Global State and Snapshot Recording Algorithms

Dr. Rajiv Misra, Professor
Dept. of Computer Science & Engg.
Indian Institute of Technology Patna
rajivm@iitp.ac.in

Preface

Content of this Lecture:

- In this lecture, we will discuss about the Global states (i.e. consistent, inconsistent), Models of communication and Snapshot algorithm *i.e.* Chandy-Lamport algorithm to record the global snapshot.

Snapshots

Here's Snapshot: Collect at a place

Distributed Snapshot

How do you calculate a "global snapshot" in this distributed system?

What does a "global snapshot" even mean?

In the Cloud: Global Snapshot

- In a cloud each application or service is running on multiple servers
- Servers handling concurrent events and interacting with each other
- The ability to obtain a “global photograph” or “Global Snapshot” of the system is important
- Some uses of having a global picture of the system
  - ***Checkpointing***: can restart distributed application on failure
  - ***Garbage collection* of objects**: objects at servers that don’t have any other objects (at any servers) with pointers to them
  - **Deadlock detection**: Useful in database transaction systems
  - **Termination of computation**: Useful in batch computing systems

Global State: Introduction

- **Recording the global state** of a distributed system on-the-fly is an important paradigm.
- The **lack of globally shared memory, global clock and unpredictable message delays** in a distributed system make this problem non-trivial.
- This lecture first defines consistent global states and discusses issues to be addressed to compute consistent distributed snapshots.
- Then the algorithm to determine on-the-fly such snapshots is presented.

System Model

- The system consists of a collection of n processes $p_1, p_2, \ldots, p_n$ that are connected by channels.
- There are no globally shared memory and physical global clock and processes communicate by passing messages through communication channels.
- C_{ij} denotes the channel from process p_i to process p_j and its state is denoted by SC_{ij}.
- The actions performed by a process are modeled as three types of events: Internal events, the message send event and the message receive event.
- For a message m_{ij} that is sent by process p_i to process p_j, let $send(m_{ij})$ and $rec(m_{ij})$ denote its send and receive events.

System Model

- At any instant, the state of process p_i, denoted by LS_i, is a result of the sequence of all the events executed by p_i till that instant.
- For an event e and a process state LS_i, $e \in LS_i$ iff e belongs to the sequence of events that have taken process p_i to state LS_i.
- For an event e and a process state LS_i, $e \notin LS_i$ iff e does not belong to the sequence of events that have taken process p_i to state LS_i.
- For a channel C_{ij}, the following set of messages can be defined based on the local states of the processes p_i and p_j

Transit: $transit(LS_i, LS_j) = \{m_{ij} \mid send(m_{ij}) \in LS_i \quad rec(m_{ij}) \notin LS_j\}$

$\bigwedge$

Consistent Global State

- The global state of a distributed system is a collection of the local states of the processes and the channels.
- Notationally, global state GS is defined as,

$$GS = \{\cup_i LS_i, \cup_{i,j} SC_{ij}\}$$

- A global state GS is a ***consistent global state*** iff it satisfies the following two conditions :

C1: $send(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \oplus rec(m_{ij}) \in LS_j$
($\oplus$ is Ex-OR operator)

C2: $send(m_{ij}) \notin LS_i \Rightarrow m_{ij} \notin SC_{ij} \wedge rec(m_{ij}) \notin LS_j$

Global State of a Distributed System

- In the distributed execution of Figure 6.2:
- A global state GS_1 consisting of local states $\{LS_1^1, LS_2^3, LS_3^3, LS_4^2\}$ is **inconsistent** because the state of p_2 has recorded the receipt of message m_{12}, however, the state of p_1 has not recorded its send.
- On the contrary, a global state GS_2 consisting of local states $\{LS_1^2, LS_2^4, LS_3^4, LS_4^2\}$ is **consistent**; all the channels are empty except c_{21} that contains message m_{21}.

