Coordenação

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.
  - Introduz saltos de um segundo (leap second) para compensar o facto dos dias serem cada vez maiores
- Valores UTC são radiodifundidos em onda curta e satélite.
  - Satélite tem precisão na ordem dos 5ms.

Sincronização de relógios

- **Precisão**: Objectivo é manter o desvio entre o relógio de duas máquinas dentro de um limite π
  - $\forall t, \forall p, q : |Cp(t) - Cq(t)| \leq \pi$

  Para Cp(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
  - $\forall t, \forall p : |Cp(t) - t| \leq \alpha$
- Sincronização:
  - **Interna**: relógios precisos
  - **Externa**: relógios exactos

Clock drift

- Especificações de um relógio
  - Razão máxima de divergência (maximum clock drift rate) - **ρ**
  - **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

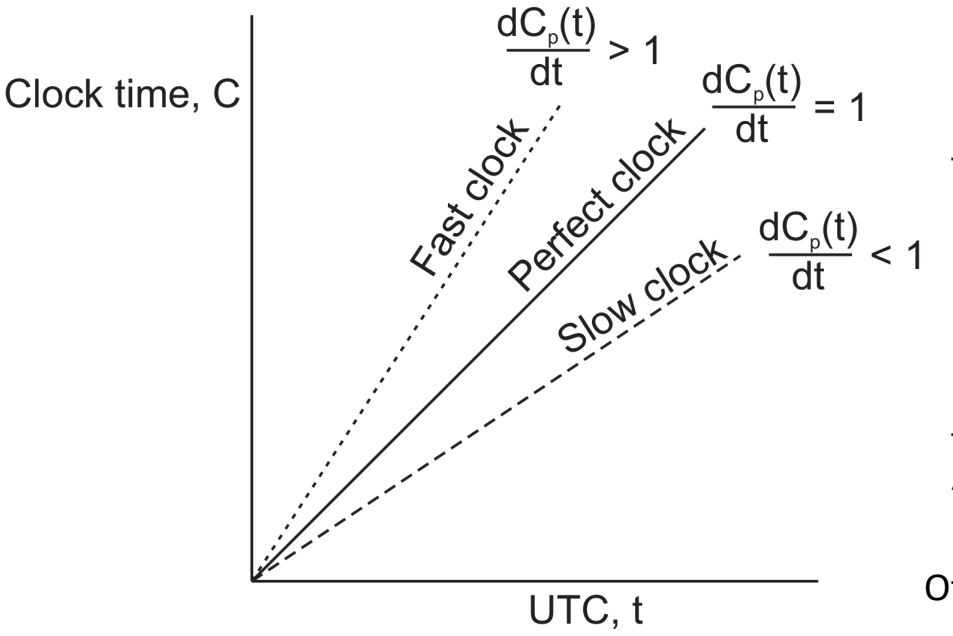
$$\forall\, t : (1-\rho) \leq F(t) \leq (1+\rho)$$

$$C_p(t) = \frac{1}{F}\int_0^t F(t)dt \Rightarrow \frac{dC_p(t)}{dt} = \frac{F(t)}{F}$$

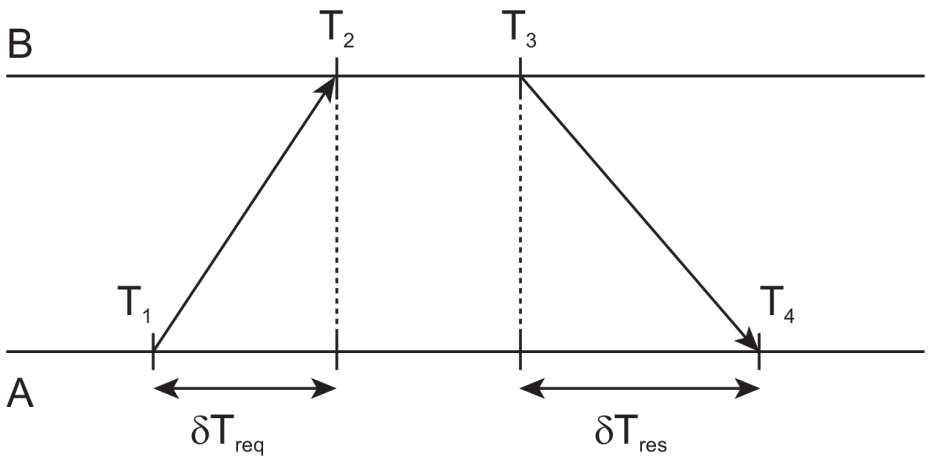
$$\Rightarrow \forall\, t : (1-\rho) \leq \frac{dC_p(t)}{dt} \leq (1+\rho)$$

Desvios de relógio

Adiantados, Certos e atrasados



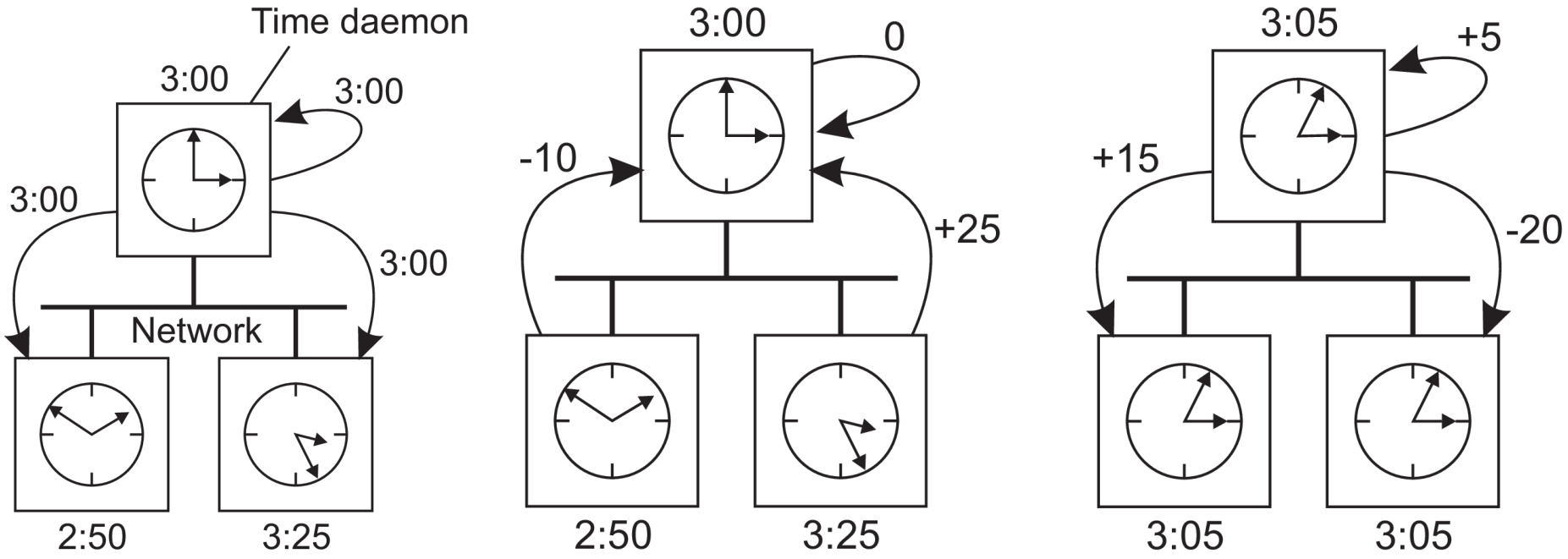
Obter tempo de um servidor (NTP)



