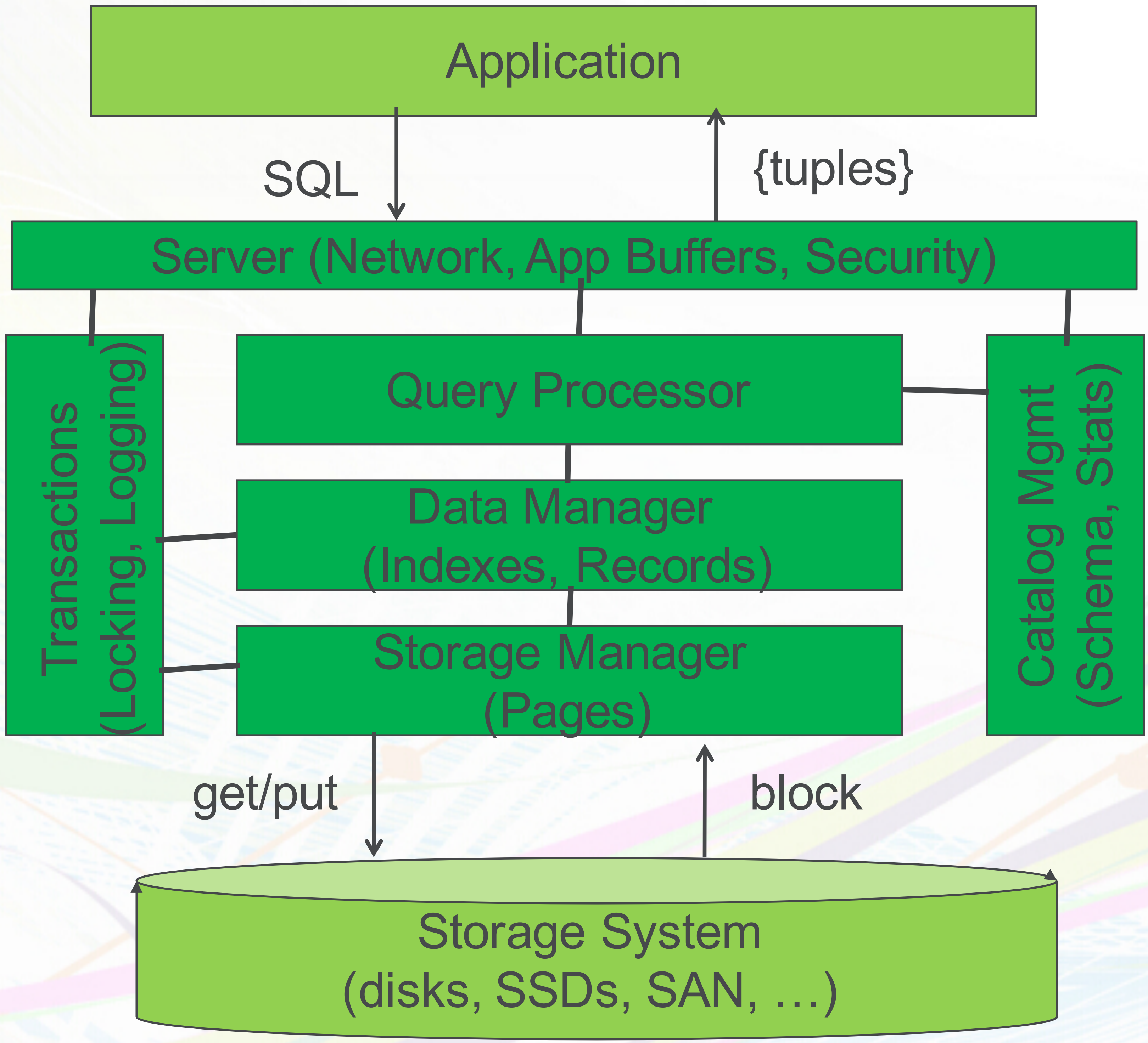
Well-prepared for exams

- Exam 1: 9:00-10:30am on Nov 16, 2015 (counted as midterm results) –BS1-42; BS3/MS: 51
- Exam 2: 9:00-10:30am on Dec 7, 2015

- Always read materials and do exercises in advance.
- Always have meals but not too much before.
- Always be on time.
- Always prepare the **right** tools required by exams, **papers, pens, computers (online exams)**, etc.
- Calm down, and do not stick on several questions, take care of your total exam time.
- Only help yourself, not others…
- You can only submit your exam results once
- google login

What we have studied...

and what about the last lecture


Application
SQL
{tuples}
Server (Network, App Buffers, Security)
Transactions
(Locking, Logging)
Query Processor
Data Manager
(Indexes, Records)
Storage Manager
(Pages)
Catalog Mgmt
(Schema, Stats)
get/put
block
Storage System
(disks, SSDs, SAN, …)

Transaction

Today's Class

- Transactions Overview
- Concurrency Control
- Recovery

Why transaction? Motivation

Lost Updates
Concurrency Control

- We both change the same record ("Smith"); how to avoid race condition?

Durability
Recovery

- You transfer 10,000 rubles from savings→checking; power failure – what happens?

Lost Updates
Concurrency Control

change the s
record (“Smit
avoi

Durabilit
Re

g, power
hat happens?

DBMSs automatically handle both issues: ‘transactions’

Transactions

- A ***transaction*** is the execution of a sequence of one or more operations (e.g., SQL queries) on **a shared database** to perform some higher-level function.

- It is the basic unit of change in a DBMS:
  - Partial transactions are not allowed!

Transaction Example

- *Move 10,000 rubles from Qiang' bank account to Sadegh's.*
- The txn is as a sequence of operations:
  1. Read Qiang's balance and check whether Qiang has 10,000 rubles
  2. Deduct 10,000 rubles from his account
  3. Write the new balance into DB
  4. Read Sadegh's balance
  5. Add 10,000 rubles to Sadegh's account
  6. Write the new balance into DB

Strawman System, 1982

- Execute each txn one-by-one (i.e., ***serial order***) as they arrive at the DBMS.
- One and only one txn can be running at the same time in the DBMS.

Issues ???

What we want

•Concurrent execution of independent transactions.

•**Q:** Why do we want that?
  –Utilization/throughput ("hide" waiting for I/Os)
  –Increased response times to users.

•But we also would like:
  –Correctness
  –Fairness

Challenges…

- Hard to ensure correctness…
  - *What happens if Qiang only has 10,000 rubles and tries to transfer twice at the same time?*
- Hard to execute quickly…
  - *What happens if Qiang needs to pay off his debts very quickly all at once?*

N.B. …What DB is dealing with

- A program may carry out many operations on the data retrieved from the database
- However, the **DBMS is only concerned about what data is read/written from/to the database**.
  - Changes to the “outside world” are beyond the scope of the DBMS.

