Time in Distributed Systems

Web Applications and Services

Spring Term

Naercio Magaia

UNIVERSITY OF SUSSEX

Contents

- Time in Distributed Systems
- UTC and Atomic Time
- Clock Skew and Drift
- Synchronisation
- Logical Clocks and Causal Ordering

UNIVERSITY
OF SUSSEX

Why is time important?

- It is an important issue in Distributed Systems. For example, it
  - timestamps events and actions
  - supports distributed algorithms
    - consistency of distributed data (i.e., timestamps in transactions)
    - checking authenticity in security algorithms, hence defending against replay attacks
    - eliminate processing of duplicate updates
  - debugging and troubleshooting
    - much easier when the timestamps in the log files of all devices are synchronised
  - Digital certificates
    - To ensure that valid certificates are used based on date and time

Measuring time

- The notion of physical time is problematic in distributed systems

  “A man with one watch knows what time it is. However,
  a man with two watches is never quite sure”

- Timestamping events at different nodes, i.e., ordering of events
  - clocks are not accurate
  - no absolute, global time
  - network delays, message processing delays

UNIVERSITY OF SUSSEX

More on Distributed Systems

- It consists of a collection P of N processes $p_i, i = 1,2, \dots N$
- Each process executes on a single processor and has a state s_i that it transforms as it executes
- Processes only communicate by sending messages through the network
- As each process p_i executes it takes a series of actions
  - a message send or receive operation
  - an operation that transforms p_i's state s_i

UNIVERSITY OF SUSSEX

More on Distributed Systems

- An event is the occurrence of a single action that a process carries out as it executes
  - A communication action or a state-transforming action
- total ordering (relation $\rightarrow_i$) is a sequence of events within a single process
  - $e \rightarrow_i e'$ iff event e happens before event e'
- $\text{history}(p_i) = h_i = < e_i^0, e_i^1, e_i^2, \ldots >$

UNIVERSITY OF SUSSEX

Clocks

A miniature 16 MHz quartz crystal

- Each computer contains its own physical clock
  - They are electronic device that counts oscillations occurring in a crystal at a specific frequency
- The operating system
  - reads the node's the hardware clock value $H_i(t)$
  - Scales it and adds an offset to produce a software clock $C_i(t) = \alpha H_i(t) + \beta$
- Clocks are not accurate - $C_i(t)$ will differ from t
- If C_i behaves sufficiently well, it values can be used to timestamp events at p_i

Any ideas of properties of a well-behaving clock?

