SCE616
Sistemas Computacionais Distribuídos

Sincronismo em Sistemas Distribuídos

Profa. Sarita Mazzini Bruschi
sarita@icmc.usp.br

Sincronismo em Sistemas Distribuídos

- As técnicas de sincronização são importantes pois frequentemente os sistemas distribuídos precisam realizar tarefas que necessitam de um comportamento sincronizado
- Em sistemas com uma única CPU, problemas tais como regiões críticas, exclusões mútuas e outros problemas de sincronização são geralmente resolvidos utilizando métodos tais como semáforos e monitores
- Em sistemas distribuídos esses métodos não podem ser utilizados pois os vários processadores não compartilham o mesmo espaço de endereçamento

Sincronização de *clock*

- Sincronismo em sistemas distribuídos é mais complicado do que nos sistemas centralizados pois há a necessidade de utilizar algoritmos distribuídos
- Não é possível, ou desejável, coletar todas as informações sobre o sistema em um único lugar e deixar que um único processo analise as informações e tome uma decisão

Sincronização de *clock*

- Em geral, nos algoritmos distribuídos:
  - As informações relevantes devem estar espalhadas nas várias máquinas;
  - Os processos tomam decisões baseadas somente em informações locais;
  - Deve ser evitado um único ponto de falha no sistema que paralise todo o sistema;
    - Caso contrário, torna o sistema não confiável
  - Não existe um “clock” comum ou um único tempo global
    - Obter um acordo de tempo não é um trivial
    - Exemplo: comando *make*

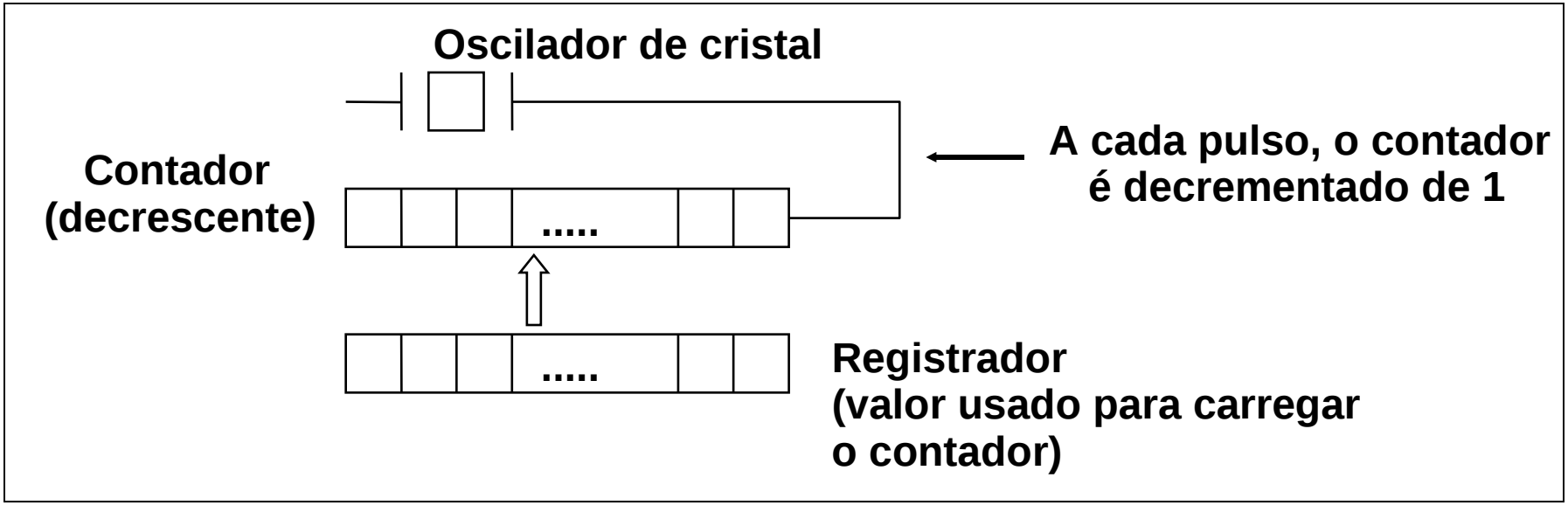
Sincronização de *clock*

- Para alguns algoritmos, o que importa é a consistência interna
  - Relógios conhecidos como **relógios lógicos**
- Outros algoritmos possuem a restrição de que os relógios devem não somente ser os mesmos, mas também o tempo real não pode divergir de mais de uma certa quantidade
  - Relógios conhecidos como **relógios físicos**

Sincronismo de *clock*

Clock físico

- Alguns sistemas, como por exemplo em sistemas de tempo real, é importante tem um sincronismo de relógio físico
- Para isso, é necessária a utilização de relógios físicos externos



Sincronismo de *clock*
Clock físico

- TAI (International Atomic Time):
  - Baseado nas transições do Cesium 133
  - Problema: 86.400 segundos TAI (24 horas no dia * 3600 segundos por hora) é atualmente 3 mseg menos do que um dia solar médio
  - Conforme o tempo vai passando, o dia vai ficando mais longo:
    - 300 milhões de anos atrás, havia 400 dias no ano (tempo para uma volta ao redor do sol)

