Ciência da Computação

GBC043 Sistemas de Banco de Dados

Processamento de Transações

Profa. Maria Camila Nardini Barioni

camila.barioni@ufu.br

Bloco B - sala 1B137

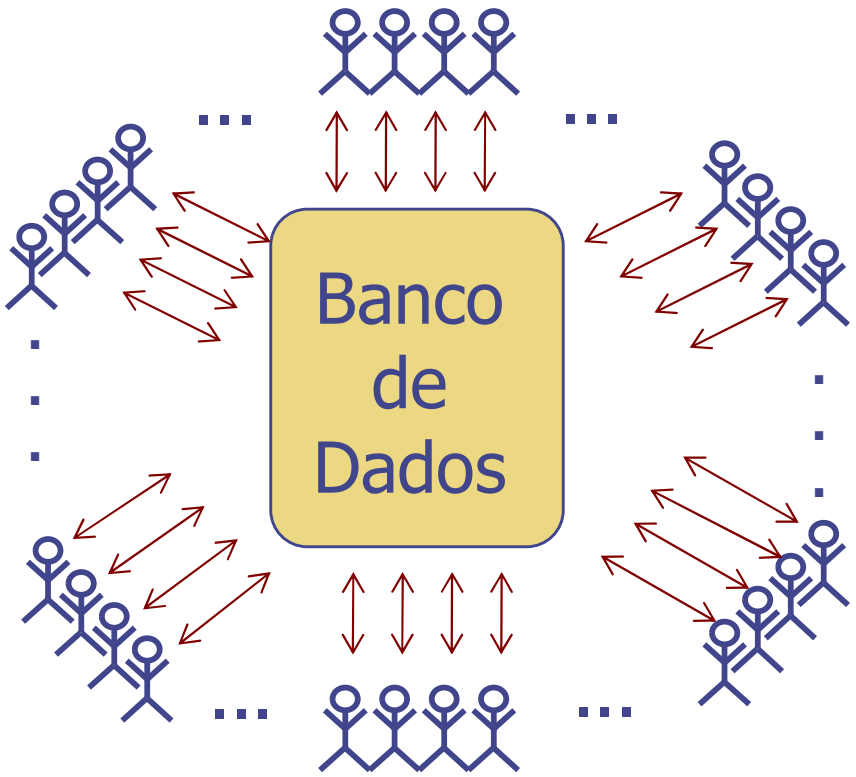
1° semestre de 2024

Roteiro Próximas aulas

- **Hoje :** Introdução aos conceitos de transações e laboratório de transações
- 24/10: vista da segunda prova (Prevista! A ser confirmada!)
- 24/10 e 31/10: Continuação da apresentação dos conceitos sobre transações
- 25/10 Aula reservada para finalização do projeto. Grupos irão se reunir para finalizar e entregar o projeto
- 01/11, 07/11, 08/11 e 14/11: Apresentações dos projetos
- 21/11: Prova Substitutiva (Toda a matéria!!!)
- 22/11: Vista final

Introdução

- SGBDs são em geral multi-usuários
  - processam ***simultaneamente*** operações disparadas por vários usuários
  - deseja-se **alta disponibilidade** e **tempo de resposta pequeno**



- Exemplos:
- Sistemas para reserva de passagens
- Banco
- Processamento de cartões de crédito
- Sistemas de compra coletiva
- etc.

Introdução

- Diversos usuários podem acessar o BD simultaneamente → conceito de multiprogramação
- Modelos de processamento
  - a) Processamento intercalado: enquanto um processo ***A*** faz I/O, outro processo ***B*** é selecionado para execução
  - B) Processamento paralelo

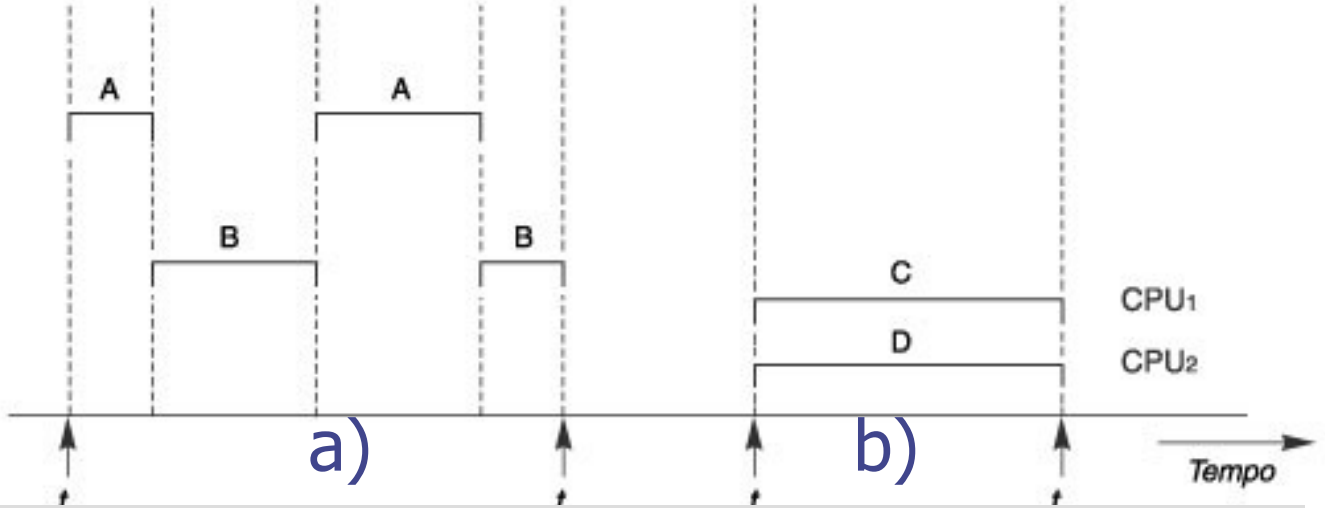


Figura 17.1 **Processamento intercalado *versus* processamento paralelo de transações concorrentes.**

Introdução

- Diversos usuários podem acessar o BD simultaneamente → conceito de multiprogramação
- Modelos de processamento
  - a) Processamento intercalado → teoria de controle de concorrência em BD
  - B) Processamento paralelo

