[MAC0426] Sistemas de Bancos de Dados
[IBI5013] Bancos de Dados para Bioinformática
Aula 22
Introdução ao Processamento de Transações

Kelly Rosa Braghetto

DCC–IME–USP

31 de maio de 2016

Sistema Monousuário × Sistema Multiusuário

Se refere ao número de usuários que podem usar o sistema simultaneamente

- **SGBD monousuário** – no máximo um usuário de cada vez pode utilizar o sistema
  - De forma geral, são restritos a computadores pessoais
- **SGBD multiusuário** – muitos usuários podem acessar o banco de dados simultaneamente
  - Isso é possível porque o SO pode executar vários programas "ao mesmo tempo", por *multiprogramação* ou *processamento paralelo*
  - Na multiprogramação, uma CPU fica se alternando entre a execução de vários processos
  - O processamento paralelo ocorre se o computador tiver várias CPUs

Transação

- É um programa em execução que forma uma **unidade lógica de processamento** em banco de dados
- **Inclui uma ou mais operações** de acesso a banco de dados
  - recuperação, inserção, remoção e alteração
- Pode estar **embutida em um programa** ou pode ser **especificada interativamente com um linguagem de consulta**, como a SQL
- Pode ser de dois tipos:
  - Transação **somente de leitura**
  - Transação **de leitura-gravação**

Modelo de banco de dados simplificado

- Para o processamento de transações, um banco de dados pode ser visto como uma coleção de **itens de dados**
- Itens de dados podem ser de diferentes **granularidades** (= tamanhos). Exemplos:
  - valor de um campo (= atributo)
  - um registro
  - um bloco do disco inteiro
- Cada item de dados tem um nome que o identifica exclusivamente no BD
- Os conceitos de processamento de transações desta aula independem de granularidade, eles se aplicam a itens de dados em geral

Modelo de banco de dados simplificado

Operações básicas de acesso ao BD que uma transação pode conter:

- **read_item(X)**: lê o item chamado X do BD e o armazena em uma variável do programa
  - Para manter a notação simples, supõe-se que a variável também se chama X
- **write_item(X)**: grava o valor da variável de programa X no item de BD chamado X

Operações básicas de acesso ao BD

Etapas da execução do comando **read_item(X)**:

1. Ache o endereço do bloco de disco que contém X
2. Copie esse bloco para um buffer na memória principal (se ele já não estiver lá)
3. Copie o item X do buffer para a variável de programa X

Etapas da execução do comando **write_item(X)**:

1. Ache o endereço do bloco de disco que contém X
2. Copie esse bloco para um buffer na memória principal (se ele já não estiver lá)
3. Copie o item X da variável X para o seu local correto no buffer
4. Armazene o bloco atualizado do buffer no disco (imediatamente ou em algum momento posterior)

Sobre o buffer de blocos em memória principal

- Quem decide quando um bloco modificado no buffer é gravado de volta no disco é **gerenciador de recuperação do SGBD** em cooperação com o SO subjacente
- O SGBD manterá em memória principal (como cache) vários buffers de dados
- Cada buffer costuma manter um bloco do disco
- Alguma política de substituição é usada para determinar quais buffers atuais devem ser substituídos quando novos blocos precisam ser acessados e não há mais buffers livres
- Antes de ser reutilizado, um buffer precisa ser gravado de volta no disco

Duas transações de exemplo

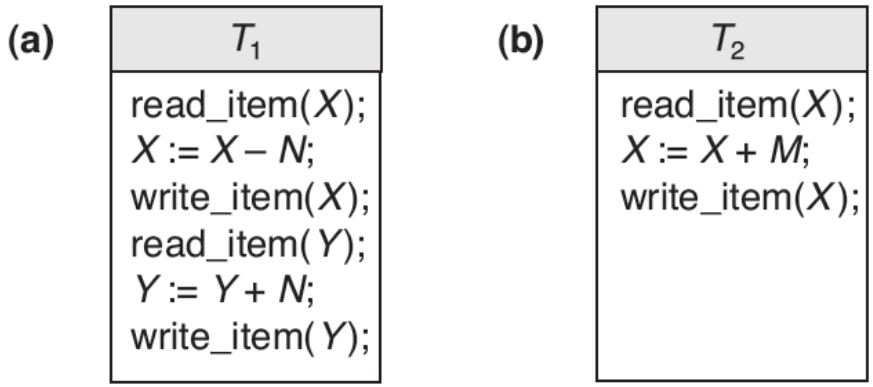


Figura 21.2
Duas transações de exemplo. (a) Transação T_1. (b) Transação T_2.

T1

- Conjunto de leitura: $\{X, Y\}$
- Conjunto de gravação: $\{X, Y\}$

T2

- Conjunto de leitura: $\{X\}$
- Conjunto de gravação: $\{X\}$

Interpretação do exemplo

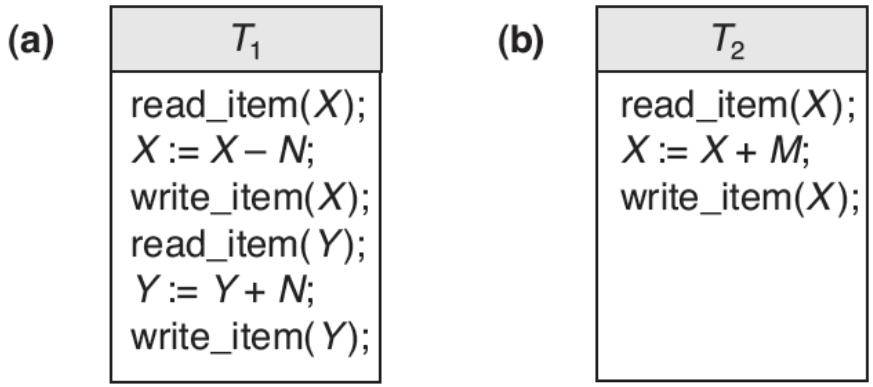


Figura 21.2

Duas transações de exemplo. (a) Transação T_1. (b) Transação T_2.

Considere um BD simplificado de reservas aéreas

- Um registro é armazenado para cada voo
- Um registro inclui, entre outras coisas, o *número de assentos reservados* no voo como um item de dado nomeado
- A transação T1 transfere N reservas de um voo cujo número de assentos reservados está no item de dados X para um outro voo cujo número de assentos reservados está no item de dados Y
- A transação T2 apenas reserva M assentos no primeiro voo (X)

Interpretação alternativa do exemplo

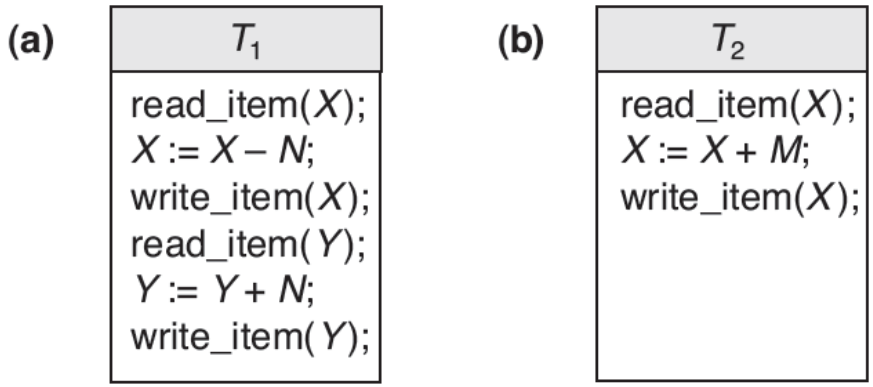


Figura 21.2

Duas transações de exemplo. (a) Transação T_1. (b) Transação T_2.

