DISTRIBUTED SYSTEMS

09

Time Synchronization

Konrad Iwanicki

Copyright notice

Acknowledgments

This lecture is (partly) based on:

1. M. van Steen and A. Tanenbaum: *Distributed Systems*, CreateSpace Independent Publishing Platform, 3.01 edition (February 1, 2017), 596 pages, ISBN 978-1543057386, Chapter 6.
2. B. Liskov: “Practical Uses of Synchronized Clocks in Distributed Systems,” *Distributed Computing*, vol. 6(4), Springer-Verlag, July 1993, pages 211-219.
3. J.C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, JJ Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford: “Spanner: Google’s Globally-Distributed Database,” *ACM Transactions on Computer Systems*, vol. 31(3), ACM, August 2013, pages 8:1-8:22.

Acknowledgments

This lecture is (partly) based on (cont.):

4. D.K. Gifford: "Information Storage in Decentralized Computer Systems," *Technical Report CSL-81-8*, Xerox Corporation, Palo Alto, CA, USA, March 1982, 153 pages.

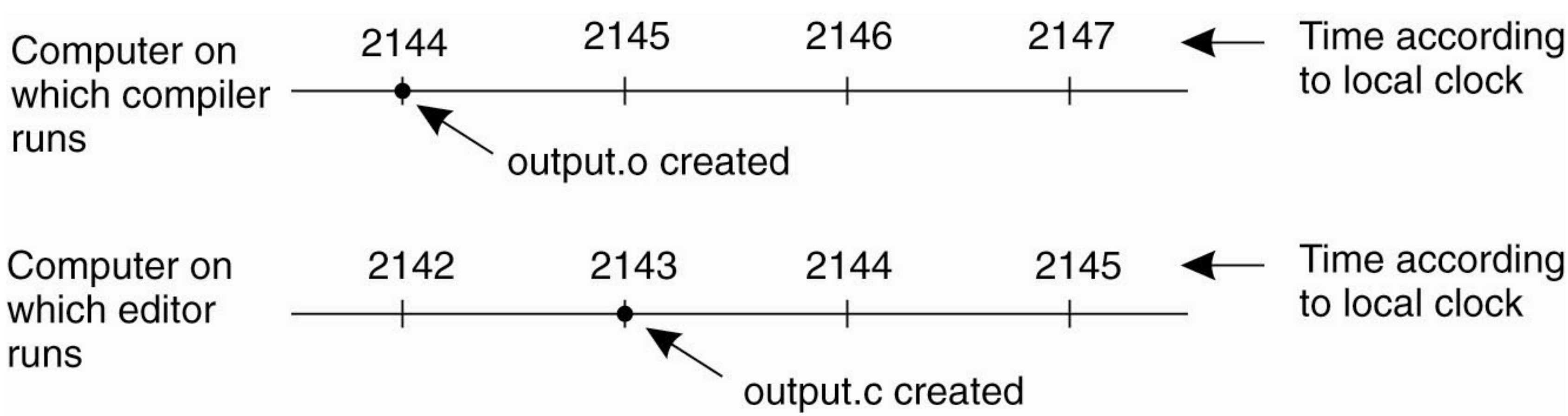
Introduction

- In a centralized (single-node) system time is unambiguous:
  - Process P_A asks for time and gets T_A.
  - Later, process P_B asks for time and gets T_B.
  - For sure, $T_A \leq T_B$.
  - In other words, P_A and P_B always agree on the current time.
- This fact is made use of in various cases:
  - e.g., the *make* tool

Introduction

- Achieving agreement on time in a distributed system is not trivial.
- In some cases, a lack of such an agreement can have grave consequences.

Introduction



Introduction

- There are many cases in which agreeing on time is important:
  - Financial brokerage
  - Security auditing
  - Collaborative sensing
- In general, people analyze events wrt time.

Introduction

- There are many cases in which agreeing on time is important:
  - Financial brokerage
  - Security auditing
  - Collaborative sensing
- In general, people analyze events wrt time.

- Is it possible to synchronize all the clocks in a distributed system?

Physical clocks

- Each computer has a so-called timer:
  - A quartz oscillator with two registers.
- A counter register is decremented on each oscillation.
- When it goes to zero,
  - it is reloaded with the value from a holding register.
  - a clock interrupt is generated => the clock ticks.

Physical clocks

- Each computer has a so-called timer:
  - A quartz oscillator with two registers.
- A counter register is decremented on each oscillation.
- When it goes to zero,
  - it is reloaded with the value from a holding register.
  - a clock interrupt is generated => the clock ticks.
- Effect: we can make the clock tick every second to maintain time for our computer.

Physical clocks

- However, with multiple clocks the situation changes.
- Timers are imperfect oscillators:
  - N computers => N different oscillation frequencies

Physical clocks

- However, with multiple clocks the situation changes.
- Timers are imperfect oscillators:
  - N computers => N different oscillation frequencies

- How do we keep them in sync with each other?
- How do we keep them in sync with the external world (the real time)?

Measuring time

- In the past, time was measured astronomically:
  - Solar day = the period between two consecutive appearances of the sun at the peek point in the sky
  - Solar second = 1 / (24 * 60 * 60) of a solar day

Measuring time

- Solar day is not constant!
  - Permanent changes in the Earth's rotation speed:
    - Days are getting longer.
  - Temporal variations.

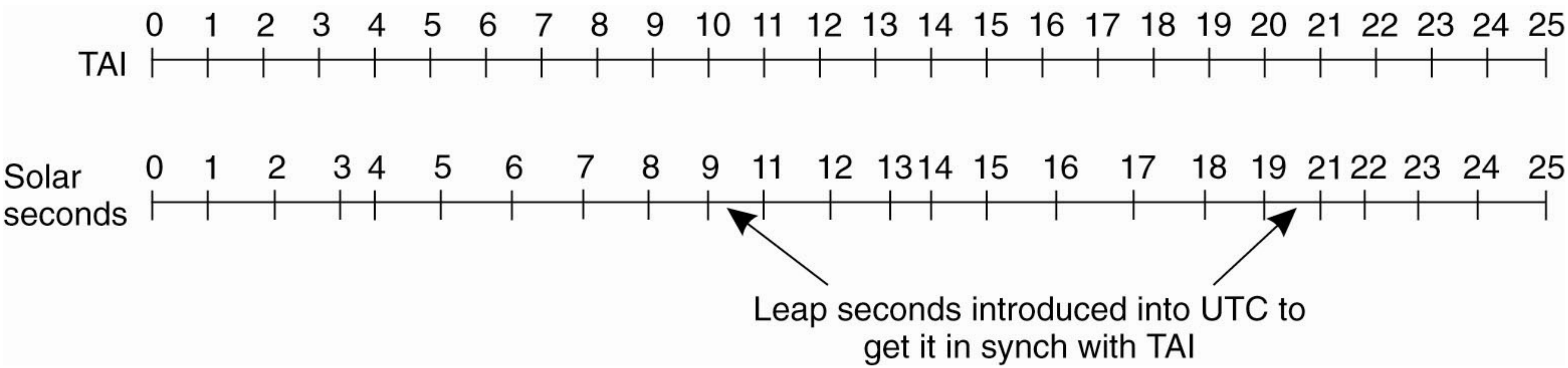
Measuring time

- Atomic clocks can provide accurate time
  - Idea: counting the number of transitions of the cesium 133 atom (earlier also rubidium 87 and thallium 205).
  - 1 second = 9,192,631,770 transitions
- Several laboratories have atomic clocks
- Periodically, they inform the International Time Bureau about the number of ticks
- The average is known as International Atomic Time (TAI)

Measuring time

  - TAI is highly stable.
  - Solar day is getting longer.
- => 86,400 TAI seconds is now about 3 ms less than a mean solar day
- Tolerating this discrepancy = bad idea.
- Solution: leap seconds.

Measuring time



- This correction is a base of Universal Coordinated Time (UTC).

