TIME IN DISTRIBUTED SYSTEMS I

DISTRIBUTED SYSTEMS

Master of Science in Cyber Security

A.Y. 2023/2024

SAPIENZA
UNIVERSITÀ DI ROMA

CIS SAPIENZA
CYBER INTELLIGENCE AND INFORMATION SECURITY

OUTLINE

- The Three Models of Times:
  - Eventual Synchrony
- Clock Synchronisation in Synchronous Systems:
  - Physical Clocks
  - Christian's Algorithm
  - Berkley's Algorithm
  - Synchronisation in Arbitrary Graphs

Hourglass, Inga Dambrauskienė

TIME MODELS: ASYNCHRONOUS, SYNCHRONOUS AND EVENTUALLY SYNCHRONOUS.

SAPIENZA
UNIVERSITÀ DI ROMA

CIS SAPIENZA
CYBER INTELLIGENCE AND INFORMATION SECURITY

MODELS

- Asynchronous:
  - The model we have seen;
  - There is no bound on the delay that a message can cumulate.
- Synchronous:
  - There is a known bound on the delay that a message can cumulate (i.e., If I send a message a time t, it has to be received by t+\delta).
- Eventually-Synchronous:
  - There is a time, unknown to us, after which the system will be synchronous.

ASYNCHRONOUS

- No Global-clock (you have only a local clock that can be used to order the events locally);
- No Bound on message delays;
- Local-computational steps (Exec) happen at unpredictable time (the scheduler…).

SYNCHRONOUS

- Delays are bounded
- You can synchronise clocks (up-to a certain precision…)
- You can assume that local execution steps happen at certain predetermined interval, and that they take bounded time.

Asynchronous

Agreement Impossible and a lot of other things
Models Everything

Synchronous

Almost everything is possible
Models only restricted networks

EVENTUALLY-SYNCHRONOUS

- Real systems are synchronous most of the time

```
Giuseppe$ ping google.com
PING google.com (172.217.21.78): 56 data bytes
64 bytes from 172.217.21.78: icmp_seq=0 ttl=53 time=43.239 ms
64 bytes from 172.217.21.78: icmp_seq=1 ttl=53 time=44.621 ms
64 bytes from 172.217.21.78: icmp_seq=2 ttl=53 time=43.540 ms
64 bytes from 172.217.21.78: icmp_seq=3 ttl=53 time=42.740 ms
64 bytes from 172.217.21.78: icmp_seq=4 ttl=53 time=36.479 ms
64 bytes from 172.217.21.78: icmp_seq=5 ttl=53 time=43.676 ms
64 bytes from 172.217.21.78: icmp_seq=6 ttl=53 time=43.546 ms
64 bytes from 172.217.21.78: icmp_seq=7 ttl=53 time=44.264 ms
64 bytes from 172.217.21.78: icmp_seq=8 ttl=53 time=43.370 ms
…
```

EVENTUALLY-SYNCHRONOUS

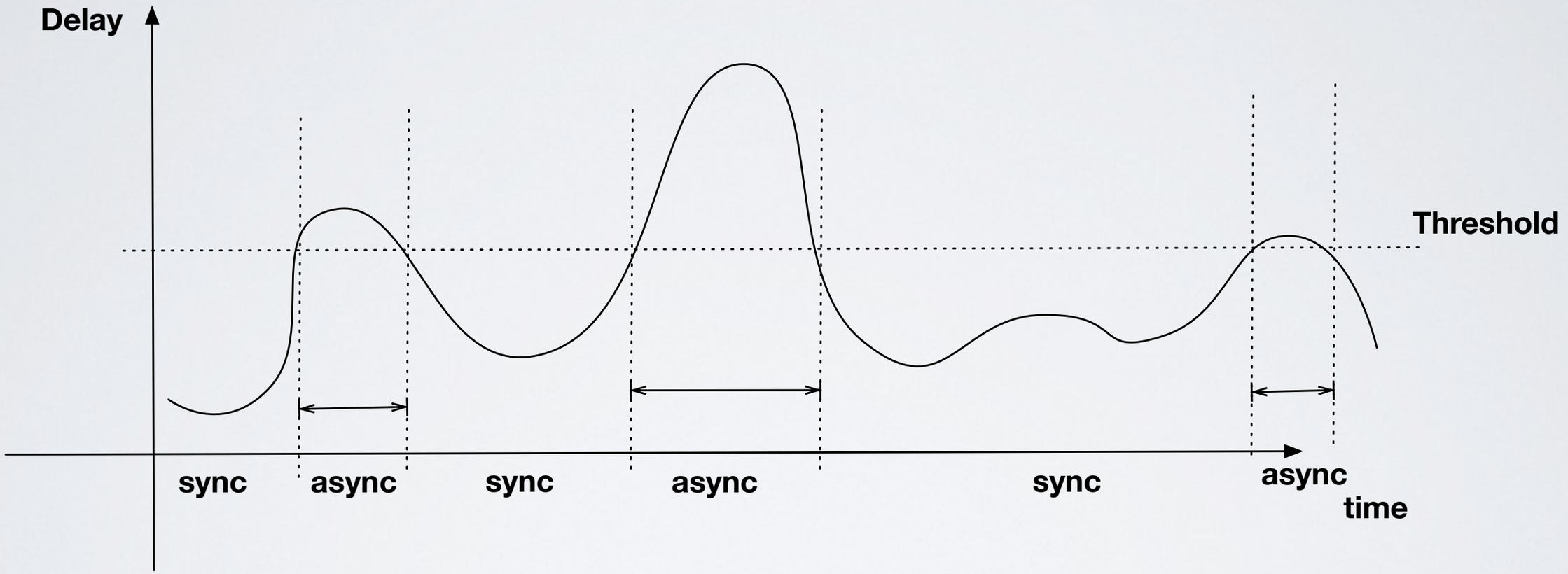
- Real systems are synchronous **most of the time**

```
Giuseppe$ ping google.com
PING google.com (172.217.21.78): 56 data bytes
64 bytes from 172.217.21.78: icmp_seq=0 ttl=53 time=43.239 ms
64 bytes from 172.217.21.78: icmp_seq=1 ttl=53 time=44.621 ms
64 bytes from 172.217.21.78: icmp_seq=2 ttl=53 time=43.540 ms
64 bytes from 172.217.21.78: icmp_seq=3 ttl=53 time=42.740 ms
64 bytes from 172.217.21.78: icmp_seq=4 ttl=53 time=36.479 ms
64 bytes from 172.217.21.78: icmp_seq=5 ttl=53 time=43.676 ms
64 bytes from 172.217.21.78: icmp_seq=6 ttl=53 time=43.546 ms
64 bytes from 172.217.21.78: icmp_seq=7 ttl=53 time=44.264 ms
64 bytes from 172.217.21.78: icmp_seq=8 ttl=53 time=43.370 ms
…
64 bytes from 172.217.21.78: icmp_seq=17 ttl=53 time=38.378 ms
64 bytes from 172.217.21.78: icmp_seq=18 ttl=53 time=739.609 ms
64 bytes from 172.217.21.78: icmp_seq=19 ttl=53 time=38.159 ms
```

- We have to formalise **most of the time**, that is the concept of ``partial-synchrony".

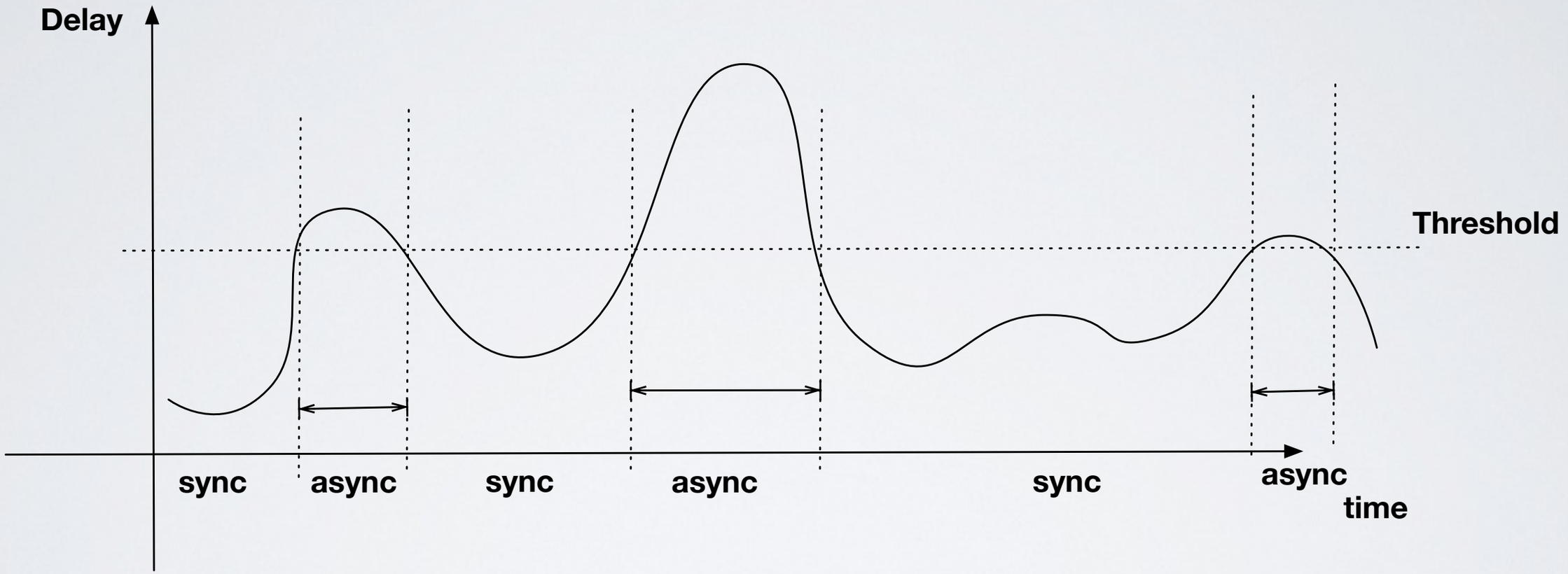
EVENTUALLY-SYNCHRONOUS

- We have to formalise **most of the time**, that is the concept of ``partial-synchrony".



EVENTUALLY-SYNCHRONOUS

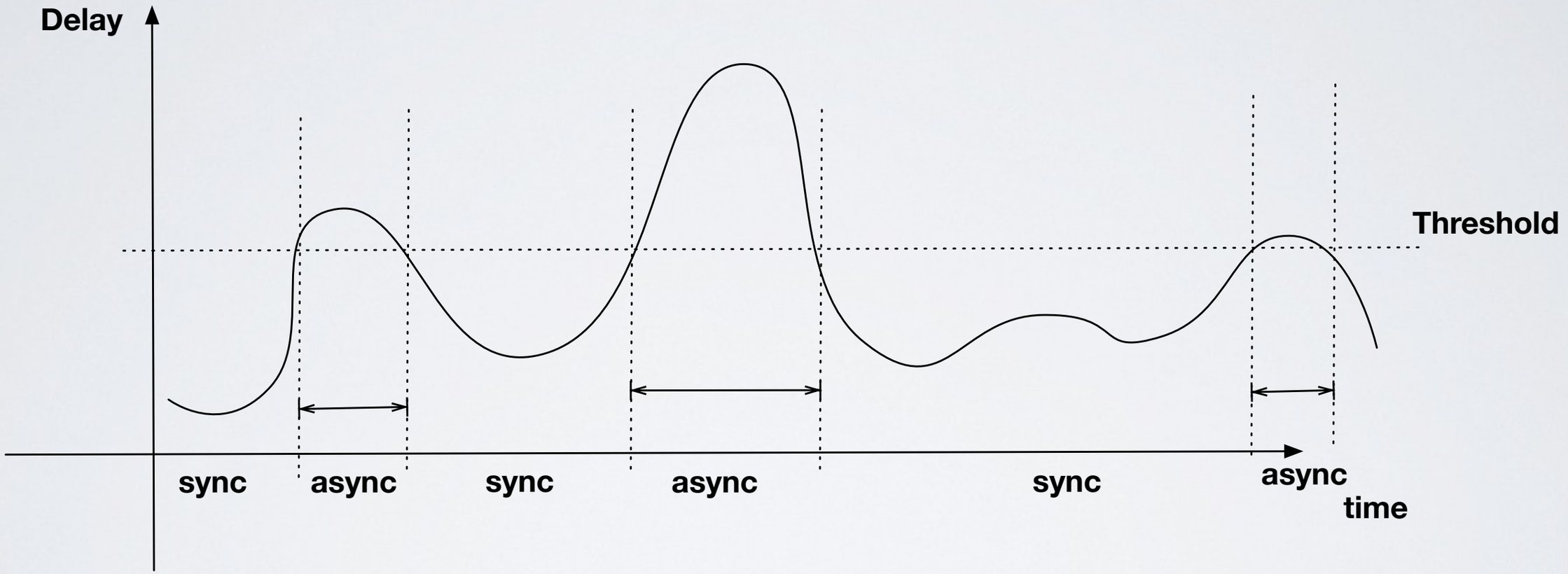
- We have to formalise **most of the time**, that is the concept of ``partial-synchrony".



- Safety property (e.g, Mutex).

