# Distributed Systems
# 分布式系统

Distributed Time and Clock Synchronization (1)

Physical Time

物理时间

# Why Timestamps in Systems?

- Precise performance measurements
- Guarantee “up-to-date” or recentness of data
- Temporal ordering of events produced by concurrent processes
- Synchronization between senders and receivers of messages
- Coordination of joint activities
- Serialization of concurrent accesses to shared objects
- ……

# Physical time

## Solar time

- 1 sec = 1 day / 86400
- Problem: days are of different lengths (due to tidal friction, etc.)
- mean solar second: averaged over many days

## Greenwich Mean Time (GMT)

- The mean solar time at Royal Observatory in Greenwich, London
- Greenwich located at longitude 0, the line that divides east and west

# Coordinated Universal Time (UTC)

## International atomic time (TAI) 国际原子时间

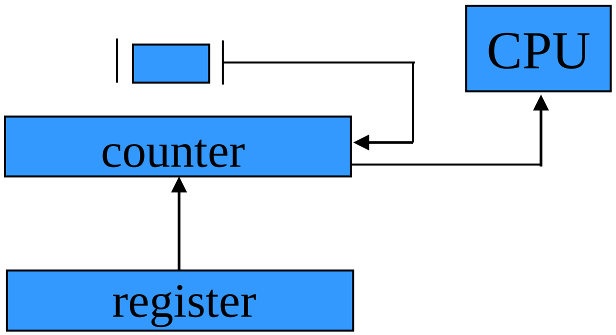
- 1 sec ≡ time for Cesium-133 atom to make 9,192,631,770 state transitions.
- TAI time is simply the number of Cesium-133 transitions since midnight on Jan 1, 1958.
- Accuracy: better than 1 second in six million years
- Problem: Atomic clocks do not keep in step with solar time

## Coordinated Universal Time (UTC) 通用协调时间

- Based on the atomic time (TAI) and introduced from 1 Jan 1972
- A leap second is occasionally inserted or deleted to keep in step with solar time when the difference btw a solar-day and a TAI-day is over 800ms

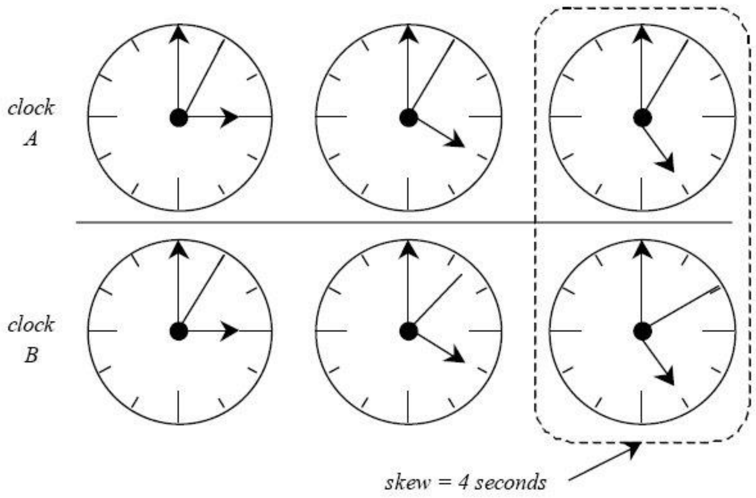
# Computer Clocks

- CMOS clock circuit driven by a quartz oscillator
  - battery backup to continue measuring time when power is off
- The circuit has a counter（计数器） and a register（寄存器）. The counter decrements by 1 for each oscillation; an interrupt（中断） is generated when it reaches 0 and the number in the register is loaded to the counter. Then, it repeats…
- OS catches interrupt signals to maintain a computer clock
  - e.g., 60 or 100 interrupts per second
  - Programmable Interrupt Controller (PIC)
  - Interrupt service routine increments the “clock” by 1 for each interrupt

