



**PUCRS**  
Pontifícia Universidade Católica  
do Rio Grande do Sul

**ESCOLA  
POLITÉCNICA**

# Implementação de Bancos de Dados

Aula 24

- Unidade 3 – Recuperação Após Falha

- Garantia de atomicidade e durabilidade de transações
  - Requer um SGBD tolerante a falhas
- Tolerância a falhas em BDs
  - Capacidade de conduzir o BD a um estado passado consistente, após a ocorrência de uma falha que o deixou em um estado inconsistente
  - Baseia-se em redundância de dados
  - Não existe um mecanismo 100% seguro
  - Responsabilidade do subsistema de *recovery* do SGBD

- Controles
  - durante o funcionamento normal do SGBD
    - manter informações sobre o que foi atualizado no BD pelas transações
    - realizar cópias periódicas do BD
  - após a ocorrência de uma falha
    - executar ações para retornar o BD a um estado consistente
    - ações básicas
      - UNDO: desfazer uma atualização no BD
      - REDO: refazer uma atualização no BD
- Considerações sobre o seu projeto
  - tipos de falhas a tratar
  - técnica de recovery a aplicar

- Transaction UNDO
  - uma transação não concluiu suas operações
  - as modificações realizadas por esta transação no BD são desfeitas
- Global UNDO
  - uma ou mais transações não concluíram as suas operações
  - as modificações realizadas por todas estas transações no BD são desfeitas

- Partial REDO
  - na ocorrência de uma falha, algumas transações podem ter concluído suas operações (committed), mas suas ações podem não ter se refletido no BD
  - as modificações realizadas por estas transações são refeitas no BD
- Global REDO
  - no caso de um comprometimento do BD, todas as transações committed no BD são perdidas
  - as modificações realizadas por todas estas transações no BD são refeitas, normalmente a partir de um backup

- Falha de Transação
  - uma transação ativa termina de forma anormal
  - causas
    - violação de RI, lógica da transação mal definida, *deadlock*, cancelamento pelo usuário, etc.
  - não compromete a memória principal e a memória secundária (disco, em geral).
  - falha com maior probabilidade de ocorrência
  - seu tempo de recuperação é pequeno
    - ação: Transaction UNDO

- Falha de sistema
  - o SGBD encerra a sua execução de forma anormal
  - causas
    - interrupção de energia, falha no SO, erro interno no SW do SGBD, falha de HW, ...
  - compromete a memória principal e não compromete o disco
  - falha com probabilidade média de ocorrência
  - seu tempo de recuperação é médio
    - ações: Global UNDO e Partial REDO

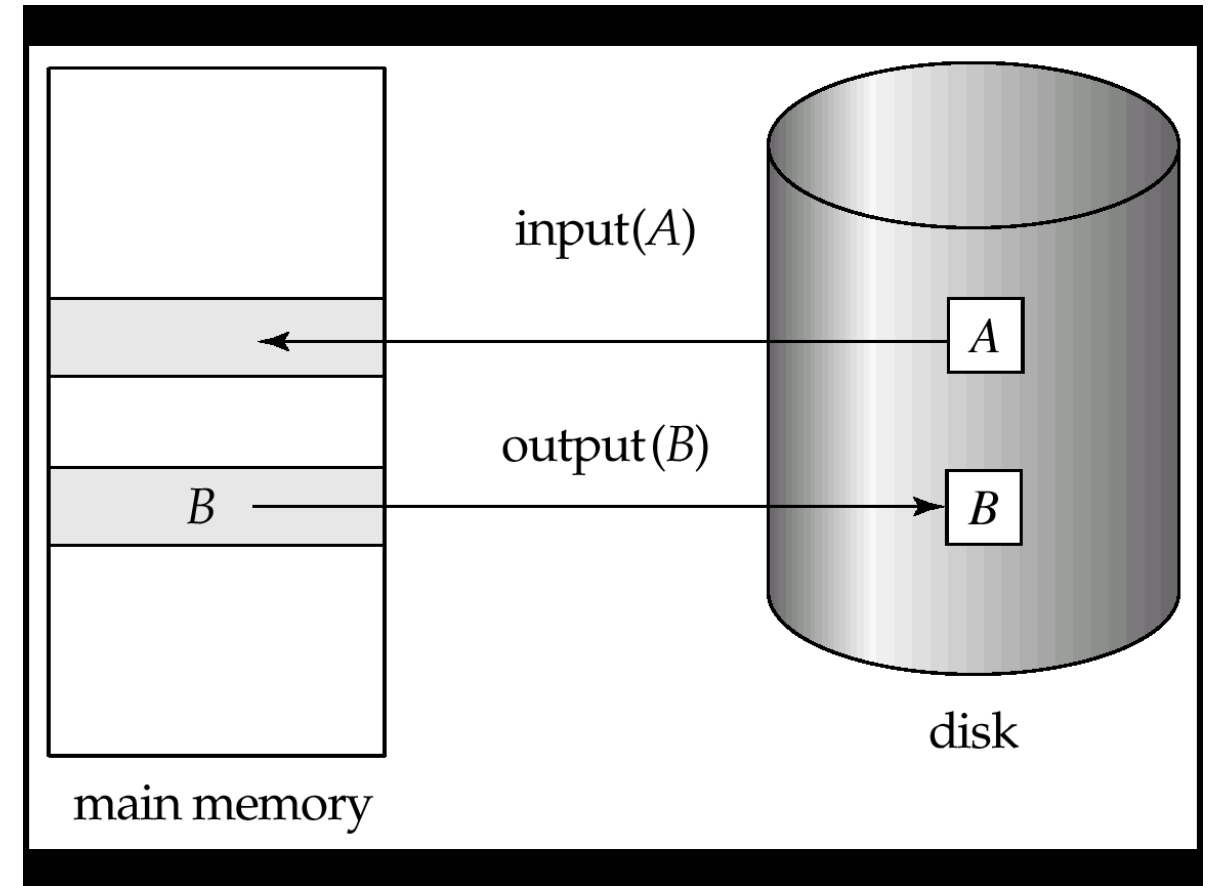
- Falha de meio de armazenamento
  - o BD torna-se total ou parcialmente inacessível
  - causas
    - setores corrompidos no disco, falhas na controladora do disco, falha no cabeçote de leitura/gravação, ...
  - não compromete a memória principal e compromete o disco
  - falha com menor probabilidade de ocorrência
  - seu tempo de recuperação é grande
    - ação: Global REDO



- Como vimos
  - Sistema de BD reside permanentemente em armazenamento não volátil
    - Geralmente discos
  - Particionado em unidades de armazenamento
    - Chamadas blocos
      - Unidades de transferência de dados para e a partir do disco
      - Pode conter vários itens de dados
  - Assuma que nenhum item abrange dois ou mais blocos

- Transações transferem informações do disco para a memória principal e reenviam estas informações de volta para o disco
- Operações de entrada e saída (da memória) são feitas em unidades de bloco
  - Blocos residentes no disco
    - Blocos físicos
  - Blocos residentes temporariamente na memória principal
    - Blocos de buffer

- Movimentos de blocos entre disco e memória principal são iniciados por meio de 2 operações
  - Input (B): do disco para a memória
    - Transfere o bloco físico B para a memória principal
  - Output (B): da memória pro disco
    - Transfere o bloco de buffer B para o disco e substitui o bloco físico apropriado no disco



- Cada transação  $T_i$  tem uma área de trabalho privada
  - Cópias de itens de dados acessados e atualizados são mantidas
- Esta área de trabalho é efetivada ou abortada
- Cada item de dados  $x$  mantido na área de trabalho da transação  $T_i$  é denotado por  $x_i$
- Transação  $T_i$  interage com o sistema de BD pela transferência de dados para e de sua área de trabalho até o buffer de sistema
- Os dados são transferidos usando duas operações:
  - Read
  - Write

- Ambas as operações podem exigir transferência de um bloco do disco para a memória principal
- Bloco de buffer será eventualmente escrito no disco
  - se o gerenciador de buffer necessitar de espaço em memória para outros propósitos
  - ou porque o sistema de BD deseja refletir a mudança em B sobre o disco
- Quando uma transação necessita de acesso a um dado x pela primeira vez
  - Executar read(x)
  - Todas as atualizações de x são realizados sobre xi
  - Após o último acesso feito pela transação
    - Executar write(x)

- Operação output(B) para o buffer de bloco Bx em que x reside não precisa ter efeito imediato após a execução do write(x)
  - Já que o bloco Bx pode conter outros dados que ainda estão sendo acessados
- Então a saída real aparecerá mais tarde
- Se o sistema cair após a operação write(x) ter sido executada
  - Mas antes do output(Bx)
    - O novo valor de x nunca será escrito no disco
      - E portanto é perdido.

