

MC536



Processamento de  
Transações

# Sumário

---

- Noções básicas
- Propriedades
- Estados
- Mecanismos no processamento de transações
  - Recuperação de falhas
  - Controle de concorrência

# Sumário

---

- **Noções básicas**
- Propriedades
- Estados
- Mecanismos no processamento de transações
  - Recuperação de falhas
  - Controle de concorrência

# Transações

---

- Unidade lógica de processamento em um BD que inclui uma ou mais operações de acesso:
  - leitura – recuperação
  - escrita – inserção, atualização, remoção

# Motivação

---

- Origem
  - aplicações comerciais e financeiras
- Características das aplicações-alvo
  - curta duração
  - tipos de dados restritos
  - tipos de operações restritos
- Problemas
  - compartilhamento de dados
  - ocorrência de falhas

# Exemplo de aplicação

---

## □ Transferência de fundos

1. obter n. contas origem e destino, quantidade
2. obter saldo conta origem
3. se saldo  $\geq$  quantidade então
  1. atualizar saldo das contas origem e destino

# Objetivos

---

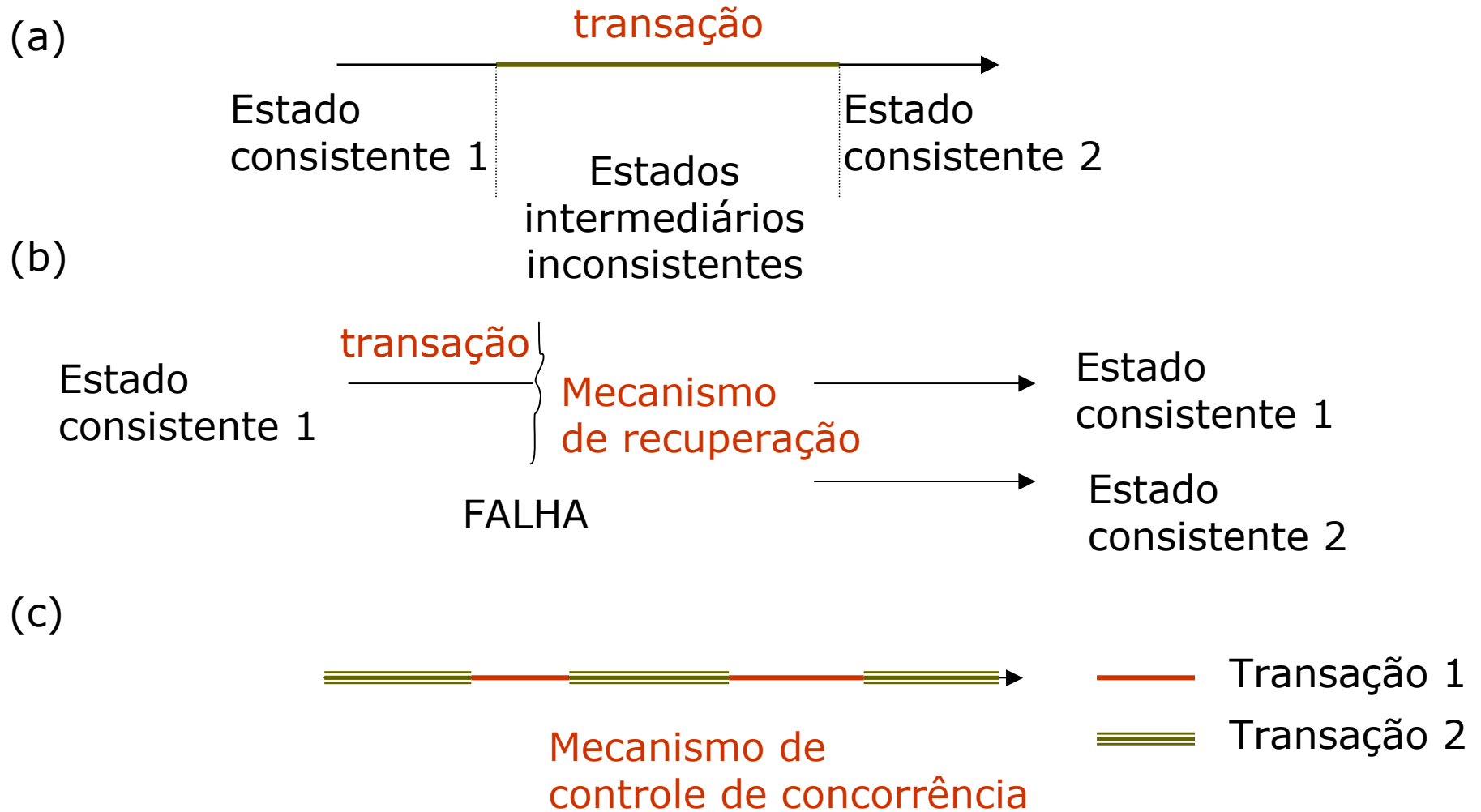
- Transparência
  - Compartilhamento de dados
  - Recuperação de falhas
- Simplificação da programação

# Operações de transações

---

- Operações básicas:
  - **read\_item(X)**: Lê um item do banco de dados chamado X colocando o valor em uma variável de programa
  - **write\_item(X)**: Escreve o valor de uma variável de programa X em um item de banco de dados chamado X
- Operações delimitadoras:
  - Begin transaction
  - End transaction

# Execução de transações



# Sumário

---

- Noções básicas
- **Propriedades**
- Estados
- Mecanismos no processamento de transações
  - Recuperação de falhas
  - Controle de concorrência

# Propriedades de transações - ACID

---

- **Atomicidade:** uma transação é uma unidade atômica de processamento; ou todas suas operações são executadas ou nenhuma é executada.
- **Consistência:** a execução de uma transação deve levar o BD de um estado consistente para outro.

