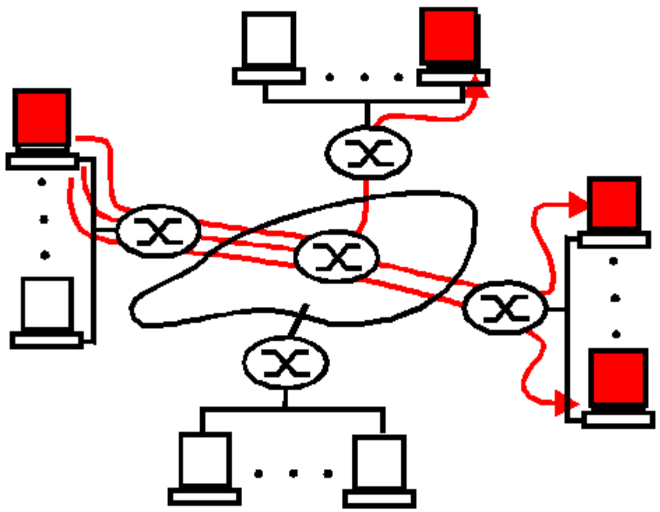
Lecture 28

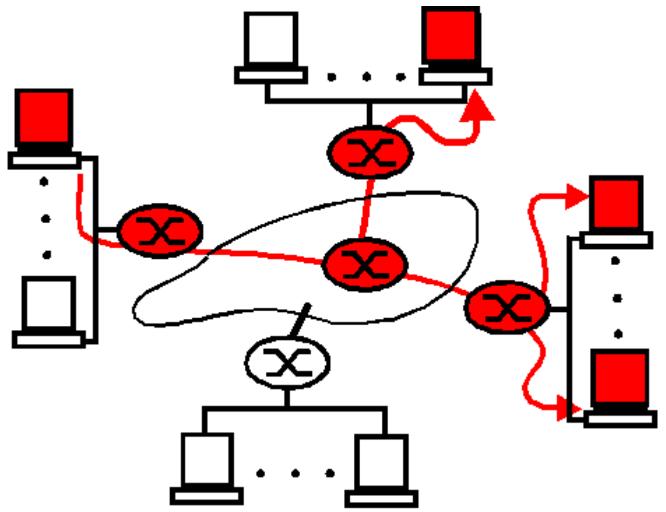
- Administration

Multicast communication

- Send message over a distribution tree.
- Use network hardware support for broadcast or multicast when it is available.
- Minimize the time and bandwidth utilization

