COMP207 Database Development

Lecture 3

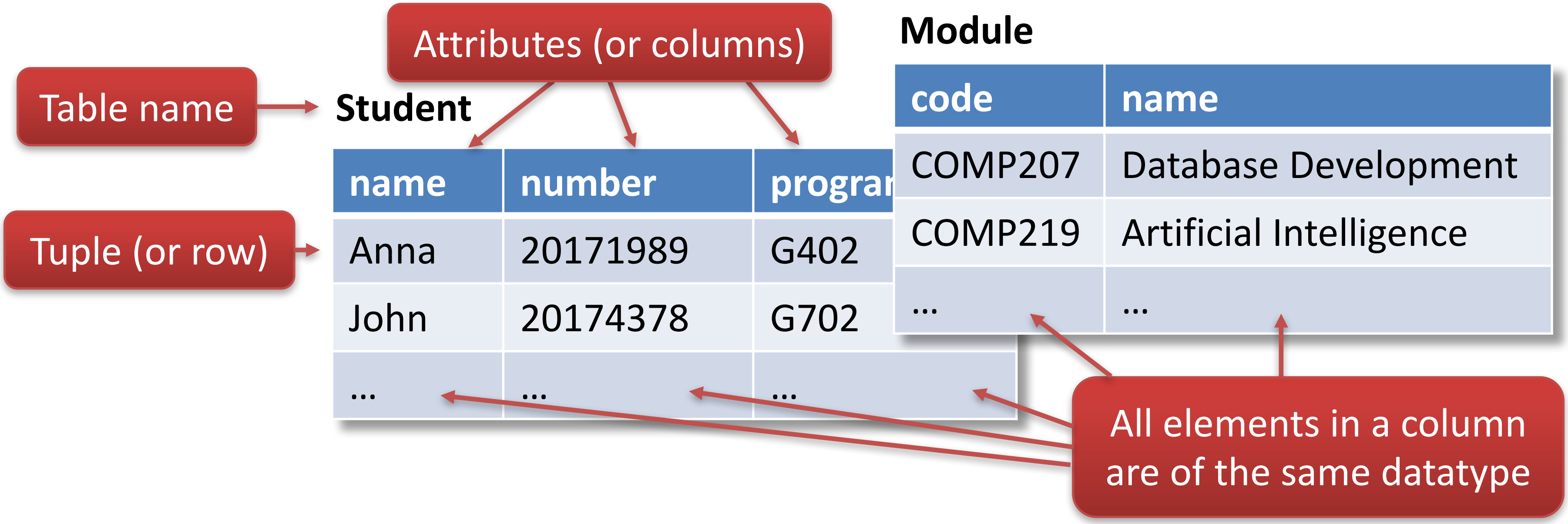
Transaction Management: Introduction

Transaction Management

- Transactions: Sequence of queries
- Ensures that operations on a database are executed without “damaging” the database
- This lecture:
  - Introduces the central concept of a transaction
  - Outlines how transactions help to execute database operations safely
  - Discusses desired properties of transactions
- Lectures 4-11:
  - How DBMS process transactions

Relational Model

- Data is organised in **tables** (also called **relations**)



Student

name	number	progra
Anna	20171989	G402
John	20174378	G702
…	…	…

Module

code	name
COMP207	Database Development
COMP219	Artificial Intelligence
…	…

- **Schema**: description of all tables in the database

Student(name, number, programme)

Module(code, name)

Accessing Relational Databases

- Modern DBMS allow it to define & access relational databases using **SQL statements**
  - SQL as a *Data Definition Language (DDL)*

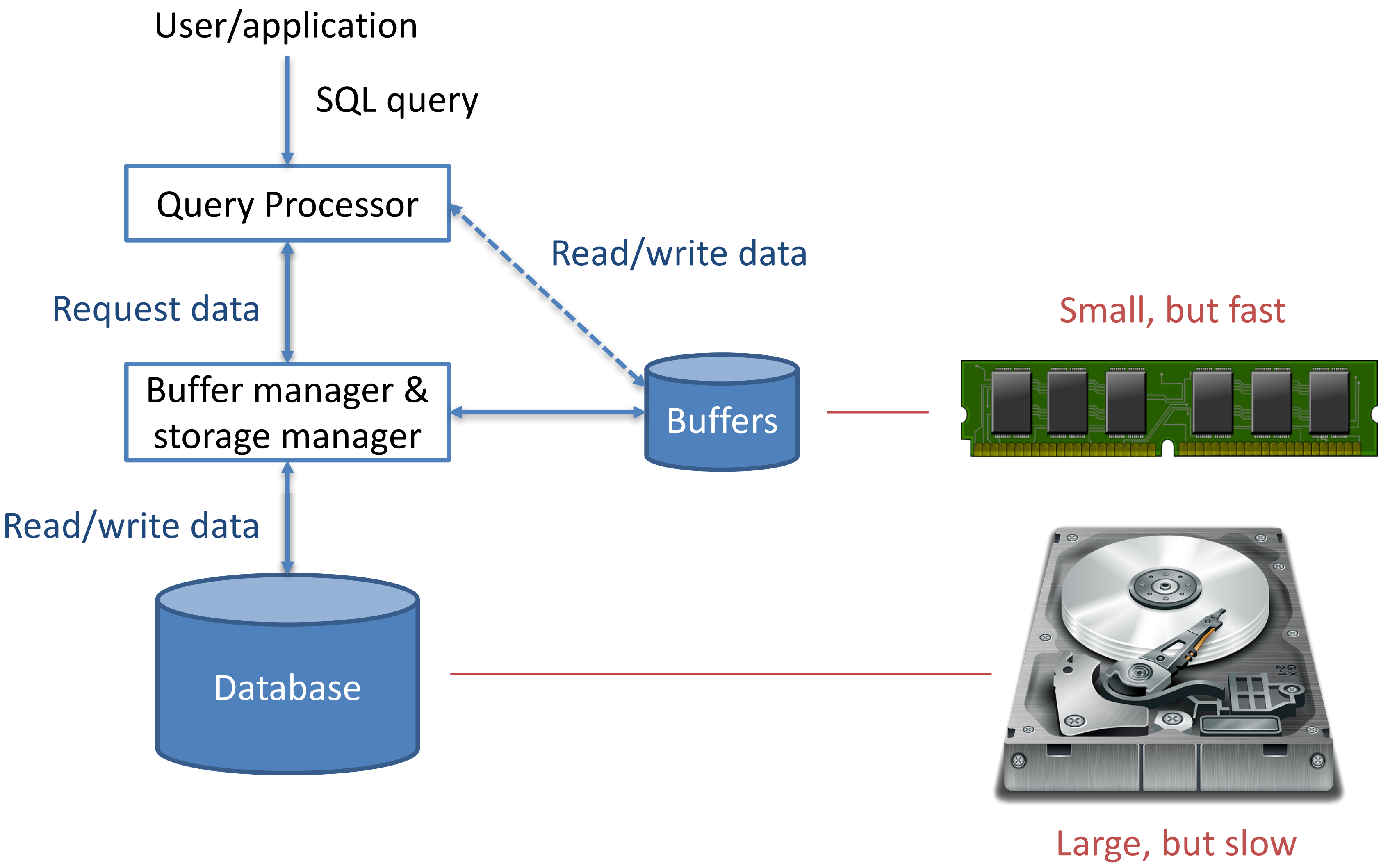
```
CREATE TABLE Student
( name      VARCHAR(15),
  number    INT          PRIMARY KEY,
  programme VARCHAR(15) );
```

