Recuperação após Falha

Banco de Dados II

Cap. 18 (Ramakrishnan)
Cap. 23 (Elmasri)
Cap. 17 (Silberschatz)

Prof. Denio Duarte

Introdução

- A recuperação de falha significa que o BD é restaurado ao estado consistente mais recente antes do momento da falha
- SGBD precisar manter informações sobre as mudanças que estão sendo aplicadas aos itens pelas diversas transações
  - Armazenadas no log do sistema

Introdução

- Tipos de falhas
  - Transação
  - Sistema
  - Meio de armazenamento
  - Problemas físicos e catástrofes

Introdução

- Transação: uma transação ativa termina de forma anormal
  - Exemplos: violação de Restrição de Integridade, lógica da transação mal definida, deadlock ou cancelamento pelo usuário.
    - Não compromete a memória principal nem a memória secundária (disco, em geral)
    - Falha de maior probabilidade de ocorrência
    - Tempo de recuperação pequeno

Introdução

- Sistema: o SGBD encerra a sua execução de forma anormal
  - Interrupção de energia, falha no SO, erro interno no
  - SW do SGBD, falha de HW, por exemplo
    - Compromete a memória principal mas não compromete o disco
    - Falha com probabilidade média de ocorrência
    - Seu tempo de recuperação é médio

