CSE 486/586 Distributed Systems

Reliable Multicast (part 1)

Slides by Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time

- Global state
  - A union of all process states
  - Consistent global state *vs.* inconsistent global state
- The snapshot algorithm
  - Take a snapshot of the local state
  - Broadcast a marker message to tell other processes
  - Start recording all incoming messages for each channel until receiving a marker on that channel
  - Outcome: a consistent global state

Today

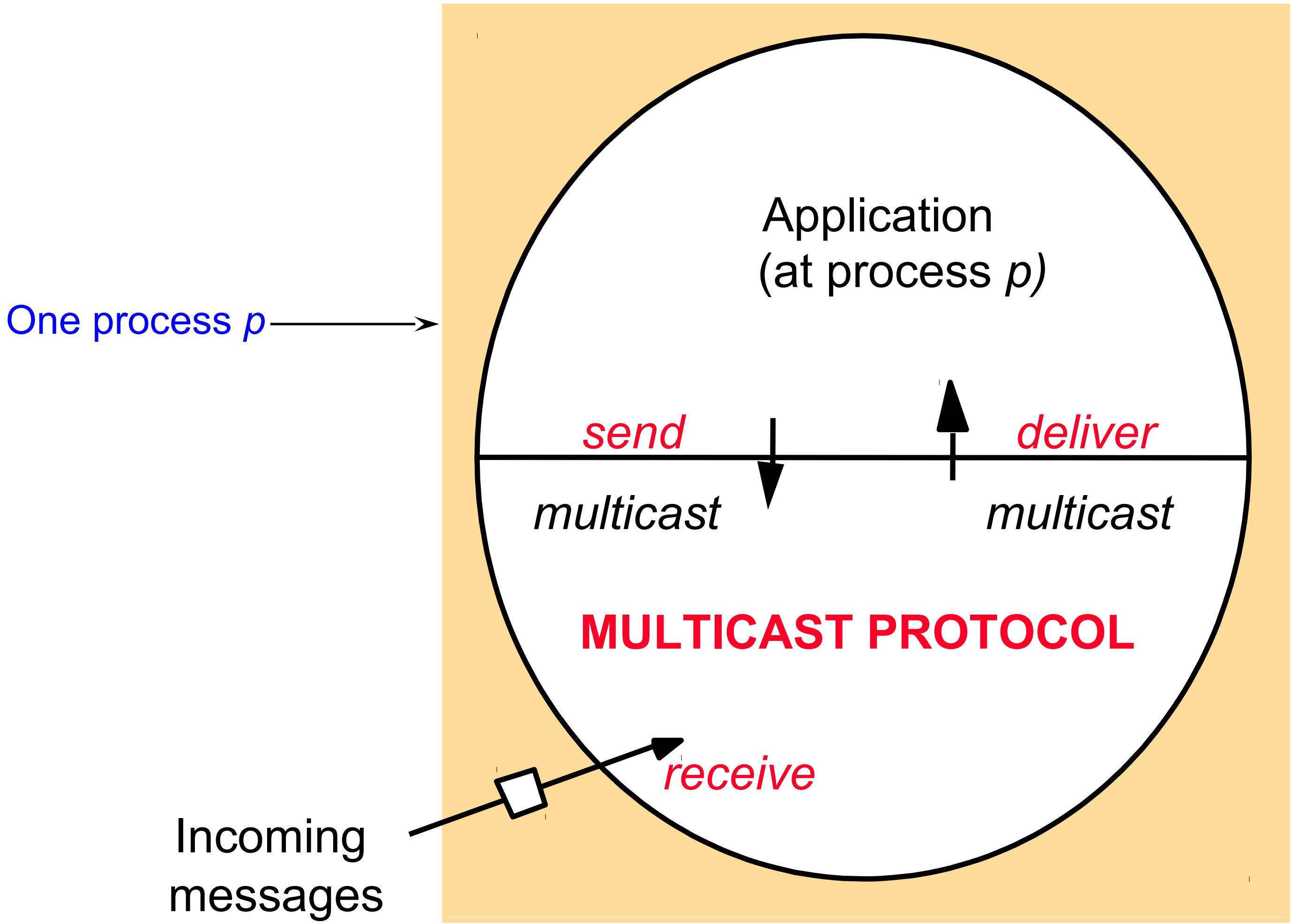
- How does a group of processes communicate?
- Unicast (best effort or reliable)
  - One-to-one: message from process *p* to process *q*.
  - Best effort: message may be delivered, but will be intact
  - Reliable: message will be delivered intact
- Broadcast
  - One-to-all: Message from process *p* to all processes
  - Impractical for large networks
- Multicast
  - One-to-many: "local" broadcast within a group *g* of processes
- What are the issues with multicast?
  - Processes crash (we assume crash-stop failures)
  - Messages get delayed

