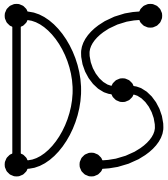


Research Group
Distributed Systems

Distributed Systems

Clocks & Time I

Olaf Landsiedel

Labs

- Today a single group, Room “F”
  - Q&A lab 2
  - Missing presentations

Last Time

- Topic:
  - Naming

Distributed Name Resolution

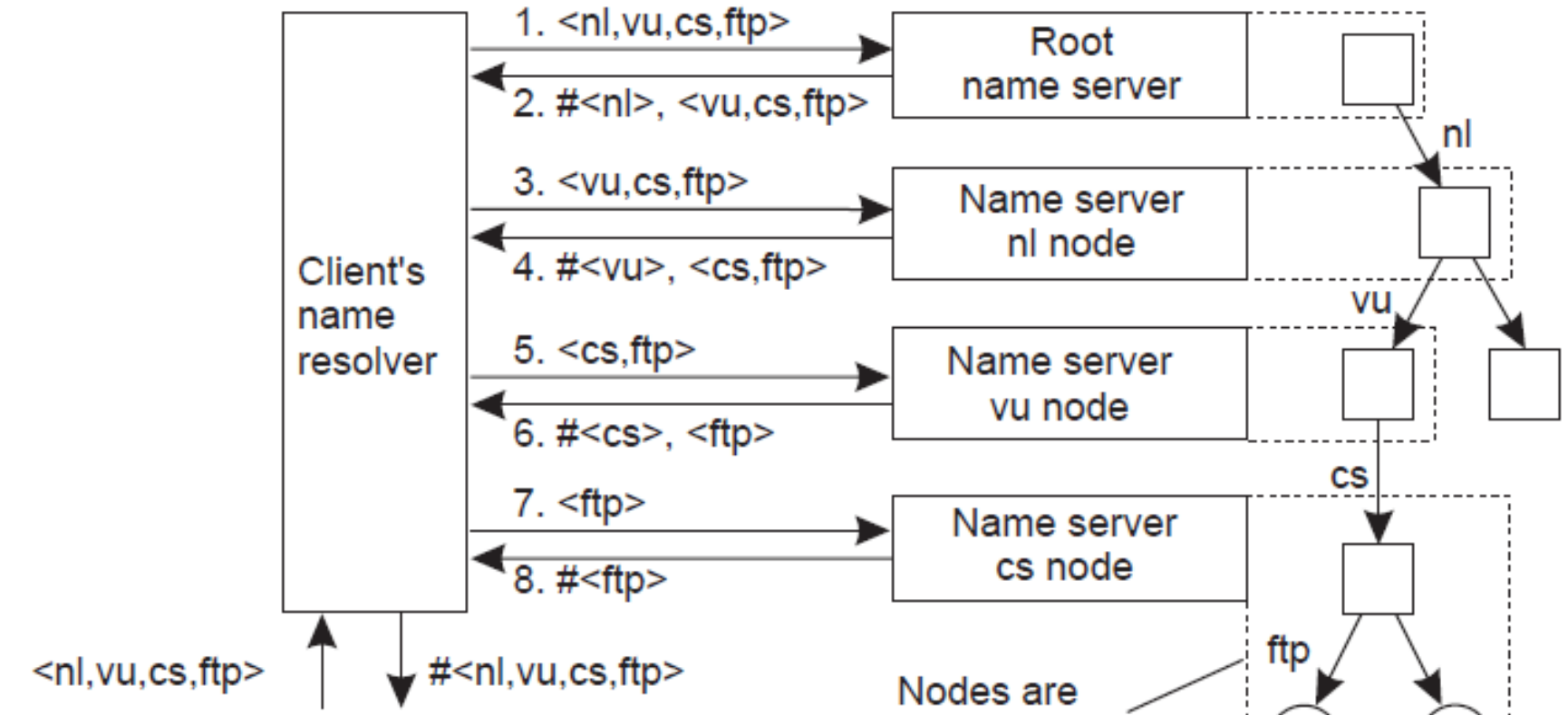
- Distributed Name Resolution is responsible for mapping names to address in a system where:
  - Name servers are distributed among participating nodes
  - Each name server has a local *name resolver*

- We will study two distributed name resolution algorithms:
  1. Iterative Name Resolution
  2. Recursive Name Resolution

1. Iterative Name Resolution

1. Client hands over the complete name to *root name server*
2. Root name server resolves the name as far as it can, and returns the result to the client
   - The root name server returns the address of the next-level name server (say, NLNS) if address is not completely resolved
3. Client passes the unresolved part of the name to the NLNS
4. NLNS resolves the name as far as it can, and returns the result to the client (and probably its next-level name server)
5. The process continues till the full name is resolved

1. Iterative Name Resolution – An Example



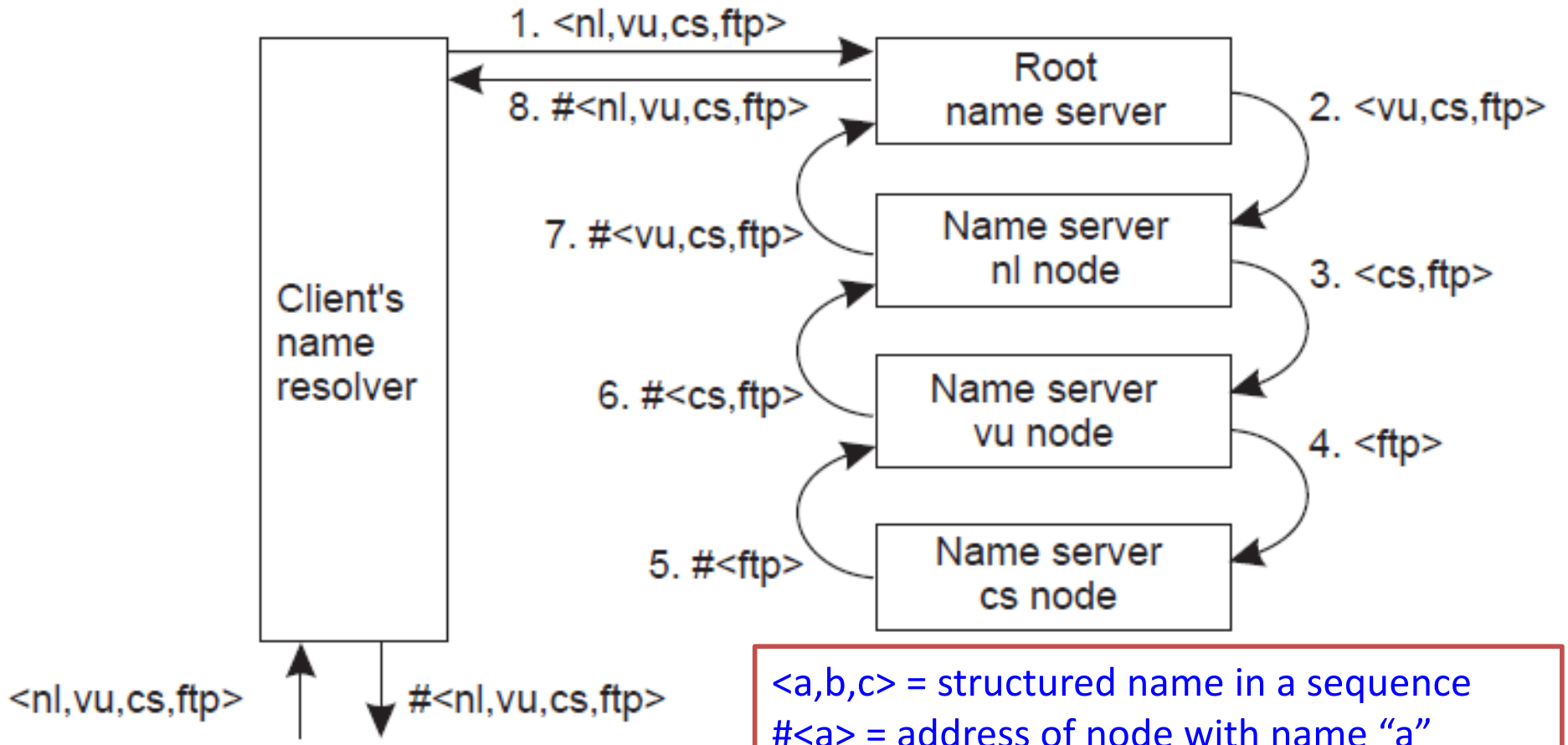
<a,b,c> = structured name in a sequence
#<a> = address of node with name "a"

Resolving the name "*ftp.cs.vu.nl*"

2. Recursive Name Resolution

- Approach
  - Client provides the name to the root name server
  - The root name server passes the result to the next name server it finds
  - The process continues till the name is fully resolved
- Drawback:
  - Large overhead at name servers (especially, at the high-level name servers)

2. Recursive Name Resolution – An Example



Resolving the name "*ftp.cs.vu.nl*"

Attribute-based Naming

- In many cases, it is much more convenient to name, and look up entities by means of their attributes
  - Similar to traditional directory services (e.g., yellow pages)
- However, the lookup operations can be extremely expensive
  - They require to match requested attribute values, against actual attribute values, which needs to inspect all entities
- Solution: Implement basic directory service as database, and combine with traditional structured naming system
- We will study Light-weight Directory Access Protocol (LDAP); an example system that uses attribute-based naming

