Week 10 Workshop - Database Transactions

Transactions

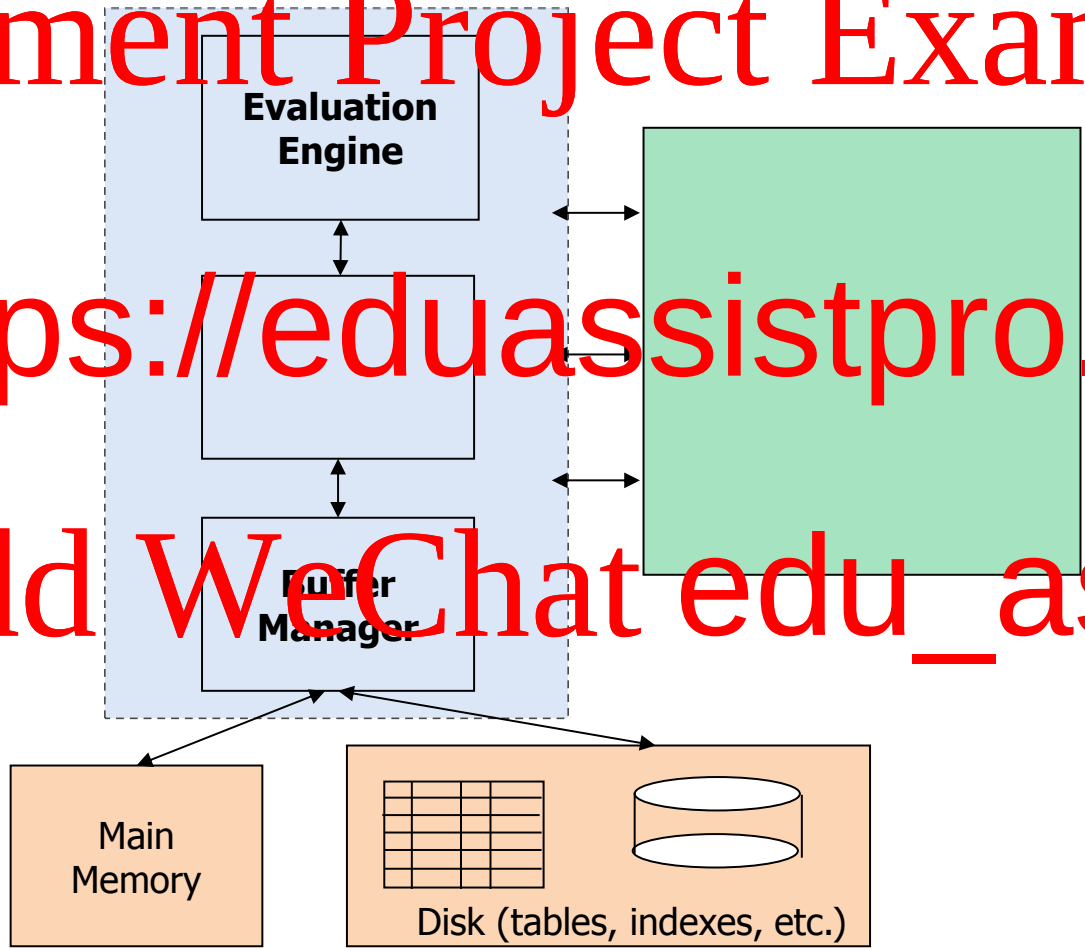
- A **transaction** is a sequence of database operations grouped together for execution as a logic unit in a DBMS.


```
5    COMMIT;
```

Transactions

- What's the difference between database transactions and programs written by a programming language like C, Java and Python?
- How are transactions handled in the query processing?

