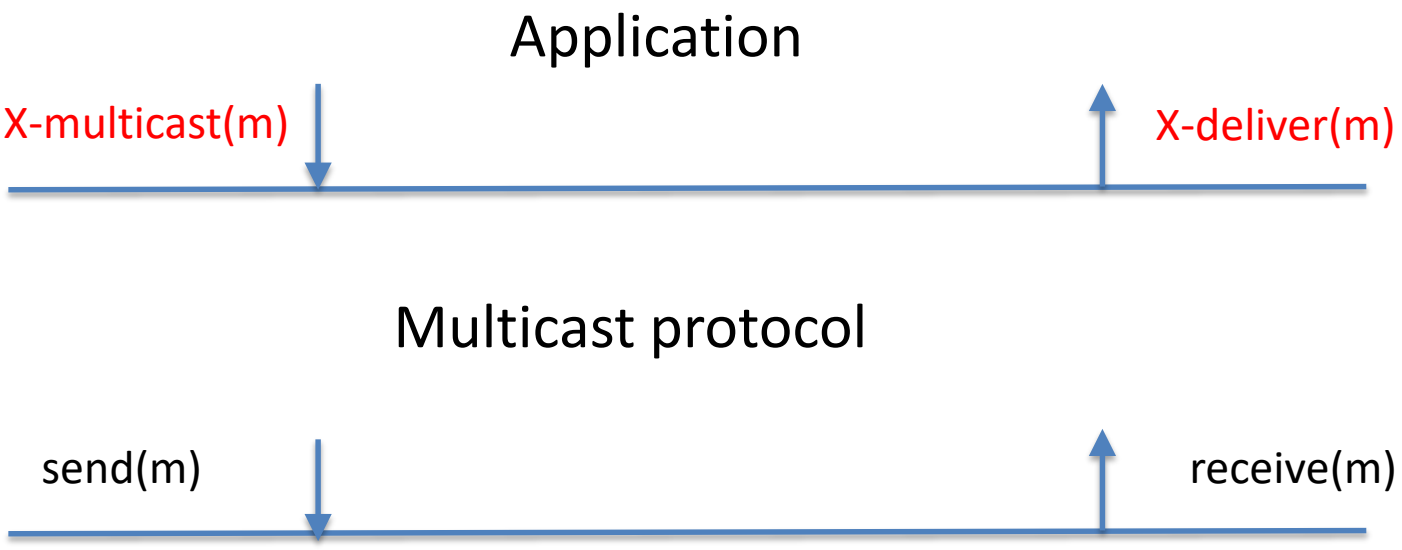
Multicast protocols (protocolos de radiado)