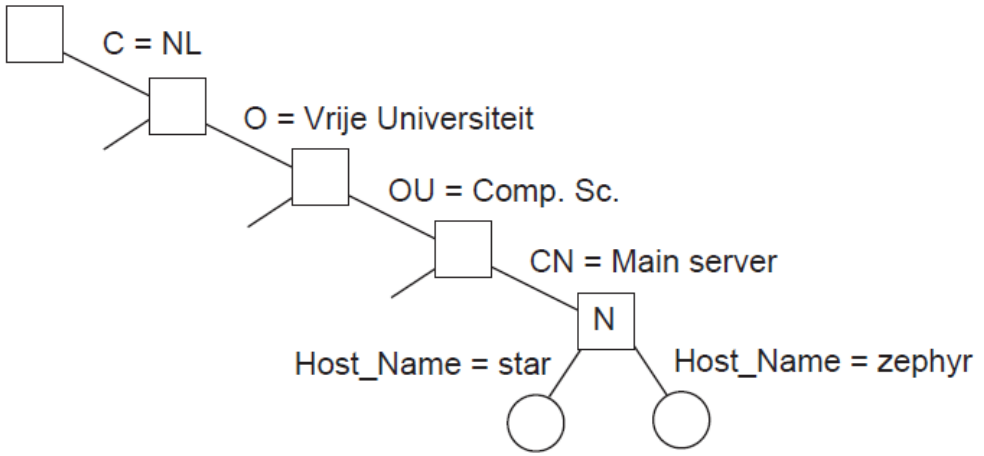
Light-weight Directory Access Protocol (LDAP)

- LDAP Directory Service consists of a number of records called “directory entries”
  - Each record is made of (attribute, value) pair
  - LDAP Standard specifies five attributes for each record
- Directory Information Base (DIB) is a collection of all directory entries
  - Each record in a DIB is unique
  - Each record is represented by a
  - distinguished name
  - e.g., /C=NL/O=Vrije Universiteit/OU=Comp. Sc.

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Directory Information Tree in LDAP

- All the records in the DIB can be organized into a hierarchical tree called Directory Information Tree (DIT)
- LDAP provides advanced search mechanisms based on attributes by traversing the DIT
- Example syntax for searching all Main_Servers in Vrije Universiteit:
- search("&(C = NL) (O = Vrije Universiteit) (OU = *) (CN = Main server)")



Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	137.37.20.10

Summary

- Naming and name resolutions enable accessing entities in a Distributed System
- Three types of naming
  - Flat Naming
    - Home-based approaches, Distributed Hash Table
  - Structured Naming
    - Organizes names into Name Spaces
    - Distributed Name Spaces
  - Attribute-based Naming
    - Entities are looked up using their attributes

8:00:00

8:00:00

Sept 18, 2006
8:00:00

5 Weeks later?

8:00:00

8:00:00

Sept 18, 2006
8:00:00

Clocks drift -> Synchronization

8:01:24

Skew = +84 seconds

+84 seconds/35 days

Drift = +2.4 sec/day

Oct 23, 2006

8:00:00

8:01:48

Skew = +108 seconds

+108 seconds/35 days

Drift = +3.1 sec/day

Outline

- Motivation
  - Learning outcome: know basic problem

- **Why:** clock synchronization?
  - Outcome: learn why clock synchronization matters

- **How:** clock synchronization algorithms
  - Outcome: explore how it can be done

Without Synchronized Clocks…

Bus at 11:50?

Friend at 3pm?

Class at 10:00?

… anything involving other people is difficult to setup.

Without Time Synchronization?

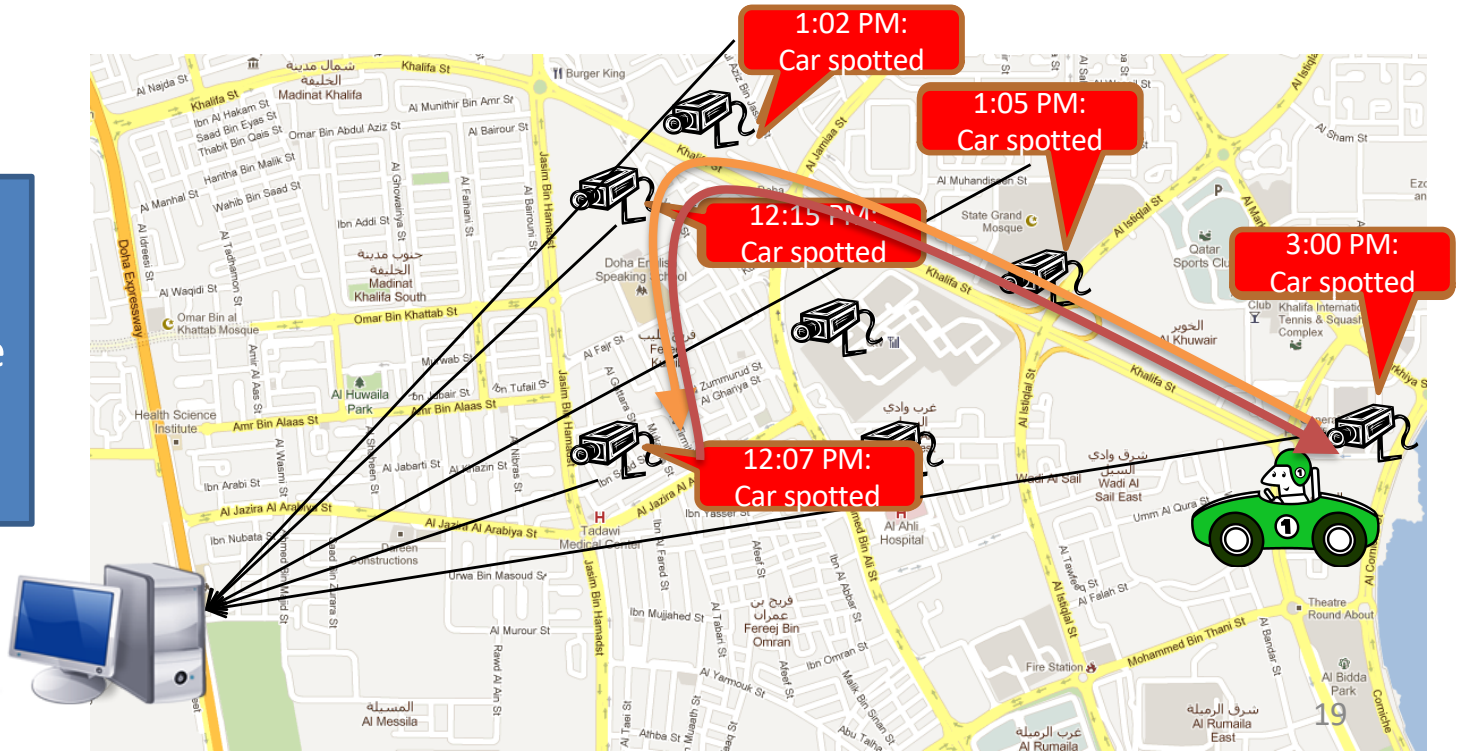
- What is time synchronization good for?

- Imagine
  - The clocks we just looked at would be used
    - In computers?
    - In cell phones?
    - In banks?
  - What could go wrong?

Example 1: Vehicle Tracking

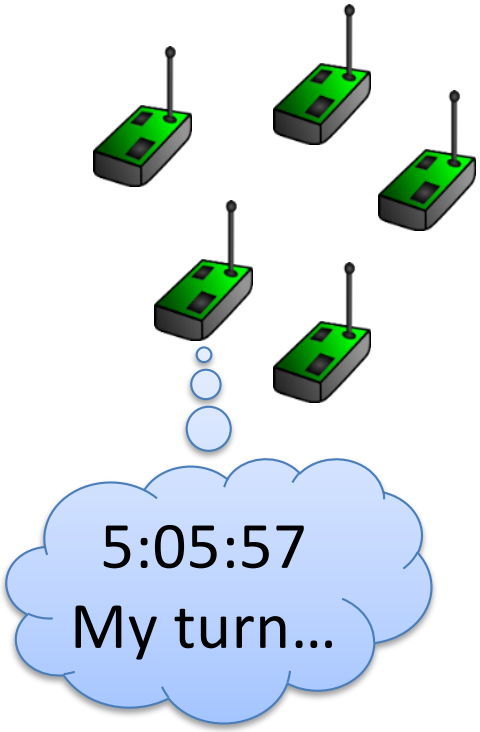
- Vehicle tracking in a City Surveillance System using a Distributed Sensor Network of Cameras
  - *Objective:* To keep track of suspicious vehicles
  - Camera Sensor Nodes are deployed over the city
  - Each Camera Sensor that detects the vehicle reports the time to a central server
  - Server tracks the movement of the suspicious vehicle

If the sensor nodes do not have a consistent version of the time, the vehicle cannot be reliably tracked



Example 2: Medium Access Control

- Access to shared resources
  - Often based on time synchronization
- Example
  - TDMA for medium access control
    - Schedule based
  - Who defines the schedule?
    - Central entity
    - Distributed
- Which technologies use TDMA?
  - GSM, UMTS, LTE
  - Bluetooth



What is Time and Synchronization for?

- Temporal ordering of events produced by concurrent processes
- Synchronization between senders and receivers of messages
- Coordination of joint activity
- Serialization of concurrent access to shared objects / resources
- …

Outline

- Motivation
  - Learning outcome: know basic problem
- **Why:** clock synchronization?
  - Outcome: learn why clock synchronization matters
- **How:** clock synchronization algorithms
  - Outcome: explore how it can be done

Today & Next Time

- Clocks, time, and time synchronization
- Today: Physical clocks
- Next time: Logical clocks

Introduction, Drift, Synchronization

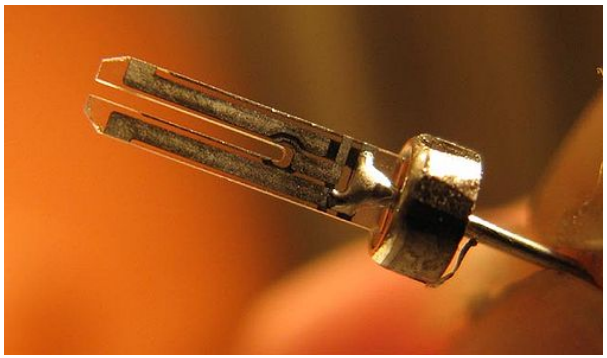
PHYSICAL CLOCKS

Logical vs. Physical Clocks

- Logical clock keeps track of event ordering
  - Among related (causal) events
    - Topic of next lecture
- Physical clocks keep time of day
  - Consistent across systems
    - Topic today

What makes a clock tick?

