28|09|23

Dependable Distributed Systems Master of Science in Engineering in Computer Science

AA 2023/2024

LECTURE 3: TIME IN DISTRIBUTED SYSTEMS

PHYSICAL CLOCKS AND CLOCK SYNCHRONIZATION

Introduction

In a Distributed System

- Processes run on different nodes interconnected by mean of a network (LAN or WAN)
- Processes cooperate to complete a computation
- Processes communicate only through message-based communications

Many applications require ordering between events and synchronization to terminate correctly

- E.g. Ait traffic control, Network monitoring, measurement and control, Stock market, buy and sell orders, etc…

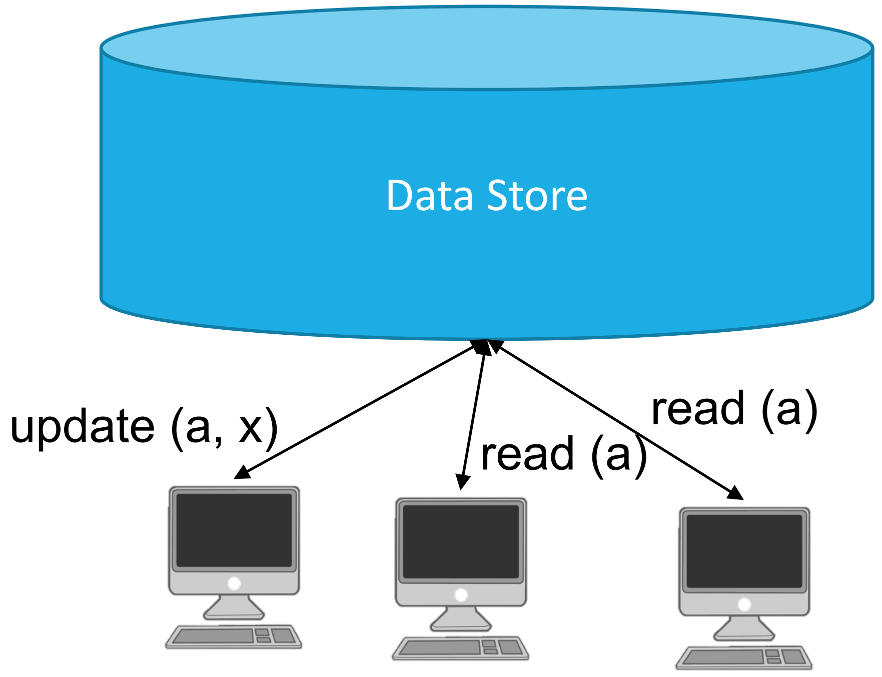
TIME IS A CRITICAL FACTOR FOR DISTRIBUTED SYSTEMS!

Why Time is so Important?

It is a quantity we are interested to measure

A lot of algorithms depends on time

- data consistency
- authentication
- double processing avoidance



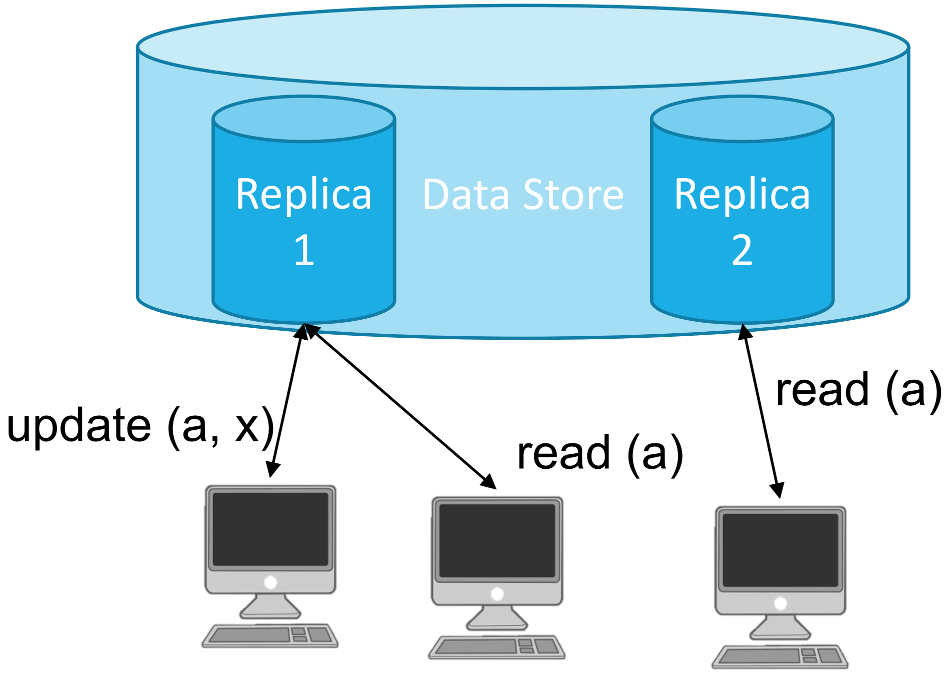
Why Time is so Important?