Obtaining UTC

- Most electric companies synchronize the timing of their 60-Hz or 50-Hz clocks to UTC.
- Shortwave pulses at the start of every second:
  - NIST, Fort Collins, CO, USA
  - MSF, Rugby, England

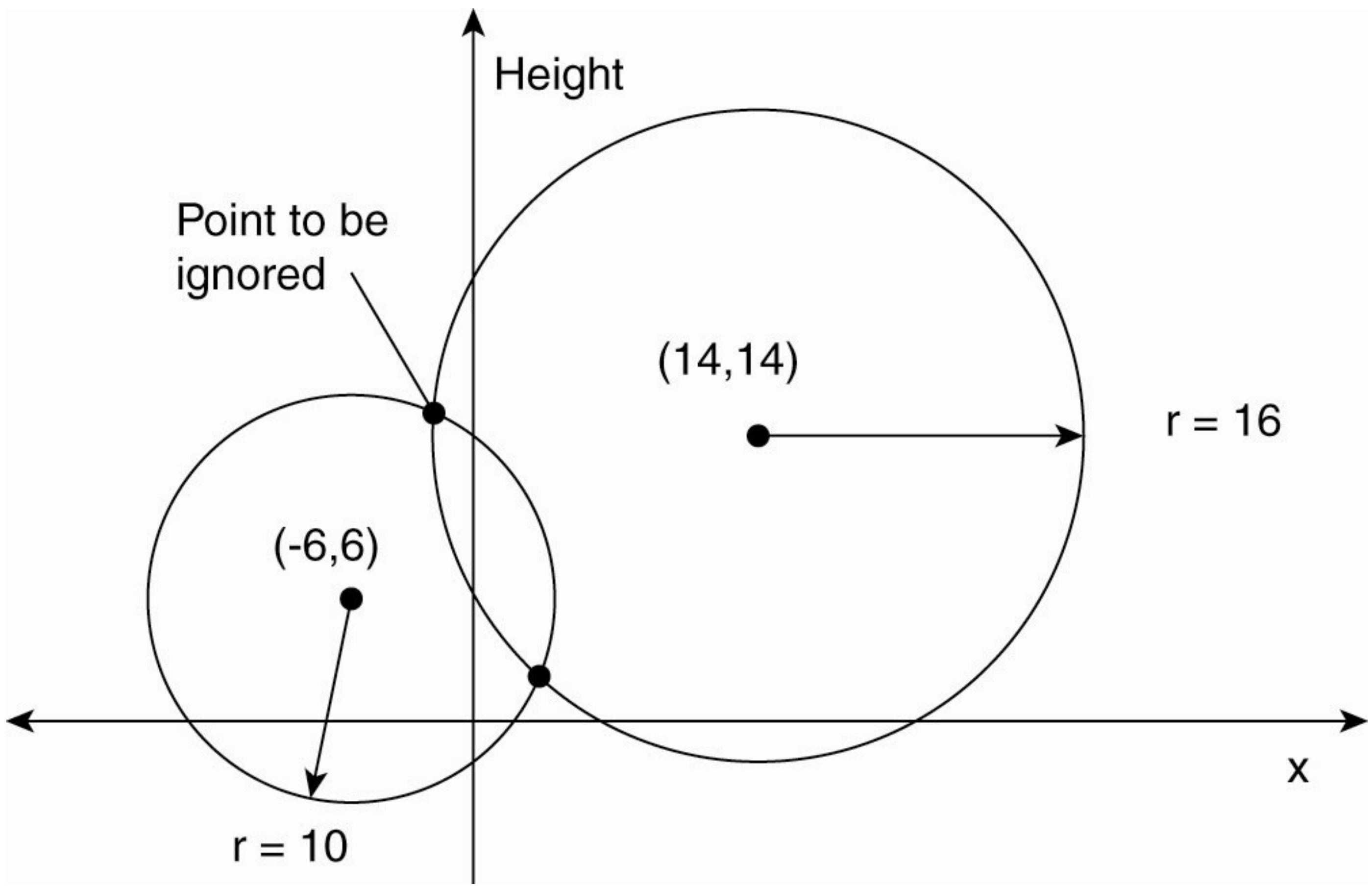
  Accuracy: ± 1 ms (broadcaster), ± 10 ms (recv)
- Earth satellites also offer UTC:
  - GEOS

  Accuracy: ± 0.5 ms

Global Positioning System

- Global Positioning System (GPS) offers time synchronization as a by-product:
  - 29 satelites
  - At ~20,000 km
- Each satellite has up to 4 atomic clocks.
- The clocks are calibrated from stations on Earth.
- Each satellite continuously broadcasts its position and local time.

Global Positioning System



Global Positioning System

- Problem: Assuming that the clock's of satellites are accurate and synchronized:

Global Positioning System

- Problem: Assuming that the clock's of satellites are accurate and synchronized:
  - It takes a while before a satellite's position reaches a GPS receiver.

Global Positioning System

- Problem: Assuming that the clock's of satellites are accurate and synchronized:
  - It takes a while before a satellite's position reaches a GPS receiver.
  - The receiver's clock need not be in sync with the satellite's clock.

Global Positioning System

- Principal operation:
  - Δ_r : unknown deviation of the receiver's clock
  - x_r, y_r, z_r : unknown coordinates of the receiver's clock
  - T_i : timestamp on a message from satellite i
  - $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i
  - Δ_i x c : measured distance to satellite i
  - Real distance is:

$$d_i = c\Delta_i - c\Delta_r = \sqrt{\left((x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2\right)}$$

  - 4 satellites = 4 equations with 4 unknowns

Global Positioning System

- Principal operation:
  - Δ_r : unknown deviation of the receiver's clock

Global Positioning System

- Principal operation:
  - Δ_r : unknown deviation of the receiver's clock
  - x_r, y_r, z_r : unknown coordinates of the receiver's clock

Global Positioning System

- Principal operation:
  - Δ_r : unknown deviation of the receiver's clock
  - x_r, y_r, z_r : unknown coordinates of the receiver's clock
  - T_i : timestamp on a message from satellite i

Global Positioning System

- Principal operation:
  - Δ_r : unknown deviation of the receiver's clock
  - x_r, y_r, z_r : unknown coordinates of the receiver's clock
  - T_i : timestamp on a message from satellite i
  - $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i

Global Positioning System

- Principal operation:
  - Δ_r : unknown deviation of the receiver's clock
  - x_r, y_r, z_r : unknown coordinates of the receiver's clock
  - T_i : timestamp on a message from satellite i
  - $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i
  - $\Delta_i \times c$: measured distance to satellite i

Global Positioning System

- Principal operation:
  - Δ_r : unknown deviation of the receiver's clock
  - x_r, y_r, z_r : unknown coordinates of the receiver's clock
  - T_i : timestamp on a message from satellite i
  - $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i
  - Δ_i x c : measured distance to satellite i
  - Real distance is:

$$d_i = c\,\Delta_i - c\,\Delta_r = \sqrt{\left(\left(x_i - x_r\right)^2 + \left(y_i - y_r\right)^2 + \left(z_i - z_r\right)^2\right)}$$

Global Positioning System

- Principal operation:
  - Δ_r : unknown deviation of the receiver's clock
  - x_r, y_r, z_r : unknown coordinates of the receiver's clock
  - T_i : timestamp on a message from satellite i
  - $\Delta_i = (T_{now} - T_i) + \Delta_r$: measured delay of the message sent by satellite i
  - Δ_i x c : measured distance to satellite i
  - Real distance is:

$$d_i = c\,\Delta_i - c\,\Delta_r = \sqrt{((x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2)}$$

  - 4 satellites = 4 equations with 4 unknowns

Global Positioning System

- The measurements are not accurate.
  - GPS does not consider leap seconds.
  - Atomic clocks of satellites are not in perfect sync.
  - The position of a satellite is not known precisely.
  - The receiver's clock has a finite accuracy.
  - Signal propagation is not constant.
  - Earth is not a perfect sphere.
- Computing a position and time is far from trivial.
- Nevertheless, GPS offers good accuracy:
  - Professional receivers: 20-35 nanosecs.

