# Tempo em Sistemas Distribuídos

# Relógios Físicos

# Relógios Físicos

- A noção de tempo é usada frequentemente nos sistemas
  - Protocolos de autenticação
  - Data de criação/ alteração nos sistemas de ficheiros
    - Esta informação é usada pelo *make* para decidir quais os ficheiros a re-compilar
  - Medidas de desempenho
  - Timeouts
  - etc

# Relógios Físicos

- Cada computador tem um relógio interno
- Idealmente estes relógios não se desviariam de uma referência universal de tempo, por exemplo o “Coordinated Universal Time (UTC).
- Na realidade, os relógios sofrem desvios (*drift*):
  - Relógios de quartzo correntes podem sofrer um desvio de 1 seg em 11-12 dias ($10^{-6}$ segs/seg)
  - Relógios de quartzo de alta precisão podem ter desvios menores ($10^{-8}$ segs/seg) mas também se desviam

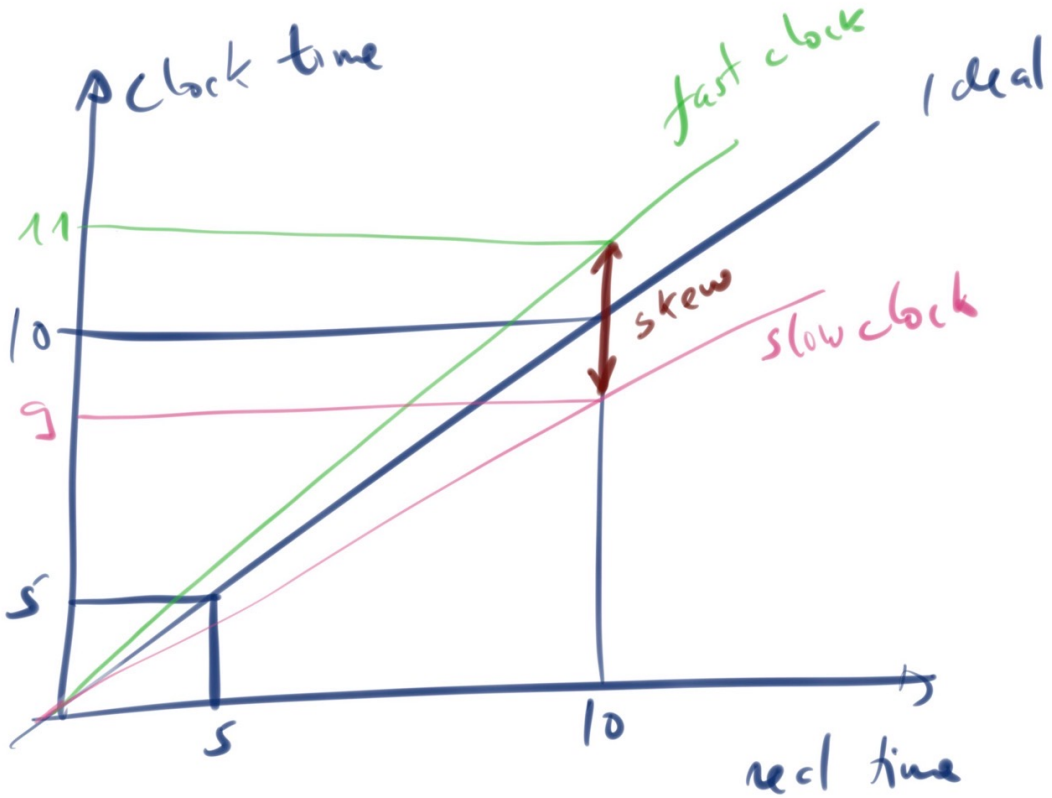
# Relógios Físicos

- Mesmo que num dado instante, todos os relógios tivessem o mesmo valor, com o passar do tempo, apresentarias diferenças entre si (*skew*).
- Isto pode levar a que as marcas temporais violem as leis da física:
  - Por exemplo, o valor do relógio no emissor de uma mensagem pode estar no futuro em relação ao valor do relógio do receptor: a mensagem aparenta ter sido recebida antes de ter sido enviada!

