Práticas de Banco de Dados

Parte 3 – Transações

Professor Eduardo xavier

Transações: Conceitos

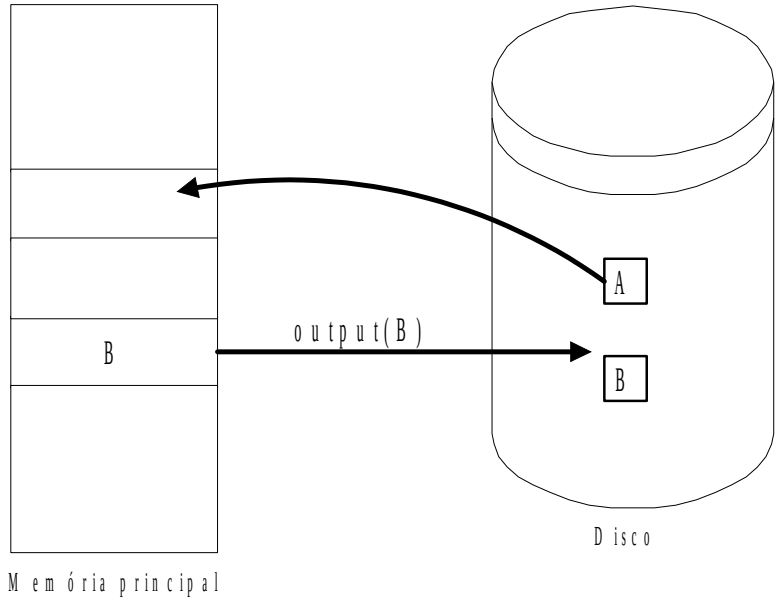
- Definição
  - Uma transação é um mecanismo para se descrever unidade lógica de processamento em banco de dados.
- Por que usar?
  - SGBDs são projetados, em geral, para funcionamento multiusuário.
  - É preciso garantir que as unidades lógicas de processamento sejam executadas completamente e sem interferência de outras operações.
  - É desejável que qualquer pedaço de informação possa ser reconstruído a partir de outras informações armazenadas.

Transações: Conceitos

- Uma transação pode ser implícita ou explícita
  - **Implícita**: Não necessita de declaração formal via comando específico para determinar sua existência.
    - Exemplo: Um comando único de INSERT, UPDATE ou DELETE
  - **Explícita**: Exige a declaração formal via comando específico de sua existência para ser reconhecida como tal.
    - Exemplo: A transação completa é composta de diversos comandos de INSERT, UPDATE ou DELETE que precisam ser executados juntos para que a tarefa seja cumprida.
    - A linguagem SQL tem comandos específicos para isso: BEGIN TRANSACTION e END TRASACTION (COMMIT)

Transações: Gerenciamento

- **Read(x)**
  - Encontra bloco do disco em que x está armazenado.
  - Copia bloco para buffer da memória principal.
  - Copia o ítem x do buffer para a variável do programa.
- **Write (x)**
  - Encontra bloco do disco em que “x” está armazenado.
  - Copia o bloco para buffer da memória principal.
  - Copia x da variável do programa para o buffer.
  - Armazena o bloco atualizado do buffer para o disco (imediatamente ou mais tarde)



Transações: Gerenciamento

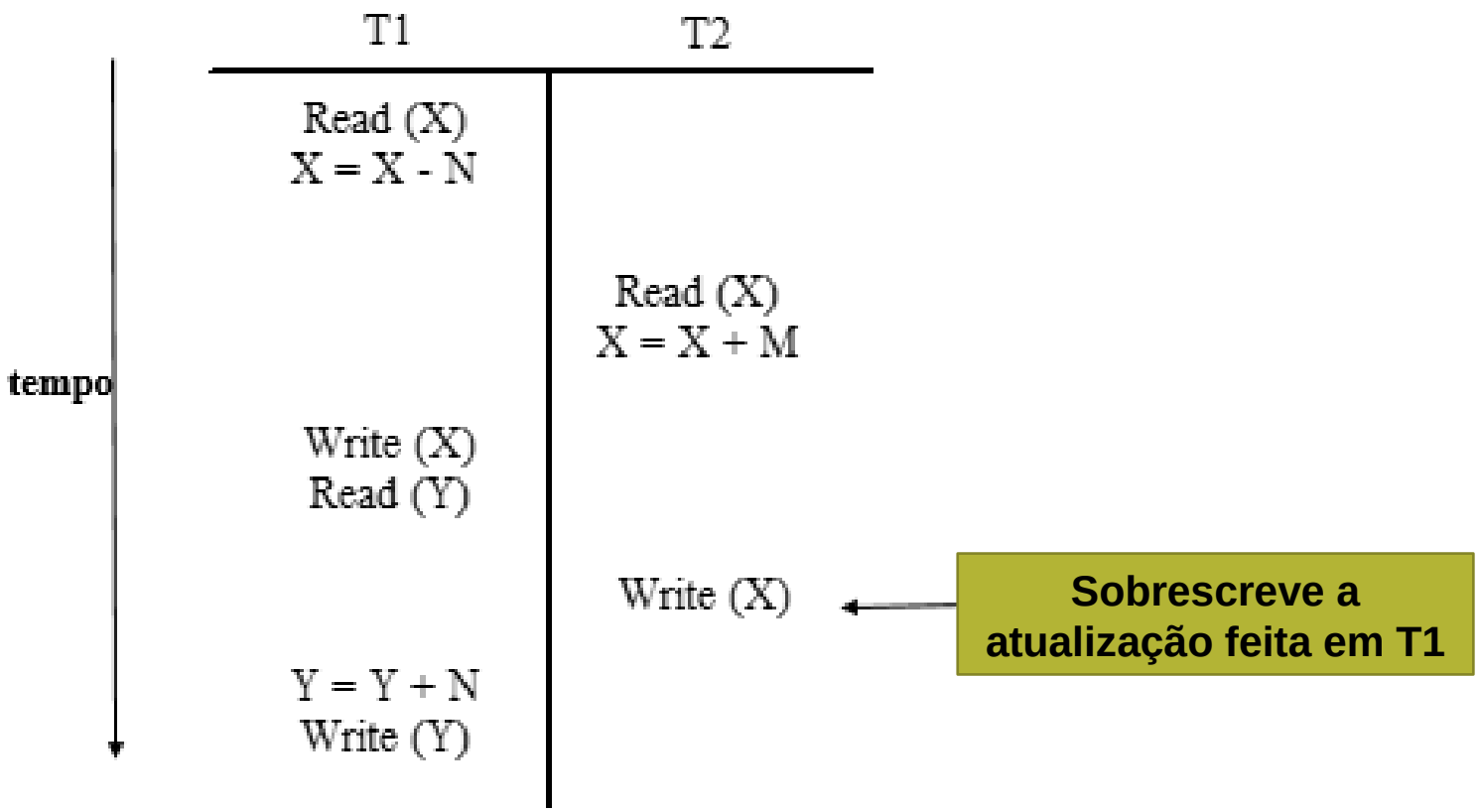
- **Controle de concorrência**
  - Para garantir o processamento de várias transações o SGBD deve ter alguma estratégia que implemente um controle de concorrência pelos recursos manipulados pelas transações.
  - O controle de concorrência coordena as ações de processos que atuam em paralelo (ou de forma concorrente) acessando dados compartilhados e que possuem potencial para interferir uns com os outros.
- **Recuperação**
  - É necessário existir algum mecanismo de recuperação que atua em caso de falhas na transação, visando reestabelecer a integridade dos dados.
  - Esse mecanismo assegura que falhas de hardware ou software não corrompem os dados persistentes.

