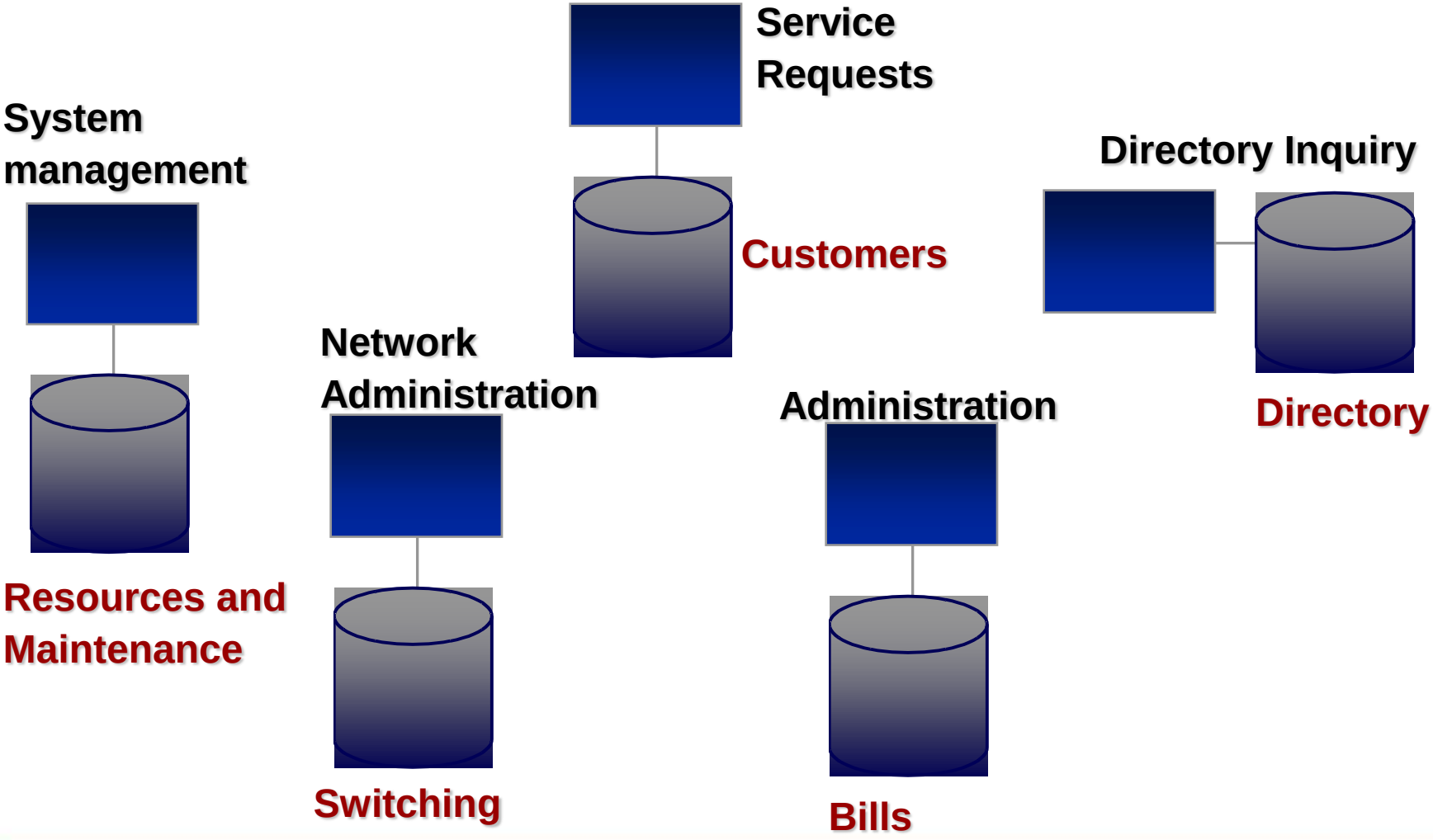
Advanced Databases

1 Transactional Systems

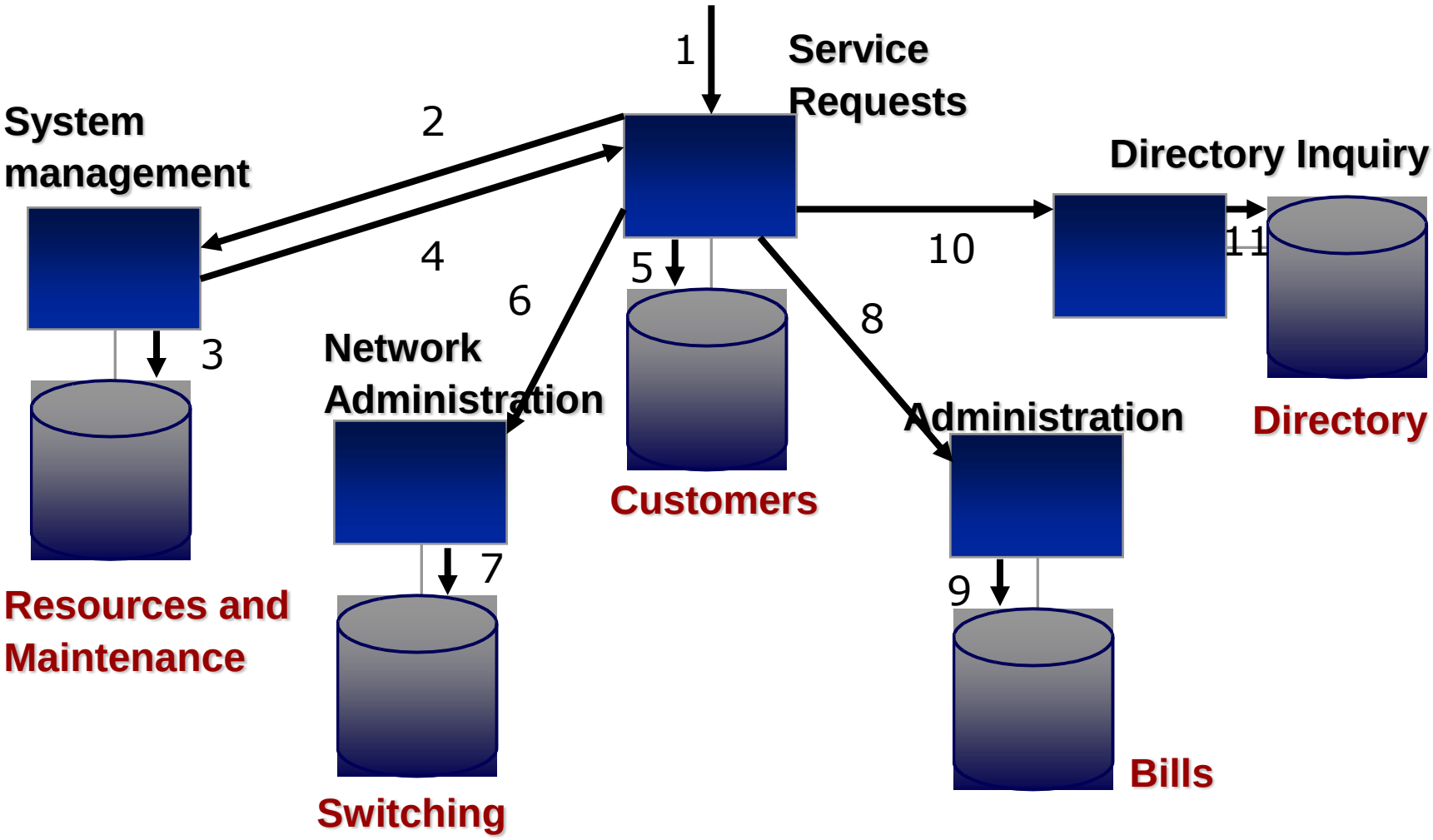
Transactional Systems

Example of Information System



Transactional Systems

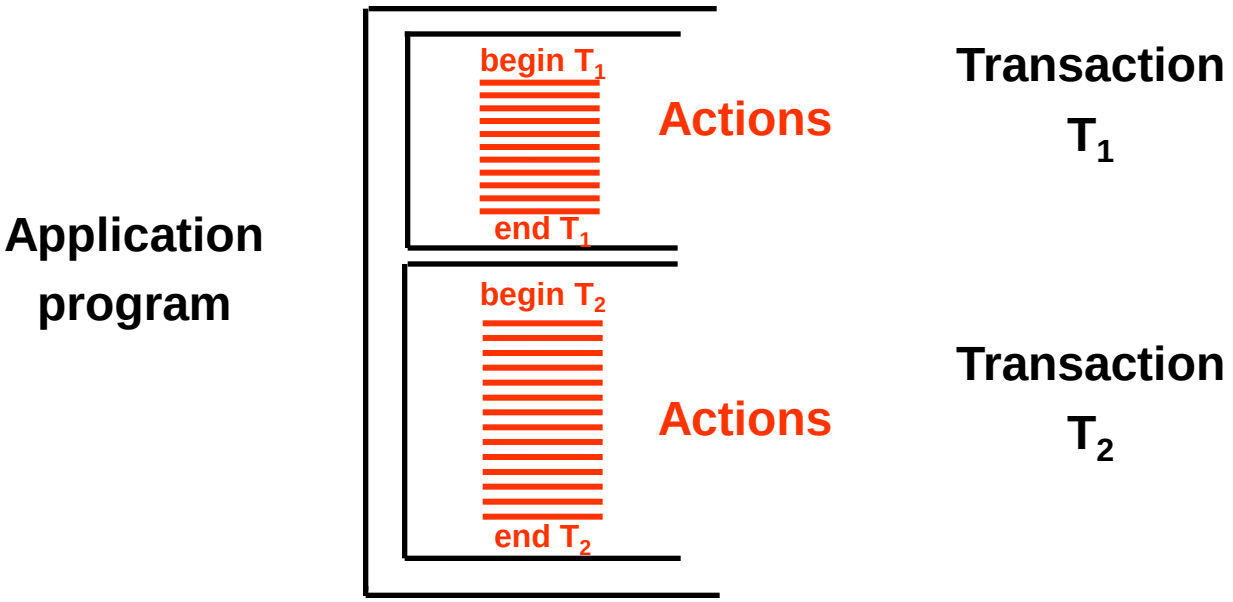
Example of Transaction



Definition of Transaction

- An elementary 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 capable of providing the definition and execution of transactions on behalf of multiple, concurrent applications

Difference between Application and Transaction



Transaction: Example

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

Transaction: Example with Alternative

