


# **Transações**

Controle de Concorrência

Serializabilidade

# Introdução a Transações

- **SGBD**  sistema de processamento de operações de acesso ao BD.
- SGBDs são em geral **multi-usuários**
  - processam simultaneamente operações disparadas por vários usuários
    - deseja-se alta disponibilidade e tempo de resposta pequeno
  - execução intercalada de conjuntos de operações
    - exemplo: enquanto um processo  $i$  faz I/O, outro processo  $j$  é selecionado para execução.
- Operações são chamadas **transações**

# Transação

- Uma transação é uma unidade de execução de programa que acessa e, possivelmente atualiza vários itens de dados.
- Uma transação geralmente é resultado da execução de um programa de usuário escrito em uma linguagem de manipulação de dados de alto nível ou em uma linguagem de programação (por exemplo, SQL, Cobol, C ou Pascal), e é delimitada por declarações (ou chamadas de funções) da forma “Início da transação” e “Final da Transação”.

# Transação

- De forma abstrata e simplificada, uma transação pode ser encarada como um conjunto de operações de leitura(read) e escrita(write) de dados.
- Read(x) – transfere o item X do banco de dados para um buffer local alocado a transação que executou a operação de **read**.
- Write(x) – transfere o item de dados X do buffer local da transação que executou a **write** de volta ao banco de dados.

# Exemplo

- Seja  $T_i$  uma transação que transfere 50 dólares da conta A para conta B. Essa transação pode ser definida como:

$T_i$ : read(A);  
     $A:=A-50$ ;  
    write(A);  
    read(B);  
     $B:=B+50$ ;  
    write(B)

passo a passo

Leitura da Conta A: 1.000

$A:=1000-50$ ;

$A=950$ ;

Leitura da Conta B: 2.000

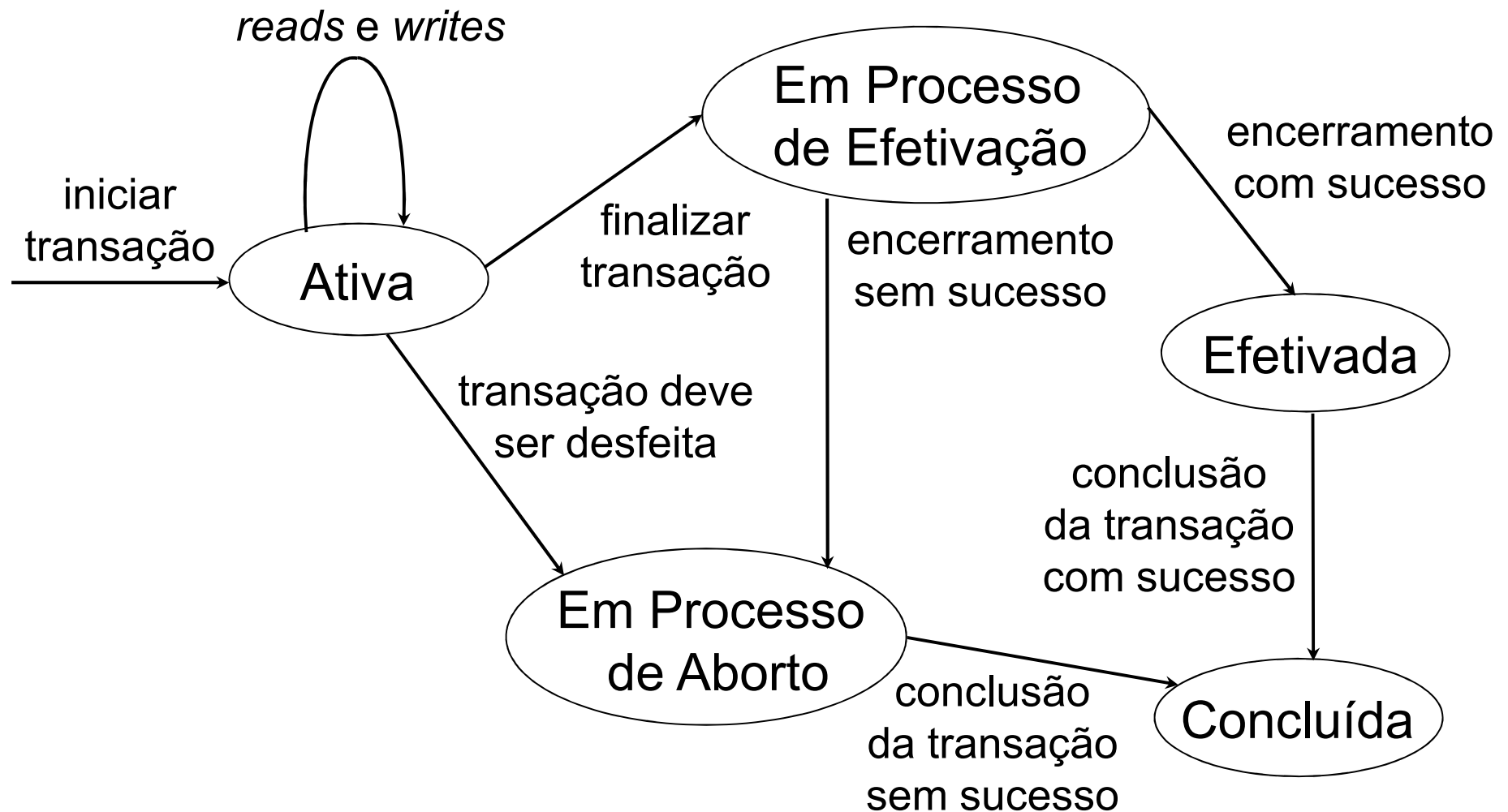
$B:=2.000+50$ ;

