Transações e controle de concorrência

Notas de Aula 1 - referentes a textos
dos livros dos autores:
Elmasri e Silberschatz (veja referencias)

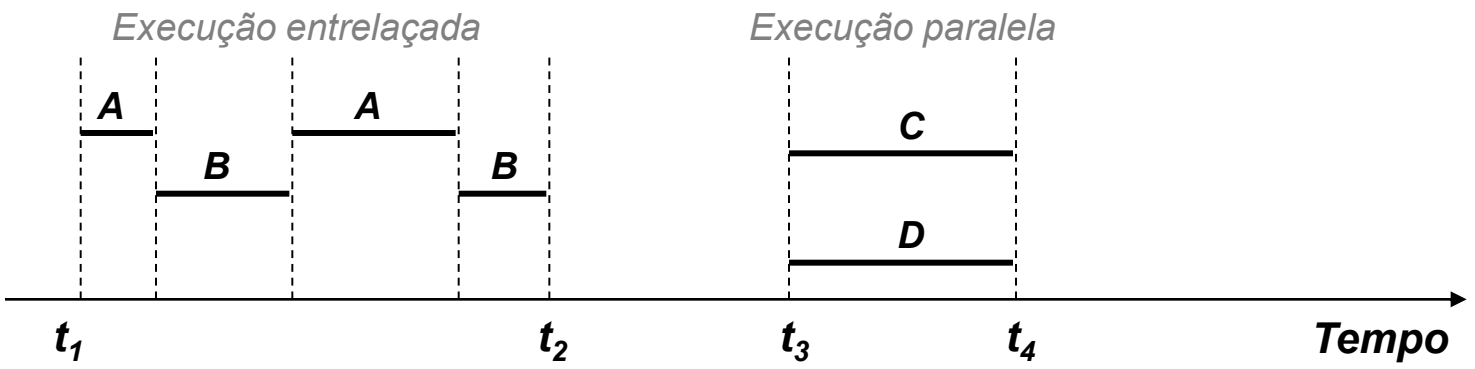
UTFPR

Curso: Eng. De Computação

Disciplina: Banco de Dados 2

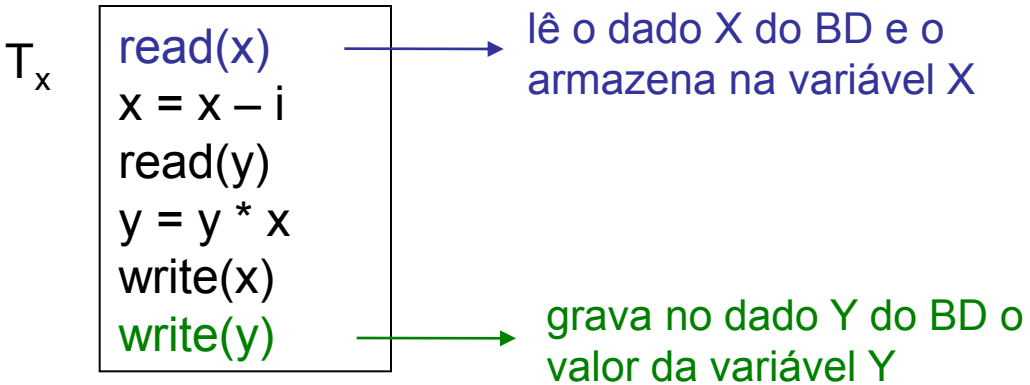
Introdução ao processamento de Transações

- 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



O Conceito de Transação

- Uma transação é uma unidade lógica de processamento de banco de dados
  - Composta de uma ou mais operações
    - Seus limites podem ser determinados em SQL
    - Durante a execução de uma transação o BD pode estar inconsistente
  - De forma simplificada, uma transação pode ser encarada como um conjunto de operações de leitura e escrita de dados



Desafios

- Questões principais devem ser tratadas
  - Falhas de diversos tipos, tais como falhas de hardware e quedas de sistema
  - Execução concorrente de múltiplas transações
- Exemplo

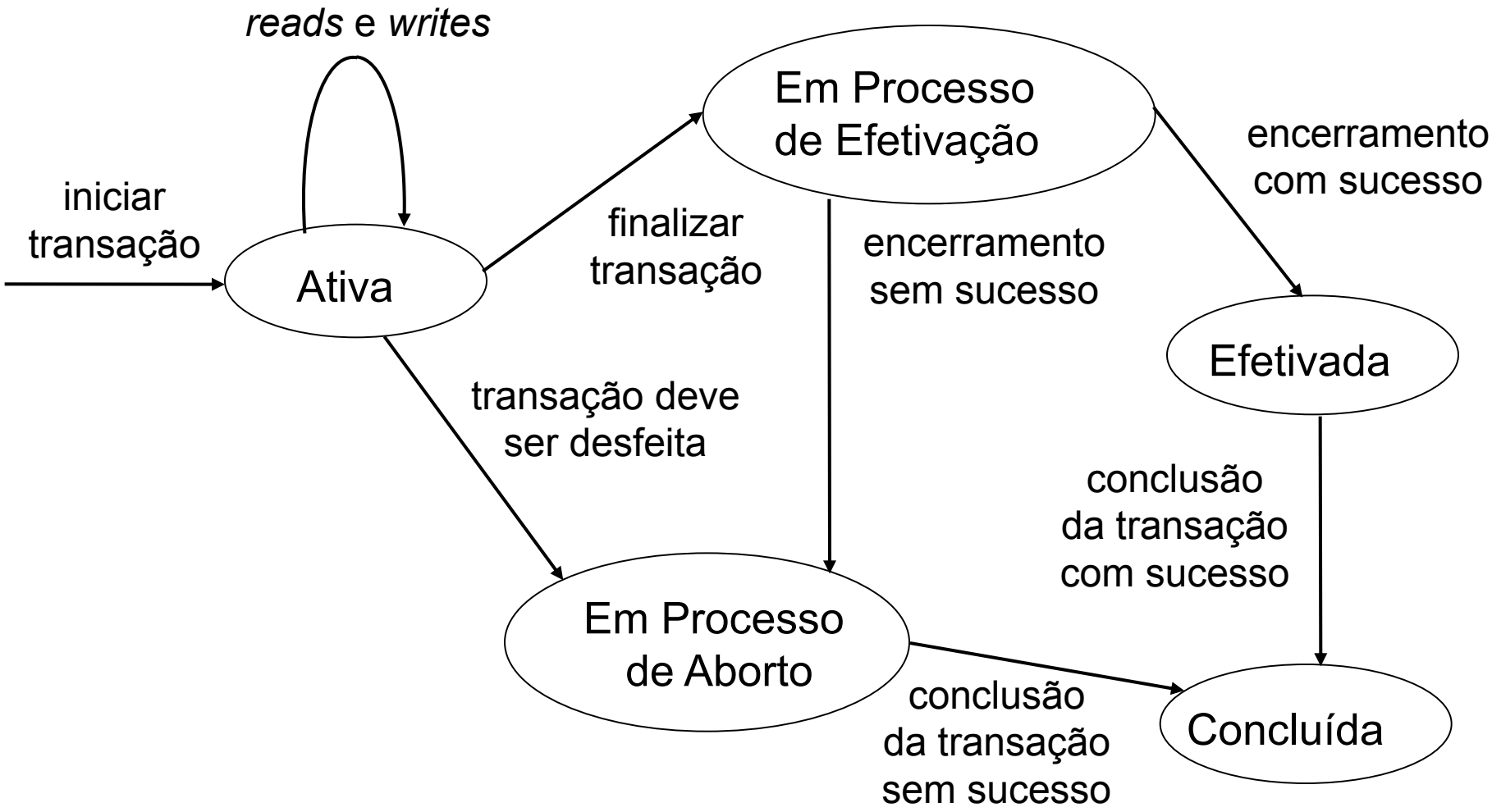
T1- Transação para transferir $ 50 da conta ***A*** para conta ***B***

