Time and Global States

based on Distributed Systems: Concepts and Design, Edition 5

Ali Fanian

Isfahan University of Technology
www.Fanian.iut.ac.ir

Titles

- *Introduction*
- *Clocks, events and process states*
- *Synchronizing physical clocks*
- *Logical time and logical clocks*
- *Global states*
- *Distributed debugging*
- *Summary*

Why time?

- *Time is important because:*
  1. *time is a quantity we often want to measure accurately, e.g. in e-commerce transactions*
  2. *consistency of distributed data, checking the authenticity of a request sent to a server, eliminating the processing of duplicate updates*
- *But time is Problematic in DS.*
- *Each computer has its own physical clock, clocks typically deviate, we cannot synchronize them perfectly.*

What we study?

- *We will examine algorithms for synchronization physical clocks using message passing*
- *We will study logical clocks: vector clocks.*
- *We will also look at algorithms to capture global states of DS as they execute.*

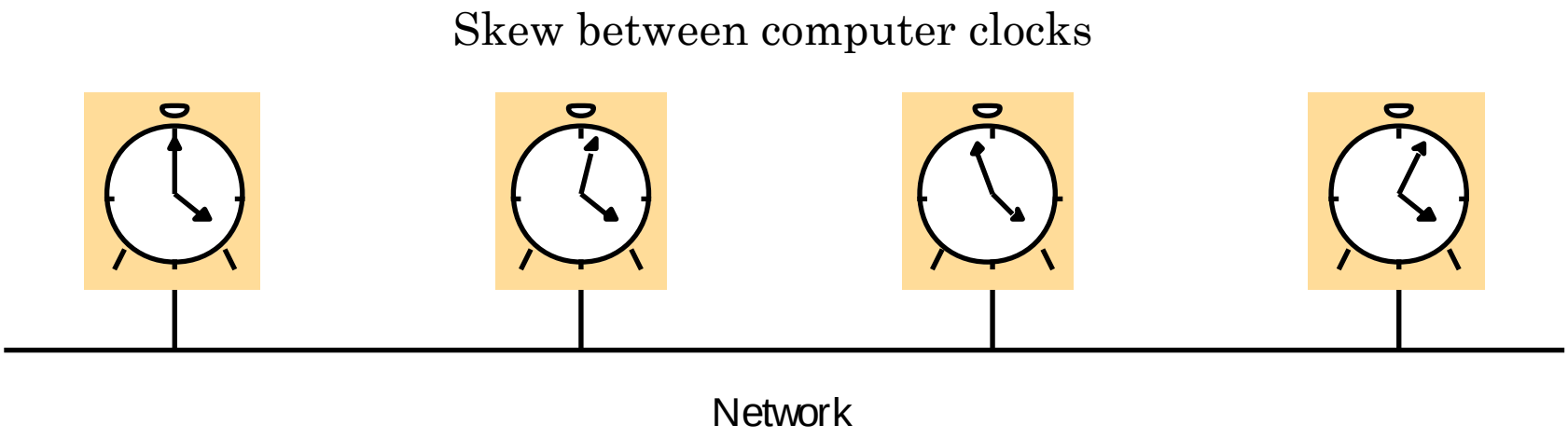
Events

- *As each process p_i executes it takes a series of actions such as:*
  1. *message send or receive operation*
  2. *an operation that transforms p_i 's state*
- *We define an event to be the occurrence of a single action that a process carries out as it executes – a communication action or a state-transforming action.*
- *The sequence of events within a single process p_i denote by the relation $\longrightarrow i$ between the events.*
- *That is, $e \longrightarrow i\ e'$ if and only if the event e occurs before e' at p_i .*
- *history of process p_i:*

$$history(p_i) = h_i = \langle e_i^0, e_i^1, e_i^2, \ldots \rangle$$

Clock skew and clock drift

- *The instantaneous difference between the readings of any two clocks is called their skew.*
- *Clock drift means that clocks count time at different rates.*



Coordinated Universal Time

- *Computer clocks can be synchronized to external sources.*
- *The most accurate physical clocks use atomic oscillators, whose drift rate is about one part in* 10^{13}.
- *Coordinated Universal Time – abbreviated as UTC – is an international standard for timekeeping.*
- *UTC signals are synchronized and broadcast regularly from land based radio stations and satellites.*
- *Satellite sources include the Global Positioning System (GPS).*
- *Signals received from land-based stations have an accuracy on the order of 0.1–10 milliseconds.*
- *Signals received from GPS satellites are accurate to about 1 microsecond.*
- *Computers with receivers attached can synchronize their clocks with these timing signals.*

Synchronizing physical clocks

- *In order to know at what time of day events occur, it is necessary to synchronize the processes' clocks, C_i*
- *If we synchronize them with an authoritative, external source of time, it is external synchronization.*
- *And if the clocks C_i are synchronized with one another to a known degree of accuracy, this is internal synchronization.*
- *External synchronization: For a synchronization bound $D > 0$, and for a source S of UTC time, $|S(t) - C_i(t)| < D$, for $i = 1, 2, \ldots N$ and for all real times t in I.*
- *Internal synchronization: For a synchronization bound $D > 0$,*

Synchronizing physical clocks

- *Clocks that are internally synchronized are not necessarily externally synchronized.*
- *If the system P is externally synchronized with a bound D, then the same system is internally synchronized with a bound of 2D.*
- *We define a hardware clock H to be correct if its drift rate falls within a known bound $\rho > 0$, (such as 10^6 seconds/second).*

Internal Synchronization

- *One process sends the time t on its local clock to the other in a message m.*
- *Receiving process could set its clock to the time* $t + T_{trans}$
- T_{trans} *is the time taken to transmit m between them.*
- T_{trans} *is unknown. Because:*
  1. *other processes are competing for resources with the processes to be synchronized*
  2. *other messages compete with m for the network resources*

