L6
APIs and DB Security

Eugene Wu

Topics

Interfacing with applications

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

Security

- SQL injection, the world's most popular DB "hack"
- Access controls and GRANT
- Encryption

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

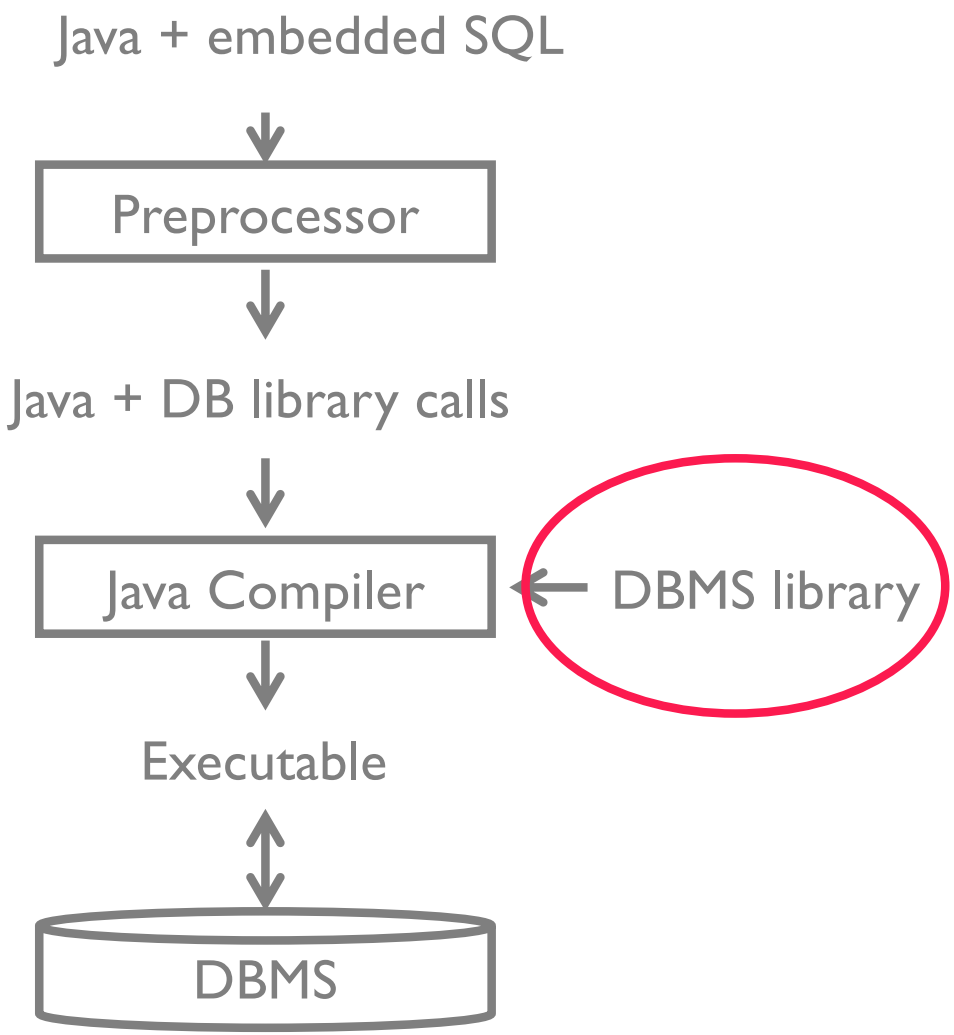
Extend host language (python) with SQL syntax