T2- Transação para transferir 10% do Saldo da conta A para a conta B

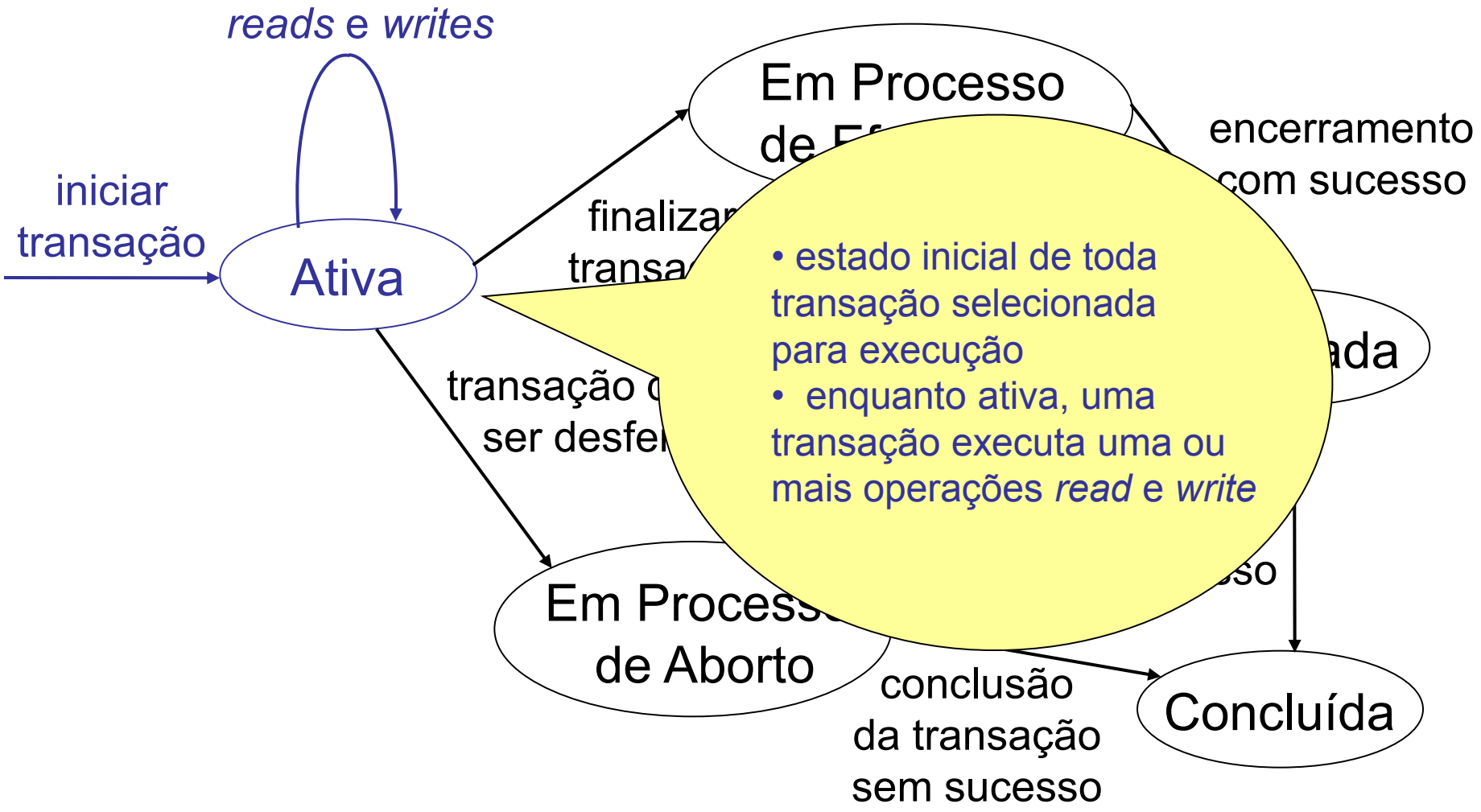
Estados de uma Transação

- 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

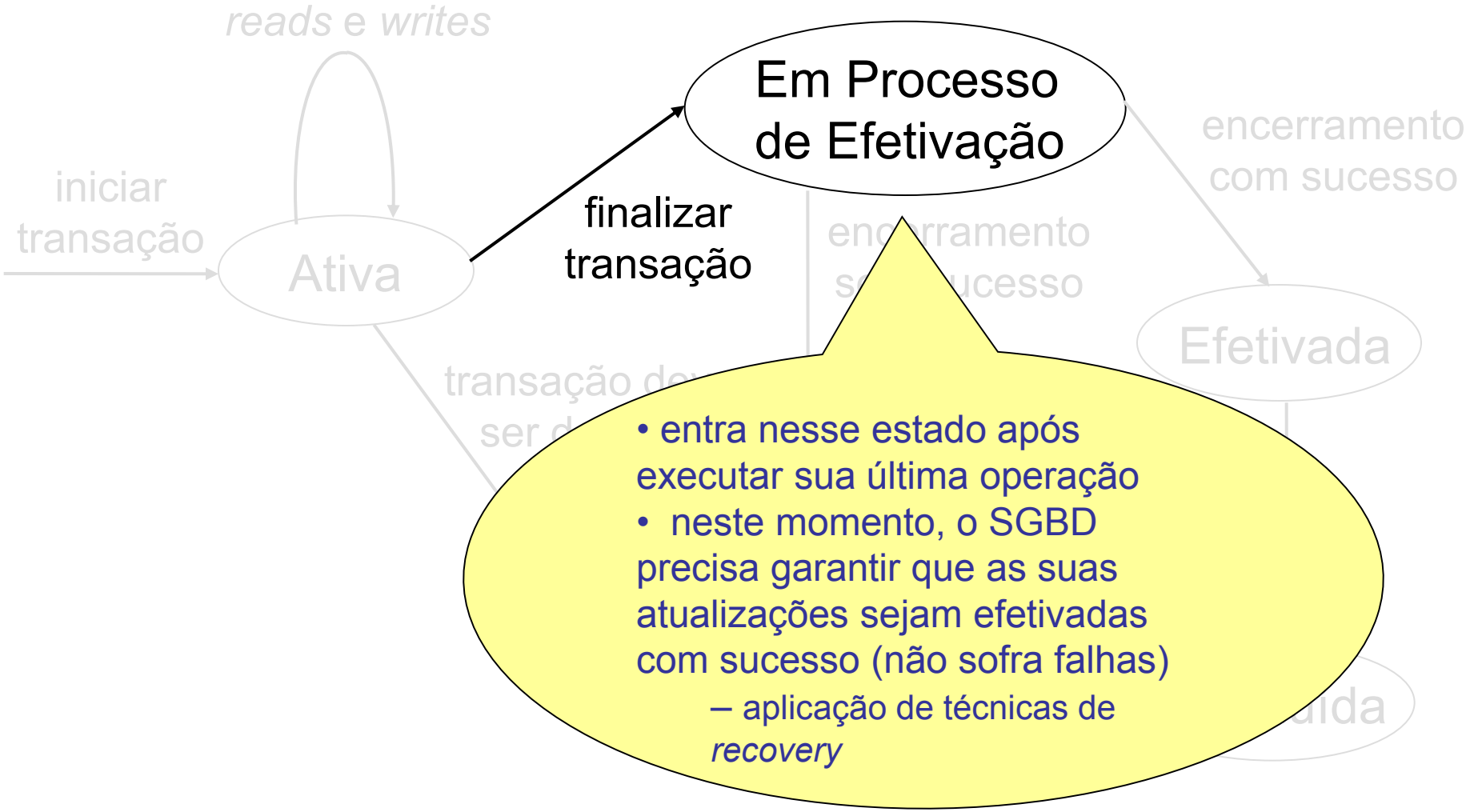
Transição de Estados de uma Transação



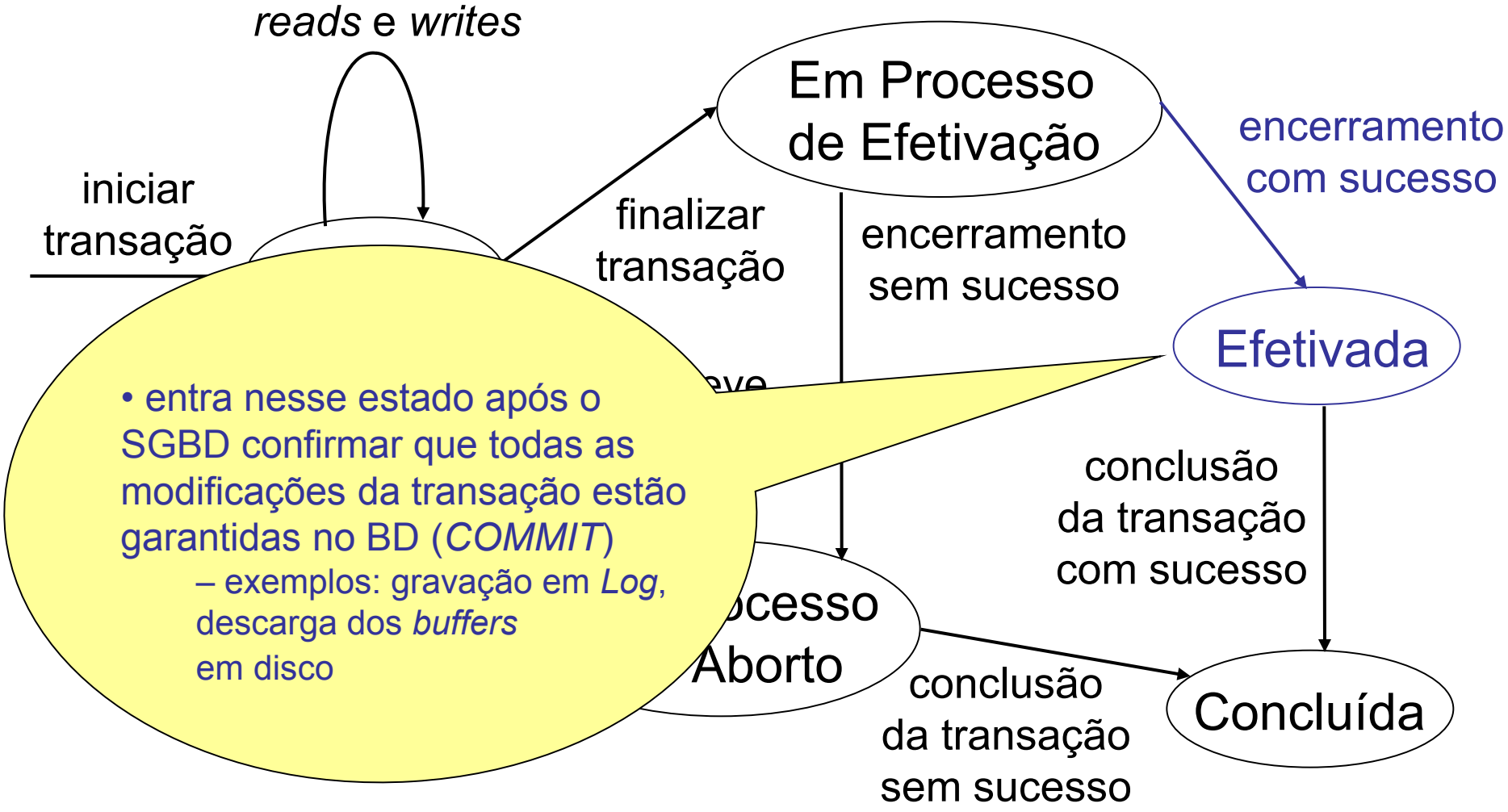
Transição de Estados de uma Transação



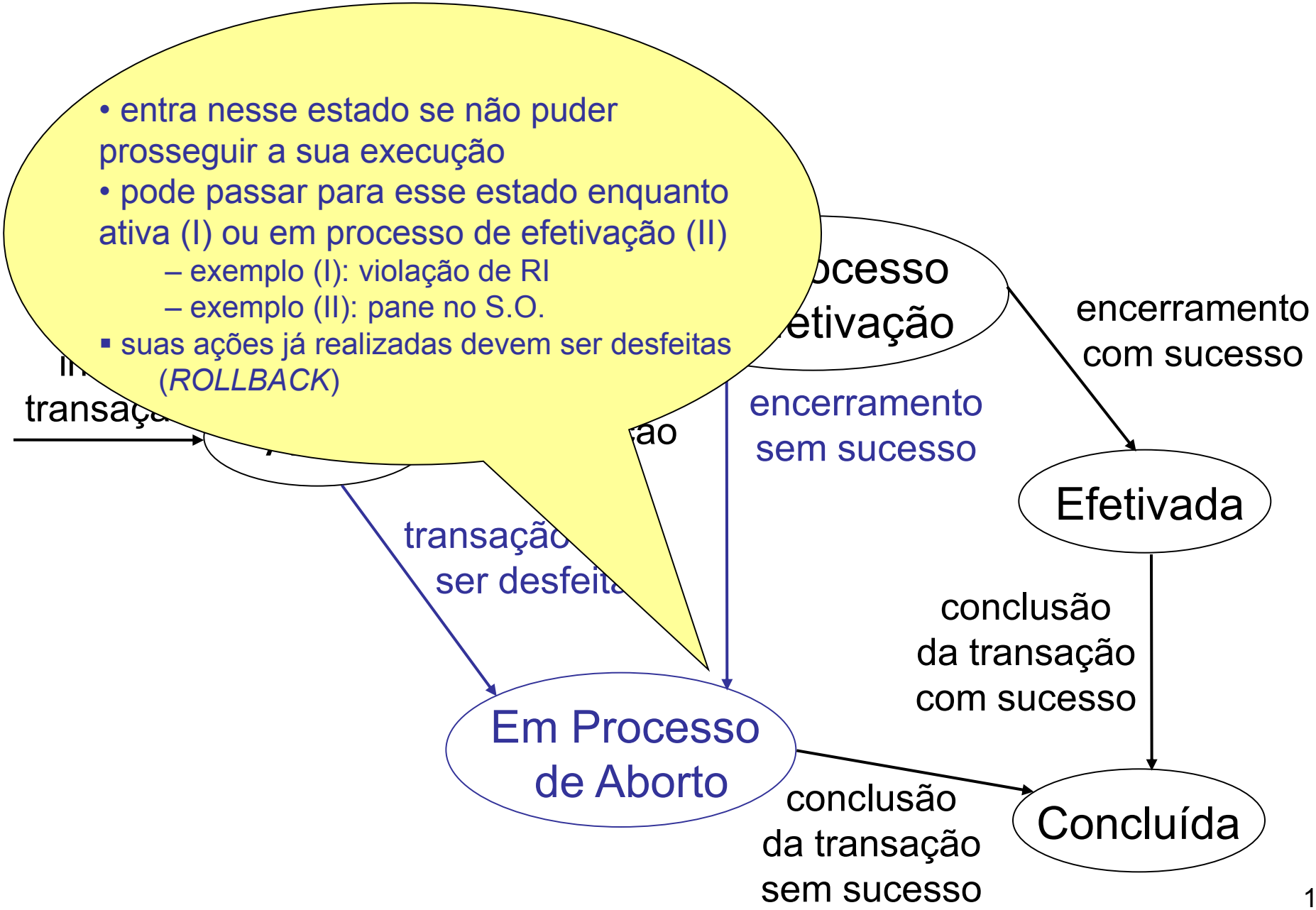
Transição de Estados de uma Transação



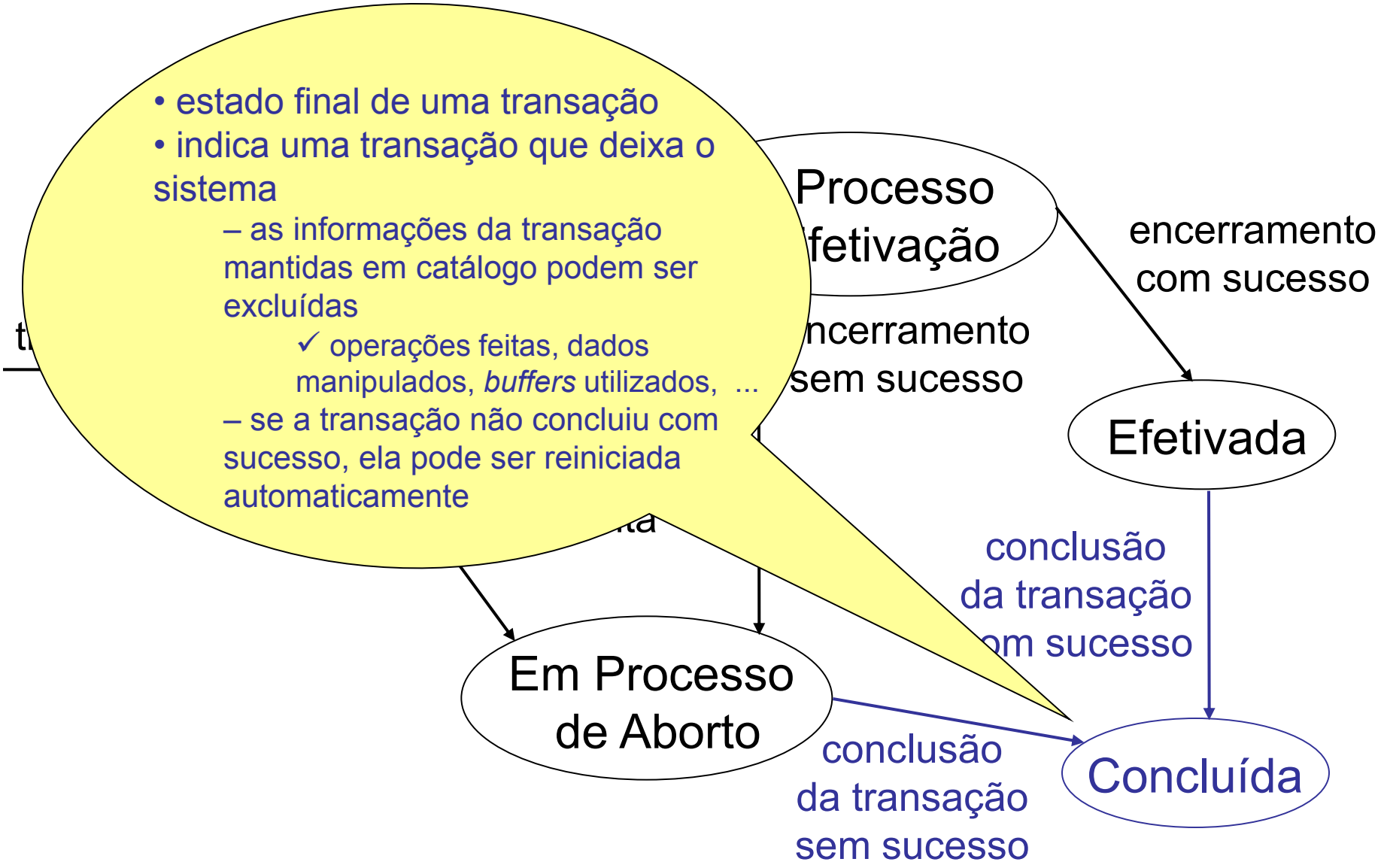
Transição de Estados de uma Transação



Transição de Estados de uma Transação



Transição de Estados de uma Transação



Propriedades de uma Transação (ACID)

- Requisitos que sempre devem ser atendidos por uma transação
- Chamadas de propriedades ACID
  - Atomicidade
  - Consistência
  - Isolamento
  - Durabilidade ou Persistência

Atomicidade

- Princípio do *"Tudo ou Nada"*

Ou todas as operações de uma transação são efetivadas com sucesso no BD ou nenhuma delas se efetiva

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 estado válidos para os dados
    - Sistema 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

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

Exercício 1

- Utilizando as transações T1 e T2, considere que os valores para A e B sejam 1000 e 2000 reais respectivamente. Faça passo a passo a execução de T1 e T2 nas sequencias abaixo :
  - T1 e T2
  - T2 e T1

- Responda: O montante nas contas A e B foram preservados depois da execução de ambas as transações?

Exercício 2

Idem ao exercício 1, mas executando em outra sequencia, desta vez T2 seguida de T1.

- Qual o valor de A e B?
- A soma de A + B é preservada?

