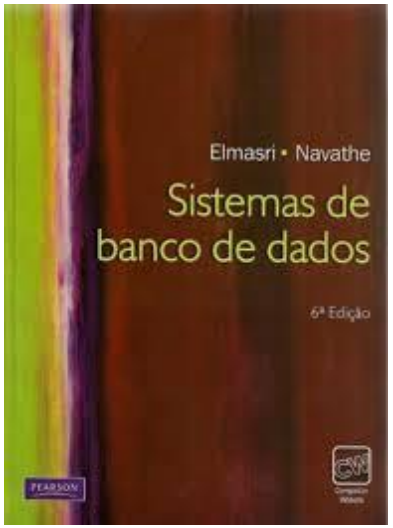
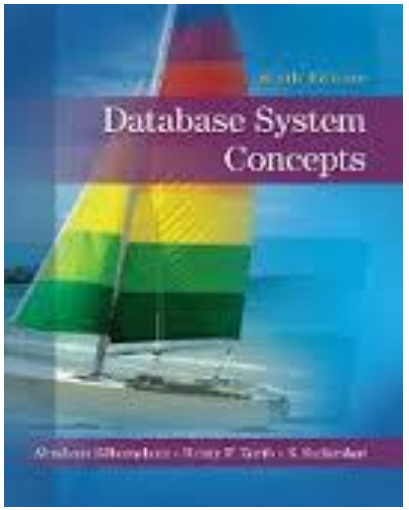
Unidade 12 – Controle de Concorrência e Recuperação de Banco de dados

Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP

Bibliografia



Sistemas de Banco de Dados
Elmasri / Navathe 6ª edição



Sistema de Banco de Dados
Korth, Silberschatz – Sixth Editon

Introdução

- O processamento de transações pode acarretar inconsistências nos bancos de dados;
- Um **schedule serializável** tem a propriedade de garantir a **consistência** do banco de dados, uma vez que tem comportamento equivalente a um **schedule serial**.
- A teoria de processamento de transações permite, por meio de grafos de precedência, que se verifique se um dado **schedule** é **serializável**.
- Porém, uma questão importante surge: **Como garantir a Serialização de um Schedule?**
- A técnica de **lock** (**bloqueio**) é geralmente empregada para o controle de concorrência de transações.

Lock (Bloqueio)

- Um **lock** é uma variável associada a um item de dados que descreve o status do tem em relação a possíveis operações que podem ser aplicadas a ele;
- Em geral, existe um **bloqueio** para cada item no banco de dados;
- São empregados como meio de **sincronizar** o acesso por transações concorrentes aos itens do banco de dados;
- Vários tipos de bloqueios podem ser usados no controle de concorrência.

Bloqueios Binários

- Podem ter dois estados ou valores: **Bloqueado** ou **Desbloqueado** (**1** ou **0**);
- Para um item **X** do Banco de Dados, se **lock**(**X**) = **1** o item **X** não pode ser acessado por uma operação que o requisite.
- Se **lock**(**X**) = **0**, o item **X** pode ser acessado quando requisitado;
- Duas operações são usadas com o bloqueio binário: **lock_item**(**X**) e **unlock_item**(**X**);
- Um bloqueio binário impõe a **exclusão mútua** no item de dados.

Operação lock_item

- Uma transação requisita acesso a um item de dados **X** do banco de dados, emitindo primeiro uma operação **lock_item**;
- Se **lock**(**X**) = **1**, então a transação é forçada a esperar;
- Se **lock**(**X**) = **0**, ela é configurada como **1** (a transação bloqueia o item) e a transação tem permissão para acessar o item **X**.

Operação unlock_item

- Quando a transação termina de usar o item, ela emite uma operação **unlock_item**(**X**), que define **lock**(**X**) de volta para **zero** (desbloqueia o item);
- Com isso, **X** pode ser acessado novamente por outras transações.

Operação lock_item

lock_item(*X*):

B: se LOCK(*X*) = 0 (* item está desbloqueado *)

então LOCK(*X*) ←1 (* bloqueia o item *)

se não

início

wait (until LOCK(*X*) = 0

e o gerenciador de bloqueio desperta a transação);

go to **B**

fim;

Fonte: Navathe/Elmasri

Operação unlock_item

unlock_item(*X*):

LOCK(*X*) ← 0; (* desbloqueia o item *)

se alguma transação estiver esperando

então acorda uma das transações em espera;

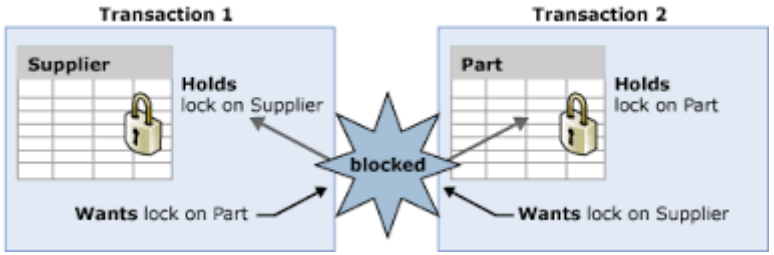
Fonte: Navathe/Elmasri

Bloqueio Binário – Observações

- As operações **lock_item** e **unlock_item** devem ser implementadas como unidades indivisíveis (conhecidas como **seções críticas** em sistemas operacionais);
- Isso significa que nenhuma intercalação deve ser permitida quando uma operação de bloqueio ou desbloqueio é iniciada, até que a operação termine;
- O comando **wait** na operação **lock_item** usualmente coloca a transação em uma **fila** de espera para o item **X** até que **X** seja desbloqueado;
- Outras transações que também queiram acessar **X** são colocadas na mesma **fila**;

Implementação do Bloqueio Binário

- O SGBD possui um **subsistema** de gerenciador de bloqueios para registrar e controlar o acesso aos bloqueio;
- O controle basicamente consiste em uma **tabela de bloqueio** associada ao item de dados e a identificação da transação que o requisitou. Essa tabela pode ser organizada em um arquivo **hash**.
- Itens que **não** estão gravados na tabela de bloqueio são considerados **desbloqueados**.
- Adicionalmente, o subsistema mantém uma **fila** das transações que estão em estado de espera para acessar o item.



Bloqueio Binário – Regras

- Para o esquema de bloqueio binário apresentado, cada transação precisa obedecer às seguintes **regras**:

1. Uma transação T precisa emitir a operação lock_item(X) antes de quaisquer operações read_item(X) ou write_item(X) serem realizadas em T.
2. Uma transação T precisa emitir a operação unlock_item(X) após todas as operações read_item(X) e write_item(X) serem completadas em T.
3. Uma transação T não emitirá uma operação lock_item(X) se já mantiver o bloqueio no item X.
4. Uma transação T não emitirá uma operação unlock_item(X) a menos que ela já mantenha o bloqueio no item X.

Fonte: Navathe/Elmasri

Bloqueio Binário - Observações

- No máximo uma transação pode manter o bloqueio de **um** item em particular;
- Assim, duas transações **não** podem acessar o mesmo item simultaneamente.