It is a quantity we are interested to measure

A lot of algorithms depends on time

- data consistency
- authentication
- double processing avoidance

Ordering events is fundamental to solve these problems



Why using time in a DS is difficult?

We need to agree on a common time reference

System Model

A distributed system is composed by a set $\Pi=\{p_1, p_2, \ldots p_n\}$ of *n* processes

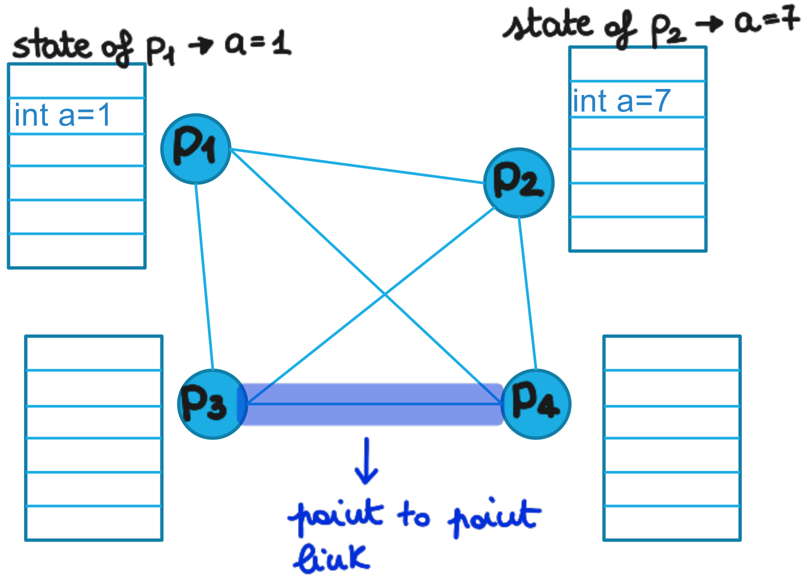
Each process runs on a single mono-processor machine with no shared memory

they can't use the same memory

Each process p_i has a state s_i that is changed by the actions it takes during the algorithm execution

- The state s_i includes all the values of variables maintained by p_i

Each process can communicate with other processes only by $\boxed{\text{exchanging messages}}$



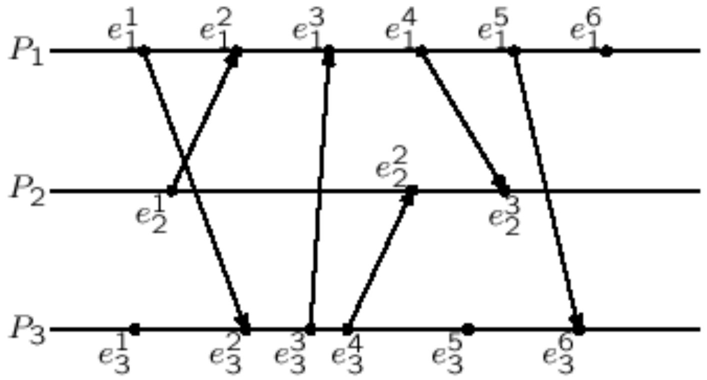
state of $p_1 \rightarrow a=1$
int a=1
state of $p_2 \rightarrow a=7$
int a=7
P1
P2
P3
P4
point to point link

Computation Model

Each process generates a sequence of events

- Internal event (event that transforms the process state)
- external event (send/receive)
- e_i^k, k-th event generated by P_i

The evolution of the computation can be represented with a space-time diagram.



P_1
P_2
P_3
e_1^1 e_1^2 e_1^3 e_1^4 e_1^5 e_1^6
e_2^1 e_2^2 e_2^3
e_3^1 e_3^2 e_3^3 e_3^4 e_3^5 e_3^6

events generated when I receive or send a message

History of Computations

causality relationship

We denote with $\rightarrow_i$ the ordering relationship between two events in the same processes p_i:

$e \rightarrow_i e'$ if and only if e is happened before e' in p_i

Local History

Sequence of events produced by a process

history(p_1) = $\mathbf{h_1 = <e_1^1, e_1^2, e_1^3, e_1^4, e_1^5, e_1^6>}$

Partial Local History

Prefix of local history

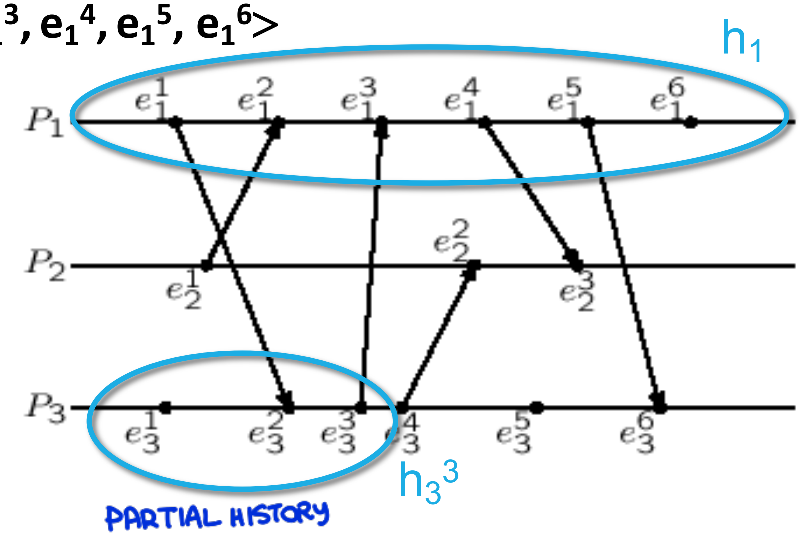
$\mathbf{h_1^m = e_1^1 \ldots e_1^m}$

Global History

Set containing every local history

$\mathbf{H} = \cup_i\ \mathbf{h_i}$ per $1 \leq i \leq n$

union of local execution history


h1
P1
P2
P3
e1 1, e1 2, e1 3, e1 4, e1 5, e1 6
e2 1, e2 2, e2 3
e3 1, e3 2, e3 3, e3 4, e3 5, e3 6
h3 3
PARTIAL HISTORY

Time in Distributed Systems (1/3)

Timestamping

- Each process attaches a label to each event (using a timestamp). In this way it should be possible to realize a global history of the system.

"Naïf" solution

- Each process timestamps events by mean of its physical clock

Time in Distributed Systems (2/3)

But… Is timestamping really a solution?

- It is always possible to define an order among events produced by the same process!!!
- But what’s happen when we consider several distinct processes running on different PCs?

In a distributed system in presence of network delay, processing delay, etc… it is impossible to realize a common clock shared among every process.

Time in Distributed Systems (3/3)

Using timestamps it is possible to synchronize physical clocks (with a certain degree of approximation), through appropriate algorithms.

A process can label events using its physical local clock (synchronized with a certain *synchronization accuracy* or *synchronization error*).

↳ bound over the error that we can have

Physical Computer Clocks

Application processes access a local clock obtained by operating system reading a local hardware clock.

- Hardware clocks consist of an oscillator and a counting register that is incremented at every tick of the oscillator.

At real time t, the operating system reads the hardware clock $H_i(t)$, therefore it produces the software clock

$\downarrow$ depends on the HARDWARE: quality HIGH or LOW level

$$C_i(t) = \alpha H_i(t) + \beta$$

Physical Computer Clocks

$C_i(t)$ approximates the physical time t at process p_i.

- Example: $C_i(t)$ may be implemented by a 64-bit word, representing nanoseconds that have elapsed at time t.

Generally this clock is not completely accurate

- it can be different from the real time t
- It can be different at any process due to the precision of the approximation

C_i can be used such as timestamp for event produced by p_i.

But… How much should be smaller the granularity (resolution) of software clocks (the time interval between two consecutive increments of software clock) to distinguish between two different events?

$T_{resolution} < \Delta T$ between two notable events

Parameters affecting the Clock Synchronization accuracy

Different local clocks can have different values:

- Skew: “the difference in time between two clocks” |Ci(t) – Cj(t)| (Galli)

- Drift Rate: “the gradual misalignment of once synchronized clocks caused by the slight inaccuracies of the time-keeping mechanisms” (difference for time unit with respect to an ideal clock)

  e.g. drift rate of 2microsec/sec means clock increases its value of 1sec+2microsec for each second.

  - Ordinary quartz clocks deviate nearly by 1 sec in 11-12 days. (10^{-6} secs/sec).
  - High-precision quartz clock drift rate is 10^{-7}-10^{-8} secs/sec

Universal Time Coordinated (UTC)

UTC is an international standard: the base of any international measure.

Based on International Atomic Time: 1 sec = time a cesium atom needs for 9192631770 state transitions.

- Pyhisical clocks based on atomic oscillators are the most accurate clocks (drift rate 10^{-13})

Internal/External Synchronization

External Synchronization

- Processes synchronize their clock C_i with an authoritative external source S
- Let D>0 be the synchronization bound and S be the source of UTC
- Clocks C_i (for i = 1, 2, … N) are **externally synchronized** with a time source S (UTC) if for each time interval I:
  - $\underset{\text{UTC}}{\underset{\uparrow}{|S(t)}} - \underset{\text{own time}}{\underset{\uparrow}{C_i(t)}}| < D$ for i = 1, 2, … N and for real time t in I

We say that clocks C_i are accurate within the bound of D

Internal/External Synchronization

Internal Synchronization

- All the processes synchronize their clocks C_i between them
- Let $D>0$ be the synchronization bound and let C_i and C_j the clocks at processes p_i and p_j respectively

- Clocks are **internally synchronized** in a time interval I:
- $\left| \underset{\overset{\uparrow}{p_i}}{C_i}(t) - \underset{\overset{\uparrow}{p_j}}{C_j}(t) \right| < D$ for i,j = 1, 2, … N and for all time t in I

We say that clocks C_i, C_j *agree* within the bound of D

Physical Clock Synchronization

Notes:

1) Note that clocks that are internally synchronized are not necessarily externally synchronized. i.e. even thought they agree with each other, they drift collectively from the external time source.

