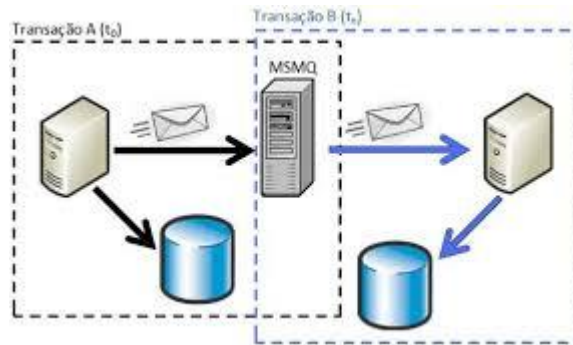




## Unidade 11 – Processamento de Transações



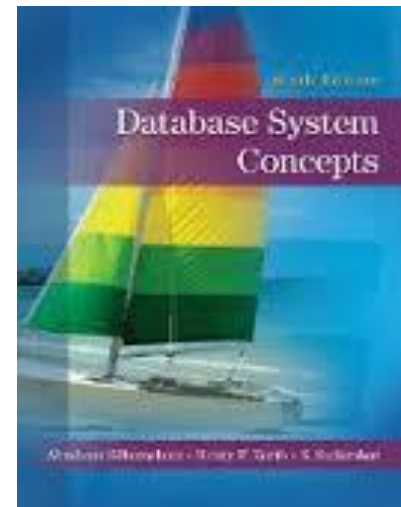
Prof. Aparecido V. de Freitas  
Doutor em Engenharia  
da Computação pela EPU SP



# Bibliografia



Sistemas de Banco de Dados  
Elmasri / Navathe 6ª edição



Sistema de Banco de Dados  
Korth, Silberschatz – Sixth Edition



# Introdução

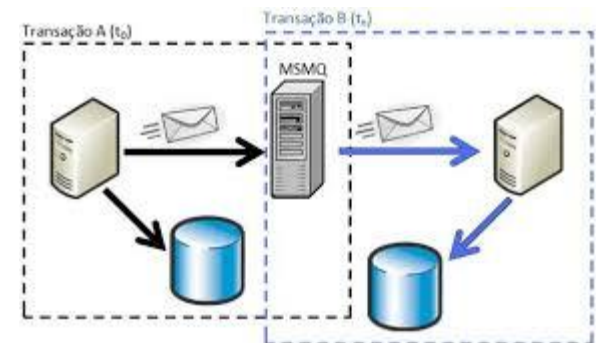
- ✦ **Sistemas de Processamento de Transações** são sistemas com grandes bancos de dados e centenas (milhares) de usuários simultâneos que executam **transações** de banco de dados.
- ✦ Exemplos: Sistemas de grande porte com aplicações de reservas aéreas, sistemas bancários, processamento de cartão de crédito, mercado de ações, etc.
- ✦ Esses sistemas exigem **alta disponibilidade** e **tempo de resposta rápido** para centenas de usuários simultâneos.





# Transação

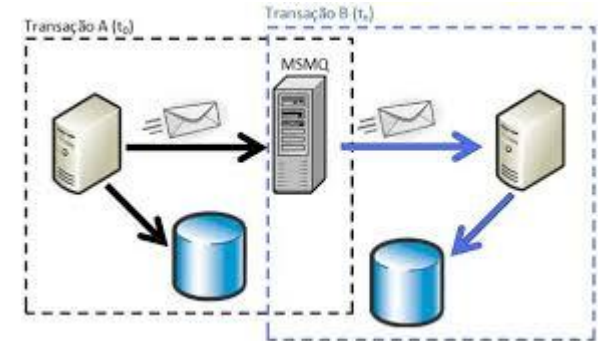
- ⊕ Uma transação é um programa em execução que forma uma Unidade Lógica de processamento de banco de dados;
- ⊕ Ela inclui uma ou mais operações de acesso ao banco de dados – estas podem ser de inserção, exclusão, modificação ou recuperação ;
- ⊕ Essas operações podem ser embutidas em um programa de aplicação;
- ⊕ As instruções **BEGIN TRANSACTION** e **END TRANSACTION** especificam os limites da transação dentro do programa de aplicação;
- ⊕ Se as operações não alterarem o banco de dados, a transação é chamada transação de consulta; se alterarem a transação é chamada transação de update;





# Transação e Concorrência

- ⊕ **Transação:** Uma sequência de leituras e gravações no banco de dados;
- ⊕ **Perspectiva do Usuário:** A transação é executada individualmente;
- ⊕ **Perspectiva do SGBD:** Concorrência intercalando leituras/gravações de várias transações.





# Operações básicas



As operações básicas de acesso ao banco de dados que uma transação pode incluir são as seguintes:

- **Read\_item(X)**. Lê um item do banco de dados chamado X para uma variável do programa. Para simplificar a notação, considera-se que variável de programa também é chamada X.
- **Write\_item(X)**. Grava o valor de uma variável de programa X no item de banco de dados chamado X.



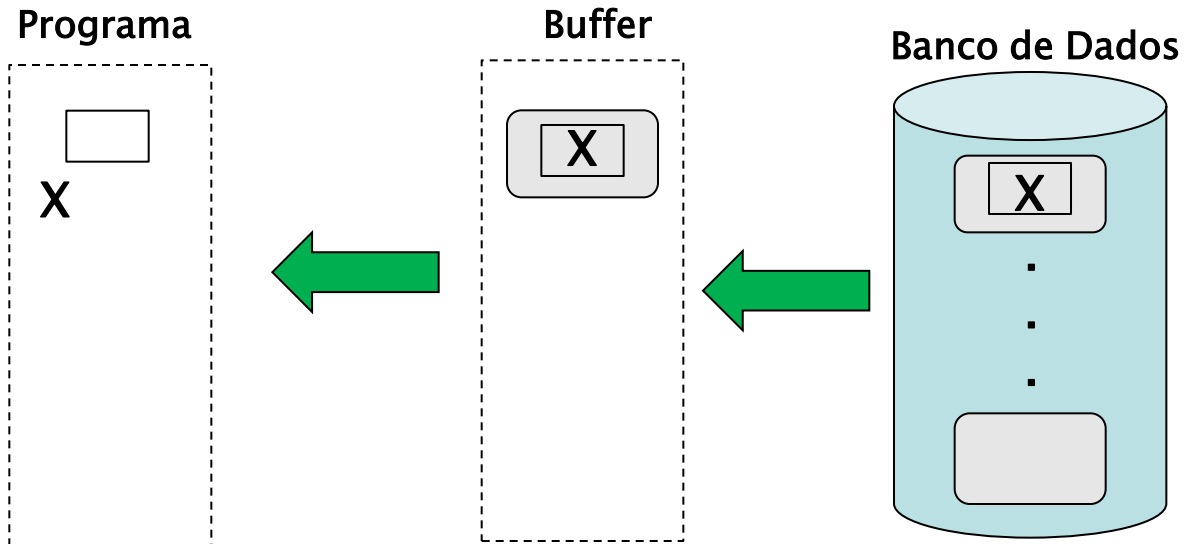
A unidade básica de transferência de dados do disco para a memória principal é um **BLOCO**.





## Operações básicas `read_item(x)`

1. Ache o endereço do bloco de disco que contém o item **X**.
2. Copie esse bloco de disco para um buffer na memória principal (se esse bloco de disco ainda não estiver em algum buffer da memória principal).
3. Copie o item **X** do buffer para a variável de programa também, por simplicidade, chamada **X**.

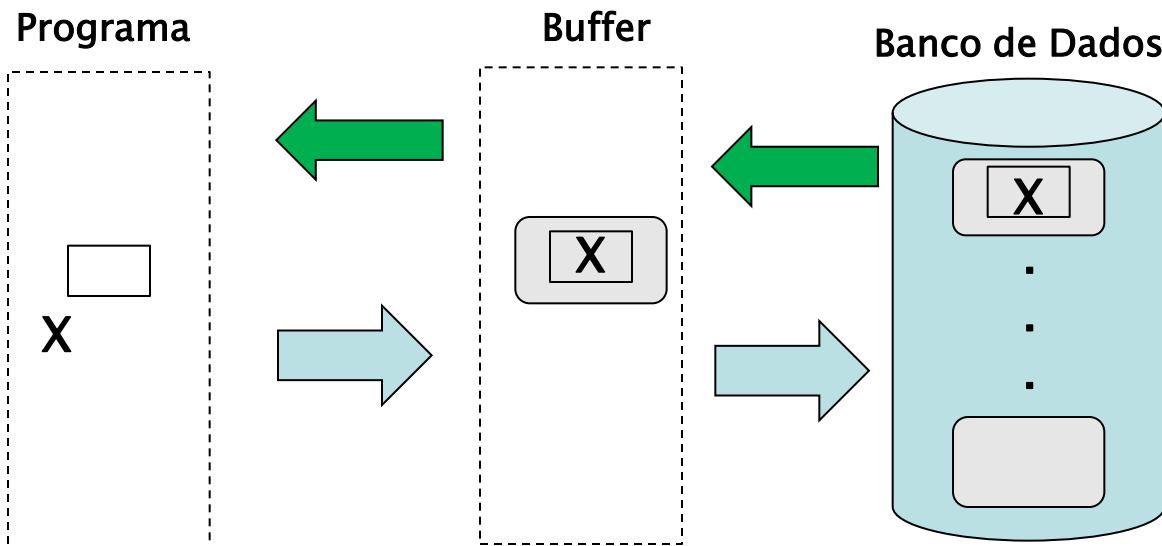




## Operações básicas `write_item(x)`

1. Ache o endereço do bloco de disco que contém o item **X**.
2. Copie esse bloco de disco para um **buffer** na memória principal (se esse bloco de disco ainda não estiver em algum buffer da memória principal).
3. Copie o item **X** da variável de programa, por simplicidade, também chamada **X** para o local correto no **buffer**.
4. Armazene o bloco **atualizado** do **buffer** de volta no disco (imediatamente ou em um momento posterior).

