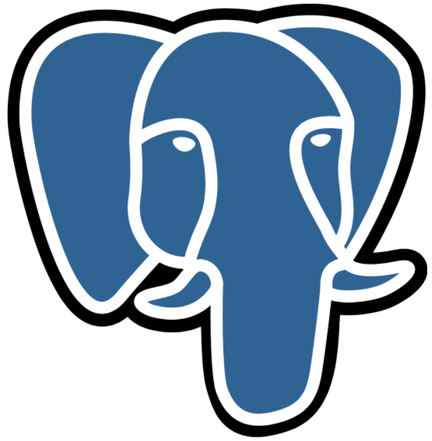
REMEMBER TO RECORD

Databases & ORMs

Some Postgres Client

Postgres Protocol



CLIENT SIDE

SERVER SIDE

Request / Response HTTP

Express JS

node JS

PSQL(command line) or Postico (GUI)

PostgreSQL

CLIENT SIDE

SERVER SIDE

Request / Response HTTP

Express JS

PG

PostgreSQL

CLIENT SIDE

SERVER SIDE

Request / Response HTTP

Express JS

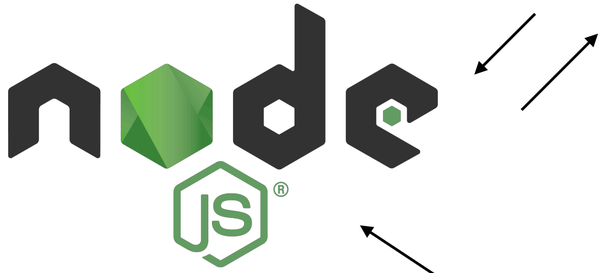
PG

PostgreSQL

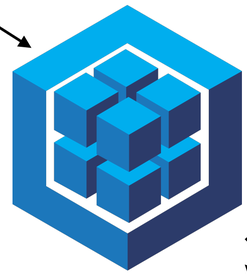
CLIENT SIDE

SERVER SIDE

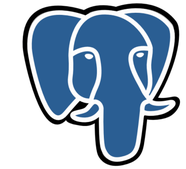
Request / Response HTTP



Express JS



PG



PostgreSQL

Object Relational Mapper

- Acts as a “bridge” between your code and the RDBMS.

- Using ORM, data can be easily stored and retrieved from a database without writing SQL statements directly.

Pros/Cons

- **Pros**
  - **Huge reduction in code.**
  - **Easier to read and understand (more javascripty)**
  - **No need to write SQL (unless….)**
  - **ORM's are DBMS agnostic**
- **Cons**
  - Often means people don't take the time to understand SQL and all its power
  - Abstraction layer adds computational time
  - Not the best when it comes to complex operations

Sequelize

- **Sequelize is an Object-Relational Mapper (ORM)**
- **Access SQL databases from Node.js**
  - Using JS objects and methods instead of SQL statements
- Represents tables as “classes” and rows as objects (instances)

Without ORM

```
client.query(`select * from dogs`)

client.query(`select * from cats`)

client.query(`select * from hippos`)
```

With ORM

```
Dog.findAll()

Cat.findAll()

Hippo.findAll()
```

Tables

\+

Rows

=

Models

\+

Instances

Basic Workflow

Sequelize Basics: Workflow

- Instantiate Sequelize

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

Sequelize Basics: Workflow

- **Instantiate Sequelize**

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- **Define your Model(s)**
  - Add options to Model fields (validations, default values & more)

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

Sequelize Basics: Workflow

- **Instantiate Sequelize**

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- **Define your Model(s)**
  - Add options to Model fields (validations, default values & more)

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

Sequelize Basics: Workflow

- **Instantiate Sequelize**

```
const Sequelize = require('sequelize')
const db = new Sequelize('postgres://localhost/wiki')
```

- **Define your Model(s)**
  - Add options to Model fields (validations, default values & more)

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

- **Connect/sync the Model to an *actual* table in the database**

```
await User.sync()
```

Sequelize Basics: Workflow

- Use the **Model** (Table) to find/create **Instances** (row)

```
const users = await User.findAll();
```

Sequelize Basics: Workflow

- Use the **Model** (Table) to find/create **Instances** (row)
- Use the **Instances** to save / update / delete

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

```
await person.save();
```

```
const pug = await User.create({
  name: “Cody",
  pictureUrl: "http://fillmurray.com/10/10"
});
```

Additional Model Options

- Sequelize models can be extended Hooks, Class & Instance Methods, Getter & Setters, Virtuals, etc.