# Propriedades de transações - ACID

---

- **Isolamento:** as atualizações de uma transação em execução não são visíveis para outras transações.
- **Durabilidade:** atualizações de transações efetivadas nunca devem ser perdidas em caso de falha.

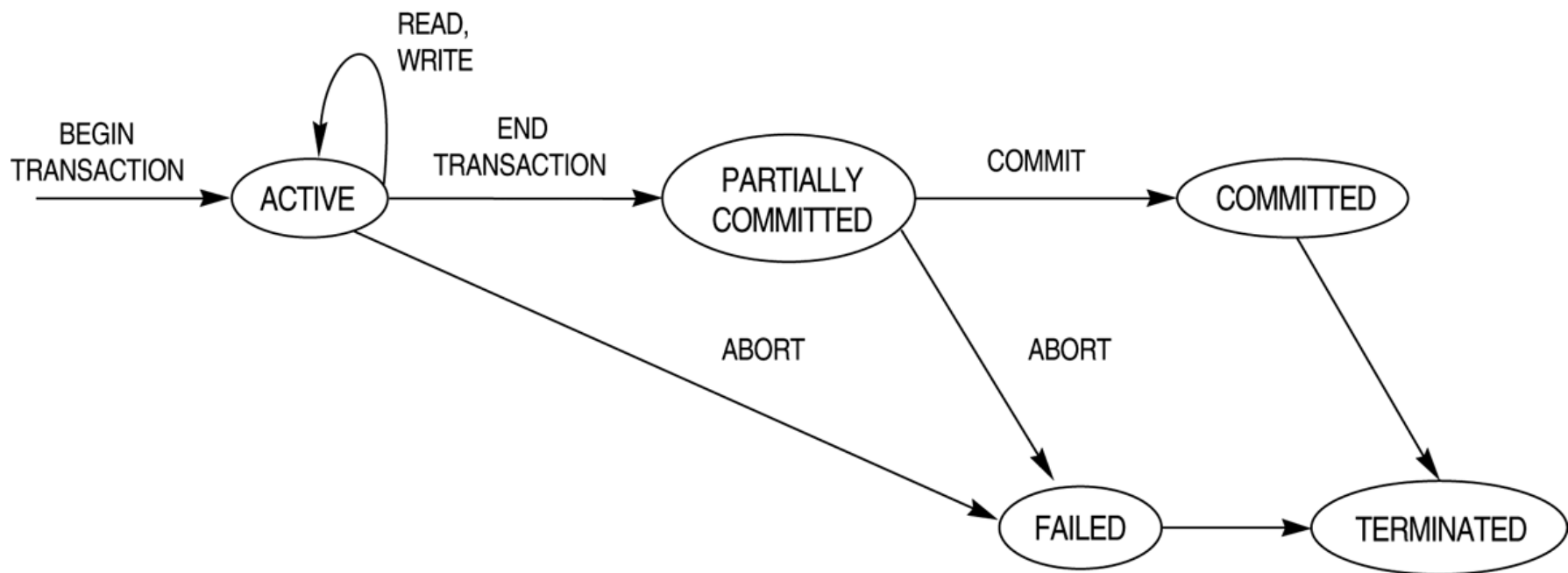
# Sumário

---

- Noções básicas
- Propriedades
- Estados
- Mecanismos no processamento de transações
  - Recuperação de falhas
  - Controle de concorrência

# Estados de uma transação

---



# Sumário

---

- Noções básicas
- Propriedades
- Estados
- Mecanismos no processamento de transações
  - Recuperação de falhas
  - Controle de concorrência

# Recuperação de falhas

---

## □ Tipos de falha

- falhas de transação
  - sem perda de informação
- falhas de sistema
  - com perda de informação da memória volátil
- falhas de memória secundária
  - com perda de informação da memória secundária
- falhas de comunicação
  - perda de mensagem, corrupção, duplicação,
  - mensagens fora de ordem
  - particionamento de rede

# Recuperação de falhas de transação e de sistema

---

- manutenção de dois estados
  - estado do sistema antes de a transação iniciar
  - estado contendo as atualizações da transação em execução
- mecanismos para recuperação
  - versões
  - logs

# Versões

---

- Histórico de versões
- Versão shadow

# Log

---

- ❑ mantém registro de todas as operações de transações que afetam os valores de itens do BD.
- ❑ Estas informações podem ser necessárias para recuperação
- ❑ É mantido no disco; não é afetado por falhas

# Log

---

1. [start\_transaction,T]: registra que a transação T começou a executar
2. [write\_item,T,X,old\_value,new\_value]: registra que a transação T mudou o valor do item do BD X de old\_value para new\_value
3. [read\_item,T,X]: registra que a transação T leu o valor do item X do BD
4. [commit,T]: registra que a transação T completou com sucesso e "efeitos" foram registrados no BD
5. [abort,T]: registro que a transação T foi abortada

