centro universitário FEI

AULA 8 – TRANSAÇÕES

PROFA. DRA. LEILA BERGAMASCO

CC6240 – Tópicos Avançados em Banco de Dados

TRANSAÇÕES - CONCEITOS

- Conceitos
  - Transação: unidade de execução do programa que acessa/atualiza vários itens de dados.
    - Uma sequência de leituras e gravações
    - Em geral, iniciada por um programa do usuário, escrita em linguagem de alto nível.
    - Delimitada pelas instruções: `begin transaction` e `end transaction`
    - `PostgreSQL`
  - Para garantir integridade dos dados, SGBD deve manter as propriedades ACID: Atomicidade, Consistência, Isolamento e Durabilidade.

TRANSAÇÕES - CONCEITOS

- Propriedades ACID:
  - Atomicidade: todas as operações executadas ou nenhuma delas.
  - Consistência: execução de transação isolada (sem outra transação simultânea) deve manter consistência dos dados.
  - Isolamento: uma transação não "percebe" outra transação – dada uma determinada transação → ou transação terminou antes dela ou começou depois.
  - Durabilidade: após uma transação finalizada, mudanças persistem no BD, mesmo se houver falhas no sistema.

TRANSAÇÕES - CONCEITOS

- Propriedades ACID:
  - Exemplo: transações acessando dados com as operações:
    - read (x): transfere dado do BD para o buffer
    - write (x): transfere dado do buffer para o BD
- Exemplo clássico de transferência de dinheiro entre 2 contas bancárias A e B:

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

TRANSAÇÕES - CONCEITOS

- Exemplo clássico de transferência de dinheiro entre 2 contas bancárias: A e B

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

Consistência:

-soma de A e B permanecem inalteradas no final

-garantir a consistência de uma transação individual é dever do programador da aplicação

TRANSAÇÕES - CONCEITOS

- Exemplo clássico de transferência de dinheiro entre 2 contas bancárias: A→B

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

Atomicidade:
-ou faz todas as operações ou desfaz tudo se houver falha no SGBD
- tarefa do SGBD (componente de gerenciamento de transação e recuperação)

TRANSAÇÕES - CONCEITOS

- Exemplo clássico de transferência de dinheiro entre 2 contas bancárias: A→B

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

Durabilidade:
-quando a transação terminar, os valores da nova conta persistirão no BD, mesmo se houver falha do sistema
- tarefa do SGBD (componente de gerenciamento de recuperação)

TRANSAÇÕES - CONCEITOS

- Exemplo clássico de transferência de dinheiro entre 2 contas bancárias: A→B

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

Isolamento:

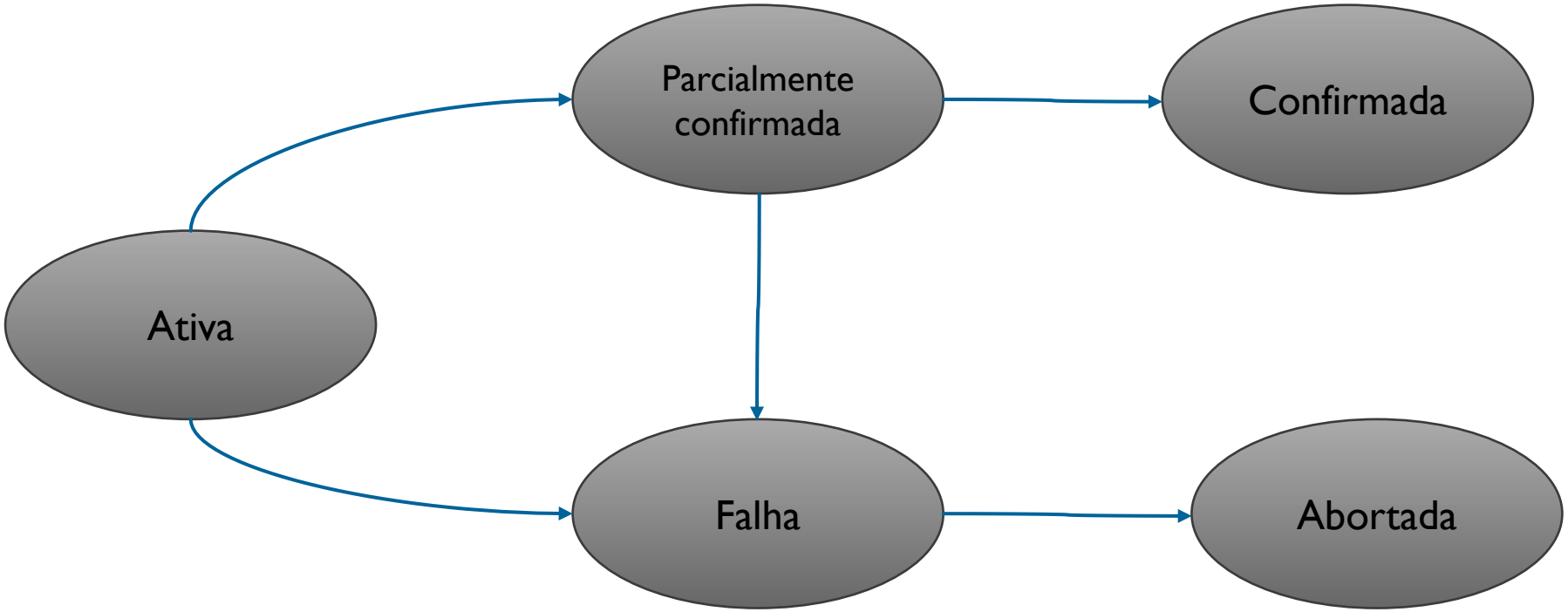
-se houver outra transação sendo executada ao mesmo tempo, esta não influenciará na transação atual

- tarefa do SGBD (componente de controle de concorrência)

ESTADOS DE TRANSAÇÃO

- Se SGBD não apresentar falhas:
  - transação completada com sucesso → transação confirmada (commited)
  - Em caso de falhas: transação abortada → transação revertida (rolled back).
- Após confirmada, transação somente poderá ser revertida se houve uma transação de compensação e tal tarefa é responsabilidade do usuário.
  - Usuário = usuário final ou programador
    - Exemplo de compensação: dinheiro não sai no caixa eletrônico e aplicação deve atualizar a CC do cliente.
- Estados possíveis de uma transação:
  - ativa: enquanto está executando;
  - parcialmente confirmada: após execução instrução final;
  - falha: quando descobre-se que execução normal não pode prosseguir;
  - abortada: depois que transação foi revertida e o BD foi restaurado ao estado anterior ao início da transação;
  - confirmada: após término bem sucedido.

