Computer Science 425 Distributed Systems (Fall2009)

Lecture 4
Chandy-Lamport Snapshot Algorithm and Multicast Communication
Reading: Section 11.5&12.4

Klara Nahrstedt

Acknowledgement

- **The slides during this semester are based on ideas and material from the following sources:**
  - **Slides prepared by Professors M. Harandi, J. Hou, I. Gupta, N. Vaidya, Y-Ch. Hu, S. Mitra.**
  - **Slides from Professor S. Gosh's course at University o Iowa.**

Administrative

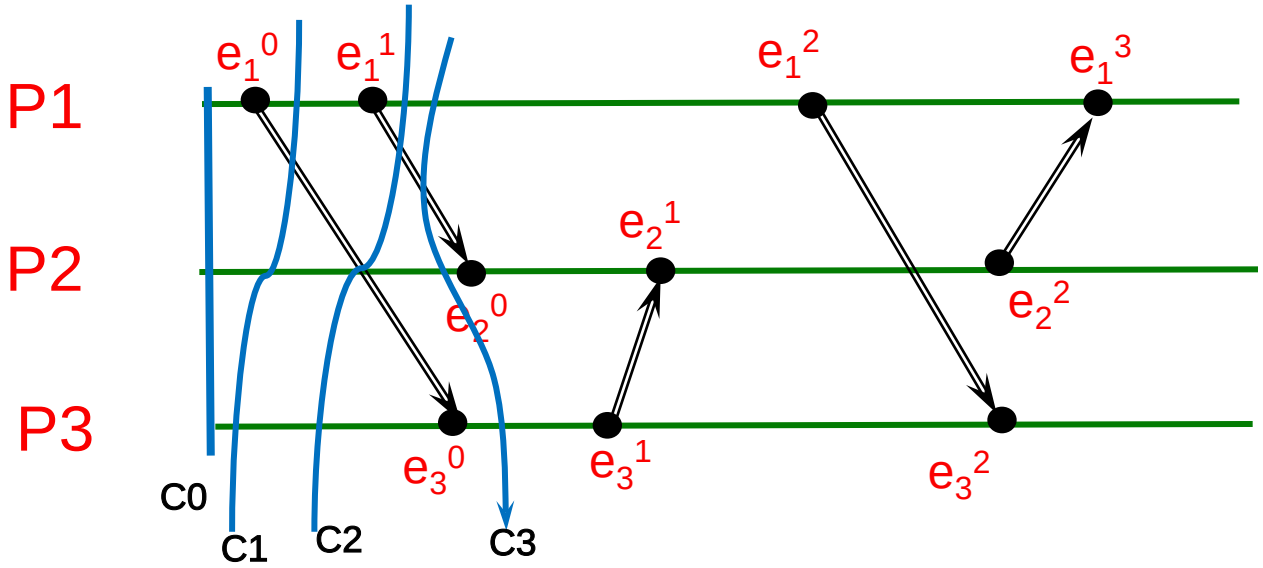
- **Form Groups for MP projects**
  - **Up to 3 members per group**
  - **Email names to TA today**
- **Homework 1 posted today, September 3 (Thursday)**
  - **Deadline, September 17 (Thursday)**
- **Introductory material to get introduced to the Eclipse/Android Programming Environment is posted –**
  - **Optional MP0**
  - **Posted on the class website**

Plan for Today

- **Chandy-Lamport Global Snapshot Algorithm**
- **New Topic: Multicast Communication**

Review

- **A run is a total ordering of events in *H* that is consistent with each h_i's ordering**
  - **E.g., $<e_1^0, e_1^1, e_1^2, e_1^3, e_2^0, e_2^1, e_2^2, e_3^0 e_3^1, e_3^2>$**
- **A linearization is a run consistent with happens-before (→) relation in H**
  - **E.g., $< e_1^0, e_1^1, e_3^0, e_2^0, \ldots>$, $< e_1^0, e_3^0, e_1^1, e_2^0, \ldots>$**
  - ***Concurrent events are ordered arbitrarily***
  - ***Linearizations pass through consistent global states***



