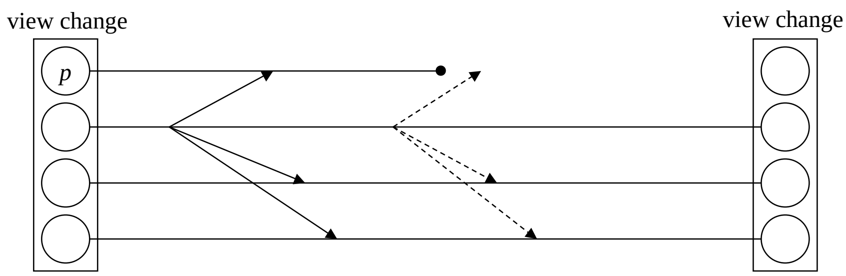
Virtual Synchrony

- "Send to all members or to none"
  - Who are the members, in particular in presence of failures?
- **Group view**: current list of members in the group.
  - Group view is consistent among all processes.
  - Members are added/deleted through **view changes**.
- **Virtually synchronous atomic multicast**:
  - 1. There is a unique group view in any consistent state on which all members of the group agree.
  - 2. If a message *m* is multicast in group view *v* before view change *c*, either no processor in *v* that executes *c* ever receives *m*, or every processor in *v* that executes *c* receives *m* before performing *c*.

Virtual Synchrony (2)

- Define *G* as set of messages multicast between any two consecutive view changes.
- All processors in a group view *v* that do not fail receive all messages in *G*.
- A processor *p* that fails may not receive all of *G*; but we know what *p* received; this simplifies recovery.



- View change managed by group membership protocol.

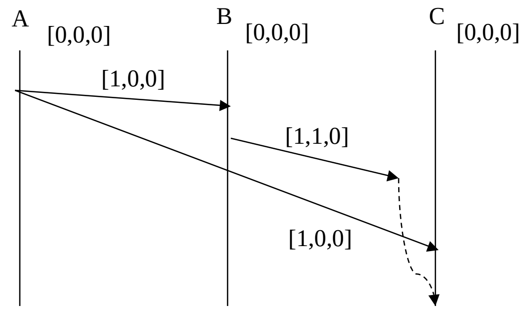
ISIS

http://simon.cs.cornell.edu/Info/Projects/ISIS

- Group communication toolkit
- Facilities:
  - Multicast
  - Group view maintenance
  - State transfer
- Synchrony
  - Closely synchronous
    - All common events are processes in same oreder (total and causal ordering)
  - Virtually synchronous
    - Failures are synch-ordered
- Multicast protocols:
  - FBCAST: unordered
  - CBCAST: causally ordered
  - ABCAST: totally ordered
  - GBCAST: sync-ordered
    - used for managing group membership

ISIS: CBCAST

- Group has *n* members
- Each member *i* maintains timestamp vector TS_i with *n* components.
- $TS_i[j]$ = timestamp of last message received by *i* from *j*.

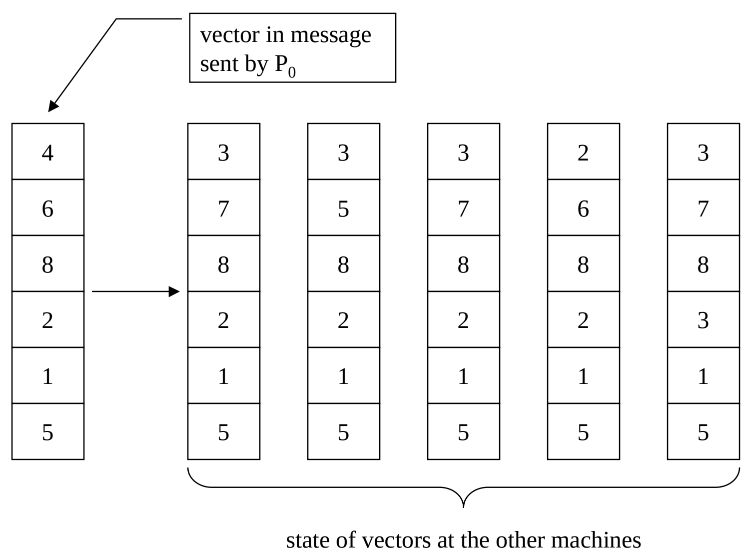


CBCAST (2)

```
mc_send(msg m, view v)
Pi: TSi[i] := TSi[i]+1
    send m to all members of view v
    send TSi[] as part of message m.
```

```
mc_receive(msg m)
Pi: let Pj be sender of m
    let tsj be timestamp vector in m
    check:
    1. tsj[j] = TSi[j]
       /* this is next message in sequence from Pj
          no messages have been missed. */
    2. for all k<>j: tsj[k] <= TSi[k]
       /* Sender has not seen a message that the
          receiver has missed. */
    If both tests passed, message is delivered, else
    it is buffered.
```