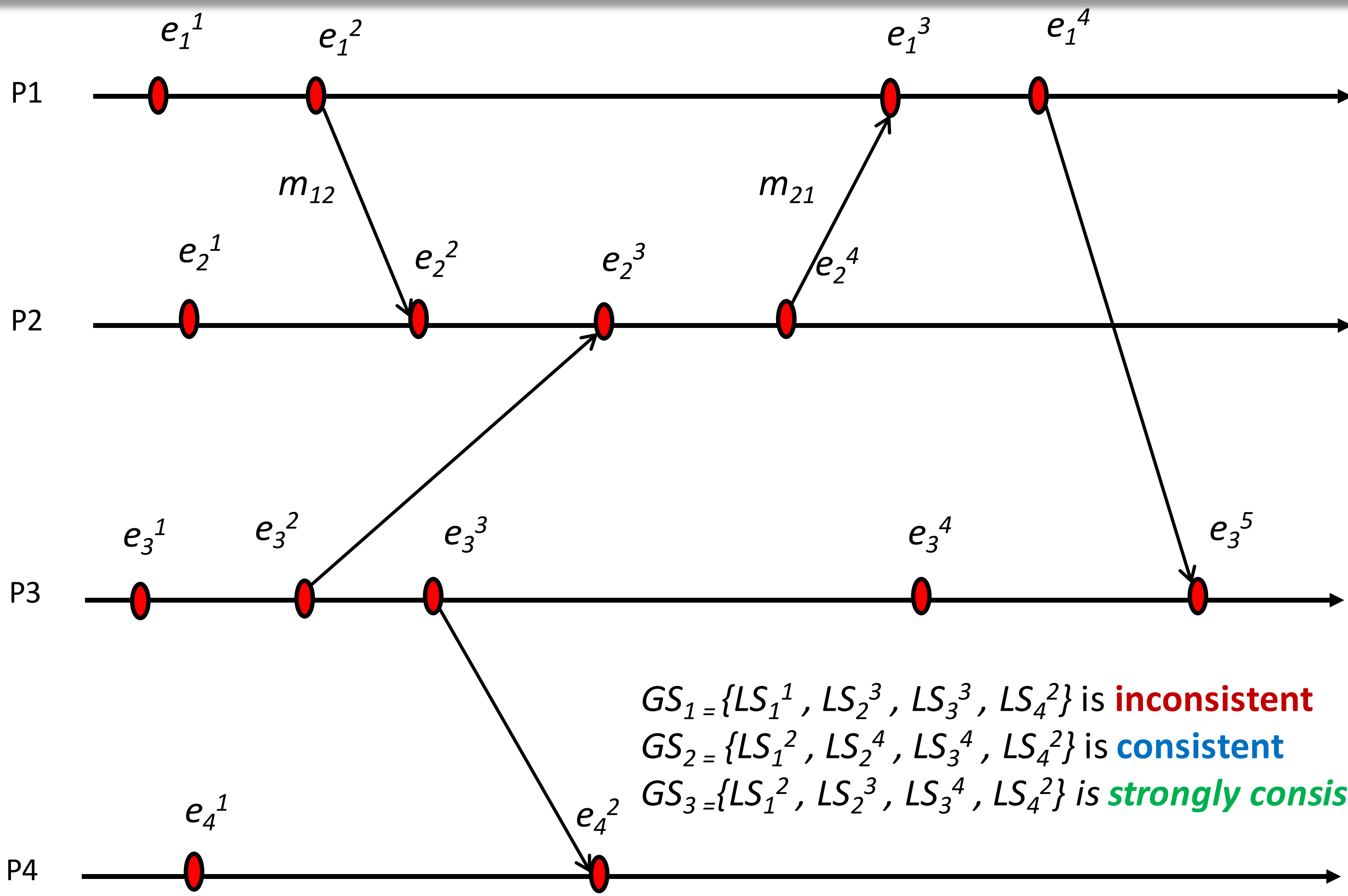
Global State of a Distributed System

- A global state GS = $\{\bigcup_i LS_i^{x_i}, \bigcup_{j,k} SC_{jk}^{y_j,z_k}\}$ is transitless iff

$$\forall i, \forall j : 1 \le i, j \le n :: SC_{jk}^{y_j,z_k} = \emptyset$$

- Thus, all channels are recorded as empty in a transitless global state. A global state is ***strongly consistent*** iff it is transitless as well as consistent. Note that in figure 6.2, the global state of local states $\{LS_1^2, LS_2^3, LS_3^4, LS_4^2\}$ *is* ***strongly consistent****.*
- Recording the global state of a distributed system is an important paradigm when one is interested in analyzing, monitoring, testing, or verifying properties of distributed applications, systems, and algorithms. Design of efficient methods for recording the global state of a distributed system is an important problem.

