DBMS Architecture

What is the goal of relational DBMSs?

Electronic record-keeping:

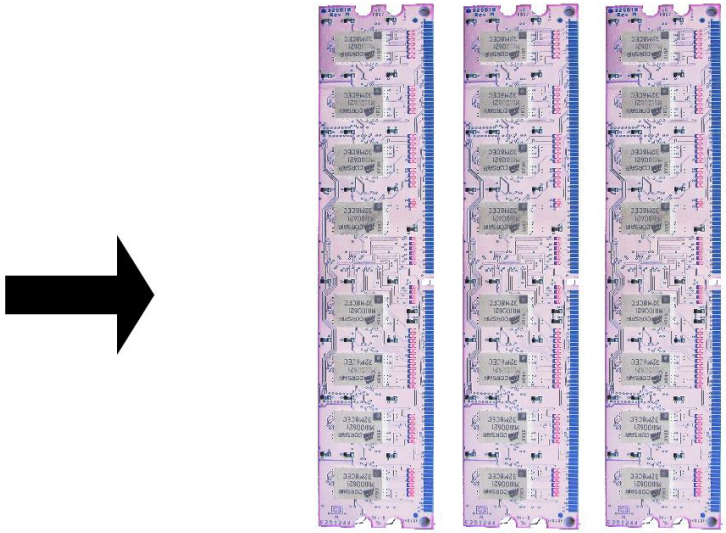
Fast and convenient access to information.

Why are databases cool?

- 3 reasons:
  - Normal forms
    - 1NF, 2NF, 3NF, BCNF, …
  - E/R model
  - SQL
- But…it's better to understand the technology!!
- So, why is Database **technology** so cool?
- Database Systems: **6 different things**

1 – Data Layouts

Employees		
ID	name	city code
23	Albert	45000
42	Rob	37000
77	Peter	50000

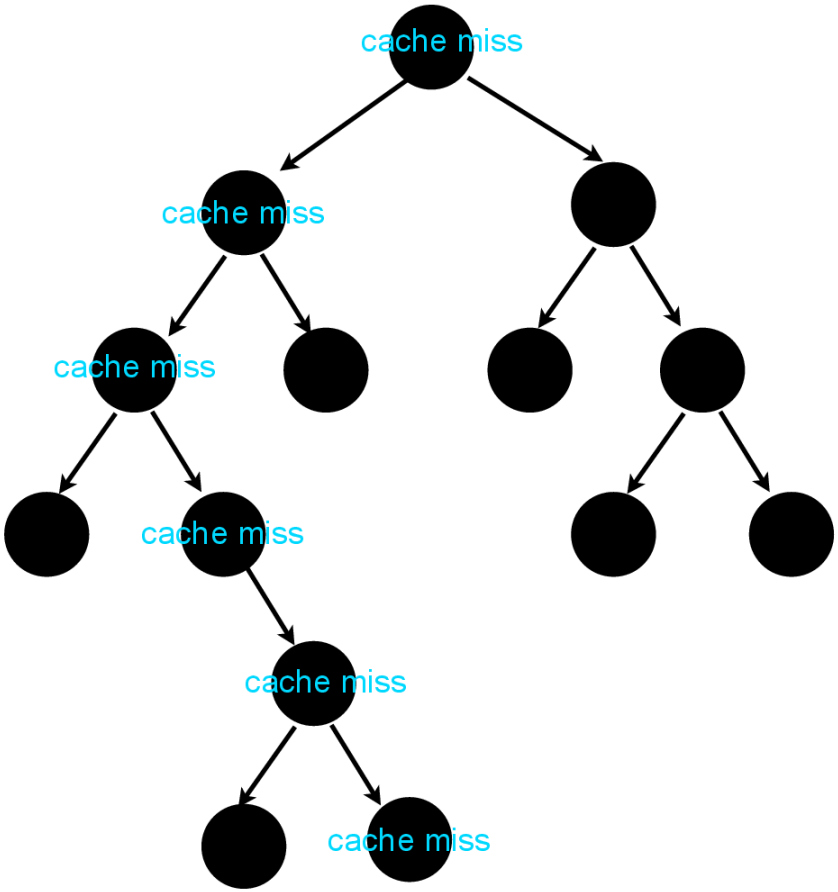


2D

Employees		
ID	name	city code
23	Albert	45000
42	Rob	37000
77	Peter	50000

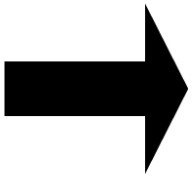
1D

2 – Data Structures



3 – Algorithms

```
SELECT    *
FROM      Fotographen
WHERE NOT EXISTS
   (...);
```

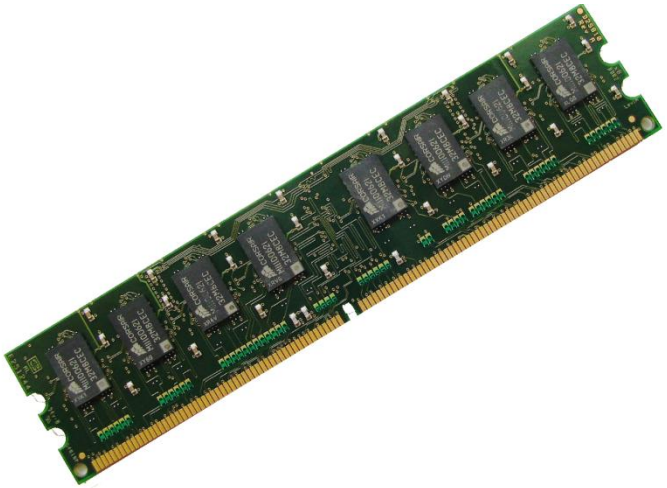


```
for (int i=0; i<rowMax;i++){
        row = read(Student);
        ...
}
```

