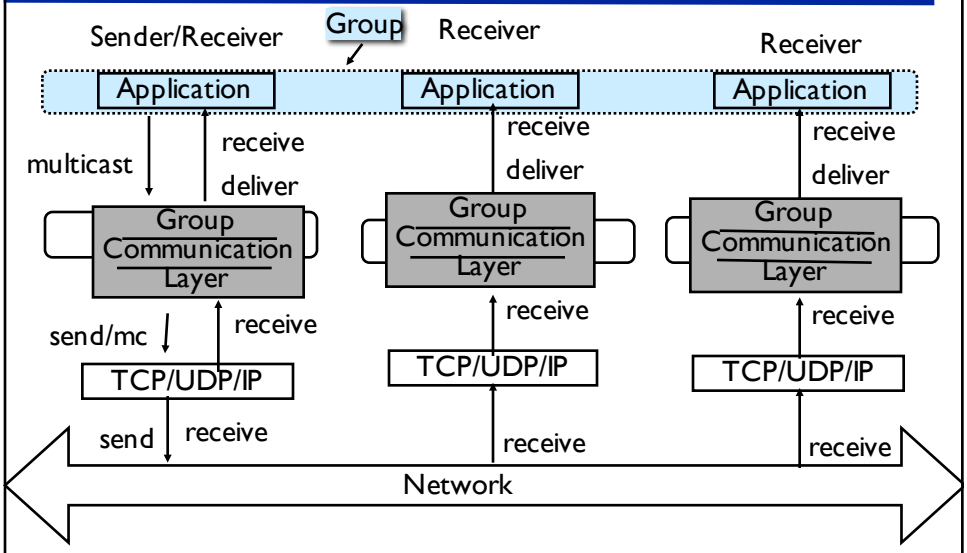
Group Communication Systems

Group Concept

- A group is a collection of processes that cooperate
  - to provide a service
  - run a distributed execution
  - ….
- Services
  - Group Maintenance
  - Multicast

Architecture



Basics

- Services
  - ☆ Multicast to all processes in the group:
    - ordering
      - ▲ messages are delivered in a special order
        - FIFO, causal, total
    - reliability
      - ▲ messages are guaranteed to be delivered
    - interface:
      - ▲ multicast(group-name, message)
      - ▲ receive(group-name, message)
  - ☆ Group maintenance
    - create / destroy groups
      - ▲ create(group-name)
      - ▲ destroy(group-name)
    - processes can leave/join
      - ▲ join(group-name)
      - ▲ leave(group-name)
    - automatic removal of failed nodes
      - ▲ (internal failure detection)
    - application is informed whenever group composition changes
      - ▲ deliver group-change-event(new-config)

Users of GCS

- Highly available servers (process replication)
- Conferencing
- Database replication
- Cluster management

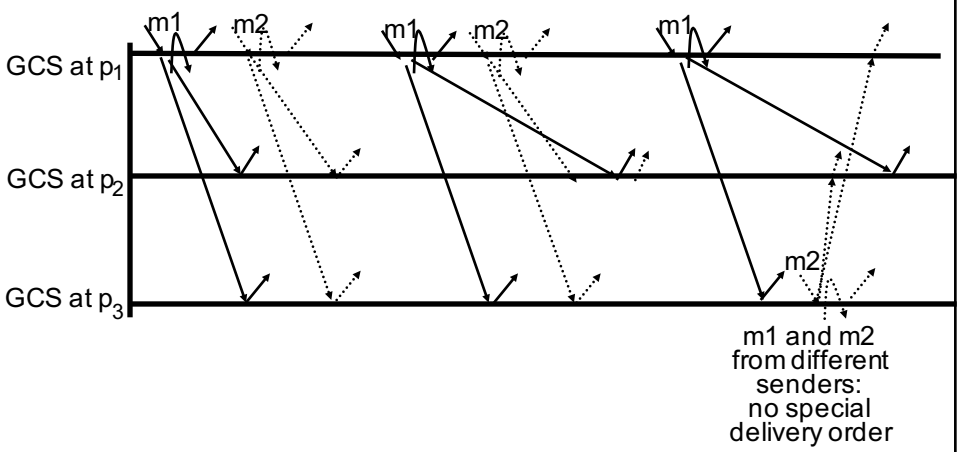
Multicast Communication

- System model for now:
  - A group g consists of member processes $p_1, \ldots p_n$
  - no failures
  - underlying communication protocol provides reliable point-to-point send/receive
- Layers:
  - application layer: AL
  - gcs layer: GCSL
  - process: AL and GCSL
    - from context it becomes clear which layer is referred to
  - communication layer: CL
- Interface of GCSL provided to AL
  - multicast(g, message)
  - receive(g,sender,message)
    - internally the GCSL has deliver(g,sender,message)

