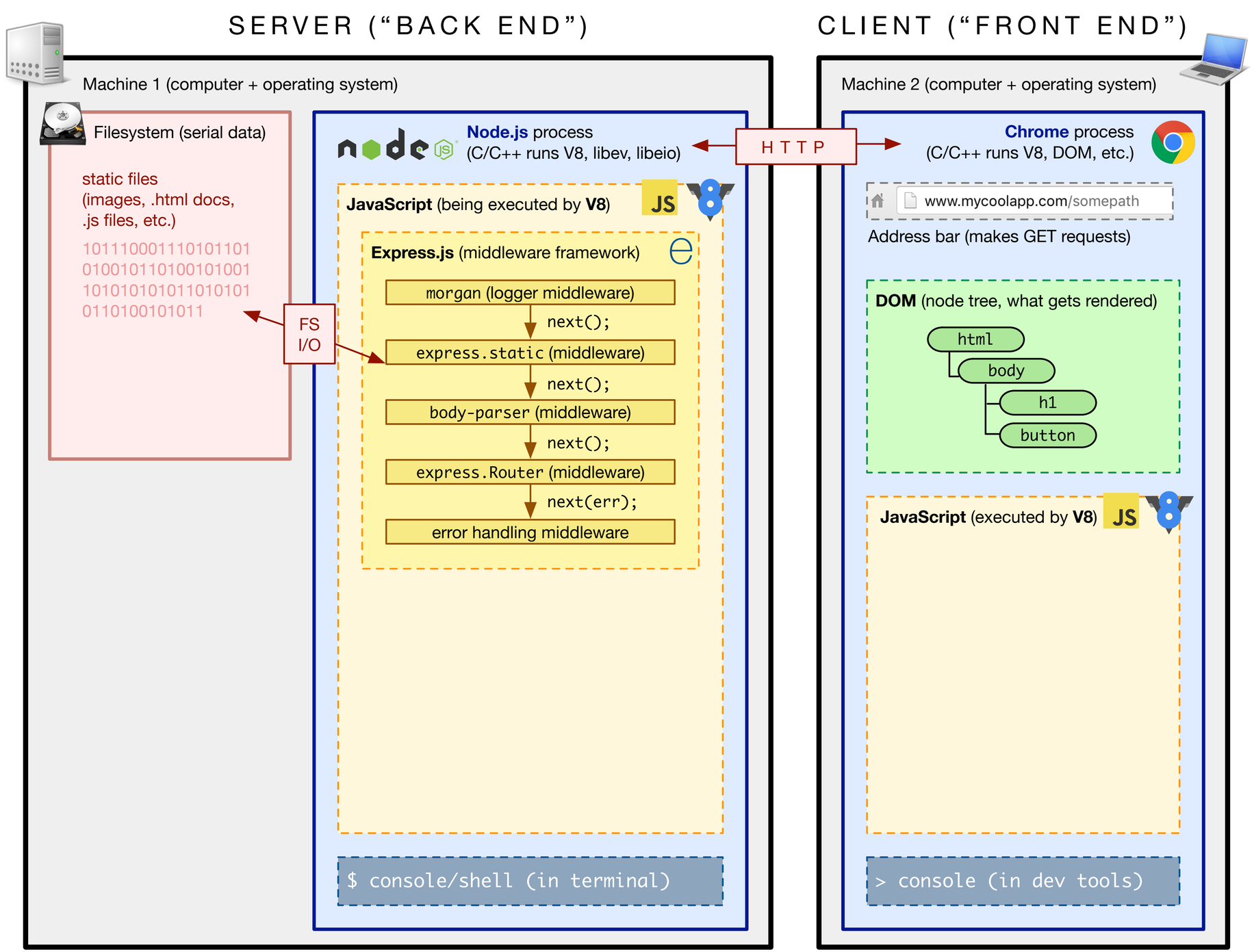
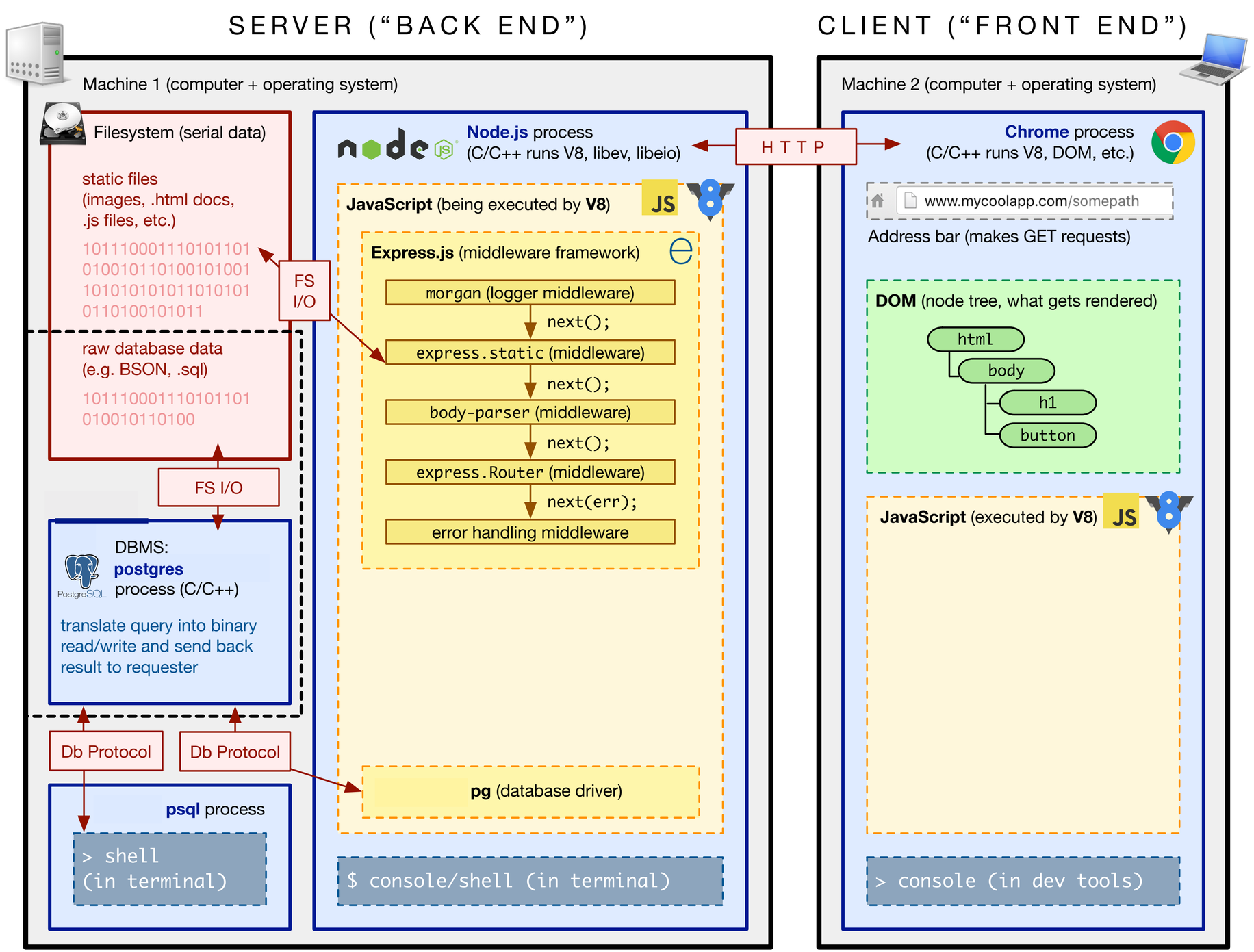
