

Coordenação

→ Um dos maiores problemas ...

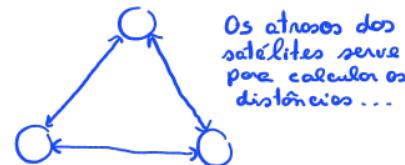


Relógios

- Por vezes é necessário o tempo exacto, não apenas uma ordem
- UTC: Universal Coordinated Time
 - Baseado no numero de transições por segundo de um atómo de cesium 133
→ Relógio atómico
 - Atualmente o valor UTC é uma média de 50 relógios atómicos espalhados pelo mundo.
→ Todos os SI têm como verdade do tempo
 - Introduz saltos de um segundo (leap second) para compensar o facto dos dias serem cada vez maiores
→ em broadcast!
- Valores UTC são radiodifundidos em onda curta e satélite.
- Satélite tem precisão na ordem dos 5ms.



Eles verificam se está perfeito com o UTC
eles comunicam com outros centros ...



Os altíssimos dos satélites servem para calcular as distâncias ...

• Nos caixas de multiboneca a coordenação tem de ser super coordenada ...
→ Relógio para determinar a ordem em que as operações são feitas!

Sincronização de relógios

Precisão: 2 relógios mantêm-se precisos entre eles

Exatidão: Estão exatos relativamente ao UTC

- Precisão: Objectivo é manter o desvio entre o relógio de duas máquinas dentro de um limite π

$$\bullet \forall t, \forall p, q : |C_p(t) - C_q(t)| \leq \pi \rightarrow \text{é preciso se 2 relógios lado a lado mantinham a mesma diferença!} \checkmark$$

Para $C_p(t)$ o valor calculado do tempo na máquina p no instante UTC t

- Exatidão: Objectivo é manter o desvio entre o relógio e o tempo UTC

$$\bullet \forall t, \forall p : |C_p(t) - \overset{\text{UTC}}{t}| \leq \alpha \leftarrow \text{Aqui sim! (a mesma diferença com o UTC)}$$

- Sincronização:

- Interna: relógios precisos \rightarrow Sincronizam 2 relógios internamente...
• A hora pode estar errada, mas é preciso!
- Externa: relógios exactos \rightarrow Comunicação com fonte externa do valor UTC
• A hora correta...

Clock drift

(Os servidores têm relógios muito corados)

ntpdate pool.ntp.org

Calcular isto!

(O Power Saver aumenta o Drift)

Relógios para satélites / comunicações / ...

Relógios de quarto, por exemplo

- Especificações de um relógio

- Razão máxima de divergência (maximum clock drift rate) - **p**
- F(t)** é a frequência de oscilação do relógio de hardware no momento **t**
- F é a frequência (constante) ideal do relógio – se o mesmo cumprir com as especificações

$$\forall t : (1-\rho) \leq F(t) \leq (1+\rho)$$

!

$$C_p(t) = \frac{1}{F} \int_0^t F(t) dt \Rightarrow \frac{dC_p(t)}{dt} = \frac{F(t)}{F} \leftarrow \text{"Tipo de desvio"}$$

$$\Rightarrow \forall t : (1-\rho) \leq \frac{dC_p(t)}{dt} \leq (1+\rho)$$

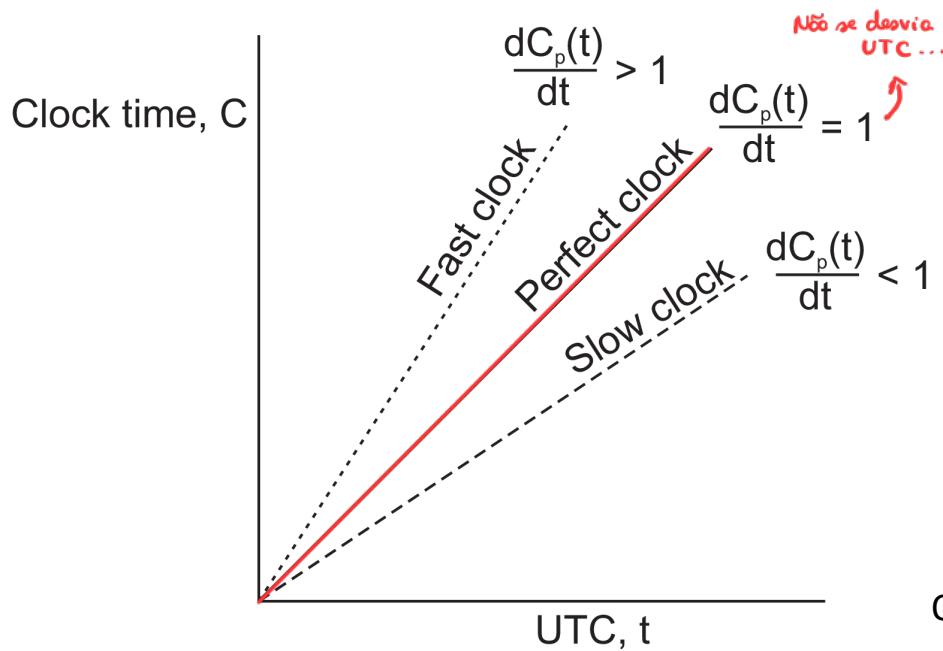
// → Sabemos o ideal de frequência !

↑
Se conseguir calcular
o Clock Drift, por software,
posso sincronizar...

Desvios de relógio

Clock Drift

Adiantados, Certos e atrasados

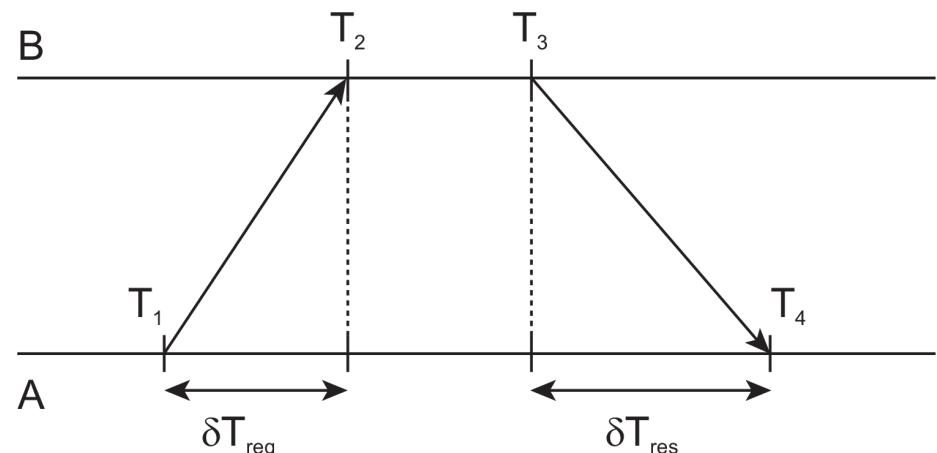


ntpdate pool.ntp.org

→ A latência do servidor vai aumentar o erro !!!

↑
Posso calcular isto com o erro ...

Obter tempo de um servidor (NTP)

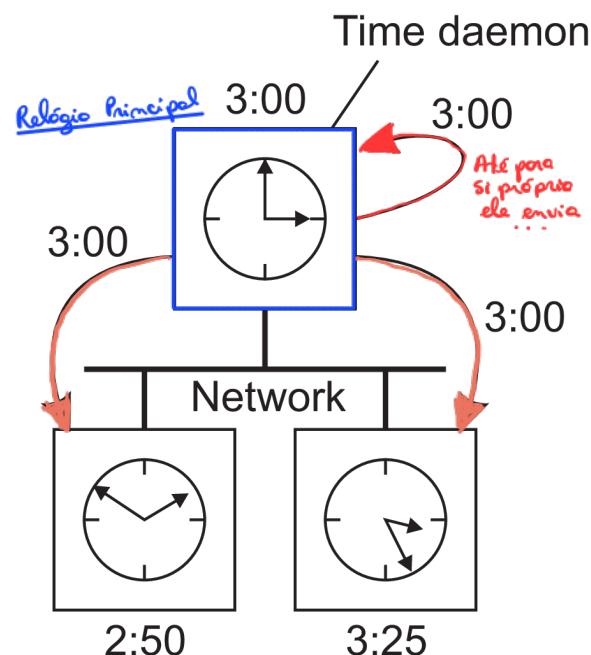


Offset/Desvio:

$$\theta = T_3 + \frac{(T_2 - T_1) + (T_4 - T_3)}{2} - T_4 = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

Sem UTC!
 ⇒ Relógios exatos!
 (não é Precisão...)

