unioeste
Universidade Estadual do Oeste do Paraná

Módulo III: Técnicas de Recuperação

Clodis Boscarioli

Informática
UNIOESTE - CAMPUS CASCAVEL

Agenda:

- Introdução;
- Conceitos de recuperação em BD;
- Técnicas de Recuperação:
  - Com atualização adiada;
  - Com modificação imediata.
- Recuperação com transações concorrentes;
- Paginação *Shadow.*

Introdução

- A recuperação de falhas existe para garantir as propriedades de *atomicidade* e *durabilidade* de transações.
- O sistema de recuperação(restauração) de falhas é responsável pela restauração do banco de dados para um estado – o que havia antes da ocorrência de uma falha.

Introdução - Revisão

- Os tipos de falhas tratáveis por um sistema de recuperação de falhas:
  - Falha de transação → SGBD entra em ação;
    - Erro lógico: a transação não pode mais continuar devido a alguma condição adversa interna;
    - Erro do sistema: uma transação não pode mais continuar porque o sistema entrou num estado inadequado.
  - Queda de sistema: perda de conteúdo volátil → continua-se com a base em meio não volátil;
  - Falha de disco: perda - parcial ou total - do conteúdo não volátil → sistemas de *backup.*

Introdução

Os algoritmos de recuperação de falhas possuem duas partes:

- Ações tomadas durante o processamento normal da transação a fim de garantir informações suficientes para permitir a recuperação de falhas;
- Ações tomadas em seguida à falha, recuperando o conteúdo do banco de dados para um estado que assegure sua consistência e a atomicidade e durabilidade das transações.

Introdução - Revisão

Meios de armazenamento:

- Armazenamento volátil: as informações neste meio, geralmente, não resistem à quedas de sistema.
- Armazenamento não-volátil: as informações neste meio sobrevivem a quedas do sistema; No entanto, estão sujeitas a se perderem em falhas mais drásticas.
- Armazenamento estável: a informação neste meio *nunca* é perdida. Teoricamente é impossível de ser obtido, mas pode-se chegar perto deste meio pelo uso de técnicas para tornar improvável a perda de dados.

Introdução

- Implementação de um meio estável:
  - Replicar a informação em vários meios de armazenamento não-volátil independentes, controlando a atualização das informações;
  - Sistemas RAID: coleções de formas de organização de discos;
  - Armazenar *backups* em fitas colocadas em diferentes locais, para proteção contra desastres;
  - Manter cada bloco de armazenamento em sites remotos (transferência de blocos via rede).

Introdução

- A transferência de dados entre memória e disco pode resultar em:
  - Conclusão bem sucedida: o destino recebeu a informação;
  - Falha parcial: o destino recebeu informação incorreta;
  - Falha total: o destino permaneceu intacto;
- É interessante manter dois blocos físicos (destino) para cada bloco lógico (origem);
- Um bloco contém diversos dados.

Obs.: Assumiremos que um dado não pertence a mais de um bloco.

Introdução

- Movimentos de blocos entre memória principal e disco envolvem duas operações:
  - **Input(*B*):** transfere o bloco físico *B* para a memória principal;
  - **Output(*B*):** transfere o bloco de *buffer B* para o disco.

Introdução

- Cada transação possui uma área de trabalho privada na qual mantém cópia de todos os itens de dados acessados por ela.
- Essa área de trabalho é criada quando a transação é iniciada; e é removida quando a transação é abortada ou efetivada.
- A interação da transação com o sistema se dá por meio da transferência de dados desta área de trabalho para o *buffer* do sistema e vice-versa.

