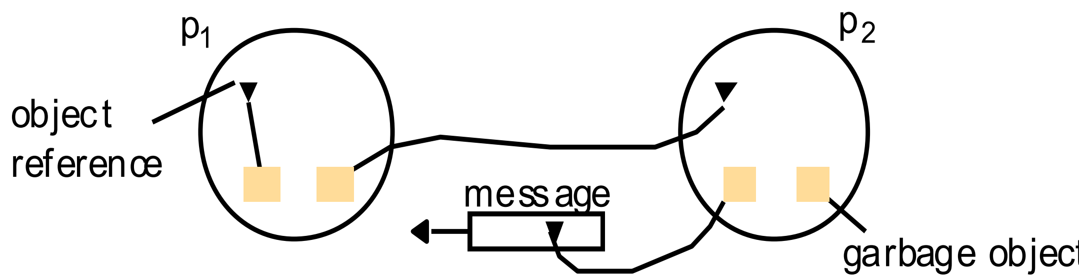
Distributed Systems

CS 425 / ECE 428

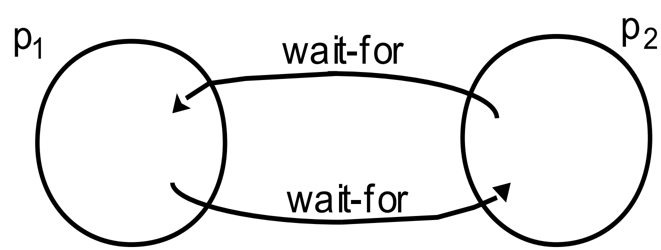
Global States, Distributed Snapshots

Detecting Global Properties

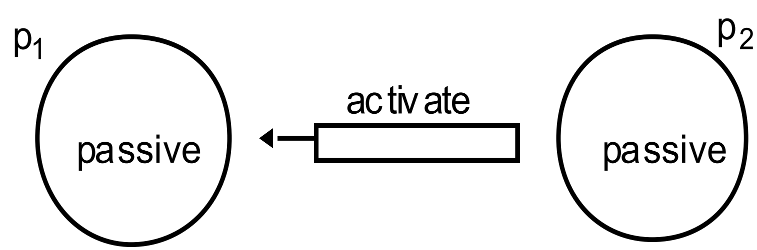
a. Garbage collection



b. Deadlock



c. Termination



Algorithms to Find Global States

- **Why?**
  - (Distributed) garbage collection [think multiple processes sharing and referencing objects]
  - (Distributed) deadlock detection, termination [think database transactions]
  - Global states most useful for detecting stable predicates : once true always stays true (unless you do something about it)
    - » e.g., once a deadlock, always stays a deadlock
- **What?**
  - Global state=states of all processes + states of all communication channels
  - Capture the instantaneous *state* of each process
  - And the instantaneous *state* of each communication channel, i.e., *messages* in transit on the channels
- **How?**
  - We'll see this lecture!