Offset/Desvio:

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

Sincronizar sem UTC (algoritmo Berkeley)



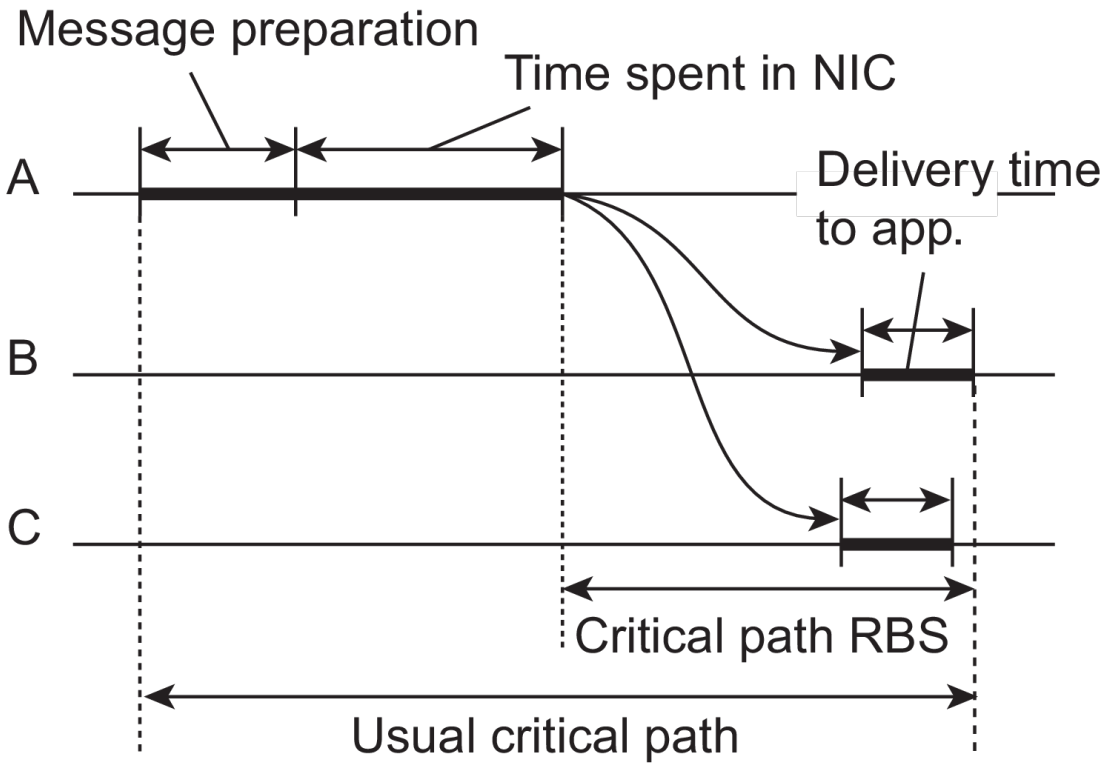
Atenção! Não é permitido atrasar relógio, apenas alterar velocidade do tempo.

Sincronização por difusão

Reference broadcast syncronization – RBS

Um nó envia uma mensagem m => 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)



Relação Aconteceu-Antes

- 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 $\boldsymbol{a \rightarrow b}$.
  - Se **a** é o emissor de uma mensagem, e **b** é o receptor da mesma mensagem, então $\boldsymbol{a \rightarrow b}$.
  - Se $\boldsymbol{a \rightarrow b}$ e $\boldsymbol{b \rightarrow c}$ então $\boldsymbol{a \rightarrow c}$
- É também este o conceito de ordenação parcial de eventos em sistemas de processos concorrentes.