Chandy-Lamport "Snapshot" Algorithm

- "Snapshot" is a set of process and channel states for set of processes P_i $(i=1,\ldots N)$ such that the recorded global state is consistent
  - Even those the combination of recorded states may never occurred at the same time
- Chandy-Lamport Algorithm records a set of process and channel states such that the combination is a consistent global state.
- *Assumptions:*
  - No failure, all messages arrive intact, exactly once
  - Communication channels are unidirectional and FIFO-ordered
  - There is a comm. channel between each pair of processes
  - Any process may initiate the snapshot (send "Marker")
  - Snapshot does not interfere with normal execution

Chandy-Lamport Snapshot Algorithm (2)

1. **Initiator process P_0 records its state locally**

2. **Marker sending rule for process P_i:**

   After P_i has recorded its state, for each outgoing channel Ch_{ij}, P_i sends *one marker* message over Ch_{ij} (before P_i sends any other message over Ch_{ij})

3. **Marker receiving rule for process P_i:**

   Process P_i on receipt of a marker over channel Ch_{ji}

   If (P_i has not yet recorded its state) it

   Records *its process state* now;

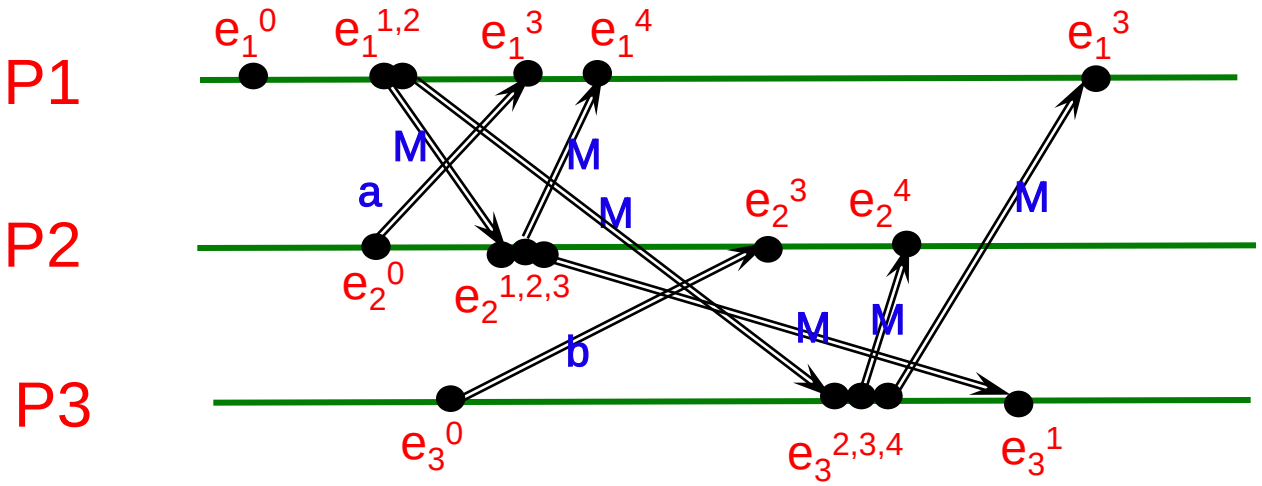
   Records the state of Ch_{ji} as *empty set;*

   Starts recording messages arriving over other incoming channels;

   else (P_i has already recorded its state)

   P_i records the state of Ch_{ji} as the set of all messages it has received over Ch_{ji} since *it* saved its state

Snapshot Example



1- P1 initiates snapshot: records its state (S1); sends Markers to P2 & P3; turns on recording for channels Ch_{21} and Ch_{31}

2- P2 receives Marker over Ch_{12}, records its state (S2), sets state(Ch_{12}) = {} sends Marker to P1 & P3; turns on recording for channel Ch_{32}