On other hand, a set of processes *P*, externally synchronized within the bound of D, is also internally synchronized within the bound of 2D.

- This property directly follows from the definition of internal and external clock synchronization.

$D = 2\,m$

●	●
UTC	p_1
10:00	10:01

$\rightarrow$ EXTERNAL SYNCHRONIZED

$$[(10:00)-(10:01)] < D$$

Correct Clock (1/2)

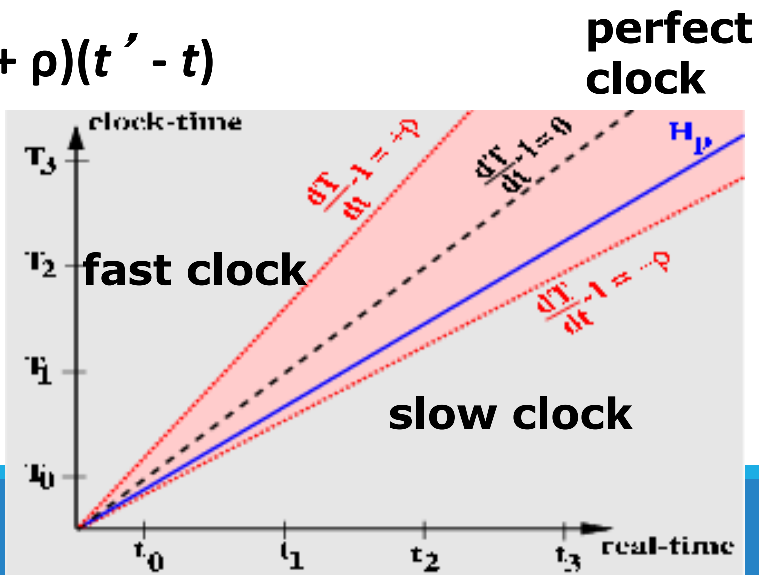
An hardware clock *H* is ***correct*** if its drift rate is within a limited bound of $\rho > 0$ (e.g. 10^{-6} secs/ sec).

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

In presence of a correct hardware clock H we can measure a time interval [t,t'] (for all t' >t) introducing only limited errors.

$$(1 - \rho)(t' - t) \leq H(t') - H(t) \leq (1 + \rho)(t' - t)$$

- Max skew D: resynchronize at least every D/2ρ seconds.



Correct Clock (2/2)

Software clocks have to be monotone

$$\underbrace{t' > t}_{\text{real time}} \rightarrow C(t') > C(t)$$

you cou go back with the time

The monotonic property can be guaranteed choosing opportune α and β.

- Note that α and β can be a function of time

Clock failure:

- crash failure – clock simply stops
- arbitrary failure – clock behaves arbitrarily (e.g. Y2K bug: the day after the 31/1/1999 becomes 1/1/1900 rather than 1/1/2000)

Correctness is no accuracy...

Synchronization Algorithms

Synchronization by mean of a Time Server

Centralized Time Service

- Request-driven
  - Christian’s Algorithm
- Broadcast-based
  - Berkeley Unix algorithm - Gusella & Zatti (1989)

Distributed Time Service (Network Time Protocol)

Christian's Algorithm

→ External synchronization algorithm

Use a time server S that receives a signal from an UTS source

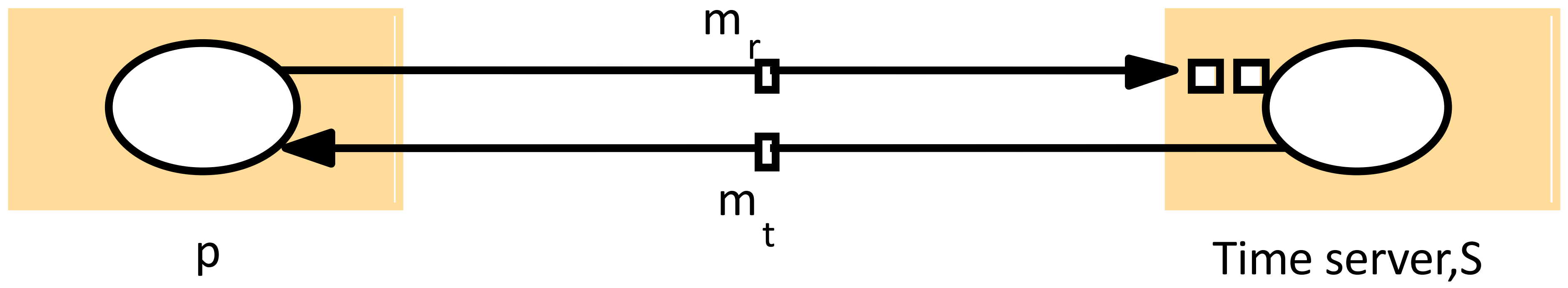
Works (probabilistically) also in an asynchronous system