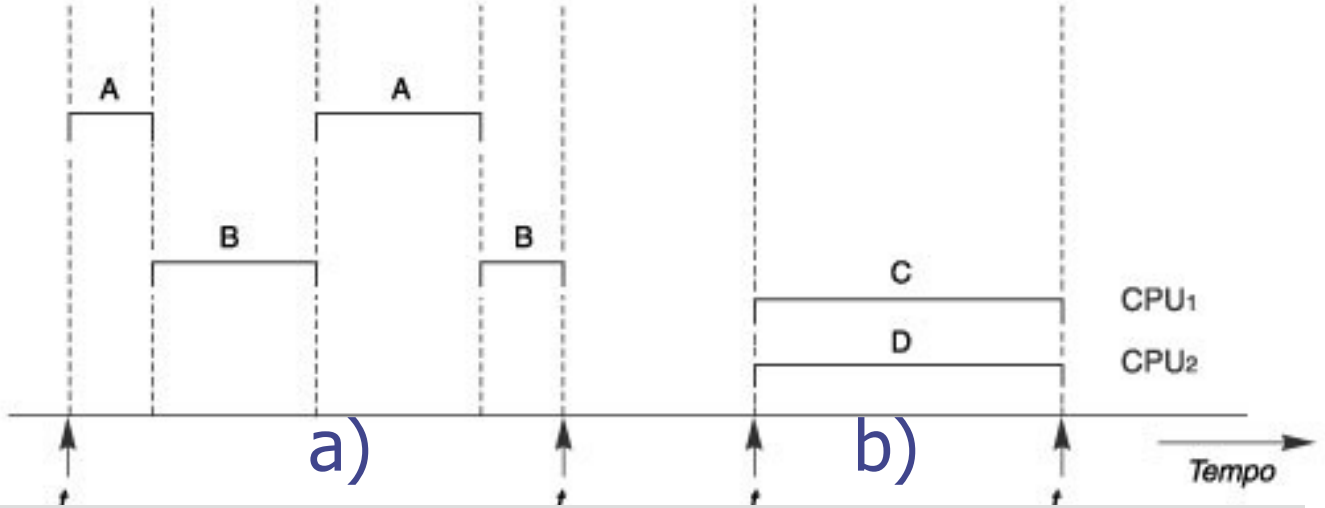


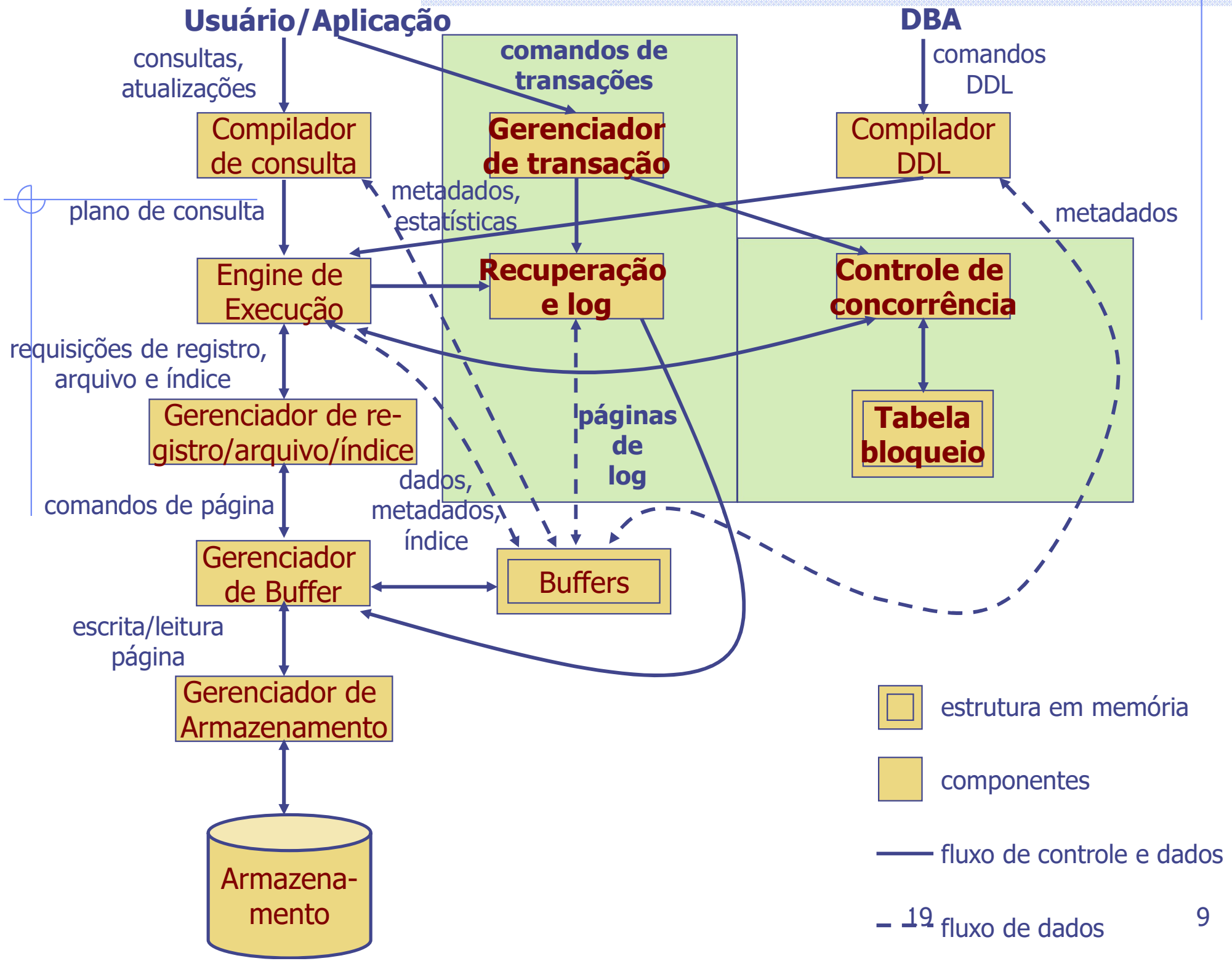
Figura 17.1 **Processamento intercalado *versus* processamento paralelo de transações concorrentes.**

Transações

- Transação
  - é uma unidade atômica de trabalho que é completada integralmente ou não é realizada
  - engloba operações de acesso ao BD, como: inserção, exclusão, alteração ou recuperação
  - as operações que formam uma transação podem ser embutidas em um programa de aplicação ou ser especificadas em uma linguagem como a SQL
  - declarações de **início** e **fim** são utilizados para delimitar uma transação

Transações

- Uma transação precisa ver um banco de dados consistente
  - Durante a execução da transação, o banco de dados pode ser temporariamente inconsistente
  - Quando a transação é completada com sucesso (é confirmada), o banco de dados precisa ser consistente
  - Após a confirmação da transação, as mudanças que ele faz no banco de dados persistem, mesmo se houver falhas de sistema
  - Várias transações podem ser executadas em paralelo
- Dois problemas principais para resolver:
  - Falhas de vários tipos, como falhas de hardware e falhas de sistema
  - Execução simultânea de múltiplas transações


Usuário/Aplicação
DBA
consultas, atualizações
comandos de transações
comandos DDL
Compilador de consulta
Gerenciador de transação
Compilador DDL
plano de consulta
metadados, estatísticas
metadados
Engine de Execução
Recuperação e log
Controle de concorrência
requisições de registro, arquivo e índice
Gerenciador de re-gistro/arquivo/índice
páginas de log
Tabela bloqueio
dados, metadados, índice
comandos de página
Gerenciador de Buffer
Buffers
escrita/leitura página
Gerenciador de Armazenamento
Armazena-mento
estrutura em memória
componentes
fluxo de controle e dados
fluxo de dados

Modelo de BD Exemplo

- Para exemplificar os conceitos:
  - Modelo de Banco de Dados → composto por itens de dados
  - Operações básicas de acesso ao Banco de dados
    - ler_item(X) ou read(X): Lê um item do banco de dados em uma variável de programa
    - escrever_item(X) ou write(X): Grava o valor de uma variável de programa em um item de banco de dados
    - **obs:** em um SBD real a operação de escrita não necessariamente resulta na atualização imediata dos dados no disco

Operações de Leitura e Escrita de uma Transação

- Read (x) - Lê um item de nome x em variável de programa que também se chama x.
  1. Acha endereço do bloco que contém x
  2. Copia o bloco em buffer na memória principal
  3. Copia item x do buffer p/ a variável de programa chamada x.

- Write (x) - Escreve o valor da variável de programa x no item chamado x.
  1. Acha endereço do bloco que contém item x
  2. Copia o bloco em buffer na memória principal
  3. Copia item x da variável de programa x p/ a sua localização correta no buffer.
  4. Armazena bloco atualizado do buffer p/ o disco

Transfere **k** reais da conta X para a conta Y

T_x

```
read(x)
x = x – k
write(x)
read(y)
y = y + k
write(y)
```

lê o "saldo" X do BD e o armazena na variável X

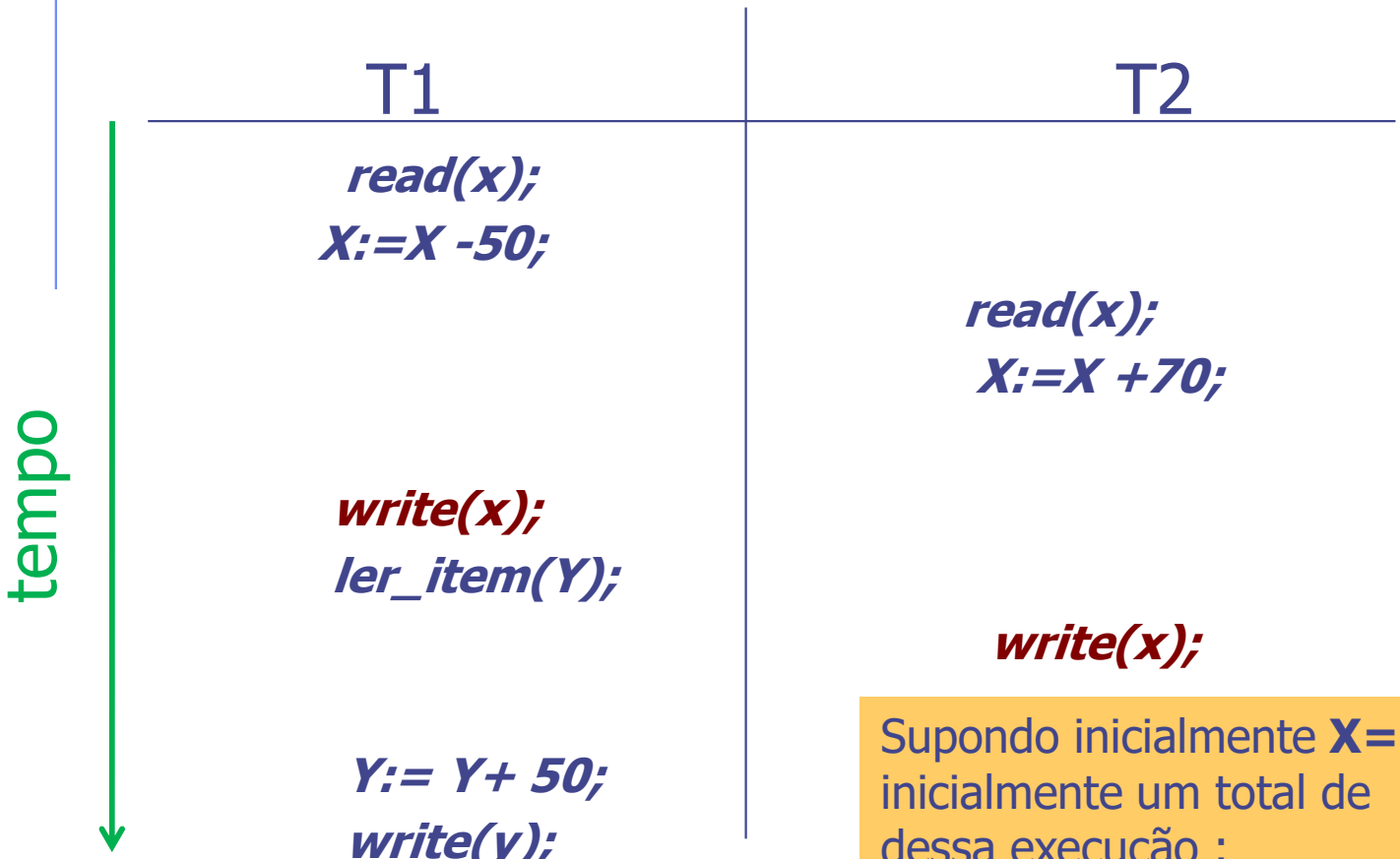
grava no "saldo" Y do BD o valor da variável Y

Por que o controle de concorrência é necessário

- Diversos problemas podem ocorrer quando transações concorrentes são executadas de maneira descontrolada:
  - Falhas de diversos tipos, tais como falhas de hardware e quedas de sistema
  - Execução concorrente de múltiplas transações
    - Problema de atualização perdida
    - Problema da atualização temporária (Leitura Suja)
    - Problema de agregação (Soma, resumo) incorreta

Situação

- Considere duas transações que acessam os mesmos itens do banco de dados, com suas operações intercaladas.