Transactions in SQL

- A new txn starts with the **begin** command.
- The txn stops with either **commit** or **abort**:
  - If **commit**, all changes are saved.
  - If **abort**, all changes are undone so that it's like as if the txn never executed at all.

A txn can abort itself or the DBMS can abort it.

Correctness Criteria: **ACID**

- **Atomicity:** All or nothing. Undo changes if there is a problem
- **Consistency:** If DB is consistent before a txn, it ends up consistent. Check integrity constraints at the end of a txn
- **Isolation:** Execution of one txn is isolated from that of other txns.
- **Durability:** Updates of a completed txn must never be lost. Redo changes if there is a problem

Correctness Criteria: **ACID**

- **Atomicity:** *"all or nothing"*
- **Consistency:** *"it looks correct to me"*
- **Isolation:** *"as if alone"*
- **Durability:** *"survive failures"*

Overview

- Problem & 'ACID'
- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

A

Atomicity

- Two possible outcomes of executing a txn:
  - –Txn might *commit* after completing all its actions.
  - –or it could *abort* (or be aborted by the DBMS) after executing some actions.
- DBMS guarantees that txns are **atomic**.
  - –From user's point of view: **txn always either executes all its actions, or executes no actions at all**.

Mechanisms for Ensuring Atomicity

- We take 10,000 rubles out of Qiang' account **but then** there is a power failure before we transfer it to Sadegh's.

Q: When the database comes back on-line, what should be the correct state of Qiang' account and Sadegh's?

Mechanisms for Ensuring Atomicity

- One approach: **LOGGING**
  - DBMS *logs* all actions so that it can *undo* the actions of aborted transactions.

Mechanisms for Ensuring Atomicity

- Logging used by all modern systems.
- **Q:** Why?

Mechanisms for Ensuring Atomicity

- Logging used by all modern systems.
- **Q:** Why?
- **A:** Audit Trail & Efficiency Reasons

- *What other mechanism can you think of?*

Mechanisms for Ensuring Atomicity

- Another approach: **SHADOW PAGING**
  - –DBMS makes copies of pages and txns make changes to those copies. Only when the txn commits is the page made visible to others.
  - –Originally from System R.
- Nobody actually does this…

Overview

- Problem & '**ACID**'
- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Database Consistency

- **Database Consistency:** Data in the DBMS is accurate in modeling the real world and follows integrity constraints

Transaction Consistency

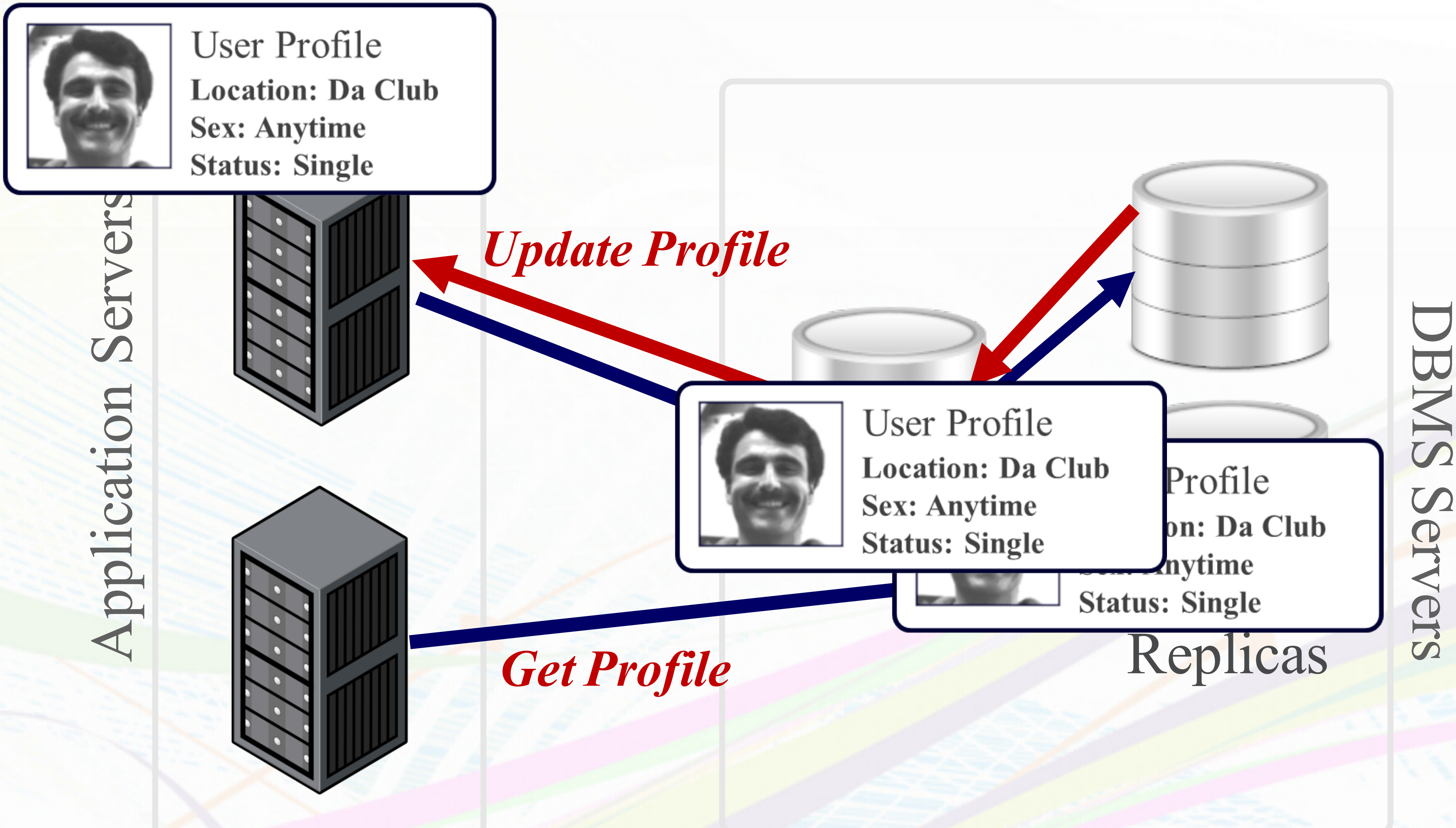
- **Transaction Consistency:** if the database is consistent before the txn starts (running alone), it will be after also.