- Is based on message round trip time (RTT)
- Synchronization is reached only if RTTs are small with respect to the required accuracy

Christian’s Algorithm

A process *p* asks the current time through a message m_r and receives *t* in m_t from *S*

p sets its clock to $t + T_{round}/2$, T_{round} is round trip time experienced by p



Notes:

- A time server can crash
- Cristian suggests to use a cluster of synchronized time servers
- A time server can be attacked…

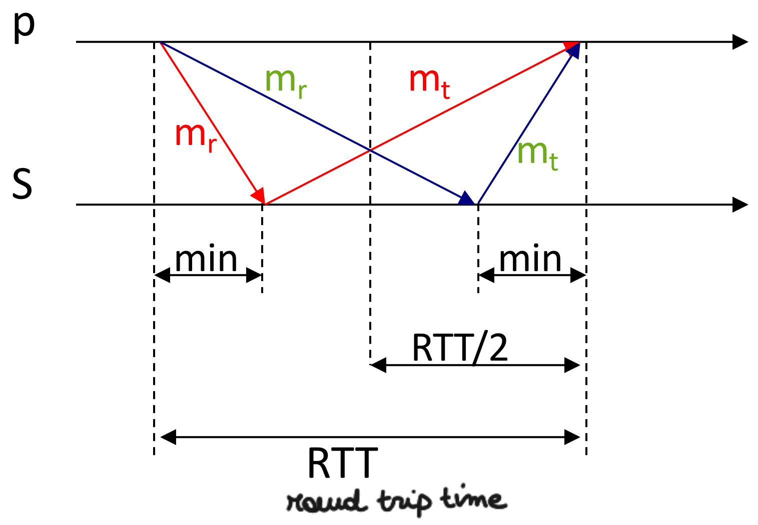
Accuracy

Case 1

Reply time is greater than estimate one (obtained by RTT/2), in particular is equal to (RTT-min)

Δ = estimate of response – real time = (RTT/2) - (RTT - min) =

(-RTT+2min)/2 = -RTT/2 +min = **-(RTT/2 - min)**



Case 2

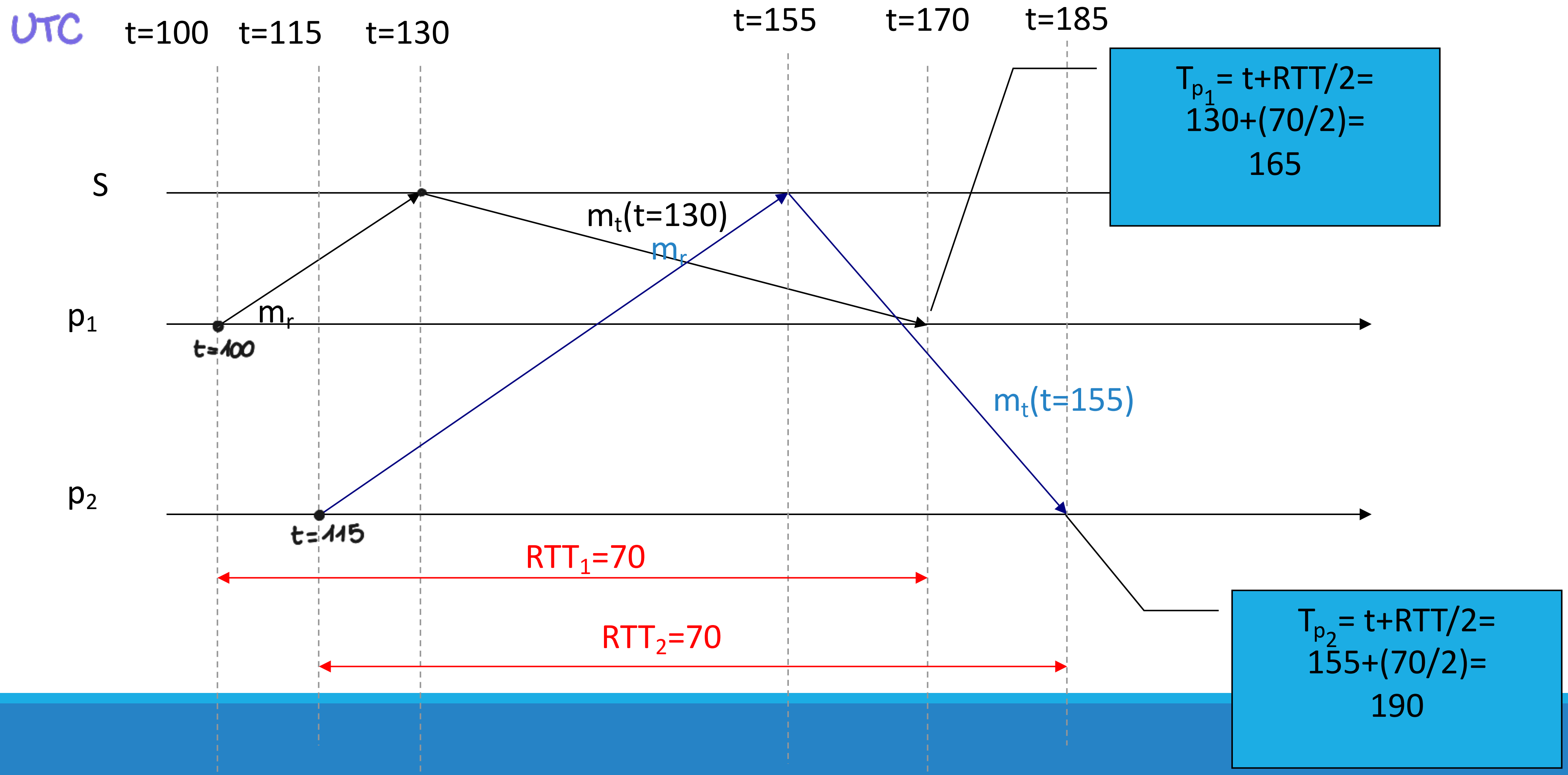
Reply time is smaller than estimate one (obtained by RTT/2), in particular is equal to (RTT-min)

