Introdução aos conceitos e teoria de processamento de transações

André Luís Schwerz
Rafael Liberato Roberto

Tópicos

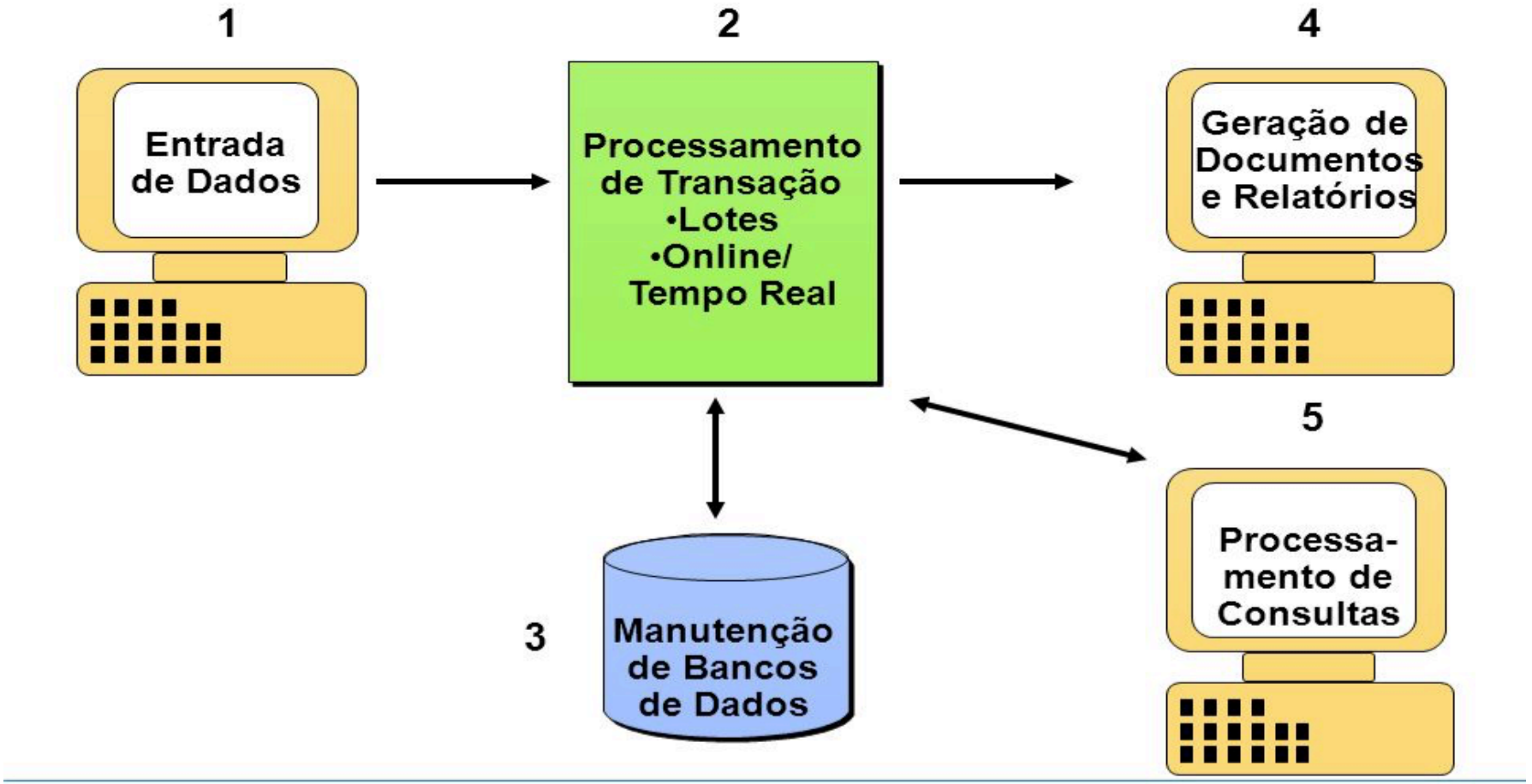
- Introdução ao processamento de transações
- Transações, itens de dados operações e buffers
- Conceitos de transação e operações adicionais
- O Log do Sistema
- Propriedades Transacionais
- Schedules seriais, não seriais e serializáveis por conflito
- Testando a serialização por conflito em schedule

Introdução ao processamento de transações

Sistemas de processamento de transação

- Definição:
  - Sistemas com grandes bancos de dados
  - Centenas de usuários simultâneos que executam transações de banco de dados
  - Exigem
    - Alta disponibilidade
    - Tempo de resposta rápido para centenas de usuários
- Exemplos:
  - Reservas de passagens aéreas
  - Sistemas bancários
  - Processamento de cartão de crédito
  - Compras on-line
  - Mercados de ações
  - Caixas de supermercados

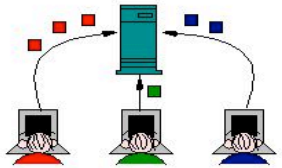
Sistemas de processamento de transação



Sistemas de Monousuários vs Multiusuário

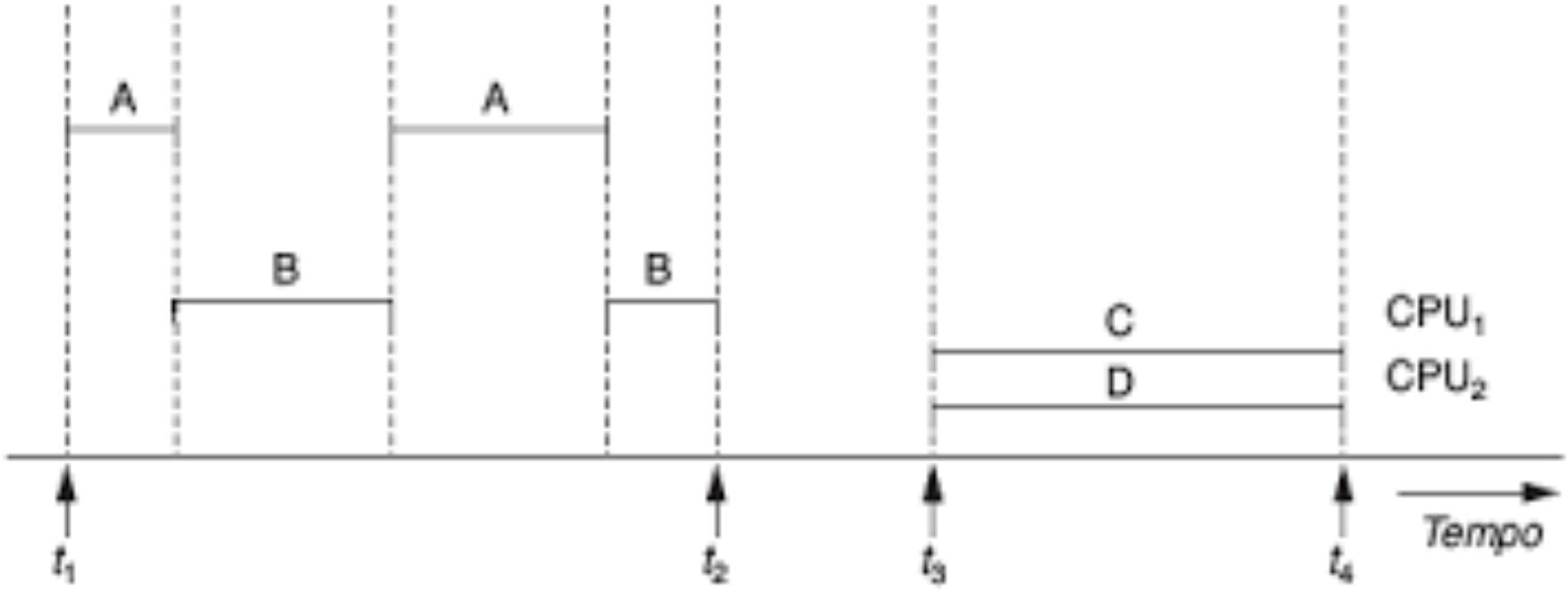
- Monousuários
  - Apenas um usuário por vez pode utilizar o sistema.
- Multiusuário
  - Muitos usuários podem acessar o banco de dados simultaneamente.

Monousuário **x** **Multiusuário**



Multiprogramação

- Multiprogramação
  - O sistema operacional executa vários programas (processos) ao mesmo tempo.