Sincronismo de *clock*
Clock físico

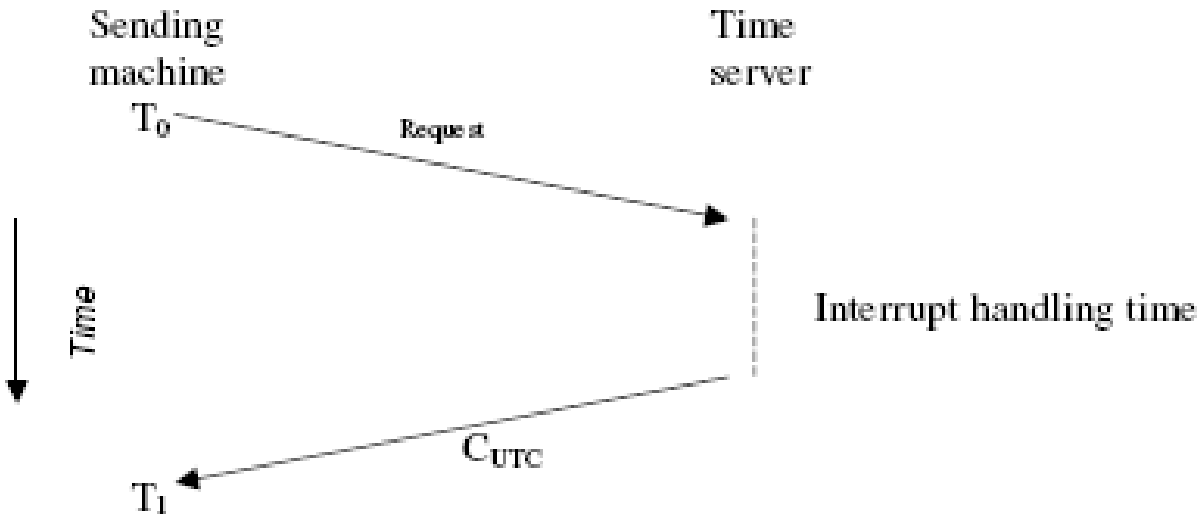
- UTC (*Universal Coordinated Time*):
  - Em fase com o movimento do sol
  - Estações de rádio operando em ondas curtas com prefixo WWV enviam regularmente em *broadcast* um pulso no início de cada segundo UTC
  - A precisão fornecida pela WWV é de mais ou menos 1 mseg mas na prática, condições atmosféricas fazem a precisão ser de mais ou menos 10 mseg

Sincronismo de *clock*
Algoritmos de sincronização de relógio

- Algoritmo de Cristian
  - Trabalha utilizando máquinas que possuem uma máquina receptora WWV, a qual será denominada *servidor de tempo*
  - Periodicamente, cada máquina envia uma mensagem para o servidor de tempo perguntando pelo tempo corrente (atual). Esta máquina responde o mais rápido possível com uma mensagem contendo o tempo corrente C_{UTC}

Sincronismo de *clock*
Algoritmos de sincronização de relógio

- Algoritmo de Cristian
  - T_0 e T_1 são medidos usando o mesmo clock



Sincronismo de *clock*
Algoritmos de sincronização de relógio

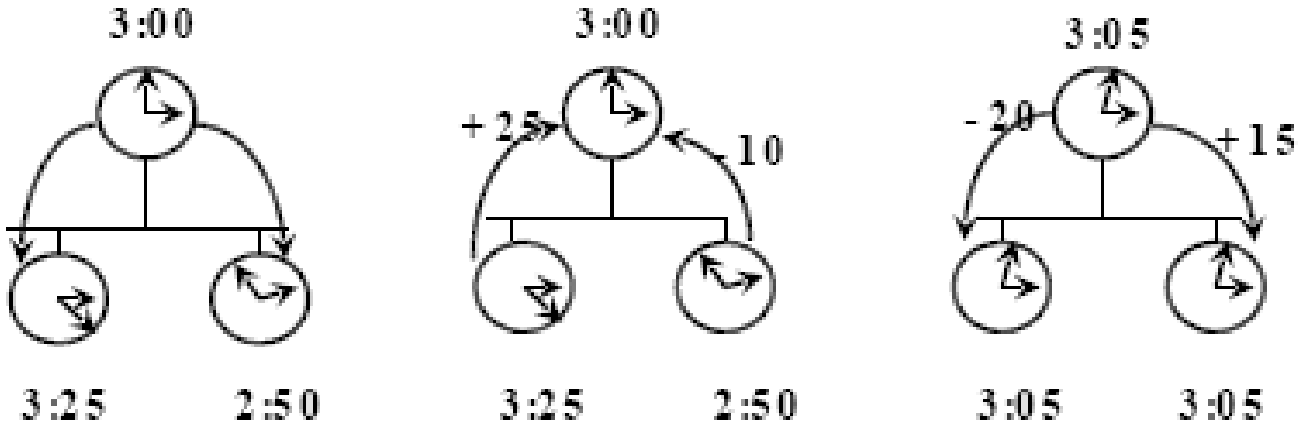
- Tempo de propagação da mensagem:
  - $TP = (T_1 – T_0)/2$
  - $Clock = C_{UTC} + TP$
- A estimativa pode ser melhorada se for considerado o tempo de manipulação da interrupção
  - $TP = ((T_1 – T_0 – I)/2$
- Problema:
  - O tempo UTC nunca deve estar menor do que a hora local da máquina (o relógio nunca pode ser decrementado)

Sincronismo de *clock*
Algoritmos de sincronização de relógio

- Algoritmo de Berkeley
  - Não existe nenhuma máquina com WWV (receptor UTC)
  - No caso do Algoritmo de Cristian, o Servidor de Tempo é passivo (outras máquinas requisitam o tempo periodicamente)
  - No Algoritmo de Berkeley, o Servidor de Tempo é ativo e requer, periodicamente de cada máquina, o tempo do seu relógio
  - O Servidor de Tempo calcula uma média dos tempos e diz para cada máquina como ajustar o seu relógio

Sincronismo de *clock*
Algoritmos de sincronização de relógio

- Algoritmo de Berkeley



Sincronismo de *clock*
Clock lógico

- Os relógios dos computadores são dispositivos físicos que geram interrupções com freqüência contínua
- Lamport (1978), em seu artigo "*Time, Clocks and the ordering of events in a Distributed System*" mostra que a sincronização dos relógios é possível e apresenta um algoritmo

Sincronismo de *clock*
Clock lógico

- A sincronização dos clocks não precisa ser absoluta
  - Se dois processos não interagem, não é necessário que seus clocks sejam sincronizados pois a falta de sincronização não será observada e não causará problemas
  - O importante não é que todos os processos concordem com o exato tempo em que os eventos acontecem, mas que concordem na **ordem** em que os eventos ocorrem