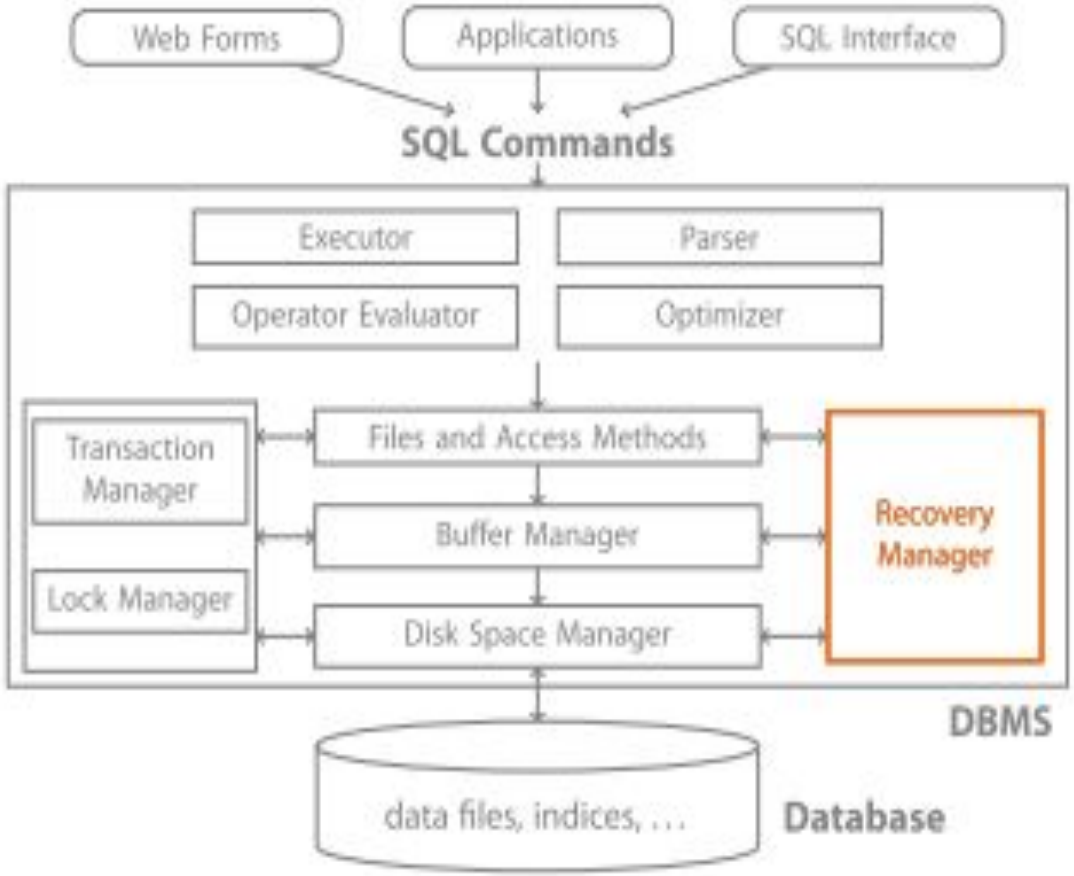
Introdução

- Meio de armazenamento: o BD se torna total ou parcialmente inacessível
  - Setores corrompidos no disco, falha no cabeçote de leitura/gravação, entre outras
    - Não compromete a memória principal mas compromete o disco
    - Falha com menor probabilidade de ocorrência
    - Seu tempo de recuperação é grande

Introdução

- Problemas físicos e catástrofes: erros do ambiente
  - Falha de energia, ar condicionado, incêndio, alagamento, roubo
    - Falha com menor probabilidade de ocorrência
    - Seu tempo de recuperação é grande

Gerenciador de Recuperação



Recuperação de Falhas

- Garante a atomicidade e a durabilidade das transações
  - Um SGBD deve ser tolerante a falhas
- Tolerância a falhas
  - Capacidade de retornar 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 é um mecanismo 100% seguro
  - Responsabilidade do subsistema de recuperação (*recovery)* do SGBD

Recuperação

- Estratégia básica
  - Caso dano extensivo (catástrofe, falha disco): restaura backup e reconstrói um estado mais recente, reaplicando ou refazendo as operações das transações no log.
  - Caso dano não extensivo (não houve falha no disco, etc): refazer as operações das transações efetivadas e desfazer as operações das transações abortadas

Subsistema de Recuperação

- Controles
  - Durante a operação do SGBD
    - Manter informações sobre quais itens do BD foram atualizados
    - Realizar cópias periódicas do BD
  - Após a falha
    - Executar ações para retornar o BD a um estado consistente
    - Ações básicas
      - Undo: desfazer uma atualização
      - Redo: refazer uma atualização

Subsistema de Recuperação

- As técnicas de recuperação (dano não extensivo) podem ser baseadas em
  - Reescrita no disco
  - Cópia dos dados (paginação de sombra - shadowing)

Subsistema de Recuperação

- Técnica de reescrita no disco
  - Baseada em log
  - Estratégias
    - Atualização adiada (deferred update)
    - Atualização imediata (immediate update)

Subsistema de Recuperação

- Atualização adiada (deferred update)
  - Não atualiza fisicamente o BD até que a transação atinja o commit
  - Atualização são registradas no espaço de trabalho da transação ou nos buffers (através dos logs)
  - Quando a transação entra em *Parcialmente Efetivada* as atualizações do log são efetivadas

Subsistema de Recuperação

- Atualização adiada (deferred update)
  - Em caso de falha antes do fim (system crash ou aborto), as informações no log são ignoradas
    - Não é necessário o UNDO das ações mas pode ser preciso o REDO
  - Algoritmo de recuperação, neste caso, é chamado NO-UNDO/REDO

Subsistema de Recuperação

- Atualização adiada (deferred update)
  - Funcionamento
    - Antes de T iniciar sua execução, a tupla `<T,start>` é registrada no log
    - A cada write(item), uma tupla `<T,item,novo valor>` é registrada também
      - Como UNDO é desnecessário, a tupla contém apenas o novo valor do item
    - Ao fim da transação, registra `<T,commit>`
    - Antes de escrever o valor do item no banco de dados (`output(item)`) o registro do log deve ser gravado em um meio estável de armazenamento

Subsistema de Recuperação

- Atualização adiada (deferred update)

T_0 (A=100,B=200)	T_1 (C=700)
read(A)	read(C)
A=A-50	C=C-100
write(A)	write(C)
read(B)	
B=B+50	
write(B)	

log
(T_0->T_1)
<T_0,start>
<T_0,A,50>
<T_0,B,250>
<T_0,commit>
<T_1,start>
<T_1,C,600>
<T_1,commit>

Subsistema de Recuperação

- Atualização adiada (deferred update)
  - Em caso de falha o procedimento `redo(T)` é executado e todos os novos itens da transação T são confirmados no banco
    - `redo(T)` deve ser idempotente, ou seja, pode ser executada várias vezes para a mesma transação mas o resultado deve ser sempre o mesmo
  - Uma transação T' é refeita se possuir as tuplas `<T',start>` e `<T',commit>` no log.