3- P1 receives Marker over Ch_{21}, sets state(Ch_{21}) = {a}

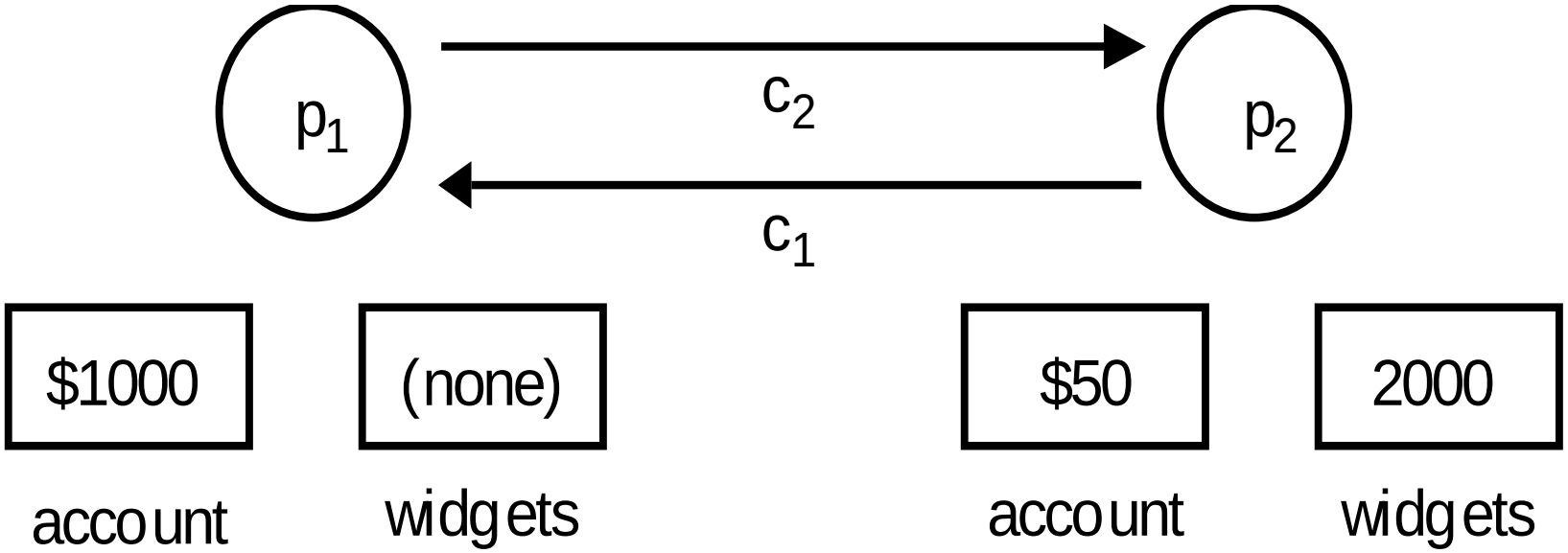
4- P3 receives Marker over Ch_{13}, records its state (S3), sets state(Ch_{13}) = {} sends Marker to P1 & P2; turns on recording for channel Ch_{23}

5- P2 receives Marker over Ch_{32}, sets state(Ch_{32}) = {b}

6- P3 receives Marker over Ch_{23}, sets state(Ch_{23}) = {}

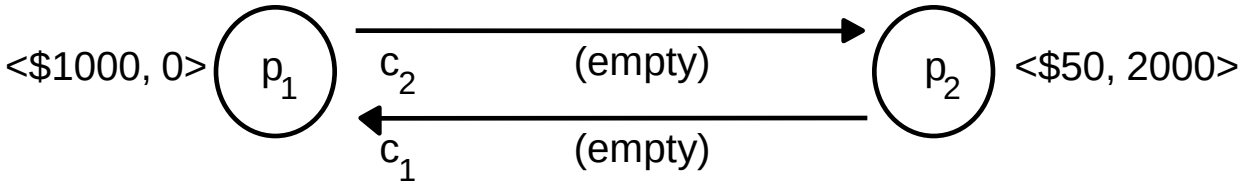
7- P1 receives Marker over Ch_{31}, sets state(Ch_{31}) = {}