EVENTUALLY-SYNCHRONOUS

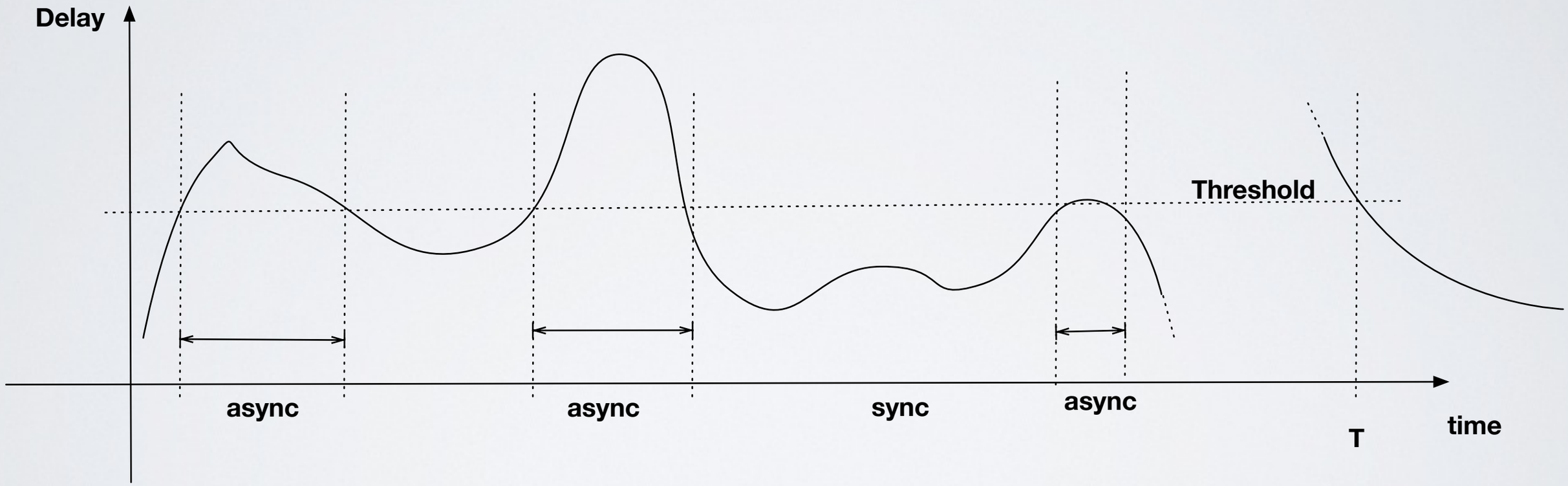
- We have to formalise **most of the time**, that is the concept of ``partial-synchrony".



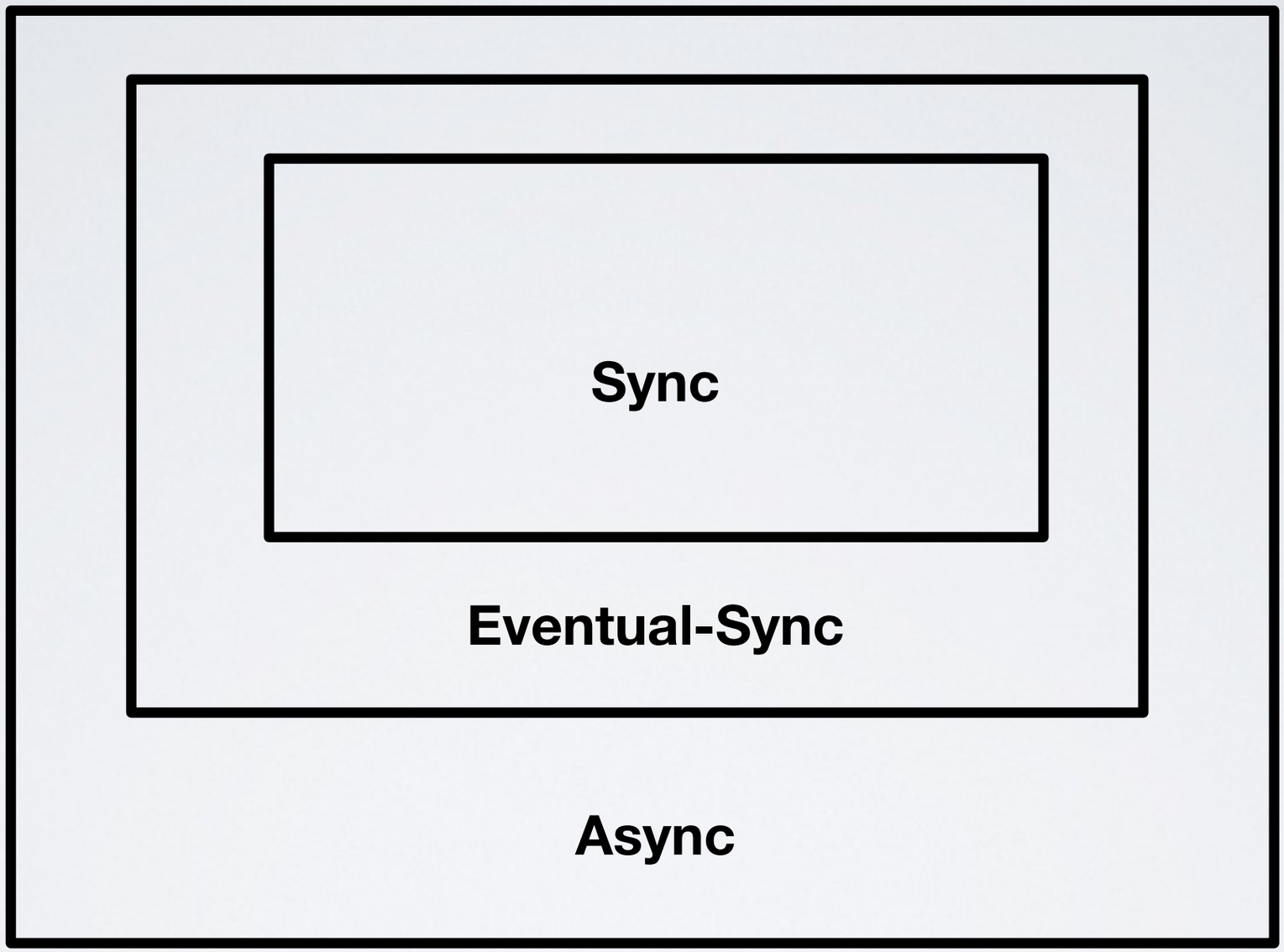
- Liveness property (e.g, No-Starvation).

EVENTUALLY-SYNCHRONOUS

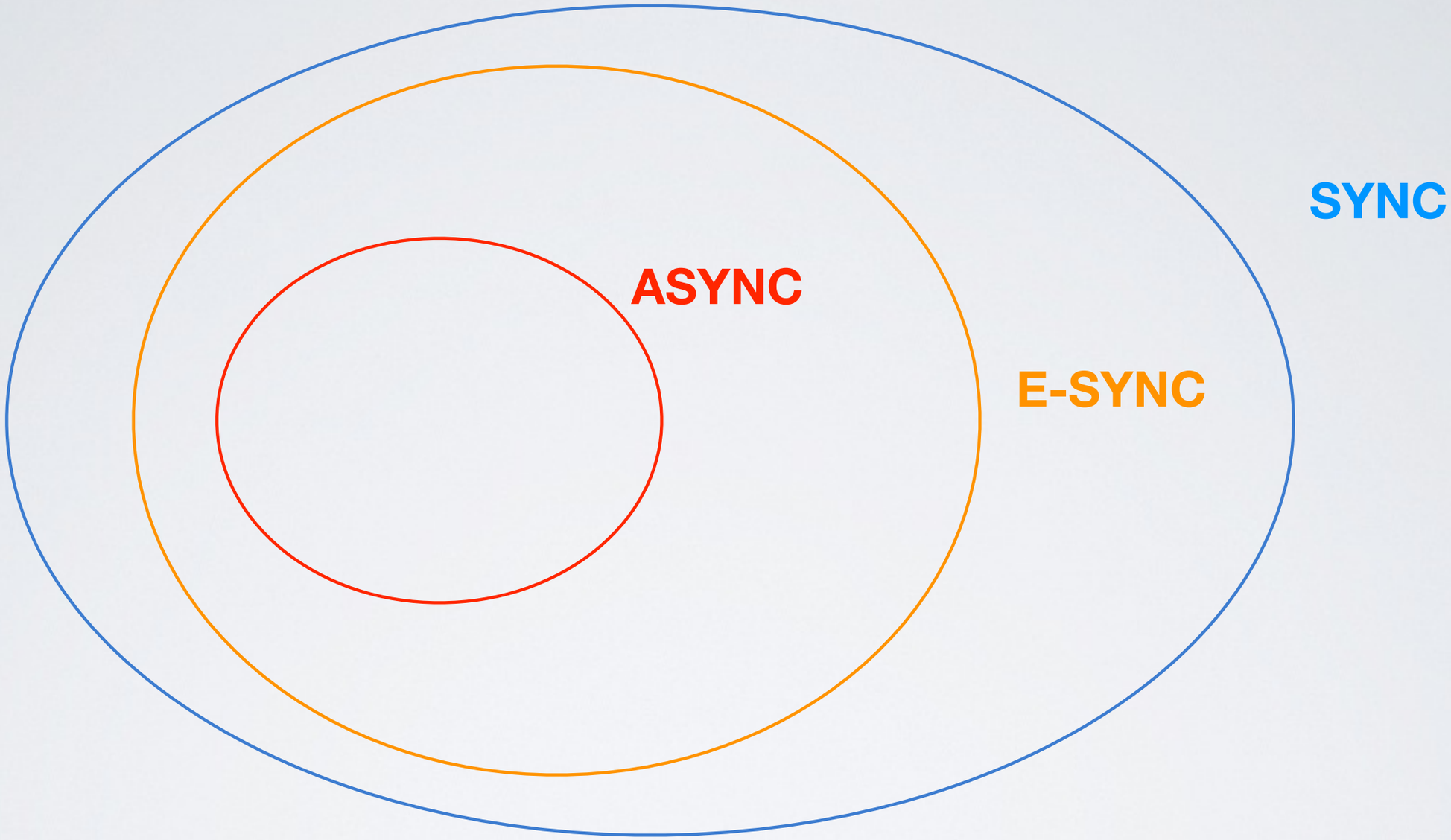
- The system is initially asynchronous.
- There is a time t, unknown to us, after which the system is synchronous and will be synchronous forever.



Relationship between executions



Relationship of solvable problems



CLOCK SYNCHRONISATION IN SYNCHRONOUS SYSTEMS

SAPIENZA
UNIVERSITÀ DI ROMA

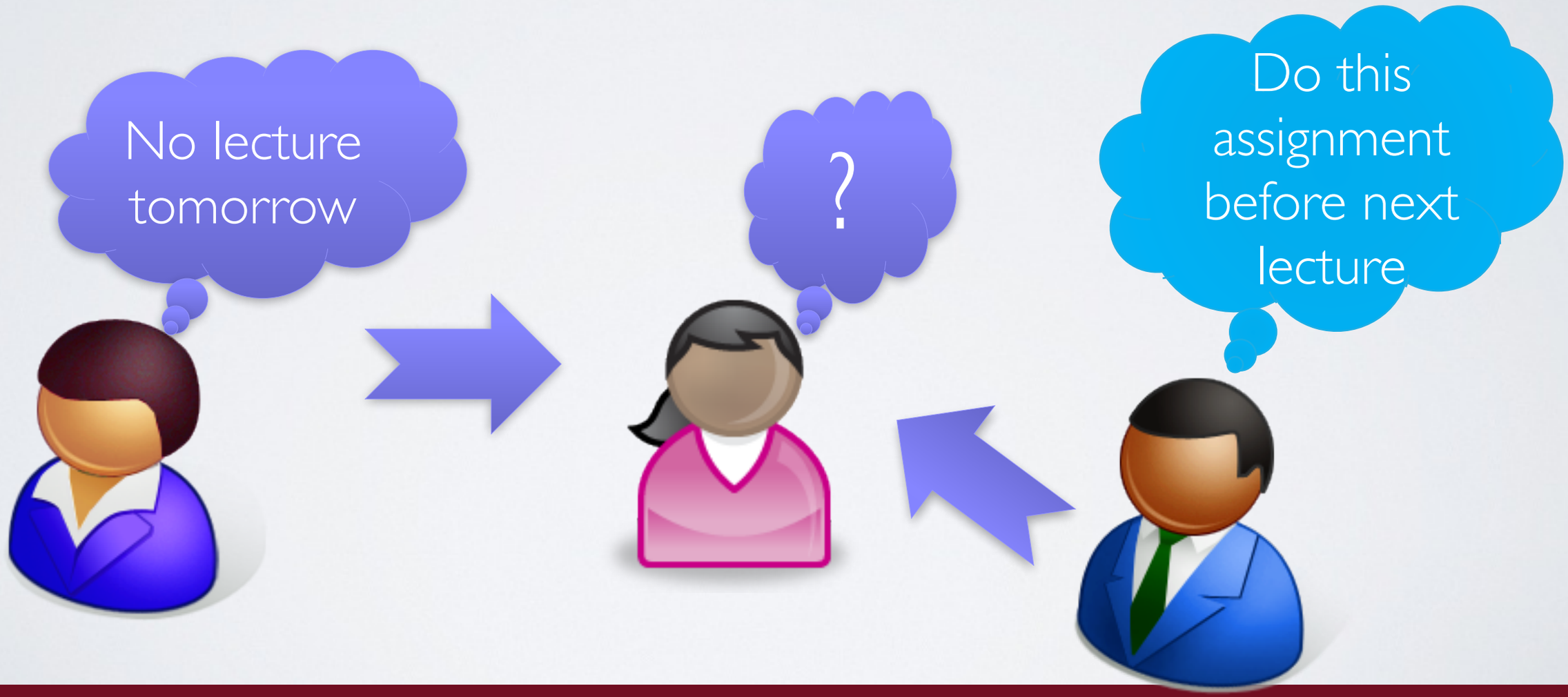
CIS SAPIENZA
CYBER INTELLIGENCE AND INFORMATION SECURITY