Supondo inicialmente **X=100** e **Y = 300,** tem-se inicialmente um total de (100+300=400). Ao final dessa execução :

- **qual deveria ser o total nas duas contas?**
- **qual será o total nas duas contas?**

Problema de atualização perdida

Inicialmente **X=100** e **Y = 300 ➔ Total = 400**

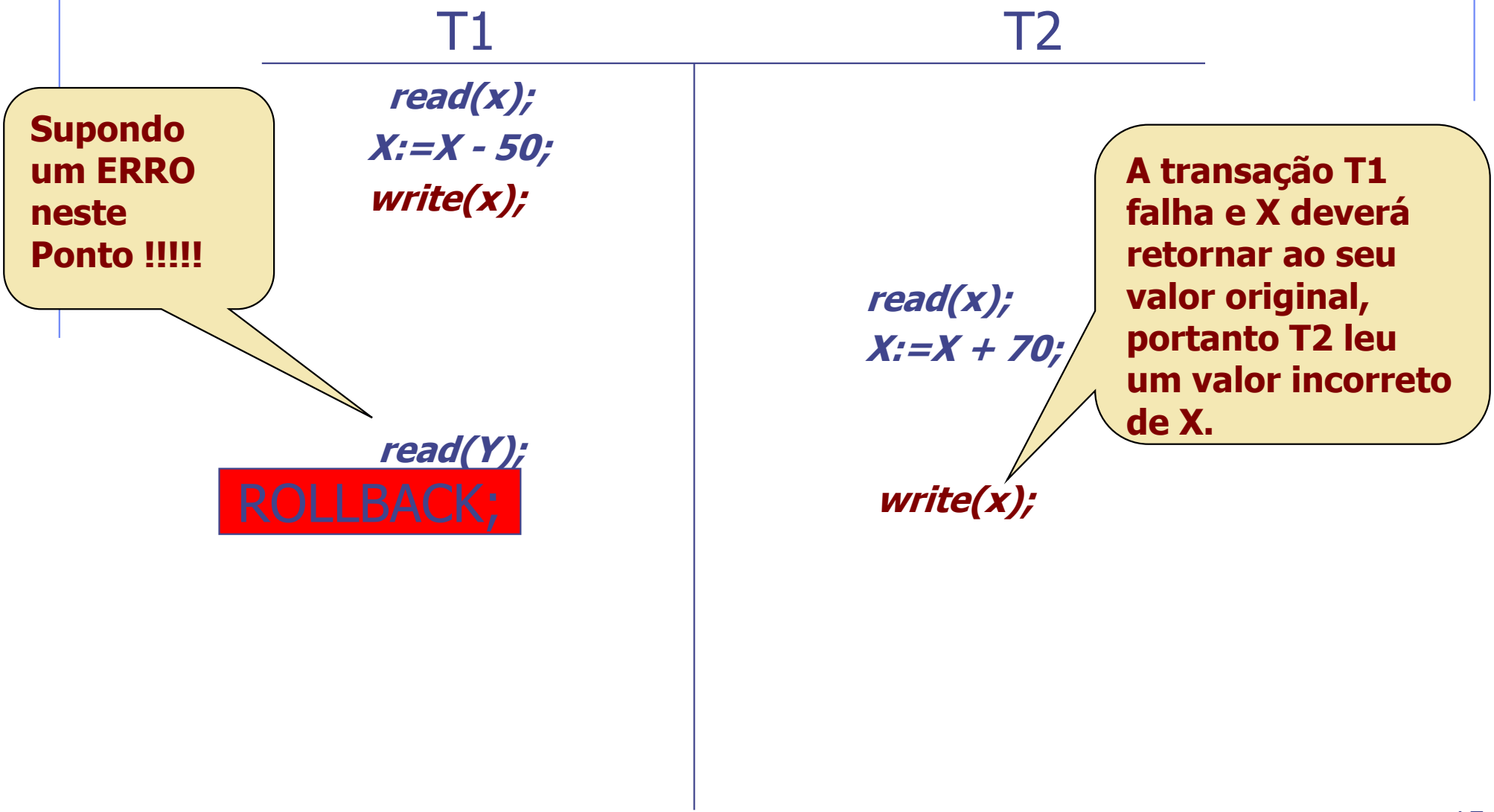
- Subtrai 50 de X e depois soma 70.
  Ao final X= 120 (x deveria ter 120)
  Mas X ficou com 170!!!.
- Soma 50 em Y
  Ao final Y= 350 (Y deveria ter 350).

- Esse problema ocorre quando duas transações que acessam os mesmos itens do banco de dados tiverem suas operações intercaladas, de modo que tornem o valor de alguns dos itens de banco de dados incorretos.

T1	T2
read(x);	
X:=X -50;	
	read(x);
	X:=X +70;
write(x);	
ler_item(Y);	
	write(x);
Y:= Y+ 50;	
write(y);	

Item X tem um valor incorreto pois a atualização feita por T1 foi perdida.

Problema da atualização temporária (Leitura Suja)



Problema de agregação (Soma, resumo) incorreta

T1	T3
	SUM:=0;
read(x);	
X:=X -N;	
write(x);	
	read(x);
	SUM:= SUM + X
	read(Y);
	SUM:= SUM + Y
read(y);	
Y:= Y+ N;	
write(Y)	

T3 lê X depois da subtração de N e lê Y antes da adição de N. O resultado é uma soma errada (sem N)

Resumindo os tipos de problema

- Atualização perdida: ocorre quando duas transações que acessam os mesmos itens de dados tiverem suas operações intercaladas de modo que isso torna o valor de alguns itens do banco de dados incorreto
- Atualização temporária: ocorre quando uma transação atualizar um item de dado e, a seguir, falhar por alguma razão
- Sumário incorreto: ocorre quando uma transação aplicar uma função agregada para um sumário de um número de registros e outras transações estiverem atualizando alguns desses registros

Porque a restauração (recuperação) é necessária

- Sempre que uma transação é submetida, o sistema deve garantir que:

1- Todas as operações na transação se completam com sucesso e seu efeito é registrado permanentemente no banco de dados

ou

2- A transação não terá absolutamente nenhum efeito sobre o banco de dados ou sobre quaisquer outras transações

Porque a recuperação é necessária

- O SGBD não deve permitir que algumas operações de uma transação *T* sejam aplicadas enquanto outras não

- *Tipos de falhas:*
  - 1- Falha de computador
  - 2- Erro de transação ou de sistema (ex.: divisão por zero, etc)
  - 3- Erros locais ou de condições de exceção detectados pelas transações (ex.: Saldo insuficiente)
  - 4- Imposição do controle de concorrência (Deadlock)
  - 5- Falha de disco
  - 6- Problemas físicos e catástrofes

Conceitos de Transação e Sistema

- O gerenciador de recuperação mantém o controle das seguintes operações
  - BEGIN_TRANSACTION: marca o início da execução da transação
  - READ ou WRITE: especificam operações de leitura ou gravação em itens de banco de dados, que são executados como parte da transação
  - END_TRANSACTION: Especifica que as operações de READ e WRITE terminaram, e marca o fim da execução da transação. Nesse ponto, verifica se as mudanças devem ser efetivadas ou se a transação deve ser abortada
  - COMMIT_TRANSACTION: Indica o término com sucesso da transação de forma que as atualizações podem ser seguramente efetivadas
  - ROLLBACK: Indica que a transação não terminou com sucesso e possíveis atualizações devem ser desfeitas

Conceitos de Transação e Sistema

- ***Log do sistema*** – O sistema mantém um *log* para acompanhar todas as operações das transações que afetem os valores dos itens do banco de dados.
- Estas informações podem ser necessárias para possibilitar a recuperação de falhas.
- O *log* é mantido em disco para que não seja afetado por qualquer tipo de falha.

