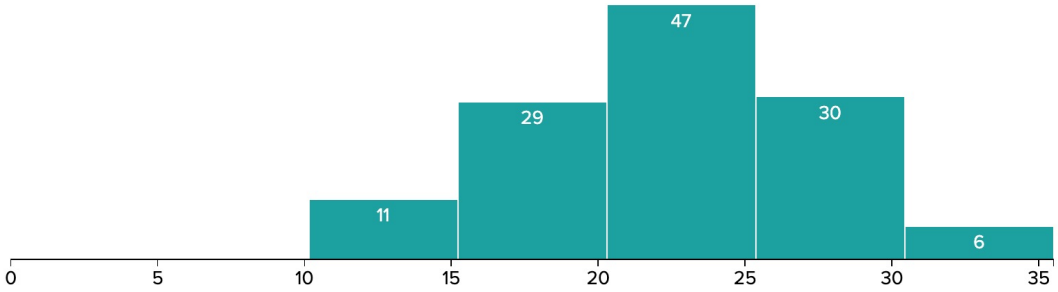
Administrivia

Midterm 1 results out. See gradescope

Regrade requests end of Oct 18.



MINIMUM	MEDIAN	MAXIMUM	MEAN	STD DEV
12.0	22.5	34.0	22.46	4.88

Project 1 part 2 due today 10AM

APIs

Eugene Wu

Topics

Interfacing with applications

- Types of DBMSes
- Database APIs (DBAPIS)
- Cursors

How to use SQL in an application?

SQL is not a general purpose programming language

- Designed for data access/transform, and optimization
- Applications are complex, require "business logic"

Many Options

Extend SQL to be turing complete

- Makes optimization very very hard
- Technically, `recursive WITH` clause makes SQL turing complete…

Embedded SQL

- extend language by "embedding" SQL into it

DBAPI

- Low-level library with core database calls

Object-relational mapping (ORM)

- Define DB-backed classes, magically map between objects & DB tuples

Embedded SQL

Extend programming language

e.g., `EXEC SQL sql-query`

goes through a preprocessor

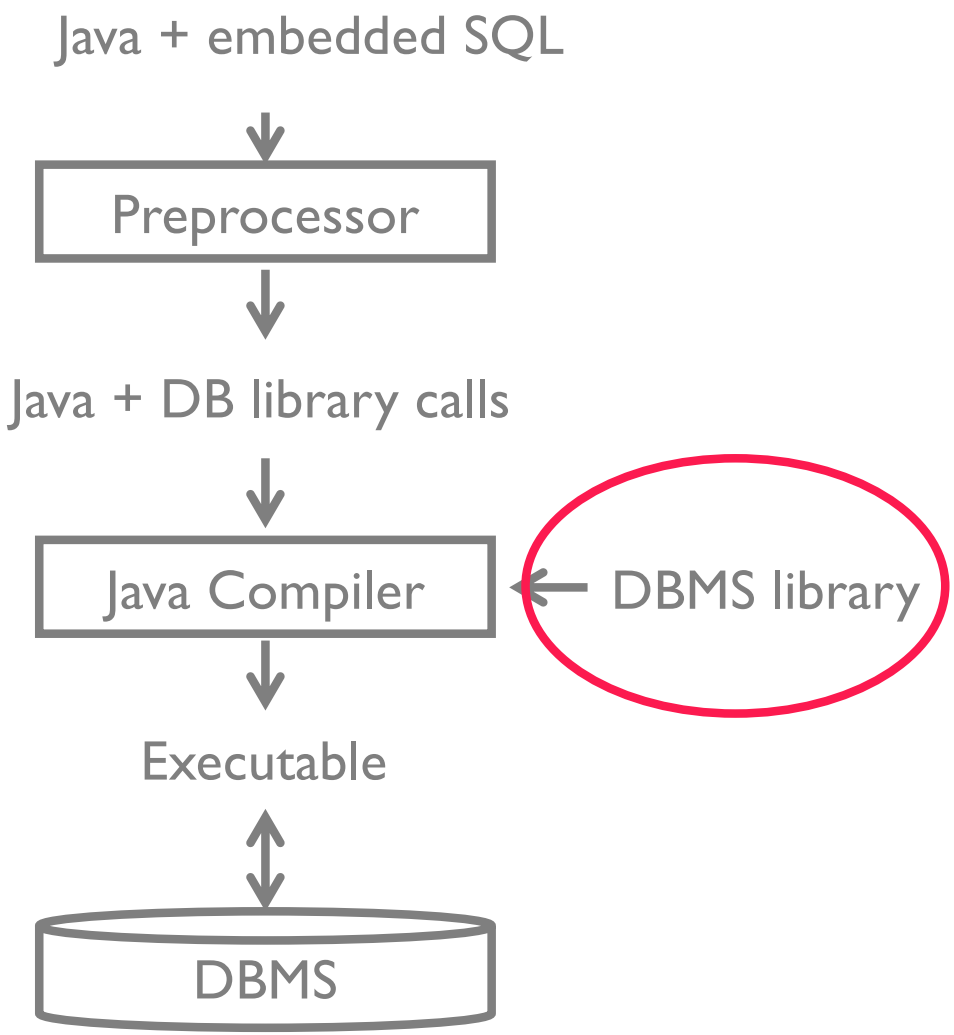
Compiled into program that interacts with DBMS directly

```
...
if (user == 'admin'){

  EXEC SQL select * …

} else {
...
```

Embedded SQL




```
...
if (user == 'admin'){

  EXEC SQL select * …

} else {
...
```

What does a library need to do?

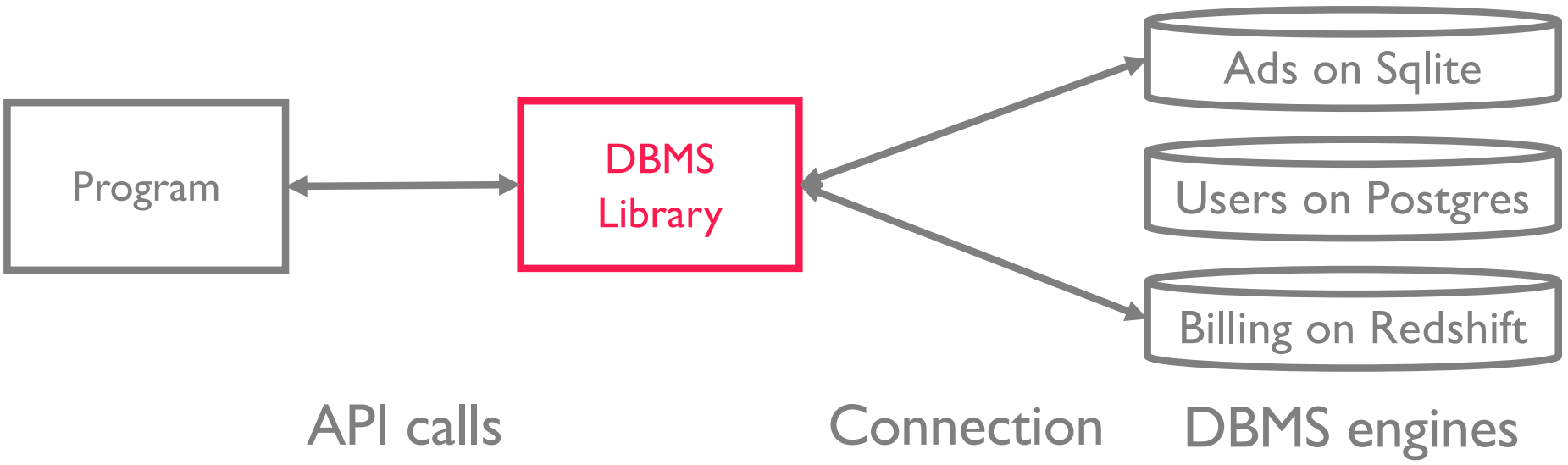
Single interface to possibly multiple **DBMS engines**

