بسم الله الرحمن الرحیم

تکنولوژی کامپیوتر

جلسه‌ی شانزدهم
ترنزکشن

TRANSACTIONS

the harsh reality of data systems

- transactions have been the mechanism of choice for simplifying these issues.
- Purpose: simplify the programming model for applications accessing a database

Transaction

- A transaction is a series of operations grouped together as a unit, for which the system provides the certain set of guarantees mentioned above

- Although the concept of transactions applies to distributed databases, the examples in this chapter are all framed from a single-node perspective for the sake of introducing the topic with simpler examples

ACID Safety guarantees

- The safety guarantees provided by transactions are often characterized by the well-known acronym ACID, which stands for **Atomicity**, **Consistency**, **Isolation**, and **Durability**

ACID Safety guarantees

- Atomicity:
  - *Different from the atomic in concurrency*
  - *all writes will either succeed or all will fail. Upon error, any partial writes need to be undone.*

ACID Safety guarantees

- Consistency:
  - *in this context, used to mean DB will always be in a consistent state.*
  - Users probably interpret this as saying invariant properties will always remain true
  - But preserving invariants is really about how the application blocks operations in transactions
  - So consistency is really an application property, and shouldn't be part of this set of database guarantees

Consistency

- In Chapter 5 we discussed replica consistency and the issue of eventual consistency that arises in asynchronously replicated systems
- Consistent hashing is an approach to partitioning that some systems use for rebalancing
- In the CAP theorem (see Chapter 9), the word consistency is used to mean linearizability (see “Linearizability” on page 324).
- In the context of ACID, consistency refers to an application-specific notion of the database being in a “good state.

Consistency

- Atomicity, isolation, and durability are properties of the database, whereas consistency (in the ACID sense) is a property of the application.

ACID Safety guarantees

- Isolation:
  - *definition is that concurrent transactions won't interfere with each other at all, but that's rarely fully true*
  - Not interfering at all would be *serializability*, or acting as if the transactions had run one at a time (in some order), even though they may have actually had some or all parts running concurrently.
  - This is costly, so most systems don't provide full serializability
  - Most of the rest of this chapter is about isolation

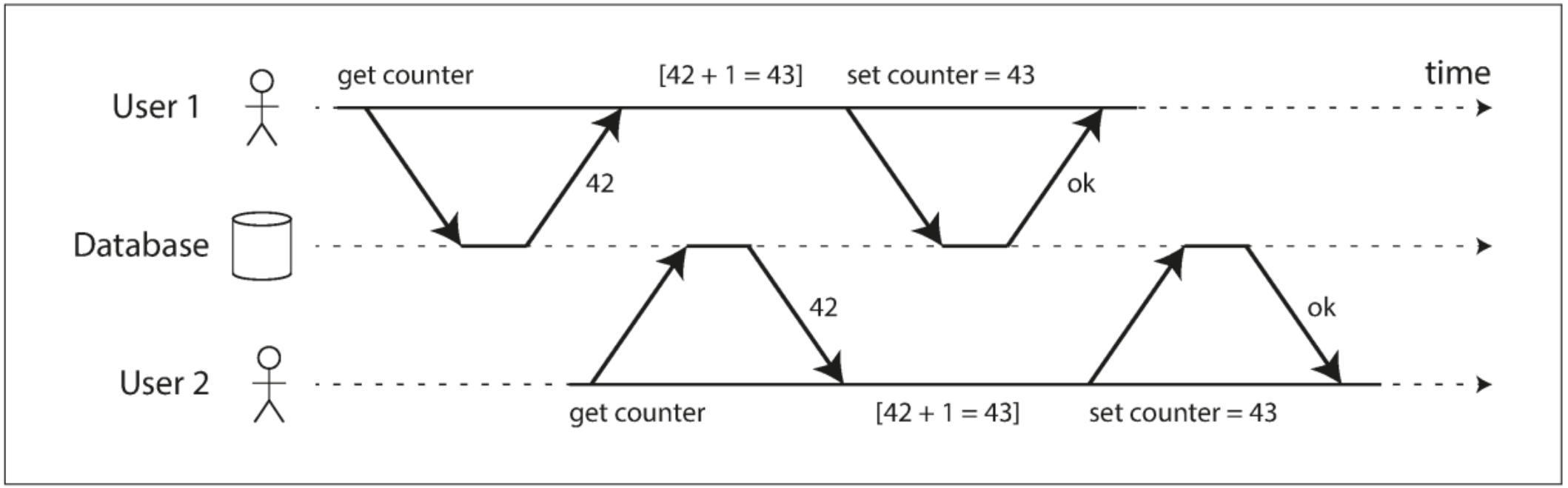

Figure 7-1. A race condition between two clients concurrently incrementing a counter.

ACID Safety guarantees

- Durability:
  - *once a transaction is reported success, any data written won't be lost*
  - In practice, this is pretty intuitive, e.g. the data has been written to disk
  - Technically, (multiple) disks can fail, and other storage anomalies mean that data written to disk isn't fully 100.0% safe

Weak Isolation Levels

- Isolation levels weaker than serializable have been implemented to provide better scalability
- Because most systems do not provide serializability
- Rather than blindly relying on tools, we need to develop a good understanding of the kinds of concurrency problems that exist, and how to prevent them.

