Sincronização

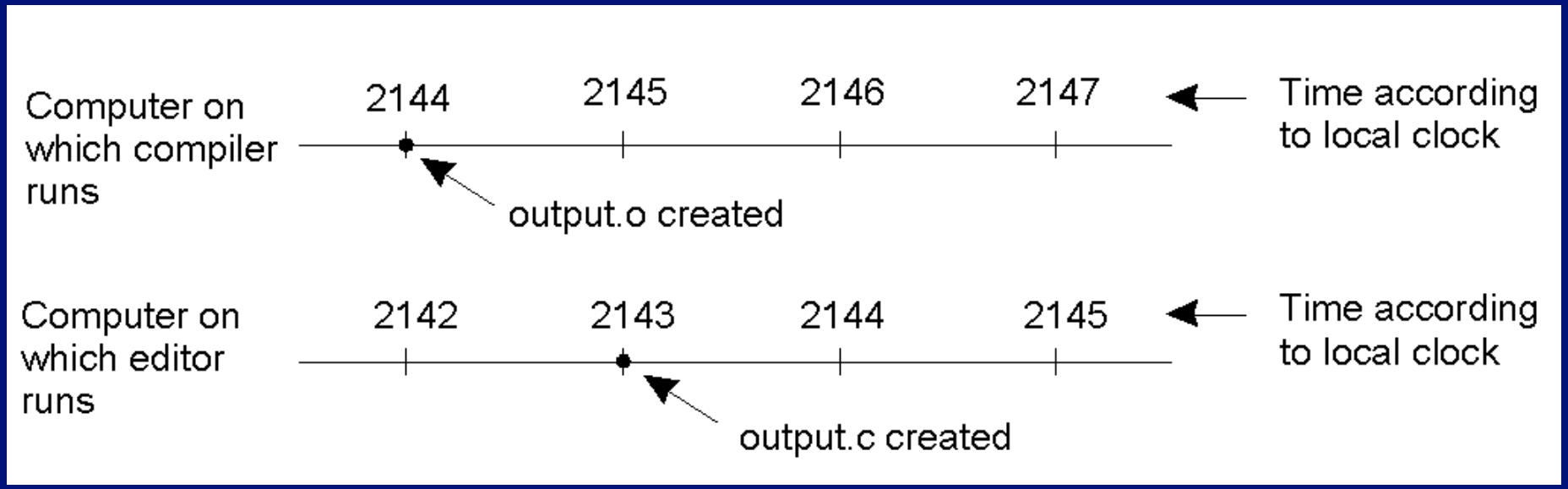
Introdução

- Como os processos se sincronizam entre si
  - Várias máquinas, vários relógios
  - Relógios Físicos
  - Relógios Lógicos
- Exclusão mútua de execução
- Algoritmos eletivos
- Transações atômicas distribuídas

Sincronização em Sistemas Distribuídos

- Em sistemas centralizados o tempo é um conceito não ambíguo
  - A relação “A acontece antes de B” é obtida de forma trivial
  - Há um tempo global
- Em um sistema distribuído, não há tempo global
  - Não há uma fonte única de geração de tempo
  - Isso dificulta obter conformidade em relação ao tempo

Sincronização em Sistemas Distribuídos



É possível sincronizar todos os relógios de um sistema distribuído?

Relógios Físicos

- Noção de tempo físico/real
- Escorregamento do clock
  - Não importa muito em um sistema centralizado
    - O que realmente importa são os tempos relativos
  - Em um sistema distribuído, fará com que os relógios das diferentes máquinas percam a sincronização
    - Exemplo: make
- A sincronização envolve relógios físicos externos
  - Os relógios precisam estar sincronizados entre si
  - Os relógios não podem diferir do tempo real mais que um determinado valor

Relógios Físicos

- Serviço de sincronização de relógios físicos mais amplamente utilizado na Internet
  - NTP - Network Time Protocol
  - Precisão: 1- 50msec

Relógios Lógicos

- O que importa é a ordem na qual os eventos ocorrem
  - E não necessariamente o valor exato do tempo em determinado momento
- A sincronização de relógios lógicos se baseia na relação "happens-before"
  - a → b : a acontece antes de b
  - Todos os processos devem concordar a respeito disso

Relógios Lógicos

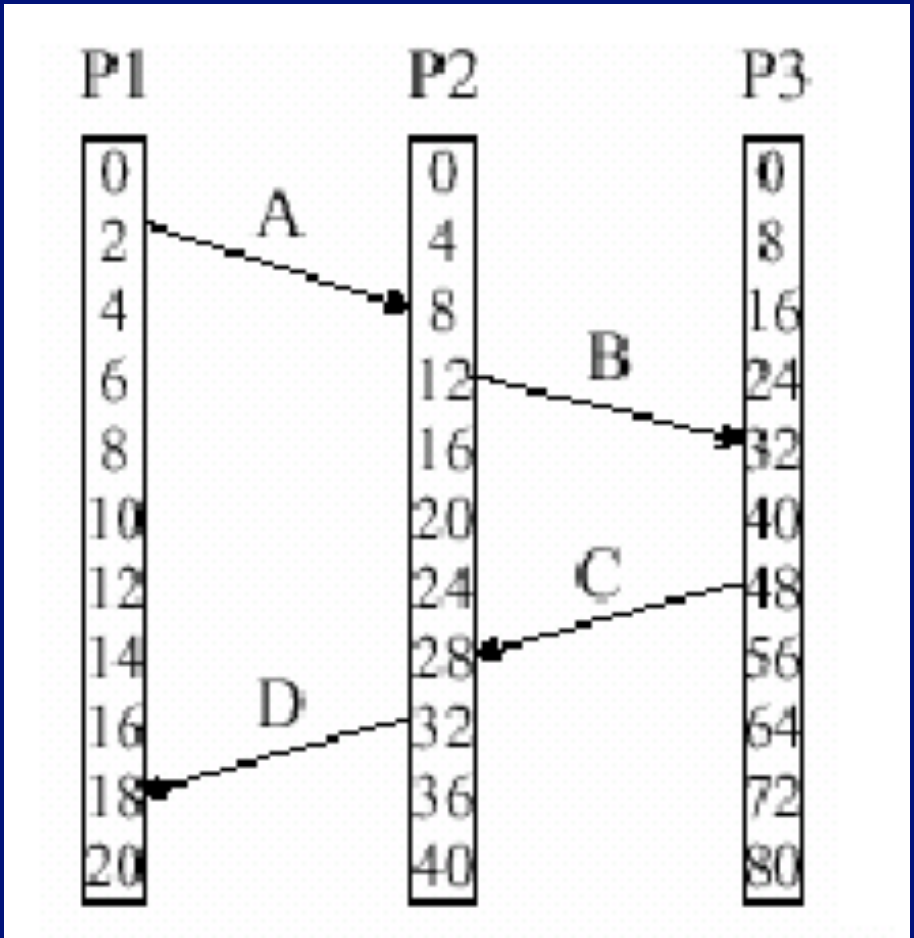
- A relação a → b pode ser observada em duas situações:
  - Se a e b são eventos no mesmo processo, e se a ocorre antes de b
  - Se a representa o envio de uma mensagem por um processo e b representa o recebimento dessa mensagem por outro processo
- A relação a→b também é transitiva
  - a → b e b → c, então a → c
- Se x e y são dois eventos concorrentes, então nem x→y nem y→x são verdadeiros

