Broadcasts

- ◆ Several types of Broadcast
  - Broadcast = (1-to-all)
  - Multicast = (1-to-many)
  - Unicast = (1-to-1)
- ◆ Properties of interest are
  - reliability
  - consistent ordering
  - preservation of causality
- ◆ Broadcast primitives
  - Reliable broadcast
  - Atomic broadcast
  - Causal broadcast

Reliable Broadcast

- ◆ The message to be broadcast should be received by all non-faulty processes
- ◆ A protocol for reliable broadcasting using *Message Forwarding*
  - Originally proposed in:
    - » Schneider, Gries and Schlichting, *Fault-tolerant Broadcasts*, Science of Computer Programming, 4:1-15, 1984.

Reliable Broadcast

◆ Reliable Broadcast using Message Forwarding

    – Considers network as a tree. Tree is logical structure with no direct relationship to the network topology. An edge from node p to q indicates that during broadcast p will forward the message to q.

    – Original sender (initiator) is root of tree

    – Receiving node i forwards message to all of its children, (which in turn forward).

    – Children in turn send acknowledge to i

    – If child j does not acknowledge, node i takes over and forwards message to j's children.

    – Protocol works great, except when root fails. If root fails after partial broadcast, some node that already received the message has to finish the broadcast.

        – => node i has to monitor root

        – => implemented as root sending completion-message

Reliable Broadcast

◆ *Trans Protocol:* Piggybacking Acknowledgment

    – based on article [Melliar-Smith-1990] by Melliar-Smith, Moser and Agrawala, "Broadcast Protocols for Distributed Systems", IEEE Trans. Parallel and Dist. Systems, 1(1):17-25, January 1990.

    – Assumption: when broadcasting some nodes will receive the message and some will miss it.

        » e.g. Ethernet

        » e.g. unreliable broadcast protocol in point-to-point network

    – the protocol described builds a reliable broadcast primitive from the unreliable broadcast primitive which it assumes is available to it.

Reliable Broadcast

- Trans protocol
  - » attach (piggyback) positive and negative acknowledgements on a broadcast message.
  - » each broadcast message carries ID of sending node and unique sequence number for the message
  - » Outline of protocol given three processors P, Q and R
    - ▪ P broadcasts a message.
    - ▪ The message from P is received uncorrupted by Q.
    - ▪ Q includes a positive acknowledgment for P's message in its next message.
    - ▪ R upon receiving Q's message is aware that P's message has been acknowledged and that there is no need to also acknowledge it in its next message; instead R acknowledges Q's message.
    - ▪ If R has not received the message from P, the message from Q alerts R of this loss and, therefore, R includes a negative acknowledgment for P's message in R's next message.

Reliable Broadcast

- ◆ Example (see Melliar-Smith-1990 - page 20)
  - assume capital letters are messages
  - lower case letters are the respective positive acknowledge, overhead bars denote negative acknowledgement
  - One message gets lost. Which one and where?

  - sequence:

$$A - Ba - Cb - Dc - E\bar{c}d - Cb - Fec$$

Reliable Broadcast

◆ Example (see Melliar-Smith-1990 - page 20)
  – assume that the processor broadcasting message *E* received neither message *C* nor *B*.

  – sequence:

$$A - Ba - Cb - Dc - E\bar{c}d - Cb - F\bar{b}ec - Ba - Gfb$$

Reliable Broadcast

◆ Example (see Melliar-Smith-1990 - page 20)
  – now assume that messages are not processed instantaneously
  – in the next example the assumption is made that no message is acknowledged by the next broadcast message

  – sequence:

$$A - B - Ca - Dab - Ebc - Fcd - G\bar{c}de - Hef - Ca - Igh - Jghc$$

Atomic Broadcast

◆ Atomic broadcast is a 1-to-many communication paradigm that is stricter than reliable broadcast
   – reliable broadcast made sure that all non-faulty nodes get the message
   – atomic broadcast adds the requirement that all messages need to be <u>received in the same order</u>.
   – discussion is based on [Birman-1987]:
      » K. Birman, and T. Joseph, *Reliable Communication in the Presence of Failures*, ACM Trans. on Computer Systems, 5(1):47-76, Feb. 1987.

