



## **PCS3413**

# Engenharia de Software e Banco de Dados

## **Aula 21**

# **TRANSAÇÃO E CONTROLE DE CONCORRÊNCIA**

# Transação

- unidade de execução de programa que acessa e, possivelmente, atualiza vários itens de dados.
- do ponto de vista do usuário:
  - podem ser várias operações.
- do ponto de vista do BD:
  - uma única operação.

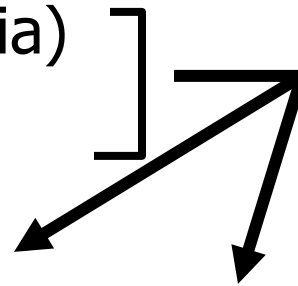
## transação – uma operação - exemplo

- transferência de fundos de uma conta corrente para uma conta investimentos

cc (nconta, saldo<sub>cc</sub>,nagência)

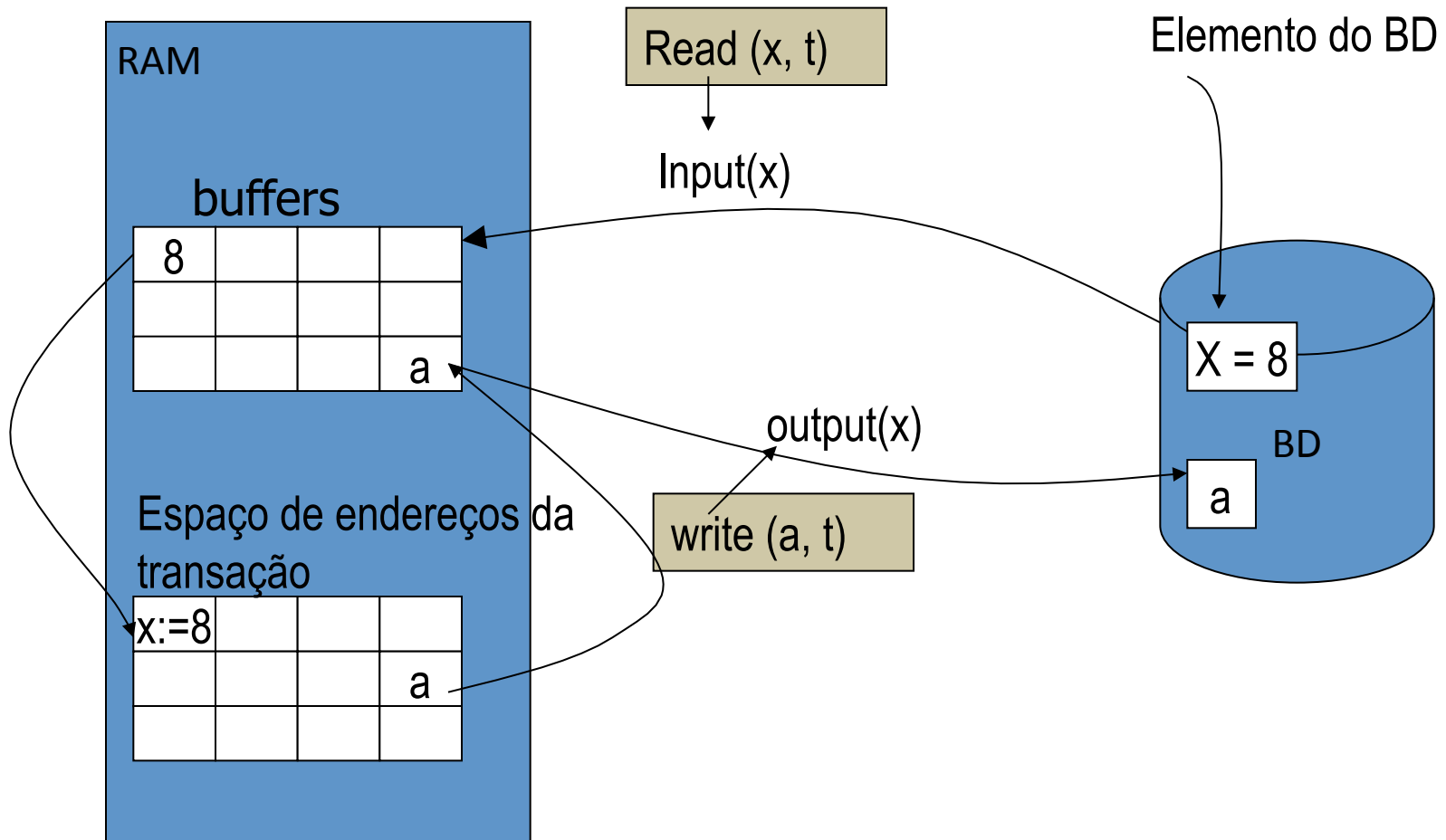
ci (nci,nconta, saldo<sub>ci</sub>)

```
update cc
set saldocc=saldocc-1000
where nconta=0234
```



```
update ci
set saldoci=saldoci+1000
where nconta=0234
```

# ◆ análise das operações: update



## exemplo – continuação

```
update cc  
set saldooc=saldooc-1000  
where nconta=0234
```

read  
(cc.saldooc)

saldooc=3000

write  
(cc.saldosc)

saldooc=2000

```
update ci  
set saldoci=saldoci+1000  
where nconta=0234
```

read  
(ci.saldoci)

saldoci=12000

write  
(ci.saldoci)

saldoci=13000

**FALHA**

**BANCO DE DADOS INCONSISTENTE**

# Transação - continuação

- precisa que as duas operações sejam executadas como se fossem uma única operação!

Begin transaction

op1

op2

op3

...

opn

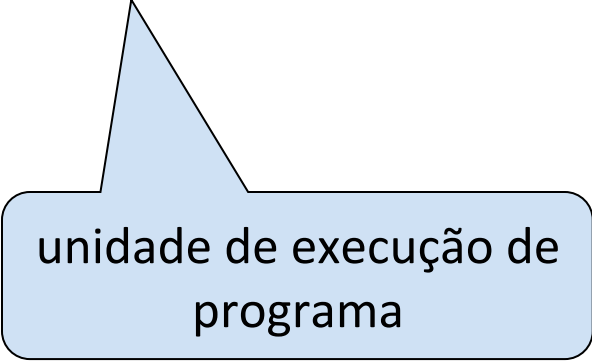
end transaction

corpo da  
transação

uma operação

# Propriedade de Transações

- ACID
  - Atomicidade
  - Consistência
  - Isolamento
  - Durabilidade



unidade de execução de  
programa

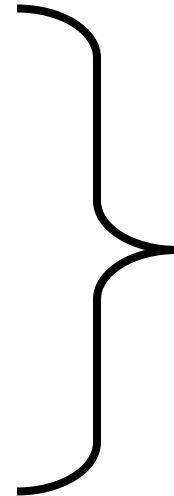


# Atomicidade

- todas as operações são executadas e refletidas no BD ou nenhuma é.

```
update cc  
set saldooc=saldooc-1000  
where nconta=0234
```

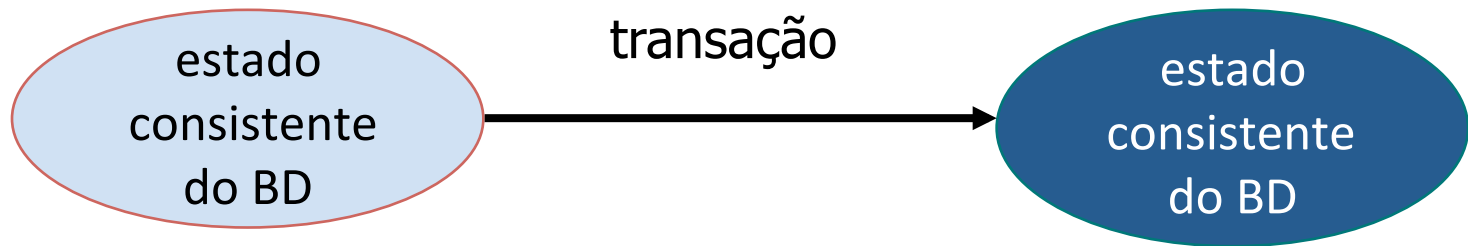
```
update ci  
set saldoci=saldoci+1000  
where nconta=0234
```



como se fosse  
uma única  
operação

# Consistência

- a execução de uma transação isolada (sem concorrência) preserva a consistência do BD.