- Transaction consistency is the application's responsibility.
  - *We won't discuss this further…*

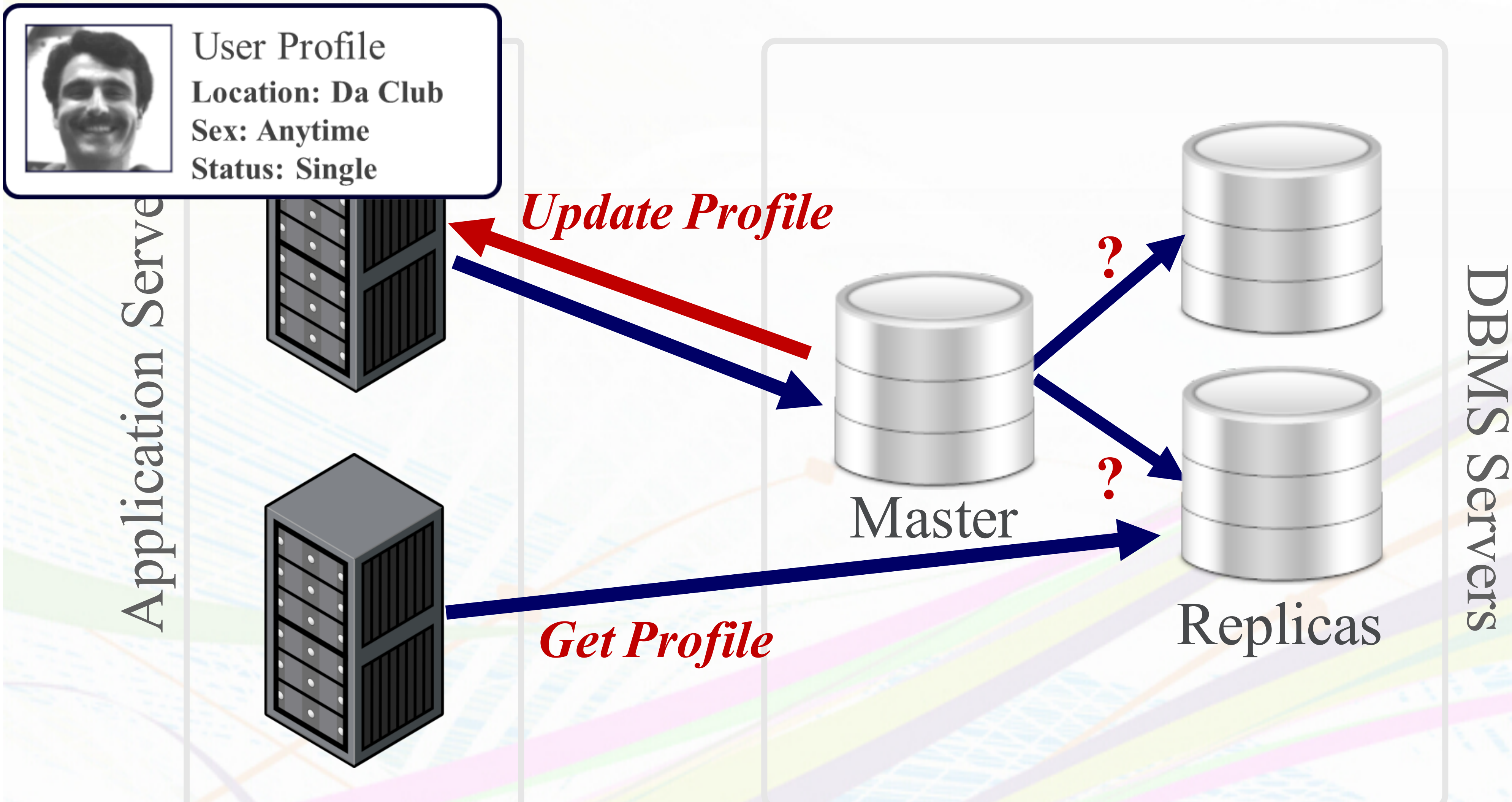
Strong vs. Weak Consistency

- In a distributed DBMS, the consistency level determines when other nodes see new data in the database:
  - **Strong:** Guaranteed to see all writes immediately, but txns are slower.
  - **Weak/Eventual:** Will see writes at some later point in time, but txns are faster.

Strong Consistency



Eventual Consistency



Overview

- Problem definition & ‘ACID’
- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Isolation of Transactions

- Users submit txns, and each txn executes *as if it* was running *by itself*.
- Concurrency is achieved by DBMS, which interleaves actions (reads/writes of DB objects) of various transactions.
- **Q:** How do we achieve this?

Isolation of Transactions

- **A:** Many methods - two main categories:
  - **Pessimistic** – Don't let problems arise in the first place.
  - **Optimistic** – Assume conflicts are rare, deal with them after they happen.

Example

T1	T2
BEGIN A=A+10,000 B=B–10,000 **COMMIT**	**BEGIN** A=A*1.06 B=B*1.06 **COMMIT**

- Consider two txns:
  - T1 transfers 10,000 rubles from Qiang's account (B) to Sadegh's (A)
  - T2 credits both accounts with 6% interest.

Example

T1	T2
BEGIN A=A+10,000 B=B–10,000 **COMMIT**	**BEGIN** A=A*1.06 B=B*1.06 **COMMIT**

- Assume at first Qiang and Sadegh each have 10,000 rubles.
- **Q:** What are the *legal outcomes* of running T1 and T2?

Example

Q: What are the *legal outcomes* of running T1 and T2?

A: Many! But Qiang+Sadegh should be:
20,000*1.06=21200 rubles

- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. But, the net effect must be equivalent to these two transactions running **serially** in some order.

Example

- The outcome depends on whether T1 executes before T2 or vice versa.

Serial Execution Example

TIME

Schedule

T1	T2
BEGIN	
A=A+10000	
B=B–10000	
COMMIT	
	BEGIN
	A=A*1.06
	B=B*1.06
	COMMIT

A=??, B=??

≡

Schedule

T1	T2
	BEGIN
	A=A*1.06
	B=B*1.06
	COMMIT
BEGIN	
A=A+10000	
B=B–10000	
COMMIT	

A=??, B=??

Interleaving Transactions

- We can also interleave the txns in order to maximize concurrency.
  - Slow disk/network I/O.
  - Multi-core CPUs.

Interleaving Example

Schedule

TIME

T1	T2
BEGIN	
A=A+10000	
	BEGIN
	A=A*1.06
B=B–10000	
COMMIT	
	B=B*1.06
	COMMIT

What are the results?

Correctness

- **Q:** How do we judge that a schedule is correct?
- **A:** If it is ***equivalent*** to some ***serial*** execution

Formal Properties of Schedules

- **Serial Schedule:** A schedule that does not interleave the actions of different transactions.
- **Equivalent Schedules**: For any database state, the effect of executing the first schedule is identical to the effect of executing the second schedule.*

(*) no matter what the arithmetic operations are!

Formal Properties of Schedules

- **Serializable Schedule:** A schedule that is *equivalent* to some serial execution of the transactions.
- Note: If each transaction preserves consistency, every serializable schedule preserves consistency.

Formal Properties of Schedules

- **Serializability** is a less intuitive notion of correctness compared to txn initiation time or commit order, but it provides the DBMS with significant additional flexibility in scheduling operations.