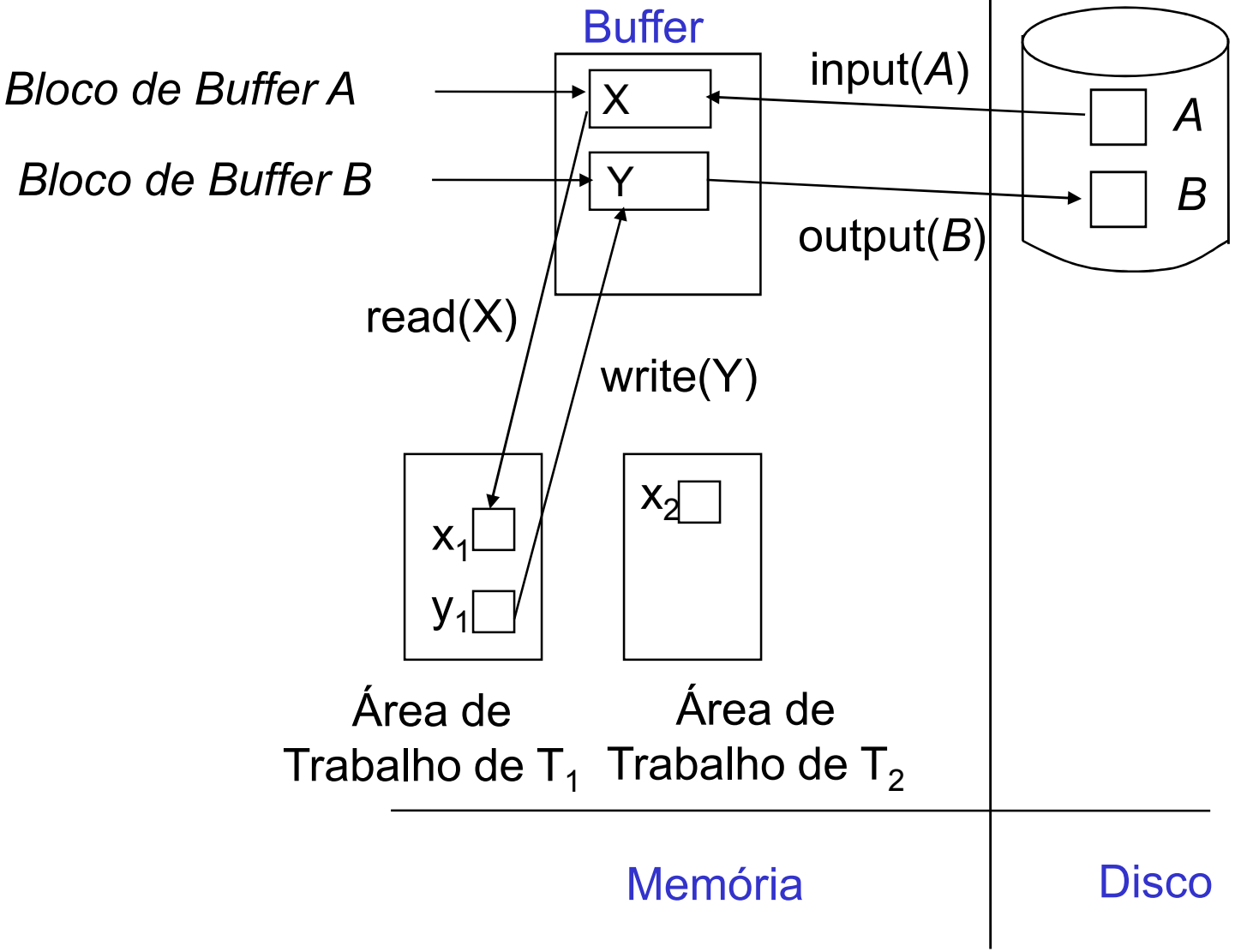
Introdução

Operações de transferência:

- **read(*X*)**: designa o valor do item de dado *X* para a variável de programa x_i.
  a. Se o bloco B_x, no qual X reside não está em memória principal, então é emitido um **input(Bx)**.
  b. Designa a x_i o valor de X a partir do bloco de buffer.

- **write(*X*)**: atribui o valor da variável de programa x_i para o item de dado *X* no bloco de buffer.
  a. Se o bloco B_x no qual reside *X* não está na memória principal, então emite o **input(B_x).**
  b. Designa o valor de x_i para *X* no buffer B_x.

Exemplo de Acesso aos Dados



Introdução

- Ambas as operações podem exigir a transferência de um bloco de disco para a memória principal, mas não exigem a transferência de um bloco da memória principal para o disco.
- O bloco de *buffer* é eventualmente escrito no disco ou porque o buffer está cheio e é preciso liberar espaço, ou porque o sistema deseja refletir a mudança em *B* sobre o disco. Assim, o sistema força uma saída com a operação **output(B)**.
- Nem sempre o **output(B)** ocorre imediatamente depois de um **write(B)**. Então, se o sistema falhar, o valor escrito em *B* por uma transação pode se perder.

Introdução

A = 1000,00
B = 2000,00

T_i
Read(A);
A = A – 50;
Write (A);
Read(B);
B = B + 50;
Write(B)

Sistema:
Output(B_A) → Queda do sistema
Output(B_B)

Possíveis procedimentos de recuperação:

- **Re-executar T_i:** Faz com que o valor de *A* torne-se 900,00, em vez de 950,00. O sistema entra em um estado inconsistente.
- **Não re-executar T_i:** O valor de *A* permanece 950,00 e o valor de *B* permanece 2000,00. O sistema entra em estado inconsistente.

Introdução

- O problema anterior ocorreu porque modificou-se o banco de dados sem ter certeza que a transação seria efetivada de fato.
- O objetivo seria realizar todas ou nenhuma das modificações da transação.
- Se a transação executou diversas alterações, diversas operações de saída podem ser requeridas.
- Para garantir a atomicidade, é necessário primeiro enviar as informações sobre as modificações para um meio estável, sem modificar o banco de dados.
    - **Solução: Registrar histórico em arquivo (*LOG*) !!!**

Recuperação Baseada em LOG

→ Assumindo apenas execuções seriais.

O *LOG* é uma seqüência de registros de log que mantém um arquivo atualizado sobre as atividades realizadas com os dados de um banco de dados.

O Registro do LOG

Um registro de atualização de *log* descreve uma única operação (de escrita) com um dado e possui os seguintes campos:

- T_i, o identificador da transação;
- X_j, o identificador do item de dado;
- V_1, o valor antigo;
- V_2, o valor novo;

$< T_i, X_j, V_1, V_2 >$

- Outros registros de *log*:
  - Início de transação;
  - Efetivação de transação;
  - Aborto de transação.

$< T_i$, *start* $>$
$< T_i$, *commit* $>$
$< T_i$, *abort* $>$

Recuperação baseada em LOG

- Sempre que uma operação de escrita é realizada é essencial que o registro de log equivalente seja criado antes que o banco de dados seja modificado;
- A inutilização de uma operação de escrita (*undo*) pode ser realizada com o uso do valor antigo;
- O *log* deve residir em armazenamento estável;
- O *log* contém o registro completo de todas as atividades realizadas com o banco de dados;
- O tamanho do *log* pode se tornar absurdamente grande. Eventualmente, pode-se apagar o seu conteúdo.

Modificações Adiadas do BD

Garante a atomicidade das transações, quando todas as modificações são escritas no log, adiando a execução das operações de escrita até sua efetivação parcial.

- Assim que uma transação é parcialmente efetivada, as informações no *log* associadas àquela transação são usadas para executar as *escritas adiadas*.

- Se o sistema cair antes do término das escritas ou a transação for abortada, as informações no *log* serão ignoradas.

Modificações Adiadas do BD

- Quando todas as modificações do banco de dados são escritas no *log*, garante-se a propriedade de atomicidade de transação.
- Na técnica de modificações adiadas (ou postergadas), adia-se a execução de todas as operações de *write* (em relação ao Banco de Dados) de uma transação até que ela seja parcialmente efetivada (tenha executado todas as suas ações).

Execução de uma transação

- Um registro <T_i ***start***> é escrito no *log* antes da transação T_i iniciar sua execução.
- Toda operação *write*(x) de T_i, resulta em um novo registro no *log*.
- Quando T_i é parcialmente efetivada, um registro <T_i ***commit***> deve ser escrito no *log*.

Assim que a transação é parcialmente efetivada, os registros no *log* podem ser usados para atualizar o banco de dados e a transação entra em seu estado de efetivação. Neste momento, deve-se garantir que o *log* esteja armazenado em meio estável.