$B=2.050$

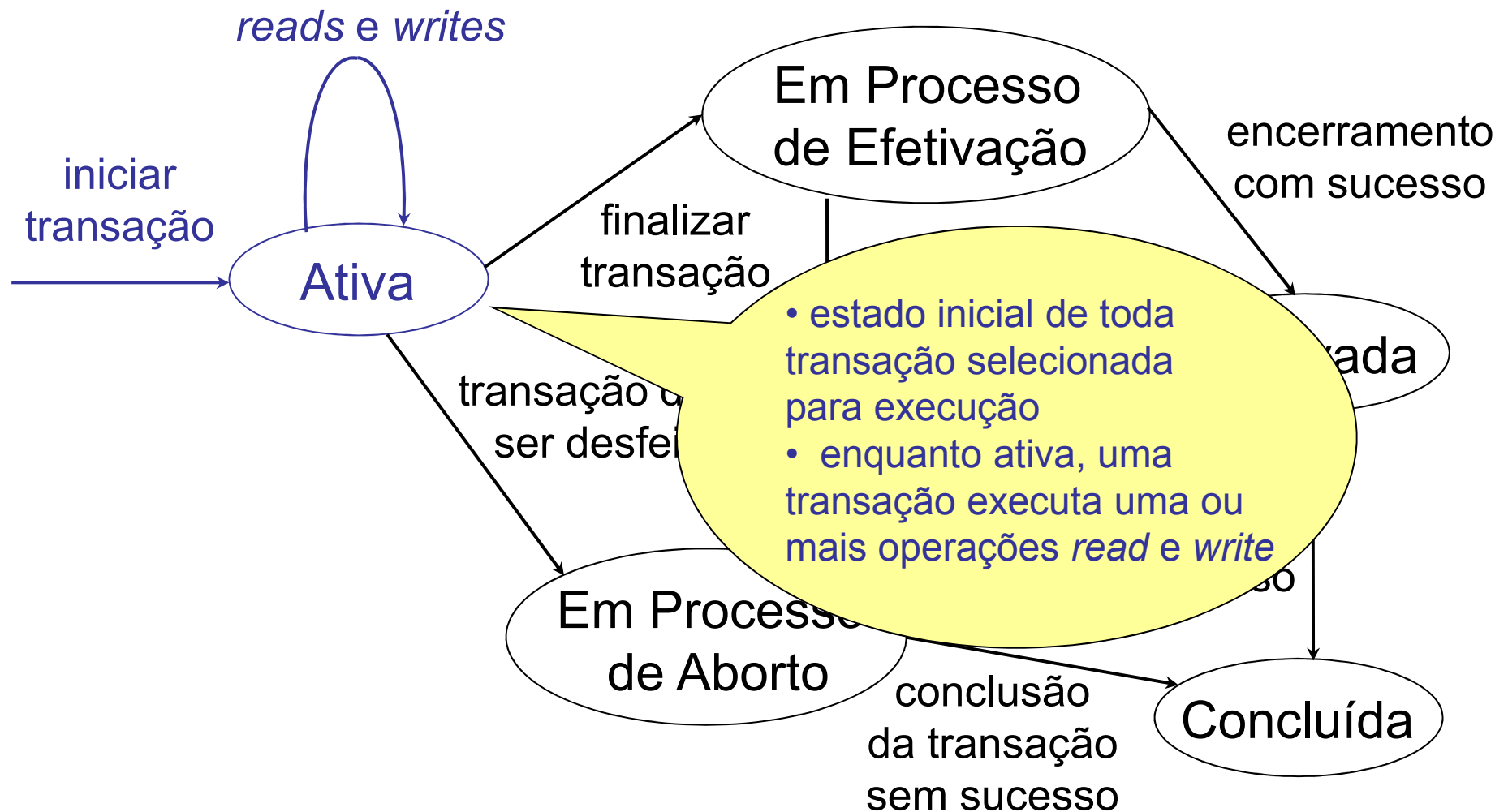
# Estados de uma Transação

- ❏ 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**.
  - 👤 Respeita um Grafo de Transição de Estados