Relógios Lógicos

- Precisamos de uma forma de atribuir tempos a eventos
  - Relógios Lógicos!!!
  - C(a) é o valor que o relógio lógico C atribui ao evento a
  - Se a→b, então C(a) < C(b)
  - Utilização do algoritmo de Lamport para sincronização dos relógios lógicos

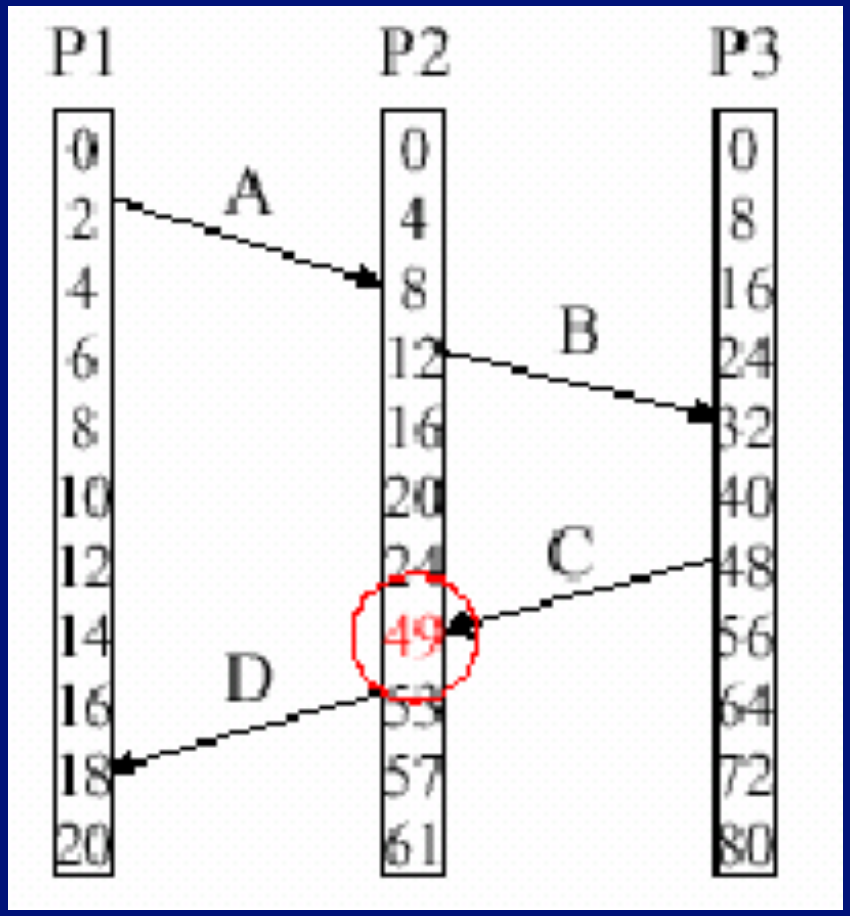
Relógios Lógicos

- Três processos em máquinas diferentes com relógios diferentes

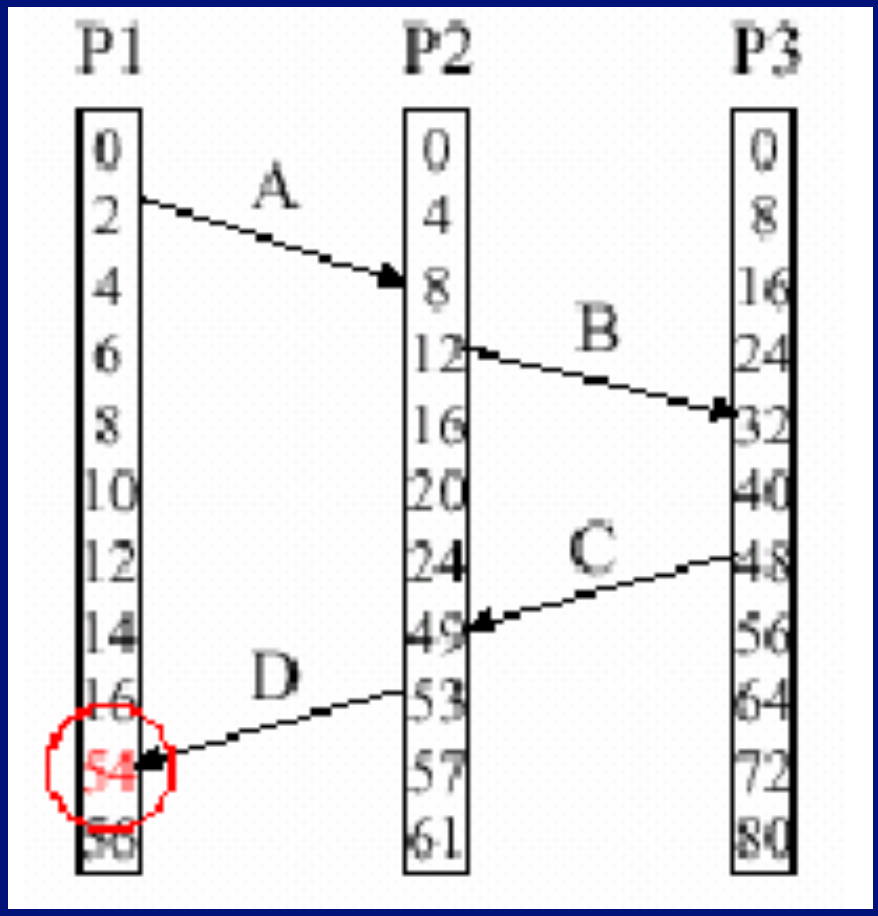


Relógios Lógicos

O processo P2 não pode ter recebido a mensagem C antes dela ter sido enviada por P3.



O processo P1 não pode ter recebido a mensagem D antes dela ter sido enviada por P2.