Connect to a database

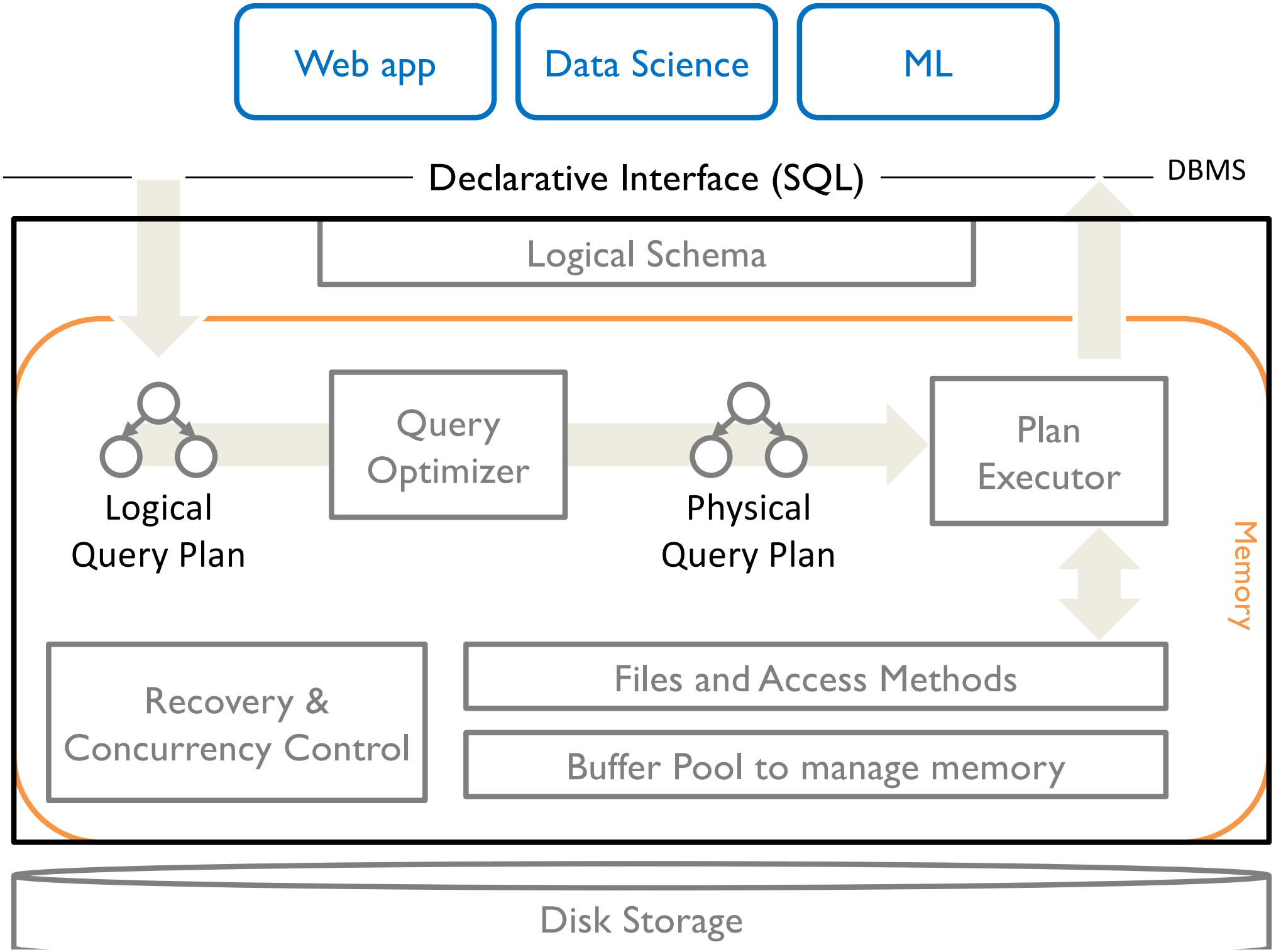
Manage transactions (later)

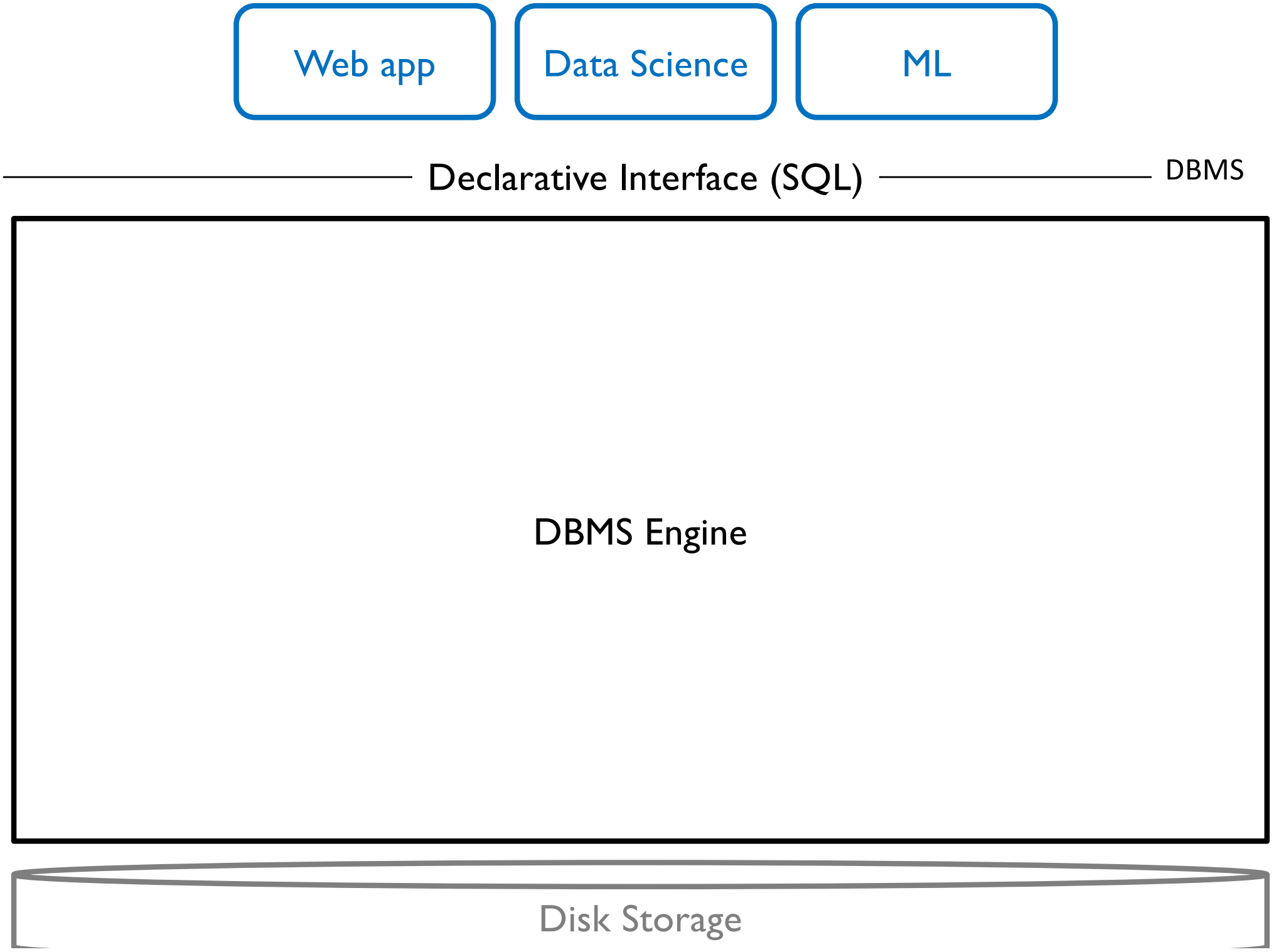
Map objects between host language and DBMS

