Computer Science 425
Distributed Systems

CS 425 / ECE 428

Fall 2013

Indranil Gupta (Indy)

September 17, 2013

Lecture 7

Multicast

Reading: Sections 15.4

Communication Modes in Distributed System

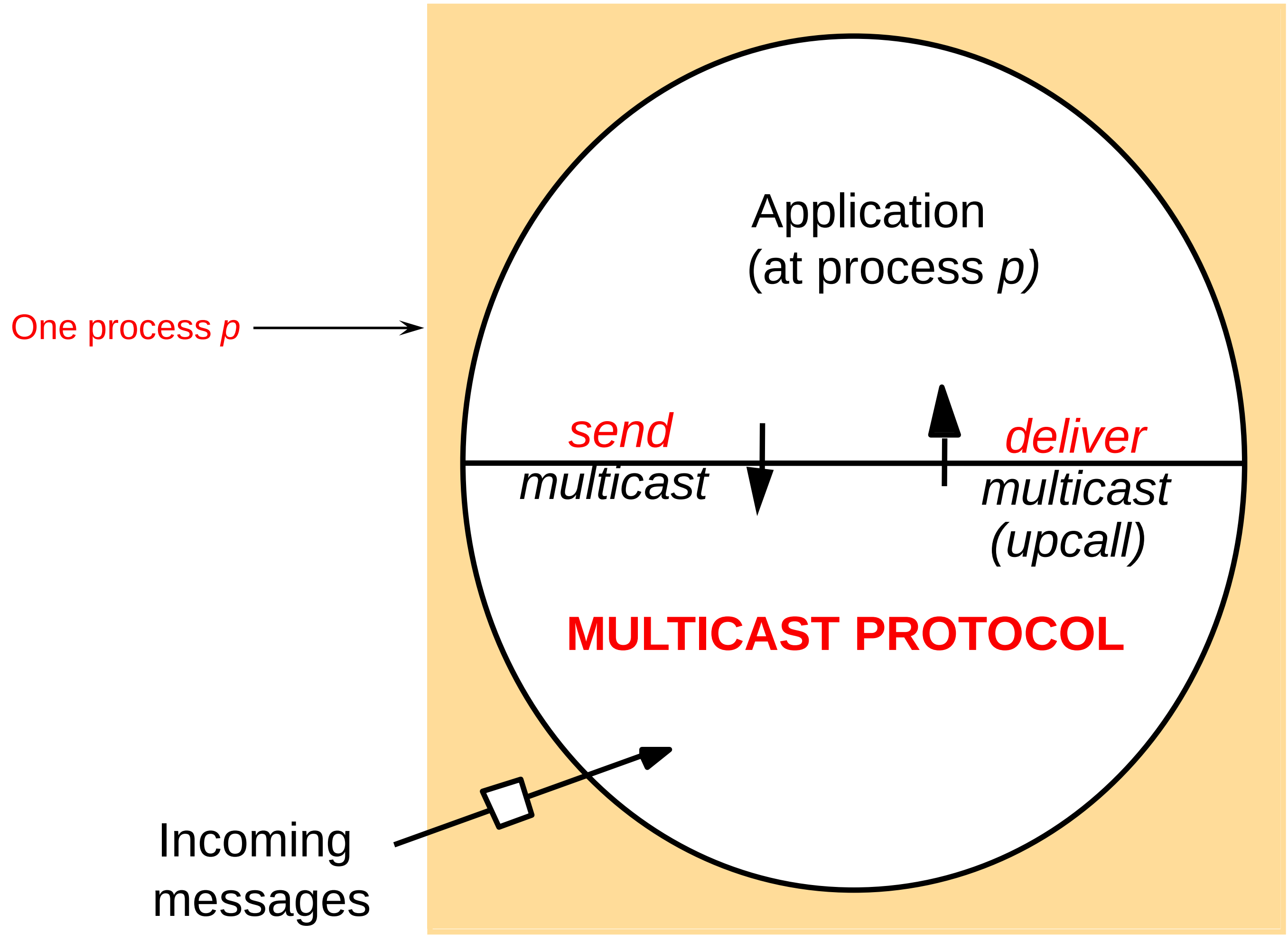
- **Unicast (*best effort* or *reliable*)**
  - **Messages are sent from exactly <u>one</u> process to <u>one</u> process.**
  - ***Best effort*: if a message is delivered it would be intact; no reliability guarantees.**
  - ***Reliable:* guarantees delivery of messages.**
- **Broadcast**
  - **Messages are sent from exactly <u>one</u> process to <u>all</u> processes on the network.**
  - **Broadcast protocols are not practical.**
- **Multicast**
  - **Messages broadcast within a <u>group</u> of processes.**
  - **A multicast message is sent from any <u>one</u> process to the group of processes on the network.**
  - **Reliable multicast can be implemented "above" (i.e., "using") a reliable unicast.**
    - **This lecture!**