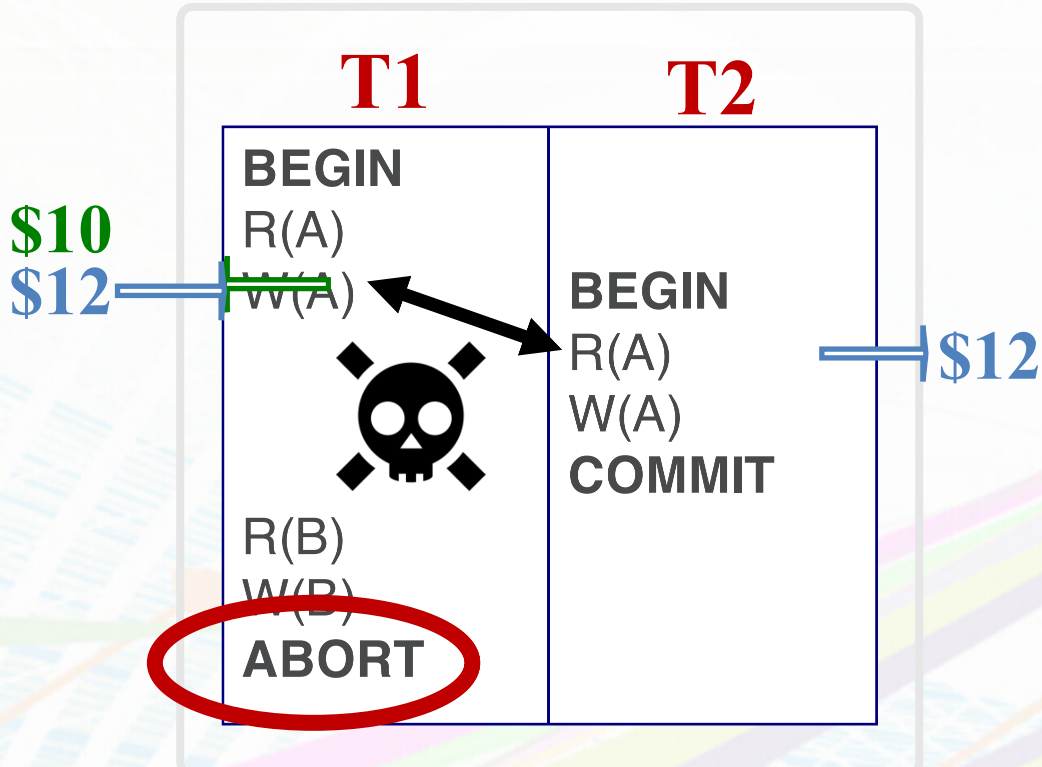
Interleaved Execution Anomalies

- **Read-Write (RW)** conflicts – Unrepeatable read (re-read committed data)
- **Write-Read (WR)** conflicts -Dirty read (read uncommitted data)
- **Write-Write (WW)** conflicts – Blind write (overwrite uncommitted data)

Q: Why not RR conflicts?