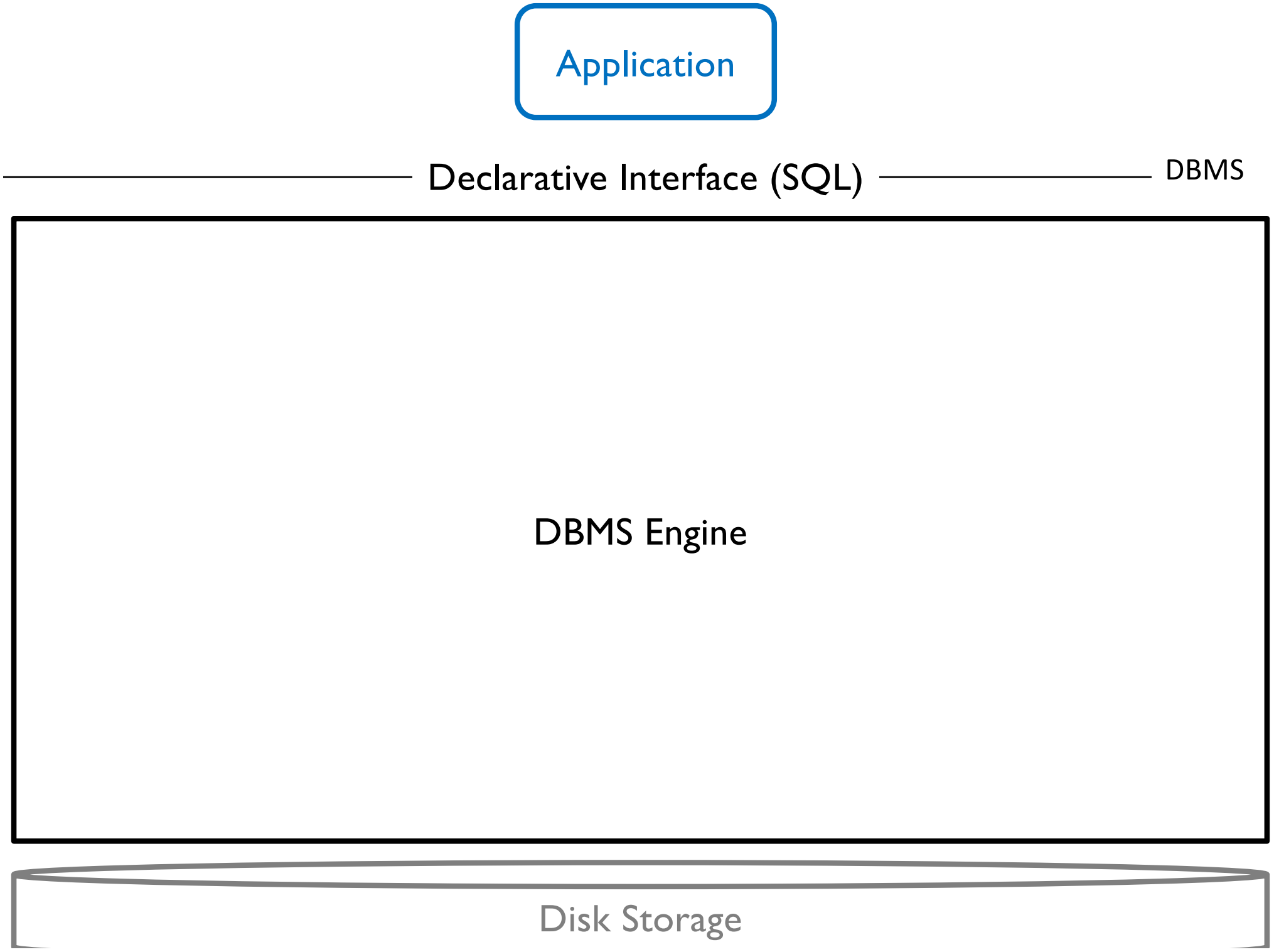
Manage query results

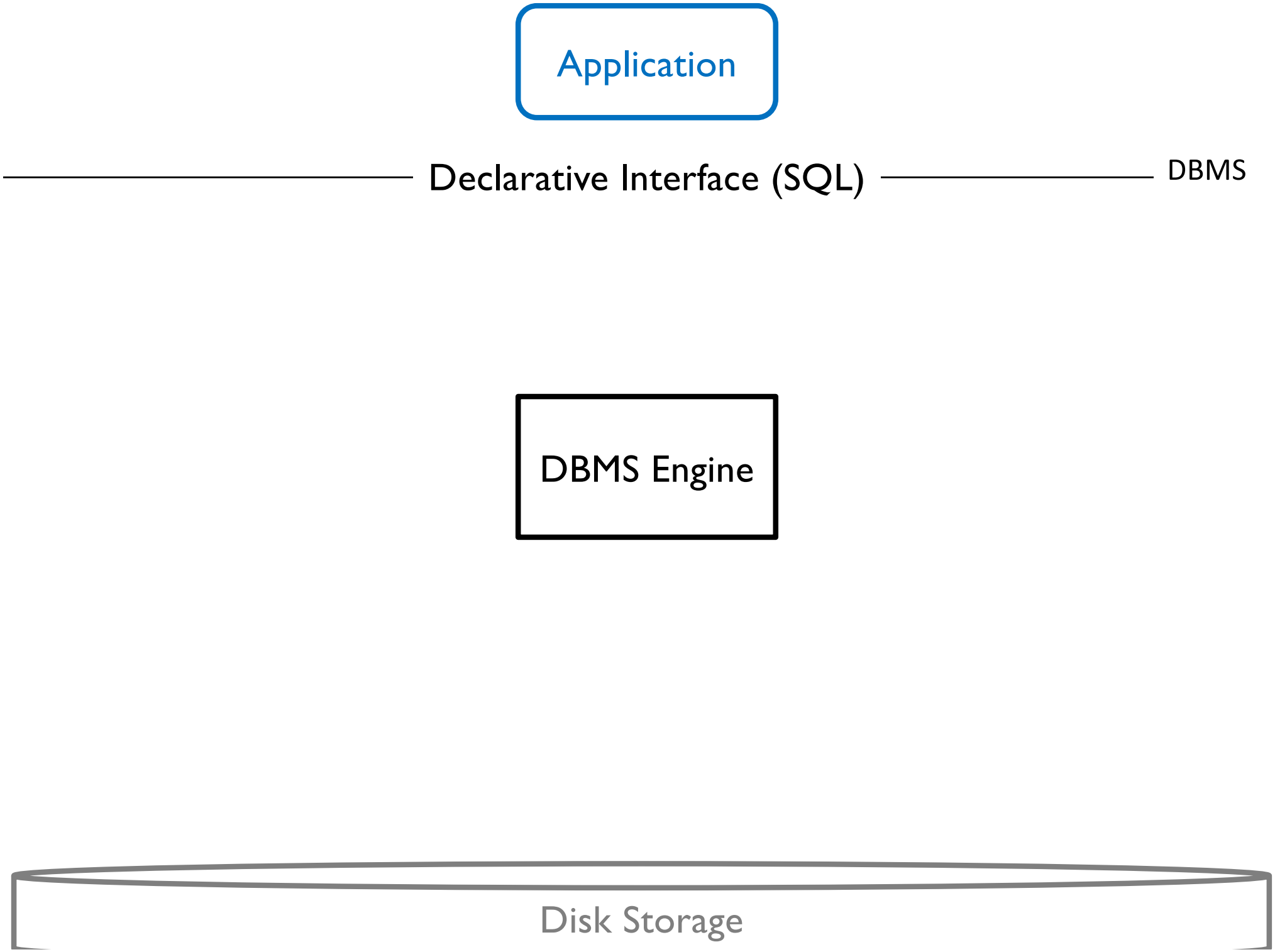


DBMS Engines

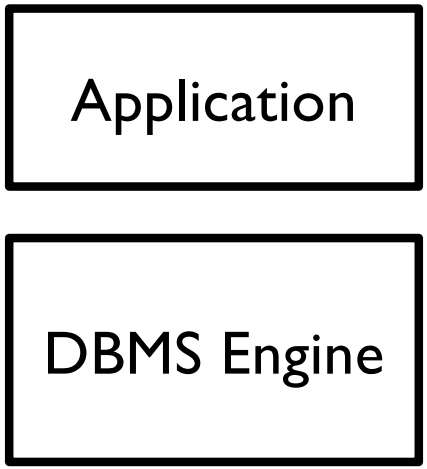

Web app
Data Science
ML
Declarative Interface (SQL)
DBMS
Logical Schema
Logical
Query Plan
Query
Optimizer
Physical
Query Plan
Plan
Executor
Memory
Recovery &
Concurrency Control
Files and Access Methods
Buffer Pool to manage memory
Disk Storage