Modificações Adiadas do BD

Também chamadas de *NO-UNDO/REDO*. Nesta técnica, somente o valor novo do item de dado que sofreu alteração é necessário ser guardado no registro de *log*, pois:

- Somente escalonamentos seriais estão sendo considerados;
- As escritas são realizadas após a efetivação parcial da transação;
- No caso de falhas:
  - Antes das escritas no BD: os registros no *log* serão ignorados (o valor antigo do item de dado permanecerá).
  - Após as escritas no BD: executa-se operações de *redo*.

Exemplo:

T_0	T_1	*Log*
		<T_0, start>
Read(A);	Read(C);	<T_0, A, 950>
A := A – 50;	C := C – 100;	<T_0, B, 2050>
Write(A);	Write(C);	<T_0, commit>
Read(B);		<T_1, start>
B := B + 50;		<T_1, C, 600>
Write(B)		<T_1, commit>

Valores iniciais:

A = 1.000
B = 2.000
C = 700

Alterações no BD:

Log	BD
$<T_0$, start>	
$<T_0$, A, 950>	
$<T_0$, B, 2050>	
$<T_0$, commit>	
	A = 950
	B = 2050
$<T_1$, start>	
$<T_1$, C, 600>	
$<T_1$, commit>	
	C = 600

Recuperação

- Procedimento de recuperação em caso de falhas, que resultem em perda de informação no armazenamento volátil:

Redo(T_i): Refazer T_i → define o valor de todos os itens de dados atualizados pela transação T_i para os novos valores (operação idempotente).

Uma transação T_i deverá ser refeita, se e somente se, o sistema de recuperação encontrar os registros <T_i, *start*> e <T*i*, *commit*> no *log*.

Modificações Adiadas do BD - Exemplo

Log	Log	Log
$<T_0$ start>	$<T_0$ start>	$<T_0$ start>
$<T_0, A, 950>$	$<T_0, A, 950>$	$<T_0, A, 950>$
$<T_0, B, 2050>$	$<T_0, B, 2050>$	$<T_0, B, 2050>$
	$<T_0$ commit>	$<T_0$ commit>
	$<T_1$ start>	$<T_1$ start>
	$<T_1, C, 600>$	$<T_1, C, 600>$
		$<T_1$ commit>
(a)	(b)	(c)

Modificações Adiadas do BD - Exemplo

- Ações para o exemplo anterior:

  (a) Nenhuma ação *redo* é necessária;

  (b) É necessário realizar **redo**(T_0), uma vez que há um <T_0 **commit**> registrado;

  (c) Realizar **redo**(T_0) seguido de **redo**(T_1), uma vez que <T_0 **commit**> e <T_1 **commit**> estão presentes no *log*.

Modificação Imediata de BD

- Permite que as modificações no banco de dados sejam realizadas enquanto as transações ainda estão num estado ativo.
- As escritas emitidas por transações ativas são chamadas de modificações não-efetivadas.
- Neste esquema de modificação de BD, o *log* deverá armazenar o valor antigo e o valor novo oriundos das operações de *write.*

Execução de uma Transação

- Um registro <T_i **start**> antes da transação T_i iniciar sua execução.
- Toda operação *write(**X**)* de *T*i é precedida pela escrita de um novo registro no *log*.
- Quando T_i é parcialmente efetivada, um registro <T_i **commit**> deve ser escrito no *log*.

As informações podem ser atualizadas no banco de dados assim que o *log* esteja salvo em meio estável.

Exemplo

T_0	T_1	*Log*
		$<T_0$, *start*$>$
Read(A);	Read(C);	$<T_0$, A, 1000, 950$>$
A := A – 50;	C := C – 100;	$<T_0$, B, 2000, 2050$>$
Write(A);	Write(C);	$<T_0$, *commit*$>$
Read(B);		$<T_1$, *start*$>$
B := B + 50;		$<T_1$, C, 700, 600$>$
Write(B)		$<T_1$, *commit*$>$

Valores iniciais:

A = 1.000
B = 2.000
C = 700

Alterações no BD

Log	BD
<T_0, *start*>	
<T_0, A, 1000, 950>	
<T_0, B, 2000, 2050>	
	A = 950
	B = 2050
<T_0, *commit*>	
<T_1, *start*>	
<T_1, C, 700, 600>	
	C = 600
<T_1, *commit*>	

Recuperação

- Procedimentos de recuperação em caso de falhas, que resultem em perda de informação no armazenamento volátil:

Undo(T_i): Desfazer T_i → retorna aos valores antigos todos os itens de dados atualizados pela transação T_i.

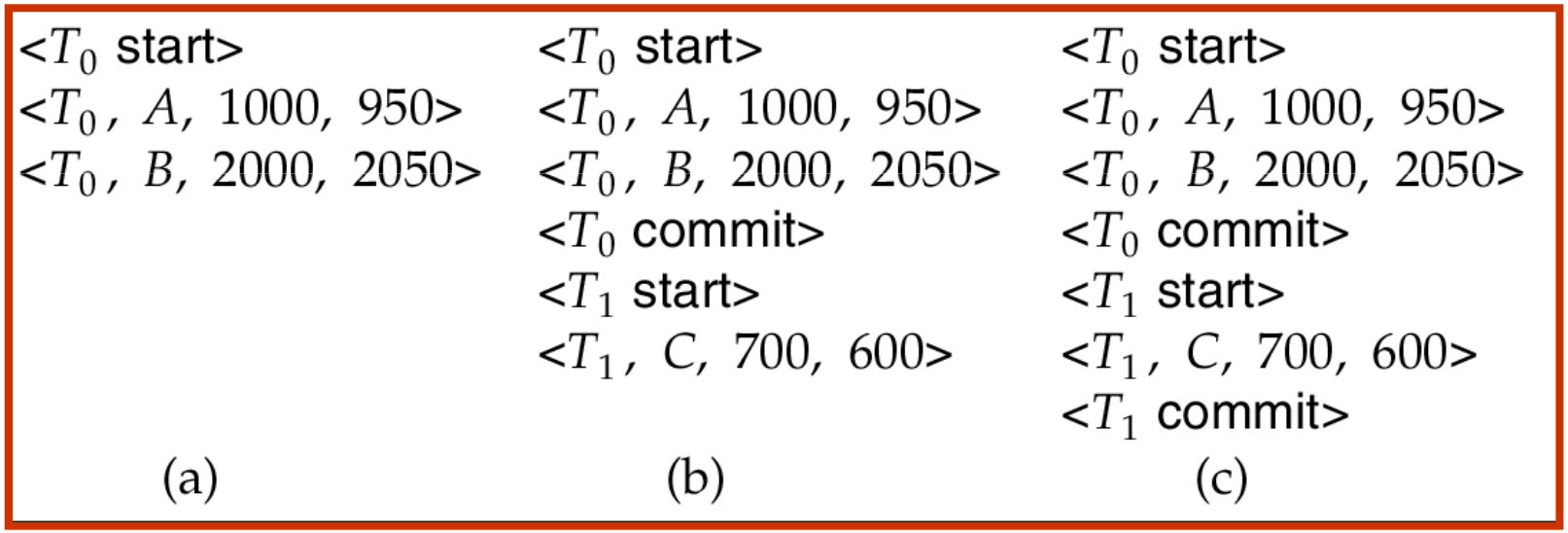
Redo(T_i): Refazer T_i → ajusta os valores de todos os itens de dados atualizados pela transação T_i para os novos valores.

