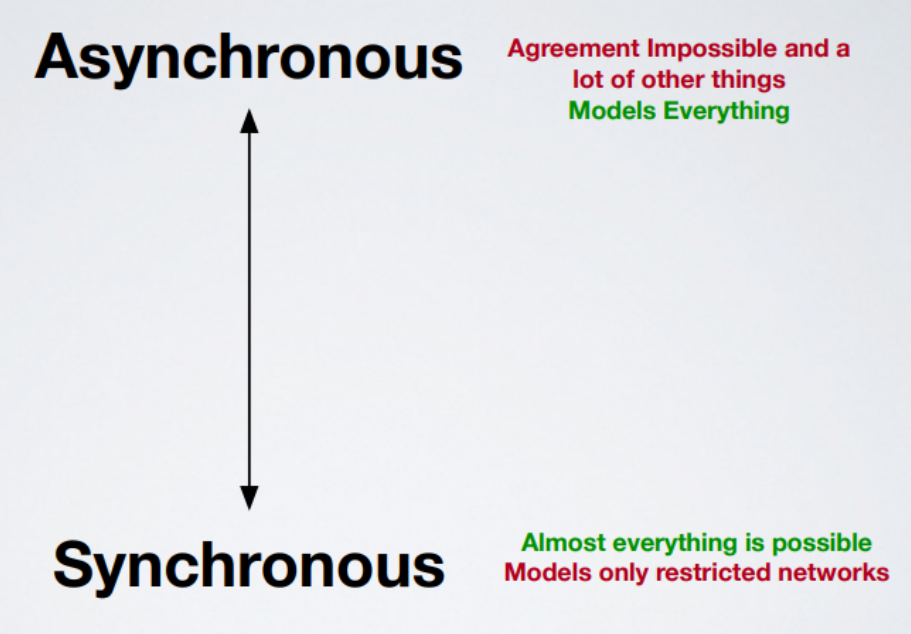
# 3. TIME, SYNCHRONIZATION AND ALGORITHMS

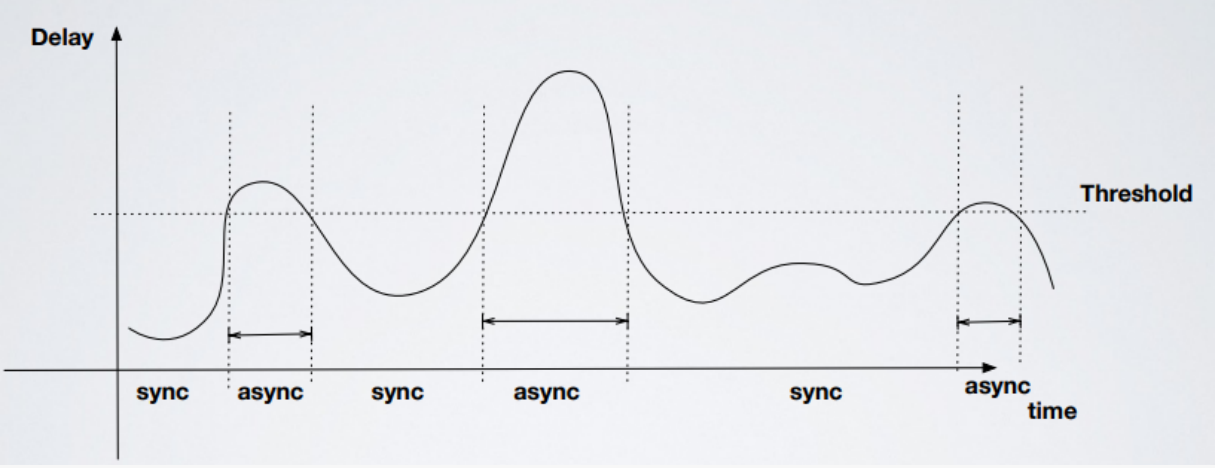
We can have 3 models of time:

- **Asynchronous**:
  - no bound on the delay that message can cumulate
- **Synchronous**:
  - known bound on the delay that a message can cumulate (if I send a message at time $t$ , it has to be recived by $t + \Delta$).
  - we can synchronize clocks
  - local executions steps happen at certain predeterminated interval, and that they take bounded time.
- **Eventually Synchronous**:
  - time that is a time $t$, unknown to us, after which the system will be synchronous