Web app
Data Science
ML
Declarative Interface (SQL)
DBMS
DBMS Engine
Disk Storage

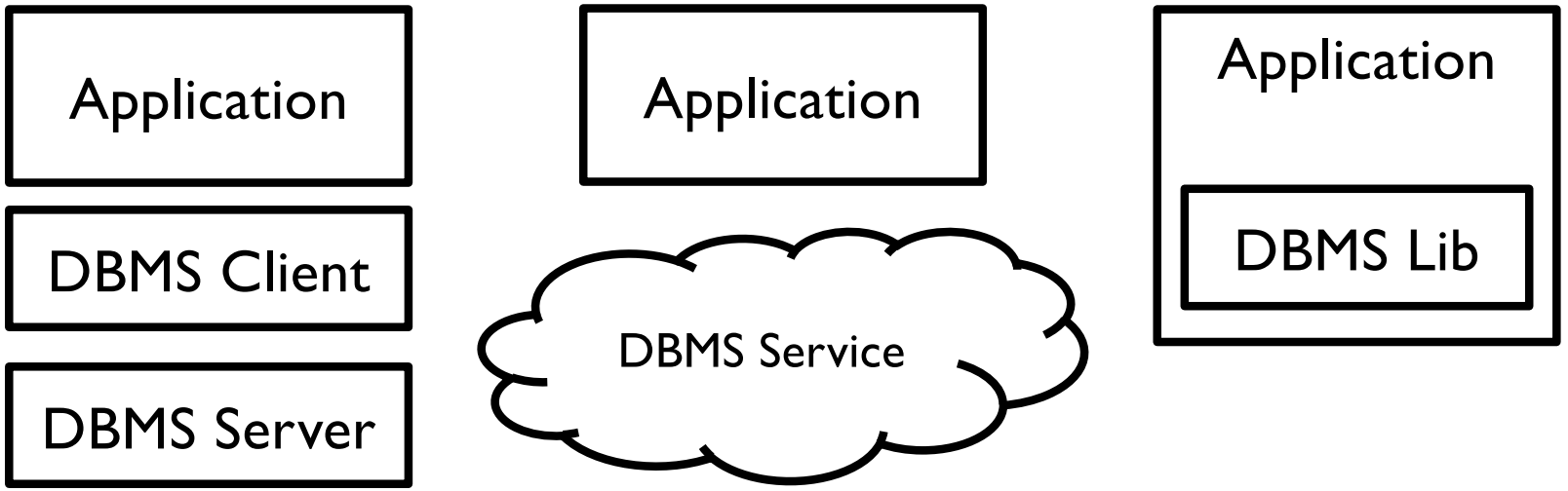

Application
Declarative Interface (SQL)
DBMS
DBMS Engine
Disk Storage


Application
Declarative Interface (SQL)
DBMS
DBMS Engine
Disk Storage

DBMS Engines



DBMS Engines: 3 Types



Client-Server DBMS

Cloud DBMS

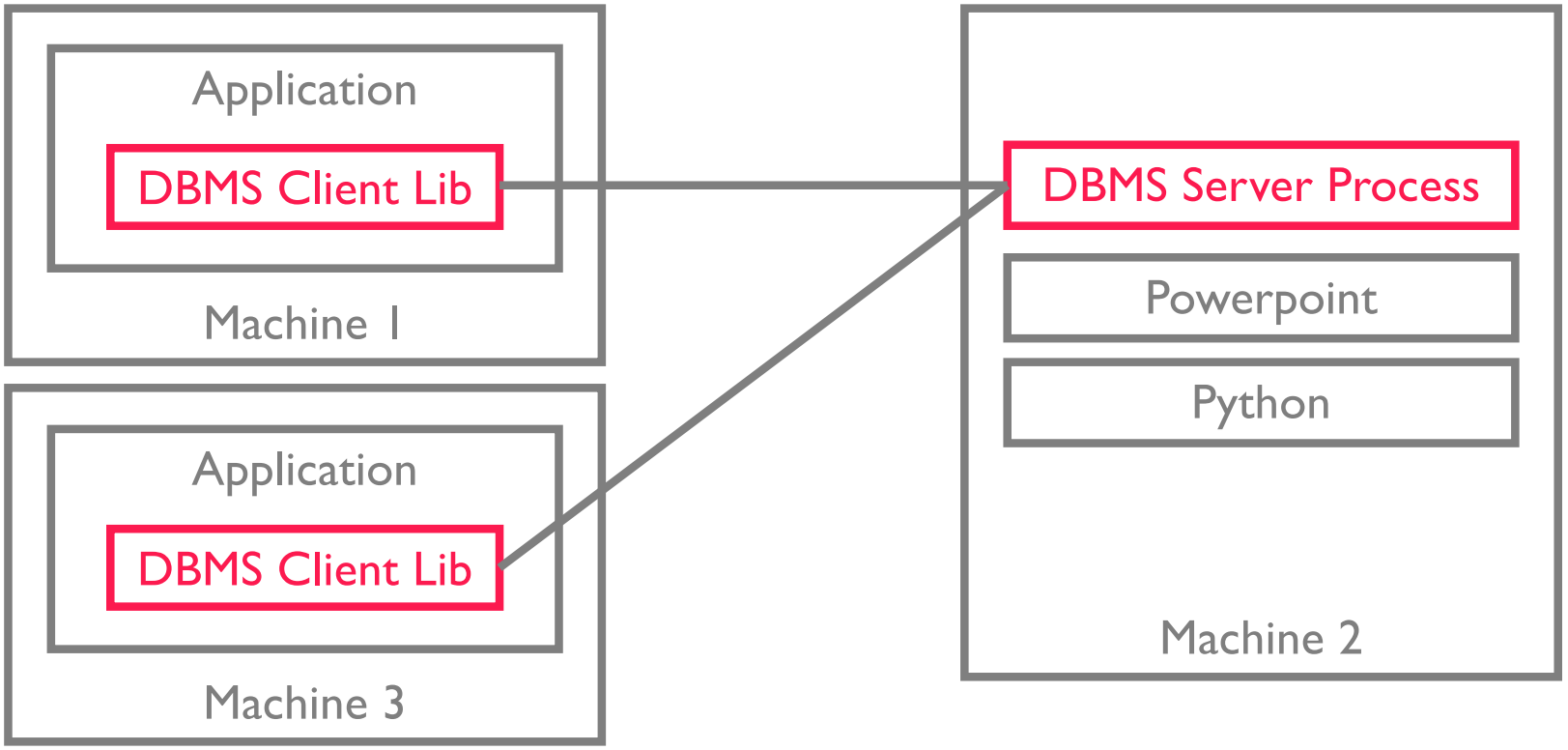
Embedded DBMS

Client-Server DBMS on Different Machines

Main DBMS logic runs on server process

Apps use DBMS client library to connect with server

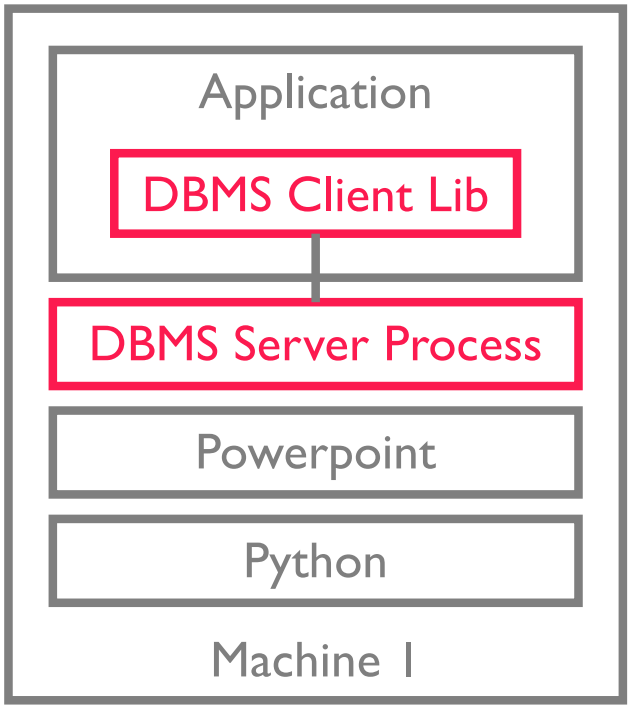
Usually communicate via a network protocol such as TCP



