Distributed Systems

Aleardo Manacero Jr.

UNIVERSITY OF OREGON

Time and Clocks

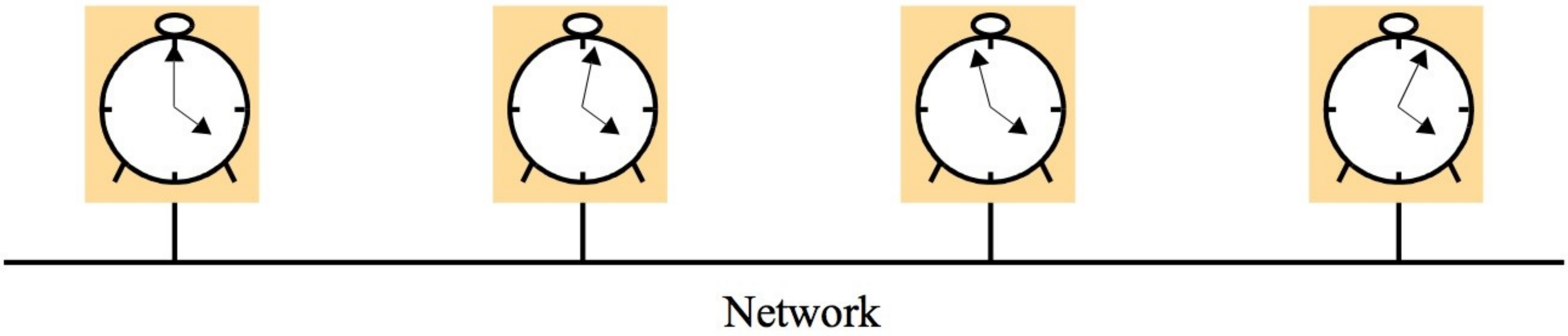
Introduction

- Distributed computers cannot keep their local clocks perfectly synchronized
- However, DS must offer some kind of event ordering in order to function properly
- Two different actions can be performed to achieve this restriction:
  - Actual clock synchronization
  - Use of logical clocks

Basics

- Differences in clocks are due to clock skew and clock drift
- Skews are instantaneous differences among clocks
- Drifts are rates of error acceleration in the clock
- Usual drift values range in 10^{-6} to 10^{-8} sec/sec,
  what is equivalent to drifting 10ms in something between 3 hours up to 11 days