Log do Sistema

- O LOG (registro de ocorrências do sistema) registra os seguintes tipos de entradas:
- Id_transação: Identificador da transação, gerado automaticamente pelo sistema.
- [START_TRANSACTION, T]
- [WRITE_ITEM, T, X, valor-antigo, valor_novo]
- [READ_ITEM, T, X]
- [COMMIT, T]
- [ABORT, T]

ID da transação

Ponto de Confirmação (Commit)

- Uma transação T alcança seu ponto de confirmação (ponto commit) quando **todas** as suas operações que acessam o banco de dados tiverem sido ***executadas com sucesso*** e o efeito de todas as operações da transação no banco de dados tiverem sido registradas no log
- Diz-se que a transação foi confirmada (commited) quando seus efeitos tiverem sido ***permanentemente registrados no banco de dados***

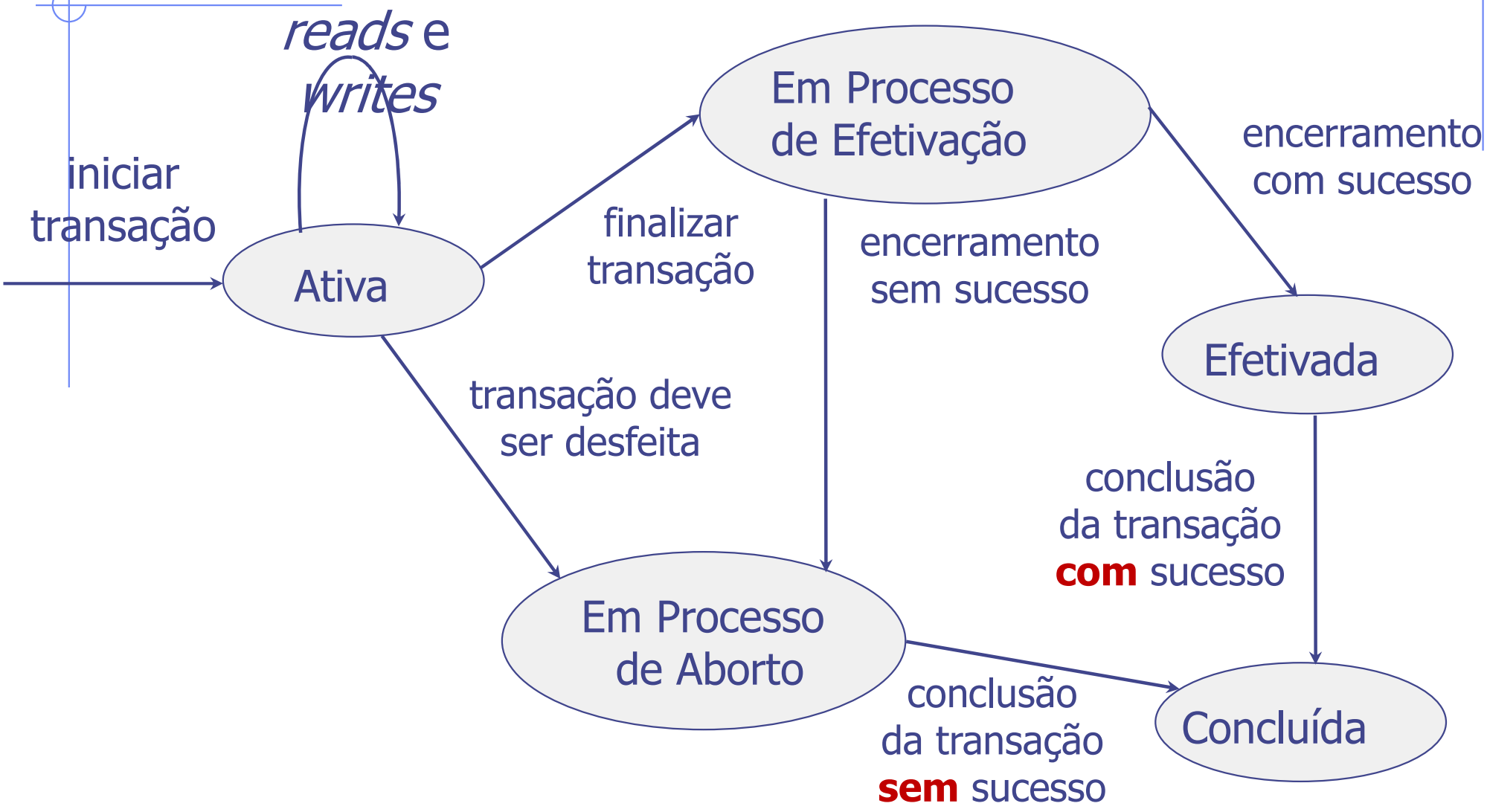
Ponto de Confirmação (Commit)

- Antes que uma transação atinja o ponto de commit, qualquer parte do log que ainda não tenha sido gravada no disco deve ser gravada imediatamente
- Isto é necessário pois em caso de falhas somente as entradas de log que tiverem sido **registradas** em disco podem ser usadas para recuperação

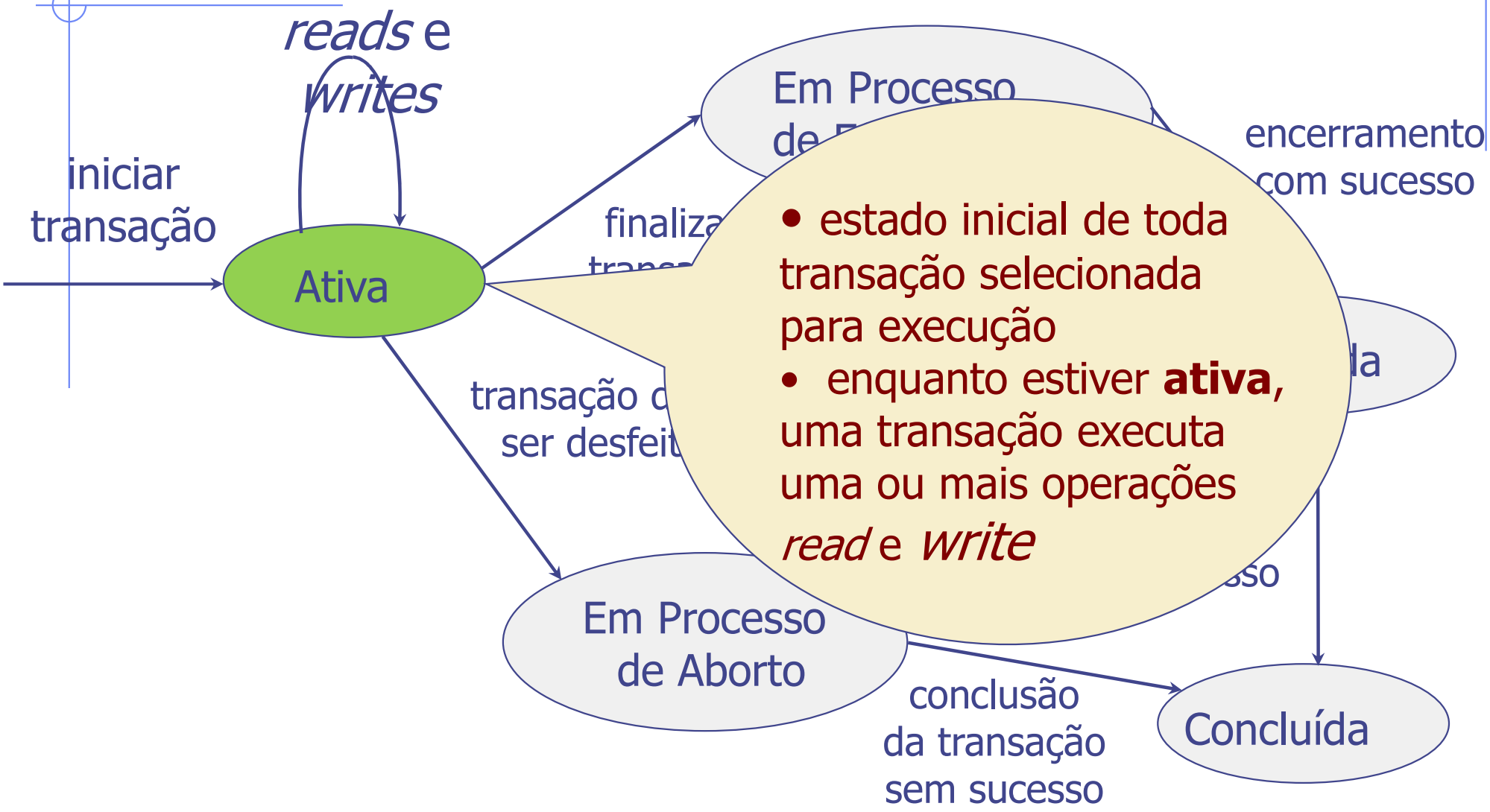
Estados de uma Transação

- Uma transação é sempre monitorada pelo SGBD quanto ao seu estado
  - Que operações já fez?
  - Concluiu suas operações?
  - Deve abortar?
- Estados de uma transação
  - Ativa
  - Em processo de efetivação
  - Efetivada
  - Em processo de aborto
  - Concluída

