Distributed Systems **Spring 2021**

LECTURE 2

Prepared by:Mahipala Vasant (201564018) Vishal Malhotra (2019201018) Sartak Periwal(2018101024)

Contents

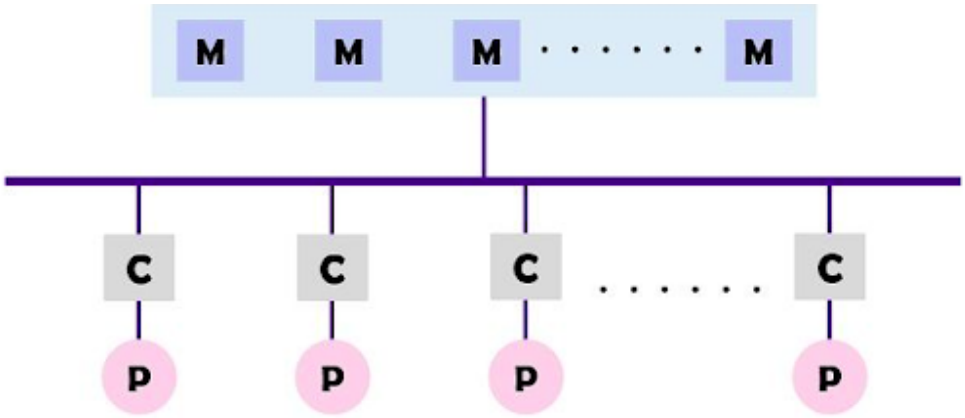
1 Presentation scribe

Shared memory is memory that may be simultaneously accessed by multiple programs with an intent to provide communication among them or avoid redundant copies. Shared memory is an efficient means of passing data between programs. Depending on context, programs may run on a single processor or on multiple separate processors.

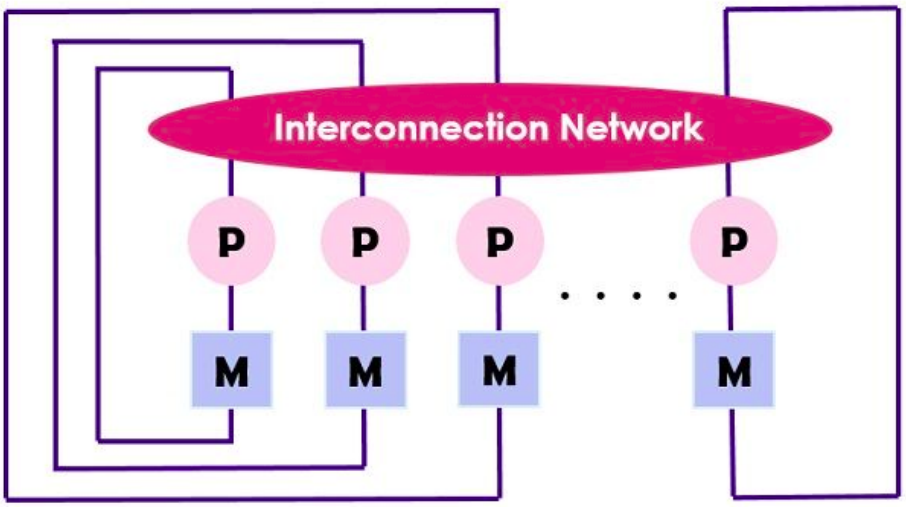
- Processes communicate with small address spaces.
- Shared memory allows multiple processes to share virtual memory.
- The size and access permission for the segment is set when it is created.
- In general one process creates or allocates the shared memory segment.
- This is the fastest but not necessarily the easiest way (synchronization wise) way for process to communicate with one another.

Types of shared memory

1. **UMA (uniform memory access)** system is a shared memory architecture for the multiprocessors. In this model, a single memory is used and accessed by all the processors present the multiprocessor system with the help of the interconnection network.