- 1880: Piezoelectric effect
  - Curie brothers (Jacques and Pierre Curie)
  - Squeeze a quartz crystal & it generates an electric field
  - Apply an electric field and it bends



- 1929: Quartz crystal clock
  - Resonator shaped like tuning fork
  - Laser-trimmed to vibrate at 32,768 kHz
  - Standard resonators accurate to 6 parts per million at 31° C
    - Watch will gain/lose < ½ sec/day
  - Stability > accuracy: stable to 2 sec/month
  - Good resonator can have accuracy of 1 second in 10 years
    - Frequency changes with age, temperature, and acceleration

Atomic Clocks

- Second is defined as 9,192,631,770 periods of radiation corresponding to the transition between two hyperfine levels of cesium-133

- Accuracy:
  better than 1 second in six million years

- NIST standard since 1960
- Coordinated Universal Time (UTC)

Physical Clocks in Computers

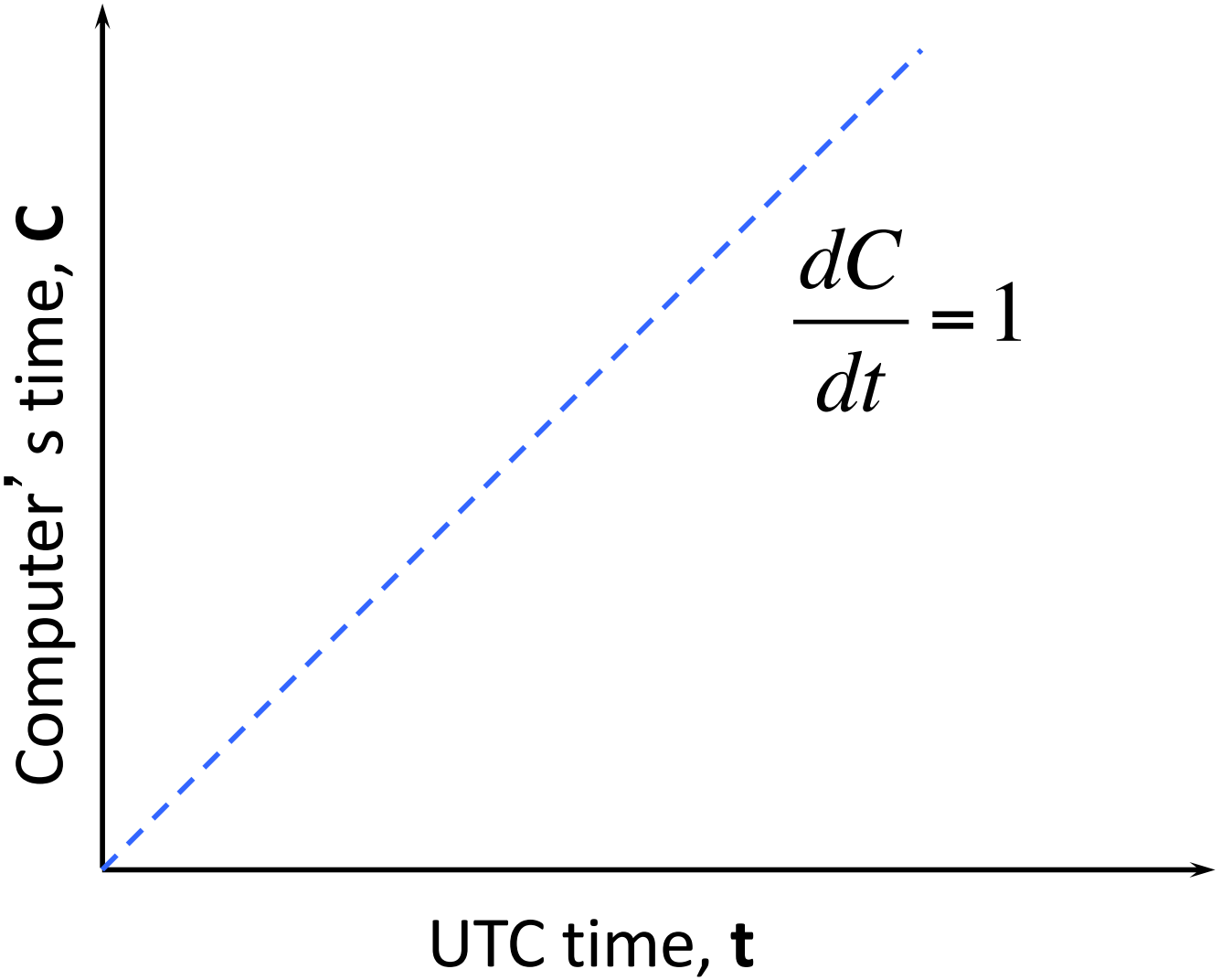
- Real-time Clock:
  CMOS clock (counter) circuit driven by a quartz oscillator
  - Battery backup to continue measuring time when power is off

- OS generally programs a timer circuit to generate an interrupt periodically
  - e.g., 60, 100, 250, 1000 interrupts per second
    (Linux 2.6+ adjustable up to 1000 Hz)
  - Programmable Interval Timer (PIT) – Intel 8253, 8254
  - Interrupt service procedure adds 1 to a counter in memory

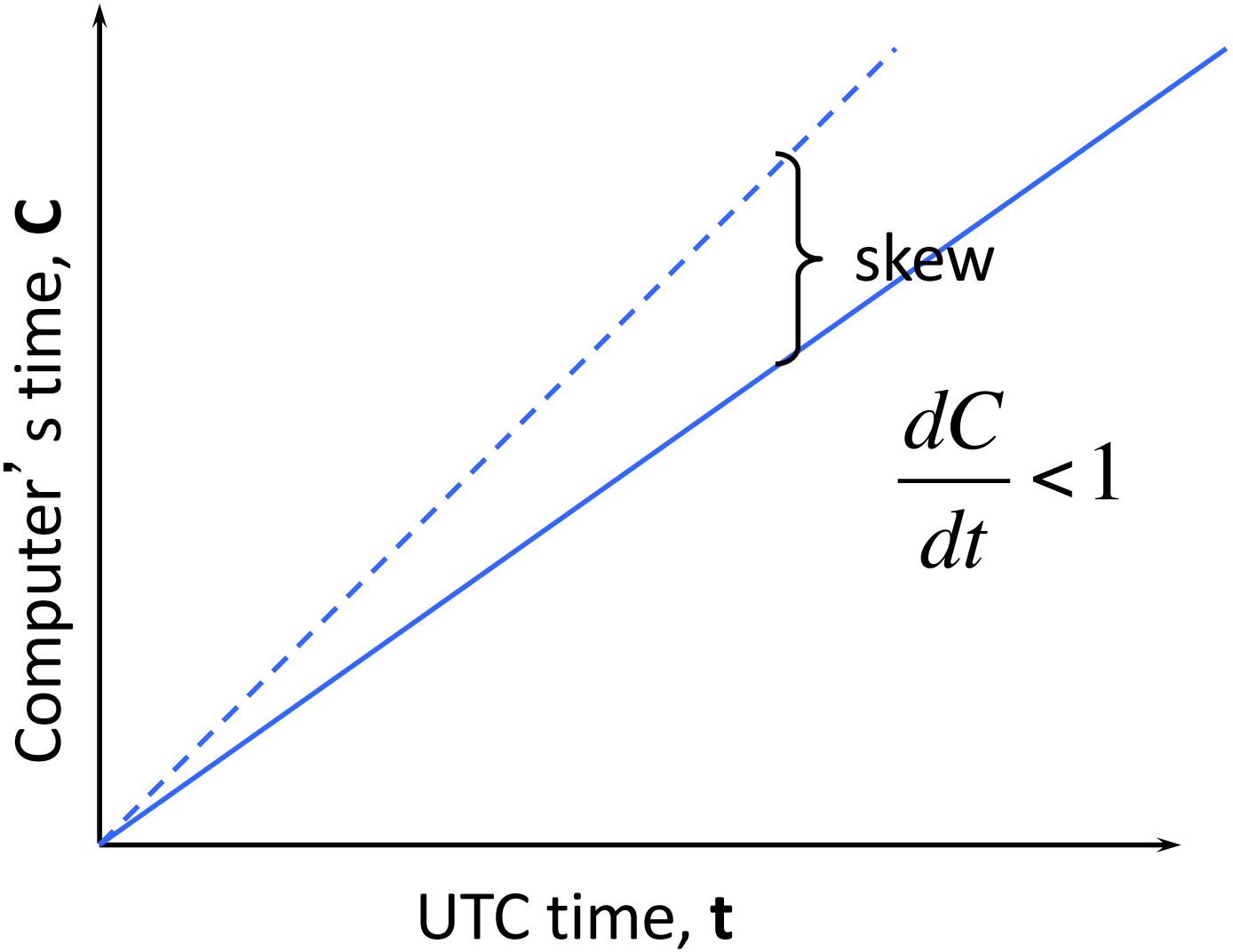
Problem

- Getting two systems to agree on time
  - Two clocks hardly ever agree
  - Quartz oscillators oscillate at slightly different frequencies
- **Clock Drift**
  - Clocks tick at different rates
  - Create ever-widening gap in perceived time
  - See example in the warm-up
- **Clock Skew**
  - Difference between two clocks at one point in time