Read Committed

- **dirty read:**
  - *Imagine a transaction has written some data to the database*
  - *but the transaction has not yet committed or aborted.*
  - *Can another transaction see that uncommitted data?*
  - *If yes, that is called a dirty read*

Read Committed

- **dirty read**

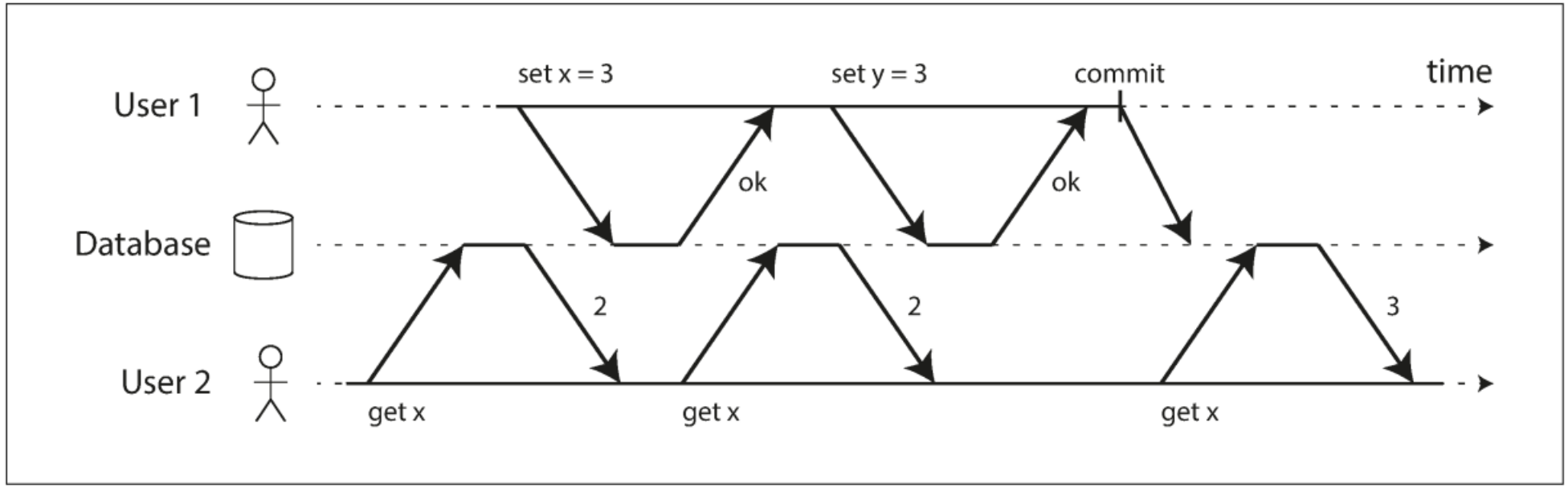

Figure 7-4. No dirty reads: user 2 sees the new value for x only after user 1's transaction has committed.

Read Committed

- **dirty write:**
  - *Imagine two transactions concurrently try to update the same object in a database.*
  - *assume that the later write overwrites the earlier write.*
  - *However, what happens if the earlier write is part of a transaction that has not yet committed so the later write overwrites an uncommitted value?*
  - *This is called a dirty write*

