



# Databases & ORMs

---

# ACID

- **Atomicity**
- **Consistency**
- **Isolation**
- **Durability**

# ACID

- All about transactions (key in financial transactions)
- **Atomicity** — all or nothing
- **Consistency** — must bring us to a valid state
- **Isolation** — concurrent execution that mirrors serial execution results
- **Durability** — transactions persist

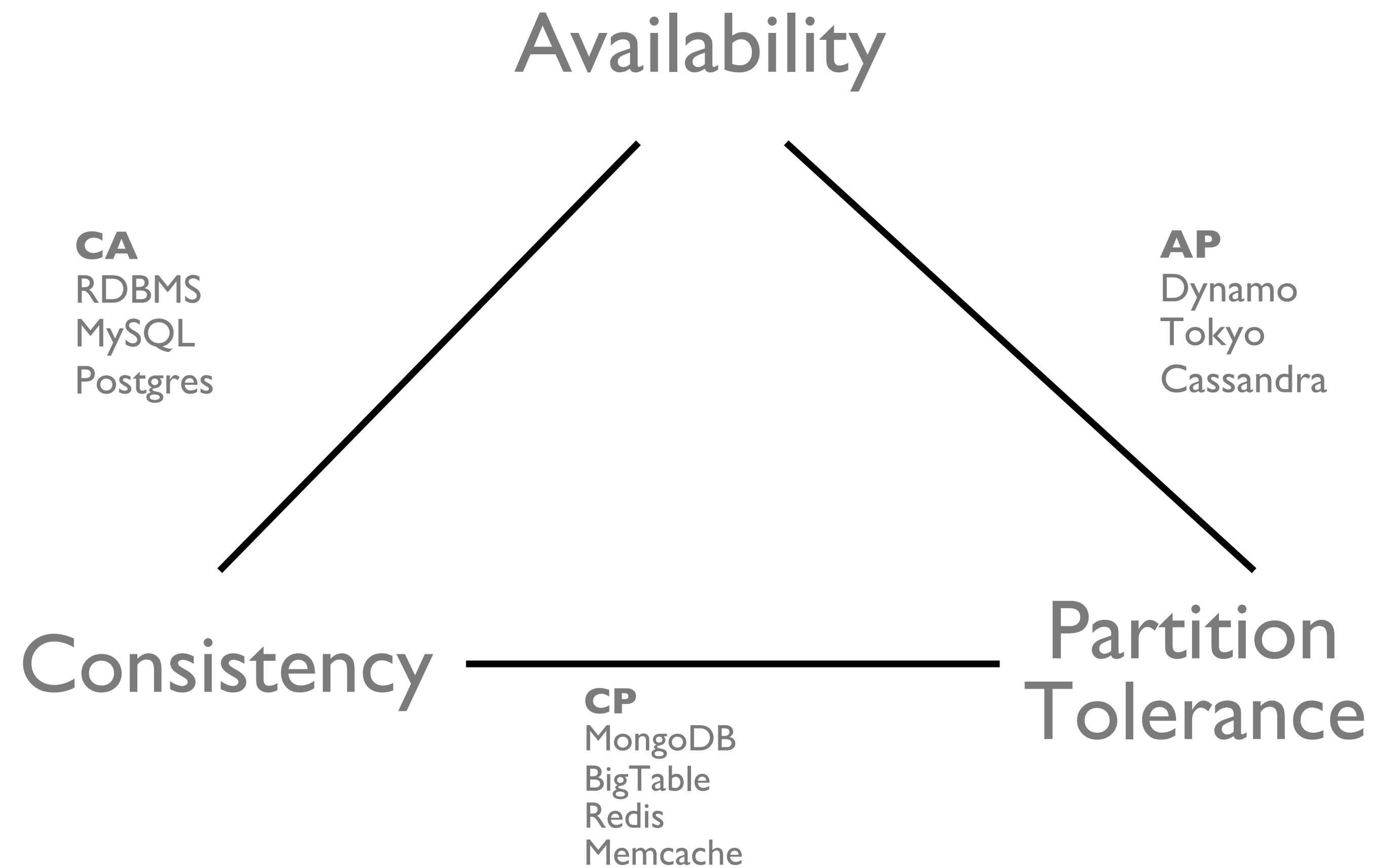
# CAP Theorem

- **Consistency:** Every read receives the most recent write, or an error
- **Availability:** Every request receives an immediate (non-error) response – (without guarantee that it contains the most recent write)
- **Partition Tolerance:** system works even if network splits
- **"Pick 2"**
  - (Reality is more complex...)





# CAP, metaphorically speaking



# BREAK

# Sequelize

- **Sequelize is an Object-Relational Mapper (ORM)**
- **Access SQL databases from Node.js**
- **Sequelize features:**
  - Schema modeling/validation
  - Data casting (convert SQL types to JS types)
  - Query building
  - Hooks (code that runs pre/post save/delete/update)
  - Class and instance methods of models
  - Getters, setters, and virtual fields





**Tables**

**Models**

**+**

**=**

**+**

**Rows**

**Instances**

# Sequelize Basics

- Make a **Model** (interactive blueprint object)
- Extend the **Model** with **Hooks**, **Class & Instance Methods**, **Virtuals**, etc.
- Connect/sync the completed **Model** to an *actual* table in an *actual* SQL database
- Use the **Model** (Table) to create/find **Instances** (row)
- Use the **Instances** to save/update/delete



# Create a Model

```
const Sequelize = require('sequelize');

const db = new Sequelize('postgres://localhost:5432/twitter');

const User = db.define('user', {
  name: Sequelize.STRING,
  pictureUrl: Sequelize.STRING
});
```



# Sync Model to Table

```
User.sync().then(...);
```



# Model & Instance Usage

```
const person = User.build({  
  name: "Kate",  
  pictureUrl: "http://fillmurray.com/10/10"  
});
```

```
person.save()  
  .then(...);
```

```
User.findAll()  
  .then(...);
```