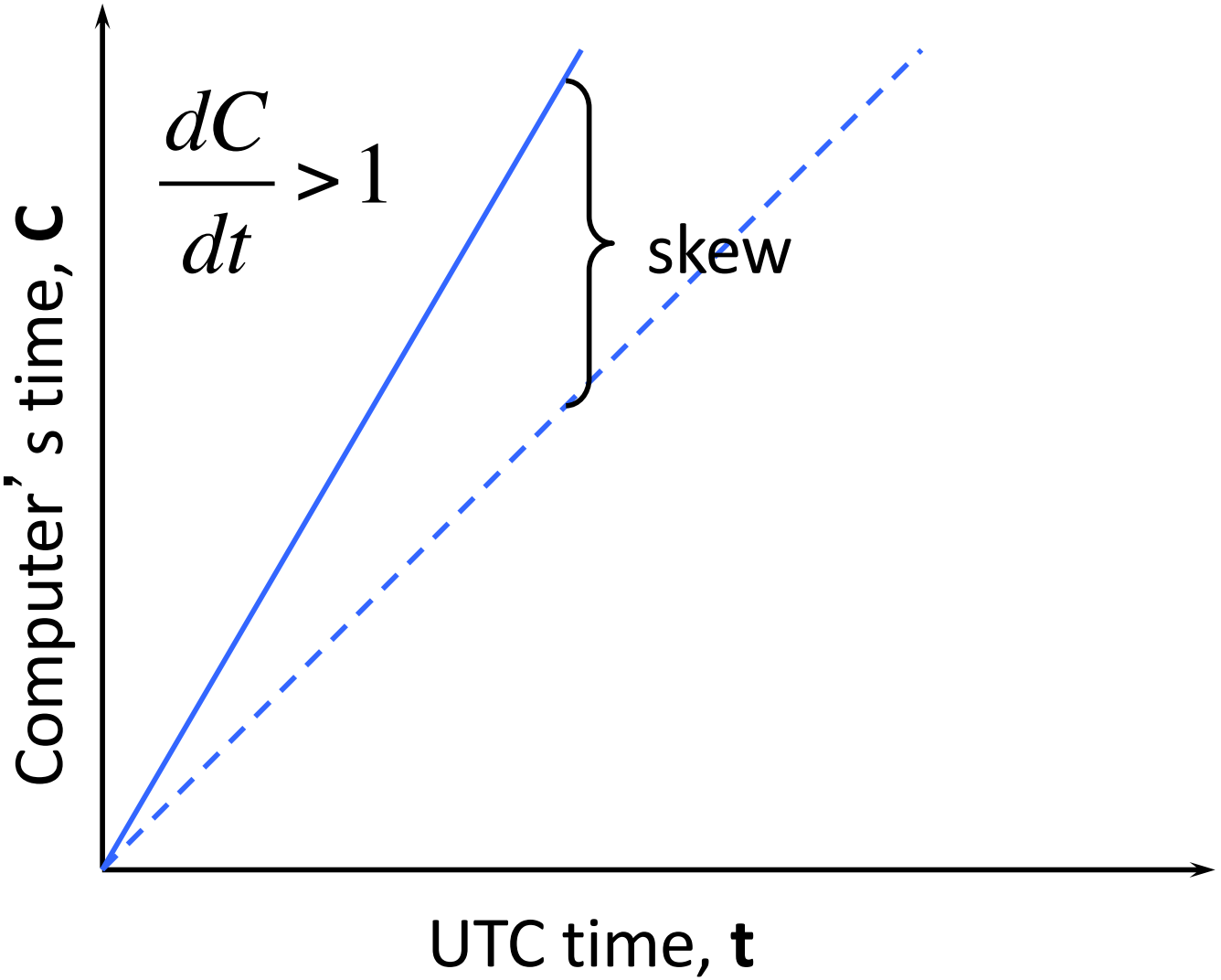
Perfect Clock



Drift with Slow Clock



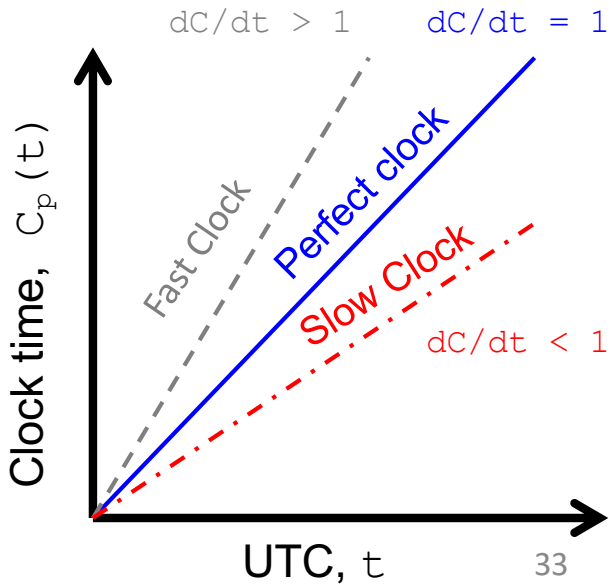
Drift with Fast Clock



Definitions

- **Frequency** of the clock is defined as the ratio of the number of seconds counted by the software clock for every UTC second
  - Frequency = dC/dt
- **Skew** of the clock is defined as the extent to which the frequency differs from that of a perfect clock
  - Skew = dC/dt – 1
- Hence,

$$Skew \begin{cases} > 0 & \text{for a fast clock} \\ = 0 & \text{for a perfect clock} \\ < 0 & \text{for a slow clock} \end{cases}$$



Dealing with Drift

- Assume we want set computer to true time
- Not good idea to set clock back or forward
  - Why not?

Why not?

- Example
  - Bank: At midnight send $2000 to X
  - Server: Backup at midnight

- Setting clock back?
- Setting clock forward?

Dealing with Drift

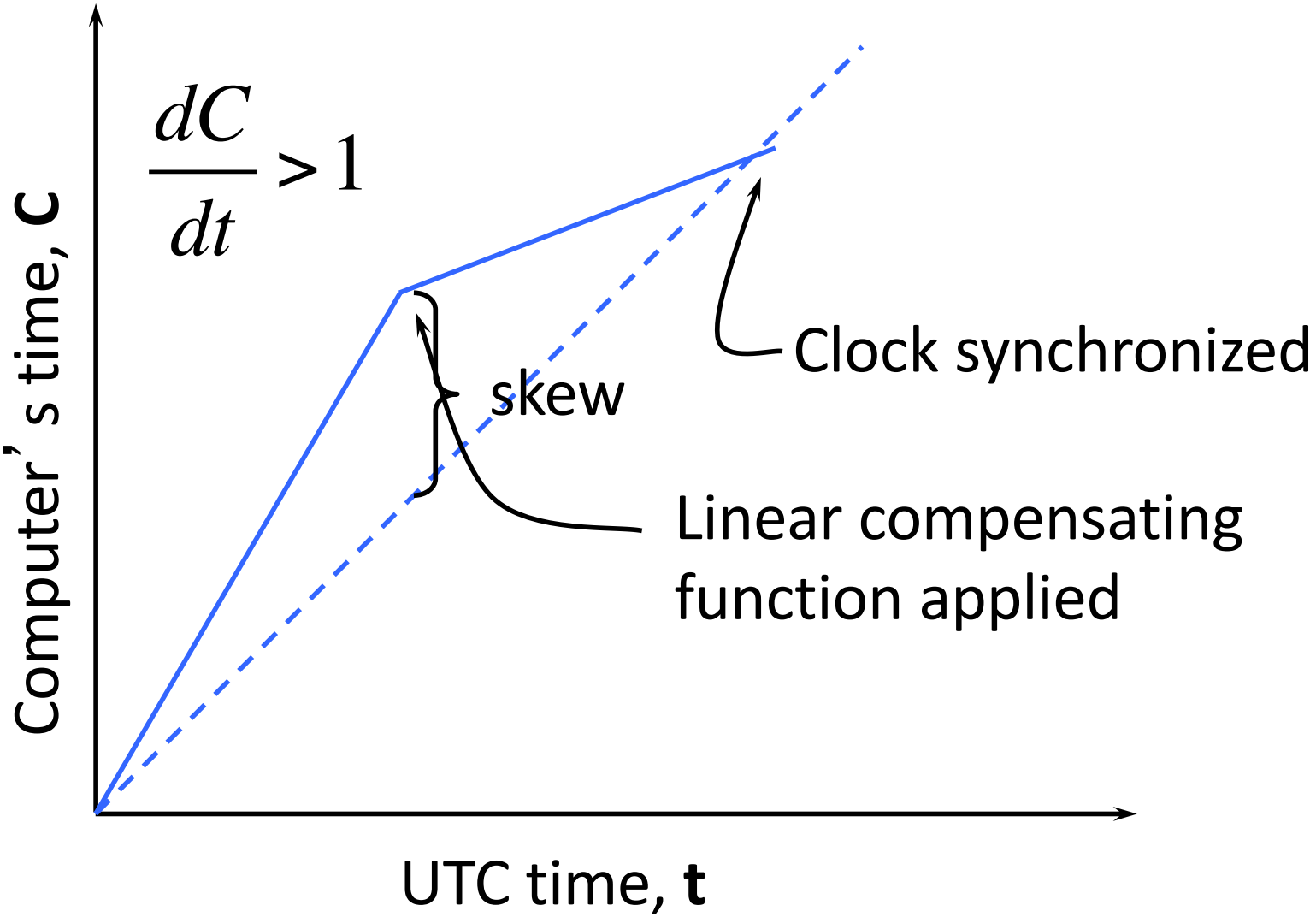
- If we cannot set the clock forward / backward
  - Go for gradual clock correction
- If fast:
  - Make clock run slower until it synchronizes
- If slow:
  - Make clock run faster until it synchronizes