Quais as limitações do Bloqueio Binário ?

Bloqueio Binário - Limitações

- O esquema de **bloqueio binário** apresentado é muito **restritivo**, pois no máximo **uma** transação pode manter um bloqueio em um determinado item;
- Quando diferentes transações acessam um item de dados com operações de leitura, **não há conflito** (conforme teoria de processamento de transações);
- Assim, bloqueios binários **não** permitem que várias operações de leituras sejam compartilhadas por diferentes transações em um determinado item de dados;
- O **esquema de bloqueio múltiplo** pode ser empregado para amenizar essa restrição.

Bloqueio Múltiplo

- Também chamados Bloqueios Compartilhados/Exclusivos ou de Leitura/Gravação;
- Neste esquema há três operações de bloqueios: read_lock(X), write_lock(X) e unlock(X);
- Um bloqueio associado a um item **X**, agora tem três estados possíveis: **bloqueado para leitura**, **bloqueado para gravação** e **desbloqueado**.
- Um item de dados **bloqueado para leitura** também é chamado de **bloqueado por compartilhamento**, pois outras transações podem ler o item de dados;
- Um item de dados **bloqueado para gravação** também é chamado de **bloqueado exclusivo**, uma vez que uma única transação mantém exclusivamente o bloqueio no item de dados.

read_lock(X)

```
read_lock(X):
B:   se LOCK(X) = "unlocked"
          então início LOCK(X) ← "read-locked";
                    num_de_leituras(X) ← 1
                    fim
     se não se LOCK(X) = "read-locked"
          então num_de_leituras(X) ← num_de_leituras(X) + 1
     se não início
               wait (até que LOCK(X) = "unlocked"
                    e o gerenciador de bloqueio desperta a transação);
               go to B
               fim;
```

Fonte: Navathe/Elmasri

write_lock(X)

```
write_lock(X):
B:   se LOCK(X) = "unlocked"
          então LOCK(X) ← "write-locked"
     então início
          wait (até que LOCK(X) = "unlocked"
               e o gerenciador de bloqueio desperta a transação);
          go to B
          fim;
```

Fonte: Navathe/Elmasri

unlock(X)

```
unlock (X):
    se LOCK(X) = "write-locked"
        então início LOCK(X) ← "unlocked";
                desperta uma das transações aguardando, se houver
                fim
    se não se LOCK(X) = "read-locked"
        então início
                num_de_leituras(X) ← num_de_leituras(X) – 1;
                se num_de_leituras(X) = 0
                    então início LOCK(X) = "unlocked";
                            desperta uma das transações aguardando, se houver
                            fim
                fim;
```

Fonte: Navathe/Elmasri

Bloqueio Múltiplo – Regras

- Para o esquema de bloqueio múltiplo apresentado, cada transação precisa obedecer às seguintes **regras**:

1. Uma transação T precisa emitir a operação read_lock(X) ou write_lock(X) antes que qualquer operação read_item(X) seja realizada em T.

Fonte: Navathe/Elmasri

Bloqueio Múltiplo - Regras

2. Uma transação T precisa emitir a operação write_lock(X) antes que qualquer operação write_item(X) seja realizada em T.

Fonte: Navathe/Elmasri

Bloqueio Múltiplo - Regras

3. Uma transação T precisa emitir a operação unlock(X) após todas as operações read_item(X) e write_item(X) serem completadas em T.

Fonte: Navathe/Elmasri

Bloqueio Múltiplo - Regras

4. Uma transação T não emitirá uma operação read_lock(X) se ela já mantiver um bloqueio de leitura (compartilhado) ou um bloqueio de gravação (exclusivo) no item X.

Fonte: Navathe/Elmasri

Bloqueio Múltiplo – Regras

5. Uma transação T não emitirá uma operação write_lock(X) se ela já mantiver um bloqueio de leitura (compartilhado) ou um bloqueio de gravação (exclusivo) no item X.

Fonte: Navathe/Elmasri

Bloqueio Múltiplo - Regras

6. Uma transação T não emitirá uma operação unlock(X) a menos que já mantenha um bloqueio de leitura (compartilhado) ou um bloqueio de gravação (exclusivo) no item X.

Fonte: Navathe/Elmasri

O uso de bloqueios garante a serialização de uma Transação ?

Bloqueios e Serialização

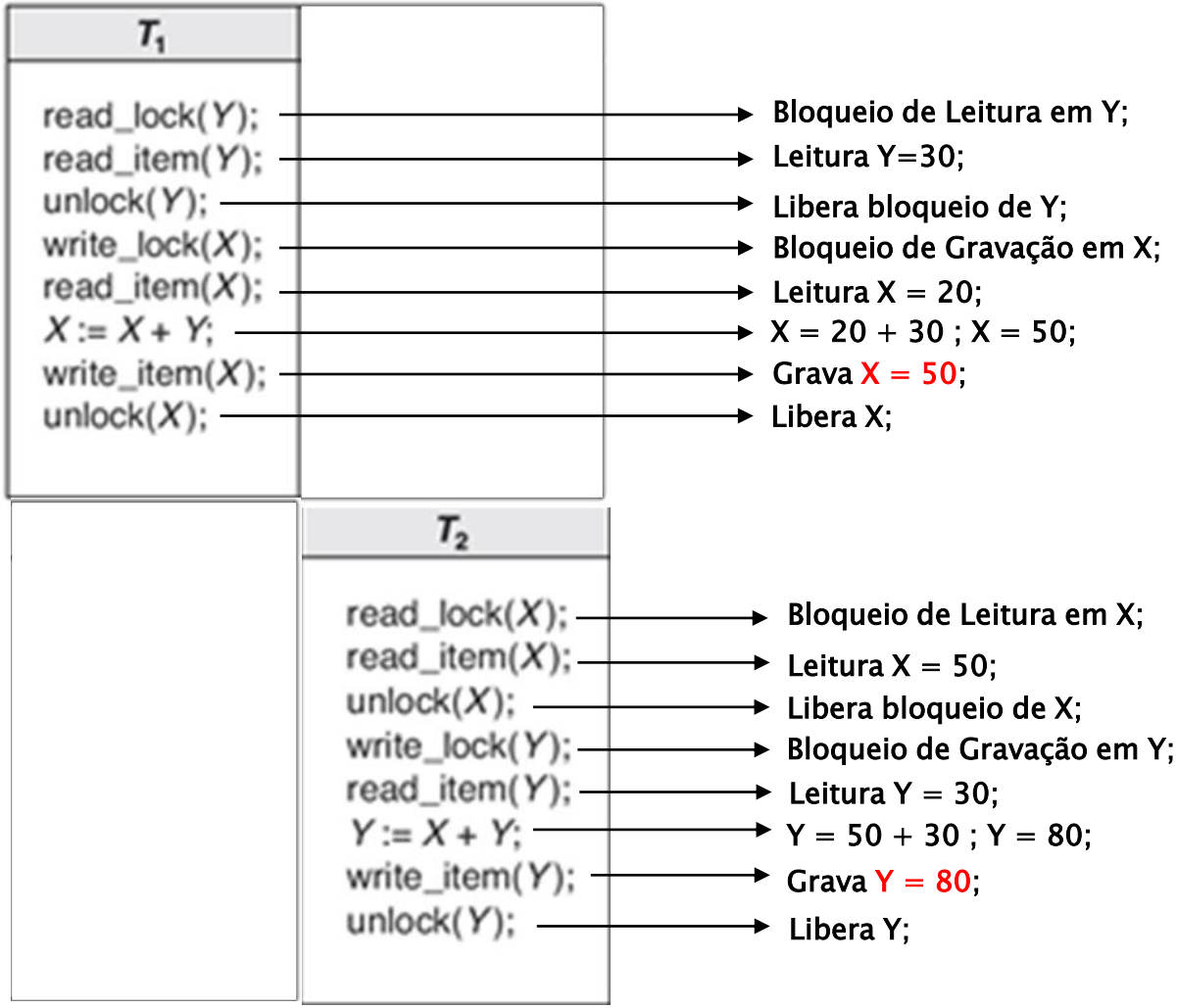
- ✓ Consideremos duas Transações T1 e T2.

