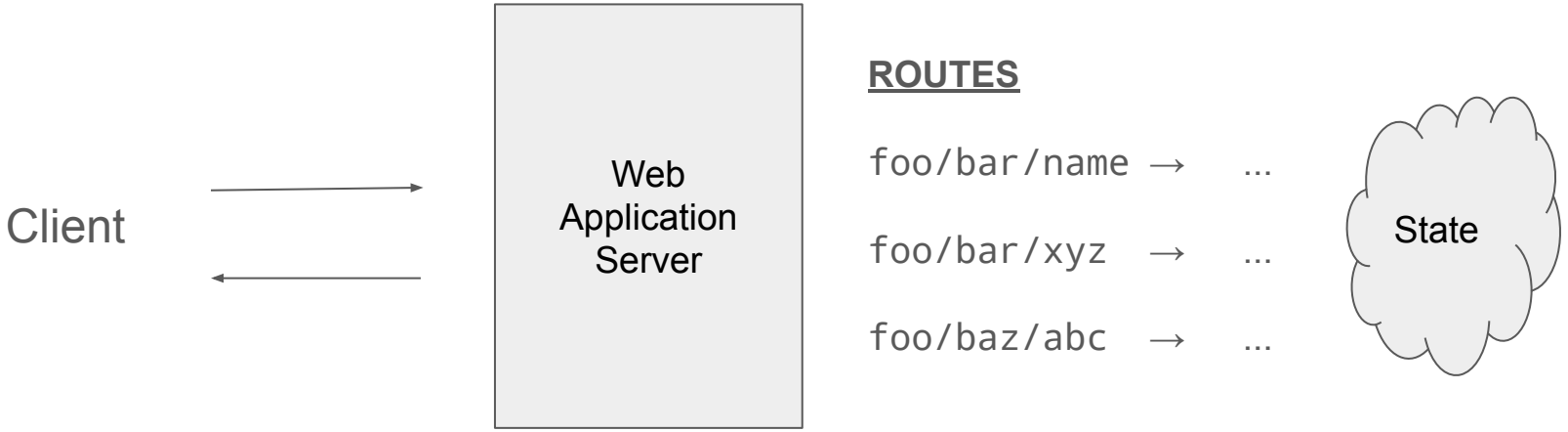
Persistence

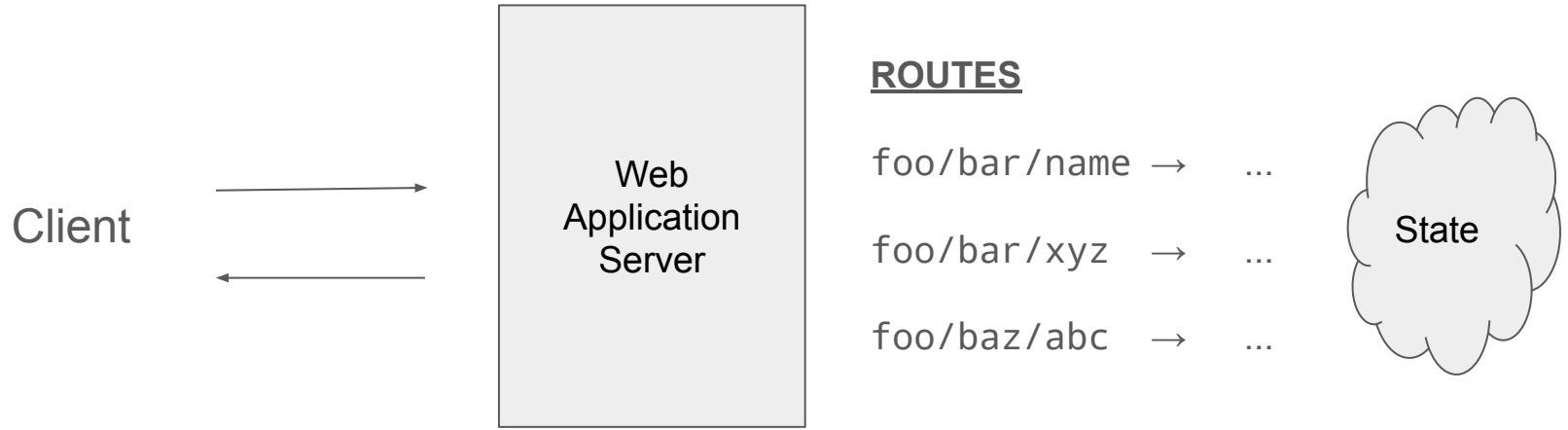
Spring 2024

Web Application Server



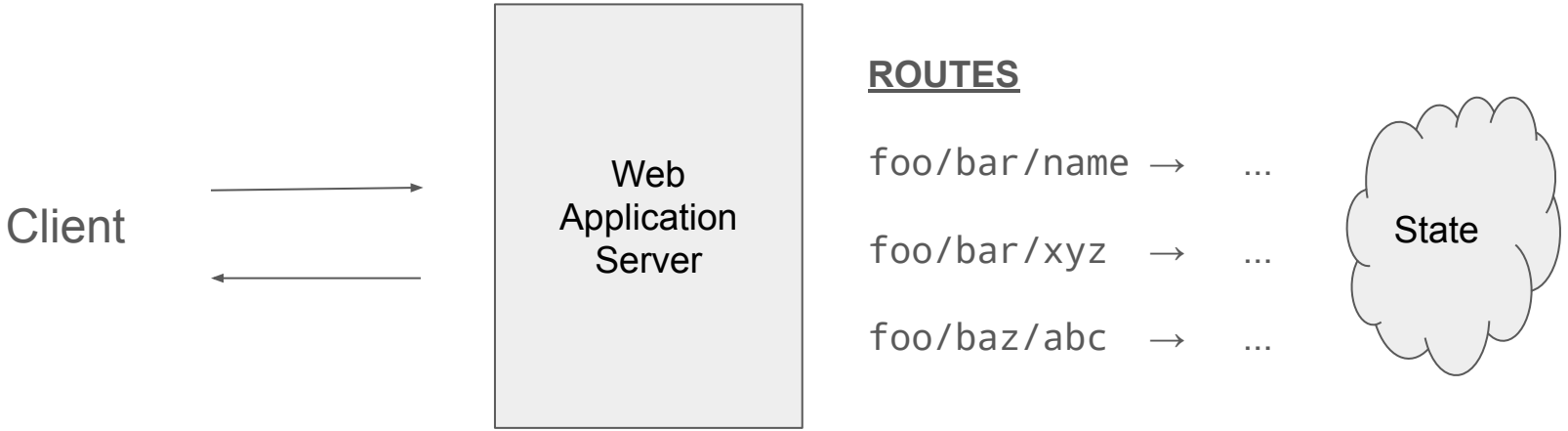
Web application servers generally need to maintain some state to handle requests from clients

Web Application Server



- list of available pictures
- the pictures themselves
- comments
- list of users

Web Application Server



When the server stops, that state is lost (crash, update deployment)

How do we **persist the server state** so that it survives restarts?

Approach 1: The file system

Store the state of the web application server on the file system

- it's right there
- it's easy to read/write to the file system

It's not a great approach though:

- it's slow
- concurrent writes can clash
- it doesn't scale

Still, it provides a starting point for a better approach

Adding persistence

Steps:

1. Determine what you want to persist across server instances
2. Isolate data structures that hold data to be persisted
3. Write getter/setter methods to pull/push data into the data structures
4. Augment getter/setter methods to mirror data structure changes to disk

5. Initialize data structures with content from disk

DEMO - persistence via filesystem

Approach 2: Databases

A **database** is a structured way to manage data (stored on disk) via a piece of software called a **Database Management System** (DBMS).

Many kinds of DBMS out there, but they mostly split into two groups:

- **Relational** DBMS: data is structured into tables
- **Nonrelational (NoSQL)** DBMS: data is structured some other way:
  - collections of JSON objects
  - key-value pairs (one big flat dictionary)

Relational DBMS

Examples: MySQL, PostgreSQL, Oracle, Microsoft SQL Server

Data is stored in **tables**

- a table has fields and each row in the table assigns a value to those fields

name	url	created
Cat	http://localhost:8080/image/cat.png	2021-03-17 15:03:45
Dog	http://localhost:8080/image/dog.jpg	2021-03-17 15:06:03
...	...	...