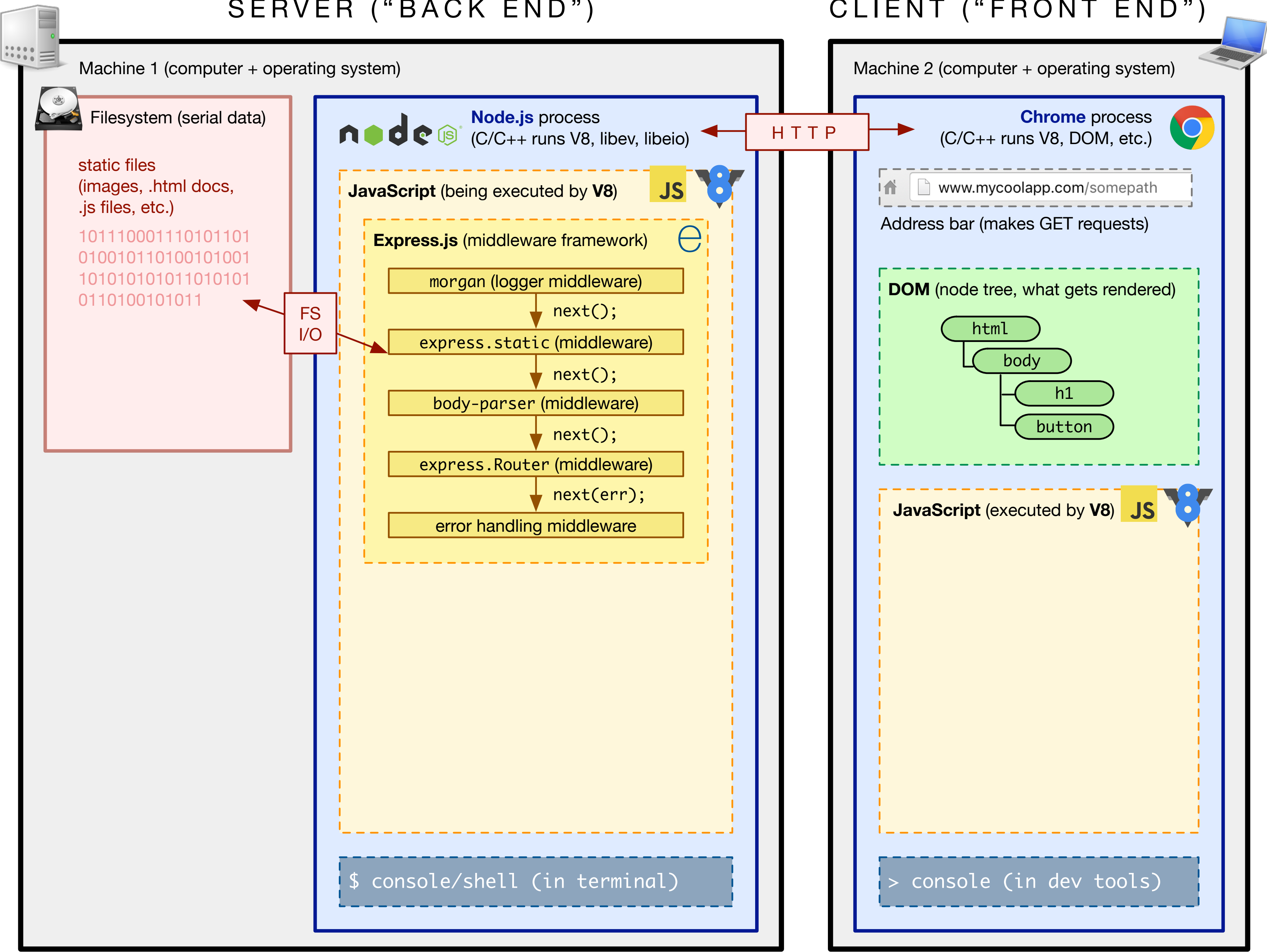
# Sequelize

- Lives inside Node.js process
- Knows how to communicate to a few SQL DBMSs, including PostgreSQL and sqlite3