Subsistema de Recuperação

- Atualização adiada (deferred update)

	log (T_0-> T_1)	
<T_0,start> <T_0,A,50> <T_0,B,250> **crash**	<T_0,start> <T_0,A,50> <T_0,B,250> <T_0,commit> <T_1,start> <T_1,C,600> **crash**	<T_0,start> <T_0,A,50> <T_0,B,250> <T_0,commit> <T_1,start> <T_1,C,600> <T_1,commit> **crash**

Subsistema de Recuperação

- Atualização adiada (deferred update)

log (T_0-> T_1)		
<T_0,start>	<T_0,start>	<T_0,start>
<T_0,A,50>	<T_0,A,50>	<T_0,A,50>
<T_0,B,250>	<T_0,B,250>	<T_0,B,250>
crash	<T_0,commit>	<T_0,commit>
	<T_1,start>	<T_1,start>
	<T_1,C,600>	<T_1,C,600>
	crash	<T_1,commit>
		crash
Nada a fazer.	redo(T_0)	redo(T_0)
log atualizado.	Log atualizado.	redo(T_1)

Subsistema de Recuperação

- Atualização imediata (immediate update)
  - Pode atualizar o BD antes da transação atingir o commit (atualizações não comitadas)
  - As operações são registradas no log
  - Se uma transação falhar antes do commit, seus efeitos devem ser desfeitos (UNDO)
  - Se houver uma falha após a commit, os efeitos da transação devem ser refeitos (REDO)
  - Algoritmo de recuperação UNDO/REDO

Subsistema de Recuperação

- Atualização imediata (immediate update)
  - Funcionamento
    - Antes de T iniciar sua execução, `<T,start>` é registrada no log
    - A cada write(item), uma tupla `<T,item,valor antigo, novo valor>` é registrada também
      - Como UNDO é necessário, a tupla deve conter o valor antigo do item
    - Ao fim da transação: `<T,commit>` é registrada
    - Antes de escrever o valor do item no banco de dados (`output(item)`) o registro do log deve ser gravado em um meio estável de armazenamento

Subsistema de Recuperação

- Atualização imediata (immediate update)

T_0 (A=100,B=200)	T_1 (C=700)
read(A)	read(C)
A=A-50	C=C-100
write(A)	write(C)
read(B)	
B=B+50	
write(B)	

log (T_0-> T_1)

<T_0,start>
<T_0,A,100,50>
<T_0,B,200,250>
<T_0,commit>
<T_1,start>
<T_1,C,700,600>
<T_1,commit>

Subsistema de Recuperação

- Atualização imediata (immediate update)
  - Em caso de falha os procedimentos `redo(T)` e `undo(T)` são executados e todos os novos itens da transação T são confirmados no banco ou os antigos retornados, respectivamente.
    - `undo(T)` deve ser idempotente, ou seja, pode ser executada várias vezes para a mesma transação mas o resultado deve ser sempre o mesmo
    - `redo(T)` deve ser idempotente, ou seja, pode ser executada várias vezes para a mesma transação mas o resultado deve ser sempre o mesmo

Subsistema de Recuperação

- Atualização imediata (immediate update)
  - Uma transação T' é refeita se possuir as tuplas `<T',start>` e `<T',commit>` no log.
  - Uma transação T' é desfeita se possuir a tupla `<T',start>` e não a tupla `<T',commit>`.

