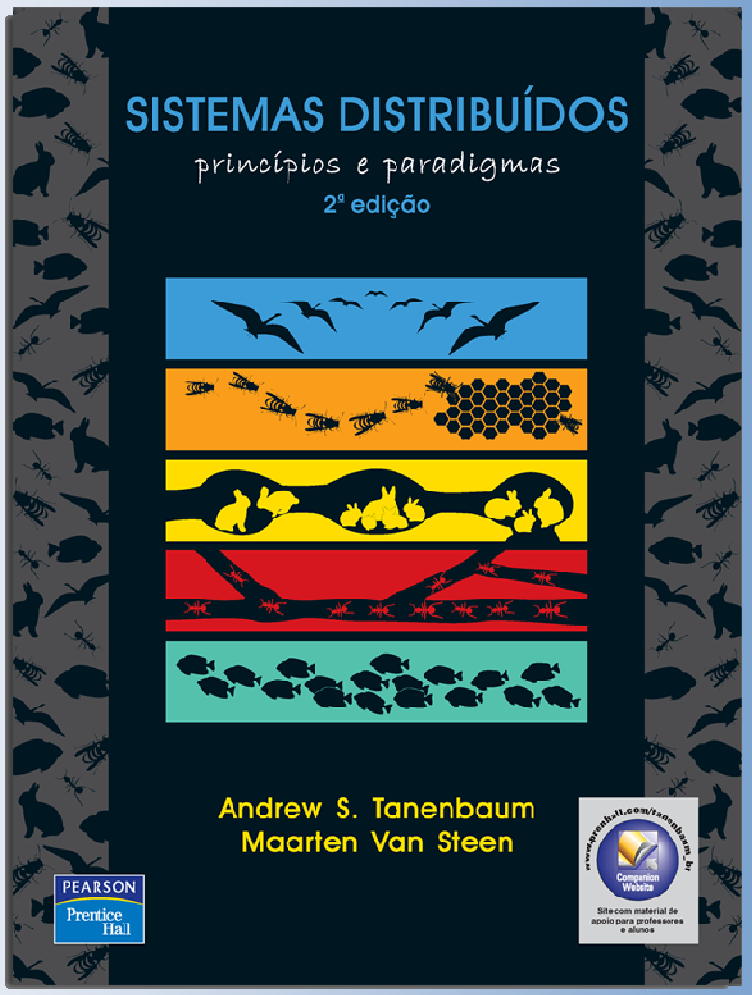
Sincronização



capítulo 6

Sincronização

1. Sincronização de Relógios
2. Relógios Lógicos
3. Exclusão Mútua
4. Posicionamento global de nós
5. Algoritmos de eleição

Sincronização de Relógios

- Relógios são essenciais no uso de computação, seja para medir o tempo ou identificar seqüências de eventos diversos.
  - Em sistemas centralizados, o tempo não é ambíguo, sendo gerenciado em apenas 1 máquina e obtido por chamada ao núcleo;
  - Em sistemas distribuídos, obter um horário comum a vários computadores **não é trivial**.

Sincronização de Relógios
Relógios físicos – Medida do tempo

- Até invenção dos relógios mecânicos (sec. XVII):
  - Tempo medido com auxílio dos astros;
    - Sol nasce no horizonte a leste;
    - Alcança uma altura máxima no céu, ao meio dia (**trânsito solar**);
    - Sol se põe no horizonte a oeste;
  - Intervalo entre 2 trânsitos solares consecutivos do Sol é um **dia solar**;
  - Portanto, se um dia possui 24 horas:
    - Cada hora possui 3600 segundos;
    - Um **segundo solar** é exatamente 1/86.400 de um dia solar.

Sincronização de Relógios
Relógios Físicos – Medida de tempo

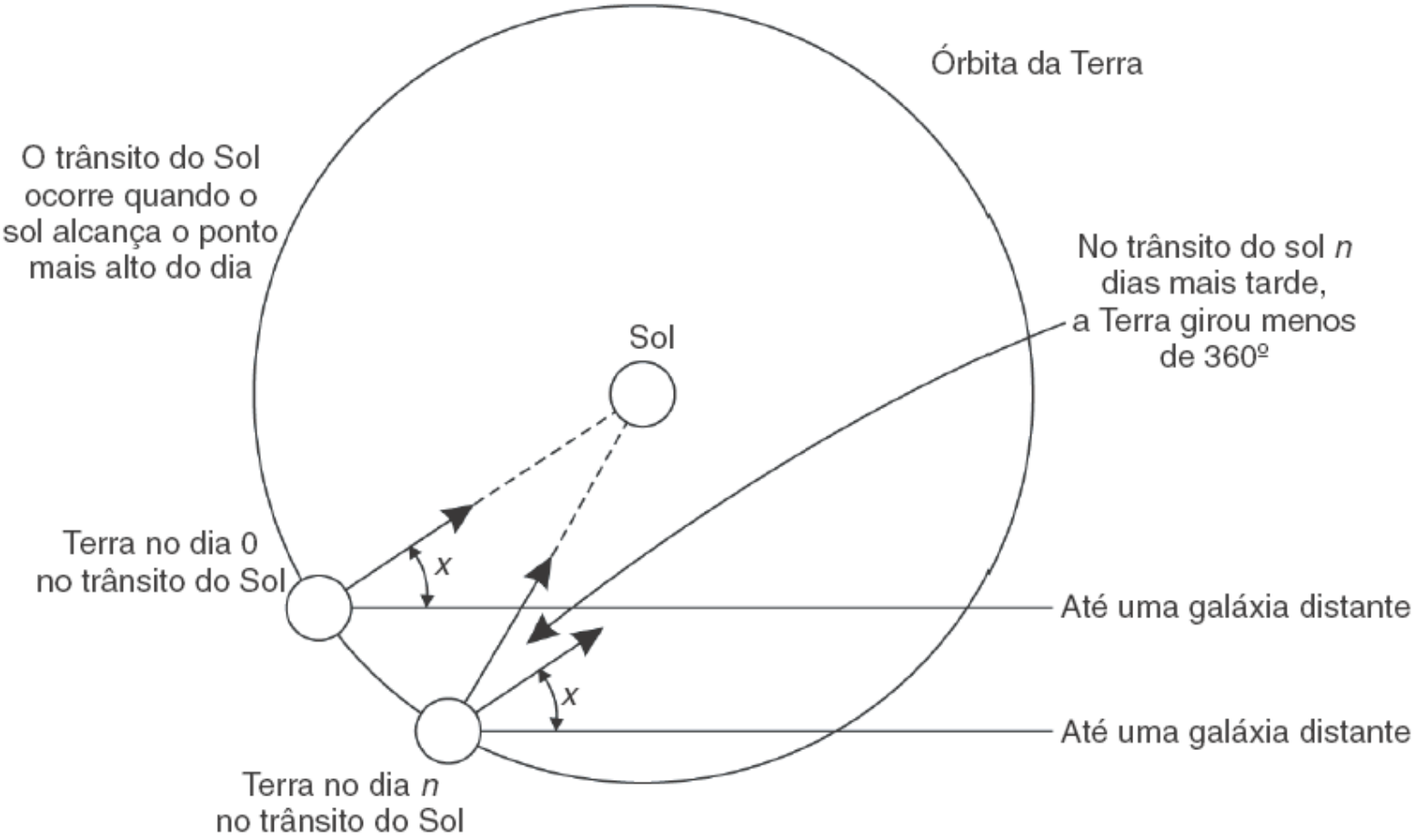


Figura 6.2 Cálculo do dia solar médio.

Sincronização de Relógios
Relógios Físicos – Medida de tempo

- Em meados de 1940:
  - Estabelecido que a rotação da Terra não é constante:
    - Devido à desaceleração gradativa resultante de marés e atrito com atmosfera;
    - Há 300 milhões de anos o ano tinha 400 dias!
      - A Terra gira mais devagar, mas não alterou sua órbita. Logo, o tamanho do ano aparenta ser o mesmo, mas os dias ficaram mais longos!
    - Além disso há pequenas alterações ao longo do dia, devido a turbulências no núcleo da Terra;
  - Necessário definir nova medida para o tempo:
    - Calculado o tempo de vários dias, obtendo uma média do dia.
    - Ao dividi-la por 86.400, obtém-se o **segundo solar médio**.

Sincronização de Relógios
Relógios Físicos – Medida de tempo

- Em 1948 foi inventado o relógio atômico:
  - Calcula o tempo através de contagens de transições do césio 133 (1 segundo solar médio = 9.192.631.770 transições).
- Tal fato tornou possível:
  - Medir o tempo com maior precisão;
  - Medi-lo independentemente das condições do globo terrestre e da atmosfera.
  - Cálculo da **hora atômica internacional** (**TAI**) pelo BIH.