Synchronization in a synchronous system

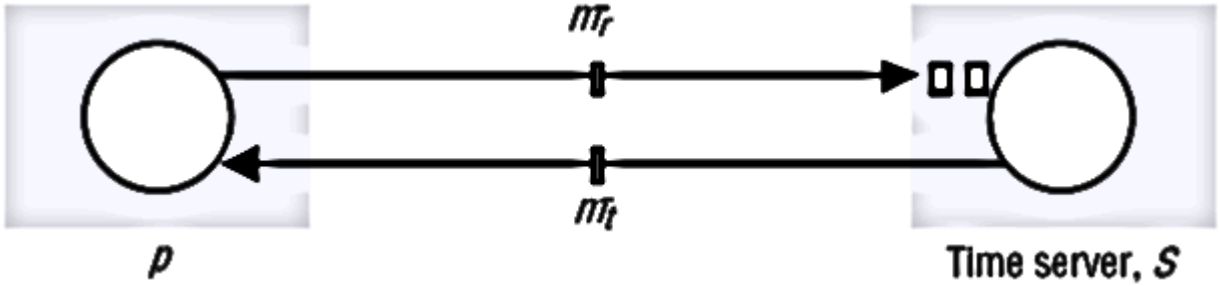
- *There is always a minimum transmission time, min.*
- *It is obtained if no other processes executed and no other network traffic existed.*
- *There is also an upper bound max on the time taken to transmit any message.*
- *Uncertainty in the message transmission time = u,*

$$u = (max – min)$$

- *If the receiver sets its clock to be t + min, then the clock skew may be as much as u.*
- *If it sets its clock to t + max, the skew may be as large as u.*
- *If it sets its clock to the halfway point, t + (max + min) / 2 , then the skew is at most u / 2.*

Cristian's method for synchronizing clocks

- *Cristian suggested the use of a time server.*
- *Time server connected to a device that receives signals from a source of UTC.*
- *Upon request, the server process S supplies the time according to its clock.*



- *Process p records the total round-trip time T_{round} taken to send the request m_r and receive the reply m_t .*

Cristian's method for synchronizing clocks

- *The method achieves synchronization only if the observed round-trip times between client and server are sufficiently short compared with the required accuracy.*
- *A simple estimate of the time to which p should set its clock is* $t + T_{round} / 2$.
- *This is accurate assumption, unless the two messages are transmitted over different networks.*

Cristian's method for synchronizing clocks

- *If min is known, then the accuracy is as follows:*
- *The earliest point at which S could have placed the time in* m_t *was min after p dispatched* m_r.
- *The latest point at which it could have done this was min before* m_t *arrived at p.*
- *The time when reply message arrives is in the range*

$$[t + min,\ t + T_{round} - min]$$

- *The width of this range is* $T_{round} - 2min$ *, so the accuracy is*

$$\pm(T_{round} \ / \ 2 - min)$$

Discussion of Cristian's algorithm

- ***Single time server*** *might fail and thus render synchronization temporarily impossible.*
  - *Cristian suggested, that time should be provided by a group of synchronized time servers, each with a receiver for UTC time signals.*
  - *For example, a client could multicast its request to all servers and use only the first reply obtained.*
- ***Faulty time server****: These problems were beyond the scope of the work described by Cristian, which assumes that sources of external time signals are self-checking.*

The Berkeley algorithm

- *Gusella and Zatti describe an algorithm for internal synchronization.*
- *A coordinator computer is chosen to act as the **master**.*
- *This computer periodically polls the other computers whose clocks are to be synchronized, called **slaves**.*
- *The slaves send back their clock values to it.*
- *The master estimates their local clock times by observing the round-trip times and it averages the values obtained.*
- *Instead of sending the updated current time back to the other computers, the master sends the amount by which each individual slave's clock requires adjustment.*
- *This can be a positive or negative value.*

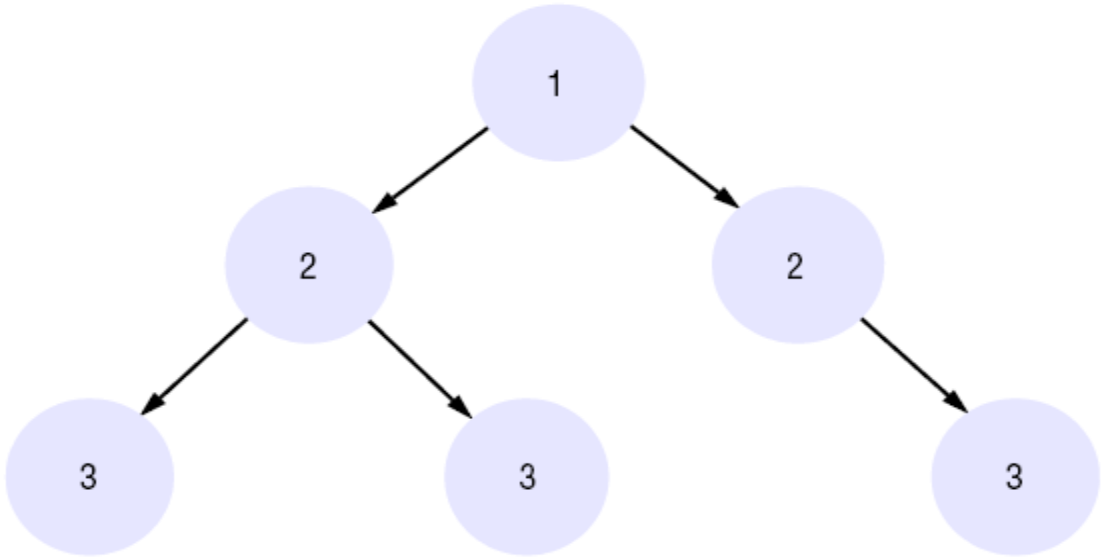
The Berkeley algorithm

- *The Berkeley algorithm eliminates readings from faulty clocks.*
  - *A subset is chosen of clocks that do not differ from one another by more than a specified amount, and the average is taken of readings from only these clocks.*
- *If the master fail, then another can be elected.*
- *These are not guaranteed to elect a new master in bounded time, so the difference between two clocks would be unbounded.*

