Databases & ORMs

ACID

- Atomicity
- Consistency
- Isolation
- Durability

ACID

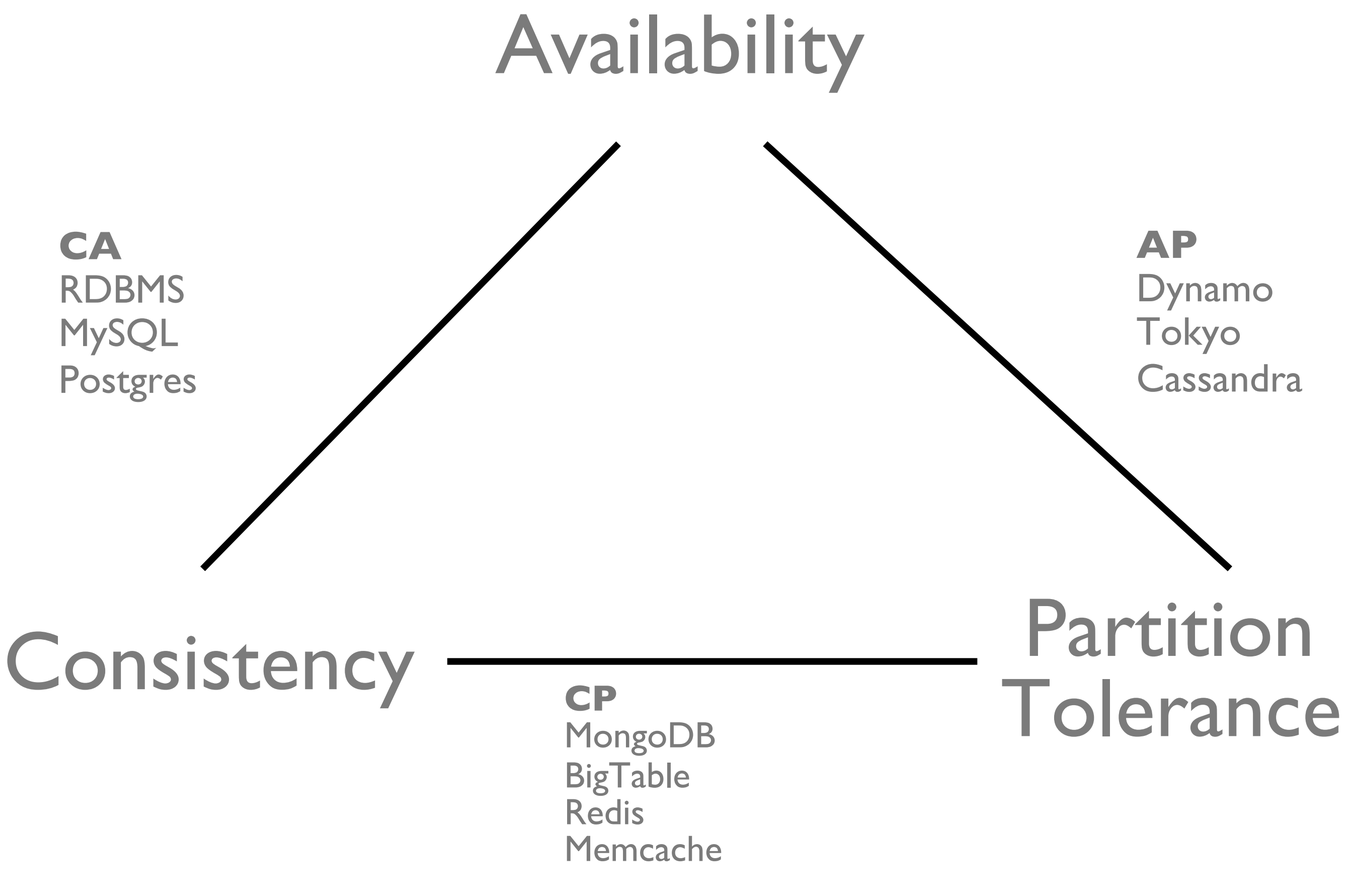
- All about transactions (key in financial transactions)
- **Atomicity** — all or nothing
- **Consistency** — must bring us to a valid state
- **Isolation** — concurrent execution that mirrors serial execution results
- **Durability** — transactions persist