Sincronização de Relógios
Relógios físicos – Medida do tempo

- Problema:
  - 86.400 segundos TAI equivalem a aproximadamente 3ms a menos que um dia solar médio.
- Solução:
  - Segundos extras a cada 800ms acumulados

Figura 6.3 Segundos TAI têm comprimento constante, diferentes dos segundos solares. Os segundos extras são introduzidos quando necessário para se manterem em fase com o sol.

Sincronização de Relógios
Relógios Físicos e GPS

- Com base no TAI com correções de segundos foi estabelecido o sistema **UTC** (Universal Coordinated Time), que é a base de toda a moderna medição de tempo.
  - O Nist fornece UTC através de rádios de ondas curtas (WWV).
  - Além disso vários satélites fornecem UTC.
  - O GPS faz triangulação usando satélites de modo a calcular as diferenças de tempo entre o UTC de cada satélite usado, calculando, assim, a localização geográfica do ponto.

Relógios Físicos
Algoritmos de sincronização de relógios

- Se uma das máquinas possui receptor WWV:
  - manter todas as outras máquinas sincronizadas com ela.
- Se nenhuma possui receptor WWV:
  - cada uma cuida de seu próprio horário;
  - o problema passa a ser manter o horário de todas máquinas o mais próximo possível.
- Foram propostos vários algoritmos, todos seguindo as mesmas idéias básicas.

Relógios Físicos
Algoritmos de sincronização de relógios

- Algoritmo:
  - Todas máquinas possuem um temporizador que gera interrupções na taxa H vezes por segundo;
  - Quando o temporizador expira, adicionar 1 unidade ao clock C conhecido por todas máquinas do sistema;
  - Quando o tempo UTC for *t* o valor de clock de uma máquina *p* é $C_p(t)$.
  - Em condições ideais:
    - $C_p(t) = t$, para qualquer valor de *p* e *t*
    - $C'_p(t) = dC/dt = 1$
    - Condições impossíveis de se obter na prática!
      - O erro relativo aproximado dos temporizadores é 10^{-5}

Relógios Físicos
Protocolo de tempo de Rede - NTP

- Proposto por Cristian(1989), baseia-se em clientes consultarem um servidor de tempo.
- Funcionamento:
  1. Cada máquina envia uma mensagem para o servidor de tempo (máquina com receptor WWV ou relógio de precisão), perguntando pelo tempo corrente
  2. Servidor de tempo responde o mais rápido possível, com uma mensagem contendo o tempo corrente C_{UTC}
  3. Quando o transmissor obtém uma resposta, ajusta seu clock.
- Problemas:
  - O tempo não pode retroceder;
  - Há atraso no envio das mensagens.

Relógios Físicos
Protocolo de tempo de Rede - NTP

- Solução:
  - Corrigir a hora mudando o tempo gradativamente:
    - Atrasa atualizando a cada 9ms ao invés de 10ms;
    - Adianta atualizando a cada 11ms ao invés de 10ms;
  - Calcular o atraso com base na medida do tempo da transmissão através do uso do deslocamento θ e estimativa de atraso δ.
  - Resultado é melhorado usando histórico de médias.
- Qual horário prevalece?
  - Cada computador possui seu "estrato".
    - Relógio de referência: 0;
    - Servidor que possui o relógio de referência: 1;
    - Se servidor A possui estrato *k,* B obtém horário de A, B possui estrato (*k*+1)
  - Prevalece o horário do servidor com menor estrato.

Relógios Físicos
Protocolo de tempo de Rede - NTP

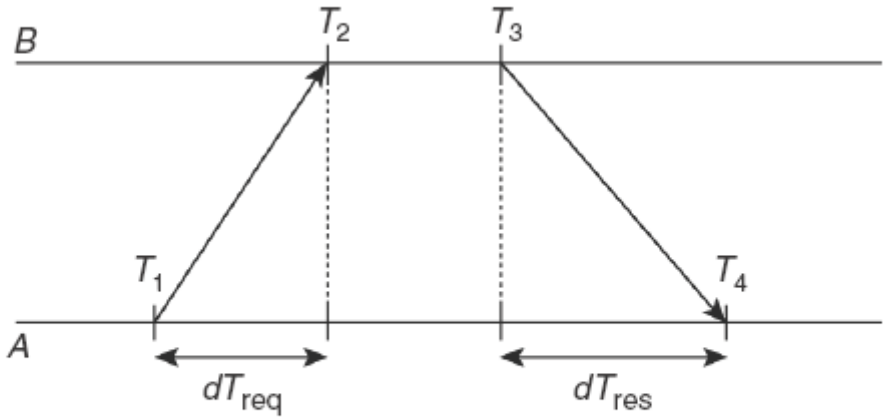


Figura 6.6 Obtenção da hora corrente por meio de um servidor de tempo.

- $\theta=[(T2-T1)+(T3-T4)]/2$
  - Será o tempo de erro entre os relógios
- $\delta=[(T2-T1)-(T4-T3)]/2$
  - Será o atraso estimado do envio

Relógios Físicos
Algoritmo de Berkeley

- Ao contrário do NTP, onde o servidor de tempo é passivo, o algoritmo Berkeley consulta todas as máquinas de tempos em tempos, obtendo o horário de cada máquina;
- Gera uma média de todas as horas, e informa a todos os computadores o deslocamento de tempo a ser feito
  - O servidor muda inclusive a própria hora!

Relógios Físicos
Algoritmo de Berkeley

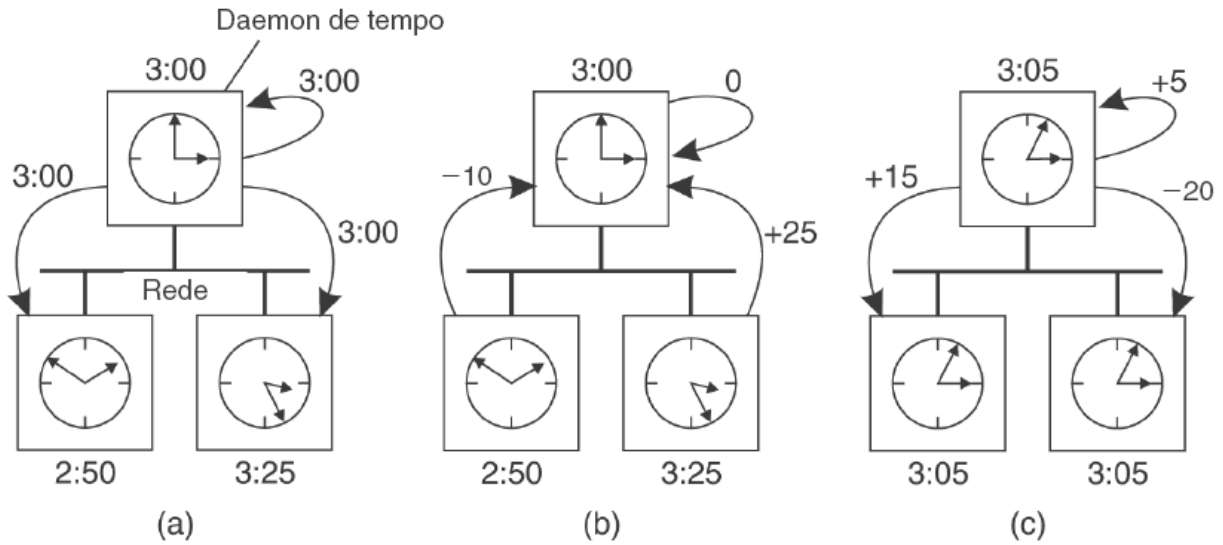


Figura 6.7 (a) O daemon de tempo pergunta a todas as outras máquinas os valores marcados por seus relógios. (b) As máquinas respondem. (c) O daemon de tempo informa a todas como devem ajustar seus relógios.

Relógios Físicos
Sincronização de relógios em redes sem fio

- Usa **Sincronização em broadcast de referência** – RBS.
  - Um dos objetivos é economia de energia;
  - Não adota que há dispositivos com hora real;
  - Visa mera sincronização (como Berkeley);
  - Não utiliza duas vias para sincronização:
    - Nó transmite uma mensagem de referência *m*;
    - Cada nó *p* registra a hora $T_{p,m}$ em que recebeu *m*;
    - Deslocamento de tempo entre nós *p* e *q* é dado pela média aritmética das diferenças de tempo de chegada de cada mensagem *m* trocadas entre *p* e *q.*
    - Cada máquina mantém o deslocamento médio de tempo entre si, não necessitando ajustar o relógio (economizando energia)!
    - Pode-se aplicar outras técnicas como regressão linear padrão para minimizar efeito cumulativo pelo tempo.