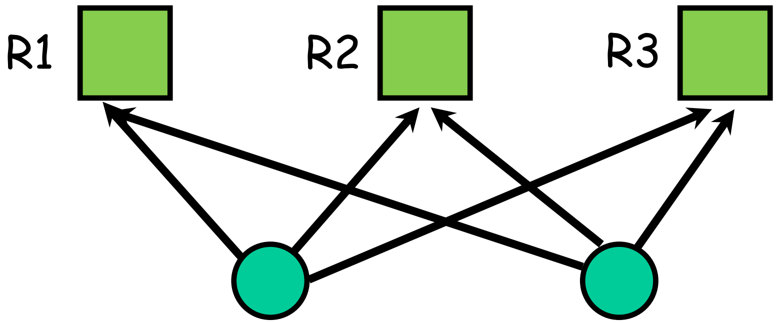


multicast via unicast



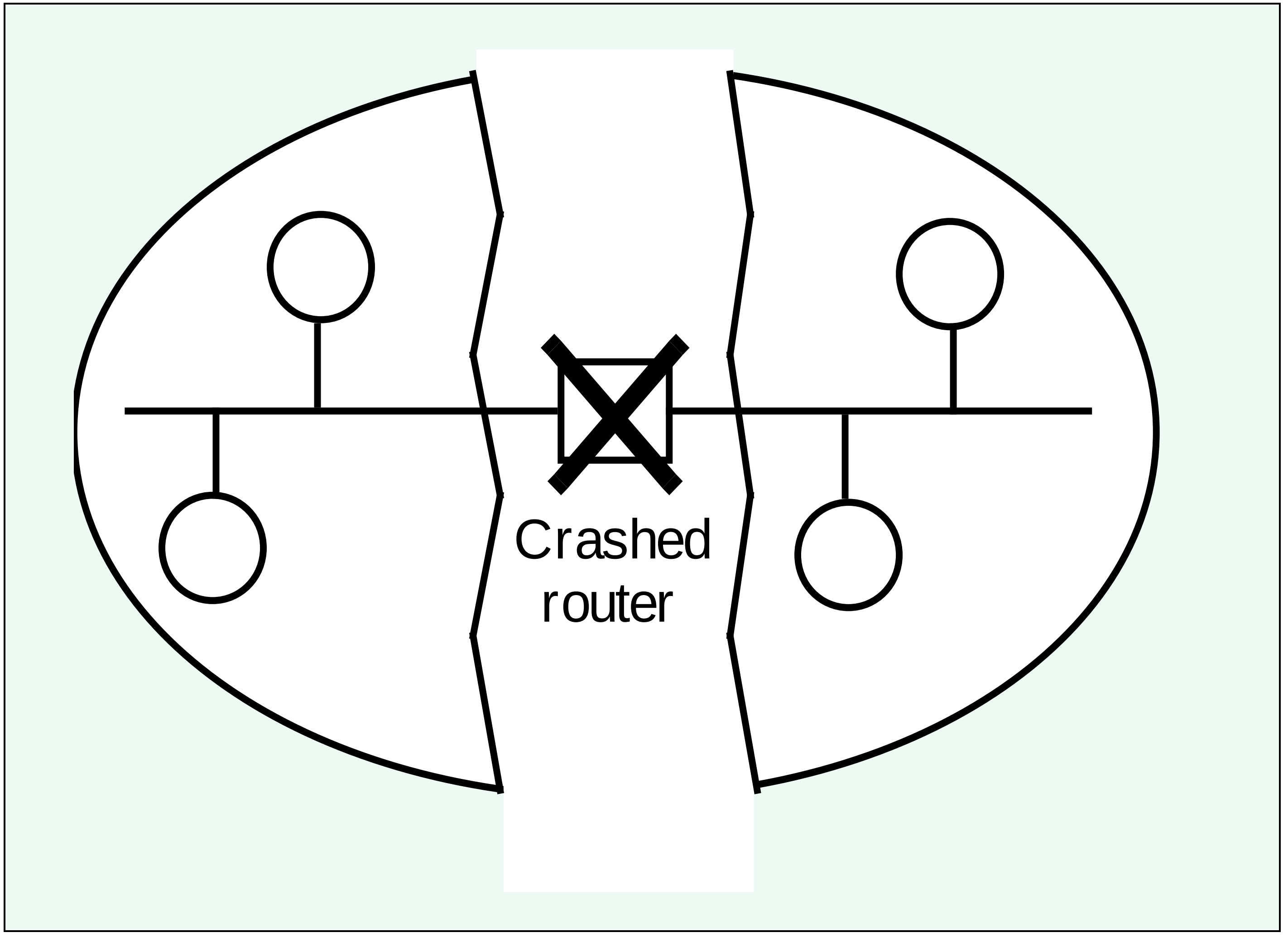
network multicast

Replicated Data Storage



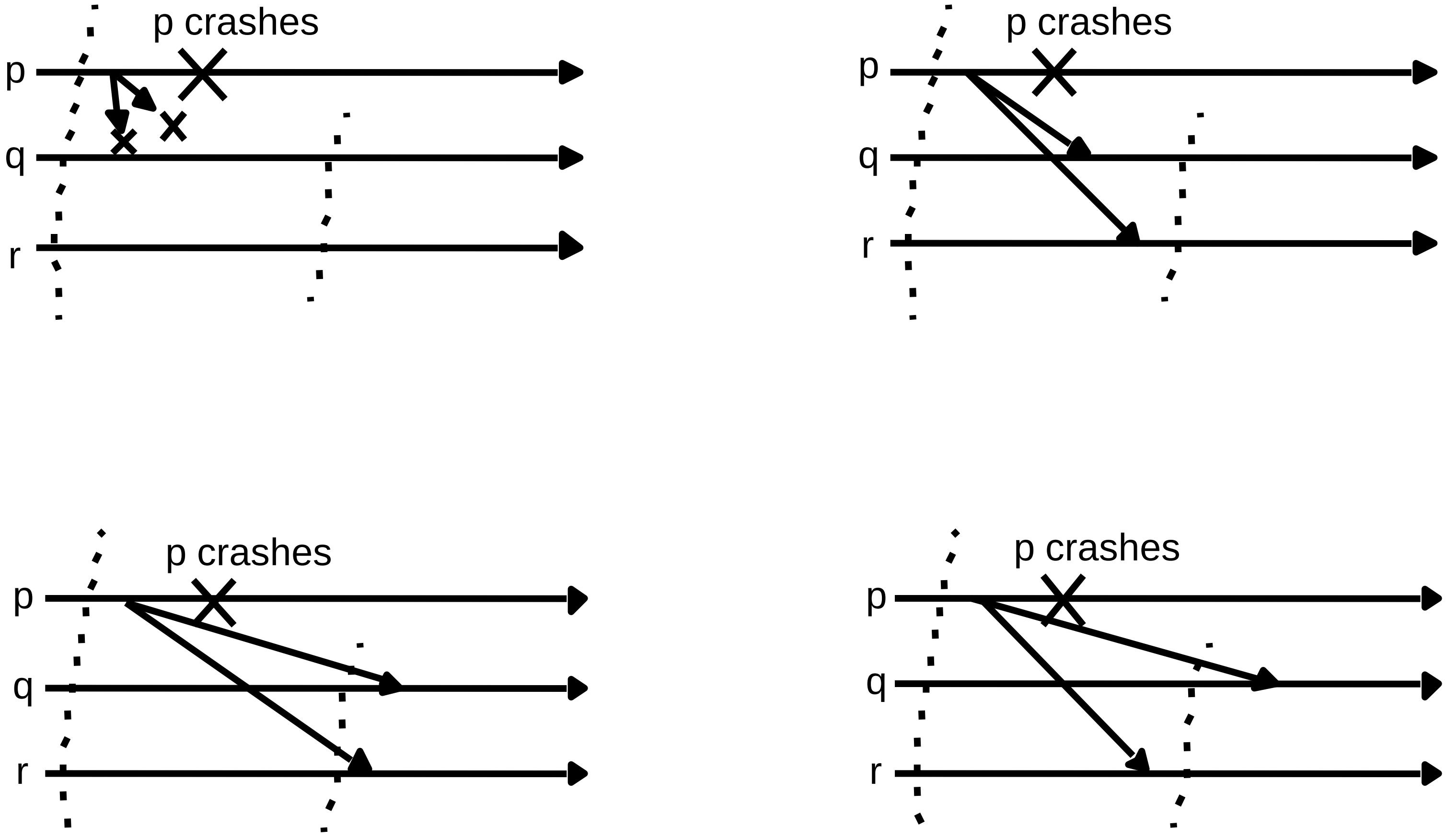
Two processes

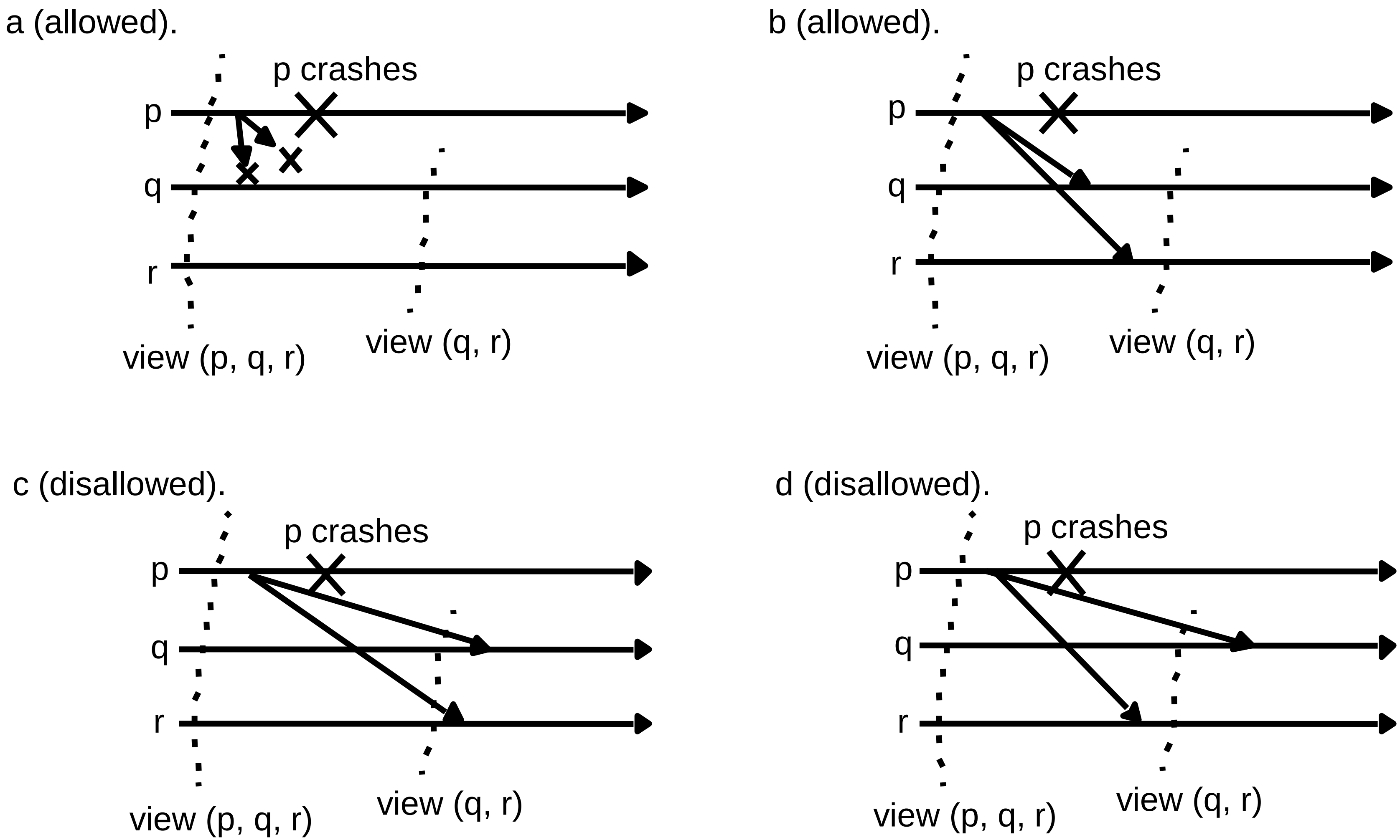
A network partition



Reliable Multicast

- Multicast - general idea
- Support
  - None
  - IP-multicast


p crashes
p
q
r
p crashes
p
q
r
p crashes
p
q
r
p crashes
p
q
r



a (allowed).
p crashes
p
q
r
view (p, q, r)
view (q, r)
b (allowed).
p crashes
p
q
r
view (p, q, r)
view (q, r)
c (disallowed).
p crashes
p
q
r
view (p, q, r)
view (q, r)
d (disallowed).
p crashes
p
q
r
view (p, q, r)
view (q, r)

Reliable Multicast Communication

- IP multicast uses UDP which means that it is not reliable.
- An IP multicast message may be lost part way and delivered to some, but not all, of the intended receivers.
- A process may still send a message to a group using TCP.
  - This is point to point i.e., this means that the process sends a message to the first replica, then sends a message to the second replica etc;
  - This is still not reliable. Consider what happens if the sender fails after sending to a subset of the group.
- Reliable multicast means that every message should be delivered to each current group member

Reliable Multicast

- Simple multicasting is sending a message to every process that is a member of a defined group. Reliable multicasting requires these properties:
- **Integrity**—a correct process sends a message to only a member of the group and does it only once.
- **Validity**—if a correct process sends a message, it will eventually be delivered.
- **Agreement**—if a message is delivered to a correct process, all other correct processes in the group will deliver it.

