Concurrency Control

Concurrent transactions

When two or more transactions appear to be running against a database at the same time.

However: The CPU can execute only one instruction at a time.

Transactions are **interleaved,** meaning that the operating system quickly switches CPU services among tasks so that some portion of each task is carried out in a given interval, but the transactions appear to be concurrent.

Concurrency Control

Concurrency Defined:

Multiple transactions can execute at the same time without compromising data integrity

Concurrency Control

Concurrency challenges

Lost Update: One user's changes overlay or interfere with another user's changes.

Inconsistent Reads: Two users can read the same thing at the same time and get different results.

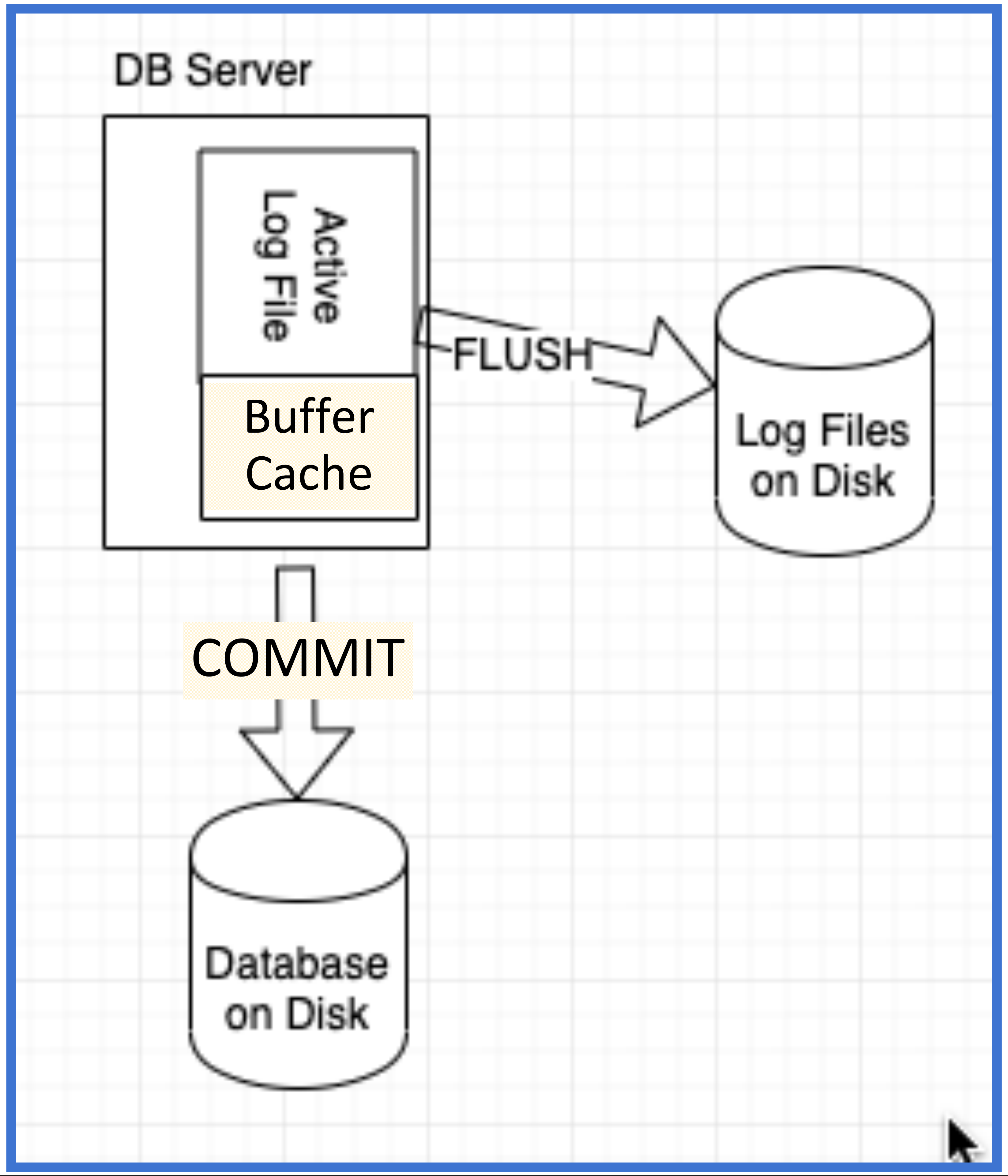
Deadlock: Occurs when two transactions are each waiting on a resource that the other transaction holds.

Concurrency Control

How can this happen?

All database updates happen within cache, and are only written out to the database on disk when a COMMIT is issued.

Transactions can access uncommitted data that resides in cache, if allowed.