The network time protocol

- *Cristian's method and the Berkeley algorithm are intended primarily for use within intranets.*
- *The NTP defines an architecture for a time service and a protocol to distribute time information over the Internet.*

The network time protocol



- *The NTP service is provided by a network of servers located across the Internet. Primary servers are connected directly to a time source such as a radio clock receiving UTC; secondary servers are synchronized, ultimately, with primary servers.*
- *Stratum 2 servers are secondary servers that are synchronized directly with the primary servers;*
- *stratum 3 servers are synchronized with stratum 2 servers, and so on.*
- *The lowest-level (leaf) servers execute in users' workstations.*
- *The servers are connected in a logical hierarchy called a synchronization subnet whose levels are called strata.*

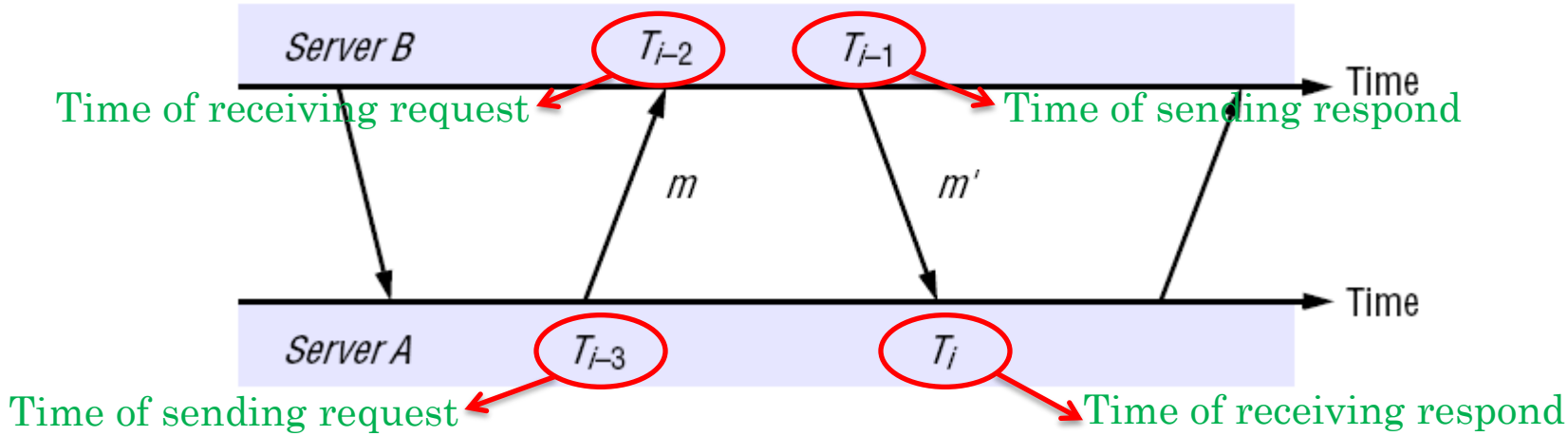
The network time protocol

- *Multicast mode is intended for use on a high-speed LAN. One or more servers periodically multicasts the time to the other computers connected by the LAN, which set their clocks assuming a small delay.*

- *Procedure-call mode is similar to the operation of Cristian's algorithm. In this mode, one server accepts requests from other computers, which it processes by replying with its timestamp. This mode is suitable where higher accuracies are required than can be achieved with multicast, or where multicast is not supported in hardware.*

- *Symmetric mode is intended for use by the servers that supply time information in LANs and by the higher levels (lower strata) of the synchronization subnet, where the highest accuracies are to be achieved. A pair of servers operating in symmetric mode exchange messages bearing timing information.*

The network time protocol



- *In procedure-call mode and symmetric mode, processes exchange pairs of messages.*
- *Each message bears timestamps of recent message events.*

Happend-before relation

- *From the point of view of any single process, events are ordered uniquely by times shown on the local clock.*
- *since we cannot synchronize clocks perfectly across a distributed system, we cannot in general use physical time to find out the order of any arbitrary pair of events occurring within it.*
- *For ordering in DS, there is two points:*
  1. *If two events occurred at the same process p_i ($i = 1, 2, \ldots N$), then they occurred in the order in which p_i observes them ($\longrightarrow_i$).*
  2. *Whenever a message is sent between processes, the event of sending the message occurred before the event of receiving the message.*
- *These two relationships are called the happened-before relation (or causal ordering or potential causal ordering).*

