RAMAIAH
Institute of Technology

M. S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)

Department of Computer Science and Engineering

Distributed Systems
CSE751

Sini Anna Alex

8x8 Omega Network (logical left shift)

000->000->000->000

001->010->100->001

010->100->001->010

011->110->101->011

100->001->010->100

101->011->110->101

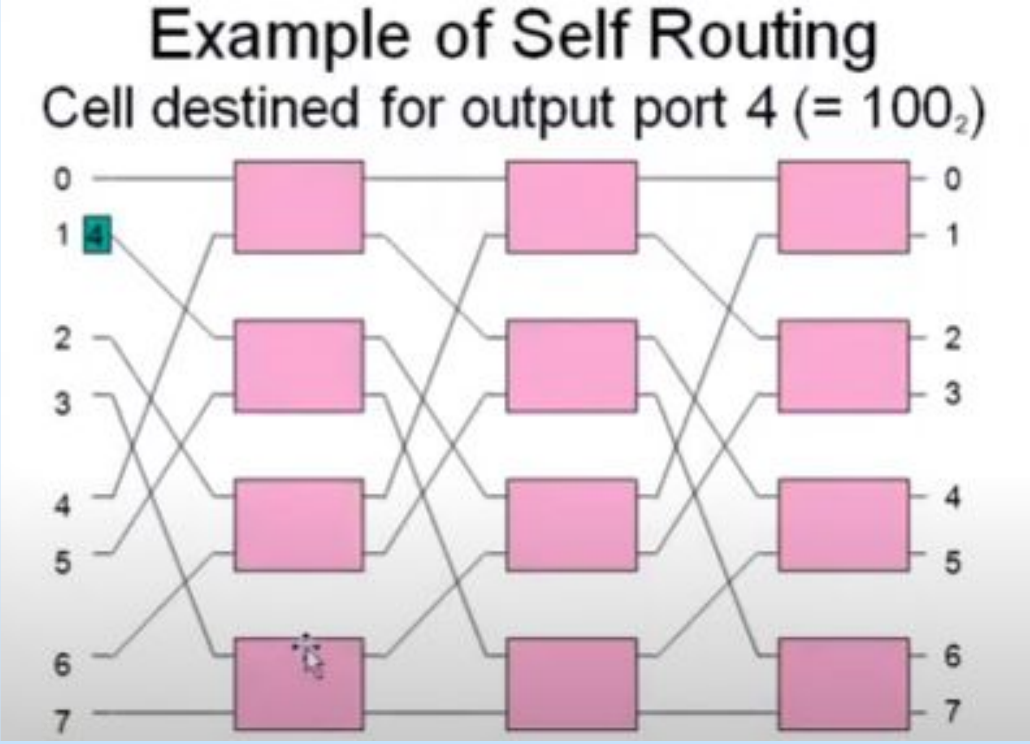
110->101->011->110

111->111->111->111

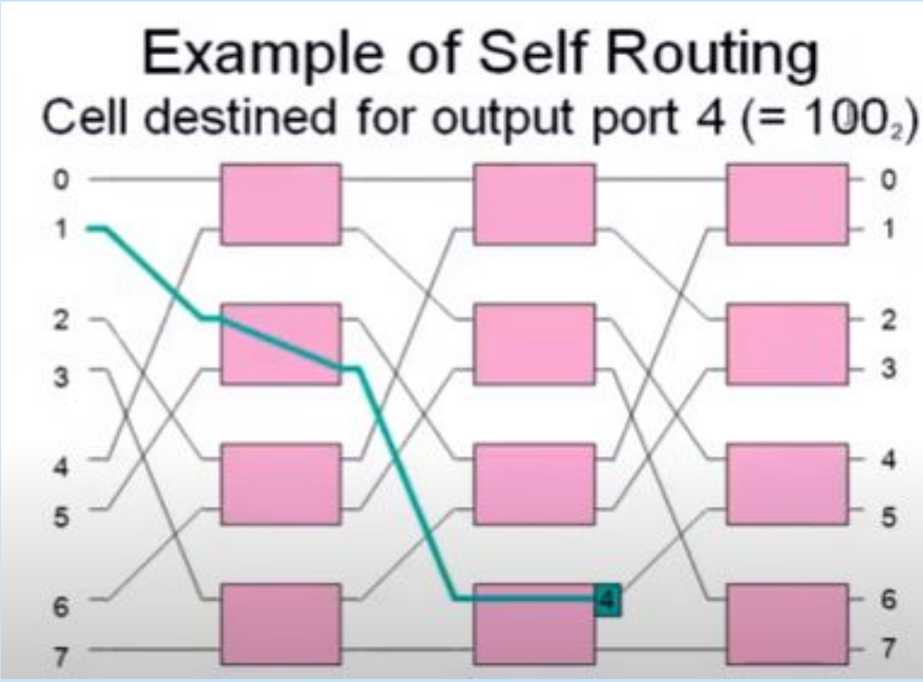
Input	Output
0	0
1	2
2	4
3	6
4	1
5	3
6	5
7	7

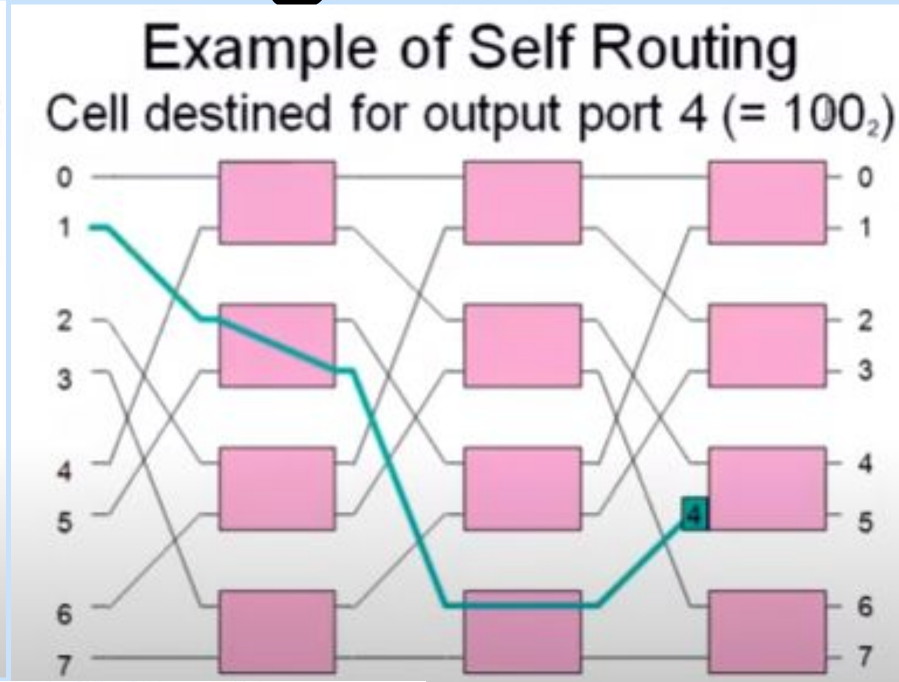
Omega Network (Self Routing)

