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

Course Name: Distributed Systems

Course Code: CSE751

Credits: 3:0:0

Term: September – December 2020

Faculty:
Sini Anna Alex

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's 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

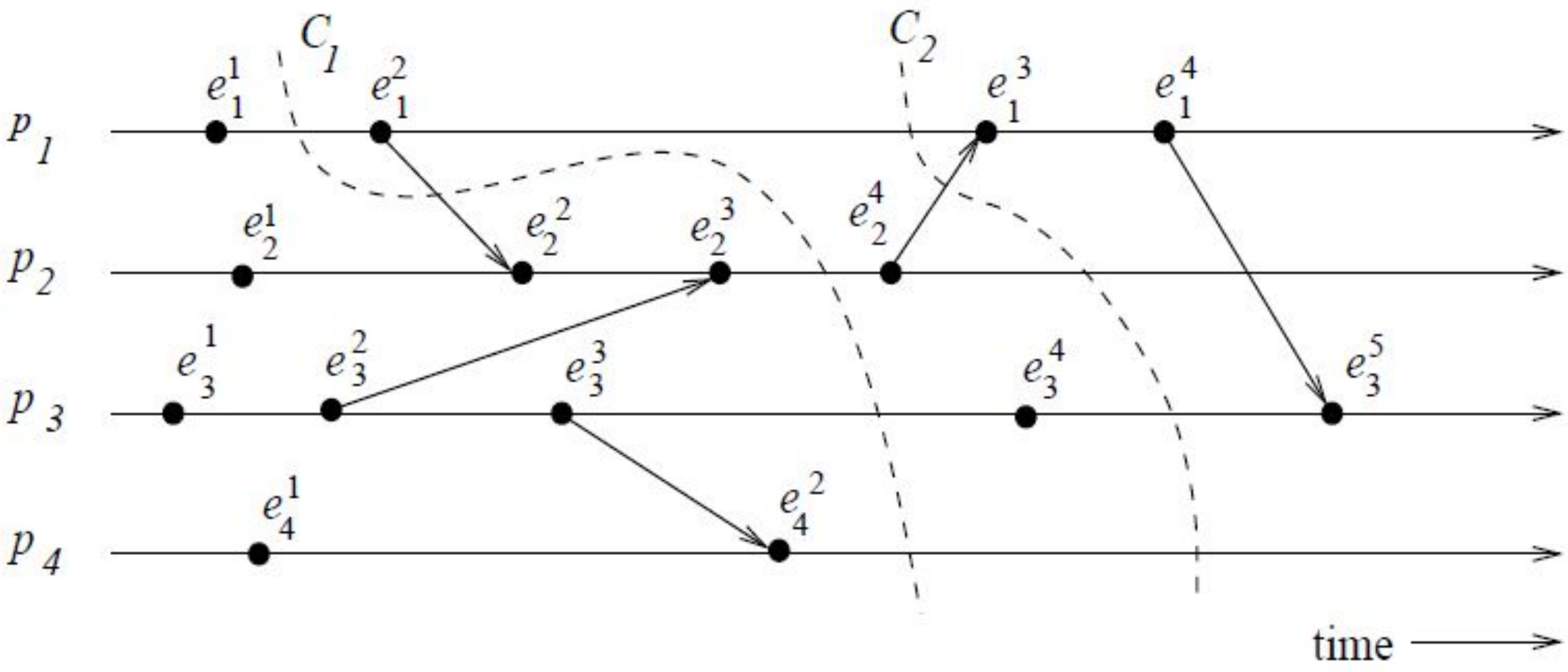
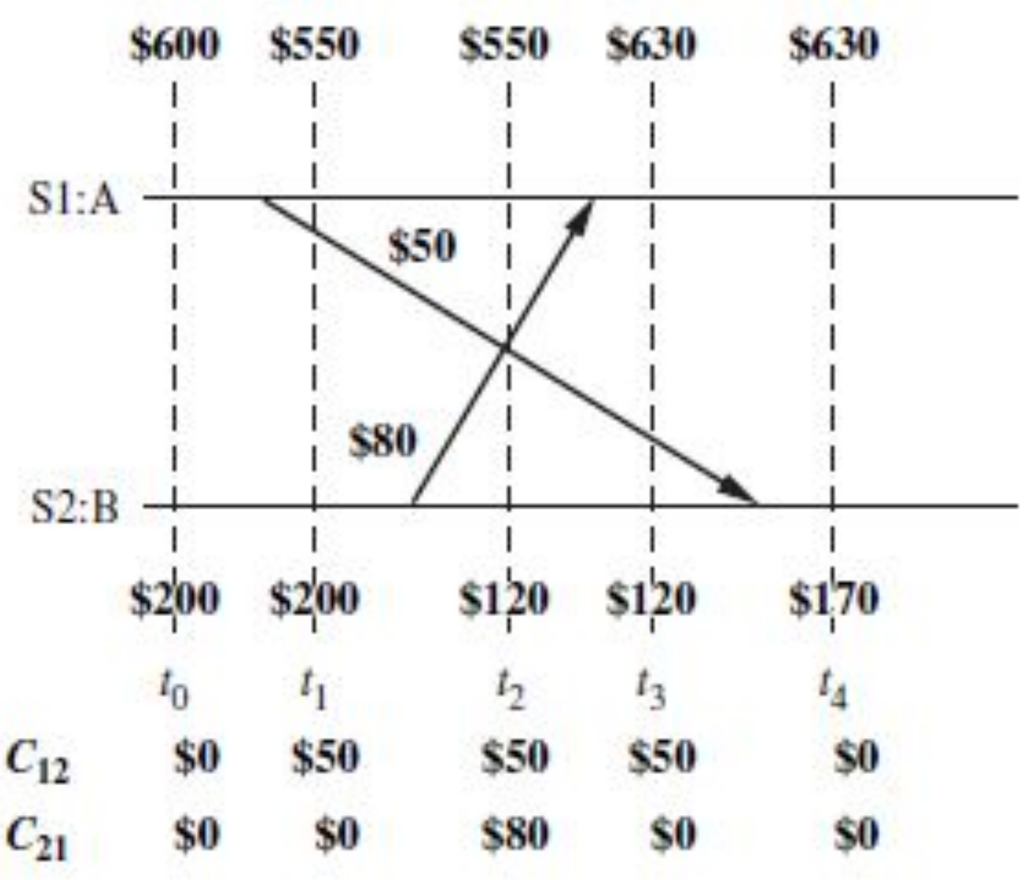