  - SQL as a *Data Modification Language (DML)*

```
SELECT number
FROM Student
WHERE programme = ’G402’;
```

- **SQL query** = SELECT/INSERT/UPDATE/DELETE statement

Execution of SQL Queries
(Simplified)



User/application

SQL query

Query Processor

Request data

Read/write data

Buffer manager & storage manager

Buffers

Small, but fast

Read/write data

Database

Large, but slow

Relational Databases on Disk

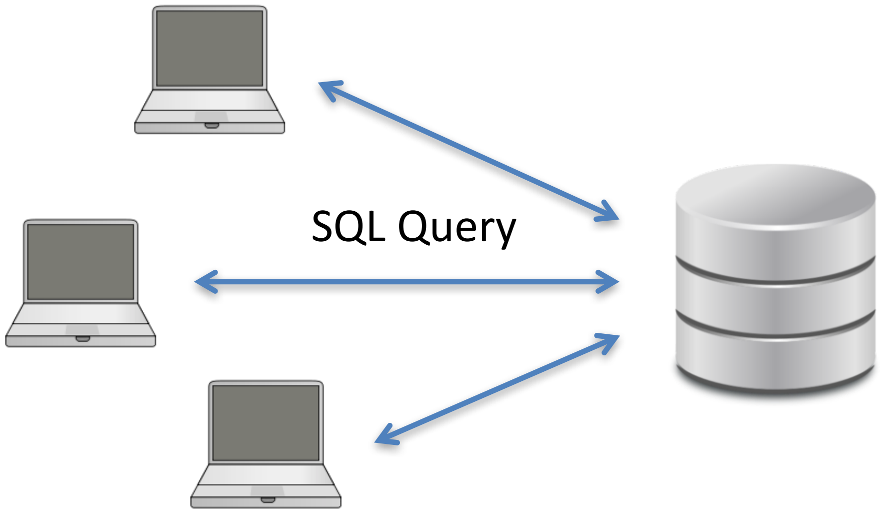
Relational Model				File Model
Relation	=	“Table”	=	File
Tuple	=	“Row”	=	Record
Attribute	=	“Column”	=	Value

- Further terminology:
  - **Environment**: a relational DBMS on a computer system
  - **Catalog**: a collection of schemas in an environment

Transactions in SQL

Executing SQL Queries in ”Real Life”

- So far: SQL queries in isolation

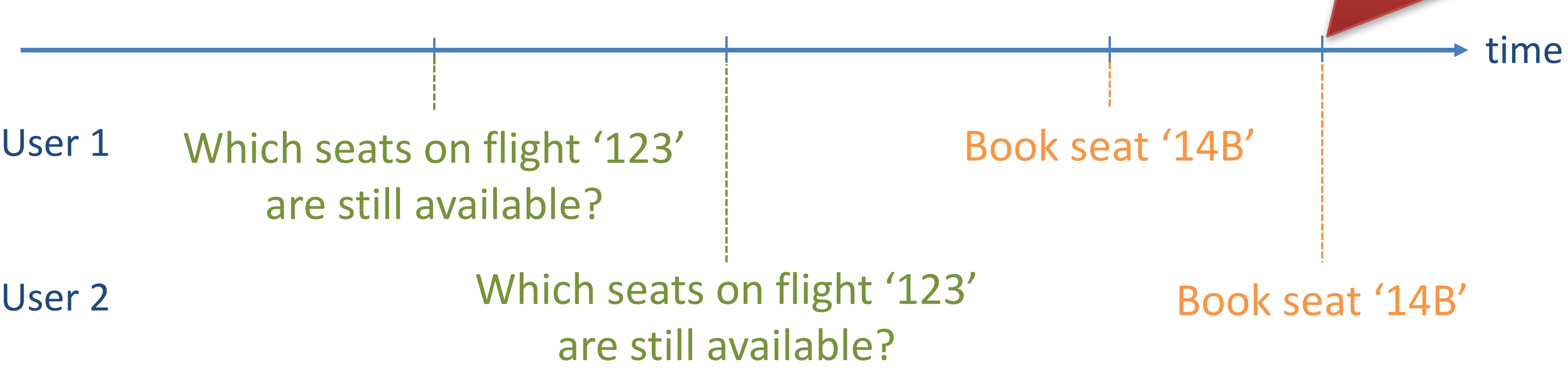


- In practice, problems may arise due to
  - **Concurrency**: SQL statements that overlap in time
  - **Partial execution** of SQL statements (e.g., due to failures)

Problem 1: Concurrency

Flights(flightNo, date, seatNo, seatStatus)