Atomic Broadcast

◆ Three-Phase Protocol
   – Assign priorities to messages and deliver the messages in the order of priority
   – But how do the processors agree on what the next priority is?
      » made even more difficult if communication delays differ
   – Thus, first all nodes must explicitly agree on a priority of a message and then assigning non-conflicting priorities to later messages
   – Protocol for <u>assigning priorities</u> works in three rounds of exchange
   – Each message in a processor's buffer is tagged *deliverable* or *undeliverable*
   – Assume there is a primitive called *abcast(m,p)*
   » *m* is the message
   » *p* is the (integer) priority assigned to the message by the node broadcasting *m*

Atomic Broadcast

ABCAST(msg, label, dests) implementation (Birman-1987, pg62)

(1) The sender transmits *msg* to its destinations.

(2) Each recipient adds the message to the priority queue associated with label, tagging it as undeliverable. It assigns this message a priority larger than the priority of any message that was placed in the queue, with the process ID of the recipient as a suffix. It then informs the sender of the priority that it assigned to the message.

(3) The sender collects responses from recipients that remain operational. It then computes the maximum value of all the priorities it received, and sends this value back to the recipients.

(4) The recipients change the priority of the message to the value they receive from the sender, tag the message as deliverable, and re-sort their priority queues. They then transfer messages from the priority queue to the delivery queue in order of increasing priority, until the priority queue becomes empty or the message with the lowest priority is undeliverable. In the latter case no more messages are transferred until the message at the head of the queue becomes deliverable.

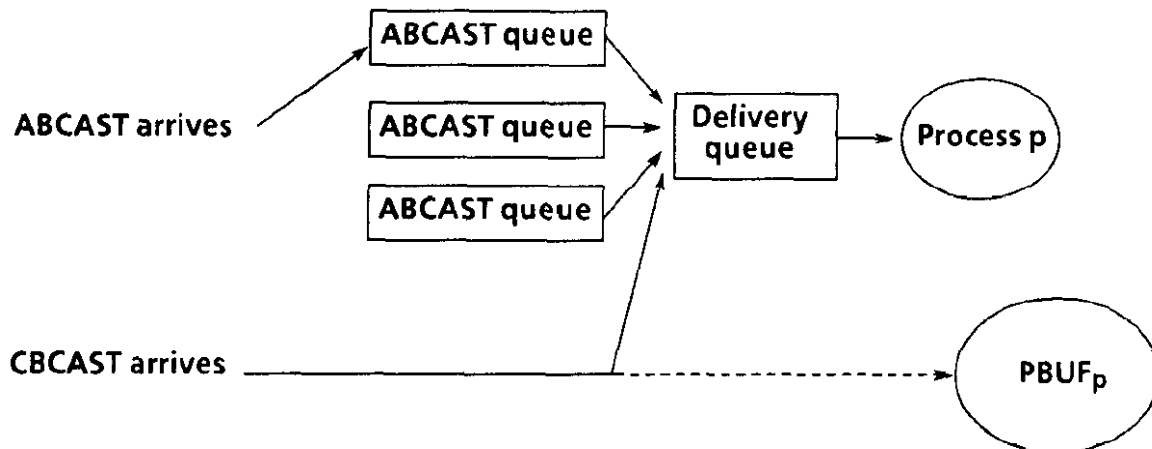# *Atomic Broadcast*

(from Birman-1987, pg62)

Fig. 3. Data structures used by ABCAST and CBCAST primitives.

Atomic Broadcast

- ◆ Three-Phase Protocol
  - **Phase I**: Sender transmits message (*m,p*) to all nodes
  - **Phase II**: Each receiver adds the message into their local queue
    - message is tagged *undeliverable*
    - assign this message a priority higher than the priority of any message that was placed in the buffer, *p = max(priorities in queue) +1*
    - send new *p* back to sender
  - **Phase III**:
    - Sender
      - collects priorities
      - sets *p = max(priorities returned)*
      - sends *p* back to receivers
    - Receiver
      - assigns new *p* to message, tag message as *deliverable*
      - re-sort the priority queue
      - transfer messages from queue head

Causal Broadcast

- ◆ In atomic broadcast messages broadcast from different nodes needed to be received by all nodes in the same order, however, there was no preference to the order as long as all processors received them in the same order.
- ◆ Causal broadcast addresses the case where the order of the broadcasts matter
  - e.g. consider requests in distributed databases
    - » should deposit money first, the issue a withdraw
    - » if this order is not preserved it could be that overdraft charges are applied.

Causal Broadcast

◆ Causal broadcast flavors
- causal broadcast without total ordering
  - » assume that messages m and m' such that $m \rightarrow m'$
  - » then m is delivered before m' at all nodes
  - » however, if there is no causal relationship between m and m'
    - ▪ then m and m' can be delivered in any order and
    - ▪ this order may not be the same at all nodes
- causal broadcast with total ordering
  - » requires that messages are delivered to different nodes in the same order such that the order preserves the causality between messages.