Considere um BD de dados bancários

- A transação T1 realiza uma transferência de fundos da conta X para a conta Y
- A transação T2 realiza um depósito na conta X.

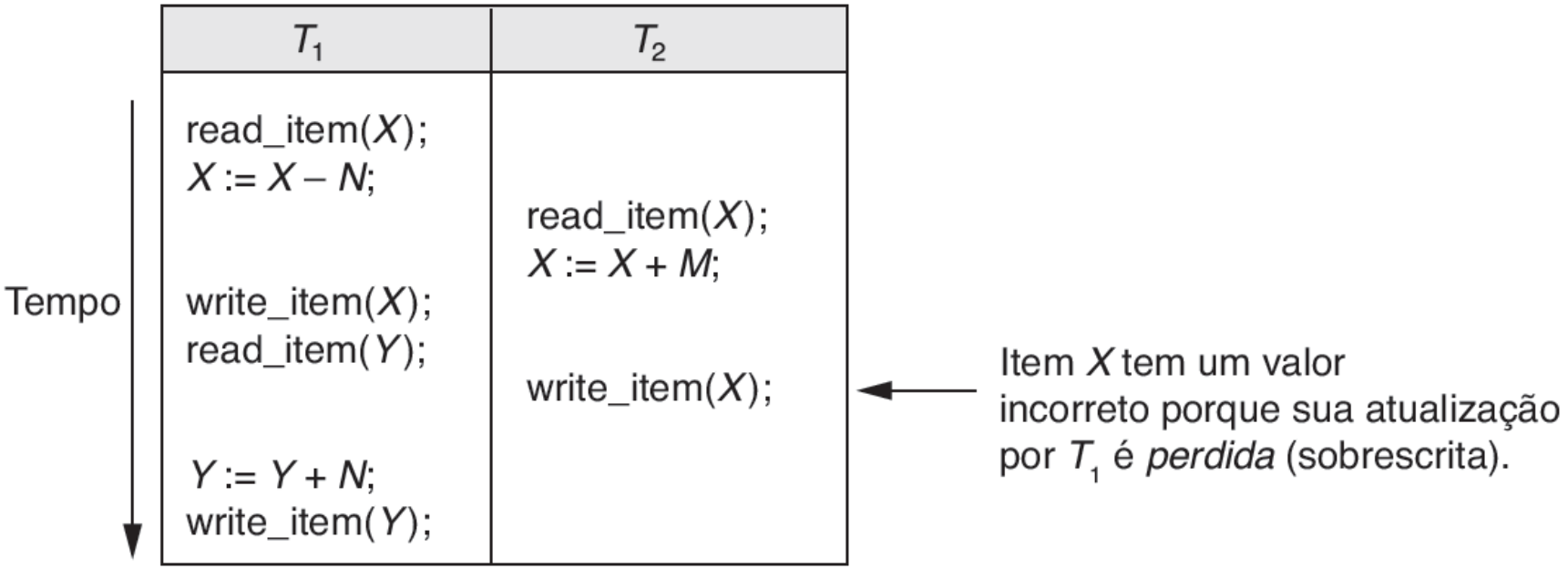
Por que controle de concorrência é necessário?

Problemas que podem ser causados por transações simultâneas:

- Atualização perdida
- Atualização temporária (ou leitura suja)
- Resumo incompleto
- Leitura não repetitiva

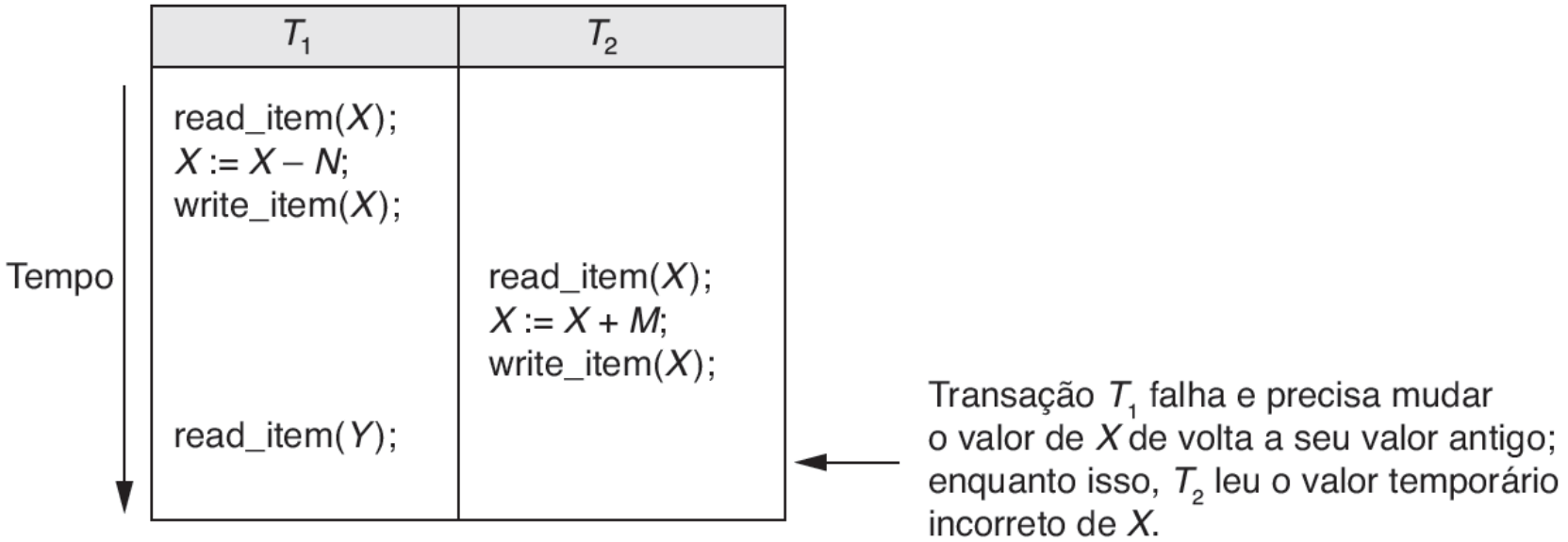
O problema da atualização perdida

- Ocorre quando a intercalação das operações de duas transações que acessam os mesmos itens do BD torna o valor de alguns itens incorreto.
- Exemplo:



O problema da atualização temporária (ou leitura suja)

- Ocorre quando um transação atualiza um item do BD e depois a transação falha por algum motivo. Nesse meio tempo, o item atualizado é acessado (lido) por outra transação, antes de ser atualizado de volta ao seu valor original.
- Exemplo:



O problema do resumo incorreto

- Se uma transação está calculando uma função de agregação sobre uma série de itens do BD, enquanto outras transações estão atualizando alguns desses itens, a função de agregação pode usar alguns valores antes que eles sejam atualizados e outros depois que foram atualizados.
- Exemplo:

T_1	T_3
	$sum := 0;$ read_item(A); $sum := sum + A;$ ⋮

read_item(X);
$X := X - N;$
write_item(X);	

	read_item(X);
	$sum := sum + X;$
	read_item(Y);
	$sum := sum + Y;$

read_item(Y);
$Y := Y + N;$
write_item(Y);	

T_3 lê X depois que N é subtraído e lê Y antes que N seja somado; um resumo errado é o resultado (defasado por N).

O problema da leitura não repetitiva

- Ocorre quando uma transação T lê o mesmo item duas vezes e o item é alterado por outra transação T' entre as duas leituras. Logo, T recebe valores diferentes para suas duas leituras do mesmo item.