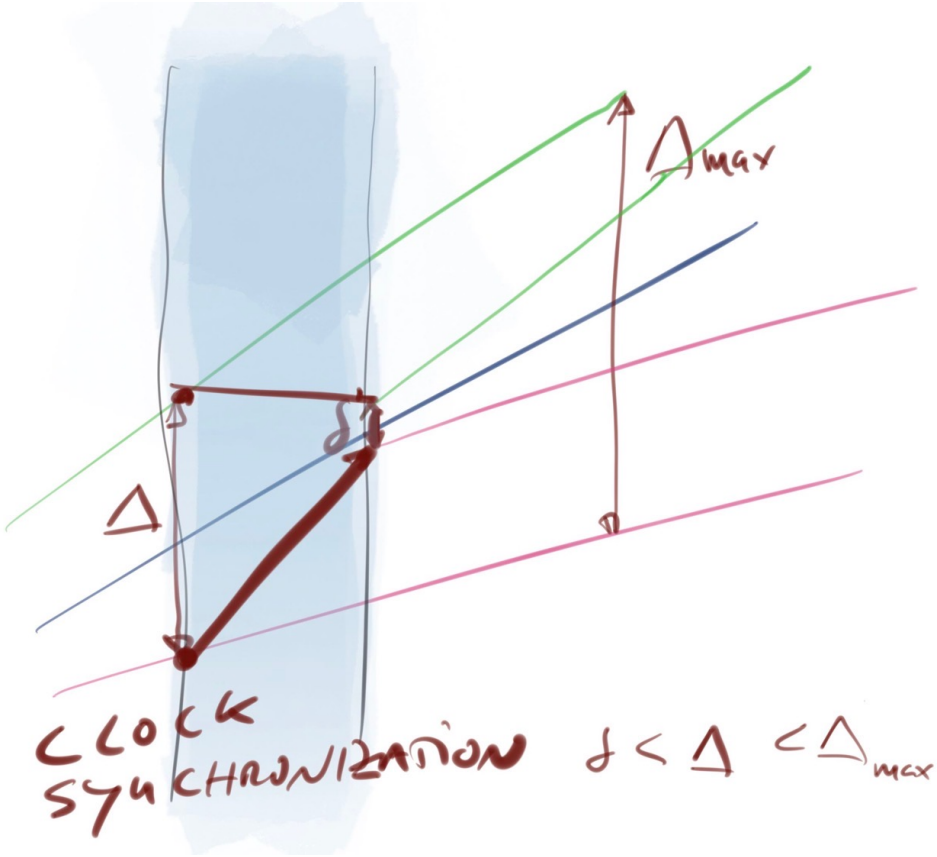
# Sincronização de Relógios

- Mecanismo que evita que a diferença entre os relógios aumente de forma ilimitada.
- Os nós trocam mensagens entre si e ajustam os seus relógios de forma a reduzir a diferença dos seus valores
- Isto é feito periodicamente

# Sincronização de Relógios



# Sincronização de Relógios



# Sincronização de Relógios

- Duas fases:
  - Função de convergência (fase 2)
    - Função que um nó usa para, com base no valor do seu relógio e no valor estimado de um ou mais relógios de outros nós, fazer um ajuste ao valor do seu próprio relógio
  - Isto pressupõe que existe um algoritmo para ler um relógio remoto (fase 1)
    - Que permite a um nó estimar o valor do relógio de outro nó

# Sincronização Interna e Externa

- Sincronização externa
  - Quando existem um ou mais nós com acesso a uma fonte de tempo “real”
- Sincronização interna
  - Os nós tentam reduzir a disparidade dos valores que os seus relógios apresentam sem terem acesso a nenhuma fonte externa
    - Podem estar sincronizados entre si mas desfasados do tempo universal

# Tempo Universal Coordenado (UTC)

- International Atomic Time é baseado em relógio de grande precisão (drift rate $10^{-13}$)
- UTC é um standard internacional baseado em relógios atómicos e ocasionalmente ajustado de acordo com medidas astronómicas
- É difundido através de sinal rádio terrestres e a partir de satélites (e.g. GPS)
- Computadores podem ter receptores que lhes permitem sincronizar os seus relógios com o UTC
- Sinais rádio de estações terrestres têm precisão entre 0.1-10 millisegundos
- Sinais de satélite (GPS) tem precisão de cerca de 1 microsegundo

# Sincronização Externa

**Sem tolerância a faltas**

- Existe um processo com acesso ao UTC
- Os outros processos lêem o valor do relógio desse processo
- Atrasam ou adiantam o valor do seu relógio para acompanhar a fonte externa

# Sincronização Externa

**Com tolerância a faltas**

- Considere-se, por exemplo, o caso em que f processos podem falhar, fornecendo valores errados do relógio (mas sem enviarem valores diferentes para nós distintos).
- Configuram-se 2f+1 processos com acesso ao UTC
- Ou outros precessos lêm o valor do relógio desses processos
- Descartam os f valores mais altos e os f valores mais baixos. Escolhem o valor que sobra

- Nota: há outras alternativas, isto é apenas um exemplo

# Sincronização Interna

**Sem tolerância a faltas**

- Cada processo lê o valor de todos os outros processos
- Aplica uma função a todos os valores, incluindo o valor do seu relógio (por exemplo, a média)
- Atrasa ou adianta o valor do seu relógio para acompanhar o resultado da função

# Sincronização Interna

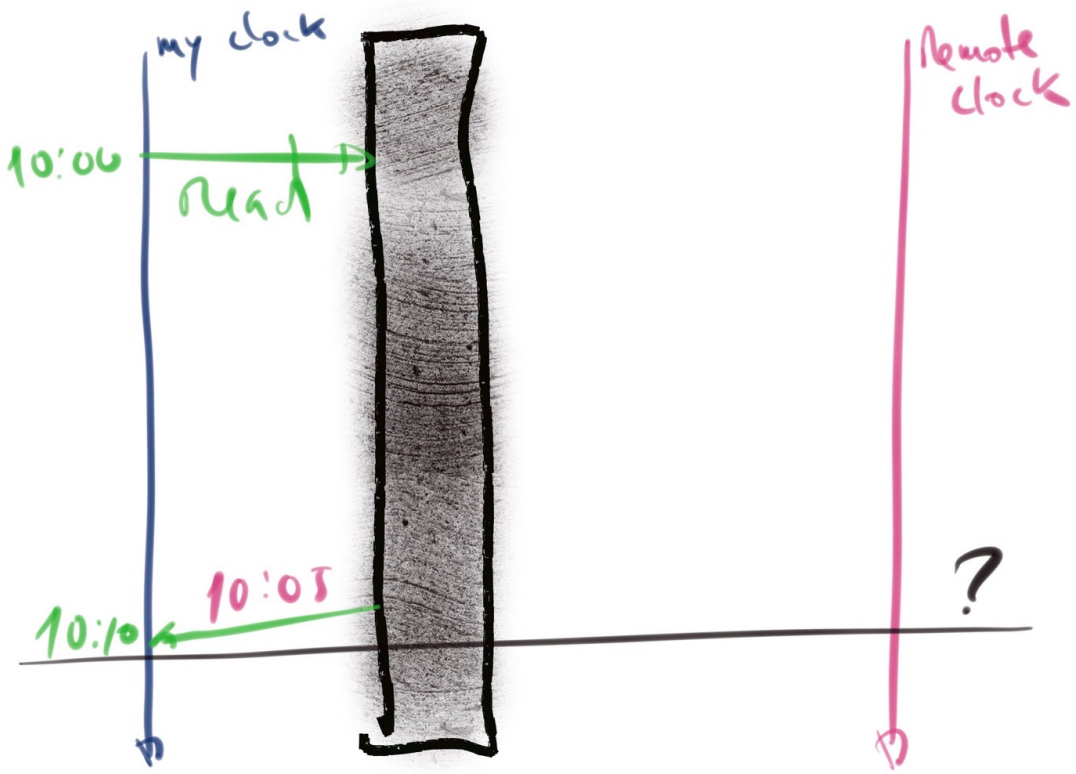
**Com tolerância a faltas**

- Cada processo lê o valor de todos os outros processos
- Descarta alguns dos valores lidos
  - Quais?
- Aplica uma função aos valores restantes, incluindo o valor do seu relógio (por exemplo, a média)
- Atrasa ou adianta o valor do seu relógio para acompanhar o resultado da função