# Ponto de Efetivação (commit point)

---

- Uma transação alcança seu ponto de efetivação quando todas as operações desta transação foram registradas no log
- Além deste ponto, a transação é dita efetivada (committed), e seu efeito foi registrado permanentemente no BD.
- A transação escreve, então, uma entrada do tipo [commit,T] no log.
- Cancelamento de transações: necessário para transações que tenham uma entrada [start\_transaction,T] no log mas nenhuma entrada [commit,T]

# Sumário

---

- Noções básicas
- Propriedades
- Estados
- Mecanismos no processamento de transações
  - Recuperação de falhas
  - Controle de concorrência

# Controle de concorrência

---

- transações sequenciais
  - garantia de consistência
  - compromete desempenho
- transações concorrentes
  - entrelaçamento de operações provoca interferência
  - melhora desempenho

# Controle de concorrência

---

- Tipos de interferência
  - Atualização perdida (lost update)
  - Atualização temporária (dirty read)
  - Sumário incorreto
- Objetivos
  - evitar interferência
  - produzir históricos corretos

# Exemplos de interferência

---

(a)

$T_1$

---

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

(b)

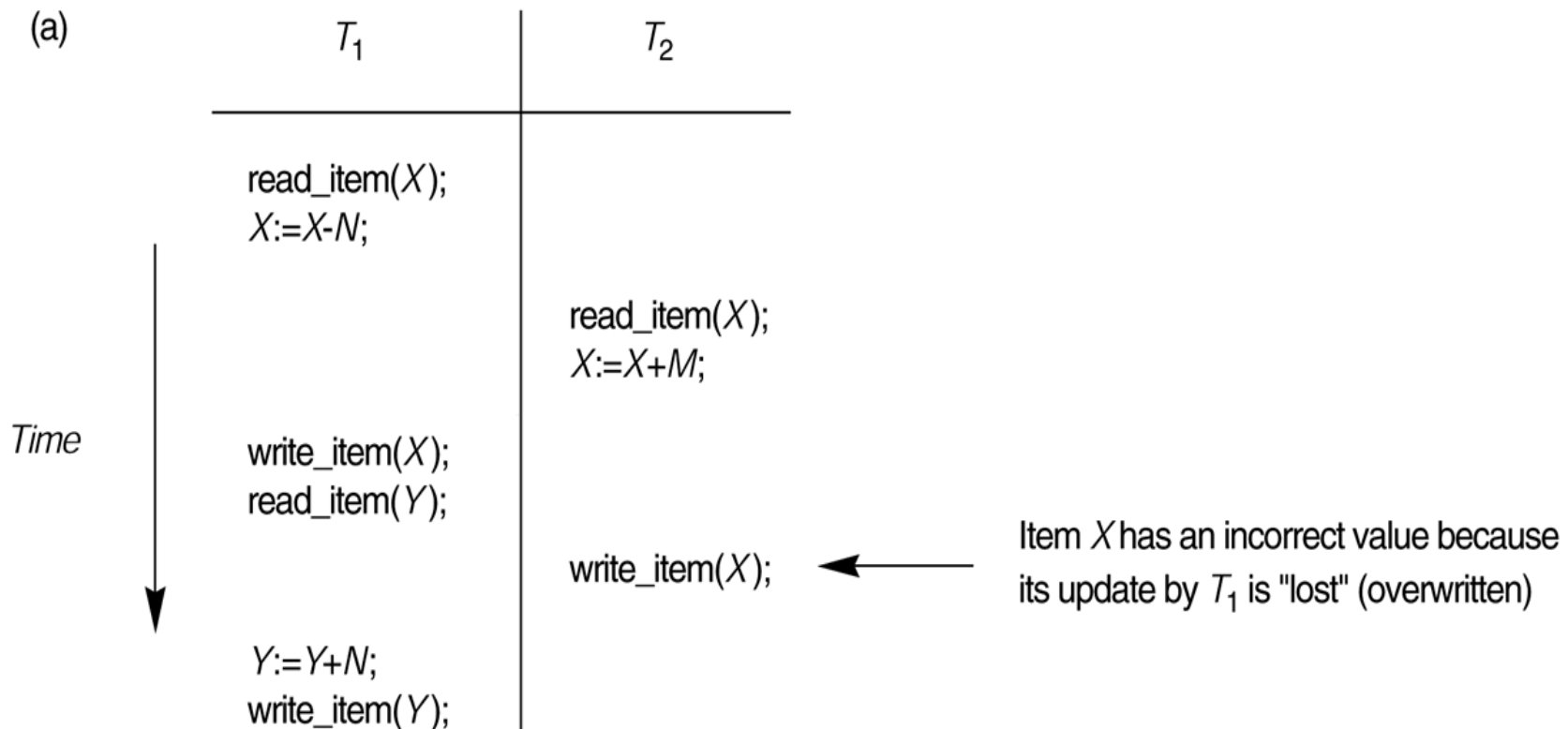
$T_2$

---

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

# Atualização perdida

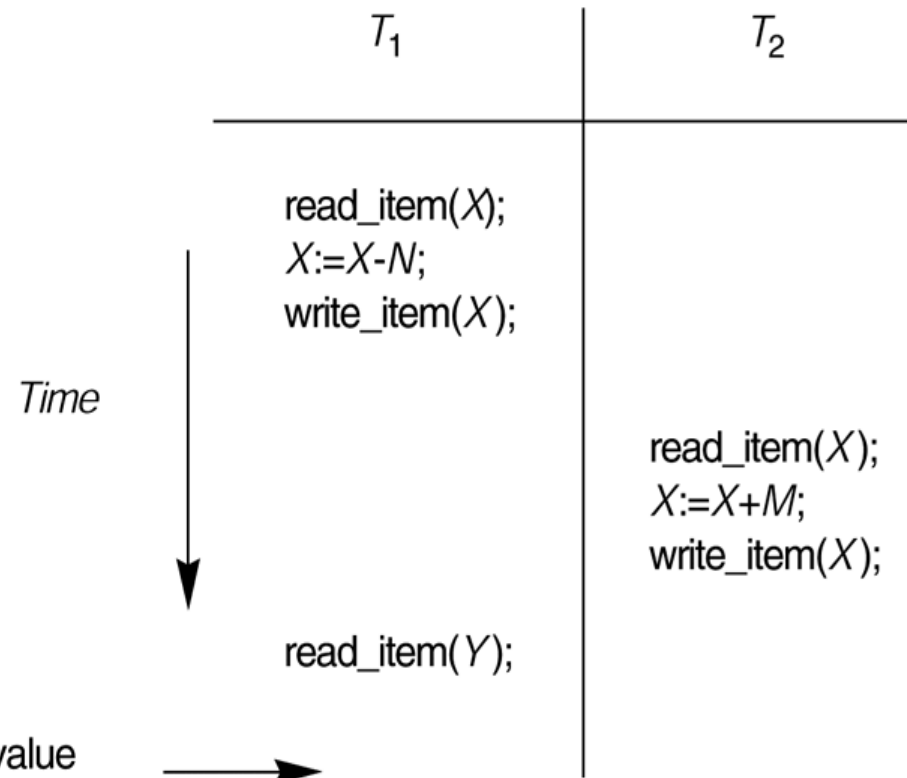
$X=80, N=5, M=4$



$X=84$

# Atualização temporária (dirty read)

(b)



Transaction  $T_1$  fails and must change the value of  $X$  back to its old value; meanwhile  $T_2$  has read the "temporary" incorrect value of  $X$ .

# Sumário incorreto

(c)	$T_1$	$T_3$
		<i>sum:=0;</i> <i>read_item(A);</i> <i>sum:=sum+A;</i>
		• • •
	<i>read_item(X);</i> <i>X:=X-N;</i> <i>write_item(X);</i>	
		<i>read_item(X);</i> <i>sum:=sum+X;</i> <i>read_item(Y);</i> <i>sum:=sum+Y;</i>
	<i>read_item(Y);</i> <i>Y:=Y+N;</i> <i>write_item(Y);</i>	