INTRODUCTION

- Many applications require ordering among events and synchronization to terminate correctly
  - E.g. Air traffic control, Network monitoring, measurement and control, Stock market, buy and sell orders, etc…

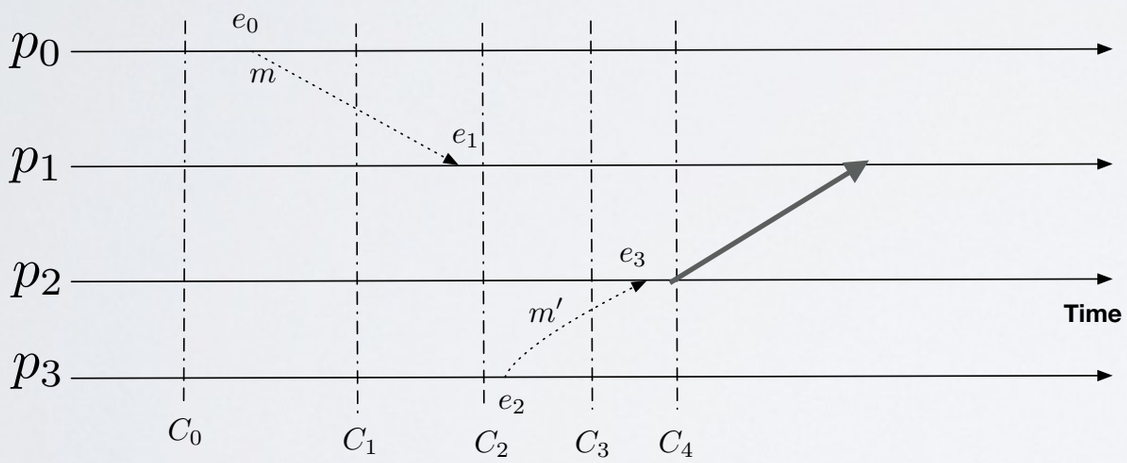
INTRODUCTION

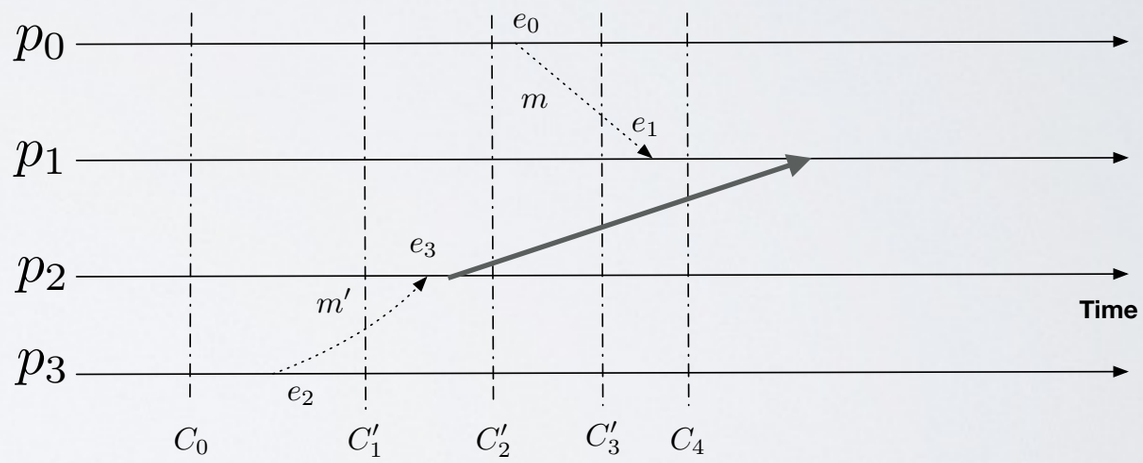
- Many applications require ordering among events and synchronization to terminate correctly
  - E.g. Air traffic control, Network monitoring, measurement and control, Stock market, buy and sell orders, etc…



INTRODUCTION

- Many applications require ordering among events and synchronization to terminate correctly
  - E.g. Air traffic control, Network monitoring, measurement and control, Stock market, buy and sell orders, etc…





TIME IS IMPORTANT -2

- It is a quantity we are interested to measure

- A lot of algorithms depends on time
  - data consistency
  - authentication
  - double processing avoidance (**make**)
  - Sensors network TDMA

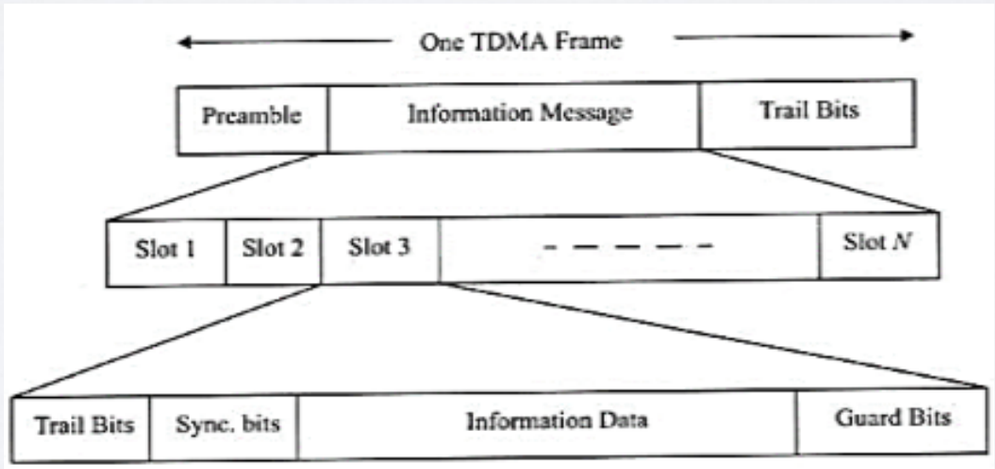


Fig: TDMA frame structure

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 realise a global history of the system.

“Naïf” solution

- Each process timestamps events by mean of its physical clock

PHYSICAL CLOCKS

PHYSICAL COMPUTER CLOCKS

Your local clock is obtained by the operating system by reading a local hardware clock.

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

$$H_i(t) = \int_0^t h_i(\tau) d\tau$$

At real time *t*, the operating system reads the hardware clock $H_i(t)$, and produces the local (software) clock

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

PARAMETERS AFFECTING ACCURACY

Different local clocks may have different values:

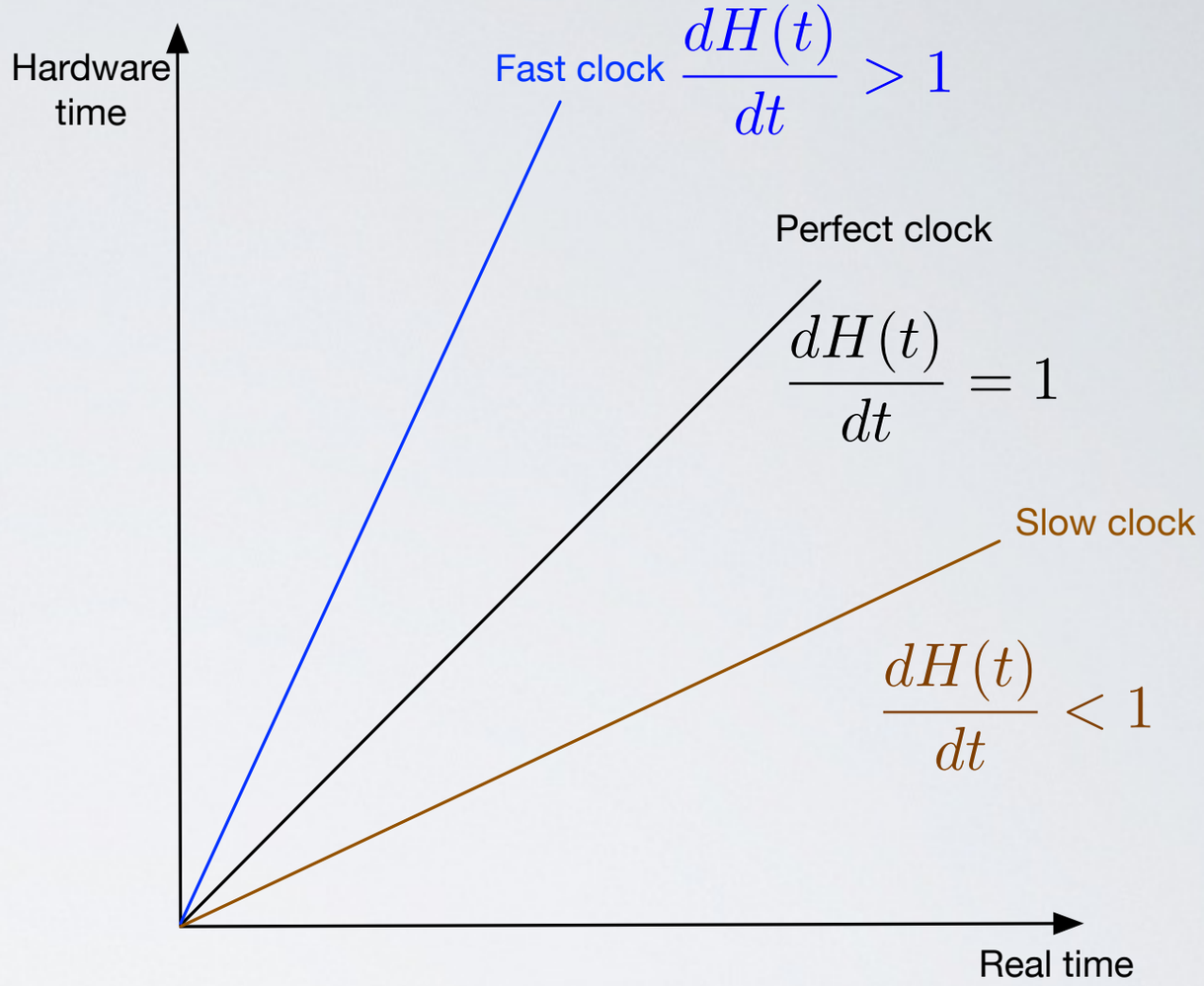
Skew: difference in time between two software clocks

$$Skew_{i,j}(t) = | C_i(t) - C_j(t) |$$

Drift Rate: gradual misalignment of synchronized clocks caused by the slight inaccuracies of the time-keeping mechanisms

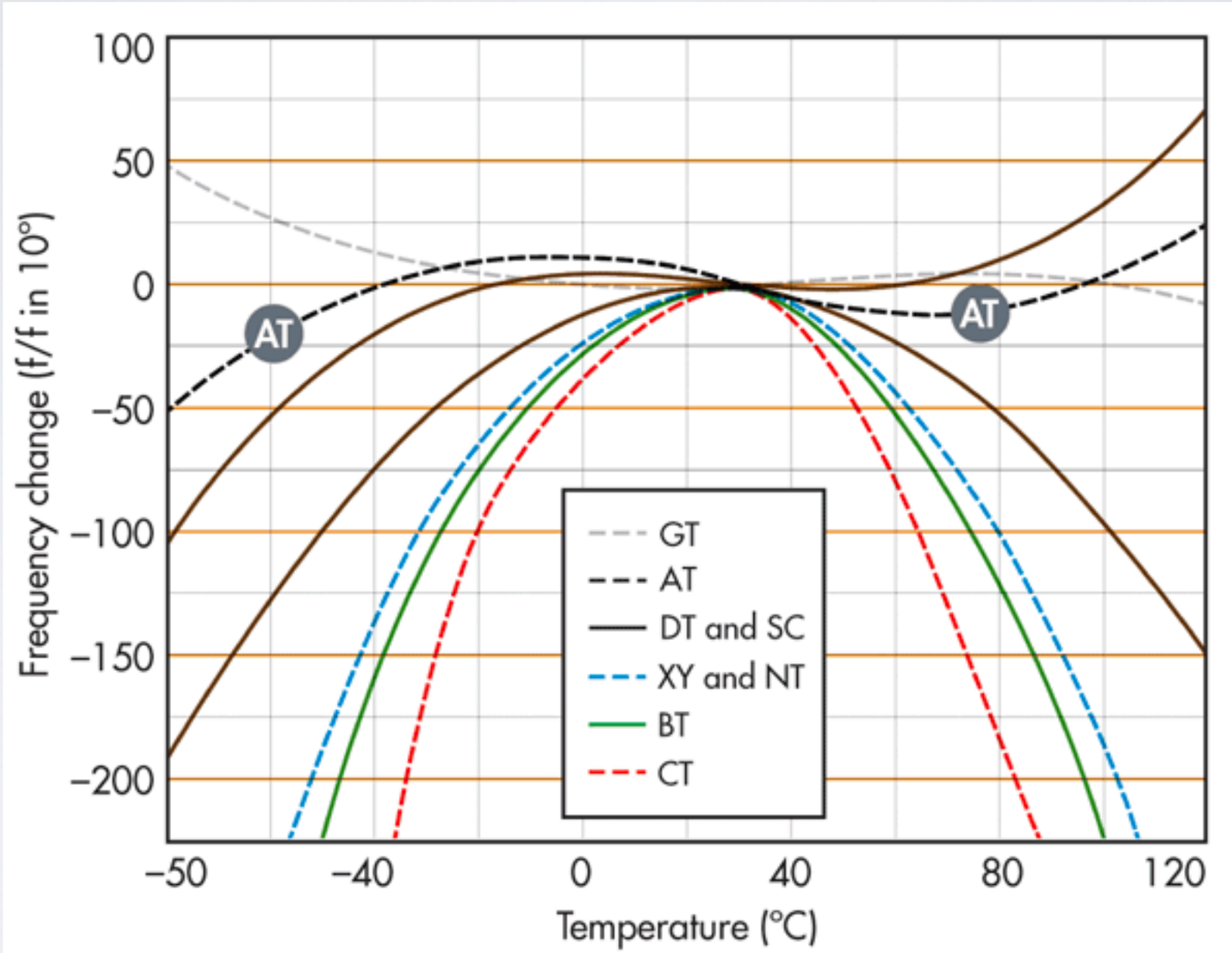
DRIFT RATE

$$\frac{dH(t)}{dt}$$



- A drift rate of 2 microsec/sec means clock increases its value of 1sec+2 microsec for each second.
  - Ordinary quartz clocks deviate nearly by 1 sec in 11-12 days. (10^{-5} secs/sec).
  - High-precision quartz clock drift rate is 10^{-7}-10^{-8} secs/sec