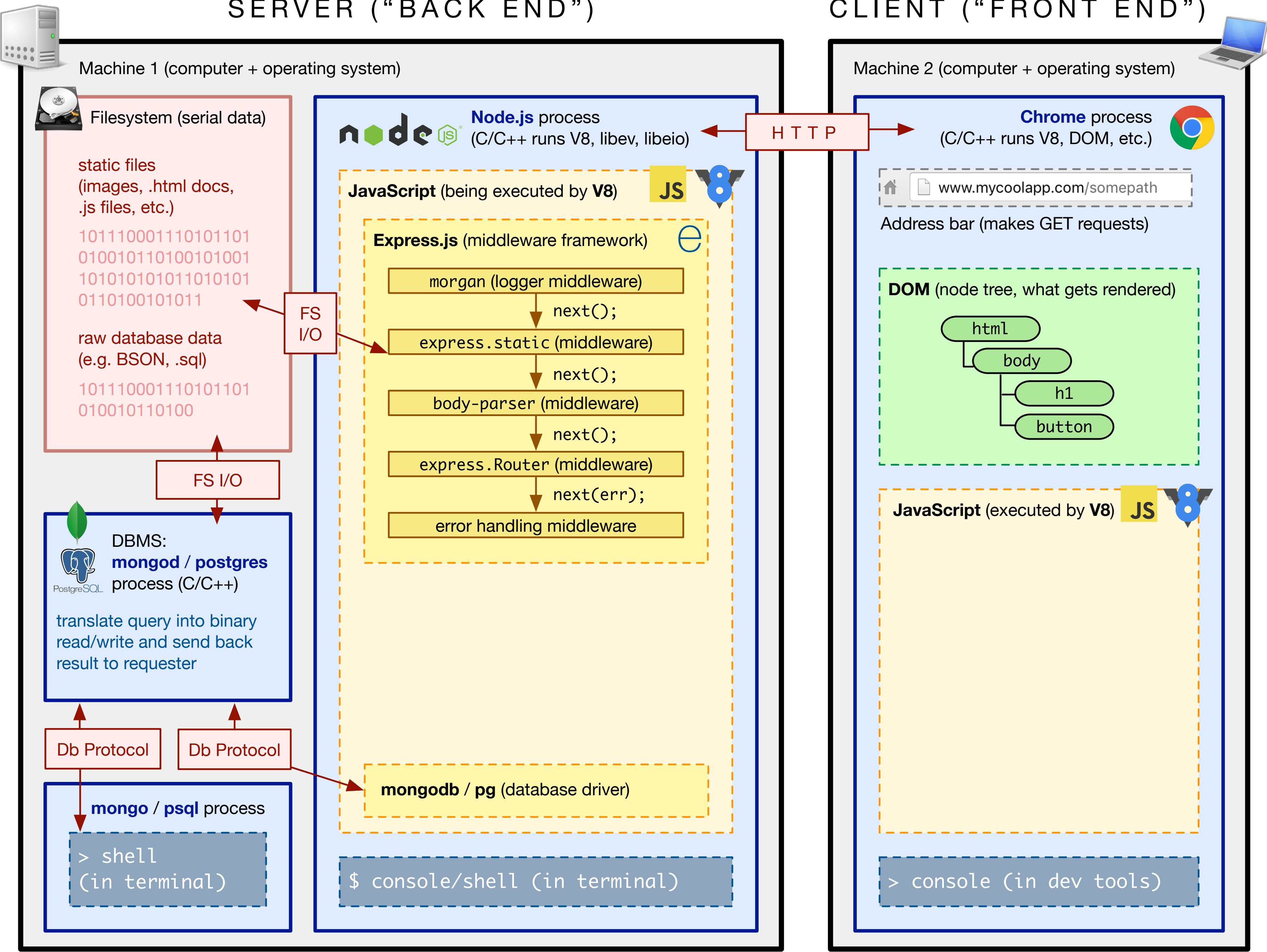
SERVER ("BACK END")

CLIENT ("FRONT END")