Subsistema de Recuperação

- Atualização imediata (immediate update)

	log (T_0-> T_1)	
<T_0,start> <T_0,A,100,50> <T_0,B,200,250> **crash**	<T_0,start> <T_0,A,100,50> <T_0,B,200,250> <T_0,commit> <T_1,start> <T_1,C,700,600> **crash**	<T_0,start> <T_0,A,100,50> <T_0,B,200,250> <T_0,commit> <T_1,start> <T_1,C,700,600> <T_1,commit> **crash**

Subsistema de Recuperação

- Atualização imediata (immediate update)

	log (T_0-> T_1)	
<T_0,start>	<T_0,start>	<T_0,start>
<T_0,A,100,50>	<T_0,A,100,50>	<T_0,A,100,50>
<T_0,B,200,250>	<T_0,B,200,250>	<T_0,B,200,250>
crash	<T_0,commit>	<T_0,commit>
	<T_1,start>	<T_1,start>
	<T_1,C,700,600>	<T_1,C,700,600>
	crash	<T_1,commit>
		crash
undo(T_0)	redo(T_0) e undo(T_1)	redo(T_0)
log atualizado	Log atualizado.	redo(T_1)

CheckPoint

- Quando ocorre uma falha é necessário percorrer o log para procurar transações para realizar o undo ou redo
  - O processo de procura consome tempo
  - A maioria das transações que precisariam ser refeitas já tiveram seus dados gravados no banco (overhead)
- Checkpoints resolvem esses problemas

CheckPoint

- Checkpoints
  - O sistema, periodicamente, executa checkpoints
    - Coloca em memória estável todos os registros que estão na memória RAM
    - Coloca no disco todos os buffers modificados
    - Coloca a tupla <checkpoint> no log
  - Quando a ação está sendo executada, nenhuma outra ação no banco pode ocorrer
  - O sistema ao encontrar um <checkpoint> “sabe” que as ações anteriores já foram confirmadas

CheckPoint

```
<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>
<checkpoint>
<write T3,A,10,19>
<commit T3>
CRASH
```

Paginação de Sombra (Shadow Paging)

- Técnica alternativa para recuperação que não necessita de log
- O BD é considerado como uma série de páginas no disco (ou blocos) de tamanho fixo
  - Um diretório com **n** entradas (n=número de páginas) no qual a **i-ésima** entrada aponta para a **i-ésima** página
    - Quando uma transação começa a ser executada, o diretório atual é copiado para um diretório sombra (que ficará em um meio não volátil)

Paginação de Sombra (Shadow Paging)