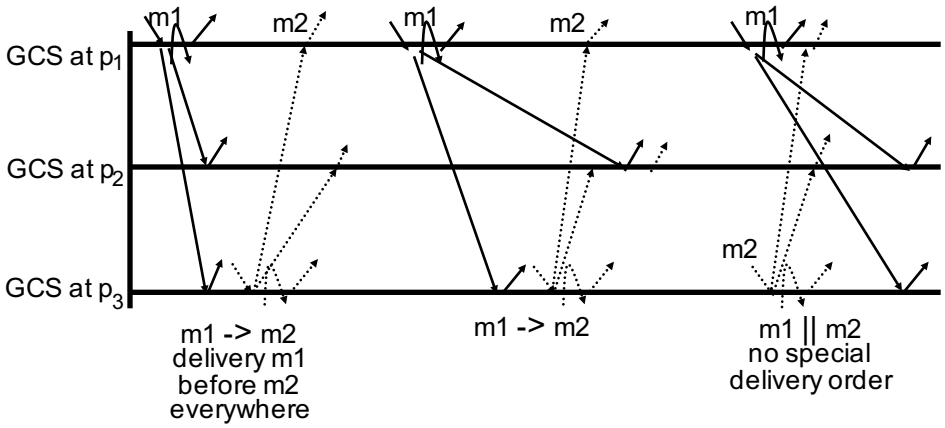
Ordering Semantics

- No order
- FIFO order
  - If the AL of process p_i has called *multicast(g,m)* and then *multicast(g,m')*, then the GCSL delivers *m* before *m'* at all member processes of *g*
- Causal Order
  - if multicast(g,m) -> multicast (g,m'), then the GCSL delivers *m* before *m'* at all member processes of *g*.
  - -> is the happened before relation induced only by message sent between the members of the group
  - note that the two multicast can be initiated by different processes
- Total order
  - for any two messages m and m', if the GCSL delivers *m* before *m'* at any member process, then it delivers *m* before *m'* at all member processes