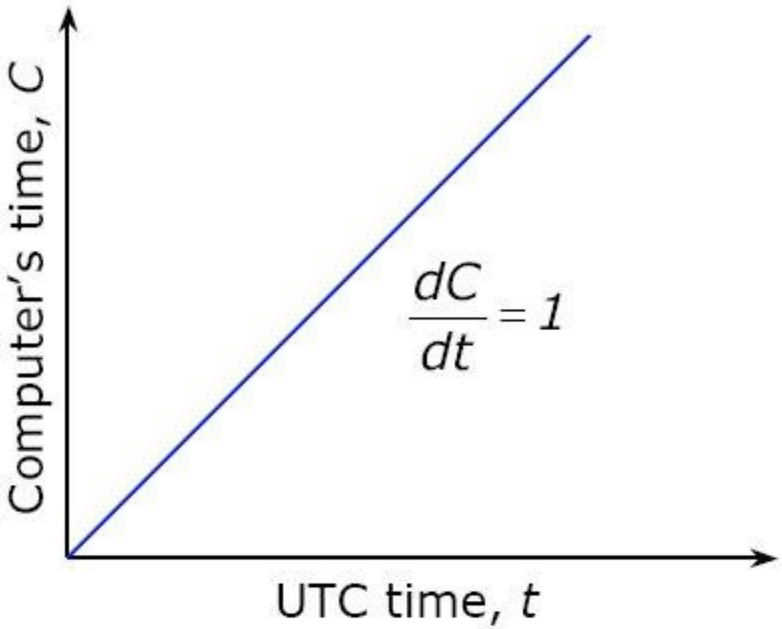


# Clock drift and clock skew

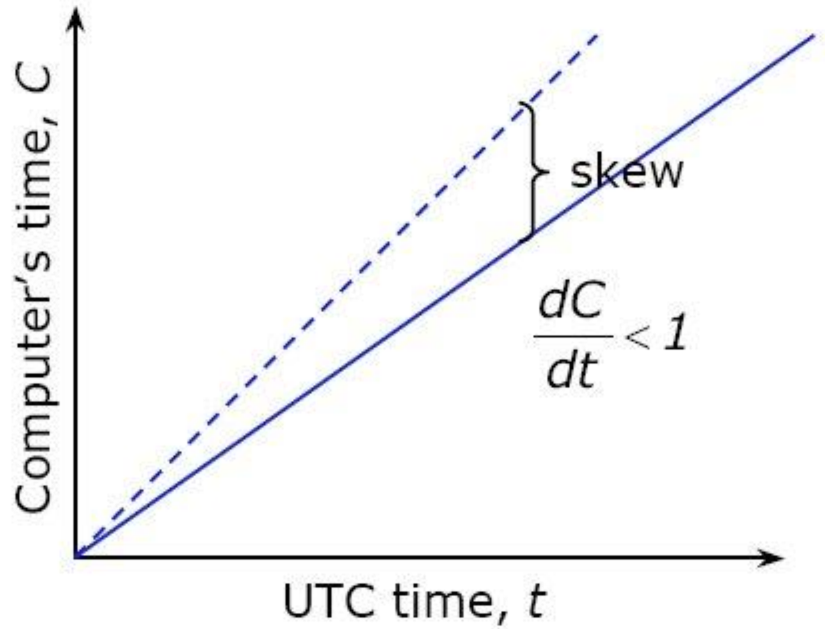
- Clock Drift 时钟漂移
  - Clocks tick at different rates
    - Ordinary quartz clocks drift by ~ 1sec in 11-12 days. ($10^{-6}$ secs/sec).
    - High precision quartz clocks drift rate is ~ $10^{-7}$ or $10^{-8}$ secs/sec
  - Create ever-widening gap in perceived time
- Clock Skew (offset) 时钟偏移
  - Difference between two clocks at one point in time