Sincronismo de *clock*
Clock lógico

- Lamport definiu a relação *happens-before*, ou "acontece-antes"
- A expressão $a \rightarrow b$ quer dizer "a acontece antes de *b*" e significa que todos os processos concordam que primeiro o evento *a* ocorreu e depois disto, o evento *b* ocorreu

Sincronismo de *clock*
Clock lógico

- Algoritmo de Lamport:
  - Se *a* e *b* são eventos no mesmo processo, e *a* ocorre antes de *b*, então $a \rightarrow b$ é verdadeira
  - Se *a* é um evento de uma mensagem sendo enviada por um processo, e *b* é o evento da mensagem sendo recebida por outro processo, então $a \rightarrow b$ é também verdadeira. Uma mensagem não pode ser recebida antes de ser enviada ou mesmo tempo em que foi enviada

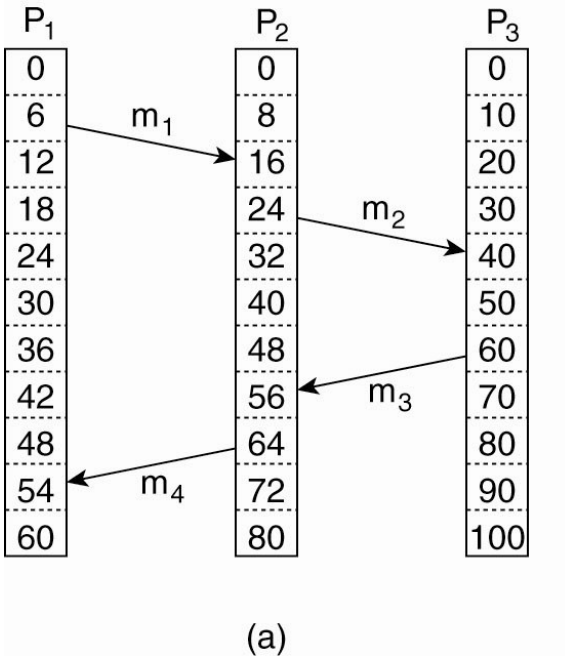
Sincronismo de *clock*
Clock lógico

- A relação “acontece-antes” é uma relação transitiva:
  - Se $a \rightarrow b$ e $b \rightarrow c$, então $a \rightarrow c$
- Se dois eventos *x* e *y* acontecem em diferentes processos que não trocam mensagens, então $x \rightarrow y$ não é verdadeiro nem $y \rightarrow x$ é verdadeiro. Estes eventos são ditos concorrentes, o que significa que nada pode ser dito sobre quando eles aconteceram
- Precisa-se de um modo de medir o tempo tal que para cada evento *a* possa-se associar o valor *C(a)* de modo que todos os processos concordem

Sincronismo de *clock*
Clock lógico

- Considere que os processos abaixo estão sendo executados em máquinas diferentes, cada uma com um clock, executando a sua velocidade

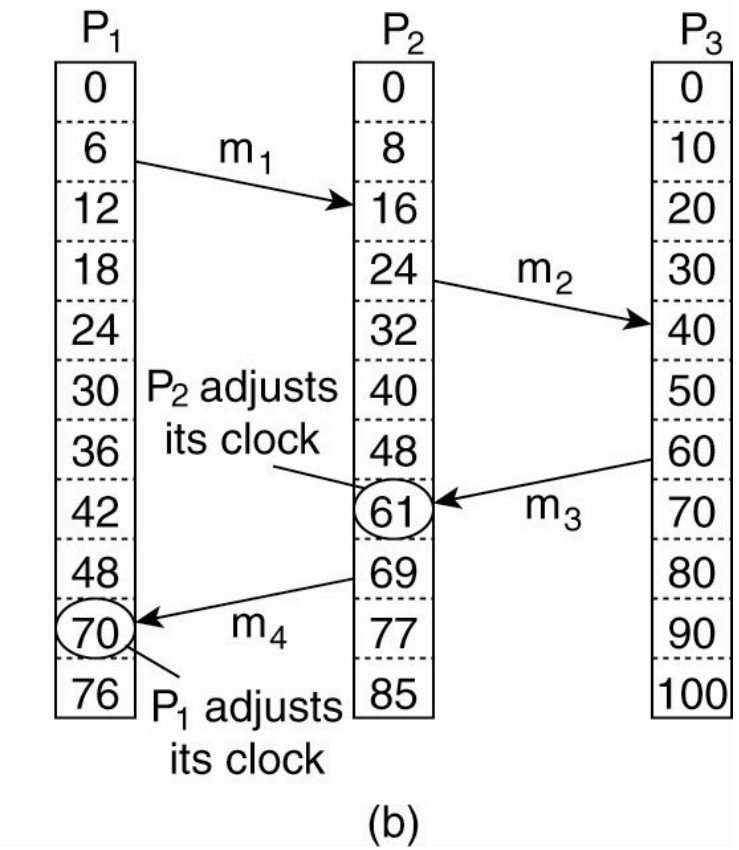
	1	2	3
Clock tick	6	8	10



(a)