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

goes through a preprocessor

Compiled into program that interacts with DBMS directly

Embedded SQL




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

   EXEC SQL select * …

} else {
...
```

What does a library need to do?

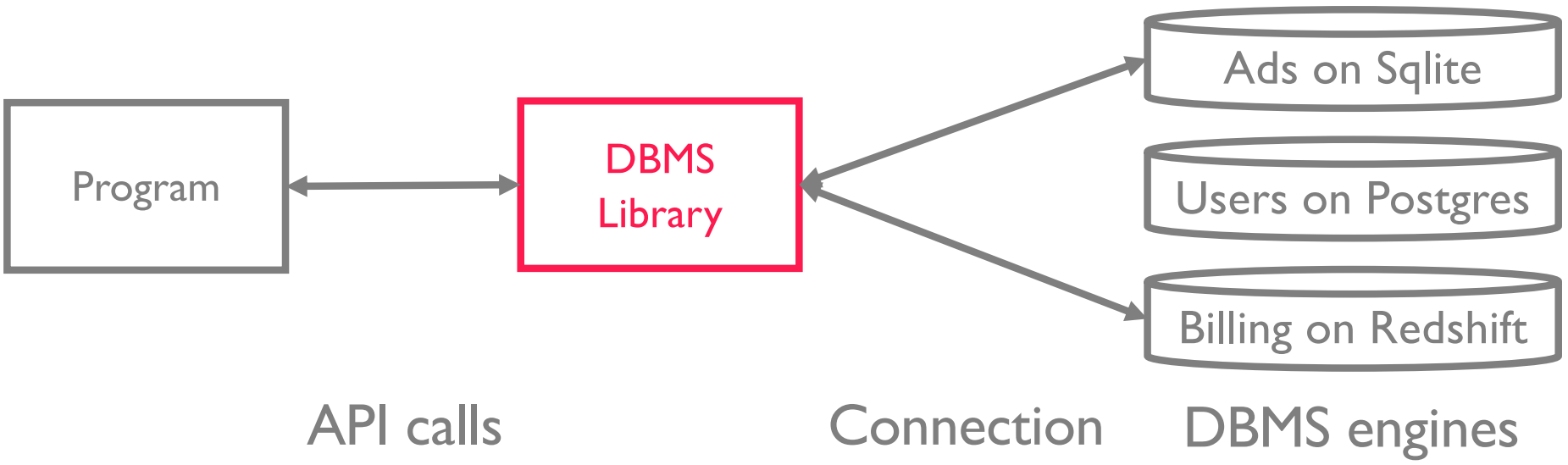
Single interface to possibly multiple **DBMS engines**

Connect to a database

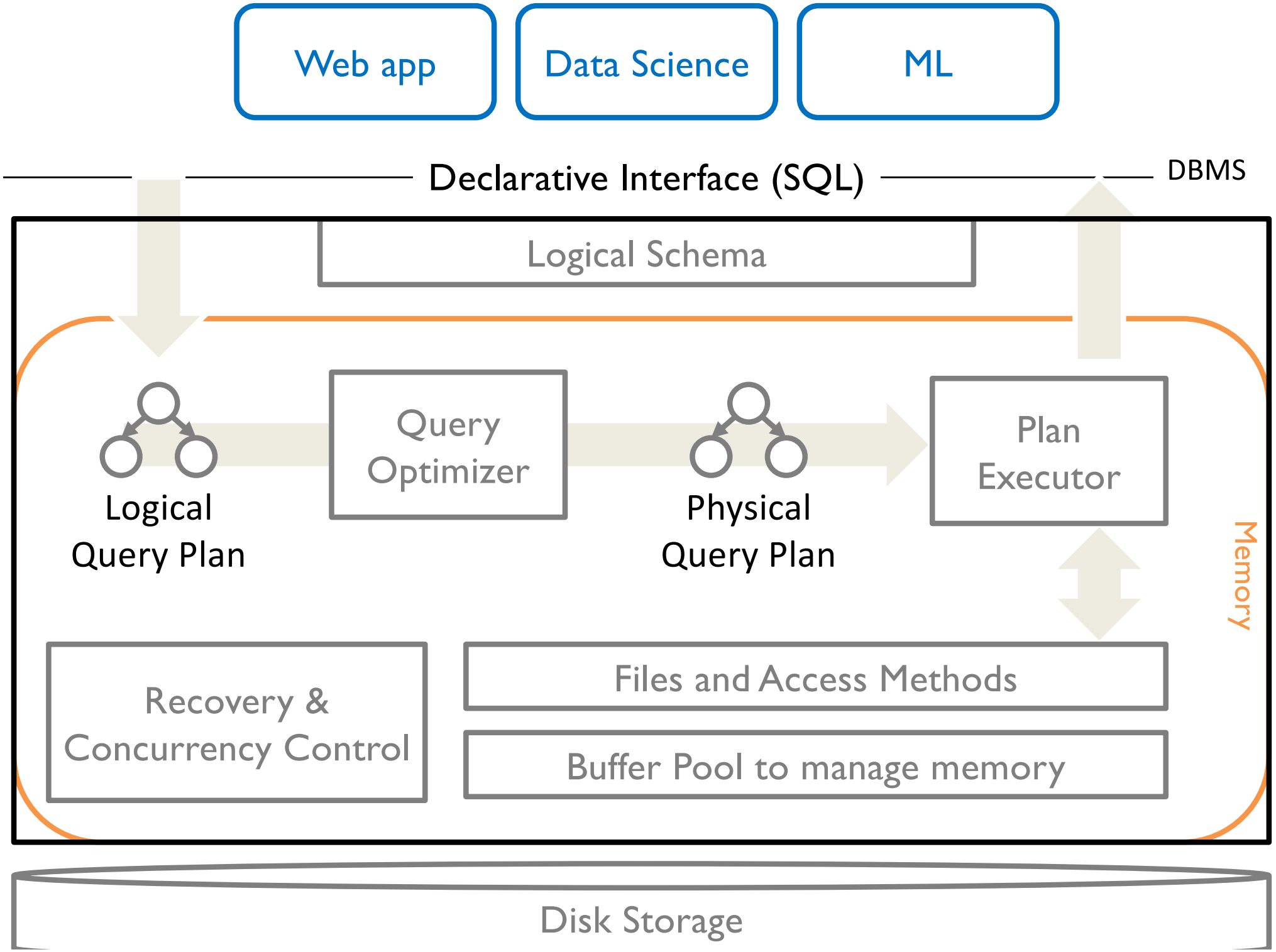
Manage transactions (later)

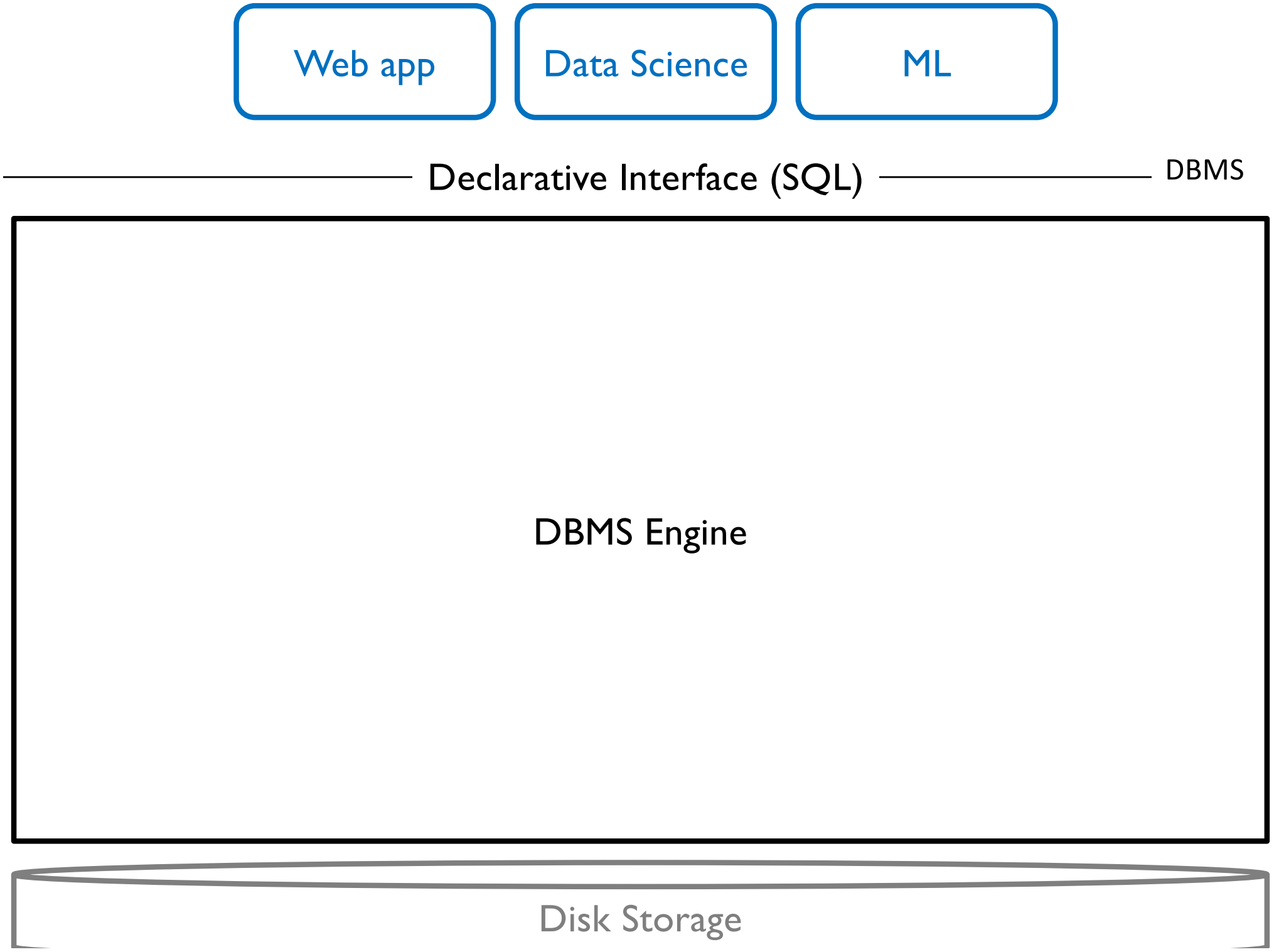
Map objects between host language and DBMS

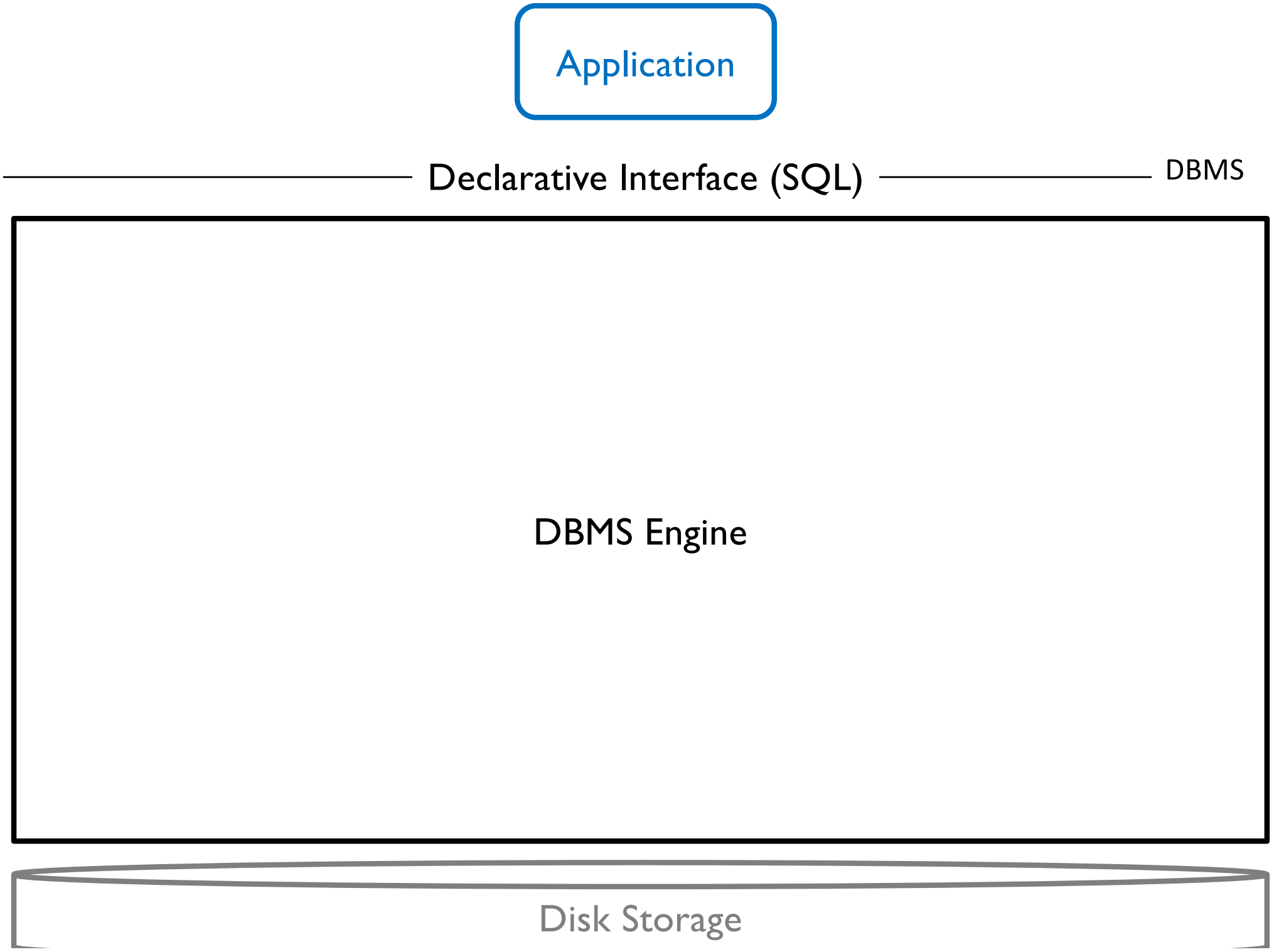
Manage query results

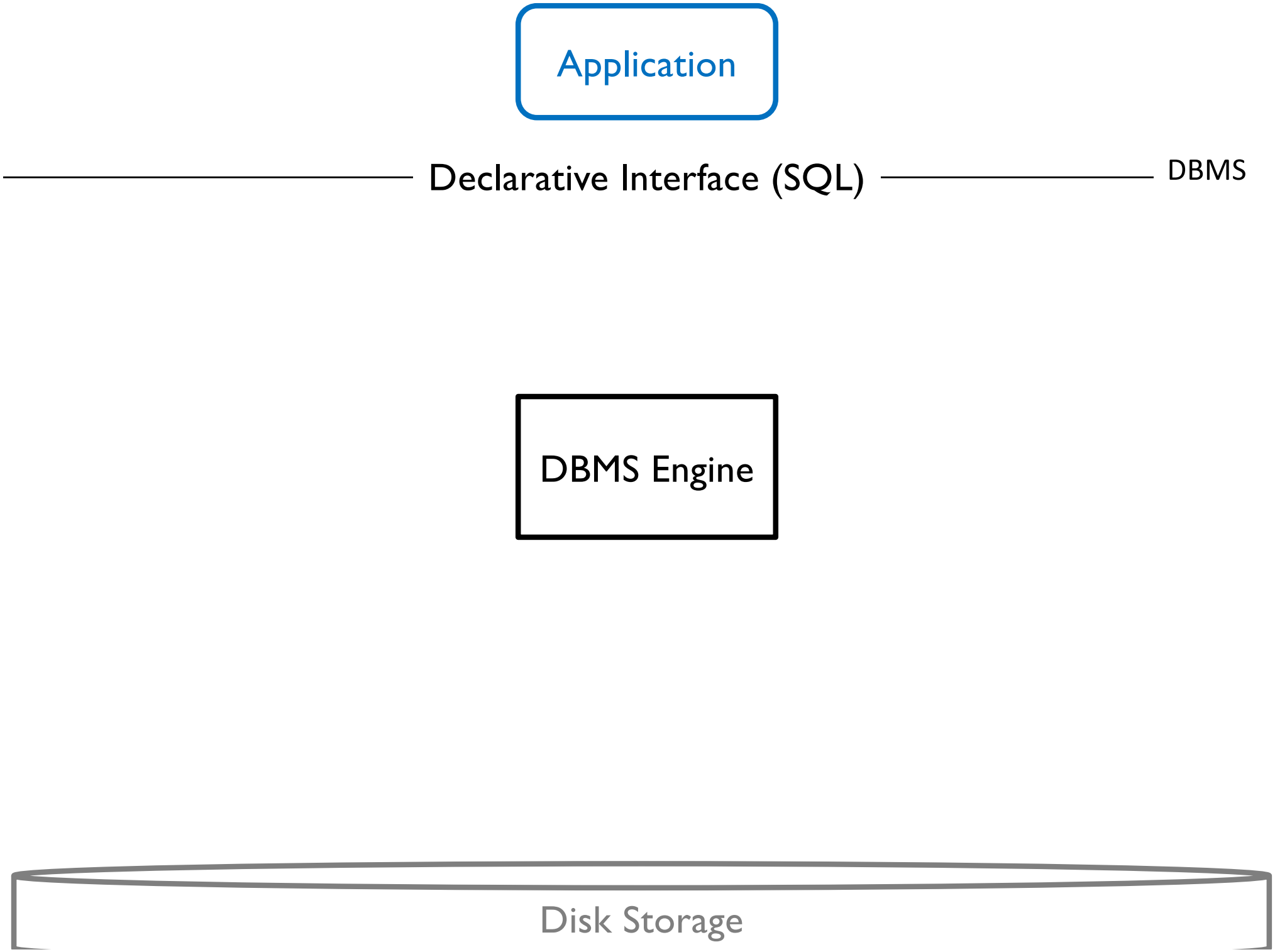