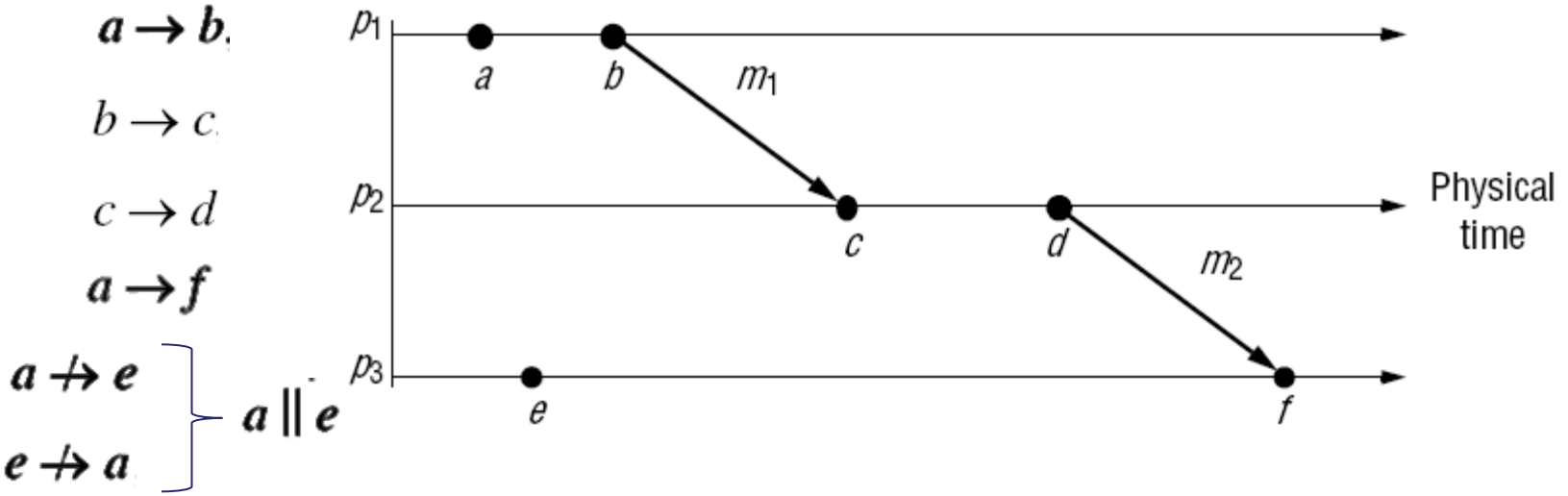
Happend-before relation

- *happened-before relation, denoted by* $\longrightarrow$

HB1: If $\exists$ process $p_i : e \rightarrow_i e'$, then $e \rightarrow e'$.

HB2: For any message m, $send(m) \rightarrow receive(m)$
– where $send(m)$ is the event of sending the message, and $receive(m)$ is the event of receiving it.

HB3: If e, e' and e'' are events such that $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$.



Logical clocks

- *Lamport invented a simple mechanism by which the happened-before ordering can be captured numerically, called a logical clock. A Lamport logical clock is a monotonically increasing software counter.*
- *Each process p_i keeps its own logical clock, L_i, which it uses to apply so-called Lamport timestamps to events.*
- *We denote the timestamp of event e at pi by $L_i(e)$, and by $L(e)$ we denote the timestamp of event e at whatever process it occurred at.*
- *Processes update their logical clocks and transmit the values of their logical clocks in messages as follows:*

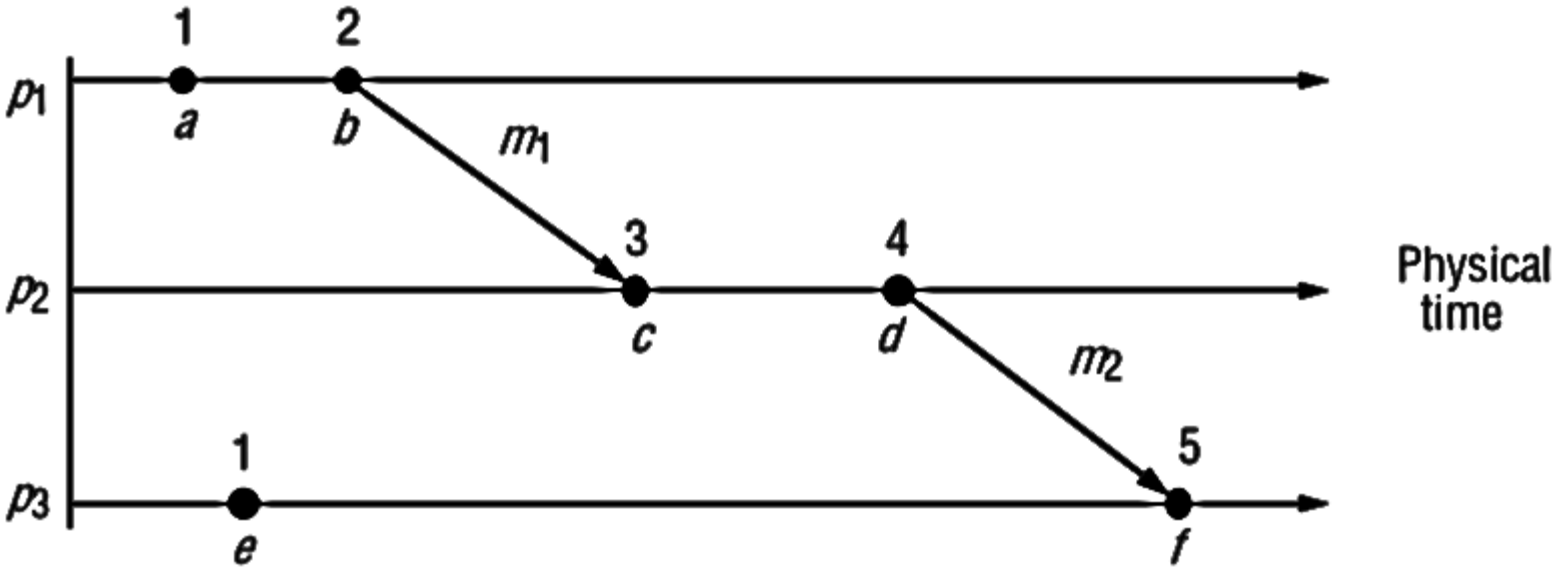
LC1: *Li* is incremented before each event is issued at process *pi* : *Li* := *Li* + 1.
LC2:

(a) When a process *pi* sends a message *m*, it piggybacks on *m* the value *t* = *Li* .