Relógios Físicos
Sincronização de relógios em redes sem fio

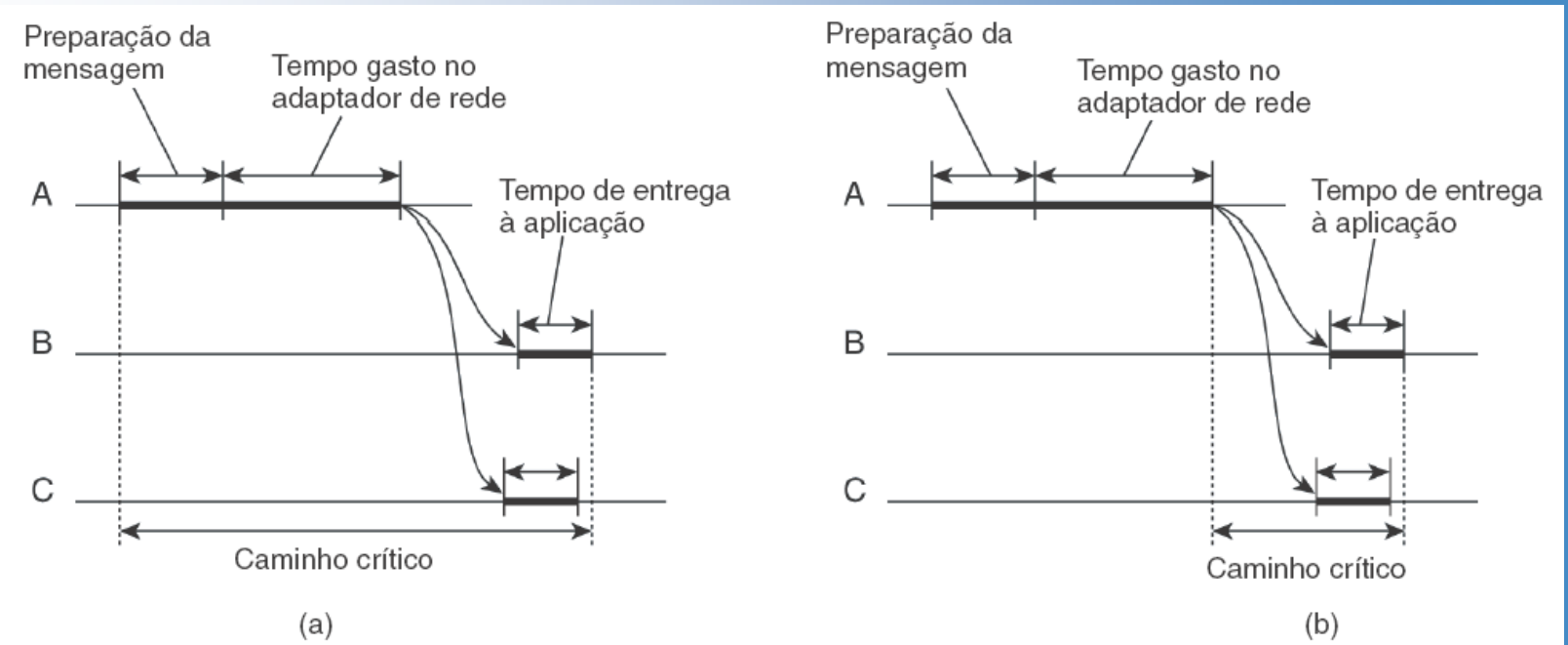


Figura 6.8 (a) Caminho crítico usual na determinação de atrasos de rede. (b) Caminho crítico no caso de RBS.

Sincronização

1. Sincronização de Relógios
2. Relógios Lógicos
3. Exclusão Mútua
4. Posicionamento global de nós
5. Algoritmos de eleição

Relógios Lógicos

- Até o momento foi considerada a sincronização de relógios como naturalmente relacionada com a hora real.
  - Lamport mostrou que, embora a sincronização de relógios seja possível, não precisa ser absoluta:
    - É necessário sincronizar processos que não interagem entre si? Não!
  - Soluciona problemas relativos a ordem de eventos independentemente da hora real.

Relógios Lógicos
Problemas na ordenação de eventos

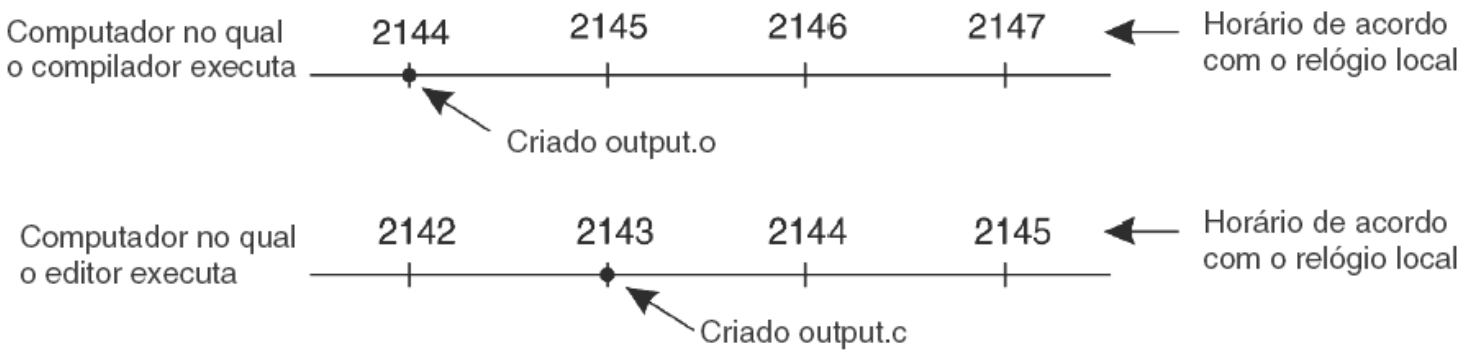


Figura 6.1 Quando cada máquina tem seu próprio relógio, um evento que ocorreu após outro evento pode, ainda assim, receber um horário anterior.

Relógios Lógicos
Relógios Lógicos de Lamport

- Definida a relação **acontece antes**:
  - Se *a* e *b* são eventos do mesmo processo:
    - Se *a* acontece antes de *b*: $a \rightarrow b$.
    - Sendo *a* é o envio da mensagem e *b* o recebimento da mensagem: $a \rightarrow b$.
    - Se $a \rightarrow b$ e $b \rightarrow c$, então $a \rightarrow c$ (propriedade transitiva).
    - Se *x* e *y* acontecem em processos diferentes que não trocam mensagens, então tanto $x \rightarrow y$ quanto $y \rightarrow x$ são **falsas**! Esses processos são ditos concorrentes.
    - Tempos são medidos em função: se $a \rightarrow b$, $C(a) < C(b)$