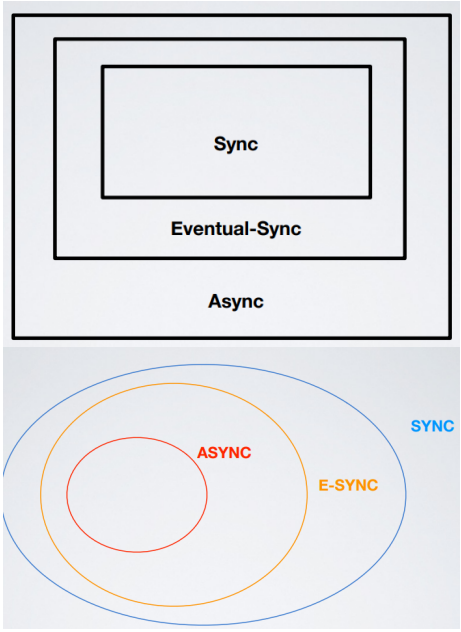

---

**Partial Synchrony**
a system that in certain point is sync and in some points is async



An algorithm with Safety property (ex: Mutex) is not good for partial synchrony sistems. There can be some points where the proprery is broken by async parts.

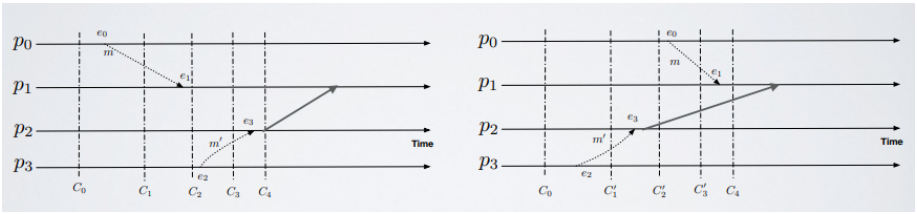
An algorithm with liveness property (ex: No-Starvation) can be used for partial synchrony sistems (There is a time $t$, unknown to us, after which the system is synchronous and will be synchronous forever).




---

## CLOCK SYNCHRONISATION

Time breaks the diagram of indistinguishability because time can order the events.




As we can see, an indistingushable diagram can be decifrated thanks to the time using:

- **Timestamps**: 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.
- **Naif solution**: each process timestamps events by mean of its physical clock.

**Physical clocks in computers**:
The clock is obtained by the operating system by reading a local hardware clock, they are made with an oscilator 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 ($C_i(t)$):

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

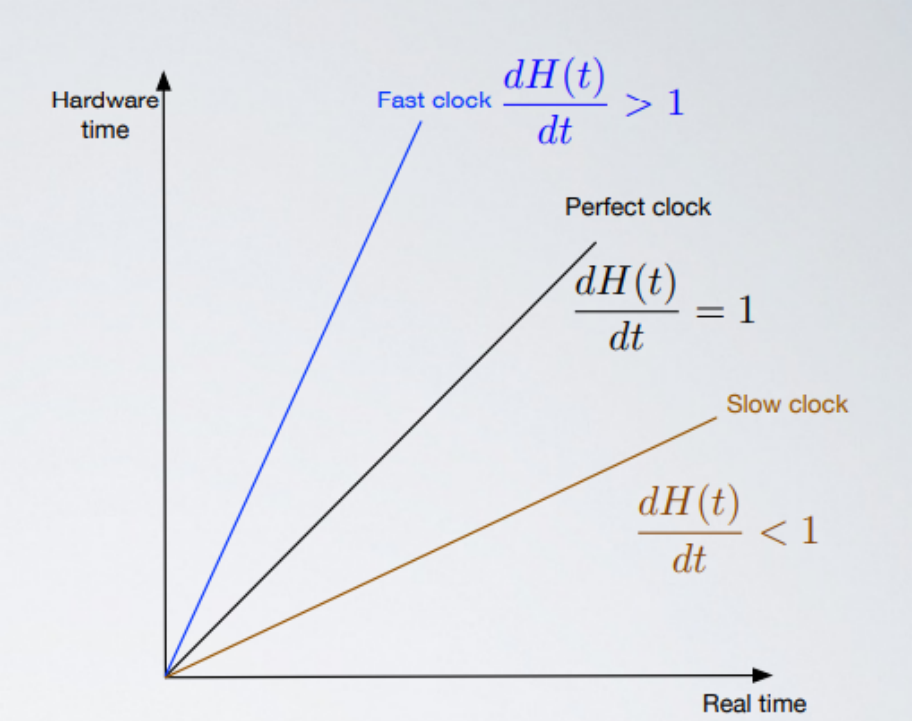
1. Different local clocks may have different values.