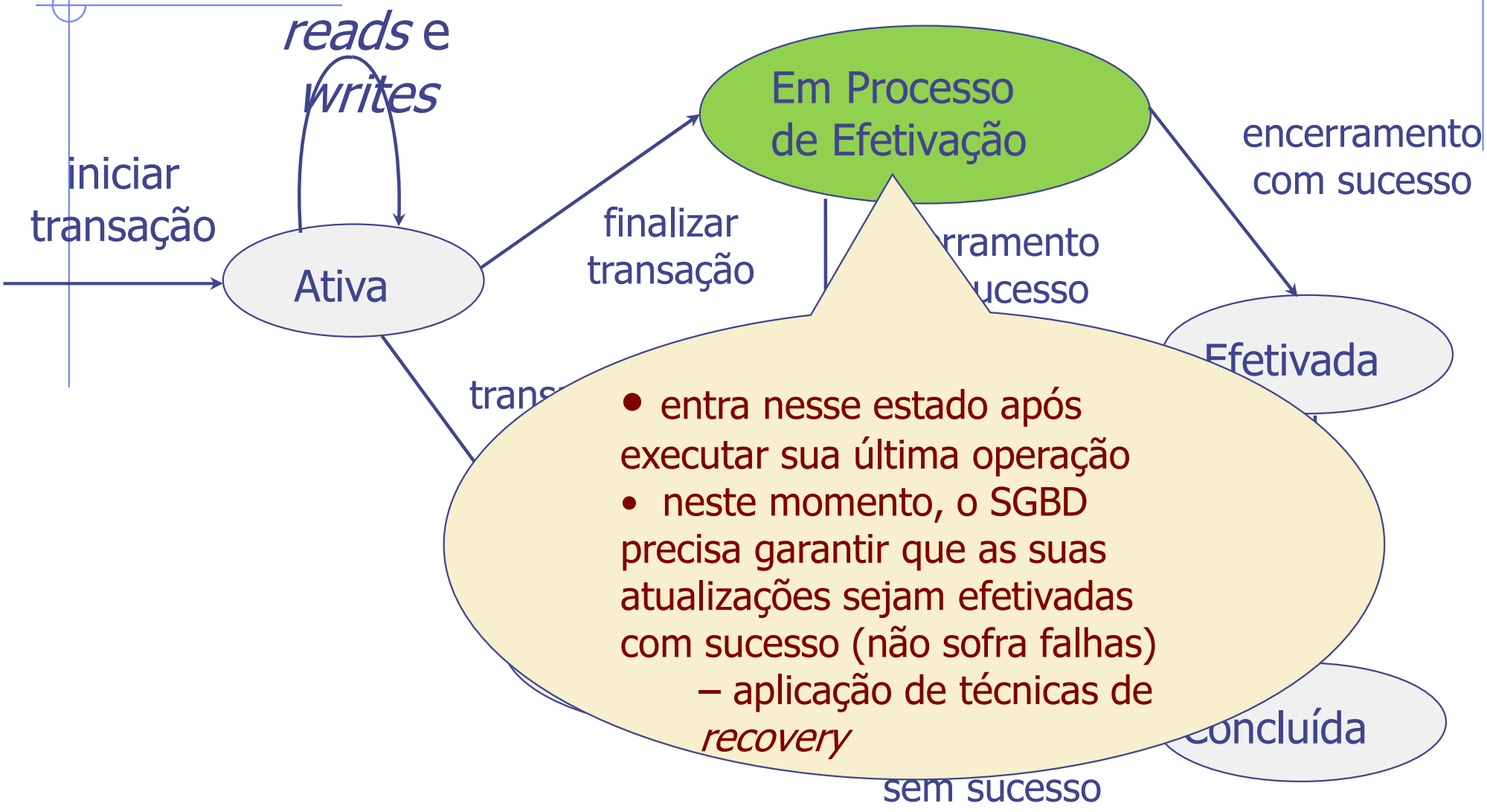
Transição de Estados de uma Transação



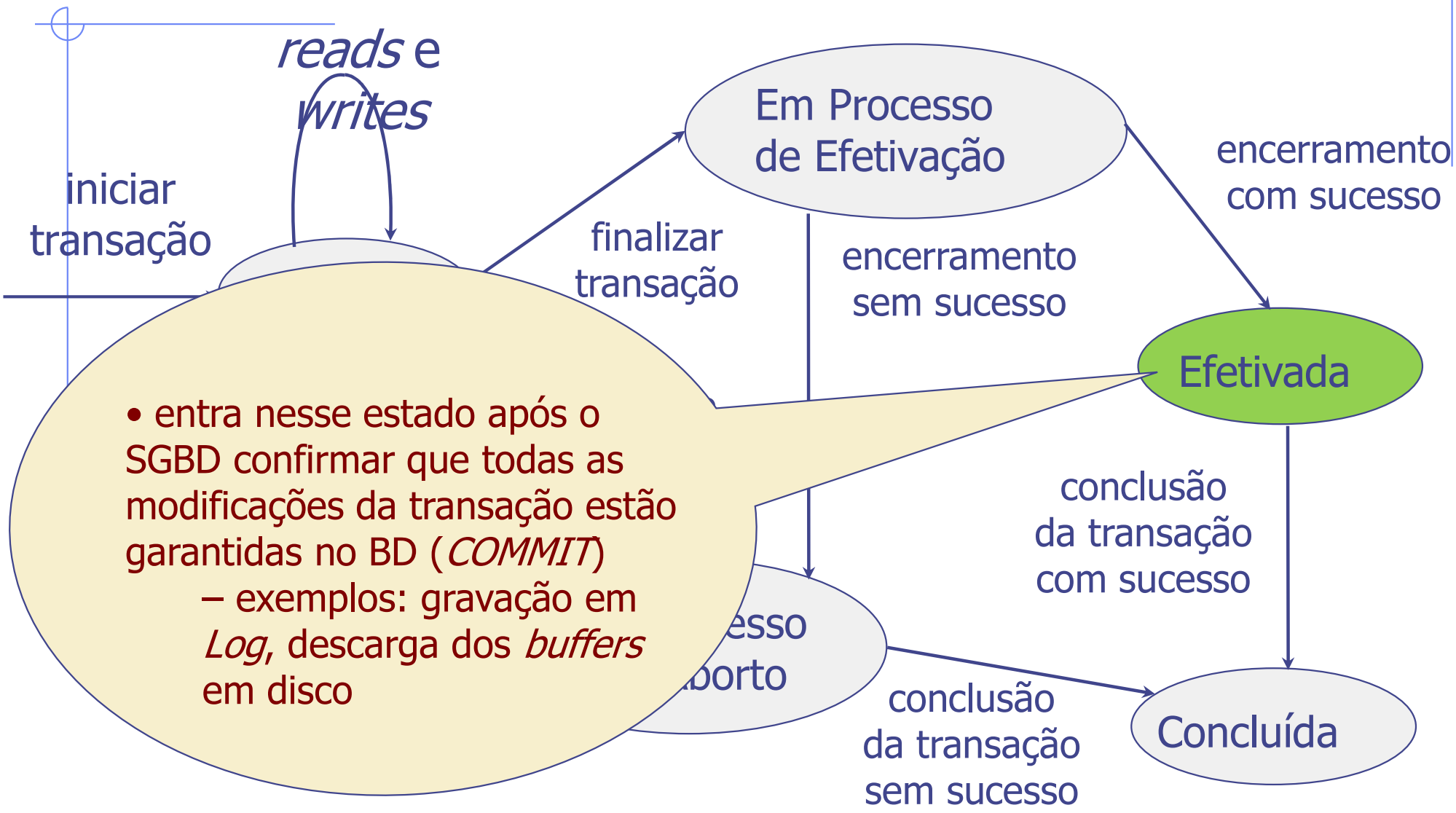
Transição de Estados de uma Transação



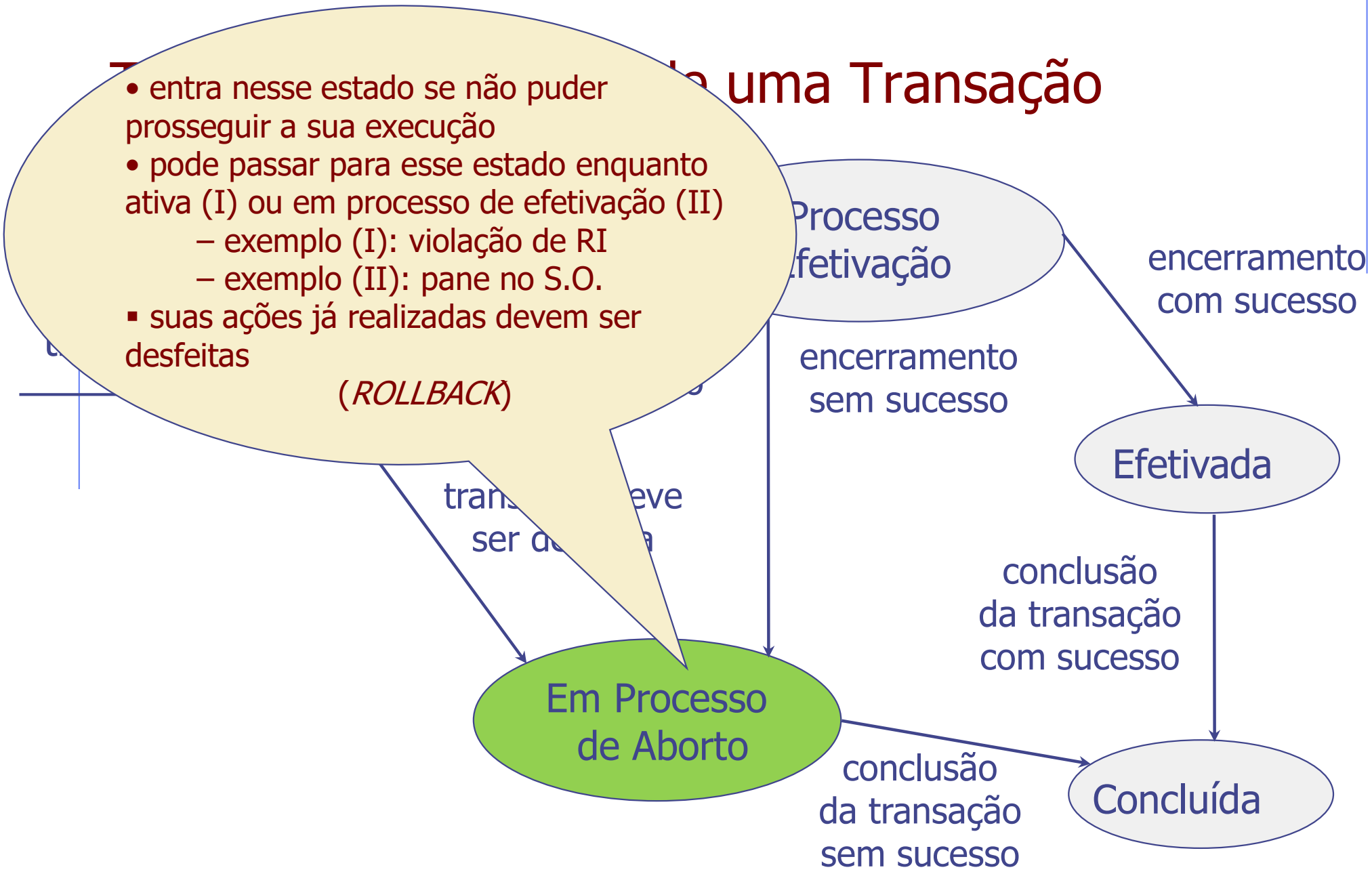
Transição de Estados de uma Transação



Transição de Estados de uma Transação

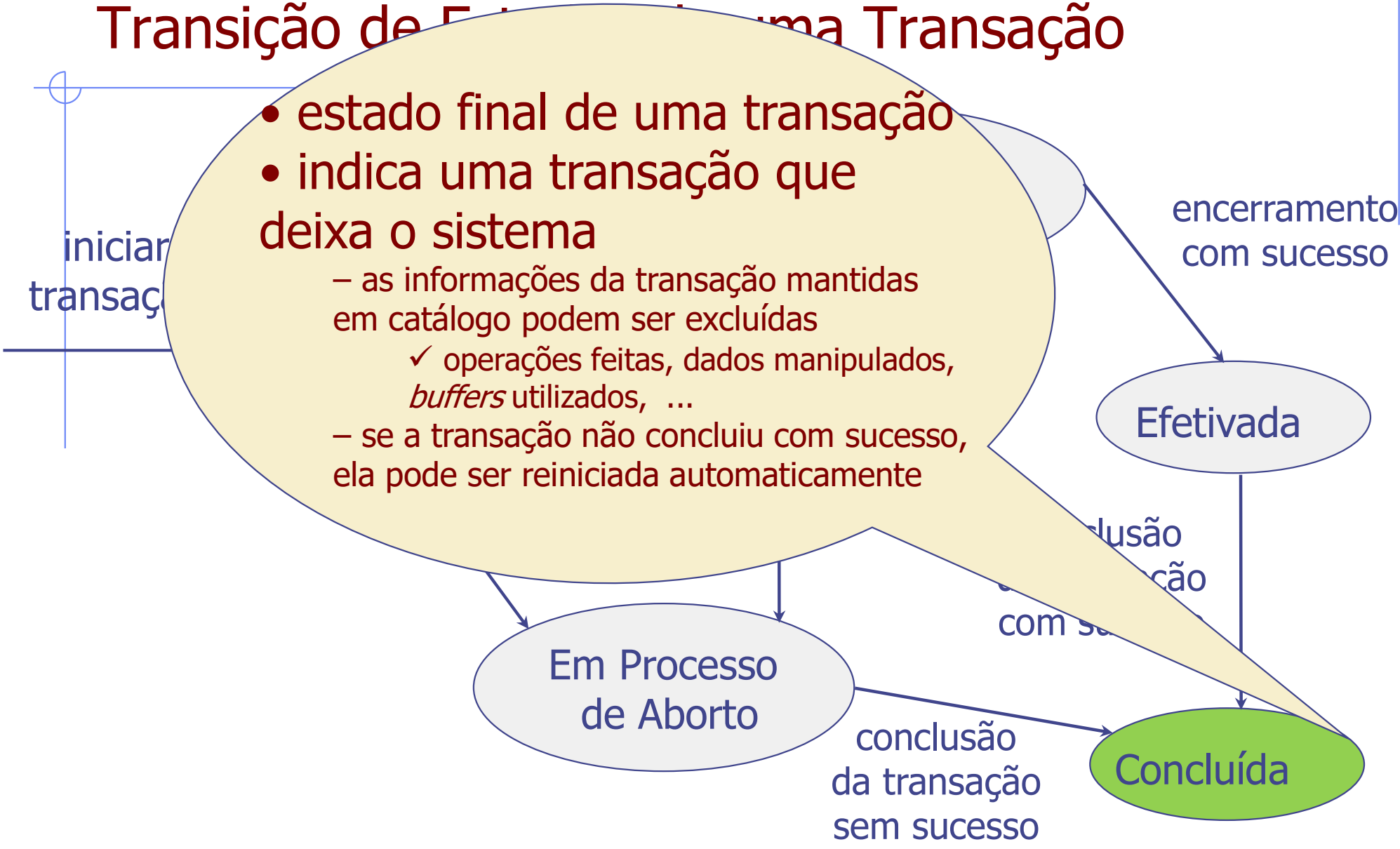


Transação uma Transação



Transição de Estado de uma Transação

- estado final de uma transação
- indica uma transação que deixa o sistema
  - as informações da transação mantidas em catálogo podem ser excluídas
    - ✓ operações feitas, dados manipulados, *buffers* utilizados, ...
  - se a transação não concluiu com sucesso, ela pode ser reiniciada automaticamente



Propriedades desejáveis (ACID)

- Requisitos que sempre devem ser atendidos por uma transação para garantir a integridade dos dados
  - Atomicidade
  - Consistência
  - Isolamento
  - Durabilidade

Propriedades desejáveis

- Para garantir a integridade dos dados
  - Atomicidade: todas as operações da transação são refletidas corretamente no BD, ou nenhuma delas
  - Consistência: a execução de uma transação isolada preserva a consistência do BD
  - Isolamento: Cada transação não está ciente das outras transações executando simultaneamente no sistema
  - Durabilidade: As alterações realizadas por uma transação que completou com sucesso são persistidas mesmo que ocorram falhas no sistema

Atomicidade

- Princípio do *"Tudo ou Nada"*