DIAGRAMA DE ESTADOS



ESTADOS DE TRANSAÇÃO

- SGBD pode:
  - reiniciar a transação: somente se tiver sido abortada como resultado de algum erro de HW ou SW que não foi criado por meio da lógica interna da transação.
    - transação reiniciada é considerada nova transação.
  - matar a transação: devido a algum erro lógico interno que só pode ser corrigido com reescrita do programa de aplicação.
    - exemplos de motivos: entrada defeituosa de dados, dados não encontrados no BD.

- Maioria dos SGBDs não permite escritas externas observáveis em transações (impressões, caixa eletrônico)
  - Em geral são armazenadas em memória volátil e depois executadas

Garantir Atomicidade!

centro universitário FEI

EXECUÇÕES SIMULTÂNEAS

- Para executar transações a abordagem mais simples é:
  - Executo a transação
  - SGBD valida que está ok
  - Transação concluída
- Executar transações serialmente é mais fácil, mas há motivos para permitir concorrência. Quais?
  - Melhor *throughput* (número de transações executadas em determinada quantidade de tempo);
  - Melhor utilização de recursos (processador, E/S);
  - Tempo de espera reduzido (transações curtas e longas).
- Qual é o problema de execução simultânea de transações?
  - Consistência!

EXECUÇÕES SIMULTÂNEAS

- Schedule
  - define uma ordem de execução das operações de várias transações

Como garantir consistência ao final da execução das duas transações?

T1:

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

T2:

```
read(A);
temp := A  * 0,1;
A := A - temp;
write (A)
read (B);
B := B + temp;
write (B);
```

EXECUÇÕES SIMULTÂNEAS

- Schedule: ordem cronológica de executar transações

Retira 50 de A e adiciona 50 em B -> soma final deve ser igual a inicial.

T1:

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

T2:

```
read(A);
temp := A  * 0,1;
A := A - temp;
write (A)
read (B);
B := B + temp;
write (B);
```

Retira 10% de A e adiciona valor em B -> Soma final deve ser igual a inicial.

Logo, a SOMA dos saldos após as transações T1 e T2 não muda. Supondo saldo atual = 1000 e 2000 para A e B, respectivamente. Soma final deve ser = 3000.

- Alternativa 1: Uma transação após a outra → schedules seriais

EXECUÇÕES SIMULTÂNEAS

T1	T2

T1:

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

Ao final: A=950 e B=2050

T2:

```
read(A);
temp := A  * 0,1;
A := A - temp;
write (A)
read (B);
B := B + temp;
write (B);
```

I

Retira 10% de A (950) → temp = 95 e A= 855
Adiciona valor em B -> B= 2050+95 = 2145

Soma final = 2145+855 = 3000

- Alternativa 1: Uma transação após a outra

EXECUÇÕES SIMULTÂNEAS

T1 | T2

T2:

```
read(A);
temp := A  * 0,1;
A := A - temp;
write (A)
read (B);
B := B + temp;
write (B);
```

Retira 10% de A (1000) → temp = 100 e A= 900
Adiciona valor em B -> B= 2000+100=2100

Ao final: A=900-50 = 850
e B=2100+50 = 2150

T1:

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

Soma final = 2150+850 = 3000

centro universitário FEI

EXECUÇÕES SIMULTÂNEAS

- Schedule serial:
  - ordem cronológica de executar transações;
  - precisa consistir todas as instruções da transação;
  - precisa preservar a ordem das instruções em cada transação;
    - Ou seja, uma começa somente quando outra termina

