Attention please!

- March 12 class is cancelled.
- Assignment#2 will be due in class on March 19.
- The text of Assignment#2 is also online.
- The project intermediate report will be due on April 2 in class.
- Midterm Exam is re-scheduled to April 18 in class and it will cover all chapters except Chapter 17.
- Please read the new Class Schedule online.

Global state of a distributed system

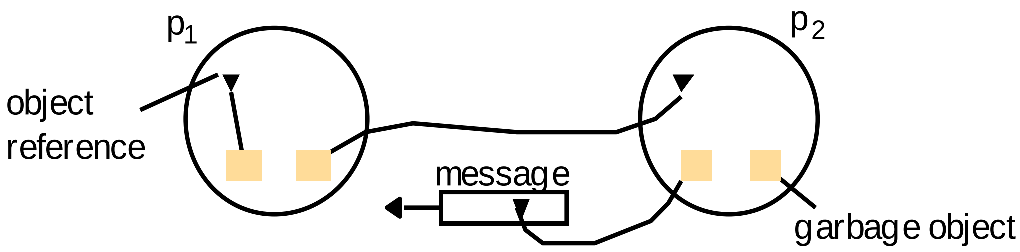
- Local state of each process
- The messages that are currently in transit (sent but not received)
- Purpose: Finding out whether a particular property is true of a distributed system as it executes.

Who cares, globally speaking?

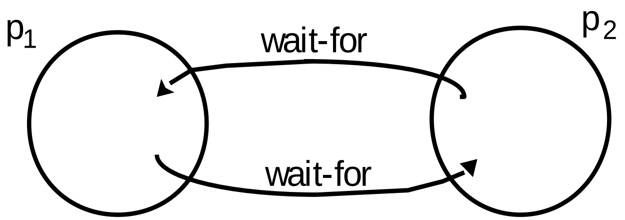
- When it is known that local computations have stopped and that there are no more messages in transit, the system has obviously entered a state in which no more progress can be made.
  - deadlocked?
  - correctly terminated?

They all need a global state!

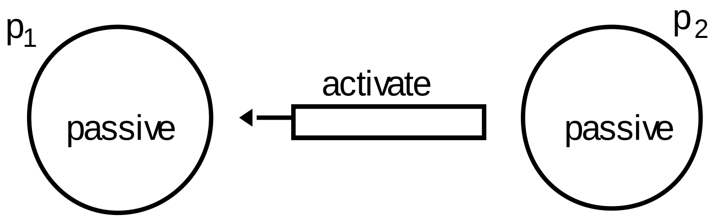
a. Garbage collection



b. Deadlock



c. Termination



It’s much harder…

- To observe the succession of states of an individual process is relatively easy

- To ascertain a global state of a distributed system, which includes a collection of processes, is much harder

- Why?

No global time

How to record the global state

- Distributed snapshot
  - reflects a state in which the distributed system *might* have been
  - reflects a consistent global state
  - If we have recorded that process P has received a msg from another process Q, then we should also have recorded that process Q had actually *sent* the msg
  - The reverse condition (Q has sent a msg that P has not yet received) is allowed.

Important Terms (P612-614)