# Transição de Estados de uma Transação

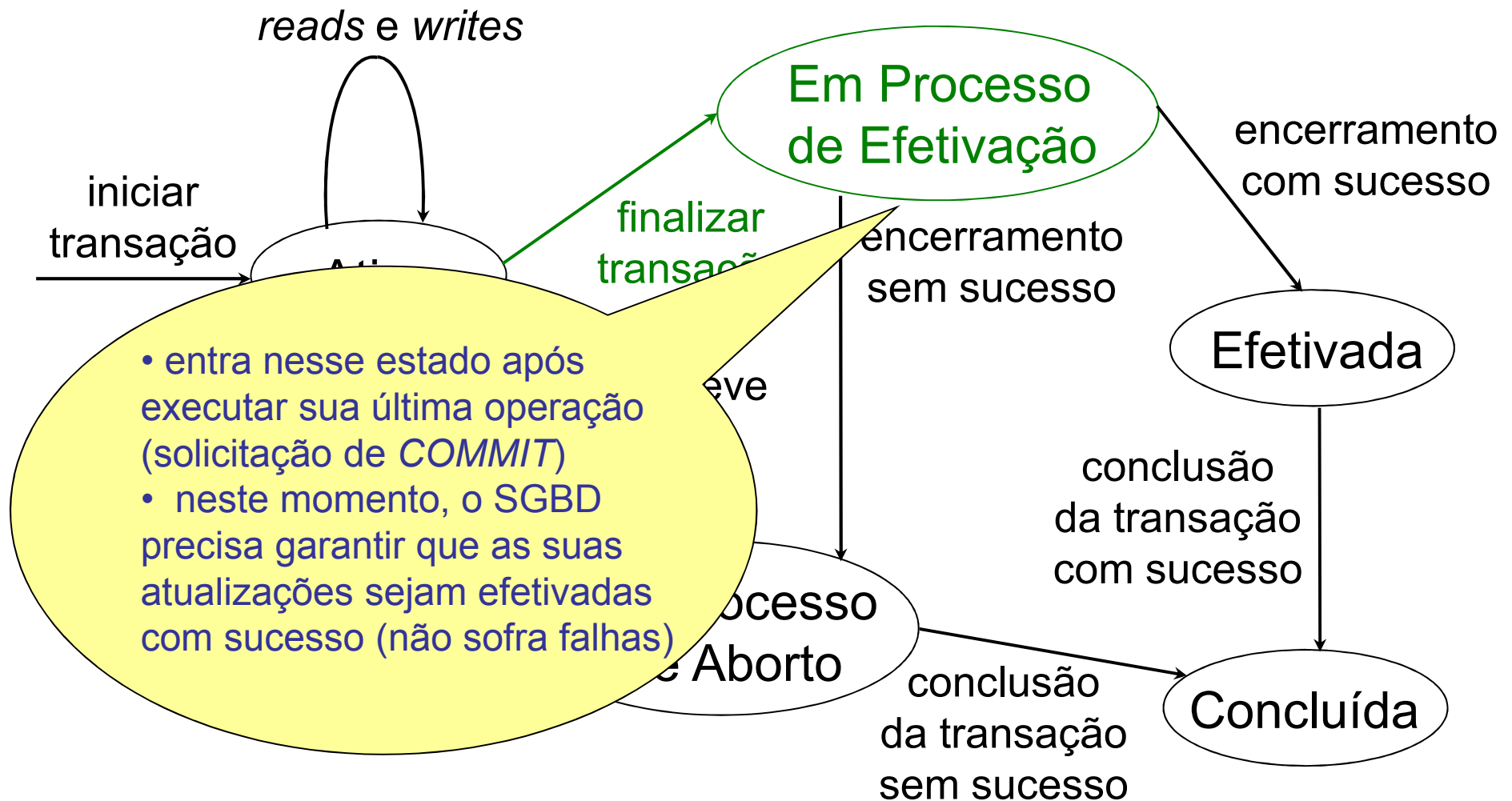


# Transição de Estados de uma Transação

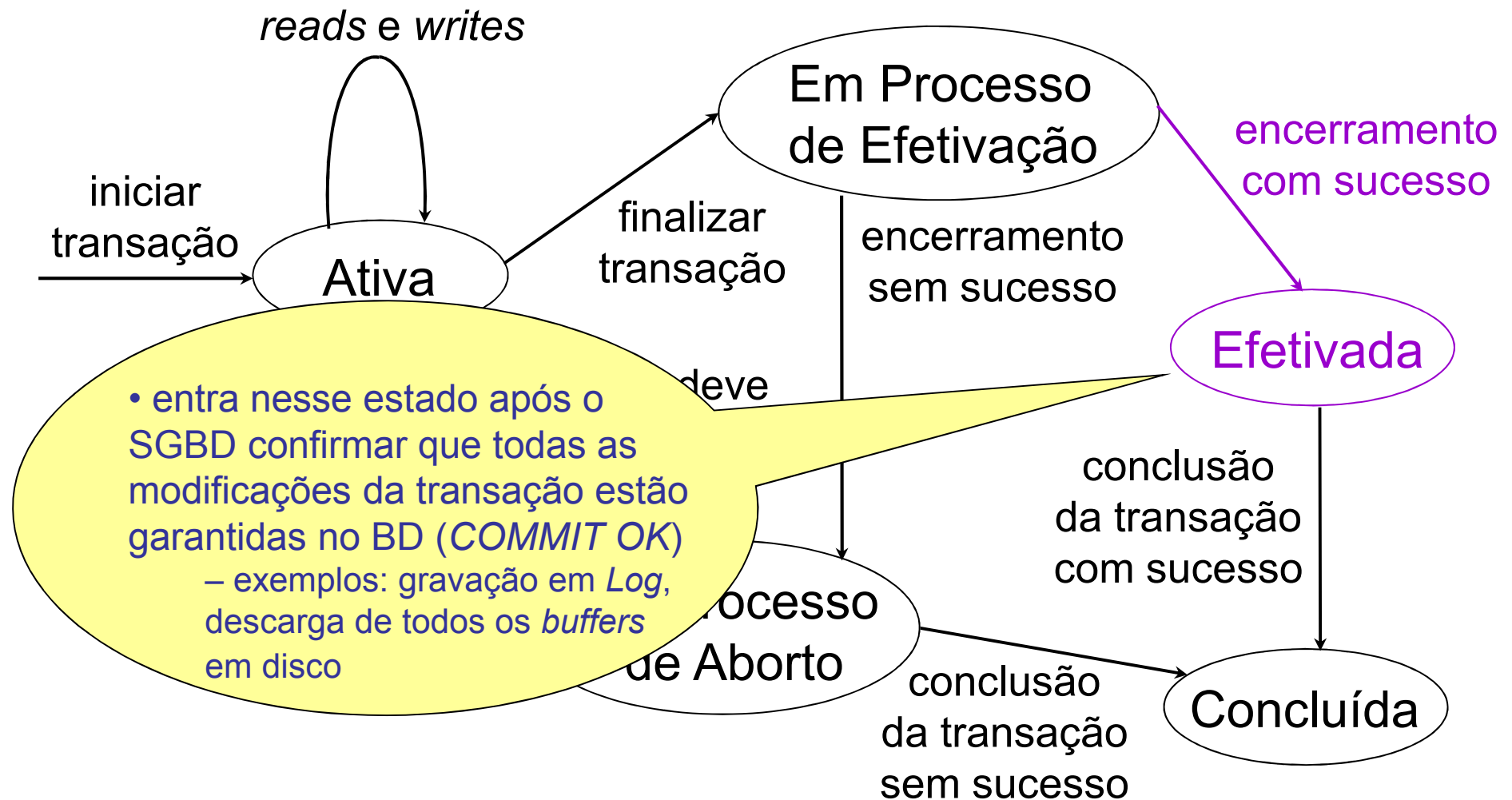




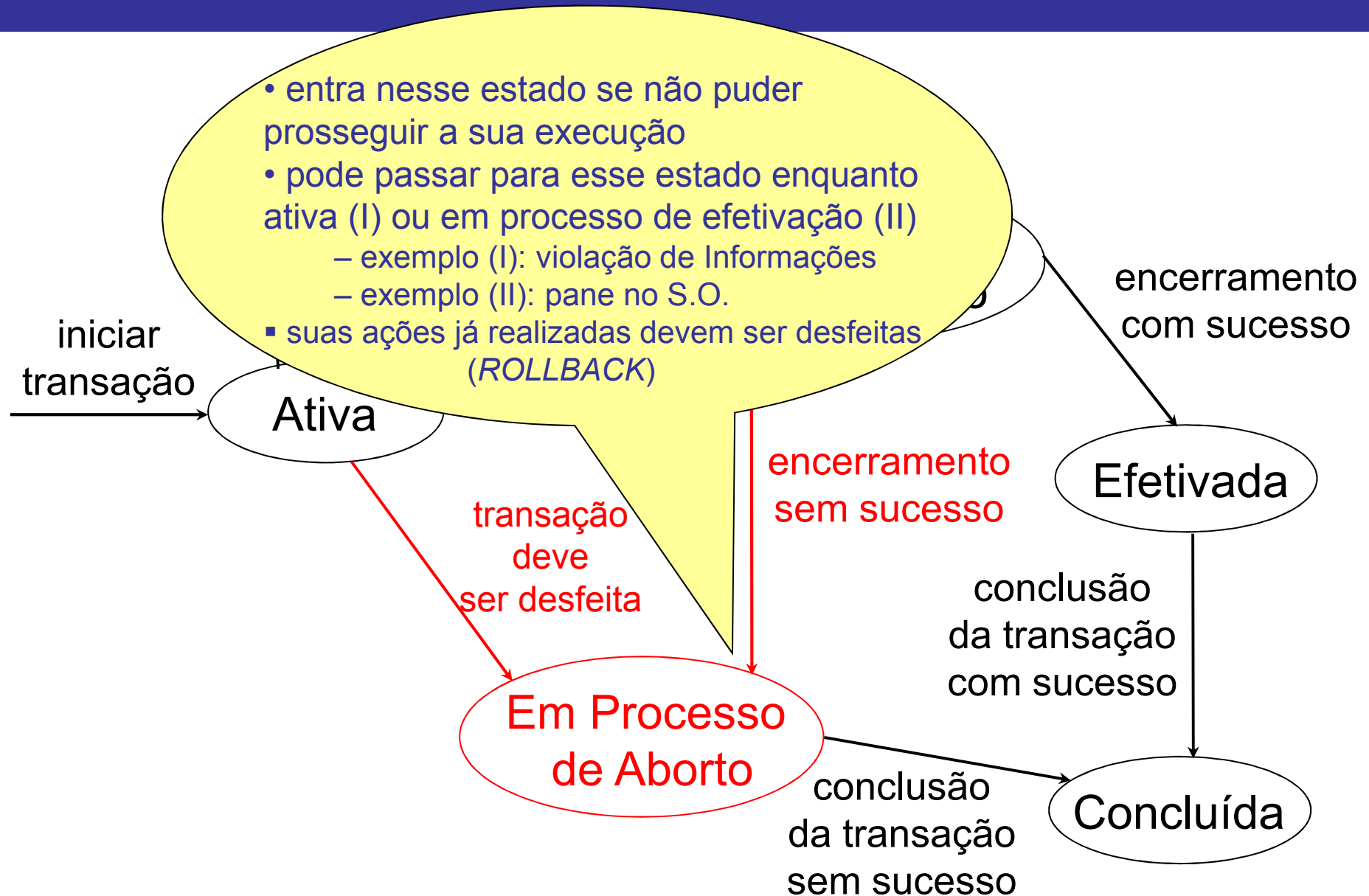
# Transição de Estados de uma Transação



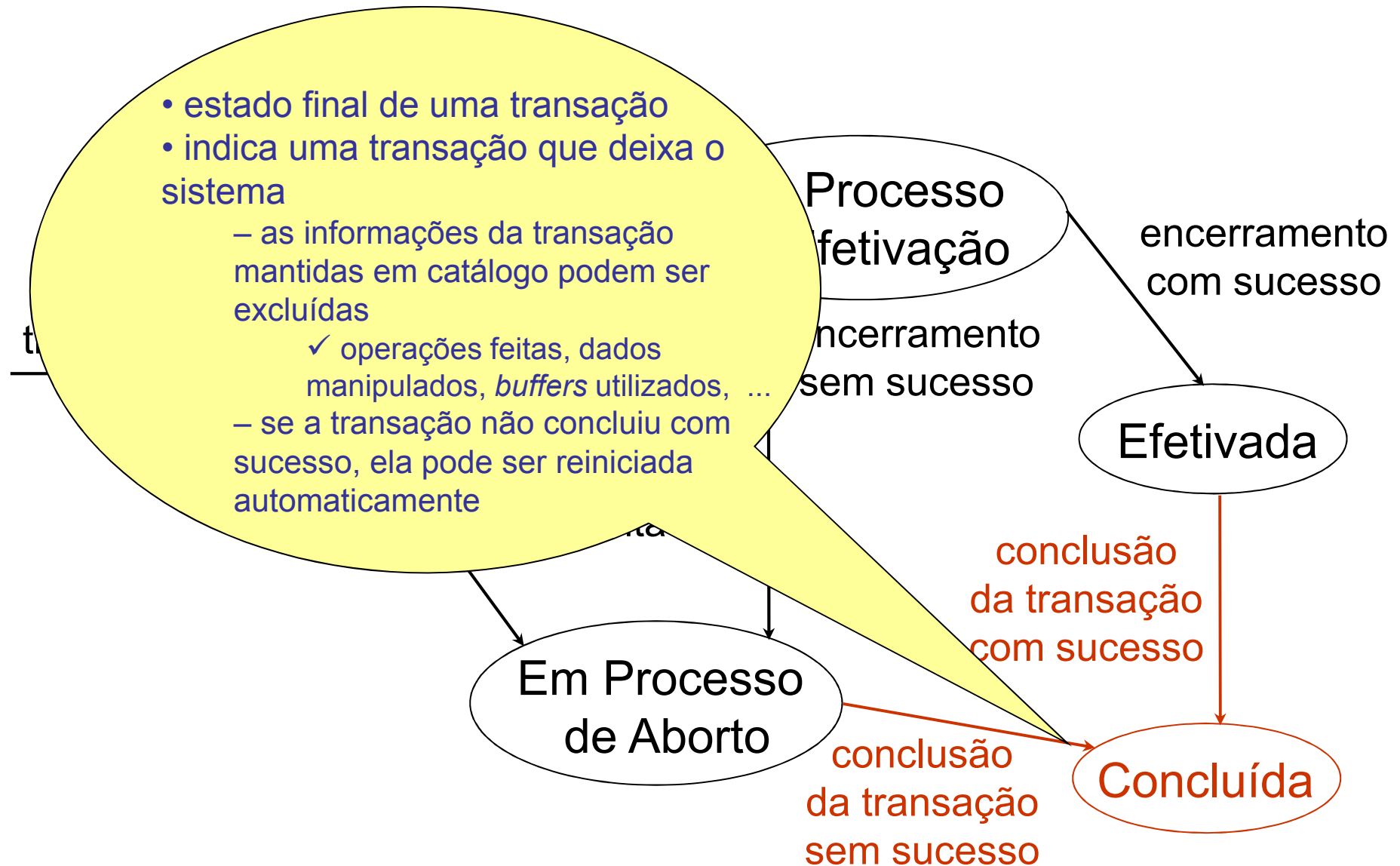
# Transição de Estados de uma Transação



# Transição de Estados de uma Transação



# Transição de Estados de uma Transação



# LOG

- O sistema mantém um log para registrar todas as operações de transação que afetam os valores do BD.
- Também mantém informações da transação que possam ser necessárias para permitir a recuperação de falhas.
- O log é um arquivo sequencial, apenas para inserção.
- Primeiramente as entradas do log são acrescentadas a um buffer.
- Quando o buffer é preenchido, ou quando ocorre certas condições, o buffer é anexado ao final do arquivo de log.
- O arquivo de log do disco é periodicamente copiado para arquivamento.

# Registros de Log

As entradas que são gravadas para o arquivo de log:

- **[start\_transaction, T]** → Indica que a transação T foi iniciada;
- **[write\_item, T, X, valor\_antigo, valor\_novo]** → Indica que a transação 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;
- **[commit, T]** → Indica que a transação T foi concluída com sucesso e afirma que seu efeito pode ser confirmado no BD;
- **[abort, T]** → Indica que a transação foi abortada.

