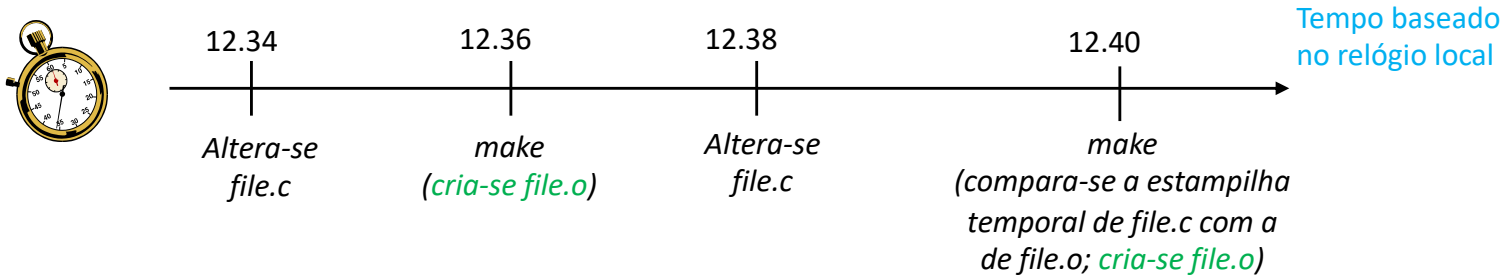
Tempo em Sistemas Distribuídos

Medir e comparar tempo

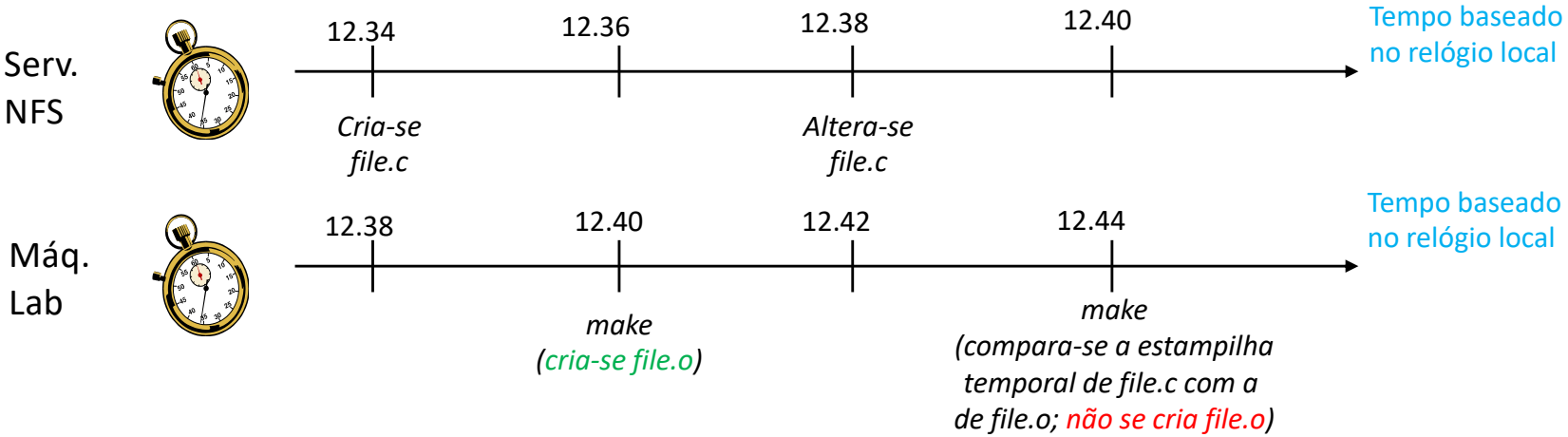
- A noção de tempo é usada frequentemente nos sistemas
  - Protocolos de autenticação
  - Data de criação/ alteração nos sistemas de ficheiros
  - Medidas de desempenho
  - Timeouts
  - etc

Motivação

- Exemplo : comando *make* num único computador



- Exemplo : comando *make* num sistema distribuído



Problema

- Dada a importância de termos relógios sincronizados, será possível **sincronizar** todos os relógios num sistema distribuído?

Relógios Físicos

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)
- Mas…

Desvios entre relógios físicos

- Na realidade, os relógios sofrem desvios (*drift*):
  - Relógios de quartzo correntes: 1 seg em 11-12 dias (10^{-6} segs/seg)
  - Relógios de quartzo de alta precisão: desvios menores (10^{-8} segs/seg) mas também se desviam
- Mesmo que, num dado instante, todos os relógios tivessem o mesmo valor, com o passar do tempo, apresentariam diferenças entre si (*skew*)

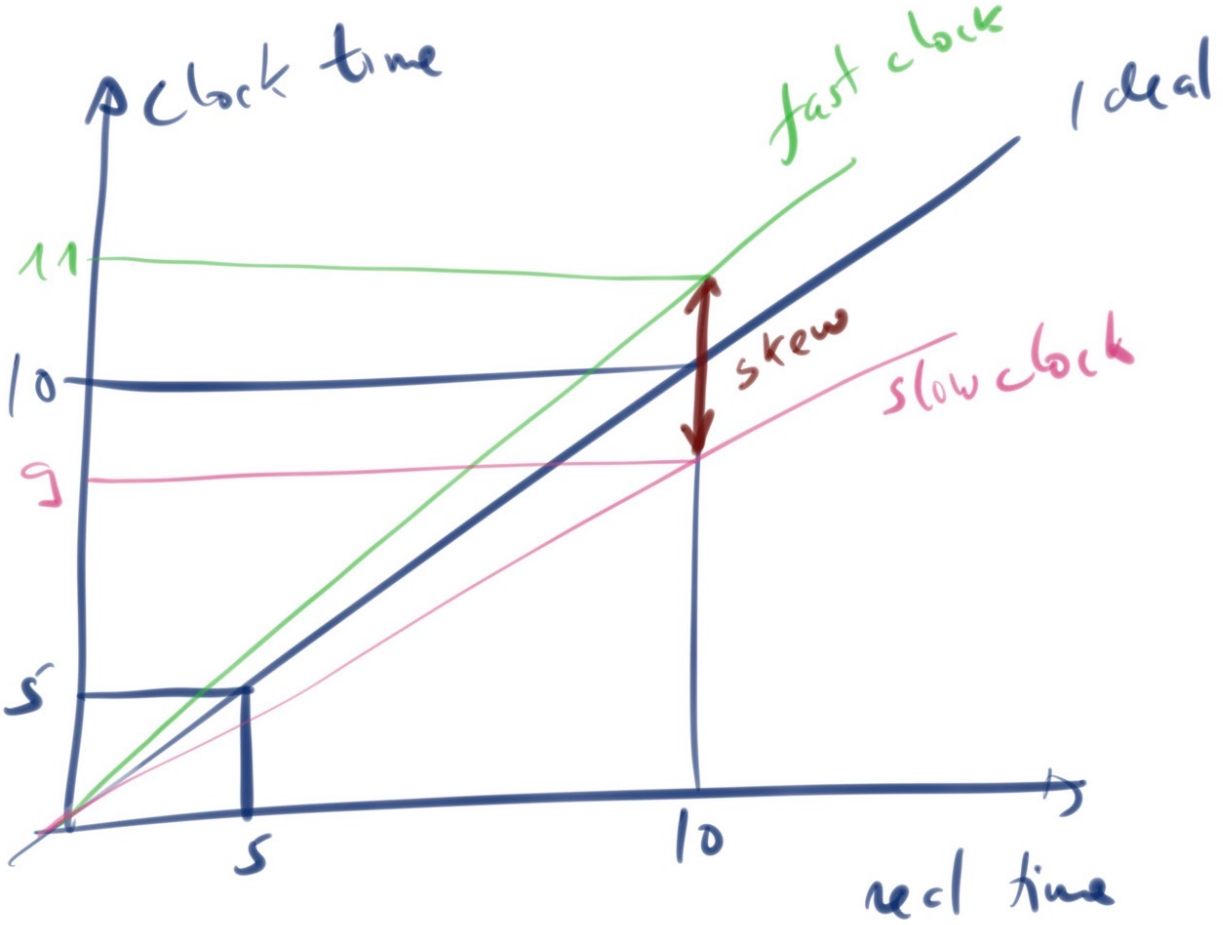
Desvios entre relógios físicos (II)

- 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!