Why: Examples

Why: Examples

- Akamai's Configuration Management System (called ACMS)
  - A core group of 3-5 servers.
  - Continuously multicast the latest updates to each other.
  - After an update is reliably multicast within this group, it is then sent out to all the (1000s of) servers Akamai has all over the world.
- Air Traffic Control System
  - Commands by one ATC need to be ordered and (reliably) multicast out to other ATCs.
- Newsgroup servers
  - Multicast to each other in a reliable and ordered manner.

The Interface



What: Properties to Consider

- Liveness: guarantee that something good will happen eventually
  - From the initial state, there exists a reachable state where the predicate becomes true.
  - "Guarantee of termination" is a liveness property
- Safety: guarantee that something bad will never happen
  - For any state reachable from the initial state, the predicate is false.
  - Deadlock avoidance algorithms provide safety
- Liveness and safety are used in many other CS contexts.

Basic Multicast (B-multicast)

- A straightforward way to implement B-multicast is to use a reliable one-to-one send (unicast) operation:
  - B-multicast(*g*,*m*): for each process *p* in *g*, send(*p*,*m*).
  - receive(*m*): B-deliver(*m*) at *p*.
- Guarantees?
  - All processes in *g* eventually receive every multicast message…
  - … as long as the sender doesn't crash
  - This guarantee is not so good
- What guarantees do we want?

Reliable Multicast Goals

- Integrity: A correct (*i.e.*, non-faulty) process *p* delivers a message *m* at most once.
  - "Non-faulty": doesn't deviate from the protocol or crash-stop
- Agreement: If a correct process delivers message *m*, then all the other correct processes in group(*m*) will eventually deliver *m*.
  - Property of "all or nothing."
- Validity: If a correct process multicasts (sends) message *m*, then it will eventually deliver *m* itself.
  - Guarantees liveness to the sender.
- Validity and agreement together ensure overall liveness: if some correct process multicasts a message *m*, then, all correct processes deliver *m* too.

Overview of Reliable Multicast

- Keep a history of messages
  - Integrity: at-most-once delivery
- Every host repeats each new message upon receipt
  - Agreement: even if the sender fails, *m* will be delivered if one correct process received it
- Processes self-deliver
  - Validity

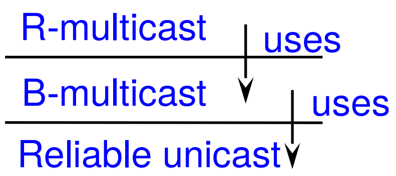
Reliable R-Multicast Algorithm

On initialization:

Received := {};

For process *p* to R-multicast message *m* to group *g*:

B-multicast(*g*,*m*);



($p \in g$ is included as destination)

On B-deliver(*m*) at process *q* with *g* = group(*m*):

if ($m \notin Received$):

Integrity

Received := *Received* ∪ {*m*};

if ($q \neq p$):