Example:



$GS_1 = \{LS_1^1, LS_2^3, LS_3^3, LS_4^2\}$ is **inconsistent**

$GS_2 = \{LS_1^2, LS_2^4, LS_3^4, LS_4^2\}$ is **consistent**

$GS_3 = \{LS_1^2, LS_2^3, LS_3^4, LS_4^2\}$ is ***strongly consistent***.

Figure 6.2: The space-time diagram of a distributed execution.

Issues in Recording a Global State

- The following two issues need to be addressed:

I1: How to distinguish between the messages to be recorded in the snapshot from those not to be recorded.

- -Any message that is sent by a process before recording its snapshot, must be recorded in the global snapshot (from **C1**).
- -Any message that is sent by a process after recording its snapshot, must not be recorded in the global snapshot (from **C2**).

I2: How to determine the instant when a process takes its snapshot.

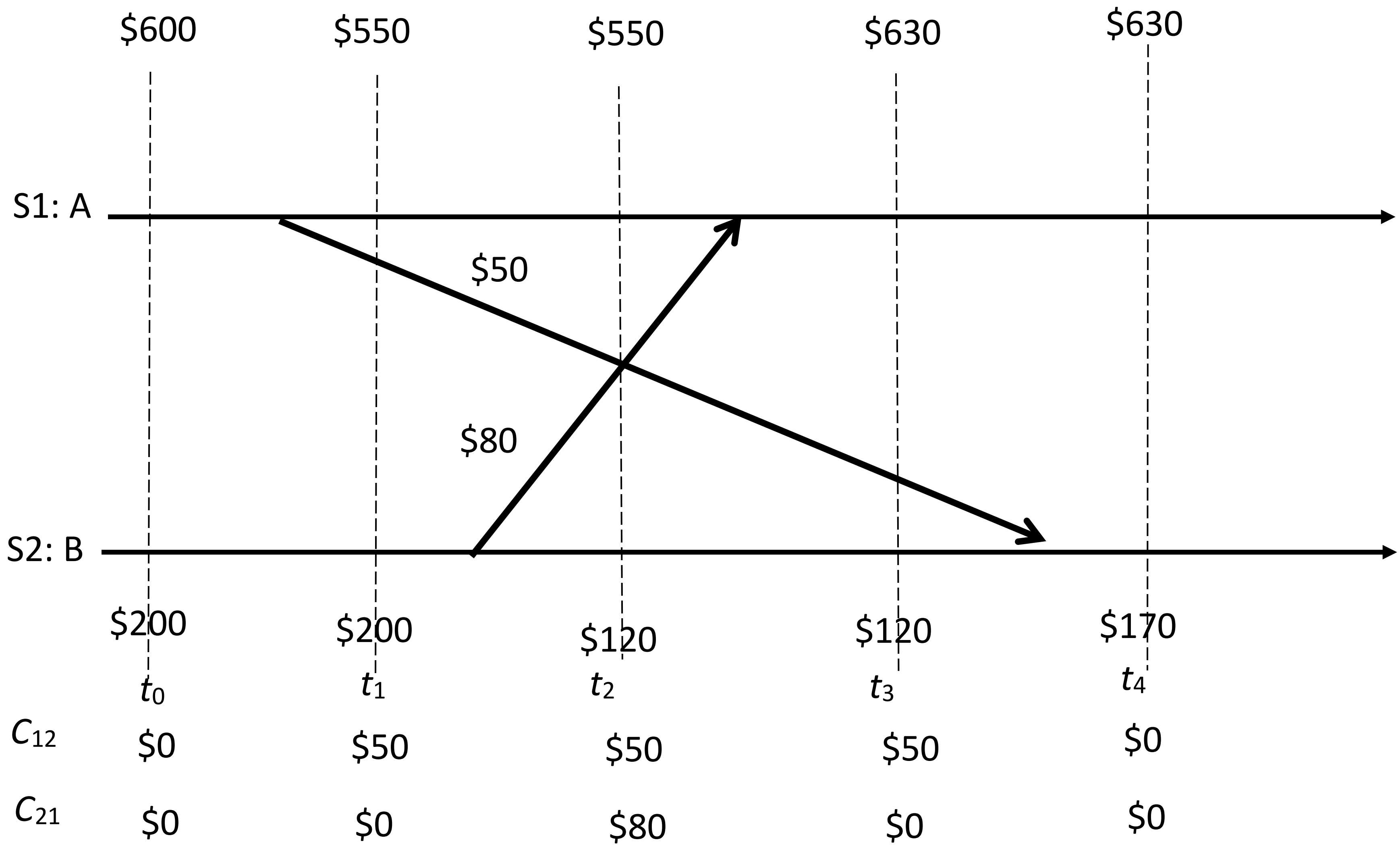
-A process p_j must record its snapshot before processing a message m_{ij} that was sent by process p_i after recording its snapshot.

