M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Distributed Systems
Course Code: CSE20
Credits: 3:0:0:1

Term: September – December 2020

Faculty:
Sini Anna Alex

Mattern’s Algorithm for Non-FIFO channels

Mattern's algorithm is based on vector clocks and assumes a single initiator process and works as follows:

1. The initiator "ticks" its local clock and selects a future vector time s at which it would like a global snapshot to be recorded. It then broadcasts this time s and freezes all activity until it receives all acknowledgements of the receipt of this broadcast.
2. When a process receives the broadcast, it remembers the value s and returns an acknowledgement to the initiator.
3. After having received an acknowledgement from every process, the initiator increases its vector clock to s and broadcasts a dummy message to all processes.
4. The receipt of this dummy message forces each recipient to increase its clock to a value $\geq s$ if not already $\geq s$.
5. Each process takes a local snapshot and sends it to the initiator when (just before) its clock increases from a value less than s to a value $\geq s$.
6. The state of C_{ij} is all messages sent along C_{ij}, whose timestamp is smaller than s and which are received by p_j after recording LS_j.

Matter's Algorithm

- A termination detection scheme for non-FIFO channels is required to detect that no white messages are in transit.
- One of the following schemes can be used for termination detection:

First method:

- Each process i keeps a counter $cntr_i$ that indicates the difference between the number of white messages it has sent and received before recording its snapshot.
- It reports this value to the initiator process along with its snapshot and forwards all white messages, it receives henceforth, to the initiator.
- Snapshot collection terminates when the initiator has received $\sum_i cntr_i$ number of forwarded white messages.