2. **NUMA(non uniform memory access)** is also a multiprocessor model in which each processor connected with the dedicated memory. However, these small parts of the memory combine to make a single address space.Multiprocessor model in which each processor connected with the dedicated memory. However, these small parts of the memory combine to make a single address space..



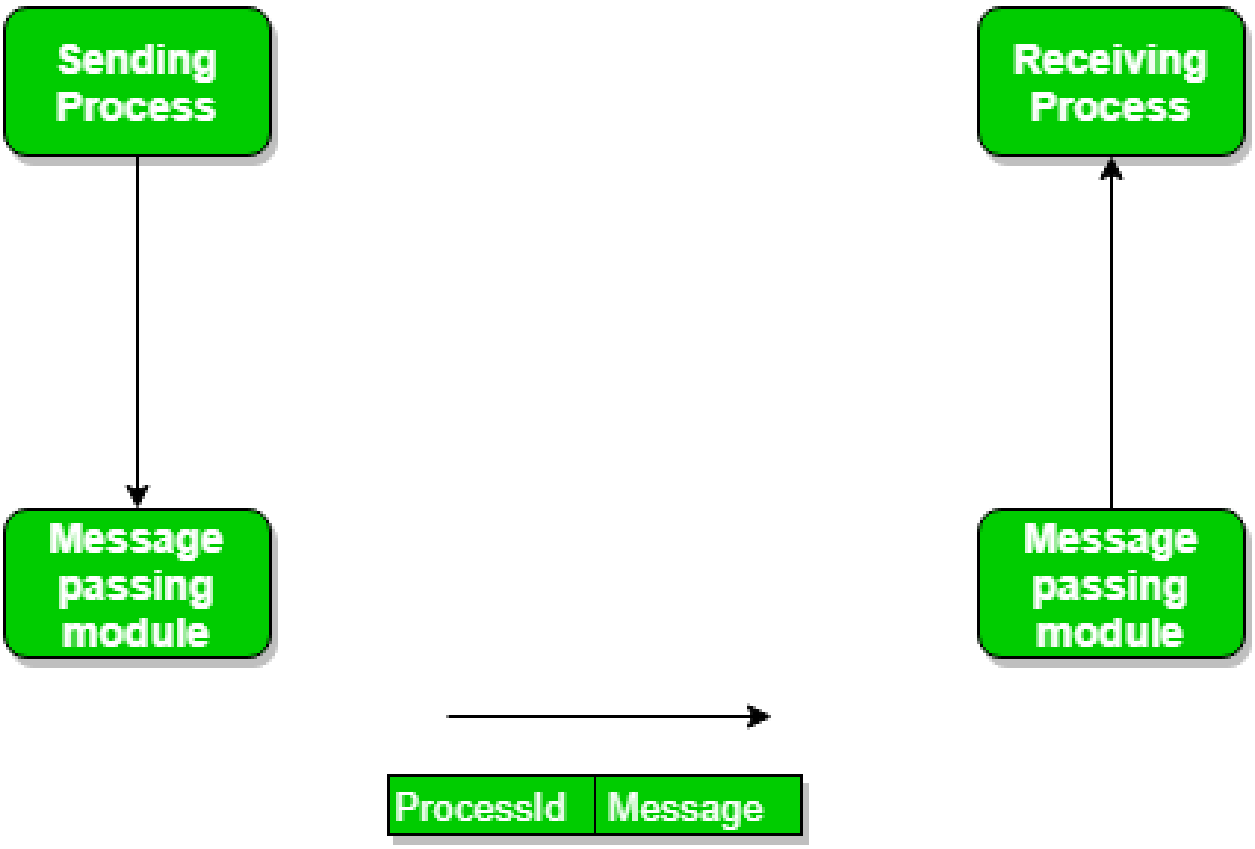
BASIS FOR COMPARISON	UMA	NUMA
Basic	Uses a single memory controller	Multiple memory controller
Type of buses used	Single, multiple and crossbar.	Tree and hierarchical
Memory accessing time	Equal	Changes according to the distance of microprocessor.
Suitable for	General purpose and time-sharing applications	Real-time and time-critical applications
Speed	Slower	Faster
Bandwidth	Limited	More than UMA.