Read Committed

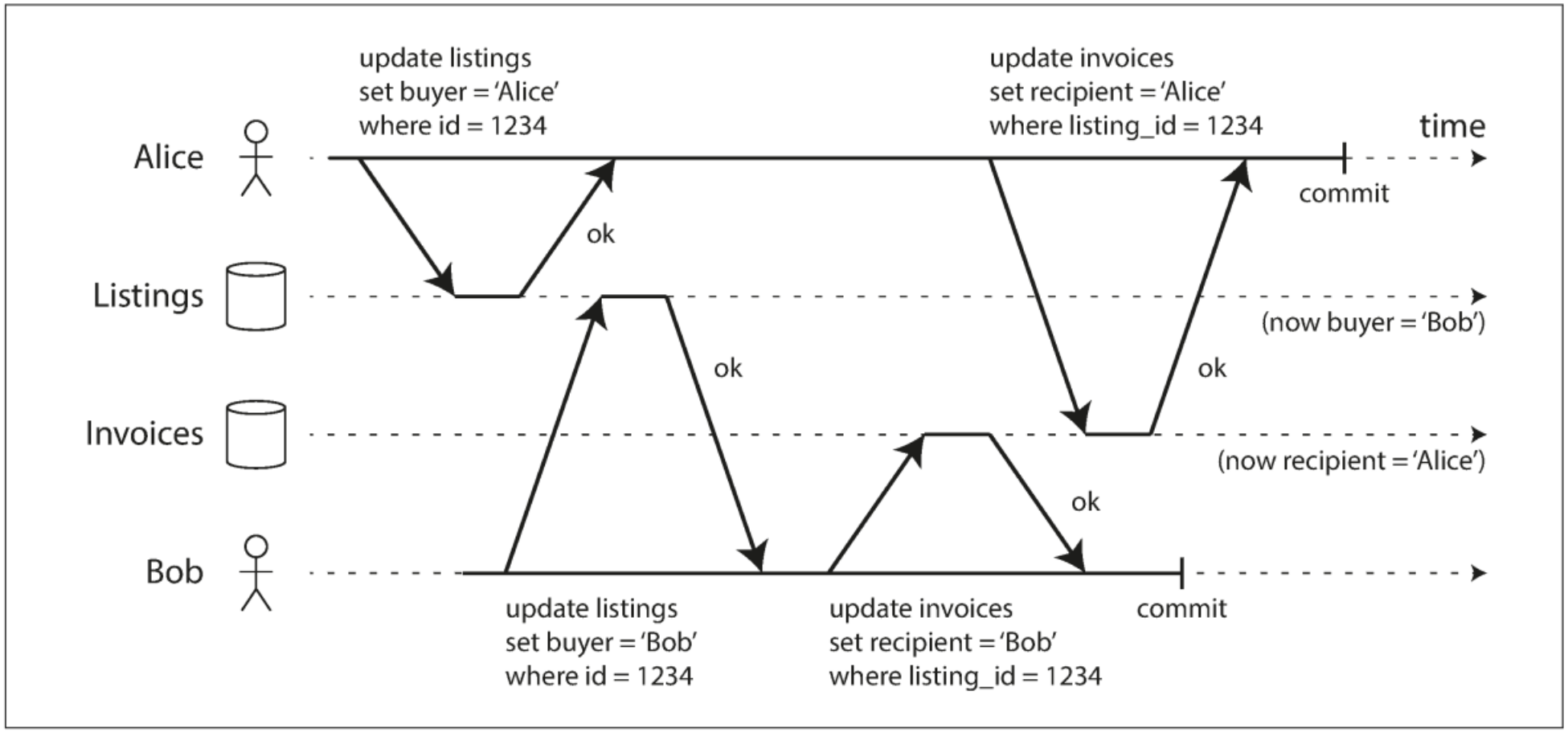

Figure 7-5. With dirty writes, conflicting writes from different transactions can be mixed up.

Read Committed

- When reading from the database, you will only see data that has been committed
  - *no dirty reads*
- When writing to the database, you will only overwrite data that has been committed
  - *no dirty writes*

Read Committed

- How to implement it

Snapshot Isolation / Repeatable Read

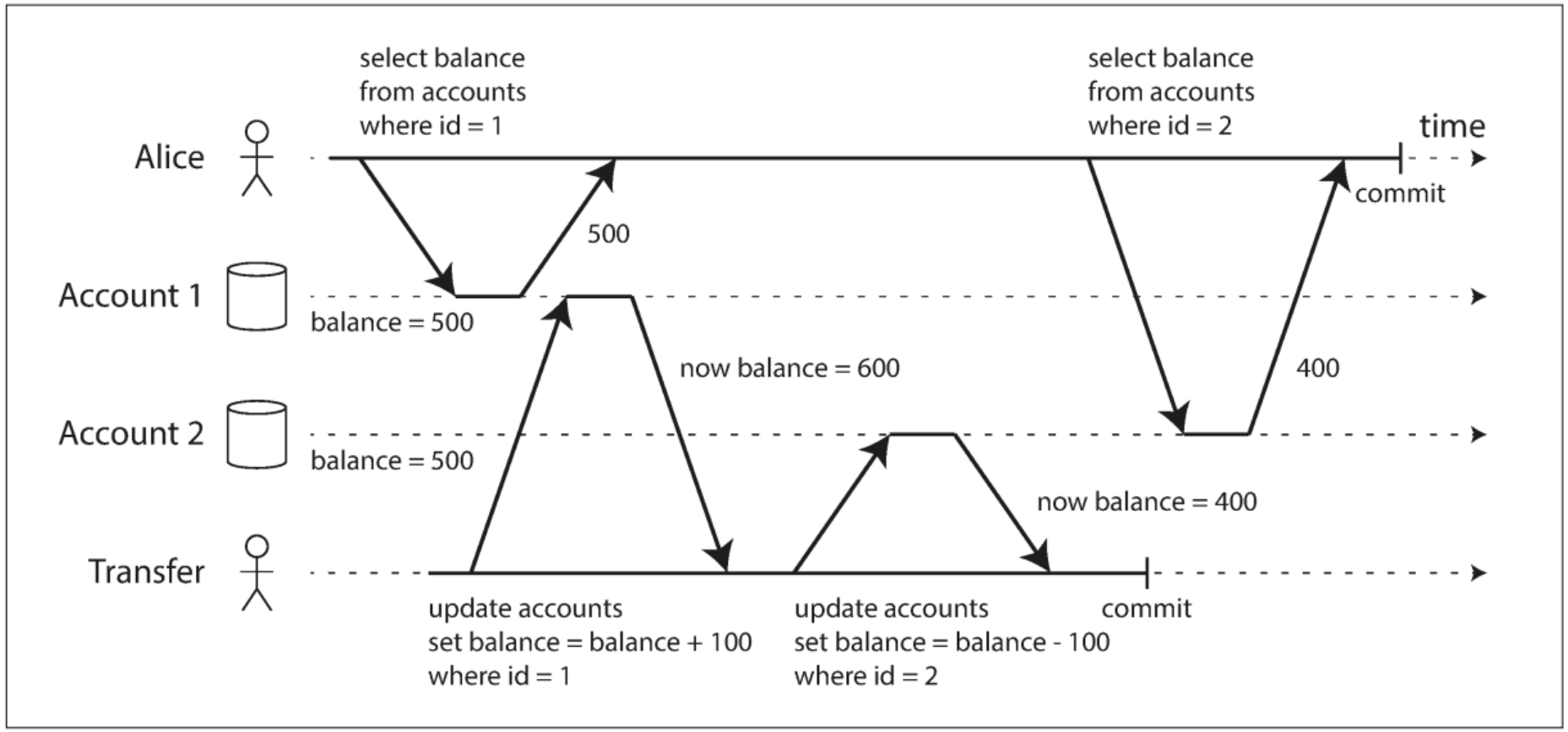

Figure 7-6. Read skew: Alice observes the database in an inconsistent state.