Por que a recuperação é necessária?

Sempre que uma transação é submetida a um SGBD para execução, ele é responsável por garantir que:

- todas as operações na transação sejam concluídas com sucesso e seu efeito seja registrado permanentemente no BD
  - **transação confirmada** (**commited**)

ou

- que a transação não tenha qualquer efeito no BD ou em quaisquer outras transações
  - **transação abortada**

Se a transação **falhar** antes de terminar de executar todas suas operações, **as operações já executadas precisam ser desfeitas**.

Causas de falhas na execução de transações

- **Falha do computador:** erros de hardware, software ou rede
- **Falha de transação:** problema em alguma operação na transação (ex.: divisão por zero, estouro de inteiro, etc.)
- **Erros locais ou condições de exceção detectados pela transação:** exceções que podem ser programadas na própria transação para cancelá-la quando necessário
- **Imposição do controle de concorrência,** que pode decidir abortar uma transação porque ela viola seriação ou para resolver um estado de deadlock entre várias transações. Nesses casos, as transações abortadas são reiniciadas automaticamente posteriormente.
- **Falha de disco:** defeito de leitura/gravação ou falha na cabeça de leitura/gravação do disco
- **Problemas físicos e catástrofes:** falha de energia, incêndio, roubo, sabotagem...

Operações adicionais de transações

Para fins de recuperação, um SGBD precisa registrar quando cada transação começa, termina e confirma ou aborta.

O gerenciador de recuperação acompanha as seguintes operações:

- **BEGIN_TRANSACTION** – marca o início da execução da transação
- **READ** ou **WRITE** – leituras ou gravações nos itens de BD executadas nas transações
- **END_TRANSACTION** – marca o fim da execução das operações de leitura e gravação da transação. Neste ponto, pode ser necessário verificar se as mudanças feitas pela transação podem ser permanentemente aplicadas no BD (= *commited*) ou se precisarão ser abortadas

Operações adicionais de transações (continuação)

Para fins de recuperação, um SGBD precisa registrar quando cada transação começa, termina e confirma ou aborta.

O gerenciador de recuperação acompanha as seguintes operações (cont.):

- **COMMIT_TRANSACTION** – sinaliza um final bem sucedido da transação; quaisquer mudanças feitas pela transação podem ser seguramente confirmadas no BD
- **ROLLBACK** (ou **ABORT**) – sinaliza que a transação foi encerrada sem sucesso; quaisquer mudanças feitas pela transação no BD precisam ser desfeitas

Estados de uma transação

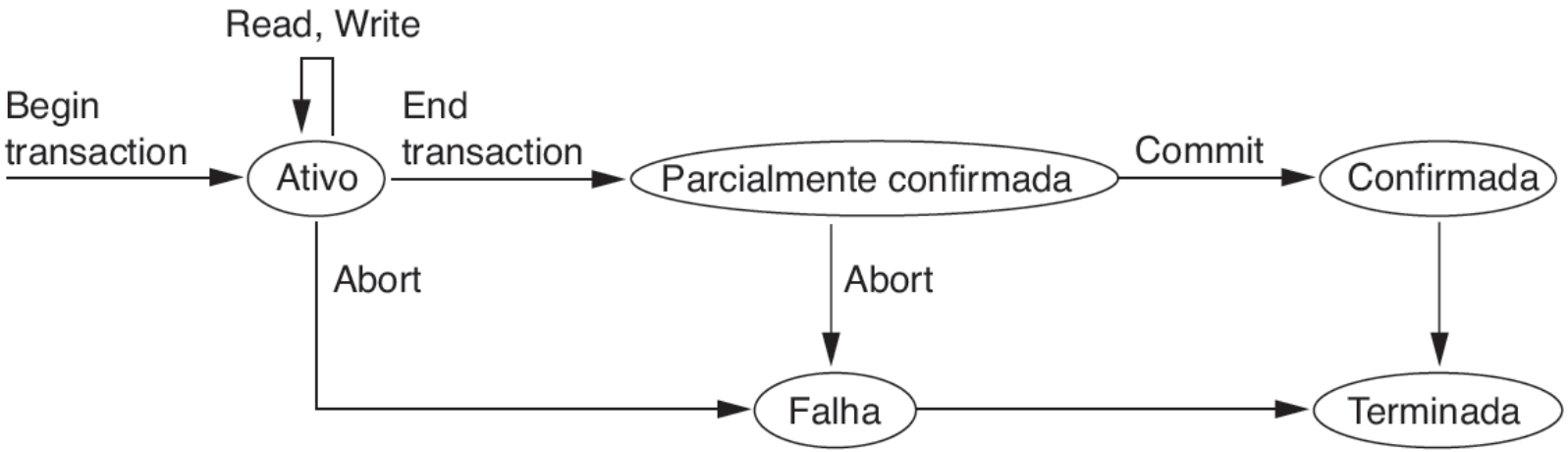


Figura 21.4

Diagrama de transição de estado ilustrando os estados para execução da transação.

O log do SGBD

- É um arquivo sequencial, apenas para inserção, mantido no disco
- **Registra as operações realizadas em transações** (tipicamente, as que afetam os valores dos itens de dados)
- As operações primeiro são registradas em **buffer de log** (em memória principal)
- Quando o buffer de log está cheio, ou quando é necessário, o buffer é anexado ao final do arquivo no disco
- O log possibilita que o SGBD se **recupere de falhas** e é usado também para auditoria

Operações registradas no log

- **[start_transaction, T]** – indica que a transação T iniciou sua execução
- **[write_item, T, X, valor_antigo, valor_novo]** – indica que a transação T mudou o valor do item X de valor_antigo para novo_valor
- **[read_item, T, X]** – indica que a transação T leu o valor do item X
- **[commit, T]** – indica que a transação T foi concluída com sucesso e que seu efeito foi registrado permanentemente no BD
- **[abort, T]** – indica que a transação T foi abortada

O log do SGBD

- O log pode ser usados pelos **protocolos de recuperação** de diferentes formas
- Como o log contém o registro de cada WRITE, é possível **desfazer** o efeito das operações de uma transação
- Também pode ser necessário **refazer** operações quando, por exemplo, uma transação tiver suas operações de escrita registradas no log mas houver uma falha antes que o SGBD esteja certo de que os novos valores tenham sido gravados no BD em disco

Ponto de Confirmação de uma Transação

Uma transação T está

- em seu **ponto de confirmação** quando **todas suas operações de acesso ao BD** tiverem sido **executadas com sucesso** e o **efeito** delas no BD tiverem sido **registradas no log**
- **confirmada** depois de passar pelo ponto de confirmação e ter seu efeito registrado permanentemente no BD (no disco)
  - Grava-se um registro de confirmação **[commit, T]** no log

Em caso de falha, pode-se buscar no log todas as transações T que gravaram um registro **[start_transaction, T]**, mas que ainda não gravaram um **[commit, T]**. Essas transações podem ter de ser **descartadas**.