(b) On receiving (*m*, *t*), a process *pj* computes *Lj* := *max*(*Lj*, *t*) and then applies LC1 before timestamping the event *receive*(*m*).

Logical clocks

- *Although we increment clocks by 1, we could have chosen any positive value.*
- *If $e \longrightarrow e'$ then $L(e) < L(e')$*
- *The converse is not true. If $L(e) < L(e')$ then we can not say $e \longrightarrow e'$*



$L(b) > L(e)$ but $b \parallel e$

Logical clocks

- *Some pairs of distinct events, generated by different processes, have numerically identical Lamport timestamps.*
- *However, we can create a total order on the set of events.*
- *If e is an event occurring at p_i with local timestamp T_i, and e´ is an event occurring at p_j with local timestamp T_j , we define the global logical timestamps for these events to be (T_i, i) and (T_j, j), respectively. And we define $(T_i, i) < (T_j, j)$ if and only if either $T_i < T_j$, or $T_i = T_j$ and $i < j$.*

Vector clocks

- *the fact that from $L(e) < L(e')$ we cannot conclude that $e \longrightarrow e'$*
- *A vector clock for a system of N processes is an array of N integers.*
- *Each process keeps its own vector clock, V_i, which it uses to timestamp local events.*
- *Like Lamport timestamps, processes piggyback vector timestamps on the messages they send to one another, and there are simple rules for updating the clocks.*

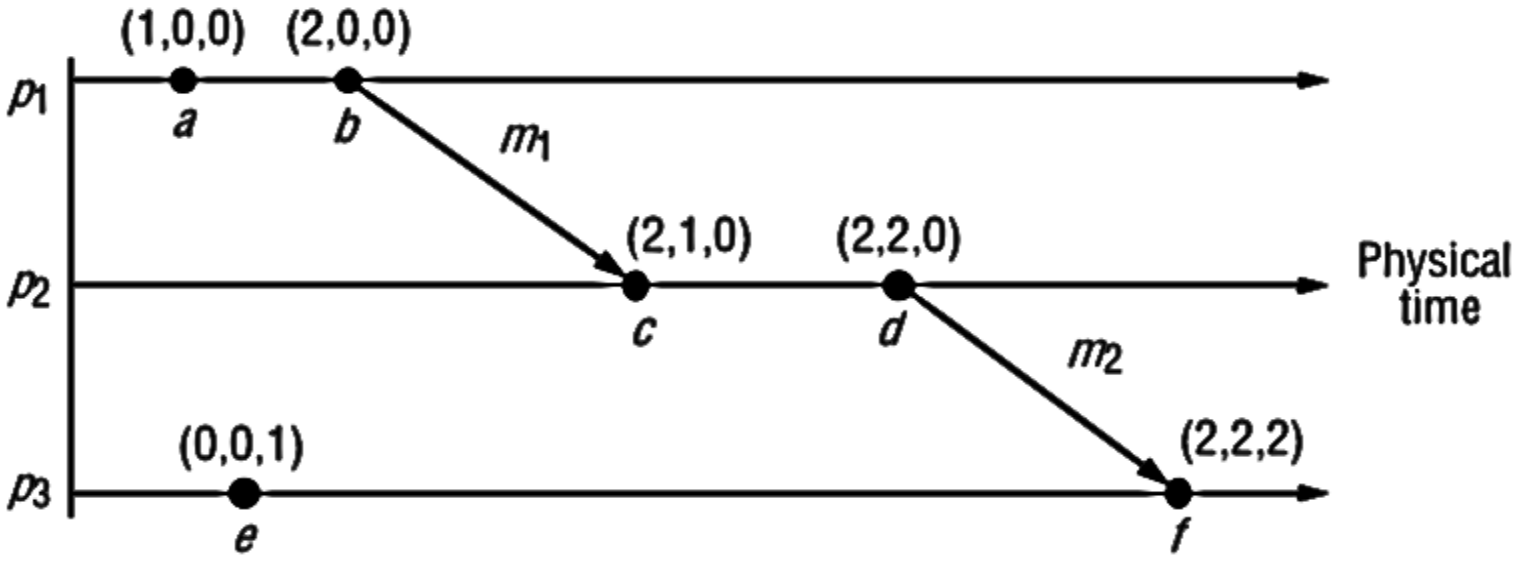
Vector clocks

VC1: Initially, $V_i[j] = 0$, for $i, j = 1, 2..., N$.

VC2: Just before p_i timestamps an event, it sets $V_i[i] := V_i[i] + 1$.

VC3: p_i includes the value $t = V_i$ in every message it sends.

VC4: When p_i receives a timestamp t in a message, it sets $V_i[j] := max(V_i[j], t[j])$, for $j = 1, 2..., N$.



Vector clocks

- *For a vector clock V_i, $V_i[i]$ is the number of events that p_i has timestamped, and $V_i[j]$ $(j \neq i)$ is the number of events that have occurred at p_i that have potentially affected p_i.*
- *Comparing vector timestamps:*