Example of Money Transfer

- Let S1 and S2 be two distinct sites of a distributed system which maintain bank accounts A and B, respectively. A site refers to a process in this example. Let the communication channels from site S1 to site S2 and from site S2 to site S1 be denoted by C_{12} and C_{21}, respectively. Consider the following sequence of actions, which are also illustrated in the timing diagram of Figure 6.3:
- Time t_0: Initially, Account A=\$600, Account B=\$200, C_{12} =\$0, C_{21} =\$0.
- Time t_1: Site S1 initiates a transfer of \$50 from Account A to Account B.
- Account A is decremented by \$50 to \$550 and a request for \$50 credit to Account B is sent on Channel C_{12} to site S2. Account A=\$550, Account B=\$200, C_{12} =\$50, C_{21} =\$0.

- Time t_2 : Site S2 initiates a transfer of \$80 from Account B to Account A.
- Account B is decremented by \$80 to \$120 and a request for \$80 credit to Account A is sent on Channel C_{21} to site S1. Account A=\$550, Account B=\$120, C_{12} =\$50, C_{21} =\$80.
- Time t_3: Site S1 receives the message for a \$80 credit to Account A and updates Account A.
  Account A=\$630, Account B=\$120, C_{12} =\$50, C_{21} =\$0.
- Time t_4: Site S2 receives the message for a \$50 credit to Account B and updates Account B.
  Account A=\$630, Account B=\$170, C_{12}=\$0, C_{21} =\$0.

***T*$_3$: Site S1 receives the message for a $80 credit to Account A and updates**



***T*$_4$: Site S2 receives the message for a $50 credit to Account B and updates Account B**

- Suppose the local state of Account A is recorded at time t_0 to show \$600 and the local state of Account B and channels C_{12} and C_{21} are recorded at time t_2 to show \$120, \$50, and \$80, respectively. Then the recorded global state shows \$850 in the system. An extra \$50 appears in the system.
- **The reason for the inconsistency** is that Account A's state was recorded before the \$50 transfer to Account B using channel C_{12} was initiated, whereas channel C_{12}'s state was recorded after the \$50 transfer was initiated.
- This simple example shows that recording a consistent global state of a distributed system is not a trivial task. Recording activities of individual components must be coordinated appropriately.

Model of Communication

- Recall, there are three models of communication: FIFO, non-FIFO, and Co.

- In **FIFO model**, each channel acts as a first-in first-out message queue and thus, message ordering is preserved by a channel.

- In **non-FIFO model**, a channel acts like a set in which the sender process adds messages and the receiver process removes messages from it in a random order.