T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
X := X + Y;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	Y := X + Y;
	write_item(Y);
	unlock(Y);

Bloqueios e Serialização

- ✓ Seja S_1 um Schedule Serial de **T1** seguido de **T2**;
- ✓ Valores Iniciais: **X**=20 e **Y** = 30



Bloqueios e Serialização

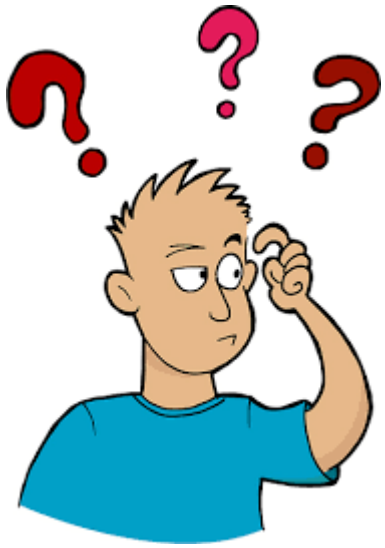
- ✓ O Schedule abaixo está usando Bloqueios, mas é serializável ?

T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);	

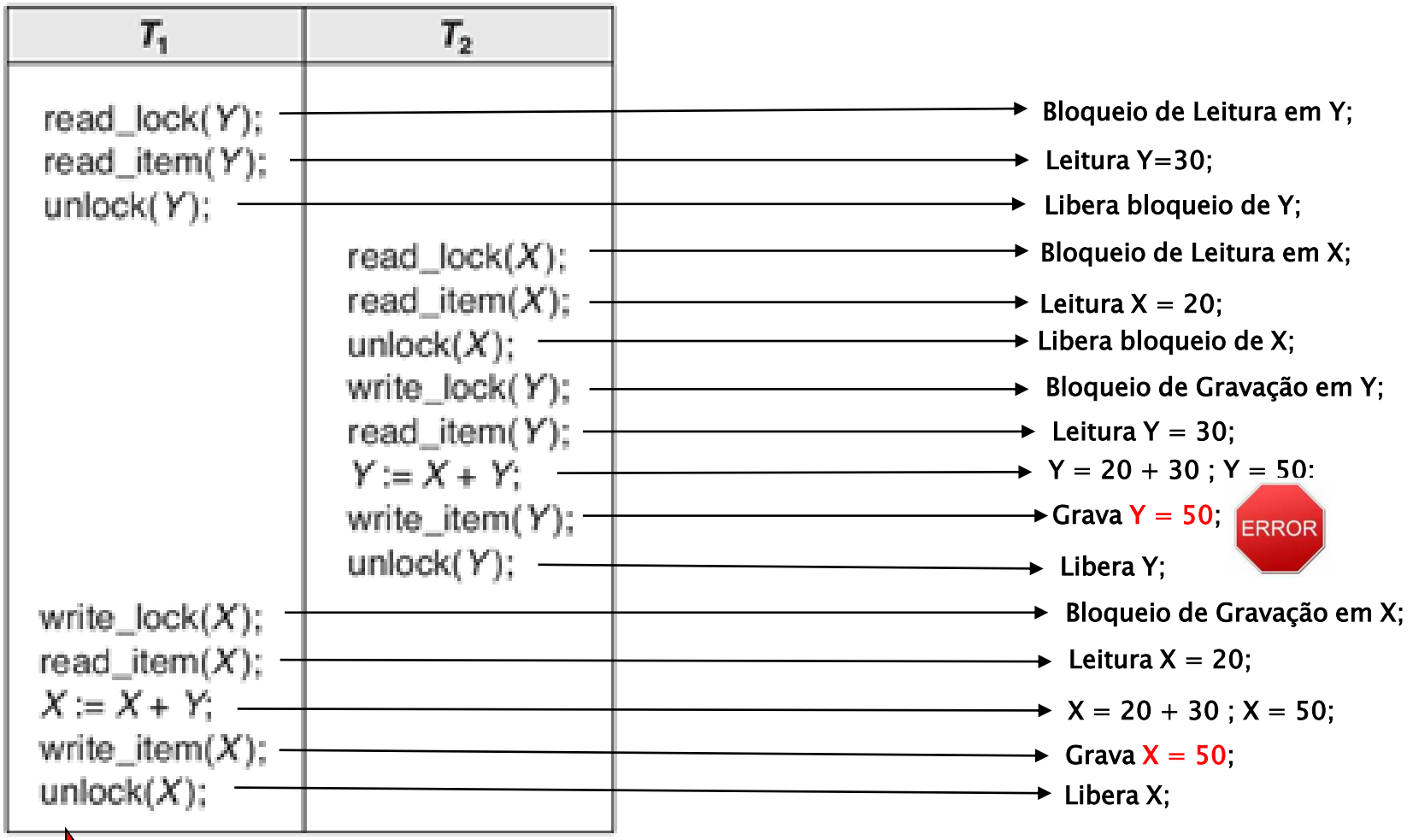
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	Y := X + Y;
	write_item(Y);
	unlock(Y);

write_lock(X);
read_item(X);
X := X + Y;
write_item(X);
unlock(X);	



Bloqueios e Serialização

- ✓ O Schedule abaixo está usando Bloqueios, mas é serializável ?
- ✓ Valores iniciais: **X** = 20 e **Y** = 30



NÃO É SERIALIZÁVEL

Bloqueios e Serialização

- Assim, bloqueios são mecanismos úteis e essenciais para o controle de concorrência mas, **NÃO** garantem a serialização de schedules por si só;
- Para se garantir a serialização, deve-se seguir um **PROTOCOLO** adicional em relação ao posicionamento das operações de bloqueio e desbloqueio em cada transação;
- O protocolo mais empregado em Sistemas Gerenciadores de Banco de Dados é o **Two-Phase Locking** (Bloqueio de duas fases).