Snapshot Isolation / Repeatable Read

- Backups
- Analytic queries and integrity checks

Snapshot Isolation / Repeatable Read

- Solution:
  - *Multi version concurrency control (MVCC).*

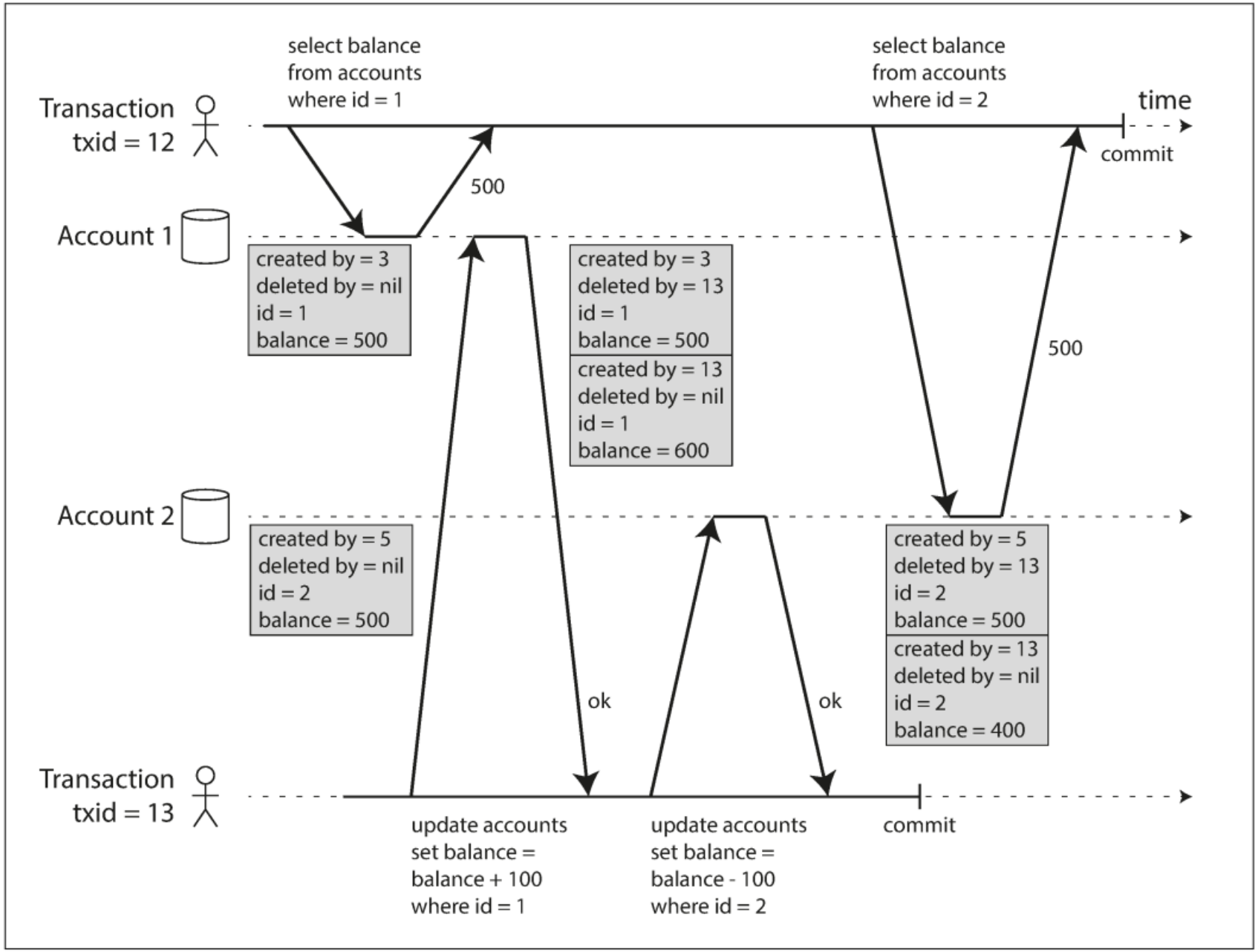

Figure 7-7. Implementing snapshot isolation using multi-version objects.