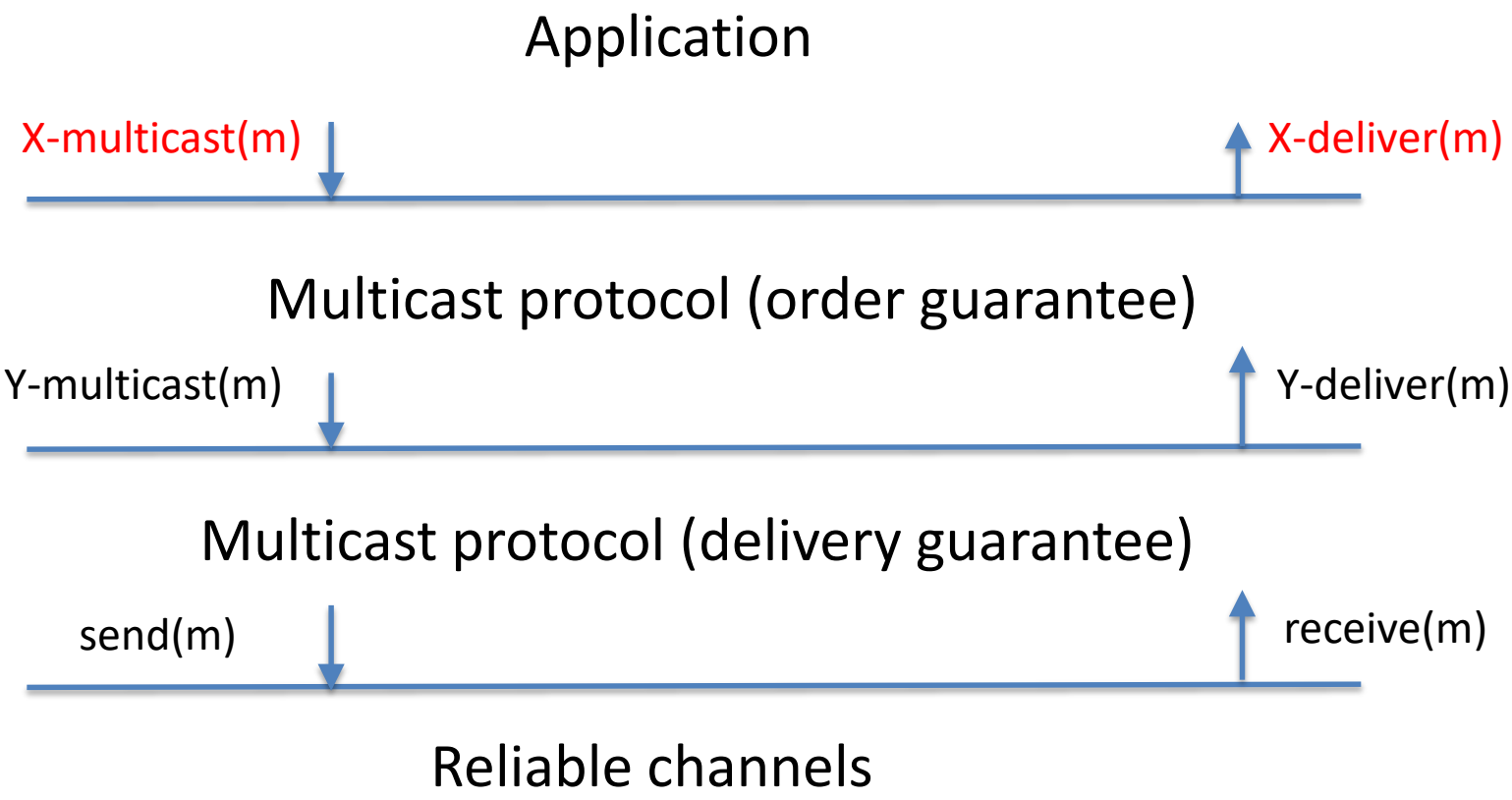
Ernesto Jiménez

Prometeo

Multicast Protocol one layer architecture



Multicast Protocol two layer architecture

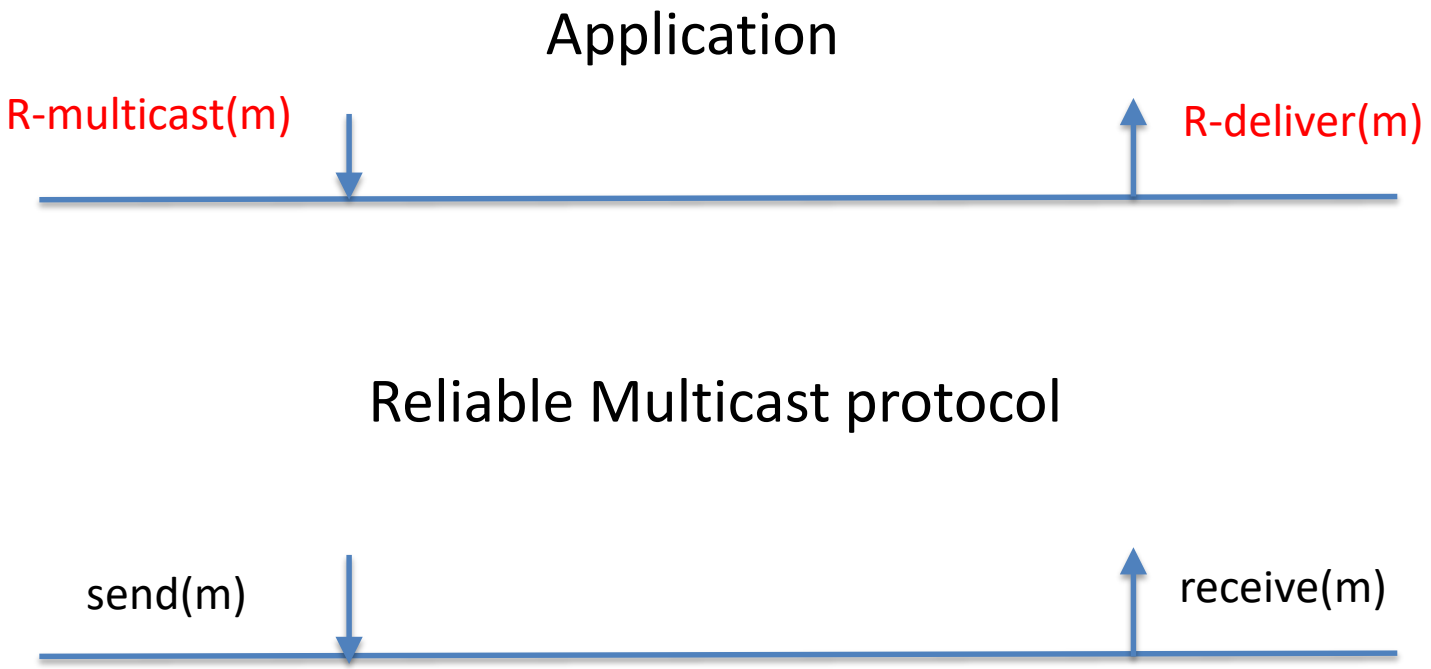


Reliable Channel properties

- Validity: If a correct process send(m), this message m must be eventually receive(m) in all correct processes.