Processamento intercalado vs Processamento paralelo

Transações, itens de dados operações e buffers

Transação

- O que é uma Transação?
  - É um programa em execução que forma uma unidade logica de processamento de banco de dados.

- Uma transação pode incluir uma ou mais operações de acesso ao banco de dados:
  - Inserção, exclusão, modificação ou recuperação
  - Pode também ser embutida dentro de um programa

- Limites de uma transação devem ser explícitos:
  - `Begin transaction`
  - `End transaction`

- Um programa podem conter várias transações separadas pelos limites `begin_transaction e end_transaction.`

Item de Dados

- Um banco de dados é formado por uma coleção de **itens de dados**.
- **Granularidade** de um item de dado pode ser:
  - Um atributo
  - Uma tupla
  - Um bloco de disco
- Conceitos de transações são apresentados independentemente da granularidade**.**

Operações sobre os itens de dados

- Há duas operações básicas de acesso ao banco de dados.
- Mais baixo nível nível semântico.
- **`read_item(x):`**
  - Lê um item do banco de dados chamado X para uma variável do programa.
- **`write_item(x):`**
  - Grava o valor da variável de programa X no item de banco de dados chamado X.

Operação de leitura

- **`read_item(X)` inclui as seguintes etapas:**

1. Encontrar o endereço do bloco de disco que contém o item X.
2. Copie esse bloco de disco para um buffer na memória principal (caso já não esteja).
3. Copie o item X do buffer para a variável de programa chamada X.

Operação de escrita

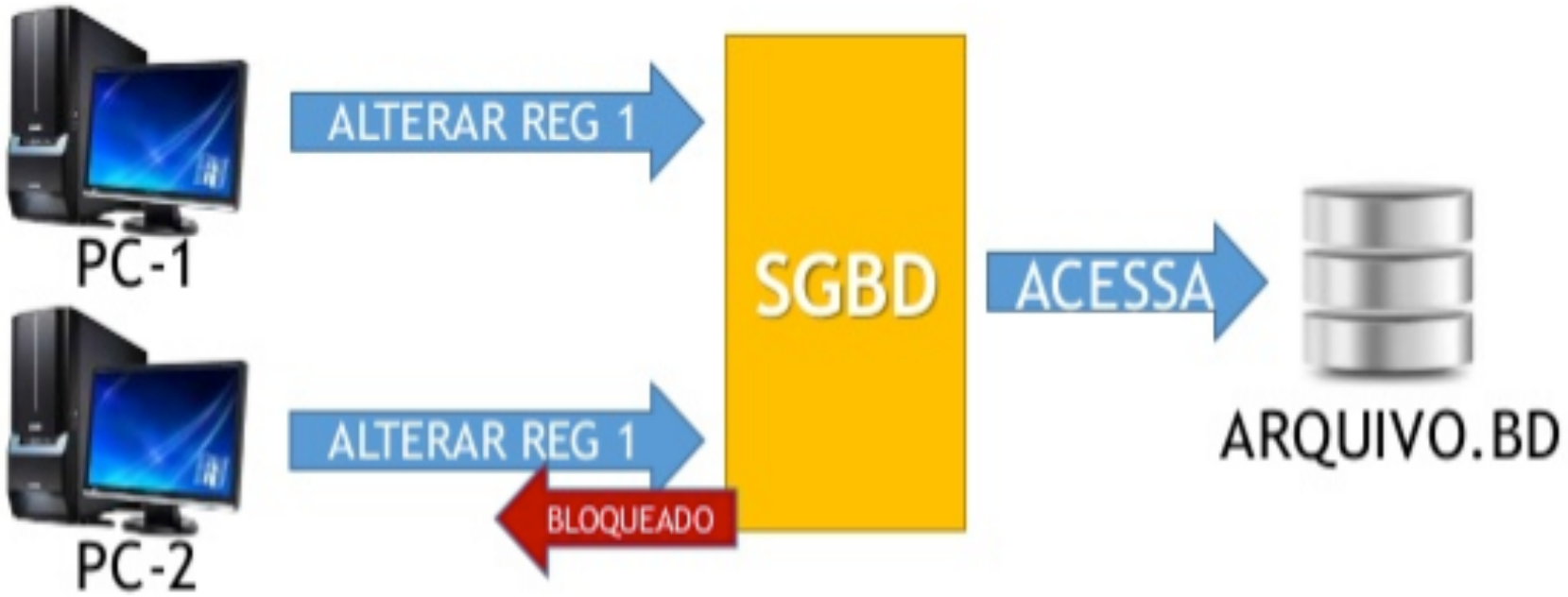
- **`write_item(X)` inclui as seguintes etapas:**

1. Encontrar o endereço do bloco de disco que contém o item X.
2. Copie esse bloco de disco para um buffer na memória principal (Caso já não esteja).
3. Copie o item X da variável de programa chamada X para o local correto no buffer.
4. Armazene o bloco atualizado do buffer de volta no disco.