É a etapa 4 que de fato atualiza o banco de dados no disco !







## Exemplo de uma Transação T1

$T_1$
<pre>read_item(X); X := X - N; write_item(X); read_item(Y); Y := Y + N; write_item(Y);</pre>

- ✓ Conjunto de Leitura de T1 = { X, Y }
- ✓ Conjunto de Gravação de T1 = { X, Y }





## Exemplo de uma Transação T2

$T_2$
<pre>read_item(X); X := X + M; write_item(X);</pre>

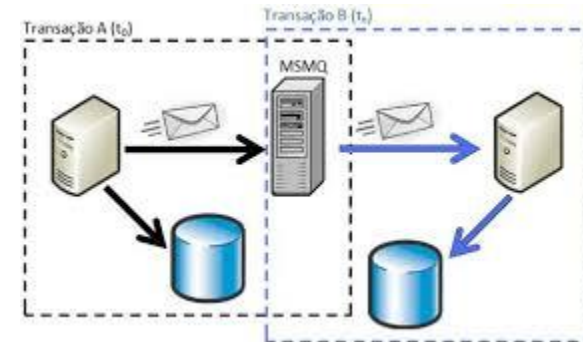
- ✓ Conjunto de Leitura de T2 = { X }
- ✓ Conjunto de Gravação de T2 = { X }





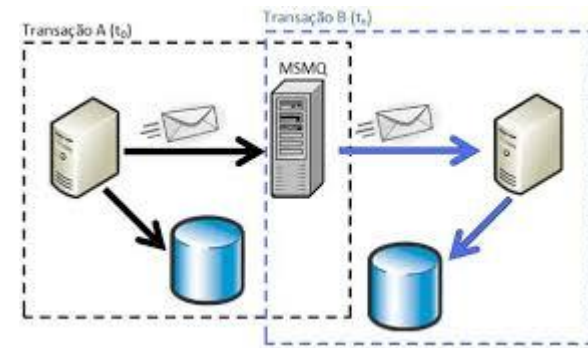
# Controle de Concorrência

- Os mecanismos de controle de concorrência e recuperação tratam principalmente dos comandos de banco de dados em uma Transação;
- As transações submetidas pelos diversos usuários podem ser executadas simultaneamente, de modo a acessar e atualizar os mesmos itens de dados;
- Se essa execução simultânea for descontrolada, ela pode ocasionar problemas. O banco de dados pode se tornar inconsistente.





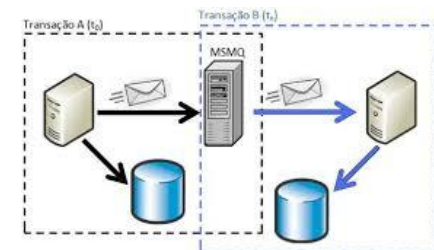
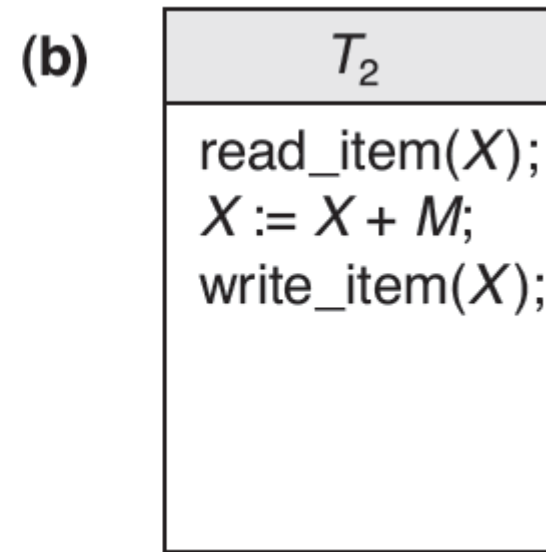
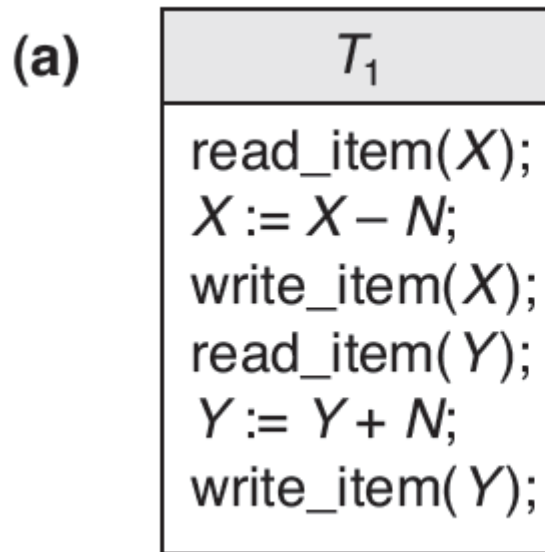
## Porque é necessário Controle de Concorrência ?





# Controle de Concorrência

- ⊕ Vários problemas podem ocorrer quando transações simultâneas são executadas de uma forma descontrolada (anomalias).





## Exemplo

# Transação 1: Transferência

**T1**

**ler** (X)

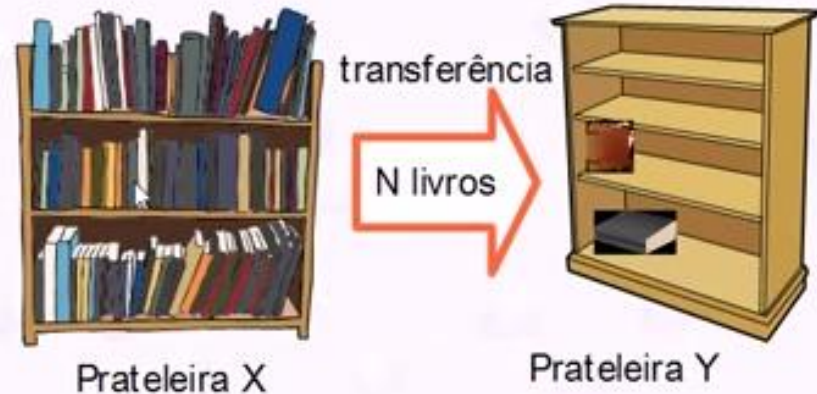
$X = X - N$

**gravar** (X)

**ler** (Y)

$Y = Y + N$

**gravar** (Y)



**Objetivo:** Transferir **N** livros de uma prateleira para outra;

**X:** número de livros da prateleira X

**Y:** número de livros na prateleira Y

**N:** número de livros a serem transferidos



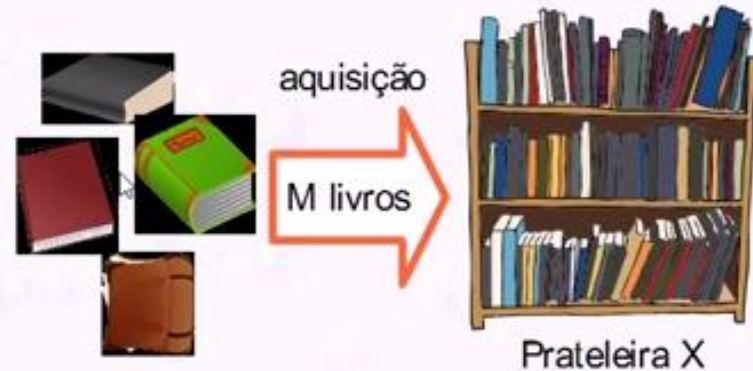
## Exemplo

# Transação 2: Aquisição

**T2**

---

ler (X)  
 $X = X + M$   
gravar (X)



**Objetivo:** Incluir novos livros na prateleira X;

**M:** novos livros que serão acrescentados na prateleira X





## Transações Concorrentes Plano de Execução

- Necessidade de um Plano de Execução

T1	T2
<b>ler</b> (X)	<b>ler</b> (X)
$X = X - N$	$X = X + M$
<b>gravar</b> (X)	<b>gravar</b> (X)
<b>ler</b> (Y)	
$Y = Y + N$	
<b>gravar</b> (Y)	

Será que a execução concorrente pode resultar em algum problema ?







## Plano de Execução Serial

T1	T2
<b>ler</b> (X) $X = X - N$ <b>gravar</b> (X) <b>ler</b> (Y) $Y = Y + N$ <b>gravar</b> (Y)	<b>ler</b> (X) $X = X + M$ <b>gravar</b> (X)



Com a serialização das transações, não haverá problemas de concorrência !





## Plano de Execução Serial

T1	T2
<code>ler (X)</code> <code>X = X - N</code> <code>gravar (X)</code> <code>ler (Y)</code> <code>Y = Y + N</code> <code>gravar (Y)</code>	<code>ler (X)</code> <code>X = X + M</code> <code>gravar (X)</code>

Com a inversão das transações serializadas também, não haverá problemas de concorrência !





## Plano de Execução Intercalado

T1	T2
<b>ler</b> (X) $X = X - N$	
	<b>ler</b> (X) $X = X + M$
<b>gravar</b> (X) <b>ler</b> (Y)	
	<b>gravar</b> (X)
$Y = Y + N$ <b>gravar</b> (Y)	





## Problema da Atualização Perdida

- ⊕ Esse problema ocorre quando duas transações que acessam os mesmos itens do banco de dados têm suas operações intercaladas de modo que isso torna o valor de alguns itens do banco de dados incorreto.

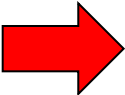


$T_1$	$T_2$
read(x); $x := x - N$ ;	read(x) $x := x + M$ ;
write(x); read(y);	write(x)
$y := y + N$ ; write(y)	



## Problema da Atualização Perdida

- Item X tem um valor **incorreto**, pois sua atualização por T1 é **perdida** (sobrescrita).