# Durabilidade

- os efeitos da transação bem sucedida persistem no BD mesmo se houver falhas.
  - propriedade assegurada pelo componente de gerenciamento de recuperação

# Isolamento

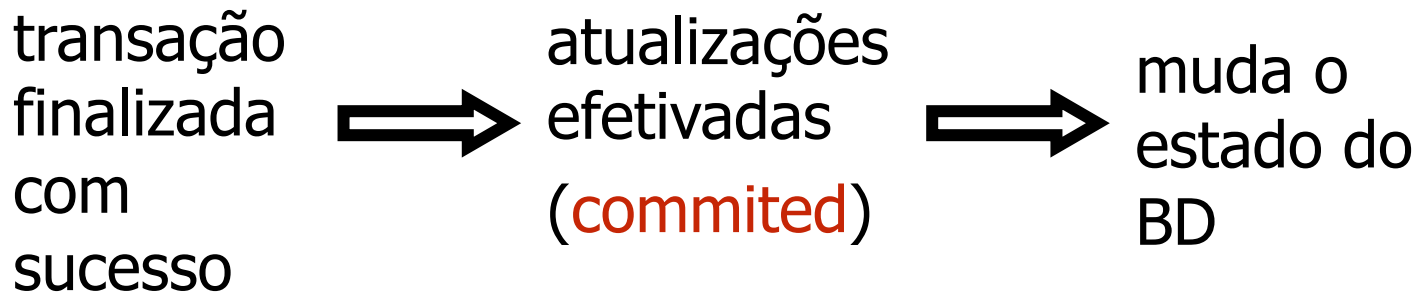
- podem existir diversas transações em execução, atualizações feita por uma transação são “escondidas” da outra, até que a primeira termine.
  - o nível de isolamento afeta diretamente a concorrência ao item de dado num ambiente multi-usuário.

# Estados de uma transação

- quando uma transação não termina com sucesso : abortada



- quando uma transação termina com sucesso : **committed**

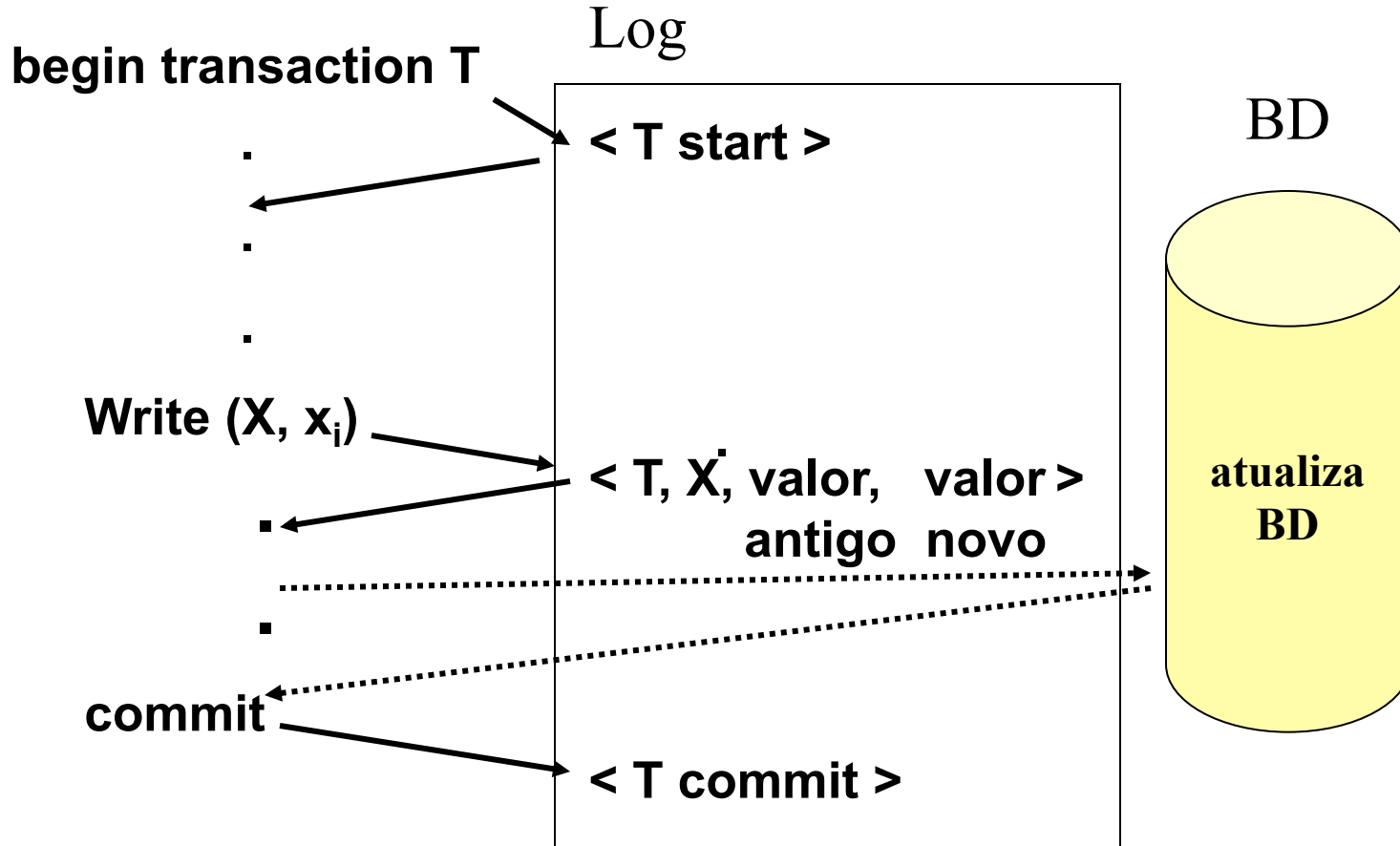


# **RECUPERAÇÃO DE FALHAS**

# Falhas

- mais simples: que resultem em perda de informação na memória volátil.
- Há vários mecanismos que fazem o sistema voltar ao último estado consistente antes da ocorrência da falha.
- Alguns usam o “log” (arquivo que registra as alterações no BD; o log é implementado numa *memória estável*).

# Log incremental com atualização imediata



# Exemplo – execução serial

- duas transações  $T_0$  e  $T_1$  que atualizam saldos de contas.

saldocc=3000 e saldoci=12000

$T_0$ : begin transaction

read (saldocc,  $cc_1$ )

$cc_1 := cc_1 - 1000$

write (saldocc,  $cc_1$ )

read (saldoci,  $ci_1$ )

$ci_1 := ci_1 + 1000$

write (saldoci,  $ci_1$ )

commit

$T_1$ : begin transaction

read (saldocc,  $cc_1$ )