Message passing method -: Now, We will start our discussion of the communication between processes via message passing. In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

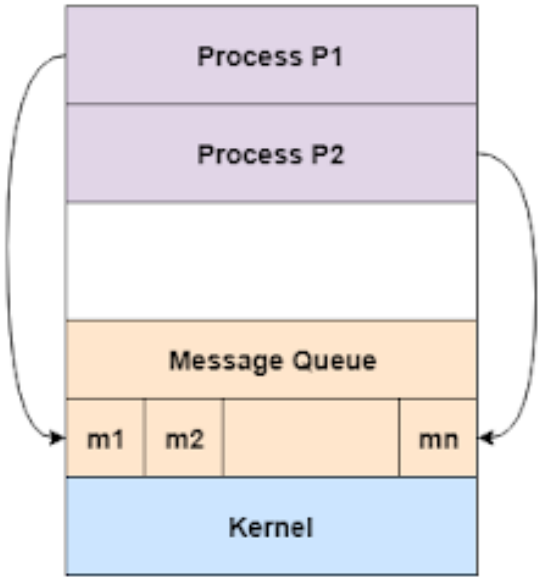
Establish a communication link (if a link already exists, no need to establish it again.) Start exchanging messages using basic primitives. We need at least two primitives:

- send(message, destination) or send(message)
- receive(message, host) or receive(message)

The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for an OS designer but complicated for a programmer and if it is of variable size then it is easy for a programmer but complicated for the OS designer. A standard message can have two parts: header and body. The header part is used for storing message type, destination id, source id, message length, and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.



A **link** has some capacity that determines the number of messages that can reside in it temporarily for which every link has a queue associated with it which can be of zero capacity, bounded capacity, or unbounded capacity.Implementation of the link depends on the situation,it can be either a direct / indirect communication.



Message Passing Model

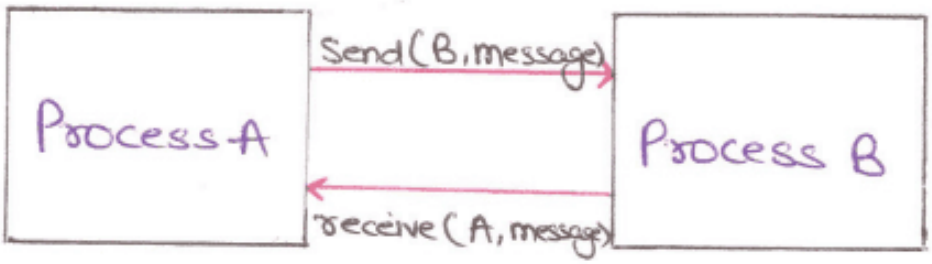
Synchronous and Asynchronous-: A process that is blocked is one that is waiting for some event, such as a resource becoming available or the completion of an I/O operation. Blocking is synchronous whereas Non-blocking is asynchronous. The process may or may not be blocked while sending a message or attempting to receive a message so message passing may be blocking or non-blocking.

- Blocking send : The sender will be blocked until the message is received by receiver.
- Blocking receive : The receiver will block until the message is available.
- Non blocking send : The sender sends the message without waiting for confirmation.
- Non blocking receive : The receiver either receives a valid message or Null.

Types of message passing

1. **Direct communication**In Direct communication the process which want to communicate must explicitly name the recipient or sender of communication. For example
   - send(p1, message) means send the message to p1.
   - receive(p2, message) means receive the message from p2.