Another Example: Trading

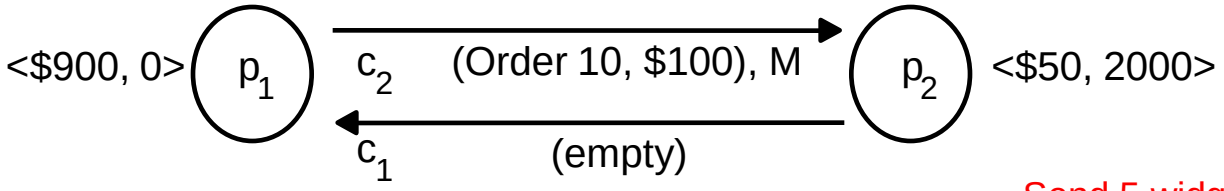


Execution of the Processes

1. Global state S_0

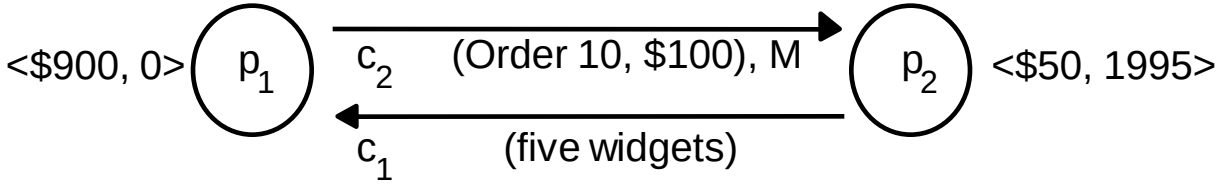


2. Global state S_1

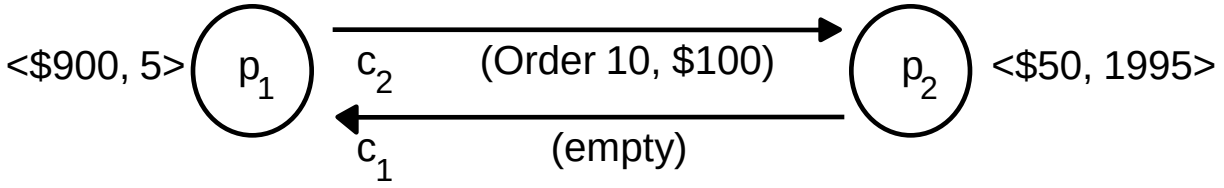


Send 5 widgets

3. Global state S_2

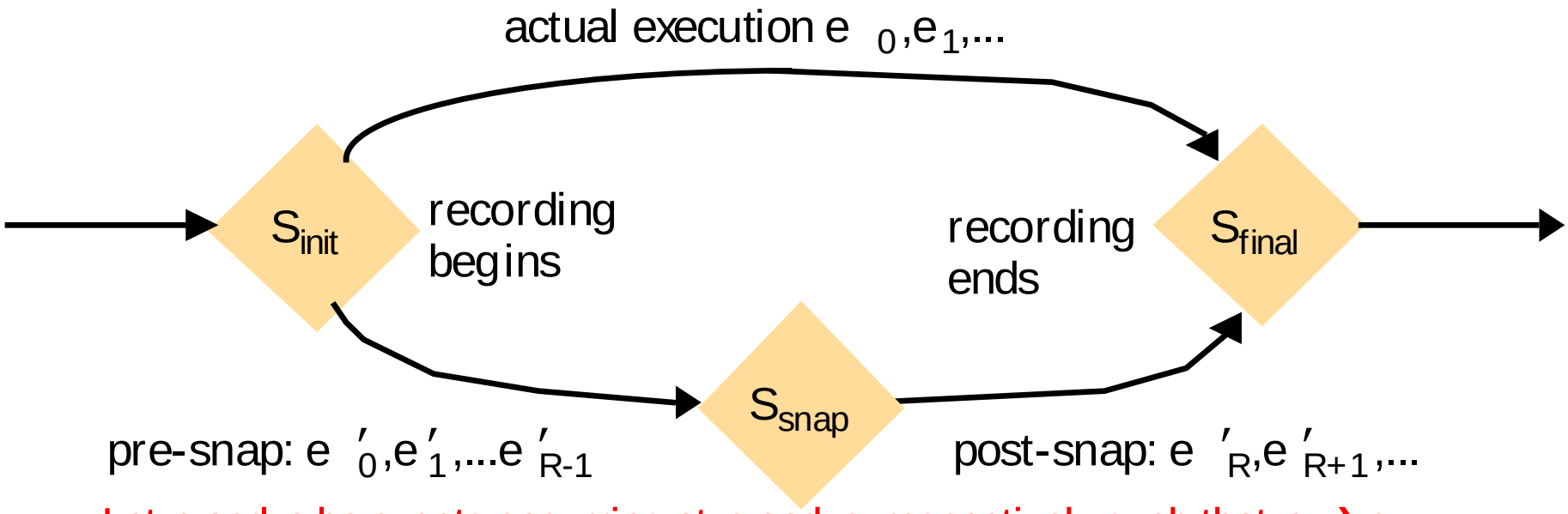


4. Global state S_3



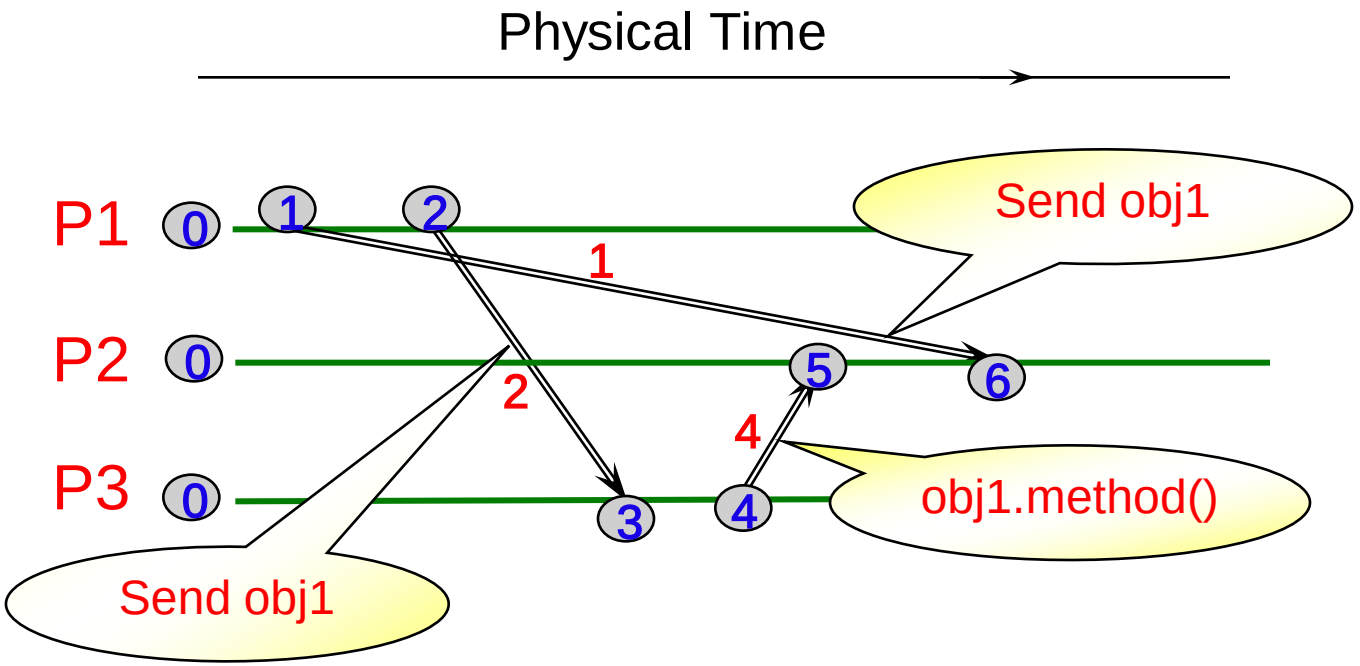
(M = Marker Message)

Reachability between States in Snapshot Algorithm



Let e_i and e_j be events occurring at p_i and p_j, respectively such that $e_i \rightarrow e_j$
The snapshot algorithm ensures that
- if e_j is in the cut then e_i is also in the cut.
- if e_j occurred before p_j recorded its state, then e_i must have occurred before p_i recorded its state.