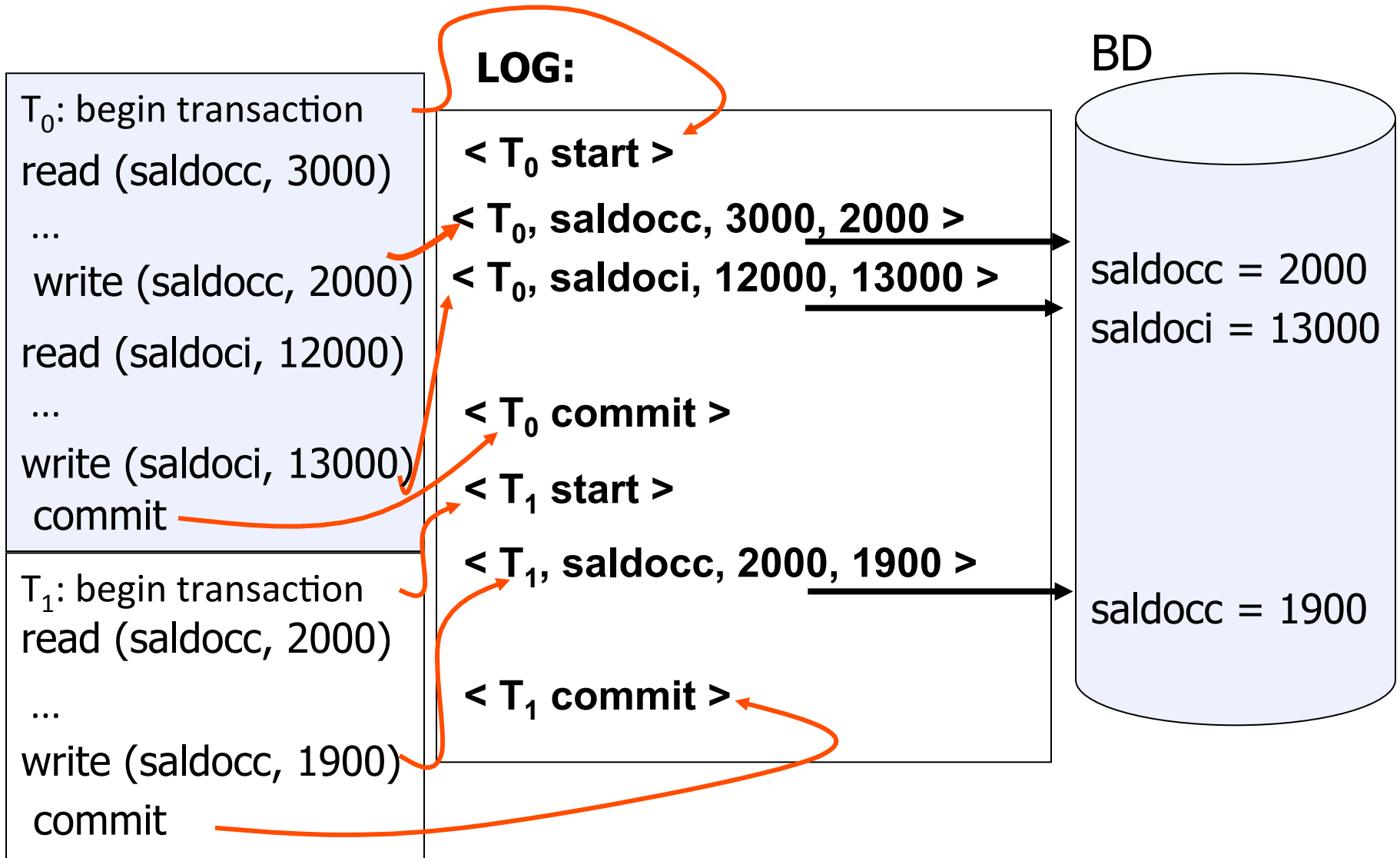
$cc_1 := cc_1 - 100$

write (saldocc,  $cc_1$ )

commit



# execução serial: $T_0 \rightarrow T_1$



# Exemplos de Falha

- falha durante a execução de  $T_0$ :

```
LOG:  < T0 start >  
      < T0, saldooc, 3000, 2000 >  
      < T0, saldoci, 12000, 13000 >
```

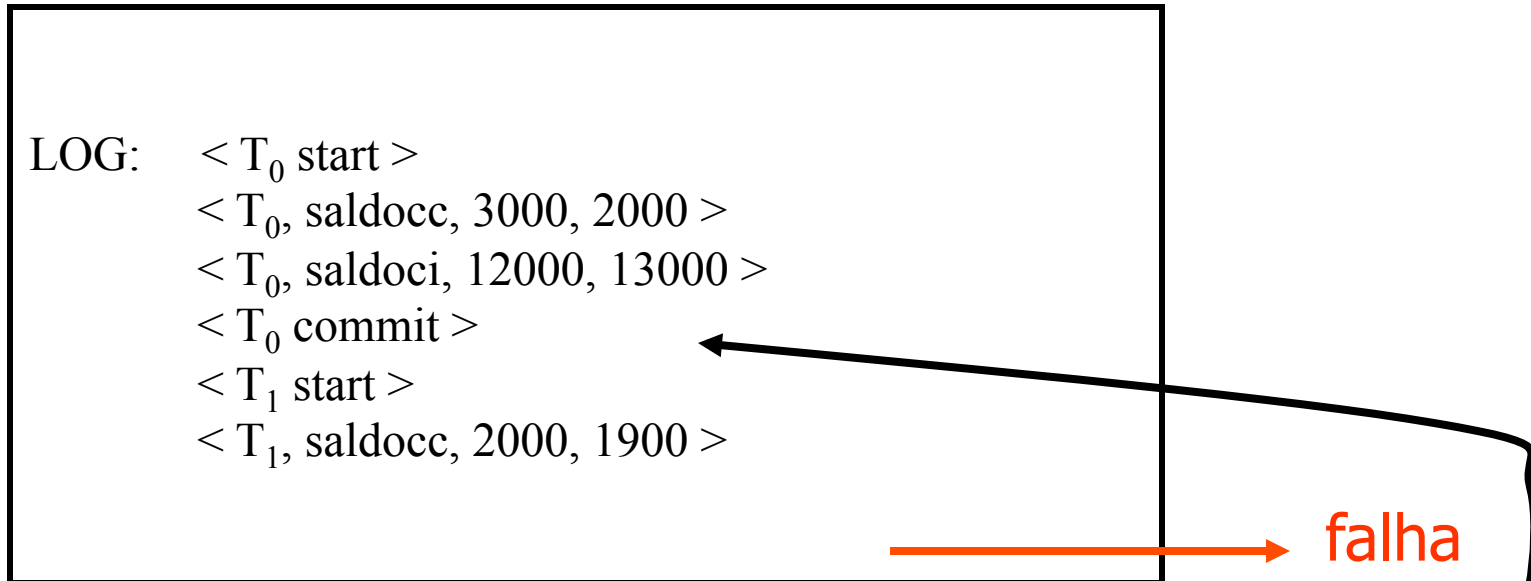
→ falha

O sistema faz retroativamente:

- UNDO ( $T_0$ ) ou ROLLBACK ( $T_0$ ):  
desfaz as eventuais atualizações de  $T_0$ .

# Mais Exemplos de Falha

- falha durante a execução de  $T_1$ :



O sistema faz retroativamente:

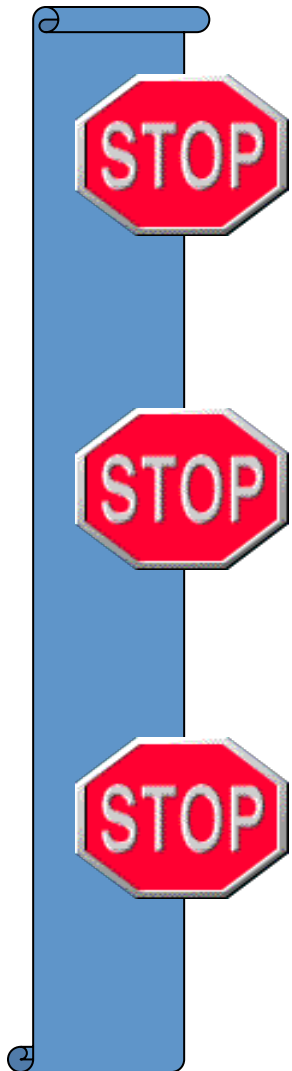
- REDO ( $T_0$ ), pois  $T_0$  tem registro commit no log.
- UNDO ( $T_1$ ), pois  $T_1$  não tem registro commit.

# Pontos de Sincronismo (Checkpoints)

- Após a falha, quando o sistema volta, o gerenciador de recuperação examina o log para tomar as providências cabíveis:
  - refazer (REDO) ou
  - desfazer (UNDO ou ROLLBACK) transações.

- é um processo demorado
- muitas das transações já atualizaram o BD, e acabam sendo refeitas sem necessidade.