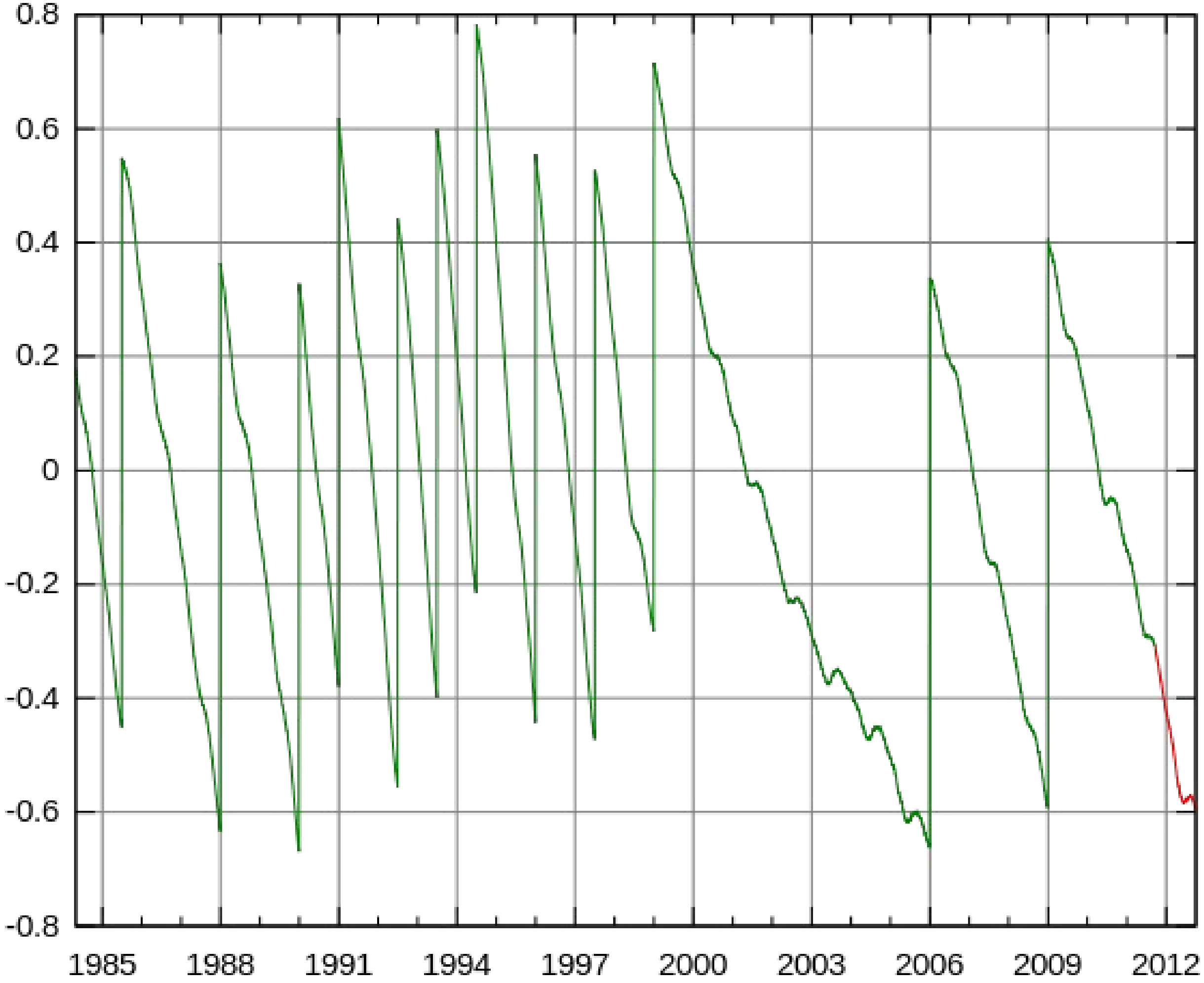
Clock skew



Basics

- Astronomical time X Atomic time
  - Atomic time is based on Cs-133 transitions
  - Astronomical time is based on Earth's revolutions
- UTC (Coordinated Universal Time) is the current standard for time keeping. It follows the atomic time with leap seconds inserted/deleted every few years to keep up with astronomical time

Drift and leap seconds in UTC



Clock synchronization

- Physical clocks may be regularly synchronized in two ways:
  - Externally, where

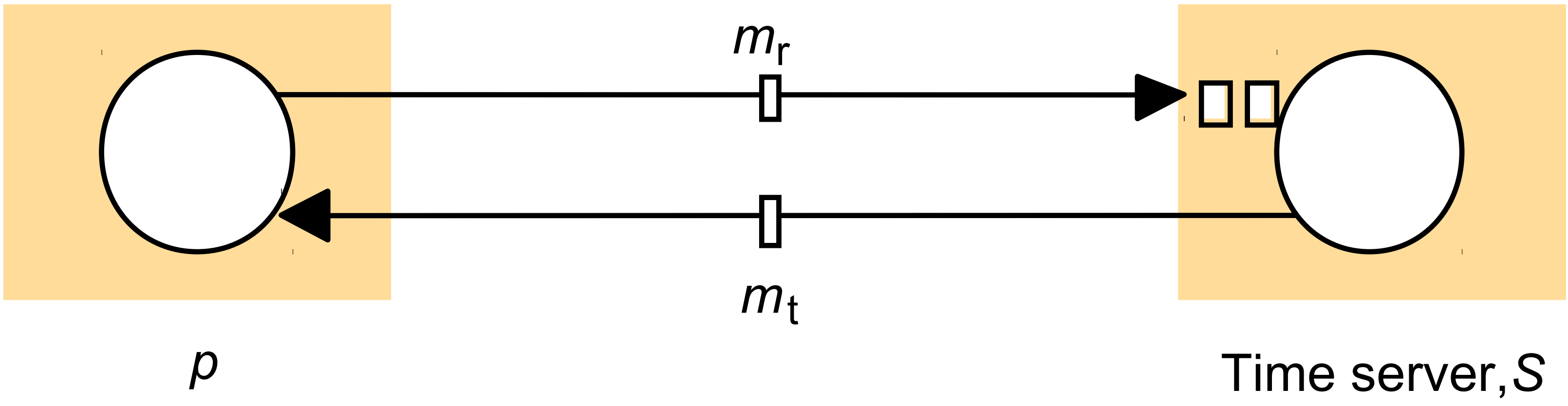
$$|S(t) - C_i(t)| < D,\ for\ i = 1 \ldots N$$

