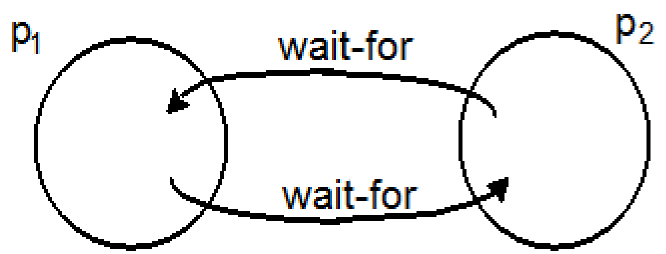
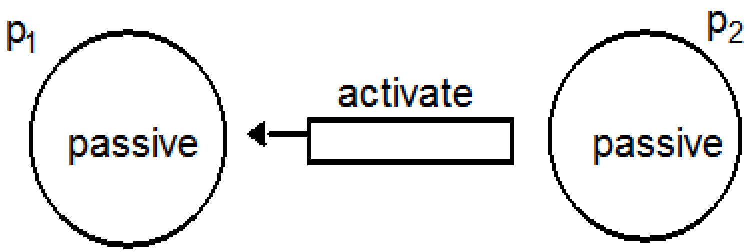
Tutorial week 5

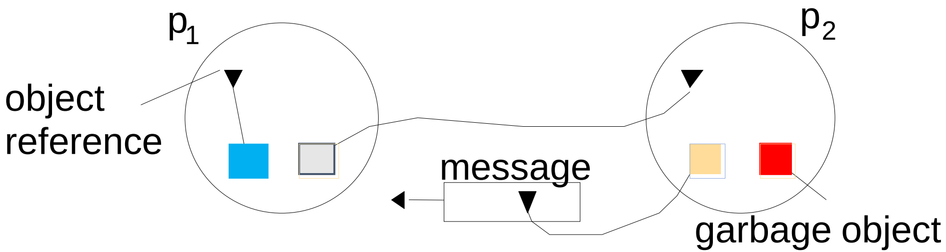
Global Snapshot

- What: Take a snapshot of the global computation
  - Global state = states of all processes + states of all communication channels

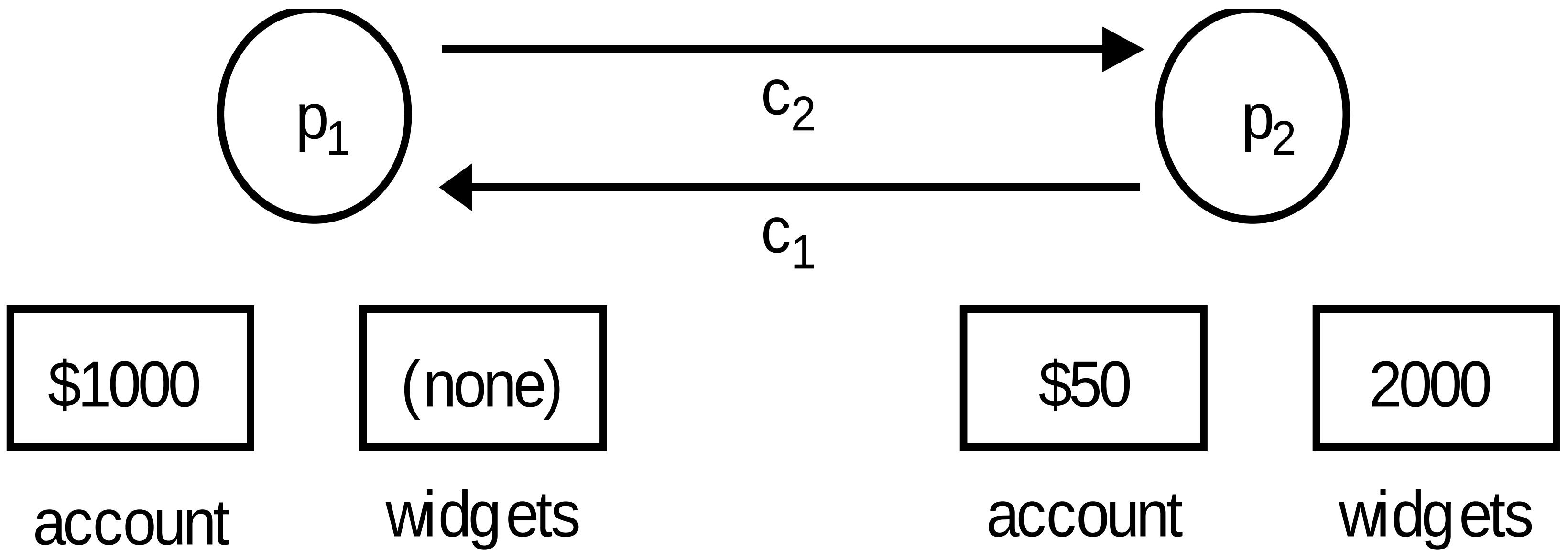
- Why?
  - Useful for debugging
  - Useful for backup/check-pointing
  - Useful for calculating global predicate: liveness and safety
  - Useful for deadlock detection
  - Useful for rollback recovery
  - Useful for termination detection
  - Useful for garbage collection