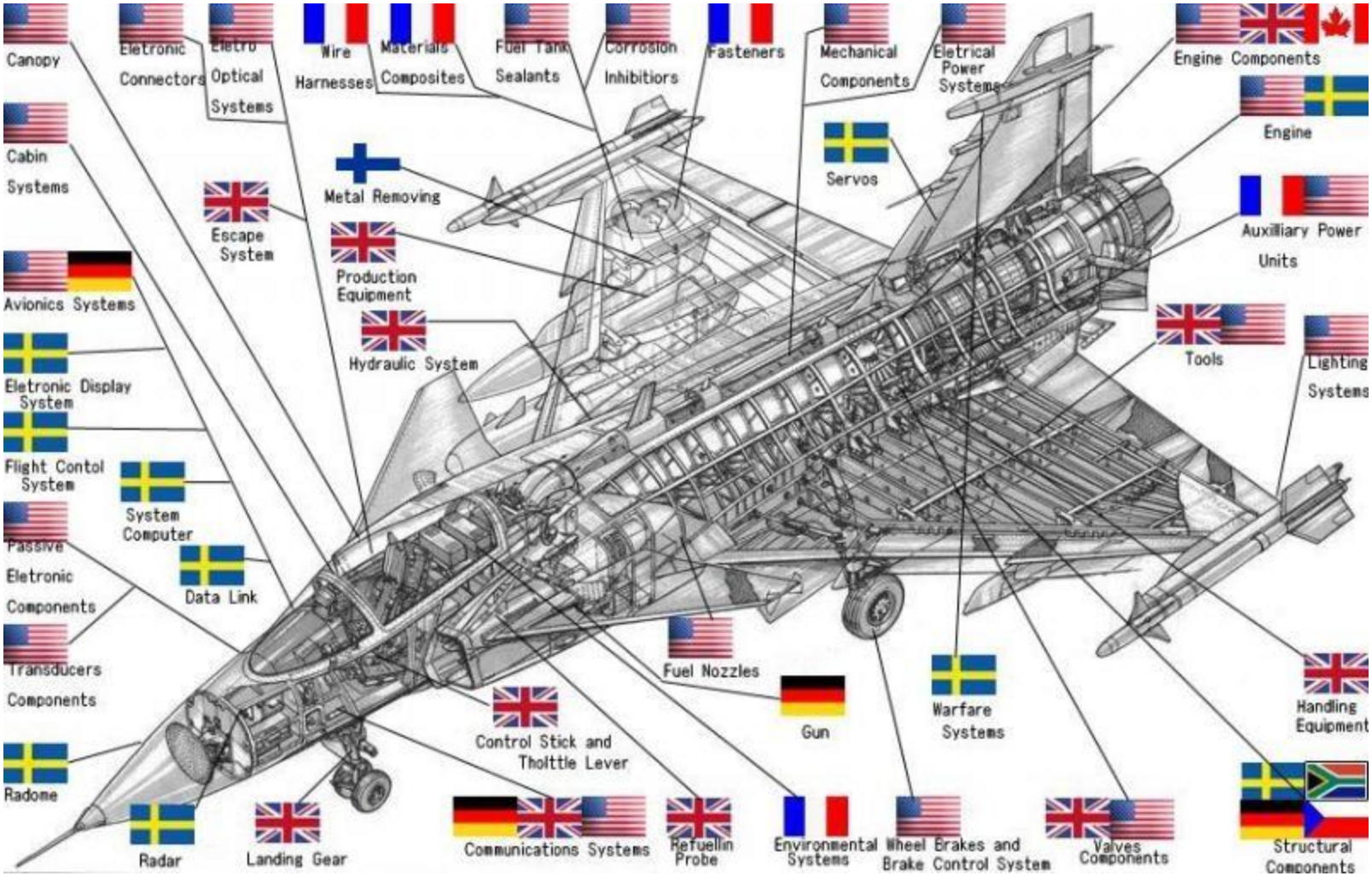
SQL

program

4 - Hardware



5 – Systems



6 - Users

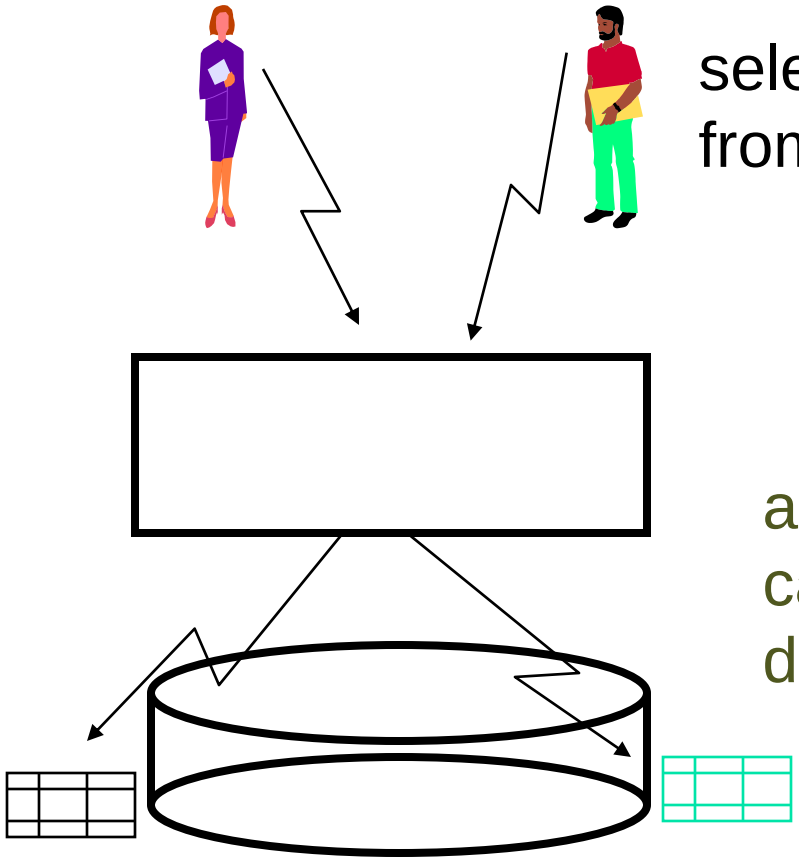
Database Systems

=

Data Layout
+
Data Structures
+
Algorithms
+
Hardware
+
Systems
+
Users

How do DBs work?

Pictorially:



How do DBs work?

```
%isql mydb
sql>create table student (
        ssn fixed;
        name char(20) );
```

/mydb

student	
ssn	name

How do DBs work?

```
sql>insert into student values
   (123, “Smith”);
sql>select * from student;
```

student	
ssn	name
123	Smith

How do DBs work - cont'd

More than one tables - joins

Eg., roster (names only) for 'db'

```
sql> select name
        from student, takes
        where student.ssn = takes.ssn
        and takes.c-id = 'db'
```

student	
ssn	name

takes		
ssn	c-id	grade

Views - a powerful tool!

what and why?

- suppose dtsouma is allowed to see **only** ssn's and GPAs, but not individual grades
- -> VIEWS!
- Views = 'virtual tables'

Views

```
sql> create view fellowship as (
        select ssn,  avg(grade)
        from takes   group by ssn);
```

takes		
ssn	c-id	grade
123	ai	4
123	os	3
234	ai	3

ssn	avg(grade
123	3.5
234	3

Views

sql> select * from fellowship;

takes		
ssn	**c-id**	**grade**
123	ai	4
123	os	3
234	ai	3

ssn	avg(grade
123	3.5
234	3

Three-Schema Architecture

- Proposed to support DBMS characteristics of:
  - **Program-data independence.**
  - Support of **multiple views** of the data.
- Not explicitly used in commercial DBMS products, but has been useful in explaining database system organization

Data Independence

- **Logical Data Independence:**
  - Can change the conceptual schema without having to change the external schemas and their associated application programs.
- **Physical Data Independence:**
  - The capacity to change the internal schema without having to change the conceptual schema.
  - For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance

Data Independence (continued)

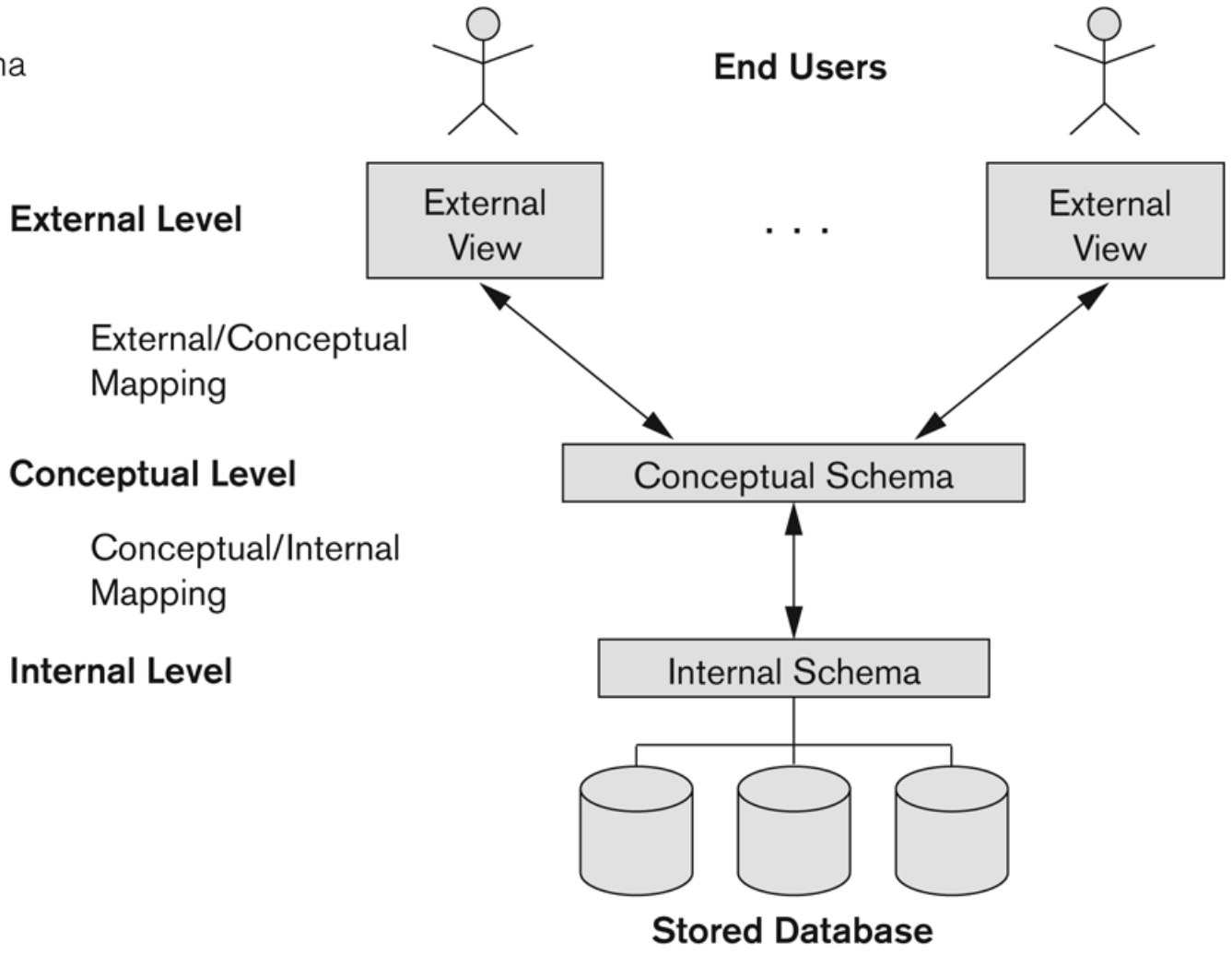
- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence.
- The higher-level schemas themselves are **unchanged**.
  - Hence, the application programs need not be changed since they refer to the external schemas.