Dealing with Drift

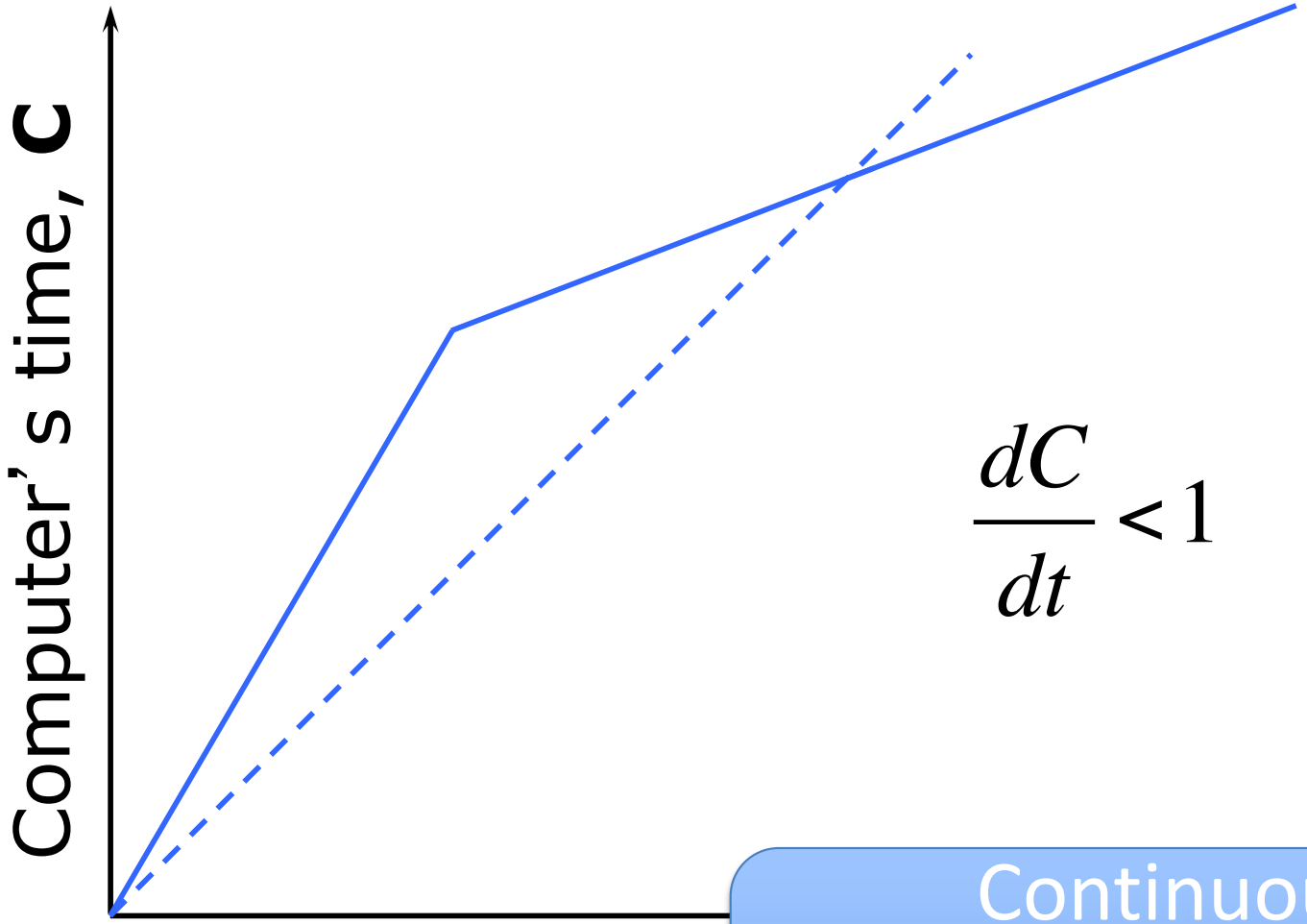
- OS can do this:
  - Change rate at which it requests interrupts
  - e.g.:
    if system requests interrupts every
    17 msec but clock is too slow:
    request interrupts at (e.g.) 15 msec
  - Or software correction: redefine the interval

- Adjustment changes slope of system time:
  - Linear compensating function

Compensating for a Fast Clock



Compensating for a Fast Clock



$$\frac{dC}{dt} < 1$$

Continuous Synchronization needed

Maximum Drift Rate of a Clock

- The manufacturer of the timer specifies the upper and the lower bound that the clock skew may fluctuate. This value is known as maximum drift rate (ρ)

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

- How far can two clocks drift apart?
  - If two clocks were synchronized Δt seconds before to UTC?
    - then the two clocks can be as much as 2ρΔt seconds apart
- Guaranteeing maximum drift between computers in a DS
  - If maximum drift permissible in a DS is X seconds,
    - then clocks of every computer has to resynchronize at least X/2ρ seconds

Synchronization Algorithms

PHYSICAL CLOCKS

Resynchronizing

- After synchronization period is reached
  - Resynchronize periodically
  - Successive application of a second linear compensating function can bring us closer to true slope
- Keep track of adjustments and apply continuously

Getting Accurate Time

- Attach GPS receiver to each computer
  - ± 1 msec of UTC
- Or similar technologies

- Not practical solution for every machine
  - Cost, size, convenience, environment

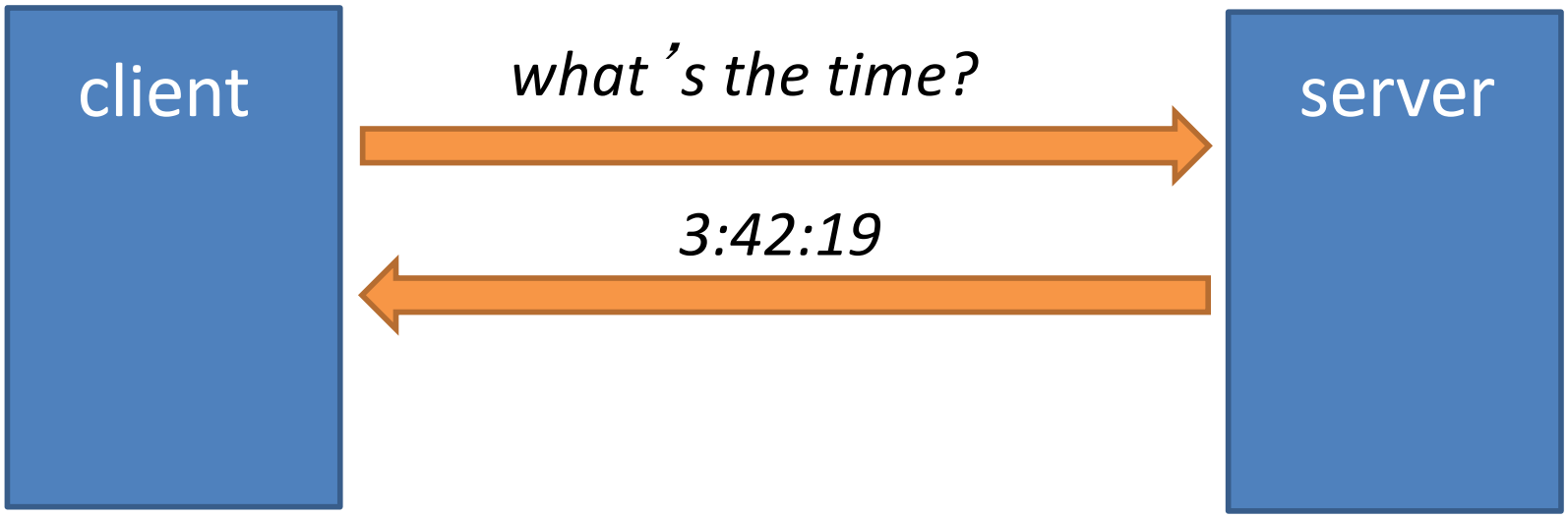
Getting Accurate Time

- Synchronize from another machine
  - One with a more accurate clock
- Machine/service that provides time information:
  - Time server

Query Time Server

- Simplest synchronization technique?
  - Send query to obtain time
  - Set time

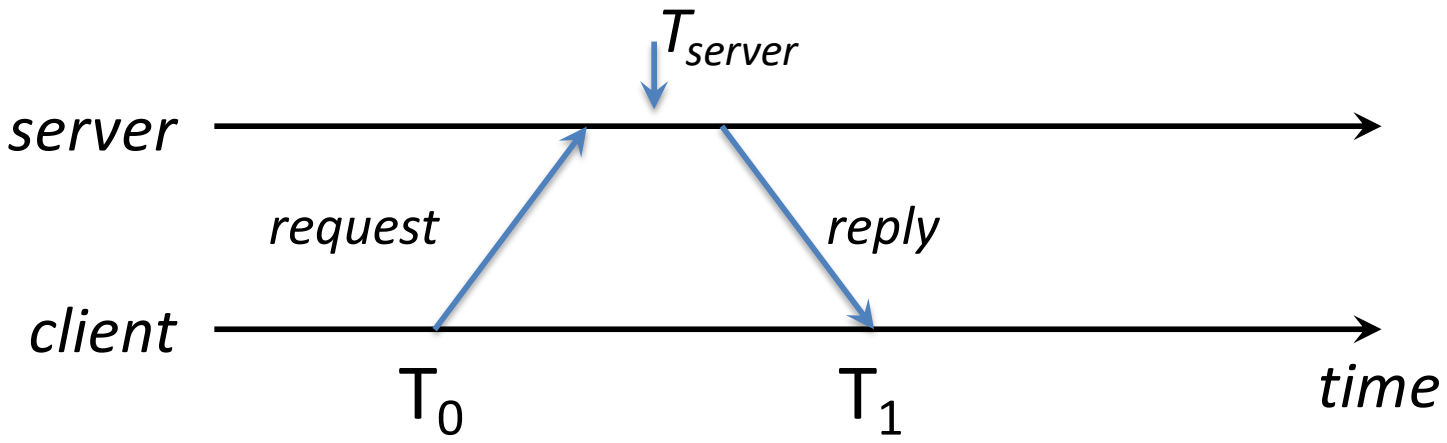
Typical RTTs?
<45ms: within North America.
<30ms: within Europe.
<90ms: London <-> New York.



