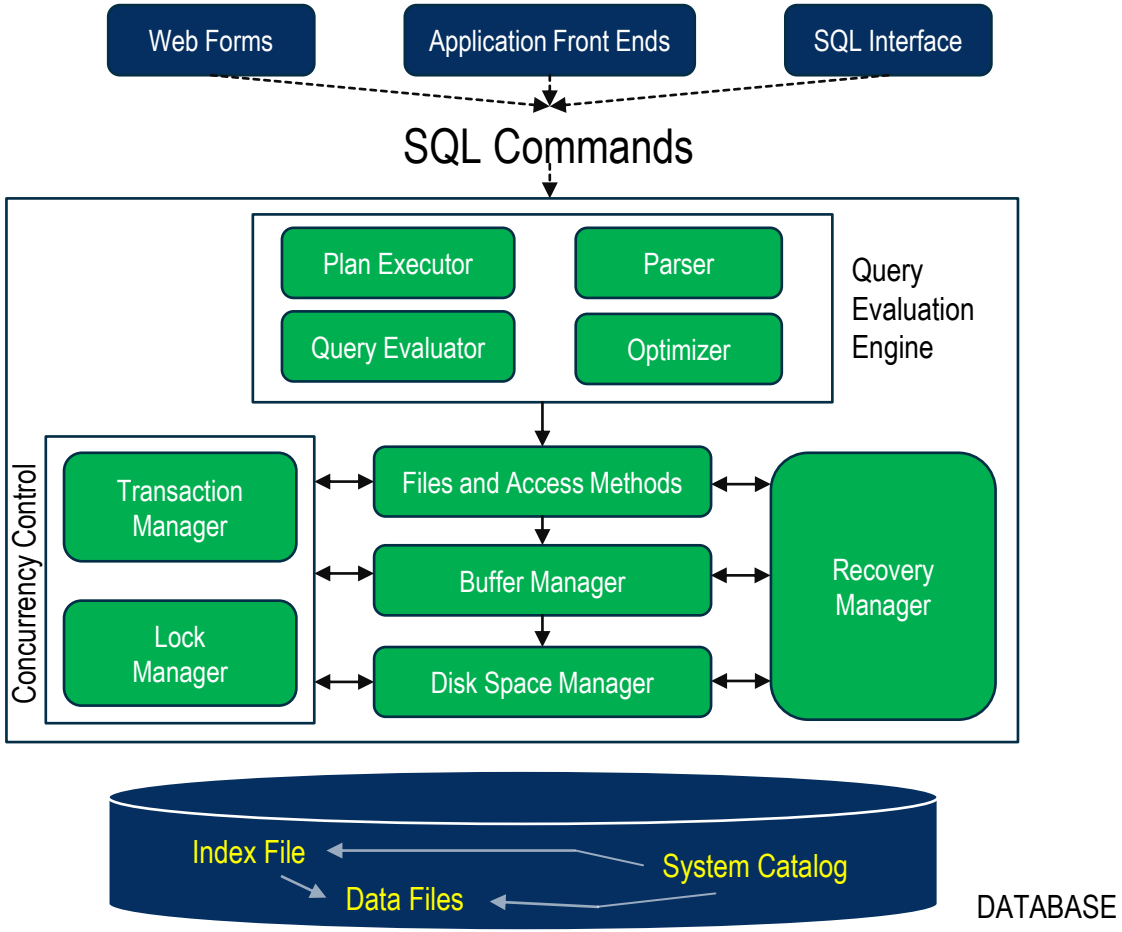


INTRODUCTION TO DATABASES USING ORACLE

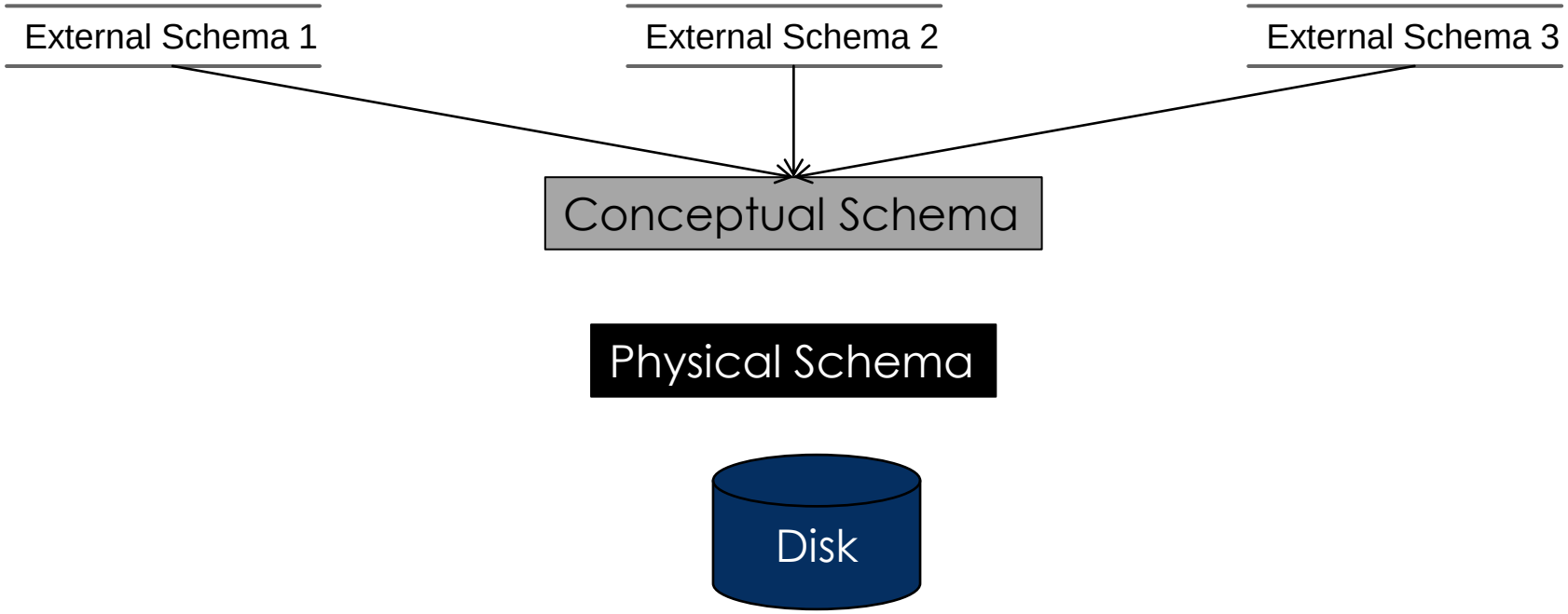
420-983-VA

STRUCTURE OF DBMS

LEVELS OF ABSTRACTION IN A DBMS

The database description consists of a schema at each of these three levels of abstraction: the conceptual, physical, and external.

A **D**ata **D**efinition **L**anguage (DDL) is used to define the external and conceptual schemas. Information about the conceptual, external, and physical schemas is stored in the system catalogs.



CONCEPTUAL SCHEMA

The conceptual schema (sometimes called the logical schema) describes the stored data in terms of the data model of the DBMS. In a relational DBMS, the conceptual schema describes all relations that are stored in the database.

The choice of relations, and the choice of fields for each relation, is not always obvious, and the process of arriving at a good conceptual schema is called conceptual database design.

CONCEPTUAL SCHEMA

Example: Vanier College database.

```
Students (sid: string, name: string, login: string, age: integer, gpa: real)

Faculty (fid: string, fname: string, sal: real)

Courses (cid: string, cname: string, credits: integer)

Rooms (rno: integer, address: string, capacity: integer)

Enrolled (sid: string, cid: string, grade: string)

Teaches (fid: string, cid: string)

Meets_In ( cid: string, rno: integer, time: string)
```

PHYSICAL SCHEMA

The physical schema specifies additional storage details. Essentially, the physical schema summarizes how the relations described in the conceptual schema are actually stored on secondary storage devices such as disks and tapes.

Example:

- Store all relations as unsorted files of records. (A file in a DBMS is either a collection of records or a collection of pages, rather than a string of characters as in an operating system.)

- Create indexes on the first column of the Students, Faculty, and Courses relations and the capacity column of Rooms.