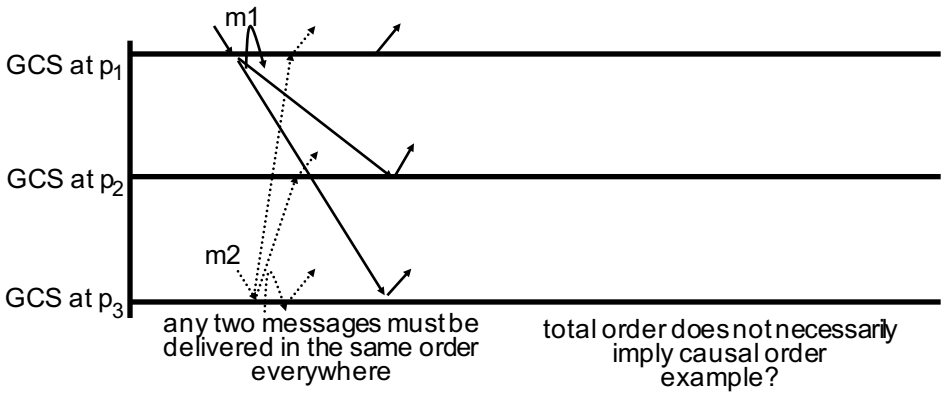
Examples FIFO



Examples CAUSAL



Example total order



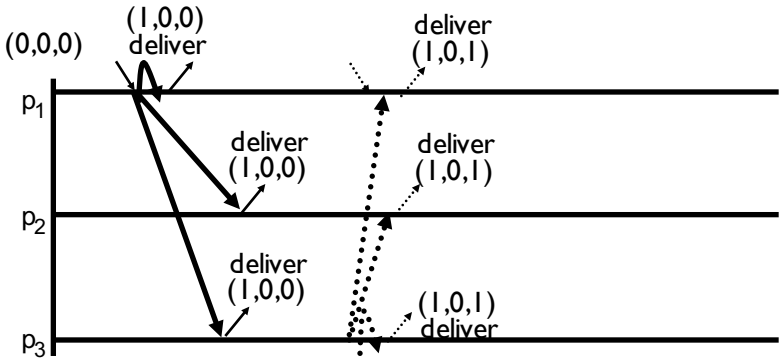
Simple Multicast

- multicast/deliver algorithm for GCS layer of process p_i (i = 1, 2, …. N)
- Assumption: Underlying network provides send/receive pair (without ordering)
- Upon multicast(g,m) from AL
  - for all p_j of g: send(p_j, <g,m>)
- Upon receiving <g,*m*> from p_j from underlying network
  - deliver(g,p_j,m)

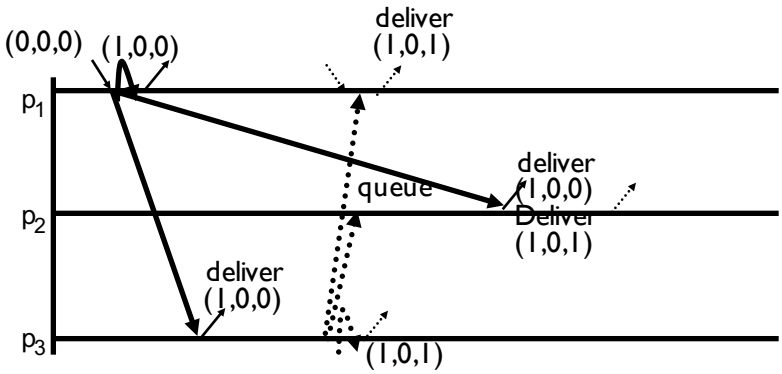
Causal Ordering using vector timestamps

- CO-multicast/deliver algorithm for GCS layer of process p_i (i = 1, 2, …. N)
- Assumption: existing multicast layer
- On initialization for each group g
  - $V_i[j] := 0$ (j = 1, 2, … N)
    - each group has own vector clock
- Upon CO-multicast(g,m) from AL
  - $V_i[i] := V_i[i] + 1$;
    - (NOTE: this is the only time the local timestamp is increased; other actions in the system do not increase the timestamp!)
  - multicast(g, <V_i, m>)
  - CO-deliver(g,p_i, m) immediately to own AL
- Upon receiving (g, p_j , <V_j, m>) from p_j from simple multicast layer
  - If j == i then ignore;
  - Place (p_j, <V_j,m>) in queue;
  - Wait until $V_j[j] == V_i[j] + 1$ and $V_j[k] <= V_i[k]$ ($k \neq j$) for vector clock of g;
  - Remove m from the queue and CO-deliver(g,p_j,m)
  - $V_i[j] := V_i[j] + 1$;

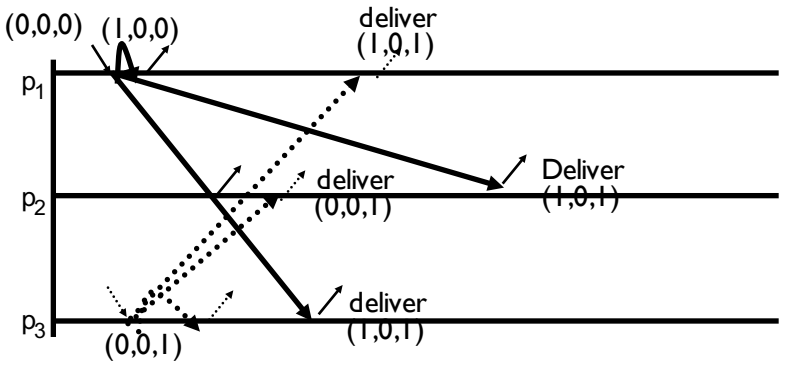
Example Algorithm I



Example Algorithm II



Example Algorithm III



Total Order with master process

- At the GCS layer
  - A specified process is the master process
  - Every process multicasts a message to all processes (using CO-multicast)
  - When a non-master receives a message m, it queues the message
  - When the master receives a message it delivers it according to CO-rules
  - Then it multicasts a "sequencer" message, indicating the order of the message in the stream of messages
  - When a non-master receives a "sequencer" message it delivers the corresponding message in sequence order