DRIFT RATE


100
50
0
−50
−100
−150
−200
Frequency change (f/f in 10⁶)
AT
AT
GT
AT
DT and SC
XY and NT
BT
CT
−50
−40
0
40
80
120
Temperature (°C)

CORRECT CLOCK (1/2)

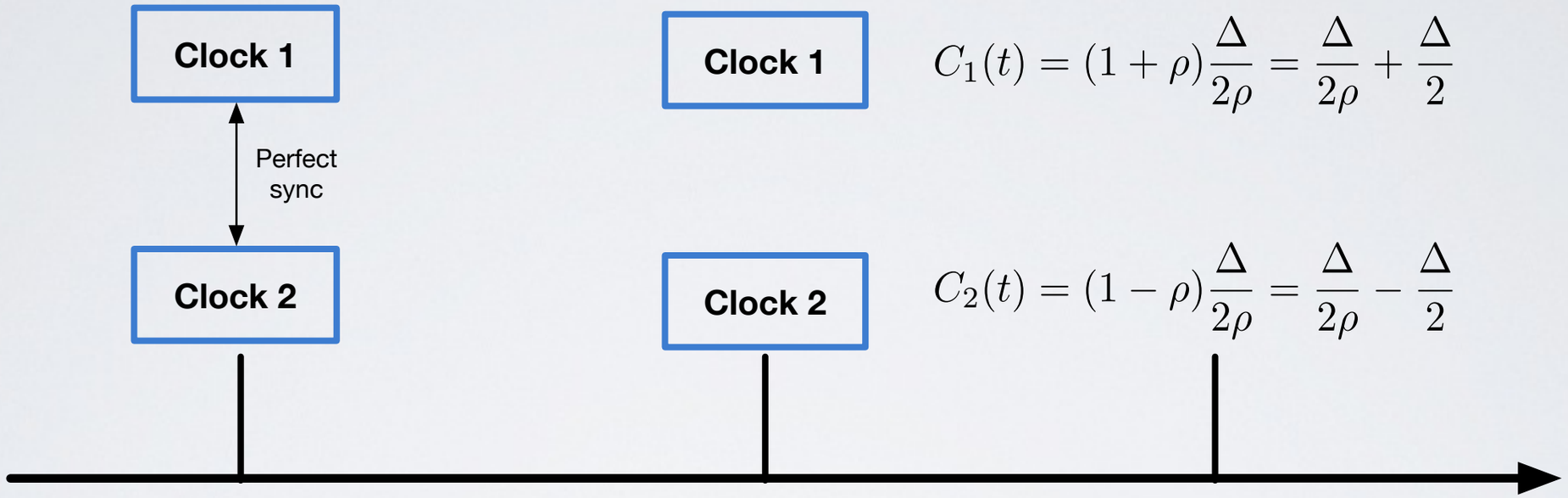
An hardware clock H is correct if its drift rate is within a limited bound of $\rho > 0$ (e.g. 10^{-5} secs/sec).

$$1 - \rho \leq \frac{dH(t)}{dt} \leq 1 + \rho$$

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

$$(1 - \rho)(t_1 - t_0) \leq H(t_1) - H(t_0) \leq (1 + \rho)(t_1 - t_0)$$

BOUNDING THE SKEW



Clock 1

$$C_1(t) = (1+\rho)\frac{\Delta}{2\rho} = \frac{\Delta}{2\rho} + \frac{\Delta}{2}$$

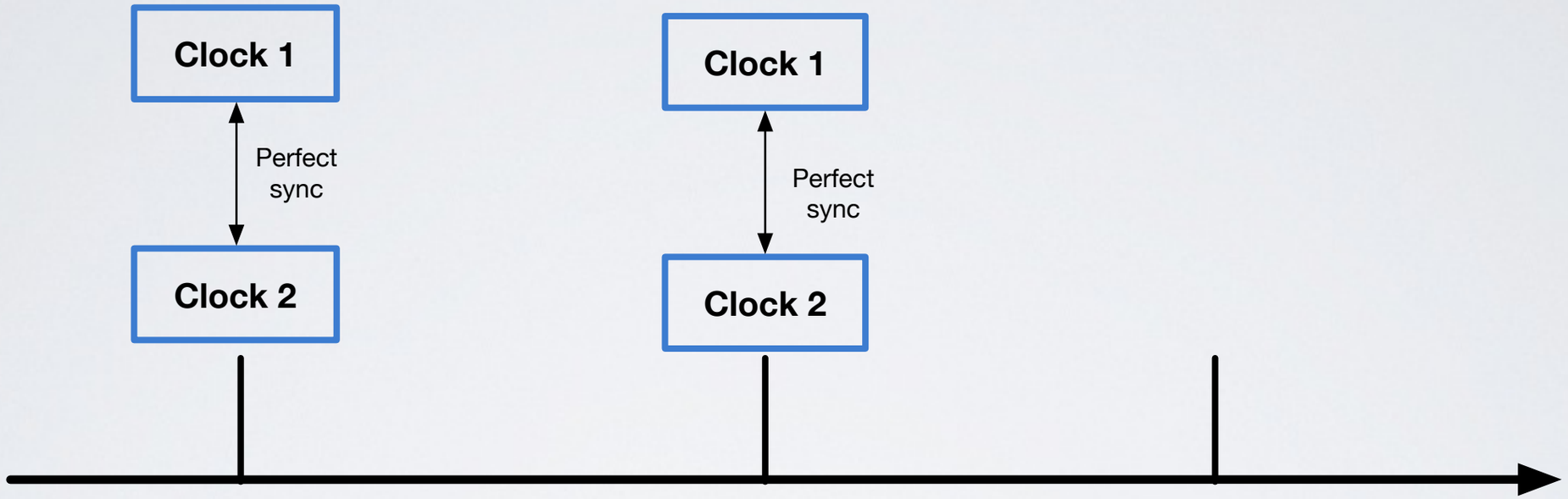
Clock 2

$$C_2(t) = (1-\rho)\frac{\Delta}{2\rho} = \frac{\Delta}{2\rho} - \frac{\Delta}{2}$$

$$t = \frac{\Delta}{2\rho}$$

$$|C_1(t) - C_2(t)| = \Delta$$

BOUNDING THE SKEW



$$t = \frac{\Delta}{2\rho}$$

$$|C_1(t) - C_2(t)| = \Delta$$

$$C_1(t) = \frac{\Delta}{2\rho} + \frac{\Delta}{2}$$

$$C_2(t) = \frac{\Delta}{2\rho} - \frac{\Delta}{2}$$

Clock 1

Clock 2

$$C_1(t) = \frac{\Delta}{2\rho} + \frac{\Delta}{2} - \Delta$$

Correction

$$C_1(t) = \frac{\Delta}{2\rho} + \frac{\Delta}{2}$$

Clock 1

$$C_2(t) = \frac{\Delta}{2\rho} - \frac{\Delta}{2}$$

Clock 2

MONOTONICITY

Software clocks have to be monotone

$$t' > t \rightarrow C(t') > C(t)$$

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

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

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

Observation: what does slowing down a clock mean?

It is not possible to impose a clock value in past.

- This action can violate the cause/effect ordering of the events produced by a process 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 divided by the interrupt period.

UTC, INTERNAL AND EXTERNAL CLOCK SYNCHRONISATION

UNIVERSAL TIME COORDINATED (UTC)

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

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

- Physical clocks based on atomic oscillators are the most accurate clocks (drift rate 10-13)

UTC-signals come from shortwave radio broadcasting stations or from satellites (GPS) with an accuracy of

- 1 msec for broadcasting stations
- 1 μsec for GPS

UTC-signals receivers can be connected to Computers and can be used to synchronize local clocks with the real time

INTERNAL/EXTERNAL SYNCHRONIZATION

External Synchronization

- Processes synchronize their clock C_i with an authoritative external source S
- Let $D>0$ be a synchronization bound and S be the source of UTC
- Clocks C_i (for $i = 1, 2, \ldots N$) are externally synchronized with a time source S (UTC) if for each time interval I:

$$|S(t) - C_i(t)| < D \qquad \forall i \in \{1,2,\ldots,N\},\ \forall t \in I$$

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 among them
- Let $D>0$ be a synchronization bound and let C_i and C_j be the clocks at any two processes p_i and p_j
- Clocks are internally synchronized in a time interval I:

$$|C_i(t) - C_j(t)| < D \quad \forall i, j \in \{1,2,\ldots,N\}, \forall t \in I$$

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

PHYSICAL CLOCK SYNCHRONIZATION

Notes:

- 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.
- 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.

SYNCHRONISATION IN COMPLETE GRAPHS

SYNCHRONISATION 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 UTC source

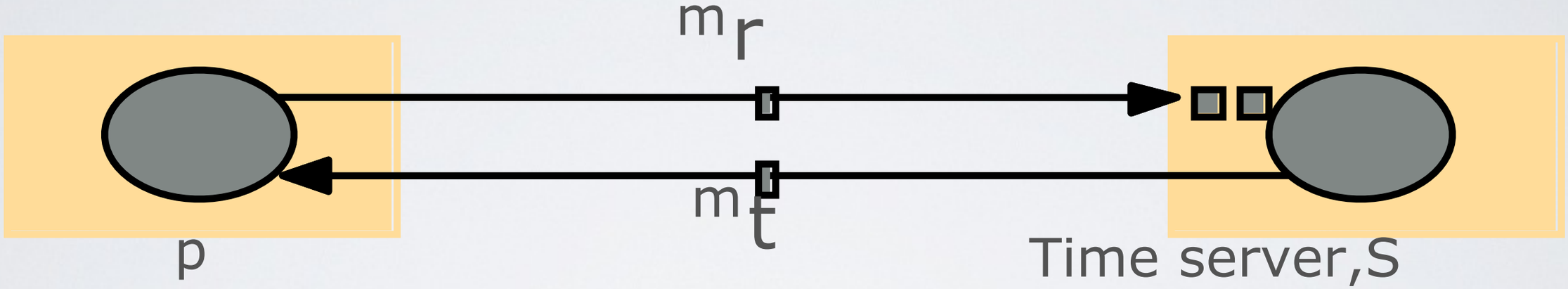
Works (probabilistically) also in an asynchronous system

- Is based on the measurement of message round trip times (RTTs)
- 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 measured by *p*



Notes:

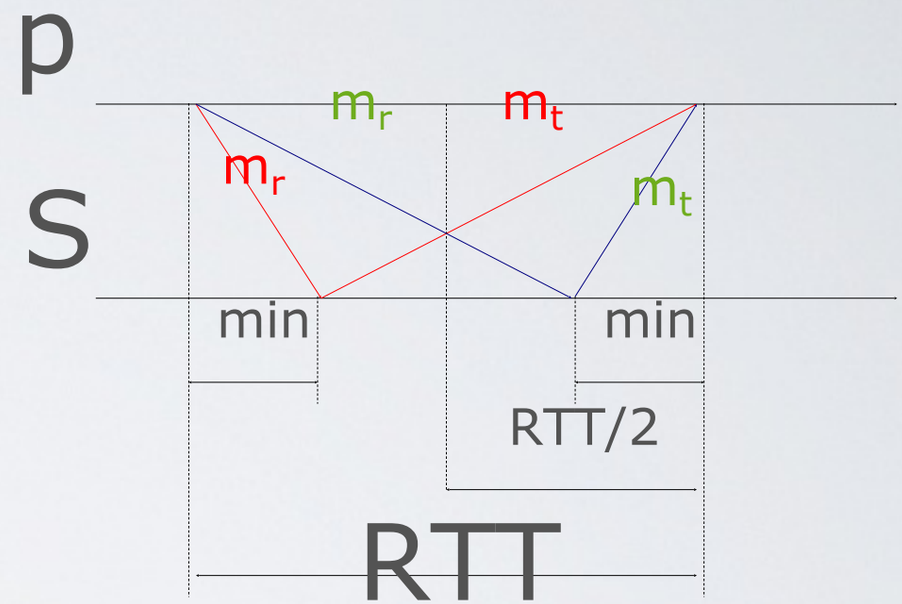
- A time server can crash
- Cristian suggests to use a cluster of synchronized time servers
- A time server can be attacked…

ACCURACY

Case 1

- Reply time is greater than estimate (*RTT/2*)
- Assume it is equal to *RTT-min*

$$\Delta = RTT/2 - (RTT - min) = \frac{-RTT + 2min}{2} = -\frac{RTT}{2} + min = -\left(\frac{RTT}{2} - min\right)$$



Case 2

- Reply time is smaller than estimate (*RTT/2*)
- Assume it is equal to *min*

$$\Delta = \frac{RTT}{2} - min$$

Accuracy is ± (RTT/2 – min)

DISCUSSION

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

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

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 computes the difference between the clocks.