Three-Schema Architecture

- Defines DBMS schemas at ***three*** levels:
  - **Internal schema** at the internal level to describe physical storage structures and access paths (e.g indexes).
    - Typically uses a **physical** data model.
    - how are these tables stored, how many bytes / attribute etc
  - **Conceptual schema** at the conceptual level to describe the structure and constraints for the whole database for a community of users.
    - Uses a **conceptual** or an **implementation** data model.
  - **External schemas** at the external level to describe the various user views.
    - Usually uses the same data model as the conceptual schema.

The three-schema architecture

Figure 2.2
The three-schema architecture.



Three-Schema Architecture

- Mappings among schema levels are needed to transform requests and data.
  - Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.
  - Data extracted from the internal DBMS level is reformatted to match the user's external view (e.g. formatting the results of an SQL query for display in a Web page)

Data Models

- **Data Model:**
  - A set of concepts to describe the ***structure*** of a database, the ***operations*** for manipulating these structures, and ***constraints*** that the database should obey.
- **Data Model Structure and Constraints:**
  - Constructs are used to define the database structure
  - Constructs typically include ***elements*** (and their ***data types***) as well as groups of elements (e.g. ***entity, record, table***), and ***relationships*** among such groups
  - Constraints specify some restrictions on valid data; these constraints must be enforced at all times

Categories of Data Models

- **Conceptual (high-level, semantic) data models:**
  - Provide concepts that are close to the way many users perceive data.
    - (Also called ***entity-based*** or ***object-based*** data models.)
- **Physical (low-level, internal) data models:**
  - Provide concepts that describe details of how data is stored in the computer. These are usually specified in an ad-hoc manner through DBMS design and administration manuals
- **Implementation (representational) data models:**
  - Provide concepts that fall between the above two, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).

Schemas versus Instances

- Database Schema:
  - The ***description*** of a database.
  - Includes descriptions of the database structure, data types, and the constraints on the database.
- Schema Diagram:
  - An ***illustrative*** display of (most aspects of) a database schema.
- Schema Construct:
  - A ***component*** of the schema or an object within the schema, e.g., STUDENT, COURSE.

Schemas versus Instances

- Database State:
  - The actual data stored in a database at a ***particular moment in time***. This includes the collection of all the data in the database.
  - Also called database **instance (or occurrence or snapshot)**.
    - The term *instance* is also applied to individual database components, e.g. *record instance, table instance, entity instance*

Example of a Database Schema

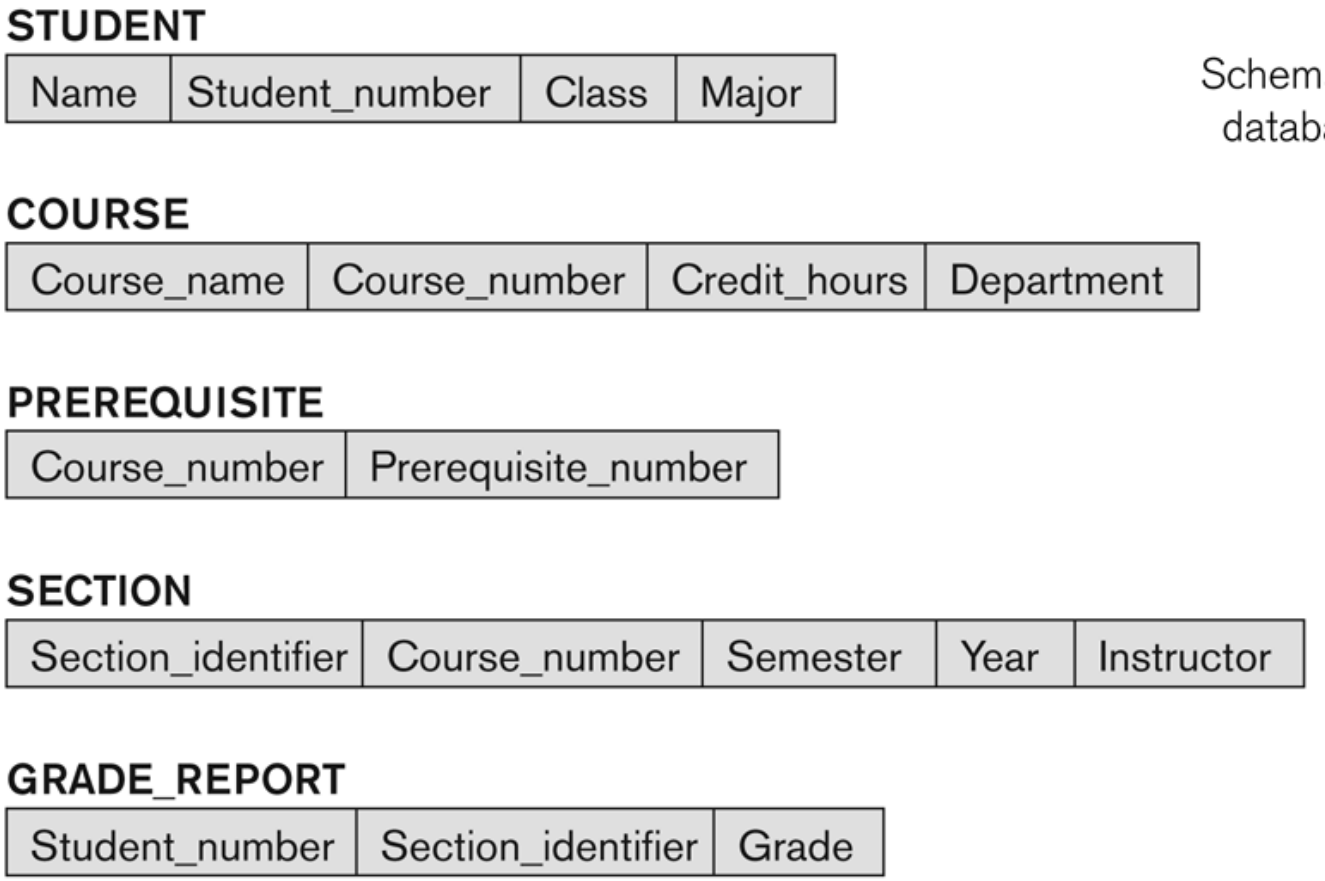


Figure 2.1
Schema diagram for the database in Figure 1.2.