Controle de Concorrência

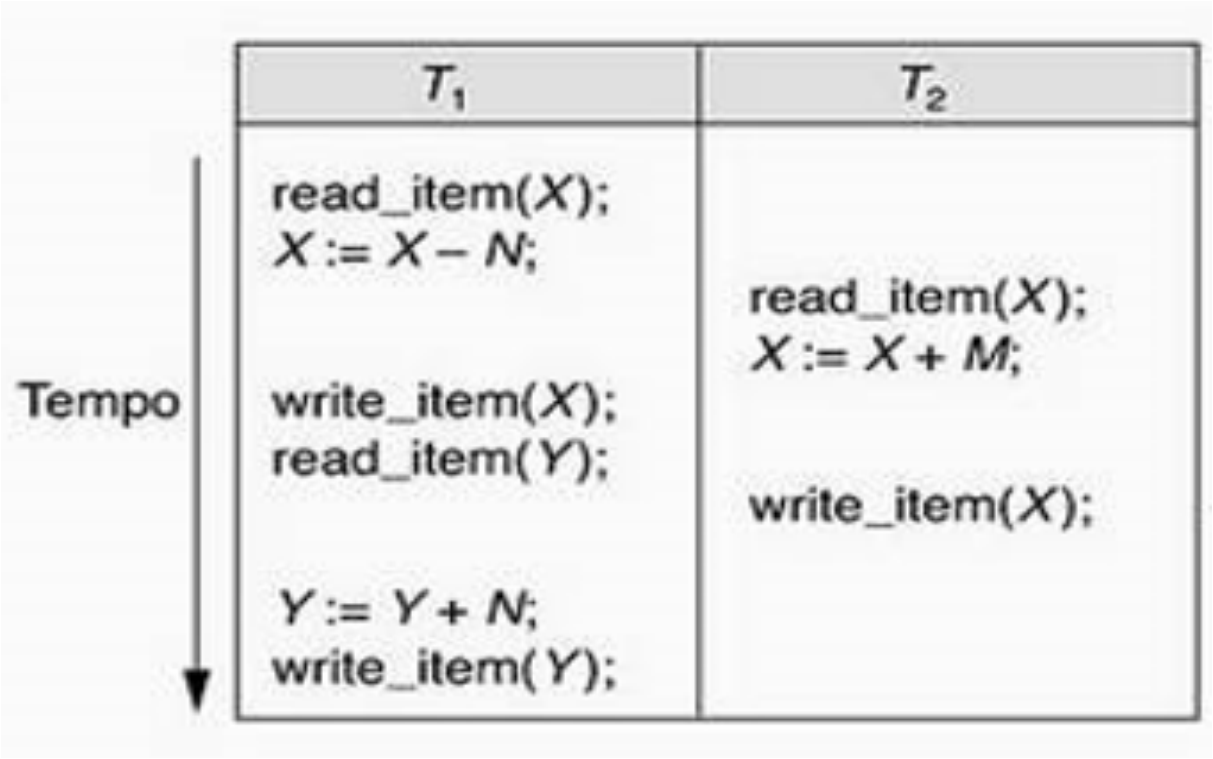
Controle de concorrência

- **Por que o controle de concorrência é necessário?**



Controle de concorrência – Anomalias

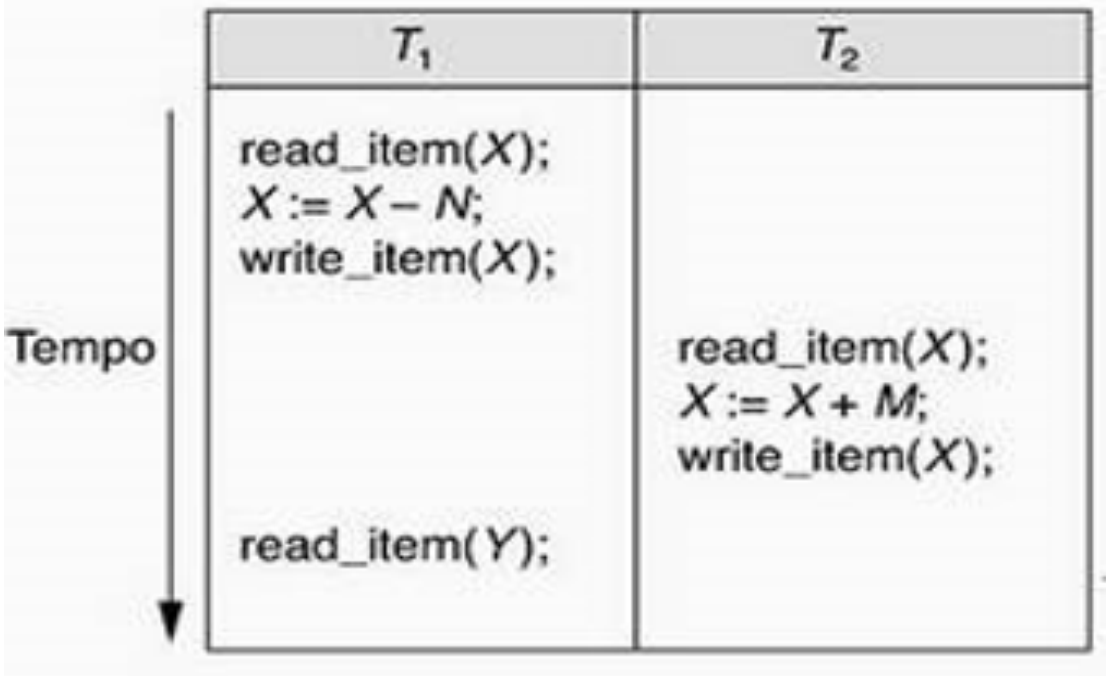
- Problema de atualização perdida
  - Esse problema ocorre quando duas transações que acessam os mesmos itens de dados têm suas operações intercaladas de modo que isso torna o valor de alguns itens do dados incorreto.



Item X tem um valor incorreto porque sua atualização de T_1 é perdida (sobrescrita)

Controle de concorrência – Anomalias

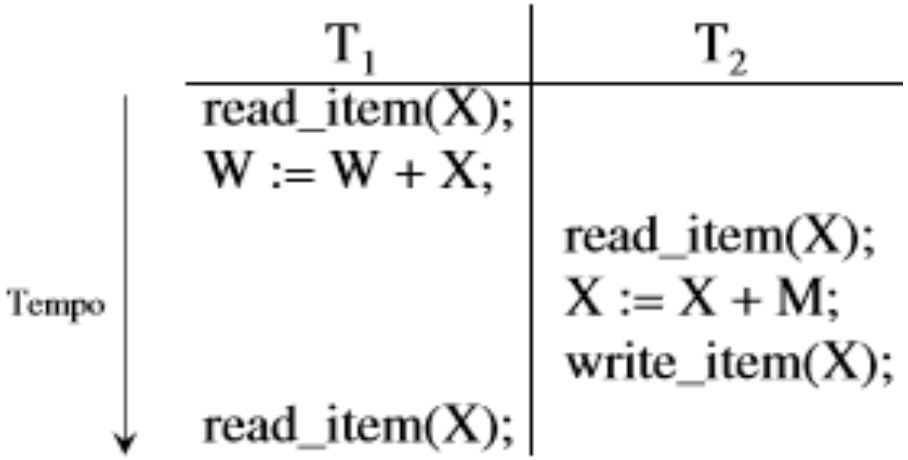
- Problema de atualização temporária
  - Esse problema ocorre quando uma transação atualiza um item de dado e depois a transação falha por algum motivo.
  - Problema também conhecido como leitura suja.



Transação T_1 falha e precisa mudar o valor de x de volta a seu valor antigo; enquanto isso, T_2 leu o valor temporário e incorreto de X.

Controle de concorrência – Anomalias

- Problema da leitura não repetitiva
  - Uma transação T lê o mesmo item duas vezes e o item é alterado por outra transação T_1 entre as duas leituras.
  - Logo, T recebe dois valores diferentes para suas duas leituras do mesmo item.



Controle de concorrência – Anomalias

- Problema do resumo incorreto

T_1	T_3

	sum := 0;
	read_item(*A*);
	sum := *sum* + *A*;
	. . .

read_item(*X*);
X := *X* – *N*;
write_item(*X*);	

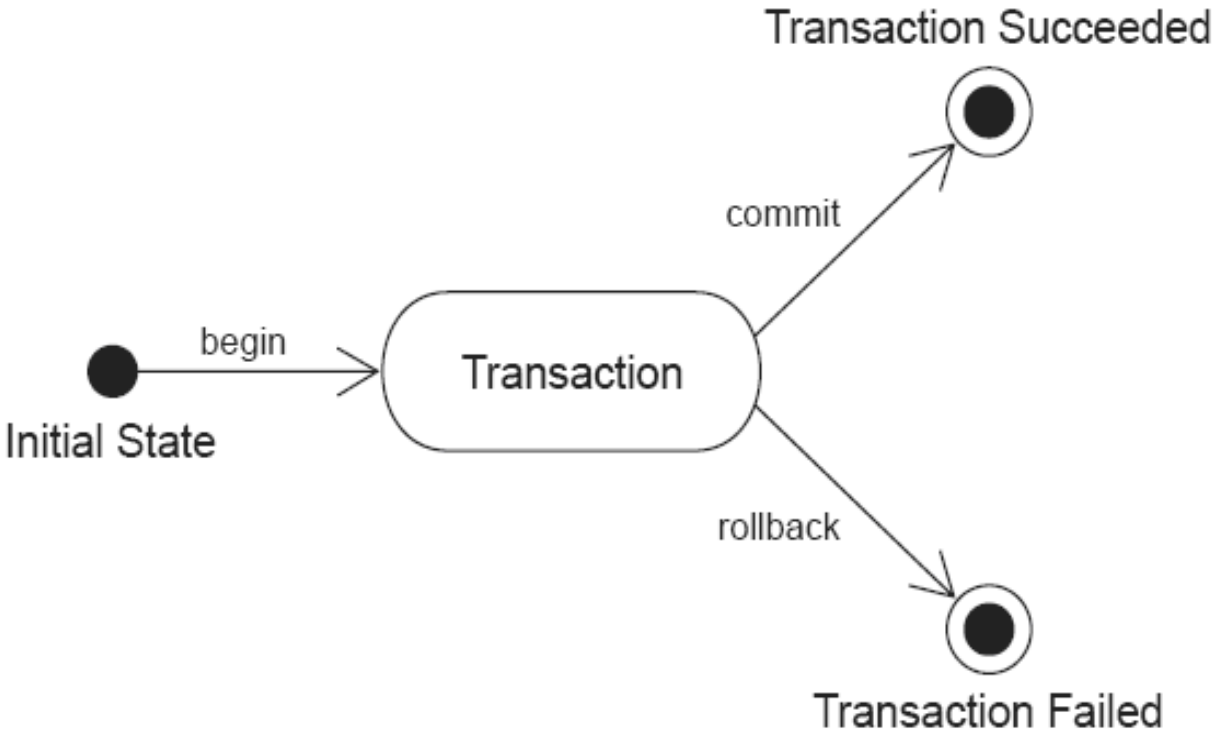
	read_item(*X*);
	sum := *sum* + *X*;
	read_item(*Y*);
	sum := *sum* + *Y*;

