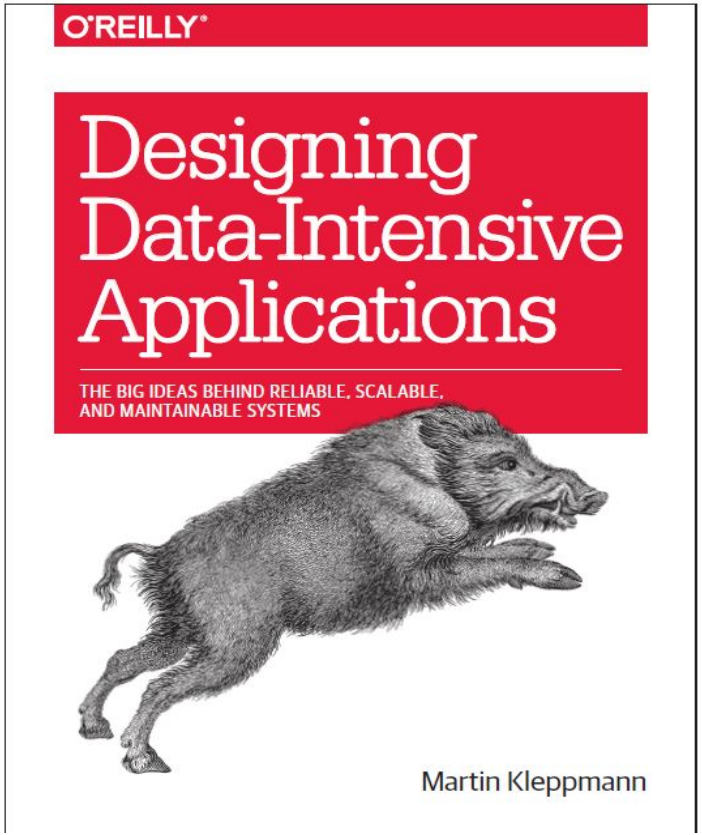
TRANSAÇÕES



Capítulo 7

A dura realidade dos sistemas de dados

concorrência entre clientes

hardware

interrupções na rede

a aplicação

clientes podem sobrescrever

software de bd

dados parcialmente atualizados

TRANSAÇÃO

Como garantir que a **solução funciona**?

- Pensar em **todas** as coisas que podem dar erradas
- Fazer **grande** quantidade de testes

Simplificando

Uma transação agrupa vários comandos de *read* e *write* em uma unidade lógica

Já que os bancos de dados garantem:

- segurança da integridade
- sigilo
- autenticidade

TRANSAÇÕES

Uma transação é executada **como uma única operação**:

(***commit*** - ***confirmada***)
a transação inteira
é bem-sucedida

ou (**abort**, **rollback**)
a transação inteira
falha

TRANSAÇÕES -**Operações de único-objeto ou com vários objetos**

Atomicidade

- **não** pode ser **dividido** em partes menores

- habilidade de **anular** a transação por **erro** e todas as escritas serem **descartadas**

Consistência

- A aplicação define **quais dados** (invariantes) são válidos ou inválidos Ex: Total

- quais **comandos** sobre seus dados devem ser sempre *verdade*

Isolamento

- maneira de se tratar **concorrência**

- as transações s**imultaneamente** em execução são isoladas uma das outras

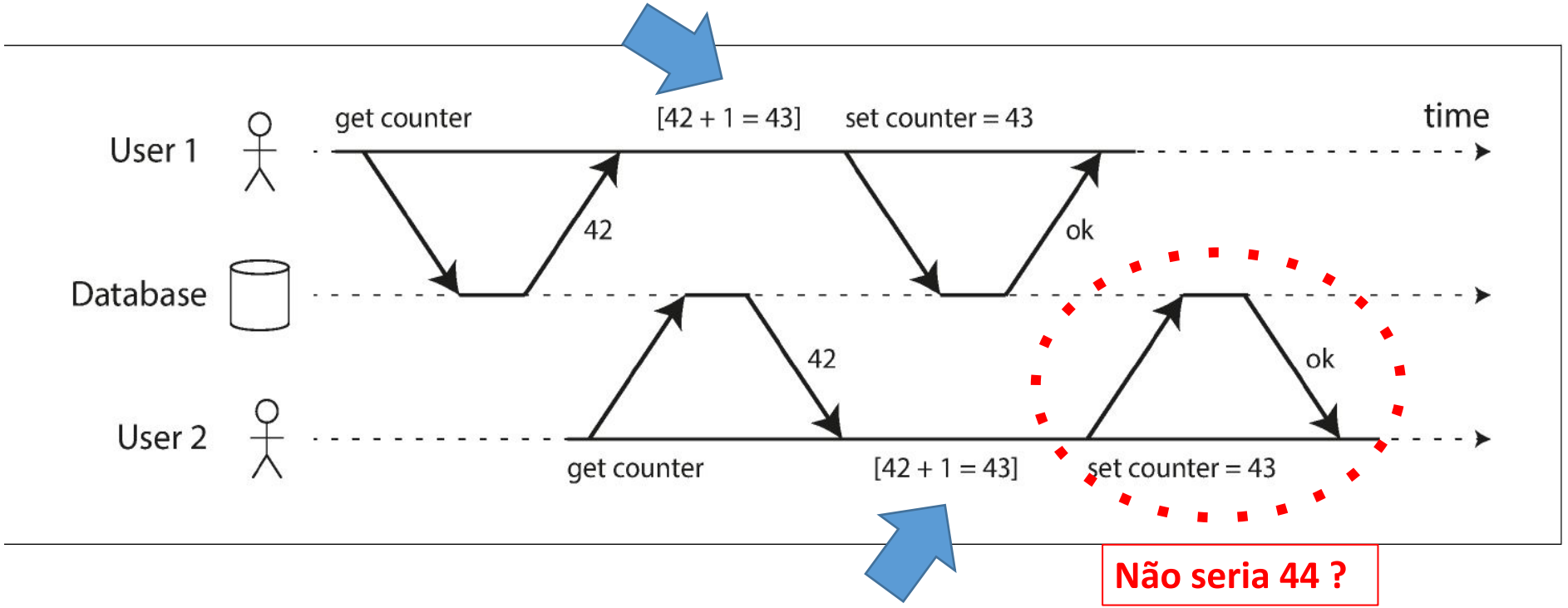
Durabilidade

- Se uma transação teve ***commit***, todos os dados que foram escritos **não serão esquecidos**, mesmo que o hardware falhe ou ocorra *crash* de bd

propriedade da aplicação

propriedade do BD

TRANSAÇÕES - **Isolamento**



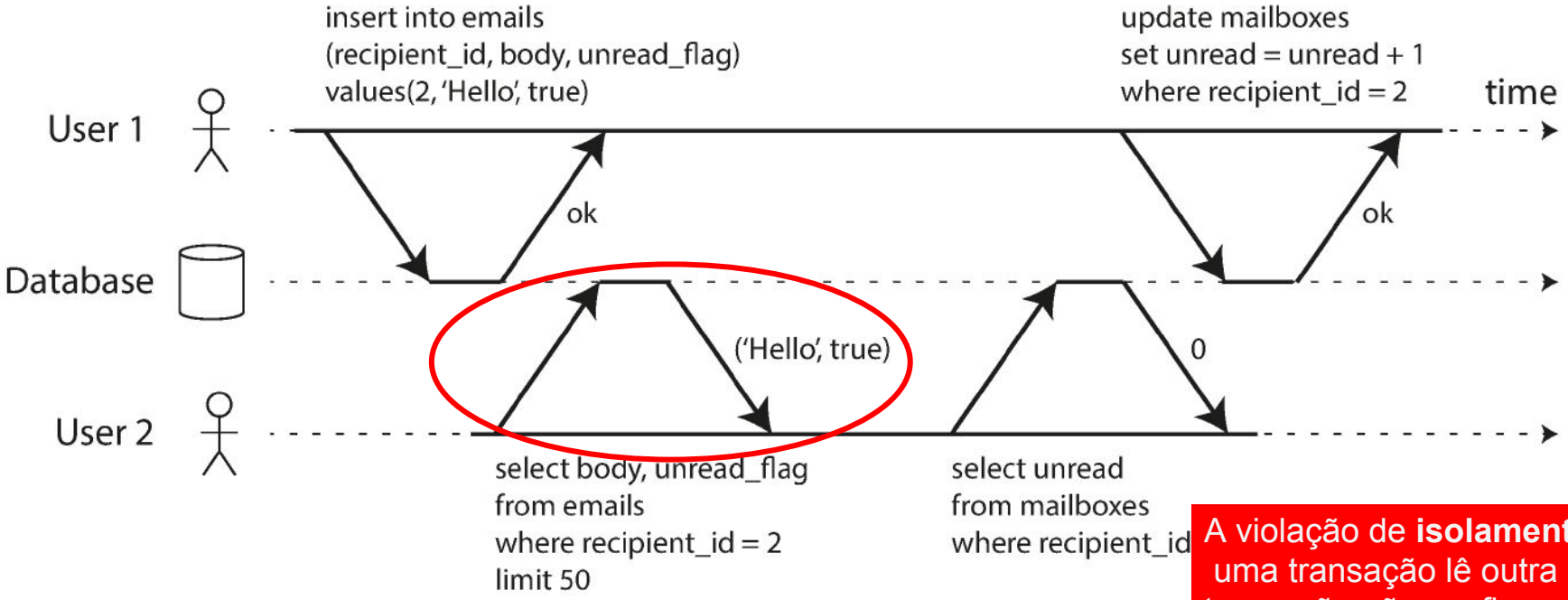
Condição de concorrência:

2 clientes simultaneamente incrementando um contador sem isolamento

(usado isolamento serializável - fingir que é a única transação ou isolamento instantâneo(snapshot)