Transações: Controle de Concorrência

- Se duas transações acessam o mesmo item do banco e suas operações são entrelaçadas é necessário o controle da concorrência.
- Quais os problemas que o controle de concorrência tenta evitar?
  - **Problema de perda de atualização**
  - **Problema da dependência de atualização não confirmada**
  - **Problema de função de agregação incorreta (sumário incorreto)**

Transações: Controle de Concorrência

- **Perda de atualização**

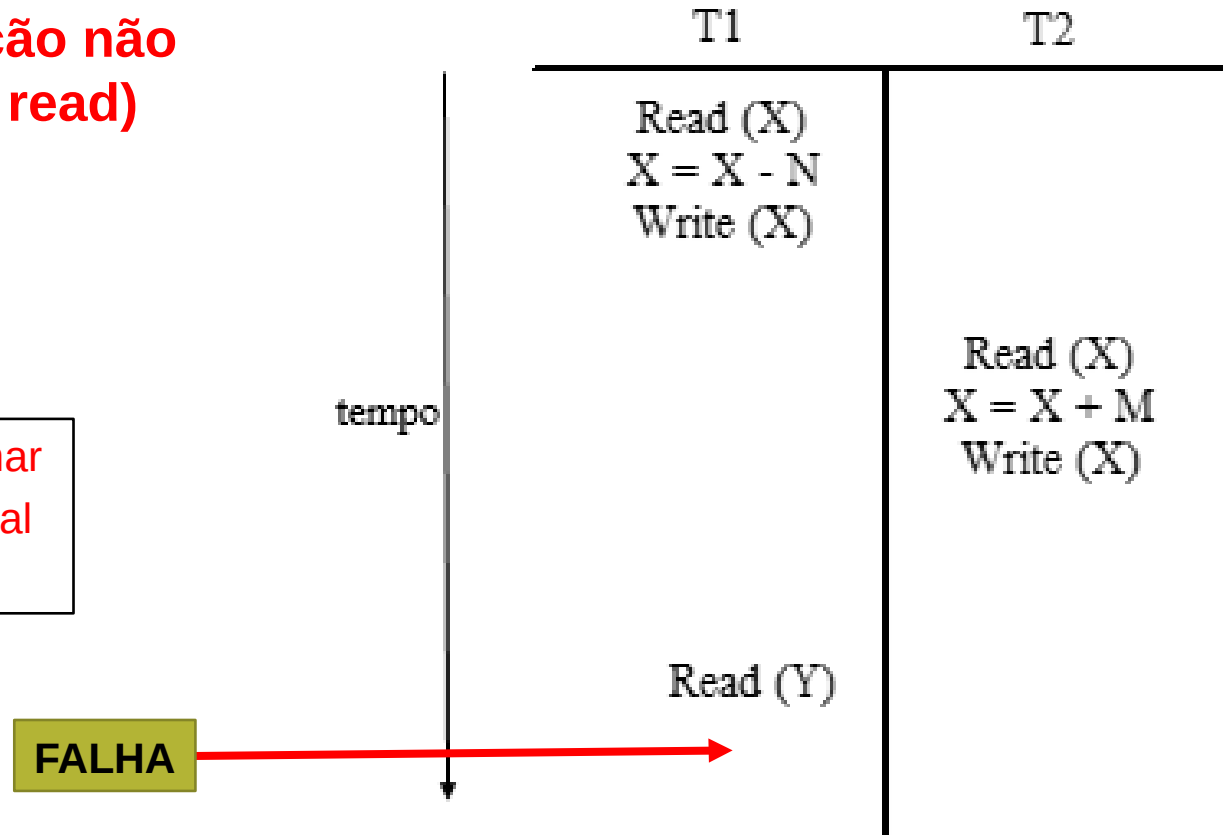


O item X tem um valor incorreto pois sua atualização através de T1 foi perdida

Sobrescreve a atualização feita em T1

Transações: Controle de Concorrência

- **Dependência de uma atualização não confirmada (leitura suja / dirty read)**



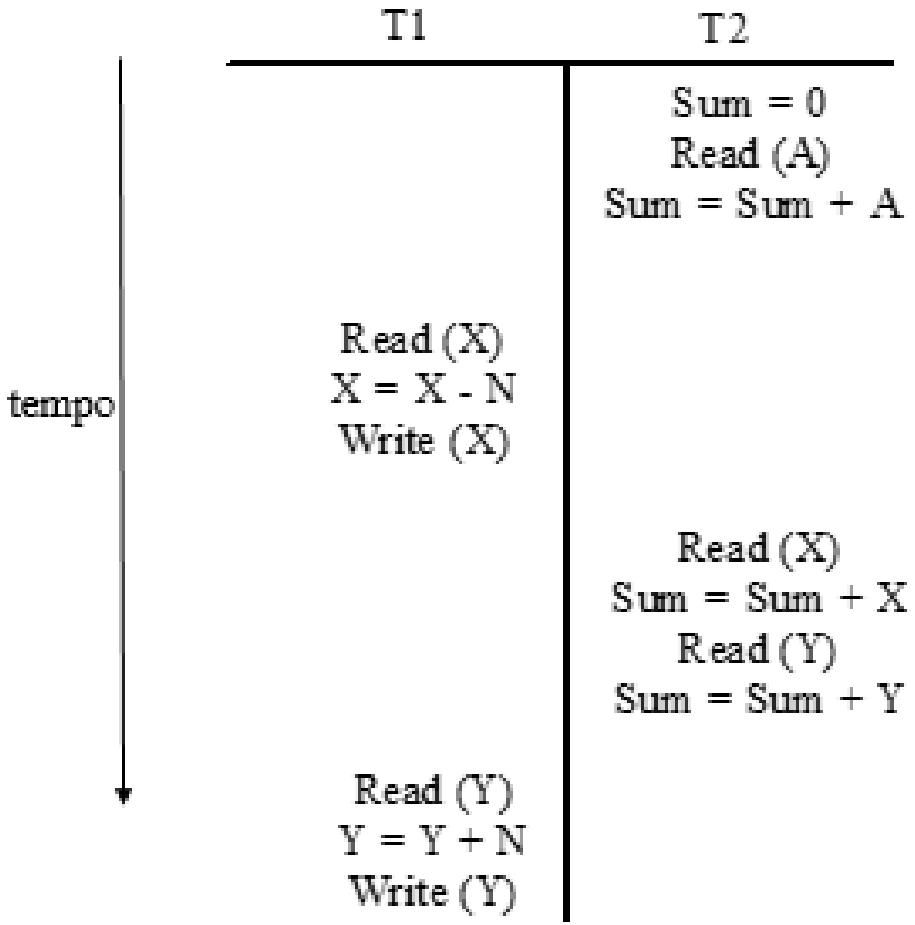
A transação T1 falha e deve retornar o valor de **X** para seu valor original mas T2 já leu o valor incorreto

Transações: Controle de Concorrência

- **Sumário incorreto**

T2 lê X depois que ele é subtraido e lê Y antes que seja adicionado.

O resultado será uma soma incorreta



O sumário calculado neste momento se torna incorreto assim que T1 alterar o valor de Y