Reliability

Correct processes: those that never fail.

- Integrity

  A *correct* process *delivers* a message at most once.
- Validity

  A message from a *correct* process will be *delivered* by the process eventually.
- Agreement

  A message *delivered* by a *correct* process will be *delivered* by all other *correct* processes in the group.

$\Rightarrow$ Validity + Agreement = Liveness

B-multicast

- Assumption:
  - Reliable one-to-one *send* operation (e.g. TCP)

- Basic multicast
  - Requirement:
    - All correct processes will eventually deliver the message from the *correct* multicaster.
  - Implementation:
    - ***B-multicast(g, m)***: $\forall p \in g$: *send(p, m);*
    - *On receive(m) at p:* ***B-deliver(m)*** *at p.*

    $\Rightarrow$ Properties: integrity, validity.

R-multicast

- Reliable multicast
  - Requirements: integrity, validity, *agreement*
  - Implementation:
    - *Received := {};*
    - ***R-multicast(g, m)*** *at process p: B-multicast(g, m);*
    - *On B-deliver(m) at process q*

      if(m $\notin$ Received)

      Received := Received $\cup$ {m};

      if(q $\neq$ p) B-multicast(g, m);

      R-deliver(m)*;*

      end if

$\Rightarrow$ Inefficient: each message is sent |g| times to each process

Reliable Multicast Algorithm

When a message is delivered, the receiving process multicasts it. Duplicate messages are identified (possible by a sequence number) and not delivered.