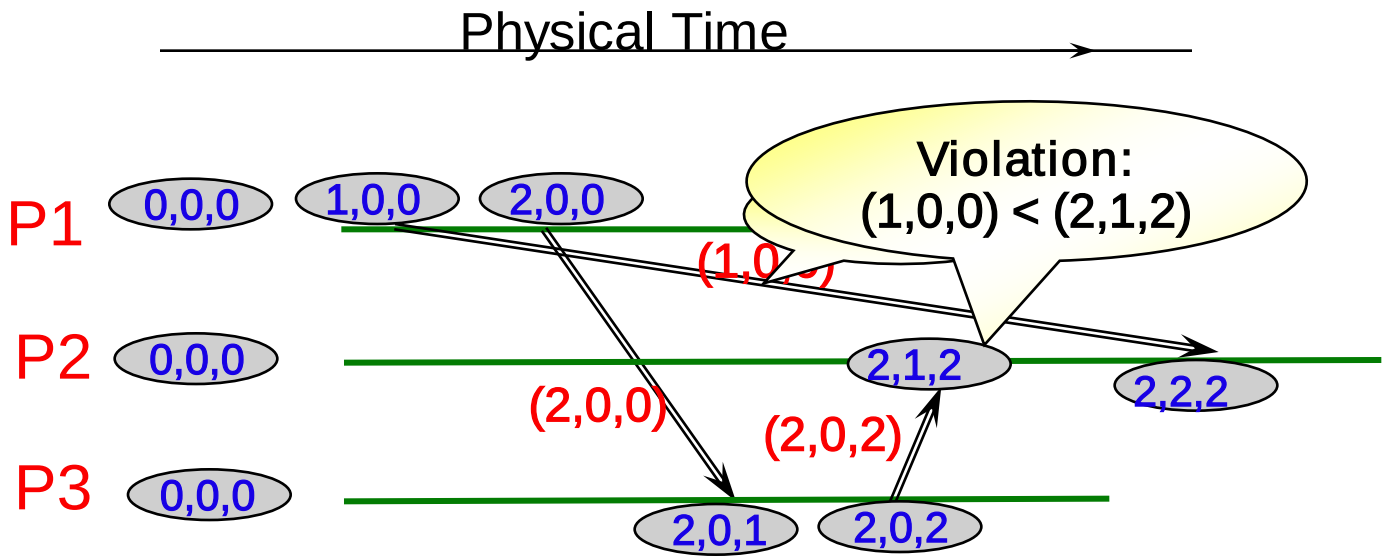
Causality Violation



- Causality violation occurs when order of messages causes an action based on information that another host has not yet received.

- In designing a DS, potential for causality violation is important

Detecting Causality Violation



- Potential causality violation can be detected by vector timestamps.

- If the vector timestamp of a message is less than the local vector timestamp, on arrival, there is a **potential causality violation**.