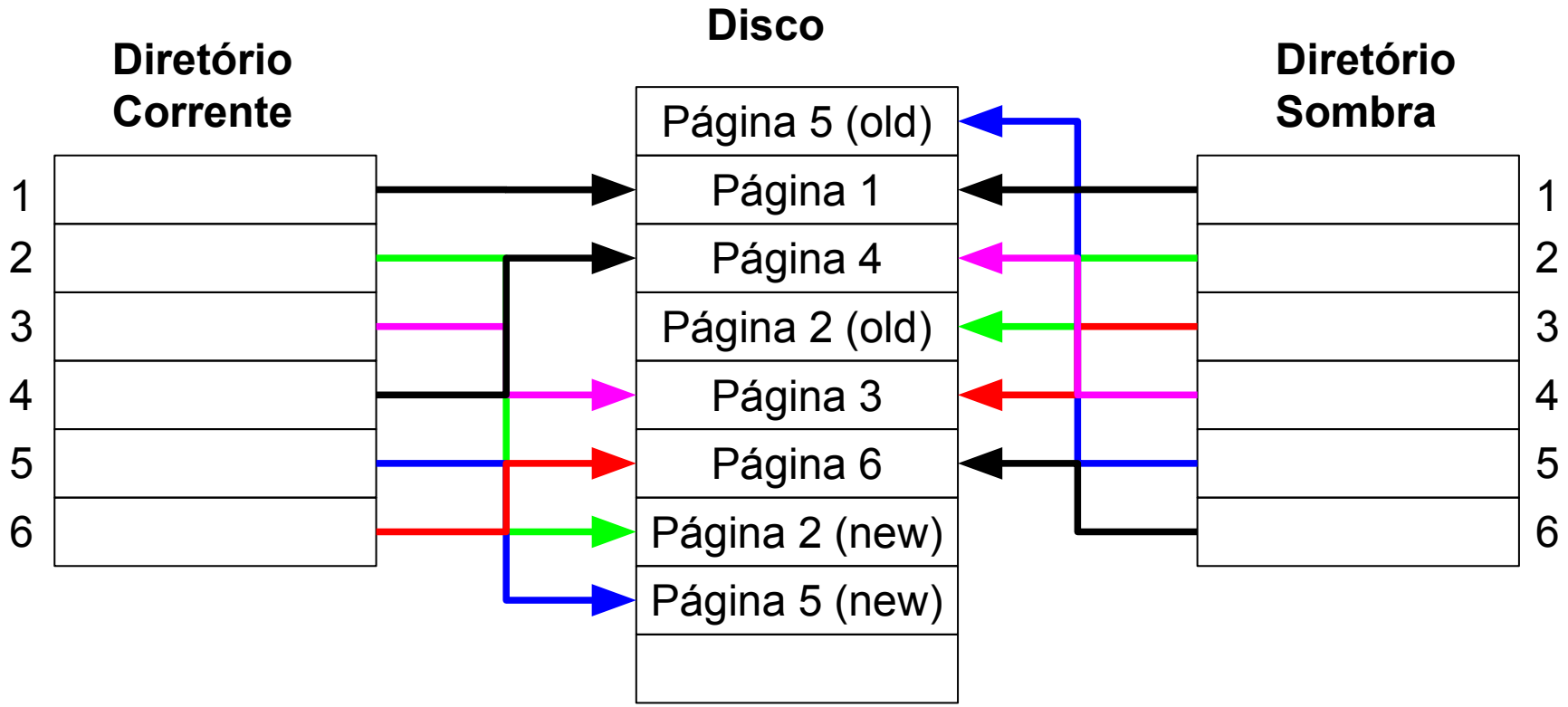
- Funcionamento
  - Durante a execução da transação, o diretório sombra não pode ser atualizado
  - Quando um write(X) é executado, a página onde **X** está e copiada e o diretório atual apontará para a página com **X** modificado e o diretório sombra apontará para a página com **X** original
  - O undo é feito copiando as cópias apontados pelo diretório sombra
  - O redo é feito ignorando as cópias do diretório sombra

Paginação de Sombra (Shadow Paging)

- Funcionamento
  - Uma transação **T'** executa *write(X)* e *X* está na **i-ésima** página:
    - Se a **i-ésim**a página não está no buffer, o sistema copia para o buffer
    - Se é o primeiro write executado na i-ésima página pela transação, o sistema o modifica o diretório corrente:
      - Procura uma página livre no disco
      - Cópia a **i-ésima** página para a página livre (torna-se ocupada)
      - Aponta o diretório corrente para a nova página
      - O diretório sombra aponta para a página antiga

Paginação de Sombra (Shadow Paging)

- Funcionamento



Paginação de Sombra (Shadow Paging)

- Vantagens sobre log
  - Overhead na gravação do log é eliminado
  - Recuperação de falha é mais rápida (pois não tem undo nem redo)
- Desvantagens
  - O commit de cada transação exige a atualização das página (commit overhead)

Paginação de Sombra (Shadow Paging)

- Desvantagens (cont …)
  - Fragmentação: as páginas do banco mudam frequentemente de lugar
  - Páginas do diretório fantasma devem ser liberadas
- A técnica que utiliza log é conhecida como Atualização no Local

Vamos complicar?
Transações Concorrentes

Recuperação com Escalonamentos Concorrentes

- Um único log para as transações
  - As entradas no log são compartilhadas
- Um único buffer
  - As páginas do buffer são compartilhadas pelas transações

Recuperação com Escalonamentos Concorrentes

- O esquema de recuperação depende do esquema de controle de concorrência
  - **T'** executa write(X) e é abortada (o valor antigo de X deve ser restaurado)
  - **T''** executa write(X) sobre o X atualizado por **T'**, como **T'** abortou **T''** terá que abortar também
  - 2PL evita esse problema