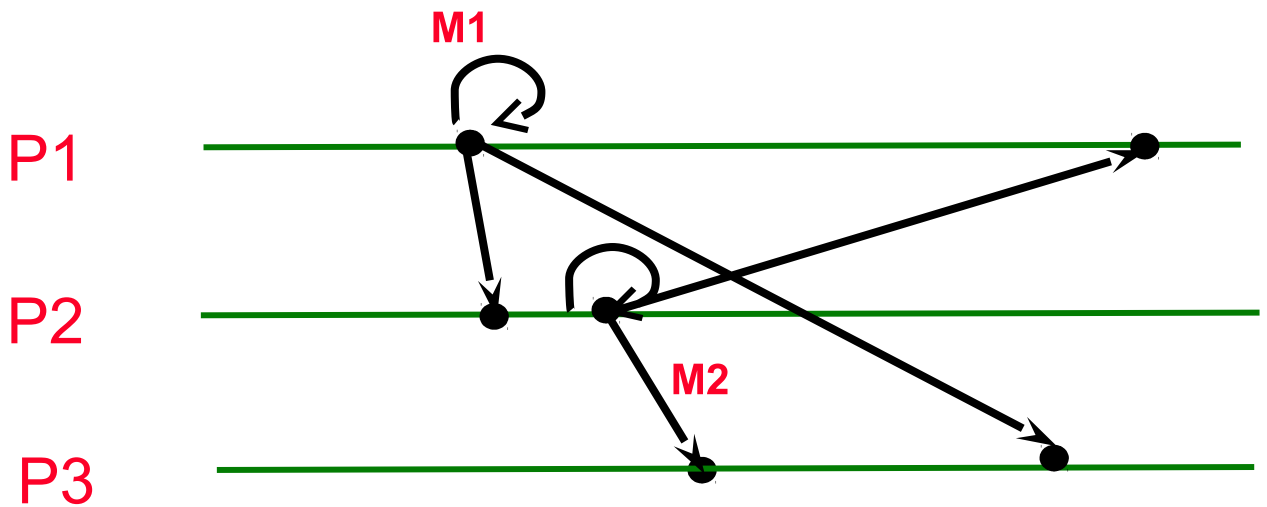
B-multicast(*g*,*m*);

Agreement

R-deliver(*m*)

Validity

Ordered Multicast Problem



- Each process delivers received messages independently.
  - What is the order of delivery for each process if they deliver as soon as they receive?
- There are other possibilities: what should we use?
- Three meaningful types of ordering
  - FIFO, Causal, Total

FIFO Ordering

- Message delivery in every process should preserve the sending order for each individual process.
- Messages from different processes can be interleaved in any order!
- With these sends:
  - P1: m0, m1, m2
  - P2: m3, m4, m5
  - P3: m6, m7, m8
- Are these FIFO?
  - P1: m0, m3, m6, m1, m4, m7, m2, m5, m8
  - P2: m0, m4, m6, m1, m3, m7, m2, m5, m8
  - P3: m6, m7, m8, m0, m1, m2, m3, m4, m5

Causal Ordering

- Message delivery at each individual process preserves the happened-before relationship across all processes
- Each process may deliver messages in a different order
- For example, given:
  - P1: m0, m1, m2
  - P2: m3, m4, m5
  - P3: m6, m7, m8
  - Cross-process happened-before: m0 → m4, m5 → m8
- Is this causal ordering?
  - P1: m0, m3, m6, m1, m4, m7, m2, m5, m8
  - P2: m0, m4, m1, m7, m3, m6, m2, m5, m8
  - P3: m0, m1, m2, m3, m4, m5, m6, m7, m8

Total Ordering

- Every process delivers all messages in the same order
- For example, given:
  - P1: m0, m1, m2
  - P2: m3, m4, m5
  - P3: m6, m7, m8
- Is this total ordering?
  - P1: m7, m1, m2, m4, m5, m3, m6, m0, m8
  - P2: m7, m1, m2, m4, m5, m3, m6, m0, m8
  - P3: m7, m1, m2, m4, m5, m3, m6, m0, m8
- What about this?
  - P1: m7, m1, m2, m4, m5, m3, m6, m0, m8
  - P2: m7, m2, m1, m4, m5, m3, m6, m0, m8
  - P3: m7, m1, m2, m4, m5, m3, m6, m0, m8