DBMS Engines

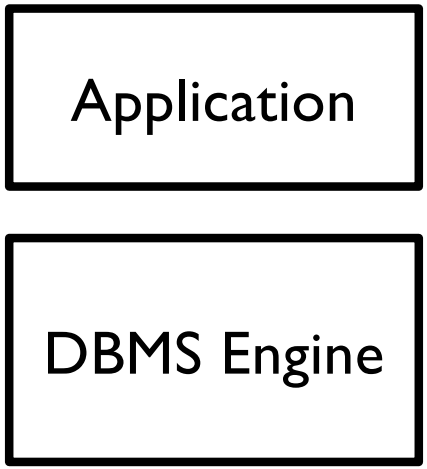

Web app
Data Science
ML
Declarative Interface (SQL)
DBMS
Logical Schema
Logical
Query Plan
Query
Optimizer
Physical
Query Plan
Plan
Executor
Memory
Recovery &
Concurrency Control
Files and Access Methods
Buffer Pool to manage memory
Disk Storage


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

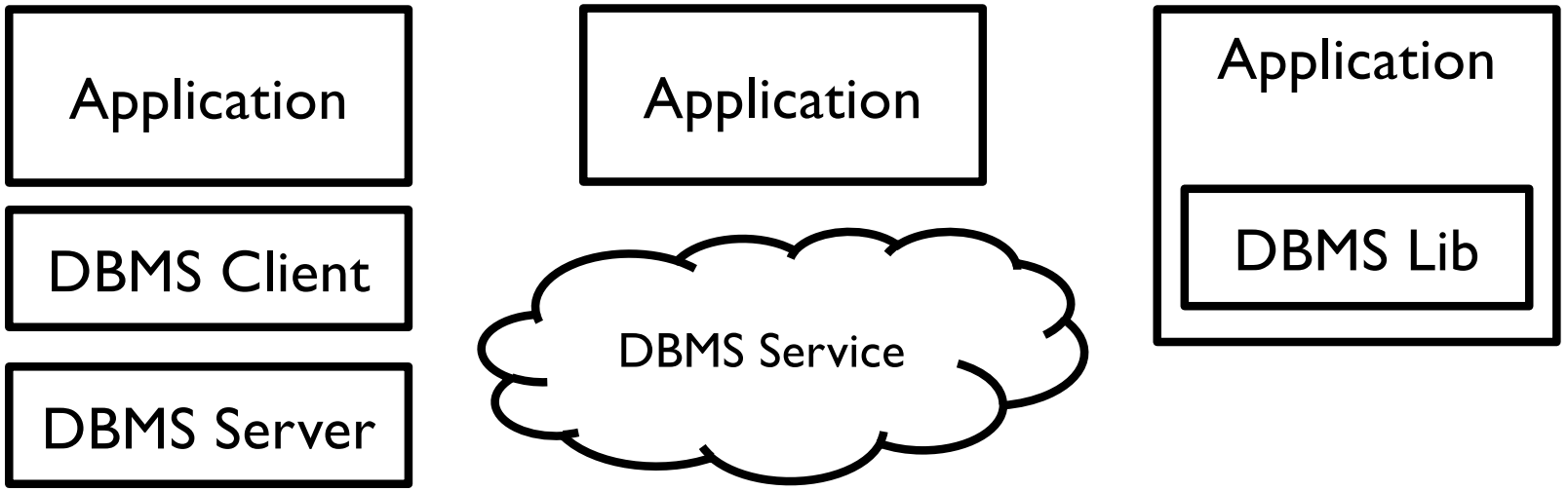

Application
Declarative Interface (SQL)
DBMS
DBMS Engine
Disk Storage


Application
Declarative Interface (SQL)
DBMS
DBMS Engine
Disk Storage

DBMS Engines



DBMS Engines: 3 Types



Client-Server DBMS

Cloud DBMS

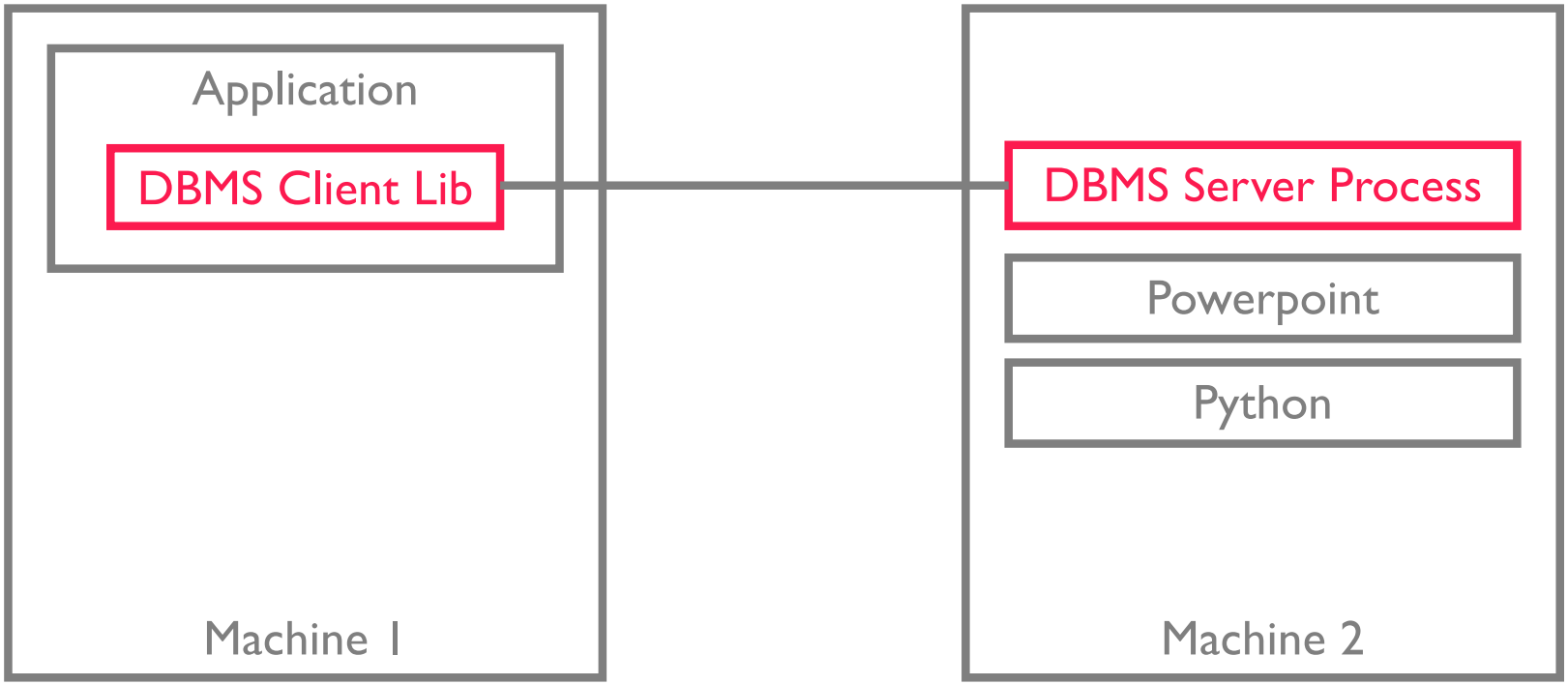