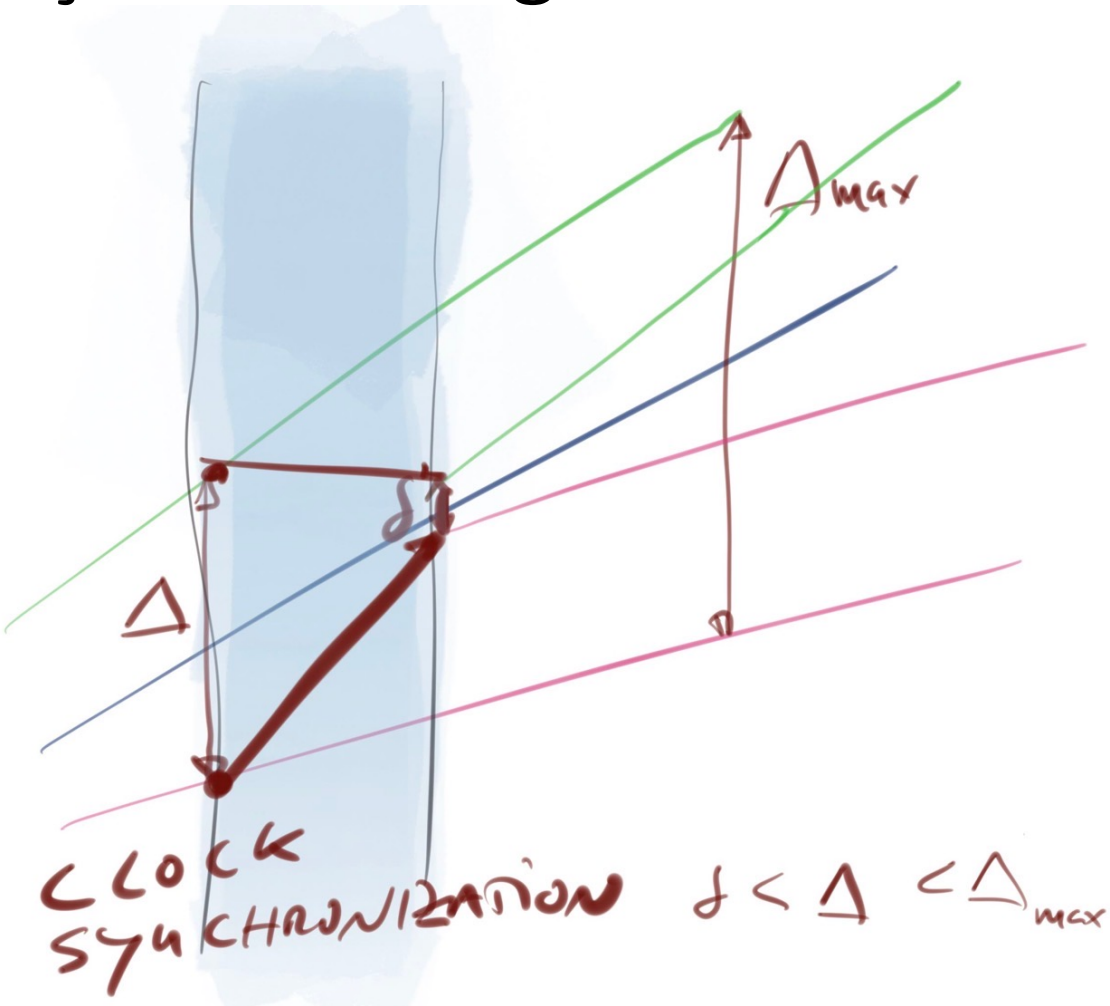
Sincronização de Relógios

- 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



As duas fases da sincronização de relógios

- Genericamente, há duas fases:
  1) Um processo estima o valor de um ou mais relógios de outros processos
  2) Com base no valor do seu relógio local e no valor obtido na fase anterior, o processo ajusta o seu relógio local

Sincronização Interna e Externa

- Sincronização externa
  - Quando um ou mais nós têm acesso a uma **fonte de tempo “real”**

- Sincronização interna
  - Os nós tentam reduzir a disparidade dos valores que os seus relógios apresentam **sem acesso a nenhuma fonte externa**
    - Podem estar sincronizados entre si mas desfasados do tempo universal

Sincronização externa

Relógios físicos

Tempo Universal Coordenado (UTC)

- UTC é um standard internacional baseado em relógios atómicos e ocasionalmente ajustado de acordo com medidas astronómicas
- 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: precisão entre 0.1-10 milisegundos
  - Sinais de satélite (GPS): 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
- Todos acertam 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 lêem 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

