BITS Pilani
Hyderabad Campus

# Distributed Computing (CS 3)

## Global State & Snapshot Recording Algorithms


Prof. Geetha
Associate Professor, Dept. of Computer Sc. & Information Systems
BITS Pilani Hyderabad Campus
geetha@hyderabad.bits-pilani.ac.in

# Introduction – global state and snapshot

- Recording the global state of a distributed system on-the-fly is required for analyzing, testing, or verifying properties associated with distributed executions
- problems in recording global state
  - lack of a globally shared memory
  - lack of a global clock
  - message transmission is asynchronous
  - message transfer delays are finite but unpredictable
- problem is non-trivial

# Global state and snapshot contd..

- Every distributed system component has a local state
- state of a process is characterized by
  - state of its local memory
  - history of process activity
- channel state is characterized by the set of messages sent along the channel less the set of messages received along the channel
- Global state of a distributed system is a collection of the local states of its components
- Snapshot is the state of a system at a particular point in time

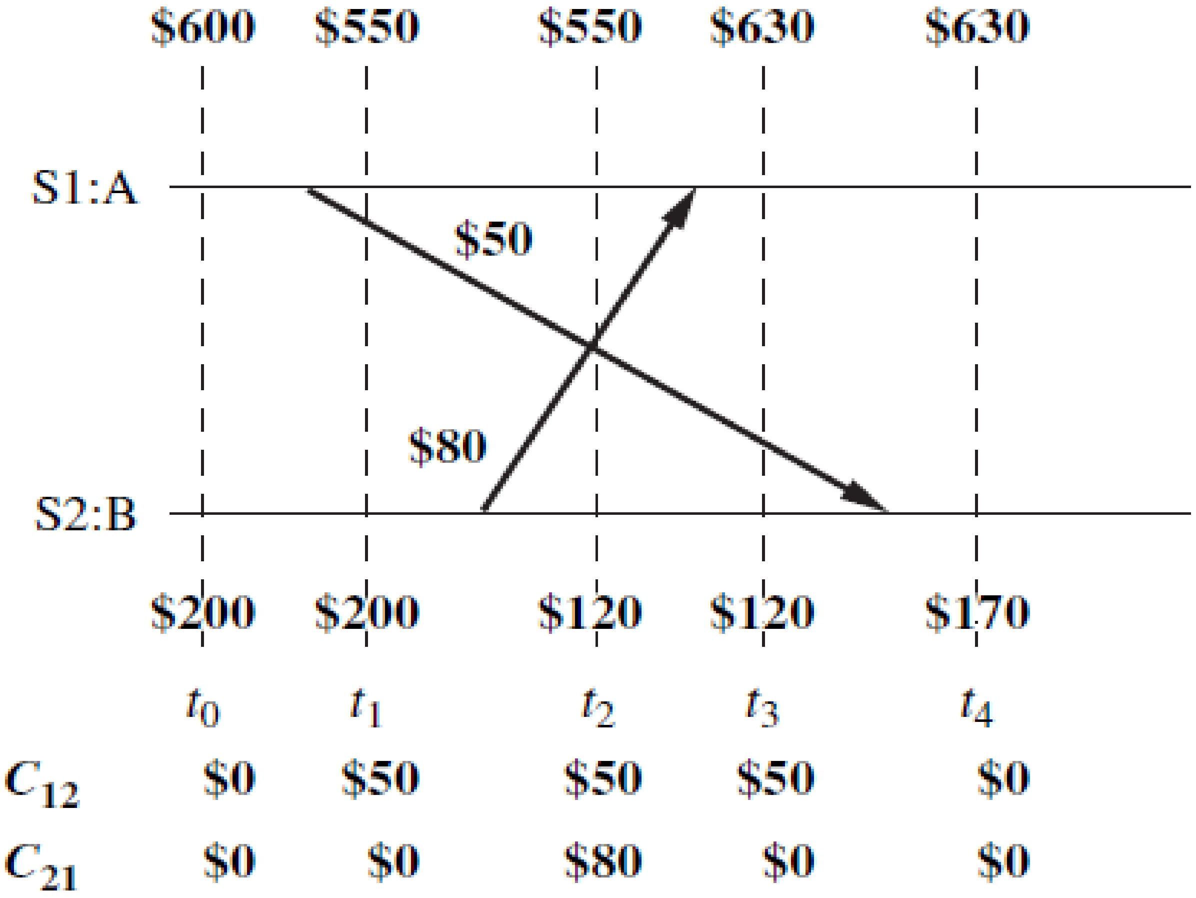
# Global state and snapshot contd..