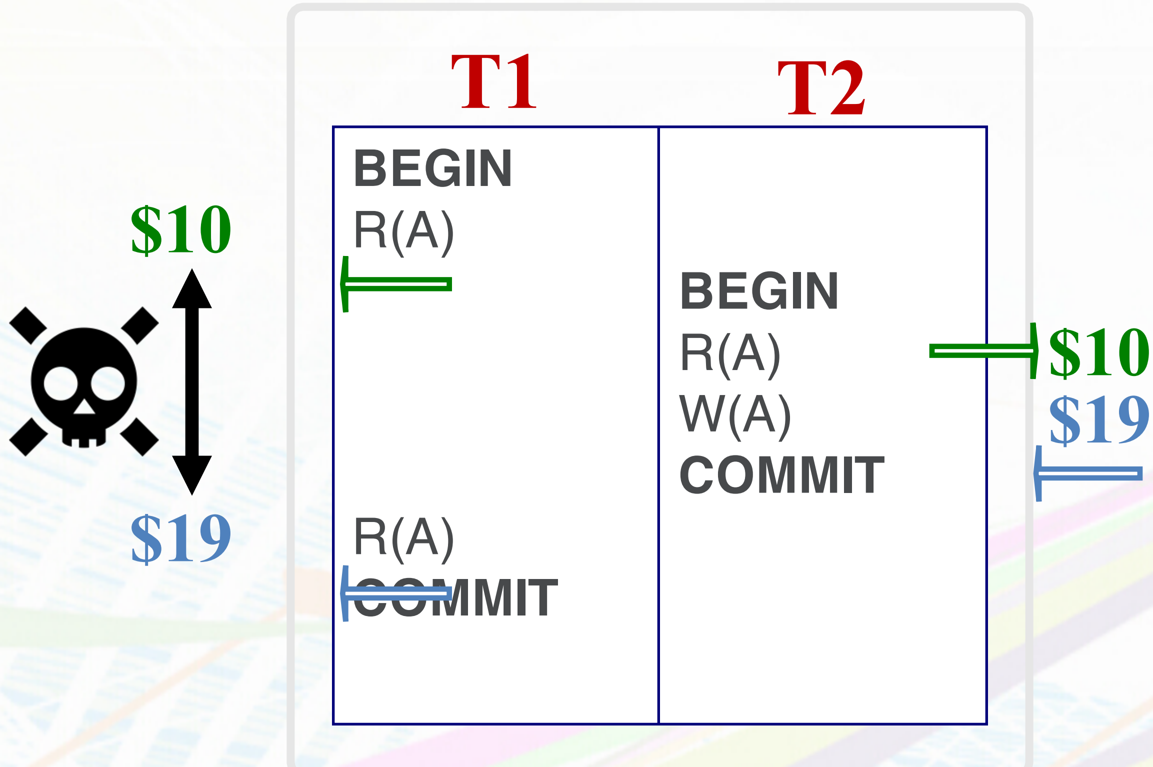
Write-Read Conflicts

- Reading Uncommitted Data, "Dirty Reads":
-



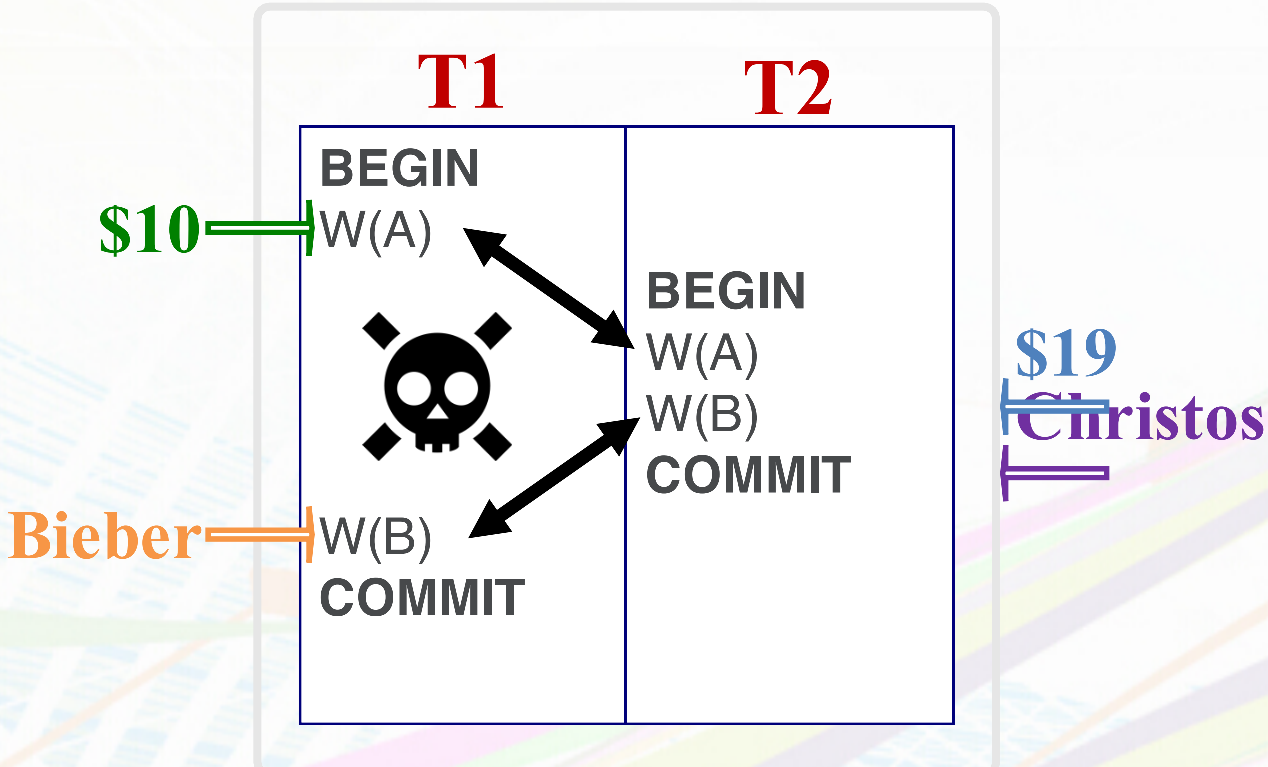
Read-Write Conflicts

- Unrepeatable Reads



Write-Write Conflicts

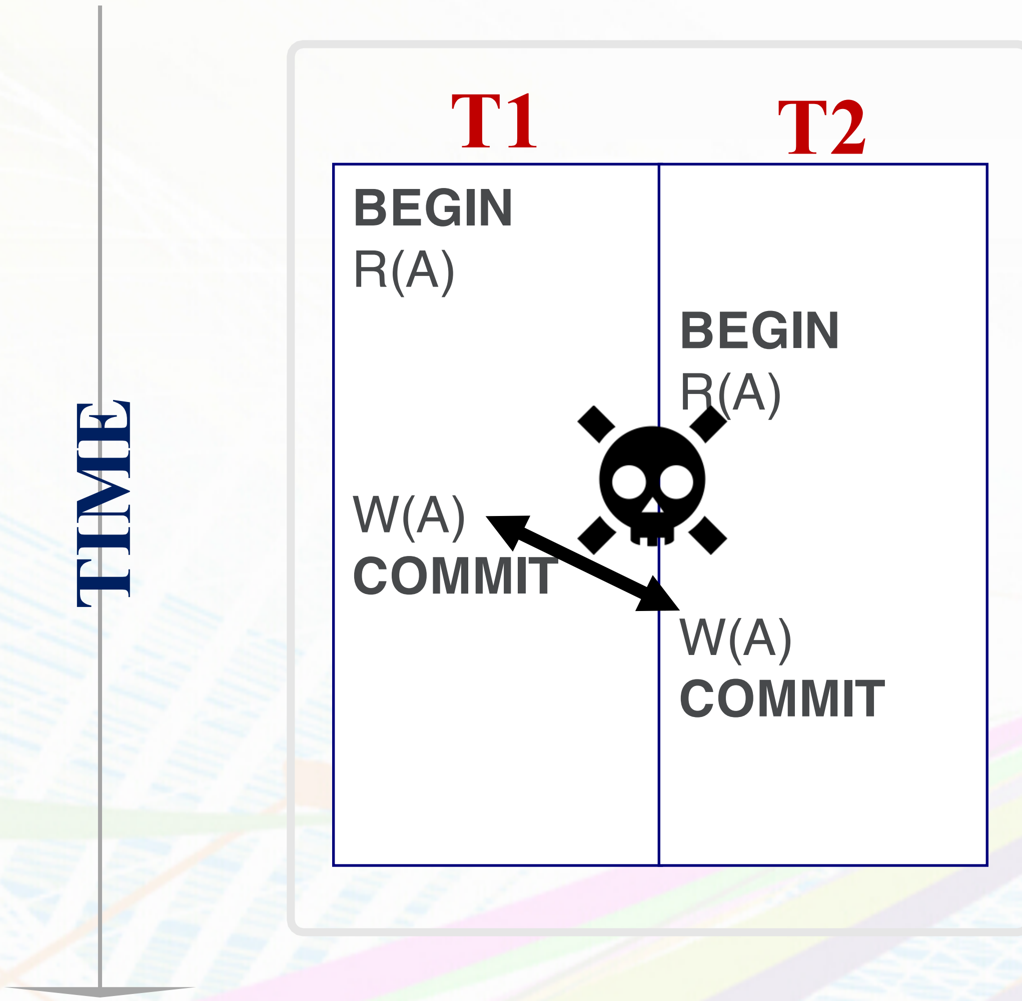
- Overwriting Uncommitted Data



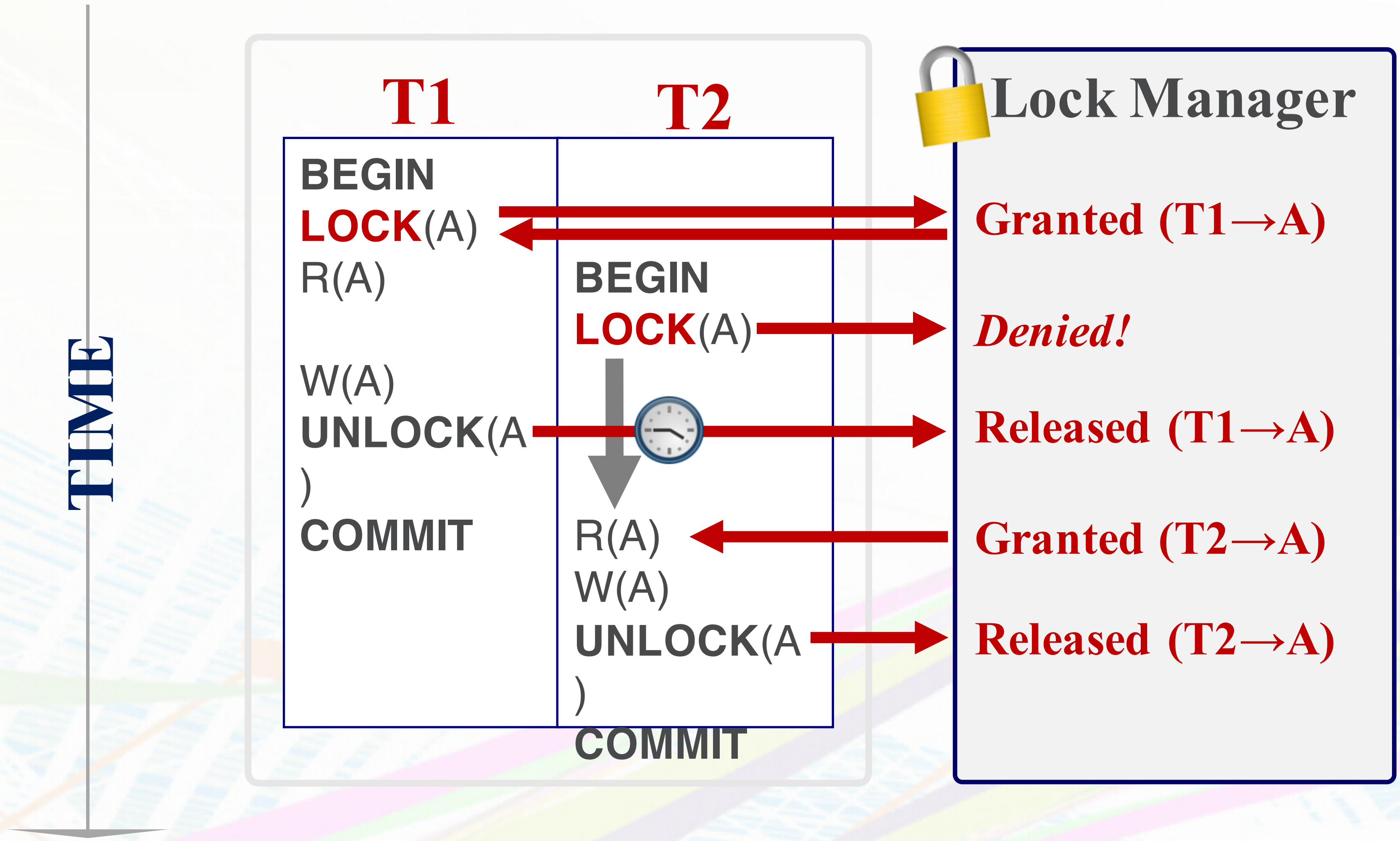
Solution

- **Q:** How could you guarantee that all resulting schedules are correct (i.e., serializable)?
- **A:** Use locks!

Executing without Locks



Executing with Locks



Executing with Locks

- **Q:** If a txn only needs to read ‘A’, should it still get a lock?
- **A:** Yes, but you can get a shared lock.

Lock Types

- Basic Types:
  - **S-LOCK** – Shared Locks (reads)
  - **X-LOCK** – Exclusive Locks (writes)

Compatibility Matrix

	Shared	Exclusive
Shared	✔	X
Exclusive	X	X

Executing with Locks

- Transactions request locks (or upgrades)
- Lock manager grants or blocks requests
- Transactions release locks
- Lock manager updates lock-table

- *But this is not enough…*

