Preâmbulo

A recuperação de falhas de transação, em geral, significa que o banco de dados é restaurado ao estado consistente mais recente antes do momento da falha.

Para fazer isso, o sistema precisa manter informações sobre as mudanças que foram aplicadas aos itens de dados pelas diversas transações.

Em geral, os SGBDs mantêm um **arquivo de *log***, que possui o registro cronológico das modificações aplicadas ao banco de dados.

Relembrando... Tipos de Falha

As falhas são geralmente classificados como **falhas de transação**, **falhas de sistema** e **falhas de mídia**.

Há várias razões possíveis para uma transação falhar:

1. Falha do computador (falha do sistema). Um erro de hardware, de software ou de rede ocorre no sistema durante a execução da transação. Falhas de hardware são geralmente falhas de media; por exemplo, falhas de memória principal.

2. Erro de transação ou sistema. Algumas operações na transação podem falhar, tal como *integer overflow*, divisão por zero ou erros lógicos de programação. Também, o usuário pode interrompera transação durante a sua execução.

Relembrando... Tipos de Falha

3. Condições de exceção detectadas pela transação. Por exemplo, uma condição de exceção pode ser detectada para uma regra de negócio, tal como saldo insuficiente em uma transação bancária. Esta excepção poderia ser programada na própria transação e, em tal caso, não seria considerada como uma falha de transação.

4. Aplicação de controle de concorrência. O mecanismo de controle de concorrência pode decidir:
(i) abortar uma transação, porque ela viola serialização, ou
(ii) abortar uma ou mais transações para resolver uma situação de impasse (*deadlock*) entre várias transações.
As transações abortadas nesses casos são normalmente reiniciadas automaticamente em um momento posterior.

Relembrando... Tipos de Falha

5. Falha de disco. Alguns blocos de disco podem perder seus dados (exemplo, falha no mecanismo físico de acesso). Isto pode acontecer durante operações de leitura ou de escrita da transação.

6. Problemas físicos e catástrofes. Refere-se a uma lista interminável de problemas que inclui falha de energia, falha de ar condicionado, incêndio, roubo, sabotagem, etc.

Falhas dos Tipos 1, 2, 3 e 4 são mais comuns que as de Tipos 5 e 6.

Sempre que uma falha dos Tipo 1 a 4 ocorre, o sistema tem de **manter informação suficiente** para recuperar rapidamente a falha.

Falhas do tipo 5 ou 6 requerem mecanismo de recuperação mais demorado.

Preâmbulo

Cenário 1

Se houver dano extensivo a uma grande parte do banco de dados devido à falha grave (falhas do Tipo **5** ou **6**), o método de recuperação restaura uma cópia antiga do banco de dados que teve *backup* (em geral, é uma cópia em dispositivo externo).

Em seguida, tenta reconstruir o estado "atual" do banco de dados, a partir do instante da cópia *backup*, até um estado o mais próximo possível da ocorrência da falha:

>> reaplica as transações que foram concluídas com sucesso (*committed*) após o momento do *backup*.

Preâmbulo

Cenário 2

Quando uma falha não catastrófica tiver ocorrido (falhas do Tipo **1** a **4**), a estratégia de recuperação é identificar quaisquer mudanças que possam causar uma inconsistência no banco de dados.

Por exemplo, uma transação que atualiza alguns itens de banco de dados no disco, mas não foi confirmada (ainda não efetuou o *commit*), **precisa ter suas alterações de dados revertidas**, desfazendo suas operações de escrita.

Também pode ser necessário **refazer algumas operações**, a fim de restaurar um estado consistente do banco de dados, por exemplo, se uma transação tiver sido confirmada (*committed*), mas algumas de suas operações de gravação ainda não foram gravados no disco.

Conceitualmente, há duas técnicas pertinentes a falhas do Cenário 2: **atualização adiada** e **atualização imediata**.

Preâmbulo

Operações UNDO e REDO

UNDO.
Desfazer uma alteração de uma transação que está registrada no *log*.

REDO.
(Re)Aplicar uma alteração de uma transação que está registrada no *log*.

UNDO e REDO são idempotentes:
>> executar a operação múltiplas vezes equivale a executar um única vez.

Durante o processo de recuperação, várias operações UNDO e REDO são executadas.

Se o processo de recuperação falhar, operações UNDO e REDO poderão ser reexecutadas.

Classificação de Algoritmos

Atualização adiada. (algoritmo NO-UNDO/REDO)

As atualizações **não são aplicadas** fisicamente ao banco de dados em disco até que uma transação alcance o ponto de *commit*.

Antes do *commit*:

>> todas as atualizações são registradas em *buffers* do SGBD (*cache* do banco de dados),

>> as atualizações são registradas persistentemente no *log*.

Após o *commit*:

>> as atualizações são registradas no banco de dados em disco.

Se uma transação falhar antes de alcançar o *commit*, nenhuma alteração foi aplicada ao banco de dados:

>> operação UNDO não é necessária.

Pode ser necessário aplicar as operações confirmadas (*committed*) no *log*, pois não há a garantia que elas tenham sido aplicadas ao banco de dados em disco:

>> operação REDO é necessária.

Classificação de Algoritmos

Atualização imediata. (algoritmos UNDO/REDO e UNDO/NO-REDO)

O banco de dados pode ser atualizado por algumas operações das transações, antes que as transações alcancem o ponto de *commit*.

>> contudo, as operações **devem ser** gravadas no *log*, antes que possam ser gravadas no banco de dados propriamente dito.

Se uma transação falhar antes de alcançar o *commit*, o efeitos de suas operações aplicadas ao banco de dados podem ser desfeitos:

>> operação UNDO é aplicada às operações que estão no *log*.

>> operação REDO é aplicada às operações, quando há o *commit* da transação no *log*.

Há uma variação do algoritmo, onde todas as atualizações são gravadas no banco de dados antes do *commit* da transação:

>> operação REDO não é necessária;

>> algoritmo UNDO/NO-REDO.