←  $T_3$  reads  $X$  after  $N$  is subtracted and reads  $Y$  before  $N$  is added; a wrong summary is the result (off by  $N$ ).

# Históricos (Schedules)

---

- Ordem de execução de operações de várias transações
- Dado um histórico  $S$  de  $n$  transações  $T_1, T_2, \dots, T_n$ :
  - a ordem das operações de uma transação  $T_i$  em  $S$  deve ser a mesma em que ocorrem em  $T_i$
  - As operações de uma transação podem ser intercaladas com outras transações de  $S$
- Tipos de históricos:
  - seriais - consistência
  - serializáveis - consistência e concorrência

# Históricos - conflitos

---

- Duas operações em um histórico conflitam se:
  - elas pertencem a duas transações diferentes,
  - acessam o mesmo item de dado e
  - uma das operações é de escrita
- Exemplo:

`r1(X); r2(X); w1(X); r1(Y); w2(X); c2; w1(Y); c1`

# Históricos - conflitos

---

- Duas operações em um histórico conflitam se:
  - elas pertencem a duas transações diferentes,
  - acessam o mesmo item de dado e
  - uma das operações é de escrita
- Exemplo:

r1(X); r2(X); w1(X); r1(Y); w2(X); c2; w1(Y); c1

# Históricos - conflitos

---

- Duas operações em um histórico conflitam se:
  - elas pertencem a duas transações diferentes,
  - acessam o mesmo item de dado e
  - uma das operações é de escrita
- Exemplo:

r1(X); r2(X); w1(X); r1(Y); w2(X); c2; w1(Y); c1

# Históricos - conflitos

---

- Duas operações em um histórico conflitam se:
  - elas pertencem a duas transações diferentes,
  - acessam o mesmo item de dado e
  - uma das operações é de escrita
- Exemplo:

`r1(X); r2(X); w1(X); r1(Y); w2(X); c2; w1(Y); c1`

# Classificação de históricos quanto à restaurabilidade

---

□ **Histórico recuperável:** Um histórico  $S$  é recuperável se nenhuma transação  $T$  em  $S$  é efetivada até que todas as transações  $T'$  que tenham escrito um item que  $T$  tenha lido tenham efetivado