Embedded DBMS

Client-Server DBMS on Different Machines

Main DBMS logic runs on server process

Apps use DBMS client library to connect with server

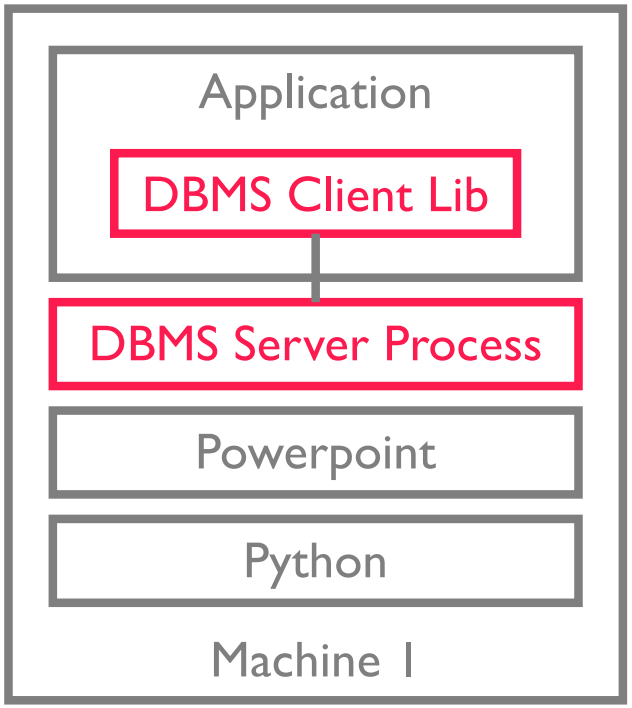
Usually communicate via a network protocol such as TCP



Client-Server DBMS on Same Machine

Server process can run on the same machine as application

Usually communicate via TCP, Interprocess Communication (IPC)



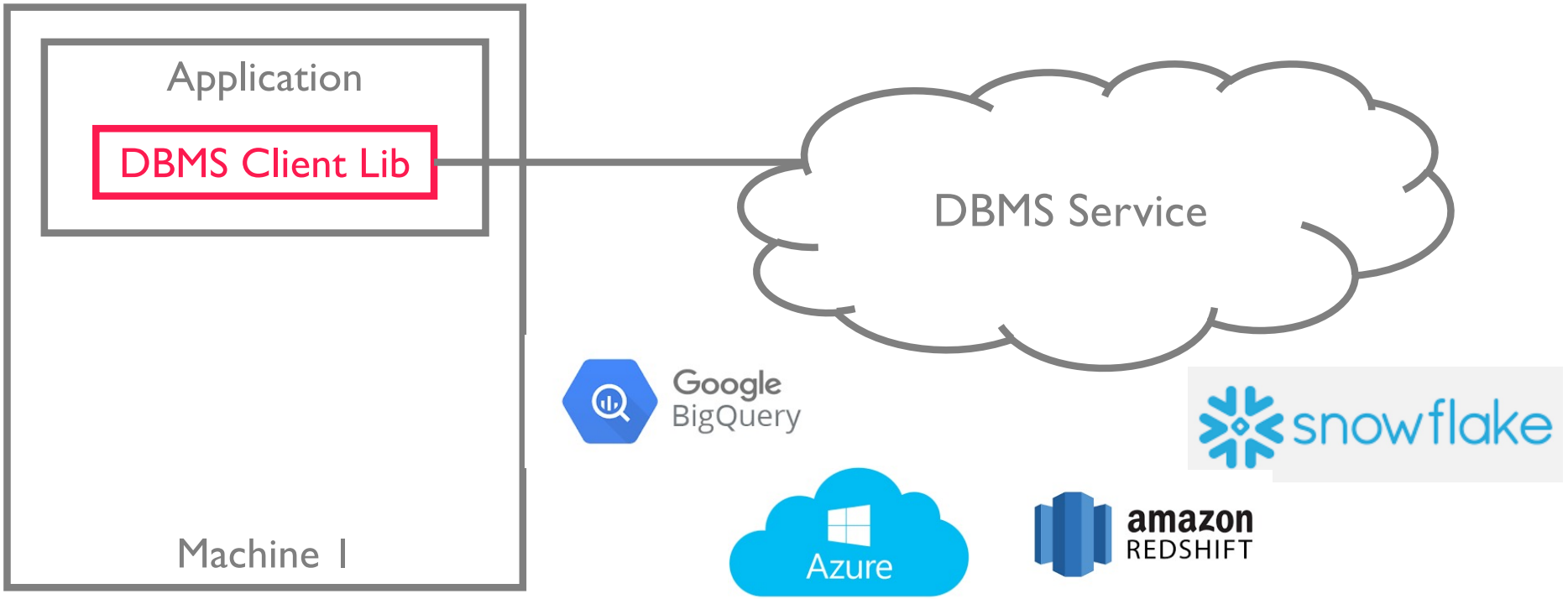
MySQL®

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



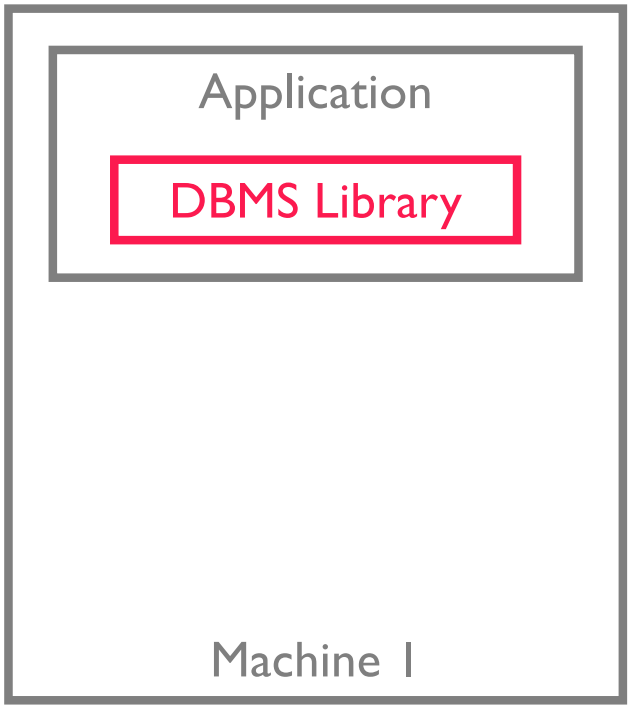
Embedded DBMS

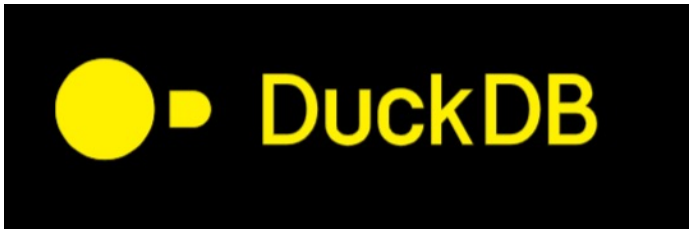
DBMS is a library linked by the application