Transaction Manager - A Simplified View



Transactions - ACID Properties

Transactions

T_2 : `SELECT ...`

T_3 : `INSERT ...`

T_4 :
```
BEGIN TRANSACTION
    SELECT ...
    DELETE ...
    ABORT
```

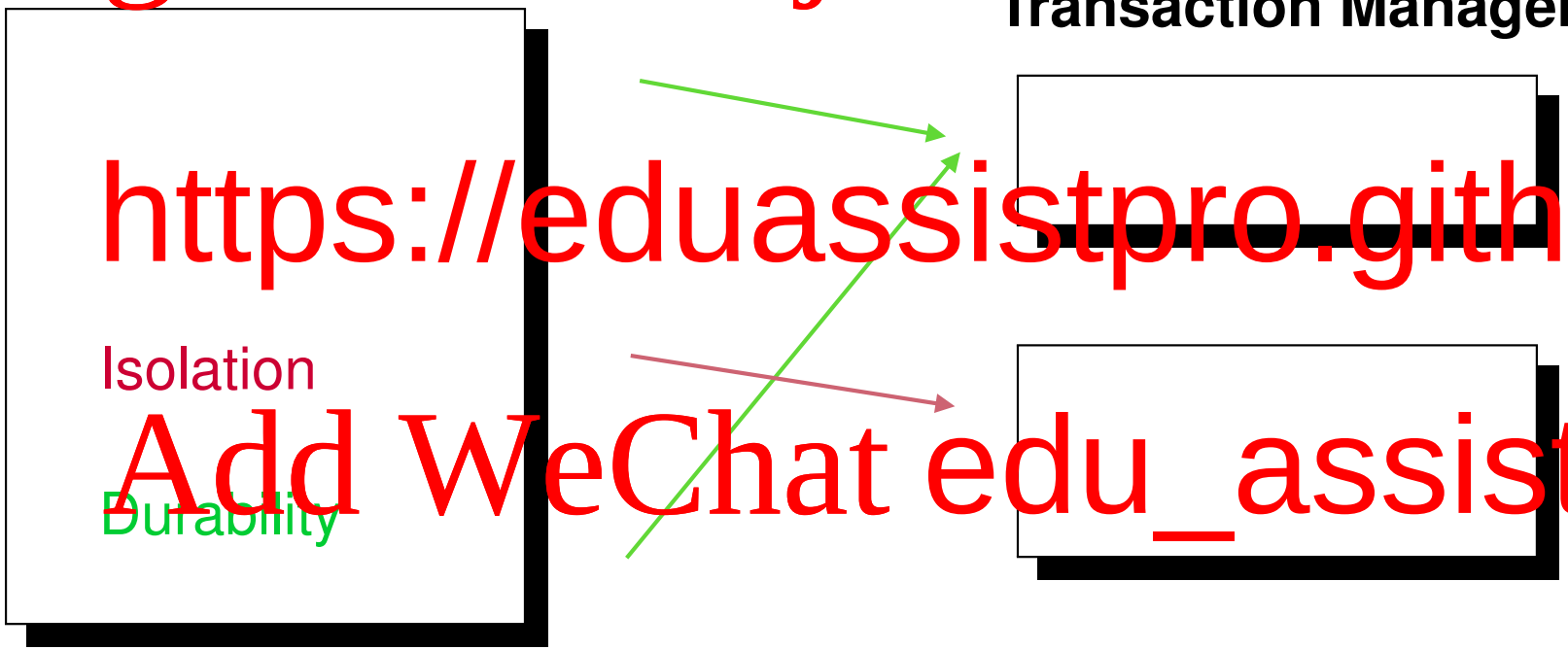
ACID properties

Consistency

Durability

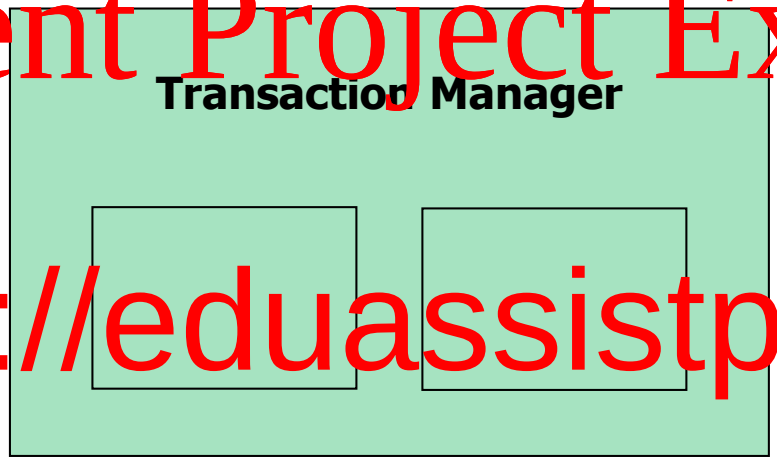
Transactions - ACID Properties

Transactions - ACID Properties



Consistency is the responsibility of an application developer.

Transaction Manager - Common Techniques



- **Logging** for recovery—assuring **atomici**

  e.g., Write-Ahead Log (WAL) Protocol

Logging - Introduction

- A **transaction log** is an append-only file that records changes to objects made by transactions.
- W
- A t
  da
- Recovery amounts to **either undoing or red** :
  - **Undo** the operations that have not been co
  - **Redo** the operations that have been com written to disk.
- **Checkpoints** tell the points from which to begin applying transaction logs during database recovery.