Relógios físicos

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

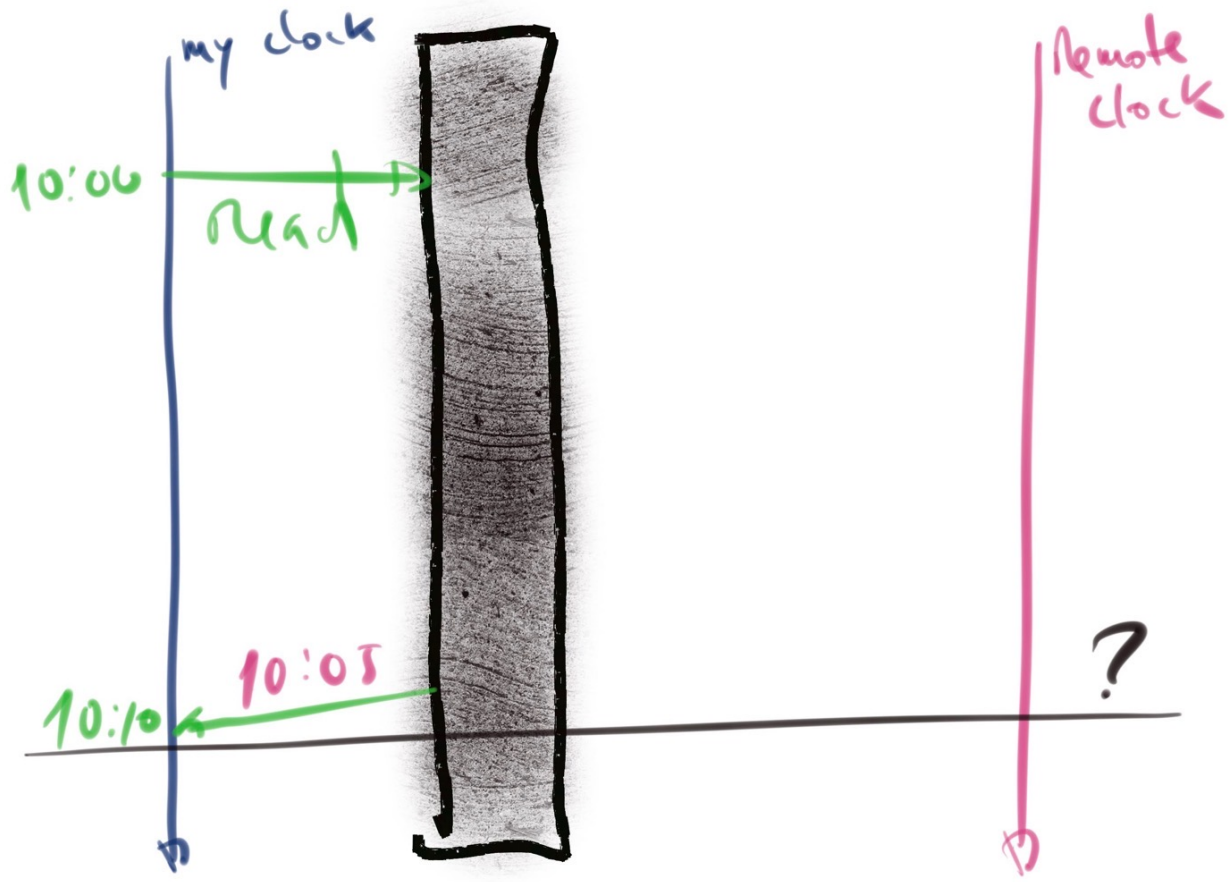
Leitura de Relógios Remotos

- 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

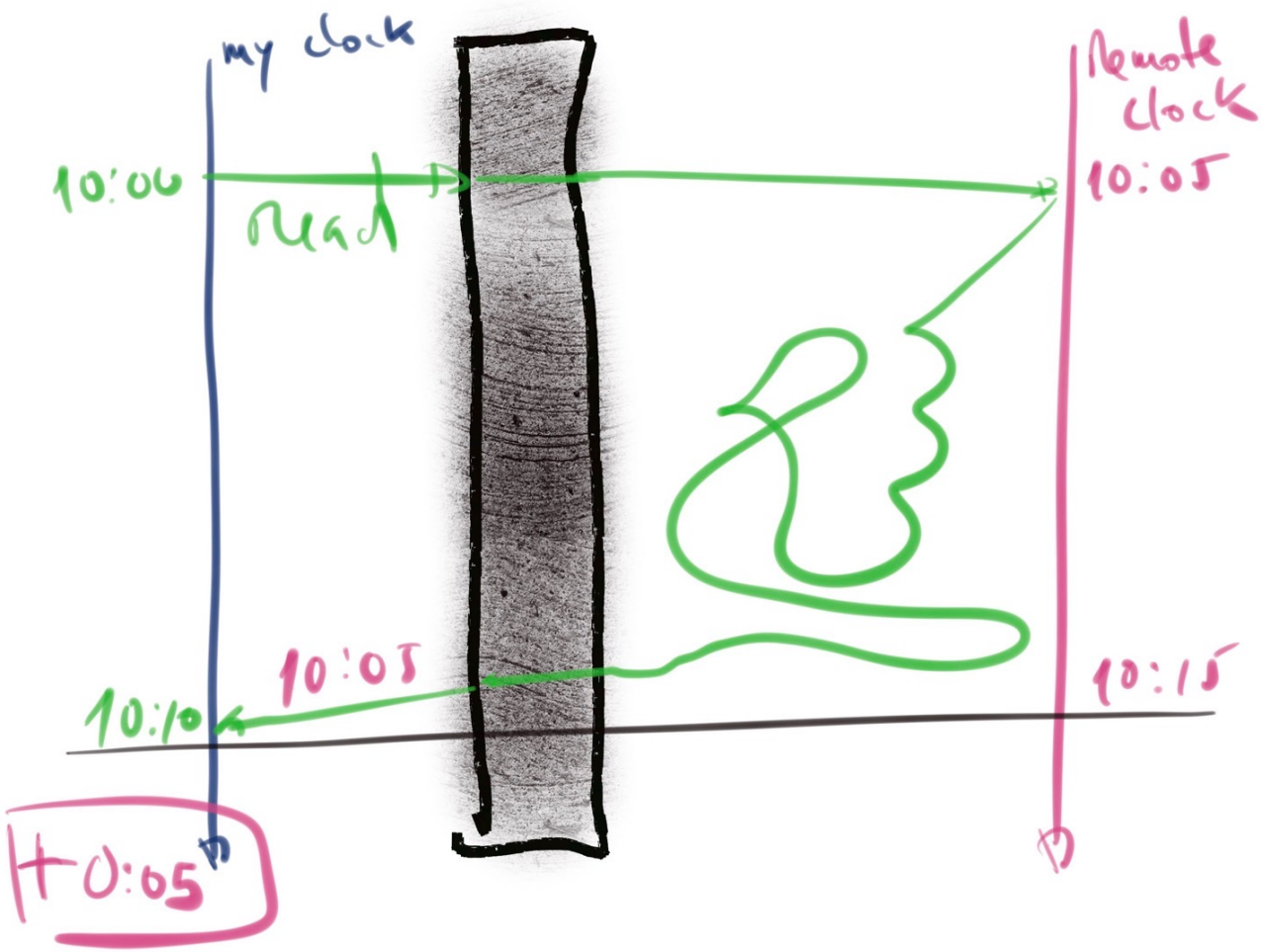
Leitura Remota de Relógios

- Vamos assumir que p1 quer ler o valor do relógio de p2:
  - t1: p1 envia pedido de leitura a p2
  - t2: p2 recebe o pedido, lê o valor do seu relógio (seja este valor X) e envia X a p1
    - Nota: estes 3 passos costumam ser muito rápidos, logo tipicamente assumimos que acontecem no mesmo instante
  - t3: p1 recebe o valor X
- p1 deve acertar o seu relógio para que valor?
  - Resposta: Para X + (t3-t2)
- Mas como é que pode p1 pode determinar (t3-t2)?...