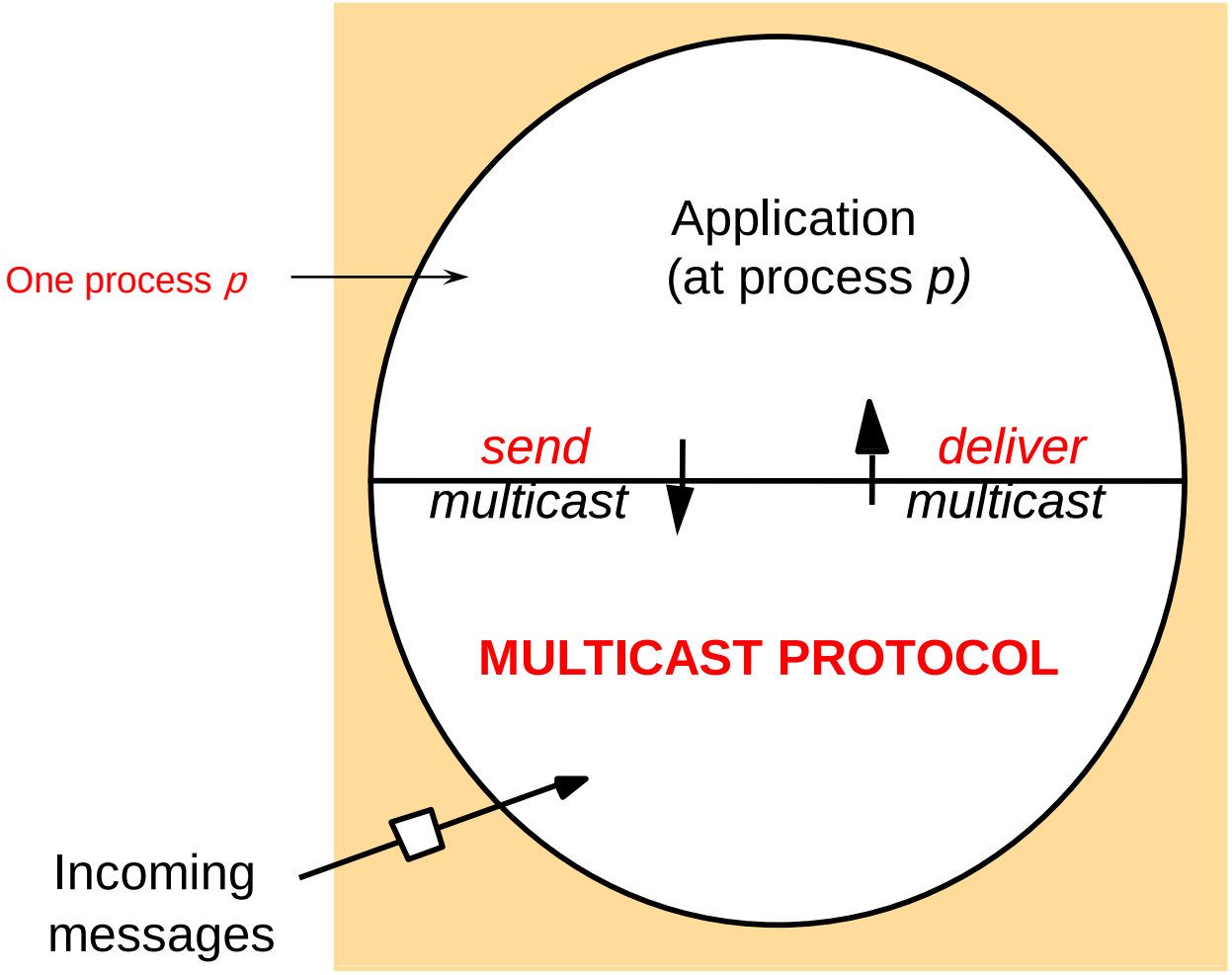
Multicast Communication

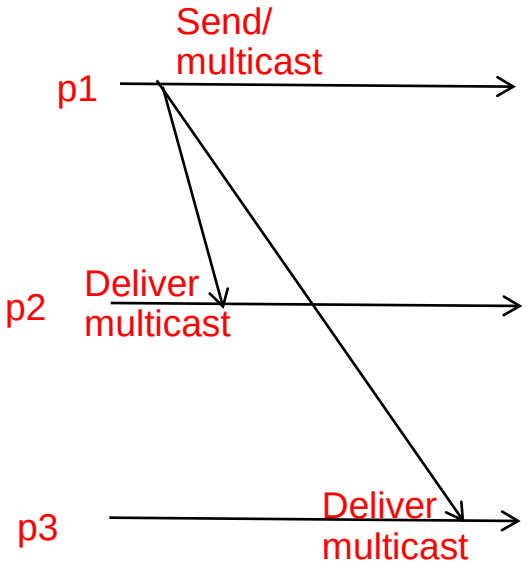
Communication Modes in DS

- **Unicast (*best effort* or *reliable*)**
  - Messages are sent from exactly one process to one process.
  - *Best effort* guarantees that if a message is delivered it would be intact.
  - *Reliable* guarantees delivery of messages.
- **Broadcast**
  - Messages are sent from exactly one process to all processes on the network.
  - Reliable broadcast protocols are not practical.
- **Multicast**
  - Messages are sent from exactly one process to several processes on the network.
  - Reliable multicast can be implemented "above" (i.e., "using") a reliable unicast.

