Clock Synchronization in a Distributed System

Synchronized Distributed Clocks

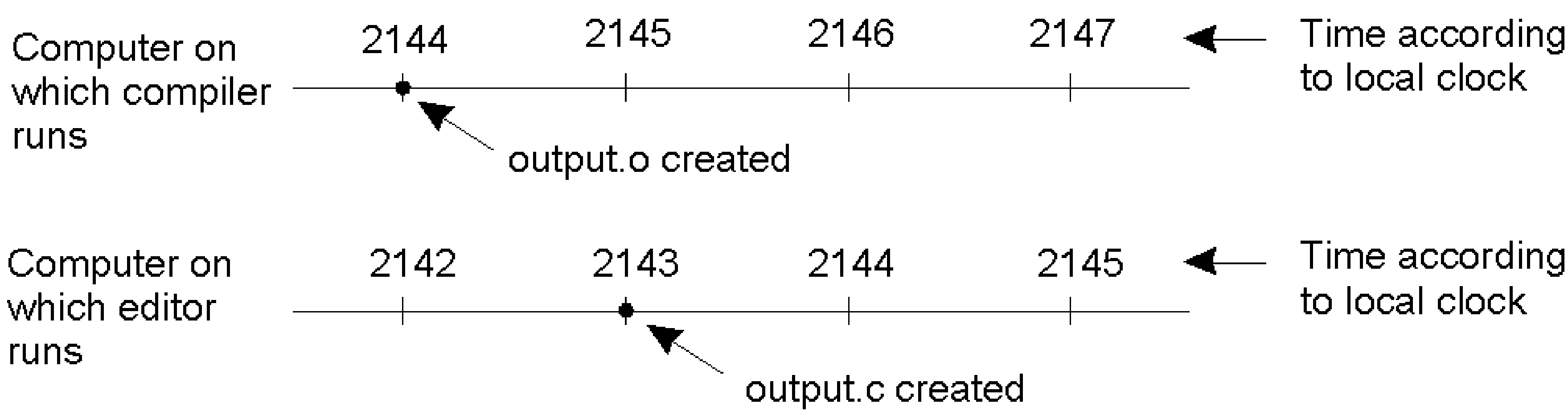
NEED??

- *Time driven systems:* in statically scheduled systems activities are started at "precise" times in different points of the distributed system.
- *Time stamps:* certain events or messages are associated with a time stamp showing the actual time when they have been produced; certain decisions in the system are based on the "exact" time of the event or event ordering.
- Calculating the duration of activities: if such an activity starts on one processor and finishes on another (e.g. transmitting a message), calculating the duration needs clocks to be synchronized.

Lack of Global Time in DS

- It is impossible to guarantee that physical clocks run at the same frequency
- Lack of global time, can cause problems
- Example: UNIX make
  - Edit output.c at a client
  - output.o is at a server (compile at server)
  - Client machine clock can be lagging behind the server machine clock

Lack of Global Time – Example



When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Physical Clock

- Every computer is equipped with CMOS clock circuit. These are electronic devices that count oscillations occurring in a crystal.
- Also called timer, usually a quartz crystal, oscillating at a well defined frequency.
- Timer is associated with two registers: A Counter and a Holding Register, counter decreasing one at each oscillations.
- When counter reaches zero, an interrupt is generated; this is the *clock tick*.
- Clock tick have a frequency of 60-100 ticks per second.

Drifting of Clock (cont'd)

- The problems:

1. Crystals cannot be tuned perfectly. Temperature and other external factors can also influence their frequency.

Clock drift: the computer clock differs from the real time.

2. Two crystals are never identical.

Clock skew: the computer clocks on different processors of the distributed system show different time.

Clock Synchronization Algorithms

☞ ***Distributed Algorithms***

- There is no particular time server.

- The processors periodically reach an agreement on the clock value by averaging the time of neighbors clock and its local clock.

  -This can be used if no UTC receiver exists (no external synchronization is needed). Only internal synchronization is performed.