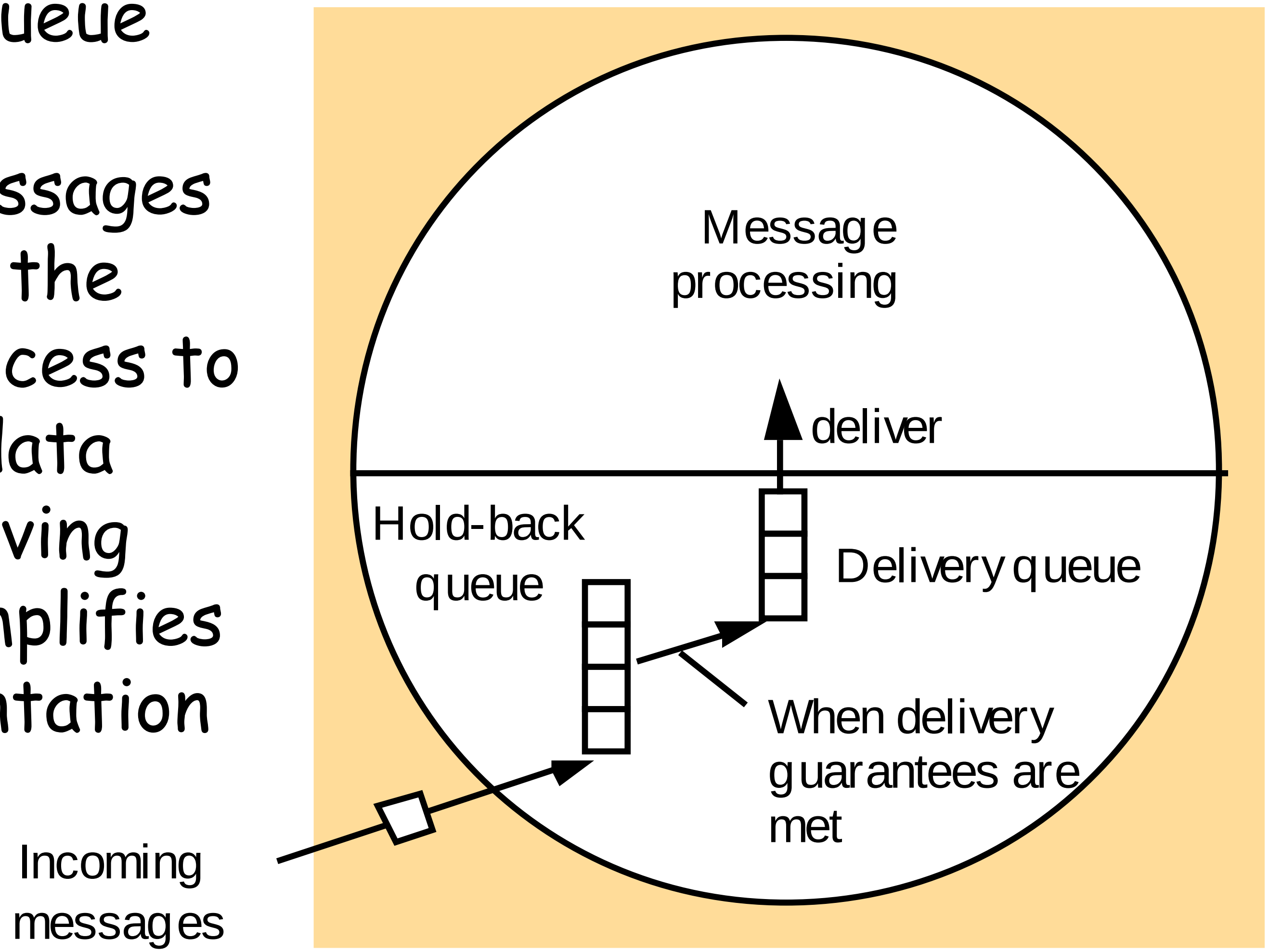
On initialization
  $Received := \{\};$

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

On B-deliver(m) *at process q with* $g = group(m)$
  if $(m \notin Received)$
  then
    $Received := Received \cup \{m\};$
    if $(q \neq p)$ *then B-multicast*(g, m); *end if*
    R-deliver m;
  end if