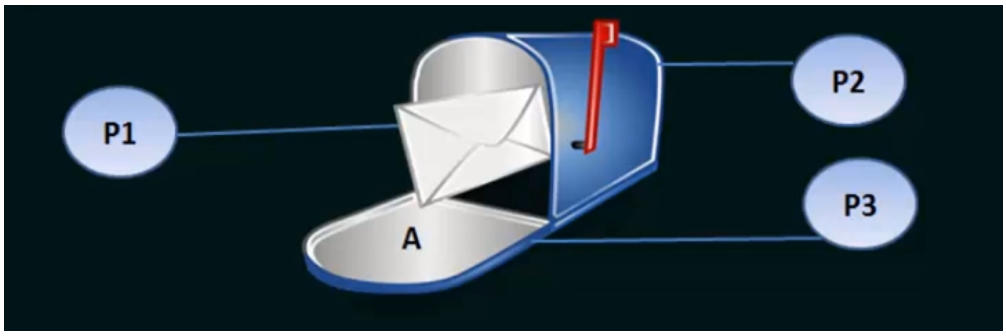
   The communication link established can be unidirectional or bi directional. The problem with this method of communication is that if the name of one process changes, this method will not work.



2. **Indirect communication** In Indirect message passing, processes use mailboxes (also referred to as ports) for sending and receiving messages. Each mailbox has a unique id and processes can communicate only if they share a mailbox. Link established only if processes share a common mailbox and a single link can be associated with many processes. The standard primitives used are-

   - send(A,message)
   - receive(A,message)

   Mailbox A is under consideration.



Comparison

1. In Shared memory systems,there is a (common) shared address space throughout the system. In message passing, there is no shared address space among the systems.
2. Memory communication is faster on the shared memory model as compared to the message passing model as all processors share a single view of data and the communication between processors can be as fast as memory accesses to a same location. In message passing Communication depends on the capability of underlying network connecting the systems
3. Communication among processors takes place via shared data variables, and control variables for synchronization among the processors . In Message passing communication among processors takes place via send() and receive() statements.
4. Adding more CPUs can geometrically increase the traffic on the shared memory CPU path. Adding more systems to the network can increase the throughput but may lead to congestion in the link.
5. There is an overhead of maintaining cache coherency in shared memory. Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.

1.1 Questions on presentation

Question 1. how will a inter-process communication happen when a queue is shared between two process ?

- In shared memory there will be a single copy of queue shared by all the processes
- In message passing each process will have its own copy of the queue.

Question 2. How is the source and destination decided in indirect message passing?

- In indirect message passing both the sender and receiver must know the id(port) of the mailbox to communicate.
- A mailbox is comprised of two FIFO queues: a queue of unreceived messages and a queue of waiting processes.

Question 3. By linking in message passing, do you mean establishing TCP/UDP connection based on application?

- Yes, most implementations of MPI use TCP and sockets to communicate messages.

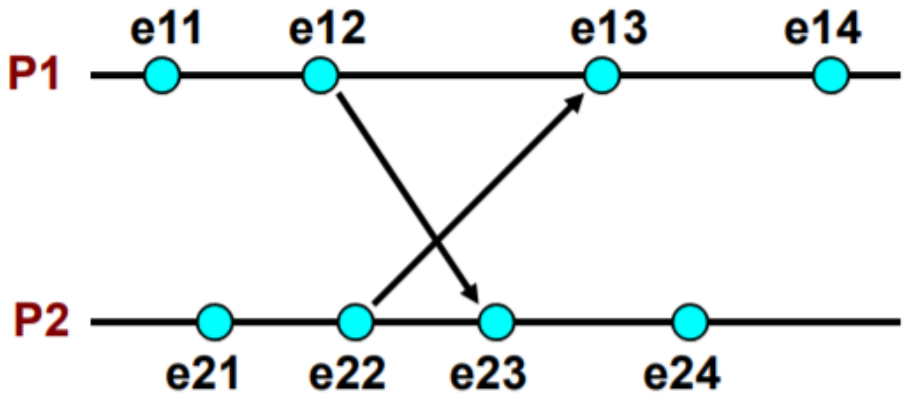
2 Class overview

Logical time A logical time is a mechanism for capturing chronological and causal relationships in a distributed system. **Causality** Let hi = e_i^1,e_i^2, ..., be the events at process Pi. A linear order Hi is a binary relation →i on the events hi such that $e_i^k \rightarrow_i e_i^j$ if and only if the event e_i^k occurs before the event e_i^j at Process Pi. The dependencies captured by $\rightarrow_i$ are often called as causal dependencies among the events hi at Pi. Binary relations $e_i^x \rightarrow e_j^y$ are causal iff for the following conditions

