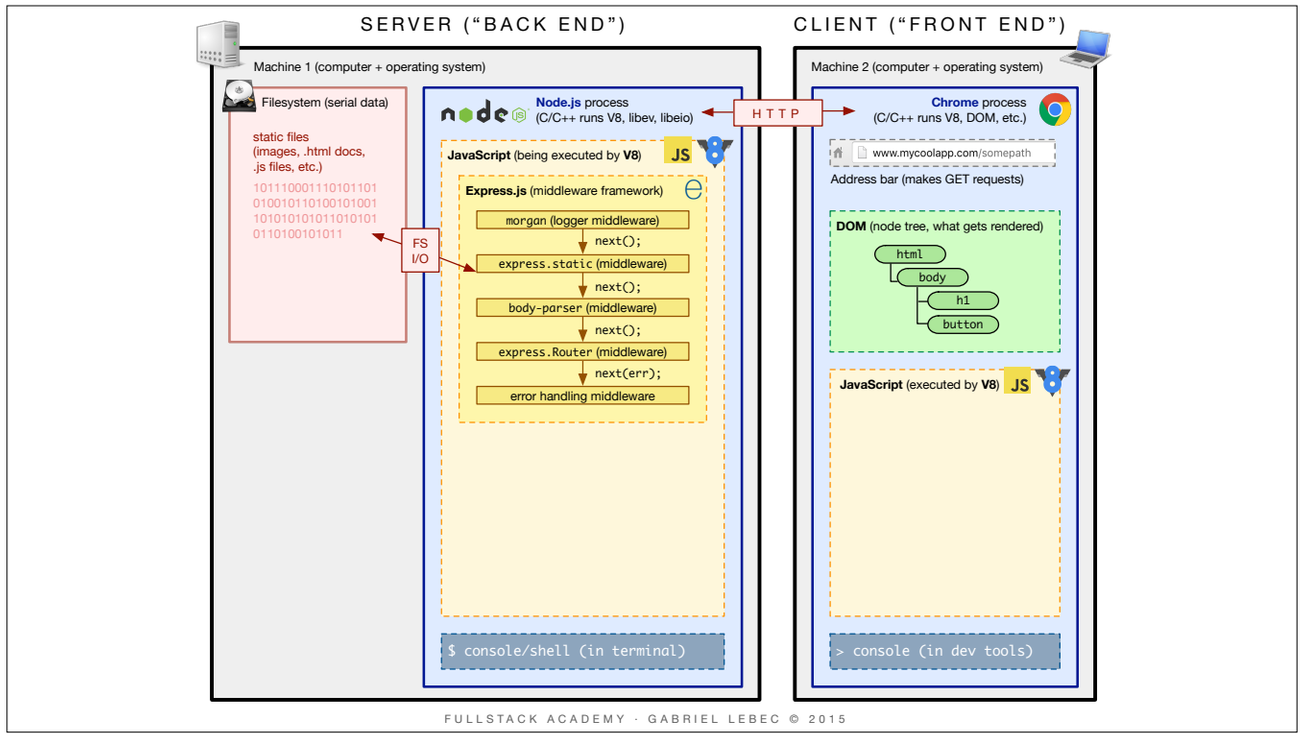
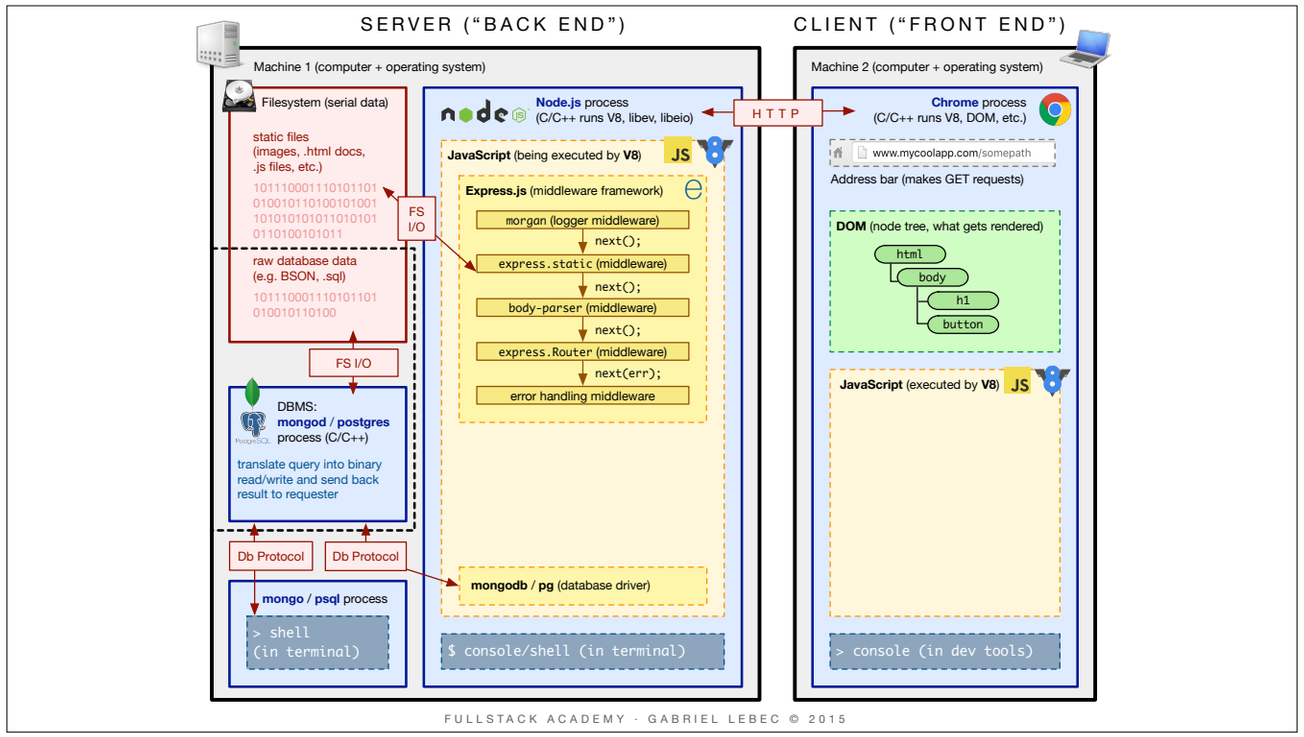
Node-Postgres