Recuperação

Após a falha, o sistema de recuperação consulta o *log* para determinar quais transação precisam ser desfeitas e quais precisam ser refeitas:

- A transação T_i tem que ser desfeita se o *log* contiver o registro <T_i, ***start***>, mas não possuir o registro <T_i, ***commit***>.
- A transação T_i tem que ser refeita se o *log* contiver tanto o registro <T_i, **start**> quando o registro <T_i, ***commit***>.

Modificação Imediata de BD - Exemplo



Modificação Imediata de BD - Exemplo

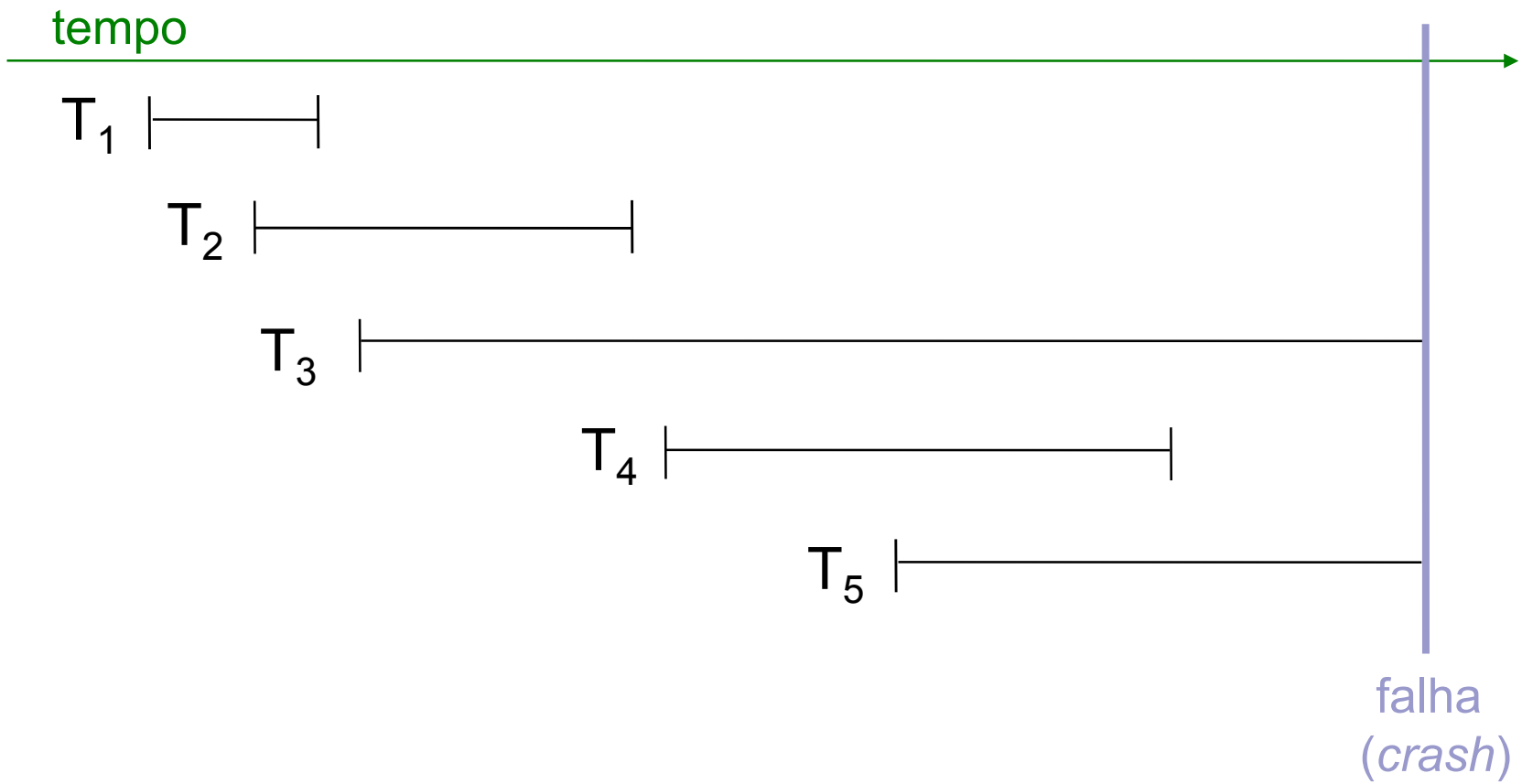
- Ações para o exemplo anterior:

  (a) **undo** (T_0): *B* é restaurado para 2000 e *A* para 1000.

  (b) **undo** (T_1) e **redo** (T_0): *C* é restaurado para 700, e então, *A* e *B* são retornam aos valores 950 e 2050, respectivamente.

  (c) **redo** (T_0) and **redo** (T_1): Os valores de *A*, *B* e *C* na conta, após esses procedimentos, são 950, 2050 e 600, respectivamente.

Outro exemplo:



lista-UNDO: T_3, T_5 (devem sofrer *undo*)

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

Checkpoints

Na existência de uma falha, o sistema de recuperação deve, a princípio, percorrer todo o *log* para saber quais transações devem ser desfeitas. Dificuldades:

- O processo de pesquisa consome tempo;
- Muitas das transações que necessitam ser refeitas já escreveram suas atualizações no BD. Embora refazê-las não cause dano algum, a recuperação se tornará mais onerosa.