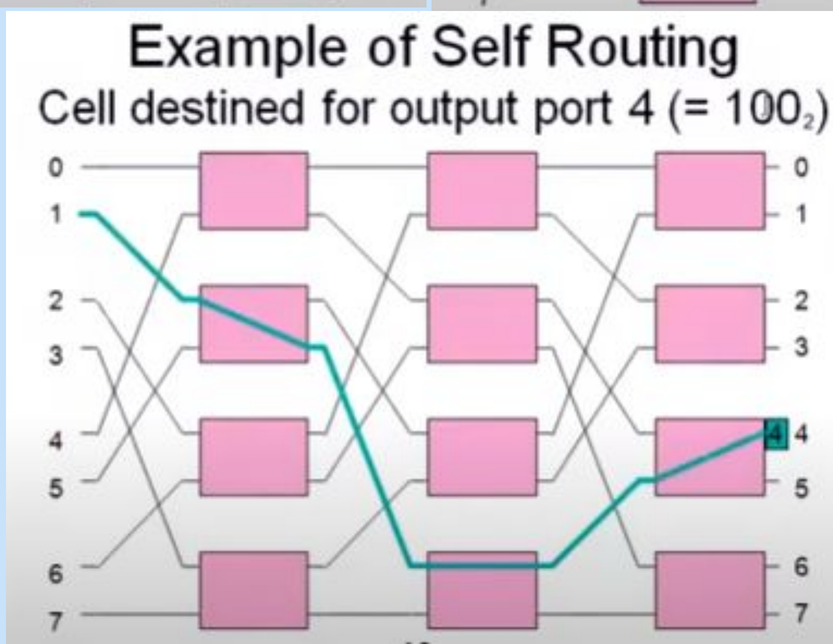
- Omega Network has self-routing-property
- The path for a cell to take to reach its destination can be determined directly from its routing tag.