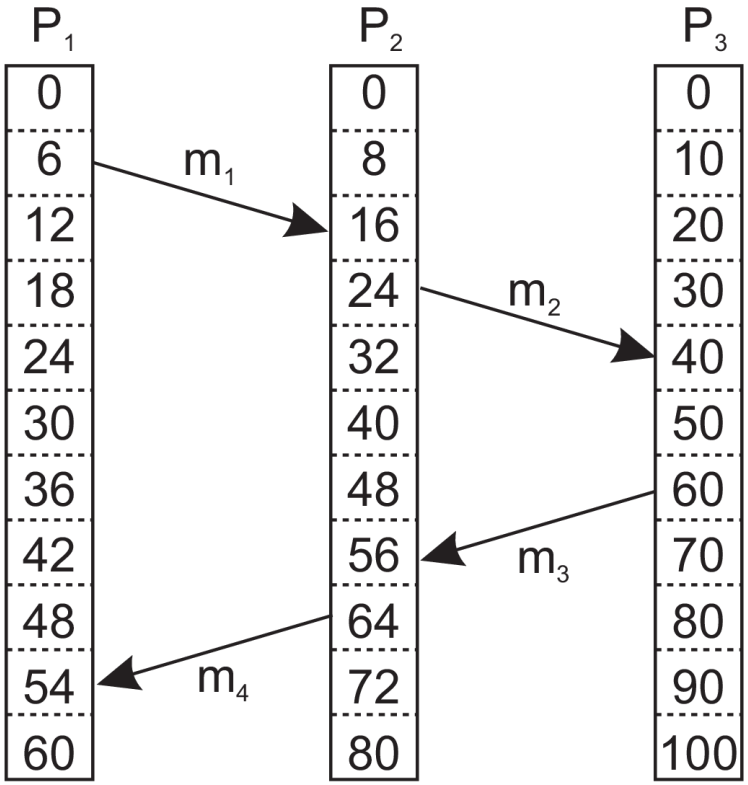
Relógios Lógicos

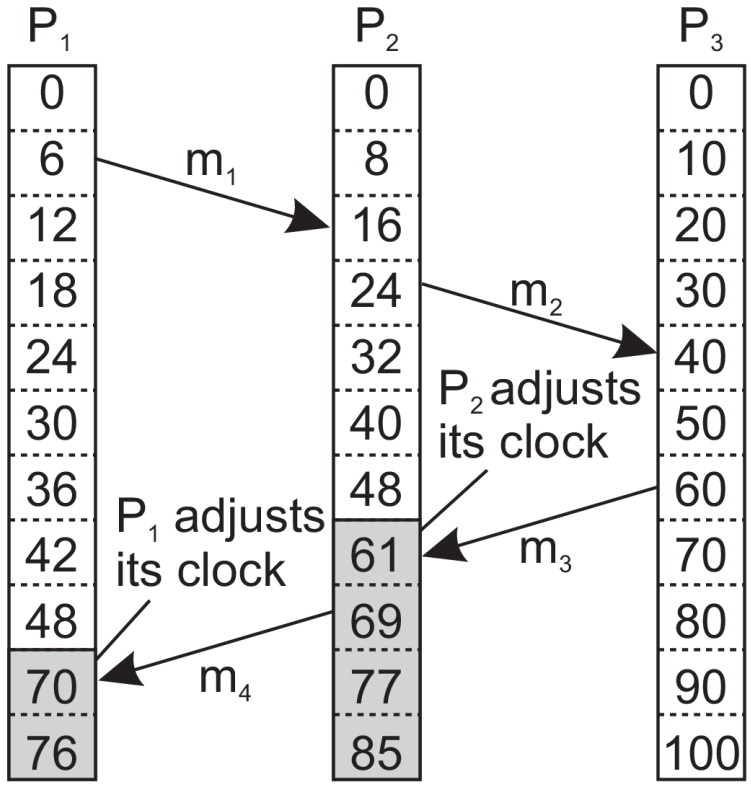
- Como é que mantemos uma visão global do comportamento do sistema através de relações aconteceu-antes ?
- Associamos um *timestamp* **C(e)** a cada evento **e**, satisfazendo as seguintes propriedades:
  - Se **a** e **b** são 2 eventos no mesmo processo, e $\boldsymbol{a \rightarrow b}$, então exigimos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

Relógios Lógicos (Lamport)

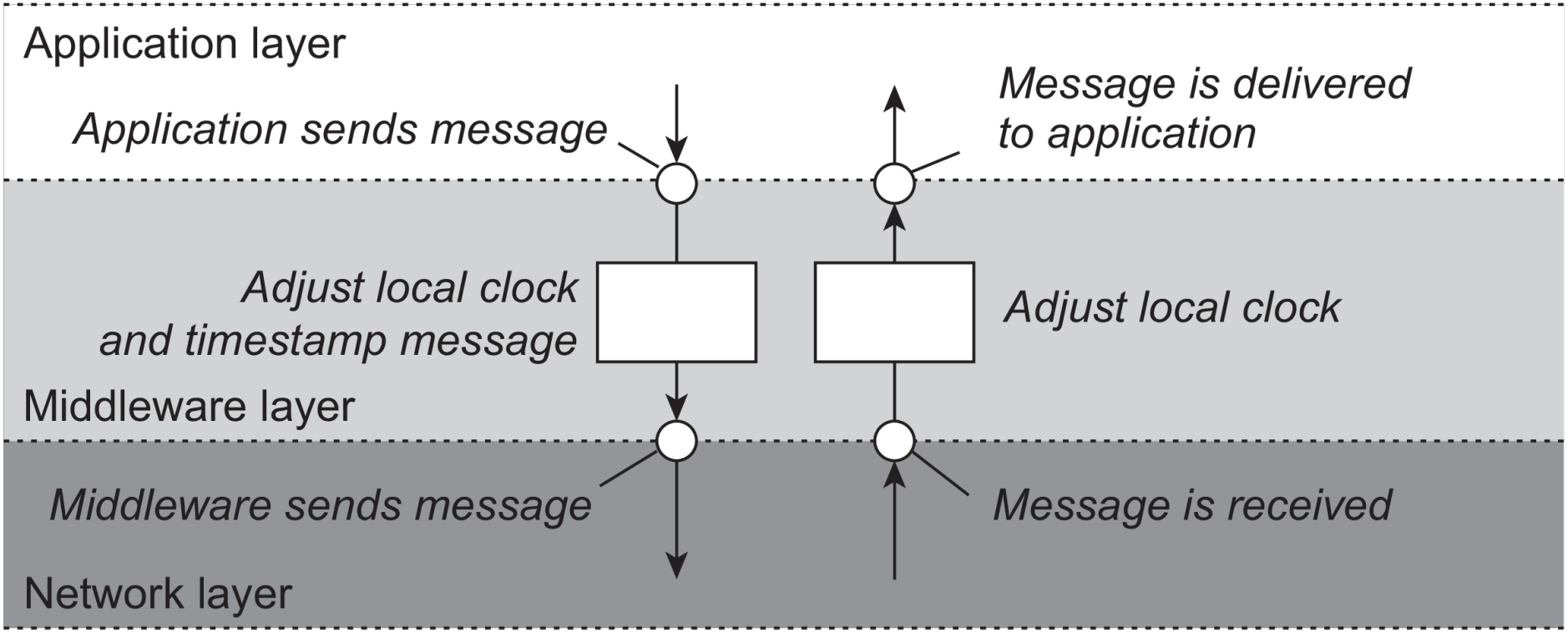
- Cada processo $\mathbf{P_i}$ mantem um contador $\mathbf{C_i}$ e ajusta este contador
  - Para cada novo evento que ocorre em $\mathbf{P_i}$, $\mathbf{C_i}$ é incrementado em 1.
  - Cada vez que uma mensagem m é enviada pelo processo $\mathbf{P_i}$, a mensagem recebe um timestamp $\mathbf{ts(m) = C_i}$.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

Relógios Lógicos (Lamport)