- Exemplo
  - Transferência de 50 reais da conta A para B
  - Valores iniciais
    - A=1000
    - B=2000
  - Suponha uma queda de sistema durante a execução de Ti
    - Após o output (BA)
      - Bloco de buffer em que A reside
    - Mas antes do output (BB)
      - Bloco de buffer em que B reside
  - Conteúdos da memória foram perdidos.
  - Dois possíveis procedimentos de recuperação:

- Reexecutar Ti
  - Faz com que o valor de A torne-se 900, em vez de 950.
  - Sistema entra num estado inconsistente.
- Não reexecutar Ti
  - No estado corrente, A e B são 950 e 2000.
  - Sistema entra em estado inconsistente.
- Logo
  - Este esquema de recuperação simples não funciona:
    - Precisamos garantir que todas ou nenhuma das modificações feitas por Ti persista no banco.



- Baseadas em Log
  - modificação adiada do BD
    - técnica NO-UNDO/REDO
  - modificação imediata do BD
    - técnica UNDO/REDO
    - técnica UNDO/NO-REDO
  - recuperação de meio de armazenamento
    - técnica ARCHIVE/DUMP/REDO
- Baseadas em Shadow Pages
  - técnica NO-UNDO/NO-REDO

recuperação de  
falhas de transação  
e de sistema

recuperação de  
falhas de transação  
e de sistema

- Técnicas mais comuns de recovery
- Utilizam um (ou vários) arquivo de Log
  - registra seqüencialmente as atualizações feitas por transações no BD
    - é consultado em caso de falhas para a realização de UNDO e/ou REDO de transações
  - mantido em uma ou mais cópias em memória secundária (disco, fita, ...)
  - tipos de log
    - log de UNDO
      - mantém apenas o valor antigo do dado (before image)
    - log de REDO
      - mantém apenas o valor atualizado do dado (after image)
    - log de UNDO/REDO (mais comum)
      - mantém os valores antigo e atualizado do dado

- Supõe-se que toda transação possui um identificador único gerado pelo SGBD
- Para fins de recuperação de falhas, operações *read* não precisam ser gravadas
  - úteis apenas para outros fins (auditoria, estatísticas, ...)
- Principais tipos de registro
  - **início de transação**: `<start Tx>`
  - **commit de transação**: `<commit Tx>`
  - **atualização**: `<write Tx, X, beforeImage, afterImage>`

não é necessário em técnica REDO/NO-UNDO ←

não é necessário em técnica NO-REDO/UNDO ←

## Log

```
<start T3>  
<write T3, B, 15, 12>  
<start T2>  
<write T2, B, 12, 18>  
<start T1>  
<write T1, D, 20, 25>  
<commit T1>  
<write T2, D, 25, 26>  
<write T3, A, 10, 19>  
<commit T3>  
<commit T2>  
...
```

T<sub>1</sub>

```
read(A)  
read(D)  
write(D)
```

T<sub>2</sub>

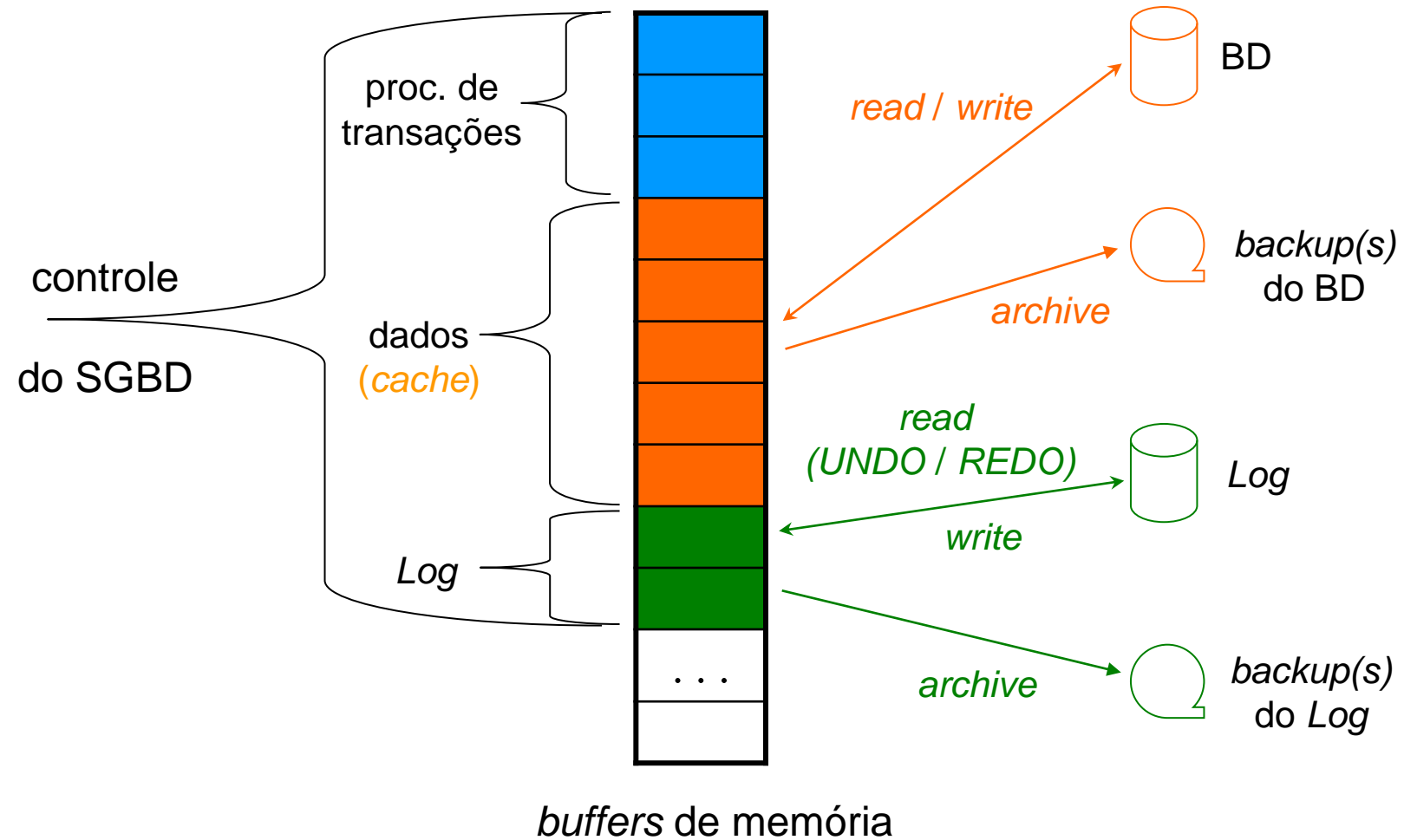
```
read(B)  
write(B)  
read(D)  
write(D)
```

T<sub>3</sub>

```
read(C)  
write(B)  
read(A)  
write(A)
```

- Forma alternativa de representar atualizações
  - considera a operação DML feita no BD
    - **insert**: `<write Tx, X, INSERT, afterImage>`
    - **update**: `<write Tx, X, UPDATE, beforeImage, afterImage>`
    - **delete**: `<write Tx, X, DELETE, beforeImage>`
- A indicação do tipo de operação facilita o entendimento do que deve ser desfeito (undo) ou refeito (redo) no BD

- *Buffer*
  - conjunto de blocos da memória principal
- O SGBD é responsável pela gerência de dos *buffers*:
  - *buffers* para dados, para processamento de transações e para o *Log*.
  - ele assume o controle desses *buffers*, ao invés do SO, requisitando apenas serviços de leitura/escrita de blocos ao SO.