Decisions about the physical schema are based on an understanding of how the data is typically accessed. The process of arriving at a good physical schema is called physical database design.

EXTERNAL SCHEMA

External schemas, which usually are also in terms of the data model of the DBMS, allow data access to be customized (and authorized) at the level of individual users or groups of users. Any given database has exactly one conceptual schema and one physical schema because it has just one set of stored relations, but it may have several external schemas, each tailored to a particular group of users. Each external schema consists of a collection of one or more views and relations from the conceptual schema. A view is conceptually a relation, but the records in a view are not stored in the DBMS.

EXTERNAL SCHEMA

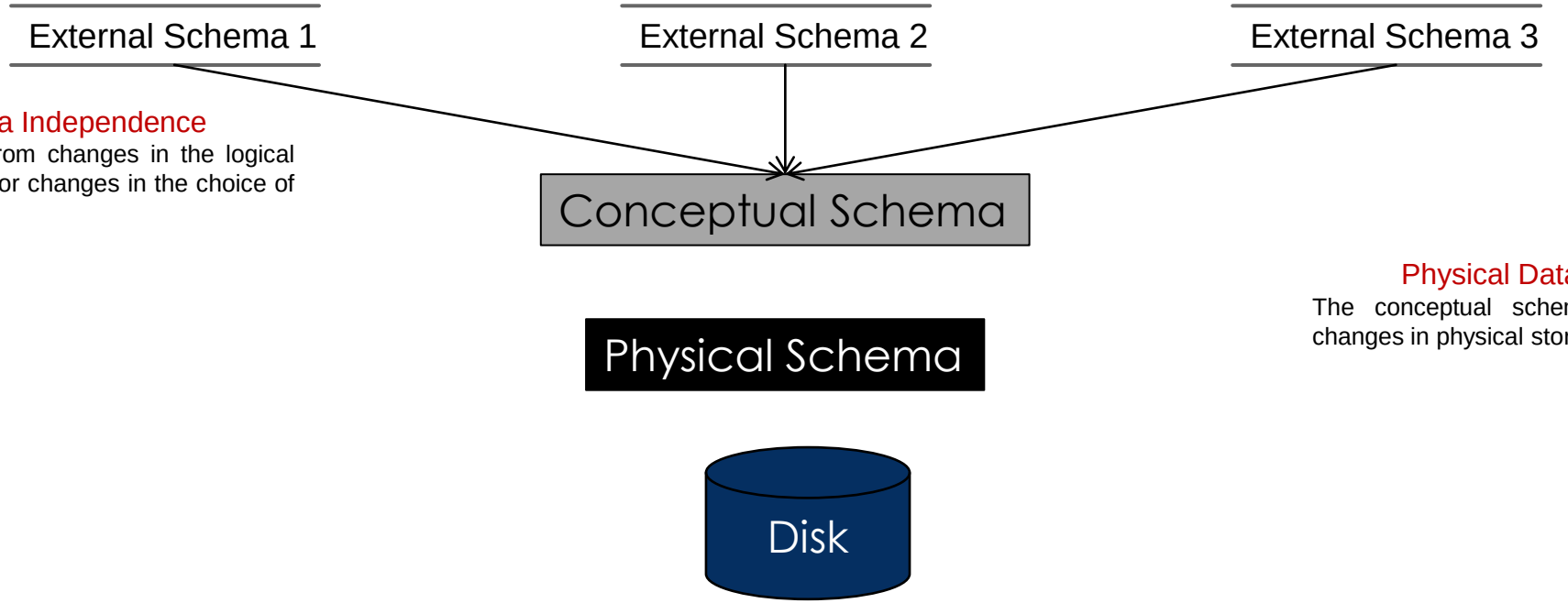
Example: We might want to allow students to find out the names of faculty members teaching courses as well as course enrollments. This can be done by defining the following view

```
Courseinfo (rid: string, fname: string, enTollment: integer)
```

A user can treat a view just like a relation and ask questions about the records in the view. Even though the records in the view are not stored explicitly, they are computed as needed. We did not include Courseinfo in the conceptual schema because we can compute Courseinfo from the relations in the conceptual schema, and to store it in addition would be redundant.