- **benefit of shared memory** - up-to-date state of the entire system is available to the processes sharing the memory
- absence of shared memory requires ways of getting a coherent and complete view of the system based on the local states of individual processes
- meaningful global snapshot can be obtained
  - if the components of the distributed system record their local states at the same time
  - requires local clocks at processes to be perfectly synchronized
  - existence of global system clock that could be instantaneously read by the processes

# Global state and snapshot contd..

- technologically infeasible to have perfectly synchronized clocks at various sites
- clocks are bound to drift
- reading time from a single common clock maintained at one process will not work
- indeterminate transmission delays during read operation cause processes to identify various physical instants as the same time
- collection of local state observations will be made at different times
- may not be meaningful

# global state and snapshot - a challenging scanario



# System Model

- system consists of a collection of n processes, $p_1$, $p_2$,…, $p_n$ connected by channels
- no globally shared memory
- processes communicate solely by passing messages
- no physical global clock in the system
- message send and receive are asynchronous
- message delivery is reliable, occurs within finite time but has arbitrary time delay

# System Model contd..

- system can be
  - represented as a directed graph
  - vertices represent processes
  - edges represent unidirectional communication channels
- both processes & channels have states
- process state consists of contents of
  - processor registers
  - stacks
  - local memory

# System Model contd..

- process state is highly dependent on the local context of the distributed application
- $C_{ij}$: channel from process $p_i$ to process $p_j$
- $SC_{ij}$:
  - state of channel $C_{ij}$
  - consists of in-transit messages
- 3 types of events
  - internal events
  - message send events
  - message receive events

# System Model contd..

- send($m_{ij}$): send event of message $m_{ij}$ from $p_i$ to $p_j$
- rec($m_{ij}$): receive event of message $m_{ij}$ from $p_i$ to $p_j$
- occurrence of events
  - changes the states of respective processes and channels
  - causes transitions in global system state
- internal event: changes the state of the process at which it occurs
- send event changes:
  - state of the process that sends the message
  - state of the channel on which the message is sent
- events at a process are linearly ordered by their order of occurrence

# System Model contd..

- receive event:
  - changes state of the receiving process
  - state of the channel on which the message is received
- $LS_i$: state of process $p_i$
- at an instant t, $LS_i$: state of $p_i$ as a result of the sequence of events executed by $p_i$ up to t
- an event $e \in LS_i$ iff e belongs to the sequence of events that have taken $p_i$ to $LS_i$
- $e \notin LS_i$ iff e does not belong to the sequence of events that have taken $p_i$ to $LS_i$

# System Model contd..

- for a channel $C_{ij}$ , in-transit messages are:
  - $transit(LS_i, LS_j) = \{m_{ij} \mid send(m_{ij}) \in LS_i \wedge rec(m_{ij}) \notin LS_j\}$
- if a snapshot recording algorithm records the states of $p_i$ and $p_j$ as $LS_i$ and $LS_j$ , respectively,
  - it must record the state of channel $C_{ij}$ as $transit(LS_i, LS_j)$

# System Model contd..

- Several models of communication among processes exist
- FIFO model:
  - each channel acts as a first-in first-out message queue
  - message ordering is preserved by a channel
- non-FIFO model:
  - channel acts like a set
  - sender process adds messages to the channel in a random order
  - receiver process removes messages from the channel in a random order

# A Consistent Global State

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

  - **C1:** $send(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \oplus rec(m_{ij}) \in LS_j$ ($\oplus$: XOR)