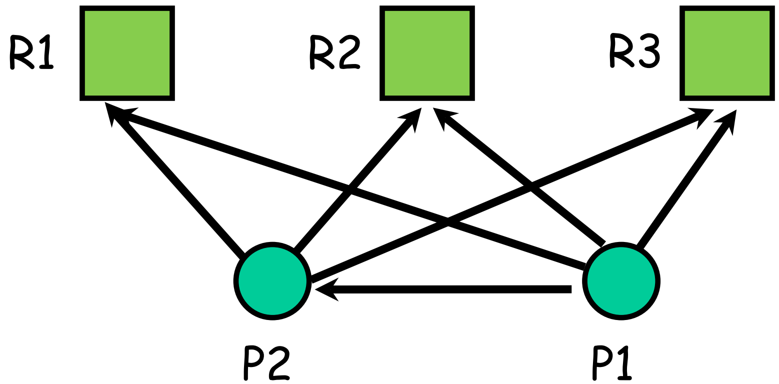
Hold-back queue

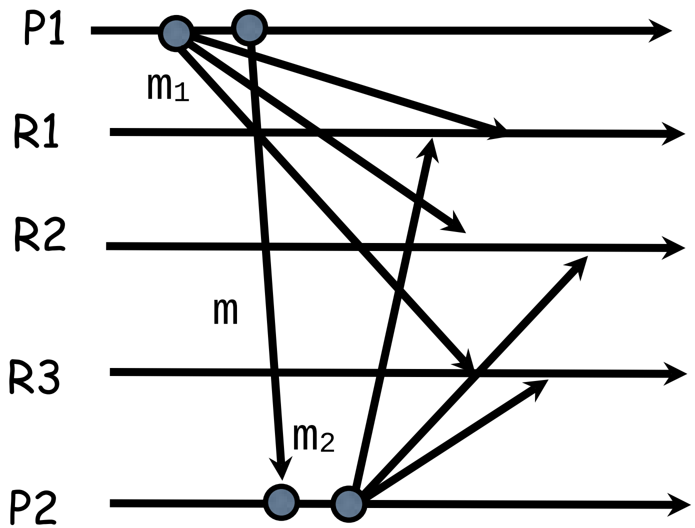
A holdback queue for arriving multicast messages that enables the receiving process to obtain metadata about an arriving messages simplifies the implementation of reliable multicast.



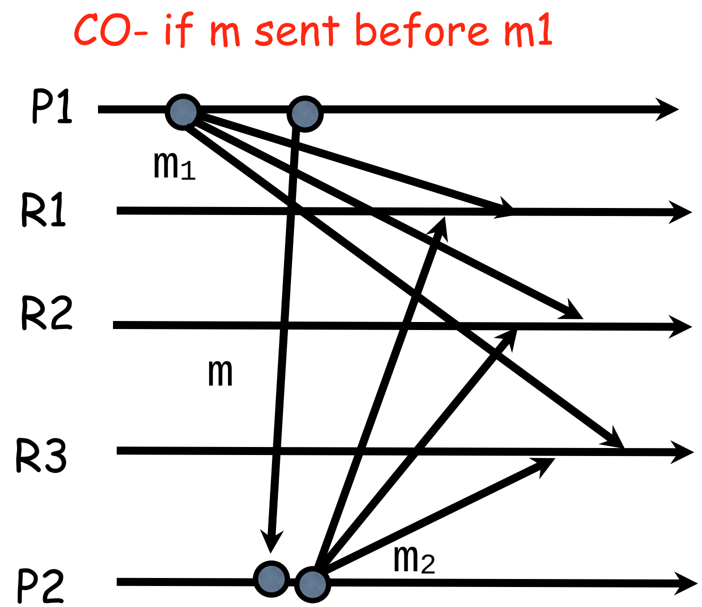
Ordered messages

- If it is important that messages be delivered in order, there are three types of ordering:
- FIFO—(First-in, first-out) if a correct process delivers a message before another, every correct process will deliver the first message before the other.
- Casual—any correct process that delivers the second message will deliver the previous message first.
- Total—if a correct process delivers a message before another, any other correct process that delivers the second message will deliver the first message first.