Time synchronization

- Suppose that one computer has a shortwave time pulse receiver.
- The goal is to synchronize other machines with the time provided by the receiver...

Time synchronization

- Suppose that one computer has a shortwave time pulse receiver.
- The goal is to synchronize other machines with the time provided by the receiver...
- … and then, to keep the machines in sync.

Time synchronization

- Assumptions:
  - Each machine, P, has a timer that ticks H times per second.

Time synchronization

- Assumptions:
  - Each machine, P, has a timer that ticks H times per second.
  - The timer is used as a base of P's clock that ticks on each interrupt. Let's denote the value of this clock at UTC time t as $C_p(t)$.

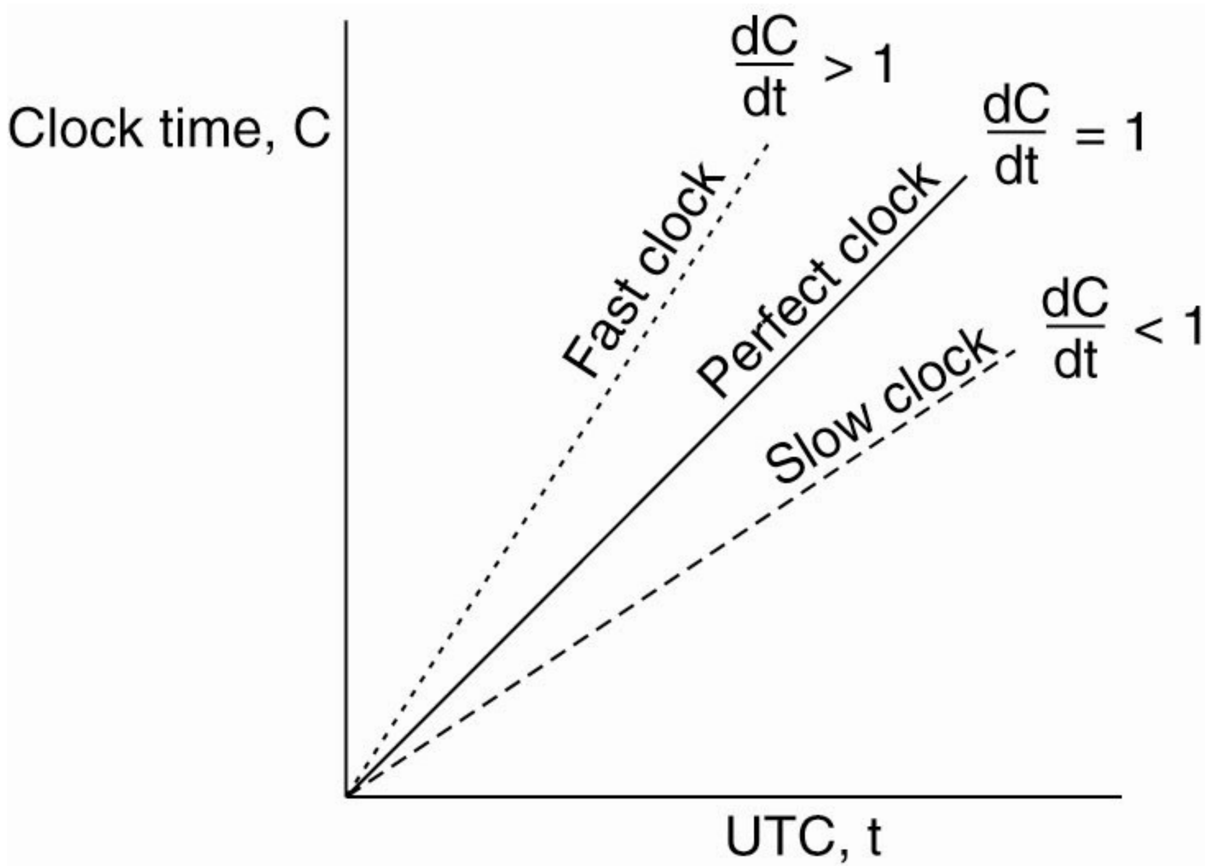
Time synchronization

- Assumptions:
  - Each machine, P, has a timer that ticks H times per second.
  - The timer is used as a base of P's clock that ticks on each interrupt. Let's denote the value of this clock at UTC time t as $C_p(t)$.
- Ideally, we would like to have $C_p(t) = t$, that is:
  - $dC / dt = 1$.

Time synchronization

- Real timers do not interrupt exactly H times per second.
  - In theory, with H = 60, we should have 216,000 ticks per hour.
  - In practice, with modern oscillators, the relative error is about 10^{-5}:
    - Between 215,998 and 216,002 ticks per hour.
- Clock skew = $C_p(t) - 1$

Time synchronization



Time synchronization

- In practice, for a given clock, there exists a maximum drift rate, ρ:

  $$1 - \rho \leq dC / dt \leq 1 + \rho$$

Time synchronization

- In practice, for a given clock, there exists a maximum drift rate, ρ:

  $1 - \rho \leq dC / dt \leq 1 + \rho$

- Goal: Never let two clocks drift more than δ time units.

Time synchronization

- In practice, for a given clock, there exists a maximum drift rate, ρ:

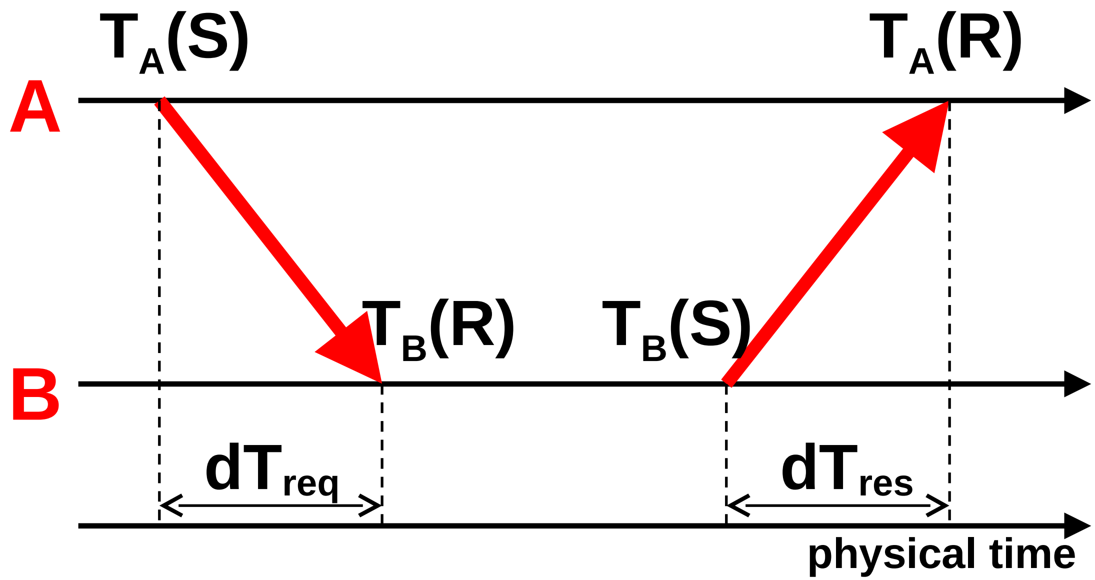
  $1 - \rho \leq dC / dt \leq 1 + \rho$

- Goal: Never let two clocks drift more than δ time units.
- Solution: Resynchronize at least every $\delta / (2\rho)$ time units.