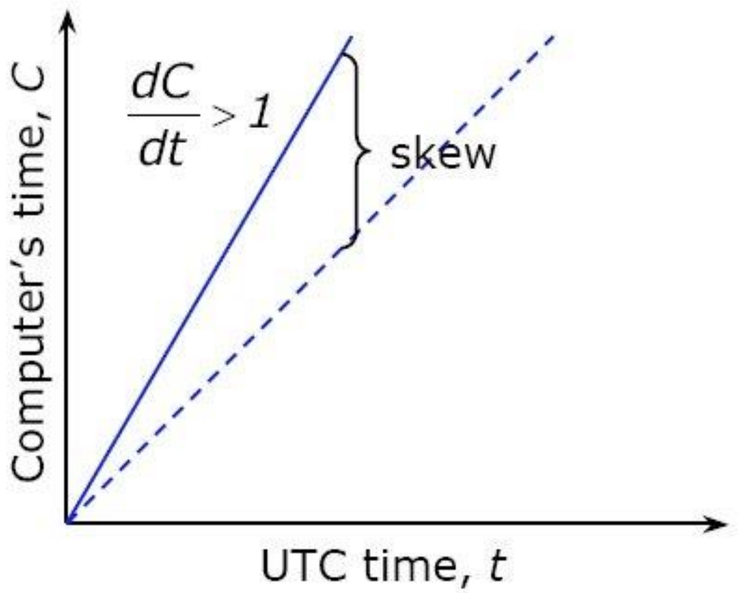
# Perfect clock



# Drift with a slow computer clock



# Drift with a fast computer clock



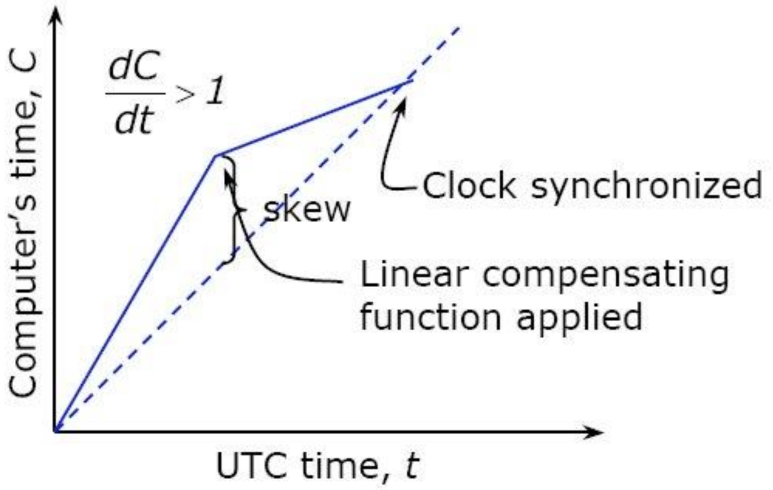
# Dealing with drift

- No good to set a clock *backward*
  - Illusion of time moving backwards can confuse message ordering and software development environments

- Go for *gradual* clock correction
  - If fast: Make clock run slower until it synchronizes
  - If slow: Make clock run faster until it synchronizes

# Linear compensating function

- OS can do this: Change the frequency of clock interrupts
  - e.g.: if the system generates an interrupt every 17 ms but clock is too slow: generates an interrupt at (e.g.) 15 ms
- Adjustment changes slope of system time: **Linear compensating function**（线性补偿函数）



# Resynchronization

- After synchronization period is reached
  - Resynchronize periodically, or
  - The skew is beyond a threshold

- Keep track of adjustments and apply continuously
  - UNIX *adjtime* system call:

```
int adjtime(struct  timeval  *delta,  struct  timeval  *old-delta)
```

  - adjusts the system's notion of the current time, advancing or retarding it, by the amount of time specified in the struct timeval pointed to by delta. “old-delta”, output parameter, returns time left uncorrected since last call of “adjtime”

# Getting UTC from Top Sources

- Attach GPS receiver to each computer
  - $\pm$ 1 ms of UTC
- Attach WWV (http://tf.nist.gov) radio receiver
  - Obtain time broadcasts from Boulder or DC
  - $\pm$ 3 ms of UTC (depending on distance)
- Attach GOES receiver (Geostationary Operational Environmental Satellites, http://www.goes.noaa.gov/)
  - $\pm$ 0.1 ms of UTC

Not practical for every machine

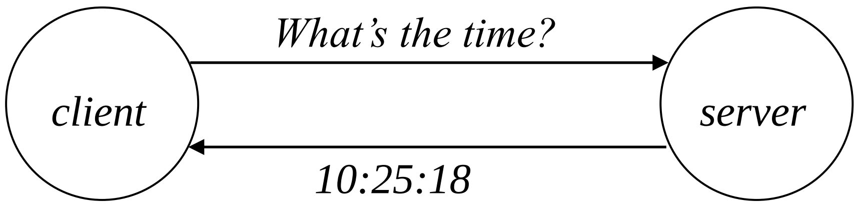
– Cost, size, convenience, environment

# Getting UTC for Client Computers

- Synchronize clock of a client to a time server that
  - with a more accurate clock, or
  - connected to UTC time source
- Also called external clock synchronization

# Synchronizing Clocks by using RPC

- Simplest synchronization technique
  - Make an RPC to obtain time from the server
  - Set the local clock to the server time