Might lead to an inconsistent database

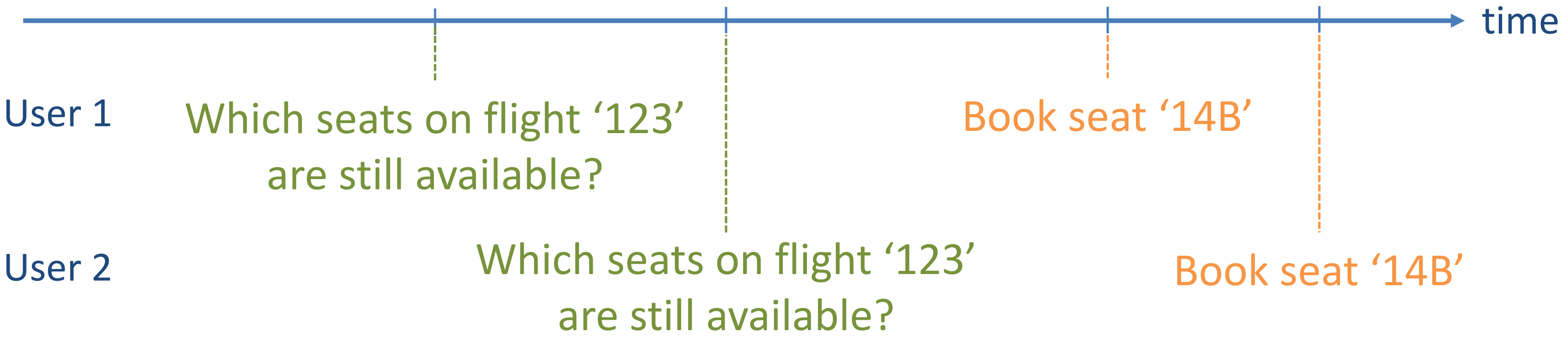

time
User 1
Which seats on flight ‘123’ are still available?
Book seat ‘14B’
User 2
Which seats on flight ‘123’ are still available?
Book seat ‘14B’

```
SELECT seatNo
FROM   Flights
WHERE  flightNo = 123
   AND date = ’2018-10-2
   AND seatStatus = ’available’;
```

```
UPDATE Flights
SET    seatStatus = ’occupied’
WHERE  flightNo = 123
   AND date = ’2018-10-2’
   AND seatNo = ’14B’;
```

Problem 1: Concurrency

Flights(flightNo, date, seatNo, seatStatus)



time

User 1

Which seats on flight ‘123’ are still available?

Book seat ‘14B’

User 2

Which seats on flight ‘123’ are still available?

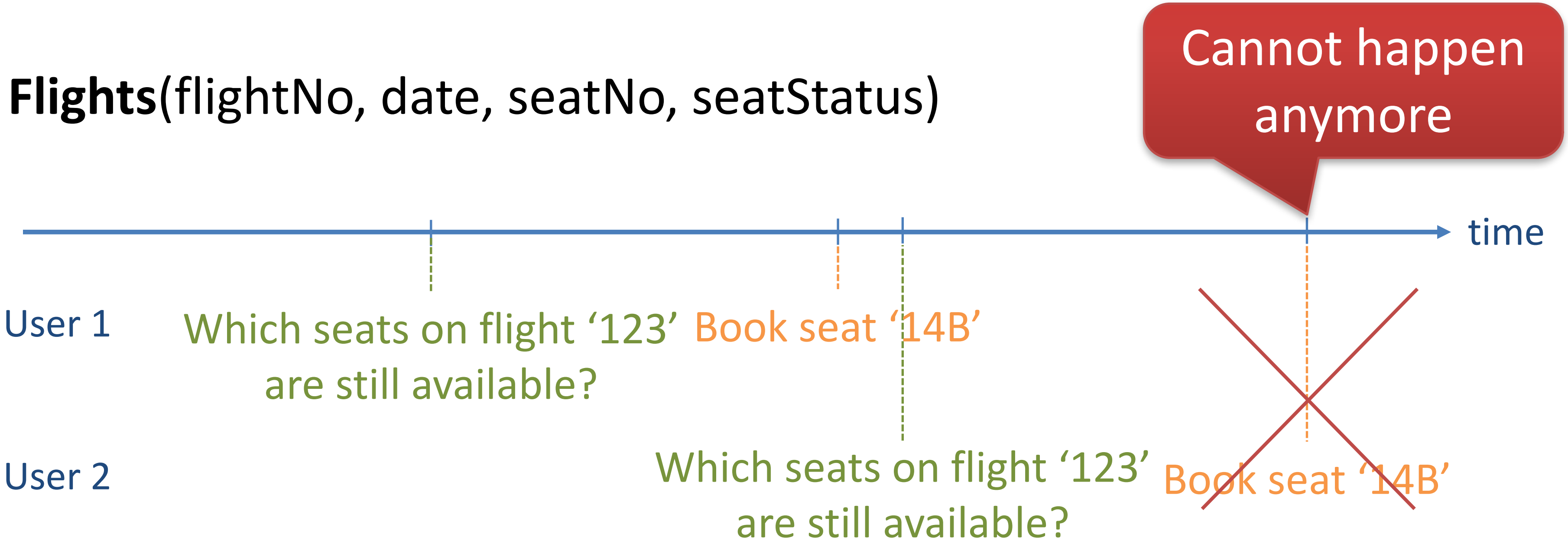
Book seat ‘14B’

```
SELECT seatNo
FROM   Flights
WHERE  flightNo = 123
   AND date = ’2018-10-2
   AND seatStatus = ’available’;
```

```
UPDATE Flights
SET    seatStatus = ’occupied’
WHERE  flightNo = 123
   AND date = ’2018-10-2’
   AND seatNo = ’14B’;
```

Problem 1: Concurrency

Flights(flightNo, date, seatNo, seatStatus)