- Técnicas de recovery devem sincronizar os buffers de log e de dados
  - princípio básico
    - um bloco atualizado no buffer só pode ser gravado no BD após o histórico dos dados atualizados neste bloco ter sido gravado no Log em disco
      - Write-Ahead-Log (WAL)
  - uma transação Tx só pode passar para o estado de efetivada (committed) após todas as suas atualizações terem sido gravadas no Log segundo o princípio WAL.
- O SGBD aplica técnicas de gerenciamento de buffer
  - estas técnicas influenciam as técnicas de recovery



- NOT-STEAL
  - um bloco na cache utilizado por uma transação Tx não pode ser gravado antes do commit de Tx
    - bloco possui um bit de status indicando se foi (1) ou não (0) modificado
    - vantagem: processo de recovery mais simples - evita dados de transações inacabadas sendo gravadas no BD, não precisa de UNDO.
- STEAL
  - um bloco na cache utilizado por uma transação Tx pode ser gravado antes do commit de Tx
    - necessário se algum dado é requisitado do BD por outra transação e não há blocos disponíveis na cache
    - o bloco “vítima” é escolhido através de alguma técnica de SO
      - LRU, FIFO, Clock, ...
    - vantagem: não há necessidade de manter blocos bloqueados por transações

- **FORCE**
  - os blocos que mantêm dados atualizados por uma transação Tx são imediatamente gravados no BD quando Tx alcança o commit
    - deve-se saber quais os blocos em que Tx atualizou dados
  - vantagem: garante a durabilidade de Tx o mais cedo possível – não necessita REDO de Tx em caso de falha
- **NOT-FORCE**
  - os blocos que mantêm dados atualizados por Tx não são imediatamente gravados no BD quando Tx alcança o commit
  - vantagem: blocos atualizados podem permanecer na cache e serem utilizados por outras transações, após o commit de Tx (reduz custo de acesso a disco)

- Baseadas em Log
  - modificação adiada do BD
    - técnica NO-UNDO/REDO
  - modificação imediata do BD
    - técnica UNDO/REDO
    - técnica UNDO/NO-REDO
  - recuperação de meio de armazenamento
    - técnica ARCHIVE/DUMP/REDO
- Baseadas em Shadow Pages
  - técnica NO-UNDO/NO-REDO

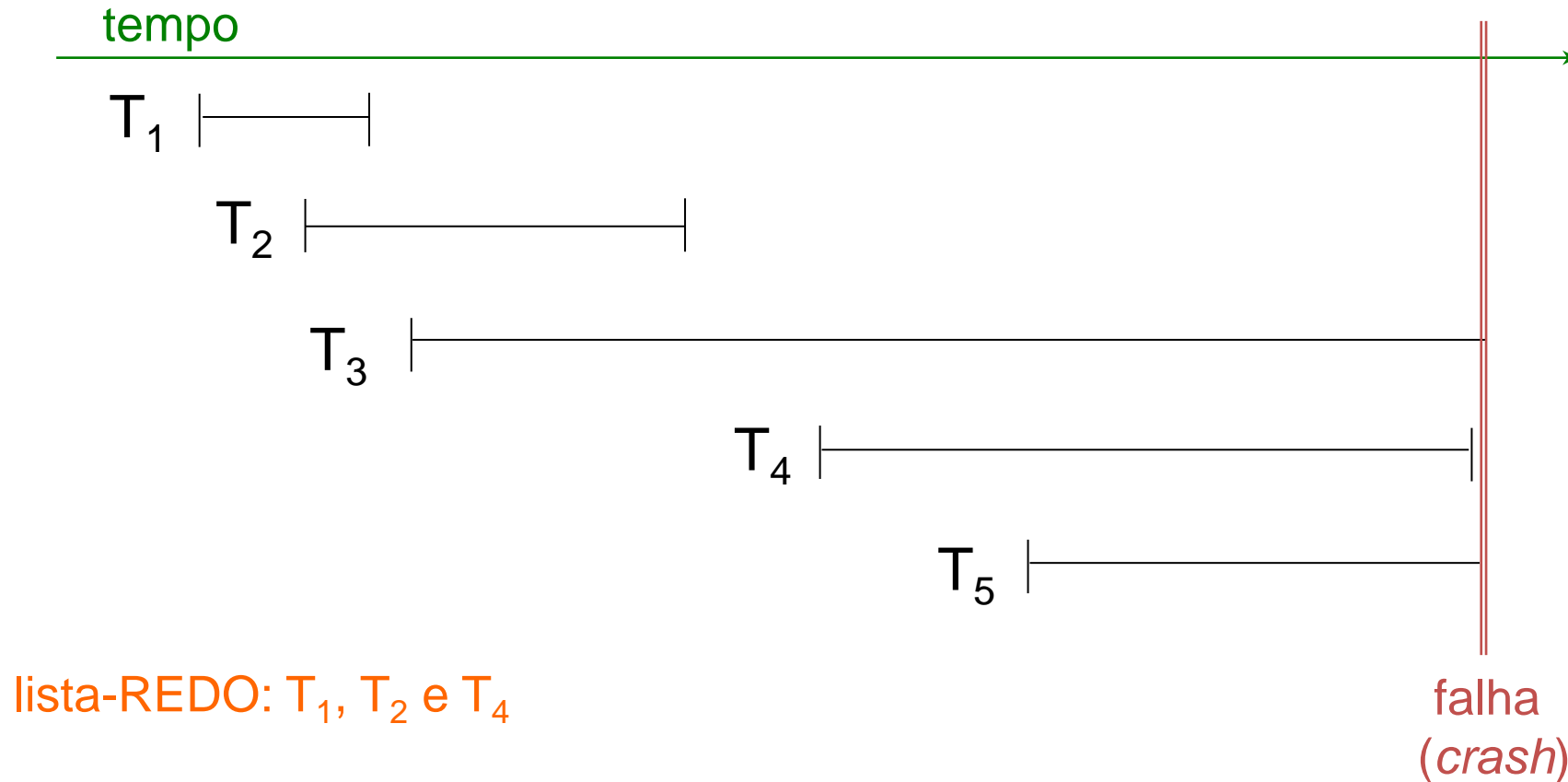
recuperação de  
falhas de transação  
e de sistema

recuperação de  
falhas de transação  
e de sistema

- Abordagem na qual dados atualizados por uma transação Tx não podem ser gravados no BD antes do commit de Tx
- Gerenciamento de buffer mais complexo
  - utiliza técnica NOT-STEAL
    - blocos atualizados por Tx não podem ser “roubados” enquanto Tx não realizar commit.
  - por outro lado, o recovery é mais simples
    - transações não precisam sofrer UNDO.
- Técnica
  - NO-UNDO/REDO

- A técnica de adiar modificações garante atomicidade quando todas as modificações do BD são escritas no log
  - Adiando a execução de todas as operações de escrita de uma transação até sua efetivação
- Uma transação é considerada parcialmente efetivada quando a última ação tiver sido executada
  - Informações no log associadas àquela transação são usadas para a execução das escritas adiadas.
  - Se o sistema cair antes de completar a transação ou se a transação for abortada
    - Informações do log serão ignoradas.