Relógios Lógicos

- Algoritmo de Lamport para a sincronização de relógios lógicos
  - O relógio lógico C_i é incrementado entre dois eventos sucessivos quaisquer dentro de um mesmo processo
  - Se a é o envio de uma mensagem m a partir de P_i, $T_m = C_i(a)$
  - Quando P_j receber m, $C_j = \max(C_j, T_m + 1)$

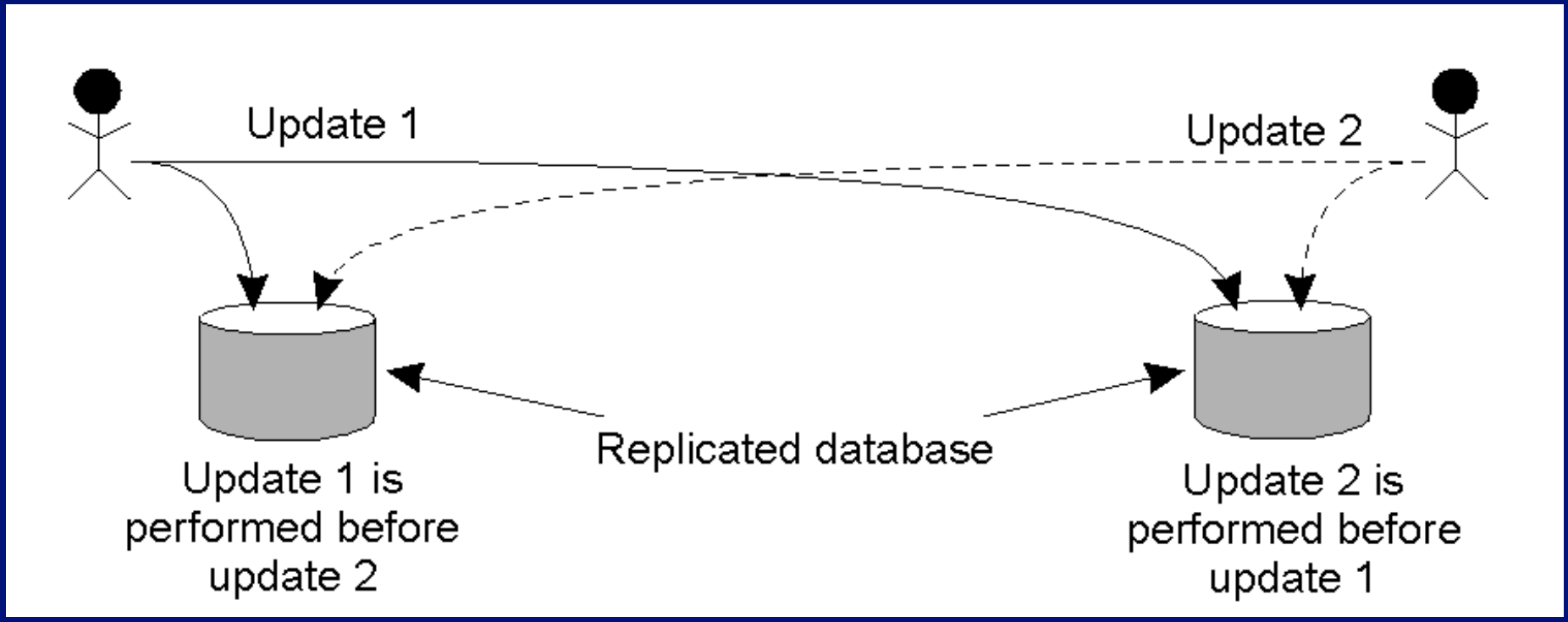
Relógios Lógicos

- Para obter ordenação total de eventos, dois eventos não podem nunca ocorrer exatamente no mesmo instante de tempo
- Usa-se uma relação de precedência entre os processos
  - $C_i(a)$ equivale a $C_i(a).\ P_i$
  - $C_j(b)$ equivale a $C_j(b).\ P_j$
  - $C_i(a).P_i < C_j(b).P_j$ se $C_i(a) < C_j(b)$ ou $C_i(a) = C_j(b)$ e $P_i < P_j$
- A ordenação total depende da função de precedência escolhida

Relógios Lógicos

- Ordenação Total de eventos
  - Se a→b no mesmo processo, então C(a) < C(b)
  - Se a e b são o envio e o recebimento de uma mensagem, então C(a) < C(b)
  - Para quaisquer eventos a e b, C(a) <> C(b)

Exemplo de Uso da Ordenação Total de Eventos



- A falta de ordenação leva a inconsistências nas bases!
- Uma solução: multicast com ordenação total
  - Todas as mensagens são entregues na mesma ordem a cada receptor

Exemplo de Uso da Ordenação Total de Eventos

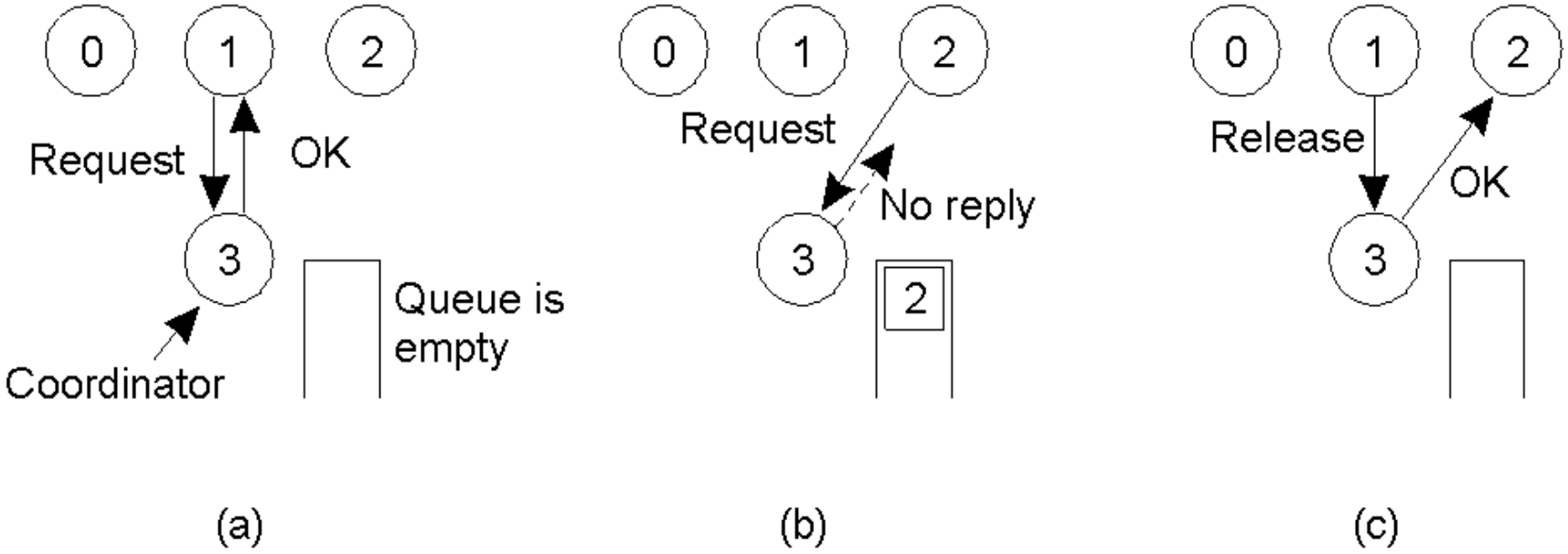
- Implementando o multicast com ordenação total via relógios lógicos
  - Cada mensagem carrega o timestamp com o tempo lógico do transmissor
  - Quando ocorre um multicast, o próprio transmissor também recebe a mensagem
  - Canais FIFO e confiáveis
  - Quando um processo recebe uma mensagem, ele a coloca numa fila local, ordenada de acordo com o seu timestamp, e faz um multicast, confirmando o recebimento
    - $T_{m(ACK)} > T_{M(MSG)}$

