Distributed Computing Module 1 -Lecture 5/6

Mahesh C.
Centre for High Performance Computing
FISAT

Logical vs. Physical Concurrency

- Two events are logically concurrent if and only if they do not causally affect each other.
- Physical concurrency has a connotation that the events occur at the same instant in physical time.
- Two or more events may be logically concurrent even though they do not occur at the same instant in physical time.

Models of Communication Network

- FIFO Model
- non-FIFO model
- Causal ordering Model (CO)

CO: 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})

Global State of a Distributed System

- The global state of a distributed system is a collection of the local states of the processes and the channels.

Process State

Let LS_i^x denote the state of process p_i after the occurrence of event e_i^x and before the event e_i^{x+1}. LS_i^0 denotes the initial state of process p_i. LS_i^x is a result of the execution of all the events executed by process p_i till e_i^x. Let $send(m) \leq LS_i^x$ denote the fact that $\exists y : 1 \leq y \leq x :: e_i^y = send(m)$. Likewise, let $rec(m) \not\leq LS_i^x$ denote the fact that $\forall y : 1 \leq y \leq x :: e_i^y \neq rec(m)$.

Global State of a Distributed System

The state of a channel is difficult to state formally because a channel is a distributed entity and its state depends upon the states of the processes it connects. Let $SC_{ij}^{x,y}$ denote the state of a channel C_{ij} defined as follows:

$$SC_{ij}^{x,y} = \{m_{ij} \mid send(m_{ij}) \leq e_i^x \bigwedge rec(m_{ij}) \not\leq e_j^y\}$$

Thus, channel state $SC_{ij}^{x,y}$ denotes all messages that p_i sent upto event e_i^x and which process p_j had not received until event e_j^y.