Δ = estimate of response – real time = **(RTT/2) - min = + (RTT/2 - min)**

Consequently the accuracy of Cristian’s algorithm is
± (RTT/2 – *min*) where *min* is the minimum transmission delay

depends on the channel

Christian's algorithm example



T_{p_1}= t+RTT/2=
130+(70/2)=
165

T_{p_2}= t+RTT/2=
155+(70/2)=
190

In the previous scenario

- If the minimum message transmission time is t_{min}= 30 then the accuracy is ±5 (i.e. ± RTT/2-t_{min} = 70/2 – 30 = ± 5)

- If the minimum message transmission time is t_{min}= 20 then the accuracy is ±15 (i.e. ± RTT/2-t_{min} = 70/2 – 20 = ± 15)

Discussion if UTC faie, there is a probleu

The synchronization server is a single point of failure

- There could exists periods in which the synchronization is not possible -> Ask to multiple servers at the same time (synchronization group)

Servers in the group may be arbitrarily faulty or malicious

- Add redundancy
- Use authentication

$\left.\right] \rightarrow$ to solve failure

Berkeley's Algorithm

- Internal synchronization algorithm
  - master-slave structure
  - Based on steps
    - gathering of all the clocks from other processes and computation of the difference
    - computation of the correction

grappo

P1 : RUN MASTER PROTOCOL

P2, ..., Pn : RUN SLAVE PROTOCOLS

Measuring the difference between clocks

The master process p_m sends a message with a timestamp t_1 (local clock value) to each process of the system (p_m included)

When a process p_i receives a message from the master, it sends back a reply with its timestamp t_2 (local clock value)

When the master receives the reply message it reads the local clock (t_3) and compute the difference between the clocks $\Delta=(t_1 + t_3)/2 - t_2$

Synchronization Algorithm

Master behavior

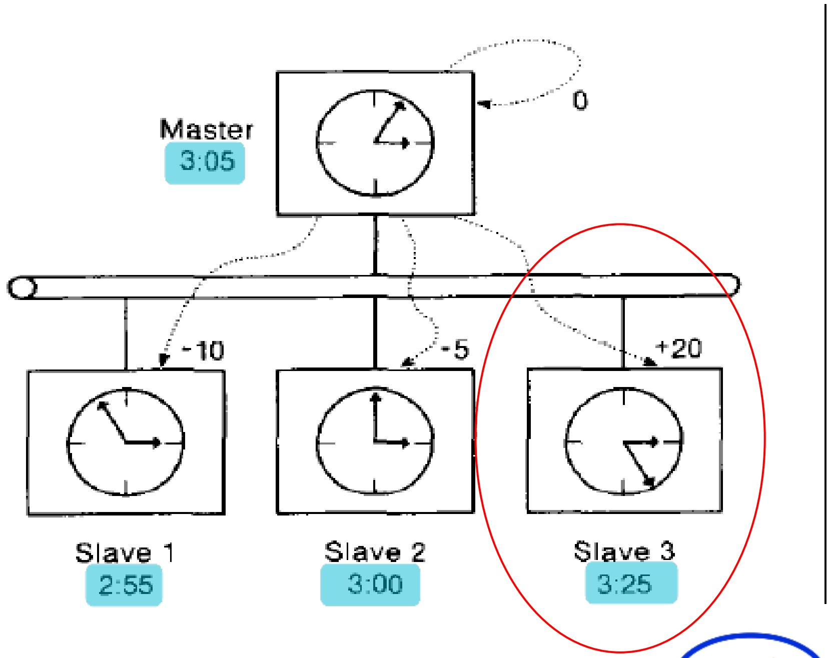
- Computes of the differences Δp_i between the master clock and the clock of every other process p_i (including also the master)
- Computes the average avg of all Δp_i without considering faulty[1] processes
- Computes the correction of each process (including faulty processes)
  - $Adg_{pi} = avg - \Delta p_i$