- Quando Tx conclui suas atualizações, força-se a gravação do Log em disco (com <commit Tx> )
  - FORCE no Log
- Vantagem
  - se Tx falha antes de alcançar o commit, não é necessário realizar UNDO de Tx
    - nenhuma atualização de Tx foi gravada no BD
    - requer apenas um Log de REDO
- Desvantagem
  - overhead no tempo de processamento (NOT-STEAL)
    - um bloco da cache pode permanecer em memória por muito tempo
      - dependente do commit de uma ou mais transações que atualizaram dados nele
      - se a cache fica cheia, é possível que algumas transações requisitando dados do BD tenham que esperar pela liberação de blocos



- $T_1$  e  $T_2$  concluíram e atualizaram o BD  $\Rightarrow$  sofrem REDO
- $T_4$  concluiu, mas não chegou a atualizar o BD  $\Rightarrow$  sofre REDO
- $T_3$  e  $T_5$  não concluíram e portanto não atualizaram o BD  $\Rightarrow$  não sofrem UNDO

- Execução de Ti funciona como segue
  - Registro <Ti start> é escrito no log antes de Ti ter início.
  - Write(x) feita por Ti resulta na escrita de um novo registro no log.
  - Quando Ti é parcialmente efetivada, um registro <Ti commit> é escrito no log.
- Quando Ti é parcialmente efetivada
  - Registros no log são usados para execução das escritas adiadas.
  - Como falhas podem ocorrer durante este passo
    - Escrever os registros de log em armazenamento estável.
  - Uma vez escritas
    - Atualizações reais podem ocorrer de fato e a transação entra no estado de efetivação.
- Somente o novo valor do dado é necessário para a técnica de modificação adiada
  - Omitir o campo valor antigo no registro de log.



- Exemplo

*T0:*  
**read** (*A*)  
*A*:- *A* - 50  
**Write** (*A*)  
**read** (*B*)  
*B*:- *B* + 50  
**write** (*B*)

*T1:*  
**read** (*C*)  
*C*:-*C*- 100  
**write** (*C*)

- Suponha que executem serialmente

- T0 é seguida por T1
- Inicialmente A, B e C=1000, 2000 e 700
- Log correspondente:

<T0 start>  
<T0, A, 950>  
<T0, B, 2050>  
<T0 commit>  
<T1 start>  
<T1, C, 600>  
<T1 commit>

- Como resultado da execução de T0 e T1
  - Há várias ordens possíveis em que as saídas reais podem ocorrer
    - Tanto para o sistema de BD qto para o log
- Ordem

## Log

<T0 start>  
<T0, A, 950>  
<T0, B, 2050>  
<T0 commit>

<T1 start>  
<T1, C, 600>  
<T1 commit>

## BDados

A=950  
B=2050

C=600

Note que A é alterado  
somente após o registro  
<T0, A, 950> ter sido  
colocado no log

- Usando o log
  - O sistema pode lidar com qualquer falha que resulte em perda de informação no armazenamento volátil
- Procedimento usado pelo esquema de recuperação
  - Redo(Ti)
    - Define o valor de todos os itens de dados atualizados por Ti para os novos valores
    - Conjunto de itens de dados atualizados por Ti e seus respectivos novos valores podem ser encontrados no log
    - Operação redo (refazer)
      - Executá-la várias vezes deve ser equivalente a executá-la uma vez só
        - Característica exigida para garantir comportamento correto

- Após a ocorrência de falha
  - O sistema de recuperação consulta o log
    - Determinar quais operações têm de ser refeitas
  - Transação Ti deverá ser **refeita** se e somente se o log contiver os registros **<Ti start>** e **<Ti commit>**
    - Se o sistema cair depois que a transação completar sua execução
      - Informações no log serão usadas na restauração do sistema para o estado consistente anterior

- Log resultante da execução completa de T0 e T1:
- Suponha q o sistema caia antes que as transações terminem
  - Logo após o registro do log write(B) ter sido escrito em armazenamento estável
  - Log, no momento da queda, é:
- Quando o sistema retorna
  - Nenhuma ação refazer tem de ser tomada
    - Já que nenhum registro de efetivação aparece no log
  - A e B continuam com 1000 e 2000
  - Registros de log da transação incompleta T0 podem ser removidos do log

<T0 start>  
<T0, A, 950>  
<T0, B, 2050>

<T0 start>  
<T0, A, 950>  
<T0, B, 2050>  
<T0 commit>  
<T1 start>  
<T1, C, 600>  
<T1 commit>

- Agora assuma que a queda venha logo após o registro de log para o passo write(C) ter sido escrito em armazenamento estável
  - O log, no momento da queda, está como:
    - <T0 start>
    - <T0, A, 950>
    - <T0, B, 2050>
    - <T0 commit>
    - <T1 start>
    - <T1, C, 600>
- Quando o sistema retorna
  - Redo(T0) é realizada
    - Já que o registro <T0, commit> aparece no log em disco
  - Após esta operação
    - A=950, B=2050 e C=700
    - Registros de log da transação T1 incompleta podem ser removidos do log

- Finalmente, assuma que a queda venha logo após o registro de log *<T1 commit>* ter sido escrito em armazenamento estável
  - O log, no momento da queda, está como:
    - <T0 start>
    - <T0, A, 950>
    - <T0, B, 2050>
    - <T0 commit>
    - <T1 start>
    - <T1, C, 600>
    - <T1 commit>
- Quando o sistema retorna
  - 2 registros de efetivação estão no log
    - T0 e T1
  - Operações **redo(T0)** e **redo(T1)** devem ser processadas
  - Após estas operações
    - A=950, B=2050 e C=600

- Baseadas em Log
  - modificação adiada do BD
    - técnica NO-UNDO/REDO
  - modificação imediata do BD
    - técnica UNDO/REDO
    - técnica UNDO/NO-REDO
  - recuperação de meio de armazenamento
    - técnica ARCHIVE/DUMP/REDO
- Baseadas em Shadow Pages
  - técnica NO-UNDO/NO-REDO

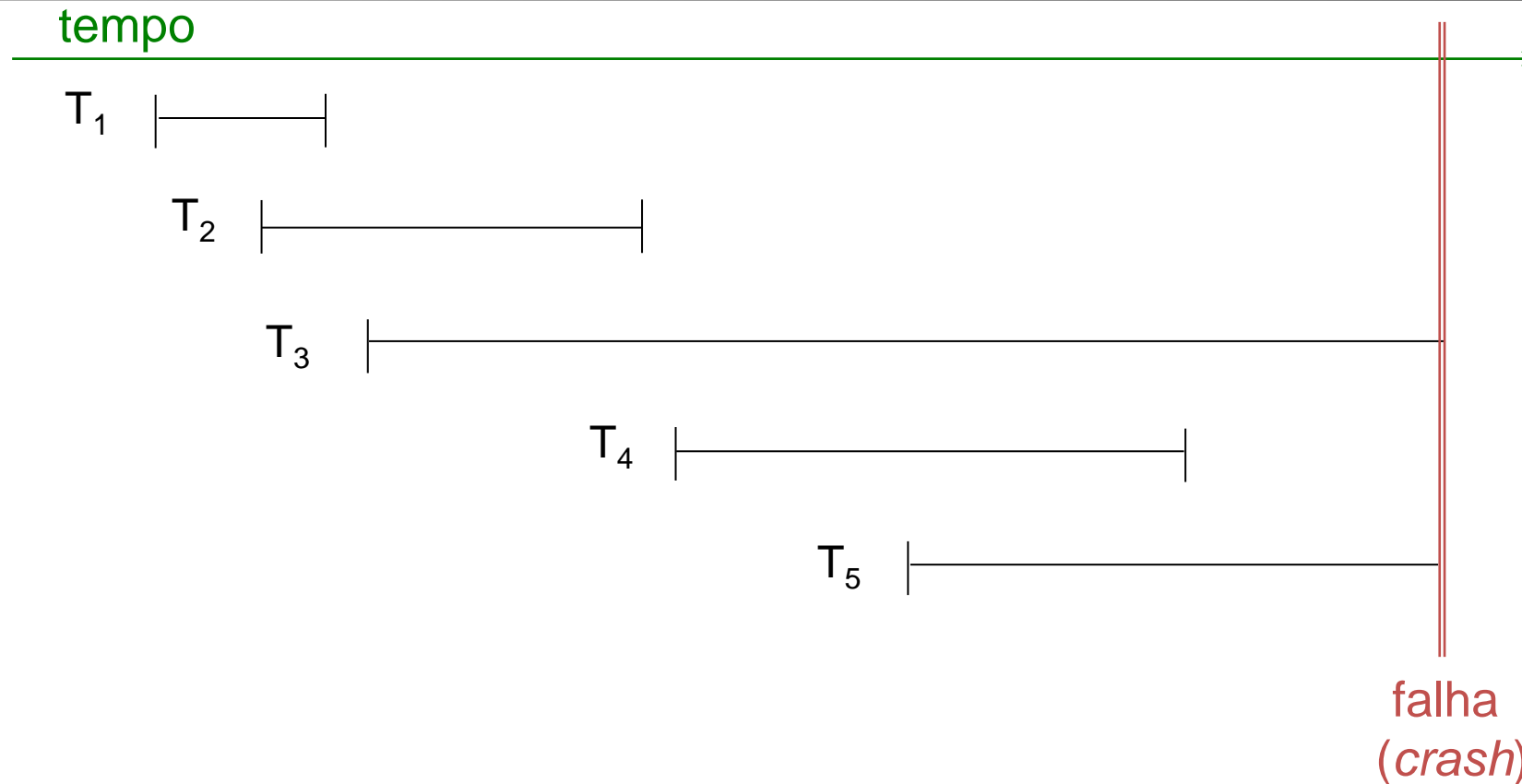
recuperação de  
falhas de transação  
e de sistema