DATA INDEPENDENCE

Application programs are insulated from changes in the way the data is structured and stored. Data independence is achieved through use of the three levels of data abstraction; in particular, the conceptual schema and the external schema provide distinct benefits in this area.



Logical Data Independence

Users are shielded from changes in the logical structure of the data, or changes in the choice of relations to be stored.

Physical Data Independence

The conceptual schema insulates users from changes in physical storage details..

QUERYING A DBMS

The ease with which information can be obtained from a database often determines its value to a user.

Here are some questions users may ask:

1. What is the name of the student with student ID 1234567?
2. What is the average salary of professors who teach course CS5647?
3. How many students are enrolled in CS5647?
4. What fraction of students in CS564 received a grade better than B7?
5. Is any student with a CPA less than 3.0 enrolled in CS5647?

Such questions involving the data stored in a DBMS are called queries.

QUERYING A DBMS

A DBMS provides a specialized language, called the query language, in which queries can be posed. A very attractive feature of the relational model is that it supports powerful query languages. A DBMS takes great care to evaluate queries as efficiently as possible.

A DBMS enables users to create, modify, and query data through a **D**ata **M**anipulation **L**anguage (DML). Thus, the query language is only one part of the DML, which also provides constructs to insert, delete, and modify data.

TRANSACTION MANAGEMENT

When several users access (and possibly modify) a database concurrently, the DBMS must order their requests carefully to avoid conflicts.

For example: when one travel agent looks up Flight 100 on some given day and finds an empty seat, another travel agent may simultaneously be making a reservation for that seat, thereby making the information seen by the first agent obsolete.

TRANSACTION MANAGEMENT

DBMS must protect users from the effects of system failures by ensuring that all data (and the status of active applications) is restored to a consistent state when the system is restarted after a crash.

For example: if a travel agent asks for a reservation to be made, and the DBMS responds saying that the reservation has been made, the reservation should not be lost if the system crashes.

TRANSACTION MANAGEMENT

A transaction is anyone execution of a user program in a DBMS. This is the basic unit of change as seen by the DBMS: Partial transactions are not allowed, and the effect of a group of transactions is equivalent to some serial execution of all transactions.

CONCURRENT EXECUTION OF TRANSACTIONS

An important task of a DBMS is to schedule concurrent accesses to data so that each user can safely ignore the fact that others are accessing the data concurrently. A DBMS allows users to think of their programs as if they were executing in isolation, one after the other in some order chosen by the DBMS.

CONCURRENT EXECUTION OF TRANSACTIONS

For example: if a program that debits 25$ from an account is submitted to the DBMS at the same time as another program that debits 50$ from the same account, either of these programs could be run first by the DBMS, but their steps will not be interleaved in such a way that they interfere with each other.

John's Bank Account

100$

What can go wrong?

Program 1
#spend 25$

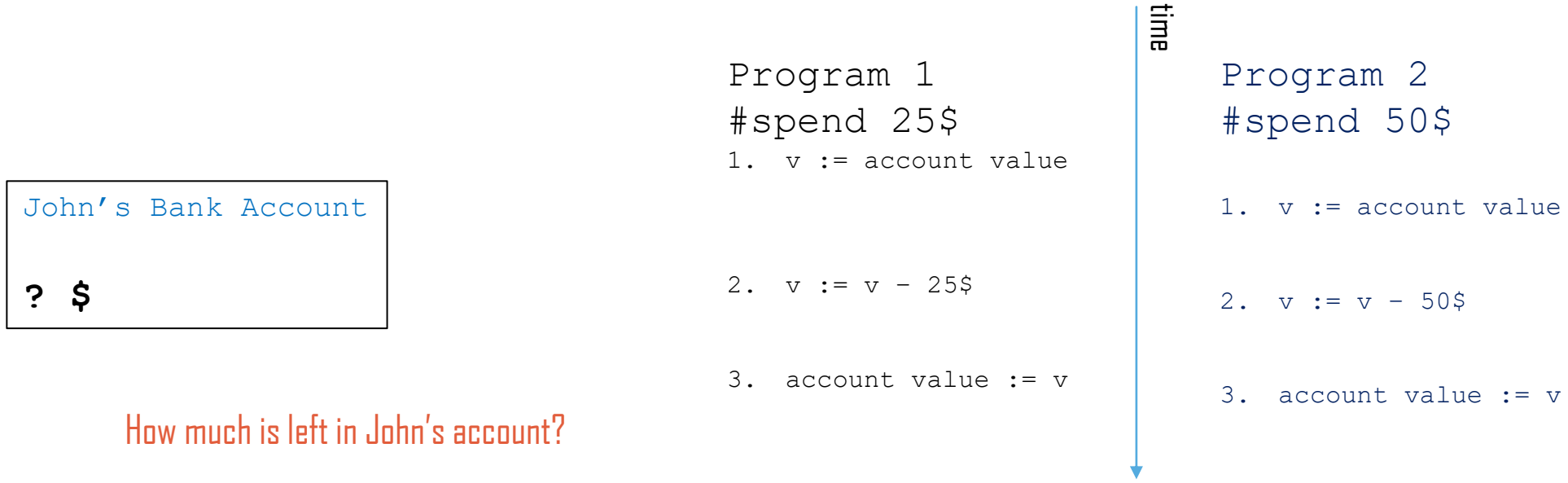
```
1.  v := account value
2.  v := v - 25$
3.  account value := v
```