$$history(p_i) = h_i = <e_i^0, e_i^1, e_i^2, \ldots>$$

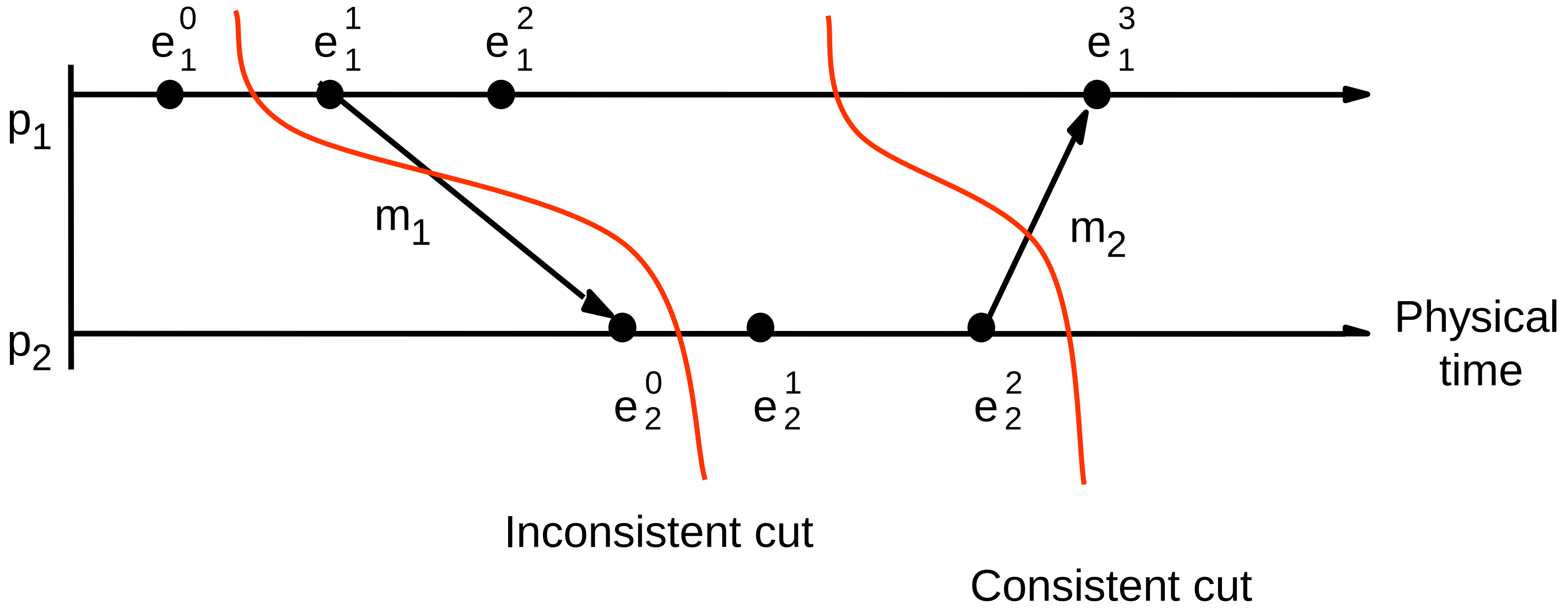
$$h_i^k = <e_i^0, e_i^1, \ldots, e_i^k>$$

- ***History***$(p_i)=h_i$
- The ***global history*** of a distributed system is the union of the individual process histories $H = h_0 \cup h_1 \cup \ldots \cup h_{N-1}$
- A ***global state*** corresponds to initial prefixes of the individual process histories $S = (s_1, s_2, \ldots s_N)$
- A ***cut*** of the system's execution is a subset of its global history $C = h_1^{c_1} \cup h_2^{c_2} \cup \ldots \cup h_N^{c_N}$
- A ***cut*** is ***consistent*** if, for each event it contains, it also contains all the events that happened-before that event
- A ***consistent global state*** is one that corresponds to a consistent cut

Cut!

- A ***cut frontier*** represents the last event that has been recorded for each of several processes.
  - All recorded msg receipts have a corresponding recorded send event $\{e_i^{c_i} : i = 1,2,\ldots,N\}$
- An ***inconsistent cut*** would have a receipt of a msg but no corresponding send event

Cut examples



$<e_1^0, e_2^0>$ and $<e_1^2, e_2^2>$

Consistent Global State

- A consistent global state is one that corresponds to a consistent cut.
- The execution of a distributed system is a series of transitions between global states of the system:

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots$$

- In each transition, precisely one event occurs at some single process in the system.

Global State Predicates

- Detecting a condition such as deadlock or termination amounts to evaluating a *global state predicate*.
- A global state predicate is a function that maps from the set of global states of processes in the system to {True, False}.
- Once the system enters a state in which the predicate is True, it remains True in all future states reachable from that state.

The Snapshot Algorithm

- The goal of this algorithm is to record a set of process and channel states (a ‘snapshot’) for a set of processes pi such that, even though the combination of recorded states may never have occurred at the same time, the recorded global state is consistent.