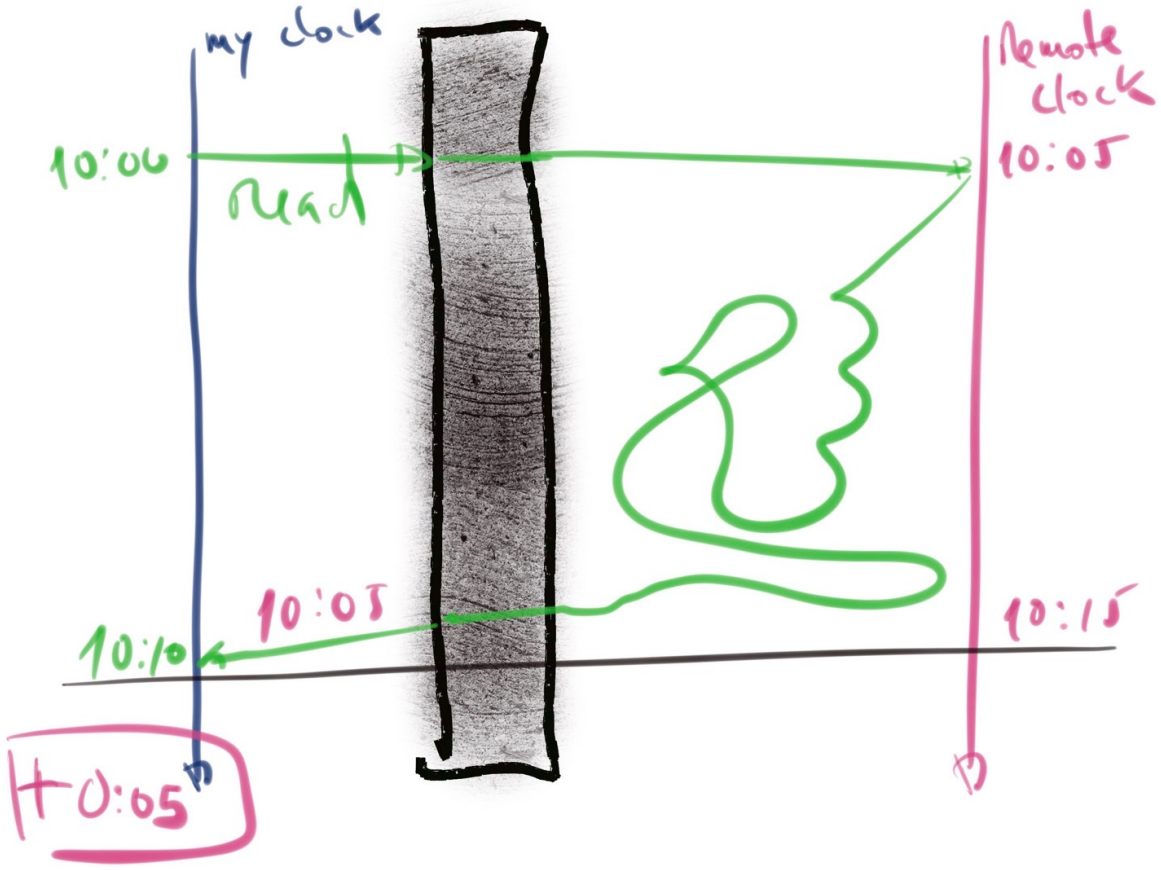
# Leitura Remota de Relógios

- Os mecanismos anteriores pressupõem que um nó consegue estimar o valor do relógio de outro nó
- Obter esta estimativa é, por si só, um desafio