Concurrency Control

- We need to use a well-defined protocol that ensures that txns execute correctly.
- Two categories:
  - Two-Phase Locking (2PL)
  - Timestamp Ordering (T/O)

Two-Phase Locking

- **Phase 1: Growing**
  - –Each txn requests the locks that it needs from the DBMS's lock manager.
  - –The lock manager grants/denies lock requests.
- **Phase 2: Shrinking**
  - –The txn is allowed to only release locks that it previously acquired. It cannot acquire new locks.

2PL Observations

- There are schedules that are serializable but would not be allowed by 2PL.
- Locking limits concurrency.
- May lead to deadlocks.
- May still have "dirty reads"
  - Solution: **Strict 2PL**

Strict Two-Phase Locking

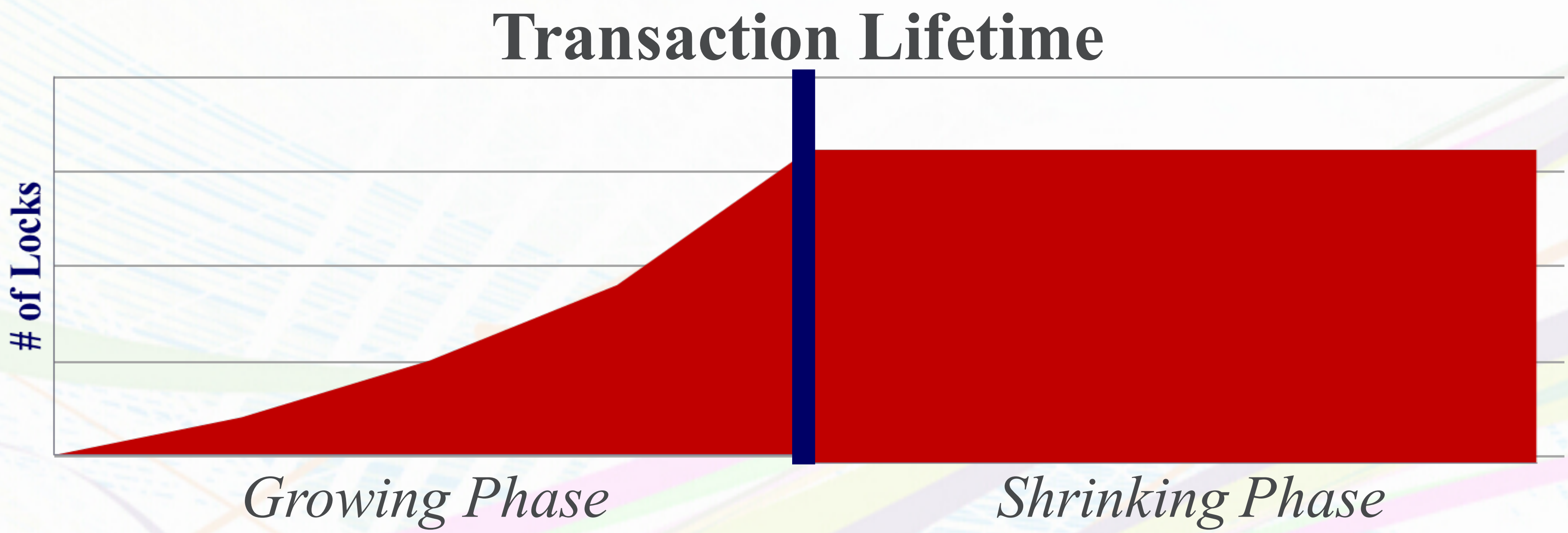
- A schedule is ***strict*** if a value written by a txn is not read or overwritten by other txns until that txn finishes.
- Advantages:
  - Recoverable.
  - Do not require cascading aborts.
  - Aborted txns can be undone by just restoring original values of modified tuples.