Assumptions for Chandy & Lamport Algorithm

- Neither channels nor processes fail
- Channels are unidirectional and provide FIFO-ordered message delivery
- The graph of processes and channels is strongly connected
- Any process may initiate a global snapshot at any time
- The processes may continue their execution and send and receive normal messages while the snapshot takes place

The algorithm (Chandy & Lamport)

- Assume the distributed system can be represented as a collection of processes connected to each other through uni-directional point-to-point communication channels.
- *Any* process may initiate the algorithm.
  - *P records its own local state*
  - *It sends a* ***marker*** *along each of its* outgoing *channels, indicating that the receiver should participate in recording the global state*
  - ...

Marker

- A marker is a special message, which is distinct from any other messages that the processes send and receive

- It has two roles:

1. As a prompt for the receiver to save its own state, if it has not already done so
2. As a means of determining which messages to include in the channel state

Chandy & Lamport algorithm (continued)

- When process Q receives the marker through an incoming channel C, its action depends on whether or not it has already saved its local state
- If it has not
  - it first records its local state and also sends a marker along its own outgoing channels
- If it has
  - the marker on channel C is an indicator that Q should record the state of the *channel*, namely, the sequence of messages received by Q since the last time it recorded its own local state and before it received the marker.

Chandy & Lamport algorithm (continued)

- A process has finished its part of the algorithm when it has received a marker along each of its incoming channels and processed each one.
- Its recorded local state as well as the state it recorded for each incoming channel, can be collected and sent to the process that initiated the snapshot
- The initiator can subsequently analyze the current state
- Meanwhile, the distributed system as a whole can continue to run normally

Figure 11.10

```
Marker receiving rule for process $p_i$
  On $p_i$'s receipt of a marker message over channel $c$:
    if ($p_i$ has not yet recorded its state) it
        records its process state now;
        records the state of $c$ as the empty set;
        turns on recording of messages arriving over other incoming channels;
    else
        $p_i$ records the state of $c$ as the set of messages it has received over $c$
        since it saved its state.
    end if
Marker sending rule for process $p_i$
  After $p_i$ has recorded its state, for each outgoing channel $c$:
    $p_i$ sends one marker message over $c$
    (before it sends any other message over $c$).
```

Two rules

- The ***marker sending rule*** obligates processes to send a marker after they have recorded their state, but before they send any other messages

- The ***marker receiving rule*** obligates a process that has not recorded its state to do so.