Problem?
Does not account for network or processing latency

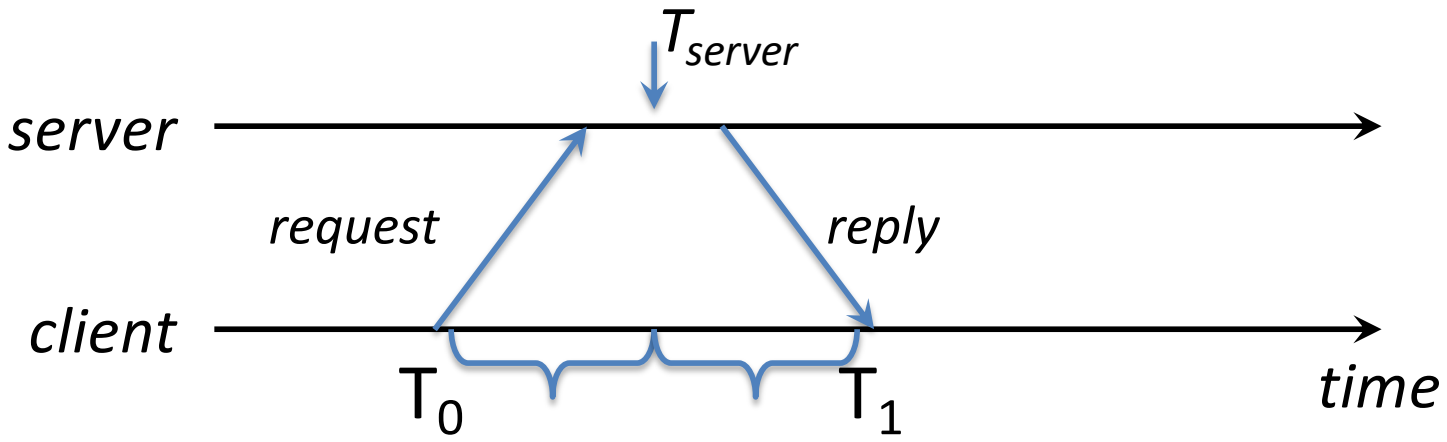
Cristian's Algorithm

- Compensate for delays
  - Note times:
    - Request sent: T0
    - Reply received: T1
  - Assume network delays are symmetric
    - Is this assumption realistic?



Cristian Algorithm

- Client sets time to:



$$d = \frac{T_1 - T_0}{2}$$

d: estimated delay per direction

$$T_{new} = T_{server} + d = T_{server} + \frac{T_1 - T_0}{2}$$

Delay and Asymmetric Delays

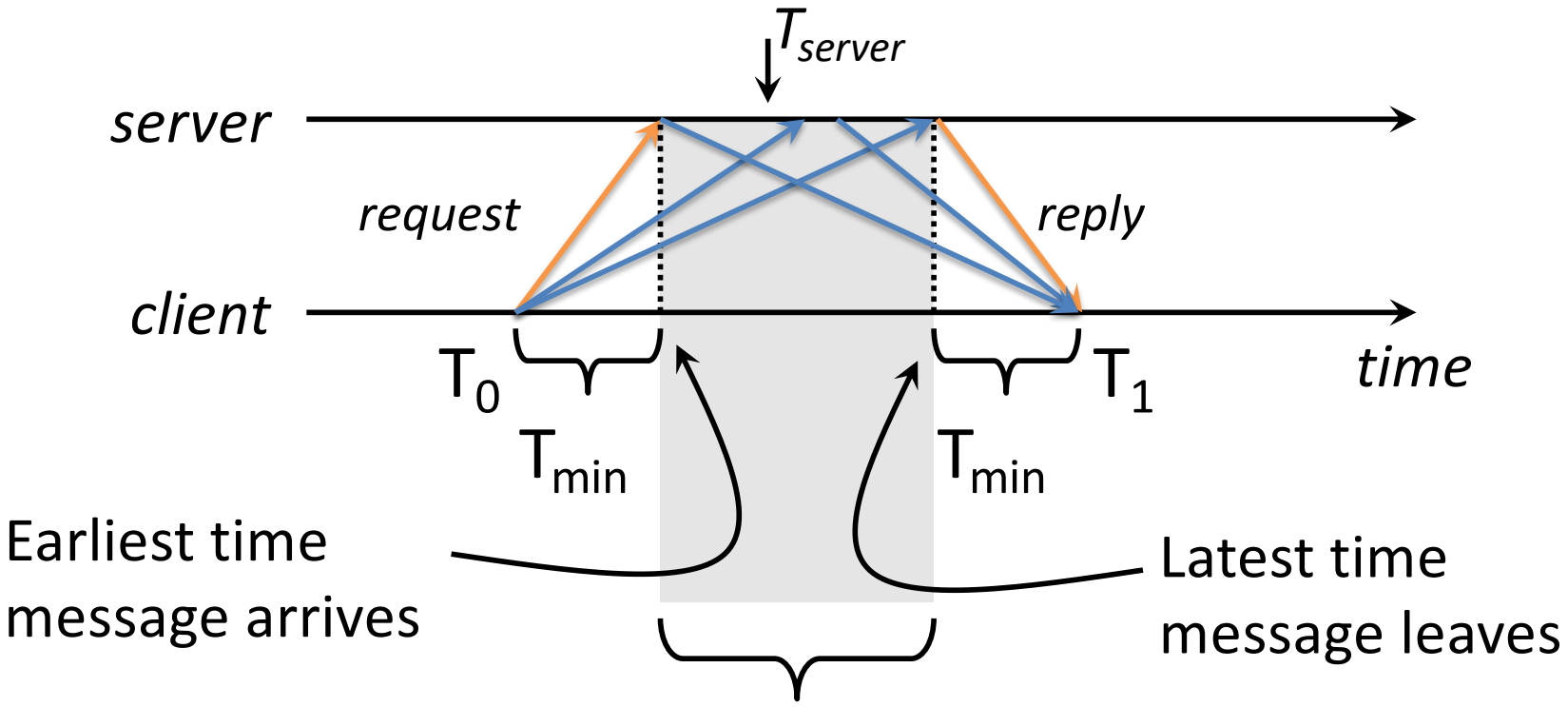
- Network
  - Buffers in routers
  - Medium contention
  - …
- OS processing
  - Interrupts
  - Scheduler
  - …
- Delay is not symmetric
  - But often close

Error Bounds

- Assume:
  - Minimum message transit time (T_{min}) is known
  - Place bounds on accuracy of result

- Determine T_{min}
  - Physical limitations:
    - Example: A packet cannot travel faster than light
  - Protocol specifications
    - Example: Known min. backoff before transmissions
  - …

Error Bounds



$$range = T_1 - T_0 - 2T_{\min}$$

$$accuracy = \pm \frac{T_1 - T_0}{2} - T_{\min}$$

Cristian's Algorithm: Example

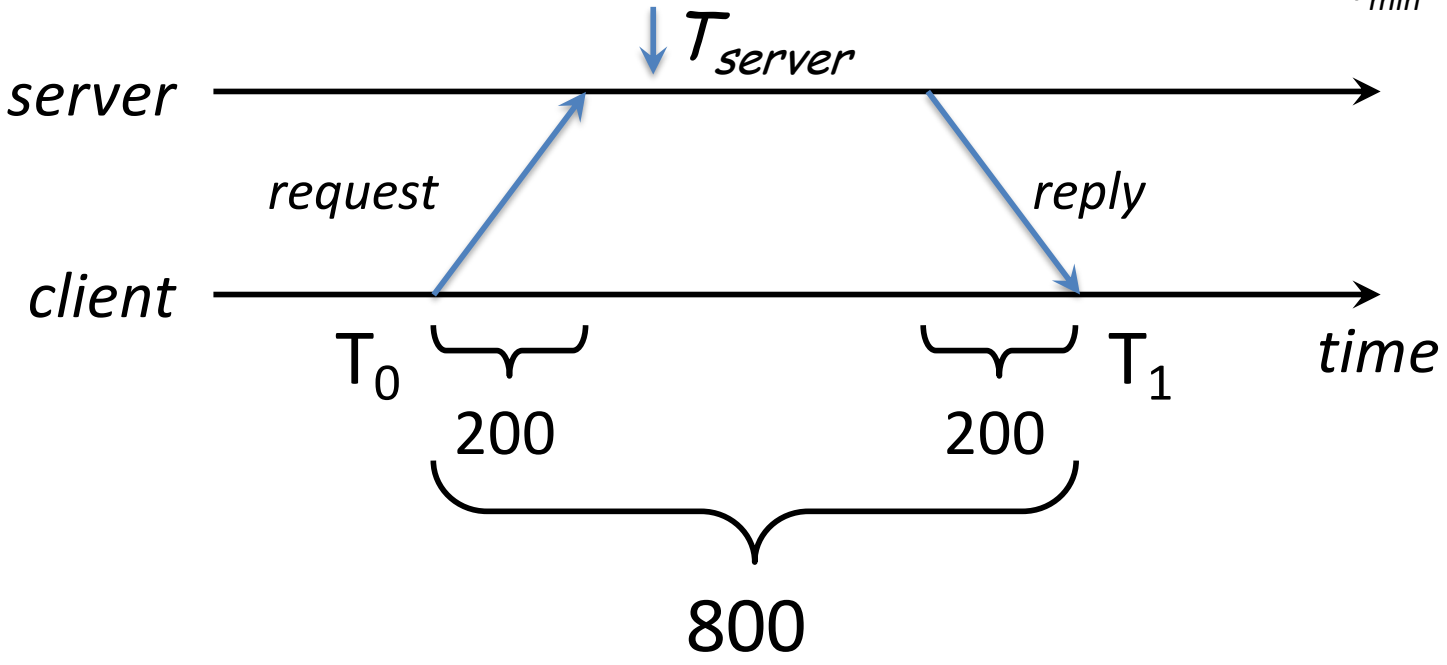
- Send request at 5:08:15.100 (T0)
- Receive response at 5:08:15.900 (T1)
  - Response contains 5:09:25.300 (Tserver)