$T_1$	$T_2$
 read(x); $x := x - N$ ;	
	read(x) $x := x + M$ ;
write(x); read(y);	
	write(x)
$y := y + N$ ; write(y)	



## Problema da Atualização Temporária ou **Leitura Suja**

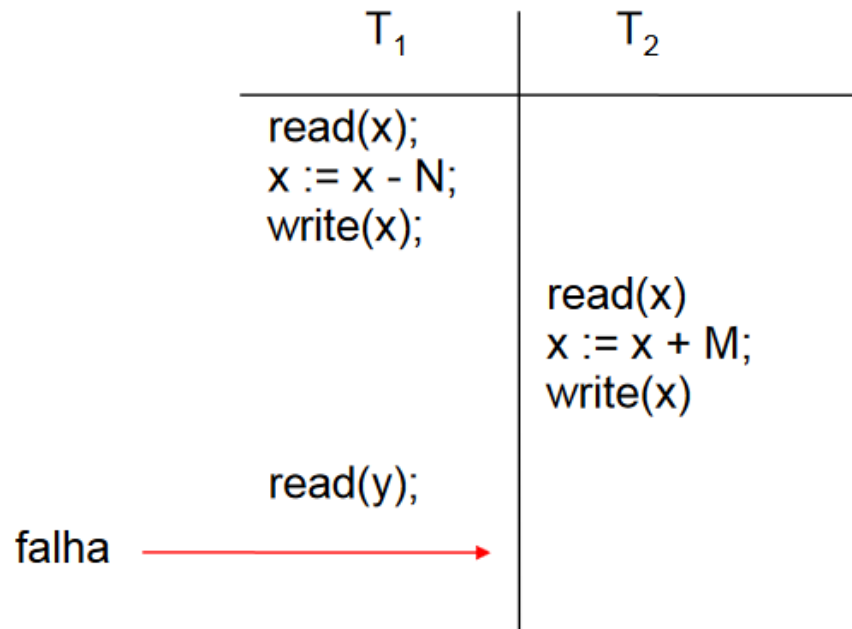
### Problema?

T1	T2
<b>ler</b> (X) $X = X - N$ <b>gravar</b> (X)	
	<b>ler</b> (X) $X = X + M$ <b>gravar</b> (X)
<b>ler</b> (Y) <b>***crash***</b>	



## Problema da Atualização Temporária ou **Leitura Suja**

- ⊕ Esse problema ocorre quando uma transação atualiza um item do banco de dados e depois a transação **falha** por algum motivo.
- ⊕ Nesse meio-tempo, o item atualizado é acessado (lido) por outra transação, antes de ser alterado de volta para seu valor original.
- ⊕ O valor do item **X** que foi lido por **T2** é chamado dado **sujo**, pois foi criado por uma transação que não foi concluída nem confirmada. Portanto, esse problema também é conhecido como **Problema da Leitura Suja**.





## Exemplo

# Transação 3: Sumário

### T3

soma = 0

**ler** (A)

soma = soma + A

...

**ler** (X)

soma = soma + X

**ler** (Y)

soma = soma + Y

...



## Problema?



T3 só faz leitura !





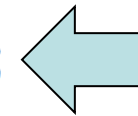
- ⊕ T1 está transferindo livros de uma prateleira para outra, enquanto que , concorrentemente, T3 está somando livros das prateleiras,



## Problema do Resumo Incorreto

- Se uma transação está calculando uma função agregada com um conjunto de tuplas e outras transações estão alterando algumas destas tuplas, a função agregada pode calcular alguns valores antes deles serem alterados e outros depois de serem alterados.

$T_1$	$T_3$
	$\text{sum} := 0;$ $\text{read}(a);$ $\text{sum} := \text{sum} + a;$
$\text{read}(x);$ $x := x - N;$ $\text{write}(x);$	$\text{read}(x)$ $\text{sum} := \text{sum} + x;$ $\text{read}(y);$ $\text{sum} := \text{sum} + y;$
$\text{read}(y);$ $y := y + N;$ $\text{write}(y)$	



T3 lê X depois que N é subtraído e lê Y antes que N seja somado; um resumo errado é o resultado (defasado por N)



# 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.

<b>T4</b>	<b>T4'</b>
<b>ler</b> (B)	
verifica (B)	
...	
	<b>ler</b> (B)
	verifica (B)
	...
	<b>ler</b> (B)
	reserva (B)
	<b>gravar</b> (B)
<b>ler</b> (B)	
reserva (B)	
<b>gravar</b> (B)	



## Porque a recuperação da transação é necessária ?

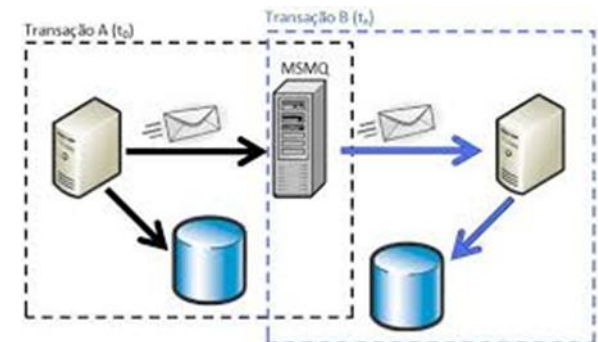
- ⊕ O **SGBD** deve **garantir** que todas as operações na transação sejam concluídas com sucesso e seu efeito seja registrado permanentemente no banco de dados.
- ⊕ O **SGDB** não deve permitir que algumas operações de uma transação **T** sejam aplicadas no banco de dados enquanto que outras operações de **T** não o são, pois a **transação inteira é uma unidade lógica de processamento de banco de dados**.
- ⊕ Se a transação **falhar** depois de executar algumas de suas operações, mas antes de executar todas elas, as operações já executadas devem ser **desfeitas** e não têm efeito duradouro.





# Resumo – Problemas com Transações Concorrentes

- ⊕ Atualização Perdida;
- ⊕ Atualização Temporária;
- ⊕ Resumo Incorreto;
- ⊕ Leitura Não Repetitiva;





# Transação – Definição

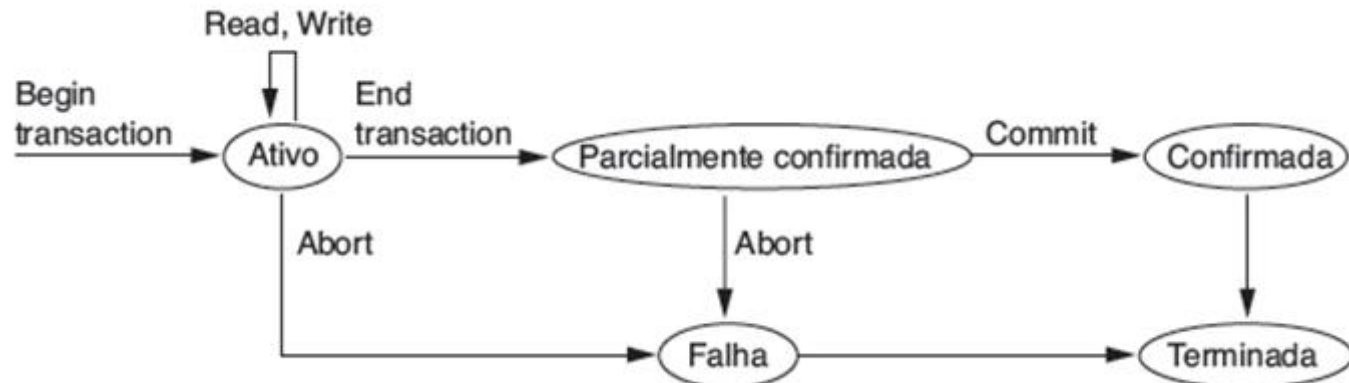
- ⊕ Uma **transação** é uma unidade **atômica** de trabalho, que deve ser concluída totalmente ou não ser feita de forma alguma.





# Estados de Transação

- ⊕ Para fins de recuperação, o sistema precisa registrar quando cada transação começa, termina e confirma ou aborta;
- ⊕ Portanto, o recuperador do SGBD precisa acompanhar as seguintes operações:
  - **BEGIN TRANSACTION**. Marca o início da transação;
  - **READ** ou **WRITE**. Operações nos itens dos bancos de dados, como parte da transação;
  - **END\_TRANSACTION**. Marca o final da transação;
  - **COMMIT\_TRANSACTION**. Sinaliza que a transação foi bem sucedida;
  - **ROLLBACK**. Sinaliza que a transação foi encerrada sem sucesso.





# Log do Sistema

- ⊕ Para recuperação das transações, o **SGBD** mantém um **log** para **registrar** todas as operações de transação que afetam os itens do banco de dados;
- ⊕ É um arquivo sequencial, apenas para inserção, mantido em disco. Em geral, um (ou mais) buffers de memória mantêm a última parte do arquivo de log. Quando o buffer de log é preenchido, seu conteúdo é anexado ao final do arquivo de log.
- ⊕ **Periodicamente**, é salvo em fita;
- ⊕ É estruturado pelos seguintes registros:
  - [start\_transaction,T]
  - [write\_item,T,X,valor\_antigo,valor\_novo]
  - [read\_item,T,X]
  - [commit,T]
  - [abort,T]







# Propriedades das Transações (ACID)

- ⊕ **Atomicidade.** Uma transação é uma unidade de processamento atômica. Deve ser realizada em sua totalidade ou não ser realizada de forma alguma.
- ⊕ **Consistência.** Uma transação deve preservar a consistência, ou seja deve levar o banco de dados de um estado consistente para outro.
- ⊕ **Isolamento.** A execução de uma transação não deve ser interferida por quaisquer outras transações que ocorram simultaneamente.
- ⊕ **Durabilidade.** As mudanças aplicadas ao banco de dados pela transação confirmada precisam persistir no banco de dados. Essas mudanças não devem ser perdidas por causa de alguma falha.





# Schedules de Transações

- ⊕ Um Schedule (ou histórico) **S** de  $n$  transações  $T_1, T_2, \dots, T_n$  é uma ordenação das operações das transações;
- ⊕ Aplicável à várias transações simultâneas;
- ⊕ Também chamado de Plano de Execução;
- ⊕ As operações das diferentes transações podem ser intercaladas no Schedule **S**;
- ⊕ Contudo, para cada transação  $T_i$  que participa no Schedule **S**, as operações de  $T_i$  em **S** precisam aparecer na mesma ordem em que ocorrem em  $T_i$ .