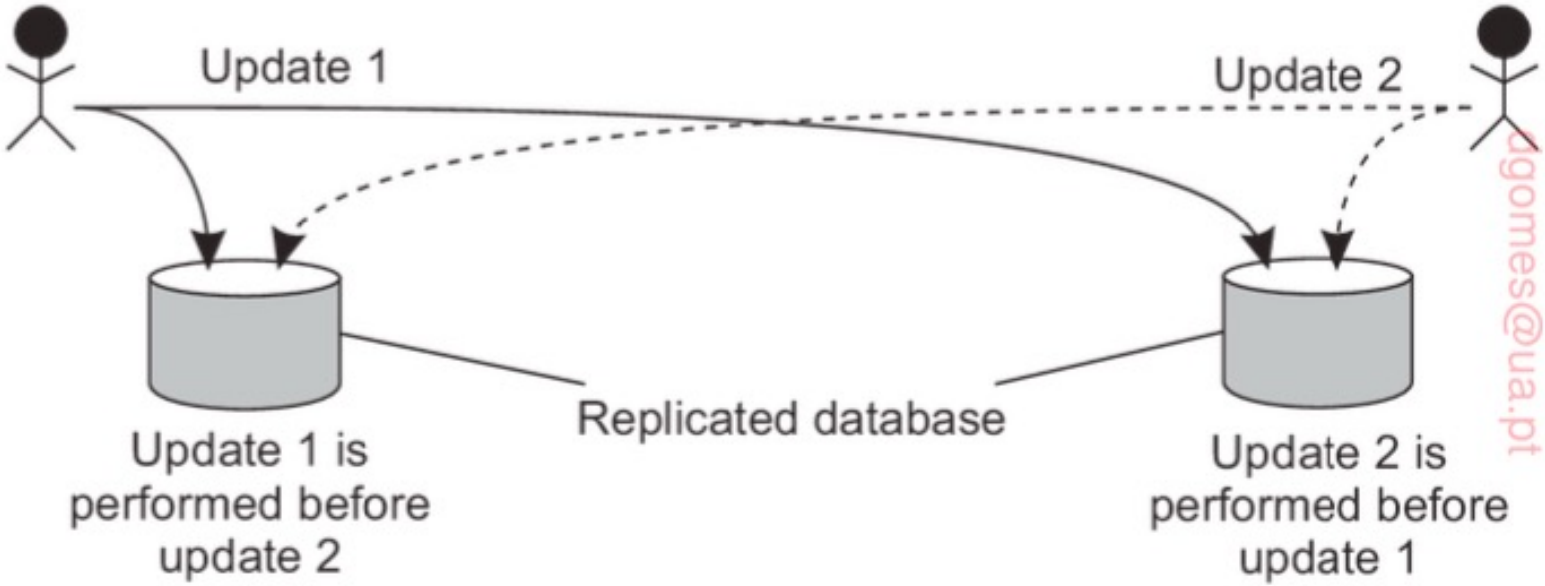




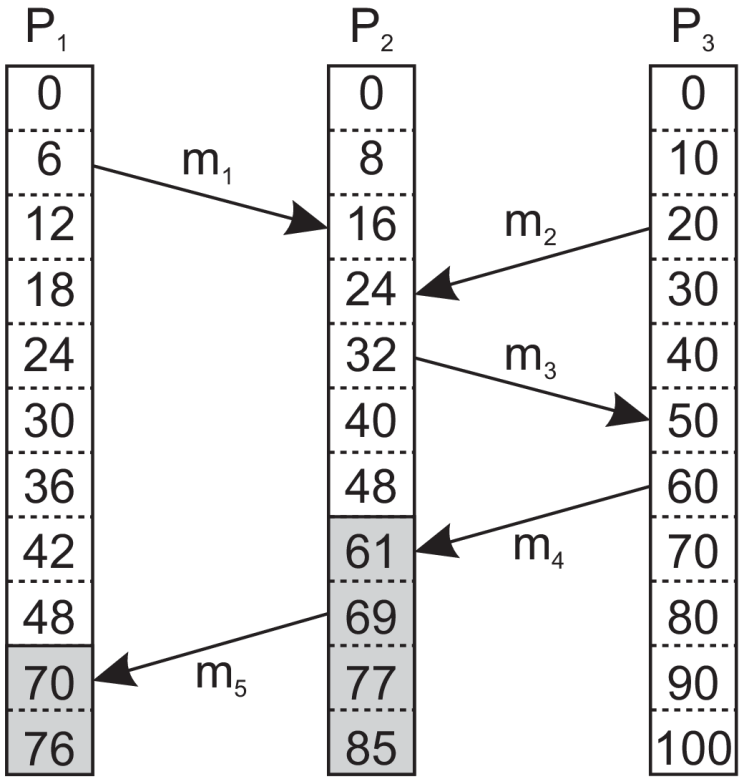
Relógios lógicos (2)



Updates que levam a inconsistências

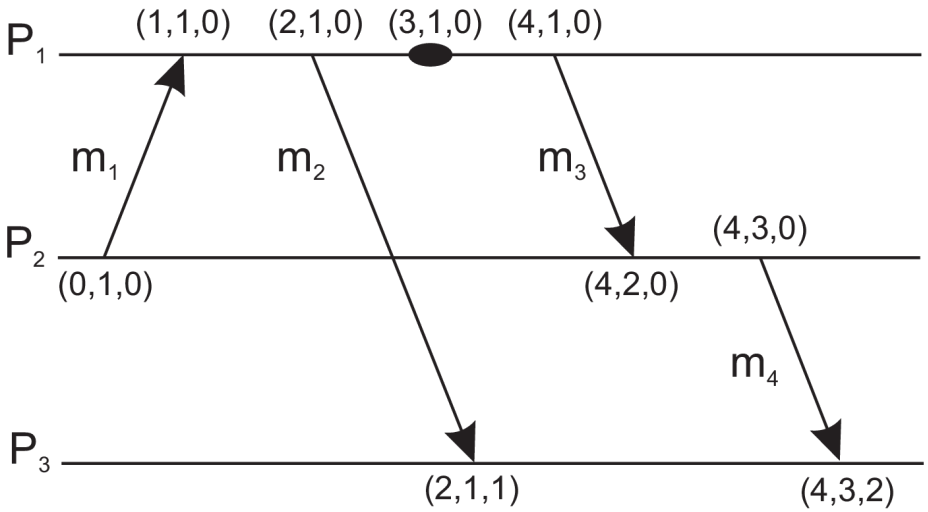


Relógio vectorial (vector clocks)



