intro to orms

Intro to ORMs

Sequelize

- Sequelize is an Object-Relational Mapper (ORM)
- Access SQL databases from Node.js

Tables

\+

Rows

Classes

\+

Instances

Sequelize Basics

1. Define a Model (interactive blueprint object)
2. Extend the Model with Hooks, Class & Instance Methods, Virtuals, etc.
3. Connect/sync the completed Model to an *actual* table in an *actual* SQL database
4. Use the Model (Table) to create/find Instances (row)
5. Use the Instances to modify and delete rows

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

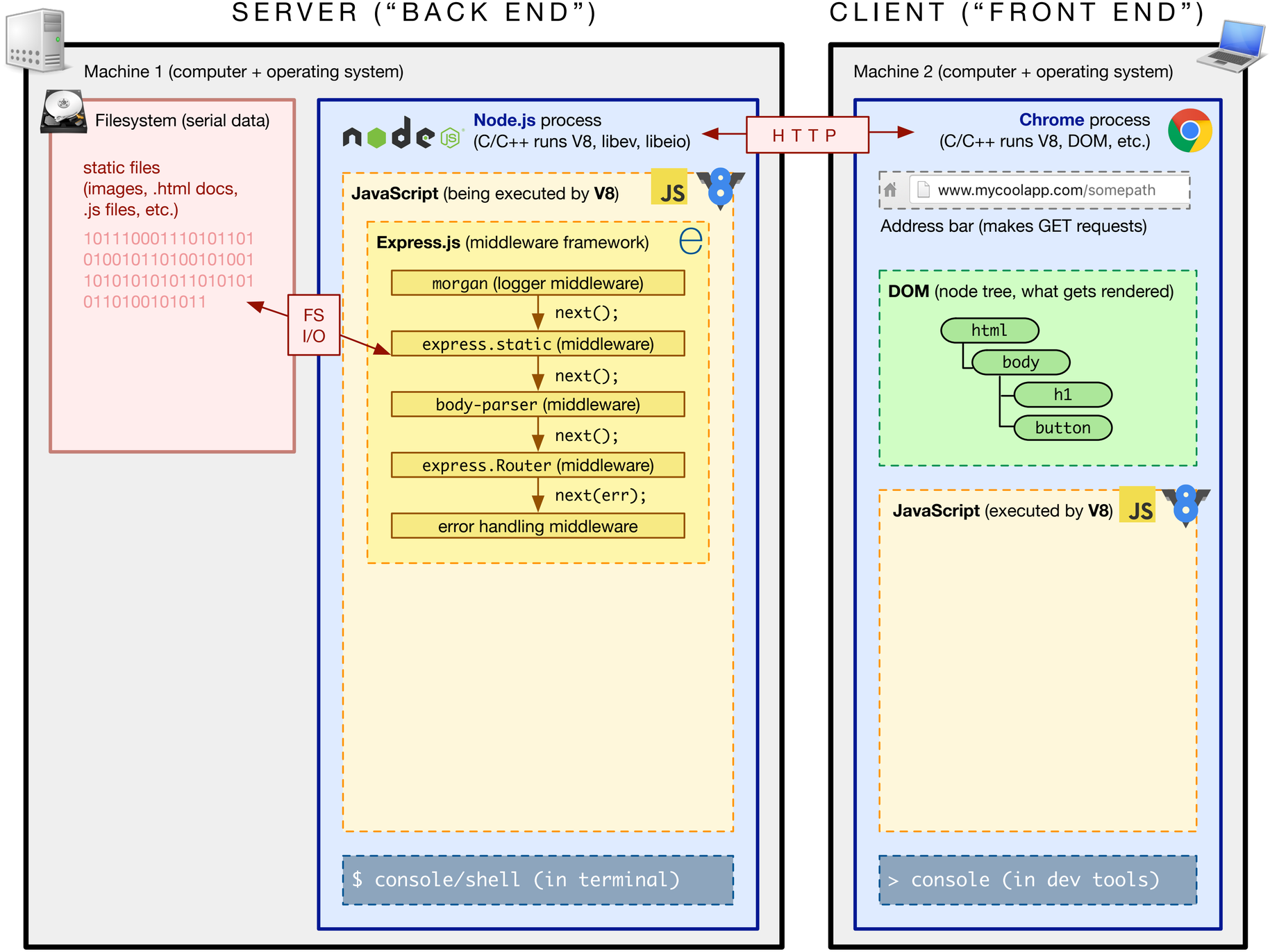
```
User.sync().then(…);
```

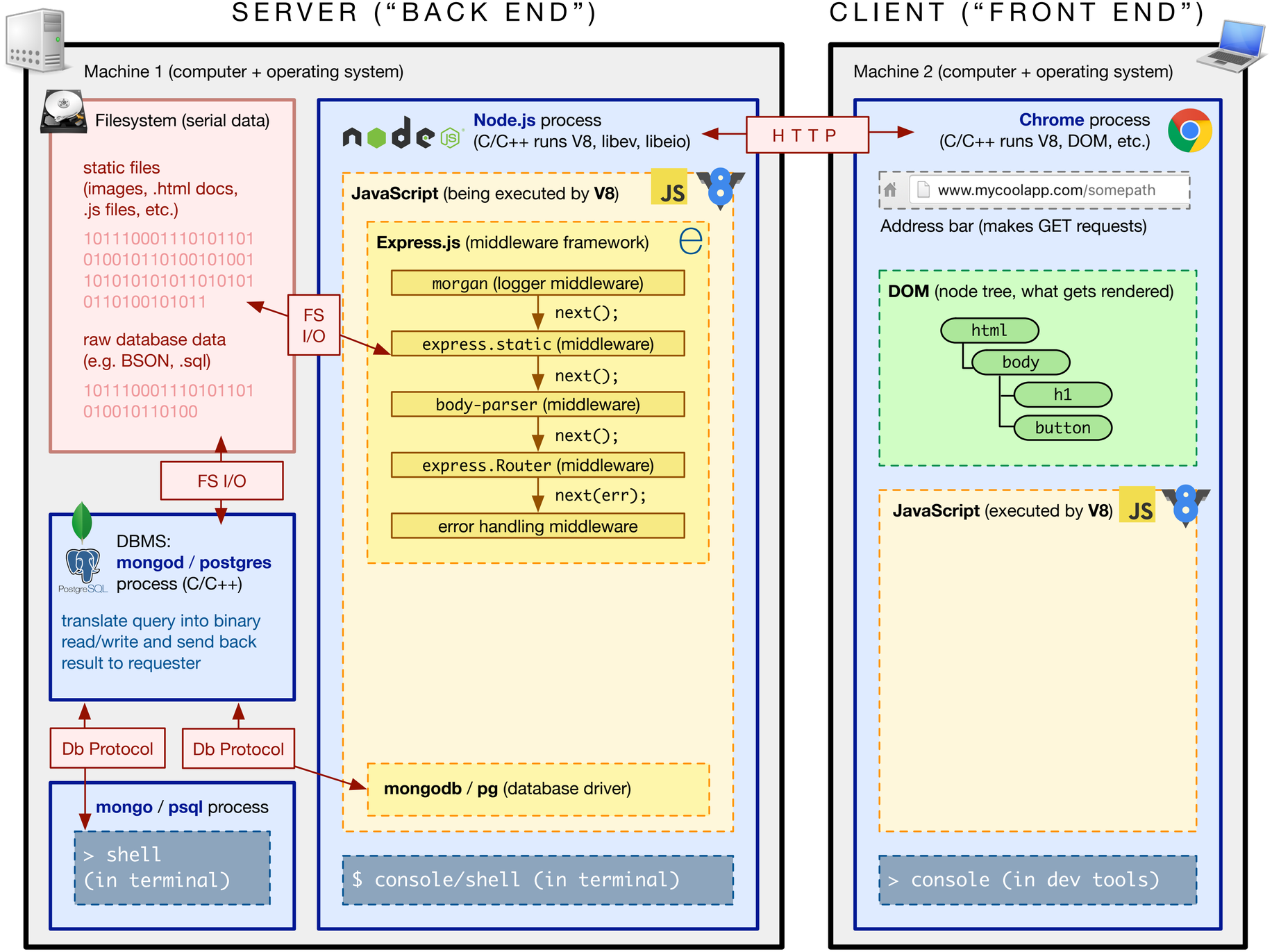
Model & Instance Usage

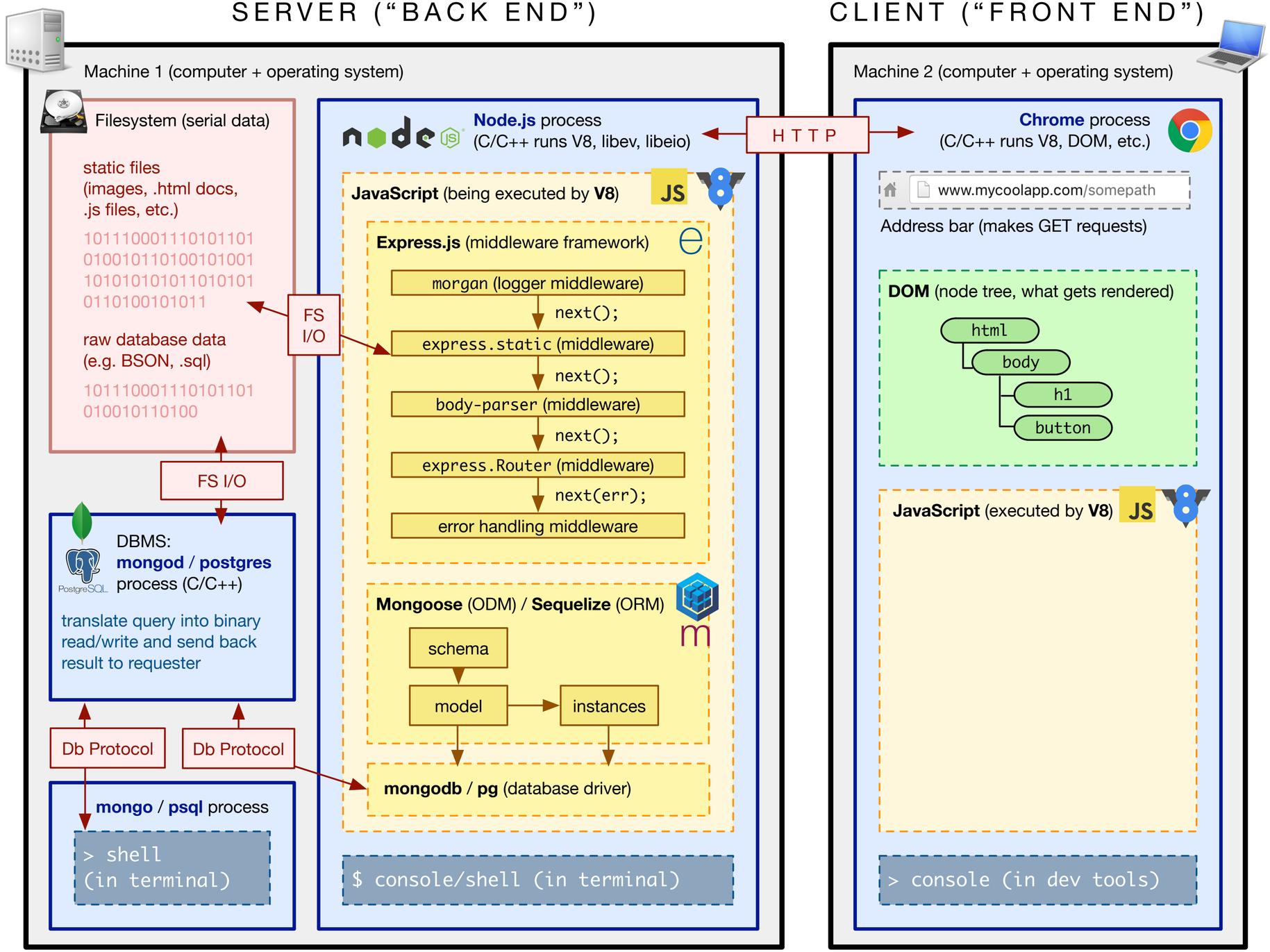
```
const person = User.build({
  name: "Kate",
  pictureUrl: "http://fillmurrary.com/10/10"
});

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")
Machine 1 (computer + operating system)
Filesystem (serial data)
static files
(images, .html docs,
.js files, etc.)
101110001110101101
010010110100101001
101010101011010101
0110100101011
FS
I/O
Node.js process
(C/C++ runs V8, libev, libeio)
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
$ console/shell (in terminal)
HTTP
Machine 2 (computer + operating system)
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
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)
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)
HTTP
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
101010101011010101
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)
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)
HTTP
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)

Let's Walk Through What Happens

- `Users.create({name: 'Kate'})`
- **"INSERT INTO users (name) VALUES ("Kate") RETURNING (id, name)"**
  - **to pg npm**

pg (in node process)

- 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

- Parses the query, builds a query plan
- Runs query plan, changes the data on disk
- Sends a response back to `pg` via the postgres protocol on the TCP connection

pg library

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

- 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