Transações: Controle de Concorrência

- **BEGIN TRANSACTION**
  - Informa explicitamente ao SGBD que uma nova transação vai iniciar.
- **COMMIT TRANSACTION / END TRANSACTION**
  - Informa explicitamente ao SGBD que uma determinada transação que está em andamento será encerrada com sucesso e que é preciso tomar providências para tornar as atualizações feitas permanentes.
- **ROLLBACK TRANSACTION**
  - Informa explicitamente ao SGBD que uma determinada transação que está em andamento será encerrada com erro e que é preciso toar providências para retornar os dados à situação anterior à execução da transação (desfazer as alterações realizadas).

Transações: Recuperação

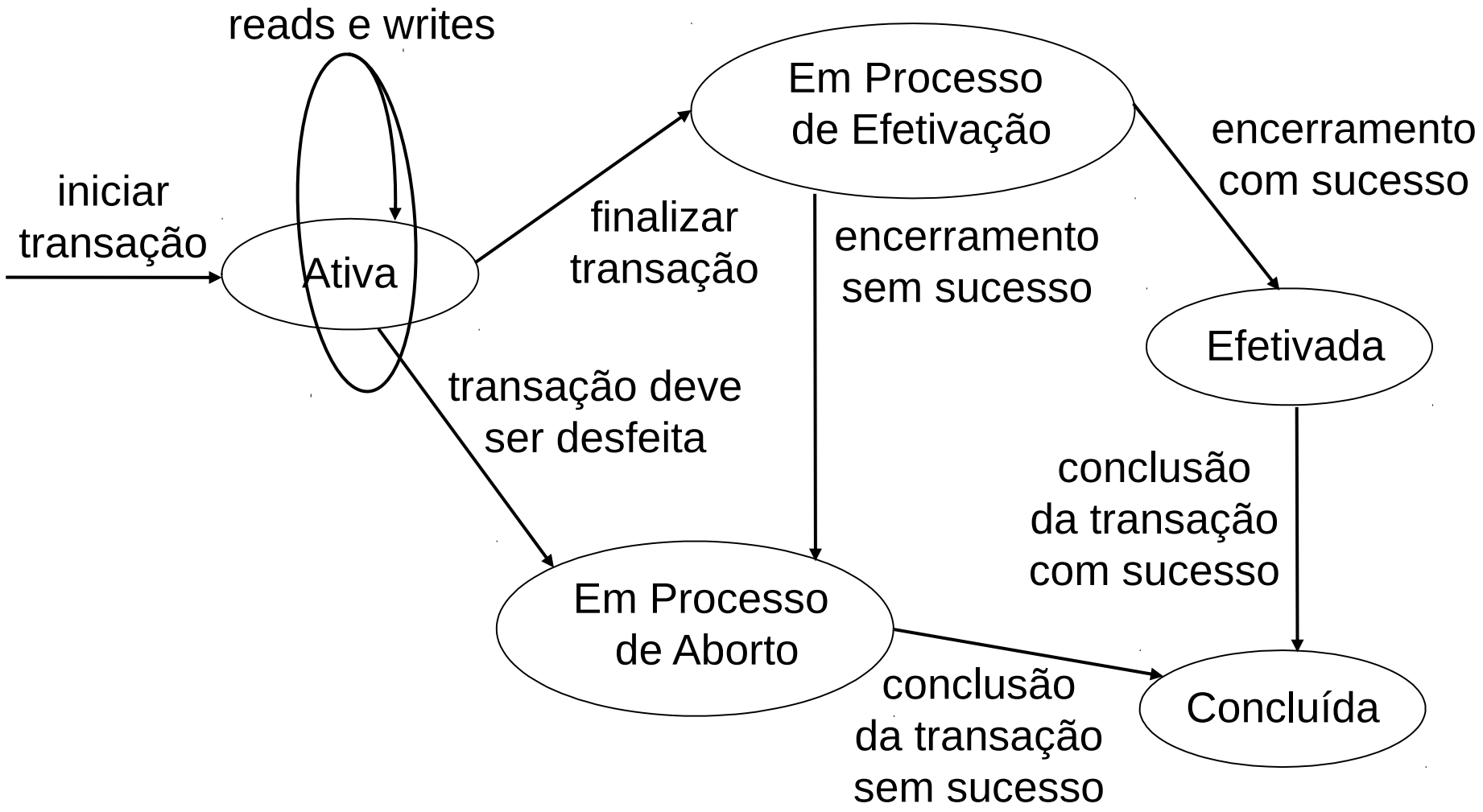
- Se todas as operações na transação são completadas com sucesso as alterações nos dados devem ter efeito permanente no banco de dados
- Se uma operação falhar, nada relacionado a esta transação terá efeito no banco de dados.
- O que pode causar falhas?
  - Erros de hardware, software ou rede
  - Erros de sistemas ou transação
  - Condições de exceção detectadas na transação