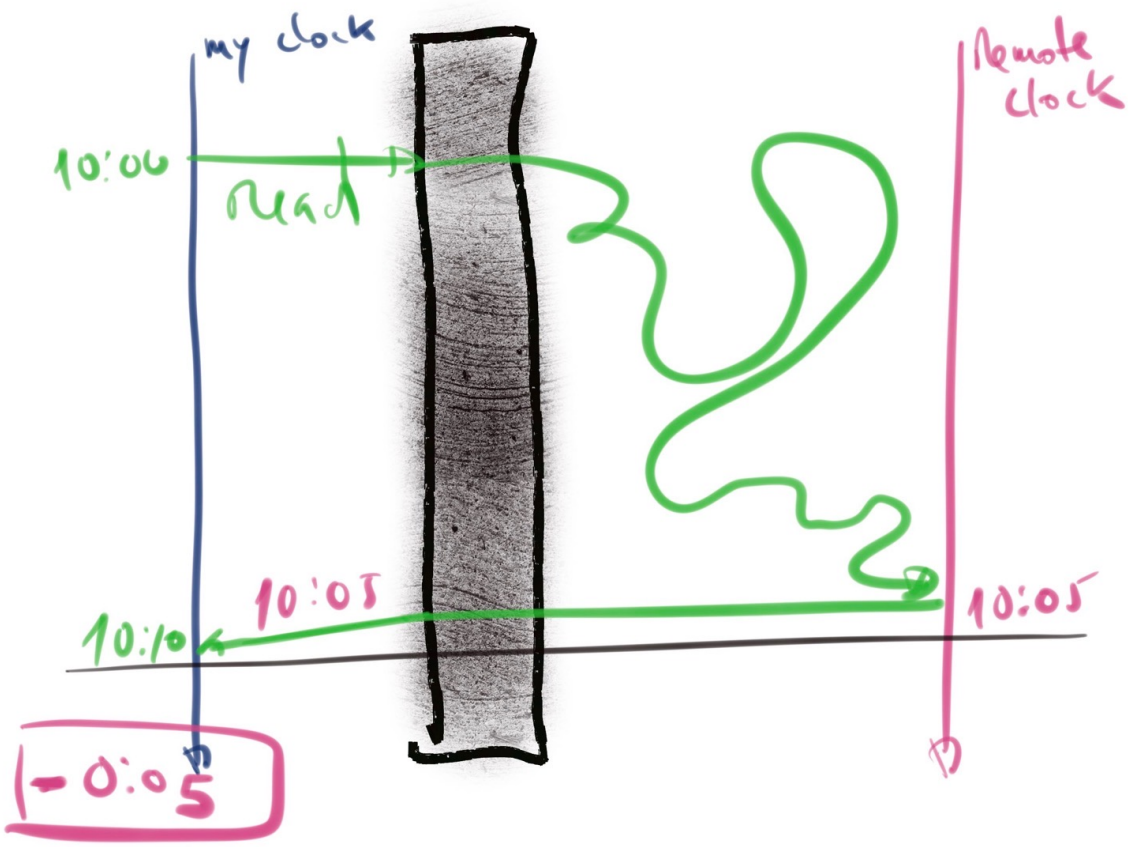
# Leitura Remota de Relógios



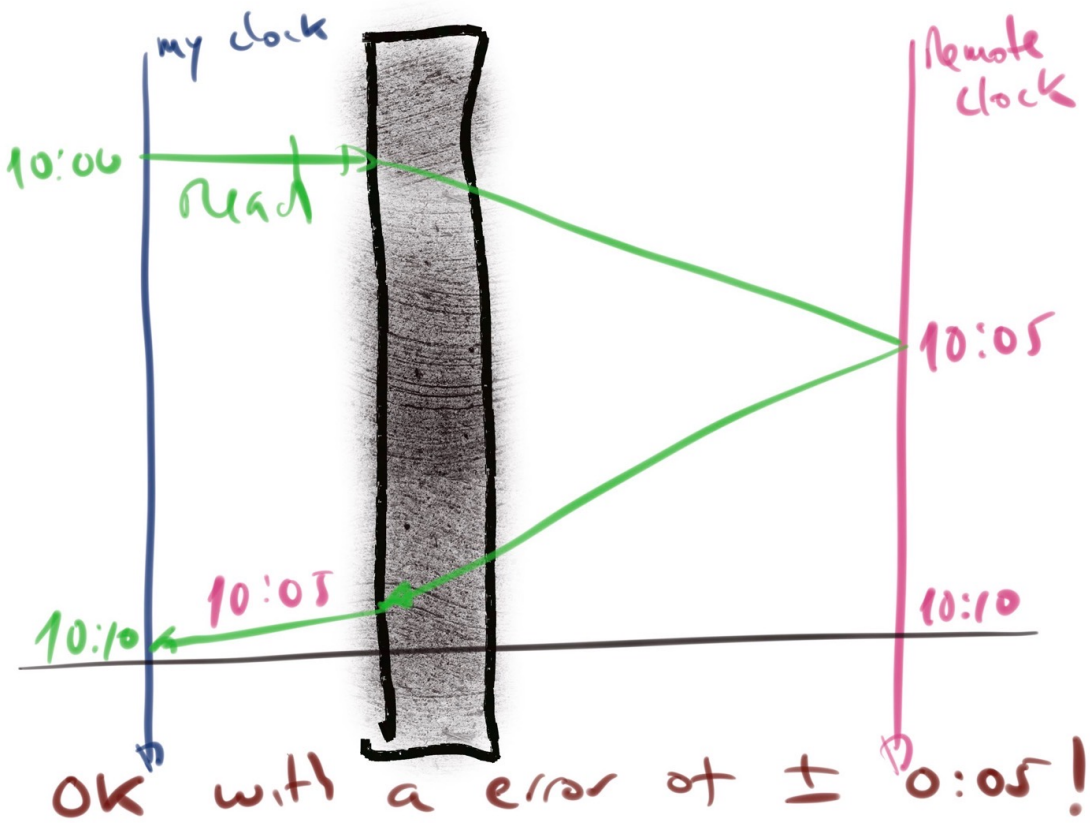
# Leitura Remota de Relógios



# Leitura Remota de Relógios



# Leitura Remota de Relógios



# Leitura Remota de Relógios

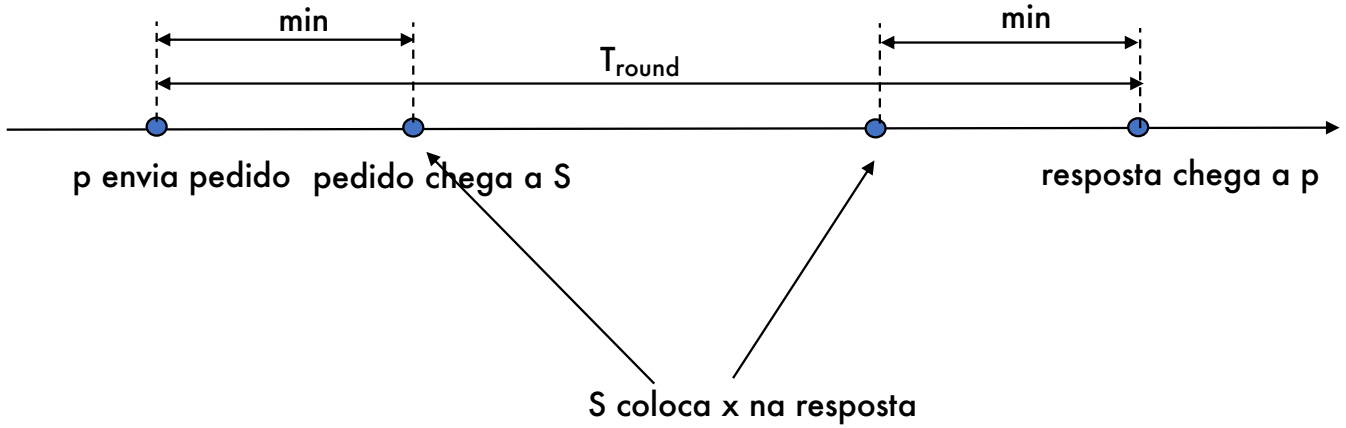
- Vamos assumir que p1 quer ler o valor do relógio de p2
- O processo envia no instante t1 um pedido de leitura ao processo p2.
- O processo p2 recebe o pedido no instante t2. Lê o valor do seu relógio (seja este valor X) e envia este valor ao p1 no instante t3.
  - Nota, se este passo for muito rápido t2 ≅ t3
- O processo p1 recebe o valor X no instante t4. A mensagem de resposta demorou “message_delay=t4-t3” unidades na rede.
- O processo p1 estima que o valor do relógio no processo p2, no instante p4, é “X+message_delay”