   - **Skew**: difference in time between two software clocks.

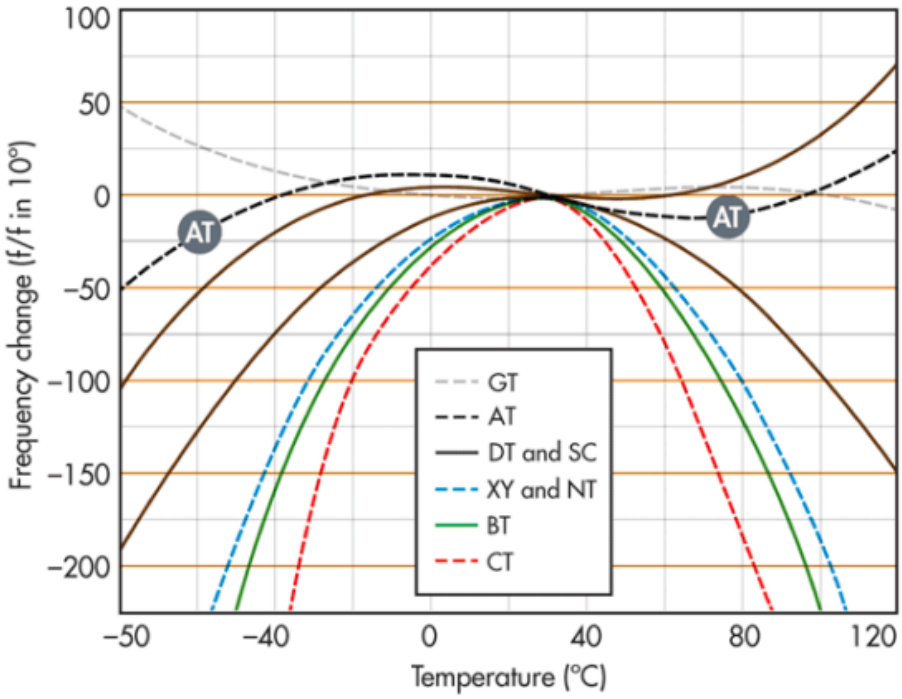
     $$Skew_{i,j}(t) = |C_i(t) - C_j(t)|$$

2. **Drift Rare**: gradual misalignment of synchronized clocks caused by the slight inaccuracies of the time-keeping mechanisms

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




A drift rate of 2 microsec/sec means clock increases its value of $1sec + 2microsec$ for each seconds.
Ordinary quartz clock derivate nearly by $1sec$ in 11-12 days. Hig precision quartz clocl dirft rate is $10^{-7}/10^{-8}$ ssec/sec.



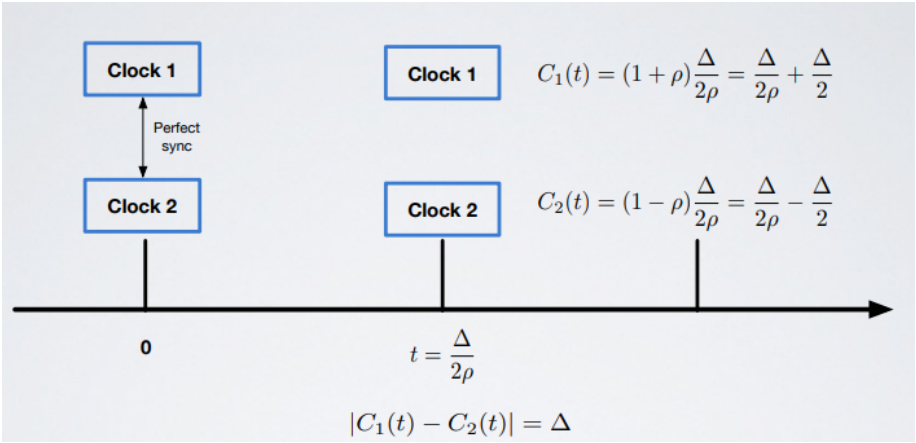

This image shows how difficult is to set a correct $\alpha$ because quartz can oscillate from slow bound to fast bound.