Illustration of cuts in a distributed execution

Cuts of a Distributed Computation

- In a consistent cut, every message received in the PAST of the cut was sent in the PAST of that cut. (In Figure 2.3, cut C_2 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 2.3, cut C_1 is an inconsistent cut.)

Illustration of consistent states



Past and Future Cones of an Event

Past Cone of an Event

- An event e_j could have been affected only by all events e_i such that $e_i \rightarrow e_j$.
- In this situtaion, all the information available at e_i could be made accessible at e_j.
- All such events e_i belong to the past of e_j.

Let $Past(e_j)$ denote all events in the past of e_j in a computation $(H, \rightarrow)$. Then,

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

- Figure 2.4 (next slide) shows the past of an event e_j.

Jard–Jourdan’s adaptive technique

Direct-dependency technique(The Fowler–Zwaenepoel) does not allow the transitive dependencies to be captured in real time during the execution of processes.

In addition, **a process must observe an event after receiving a message but before sending out any message**. Otherwise, during the reconstruction of a **vector timestamp from the direct-dependency vectors, all the causal dependencies will not be captured**.

If events occur very frequently, this technique will require recording the history of a large number of events.

Jard–Jourdan’s adaptive technique

In the Jard–Jourdan’s technique, events can be adaptively observed while maintaining the capability of retrieving all the causal dependencies of an observed event. (Observing an event means recording of the information about its dependencies.)

This method uses the idea that when an observed event e records its dependencies, then events that follow can determine their transitive dependencies of an observed event.

Jard–Jourdan’s adaptive technique

Jard–Jourdan defined a *pseudo-direct* relation << on the events of a distributed computation as follows:

If events e_i and e_j happen at process p_i and p_j, respectively, then $e_j \ll e_i$ iff there exists a path of message transfers that starts after e_j on the process p_j and ends before e_i on the process e_i such that there is no observed event on the path.

Technique is implemented using the following mechanism

Initially, at a process p_i: $p\_vt_i=\{(i, 0)\}$.

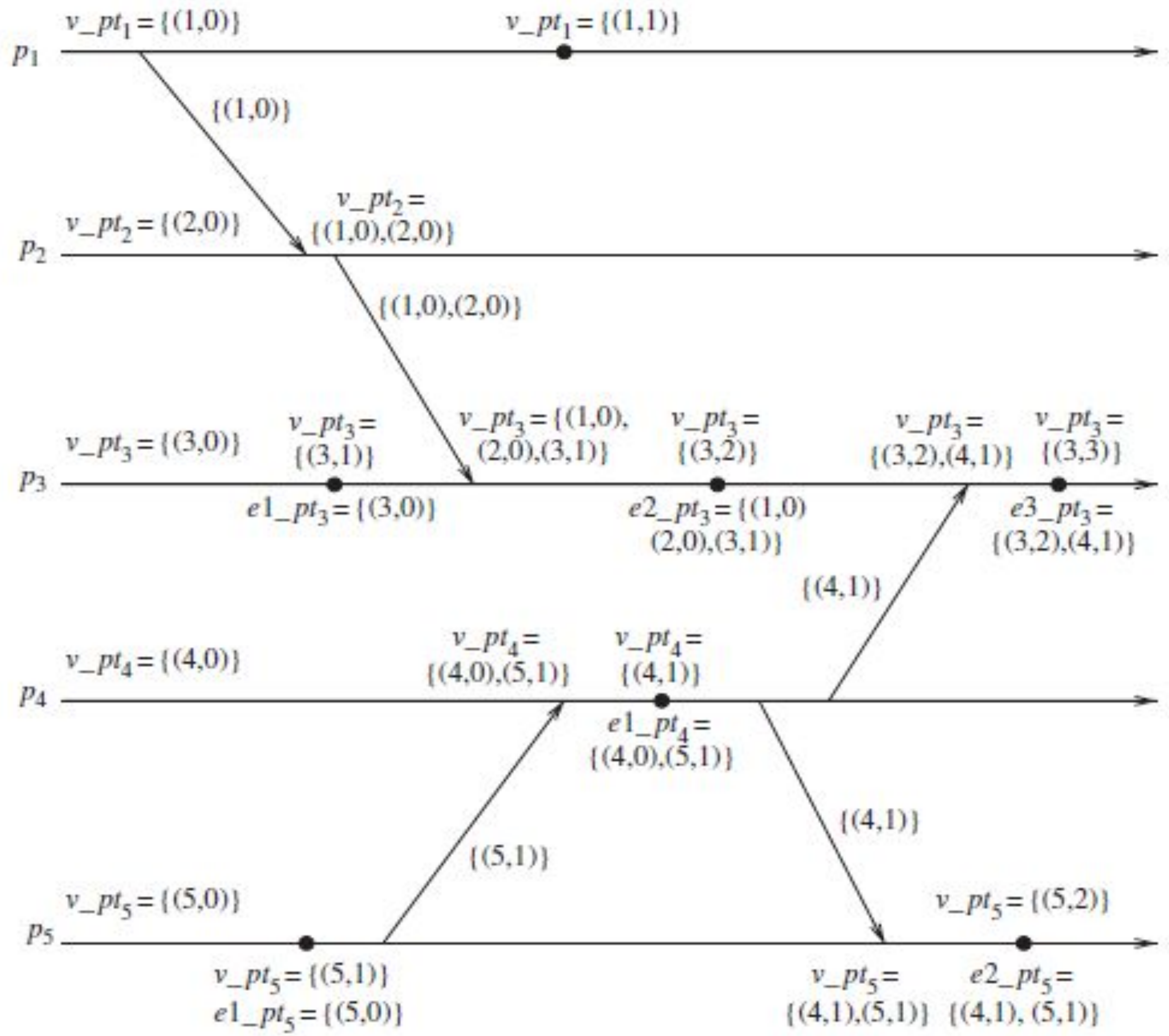
Let $p\_vt_i = \{(i_1, v_1), \ldots, (i, v), \ldots (i_n, v_n\}$ denote the current partial vector clock at process p_i. Let e_vt_i be a variable that holds the timestamp of the observed event.