# Leitura Remota de Relógios

- Qual é o problema do esquema anterior?
  - O processo p1 não tem maneira de saber quando é que a mensagem de resposta foi enviada (isto é o valor de t3).
  - Portanto não consegue calcular o “message_delay” de forma precisa
  - Vai ter de estimar de alguma forma o “message_delay”
  - Esta estimativa vai ter um erro
  - Que vai afectar a estimativa do valor do relógio remoto
- Independentemente da função de convergência
  - **Os relógios nunca ficarão perfeitamente sincronizados, devido aos erros na leitura dos relógios remotos!**

# Algoritmo de Cristian (1989)

- O processo p ajusta o seu relógio para x + Tround/2
- Com um erro de ± (Tround-2min)/2
  - Se o valor de *min* for desconhecido ou muito pequeno, assume-se que é 0, para obter uma estimativa (pessimista) do erro

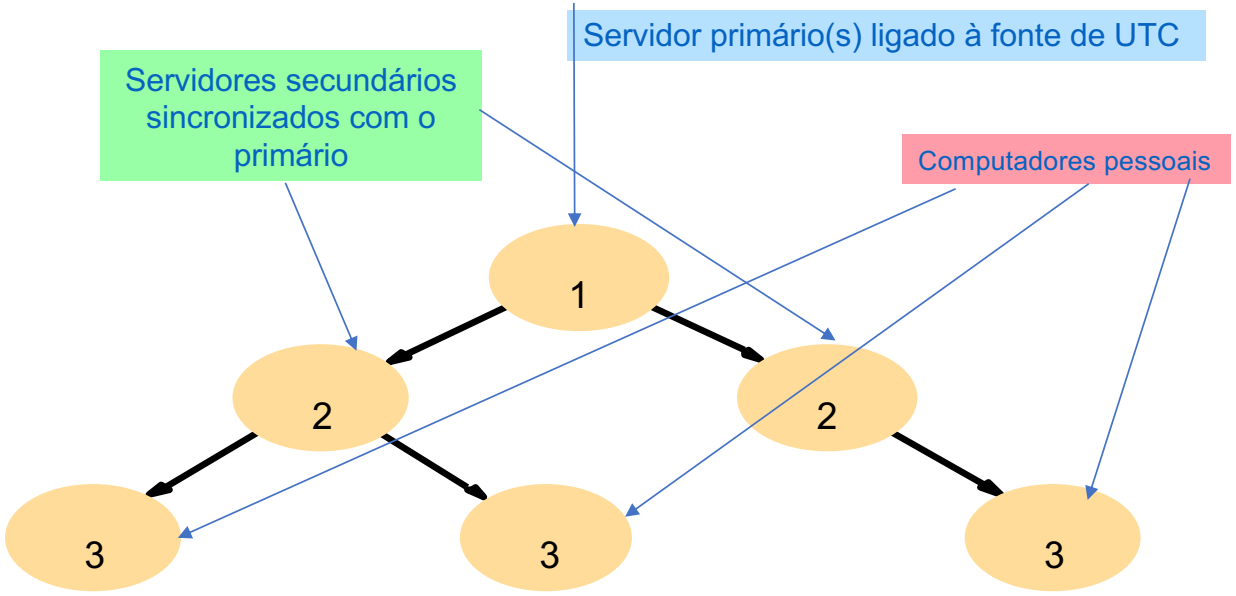


# Algoritmo de Berkeley

- Permite fazer a sincronização interna de um grupo de computadores
- O servidor mestre:
  - Lê o valores dos escravos (usando o “round trip time” para estimar os valores lidos)
  - Obtém a média dos valores obtidos
  - Envia os ajustes requeridos para cada um dos escravos
    - Note-se que não envia o tempo para acerto do relógio pois isso dependeria do round trip time. Q: Qual o problema que surgiria?
- Se o mestre falha é preciso eleger um novo

# Network Time Protocol (NTP)

- Serviço de tempo para a Internet:
  - Permite que os seus clientes sincronizem os relógios de acordo com o UTC
  - Robustez e escalabilidade resultante da redundância nos caminhos de disseminação



# Network Time Protocol (NTP)

- A rede pode reconfigurar-se quando ocorrem falhas:
- Se um servidor primário perde a ligação com a fonte de UTC então torna-se um servidor secundário
- Um secundário que perca o seu servidor primário pode usar outro primário