Checkpoints

Solução: introdução de *checkpoints* (pontos de controle). Os *checkpoints* são registros inseridos no *log* periodicamente e exigem a execução da seqüência de operações abaixo:

- Suspender, temporariamente, a execução de todas as transações.
- Forçar as escritas das operações de *write* das transações, da memória principal para o disco.
- Escrever o *checkpoint* no *log* e forçar a escrita no *log* em disco.
- Reassumir a execução das transações.

Checkpoints

Se T_i é efetivada antes do *checkpoint*, o registro <T_i, ***commit***> aparece no *log* antes do *checkpoint*, quaisquer modificações feitas por T_i ou já foram escritas no banco de dados antes do *checkpoint* ou o foram como parte do *ckeckpoint* propriamente dito.

Assim, no momento de recuperação não haverá necessidade de uma operação de **redo** sobre T_i.

Checkpoints

- Lembre-se, apenas escalonamentos seriais são permitidos!

Após uma falha o sistema de recuperação examina o *log* para determinar a última transação T_i anterior ao *checkpoint* mais recente.

A pesquisa é retroativa no *log* e procura a última transação T_i anterior ao *checkpoint* mais recente (o último registro <T_i, ***start***> antes do *checkpoint*).

As operações de ***undo*** e ***redo*** devem ser aplicadas, de forma apropriada, a T_i e a todas as transações T_j que iniciaram depois dela.

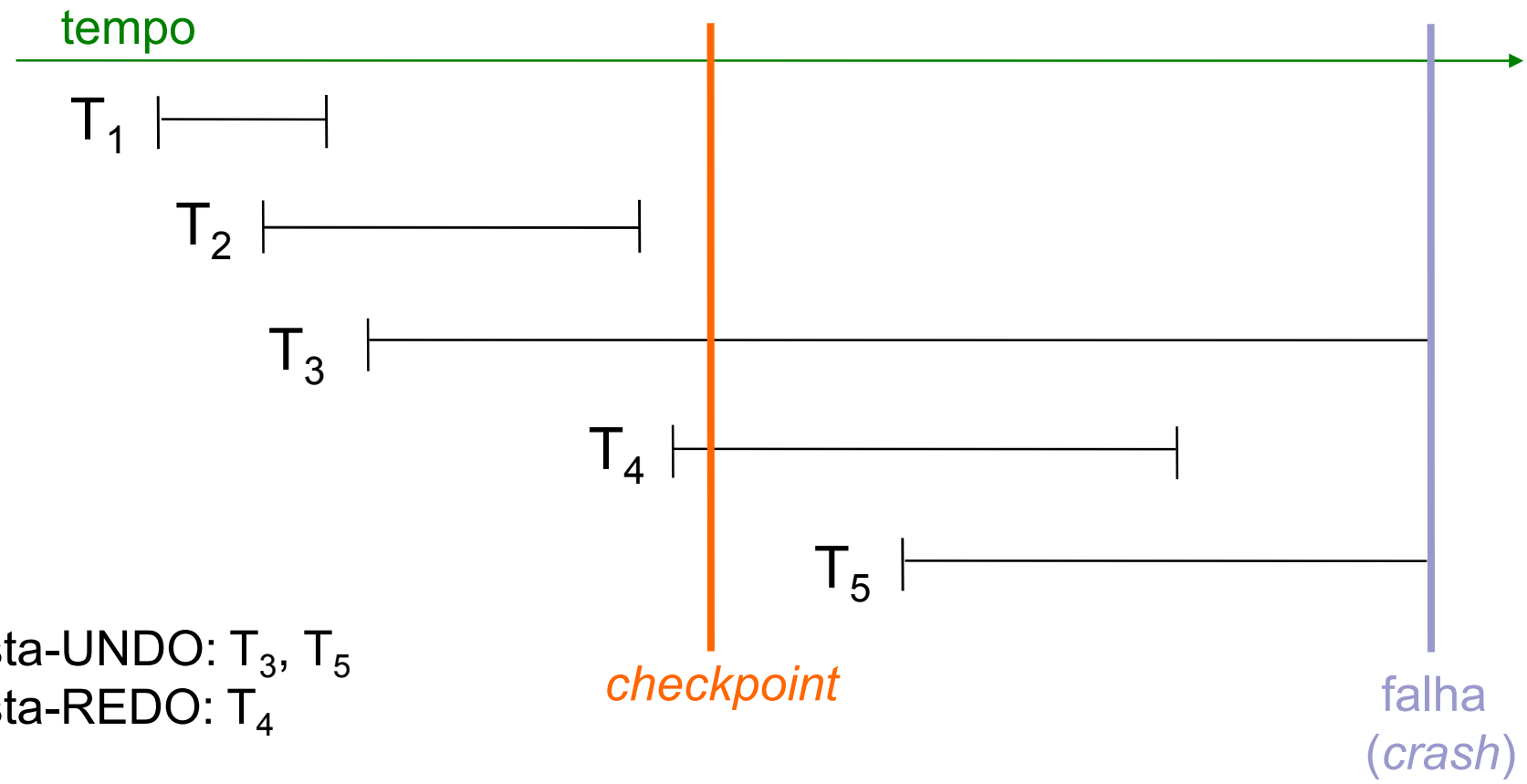
Checkpoints

As operações de recuperação exigidas, se a técnica de modificação imediata é empregada, são as seguintes:

- Para todas as transações T_k em T que não possuem nenhum registro <T_k, ***commit***> no *log*, execute **undo**(T_k).
- Para todas as transações T_k em T tais que o registro <T_k, ***commit***> aparece no *log*, execute **redo**(T_k)

Se a técnica de modificação adiada está em uso, nenhuma operação de ***undo*** será aplicada.

Técnica UNDO/REDO com *Checkpoint*



lista-UNDO: T_3, T_5
lista-REDO: T_4

- 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***.

Exemplo:

- Considere as transações {T_0, T_1, ... T_{100}}, executadas nesta ordem.
- Suponha que o *checkpoint* mais recente tenha ocorrido durante a execução da transação T_{67}.
- Então somente as transações T_{67} até T_{100} necessitam ser consideradas durante o esquema de recuperação.
- Cada uma delas será refeita se tiver sido efetivada. Caso contrário, será desfeita.