Hooks

- **When you perform various operations in Sequelize (creating, updating, destroying, etc), various "events" occur. These are called "lifecycle events"**

- **Hooks are like adding an event listener to these events**
  - *"Every time a journal entry is created or updated, escape any dangerous sequences that could result in an XSS attack"*
  - *"Every time a user is updated with a new password, hash it so that the plaintext password doesn't get saved in the database"*

beforeValidate

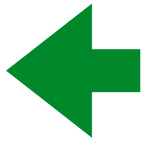
validation

afterValidate

beforeCreate

creation

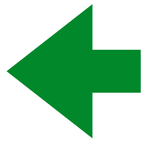
afterCreate



```
User.beforeValidate((user) => {

})
```

beforeValidate

validation



afterValidate

beforeCreate

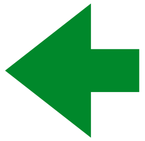
creation

afterCreate

beforeValidate

validation

afterValidate



beforeCreate

creation

afterCreate

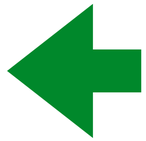
```
User.afterValidate((user) => {

})
```

beforeValidate

validation

afterValidate

beforeCreate



creation

afterCreate

```
User.beforeCreate((user) => {

})
```

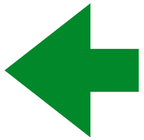
beforeValidate

validation

afterValidate

beforeCreate

creation



afterCreate

beforeValidate

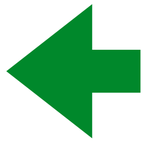
validation

afterValidate

beforeCreate

creation

afterCreate



```
User.afterCreate((user) => {

})
```

Associations

Associations

- **Establishes** a **relationship** between **two tables**
  (using a foreign-key or a join-table)
- **Creates** several **special instance methods** **(like** `getAssociation` **&** `setAssociation`**),** that an **instance** can use to **search for the instances that they are related to.**
- **And more… (eager loading, etc)**

Associations

```
const User = db.define("user", {…})
const Pet  = db.define(“pet", {…})

Pet.belongsTo(User)
User.hasMany(Pet)
```

Associations

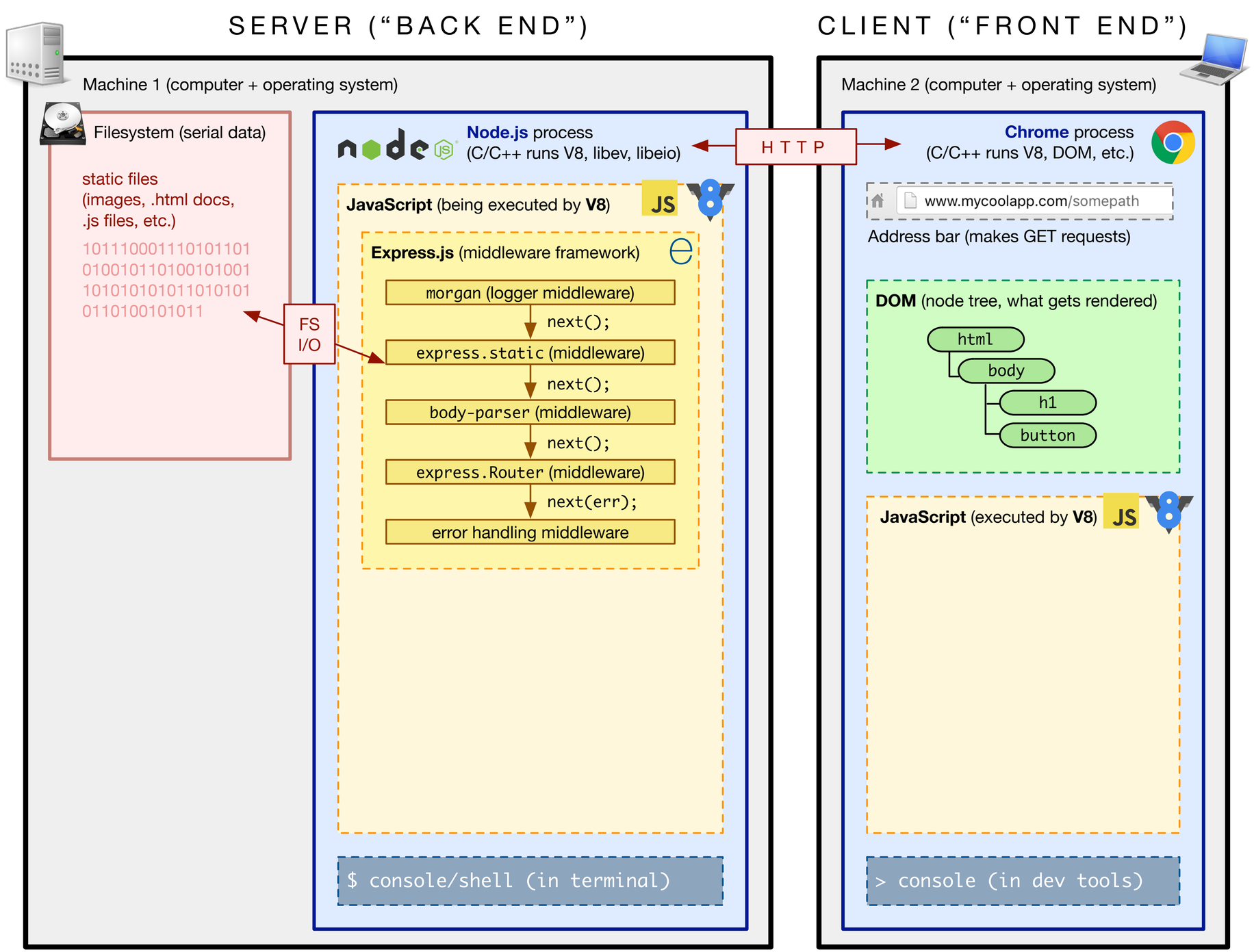
```
const User = db.define("user", {…})
const Pet  = db.define(“pet", {…})

Pet.belongsTo(User)
User.hasMany(Pet)
```