# Network Time Protocol (NTP)

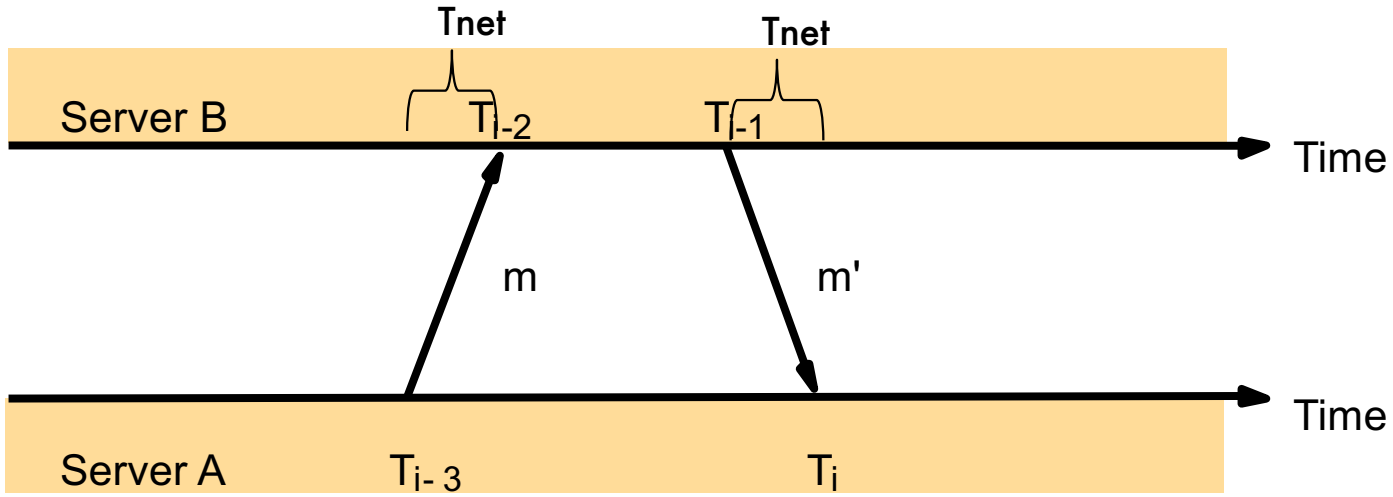
- Multicast (vocacionado para uma rede LAN)
  - Servidor faz LAN multicast para outros servidores que ajustam os seus relógios assumindo um dado delay (não é muito preciso, aproximação de síncrono)
- Invocação remota (análogo ao Cristian)
  - Servidor aceita pedidos de outros
  - Responde com o seu tempo actual
  - Útil se não houver multicast hardware
- Simétrica:
  - Pares de servidores trocam mensagens que contêm informação sobre o tempo
  - Usado quando é necessário maior precisão
- Em todos os casos é usado UDP

# Mensagens entre Pares NTP

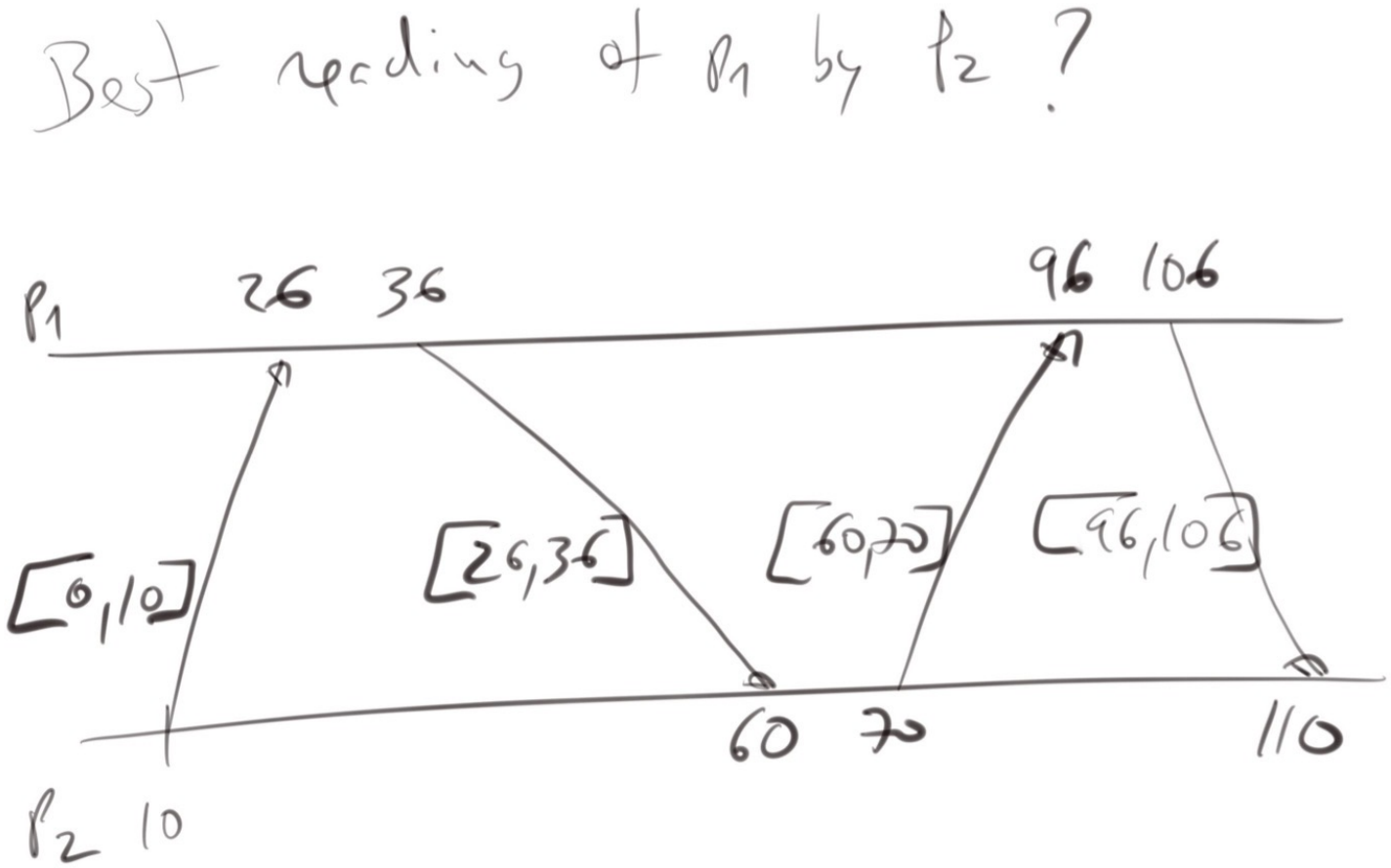
- Baseado em UDP
- No modo simétrico pode haver um atraso significativo entre a recepção de uma mensagem e o envio da resposta
- Para permitir ter este atraso em consideração cada mensagem é estampilhada com dois valores:
  - Tempo local de emissão da mensagem actual e da recepção da mensagem anterior

