# Databases 2

1 # Transactional Systems

In this lecture

- Introduction to the concept of *transaction*
- *Properties* of a transaction
- Modules of the DBMS responsible of guaranteeing such properties

1 Transactional Systems

Definition of Transaction

- An elementary, atomic unit of work performed by an application

- Each transaction is encapsulated within two commands:
  - `begin transaction` (bot)
  - `end transaction` (eot)

- Within a transaction, **one** of the commands below is executed (exactly **once**):
  - `commit-work` (commit)
  - `rollback-work` (abort)

- **Transactional System** (OLTP): a system that supports the execution of transactions on behalf of concurrent applications

Difference between Application and Transaction

Application

begin T$_1$

Transaction T$_1$

end T$_1$

Actions

begin T$_2$

Transaction T$_2$

end T$_2$
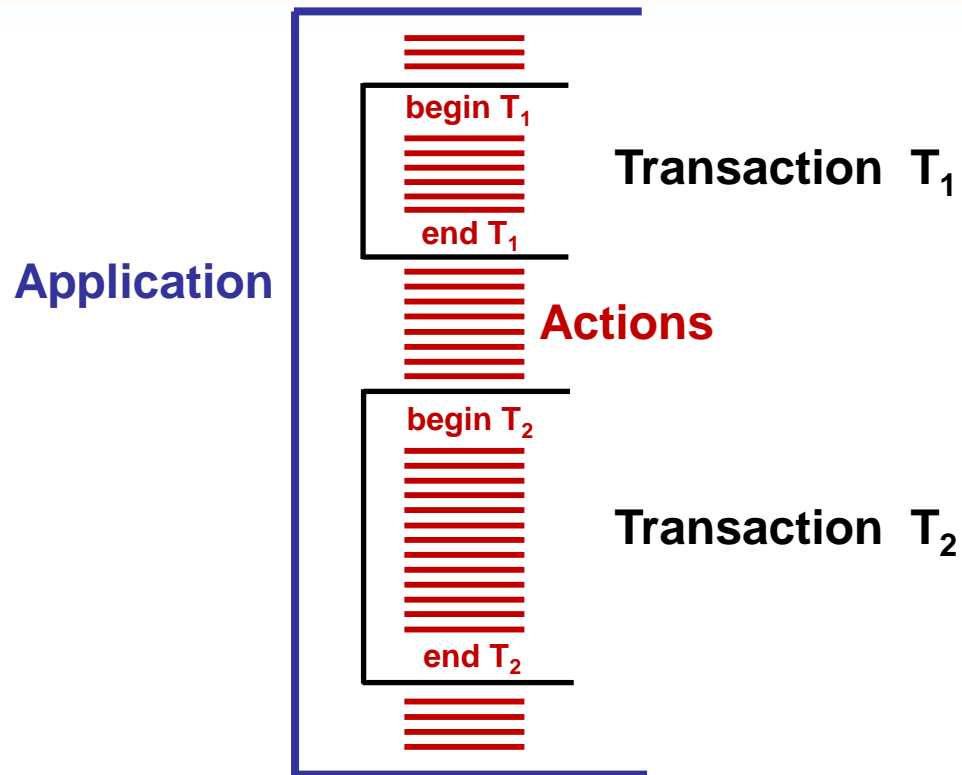
Transactions: an example

```
begin transaction;
update Account
    set Balance = Balance + 10 where AccNum = 12202;
update Account
    set Balance = Balance - 10 where AccNum = 42177;
commit-work;
end transaction;
```