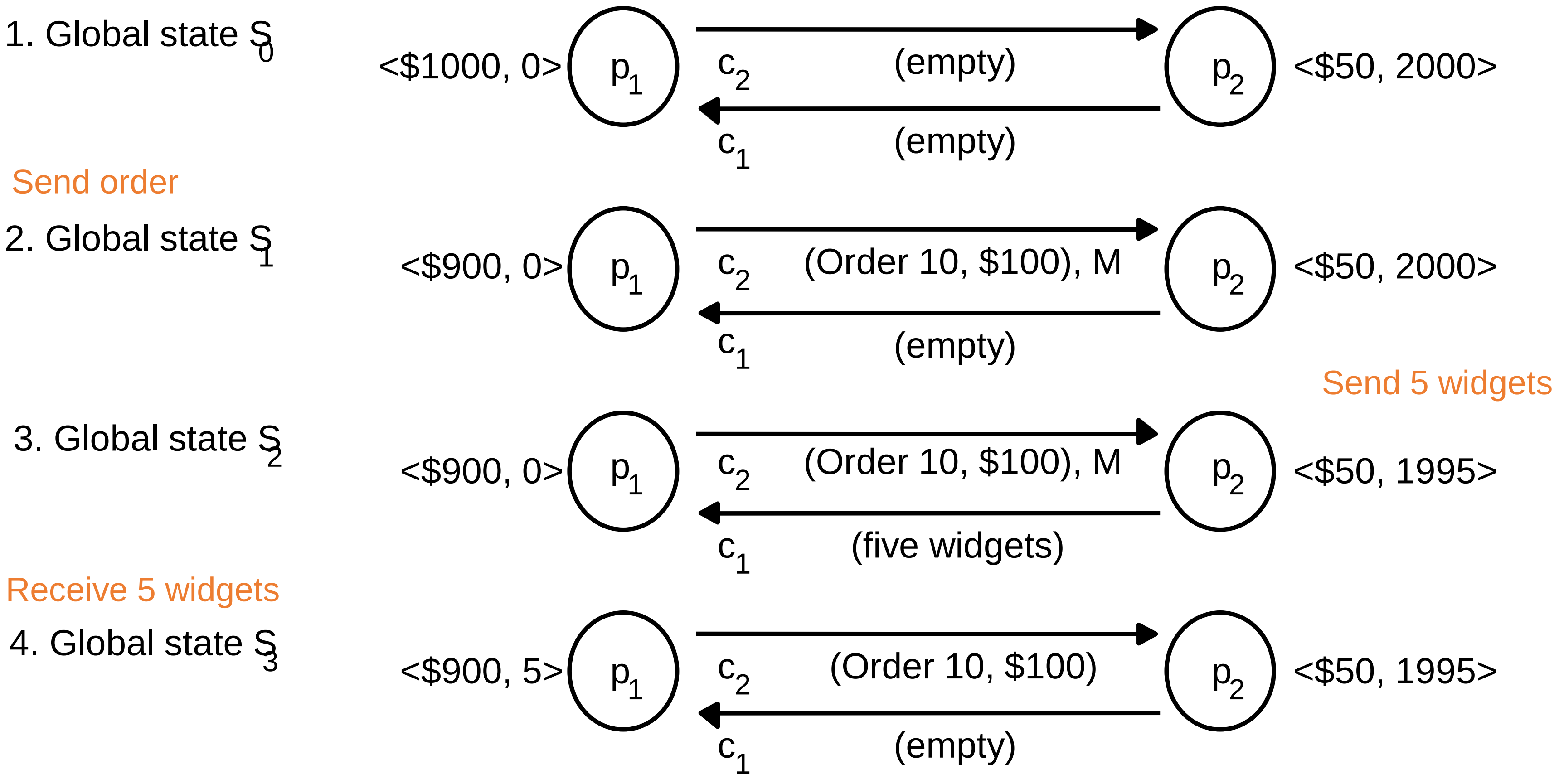




Global Snapshot: Trading Example