CAP Theorem

- Consistency: all clients see the same data
- Availability: each client can always read or write
- Partition Tolerance: system works even if network splits

- "Pick 2"
  - (Reality is more complex…)

CAP, metaphorically speaking


Availability
CA
RDBMS
MySQL
Postgres
AP
Dynamo
Tokyo
Cassandra
Consistency
Partition
Tolerance
CP
MongoDB
BigTable
Redis
Memcache

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

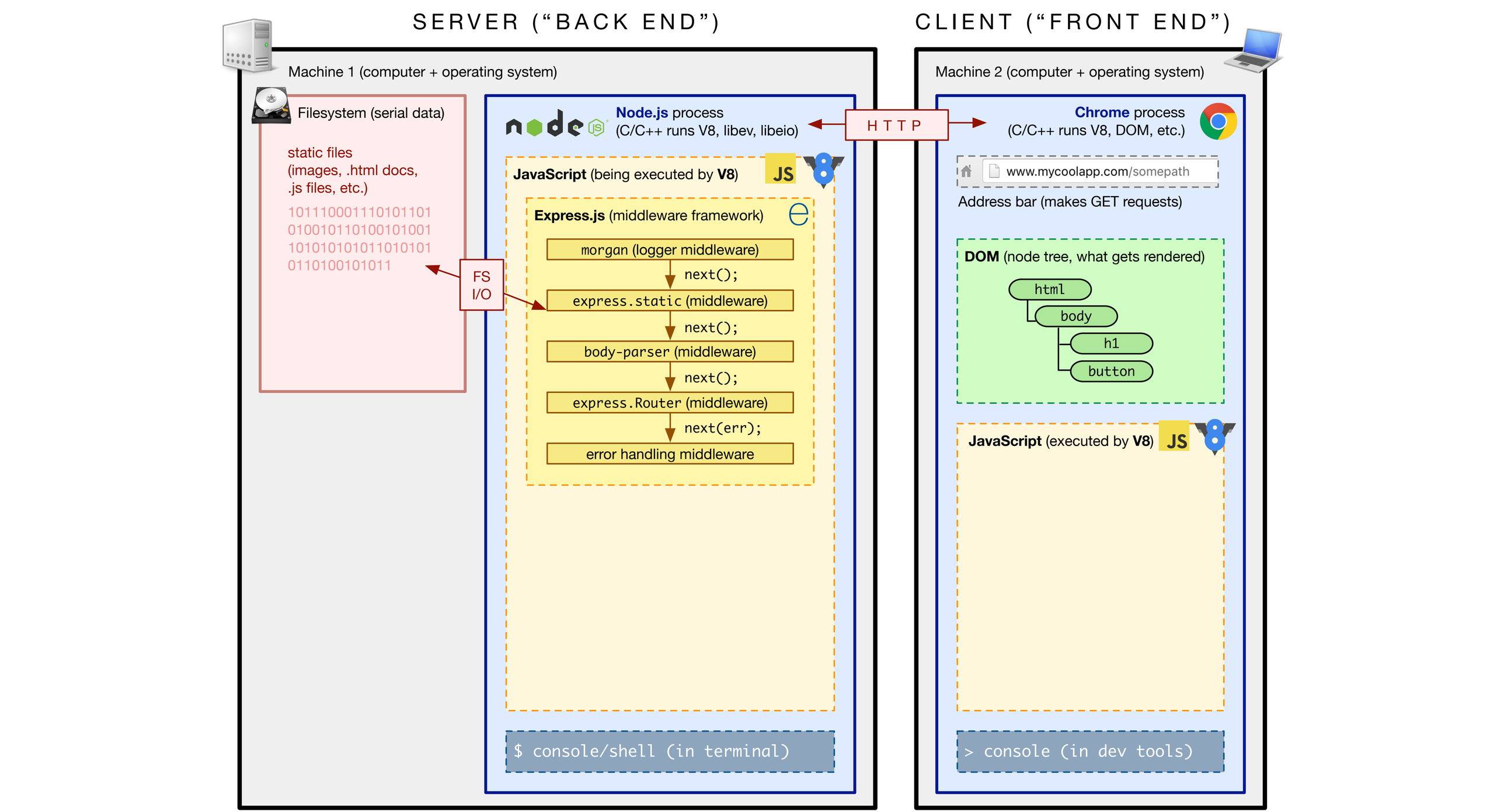