Cannot happen anymore
time
User 1
Which seats on flight ‘123’ are still available?
Book seat ‘14B’
User 2
Which seats on flight ‘123’ are still available?
~~Book seat ‘14B’~~

```
SELECT seatNo
FROM   Flights
WHERE  flightNo = 123
   AND date = ’2018-10-2
   AND seatStatus = ’available’;
```

```
UPDATE Flights
SET    seatStatus = ’occupied’
WHERE  flightNo = 123
   AND date = ’2018-10-2’
   AND seatNo = ’14B’;
```

Transactions to the Rescue

- SQL allows us to state that a group of SQL statements must be executed so that no conflicts arise
  (we’ll see later how this is enforced)

```
START TRANSACTION;
```

Can often be omitted

```
SELECT seatNo
FROM   Flights
WHERE  flightNo = 123 AND date = ’ 2018-10-2’
   AND seatStatus = ’available’;
```

```
UPDATE Flights
SET    seatStatus = ’occupied’
WHERE  flightNo = 123 AND date = ’ 2018-10-2’
   AND seatNo = ’14B’;
```

```
COMMIT;
```

Before this, all changes to the database are tentative.

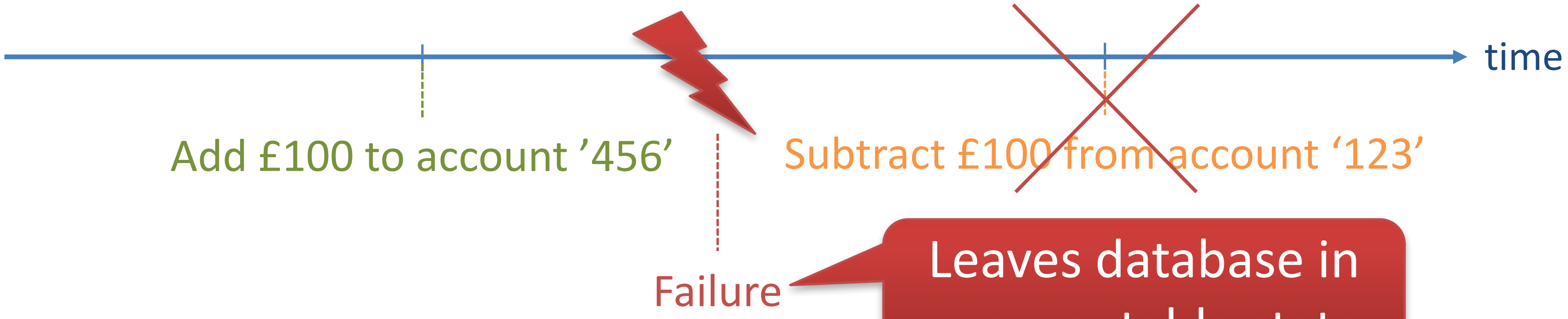
Transactions in SQL

- Transaction in SQL: a sequence of SQL statements
  - Special case: each individual SQL statement is a transaction
- By telling a DBMS that a sequence of SQL statements forms a transaction, it ensures *serialisable behaviour*
  - The transaction is executed as if was executed in isolation from all other transactions
  - *Equivalently:* … as if all transactions were executed one after the other
- It also ensures other properties…

Problem 2: Partial Execution

Accounts(accountNo, accountHolder, balance)

Goal: Transfer £100 from account ‘123’ to account ‘456’



Add £100 to account ’456’

Failure

~~Subtract £100 from account ‘123’~~

Leaves database in unacceptable state

```
UPDATE Accounts
SET    balance = balance - 100
WHERE  acountNo = 123;
```

```
UPDATE Accounts
SET    balance = balance + 100
WHERE  accountNo = 456;
```

Transactions to the Rescue

- SQL allows us to state that a transaction must be executed atomically (as a whole or not at all)

```
START TRANSACTION;
```

```
UPDATE Accounts
SET    balance = balance + 100
WHERE  accountNo = 456;
```

```
UPDATE Accounts
SET    balance = balance - 100
WHERE  acountNo = 123;
```

```
COMMIT;
```

Transactions in SQL – Summary

- Transaction in SQL: a sequence of SQL statements
  - Special case: each individual SQL statement is a transaction
  - General form:

```
START TRANSACTION;
```

Starts a transaction (can often be omitted)

SQL statements

```
COMMIT;
```

or

```
ROLLBACK;
```

Writes all changes to the database

Aborts the transaction

- Most DBMS ensure that transactions are executed
  - as if they were executed in series (“serialisability”)
  - as a whole or not at all (“atomicity”)

Let’s see how this works in detail...

Transaction: Overview

- **Transaction:**
  - Is an executing program, often comprising several queries (in **SQL**)
  - Can be submitted interactively or embedded within another programming language
  - **More details:** We consider SQL statements as transaction operations (**read** and **write** – “access operations”)
  - Executed **concurrently** at hundreds per second
  - Operations must leave the database in a valid or consistent state – enforced using **ACID** properties

Translating SQL into Low-Level Operations

STAFF(staffNo, familyName, firstName, jobTitle, gender, DOB, salary, department)

Two SQL Statements produces → **Three Transaction Operations**

```
SELECT salary
FROM Staff
WHERE staffNo = 1234;

UPDATE Staff
SET salary = salary*1.1
WHERE staffNo = 1234;
```

1. read(staffNo=1234, salary);
2. salary=salary*1.1;
3. write(staffNo=1234, salary);

Notes:

- Abstraction (at a high level)
- Read data item 'salary' from tuple with primary key 1234
- **Two database operations**
  - op1 (**read**) and op3 (**write**)
- **One non-database operation**
  - op2 (the calculation)

Transaction

- A logical unit of processing using access operations
  - Begin
  - End
  - read (retrieval – SELECT etc.)
  - write (insert, update, or delete)
  - + other non-database operations