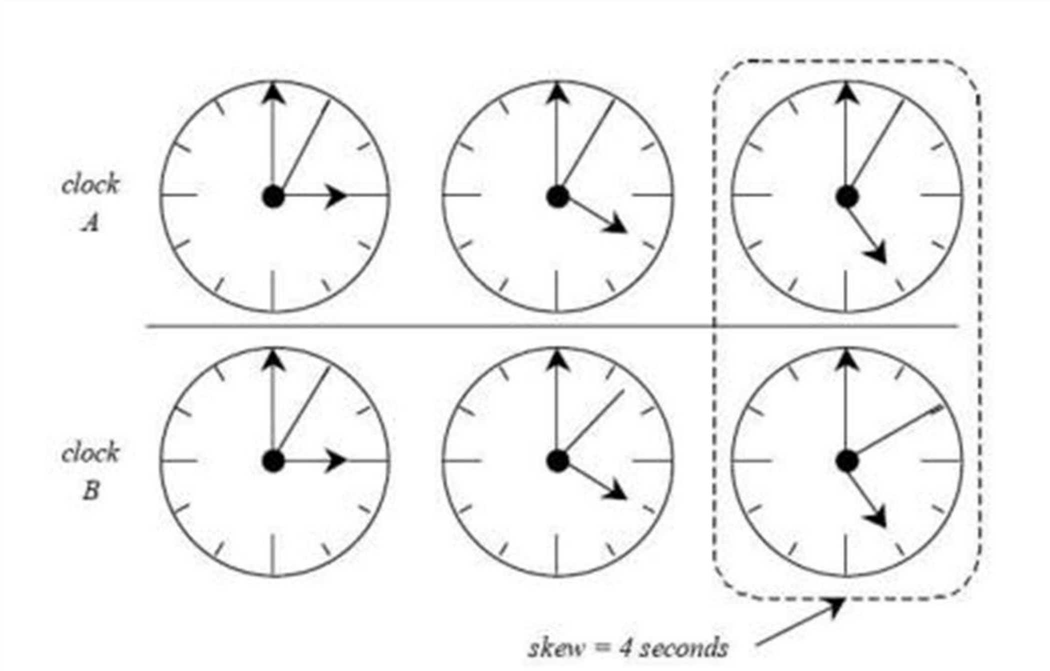
UNIVERSITY OF SUSSEX

Clock Skew and Drift

- Computer clocks tend not to be in perfect agreement
- **Clock drift** is the difference in reading between the clock and a perfect reference clock per unit of time
  - ordinary quartz clocks
    - 10^{-6} seconds/second, i.e., 1 second in 11.6 days
  - ‘high-precision’ quartz clocks
    - 10^{-7} or 10^{-8} seconds/second

Clock Skew and Drift

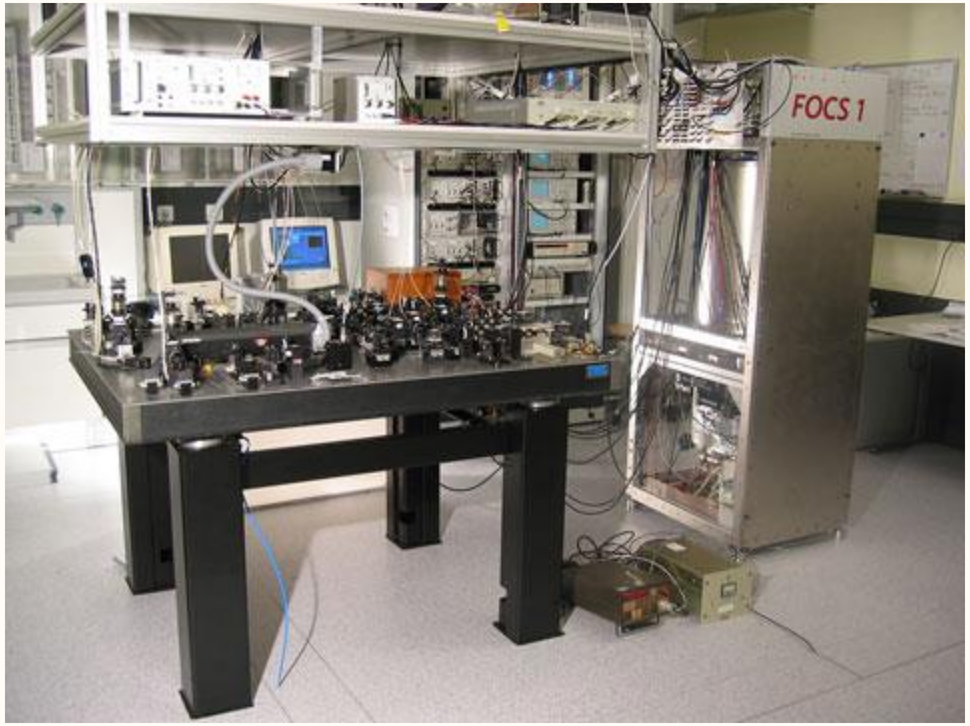
- **Clock skew** is the difference between the readings of any two clocks



UNIVERSITY OF SUSSEX

UTC and Atomic Time

- Clocks can be synchronised to external sources of highly accurate time
  - Atomic oscillators drift rate is about one part in 10^{13}
- The output of these atomic clocks is used as the standard for elapsed real time, i.e., International Atomic Time (TAI)
  - Since 1967, the standard second has been defined as 9,192,631,770 periods of transition between the two hyperfine levels of the ground state of Caesium-133 (Cs^{133})



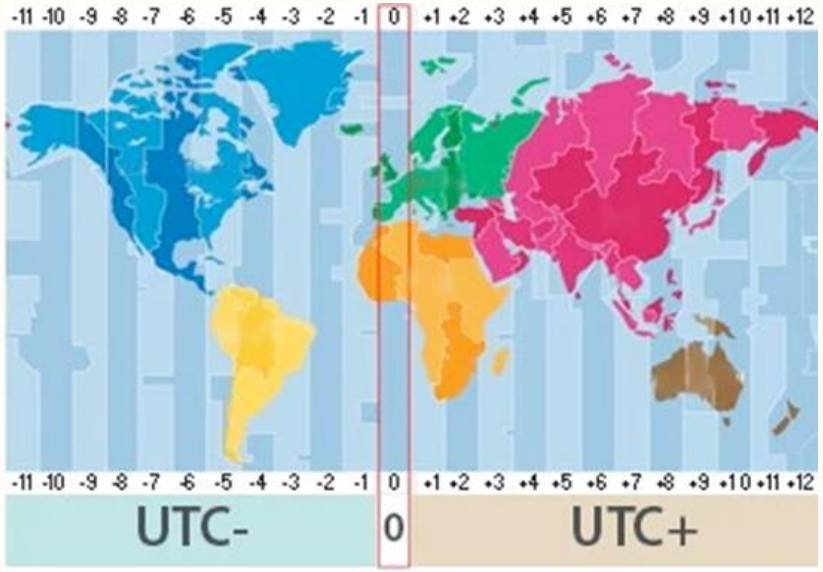
TAI is an international time scale that is computed by taking the weighted average of more than 300 atomic clocks