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

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

Controle de Concorrência

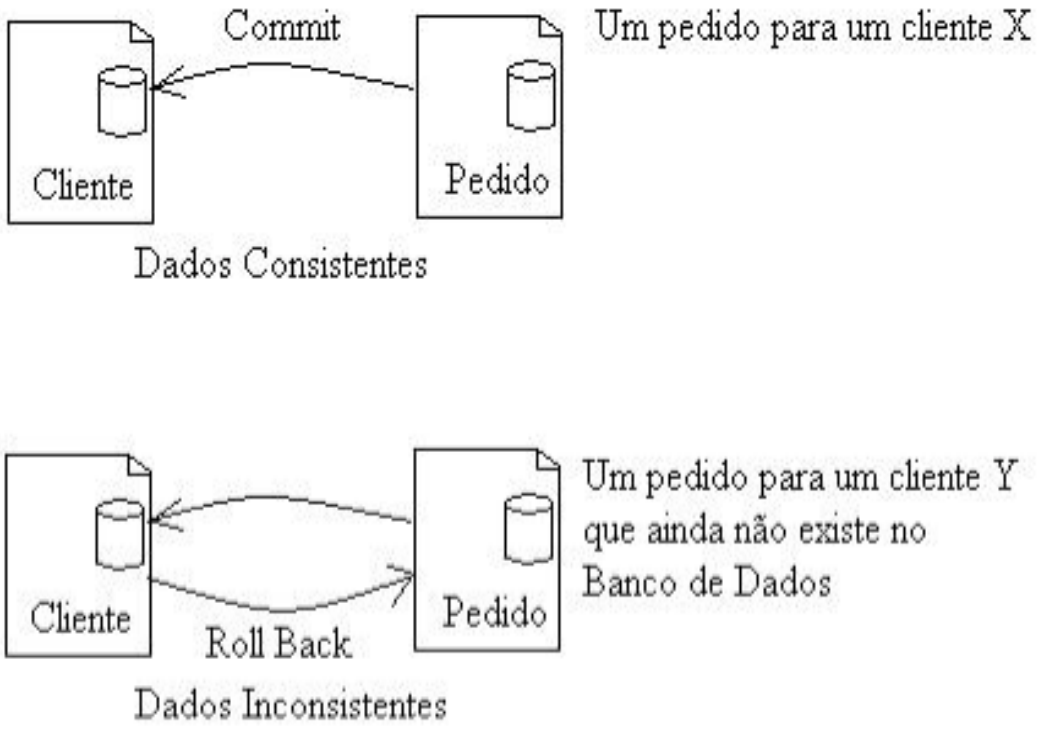
- Regra de Ouro:
  - O SGBD NÃO deve permitir que algumas operações de uma transação T sejam aplicadas ao banco de dados enquanto que outras operações de T não o são, pois a transação inteira é uma unidade logica de processamento de banco de dados.




```
Ti: read(A);
    A := A – 50;
    write(A);
    read(B);
    B := B + 50;
    write(B);
```

Recuperação

- Por que a recuperação é necessária?



Recuperação – Falhas

- Possíveis motivos para uma falha:
  - Falha do computador:
    - Erro de hardware, software ou de rede.
  - Erro de transação ou do sistema:
    - Erro em operações, como estouro de inteiros ou divisões por zero.
    - Interrupção da execução pelo usuário.
  - Erros locais ou condições de execução detectadas pela transação:
    - Durante a execução podem ocorrer certas condições que necessitam cancelar a transação.
    - Como por exemplo saldo insuficiente para saque em uma conta bancária.

Recuperação – Falhas

- Possíveis motivos para uma falha:
  - Imposição de controle de concorrência:
    - Abortar por motivo de violação da serialização, ou para resolver um estado de *deadlock*.
  - Falha de Disco:
    - Alguns blocos podem perder seus dados devido a um erro de leitura ou escrita.
  - Problemas físicos e catástrofes:
    - Se refere a uma lista sem fim de problemas que incluem falhas de energia, roubo, sabotagem, refrigeração e falhas em discos.

Conceitos de transação e operações adicionais

Conceitos de Transação

- Para fins de recuperação, o sistema precisa registrar quando cada transação começa, termina e confirma ou aborta. Portanto o gerenciador de recuperação SGBD precisa acompanhar as seguintes operações:

  - **BEGIN_TRANSACTION**
    - Marca o inicio da transação ou execução.
  - **READ ou WRITE**
    - Especificam operações de leitura ou escrita.
  - **END_TRANSACTION**
    - Especifica que operações READ ou WRITE terminaram e marca o final da transação, e pode ser necessário também verificar se as mudanças podem ser aplicadas permanentemente ou se precisam ser abortadas.

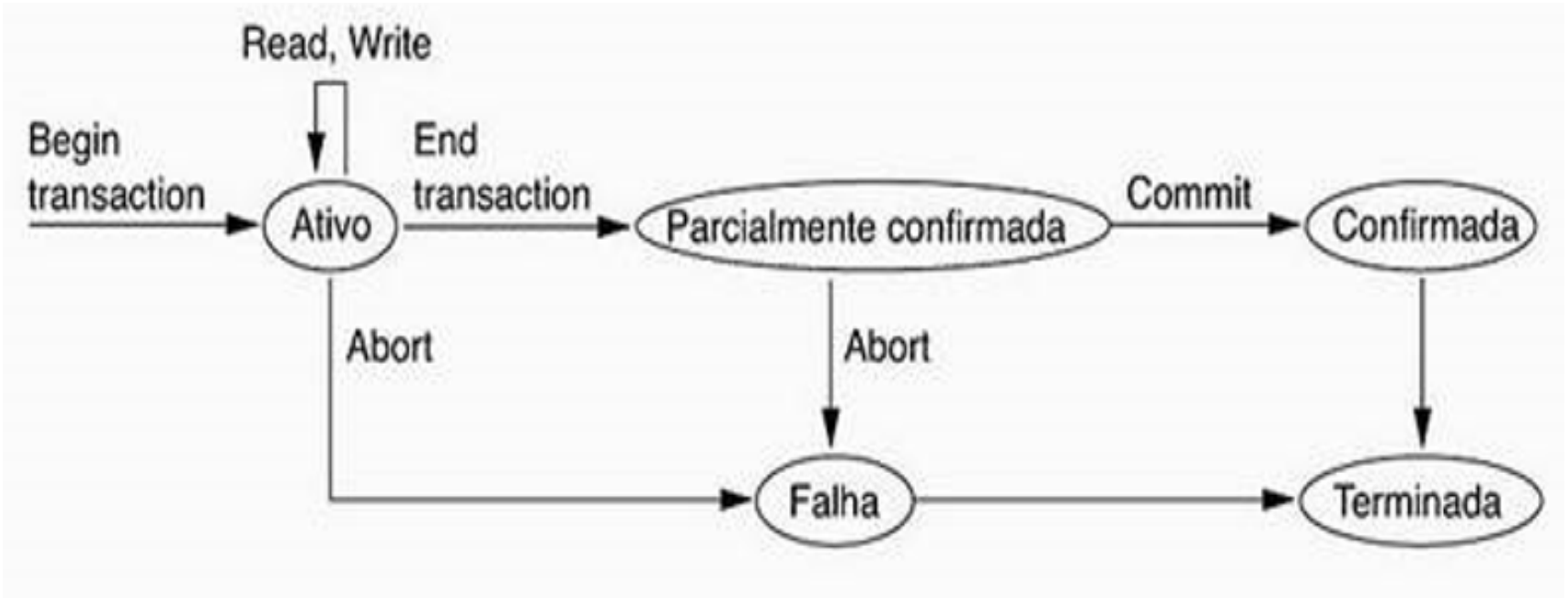
Conceitos de Transação

- Para fins de recuperação, o sistema precisa registrar quando cada transação começa, termina e confirma ou aborta. Portanto o gerenciador de recuperação SGBD precisa acompanhar as seguintes operações:

  - **COMMIT_TRANSACTION**
    - Sinaliza um final bem sucedido da transação.
  - **ROLLBACK (ou ABORT)**
    - Sinaliza que a transação foi encerrada sem sucesso, e que qualquer mudança feita no banco deve ser desfeita.

Conceitos de Transação

- Estado de transações



O Log do Sistema

Registros de log

- `[start_transaction, T]`
  - Indica que a transação T iniciou sua execução
- `[write_item, T, X, valor_antigo, valor_novo]`
  - Indica que a transição T mudou o valor do item X do banco de dados de valor_antigo para valor_novo
- `[read_item, T, X]`
  - Indica que a transação T leu o valor do item X no banco de dados
- `[commit, T]`
  - Indica que a transição T foi concluída com sucesso, e afirma que seu efeito pode ser confirmada no banco de dados
- `[abort, T]`
  - Indica que a transação T foi abortada

Ponto de confirmação (*commit*)

- Uma transação T alcança seu ponto de confirmação quando todas as suas operações que acessam o banco de dados tiverem sido executadas com sucesso e o efeito de todas as operações tiverem sido registradas no log.