# Schedules de Transações

⊕ Uma notação para descrever um Schedule utiliza os símbolos:

b	=>	begin_transaction
r	=>	read_item
w	=>	write-item
e	=>	end-transaction
c	=>	commit
a	=>	abort





# Schedules de Transações – Exemplo

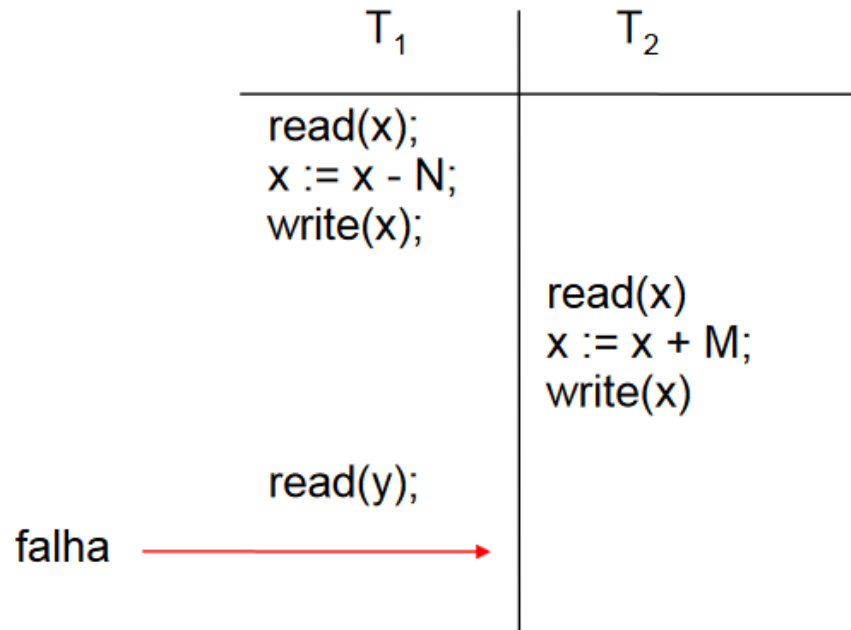
$T_1$	$T_2$
read(x); $x := x - N$ ;	
	read(x) $x := x + M$ ;
write(x); read(y);	
	write(x)
$y := y + N$ ; write(y)	

$S_a$ :  $r_1(\text{X})$ ;  $r_2(\text{X})$ ;  $w_1(\text{X})$ ;  $r_1(\text{Y})$ ;  $w_2(\text{X})$ ;  $w_1(\text{Y})$ ;





# Schedules de Transações – Exemplo



$S_b: r_1(\text{X}); w_1(\text{X}); r_2(\text{X}); w_2(\text{X}); r_1(\text{Y}); a_1;$





# Operações de conflito num Schedule

⊕ Duas operações em um schedule são consideradas como entrando em **CONFLITO** se satisfizerem a todas as três condições a seguir:

1. Elas pertencem a diferentes transações;
2. Elas acessam o mesmo item **X**;
3. E pelo menos uma das operações é um **write\_item(X)**;



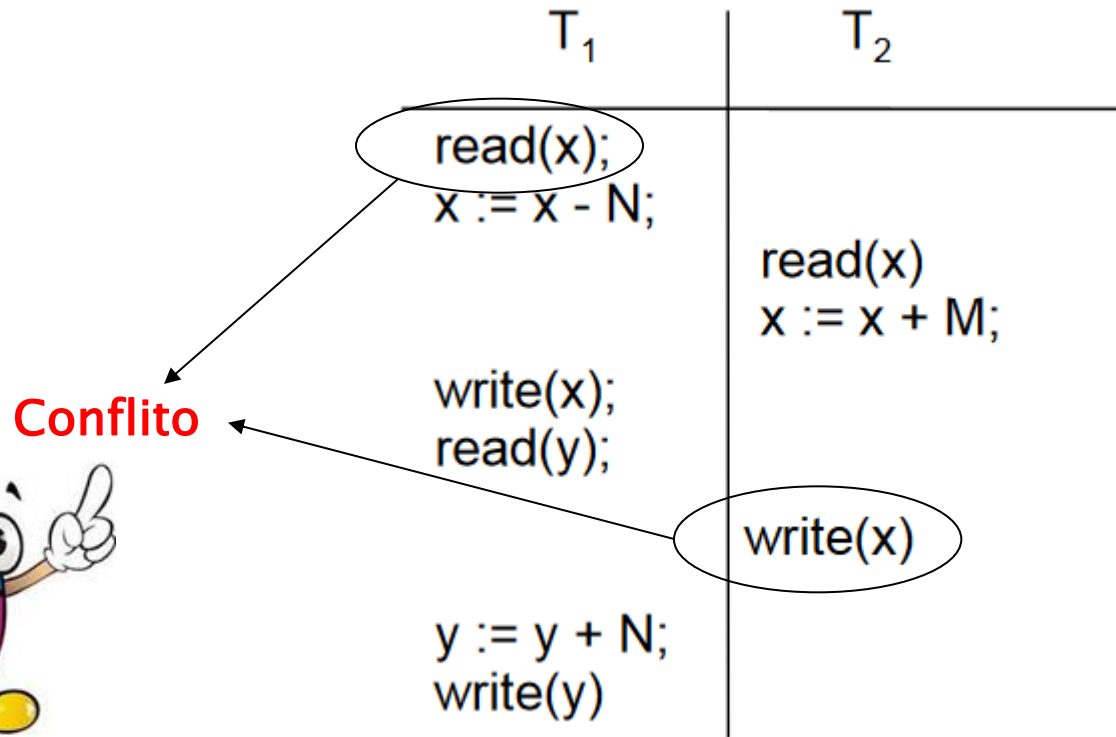
⊕ Observação: Duas operações de leitura NÃO estão em conflito, porque mudar sua ordem NÃO faz diferença no resultado.





# Exemplo – Operação de Conflito

⊕ No schedule  $S_a$ , as operações  $r_1(\mathbf{X})$  e  $w_2(\mathbf{X})$  estão em conflito

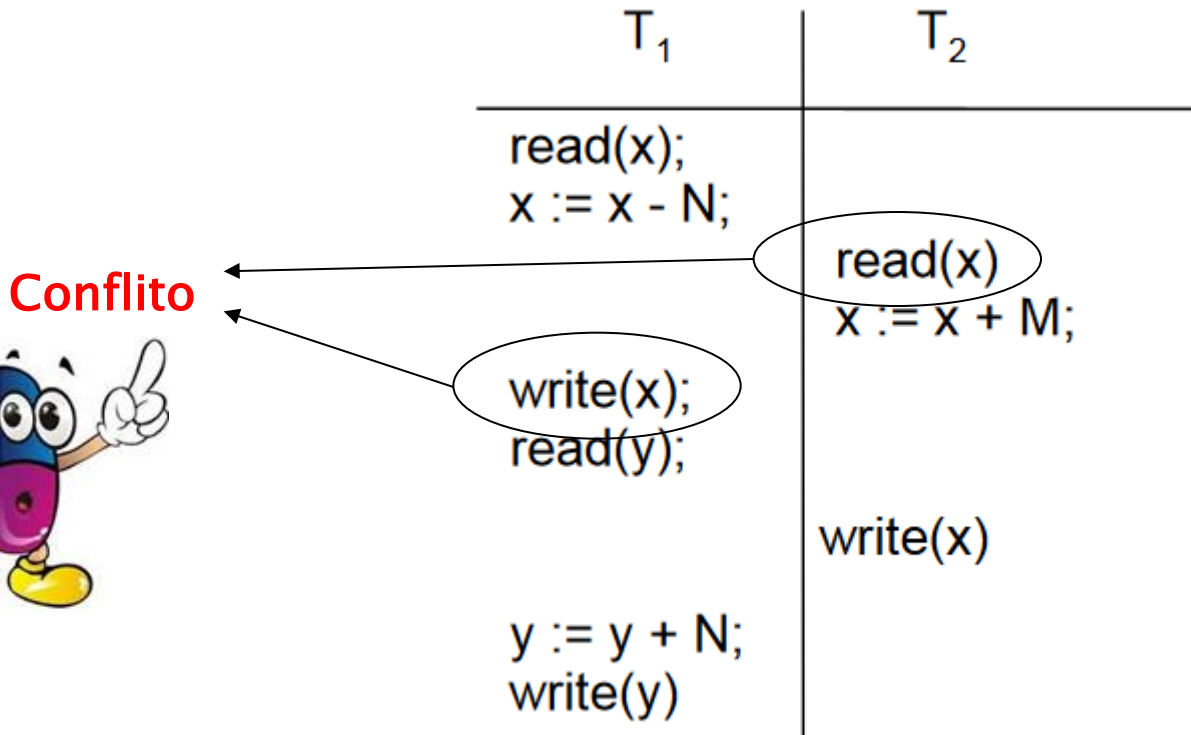


$S_a: r_1(\mathbf{X}); r_2(\mathbf{X}); w_1(\mathbf{X}); r_1(\mathbf{Y}); w_2(\mathbf{X}); W_1(\mathbf{Y});$