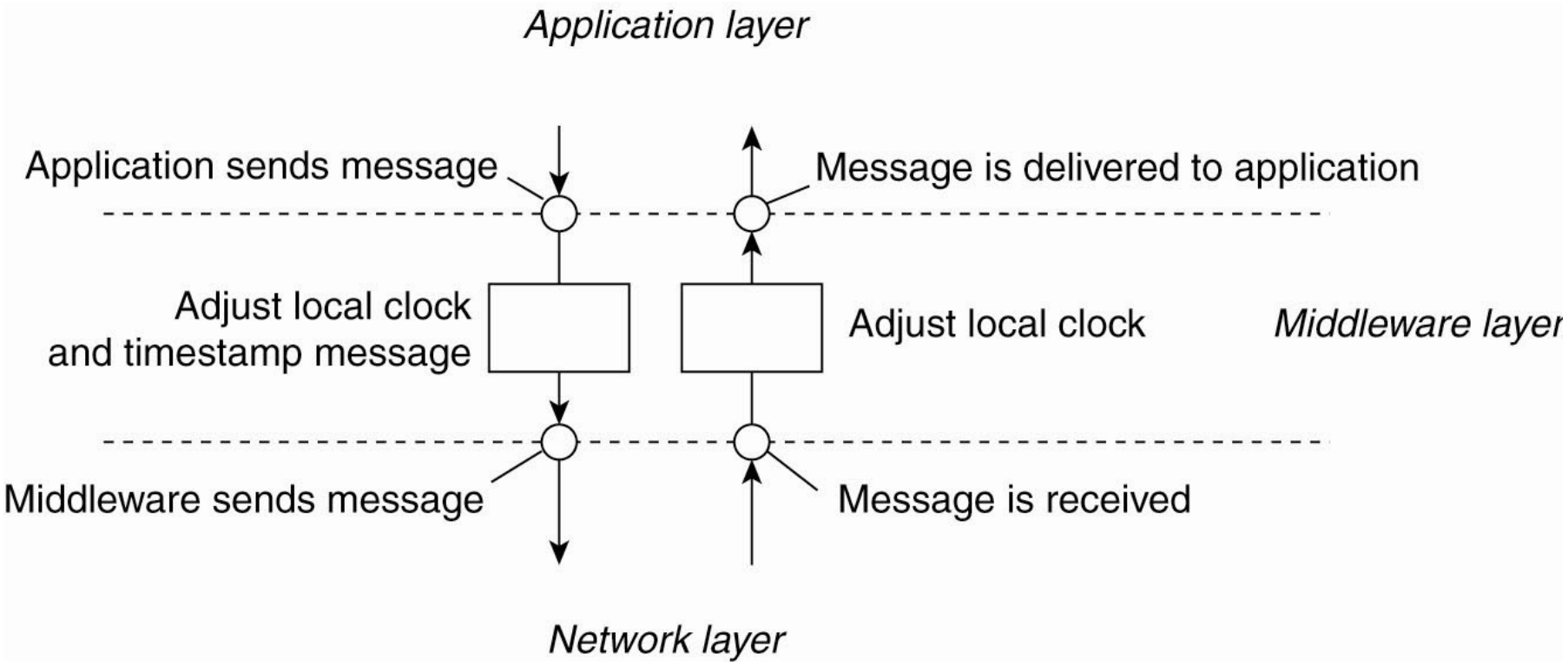
Sincronismo de *clock*
Clock lógico

- Solução de Lamport:
  - Cada mensagem carrega o tempo de envio, de acordo com o clock da máquina que está enviando. Quando uma mensagem chega e o relógio do receptor possui um valor anterior ao da mensagem, este avança seu relógio para o tempo de envio da mensagem mais um
  - O clock deve sempre avançar, nunca retroceder



Sincronismo de *clock*
Clock lógico

- Localização do relógio lógico de Lamport



Sincronismo de *clock*
Clock lógico

- Implementação dos relógios lógicos, sendo que cada processo P_i possui um contador C_i
  - Antes de executar um evento P_i , faz $C_i \leftarrow C_i + 1$.
  - Quando um processo P_i envia uma mensagem *m* para P_j, ele ajusta o timestamp de m, *ts (m),* para igual ao de C_i após ter executado a etapa anteior.
  - Ao receber uma mensagem *m*, o processo P_j atualiza seu próprio contador local como $C_j \leftarrow \max\{C_j , ts\ (m)\}$, e, depois disso, executa a primeira etapa e entrega a mensagem para a aplicação.

Sincronismo de *clock*
Clock lógico

- Relógios vetoriais
  - Com os relógios de Lamport, nada se pode dizer sobre dois eventos *a* e *b*, tendo somente os valores *C(a)* e *C(b)*.
    - *C(a)* < *C(b)* não implica que $a \rightarrow b$
  - Exemplo
    - Mensagens enviadas pelos processos da figura do próximo slide
    - $T_{snd}(m_i)$: instante lógico em que a mensagem mi foi enviada
    - $T_{rcv}(m_i)$: instante lógico em que ela foi recebida

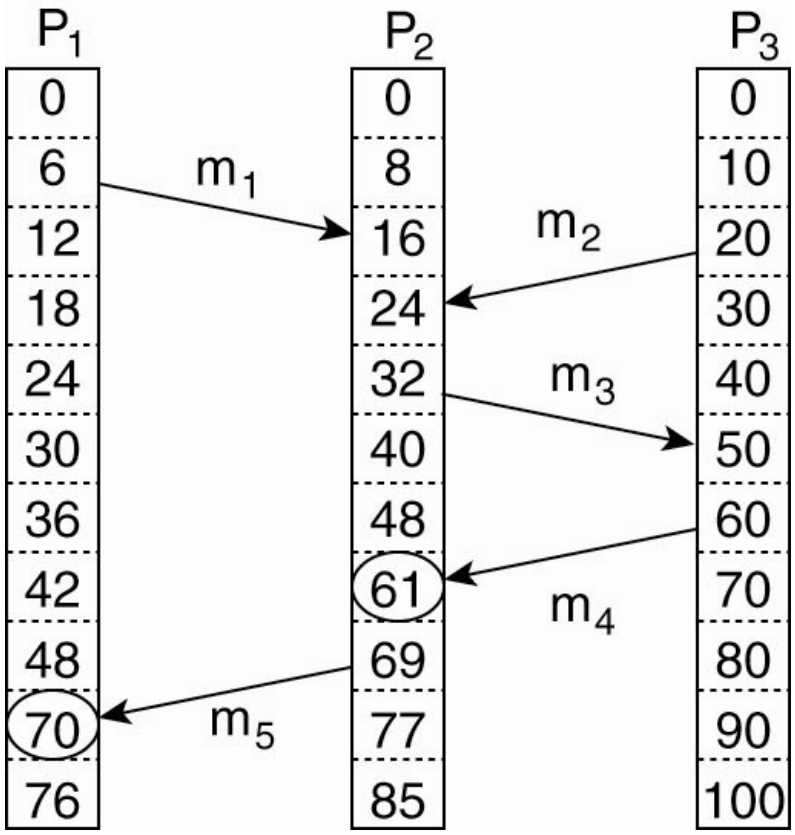
Sincronismo de *clock*

Clock lógico – Relógios vetoriais

- Eventos que ocorreram em P_2:

$T_{rcv}(m_1) < T_{snd}(m_3)$, pois provavelmente m_3 dependeu do recebimento de m1

- $T_{rcv}(m_1) < T_{snd}(m_2)$, no entanto, o envio de m_2 nada tem a ver com o recebimento de m_1.



- Problema: o relógio lógico de Lamport não considera, a causalidade

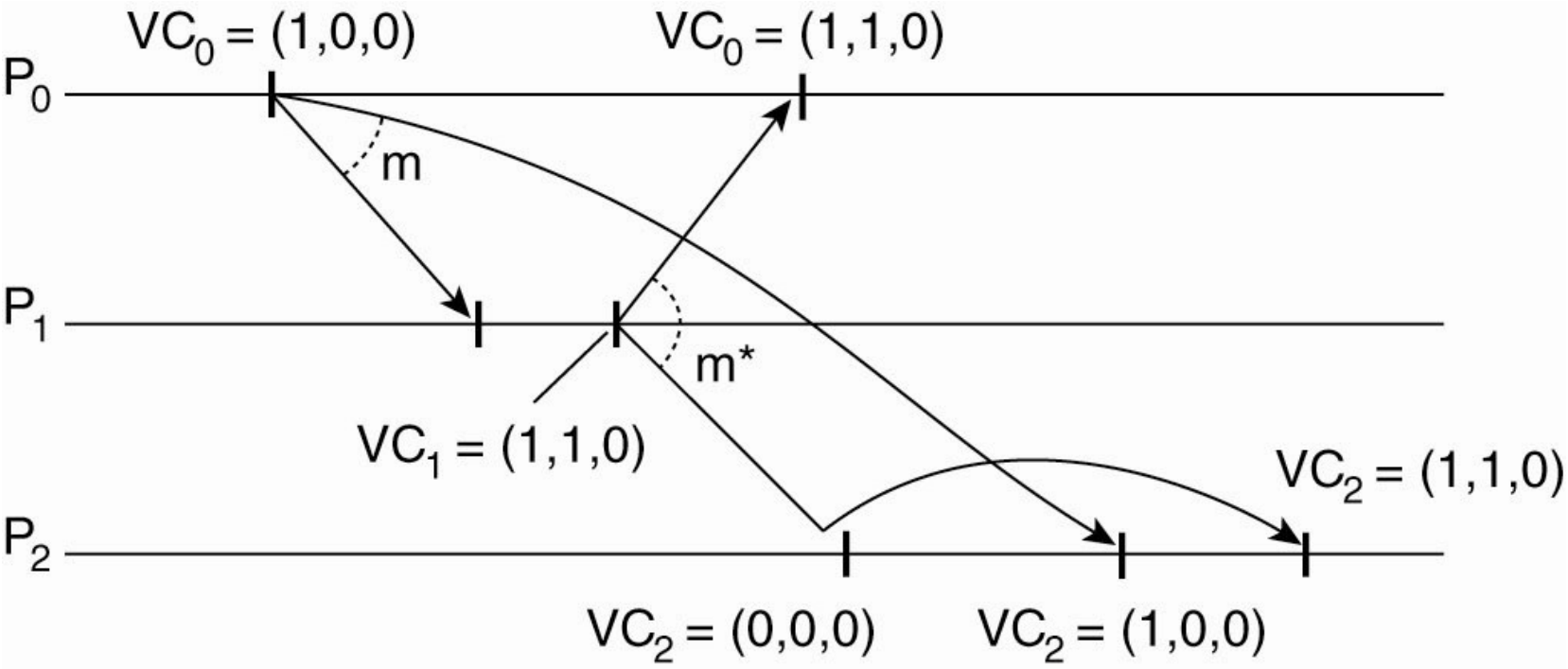
Sincronismo de *clock*
Clock lógico – Relógios vetoriais

- Relógios vetoriais são definidos de modo a permitir que cada processo mantenha um vetor VC_i com as seguintes propriedades:
  - $VC_j[i]$ é o número de eventos que ocorreram dm P_i até o instante em questão. $VC_i[i]$ é o relógio lógico local no processo P_i.
  - Se $VC_i[j] = k$, então P_i sabe que k eventos ocorreram em P_j, portanto, P_i conhece o tempo local em P_j.

Sincronismo de *clock*
Clock lógico – Relógios vetoriais

- Passos para se conseguir a propriedade 2 do slide anterior:
  - Antes de executar um evento P_i fazer $VC_i[i] \leftarrow VC_i[i] + 1$.
  - Quando um processo P_i envia uma mensagem *m* para P_j, ele iguala o *timestamp* (vetorial) *ts (m)* ao *timestamp* de VC_i , após ter executado a etapa anteior.
  - Ao receber uma mensagem *m*, o processo P_j ajusta seu próprio vetor fazendo $VC_j[k] \leftarrow \max\{VC_j[k], ts(m)[k]\}$ para cada *k;* em seguinda, executa a primeira etapa e entrega a mensagem à aplicação.