Transações: Estados

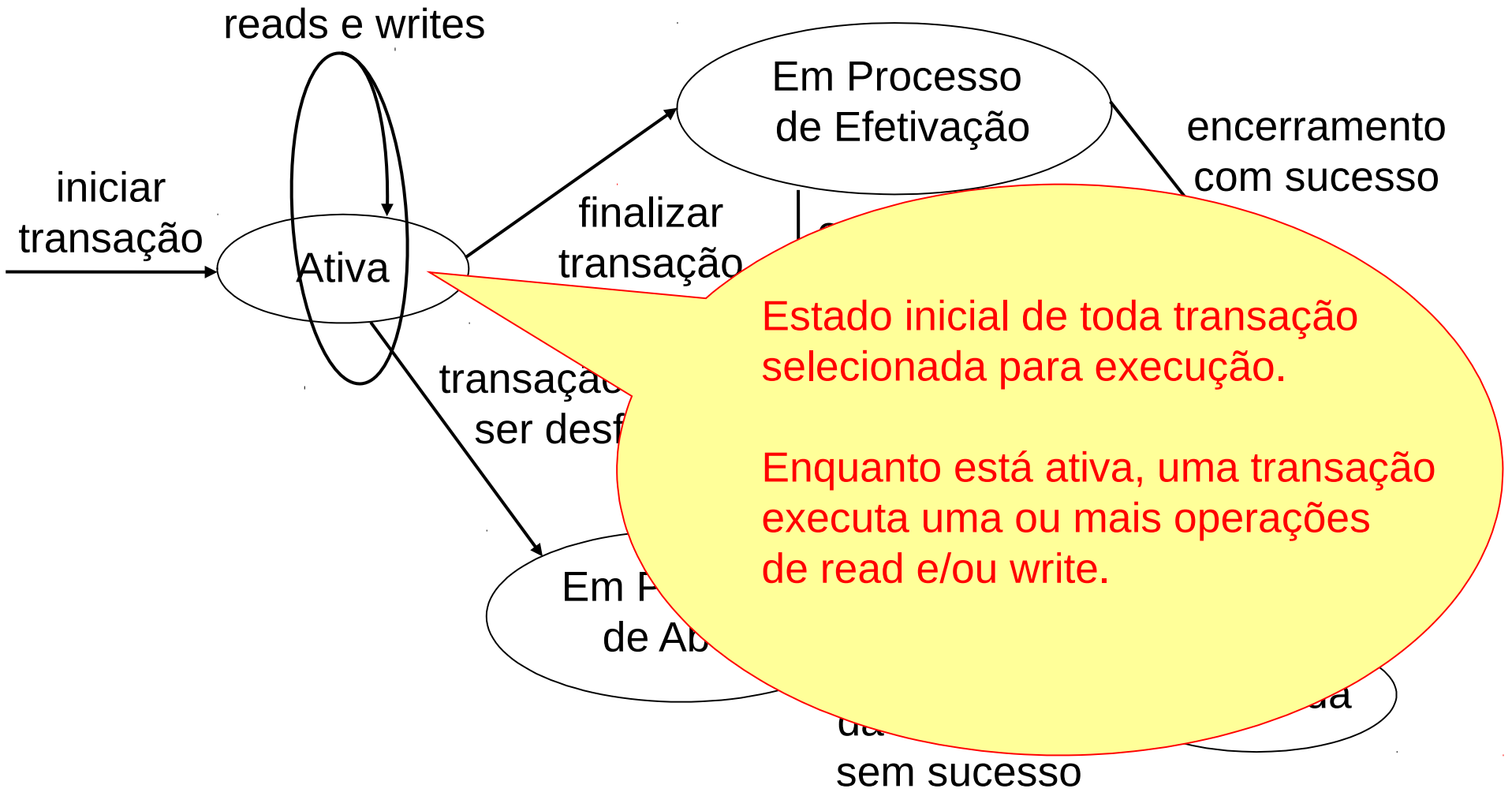
- Uma transação é sempre monitorada pelo SGBD quanto ao seu estado.
  - Que operações a transação já fez?
  - Que operações a transação já concluiu?
  - A transação deve abortar ou seguir executando?

- Estados de uma transação:
  - Ativa
  - Em processo de efetivação
  - Efetivada
  - Em processo de aborto
  - Concluída