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

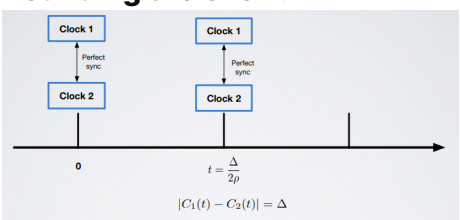
$$1 - \rho \leq \frac{dH(t)}{dt} \leq 1 + \rho$$

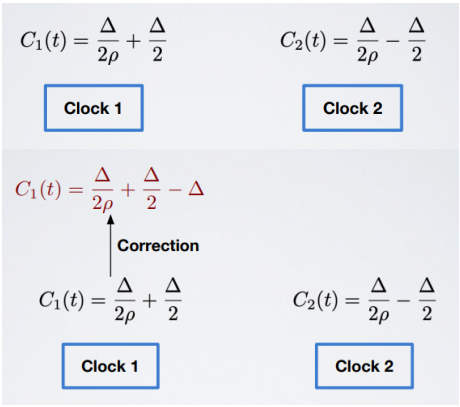
In presence of a correct hardware clock $H$ we can measure a time interval $[t, t']$ (for all $t' > t$) introducing only limited errors.

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




**Bounding the Skew**:






**Monotonicity** (don't go back in time NEVER).
software clocks have to be monotone:
$t' > t \rightarrow C(t') > C(t)$

The monotonic property can br guaranteed choosing opportune values for $\alpha$ and $\beta$.
note that $\alpha$ and $\beta$ can be a function of time:

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

So to have a situation where two pc have the same clock is **slow down one and speed up the other one**.

Slow down means:

- not possible to impose a clock value in past so 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.

---

## UNIVERSAL TIME COORDINATED (UTC)

Based on international atomic time ($1sec$).
UTC-signals come from shortwave radio broadcasting stations or from satellites (GPS) with an accuracy of:

- $1\,msec$ for broadcasting.
- $1\,\mu sec$ for GPS.

**Extenal Synchronization**:

- process synchronize their clock $C_i$ with an authoritative external source $S$.
- Let $D > 0$ be a synchronization bound and $S$ be the soure of the 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 clock $C_i$ are **accurate** within the bound of $D$.

**Internal Synchronization**:

- all the processes synchronize thei clocks $C_i$ among them
- let $D > 0$ be a synchronization bound an let $C_i$ and $C_j$ be the clock at any two processes $p_i$ and $p_j$
- clocks are **internally synchronized** in a time interval $I$:

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

  We say that clock $C_i, C_j$ **agree within the bound of $D$.
- Internal Synchronization implies External Synchronization but the opposite isn't true.
- 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.

---

## SYNCHRONIZATION ALGORITHMS

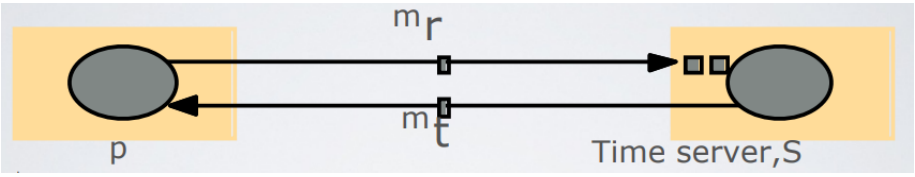
There are **3 algorithms** used of time synchronization:

### Christian's Algorithm

Uses a **time server** $S$ that recives a signal from an UTC source.

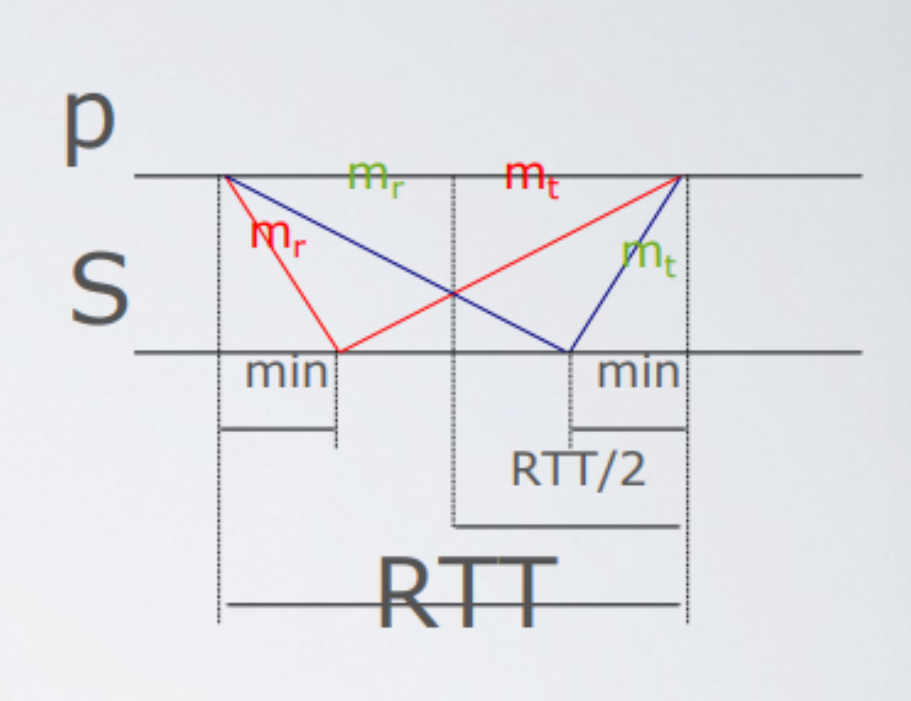
- is based on the measurement of message $RTTs$ (Round-Trip Times).
- synchronization is reached only if $RTTs$ are small with respect to the required accuracy.