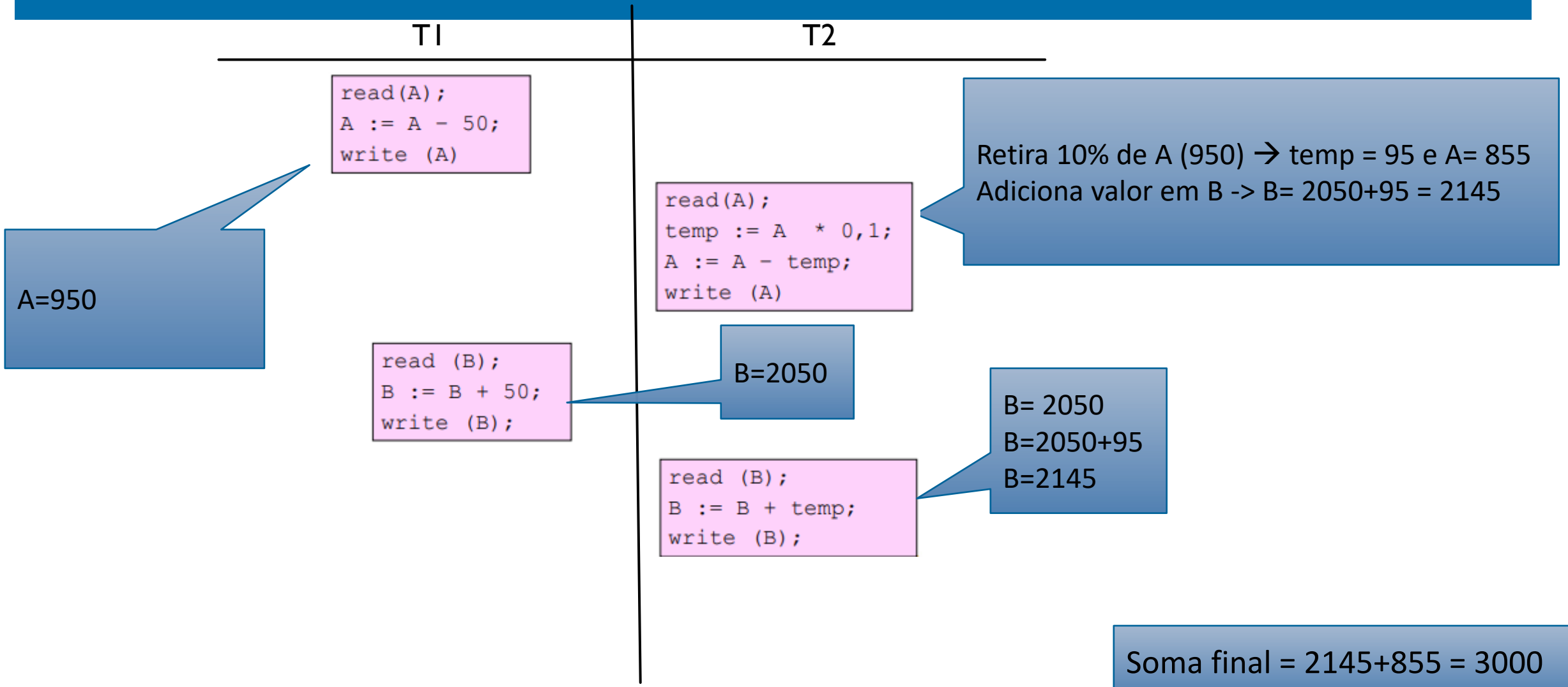
- Quantos schedules seriais válidos existem para um conjunto de n transações ?
  - N! (5 transações = 120 possibilidades)
  - Schedule serial executa transações simultaneamente?
  - NÃO!

EXECUÇÕES SIMULTÂNEAS

- Se SGBD executa transações simultaneamente:
  - schedule não precisa ser serial;
  - SO pode comutar entre transações: fatia de tempo para cada uma delas.
  - Vantagem
    - Compartilhamento de recursos.
  - Várias sequências de execução possíveis → intercalação de instruções:
    - difícil prever quantas instruções de cada transação serão executadas em sua fatia de tempo;
    - quantos schedules são possíveis para n transações?
- Muito maior que n!

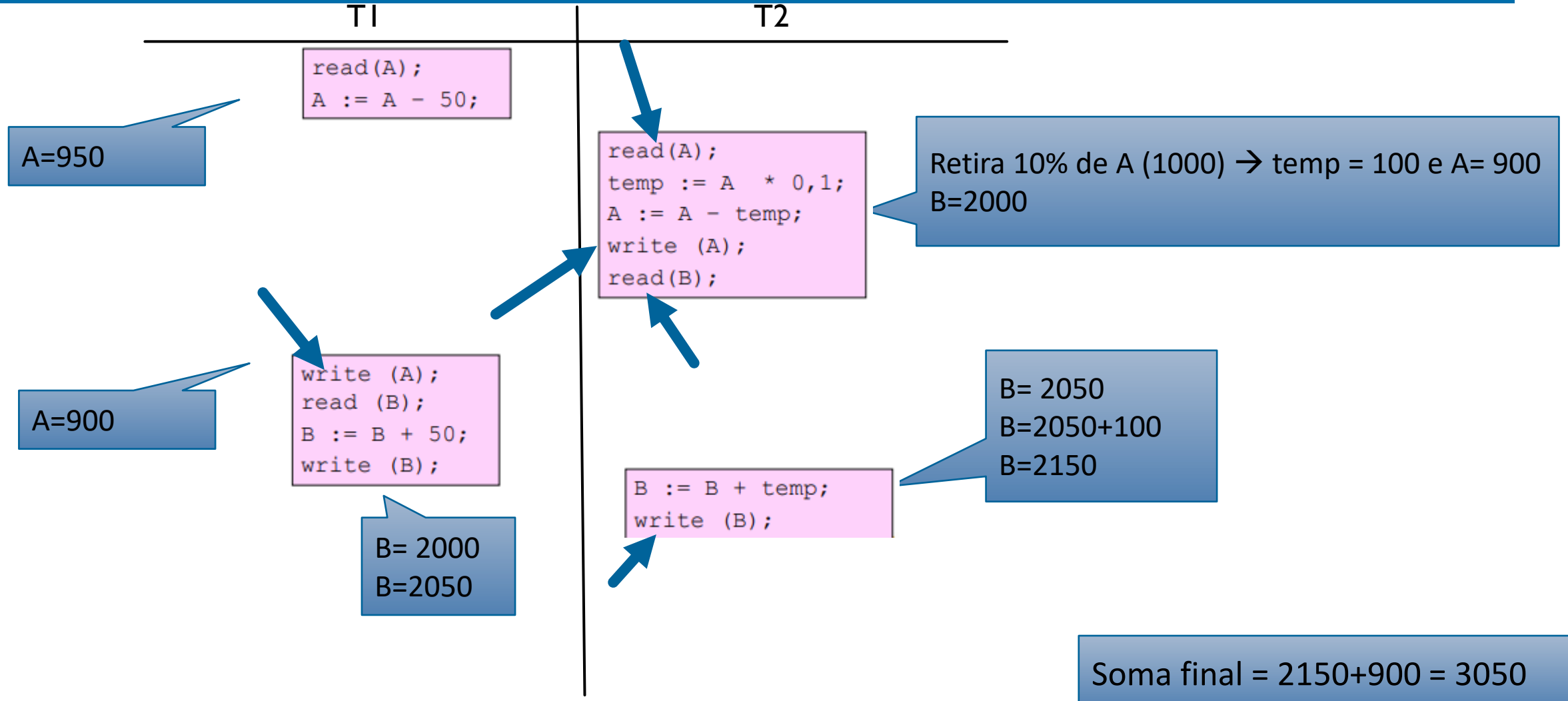
Alternativa 2: Intercalando transações

EXECUÇÕES SIMULTÂNEAS



- Execuções concorrentes geram sempre estados válidos?