Program 2
#spend 50$

```
1.  v := account value
2.  v := v - 50$
3.  account value := v
```

CONCURRENT EXECUTION OF TRANSACTIONS

For example: if a program that debits 25$ from an account is submitted to the DBMS at the same time as another program that debits 50$ from the same account, either of these programs could be run first by the DBMS, but their steps will not be interleaved in such a way that they interfere with each other.



CONCURRENT EXECUTION OF TRANSACTIONS

How DBMS ensures serial like execution?

A locking protocol is a set of rules to be followed by each transaction (and enforced by the DBMS) to ensure that, even though actions of several transactions might be interleaved, the net effect is identical to executing all transactions in some serial order. A lock is a mechanism used to control access to database objects. Two kinds of locks are commonly supported by a DBMS: shared locks on an object can be held by two different transactions at the same time, but an exclusive lock on an object ensures that no other transactions hold any lock on this object.

INCOMPLETE TRANSACTIONS AND SYSTEM CRASHES

Transactions can be interrupted before running to completion for a variety of reasons. A DBMS must ensure that the changes made by such incomplete transactions are removed from the database.

John's Bank Account A

200$

John's Bank Account B

100$

```
Program 1
#Transfer 100$ from A to B

1.  account A value := account A value - 100$
```

```
2.  account B value := account B value + 100$
```

What is John's balance?

INCOMPLETE TRANSACTIONS AND SYSTEM CRASHES

Transactions can be interrupted before running to completion for a variety of reasons. A DBMS must ensure that the changes made by such incomplete transactions are removed from the database.

John's Bank Account A

100$

John's Bank Account B

100$

```
Program 1
#Transfer 100$ from A to B

1.  account A value := account A value - 100$
```

John's A account needs to be restored

```
2.  account B value := account B value + 100$
```

INCOMPLETE TRANSACTIONS AND SYSTEM CRASHES

John's A account needs to be restored

How does a DBMS achieve this?

It maintains a log of all writes to the database. A crucial property of the log is that each write action must be recorded in the log (on disk) before the corresponding change is reflected in the database itself--otherwise, if

the system crashes just after making the change in the database but before the change is recorded in the log, the DBMS would be unable to detect and undo this change. This property is called **W**rite-**A**head **L**og (WAL).