Sincronizar sem UTC (algoritmo Berkeley)



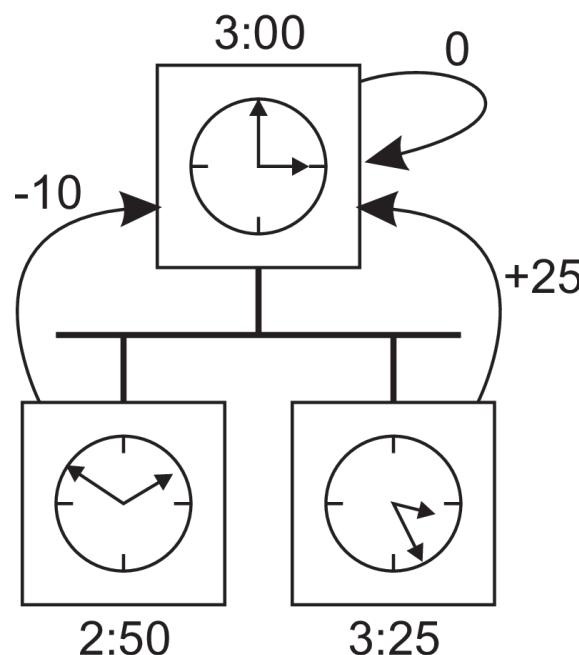
① Broadcast do meu tempo

Atenção! Não é permitido atrasar relógio, apenas alterar velocidade do tempo.

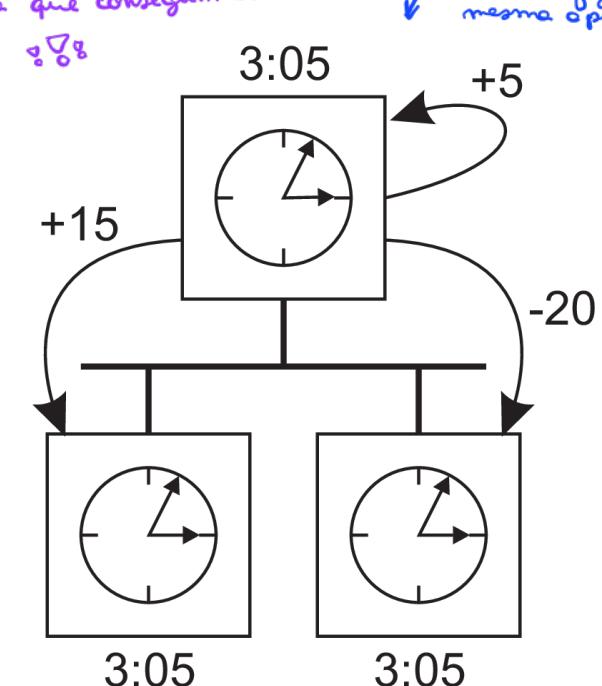
Sem acesso à internet!

Aqui a exatidão é a única coisa que conseguimos

↓ A média para todos e todos fazem a mesma operação!



② Qual é o meu offset



③ OK, calculando a média
 tu deves fazer isto...

Com base no UTC!

Em Portugal é em Português



• Video no Slack muito bom!
(explica este protocolo)

→ How does Britain know what time it is?

• Eles usam fibra para empresas...
(monosegundos de atraso)

Sincronização por difusão

⇒ Fazem em broadcast

Reference broadcast synchronization – RBS

Um nó envia uma mensagem $m \Rightarrow$ cada nó p grava o tempo T (do próprio nó) em que recebeu a mensagem m .

Dois nós podem comparar os seus tempos T e determinar o seu offset relativo. (usando uma média)

• Isto leva atrasos...
(emissão e receção)

⇒ Os próprios relógios
dos nós... por isso nós
fazemos a média com o
relógio base

- GPS
- Sistema Financeiro
- Sincronização da rede elétrica
- HFT - High Frequency Trading

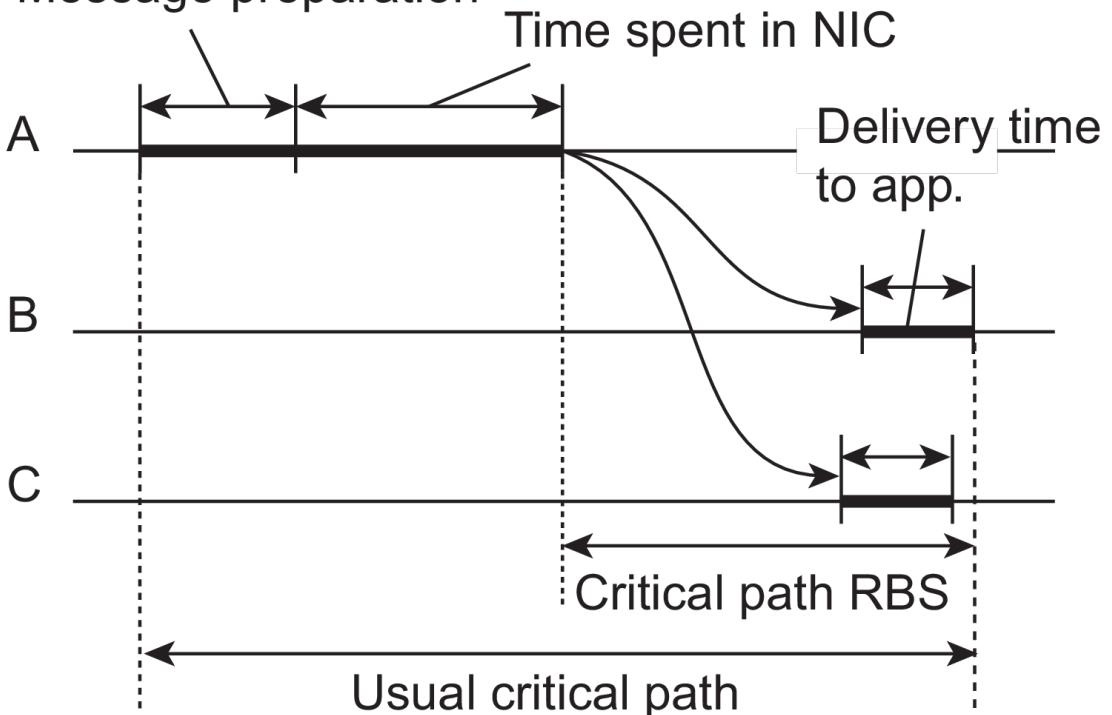
ocorre no meio da
compra-venda



Eles ficam mesmo do
 lado do data center
 da bolsa, e conseguem
 ver a mensagem e fazer
 o trade (o relógio tem de estar certíssimo)

Para criarem
os ondas eles
têm de estes
sincronizados

Message preparation



Relação Aconteceu-Antes

*Só preciso de saber a ordem
em que os operações
foram feitos !!*

- Na maioria das vezes não interessa se todos processos têm a mesma hora, mas sim que eles concordem na ordem pela qual as coisas acontecem. É necessário o conceito de ordenação.
- Relação Aconteceu-Antes:
 - Se **a** e **b** são 2 eventos do mesmo processo, e **a** antecede **b**, então $a \rightarrow b$.
 - Se **a** é o emissor de uma mensagem, e **b** é o receptor da mesma mensagem, então $a \rightarrow b$.
 - Se $a \rightarrow b$ e $b \rightarrow c$ então $a \rightarrow c$
- É também este o conceito de ordenação parcial de eventos em sistemas de processos concorrentes.

"a" antecede "b"

transposição

*Só precisamos de saber a ordem
"Aconteceu - Antes"*

Relógios Lógicos

- Como é que mantemos uma visão global do comportamento do sistema através de relações aconteceu-antes ?
- Associamos um timestamp $C(e)$ a cada evento e , satisfazendo as seguintes propriedades:
 - Se a e b são 2 eventos no mesmo processo, e $a \rightarrow b$, então exigimos que $C(a) < C(b)$
Se $a \rightarrow b$: $C(a) < C(b)$
↑ Um evento que segue tem um timestamp maior!
↓
 - Se a corresponde ao envio da mensagem m , e b à recepção dessa mensagem, então $C(a) < C(b)$
Como criamos os timestamps?
- Se não existe um relógio global, como fazer o timestamp?
 - Mantém-se a consistência através de um conjunto de relógios lógicos, 1 por processo

Solução: Relógios lógicos!

Relógios Lógicos (Lamport)

↳ Relógio só marca quando acontece um evento

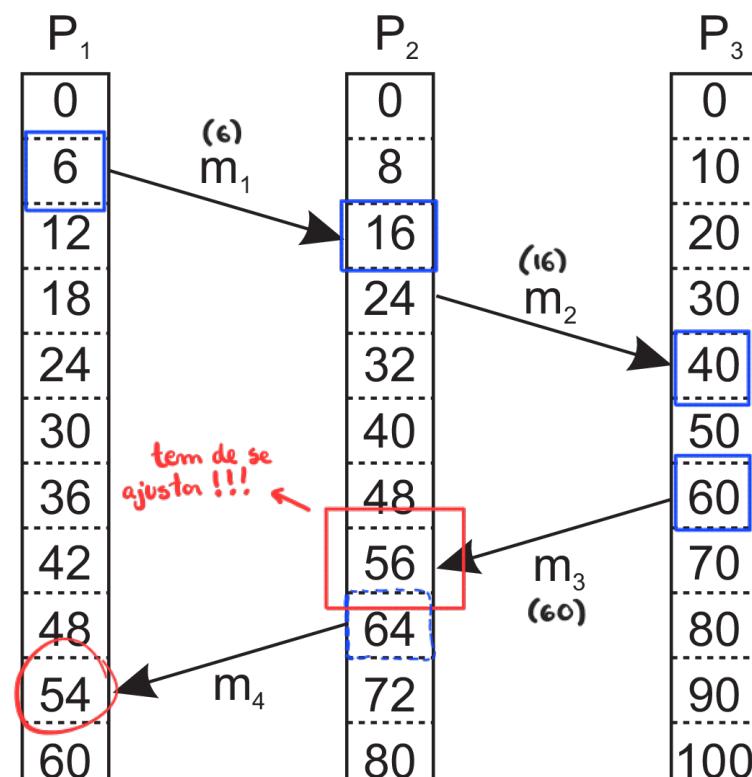
↳ Relógio é uma analogia a um simples contador!