SYNCHRONIZATION ALGORITHM

Master behavior

- Computes of the differences Δp_i between the master clock and the clock of every other process p_i (including the master itself)
- Computes the average *avg* of all Δp_i without considering faulty[1] processes
- Computes the correction for each process (including faulty processes)

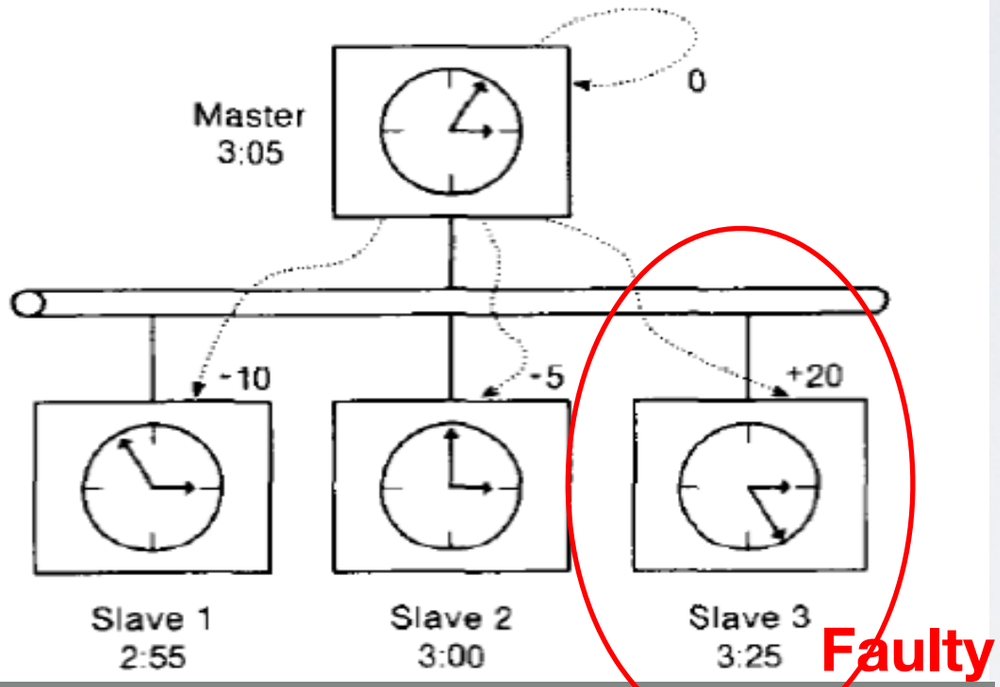
$$adg_{p_i} = avg - \Delta p_i$$

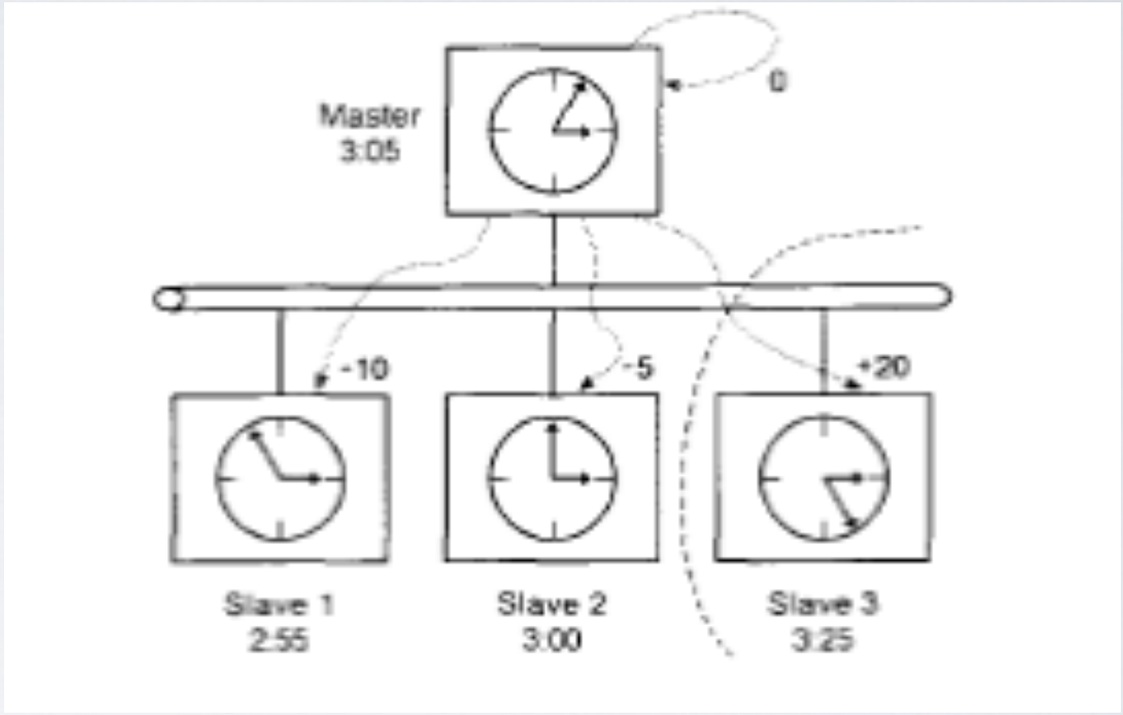
Slaves behavior

- Each process receives the correction and applies it to the local clock
- If the correction is a negative value, it slows 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





Measuring the differences

$\Delta p_m = 3:05 - 3:05 = 0$

$\Delta p_1 = 2:55\text{-}3:05 = -10$

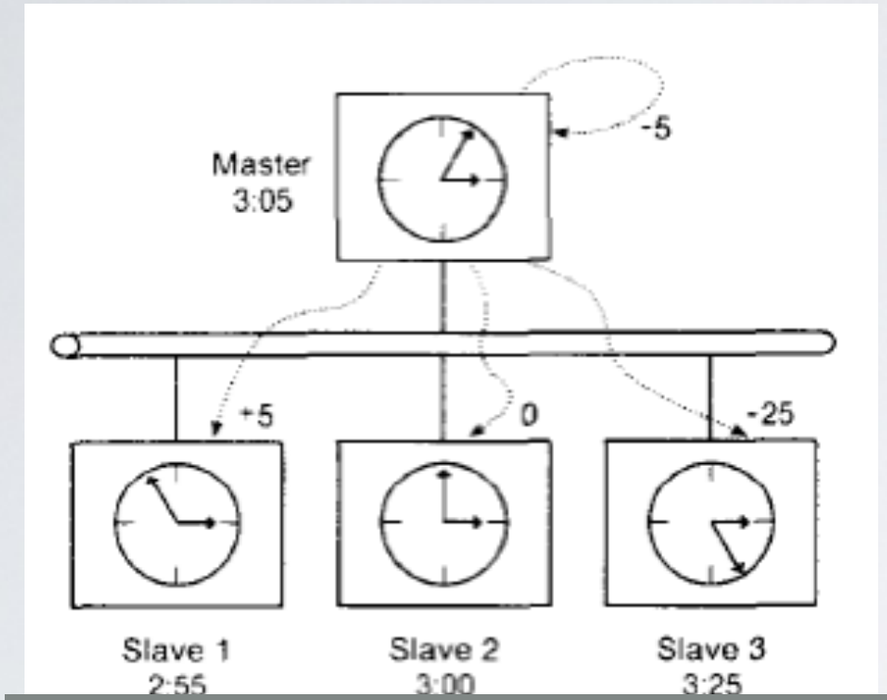
$\Delta p_2 = 3:00 - 3:05 = -5$

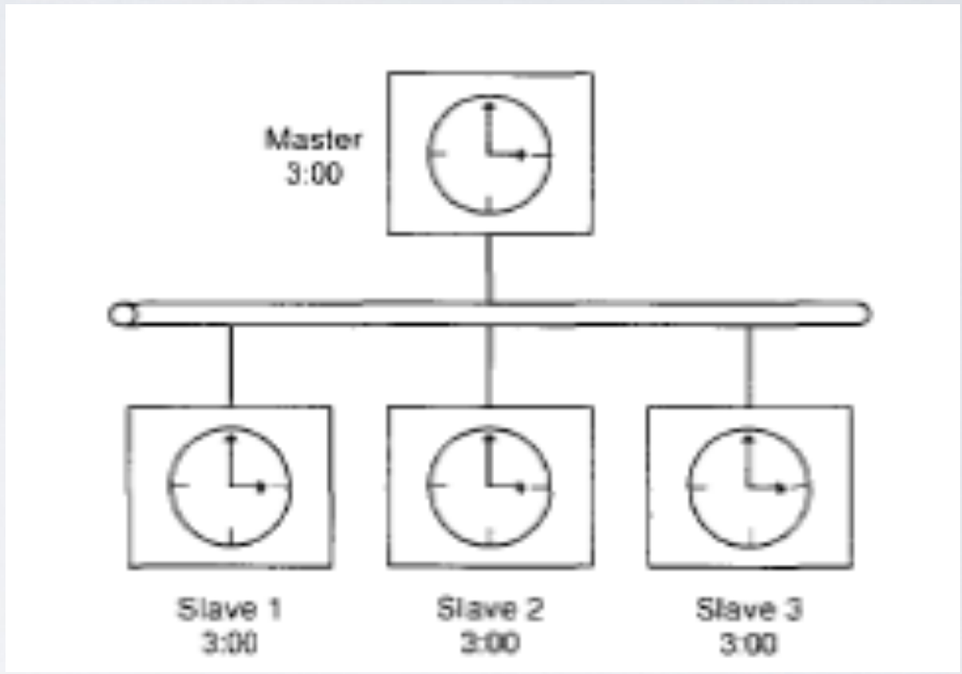
$\Delta p_3 = 3:25 - 3:20 = 20$

Computing the average

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

EXAMPLE





Compute and send the Correction

Adjm = Avg - Δp_m= -5 -0 = -5

Adj1 = Avg - Δp_1= -5 –(-10) = 5

Adj2 = Avg - Δp_2= -5 –(-5) = 0

Adj3 = Avg - Δp_3= -5 -20 = -25

Apply the correction

BERKELEY'S ALGORITHM: ACCURACY

The protocol accuracy depends on the maximum round-trip time

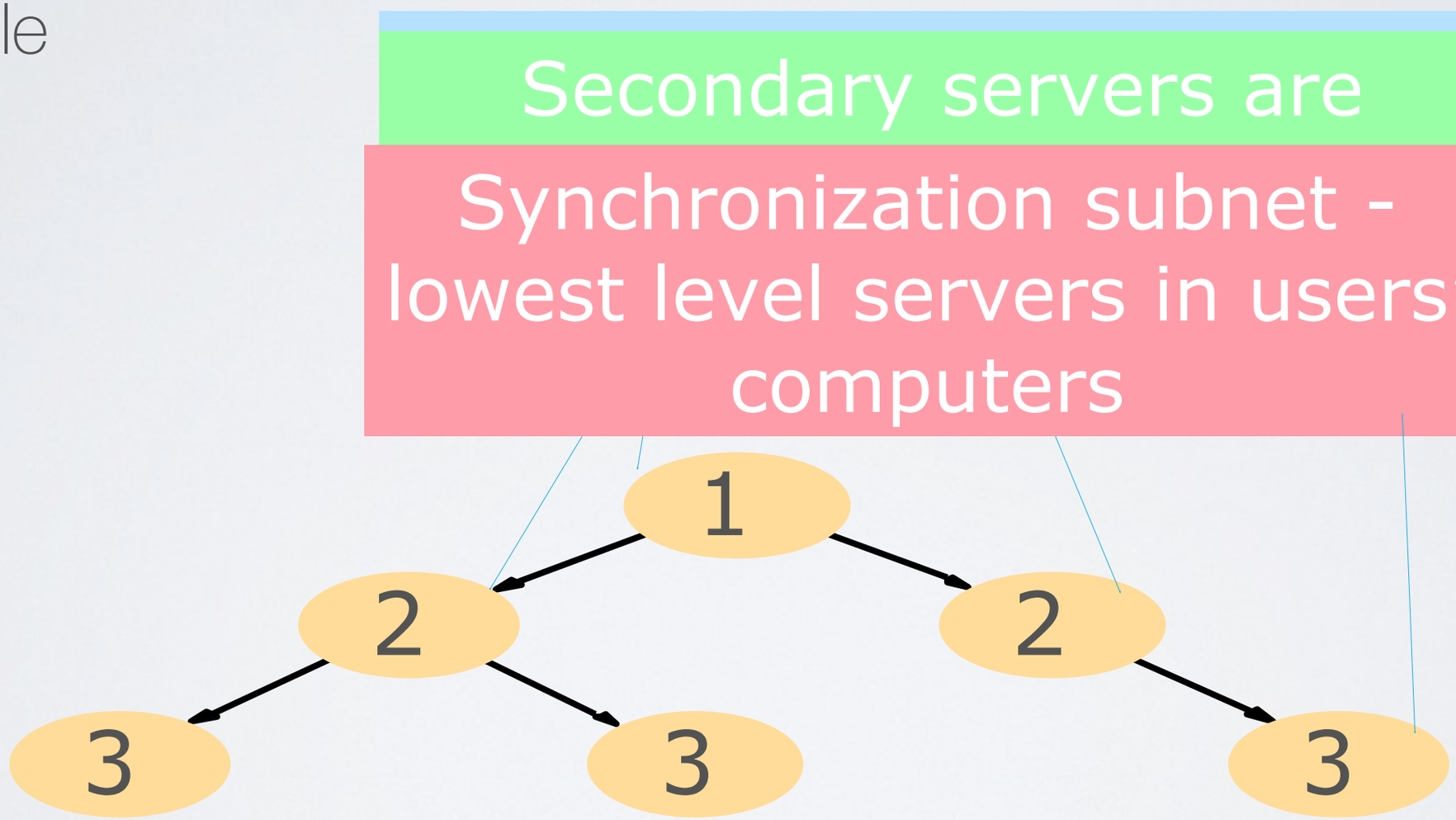
- The master does not consider clock values associated to RTT grater than the maximum one