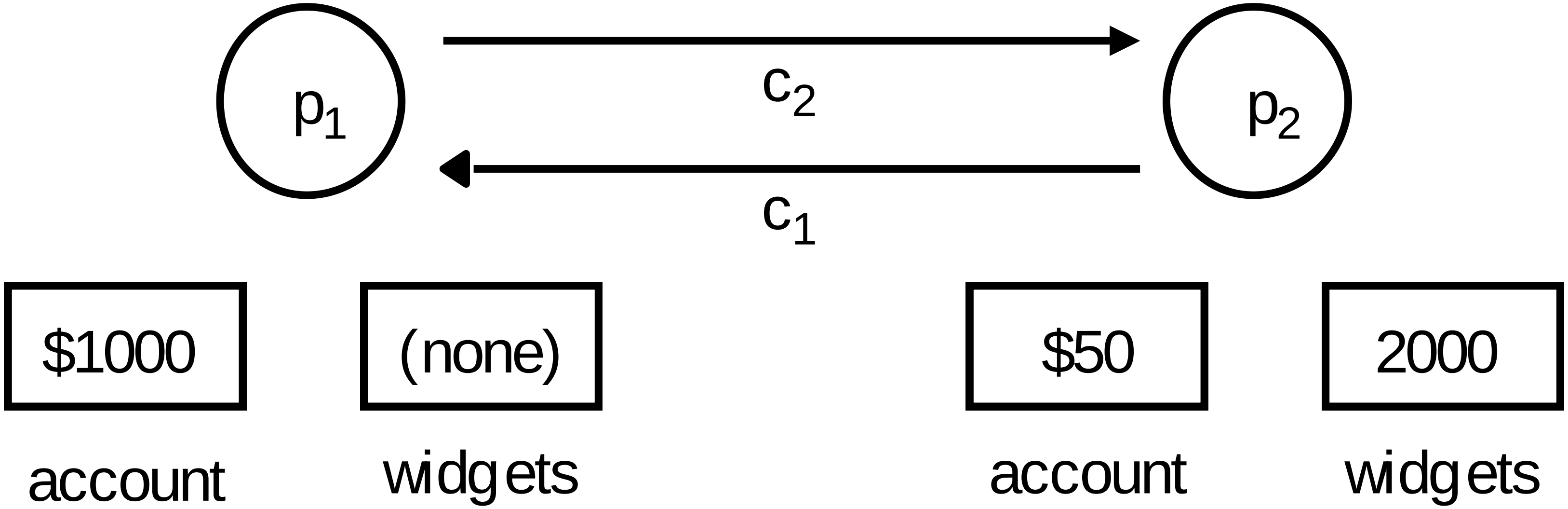
Photo album

- Because *any* process can initiate the algorithm, the construction of several snapshots may be in progress at the same time
- A marker is tagged with the identifier and possibly also a version number of the process that initiated the snapshot
- Only after a process has received *that marker* through each of its incoming channels, can it finish its part in the construction of the marker’s associated snapshot

Initializer

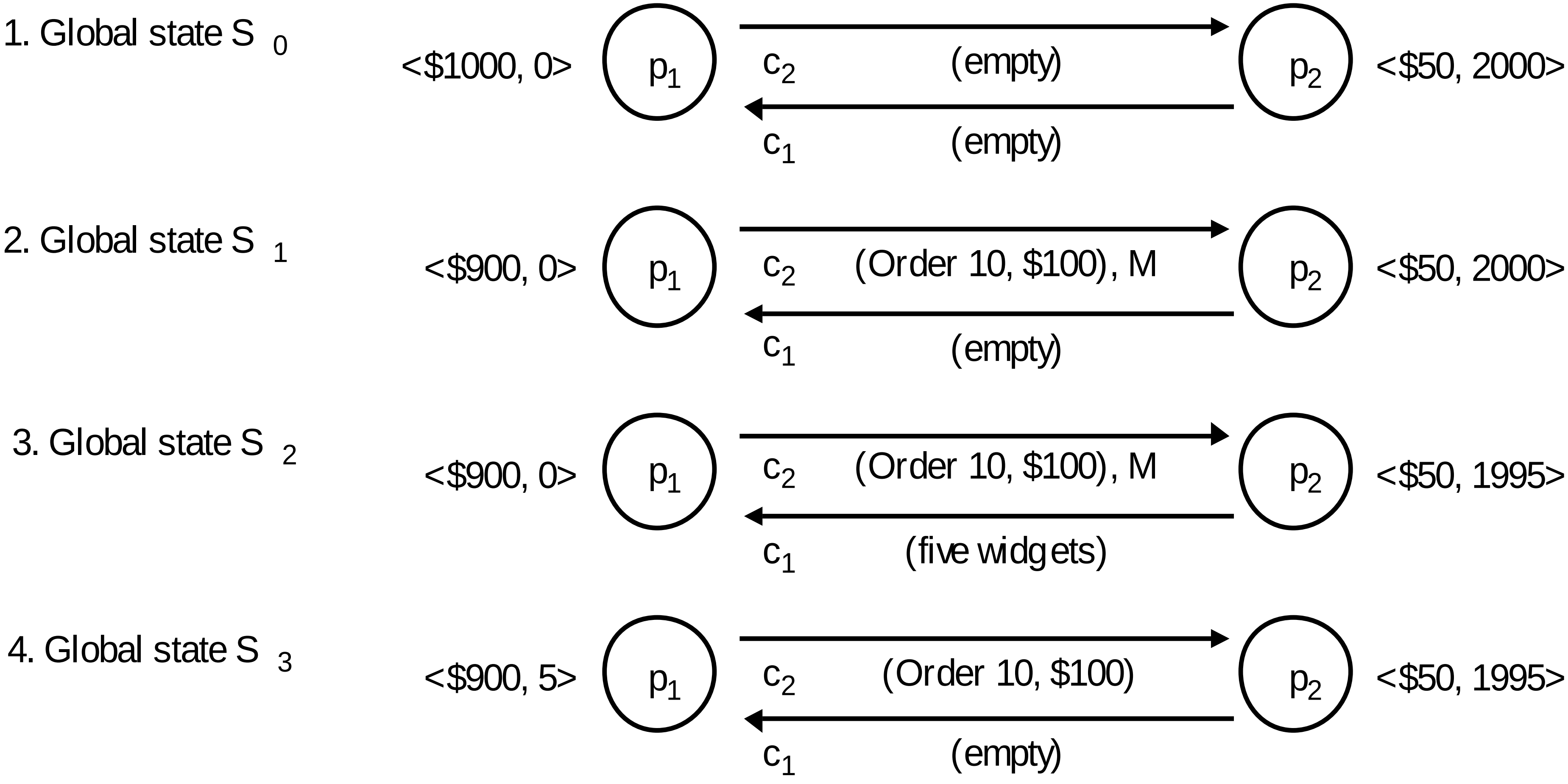
- Any process may begin the algorithm at any time.
- It acts as if it has received a marker (over a non-existent channel) and follows the marker receiving rule.
- Thus, it records its state and begins to record messages arriving over all its incoming channels.