```
start 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;
```

Transactions in JDBC

- Transaction mode is chosen via a method defined in the **`Connection`** interface

  `setAutoCommit(boolean autoCommit)`

- **`con.setAutoCommit(true)`**
  - (Default) "autocommit": every single operation is a transaction
- **`con.setAutoCommit(false)`**
  - Transactions are handled in the program
    - **`con.commit()`**
    - **`con.rollback()`**
  - There is no **`start transaction`**

Well-formed Transactions

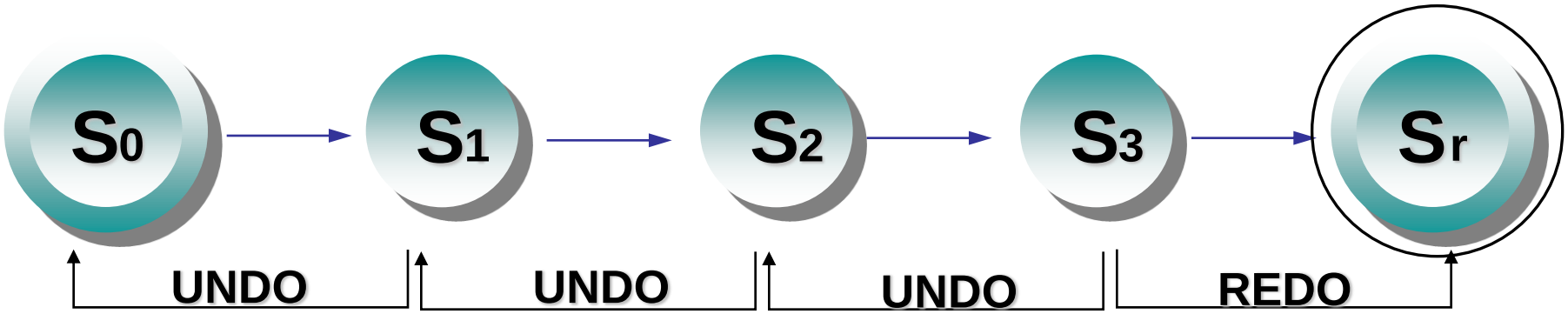
- **`begin transaction`**
- code for data manipulation (reads and writes)
- **`commit work – rollback work`**