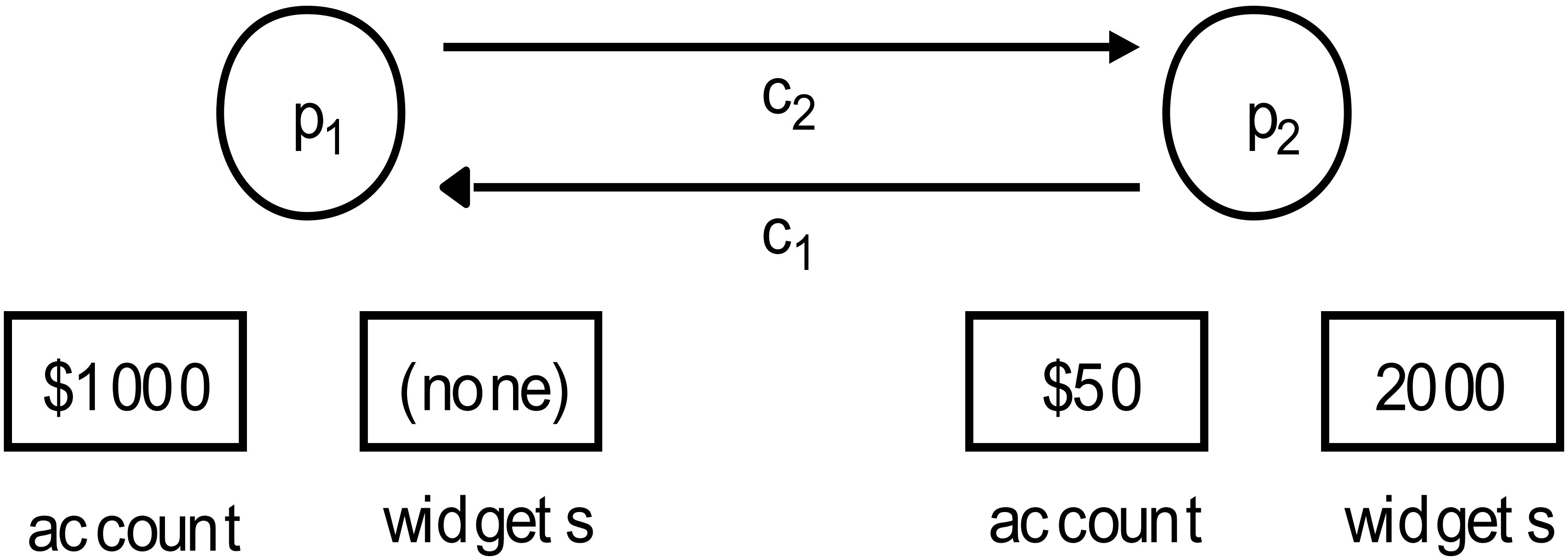
Obvious First Solution…

- Synchronize clocks of all processes
- Ask all processes to record their states at known time t

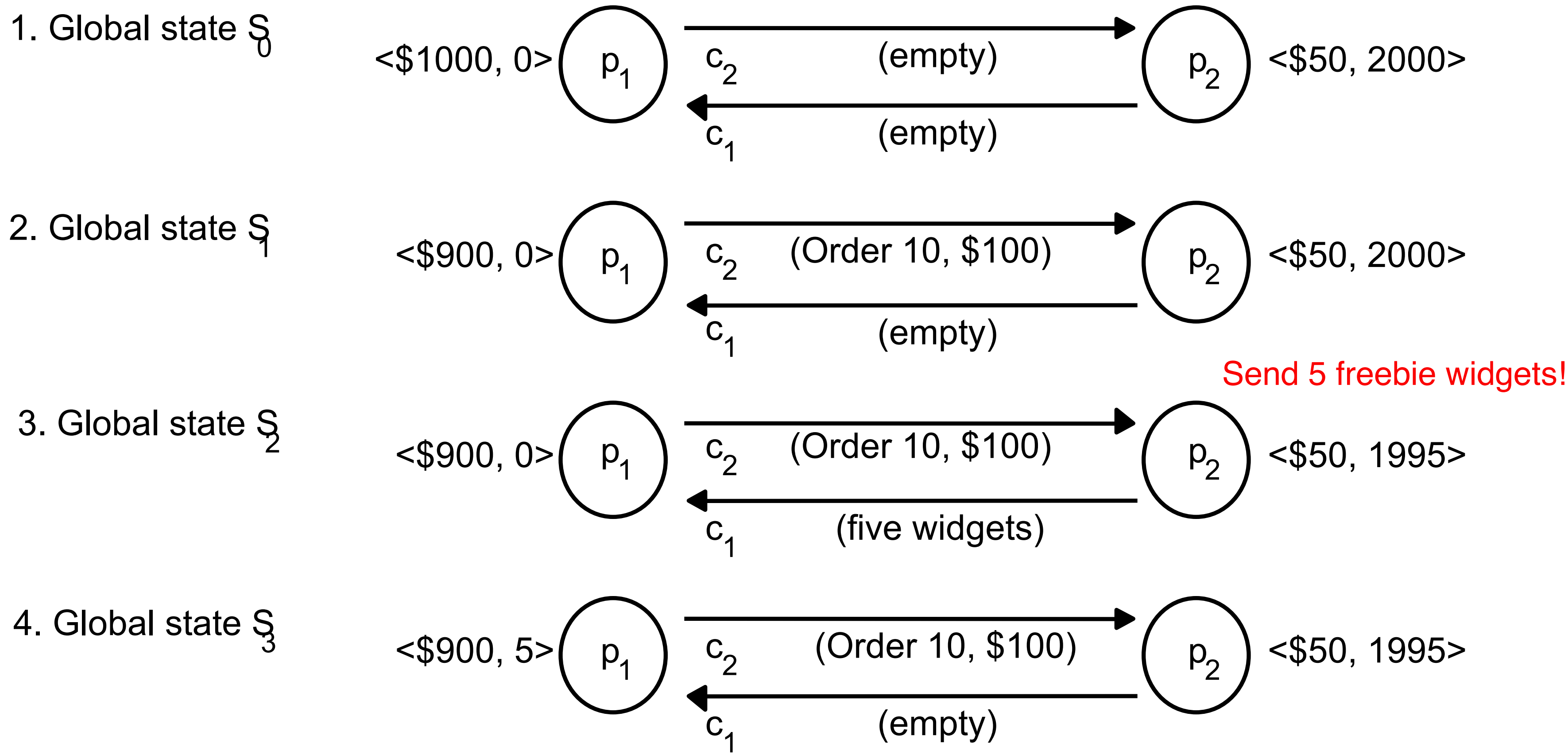
- Problems?
  - Time synchronization possible only approximately (but distributed banking applications cannot take approximations)
  - Does not record the state of messages in the channels