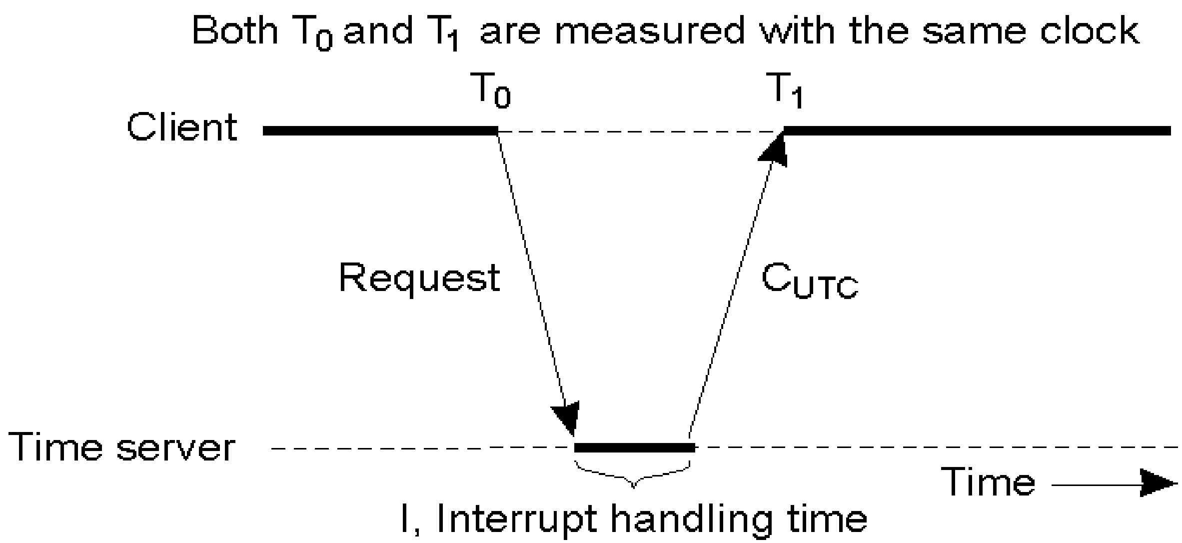
  - Processes can run on different machines and no global clock to judge which event happens first.

Cristian's Algorithm

- ***Cristian's Algorithm is centralized algorithm.***

- The simplest algorithm for setting time, it issues a Remote Procedure Call to time server and obtain the time.
- A machine sends a request to time server in "d/2" seconds, where d=max difference between a clock and UTC.
- The time server sends a reply with current UTC when receives the request.
- The machine measures the time delay between time server sending the message and machine receiving it. Then it uses the measure to adjust the clock.

Cristian's Algorithm



- The best estimate of message propagation time= $(T_0 + T_1)/2$
- The new time can be set to the time returned by server plus time that elapsed since server generated the timestamp:

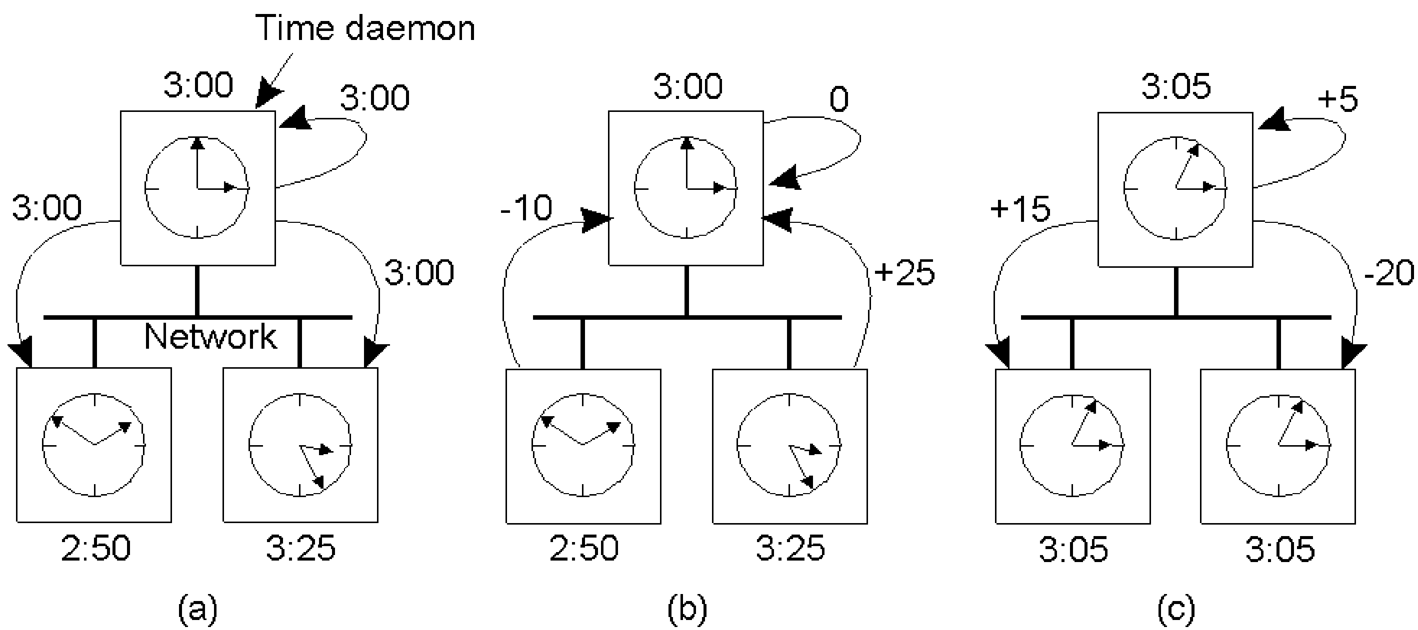
$$T_{new} = T_{server} + (T_0 + T_1)/2$$