Time synchronization

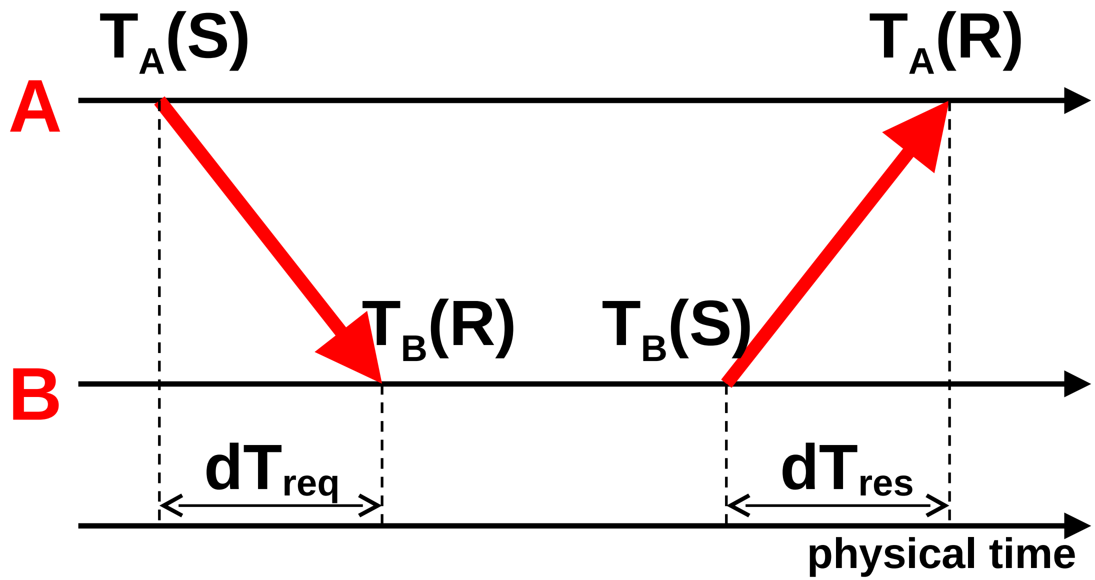
- Approach I:
  - Every machine asks a time server for the current time at least every $\delta / (2\rho)$ time units (Network Time Protocol – NTP).

Time synchronization



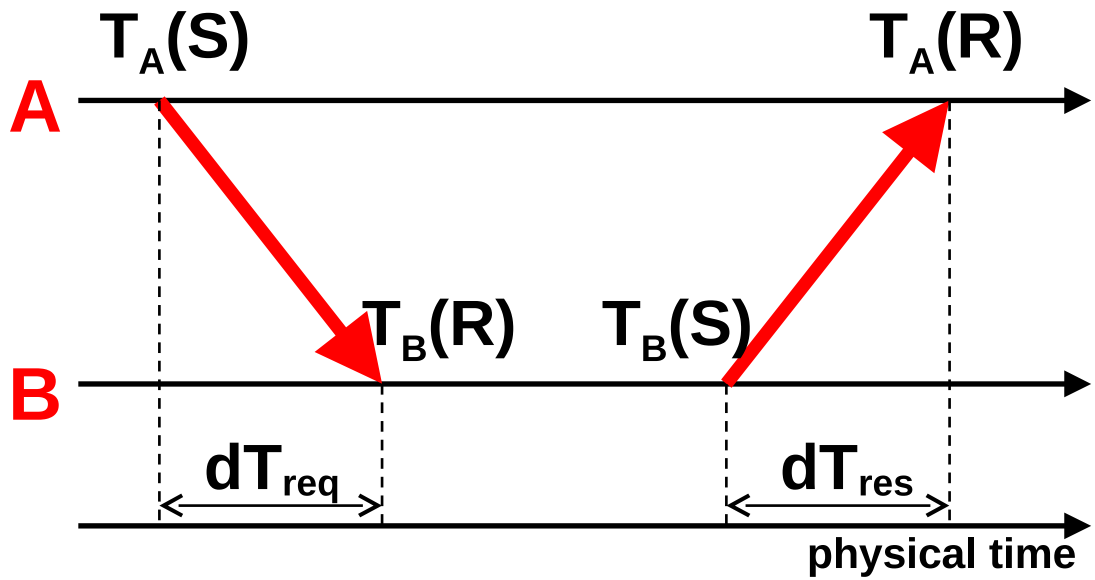
- Assuming $dT_{req} = dT_{res} = 0$, A's offset from B:
  - $\theta = T_B(S) - T_A(R)$

Time synchronization



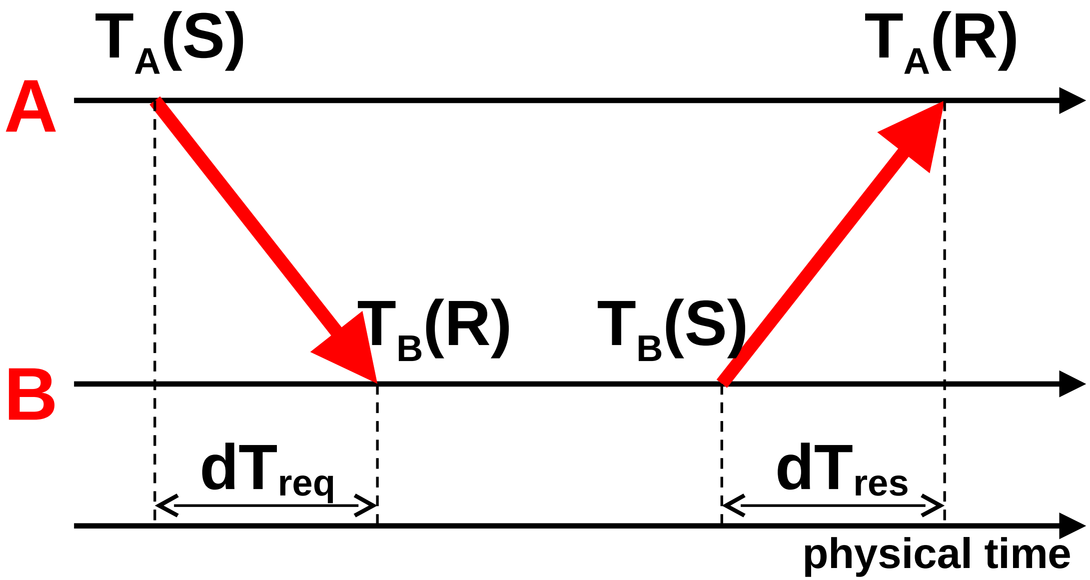
- Assuming $dT_{req} = dT_{res} = 0$, A's offset from B:
  - $\boldsymbol{\theta} = T_B(S) - T_A(R)$
- In practice, dT_{req} , $dT_{res} > 0$

Time synchronization



- Assuming $dT_{req} = dT_{res} = 0$, A's offset from B:
  - $\boldsymbol{\theta} = T_B(S) - T_A(R)$
- In practice, dT_{req} , $dT_{res} > 0$
- Problem: How to estimate the offset?

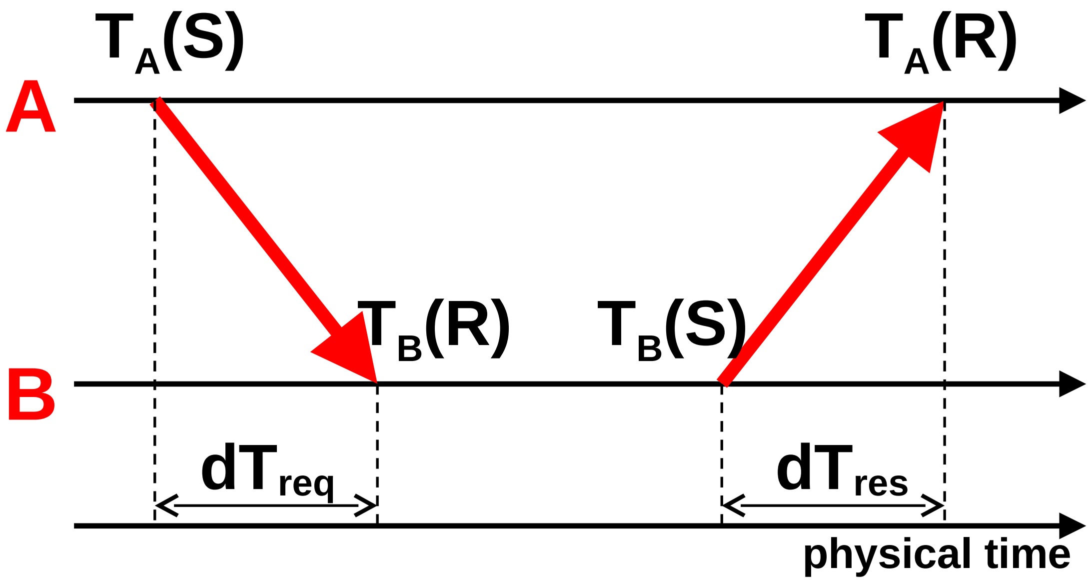
Time synchronization



- Round-trip delay:

$$\boldsymbol{\delta} = T_A(R) - T_A(S) - (T_B(S) - T_B(R))$$

Time synchronization

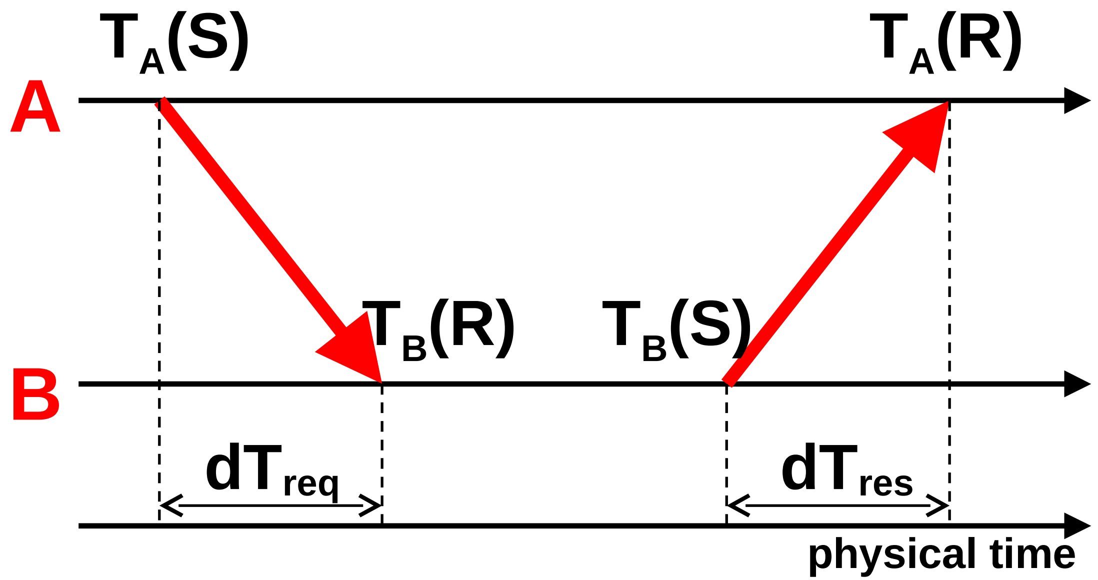


- Round-trip delay:

$$\delta = T_A(R) - T_A(S) - (T_B(S) - T_B(R))$$

- Assume $dT_{req} = dT_{res}$

Time synchronization



- Round-trip delay:

$$\boldsymbol{\delta} = T_A(R) - T_A(S) - (T_B(S) - T_B(R))$$

- Assume $dT_{req} = dT_{res}$
- Time offset: $\boldsymbol{\theta} = T_B(S) + \frac{1}{2} \times \delta - T_A(R)$

Time synchronization

- Assuming $dT_{req} = dT_{res}$ introduces errors.
- The reasons for errors:
  - Network delays
  - Interrupt handling
  - OS delays
  - Message processing

Time synchronization

- NTP:
  - estimates errors using round trip delays.
  - rejects samples that suffer from large errors.
  - divides servers into strata:
    - Stratum 0: an atomic clock
    - Stratum 1: a machine with shortwave time pulse receiver
    - Stratum i + 1: a machine that obtained its time from synchronizing with a stratum-i machine
- NTP's accuracy (world-wide): 1-50 ms
- Stratum-less synchronization: Gossiping Time Protocol (GTP).

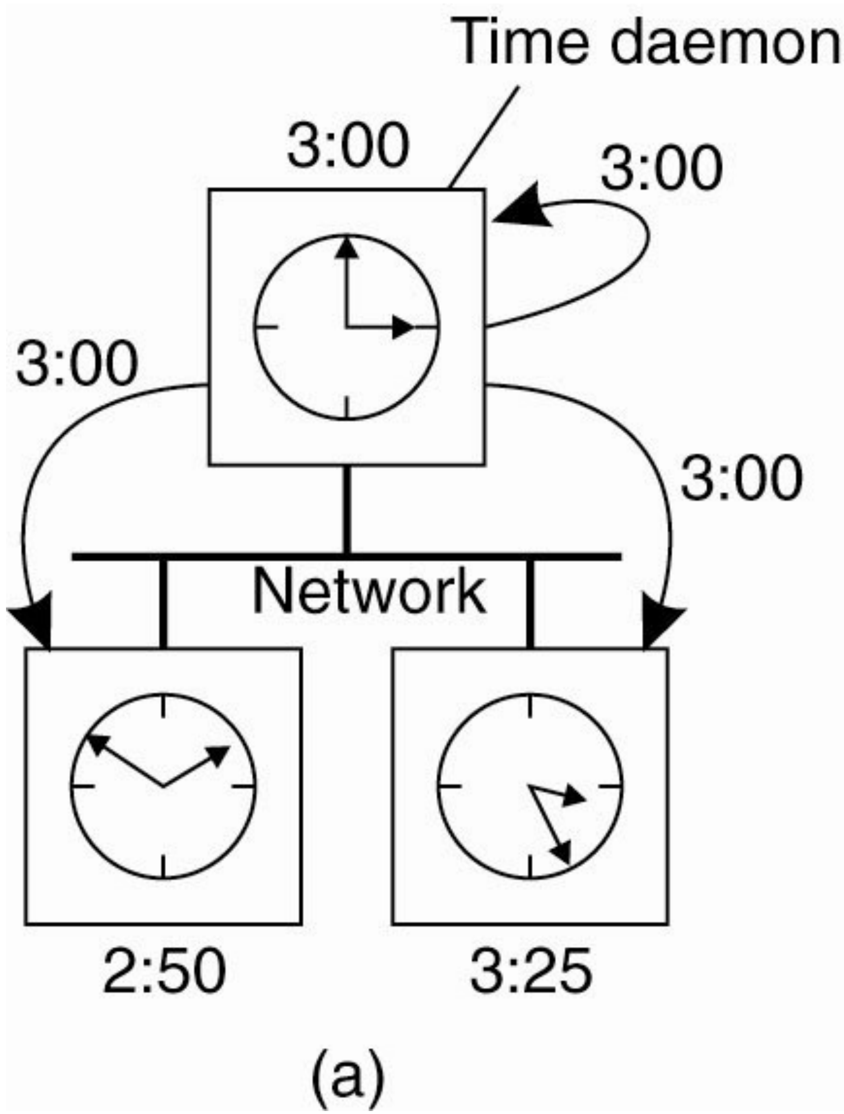
Time synchronization

- Approach II:
  - NTP provides external synchronization (to a stratum-0 clock).
  - An alternative is internal synchronization:
    - Machines synchronize with each other.
    - Not necessarily with an external clock.