recuperação de  
falhas de transação  
e de sistema



- Abordagem na qual dados atualizados por uma transação Tx podem ser gravados no BD antes do commit de Tx
- Abordagem mais comum de recovery
  - gerenciamento de buffer mais simples
    - utiliza técnica STEAL
- Duas técnicas
  - UNDO/REDO
    - técnica mais comum de recovery
  - UNDO/NO-REDO

- Grava o commit de Tx no Log depois de todas as atualizações de Tx terem sido gravadas no Log, e antes dessas atualizações serem gravadas no BD
  - requer um Log de UNDO/REDO
- Utiliza 2 listas de transações
  - lista-REDO: IDs de transações committed
    - possuem commit gravado no Log
  - lista-UNDO: IDs de transações ativas
- Procedimento
  - faz uma varredura backward do Log, realizando UNDO das transações na lista-UNDO
  - faz uma varredura forward do Log, realizando REDO das transações na lista-REDO



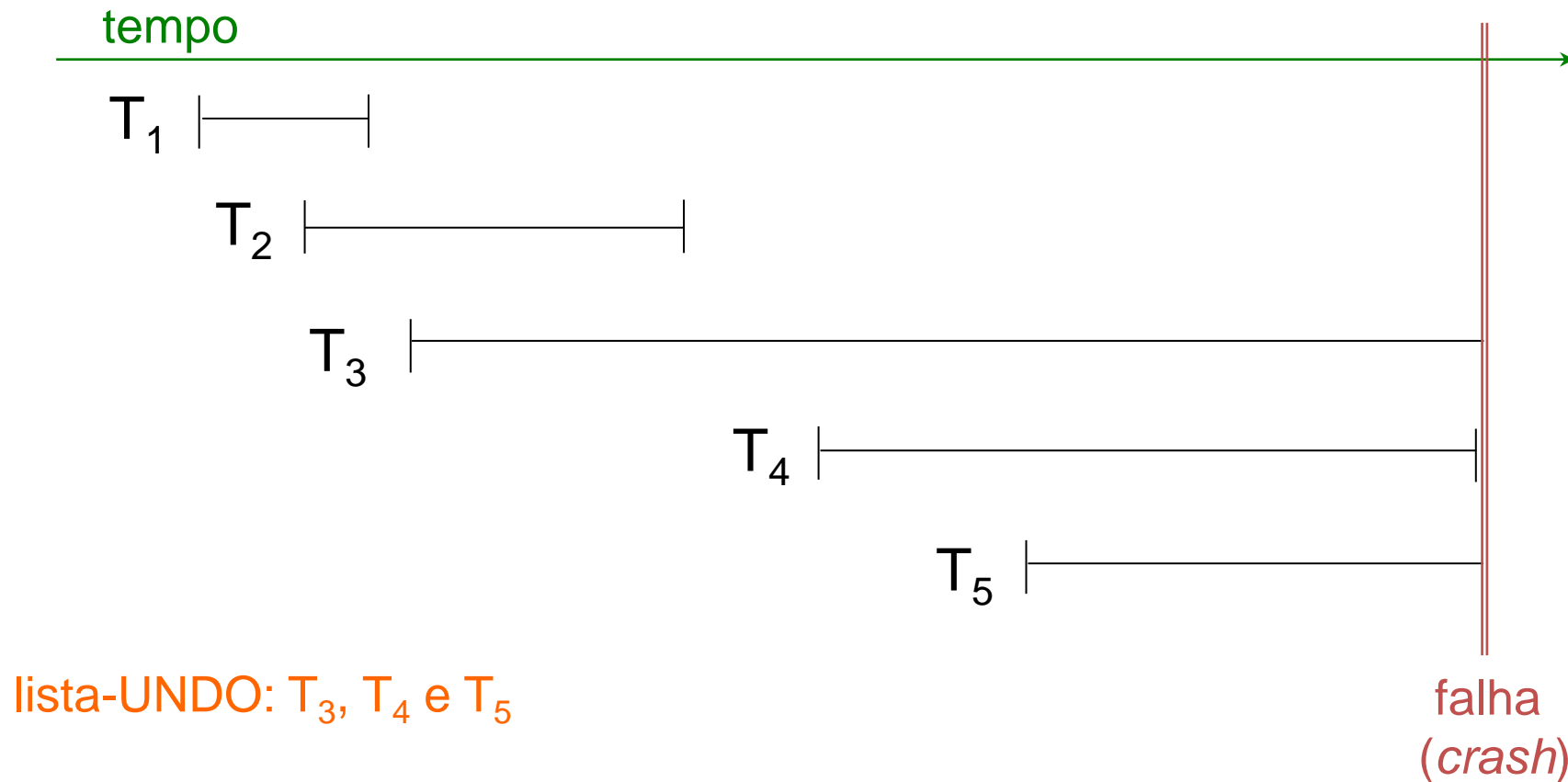
lista-UNDO:  $T_3$ ,  $T_5$  (devem sofrer UNDO)

lista-REDO:  $T_1$ ,  $T_2$ ,  $T_4$  (devem sofrer REDO)

- A propriedade de idempotência de operações UNDO e REDO é válida
  - fazer UNDO ou REDO uma vez ou várias vezes produz o mesmo resultado
    - situações em que ocorrem falhas durante o processo de recovery
- Técnica mais trabalhosa de recovery
  - tanto UNDO quanto REDO devem ser realizados
    - porém, o gerenciamento de buffer é mais simples

- Quando se percorre o Log forward para fazer REDO, é possível que um dado X tenha sido atualizado por mais de uma transação committed
- Variante da técnica UNDO/REDO
  - detectar que X é atualizado mais de uma vez e realizar apenas a última atualização
    - técnica UNDO/REDO com REDO único para cada dado
  - estratégia
    - na varredura backward do Log para fazer UNDO, quando for encontrada a primeira atualização de um dado X por uma transação committed, inclui-se X e sua afterImage na lista-REDO-dados
      - novas atualizações de X feitas por transações committed que forem encontradas são ignoradas
    - após, varre-se a lista-REDO-dados, atualizando os dados