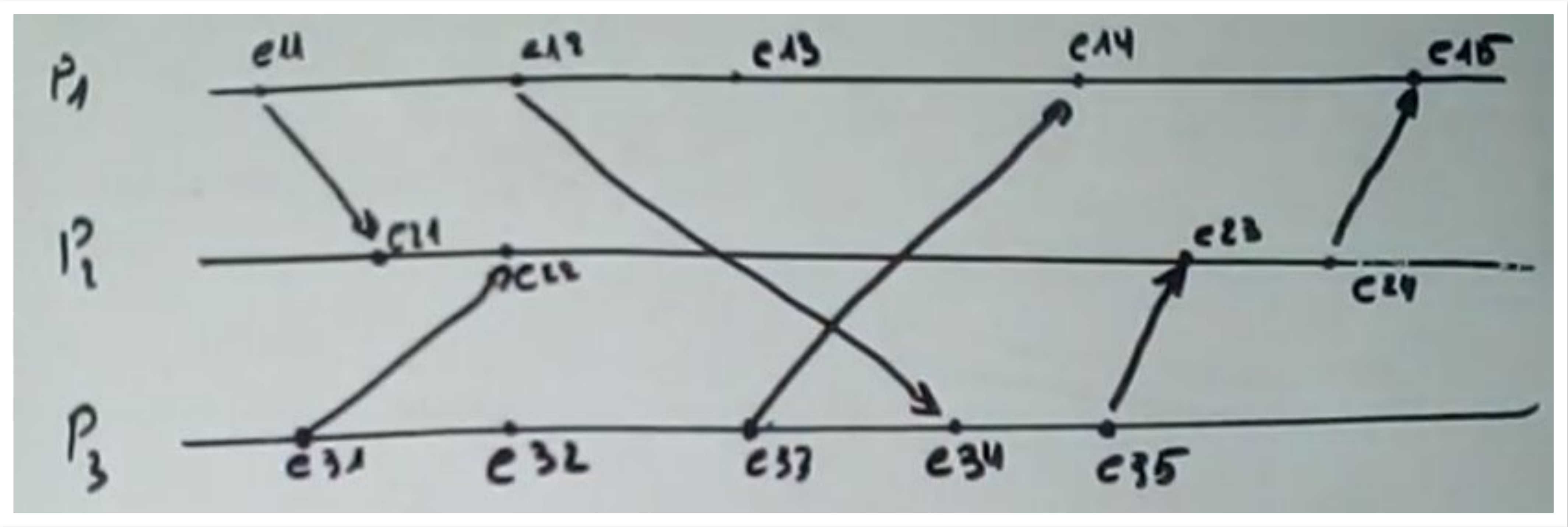
RAMAIAH
Institute of Technology

Matter's Algorithm

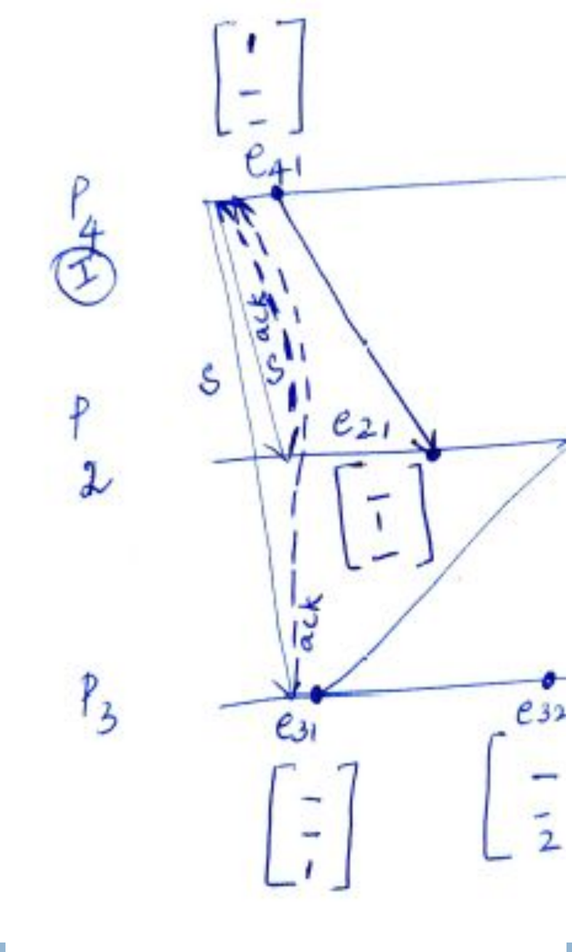
Second method:

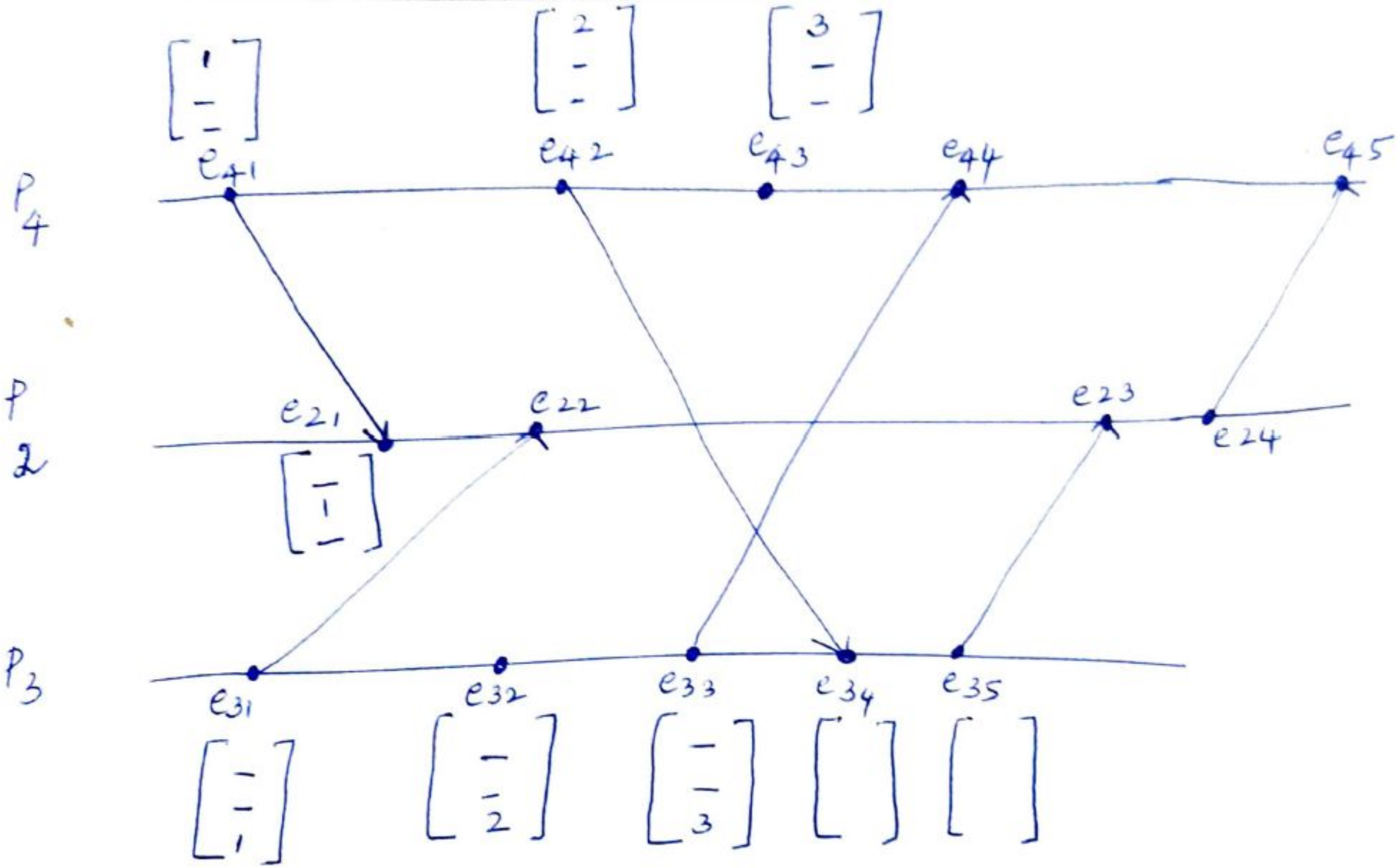
- Each red message sent by a process carries a piggybacked value of the number of white messages sent on that channel before the local state recording.
- Each process keeps a counter for the number of white messages received on each channel.
- A process can detect termination of recording the states of incoming channels when it receives as many white messages on each channel as the value piggybacked on red messages received on that channel.