Exemplo de Uso da Ordenação Total de Eventos

- Implementando o multicast com ordenação total via relógios lógicos (continuação)
  - Todos os processos terão a mesma cópia da fila local
  - Cada mensagem tem um timestamp diferente (ordenação total)
  - Uma mensagem é consumida quando ela possuir o menor timestamp da fila e tiver sido reconhecida por todos os demais processos

Exclusão Mútua

- Como implementar exclusão mútua e regiões críticas em sistemas distribuídos?
- Solução 1: Um Algoritmo Centralizado
  - Simular um sistema com um único processador



Exclusão Mútua

- Solução 1: Algoritmo Centralizado
  - Garante exclusão mútua
  - É justo
  - Nenhum processo espera indefinidamente (não há *starvation*)
  - É fácil de ser implementado
  - Coordenador é um ponto único de falha
  - Coordenador pode representar um gargalo em termos de desempenho

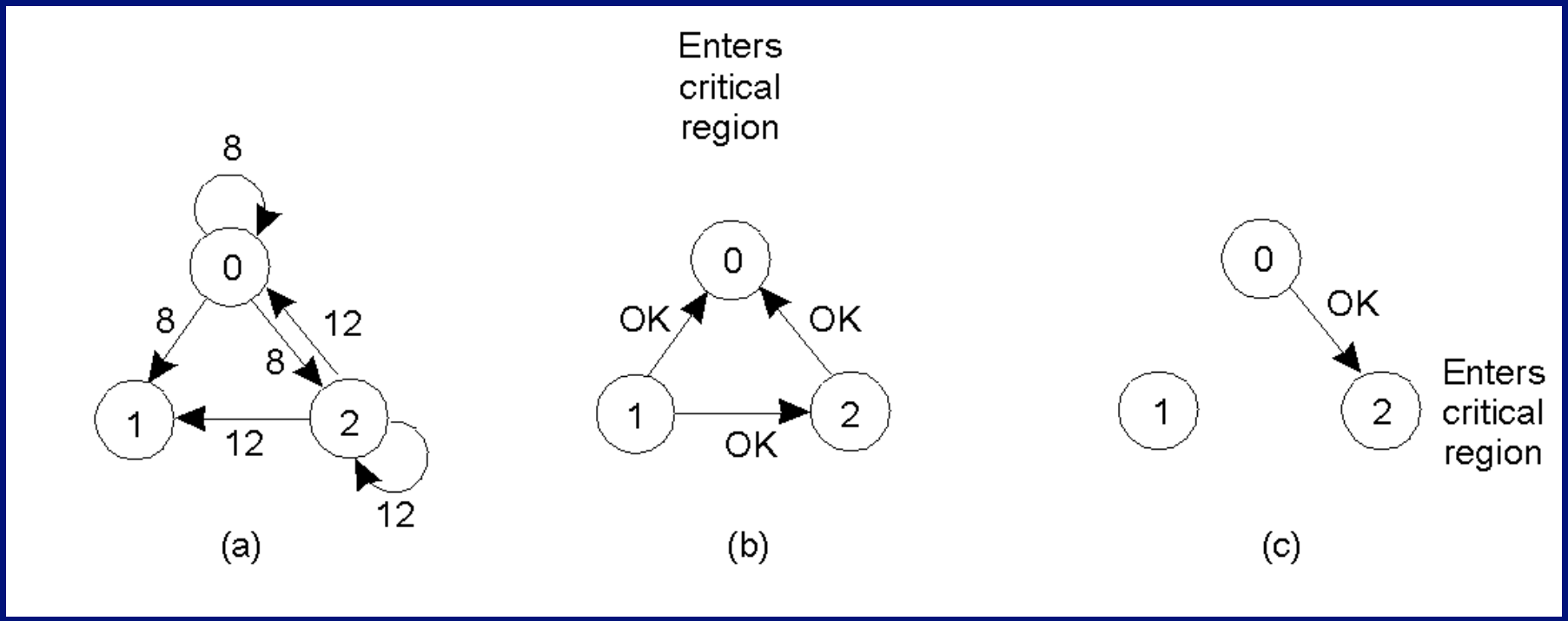
Exclusão Mútua

- Solução 2: Um Algoritmo Distribuído
  - Exige ordenação total de eventos
  - Para entrar em uma região crítica, o processo deve enviar uma mensagem m = {nome_região_crítica, P_i, C_i) para todos os processos (inclusive ele próprio)
  - Quando um processo P_j recebe m de P_i, ele deve:
    - Retornar OK se já não estiver naquela região crítica
    - Não retornar nada se já estiver naquela região crítica e guardar m numa fila
    - P_i deve aguardar até que todos os demais P_j lhe dêem permissão para entrar na região crítica

Exclusão Mútua

Solução 2: Um Algoritmo Distribuído

- E se dois processos quiserem entrar na mesma região crítica ao mesmo tempo?