- $S1: r1(X); w1(X); r2(X); r1(Y); w2(X); c2; a1;$
- $S2: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(y); c1; c2;$
- $S1$  não é recuperável

# Classificação de históricos quanto à restaurabilidade

---

- **Históricos que requerem cancelamento em cascata:** um histórico no qual transações não efetivadas lêem um item de uma transação que falhou
  - S1:  $r1(X); w1(X); r2(X); r1(y); w2(X); w1(Y); a1;$
  - T2 deve ser cancelada
- **Históricos sem cascata:** transações só lêem itens que tenham sido escritos por transações efetivadas
  - $r2(X)$  deveria ser atrasado até que T1 efetive
- **Históricos restritos:** histórico no qual uma transação não pode ler nem escrever um item X até que a última transação que escreveu X tenha efetivado ou abortado
  - Recuperável e evita cancelamento em cascata

# Histórico serial

---

- Um histórico  $S$  é **serial** se, para cada transação  $T$  no histórico  $S$ , todas as operações de  $T$  são executadas consecutivamente nesse histórico
  - Todo histórico serial mantém consistência
  - Limita concorrência

# Histórico serial

---

T1	T2
r(X)	
X:=X-N	
w(X)	
r(y)	
Y:=Y+N	
w(Y)	
	r(X)
	X:=X+M
	w(X)

# Histórico Serializável

---

- Um histórico  $S$  é **serializável** se ele é equivalente a algum histórico serial das mesmas  $n$  transações
  
- Dizer que um histórico não serial é serializável é equivalente a dizer que é correto, porque é equivalente a um histórico serial (que é considerado correto)
  
- Histórico serializável é um histórico correto:
  - Deixa o BD em um estado consistente
  - A intercalação é apropriada e o resultado é o mesmo se caso as transações tivessem sendo executadas serialmente
  - Porém mais eficiente dada a execução concorrente

# Equivalência de históricos

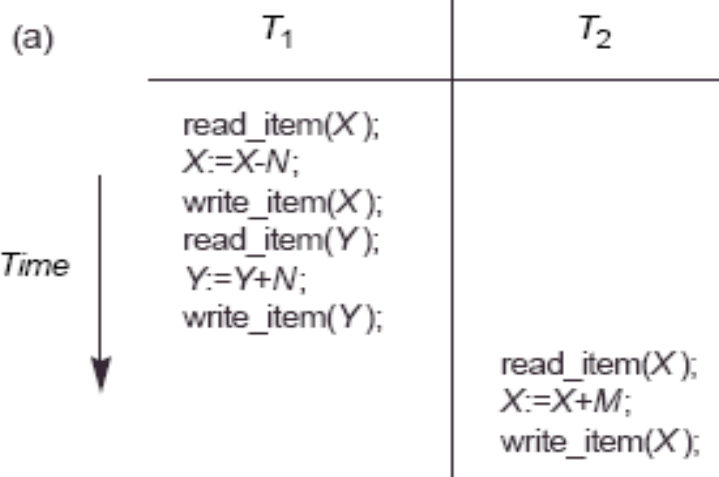
---

- **Equivalência quanto ao resultado:** dois históricos são equivalentes quanto ao resultado se eles produzem o mesmo estado final no BD
  - S1: r(X); X:=X+10; w(X)
  - S2: r(X); X:=X\*1,1; w(X)
  - Com valor de X inicial igual a 100
  
- **Equivalência quanto ao conflito:** dois históricos são equivalentes quanto ao conflito se a ordem de qualquer duas operações conflitantes é a mesma em ambos os históricos
  
- **Serialidade quanto ao conflito:** um histórico S é serializável quanto ao conflito se ele é equivalente quanto ao conflito a algum histórico serial S'