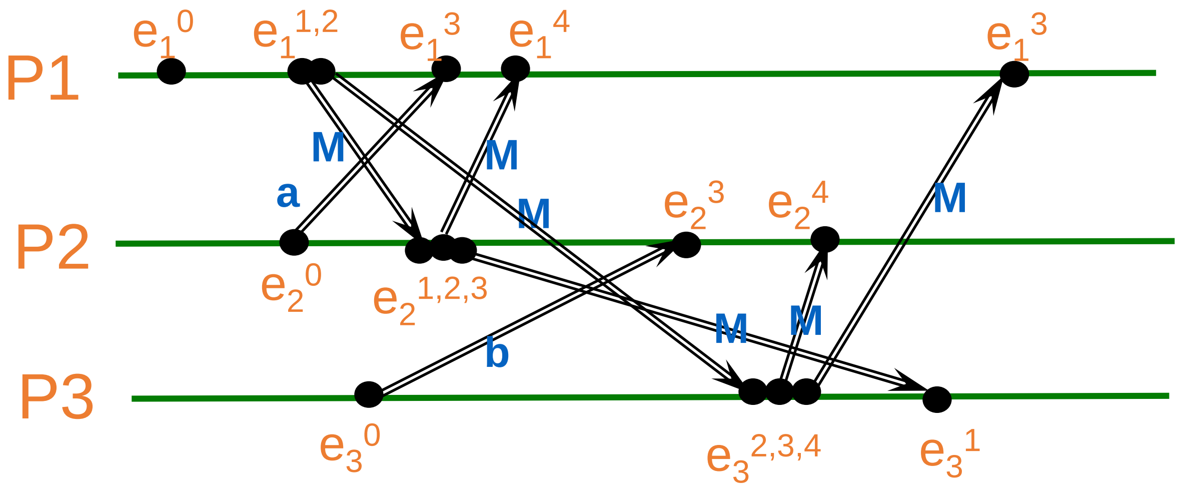
Execution of the Processes



(M = Marker Message)