- A system that supports **causal delivery** of messages satisfies the following property: "For any two messages m_{ij} and m_{kj},

  if $send(m_{ij}) \rightarrow send(m_{kj})$, then $rec(m_{ij}) \rightarrow rec(m_{kj})$"

Snapshot algorithm for FIFO channels

Chandy-Lamport algorithm:

- The **Chandy-Lamport** algorithm uses a **control message**, called a ***marker*** whose role in a **FIFO system** is to separate messages in the channels.
- After a site has recorded its snapshot, it sends a ***marker***, along all of its outgoing channels before sending out any more messages.
- A marker separates the messages in the channel into those to be included in the snapshot from those not to be recorded in the snapshot.
- A process must record its snapshot no later than when it receives a marker on any of its incoming channels.

Chandy-Lamport Algorithm

- The algorithm can be initiated by any process by executing the "**Marker Sending Rule**" by which it records its local state and sends a marker on each outgoing channel.
- A process executes the "**Marker Receiving Rule**" on receiving a marker. If the process has not yet recorded its local state, it records the state of the channel on which the marker is received as empty and executes the "Marker Sending Rule" to record its local state.
- The algorithm terminates after each process has received a marker on all of its incoming channels.
- All the local snapshots get disseminated to all other processes and all the processes can determine the global state.

Chandy-Lamport Algorithm

Marker Sending Rule for process *i*

1) Process *i* records its state.

2) For each outgoing channel C on which a marker has not been sent, *i* sends a marker along C before *i* sends further messages along C.

Marker Receiving Rule for process *j*

On receiving a marker along channel C:

if *j* has not recorded its state **then**

Record the state of C as the empty set

Follow the "Marker Sending Rule"

else

Record the state of C as the set of messages received along C after *j*'s state was recorded and before *j* received the marker along C

Properties of the recorded global state

- The recorded global state may not correspond to any of the global states that occurred during the computation.
- Consider two possible executions of the snapshot algorithm (shown in Figure 6.4) for the previous money transfer example .