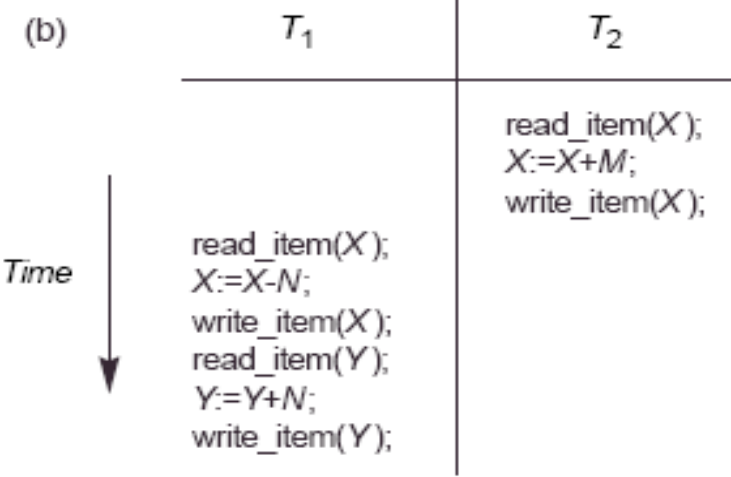
# Equivalência quanto ao conflito

---

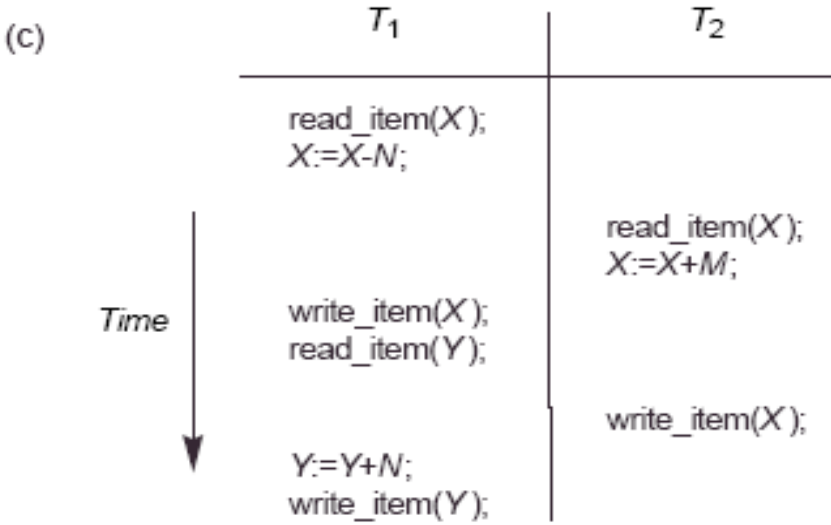
- Algoritmo para testar se um histórico é serializável quanto ao conflito:
  1. Identifique somente as operações `read_Item (X)` e `write_Item (X)`
  2. Construa um grafo de precedência (grafo de serialização) – um grafo com arestas direcionadas
  3. Uma aresta é criada de  $T_i$  para  $T_j$  se uma das operações em  $T_i$  aparece antes de uma operação conflitante em  $T_j$
  4. Um histórico é serializável se, e somente se, o grafo de precedência não tem ciclos



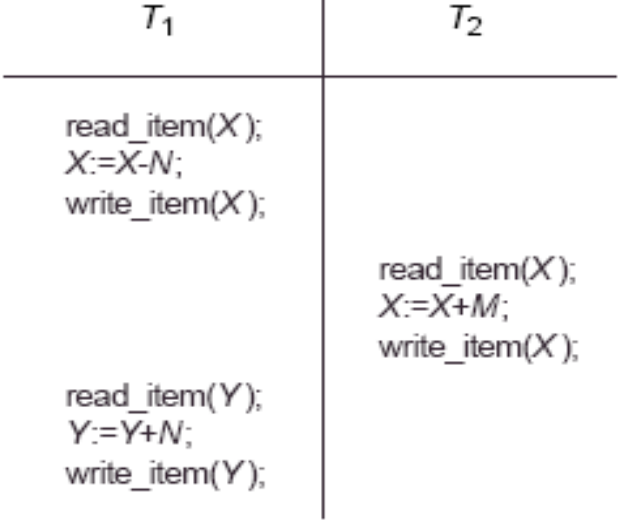
Schedule A



Schedule B



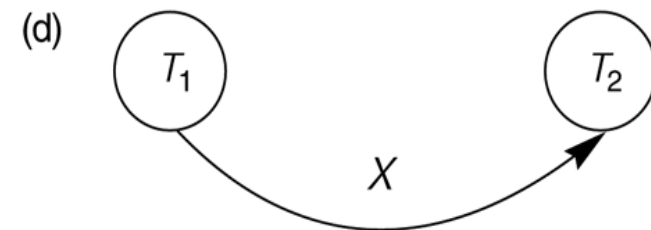
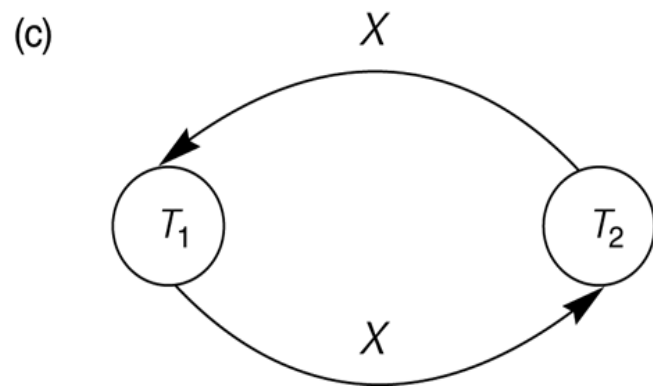
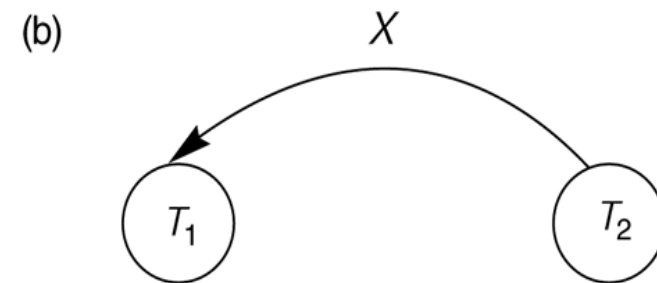
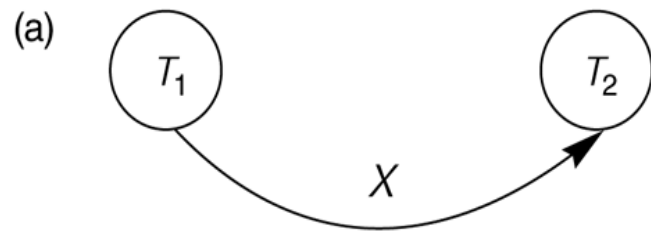
Schedule C