Propriedades (ACID) das transações

Propriedades que devem ser impostas às transações pelos métodos de controle de concorrência e recuperação do SGBD:

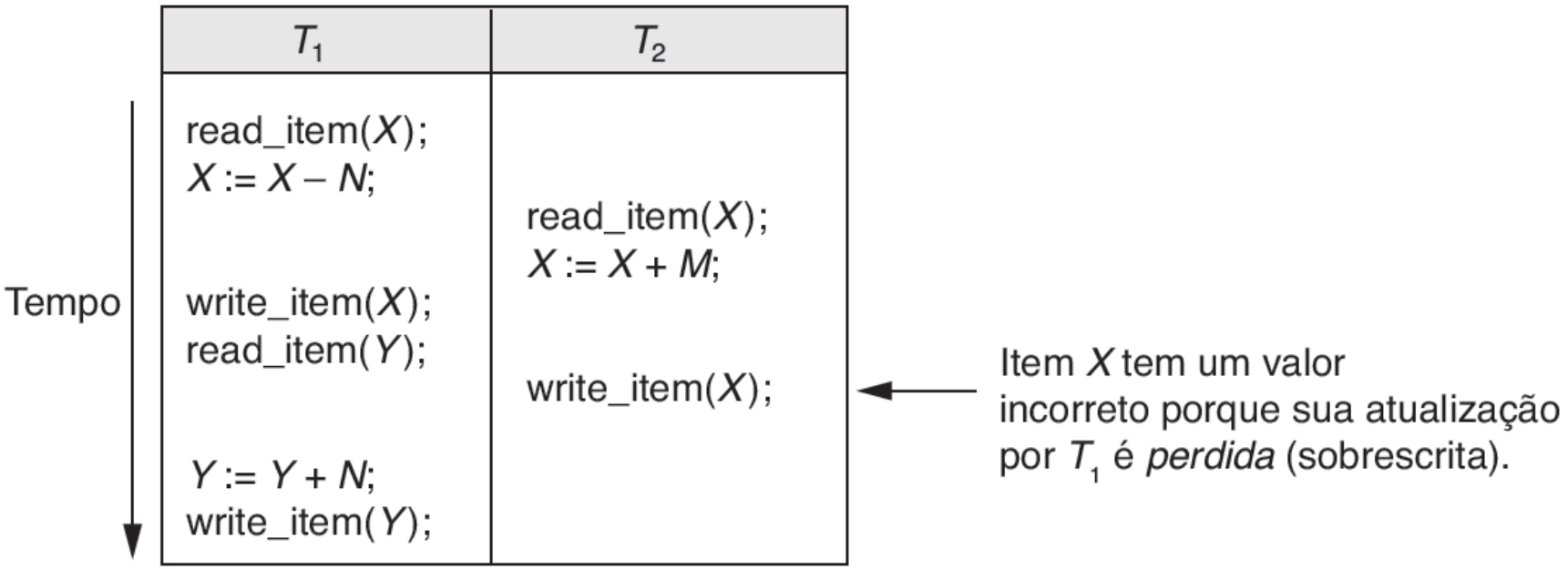
- **Atomicidade** – uma transação deve ser realizada em sua totalidade ou não ser realizada de forma alguma
- **(Preservação da) Consistência** – uma transação deve preservar consistência, levando o BD de um estado consistente para outro, se for completamente executada do início ao fim e sem interferência de outras transações
- **Isolamento** – a execução de uma transação não deve ser interferida por quaisquer outras transações que ocorrem simultaneamente
- **Durabilidade (ou Permanência)** – as mudanças applicadas ao BD pela transação confirmada precisam persistir no BD (não podem ser perdidas por causa de falhas)

Escalonamentos de transações

- Um **escalonamento** S de n transações $T_1, T2, \ldots, T_n$ é uma ordenação das operações dessas transações
- As operações das diferentes transações podem ser intercaladas em S
  - Mas, para cada T_i que participa de S, as operações de T_i em S precisam aparecer na mesma ordem em que ocorrem em T_i
- S define uma *ordenação total* sobre as operações; mas teoricamente é possível lidar com escalonamentos que formam *ordens parciais*
- Para recuperação e controle de concorrência, o interesse está nas operações read_item (r), write_item (w), commit (c) e abort (a)

Notação abreviada para descrever escalonamentos

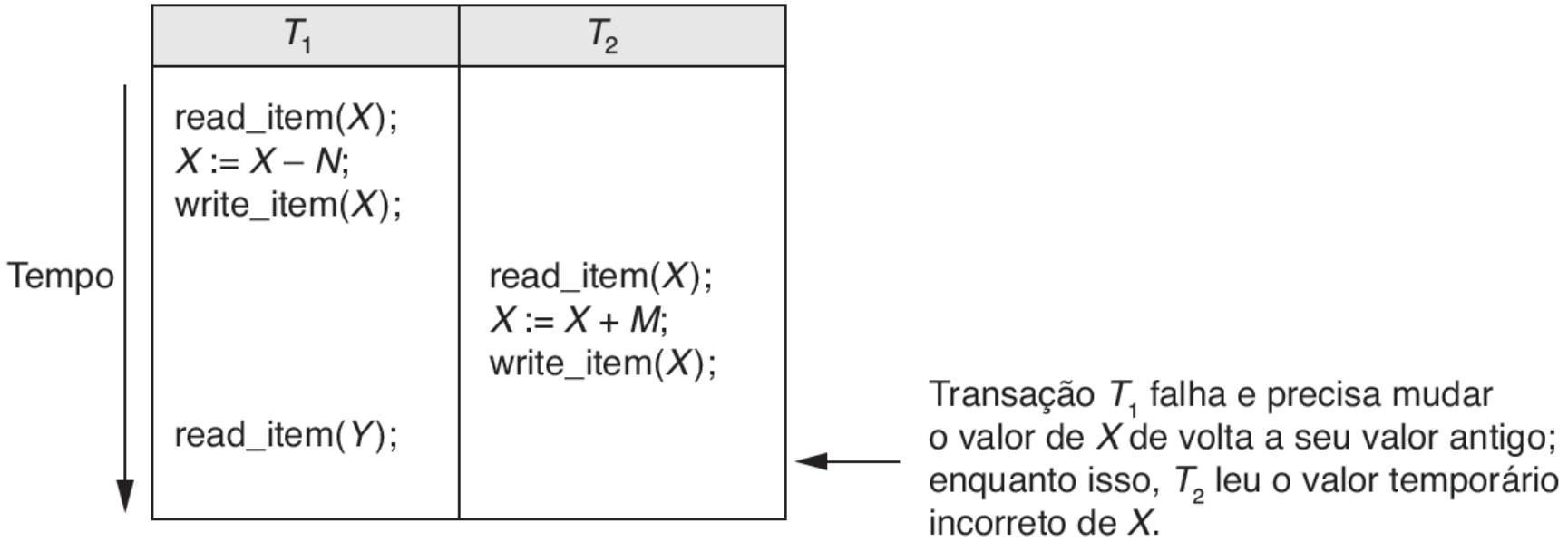
Exemplo 1:



- $S_a : r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$

Notação abreviada para descrever escalonamentos

Exemplo 2:



- $S_b : r_1(X); w_1(X); r_2(X); w_2(X); r_1(Y); a_1;$

Aqui, estamos considerando que T_1 foi cancelada após a operação read_item(Y)

