Ciência da Computação

GBC043 Sistemas de Banco de Dados

Processamento de Transações
Recuperação de Banco de Dados

Profa. Maria Camila Nardini Barioni

camila.barioni@ufu.br

Bloco B - sala 1B137

1° semestre de 2024

DISCUSSÃO LAB

Atividade em lab

Modelo Relacional:

CANDIDATO (SG_UF, NM_UE, CD_CARGO (CARGO.CD_CARGO), SQ_CANDIDATO, NR_CANDIDATO, NM_CANDIDATO, NM_URNA_CANDIDATO, NM_SOCIAL_CANDIDATO, TP_AGREMIACAO, NR_PARTIDO (PARTIDO.NR_PARTIDO), NR_FEDERACAO (FEDERACAO.NR_FEDERACAO), SQ_COLIGACAO (COLIGACAO.SQ_COLIGACAO), SG_UF_NASCIMENTO, DT_NASCIMENTO, CD_GENERO (GENERO.CD_GENERO), CD_GRAU_INSTRUCAO (GRAU_INSTRUCAO.CD_GRAU_INSTRUCAO), CD_ESTADO_CIVIL (ESTADO_CIVIL.CD_ESTADO_CIVIL), CD_COR_RACA (COR_RACA.CD_COR_RACA), CD_OCUPACAO (OCUPACAO.CD_OCUPACAO));

PARTIDO (NR_PARTIDO, SG_PARTIDO, NM_PARTIDO);

Atividade Lab

/* (b) Estava previsto no MER ter um campo com a quantidade de candidatos inscritos por partido na eleição na tabela PARTIDO (atributo derivado). (a) Incluir esse novo atributo na tabela PARTIDO. (b) faça um procedimento armazenado para atualizar o valor deste atributo. (c) crie um gatilho que cuide da integridade deste campo – ou seja, que garanta que o valor ali presente corresponda sempre ao número real de candidatos registrados por partido (não pode chamar o procedimento criado em (b), pois seria muito ineficiente). (d) Realize inserções e deleções no banco de dados para observar o disparo do gatilho (mostrar os resultados).*/

Atividade Lab

```
/* (b) */

-- Item A
ALTER TABLE PARTIDO
    ADD COLUMN TOTAL_CANDIDATOS INTEGER;
```

Atividade Lab

```
/* (b) *  Item B */
CREATE OR REPLACE FUNCTION AtualizaCandidatoPartido()
RETURNS VOID AS $$
DECLARE AUX PARTIDO%rowtype;
BEGIN
   FOR AUX IN SELECT NR_PARTIDO FROM PARTIDO
        LOOP
             UPDATE PARTIDO SET TOTAL_CANDIDATOS =
                  (SELECT COUNT(SQ_CANDIDATO) FROM CANDIDATO
                   WHERE NR_PARTIDO = AUX.NR_PARTIDO)
            WHERE NR_PARTIDO = AUX.NR_PARTIDO;

    END LOOP;
END $$ LANGUAGE 'plpgsql';

SELECT * FROM AtualizaCandidatoPartido()
```

Atividade Lab

```
/* (b) *  Item C */
CREATE OR REPLACE FUNCTION VERIFICA_CANDIDATO_PARTIDO_SP()
RETURNS trigger AS $$
BEGIN
 IF (TG_OP = 'DELETE') THEN
   UPDATE PARTIDO SET TOTAL_CANDIDATOS = TOTAL_CANDIDATOS - 1 WHERE
   NR_PARTIDO = old.NR_PARTIDO;
   RETURN OLD;
 ELSEIF (TG_OP = 'INSERT') THEN
   UPDATE PARTIDO SET TOTAL_CANDIDATOS = TOTAL_CANDIDATOS + 1 WHERE
   NR_PARTIDO = new.NR_PARTIDO;
   RETURN NEW;
 END IF;
END $$ language 'plpgsql';

CREATE TRIGGER VERIFICA_CANDIDATO_PARTIDO
BEFORE INSERT OR DELETE ON CANDIDATO
FOR EACH ROW EXECUTE PROCEDURE VERIFICA_CANDIDATO_PARTIDO_SP();
```

Atividade Lab

```
/* (b) *  Item C */
CREATE OR REPLACE FUNCTION VERIFICA_CANDIDATO_PARTIDO_SP()
RETURNS trigger AS $$
BEGIN
 IF (TG_OP = 'DELETE') THEN
   UPDATE PARTIDO SET TOTAL_CANDIDATOS = TOTAL_CANDIDATOS - 1 WHERE
   NR_PARTIDO = old.NR_PARTIDO;
   RETURN OLD;
 ELSEIF (TG_OP = 'INSERT') THEN
   UPDATE PARTIDO SET TOTAL_CANDIDATOS = TOTAL_CANDIDATOS + 1 WHERE
   NR_PARTIDO = new.NR_PARTIDO;
   RETURN NEW;
 END IF;
END $$ language 'plpgsql';

CREATE TRIGGER VERIFICA_CANDIDATO_PARTIDO
BEFORE INSERT OR DELETE ON CANDIDATO
FOR EACH ROW EXECUTE PROCEDURE VERIFICA_CANDIDATO_PARTIDO_SP();
```