TRANSAÇÕES - **Operações com vários objetos**

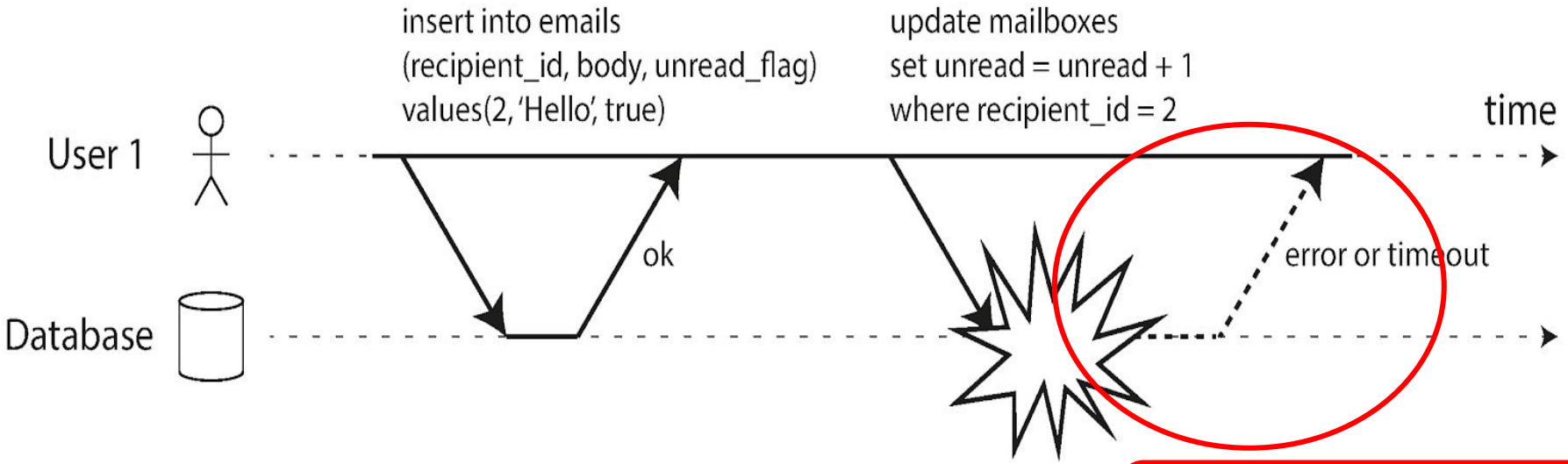
Isolamento - Operações executadas simultaneamente não devem interferir entre si



A violação de **isolamento**: uma transação lê outra transação não confirmada

TRANSAÇÕES - **Operações com vários objetos**

Atomicidade - Quando ocorre um **erro no meio do caminho** de uma sequência de escritas, a transação deve ser **abortada**, e as escritas feitas até aquele ponto, devem ser **descartadas**



Necessário RollBack
Senão, fora de sincronia

TRANSAÇÕES - **Operações com um único objeto**

O que o BD deve fazer se um cliente faz **várias escritas dentro da mesma transação**. Também se aplicam quando **um único objeto** que está sendo alterado

Mecanismos de armazenamento fornecem **atomicidade** e **isolamento** no nível de um único objeto (como um key-value par) em um nodo

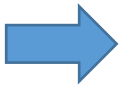
- **Atomicidade:** implementada usando um **log para recuperação** de *crash*

- **Isolamento:** implementado usando um **bloqueio (*lock*) em cada objeto** (permitindo que apenas uma thread acesse um objeto em um dado momento)

TRANSAÇÕES - **Operações com um único objeto**

Em um modelo de dados em um **documento:**

- Campos que precisam ser atualizados juntos estão muitas vezes **dentro do mesmo documento**

  o que é tratado como um **único objeto!**

- **Nenhuma transação multi-objeto é necessária** quando a atualização é de um único documento.

TRANSAÇÕES - **A necessidade de transações multi-objeto**

- **Modelo de dados relacional - Chaves estrangeiras têm que estar corretas e atualizadas**

- **Modelo com documento** dados desnormalizados precisam estar em sincronia

- **BDs com índices secundários** precisam ser **sempre atualizados**

TRANSAÇÕES - **Anulações e tratamento de erros**

- Uma transação pode ser **anulada e repetida**, com **segurança**, se um erro ocorreu

- Bancos de dados ACID

perigo de violar a **garantia de atomicidade** **isolamento** ou **durabilidade**

abandona a transação inteiramente

TRANSAÇÕES - **Anulações e tratamento de erros**

- Tentar novamente uma transação **anulada** é simples e eficaz mas não perfeito

TRANSAÇÕES - **Anulações e tratamento de erros**

- Tentar novamente uma transação **anulada** é simples e eficaz mas não perfeito

Transação bem-sucedida
A rede falhou enquanto o servidor tentou reconhecer o commit
para que o cliente acha que falhou

É executada **duas vezes**!

TRANSAÇÕES - **Anulações e tratamento de erros**

- Tentar novamente uma transação **anulada** é simples e eficaz mas não perfeito

Transação bem-sucedida
A rede falhou enquanto o servidor tentou reconhecer o commit
para que o cliente acha que falhou

É executada **duas vezes**!

- Necessário criar um mecanismo de **eliminação da duplicação** de nível de aplicativo

TRANSAÇÕES - **Anulações e tratamento de erros**

- Tentar novamente uma transação **anulada** é simples e eficaz mas não perfeito

TRANSAÇÕES - **Anulações e tratamento de erros**

- Tentar novamente uma transação **anulada** é simples e eficaz mas não perfeito

- **limitar o número de tentativas** com backoff exponencial e manipulando erros relacionados a sobrecarga de forma diferente de outros erros

TRANSAÇÕES - **Anulações e tratamento de erros**

- Só vale a pena tentar novamente após erros transitórios

bloqueio, isolamento violação, interrupções de rede temporário e failover

- Depois de um **erro permanente** uma nova tentativa seria inútil. Ex: Violação de Restrição

TRANSAÇÕES - **Anulações e tratamento de erros**

- Se a transação também tem efeitos secundários fora do bd podem acontecer mesmo se a transação for anulada

- Se é uma certeza que vários sistemas diferentes ou cometem ou abortar juntos, pode-se usar a confirmação (commit) de duas fases

- Se o processo do cliente falhar ao tentar novamente, todos os dados que estava tentando escrever para o banco de dados são **perdidos**

TRANSAÇÕES - **Níveis de isolamento fraco**

- Se duas transações não tocam os mesmos dados, podem **com segurança ser executadas em paralelo**

- **Problemas de concorrência** só entram em jogo quando uma transação lê dados que simultaneamente são modificados por outra transação ou tenta modificá-los

TRANSAÇÕES - **Níveis de isolamento fraco**

- **Bugs de concorrência** são difíceis de encontrar por meio de testes
- Sistemas usam níveis **mais fracos de isolamento**, que protegem contra alguns problemas de simultaneidade, mas não todos
- **Os bugs de concorrência** causados por um **isolamento fraco** de transação causam

perda substancial de dinheiro ou de dados dos consumidores

TRANSAÇÕES - **Leitura confirmada - commited**

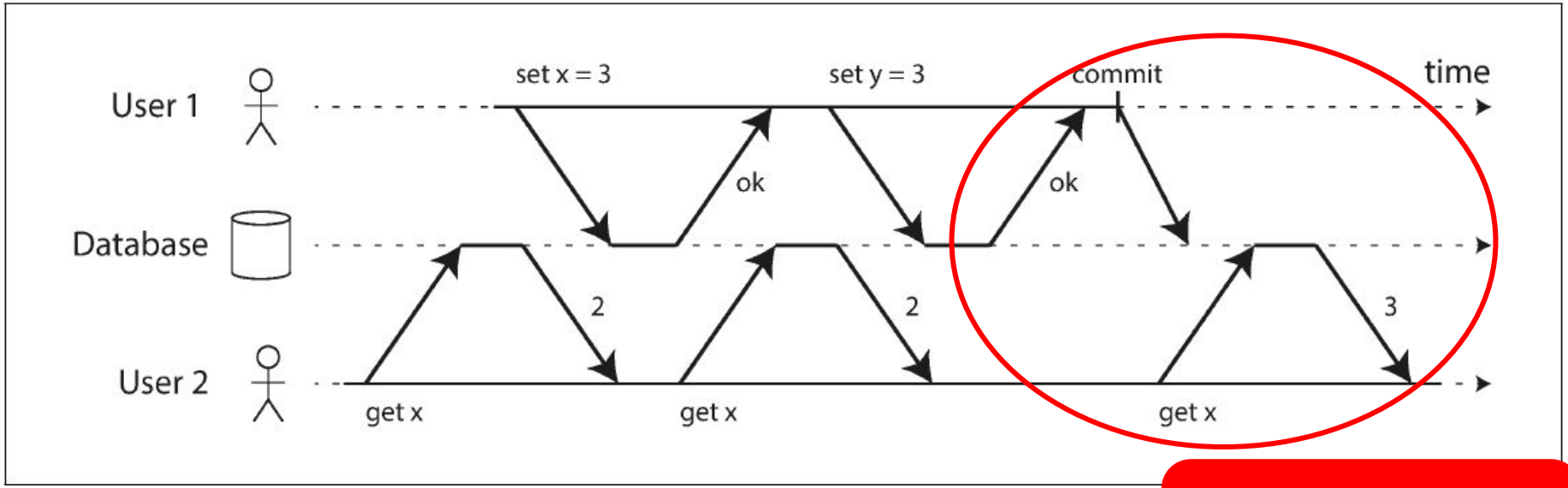
O **nível mais básico de isolamento** da transação é a **leitura confirmada**
Ela faz duas garantias:

- Na leitura do banco de dados, somente serão **mostrados dados que tem *commit*** (não leituras sujas)

- a escrita no banco de dados, somente serão **sobrescritos dados que tem *commit*** (não escreve sujo)

TRANSAÇÕES - **Leitura confirmada (commited)**

- Chama-se **leitura suja** ocorre quando outra transação enxerga dados não confirmados
- Transações executando no nível isolamento de **leitura *commited*** devem evitar leituras sujas