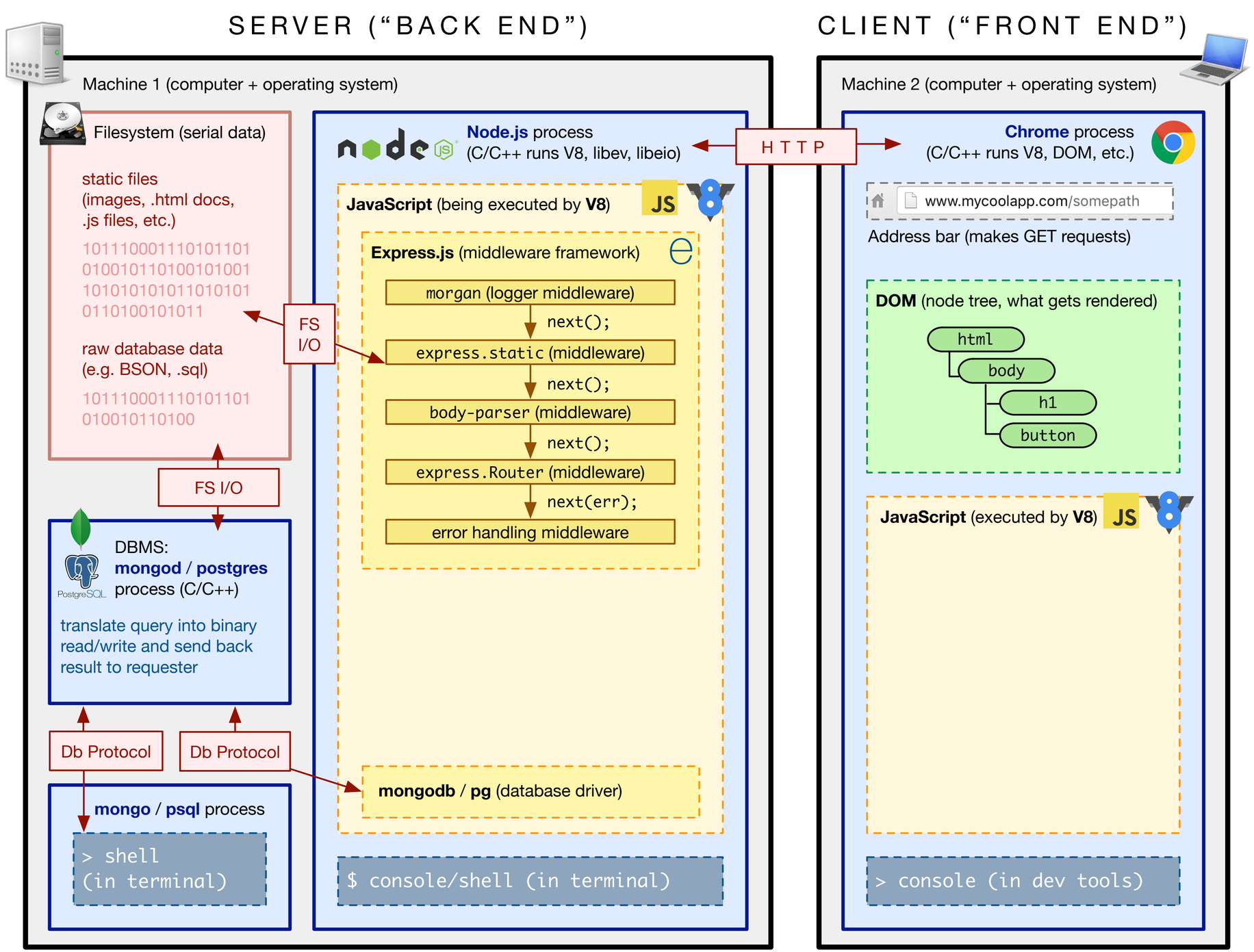
```
const someUser   =  await User.findById(12)
const andHisPet  =  await someUser.getPets()
```