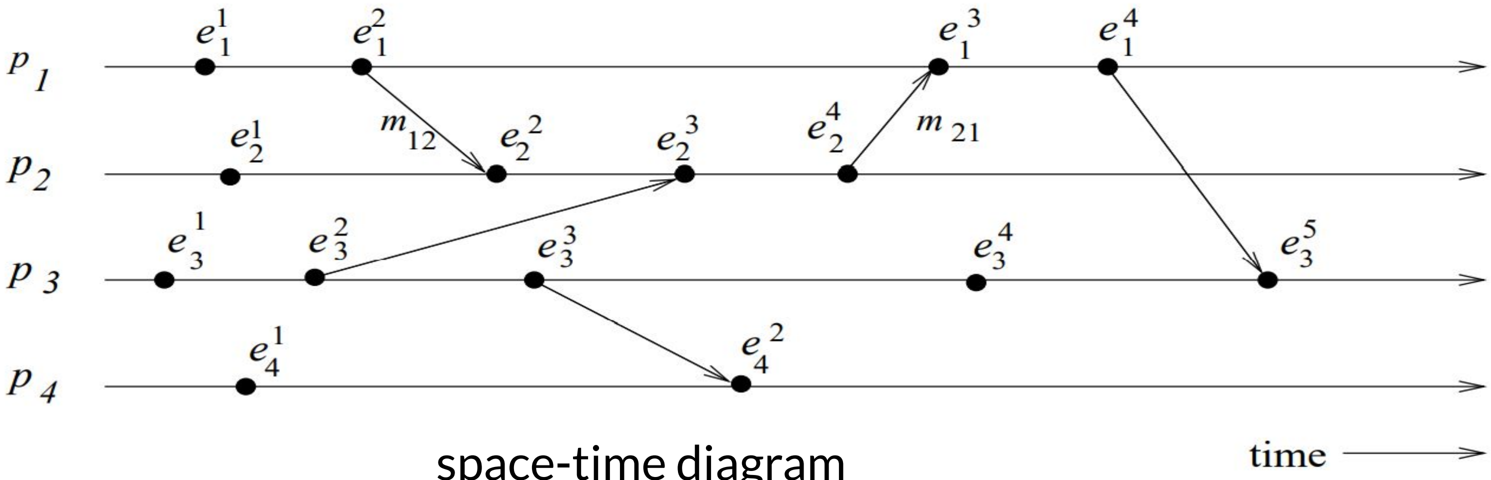
Global State of a Distributed System

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

$$GS = \{\bigcup_i LS_i^{x_i}, \bigcup_{j,k} SC_{jk}^{y_j, z_k}\}$$

Global State - consistent

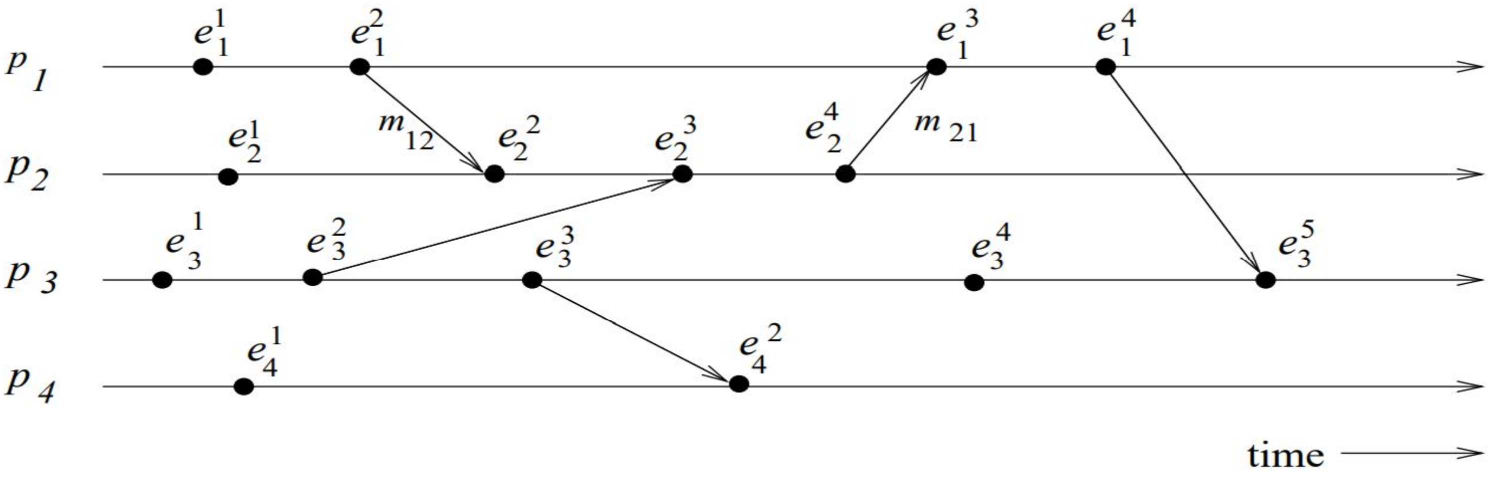
a state will be meaningful provided every message that is recorded as received is also recorded as sent.