  - Internally, where

$$|C_i(t) - C_j(t)| < D,\ for\ all\ pairs\ of\ i,\ j = 1 \ldots N$$

Techniques to reduce skew – Christian's method

- Christian's method uses a time server that reads UTC time
- Uses also the round-trip time to update a local clock



Techniques to reduce skew – Christian's method

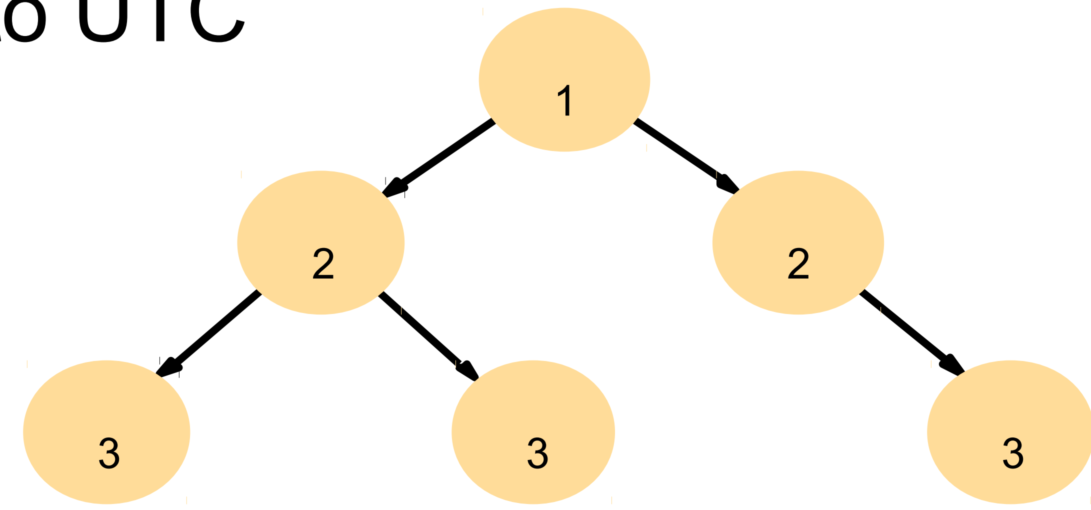
- In Christian's method each host requests the current time from the time server
- The server answers with a message containing the current time (inserted in the very last moment)
- The host measures the message's round-trip and adjusts the received time with half of this value

Techniques to reduce skew – Berkeley's algorithm

- Berkeley's algorithm is an internal technique where a master checks out the other clocks periodically
- It averages the clock values, eliminating outliers, and sending back the difference between local and average time
- Each host adjusts its clock with this difference

Techniques to reduce skew - NTP

- NTP (Network Time Protocol) is aimed to global networks
- Uses a hierarchical tree to synchronize local clocks to UTC



Note: Arrows denote synchronization control, numbers denote strata.

Techniques to reduce skew - NTP

- Server synchronize in one of three modes:
    - Multicast, used in high-speed LANs
    - Procedure-call, which is similar to Christian's method. It is more accurate than multicast
    - Symmetric, which is performed by pairs of servers in the higher part of the tree

Logical time and clocks

- Although the relevance of physical clocks, most applications need only to know the order in which events occur
- In these cases one can use logical clocks to establish an event ordering
- Lamport suggested the principal technique for logical clocks

Logical time and clocks

- Lamport clock is based in two points:
  - If two events occurred at the same process, then they occurred in the order seen by this process
  - If a message is sent between processes, the event of sending it precedes the event of receiving it
- The generalization of these points is called the happened-before relation

Happened-before relation

- This relation (denoted by $\rightarrow$) is defined as:

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

HB2: for any message m, $send(m) \rightarrow recv(m)$

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

Lamport's clock

- It is implemented considering that:
  - L_i is increased before every event ocurrence in process P_i
  - Every time that P_i sends a message, L_i's value (*t*) is added to the message *m*
  - Every time that P_i receives a message *(m,t)*, its clock is adjusted in order that the new value becomes $L_i = max(L_i, t)$, before applying the first rule