$$V = V' \text{ iff } V[j] = V'[j] \text{ for } j = 1, 2..., N$$

$$V \leq V' \text{ iff } V[j] \leq V'[j] \text{ for } j = 1, 2..., N$$

$$V < V' \text{ iff } V \leq V' \wedge V \neq V'$$

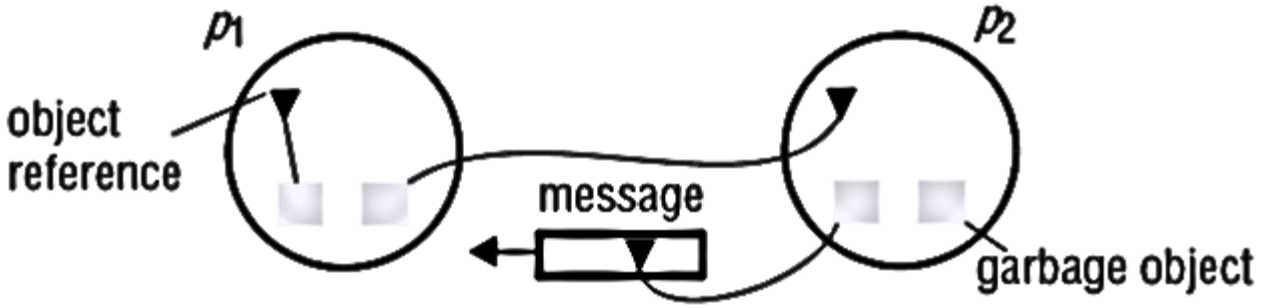
- *In previous example:*
- *$V(a) < V(f)$ which reflects the fact that $a \longrightarrow f$.*
- *Neither $V(c) \leq V(e)$ nor $V(e) \leq V(c)$ then $c \,||\, e$*

Vector clocks

- *Vector timestamps have the disadvantage, compared with Lamport timestamps, of taking up an amount of storage and message payload that is proportional to N, the number of processes.*
- *However, techniques exist for storing and transmitting smaller amounts of data.*

Garbage collection

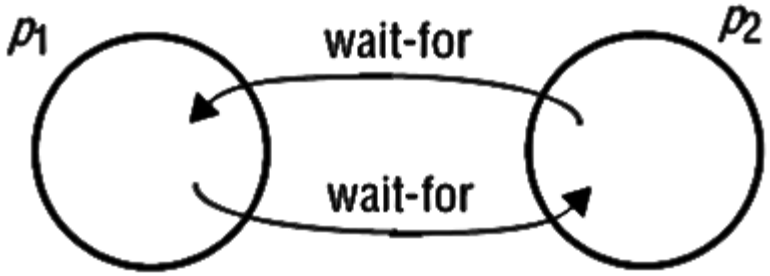
- ***Distributed garbage collection***: An object is considered to be garbage if there are no longer any references to it anywhere in the DS.
- The memory taken up by that object can be reclaimed once it is known to be garbage.



- When we consider properties of a system, we must include the state of communication channels as well as the state of the processes.

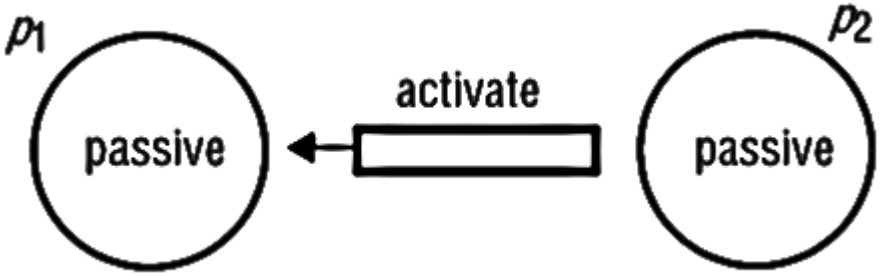
Deadlock detection

- ***Distributed deadlock detection***: A distributed deadlock occurs when each of a collection of processes waits for another process to send it a message, and where there is a cycle in the graph of this 'waits-for' relationship.



Termination detection

- ***Distributed termination detection**: The problem here is how to detect that a distributed algorithm has terminated.*
- *It sounds easy to solve at first: it seems at first only necessary to test whether each process has halted.*
- *But this is not so.*
- *A process is either active or passive.*
- *a passive process is not engaged in any activity of its own but is prepared to respond with a value requested by the other.*



Debugging

- ***Distributed debugging**: Distributed systems are complex to debug.*
- *For example, suppose Smith has written an application in which each process p_i contains a variable x_i ($i = 1, 2..., N$). The variables change as the program executes, but they are required always to be within a value δ of one another.*
- *Unfortunately, there is a bug in the program, and Smith suspects that under certain circumstances $| x_i - x_j | > \delta$ for some i and j.*
- *Her problem is that this relationship must be evaluated for values of the variables that occur at the same time.*

Global states & Consistent cuts

- *A series of events occurs at each process*

$$history(p_i) = h_i = \langle e_i^0, e_i^1, e_i^2, \ldots \rangle$$

- *Any finite prefix of the process's history:* $h_i^k = \langle e_i^0, e_i^1, \ldots e_i^k \rangle$
- *Each event either is an internal action of the process (for example, the updating of one of its variables), or is the sending or receipt of a message over the communication channels.*
- *Each process can record the events that take place there, and the succession of states it passes through.*
- s_i^k*: the state of process* p_i *immediately before the kth event occurs, so that* s_i^0 *is the initial state of* p_i*.*
- *Processes should record the sending or receipt of all messages as part of their state.*

Global states & Consistent cuts

- ***Global history*** *of P is defined as the union of the individual process histories:*