Propriedades Transacionais

Propriedades ACID

- **Atomicidade**
  - Deve ser realizada em sua totalidade ou não ser realizada de forma alguma.
- **Consistência**
  - Deve levar o banco de dados de um estado consistente para outro estado consistente, sendo executado do início ao fim sem interferências de outras transações.
- **Isolamento**
  - As transação não deve ser interferida por quaisquer outras transações que aconteçam simultaneamente.
- **Durabilidade**
  - Mudanças aplicadas ao banco de dados pela transação confirmada precisam persistirem no banco de dados, não podendo serem perdidas por falhas.

Schedules

Schedules de transações

- Quando as transações estão executando simultaneamente em um padrão intercalado, então a ordem da execução das operações de todas as diversas transações é conhecida como um **schedule** (ou **histórico**)

- Um schedule *S* de *n* transações T_1, T_2, ..., T_n é uma ordenação das operações das transações.
  - 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

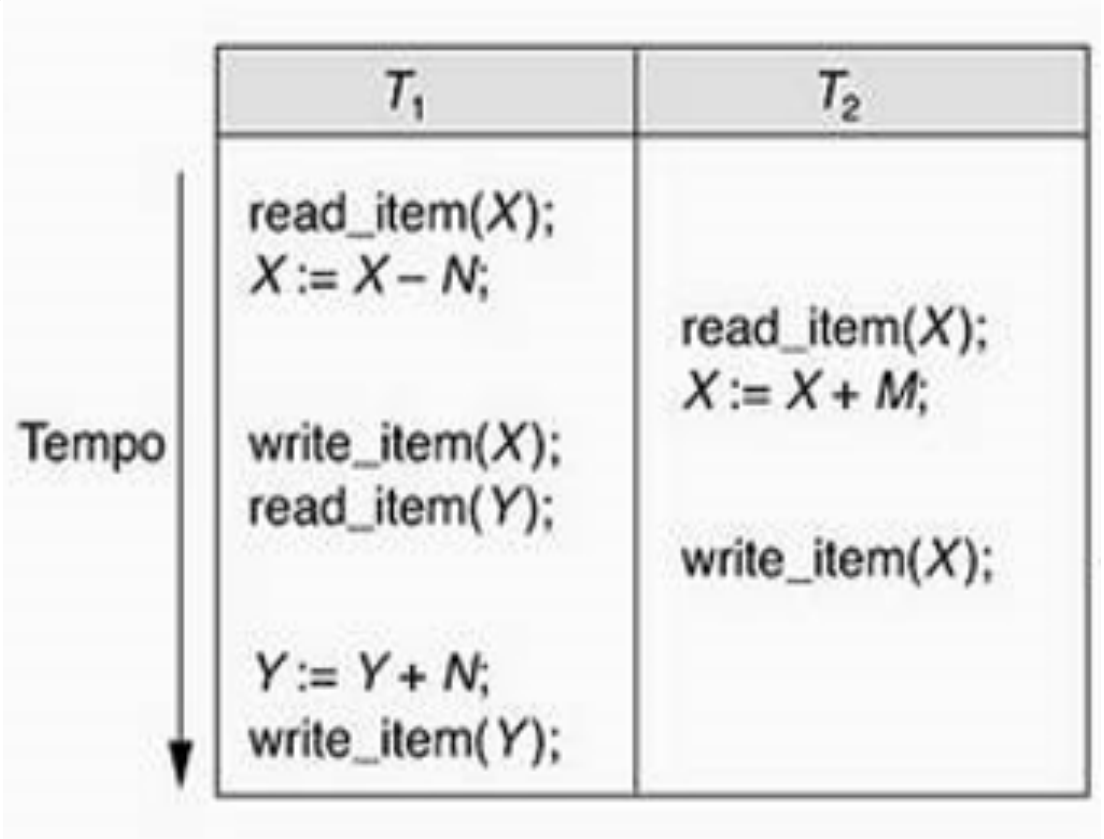
Schedules de transações

- Uma notação abreviada:

Sigla	Operação
b	begin_transaction
r	read_item
w	write_item
e	end_transaction
c	Commit
A	Abort

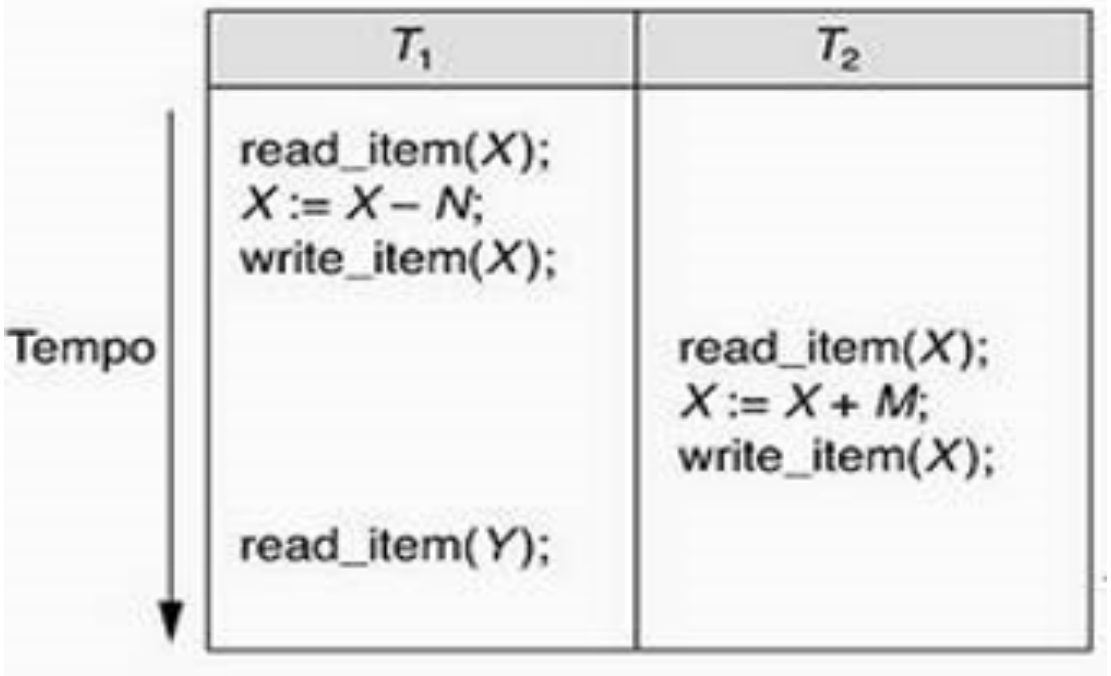
Schedules de transações

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



Schedules de transações

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



Operações em conflito

- Duas operações em schedule são consideradas em **conflito** se satisfazem a todas as três condições a seguir:

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

	$\mathbf{read_j(x)}$	$\mathbf{write_j(x)}$
$\mathbf{read_i(x)}$	false	true
$\mathbf{write_i(x)}$	true	true

Operações em conflito

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

Intuitivamente duas operações estão em conflito se a alteração de sua ordem resultar em resultados diferentes

Operações em conflito

Conflito de leitura-gravação

$r_1(X);\ w_2(X)$

$w_2(X);\ r_1(X)$

Operações em conflito

Conflito de gravação-gravação

$w_1(X);\ w_2(X)$

$w_2(X);\ w_1(X)$

Schedule completo

- Um schedule *S* é dito ser **completo** se:
  - As operações em *S* são exatamente aquelas operações em T_1, T_2, ..., T_n, incluindo uma operação de confirmação ou aborto com última operação em cada transação no schedule;
  - Para qualquer par de operações da mesma transação T_i, sua ordem de aparecimento relativa em *S* é a mesma que sua ordem de aparecimento em T_i;
  - Para duas operações quaisquer em conflito, uma das duas precisa ocorrer antes da outra no schedule.

Schedules recuperáveis

Schedules recuperáveis

Um schedule S é recuperável se nenhuma transação T em S for concluída até que todas as transações T’ que gravaram dados lidos por T tenham sido concluídas.

Schedules recuperáveis

Schedules recuperáveis

Não Recuperável

T1	T2
read(A)	
A = A – 20	
write(A)	
	read(A)
	A = A + 10
	write(A)
	commit( )
abort( )	

Recuperável

T1	T2
read(A)	
A = A – 20	
write(A)	
	read(A)
	A = A + 10
	write(A)
commit( )	
	commit( )

Schedules recuperáveis

Rollback em cascata

Em um escalonamento recuperável, pode ocorrer um fenômeno conhecido como *rollback* em cascata, no qual uma transação não-confirmada tenha que ser desfeita porque leu um item de uma transação que falhou.