EXECUÇÕES SIMULTÂNEAS

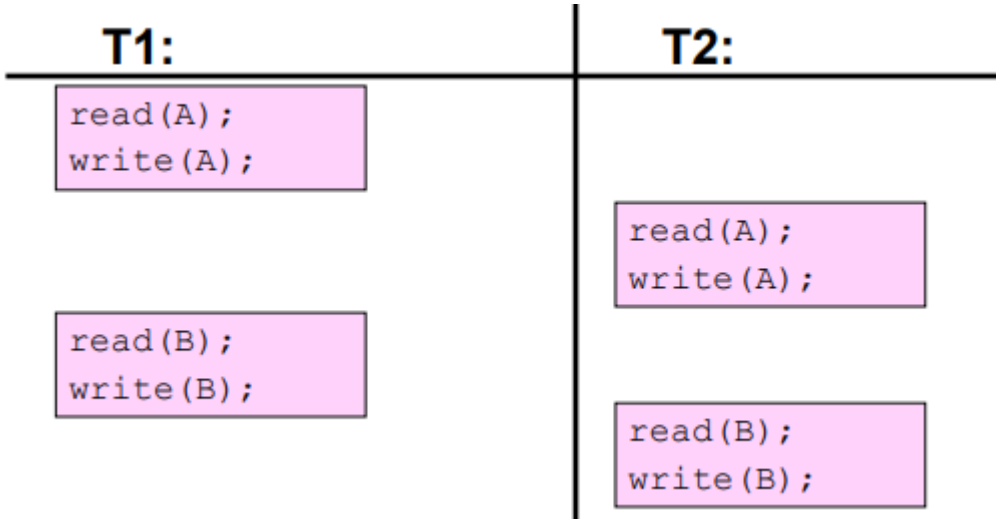


EXECUÇÕES SIMULTÂNEAS

- Se o controle da execução ficar a cargo do SO, muitos schedules errados serão possíveis.
- Por isso, é tarefa do SGBD garantir que qualquer schedule executado deixe o BD em estado consistente.
  - o schedule precisa ser equivalente a um schedule serial.
  - Responsável por isso: componente de controle de concorrência.
- Logo, a questão se resume a: obter um schedule que contenha execuções simultâneas equivalente a um schedule serial.
- Há estratégias para verificar a equivalência
  - Serialização por conflito
  - Serialização por visão

EXECUÇÕES SIMULTÂNEAS

- Para efeitos de escalonamento, interessam somente as instruções: read (Q) e write (Q).



Duas transações com dois conjuntos de instruções

SERIAÇÃO (SERIALIZAÇÃO) DE CONFLITO

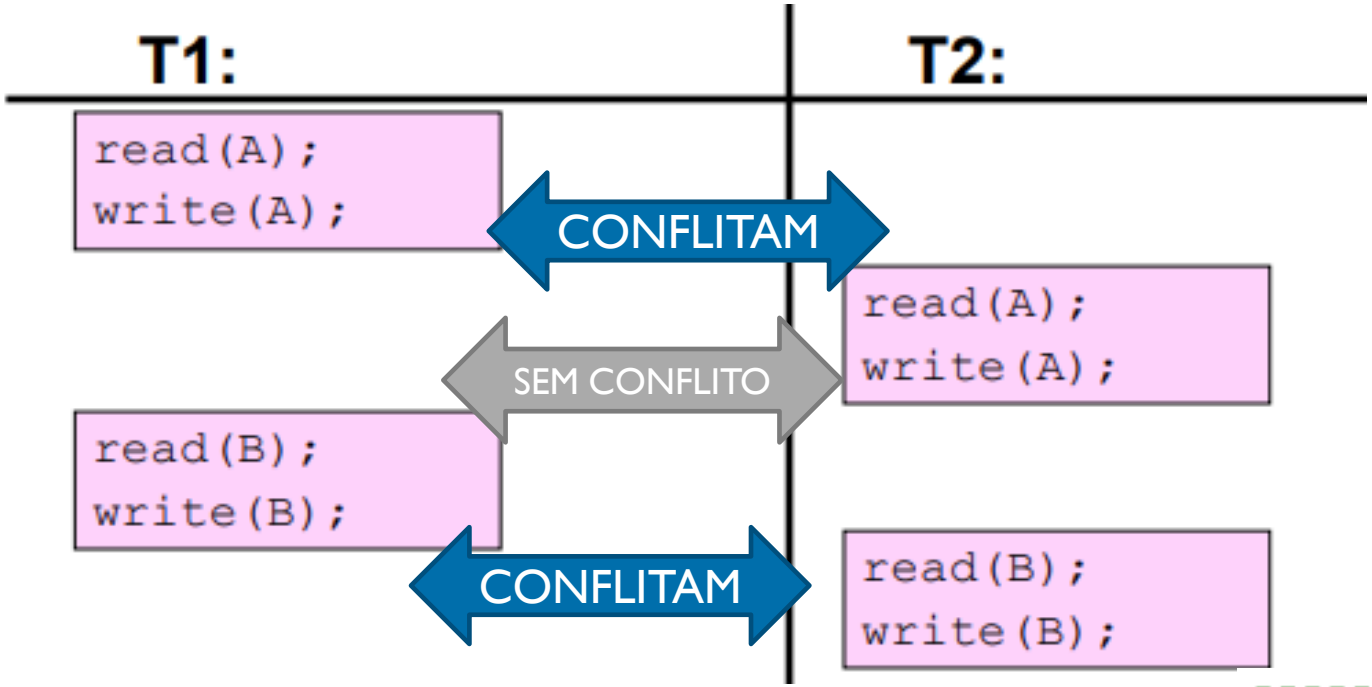
- Schedule S com duas instruções consecutivas I_i e I_j, pertencentes às transações T_i e T_j , respectivamente.
- Se I_i e I_j se referem a diferentes itens de dados, podemos inverter instruções I_i e I_j consecutivas sem afetar resultados de qualquer instrução do schedule.
- Se I_i e I_j se referem ao mesmo item de dado a ordem das duas etapas pode importar