Write-Ahead Log (WAL) Protocol

- **Write-Ahead Log** (WAL) requires that a record of every change to a database is available while attempting to recover from a crash.

  - A record in the log must be written to persistent committing the transaction.

- Accordingly, the definition of a **committe**

  " **A transaction, all of whose log records, including a commit record, have been written to persistent storage**".

Write-Ahead Log (WAL) Protocol

- Typical fields in a log record:

  - Each log record has a unique id called LSN (Log
  - Possible types include: update, commit, abort, end, etc.

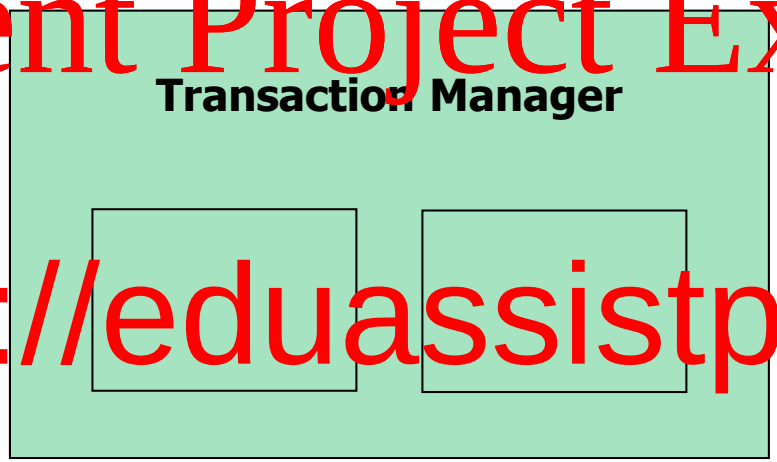
- Does WAL bring in some benefits for performance
  - Often results in a significantly reduced number of disk writes
  - Supports one sync against the log file instead of potentially many against the data files
  - Enables online backup and point-in-time recovery

Transaction Manager - Recovery

- Key concepts to aid in recovery:
  -
    - **Undo** ...
    - **Redo** ...
  -
  - **Checkpoint**: snapshot of the state of a data

    (Widely used in practice, but not covered in this course)

Transaction Manager - Common Techniques



- **Logging** for recovery – assuring **atomici**

  e.g., Write-Ahead Log (WAL) Protocol

- **Locking** for concurrency control – assuring **isolation** of transactions

  e.g., Two-Phase Locking (2PL) Protocol

Locking - Introduction

- A lock is associated with an object, e.g., file, table, record, page, etc.
- T

(Note: there are other types of locks defined by differ

- Lock compatibility:

Lock type	read-lock	write-lock
read-lock	Yes	No
write-lock	No	No

Two-Phase Locking (2PL) Protocol

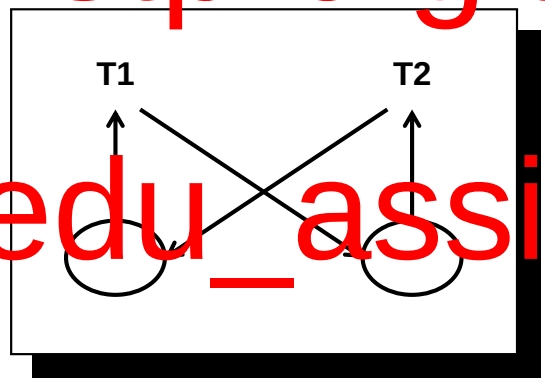
- Locks are handled in two phases:
  -
  -

Two-Phase Locking (2PL) Protocol

Bad news:

- 2PL can radically limit interleaving among transactions in some cases ...
- 2 tra

	1	2
1	lock-r(A)	
2	read(A)	
3		lock-r(B)
4		read(B)
5	lock-w(B)	
6	write(B)	
7		lock-w(A)
8		write(A)



- T_1 is waiting for T_2 to get a write-lock on B. T_2 is waiting for T_1 to get a write-lock on A.