```
import duckdb, sqlite
```

Runs in same process and memory space

No communication, usually in-memory



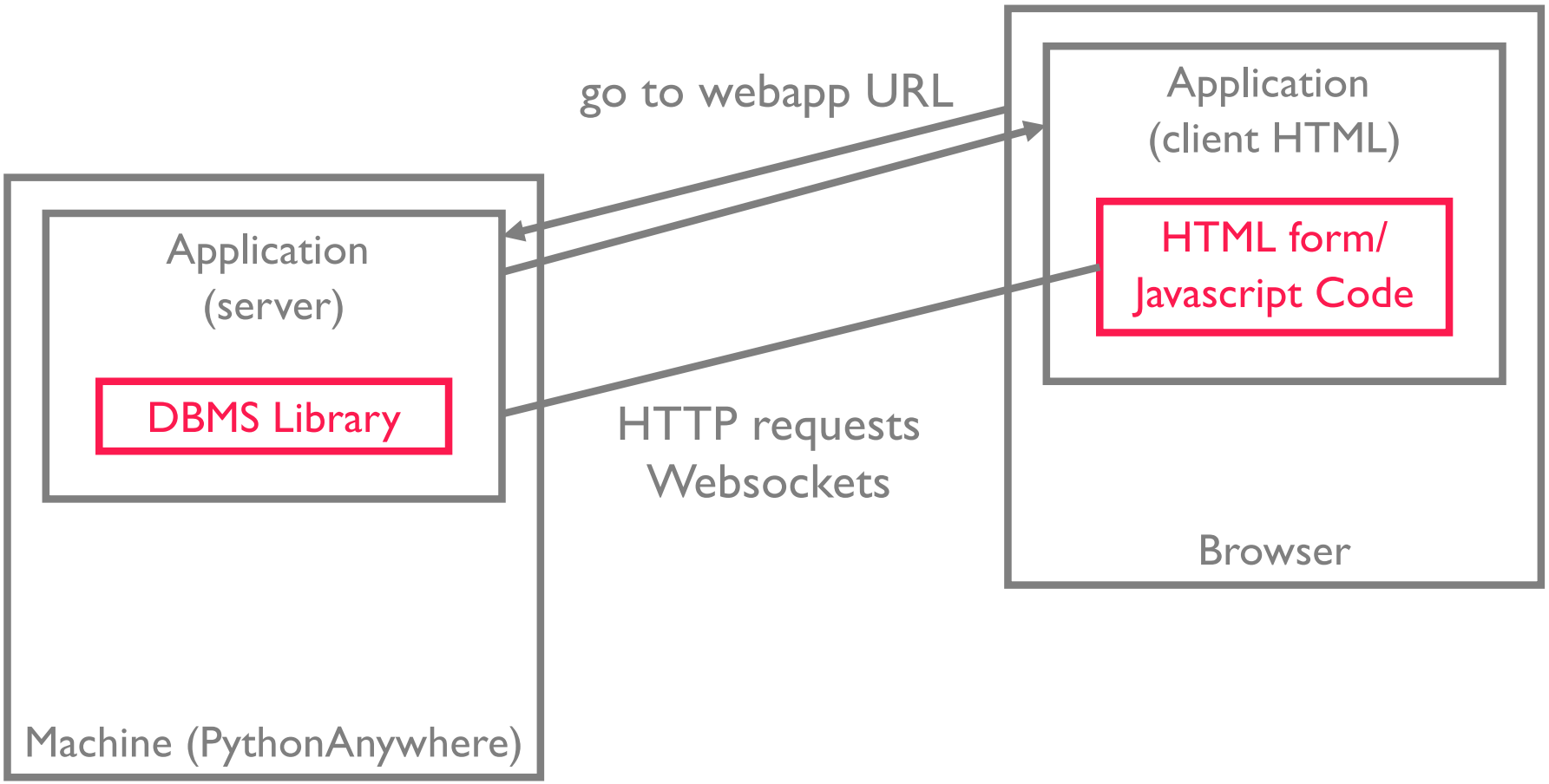


SQLite, DuckDB compiled to javascript!
Can run directly in webpage.

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)

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