1 Transactional Systems

Transactions: an example with alternatives

```
begin transaction;
update Account
    set Balance = Balance + 10 where AccNum = 12202;
update Account
    set Balance = Balance – 10 where AccNum = 42177;
select Balance into A from Account
    where AccNum = 42177;
if ( A >= 0 ) then commit-work
                else rollback-work;
end transaction;
```
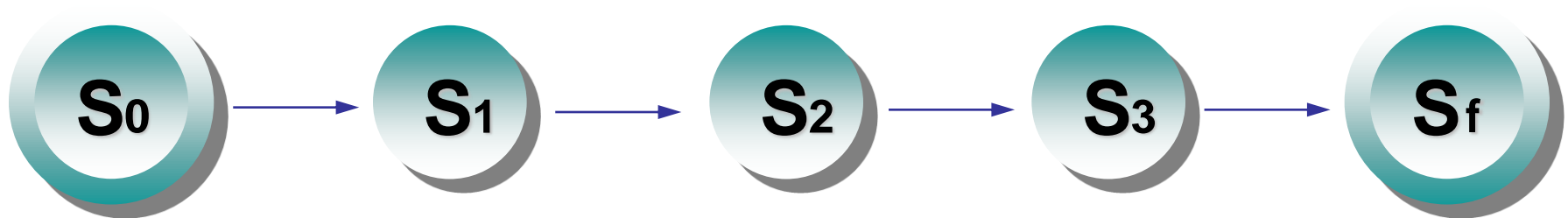
1 Transactional Systems

Well-formed Transactions

begin `transaction`

    code for data manipulation (reads and writes on DB)

commit`-work` / **rollback**`-work`

    possibly more code, but <u>no data manipulation</u>

end `transaction`

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_f$$

ACID Properties of Transactions

- A transaction is a unit of work enjoying the following properties:

  - Atomicity
  - Consistency
  - Isolation
  - Durability

1 Transactional Systems

Atomicity

- A transaction is an atomic unit of execution, it can not be divided
  - Either all the operations in the transaction are executed or none is executed