- Integrity: If a process receive(m), then this message m has been previously send(m), and a process receive(m) at most once.

Reliable Multicast protocol architecture



Reliable Multicast protocol properties

- Validity: if a correct process R-multicast a message m, m must eventually R-deliver it in all correct processes.
- Integrity: if a process R-deliver(m), then this message m has been R-multicast by some process, and each process must R-deliver m at most once.
- Agreement: if a correct process R-deliver a message m, it must eventually R-deliver m in all correct processes.

Reliable multicast protocol

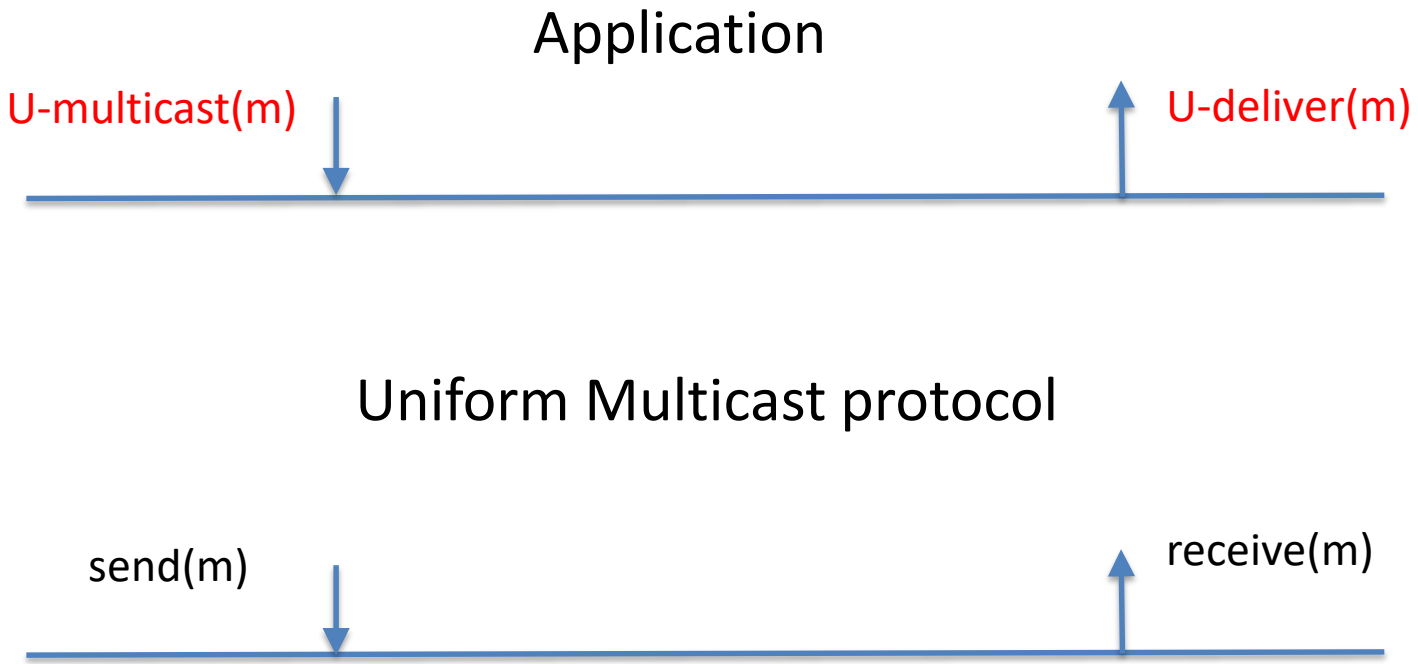
init:
received$_i$ ← Ø; //set of messages received by process p_i

when R-multicast (m) is executed by p_i's application:
send (m) to all processes.

when receive (m) **is executed by p_i's system:**
if (m ∉ received$_i$) **then**
received$_i$ ← received$_i$ ∪ {m};
R-deliver(m);
send (m) to all processes
end if.

Process p_i pseudocode

Uniform Multicast Protocol architecture



Uniform Multicast protocol properties

- Validity: if a correct process U-multicast a message m, m must eventually U-deliver it in all correct processes.
- Integrity: if a process U-deliver(m), then this message m has been U-multicast by some process, and a process must U-deliver m at most once.
- Uniform Agreement: if a process (correct or not) U-deliver a message m, it must eventually U-deliver m in all correct processes.