Example of a database state

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

DBMS Languages

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
  - High-Level or Non-procedural Languages: These include the relational language SQL
    - May be used in a standalone way or may be embedded in a programming language
  - Low Level or Procedural Languages:
    - These must be embedded in a programming language

DBMS Interfaces

- Stand-alone query language interfaces
  - Example: Entering SQL queries at the DBMS interactive SQL interface (e.g. SQL*Plus in ORACLE)
- Programmer interfaces for embedding DML in programming languages
- User-friendly interfaces
  - Menu-based, forms-based, graphics-based, etc.

Database System Utilities

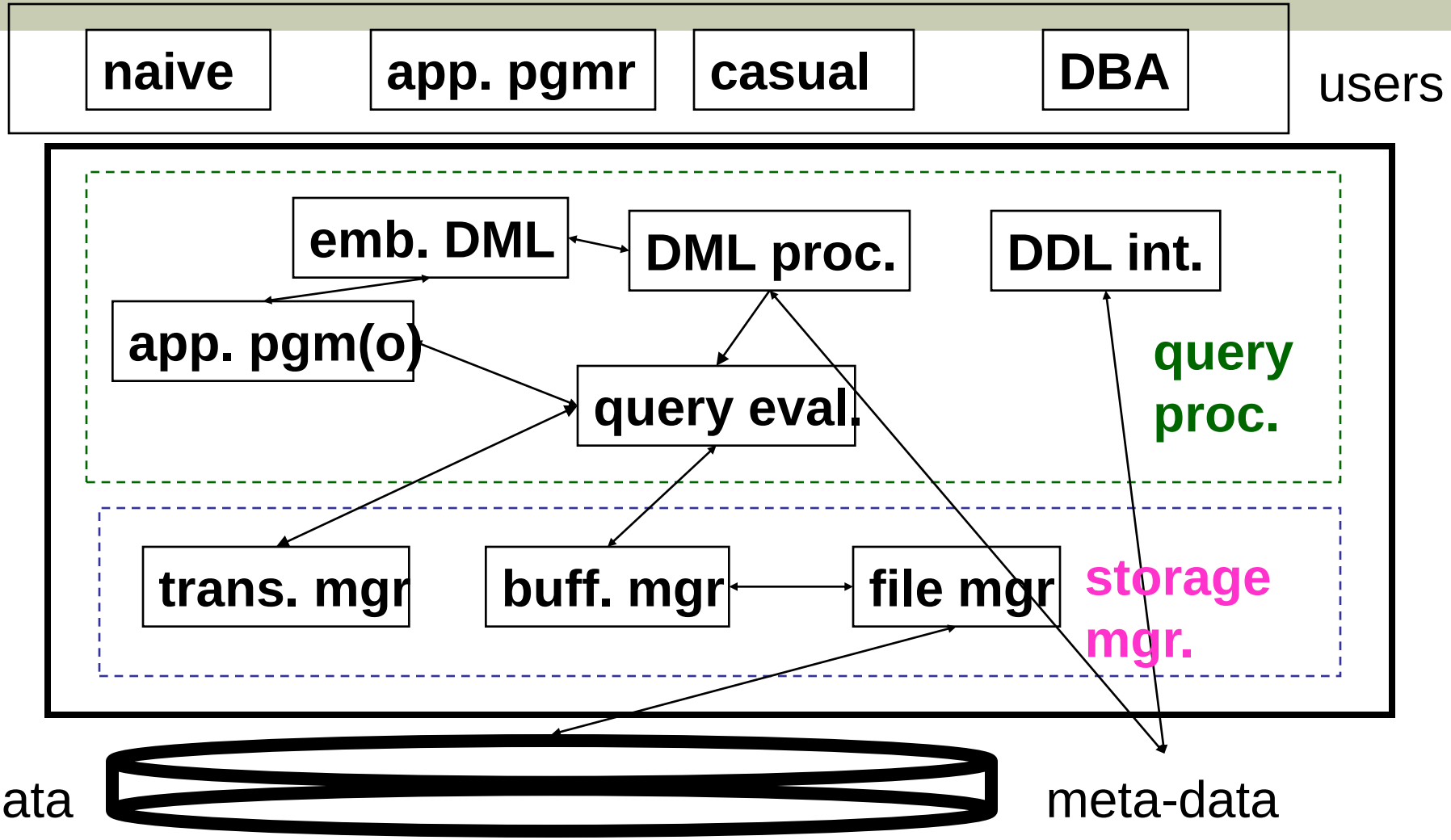
- To perform certain functions such as:
  - Loading data stored in files into a database. Includes data conversion tools.
  - Backing up the database periodically on tape.
  - Reorganizing database file structures.
  - Report generation utilities.
  - Performance monitoring utilities.
  - Other functions, such as sorting, user monitoring, data compression, etc.

Other Tools

- Data dictionary / repository:
  - Used to store schema descriptions and other information such as design decisions, application program descriptions, user information, usage standards, etc.
  - **Active data dictionary** is accessed by DBMS software and users/DBA.
  - **Passive data dictionary** is accessed by users/DBA only.

Overall system architecture

- [Users]
- DBMS
  - query processor
  - storage manager
- [Files]



Overall system architecture

- query processor
  - DML compiler
  - embedded DML pre-compiler
  - DDL interpreter
  - Query evaluation engine

Overall system architecture (cont'd)

- storage manager
  - authorization and integrity manager
  - transaction manager
  - buffer manager
  - file manager

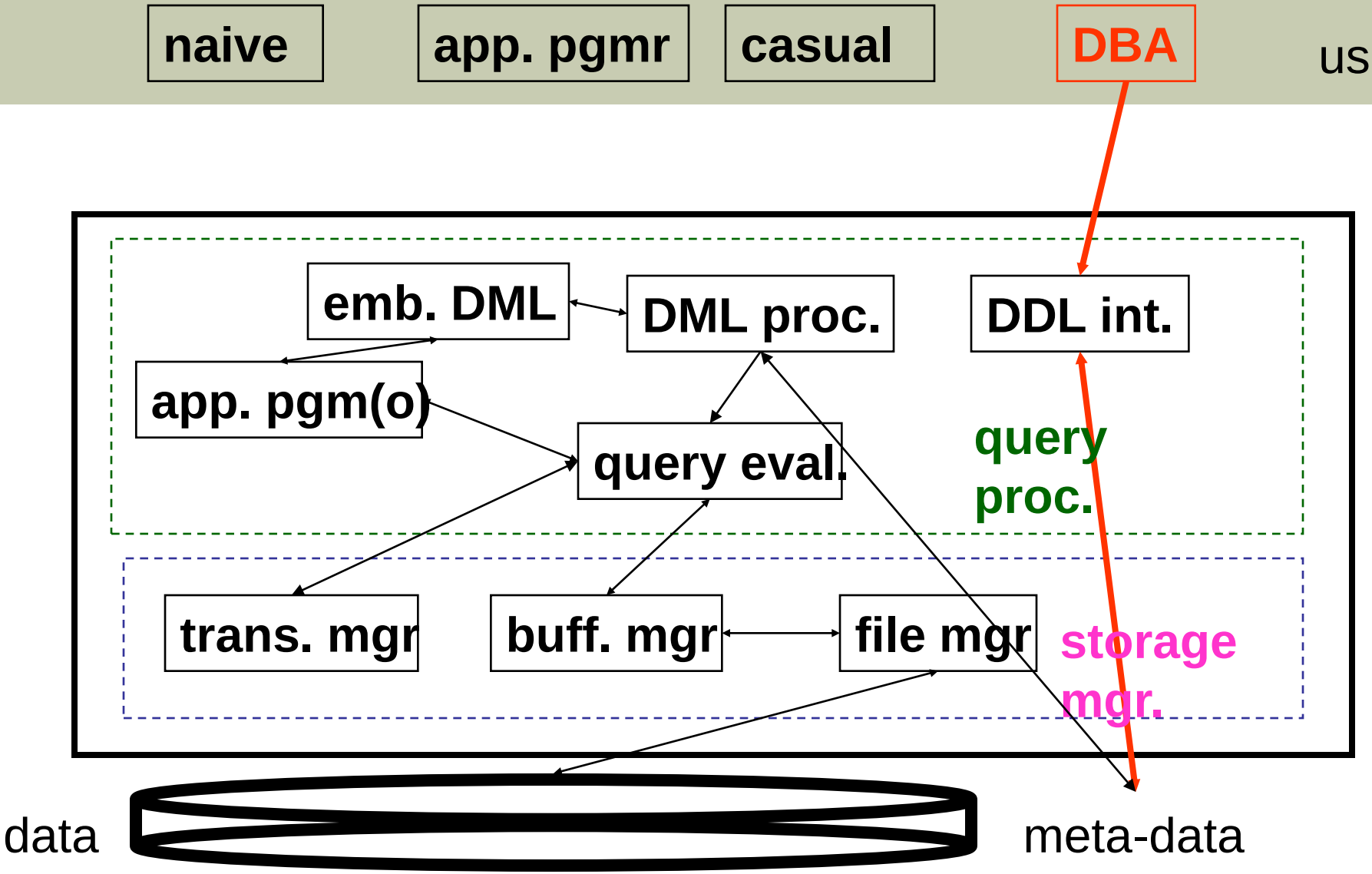
Overall system architecture (cont'd)