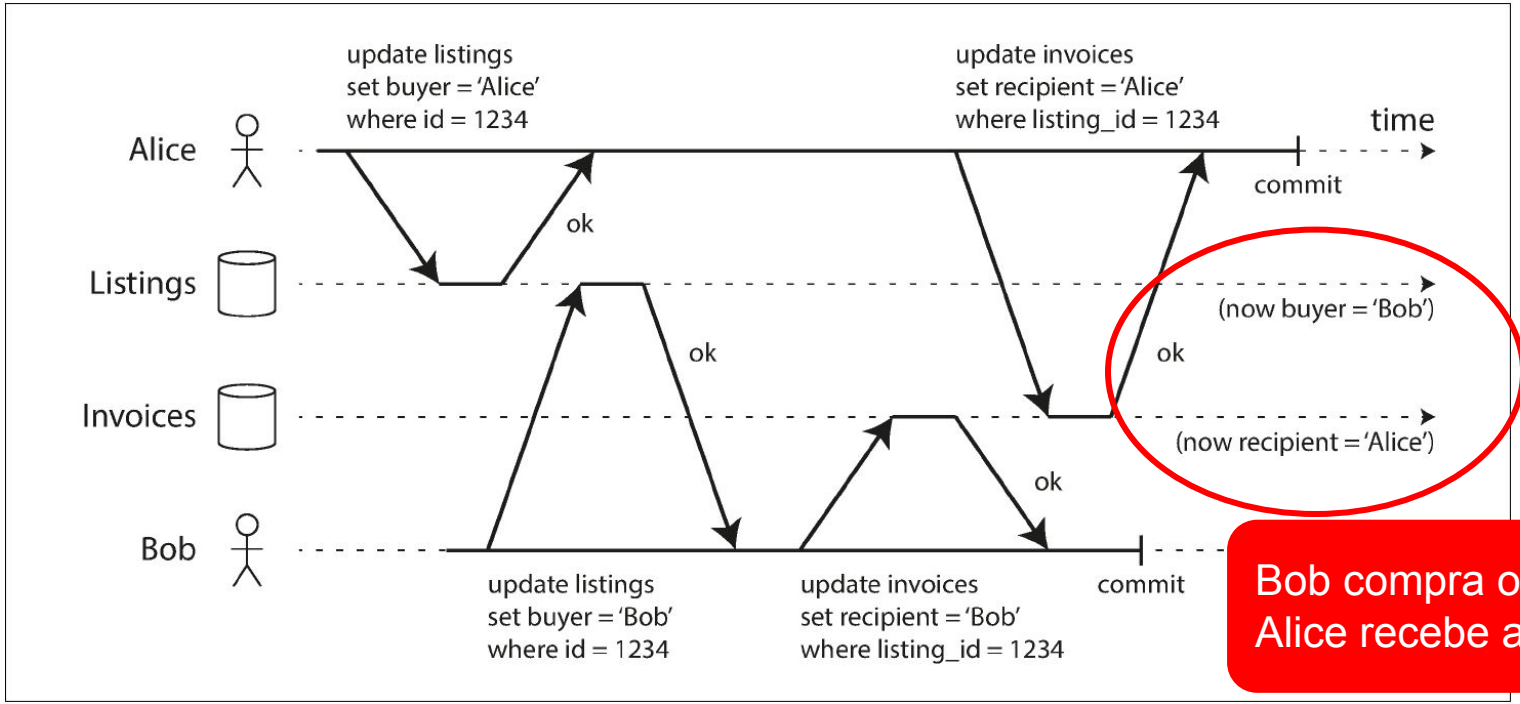


Usuário 2 só vê x=3, após commit

TRANSAÇÕES – **Escrita commited**

- **Escrita suja** : gravação anterior é parte de uma transação que ainda não foi confirmada *(commited)* então a escrita mais tarde substitui um valor não confirmado

- **Isolamento commited** atrasa segunda escrita até a primeira transação de escrita ser confirmada ou anulada

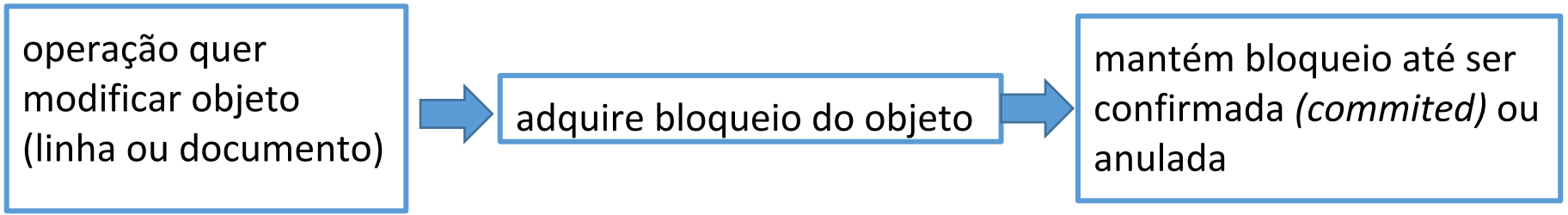


TRANSAÇÕES – **Implementar a leitura confirmada (commited)**

- **Leitura confirmada** é um nível de isolamento muito popular

É a configuração padrão no Oracle 11g, PostgreSQL, SQL Server 2012, MemSQL ...

- Bds impedem **escritas sujas** usando bloqueios de nível de linha:



Apenas uma transação pode manter o bloqueio para qualquer dado objeto

TRANSAÇÕES – **Implementar a leitura confirmada (commited)**

- Como evitar **leituras sujas**?

TRANSAÇÕES – **Implementar a leitura confirmada (commited)**

- Como evitar **leituras sujas**?

Uma opção

1. Usar a mesma **fechadura *(lock)***
2. Exigir que qualquer transação que ler um objeto adquira o **lock** - **brevemente**
3. Liberar o **lock** **imediatamente** depois da leitura

TRANSAÇÕES – **Implementar a leitura confirmada (commited)**

- Como evitar **leituras sujas**?

Uma opção

1. Usar a mesma **fechadura *(lock)***
2. Exigir que qualquer transação que ler um objeto adquira o **lock** - **brevemente**
3. Liberar o **lock** **imediatamente** depois da leitura

Garante
que a leitura não ocorre enquanto um objeto tem um **valor sujo, não confirmado**

TRANSAÇÕES – **Implementar a leitura confirmada (commited)**

- Como evitar **leituras sujas**?

Uma opção

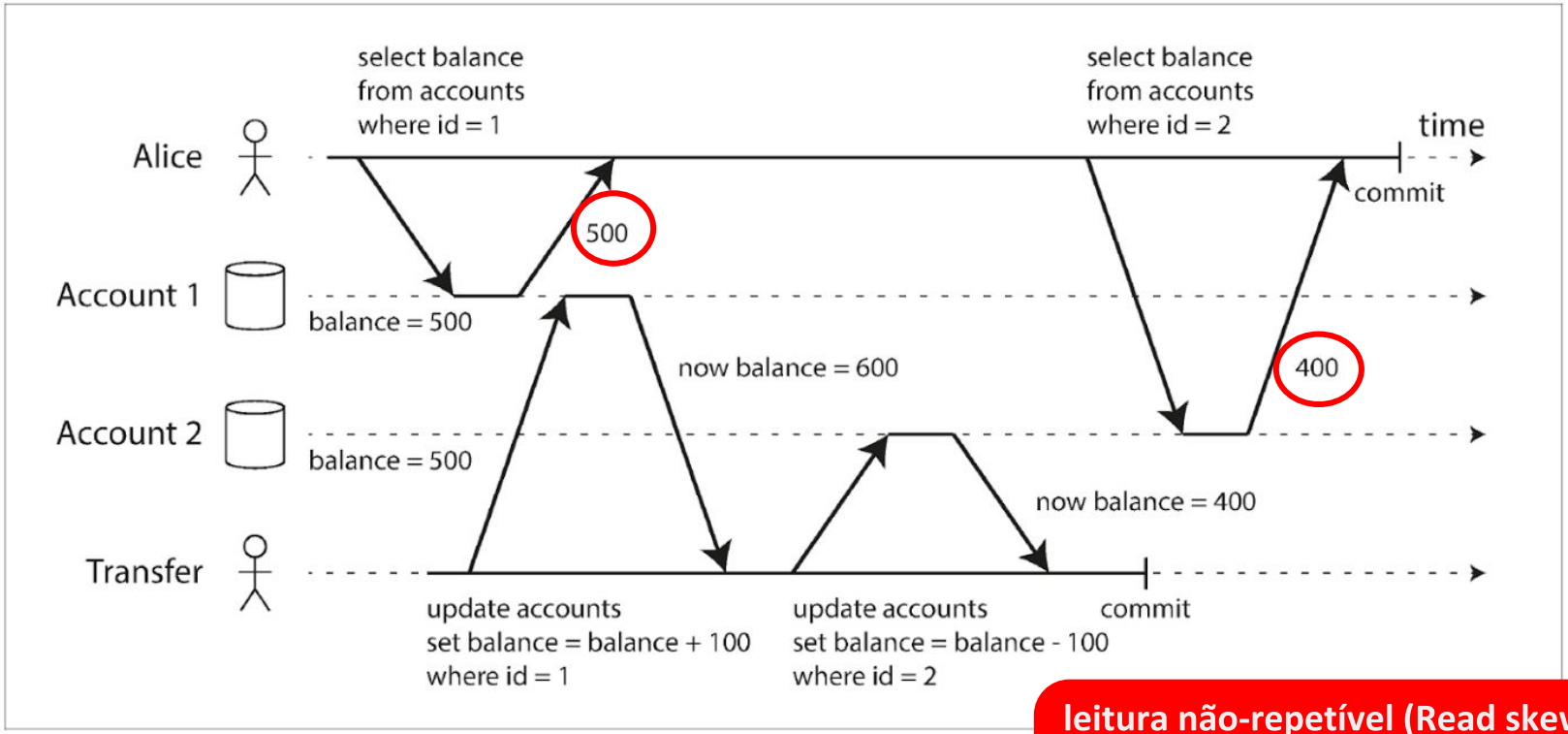
1. Usar a mesma **fechadura *(lock)***
2. Exigir que qualquer transação que ler um objeto adquira o **lock** - **brevemente**
3. Liberar o **lock** **imediatamente** depois da leitura

Garante
que a leitura não ocorre enquanto um objeto tem um **valor sujo, não confirmado**

- **No entanto**, não funciona bem na prática

Uma **longa transação de escrita** pode forçar que **muitas transações com somente leitura** esperem até que seja concluída

TRANSAÇÕES – **leitura repetida** (Read skew)



leitura não-repetível (Read skew)

Inicial 500+500 = 1000
Para Alice 500+400=900

TRANSAÇÕES – **Consultas analíticas e verificações de integridade**

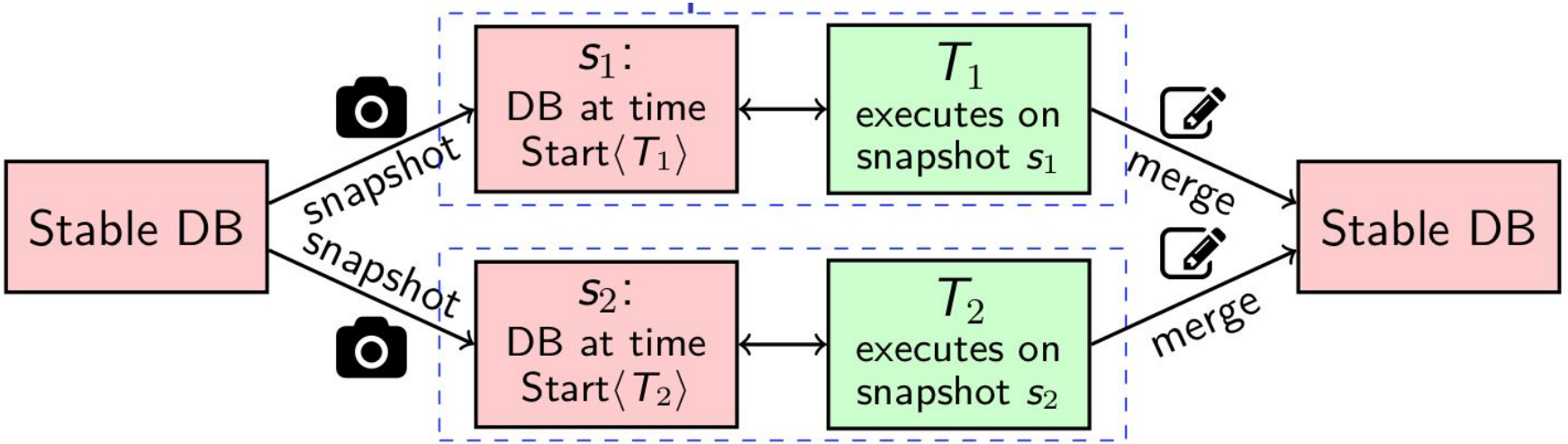
Um backup

- Requer uma cópia do bd inteiro (**pode levar horas**)
- Durante a execução processo de backup a escrita continuará a ser feita no BD
- Assim, é possível ter algumas partes do backup que contém uma **versão mais antiga** dos dados e outras partes que contém uma **versão mais recente**
- Se for preciso **restaurar a partir de um backup**, a inconsistência torna-se permanente