Cache do Banco de Dados

O processo de recuperação em geral está bastante interligado às funções do sistema operacional — em particular, o *buffering* de páginas de disco do banco de dados no ***cache*** de memória principal do SGBD.

Normalmente, várias páginas de disco que incluem os itens de dados a serem atualizados são mantidas em ***cache*** nos *buffers* da memória principal e, depois, atualizados na memória antes de serem gravados de volta no disco.

Cache do Banco de Dados

O processo de recuperação é ligado com funções do sistema operacional.

>> O SGBD invoca funções de “baixo-nível” : manuseio de blocos de memória.

***Cache* de banco de dados**

É uma coleção de blocos de memória que são mantidos sob o controle do SGBD.

Quando o SGBD precisa fazer uma operação (leitura ou gravação) em algum item, ele checa se o item está presente na *cache* do banco de dados.

Em caso negativo, as páginas de disco são copiadas para a cache.

Pode ser necessário a substituição de buffers ocupados, aplicando-se alguma técnica de gerenciamento de memória:

>> *least recently used* (LRU) ou *first-in-firstout* (FIFO),

>> específica do SGBD, tal como *Least-Likely-to-Use*.

Cache do Banco de Dados

dirty bit.

Cada *buffer* na cache possui um ***dirty bit***, que indica se o *buffer* foi modificado:

>> quando uma página for lida (do disco para o *buffer*), ***dirty-bit* = 0**;

>> quando o buffer é modificado, ***dirty-bit* = 1**.

Quando o conteúdo de um *buffer* da cache é substituído (*flushed*), o conteúdo antigo deverá ser escrito de volta na página correspondente em disco se ***dirty-bit* = 1**.

pin bit.

Uma *buffer* na cache possui ***pin-bit*=1**, se ele temporariamente não puder ser escrito de volta para o bloco correspondente no disco:

>> por exemplo, o protocolo de recuperação exige que *buffers* não sejam escritos em disco, até que as transações que alteraram tais *buffers* tenham sido confirmadas (*committed*).

Cache do Banco de Dados

Flushing a buffer.
Descarregar um *buffer* modificado para o disco.

BeFore Image (BFIM). Valor do item de dado antes a operação de atualização.
AFter Image (AFIM). Valor do item de dado após a operação de atualização.

In-place updating.
Escreve o *buffer* modificado no mesmo local de disco original, sobrepondo o valor antigo:

>> uma única cópia do bloco em disco é mantida;
>> apenas AFIM é mantida m disco (AFIM sobrepõe BFIM).

Shadowing.
Escreve o *buffer* modificado em uma local diferente do disco:

>> múltiplas versões de itens de dados podem ser mantidas;
>> são mantidas em disco AFIM e BFIM;
>> estratégia pouco comum.

Cache do Banco de Dados

Quando se usa *in-place updating*, é necessário o uso de arquivo de *log* para guardar a BFIM do item de dado:

>> lembrando, em *in-place updating*, AFIM sobrepõe BFIM.

Write-ahead logging (escrever a frente no *log*)

O mecanismo de recuperação garante que a BFIM do item de dado é registrada no *log*, antes que a BFIM seja sobreposta pela AFIM no banco de dados em disco.

Este processo é necessário para as operações UNDO durante o processo de recuperação.

Log

Tipos de registro de log.

Registro do tipo REDO.
Possui o novo valor (AFIM) de um item de dado gravado.

Registro do tipo UNDO.
Possui o antigo valor (BFIM) de um item de dado gravado.

Em um algoritmo UNDO/REDO, ambos os tipos de registro são combinados no arquivo de *log*, por quê ?