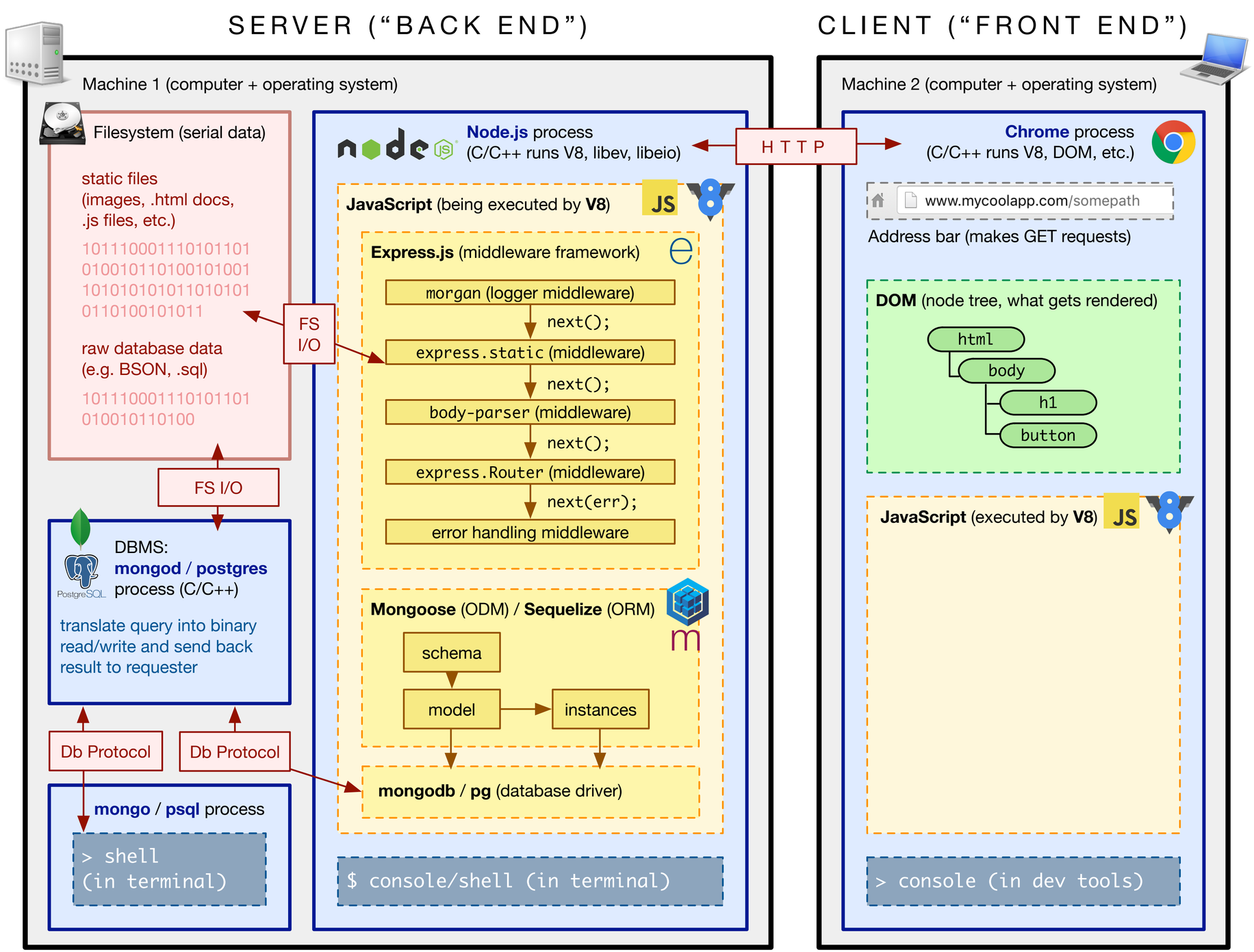
A little more context

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")
Machine 1 (computer + operating system)
Machine 2 (computer + operating system)
Filesystem (serial data)
static files
(images, .html docs,
.js files, etc.)
10111000111010110101001010110100101001010101011010101011010010101011
FS
I/O
Node.js process
(C/C++ runs V8, libev, libeio)
HTTP
Chrome process
(C/C++ runs V8, DOM, etc.)
JavaScript (being executed by V8)
JS
Express.js (middleware framework)
morgan (logger middleware)
next();
express.static (middleware)
next();
body-parser (middleware)
next();
express.Router (middleware)
next(err);
error handling middleware
www.mycoolapp.com/somepath
Address bar (makes GET requests)
DOM (node tree, what gets rendered)
html
body
h1
button
JavaScript (executed by V8)
JS
$ console/shell (in terminal)
> console (in dev tools)


SERVER ("BACK END")
CLIENT ("FRONT END")
Machine 1 (computer + operating system)
Machine 2 (computer + operating system)
Filesystem (serial data)
static files
(images, .html docs,
.js files, etc.)
101110001110101101
010010110100101001
101010101011010101
0110100101011
raw database data
(e.g. BSON, .sql)
101110001110101101
010010110100
FS I/O
FS I/O
DBMS:
mongod / postgres
process (C/C++)
PostgreSQL
translate query into binary
read/write and send back
result to requester
Db Protocol
Db Protocol
mongo / psql process
> shell
(in terminal)
Node.js process
(C/C++ runs V8, libev, libeio)
HTTP
JavaScript (being executed by V8)
JS
Express.js (middleware framework)
morgan (logger middleware)
next();
express.static (middleware)