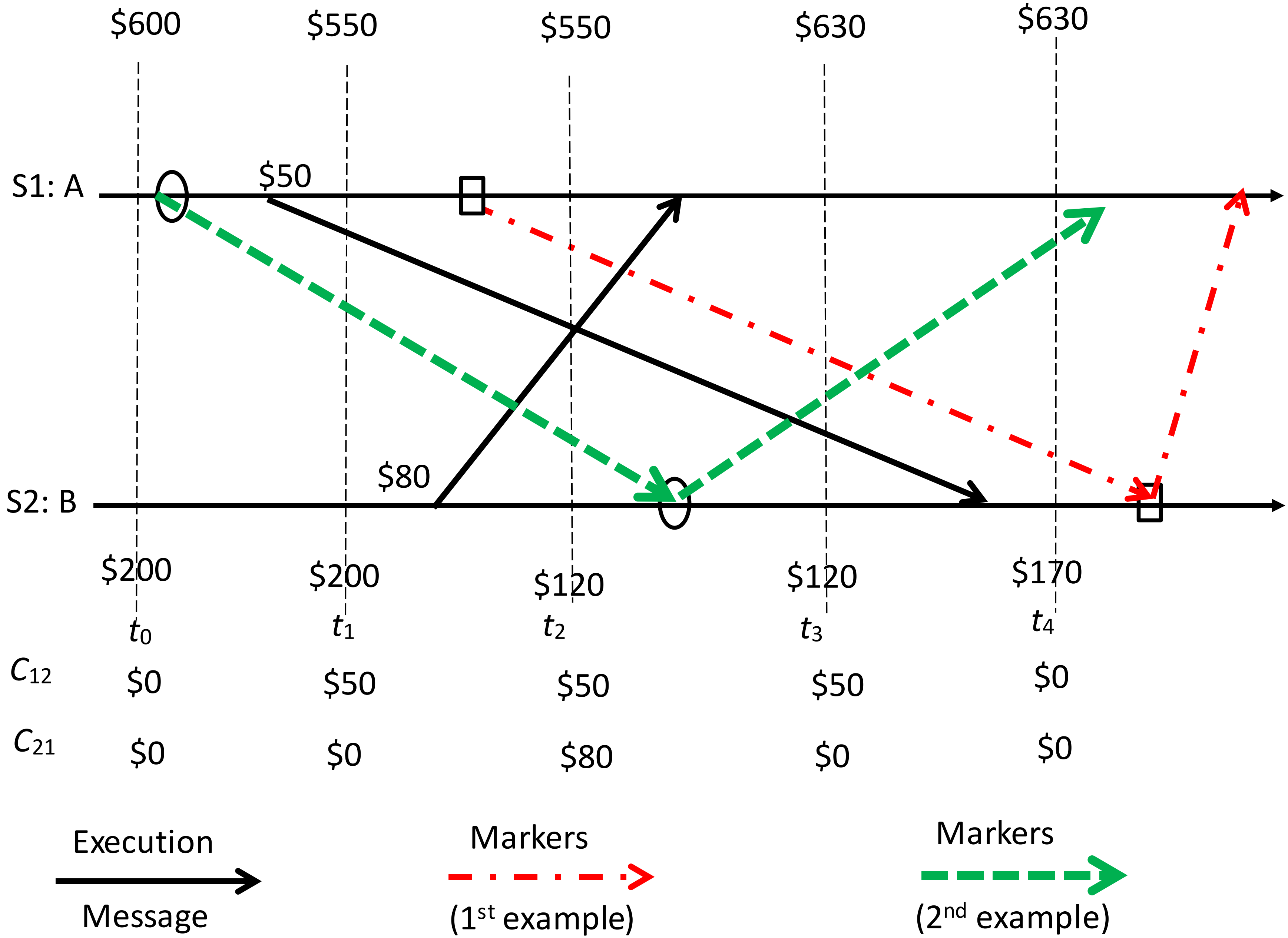


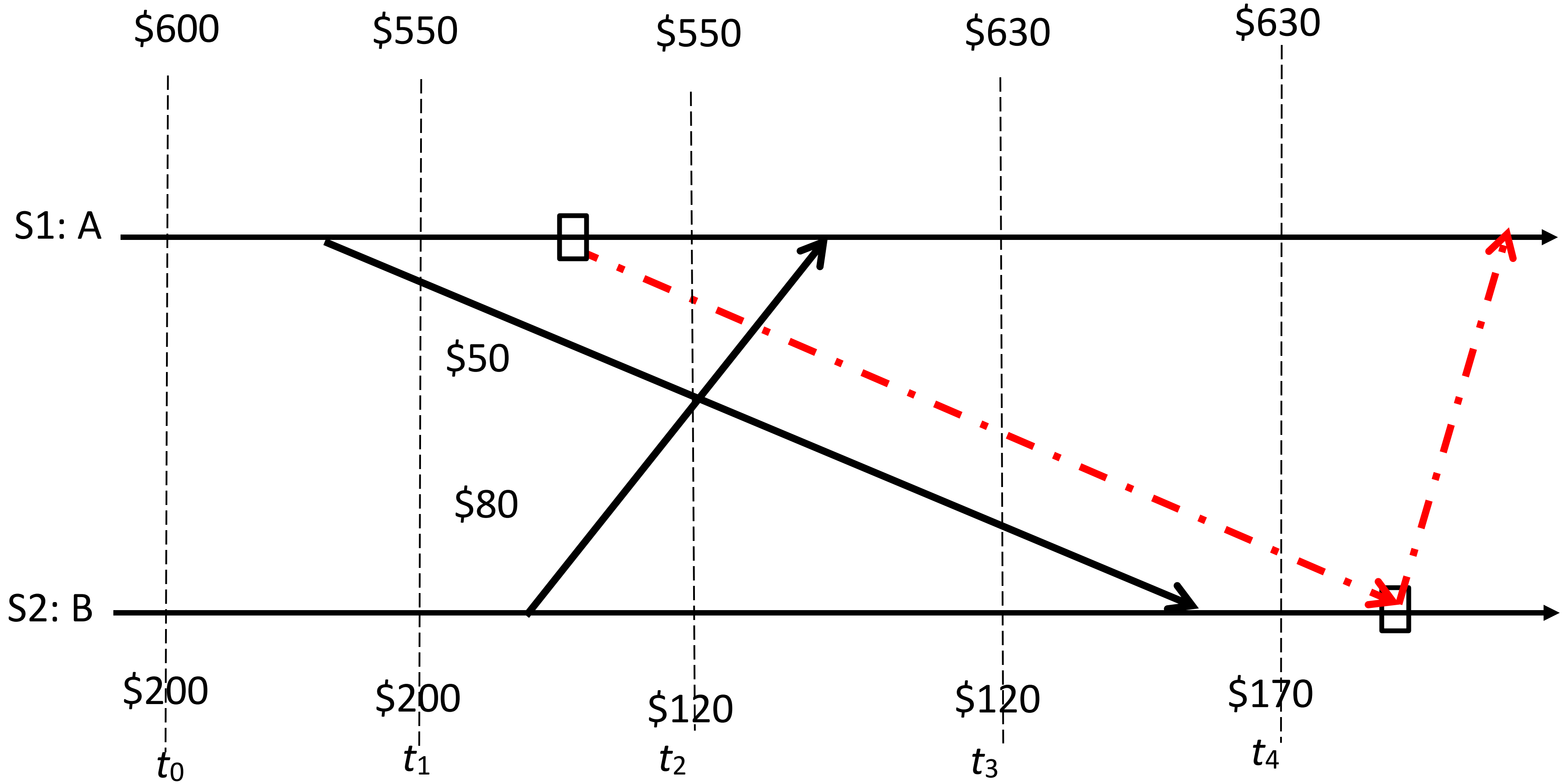
Figure 6.4: Timing diagram of two possible executions of the banking example