Conflito de operações

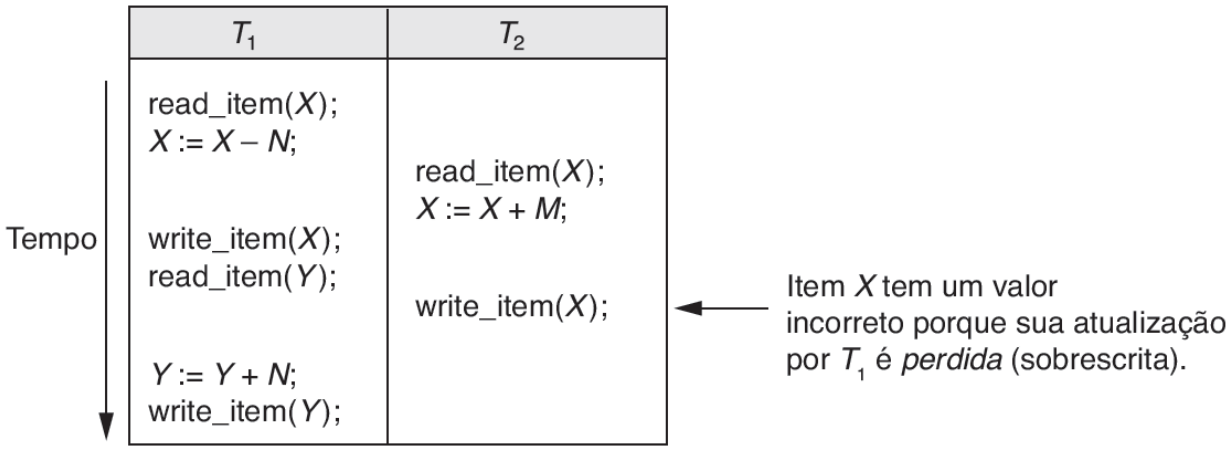
Duas operações em um escalonamento estão em **conflito** se satisfizerem as **três condições** a seguir:

1. Elas pertencem a transações diferentes
2. Elas acessam o mesmo item X
3. Pelo menos uma das operações é um write_item(X)

Intuitivamente: duas operações estão em conflito se a mudança de sua ordem pode gerar um resultado diferente.

Conflito de operações

Exemplo:



- $S_a : r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$

Em S_a, estão em conflito os pares:
$r_1(X)$ e $w_2(X)$; $r_2(X)$ e $w_1(X)$; e $w_1(X)$ e $w_2(X)$

Os dois primeiros pares são conflito de **leitura-gravação**; já o último é um conflito de **gravação-gravação**.

Facilidade de recuperação de escalonamentos

- Para alguns escalonamentos, é fácil de se recuperar de falhas (de transação e sistema)
- Para outros, a recuperação pode ser bem complicada ou até mesmo impossível
- Vamos discutir três tipos de escalonamentos:
  - os **recuperáveis** > os que **evitam *rollback* em cascata** > **os estritos**

Escalonamento recuperável

- Um escalonamento S é **recuperável** se nenhuma transação T em S for confirmada até que todas as transações T' que tiverem gravado algum item X que T lê sejam confirmadas
- Isso garante que, quando uma transação T é confirmada, nunca deve ser necessário cancelar T
  - ou seja, a *propriedade de durabilidade das transações não é violada*

Escalonamento recuperável

Exemplo 1:

$$S'_a : r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); c_2; w_1(Y); c_1;$$

- S'_a é recuperável porque nele T_2 não lê nada que T_1 escreveu antes. E T_1 também não lê nada que T_2 tenha escrito antes.

  Mas S'_a sofre do problema da atualização, porque T2 sobrescreve o valor de X gravado por T1 (que veremos como tratar depois!).

Escalonamento **não** recuperável

Exemplo 2:

$$S_c : r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); c_2; a_1;$$

- S_c não é recuperável porque nele T_2 lê X de T_1, mas T_2 confirma antes que T_1 confirme. Se T_1 abortar depois de c_2, então o valor de X que T_2 leu não é mais válido e T_2 precisaria ser **abortado depois de já ter sido confirmado**.

  S_d e S_e mostra dois escalonamentos correspondentes recuperáveis:

$$S_d : r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); c_1; c_2;$$

$$S_e : r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); a_1; a_2;$$

Rollback em cascata (= propagação do cancelamento)

- Um ***rollback* em cascata** ocorre quando uma transação *não confirmada* foi cancelada porque leu um item de uma transação que falhou
  - Observe que isso pode ocorrer mesmo em escalonamentos recuperáveis
  - Pode ser uma operação bem demorada
- Um escalonamento evita *rollback* em cascata se cada transação nele apenas ler itens que foram gravados por transações já confirmadas
- Um escalonamento que evita *rollback* em cascata sempre é também um escalonamento recuperável

Rollback em cascata (= propagação do cancelamento)

Exemplos

- S_d : $r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); c_1; c_2;$
  não evita *rollback* em cascata por causa de $r_2(X)$, que ocorre antes de c_1

- S'_d : $r_1(X); w_1(X); r_1(Y); c_1; r_2(X); w_2(X); c_2;$
  evita *rollback* em cascata (mas adia a execução de T_2!)

Escalonamento estrito

- Em um **escalonamento estrito**, as transações *não podem ler nem gravar* um item X até que a última transação que gravou X tenha sido confirmada ou cancelada.
- Esse tipo de escalonamento simplifica o processo de recuperação, já que nele, desfazer um operação write_item(X) de uma transação abortada requer apenas a restauração do valor anterior de X
  - esse procedimento de restauração simples nem sempre pode ser aplicado para escalonamentos recuperáveis ou para os que evitam *rollback* em cascata
- Um escalonamento estrito também evita *rollback* em cascata

Escalonamento estrito

Exemplo

- $S_f : w_1(X, 5); w_2(X, 8); a_1$
  Esse escalonamento não é estrito porque permite que T_2 grave X embora T_1 (que gravou X por último) ainda não tenha sido confirmada ou cancelada.

  Suponha que $X = 9$ antes da execução de S_f (ou seja, o valor antigo de X armazenado no log junto com a operação $w_1(X, 5)$ é 9). Quando T_1 é cancelada em S_f, se o procedimento de recuperação restaurar o valor antigo de X associado à escrita feita em X por T_1, acabará recuperando o valor 9 para X, embora o valor de X já tivesse sido alterado para 8 pela transação T_2, levando a resultados incorretos.

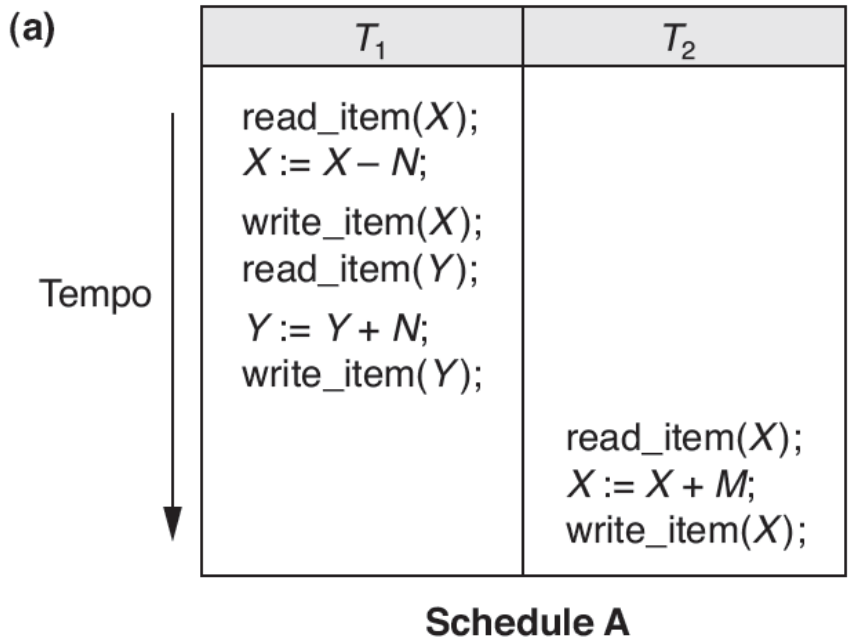
Escalonamentos seriais

- Quando **não permitimos intercalação** da execução das operações **das transações concorrentes**, **sempre temos um escalonamento correto** (propriedade de preservação da consistência).
  Há sempre dois desse tipo – os chamados **escalonamentos seriais**:
  - Executar todas as operações da transação T_1 seguidas de todas as operações da transação T_2
  - Executar todas as operações da transação T_2 seguidas de todas as operações da transação T_1
- Com intercalação de operações, há muitas ordens de execução possíveis