# Pontos de Sincronismo (Checkpoints)



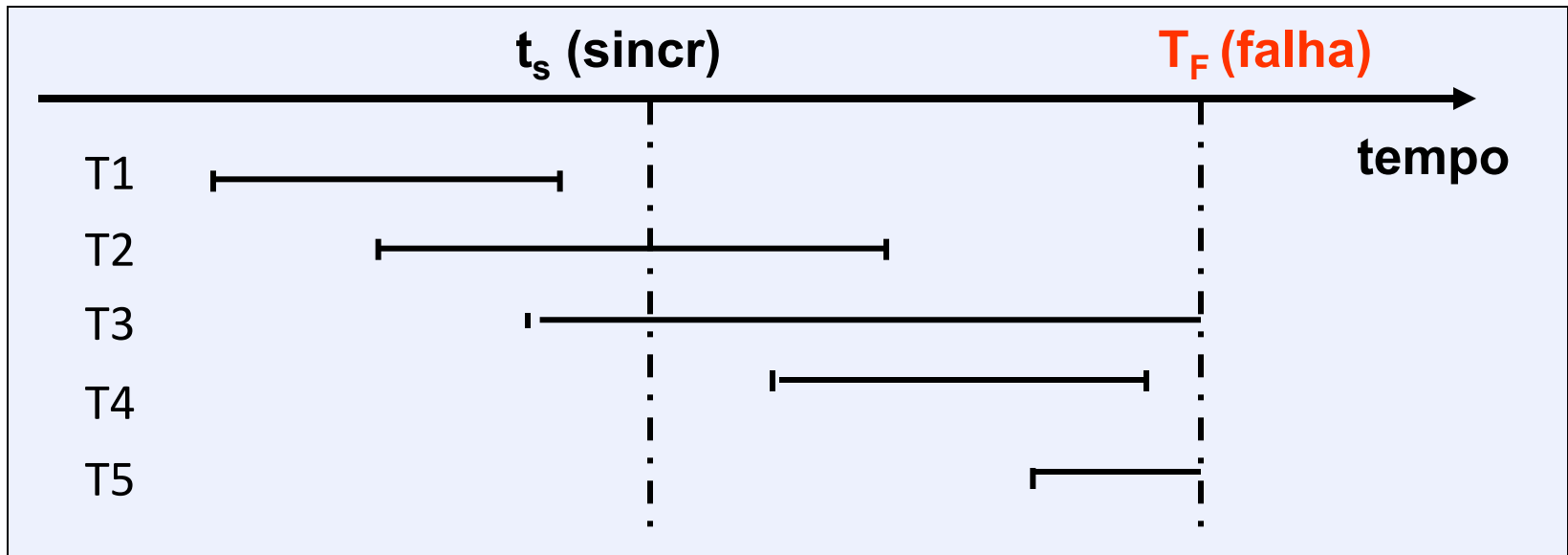
→ **CHECKPOINTS** (pontos de sincronismo)

atualização do log (da memória principal) →  
memória estável (registros do log)

atualização do BD - blocos de buffer (da  
memória principal) → disco

registro *<checkpoint>* → log (em memória  
estável)

# Exemplo - checkpoint



T1 - nada a fazer: o BD já foi atualizado no último ponto de sincronismo.

T2, T4 - devem ser refeitas: apesar de terminadas, não há garantia de que elas atualizaram o BD.

T3, T5 - devem ser desfeitas: ainda não terminaram.

# **CONTROLE DE CONCORRÊNCIA**

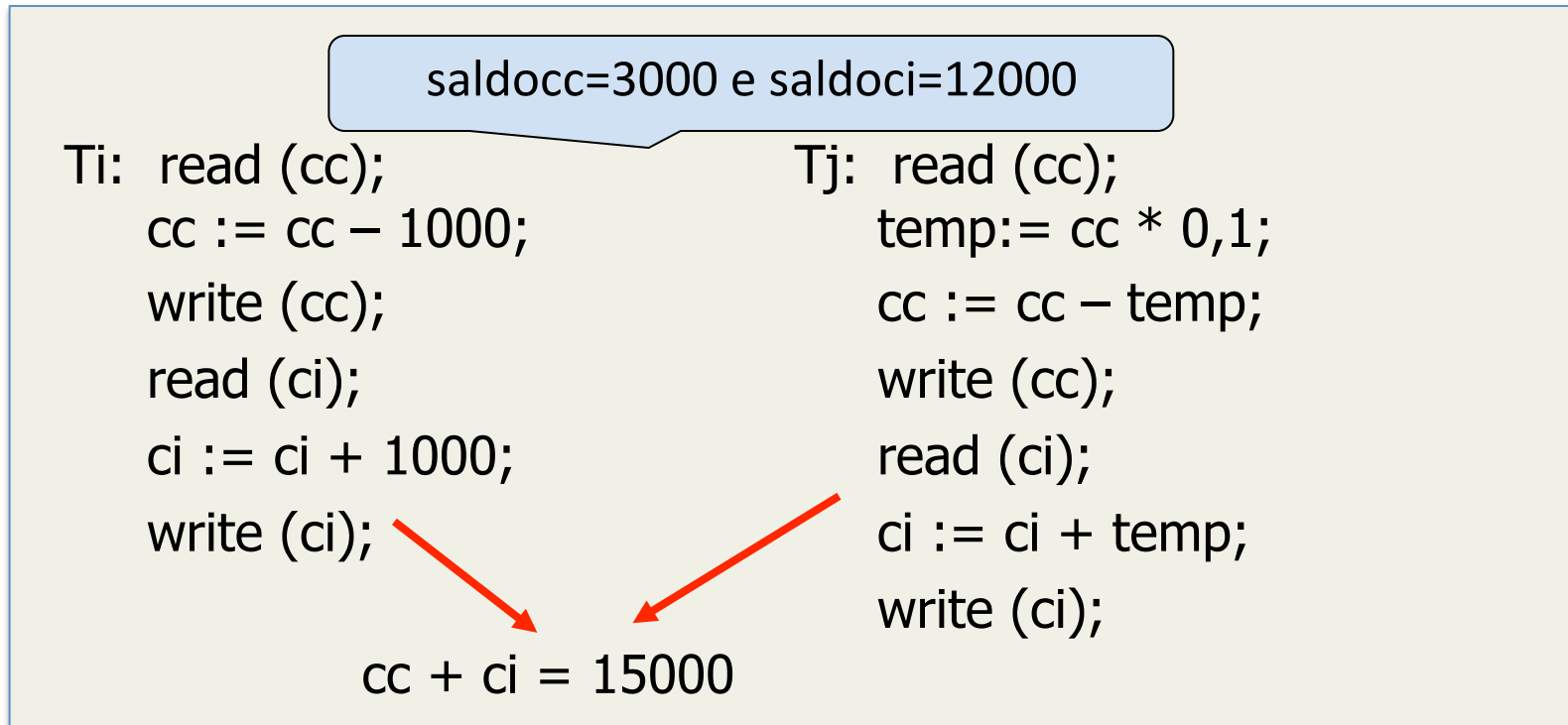
# Controle de Concorrência

- problema: **ambientes multiusuário**
  - num ambiente onde transações são executadas de modo concorrente a propriedade do isolamento pode não ser preservada.
- Quando diversas transações concorrentes são executadas, suas operações podem ser intercaladas de modo inconvenientemente.