Exercicio 3 – obtenha os valores finais para A e B

T1	T2
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)

Exercicio 4 – obtenha os valores finais para A e B

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)	
	B:=B + temp
	Write (B)

Referencias

ELMASRI, R.; NAVATHE, S. B.. Sistemas de Banco de Dados. 4a ed., Pearson-Addison-Wesley, 2005.

SILBERSCHATZ, Abraham; KORTH, Henry F; SUDARSHAM, S. Sistema de Banco de Dados. 3ª ed. São Paulo, MAKRON Books, 1999.

Transações e controle de concorrência

Notas de Aula 2 - referentes a textos
dos livros dos autores:
Elmasri e Silberschatz (veja referencias)

UTFPR
Curso: Eng. De Computação
Disciplina: Banco de Dados 2

Por que o controle de concorrencia é necessário ?

- Diversos problemas podem ocorrer quadno transações são executadas de maneira descontrolada.
  - Atualização perdida
  - Atualização temporária (Dyrty Read)
  - Sumário Incorreto
  - Leitura sem Repetição

Atualizaçao Perdida

- Resevas de cia aérea armazenado registros de Voo da cia
- Cada registro contem o número depoltronas reservada para o voo.
- Figura ao lado representa : T1 transfere N reservas de um voo, cujo numero de assentos reservados está armazenado em um item de dados chamado X, para um outro voo, cujo número de de assentos reservados está armazenado em um item de dados chamado Y.

X	Y
READ(X)	
X=X-N	
	READ(X)
	X=X+M
WRITE(X)	
READ(Y)	
	WRITE(X)
Y=Y+N	
WRITE(Y)	

Atualização Perdida

- Se duas transaçoes acessarem os mesmos itens com operaçoes intercaladas.
- Suponha T1 e T2, sejam submetidas aproximadamente ao mesmo tempo.
- Qual o valor final de X?
- T2 – Lerá o valor de x antes de T1 mudá-lo no

X	Y
READ(X)	
X=X-N	
	READ(X)
	X=X+M
WRITE(X)	
READ(Y)	
	WRITE(X)
Y=Y+N	
WRITE(Y)	

item X tem um valor incorreto porque a atualização feita por T1 foi "perdida" (sobrescrita)

- Testem com os seguintes valores
- X=80
- N=5
- M=4
- O resultado deveria ser 79 porém resulta 84 pois perdeu-se a remoção dos 5 assentos.

Atualização Temporária
Dirty read

- T1 atualiza um item de bd, e a seguir, acontece um erro qualquer ...
- Logo em seguida T2 acessa estes dados antes de T1 fazer rollback!!!
- T2 nao pode ser gravado em BD por que T1 falhou, o BD deve tratar esta inconsistência.

X	Y
READ(X)	
X=X-N	
WRITE(X)	
	READ(X)
	X=X+M
	WRITE(X)
READ(Y)	

Sumário Incorreto

- Se uma transação aplicar uma função agregada para sumário de um número de registros enquanto outras transações estiverem atualizando alguns desses registros.
- T3 Está calculando o número total de reservas em todos os voos
- Enquanto isso... T1 inicia...
- Se acontecer a intercalação de operações mostrado ao lado
- O resultado de T3 não contabilizará N, pois T3 leu o valor de X depois que os N assentos foram subtraídos, mas lerá o valor Y antes que esses N assentos tenham sido adicionados a ele.

T1	T3
X	Y
	sum=0
	READ(A)
	SUM=SUM+A
	.
	.
	.
READ(X)	
X=X+N;	
WRITE(X)	
	READ(A)
	SUM=SUM+X
	READ(Y)
	SUM=SUM+Y
READ(Y)	
Y=Y+N	
WRITE(Y)	

Leitura sem Repetição

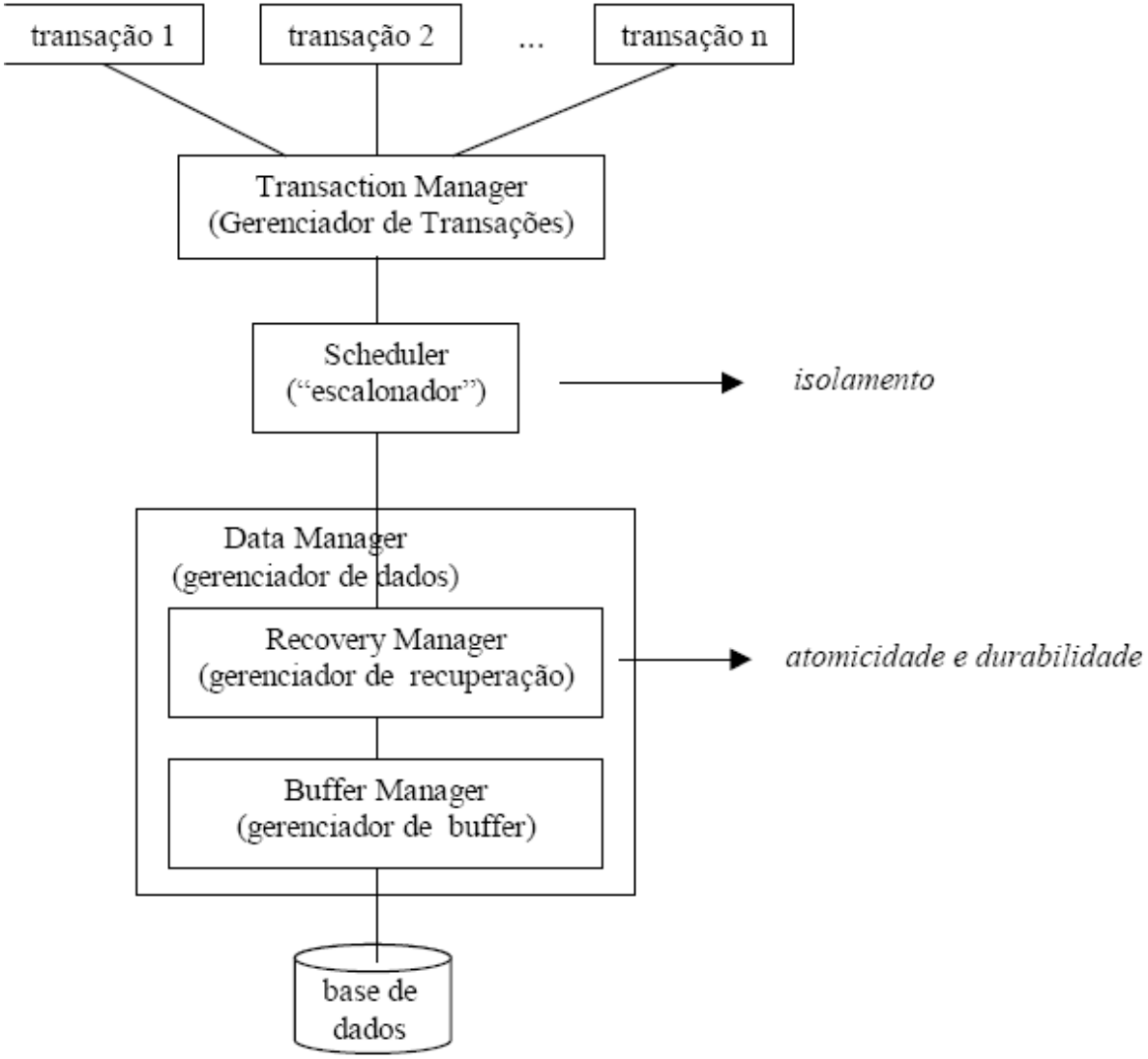
- Quando uma transação T duas vezes e o item for mudado por uma outra transação T' entre essas duas leituras.
- Pode acontecer de T receber valores diferentes para duas leituras do mesmo item.