next();
body-parser (middleware)
next();
express.Router (middleware)
next(err);
error handling middleware
mongodb / pg (database driver)
$ console/shell (in terminal)
Chrome process
(C/C++ runs V8, DOM, etc.)
www.mycoolapp.com/somepath
Address bar (makes GET requests)
DOM (node tree, what gets rendered)
html
body
h1
button
JavaScript (executed by V8)
JS
> console (in dev tools)


SERVER ("BACK END")
CLIENT ("FRONT END")
Machine 1 (computer + operating system)
Machine 2 (computer + operating system)
Filesystem (serial data)
static files
(images, .html docs,
.js files, etc.)
101110001110101101
010010110100101001
101010101101010101
0110100101011
raw database data
(e.g. BSON, .sql)
101110001110101101
010010110100
FS I/O
FS I/O
DBMS:
mongod / postgres
process (C/C++)
translate query into binary
read/write and send back
result to requester
Db Protocol
Db Protocol
mongo / psql process
> shell
(in terminal)
Node.js process
(C/C++ runs V8, libev, libeio)
HTTP
JavaScript (being executed by V8)
Express.js (middleware framework)
morgan (logger middleware)
next();
express.static (middleware)
next();
body-parser (middleware)
next();
express.Router (middleware)
next(err);
error handling middleware
Mongoose (ODM) / Sequelize (ORM)
schema
model
instances
mongodb / pg (database driver)
$ console/shell (in terminal)
Chrome process
(C/C++ runs V8, DOM, etc.)
www.mycoolapp.com/somepath
Address bar (makes GET requests)
DOM (node tree, what gets rendered)
html
body
h1
button
JavaScript (executed by V8)
> console (in dev tools)

Wikistack

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