Querying relational DBMS via **SQL** (Structured Query Language):

```
SELECT name, url, created FROM pictures WHERE created >= '2021-03-01'
```

Nonrelational (NoSQL) DBMS

Examples: Redis, Cassandra, MongoDB, CouchDB

Redis/Cassandra store data as **key-value pairs**

MongoDB/CouchDB store data as collections of **JSON objects**

Query language is idiosyncratic to each nonrelational DBMS

Nonrelational DBMS scale a lot better than relational databases

- developed to deal with huge datasets (full content of the web @ Google)

Consistency guarantees

Relational DBMS offer **ACID** guarantees:

- Atomicity, Consistency, Isolation, Durability
- When you interact with a database, it's as if you're the only one interacting with the database - what somebody else does at the same time will not interfere with your work. (Your work may be delayed or even rejected)
- Cannot get the database in an inconsistent state

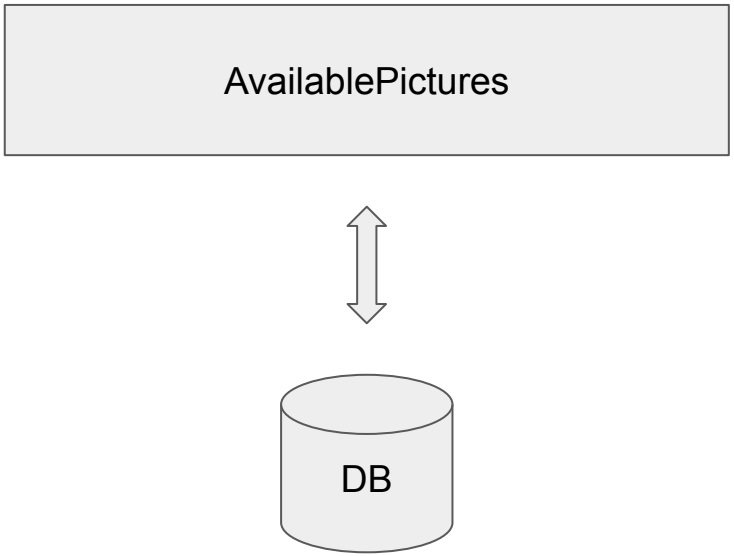
NoSQL DBMS offer… no such guarantees:

- Conflicts may need to be resolved manually
- At best, the data will **eventually** be **consistent**

Demo - using SQLite

(As a proxy for MySQL or PostgreSQL)

Systematizing DB interactions

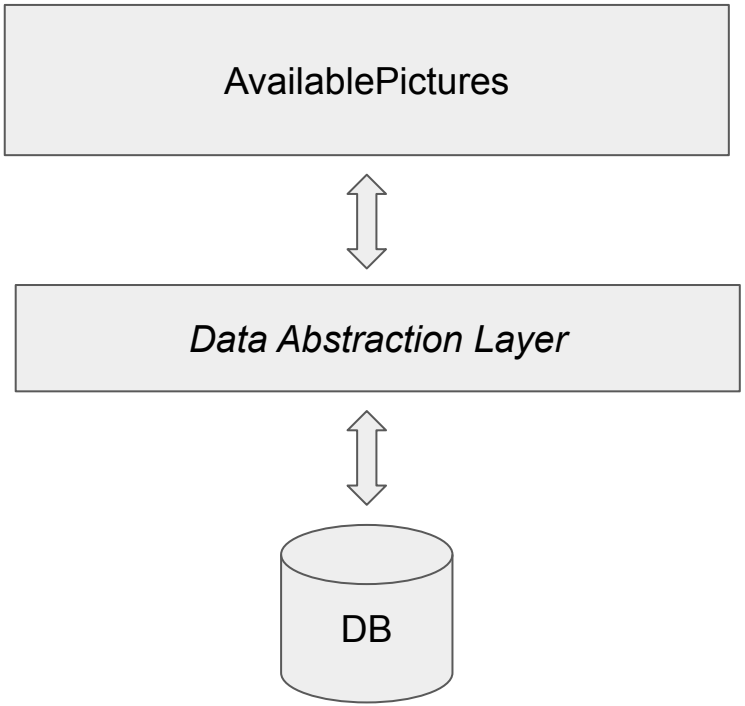


Easy enough to sprinkle SQL within server code to persist data to the DB

BUT:

- it doesn't scale
- hard to maintain
- hard to retarget to a new DB

Systematizing DB interactions



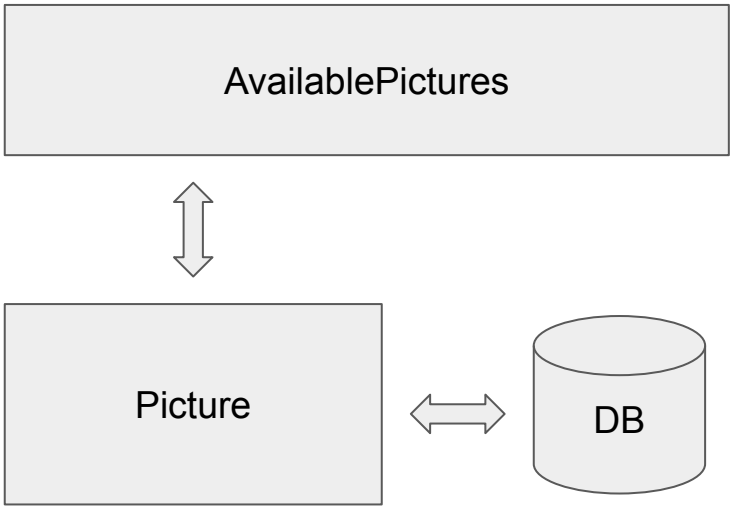
Modern web application server infrastructures tend to advocate ORMs

- ORM = Object-Relational Mapping
- attempt at abstracting away the DB by connecting application objects to DB tables

Mostly used with relational databases

- ideas apply to NoSQL though

Active Records



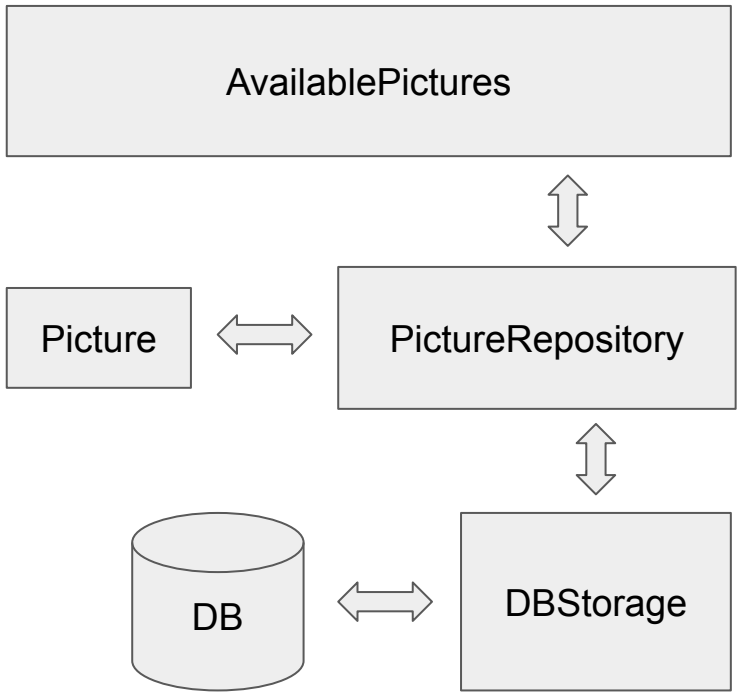
Associate with every table T of the DB an application object representing a row of data

Interact with the table via application object

- creating a new row
- updating an existing row
- reading rows

Application objects know how they are persisted

Data Mapper



Used by most ORMs

Disentangle application objects (Picture) from how they are persisted (DBStorage)

Connection done through a Repository

Can retarget by changing the Storage component only