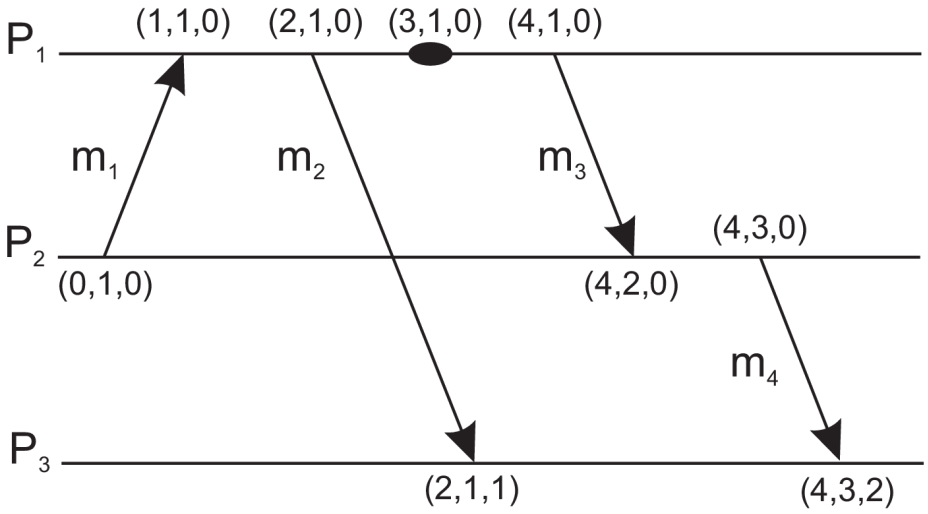
- Relogio de Lamport não garante que se C(a) < C(b) existe uma causa efeito de **a** sobre **b** (**a** -> **b**)
- Evento **a**: m1 é recebido a T=16
- Evento **b**: m2 é enviado a T=20
- Não podemos concluir que **a** causa **b**.

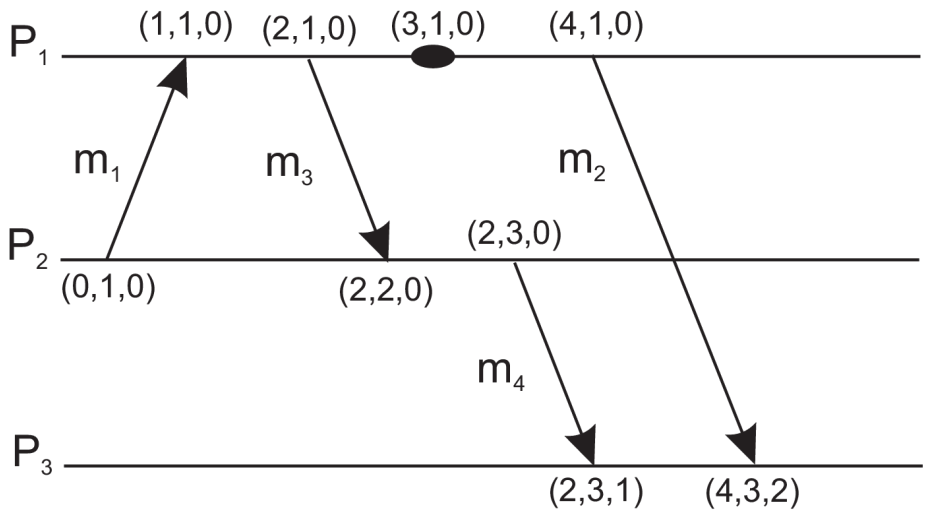
Relógio vectorial



- Antes de executar um evento P_i executa $VC_i[i] = VC_i[i] + 1$
- 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 $VC_j[k] = \max(VC_j[k], ts(m)[k])$ para cada k, seguindo-se a execução do passo inicial, e só depois a entrega à aplicação.

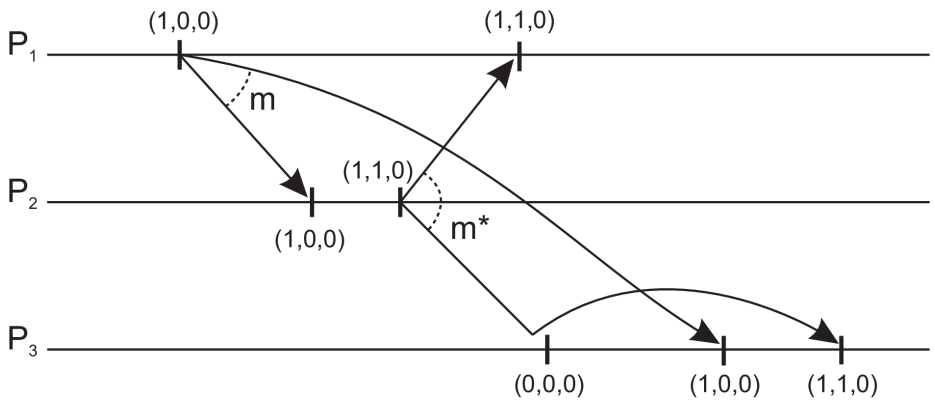
Relógio vectorial





Situação	ts(m2)	ts(m4)	ts(m2) < ts(m4)	ts(m2) > ts(m4)	Conclusão
a	(2,1,0)	(4,3,0)	Sim	Não	m2 aconteceu antes de m4
b	(4,1,0)	(2,3,0)	Não	Não	m2 e m4 podem estar em conflito

Relógio vectorial



- É necessário garantir que m só é entregue após todas mensagens casuísticas terem sido entregues.
- P_i incrementa $VC_i[i]$ apenas quando envia a mensagem, P_j ajusta VC_j quando recebe a mensagem