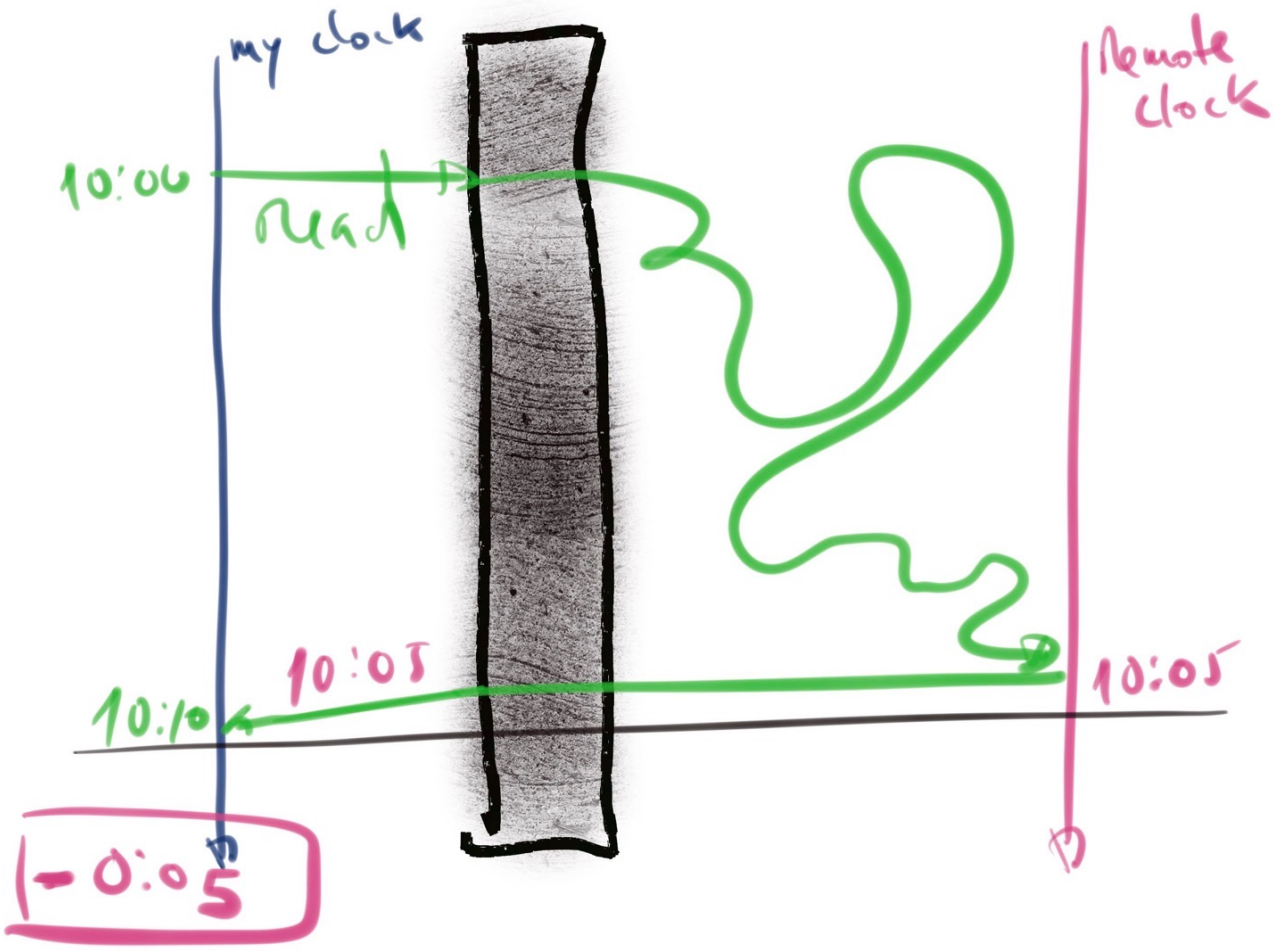
Leitura Remota de Relógios



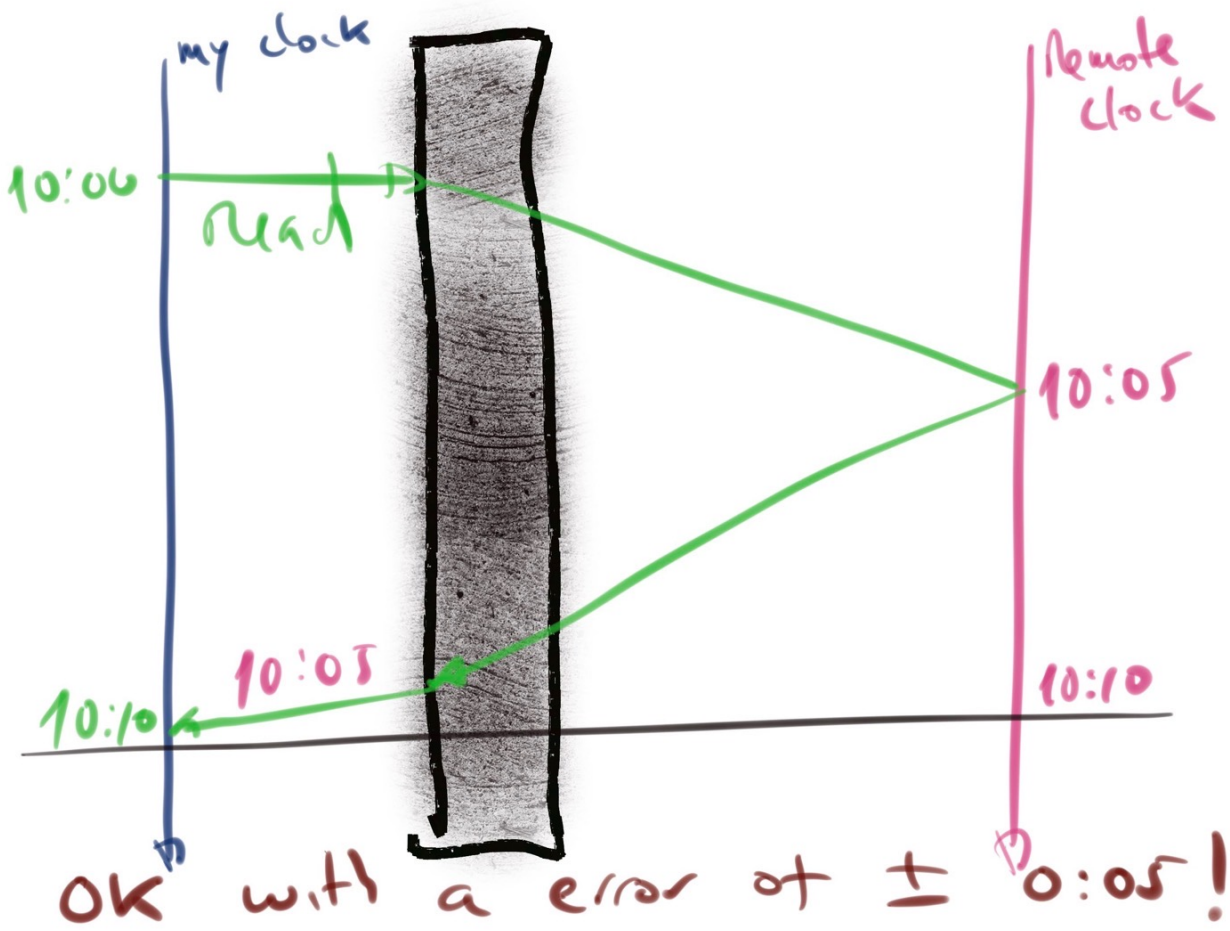
Leitura Remota de Relógios



Leitura Remota de Relógios



Leitura Remota de Relógios



Recapitulando…

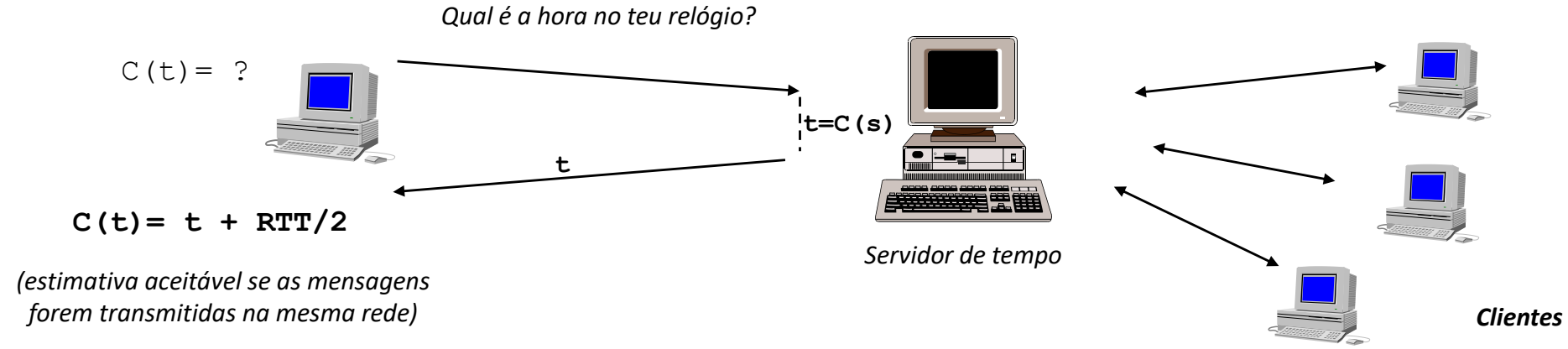
- O processo p1 não tem maneira de saber quando é que a mensagem de resposta foi enviada
  - Vai ter de estimar de alguma forma o tempo de envio dessa mensagem
  - Esta estimativa vai ter um erro…
  - …Que vai afectar a estimativa do valor do relógio remoto
- Logo, os relógios nunca ficarão perfeitamente sincronizados!

Algoritmo de Cristian
Algoritmo de Berkeley
Network Time Protocol (NTP)

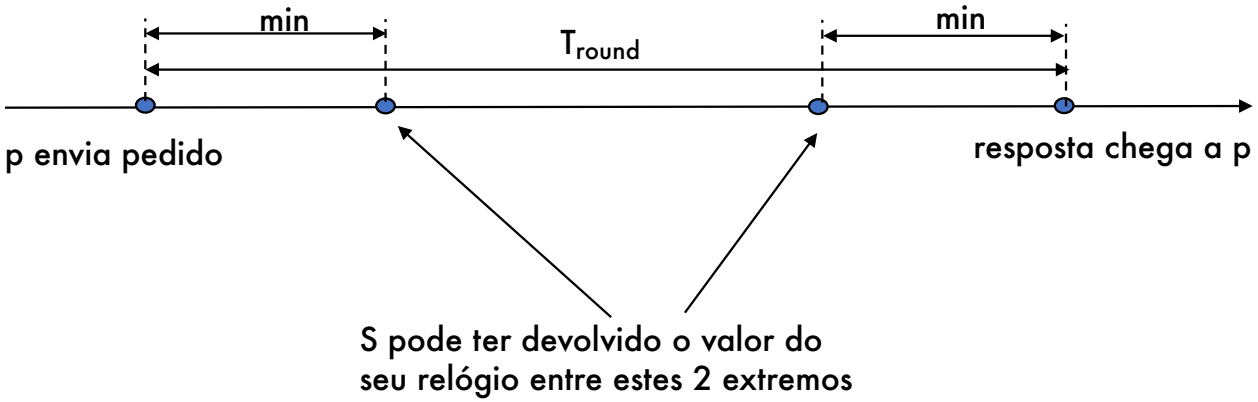
Diferentes formas de concretizar sincronização de relógios físicos

Algoritmo de Cristian (1989)

- Relógios dos clientes sincronizados pelo relógio de um servidor de tempo (sincronização externa) [Cristian1989]



Algoritmo de Cristian: qual a precisão?