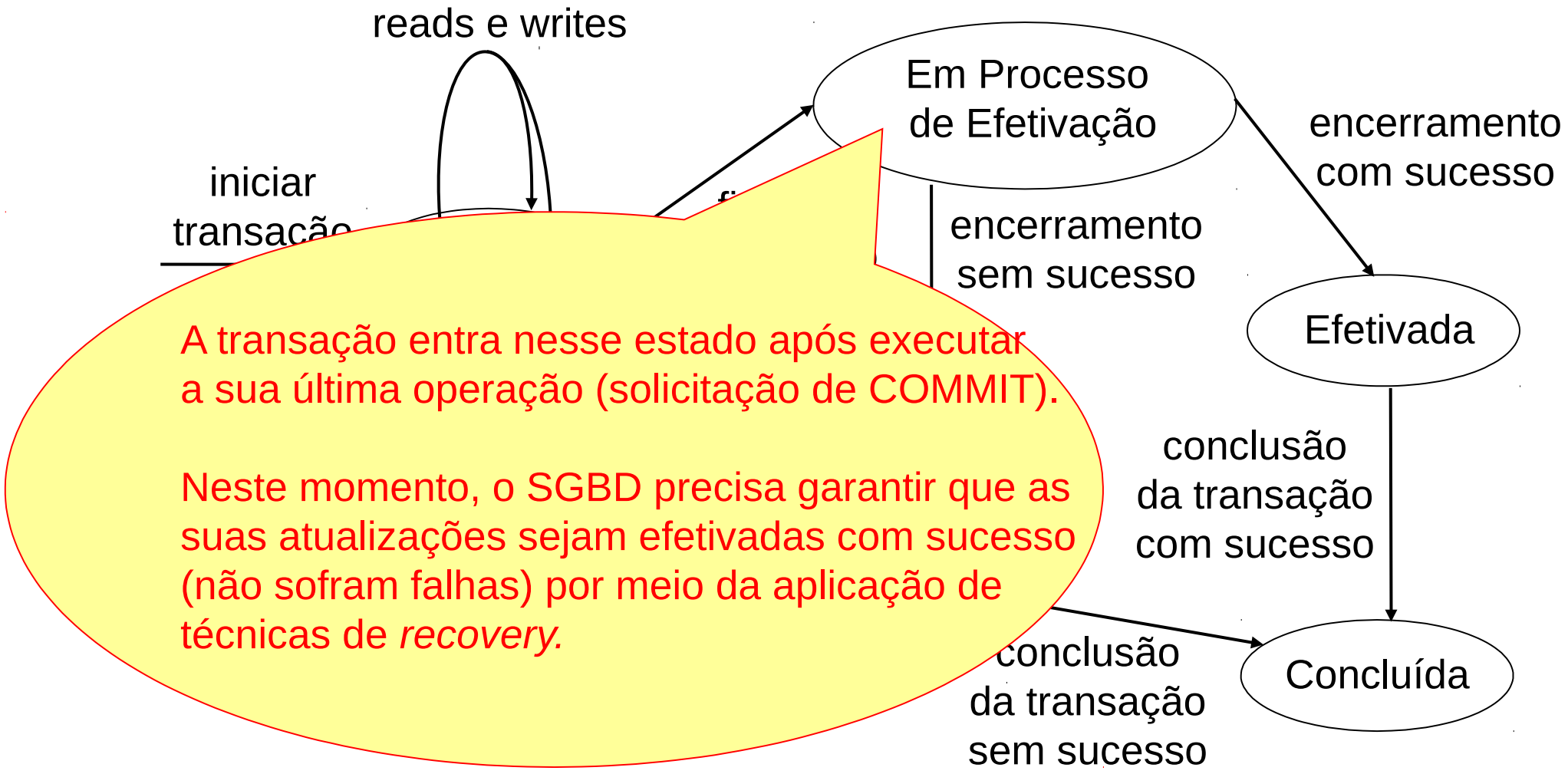
Transações: Estados



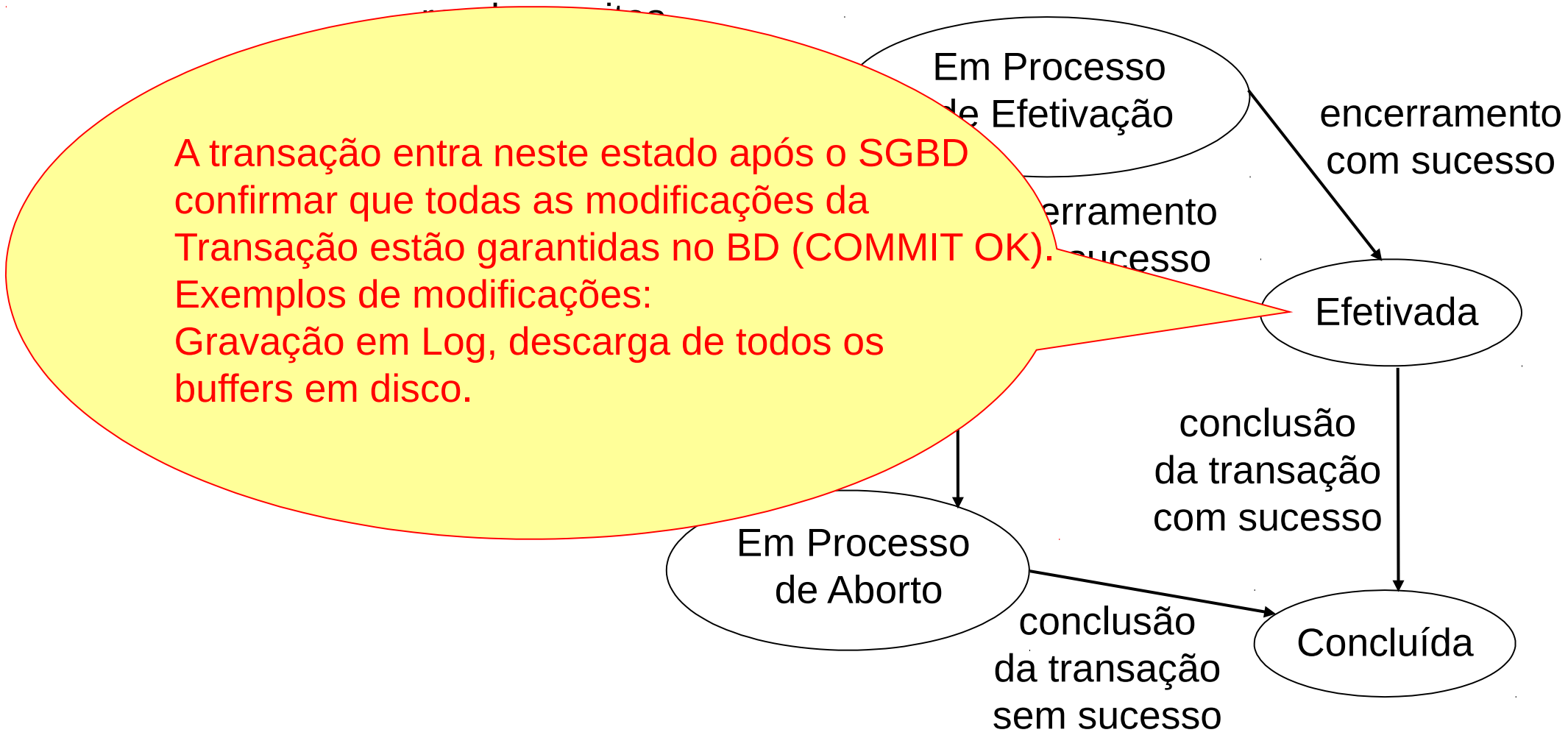
Transações: Estados



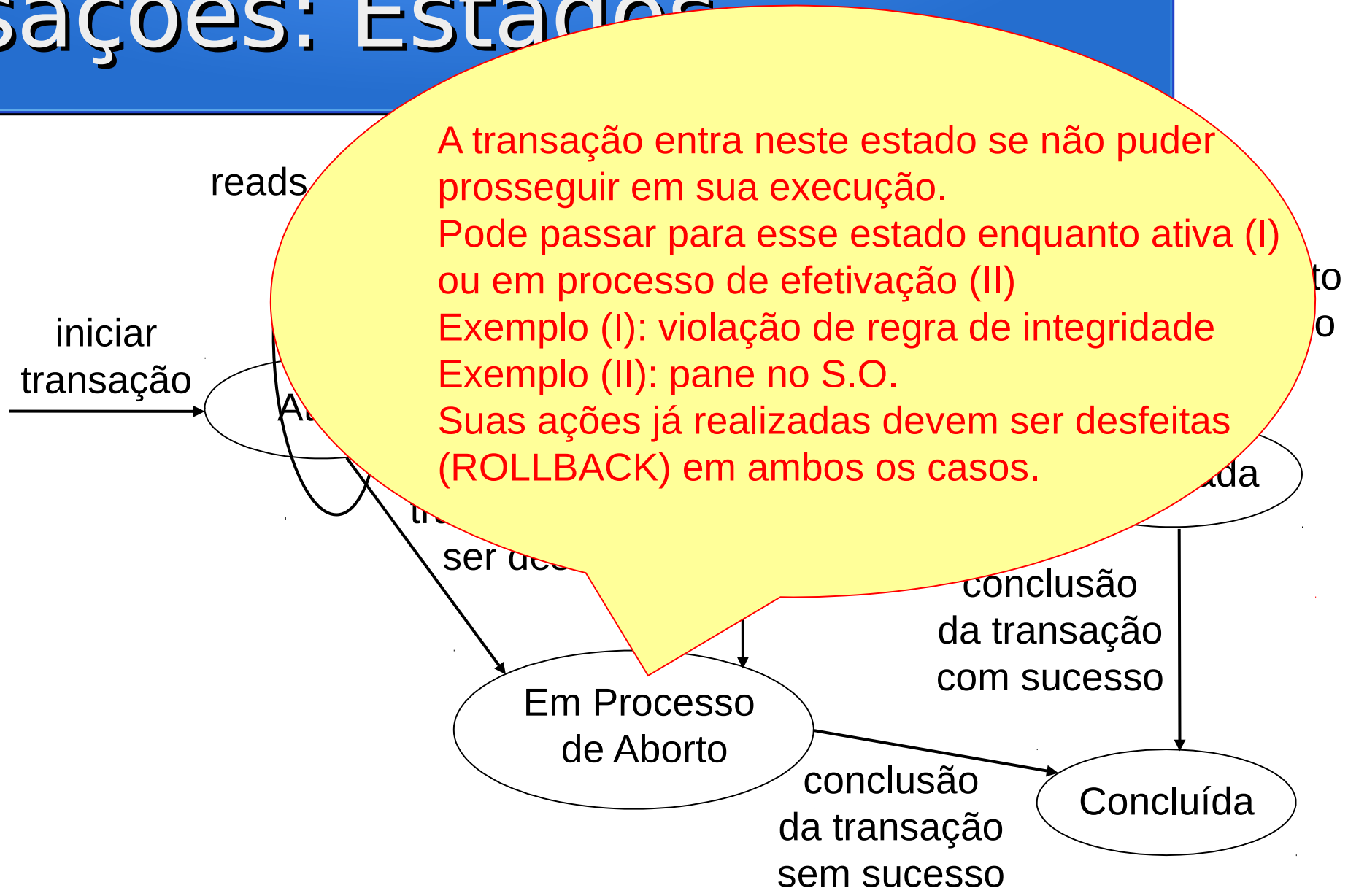
Transações: Estados



Transações: Estados

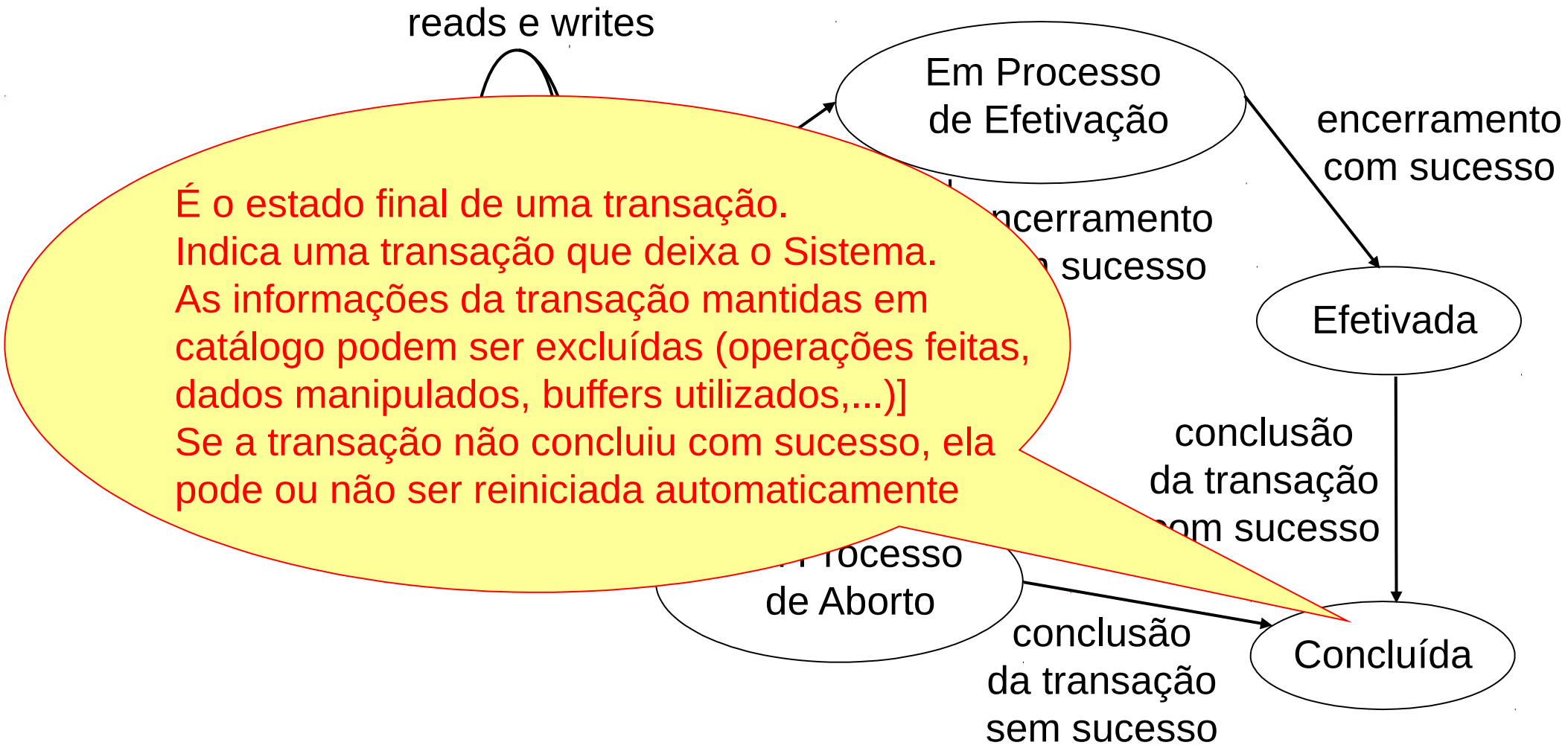


Transações: Estados



A transação entra neste estado se não puder prosseguir em sua execução.
Pode passar para esse estado enquanto ativa (I) ou em processo de efetivação (II)
Exemplo (I): violação de regra de integridade
Exemplo (II): pane no S.O.
Suas ações já realizadas devem ser desfeitas (ROLLBACK) em ambos os casos.

Transações: Estados



Transações: Propriedades

- Existem requisitos que sempre devem ser atendidos por uma transação relacional. Estes requisitos são propriedades chamadas de **propriedades ACID**.

  - **Atomicidade**
  - **Consistência**
  - **Isolamento**
  - **Durabilidade ou Persistência**

Transações: Propriedades

- **ATOMICIDADE**.
  - Princípio do “Tudo ou Nada”
    - Todas as operações da transação são efetivadas com sucesso no BD ou nenhuma delas se efetiva.
    - O objetivo é preservar a integridade do banco de dados garantindo que nenhuma transação é executada parcialmente.
  - Garantir essa propriedade é responsabilidade do **subsistema de recuperação contra falhas (subsistema de *recovery*)** do SGBD
    - Este subsistema é encarregado de desfazer as ações de transações parcialmente executadas.

Transações: Propriedades