Relógios Lógicos
Relógios Lógicos de Lamport

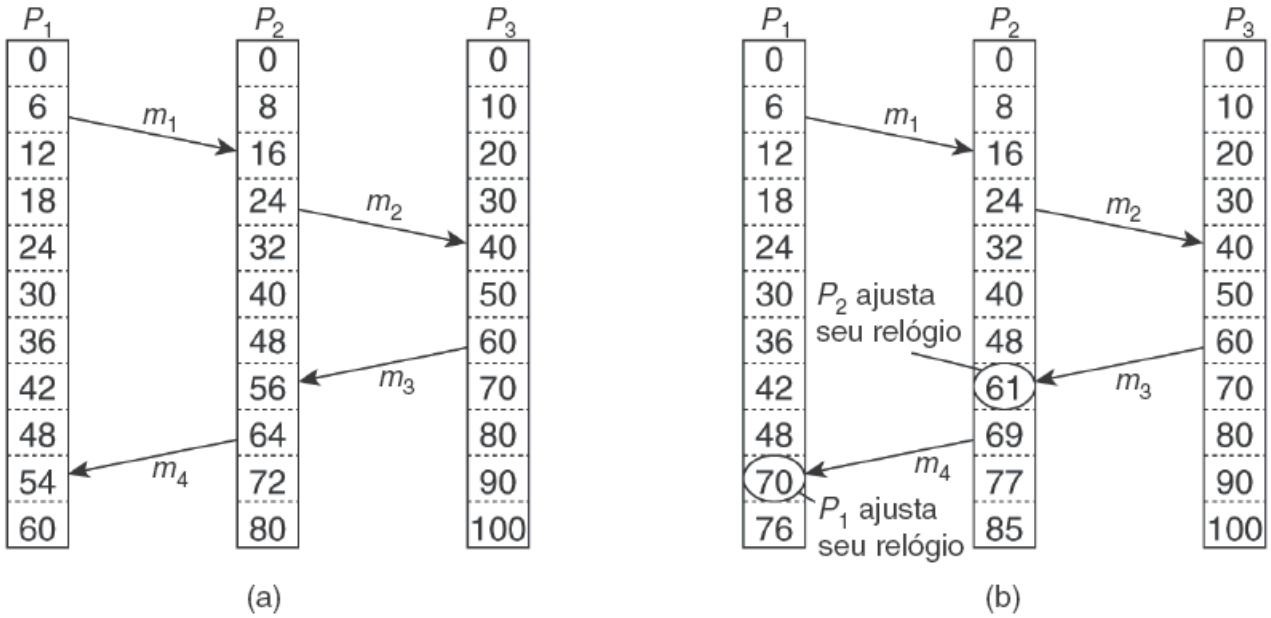


Figura 6.9 (a) Três processos, cada um com seu próprio relógio. Os relógios funcionam a taxas diferentes. (b) O algoritmo de Lamport corrige os relógios.

Relógios Lógicos
Arquitetura - Relógios Lógicos de Lamport

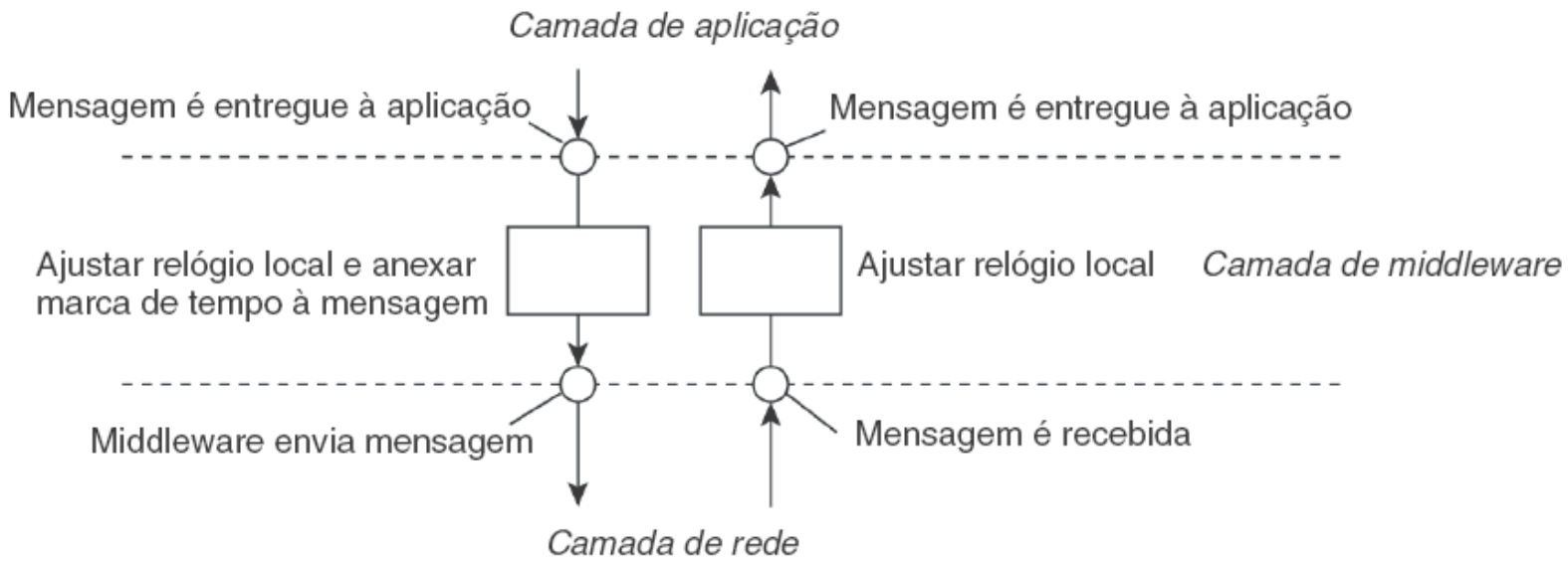


Figura 6.10 Posicionamento de relógios lógicos de Lamport em sistemas distribuídos.

Relógios Lógicos
Relógios Lógicos de Lamport

- **Ordenação total de eventos**: Uma operação multicast pela qual todas as mensagens são entregues na mesma ordem a cada receptor.
  - Cada mensagem será enviada em multicast e sempre transportará a marca de tempo (lógico) de seu remetente;
  - Mensagens são ordenadas em fila de cache local pela marca lógica de tempo;
  - Quando uma mensagem é recebida, a mesma é adicionada a seu cache local, e uma mensagem de reconhecimento é enviada em multicast;
  - Mensagens só podem ser entregues à aplicação após reconhecimento de todos processos, apenas quando forem a primeira mensagem da fila;

Relógios Lógicos
Uso em Multicast Totalmente Ordenado

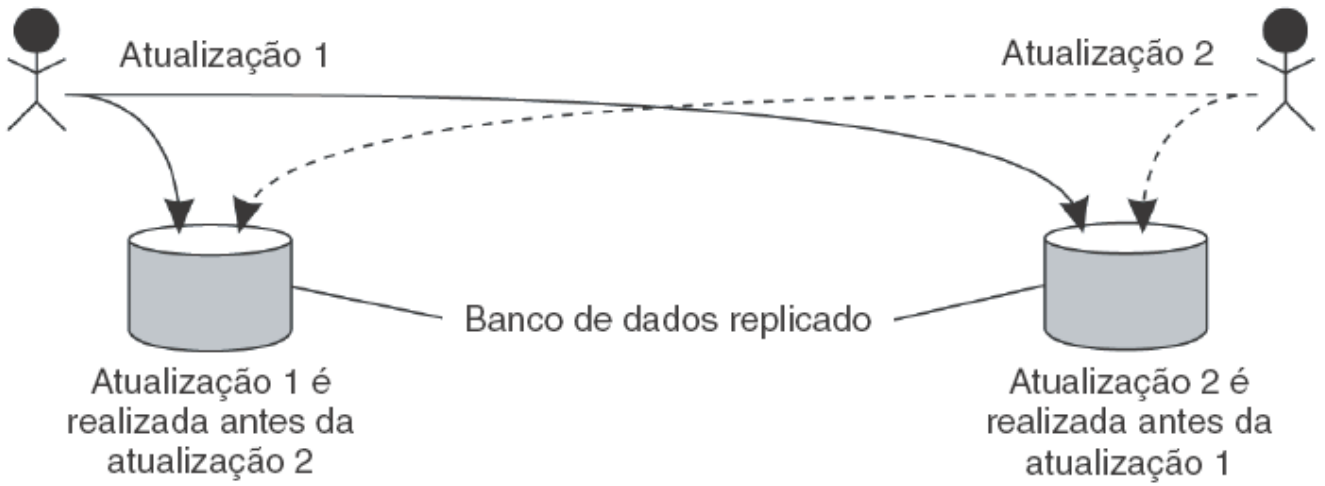


Figura 6.11 Atualização de banco de dados replicado que o deixa em estado inconsistente.

Relógios Lógicos
Relógios Lógicos Vetoriais

- Tenta solucionar problemas de **causalidade** apresentados nos relógios de Lamport através do uso de **relógios vetoriais**.
- Cada processo P_i mantém um vetor VCi com as propriedades:
  1. $VC_i[i]$ é o número de eventos que ocorreram em P_i até o instante em questão;
  2. Se $VC_i[j] = k$, então P_i sabe que k eventos ocorreram em P_j. Portanto, P_i conhece o tempo local de P_j.

Relógios Lógicos
Causalidade em relógios lógicos vetoriais

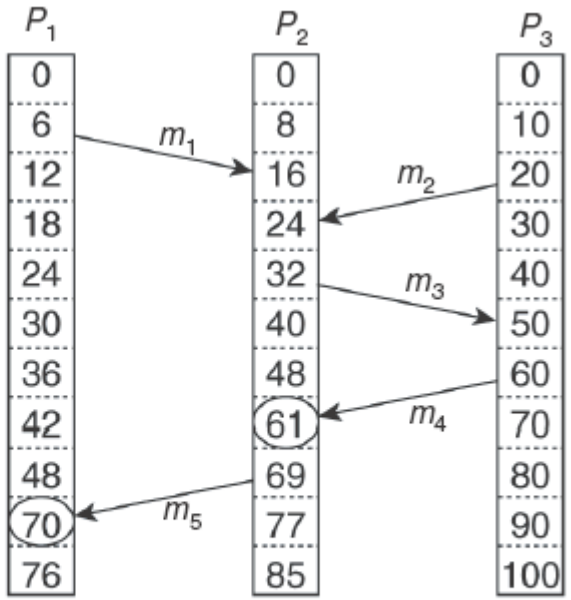


Figura 6.12 Transmissão de mensagens concorrentes com utilização de relógios lógicos.