# Mensagens entre Pares NTP

- Tempo na rede, Tnet = $(T_i - T_{i-3}) - (T_{i-1} - T_{i-2})$



# Combinando mensagens



# Relógios Lógicos

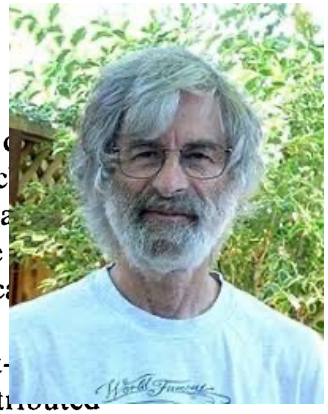
# Tempo lógico

- Como vimos, qualquer algoritmo de sincronização de relógios está limitado pelo erro de leitura
- Nalguns sistemas, em que a rede pode estar sobrecarregada e os atrasos na rede variam muito, este erro pode ser grande
- Pode ser impossível garantir que, para uma dada mensagem, o valor do relógio no emissor é inferior ao valor do relógio do receptor!
  - Isto é o que em linguagem técnica se designa por “uma chatice”

# Tempo lógico

- Estampilhas temporais lógicas (1, 2, 3, ….) que não têm nenhuma relação com o UTC
- Mas que possuem a seguinte propriedade:
  - Se um evento e2 está no futuro de outro evento e2 então t(e2)>(te1).
- A noção de passado/ futuro é caracterizada por uma relação designada por “aconteceu-antes”

# Relação “aconteceu-antes”



Operating Systems — R. Stockton Gaines, Editor

## Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

**The concept of one event happening before another in a distributed system is examined, and is shown to define a partial ordering of the events. A distributed algorithm is given for synchronizing a system of logical clocks which can be used to totally order the events**

A distributed system consists of a collection of d processes which are spatially separated, and whic municate with one another by exchanging messa network of interconnected computers, such as the net, is a distributed system. A single computer ca be viewed as a distributed system in which the control unit, the memory units, and the input- channels are separate processes. A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process.

We will concern ourselves primarily with systems of spatially separated computers. However, many of our remarks will apply more generally. In particular, a multiprocessing system on a single computer involves problems similar to those of a distributed system because of the unpredictable order in which certain events can occur.

In a distributed system, it is sometimes impossible to say that one of two events occurred first. The relation “happened before” is therefore only a partial ordering of the events in the system. We have found that problems often arise because people are not fully aware of this fact

# Relação “aconteceu-antes” Relação “aconteceu-antes”

## Leslie Lamport

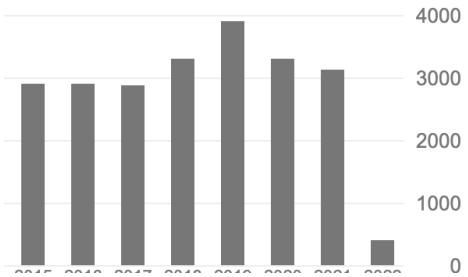
Unknown affiliation
No verified email

FOLLOW

| TITLE | CITED BY | YEAR |
|---|---|---|
| Time, clocks, and the ordering of events in a distributed system<br>L Lamport<br>Concurrency: the Works of Leslie Lamport, 179-196 | 13103 | 2019 |
| The Byzantine generals problem<br>L Lamport, R Shostak, M Pease<br>Concurrency: the Works of Leslie Lamport, 203-226 | 8078 | 2019 |

Cited by VIEW ALL

| | All | Since 2017 |
|---|---|---|
| Citations | 79240 | 17385 |
| h-index | 82 | 45 |
| i10-index | 189 | 112 |



# Relação “aconteceu-antes”

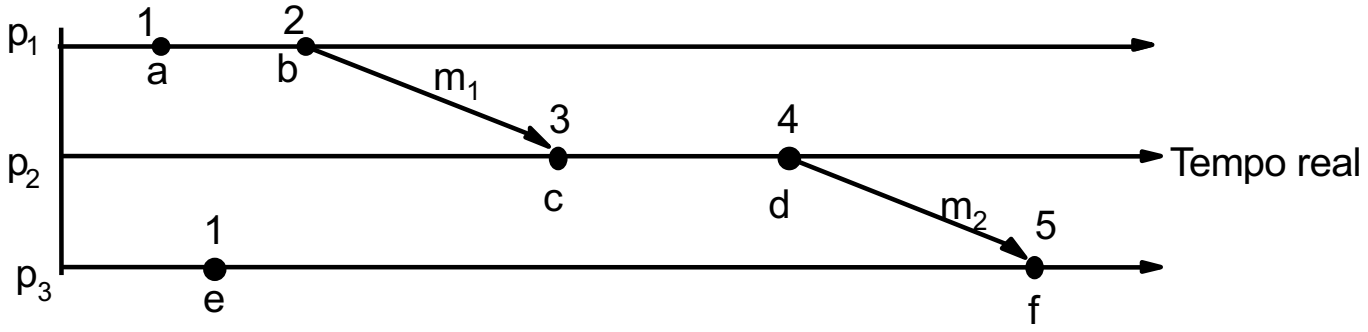
- Se e1 e e2 ocorreram no mesmo processo, e e1 foi executado antes de e2, então “e1 aconteceu-antes de e2”
- Quando uma mensagem m é trocada entre dois processos p1 e p2 a “emissão(m) aconteceu-antes da recepção(m)”
- A relação “aconteceu-antes” é transitiva

- Também designada por relação de causa-efeito potencial ou simplesmente, ordem causal

# Relógios lógicos de Lamport

- Um relógio lógico é um contador (software) monotónico (não é preciso dispor de um relógio físico nem se relaciona com tal)
- Cada processo pi tem um relógio lógico Li, inicializado a zero, que é usado para carimbar (timestamping) os eventos:
  - LC1: Li é incrementado de 1 unidade antes de cada evento no processo pi
  - LC2:
    - (a) quando o processo pi envia uma mensagem m, envia o valor de t = Li
    - (b) quando pj recebe (m,t) faz Lj := max(Lj, t) e aplica LC1 antes de carimbar o evento receive (m)

# Relógios lógicos de Lamport



# Relógios lógicos de Lamport

- Note-se que *e aconteceu-antes de e'* implica L(e)<L(e') mas o inverso não é verdade.