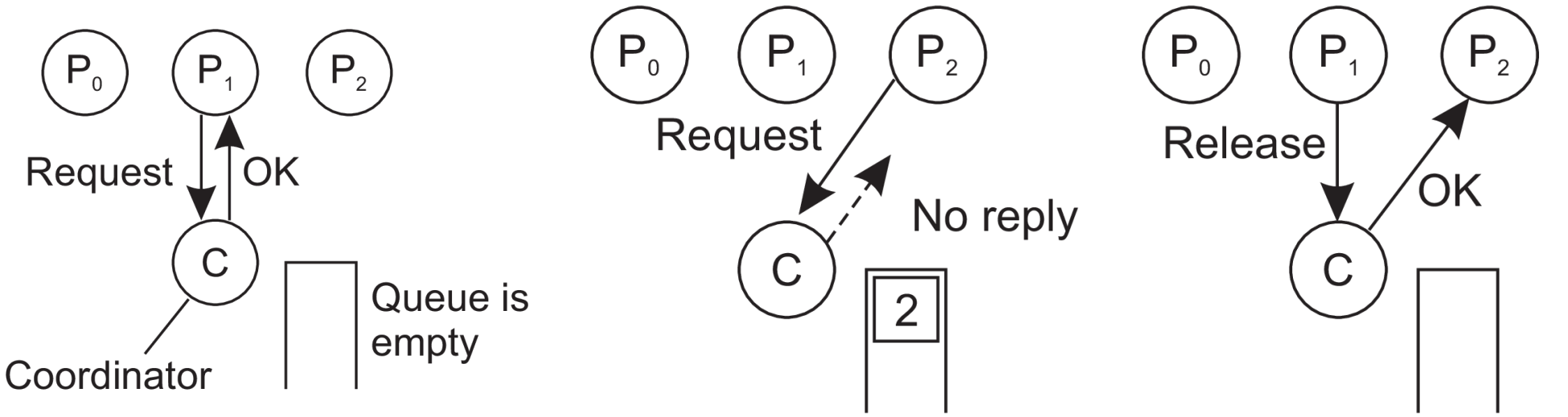
$ts(m)[i] = VC_j[i] + 1$

$ts(m)[k] <= VC_j[k]$ para todo k!=i

Algoritmos de exclusão mútua

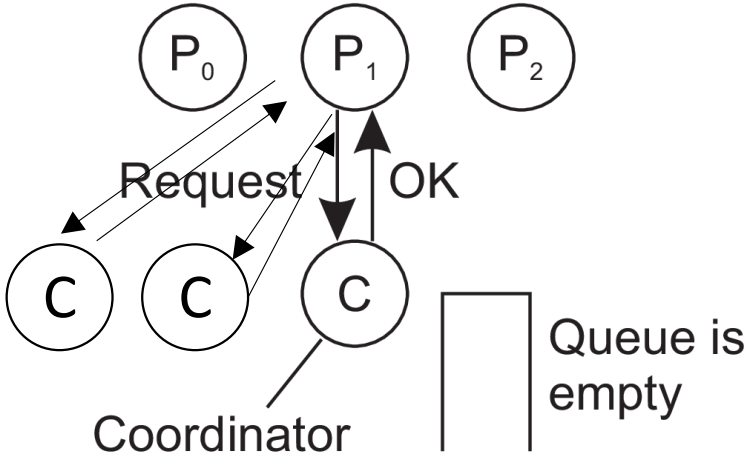
- Baseados em Permissões
- Baseados em Token

Exclusão Mútua - Centralizado



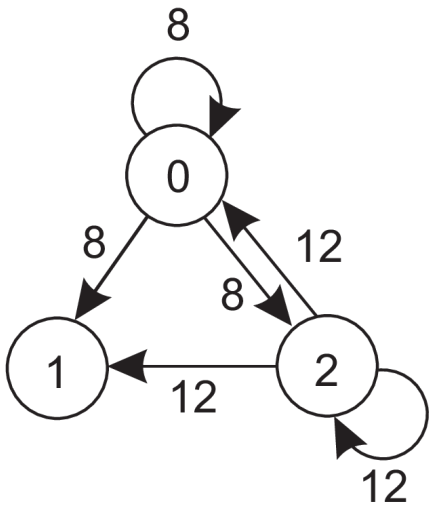
Exclusão Mútua - Descentralizado

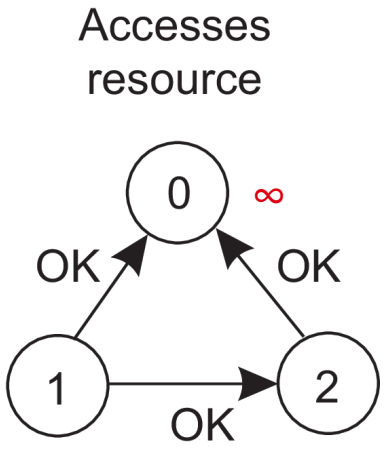
- Vários coordenadores,
- Acesso é atribuído por maioria de votos

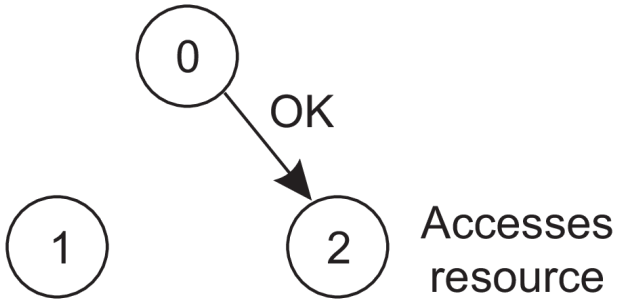


Exclusão Mútua - Distribuído

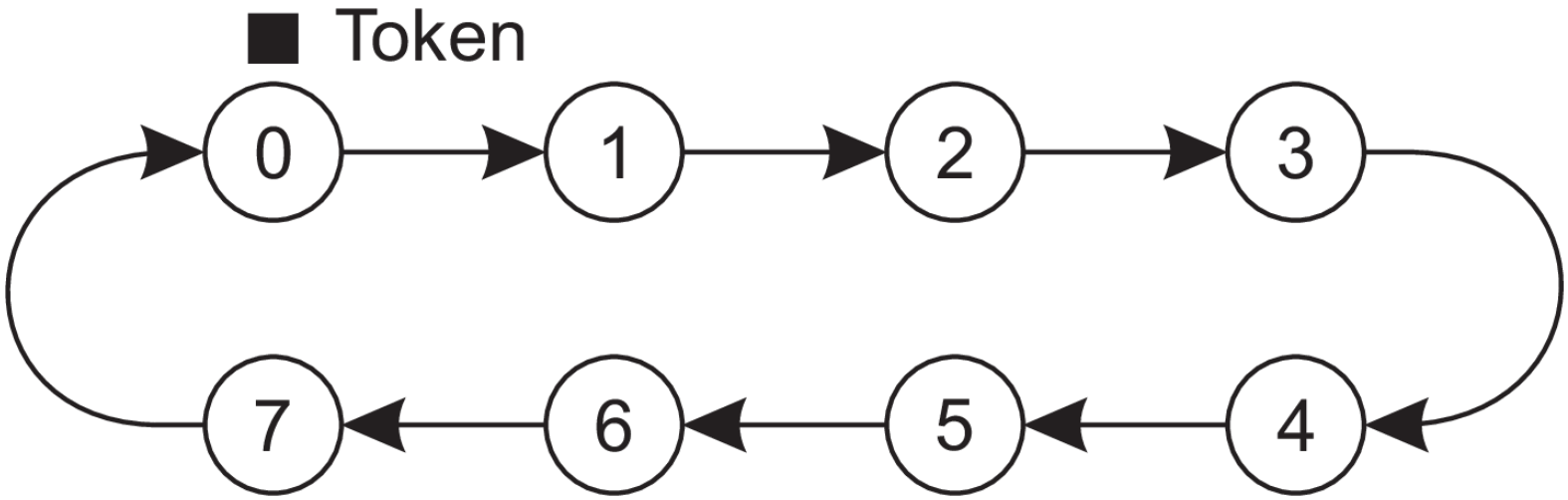
- Cada nó que pretende aceder a um recurso envia o seu relógio interno
- Demais nós enviam OK ou seu relógio.
- Menor relógio é prioritário e faz queue do próximo nó







Exclusão Mútua – Token Ring



Comparação

Algoritmo	Mensagens por entrada/saída	Atraso antes da entrada (em tempos de mensagem)	Problemas
Centralizado	3	2	Coordenador crasha
Descentralizado	2 m k + m, k = 1,2,....	2 m k	Fome, baixa eficiência
Distribuído	2 (n-1)	2 (n-1)	Crash de um qq processo
Token Ring	1 a ∞	0 a n - 1	Perda token

n: número de processos
m: número de coordenadores
k: número de tentativas

Algoritmos de Eleição

- 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?
- Um sistema completamente distribuído (sem qualquer coordenador), será sempre mais robusto do que um centralizado/coordenado ?