Berkeley Algorithm

- *It is also a Centralized algorithm and its Time server is an Active Machine.*

- The server polls each machine periodically, asking it for the time.
- When all the results are in, the master computes the average time.
- Instead of sending the updated time back to slaves, which would introduce further uncertainty due to network delays, it sends each machine the offset by which its clock needs adjustment.
- If master machine fails, any other slave could be elected to take over.

Berkeley Algorithm



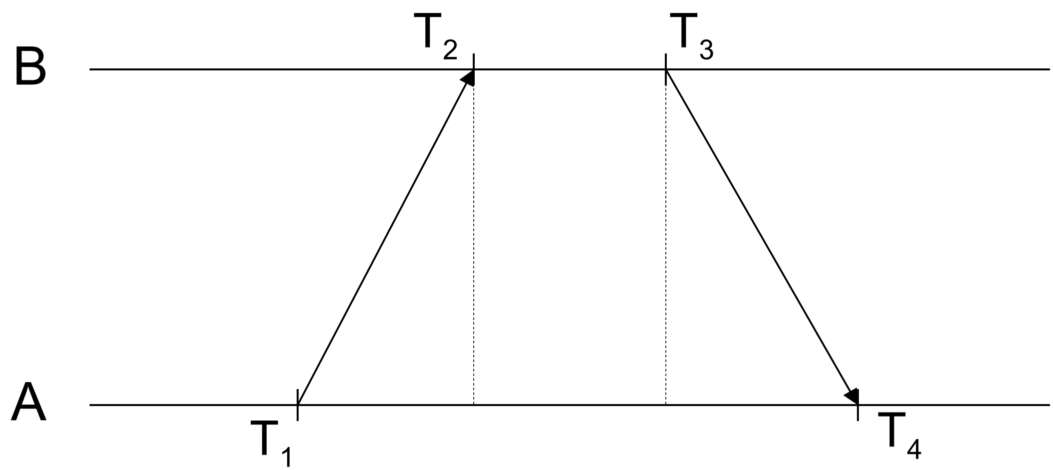
a) The time daemon sends synchronization query to other machines in group.
b) The machines sends timestamps as a response to query.
c) The Server averages the three timestamps and tells everyone how to adjust their clock by sending offsets.