  - ou todas as operações da transação são efetivadas com sucesso no BD ou nenhuma delas se efetiva
    - preservar a integridade do BD
  - Responsabilidade do subsistema de recuperação contra falhas (subsistema de *recovery*) do SGBD
    - desfazer as ações de transações parcialmente executadas

Consistência

- Uma transação sempre conduz o BD de um estado consistente para outro estado também consistente
  - durante a execução da transação, a base de dados pode passar por um estado inconsistente
- Responsabilidade conjunta do
  - DBA
    - definir todas as RIs para garantir estados e transições de estados válidos para os dados
      - exemplo: saldo > 0
  - subsistema de *recovery*
    - desfazer as ações da transação que violou a integridade

Isolamento

- A execução de uma transação T_x deve funcionar como se T_x executasse de forma isolada
  - T_x não deve sofrer interferências de outras transações executando concorrentemente
  - Resultados intermediários das transações devem ser escondidos de outras transações executadas concorrentemente
  - Responsabilidade do subsistema de controle de concorrência (*scheduler*) do SGBD

Isolamento

T_1	T_2
read(A)	
A = A – 50	
write(A)	
	read(A)
	A = A+A*0.1
	write(A)
read(B)	
B = B + 50	
write(B)	
	read(B)
	B = B - 100
	write(B)

escalonamento válido

T_1	T_2
read(A)	
A = A – 50	
	read(A)
	A = A+A*0.1
	write(A)
	read(B)
write(A)	
read(B)	
B = B + 50	
write(B)	
	B = B - 100
	write(B)