- **CONSISTÊNCIA**
  - Uma transação sempre conduz o banco de dados de um estado consistente para outro estado também consistente
  - Responsabilidade conjunta do
    - **DBA** → define (ou ajuda os analistas a definir) todas as regras de integridade para garantir estados e transições de estado válidos para os dados.
      - exemplos: salário > 0; salário novo > salário antigo
    - **Subsistema de *recovery*** → desfazer as ações da transação que violou qualquer regra de integridade

Transações: Propriedades

- **ISOLAMENTO**
  - No contexto de um conjunto de transações concorrentes, a execução de uma transação *Tx* deve funcionar como se *Tx* executasse de forma isolada.
    - *Tx* não deve sofrer interferências de outras transações executando concorrentemente
  - Responsabilidade do **subsistema de controle de concorrência (*scheduler*)** do SGBD, que garante escalonamentos sem interferências

Transações: Propriedades

- **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 - A
	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 - A
	write(B) ←

T_1 interfere em T_2

T_2 interfere em T_1

escalonamento inválido

Transações: Propriedades

- **DURABILIDADE (ou PERSISTÊNCIA)**
  - Deve-se garantir que as modificações realizadas por uma transação que concluiu com sucesso persistam no banco de dados.
    - Nenhuma falha posterior ocorrida no BD deve perder essas modificações