Consultas analíticas e verificações de integridade

- Consultas que varrem sobre grandes partes do bd são **comuns em análises**
- Podem também ser parte de uma verificação da integridade (**monitoramento de corrupção de dados**)
- São **propensas** a retornar resultados absurdos se observam partes do bd em **diferentes momentos**

TRANSAÇÕES – **isolamento snapshot**



TRANSAÇÕES – **isolamento snapshot**

- Cada transação lê em um instante consistente do bd - a transação vê todos os dados que foram **confirmados *(commited)*** no bd no seu início

- Mesmo se os dados são **posteriormente alterados** por outra transação, ela vê apenas os dados antigos daquele ponto específico no tempo

- **Isolamento snapshot** é bom para consultas de execução demorada, com somente leitura (**backups e análises**)

TRANSAÇÕES – **isolamento snapshot**

- É muito difícil falar sobre o **significado de uma consulta**, se os dados nos quais ela opera estão mudando ao mesmo tempo que a consulta está em execução

- Quando a transação pode ver um instantante consistente do bd, **congelado em um ponto específico no tempo**, é muito mais fácil de entender

- **Isolamento Snapshot** é uma característica popular: implementado pelo PostgreSQL, MySQL com o mecanismo de armazenamento InnoDB, Oracle, SQL Server

TRANSAÇÕES – **Implementação de isolamento snapshot**

- De um ponto de vista do **desempenho**, é um princípio-chave de isolamento snapshot "leitores nunca bloqueiam os escritores", e "escritores nunca bloqueiam os leitores“

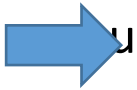
- BD usam uma generalização do mecanismo para evitar **leituras sujas**

- O bd deve manter **várias versões confirmadas (*commited*)** diferentes de um objeto, porque várias operações em andamento podem precisar ver o estado do bd em diferentes pontos no tempo

- Por manter **várias versões de um objeto** ao lado, esta técnica é conhecida como controle de concorrência de multiversão (*multiversion concurrency control (MVCC)*).

TRANSAÇÕES – (*multiversion concurrency control (MVCC)*).

- Se só precisasse fornecer isolamento de leitura confirmada *(commited)*, mas não o isolamento snapshot



uficiente **duas versões de um objeto**:

a versão confirmada e a substituída-mas-não-ainda-confirmada

- Geralmente, mecanismos de armazenamento que **implementam o isolamento snapshot** usam MVCC para o nível de isolamento de leitura confirmada*(commited)*

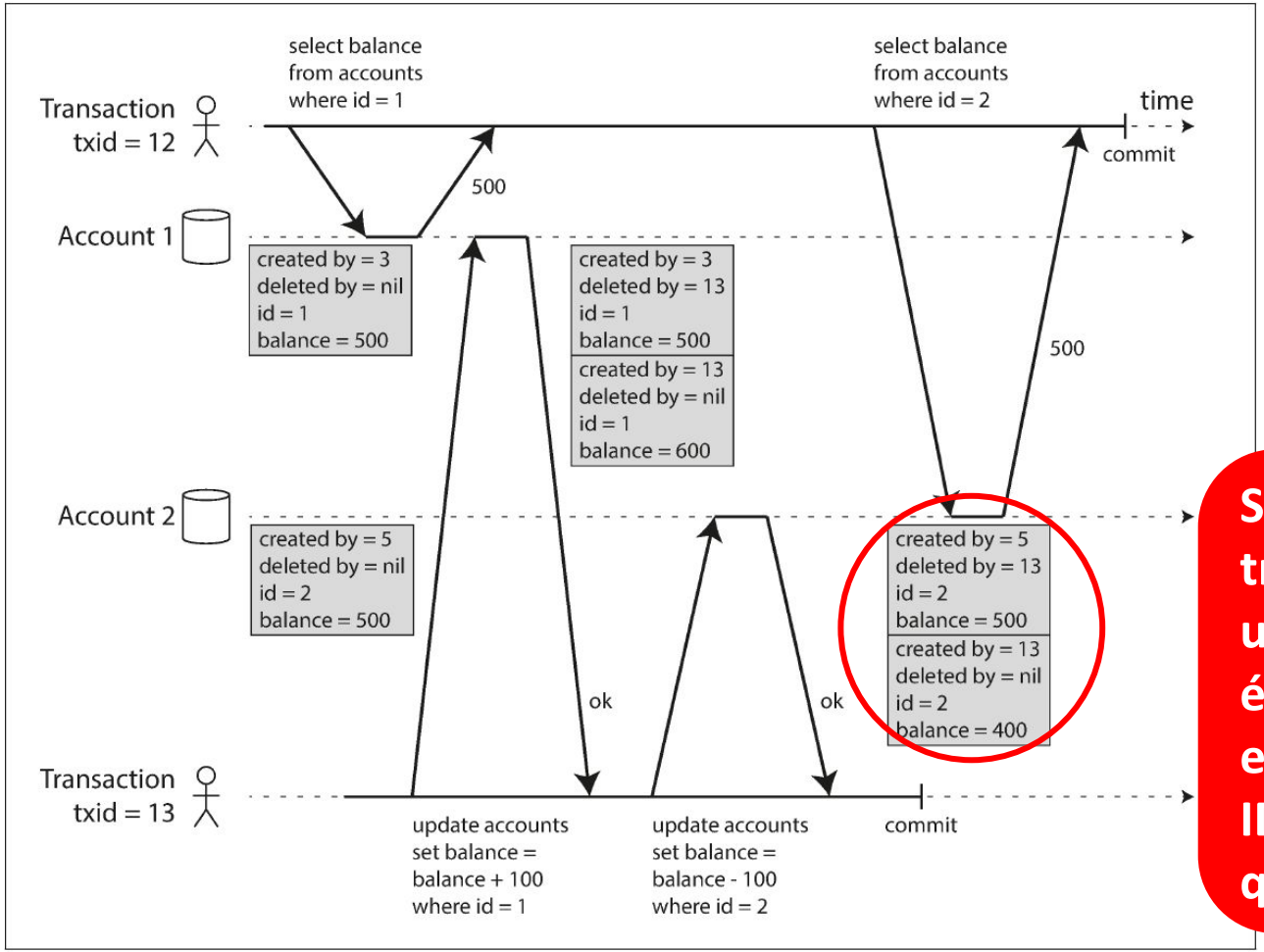
- Uma abordagem típica é a **leitura confirmada***(commited)* usar um instante separado para cada consulta

- O **isolamento snapshot** usa a mesma cópia para uma transação inteira

TRANSAÇÕES – (*multiversion concurrency control (MVCC)*).

- MVCC-baseado em isolamento snapshot é implementado no **PostgreSQL**
- Cada transação tem ID único
- Os dados que a transação escreve são marcados com o seu ID

TRANSAÇÕES – (*multiversion concurrency control (MVCC)*).



Se uma transação exclui uma linha, é marcada a exclusão com o ID da transação que solicitou

TRANSAÇÕES – **Regras de visibilidade para observar um snapshot consistente**

Regras para a **criação e exclusão de objetos** para um snapshot consistente:

1. No início de cada transação, o BD faz uma lista de todas as outras **operações que estão em andamento**. Quaisquer escritas que fizerem são ignoradas, mesmo se as transações posteriormente foram **confirmadas**

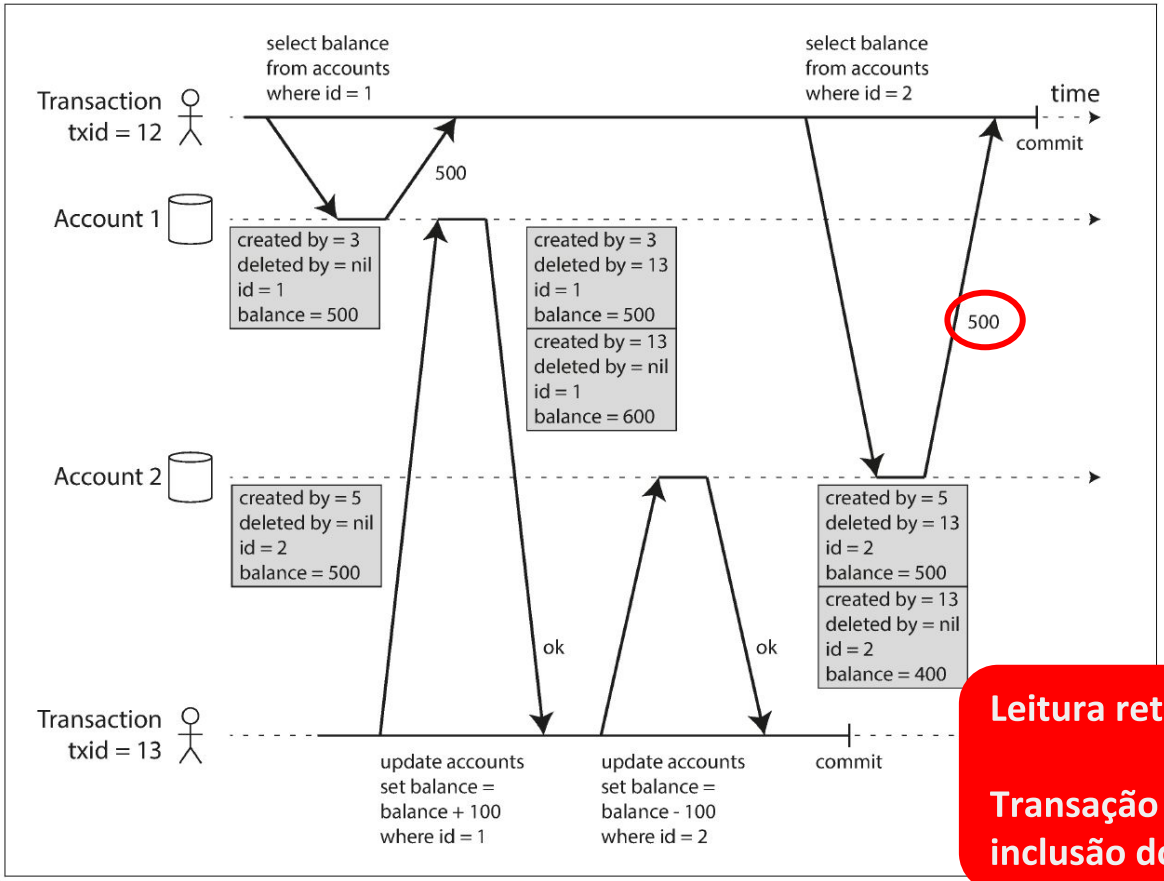
2. Escritas feitas por **transações abortadas** são ignoradas