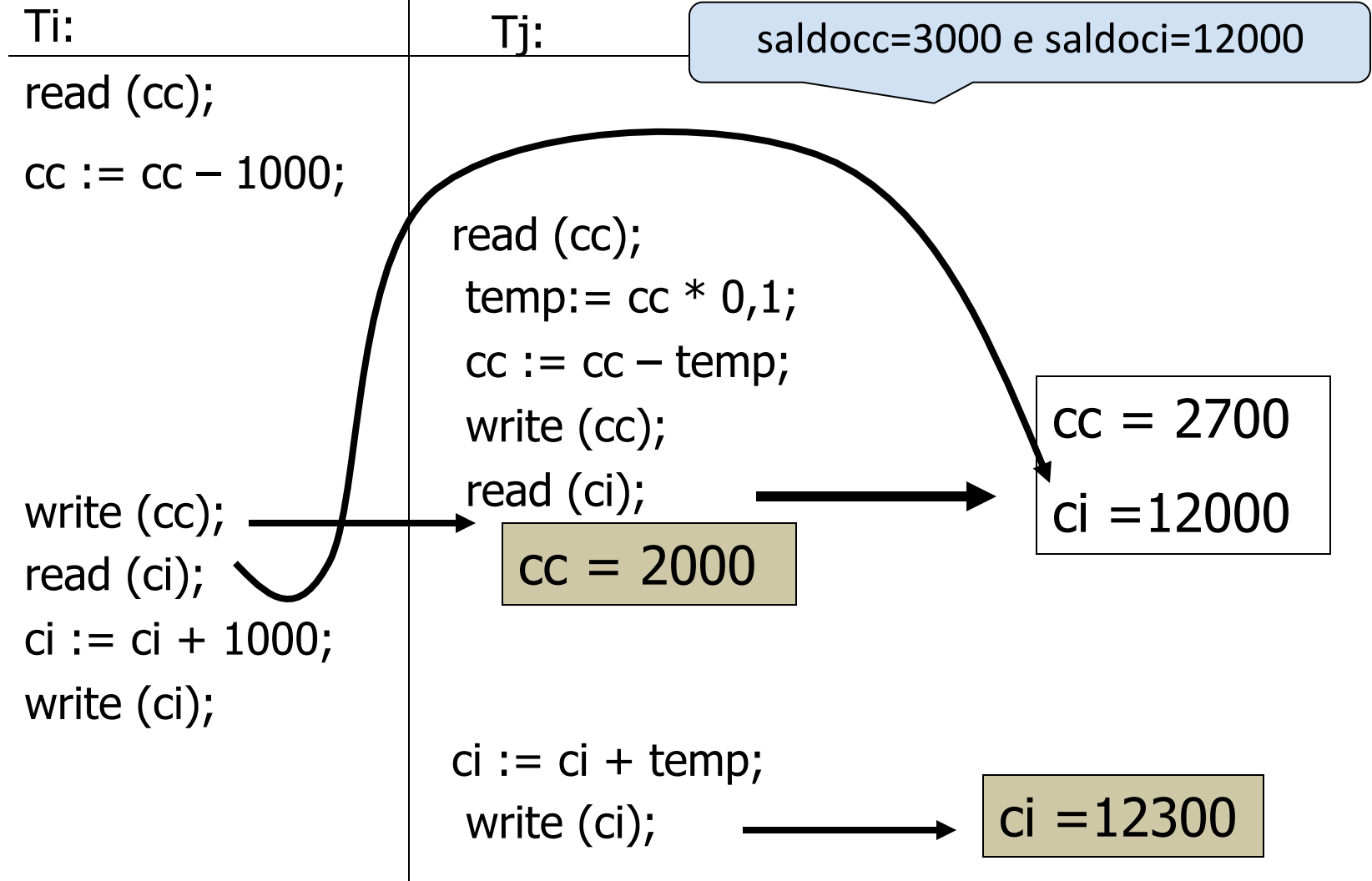
# Isolamento

- podem existir diversas transações em execução, atualizações feita por uma transação são “escondidas” da outra, até que a primeira termine.
  - o nível de isolamento afeta diretamente a concorrência ao item de dado num ambiente multi-usuário.



# execução paralela:

# exemplo de escalonamento



$cc + ci = 2000 + 12300 = 14300$  **inconsistência**

# serializabilidade (serializability)

- É a propriedade (assegurada pelo sistema) de que o resultado da execução concorrente de um conjunto de transações seja o mesmo que o produzido pela sua execução serial.

## Serializabilidade de Conflito

- Garantido por sistemas comerciais.
- Garante que o escalonamento obedece a propriedade da serializabilidade e que a ordem de execução das ações de transações no escalonamento respeita o resultado que deveria ser apresentado se as ações fossem seriais. Ou  
 $T_i \rightarrow T_j$  escalonamento serial, então  
Ações de  $T_i \rightarrow T_j$  em qualquer escalonamento são respeitadas

# Serializabilidade de conflito -exemplo

```
cc = 3000  
ci = 12000
```

```
Ti: read (cc);  
   cc := cc - 1000;  
   write (cc);  
   read (ci);  
   ci := ci + 1000;  
   write (ci);
```

```
Tj: read (cc);  
   temp:= cc * 0,1;  
   cc := cc - temp;  
   write (cc);  
   read (ci);  
   ci := ci + temp;  
   write (ci);
```

Escalonamento serial:  $T_i \rightarrow T_j$  ou  $T_j \rightarrow T_i$

Consistência:  $cc + ci = 15000$

Serializabilidade, mas  
não a

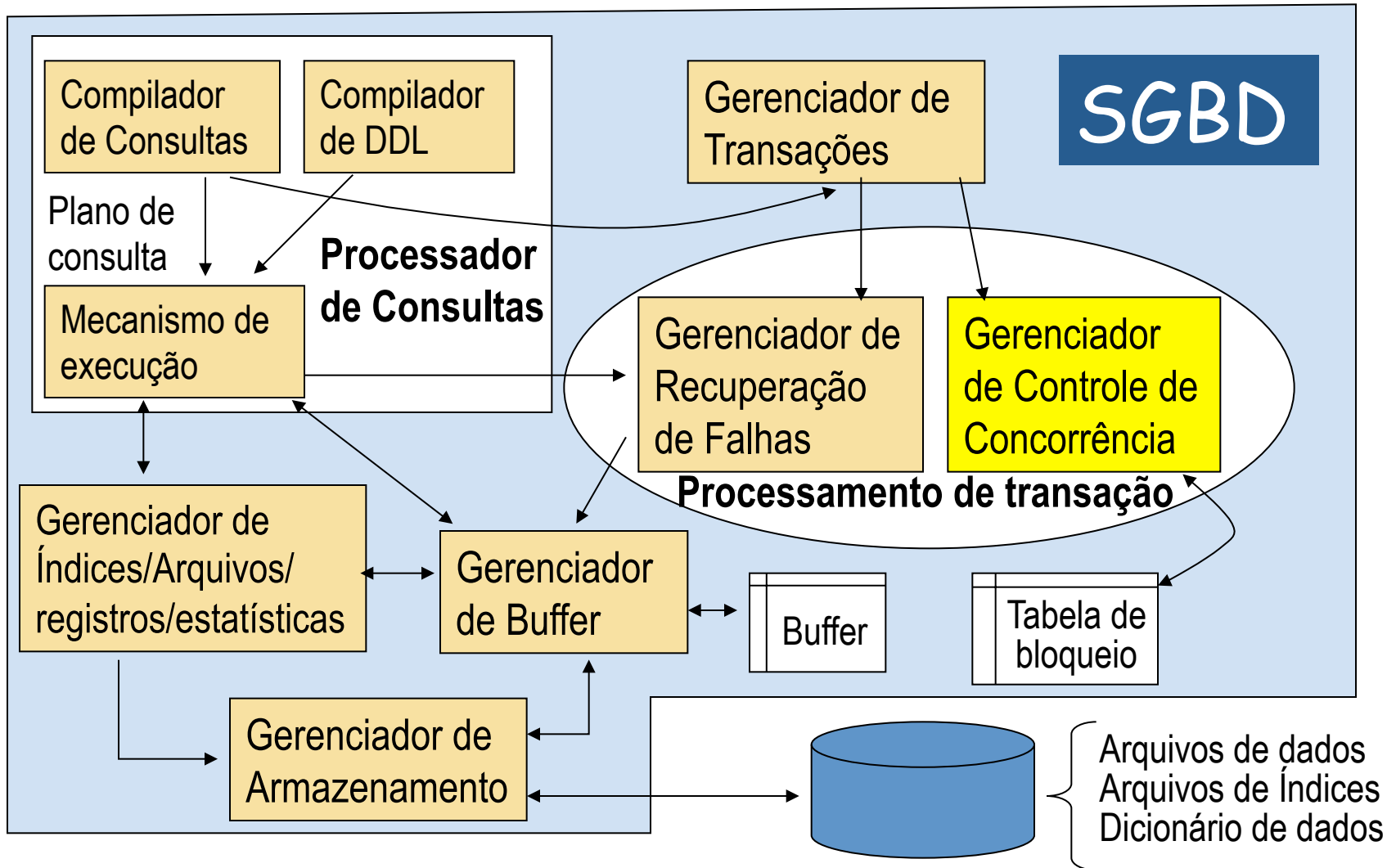
$T_i \rightarrow T_j$  :  $cc = 1.800$  e  $ci = 13.200$