Atividade Lab

/* (b) * **Item C** */

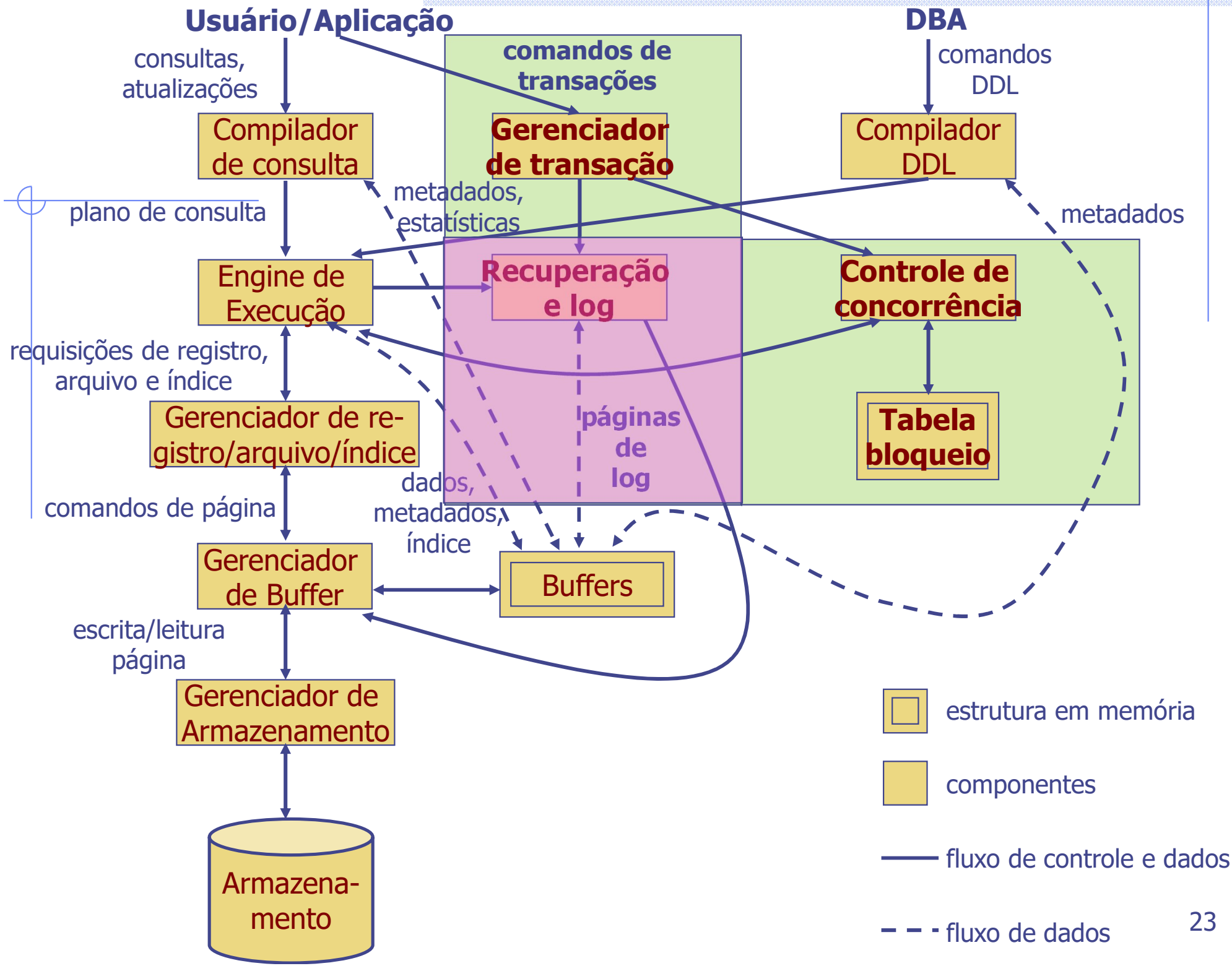
Para testar...

```
INSERT INTO CANDIDATO (SQ_CANDIDATO, NR_PARTIDO) VALUES (270002394752, 30);

SELECT NR_PARTIDO, COUNT(SQ_CANDIDATO)
FROM CANDIDATO
GROUP BY NR_PARTIDO

DELETE FROM CANDIDATO WHERE SQ_CANDIDATO = 270002394752;
```

TÉCNICAS DE RECUPERAÇÃO

Roteiro da aula

- Conceitos de Recuperação
- Técnicas de recuperação de falhas
  - Atualização adiada
  - Atualização imediata


Usuário/Aplicação
DBA
consultas, atualizações
comandos de transações
comandos DDL
Compilador de consulta
Gerenciador de transação
Compilador DDL
plano de consulta
metadados, estatísticas
metadados
Engine de Execução
Recuperação e log
Controle de concorrência
requisições de registro, arquivo e índice
Gerenciador de re-gistro/arquivo/índice
páginas de log
Tabela bloqueio
dados, metadados, índice
comandos de página
Gerenciador de Buffer
Buffers
escrita/leitura página
Gerenciador de Armazenamento
Armazena-mento
estrutura em memória
componentes
fluxo de controle e dados
fluxo de dados

Algoritmos de recuperação

- Algoritmos de recuperação são técnicas para garantir a **consistência** do banco de dados e a **atomicidade** e **durabilidade** da transação apesar das falhas.
- Algoritmos de recuperação têm duas partes:
  - Ações tomadas durante o processamento normal da transação para garantir que existem informações suficientes para recuperação de falhas
  - Ações tomadas após uma falha para recuperar o conteúdo do banco de dados a um estado que garante atomicidade, consistência e durabilidade

Relembrando...
Porque a recuperação é necessária

- O SGBD não deve permitir que algumas operações de uma transação *T* sejam aplicadas enquanto outras não

- *Tipos de falhas:*
  - 1- Falha de computador
  - 2- Erro de transação ou de sistema (ex.: divisão por zero, etc)
  - 3- Erros locais ou de condições de exceção detectados pelas transações (ex.: Saldo insuficiente)
  - 4- Imposição do controle de concorrência (Deadlock)
  - 5- Falha de disco
  - 6- Problemas físicos e catástrofes

Estrutura de armazenamento

- Armazenamento volátil:
  - não sobrevive a falhas do sistema
  - exemplos: memória principal, memória cache
- Armazenamento não volátil:
  - sobrevive a falhas do sistema
  - exemplos: disco, fita, memória flash,
    RAM não-volátil (alimentada por bateria)
- Armazenamento estável:
  - uma forma mítica de armazenamento que sobrevive a todas as falhas
  - aproximado mantendo-se várias cópias em meios não voláteis distintos