```
from sqlalchemy import create_engine
uri1 = “postgresql://localhost:5432/testdb”
#  embedded dbmses have no host:port
uri2 = “sqlite:///testdb.db”

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

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 big_table")
```

Challenges

- Impedance mismatches
- What is the return type of `execute()`?
- How to pass data between DBMS and host language?
- Can we only pass data between DBMS and host language?

(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): ...
```

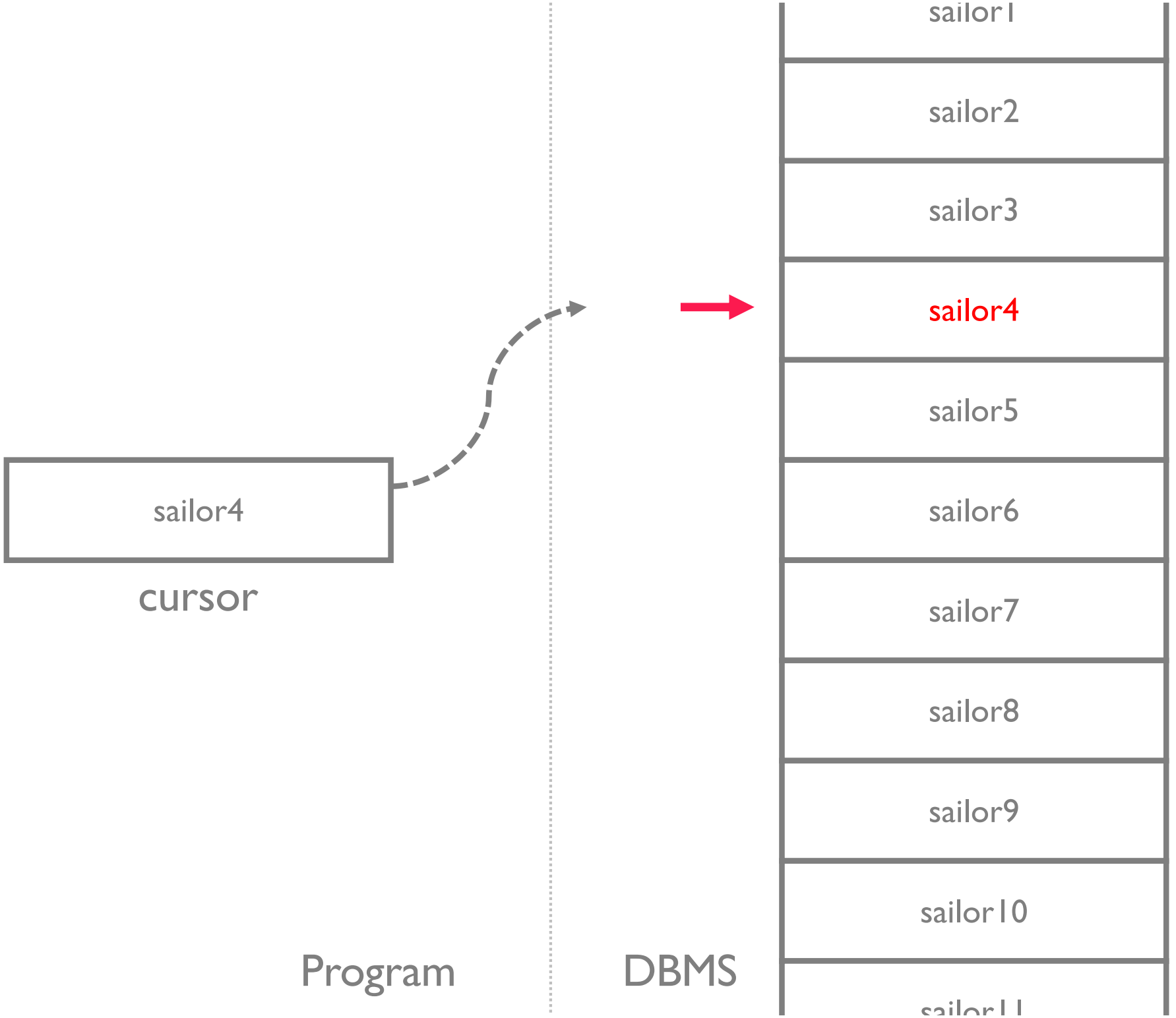
(results) Impedance Mismatch

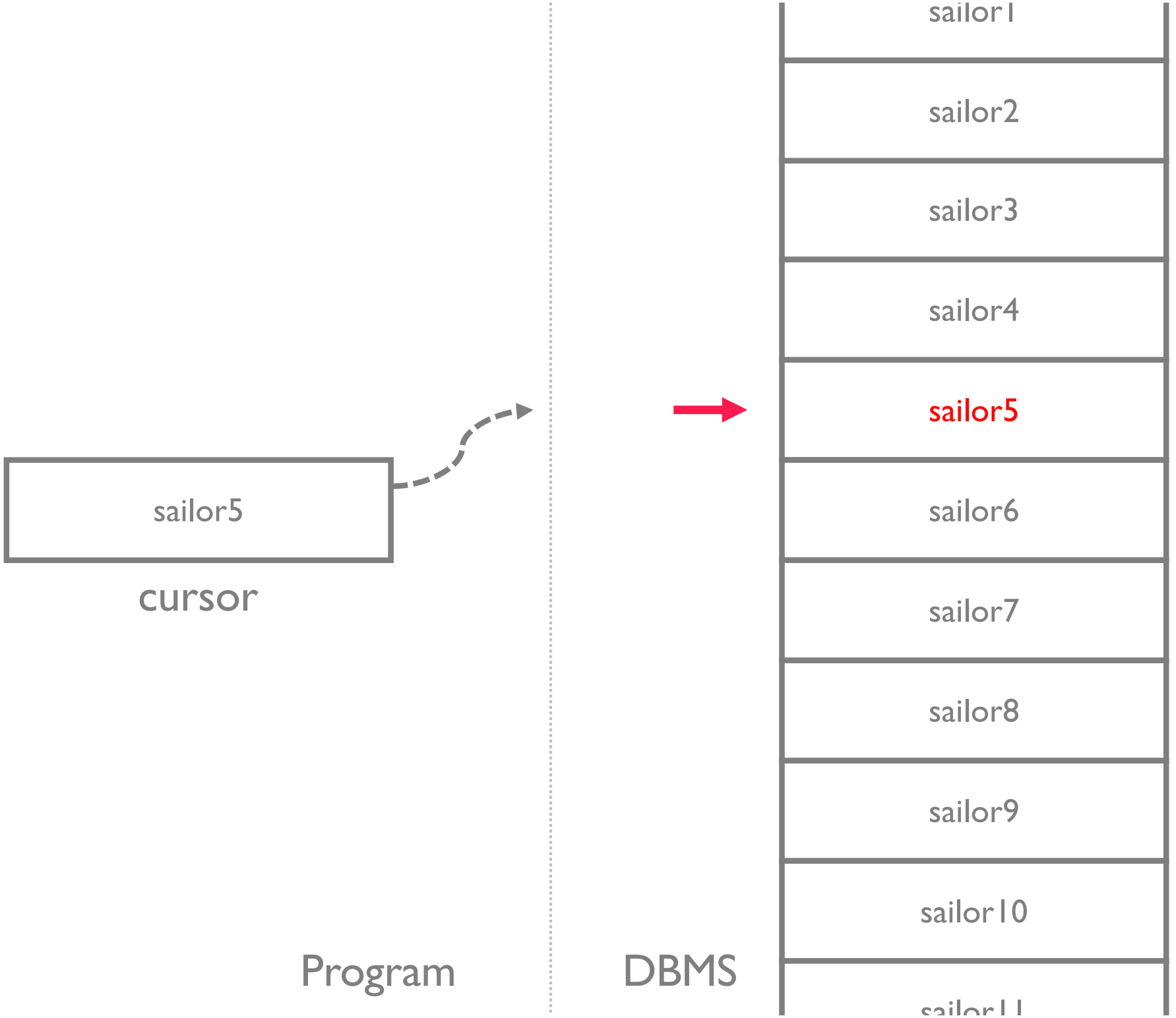
What is the type of `table` below?

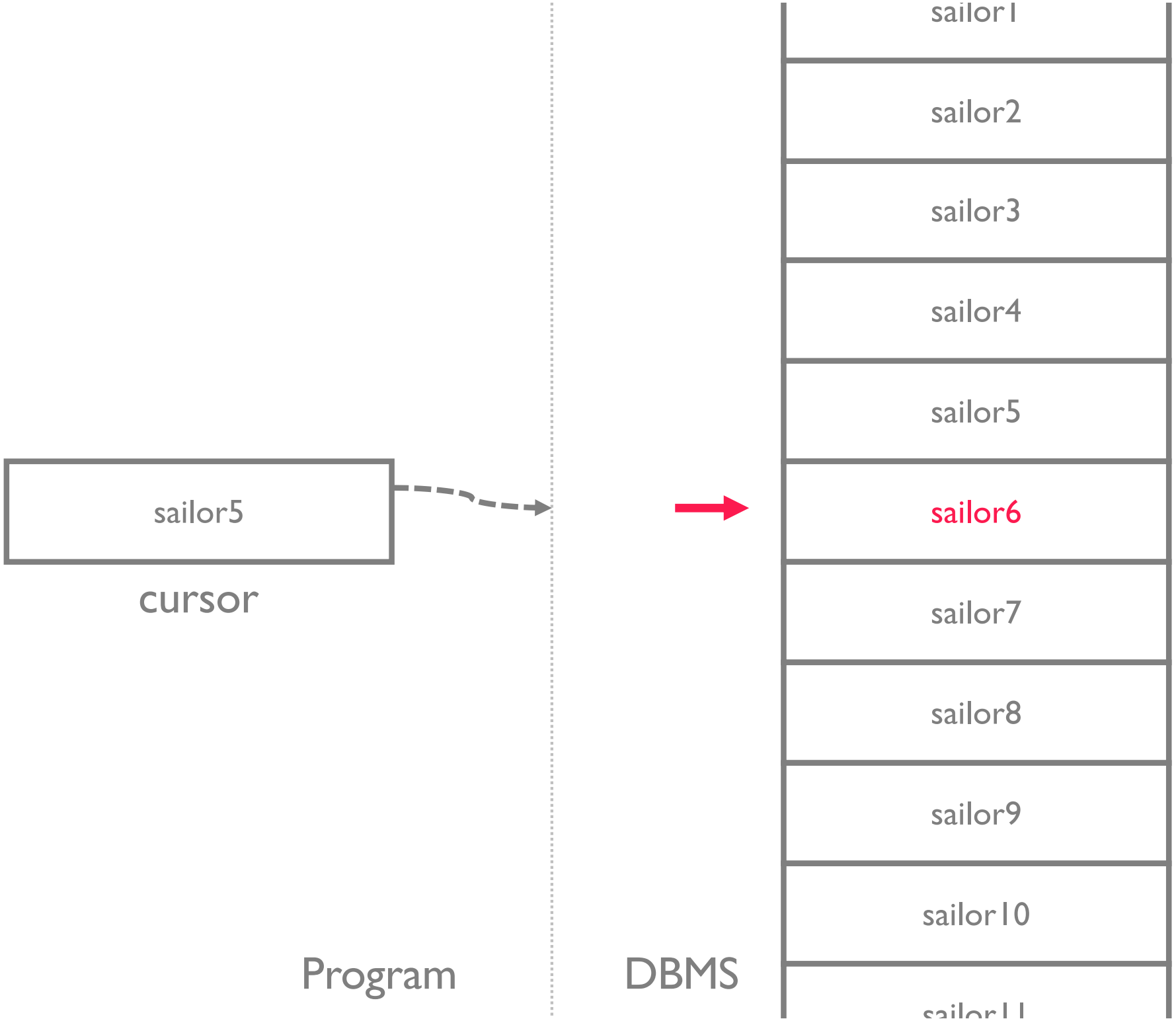
```
table = conn.execute("SELECT * FROM big_table")
```