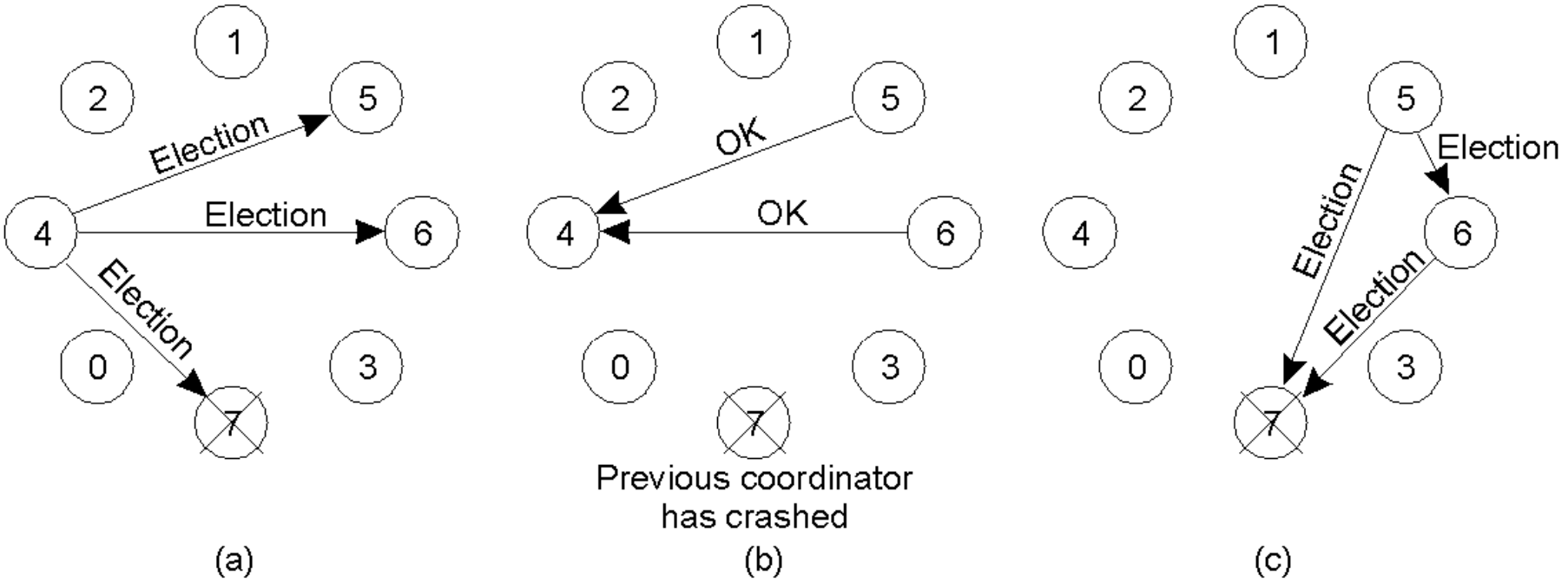
Exclusão Mútua

- Solução 2: Um Algoritmo Distribuído
  - Garante exclusão mútua
  - Não há starvation
  - Gera mais tráfego na rede
  - Não há um ponto único de falha, mas n pontos de falha (n = número de processos)
  - Gerência do grupo é mais difícil
  - Todos os processos estão envolvidos em todas as decisões
    - Cada processo representa um gargalo em potencial

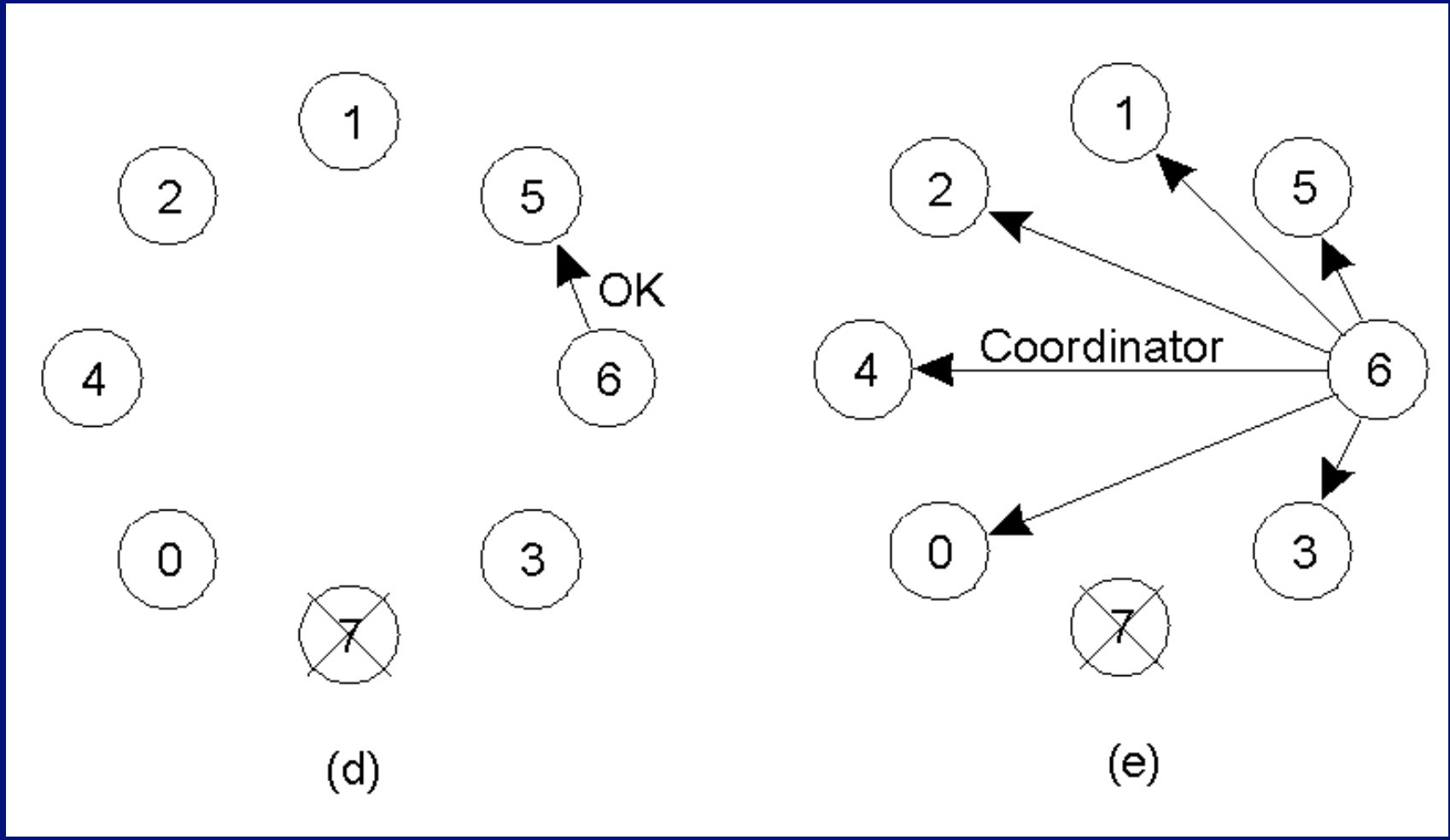
Algoritmos Eletivos

- Como eleger um coordenador?
  - É preciso identificar os processos univocamente
- Coordenador = processo com o ID mais alto
- Objetivo de um algoritmo eletivo: obter consenso entre os processos sobre quem é o novo coordenador