- Files
  - data files
  - data dictionary = catalog (= meta-data)
  - indices
  - statistical data

Some examples:

- DBA doing a DDL (data definition language) operation, eg.,

  create table student ...



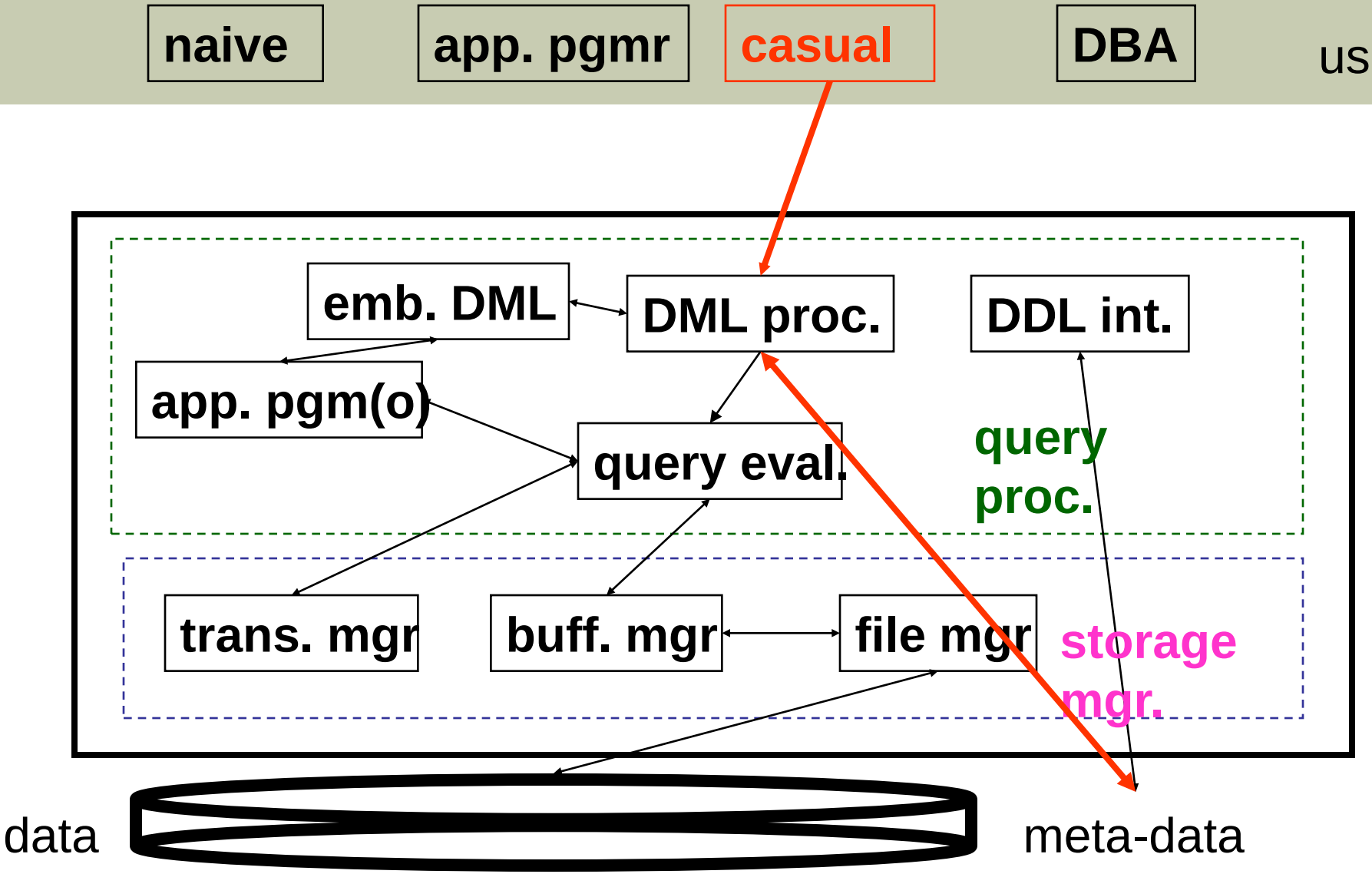
Some examples:

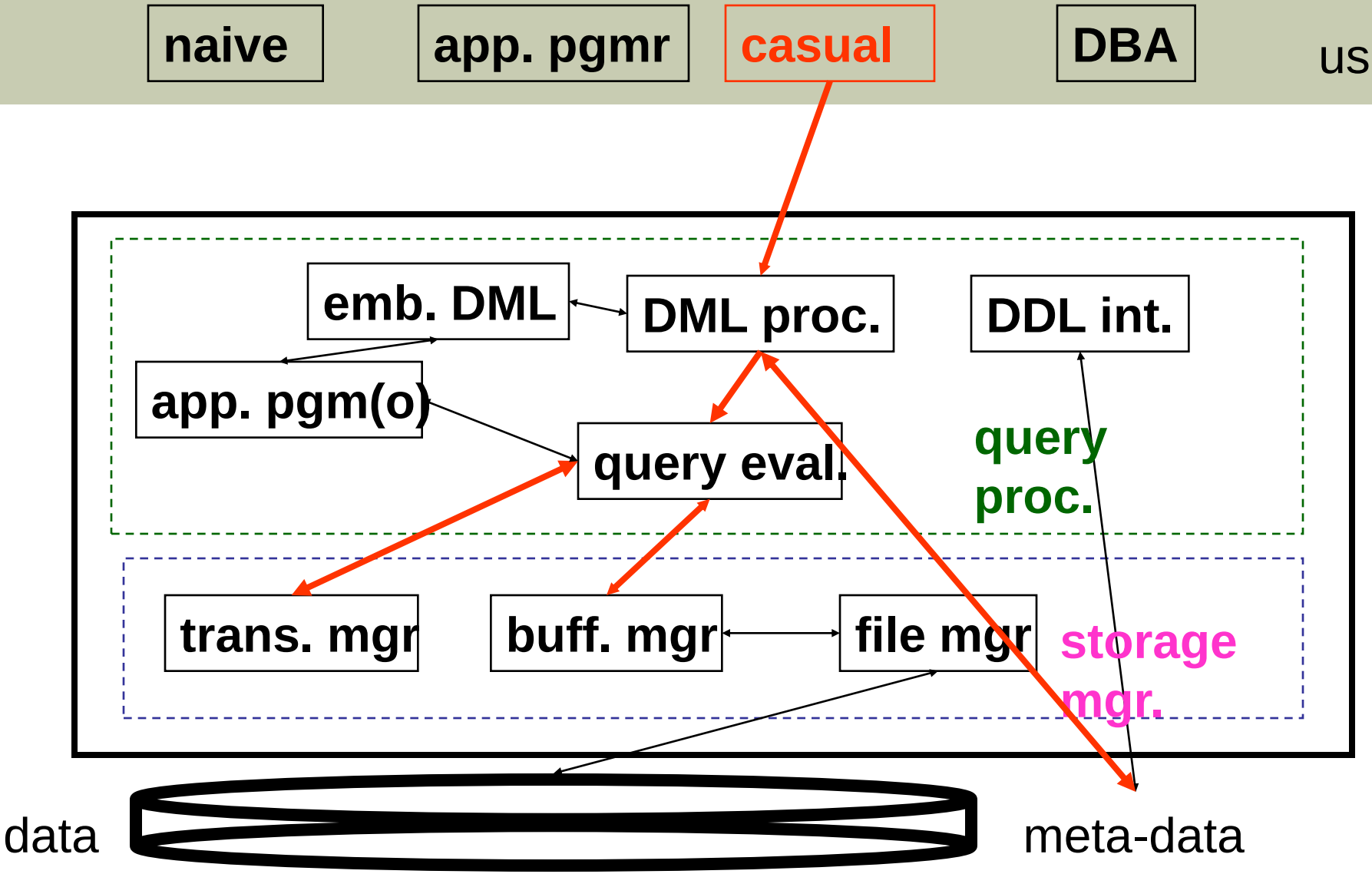
- casual user, asking for an update, eg.:

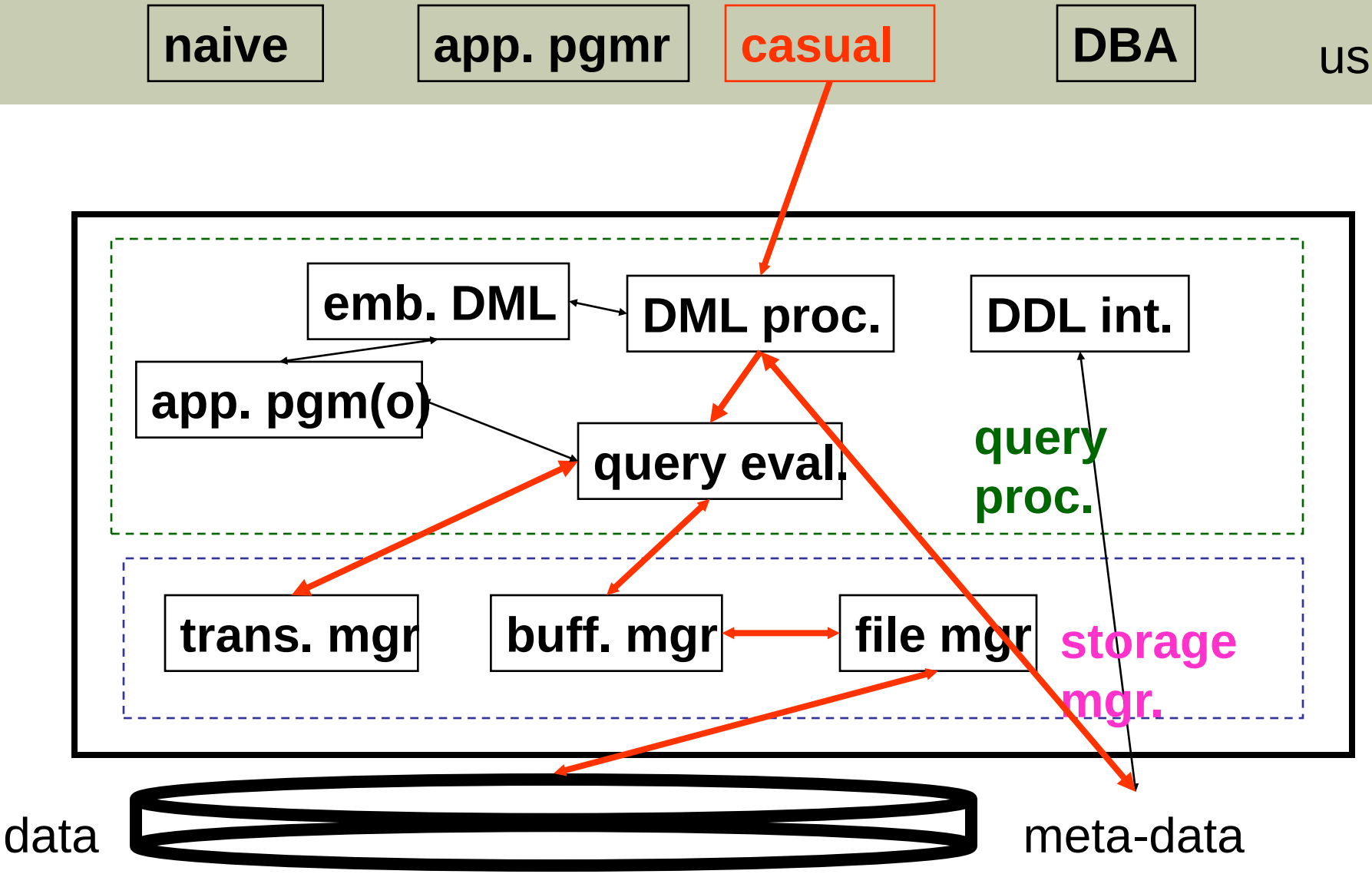
  update student

  set name to ‘smith’

  where ssn = ‘345’