UNIVERSITY OF SUSSEX

UTC and Atomic Time

- The Coordinated Universal Time (UTC) is an international standard for timekeeping
  - UTC signals are synchronised and broadcast regularly from land-based radio stations and satellites (leap second – December 31, 2016)
- Land-based stations: 0.1-10 millisecond accuracy
- GPS satellite-based: 1 microsecond accuracy

TAI is one of the main components of Coordinated Universal Time

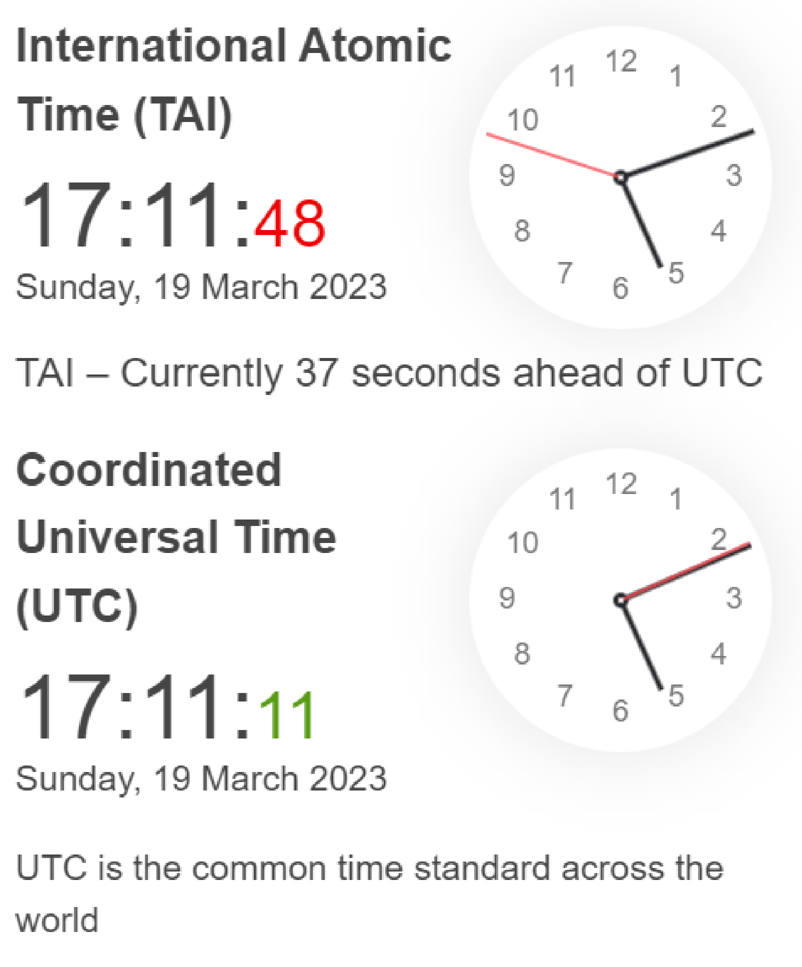


The Leap Second

- A leap second is a one-second adjustment
- Is occasionally applied to UTC to accommodate the difference between precise time (as measured by atomic clocks) and imprecise observed solar time
  - varies due to irregularities and long-term slowdown in the Earth's rotation

The Leap Second

- From time to time, leap seconds are
  - introduced into UTC to get in sync with TAI
  - add to ensure our clocks reflect the Earth's rotation speed as accurately as possible.