Relógios Lógicos
Relógios Lógicos Vetoriais – Imposição Causal

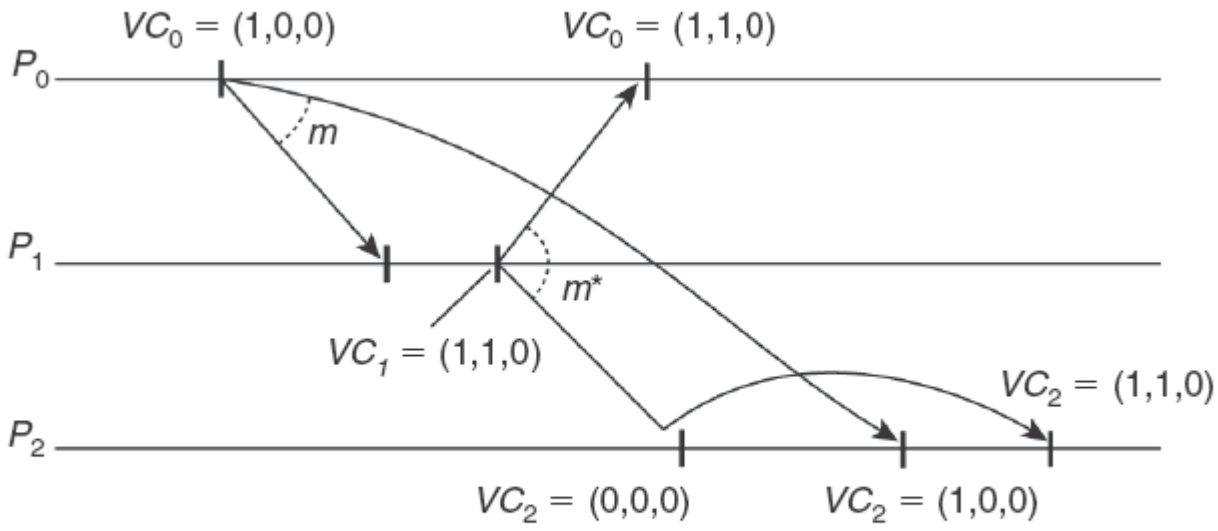


Figura 6.13 Imposição de comunicação causal.

Sincronização

1. Sincronização de Relógios
2. Relógios Lógicos
3. Exclusão Mútua
4. Posicionamento global de nós
5. Algoritmos de eleição

Sincronização
Exclusão Mútua

- Como garantir que o acesso concorrente de recursos não gere situações de inconsistência de dados?
- Através da *Exclusão Mútua*, que, em S.D. pode ser dividida em duas categorias:
  - *Baseadas em fichas;*
    - Evita inanição (Starvation)
    - Fácil evitar deadlocks
  - *Baseadas em permissão.*
    - Processo que quer recursos deve antes pedir permissão aos demais

Sincronização
Algoritmo centralizado

- Simula o que é feito em um sistema monoprocessador:
  - Um processo é eleito como coordenador;
  - Sempre que um processo quiser acessar determinado recurso, é necessário pedir permissão para acessar o mesmo ao coordenador, através de uma mensagem;
  - O coordenador permite acesso ao recurso através de uma mensagem de concessão, desde que nenhum outro processo esteja acessando o recurso neste momento.

Sincronização
Algoritmo centralizado

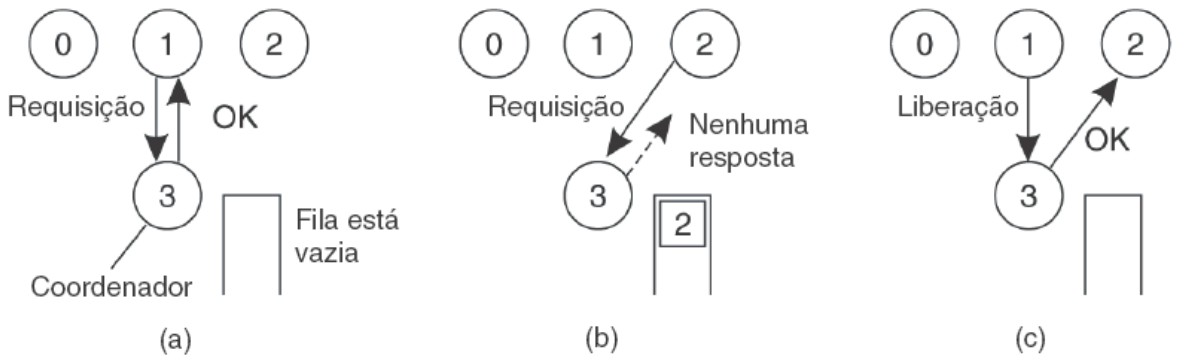


Figura 6.14 (a) O processo 1 solicita ao coordenador permissão para acessar um recurso compartilhado. A permissão é concedida. (b) Depois, o processo 2 solicita permissão para acessar o mesmo recurso. O coordenador não responde. (c) Quando o processo 1 libera o recurso, informa ao coordenador, que então responde a 2.

- Prós:
  - ☺ Simples, fácil de entender e de implementar;
  - ☺ É justo (segue o FCFS);
  - ☺ Como no FCFS, não há inanição (starvation);
- Contras:
  - ☹ Como todo sistema centralizado: *um ponto de erro* e *um ponto de gargalo*;
  - ☹ Processos não conseguem distinguir coordenador inativo de *permissão negada*;

Sincronização
Algoritmo descentralizado

- Usar um algoritmo de votação em um sistema baseado em DHT:
  - Cada recurso é replicado *n* vezes;
  - Cada réplica possui um coordenador;
  - Processo requisitante precisa receber voto majoritário de coordenadores: $m > n/2$
  - Caso a permissão seja negada (processo obtém menos que *m* votos), o mesmo desistirá do recurso por um período de tempo aleatório, e fará a tentativa novamente mais tarde.

Sincronização
Algoritmo descentralizado

- Prós
  - ☺ Torna a solução centralizada original menos vulnerável a falhas de um único coordenador;
  - ☺ Possibilita uso de réplicas de recursos;
- Contras
  - ☹ Não protege contra *starvation*;
  - ☹ Se muitos nós querem acessar o mesmo recurso, nenhum nó conseguirá votos suficientes, e os recursos deixarão de ser usados;
  - ☹ Há uma probabilidade positiva (embora muito baixa) de permitir a 2 nós acesso ao mesmo recurso, ao mesmo tempo (ou seja, não garantir a exclusão mútua).

Sincronização
Algoritmo distribuído

- Para muitos, um algoritmo correto segundo as leis da probabilidade não é bom o bastante (incluindo o professor).
- Para resolver esse problema, pesquisadores procuraram algoritmos distribuídos determinísticos de exclusão mútua.
  - Lamport apresentou o primeiro em 1978;
  - Ricart e Agrawala o tornaram mais eficiente (1981).
- Será analisada a versão de Ricart e Agrawala.