NYSE
NYSE
NYSE
NYSE
NYSE

Other Examples of Multicast Use

- **Akamai's Configuration Management System (called ACMS) uses a core group of 3-5 servers. These servers continuously multicast to each other the latest updates. They use *reliable multicast*. 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: orders by one ATC need to be *ordered* (and reliable) multicast out to other ATC's.**
- **Newsgroup servers multicast to each other in a reliable and ordered manner.**
- **Facebook servers multicast your updates to each other**

What're we designing in this class



Basic Multicast (B-multicast)

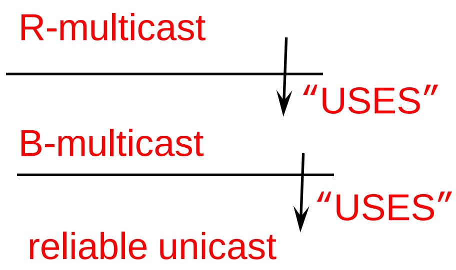
- ***Let's assume the all processes know the group membership***
- ***A straightforward way to implement B-multicast is to use a reliable one-to-one send (unicast) operation:***
  - ***B-multicast(group g, message m):***

    for each process *p* in *g*, *send (p,m).*
  - **receive(m): *B-deliver(m)* at p.**
- **A "correct" process= a "non-faulty" process**
- **A basic multicast primitive guarantees a correct process will eventually deliver the message, as long as the sender (multicasting process) does not crash.**
  - ***Can we provide reliability even when the sender crashes (after it has sent the multicast)?***

Reliable Multicast

- ***Integrity***: **A *correct* (i.e., non-faulty) process *p* *delivers* a message m at most once.**
- ***Validity***: **If a correct process multicasts (sends) message *m*, then it will eventually deliver *m* itself.**
  - **Guarantees liveness to the sender.**
- ***Agreement***: **If some one correct process delivers message *m*, then all other correct processes in *group(m)* will eventually deliver *m*.**
  - **Property of "all or nothing."**
  - **Validity and agreement together ensure overall liveness: if some correct process multicasts a message m, then, all correct processes deliver m too.**

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 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

Reliable Multicast Algorithm (R-multicast)

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

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

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

if <u>some</u> correct process B-multicasts a message m, then, all correct processes R-deliver m too. If no correct process B-multicasts m, then no correct processes R-deliver m.