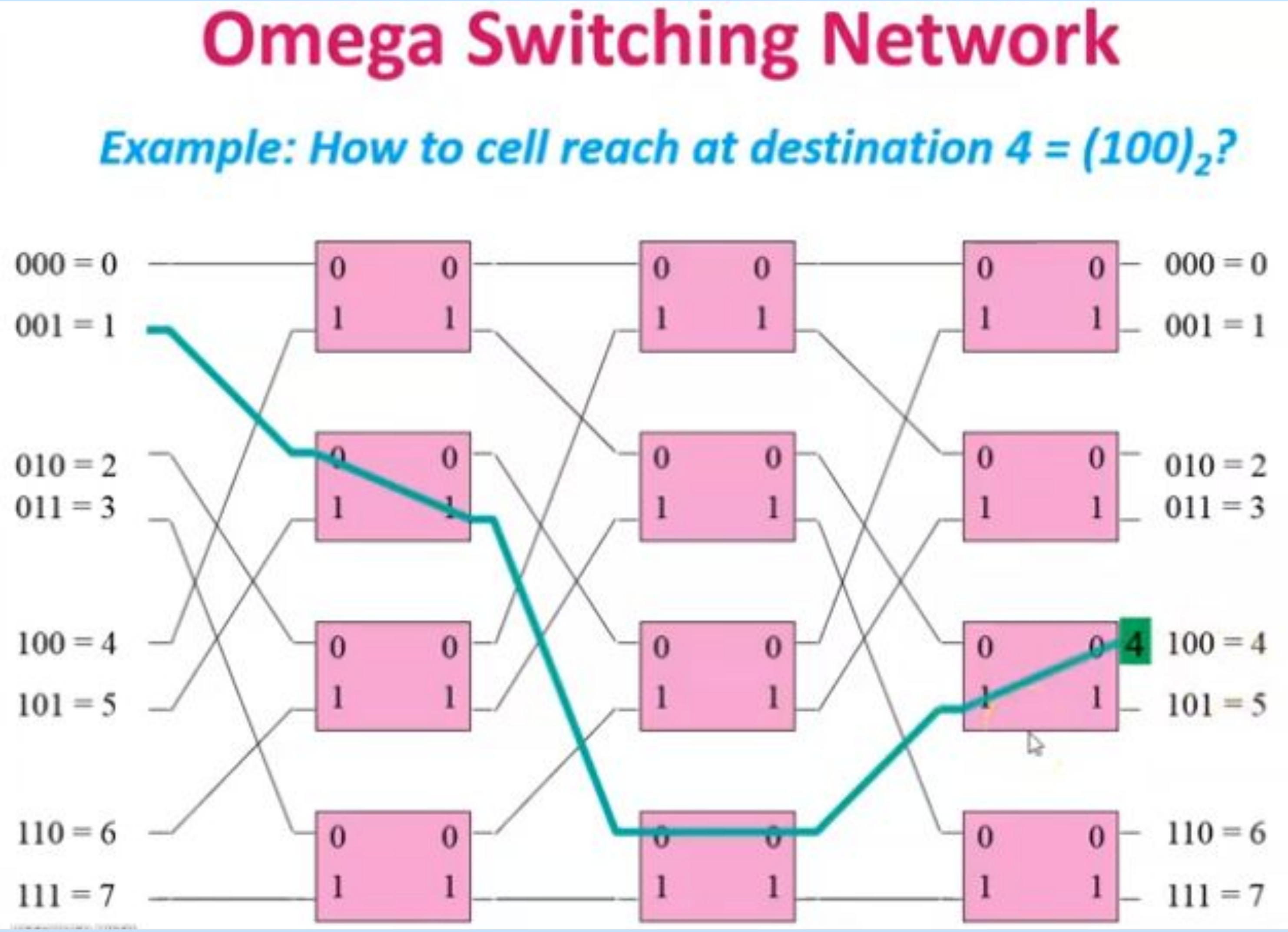
Self Routing





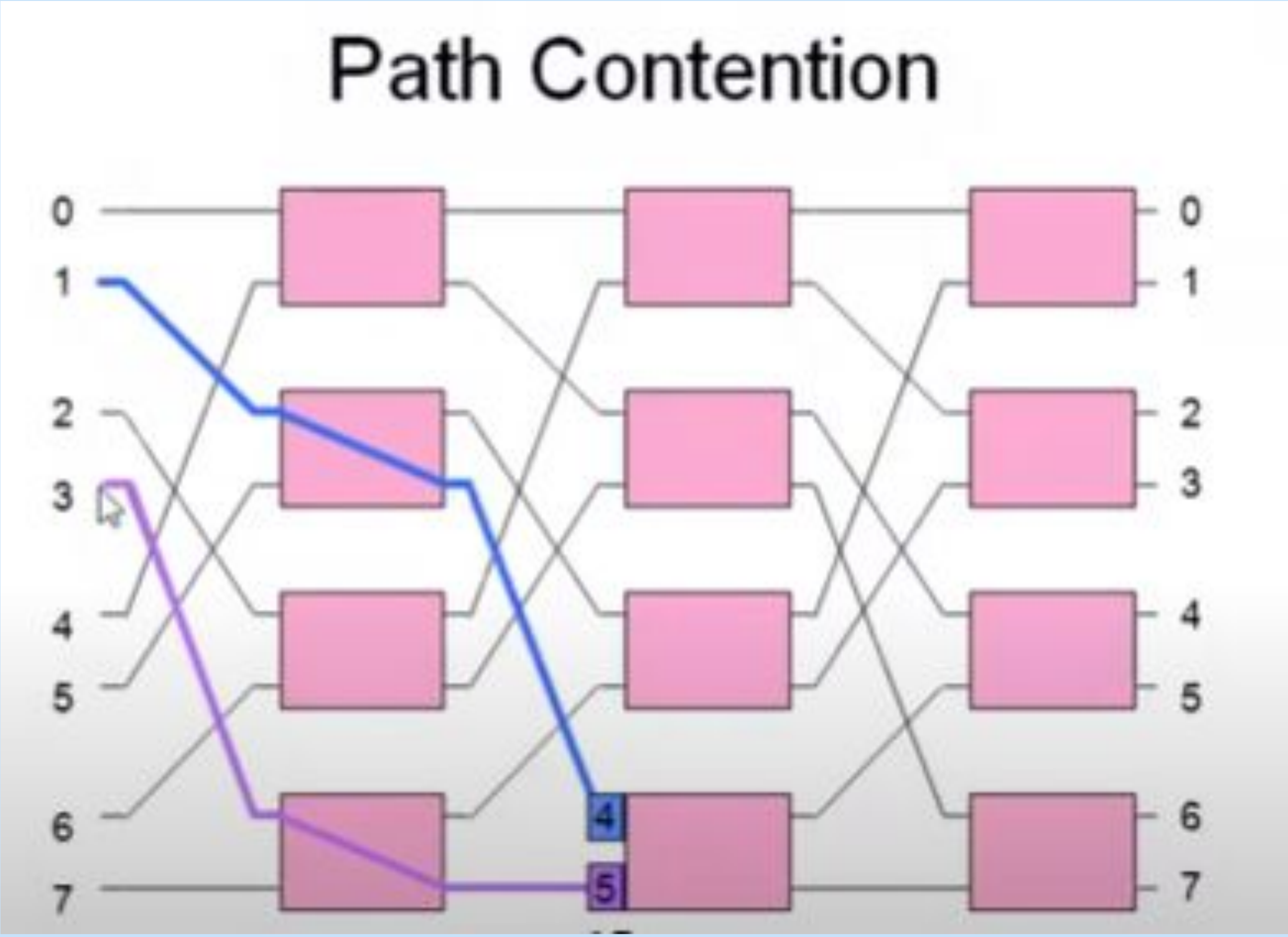


How to reach at destination cell 4=$(100)_2$?

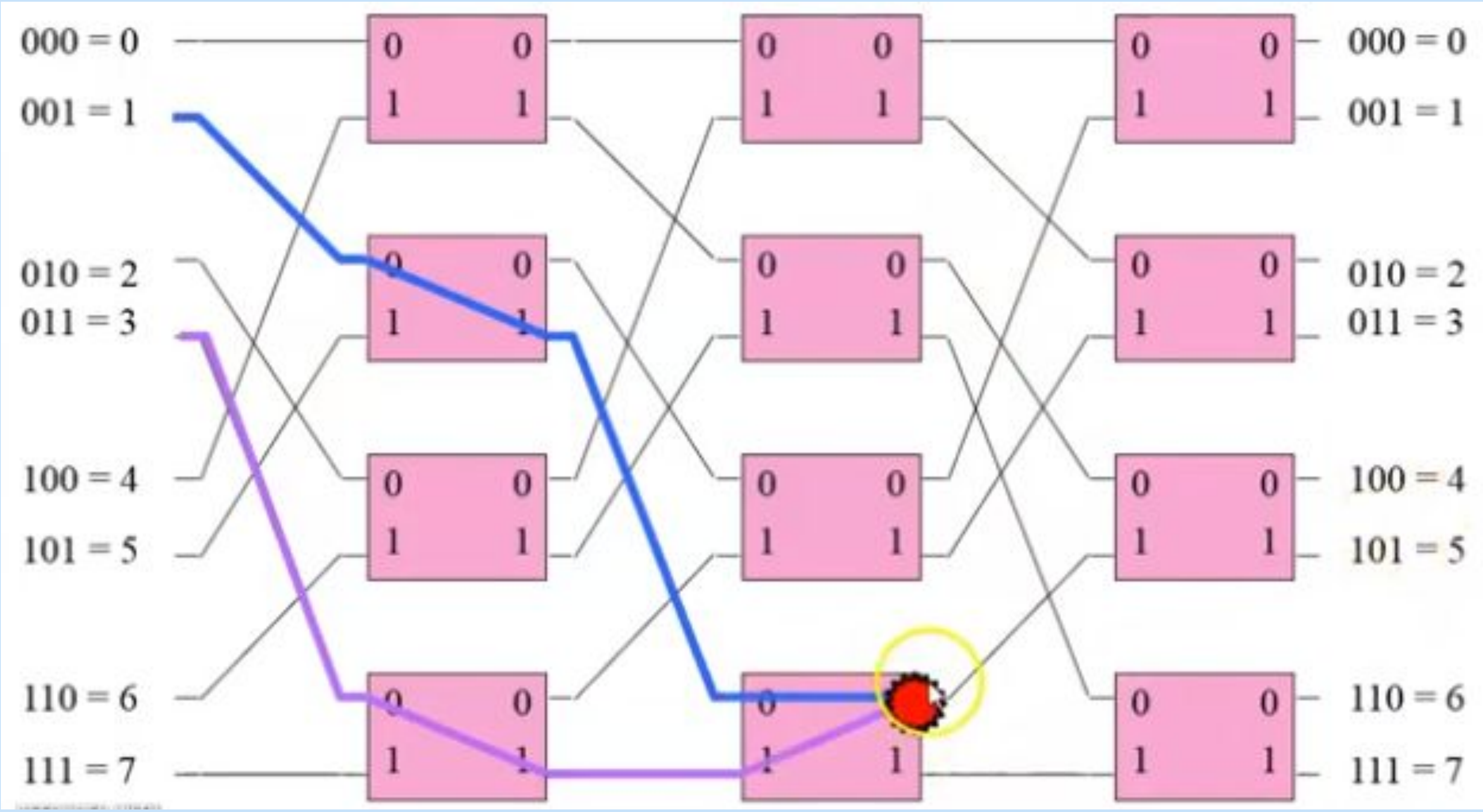


Path Contention(Cell Loss)

- The omega network has the problems with output port contention and path contention.