CBCAST: Example



Virtually Synchronous Group View Changes

- Virtual synchrony: all messages sent during a view v_i are guaranteed to be delivered to all operational members of v_i before ISIS delivers notification of v_{i+1}.
- Process *p* joins to produce group v_{i+1}:
  - no message of v_i is delivered to *p*
  - all messages sent by members of v_{i+1} after notification has been sent by ISIS will be delivered to *p*.
- Sender *s* fails in view v_i:
  - messages are stored at receivers until they are *group stable*.
  - if sender of non group stable message fails, holder of message is elected, and continues multicast.
- Some member *q* of v_i fails, producing v_{i+1}:
  - did *q* receive all messages in v_i?
  - did *q* send messages to other failed processes?

ABCAST: causally and totally ordered

Originally: form of 2PC protocol

1. Sender *S* assigns timestamp (sequence number) to message.
2. *S* sends message to all members.
3. Each receivers picks timestamp, larger than any other timestamp it has received or sent, and sends this to *S*.
4. When all acks arrived, *S* picks largest timestamp among them, and sends a commit message to all members, with the new timestamp.
5. Committed messages are sent in order of their timestamps.

Alternatives:

Sequencers

Interlude: Causally and Totally Ordered Communication: A Dissenting Voice

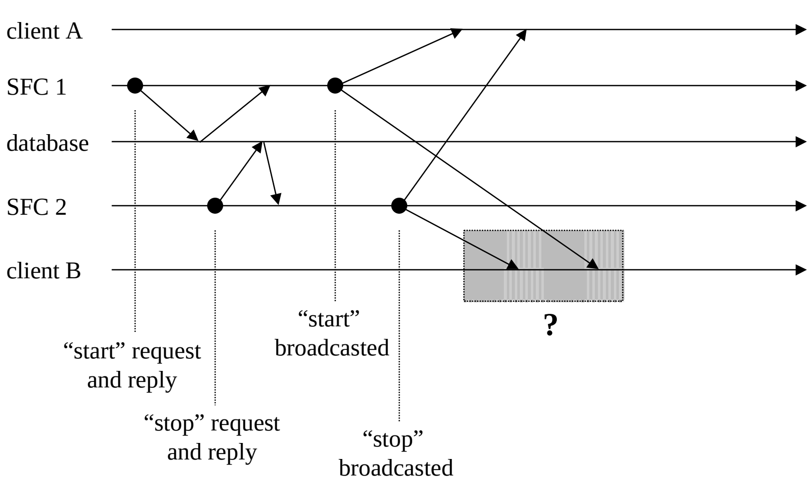
Reference: D. Cheriton and D. Skeen

"Understanding the Limitations of Causally and Totally Ordered Communication", *14th ACM Symposium on Operating Systems Principles*, 1993

- Unrecognized causality (can't say "for sure")
  - causal relationhips between messages at semantic level may not be recognizable by the *happens-before* relationship on messages.
- Lack of serialization ability (can't say "together")
  - cannot ensure serializable ordering between operations that correspond to groups of messages.
- Unexpressed semantic ordering constraints (can't say "whole story")
  - many semantic ordering constraints are not expressible in *happens-before* relationship
- No efficiency gain over state-level techniques (can't say efficiently)
  - not efficient, not scalable

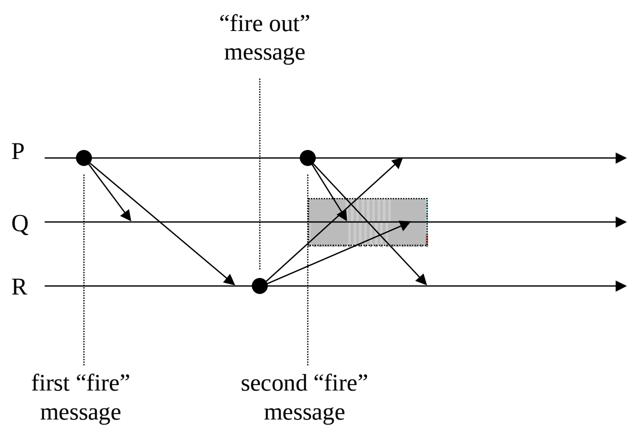
Interlude (2): Unrecognized Causality

Example 1: Shop Floor Control



Interlude (3): Unrecognized Causality

Example 2: Fire Control



Reliable Multicast Protocol

(B.Whetten,T.Montgomery,S.Kaplan.
"A High-Performance, Totally Ordered Multicast Protocol",
`ftp://research.ivv.nasa.gov/pub/doc/RMP/RMP_dagstuhl.ps`...)