Pressuposto iniciais

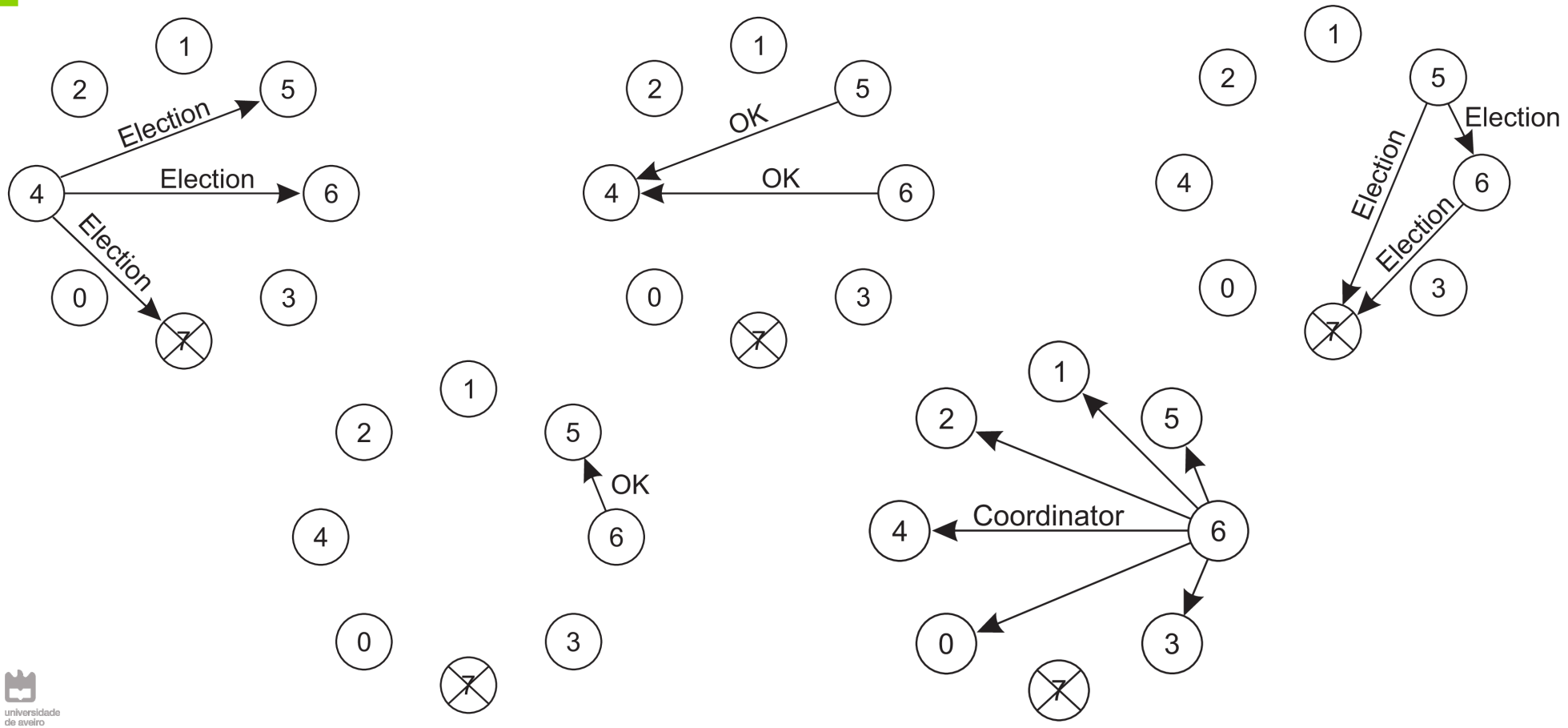
- Todos processos têm um identificador único (id)
- Todos processos conhecem os id's de todos processos no sistema (mas não o seu estado – ligado/desligado)
- Eleger significa identificar o processo com o id mais elevado que esteja ligado

Eleição por bullying

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

1. P_k envia uma mensagem ELECTION a todos processos com identificadores mais elevados: $P_{k+1}, P_{k+2}, ..., P_{N-1}$.
2. Se nenhum responder, P_k ganha a eleição e torna-se no coordenador
3. Se um dos processos com id superior responder, este assume a posição e o trabalho de P_k termina.