Properties of the recorded global state

1. (Markers shown using red dashed-and-dotted arrows.)

Let site S1 initiate the algorithm just after t_1. Site S1 records its local state (account A=\$550) and sends a marker to site S2. The marker is received by site S2 after t_4. When site S2 receives the marker, it records its local state (account B=\$170), the state of channel C_{12} as \$0, and sends a marker along channel C_{21}. When site S1 receives this marker, it records the state of channel C_{21} as \$80. The \$800 amount in the system is conserved in the recorded global state,

$$A = \$550,\ B = \$170,\ C_{12} = \$0,\ C_{21} = \$80$$



A = \$550 **B = \$170** **C_{12} = \$0** **C_{21} = \$80**

The \$800 amount in the system is conserved in the recorded global state

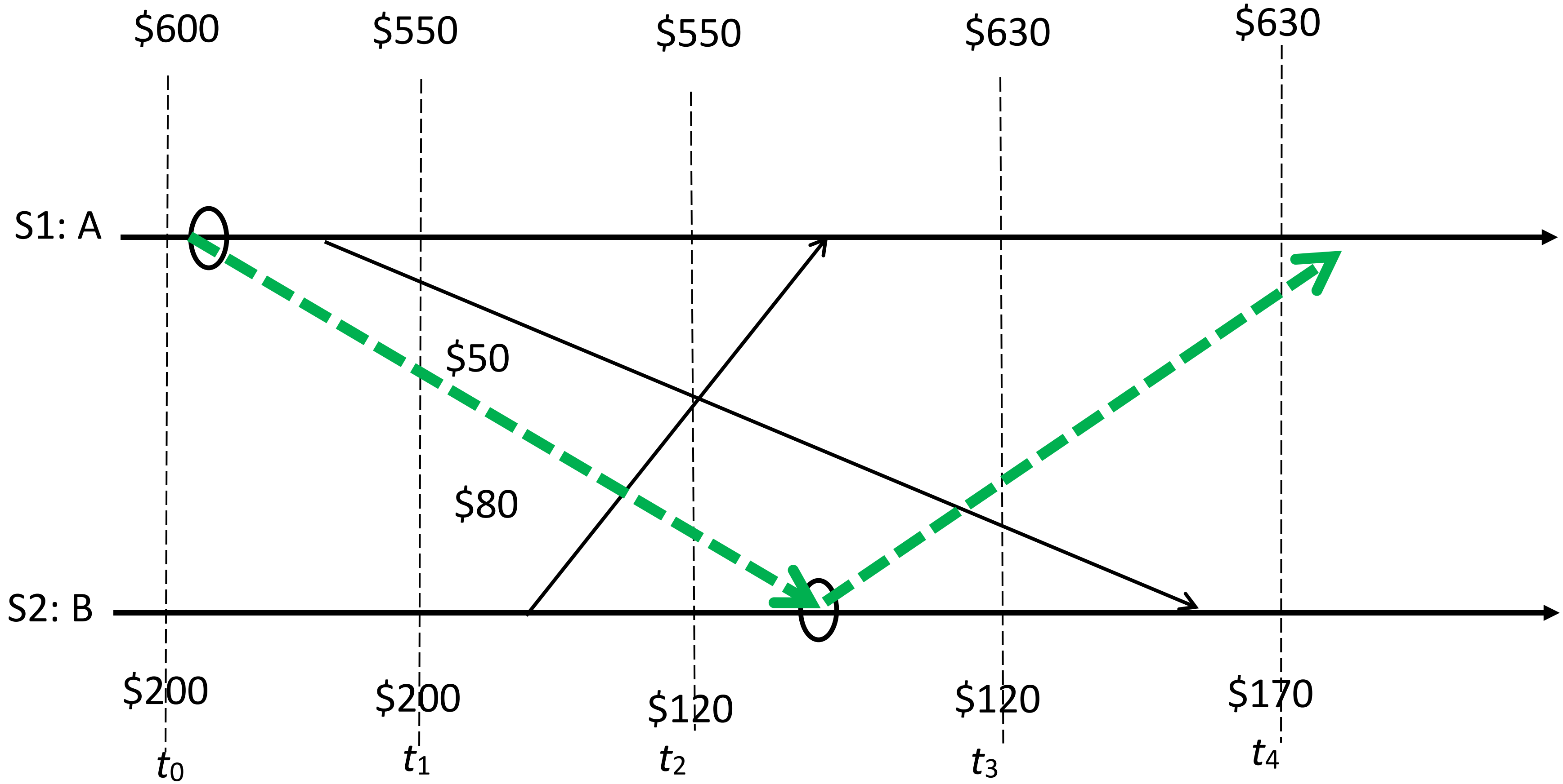
Figure 6.4: Timing diagram of two possible executions of the banking example

Properties of the recorded global state

2. (Markers shown using green dotted arrows.)

Let site S1 initiate the algorithm just after t_0 and before sending the $50 for S2. Site S1 records its local state (account A = $600) and sends a marker to site S2. The marker is received by site S2 between t_2 and t_3. When site S2 receives the marker, it records its local state (account B = $120), the state of channel C_{12} as $0, and sends a marker along channel C_{21}. When site S1 receives this marker, it records the state of channel C_{21} as $80. The $800 amount in the system is conserved in the recorded global state,

A = $600, B = $120, C_{12} = $0, C_{21} = $80



A = $600 **B = $120** **$C_{12}$ = $0** **$C_{21}$ = $80**

The $800 amount in the system is conserved in the recorded global state

Figure 6.4: Timing diagram of two possible executions of the banking example

Properties of the recorded global state

- In both these possible runs of the algorithm, the recorded global states never occurred in the execution.
- This happens because a process can change its state asynchronously before the markers it sent are received by other sites and the other sites record their states.
  - But the system could have passed through the recorded global states in some equivalent executions.
  - The recorded global state is a valid state in an equivalent execution and if a stable property (i.e., a property that persists) holds in the system before the snapshot algorithm begins, it holds in the recorded global snapshot.
- Therefore, a recorded global state is useful in detecting stable properties.

Conclusion

- Recording global state of a distributed system is an important paradigm in the design of the distributed systems and the design of efficient methods of recording the global state is an important issue.

- This lecture first discussed a formal definition of the **global state of a distributed system and issues** related to its capture; then we have discussed the **Chandy-Lamport Algorithm** to record a snapshot of a distributed system.