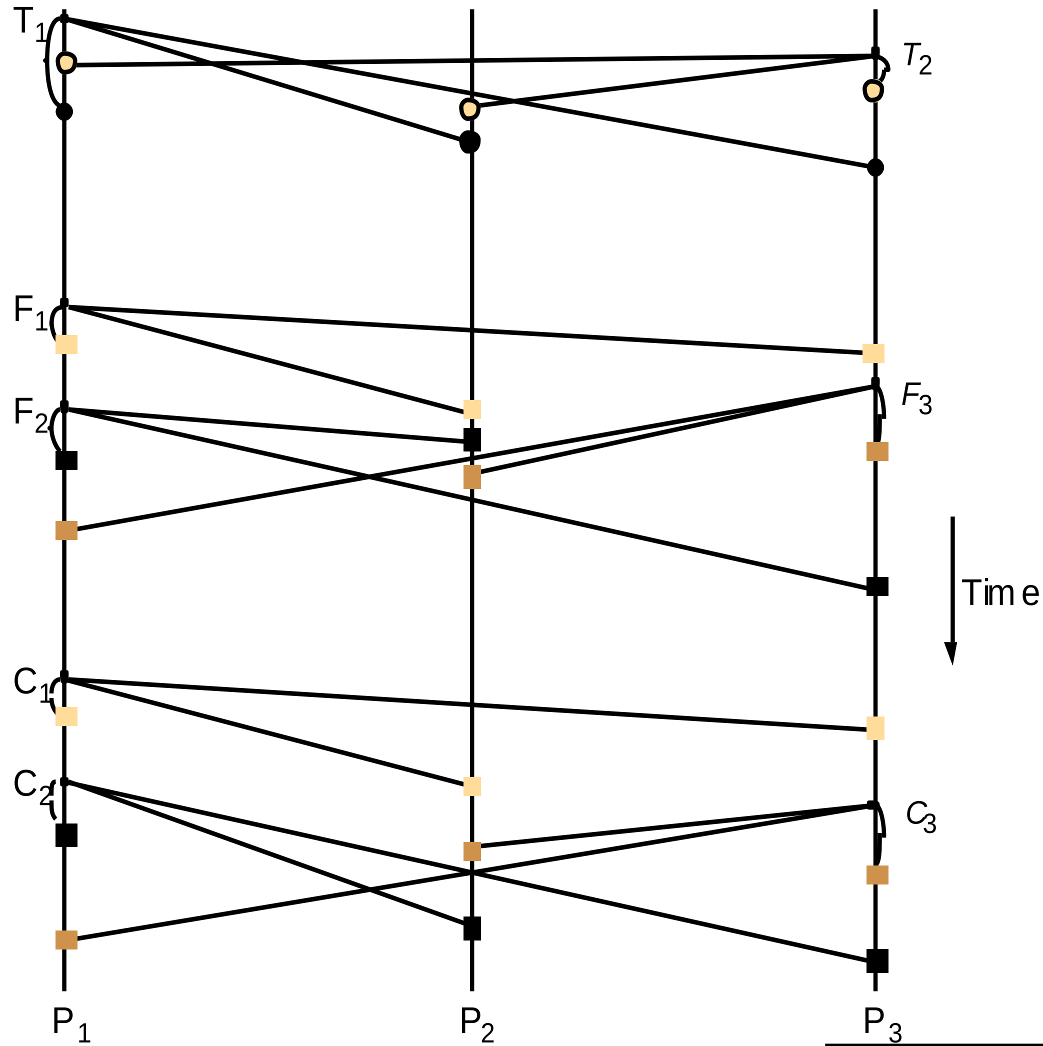
What about Multicast Ordering?

- ***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 any correct process that delivers *m′* will have already delivered m.**
- ***Total ordering***: **If a correct process delivers message *m* before *m′* (independent of the senders), then any other 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 and F_2.
- Causally related messages C_1 and C_3

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



Display From Newsgroup

Newsgroup: *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

What is the most appropriate ordering for Facebook posts?