space-time diagram

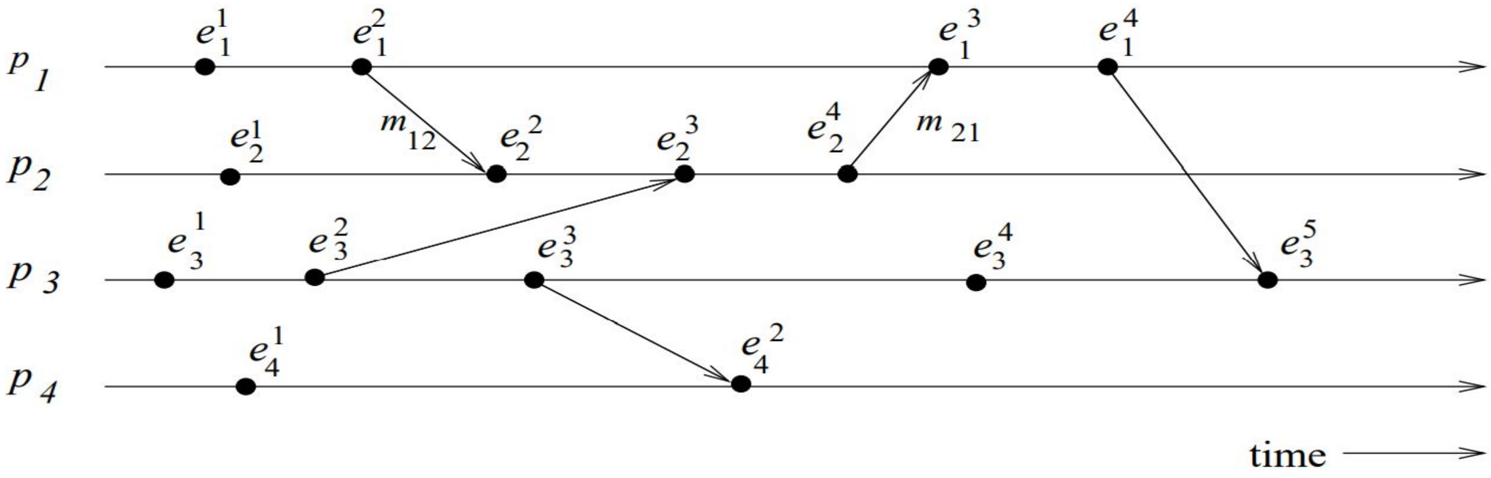
Global State - consistent or not

$$\{LS_1^1,\ LS_2^3,\ LS_3^3,\ LS_4^2\}$$



Global State - consistent or not

$$\{LS_1^2,\ LS_2^4,\ LS_3^4,\ LS_4^2\}$$

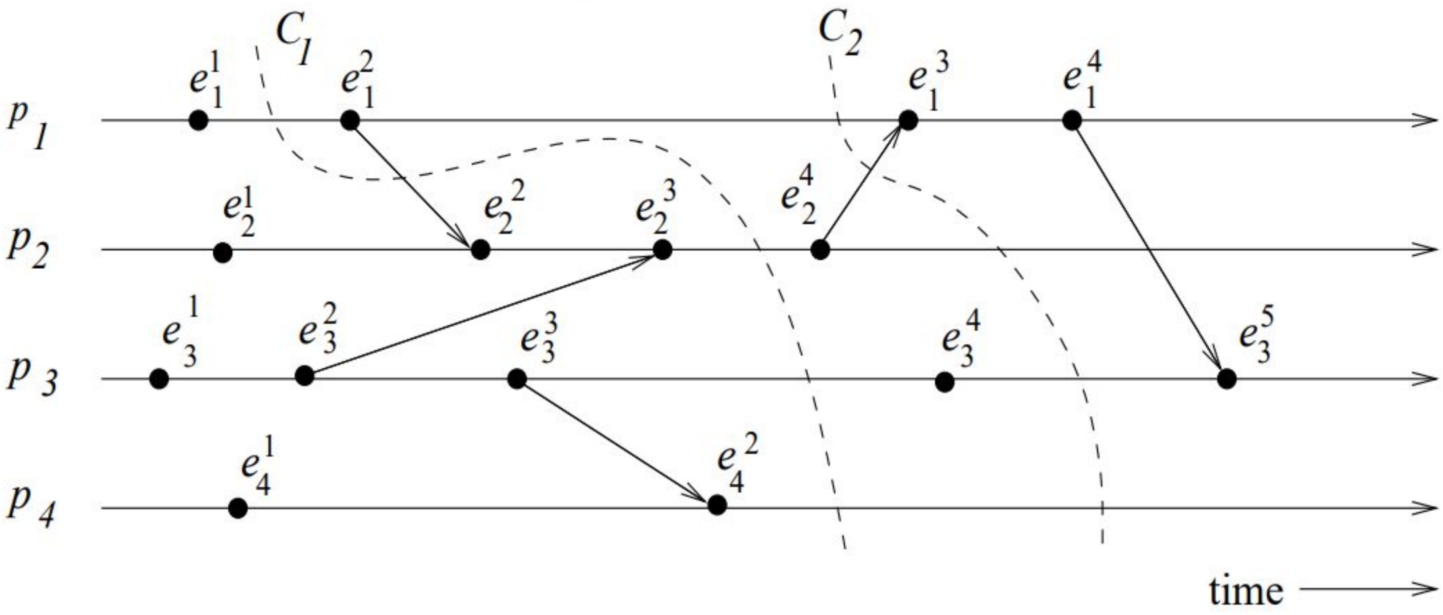


Cuts of a Distributed Computation

“In the space-time diagram of a distributed computation, a cut is a zigzag line joining one arbitrary point on each process line.”

- A cut slices the space-time diagram, and thus the set of events in the distributed computation, into a PAST and a FUTURE.
- The PAST contains all the events to the left of the cut and the FUTURE contains all the events to the right of the cut
- For a cut C, let PAST(C) and FUTURE(C) denote the set of events in the PAST and FUTURE of C, respectively
- Every cut corresponds to a global state and every global state can be graphically represented as a cut in the computation space-time diagram.
- Cuts in a space-time diagram provide a powerful graphical aid in representing and reasoning about global states of a computation.

Cuts of a Distributed Computation



Cuts of a Distributed Computation

"In the space-time diagram of a distributed computation, a cut is a zigzag line joining one arbitrary point on each process line."

- In a consistent cut, every message received in the PAST of the cut was sent in the PAST of that cut. (In, cut C2 is a consistent cut.)
- All messages that cross the cut from the PAST to the FUTURE are in transit in the corresponding consistent global state.
- A cut is inconsistent if a message crosses the cut from the FUTURE to the PAST. (In Figure, cut C1 is an inconsistent cut.)