Algoritmo do Ditador



Algoritmo do Ditador

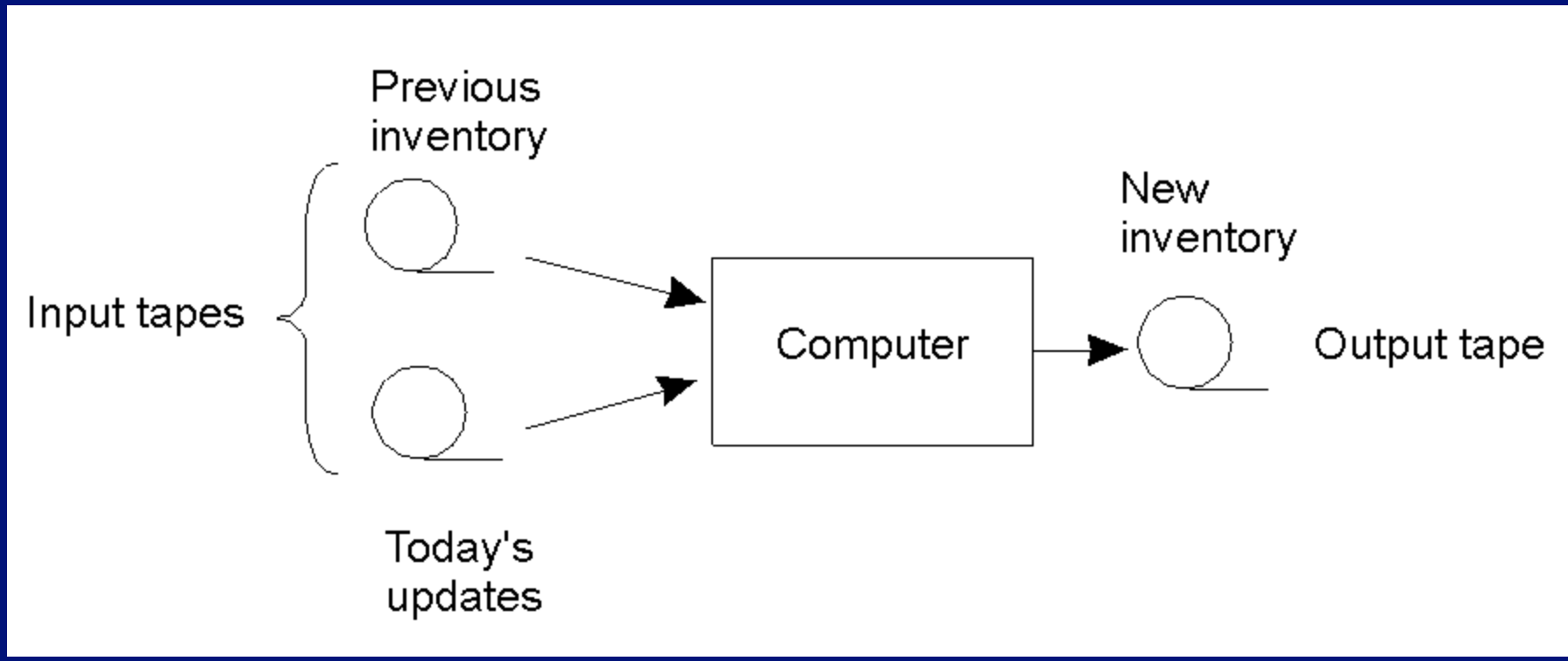


Transações Distribuídas

- **O modelo transacional**
  - **Um processo anuncia seu desejo de iniciar uma transação com outros processos**
  - **Os processos negociam entre si livremente**
  - **Quando o iniciante pede um COMMIT, todos DEVEM concordar**
  - **Se pelo menos um não concordar, a transação é abortada, e a situação volta para o estado anterior à negociação**
  - **É tudo ou nada**

Transações Distribuídas

- O modelo transacional



Primitivas Transacionais

Primitiva	Descrição
BEGIN_TRANSACTION	Marca o início de uma transação
END_TRANSACTION	Finaliza uma transação e faz tentativa de acordo entre as partes envolvidas
ABORT_TRANSACTION	Mata a transação e restaura os valores anteriores ao início da transação
READ	Leitura dos dados de um arquivo ou de qualquer outro objeto
WRITE	Escrita de dados em um arquivo ou em qualquer outro objeto

Primitivas Transacionais

- Exemplo

BEGIN_TRANSACTION

reserva Bsb – Salvador

reserva Salvador – João Pessoa

reserva João Pessoa - Natal

END_TRANSACTION

Propriedades de Transações

- **A**tomicidade: a transação é indivisível para o mundo exterior
- **C**onsistência: a transação não viola restrições do sistema
- **I**solamento: as transações concorrentes não podem interferir umas nas outras
- **D**urabilidade: uma vez que as partes envolvidas em uma transação entrem em acordo, as mudanças efetuadas passam a ser permanentes