Imposição de comunicação causal



Exclusão Mútua

- Sistemas envolvendo múltiplos processos são programados mais facilmente utilizando regiões críticas
- Quando um processo deve ler ou atualizar estruturas de dados compartilhadas, ele tenta primeiro garantir a exclusão mútua para acesso à região crítica
- Em um sistema centralizado, isso é facilmente conseguido através de semáforos ou monitores. Em sistemas distribuídos impor a exclusão mútua exige um cuidado maior

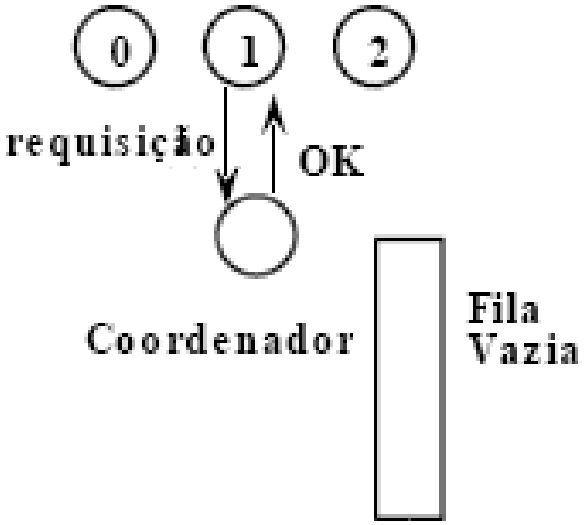
Exclusão Mútua
Algoritmo Centralizado

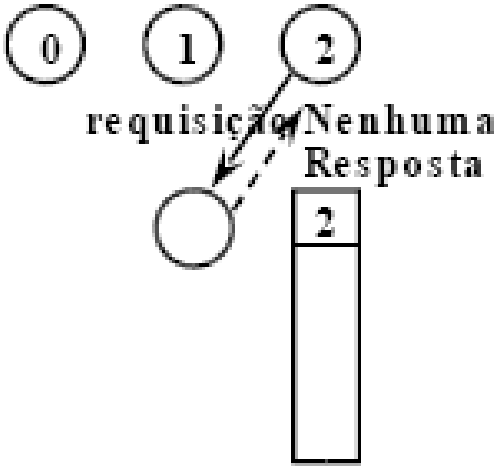
- Algoritmo que imita o sistema centralizado:
  - Um processo é eleito como coordenador
  - Quando um processo precisa entrar numa região crítica, este envia uma mensagem *request* ao coordenador informando qual a região crítica e requisitando a permissão
  - Se nenhum outro processo está na região crítica, o coordenador envia uma mensagem *reply* garantindo permissão
  - Quando uma mensagem resposta chega, o processo entra na região crítica

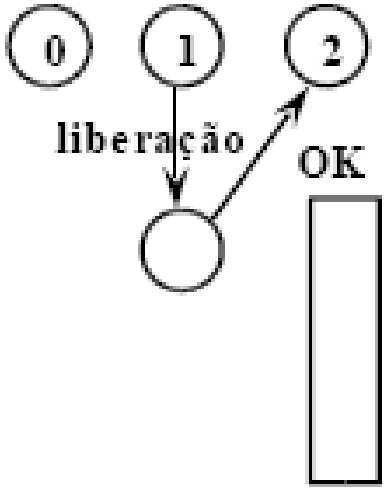
Exclusão Mútua
Algoritmo Centralizado

- Se existir outro processo na região crítica, o processo coordenador pode tomar duas atitudes:
  - Envia uma mensagem *reply* com o conteúdo *permission denied*
  - Não envia resposta enquanto a região crítica não estiver disponível
- Quando o processo deixa a região, ele envia para o coordenador uma mensagem liberando a região
- Este algoritmo impõe a exclusão mútua, no entanto algoritmos centralizados podem se tornar pontos de falhas e de gargalo da rede

Exclusão Mútua
Algoritmo Centralizado







Exclusão Mútua
Algoritmo Distribuído

- O algoritmo centralizado tem o problema de uma falha no coordenador inviabilizar o mecanismo
- No algoritmo distribuído, quando um processo entra na região crítica, ele constrói uma mensagem contendo o nome da região crítica, o número do processo e o tempo corrente e envia essa mensagem para todos os processos, inclusive ele próprio

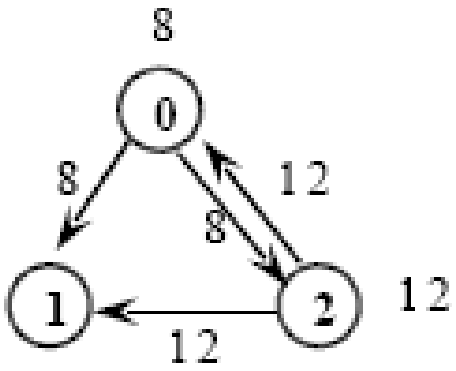
Exclusão Mútua
Algoritmo Distribuído

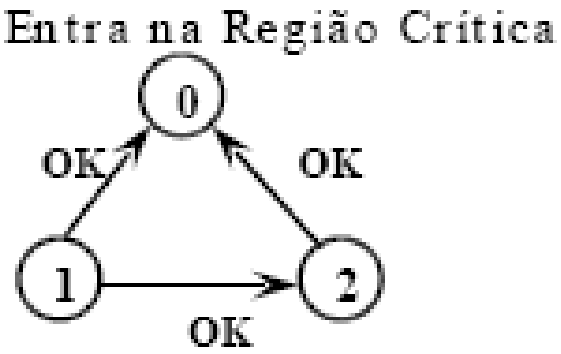
- Quando um processo recebe uma mensagem de requisição de outro processo:
  - Se o receptor não está na região crítica e não quer entrar, ele envia de volta uma mensagem de OK
  - Se o receptor já está na região crítica, ele não responde e coloca a requisição na fila
  - Se o receptor quer entrar na região crítica mas ainda não conseguiu, ele compara o tempo da mensagem (*timestamp*) que chegou com o *timestamp* da mensagem que ele enviou. A que tiver *timestamp* menor vence
    - Se a mensagem que chegou é menor, o receptor envia uma mensagem OK para o processo emissor
    - Se a sua mensagem é a que possui o *timestamp* menor, o receptor coloca na fila a mensagem *request* e não responde ao emissor