- Cada processo P_i mantem um contador C_i e ajusta este contador
 - Para cada novo evento que ocorre em P_i , C_i é incrementado em 1.
 - Cada vez que uma mensagem m é enviada pelo processo P_i , a mensagem recebe um timestamp $ts(m) = C_i$. !
 - Sempre que uma mensagem m é recebida por um processo P_j , P_j ajusta o seu contador local C_j a $\max(C_j, ts(m))$ e executa o 1º passo antes de passar m à aplicação.

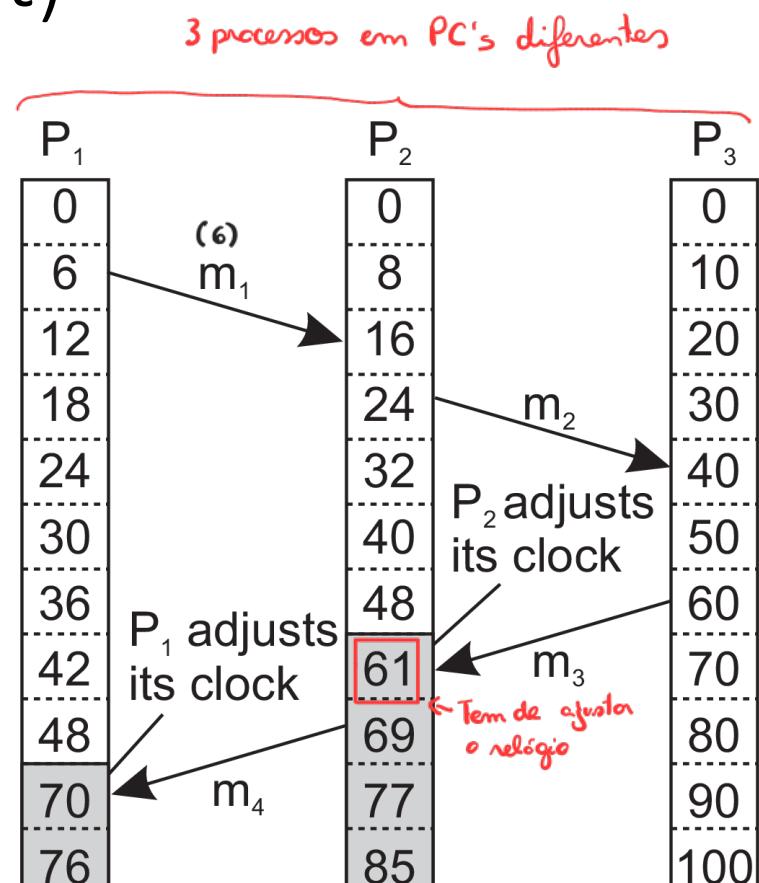
Ou seja, se receber um maior ele ajusta-se!

Incrementa na receção! //

Relógios Lógicos (Lamport)

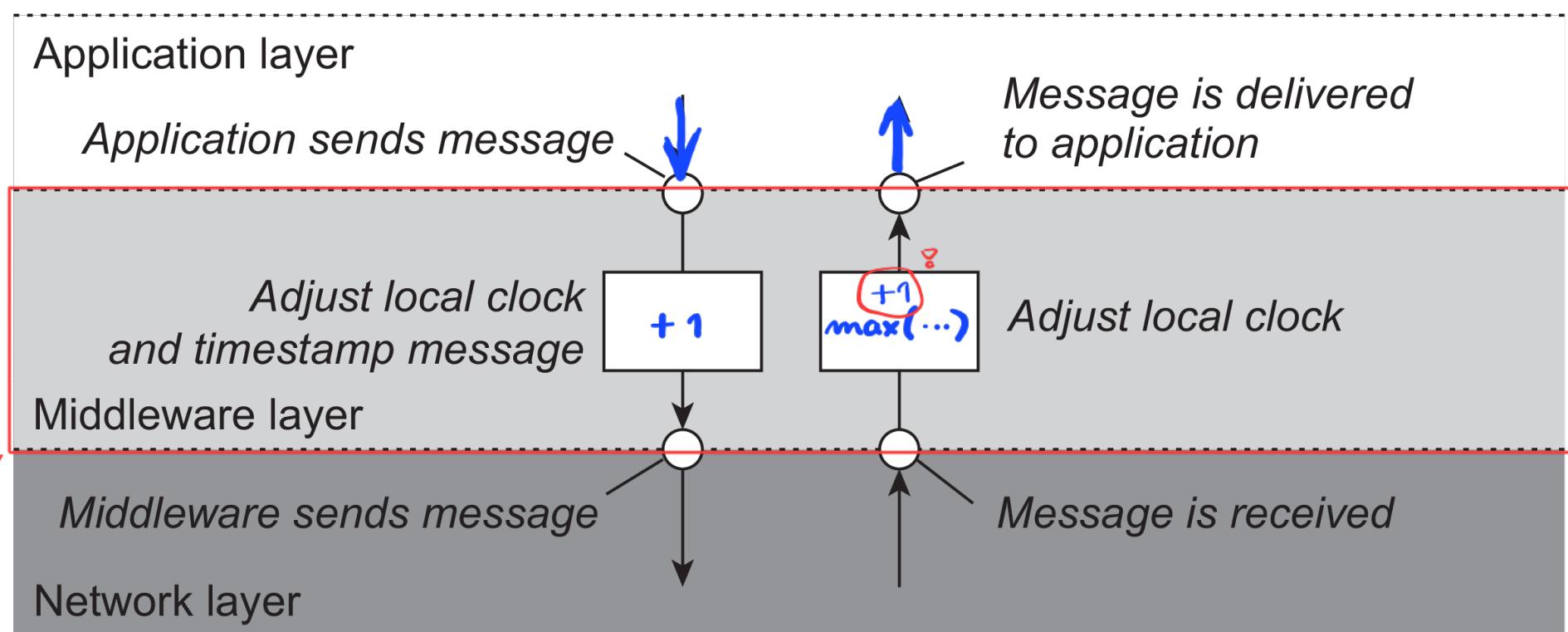


Intervinente os relógios não avançam todos à mesma velocidade!



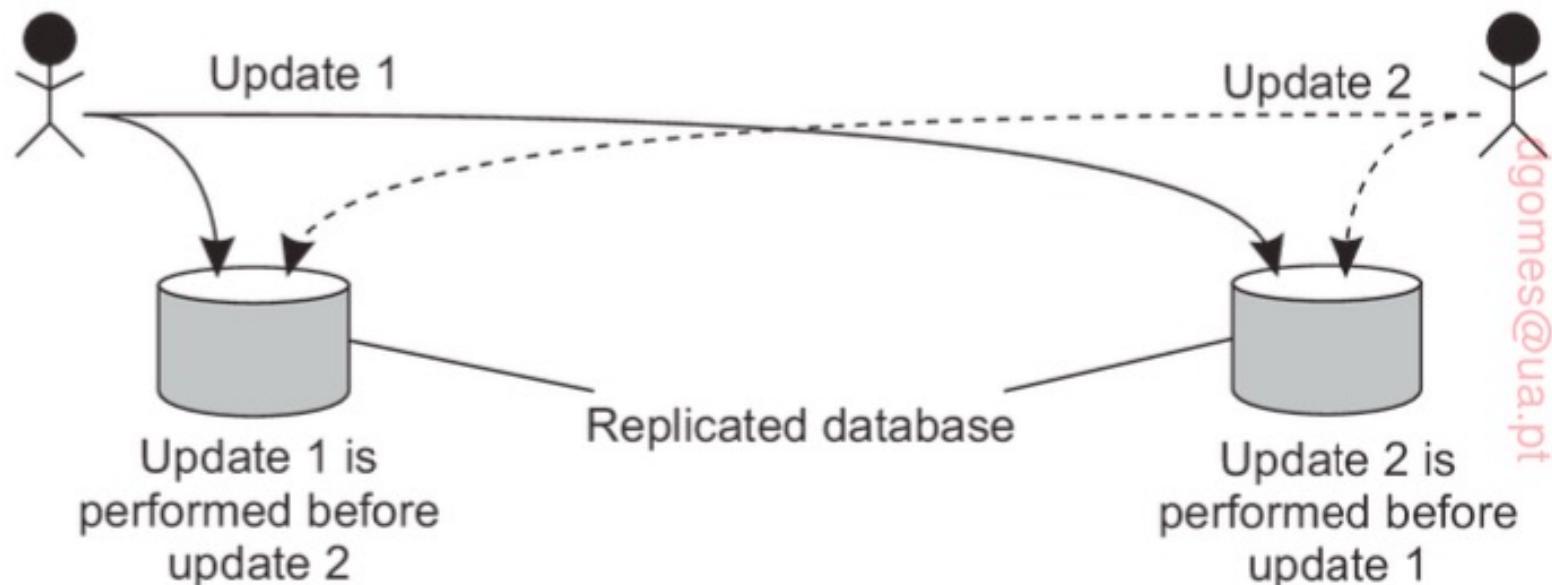
Vai ordenando...
"drifts" (variação no tempo)