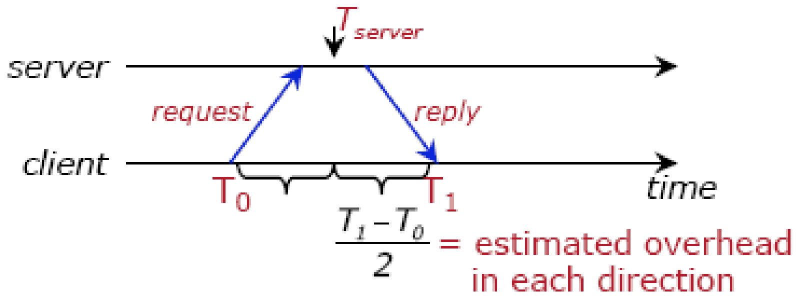



Does not count network or processing latency

# Cristian's algorithm

Compensate for network delays (assuming symmetric)

- client sends a request at $T_0$
- server replies with the current clock value $T_{server}$
- client receives response at $T_1$
- client sets its clock to: $T_{client} = T_{server} + \frac{T_1 - T_0}{2}$

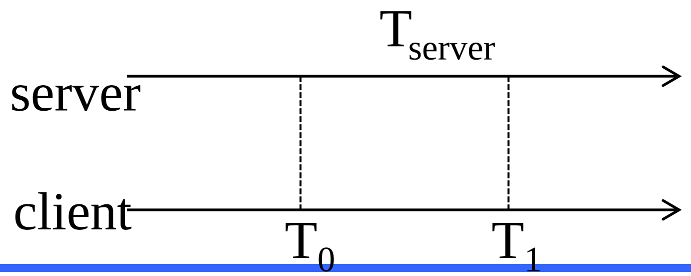


# Cristian's algorithm: example

- Send request at 5:08:15.100 ($T_0$)
- Receive response at 5:08:15.900 ($T_1$)
  - Response contains 5:09:25.300 ($T_{server}$)
- Round-trip time is $T_1 - T_0$

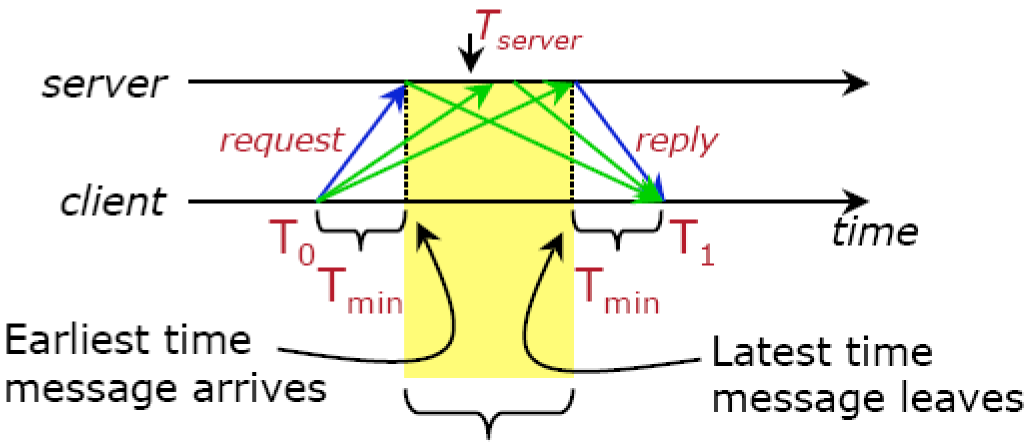
  5:08:15.900 - 5:08:15.100 = 800 ms
- Best guess: timestamp was generated 400 ms ago
- Set the local time to $T_{server}$ + *round-trip-time/2*

  5:09:25.300 + 400 = 5:09.25.700
- Accuracy: $\pm$ *round-trip-time/2*



# Cristian's algorithm: error bound

*$T_{min}$: Minimum message travel time*




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

$$\boxed{\textbf{accuracy of result} = \pm\left(\frac{T_1 - T_0}{2} - T_{min}\right)}$$

# Problems with Cristian's algorithm

- Server might fail
- Subject to malicious interference

# Berkeley Algorithm

- Proposed by Gusella & Zatti, 1989 and implemented in BSD version of UNIX
- Aim: synchronize clocks of a group of machines as close as possible (also called internal synchronization)
- Assumes no machine has an accurate time source (i.e., no differentiation of client and server)
- Obtains average from participating computers
- Synchronizes all clocks to average