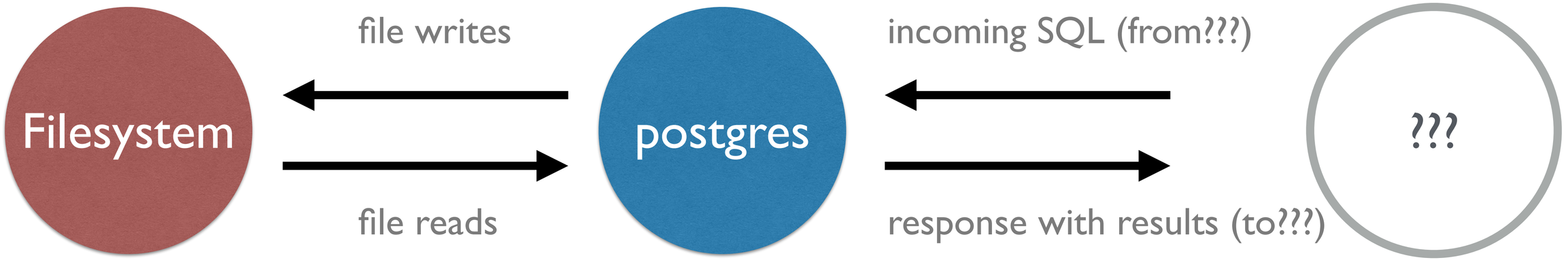
Node-Postgres (pg)