- Synchronization not required – causality is enough!

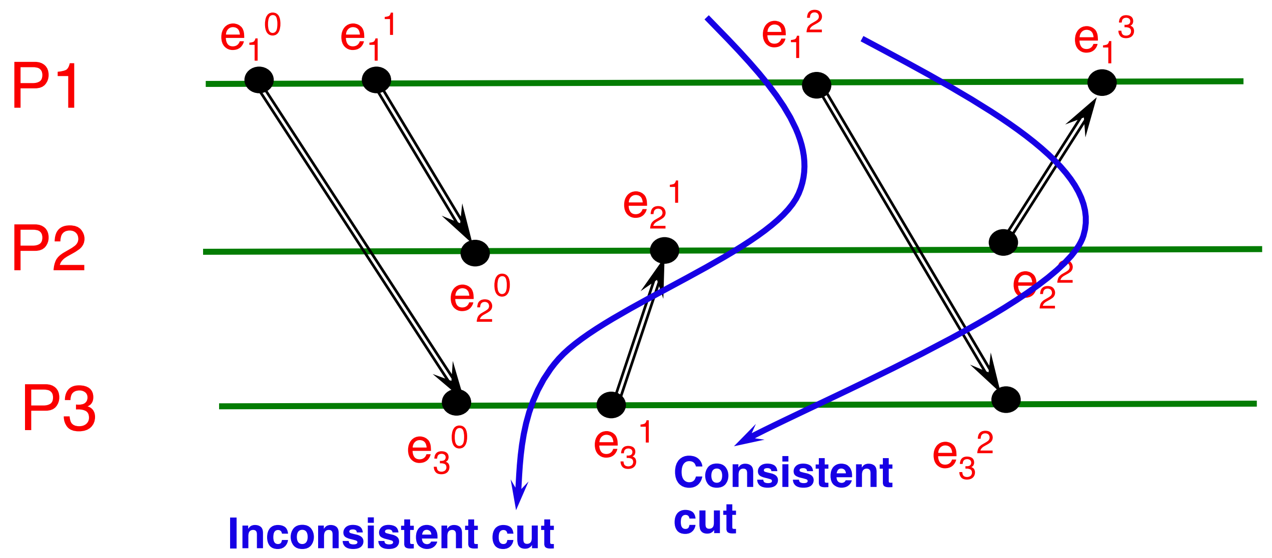
Two Processes and Their Initial States



Execution of the Processes

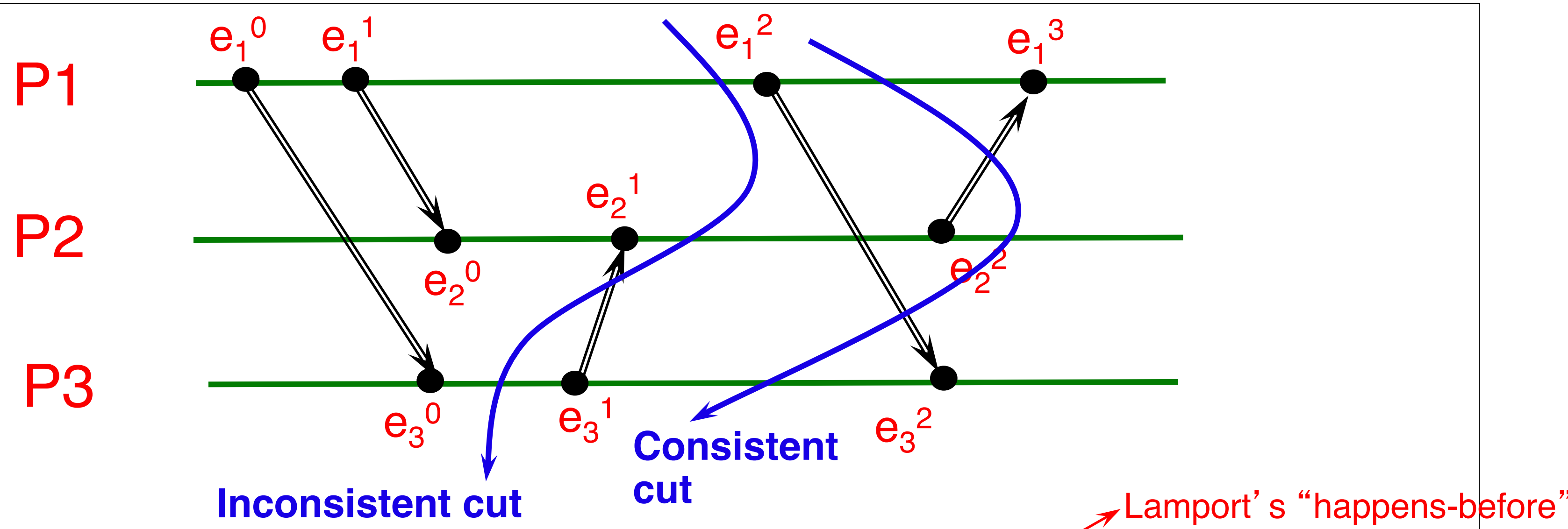


Cuts



- **Cut** = time frontier, one at each process
- *f* $\in$ cut *C* iff *f* is to the left of the frontier *C*

Consistent Cuts



- ***f*** $\in$ **cut *C* iff *f* is to the left of the frontier *C***
- **A cut C is consistent if and only if**

$$\forall_{e \in C} (\text{if } f \rightarrow e \text{ then } f \in C)$$

- **A global state S is consistent if and only if it corresponds to a consistent cut**
- **A consistent cut == a global snapshot**

The "Snapshot" Algorithm

- **Problem:** **Record a set of process and channel states such that the combination is a global snapshot/consistent cut.**
- ***System Model:***
  - **There is a uni-directional communication channel between each ordered process pair (Pj → Pi and Pi → Pj)**
  - **Communication channels are FIFO-ordered**
  - **No failure, all messages arrive intact, exactly once**
  - **Any process may initiate the snapshot (by sending a special message called "Marker")**
  - **Snapshot does not require application to stop sending messages, does not interfere with normal execution**
  - **Each process is able to record its state and the state of its incoming channels (no central collection)**

The "Snapshot" Algorithm (2)

1. **Algorithm for for initiator process P_0**
   - **After P_0 has recorded its own state**
     - **for each outgoing channel C, send a marker message on C, and start recording messages on all incoming channels**
2. **Marker receiving rule for a process P_k on receipt of a marker over channel C**

CORRECTIONS MADE HERE

   - **if P_k has not yet recorded its own state**
     - **record P_k's own state**
     - **record the state of C as "empty"**
     - **for each outgoing channel C, send a marker on C**
     - **turn on recording of messages over other incoming channels**
   - **else**
     - **record the state of C as all the messages received over C since P_k saved its own state; stop recording state of C**

Chandy and Lamport's 'Snapshot' Algorithm

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.