Client-Server DBMS on Same Machine

Server process can run on the same machine as application

Usually communicate via TCP, Interprocess Communication (IPC)



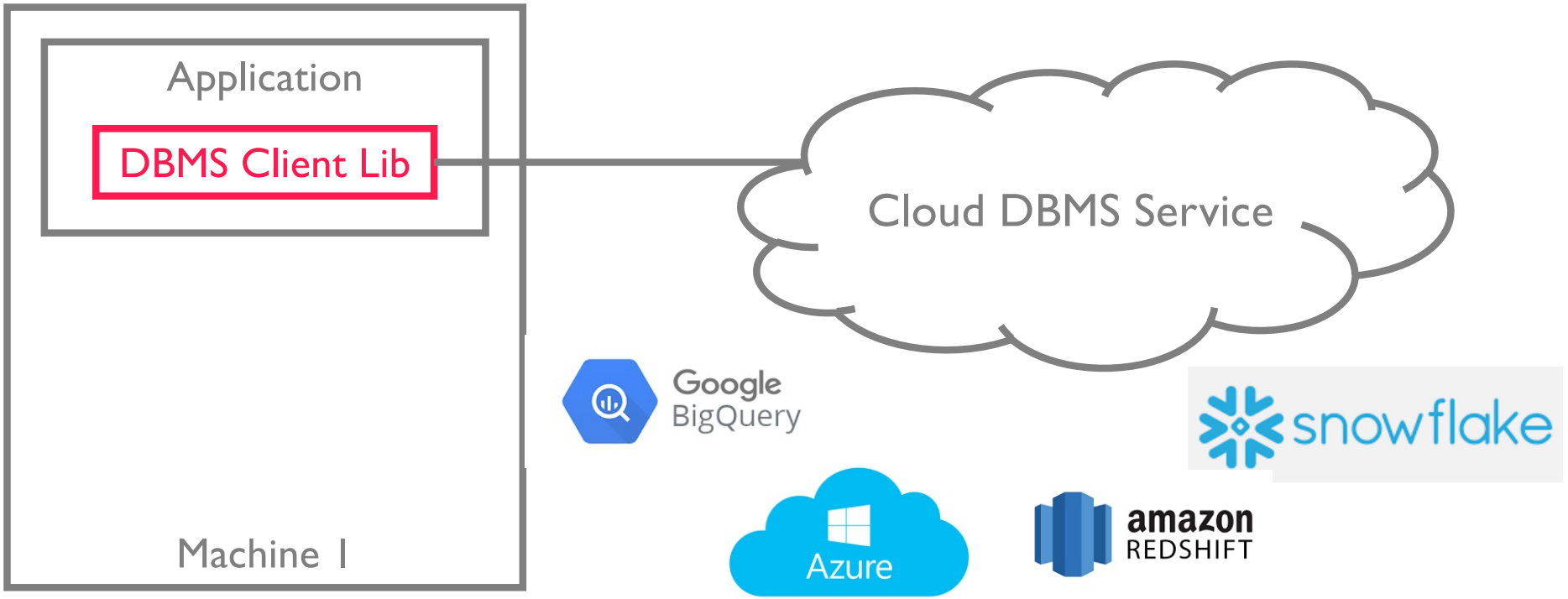
MySQL®

Cloud DBMS

DBMS service managed by someone else

Meant to be auto-scaling (add/remove machines based on load)

Communicate via network protocol e.g., TCP/HTTPS



Embedded DBMS

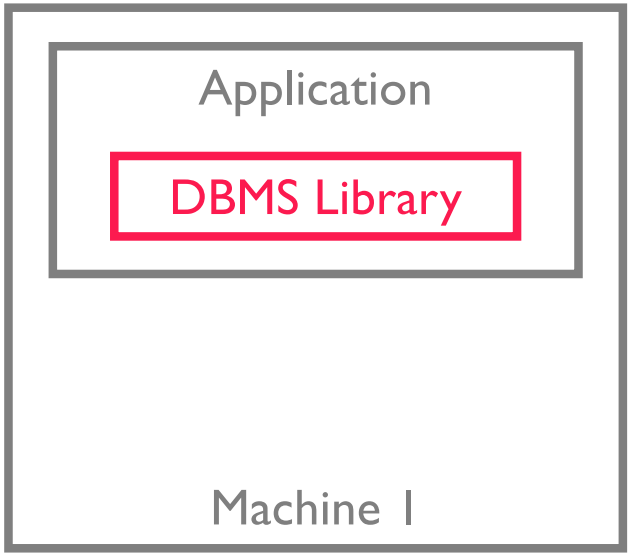
DBMS is a library linked by the application

```
import duckdb, sqlite
```