Providing Ordering Guarantees (FIFO)

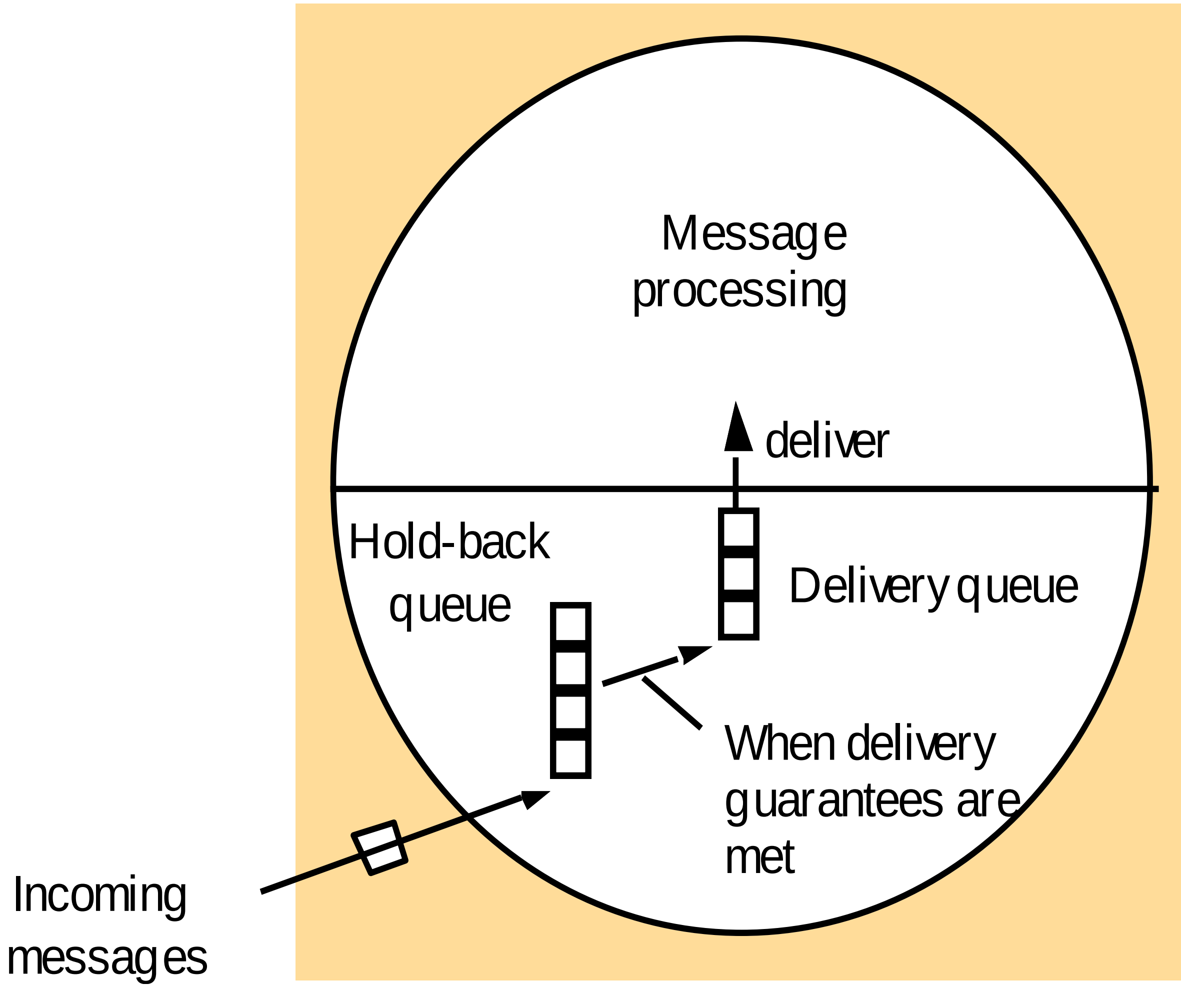
- **Look at messages from each process in the order they were sent:**
  - **Each process keeps a sequence number for each other process (vector)**
  - **When a message is received,**

If Message# is
- **as expected (next sequence), accept**
- **higher than expected, buffer in a queue**
- **lower than expected, reject**