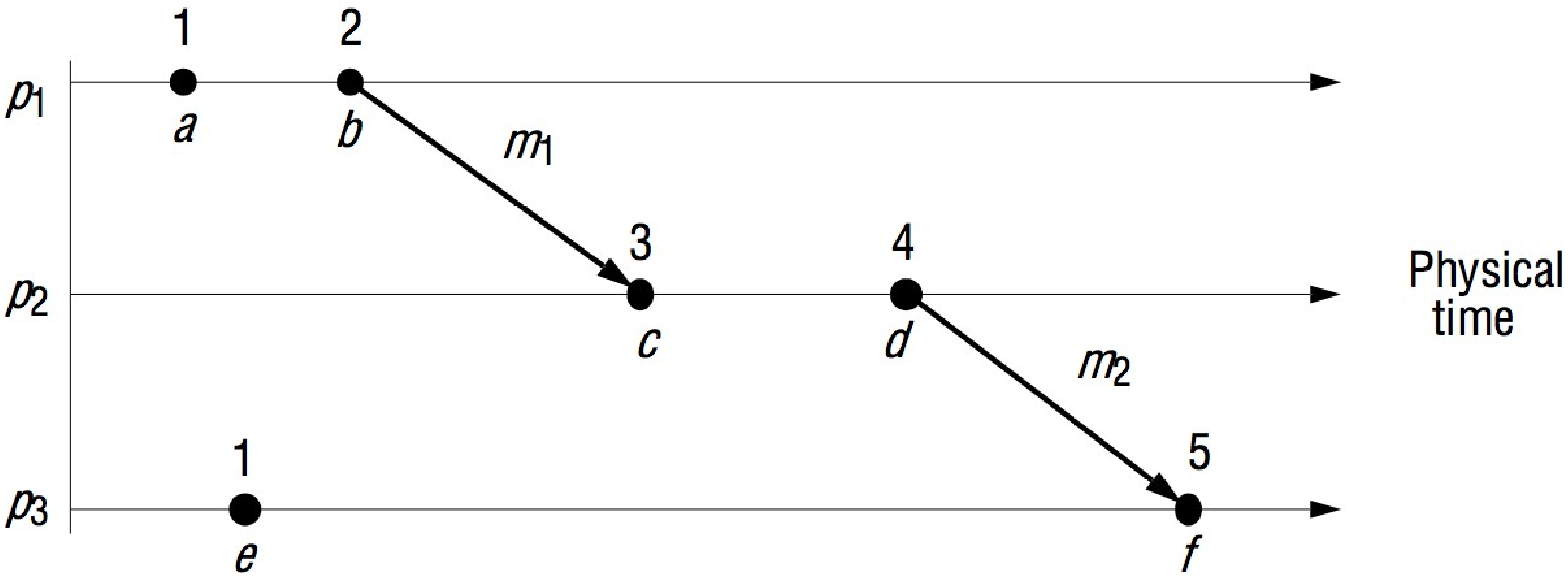
Lamport's clock

- It is verifiable that with these rules the ordering relationship between events is preserved, that is:

If e $\rightarrow$ e', then $L(e) < L(e')$

It must be observed, however, that if $L(e) < L(e')$ we cannot assure that $e \rightarrow e'$

Lamport's clock progression



Total order in logical clocks

- The problem with events in different processes having the same timestamp can be corrected using the process identifier to untie the timestamps.
- Then we have:

$(T_i, i) < (T_j, j)$ *if and only if* $T_i < T_j$ *or* $T_i = T_j$ *and* $i < j$

Vector clocks

- Fix the indetermination with L(e) and L(e') creating vector clocks in each process. The rules to update clocks are:

  VC1: Vi[j] = 0, for all i,j = 1, 2, … , N

  VC2: Vi[i] is incremented before any event in Pi

  VC3: Pi includes Vi[i] in every message it sends

  VC4: When Pi receives a message from j it sets Vi[j] = max (Vi[j], t[j])

Vector clocks

- To compare vectors the rules are:

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

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

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

Global states and properties

- The use of clocks in a DS is demanded by applications where events change its state and may imply in corrupted actions
- These applications include garbage collection and deadlock detection, for example
- They demand what is called global states determination (or global properties detection)

Global states and properties

- The evaluation of global properties enable the validation of distributed algorithms through predicate analysis
- To do so one has to define consistent cuts and determine the system's status in these cuts