A process $p$ asks the current time through a message $m_r$ and receives $t$ in $m_t$ from $S$.
$p$ sets is clock to $t + T_{round}/2$ where $T_{round}$ is round trip time measured by $p$




- a time server can crash
- cristian's algorithm suggests to use a cluster of synchronized time server.
- a time server can be attacked

**If the delay is equal in direction $p \to S$ and $S \to p$, then this algorithm works in a perfect way**. But if the message from me to the server is very fast and the message from the server to me is slower, so in real life it doesn't work properly.




1. 
   - Reply time is greater than estimate $(RTT/2)$.
   - Assume it is equal to $RTT$-$min$, we have:

     $$\Delta = RTT/3 - (RTT - min) = \frac{-RTT + 2\,min}{2} = -\frac{RTT}{2} + min = -(\frac{RTT}{2} - min)$$

     it means that the algorithm will set $p$ into the past.
2. 
   - Reply time is smaller than estimate $(RTT/2)$.
   - Assume it is equal to $min$, we have:

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

     it means that the algorithm will set $p$ into the future.

We can correct that using **several servers** and adding **redundancy**.

## Barkeley's Algorithm

Uses a master-slave structure.
Works in 2 steps:

- gathering of all the clocks from other processes and computation of the difference.
- computations of the correction.

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 $t2$ (local clock value).

When the master receives the reply message it reads the local clock ($t3$) and computes the difference between the clock.

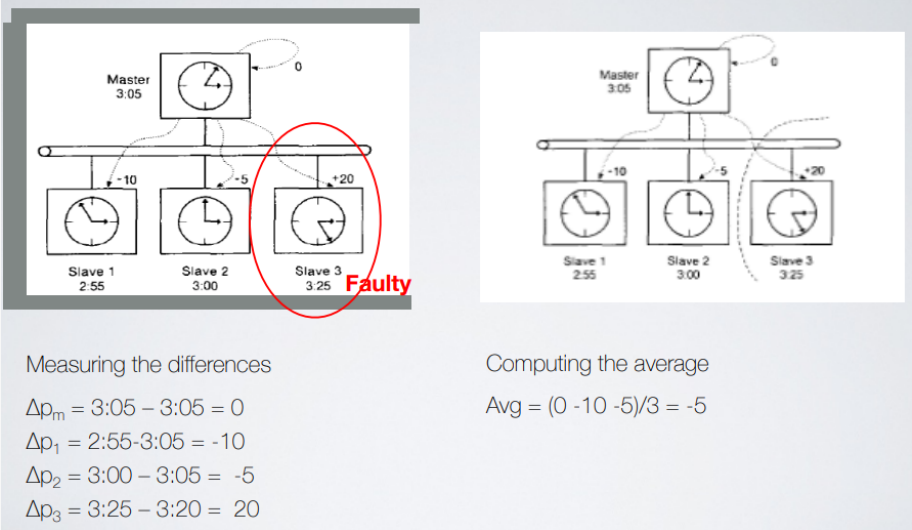
**Master Behavior**:

- computes of the differences $\Delta p_i$ between the master clock and the clock of every other process $p_i$ (including the master itself)
- computes the average $avg$ of all $\Delta p_i$ without considering faulty processes (processes that have a clock wich differ from the one of the master more than a given threshold $\gamma$).
- Computes the correction for wach process (including faulry processes):

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

**Slaves Behavior**:

- each process recives the correction and applies it to the local clock.
- if the correction is a negative valure, it slows down its clock.