# Exemplo – Operação de Conflito

⊕ No schedule  $S_a$ , as operações  $r_2(X)$  e  $w_1(X)$  estão em conflito



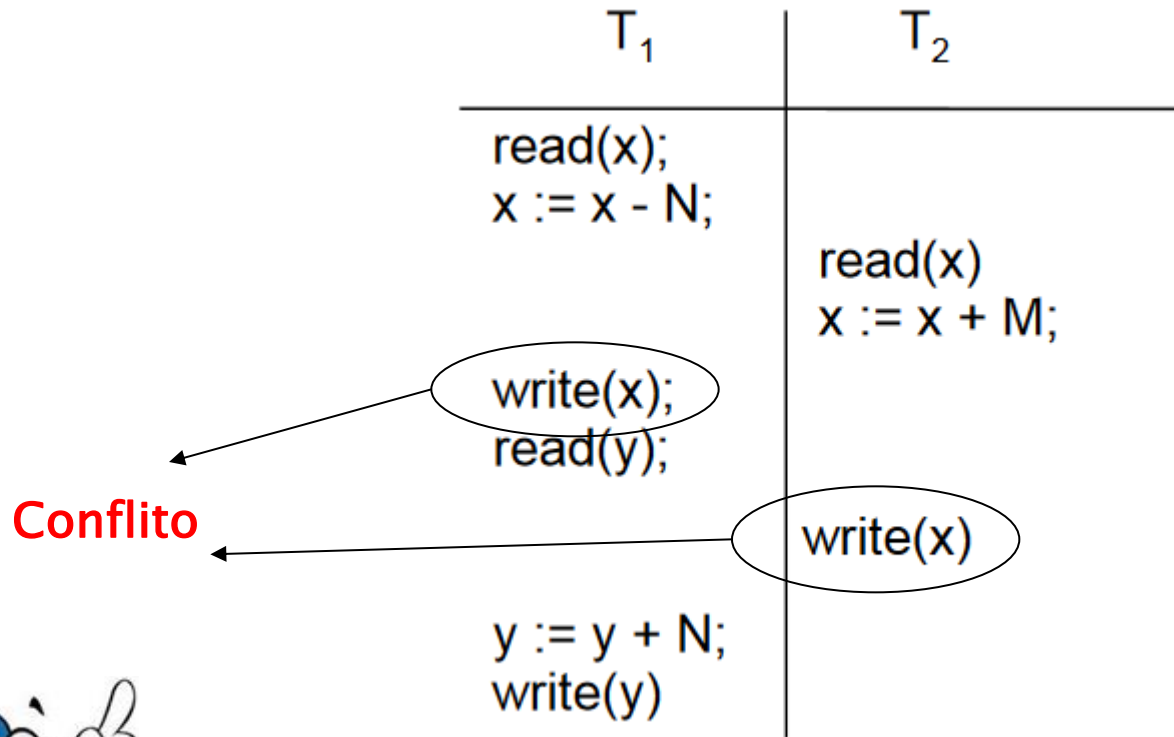
$S_a$ :  $r_1(X)$ ;  $r_2(X)$ ;  $w_1(X)$ ;  $r_1(Y)$ ;  $w_2(X)$ ;  $W_1(Y)$ ;





# Exemplo – Operação de Conflito

⊕ No schedule  $S_a$ , as operações  $w_1(\mathbf{x})$  e  $w_2(\mathbf{x})$  estão em conflito



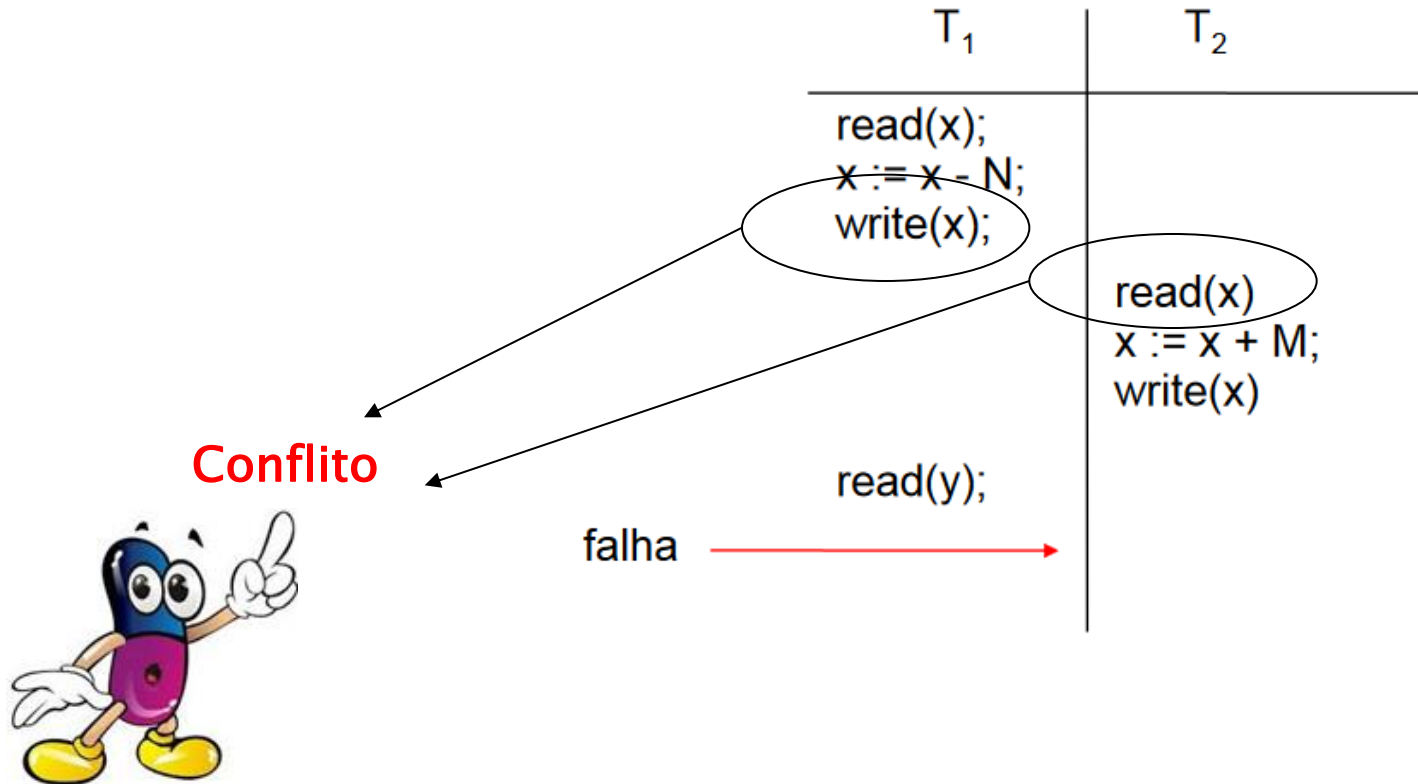
$S_a$ :  $r_1(\mathbf{x})$ ;  $r_2(\mathbf{x})$ ;  $w_1(\mathbf{x})$ ;  $r_1(\mathbf{Y})$ ;  $w_2(\mathbf{x})$ ;  $W_1(\mathbf{Y})$ ;





# Exemplo – Operação de Conflito

⊕ No schedule  $S_b$ , as operações  $w_1(X)$  e  $r_2(X)$  estão em conflito

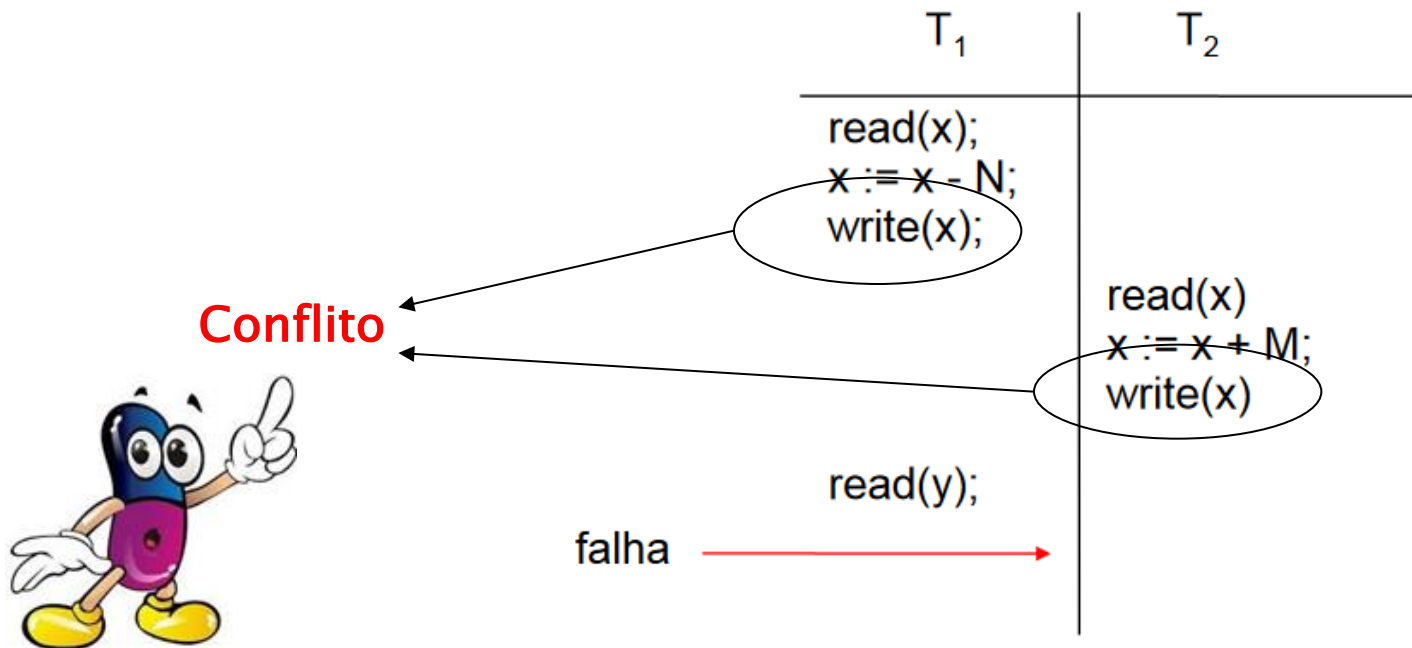


$S_b: r_1(X); w_1(X); r_2(X); w_2(X); r_1(Y); a_1;$



# Exemplo – Operação de Conflito

⊕ No schedule  $S_b$ , as operações  $w_1(\mathbf{x})$  e  $w_2(\mathbf{x})$  estão em conflito