# Berkeley Algorithm

One machine is elected (or designated) as the **master**; others are **slaves**:

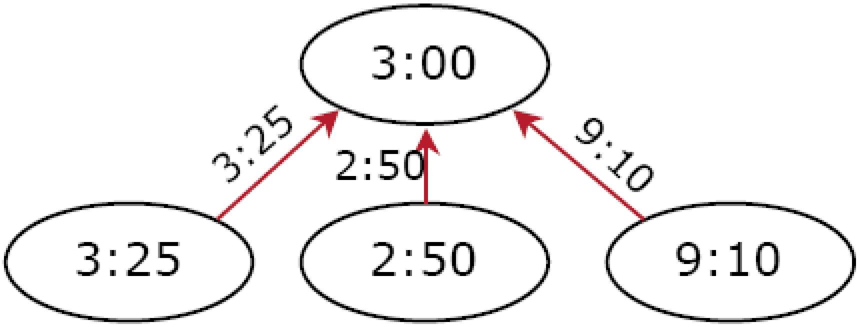
1. Master polls all slaves periodically, asking for their time
   - Cristian's algorithm can be used to obtain more accurate clock values from other machines by counting network latency
2. When results are collected, compute the average
   - Including master's time
3. Send each slave the offset that its clock need be adjusted
   - Avoids problems with network delays by sending "offset" instead of "timestamp"

# Berkeley Algorithm

- Algorithm has provisions for ignoring readings from clocks whose skew is too large
  - Compute a **fault-tolerant average**

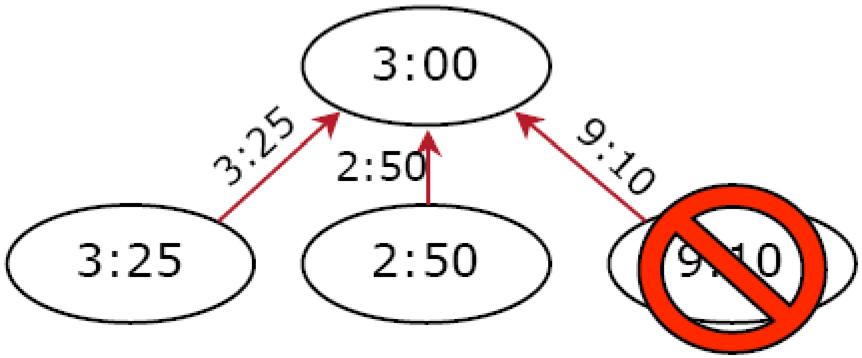
- Any slave can take over the master if master fails