Fault tolerance

- If the master crashes, another master is elected (in an unknown amount of 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 more than a certain threshold

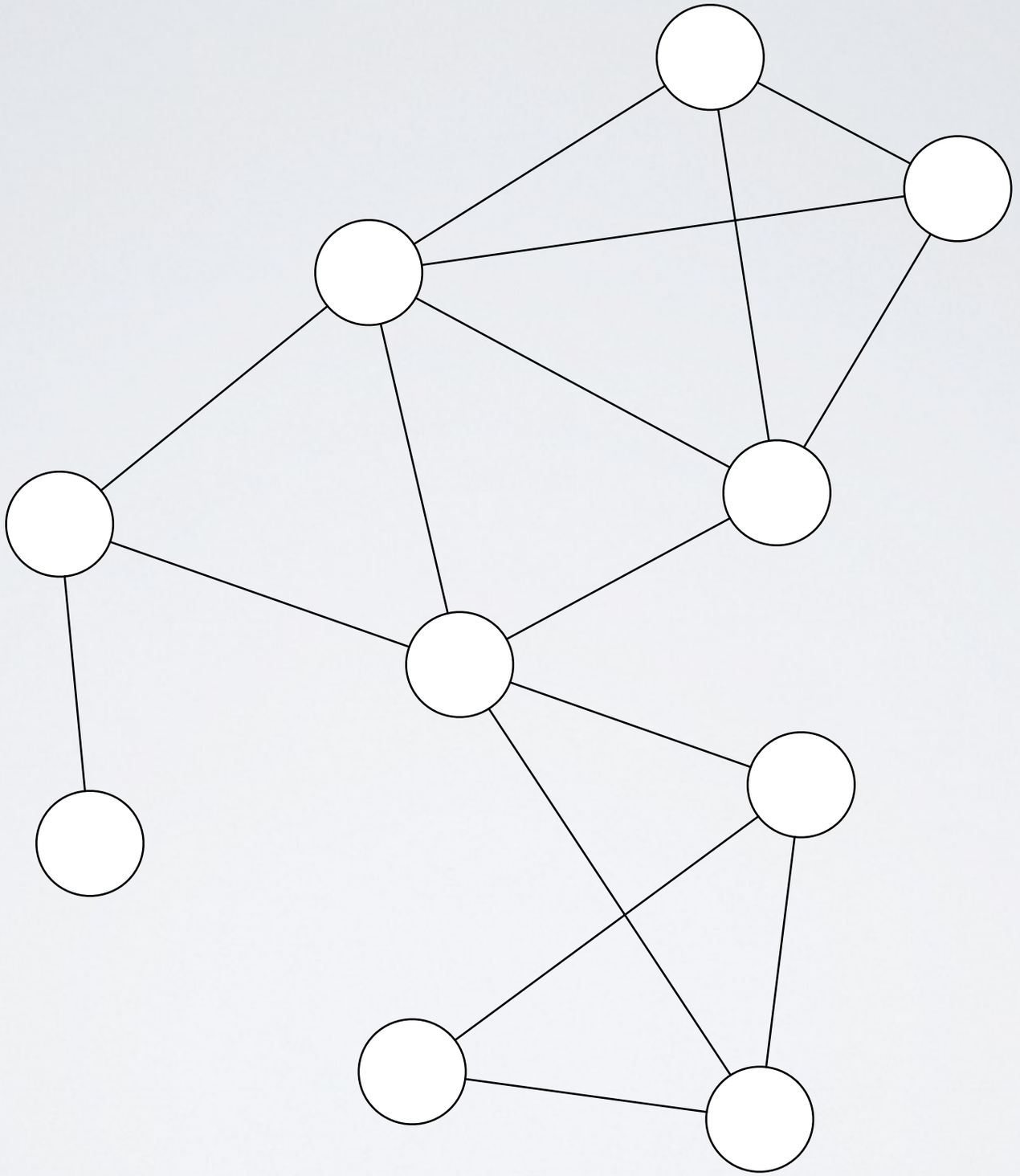
NETWORK TIME PROTOCOL (NTP)

- Time service over Internet - synchronizes clients with UTC:
- Reliability by mean of redundant servers and paths
- Scalable

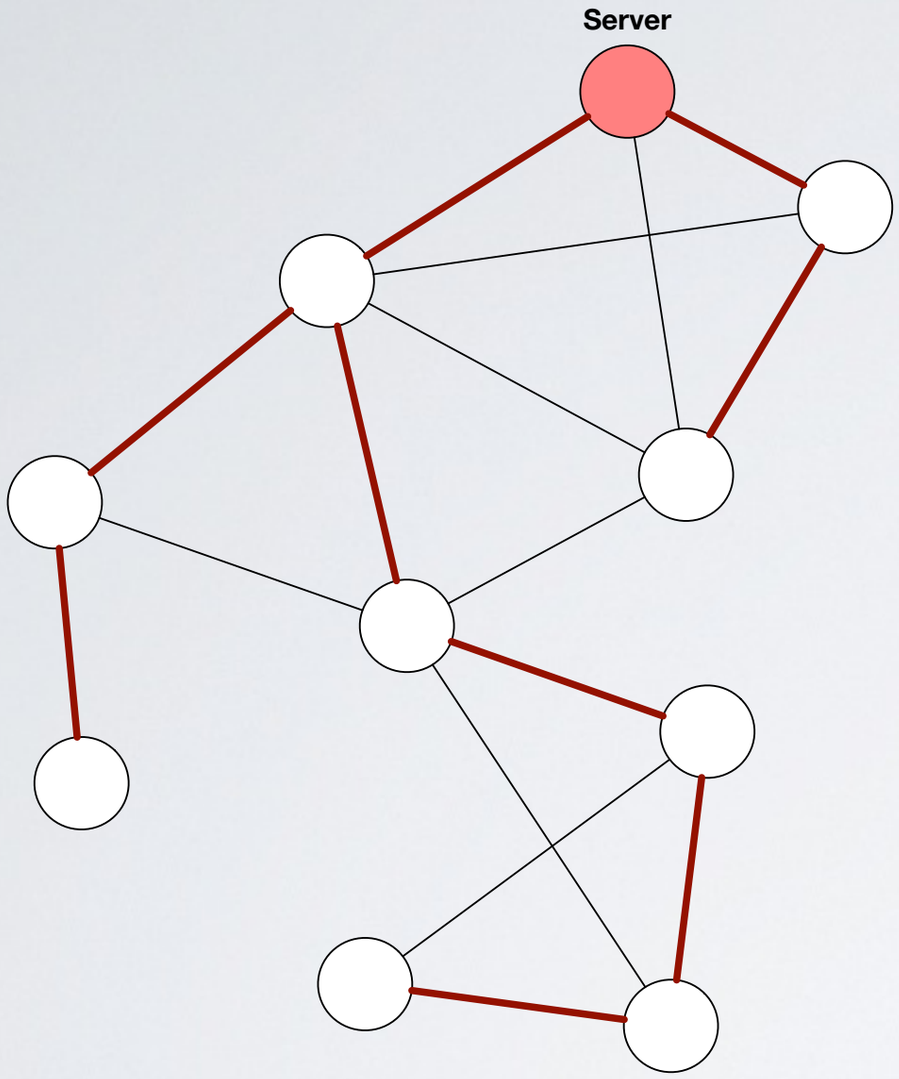


ARBITRARY GRAPHS

$G = (V, E)$

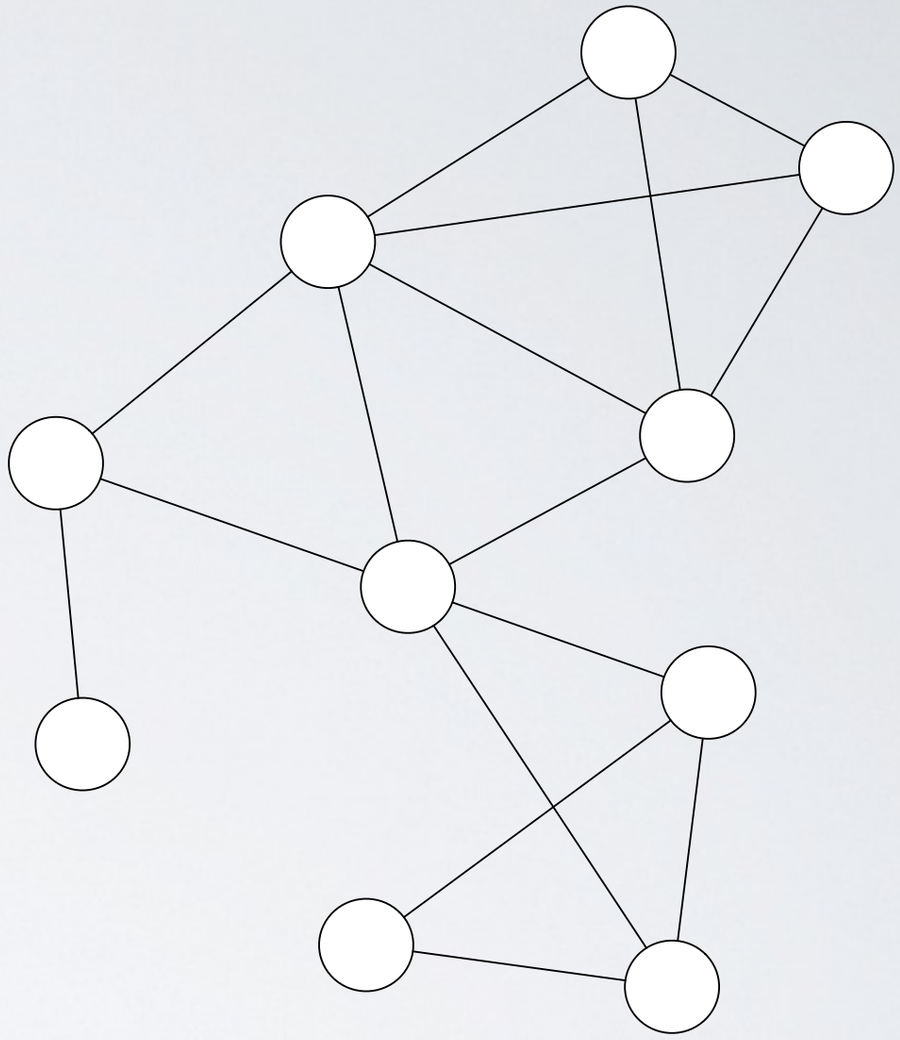


Tree like



Server
Imposes its clock

Fully distributed



Adjust the clocks
Based on your neighbours

GLOBAL SKEW VS LOCAL SKEW

Global skew:

- Maximum possible skew between two nodes in G.