3. Escritas feitas por transações com um **ID posterior** são ignoradas, mesmo se essa transações foram confirmadas

4. Todas as **outras escritas** são visíveis para consultas do aplicativo

Os **IDs da transação** são usados para decidir quais os objetos pode ver e quais são invisíveis

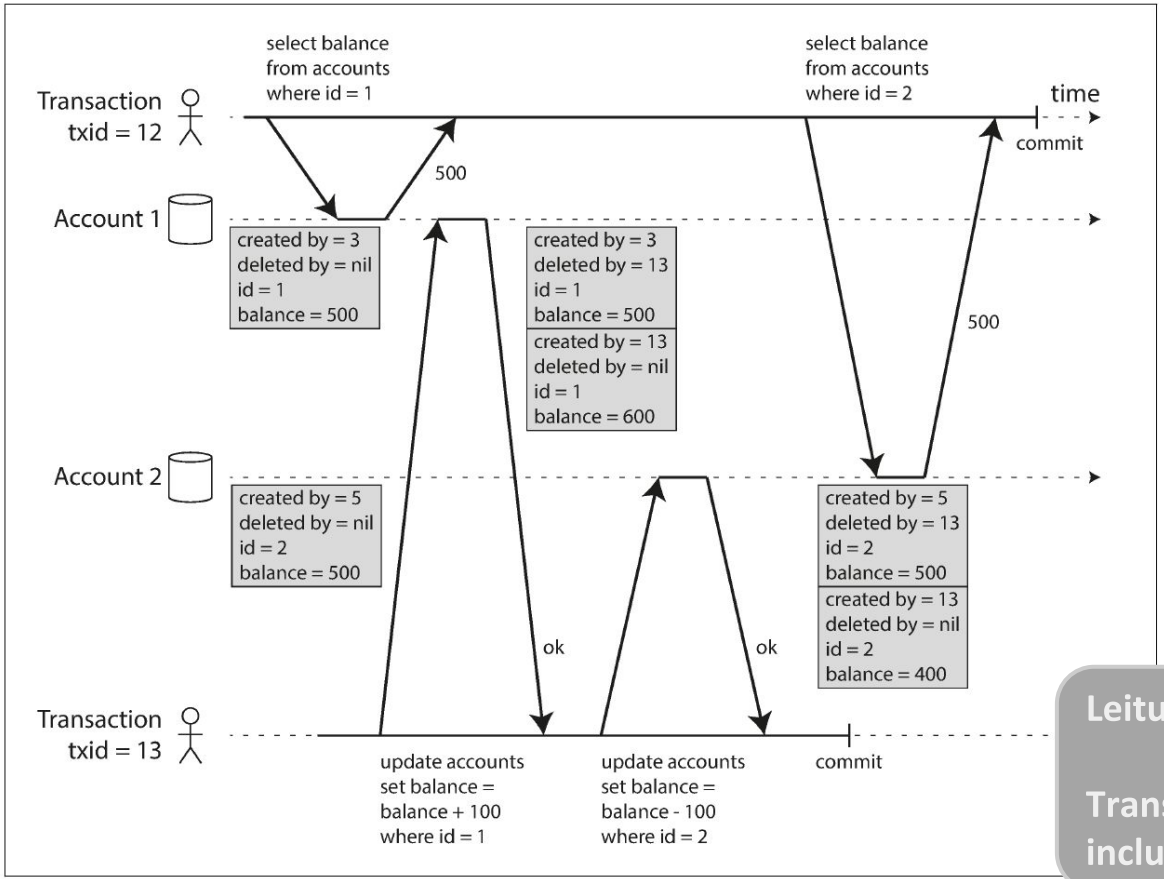
TRANSAÇÕES – **Regras de visibilidade para observar um snapshot consistente**



Leitura retorna 500

Transação 12 não ve exclusão nem inclusão do saldo 400 pela transação 13

TRANSAÇÕES – **Regras de visibilidade para observar um snapshot consistente**



Um **objeto é visível** se:

- Quando **iniciou a transação** de leitura, a que criou o objeto já tinha confirmado
- O objeto **não está marcado para exclusão**, ou a transação que solicitou a exclusão ainda não tinha confirmado

Leitura retorna 500

Transação 12 não ve exclusão nem inclusão do saldo 400 pela transação 13

TRANSAÇÕES – **Índices e isolamento snapshot**

Como funcionam os **índices** em um bd multi-versão?

TRANSAÇÕES – **Índices e isolamento snapshot**

Como funcionam os **índices** em um bd multi-versão?

Uma opção
fazer com que o índice aponte para todas as versões de um objeto e exija **uma consulta de índice** para filtrar quaisquer versões do objeto que **não são visíveis** para a transação atual

Quando a **coleta de lixo remove as versões antigas** do objeto, as entradas dos índices correspondentes podem ser removidas

TRANSAÇÕES – **Operações de gravação atômicas**

- **Atualização atômicas** removem a necessidade de implementar ciclos ler-modificar-escrever

- São geralmente a melhor solução se é possível utilizar

- Porém, nem todas as escritas podem ser facilmente expressas em termos de **operações atômicas**

- Geralmente implementadas por um **bloqueio exclusivo** no objeto quando ele é lido de forma que nenhuma outra transação pode lê-lo até que a atualização tenha sido feita. Esta técnica é conhecida como *cursor estabilidade*

TRANSAÇÕES –**Travamento explícito**

- Para **impedir atualizações perdidas**:
  é **função do aplicativo** trancar explicitamente objetos que vão ser atualizados
  se as **operações atômicas** embutidas no BD não fornecem a funcionalidade necessária

- Neste caso, **uma operação** atômica pode não ser suficiente

- Em vez disso, é possível usar **um bloqueio para impedir** que dois jogadores movam simultaneamente a mesma peça

Detecção automática de perda de atualizações

- **Operações atômicas e bloqueios** impedem atualizações perdidas pois forçam os ciclos de leitura modificar-escrever a **acontecer sequencialmente**

- Uma alternativa é permitir que **executem em paralelo**. Se for detectada uma atualização perdida, **aborta a transação** e **força a repetir** o seu ciclo de leitura-modificação-escrita

Compare-and-set

- O objetivo é **evitar atualizações perdidas**, permitindo que uma atualização aconteça somente se o valor não foi alterado desde a última vez que lê-lo. Se o valor atual não corresponde ao que foi lido anteriormente, a atualização **não tem nenhum efeito**, e o ciclo de leitura modificação-escrita deve ser **repetida**

Write Skew and Phantoms

Write Skew

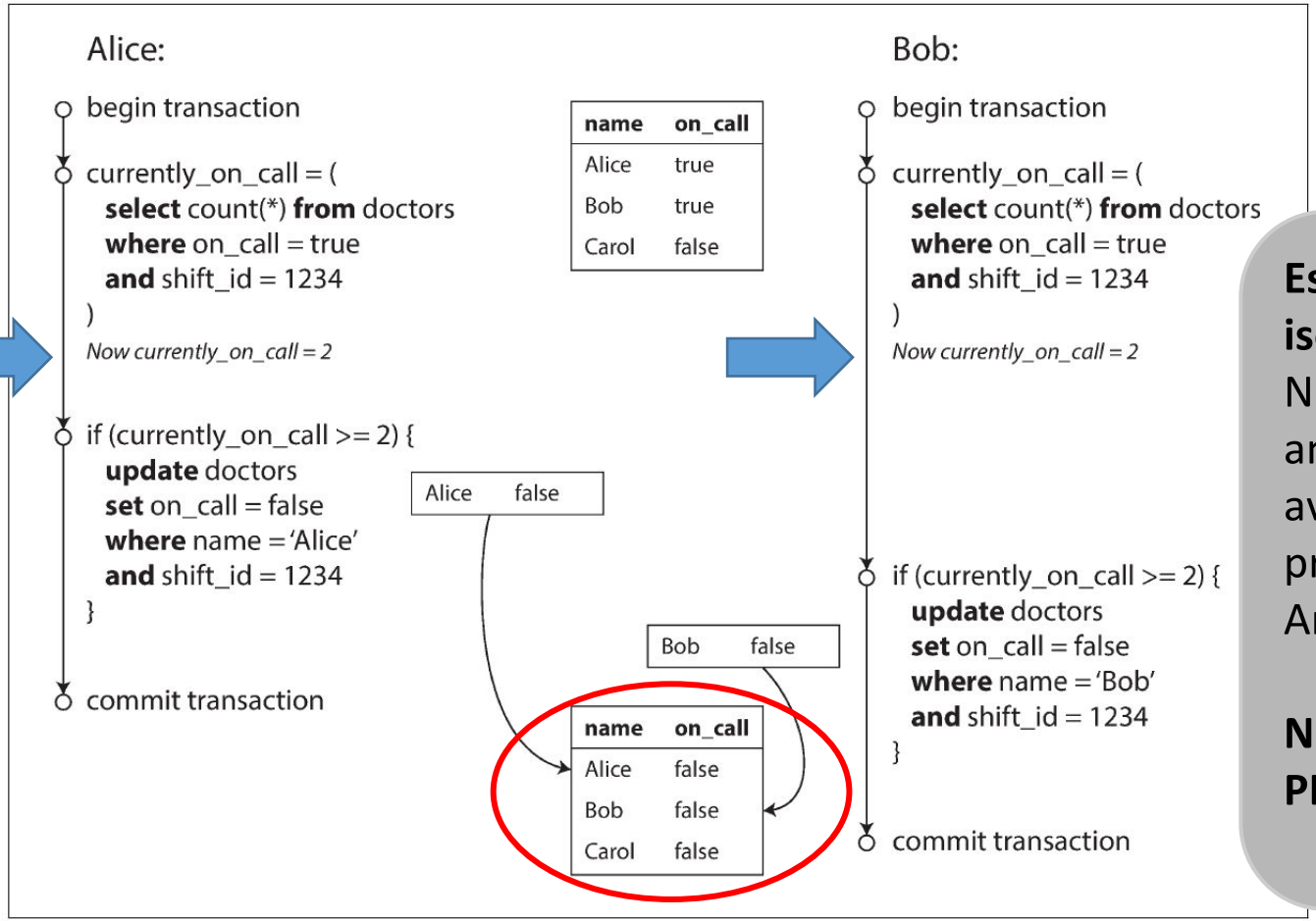
- A fim de **evitar a corrupção de dados**, essas condições de corrida, como *escritas sujas* e *atualizações perdidas*, precisa ser **prevenidas**
- Sendo automaticamente salvas pelo banco de dados
- por manuais, tais como usando bloqueios ou operações de escrita atômicas

