INTRO TO PROTOCOLS

CLIENTS & SERVERS

- Client requests a resource
- Server responds with resource
- These are *roles* — not technical specs or computer types

CLIENTS & SERVERS

CLIENTS & SERVERS

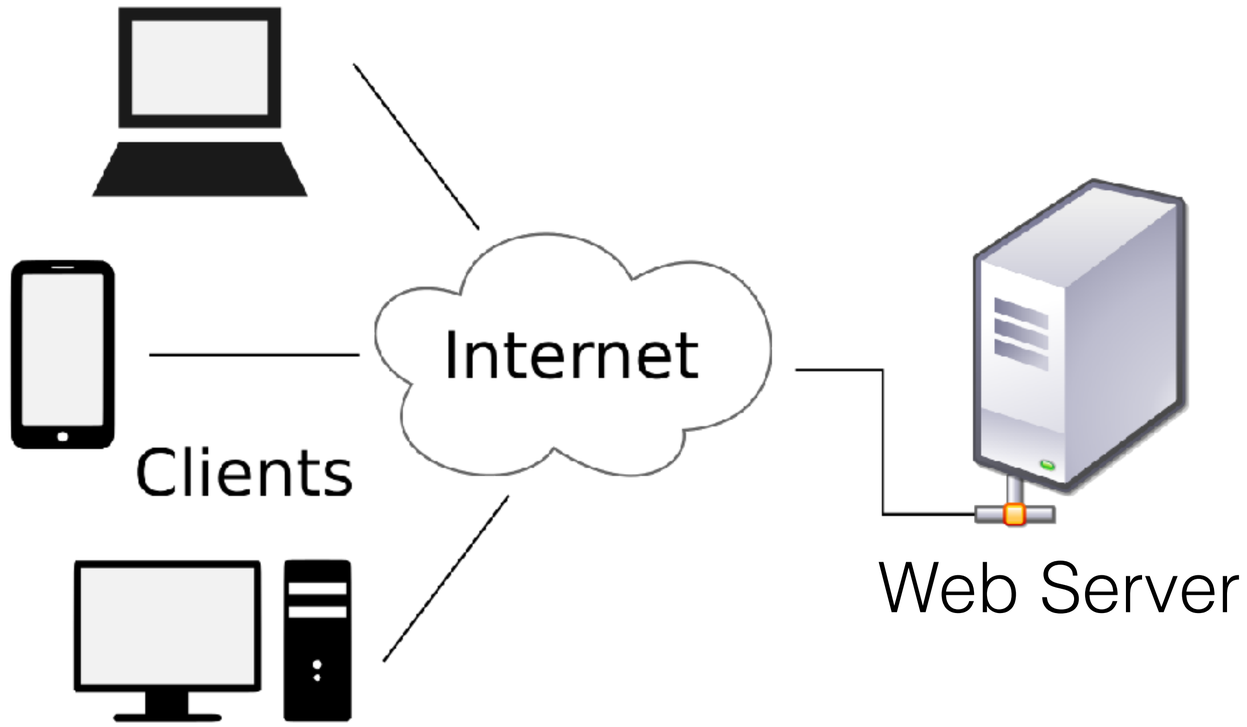
- These are *roles* — not technical specs or computer types

CLIENTS & SERVERS

CLIENTS & SERVERS

WEB SERVERS

- **Processes (running programs) not physical machines**
  - Might be running on a laptop,
  - or a Raspberry Pi,
  - or an enterprise-grade workstation…
- **Listening on a port for incoming requests**
- **Send back responses**



...but we are getting ahead of ourselves.

INTERNET COMMUNICATION PROTOCOLS

AFP

BitTorrent

TCP

HTTPS

POP

9P

IMAP

BGP

HTTP

UDP

SCP

SSL

ICMP

FTP

SSH

IP

PPP

NFS

SMTP

PROTOCOL

- Rules for interaction / communication
- Specification, not implementation

PROTOCOL

MESSAGING / APP VS. TRANSMISSION

- KnockKnock is an *application level* protocol
- It specifies the sequence and content of messages
- It does NOT specify how those messages are transmitted

HTTP

HTTP

- An *application-level* communications protocol. You might call it a *messaging* protocol.
- Specifies allowable *metadata* and *content* of messages.
- Does *NOT* specify *how* messages are transmitted!
- STATELESS: does *not* need to remember previous req-res!