Tipos de Transações

- Transações flat
- Transações aninhadas
- Transações distribuídas

Transações Flat

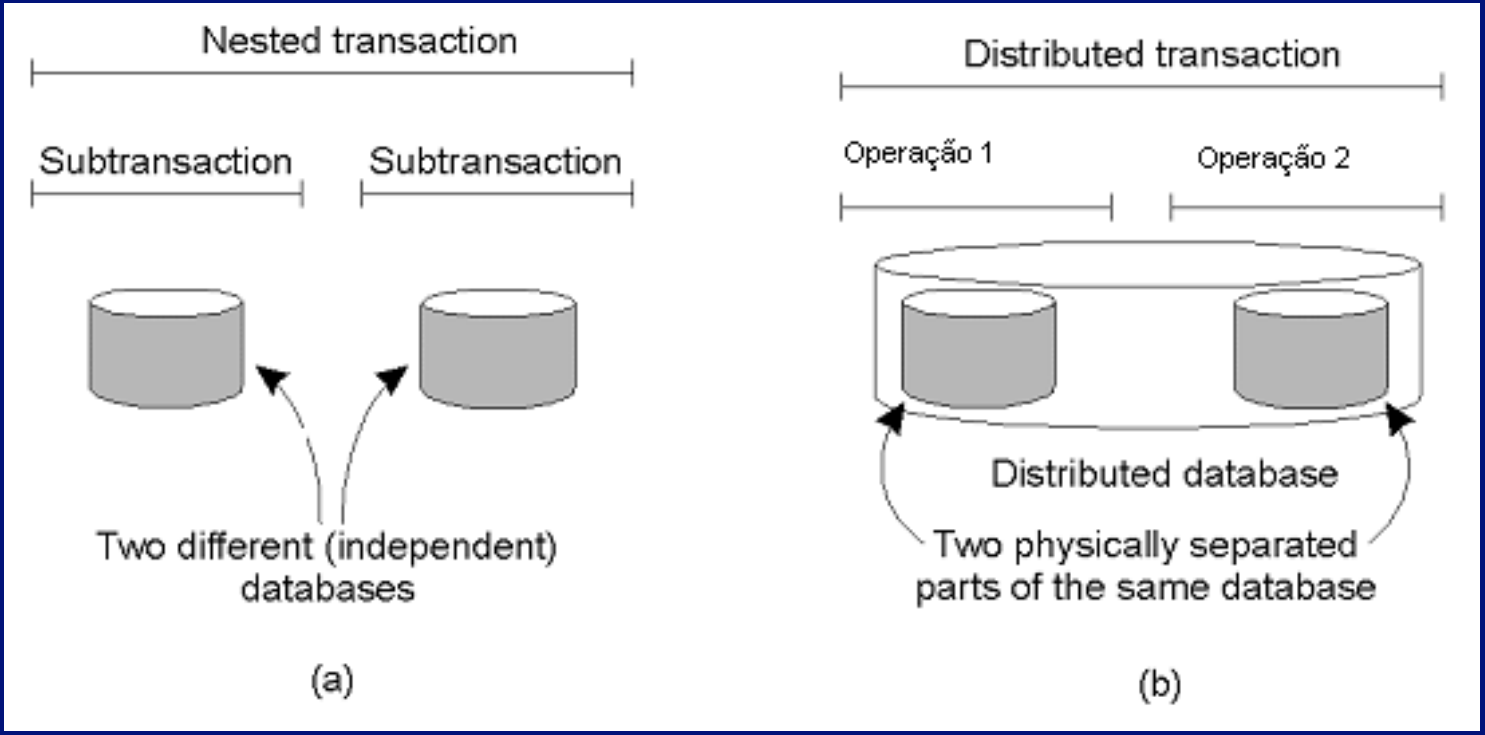
- Transação = série única de operações
- Exemplos anteriores

Transações Aninhadas

- Transação = conjunto de sub-transações
- A propriedade de durabilidade aplica-se apenas à transação de mais alto nível
- O nível de aninhamento das transações é arbitrário
  - Mas a semântica é clara
- Sub-transações podem ser executadas em paralelo, em máquinas diferentes
  - Ganhos em desempenho

Transações Distribuídas

- Transação distribuída = transação flat que opera sobre dados distribuídos

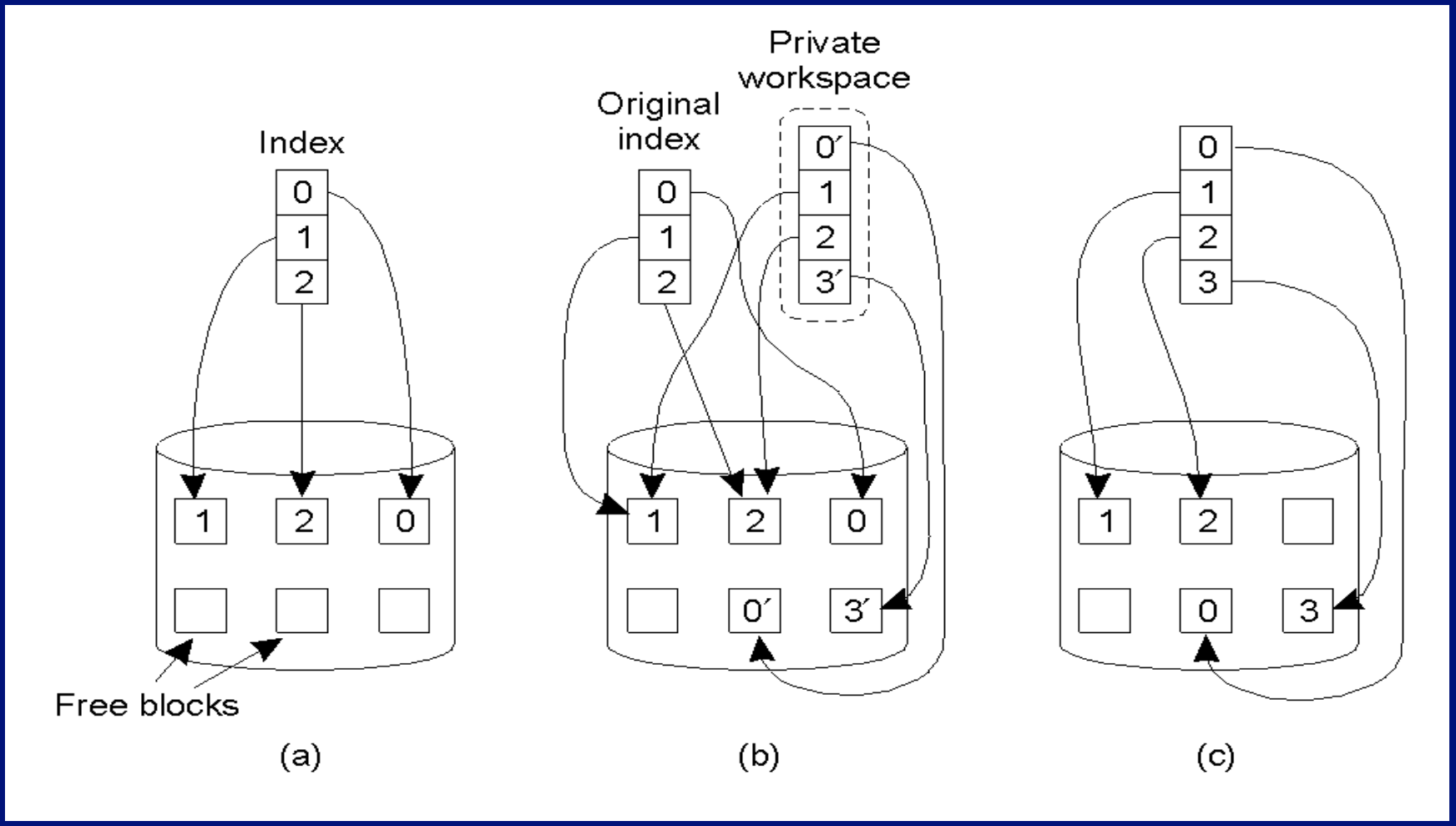


Implementação de Transações

- Área de trabalho privada
  - Cada transação trabalha dentro de uma área privada até que seja feito um COMMIT ou ABORT
    - Os objetos que a transação manipula são copiados para dentro dessa área
  - Alto custo para gerar cópias privadas
    - Arquivos de leitura não precisam ser copiados para a área privada
    - Arquivos de escrita não precisam ser copiados por inteiro