Schedule D

# Exemplos

---



# Exemplos

---

(a)

transaction $T_1$
read_item ( $X$ ); write_item ( $X$ ); read_item ( $Y$ ); write_item ( $Y$ );

transaction $T_2$
read_item ( $Z$ ); read_item ( $Y$ ); write_item ( $Y$ ); read_item ( $X$ ); write_item ( $X$ );

transaction $T_3$
read_item ( $Y$ ); read_item ( $Z$ ); write_item ( $Y$ ); write_item ( $Z$ );

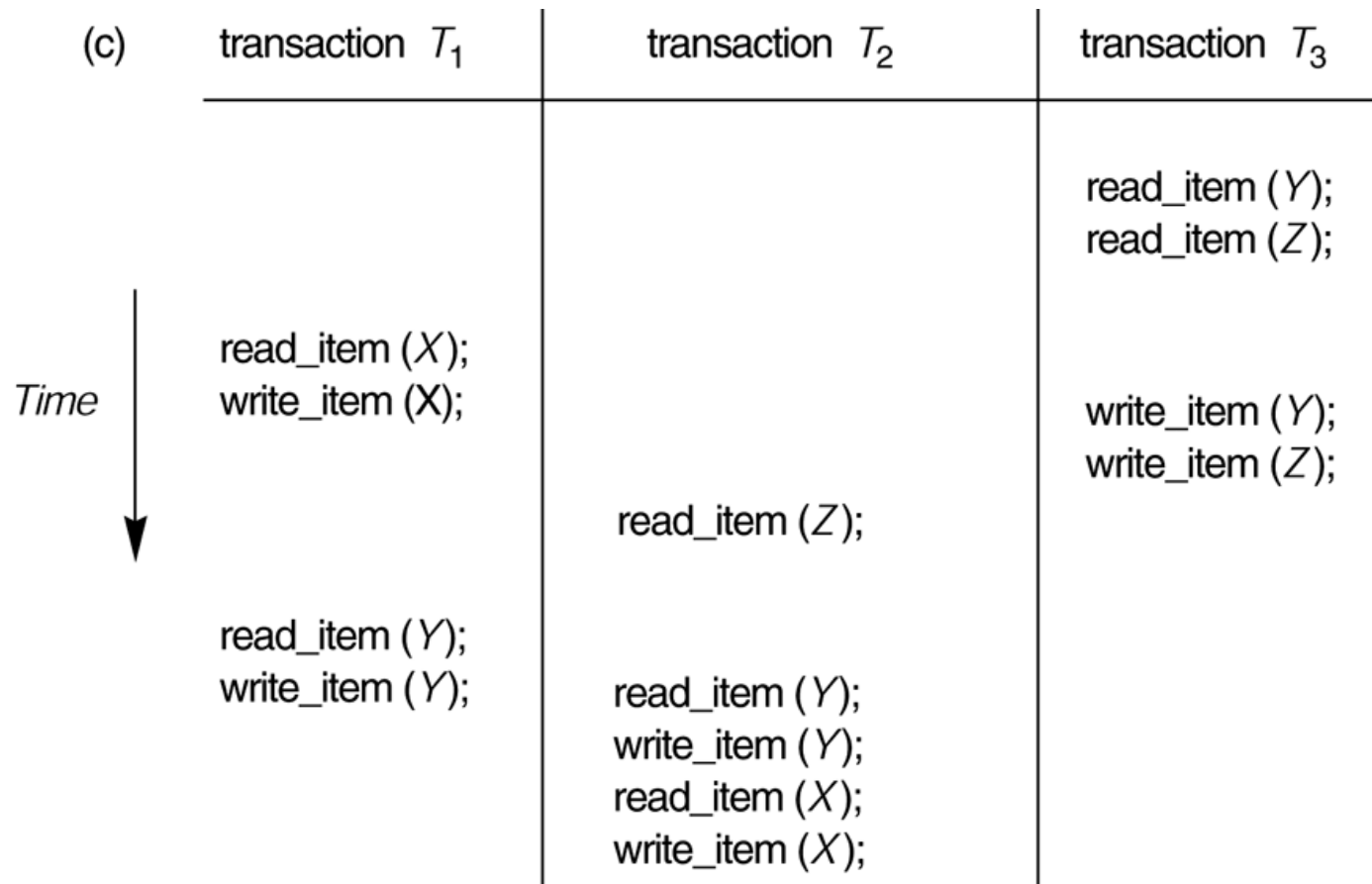
# Exemplos

(b)

	transaction $T_1$	transaction $T_2$	transaction $T_3$
		read_item (Z); read_item (Y); write_item (Y);	
			read_item (Y); read_item (Z);
Time ↓	read_item (X); write_item (X);		write_item (Y); write_item (Z);
		read_item (X);	
	read_item (Y); write_item (Y);	write_item (X);	