Time synchronization

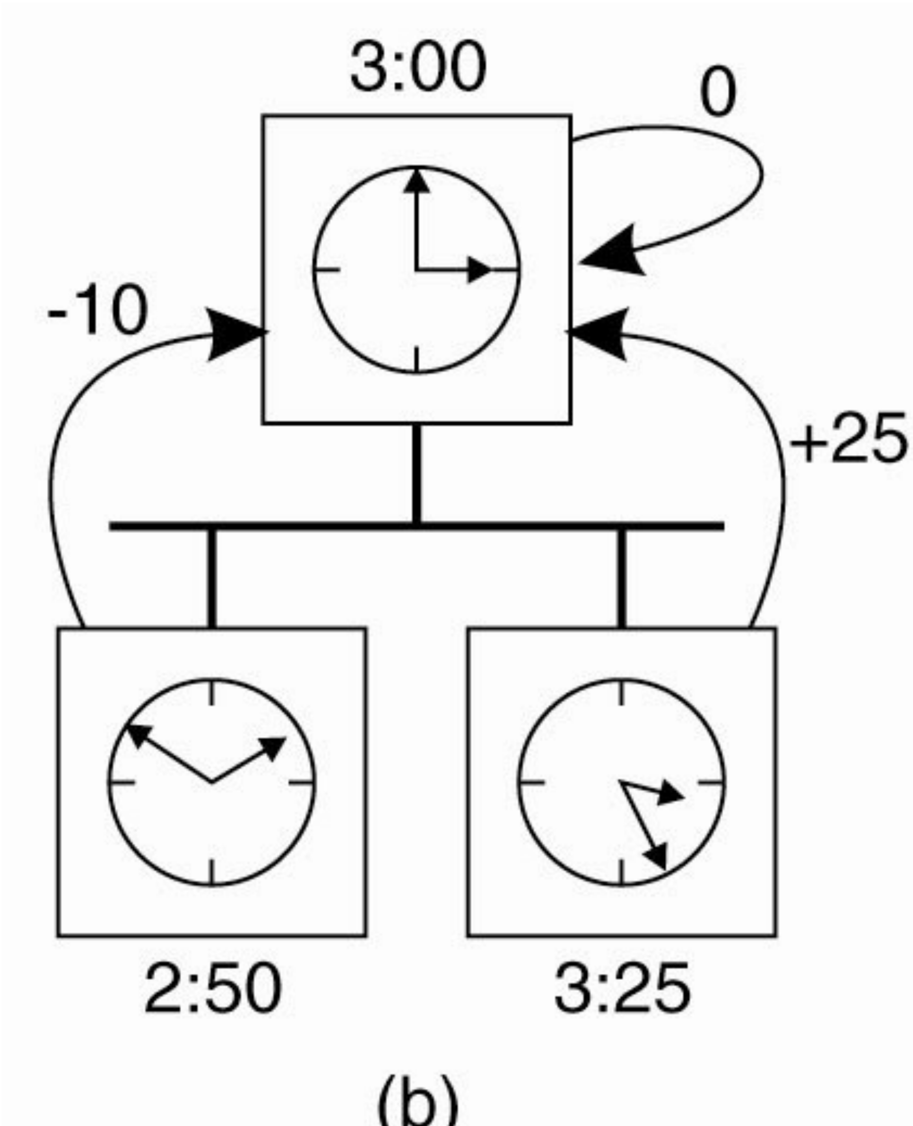
- The Berkeley algorithm:
  - Works in a local area network.
  - A special process, time daemon is responsible for synchronizing clocks of different machines.

Time synchronization



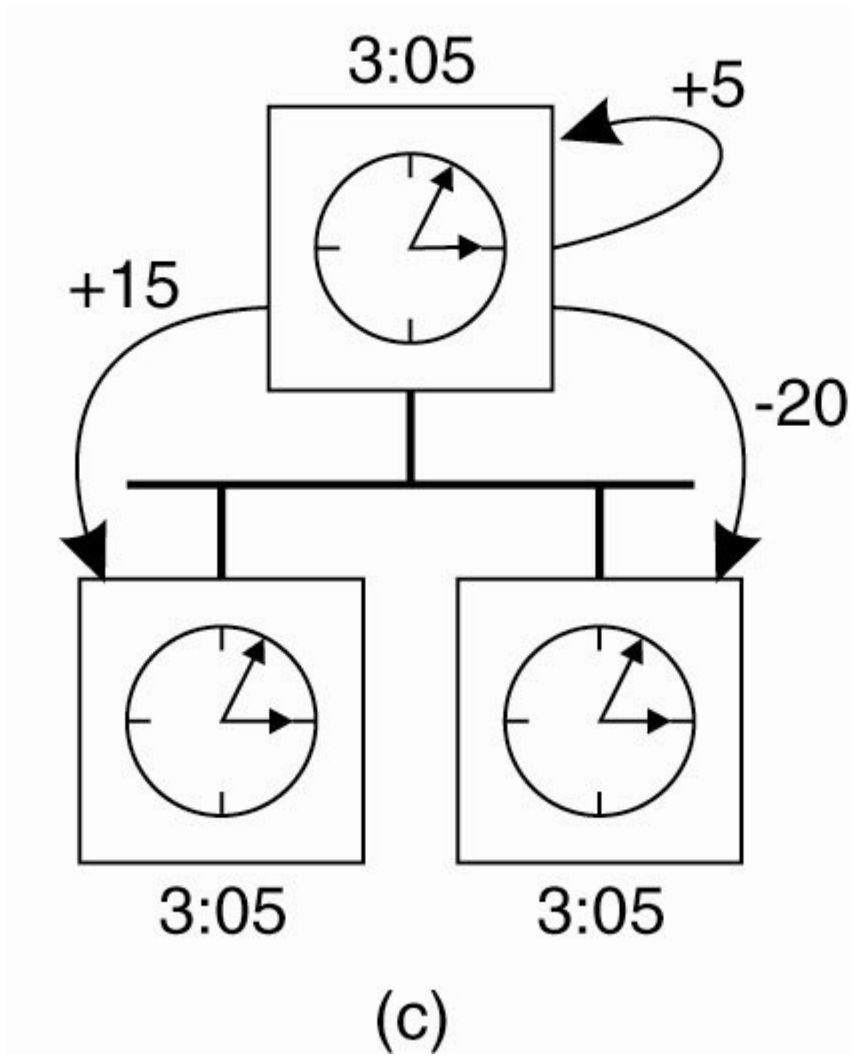
The time daemon periodically asks other machines for their local time.

Time synchronization



The machines
reply with their offsets.

Time synchronization



The time deamon
tells each machine
how to adjust its clock.