A consistent cut is one where for each event that it contains, it also contains all events that happened-before it, or:

$\forall$ all events $e \in C, f \rightarrow e \Rightarrow f \in C$

Global states and properties

- The analysis over consistent cuts is performed through predicates built following techniques such as Hoare Logic, where safe/unsafe states are defined and evaluated in order to prove correctness.
- This kind of analysis can be applied in several problems, such as deadlock detection, termination detection, and garbage collection

Parenthesis : Hoare Logic

- Hoare Logic defines predicate rules, formed by triples for programming events, such as assignments, decisions, and so on.
- It allows the evaluation of program correctness, and even program design, through the establishment of desired/undesired program's status

Parenthesis : Hoare Logic

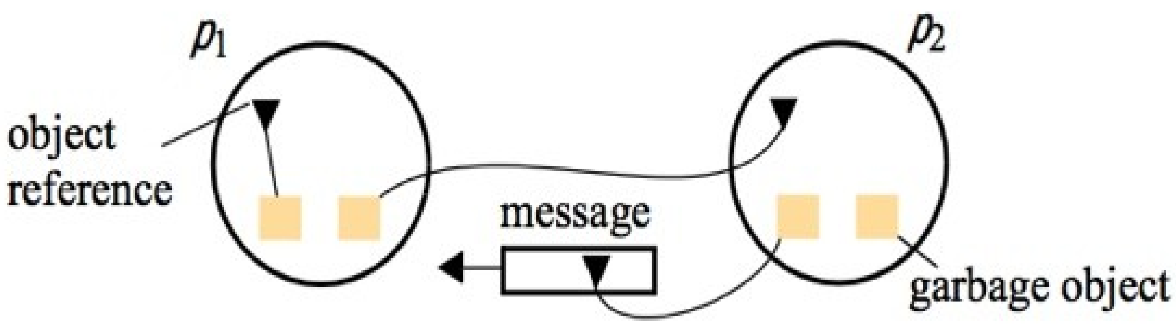
- Examples of Hoare Logic rules:
  - Assignment

    {Q[E/id]} id=E; {Q}
  - If-then-else

$$\frac{\{P \wedge E\}\ S1\ \{Q\} \quad \{P \wedge \neg E\}\ S2\ \{Q\}}{\{P\}\ \text{if } (E)\ S1 \text{ else } S2\ \{Q\}}$$

Distributed garbage collection

- An object is garbage if there are no references to it anywhere in the DS
- Problems with this include finding all references to an object

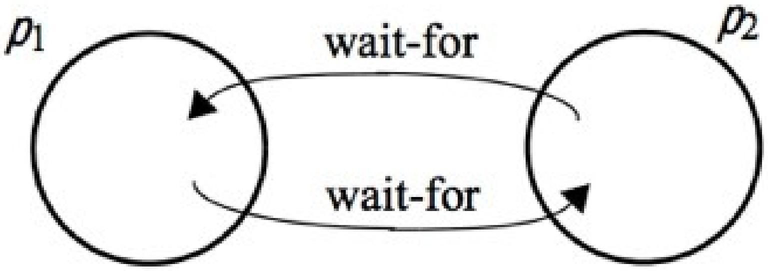


(a) Garbage collection

Deadlock detection

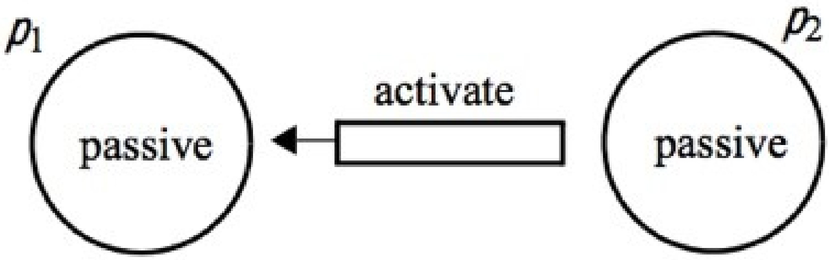
- Occurs when a waits-for cycle is created
- The problem is to determine which process is waiting for messages from who

(b) Deadlock



Termination detection

- It is similar to deadlock detection
- The problem is to determine that every process in a task actually arrived at the end and is not simply waiting for a new job



(c) Termination

THAT'S IT FOR TODAY !!