Relógios lógicos (2)



Updates que levam a inconsistências

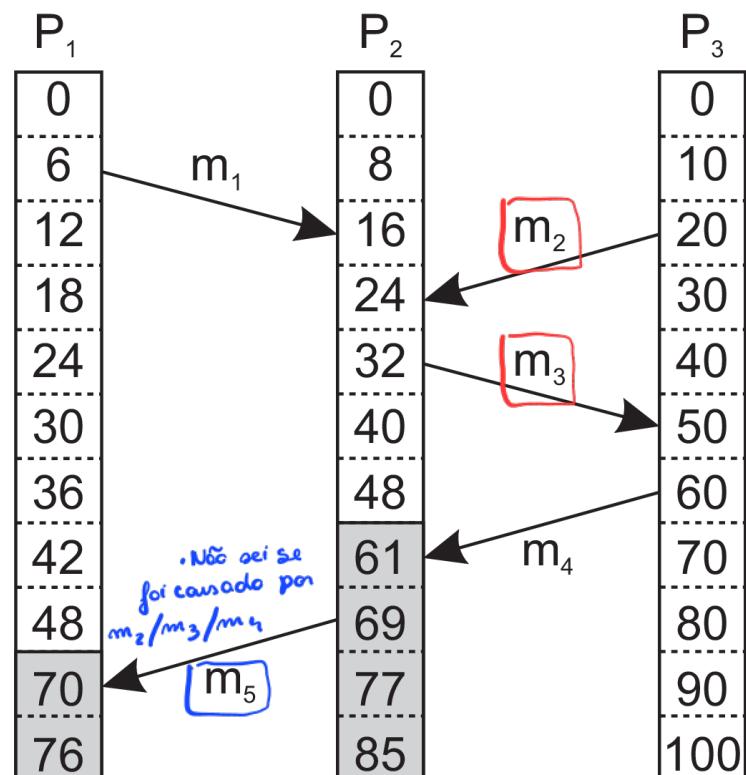
Base de dados distribuída



- Inconsistência: Quem é que chega primeiro para levantar o dinheirinho

Relógios lógicos (conclusão)

! → Com os relógios lógicos → LAMPORT
temos uma relação de tempo sem consequência



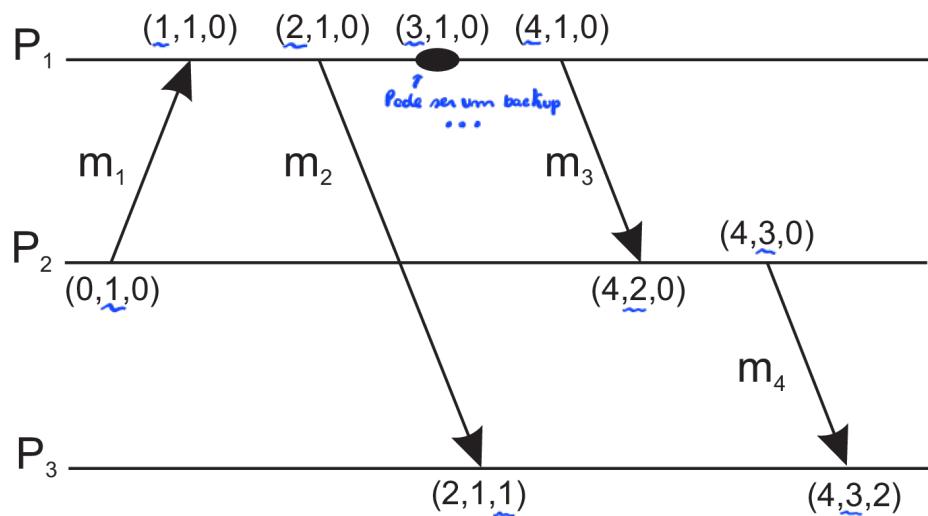
- Relogio de Lamport não garante que se $C(a) < C(b)$ existe uma causa efeito de **a** sobre **b** ($a \rightarrow b$)
- PROBLEMA!
- Evento **a**: m_1 é recebido a $T=16$
 - Evento **b**: m_2 é enviado a $T=20$
- Não houve consequência
- Não podemos concluir que **a** causa **b**.

→ Relógios lógicos não garantem a consequência,

Solução!

(Vector Clocks)

Relógio vectorial



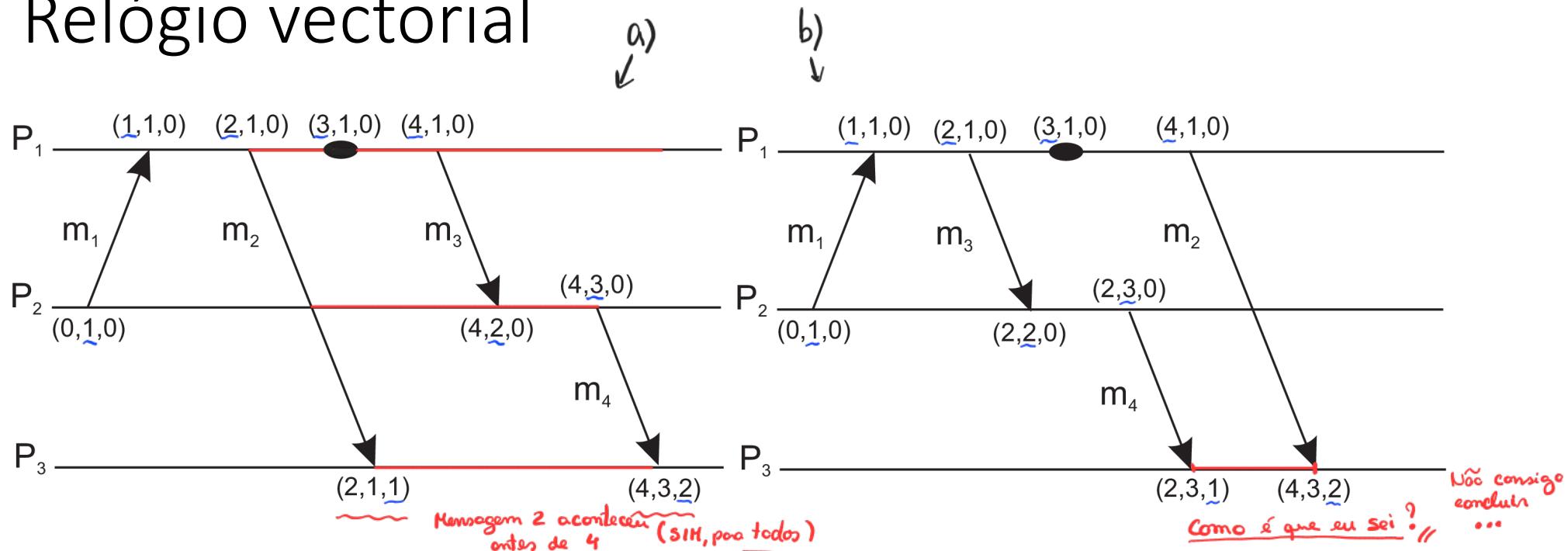
Para m processos precisamos de vetores de m dimensões

(Permite que cada processo tem visibilidade nos outros processos!)

- Antes de executar um evento P_i , executa $\underline{VC_i[i] = VC_i[i] + 1}$ → Só incrementa o contador dele!
- Quando o processo P_i envia a mensagem m para P_j , ele coloca no timestamp $ts(m)$ o valor VC_i pós execução do passo anterior
- Após receber a mensagem m , P_j guarda $\underline{VC_j[k] = \max(VC_j[k], ts(m)[k])}$ para cada k , seguindo-se a execução do passo inicial, e só depois a entrega à aplicação.

CUIDADO! incrementa na recepção e no envio !

Relógio vectorial



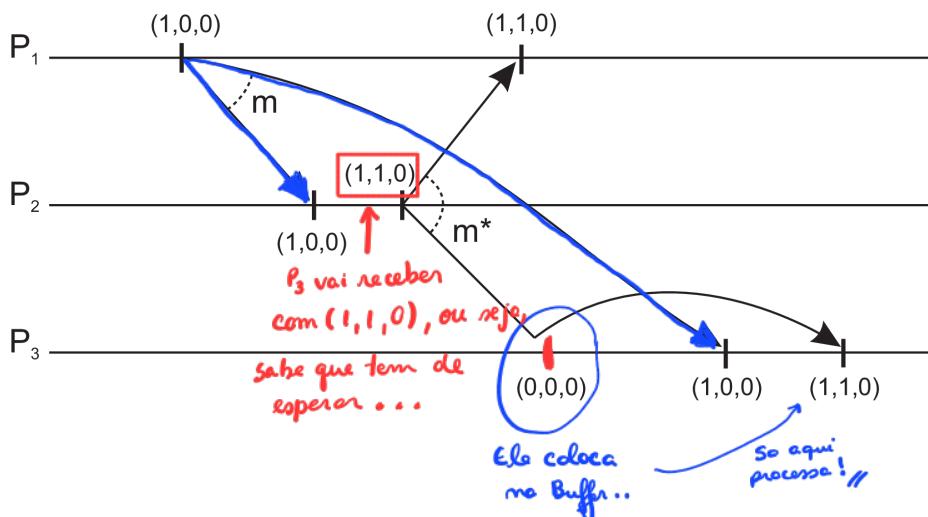
Situação	$ts(m_2)$	$ts(m_4)$	$ts(m_2) < ts(m_4)$	$ts(m_2) > ts(m_4)$	Conclusão
a	$(2, 1, 0)$	$(4, 3, 0)$	Sim	Não	m_2 aconteceu antes de m_4
b	$(4, 1, 0)$	$(2, 3, 0)$	Não	Não	m_2 e m_4 podem estar em conflito



Ainda não chega!