Schedules recuperáveis

Rollback em cascata

T1	T2
read(A)	
A = A – 20	
write(A)	
	read(A)
	A = A + 10
	write(A)
abort( )	. . .

Schedules recuperáveis

Schedule sem cascata

Um schedule S é recuperável e evita aborto em cascata se uma T_i em S só puder ler dados que tenham sido atualizados por transações que já concluíram.

Schedules recuperáveis

Schedule sem cascata

T1	T2
read(A)	
A = A – 20	
write(A)	
commit( )	
	read(A)
	A = A + 10
	write(A)
	. . .

Schedules recuperáveis

Schedule estrito

Transações não podem ler nem gravar um item X até que a última transação que gravou X tenha sido confirmada (ou cancelada)

Schedules recuperáveis

Schedule estrito

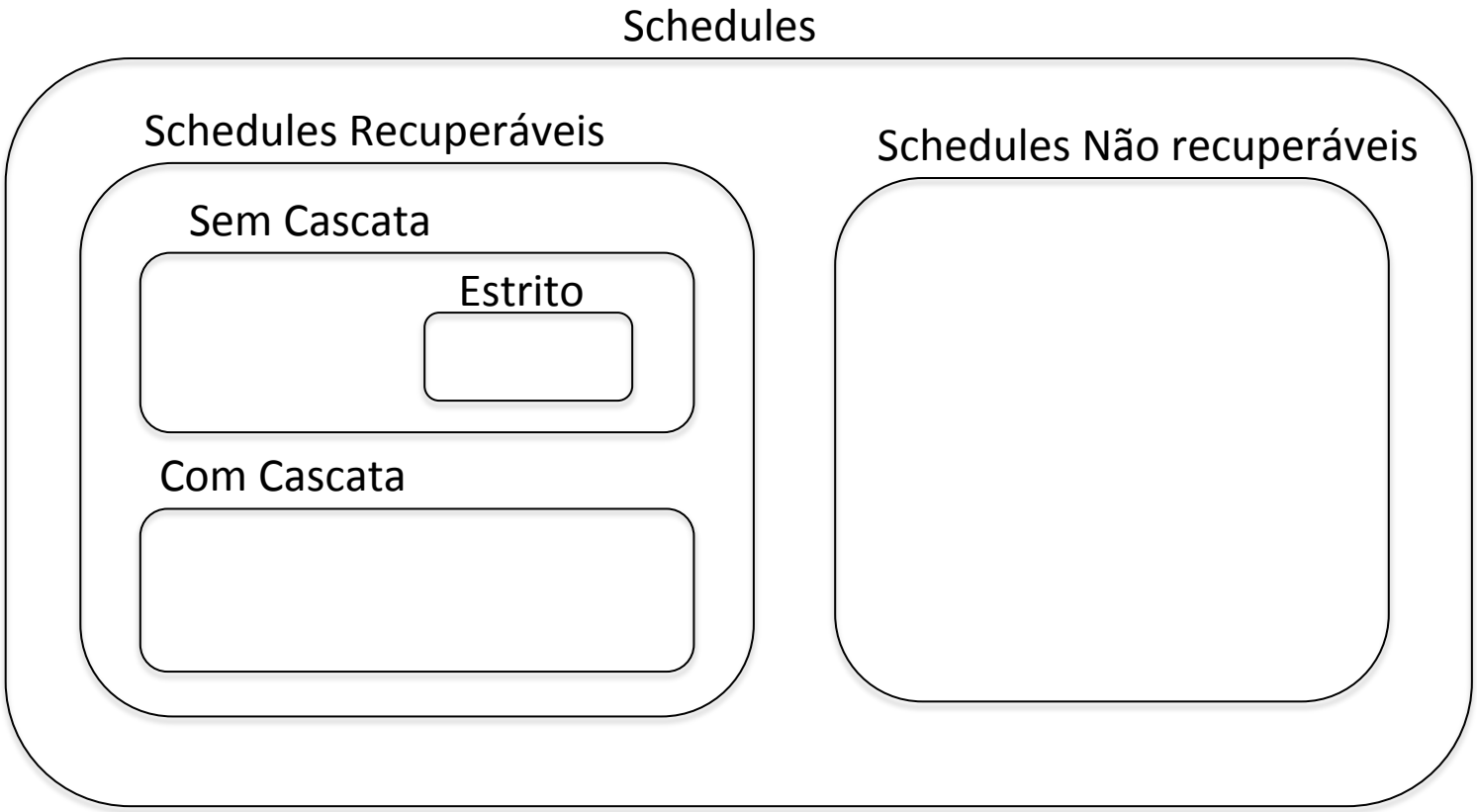
Não Estrito

T1	T2
read(A)	
A = A – 20	
write(A)	
	read(B)
	A = B + 10
	write(A)
	commit( )
abort( )	

Estrito

T1	T2
read(A)	
A = A – 20	
write(A)	
commit( )	
	read(B)
	A = B + 10
	write(A)
	commit( )

Resumo


Schedules
Schedules Recuperáveis
Schedules Não recuperáveis
Sem Cascata
Estrito
Com Cascata

Schedules seriais, não seriais e serializáveis por conflito

Schedules

- Até agora, caracterizamos schedules com base em suas propriedades de recuperação.

- Agora, caracterizaremos os tipos de schedules que sempre são considerados corretos quando transações concorrentes são executadas.

Schedule serial

Schedule serial

Uma schedule é chamada serial se as operações de cada transação forem executadas consecutivamente, sem quaisquer operações intercaladas.

Schedule serial

Schedule serial

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);

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);

Schedules não serial

Schedule não serial

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);	

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);	

Schedules serializáveis

Schedule serializáveis

Um schedule S é serializável se ele é equivalente à algum schedule serial com as mesmas *n* transações.

Exemplos

Schedule serializável

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);	

Schedule não serializável

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);	

O que é ser um schedule equivalente?

Schedule equivalentes

- Equivalentes no resultado.
- Equivalentes em conflito.

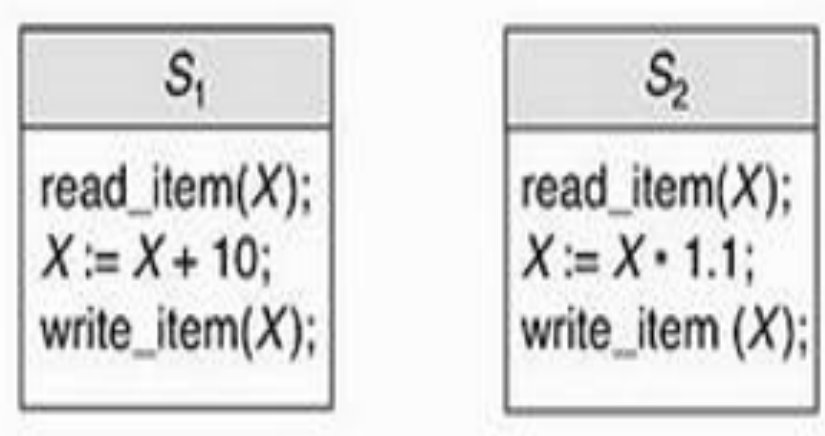
Schedules equivalentes

Schedules equivalentes no resultado

Dois schedules são chamados de equivalentes no resultado se eles produzem o mesmo estado final no banco de dados.