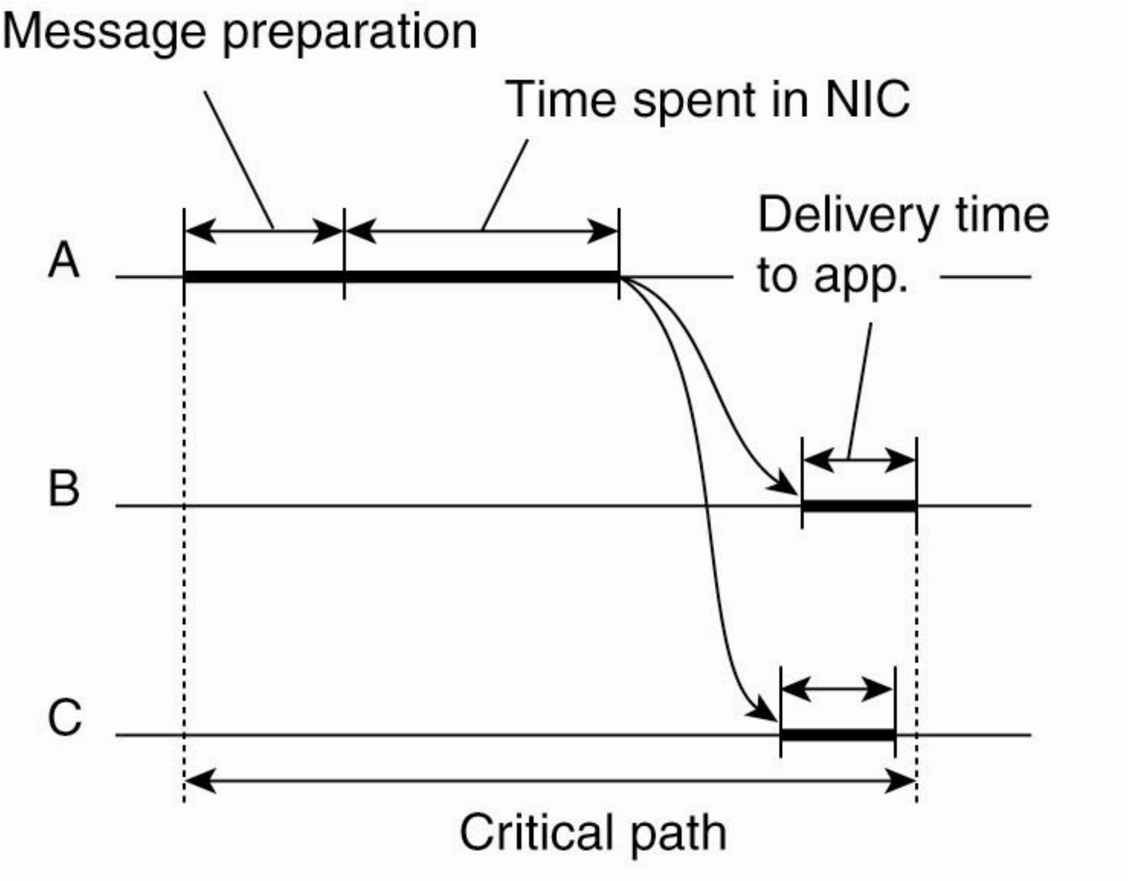
Time synchronization

- Approach III:
  - Wireless sensor networks require tight time synchronization:
    - e.g., seismic activity monitoring
  - On the other hand, they are built of inexpensive hardware.
  - Special algorithms are necessary.
    - e.g., Reference Broadcast Synchronization (RBS)

Time synchronization

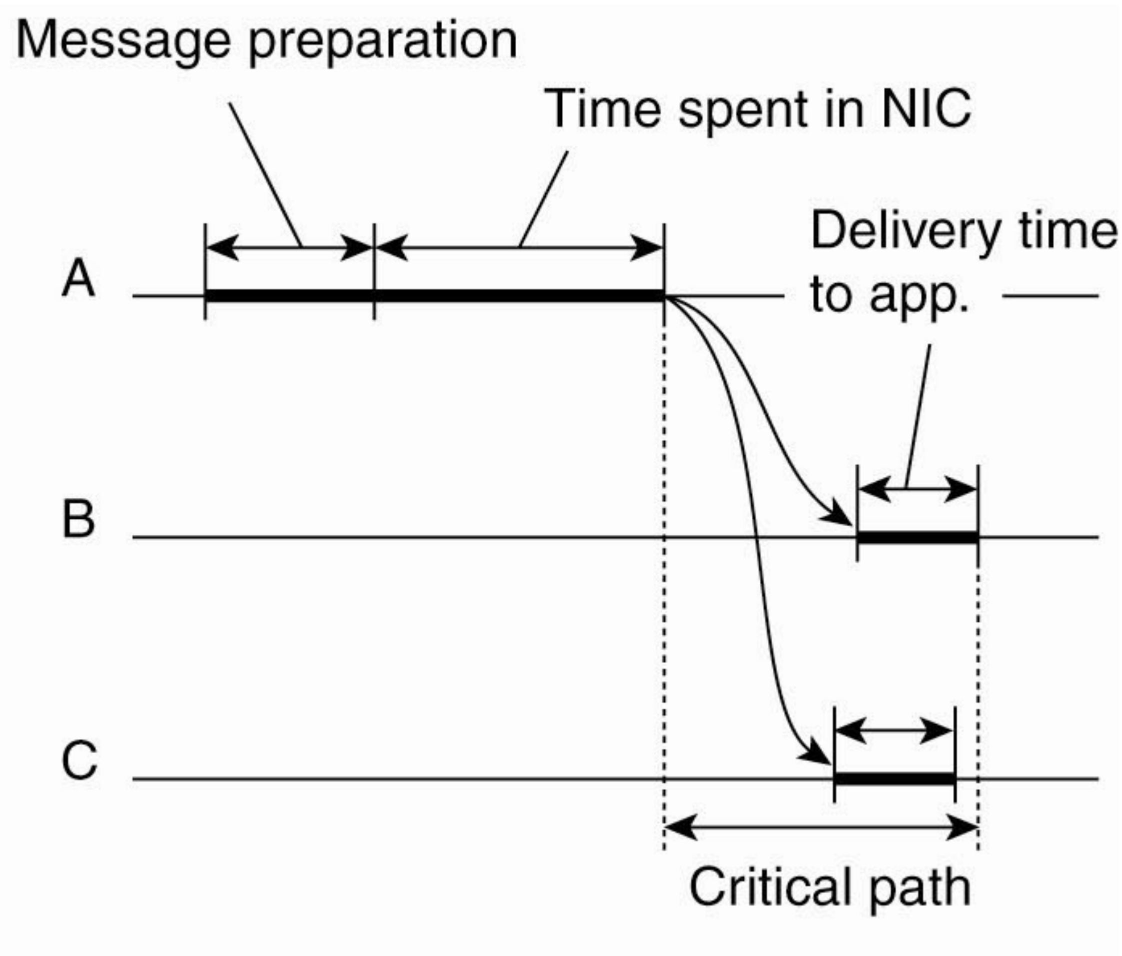
- Approach III:
  - Wireless sensor networks require tight time synchronization:
    - e.g., seismic activity monitoring
  - On the other hand, they are built of inexpensive hardware.
  - Special algorithms are necessary.
    - e.g., Reference Broadcast Synchronization (RBS)
- Idea: To eliminate various delays that introduce synchronization errors.

Time synchronization



(a)

Time synchronization



(b)

Time synchronization

- RBS:
  - A node broadcasts a reference message.

Time synchronization

- RBS:
  - A node broadcasts a reference message.
  - Each node, p, records the local time of reception, t^m_p.

Time synchronization

- RBS:
  - A node broadcasts a reference message.
  - Each node, p, records the local time of reception, t^m_p.
  - Nodes exchange their recorded reception times.

Time synchronization

- RBS:
  - A node broadcasts a reference message.
  - Each node, p, records the local time of reception, t^m_p.
  - Nodes exchange their recorded reception times.
  - Each node can compute its offset to another node.

Time synchronization

- RBS:
  - A node broadcasts a reference message.
  - Each node, p, records the local time of reception, t^{m}_{p}.
  - Nodes exchange their recorded reception times.
  - Each node can compute its offset to another node.

- Extremely tight synchronization: 1.85 ± 2.57 μs

Time-dependent Algorithms

What if one had globally synchronized clocks?

Time-dependent Algorithms

General rules:

- **It is preferable not to depend on clock synchronization for *correctness* but only for *performance*.**
- It is preferable not to depend on time being precisely synchronized but rather on clock rates being more or less similar.
- It is advised to monitor (or even enforce) synchronization not just merely rely on it (if ones does not fully control the synchronization algorithms).

Time-dependent Algorithms

Sample applications:

- security;
- cache consistency;
- atomicity with optimistic concurrency control;
- stronger consistency guarantees.

Time-dependent Algorithms

Sample applications:

- **security**;
- cache consistency;
- atomicity with optimistic concurrency control;
- stronger consistency guarantees.

- Give session keys used for authentication an expiration time.
  - A server will accept a client only if its session key has not expired (according to the server's clock).
- Prevent message replays with time-based authenticators:
  - An authenticator is a timestamp encrypted by a client with its session key.
  - It is added to each message from the client to the server.
  - If the authenticator in a message is too old, the server rejects the message.
  - The server also maintains a list of recent message authenticators and reject duplicated messages.
  - In effect, storage at the server is reduced.

Time-dependent Algorithms

Sample applications:

- security;
- **cache consistency**;
- atomicity with optimistic concurrency control;
- stronger consistency guarantees.