Log

Principais tipos de registros no *Log*

início de transação: <start T_x>

commit de transação: <commit T_x>

atualização: <write T_x,X,beforeImage,afterImage>

não é necessário em log REDO

não é necessário em log UNDO

Log

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>
...
```

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

T_2

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

T_3

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

Cache do Banco de Dados

Cache do banco de dados inclui *buffers* com: (i) blocos de dados; (ii) blocos de índices; (iii) blocos de *log*.

O *cache* do BD possui vários *buffers* destinado aos blocos de *log*;
>> tipicamente, os **n** blocos mais recentemente gravados.

Quando o SGBD solicita a gravação de um registro de *log* é gravado:
(1) o registro é gravado em um *buffer* de *log* da cache do BD; e
(2) em seguida, é persistido no arquivo de *log*.

Quando uma atualização é feita um um bloco de dados (que já estiver na *cache* do BD em memória), um registro de *log* é gravado no [último] *buffer* de log (também presente na *cache* do BD).

Devido à abordagem ***write-ahead logging***, os *buffers* de *log* são [primeiramente] gravados em disco, antes que o bloco de dados modificado seja escrito de volta em disco.

Cache do Banco de Dados

Os termos ***steal/no-steal*** e ***force/no-force*** referem-se à política sobre quando uma página de dados será escrita em disco a partir da *cache* do BD.

no-steal

Ocorre quando um *buffer* modificado da *cache* não puder ser escrito em disco antes que a transação seja confirmada (*committed*):

>> ou seja, um bloco na *cache* modificado por uma transação Tx não pode ser gravado antes do *commit* de Tx.

O **pin-bit** indica se a página pode ser escrita em disco temporariamente.

Vantagem: processo de recuperação é mais simples - evita dados de transações inacabadas sendo gravadas no BD:

>> a operação UNDO não é necessária na recuperação de dados;

>> as transações não terão quaisquer de suas atualizações no disco, antes que essas transações sejam confirmadas (*committed*).

Cache do Banco de Dados

Os termos steal/no-steal e force/no-force referem-se à política sobre quando uma página de dados será escrita em disco a partir da cache do BD.

steal