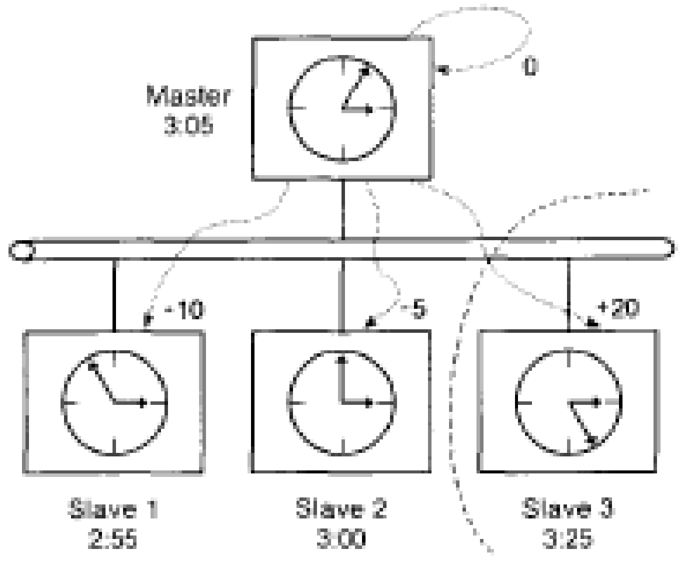
Slaves behavior

- When a process receives the correction, it is applied to the local clock
- If the correction is a negative one, the process do not adjust the value but it slow down its clock

1. A faulty process is a process that has a clock which differ from the one of the master more than a given threshold γ

Example





Faulty

by the master

Measuring the differences

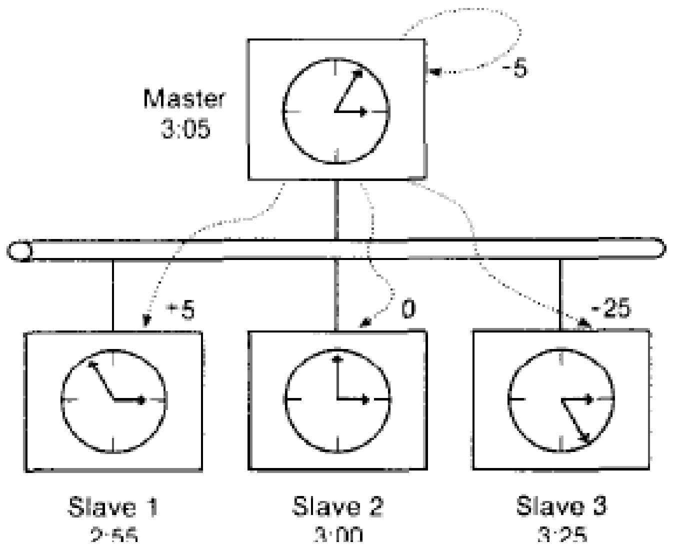
- $\Delta p_m = 3{:}05 - 3{:}05 = 0$
- $\Delta p_1 = 3{:}05 - 2{:}55 = -10$
- $\Delta p_2 = 3{:}05 - 3{:}00 = -5$
- $\Delta p_3 = 3{:}05 - 3{:}25 = 20$ value too far away

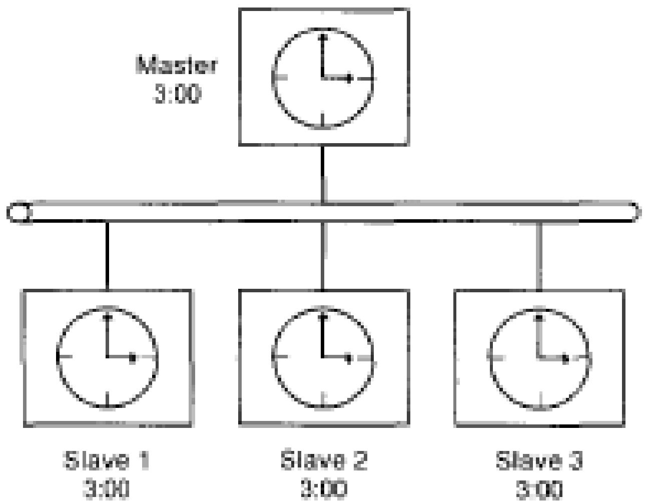
Computing the average

- Avg = (0 -10 -5) /3 = -5

CORRECTION that every slave process had to adopt

Example





Compute and send the Correction -5

- $Adj_m = Avg - \Delta p_m = -5 - 0 = -5$
- $Adj_1 = Avg - \Delta p_1 = -5 - (-10) = 5$
- $Adj_2 = Avg - \Delta p_2 = -5 - (-5) = 0$
- $Adj_3 = Avg - \Delta p_3 = -5 - 20 = -25$ RECOVER THE FAULTY PROCESS with the CORRECTION