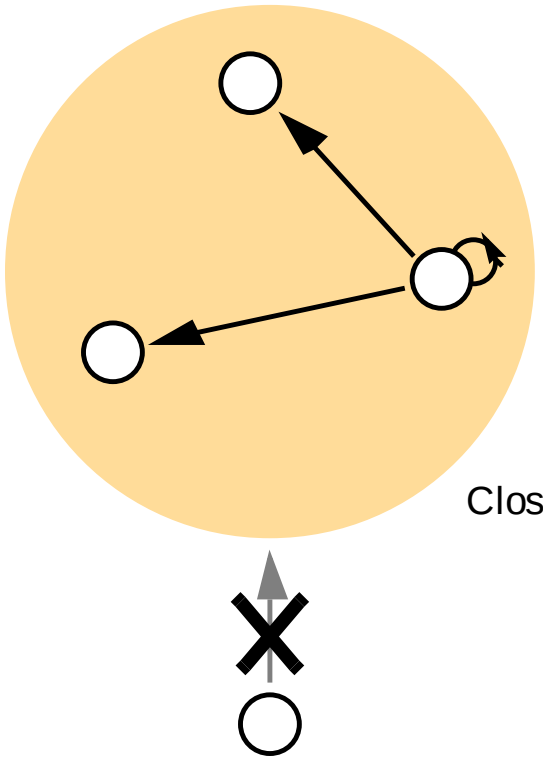

NYSE
NYSE
NYSE
NYSE
NYSE

What're we designing in this class

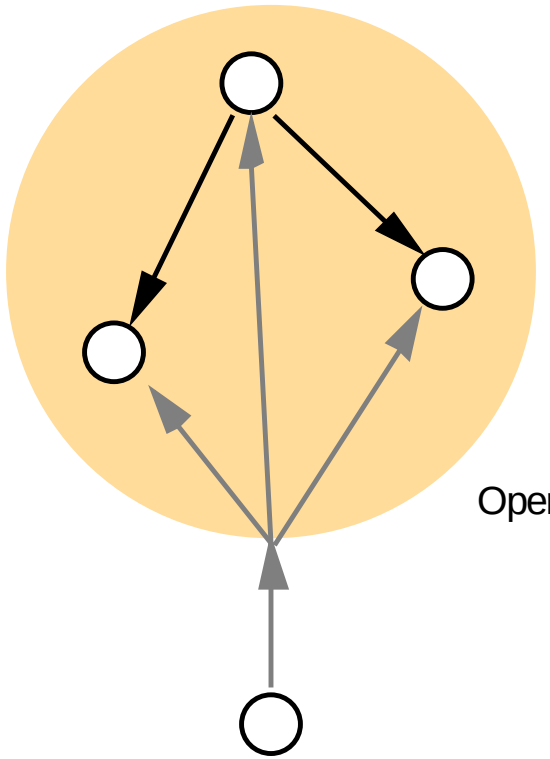




Open and closed groups



Closed group



Open group

Basic Multicast (B-multicast)

- ***A straightforward way to implement B-multicast is to use a reliable one-to-one send operation:***

  B-multicast(g,m): for each process *p* in *g*, *send (p,m).*

  On receive(m) at p: *B-deliver(m)* at *p*.

- **A "correct" process= a "non-faulty" process**

- **Basic multicast (B-multicast) primitive guarantees a correct (i.e., non-faulty) 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)?**

Summary

Multicast is operation of sending one message to multiple processes

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

Next week

- Read Sections 12.4
- Homework 1 due September 17
  - Come by office hours with your questions