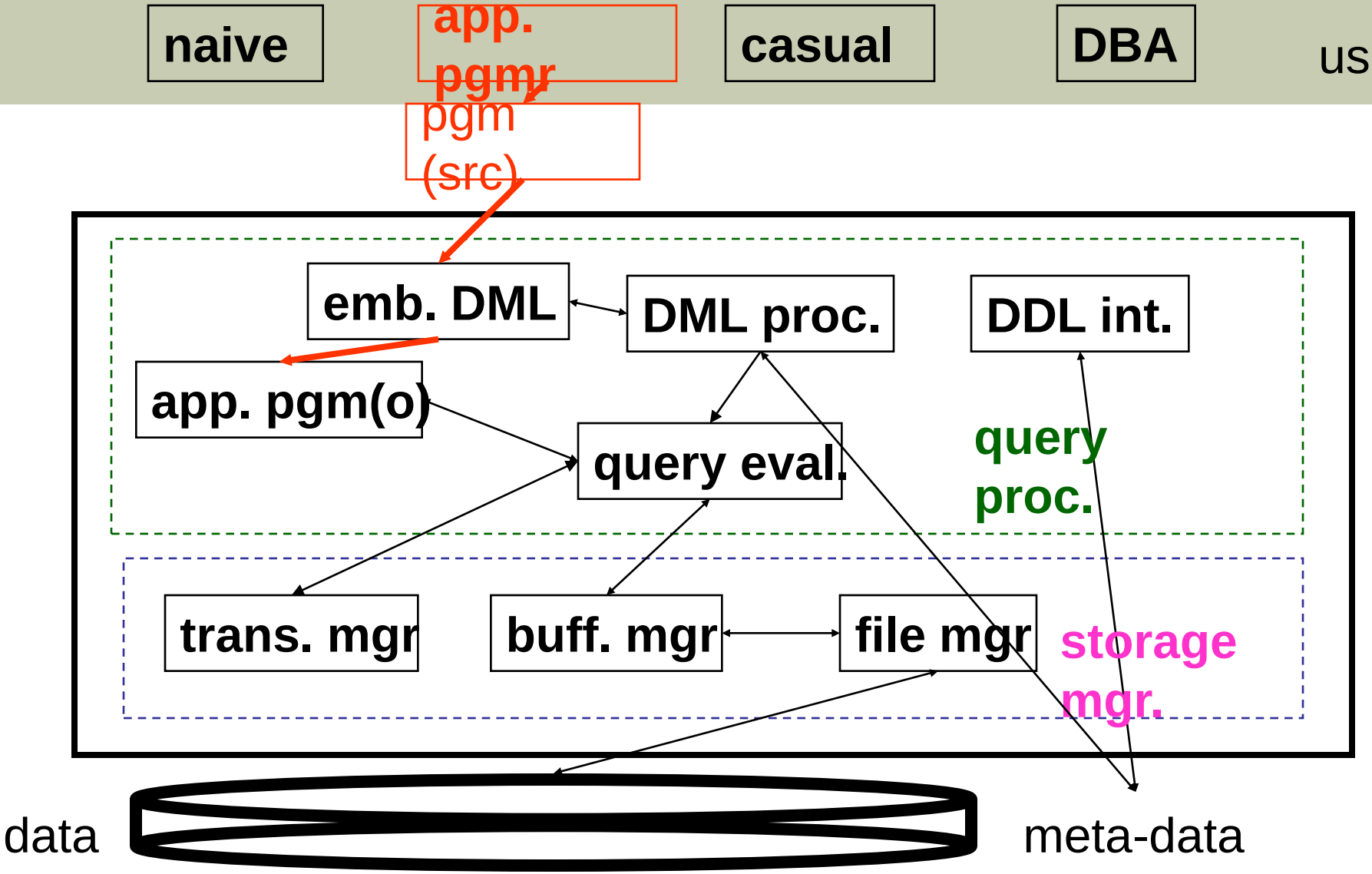




Some examples:

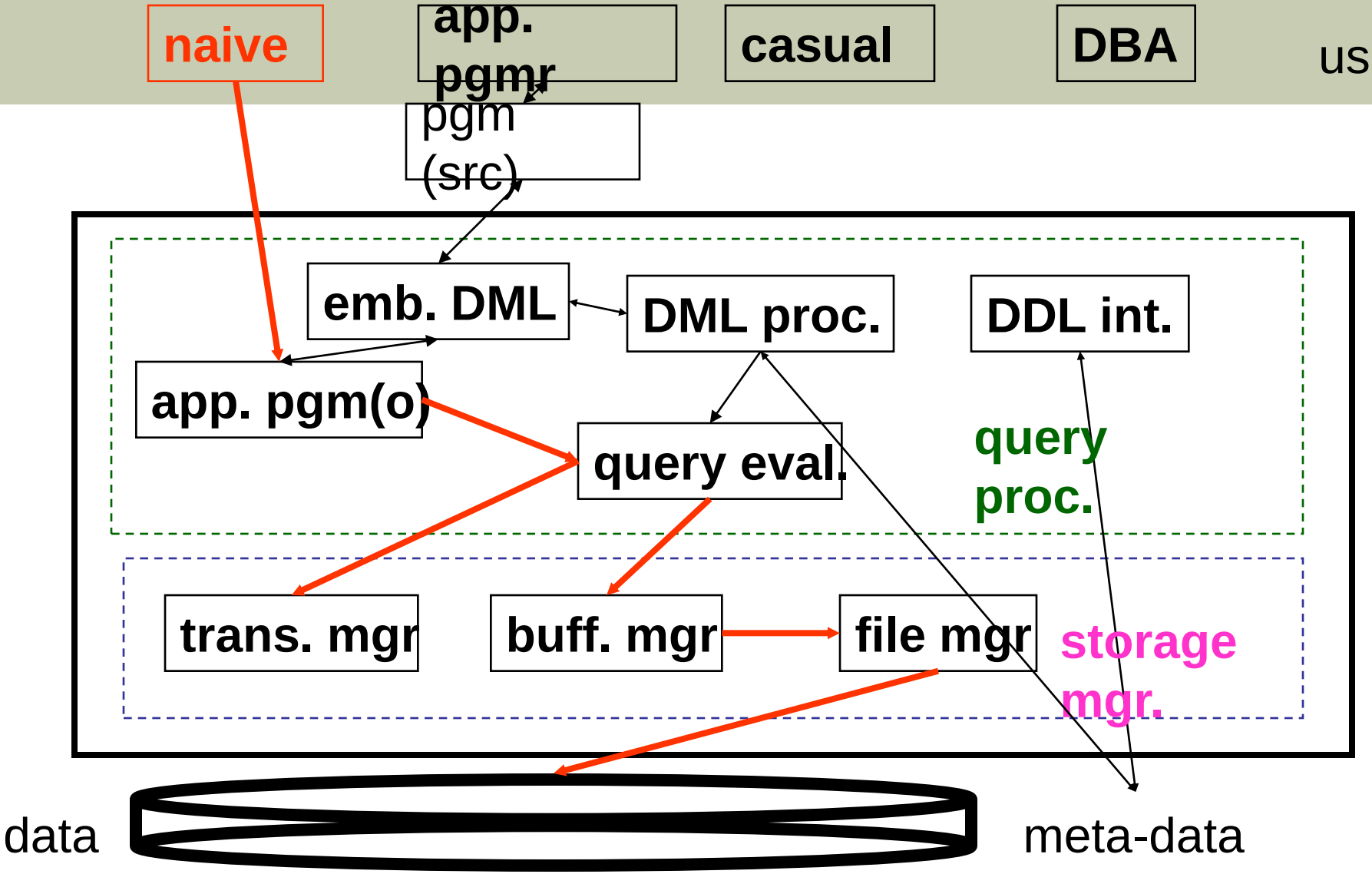
- app. programmer, creating a report, eg

```
main(){
....
exec sql “select * from student”
...
}
```



Some examples:

- ‘naive’ user, running the previous app.