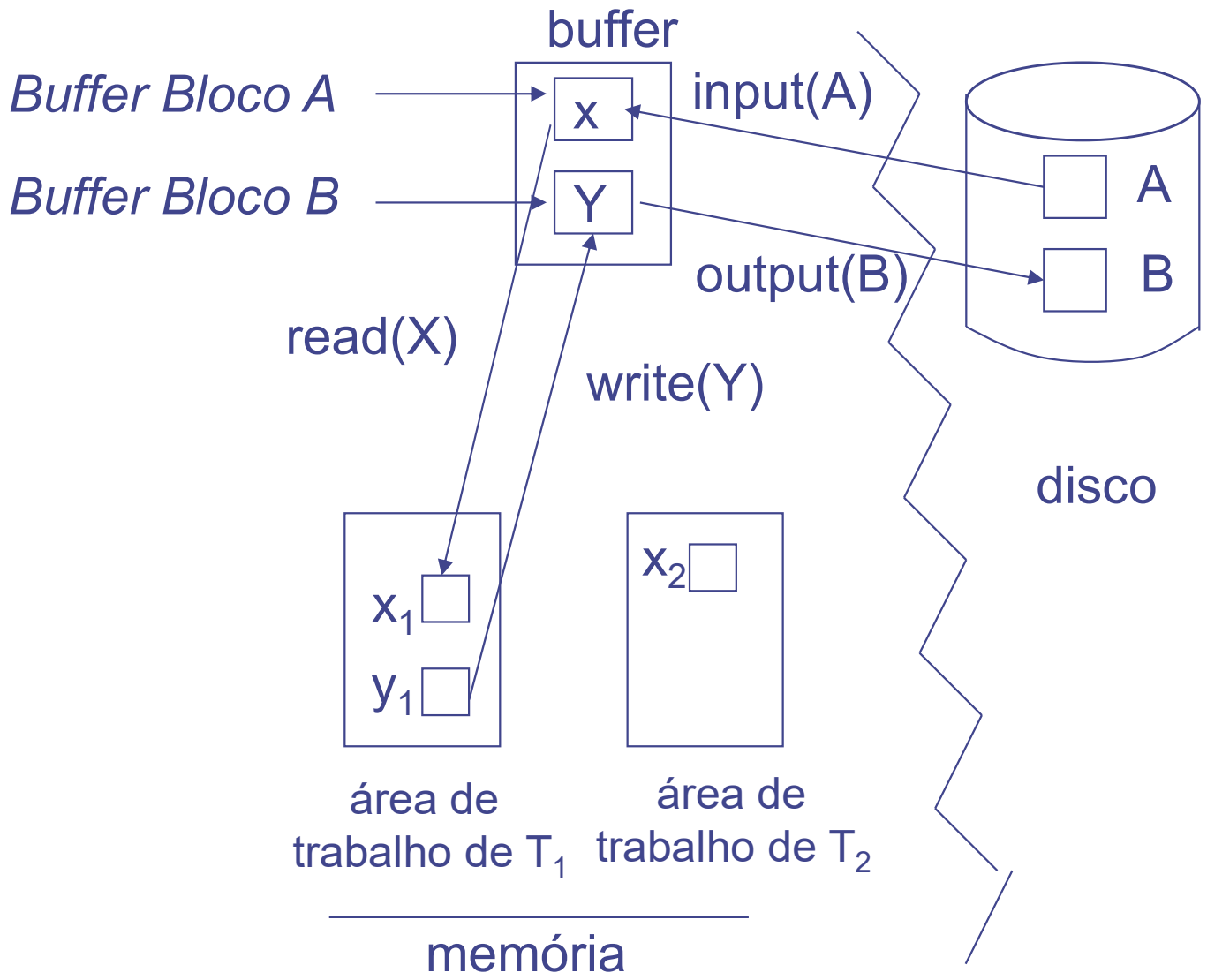
Acesso aos dados

- Blocos físicos são aqueles blocos residindo no disco.
- Blocos de buffer são os blocos residindo temporariamente na memória principal.
- Movimentos de bloco entre disco e a memória principal são iniciados por meio das duas operações a seguir:
  - input(*B*) transfere o bloco físico *B* para a memória principal.
  - output(*B*) transfere o bloco de buffer *B* para o disco, e substitui o bloco físico apropriado lá.
- Cada transação T_i possui sua área de trabalho privada, onde são mantidas as cópias locais de todos os itens de dados acessados e atualizados por ela.
  - A cópia local de T_i de um item de dados *X* é chamada de x_i.
- Consideramos, por simplicidade, que cada item de dados cabe e é armazenado dentro de um único bloco.

Acesso aos dados

- A transação transfere itens de dados entre os blocos de buffer do sistema e sua área de trabalho privada usando as seguintes operações:
  - read(X) atribui o valor do item de dados X à variável local x_i.
  - write(X) atribui o valor da variável local x_i ao item de dados {X} no bloco de buffer.
  - esses dois comandos podem precisar da emissão de uma instrução input(B_X) antes da atribuição, se o bloco B_X em que X reside ainda não estiver na memória.
- Transações
  - Execute read(X) enquanto acessa X pela primeira vez;
  - Todos os acessos subseqüentes são para a cópia local.
  - Após o último acesso, a transação executa write(X).
- output(B_X) não precisa vir imediatamente após write(X). **O sistema pode realizar a operação output quando julgar necessário.**

Exemplo de acesso aos dados



Recuperação e atomicidade

- Modificar o banco de dados sem garantir que a transação será confirmada pode levar o banco de dados a um estado inconsistente.
- Considere a transação T_i que transfere \$50 da conta *A* para a conta *B*; o objetivo é realizar todas as modificações do banco de dados feitas por T_i ou nenhuma delas.
- Várias operações de saída podem ser exigidas para T_i (para gerar *A* e *B*). Uma falha pode ocorrer após uma dessas modificações ter sido feita, mas antes que todas elas sejam feitas.

Recuperação e atomicidade

- Para garantir a atomicidade apesar das falhas, primeiro geramos informações descrevendo as modificações no armazenamento estável sem modificar o próprio banco de dados.
- Estudaremos duas técnicas:
  - recuperação baseada em log com atualização/modificação imediata
  - recuperação baseada em log com atualização/modificação adiada
- Consideramos (inicialmente) que as transações serão executadas em série, ou seja, uma após a outra.