Phantoms

- A T1 lê um conjunto C1 de dados, usando um argumento
- A T2 modifica os dados usando o mesmo argumento
- A T1 lê novamente e obtém um conjunto C2

- C1 ≠C2 Obtém resultado diferente
- O conteúdo que "aparece e desaparece" é chamado de Phantom

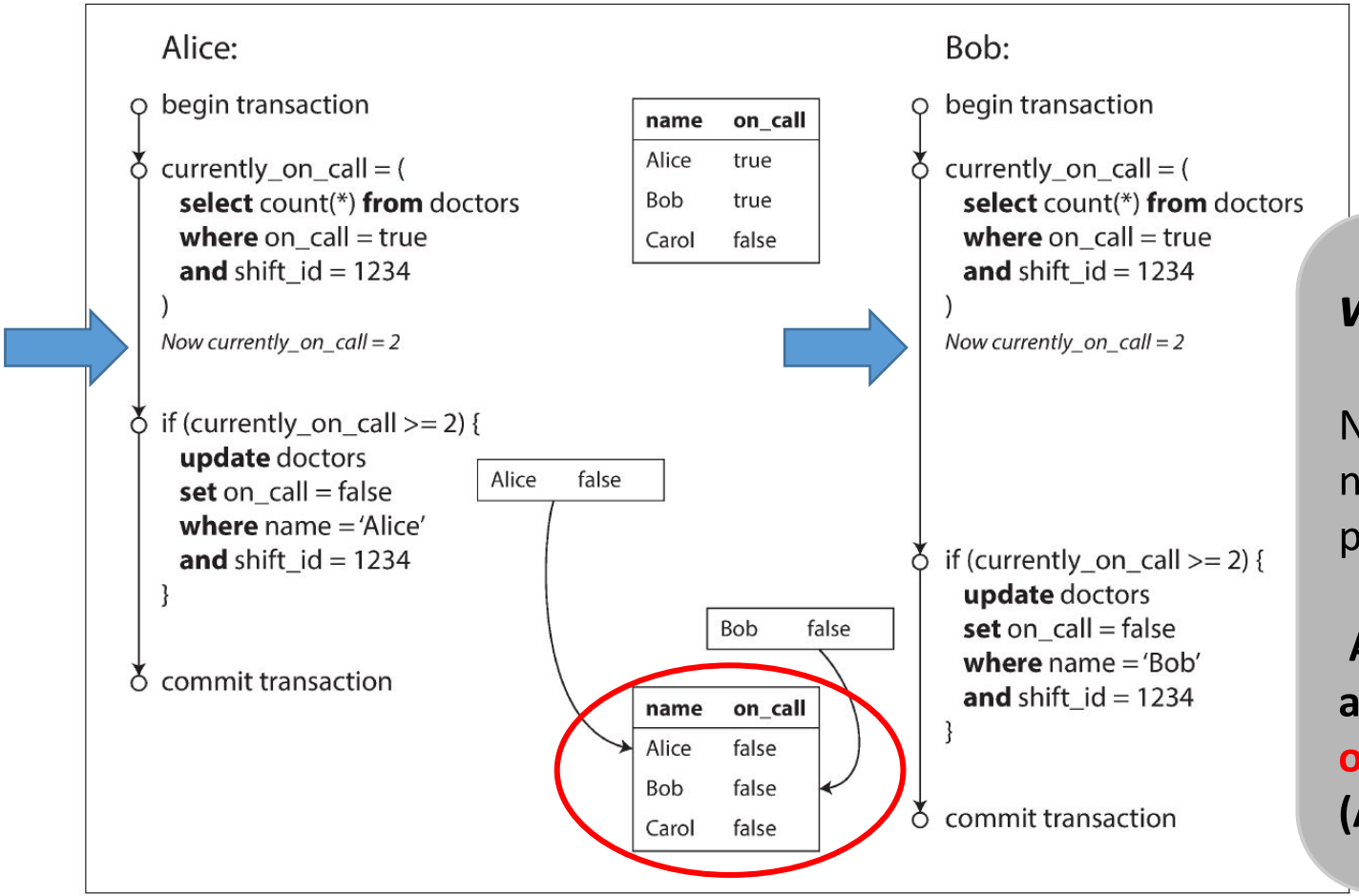
TRANSAÇÕES –**Write Skew**



Está usando o isolamento snapshot:
Na leitura retorna 2, ambas as transações avançam para a próxima fase.
Ambas confirmam.

NENHUM MÉDICO DE PLANTÃO

TRANSAÇÕES –**Write Skew**



write skew

Não é gravação suja nem atualização perdida

As transações estão atualizando dois objetos diferentes (Alice e Bob)

TRANSAÇÕES –**Write Skew**

Com *write skew*, as opções são mais restritas:

- Operações atômicas de objeto único não ajudam
- A detecção automática de atualizações perdidas também não ajuda
- Para especificar que pelo menos um médico deve ficar de plantão, era preciso uma **restrição que envolve vários objetos**. A maioria dos BD não tem suporte, mas é possível implementá-las com **gatilhos ou visões materializadas**
- Se não for possível usar um **nível de isolamento serializável**, realizar o travamento explícito das linhas que a transação depende

TRANSAÇÕES – **Isolamento serializável**

- Tem sido assim desde a década de 1970. Como **resolver**?

- A **resposta dos pesquisadores** tem sido simples: use **isolamento *serializável***!

- **Isolamento Serializável** é considerado o nível de isolamento **mais forte**
  Garante que **transações executando em paralelo** tenham o mesmo resultado que *em série*
  O bd previne todas as **possíveis condições de corrida**

- Mas se é muito melhor do que níveis de isolamento fracos, então porque não é sempre usado? Implementações:

  1. Literalmente executar **transações em uma ordem serial**
  2. **Duas fases de bloqueio** - por muitas décadas a única opção viável
  3. Técnicas de controle de concorrência otimistas tais como isolamento ***snapshot* serializável**

Partitioning

Executar **todas as transações em série** faz o **controle de concorrência** muito **mais simples**, mas **limita** o **rendimento** da transação do bd com a **velocidade de um único núcleo** da CPU em uma única máquina

- Transações **somente leitura** podem executar em **outros lugares**, usando **isolamento *snapshot***

- Mas se as aplicações têm **alta taxa de transferência de escrita**, o processador de transações **single-threaded** pode se tornar um **gargalo sério**

Two-Phase Locking (2PL)

Para **escrever** (modificar ou excluir) um objeto, o **acesso exclusivo** é necessário:

- Se a **transação A leu** um objeto e transação **B quer escrever** nesse objeto, B **deve esperar até que A confirme** *(commit)* **ou aborte** antes de poder continuar

- Se a **transação A escreveu** em um objeto e transação **B quer ler** esse objeto, B **deve esperar até que A confirme** *(commit)* **ou aborte** antes de poder continuar

Two-Phase Locking (2PL)

A diferença fundamental entre o **travamento de duas fases** e o **isolamento snapshot** :

- No **2PL**, os **escritores bloqueiam outros escritores** e também bloqueiam **os leitores e vice-versa**

- Já no **isolamento *snapshot*** tem o mantra ***"leitores não bloqueiam escritores e escritores não bloqueiam os leitores"***

Implementação de bloqueio de duas fases 2PL

O 2PL é usado pelo nível de isolamento serializável no MySQL (InnoDB) e SQL Server

- Se **uma transação quer ler** um objeto:
  Deve adquirir o bloqueio no modo compartilhado
  Várias operações podem manter o bloqueio no **modo compartilhado simultaneamente**
  Se uma transação já tem um **bloqueio exclusivo**, essas operações devem esperar

- Se uma transação quer escrever em um objeto:
  Deve adquirir o bloqueio no modo bloqueio exclusivo
  Nenhuma outra transação pode ter o bloqueio ao mesmo tempo
  Se houver qualquer bloqueio existente no objeto, a transação deve esperar

Implementação de bloqueio de duas fases 2 PL

- Se uma transação **lê** em primeiro lugar e depois **escreve** em um objeto, pode **atualizar** o **bloqueio compartilhado** para um **bloqueio exclusivo**.
  A atualização funciona como a obtenção de um **bloqueio exclusivo** diretamente

- Depois que uma transação **adquiriu o bloqueio**, deve continuar a manter o bloqueio até o final da transação

- O **bloqueio de leitores e escritores** é implementado por um bloqueio em cada objeto no bd. Pode ser em *modo compartilhado ou modo exclusivo*

- O nome "**duas fases**" significa: a **primeira** fase (enquanto a transação está em execução) é quando os **bloqueios são adquiridos**, e a **segunda** fase (no final da transação) é quando todos os **bloqueios são liberados**

Desempenho de bloqueio de duas fases 2PL

Desvantagem de bloqueio de duas fases, o **desempenho**:

- Taxa de transferência
- Tempos de resposta de consultas

são **piores** que sobre i**solamento fraco**

Por design, se duas transações simultâneas tentarem fazer qualquer coisa que possa resultar em uma condição de corrida, **uma tem que esperar a outra** para se completar

Os bds relacionais tradicionais **não limitam a duração de uma transação**, pois são projetados para aplicações interativas que esperam por intervenção humana

Predicate locks

- Similar ao **bloqueio compartilhado /exclusivo**

- Porém ao **invés de pertencer a um** determinado objeto (por exemplo, uma linha em uma tabela), pertence a todos os objetos que correspondem a alguma condição de pesquisa

Predicate locks

Restringe o acesso da seguinte forma:

A quer **ler** objetos **combinando alguma condição**

Precisa

Adquirir um **predicate lock** de **modo compartilhado**

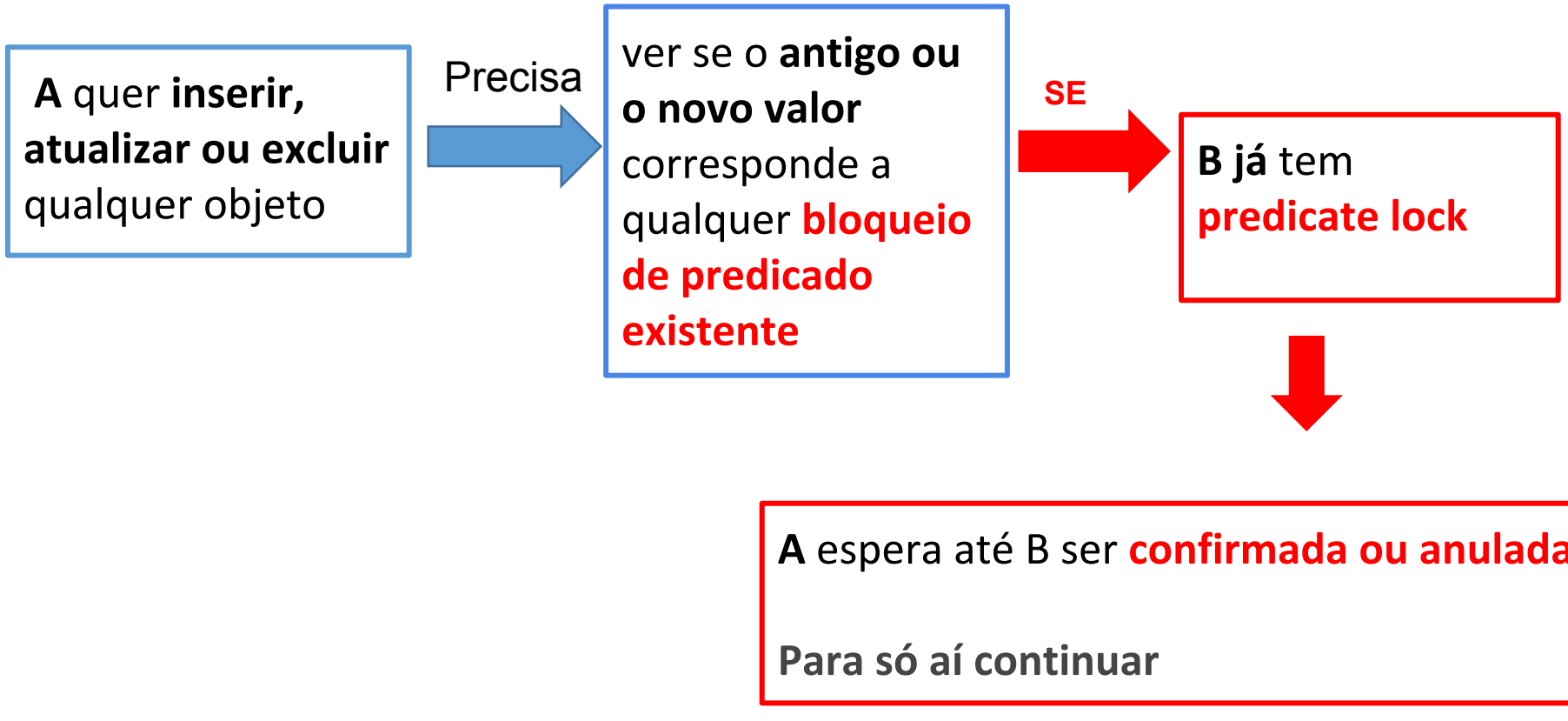
SE

B já tem **bloqueio de predicado exclusivo** em qualquer objeto combinando as condições

A espera até B liberar o **bloqueio**

Para só aí ser autorizada a fazer a sua consulta.

Predicate locks



Predicate locks

- Se aplica até aos objetos que ainda não existem no bd
  Mas que podem ser adicionados (**phantoms**)

- Se o **bloqueio de duas fases** inclui **P*redicate locks***:
  O bd **evita todas as formas** de **write skew** e outras condições de corrida
  Assim o seu **isolamento se torna serializável**

- Predicate locks **Não** executam bem:
  - se houver **muitos bloqueios** de transações ativas, verificar bloqueios correspondentes **torna-se demorado**

Index-range locks

- Por essa razão, muitos BDs com **2PL** implementam **index-range locking**
  - Que corresponde a uma aproximação simplificada de predicate locks
  - É **seguro simplificar** ao combiná-lo a um maior conjunto de objetos

- Uma **aproximação da condição de busca** é combinada em um dos índices

- Assim, **se outra transação** quer **inserir, atualizar ou excluir** uma reserva para **o mesmo objeto**,
  - Terá de atualizar a **mesma parte do índice**
  - Ao fazê-lo, **irá encontrar o bloqueio compartilhado**, e vai ser **obrigada a esperar** até que o bloqueio **seja liberado**

Isolamento Serializável Snapshot (SSI)

- **Isolamento serializável não** está em **desacordo** com **bom desempenho**

- O **SSI** é **usado** tanto em **bds de nó único** (o nível de isolamento Serializável em PostgreSQL desde versão 9.1) como em **bds distribuídos**

- **SSI** é muito recente em comparação com outros mecanismos de controle de concorrência, ainda está provando seu desempenho na prática, mas tem a **possibilidade** de ser **rápido o suficiente** para se **tornar o novo padrão no futuro**

Controle de concorrência pessimista versus otimista

Controle de concorrência pessimista versus otimista

- **Bloqueio de duas fases** é um mecanismo ***pessimista***:
  - baseia-se no princípio de que se **alguma coisa possa dar errado**, espera uma situação segura antes de fazer qualquer coisa
  - **Como exclusão mútua**, que é usada para proteger estruturas de dados na programação *multi-threaded*

- **Execução em série** é **pessimista ao extremo**:
  - Cada transação que tem um **bloqueio exclusivo** em todo o bd (ou uma partição) **durante a transação**
  - Compensa fazendo cada transação **muito rápida ao executar**, por isso só precisa segurar o “bloqueio” por um tempo curto

Controle de concorrência pessimista versus otimista

- **Isolamento serializável de *snapshot*** é uma técnica **otimista**:
  - **Em vez de bloquea**r se algo **puder dar errado**, as transações continuam assim mesmo, na **esperança de que tudo vai dar certo**
  - Quando uma transação quer confirmar *(commit)*, o bd verifica se **nada de ruim aconteceu** ou seja, se o **isolamento foi violado**
  - Se assim for, a **transação é cancelada** e tem de ser **repetida**
  - Apenas as operações que foram **executadas em série** estão **autorizadas a confirmar** *(commit)*

Decisões baseadas em uma premissa ultrapassada

Decisões baseadas em uma premissa ultrapassada

A Transação

1. **Efetua write skew** em isolamento de *snapshot*,
2. **Lê** alguns dados do bd,
3. **Analisa** o resultado da consulta, e
4. **Decide por alguma ação** (escrever no bd) com **base no resultado** que viu

Tudo bem até aí, não?

Decisões baseadas em uma premissa ultrapassada

A Transação

1. **Efetua write skew** em isolamento de *snapshot*,
2. **Lê** alguns dados do bd,
3. **Analisa** o resultado da consulta, e
4. **Decide por alguma ação** (escrever no bd) com **base no resultado** que viu

Tudo bem até aí, não?

- O resultado da consulta original **pode não ser mais o atualizado** quando a transação for **confirmada!**
  - Porque os dados podem ter sido **modificados** neste **meio tempo**

Decisões baseadas em uma premissa ultrapassada

A Transação

1. **Efetua write skew** em isolamento de *snapshot*,
2. **Lê** alguns dados do bd,
3. **Analisa** o resultado da consulta, e
4. **Decide por alguma ação** (escrever no bd) com **base no resultado** que viu

Dito de outra forma,

- A transação **está tomando uma ação** com **base em uma *premissa***
- Mais tarde, quando for confirmada,
  - Os dados originais podem **ter sido alterados**
  - **A premissa pode não ser verdade**

Decisões baseadas em uma premissa ultrapassada

É possível saber se um resultado foi alterado?

Decisões baseadas em uma premissa ultrapassada

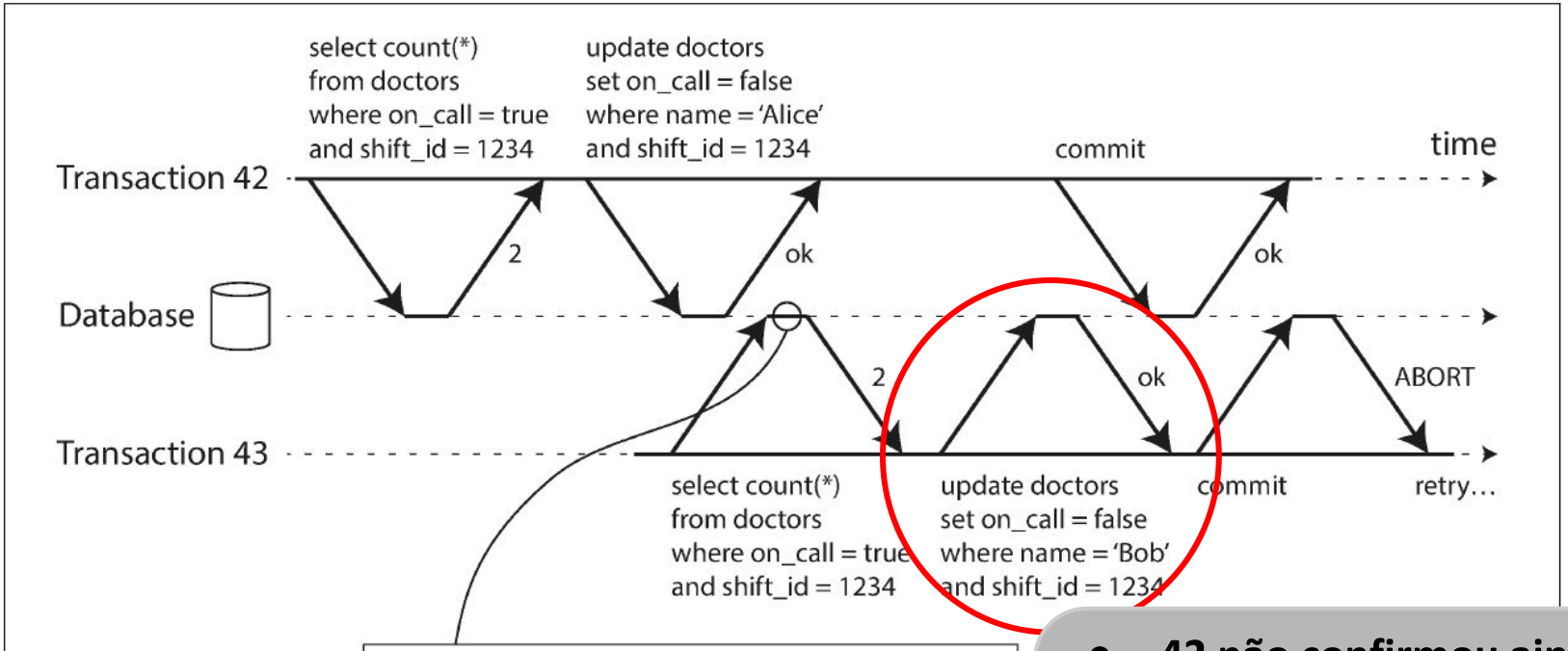
É possível saber se um resultado foi alterado?