Como o SGBD resolve isso tudo!?

ACID

- A – ATOMICIDADE
- C – CONSISTENCIA
- I – ISOLAMENTO
- D - DURABILIDADE

SGBD sob ótica de Ger. Transações



Operações read e write

- O acesso ao banco de dados é obtido pelas seguintes operações:
  - Read(X): transfere um dado X do banco para um *buffer* local.
  - Write(X): transfere um dado X do *buffer* local para o banco de dados.
- Ex: T1 transfere 50 reais da conta A para a conta B:

T1: read(A)

A=A-50

Write(A)

Read(B)

B=B+50

Write(B)

Propriedades no exemplo

Transação para transferir R$ 50 da conta A para a conta B:

1. read(***A***)
2. ***A*** **:=** ***A*** **– 50**
3. write(***A***)
4. read(***B***)
5. ***B*** **:=** ***B*** **+ 50**
6. write(***B***)

- Requisito de atomicidade — Se a transação falhar após a etapa 3 e antes da etapa 6, o sistema deve garantir que suas atualizações não sejam refletidas no banco de dados, ou uma inconsistência irá resultar.
- Requisito de consistência — A soma de A e B é inalterada pela execução da transação.
-

Propriedades no exemplo

- Requisito de isolamento — Se entre as etapas 3 e 6, outra transação receber permissão de acessar o banco de dados parcialmente atualizado, ele verá um banco de dados inconsistente (a soma *A* + *B* será menor do que deveria ser). Isso pode ser trivialmente assegurado executando transações serialmente, ou seja, uma após outra. Entretanto, executar múltiplas transações simultaneamente oferece vantagens significativas, como veremos mais adiante.

- 1. read(***A***)
- 2. ***A*** := ***A*** – 50
- 3. write(***A***)
- 4. read(***B***)
- 5. ***B*** := ***B*** + 50
- 6. write(***B***)

- Requisito de durabilidade — Quando o usuário é notificado de que a transação está concluída (ou seja, a transação dos US$ 50 ocorreu), as atualizações no banco de dados pela transação precisam persistir a pesar de falhas.

Execuções concorrentes

- Diversas complicações em relação à consistência de dados
- Porém, boas razões para permitir concorrência:
  - 1 transação -> diversos passos
  - I/O e CPU podem operar em paralelo
  - Uma transação faz I/O, outra é processada pela CPU etc.
  - Processador e disco ficam menos tempo inativos: mais transações por tempo
  - reduz o tempo médio de resposta: T1 e T2 podem ser executadas concorrentemente

Execuções concorrentes (2)

- Concorrência x consistência
  - o sistema de banco de dados deve controlar a interação entre transações concorrentes para impedi-las de destruir sua consistência
  - Identificar quais ordens de execução (escalas de execução) podem garantir a manutenção da consistência

Execuções concorrentes (3)

- Ex: T1 transfere 50 reais de A para B
- Ex: T2 transfere 10% do saldo de A para B:

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)

Supondo: Saldo A = 1000 e saldo B = 2000 Suponha que as transaçõe s sejam executada s em seqüência , T1 e T2:

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);

Ao final da execução: A=855, B=2145 (consistência: A+B é preservado)

Escala 1

Poderia ser T2 e depois T1 (escala 2)

T1	T2
	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);	

Ao final da execução: A=850, B=2150 (consistência: A+B é preservado)

Escala 2

- Em qualquer caso, a consistência é mantida
  - saldo de A+B é o mesmo, antes e depois da execução das transações.
- Essas são escalas de execução seqüenciais
  - seqüência de instruções de várias transações em que as instruções que pertencem a uma única transação aparecem agrupadas
  - Assim, para um conjunto de n transações, há *n!* escalas seqüenciais válidas.

- Quando várias transações são executadas **simultaneamente**, a escala correspondente já pode não ser seqüencial
- Se duas transações são executadas simultaneamente, o SO pode executar uma transação durante algum tempo, depois mudar o contexto e passar a executar a segunda transação durante algum tempo e então voltar à primeira transação durante algum tempo e assim por diante, alternadamente -> várias seqüências de execução são possíveis!
- Geralmente não é possível prever exatamente quantas instruções de uma transação serão executadas antes que a CPU alterne para outra transação.

Uma escala de execução concorrente possível:

T1	T2
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);

Escala 3 OK – equivalente à escala 1

Ao final da execução: (consistência: A+B é preservado)

Nem todas execuções resultam em um estado correto:

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);	
	B=B+temp
	Write(B);

Escala 4 – Inconsistente!!

Ao final da execução: (consistência: A+B **NÃO** é preservado: A=950, B=2100)

- Se o controle da execução concorrente é deixado completamente sob a responsabilidade do SO, muitas escalas de execução possíveis são factíveis, inclusive aquelas que deixam a base de dados em um estado inconsistente.
- É tarefa do sistema de BD garantir que qualquer escala executada deixe o BD num estado consistente: componente de controle de concorrência.
- Qualquer escala executada deve ter o mesmo efeito de outra que tivesse sido executada **sem** concorrência: 1 escala de execução deve ser equivalente a uma escala seqüencial

Necessidade do Controle de Concorrência :

- Se as transações fosses executadas em série, manteriam o BD consistente.
- Com a execução concorrente, as operações das transações se entrelaçam: se ambas atualizaram o mesmo dado pode haver inconsistência.

Atualização Perdida

- *Lost update: dois writes sobre o mesmo valor inicial, o valor de uma das alterações é perdido*

	T1 (+10)	T2 (+20)
1)	read(valor)	read(valor)
2)	valor=valor+10	
3)		valor=valor+20

Atualização Temporária

- *Inconsistent retrieval: o valor recuperado deixou de existir*

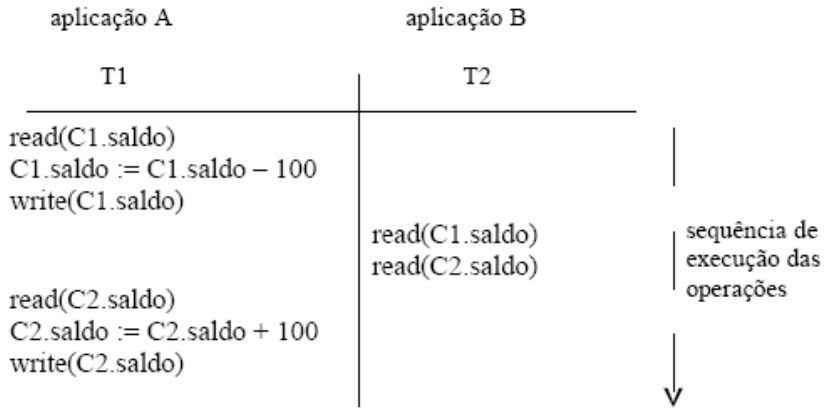
	T1 (+10)	T2 (+20)
1)	read(valor)	
2)	valor=valor+10	
3)	write(valor)	
4)		read(valor)
5)		valor=valor+20
6)		commit
7)	abort	

Sumário Incorreto

- *Phantom problem: durante uma operação executada sobre um conjunto de dados, um novo* valor é inserido no conjunto e não é computado para a operação
- Exemplo : totalização dos saldos

	T1	T2
1)	read(S1, S2, S3)	
2)	total=S1+S2+S3	
3)		insere(S4)
4)	write(total)	

Questões



- *O que define a* ***Consistência da***

transação de transferência bancária da conta C1 para a conta C2?