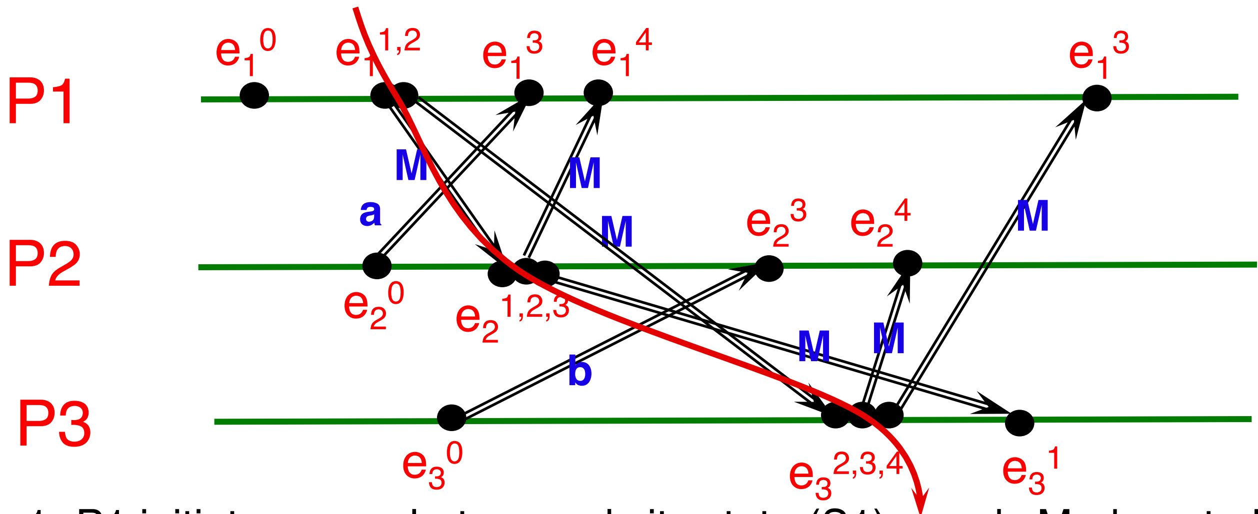
end if

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).

Snapshot Example



1- P1 initiates snapshot: records its state (S1); sends Markers to P2 & P3; turns on recording for channels C21 and C31

2- P2 receives Marker over C12, records its state (S2), sets state(C12) = {} sends Marker to P1 & P3; turns on recording for channel C32

3- P1 receives Marker over C21, sets state(C21) = {a}

4- P3 receives Marker over C13, records its state (S3), sets state(C13) = {} sends Marker to P1 & P2; turns on recording for channel C23

5- P2 receives Marker over C32, sets state(C32) = {b}

6- P3 receives Marker over C23, sets state(C23) = {}

7- P1 receives Marker over C31, sets state(C31) = {}

Provable Assertion: Chandy-Lamport algo. determines a consistent cut

- 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 $e_j \rightarrow$ <p_j records its state>, then it must be true that $e_i \rightarrow$ <p_i records its state>.
  - By contradiction, suppose <p_i records its state> $\rightarrow e_i$
  - Consider the path of app messages (through other processes) that go from $e_i \rightarrow e_j$
  - Due to FIFO ordering, markers on each link in above path precede regular app messages
  - Thus, since <p_i records its state> $\rightarrow e_i$, it must be true that p_j received a marker before e_j
  - Thus e_j is not in the cut => contradiction