SERIAÇÃO DE CONFLITO

- 4 casos a considerar I_i e I_j
  - I_i = read(Q) e I_j = read(Q) → ordem não importa
  - I_i = read(Q) e I_j = write(Q) → ordem importa:
    - Se I_i vem antes de I_j, T_i não lê o valor de Q escrito por T_j
    - Se I_j vem antes de I_i, T_i lê o valor de Q escrito por T_j
  - I_i = write(Q) e I_j = read(Q) → ordem importa: pelos mesmos motivos
  - I_i = write(Q) e I_j = write(Q) → ordem não afeta T_i ou T_j

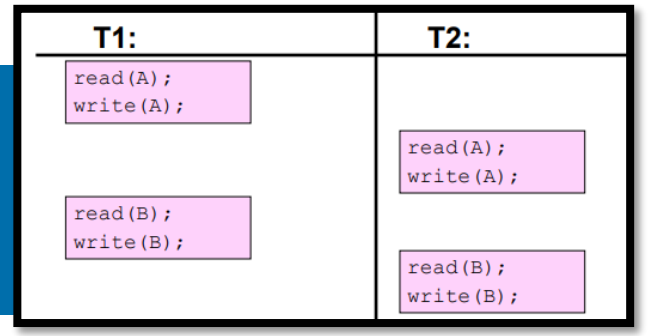
EXECUÇÕES SIMULTÂNEAS

- I_i e I_j conflitam se houver pelo menos uma operação write

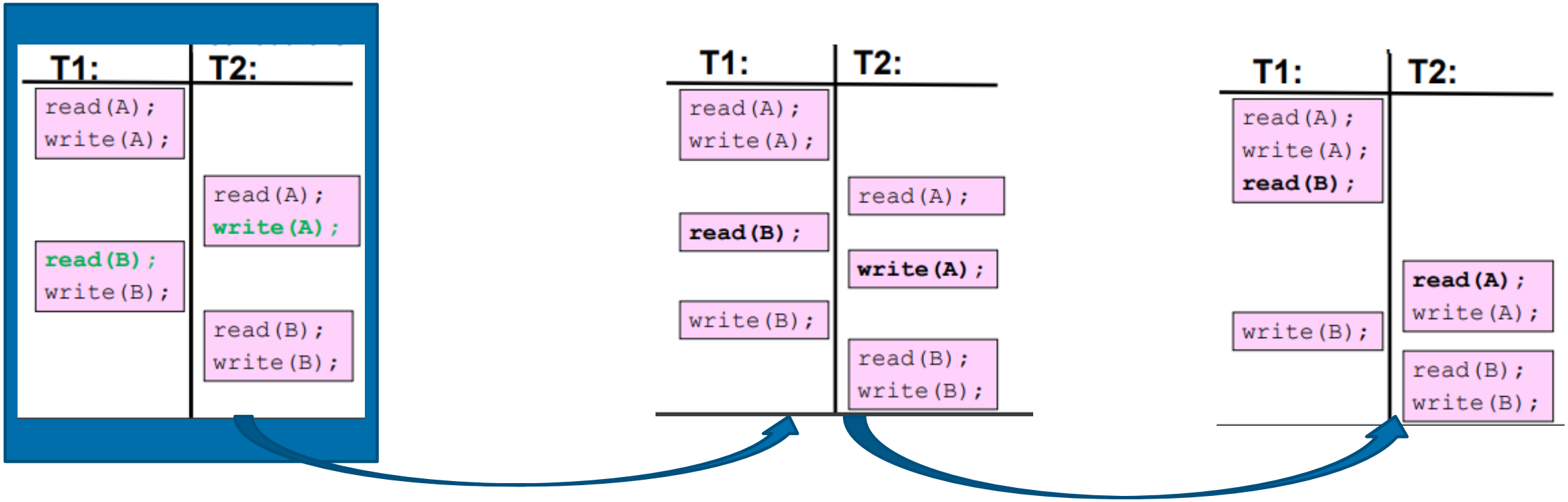


1. I_i = read(Q) e I_j = read(Q) → ordem não importa
2. I_i = read(Q) e I_j = write(Q) → ordem importa
3. I_i = write(Q) e I_j = read(Q) → ordem importa
4. I_i = write(Q) e I_j = write(Q) → ordem não afeta T_i ou T_j, mas pode afetar próximas instruções