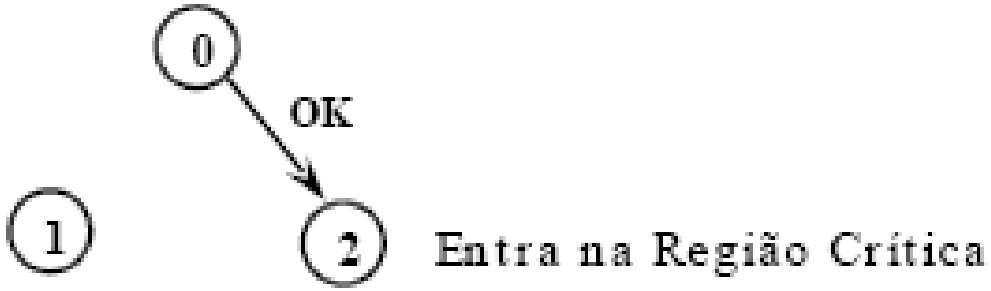
Exclusão Mútua
Algoritmo Distribuído

- Após pedir permissão um processo espera até que todos tenham dado a sua permissão. Quando todas as permissões chegarem o processo pode entrar na região crítica. Quando ele sai da região crítica, envia uma mensagem de OK para todos os processos na sua fila
- Problemas
  - Transforma um único ponto de falha do sistema centralizado em *n* pontos de falha no sistema distribuído
    - Solução: sempre responder a uma requisição e não deixar que a resposta por não ter permissão para entrar na região crítica seja não enviar nenhuma resposta
  - Precisa de mecanismos para comunicação em grupo

Exclusão Mútua
Algoritmo Distribuído







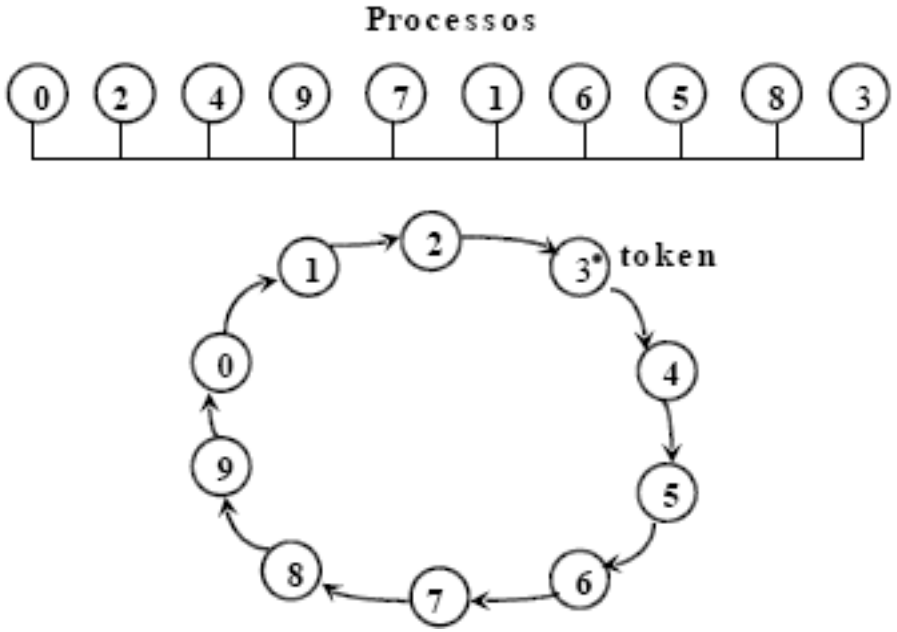
Exclusão Mútua
Algoritmo Token Ring

- Neste algoritmo, é construído um anel lógico por software no qual a cada processo é atribuído uma posição no anel
- Quando o anel é inicializado, o processo 0 ganha o *token*, o qual circula no anel (passa do processo *k* para o processo *k* + 1)
- Quando o processo ganha o *token* ele verifica se ele quer entrar na região, realiza seu trabalho e, ao deixar a região, passa o *token* para o elemento seguinte do anel.
- Não é permitido entrar numa segunda região crítica com o mesmo *token*

Exclusão Mútua
Algoritmo Token Ring

- Se o processo não quer entrar na região crítica, ele simplesmente passa o *token*
- Como consequência, quando nenhum processo quer entrar na região crítica, o *token* fica circulando no anel
- Problemas:
  - Se o *token* é perdido ele precisa ser regenerado. A detecção de um *token* perdido é difícil
  - Se um processo falha também ocorrem problemas e a solução é fazer com que o processo que recebe o *token* confirme o recebimento. O processo que falhou pode ser retirado do anel e o *token* enviado para o processo seguinte. Este procedimento requer que todos os processo saibam da configuração do anel

Exclusão Mútua
Algoritmo Token Ring



Exclusão Mútua
Comparação dos algoritmos

Algoritmo	Mensagens	Atraso	Problemas
Centralizado	3	2	Falha do Coordenador
Distribuído	2(n-1)	2(n-1)	Falha de qualquer processo
Token Ring	1 a infinito	0 a n-1	Perda do Token

Algoritmos para Eleição

- Muitos processos distribuídos requerem um processo para agir como coordenador, inicializador ou alguma outra atividade em especial
  - Exemplo: algoritmo centralizado de exclusão mútua
- Nestes casos é necessário um algoritmo para escolher qual processo deve assumir a função de coordenador

Algoritmos para Eleição

- Se todos os processos são exatamente do mesmo tipo, sem nenhuma característica que os distingam dos demais, não existe nenhuma forma de selecionar um deles em especial.
- Assume-se que cada processo possui um endereço único (por exemplo seu endereço de rede), e os algoritmos de eleição tentam localizar o processo com o maior número e designá-lo como coordenador
- O objetivo do algoritmo de eleição é garantir que quando uma eleição termina, todos os processos concordam com a decisão de quem é o novo cordenador