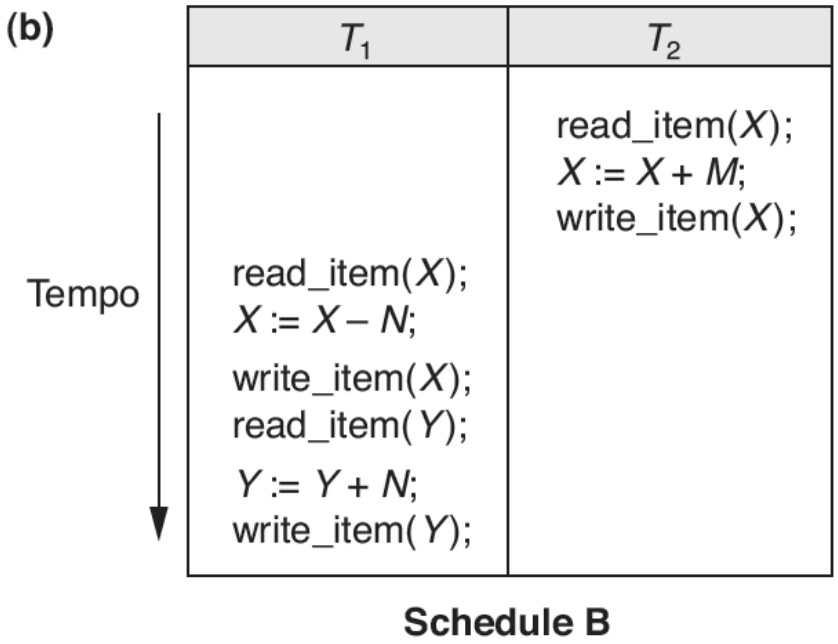
Escalonamento serial

- Em um escalonamento serial, somente uma transação de cada vez está ativa
- O commit ou o abort da transação ativa inicia a execução da próxima

Exemplos



Schedule A



Schedule B

Escalonamentos seriais

Problemas

- Desperdício de CPU (uma transação pode ficar "bloqueando" a CPU enquanto espera E/S de dados)
- Se alguma transação T for muito longa, as demais transações poderão ter que esperar que T termine todas as suas operações antes de poderem começar a executar

Escalonamentos seriais são considerados inviáveis na prática. :(

Escalonamentos (não) equivalentes a seriais

Exemplo

- O escalonamento não-serial C **não produz o mesmo resultado** que o serial A
  Teste-os para os seguintes valores iniciais:
  X = 90, Y = 90, N = 3 e M = 2

a)

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);
	read_item(X);
	$X := X + M$;
	write_item(X);

Tempo

Schedule A

c)

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);
	read_item(X);
	$X := X + M$;
	write_item(X);

Tempo

Schedule C

Escalonamentos equivalentes a seriais

Exemplo

- O escalonamento não-serial D **produz o mesmo resultado** que o serial A
  Teste-os para os seguintes valores iniciais:
  X = 90, Y = 90, N = 3 e M = 2

a)

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);	

	read_item(X);
	$X := X + M$;
	write_item(X);

Tempo

Schedule A

T_1	T_2

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$X := X + M$;
	write_item(X);

read_item(Y);
$Y := Y + N$;
write_item(Y);	

Tempo

Schedule D

Escalonamento seriável

- Um escalonamento S de n transação é **seriável** *se for* ***equivalente*** *a algum dos* $n!$ *possíveis escalonamentos seriais* das n transações
  - Mas quando dois escalonamentos podem ser considerados **equivalentes**?
- Se um escalonamento é seriável, então podemos dizer que ele é **correto**

Equivalência de escalonamentos

Nas basta ser equivalente apenas no resultado

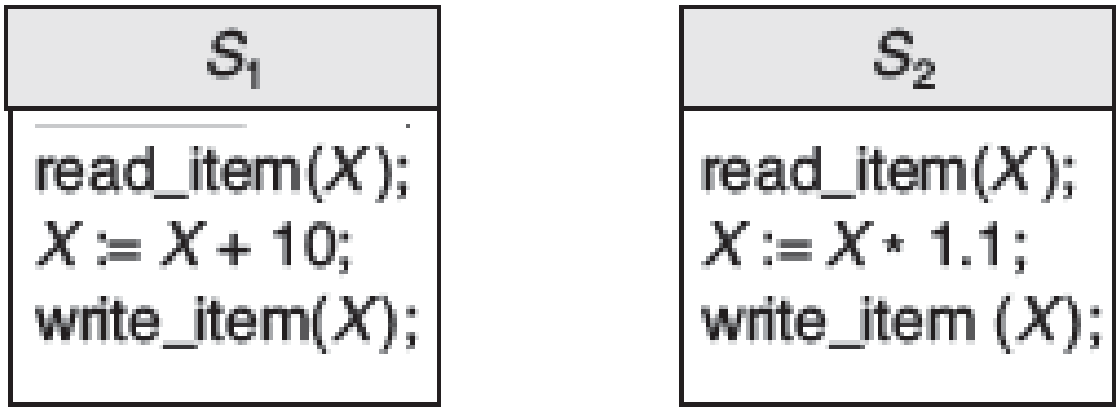


Figura 21.6

Dois schedules que são equivalentes no resultado para o valor inicial de $X = 100$, mas não são equivalentes no resultado em geral.

Equivalência de escalonamentos

Técnica mais segura para determinar equivalência:

- Não fazer suposições sobre os tipo de operações nas transações
- **Para dois escalonamentos serem equivalentes, as operações aplicadas a cada item de dado afetado devem ser aplicadas na mesma ordem nos dois escalonamentos**
- Duas definições costumam ser usadas:
  - **equivalência de conflito** (mais usado!)
  - **equivalência de visão**

Equivalência de Conflito

Definição

- Dois escalonamentos são **equivalentes em conflito** se a ordem de quaisquer *duas operações em conflito* for a mesma nos dois escalonamentos.

Escalonamento seriável de conflito

- Um escalonamento S é **seriável de conflito** se ele for equivalente em conflito a algum escalonamento serial S'
- Nesse caso, é possível reordenar *as operações não em conflito* em S até formar o escalonamento serial S'

Seriável de Conflito

Exemplo

- O escalonamento D **é seriável de conflito** porque ele é equivalente (em conflito) ao escalonamento A.

a)

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	
	read_item(X);
	$X := X + M$;
	write_item(X);

Tempo

Schedule A

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

Tempo

Schedule D

E como A é serial, então D é **seriável**.