Uniform multicast protocol

init:
next_seqnum$_i$ ← 1; received$_i$ ← ∅; set of messages received by process p_i

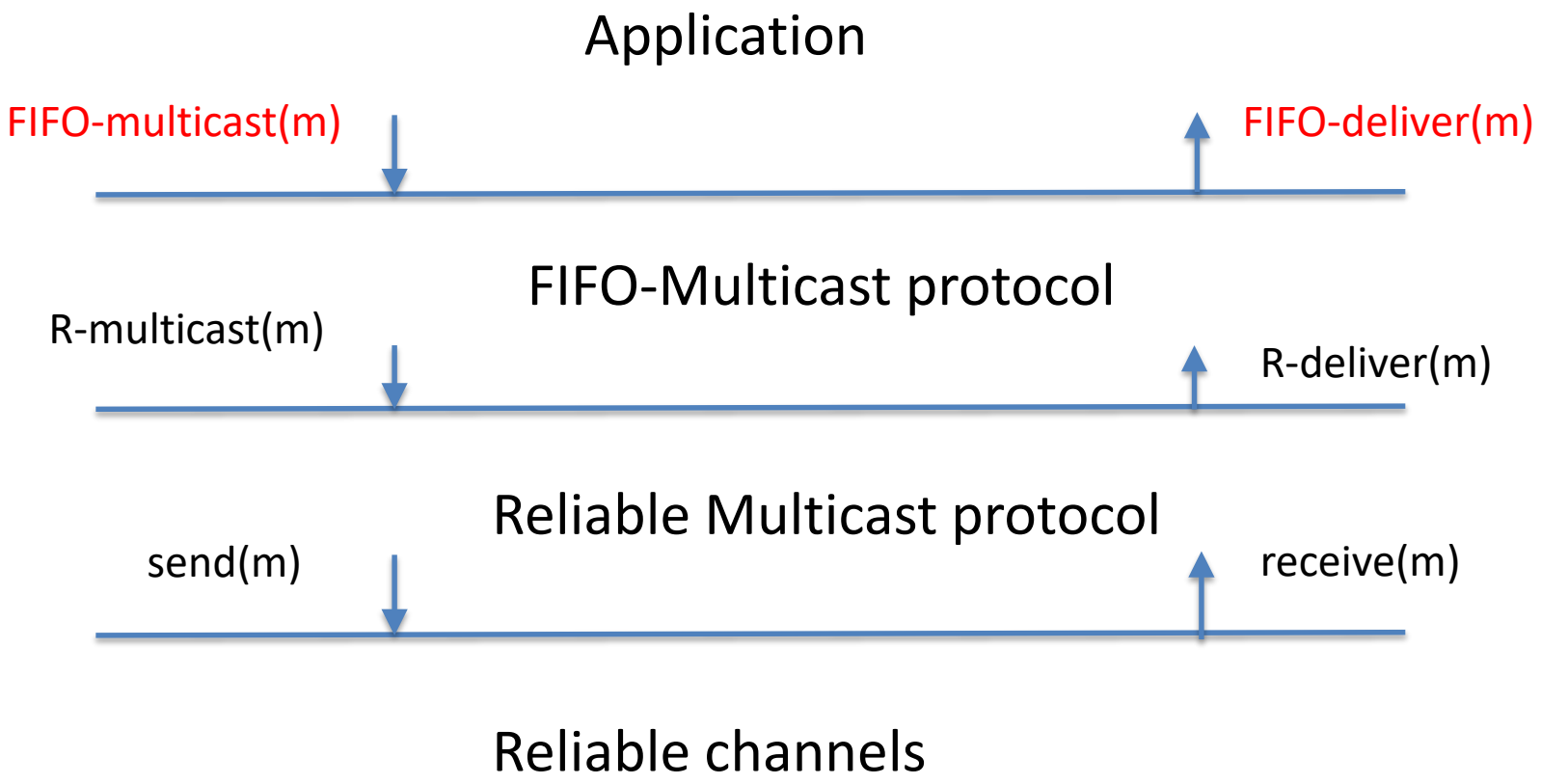
when U-multicast (m) is executed by p_i's application:
m.sender=i; m.seqnum ← next_seqnum$_i$;
send (m) to all processes; next_seqnum$_i$ ← next_seqnum$_i$ + 1;

when receive (m) **is executed by p_i's system:**
if (m ∉ received$_i$) **then**
received$_i$ ← received$_i$ ∪ {m}; pending$_i$ ← pending$_i$ ∪ {m};
send(ACK, i) to m.sender;
send (m) to all processes
end if.

when receive (ACK, id$_j$) **is executed by p_i's system:**
// set of ACK senders
let ACK-IDs(m) = {id(p) : (ACK,id$_j$) has been received **and**
(ACK,id$_j$).sender = p **and** m.sender= id$_j$};
while (∃m ∈ (pending$_i$) : m.sender = id$_j$) **and**
(|ACK-IDs(m) ∪ m.sender ∪ {i} | > ∏/2) **do** // A majority of processes received m
U-deliver(m);
pending$_i$ ← pending$_i$ \ {m}
end while.