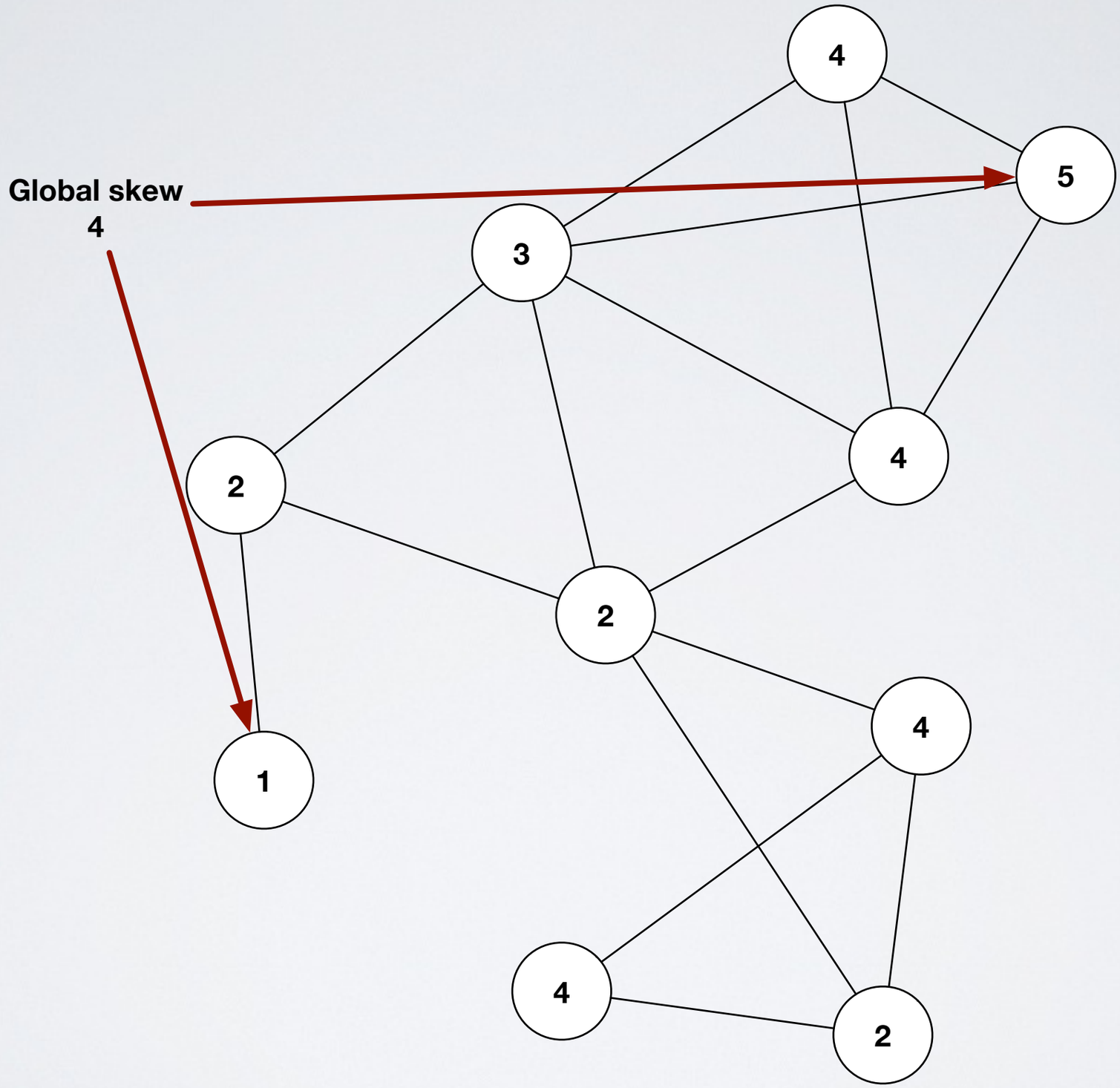
$$max_{t \in \mathbb{R}+}(max_{\forall (v,w) \in V \times V}(|C_v(t) - C_w(t)|))$$

Local skew:

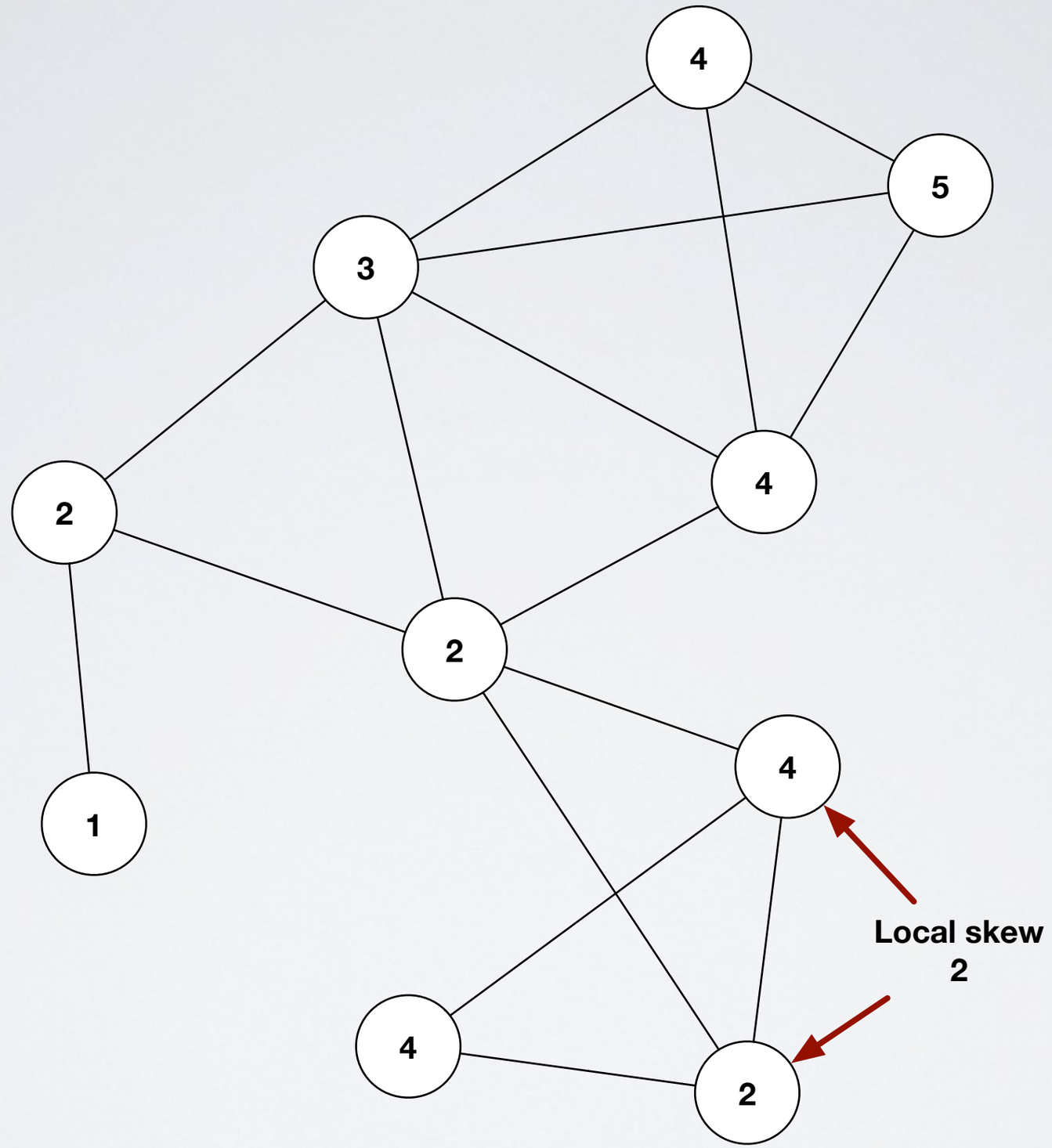
- Maximum possible skew between neighbours.