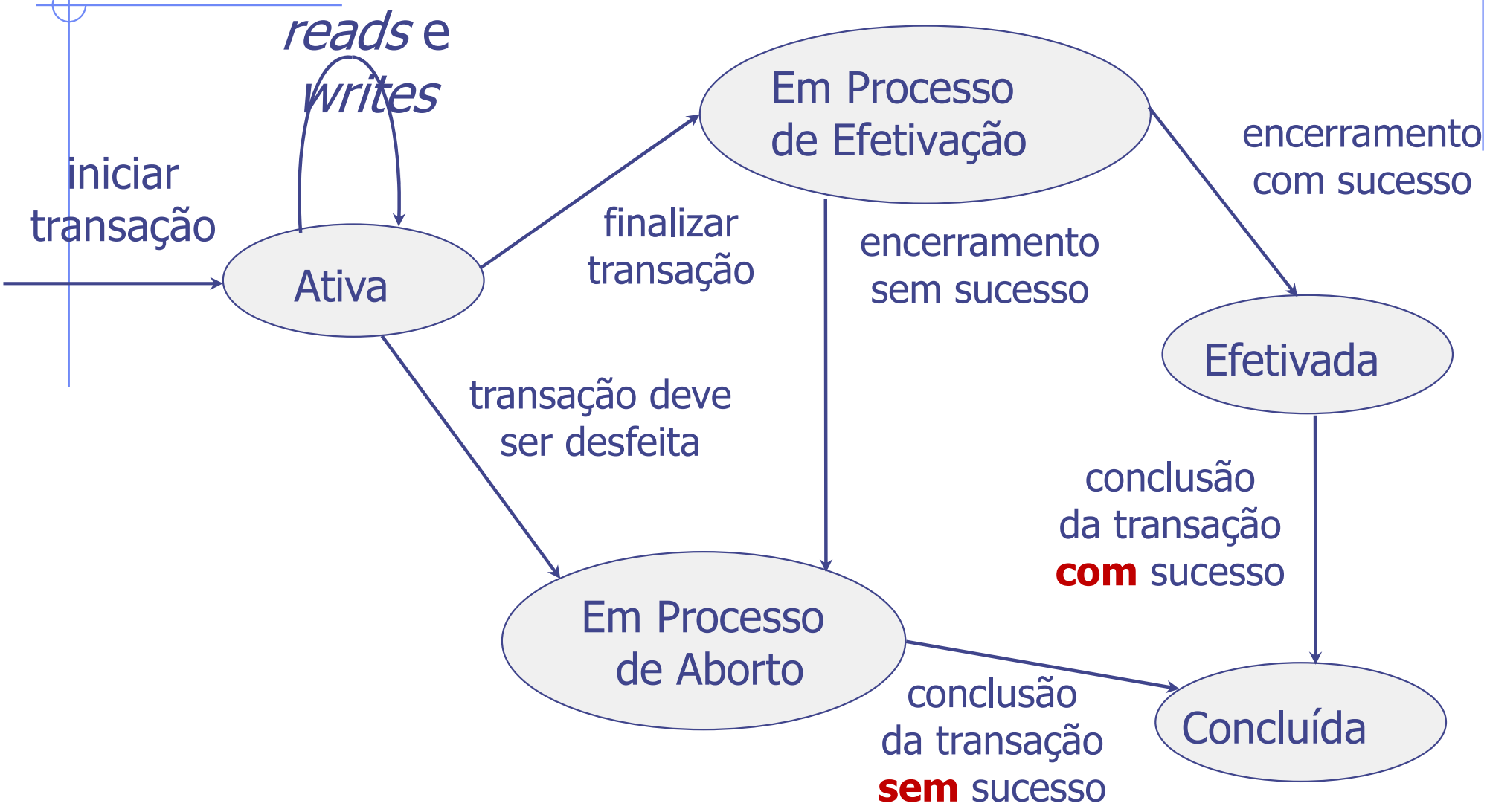
Recuperação baseada em log

- Um log é mantido no armazenamento estável.
  - O log é uma seqüência de registros de log, e mantém um registro das atividades de atualização no banco de dados.
- Quando a transação T_i inicia, ela se registra escrevendo um registro de log <T_i start>
- *Antes que* T_j execute write(X), um registro de log <T_i, X, V_1, V_2> é escrito, onde V_1 é o valor de X antes do write, e V_2 é o valor a ser escrito em X.
  - O registro de log observa que T_i realizou uma escrita no item de dados X_j. X_j tinha o valor V_1 antes da escrita, e terá o valor V_2 após a escrita.
- Quando T_i termina sua última instrução, o registro de log <T_i commit> é escrito.
- **Consideramos, por enquanto, que os registros de log são escritos diretamente no armazenamento estável (ou seja, eles não são mantidos em buffer)**
- Duas técnicas usando logs
  - Modificação de banco de dados adiada
  - Modificação de banco de dados imediata