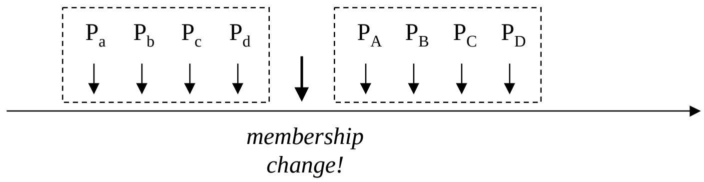
- Entities:
  - process:
    - sender/receiver of packets
  - group:
    - basic unit of group communication.
    - set of processes that receive messages sent to given IP Multicast address and port.
  - membership of a group can change over time
- Taxonomy:
  - Quality of Service
  - Synchrony
  - Fault-Tolerance

RMP: Quality of Service (QoS)

- Quality of Service related to semantics.
- <u>unreliable</u>
  - packet is received zero-or-more times at destination
  - no ordering
- <u>reliable</u>
  - packet is received at least once at each destination
- <u>source-ordered</u>
  - packet arrives exactly once at each destination
  - same order as sent from source
  - no ordering guarantee when more than one source
- <u>totally ordered</u>
  - serializes all packets to a group

RMP: Virtual Synchrony

- e.g. in ISIS (Birman *et al.*)
  - All sites see the same set of messages before and after a group membership change.



- Allows distributed applications to execute as if communication was synchronous when it actually is asynchronous.

RMP: Fault-Tolerance

- node failures, network partitions
- atomic delivery within partition:
  - If one member of the group in a partition delivers packet (to application), all members in that partition will deliver packet if they were in the group when the packet was sent.
  - No guarantee about delivery or ordering between partitions.
- *K*-resilient atomicity:
  - Totally ordered
  - Delivery is atomic at all sites that do not fail or partition, provided that no more than *K* sites fail or partition at once.
  - with $K=floor(N/2)+1$ atomicity guaranteed for any number of failures.

RMP: Fault-Tolerance (cont)

- majority resilience:
  - If two members deliver any two messages, they agree on ordering of messages.
  - Guarantees total ordering across partitions, but not atomicity.
- total resilience (safe delivery):
  - Sender knows that all members received it before it can be delivered.
  - One or more sites can fail *before delivering the packet.*

Algorithms in RMP

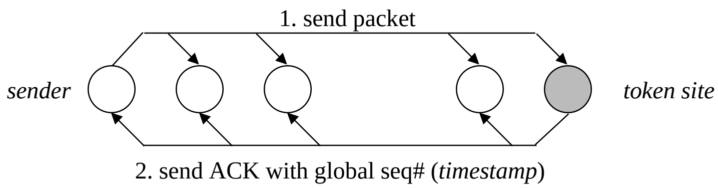
- Basic delivery algorithm
  - handles delivery of packets to members
- Membership change algorithm
  - handles membership change requests, updates view at members.
- Reformation algorithm
  - reconfigures group after failure, synchronizes members
- Multi-RPC algorithm
  - allows non-members to sent to group
- Flow control and congestion control
  - similar to Van Jacobson TCP congestion control algorithm

ACKs in Reliable Multicast

- **Def**: Packet becomes stable: Sender knows that all destinations have received packet.
- positive ACKs:
  - quick stability
  - scalability?
- cumulative ACKs:
  - parameter: number of packets per ACK
  - load *vs.* length of time for packet to go stable
- negative ACKs:
  - burden of error detection shifts to destination
  - sequence numbers
  - time to go stable unbounded
  - lost packet only detected after another packet is received.

Basic Delivery Algorithm

- NACKs for reliable delivery, ACKs for total ordering and stability.
- packet ID: *{RMP proc ID, seq # of proc, QoS level}*



- Functions of ACK:
  - positive acknowledgment to sender ("token site has received packet")
  - timestamp as global basis for detection of dropped packets.
- Q: When does packet become stable?

Reaching Stability

- While sending ACK, token site forwards token to next process in group:
  - Before accepting token, member is required to have all packets with timestamps less than in ACK.
  - If site in group with *N* members receives token, it knows that all packets with *TS >= currTS-N* have been received by all members.

Basic Delivery Algorithm

- Each site has
  - *DataList*: contains Data packets that are not yet ordered
  - *OrderingQ*: contains slots:
    - pointer to packet
    - delivery status (missing, requested, received, delivered)
    - timestamp
- Data packet arrives: placed in *DataList*
- ACK arrives: placed in *OrderingQ*, creating one or more slots at end of queue if necessary
- Data packet or ACK arrives:
  - scan *OrderingQ*: match Data packets in *DataList* with slots that have been created by an ACK.
  - when match is found, Data packet is transferred to slot.
  - when whole occurs in *OrderingQ*, send out NACK, requesting for retransmission of packet.

A Cool Homepage on Multicast Protocols:

`http://hill.lut.ac.uk/DS-Archive/MTP.html`