$T_j \rightarrow T_i$  :  $cc = 1.700$  e  $ci = 13.300$

Consistência:  $cc + ci = 15000$

serializabilidade de  
conflito

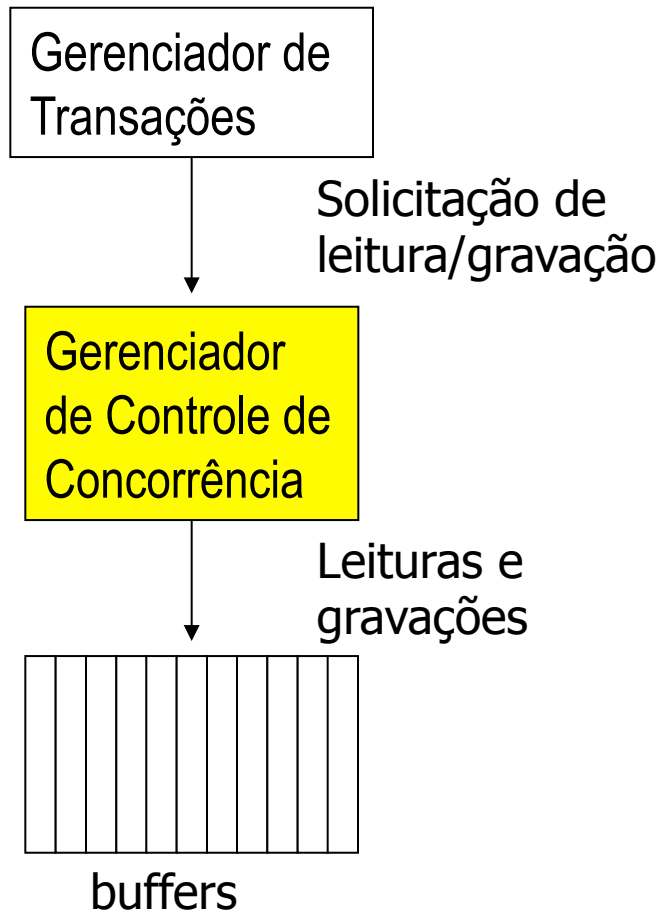
# Funções e Componentes de um SGBD



# Gerenciador de Controle de Concorrência ou Escalonador (*Scheduler*)

- As transações devem aparentar serem executadas em isolamento. Porém, na maioria dos sistemas, haverá muitas transações sendo executadas ao mesmo tempo. O scheduler deve assegurar que as ações individuais das várias transações sejam executadas em tal ordem, de forma que o efeito líquido seja igual ao que seria se as transações fossem realmente executadas em sua totalidade, uma de cada vez. Um scheduler típico utiliza bloqueios sobre os itens do banco de dados para essa finalidade. Esses bloqueios impedem que duas transações tenham acesso ao mesmo item de dado por meios que interajam de forma incorreta. Em geral, os bloqueios são armazenados em tabelas de bloqueios na memória principal.

# Escalonador ou *Scheduler*



Solicitações de leitura e gravação por parte de transações são repassadas ao escalonador. Se o elemento do banco de dados não estiver em memória, o escalonador chama o gerenciador de buffers. O escalonador pode adiar a solicitação ou até mesmo abortar a transação que emitiu a solicitação.

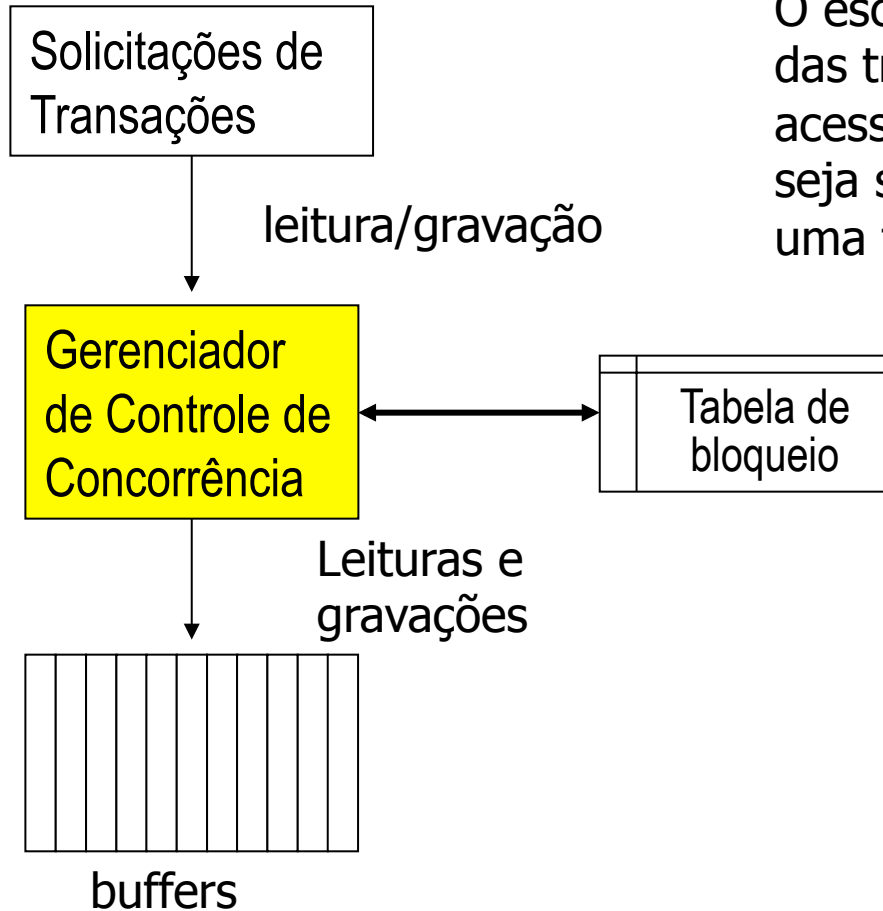
# Bloqueio

- para garantir que um escalonamento garanta a consistência do BD, isto é, seja serializável é necessário que o acesso ao dado seja feito de modo mutuamente exclusivo.
  - enquanto uma transação acessa um dado outra não poderá modificá-lo.
  - bloqueio (lock)



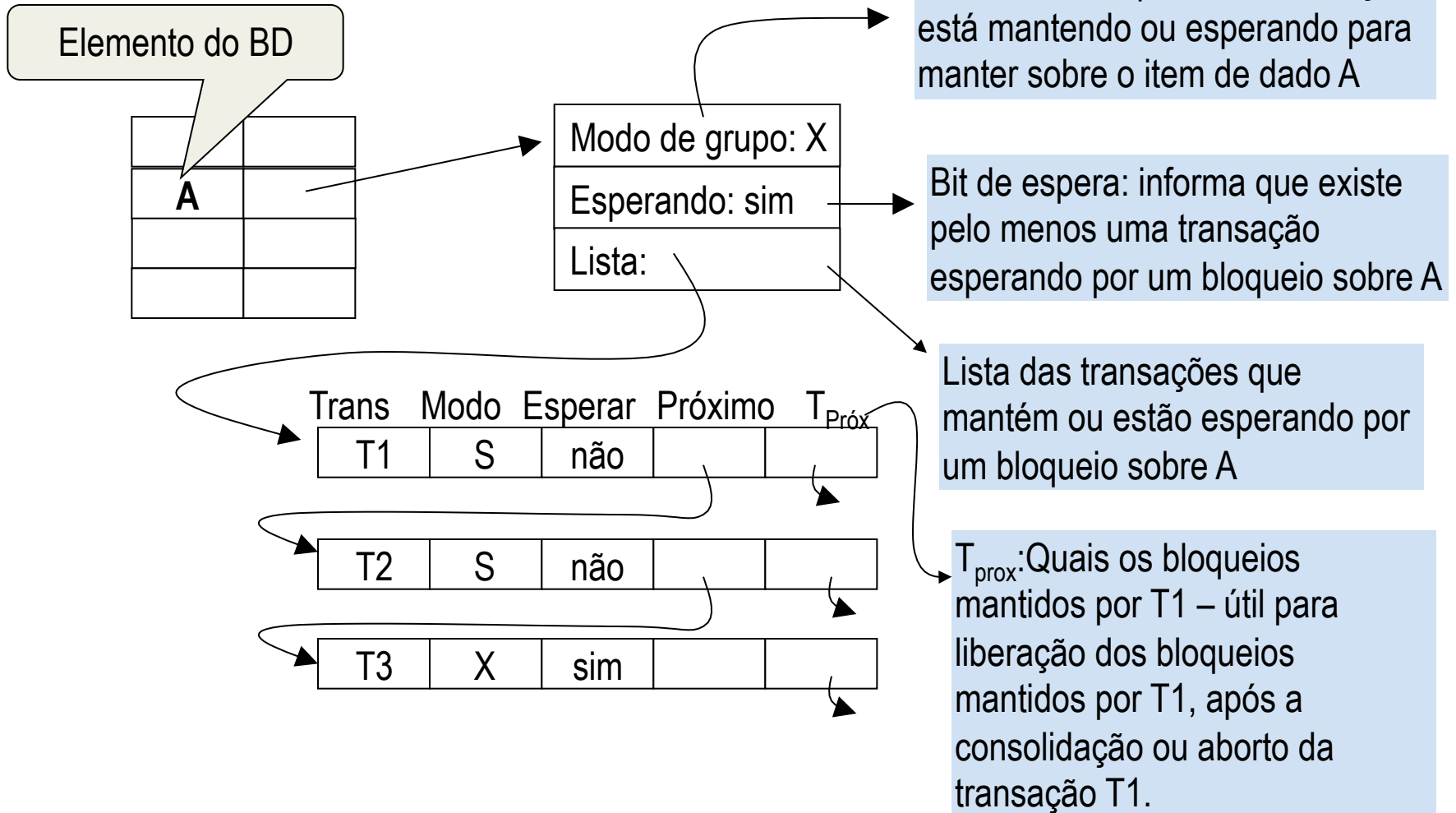


# Bloqueios



O escalonador recebe solicitações das transações e decide se elas vão acessar o dado ou esperar até que seja seguro o acesso. Para isso usa uma tabela de bloqueios

# Tabela de bloqueio



# exercício

- Verificar nos SGBDs listados a seguir quais informações são mantidas nas tabelas de bloqueio.
- Obs: os SGBDs comerciais mantêm views que permitem verificar quais bloqueios estão sendo mantidos ou estão em espera e sobre que elementos do BD (um item de dado, uma relação, o BD)
- SGBDs: oracle, DB2, sqlServer, postgre, outros.