Whenever an event is observed at process p_i, the contents of the partial vector clock p_vt_i are transferred to e_vt_i and p_vt_i is reset and updated as follows:

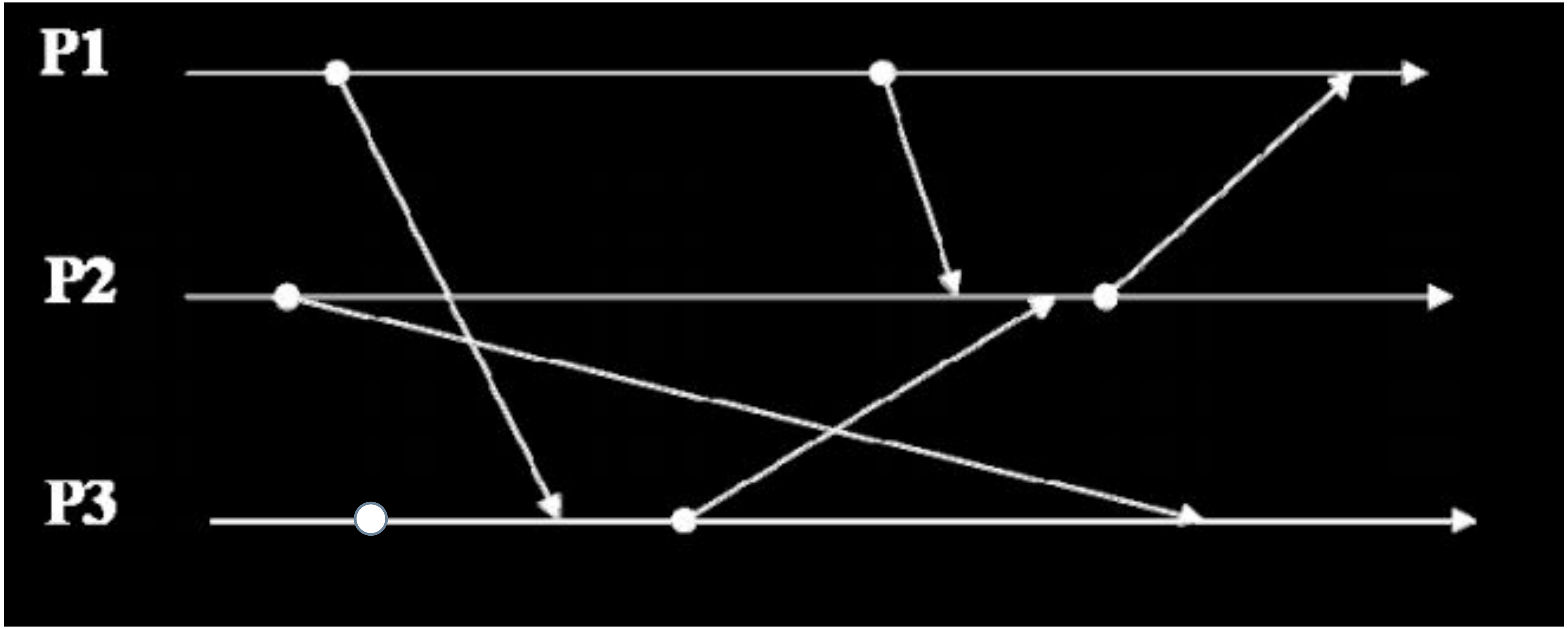
$$e\_vt_i = \{(i_1, v_1), \ldots, (i, v), \ldots, (i_n, v_n)\}$$
$$p\_vt_i = \{(i, v+1)\}.$$