Ocorre quando o protocolo de recuperação permitir a escrita de *buffers* de dados modificados antes da confirmação (*commit*) da transação:

>> ou seja, um bloco na *cache* modificado por uma transação Tx pode ser gravado antes do *commit* de Tx.

É útil quando o SGBD necessita de um *buffer* da *cache* para outra transação, podendo substituir um *buffer* pertencente a uma transação não confirmada.

Vantagem: não há necessidade de manter blocos bloqueados por transações.

Cache do Banco de Dados

force
Ocorre quando todas as páginas atualizadas por uma transação são imediatamente escritas em disco, antes mesmo que a transação seja confirmada (*committed*).

no-force
Não ocorre a gravação imediata em disco das páginas de dados modificadas.

Em ***force***, a operação de REDO nunca será necessária durante a recuperação, visto que qualquer transação [confirmada] terá todas as suas atualizações escritas em disco antes de sua confirmação (*commit*).

Cache do Banco de Dados

No esquema de recuperação com atualização adiada (NO-UNDO), aplica-se a abordagem ***no-steal*** :

>> pode ser necessário muito espaço na *cache* (vários *buffers*) para uma única transação

Segundo Elmasri e Navathe (2011), a abordagem mais usada é ***steal/no-force***:

>> ***steal*** pode reduzir o espaço em *cache* para uma transação;

>> ***no-force*** favorece que uma página atualizada possa estar ainda em *cache* quando outra transação necessitar atualizá-la (reduzindo I/O);

>> se uma página (bloco) for atualizada intensivamente por múltiplas transações, a economia pela redução de I/O será importante.

Cache do Banco de Dados

Recuperação quando atualização ***in-place*** é usada :

>> atualizações devem ser [primeiramente] gravadas antes no *log*, antes de serem aplicadas ao banco de dados.

Exemplo com ***write-ahead logging*** (WAL) e algoritmos UNDO/REDO:

(1) A BFIM de um item não pode ser sobrescrita por sua AFIM no banco de dados em disco, até que todos os registros do tipo UNDO da transação (até o ponto de atualização) tenham sido escritos no *log* em disco.

(2) A operação *commit* de uma transação não pode ser completada até que todos os registros do tipo REDO e UNDO da transação tenha sido escritos no *log* em disco.

Checkpoints

Outro tipo de registro no *log* é chamado ***checkpoint*** :

[*checkpoint* , lista-de-transações-ativas]