Network Time Protocol

- NTP is an application layer protocol
- Default port number 123
- Uses standard UDP Internet transport protocol
- Adjust system clock as close to UTC as possible over the Internet.
- Enable sufficiently frequently resynchronizations. (scaling well on large num
- The NTP service is provided by a network of servers located across the Internet.
- Primary servers are connected directly to a time source. (e.g. a radio clock receiving UTC, GPS). bers of clients and servers)

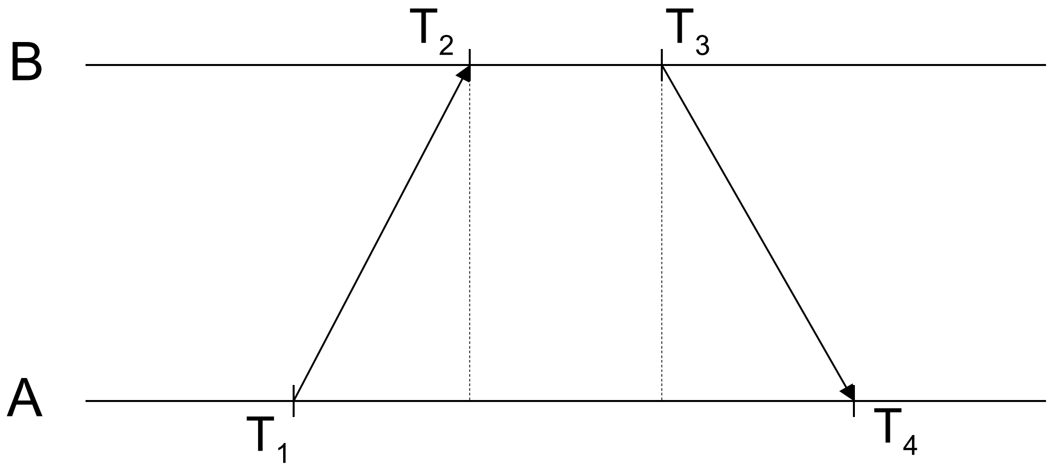
- Secondary servers are synchronized with primary servers.
- The servers are connected in a logical hierarchy called a synchronization subnet. Each level of the synchronization subnet is called stratum.

NTP (Network Time Protocol)



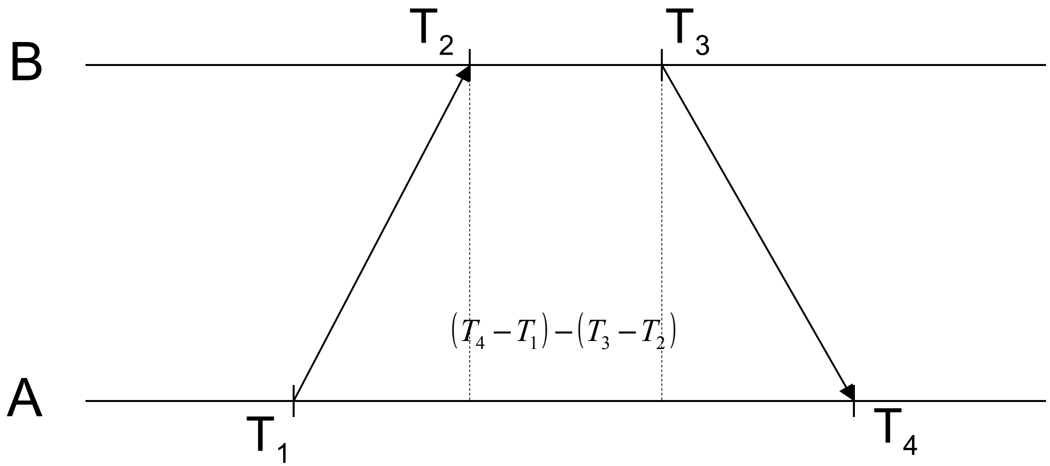
- *A* requests time of *B* at its own T_1
- *B* receives request at its T_2, records
- *B* responds at its T_3, sending values of T_2 and T_3
- *A* receives response at its T_4
- Question: what is $\theta = T_B - T_A$?