MVCC

- Visibility rules for observing a consistent snapshot
  - *At the start of each transaction, the database makes a list of all the other transactions that are in progress (not yet committed or aborted) at that time. Any writes that those transactions have made are ignored, even if the transactions subsequently commit.*
  - *Any writes made by aborted transactions are ignored.*
  - *Any writes made by transactions with a later transaction ID (i.e., which started after the current transaction started) are ignored, regardless of whether those transactions have committed.*
  - *All other writes are visible to the application's queries*

MVCC

- Indexes and snapshot isolation

- Snapshot isolation naming confusion
  - *In Oracle it is called serializable*
  - *in PostgreSQL and MySQL it is called repeatable read*

Lost Updates

- Race conditions that result a lost update:
  - *Incrementing a counter or updating an account balance (requires reading the current value, calculating the new value, and writing back the updated value)*
  - *Making a local change to a complex value, e.g., adding an element to a list within a JSON document (requires parsing the document, making the change, and writing back the modified document)*
  - *Two users editing a wiki page at the same time, where each user saves their changes by sending the entire page contents to the server, overwriting whatever is currently in the database*

Lost Updates

- Atomic write operations
  - *UPDATE counters SET value = value + 1 WHERE key = 'foo';*

Lost Updates

- Explicit locking
  - *BEGIN TRANSACTION;*
  - *SELECT * FROM figures WHERE name = 'robot' AND game_id = 222* ***FOR UPDATE;***
  - *UPDATE figures SET position = 'c4' WHERE id = 1234; COMMIT;*
- The FOR UPDATE clause indicates that the database should take a lock on all rows returned by this query.

Lost Updates

- Atomic and lock preventing lost updates by forcing the read-write-modify cycles to happen sequentially
- Automatically detecting lost updates
  - *Allow them execute in parallel*
  - *If lost update occur, then abort the transaction.*

Lost Updates

- Automatically detecting lost updates
  - *Using snapshot isolation with MVCC*

Lost Updates

- Compare and set
  - *UPDATE wiki_pages SET content = 'new content' WHERE id = 1234 AND content = 'old content';*
- Note: Check whether your database's compare-and-set operation is safe before relying on it

Lost Updates in replicated DBs

- Locks and compare and set do not apply on multi-leader or leader-less replications.
- Detecting concurrent writes which we learned before
- Using atomic operations and replicate them if operations are commutative
  - *Each replica can increment the value.*
  - *If one replica receive increment, then increment its value.*
- LWW is prone to lost updates.

Write Skew and Phantoms

- Example:
  - *Doctors*
  - *At-least one doctor must be on-call*
  - *One doctor can give up their shifts if there is another doctor oncall.*

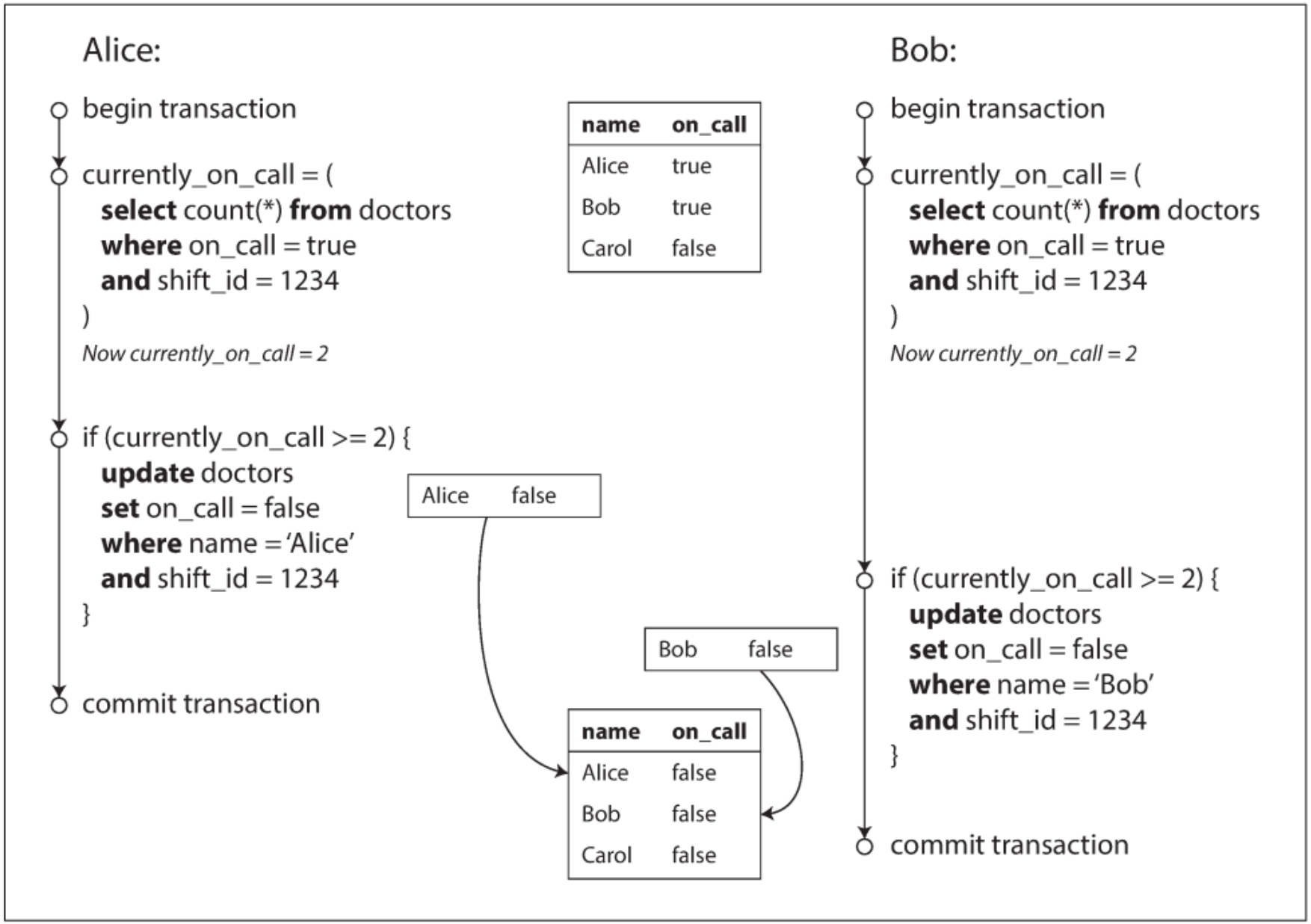

Figure 7-8. Example of write skew causing an application bug.