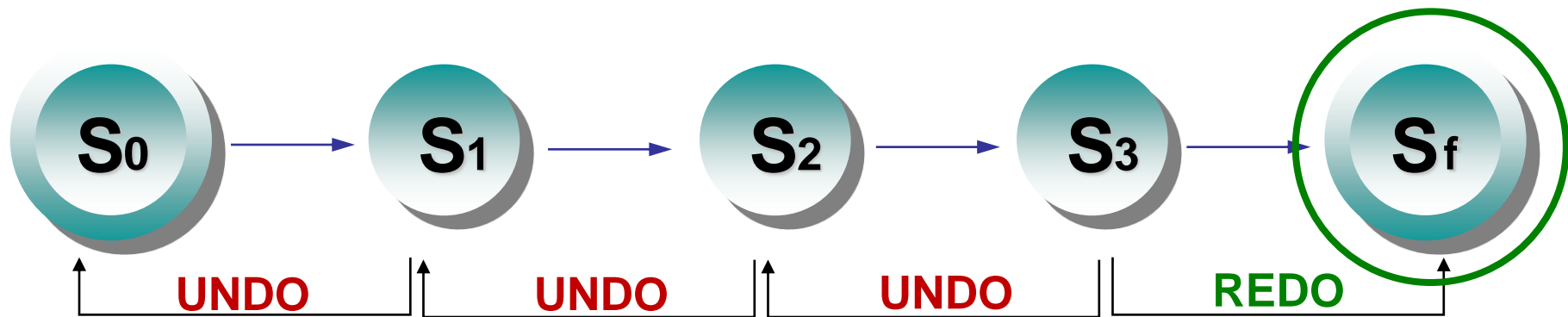S_0 → S_1 → S_2 → S_3 → S_f

1 Transactional Systems
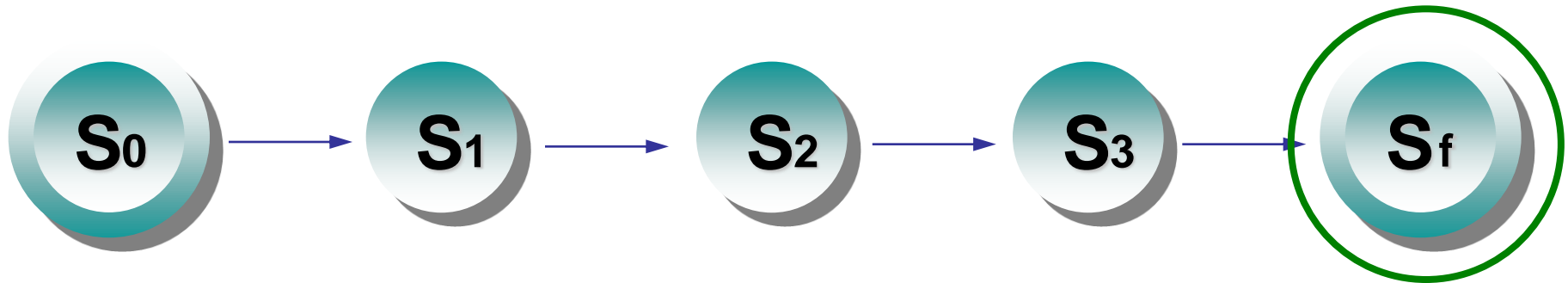
Atomicity

- In the case of the bank transfer, the execution of a single update statement would be disastrous

- The instant in which COMMIT is executed marks the atomic and indivisible instant in which the transaction ends successfully:

  - An error before should cause the rollback of the work performed

  - An error afterwards should not alter the effect of the transaction

1 Transactional Systems

Atomicity

- The ROLLBACK of the work performed can be caused
  - By a ROLLBACK statement
  - By the DBMS, for example for the violation of integrity constraints or for concurrency management

- In case of a rollback, the work performed must be undone, bringing the database to the state it had before the start of the transaction

1 **Transactional Systems**

Atomicity

- Possible behaviours:
    1. commit-work: **SUCCESS**
    2. rollback-work or fault before commit-work: **UNDO**
    3. fault after commit-work: **REDO**

$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_f$

UNDO   **UNDO**   **UNDO**   **REDO**

1 **Transactional Systems**

Consistency

- A transaction must satisfy the DB integrity constraints

- Consequence:
  - **if** the initial state S_0 is consistent
  - **then** the final state S_f is also consistent

This is not necessarily true for the intermediate states S_i

1 Transactional Systems

Consistency - EXAMPLE

- A table can not contain two rows with the same value for a column declared as a primary key
- The sum of the balances of every couple of accounts must remain the same
  - If Balance(a)+Balance(b)=Costant before the transaction then Balance(a)+Balance(b)=Costant after the transaction
  - This type of constraint can be temporarily violated during the execution of the transaction, but must be satisfied at the end

Isolation

- The execution of a transaction must be independent from the concurrent execution of other transactions
- In particular, the concurrent execution of a number of transaction must produce the same result as the execution of the same transactions in a sequence
- E.g. the concurrent exection of T1 and T2 must produce the same results that can be obtained by executing one of these sequences
  - T1,T2
  - T2,T1

Durability

- The effect of a transaction that has successfully committed will last "**forever**"
  - Independently of any system fault
  - Sounds obvious… but every DBMS manipulates data in *main memory*