EXECUÇÕES SIMULTÂNEAS

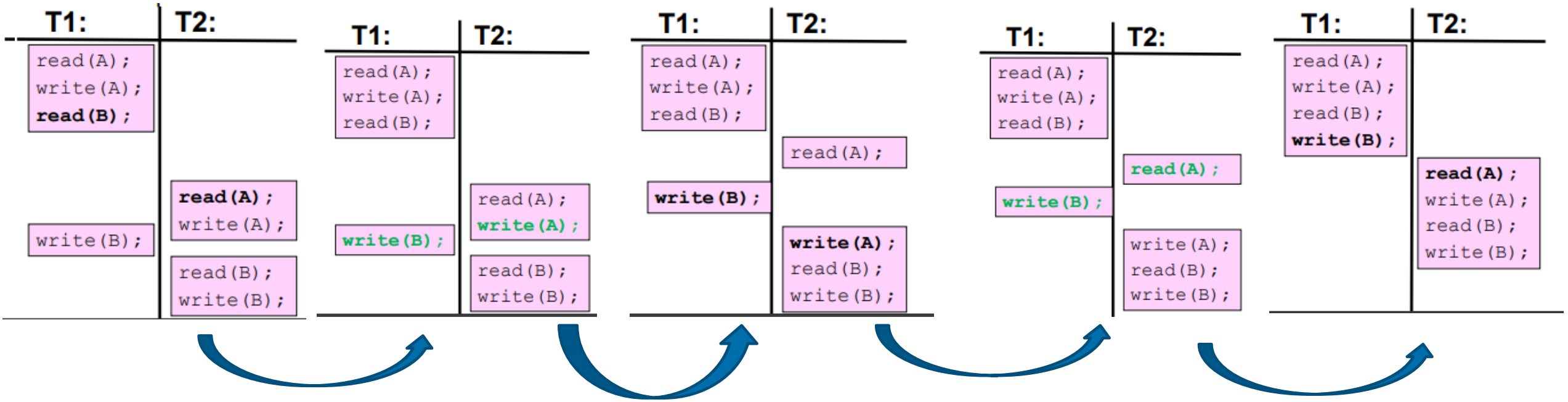


- Considerando I_i e I_j como instruções consecutivas de um schedule S:
  - Se I_i e I_j forem instruções de diferentes transações e não conflitarem, podemos inverter a ordem I_i e I_j, gerando um novo schedule S, ou seja, schedules equivalentes.



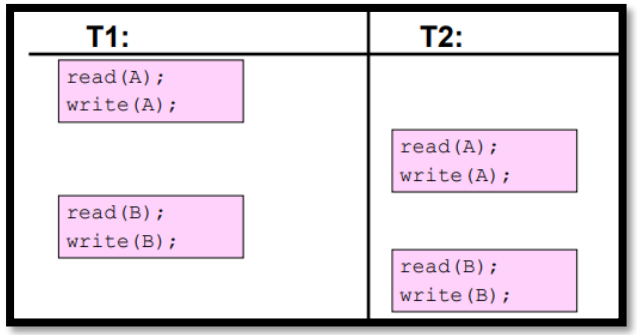
EXECUÇÕES SIMULTÂNEAS

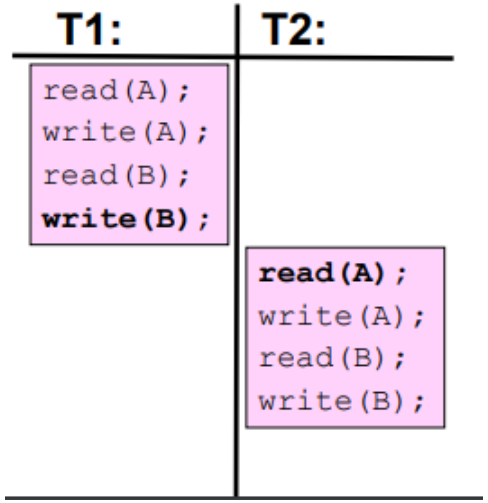
- Considerando I_i e I_j como instruções consecutivas de um schedule S:
  - Se I_i e I_j forem instruções de diferentes transações e não conflitarem, podemos inverter a ordem I_i e I_j, gerando um um novo schedule S, ou seja, schedules equivalentes.
  - Se continuarmos a inverter as instruções não conflitantes, obtemos um schedule equivalente a um schedule serial.



SERIAÇÃO DE CONFLITO

- Considerando I_i e I_j como instruções consecutivas de um schedule S:
  - Se um schedule S puder ser transformado em um schedule S´ por uma série de trocas de instruções não conflitantes, dizemos que S e S' são equivalentes em conflito. Já que todas as instruções aparecem na mesma ordem em ambas as escalas de execução com exceção de Ii e Ij, cuja ordem não importa.