# Modos de bloqueio

- compartilhado - só pode ler
- exclusivo - ler e escrever
  
- a transação só acessa um dado sobre o qual ela mantém algum tipo de bloqueio.
- é necessário definir o modo de bloqueio apropriado para o tipo de acesso desejado
- a transação pode realizar suas operações somente depois que o gerenciador de controle de concorrência concedeu o bloqueio a transação.

# Matriz de Compatibilidade

- Durante um bloqueio, transações com modos incompatíveis esperam.

		Bloqueio Solicitado	
		compartilhado	exclusivo
Bloqueio Mantido	compartilhado	sim	não
	exclusivo	não	não

# Instruções de bloqueio

- bloqueio compartilhado (Shared Lock)
  - lock-S(Q)
- bloqueio exclusivo (Exclusive Lock)
  - lock-X(Q)
- Liberação (Unlock)
  - unlock(Q)

# exemplo

T6 - transfere 50 da conta A para a B.

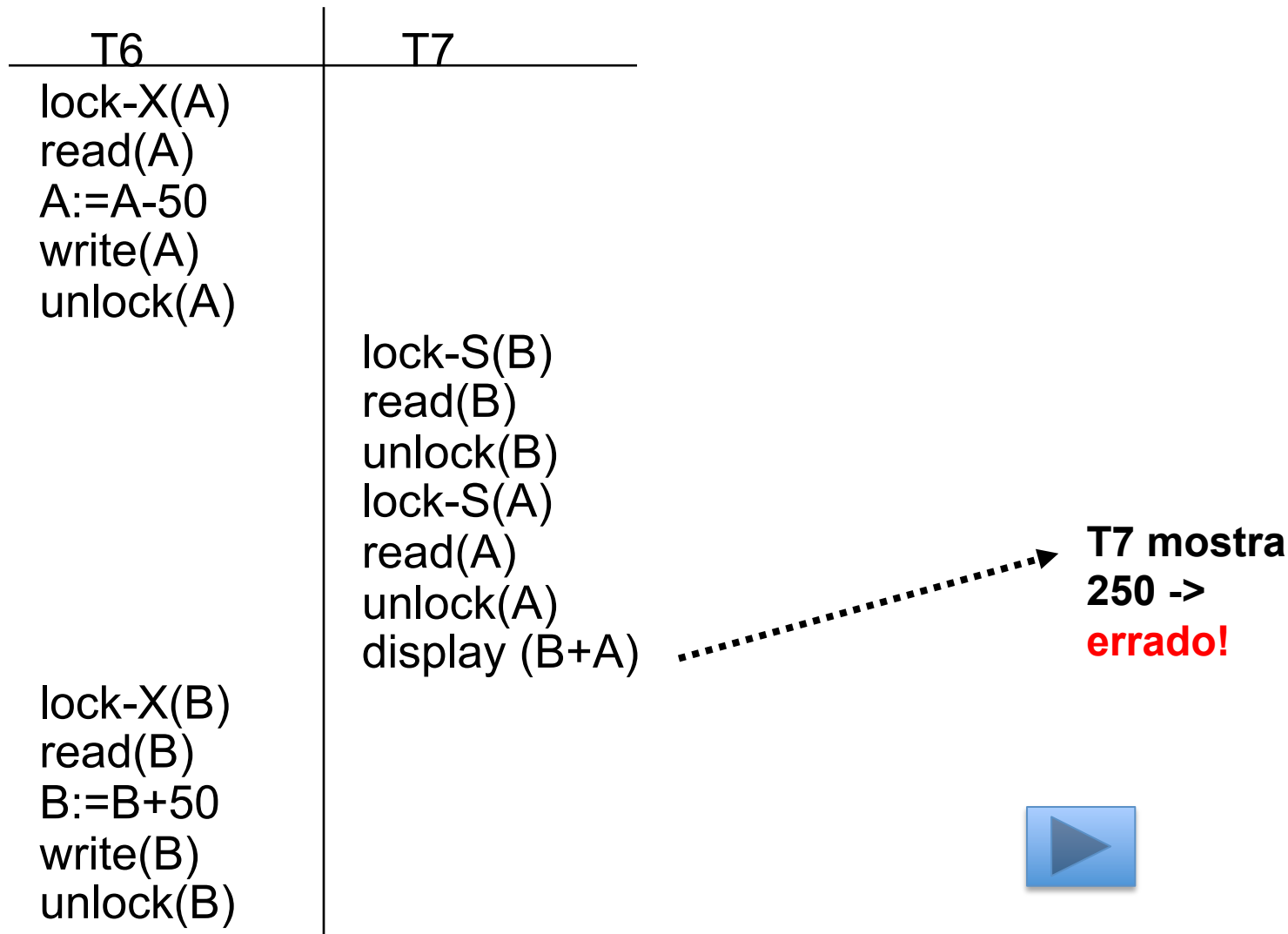
T7 - apresenta o total B + A.

```
T6: lock-X(A)
    read(A)
    A:=A-50
    write(A)
    unlock(A)
    lock-X(B)
    read(B)
    B:=B+50
    write(B)
    unlock(B)
```

```
T7: lock-S(B)
    read(B)
    unlock(B)
    lock-S(A)
    read(A)
    unlock(A)
    display (B+A)
```