Process p_i pseudocode

FIFO Reliable Multicast Protocol architecture



FiFO-Reliable Multicast protocol properties

- Validity: if a correct process FIFO-multicast a message m, m must be FIFO-deliver in all correct processes.
- Integrity: if a process FIFO-deliver(m), then m has been FIFO-multicast by some process, and a process must FIFO-deliver m at most once.
- Agreement: if a correct process FIFO-deliver a message m, it must be FIFO-deliver in all correct processes.
- FIFO message delivery. If a process p_i FIFO-multicast(m) and then FIFO-multicast(m'), then no process FIFO-deliver(m') unless it has previously FIFO-deliver(m).

FIFO reliable multicast protocol

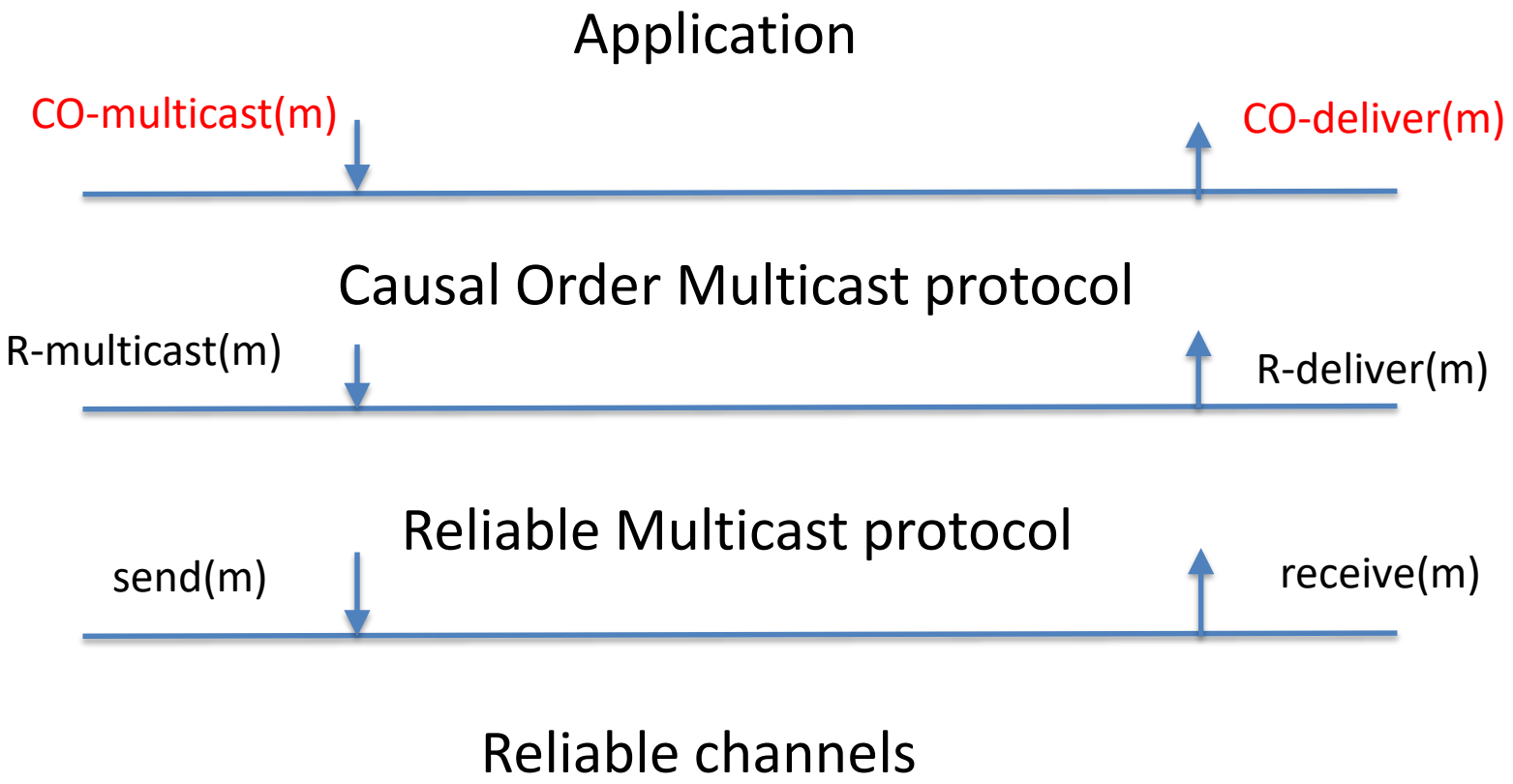
init:
  $seqnum_i \leftarrow 1$; //sequence number of process p_i
  $\forall k$, $next\_seqnum_i[k] \leftarrow 1$; //next message to delivered by process p_i
  $msg\_set_i \leftarrow \emptyset$; //set of messages waiting to be delivered by process p_i

when FIFO-multicast (m) is executed by p_i's application:
  m.sender ← i;
  m.seqnum ← $seqnum_i$;
  R-multicast (m) ;
  $seqnum_i \leftarrow seqnum_i + 1$.