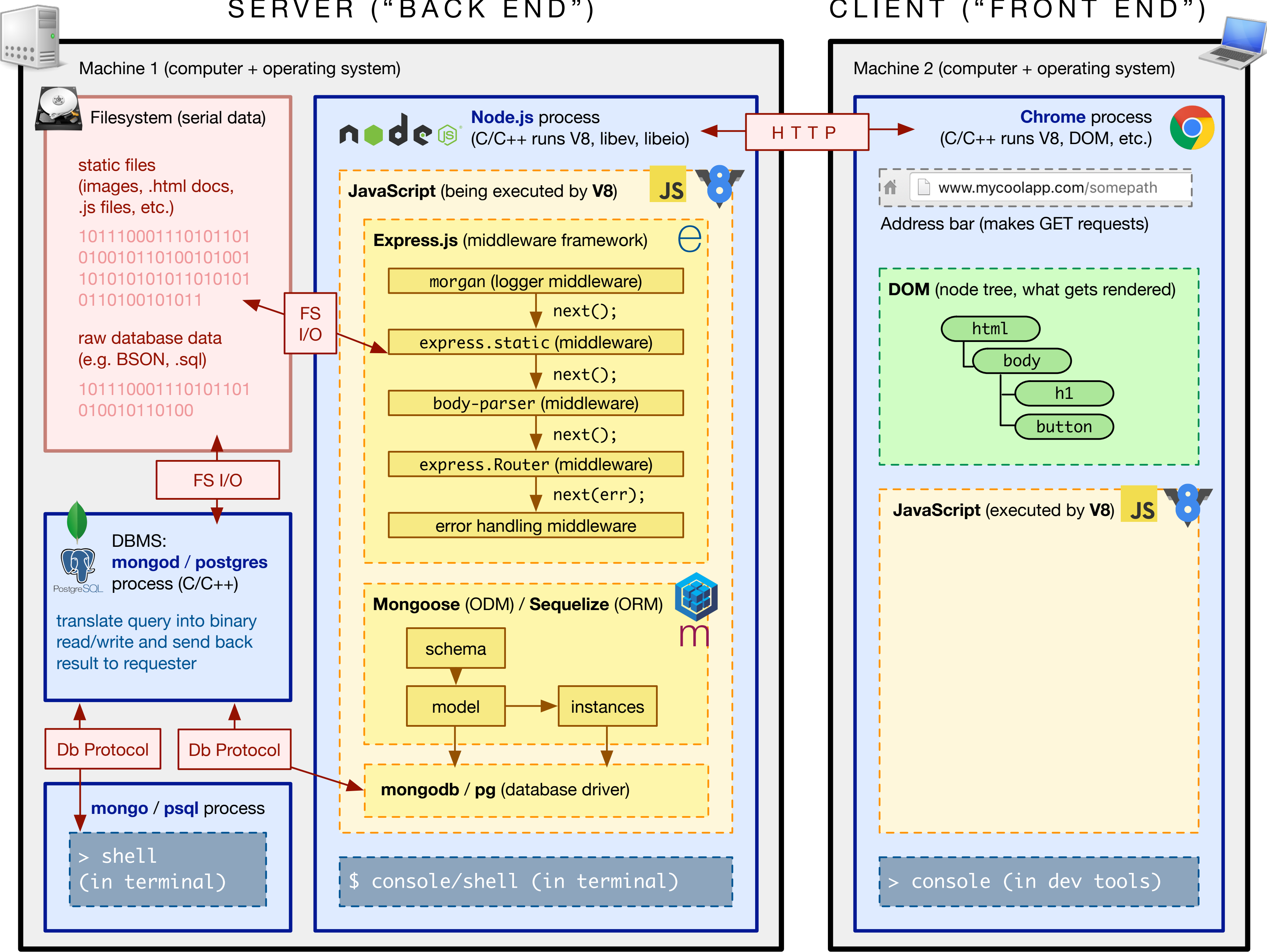
SERVER ("BACK END")

CLIENT ("FRONT END")



SERVER ("BACK END")

CLIENT ("FRONT END")



# Let's Walk Through What Happens



## Sequelize — ORM

- Reads the JS code `Users.create({name: 'Kate'})` =>
- Constructs the SQL query `INSERT INTO users (name) VALUES ("Murphy") RETURNING (id, name)` =>
- Passes that SQL query to the JS library `pg` =>

# pg module

- ◉ Connects via TCP/IP to Postgres =>
- ◉ Uses the postgres protocol to tell Postgres it has an incoming SQL query =>
- ◉ Sends the SQL query to Postgres =>



PostgreSQL — dialect, database

- ◉ Parses the query =>
- ◉ Changes the data on disk =>
- ◉ Sends a response back to `pg` via the postgres protocol on the TCP connection =>



# pg module

- Receives raw string data, perhaps something like  
“created 1 row in users table (id, name) values (1, Murphy)”  
=>
- Turns raw string into an array of row objects  
`[{id: 1, name: 'Murphy'}]` and hands it to Sequelize =>



Sequelize

(ORM)

- Mutates the returned data and constructs new, powerful objects with prototypal methods, e.g. `save` =>
- Resolves the promise it returned from `create` with this array of Sequelize instance objects.



# Wikistack

- Build a Wikipedia clone
- Walk you through installing and using sequelize
- Application of everything we've learned so far