Este registro é escrito no *log* periodicamente, quando o sistema escreve no banco de dados em disco **todos os *buffers*** que tenham sido modificados.

Como consequência, todas as transações que tenham escrito no *log* o registro **[commit,T]** antes do *checkpoint*, não precisarão refazer suas operações de escrita em caso de falha do sistema:

>> as atualizações dessas transações foram registradas em disco no ponto do *checkpoint*.

O registro de *checkpoint* possui uma **lista-de-transações-ativas**, que devem identificadas durante o processo de recuperação.

Checkpoints

O subsistema de recuperação deverá aplicar a frequência na qual ocorrerá *checkpoint*.

Os parâmetros podem ser:

>> intervalo de tempo, ou

>> número de transações confirmadas (*committed*) desde o último *checkpoint*.

Ações previstas no *checkpoint*:

1. suspender a execução de transações temporariamente;
2. forçar a escrita em disco de todos os *buffers* de memória que tenham sido modificados;
3. escrever um registro de *checkpoint* no *log*, e forçar a escrita desse registro em disco;
4. retomar a execução de transações.

Checkpoints

O tempo necessário para forçar a escrita de todos os *buffers* modificados pode atrasar as transações suspensas (Passo 1).

Para reduzir esse atraso, pode-se usar o ***fuzzy checkpointing***:

(1) o sistema escreve no log um registro **[begin_checkpoint]** e retorma as transações suspensas, sem ter que esperar a conclusão do Passo 2;

(2) quando o Passo 2 é completado, um registro **[end_checkpoint, …]** é escrito no *log*, com a informação coletada durante o processo;

(3) até que o Passo 2 seja completado, o registro de *checkpoint* anterior é o último (mais recente) registro de *checkpoint* válido.

Rollback

Se uma Transação **T** falhar após alguma atualização no banco de dados, mas antes de **T** alcançar o *commit*, é necessário desfazer as alterações realizadas por **T**.

Se algum item de dado **X** for modificado por **T**, é preciso restaurar os valores anteriores de **X** (BFIM):

>> operação UNDO.

Se uma Transação **T** sofreu *rollback*, qualquer Transação **S** que tenha, nesse ínterim, lido o valor **X** escrito por **T**, deverá também sofrer *rollback*.