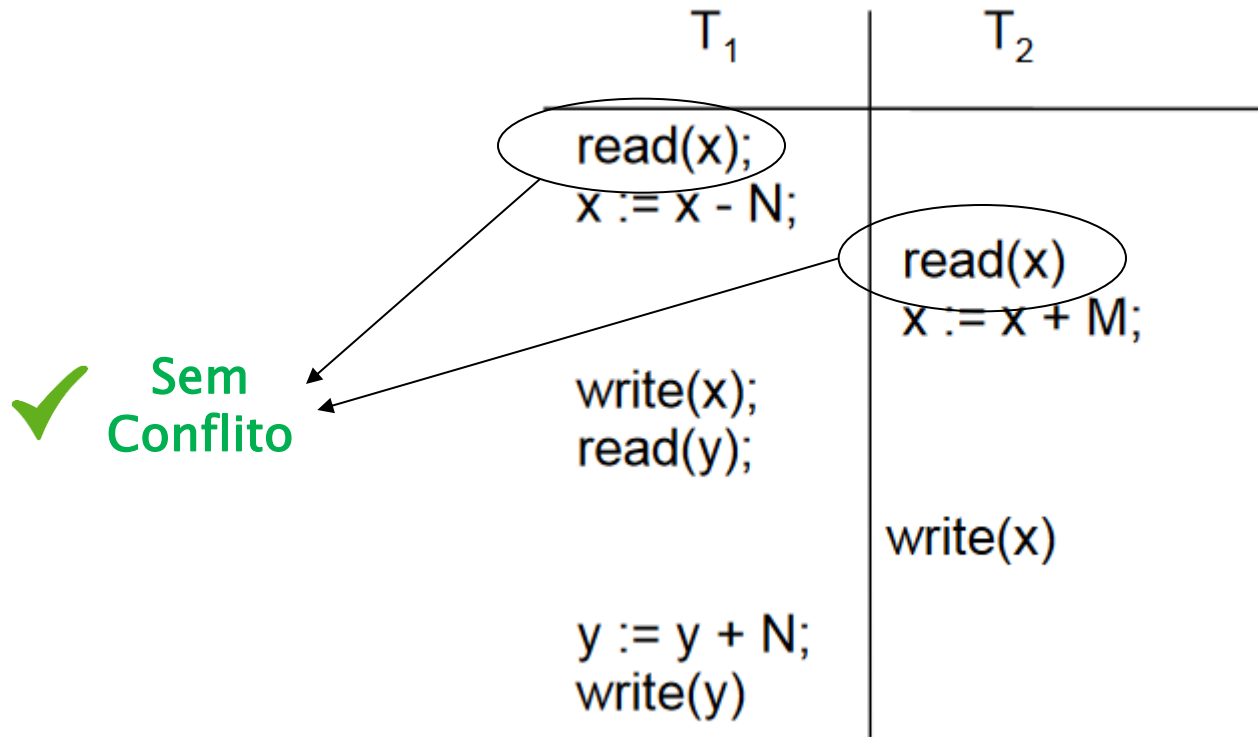


$S_b: r_1(\mathbf{x}); w_1(\mathbf{x}); r_2(\mathbf{x}); w_2(\mathbf{x}); r_1(\mathbf{y}); a_1;$



# Exemplo – Operação de Conflito

⊕ No schedule  $S_a$ , as operações  $r_1(X)$  e  $r_2(X)$  não estão em conflito

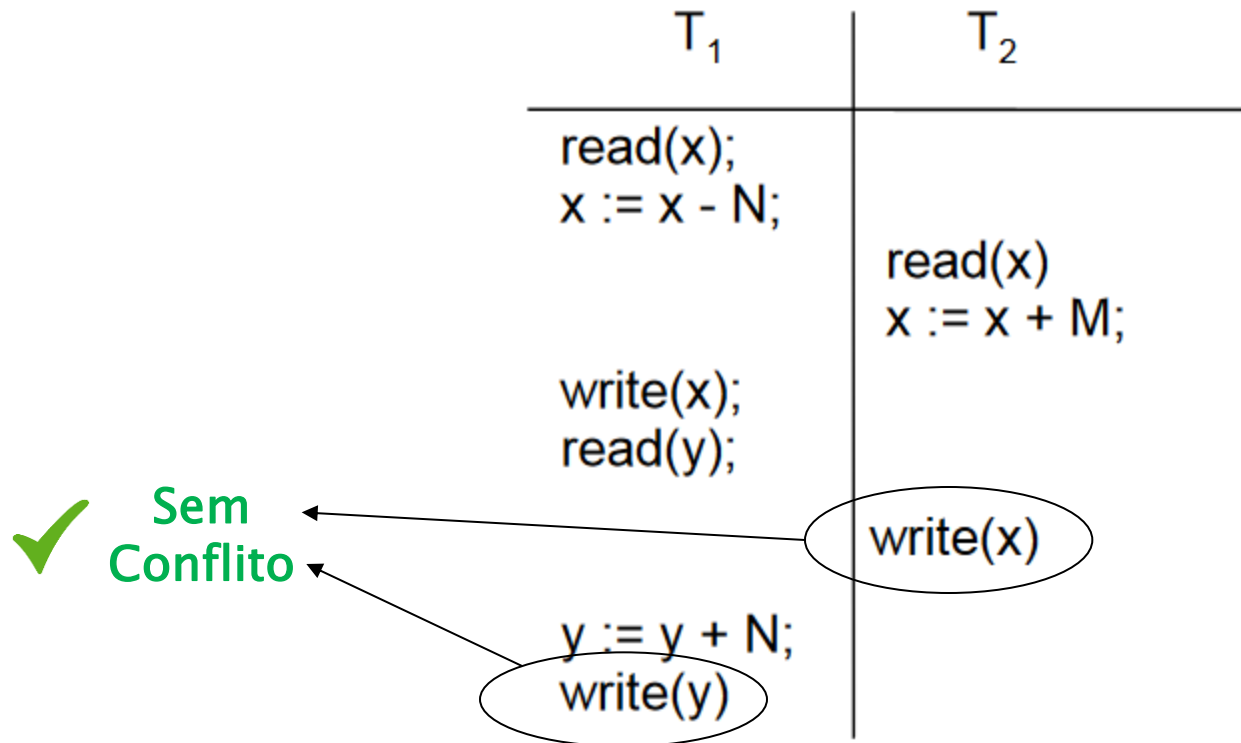


$S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); W_1(Y);$



# Exemplo – Operação de Conflito

⊕ No schedule  $S_a$ , as operações  $w_2(\text{X})$  e  $w_1(\text{Y})$  não estão em conflito

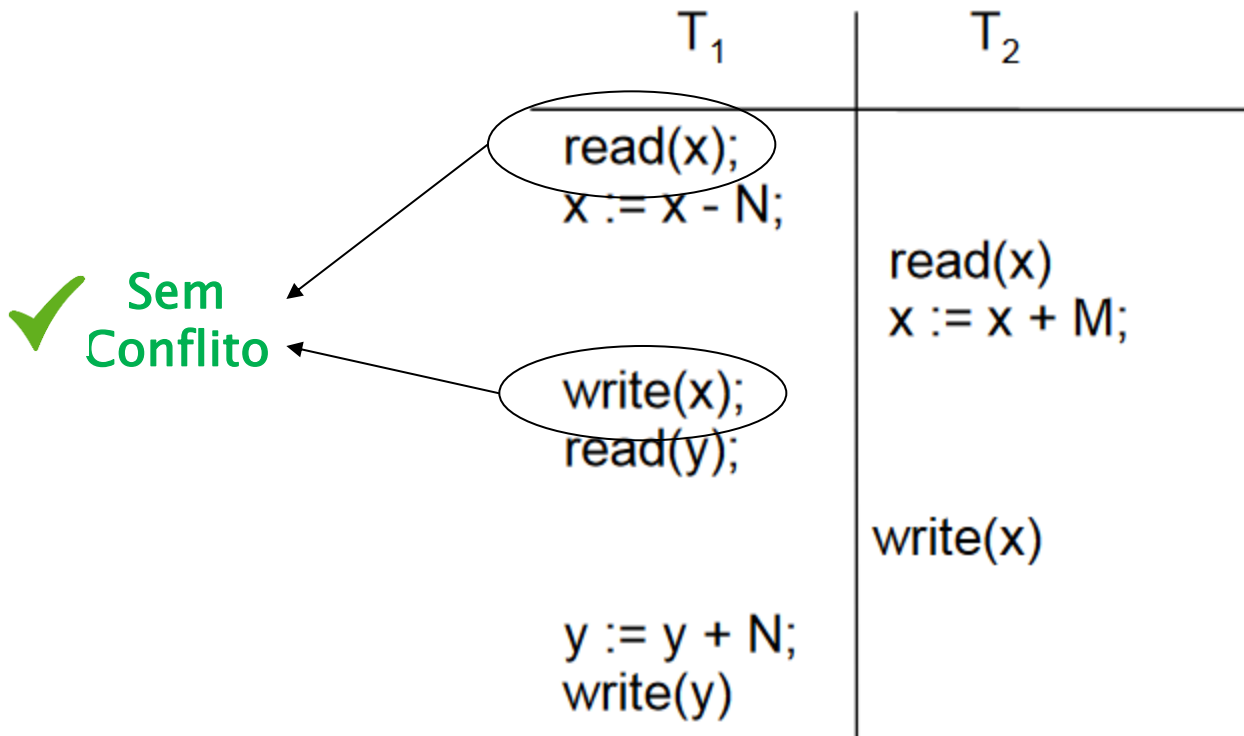


$S_a$ :  $r_1(\text{X})$ ;  $r_2(\text{X})$ ;  $w_1(\text{X})$ ;  $r_1(\text{Y})$ ;  $w_2(\text{X})$ ;  $w_1(\text{Y})$ ;



# Exemplo – Operação de Conflito

⊕ No schedule  $S_a$ , as operações  $r_1(X)$  e  $w_1(X)$  não estão em conflito



$S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); W_1(Y);$



# Recuperação de Transações

- ✦ O **SGBD** é responsável por garantir que todas as operações na transação sejam concluídas com sucesso e seu efeito seja registrado permanentemente no banco de dados, ou que a transação **não** tenha qualquer efeito no banco de dados;
- ✦ No primeiro caso, a transação é considerada confirmada (**committed**);
- ✦ No segundo caso, a transação é **abortada**;





# Schedule Serial e Serializável

- ⊕ No Schedule serial transações completas são executadas em série;
- ⊕ No Schedule serial não há intercalação de operações entre transações;
- ⊕ Um Schedule serializável é equivalente a algum Schedule serial.