# Berkeley Algorithm: example




1. Request timestamps from all slaves

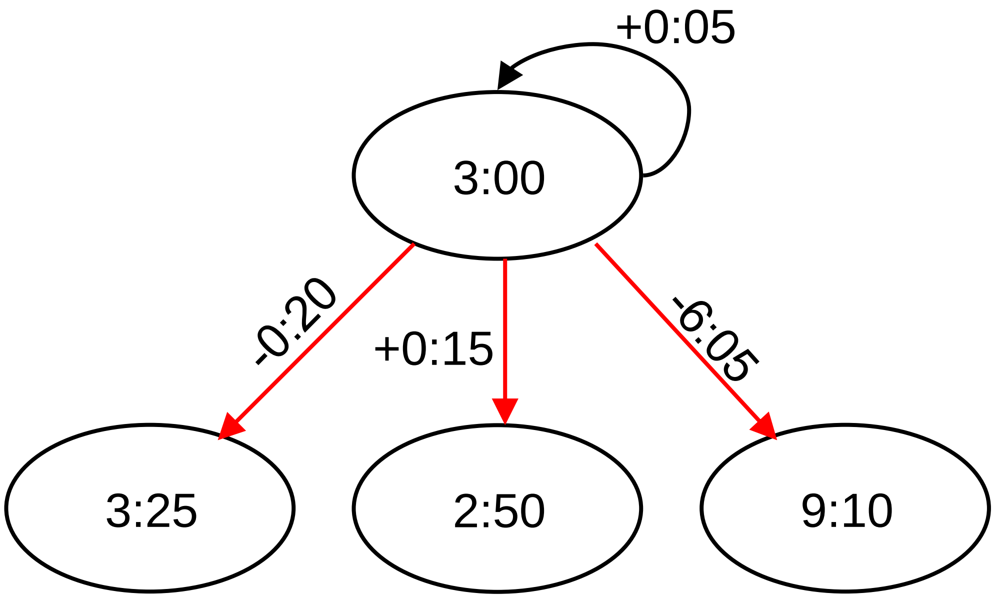
# Berkeley Algorithm: example




2. Compute fault-tolerant average:

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

# Berkeley Algorithm: example

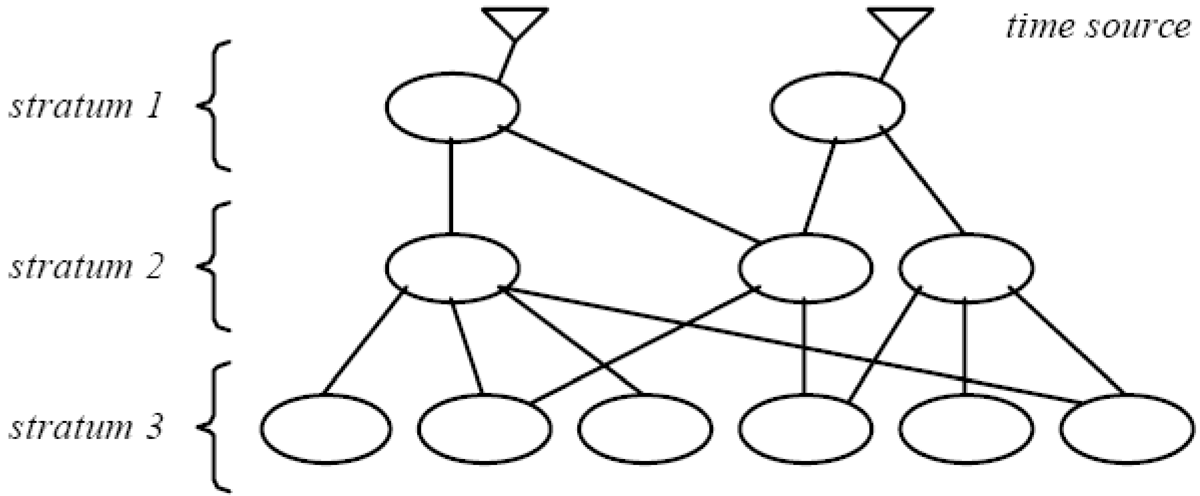



3. Send offset to each client

# Network Time Protocol (NTP)

- NTP is the most commonly used Internet time protocol and the one provides best accuracy (RFC 1305, http://tf.nist.gov/service/its.htm ).
- Computers often include NTP software in OS. The client software periodically gets updates from one or more servers (average them).
- Time servers listen to NTP requests on port 123, and reply a UDP/IP data packet in NTP format, which is a 64-bit timestamp in UTC seconds since Jan 1, 1900 with a resolution of 200 pico-s.
- Many NTP client software for PC only gets time from a single server (no averaging). The client is called SNTP (Simple Network Time Protocol, RFC 2030), a simple version of NTP.

# NTP synchronization subnet




1st stratum: machines connected directly to accurate time source

2nd stratum: machines synchronized from 1st stratum machines

…

# NTP goals

- Enable clients across Internet to be accurately synchronized to UTC despite message delays
  - Use statistical techniques to filter data and improve quality of results
- Provide reliable service
  - Survive lengthy losses of connectivity
  - Redundant paths
  - Redundant servers
- Enable clients to synchronize frequently
  - Adjustment of clocks by using offset (for symmetric mode)
- Provide protection against interference
  - Authenticate source of data

# NTP Synchronization Modes

- Multicast (for quick LANs, low accuracy)
  - server periodically multicasts its time to its clients in the subnet

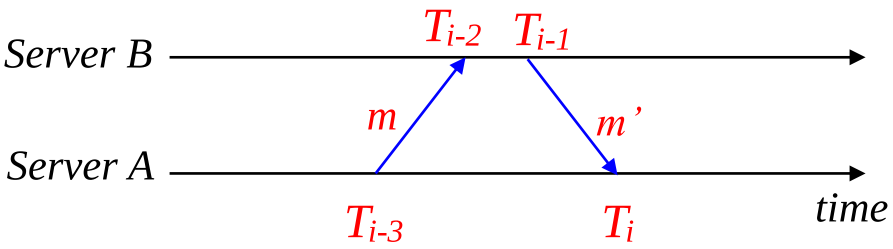
- Remote Procedure Call (medium accuracy)
  - server responds to client requests with its actual timestamp
  - like Cristian's algorithm