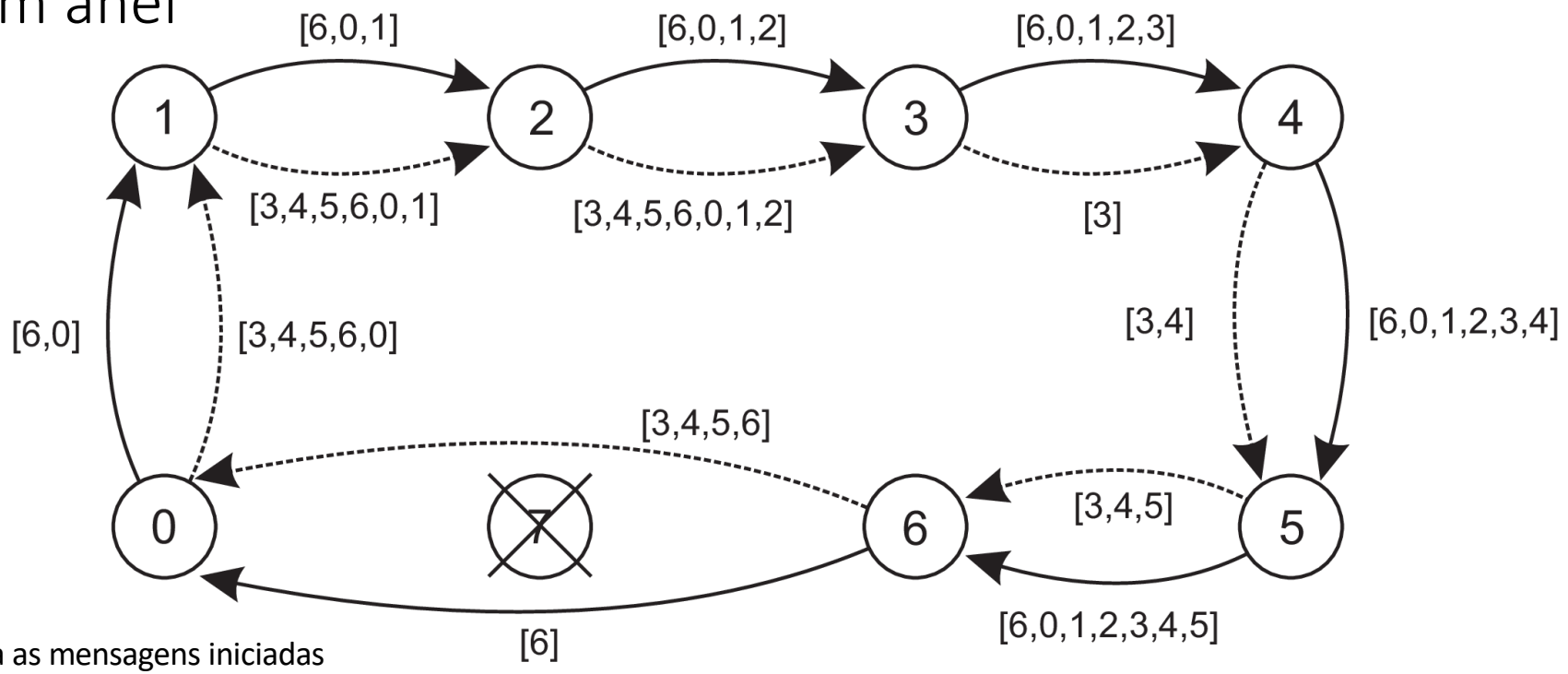
Eleição por bullying



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 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.

Eleição num anél

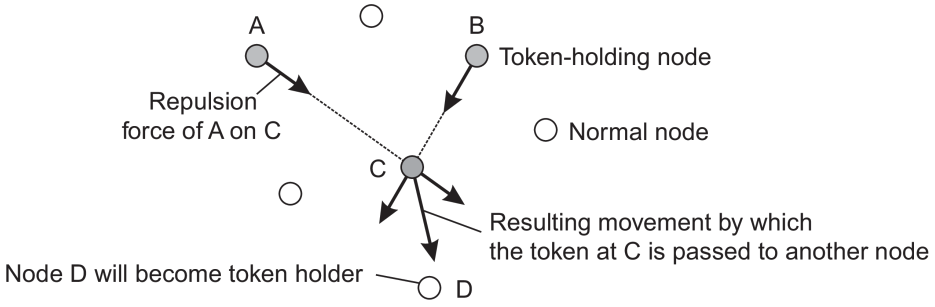


A linha sólida denota as mensagens iniciadas por P_6

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

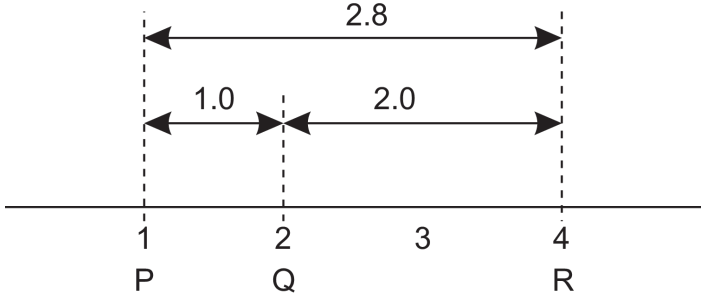


Posicionar nós

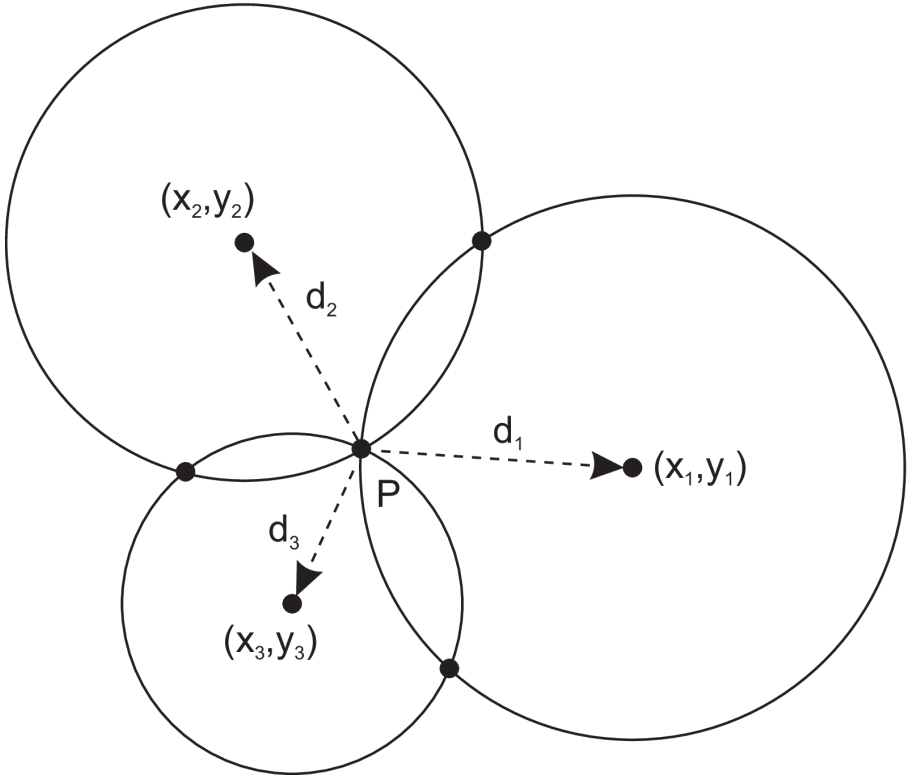
- Num sistema distribuído de larga escala, em que os nós se encontram dispersos numa WAN (leia-se Internet), é comum o sistema ter em atenção a noção de **proximidade** ou **distância**.
- Para tal é necessário determinar a localização do nó.

Calcular posição

- Um nó P necessita de d+1 marcos para poder calcular a sua própria posição num espaço d-dimensional.
- GPS ?
- Sinal de AP (Wifi/Bluetooth/etc)



Inconsistência entre medidas



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)