when R-deliver (m) **is executed by p_i's system:**
  let j = m.sender;
  $msg\_set_i \leftarrow msg\_set_i \cup \{m\}$;
  while ($\exists$m' $\in$ msg_set_i : (*m'.sender* = j) **and** ($next\_seqnum_i[j]$ = *m'.seqnum*)) **do**
    FIFO-deliver(m');
    $msg\_set_i \leftarrow msg\_set_i \setminus \{m'\}$;
    $next\_seqnum_i[j] \leftarrow next\_seqnum_i[j] + 1$
  end while.

Process p_i pseudocode

Causal Order Reliable Protocol architecture



Causal Reliable Multicast protocol properties

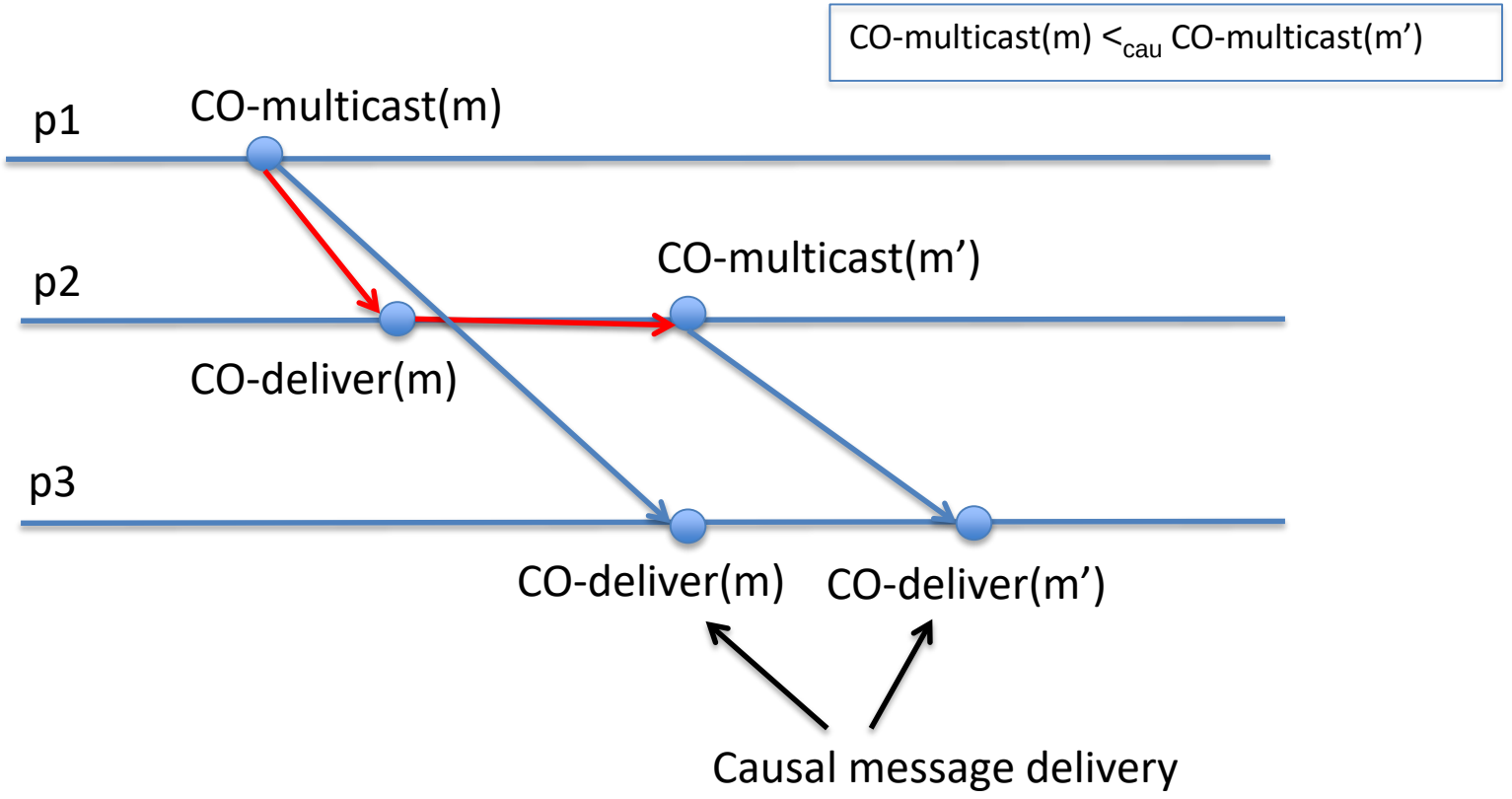
- Validity: if a correct process CO-multicast a message m, m must be delivered in all correct processes.
- Integrity: if a process CO-deliver(m), then this message m has been CO-multicast by some process, and a process must CO-deliver m at most once.
- Agreement: If a correct process CO-deliver a message m, this message m must be CO-deliver in all correct processes.
- Causal message delivery ($<_{cau}$): if CO-multicast(m) causally precedes CO-multicast(m'), then no process CO-deliver(m') unless it has previously CO-deliver(m).