When process p_j sends a message to p_i, it piggybacks the current value of p_vt_j in the message.

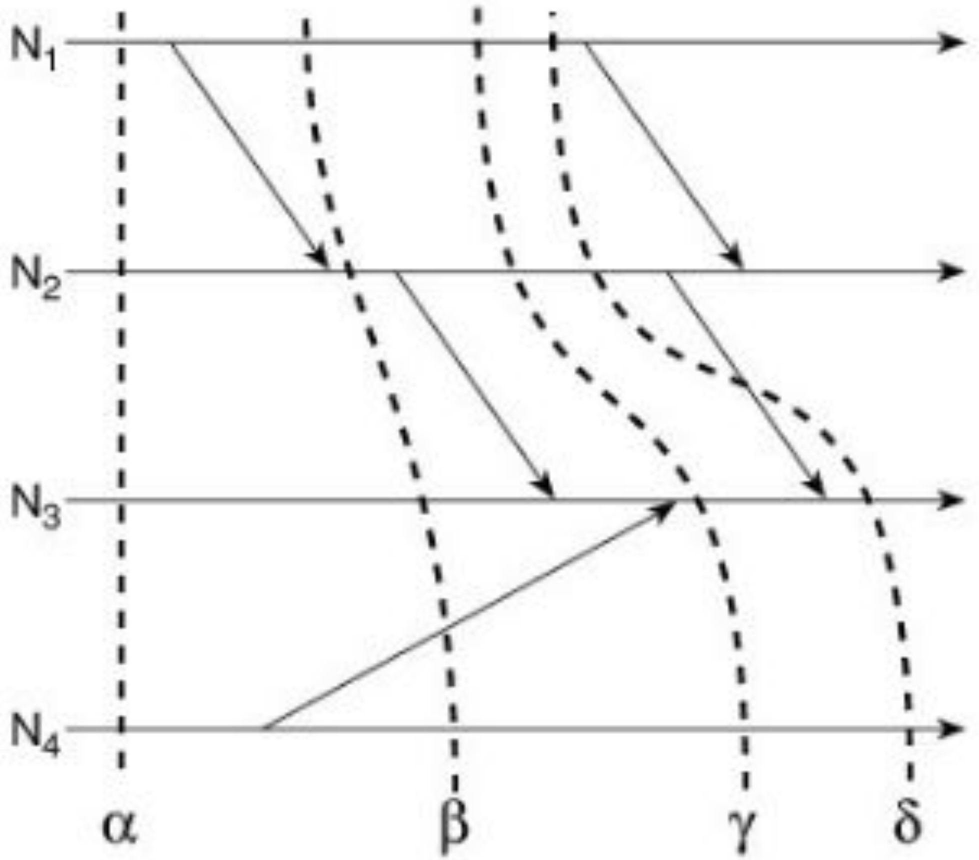
Vector clocks progress in the Jard–Jourdan technique



Jard–Jourdan's adaptive technique



Identify the consistent and inconsistent cuts in distributed computation



Text Book

Ajay D. Kshemkalyani and MukeshSinghal, *Distributed Computing: Principles, Algorithms, and Systems*, Cambridge University Press.

Thank you