Implementação de Transações

- Área de trabalho privada



Implementação de Transações

Log & Rollback

- Modificações são realizadas nos objetos reais
- Mas antes, um log é gravado
  - Qual a transação responsável pela mudança
  - Qual objeto está sendo modificado
  - Quais os valores velhos e novos

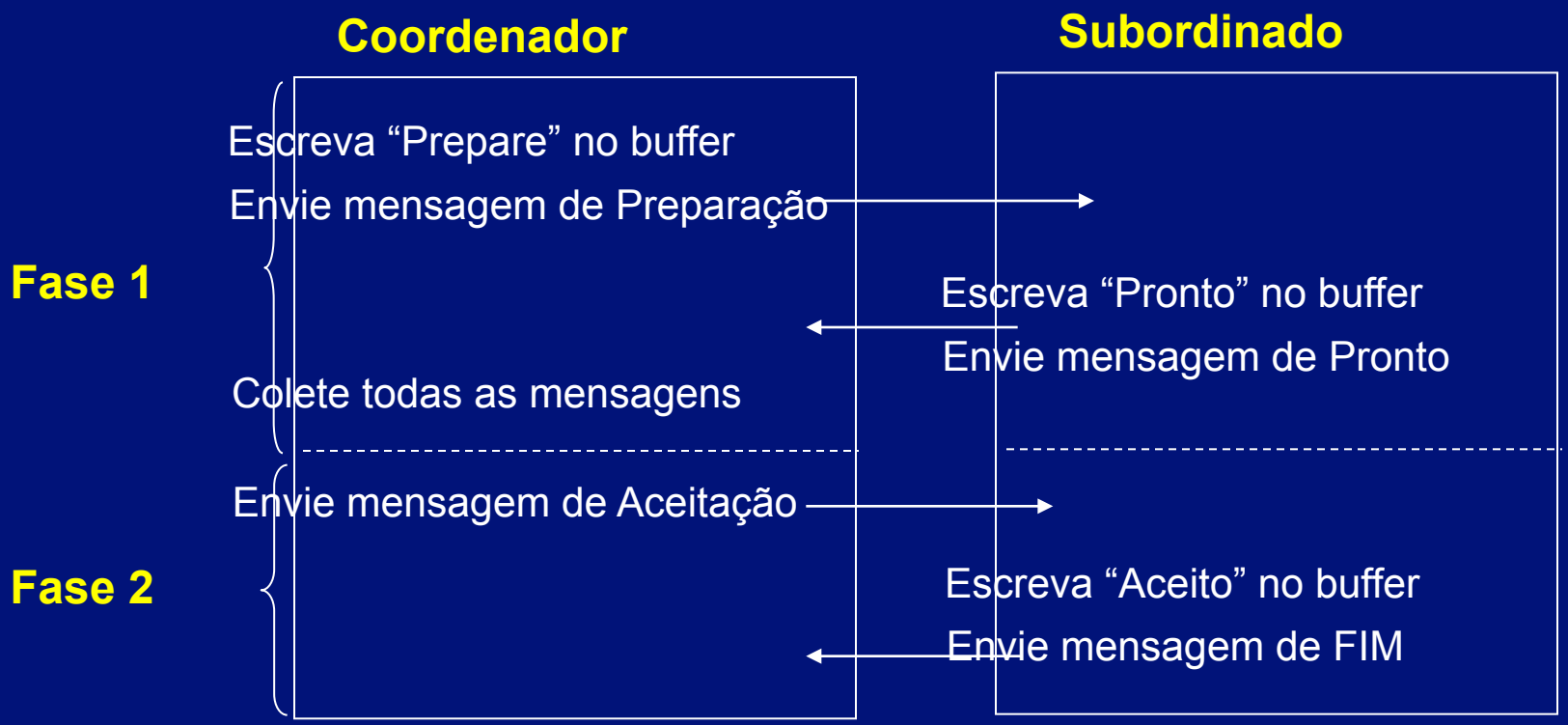
x = 0;	Log	Log	Log
y = 0;			
BEGIN_TRANSACTION;			
x = x + 1;	[x = 0 / 1]	[x = 0 / 1]	[x = 0 / 1]
y = y + 2		[y = 0/2]	[y = 0/2]
x = y * y;			[x = 1/4]
END_TRANSACTION;			
(a)	(b)	(c)	(d)

Implementação de Transações

- Log & Rollback
  - COMMIT: nenhuma modificação é necessária
  - ABORT: o log é utilizado para voltar ao estado original

Protocolo de Aceitação em Duas Fases

- **Um COMMIT é atômico**
  - **Em um sistema distribuído, a aceitação de uma transação pode requerer a cooperação de vários processos**



Controle de Concorrência

- Isolamento/Serialização

```
BEGIN_TRANSACTION
  x = 0;
  x = x + 1;
END_TRANSACTION
```

(a)

```
BEGIN_TRANSACTION
  x = 0;
  x = x + 2;
END_TRANSACTION
```

(b)

```
BEGIN_TRANSACTION
  x = 0;
  x = x + 3;
END_TRANSACTION
```

(c)

Escalonamento 1	x = 0; x = x + 1; x = 0; x = x + 2; x = 0; x = x + 3	Legal
Escalonamento 2	x = 0; x = 0; x = x + 1; x = x + 2; x = 0; x = x + 3;	Legal
Escalonamento 3	x = 0; x = 0; x = x + 1; x = 0; x = x + 2; x = x + 3;	Ilegal

Controle de Concorrência

- **Bloqueio**
  - Quando um processo precisa de um recurso (objeto qualquer), ele o obtém de forma exclusiva
    - Bloqueia o acesso ao recurso por parte dos demais processos
  - Bloqueios de leitura são compartilháveis
  - Bloqueios de escrita são exclusivos
  - A granularidade do bloqueio pode variar
    - Granularidade fina: maior paralelismo, maior complexidade, maior possibilidade de deadlocks

Controle de Concorrência

- Bloqueio em duas Fases