Begin/end are are omitted
when the beginning/end
of a transaction are understood

Transactions Preserve Consistency

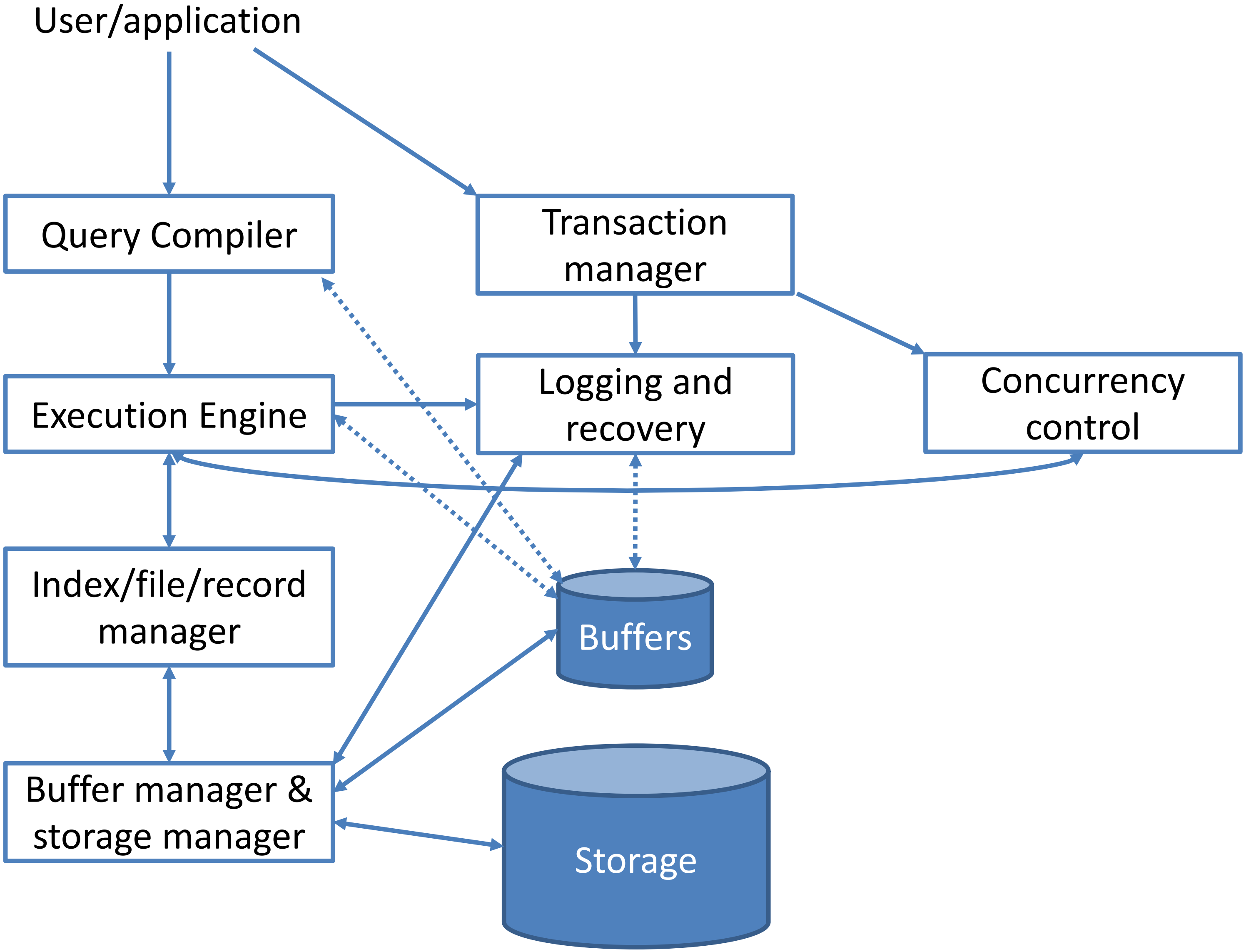
- Fundamental assumption:

Transactions always transform a *consistent* database state into another *consistent* database state.

- They produce one of two outcomes
  - Commit (i.e. Successful)
    - Execution was successful and database is left in a consistent state
  - Abort (i.e. Failed)
    - Execution was not successful and we need to restore the database to the state it was in before execution

Relational DBMS Components

(Simplified, from lecture 1)



Transaction - ACID Properties

- Transactions must maintain the correctness of the database, so we use **ACID** properties to validate transaction execution
- **A: Atomicity**
  - via Recovery Control (Logging and Recovery)
- **C: Consistency**
  - via Scheduler – Concurrency Control
- **I: Isolation**
  - via Scheduler – Concurrency Control
- **D: Durability** (or permanency)
  - via Recovery Control

A - Atomicity

- A transaction is an **atomic unit of processing**
  - An indivisible unit of execution
  - Executed in its entirety or not at all
- Deals with failure (“aborts”)
  - User aborts transaction (e.g., cancel button)
  - System aborts transaction (e.g., deadlock)
  - Transaction aborts itself (e.g., unexpected database state)
  - System crashes, network failure, etc.

A - Atomicity

- Abort - an error prevented full execution
  - We **UNDO** the work done up to the error point
    - System re-creates the database state as it was before the start of the aborted transaction
- Commit - no error, entire transaction executes
    - The system is updated correctly

C - Consistency

- A correct execution of the transaction must take the database from one consistent state to another
  - It should correctly transform the database state to reflect the effect of a real world event
  - Transactions may not violate integrity constraints

I - Isolation

- A transaction only makes its updates visible to other transactions after it has committed
  - The effect of **concurrently** executing a set of transactions is the same as if they had executed **serially** (“*serialisable”)*
  - When enforced strictly, this solves the **temporary update problem** and makes **cascading rollbacks** of transactions unnecessary