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 =`

```
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)”)
```

Would need to embed a language runtime into DBMS

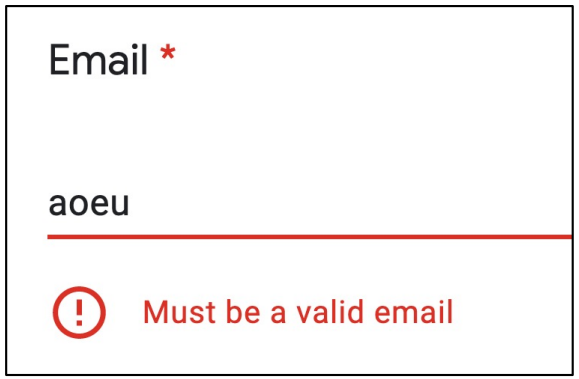
- Many DBMSes support runtimes e.g., python
- Can register User Defined Functions (UDFs)

(constraints) Impedance Mismatch

DB Constraints often duplicated throughout program

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

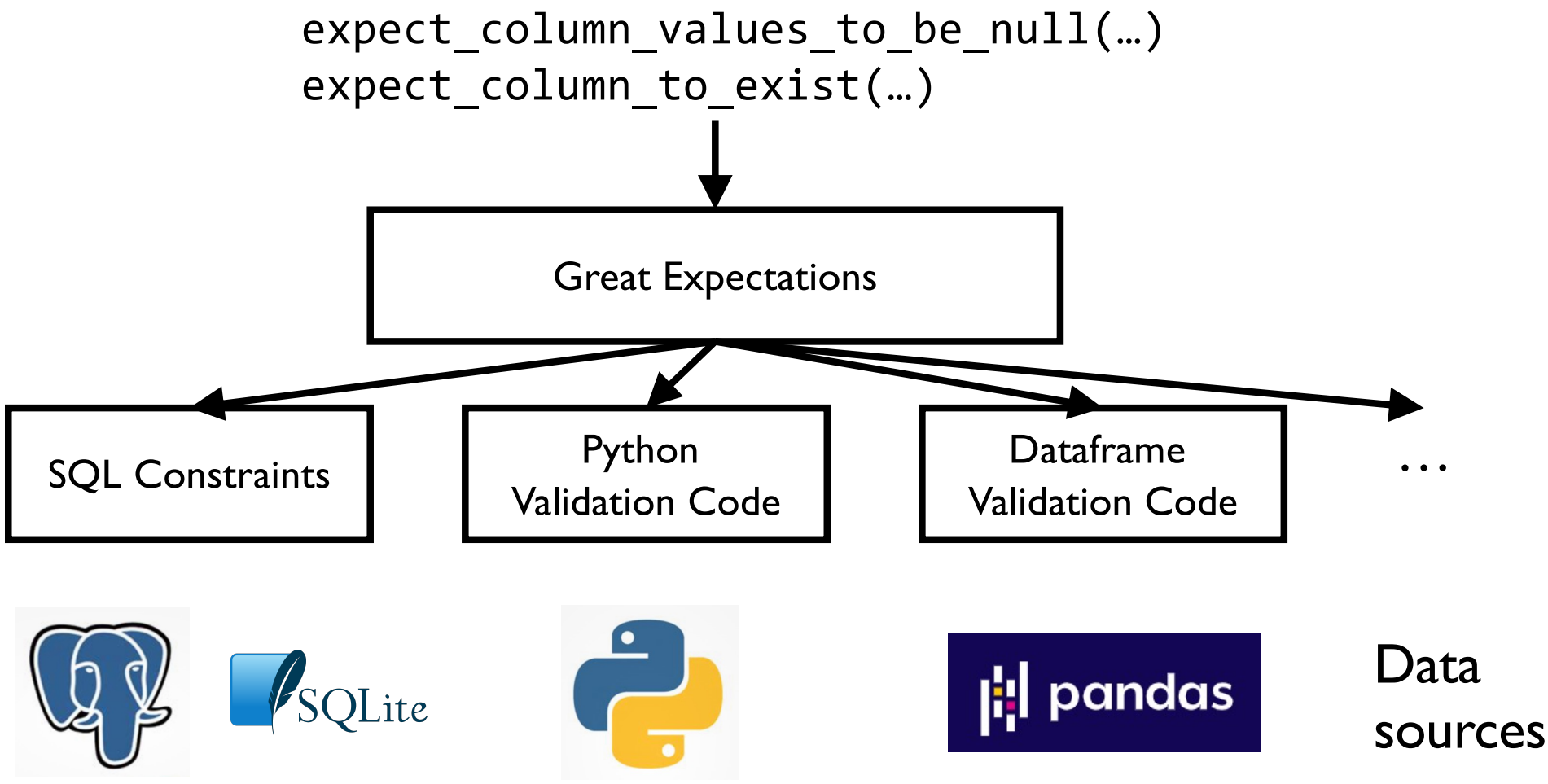
ORMs let you define basic constraints

```
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()
```

Security

SQL Injection

Privacy vs Security

Access Controls and GRANT

Encryption

SQL Injection

Pass *sanitized* values to the database

```
args = ('Dr Seuss', '40')
conn1.execute(
    "INSERT INTO users(name, age) VALUES(%s, %s)",
    args)
```

Pass in a tuple of query arguments

DBAPI library will *properly escape* input values

Most libraries support this

Never construct raw SQL strings

SQL Injection

Why pass values using query parameters?

```
name = “eugene”

conn1.execute(
  “SELECT * FROM users WHERE name=%s”, name)
```

```
conn1.execute(
  “SELECT * FROM users WHERE name=‘{name}’”.format(name=name))

   SELECT * FROM users WHERE name=‘eugene’
```

SQL Injection

Why pass values using query parameters?

```
name = "eugene';\nDELETE * FROM users;--"

conn1.execute(
  "SELECT * FROM users WHERE name=%s", name)
```

```
conn1.execute(
  "SELECT * FROM users WHERE name='{name}'".format(name=name))
```

```
SELECT * FROM users WHERE name='eugene';
DELETE * FROM users;
--'
```

FEATURES | By Joseph Cox | Nov 20 2015, 8:00am

The History of SQL Injection, the Hack That Will Never Go Away

Over 15 years after it was first publicly disclosed, SQL injection is still the number one threat to websites.

https://motherboard.vice.com/en_us/article/aekzez/the-history-of-sql-injection-the-hack-that-will-never-go-away

sql injection

FILTER

Running an SQL Injection Attack - Computerphile

Computerphile 1.6M views • 2 years ago

Just how bad is it if your site is vulnerable to an **SQL Injection**? Dr Mike Pound shows us how they work. Cookie Stealing: ...

SQL Injection Attack Tutorial (2019)

HackHappy • 76K views • 8 months ago

SQL Injection attacks are still as common today as they were ten years ago. Today I'll discuss what are SQLi and how you can ...

CC

SQL Injection Basics Demonstration

Imperva • 360K views • 9 years ago

Imperva presents an educational video series on Application and Database Attacks in High Definition (HD)

Hacking Websites with SQL Injection - Computerphile

Computerphile 1.4M views • 5 years ago

Websites can still be hacked using **SQL injection** - Tom explains how sites written in PHP (and other languages too) can be ...

CC

DEFCON 17: Advanced SQL Injection

Christiaan008 • 220K views • 8 years ago

Speaker: Joseph McCray Founder of Learn Security Online **SQL Injection** is a vulnerability that is often missed by web application ...