Example (Initial State)



Initial state: Process P2 has already received an order of five widgets, which it will shortly dispatch to P1.

Example (The execution of the processes)



1. Global state S_0

<\$1000, 0> p_1 — c_2 (empty) — p_2 <\$50, 2000>
c_1 (empty)

2. Global state S_1

<\$900, 0> p_1 — c_2 (Order 10, \$100), M — p_2 <\$50, 2000>
c_1 (empty)

3. Global state S_2

<\$900, 0> p_1 — c_2 (Order 10, \$100), M — p_2 <\$50, 1995>
c_1 (five widgets)

4. Global state S_3

<\$900, 5> p_1 — c_2 (Order 10, \$100) — p_2 <\$50, 1995>
c_1 (empty)

(M = marker message)

Explanation to Last Slide

1. P1 records its state in S0, when P1's state is <$1000,0>. Following the marker sending rule, P1 then emits a marker message M over its outgoing channel C2 before it sends the next order message (Order 10, $100) over C2. The system enters actual global state S1;
2. Before P2 receives the marker, it emits an application message (five widgets) over C1 in response to P's previous order, yielding a new actual global state S2.
3. Now P1 receives P2's message (five widgets), and P2 receives the marker. Following the marker receiving rule, P2 records its state as <$50, 1995> and that of C2 as the empty sequence. Following the marker sending rule, it sends a marker message over C1.
4. When P1 receives P2's marker message, it records the state of C1 as the single message (five widgets) that it received after it first recorded its state. The final actual global state is S3.

Within a finite time!

- We assume that a process that has received a marker message records its state within a finite time and sends marker messages over each outgoing channel within a finite time.
- If there is a path from Pi to Pj, Pj will record its state a finite time after Pi recorded its state.
- We assume that the graph of processes and channels are strongly connected.
- So, all processes will have recorded their states and the states of incoming channels in a finite time after some process initially records its state.

Termination Detection of the Snapshot

- If a process Q receives the marker requesting a snapshot for the first time,
  - considers the process that sent that marker as its predecessor
- When Q completes its part of the snapshot, it sends its predecessor a DONE msg.
- By recursion, when the initiator of the distributed snapshot has received a DONE msg from all of its successors, it knows the snapshot has been completely taken

What if msgs still in transit?

- A snapshot may show a global state in which msgs are still in transit
- Suppose a process records that it had rec’d msgs along one of its incoming channels
  - between the point where it had recorded its local state
  - and the point where it received the marker through that channel
- Cannot conclude the distributed computation is completed
- Termination requires a snapshot in which all channels are empty

Modified algorithm

- When a process Q finishes its part of a snapshot, it either returns DONE or CONTINUE to its predecessor
- A DONE msg is returned only when
  - All of Q’s successors have returned a DONE msg
  - Q has not received any msg between the point it recorded its own local state and the point it had received the marker along each of its incoming channels
- In all other cases, Q sends a CONTINUE msg to its predecessor

Modified algorithm (continued)

- The original initiator of the snapshot will either receive at least one CONTINUE or only DONE msgs from its successors
- When only DONE messages are received, it is known that no regular msgs are in transit
- Conclusion? The computation has terminated.
- If a CONTINUE appears, P initiates another snapshot and continues to do so until only DONE msgs are returned.

(There are lots of other algorithms, too.)

Assignment#2 (chapter 14)

- 14.1
- 14.2
- 14.4
- 14.13