(preciso de enviar)
para todos!

Relógio vectorial



Todos as mensagens têm de chegar a todos

- É necessário garantir que m só é entregue após todas mensagens casuísticas terem sido entregues.

de causa!

- P_i incrementa $VC_i[i]$ apenas quando envia a mensagem, P_j ajusta VC_j quando recebe a mensagem

$$ts(m)[i] = VC_j[i] + 1$$

$$ts(m)[k] \leq VC_j[k] \text{ para todo } k \neq i$$

Mos temos de garantir que vou enviar para TODOS
isto pode ser muito difícil...

→ E quando existe uma falha?

Queremos continuar em
relógios lógicos!

Problema: Precisamos de coordenação para
garantir causalidade!

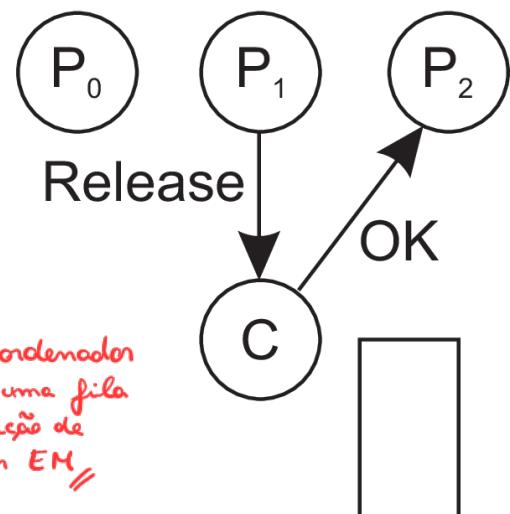
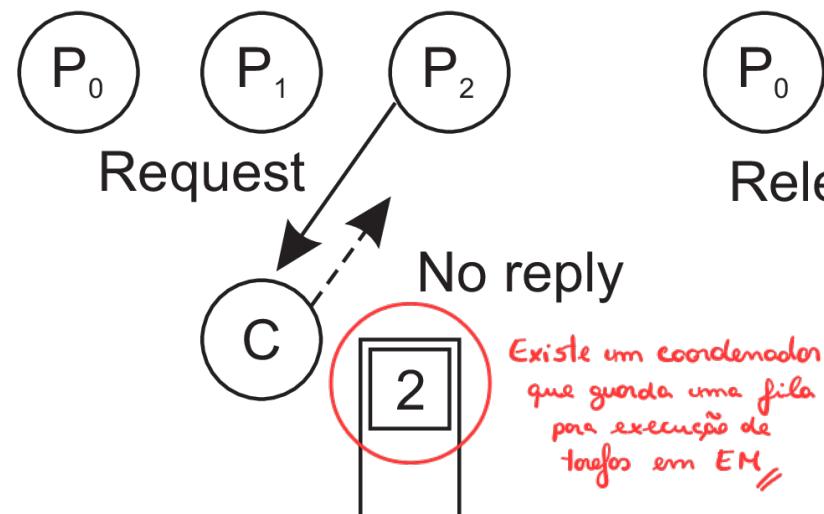
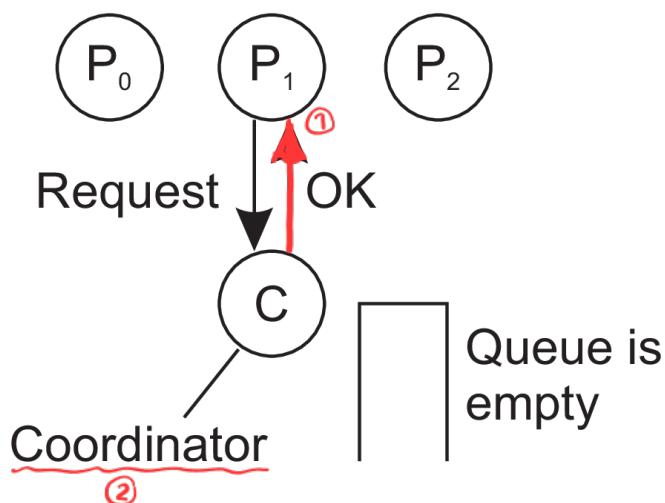
Algoritmos de exclusão mútua

- Baseados em Permissões
- Baseados em Token

Exclusão Mútua - Centralizado

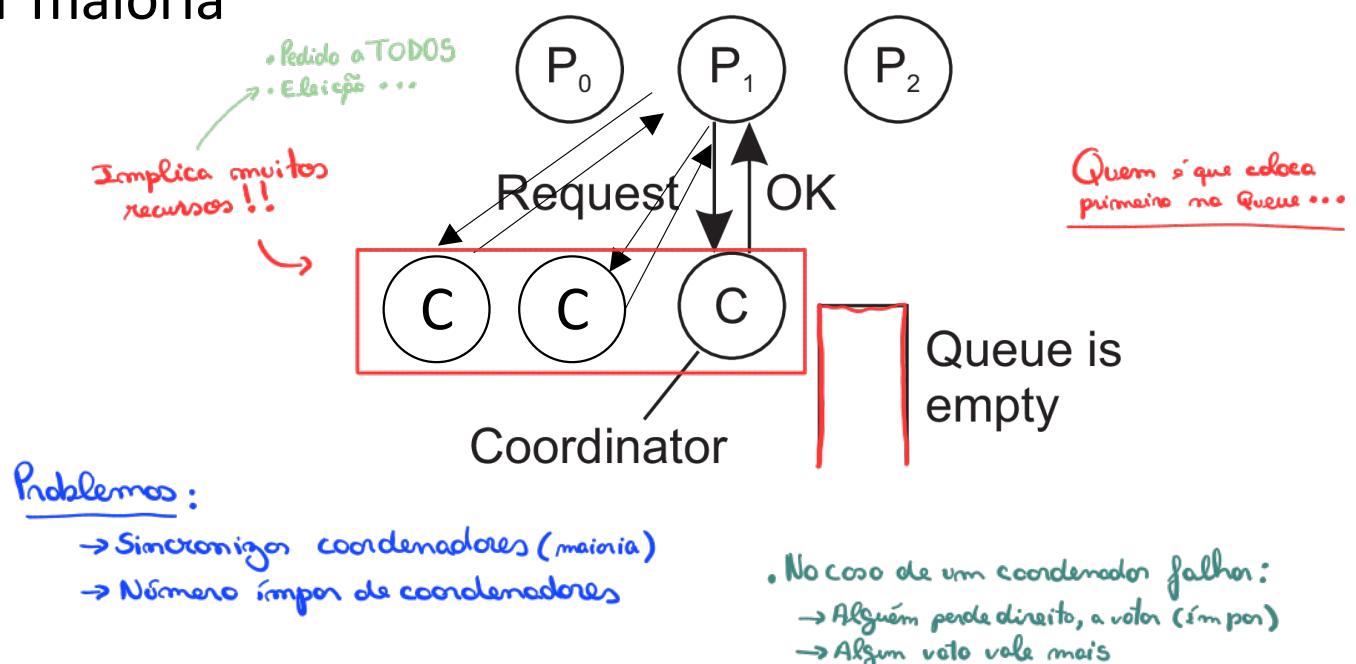
Problemas:

- ① Se um dos processos falha dentro da exclusão mútua \Rightarrow Deadlock
- ② Algoritmo é centralizado!