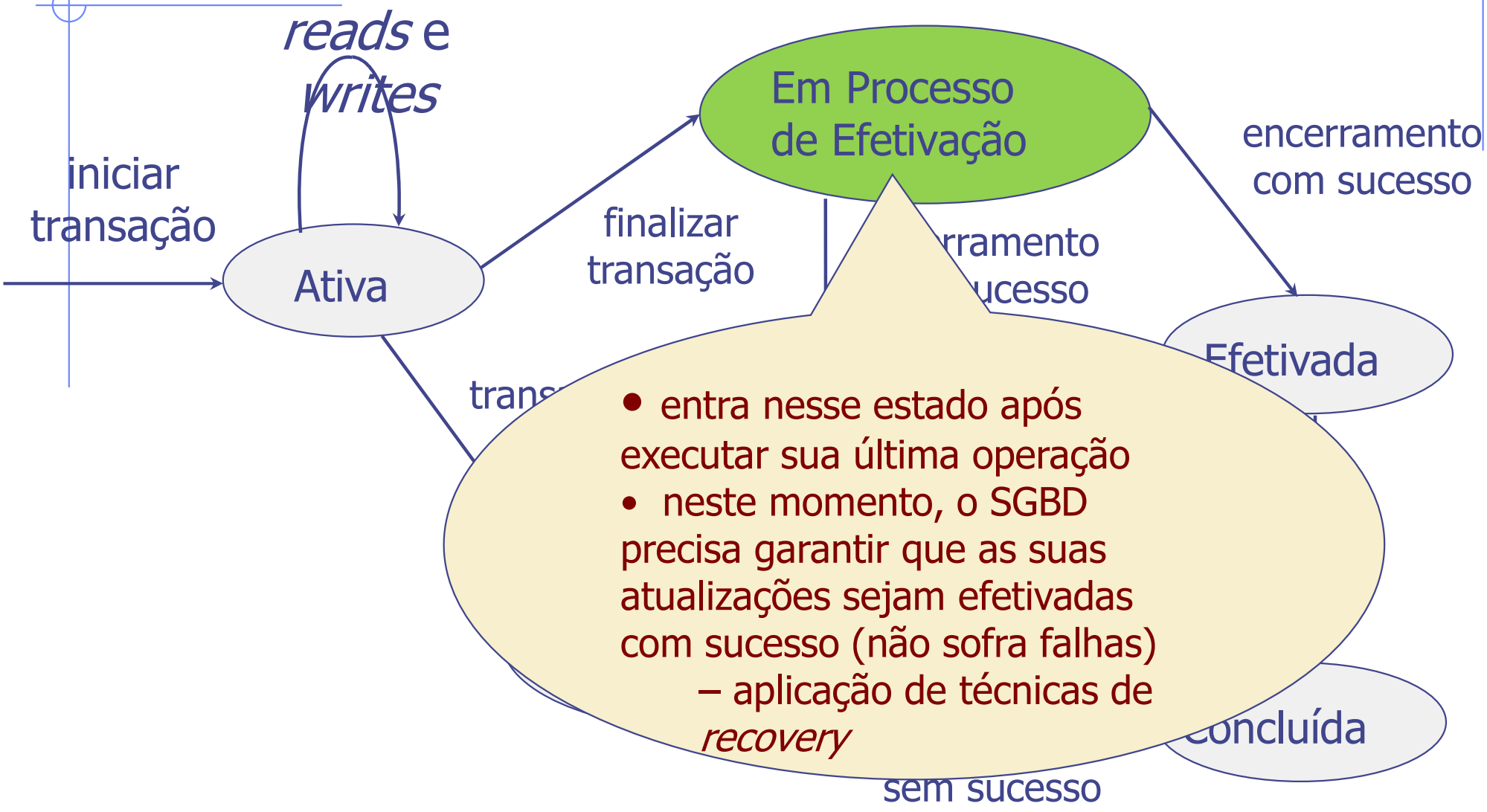
Modificação de banco de dados adiada

- O esquema de modificação de banco de dados adiada registra todas as modificações no log, mas **adia todas as escritas para depois da confirmação parcial (ponto de efetivação)**
- Protocolo típico
  - Uma transação não pode mudar o banco de dados em disco até que ela alcance seu ponto de efetivação
  - Uma transação não alcança seu ponto de efetivação até que todas as suas operações de atualização sejam registradas no log e até que seja forçada a gravação do log no disco

Relembrando...
Transição de Estados de uma Transação



Relembrando...
Transição de Estados de uma Transação



Modificação de banco de dados adiada

- Suponha que as transações são executadas serialmente
- A transação começa escrevendo o registro <T_i *start*> no log.
- Uma operação write(X) resulta em um registro de log <T_i, X, V> sendo escrito, onde V é o novo valor para X
  - Nota: o valor antigo não é necessário para esse esquema
- A escrita não é realizada em X nesse momento, mas é adiada.
- Quando T_i confirma parcialmente, <T_i commit> é escrito no log
- Finalmente, os registros de log são lidos e usados para realmente executar as escritas previamente adiadas.

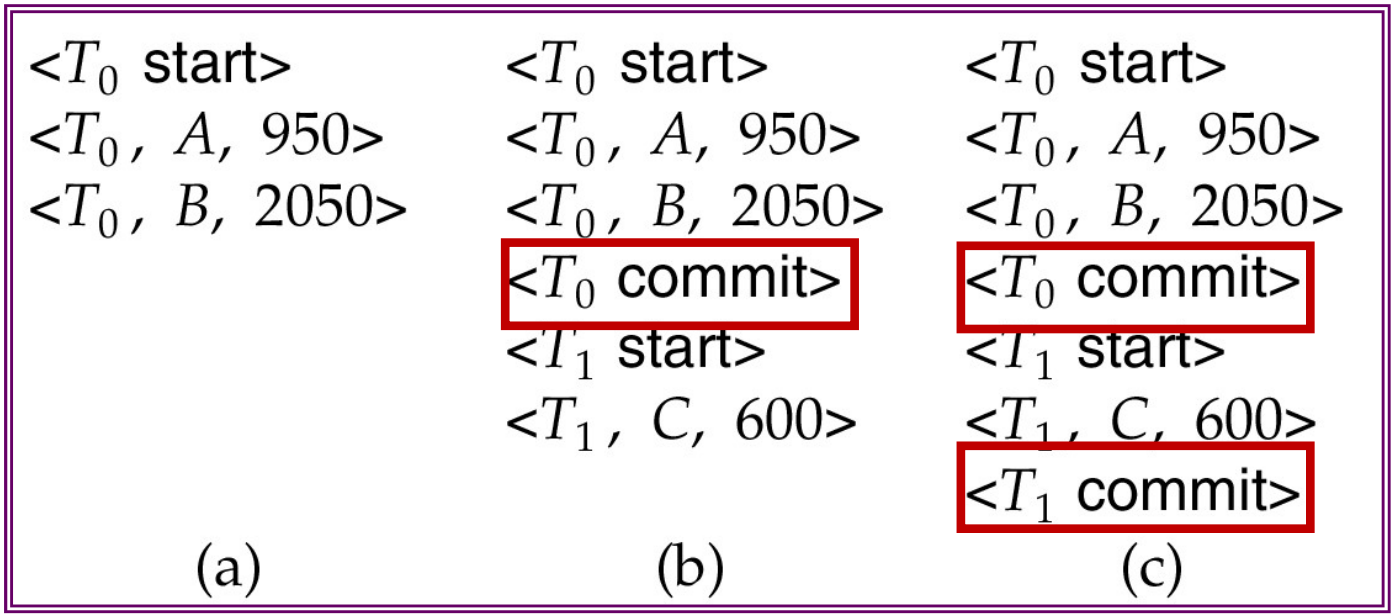
Modificação de banco de dados adiada

- **Durante a recuperação** após uma falha, uma transação precisa ser refeita se e somente se tanto <T_i start> quanto <T_i commit> existirem no log.
- Refazer uma transação T_i (redo T_i) define o valor de todos os itens de dados atualizados pela transação como os novos valores.
- Falhas podem ocorrer enquanto
  - a transação estiver executando as atualizações originais, ou
  - enquanto a ação de recuperação estiver sendo tomada
- transações de exemplo T_0 e T_1 (T_0 executa antes de T_1):

T_0	T_1
T_0: read (A)	T_1 : read (C)
A: = A - 50	C:-= C- 100
Write (A)	write (C)
read (B)	
B:= B + 50	
write (B)	

Modificação de banco de dados adiada

- A seguir mostramos o log conforme aparece em três instâncias de tempo.



- Se o log no armazenamento estável no momento da falha for como neste caso:

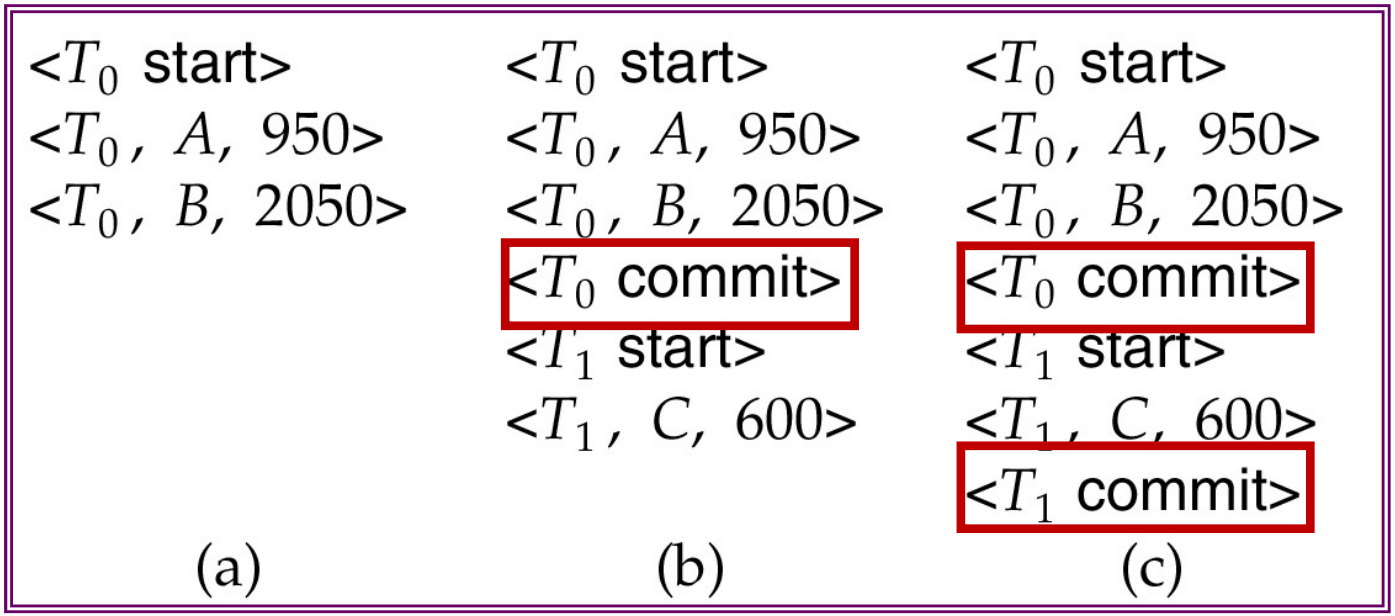
  (a) ?

  (b) ?

  (c) ?

Modificação de banco de dados adiada

- **A seguir mostramos o log conforme aparece em três instâncias de tempo.**



- **Se o log no armazenamento estável no momento da falha for como neste caso:**

  (a) Nenhuma ação de redo precisa ser tomada

  (b) redo(T_0) precisa ser realizado, pois <T_0 commit> está presente

  (c) redo(T_0) **precisa ser realizado seguido por redo(T_1), pois ambas atingiram seus pontos de confirmação**

Modificação de banco de dados imediata

- O esquema de modificação de banco de dados imediata permite que atualizações de banco de dados de uma transação não confirmada sejam feitas enquanto as escritas são emitidas
  - como pode ser preciso desfazer, os logs de atualização precisam ter valor antigo e valor novo
- O registro de log de atualização precisa ser escrito *antes* que o item do banco de dados seja escrito
  - Consideramos que o registro de log é enviado diretamente ao armazenamento estável
  - Pode ser estendido para adiar a saída do registro de log, desde que, antes da execução de uma operação output(*B*) para um bloco de dados B, todos os registros de log correspondentes aos itens *B* sejam esvaziados para o armazenamento estável
- A saída dos blocos atualizados pode ocorrer a qualquer momento antes ou depois do commit da transação
- A ordem em que os blocos são enviados pode ser diferente da ordem em que são escritos.

Exemplo:
Modificação de banco de dados imediata

Log	Write	Output
<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	
		B_B, B_C
<T_1 commit>		
		B_A

Nota: B_X indica bloco contendo X.

Modificação de banco de dados imediata

- O procedimento de recuperação possui duas operações em vez de uma:
  - undo(T_i) restaura o valor de todos os itens de dados atualizados por T_i aos seus valores antigos, indo para trás a partir do último registro para T_i
  - redo(T_i) define o valor de todos os itens de dados atualizados por T_i aos novos valores, indo para frente a partir do primeiro registro para T_i
- As duas operações precisam ser idempotentes
  - Ou seja, mesmo que a operação seja executada várias vezes, o efeito é o mesmo que se fosse executada uma vez
    - Necessário porque as operações podem ser novamente executadas durante a recuperação

Modificação de banco de dados imediata

- Ao recuperar-se após a falha:
  - A transação T_i precisa ser desfeita se o log tiver o registro <T_i start>, mas não contém o registro <T_i commit>.
  - A transação T_i precisa ser refeita se o log tiver o registro <T_i start> e o registro <T_i commit>.
- Operações de undo são realizadas primeiro, depois as operações de redo.

Exemplo:
Recuperação de modificação de BD imediata

A seguir mostramos o log conforme aparece em três instâncias de tempo

(a)	(b)	(c)
<T_0 start>	<T_0 start>	<T_0 start>
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 commit>	<T_0 commit>
	<T_1 start>	<T_1 start>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1 commit>

As ações de recuperação em cada um destes são:

(a) ?

(b) ?

(c) ?

Exemplo:
Recuperação de modificação de BD imediata

A seguir mostramos o log conforme aparece em três instâncias de tempo

(a)	(b)	(c)
<T_0 start>	<T_0 start>	<T_0 start>
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0 commit>	<T_0 commit>
	<T_1 start>	<T_1 start>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1 commit>

As ações de recuperação em cada um destes são:

(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 depois *A* e *B* são definidos para 950 e 2050, respectivamente

(c) redo (T_0) e redo (T_1): A e B são definidos para 950 e 2050 respectivamente. Depois, *C* é definido para 600