  - Responsabilidade do **subsistema de *recovery***, que refaz as transações que executaram com sucesso em caso de falha no BD.

Transações:
Log de Transações

- Para ter capacidade de se recuperar de falhas que afetam a transação o sistema mantém um log.
- No log são registradas todas as operações das transações que afetam valores nos bancos de dados.
- O log é mantido em disco.

Transações:
Log de Transações

- Seja **T** a idenficação de uma transação :
  - [Start_transaction, T] □ marca o início da transação T.
  - [Write_item, T, X, old_value, new_value] □ indica que a transação T alterou o valor de X no banco de dados do valor antigo para o 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 transação T foi completada com sucesso e confirma a gravação no banco de dados.
  - [Abort, T] □ indica que a transaçõa T foi abortada.

Transações:
Log de Transações

- Como o log contém todos os registros das operações que alteraram qualquer item do banco, é possível desfazer (undo) o efeito destas operações ou refazer as operações até um certo ponto.

- Base para o processo de recuperação – Recovery

Transações:
Modificações no Banco de Dados

- **Modificações Adiadas**
  - O BD não é atualizado até que a transação alcance o commit.

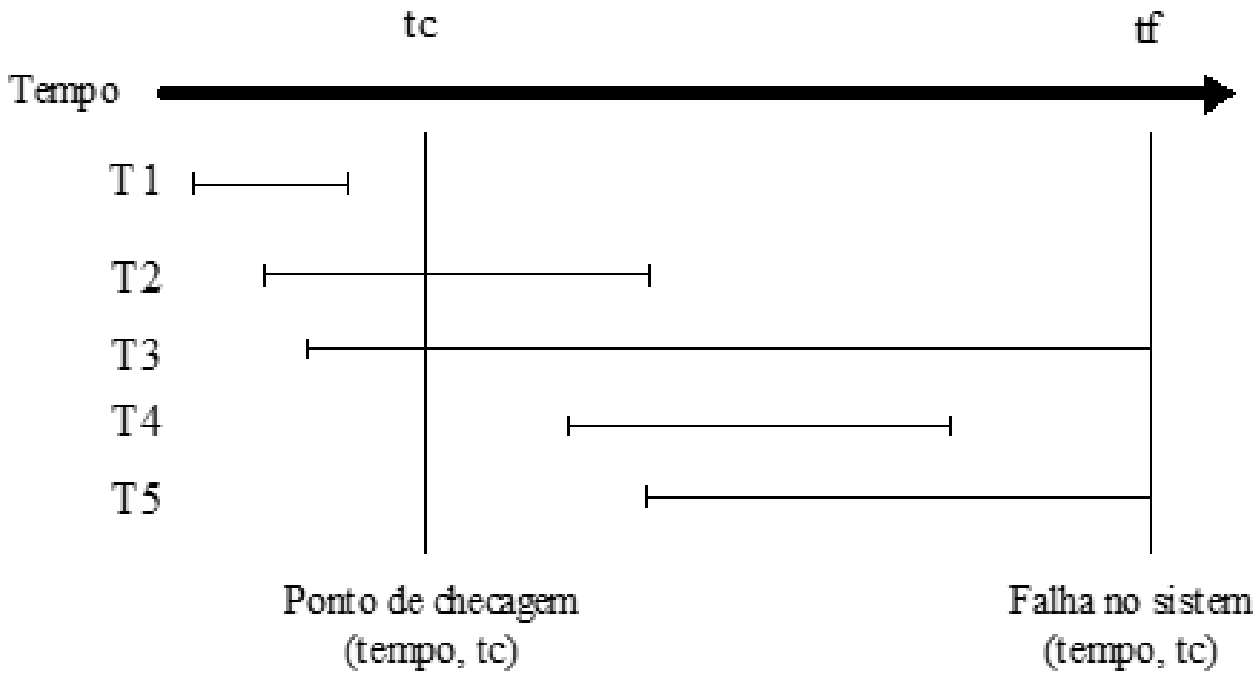
- **Modificações Imediata**
  - O BD é modificado antes de alcançar o commit.
  - As informações do log são gravadas em um meio estável a fim de possibilitar a recuperação.