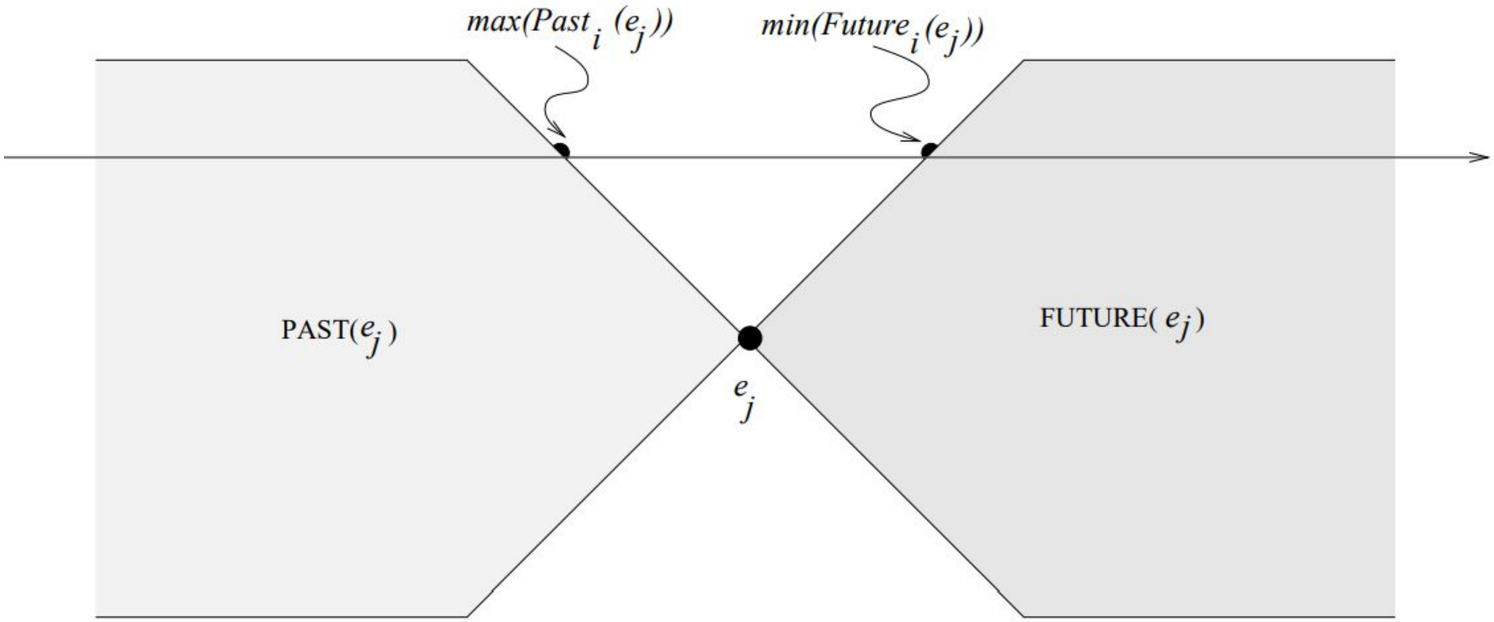
Past and Future Cones of an Event

"Past Cone of an Event"

- An event e_j could have been affected only by all events ei such that $e_i \rightarrow e_j$.
- In this situation, all the information available at e_i could be made accessible at e_j .
- All such events ei belong to the past of e_j . Let Past(e_j) denote all events in the past of e_j in a computation (H, →). Then

$$Past(e_j) = \{e_i | \forall e_i \in H, e_i \rightarrow e_j \}.$$

Past and Future Cones of an Event



Past and Future Cones of an Event

"Future Cone of an Event"

- The future of an event e_j , denoted by Future(e_j), contains all events ei that are causally affected by e_j
- In a computation (H, →), Future(e_j) is defined as:

$$Future(e_j) = \{e_i | \forall e_i \in H, \; e_j \; \rightarrow e_i\}.$$

Models of Process Communications

- There are two basic models of process communications – synchronous and asynchronous.
- The synchronous communication model is a blocking type where on a message send, the sender process blocks until the message has been received by the receiver process.
- The sender process resumes execution only after it learns that the receiver process has accepted the message.
- Thus, the sender and the receiver processes must synchronize to exchange a message.

Models of Process Communications

- Asynchronous communication model is a non-blocking type where the sender and the receiver do not synchronize to exchange a message.
- After having sent a message, the sender process does not wait for the message to be delivered to the receiver process.
- The message is buffered by the system and is delivered to the receiver process when it is ready to accept the message.

Models of Process Communications

- Asynchronous communication provides higher parallelism because the sender process can execute while the message is in transit to the receiver.
- However, A buffer overflow may occur if a process sends a large number of messages in a burst to another process.
- An implementation of asynchronous communication requires more complex buffer management. In addition, due to higher degree of parallelism and non-determinism, it is much more difficult to design, verify, and implement distributed algorithms for asynchronous communications.
- Synchronous communication is simpler to handle and implement. However, due to frequent blocking, it is likely to have poor performance and is likely to be more prone to deadlocks.