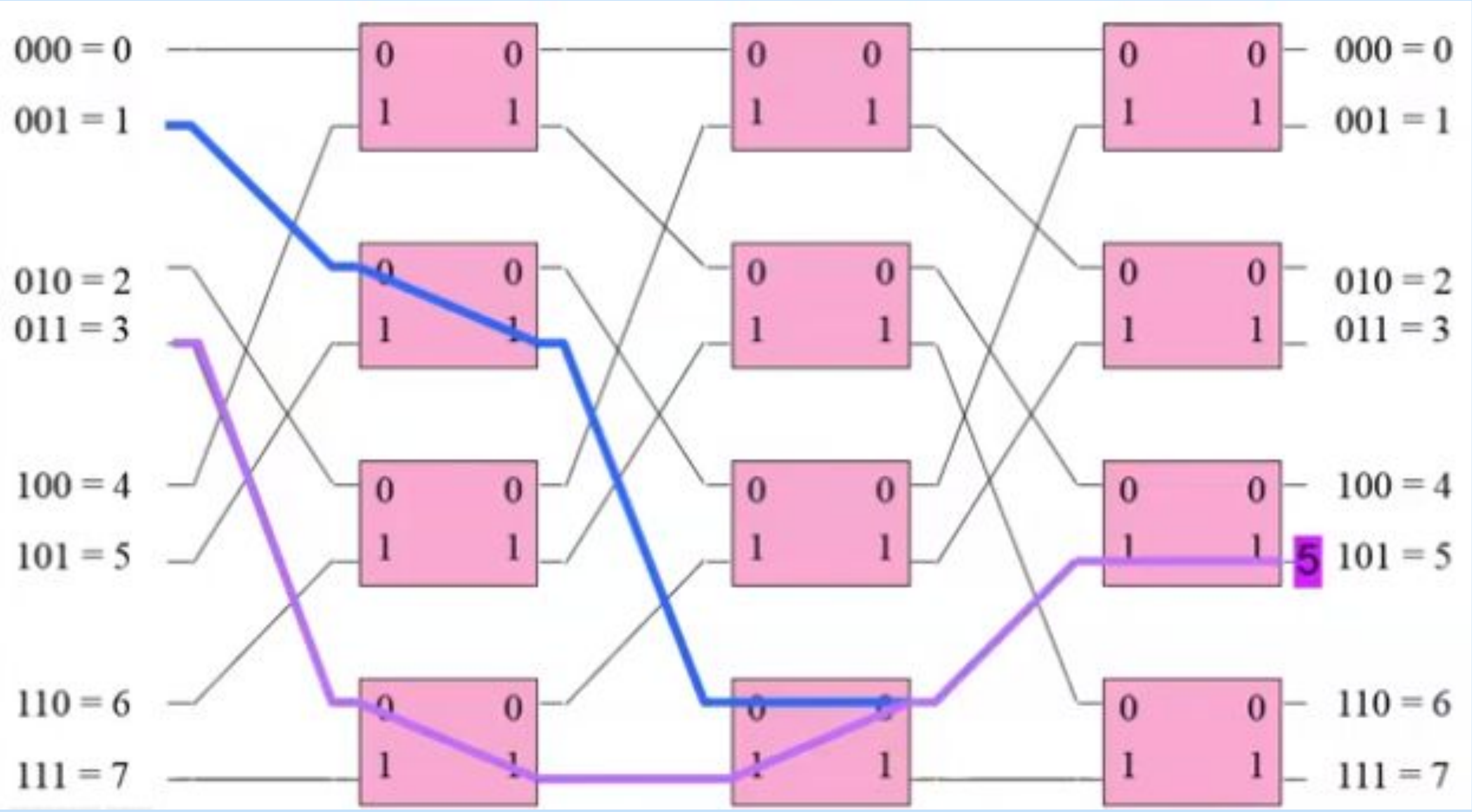


Path Contention(Cell Loss) contd...



Path Contention(Cell Loss) contd...

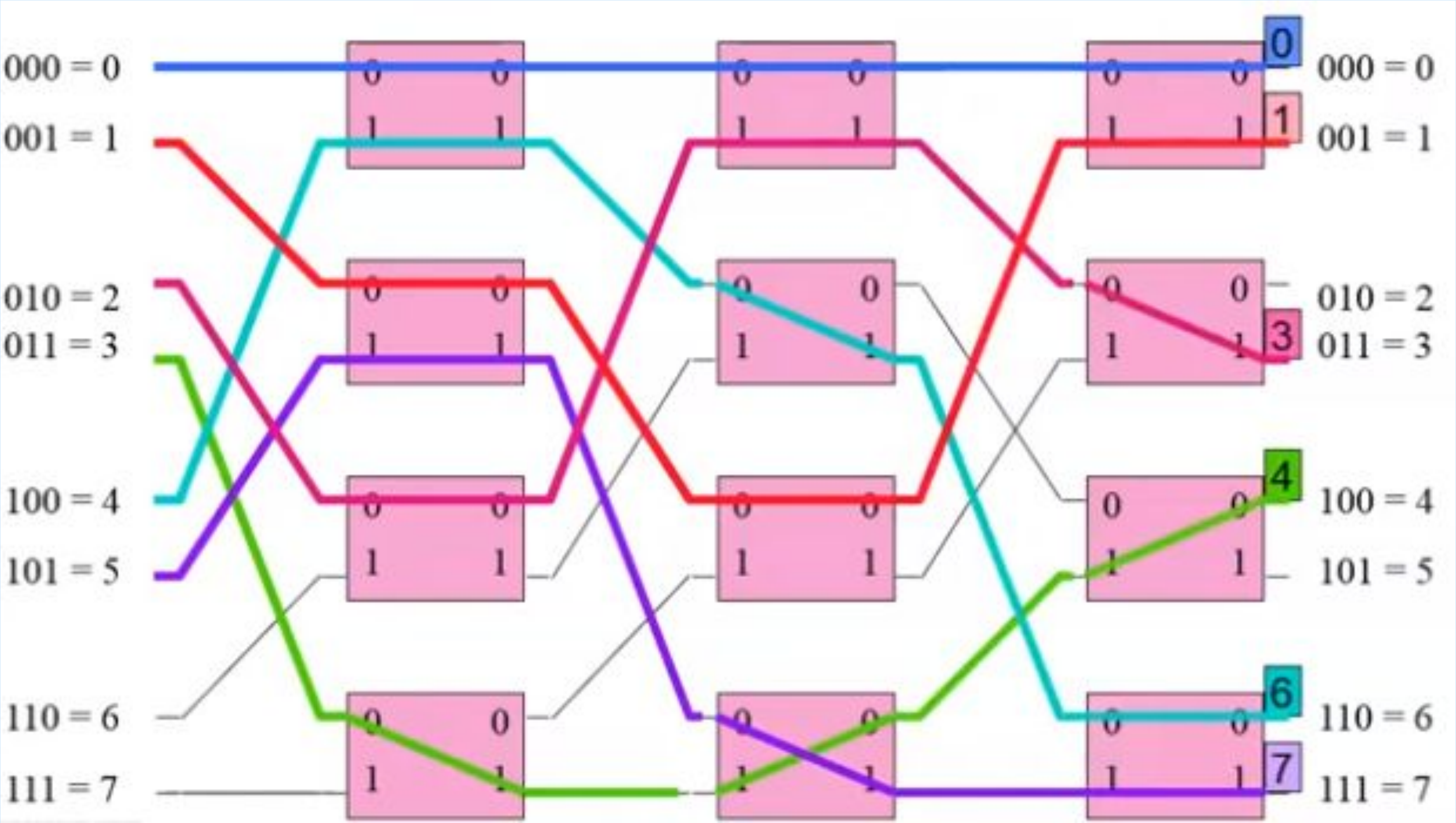
Middle level switch both conflict at 0. Higher number win and cell loss may happen



Solution of Path Contention

Batcher Sorter

- Sort the cells in ascending order based on desired destination port.
- Done using a bitonic sorter called a Batcher.
- Places the M cells into gap free increasing sequence on the first M input ports.
- Eliminate duplicate destinations.