- Elapsed time is T1 -T0
  - 5:08:15.900 - 5:08:15.100 = 800 msec
- Best guess: timestamp was generated
  - 400 msec ago
- Set time to Tserver+ elapsed time
  - 5:09:25.300 + 400 = 5:09.25.700

Cristian's Algorithm: Example

T_0 = 5:08:15.100
T_1 = 5:08:15.900
T_s = 5:09:25:300
T_{min} = 200msec

- If best-case message time=200 msec



$$accuracy = \pm\frac{900-100}{2} - 200 = \pm\frac{800}{2} - 200 = \pm 200$$

Berkeley Algorithm

- Gusella & Zatti, 1989

- Assumes no machine has an accurate time source
- Obtains average from participating computers
- Synchronizes all clocks to average

Berkeley Algorithm

- Machines run time daemon
  - Process that implements protocol
- One machine is elected (or designated) as the server (master)
  - Leader election ;-)
  - Others are slaves

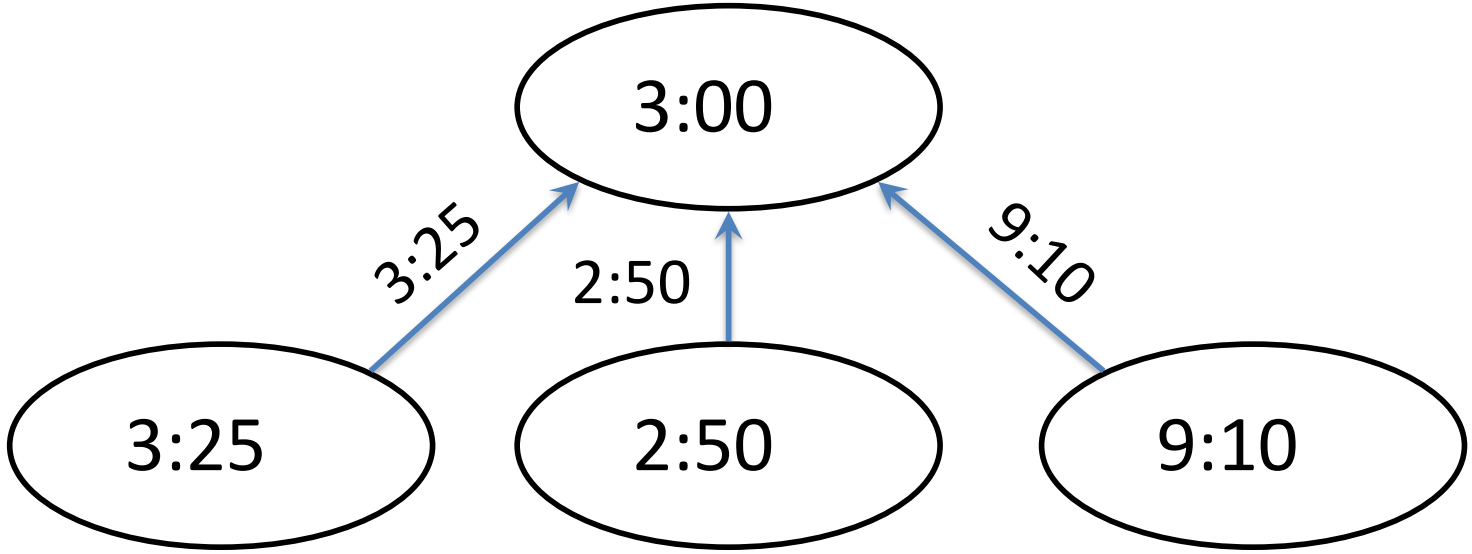
Berkeley Algorithm

- Master polls each machine periodically
  - Ask each machine for time
    - Can use Cristian's algorithm to compensate for network latency
- When results are in, compute average
  - Including master's time
- Hope: average cancels out individual clock's tendencies to run fast or slow
- Send offset by which each clock needs adjustment to each slave

Berkeley Algorithm

- Algorithm has provisions for ignoring readings from clocks whose skew is too great
  - Compute a fault-tolerant average
- If master fails
  - Any slave can take over

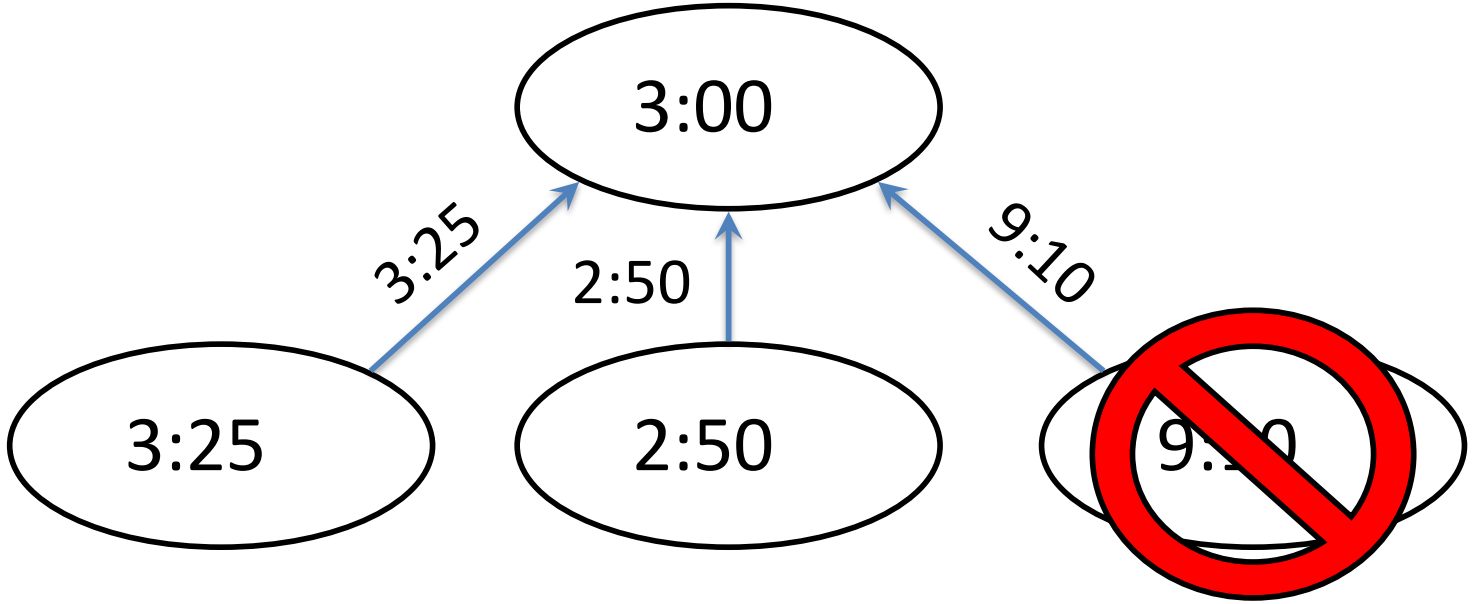
Berkeley Algorithm: Example



1. Request timestamps from all slaves

Alternative:
Master sends our own clock value, slaves reply with delta

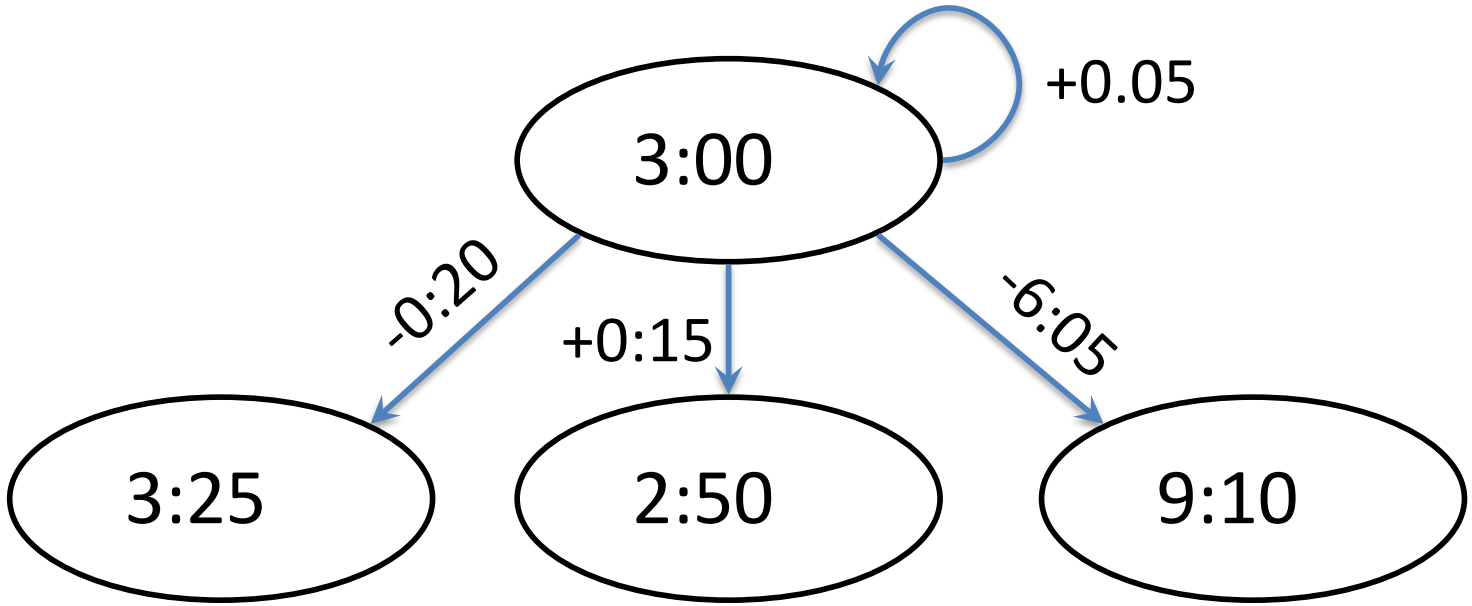
Berkeley Algorithm: Example



2. Compute fault-tolerant average:

$$time = \frac{3:25 + 2:50 + 3:00}{3} = 3:05$$

Berkeley Algorithm: Example



3. Send offset to each client: set to 3:05

Be careful when setting time directly! Why?