- Outra técnica de modificação imediata do BD
- Grava o commit de Tx no Log depois de todas as atualizações de Tx terem sido gravadas no Log, e depois delas terem sido gravadas no BD
  - assim, se <commit Tx> está no Log, Tx está garantidamente efetivada no BD
    - vantagem: não há necessidade de fazer REDO
    - desvantagem: pode-se fazer UNDO de uma transação que foi gravada com sucesso no BD, porém não foi gravado a tempo o seu commit no Log
- Requer um Log de UNDO
- Procedimento
  - faz uma varredura backward do Log, realizando UNDO das transações na lista-UNDO (transações ativas)



- $T_1$  e  $T_2$  concluíram e tem *commit* no *Log*  $\Rightarrow$  não sofrem REDO
- $T_4$  concluiu, mas não tem *commit* no *Log*  $\Rightarrow$  sofre UNDO
- $T_3$  e  $T_5$  não concluíram  $\Rightarrow$  sofrem UNDO

- Baseadas em Log
  - modificação adiada do BD
    - técnica NO-UNDO/REDO
  - modificação imediata do BD
    - técnica UNDO/REDO
    - técnica UNDO/NO-REDO
  - recuperação de meio de armazenamento
    - técnica ARCHIVE/DUMP/REDO
- Baseadas em Shadow Pages
  - técnica NO-UNDO/NO-REDO

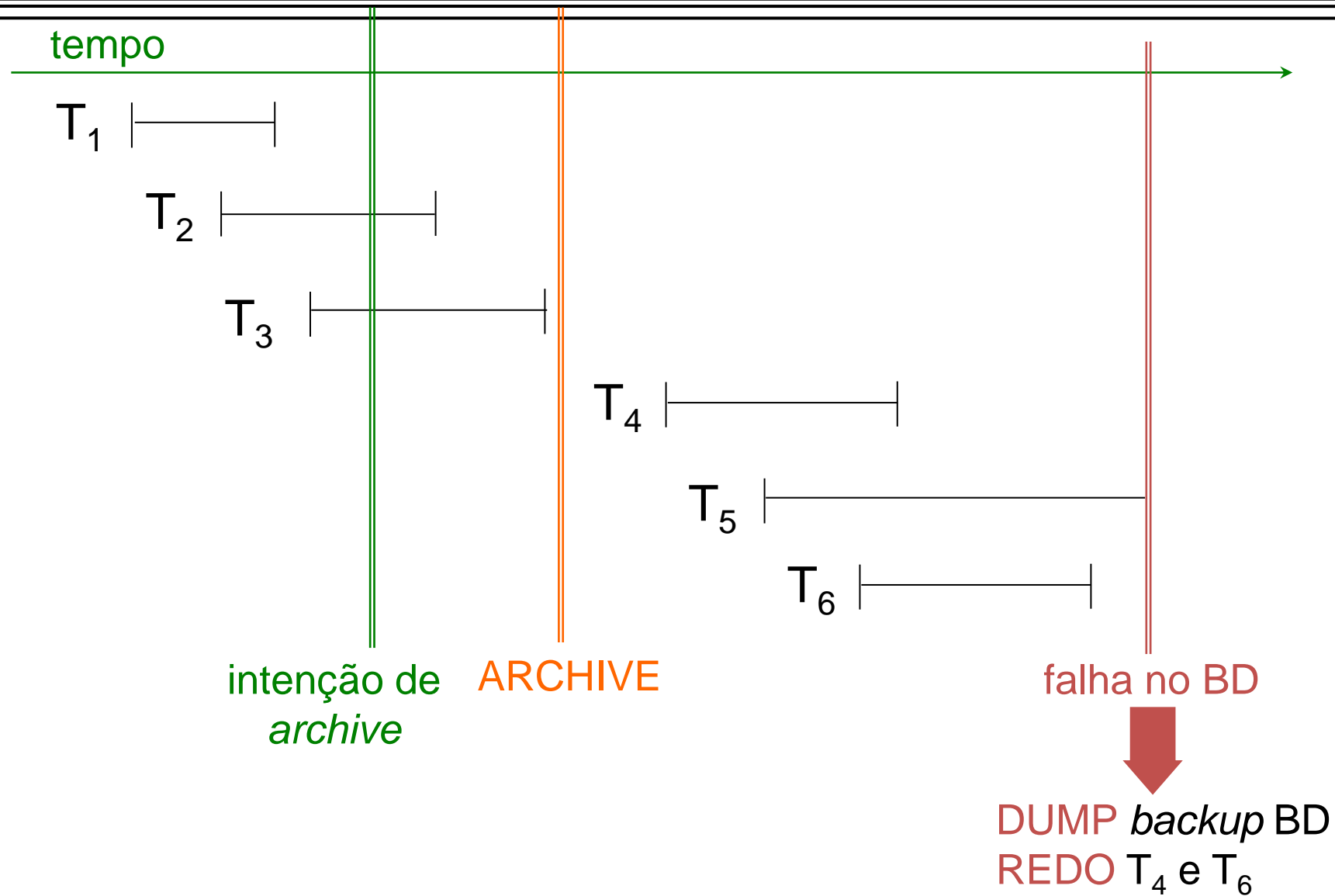
recuperação de  
falhas de transação  
e de sistema

recuperação de  
falhas de transação  
e de sistema




- Técnica baseada em Log para recuperação de falha de meio de armazenamento
- Operação ARCHIVE
  - ocorre durante o funcionamento normal do SGBD
  - gravação de uma ou mais cópias backup do BD em dispositivos diferentes de memória secundária
  - disparo manual ou automático (periódico)
  - deve-se suspender o início de novas transações
    - espera transações terminarem e não inicia novas (cold backup)
  - ou nenhuma transação pode gravar dados no disco
    - manter as atualizações somente em buffer e no novo log (hot backup).
  - o Log corrente é “descartado” (excluído ou mantido associado ao backup anterior do BD) e um novo Log (“zerado”) é iniciado

- Operações DUMP + REDO
  - realizam o recovery de uma falha no BD
  - procedimento
    - restaura o BD a partir do último backup (DUMP)
    - realiza uma varredura forward do Log, realizando REDO das transações committed
    - as transações ativas no momento da falha podem ser re-submetidas à execução pelo SGBD
      - já que não houve perda de dados na memória principal
- Técnicas baseadas em Log requerem um Log seguro
  - archive do Log também deve ser realizado
    - com frequência igual ou superior ao archive do BD