Chapter 2: A Model of Distributed Computations

A Distributed Program

- A distributed program is composed of a set of n asynchronous processes, p_1, p_2, ..., p_i, ..., p_n.
- The processes do not share a global memory and communicate solely by passing messages.
- The processes do not share a global clock that is instantaneously accessible to these processes.
- Process execution and message transfer are asynchronous.
- Without loss of generality, we assume that each process is running on a different processor.
- Let C_{ij} denote the channel from process p_i to process p_j and let m_{ij} denote a message sent by p_i to p_j.
- The message transmission delay is finite and unpredictable.

A Model of Distributed Executions

- The execution of a process consists of a sequential execution of its actions.
- The actions are atomic and the actions of a process are modeled as three types of events, namely, internal events, message send events, and message receive events.
- Let e_i^x denote the xth event at process p_i.
- For a message m, let $send(m)$ and $rec(m)$ denote its send and receive events, respectively.
- The occurrence of events changes the states of respective processes and channels.
- An internal event changes the state of the process at which it occurs.
- A send event changes the state of the process that sends the message and the state of the channel on which the message is sent.
- A receive event changes the state of the process that receives the message and the state of the channel on which the message is received.

A Model of Distributed Executions

- The send and the receive events signify the flow of information between processes and establish causal dependency from the sender process to the receiver process.
- A relation $\rightarrow_{msg}$ that captures the causal dependency due to message exchange, is defined as follows. For every message m that is exchanged between two processes, we have

$$send(m) \rightarrow_{msg} rec(m).$$

- Relation $\rightarrow_{msg}$ defines causal dependencies between the pairs of corresponding send and receive events.

A Model of Distributed Executions

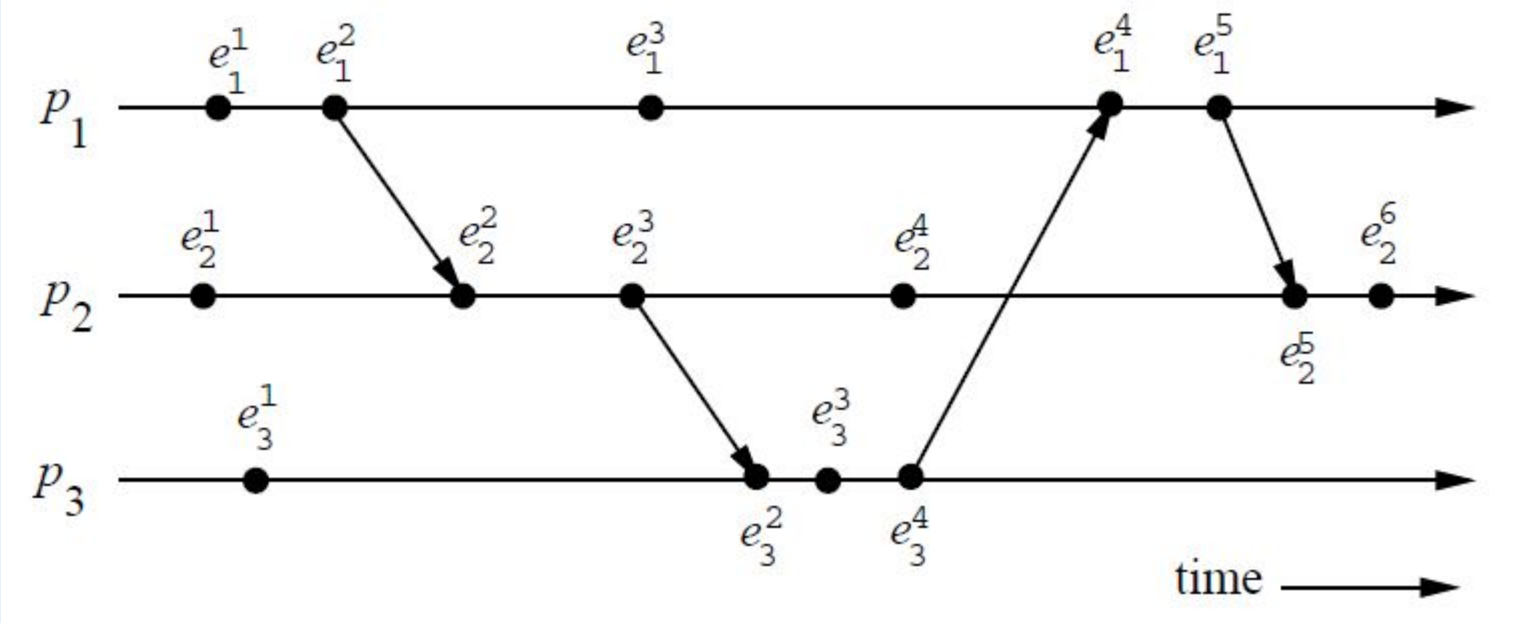
- The events at a process are linearly ordered by their order of occurrence.
- The execution of process p_i produces a sequence of events e_i^1, e_i^2, ..., e_i^x, e_i^{x+1}, ... and is denoted by $\mathcal{H}_i$ where

$$\mathcal{H}_i = (h_i, \rightarrow_i)$$

  h_i is the set of events produced by p_i and
  binary relation $\rightarrow_i$ defines a linear order on these events.
- Relation $\rightarrow_i$ expresses causal dependencies among the events of p_i.