UNIVERSITY OF SUSSEX

Synchronising Physical Clocks

- External synchronisation
  - C_i is synchronised to a UTC time source S
  - $|S(t) - Ci(t)| < D$, for $i = 1,2, \ldots, N$ and for all real time t, clock C_i is accurate to within the bound $D > 0$
- Internal synchronisation
  - C_i is synchronised with one another to a known degree of accuracy
  - $|Ci(t) - C_j(t)| < D$ for $i, j = 1,2, \ldots, N$, and for all real time t, clocks C_i agree with each other within the bound $D > 0$

Internally synchronised clocks are not necessarily externally synchronised

UNIVERSITY OF SUSSEX

Synchronisation

- One process sends the time t on its local clock to the other in a message m
- Receiving process sets its clock to the time $t + T_{trans}$, where T_{trans} is the time taken to transmit m. T_{trans} is subject to **variation** and is **unknown!**
- T_{min} is a minimum transmission time - min can be estimated
- In a synchronous system, there is also a T_{max}
- Uncertainty in the message transmission time $u, u = T_{max} - T_{min}$
- Set clock to be $t + T_{min}$ or $t + T_{max}$: the skew is as much as u
- Set the clock to be $t + (T_{min} + T_{max})/2$: the skew is at most $u/2$

UNIVERSITY OF SUSSEX

Cristian's Method for External Synchronisation

- There is no upper bound on transmission delays in an asynchronous system
- A probabilistic algorithm
  - Synchronisation is achieved only if the observed round-trip times (RTTs) between client and server are sufficiently short compared to the required accuracy
- A process p requests the time in message m_r, receives time in m_t, and total round trip time is T_{round}.
  - T_{round} can be calculated with reasonable accuracy if the local clock drift is small

UNIVERSITY OF SUSSEX

Cristian's Method for External Synchronisation

- A simple estimate of the time is $t + T_{round}/2$
  - path is symmetric
  - several request/response messages to increase accuracy

The Berkeley algorithm

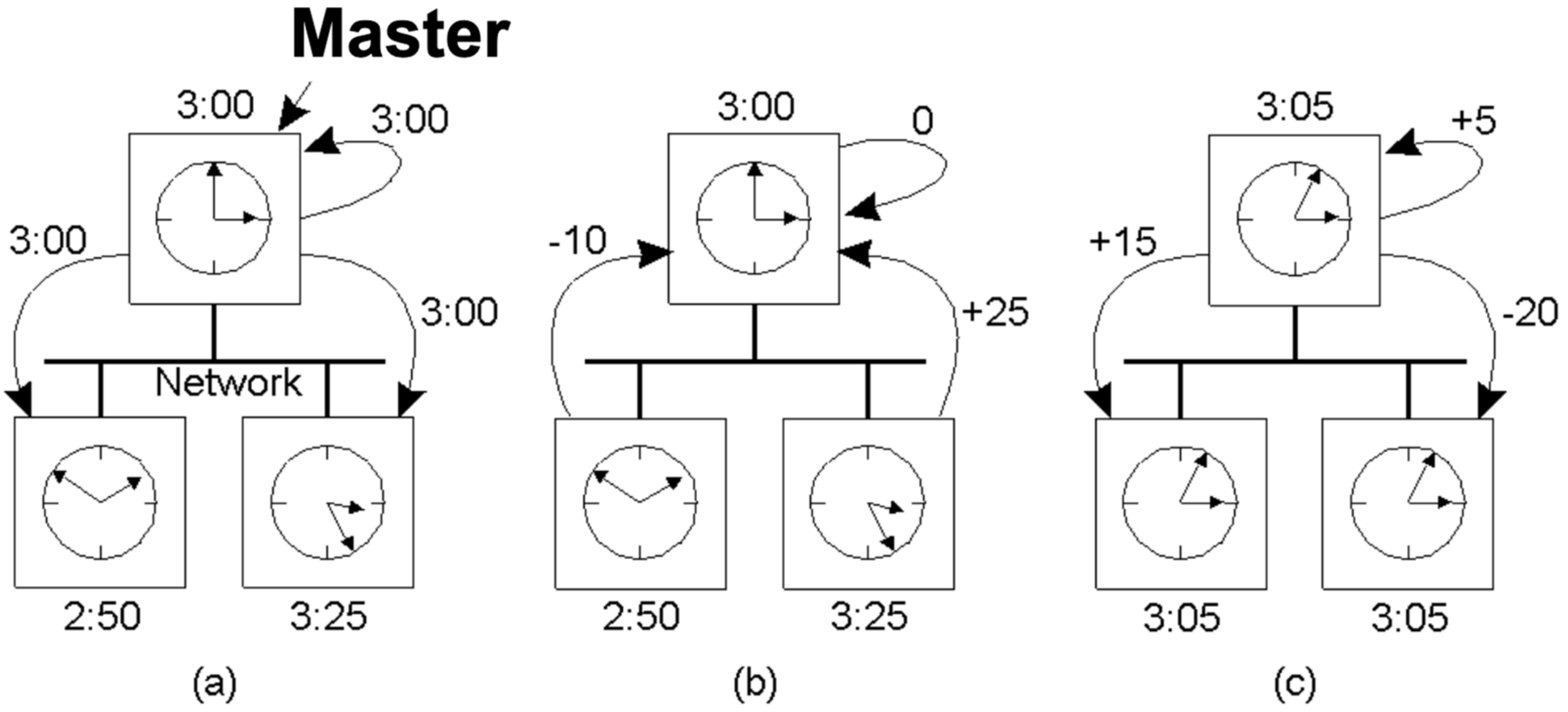
- Is an algorithm for internal synchronisation
- One process is elected as a leader (i.e., master) and the rest are slaves
- The master
  - periodically polls slaves that send back their clock values
  - estimates their local clock times by observing the round-trip times, and averages the values obtained