- Precisão: ± (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

Exercício

Round-trip (ms)	*Time (hr:min:sec)*
22	*10:54:23.674*
25	*10:54:25.450*
20	*10:54:28.342*

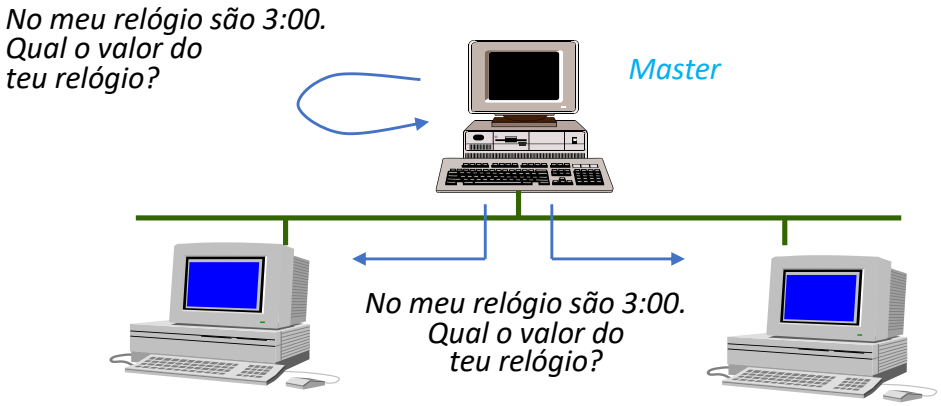
Um cliente tenta sincronizar-se com um servidor. Para tal, guarda os RTT e os tempos enviados pelo servidor, de acordo com a tabela acima.

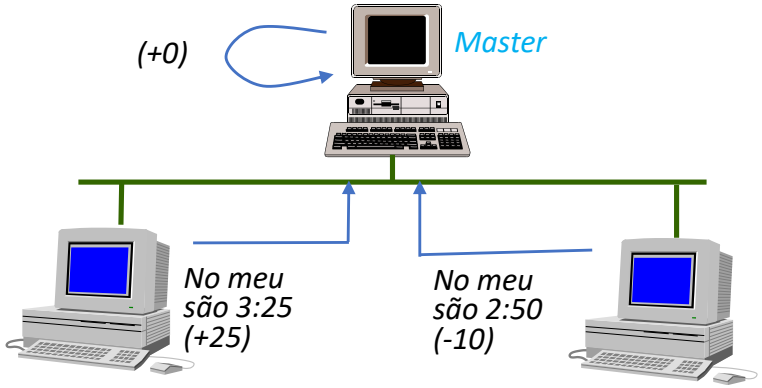
1. Com qual deste valores é que o servidor deve acertar o seu relógio de modo a conseguir a melhor precisão?
   R: Com o terceiro pq tem o menor RTT, de 20ms.
2. Qual o valor com que o deve acertar?
   R: C(t) = t + RTT/2 = 10:54:28.342 + 20ms/2 = 10:54:28.352
3. Qual a precisão desse acerto?
   R: ±10ms
4. E se soubermos que o tempo de envio de uma mensagem é no mínimo de 8ms, essa precisão é alterada? Se sim, qual o novo valor?
   R: ±(RTT/2-ms) = ±(20/2-8) = ±2ms

Algoritmo de Berkeley

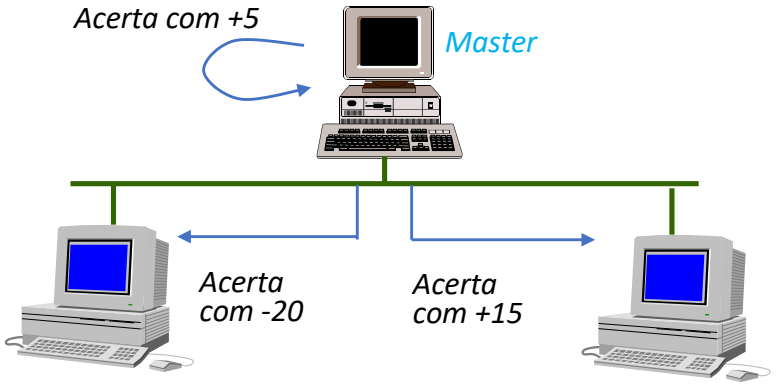
- Permite fazer a sincronização interna de um grupo de computadores
- Periodicamente o *master* pergunta aos outros processos qual o valor dos seus relógios
  - Para cada resposta, o master estima a hora do emissor usando a abordagem do Cristian
  - Calcula a nova hora do sistema por uma média dos valores (incluindo o seu)
  - Envia o acerto correspondente a cada computador
- Para tolerar falhas, não considera aqueles que têm desvios exagerados ou cuja resposta não chegou num tempo razoável
- Se o mestre falha é preciso eleger um novo

Exemplo





Média = 3:05
(+5)



(*) Na verdade, o *master* mede o RTT e corrige cada valor recebido como no Cristian. Mas neste exemplo vamos ignorar esse passo (i.e., assumimos que RTT≈0).

Exercício

- 3 máquinas (A, B, C), algoritmo de Berkeley. O *master* é A.
- A enviou a sua hora, 13:15:15, a todos e recebeu estas respostas:

[A = 13:15:15]
B = 13:15:05
C = 13:16:07

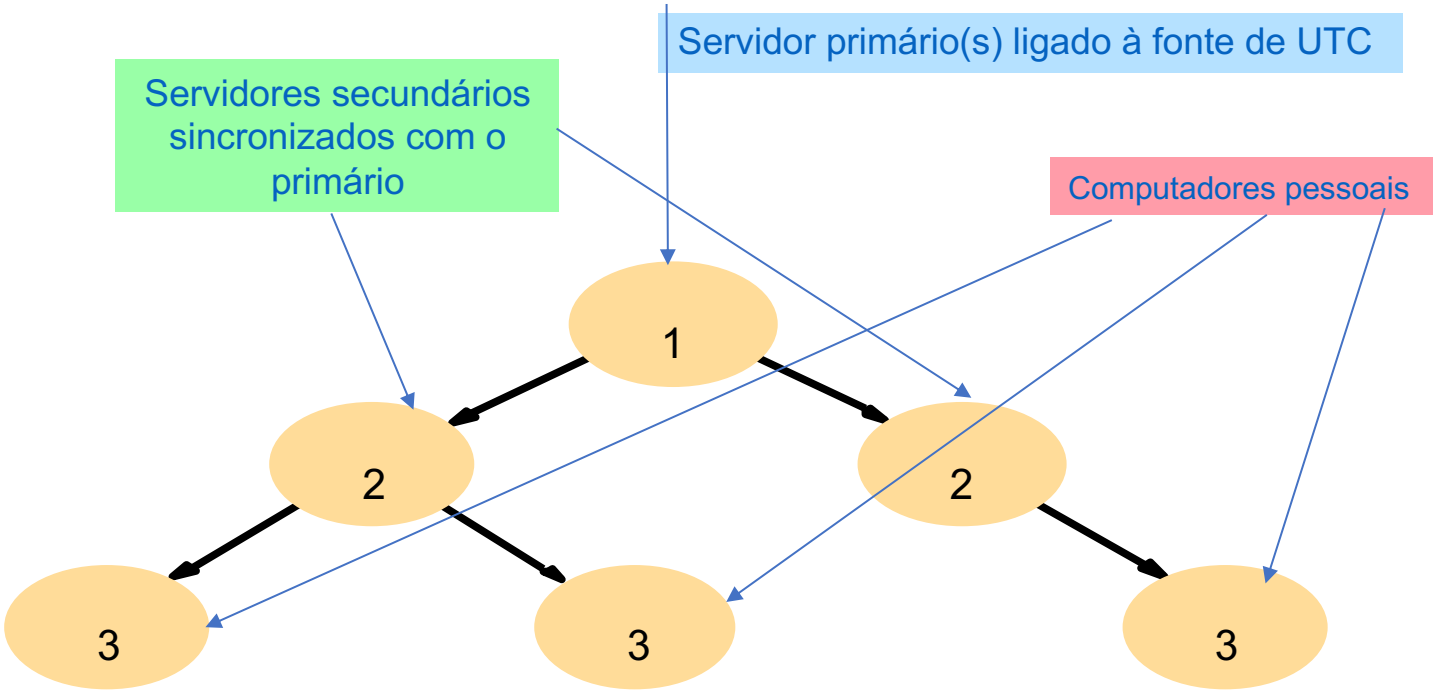
Qual o acerto de relógio que o *master* enviará às máquinas?

Por simplicidade, assuma que o tempo de *round trip* é nulo.