NTP (Network Time Protocol)



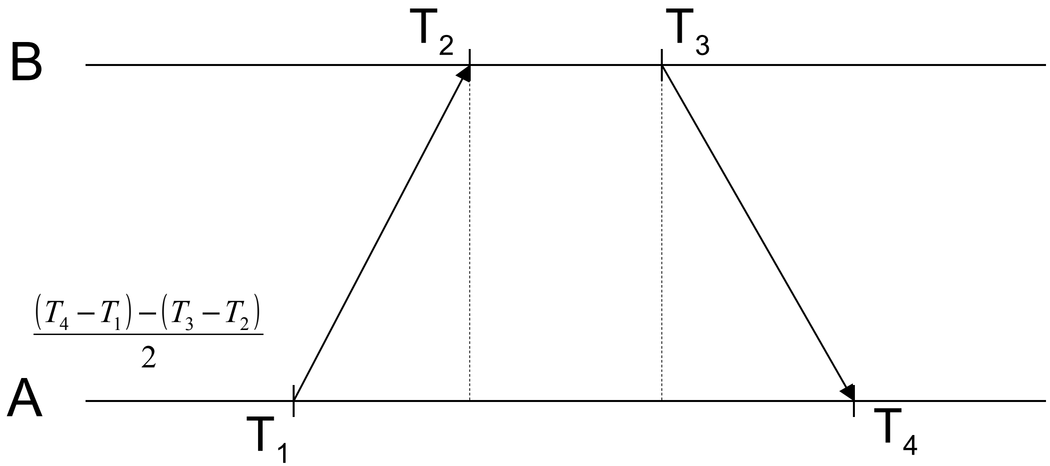
- Question: what is $\theta = T_B - T_A$?
- Assume transit time is approximately the same both ways
- Assume that *B* is the time server that *A* wants to synchronize to

NTP (Network Time Protocol)



- *A* knows (T_4 – T_1) from its own clock
- *B* reports T_3 and T_2 in response to NTP request
- *A* computes total transit time of

NTP (Network Time Protocol)



- One-way transit time is approximately ½ total, i.e.,

- *B*'s clock at T_4 reads approximately

$$T_3 + \frac{(T_4 - T_1) - (T_3 - T_2)}{2}$$

NTP (continued)

- Servers organized as *strata*
  - *Stratum 0* server adjusts itself to WWV directly
  - *Stratum 1* adjusts self to *Stratum 0* servers
  - Etc.
- Within a stratum, servers adjust with each other

Logical Clocks

Logical Clock

- Assume no central time source –
  - Each system maintains its own local clock .
  - No total ordering of events .
  - Allow to get global ordering on events.

- Assign sequence numbers to messages –
  - *All cooperating processes can agree on order of event.*

Lamport Timestamps

- It is used to provide a partial ordering of events with minimal overhead.
- It is used to synchronize the logical clock.
- It follows some simple rules:

- A process increments its counter before each event in that process i.e. Clock must tick once between every two events.
- When a process sends a message, it includes its timestamp with the message.
- On receiving a message, the receiver process sets its counter to be the maximum of the message counter and increments its own counter .

'Happened Before' Relation

- a *'Happened Before'* b : **a→b**

1. If *a* and *b* are events in the same process, and *a* comes before *b*, then $a \rightarrow b$.

2. If
   a :message sent
   b : receipt of the same message
   then $a \rightarrow b$.

1. Transitive: If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.

2. Two distinct events a and b are said to be c*oncurrent if a -/->b and b -/->a*

Lamport's Logical Clocks (2)