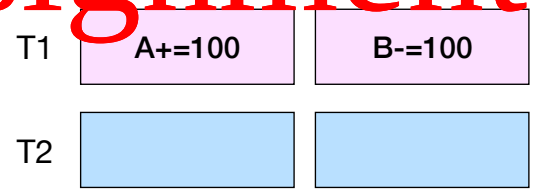
Two-Phase Locking (2PL) Protocol

Good

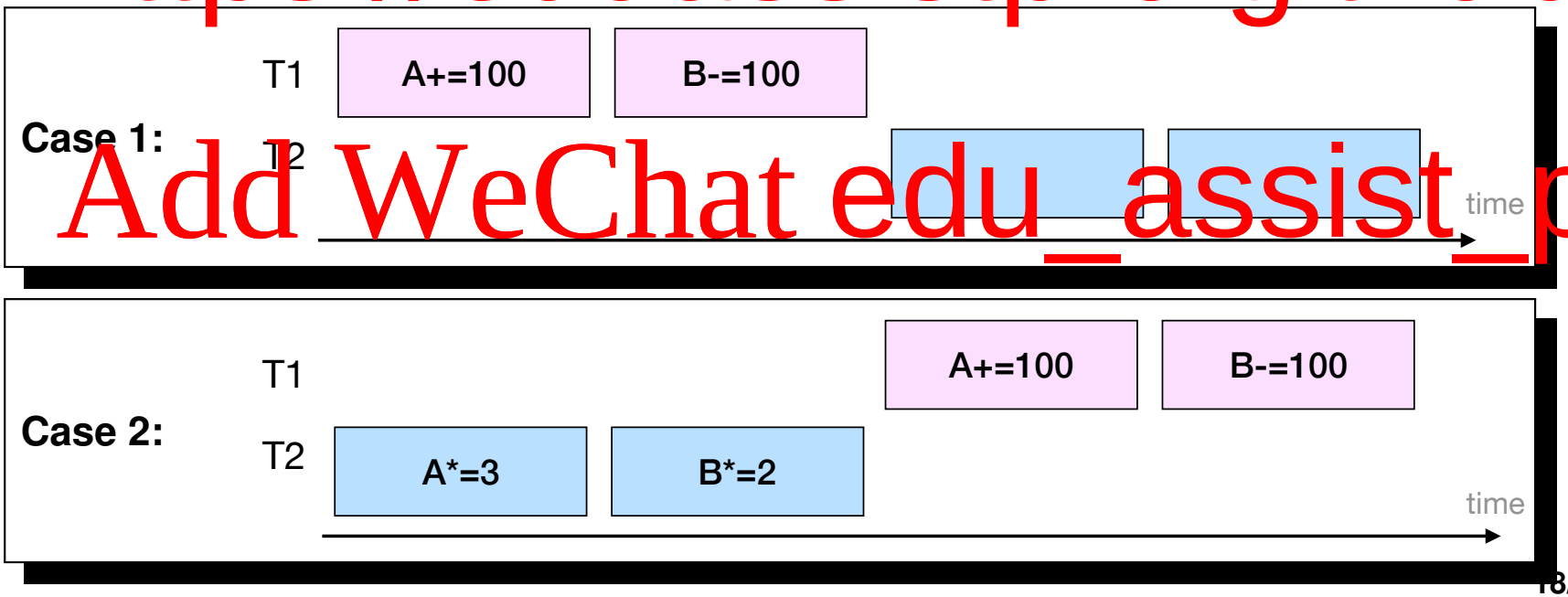
- 2
  tra
  - **Serializability** means that a resulting da
    database state of running transactions ser
  -
    Serializability is the major **correctn**
    transactions.

Serializability - Example

- Consider $A = 200$ and $B = 500$, and we have two concurrent transactions:

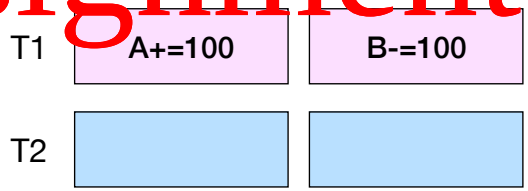


- S

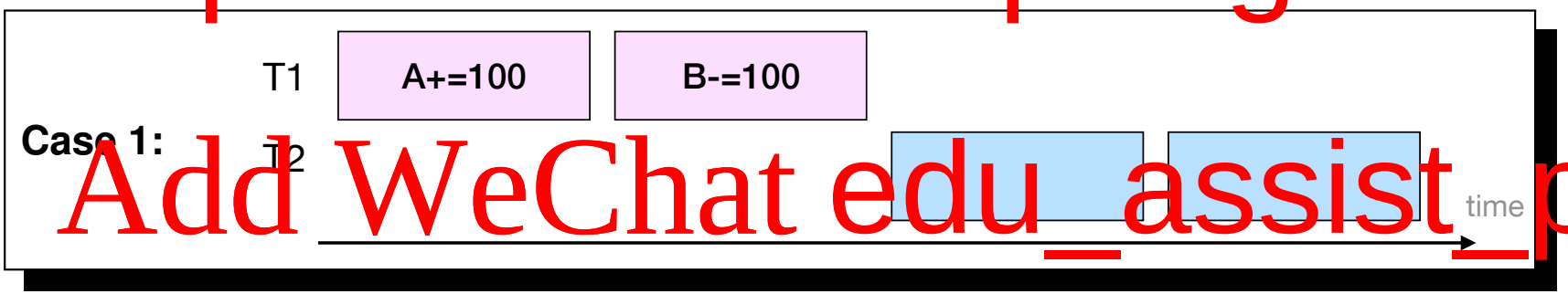


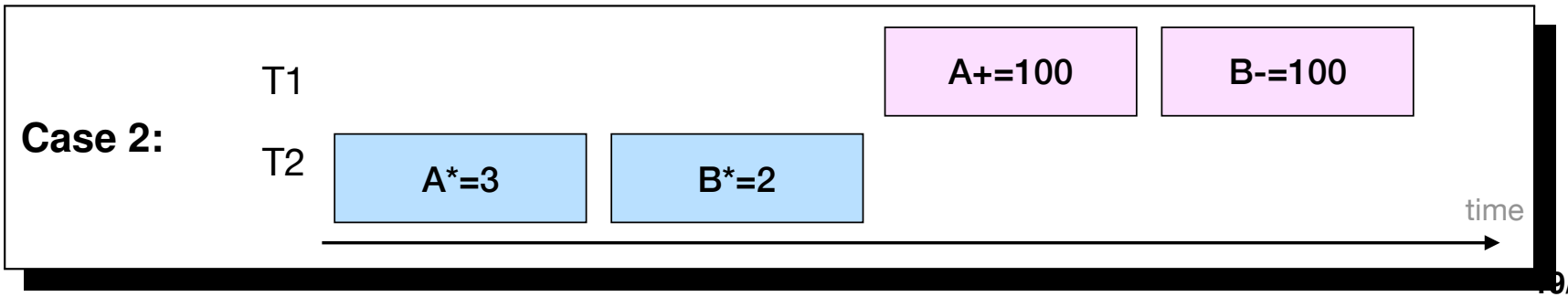
Serializability - Example

- Consider $A = 200$ and $B = 500$, and we have two concurrent transactions:



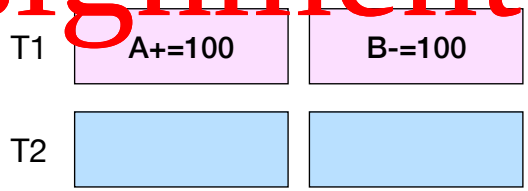
- **Case 1**: A=900 and B=800
-
- S



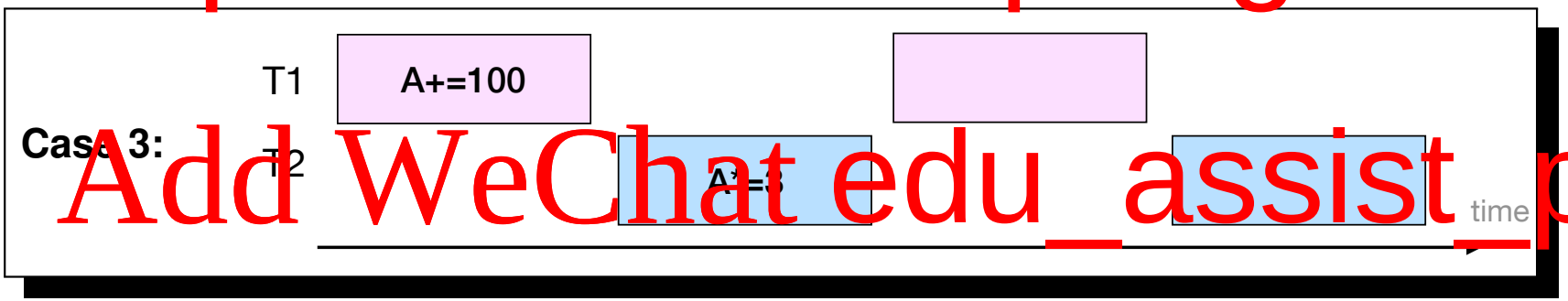


Serializability - Example

- Consider $A = 200$ and $B = 500$, and we have two concurrent transactions:



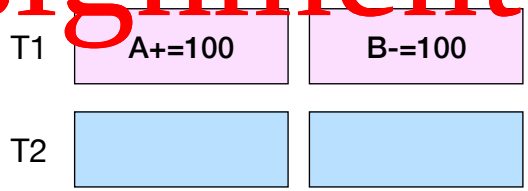
- **Case 1**: A=900 and B=800
-
- Ar



- Yes. A=900 and B=800 $\hookrightarrow$ equivalent to Case 1!