Write Skew and Phantoms

- Read more examples in Book (Page 249, 250)

Write Skew and Phantoms

- A SELECT query checks whether some requirement is satisfied by searching for rows that match some search condition.
- Depending on the result of select, application decides how to continue
- The application makes write (INSERT / DELETE / UPDATE)
- The precondition of step 2 changes.

Write Skew and Phantoms

- This effect, where a write in one transaction changes the result of a search query in another transaction, is called a phantom.

Write Skew and Phantoms

- A last resort solution If no alternative is possible:
  - *materializing conflicts*

Write Skew and Phantoms

- Predicate locks
  - *We will discuss later*

Serializability

- Serializable isolation is usually regarded as the strongest isolation level
- It guarantees that even though transactions may execute in parallel, the end result is the same as if they had executed one at a time, serially, without any concurrency

Actual Serial Execution

- Every transaction must be small and fast, because it takes only one slow transaction to stall all transaction processing.
- It is limited to use cases where the active dataset can fit in memory. Rarely accessed data could potentially be moved to disk, but if it needed to be accessed in a single-threaded transaction, the system would get very slow.
- Write throughput must be low enough to be handled on a single CPU core, or else transactions need to be partitioned without requiring cross-partition coordination.
- Cross-partition transactions are possible, but there is a hard limit to the extent to which they can be used.

Actual Serial Execution

- When writes can handled by single CPU
- Used by Redis
- Stored procedures…

Actual Serial Execution

- Stored procedures

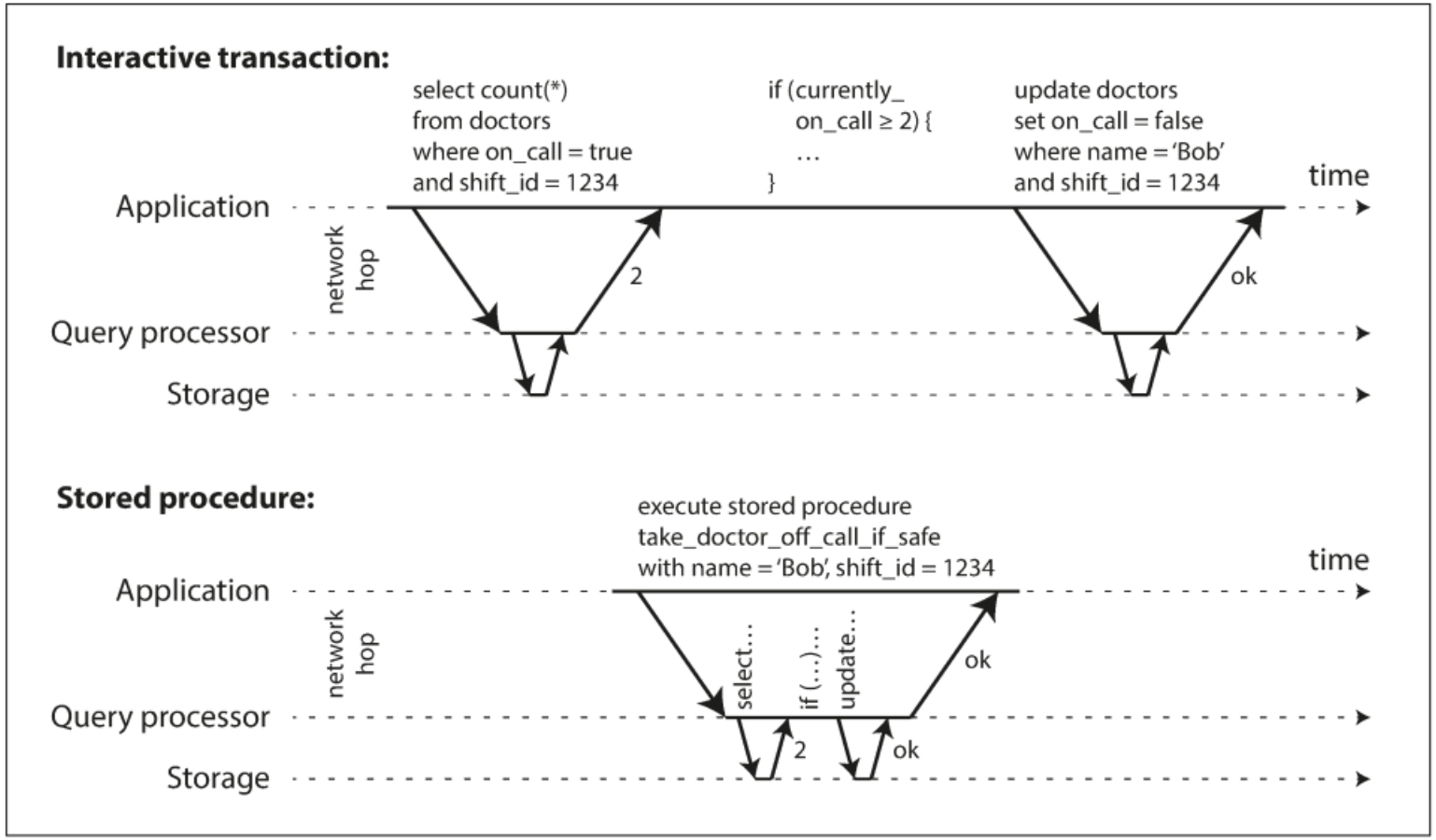

Figure 7-9. The difference between an interactive transaction and a stored procedure (using the example transaction of Figure 7-8).