Two-Phase Locking

- Diz-se uma transação segue o protocolo de Bloqueio em Duas Fases se **todas** as operações de bloqueio (**read_lock** e **write_lock**) precedem a primeira operação de desbloqueio na transação;

- Essa transação pode ser dividida em duas fases: uma fase de expansão ou crescimento e outra de encolhimento.

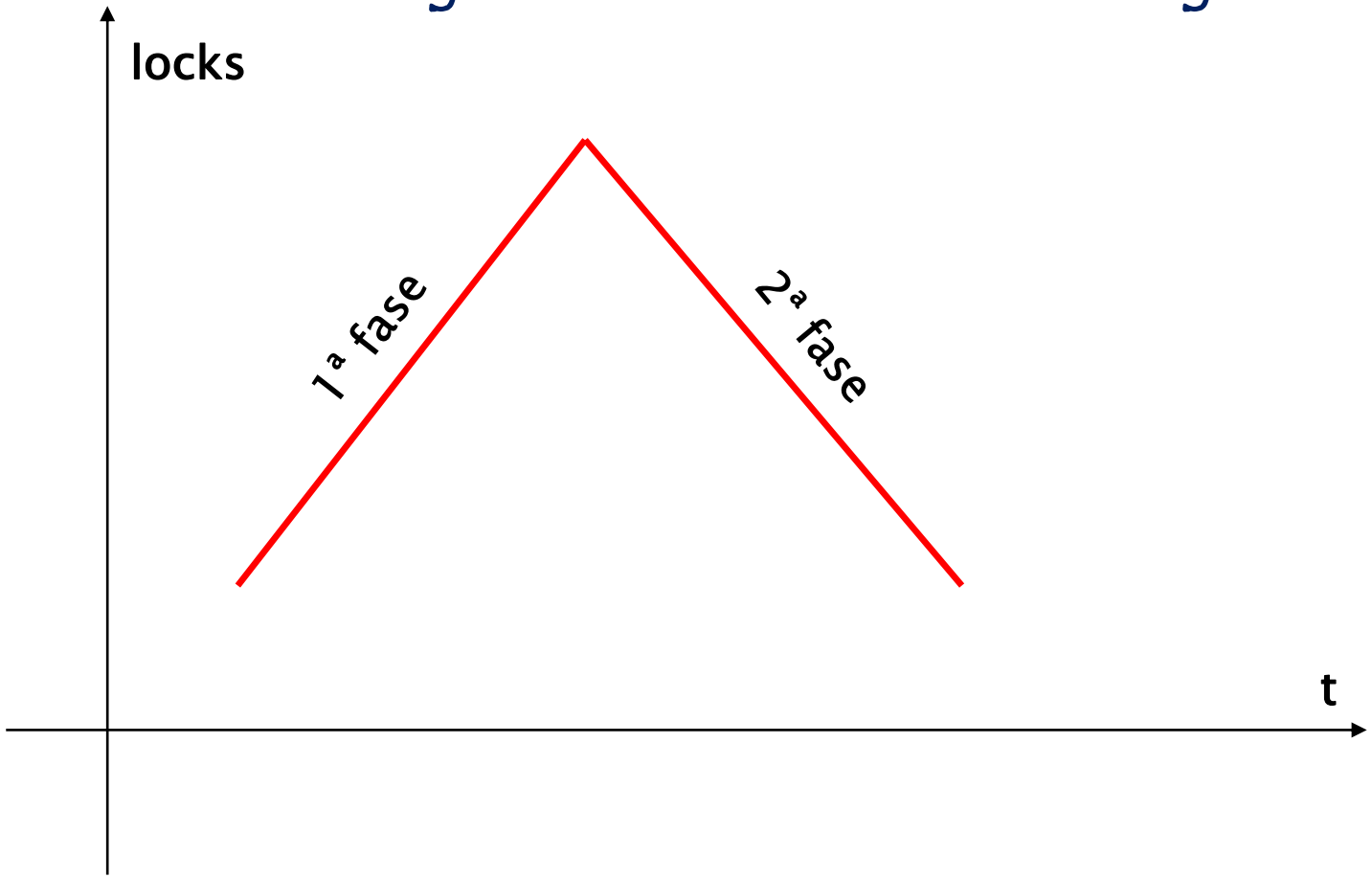
- Na fase de **crescimento** (primeira), **transação pode obter novos bloqueios**, **mas nenhum pode ser liberado**;

- Na fase de **encolhimento** (segunda), **transação pode liberar bloqueios existentes**, **mas nenhum novo bloqueio pode ser adquirido.**

Two-Phase Locking

- Prova-se que, se cada transação em um Schedule seguir o protocolo de bloqueio de duas fases, o Schedule é **garantidamente serializável**.
- Com isso, evita-se a necessidade de se testar a serialização dos schedules.
- O protocolo de bloqueio, ao impor as **regras de bloqueio** em duas fases, também impõe a **serialização**.

Diagrama Two-Phase Locking



Exemplo - Two-Phase Locking

Transação 1: Transferência

T1

```
ler(X)
X = X - N
gravar(X)
ler(Y)
Y = Y + N
gravar(Y)
```

Objetivo: Transferir N livros de uma prateleira para outra;

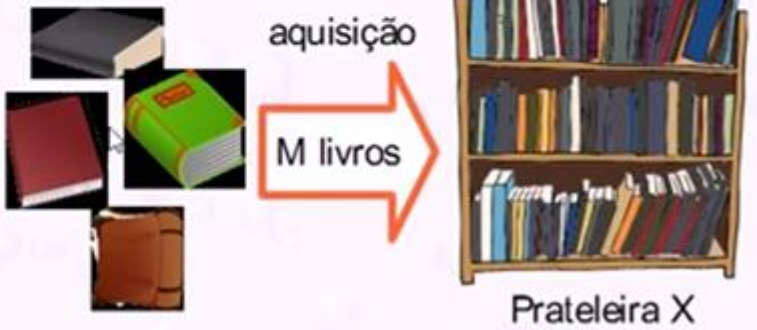
X: número de livros da prateleira X
Y: número de livros na prateleira Y
N: número de livros a serem transferidos

Exemplo - Two-Phase Locking

Transação 2: Aquisição

T2

```
ler(X)
X = X + M
gravar(X)
```



Objetivo: Incluir novos livros na prateleira X;

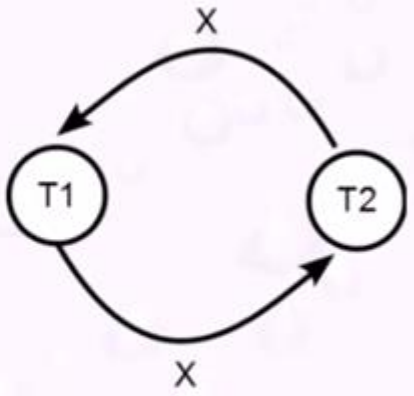
M: novos livros que serão acrescentados na prateleira X

Vimos que a concorrência pode acarretar problemas ...

T1	T2
ler(X)	**ler**(X)
X = X − N	X = X + M
gravar(X)	**gravar**(X)
ler(Y)	
Y = Y + N	
gravar(Y)	

Esse schedule NÃO é serializável...