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

# A Consistent Global State

- **Condition C1 states the law of conservation of messages**:
  - every message $m_{ij}$ that is recorded as sent in the local state of sender $p_i$ must be captured
    - in the state of the channel $C_{ij}$
    - or in the collected local state of the receiver $p_j$
- **Condition C2 states that:**
  - in the collected global state, for every effect, its cause must be present
  - if a message $m_{ij}$ is not recorded as sent in the local state of $p_i$, then it must neither be present in the state of the channel $C_{ij}$ nor in the collected local state of the receiver $p_j$

# Chandy–Lamport Approach

- uses a control message called **marker**
- after a process has recorded its snapshot, it sends a marker along all of its outgoing channels before sending out any more messages
- all messages that follow a marker on a channel have been sent by the sender after it took its snapshot
- channels are FIFO
- marker separates the messages in the channel into those to be included in the snapshot from those not to be recorded
- a process must record its snapshot no later than when it receives a marker on any of its incoming channels

# Chandy–Lamport Algorithm

***Marker sending rule for process $p_i$***

(1) Process $p_i$ records its state.

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

***Marker receiving rule for process $p_j$***

On receiving a marker along channel C:

**if** $p_j$ has not recorded its state **then**

Record the state of C as the empty set

Execute the "marker sending rule"

**else**

Record the state of C as the set of messages received along C after $p_{j's}$ state was recorded and before $p_j$ received the marker along C

# Chandy–Lamport Algorithmic approach

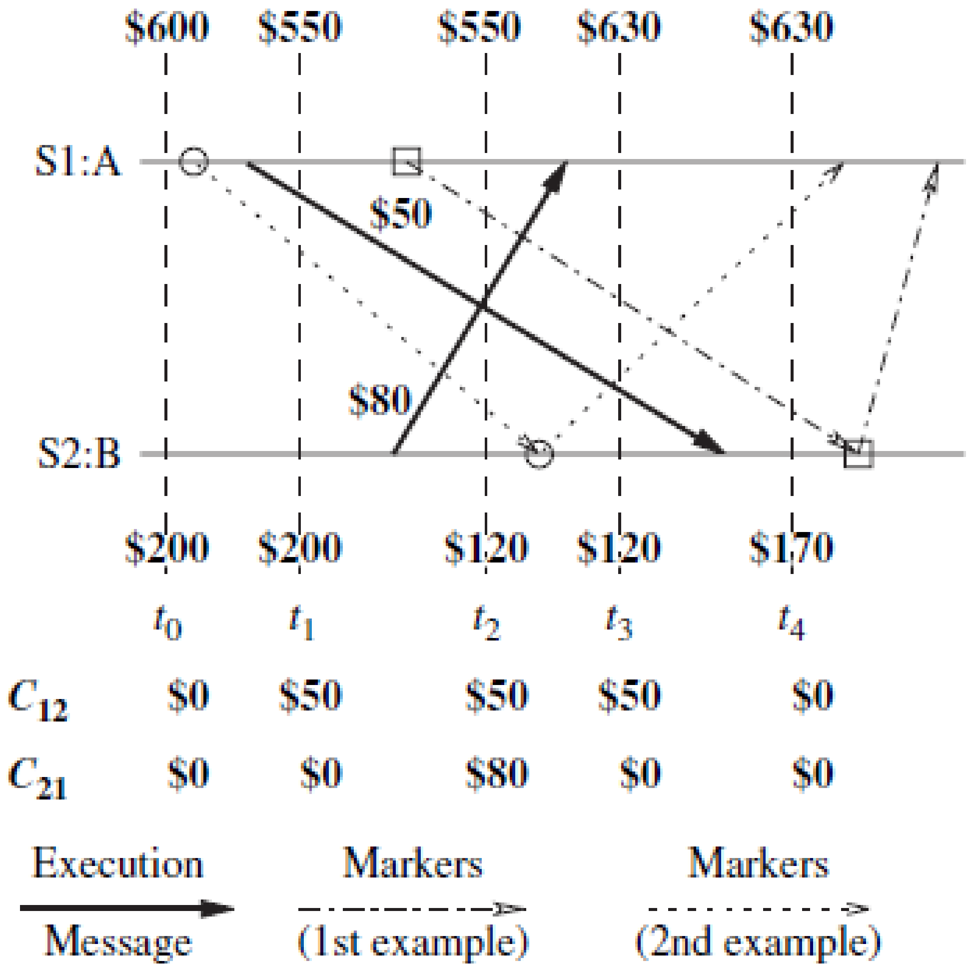
***Putting together recorded snapshots***