Interação com Controle de Concorrência

- Suponha que uma transação T_0 tenha que ser desfeita e um item de dado Q, atualizado por T_0, tenha que recuperar seu valor antigo. Para isso, utiliza-se as informações registradas no *log* e a operação ***undo***.

- Suponha ainda que uma segunda transação T_1 também tenha realizado uma atualização sobre Q antes de T_0 ser desfeita. Esta atualização, processada por T_1, será perdida quando T_0 for revertida.

- Para evitar esta situação, usa-se o bloqueio em duas fases que mantém os bloqueios de escrita até o final da transação. (SEVERO)

Reversão de Transação

- A reversão de uma transação T_i é realizada da seguinte forma:
  - O *log* é percorrido, do fim para o início;
  - Para cada registro da forma $<T_i, X_j, V_1, V_2>$, o item de dados X_j é restaurado para seu valor antigo V_1;
  - O trabalho termina quando o registro $<T_i$, ***start***$>$ é encontrado.

- Por que percorrer o *log* de trás para frente?
  - Pelo fato de que uma transação pode ter atualizado um item de dados mais de uma vez. Se uma transação atualiza um mesmo item de dados duas vezes:

$<T_i, A, 10, 20>$

$<T_i, A, 20, 30>$

Reversão de Transação

- Por que percorrer o *log* de trás para frente?
  - Considere o exemplo:

    $<T_i, A, 10, 20>$

    $<T_i, A, 20, 30>$

  - Os registros de *log* representam uma modificação do item de dados *A* por T_i seguida por outra modificação de *Ti*. A varredura do *log* ao contrário define *A* corretamente para 10. Se o *log* fosse varrido na direção normal, *A* seria definido como 20, o que seria incorreto.

O Protocolo de Bloqueio x Reversão

- Quando se está revertendo uma transação, o protocolo de bloqueio deve manter os bloqueios de escrita da transação, até que o trabalho de reversão seja concluído.

- Como se está supondo o uso do protocolo de bloqueio em duas fases severo, nenhuma outra transação terá que ser revertida por ocasião da reversão de outra transação.

Checkpoints

- Para situações onde a concorrência não existe, o uso dos *checkpoints* faz com que na recuperação se considere:
  - As transações que iniciaram após o *checkpoint* mais recente;
  - A transação, se houver alguma, que estava ativa no momento da escrita do *checkpoint* mais recente.
- Quando existe concorrência, várias transações podem estar ativas no momento em que o *checkpoint* for inserido no *log*.
- Assim, o registro de *log* referente ao *checkpoint* será da forma <*checkpoint L*>, onde *L* é a lista de transações ativas no momento de inserção do *checkpoint*.

Recuperação

- Para a recuperação com transações concorrentes, o sistema deve construir duas listas:
  - Lista inutilizar (***undo-list***): composta por transações que devem ser inutilizadas.
  - Lista refazer (***redo-list***): composta por transações que devem ser refeitas.

Recuperação

- A construção:
  - Inicialmente, ambas as listas estão vazias;
  - Examina-se o *log*, em ordem decrescente, até o primeiro *checkpoint*:
    - Para cada registro <T_i, ***commit***>, adiciona-se T_i à **lista refazer**.
    - Para cada registro <T_i, ***start***>, se T_i não está na **lista refazer,** adiciona-se T_i na **lista inutilizar**.
    - Checa-se a lista *L* no registro do *checkpoint* em questão. Para cada transação T_i em *L*, se T_i não estiver da **lista refazer**, então será adicionada à **lista inutilizar.**

Recuperação

- A recuperação propriamente dita:
  - Reexaminar o *log* a partir do registro mais recente e processar um UNDO para cada registro de *log* pertencente à transação T_i na lista inutilizar. O exame pára quando os registros <T_i ***start***>, para cada transação T_i da lista inutilizar, são encontrados (percorre-se o *log* em ordem cronológica decrescente);
  - Localizar o registro <*checkpoint L*> mais recente;
  - Examinar o *log* a partir do registro <*checkpoint L*> encontrado, até o final do *log* e executar um ***redo*** para cada registro de *log* pertencente a uma transação T_i que está na lista refazer (percorre-se o *log* em ordem cronológica crescente).

Recuperação

- Por que inutilizar antes de refazer?

Suponha que o item de dado *A* tenha inicialmente o valor 10. Suponha que uma transação T_i tenha atualizado o item de dado *A* para 20 e depois foi abortada. A reversão da transação restauraria o valor de *A* para 10.

Suponha que outra transação T_j tenha atualizado o item de dado *A* para 30 e tenha sido efetivada.

Em seguida o sistema cai e o estado do *log* é:

$<T_i, A, 10, 20>$

$<T_j, A, 10, 30>$

$<T_j,$ ***commit***$>$

Se o passo REDO for processado primeiro, *A* será ajustado para 30 e, no passo UNDO, será ajustado para 10, sendo que o valor que deve permanecer em *A* é 30.