Total Order with sequencer process

- At the GCS layer
  - A specified process is the sequencer process
  - A non-sequencer process sends a message m only to the sequencer
  - The sequencer multicasts its own messages and forwards the messages from others in FIFO order

Token protocol

- At the GCS Layer
  - A token circulates among the member processes
  - Whenever a process has the token it is allowed to multicast messages.
    - Messages are tagged with sequence numbers
    - Receivers deliver messages in order of their sequence numbers
  - After a time period the token is forwarded to the next process
    - Token contains a field with the sequence number of last message multicast
    - Next process uses this sequence number + 1 for its first message to be multicast

Token Protocol

- Messages
  - Token t
    - Seq: sequence number of the last message sent (high-water-mark)
      - Initialized to 0
  - Message m
    - Seq: sequence number of message
- Data structures at process GCSL of p_i
  - Del_i: sequence number of last delivered message
    - Initialized to 0
  - NQ_i: queue of messages to be multicast (received from AL but not yet multicast)
  - RQ_i: queue of messages received waiting to be delivered; ordered by Seq of messages
- Only considers now one group g
  - separate data structures exist for each group

Token Protocol 2

- Upon TO-multicast(g,m) from AL
  - Put (g,m) in NQ_i
- Upon receive token t by process p_i
  - While NQ_i not empty
    - Get next message of NQ_i
    - t.seq = t.seq+1
    - m.seq = t.seq
    - Multicast(g,m)
  - send token t to next process p_j
- Upon receive(g,p_j,m) from p_j from simple multicast layer with m.seq=s at process p_i
  - Insert (p_j,m) into RQ_i according to sequence number
  - If (Del_i + 1) = s
    - TO-deliver(g,p_j,m) and remove from RQ_i
    - Del_i++
    - While RQ_i not empty
      - Look at first message (p_k,m') with m'.seq=t
      - If (Del_i+1) = t
        - TO-deliver(g,p_k,m') and remove from RQ_i
        - Del_i++
      - Else break;

Group Maintenance

- Interface:
  - Create/destroy process groups
  - Add or withdraw a process to or from a group
- Notify members of group membership changes (due to add/withdraw/crash)
  - deliver(g,new-config)
- Challenge: Failures
  - detect failures
  - reconfigure group
  - coordinate with message delivery
    - reliability of delivery

Failure Assumptions (for the purpose of this lecture)

- Each pair of processes is connected by reliable channels
  - *eventually* delivers a message to recipient's buffer
  - i.e., network partitions etc. eventually heal
- processes only fail by crashing
- correct process:
  - exhibits no failures at any point in the execution under consideration
- process that suffers failure
  - *non-failed* before failure, *crashed* afterwards (never considered a correct process)

Failure Detector

- Unreliable failure detector
  - takes as input identifier of p
  - returns either "suspected" or "unsuspected"
  - implementation: if failure detector has received message from p within last x time-units, then unsuspected, otherwise suspected
  - imperfect: force falsely suspected processes to simulate crash

Group Membership Problem

- Agreement on the membership of a group
  - Consistent system-wide view of the operational members in the presence of process join/leave/failure and communication failure
- View Delivery: when a membership change occurs, inform the application about new view
  - Local view: list of current members in the group reported to the application
  - a local view reported to any member is reported to all other members unless new changes take place (reliable view delivery!)
- Before delivery of view change
  - run view change protocol among all correct processes
  - all have to agree on new view

View Change

- View change protocol is an agreement protocol
- For a given group g with current configuration "current-view"
  - When S_i suspects S_j of current-view to have crashed
    - Send new-view-proposal(current-view,new-view) to all sites in new-view
  - Upon receiving new-view-proposal from S_i
    - If not agree (can communicate with S_j or have already committed to other new-view-proposal), send Nack
    - Else send ok to S_i
  - Upon S_i receiving ok from all sites in new-view
    - Send view-change(new-view) to all sites in new-view
  - Upon receiving view-change(new-view)
    - deliver(g,new-view)