- Symmetric mode (high accuracy)
  - used to synchronize between the time servers (peer-peer)

*All messages delivered unreliably with UDP*

# Symmetric mode

- The delay between the arrival of a request (at server B) and the dispatch of the reply is NOT negligible:

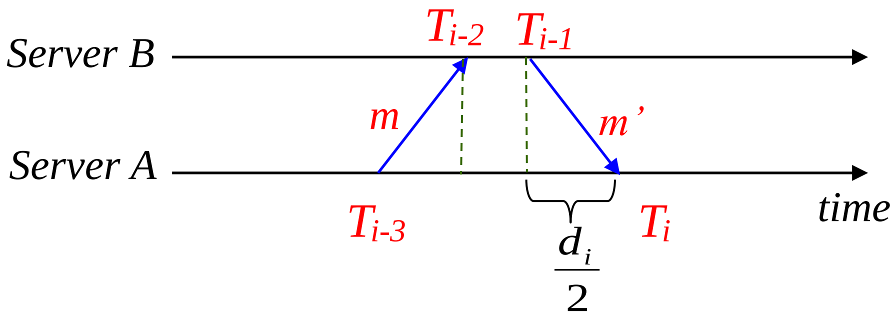



- Delay = total transmission time of the two messages

$$d_i = (T_i - T_{i-3}) - (T_{i-1} - T_{i-2})$$

- Offset of clock *A* relative to clock *B*:
  - Offset of clock *A*: $o_i = \dfrac{(T_{i-2} - T_{i-3}) + (T_{i-1} - T_i)}{2}$
  - Set clock *A*: $T_i + o_i$
  - Accuracy bound: $d_i / 2$

# Symmetric mode (another expression)



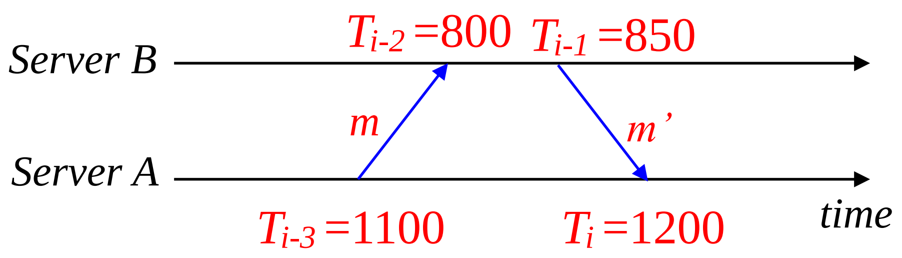

- Delay = total transmission time of the two messages

  $d_i = (T_i - T_{i-3}) - (T_{i-1} - T_{i-2})$

- Clock *A* should set its time to (the best estimate of B's time at $T_i$):

  $T_{i-1} + d_i/2$, which is the same as $T_i + o_i$

# Symmetric NTP example




假设B是正确的

*Offset* $o_i$=((800 – 1100) + (850 – 1200))/2 = – 325

Set clock *A* to: $T_i + o_i$ = 1200 – 325 = 875

Note: Server A need to adjusts it current clock (1200ms) by gradual slowdown its pace until -325ms is compensated.

# Improving accuracy

- Data filtering from a single source
  - Retain the multiple most recent pairs < $o_i$, $d_i$ >
  - Filter dispersion: choose $o_j$ corresponding to the smallest $d_j$
- Peer-selection: synchronize with lower stratum servers
  - lower stratum numbers, lower synchronization dispersion
- The stratum of a server is dynamically changing, depending on which server it synchronize with

# Simple Network Time Protocol (SNTP) RFC 2030

- Targeted for machines that have no need of full NTP implementation, particularly for machines at the end of synchronization subnet (client nodes)
- SNTP operate in one of the following modes:
  - Unicast mode, the client sends a request to a designated server
  - Multicast mode, the server periodically broadcast/multicast its time to the subnet and does not serve any requests from clients
  - Anycast mode, the client broadcast/multicast a request to the local subnet and takes the first response for time synchronization