The Berkeley algorithm

- The master (cont.)
  - eliminates any readings associated with large RTTs
  - sends the amount by which each slave adjusts the clock – positive or negative
- No single point of failure as faulty clocks are ignored
- The whole system can drift away from UTC
- Re-election required when master fails

Is master telling the truth?

The Berkeley algorithm



The Network Time Protocol (NTP)

- Enables clients across the Internet to be synchronised accurately to UTC
- Provides a reliable service that can survive lengthy losses of connectivity
- Enables clients to re-synchronise sufficiently frequently to offset the rates of drift found in most computers
  - It scales up to a large number of clients and servers
- Provides protection against interference with the time service, whether malicious or accidental

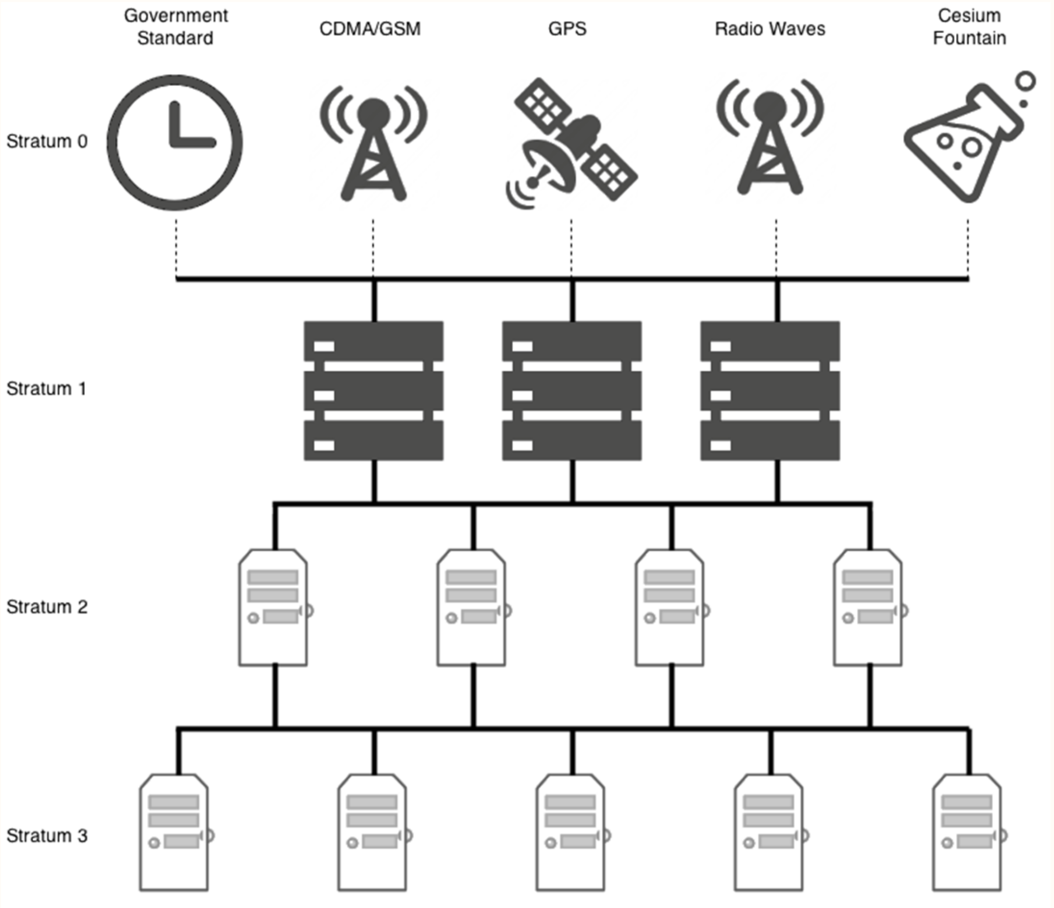
The Network Time Protocol (NTP)