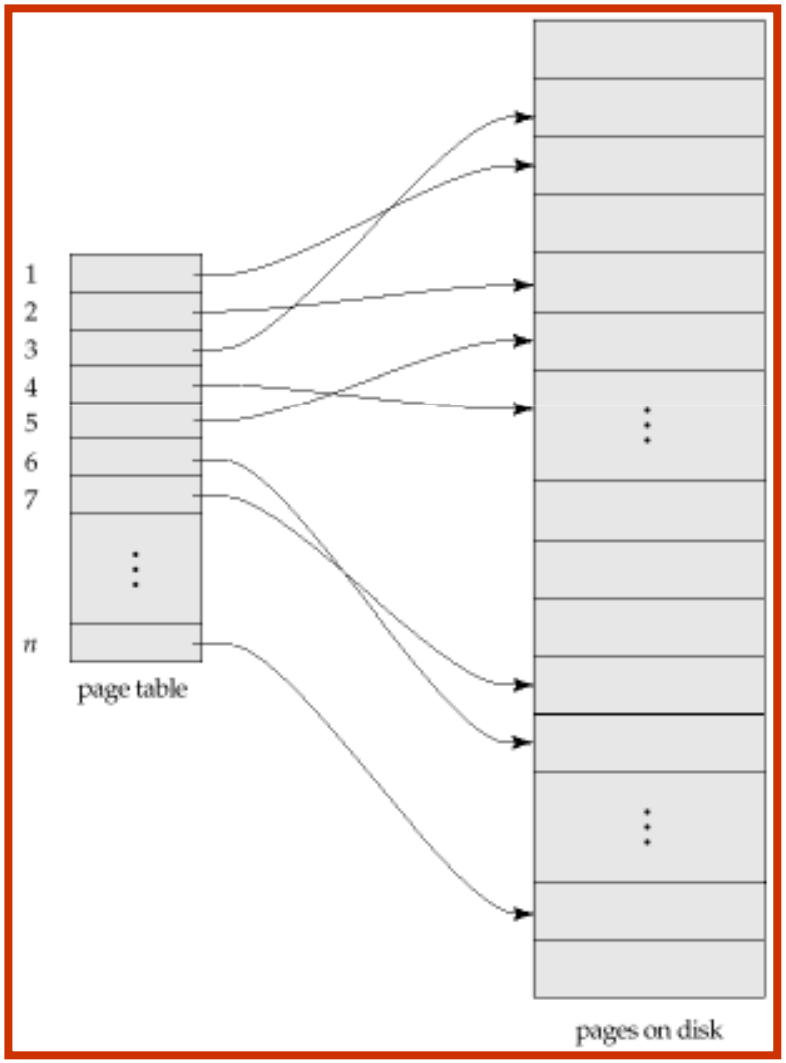
Técnica Baseada em Shadow Pages

- *Shadow Paging* é uma alternativa à recuperação baseada em *log*;
- É um esquema útil se as transações forem executadas serialmente.
- Idéia: Manter duas tabelas de páginas durante o tempo de vida de uma transação – a Tabela de Páginas Corrente (TPC), e a Tabela de Páginas Shadow (TPS);
- Armazena a Tabela de Páginas *Shadow* em meio não-volátil, tal que o estado do BD anterior à execução da transação possa ser recuperado.
- Ao iniciar, ambas as tabelas de páginas são indênticas. Somente a tabela de páginas correntes é usada para cada item de dados acessados durante a execução da transação.

Técnica Baseada em Shadow Pages

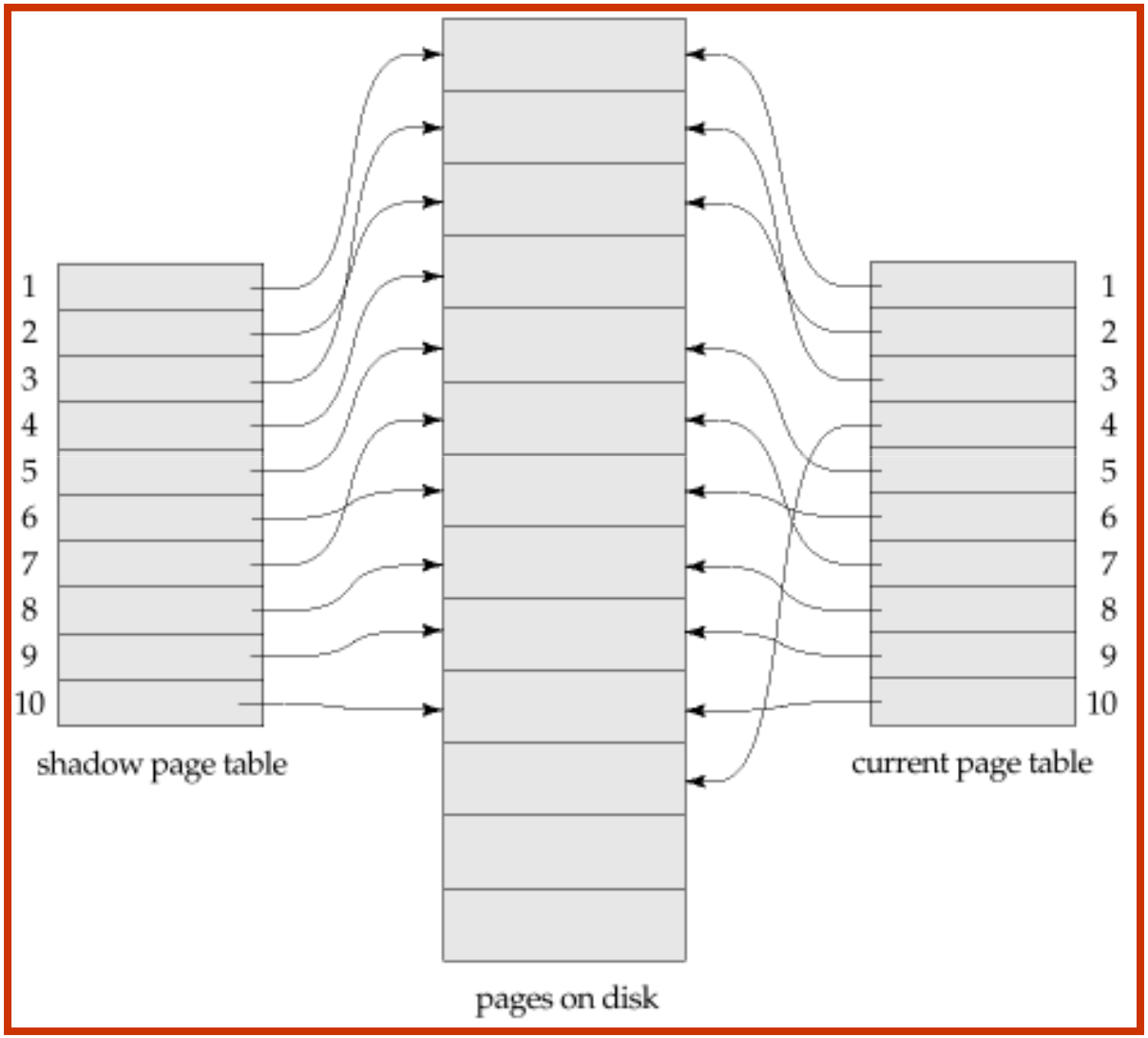
- Supõe a existência de uma tabela de blocos (páginas) de disco que mantém dados do BD (TPC)';
- A TPC é copiada para uma TPS a cada nova transação T_i
  - Páginas atualizadas por T_i são copiadas para novas páginas de disco e TPC é atualizada;
  - TPS não é atualizada enquanto T_i está ativa.
- Em caso de falha de T_i, 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).