> Example:
>
> 
> 
> the third slave is discarded because it has the most distant value compared to the others (standard deviation is often used to calculate this).

**Accuracy**
The accuracy depends on the acimum round-trip time (the master does not consider clock values associated to $RRT$ grater than the macimum one).

**Fault Tollerance**:

- If the master crashes, another master is elected (in an unknown amount of time).
- It is tolerant to arbitrary behavior (ex: 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 threashold.

## Network Time Protocol (NTP)

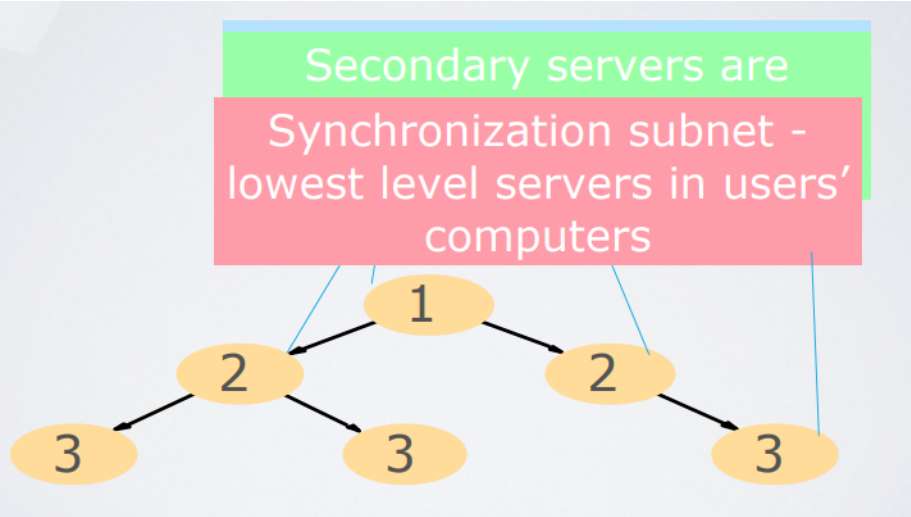
Currently used.

- time serveice over Internet - synchronizes clients with UTC:
- reliability by mean of redundant servers and paths
- scalable

NTP is a protocol used to synchronise computer clocks on a network. Here are the basic concepts:

1. **Time Synchronization**: NTP aligns a device's clock with reference servers that provide accurate time such as that provided by atomic clocks or GPS satellites.

2. **Stratification** (Stratum): NTP servers are organised in layers (called "**stratums**"). Stratum $0$ servers provide the exact time (e.g. atomic clocks), while higher level servers (Stratum $1$, $2$, ...) obtain the time from those below them.




3. **Delay correction**: NTP calculates the network delay between clients and servers to adjust the time accurately, taking into account the time it takes for packets to travel through the network.

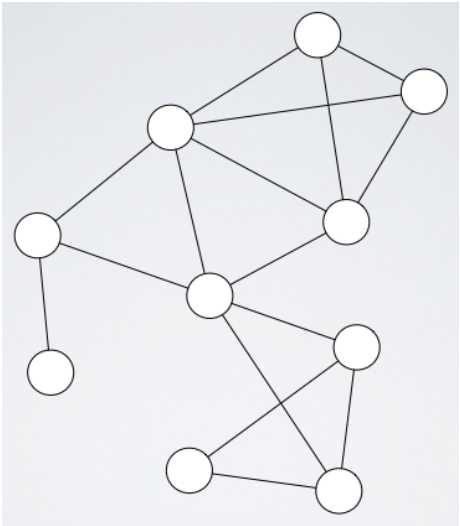
4. **Reliability**: NTP uses multiple reference sources and discards any unreliable values to ensure synchronisation accuracy.

In summary, **NTP enables devices to keep the correct time reliably and automatically**, which is essential for applications that require precise time synchronisation.