Serializability - Example

- Consider $A = 200$ and $B = 500$, and we have two concurrent transactions:

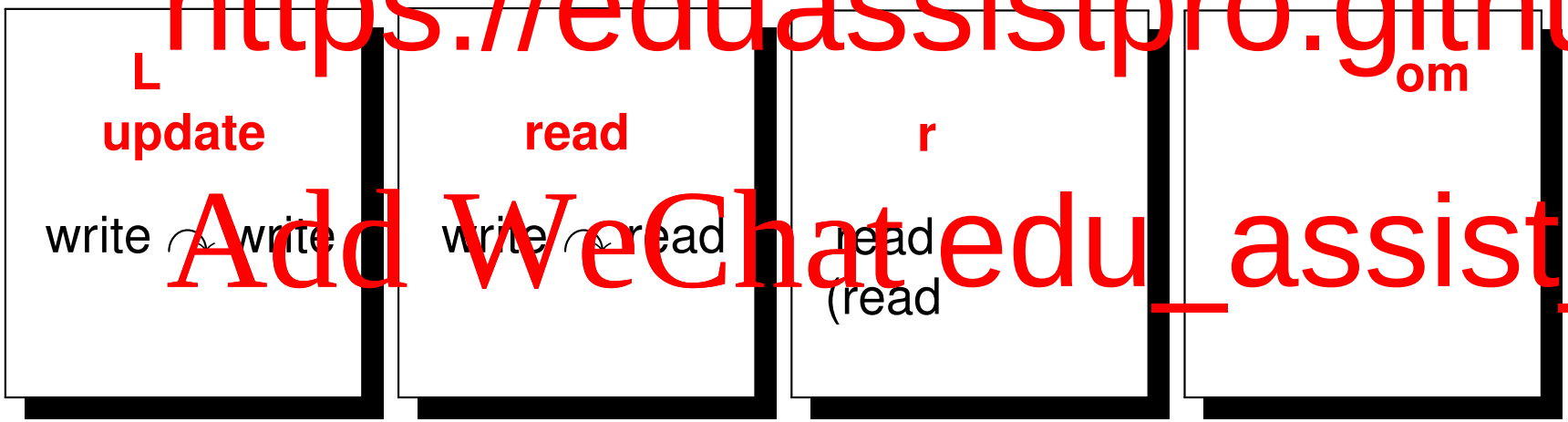


- **Case 1**: A=900 and B=800
-
- Ar

- No. A=900 and B=900 $\hookrightarrow$ not equivalent to Case 1 or Case 2!