$$H = h_0 \cup h_1 \cup \ldots \cup h_{N-1}$$

- *Any set of states of the individual processes is a* ***global state***

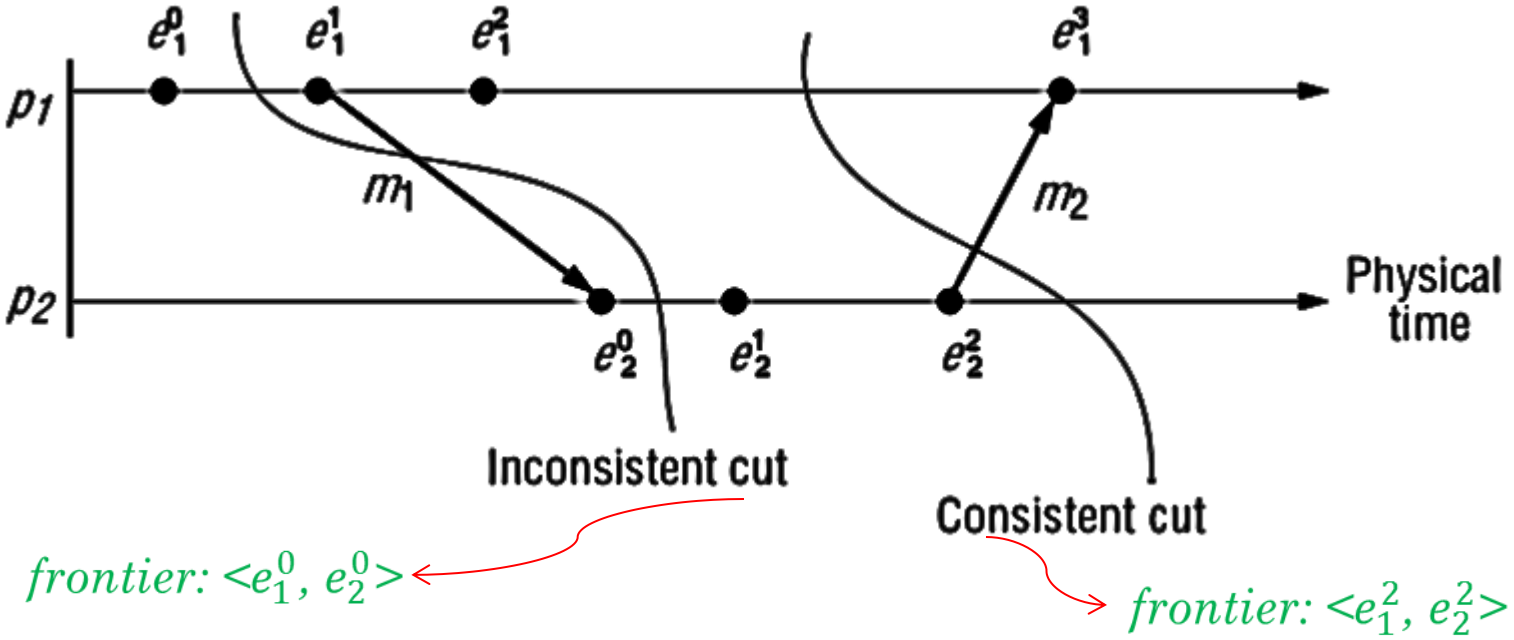
$$S = (s_1, s_2, \ldots s_N)$$

- *A* ***cut*** *of the system's execution is a subset of its global history that is a union of prefixes of process histories:*

$$C = h_1^{c_1} \cup h_2^{c_2} \cup \ldots \cup h_N^{c_N}$$

- *The set of events* $\{e_i^{c_i} : i = 1, 2, \ldots, N\}$ *is called the frontier of the cut.*

Consistent cuts



- *A cut C is **consistent** if, for each event it contains, it also contains all the events that happened-before that event:*

  For all events $e \in C$, $f \longrightarrow e$ then $f \in C$

- *The leftmost cut is inconsistent. This is because at p_2 it includes the receipt of the message m_1, but at p_1 it does not include the sending of that message. This is showing an 'effect' without a 'cause'.*

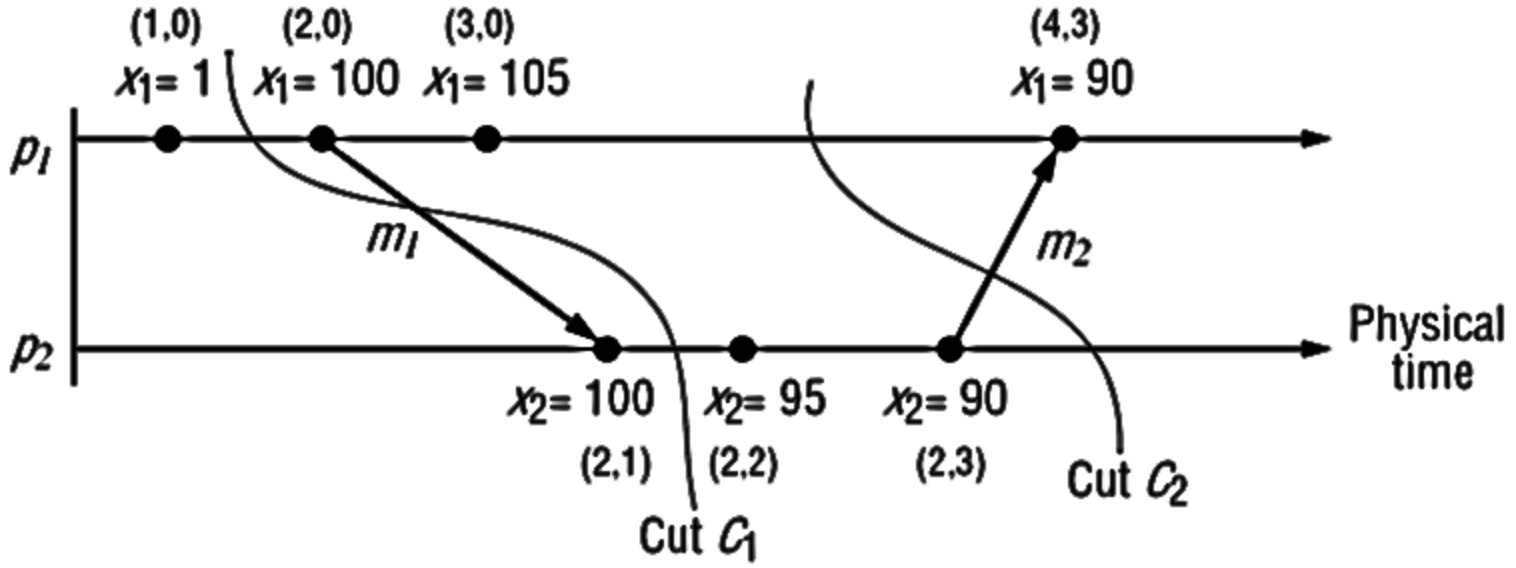
Distributed debugging

- *The challenge is to monitor the system's execution over time – to capture 'trace' information rather than a single snapshot.*
- *Snapshot algorithm collects state in a distributed fashion, and the processes in the system could send the state they gather to a monitor process for collection.*
- *Next algorithm is centralized. The observed processes send their states to a process called a monitor, which assembles globally consistent states from what it receives.*