Se ocorrer uma falha no sistema, pode-se recuperá-lo para um estado coerente do BD ao examinar o log.

# Propriedades de uma Transação

- Para assegurar integridade dos dados, exigimos que o sistema de banco de dados mantenham as seguintes propriedades das transações: ACID
- A tômica: para o mundo externo, a transação ocorre de forma indivisível.
- C onsistência: a transação não viola invariantes de sistema.
- I solamento: transações concorrentes não interferem entre si (*serializable*).
- D urabilidade: os efeitos de uma transação terminada com *commit* são permanentes.

Commit - encerra a transação (solicita efetivação das suas ações)

# 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 do SGBD
  - desfazer as ações de transações parcialmente executadas

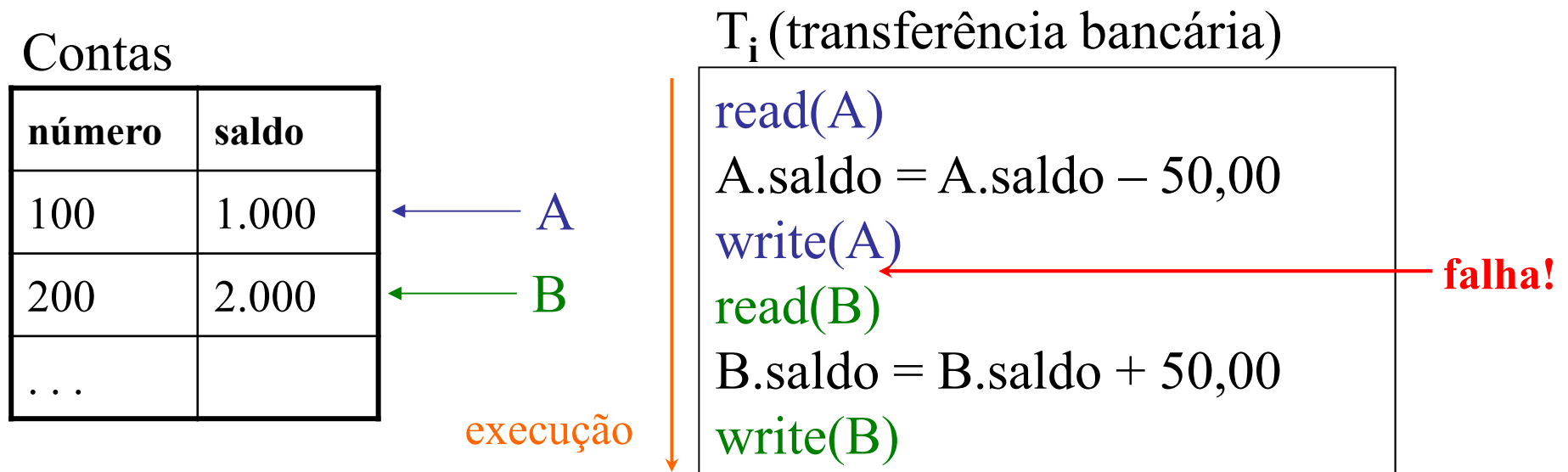


# Situação Problema

- Suponhamos que, antes da execução da transação  $T_i$  os valores das contas A e B sejam 1.000 e 2.000 reais, respectivamente. Agora suponha que, durante a execução da transação  $T_i$  uma falha (*falta de energia, falha na máquina e erros de software*) aconteceu impedindo  $T_i$  de se completar com sucesso. Suponha que a falha ocorrida tenha sido depois da operação `write(A)`, mas antes da operação `write(B)`. Nesse caso os valores das contas A e B refletidas no banco são A: 950 e B: 2000 reais. Como resultado da falha sumiram 50 reais.
- Chamamos esse estado de inconsistente. Devemos assegurar que essas inconsistências sejam imperceptíveis em um banco de dados
- Se a propriedade de atomicidade for garantida, todas as ações da transação serão refletidas no banco de dados ou nenhuma delas o será.

# Atomacidade

- Deve ser garantida, pois uma transação pode manter o BD em um estado inconsistente durante a sua execução.



• A idéia básica por trás da garantia da **ATOMICIDADE** é a seguinte: O SGBD mantém um registro (em disco) dos antigos valores de quaisquer dados sobre os quais a transação executa uma gravação e, se a transação não completar, os valores antigos são restabelecidos para fazer com que pareça que nunca foi executada. Assegurar a atomicidade é função do próprio sistema de BD.

# Consistência

- Uma transação sempre conduz o BD de um estado consistente para outro estado também consistente
- Responsabilidade do programador da aplicação que codifica a transação.

# Isolamento

- No contexto de um conjunto de transações concorrentes, a execução de uma transação  $T_i$  deve funcionar como se  $T_i$  executasse de forma isolada
  - $T_i$  não deve sofrer interferências de outras transações executando concorrentemente
- A propriedade de isolamento de uma transação garante que a execução simultânea de transação resulte em uma situação no sistema equivalente ao estado obtido caso as transações tivessem sido executadas uma de cada vez em qualquer ordem.