Similarmente, uma vez que **S** sofreu *rollback*, qualquer Transação **R** que tenha lido o valor **X** escrito por **S**, deverá também sofrer *rollback*.

>> ***rollback* em cascata** ;
>> uma situação que consome muitos recursos e tempo.

Rollback

(a)

T_1	T_2	T_3
read_item(A)	read_item(B)	read_item(C)
read_item(D)	write_item(B)	write_item(B)
write_item(D)	read_item(D)	read_item(A)
	write_item(D)	write_item(A)

(b)

		A	B	C	D
		30	15	40	20
	[start_transaction,T_3]				
	[read_item,T_3,C]				
*	[write_item,T_3,B,15,12]		12		
	[start_transaction,T_2]				
	[read_item,T_2,B]				
**	[write_item,T_2,B,12,18]		18		
	[start_transaction,T_1]				
	[read_item,T_1,A]				
	[read_item,T_1,D]				
	[write_item,T_1,D,20,25]				25
	[read_item,T_2,D]				
**	[write_item,T_2,D,25,26]				26
	[read_item,T_3,A]				

← System crash

Figure 23.1
Illustrating cascading rollback (a process that never occurs in strict or cascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.

* T_3 is rolled back because it did not reach its commit point.

** T_2 is rolled back because it reads the value of item B written by T_3.

Rollback

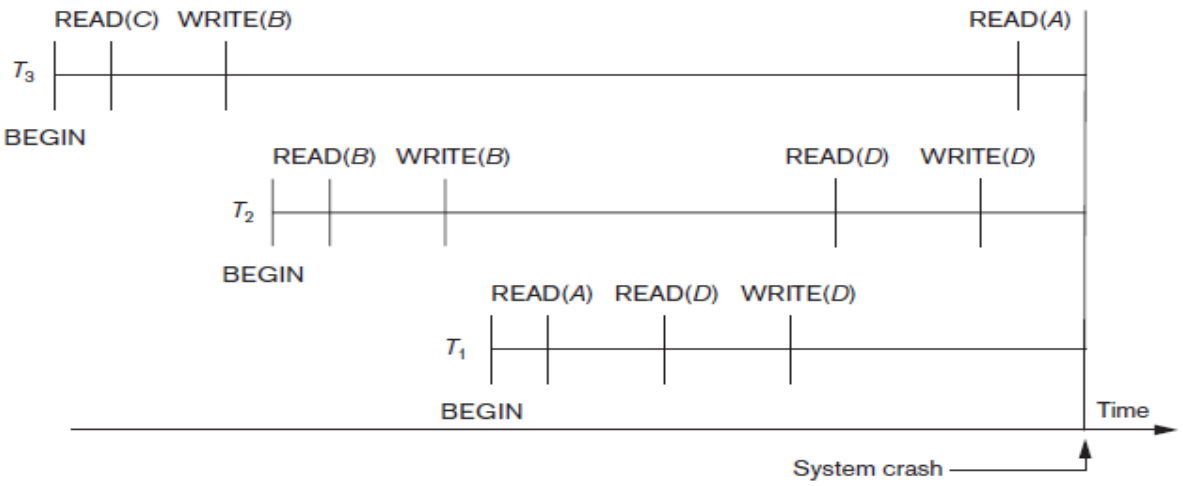
(b)

		A	B	C	D
		30	15	40	20
	[start_transaction, T_3]				
	[read_item, T_3, C]				
*	[write_item, T_3, B, 15, 12]		12		
	[start_transaction, T_2]				
	[read_item, T_2, B]				
**	[write_item, T_2, B, 12, 18]		18		
	[start_transaction, T_1]				
	[read_item, T_1, A]				
	[read_item, T_1, D]				
	[write_item, T_1, D, 20, 25]				25
	[read_item, T_2, D]				
**	[write_item, T_2, D, 25, 26]				26
	[read_item, T_3, A]				
	← System crash				

* T_3 is rolled back because it did not reach its commit point.

** T_2 is rolled back because it reads the value of item B written by T_3.

(c)



Técnica *UNDO/REDO*

Técnica de recuperação em **atualização imediata**.

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 confirmadas:
possuem *commit* gravado no *log*;

>> lista-UNDO: IDs de transações ativas.

Tanto UNDO quanto REDO devem ser realizados:

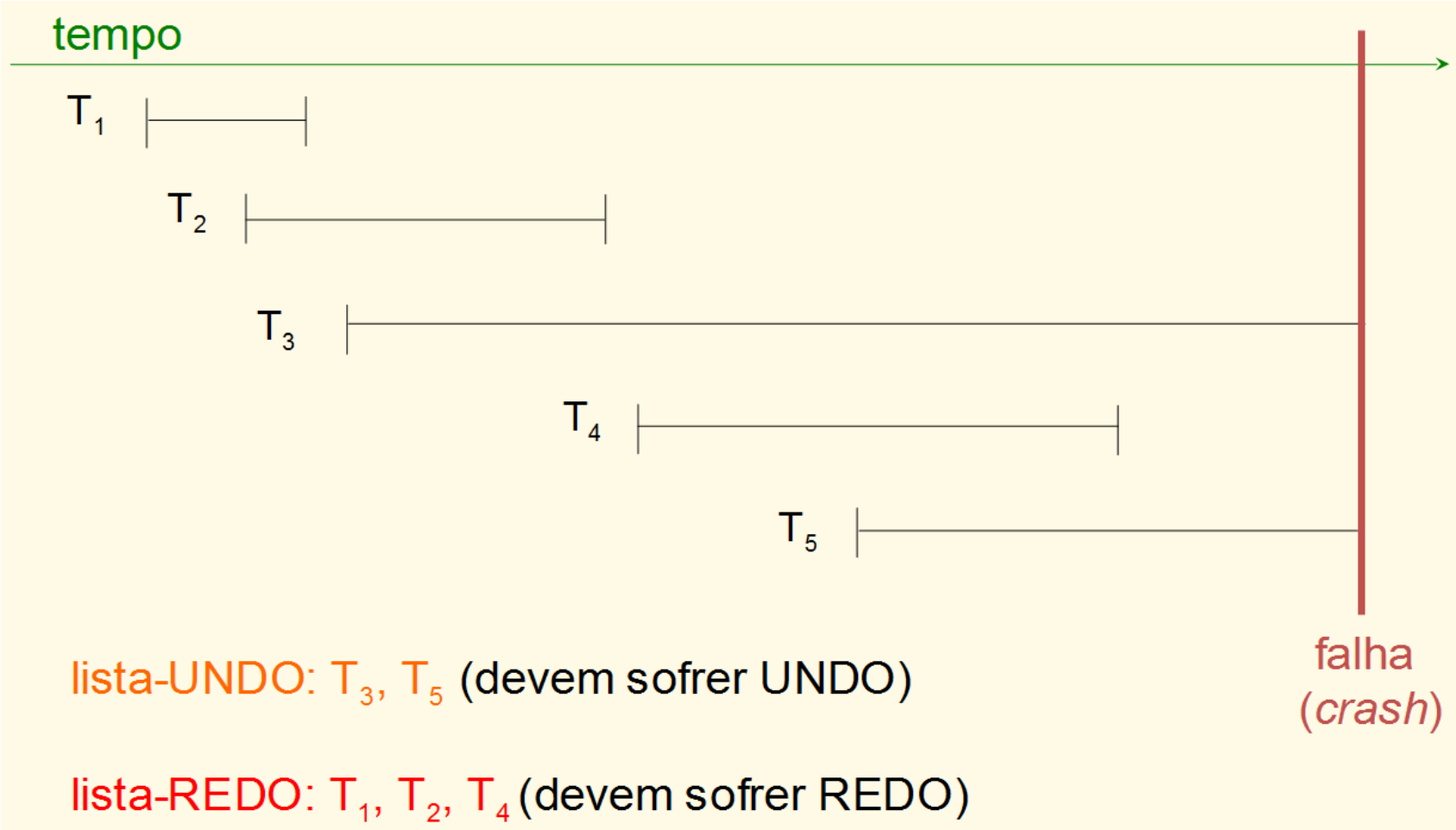
>> porém, o gerenciamento de buffer é mais simples.

Procedimento de recuperação:

>> 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.

Técnica *UNDO/REDO*



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

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

Técnica *UNDO/REDO* com *Checkpoint*

Procedimento de recuperação (sofisticado ...) :