HTTP PROTOCOL

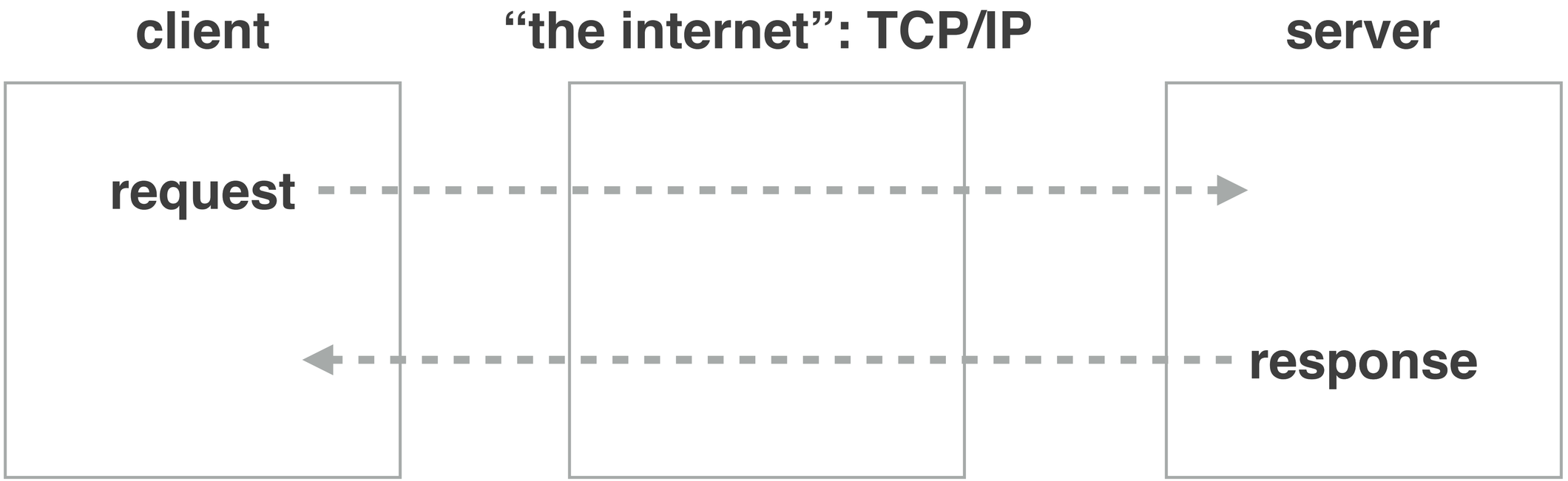
- **RFC (Request For Comments) 7230 (link)**
- **By the IETF (Internet Engineering Task Force)**
- **But a *generic* messaging protocol**
  - *"HTTP is a generic interface protocol for information systems. It is designed to hide the details of how a service is implemented... independent of the types of resources provided."*

HTTP CLIENTS & SERVERS

- **Example Clients**
  - web browsers
  - household appliances
  - stereos
  - firmware update scripts
  - command-line programs
  - mobile apps
  - communication devices
- **Example Servers**
  - web servers
  - home automation units
  - networking components
  - office machines
  - autonomous robots
  - news feeds
  - traffic cameras

NOT A TRANSMISSION PROTOCOL!

HTTP OVER TCP/IP



HTTP

Every request gets exactly one (total) response (sometimes a response is broken up into chunks)

HTTP REQUEST

just a message with a certain format

verb URI

headers

```
POST /docs/1/related HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

bookId=12345&author=Nimit
```

body

(from http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

COMMON VERBS

GET	“read”
POST	“create”
PUT	“update”
DELETE	“delete”

HTTP RESPONSE

status

headers

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

(from http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

payload/body

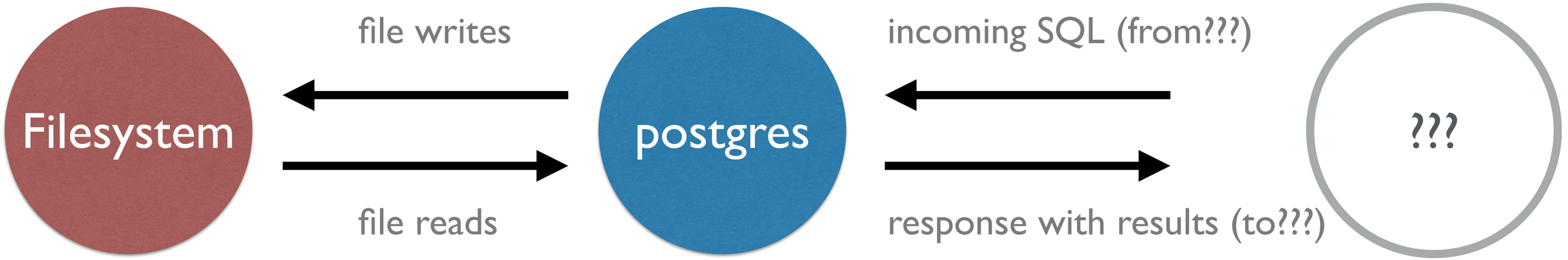
COMMON STATUSES

200	“OK”
201	“created”
304	“cached”
400	“bad request”
401	“unauthorized”
404	“not found”
500	“server error”