Seriável de Conflito

Exemplo

- O escalonamento C **não é seriável de conflito** porque ele não é equivalente (em conflito) ao escalonamento A.
  Em A aparece o conflito $w_1(X), r_2(X)$, enquanto que em C esse conflito aparece na ordem inversa $r_2(X), w_1(X)$.

a)

T_1	T_2
read_item(X); $X := X - N$;	
write_item(X); read_item(Y);	
$Y := Y + N$; write_item(Y);	

	read_item(X);
	$X := X + M$;
	write_item(X);

Tempo

Schedule A

c)

T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;
write_item(X); read_item(Y);	
	write_item(X);
$Y := Y + N$; write_item(Y);	

Tempo

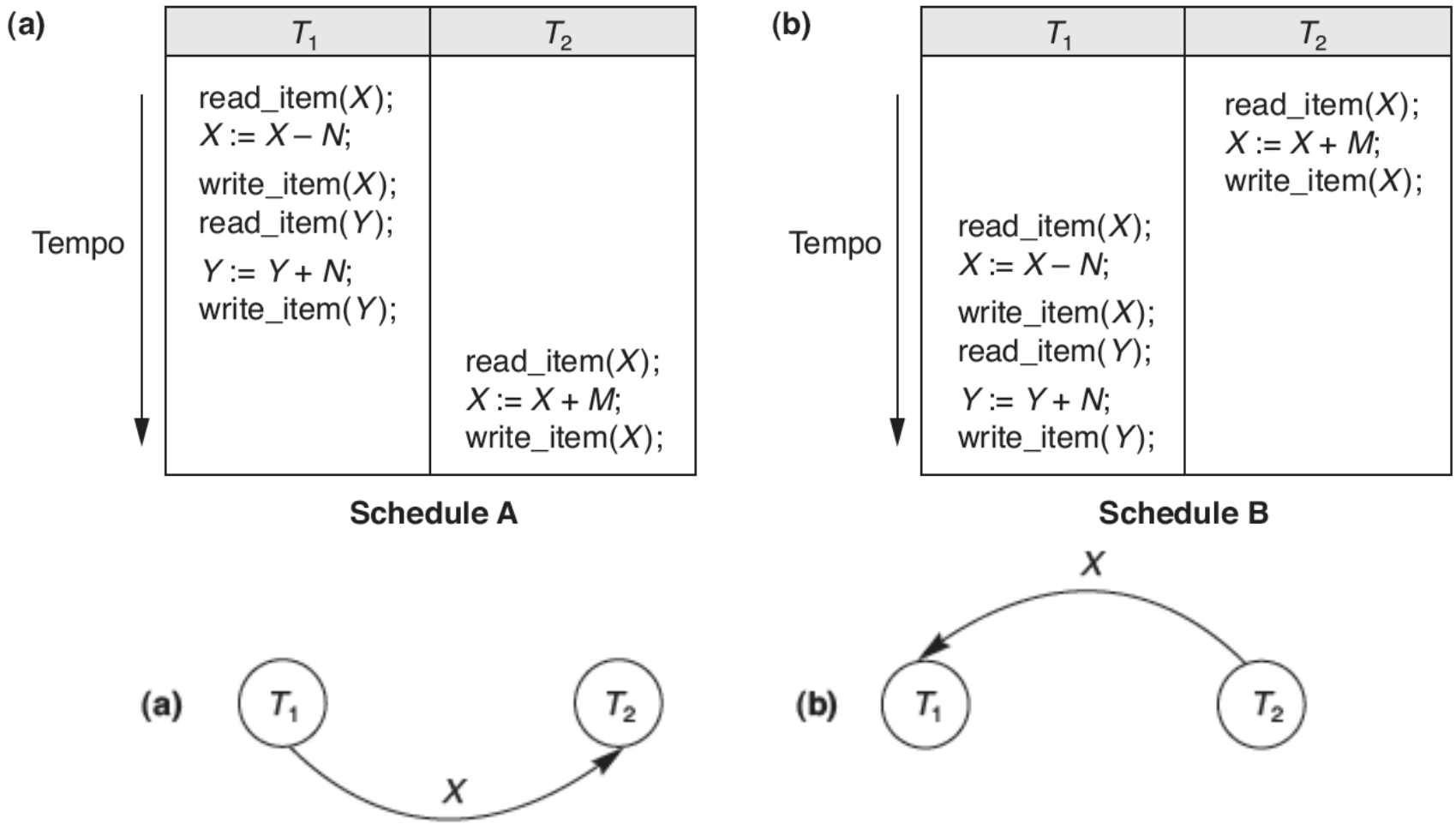
Schedule C

Testando a seriação de conflito por meio de um grafo de precedência

Algoritmo

1. Para cada transação T_i no escalonamento S, crie um nó com rótulo T_i no grafo
2. Para cada caso em S que T_j executa um read_item(X) depois de T_i executar um write_item(X), crie uma aresta no $(T_i \rightarrow T_j)$ no grafo
3. Para cada caso em S que T_j executa um write_item(X) depois de T_i executar um read_item(X), crie uma aresta no $(T_i \rightarrow T_j)$ no grafo
4. Para cada caso em S que T_j executa um write_item(X) depois de T_i executar um write_item(X), crie uma aresta no $(T_i \rightarrow T_j)$ no grafo
5. S é seriável se, e somente se, o grafo de precedência não tiver ciclos

Grafo de precedência de escalonamentos seriais



Grafo de precedência de escalonamento (D) seriável

a)

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	
	read_item(X);
	$X := X + M$;
	write_item(X);

Tempo

Schedule A

T_1	T_2
read_item(X);	
$X := X - N$;	
write_item(X);	
	read_item(X);
	$X := X + M$;
	write_item(X);
read_item(Y);	
$Y := Y + N$;	
write_item(Y);	

Tempo

Schedule D

(d)

Grafo de precedência de escalonamento (C) não seriável

a)

T_1	T_2
read_item(X); $X := X - N$;	
write_item(X); read_item(Y);	
$Y := Y + N$; write_item(Y);	

	read_item(X);
	$X := X + M$;
	write_item(X);

Tempo

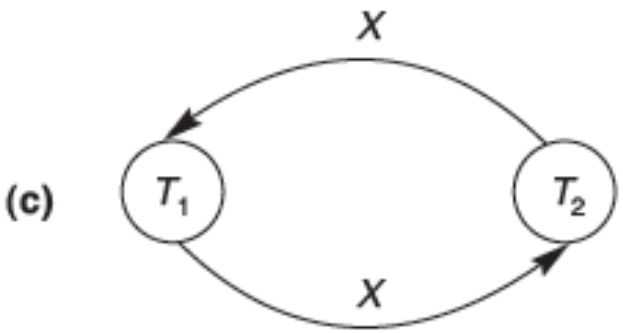
Schedule A

c)

T_1	T_2
read_item(X); $X := X - N$;	
	read_item(X); $X := X + M$;
write_item(X); read_item(Y);	
	write_item(X);
$Y := Y + N$; write_item(Y);	

Tempo

Schedule C



Outro exemplo de teste de seriação (com 3 transações)

(a)

Transação T_1
read_item(X);
write_item(X);
read_item(Y);
write_item(Y);

Transação T_2
read_item(Z);
read_item(Y);
write_item(Y);
read_item(X);
write_item(X);

Transação T_3
read_item(Y);
read_item(Z);
write_item(Y);
write_item(Z);

Escalonamento não seriável e seu grafo de precedência

(b)

Tempo

Transação T_1	Transação T_2	Transação T_3
	read_item(Z);	
	read_item(Y);	
	write_item(Y);	
		read_item(Y);
read_item(X);		read_item(Z);
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(X);	
read_item(Y);		
write_item(Y);		
	write_item(X);	

Schedule E

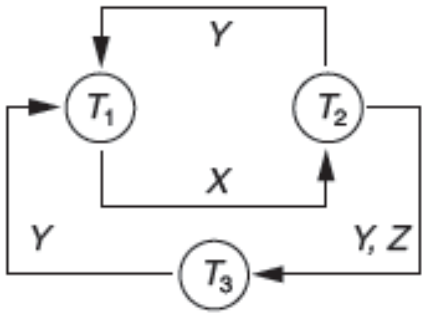
Escalonamento não seriável e seu grafo de precedência

(b)