- Problema em ser equivalentes no resultado:

X = 100

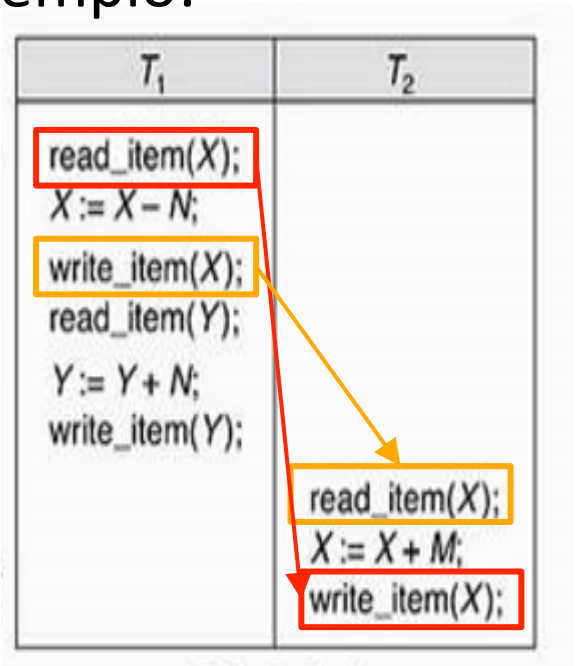


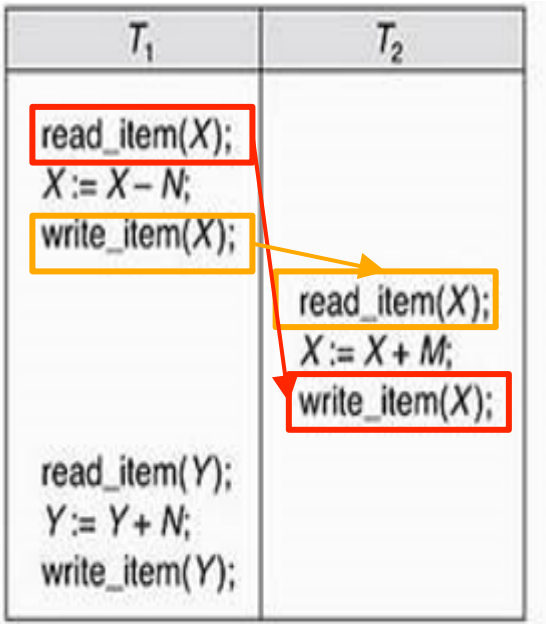
Schedules equivalentes

Schedules equivalentes em conflito

Dois schedules são ditos ser equivalentes em conflito se a ordem de quaisquer operações conflitantes é a mesma em ambos schedules.

- Exemplo:

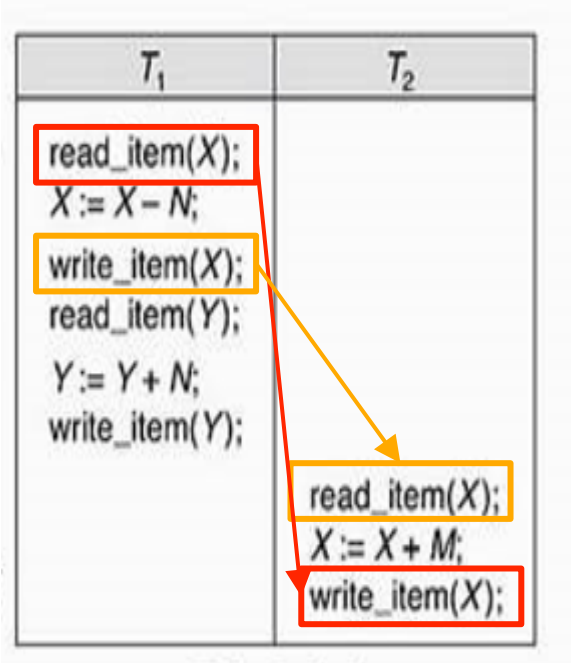


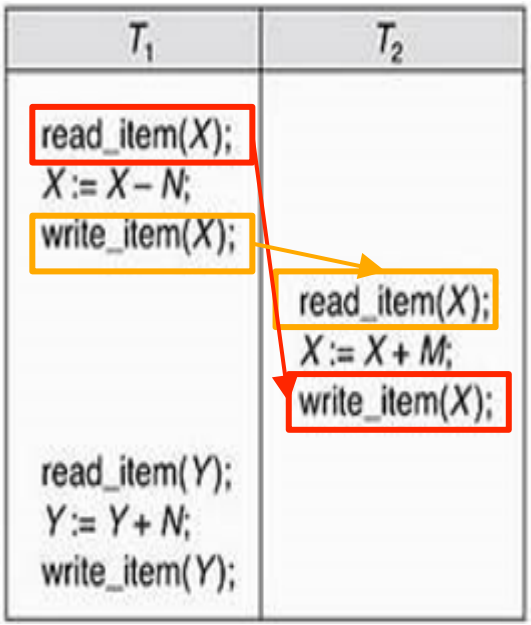


Schedules recuperáveis

Schedule equivalentes

- Equivalentes em conflito.





Schedule serializáveis em conflito

Schedule serializáveis

Um schedule S é serializável se ele é equivalente à algum schedule serial com as mesmas *n* transações.

Schedule serializáveis em conflito

Um schedule S é dito ser serializável em conflito se ele é equivalente em conflito com algum schedule serial S'.

Resumo

- Ser serializável não é o mesmo de ser serial
- Ser serializável implica que o schedule é a schedule correto.
  - Ele levará o banco de dados para um estado consistente.
  - A intercalação é apropriada e resultará em um estado como se as transações fossem executadas serialmente, ainda assim, alcançarão eficiência devido a execução concorrente.
- Verificar se um schedule é seriável é computacionalmente complexo.
  - A intercalação das operações ocorre no sistema operacional através de algum escalonador.
  - Difícil determinar de antemão como as operações em uma transação serão intercaladas.

Testando a serialização por conflito em schedule

Testando a serialização por conflito em schedule

Algoritmo para construir um grafo de precedência

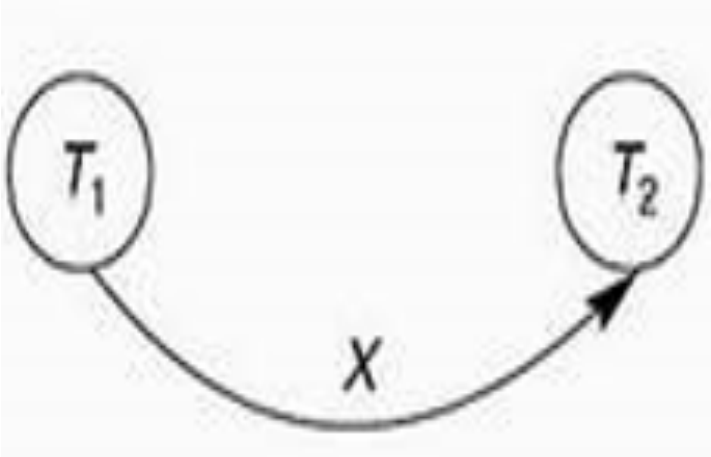
1. Para cada transação T_i participante no schedule S, crie um nó rotulado com T_i no grafo de precedência.
2. Para cada caso em S onde T_j executa um `read_item(X)` depois de T_i executar um `write_item(X)`, crie uma aresta (T_i → T_j) no grafo de precedência.
3. Para cada caso em S onde T_j executa um `write_item(X)` após T_i executar um `read_item(X)`, crie uma aresta (T_i → T_j) no grafo de precedência.
4. Para cada caso em S onde T_j executa um write_item(X) após T_i executar um `write_item(X)`, crie uma aresta (T_i → T_j) no grafo de precedência.
5. O schedule S é serializável se, e somente se, o grafo de precedência não tiver ciclos.

Testando a serialização por conflito em schedule

Algoritmo para construir um grafo de precedência

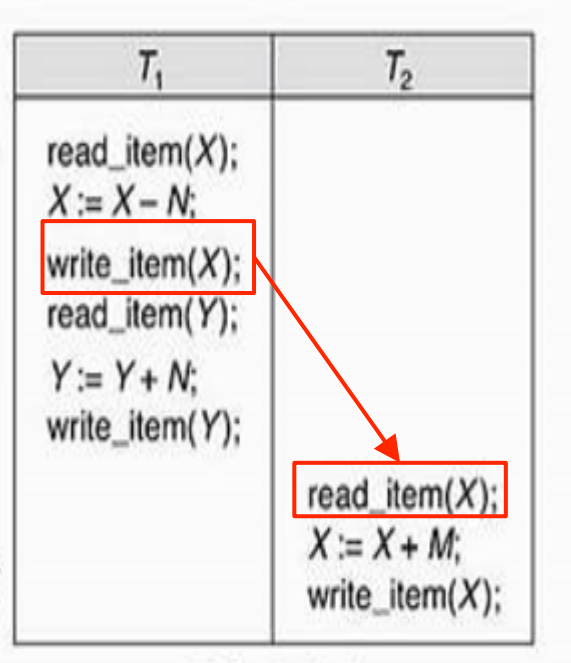
T_1	T_2
read_item(X); $X := X - N$;	
write_item(X); read_item(Y);	
$Y := Y + N$; write_item(Y);	