Runs in same process and memory space

No communication, usually in-memory

Manages storage on local machine

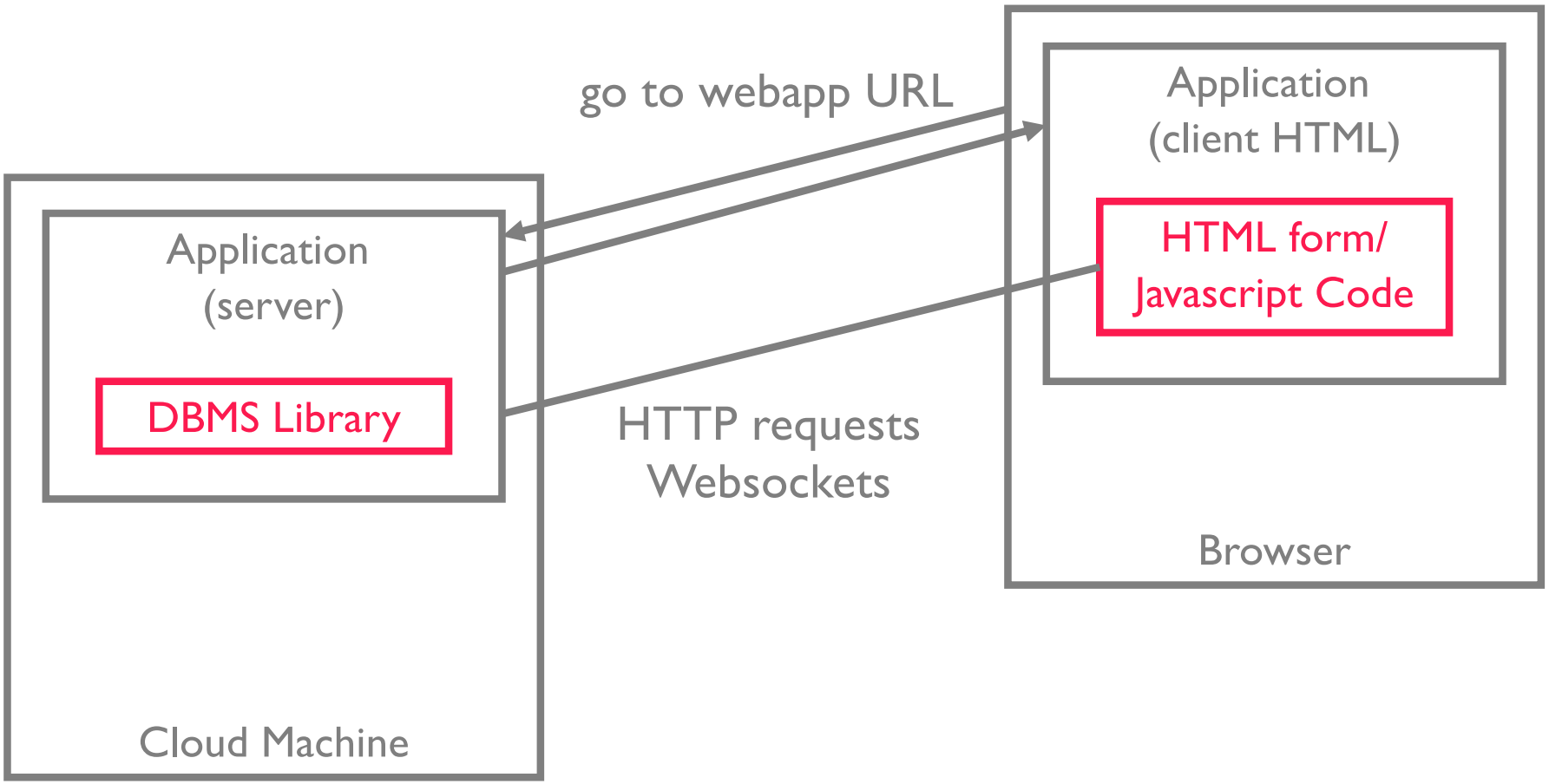


SQLite, DuckDB compiled to Javascript/WASM!
Can run directly in webpage.

Applications in Practice

Web apps usually also have a client (browser) and server.

App's server *uses* DBMS client lib to connect to DBMS Server.



Applications in Practice

Server code for http://.../query

```
@app.route(“/query/”)
def query():
  name = request.form[‘name’]
  cur = conn.execute(“...”)
  row = cur.fetchone()

  data = dict(val=row[‘attr’])
  return render_template(
    “results.html”,
    data=data)
```

Receive inputs & issue query

Package query results into http response

Applications in Practice

Server code for http://.../query

```
@app.route("/query/")
def query():
  name = request.form['name']
  cur = conn.execute("...")
  row = cur.fetchone()

  data = dict(val=row['attr'])
  return render_template(
    "results.html",
    data=data)
```

results.html

```
<div>
  result is {{data}}
</div>
```

Replace placeholder
in template with data

DB API Overview

Library Concerns

1. Establish connection
2. Submit queries/transactions
3. Retrieve results

Impedance Mismatches

1. Types
2. Classes/objects
3. Result sets
4. Functions
5. Constraints

DB API: Engines

URI to refer to a given DBMS engine (like a URL)

credentials location

driver://username:password@host:port/database

```
from sqlalchemy import create_engine
uri1 = “postgresql://localhost:5432/testdb”
#  embedded dbmses have no credentials and location info
uri2 = “sqlite:///testdb.db”
uri3 = “duckdb:///testdb.duckdb”

engine1 = create_engine(uri1)
engine2 = create_engine(uri2)
engine3 = create_engine(uri3)
```

http://docs.sqlalchemy.org/en/rel_1_0/core/engines.html

DB API: Connections

Connect to the DBMS engine

- DBMS Server allocates resources for connection
- Relatively expensive, libs often cache+reuse connections
- Defines scope of a transaction (later in semester)

```
conn1 = engine1.connect()
conn2 = engine2.connect()
```

Close connections when done to avoid leaking resources.

```
conn1.close()
```

DB API: Query Execution

```
conn1.execute(“UPDATE TABLE test SET a = 1”)
conn1.execute(“UPDATE TABLE test SET s = ‘wu’”)


# sqlite
conn1.execute(“SELECT * FROM test WHERE a = ?”, 1)
# postgres
conn1.execute(“SELECT * FROM test WHERE a = %s”, 1)
```

DB API: Query Execution

```
foo = conn1.execute("select * from table")
```

Challenges

- Impedance mismatches
- What is the return type of `execute()`?
- How to pass data between DBMS and host language?
- What about errors? metadata?

(Type) Impedance Mismatch

SQL standard maps between SQL and several languages

Most libraries support primitive types

SQL types	C types	Python types
CHAR(20)	char[20]	str
INTEGER	int	int
SMALLINT	short	int
REAL	float	float

What about complex objects { x: ‘ l ’, y: ‘hello’ }

(Class) Impedance Mismatch

Programming languages usually have classes

Want objects to persist in DBMS

```
user.name = “Dr Seuss”
user.job  = “writer”
```

```
class User { … }
class Employee extends User { … }
class Salaries {
    Employee worker;
    …
}
```

Object Relational Mappings (ORMs)
try to provide this abstraction

ORM: classes that magically sync with DBMS

Base is a special class defined by ORM

mimics CREATE TABLE in Python

We will NOT use ORMs for project 1

```
class User(Base):
  __tablename__ = "user_account"
  id        = Column(Integer, primary_key=True)
  name      = Column(String)
  addrs     = relationship(
    "Address", back_populates="user", cascade="all")

class Address(Base): ...
```

(results) Impedance Mismatch

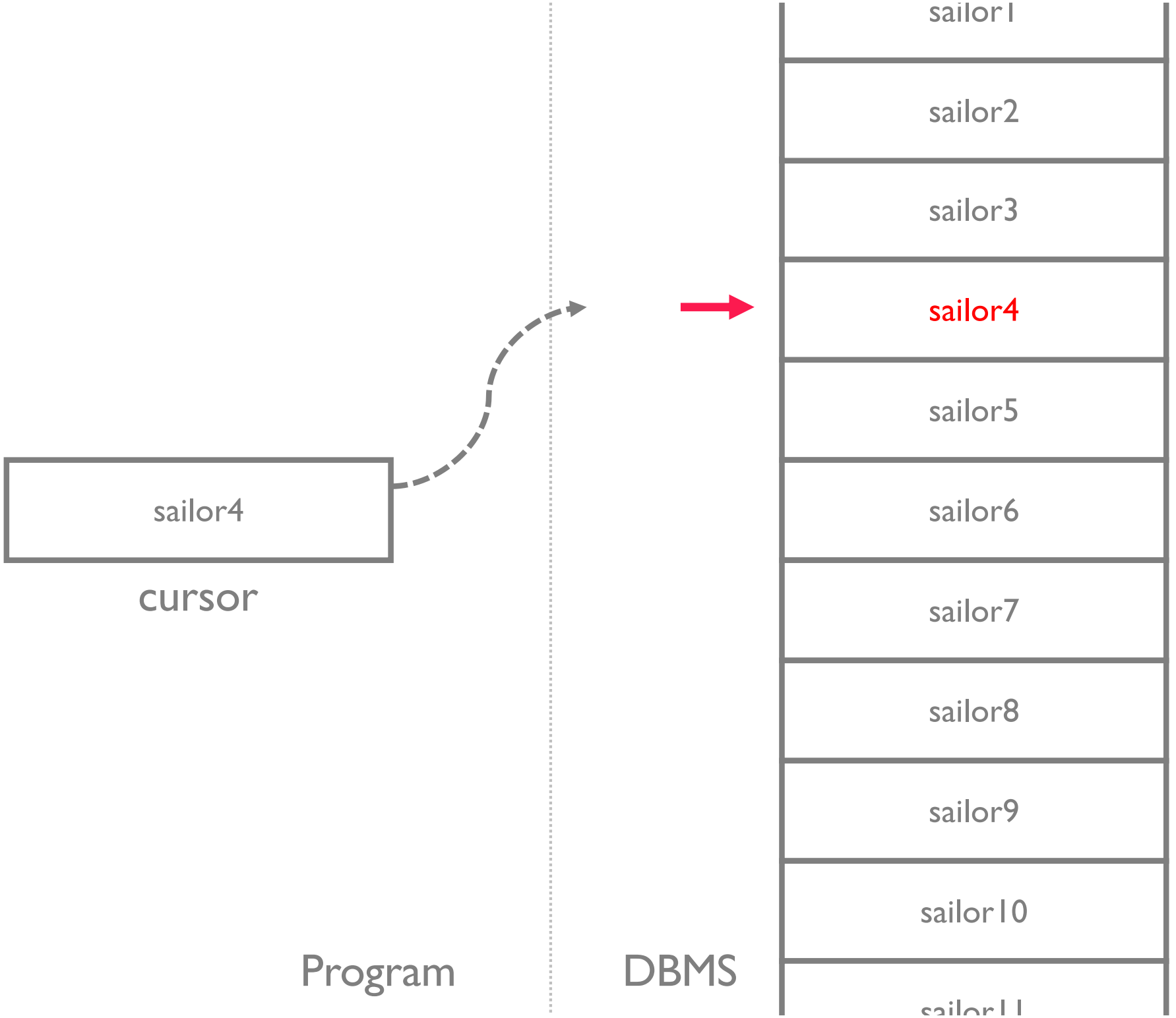
What is the type of `table` below?