1 **Transactional Systems**

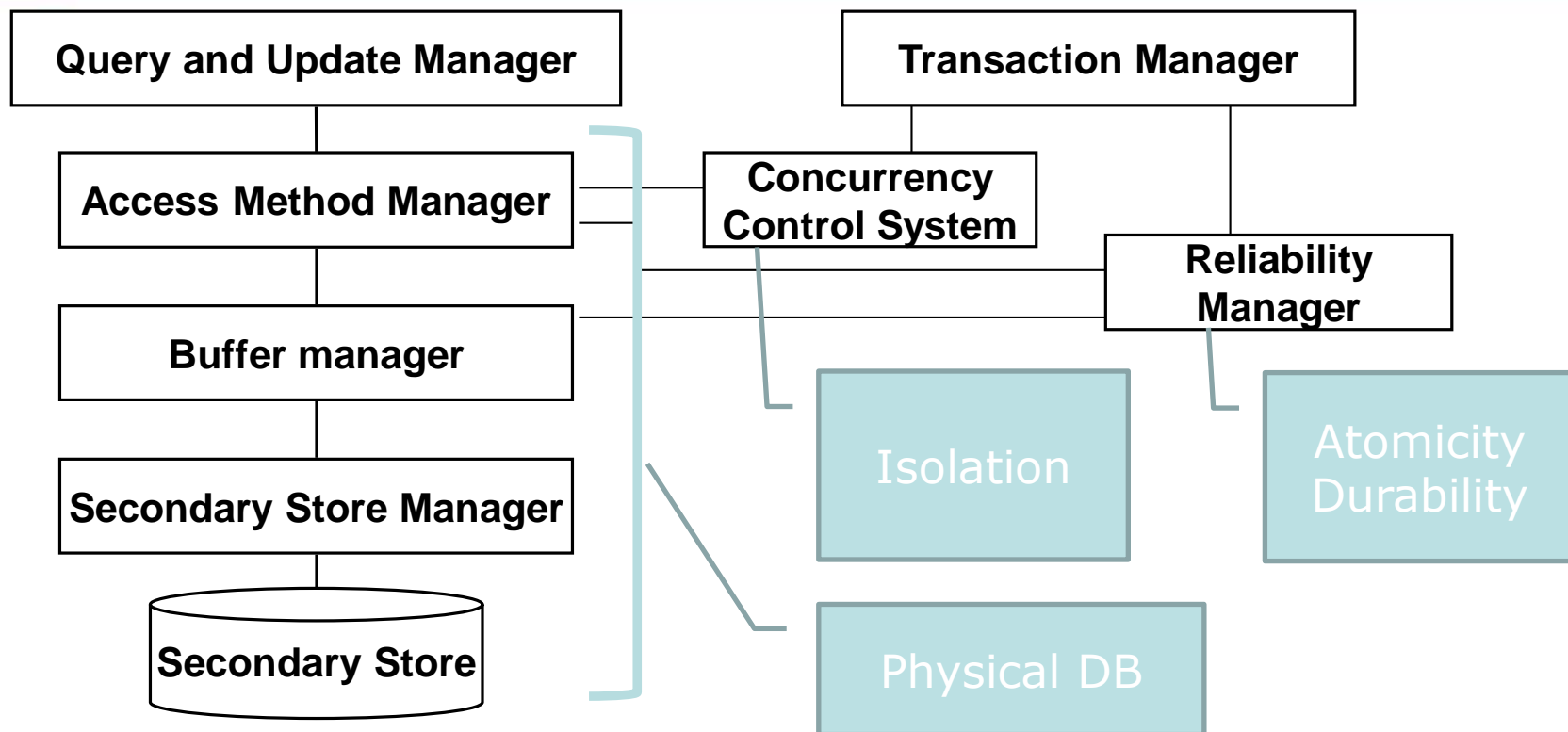## Transaction Properties and related mechanisms

- **A** tomicity
  - Abort-rollback-restart, Commit protocols
- **C** onsistency
  - Integrity checking of the DBMS
- **I** solation
  - Concurrency control
- **D** urability
  - Recovery management

**ACID
properties**

Transactions and DBMS modules

- Atomicity and Durability
  - Reliability Manager
- Isolation
  - Concurrency Control System
- Consistency
  - Integrity Control System at query execution time (with the support of the DDL compiler)

Logical Architecture of a DBMS

Query and Update Manager	**Transaction Manager**
Access Method Manager	**Concurrency Control System**
Buffer manager	**Reliability Manager**
Secondary Store Manager	Isolation
Secondary Store	Atomicity Durability
	Physical DB

Next Topics...

- Concurrency Control (Isolation)
  - Theory
  - DBMS adopted method

- Reliability Control (Atomicity and Durability)
  - Commit protocols and 2PC
  - Logging and recovery on a single DBMS