Two Phase Locking

- Several transactions are allowed to concurrently read the same object as long as nobody is writing to it.
- But as soon as anyone wants to write (modify or delete) an object, exclusive access is required.
- If transaction A has read an object and transaction B wants to write to that object
  - *B must wait until A commits or aborts before it can continue.*
- If transaction A has written an object and transaction B wants to read that object
  - *B must wait until A commits or aborts before it can continue*

Two Phase Locking

- It is different from locking in snapshot isolation
  - *Snapshot isolation: a readers never block writers, and writers never block readers (*

Two Phase Locking

- How to implement?
  - *Lock for each object*
  - *Shared lock and exclusive lock*
  - *Release on transaction commit/abort*
- Deadlocks

Two Phase Locking

- Performance
  - *Overhead of acquiring and releasing locks*
  - *Reduce concurrency*

Two Phase Locking

- Predicate Locks for write skew
  - *Create lock not on particular object*
  - *But on all objects that match some search condition*
  - *SELECT * FROM bookings WHERE room_id = 123 AND end_time > '2018-01-01 12:00' AND start_time < '2018-01-01 13:00';*

Serializable Snapshot Isolation (SSI)

- It is Young: it was first described in 2008
- Used on Postgresql since version 9.1
- Optimistic concurrency:
- instead of blocking if something potentially dangerous happens, transactions continue anyway, in the hope that everything will turn out all right.
- When a transaction wants to commit, the database checks whether anything bad happened
- if so, the transaction is aborted and has to be retried

Serializable Snapshot Isolation (SSI)

- Only transactions that executed serializably are allowed to commit.
- Detecting reads of a stale MVCC object version (uncommitted write occurred before the read)
- Detecting writes that affect prior reads (the write occurs after the read)

Serializable Snapshot Isolation (SSI)

- Detecting reads of a stale MVCC object version (uncommitted write occurred before the read)