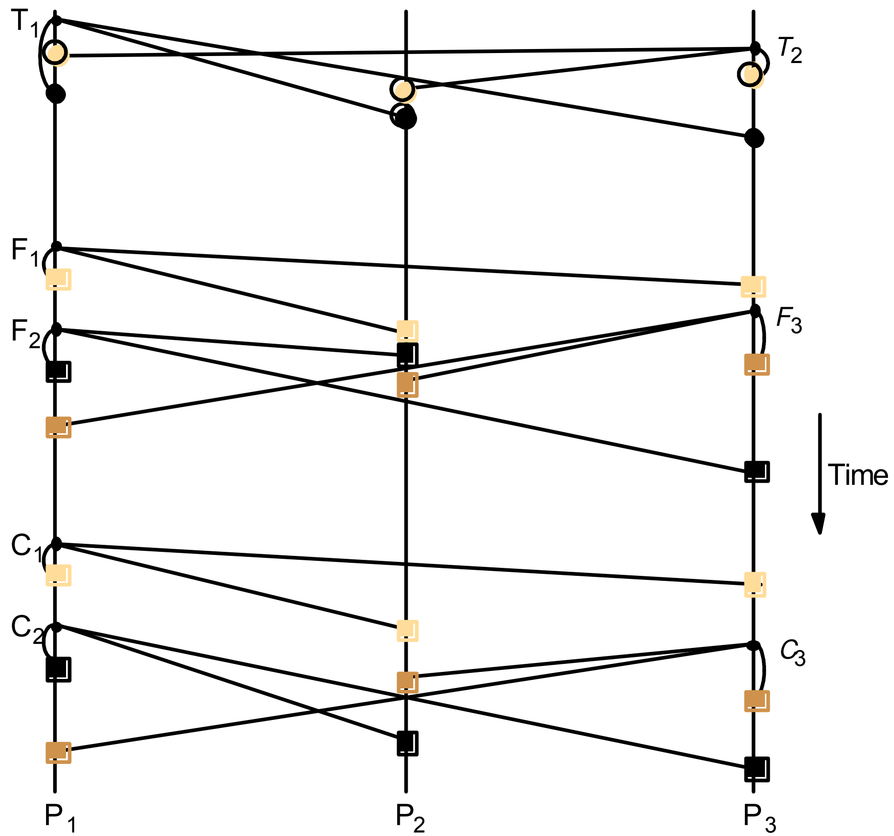
Ordered Multicast

- FIFO Ordering: If a correct process issues multicast(*g, m*) and then multicast(*g, m'*), then every correct process that delivers *m'* will have already delivered *m*.
- Causal Ordering: If multicast(*g, m*) → multicast(*g, m'*), then every correct process that delivers *m'* will have already delivered *m*.
  - Typically, → is defined over multicast communication only.
- Total Ordering: If any correct process delivers *m* before *m'*, then every correct process that delivers *m'* will have already delivered *m*.

Total, FIFO and Causal Ordering

- Totally ordered messages T_1 and T_2.
- FIFO-related messages F_1 to F_3.
- Causally related messages C_1 to C_3

- Total ordering does not imply causal ordering.
- Causal ordering implies FIFO ordering
- Causal ordering does not imply total ordering.
- Hybrid mode: causal-total ordering, FIFO-total ordering.



Display From Bulletin Board Program

Bulletin board:*os.interesting*		
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
end		

What is the most appropriate ordering for this application?