1. i = j and x ¡ y
2. e_i^x →msg e_j^y
3. For an e_k^z in H such that $e_i^x \rightarrow e_k^z$ and $e_k^z \rightarrow e_j^y$

Logical concurrency Two events are logically current if they don't causally affect each other. ei || ej Not(ei → ej) and Not(ej → ei)

Example for concurrency and causality:

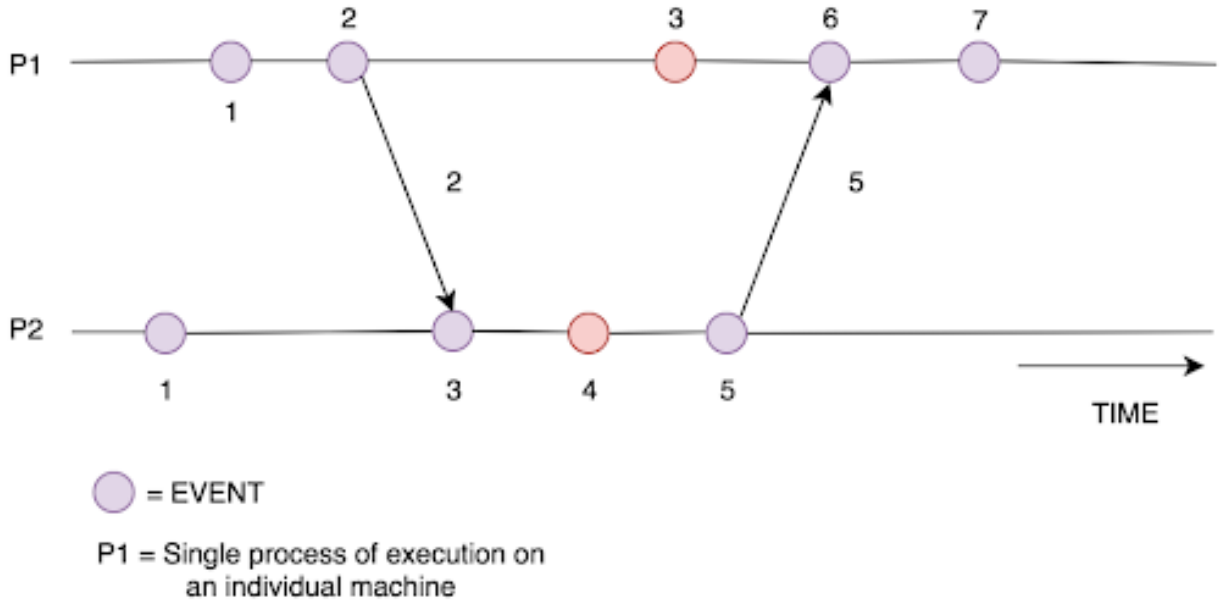


1. e11 and e21 are concurrent as they have no dependency on each other
2. e14 and e23 are concurrent if we go on to give numbering in here e23 can be completed first and then e14 vis a vis.
3. e22 causally affects e14 as e22→e13 and e13→e14 so e22→ e14.

Logical Clock C It is a function that maps an event e in a distributed system to an element in time domain T
Defined as: C: H→ T And for 2 events ei and ej ei ej implies c(ei) < c(ej) It is said to be consistent if it follows the above condition and it is said to be strictly consistent if it follows ei → ej $\iff$ c(ei) < c(ej)
Physical Clock
In this implementation of logical time, time domain is a set of non-negative integers with logical local clock and global time are combined into a integer C_i . There are 2 rules of scalar time

1. Before an event Pi executes Ci: = Ci + d (d>0) or local_clock = local_clock + 1
2. Each message piggybacks the clock value of its sender at sending time. When a process Pi receives a message with timestamp C_{msg}, it executes the following actions:

Ci := max(Ci, C_{msg}) local_clock = max(local_clock, received_clock)
Execute Rule1. Or local_clock = local_clock + 1
Deliver the message Deliver the message

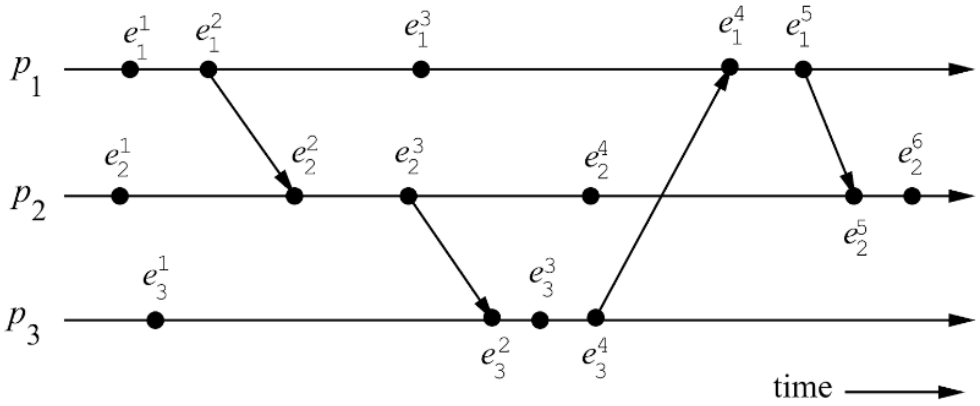


= EVENT

P1 = Single process of execution on an individual machine

January 11, 2021

2.1 Questions on class-work



Question 1. is e_3^1 and e_2^4 concurrent ?
consider the two valid ordering of events

1. e_1^1 , e_1^2 , e_2^2 , e_2^3 , e_3^1 , e_2^4
2. e_1^1 , e_1^2 , e_2^2 , e_2^3 , e_2^4 , e_3^1

As there is no causal dependency between e_3^1 and e_2^4, either of them could have happened before the other(as shown in above orderings) and hence they are concurrent.

Question 2. is e_3^1 and e_1^3 concurrent ?
consider the two valid ordering of events

1. e_1^1 , e_1^2 , e_2^2 , e_2^3 , e_3^1 , e_1^3
2. e_1^1 , e_1^2 , e_2^2 , e_2^3 , e_1^3 , e_3^1

As there is no causal dependency between e_3^1 and e_1^3, either of them could have happened before the other(as shown in above orderings) and hence they are concurrent.

Question 3. is $e_3^3 \rightarrow e_1^5$ causal ?

$$e^x_i \rightarrow e^y_j \text{ iff } \begin{cases} i = j \text{ and } x < y, & \text{or} \\ e^x_i \rightarrow_{msg} e^y_j, & \text{or} \\ \text{There exists } e^z_k \text{ in H s.t. } e^x_i \rightarrow e^z_k \text{ and } e^z_k \rightarrow e^y_j \end{cases}$$

- $e_3^3 \rightarrow e_3^4$ by rule 1
- $e_3^4 \rightarrow e_1^4$ by rule 2
- $e_1^4 \rightarrow e_1^5$ by rule 1
- hence by transitivity $e_3^3 \rightarrow e_1^5$

Question 4. is $e_1^2 \rightarrow e_2^3$ causal ?

- $e_1^2 \rightarrow e_2^2$ by rule 2
- $e_2^2 \rightarrow e_2^3$ by rule 1
- hence by transitivity $e_1^2 \rightarrow e_2^3$

Question 5. is $e_3^4 \rightarrow e_1^5$ causal ?

- $e_3^4 \rightarrow e_1^4$ by rule 2
- $e_1^4 \rightarrow e_1^5$ by rule 1
- hence by transitivity $e_3^4 \rightarrow e_1^5$

Question 6. is concurrency transitive ?