Sincronização
Algoritmo distribuído

- Requer ordenação total de todos os eventos no sistema;
  - Para isso será usado... Algoritmo de Lamport!
- Funcionamento:
  - Quando processo deseja acessar um recurso compartilhado, monta uma mensagem que contém o nome do recurso, seu número de processo e a hora corrente (lógica).
  - Envia mensagem para todos processos, inclusive ele mesmo (similar a *broadcast* ou *multicast*);
  - Quando um processo recebe uma mensagem de requisição de outro, executa uma ação de acordo com seu próprio estado em relação ao recurso:

Sincronização
Algoritmo distribuído

- Funcionamento (continuação):
  1. Se o receptor não estiver acessando o recurso nem quer acessá-lo, devolve “OK” ao remetente
  2. Se já tem acesso ao recurso, não responde e coloca requisição em uma fila;
  3. Se receptor também quer acessar o recurso, mas ainda não possui a permissão, compara a marca de tempo da mensagem que chegou com a marca de tempo da mensagem que enviou a todos. A mais baixa vence.
  - O processo remetente aguarda recebimento de todas as respostas;
  - Quando houver permissão de todos, processo acessa o recurso;
  - Processo libera o recurso enviando um “OK” a todos os processos

Sincronização
Algoritmo distribuído

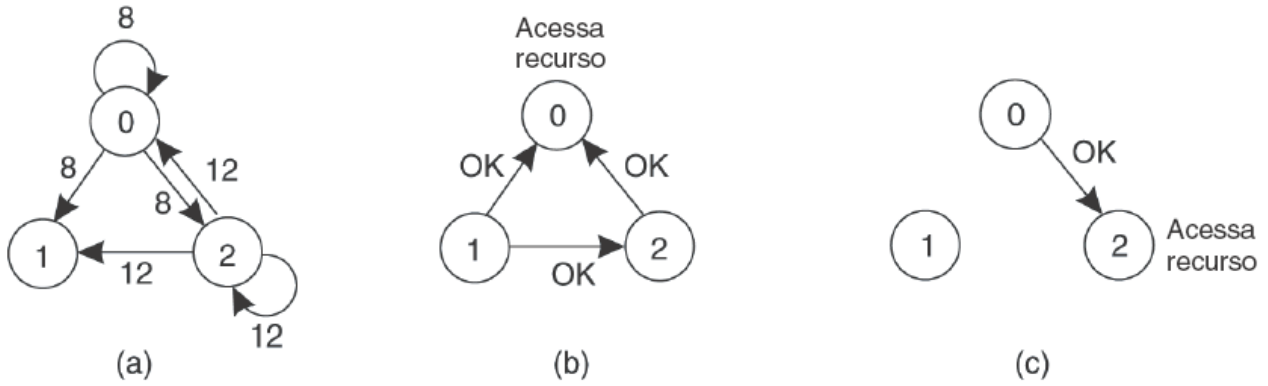


Figura 6.15 (a) Dois processos querem acessar um recurso compartilhado no mesmo momento. (b) O processo 0 tem a marca de tempo mais baixa, portanto vence. (c) Quando o processo 0 conclui, também envia uma mensagem *OK*, portanto, agora, 2 pode seguir adiante.

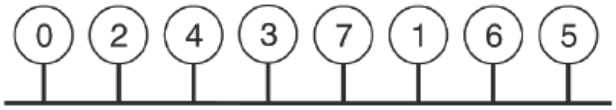
- Vantagens:
  - ☺ Exclusão mútua é garantida;
  - ☺ Não há deadlock;
  - ☺ Não há *starvation*;
  - ☺ Número de mensagens: 2(*n*-1) //*n*=número de nós
  - ☺ Não existe nenhum ponto de falha único

Sincronização
Algoritmo distribuído

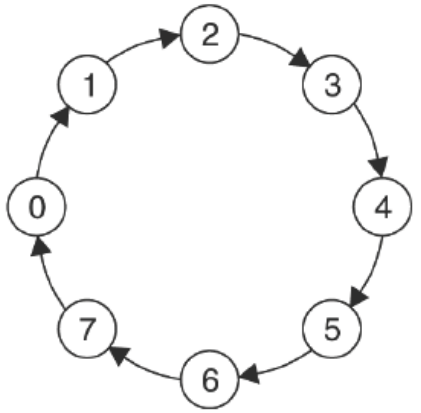
- Desvantagens:
  - ☹ Ponto de falha único foi substituído por *n* pontos de falha;
  - ☹ Deve usar comunicação multicast ou manter uma lista de associação ao grupo em cada processo, incluindo processos que entram no grupo ou caem (funciona melhor com poucos processos que se mantêm estáveis);
  - ☹ Trocou 1 gargalo por *n* gargalos;
  - ☹ **Portanto é: Mais lento, mais complicado, mais caro e menos robusto que o original centralizado.**

Sincronização
Algoritmo Token Ring

- Baseado nas redes *token ring* estudadas em rede, mas não é necessariamente uma rede física: pode ser uma rede *lógica*



(a)



(b)

Figura 6.16 (a) Grupo de processos não ordenados em uma rede. (b) Um anel lógico é construído em software.

Sincronização – Exclusão Mútua
Comparação entre os quatro algoritmos

Algoritmo	Mensagens por entrada/saída	Atraso antes da entrada (em número de tempos de mensagens)	Problemas
Centralizado	3	2	Queda do coordenador
Descentralizado	$3mk$, $k = 1,2,\ldots$	$2m$	Inanição, baixa eficiência
Distribuído	$2(n-1)$	$2(n-1)$	Queda de qualquer processo
Token Ring	1 a ∞	0 a $n-1$	Ficha perdida; processo cai

Tabela 6.1 Comparação entre quatro algoritmos de exclusão mútua.

Sincronização

1. Sincronização de Relógios
2. Relógios Lógicos
3. Exclusão Mútua
4. Posicionamento global de nós
5. Algoritmos de eleição

Sincronização
Posicionamento Global de nós

- É difícil que um nó monitore outros em um sistema distribuído quando há um crescimento do número de nós. Para solucionar o problema utiliza-se o posicionamento global dos nós, que informa onde cada nó está.
- Em **redes de sobreposição geométrica** é designada uma posição em um espaço geométrico *n-dimensional* a cada nó, de modo que a distância entre cada nó represente uma métrica de desempenho no mundo real.
  - Ex.: d(P,Q) para distância de latência entre os nós P e Q.
  - Útil para redirecionar para réplica de um servidor com menor tempo de resposta para o cliente;
  - Identificar melhor posicionamento para réplicas;
  - Roteamento baseado em posição.

Posicionamento Global de Nós

- O posicionamento global de nós requer *m+1* medições de distância.
- Como no GPS, o cálculo pode ser feito por:

$$\sqrt{(x_i - x_p)^2 + (y_i - y_p)^2} \quad (i = 1,2,3)$$

- D_i equivale a metade do RTT, sendo diferente ao longo do tempo.

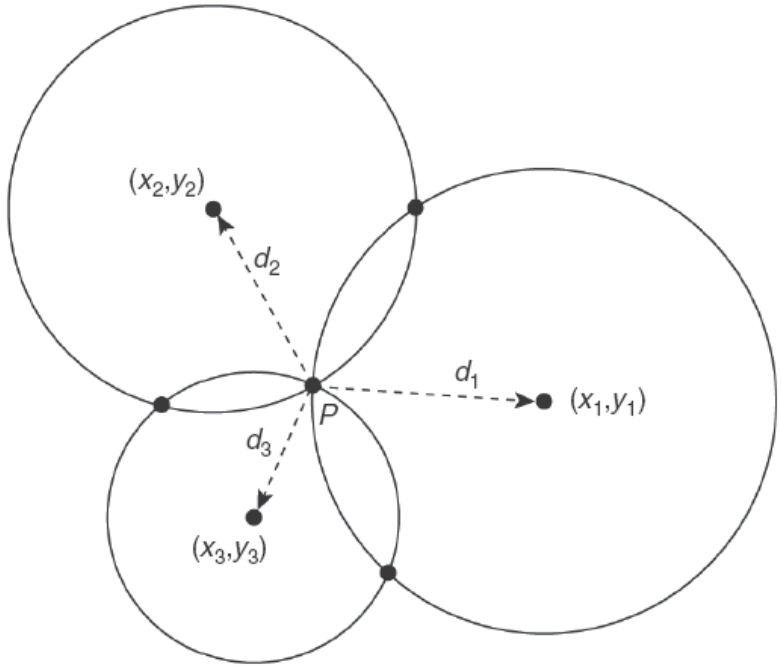


Figura 6.17 Cálculo da posição de um nó em um espaço bidimensional.

Posicionamento Global de Nós

- A variação em relação ao tempo pode gerar inconsistências.
  - As medições podem violar a **desigualdade triangular**, que implica que para 3 pontos arbitrários P, Q e R:
    - $d(P,R) \leq d(P,Q) + d(Q,R)$;

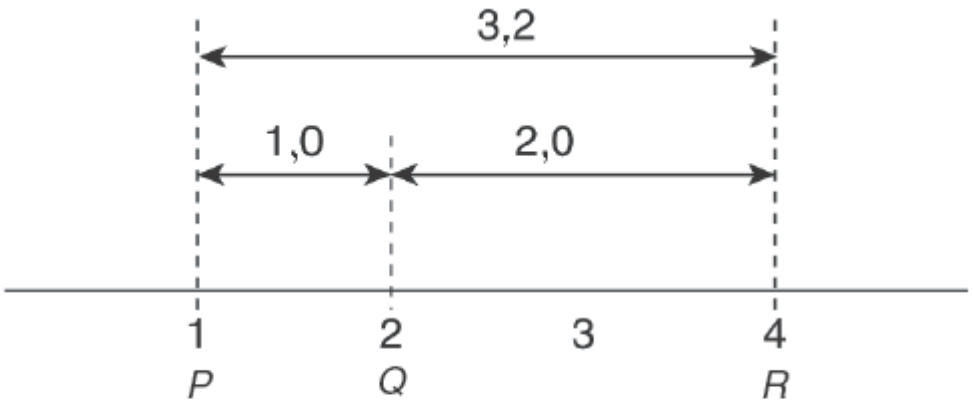


Figura 6.18 Medições inconsistentes de distâncias em um espaço unidimensional.