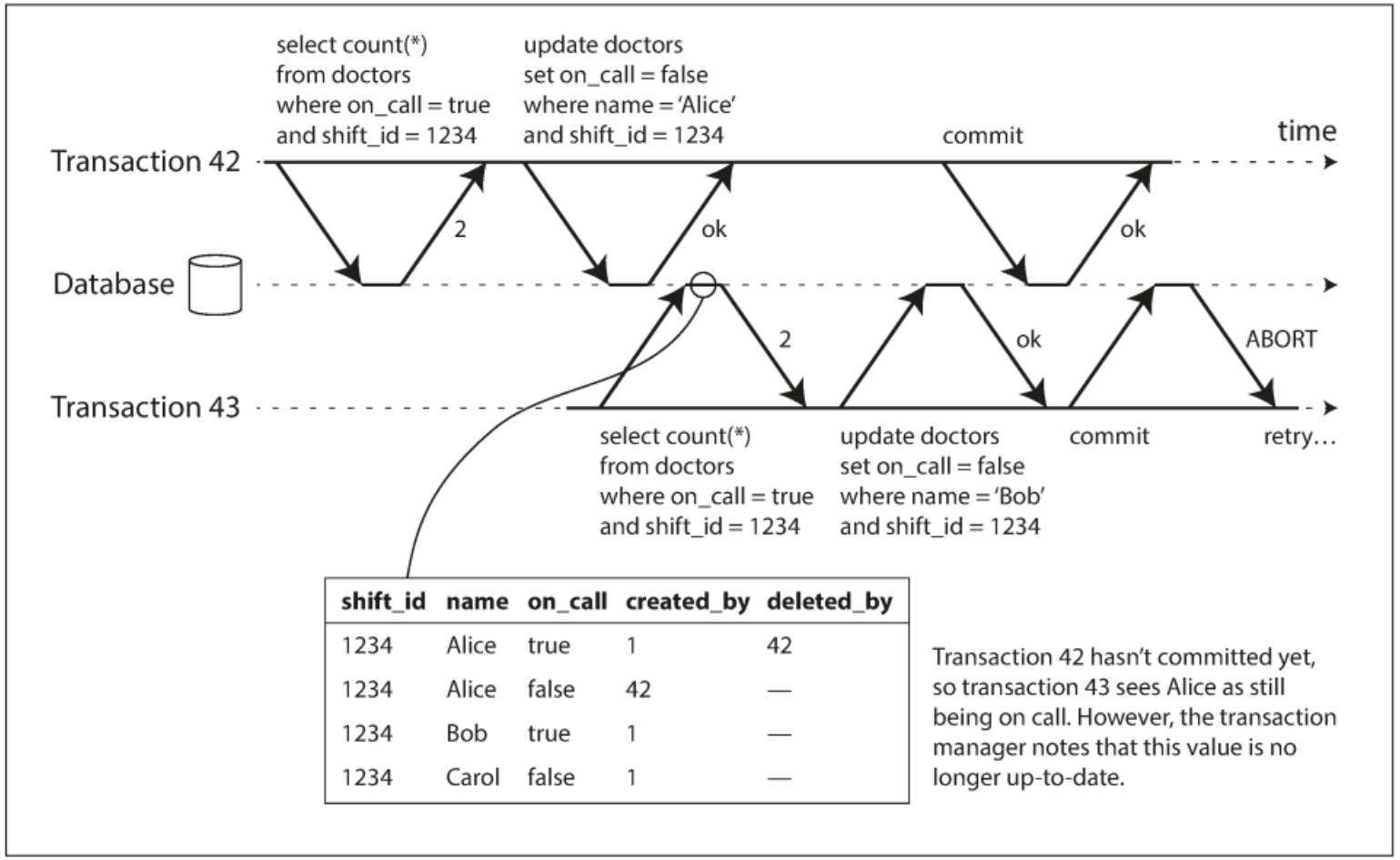

Figure 7-10. Detecting when a transaction reads outdated values from an MVCC snapshot.

Serializable Snapshot Isolation (SSI)

- Detecting writes that affect prior reads (the write occurs after the read)

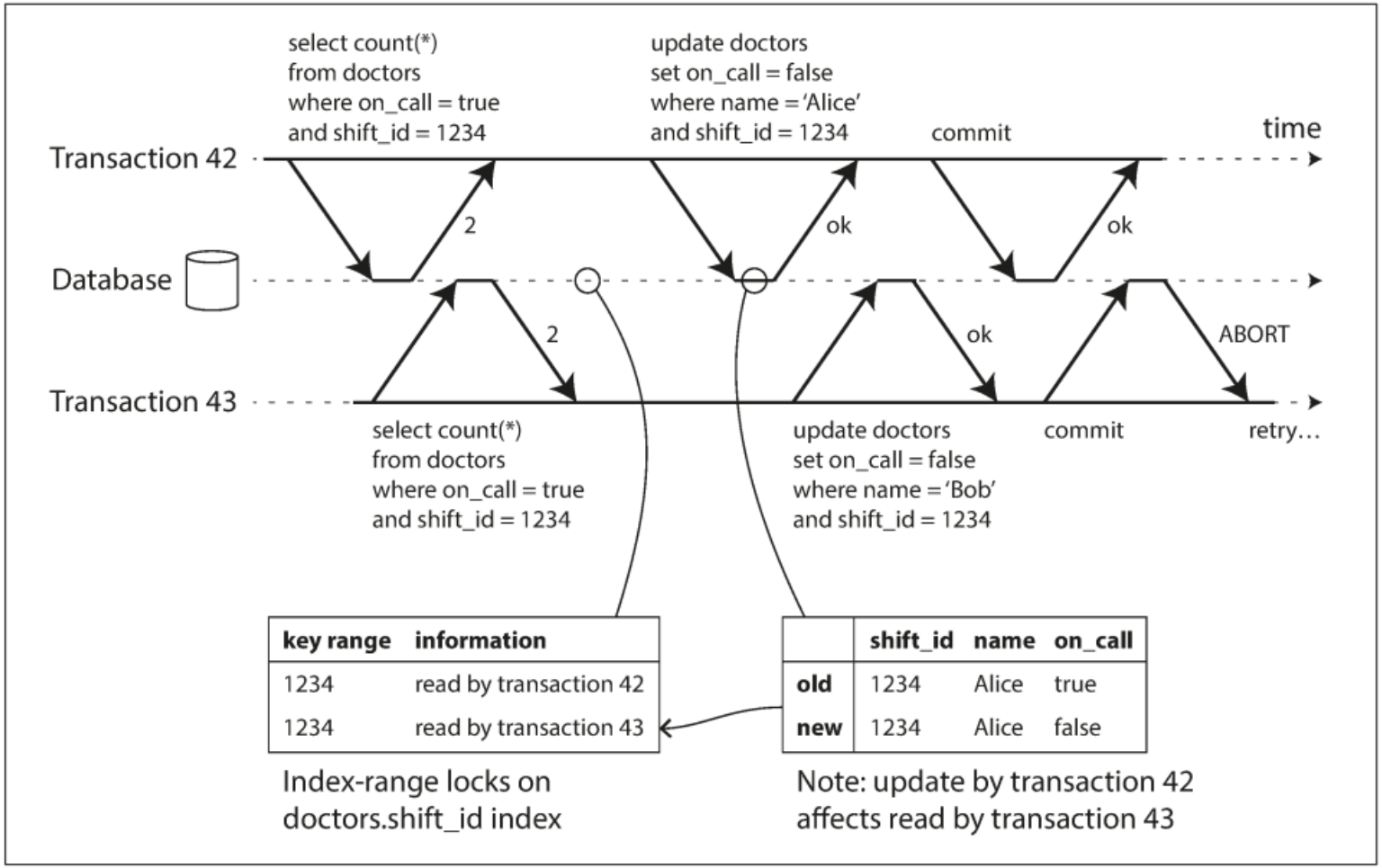

Figure 7-11. In serializable snapshot isolation, detecting when one transaction modifies another transaction's reads.

Serializable Snapshot Isolation (SSI)

- Performance?