## Banco de Dados - Unidade 11 – Processamento de Transações



## Schedule Serial 2

T1	T2
<b>ler</b> (X) $X = X - N$ <b>gravar</b> (X) <b>ler</b> (Y) $Y = Y + N$ <b>gravar</b> (Y)	<b>ler</b> (X) $X = X + M$ <b>gravar</b> (X)



## Este schedule é serializável ?

T1	T2
<b>ler</b> (X) $X = X - N$	
	<b>ler</b> (X) $X = X + M$
<b>gravar</b> (X) <b>ler</b> (Y)	
	<b>gravar</b> (X)
$Y = Y + N$ <b>gravar</b> (Y)	



Este schedule **NÃO** é serializável ?

T1	T2
<b>ler</b> (X) $X = X - N$	
	<b>ler</b> (X) $X = X + M$
<b>gravar</b> (X) <b>ler</b> (Y)	
	<b>gravar</b> (X)
$Y = Y + N$ <b>gravar</b> (Y)	

Observação: Vide problema da Atualização Perdida !





# Todo schedule NÃO serializável está errado ?





# Todo schedule NÃO serializável está errado ?

- ✦ Lembrando que um schedule serializável é todo schedule equivalente a um schedule serial (correto).
- ✦ Portanto, se o schedule é não serializável então ele está ERRADO !





## Este schedule é serializável ?

T1	T2
<b>ler</b> (X) $X = X - N$ <b>gravar</b> (X)	
	<b>ler</b> (X) $X = X + M$ <b>gravar</b> (X)
<b>ler</b> (Y) $Y = Y + N$ <b>gravar</b> (Y)	



## Este schedule é serializável ?

T1	T2
<b>ler</b> (X) $X = X - N$ <b>gravar</b> (X)	
	<b>ler</b> (X) $X = X + M$ <b>gravar</b> (X)
<b>ler</b> (Y) $Y = Y + N$ <b>gravar</b> (Y)	

- ⊕ **Sim**, considerando que as transações **não** abortaram, os resultados serão corretos;
- ⊕ Portanto, o schedule apresentado é serializável.





# Como avaliar se um schedule é serializável ?





# Algoritmo para Teste de Serialização – Grafo de Precedência

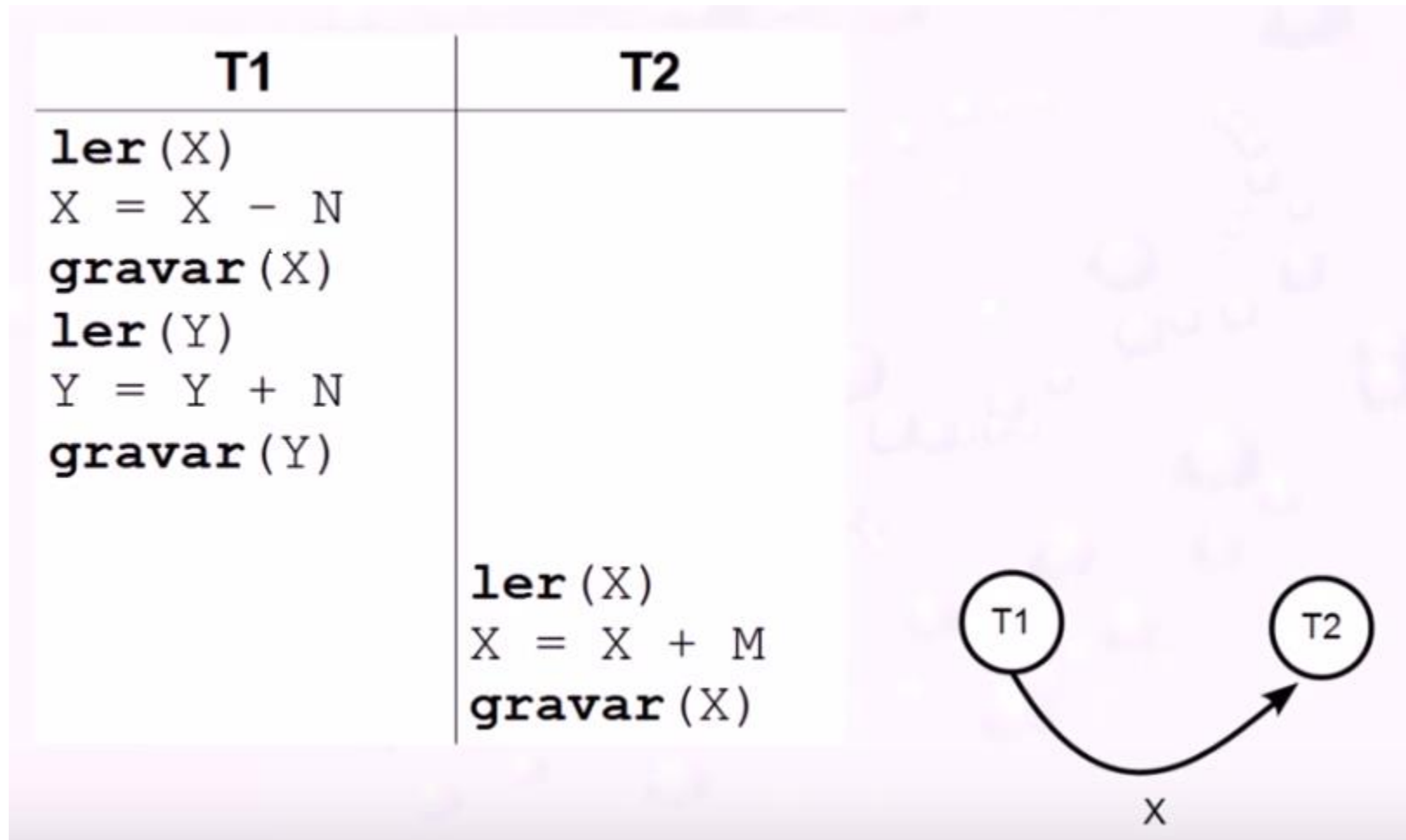
- ⊕ Para cada transação crie um nó no **grafo**;
- ⊕ Para cada caso no Schedule **S**:
  - ✓ Toda vez que  $T_j$  ler (X) depois de  $T_i$  gravar (X)  $\Rightarrow$  aresta  $T_i \rightarrow T_j$
  - ✓ Toda vez que  $T_j$  gravar (X) depois de  $T_i$  ler (X)  $\Rightarrow$  aresta  $T_i \rightarrow T_j$
  - ✓ Toda vez que  $T_j$  gravar (X) depois de  $T_i$  gravar (X)  $\Rightarrow$  aresta  $T_i \rightarrow T_j$
- ⊕ Depois de ter feito o procedimento acima para todos os itens sob conflito nas transações do Schedule S, **se não houver ciclos** (loops), o schedule é **serializável** !