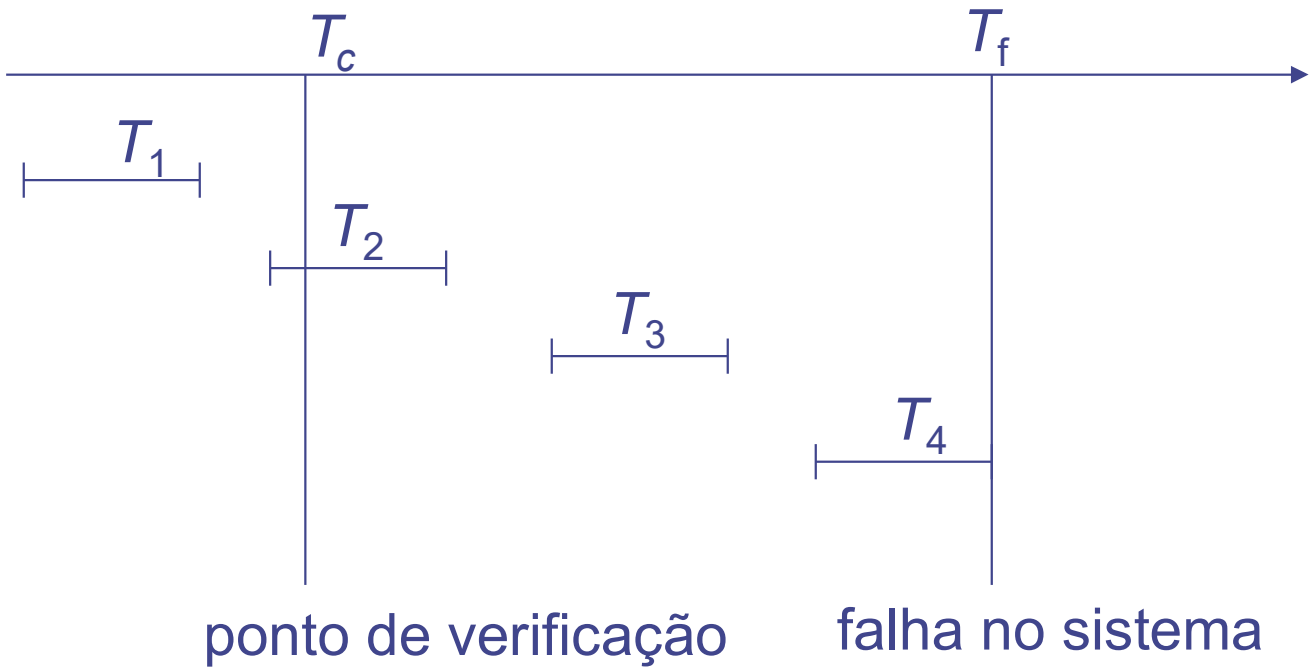
Pontos de verificação

- Problemas no procedimento de recuperação, conforme discutimos:
  - pesquisar o log inteiro é demorado
  - poderíamos desnecessariamente refazer transações que já emitiram sua saída no banco de dados.
- Facilite o procedimento de recuperação realizando periodicamente o ponto de verificação
  - Envie todos os registros de log atualmente residindo na memória principal para o armazenamento estável.
  - Envie todos os blocos de buffer modificados para o disco.
  - Escreva um registro de log < checkpoint> no armazenamento estável.

Pontos de verificação

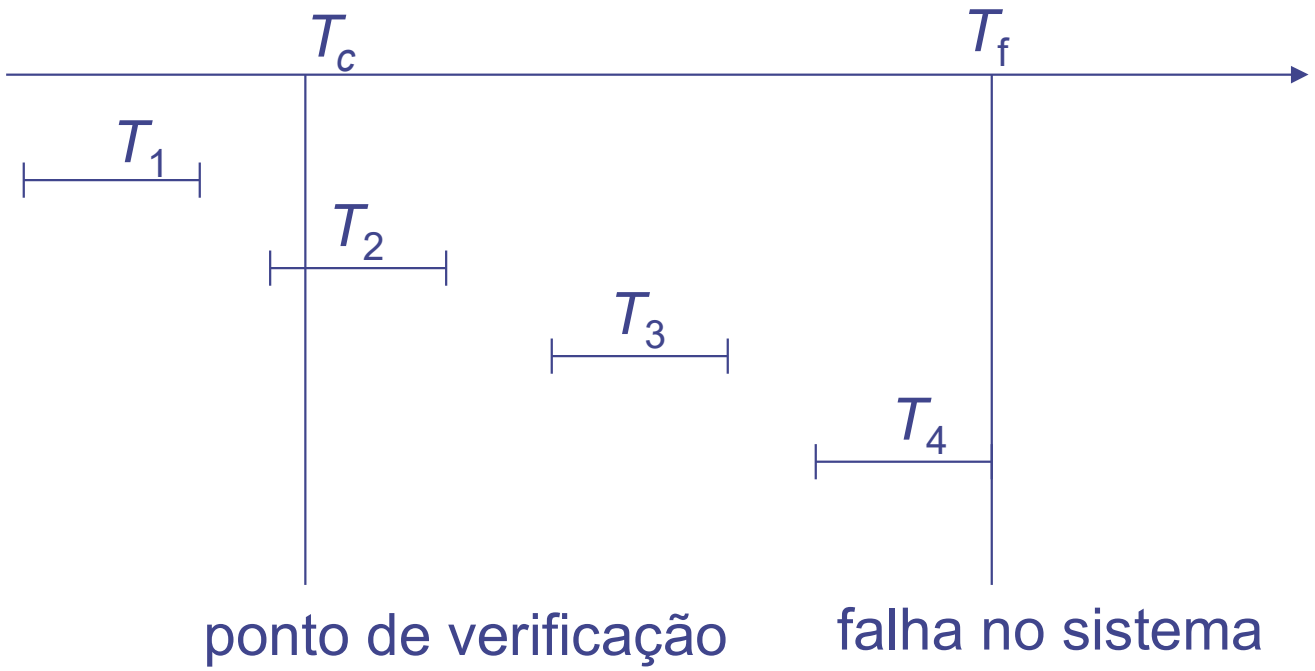
- Durante a recuperação, temos que considerar apenas a transação mais recente T_i que foi iniciada antes do ponto de verificação, e as transações que começaram após T_i.
  - Varra para trás a partir do final do log para encontrar o registro <checkpoint> mais recente.
  - Continue varrendo para trás até um registro <T_i start> ser encontrado.
  - Só precisa considerar a parte do log vindo após o registro start. A parte inicial do log pode ser ignorada durante a recuperação, e pode ser apagada sempre que for desejado.
  - Para todas as transações (começando de T_i ou mais) sem <T_i commit>, execute undo(T_i). (**Feito apenas no caso de modificação imediata**.)
  - Varrendo para frente no log, para todas as transações começando a partir de T_i ou depois com um <T_i commit>, execute redo(T_i).

Exemplo: Pontos de verificação



- ***Nesse caso o que acontece durante a recuperação?***

Exemplo: Pontos de verificação



- **T_1 pode ser ignorada (atualizações já enviadas ao disco devido ao ponto de verificação)**
  - **T_2 e T_3 refeitos**
  - **T_4 desfeito**

Recuperação com transações concorrentes

- Modificamos os esquemas de recuperação baseados em log para permitir que várias transações sejam executadas simultaneamente.
  - Todas as transações compartilham um único buffer de disco e um único log
  - Um bloco de buffer pode ter itens de dados atualizados por uma ou mais transações
- Consideramos o controle de concorrência usando um bloqueio estrito em duas fases;
  - ou seja, as atualizações de transações não confirmadas não devem ser visíveis a outras transações
    - Caso contrário, como realizar o undo se T1 atualiza A, depois T2 atualiza A e confirma, e finalmente T1 precisa abortar?