Not CO -- Processes see different view



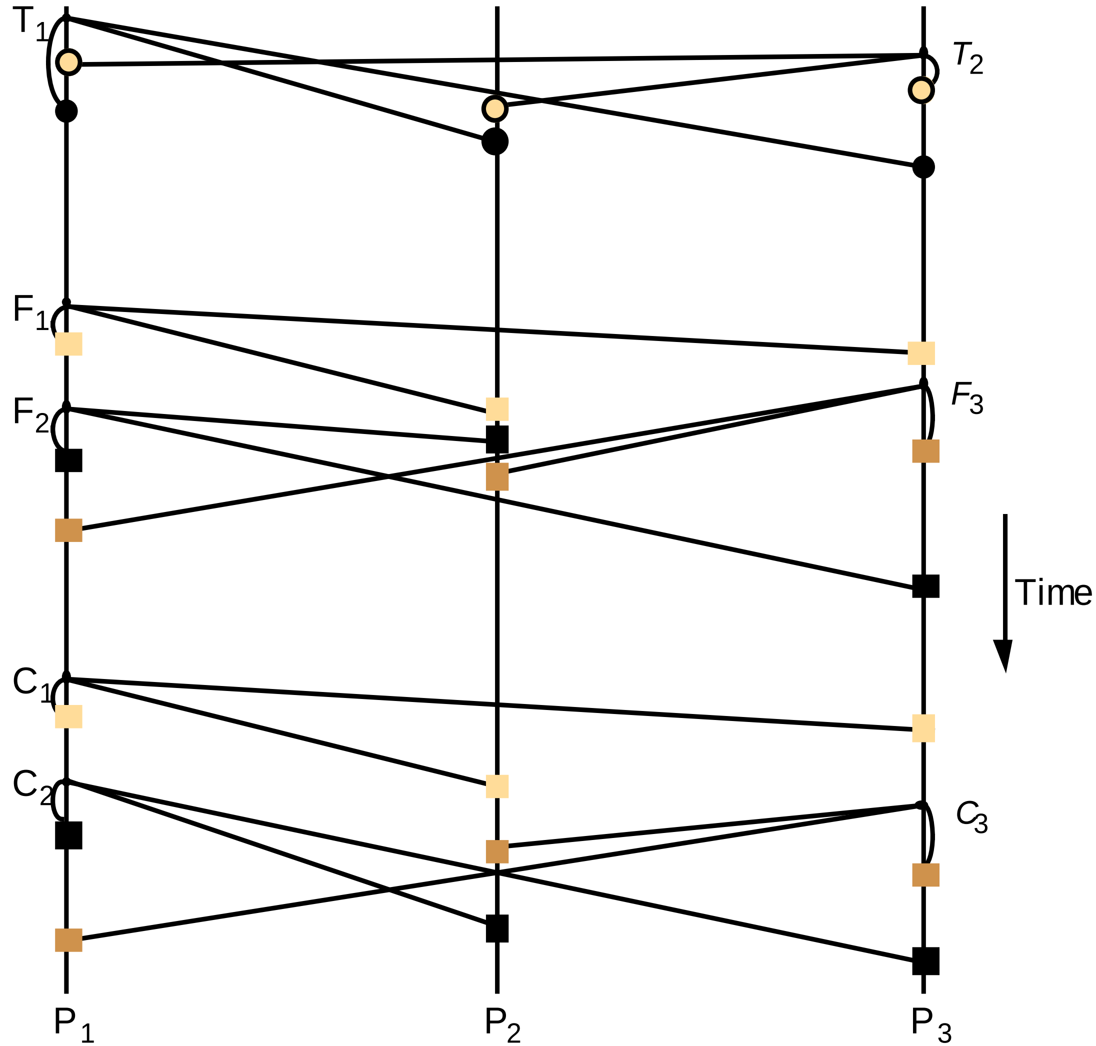
Not CO -- even when the update is consistent

Comments on Ordering

- Note that FIFO ordering and casual ordering are only partial orders. Not all messages are sent by the same sending process. In addition some multicasts are concurrent, not able to be ordered by happened-before.
- Note that T_1 and T_2 are delivered in opposite order to the physical time of message creation. Total order demands consistency, but not a particular order.

Total, FIFO and causal ordering of multicast messages

Notice the consistent ordering of totally ordered messages T_1 and T_2, the FIFO-related messages F_1 and F_2 and the causally related messages C_1 and C_3 – and the otherwise arbitrary delivery ordering of messages.



FIFO-ordered Multicast

- FIFO-ordered multicast:
  - Assumption:
    - a process belongs to at most one group.
  - Implementation:
    - Local variables at *p*: $S_p = 1$, $R_p[\ |g|\ ]=\{0\}$;
    - ***FO-multicast(g, m)*** *at p:*

      B-multicast(g, <m, S_p>);

      S_p*++;*
    - *On B-deliver(<m, S>) from q:*

      if($S = R_p[q] + 1$)

      FO-deliver(m)*;*

      $R_p[q] := S$;

      else if($S > R_p[q] + 1$)

      place <m, S> in the queue until $S = R_p[q] + 1$;

      FO-deliver(m)*;*

      $R_p[q] := S$;

      end if

Total Communication

item	From	Subject
23	A. Hanlon	Mach
24	G. Joseph	Microkernels
25	A. Hanlon	Re: Microkernels
26	T.L. Heureux	RPC performance
27	M. Walker	Re: Mach