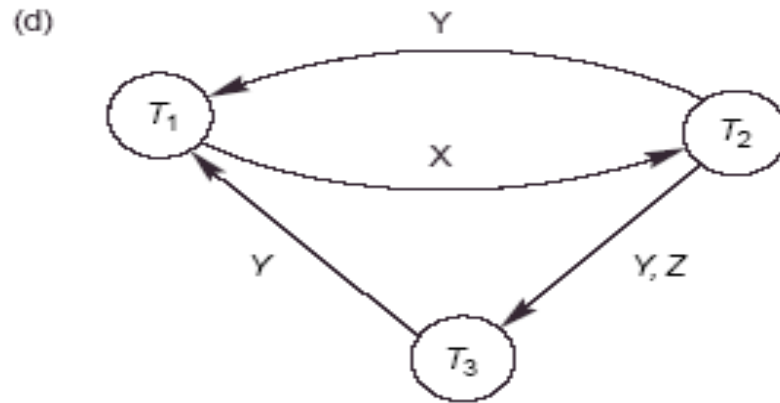
Schedule E

# Exemplos



Schedule F

# Exe

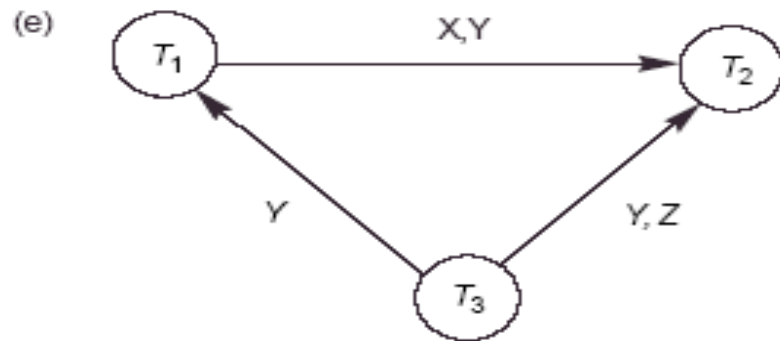


Equivalent serial schedules

None

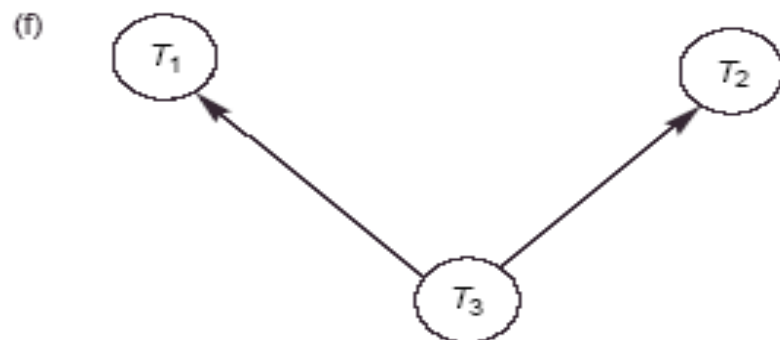
Reason

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



Equivalent serial schedules

$T_3 \rightarrow T_1 \rightarrow T_2$



Equivalent serial schedules

$T_3 \rightarrow T_1 \rightarrow T_2$

$T_3 \rightarrow T_2 \rightarrow T_1$

# Serialização – como garantir

---

- Serialização é difícil de ser verificada
- Protocolos que garantem a serialização
  - 2PL – Two-Phase Locking

# Serialização quanto à visão

---

Dois históricos são **equivalentes quanto à visão** se:

1. O mesmo conjunto de transações participam em  $S$  e  $S'$ ;  $S$  e  $S'$  incluem as mesmas operações destas transações
2. Para qualquer operação  $r_i(X)$  de  $T_i$  em  $S$ , se o valor de  $X$  lido foi escrito por uma operação  $w_j(X)$  de  $T_j$ , a mesma condição deve valer para o valor de  $X$  lido por uma operação  $r_i(X)$  de  $T_i$  em  $S'$
3. Se a operação  $w_k(Y)$  de  $T_k$  é a última operação que escreveu um item  $Y$  em  $S$ , então  $w_k(Y)$  de  $T_k$  deve ser a última operação a escrever  $Y$  em  $S'$

# Serialização quanto à visão

---

- Serialização quanto à visão:
  - Um histórico é serializável quanto à visão se ele é equivalente quanto à visão a um histórico serial
  - as operações de leitura têm a mesma *visão* em ambos os históricos
  
- Premissa:
  - Na medida em que a operação de leitura de uma transação lê o resultado da mesma operação de escrita em ambos os históricos, as operações de escrita devem produzir os mesmos resultados
  
- Equivalência quanto à visão é menos restritiva que equivalência quanto a conflito
  - Qualquer histórico serializável quanto ao conflito é também serializável quanto à visão, mas não vice-versa

# Serialização quanto à visão

---

T1: r1(X), w1(X);

T2: w2(X);

T3: w3(X);

Sa: r1(X); w2(X); w1(X); w3(X); c1; c2; c3;

- Em Sa, as operações w2(X) e w3(X) são escritas cegas, já que T1 e T3 não lêem o valor de X
- Sa é serializável quanto à visão, na medida em que ele é equivalente quanto à visão ao histórico serial T1, T2, T3.
- Sa não é serializável quanto ao conflito, já que não é equivalente quanto ao conflito a nenhum histórico serial