Implementing FIFO Ordering

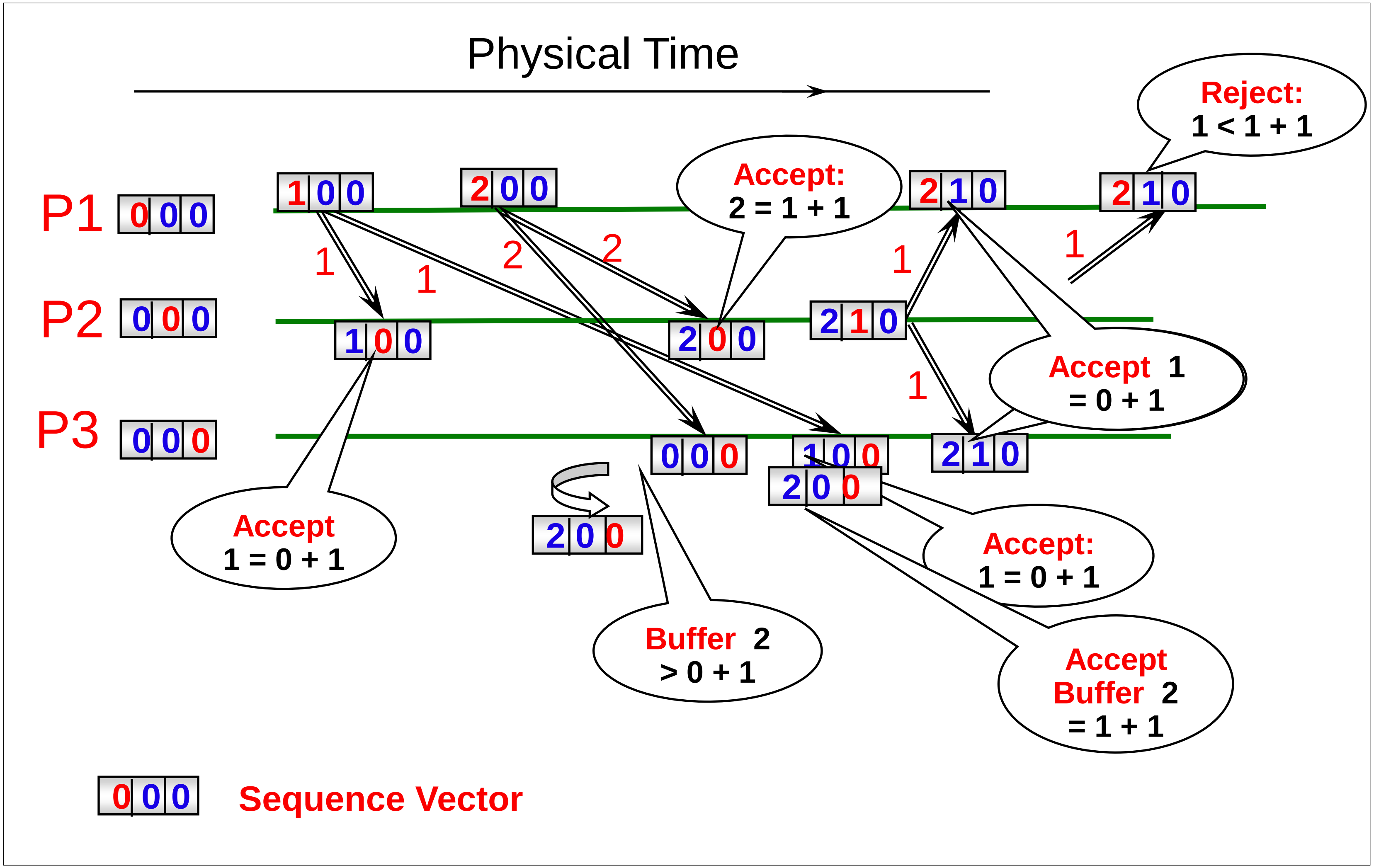
- S^p_g: the number of messages *p* has sent to *g*.
- R^q_g: the sequence number of the latest group-*g* message that *p* has delivered from *q* (maintained for all q at p)
- For *p* to FO-multicast *m* to *g*
  - *p increments* S^p_g *by 1.*
  - p "piggy-backs" the value S^p_g *onto the message.*
  - *p B-multicasts m to g.*
- At process *p*, Upon receipt of *m* from *q* with sequence number *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* $S < R^q_g+1$*, reject m*