- *E com relação às outras propriedades?*
- **Num sistema de informações, quem assegura a correção dos dados ?**

Serialização

- Para assegurar a consistência: entender quais escalas de execução podem garantir a consistência e quais não irão fazê-lo.
- Considere só read e write.
  - Ex: escala 3 (ver próximo slide)

T1	T2
read(A);	
Write(A)	
	read(A);
	Write(A)
Read(B);	
Write(B);	
	Read(B);
	Write(B);

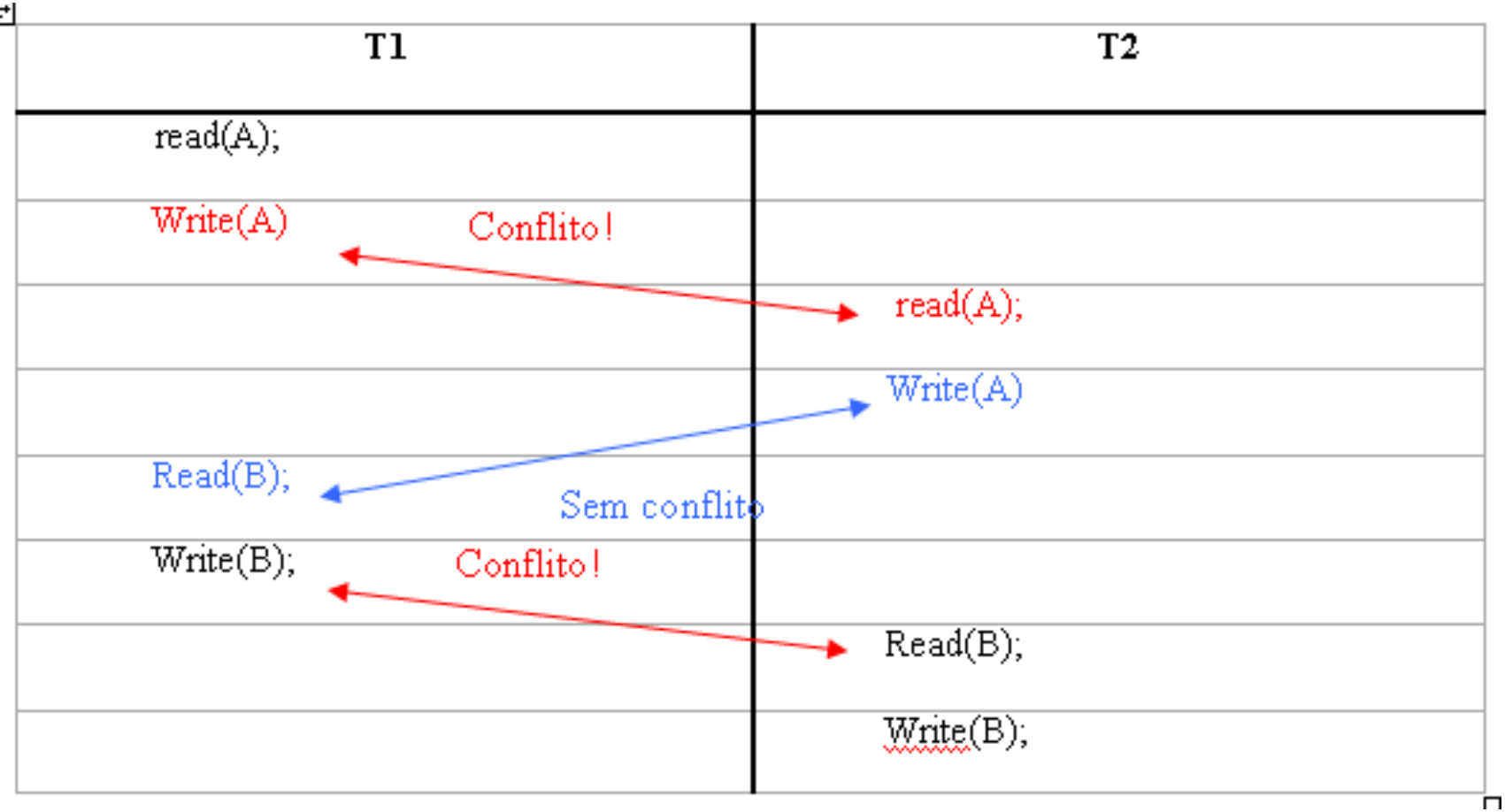
Formas de equivalência entre escalas de execução:
conduzem às noções de *serialização de conflito*

Serialização de conflito

- Considerar instruções sucessivas de transações diferentes.
- Idéia geral: se as instruções referem-se a itens de dados diferentes, então podemos alternar as instruções sem afetar os resultados; se as instruções referem-se ao mesmo item de dados, então a ordem pode importar!
- Quatro casos a considerar:
  - read(q), read(q): seqüência de execução **não importa**, pois o mesmo valor é lido
  - read(q), write(q): seqüência de execução importa: lê antes, escreve depois ou escreve antes e lê depois
  - write(q), read(q): seqüência de execução importa: lê antes, escreve depois ou escreve antes e lê depois
  - write(q), write(q): importa, pois o valor final gerado depende de quem executar por último (e isso afeta quem for ler o valor depois)

Serialização de conflito

- Assim, temos que cuidar dos três últimos casos.
- Duas instruções entram em conflito caso elas sejam operações pertencentes a diferentes transações, agindo no mesmo item de dado, e pelo menos uma dessas instruções é uma operação *write*.
- Exemplo mostrado na escala 3: (ver próximo slide)



Se duas instruções sucessivas de transações diferentes não entram em conflito, podemos trocar a ordem destas instruções para produzir uma nova escala de execução (equivalente)
Logo, no exemplo anterior, vamos trocar o que não é conflitante (em azul):

Obtendo:

T1	T2
read(A);	
Write(A)	
	read(A);
Read(B);	
	Write(A)
Write(B);	
	Read(B);
	Write(B);

Escala 5 (equivalente à 3)

Analisando novamente as instruções não conflitantes, temos (em azul):

T1	T2
read(A);	
Write(A)	
	read(A);
Read(B);	
	Write(A)
Write(B);	
	Read(B);
	Write(B);

Trocando novamente o que não é conflitante (em azul), obtemos:

T1	T2
read(A);	
Write(A)	
Read(B);	
	read(A);
	Write(A)
Write(B);	
	Read(B);
	Write(B);

Analisando novamente as instruções não conflitantes, temos (em azul):