R: A acerta com +14 segundos, B +24 segundos, C -38 segundos

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)

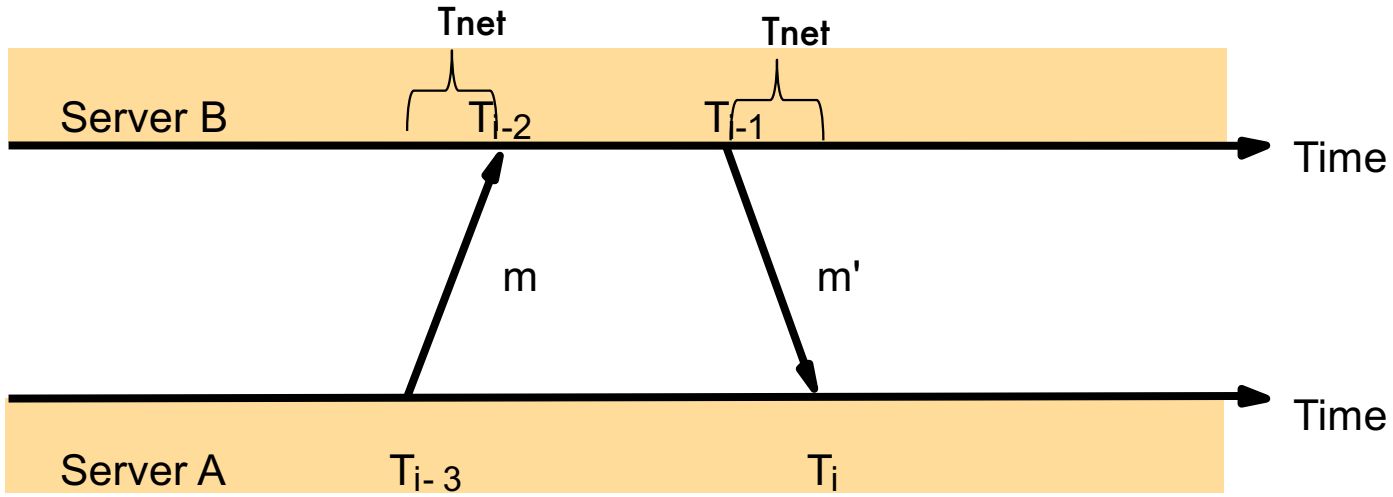
- Multicast (vocacionado para uma rede LAN)
  - Servidor faz LAN multicast para outros servidores que ajustam os seus relógios assumindo um dado delay (não é muito preciso, aproximação de síncrono)
- Invocação remota (análogo ao Cristian)
  - Útil se não houver multicast hardware
- Simétrica:
  - Pares 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

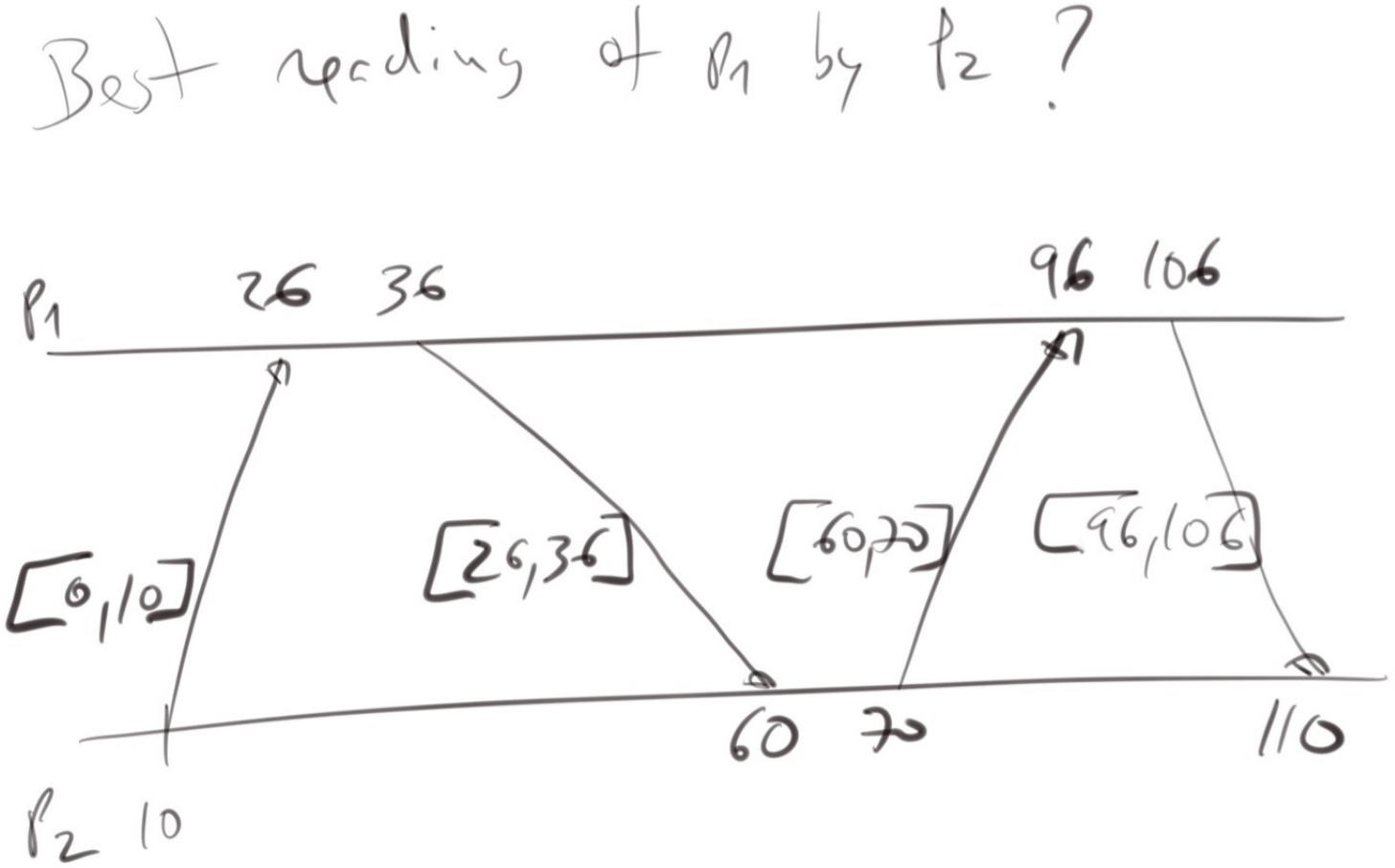
- No modo simétrico pode haver um atraso significativo entre a recepção de uma mensagem e o envio da resposta
- Para ter este atraso em consideração cada mensagem é estampilhada com dois valores:
  - Tempo local de emissão da mensagem actual e
  - Tempo local da recepção da mensagem anterior

Mensagens entre Pares NTP

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



Combinando mensagens



Limitações dos relógios físicos em SD

- 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”

Mas precisamos mesmo de tempo físico (que não conseguimos sincronizar perfeitamente)?

Muitas vezes, a aplicação só precisa capturar um **ordem causal** (em vez da ordem de tempo real)

Um exemplo em que o que interessa é **ordem causal** (em vez de tempo real)

- Consideremos um SD com diferentes serviços + um serviço de *logging*
- Sempre que cada serviço um executa uma operação local, envia-a para o serviço que mantém um *log* durável
- Se uma operação num serviço **depende causalmente** de outra operação (possivelmente executada noutro serviço), **queremos que a ordem das operações no log respeite essa ordem causal**

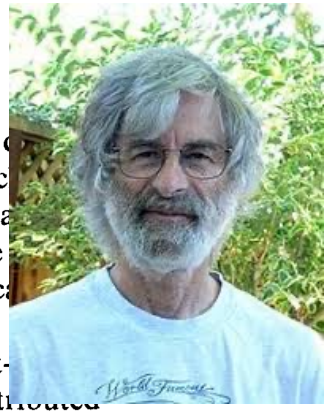
- Boas notícias: há outro tipo de relógios que resolvem este problema sem a imprecisão dos relógios físicos!