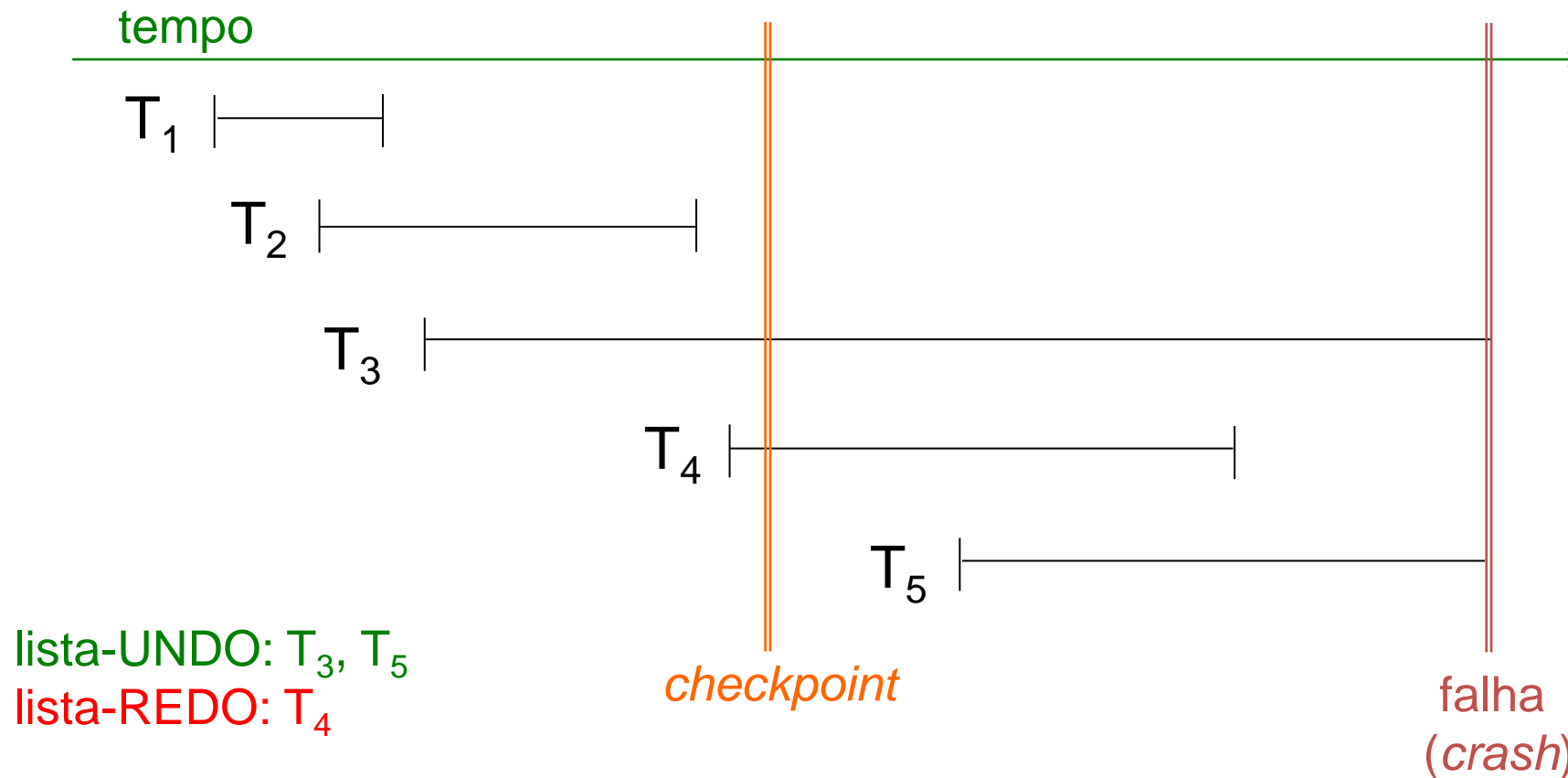
- SGBD com alta demanda de transações
  - Log de tamanho grande
    - recovery demorado
- Checkpoint
  - momento em que o SGBD grava no BD todas as atualizações feitas por transações
  - disparo manual ou automático
  - inclusão de um registro de checkpoint no Log

`<checkpoint T1, T2, ..., Tn>`



lista de transações ativas

- Procedimento de execução de checkpoint
  - suspensão de todas as transações
  - descarga do buffer de Log em disco
    - FORCE do Log
  - gravação dos blocos atualizados da cache no BD
  - inserção de um registro checkpoint no Log e sua gravação em disco
  - retomada da execução das transações
- Vantagem da técnica de checkpoint
  - transações committed antes do checkpoint não precisam sofrer REDO em caso de falha
    - elas já estão garantidamente no BD



- $T_1$  e  $T_2$  concluíram e estão garantidamente no BD  $\Rightarrow$  não sofrem REDO
- $T_4$  concluiu, mas suas atualizações não necessariamente estão no BD (supondo NOT-FORCE)  $\Rightarrow$  sofre REDO
- $T_3$  e  $T_5$  não concluíram  $\Rightarrow$  sofrem UNDO

- Procedimento de recovery
  - percorre-se o Log backward até alcançar um registro Checkpoint
    - se achou <commit Tx>, insere Tx na lista-REDO
    - se achou <start Tx> e Tx não está na lista-REDO, insere Tx na lista-UNDO
  - analisa-se cada transação Tx no registro checkpoint
    - se Tx não estiver na lista-REDO, insere Tx na lista-UNDO

- Procedimento de recovery (cont.)
  - percorre-se de novo o Log backward, até que todas as transações em lista-UNDO tenham sofrido UNDO
    - marca-se na lista-REDO as transações Tx cujos registros <start Tx> estão sendo encontrados nessa varredura
  - se existem transações não marcadas na lista-REDO ao final da varredura backward
    - continua-se a varredura backward até que todas as transações na lista-REDO tenham sido marcadas
  - percorre-se o Log forward do ponto de parada, realizado REDO das transações na lista-REDO
- Vantagem
  - não é necessário varrer sempre todo o Log

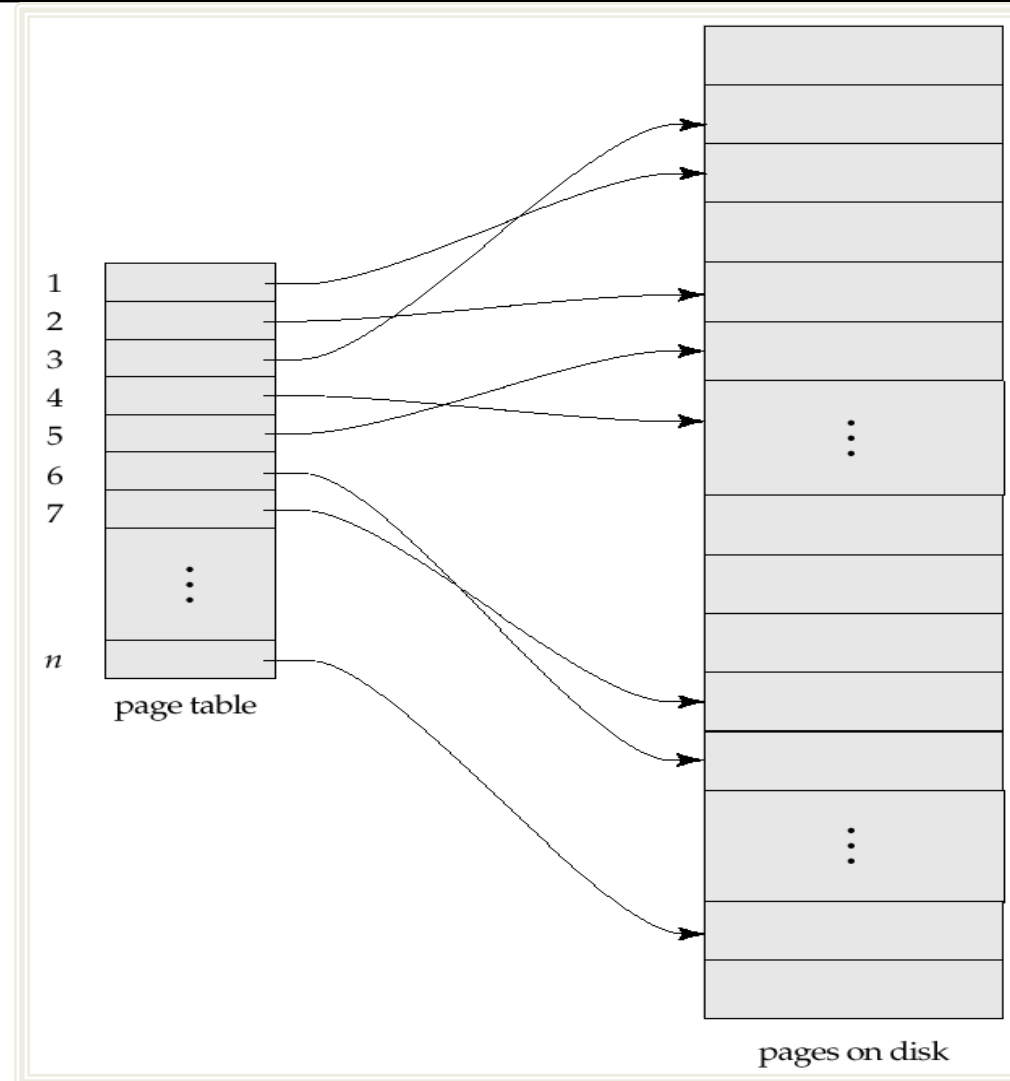


- Compare as versões de modificação adiada e imediata dos esquemas de recuperação baseados em log, em termos de facilidade de implementação e custo de overhead
- Relacione as técnicas de gerência de buffer com a técnica de Recovery a ser utilizada. Lembre que a técnica para buffers de dados e buffers de log pode ser diferente.