Exemplo de Tabela de Páginas



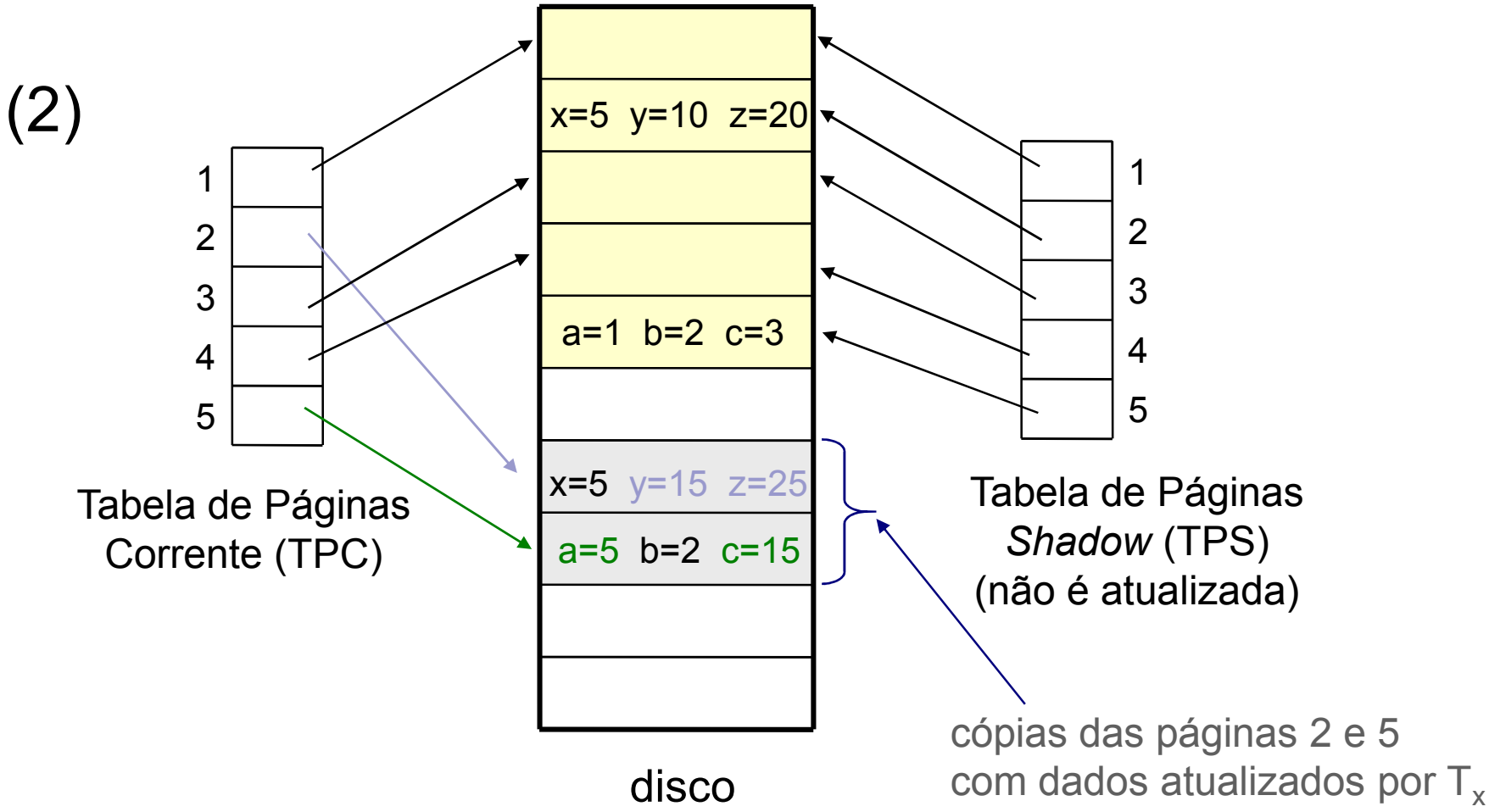
Exemplo de Paginação Shadow

(1)



→ Tabelas de páginas Shadow e corrente depois da escrita da página 4.

Exemplo de Paginação Shadow



Shadow Pages - Procedimento

- Quando uma transação T_i inicia:
  - TPS ← TPC
  - FORCE TPS
- Quando T_i atualiza dados de uma página *P*
  - Se é a primeira atualização de T_i 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'* ← *P* (grava nessa página livre em disco)
    - O apontador de *P* na TPC agora aponta para *P'*
  - Atualiza-se os dados em *P.*

Shadow Pages – Procedimento (cont.)

- Quando T_i solicita commit
  - FORCE das páginas $P_1, ..., P_n$ atualizadas por T_i que ainda não foram para disco
    - com FORCE da TPC atualizada primeiro;
    - lembre-se que $P_1, ..., P_n$ 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 T_i
  - Faz-se TPC $\leftarrow$ TPS
- Falha após o passo 3:
  - Não é preciso REDO, pois as atualizações de T_i estão, garantidamente, no BD;

Técnica Shadow Pages - Vantagens

- Adequada a SGBD monousuário
  - Uma transação executando por vez;
- Sem overhead de escrita dos registros de *log*;
- Recuperação é trivial;

Técnica Shadow Pages - Desvantagens

- Gerenciamento complexo em SGBD multiusuário;
- Os dados podem estar fragmentados no disco;
- Requer coleta de lixo:
  - Quando T_i encerra, existem páginas obsoletas;

Fontes Bibliográficas:

- **Sistema de Banco de Dados. (**Cap. 17**)** Abraham Silberchatz, Henry F. Korth, S Sudarshan. 5ª Ed. Campus, 2006.

- **Sistemas de Banco de Dados. (**Cap. 19**)** Ramez Elmarsri e Sham Navathe. 4ª Ed. Pearson, 2005.