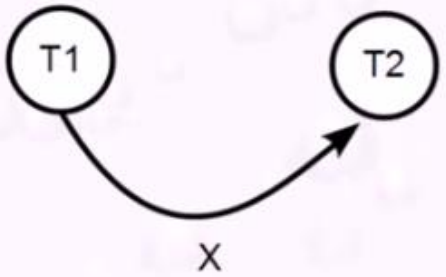
T1	T2
ler (X)	
X = X - N	
	ler (X)
	X = X + M
gravar (X)	
ler (Y)	
	gravar (X)
Y = Y + N	
gravar (Y)	



Plano Não Serializável

No entanto, esse schedule É serializável...

T1	T2
ler(X)	
X = X - N	
gravar(X)	
	ler(X)
	X = X + M
	gravar(X)
ler(Y)	
Y = Y + N	
gravar(Y)	

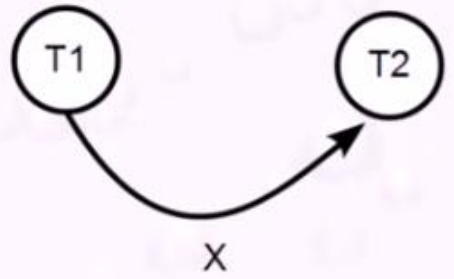


Plano Serializável

Two-Phase Locking

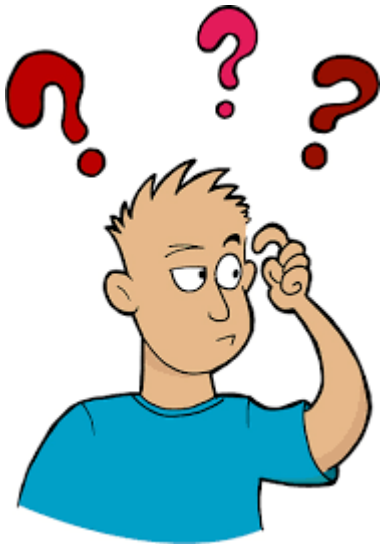
- Com o 2-Phase Locking, pode-se antecipadamente obter esse schedule...

T1	T2
ler (X)	
X = X − N	
gravar (X)	
	ler (X)
	X = X + M
	gravar (X)
ler (Y)	
Y = Y + N	
gravar (Y)	



Plano Serializável

Como se deve ajustar a transação para que ela atenda ao 2-Phase Locking?



Ajuste das Transações para atendimento ao Protocolo

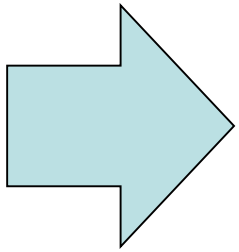
- Deve-se incluir locks e unlocks em atendimento às duas fases do protocolo;

T1	T2
ler(X)	ler(X)
X = X - N	X = X + M
gravar(X)	gravar(X)
ler(Y)	
Y = Y + N	
gravar(Y)	

Ajustando T1 ao protocolo

- T1 lê e grava X; Em seguida lê e grava Y;

T1
ler(X)
X = X - N
gravar(X)
ler(Y)
Y = Y + N
gravar(Y)



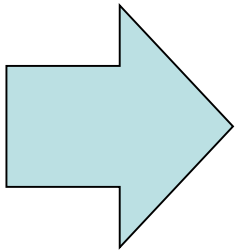
T1	
wlock(X)	
ler(X)	
X = X - N	
gravar(X)	
wlock(Y)	1ª fase
unlock(X)	
ler(Y)	
Y = Y + N	
gravar(Y)	
unlock(Y)	2ª fase

O primeiro **unlock** inicia a segunda fase

Ajustando T2 ao protocolo

- T2 lê e grava X;

T2
ler(X)
X = X + M
gravar(X)



T2
wlock(X)
ler(X)
X = X + M
gravar(X)
unlock(X)

1ª fase

2ª fase

O primeiro **unlock** inicia a segunda fase

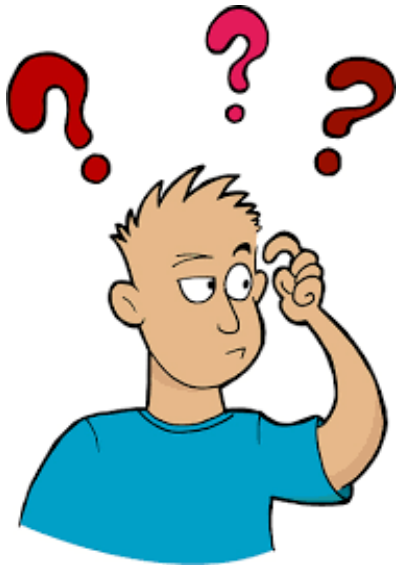
Schedule atendendo ao 2-Phase Locking

Após o ajuste de T1 e T2 ao protocolo 2-Phase Locking, pode-se definir o schedule que, como consequência, será serializável;

T1	T2
wlock(X)	
ler(X)	
X = X – N	
gravar(X)	
wlock(Y)	
unlock(X)	
ler(Y)	wlock(X)
Y = Y + N	ler(X)
gravar(Y)	X = X + M
unlock(Y)	gravar(X)
	unlock(X)

Exercício – Estas transações são 2-Phase Locking?

T1	T2
wlock(X)	**wlock**(X)
ler(X)	**ler**(X)
X = X - N	X = X + M
gravar(X)	**gravar**(X)
unlock(X)	**unlock**(X)
wlock(Y)	
ler(Y)	
Y = Y + N	
gravar(Y)	
unlock(Y)	



Exercício – Estas transações são 2-Phase Locking?

- Resposta: T1 **não** é 2-Phase Locking , pois há wlock(Y) após unlock(X)
- Assim, qualquer schedule com T1 não será serializável;

T1	T2
wlock(X)	wlock(X)
ler(X)	ler(X)
X = X - N	X = X + M
gravar(X)	gravar(X)
unlock(X)	unlock(X)
wlock(Y)	
ler(Y)	
Y = Y + N	
gravar(Y)	
unlock(Y)	

Protocolo 2-Phase Locking - Observações

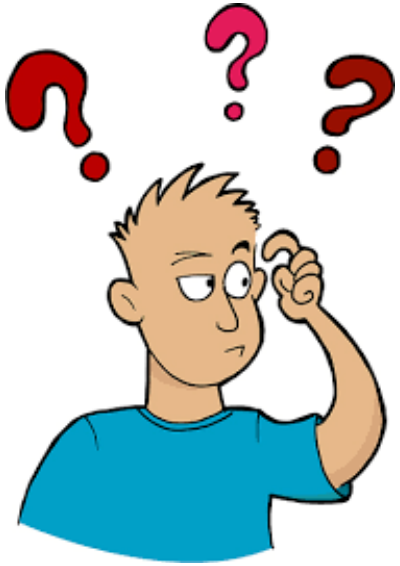
- Na prática, os SGBD's utilizam variações do protocolo 2PL;
- **2PL conservador** ou **estático**: todos os itens que a transação acessa são bloqueados **antes** do início da execução da transação. Caso algum item não esteja disponível, a transação entra em estado de espera;
- **2PL estrito**: Uma transação não libera nenhum de seus bloqueios exclusivos (gravação) até a ocorrência de um **commit** ou de um **abort**;
- **2PL rigoroso**: Uma transação não libera nenhum de seus bloqueios (exclusivo ou compartilhado) até a ocorrência de um **commit** ou **abort**;
- Em geral, o próprio subsistema de controle de concorrência é responsável por gerar as solicitações de **read_lock** e **write_lock**.