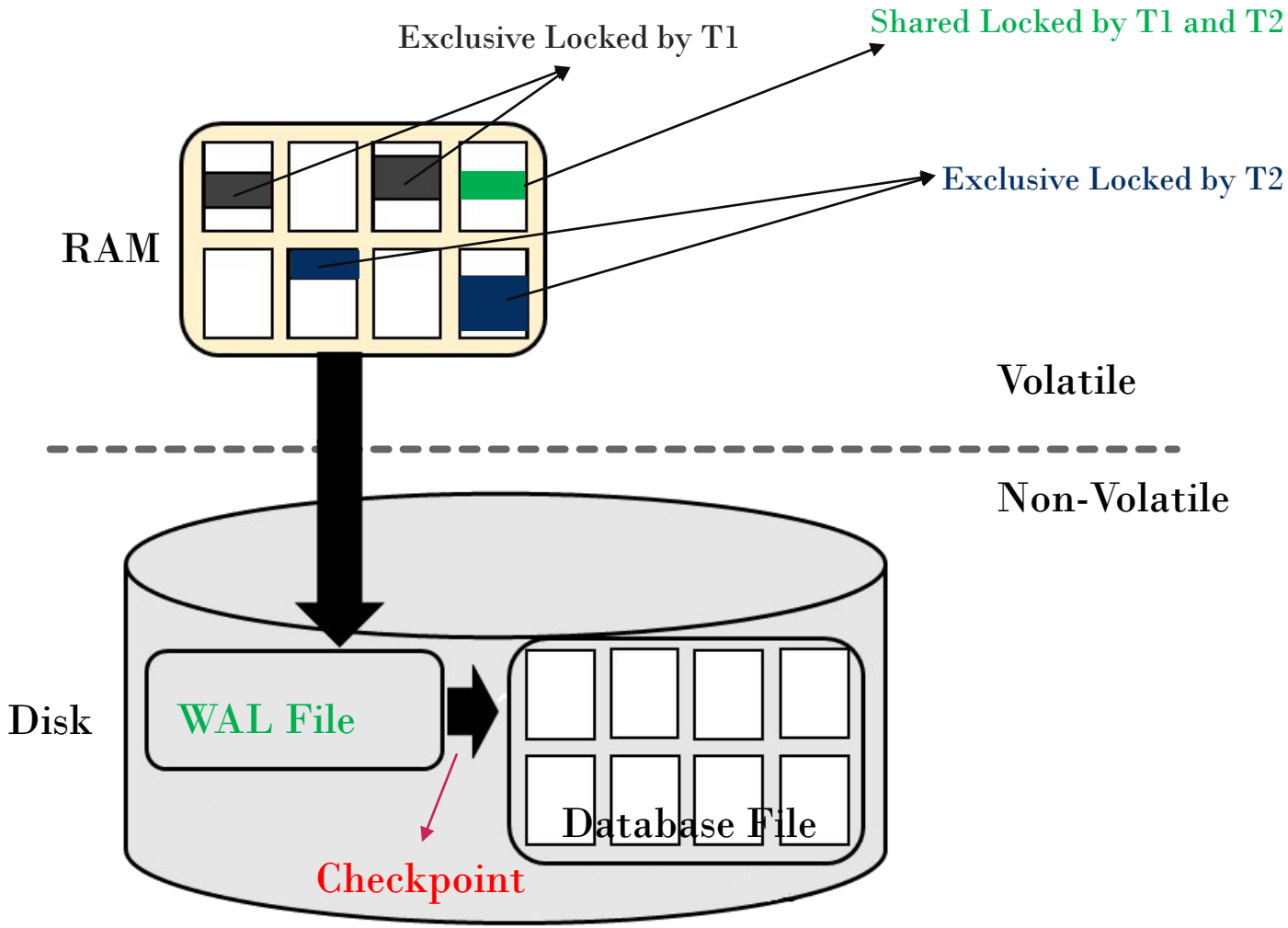
INCOMPLETE TRANSACTIONS AND SYSTEM CRASHES

Bringing the database to a consistent state after a system crash can be a slow process, since the DBMS must ensure that the effects of all transactions that completed prior to the crash are restored, and that the effects of

incomplete transactions are undone. The time required to recover from a crash can be reduced by periodically forcing some information to disk; this periodic operation is called a checkpoint.