- Hand out leases to cached items, for instance, files in a distributed file system:
  - Each client obtains a lease for a file when the file is copied to its cache.
  - The lease contains an expiration time.
- When a lease expires, the client either:
  - stops using the file or
  - requests a lease renewal from the server.
- When some client modifies a file in its cache:
  - The modification is forwarded to the server.
  - The server then revokes all leases for the file but for the client that has done the modification.
  - When all leases have been relinquished (or have expired), the server applies the modification(s).

Time-dependent Algorithms

Sample applications:

- security;
- cache consistency;
- **atomicity with optimistic concurrency control**;
- stronger consistency guarantees.

- Consider a transactional system with optimistic concurrency control, for instance, for objects:
  - Objects can be operated on.
  - Each object has a version.
- When a client wants to access an object:
  - It downloads to its cache the current version of the object from the server.
  - It performs all operations on the copy of the object in its cache in a transaction.
  - When it is done, it sends the updated version back to the server.
- The server compares the original version number of the updated version with the local version:
  - If the numbers match, the local copy is replaced with updated one and gets a new version number.
  - Otherwise, the client is informed that its transaction has aborted.
- To reduce the likelihood of aborts, the server uses leases to notify clients accessing a given object whenever it is updated.

Time-dependent Algorithms

Sample applications:

- security;
- cache consistency;
- atomicity with optimistic concurrency control;
- **stronger consistency guarantees.**

- Sequential consistency is powerful, yet sometimes unintuitive.
- Stronger consistency guarantees – tying operation order with real time – can prevent such situations.

Question: Example?

Time-dependent Algorithms

- Children phone a parent to transfer them some money as they have insufficient funds.
- The parent does the transfer and informs the children.
- The children withdraw money from an ATM (without checking the balance).
- The systems executes the two operations in an opposite order so that temporarily the balance is negative, causing some fee at the end of the month.

Time-dependent Algorithms

External Consistency*

The actual time order in which operations complete defines a unique serial schedule, a so-called *external schedule*. A system is said to provide external consistency if it guarantees that the schedule it will use to process a set of operations is equivalent to its external schedule.

* D.K. Gifford: "Information Storage in Decentralized Computer Systems," *Technical Report CSL-81-8*, Xerox Corporation, Palo Alto, CA, USA, March 1982, 153 pages.

Time-dependent Algorithms

- As a case study of a system providing a variant of external consistency, let us take Google's Spanner.
- It is a highly available global SQL-based database.
- It features multi-version data and lock-free globally consistent reads.
- It has replaced Google's BigTable and other custom solutions in a number of services and applications.

Time-dependent Algorithms

Overview:

- The database conceptually provides a mapping:

  (*key*, *timestamp*) → *value*

  where the *timestamps* correspond to real time and are assigned in a globally consistent manner from a dedicated service, TrueTime.
- The entire set of such mappings is *sharded* by *keys* into smaller sets, called tablets.
- Each tablet is independently state-machine-replicated onto a group of nodes, potentially in different data centers worldwide, using Paxos.
- If a transaction touches multiple tablets, two-phase commit (2PC) and two-phase locking are used to ensure atomicity; otherwise, 2PC can be bypassed for performance.
  - In this case, 2PC is not problematic for availability because it is done not over individual processes but fault-tolerant process groups (Paxos replication groups).

Time-dependent Algorithms

Overview:

- The database conceptually provides a mapping:

  (*key*, *timestamp*) → *value*

  where the *timestamp*s correspond to real time and are assigned in a globally consistent manner from a dedicated service, TrueTime.
- The entire set of such mappings is *sharded* by *key*s into smaller sets, called tablets.
- Each tablet is independently state-machine-replicated onto a group of nodes, potentially in different data centers worldwide, using Paxos.
- If a transaction touches multiple tablets, two-phase commit (2PC) and two-phase locking are used to ensure atomicity; otherwise, 2PC can be bypassed for performance.
  - In this case, 2PC is not problematic for availability because it is done not over individual processes but fault-tolerant process groups (Paxos replication groups).

Question: What consistency does the presented design aim at?

Time-dependent Algorithms

- Spanner uses these mechanisms to get sequential consistency.
- However, it provides even stronger guarantees, namely the following variant of external consistency:

  For any two transactions, T_1 and T_2 (even on opposite sites of the globe), if T_2 starts committing after T_1 has finished committing, then the timestamp for T_2 is greater than the timestamp for T_1.

Time-dependent Algorithms

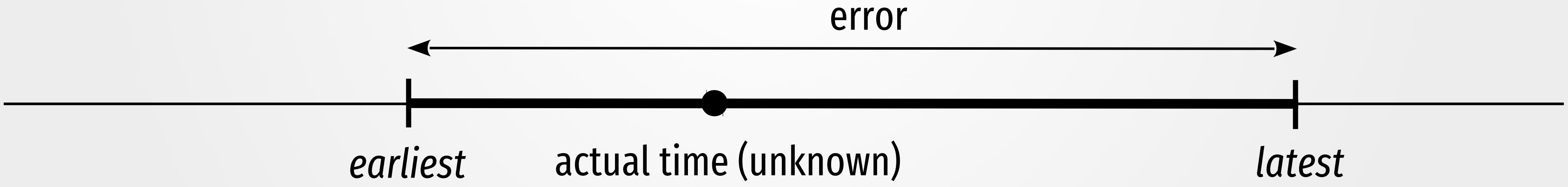
- Spanner uses these mechanisms to get sequential consistency.
- However, it provides even stronger guarantees, namely the following variant of external consistency:

  For any two transactions, T_1 and T_2 (even on opposite sites of the globe), if T_2 starts committing after T_1 has finished committing, then the timestamp for T_2 is greater than the timestamp for T_1.

Question: How to achieve this?

Time-dependent Algorithms

- The source of timestamps is the TrueTime service.
- Conceptually, it is a globally synchronized clock that exposes bounded non-zero errors to users.
- When read, it returns a time interval, [*earliest*, *latest*], that is guaranteed to contain the actual time for some time during the execution of the call.



- In particular, any process can know whether:
  - a given time moment is guaranteed to have already passed;
  - a given time moment is guaranteed not to have occurred yet.

Time-dependent Algorithms

- At the start of a commit of a transaction, the transaction is assigned a timestamp, *t*, no earlier than *latest* (i.e., *latest* ≤ *t*). For a given leader, these timestamps are also monotonic.
- No data written by the transaction is visible until *t* is guaranteed to have passed, that is, until for any process that queries the time, the returned lower bound, *earliest*, is after *t* (i.e., *t* < *earliest* for that process).
- Only after this waiting is the commit completed, the locks released, and the client notified.
- The timestamps can also be used to perform lock-free read-only transactions on consistent snapshots of the database:
  - One needs to ensure that the snapshot time is guaranteed to have passed.

Question: What is crucial for this design to work in practice?

Time-dependent Algorithms

- The error bounds on time should be as low as possible.
- TrueTime achieves this with a combination of:
  - GPSes,
  - atomic clocks,
  - private inter-datacenter networks,
  - clock synchronization algorithms estimating clock drift over time and eliminating outliers.
- In effect, the error bound most of the time is below 7 ms.
  - Hard to replicate on a global scale if you are not Google.
- **Important**: The synchronization errors influence performance not correctness: the waiting time simply gets longer but as long as the invariant about the returned time interval holds everywhere, safety is preserved.

Summary

We have:

- discussed time and methods of measuring it;
- formulated the problem of clock synchronization in distributed systems;
- presented several clock synchronization techniques;
- given sample applications of synchronized clocks.

Digression

Law of diminishing returns

The more one improves some measure of goodness, the more effort the next improvement will require.

Digression

Safety margin principle

Keep track of the distance to the cliff, or you may fall over the edge.

Next Lecture

- Will be about the broadest (and most challenging) class of failures to deal with: Byzantine failures.
- More specifically, we will revisit the three fundamental abstractions under this type of failures.