Subsistema de Controle de Concorrência

- Sempre que a transação **T** emitir um **read_item**(**X**), o sistema então chama a operação **read_lock**(**X**) em favor de **T.** Se **lock**(**X**) for bloqueado para gravação por alguma outra transação **T'**, o sistema coloca **T** na fila de espera;

- Se a transação **T** emitir um **write_item**(**X**), o sistema então chama a operação **write_lock**(**X**) em favor de **T**. Se **lock**(**X**) estiver bloqueado para gravação ou bloqueado para leitura por alguma outra transação **T**, o sistema coloca **T** na fila de espera;

Bloqueios podem acarretar problemas ?



Problemas com Bloqueios

- Deadlock;
- Starvation;

Deadlock

- Ocorre quando cada transação **T** em um conjunto de duas ou mais transações está esperando por algum item que está bloqueado por alguma outra transação **T'** no conjunto;

- Com isso, o bloqueio nunca será liberado;

T_1'

read_lock(Y);
read_item(Y);
write_lock(X);
unlock(Y)
read_item(X);
X := X + Y;
write_item(X);
unlock(X);

T_2'

read_lock(X);
read_item(X);
write_lock(Y);
unlock(X)
read_item(Y);
Y := X + Y;
write_item(Y);
unlock(Y);

Exemplo – Deadlock

- T1’ está na fila de espera para X que está bloqueado por T2’;
- T2’ está na fila de espera para Y que está bloqueado por T1’;
- Nesse meio tempo, nem T1’ , nem T2’, nem qualquer outra transação podem acessar os itens X e Y;

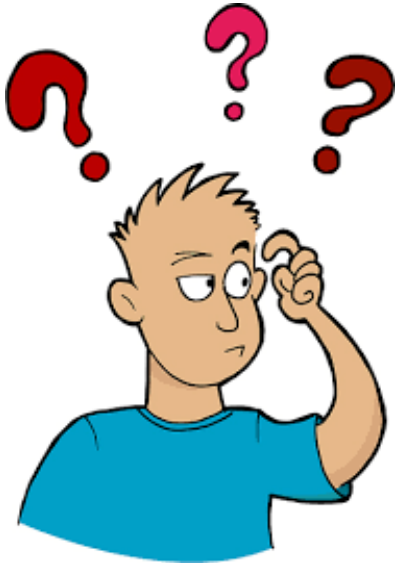
T_1'

read_lock(Y);
read_item(Y);
write_lock(X);
unlock(Y)
read_item(X);
X := X + Y;
write_item(X);
unlock(X);

T_2'

read_lock(X);
read_item(X);
write_lock(Y);
unlock(X)
read_item(Y);
Y := X + Y;
write_item(Y);
unlock(Y);

Como detectar deadlocks ?



Protocolos de Detecção de Deadlock

- Pode-se utilizar o **2PL conservador**, no qual cada transação bloqueie todos os itens que precisar com **antecedência**. Se qualquer um dos itens não puder ser obtido, a transação **espera** e, depois tenta novamente bloquear todos os itens de que precisa. (Essa solução **limita** a concorrência entre as transações).

- Outro esquema simples para lidar com o deadlock é o uso de **timeouts**. Nesse método, se uma transação esperar por um período de tempo maior que o tempo-limite definido pelo sistema, o sistema pressupõe que a transação possa estar em deadlock e a **aborta** – independentemente de um deadlock realmente existir ou não.

Starvation

- Ocorre quando uma transação **não** pode prosseguir por um período de tempo indefinido enquanto que outras transações no sistema continuam normalmente;
- Isso pode ocorrer se o esquema de espera para itens bloqueados for injusto, dando prioridade a algumas transações em relação a outras;
- Uma solução é utilizar-se a estratégia **First-In-First-Out**;
- Outro esquema seria permitir o aumento da **prioridade** da transação quanto mais tempo ela esperar.

Conceitos de Recuperação de Banco de Dados

- Recuperação de falhas de transação significa que o banco de dados é **restaurado** ao **estado consistente** mais recente antes do momento da falha;
- Para haver sucesso na recuperação, o sistema precisa manter informações sobre as mudanças que foram aplicadas aos itens de dados pelas diversas transações;
- Essas informações costumam ser mantidas no **log do sistema**.

Mecanismos de Recuperação
Propósito

- Restaurar o Banco de Dados ao seu último estado **consistente** antes de uma falha;
- Preservar as propriedades **ACID**: **Atomicidade**, **Consistência**, **Isolamento** e **Durabilidade**;

ACID

- **Atomicidade**. Uma transação é uma unidade de processamento **atômica**. Deve ser realizada em sua totalidade ou não ser realizada de forma alguma.

- **Consistência**. Uma transação deve preservar a consistência, ou seja deve levar o banco de dados de um estado **consistente** para outro.

- **Isolamento**. A execução de uma transação **não** deve ser **interferida** por quaisquer outras transações que ocorram simultaneamente.

- **Durabilidade**. As mudanças aplicadas ao banco de dados pela transação confirmada precisam **persistir** no banco de dados.

Conceitos de Recuperação

- Se houver **dano extensivo** a uma grande parte do banco de dados devido à falhas catastróficas (falhas em disco), o método de recuperação restaura uma cópia antiga do banco de dados. (procedimento de **backup**);

- Quando o banco de dados no disco **não está fisicamente danificado** (falha não catastrófica), o protocolo de recuperação não precisa de cópias de arquivamento completa do banco de dados. Em vez disso, usa-se entradas do arquivo **log** para a recuperação do banco de dados.