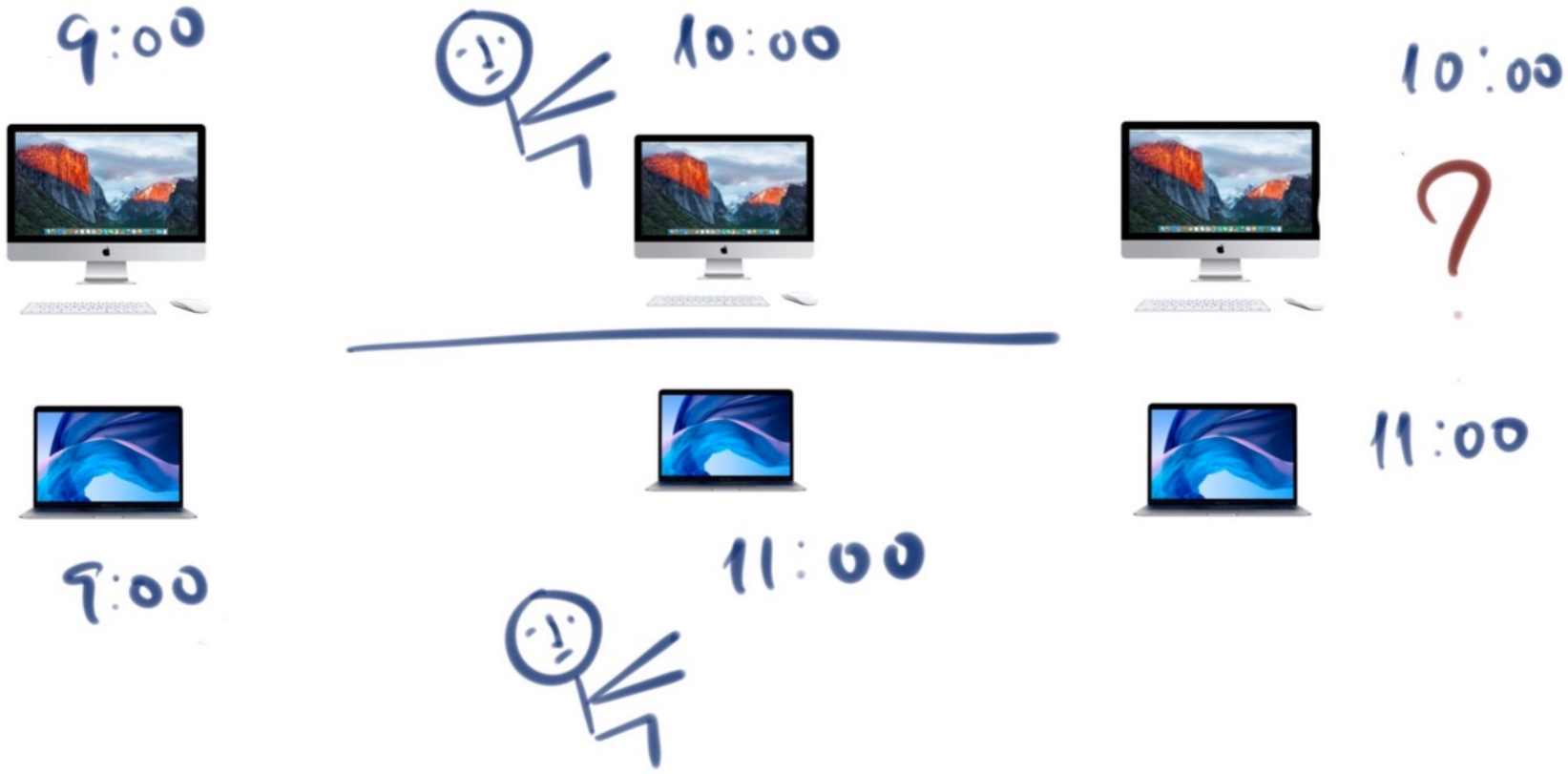
- L(e)<L(e') **não** implica e aconteceu-antes de e'

# Relógios escalares

- *Usar apenas um único escalar para estampilhar eventos tem algumas limitações:*
  - *T(e)<T(e')* **não** implica que *e aconteceu-antes de e'*
  - *T(e)=T(e´)* **não** implica *e = e'*
- *Esta limitação existe quer com os relógios físicos quer com os relógios lógicos*

# Relógios escalares

# Relógios escalares



# Relógios vectoriais

- Baseados nos relógios lógicos de Lamport
- Cada processo mantém um vector, com um relógio lógico para cada processo no sistema
- Os eventos são estampilhados com o vector

# Relógios vectoriais

- Regra VC1:
  - inicialmente Vi[ j ] = 0 para i, j = 1, 2, …N
- Regra VC2:
  - antes de pi carimbar um evento faz Vi[i] := Vi[i] +1
- Regra VC3:
  - pi envia o seu t=Vi em cada mensagem enviada
- Regra VC4:
  - quando pi recebe(m,t) faz Vi[ j ] := max(Vi[ j ] , t[ j ]),  j = 1, 2, …N

# Relógios vectoriais

- V=V’
  - sse V[j] = V’[j] para j=1,2,...N
- V<=V’
  - sse V[j] <= V’[j] para j=1,2,...N
- V<V’
  - sse V<= V’ e V !=V’

- e **aconteceu-antes** e’ **se e só se**
  - V(e)<V(e’)
- e **é concorrente com** e’ **se e só se**
  - não se verificar nenhuma destas condições: V(e)=V(e’) e V(e)<V(e’) ou V(e’)<V(e)