A Physical Centralized Architecture

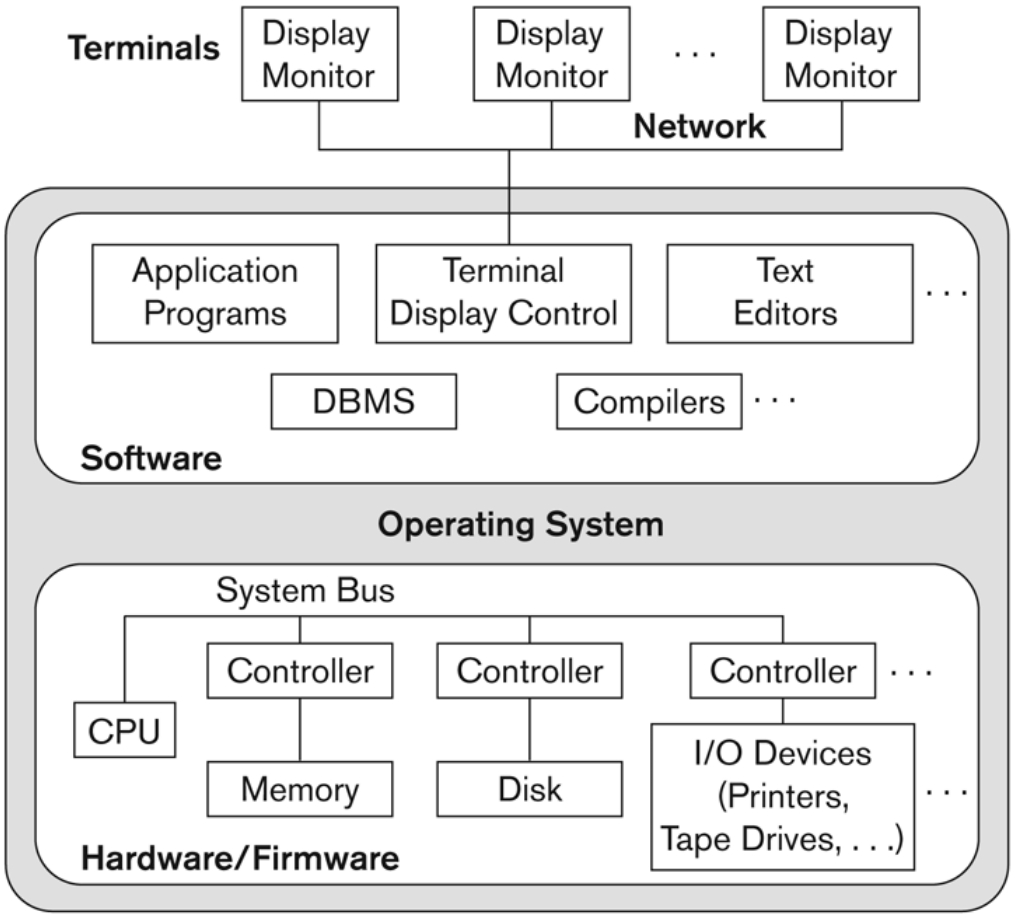
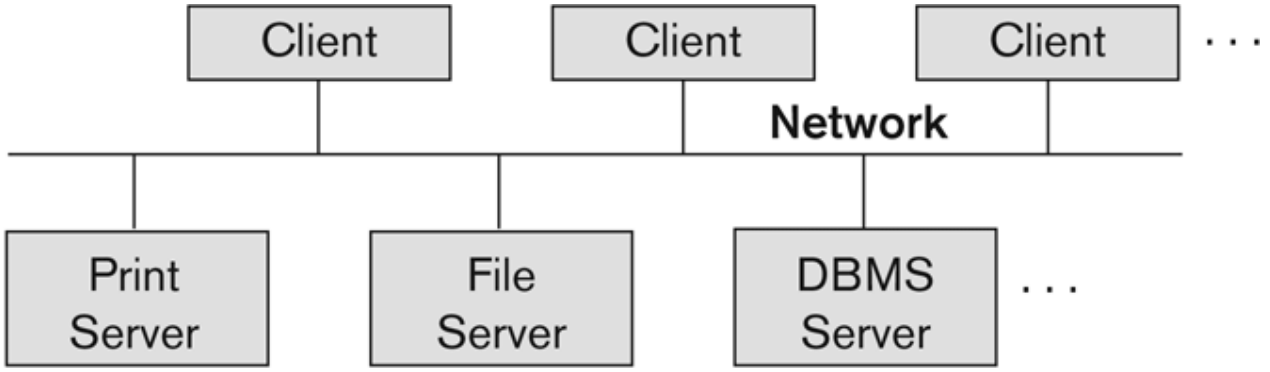


Figure 2.4
A physical centralized architecture.

Logical two-tier client server architecture

Figure 2.5
Logical two-tier client/server architecture.

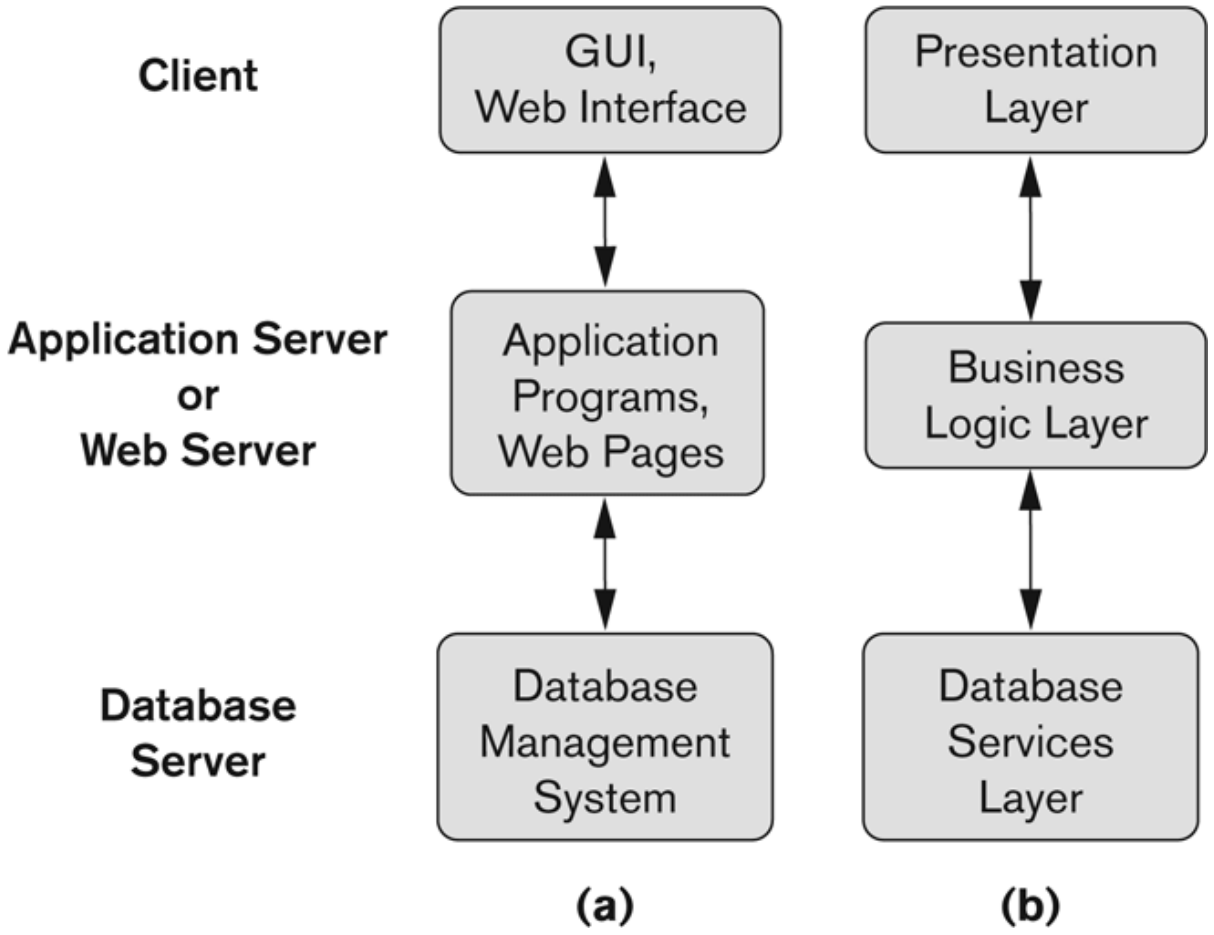


DBMS Server

- Provides database query and transaction services to the clients
- Relational DBMS servers are often called SQL servers, query servers, or transaction servers
- Applications running on clients utilize an Application Program Interface (**API**) to access server databases via standard interface such as:
  - ODBC: Open Database Connectivity standard
  - JDBC: for Java programming access
- Client and server must install appropriate client module and server module software for ODBC or JDBC

Three-tier client-server architecture

Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.



Classification of DBMSs

- Based on the data model used
  - Traditional: Relational, Network, Hierarchical.
  - Emerging: Object-oriented, Object-relational.
- Other classifications
  - Single-user (typically used with personal computers)
    vs. multi-user (most DBMSs).
  - Centralized (uses a single computer with one database)
    vs. distributed (uses multiple computers, multiple databases)

Variations of Distributed DBMSs (DDBMSs)

- Homogeneous DDBMS
- Heterogeneous DDBMS
- Federated or Multidatabase Systems
- Distributed Database Systems have now come to be known as client-server based database systems because:
  - They do not support a totally distributed environment, but rather a set of database servers supporting a set of clients.

Cost considerations for DBMSs

- Cost Range: from free open-source systems to configurations costing millions of dollars
- Examples of free relational DBMSs: MySQL, PostgreSQL, others
- Commercial DBMS offer additional specialized modules, e.g. time-series module, spatial data module, document module, XML module
  - These offer additional specialized functionality when purchased separately
  - Sometimes called cartridges (e.g., in Oracle) or blades
- Different licensing options: site license, maximum number of concurrent users (seat license), single user, etc.