A Model of Distributed Executions

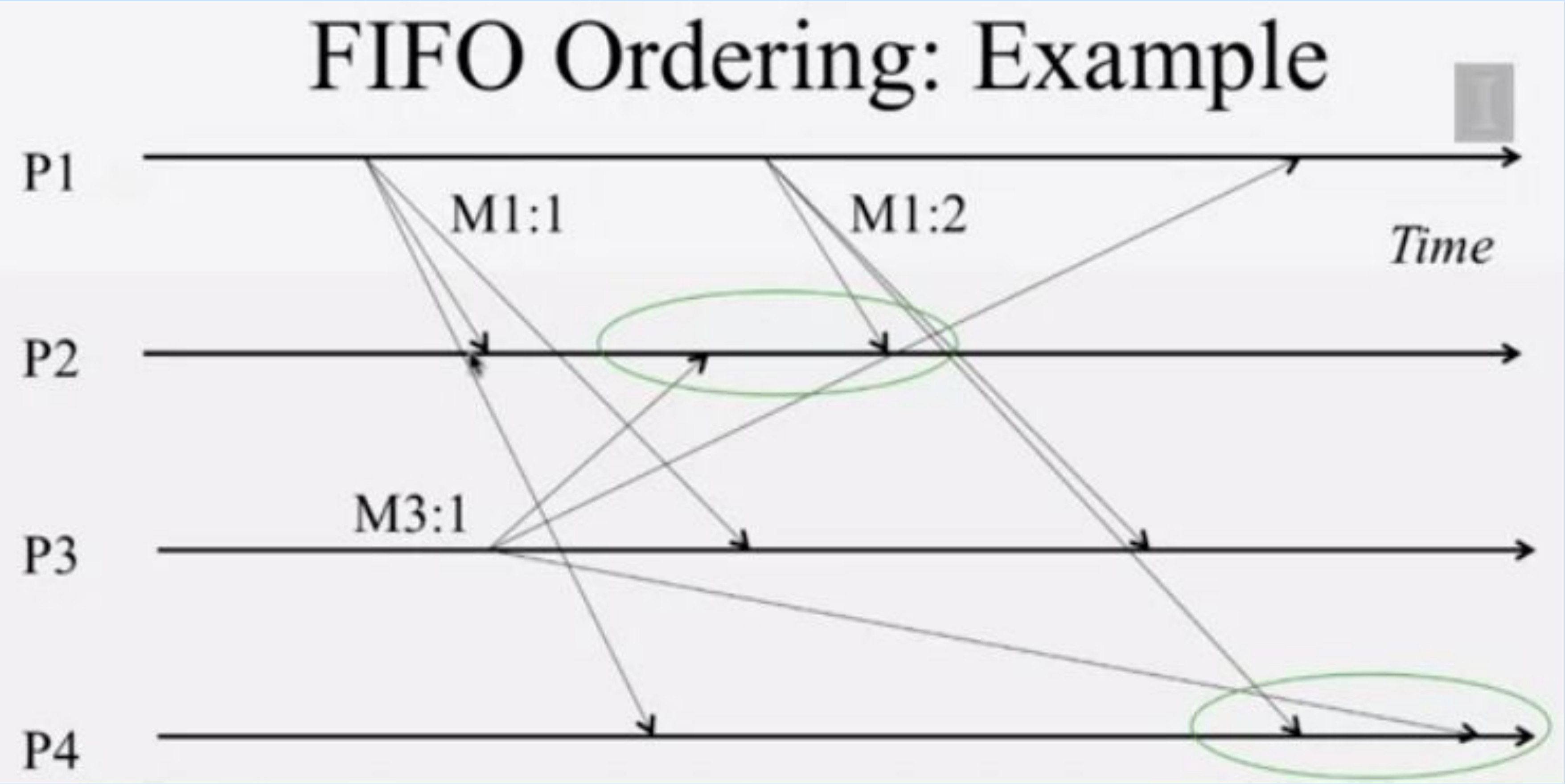
- The evolution of a distributed execution is depicted by a space-time diagram.
- A horizontal line represents the progress of the process; a dot indicates an event; a slant arrow indicates a message transfer.
- Since we assume that an event execution is atomic (hence, indivisible and instantaneous), it is justified to denote it as a dot on a process line.
- In the Figure given below, for process p1, the second event is a message send event, the third event is an internal event, and the fourth event is a message receive event.



Ordering of messages

- FIFO Ordering: Messages for each sender are received in the order they are sent, at all receivers
- Causal Ordering: Messages whose send events are causally related, must be received in the same causality-obeying order at all receivers

FIFO ordering



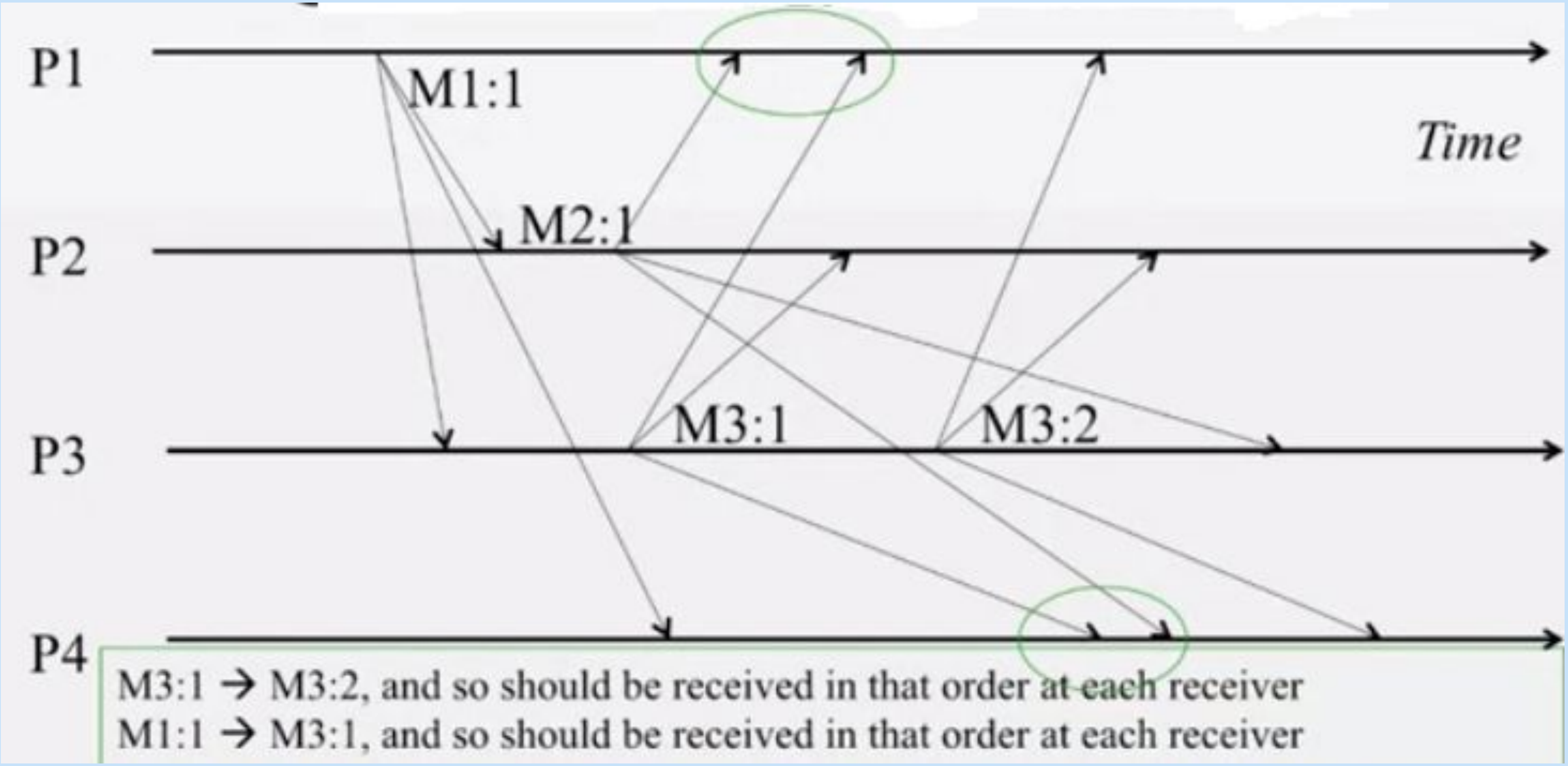
- M 1,1 and M 1,2 should be received in the same order at the receiver.
- Order of delivery of M3,1 and M1,2 could be different at different at different receivers

Causal Ordering

The "causal ordering" is based on Lamport's "happens before" relation. A system that supports the causal ordering model satisfies the following property:

CO: For any two messages m_{ij} and m_{kj}, if $send(m_{ij}) \longrightarrow send(m_{kj})$, then $rec(m_{ij}) \longrightarrow rec(m_{kj})$.