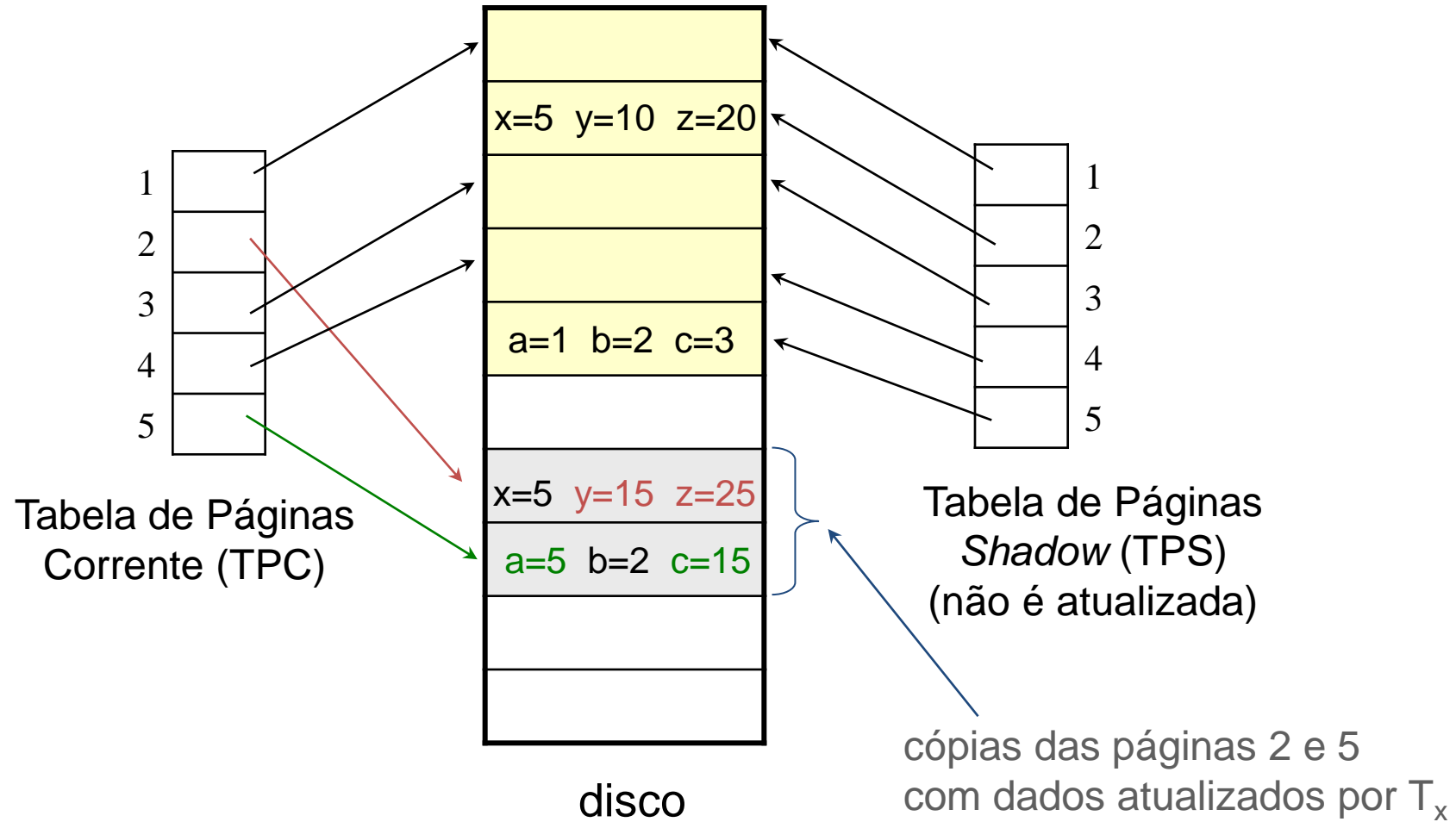
- Assuma que a modificação imediata seja usada em um sistema. Mostre, por meio de um exemplo, como poderia ocorrer um estado inconsistente do BD se os registros de log para uma transação não forem enviados para saída em armazenamento estável antes da efetivação daquela transação.
- Explique o propósito do mecanismo de checkpoint. Quantas vezes os checkpoints devem ser criados? Como essa frequência afeta:
  - O desempenho do sistema qdo nenhuma falha ocorre
  - O tempo necessário para a recuperação de uma queda de sistema
  - O tempo necessário para a recuperação de uma falha de disco

- Alternativa às técnicas de recuperação baseadas em log
- Idéia básica
  - Manter 2 tabelas de página durante o processamento da transação
    - Tab de páginas atuais
    - Tab de páginas shadow
  - Qdo a transação começa
    - Ambas são iguais
  - Tab de páginas shadow não é alterada durante a duração da transação
  - Tab de páginas atuais é alterada qdo transação processa write
  - Todas as operações de entrada (input) e saída (output) usam a tab de páginas atuais para localizar páginas do BD no disco

- BD é particionado em um número de blocos de comprimento fixo
  - Páginas:  $n$  páginas: numeradas de 1 até  $n$
  - Para encontrar as páginas no disco
    - Tabela de páginas
      - $n$  entradas
      - 1 para cada página do BD
      - Cada entrada tem um ponteiro para 1 pág no disco
      - Ordem lógica não precisa corresponder à ordem física



- Supõe a existência de uma tabela de blocos (páginas) de disco que mantém dados do BD
  - Tabela de Páginas Corrente (TPC)
- A TPC é copiada para uma Tabela de Páginas Shadow (TPS) a cada nova transação Tx
  - páginas atualizadas por Tx são copiadas para novas páginas de disco e TPC é atualizada
  - TPS não é atualizada enquanto Tx está ativa
- Em caso de falha de Tx, TPC é descartada e TPS torna-se a TPC
  - não é preciso acessar o BD para realizar restaurações
- Técnica
  - NO-UNDO/NO-REDO (com FORCE)



- Quando uma transação Tx inicia
  - $TPS \leftarrow TPC$
  - FORCE TPS
- Quando Tx atualiza dados de uma página P
  - se é a primeira atualização de Tx em P
    - se P não está na cache então busca P no disco
    - busca-se uma página livre P' na Tabela de Páginas Livres (TPL)
    - $P' \leftarrow P$  (grava nessa página livre em disco)
    - apontador de P na TPC agora aponta para P'
  - atualiza-se os dados em P

- Quando Tx solicita commit
  - FORCE das páginas P1, ..., Pn atualizadas por Tx que ainda não foram para disco
    - com FORCE da TPC atualizada primeiro
    - lembre-se que P1, ..., Pn estão sendo gravadas em páginas diferentes no disco
- Falha antes ou durante o passo 3
  - não é preciso UNDO pois TPS mantém as páginas do BD consistentes antes de Tx
  - faz-se  $TPC \leftarrow TPS$
- Falha após o passo 3
  - não é preciso REDO pois as atualizações de Tx estão garantidamente no BD



- Sobrecarga relativa ao envio dos registros de log é eliminado
- Recuperação de falhas é mais rápida
  - Não necessita undo nem redo

- Adequada a SGBD monousuário
  - uma transação executando por vez
  - SGBD multiusuário
    - gerenciamento muito complexo
    - Bloqueio de páginas
- Não mantém dados do BD clusterizados
- Requer coleta de lixo
  - quando Tx encerra, existem páginas obsoletas
    - páginas obsoletas devem ser incluídas na TPL
- Na prática não é usado