- global snapshot creation at initiator:
  - each process can send its local snapshot to the initiator of the algorithm
- global snapshot creation at all processes:
  - each process sends the information it records along all outgoing channels
  - each process receiving such information for the first time propagates it along its outgoing channels
  - all the processes can determine the global state

# Chandy–Lamport Algorithm contd..

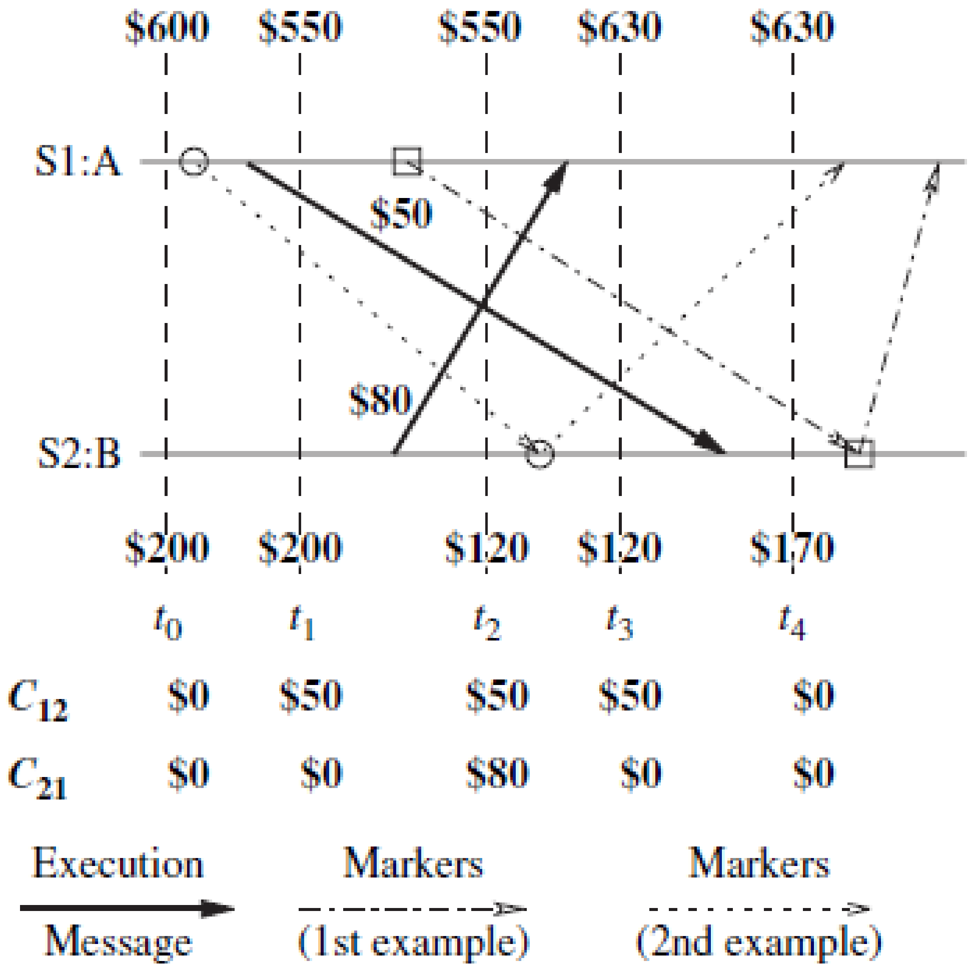
- algorithm can be initiated by any process by executing the marker sending rule
- **termination criterion** - each process has received a marker on all of its incoming channels
- if multiple processes initiate the algorithm concurrently
  - each initiation needs to be distinguished by using unique markers