- no data manipulation
- **`end transaction`**

ACID Properties of Transactions

- A transaction is a unit of work enjoying the following properties:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Atomicity

- A transaction is an atomic transformation from the initial state to the final state
- Possible behaviors:
  1. Commit work: SUCCESS
  2. Rollback work or error prior to commit: UNDO
  3. Fault after commit: REDO

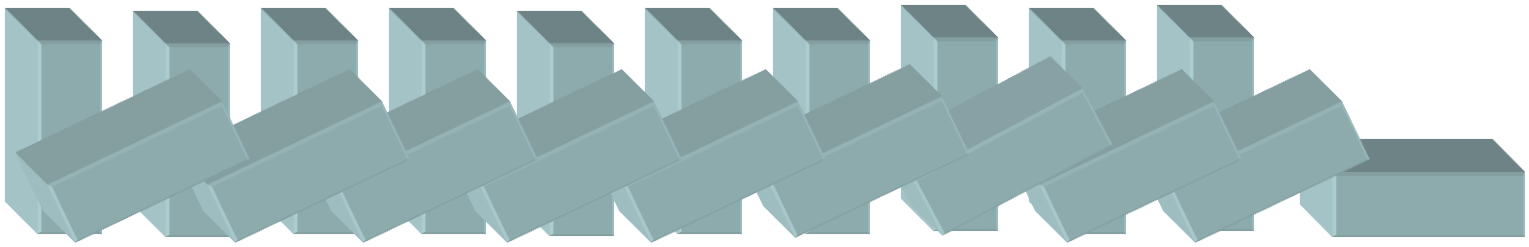


Consistency

- The transaction satisfies the integrity constraints
- Consequence:
  - If the initial state is consistent
  - Then the final state is also consistent