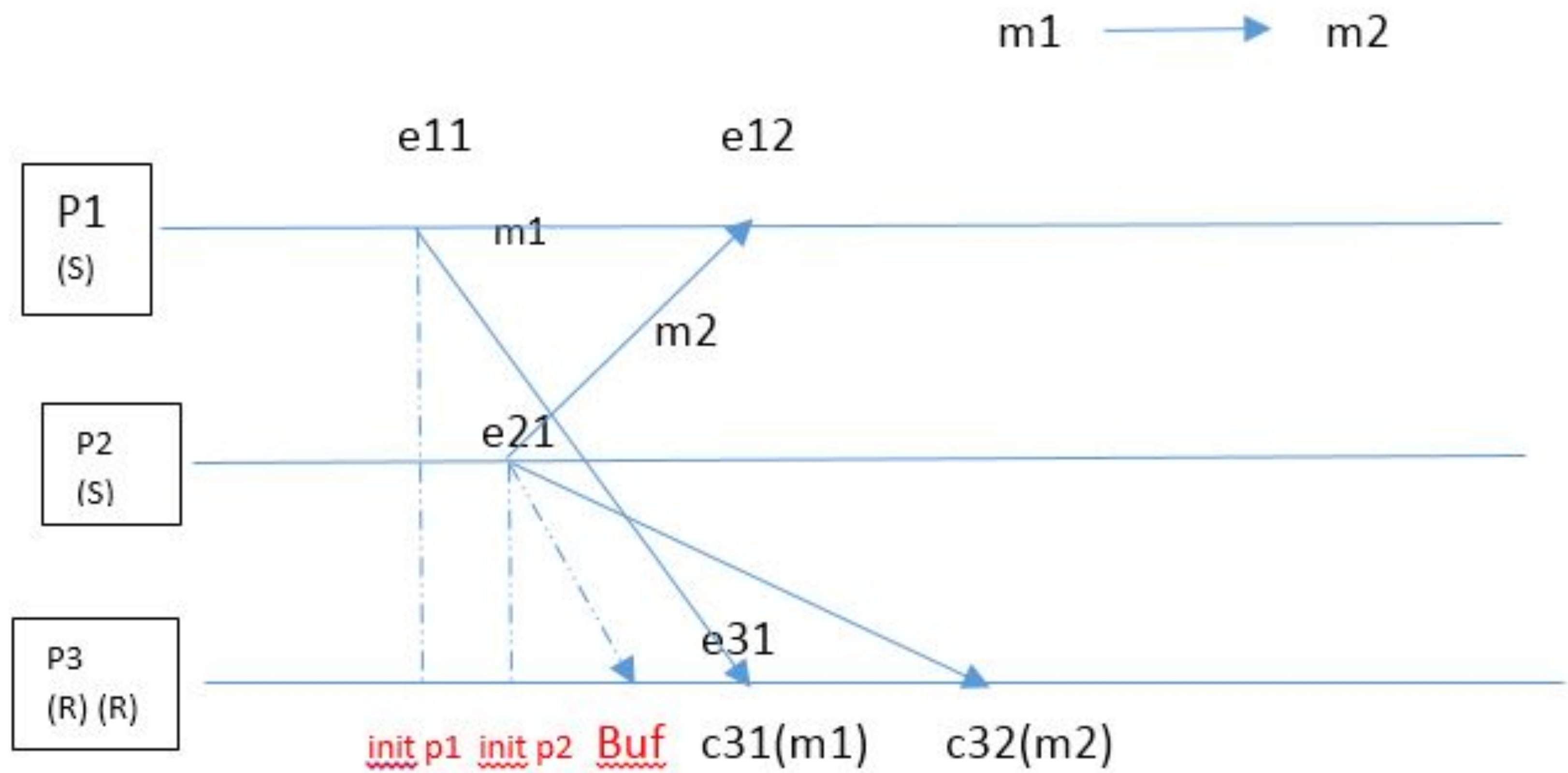
Causal Ordering



M 3:1 and M2:1 are concurrent and thus ok to be received in different orders at different receivers

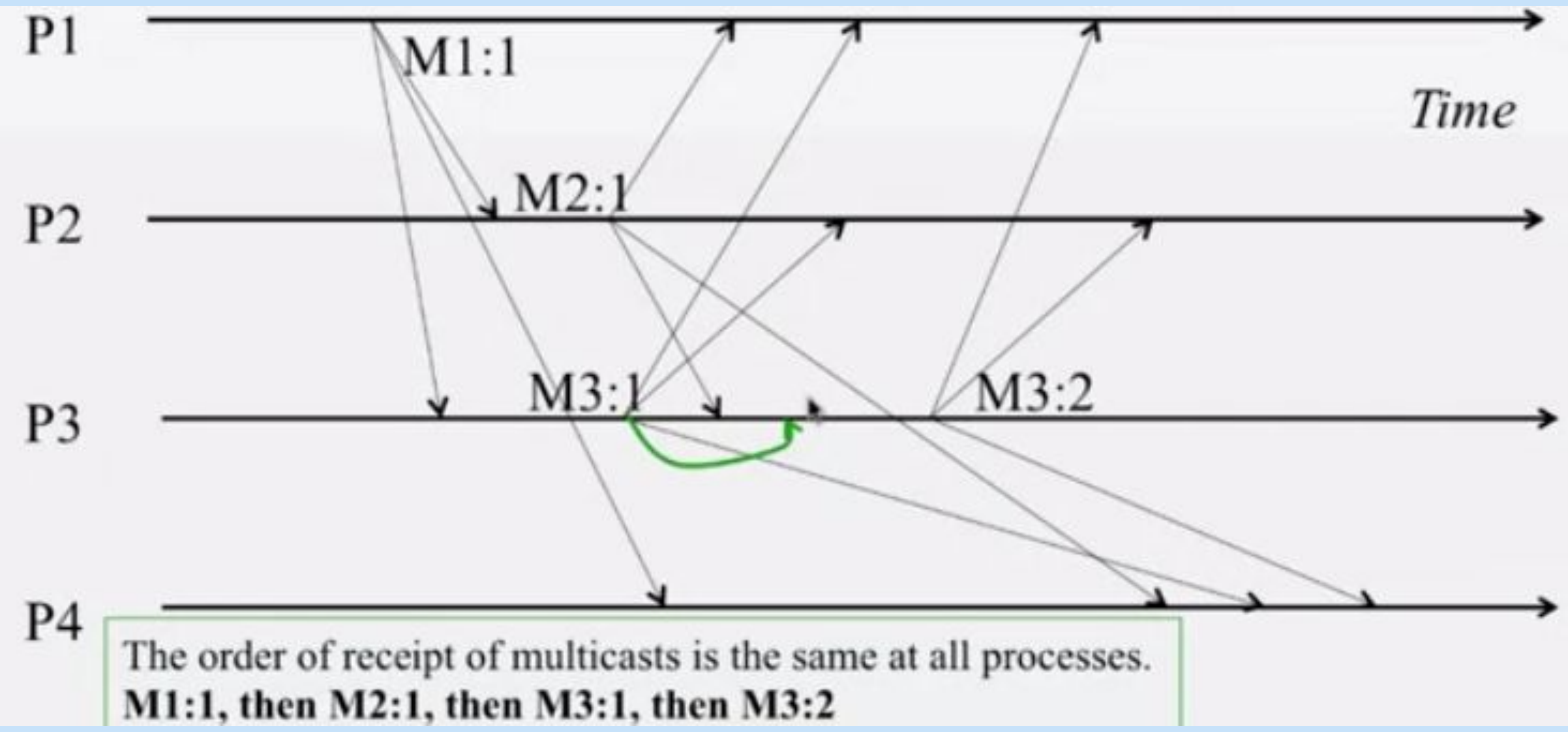
Cause-effect relations in message passing systems

An event e1 may potentially have caused another event e2 if the following relation, called, *happens-before* and denoted by e1 $\rightarrow$ e2 holds



Total Ordering

Unlike FIFO or causal it doesn't focus on order of sending messages. Ensures all receives happen in the same order.