Hold-back Queue for Arrived Multicast Messages



Example: FIFO Multicast

*(do **NOT** confuse with vector timestamps)*
"Accept" = Deliver



Total Ordering Using a Sequencer

Sequencer = Leader process

1. Algorithm for group member *p*

On initialization: $r_g := 0$;

To TO-multicast message m to group g
    B-multicast($g \cup \{sequencer(g)\}$, <*m, i*>);

On B-deliver(<*m, i*>) *with g = group*(*m*)
    Place <*m, i*> in hold-back queue;

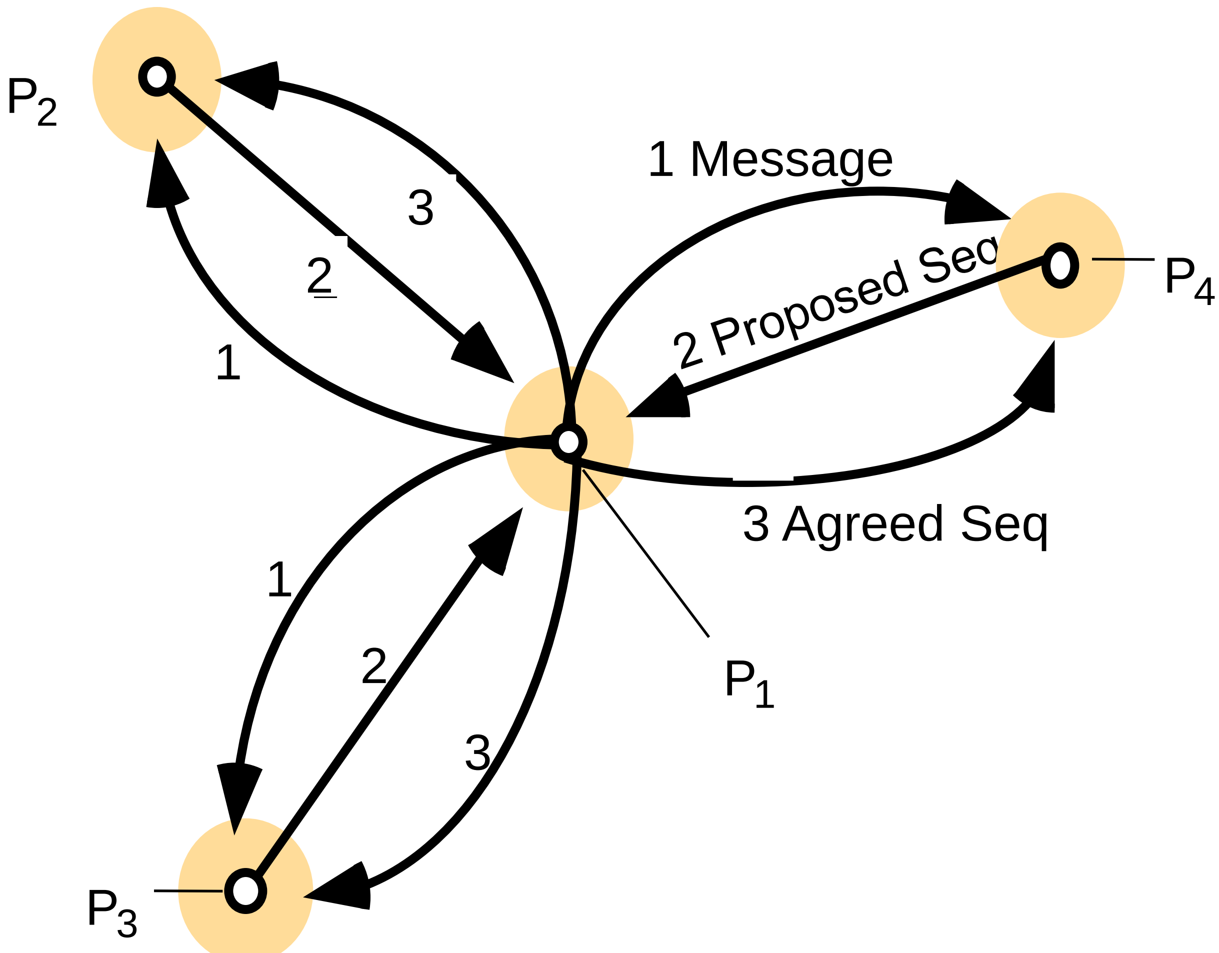
On B-deliver(m_{order} = <"order", *i, S*>) *with g = group*(m_{order})
    wait until <*m, i*> in hold-back queue and $S = r_g$;
    TO-deliver m;   // (after deleting it from the hold-back queue)
    $r_g = S + 1$;