(1) 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.

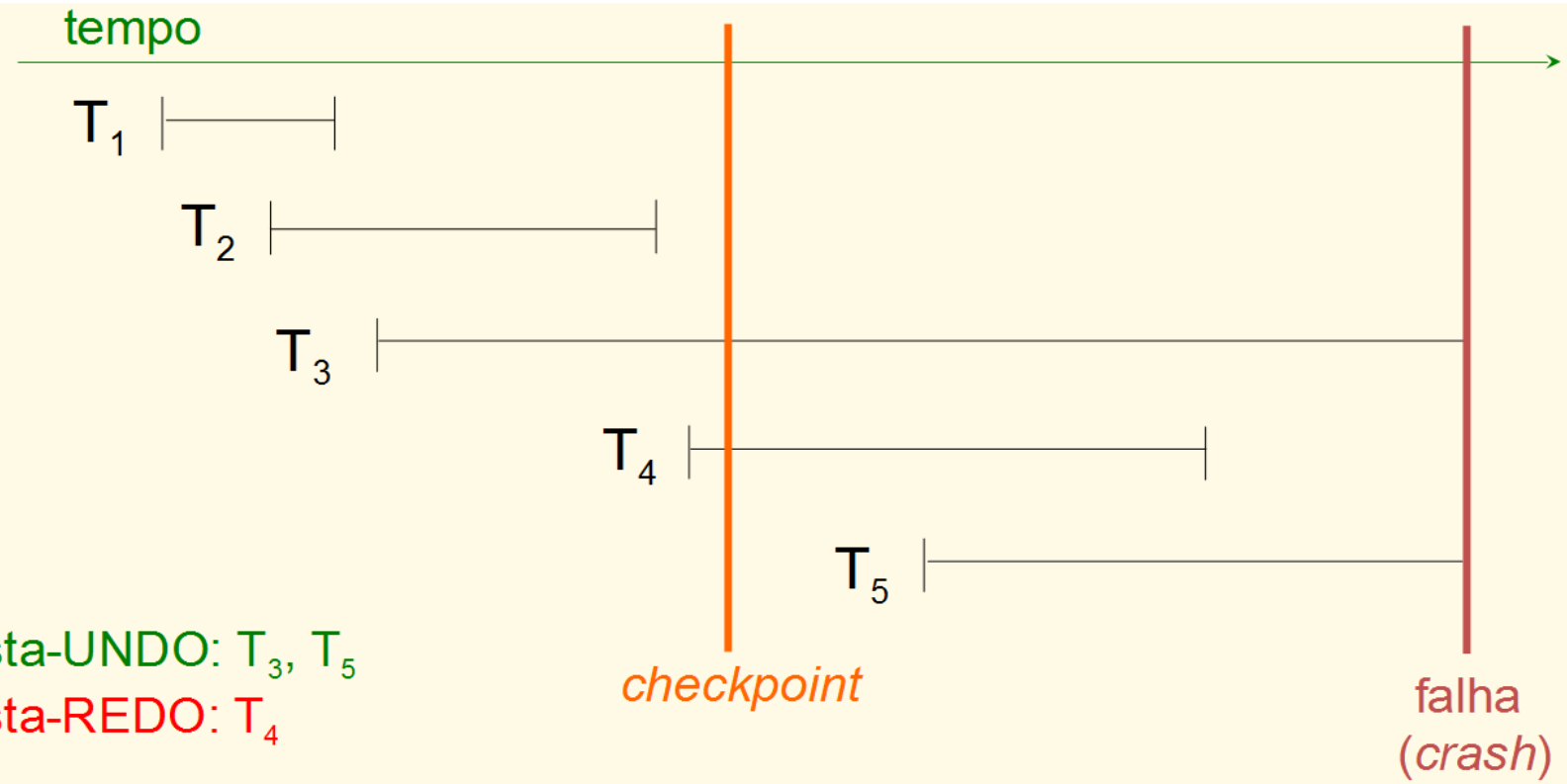
(2) analisa-se cada transação Tx no registro *checkpoint* :
>> se Tx não estiver na lista-REDO, insere Tx na lista-UNDO.

(3) 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.

(4) 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.

(5) percorre-se o *log forward* do ponto de parada, realizando REDO das transações na lista-REDO.

Técnica *UNDO/REDO* com *Checkpoint*



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

Técnica *UNDO/NO-REDO*

Técnica de recuperação em **atualização imediata**.

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>** estiver no *log*, **Tx** está garantidamente efetivada no BD;

Vantagem:

>> não há necessidade de fazer REDO.

Desvantagem:

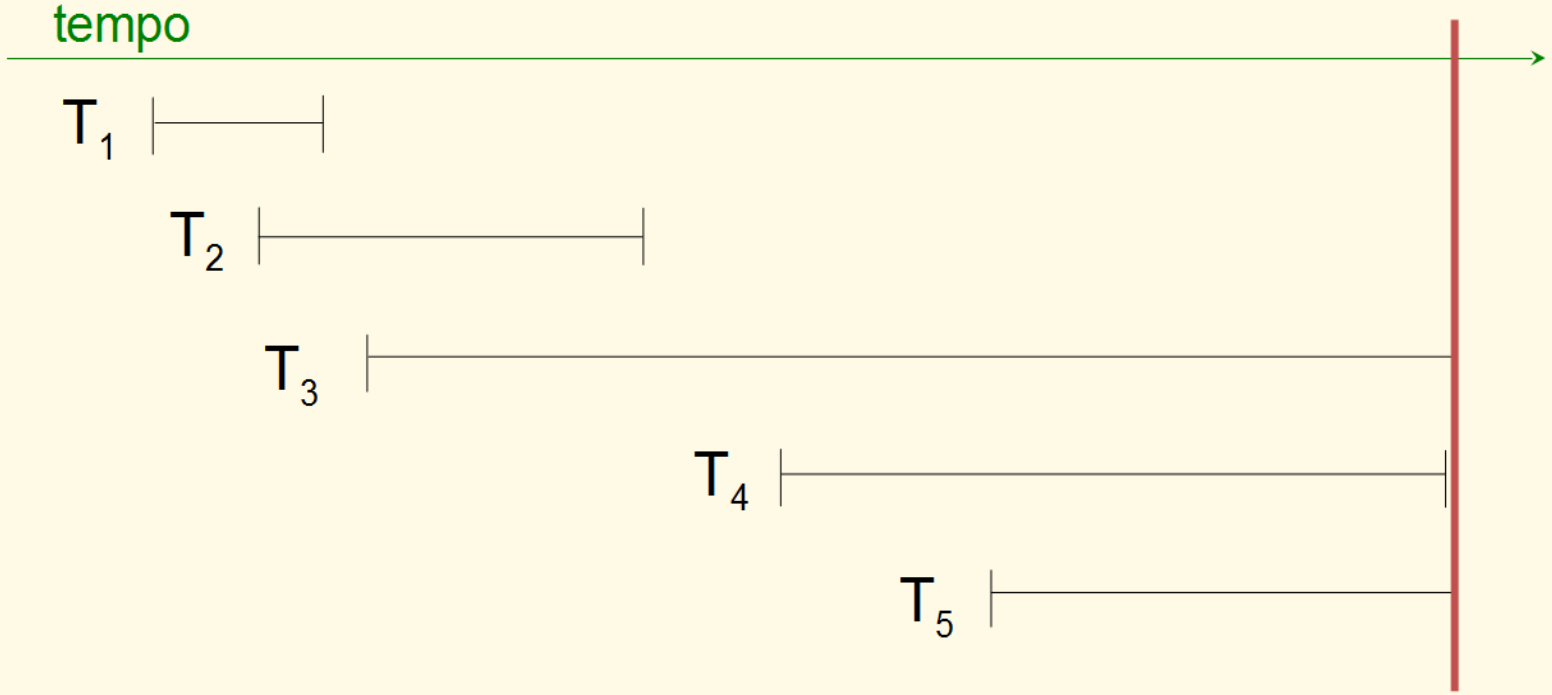
>> pode-se fazer UNDO de uma transação que foi gravada com sucesso no BD, porém não foi gravado a tempo o seu *commit* no *log*.

Requer *log* de UNDO.

Procedimento de recuperação:

>> faz uma varredura *backward* do *log*, realizando UNDO das transações na lista-UNDO (transações ativas).

Técnica *UNDO/NO-REDO*



lista-UNDO: T_3, T_4 e T_5

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

Técnica *NO-UNDO/REDO*

Modificação postergada do banco de dados:

>> 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 ***no-steal*** :

(i) blocos atualizados por Tx não podem ser "roubados" enquanto Tx não realizar *commit* ;

>> por outro lado, o *recovery* é mais simples :

(i) transações não precisam sofrer UNDO.

Técnica *NO-UNDO/REDO*

Quando Tx concluir suas atualizações, força-se a gravação do *log* em disco (com <commit Tx>) :

>> **force** no *log*;

Vantagem:

>> se Tx falhar antes de alcançar o *commit*, não é necessário realizar UNDO de Tx :

(i) nenhuma atualização de Tx foi gravada no BD, e

(ii) requer apenas um *log* de REDO.

Desvantagem:

>> *overhead* no tempo de processamento (***no-steal***), pois um bloco da *cache* pode permanecer em memória por muito tempo :

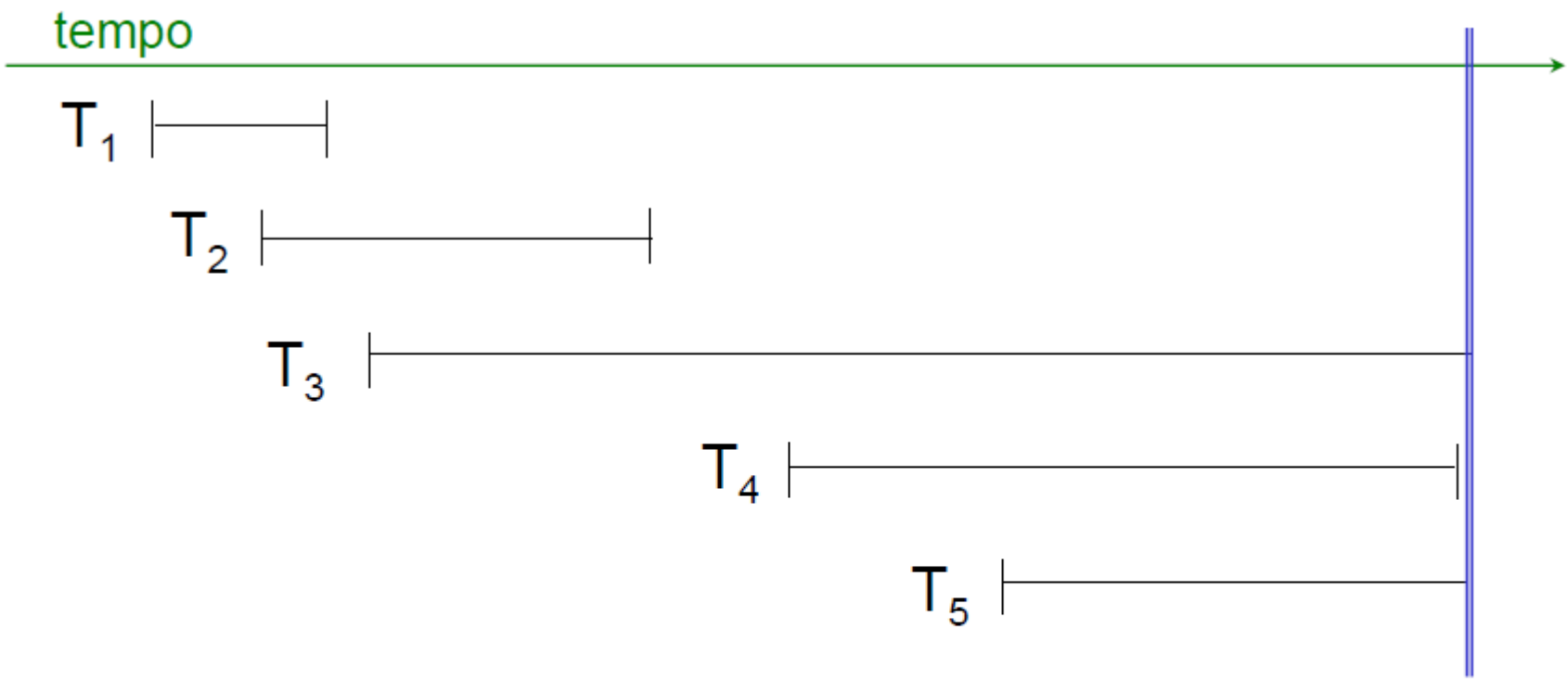
(i) dependente do *commit* de uma ou mais transações que atualizaram dados; e

(ii) se a cache ficar cheia, é possível que algumas transações requisitando dados do BD tenham que esperar pela liberação de blocos.