PostgreSQL client for node.js





postgres process

- The rDBMS itself; a *daemon* (background process)
- Waits for incoming SQL
- Knows how to read/write to disk in a performant way
- Sends back results

Where does the "incoming SQL" come from?

Query Sources ("Clients")

- **psql CLI**
  - human input as text
- **GUI like Postico, Datazenit**
  - human actions turned into SQL queries
- **…and other applications**
  - "somehow" communicate with the postgres process

How to transmit SQL text to app?
How can postgres be "waiting for SQL"?
And how do the results get "sent back"?

Postgres is a TCP server!

- Listening on a TCP port (5432 by default) for *requests*
- Does disk access
- Sends back a TCP *response* to the *client* that made the requests

OK, Postgres is a TCP server.
Is it... HTTP?

Postgres uses the postgres:// protocol

	Transport Protocol	Message Protocol	Content Type
Node + Express	TCP/IP	http://	Anything: HTML, JSON, XML, TXT, etc.
Postgres	TCP/IP	postgres://	SQL

For HTTP clients, the TCP/IP was handled for you by the browser or Node. How can our JS app communicate with the postgres server?

“Let's implement the postgres protocol in JavaScript ourselves!”

– AMBITIOUS MCOVERKILL

https://www.postgresql.org/docs/current/static/protocol.html

“On second thought…
has anyone done this for us?”

– SANEY MCREASONABLE

Node-postgres

- **npm library:** `npm install pg --save`
- *database driver*
- implements the postgres protocol in a Node module (JS!)
- Gives us a `client` object that we can pass SQL to
- Asynchronously talks via postgres protocol / TCP to postgres
- gives us a callback with `rows` array of resulting table

Example

```
client.query('SELECT * FROM users');
```


Here’s the basic usage.

Client.query returns a promise, so…

Example

```
const data = await client.query('SELECT * FROM users');
data.rows.forEach(function (rowObject) {
  console.log(rowObject); // { name: 'Claire' }
});
```


… We have to use the await keyword.
Then we can simply grab the data and loop over it.

Example

```
try {
  const data = await client.query('SELECT * FROM users');
  data.rows.forEach(function (rowObject) {
    console.log(rowObject); // { name: 'Claire' }
  });
} catch (err) {
  console.error(err);
}
```


Remember that async operations can throw error - if the server is offline, for example, so wrap it in a try…catch block.