(a) FIFO (b) causal (c) total

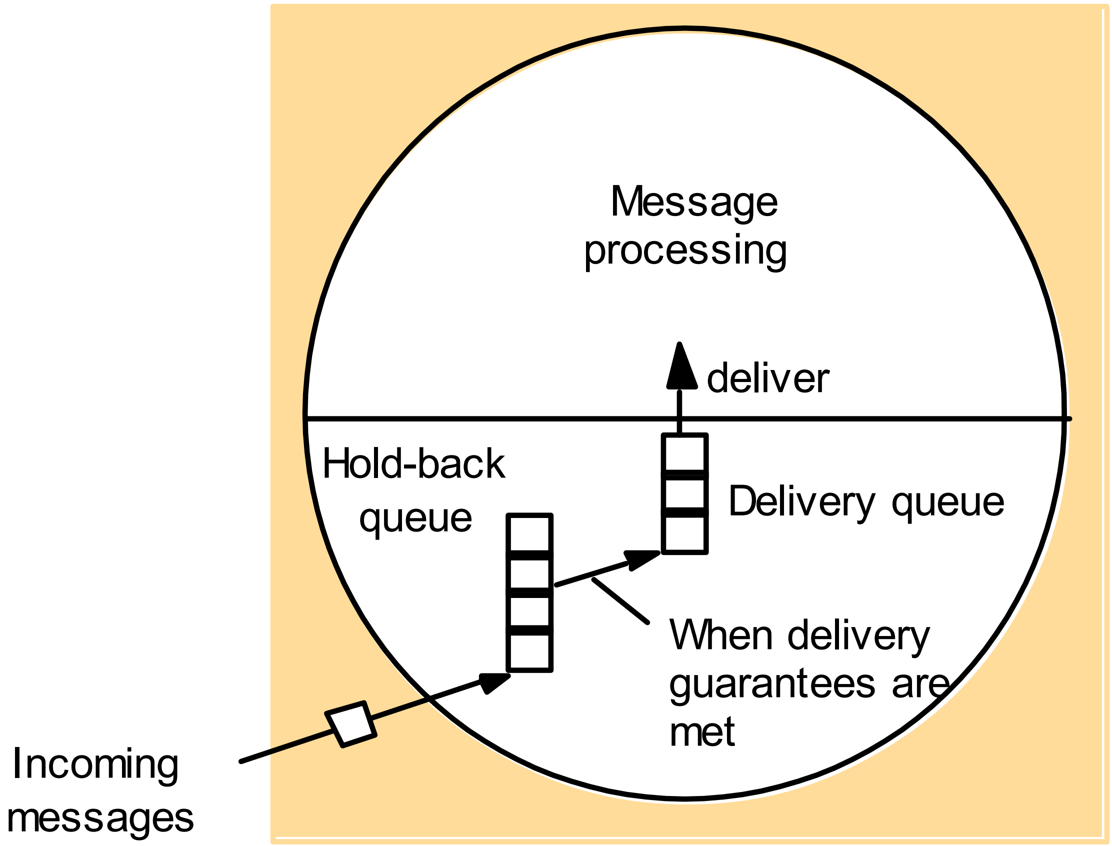
Providing Ordering Guarantees (FIFO)

- Look at messages from each process in the order they were sent:
  - Each process keeps a sequence number for each other process.
  - Every message carries its origin's sequence number.
  - When a message is received, if message # is:
    - » as expected (next sequence for that process), accept
    - » higher than expected, buffer in a queue
    - » lower than expected, reject
- Much like TCP sequence space processing!