Chandy-Lamport Snapshot Algorithm

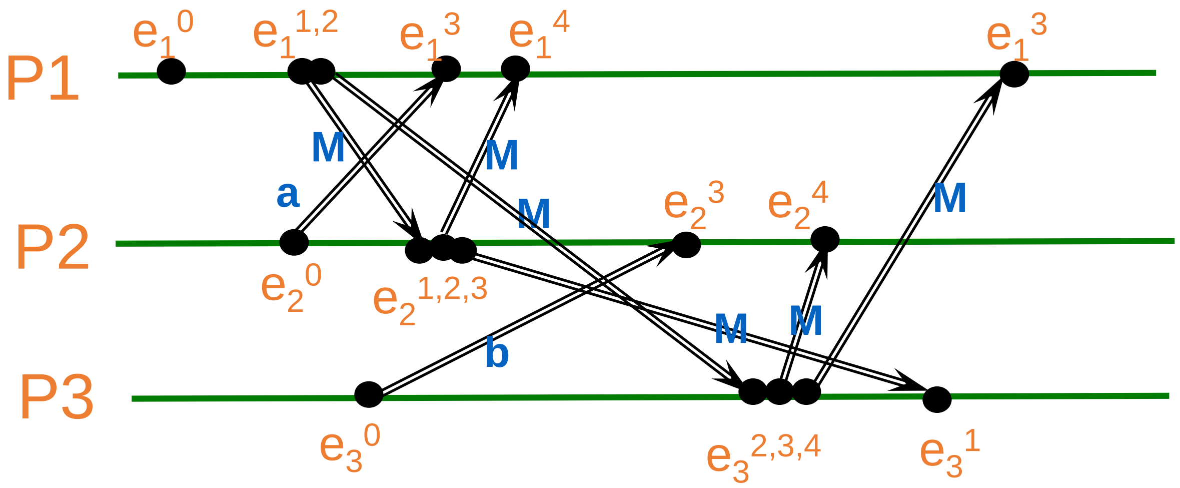
Example



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}

Example

S1, S2, S3, ch_{12} , ch_{21} , ch_{13}



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}) = {}

Chandy-Lamport Snapshot Algorithm

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

By Coulouris, Dollimore and Kindberg,

THM

How to record a consistent snapshot ??

Chandy-Lamport Snapshot Algorithm

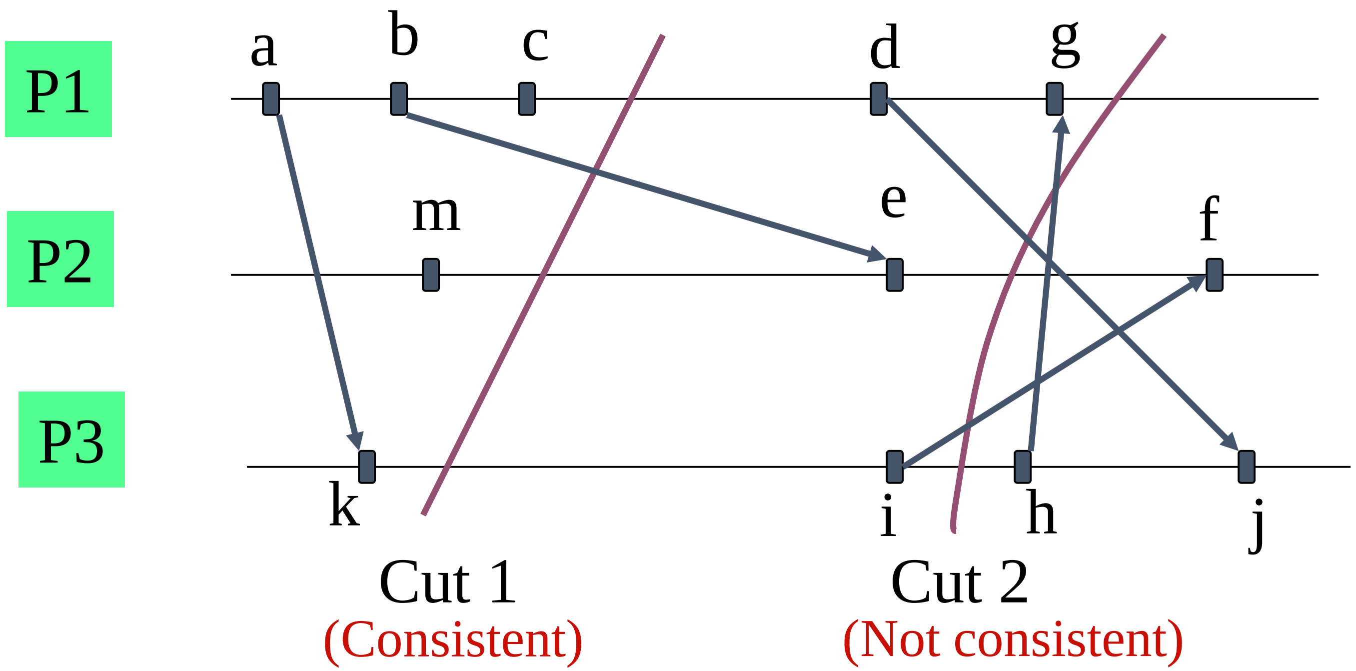
- *Assumptions:*
  - No failure, all messages arrive intact, exactly once
  - Communication channels are unidirectional and FIFO-ordered
  - There is a comm. channel between each pair of processes
  - Any process may initiate the snapshot (send “Marker”)
  - Snapshot does not interfere with normal execution

THM

Consistent cut

A cut is a set of events.