Network Time Protocol, NTP

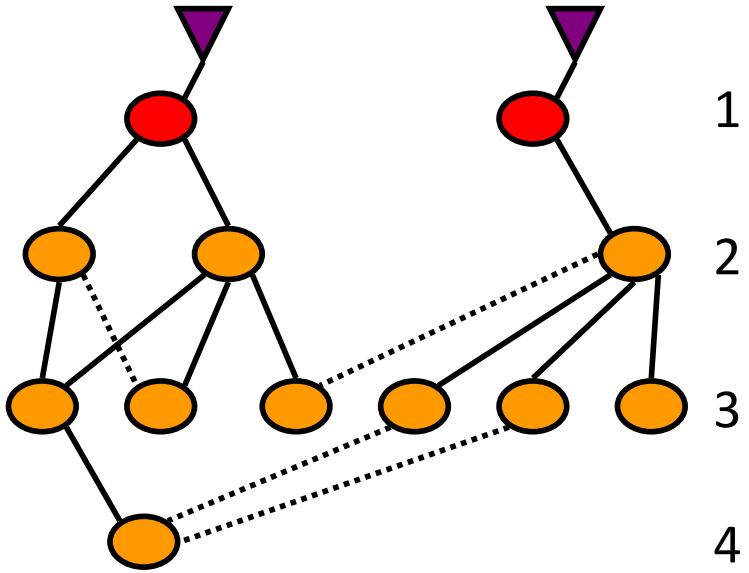
- 1991, 1992
- Internet Standard, version 3: RFC 1305

NTP Goals

- Enable clients across Internet to be accurately synchronized to UTC despite message delays
  - Use statistical techniques to filter data and gauge quality of results
- Provide reliable service
  - Survive lengthy losses of connectivity
  - Redundant paths
  - Redundant servers
- Enable clients to synchronize frequently
  - offset effects of clock drift
- Provide protection against interference
  - Authenticate source of data

NTP servers

- Arranged in strata
  - 1st stratum: machines connected directly to accurate time source
  - 2nd stratum: machines synchronized from 1st stratum machines
  - …



NTP Synchronization Modes

- **Multicast mode**
  - for high speed LANS
  - Lower accuracy but efficient
- **Procedure call mode**
  - Similar to Cristian's algorithm
- **Symmetric mode**
  - Intended for master servers
  - Pair of servers exchange messages and retain data to improve synchronization over time

- All messages delivered unreliably with UDP
  - Why?
  - Streaming, retransmission in TCP add variable delays

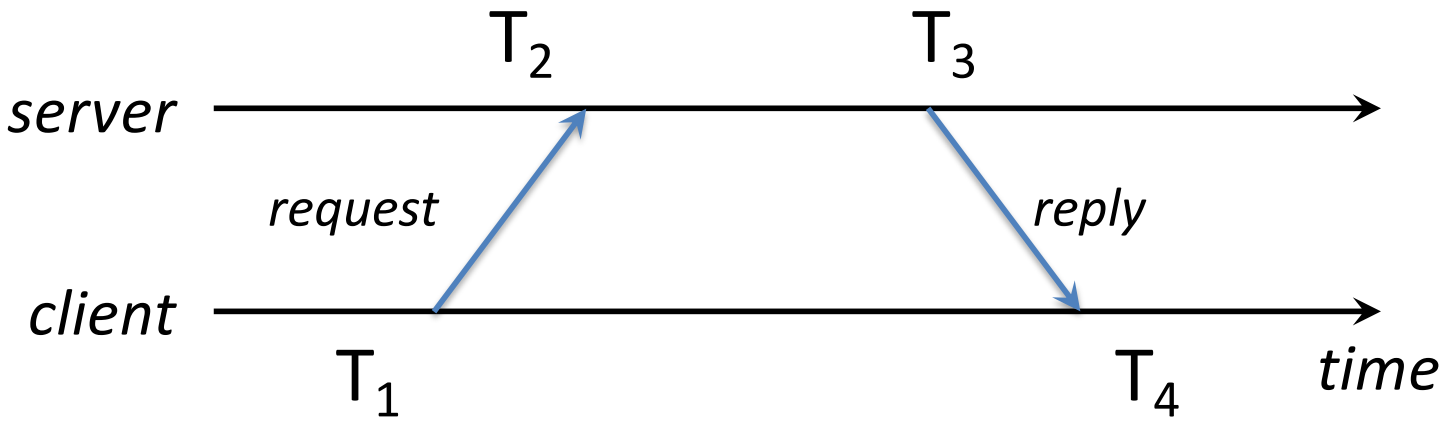
NTP Messages

- Procedure call and symmetric mode
  - Messages exchanged in pairs
- NTP calculates:
  - **Offset** for each pair of messages
    - Estimate of offset between two clocks
  - **Delay**
    - Transmit time between two messages
  - **Filter Dispersion**
    - Estimate of error – quality of results
    - Based on accuracy of server's clock and consistency of network transit time
- Use this data to find preferred server:
  - lower stratum & lowest total dispersion

SNTP

- Simple Network Time Protocol
  - Based on Unicast mode of NTP
  - Subset of NTP, not new protocol
  - Recommended for environments where server is root node and client is leaf of synchronization subnet
  - Root delay, root dispersion, reference timestamp ignored
- RFC 2030, October 1996

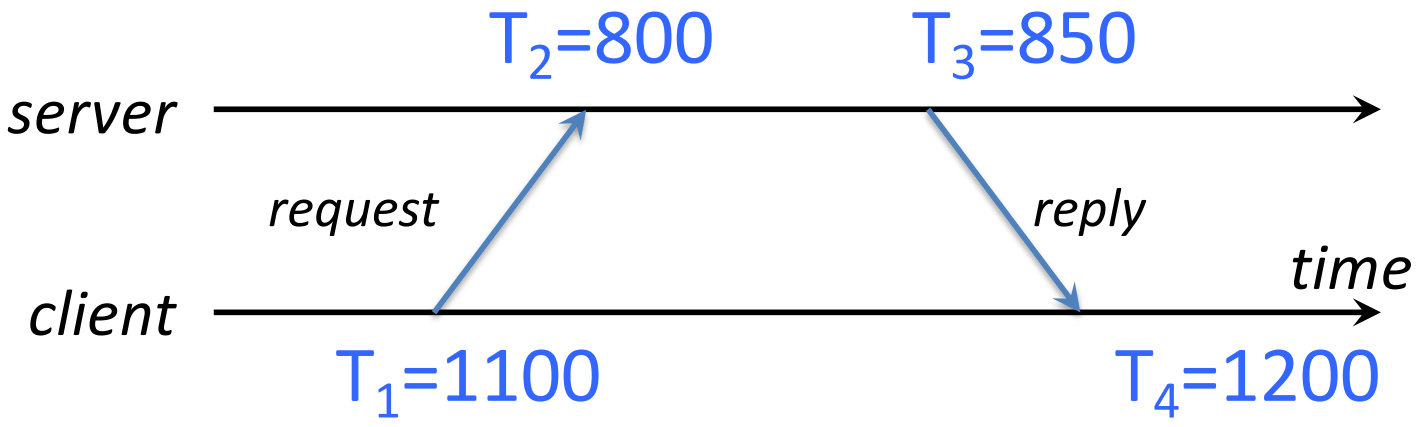
SNTP



$$delay = (T_4 - T_1) - (T_3 - T_2)$$

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

SNTP example



- Offset =
  - ((800 - 1100) + (850 - 1200))/2
  - =((-300) + (-350))/2
  - = -650/2 = -325

- Set time to
  - T4 + t
    = 1200 - 325 = 875

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

Key Points: Physical Clocks

- Cristian's algorithm & SNTP
  - Set clock from server
  - But account for network delays
  - Error:
    - uncertainty due to network/processor latency
  - Adjust for local clock skew
  - Linear compensation function

Your Turn

- Which algorithms do require a server with a valid clock?
- Which algorithms account for transmission and processing delay?
- Which algorithms can deal with a faulty clock?
- What is dangerous for a distributed system?

http://olafland.polldaddy.com/s/physicalclocks

Solutions

- Which algorithms do require a server with a valid clock?
  - SNTP, Christian's Alg
- Which algorithms account for transmission and processing delay?
  - SNTP, Christian's Alg
- Which algorithms can deal with a faulty clock?
  - Berkeley
- What is dangerous for a distributed system?
  - Travel backwards and forwards in time

Next Time

- Logical clocks

Questions?

In part, inspired from / based on slides from

- Paul Krzyzanowski
- Vinay Kolar