Implementing FIFO Ordering

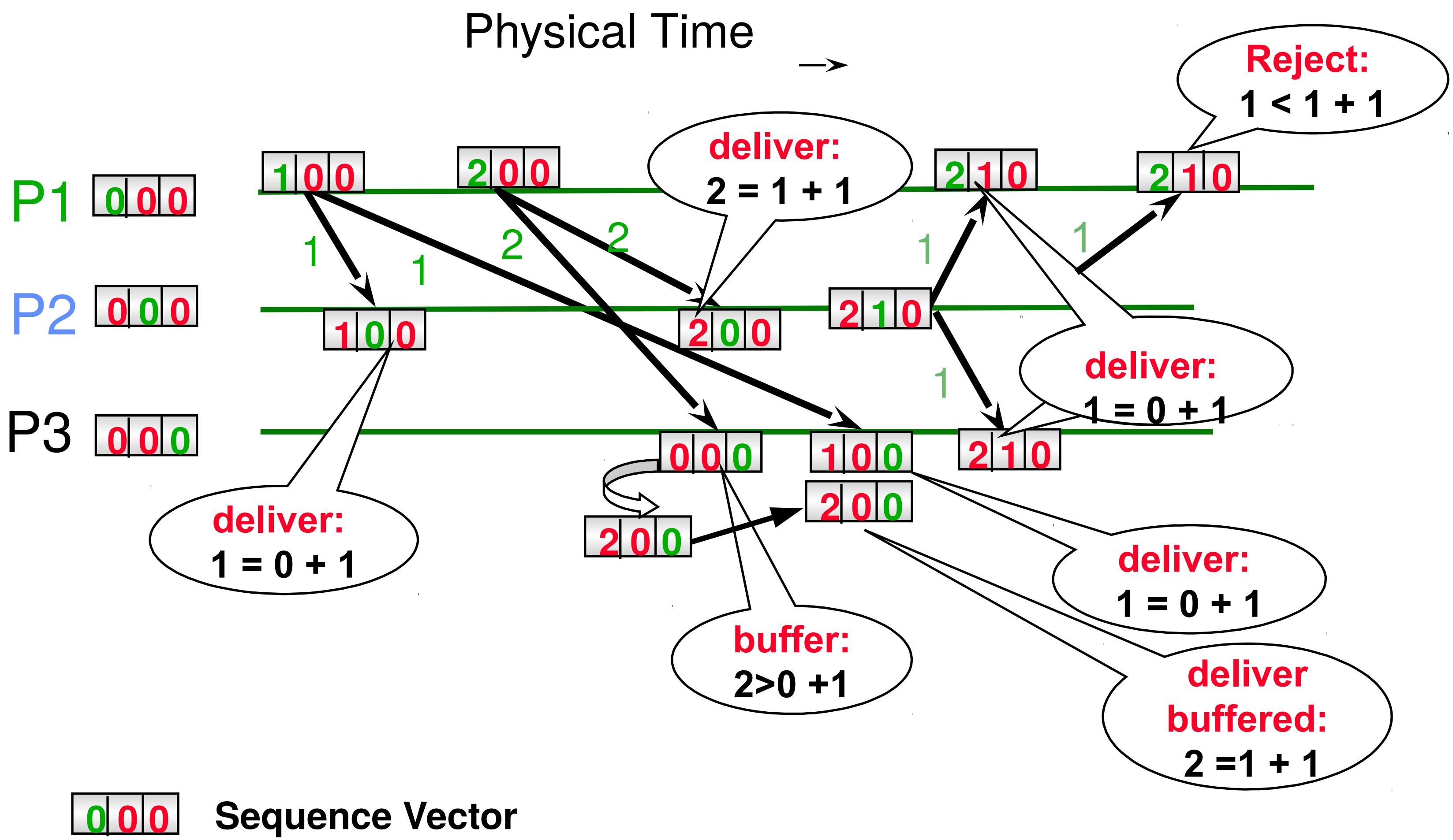
- At each process *p*:
  - S^p_g: the number of messages *p* has sent to group *g*.
  - R^q_g: the sequence number of the latest group-*g* message *p* has delivered from *q*.
- For *p* to FO-multicast *m* to *g*
  - *p* increments S^p_g by 1.
  - *p* "piggy-backs" the value S^p_g onto the message.
  - *p* B-multicasts *m* to *g*.
- At process *p*, upon receipt of *m* from *q* with sequence *S*:
  - *p* checks whether $S = R^q_g+1$. If so, *p* FO-delivers m and increments R^q_g
  - If $S > R^q_g+ 1$, *p* places the message in the hold-back queue until the intervening messages have been delivered and $S= R^q_g+1$.
  - If $S < R^q_g + 1$, *p* rejects *m*.

Hold-back Queue for Arrived Multicast Messages



Example: FIFO Multicast

(Not to be confused with vector timestamps!)



Summary

- Reliable multicast
  - Reliability
  - Ordering
  - R-multicast
- Ordered Multicast
  - FIFO ordering
  - Causal ordering
  - Total ordering
- Next time: more multicast!

References

- Textbook section 15.4. **Required Reading.**

Acknowledgements

- These slides created by Steve Ko, lightly modified and used with permission by Ethan Blanton
- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).