Sincronização

1. Sincronização de Relógios
2. Relógios Lógicos
3. Exclusão Mútua
4. Posicionamento global de nós
5. Algoritmos de eleição

Algoritmos de Eleição

- Usados quando há necessidade de um nó agir como coordenador. São exemplos tradicionais de algoritmos de eleição:
  - Algoritmo do valentão;
  - Algoritmo de anel;
- Além disso, existem soluções para ambientes específicos, como:
  - Algoritmos para *Ad Hoc*;
  - Algoritmos para sistemas de grande escala (como P2P)

Algoritmo do valentão

- Inventado por Garcia-Molina (1982);
  - Todos nós possuem um identificador;
  - Sempre que um nó qualquer *P* nota que o coordenador não responde, *P* inicia uma eleição:
    1. *P* envia uma mensagem ELEIÇÃO a todos os processos de números mais altos;
    2. Se nenhum responder, *P* vence a eleição e se torna o coordenador;
    3. Se um dos processos de número mais alto responder, ele toma o poder e o trabalho de *P* está concluído.
  - Dessa forma é eleito o nó com maior identificador!

Algoritmo do Valentão

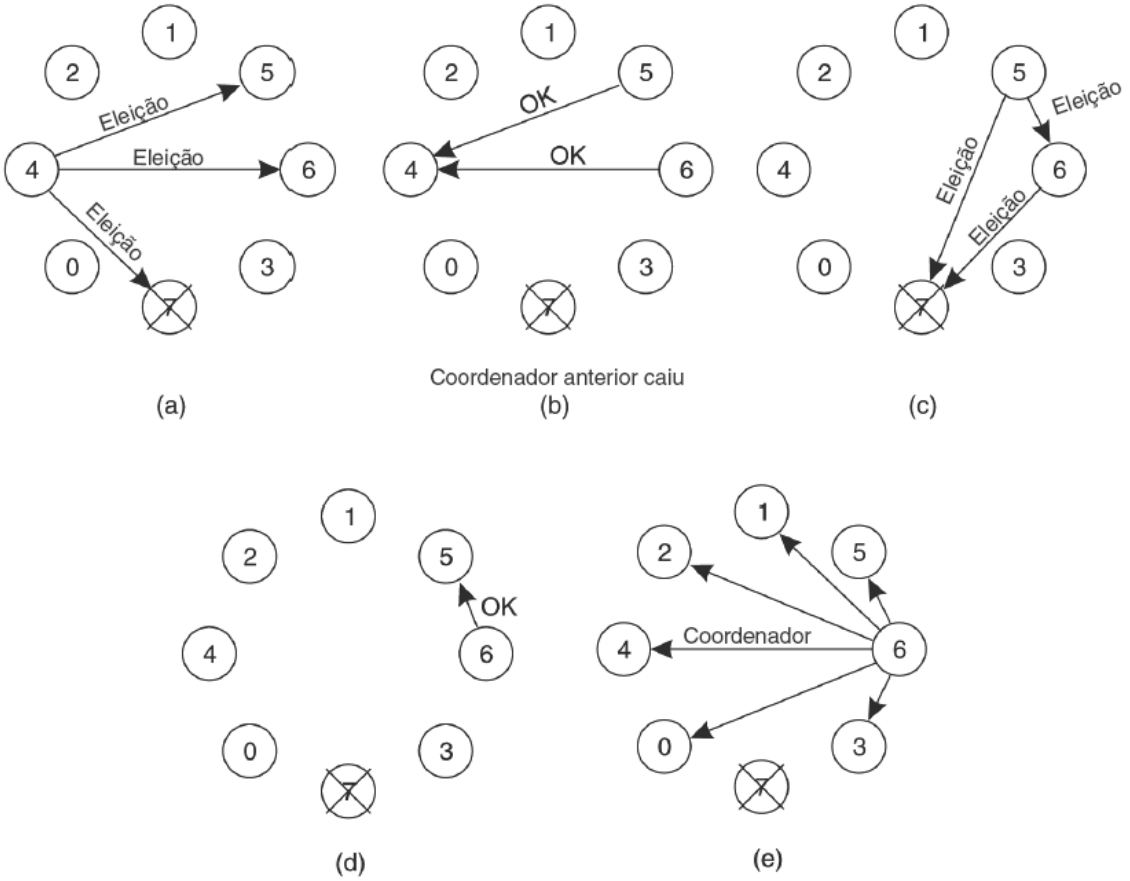


Figura 6.19 Algoritmo de eleição do valentão. (a) O processo 4 convoca uma eleição. (b) Os processos 5 e 6 respondem e mandam 4 parar. (c) Agora, cada um, 5 e 6, convoca uma eleição. (d) O processo 6 manda 5 parar. (e) O processo 6 vence e informa a todos.

Algoritmo do Anel

- Baseado na utilização de anel (físico ou lógico), mas não usa ficha:
  - Quando qualquer processo nota que o coordenador não está funcionando, monta uma mensagem ELEIÇÃO com seu próprio número e o envia a seu sucessor ou próximo que esteja em funcionamento;
  - A cada etapa, o remetente adiciona seu número de modo a se tornar também um candidato à eleição de coordenador;
  - Ao chegar ao primeiro, este envia a mensagem COORDENADOR contendo o maior identificador da lista.

Algoritmo de eleição por Anel
Ex: Nós 2 e 5 iniciam eleição simultaneamente

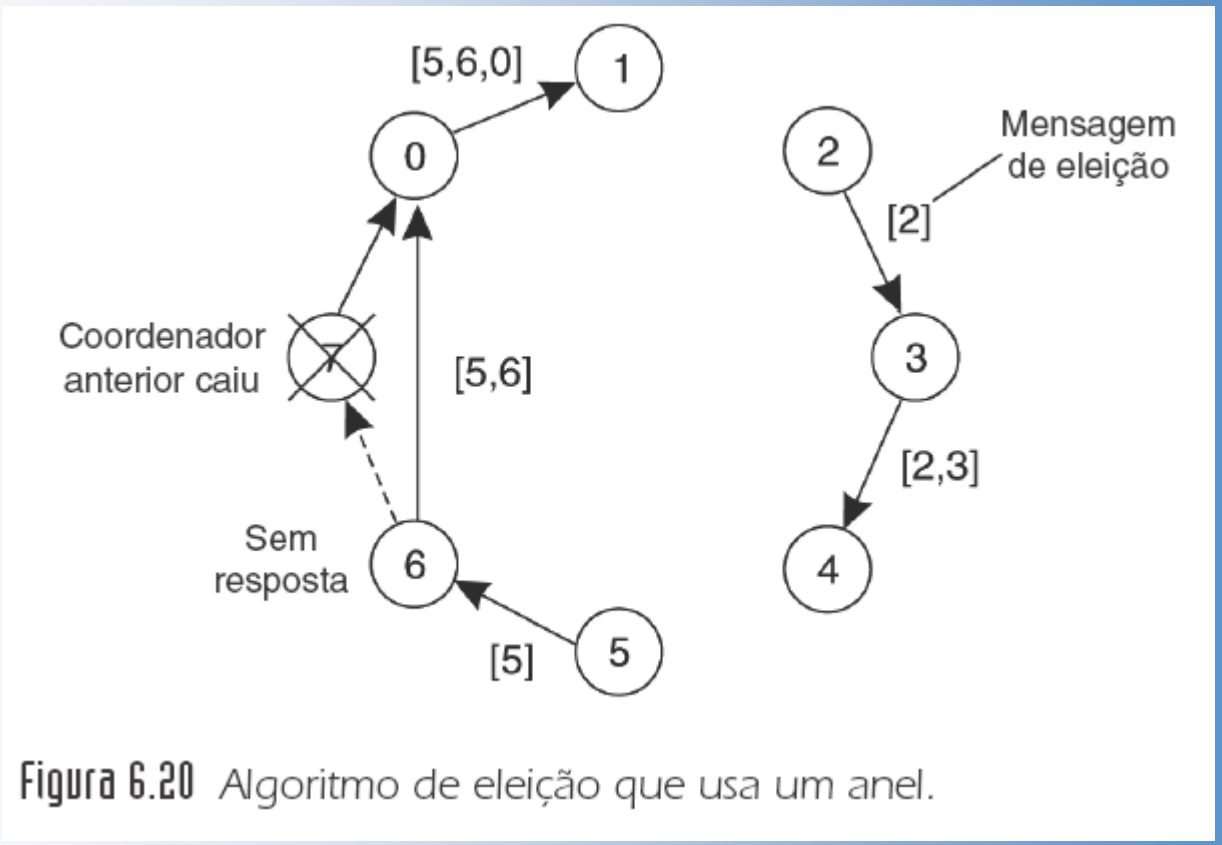


Figura 6.20 Algoritmo de eleição que usa um anel.