D - Durability

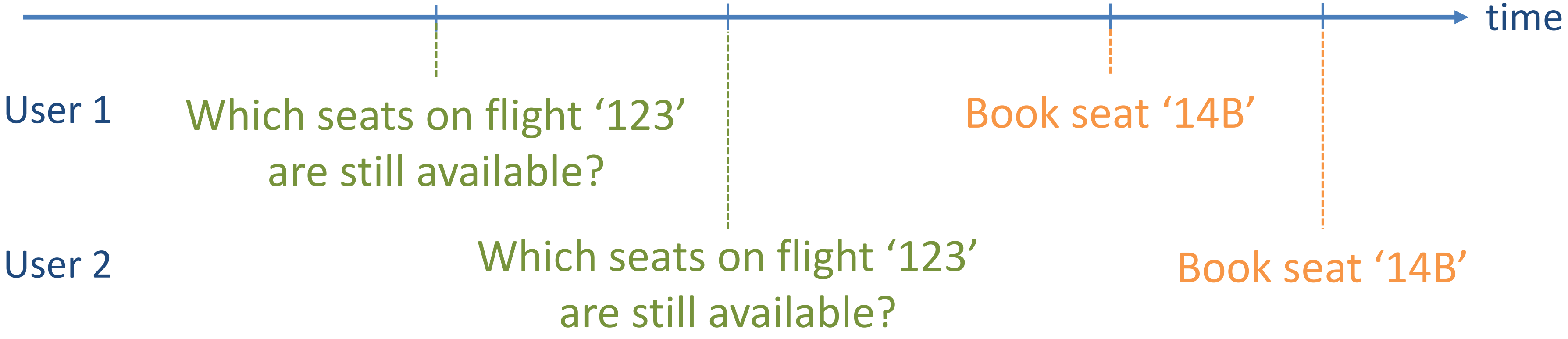
- Once a transaction commits and changes the database, these changes cannot be lost because of subsequent failure
  - The effect of a transaction on the database should not be lost after the commit point
  - We **REDO** the transaction if there are any problems after the update
  - Durability deals with things like media failure

Goal of Transactions

- DBMS’s are expected to be reliable and remain in a consistent state
- The components of a DBMS that ensure this are:
  - Concurrency Control: responsible for ‘C’ and ‘I’
  - Recovery Control: responsible for ‘A’ and ‘D’
- These help maintain the ‘ACID’ Properties

Example 1

Flights(flightNo, date, seatNo, seatStatus)

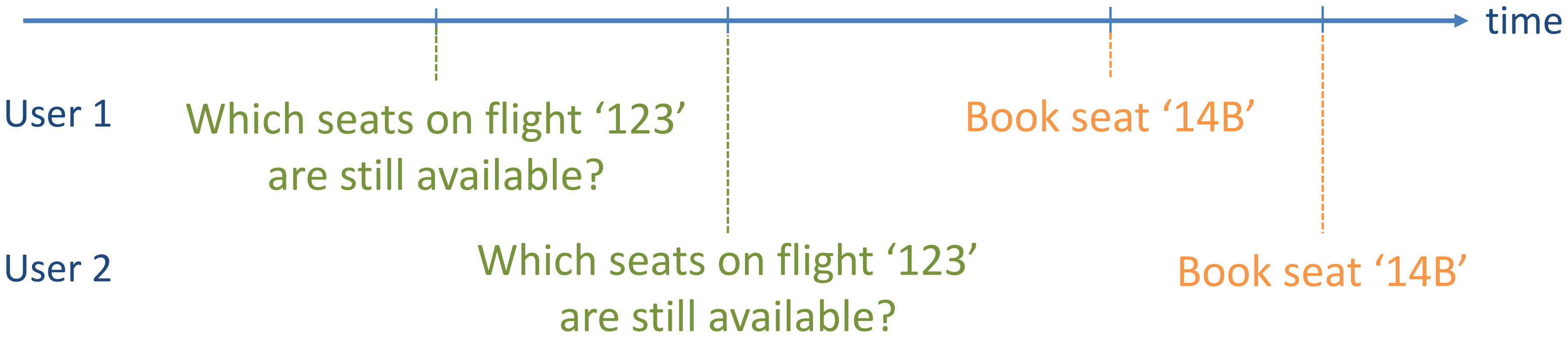


time
User 1
Which seats on flight ‘123’ are still available?
Book seat ‘14B’
User 2
Which seats on flight ‘123’ are still available?
Book seat ‘14B’

```
SELECT seatNo
FROM   Flights
WHERE  flightNo = 123
   AND date = ’2017-09-2
   AND seatStatus = ’available’;
```

```
UPDATE Flights
SET    seatStatus = ’occupied’
WHERE  flightNo = 123
   AND date = ’2017-09-28’
   AND seatNo = ’14B’;
```

Example 1



time

User 1

Which seats on flight '123' are still available?

Book seat '14B'

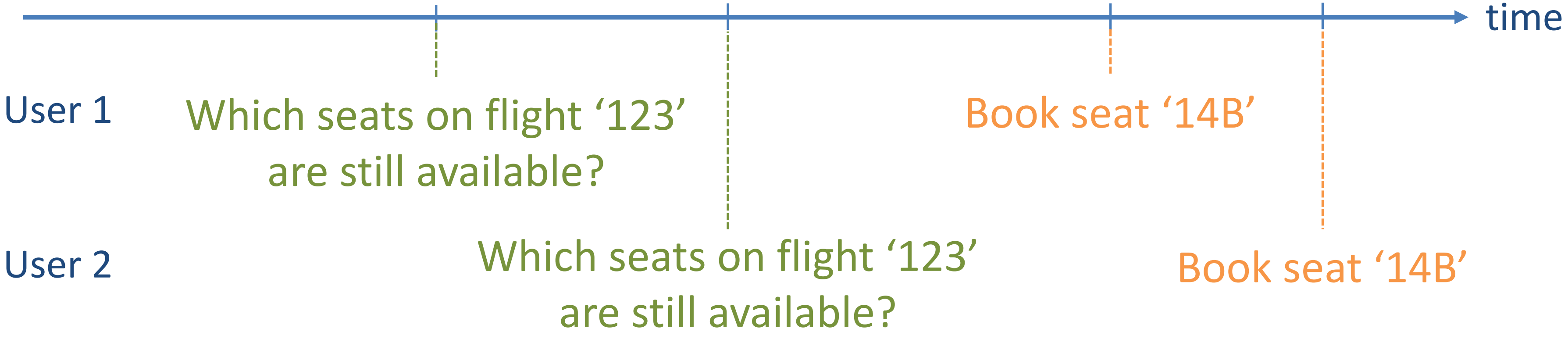
User 2

Which seats on flight '123' are still available?