Recuperação com transações concorrentes

- O logging é feito conforme descrevemos anteriormente.
  - Os registros de log de diferentes transações podem ser intercalados no log.
- A técnica de ponto de verificação e as ações tomadas na recuperação precisam ser alteradas
  - pois várias transações podem estar ativas quando um ponto de verificação é realizado.

Recuperação com transações concorrentes

- Os pontos de verificação são realizados como antes, exceto que o registro de log do ponto de verificação agora tem a forma **< checkpoint *L*>** onde *L* é a lista de transações ativas no momento do ponto de verificação
  - Consideramos que nenhuma atualização está em andamento enquanto o ponto de verificação é executado (isso será aliviado mais tarde)

Recuperação com transações concorrentes

- Quando o sistema se recupera de uma falha, ele primeiro faz o seguinte:
  1. Inicializa a *lista de undo* e *lista de redo* para vazio
  2. Varre o log para trás a partir do fim, parando quando o primeiro registro <checkpoint *L*> for encontrado.
     Para cada registro encontrado durante a varredura:
     - se o registro for <T_i commit>, acrescenta T_i à *lista de redo*
     - se o registro for <T_i start>, então se T_i não está na *lista de redo*, acrescenta T_i à *lista de undo*
  3. Para cada T_i em *L*, se T_i não estiver na *lista de redo*, acrescenta T_i à *lista de undo*

Recuperação com transações concorrentes

- Neste ponto, *lista de undo* consiste em transações incompletas, que precisam ser desfeitas, e *lista de redo* consiste em transações acabadas, que precisam ser refeitas.

4. A recuperação agora continua da seguinte forma:
   - Varra o log para trás a partir do registro mais recente, parando quando registros <T_i start> tiverem sido encontrados para cada T_i na *lista de undo*.
     - Durante a varredura, realize undo para cada registro de log que pertence a uma transação na *lista de undo*.
   - Localize o registro <checkpoint *L*> mais recente.
   - Varra o log para frente a partir do registro <checkpoint *L*> até o final do log.
     - Durante a varredura, realize redo para cada registro de log que pertence a uma transação na *lista de redo*.

Exemplo de recuperação

- **Percorra as etapas do algoritmo de recuperação no log a seguir:**
  - **<T_0 start>**
  - **<T_0, *A*, 0, 10>**
  - **<T_0 commit>**
  - **<T_1 start>**
  - **<T_1, *B*, 0, 10>**
  - **<T_2 start>** **/* Varredura na etapa 4 pára aqui */**
  - **<T_2, *C*, 0, 10>**
  - **<T_2, *C*, 10, 20>**
  - **<checkpoint {T_1, T_2}>**
  - **<T_3 start>**
  - **<T_3, *A*, 10, 20>**
  - **<T_3, *D*, 0, 10>**
  - **<T_3 commit>**

lista undo = $\{T_1, T_2\}$
lista redo = $\{T_3\}$

Buffering de registro de log

- Buffering de registro de log: os registros de log são mantidos na memória principal, em vez de serem enviados diretamente para o armazenamento estável.
  - Registros de log são enviados ao armazenamento estável quando um bloco de registros de log no buffer estiver cheio, ou uma operação de log forçado for executada.
- O log forçado é realizado para confirmar uma transação forçando todos os seus registros de log (incluindo o registro de commit) para o armazenamento estável.
- Vários registros de log, portanto, podem ser enviados por meio de uma única operação de saída, reduzindo o custo da E/S.

Buffering de registro de log

- As regras a seguir precisam ser seguidas se os registros de log forem colocados em buffer:
  - Os registros de log são enviados para o armazenamento estável na ordem em que são criados.
  - A transação T_i só entra no estado de commit quando o registro de log <T_i commit> tiver sido enviado ao armazenamento estável.
  - Antes que um bloco de dados na memória principal seja enviado ao banco de dados, todos os registros de log pertencentes aos dados nesse bloco precisam ter sido enviados ao armazenamento estável.
    - Essa regra é chamada logging de escrita antecipada ou regra WAL (*Write Ahead Logging*)
      - *Estritamente falando, WAL só requer que informações de undo sejam enviadas*

Buffering de banco de dados

- O banco de dados mantém um buffer na memória dos blocos de dados
  - Quando um novo bloco é necessário, se o buffer estiver cheio, um bloco existente precisa ser removido do buffer
  - Se o bloco escolhido para remoção tiver sido atualizado, ele terá que ser enviado para o disco
- Como resultado da regra de logging de escrita antecipada, se um bloco com atualizações não confirmadas for enviado ao disco, os registros de log com informações de undo para as atualizações são enviados ao log no armazenamento estável primeiro.

Leitura complementar para casa

- Capítulo 18 e 19 do livro: Elmasri, Ramez; Navathe, Shamkant B. Sistemas de banco de dados.
- Capítulo 16 e 17 do livro: Silberschatz, A; Korth, H. F.; Sudarshan, S. Sistema de banco de dados.

Exercícios complementares

1. Para que é usado o log de sistema? Quais são as entradas típicas de um log de sistema? O que são checkpoints e por que eles são importantes? O que são os pontos de efetivação e por que eles são importantes?
2. Descreva a técnica de recuperação baseada em log com atualização imediata.
3. Descreva a técnica de recuperação baseada em log com atualização adiada.
4. Considere que a modificação imediata é usada em um sistema. Mostre, por meio de um exemplo, como o estado de um banco de dados poderia ficar inconsistente se os registros de log para uma transação não fossem enviados ao armazenamento estável antes que os dados atualizados pela transação fossem gravados em disco.

Conteúdo Prova Recuperação

- A **prova de recuperação** valerá **35 pontos** e substituirá a menor das notas obtidas entre Prova 1 e Prova 2
  - Em caso de aprovação obtida graças ao desempenho na recuperação, a nota final será saturada em 60 pontos
- O que será avaliado
  - Toda a matéria vista na disciplina!
- Quem pode fazer
  - Alunos que não obtiveram o rendimento mínimo para aprovação e com frequência mínima de 75%
- Os alunos que desejarem realizar a prova substitutiva deverão me enviar um e-mail até o dia 14/11 confirmando sua participação na prova
- **Data da Prova: 21/11 às 13:10h (em sala de aula)**

Sugestão de revisão das matérias anteriores...

- Revisar as notas de aula e as suas anotações
- Leituras complementares para casa
  - Fazem parte do conteúdo da prova!!!
  - Ao final de cada capítulo existem as seções "Resumo" e "Perguntas de Revisão" que sumarizam os principais conceitos abordados
- Revisar as listas de exercícios
- Fazer exercícios adicionais disponíveis nos livros da bibliografia da disciplina