Rollback de Transações

- Uma transação **T** é cancelada utilizando o log
- O sistema procura (do fim para o início) as tuplas *<T,X,Va, Vn>*, e restaura ***X*** com o valor antigo ***Va***. A busca para quando encontrar *<T,start>*
- A busca do fim para o início evita inconsistências
  - <T,X,10,20>
  - <T,X,20,30>
  - O valor correto de **X** é **10**.
- Devido ao 2PL não é necessário realizar o rollback em cascata

Checkpoints

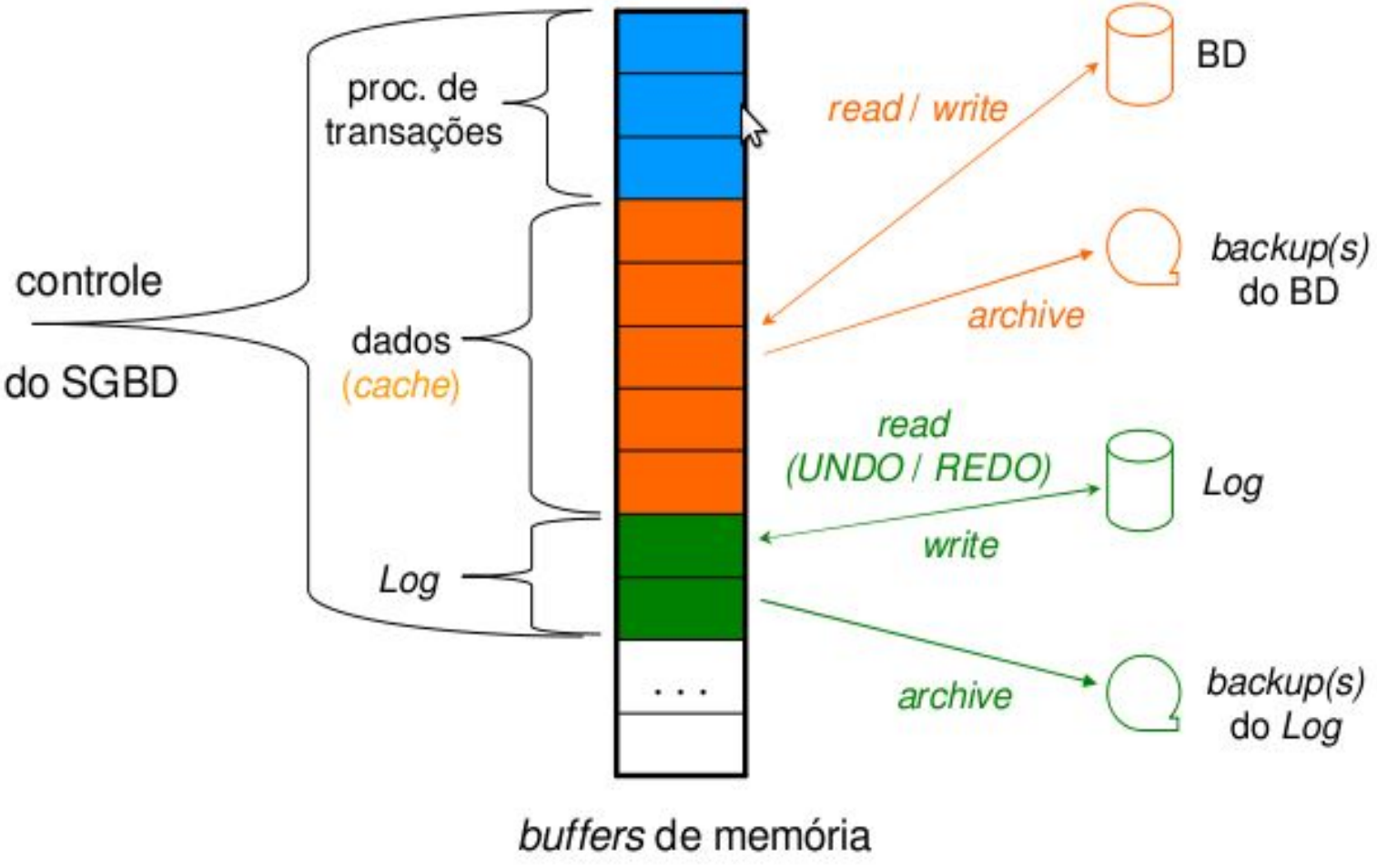
- No caso sem concorrência era considerado
  - Apenas as transações que que haviam começado após o checkpoint mais recente
  - Apenas uma transação poderia estar ativa durante um checkpoint
- Para o caso de concorrência a tupla do checkpoint deve ser modificado
  - <checkpoint, L>, onde **L** é a lista de transações efetivadas até o checkpoint.