Concurrency Control

Sequence of events without conflict

1. Read Item 500
2. Update Item 500
3. Write Item 500

User A

Processing order within the DB Server

1. Read Item 500 for A (into Cache)
2. Read Item 800 for B (into Cache)
3. Update Item 500 for A (in Cache)
4. Update Item 800 for B (in Cache)
5. Write Item 500 for A (commit)
6. Write Item 800 for B (commit)

1. Read Item 800
2. Update Item 800
3. Write Item 800

User B

Concurrency Control

Sequence of events with conflict – Lost Update

1. Read Item 500
   ("count" = 120)
2. Set "count" up by 12
3. Write Item 500

User A

Processing order within the DB Server

1. Read Item 500 for A (into Cache)
2. Read Item 500 for B (into Cache)
3. Update "count" to 132 for A
   (in Cache)
4. Write Item 500 for A (commit)
5. Update "count" to 110 for B
   (in Cache)
6. Write Item 500 for B (commit),
   overlayed "count" 132 with 110

1. Read Item 500
   ("count" = 120)
2. Set "count" down by 10
3. Write Item 500

User B

NOTE:
A's change in Step 3 is LOST

Concurrency Control

How does the DBMS Software handle concurrency problems?

Resource locking prevents multiple transactions from obtaining copies of the same record when the record is about to be changed.

Concurrency Control

Implicit locks are locks automatically placed by the DBMS

Explicit locks are issued by an application program issuing SQL "lock" commands (rare)

(Explicit locks issued by application code can increase the risk of deadlocks.)

Concurrency Control

Lock granularity refers to size of the locked resource:

- – Row (most granular)
- – Page ...
- – Table ...
- – Database (least granular)

Row level lock granularity minimizes the amount of data being locked, and so improves overall concurrency and processing throughput.

All locks require storage and processing time, so less granular locks can reduce locking overhead.

Small granularity

Concurrency Control

Types of locks:

– An **exclusive lock** prohibits other users from reading or updating the locked resource

– A **shared lock** allows other users to read the locked resource, but they cannot update it.

Concurrency Control

Serializable Transactions

Refers to two transactions that execute in parallel, but the result is the same as if the transactions had run separately, leaving the database in a consistent state.

The transactions are **fully isolated** from each other.

Concurrency Control

Serialized Transactions

1. Lock Item 500
2. Read Item 500
   ("count" = 120)
3. Set "count" up by 12
4. Write Item 500
5. Release lock on Item 500

User A

1. Lock Item 500
2. Read Item 500
3. Set "count" down by 10
4. Write Item 500
5. Release lock on Item 500

User B

Concurrency Control

Serialized Transactions

Processing order within the DB Server

1. Lock Item 500 for A
2. Read Item 500 for A
3. Lock Item 500 for B – CANNOT grab lock, Trans B goes into WAIT STATE until the lock on Item 500 is released
4. Update "count" to 132 (for A)
5. Write Item 500 for A (commit)
6. Release lock on Item 500 from A
7. Lock Item 500 for B
8. Read Item 500 for B
9. Update "count" to 122 (for B)
10. Write Item 500 for B (commit)
11. Release lock on Item 500 from B

Concurrency Control

Deadlock, or the "deadly embrace", occurs when two transactions are each waiting on a resource that the other transaction holds.

Breaking deadlock:

- Almost every DBMS has algorithms for detecting deadlock
- When a deadlock occurs, the DBMS aborts one of the transactions (selected randomly), allowing one transaction to complete, while the aborted transaction rolls back any partially completed work.

The aborted transaction must be resubmitted by the application

Concurrency Control

Deadlock

1. Lock Invoice 2022-1
2. Read Invoice 2022-1
3. Lock Item 500
4. Read Item 500
5. Update Invoice 2022-1 (commit)
6. Update Item 500 (commit)
7. Release lock on Invoice 2022-1
8. Release lock on Item 500

User A

1. Lock Item 500
2. Read Item 500
3. Lock Invoice 2022-1
4. Read Invoice 2022-1
5. Update Item 500 (commit)
6. Update Invoice 2022-1 (commit)
7. Release lock on Item 500
8. Release lock on Invoice 2022-1

User B

Concurrency Control

Deadlock

Processing order within the DB Server

1. Lock Invoice 2022-1 for A
2. Read Invoice 2022-1 for A
3. Lock Item 500 for B
4. Read Item 500 for B
4. Lock Item 500 for A – Cannot grab lock, Trans A goes into WAIT state
5. Lock Invoice 2022-1 for B - Cannot grab lock, Trans B goes into WAIT state
6. Deadlock is detected
7. DBMS selects ONE transaction (Trans B arbitrarily) to abort
8. Release lock on Item 500 from B due to abort
9. Lock Item 500 for A
10. Update Invoice 2022-1, Item 500 for A (commit)
11. Release locks on Invoice 2022-1 and Item 500 from Trans A

Concurrency Control

The acronym ACID

DBMS Transaction management must be Atomic, Consistent, Isolated, and Durable.