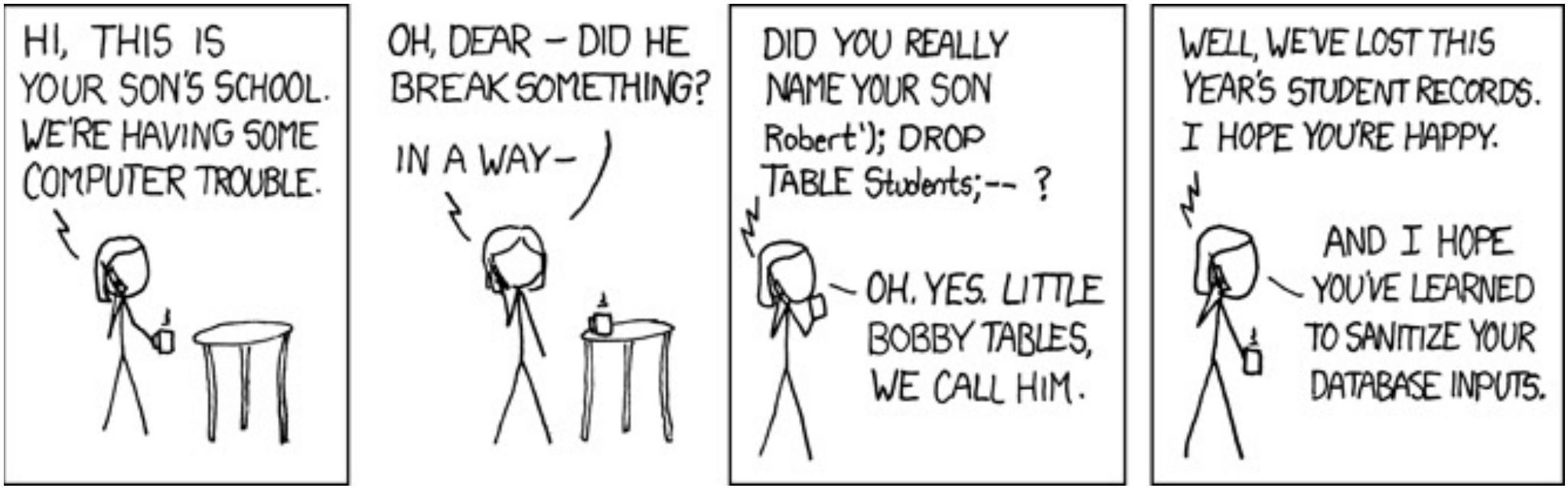
SQL Injection

http://w4111.github.io/inject

code on github:
`w4111/w4111.github.io/src/injection/`

Use query parameters by passing form values as arguments to `execute()`
Library sanitizes inputs automatically (and correctly!)

SQL Injection



Project: You'll need to protect against simple SQL injections

More examples: https://corenumb.wordpress.com/2016/05/14/mr-robot-blind-sql-injection-vulnerability/

Privacy vs Security

Privacy:

- person's right to control their personal information and how it's used
- Should not release data that can be used to *infer* user's personal information
- hard to define

Security:

- prevent unauthorized access to data
- access controls
- encryption prevents reading data even w/ access

Access Control and GRANT

Different user accounts in w4111 DB

- staff, student, ew2493

Each user only has access privileges to read/modify subset of DB:

- Privileges: read, insert, update, delete
- Objects: Databases, Schemas, Tables, Views, Attributes
- Who? users, roles

https://www.postgresql.org/docs/current/sql-grant.html

Access Control and GRANT

```
GRANT <privileges>
   ON <objects>
   TO <users/roles>
```

```
CREATE USER ew2493;
CREATE ROLE admin;

GRANT SELECT, INSERT ON users TO admin;     // users relation
GRANT CREATE, CONNECT ON DATABASE test TO admin;

GRANT admin to ew2493;
```

Access Control and GRANT

How to restrict user's access to subsets of a relation or only aggregated statistics?

Combine GRANT and Views!

```
CREATE VIEW stats AS
    SELECT department, avg(salary)
      FROM costs
  GROUP BY department

GRANT SELECT ON stats TO appuser
```

Hashing and Encryption

Prevent data from being read, even if database contents are accessed

Hashing: one way function, loses original data

- used to check equality

```
Data → hash() → hasheddata
hash(password) == hash(input)
```

Encryption: 2 way function, is reversible

- only users with key can read
- key needs to be kept safe!

```
Data → encrypt(key) → encdata → decrypt(key) → Data
```

https://www.postgresql.org/docs/current/encryption-options.html

Hashing and Encryption

DBMS support varies

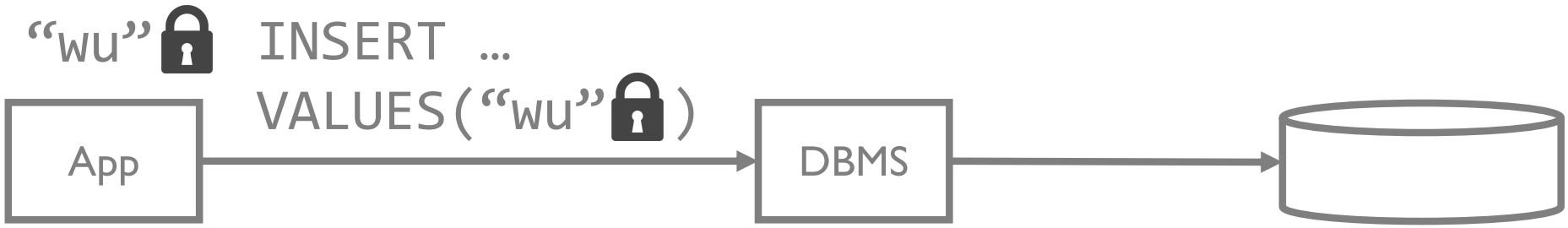
Hashing implemented as UDFs

- `INSERT INTO users VALUES(name, hash(password))`

Encryption

- encrypted hard drive, encrypted disk blocks, table, columns, …

Application encrypts `data` before issuing queries



Hashing and Encryption

DBMS support varies

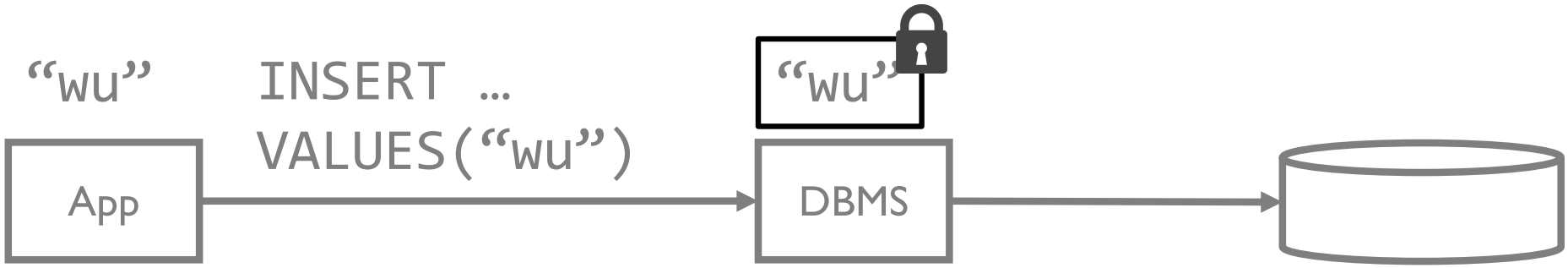
Hashing implemented as UDFs

- `INSERT INTO users VALUES(name, hash(password))`

Encryption

- encrypted hard drive, encrypted disk blocks, table, columns, …

DBMS encrypts data received from application



Hashing and Encryption

DBMS support varies

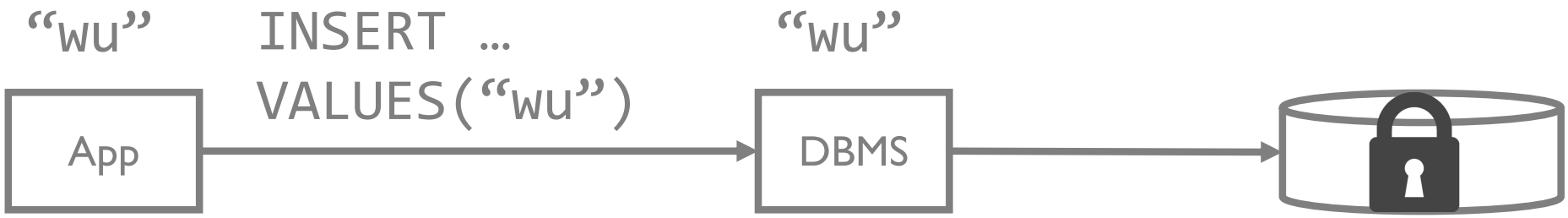
Hashing implemented as UDFs

- `INSERT INTO users VALUES(name, hash(password))`

Encryption

- encrypted hard drive, encrypted disk blocks, table, columns, …

Storage encrypts everything on e.g., hard drive



Summary

DBAPIs
- Impedance mismatch
- Cursors
- SQL injection

Some hard queries
- More in the HW

Windows are optional material

SQL Injection: only what's in slides

What to Understand

Why Embedded SQL is no good

Client-server vs embedded DBMSes

DBAPI components, cursors

Impedance mismatch: examples and possible solutions

SQL injection and protections

Levels of access controls and Views

Conceptual understanding of privacy & encryption