Conceitos de Recuperação

- Se houver **dano extensivo** a uma grande parte do banco de dados devido à falhas catastróficas (falhas em disco), o método de recuperação restaura uma cópia antiga do banco de dados. (procedimento de **backup**);

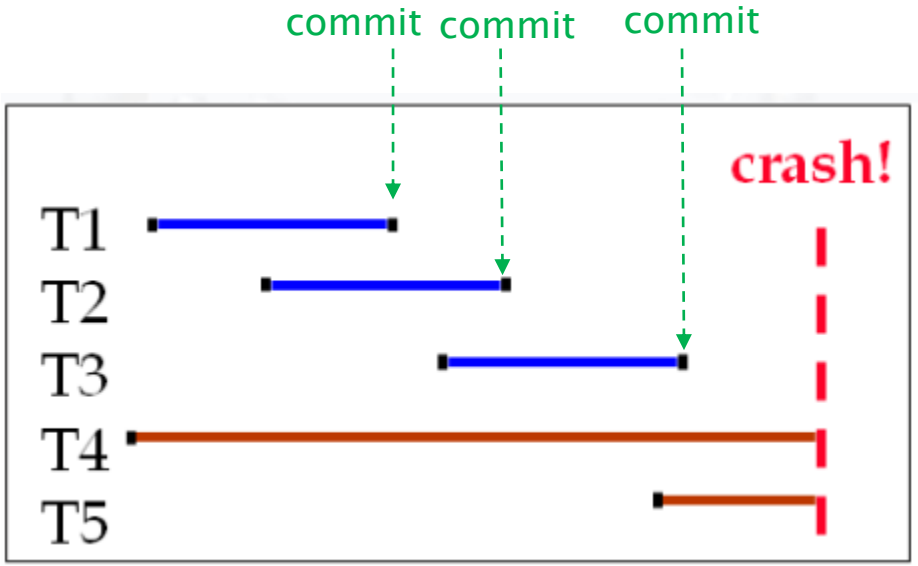
- Quando o banco de dados no disco **não está fisicamente danificado** (falha não catastrófica), o protocolo de recuperação não precisa de cópias de arquivamento completa do banco de dados. Em vez disso, usa-se entradas do arquivo **log** para a recuperação do banco de dados.

Gerenciamento de Recuperação

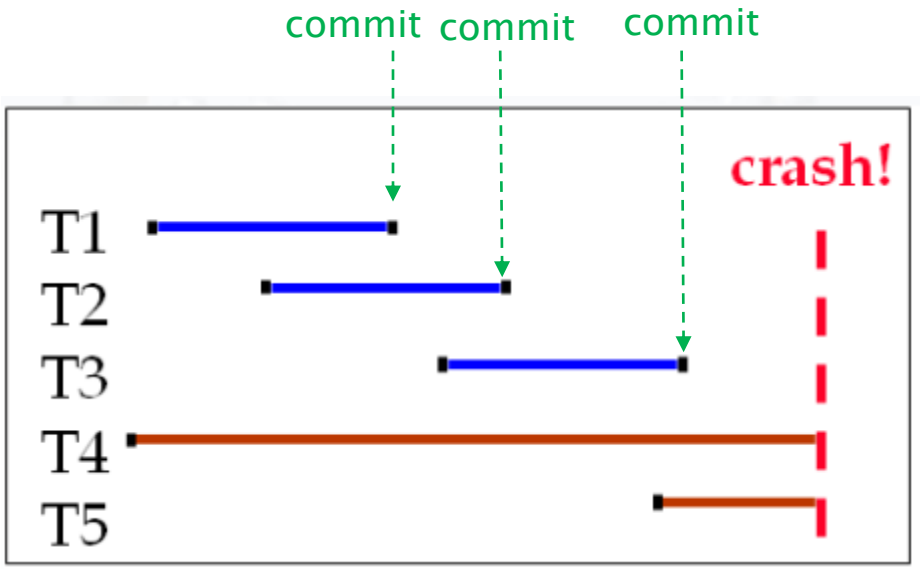
- Durante a execução de um SGBD, **não** ocorre atualização síncrona (on-line) do item do banco de dados em disco;
- As alterações dos itens de dados são registradas persistentemente (de forma permanente) no arquivo de **log** em disco e mantidas em buffers do SGBD;
- Após a transação atingir o seu ponto de confirmação (**commit**) os dados alterados serão posteriormente gravados (persistidos) em disco;

Gerenciamento de Recuperação

- O gerenciamento da recuperação garante a Atomicidade e a Durabilidade das Transações;
  - Atomicidade: Transações podem reverter (rollback)
  - Durabilidade: O que fazer quando houver algum crash do sistema ?



Gerenciamento de Recuperação



T1 , T2 e T3 devem persistir (durabilidade)
T4 e T5 devem ser desfeitas (atomicidade)

LOG

- Mantém o registro **sequencial** das operações de transação que afetam os itens do banco de dados;
- Estes dados podem ser necessários para:
  - Desfazer ações de uma transação "**abortada**"
  - Recuperar o sistema de Falhas
  - Auditoria
- O arquivo de LOG é mantido em disco
  - Afetado apenas pelas falhas em disco ou catastróficas
  - Recomenda-se gravação em disco separado

Protocolo Write-Ahead Logging (Escrita Antecipada)

- Grava registro de operação no arquivo de **log** **ANTES** que a modificação do item seja gravada em disco (garante a **atomicidade**);
- Todas as operações de uma Transação são gravadas no arquivo de **log** **ANTES** do **commit** (garante a **durabilidade**).

Tipos de Registro do LOG

- [start_transaction, T]
- [write_item, T, X, valor_antigo, novo_valor]
- [read_item, T, X]
- [commit, T]
- [abort, T]
- [checkpoint]

LOG – Observações

- Cada transação T tem um identificador único gerado automaticamente pelo sistema
- Campos para recuperação (UNDO e REDO):
  - BFIM (Before Image): estado antes da alteração
    - usado para UNDO
  - AFIM (After Image): estado depois da alteração
    - usado para REDO

LOG - Exemplo

X = 50 Y = 110
N = 20 M = 40

Plano	
início	
ler(X)	ler(50)
X = X – N	X = 50 – 20
início	
gravar(X)	gravar(30)
ler(X)	ler(30)
X = X + M	X = 30 + 40
ler(Y)	ler(110)
gravar(X)	gravar(70)
Y = Y + N	Y = 110 + 20
gravar(Y)	gravar(130)
commit	
commit	

LOG							
	Trans.	^Trás	^Frente	Op.	Item	BFIM	AFIM
1	T1	0	2	start			
2	T1	1	4	read	X		
3	T2	0	5	start			
4	T1	2	6	write	X	50	30
5	T2	3	7	read	X		
6	T1	4	9	read	Y		
7	T2	5	10	write	X	30	70
8	T1	6	9	write	Y	110	130
9	T1	9	-	commit			
10	T2	7	-	commit			

Exercício

S

```
início T1
ler(X)
X = X - N
início T2
gravar(X)
ler(X)
X = X + M
ler(Y)
gravar(X)
Y = Y + N
gravar(Y)
commit
commit
```

Que ações o banco de dados deve tomar para garantir a consistência do banco de dados no caso de falhas independentes nos pontos marcados no plano abaixo? Quais informações são necessárias para desempenhar as ações?