T1	T2
read(A);	
Write(A)	
Read(B);	
	read(A);
	Write(A)
Write(B);	
	Read(B);
	Write(B);

Trocando novamente o que não é conflitante (em azul), temos:

T1	T2
read(A);	
Write(A)	
Read(B);	
	read(A);
Write(B);	
	Write(A)
	Read(B);
	Write(B);

Analisando novamente as instruções não conflitantes, temos (em azul):

T1	T2
read(A);	
Write(A)	
Read(B);	
	read(A);
Write(B);	
	Write(A)
	Read(B);
	Write(B);

Trocando novamente o que não é conflitante (em azul):

T1	T2
read(A);	
Write(A)	
Read(B);	
Write(B);	
	read(A);
	Write(A)
	Read(B);
	Write(B);

Logo, a escala 3 (concorrente) é equivalente a uma escala seqüencial! Produzem o mesmo estado final. Logo, a escala 3 é

- Se uma escala S puder ser transformada em outra S’, por uma série de instruções não conflitantes, dizemos que S e S’ são *equivalentes no conflito*
  - Leva ao conceito de serialização de conflito
  - Uma escala S é conflito serializável se ela é equivalente no conflito a uma escala de execução sequencial
    - Assim: escala 3 é conflito serializável, já que é equivalente no conflito à escala seqüencial 1

Referencias

ELMASRI, R.; NAVATHE, S. B.. Sistemas de Banco de Dados. 4a ed., Pearson-Addison-Wesley, 2005.

SILBERSCHATZ, Abraham; KORTH, Henry F; SUDARSHAM, S. Sistema de Banco de Dados. 3ª ed. São Paulo, MAKRON Books, 1999.

Controle de DeadLock

Notas de Aula 1 - referentes a textos
do livro do autor:
Silberschatz (veja referencias)

UTFPR

Curso: Engenharia de Computação

Disciplina: Banco de Dados 2

Controle de Deadlock

Ocorre quando temos um conjunto de transações no qual todas estão em espera, uma aguardando o término da outra para prosseguir.

Técnicas para Controle de Deadlock:

Prevenção de deadlock

- Evita os deadlocks antes que estes ocorram

- Preferível se a probabilidade de ocorrerem deadlocks for muito alta

Detecção e recuperação de deadlock

- Não evita os deadlocks, mas os detecta e impede o bloqueio indefinido das transações envolvidas

- Mais eficiente se ocorrerem poucos deadlocks

• *Rollback pode ser necessário independentemente da técnica utilizada*

Controle de Deadlock – Prevenção

Estratégias de prevenção usadas por SGBDs

- **Esperar-morrer:** Se TS(Ti) < TS(Tj), Ti só espera um dado mantido por Tj se esta for mais nova; caso contrário Ti é abortada (morre)

Exemplo:

- Se T1, T2, T3 tiverem timestamps 5, 10, 15 respectivamente e T1 solicita um item mantido por T2, então T1 esperará.
- Se T3 solicitar o item mantido por T2, T3 será revertido, morrerá.

Controle de Deadlock – Prevenção

Estratégias de prevenção usadas por SGBDs

- **Esperar-morrer:** Se TS(Ti) < TS(Tj), Ti só espera um dado mantido por Tj se esta for mais nova; caso contrário Ti é abortada (morre)

Exemplo:

- Se T1, T2, T3 tiverem timestamps 5, 10, 15 respectivamente e T1 solicita um item mantido por T2, então T1 esperará.
- Se T3 solicitar o item mantido por T2, T3 será revertido, morrerá.

Controle de Deadlock – Prevenção

Estratégias de prevenção usadas por SGBDs

- Ferir-esperar: Se TS(Ti) > TS(Tj), Ti somente espera um dado mantido por Tj se esta for mais antiga; caso contrário ela obriga Tj a abortar e liberar o dado (Ti fere Tj)

Exemplo:

Se T1, T2, T3 tiverem timestamps 5, 10, 15 respectivamente e T1 solicitar um item de dados mantido por T2, então o item de dados será liberado de T2 e T2 será desfeita. Se T3 solicitar um item de dados mantido por T2, então T3 esperará.

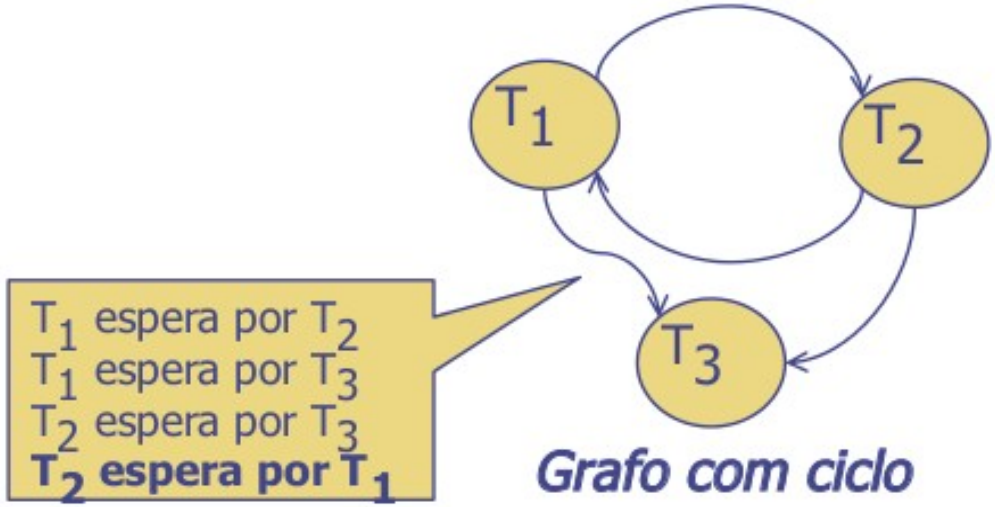
• Timeout: pedido de bloqueio possui um tempo máximo de espera; se o dado não for liberado neste tempo, a transação é abortada e reiniciada

Controle de Deadlock – Detecção

Para determinar se as transações estão em deadlock:

- O sistema precisa manter informações sobre a alocação dos dados e as solicitações pendentes

- Deadlocks podem ser detectados usando grafos de espera, nos quais um ciclo indica um deadlock



Controle de Deadlock - Recuperação

Reverter uma ou mais transações para quebrar o deadlock. Devem ser tomadas três ações:

Selecionar uma vítima (ou vítimas) de acordo como mínimo custo:

• A quanto tempo a transação está em processamento e quanto tempo será ainda necessário para que a tarefa seja completada;

• Quantos itens de dados a transação usou;

• Quantos itens ainda a transação usará até que se complete;

• Quantas transações serão envolvidas no rollback.

Rollback: Determinar até que ponto a transação deve ser revertida:

- Reverter totalmente;
- O suficiente para quebrar o deadlock (exige informações adicionais).

Inanição: Deve-se garantir que uma transação seja escolhida vítima somente um número finito e pequeno de vezes

Referencias

SILBERSCHATZ, Abraham; KORTH, Henry F; SUDARSHAM, S. Sistema de Banco de Dados. 3ª ed. São Paulo, MAKRON Books, 1999.