---

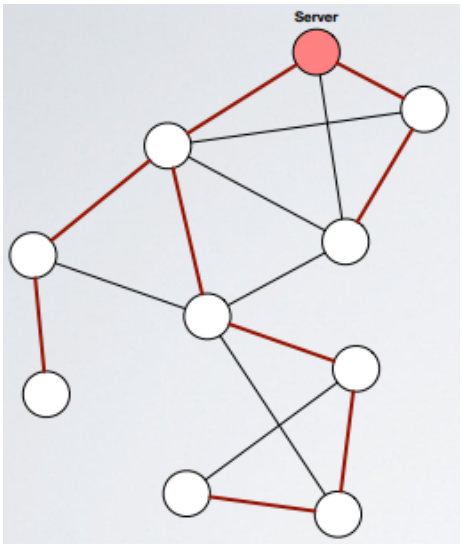
## ARBITRARY GRAPHS

$G = (V, E)$ is an arbitrary graph

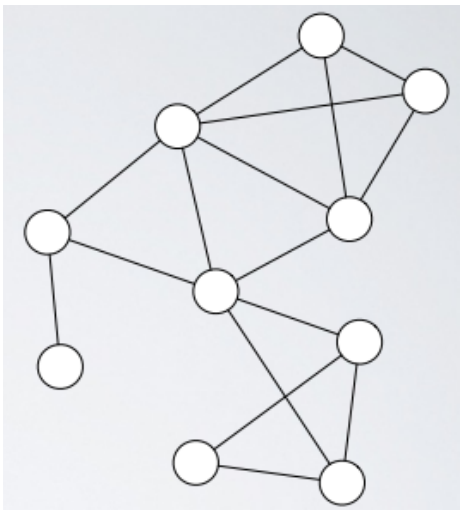


W have **2 main protocols**:

- **Tree Like**: server imposes its clock




- **Fully Ditributed** : adjust the clock based on you neighbours



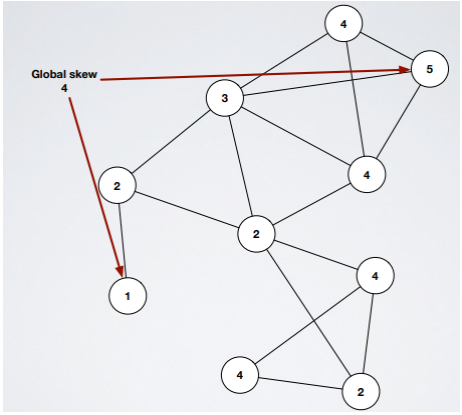
Different performances in case of Global Skew (difference in all the graph) and Local Skew (difference between two neighbours)

**Fully Distributed minimize the local skew** (look at the path from a neighbor to another one)
**Tree Like minimize the global skew**

Formal definition of global and local skew:

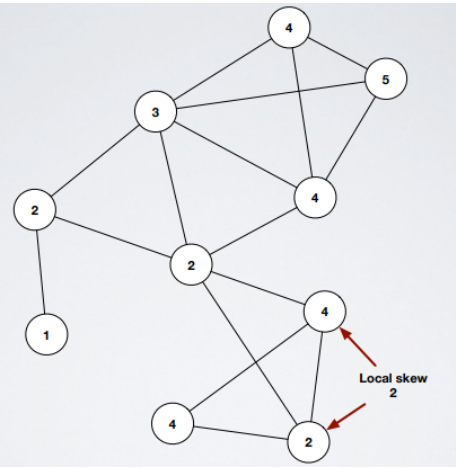
- **Global Skew**:

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

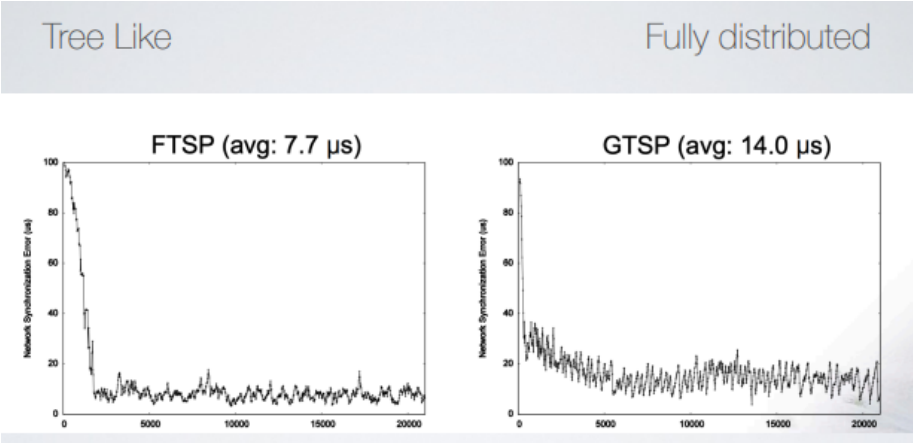


- **Local Skew**:

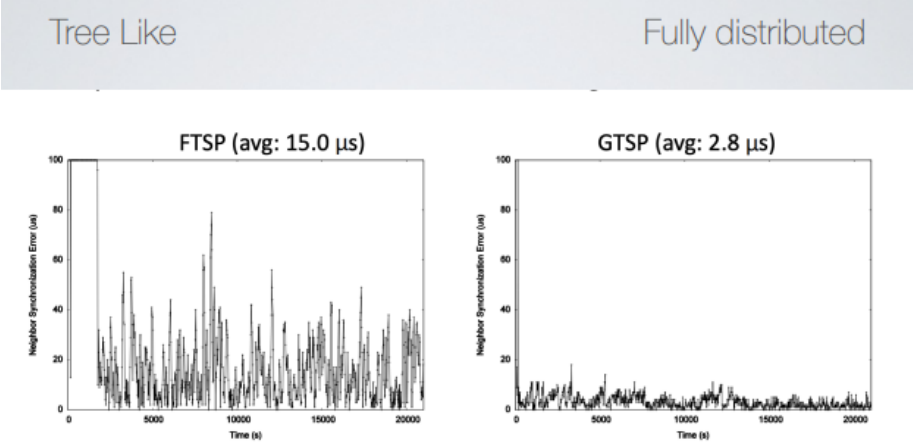
$$max_{t \in R^+}(max_{\forall (v,w) \in E}(C_v(t) - C_w(t)|))$$




**Global Skew Experiments**:



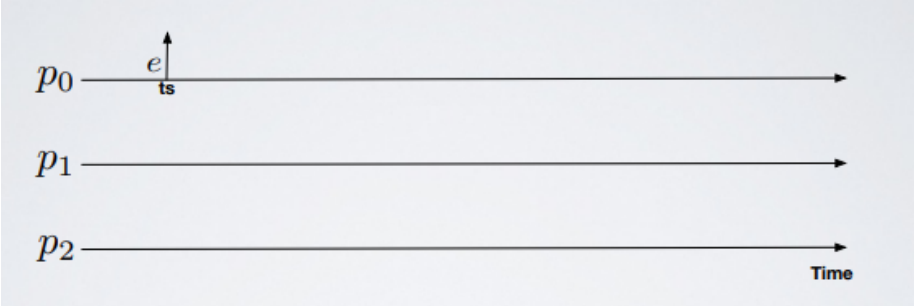

**Local Skew Experiments**:



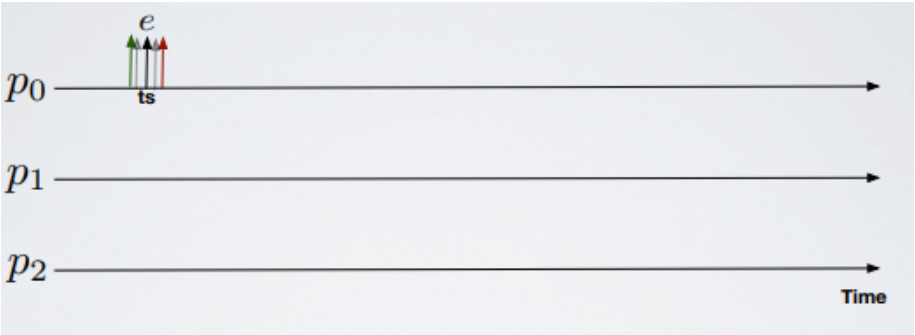

---

## THE LIMIT OF BOUNDED ACCURACY

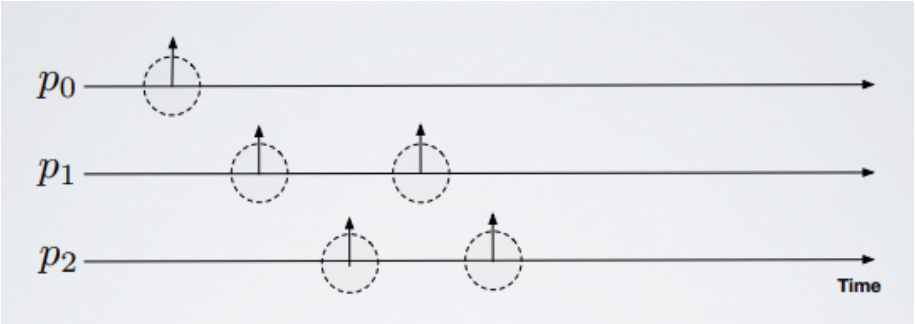
We can use **timestamps for an event**.




In real life we must consider a bound for a timestamp:




In the following case we can say that this metodology is correct




But we can't say the same thing in that other case because the timestamp are overlayed