**Obs.** A aresta é sempre da primeira ação para a segunda ação !



# Schedule Serial 1

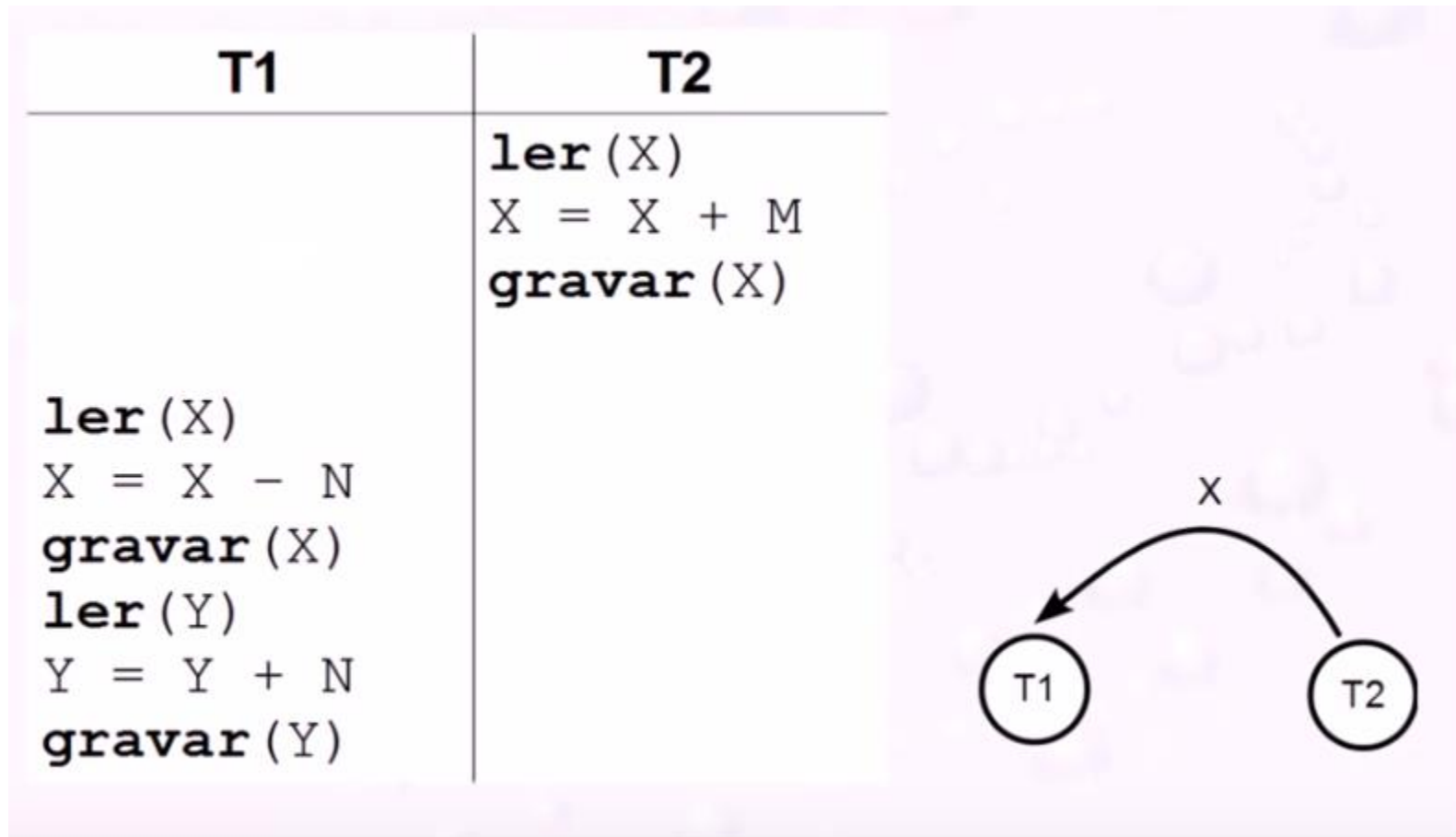


**Obs.** O grafo não tem ciclos, portanto o schedule é serializável !





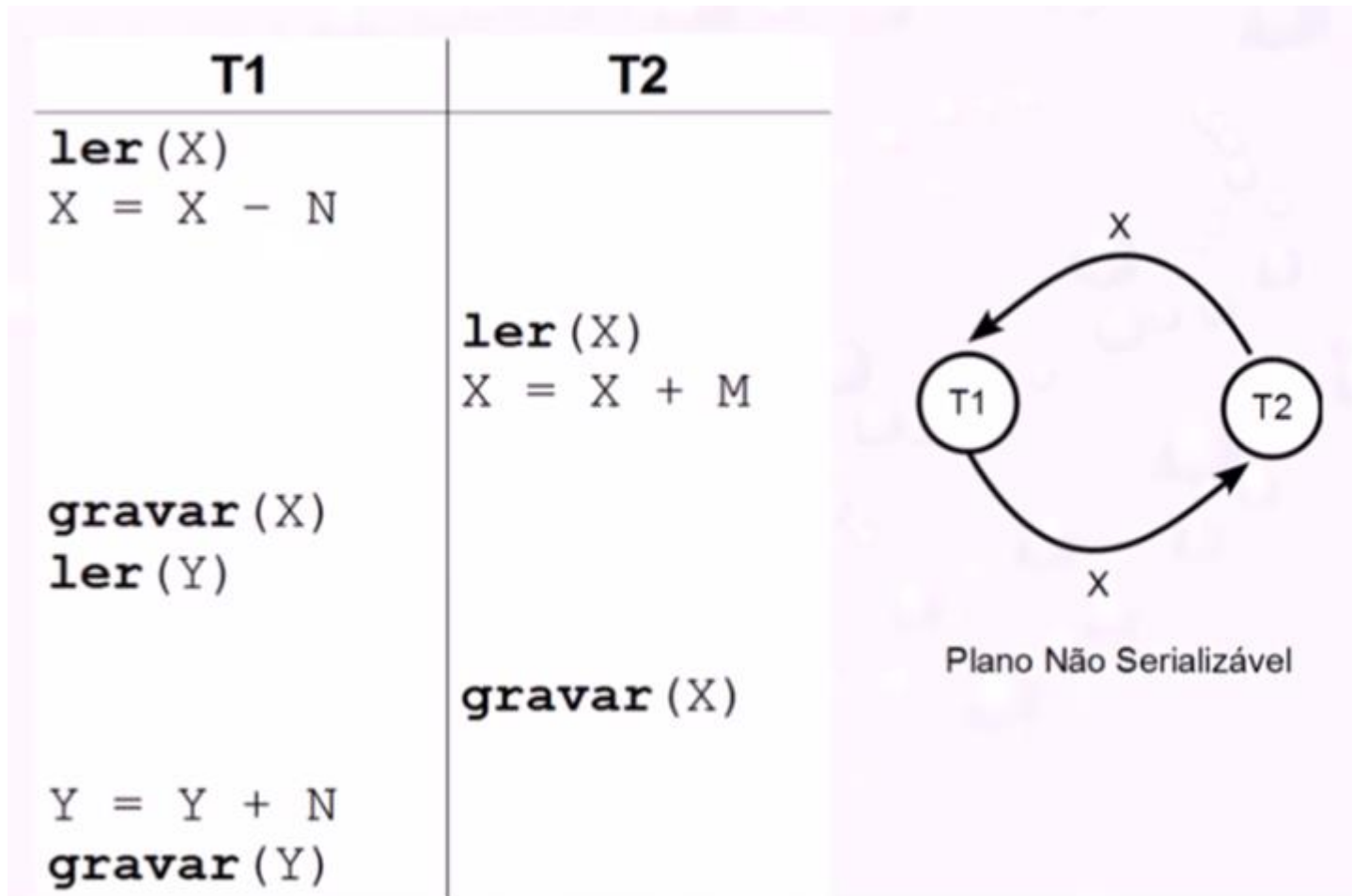
## Schedule Serial 2



**Obs.** O grafo não tem ciclos, portanto o schedule é serializável !



# Schedule 3

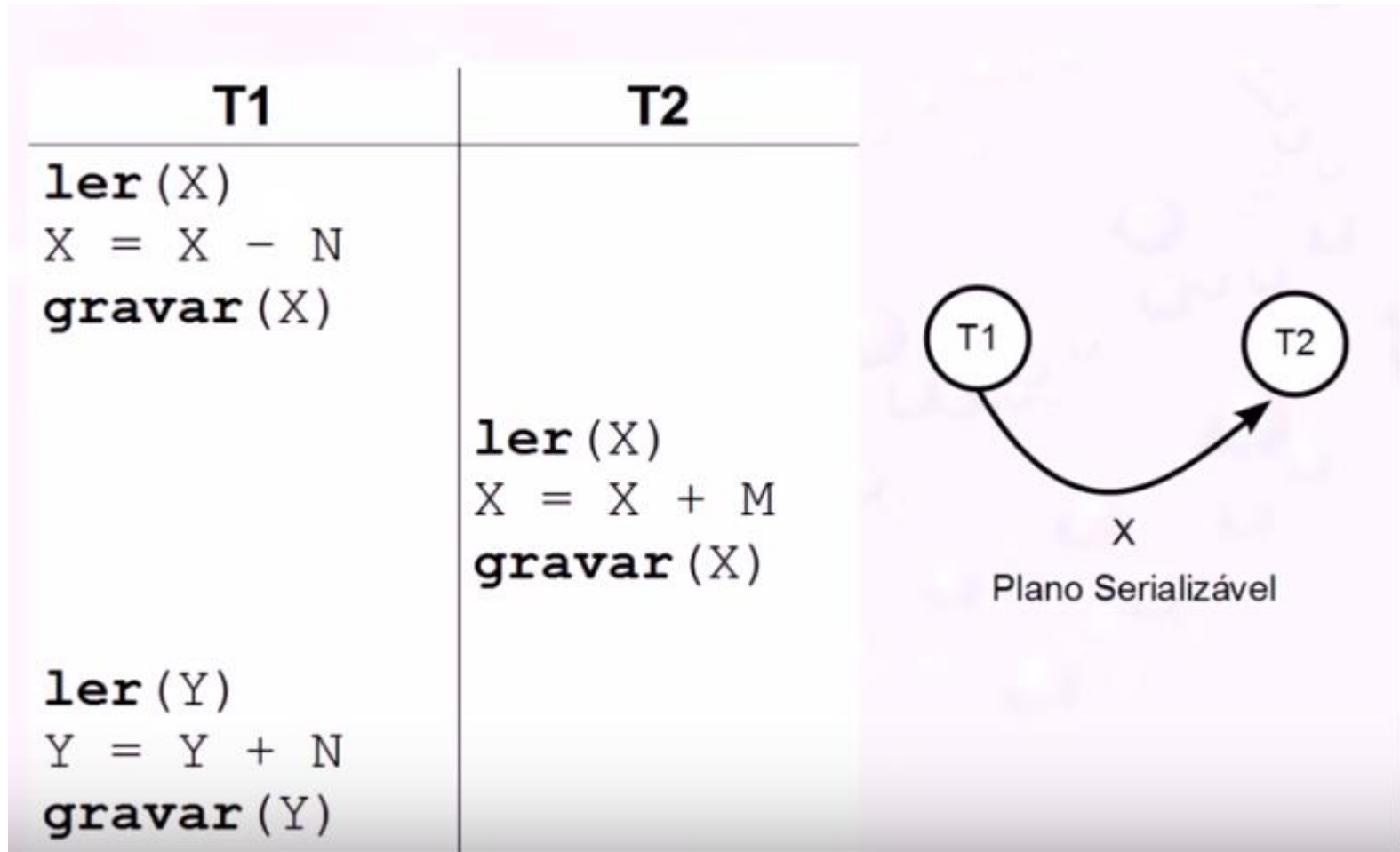


Obs. O grafo tem ciclos, portanto o schedule é **NÃO serializável** !





# Schedule 4



**Obs.** O grafo não tem ciclos, portanto o schedule é serializável !



## Serialização usada no controle de Concorrência

- ⊕ Vimos que, dizer que um schedule **S** é serializável (de conflito) – ou seja, **S** é equivalente (em conflito) a um schedule Serial – é equivalente a dizer que **S** está correto;
- ⊕ **Contudo, ser serializável é diferente de ser serial;**
- ⊕ Um schedule serial representa um processamento ineficiente, pois nenhuma intercalação de operações diferentes é permitida;
- ⊕ Um schedule serializável oferece os benefícios da execução concorrente sem abrir mão de qualquer exatidão.