Buffer

Gerenciamento de Buffer

- Buffer
  - Conjunto de blocos (páginas) da memória principal
- O SGBD gerencia os buffers
  - De dados
  - De Logs
  - Para processamento de transações
- Solicita ao SO apenas os serviços de escrita/leitura dos blocos

Gerenciamento de Buffer



Fonte: Prof. Ronaldo (UFSC)

Gerenciamento de Buffer

- Os buffers de dados e log devem ser sincronizados
  - Um bloco atualizado na cache pode ser gravado apenas no BD **após** o histórico dos dados atualizados neste bloco ter sido gravado no Log em disco
    - Técnica chamada Write-Ahead Log (WAL)
  - Uma transação T' só pode passar para o estado efetivada (committed) após todas as suas atualizações terem sido gravadas no BD segundo o princípio WAL
- O gerenciamento, aqui apresentado, é utilizado na técnica Atualização no Local

Técnicas

- STEAL (Roubo)
  - O gerenciador de buffer pode roubar a página do banco
    - Pode escrever uma página arbitrária no disco e utilizar tal página para outra coisa do disco
    - O SGBD não controla a política de troca de páginas do buffer
    - A página pode conter escritas feitas por transações que não comitaram
    - Pode ser utilizado por razões de desempenho e não há necessidade de manter blocos bloqueados por transações

Técnicas

- NO-STEAL (sem roubo)
  - Um bloco na cache utilizado por uma transação **T'** não pode ser gravado antes do commit de **T'**
    - O bloco possui um bit de status (dirty bit) indicando se foi (1) ou não (0) modificado
  - vantagem: processo de recovery mais simples - evita dados de transações inacabadas sendo gravadas no BD

Técnicas

- FORCE (Forçar)
  - O SGBD força todas as atualizações das transações a serem gravadas no disco antes da efetivação das mesmas
  - Os blocos que mantêm dados atualizados por uma transação T' são imediatamente gravados no BD
  - quando T' é efetivada (após o commit)
    - Deve-se saber quais os blocos que T' atualizou dados
  - Vantagem: garante a durabilidade de T' o mais cedo possível - permite o REDO de T' em caso de falha

Técnicas

- NO-FORCE (Sem Forçar)
  - Os blocos que mantêm dados atualizados por **T'** não são imediatamente gravados no BD quando **T'** é efetivada (após o commit)
  - Vantagem: blocos atualizados podem permanecer na cache e serem utilizados por outras transações, após a efetivação de **T'** (reduz custo de acesso a disco)

Técnicas

- Política preferida STEAL/NO-FORCE
  - Esta combinação é mais complicada porém traz os melhores desempenhos
  - STEAL (difícil implementar a durabilidade)
    - Uma transação que gravou em disco abortou
    - O sistema pode falhar antes da transação finalizar
    - Deve lembrar o valor antigo de um item (para suportar o UNDO)

Técnicas

- Política preferida STEAL/NO-FORCE
  - NO-FORCE (difícil implementar a atomicidade)
    - Uma transação que gravou em disco abortou
    - O sistema pode falhar antes da transação finalizar
    - Deve lembrar o valor antigo de um item (para suportar o UNDO)

Gerenciamento (Sumário)

	No-Steal	Steal
No-Force		Mais Rápido
Force	Mais Lento	

Desempenho

	No-Steal	Steal
No-Force	Sem Undo Redo	Undo Redo
Force	Sem Undo e Redo	Undo Sem Redo

Implicação na recuperação e no log

Leitura

- Chapter 18 Crash Recovery (Database Management Systems - third edition – Ramakrishnan & Gerhke)
  - Verificar o capítulo equivalente na versão em português.