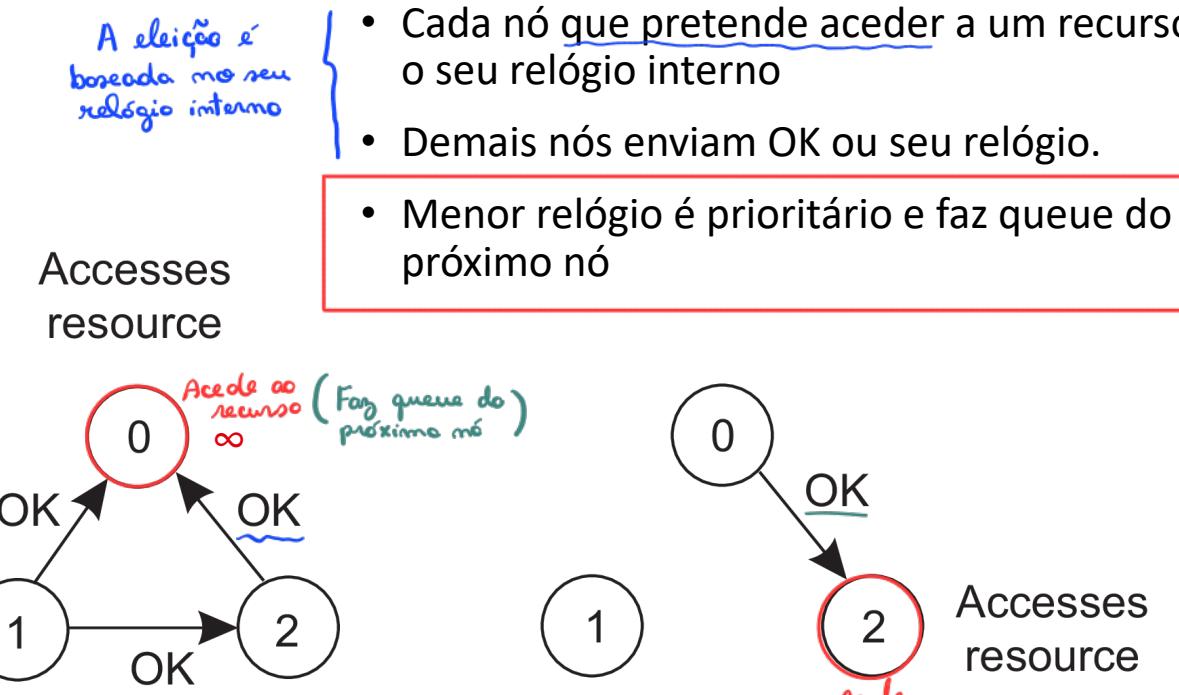
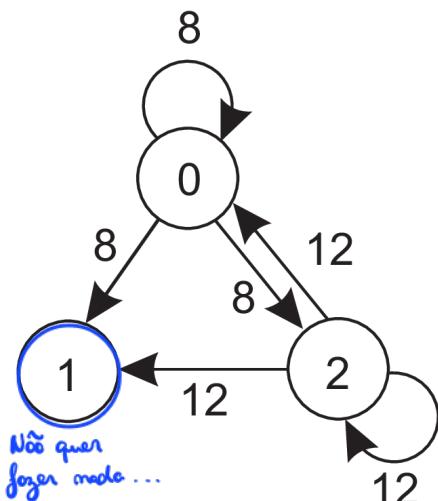


Exclusão Mútua - Descentralizado

- Vários coordenadores,
- Acesso é atribuído por maioria de votos

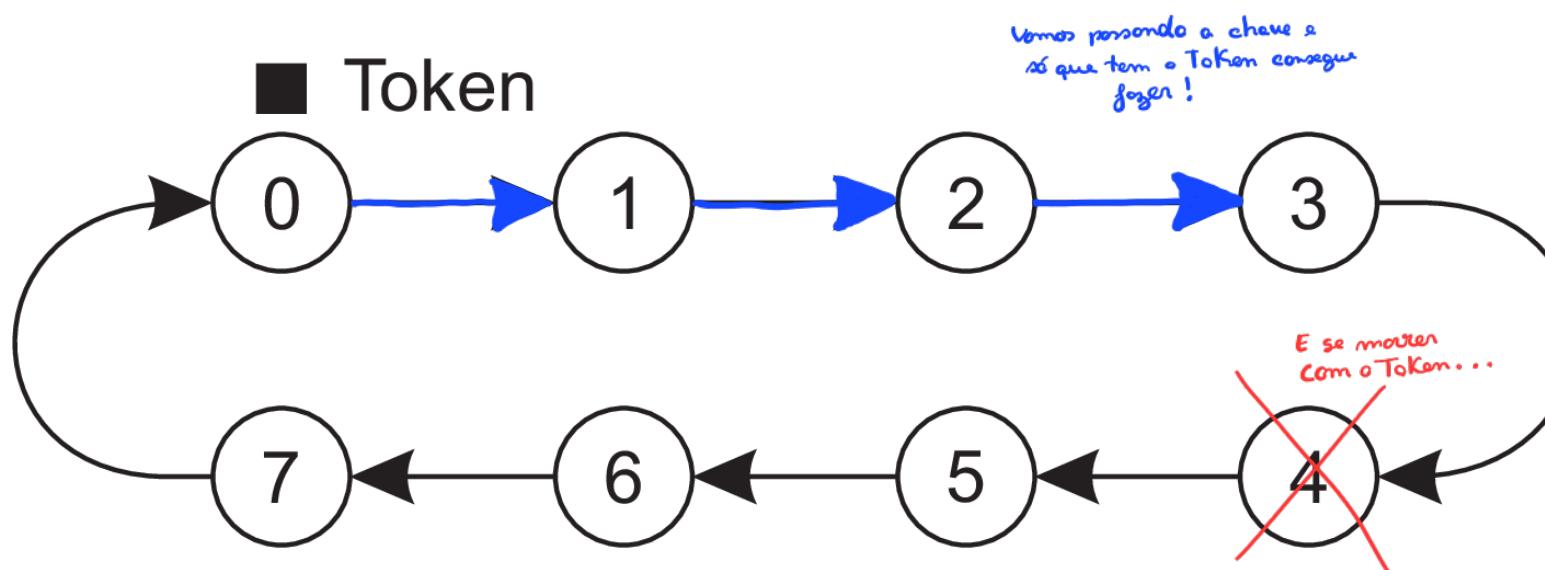


Exclusão Mútua - Distribuído



Exclusão Mútua – Token Ring

Tipo estepeia!



- Token Ring pode demorar a chegar (ou até mesmo nunca chegar)
- Token duplicado

Comparação

Algoritmo	Mensagens por entrada/saída	Atraso antes da entrada (em tempos de mensagem)	Problemas
Centralizado	3	2	<u>Coordenador crasha</u> ← único ponto de falha
Descentralizado	$2 m k + m$, $k = 1, 2, \dots$	$2 m k$	<u>Fome</u> , <u>baixa eficiência</u> → muito lento...
Distribuído	$2(n-1)$ → envio e recebe OK	$2(n-1)$	<u>Crash de um qq processo</u>
Token Ring	1 a ∞ Se o anel estiver aberto...	0 a $n - 1$	<u>Perda token</u> Crash de um processo que tem o token

n: número de processos

m: número de coordenadores

k: número de tentativas

Algoritmos de Eleição

Exclusão mútua:

- Centralizada
- Descentralizada
- Distribuída



- Quando um algoritmo precisa que um processo aja como coordenador. **Como selecionar o coordenador de forma dinâmica?**
- Em muitos casos o coordenador é fixo (único ponto de falha)
- Se coordenador for escolhido de forma dinâmica podemos dizer que o sistema é centralizado?
 - centralizado durante a coordenação
 - descentralizado no processo de eleição*{ Hybrid!*
- Um sistema completamente distribuído (sem qualquer coordenador), será sempre mais robusto do que um centralizado/coordenado ?

→ Situação de Fome!
→ Pode não ser robusto!
→ Pode não ser seguro!

Vai depender dos
casos ...

Problema: Resistência a Falha! //

Como eleger um coordenador?

Pressuposto iniciais

Nota: devem concordar em quem foi eleito!

- Todos processos têm um identificador único (id)
- Todos processos conhecem os id's de todos processos no sistema (mas não o seu estado – ligado/desligado)
- Eleger significa identificar o processo com o id mais elevado que esteja ligado

⇒ Não preciso saber se votaram ou não...
o que tenho de saber é quantos podem votar

↑ ↳ ID mais elevado é o Coordenador!