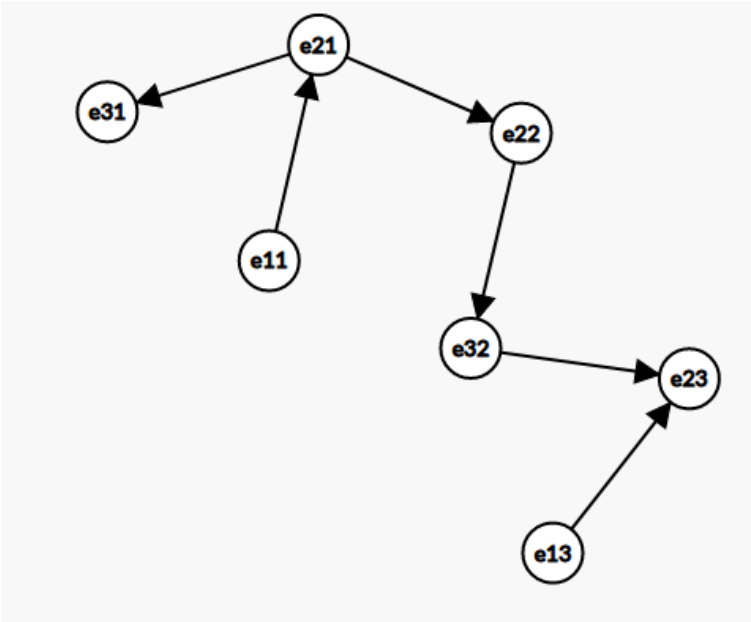
- No, as it can be seen in the figure $e_3^3 \parallel e_2^4$ and $e_2^4 \parallel e_1^5$ but e_3^3 is not concurrent with e_1^5.

3 Discussions

Discussion 1. using DAG(Directed Acyclic Graph) to identify causal and concurrent dependency

- Events are identified as nodes, and an edge from event a to b defines a direct causal dependency(by rule1 or rule2).
- If there is a path from an event a to event b then a $\rightarrow$ b (causal dependency) else they are concurrent.

A dag representation of few events from the above diagram



- As there is a path from e_1^1 to e_2^3 we can say $e_1^1 \rightarrow e_2^3$
- As there is no path from from e_1^3 to e_3^1 we can say $e_1^3 \parallel e_3^1$

3.1 Questions on discussion

Question 1. should the edges be weighted ?

- Adding weights to the edges do not provide any new information and hence can be avoided

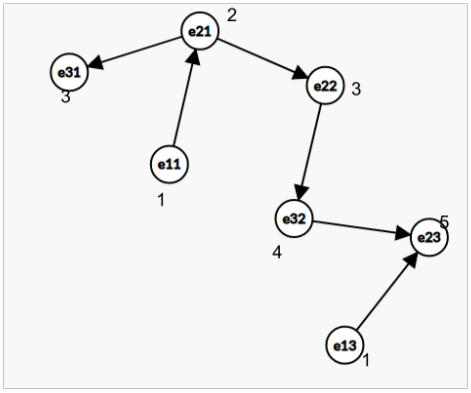
.

Question 2. how do you topologically order the concurrent events?

- For a dag multiple topological orderings are possible because for two nodes, who are not dependent on each other, can be placed in any order.Hence for concurrent events there exists topological ordering where position of those events are arbitrary.

Question 3. Will the bfs ordering of the DAG be total ordering?

BFS done by assigning unvisited node timestamp 1 and while visiting incrementing the timestamp of its parent in BFS traversal by 1.



- In the above image e_3^1 and e_2^3 are concurrent but still Timestamp(e_2^3) ¿ Timestamp(e_3^1).

4 Homework Problems

Homework-1 Prove scalar time is not strongly consistent

Homework-2 Suppose messages can be delivered out of order and we would like to deliver them in order. Can this be achieved with strong consistency

Homework-3 If a process recieves a message, how would the process know if a message of lower timestamp is on the way and yet to arrive?Will strong consistent broadcast messages resolve this issue?