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

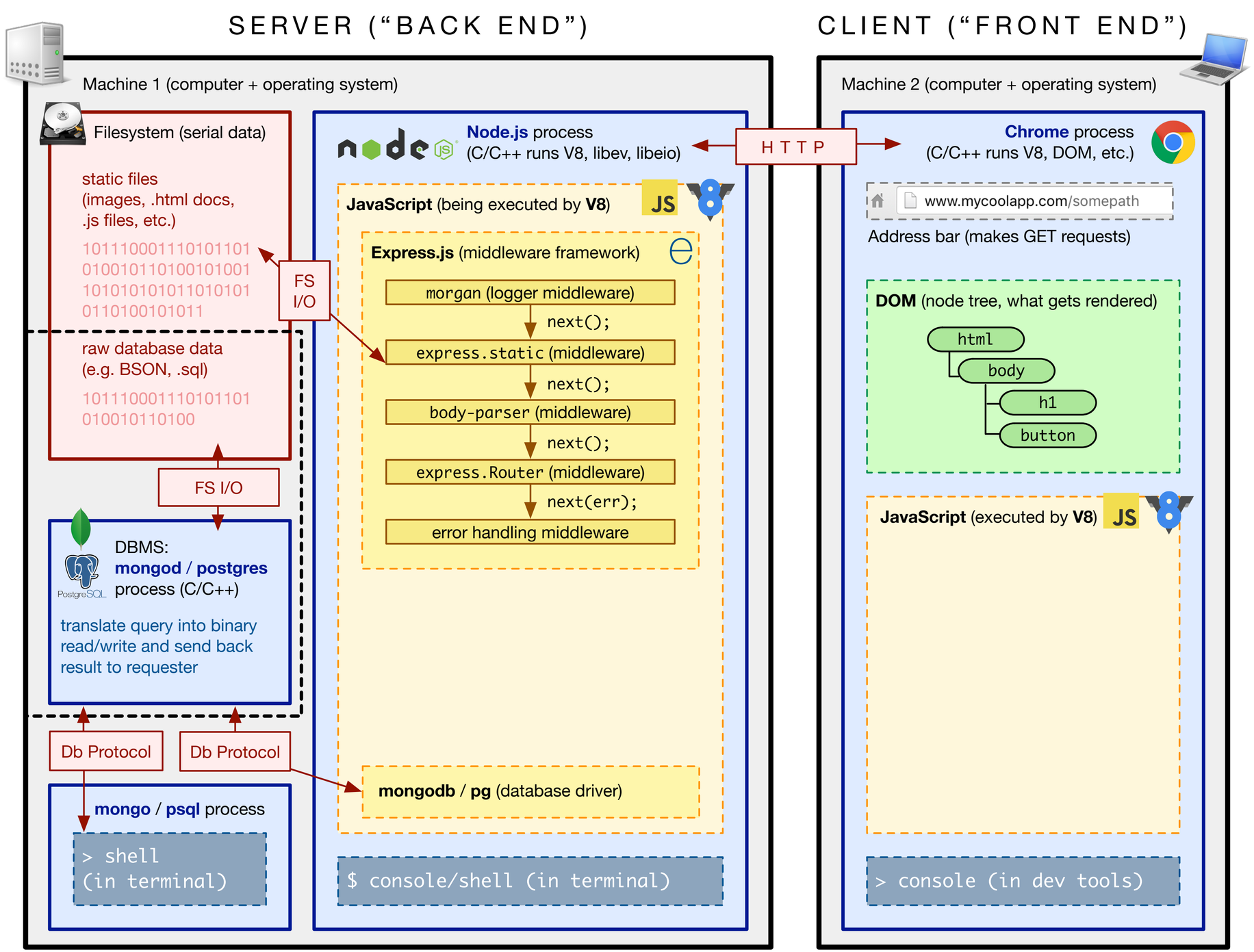
```
client.query('SELECT * FROM users;');
```

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:
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)