# Chandy-Lamport Algorithm-scenario1




- Markers shown by dashed-and-dotted arrows
- S1 initiates the algorithm just after $t_1$
- S1 records its local state (account A=$550) and sends a marker to S2
- marker is received by S2 after $t_4$
- when S2 receives the marker, it records its local state (account B=$170), state $C_{12}$ as $0, and sends a marker along $C_{21}$
- when S1 receives this marker, it records the state of $C_{21}$ as $80
- $800 amount in the system is conserved in the recorded global state
  - A = $550, B = $170, $C_{12}$ = $0, $C_{21}$ = $80

# Chandy-Lamport Algorithm-scenario2




- Markers shown using dotted arrows
- S1 initiates the algorithm just after $t_0$ and before sending the $50 for S2
- S1 records its local state (account A = $600) and sends a marker to S2
- marker is received by S2 between $t_2$ and $t_3$
- when S2 receives the marker, it records its local state (account B = $120), state of $C_{12}$ as $0, and sends a marker along $C_{21}$
- when S1 receives this marker, it records the state of $C_{21}$ as $80
- $800 amount in the system is conserved in the recorded global state
    - A = $600, B = $120, $C_{12}$ = $0, $C_{21}$ = $80

# Snapshot Algorithms for Non-FIFO Channels

- FIFO system
  - ensures that all messages sent after a marker on a channel will be delivered after the marker

- in a non-FIFO system
  - a marker cannot be used to differentiate messages into those to be recorded in the global state from those not to be recorded in the global state
- non-FIFO algorithm by Lai and Yang
  - use message piggybacking to distinguish computation messages sent after the marker from those sent before the marker

# Lai–Yang Algorithm

- **fulfills the role of marker using a coloring scheme**
- **Coloring Scheme:**
- every process is initially white
- process turns red while taking a snapshot
- equivalent of the “marker sending rule” is executed when a process turns red
- every message sent by a white process is colored white
  - a white message is a message that was sent before the sender of that message recorded its local snapshot

# Lai–Yang Algorithm

- **Coloring Scheme:**

- every message sent by a red process is colored red
  - a red message is a message that was sent after the sender of that message recorded its local snapshot

- every white process takes its snapshot no later than the instant it receives a red message

# Lai–Yang Algorithm contd..

- **Coloring Scheme:**
- when a white process receives a red message, it records its local snapshot before processing the message
- ensures that
  - no message sent by a process after recording its local snapshot is processed by the destination process before the destination records its local snapshot
- an explicit marker message is not required
- marker is piggybacked on computation messages using a coloring scheme