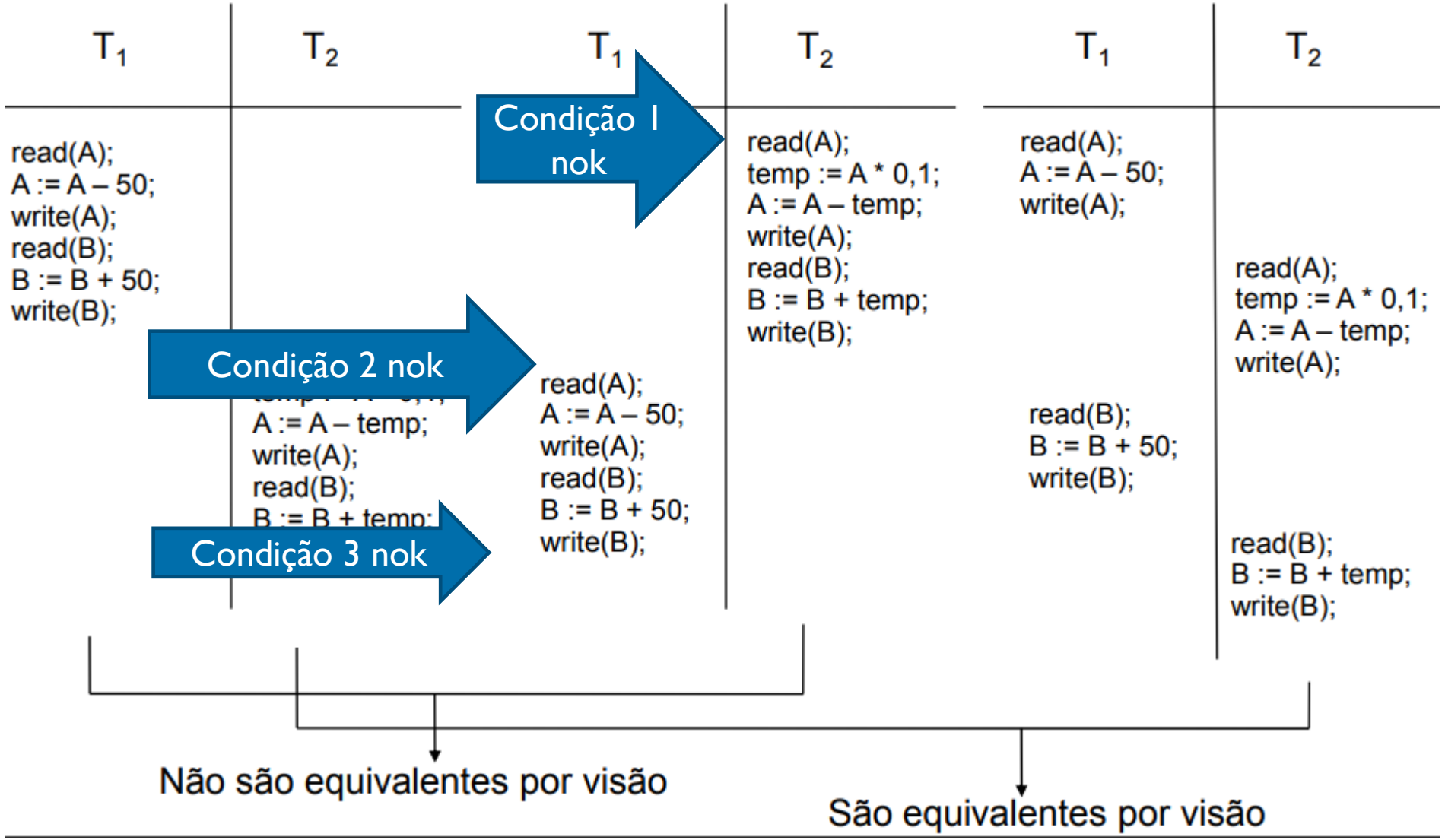


centro universitário FEI

SERIAÇÃO DE VISÃO

- Menos rigorosa do que equivalência em conflito.
- Dois schedules S e S' são equivalentes em visão se:
- Para cada item de dados Q, se a transação T_i ler o valor de Q no schedule S
  - Então a transação T_i em S' também precisa ler o valor inicial de Q.
  - Ou seja: schedule inicial e final têm que ter um read (Q) correspondente
- Para cada item de dados Q, se a transação T_i executar read(Q) no schedule S:
  - se este valor foi produzido por uma operação write(Q) executada pela transação T_j, então a operação read(Q) da transação T_i no schedule S' também precisa ler o valor inicial de Q que foi produzido pela mesma operação write (Q) da transação Tj .
- Para cada item de dados Q:
  - a transação que realiza a operação write(Q) final no schedule S precisa realizar a operação write(Q) final no schedule S´.
- Considerada seriação de visão se for equivalente à algum schedule serial

Condição 1: ler a mesma variável, na mesma transação, na mesma ordem.: se leu primeiro A em T1 em S1 A precisa ser lido em primeiro também
Condição 2: as variáveis precisam ser atualizadas nas mesmas transações e as leituras a partir da atualização também!
Condição 3: o write final precisa ser feito na mesma transação



https://www.inf.unioeste.br/~clodis/BDII/BDII_Modulo_1.pdf

VERIFICANDO A SERIAÇÃO

- Esquemas de controle de concorrência geram schedules que devem ser passíveis de seriação. Como verificar?
- Construir o gráfico de precedência:
  - par G=(V,E), onde V é um conjunto de vértices e E é um conjunto de arestas.
  - vértices: todas as transações participantes do schedule
  - arestas: arestas $T_i \rightarrow T_j$ para as quais uma das condições é verdadeira:
    - T_i executa write(Q) antes que T_j execute read (Q);
    - T_i executa read(Q) antes que T_j execute write(Q);
    - T_i executa write(Q) antes que T_j execute write (Q).
  - se uma aresta $T_i \rightarrow T_j$ existir no gráfico de precedência, em qualquer schedule serial S´ equivalente a S, T_i precisa aparecer antes de T_j

VERIFICANDO A SERIAÇÃO

T1	T2

T1:

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

T2:

```
read(A);
temp := A  * 0,1;
A := A - temp;
write (A)
read (B);
B := B + temp;
write (B);
```

Gráfico de Precedência

arestas: arestas $T_i \rightarrow T_j$ para as quais uma das condições é verdadeira:

- T_i executa write(Q) antes que T_j execute read (Q);
- T_i executa read(Q) antes que T_j execute write(Q);
- T_i executa write(Q) antes que T_j execute write (Q).

VERIFICANDO A SERIAÇÃO

T1 T2

T2:

```
read(A);
temp := A  * 0,1;
A := A - temp;
write (A)
read (B);
B := B + temp;
write (B);
```

T1:

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

Gráfico de Precedência

arestas: arestas $T_i \rightarrow T_j$ para as quais uma das condições é verdadeira:

- T_i executa write(Q) antes que T_j execute read (Q);
- T_i executa read(Q) antes que T_j execute write(Q);
- T_i executa write(Q) antes que T_j execute write (Q).

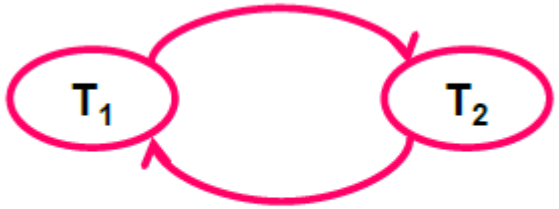
VERIFICANDO A SERIAÇÃO

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);`

Gráfico de Precedência



Se gráfico tiver um ciclo, o *schedule* não é serial de conflito.