Eleições em Ambientes sem fio

- Necessários em ambientes onde a troca de mensagens não é confiável e a topologia da rede muda com freqüência, como *ad hoc*.
- Visa eleger o *melhor* líder em vez de apenas um líder aleatório (ex. o que possui maior bateria restante).
- Será adotado que a rede é *ad hoc* mas os nós não podem se mover.

Eleições em Ambientes sem fio

- Um nó *fonte* P inicia a eleição enviando a mensagem ELEIÇÃO a seus vizinhos;
  - Quando um nó Q recebe a mensagem ELEIÇÃO pela primeira vez, designa o remetente como seu pai e envia uma mensagem ELEIÇÃO a todos vizinhos exceto seu pai;
  - Se Q recebe uma mensagem ELEIÇÃO de um nó não-pai, apenas confirma o recebimento;
- Se todos vizinhos de um nó R já possuem um pai, R é um nó folha, portanto pode reportar de volta a Q, passando informações úteis.
- Quando Q recebe todas as respostas, reporta a seu pai qual é o melhor dos recebidos por Q;
- O nó P recebe os melhores candidatos e decide pelo melhor, informando a todos via broadcast.

Eleições em Ambientes sem Fio

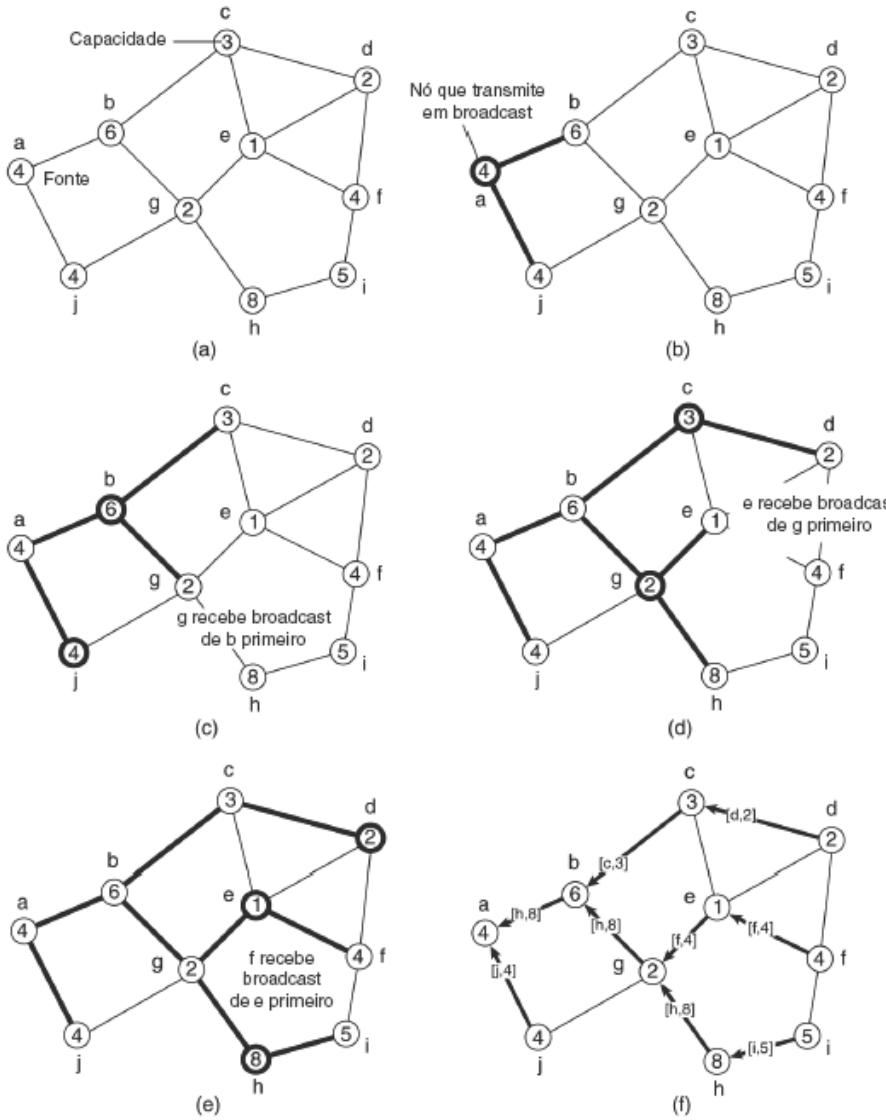


Figura 6.21 Algoritmo de eleição em uma rede sem fio, com o nó *a* como o fonte. (a) Rede inicial. (b)–(e) A fase de montagem da árvore (última etapa do broadcast pelos nós *f* e *i* não é mostrada). (f) Reportando o melhor nó ao fonte.

Eleições em sistemas de grande escala

- Há situações em que é necessário trabalhar com redes maiores e é necessário eleger maior quantidade de pares, ex.: Superpares em P2P
- Requisitos a serem cumpridos por superpar:
  1. Nós normais devem ter baixa latência de acesso com superpares;
  2. Superpares devem estar uniformemente distribuídos pela rede de sobreposição;
  3. Deve haver uma porção predefinida de superpares em relação ao número total de nós na rede de sobreposição;
  4. Cada superpar não deve precisar atender mais do que um número fixo de nós normais;

Eleições em sistemas de grande escala Superpares com *k* de *m* bits como id

- Uma solução é dada quando se usa *m* bits de identificador, separar os *k* bits da extrema esquerda para identificar superpares;
  - Ex.: $\log_2(N)$ Superpares, m=8,k=3.
  - *p* AND 11100000 = Superpar.
- Problema: não garante posicionamento geométrico para organizar os superpares *uniformemente* pela rede

Eleições em sistemas de grande escala
Eleição de pares por fichas repulsoras

- *N* fichas distribuídas aleatoriamente entre os nós;
- Nenhum nó pode ter mais de uma ficha;
- Fichas possuem uma força de repulsão;
- Um nó que mantiver a ficha por determinado tempo é eleito superpar.

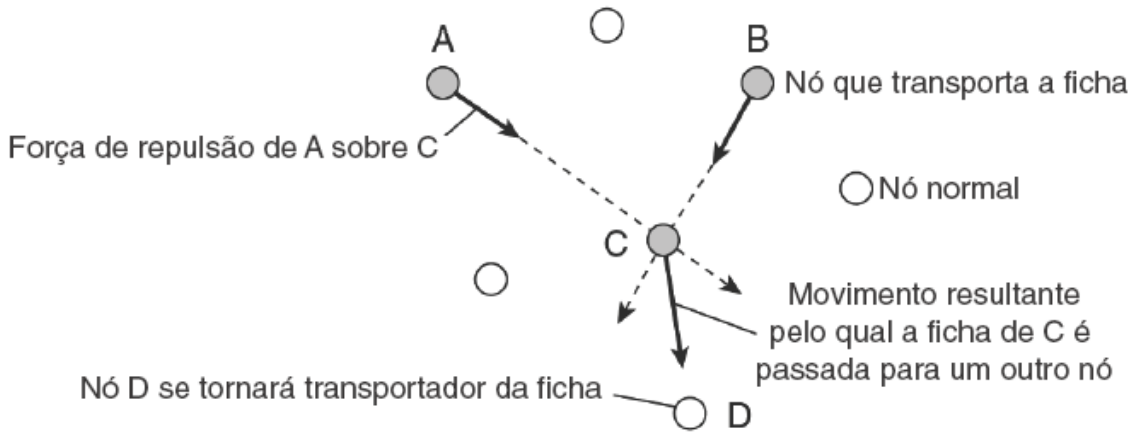


Figura 6.22 Movimentação de fichas em um espaço bidimensional que utiliza forças de repulsão.