Apply the correction

Berkeley's algorithm: accuracy

The protocol accuracy depends on the maximum round-trip time

- The master does not consider clock values associated to RTT grater than the maximum one

Fault tolerance: Collaborative approach

- If the master crashes, another master is elected (in an unknown time)
- It is tolerant to arbitrary behavior (eg. slaves that send wrong values)
    - Master process consider a certain number of clock values and these values do not differ between them over a certain threshold

Berkeley Algorithm

- Observation: what does slowing down a clock mean?

- It is not possible to impose a clock value in past to slaves that have a clock value greater than the new computed mean.
  - This action can violate the cause/effect ordering of the events produced by the slave and the time monotonicity.

- Consequently we slow down clocks hiding interrupts.
  - Hiding interrupts, the local clock is not updated so that we have to hide a number of interrupt equals to slowdown time divides the interrupt period.

Network Time Protocol (NTP)

Time service over **Internet** - synchronizes clients with UTC:

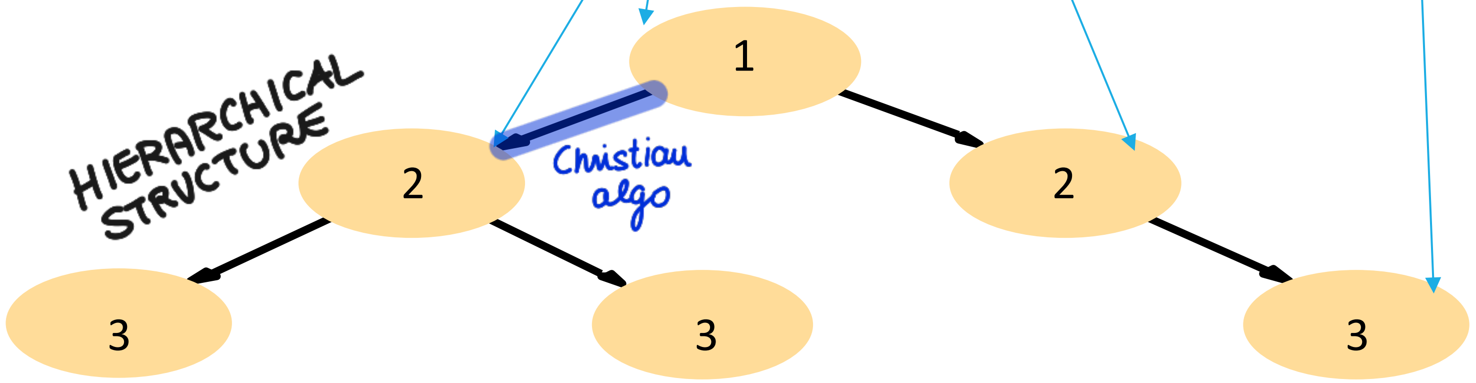
Reliability by mean of redundant server and path

Scalable

Primary servers are connected to UTC sources

Secondary servers are synchronized to primary servers

Synchronization subnet - lowest level servers in users' computers



Network Time Protocol

Synchronization of clients relative to UTC on an Internet-wide scale

- NTP is a *standard de facto* for ***external*** clock synchronization of distributed system on Internet
  internal synch. with failure between layers
- NTP employs several security mechanisms (e.g. mechanisms for authentication of time references) usually they are not required in a local area network

Based on a remote reading procedure like Cristian's Algorithm

- NTP specification adds to the basic algorithm mechanisms for clustering, filtering and evaluating data quality in order to minimize the synchronization

NTP – Server Synchronization

The NTP hierarchy is reconfigurable in presence of faults

- Primary server that loses its connection with UTC-signal can become a secondary server

- Secondary server that loses its connection with a primary server (e.g. a crash of the primary server) can contact and connect itself to another primary.

NTP Synchronization Modes

Multicast: server periodically sends its actual time to its leaves in the LAN. Leaves set their time using the received time assuming a certain delay. It is used in quick LANs but it shows a low accuracy

Procedure call: server replies to requests with its actual timestamp (like Cristian' s algorithm). High Accuracy and it is useful when it is not available hw multicast. → in case of failure

Symmetrical: used to synchronize between pairs of time servers using messages containing timing information. Only used in high level of hierarchy.

Time in Asynchronous Systems

Physical Time: A global property… Observable?

NO in a distributed asynchronous system:
different clocks are synchronizable only with a <u>certain</u> probability

- The impossibility of perfect accuracy is due to unpredictability of communication delay.
  - We can introduce a bound for the accuracy only when we known the upper and lower bounds for communication delays.