Causal Reliable Multicast protocol properties (cont.)

- Causal Order (CO-multicast(m) $<_{cau}$ CO-multicast(m')): if some of the following two cases happens:
  1. A process p_k invokes CO-multicast(m) before this same process p_k invokes CO-multicast(m').
  2. A process p_k invokes CO-multicast(m), and a process p_s invokes CO-deliver(m) before this same process p_s invokes CO-multicast(m').

- Causal message delivery ($<_{cau}$): every process must preserve causal order CO-multicast(m) $<_{cau}$ CO-multicast(m'), for all messages m and m', when it executes CO-deliver.

Causal Reliable Multicast protocol properties (cont.)



Causal Order reliable multicast protocol

init:
∀k, next_seqnum$_i$ [k] ← 1; // seq. number of next messages to be applied by p_i
msg_set$_i$ ← ∅; // set of messages waiting to be delivered

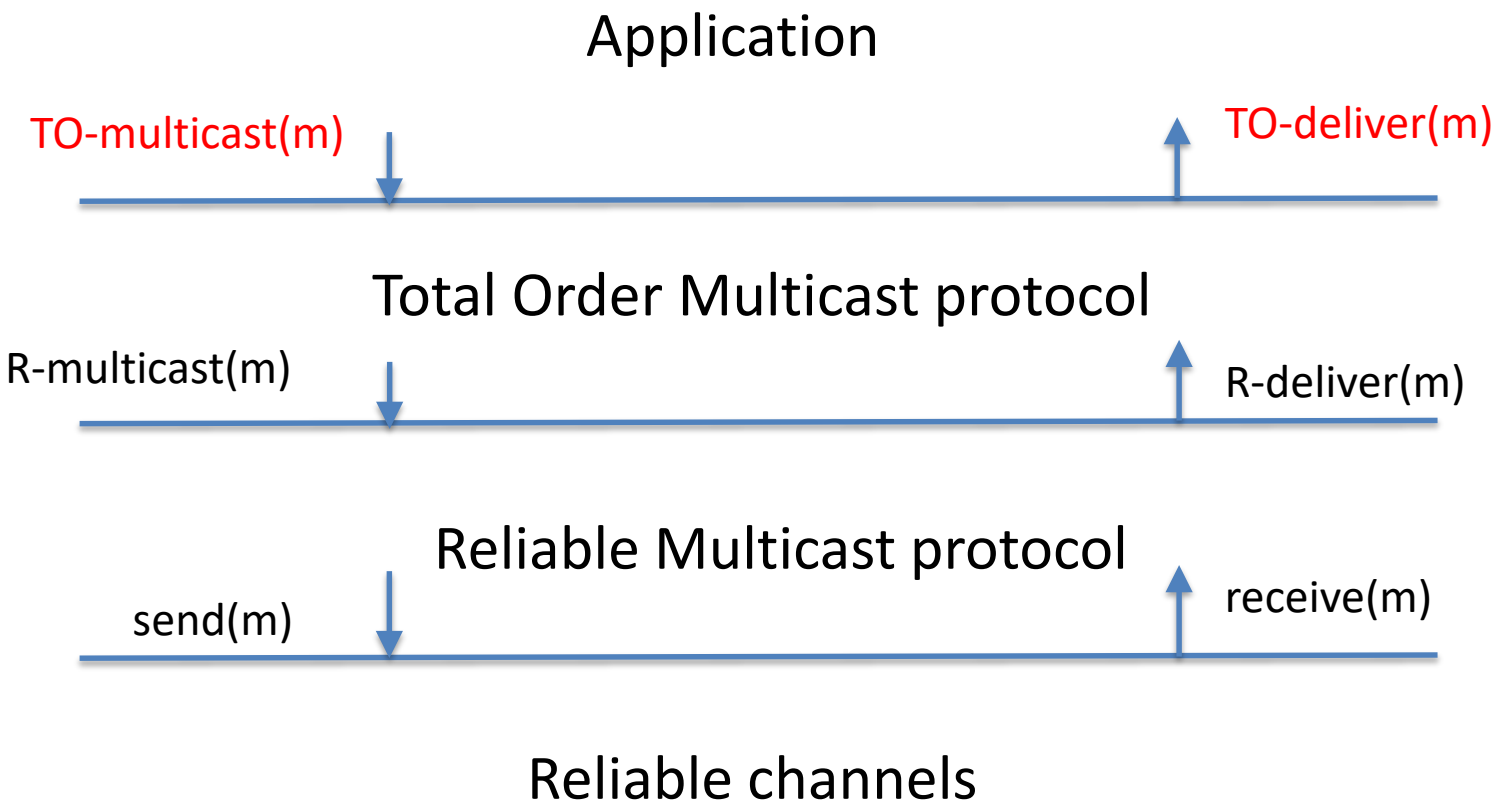
when CO-multicast (m) is executed by p_i's application:
m.sender ← i; *m.seqnum* ← next_seqnum$_i$;
R-multicast (m) ;
CO-deliver (m); // immediate delivery of p_i's message
next_seqnum$_i$ [i] ← next_seqnum$_i$[i] + 1.