Matter’s Algorithm example

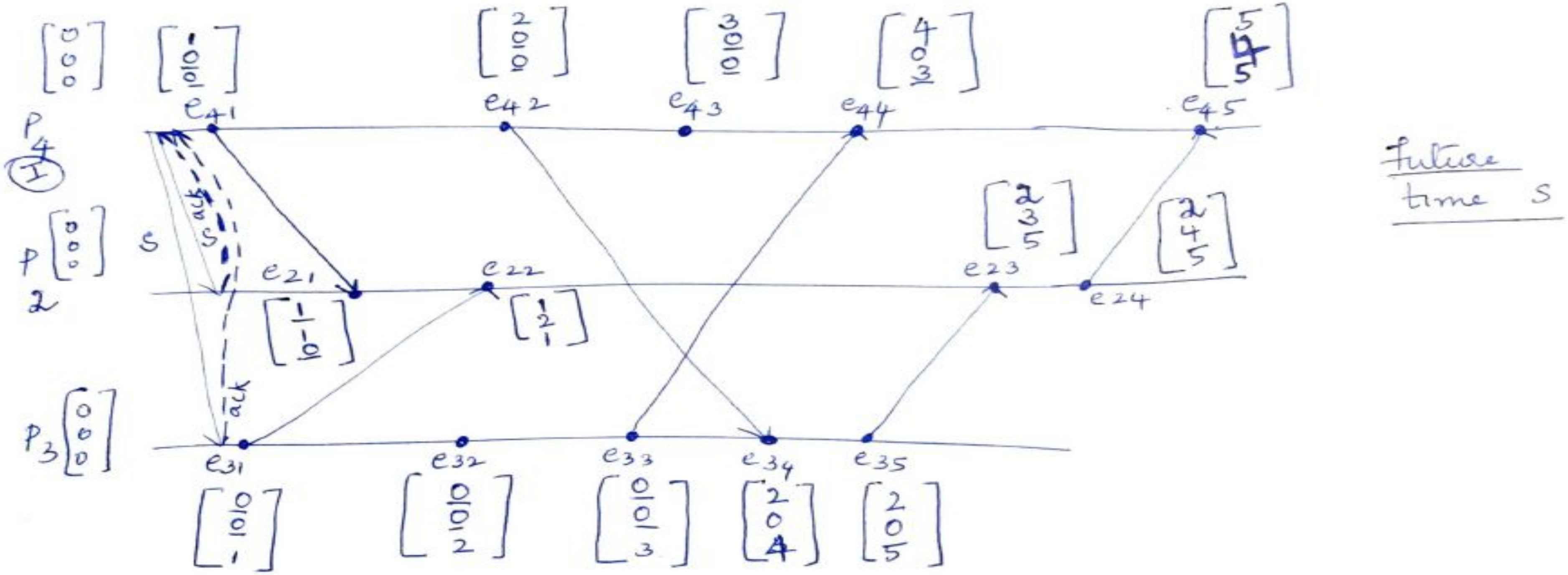


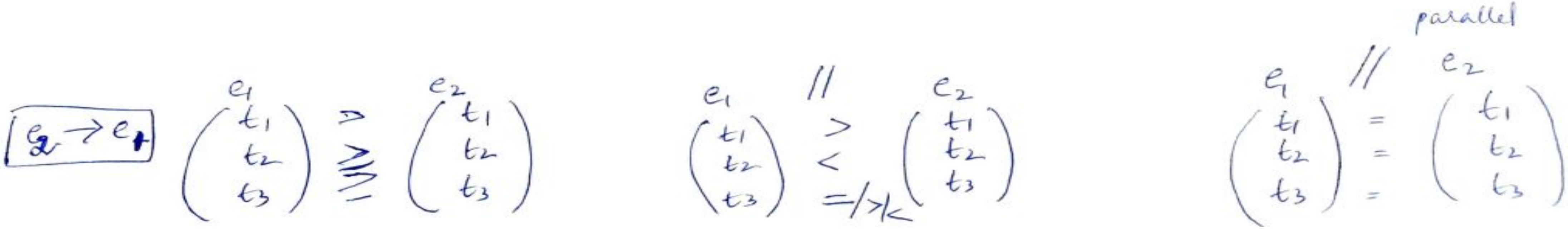
Mattern's Algorithm example





Mattern’s Algorithm example





Snapshots in a causal delivery system

- The causal message delivery property **CO** provides a built-in message synchronization to control and computation messages.
- Two global snapshot recording algorithms, namely, Acharya-Badrinath and Alagar-Venkatesan exist that assume that the underlying system supports causal message delivery.
- In both these algorithms recording of process state is identical and proceed as follows :
- An initiator process broadcasts a token, denoted as *token*, to every process including itself.
- Let the copy of the token received by process p_i be denoted $token_i$.
- A process p_i records its local snapshot LS_i when it receives $token_i$ and sends the recorded snapshot to the initiator.
- The algorithm terminates when the initiator receives the snapshot recorded by each process.

Snapshots in a causal delivery system

Correctness

For any two processes p_i and p_j, the following property is satisfied:

$$send(m_{ij}) \notin LS_i \Rightarrow rec(m_{ij}) \notin LS_j.$$

- This is due to the causal ordering property of the underlying system as explained next.
  - Let a message m_{ij} be such that $rec(token_i) \longrightarrow send(m_{ij})$.
  - Then $send(token_j) \longrightarrow send(m_{ij})$ and the underlying causal ordering property ensures that $rec(token_j)$, at which instant process p_j records LS_j, happens before $rec(m_{ij})$.
  - Thus, m_{ij} whose send is not recorded in LS_i, is not recorded as received in LS_j.
- Channel state recording is different in these two algorithms and is discussed next.

Channel state recording in Acharya-Badrinath algorithm

- Each process p_i maintains arrays $SENT_i[1, ...N]$ and $RECD_i[1, ..., N]$.
- $SENT_i[j]$ is the number of messages sent by process p_i to process p_j.
- $RECD_i[j]$ is the number of messages received by process p_i from process p_j.
- Channel states are recorded as follows:
  When a process p_i records its local snapshot LS_i on the receipt of $token_i$, it includes arrays $RECD_i$ and $SENT_i$ in its local state before sending the snapshot to the initiator.

Channel state recording in Acharya-Badrinath algorithm

When the algorithm terminates, the initiator determines the state of channels as follows:

- The state of each channel from the initiator to each process is empty.
- The state of channel from process p_i to process p_j is the set of messages whose sequence numbers are given by $\{RECD_j[i] + 1, \ldots, SENT_i[j]\}$.

Complexity:

- This algorithm requires $2n$ messages and 2 time units for recording and assembling the snapshot, where one time unit is required for the delivery of a message.
- If the contents of messages in channels state are required, the algorithm requires $2n$ messages and 2 time units additionally.

Channel state recording in Alagar-Venkatesan algorithm

- A message is referred to as *old* if the send of the message causally precedes the send of the token.
- Otherwise, the message is referred to as *new*.

In Alagar-Venkatesan algorithm channel states are recorded as follows:

1. When a process receives the *token*, it takes its snapshot, initializes the state of all channels to empty, and returns *Done* message to the initiator. Now onwards, a process includes a message received on a channel in the channel state only if it is an old message.
2. After the initiator has received *Done* message from all processes, it broadcasts a *Terminate* message.
3. A process stops the snapshot algorithm after receiving a *Terminate* message.

Thank you