- Servers are connected in a logical hierarchy (i.e., strata)
  - Primary servers are connected directly to a time source (e.g., radio UTC or GPS) – Stratum 1
  - Secondary servers are synchronised with primary servers
  - Less accuracy in higher strata
  - Reconfiguration as servers become unreachable or failures occur

UNIVERSITY OF SUSSEX

The Network Time Protocol (NTP)

- Synchronisation over UDP
  - Multicast mode used for high-speed LANs
    - Time is periodically multicast to servers running in other machines
    - Delay is very small, and clocks are set under this assumption
- Procedure-call mode is similar to the operation of Cristian's algorithm
  - A server accepts requests from other computers and replies with its time
- Symmetric mode – Higher accuracy between pair of servers
  - exchanged messages carry timing information
  - timing data are stored to improve accuracy over time



UNIVERSITY OF SUSSEX

List of Top Public Time Servers

- Google Public NTP [AS15169]
  - time.google.com
  - time1.google.com
  - time2.google.com
  - time3.google.com
  - time4.google.com
- Cloudflare NTP [AS13335]
  - time.cloudflare.com
- Microsoft NTP server [AS8075]
  - time.windows.com
- Apple NTP server [AS714, AS6185]
  - time.apple.com
- Facebook NTP [AS32934]:
  - time.facebook.com
  - time1.facebook.com
  - time2.facebook.com
  - time3.facebook.com
  - time4.facebook.com
- NIST Internet Time Service (ITS) [AS49, AS104]
  - time-a-g.nist.gov
  - time-b-g.nist.gov
  - time-c-g.nist.gov
  - time-d-g.nist.gov

UNIVERSITY OF SUSSEX

Logical Time

- No perfect synchronisation in a distributed system using physical clock
- Do we always need the absolute time to timestamp?
- Can we use relative time if causality is maintained?
- Consider the following points
  - If two events occurred at the same process p_i $(i = 1, 2, \dots N)$, then they occurred in the order in which p_i observes them. This is the order $\rightarrow_i$
  - Whenever a message is sent between processes, the event of sending the message occurred before the event of receiving the message