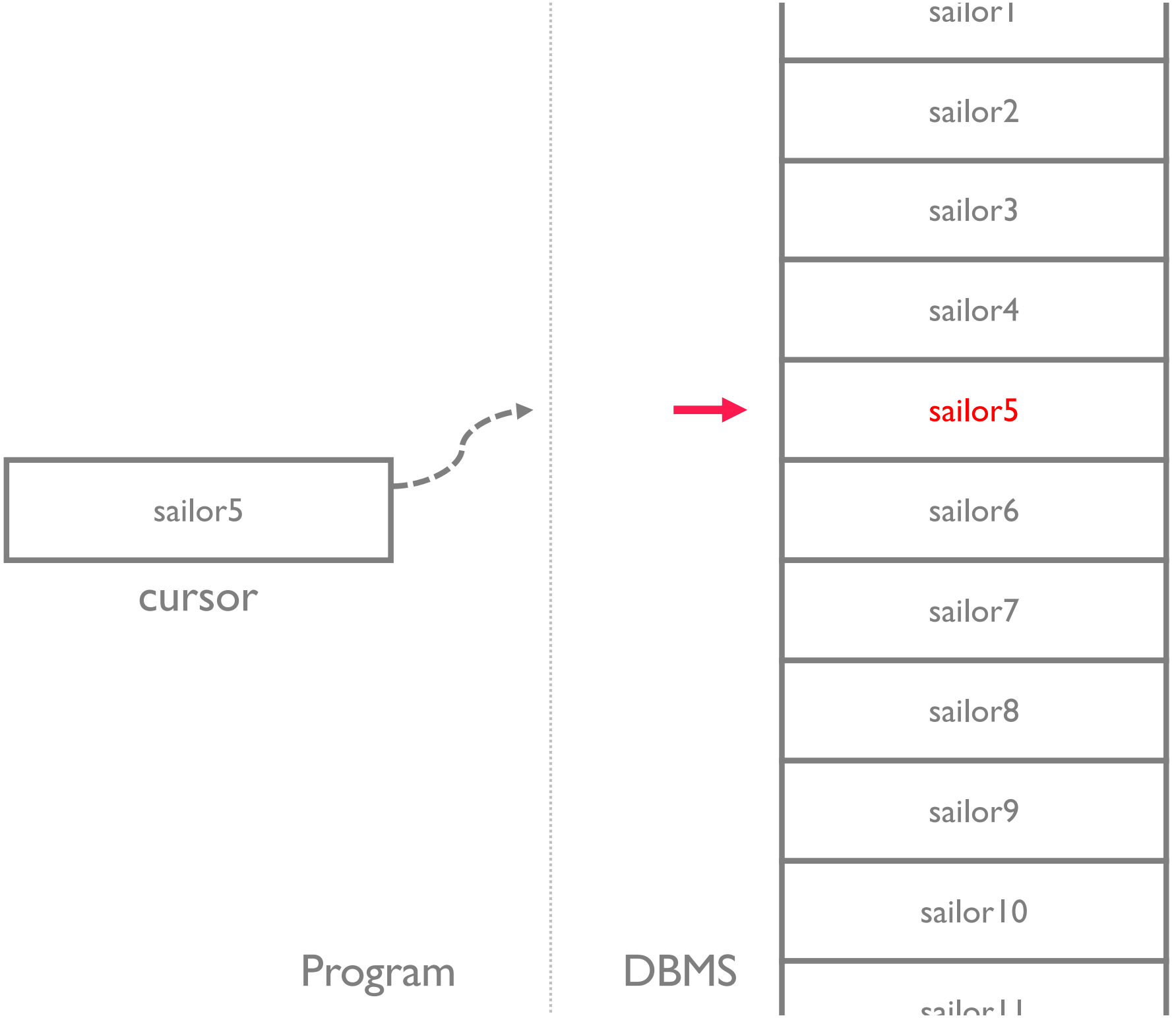
```
table = conn.execute("SELECT * FROM big_table")
```

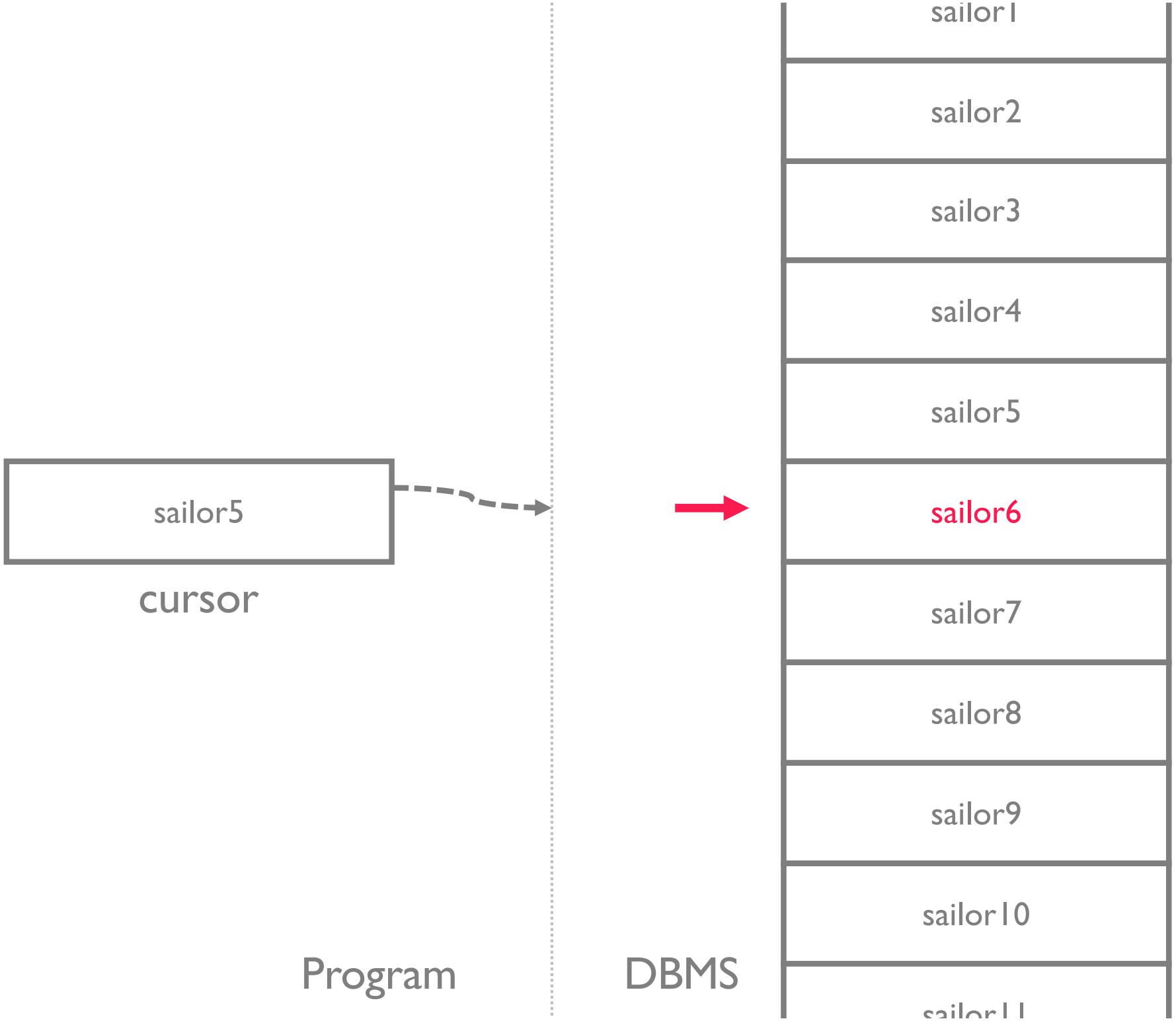
Cursor over the Result Set (similar to an iterator)

Note: relations are unordered!