Execução serial: <T6, T7> ou <T7, T6>: display (B+A) = 300

# O que acontece se o dado for liberado logo após o acesso a ele?



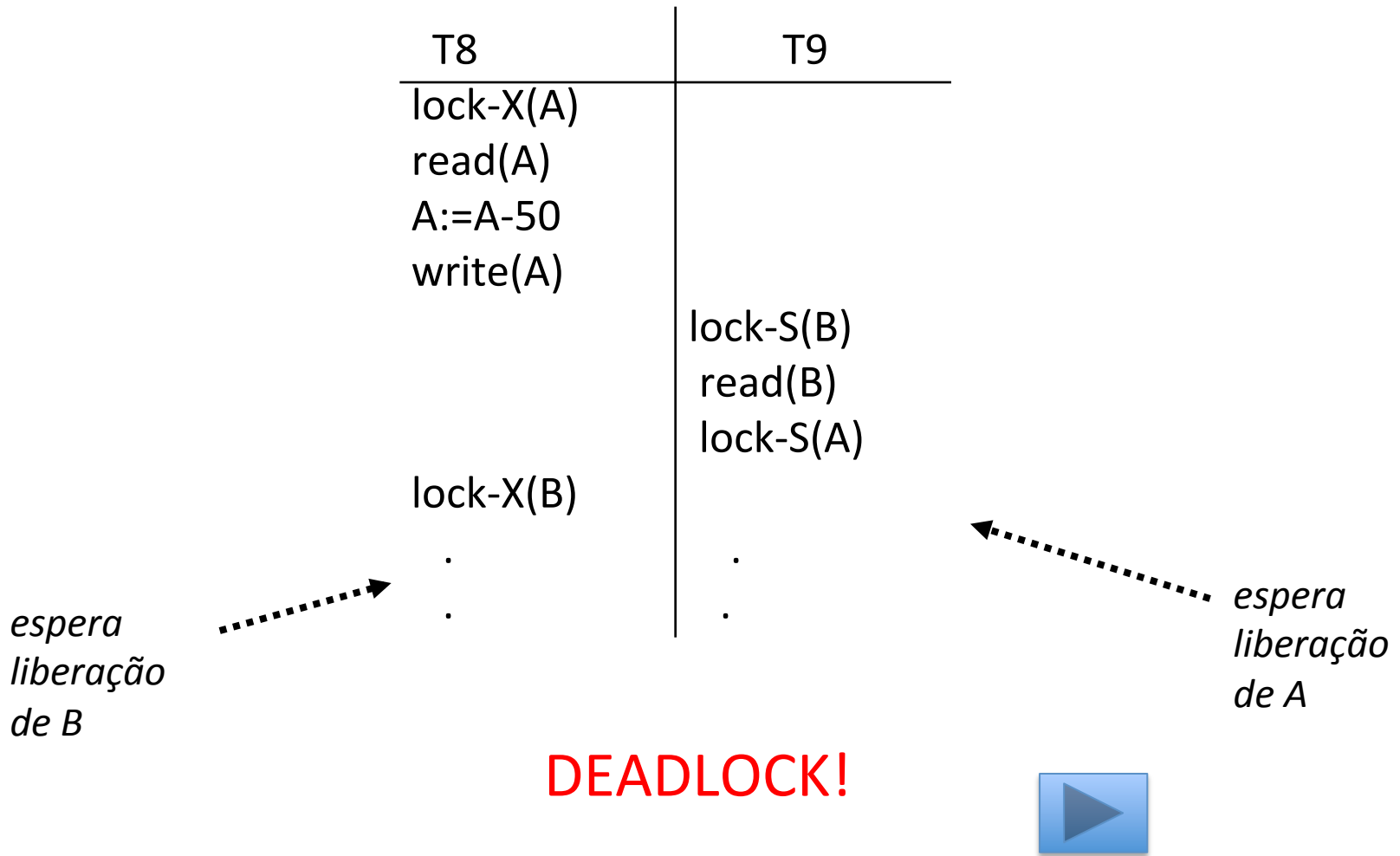


# O que acontece se a liberação for adiada?

T8: lock-X(A)  
read(A)  
A:=A-50  
write(A)  
lock-X(B)  
read(B)  
B:=B+50  
write(B)  
unlock(A)  
unlock(B)

T9: lock-S(B)  
read(B)  
lock-S(A)  
read(A)  
display (B+A)  
unlock(B)  
unlock(A)

# O que acontece se adiarmos a liberação?



# Conclusão

- Se maximizarmos a concorrência (liberação do dado o mais cedo possível)
  - estados inconsistentes
- Se não liberarmos o dado antes de bloquearmos outro, pode aparecer
  - deadlock



# Protocolos de bloqueio

- regras que indicam quando uma transação pode bloquear e liberar cada dado.

## Protocolo bifásico de bloqueio ("2 Phase Lock" – 2PL)

- Cada transação deve bloquear e liberar em duas fases:
  - Fase de expansão: a transação pode obter bloqueios mas não pode liberá-los.
  - Fase de encolhimento: a transação pode liberar bloqueios mas não pode obter outros.

# exemplo

- T8, T9 - são bifásicos 
- T6, T7 - não são bifásicos 

Tarefa:

Verificar num gerenciador comercial os tipos de bloqueio que podem ser praticados e sua granularidade (nível de tabela, de linha).

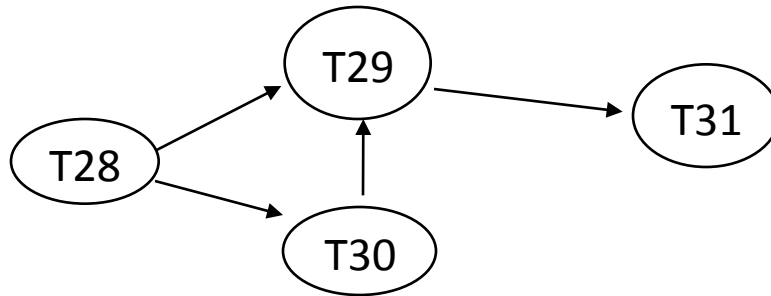
# TRATAMENTO DE DEADLOCK

- Deadlock - estado de espera.
  - O protocolo bifásico não garante imunidade contra deadlock.
- Dois métodos para lidar com deadlock:
  - Detecção e recuperação de deadlock
    - Deixar acontecer o deadlock, e depois “dar um jeito”.
    - Solução possível - desfazer alguma transação.
  - Prevenção contra deadlock
    - Não deixar entrar em deadlock. Usada quando a probabilidade e haver deadlocks é alta.

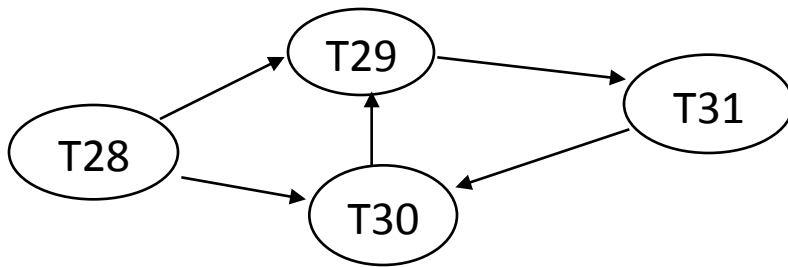
# DETEÇÃO E RECUPERAÇÃO DE DEADLOCK

- Para detetar deadlock, deverá existir um mecanismo para “monitorar” (“olhar” periodicamente) a situação.
- Para isso, o sistema deve:
  - manter informações sobre a alocação de dados às transações.
  - ter um algoritmo que usa estas informações para descobrir se está havendo algum deadlock.
  - caso for detetado algum deadlock, realizar a recuperação (“dar um jeito”).

# Grafos de Espera



não há ciclos



há ciclo:

T29->T31->T30->T29

- Detetou-se um deadlock?



# Prevenção de Deadlock


- Há várias soluções:
  - Cada transação bloqueia todos os dados que vai usar antes do início da execução das operações.
    - Desvantagem: baixo paralelismo na utilização dos dados.
    - Quando uma transação pede bloqueio de um dado já bloqueado por outra transação (mais nova ou mais velha), consulta-se sua “idade” (timestamp único) para decidir se a transação espera ou sofre rollback (caso do protocolo bifásico):
- Uso de Timeout:
  - Tempo de espera máximo antes da transação (que pediu acesso a um dado bloqueado) ser desfeita. Após o “rollback”, a transação pode ser reiniciada.

# Concorrência X Isolamento de transações

- nível de isolamento de transações afeta o grau de concorrência.
- padrão não exige a execução de transações de forma serializável.
- sistemas comerciais: permitem a escolha do nível de controle de concorrência desejado.

# Níveis de isolamento em SQL

- 1. serializável:** uma transação com esse nível de isolamento deve executar como se ocorresse num instante e as demais transações ocorressem completamente antes ou completamente depois (execução serial).
- 2. Leitura consolidada (read committed):** não exige serializabilidade, mas não permite que a transação leia dados “sujos”. É possível a uma transação ler o dado A duas vezes e obter valores diferentes, cada um gravado por transação consolidada.
- 3. Leitura repetível (repeatable read):** além de não permitir a leitura de dados “sujos”, se A for um relação, sucessivas leituras de A obterão apenas superconjuntos de A (nenhuma parte de A se perderá enquanto a transação estiver ativa).
- 4. Leitura não consolidada (read uncommitted):** a transação pode ver dados “sujos”.



Nível de isolamento	Dados sujos	Leitura não repetível	Consulta não repetível
Leitura não consolidada	sim	sim	sim
Leitura consolidada	não	sim	sim
Leitura repetível	não	não	sim
serializável	não	não	não

Leitura suja: T1 execute uma atualização em alguma linha. Então, T2 lê essa linha, T1 termina em seguida com um rollback. T2 enviou uma linha que não existe mais (pode-se dizer que T1 nunca foi executada).

Leitura não repetível: T1 lê uma linha, depois T2 atualiza essa linha. Então, T1 lê novamente a mesma linha. T1 leu duas vezes a “mesma” linha e obteve valores diferentes.

Fantasmas: T1 leu um conjunto de todas as linhas que satisfazem uma condição (ex. fornecedores de Paris). Em seguida, T2 insere uma nova linha que satisfaz a mesma condição. Se T1 repetir a leitura, obterá uma linha que não existia antes (Fantasma).