$$max_{t \in \mathbb{R}+}(max_{\forall (v,w) \in E}(|C_v(t) - C_w(t)|))$$

GLOBAL SKEW



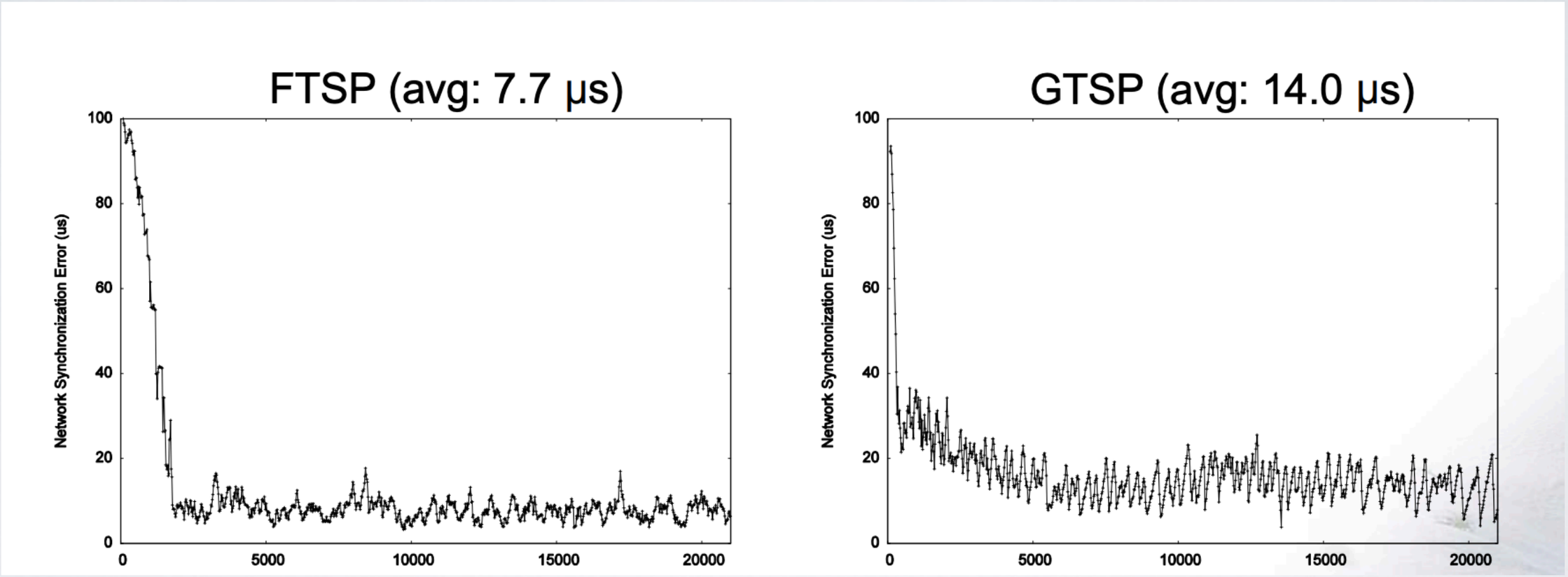
LOCAL SKEW



GLOBAL SKEW EXPERIMENTS

Tree Like

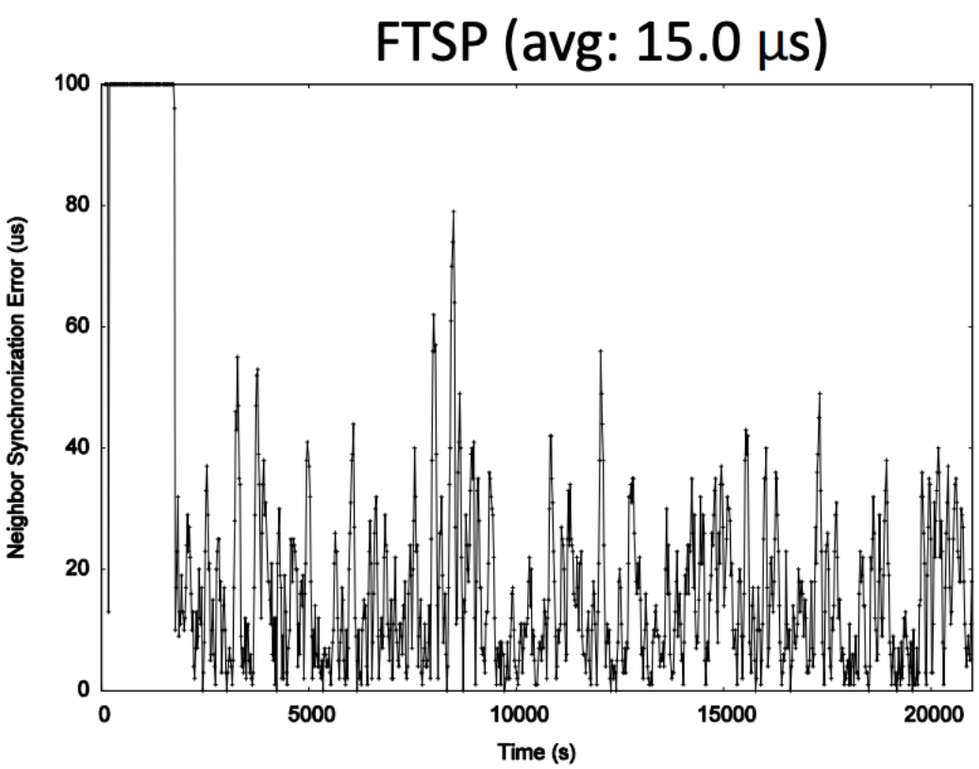
Fully distributed

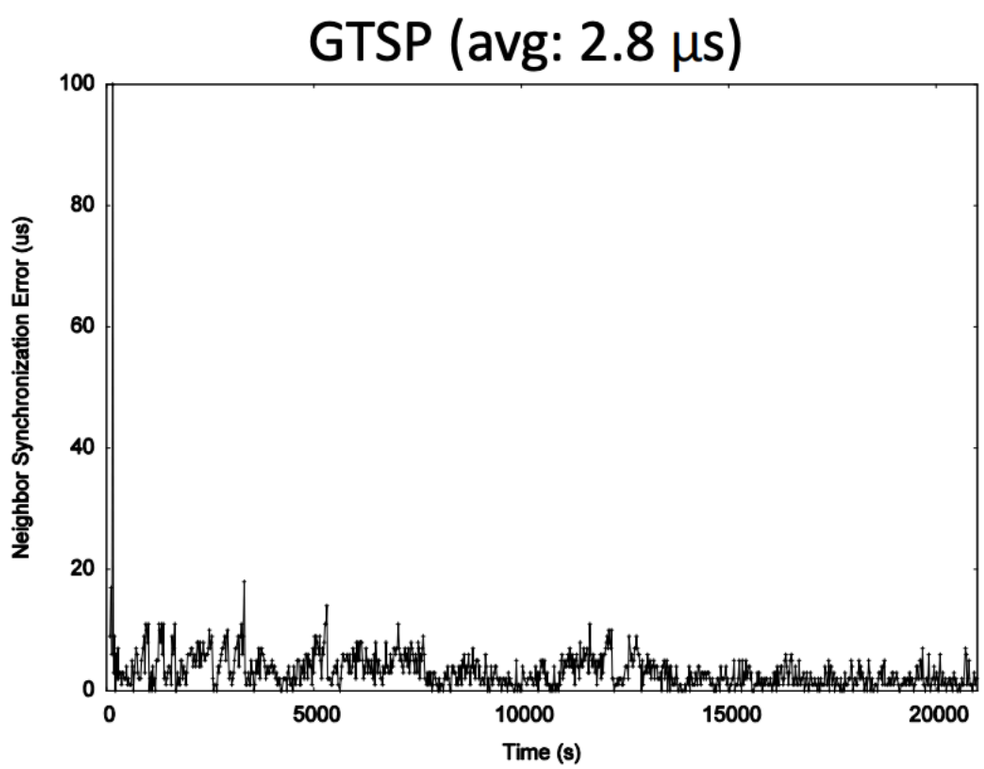


LOCAL SKEW EXPERIMENTS

Tree Like

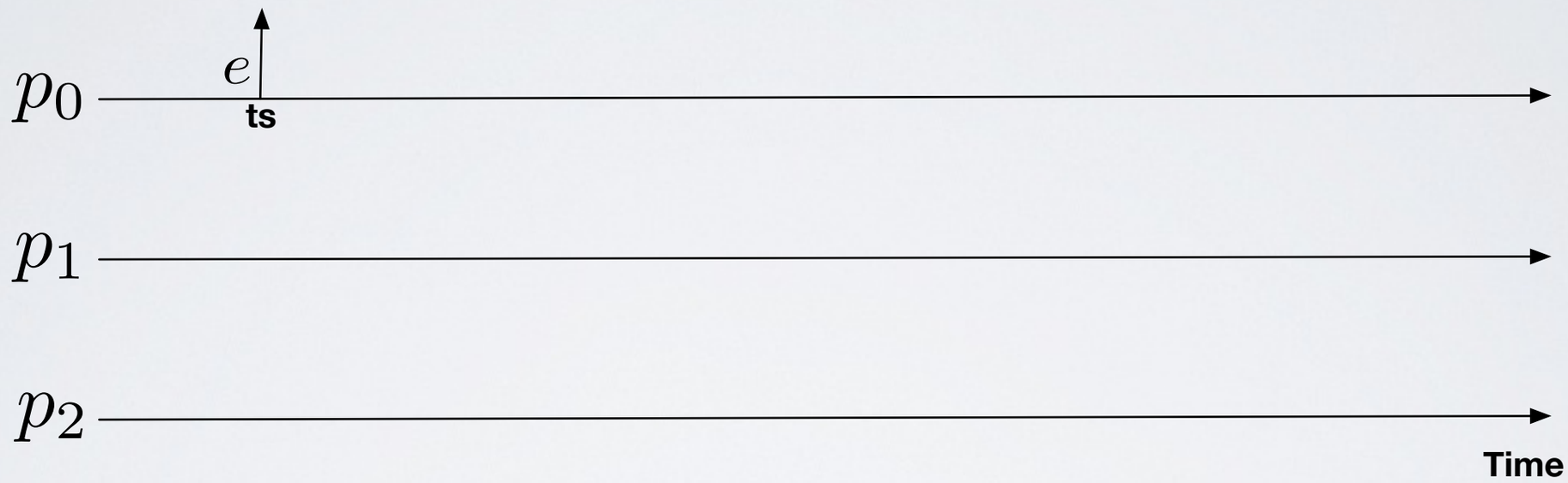
Fully distributed



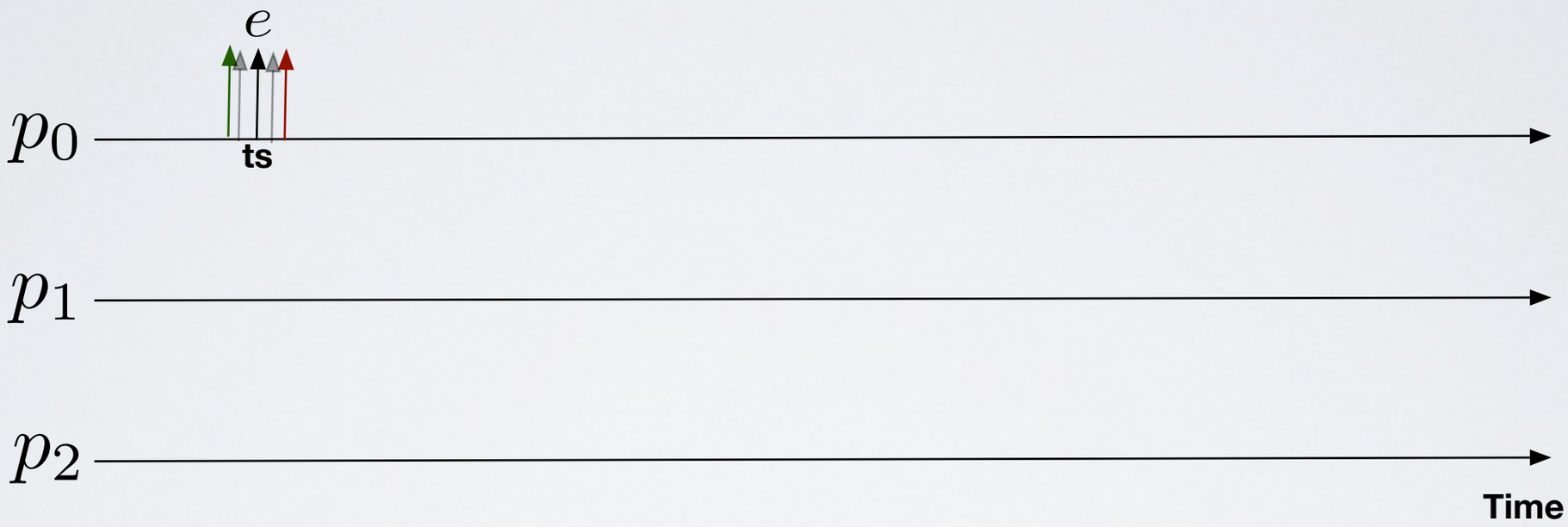


THE LIMIT OF BOUNDED ACCURACY

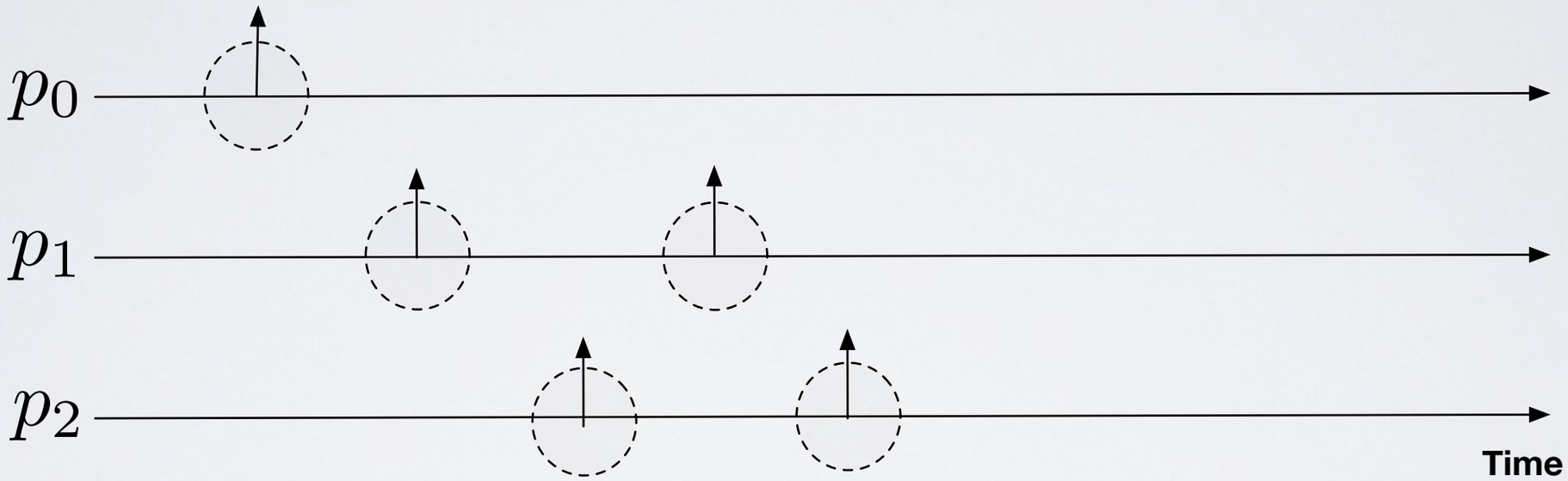
ORDERING WITH TIMESTAMPS



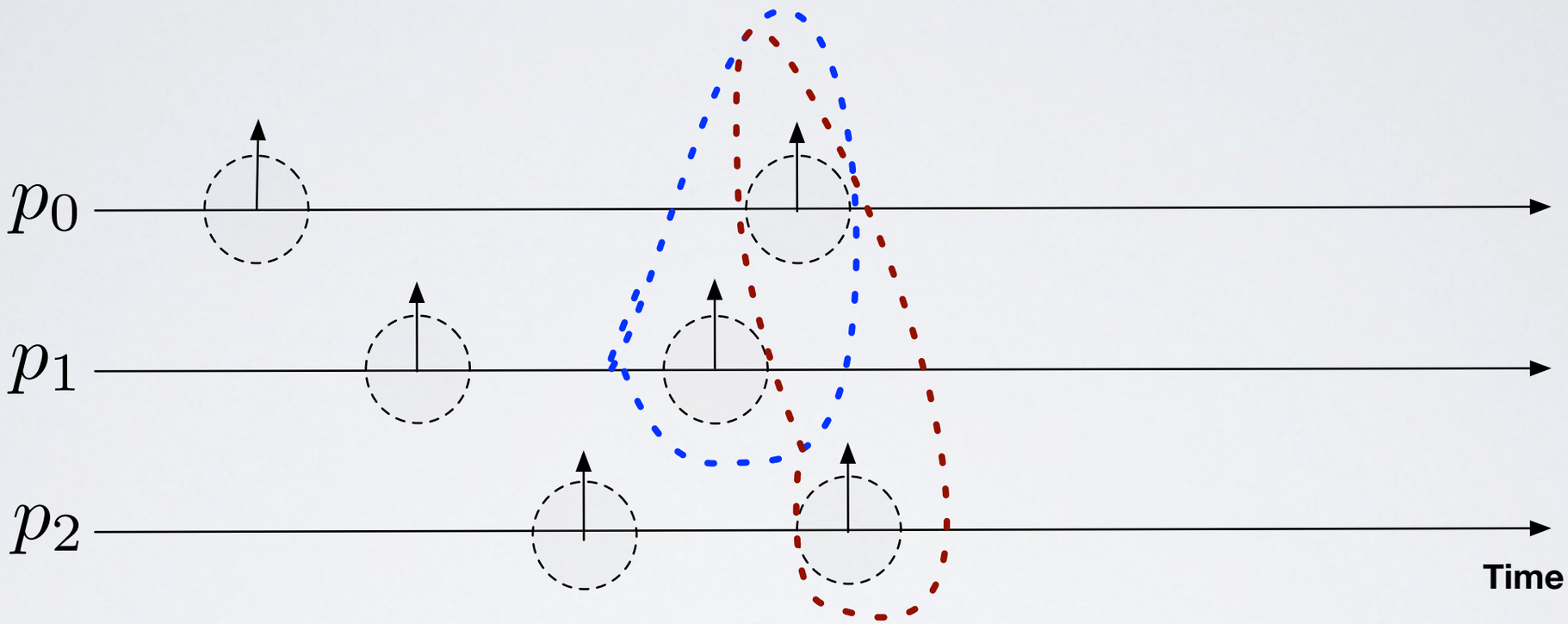
ORDERING WITH TIMESTAMPS



ORDERING WITH TIMESTAMPS

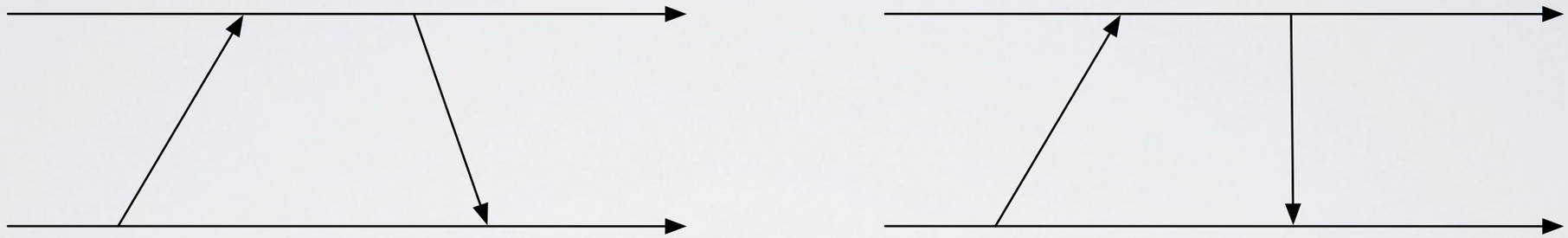


ORDERING WITH TIMESTAMPS



HOMEWORKS

Which of the following patterns is bad for Christian Algorithm:



HOMEWORKS

You are on a synchronous system, and you have a fair-lossy channel with the additional property:

- (Timed delivery) if p sends a message m to q at time t, and m is not lost, then q delivers t at time at most t+1.
- Using such channel could you create a stronger version of perfect point2point? **A version that gives you a bound on the delivery time?**