Transações: Checkpoints

- Checkpoints são gravados periodicamente no *log* no ponto em que o SGBD grava no banco de dados todos os blocos que tenham sido modificados.
  - Todas as transações *commited* no log até este ponto não precisam mais ser refeitas no banco.
  - O gerenciador de recuperações deve decidir em que intervalo realizar o *checkpoint*.
    - O intervalo pode ser mantido em termos de tempo ou em termos de número de transações confirmadas desde o último checkpoint

Transações: Checkpoints

- Funcionamento:
  - Suspende todas as transações temporariamente.
  - Grava os buffers modificados do BD da memória para o disco.
  - Grava um registro de checkpoint no log
  - Retorna



Controle de Concorrência

- O SGBD geralmente é um sistema multiusuário, ou seja, manipula diversas transações executando simultaneamente.
- É preciso garantir o **isolamento das transações**
  - Uma solução possível : **Executar um transação por vez**
    - Isso garantiria o escalonamento serial de transações
    - É uma solução bastante ineficiente!
      - Várias transações podem esperar muito tempo para serem executadas
      - A CPU pode ficar muito tempo ociosa enquanto uma transação faz I/O, por exemplo, outras transações poderiam ser executadas

Controle de Concorrência

- É preciso garantir o **isolamento das transações**
  - Solução mais eficiente: **escalonamento não-serial e íntegro**
    - Garantir a execução concorrente de transações de modo a preservar o isolamento
  - A responsabilidade do funcioamento desta estratégia é do subsistema de controle de concorrência ou *scheduler*

execução serial

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

execução não-serial ou concorrente

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

Scheduler (Escalonador)

- Responsável pela definição de escalonamentos não-seriais de transações

  "Um escalonamento E define uma ordem de execução das operações de várias transações, sendo que a ordem das operações de uma transação Tx em E aparece na mesma ordem na qual elas ocorrem isoladamente em Tx"

- Problemas de um escalonamento não-serial mal definido (inválido)
  - Atualização perdida (*lost-update*)
  - Leitura suja (*dirty-read*)

Atualização perdida

- Duas transações T1 e T2 executam concorrentemente e planejam alterar um mesmo dado X.
- Devido a um comprtamento inválido de serialização, T2 altera X sobrepondo a alteração anterior feita por T1.
- A atualização de T1 em X é Perdida antes mesmo de T1 acabar.

T1	T2
read(X)	
X = X – 20	
	read(Z)
	X = Z + 10
write(X)	
read(Y)	
	write(X)
Y = X + 30	
write(Y)	

a atualização de X por T1 foi perdida!

Leitura Suja

- Duas transações T1 e T2 executam concorrentemente e planejam alterar um mesmo dado X.
- T1 realiza a alteração em X conforme planejado e T2 adota este novo valor de X em seu processamento.
- T1 sofre uma falha posterior e desfaz todas as suas ações (inclusive a mudança em X),
- T2 não sabe disso e continua considerando o valor alterado de X

T1	T2
read(X)	
X = X – 20	
write(X)	
	read(X)
	X = X + 10
	write(X)
read(Y)	
abort( )	

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

Scheduler (Escalonador)

- Deve evitar escalonamentos inválidos
  - Isso exige análise de operações em conflito
    - Quais operações pertencem a transações diferentes estão em conflito?
    - Que transações acessam o mesmo dado?
    - Que transações pretendem alterar dados? Quais pretendem só consultá-los?
  - Estabeleceu-se uma tabela de situações de conflito de transações para evitar que se gerem estados inconsistentes no BD

		Ty	
		Read(X)	Write(X)
Tx	Read(X)	Sem Problema	Problema
	Write(X)	Problema	Problema

Teoria da Serializabilidade

- Busca a garantia de escalonamentos não-seriais válidos
- Premissa:

"Um escalonamento não-serial de um conjunto de transações deve produzir resultado equivalente a alguma execução serial destas transações"

entrada:
X = 50
Y = 40

execução serial

saída:
X = 40
Y = 60

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

entrada:
X = 50
Y = 40

execução não-serial *serializável*

saída:
X = 40
Y = 60

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

Teoria da Serializabilidade

- Equivalência de Conflito:

“Dado um escalonamento não-serial E’ para um conjunto de Transações T, E’ é serializável em conflito se E’ for equivalente em conflito a algum escalonamento serial E para T, ou seja, a ordem de quaisquer 2 operações em conflito

é a mesma em E’ e E.”

Teoria da Serializabilidade

- Equivalência de Conflito (Exemplo):

escalonamento serial *E*

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

escalonamento não-serial *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-serial *E2*

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

- *E1* equivale em conflito a *E*
- *E2* não equivale em conflito a nenhum escalonamento serial para T1 e T2
- *E1* é serializável e *E2* não é serializável

Transações em SQL

- Relembrando…
  - Por default, todo comando individual é considerado uma transação
    - **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

Variações do SQL Padrão

- Adotado por alguns SGBDs

```
BEGIN TRANSACTION T1

UPDATE Medicos
SET nroa = NULL
WHERE nroa = @nroAmb

IF @@ERROR <> 0 ROLLBACK TRANSACTION T1

DELETE FROM Ambulatorios
WHERE nroa = @nroAmb

IF @@ERROR <> 0 ROLLBACK TRANSACTION T1
ELSE COMMIT TRANSACTION T1
...
```

Referências Bibliográficas

- ELSMARI, Ramez & NAVATHE, Shamkant B. Sistema de Banco de Dados. Rio de Janeiro: Editora Addison-Wesley. 4ª. edição, 2005, 744p. Capítulo 19