Strict Two-Phase Locking

- Txns hold all of their locks until commit.
- Good:
  - –Avoids "dirty reads" etc
- Bad:
  - –Limits concurrency even more
  - –And still may lead to deadlocks

Strict Two-Phase Locking

- The txn is not allowed to acquire/upgrade locks after the growing phase finishes.



Strict Two-Phase Locking

- **Q:** Why is avoiding “dirty reads” important?
- **A:** If a txn aborts, all actions must be undone. Any txn that read modified data must also be aborted.
- Strict 2PL avoids “dirty reads” (why?)

Locking in Practice

- You typically don't set locks manually.
- Sometimes you will need to provide the DBMS with hints to help it to improve concurrency.
- Also useful for doing major changes.

Overview

- Problem & ‘ACID’
- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

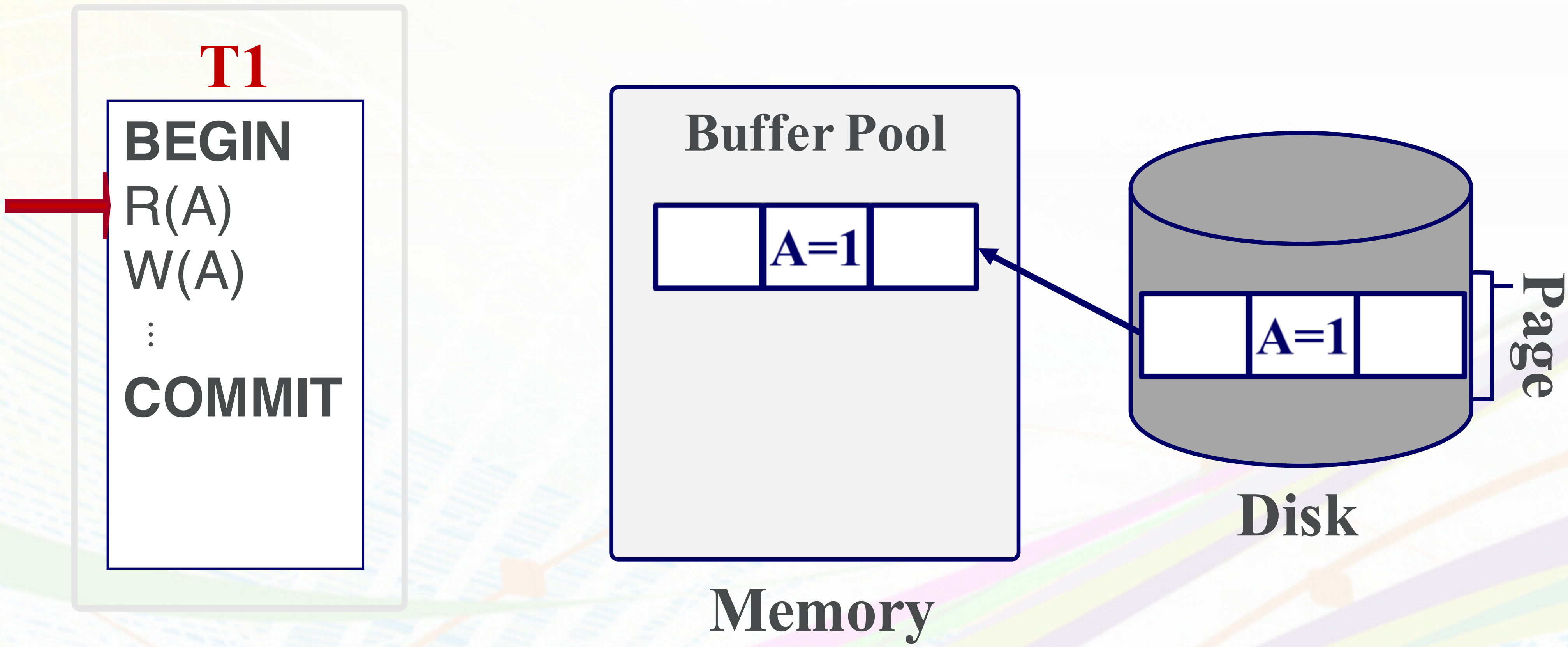
D

Transaction Durability

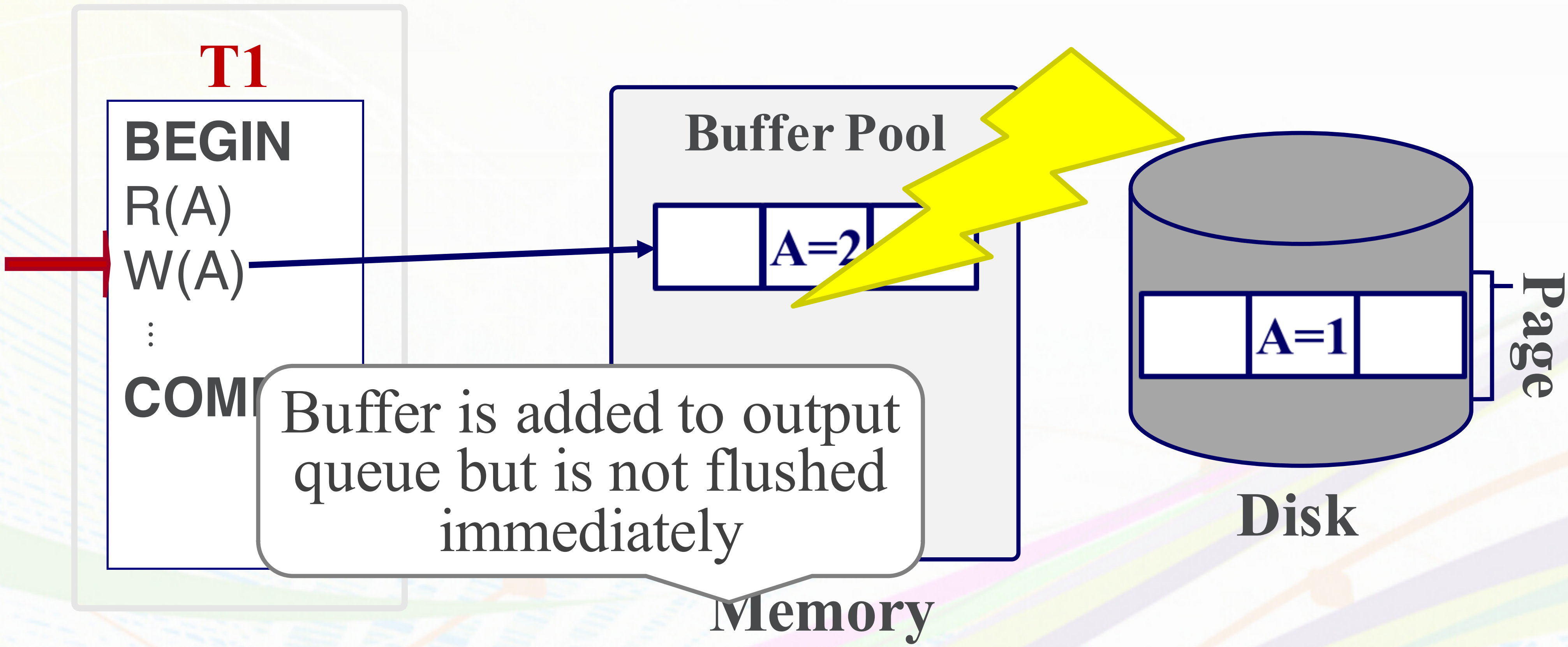
- Records are stored on disk.
- For updates, they are copied into memory and flushed back to disk at the discretion of the O.S.
  - Unless forced-output: **W(B)→fsync()**