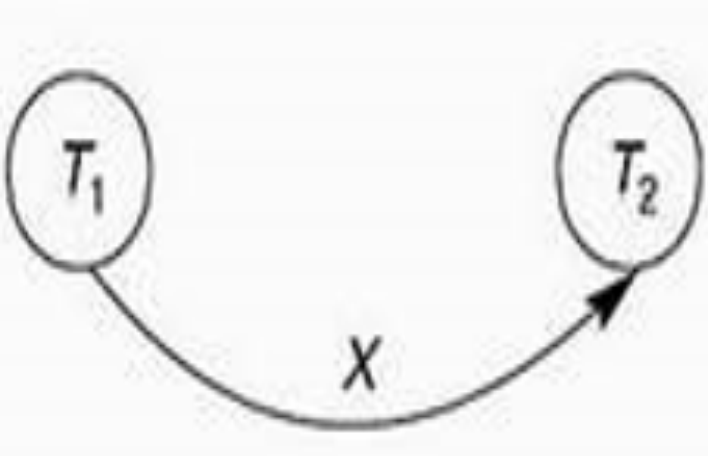
	read_item(X);
	$X := X + M$;
	write_item(X);



Testando a serialização por conflito em schedule

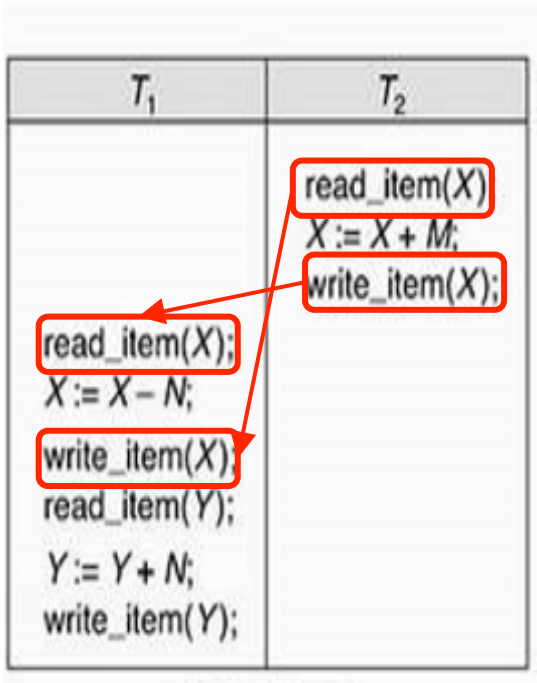
Algoritmo para construir um grafo de precedência

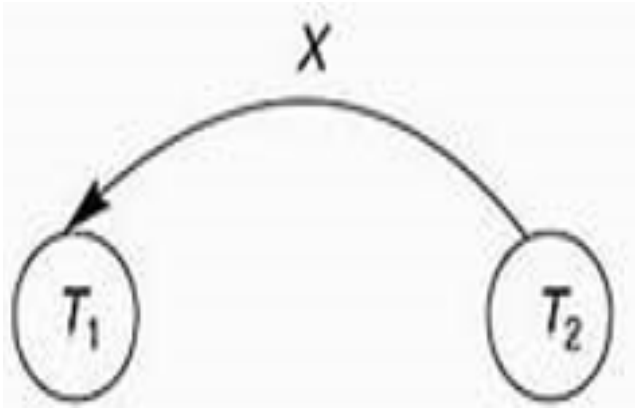




Testando a serialização por conflito em schedule

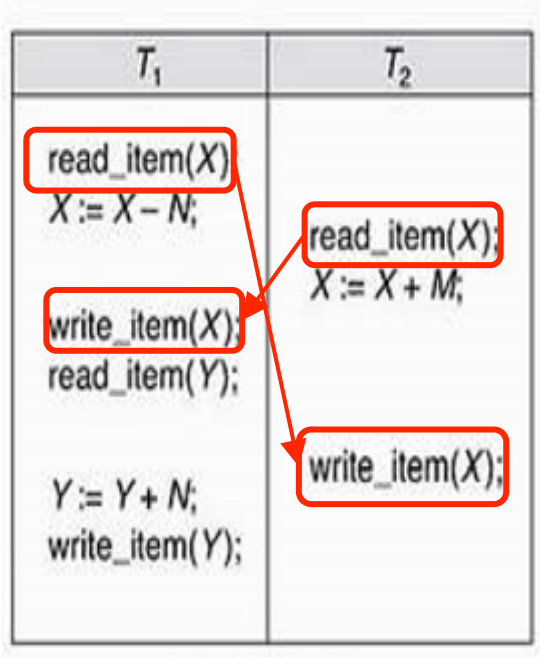
Algoritmo para construir um grafo de precedência

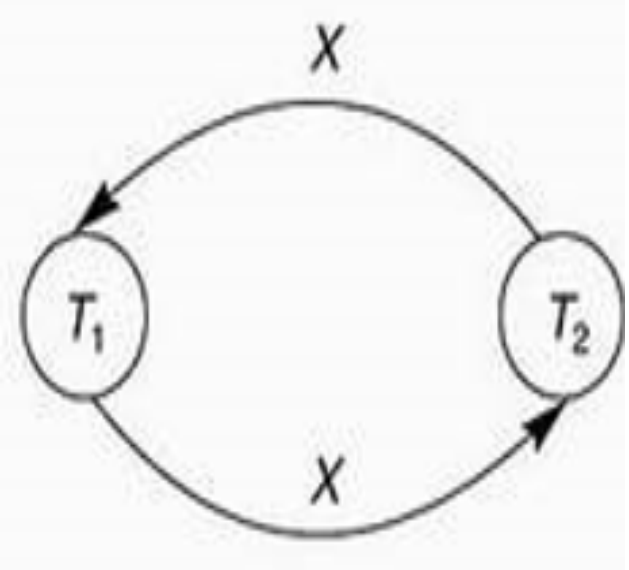




Testando a serialização por conflito em schedule

Algoritmo para construir um grafo de precedência

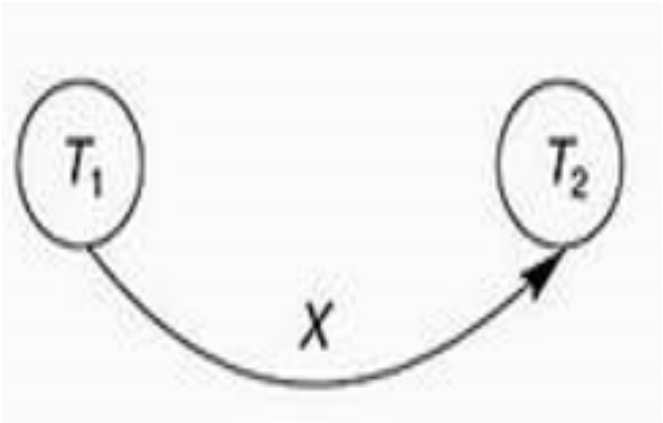




Testando a serialização por conflito em schedule

Algoritmo para construir um grafo de precedência

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);	



Exercícios

- Qual dos seguintes schedules é serializável? Para cada schedule serializável, determine os schedules seriais equivalentes.

a) $r_1(x);\ r_3(x);\ w_1(x);\ r_2(x);\ w_3(x)$

b) $r_1(x);\ r_3(x);\ w_3(x);\ w_1(x);\ r_2(x)$

c) $r_3(x);\ r_2(x);\ w_3(x);\ r_1(x);\ w_1(x)$

d) $r_3(x);\ r_2(x);\ r_1(x);\ w_3(x);\ w_1(x)$

Exercício

21.23. Considere as três transações T_1, T_2 e T_3, e os schedules S_1 e S_2 a seguir. Desenhe os grafos de serialização (precedência) para S_1 e S_2 e indique se cada schedule é serializável ou não. Se um schedule for serializável, escreva o(s) schedule(s) serial(is) equivalente(s).

T_1: $r_1(X)$; $r_1(Z)$; $w_1(X)$;

T_2: $r_2(Z)$; $r_2(Y)$; $w_2(Z)$; $w_2(Y)$;

T_3: $r_3(X)$; $r_3(Y)$; $w_3(Y)$;

S_1: $r_1(X)$; $r_2(Z)$; $r_1(Z)$; $r_3(X)$; $r_3(Y)$; $w_1(X)$; $w_3(Y)$; $r_2(Y)$; $w_2(Z)$; $w_2(Y)$;

S_2: $r_1(X)$; $r_2(Z)$; $r_3(X)$; $r_1(Z)$; $r_2(Y)$; $r_3(Y)$; $w_1(X)$; $w_2(Z)$; $w_3(Y)$; $w_2(Y)$;

Dúvidas