Relógios Lógicos

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

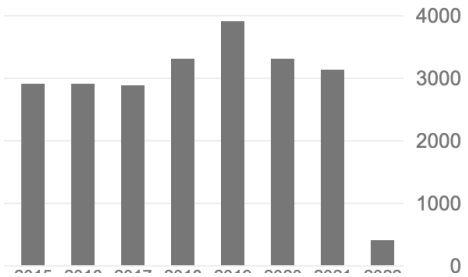
Unknown affiliation
No verified email

FOLLOW

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

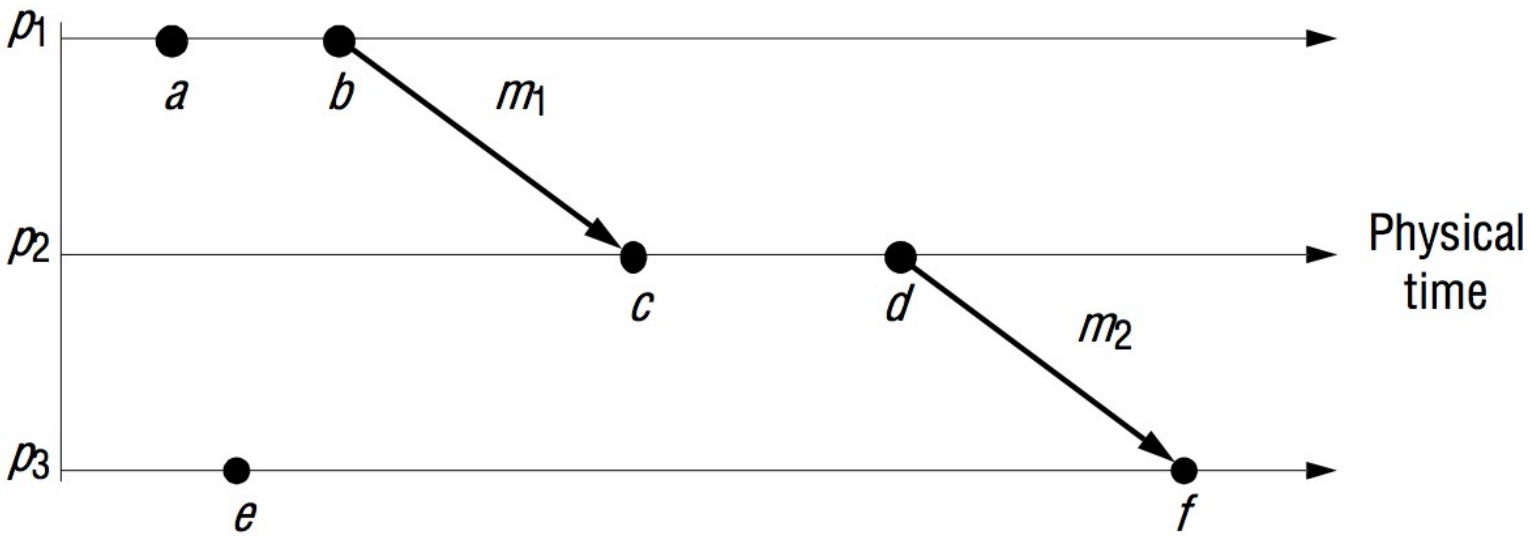


Relação “aconteceu-antes”

- 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

Exemplos de *aconteceu-antes*

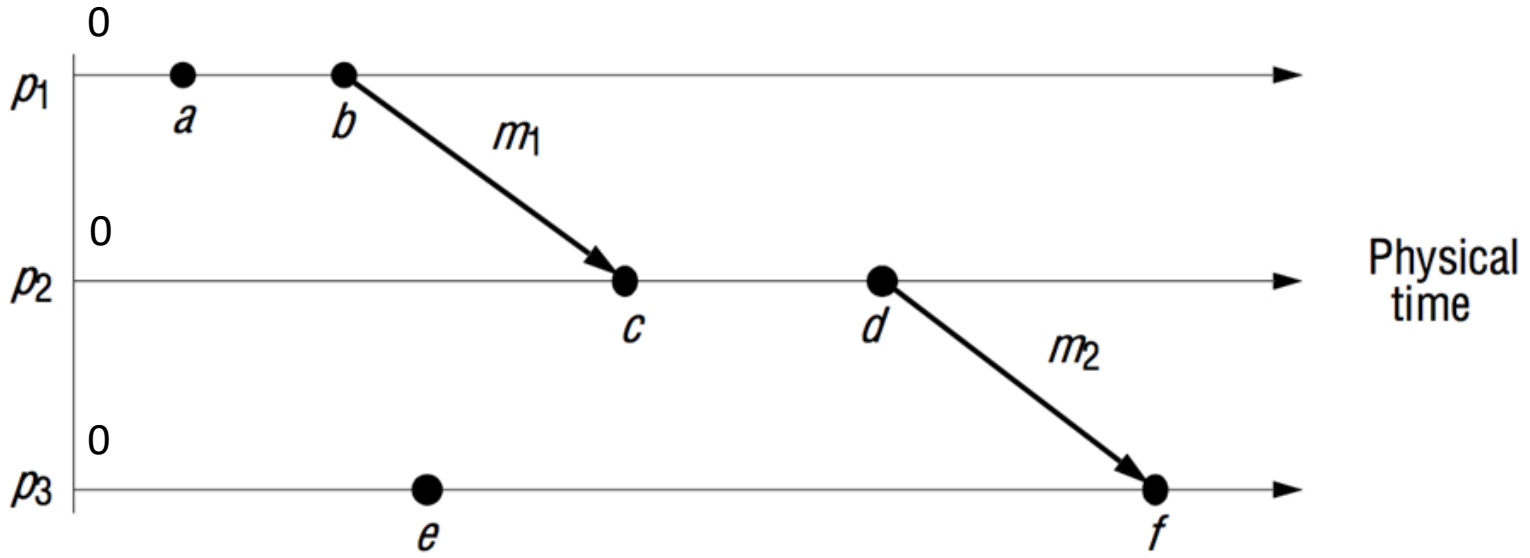


a → b	c → d	b → c	a → f	a || e

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)

Relógio lógico de Lamport: exemplo



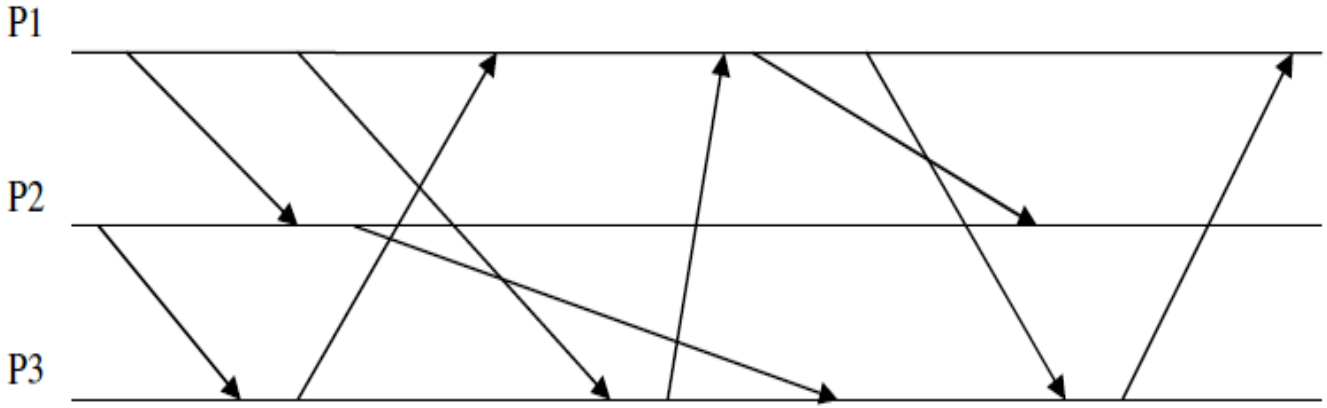
$x \rightarrow y$ então $C(x) < C(y)$

$C(x) < C(y)$ então $x \rightarrow y$

(Dois exemplos: ***e*** || ***b***, ***e*** || ***c***)