T_1 interfere em T_2

T_2 interfere em T_1

escalonamento inválido

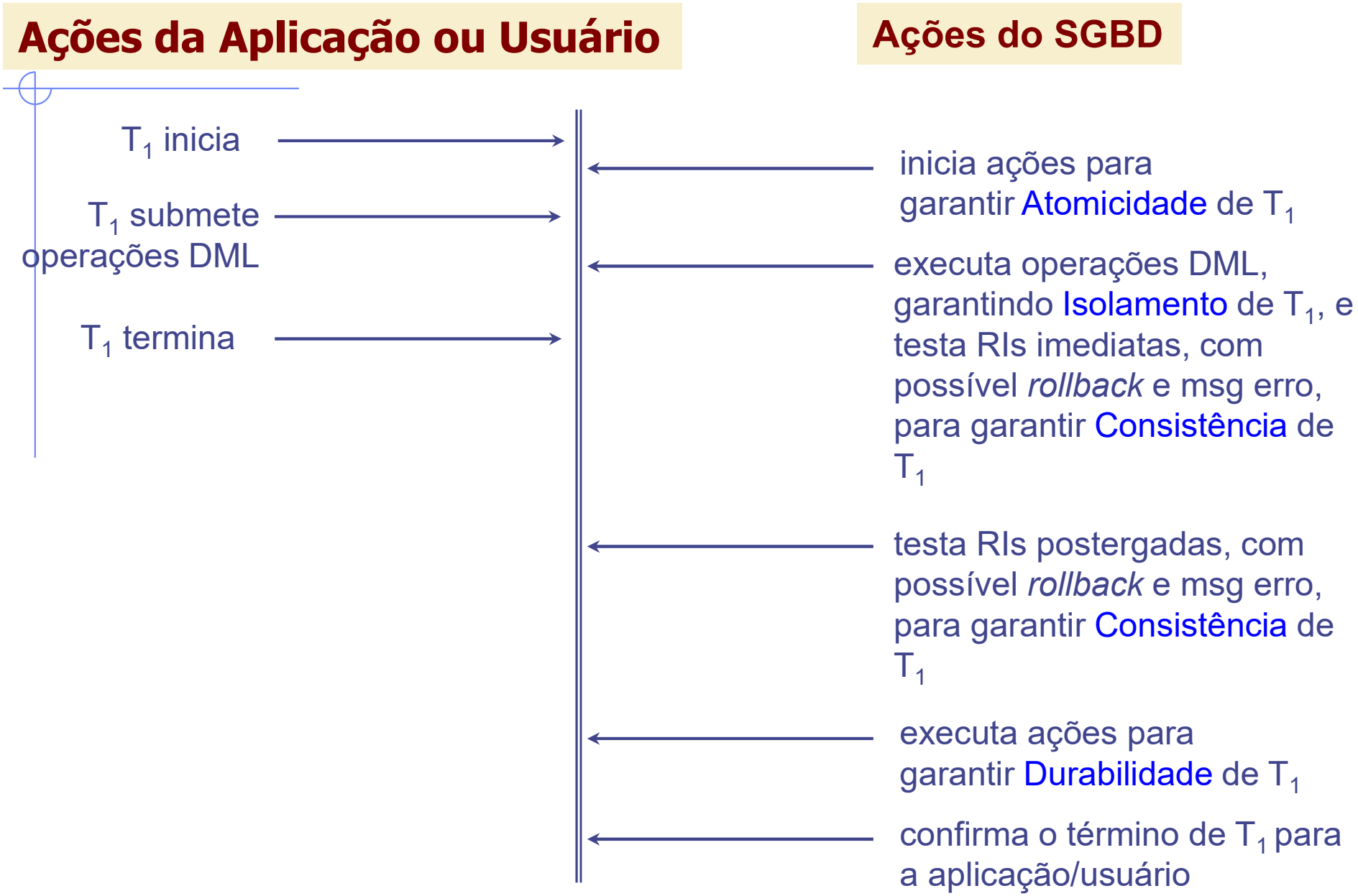
Durabilidade

- Deve-se garantir que as modificações realizadas por uma transação que concluiu com sucesso persistam no BD
  - nenhuma falha posterior ocorrida no BD deve perder essas modificações
  - Responsabilidade do subsistema de *recovery*
    - refazer transações que executaram com sucesso em caso de falha no BD

Exemplo

- Transação para transferir R$ 50 da conta ***A*** para conta ***B***
  1. Read (A)
  2. A = A – 50
  3. Write (A)
  4. Read (B)
  5. B = B + 50
  6. Write (B)
  - Requisito de ***Consistência***
    - A soma de *A* e *B* não se altera pela execução da transação (o dinheiro apenas mudou de conta)
  - Requisito de ***Atomicidade***
    - Se a transação falhar após o passo 3 e antes do passo 6, o sistema deve assegurar que as suas atualizações não sejam refletidas no banco de dados, senão resultará em uma inconsistência
  - Requisito de ***Durabilidade***
    - Uma vez que o usuário tenha sido notificado que a transação foi completada (ou seja, a transferência dos R$50 ocorreu), as atualizações no banco de dados feitas pela transação devem persistir, mesmo na ocorrência de falhas
  - Requisito de ***Isolamento***
    - Se, entre os passos 3 e 6, outra transação tiver permissão para acessar o banco de dados parcialmente atualizado, ela verá um banco de dados inconsistente (a soma A + B valerá menos do que deveria)

Gerência Básica de Transações



Execuções simultâneas

- Várias transações podem ser executadas simultaneamente no sistema. As vantagens são:
  - Melhor utilização do processador e do disco, levando a um melhor *throughput* de transação: uma transação pode estar usando a CPU enquanto outra está lendo ou escrevendo no disco
  - Tempo médio de resposta reduzido para transações: as transações curtas não precisam esperar atrás das longas
  - Esquemas de controle de concorrência – mecanismos para obter isolamento; ou seja, para controlar a interação entre as transações concorrentes a fim de evitar que elas destruam a consistência do banco de dados

Plano de execução (*Schedules*)

- Seqüências de instruções que especificam a ordem cronológica em que as instruções das transações simultâneas são executadas
  - Um plano de execução para um conjunto de transações precisa consistir em todas as instruções dessas transações
  - Precisam preservar a ordem em que as instruções aparecem em cada transação individual
  - Uma transação que completa com sucesso sua execução terá uma instrução *commit* como a última instrução (será omitida se for óbvia)
  - Uma transação que não completa com sucesso sua execução terá instrução *abort* como a última instrução (será omitida se for óbvia)

Plano de execução 1

- Suponha que T_1 transfere R$ 50 de A para B e T_2 transfere 10% do saldo de A para B. A seguir está um plano de execução serial em que T_1 é seguido de T_2

Valores iniciais
A = R$1000,00
B = R$2000,00

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

Valores finais
A = R$855,00
B = R$2145,00

Plano de execução 1

Plano de execução serial:
- **Não há intercalação de operações**

- Suponha que T_1 transfere R$ 50 de A para B e T_2 transfere 10% do saldo de A para B. A seguir está um plano de execução serial em que T_1 é seguido de T_2

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

Valores iniciais
A = R$1000,00
B = R$2000,00

Valores finais
A = R$855,00
B = R$2145,00

Plano de execução 2 - Um plano de execução serial em que *T*2 é seguido de *T*1

Valores iniciais
A = R$1000,00
B = R$2000,00

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	

Valores finais
A = R$850,00
B = R$2150,00

Plano de execução 3

- Sejam T_1 e T_2 as transações definidas anteriormente. O plano de execução a seguir não é serial, mas é equivalente ao Plano de execução 1

Valores iniciais
A = R$1000,00
B = R$2000,00

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Valores finais
A = R$855,00
B = R$2145,00

Plano de execução 3

Plano de execução não serial:
• há intercalação de operações

- Sejam T_1 e T_2 as transações definidas anteriorm[illegible]. O plano de execução a seguir não é serial, mas é equiva[illegible]nte ao Plano de execução 1

T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
read(B)	
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)

Valores iniciais
A = R$1000,00
B = R$2000,00

Valores finais
A = R$855,00
B = R$2145,00

Plano de execução 4

- O seguinte plano de execução concorrente não preserva o valor da soma *A* + *B*

Valores iniciais
A = R$1000,00
B = R$2000,00

T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

Valores finais
A = R$950,00
B = R$2100,00

Seriação

- Suposição básica – Cada transação preserva a consistência do banco de dados
- Portanto, a execução serial de um conjunto de transações preserva a consistência do banco de dados
- Um plano de execução (possivelmente simultâneo) é serializável se for equivalente a um plano de execução serial

Seriação

- Diferentes formas de equivalência de planos de execução
  - Seriação de conflito
  - Seriação de visão
- Observações:
  - Exceto *read* e *write*, todas as outras operações são ignoradas
  - Considera-se que as transações podem realizar cálculos arbitrários sobre dados em buffers locais entre *reads* e *writes*
  - Planos de execução simplificados consistem apenas em instruções *read* e *write*

Instruções conflitantes

- As instruções I_i e I_j das transações T_i e T_j respectivamente, estão em conflito se e somente se algum item Q acessado por I_i e por I_j e pelo menos uma destas instruções escreveram Q.

  1. I_i = read(Q), I_j = read(Q). I_i e I_j não estão em conflito
  2. I_i = read(Q), I_j = write(Q). Estão em conflito
  3. I_i = write(Q), I_j = read(Q). Estão em conflito
  4. I_i = write(Q), I_j = write(Q). Estão em conflito

- Intuitivamente, um conflito entre I_i e I_j força uma ordem temporal (lógica) entre eles. Se I_i e I_j são consecutivos em um plano de execução e não entram em conflito, seus resultados permanecem inalterados mesmo se tiverem sido trocados no plano de execução

Seriação de conflito

- Se um plano de execução *S* puder ser transformado em um plano de execução *S'* por uma série de trocas de instruções não conflitantes, dizemos que *S* e *S'* são equivalentes em conflito
- Dizemos que um plano de execução *S* é serial de conflito se ele for equivalente em conflito a um plano de execução serial

Seriação de conflito (cont.)

- O plano de execução 3 abaixo pode ser transformado em Plano de execução 1 (próximo slide), no qual T_2 segue T_1, por uma série de trocas de instruções não conflitantes. Portanto, o plano de execução 3 é serial de conflito

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Seriação de conflito (cont.) – Plano de execução 3 após trocar um par de instruções

T_1	T_2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

Seriação de conflito (cont.) – Um plano de execução serial que é equivalente ao plano de execução 3

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Plano de execução 1

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

Seriação de conflito (cont.)

- Exemplo de um plano de execução que não é serial de conflito:

T_3	T_4
read(Q)	
	write(Q)
write(Q)	

  - Não podemos trocar instruções no plano de execução acima para obter o plano de execução serial < T_3, T_4 >, ou o plano de execução serial < T_4, T_3 >.

Seriação de visão

- Sejam *S* e *S´* dois planos de execução com o mesmo conjunto de transações. *S* e *S´* são equivalentes em visão se as três condições a seguir forem satisfeitas:

  1. Para cada item de dados *Q,* se a transação T_i ler o valor inicial de *Q* no plano de execução *S,* então, T_i precisa, no plano de execução *S´*, também ler o valor inicial de *Q.*
  2. Para cada item de dados *Q*, se a transação T_i executar read(*Q*) no plano de execução *S*, e se esse valor foi produzido por write(Q) na transação T_j (se houver), então a transação T_i, no plano de execução *S'*, também precisa ler o valor de *Q* que foi produzido pela transação T_j.
  3. Para cada item de dados *Q*, a transação (se houver) que realiza a operação write(*Q*) final no plano de execução *S* precisa realizar a operação write(*Q*) final no plano de execução *S'*.