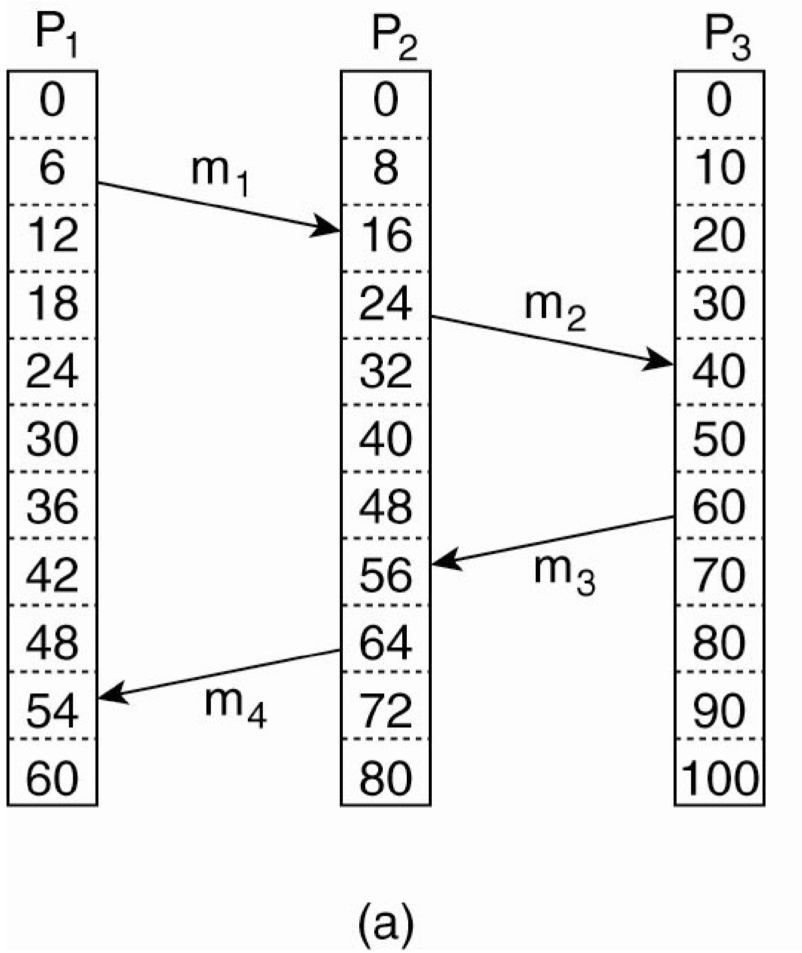
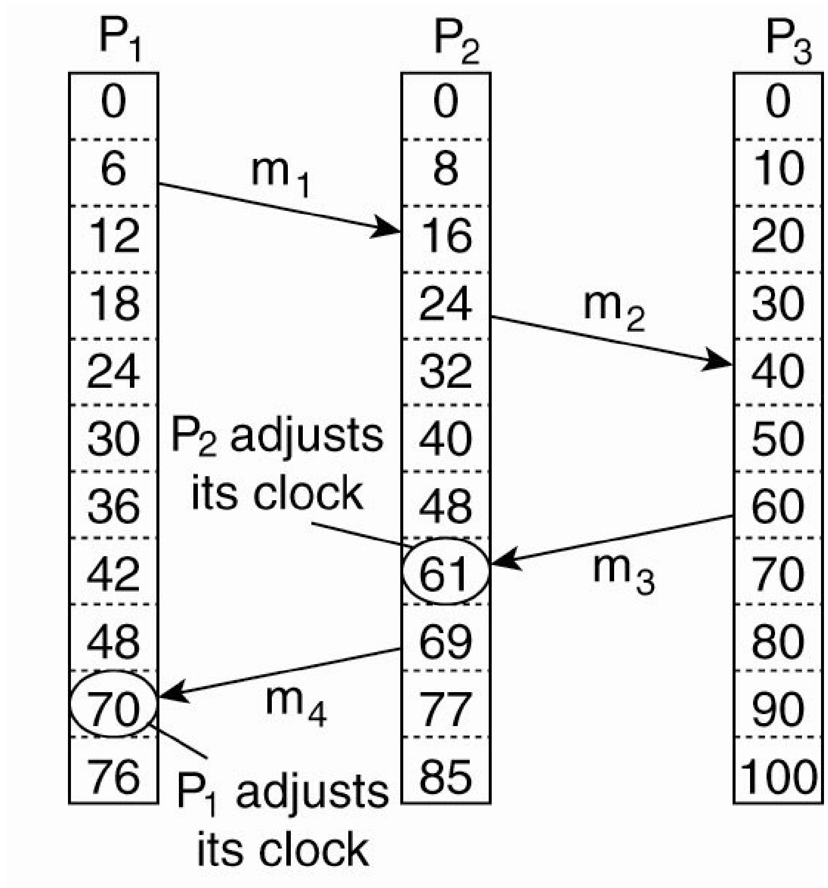


Figure 6-9. (a) Three processes, each with its own clock. The clocks run at different rates.

Rules for adjusting clocks

- For two events a and b in same process p1
  - $C(b)= C(a)+1$
- If a is sending process and b is receiving process of pi and pj then,
  - $Cj(b)=max((Ci(a)+1),Cj(b))$

Lamport's Logical Clocks (3)



(b)

Lamport's algorithm corrects the clocks.