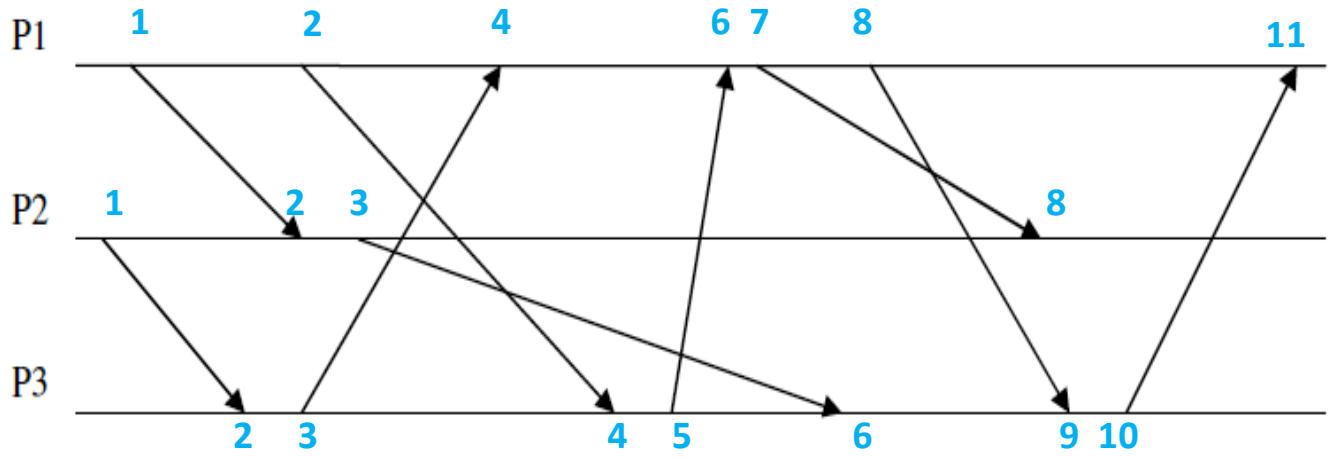
Exercício

- Observe o seguinte diagrama temporal onde está representada a execução de três processos P1, P2 e P3. Pretende-se ordenar os eventos de envio e receção de mensagens usando relógios lógicos de Lamport. Assuma que os relógios locais são inicializados a zero. Represente na figura o valor do relógio associado a cada um dos eventos de envio e recepção.



Exercício

- Observe o seguinte diagrama temporal onde está representada a execução de três processos P1, P2 e P3. Pretende-se ordenar os eventos de envio e receção de mensagens usando relógios lógicos de Lamport. Assuma que os relógios locais são inicializados a zero. Represente na figura o valor do relógio associado a cada um dos eventos de envio e recepção.

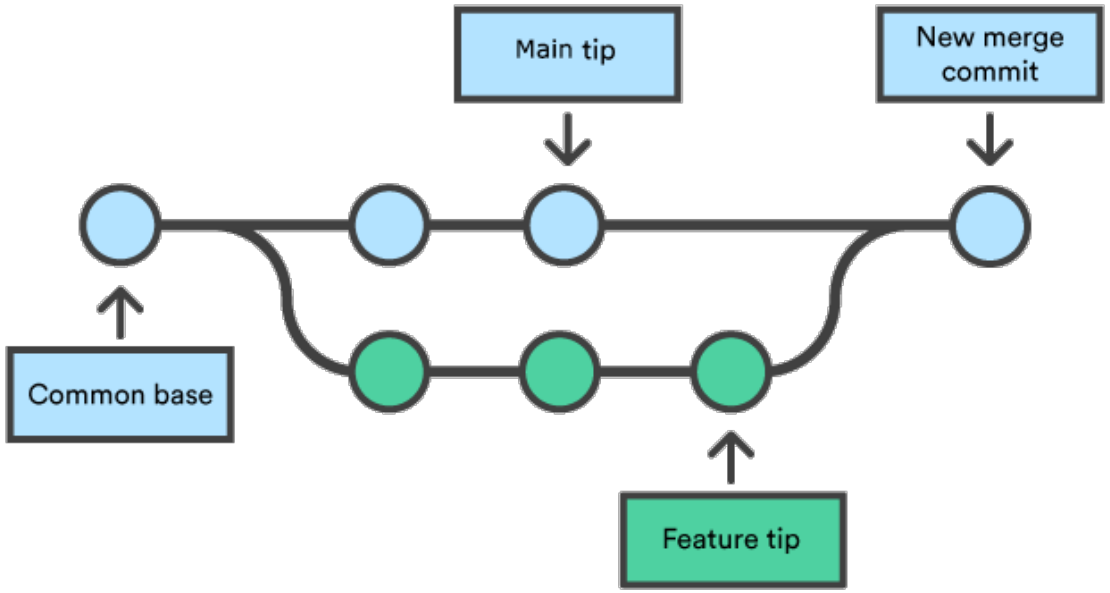


Limitações de todos os relógios que vimos até agora

- 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 nos relógios de Lamport…
- … E também em relógios físicos (mesmo que perfeitamente sincronizados, num mundo ideal)

Por vezes precisamos saber se e aconteceu-antes de e' ou não

- Um exemplo: *merge branches* em Git
- Caso o Git encontre 2 versões diferentes do mesmo ficheiro, precisa saber se uma aconteceu-antes da outra
  - Se sim, guarda a última
  - Se não, houve modificações concorrentes, logo há um merge conflict



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

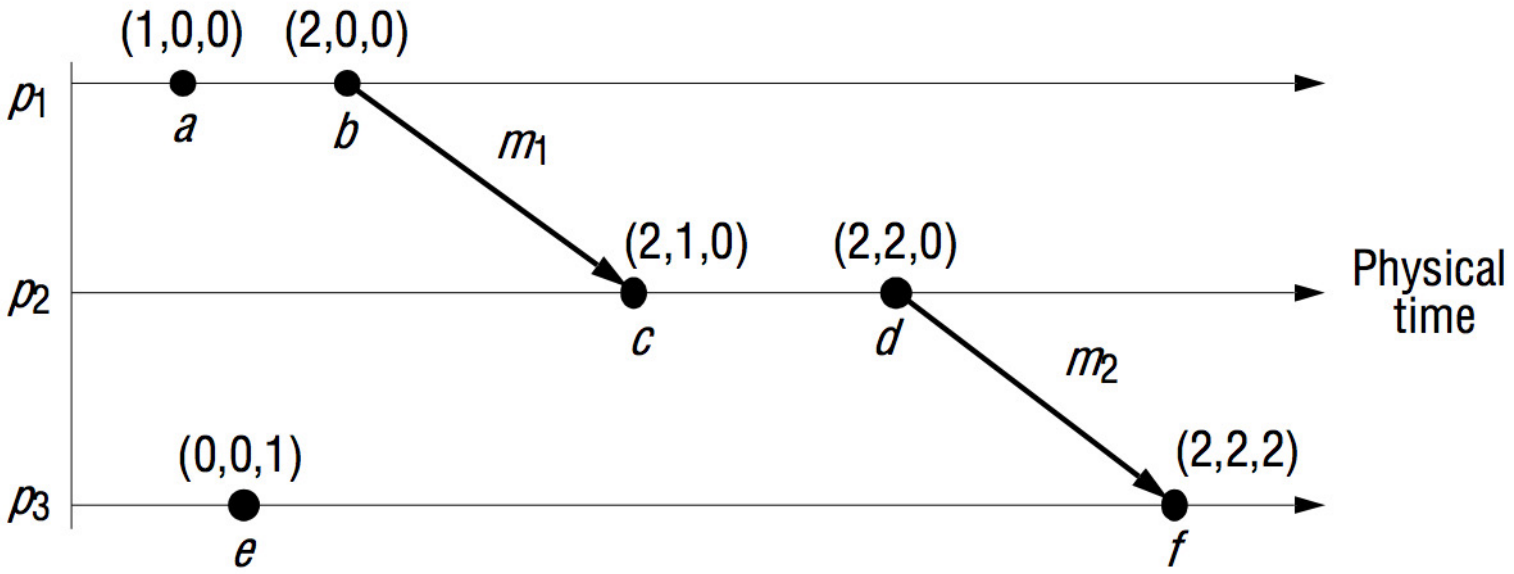
- Regra VC1:
  - inicialmente Vi[j] = 0 para i, j = 1, 2, …N
- Regra VC2:
  - antes de pi carimbar um evento faz Vi[i] := Vi[i] +1
- Regra VC3:
  - pi envia o seu t=Vi em cada mensagem enviada
- Regra VC4:
  - quando pi recebe(m,t) faz Vi[j] := max(Vi[j] , t[j]), j = 1, 2, …N

Relógios vectoriais

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

Exemplo



$x \rightarrow y$ então $V(x) < V(y)$

$V(x) < V(y)$ então $x \rightarrow y$

(Rever exemplo ***e*** || ***b***)

Fonte: [Tanenbaum2007]

Bibliografia recomendada

- [Coulouris et al]
  - Secções 14.1 a 14.4