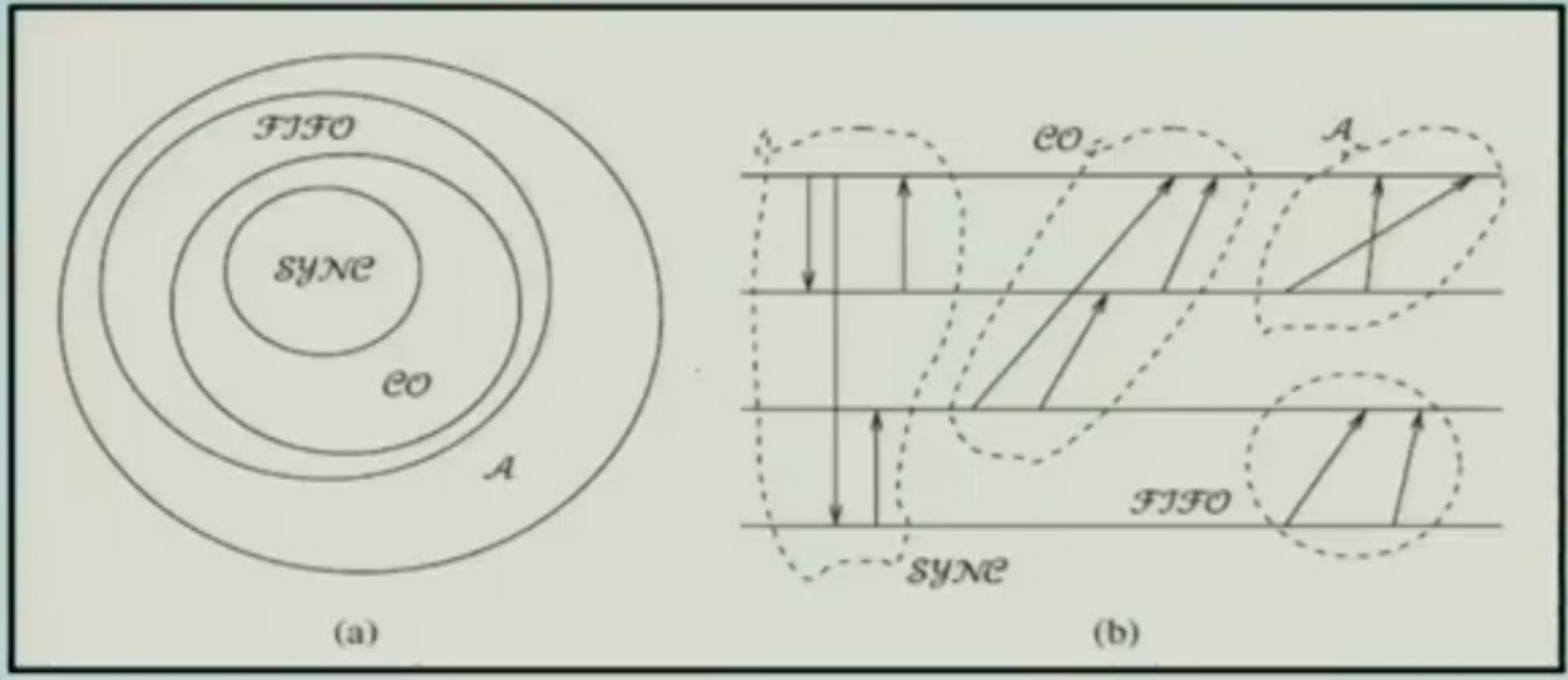
Non-FIFO ordering

- In the 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.

Hierarchy of execution classes

***SYNC*, *CO*, *FIFO*, and *A* denote** *the set of all possible* executions ordered by synchronous order, causal order, FIFO order, and non- FIFO order, respectively

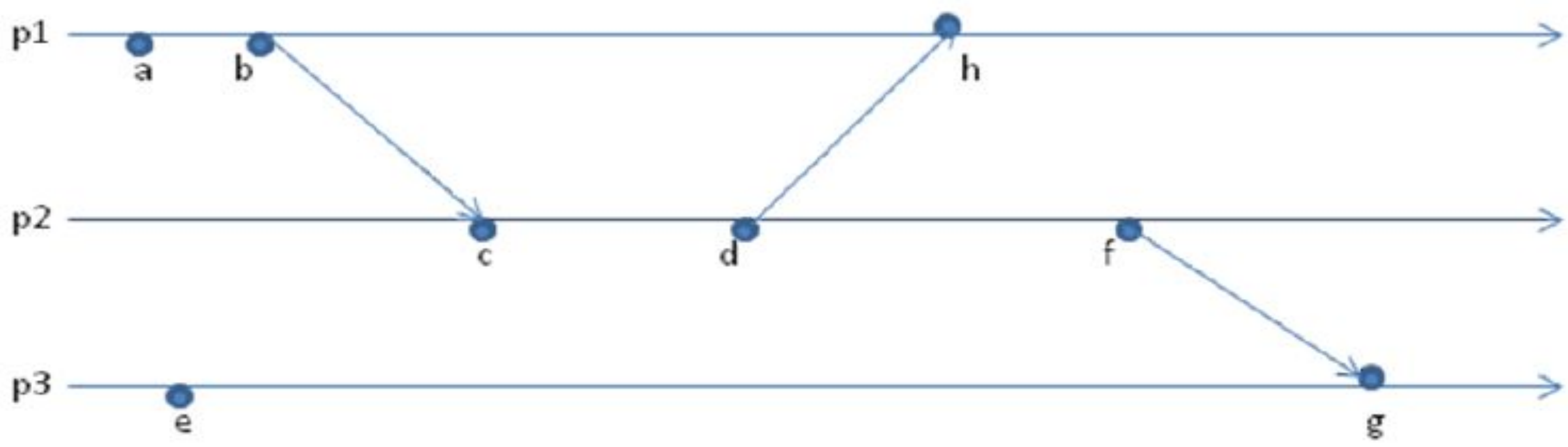
$SYNC \subset CO \subset FIFO \subset A$



Summary

- Causally ordered delivery of messages implies FIFO message delivery. (Note that CO $\subset$ FIFO $\subset$ Non-FIFO.)
- Causal ordering model considerably simplifies the design of distributed algorithms because it provides a built-in synchronization.

Modify this Space Time Diagram with corresponding event actions



Solution:

Point a- e_1^1

Point b- e_1^2

Point c- e_2^1

Point d- e_2^2

Point e- e_3^1

Point f- e_2^3

Point g- e_3^2

Point h- e_1^3

Internal Event points: a,e

Send event points: b,d,f

Receive event Points: c,h,g