Isolation

- A transaction is not affected by the behavior of other, concurrent transactions
- Consequence:
  - Its intermediate states are not exposed
  - The “domino effect” is avoided



Durability

- The effect of a transaction that has successfully committed will last "forever"
  - Independently of any system fault

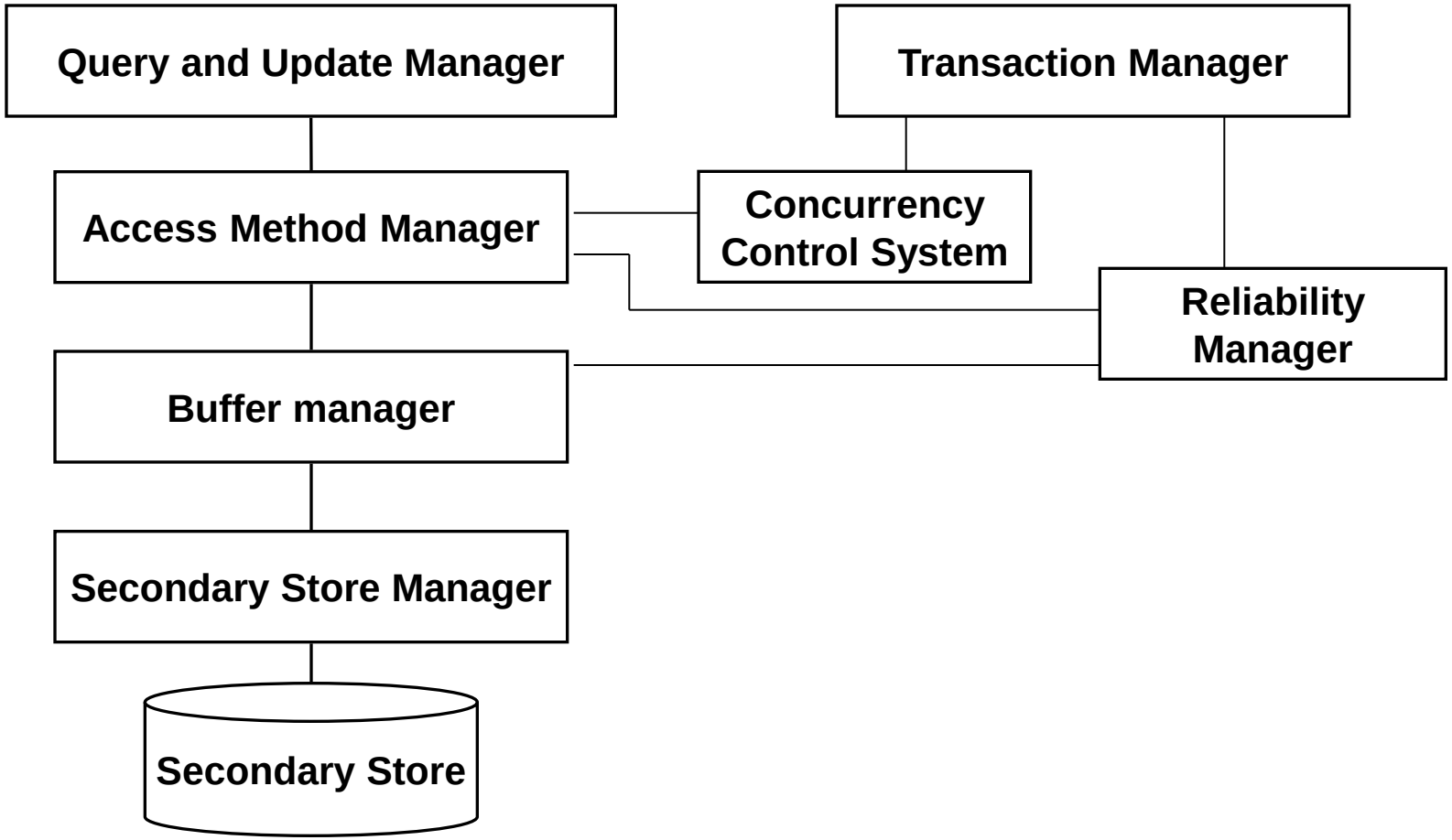
Transaction Properties and Mechanisms

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

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 Compilers)

Logical Architecture of a DBMS



Next Topics…

- Concurrency Control
  - Theory
  - 2PL method
- Reliability Control
  - Logging and recovery on a single DBMS
  - Commit protocols and 2PC
- Database Architectures
  - Distribution
  - Parallelism
  - Replication
  - Warehousing