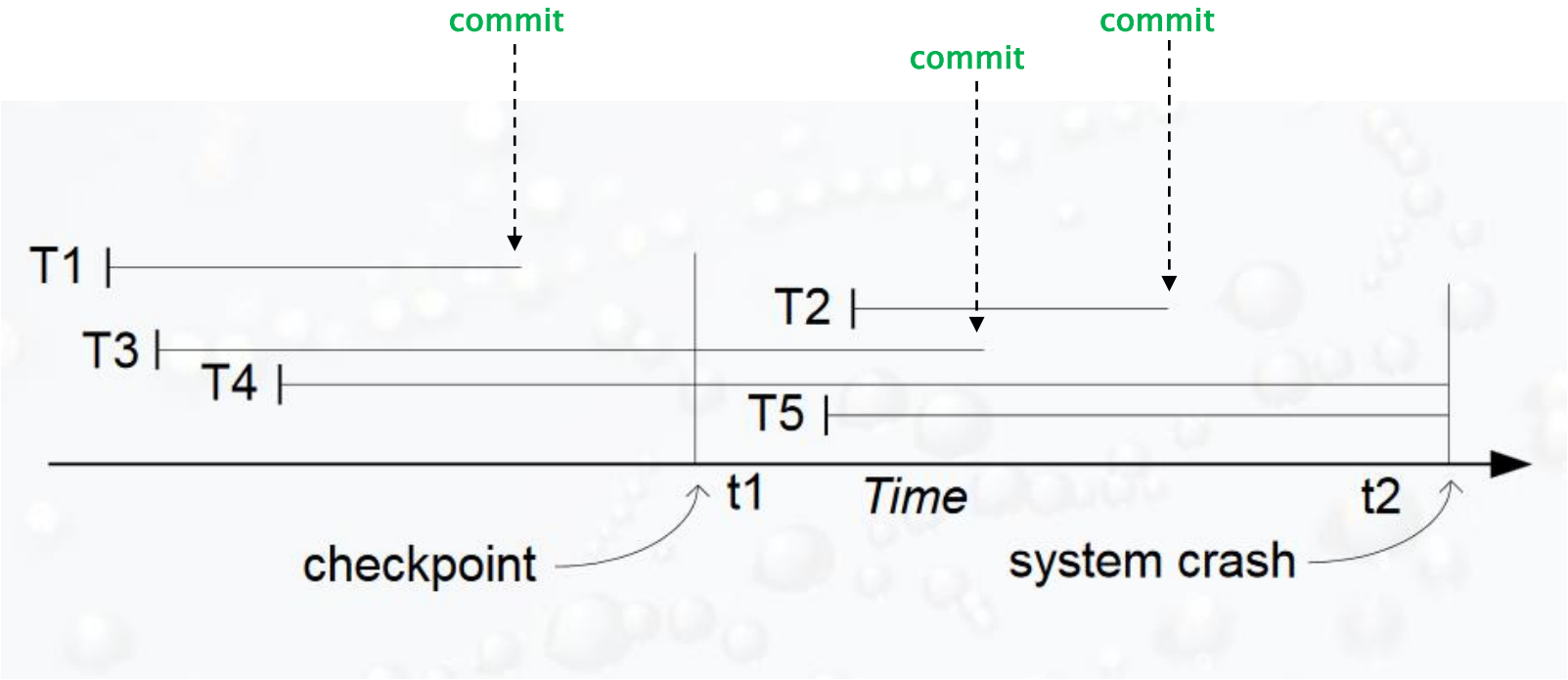
Falha (a)

Falha (b)

Registro de CHECKPOINT

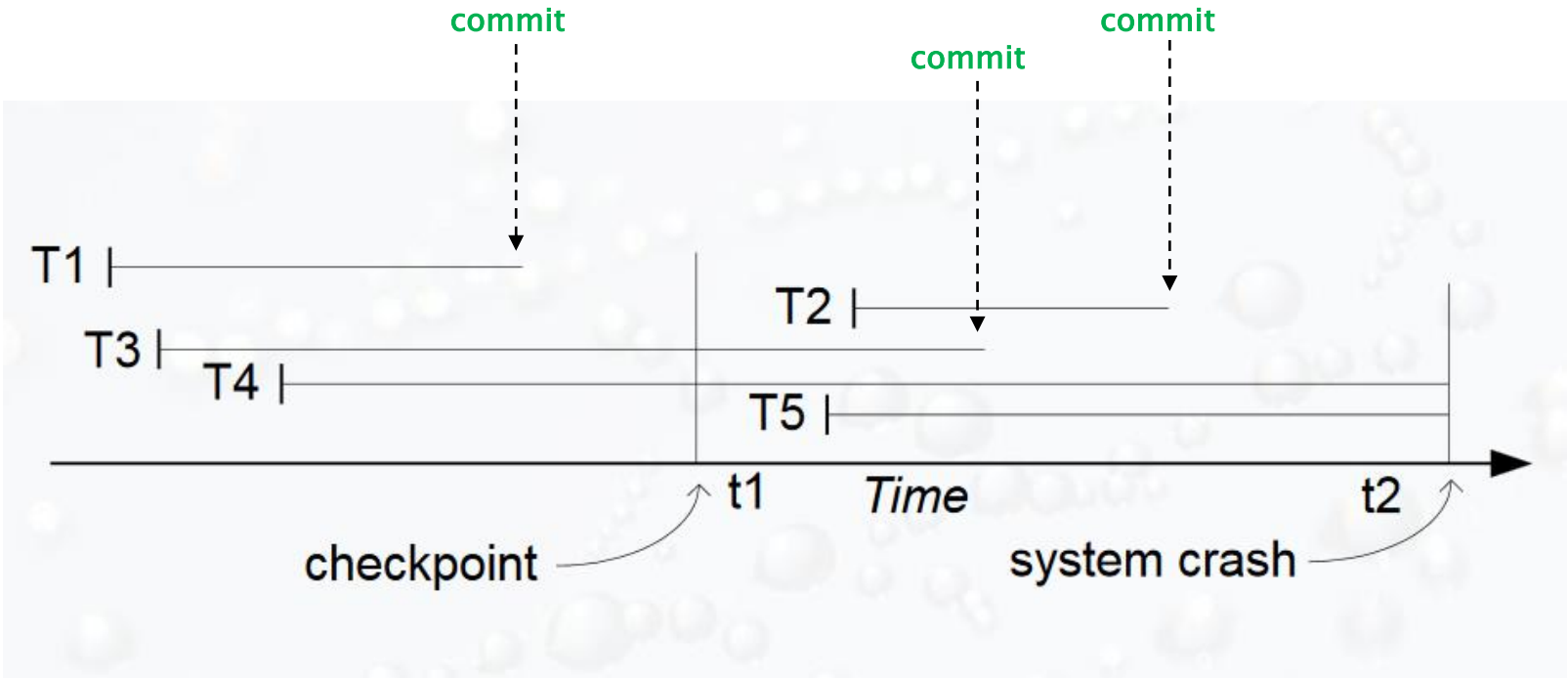
- É um dos tipos de registros gravados no arquivo de **log**;
- Um registro **CHECKPOINT** contém uma lista de todas as transações ativas que foram gravadas permanentemente em disco;
- Periodicamente o **SGBD** efetua um **FLUSHING** de transações confirmadas (**committed**) de buffer de memória principal para os discos do banco de dados (gravação física).

Recuperação



Qual o procedimento a ser feito para a recuperação das transações ?

Recuperação



- ✓ **T1** não precisa ser refeita (terminou antes do checkpoint);
- ✓ **T3** deve ser refeita do ponto de checkpoint em diante;
- ✓ **T2** deve ser refeita;
- ✓ **T4** e **T5** devem ser desfeitas;