Como podemos ver, a equivalência em visão também é baseada unicamente em reads e writes isolados.

Seriação de visão

- As condições 1 e 2 garantem que cada transação lê os mesmos valores em ambos planos de execução
- A condição 3, juntamente com 1 e 2, garantem que ambos planos de execução resultam no mesmo estado final

Seriação de visão - exemplo

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

São equivalentes em visão?

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	

Seriação de visão - e

T_1	T_2
read(A)	
$A := A - 50$	
write (A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)

NÃO são equivalentes em visão!

No plano de execução 1 o valor da conta A lida por T_2 foi produzido por T_1. O mesmo não ocorre no plano de execução 2

T_1	T_2
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	

Seriação de visão (cont.)

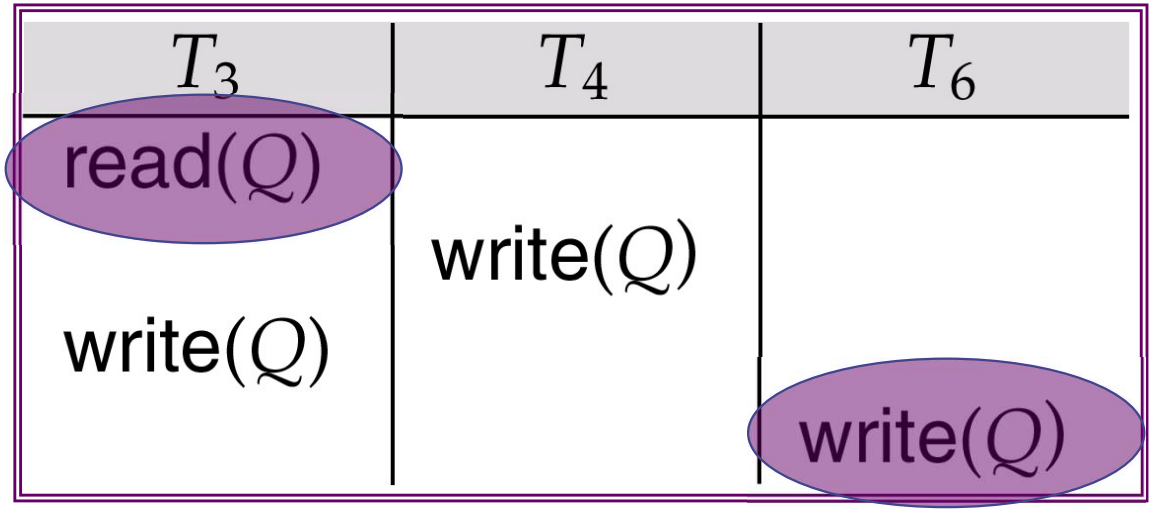
- Um plano de execução *S* é serial de visão se ele for equivalente em visão a um plano de execução serial
- Todo plano de execução serial de conflito também é serial de visão
- A seguir está um plano de execução que é serial de visão mas *não* serial de conflito

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- Todo plano de execução serial de visão que não é serial de conflito possui escritas cegas

Seriação de visão (cont.)

- A seguir está um plano de execução que é serial de visão ao plano de execução serial <T_3,T_4,T_6>



Facilidade de recuperação

- É necessário tratar do efeito das falhas de transação nas transações sendo executadas simultaneamente.
  - Plano de execução recuperável — Se uma transação T_j lê um item de dados anteriormente escrito por uma transação T_i, então a operação commit de T_i aparece antes da operação commit de T_j.
  - O plano de execução (Plano de execução 11) não é recuperável se T_9 for confirmado imediatamente após o read

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

  - Se T_8 abortasse, T_9 teria lido (e possivelmente mostrado ao usuário) um estado inconsistente. Portanto, o banco de dados precisa garantir que planos de execução sejam recuperáveis

Facilidade de recuperação (cont.)

- Rollback em cascata – Uma única falha de transação leva a uma série de rollbacks de transação. Considere o seguinte plano de execução no qual nenhuma das transações ainda foi confirmada (portanto, o plano de execução é recuperável)

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

- Se T_{10} falhar, T_{11} e T_{12} também precisam ser revertidos
- Pode chegar a desfazer uma quantidade de trabalho significativa

Facilidade de recuperação (cont.)

- Planos de execução não em cascata — Rollbacks em cascata não podem ocorrer; para cada par de transações T_i e T_j tal que T_j leia um item de dados escrito anteriormente por T_i, a operação commit de T_i apareça antes da operação read de T_j.
- Todo plano de execução não em cascata também é recuperável
- É desejável restringir os planos de execução aos não em cascata

Aspectos de implementação

- Um banco de dados precisa fornecer um mecanismo que garanta que todos os planos de execução possíveis sejam seriais de conflito ou de visão, e sejam recuperáveis e, preferivelmente, não em cascata
- Uma política em que apenas uma transação pode ser executada de cada vez gera planos de execução seriais, mas fornece um menor grau de concorrência
- Os esquemas de controle de concorrência conciliam entre a quantidade de concorrência que permitem e a quantidade de sobrecarga a que ficam sujeitos
- Um esquema de recuperação garante que as propriedades de atomicidade e durabilidade das transações sejam preservadas

Definição de transação na SQL

- A linguagem de manipulação de dados precisa incluir uma construção para especificar o conjunto de ações que compõem uma transação
- Na SQL, uma transação começa implicitamente
- Uma transação na SQL termina por:
  - Commit - confirma a transação atual e inicia uma nova transação
  - Rollback - faz com que a transação atual seja abortada
- Níveis de consistência especificados pela SQL-92:
  - Serializable — padrão
  - Repeatable read
  - Read committed
  - Read uncommitted

Níveis de consistência na SQL-92

Tabela 21.1 – Violações possíveis com base nos níveis de isolamento definidos na SQL

Nível	Leitura suja	Leitura não repetitiva	Fantasma
Read uncommitted	Sim	Sim	Sim
Read committed	Não	Sim	Sim
Repeatable read	Não	Não	Sim
Serializable	Não	Não	Não

- **Leitura suja**: uma transação T_1 pode ler a atualização de uma transação T_2, que ainda não foi confirmada. Se T_2 falhar e for abortada, então T_1 teria lido um valor que não existe e é incorreto
- **Leitura não repetitiva**: uma transação T_1 pode ler determinado valor de uma tabela. Se outra transação T_2 mais tarde atualizar esse valor e T_1 ler o valor novamente, T_1 verá um valor diferente
- **Fantasmas**: uma transação T_1 lê um conjunto de linhas de uma tabela, com base em uma cláusula WHERE. T_2 insere uma nova linha que também satisfaz a cláusula WHERE usada em T_1. Se T_1 for repetida, verá um fantasma, uma linha que anteriormente não existia.

Postgres

- Transação: bloco que deve ser executado ou desfeito

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
-- etc etc
COMMIT;
```

- A ferramenta **pgAdmin** do PostgreSQL trata cada sentença SQL como sendo executada dentro de uma transação
  - se você não disparar um BEGIN, então cada sentença será COMMIT ao final de sua execução
    - verifique a implementação do cliente

Postgres

- Exemplo: transferência de R$ 100 de Alice para Bob, ops, corrigindo, de Alice para Wally!

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
-- oops ... forget that and use Wally's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally';
COMMIT;
```

- savepoint: ponto de salvamento que permite dividir uma transação em partes que podem ser desfeitas

Postgres

```
SET TRANSACTION transaction_mode [, ...]
SET TRANSACTION SNAPSHOT snapshot_id
SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode [, ...]

where transaction_mode is one of:

    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }
    READ WRITE | READ ONLY
    [ NOT ] DEFERRABLE
```

```
CREATE TABLE table_name (
  { column_name data_type
    ...
[ CONSTRAINT constraint_name ]
    ...
[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

```
SET CONSTRAINTS { ALL | name [, ...] } { DEFERRED | IMMEDIATE }
```

Leitura complementar para casa

- Capítulo 21 do livro: Elmasri, Ramez; Navathe, Shamkant B. Sistemas de banco de dados. 6ª edição.
- Capítulo 14 do livro: Silberschatz, A; Korth, H. F.; Sudarshan, S. Sistema de banco de dados. 6ª edição.

Exercícios complementares

1. O que é execução concorrente de transações em banco de dados num sistema multiusuário? Diga por que o controle de concorrência é necessário e dê exemplos informais.
2. Defina as propriedades de atomicidade, durabilidade, isolamento e preservação de consistência de uma transação de banco de dados. Apresente e utilize pelo menos dois exemplos de transações para discutir cada um desses requisitos (obs: os exemplos têm que ser diferentes dos apresentados nas notas de aula).

Exercícios complementares

3. Defina os problemas que podem ser causados pela execução concorrente de transações: atualização perdida, atualização temporária (Leitura Suja) e agregação (Soma, resumo) incorreta. Apresente e utilize pelo menos dois exemplos de transações para discutir cada um desses problemas (obs: os exemplos têm que ser diferentes dos apresentados nas notas de aula).
4. Discuta como a seriabilidade é usada para garantir o controle de concorrência em um sistema de banco de dados.

Exercícios complementares

5. Quais dos seguintes planos é serializável (conflito)? Para cada plano serializável, determine os planos 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) $r_3(X);r_2(X);r_1(X);w_3(X);w_1(X);$
6. Para que é usado o log do sistema?
7. Discuta as diferentes medidas de equivalência de transação. Qual é a diferença entre equivalência de confito e equivalência de visão?