when R-deliver (m) **is executed by p_i's system:**
let j = *m.sender*;
if (i ≠ j) **then**
msg_set$_i$ ← msg_set$_i$ ∪ {m};
while (∃m' ∈ msg_set$_i$: CausalCondition(m')) **do**
let j = *m'.sender*;
CO-deliver(m');
next_seqnum$_i$ [j] ← *m'.seqnum[j]* + 1;
msg_set$_i$ ← msg_set$_i$ \ {m'}
end while
end if.

CausalCondition(m') ≡ (∀k : next_seqnum$_i$ [k] ⩾ *m'.seqnum[k]*)

Process p_i pseudocode

Total order reliable Protocol architecture



Total Order Reliable Multicast protocol properties

- Validity if a correct process TO-multicast a message m, this message m must be delivered in all correct processes.
- Integrity: if a process TO-deliver(m), then m has been TO-multicast by some process, and a process must TO-deliver m at most once.
- Agreement: if a correct process TO-deliver a message m, this message m must be TO-deliver in all correct processes.
- Total order message deliver. If a process TO-deliver(m) and then TO-deliver(m'), then no process TO-deliver(m') unless it has previously TO-deliver(m).

Total Order reliable multicast

init:
next_seqnum$_i$ ← 1; //next message to delivered
msg_set$_i$ ← ∅; // set of messages waiting to be delivered

when TO-multicast (m) is executed by p_i's appl.:
send (m) to p_s; // sequencer

when R-deliver (m) **is executed by p_i's system:**
msg_set$_i$ ← msg_set$_i$ ∪ {m};
while (∃ m' ∈ msg_set$_i$:
next_seqnum$_i$ = *m'.seqnum*) **do**
TO-deliver(m');
msg_set$_i$ ← msg_set$_i$ \ {m'};
next_seqnum$_i$ ← next_seqnum$_i$ + 1;
end while.

Process p_i pseudocode

init:
seq_glob$_s$ ← 1; // global sequence number

when receive (m) **is executed by p_i's system:**
m.seqnum ← seq_glob$_s$;
R-multicast (m);
seq_glob$_s$ ← seq_glob$_s$ + 1;

Process sequencer (p_s) pseudocode

NOTE: if the process sequencer crashes, then alive processes must detect it reliably and elect a new sequencer.

FIFO+Total Order reliable multicast

init:
next_seqnum$_i$ ← 1; //next message to delivered
msg_set$_i$ ← ∅; // set of messages waiting to be delivered

when FIFO+TO-multicast (m) is executed by p_i's appl.:
m.sender ← i;
m.segnum ← next_seqnum$_i$;
send (m) to p_s; // sequencer
next_seqnum$_i$ ← next_seqnum$_i$ + 1;

when R-deliver (m) **is executed by p_i's system:**
msg_set$_i$ ← msg_set$_i$ ∪ {m};
while (∃ m' ∈ msg_set$_i$:
next_seqnum$_i$ = *m'.seqnum*) **do**
FIFO+TO-deliver(m');
msg_set$_i$ ← msg_set$_i$ \ {m'};
next_seqnum$_i$ ← next_seqnum$_i$ + 1;
end while.

Process p_i pseudocode

init:
seq_glob$_s$ ← 1; // global sequence number
∀j, next_FIFO$_s$ [j] ← 1;
msg_set$_s$ ← ∅;

when receive (m) **is executed by p_i's system:**
msg_set$_s$ ← msg_set$_s$ ∪ {m};
while ((∃ m ∈ msg_set$_s$) and (∃ k = m.sender) :
next_FIFO$_s$[k]= *m.seqnum*) **do**
m.seqnum ← seq_glob$_s$;
R-multicast (m);
seq_glob$_s$ ← seq_glob$_s$ + 1;
msg_set$_i$ ← msg_set$_i$ \ {m};
next_FIFO$_s$[k] ← next_FIFO$_s$[k] + 1;
end while.

Process sequencer (p_s) pseudocode

NOTE: if the process sequencer crashes, then alive processes must detect it reliably and elect a new sequencer.