Collecting the state

- *The observed processes p_i (i = 1, 2, …N) send their initial state to the monitor initially, and thereafter from time to time, in state messages.*
- *The monitor records the state messages from each process p_i in a separate queue Q_i, for each i=1, 2, …N.*
- *There is no need to send the state except initially and when it changes.*

Observing consistent global states



- *The requirement is* $|x_1 - x_2| \leq 50$
- *In order that the monitor can distinguish consistent global states from inconsistent global states, the observed processes enclose their vector clock values with their state messages.*

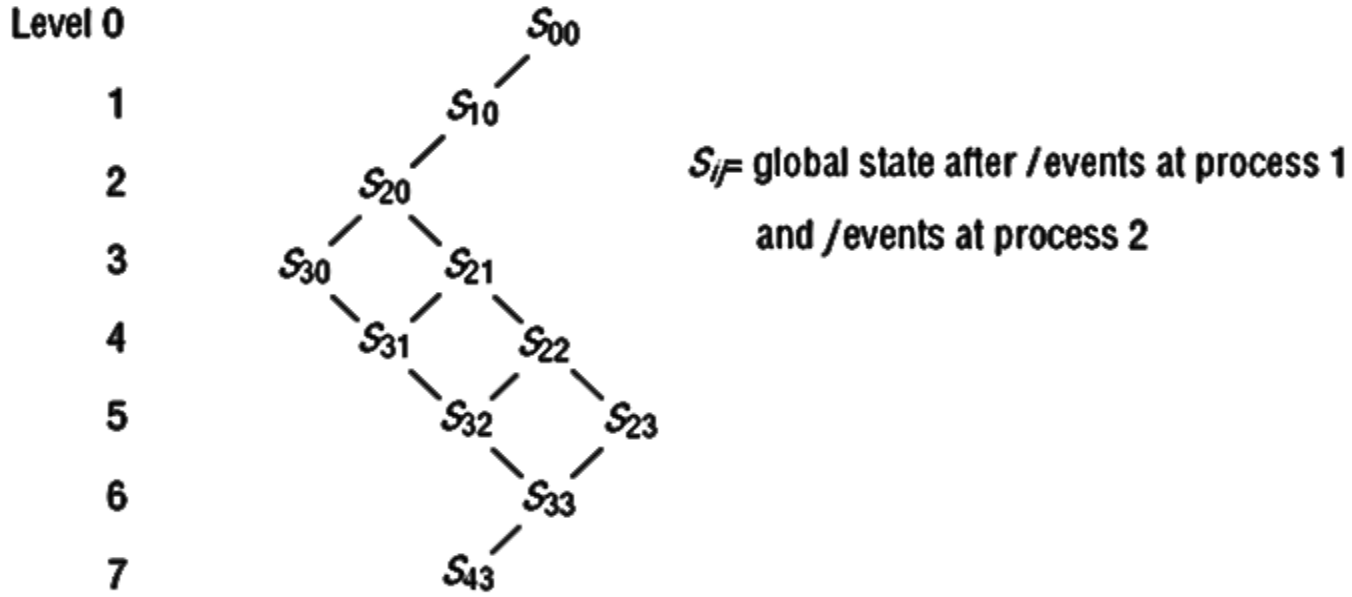
Observing consistent global states

- *$S = (s_1, s_2, \ldots, s_N)$ is a global state drawn from the state messages that the monitor has received.*
- *$V(s_i)$ is the vector timestamp of the state s_i received from p_i. Then it can be shown that S is a consistent global state if and only if:*

$$V(s_i)[i] \geq V(s_j)[i] \text{ for } i, j = 1, 2, \ldots, N$$

- *This says that the number of p_i's events known at p_j when it sent s_j is no more than the number of events that had occurred at p_i when it sent s_i.*

Observing consistent global states



- *The global state S_{00} has both processes in their initial state. S_{10} has p_2 still in its initial state and p_1 in the next state in its local history.*
- *This structure captures the relation of reachability between consistent global states.*
- *The state S_{01} is not consistent, because of the message m_1 sent from p_1 to p_2, so it does not appear in the lattice.*
- *The nodes denote global states, and the edges denote possible transitions between these states.*
- *The lattice is arranged in levels with, for example, S_{00} in level 0 and S_{10}*

Summary

- *Describing the importance of accurate timekeeping for DS.*
- *We then described* ***algorithms for synchronizing*** *clocks despite the drift between them and the variability of message delays between computers.*
- *The* ***happened-before*** *relation within a process, or via messages between processes*
- ***Lamport clocks*** *are counters that are updated in accordance with the happened-before relationship between events.*
- ***Vector clocks*** *are an improvement on Lamport clocks.*

Summary

- *The concepts of events has been introduced, local and global histories, cuts, local and global states, runs, consistent states, linearizations (consistent runs) and reachability.*
- *We went on to give algorithm that employs a monitor process to collect states. The monitor examines vector timestamps to extract consistent global states.*