# Lai–Yang Algorithm contd..

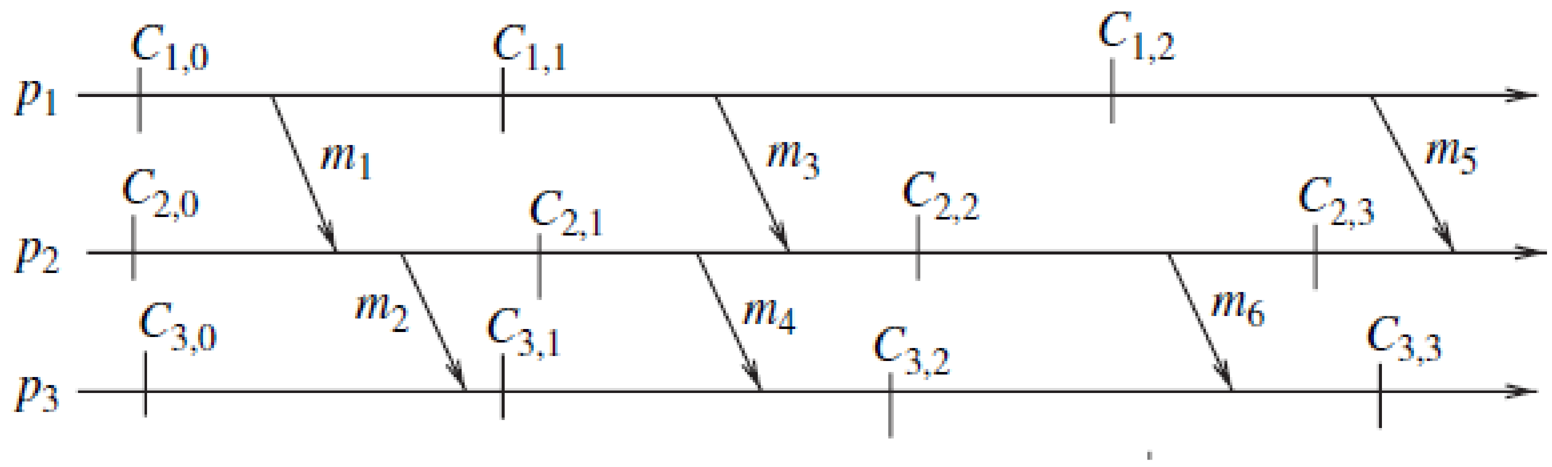
- Lai–Yang algorithm fulfills this role of the marker for channel state computation as follows:
  - every white process records a history of all white messages sent or received by it along each channel
  - when a process turns red, it sends these histories along with its snapshot to the initiator process that collects the global snapshot
  - initiator process evaluates transit($LS_i$, $LS_j$) to compute the state of a channel $C_{ij}$ as:
    - $SC_{ij}$ = {white messages sent by $p_i$ on $C_{ij}$} − {white messages received by $p_j$ on $C_{ij}$}
    - $= \{m_{ij} \mid send(m_{ij}) \in LS_i\} - \{m_{ij} \mid rec(m_{ij}) \in LS_j\}$

# Necessary and sufficient conditions for consistent global snapshots

- local checkpoint – saved intermediate state of a process during its execution
- global snapshot –
  - set of local checkpoints one from each process
- $C_{p,i}$ - $i^{th}$ ($i \geq 0$) checkpoint of process $p_p$ , assigned the sequence number i
- $i^{th}$ checkpoint interval of $p_p$ - all computation performed between $(i-1)^{th}$ and $i^{th}$ checkpoints (and includes $(i-1)^{th}$ checkpoint but not $i^{th}$).

# Necessary and sufficient conditions for consistent global snapshots

- a causal path exists between checkpoints $C_{i,x}$ and $C_{j,y}$ if a sequence of messages exists from $C_{i,x}$ to $C_{j,y}$ such that each message is sent after the previous one in the sequence is received
- a zigzag path between two checkpoints is a causal path, however, allows a message to be sent before the previous one in the path is received



# Necessary and sufficient conditions for consistent global snapshots



- **necessary condition for consistent snapshot** - absence of causal path between checkpoints in a snapshot
- **necessary and sufficient conditions for consistent snapshot** - absence of zigzag path between checkpoints in a snapshot

# Recap Quiz

1. Which of the following is not a type of event in distributed computing environments?
(a) Internal (b) external (c) message send (d) message receive

2. If the message ordering is preserved in the distributed computing environment, then this system model is called
(a)non-FIFO (b) LIFO (c) queue (d) FIFO

3. The control message used in Chandy-Lamport algorithm is called
(a) Master (b) snapshot (c) marker (d) checkpoint

4. Message piggybacking is used in snapshot algorithms for _____ channels
(a)non-FIFO (b) LIFO (c) queue (d) FIFO

5. The Lai-Yang algorithm for non-FIFO channels uses _______ instead of a marker
(a) Checkpointing (b) colouring (c) creating (d) initiating

# Recap Quiz - key

| Q1 | Q2 | Q3 | Q4 | Q5 |
|---|---|---|---|---|
| b | d | c | a | c |

# Reference


- Ajay D. Kshemkalyani, and Mukesh Singhal, Chapter 4, “Distributed Computing: Principles, Algorithms, and Systems”, Cambridge University Press, 2008.