Procedimento de recuperação:

>> faz uma varredura ***forward*** do *log*, realizando REDO das transações na lista-REDO (transações *committed*).

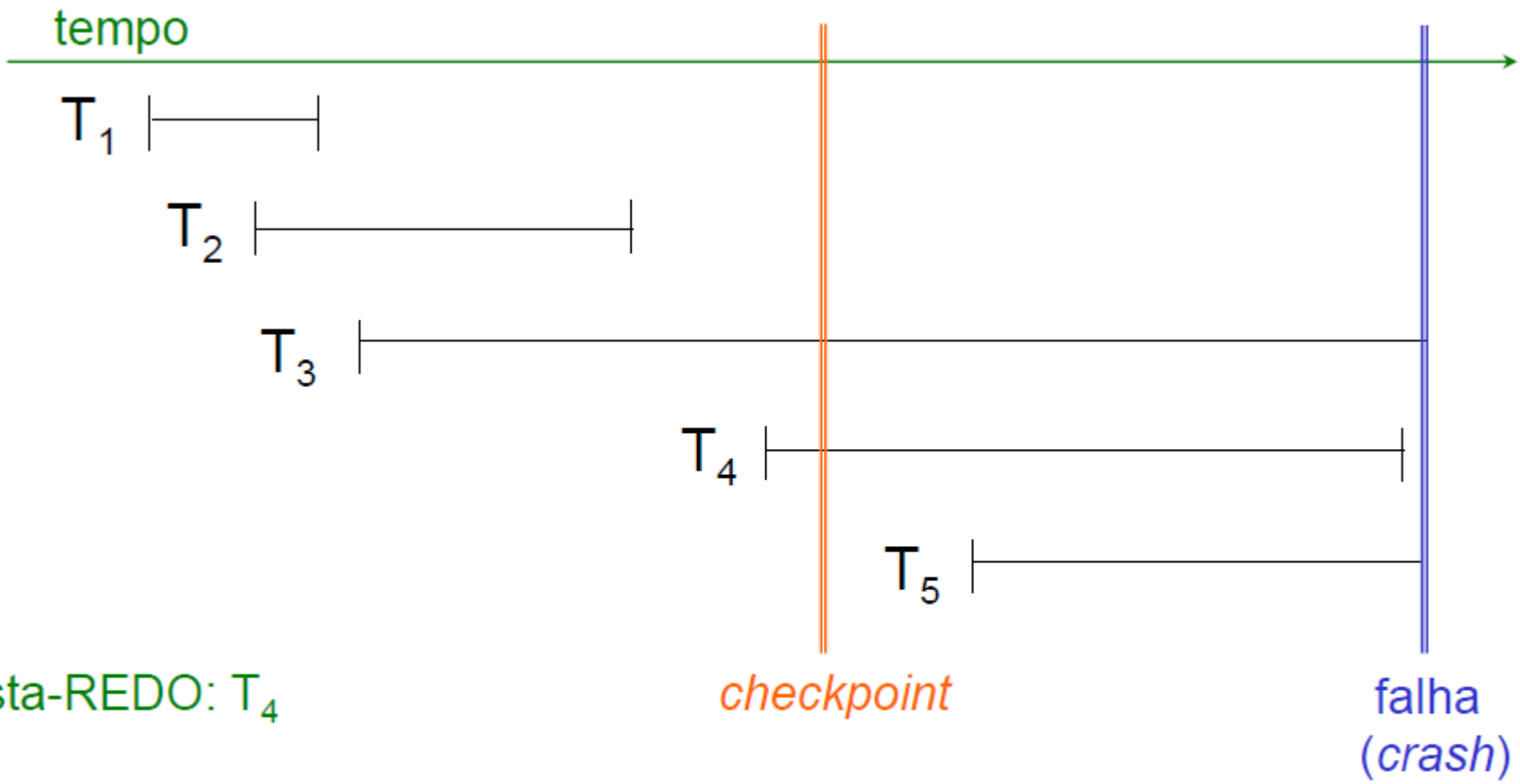
Técnica *NO-UNDO/REDO*



lista-REDO: T_1, T_2 e T_4

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

Técnica *NO-UNDO/REDO c/ Checkpoint*



- T_1 e T_2 concluíram antes do *checkpoint* ? não sofrem REDO
- T_4 concluiu depois do *checkpoint* ? sofre REDO
- T_3 e T_5 não concluíram e portanto não atualizaram o BD ? não sofrem UNDO

Técnica *ARCHIVE/DUMP/REDO*

Técnica baseada em *log* para recuperação de falha de meio de armazenamento.

Operação ARCHIVE (exemplo para descrição)

>> 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;
>> nenhuma transação pode estar ativa :
(i) se existem transações nesse estado, deve-se aguardar até elas encerrarem com sucesso;
>> o *log* corrente é "descartado" (excluído ou mantido associado ao backup anterior do BD) e um novo *log* ("zerado") é iniciado.

Técnica *ARCHIVE/DUMP/REDO*

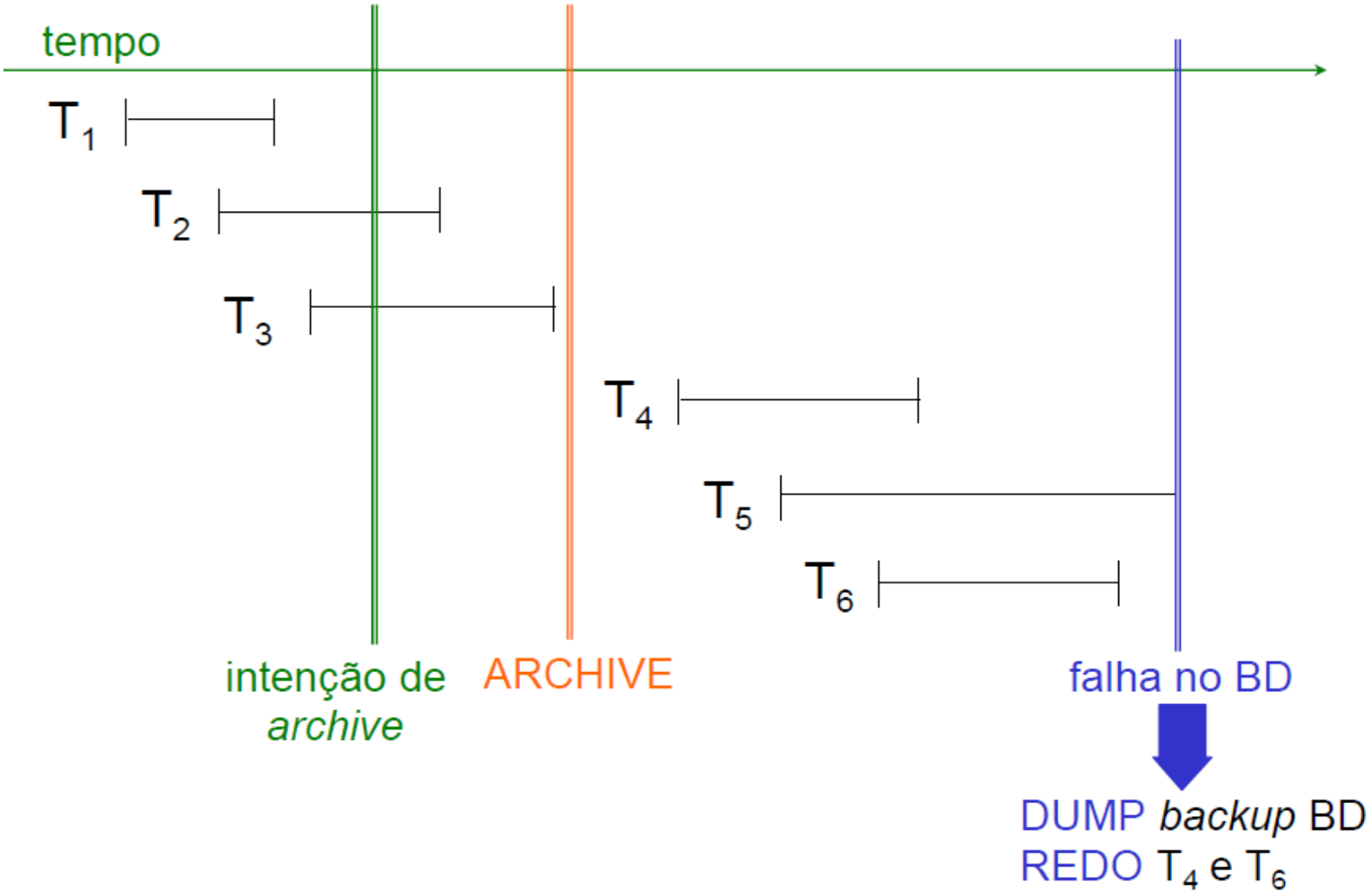
Operações DUMP + REDO

>> realizam a recuperação de uma falha no BD:
>> procedimento :
(i) restaura o BD a partir do último *backup* (DUMP);
(ii) realiza uma varredura *forward* do Log (REDO das transações *committed*).

Técnicas baseadas em *log* requerem um *log* seguro:

>> *archive* do *log* também deve ser realizado :
(i) com frequência igual ou superior ao *archive* do BD.

Técnica *ARCHIVE/DUMP/REDO*



Exercício

Que implicações uma política de gerenciamento de *buffer* ***no-steal/force*** teria sobre recuperação com *checkpoint* ?

No-steal significa que uma *buffer* modificado (na *cache*) por uma transação não pode ser escrito em disco até que a transação seja confirmada (*committed*).
Force significa que os *buffers* atualizados são escritos no disco quando uma transação é confirmada (*committed*).

Com ***no-steal***, o mecanismo de *checkpoint* não poderia escrever em disco os *buffers* modificados por transações ainda não confirmadas.

Com ***force***, uma vez que a transação é concluída, suas atualizações seriam transferidas para disco:

>> se uma falha ocorrer durante esse processo, REDO ainda será necessária;

>> entretanto, UNDO não será necessária, pois transação não confirmadas não serão propagadas ao disco.