PostgreSQL client for node.js


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.)
10111000111010110101
01001011010010100 1
10101010110101010 1
0110100101011
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:
postgres
process (C/C++)
PostgreSQL
translate query into binary
read/write and send back
result to requester
Db Protocol
Db Protocol
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
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)

postgres process



- The DBMS (postgres) is a process running in the background
- Postgres is listening, waiting for incoming SQL
- Upon receiving: read/write to disk in a performant way
- Then sends back results

- …sound familiar?

Postgres is a server!

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

Query Sources ("Clients")

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

Pitstop: Networking 101

Some quick vocab

- **HTTP**
  - A messaging protocol - specifies how messages should be formatted
  - Example: an HTTP request should have a verb, a URI, headers, etc.
- **TCP ("Transmission Control Protocol")**
  - A communications protocol - specifies how messages are exchanged between devices in a network
  - Most common protocol in networks is IP (internet protocol)

- **TCP specifies how messages are *exchanged***
- **HTTP specifies how messages are *formatted***

How is this related to postgres?

- **TCP servers will accept many formats (HTTP being one of them!)**
- **The servers we have built so far in Node are TCP servers, specifically, they are HTTP servers**
  - They accept HTTP requests, and send HTTP responses
- **Postgres is also a TCP server**
  - But it doesn't use HTTP
  - It receives and sends SQL… but how?

Postgres uses the postgres:// protocol

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

With HTTP, a lot 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`
- *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

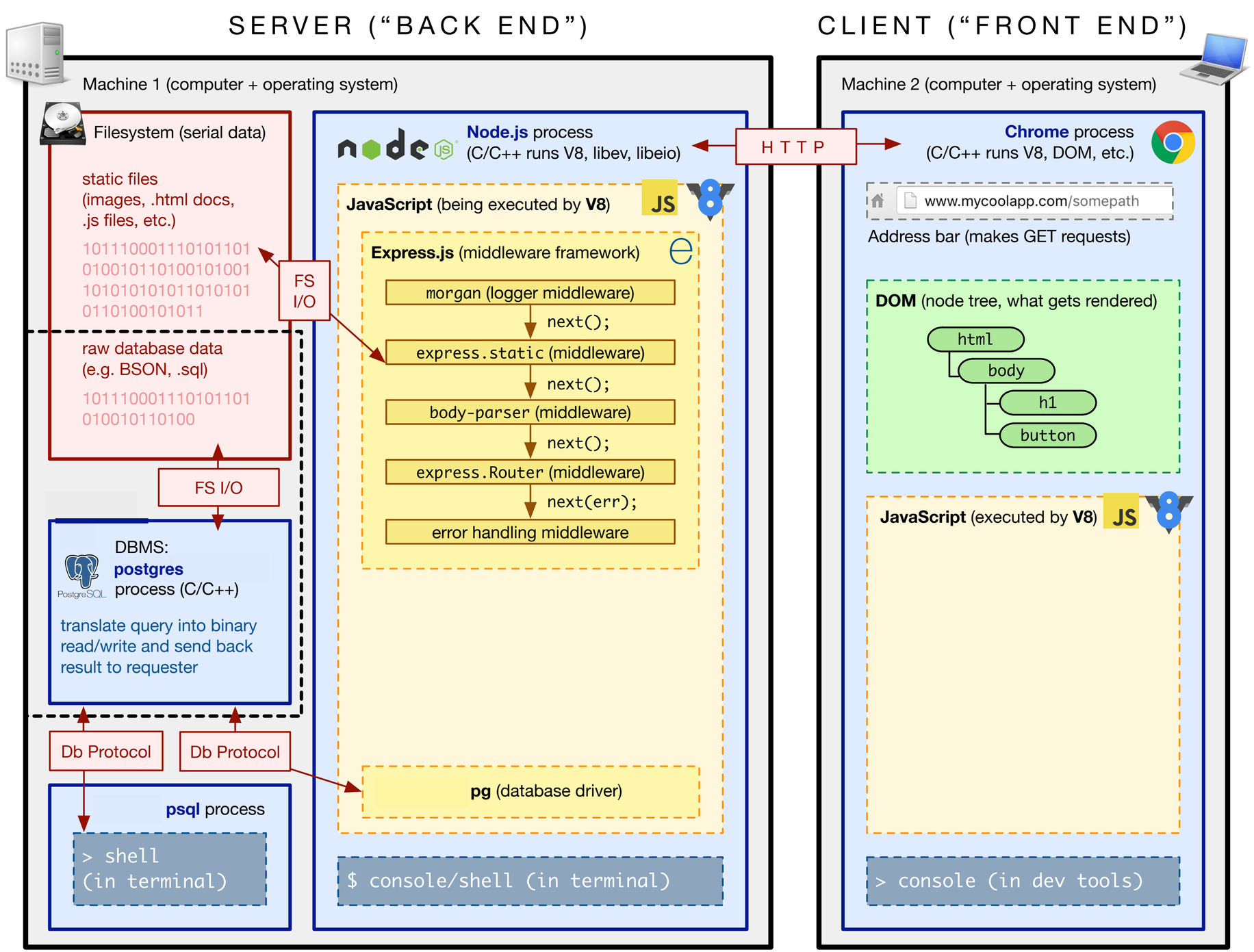
```
client.query('SELECT * FROM users;');
```

Example

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

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);
}
```


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:
postgres
process (C/C++)
PostgreSQL
translate query into binary
read/write and send back
result to requester
Db Protocol
Db Protocol
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
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)