```
User.sync().then(…);
```

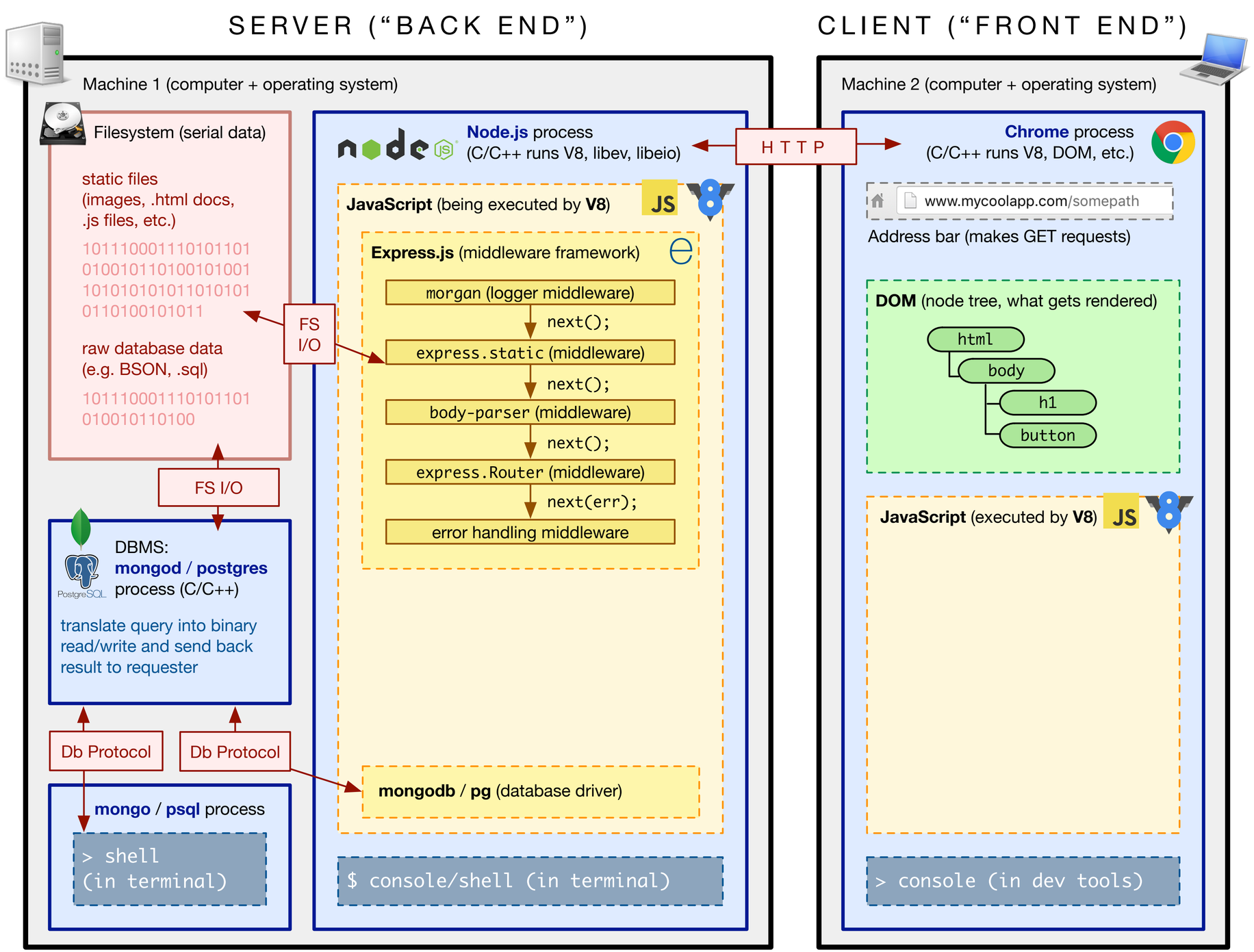
Model & Instance Usage

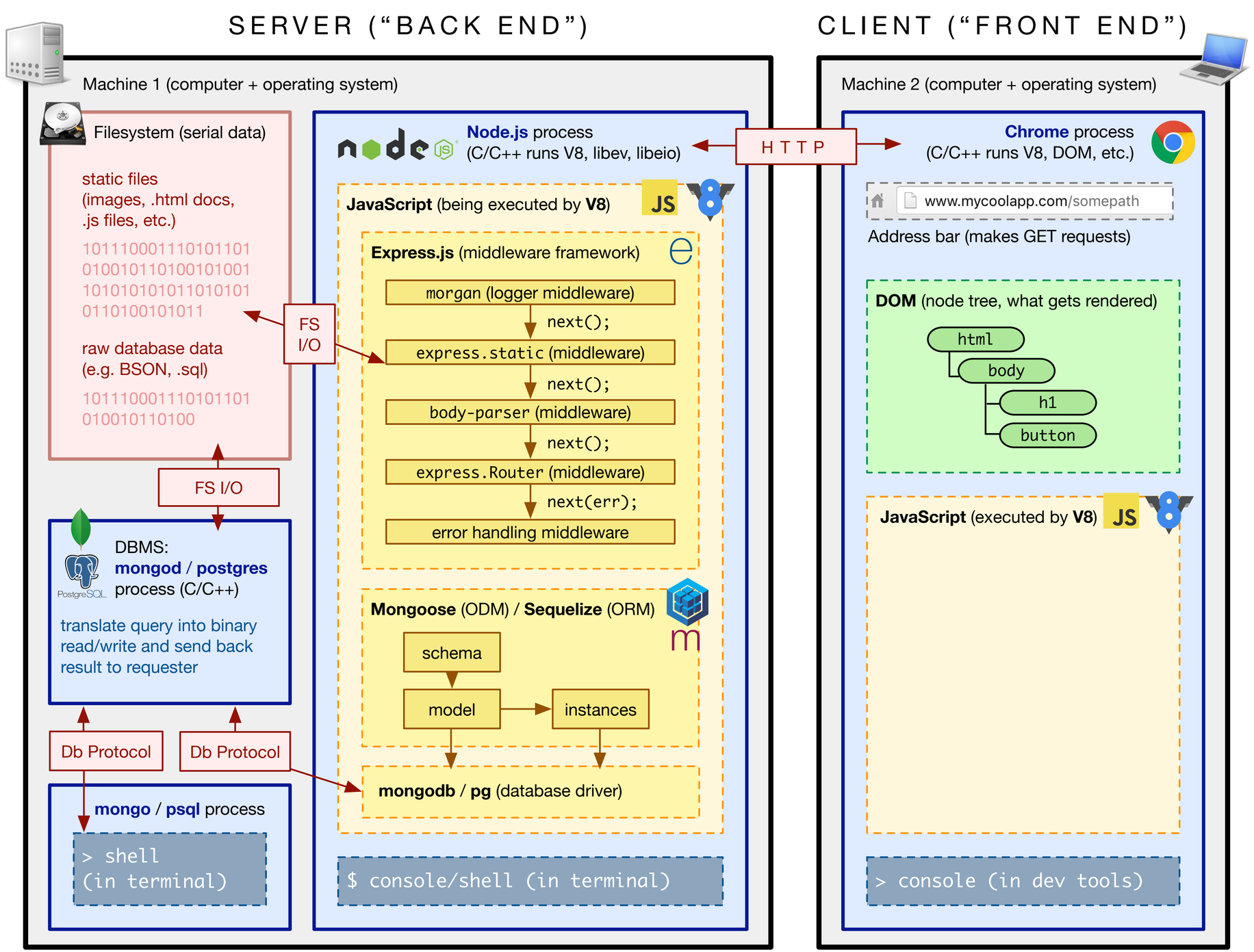
```
const person = User.build({
  name: "Kate",
  pictureUrl: "http://fillmurrary.com/10/10"
});

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)
Machine 2 (computer + operating system)
Filesystem (serial data)
static files
(images, .html docs,
.js files, etc.)
10111000111010110101001010011010010101010110101010110100101011
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
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

Sequelize — ORM

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

& 

PostgreSQL = `pg` — protocol

- 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 =>

& 

PostgreSQL = `pg` — protocol

- 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 =>

(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