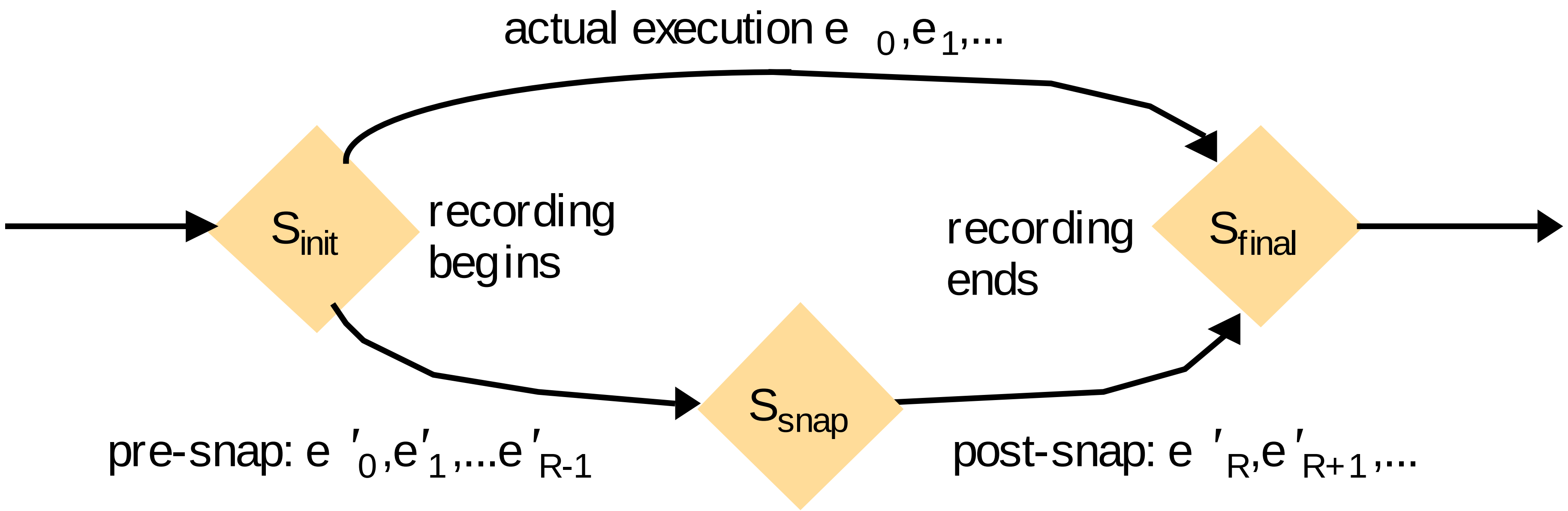
- $(\mathbf{a} \in \text{consistent cut } \mathbf{C}) \wedge (\mathbf{b} \text{ happened before } \mathbf{a}) \Rightarrow \mathbf{b} \in \mathbf{C}$



Consistent snapshot: The set of states immediately following a ***consistent cut*** forms a consistent snapshot of a distributed system.

THM

Reachability between States in Snapshot Algorithm



- The *observed state* is a feasible state that is reachable from the *initial configuration*. It may not actually be visited during a specific execution.
- The *final state* of the original computation is always reachable from the *observed state*.
- Chandy Lamport algorithm does a partial job. Each process collects a ***fragment*** of the global state, but these pieces have to be stitched together to form a global state.

THM

Run vs Linearization

- A run is a total ordering of events in *H* that is consistent with each h_i's ordering
  - E.g., $\langle e_1^0, e_1^1, e_1^2, e_1^3, e_2^0, e_2^1, e_2^2, e_3^0\ e_3^1, e_3^2 \rangle$
- A linearization is a run consistent with happens-before (→) relation in H
  - E.g., $\langle e_1^0, e_1^{1,}\ e_3^0, e_2^0, \ldots \rangle, \langle e_1^0, e_3^0, e_1^1, e_2^0, \ldots \rangle$
  - *Concurrent events are ordered arbitrarily*
  - *Linearizations pass through consistent global states*
- If there is a linearization that passes through state S and then state S', then S is said to be ***reachable*** from S