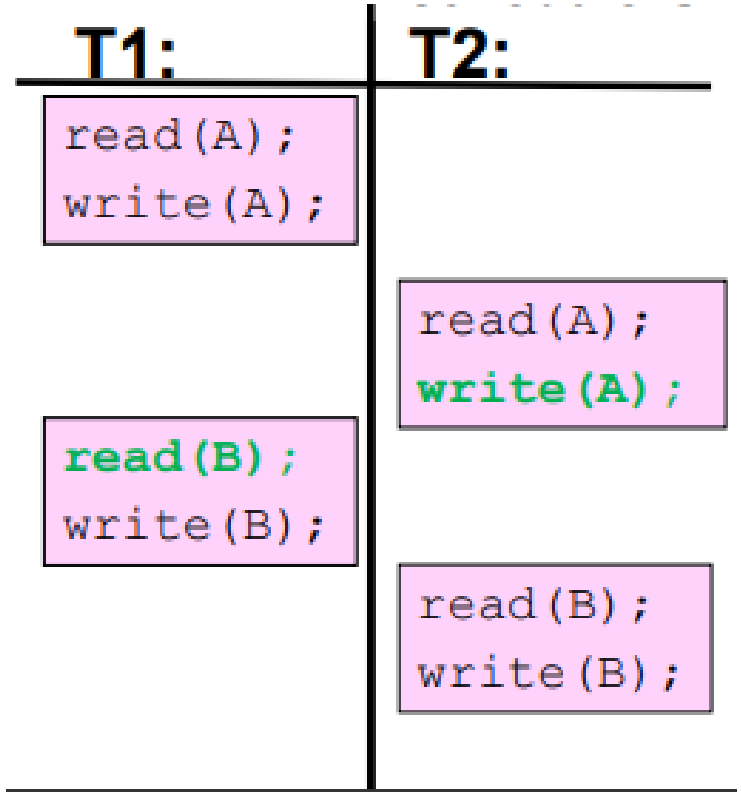
arestas: arestas $T_i \rightarrow T_j$ para as quais uma das condições é verdadeira:

- T_i executa write(Q) antes que T_j execute read (Q);
- T_i executa read(Q) antes que T_j execute write(Q);
- T_i executa write(Q) antes que T_j execute write (Q).

VERIFICANDO A SERIAÇÃO

- Ordem de seriação:
  - obtida pela classificação topológica do gráfico de precedência
  - ordem linear consistente com a ordem parcial do gráfico de precedência
  - em geral existem várias ordens de seriação possíveis.

POSSUI CICLOS?



arestas: arestas $T_i \rightarrow T_j$ para as quais uma das condições é verdadeira:

T_i executa write(Q) antes que T_j execute read (Q);

T_i executa read(Q) antes que T_j execute write(Q);

T_i executa write(Q) antes que T_j execute write (Q).

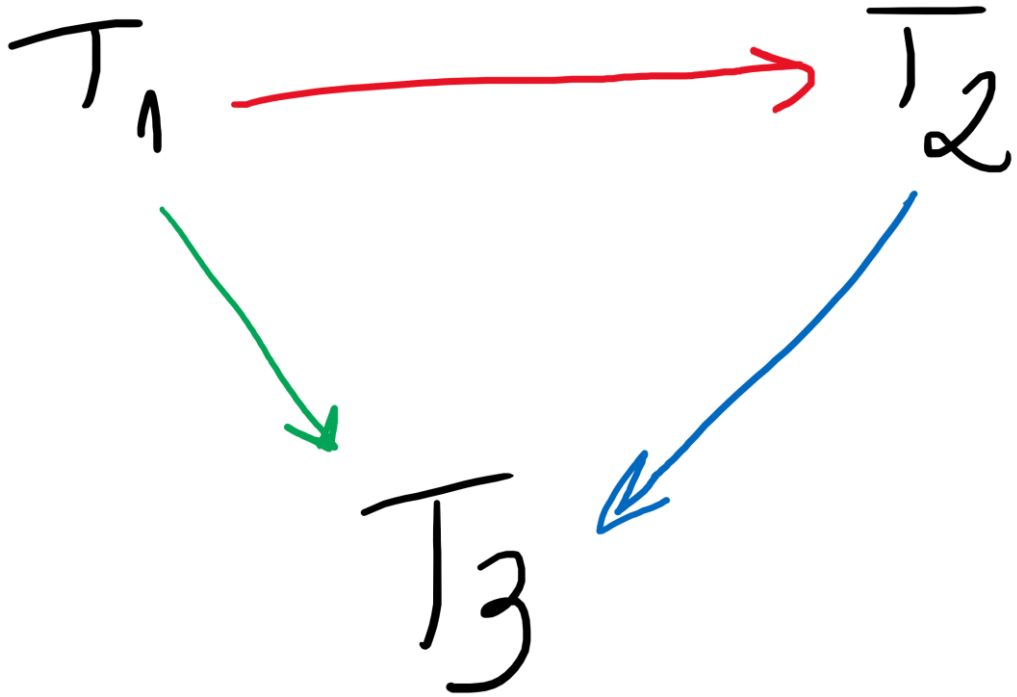
COM DIFERENTES COMPONENTES

T4	T5	T6
Read(A)		
A:= f1(A)		
Read(C)		
Write(A)		
A:= f2(C)		
	Read(B)	
Write(C)		
	Read(A)	
		Read(C)
	B:= f3(B)	
	Write(B)	
		C:= f4(C)
		Read(B)
		Write(C)
	A:=f5(A)	
	Write(A)	
		B:= f6(B)
		Write(B)

Time

Schedule S2

A
B
C



arestas: arestas $T_i \rightarrow T_j$ para as quais uma das condições é verdadeira:

T_i executa write(Q) antes que T_j execute read (Q);

T_i executa read(Q) antes que T_j execute write(Q);

T_i executa write(Q) antes que T_j execute write (Q).

VERIFICANDO A SERIAÇÃO

- Para testar seriação de conflito:
  - construir gráfico de precedência;
  - executar algoritmos de detecção de ciclos

- Ainda não existe algoritmo eficiente para testar seriação de visão: verifica-se as condições suficientes citadas

BIBLIOGRAFIA

- ABRAHAM SILBERSCHATZ, HENRY F. KORTH, S. SUDARSHAN. Sistema de Banco de Dados. 6. Campus. 0. ISBN 9788535245356.

- ELMASRI, RAMEZ, SHAMKANT B. NAVATHE. Sistemas de banco de dados. Vol. 6. São Paulo: Pearson Addison Wesley, 2011.

- DATE, CHRISTHOPER J. Introdução a Sistemas de Bancos de Dados, 5ª. Edição. Campus, Rio de Janeiro (2004).

OBRIGADO E ATÉ A PRÓXIMA AULA!