Algoritmos para Eleição
Algoritmo Bully

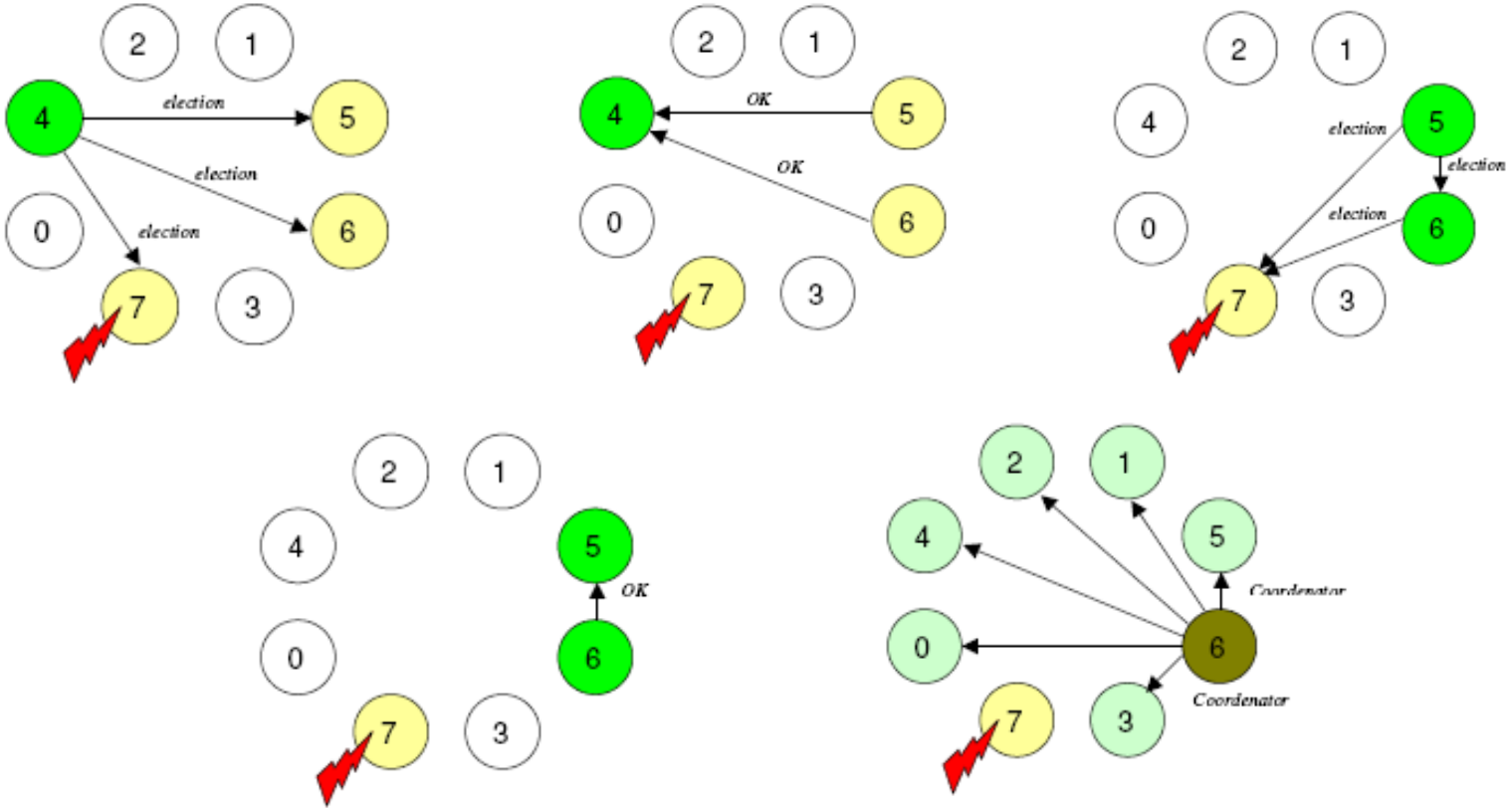
- Quando um processo (P) nota que o coordenador não está mais respondendo a uma requisição, ele inicia uma eleição:
  - P envia uma mensagem de *ELECTION* para todos os processos com números maiores que o seu;
  - Se nenhum responde, P ganha a eleição e se torna coordenador
  - Se um processo com um número maior responde, ele assume o processo de eleição

Algoritmos para Eleição
Algoritmo Bully

- Esse procedimento termina com um processo vencedor, o qual avisa a todos os outros que ele é o novo coordenador
- Se o processo que era coordenador e havia falhado retornar, ele inicia uma nova eleição. Se acontecer de seu número ser maior, ele irá vencer a eleição e retornará a ser o coordenador

Algoritmos para Eleição
Algoritmo Bully

- Exemplo



Algoritmos para Eleição
Algoritmo em Anel

- Algoritmo baseado na utilização de anel, porém sem *token*
- É assumido que os processos estão fisicamente ou logicamente ordenados, isto é, cada processo sabe quem é o seu sucessor e a configuração do anel
- Quando um processo percebe que o coordenador não está funcionando, ele constrói uma mensagem *ELECTION* contendo seu número e envia essa mensagem para o seu sucessor
- Se o sucessor estiver desativado, o emissor salta para o próximo sucessor

Algoritmos para Eleição
Algoritmo em Anel

- A cada passo o emissor adiciona o número dos processos na lista de mensagem
- Quando a mensagem dá a volta no anel e retorna ao processo que iniciou a eleição, o novo coordenador é determinado (o processo com mais alto número na lista) e esta mensagem é retirada do anel
- Uma nova mensagem denominada COORDINATOR é gerada informando quem é o novo coordenador

Algoritmos para Eleição
Algoritmo em Anel