UNIVERSITY OF SUSSEX

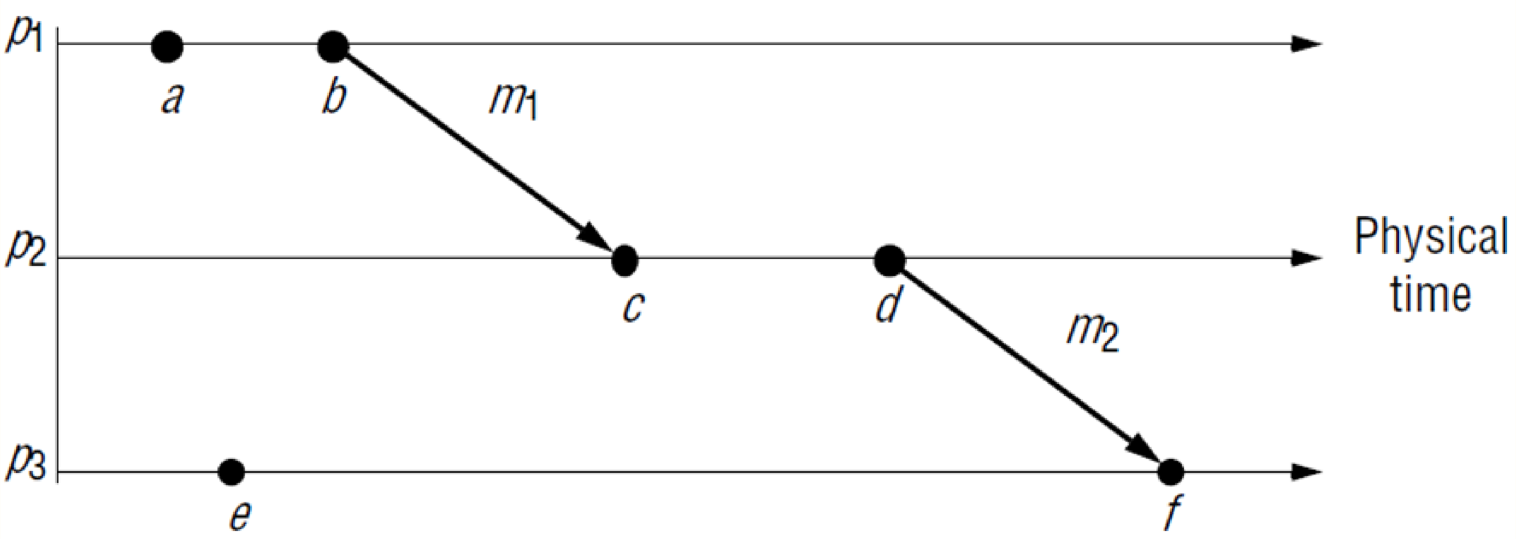
Logical Time

- *Lamport* called $\rightarrow_i$ the *happened-before* relation
  - It is also known as the relation of *causal ordering* or *potential casual ordering*
- The happened-before relation, denoted $\rightarrow$, is defined as
  - If $\exists$ process $p_i : e \rightarrow_i e'$, then $e \rightarrow e'$
  - For any message m, $send(m) \rightarrow receive(m)$
  - If e, e' and e'' are events such that $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$

UNIVERSITY OF SUSSEX

Logical Time

- Consider the case of 3 processes p_1, p_2, and p_3



- Events such as a and e are not ordered by $\rightarrow$
- Concurrent: $a \parallel e$

UNIVERSITY OF SUSSEX

Lamport Logical Clocks

- Is a simple mechanism by which the happened-before ordering is <u>captured numerically</u>
- Is a monotonically increasing software counter
  - There is no relationship to physical clock
- Each process p_i keeps its own *logical clock* L_i to apply *Lamport timestamps* to events
  - $L_i(e)$: the timestamp of event e at p_i
  - $L(e)$: the timestamp of event e at whatever process it occurred

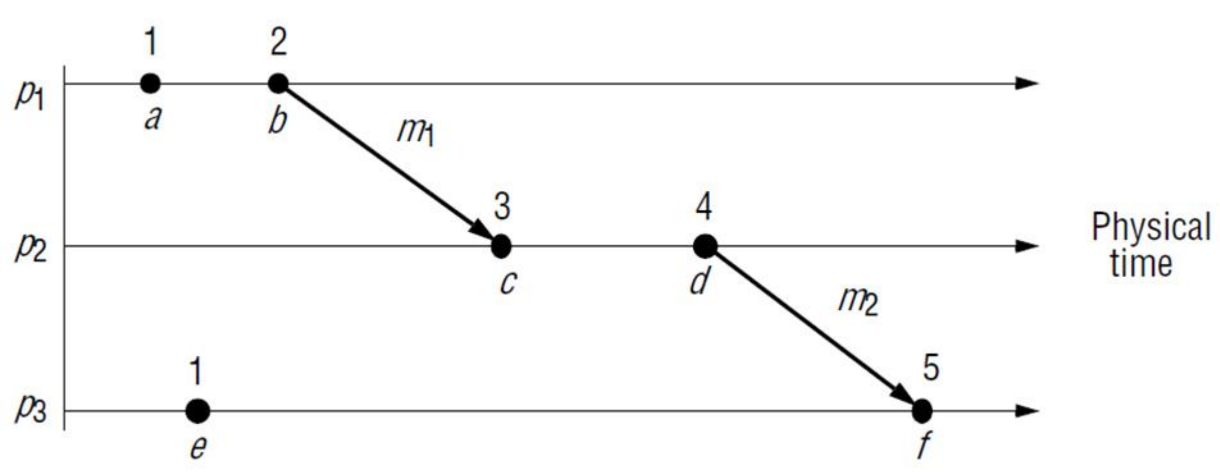
UNIVERSITY OF SUSSEX

Lamport Logical Clocks

- To capture the *happened-before* relation $\rightarrow$, processes update their logical clocks and transmit their values in messages
  - L_i is incremented before each event is issued at process p_i: $L_i := L_i + 1$
  - When a process p_i sends a message m, it piggybacks on m the value $t = L_i$
  - On receiving (m, t), a process p_j computes $L_j := \max(L_j, t)$ and then applies the first rule before timestamping the event $receive(m)$