2. Algorithm for sequencer of *g*

On initialization: $s_g := 0$;

On B-deliver(<*m, i*>) *with g = group*(*m*)
    B-multicast(*g*, <"order", *i*, s_g>);
    $s_g := s_g + 1$;

ISIS: Total ordering without sequencer



ISIS algorithm for total ordering

1. **The multicast sender multicasts the message to everyone.**
2. **Recipients add the received message to a special queue called the *priority queue*, tag the message *undeliverable*, and reply to the sender with a *proposed priority (*i.e., proposed sequence number). Further, this proposed priority is *1 more than the latest sequence number heard so far at the recipient*, suffixed with the *recipient's* process ID. *The priority queue is always sorted by priority.***
3. **The sender collects all responses from the recipients, calculates their *maximum*, and re-multicasts original message with this as the *final priority* for the message.**
4. **On receipt of this information, recipients mark the message as *deliverable*, reorder the priority queue, and deliver the set of lowest priority messages that are marked as *deliverable*.**

Proof of Total Order

- For a message m1, consider the first process *p* that delivers m1
- At *p*, when message m1 is at head of priority queue
- Suppose m2 is another message that has not yet been delivered (i.e., is on the same queue or has not been seen yet by *p*)

finalpriority(m2) >= Due to "max" operation at sender and since proposed priorities by process *p* only increase

proposedpriority(m2) > Since queue ordered by increasing priority

finalpriority(m1)

- Suppose there is some other process *p′* that delivers m2 before it delivers m1. Then at *p′*,

finalpriority(m1) >= Due to "max" operation at sender

proposedpriority(m1) > Since queue ordered by increasing priority

finalpriority(m2)

a contradiction!

Causal Ordering using vector timestamps

Algorithm for group member p_i $(i = 1, 2..., N)$

On initialization

$V_i^g[j] := 0 \; (j = 1, 2..., N);$

The number of group-g messages from process j that have been seen at process i so far