This is slow!
Nobody does this!

Transaction Durability



Transaction Durability



Write-Ahead Log

- Record the changes made to the database in a log ***before*** the change is made.
- Assume that the log is on stable storage.

- **Q:** What to replicate?
  - The complete page?

Write-Ahead Log

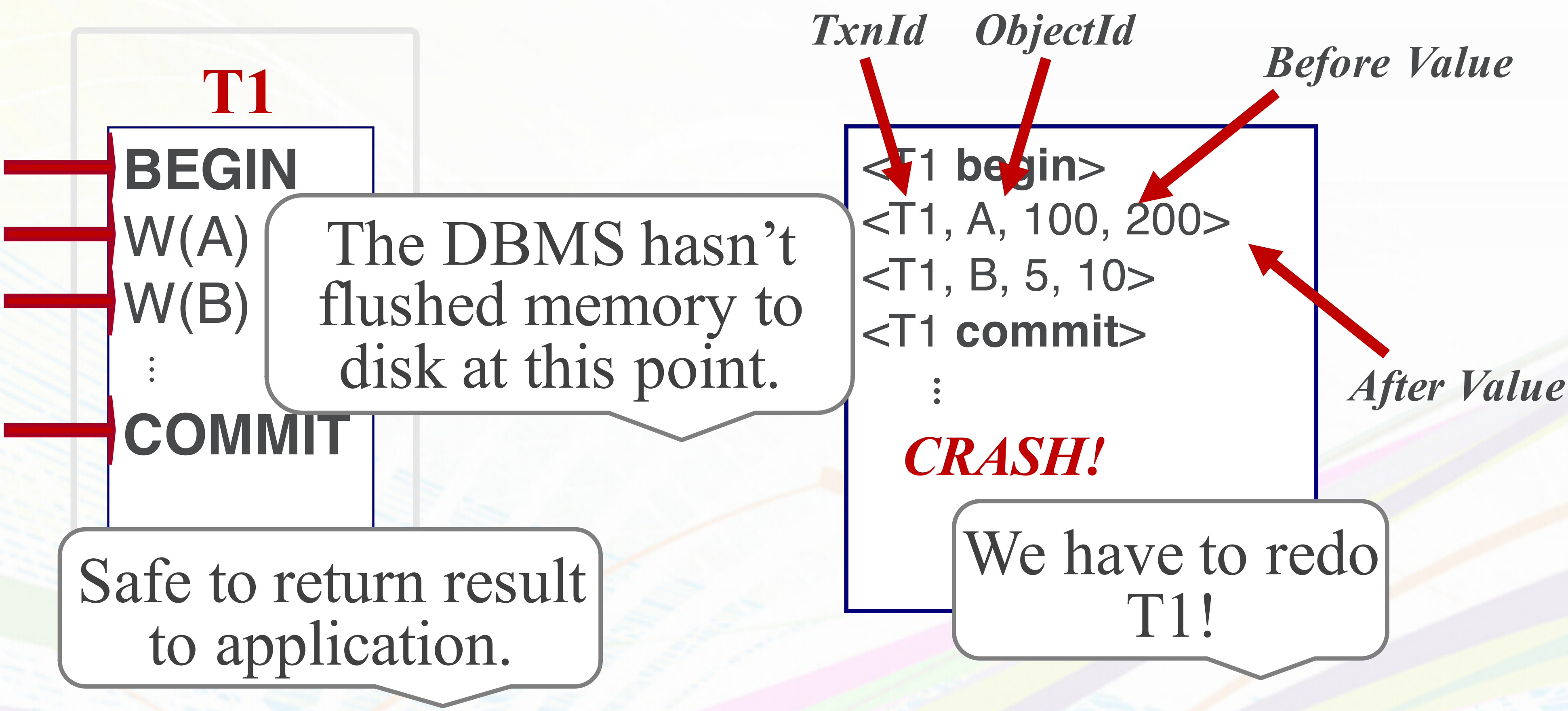
- Log record format:
  - **<txnId, objectId, beforeValue, afterValue>**
  - Each transaction writes a log record first, before doing the change
- When a txn finishes, the DBMS will:
  - Write a **<commit>** record on the log
  - Make sure that all log records are flushed before it returns an acknowledgement to application.

Write-Ahead Log

- After a failure, DBMS "replays" the log:
  - Undo uncommited transactions
  - Redo the committed ones

D

Write-Ahead Log



Write-Ahead Log

T1

BEGIN
W(A)
W(B)
⋮
COMMIT

<T1 **begin**>
<T1, A, 100, 200>
<T1, B, 5, 10>
⋮

CRASH!

We have to undo T1

Recovering After a Crash

- At the end – all committed updates and only those updates are reflected in the database.
- Some care must be taken to handle the case of a crash occurring during the recovery process!

Problems

- The log grows infinitely…
- We have to take checkpoints to reduce the amount of processing that we need to do.

ACID Properties

- **Atomicity:** All actions in the txn happen, or none happen.
- **Consistency:** If each txn is consistent, and the DB starts consistent, it ends up consistent.
- **Isolation:** Execution of one txn is isolated from that of other txns.
- **Durability:** If a txn commits, its effects persist.

Summary

- **Concurrency control** and **recovery** are among the most important functions provided by a DBMS.
- Concurrency control is automatic
  - System automatically inserts lock/unlock requests and schedules actions of different txns.
  - Ensures that resulting execution is equivalent to executing the txns one after the other in some order.

Summary

- Write-ahead logging (WAL) and the recovery protocol are used to:
  - Undo the actions of aborted transactions.
  - Restore the system to a consistent state after a crash.

Overview

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Recovery

Concurrency Control