Book seat '14B'

- Which of the ACID properties does this violate?

Example 1



time

User 1

Which seats on flight ‘123’ are still available?

Book seat ‘14B’

User 2

Which seats on flight ‘123’ are still available?

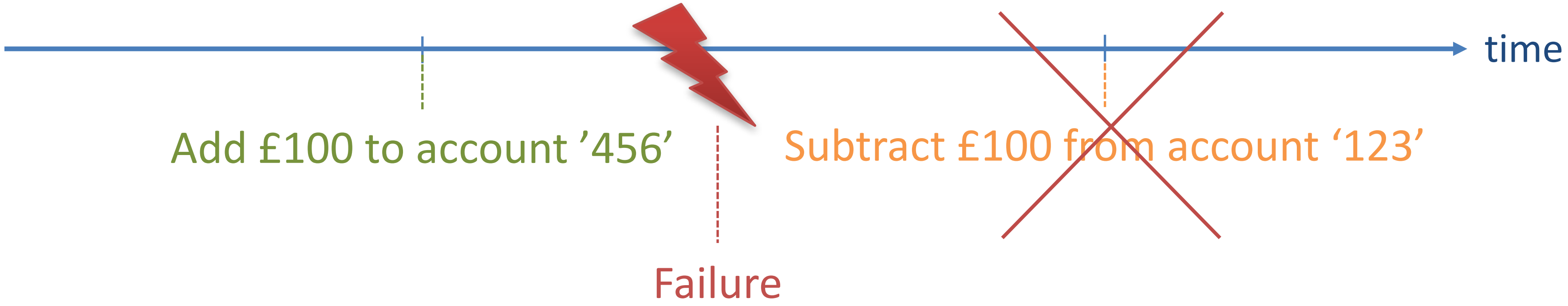
Book seat ‘14B’

- Which of the ACID properties does this violate?
  - *Consistency*
  - Note: *Isolation* is **not** violated

Example 2

Accounts(accountNo, accountHolder, balance)

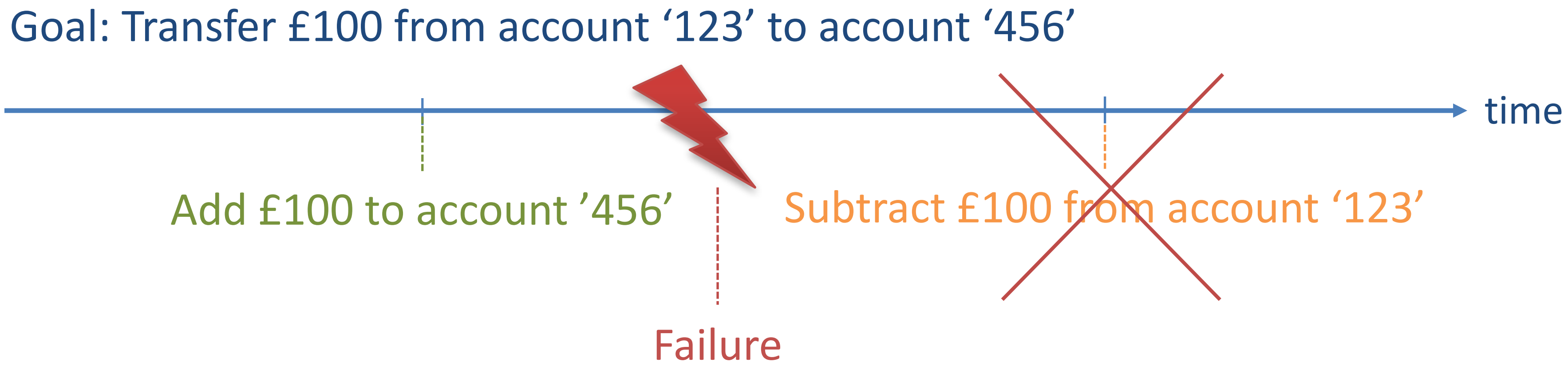
Goal: Transfer £100 from account ‘123’ to account ‘456’



```
UPDATE Accounts
SET    balance = balance - 100
WHERE  acountNo = 123;
```

```
UPDATE Accounts
SET    balance = balance + 100
WHERE  accountNo = 456;
```

Example 2



Goal: Transfer £100 from account ‘123’ to account ‘456’

time

Add £100 to account ’456’

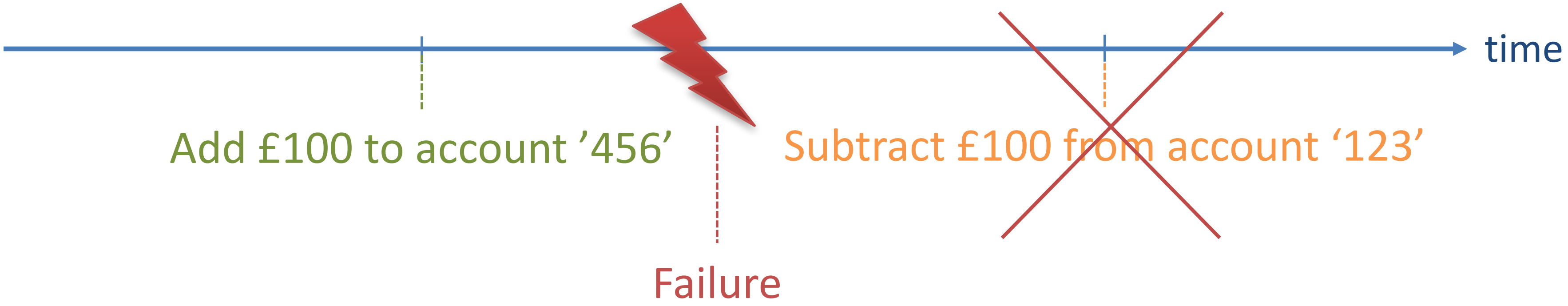
Subtract £100 from account ‘123’

Failure

- Which of the ACID properties does this violate?