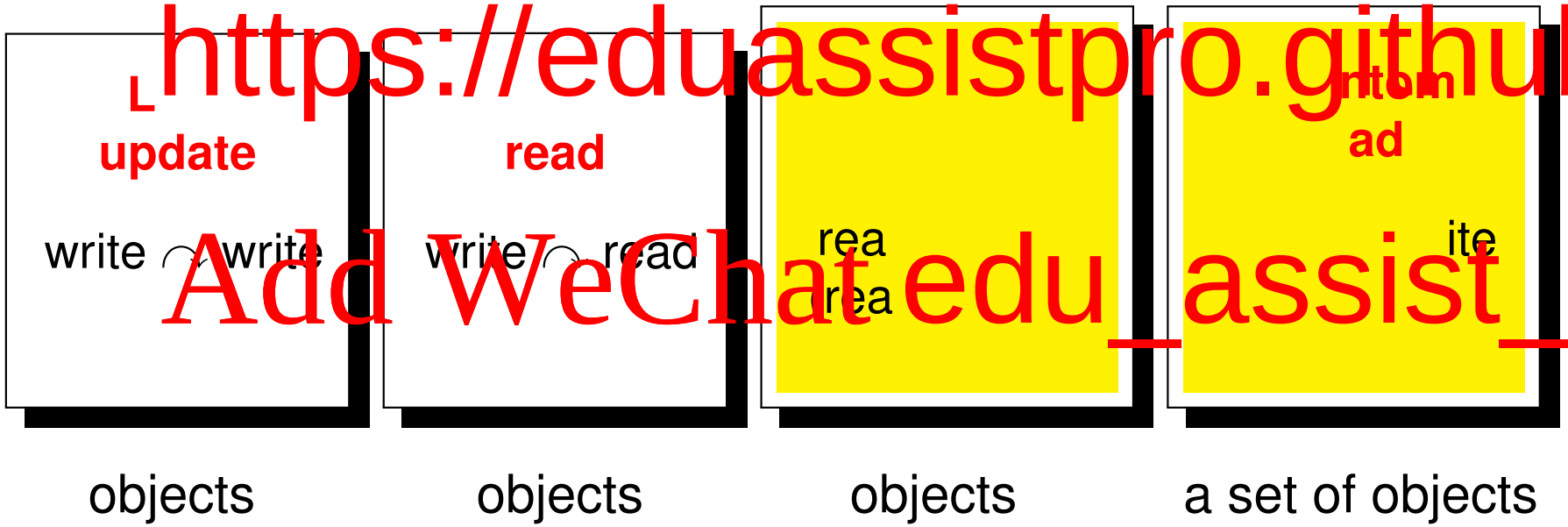
Problems in Concurrent Transactions

- If no concurrency control for transactions, some problems may occur:



Problems in Concurrent Transactions

- If no concurrency control for transactions, some problems may occur:



The Lost Update Problem - Another Example

- Ben and Amy have the same salary. T_1 sets their salaries to \$80,000, and T_2 sets their salaries to \$90,000.

1. If executing T_1 and T_2 sequentially,

2. If executing T_1 and T_2 concurrently,

	T_1	T_2
1	write(A) (A:=80000)	
2		write(A) (A:=
3		write(B) (B:=90000)
4		commit
5	write(B) (B:=80000)	
6	commit	

↪ It is not acceptable!

The Dirty Read Problem - Another Example

- Both Ben and Amy are rewarded a bonus $5,000 and a pay rise 5%. T_1 increases their salaries with $5,000 and T_2 increments their salaries by 5%.

1. If executing T_1 and T_2 sequentially, they would have .... Also, T_1 or T_2

2.

	T_1	T_2
1	read(A)	
2	write(A) (A:=A+5000)	
3		read(A)
4	read(B)	
5	write(B) (B:=B+5000)	
6	abort	
7		write(A) (A:=A+A×5%)
8		read(B)
9		write(B) (B:=B+B×5%)
10		commit

↪ It is not acceptable!

The Unrepeatable Read Problem - Another Example

- Amy and Ben are using a website to book flight tickets to Brisbane.

	T_1 (from Amy)	T_2
1	read(X)	
2		re
3		wr
4		commit
5	read(X)	

- This situation can never arise in a serial execution of T_1 and T_2.

The Phantom Read Problem - Another Example

- Amy is 30 years old, but her age in the table players is mistakenly recorded as 40. Ben is 28 years old and his age is correctly recorded in players.
- Suppose that we have the following two current transactions:

T

```
WHERE age<32;
COMMIT;
```

```
COMMIT;
```


	T_1	T_2
1	read(players)	
2		re
3		write(players)
4		commit
5	read(players)	
6	commit	

- This situation also can never arise in a serial execution of T_1 and T_2.

Discussion

→ What are the differences between "unrepeatable read" and "phantom read"?

- **Unrepeatable read**
  - t
  - DELETE from
  - Can be prevented using record-lev
- **Phantom read**
  - Executing the same SELECT tw tuples;
  - May occur when querying a set of tuples that are affected by INSERT/DELETE/UPDATE from another transaction;
  - Can be prevented using table-level locking.

What Should We lock?

- Consider the following two concurrent transactions again:

T_1: `SELECT * FROM` players
`WHERE` age<32;

T_2: `UPDATE` players

my';

- What objects should the DBMS lock in order to avoi problem?

  - **Table-level locks**
    e.g., read-lock on players for T_1, writ 2
  - **Record-level locks**
    e.g., read-lock on every record with age<32 for T_1, write-lock on every record with rating=8 and name='Amy' for T_2
  - ...

Transaction Support in SQL

- An explicit transaction may have no `BEGIN TRANSACTION` statement, but must be ended with either `COMMIT` or `ABORT` (`ROLLBACK`) statement.
- W
  st
- T
  th
  transaction is prepared to tolerate on concurrent tr
  - Key idea:

> To trade off **consistency** (i.e., increased risk of violating database integrity) with **performance** (i.e., greater concurrent access to data)

Isolation Levels

- SQL-92 defines four isolation levels:
  1. **Read Uncommitted**
  2.
  3.
  4. **Serializable**

- To specify an isolation level, e.g.,

```
SET TRANSACTION ISOLATION LEVEL serializab
```

- The SQL standard does not impose a specific locking scheme or mandate particular behaviors.