To CO-multicast message m to group g

$V_i^g[i] := V_i^g[i] + 1;$

B-multicast$(g, <V_i^g, m>);$

On B-deliver$(<V_j^g, m>)$ *from* p_j, *with* $g = group(m)$

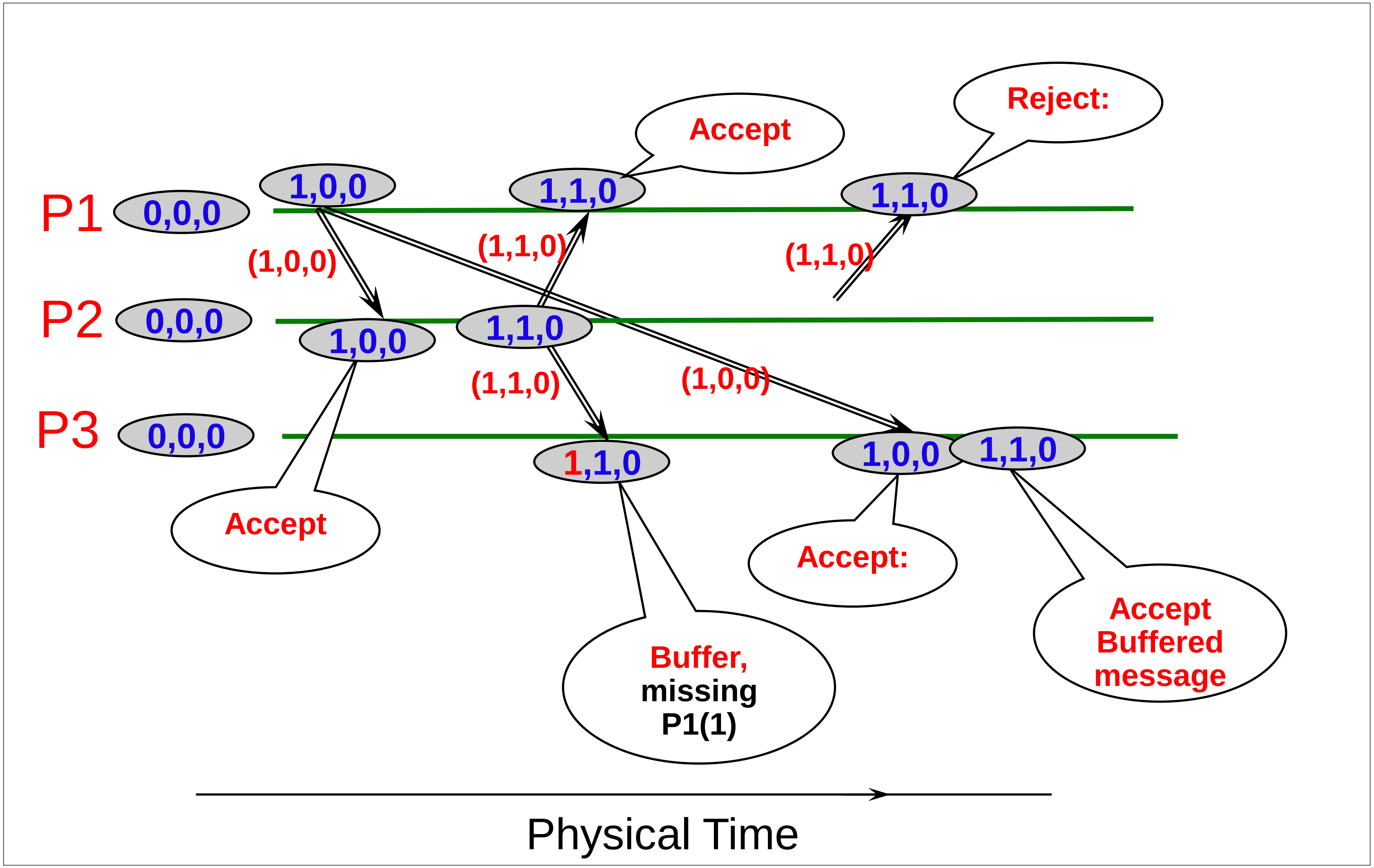
place $<V_j^g, m>$ in hold-back queue;

wait until $V_j^g[j] = V_i^g[j] + 1$ and $V_j^g[k] \le V_i^g[k] \; (k \ne j);$

CO-deliver m; // after removing it from the hold-back queue

$V_i^g[j] := V_i^g[j] + 1;$

Example: Causal Ordering Multicast



Summary

Multicast is operation of sending one message to multiple processes in a given group

- Reliable multicast algorithm built using unicast
- Ordering – FIFO, total, causal

Thursday

- RPCs: Section 4.3, parts of Chapter 5
  - Important for MP2
- Homework 1 due this Thursday
  - Hand in to me at start of lecture (not during or after lecture)