# Isolamento

T <sub>1</sub>	T <sub>2</sub>
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 - A write(B)

escalonamento válido

T <sub>1</sub>	T <sub>2</sub>
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 - A write(B)

T<sub>1</sub> interfere  
em T<sub>2</sub>

T<sub>2</sub> interfere  
em T<sub>1</sub>

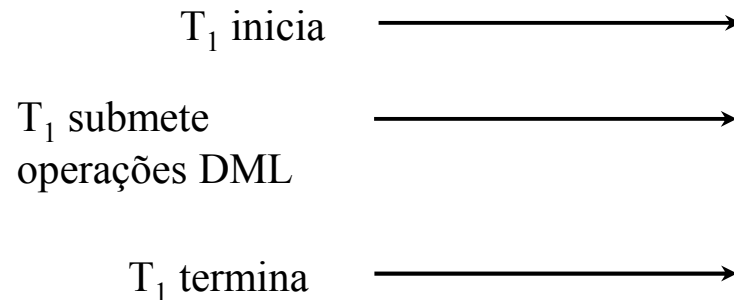
escalonamento inválido

# Durabilidade ou Persistência

- 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

# Gerência Básica de Transações

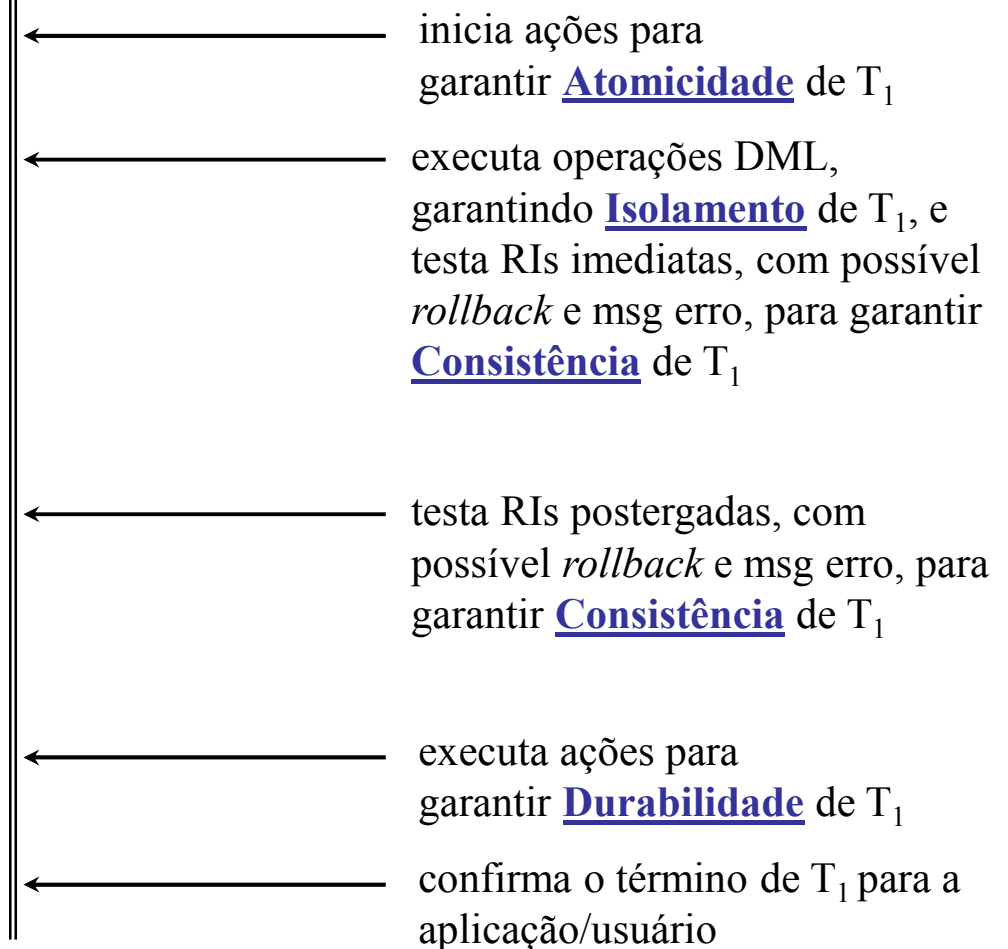
## Ações da Aplicação ou Usuário



**DML (Linguagem de Manipulação dos Dados)**  
Permite ao usuário acessar ou manipular os dados, vendo-os da forma como são definidos no nível de abstração mais alto do modelo de dados utilizado.

**ROLLBACK** - solicita que as ações da transação sejam desfeitas.

## Ações do SGBD



# Controle de Concorrência - Transações

- **SGBD**
  - sistema multiusuário em geral
    - diversas transações executando simultaneamente
- Garantia de **isolamento** de Transações
  - 1ª solução: uma transação executa por vez
    - **Escalonamento serial** de transações
    - solução bastante ineficiente!
      - várias transações podem esperar muito tempo para serem executadas
      - CPU pode ficar muito tempo ociosa
        - » enquanto uma transação faz I/O, por exemplo, outras transações poderiam ser executadas
    - solução mais eficiente
      - execução concorrente de transações de modo a preservar o isolamento
        - » escalonamento (schedule) não-serial e íntegro



# Controle de Concorrência - Transações

- As técnicas de controle de concorrência garantem que várias transações submetidas por vários usuários não interfiram umas nas outras de forma a produzir resultados inconsistentes.
- Um schedule (escala) representa uma seqüência de operações realizadas por transações concorrentes.