Transação T_1	Transação T_2	Transação T_3
	read_item(Z);	
	read_item(Y);	
	write_item(Y);	
		read_item(Y);
		read_item(Z);
Tempo ↓		
read_item(X);		
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(X);	
read_item(Y);		
write_item(Y);		
	write_item(X);	

Schedule E

(d)



Schedules seriais equivalentes

Nenhum

Motivo

Ciclo $X(T_1 \rightarrow T_2), Y(T_2 \rightarrow T_1)$

Ciclo $X(T_1 \rightarrow T_2), YZ(T_2 \rightarrow T_3), Y(T_3 \rightarrow T_1)$

Escalonamento seriável e seu grafo de precedência

(c)

Tempo

Transação T_1	Transação T_2	Transação T_3
		read_item(Y);
		read_item(Z);
read_item(X);		
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(Z);	
read_item(Y);		
write_item(Y);		
	read_item(Y);	
	write_item(Y);	
	read_item(X);	
	write_item(X);	

Schedule F

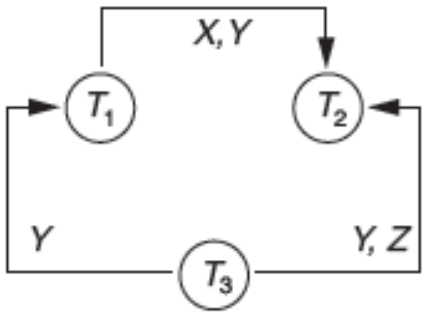
Escalonamento seriável e seu grafo de precedência

(c)

Transação T_1	Transação T_2	Transação T_3
		read_item(Y);
		read_item(Z);
read_item(X);		
write_item(X);		
		write_item(Y);
Tempo		write_item(Z);
	read_item(Z);	
read_item(Y);		
write_item(Y);		
	read_item(Y);	
	write_item(Y);	
	read_item(X);	
	write_item(X);	

Schedule F

(e)

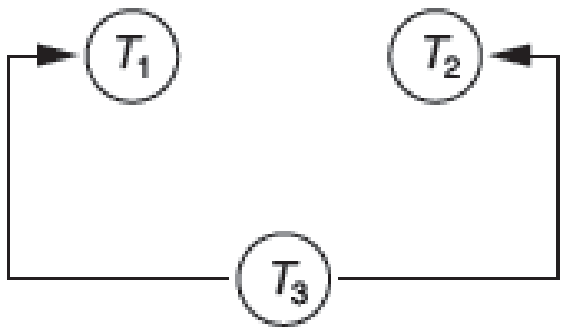


Schedules seriais equivalentes

$T_3 \rightarrow T_1 \rightarrow T_2$

Um exemplo de grafo de precedência com 2 escalonamentos seriais equivalentes

(f)



Schedules seriais equivalentes

$T_3 \rightarrow T_1 \rightarrow T_2$

$T_3 \rightarrow T_2 \rightarrow T_1$

O uso da seriação no controle de concorrência

- Dizer que um escalonamento S é equivalente (em conflito) a um escalonamento serial equivale a dizer que S é correto
- Um escalonamento serial é ineficiente, pois nenhuma intercalação de operações de transações diferentes é permitida. Consequências:
  - Baixo uso de CPU quando uma transação aguarda E/S do disco ou que outra transação termine
  - Atraso no processamento
- Um escalonamento seriável possibilita os benefícios de execução concorrente, mas sem perder correção

O uso da seriação no controle de concorrência

- Na prática, é difícil testar a seriação de um escalonamento antes de sua execução
  - Transações concorrentes costumam ser executadas como processos pelo SO; é o escalonador do SO quem acaba determinando a ordem de execução das operações
- Alternativa: executar as transações à vontade, e depois testar o escalonamento executado
  - Problema: caso o escalonamento testado for não seriável, será preciso cancelar o efeito dele no BD $\Rightarrow$ impraticável
- **Técnica usada nos SGBDs comerciais:** projetar conjunto de regras (**protocolos**) que, quando seguidos por **toda transação** (ou impostos pelo subsistema de controle de concorrência), garantem a seriação de qualquer escalonamento

Outros tipos de equivalência de escalonamentos

Equivalência de visão

- **Ideia:** desde que cada operação de leitura de uma transação leia o resultado da mesma operação de gravação nos dois escalonamentos, as operações de gravação de cada transação devem produzir os mesmos resultados.
- É menos restritiva que a equivalência por conflito
- Qualquer escalonamento seriável de conflito também é um escalonamento seriável de visão (ou seja, equivalente de visão a um escalonamento serial)
  - O contrário não é sempre verdade!

Outros tipos de equivalência de escalonamentos

Transações de débito-crédito

- Algumas aplicações podem produzir escalonamentos corretos sob condições bem menos rigorosas que as da seriação de conflito ou de visão
- **Transações de débito-crédito** são um exemplo disso:

  $T_1 : r_1(X); X = X - 10; w_1(X); r_1(Y); Y = Y + 10; w_1(Y);$
  $T_2 : r_2(Y); Y = Y - 20; w_2(Y); r_2(X); X = X + 20; w_2(X);$

  O escalonamento S a seguir não é seriável, mas considerando a semântica das operações realizadas em T_1 e T_2 podemos dizer que ele é correto:

  $S : r_1(X); w_1(X); r_2(Y); w_2(Y); r_1(Y); w_1(Y); r_2(X); w_2(X);$

Transações na SQL

Visão (bem!) geral

- Conceitualmente, uma transação em SQL é igual às demais transações que discutimos nesta aula
- Uma única instrução em SQL é sempre considerada uma transação (mesmo que ela afete múltiplos itens de dados)
  - ou ela é executada completamente ser erro, ou ela falha e deixa o BD inalterado
- Não existe na SQL padrão uma instrução "begin transaction" explícita
  - o início é feito implicitamente, quando instruções começam a ser executadas
  - mas o fim é explícito, por meio dos comandos `COMMIT` ou `ROLLBACK`

Transações na SQL

É possível configurar (ou caracterizar) uma transação em SQL, indicando:

- o **modo de acesso**:
  - `READ ONLY`
  - `READ WRITE`
- o **nível de isolamento**, com a instrução `ISOLATION LEVEL`:
  - `READ UNCOMMITED`
  - `READ COMMITED`
  - `REPEATABLE READ`
  - `SERIALIZABLE` (mas que pode não ter relação com a ideia de seriação que vimos anteriormente)

Transações na SQL

Possíveis violações por nível de isolamento

Nível de Isolamento	Leitura Suja	Leitura não repetitiva	Fantasma
READ UNCOMMITED	Sim	Sim	Sim
READ COMMITED	Não	Sim	Sim
REPEATABLE READ	Não	Não	Sim
SERIALIZABLE	Não	Não	Não

Transações em SQL

Exemplo

```
 EXEC SQL whenever sqlerror go to UNDO;
 EXEC SQL SET TRANSACTION
              READ WRITE
              ISOLATION LEVEL SERIALIZABLE;
 EXEC SQL INSERT
             INTO EMPLOYEE (FNAME, LNAME, SSN, DNO, SALARY)
             VALUES ('Robert','Smith','991004321',2,35000);
EXEC SQL UPDATE EMPLOYEE
             SET SALARY = SALARY * 1.1
             WHERE DNO = 2;
EXEC SQL COMMIT;
             GOTO  THE_END;
 UNDO: EXEC SQL ROLLBACK;
```

Referências Bibliográficas

- *Sistemas de Bancos de Dados* (6ª edição), Elmasri e Navathe. Pearson, 2010. – Capítulo 21
- *Database Systems – the complete book* (1ª edição), Garcia-Molina, Ullman e Widom. Prentice Hall. – Capítulo 18