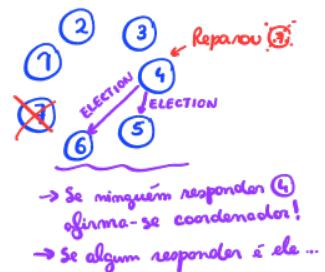
Regra na Eleição por Bullying

Eleição por bullying

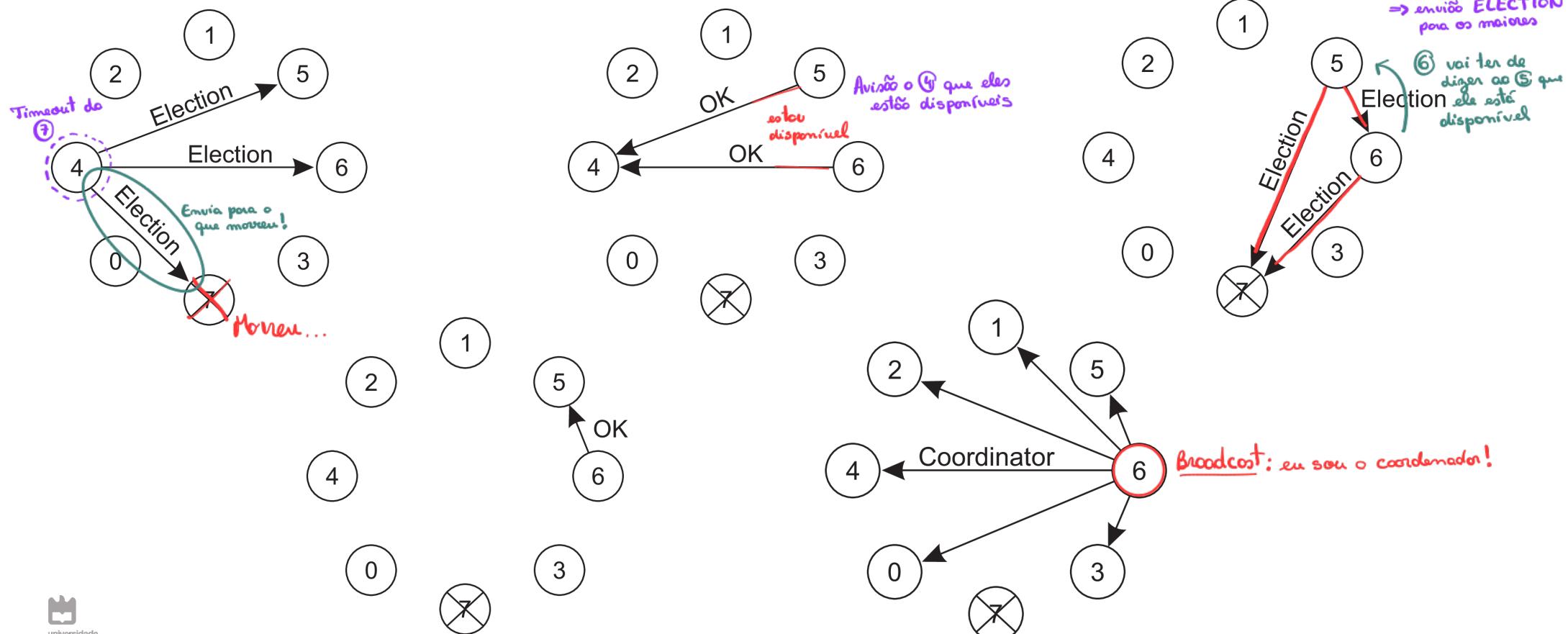
- Considerar N processos $\{P_0 \dots P_{N-1}\}$ e que $\text{id}(P_k) = k$. Quando um processo P_k repara que o coordenador não responde mais a pedidos, inicia uma eleição:

- P_k envia uma mensagem ELECTION a todos processos com identificadores mais elevados: $P_{k+1}, P_{k+2}, \dots, P_{N-1}$.
Por os Mais elevados, pois os que tiverem um mais pequeno não vão ganhar.
- Se nenhum responder, P_k ganha a eleição e torna-se no coordenador
↳ Sou eu!
- Se um dos processos com id superior responder, este assume a posição e o trabalho de P_k termina.

Timeout do coordenador!
↓



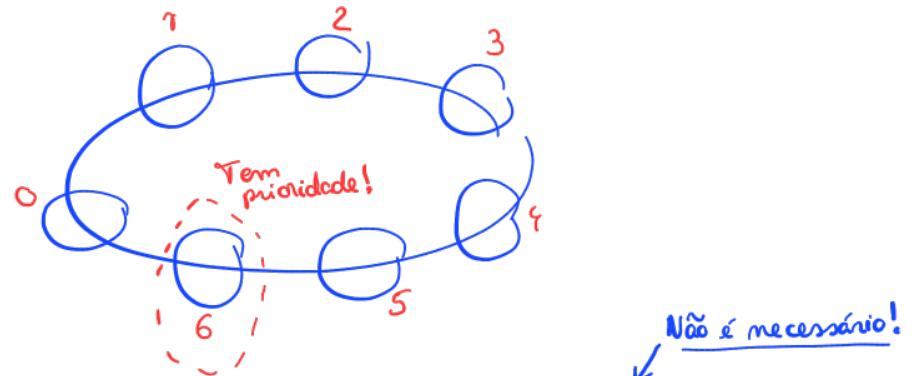
Eleição por bullying



Problemas:

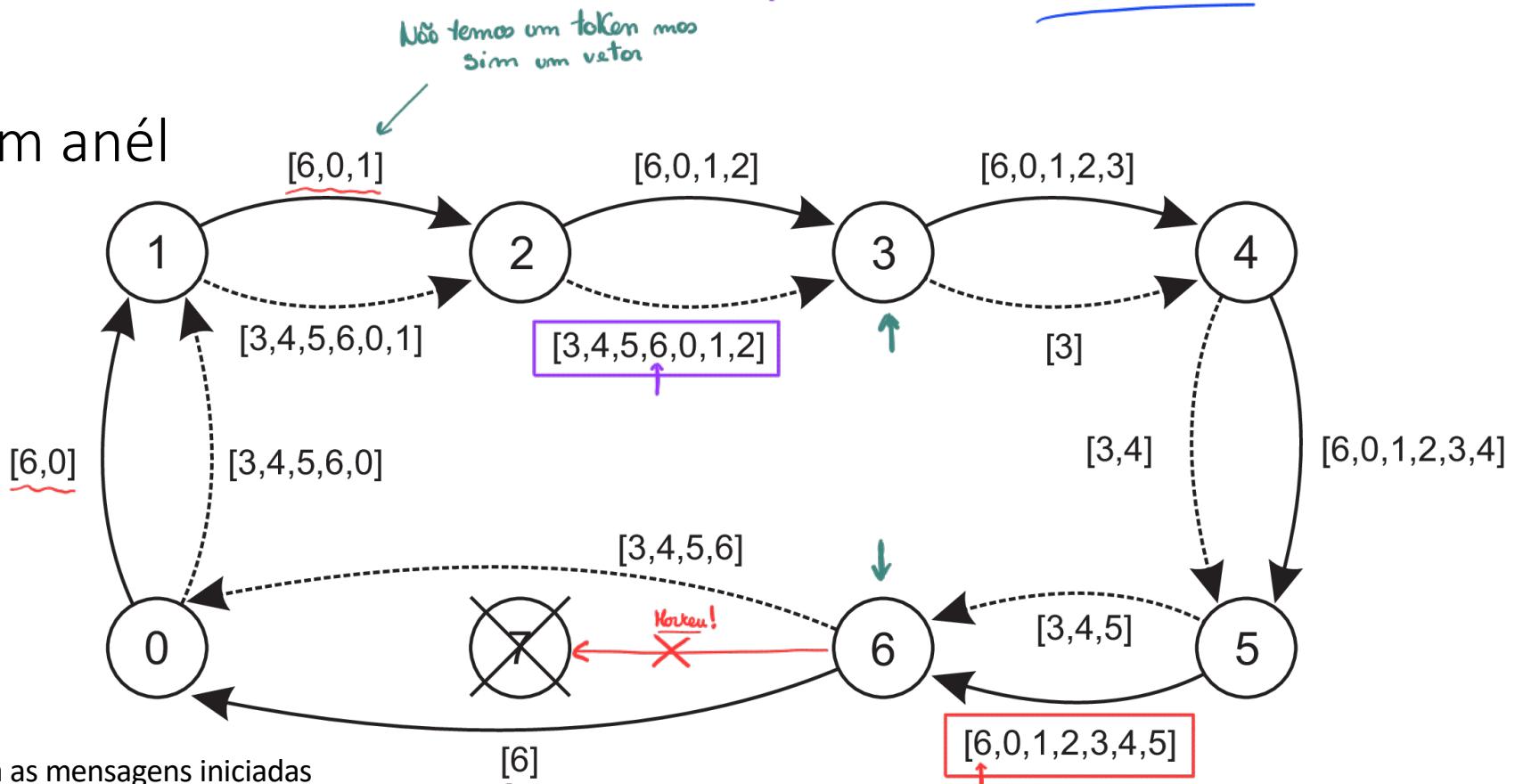
- Perte de pacotes
- Número de rondas precisas são muitos
- Todos conhecem todos! //

Eleição num anél



- Prioridade de um processo é obtida pela organização dos processos num anél lógico. O processo com prioridade mais elevada é eleito o coordenador.
 - Qualquer processo inicia uma eleição através do envio de uma mensagem ao seu sucessor. Se o sucessor estiver desligado, a mensagem é passada ao sucessor seguinte.
 - Se uma mensagem é passada, o emissor adiciona-se a si próprio à lista. Quando a mensagem regressa ao emissor, todos tiveram a oportunidade de se fazerem anunciar.
 - Quem inicia ^{O que iniciou} envia uma mensagem de coordenação através do anel contendo a lista de todos processos presumidos ligados. O processo com prioridade mais elevadas é eleito coordenador.
 - Quando ve que na lista tem o seu id ele círcula uma nova mensagem COORDINATOR para informar que o novo coordenador é o maior id na lista,,

Eleição num anel



A linha sólida denota as mensagens iniciadas por P_6

A linha tracejada denota as mensagens iniciadas por P_3

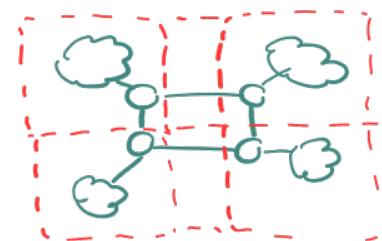
Eleição em sistemas de grande-escala

- Caso dos **super-nós** nas redes P2P
- Nós normais necessitam de baixa latência a aceder aos super-nós
- super-nós devem estar distribuídos de forma uniforme pela overlay
- Devem existir numa proporção fixa em relação ao tamanho da rede
- Um super-nó não deve servir mais do que um número fixo de nós

Regras que dependem dos requisitos da rede!