Example 2

Goal: Transfer £100 from account ‘123’ to account ‘456’



time

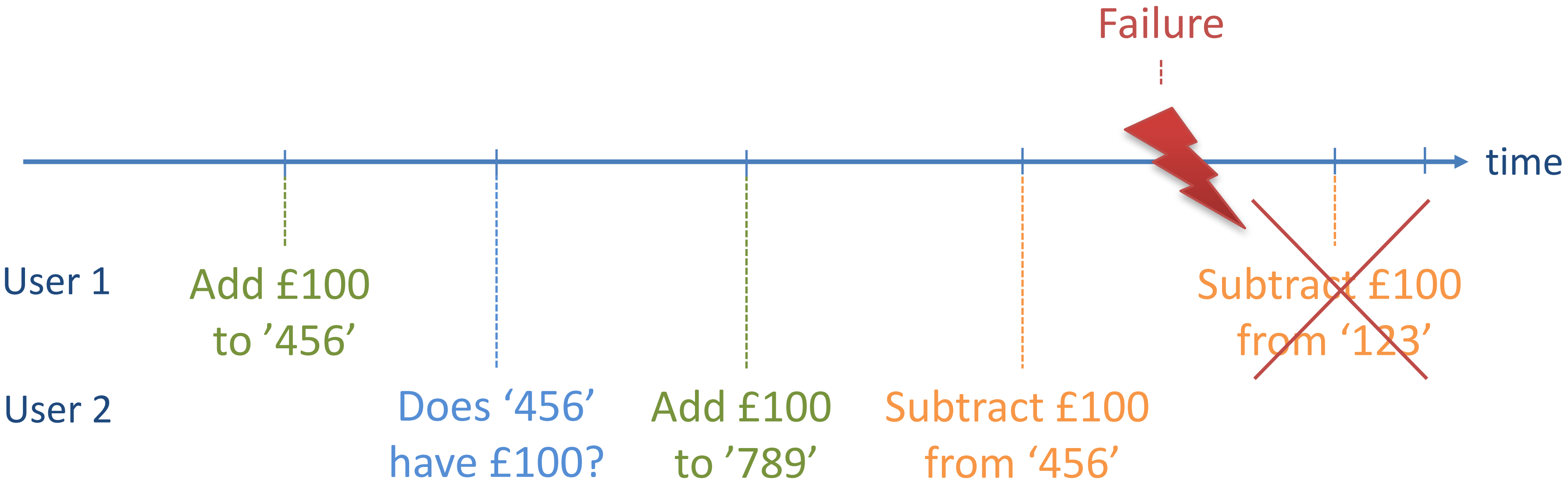
Add £100 to account ’456’

~~Subtract £100 from account ‘123’~~

Failure

- Which of the ACID properties does this violate?
  - *Atomicity*

Example 3



Failure

time

User 1

Add £100 to ’456’

~~Subtract £100 from ‘123’~~

User 2

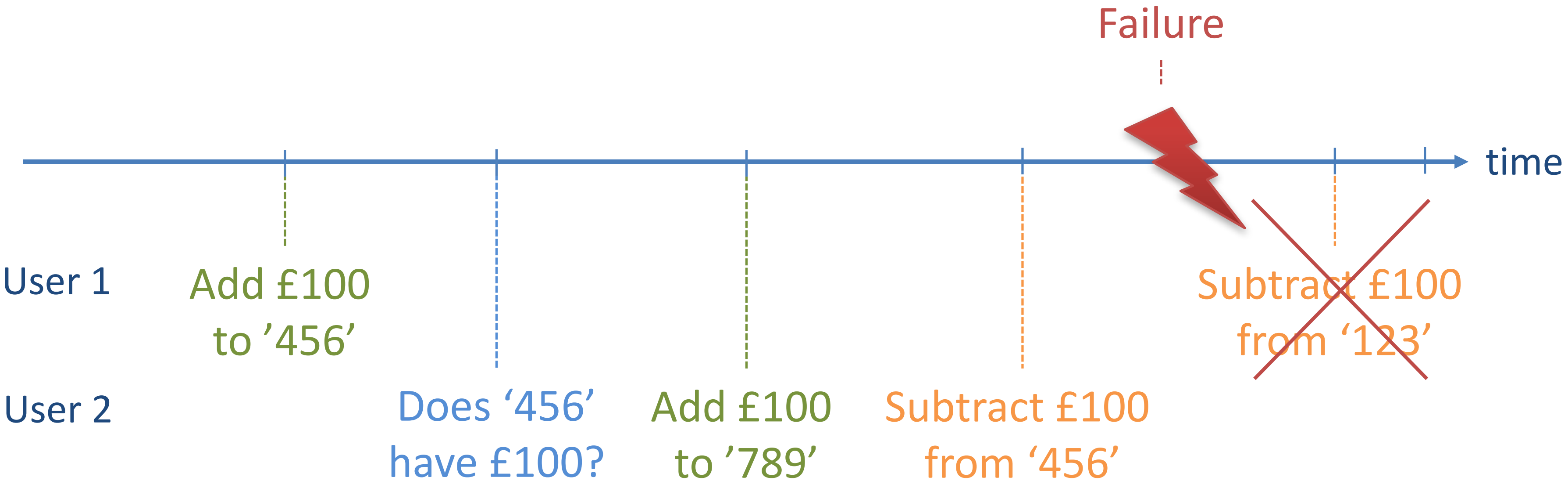
Does ‘456’ have £100?

Add £100 to ’789’

Subtract £100 from ‘456’

- Which of the ACID properties does this violate?

Example 3



- Which of the ACID properties does this violate?
  - *Atomicity*
  - *Isolation*

Can you think of any situation where *Durability* might be violated?

Summary

- Transactions are sequences of operations on a database (here: read and write operations)
- To avoid “damaging“ the database, a DBMS should enforce the ACID properties. In particular:
  - Transactions should be executed as a whole or not at all.
  - Transactions should be executed as if they were executed serially, one after the other.
- The ACID properties are enforced by:
  - Concurrency control: C and I
  - Recovery control: A and D