INCOMPLETE TRANSACTIONS AND SYSTEM CRASHES

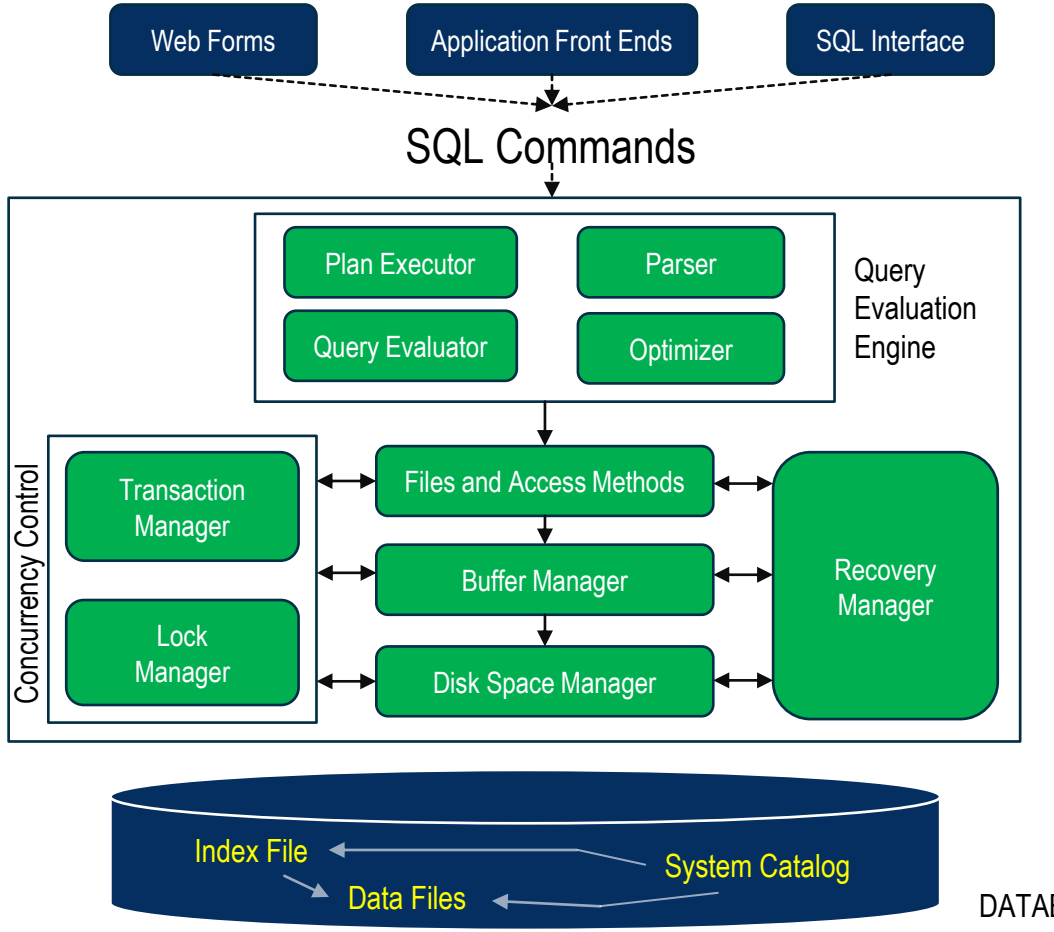
Overview



STRUCTURE OF A DBMS

The DBMS accepts SQL commands generated from a variety of user interfaces, produces query evaluation plans, executes these plans against the database, and returns the answers.

STRUCTURE OF A DBMS



When a user issues a query, the parsed query is presented to a query optimizer, which uses information about how the data is stored to produce an efficient execution plan for evaluating the query. An execution plan is a blueprint for evaluating a query, usually represented as a tree of relational operators.

The files and access methods layer code sits on top of the buffer manager, which brings pages in from disk to main memory as needed in response to read requests.

The recovery manager is responsible for maintaining a log and restoring the system to a consistent state after a crash.

PEOPLE WHO WORK WITH DATABASES

Database implementors build DBMS software, they work for vendors such as IBM, Oracle or Microsoft.

End users simply use applications written by database application programmers and so require little technical knowledge about DBMS software.

Database application programmers develop packages that facilitate data access for end users, who are usually not computer professionals, using the host or data languages and software tools that DBMS vendors provide. Such tools include report writers, spreadsheets and statistical packages. Application programmers should ideally access data through the external schema.

Database developer is responsible designing, improving and tuning databases. He/she is responsible to implement and maintain external and conceptual (logical) schema.

Database administrator (DBA) ensures that data availability and recovery from failures. Overview the physical design, helps on tuning and optimization.