- Eleição por Bullying
- Eleição num onel

Problemáticos...

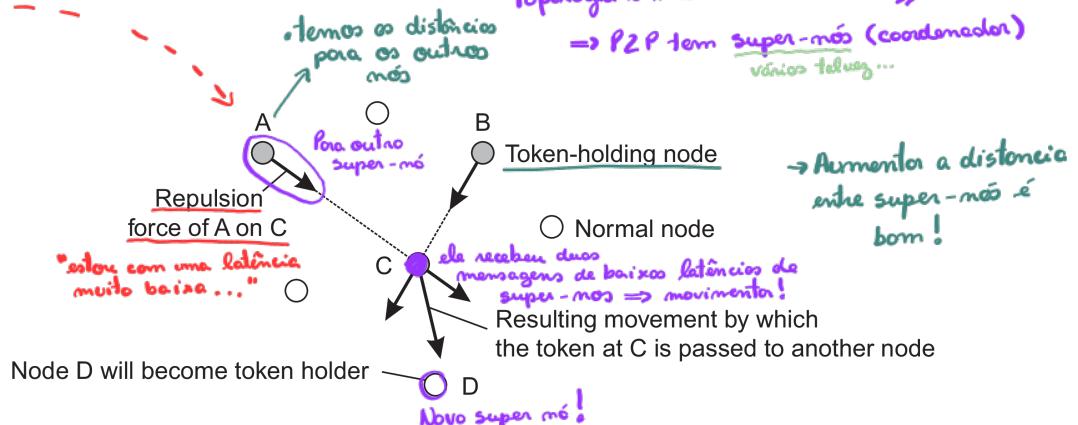


• Orientados à aplicação

- ⇒ latência
- ⇒ localização
- ⇒ importância (autenticação...)

Topologia é medida em latência!

⇒ P2P tem super-nós (coordenador) vários lugares...



• Periodicamente correm um protocolo que envia a distância do super-nó

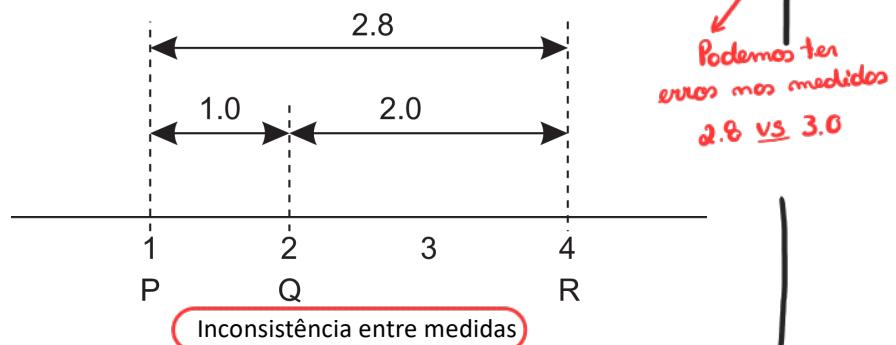
- Outra forma seria mandar para quem tem o CPU mais livre!

Posicionar nós

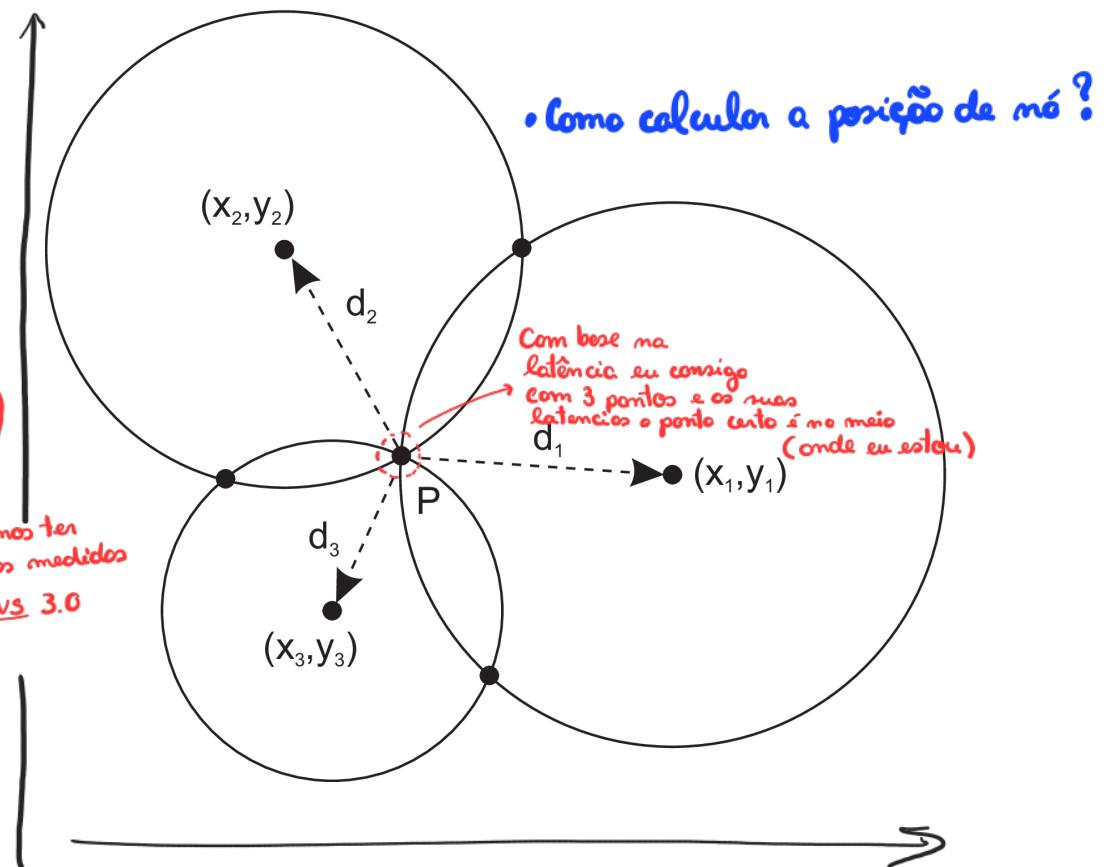
- Num sistema distribuído de larga escala, em que os nós se encontram dispersos numa WAN (leia-se Internet), é comum o sistema ter em atenção a noção de proximidade ou distância.
- Para tal é necessário determinar a localização do nó.
latência!

Calcular posição

- Um nó P necessita de $d+1$ marcos para poder calcular a sua própria posição num espaço d -dimensional.
- GPS ? *em 3D, quatro esferas! ou mais! (número ímpar!!!)*
- Sinal de AP (Wifi/Bluetooth/etc)



- O SpeedTest verifica a latência com o Servidor deles, e assim eles conseguem nos localizar
- GigaPrix
 - Antes os dados iam para o centro da Europa ...
 - ... e depois para a Espanha
 - ... e finalmente para a França
- Agora temos um servidor em Lisboa (Ist) em que todos os ligações em Portugal se ligam



Gestão de tarefas ..

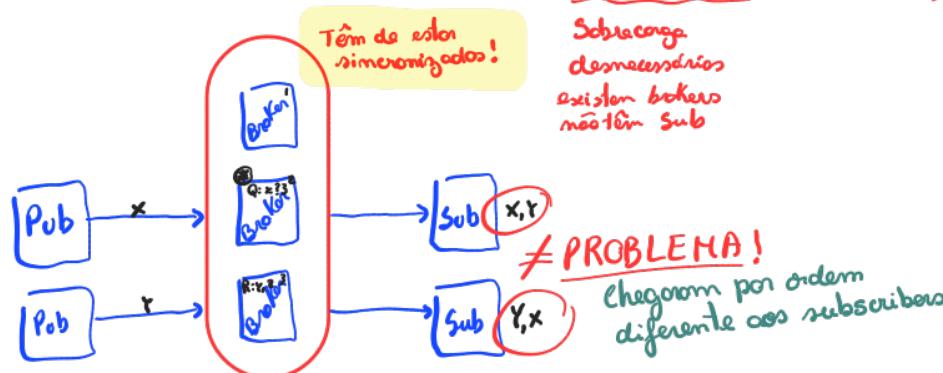


Correspondência de Eventos Distribuídos



- Correspondência de Eventos ou Filtragem de Notificações é a tarefa principal de um sistema Publish-Subscribe.
- O pressuposto é que o sistema distribuído é capaz de fazer correspondência entre subscrições e eventos.
- Solução fácil:
 - Nó centraliza subscrições e recebe todos eventos
- Solução distribuída
 - Subscrições são distribuídas por diversos nós
 - Eventos são roteados para os nós correctos (flooding, gossip, routing-filters)

Parecido com
Exclusão Mútua
- Descentralizado



Está no EleARNING!

import pika

→ Broker que implementa em Pub-Sub

• Para Brokers: RabbitMQ

• Para Multiplos Brokers: Kafka

↑
Para sistemas distribuidos!

{ Bom para o
Projeto Final
do Curso !!!

- Primeiro começo a correr o Rabbit
- E executa os Client/Writer/Reader
- O message-broker também tem filhos, ou seja ele guarda as mensagens!