# Controle de Concorrência - Transações

- Os dois schedules(escalas) apresentados são seriais, ou seja, as transações são executadas de forma seqüencial, uma seguida da outra.
- Entretanto a execução de schedules seriais limita o acesso concorrente aos dados e, por conseqüência, diminui o throughput (quantidade de trabalho realizado em um intervalo de tempo).

T1	T2
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)

T1	T2
	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)	

# Controle de Concorrência - Transações

- Nem todas as execuções concorrentes resultam em um estado correto.
- Esse estado final é um estado **inconsistente**, pois a soma de  $A + B$  não é preservada na execução das duas transações

T1	T2
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)	

**Escala Concorrente ou execução não-serial**

# Scheduler

- Responsável pela definição de escalonamentos não-seriais de transações
- “Um escalonamento  $\underline{E}$  define uma ordem de execução das operações de várias transações, sendo que *a ordem das operações de uma transação  $T_1$  em  $\underline{E}$  aparece na mesma ordem na qual elas ocorrem isoladamente em  $T_1$* ”
- Problemas de um escalonamento não-serial mal definido (inválido)
  - atualização perdida (*lost-update*)
  - leitura suja (*dirty-read*)

# Atualização Perdida (*lost-update*)

- Uma transação  $T_1$  grava em um dado atualizado por uma transação  $T_2$

T1	T2
read(A)	
$A = A - 50$	
	read(C)
	$A = D + 10$
write(A)	
read(B)	
	write(A)
$E = A + 30$	
write(B)	

← a atualização de A por T1 foi perdida!

# Leitura Suja (*dirty-read*)

- $T_1$  atualiza um dado A, outras transações posteriormente lêem A, e depois  $T_1$  falha

Neste ponto, a transação T1 falha e deve retornar o valor de A para o seu valor antigo;

T1	T2
read(A)	
$A = A - 20$	
write(A)	
	read(A)
	$A = A + 10$
	write(A)
read(Y)	
abort( )	

T2 leu um valor de A que não será mais válido!

# Scheduler

- O padrão *Scheduler (selecionador)* (Lea, 1997) tem como objetivo controlar a ordem em que requisições são escalonadas, encadeando a ordem de execução das requisições em um processador. A partir deste padrão, um processador, ao receber uma requisição, não possui mais controle sobre o momento de sua execução. Para isto, esta requisição deve ser repassada a um escalonador que, ao implementar alguma política de controle de execução, determinará o momento apropriado para a execução da requisição no processador.

# Scheduler

- Deve evitar escalonamentos inválidos
  - exige análise de **operações em conflito**
    - operações que pertencem a transações diferentes
    - transações acessam o mesmo dado
    - pelo menos uma das operações é *write*



# Scheduler X Recovery

- *Scheduler* deve cooperar com o *Recovery*!
- Categorias de escalonamentos considerando o grau de cooperação com o *Recovery*
  - recuperáveis X não-recuperáveis
  - permitem aborto em cascata X evitam aborto em cascata
  - estritos X não-estritos

# Transações em SQL

- Uma linguagem de Manipulação de dados deve possuir um construtor para especificar o conjunto de ações que constitui uma transação.
- Por *default*, todo comando individual é considerado uma transação
  - exemplo: DELETE FROM Pacientes
    - exclui todas ou não exclui nenhuma tupla de pacientes, deve manter o BD consistente, etc
- SQL Padrão (SQL-92)
  - SET TRANSACTION
    - inicia e configura características de uma transação
  - COMMIT [WORK]
    - encerra a transação (solicita efetivação das suas ações)
  - ROLLBACK [WORK]
    - solicita que as ações da transação sejam desfeitas

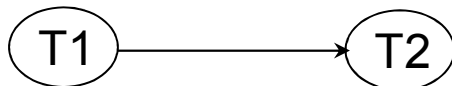
# Transações em SQL

- Principais configurações (SET TRANSACTION)
  - modo de acesso
    - **READ** (somente leitura), **WRITE** (somente atualização) ou **READ WRITE** (ambos - *default*)
  - nível de isolamento
    - indicado pela cláusula **ISOLATION LEVEL** *nível*
    - *nível* para uma transação  $T_i$  pode assumir
      - **SERIALIZABLE** ( $T_i$  executa com completo isolamento - *default*)
      - **REPEATABLE READ** ( $T_i$  só lê dados efetivados e outras transações não podem escrever em dados lidos por  $T_i$ ) – pode ocorrer que  $T_i$  só consiga ler alguns dados que deseja
      - **READ COMMITTED** ( $T_i$  só lê dados efetivados, mas outras transações podem escrever em dados lidos por  $T_i$ )
      - **READ UNCOMMITTED** ( $T_i$  pode ler dados que ainda não sofreram efetivação)

# Grafo de Precedência

escalonamento serializável  $E1$

T1	T2
read(X)	
$X = X - 20$	
write(X)	
	read(X)
	$X = X + 10$
	write(X)
read(Y)	
$Y = Y + 20$	
write(Y)	



escalonamento não-serializável  $E2$

T1	T2
read(X)	
$X = X - 20$	
	read(X)
	$X = X + 10$
write(X)	
read(Y)	
	write(X)
$Y = Y + 20$	
write(Y)	