- Cursors have no ordering guarantees
- Use ORDER BY to ensure an ordering


sailor1
sailor2
sailor3
sailor4
sailor5
sailor6
sailor7
sailor8
sailor9
sailor10
sailor11
sailor4
cursor
Program
DBMS


sailor1
sailor2
sailor3
sailor4
sailor5
sailor6
sailor7
sailor8
sailor9
sailor10
sailor11
sailor5
cursor
Program
DBMS


sailor1
sailor2
sailor3
sailor4
sailor5
sailor6
sailor7
sailor8
sailor9
sailor10
sailor11
sailor5
cursor
Program
DBMS

(results) Impedance Mismatch

Cursor similar to an iterator (`next()` calls)

```
cursor = conn.execute("SELECT * FROM T")
```

Core cursor attributes/methods (names may differ)

```
rowcount
attributes()
prev()
next()
get(idx)
```

(results) Impedance Mismatch

Cursor similar to an iterator (`next()` calls)

```
cursor = conn.execute("SELECT * FROM T")
cursor.rowcount()  # 1000000
cursor.fetchone()  # (0, 'foo', …)
for row in cursor: # iterate over the rest
   print(row)
```

Actual Cursor methods vary depending on implementation

(functions) Impedance Mismatch

What about functions?

```
def add_one(val):
   return val + 1

conn1.execute(“SELECT add_one(1)”)
# doesn’t work :(
```

Would need to embed a language runtime into DBMS

Many DBMSes support runtimes e.g., python

Register `add_one()` as User Defined Function (UDF)

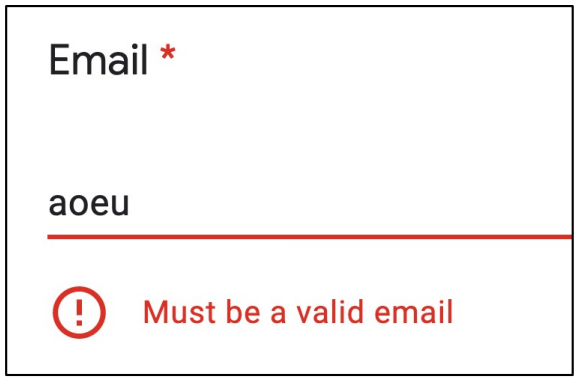
(constraints) Impedance Mismatch

DB Constraints often duplicated throughout program

Application checks are for user experience, not correctness

JS

```
email = get_email_input();
if (/@/.test(email))
  error("must be a valid email");
```



DBMS

```
CREATE TABLE Users (
    email text CHECK(email ~ '@')
)
```

(constraints) Impedance Mismatch

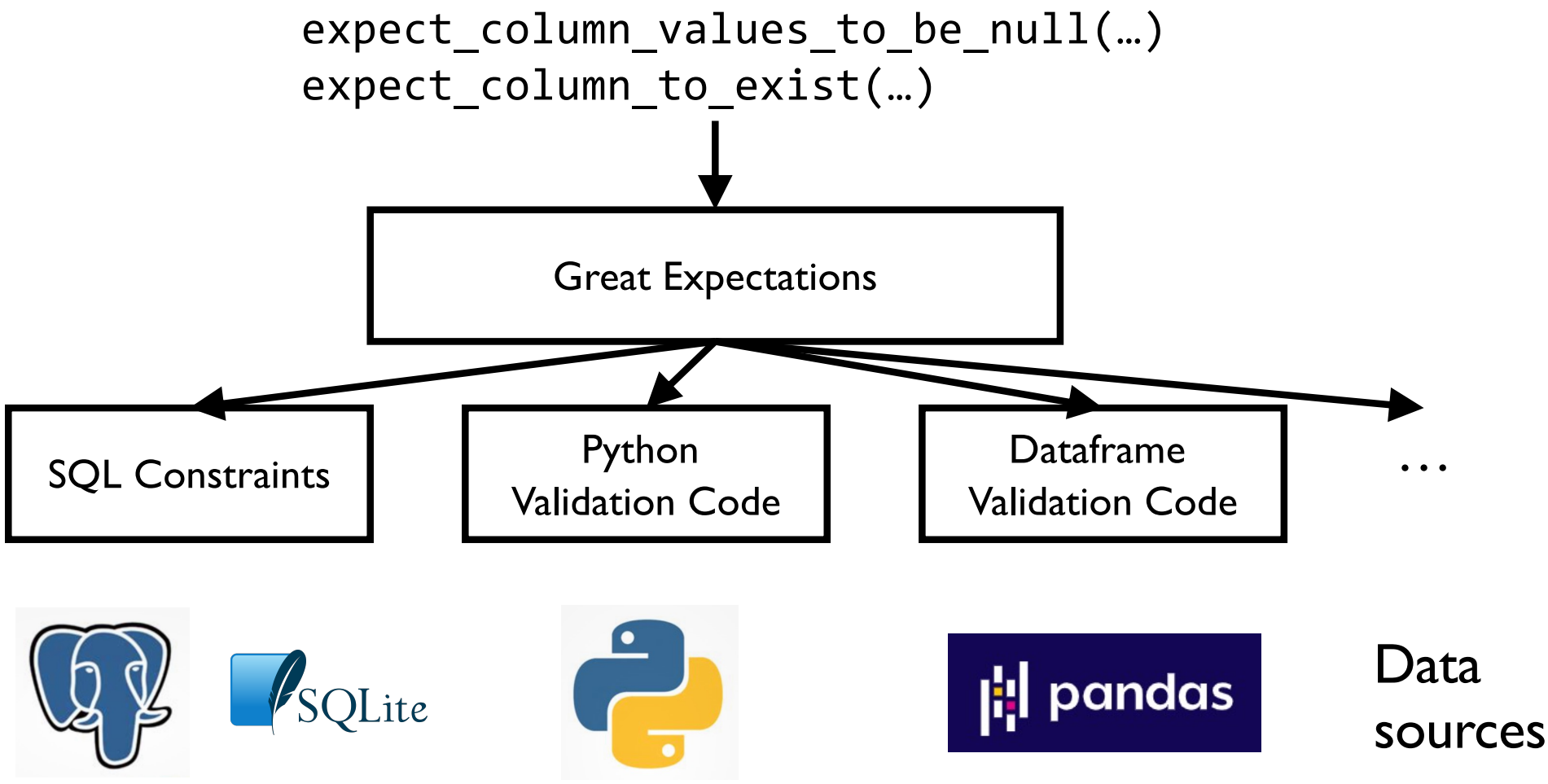
ORMs let you define basic constraints

```
class Person(models.Model):
    ...
    first_name = models.CharField(max_length=30)
    last_name  = models.CharField(max_length=30, null=True)
```

```
CREATE TABLE myapp_person (
    ...
    first_name varchar(30) NOT NULL,
    last_name  varchar(30)
);
```

(constraints) Impedance Mismatch

Third-party constraint libraries e.g., Great Expectations



Modern Database APIs

Examples: DryadLinq, SparkSQL

DBMS executor in same language (dotNET, Spark) as app code

Tricky:

- what happens to language impedance?
- what happens to exception handling?
- what happens to host language functions?

```
val lines  = spark.textFile(“logfile.log”)
val errors = lines.filter(_ startswith “Error”)
val msgs   = errors.map(_.split(“\t”)(2))

msgs.filter(_ contains “foo”).count()
```

What to Understand

Goals and flaws in Embedded SQL

Client-server vs embedded DBMSes

DBAPI components, cursors

Impedance mismatch: examples and possible solutions