Atomic means either *ALL* database updates within a transaction are committed or *ALL* updates are rolled back.

Consistent means the DBMS software will not allow any updates to violate any database constraints. Once committed, the transaction leaves the data in a consistent state.

Concurrency Control

Isolated means the DBMS software was designed and written in such a way that database administrators can declare the desired isolation level to control concurrency.

Durable means that once a transaction is commited to disk, it remains committed, even in the case of a system failure.

Concurrency Control

The ANSI SQL-92 standard defines four levels of **transaction isolation**:

- Read uncommitted – allows reads of uncommitted changes (i.e. “dirty” reads, unrepeatable reads and phantom reads)
- Read committed – allows unrepeatable reads and phantom reads, but no dirty reads
- Repeatable read – allows phantom reads, but no unrepeatable reads
- Serializable – guarantees no read anomalies

Concurrency Control

Anomalies:

“Dirty Read” – a READ request is allowed to read from cache the updated but uncommitted copy of the data

“Unrepeatable Read” – One READ request following another READ request gets two different results.

Concurrency Control

Anomalies:

“Phantom Read” – a READ request returns a different set of rows at different times. For example the first SELECT returns 10 rows; the next SELECT returns 11 rows (including a “phantom” row) that was inserted after the first read.

Concurrency Control

Why should we care about Isolation levels?

Isolation levels impact

- The speed at which processing get done
- The frequency and duration of various locks

Trade-offs

- Faster throughput versus data consistency and reliability

Concurrency Control

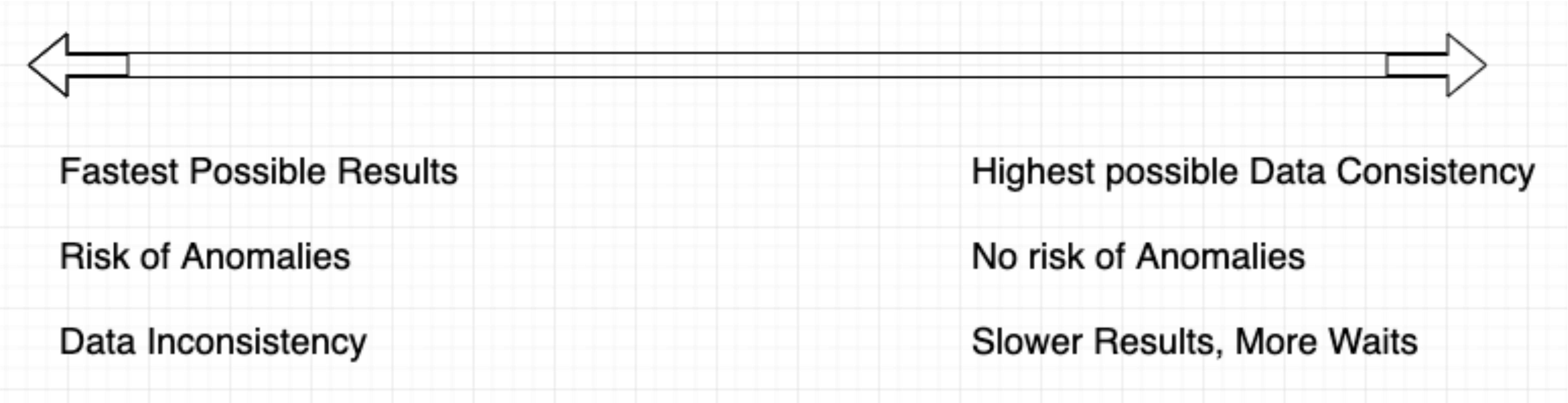
Why do we care about **ACID?**

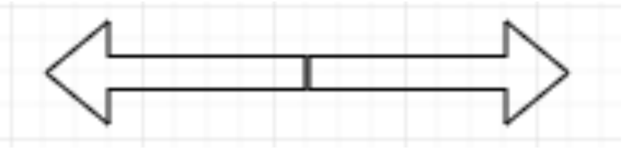
- Relational DBMS software is written with extremely complex algorithms to ensure ACID compliance

- Ensuring ACID compliance has **trade-offs**
  - Do I want the fastest possible processing? That is, applications/users never have to wait for commits, locks, etc.
  - Do I want to be sure that my application/users maintains full data integrity – that is, no risk of inconsistent data?

Concurrency Control

The DBMS software itself determines where you land on this scale.



NoSQL databases  Relational Databases

Concurrency Control

Summary

- As multiple transactions run against the database concurrently, anomalies can arise
  - Lost Updates, Inconsistent Reads, Deadlocks
- The DBMS software uses locks to minimize anomalies arising from concurrent updates
- Relational DBMS software was written to adhere to ACID transaction compliance