Lamport Logical Clocks

- Clocks (initially 0) are incremented by 1



- If $e \rightarrow e'$ then $L(e) < L(e')$
- The converse is NOT true. If $L(e) < L(e')$, then we cannot infer that $e \rightarrow e'$
  - For example, $L(e) < L(b)$ but $b \,||\, e$

Vector Clocks

- Overcome the main shortcoming of Lamport clocks
- A vector clock for a system of N processes is an array of N integers
- Each process p_i keeps its **own** vector clock V_i to timestamp local events
- Processes piggyback vector timestamps on the messages they send

Vector Clocks

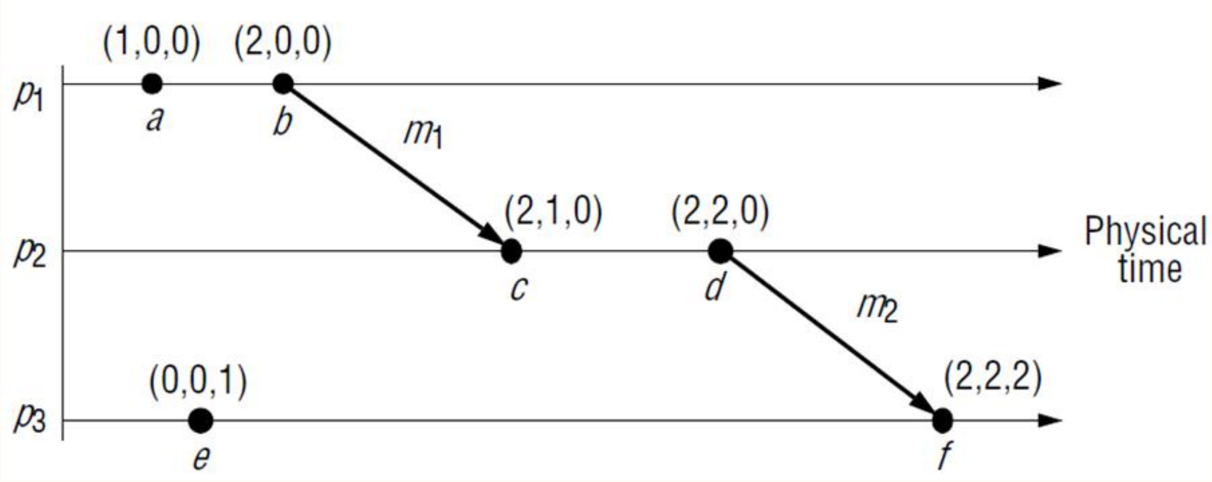
- Clocks are updated as follows
  - Initially, $V_i[j] = 0$, for $i, j = 1, 2, \ldots N$
  - Just before p_i timestamps an event, it sets $V_i[i] := Vi[i] + 1$
  - p_i includes the value $t = Vi$ in every message it sends
  - When p_i receives a timestamp t in a message, it sets

$V_i[j] := \max(Vi[j], t[j])$, for $j = 1, 2, \ldots N$ (vector merging)

UNIVERSITY OF SUSSEX

Vector Clocks

- For a vector clock V_i
  - $V_i[i]$ is the number of events that p_i has timestamped
  - $V_i[j]\ (j \neq i)$ is the number of events that have occurred at p_j that have potentially affected p_i
- Vector timestamps for the events of processes p_1, p_2, and p_3



Vector Clocks

- Vector timestamps can be compared
  - $V = V'$ iff $V[j] = V'[j]$ for $j = 1, 2, \ldots N$
  - $V \leq V'$ iff $V[j] \leq V'[j]$ for $j = 1, 2, \ldots N$
  - $V < V'$ iff $V \leq V$ and $V \neq V'$
- if $e \rightarrow e'$ then $V(e) < V(e')$ and if $V(e) < V(e')$ then $e \rightarrow e'$
- Neither $V(c) < V(e)$ nor $V(e) < V(c)$, therefore $c \,||\, e$
- Disadvantage: the amount of storage and message payload is proportional to N

UNIVERSITY OF SUSSEX

Next Lecture …

- ✓ Introduction
- ✓ HTTP, Caching, and CDNs
- ✓ Views
- ✓ Templates
- ✓ Forms
- ✓ Models
- ✓ Security
- ✓ Transactions
- ✓ Remote Procedure Call
- ✓ Web Services
- ✓ Time
- ➢ **Elections and Group Communication**
- Mutual Exclusion and Agreement
- Zookeeper

UNIVERSITY OF SUSSEX