O bd sabe se um resultado de consulta foi mudado considerando **dois casos**:

- Detectando **leituras de uma versão obsoleta** de objeto MVCC
  - **escritas não confirmadas ocorreram antes da leitura**

- Detectando **escritas que afetam leituras anteriores**
  - **a escrita ocorre após a leitura**

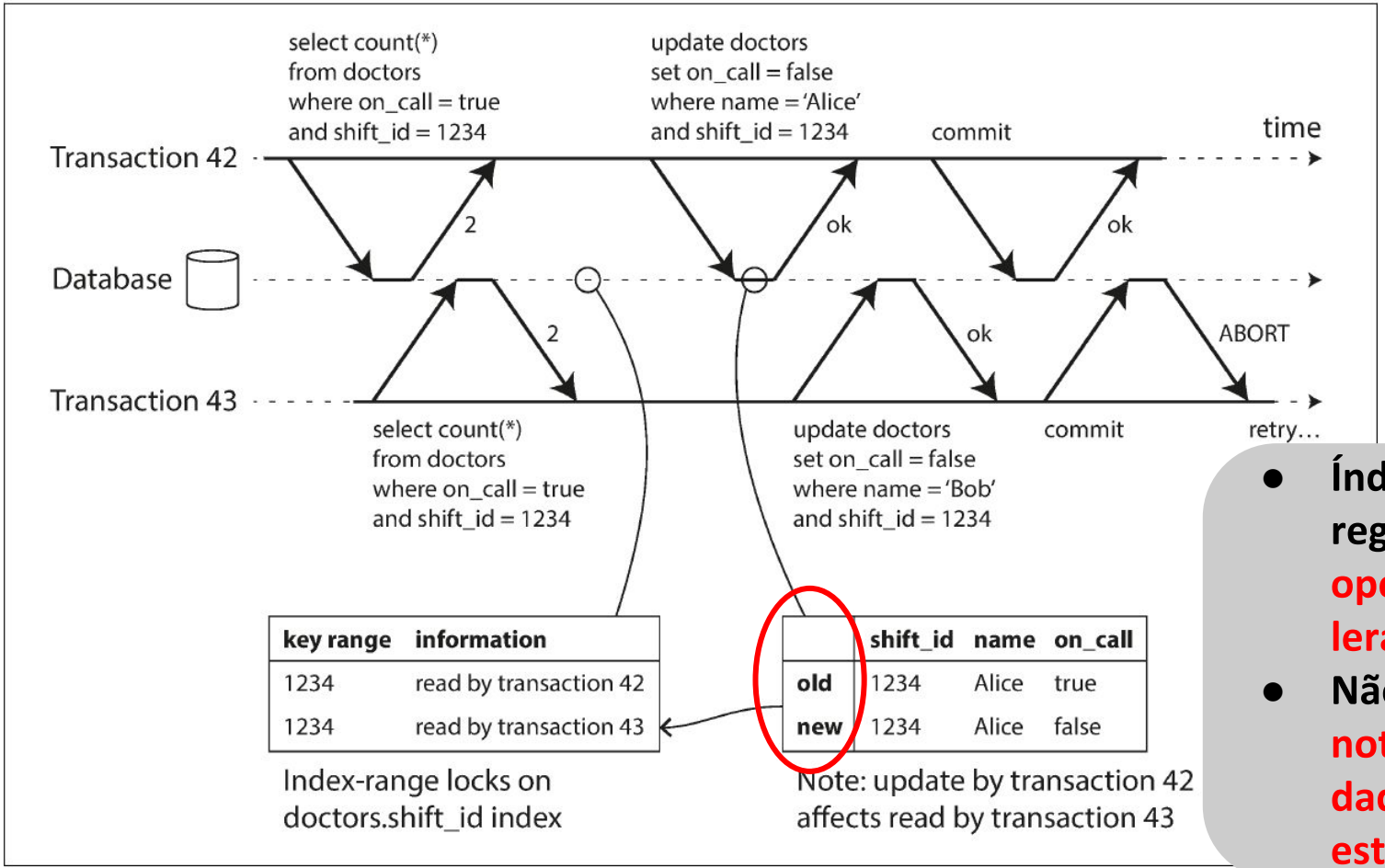
Detectando leituras obsoletas de um snapshot MVCC



shift_id	name	on_call	created_by	deleted_by
1234	Alice	true	1	42
1234	Alice	false	42	—
1234	Bob	true	1	—
1234	Carol	false	1	—

- 42 não confirmou ainda
- 43 vê um médico de plantão, então continua
- É percebido que este valor não está atualizado

Detectando *quando uma transação modifica a leitura de outra transação*



- **Índice em shift_id, registra operações que leram**
- **Não bloqueia, notifica que dados podem não estar atualizados**

Performance de isolamento snapshot serializável

- **Rastreamento** menos detalhado
  - **mais rápido** que o estritamente necessário
  - mas pode levar a **mais transações sendo abortadas**

- A **taxa de abortos** afeta significativamente o **desempenho geral do SSI**

- Pex: transação **lê e grava** dados por um **longo período de tempo**
  - Provável participar de **conflitos e abortar**
  - então o SSI requer que as operações sejam **bastante curtas**

- No entanto, SSI é provavelmente **menos sensível** a operações **demoradas** do que **bloqueio de duas fases** ou a **execução em série**

Resumo

Resumo deste Capítulo

TRANSAÇÕES são:

uma **camada de abstração**

que **permite** a um **aplicativo**

”fingir” que certos **problemas de concorrência** e certos tipos de **falhas** de hardware e software **não** existem

A grande **classe de erros** é reduzida até uma simples **transação abortar** pois a aplicação **só precisa tentar** novamente

Resumo deste Capítulo

Níveis de isolamento e anomalias evitadas

Nivel / Anomalia	Escrita Suja	Leitura Suja	Atualização Perdida	Read Skew	Write Skew	Phantom
READ UNCOMMITED / LEITURA NÃO CONFIRMADA	NÃO	SIM	SIM	SIM	SIM	SIM
READ COMMITED / LEITURA CONFIRMADA	NÃO	NÃO	SIM	SIM	SIM	SIM
SNAPSHOT / ISOLAMENTO INSTANTÂNEO	NÃO	NÃO	NÃO	NÃO	SIM	SIM
REPEATABLE READ / LEITURA REPETIDA	NÃO	NÃO	NÃO	NÃO	NÃO	SIM
SERIALIZABLE / SÉRIE	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO

Perguntas

1. **Explique o conceito de transação em um banco de dados e dê um exemplo onde se aplica o uso de transações.**

2. **Sobre Write Skew: Quais níveis de isolamento não permitem esta anomalia? Dê duas soluções, uma abordagem pessimista e outra otimista.**

Fim

Resumo deste Capítulo

Transações ajudam a prevenir muitos problemas, mas **nem todos os aplicativos** são suscetíveis a todos esses problemas

Padrões de acesso	**Uso de transações**
muito simples	ler e escrever apenas um único registro, provavelmente **pode administrar sem transações**
complexos	Transações **podem reduzir muito** o número de potenciais casos de erro que se precisa pensar (falhas de energia, disco cheio ..) **Sem transações**, torna-se difícil raciocinar sobre os efeitos que acessos complexos possam ter sobre o bd. Os dados **podem se tornar inconsistentes de várias maneiras** Pex: dados desnormalizados podem ficar fora de sincronia com os dados de origem

Resumo deste Capítulo

Para **controle de concorrência** em particular ***leitura confirmada, isolamento instantâneo*** e ***serializável***

Problema	*condições de corrida*	*níveis de isolamento que evitam*
Dirty reads	Um cliente **lê** as gravações de outro cliente **antes** que elas **sejam efetuadas**	Nível de **isolamento leitura confirmada** e níveis mais fortes evitam
Dirty writes	Um cliente **substitui** dados que **outro** cliente escreveu, mas **ainda não confirmados**	**Quase todas as implementações** de transação impedem escritas sujas
Read skew (nonrepeatable reads)	Um cliente **vê diferentes partes** do banco de dados em **diferentes pontos** no **tempo**	Evitado com **isolamento de snapshot**, que permite uma transação de **ler a partir de um snapshot consistente** em um ponto no **tempo**.

Resumo deste Capítulo

Para **controle de concorrência** existem vários níveis de isolamento, em particular ***leitura confirmada, isolamento instantâneo*** e ***serializável***.

Problema	*condições de corrida*	*níveis de isolamento que evitam*
Lost updates	*N*um **ciclo de leitura- modificação-gravação**. 1º cliente substitui a gravação do 2º **sem incorporar as mudanças que são perdidas**	**Algumas** transações de **isolamento snapshot** evitam automaticamente, enquanto outras exigem um **bloqueio manual**
Write skew	A **transação lê**, decide com base no valor que viu, e **escreve** a decisão no bd. Porém, **no momento** em que a **gravação** é feita, **a premissa da decisão já não é verdade**	Apenas **isolamento serializável** impede
Phantom reads	A transação **lê** objetos em uma pesquisa. **Outro cliente** faz uma **gravação que afeta** os resultados	**snapshot** impede leitura **phantom**. Phantoms no contexto de write skew necessitam de **tratamento especial**, tal como **index-range locks**

Resumo deste Capítulo

Níveis de isolamento fracos protegem contra **algumas** anomalias, mas é **melhor deixa**r o **desenvolvedor** lidar **manualmente** com as demais. **Apenas isolamento serializável** protege contra todas estas questões.

Abordagem	**implementação de transações serializáveis**
Literally executing transactions in a serial order	P/ transação **rápida** e taxa de transferência **baixa** o suficiente p/ processar em um **único núcleo da CPU**, esta é uma **opção simples e eficaz**
Two-phase locking	Tem sido a **forma padrão tradicional** de implementação de **serialização**, mas muitas aplicações evitam usá-la por causa do seu **desempenho**
Serializable snapshot isolation (SSI)	Usa **abordagem otimista**, permitindo que transações **prossigam sem bloqueio**