Isolation Levels

- The intention is to prohibit certain problems:

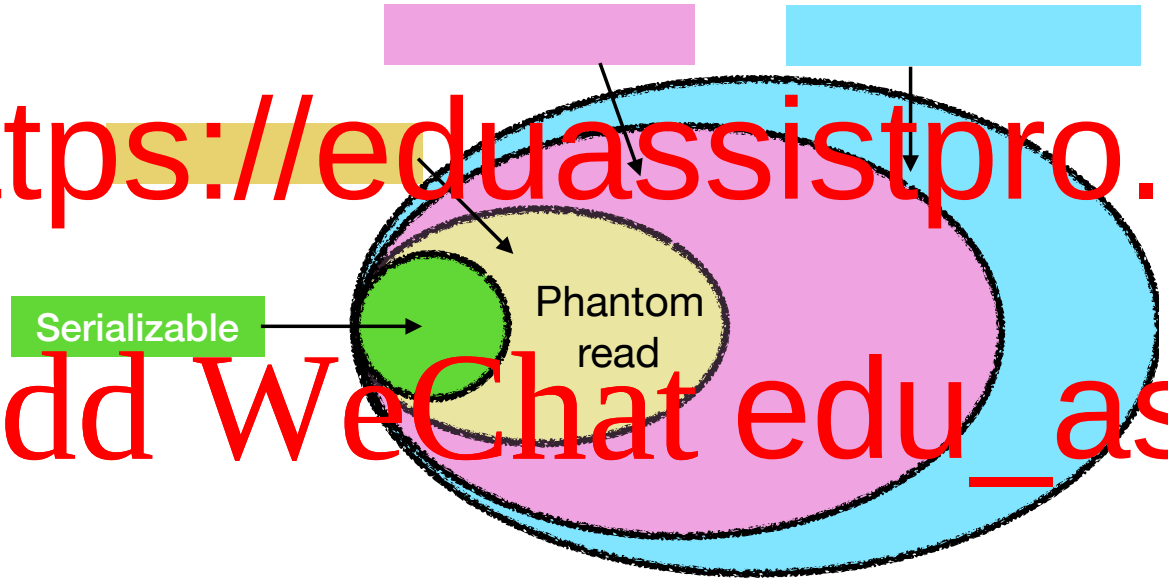
I			ead
RE R RE SERIALIZABLE			

- Different DBMSs implement isolation levels quit
- The isolation level required for **Lost Update** is debatable (depending on a DBMS's implementations). But in general, it may require the highest level `SERIALIZABLE` to prevent it. [1]

[1] https://drtom.ch/posts/2011/11/12/The_Lost_Update_Problem_-_Part_1/

Isolation Levels - Concurrency Control

- A DBMS provides different levels of isolation → different degrees of concurrency control to prevent different problems.



- Concurrency control is **NOT** binary in a database system.

Isolation Levels - Read Uncommitted

- **Read Uncommitted** is the least restrictive isolation level
  - One transaction can see changes made by other transactions which are not ye
  - U w

Isolation Levels - Read Committed

- **Read Committed**: One transaction only sees committed changes by other transactions
- It is the most commonly used isolation level in database applications.
- U
no

Isolation Levels - Repeatable Reads

- **Repeatable Reads**: The objects touched by a transaction are locked and cannot be updated or deleted by a concurrent transaction.
- U
do

Isolation Levels - Serializable

- **Serializable** is the highest solution level. All transactions are totally isolated from other transactions. It is safe but may cause significant performance hit.

- U
do
dif

Locks Taken by SQL Server for Isolation Levels [2]

[2] http://michaeljswart.com/2012/06/visualizing-transaction-isolations-for-sql-server/

Wrap-up - Isolation Levels

- A lower isolation level increases **the ability of many users to access data at the same time**, but also increases the number of concurrency effects (s
- C https://eduassistpro.github.i
  ef es
  and increases the chances that one transaction will block another.
- Choosing the appropriate isolation level depend
  - **the data integrity requirements of the ap**

    against
  - **the overhead of each isolation level**.

Research Topics

- This is an active research area covering many interesting research topics.
- Hi
  da
- H
  system is a relational database system or somethi
  - Distributed database systems
  - Graph database systems
  - Document-oriented database systems
  - ...

Research Topics

- Distributed transactions