Promises, Async, Node

Leon Noel

Photo shoot fresh, looking like wealth
I'm 'bout to call the paparazzi on myself

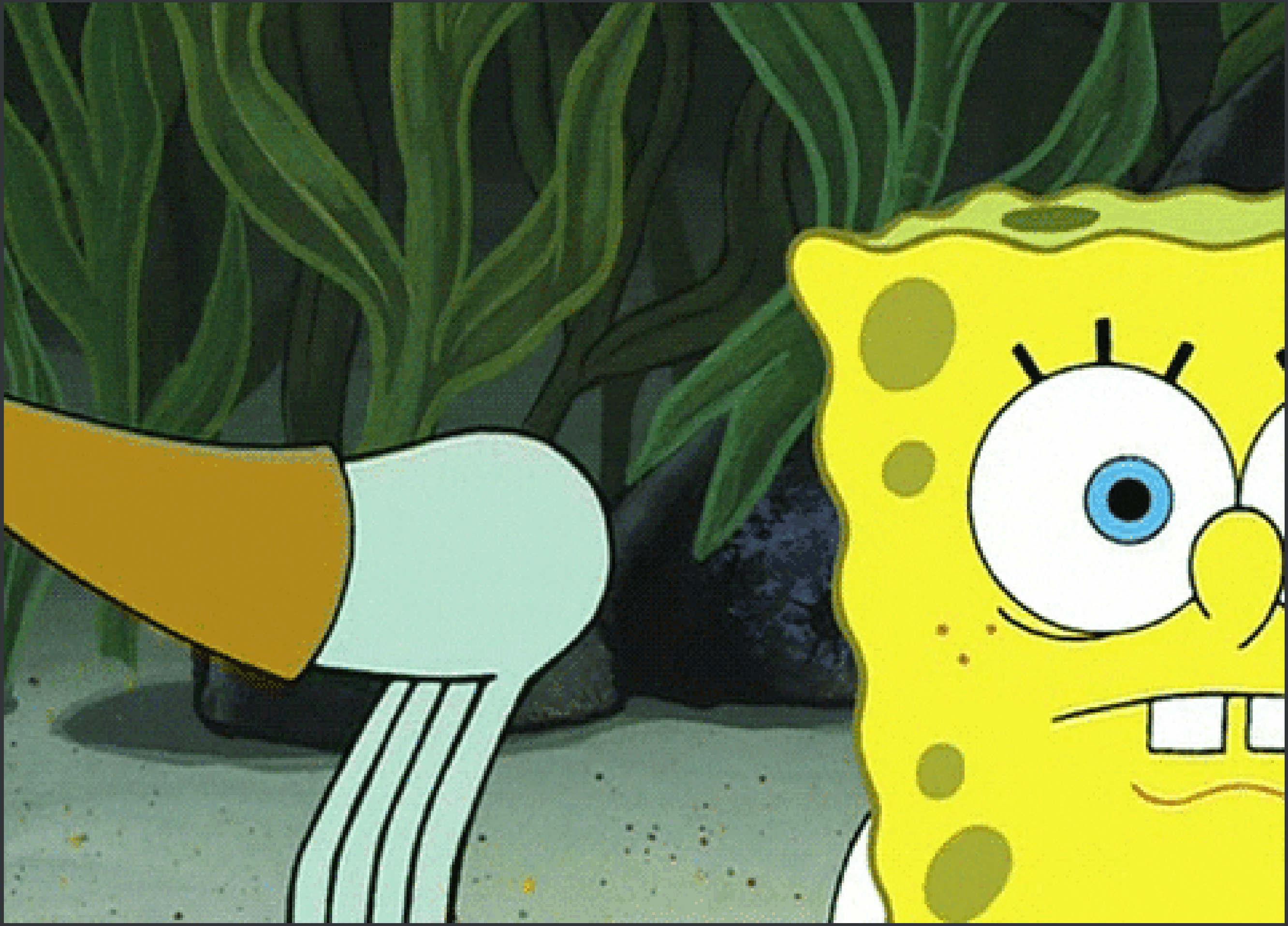
Agenda

- Questions?
- Let's Talk - Catch Up
- Learn - Callbacks
- Learn - Promises
- Learn - Async
- Learn - Node
- Homework - Simple Coin Flip

Questions

About last class or life

Backend!



Butt first!

Let's Deliver Some Papers

Synchronous

Waiting for them to come to the door

Asynchronous

Moving onto the next house

Javascript is single-threaded

Synchronous aka processes one operation at a time

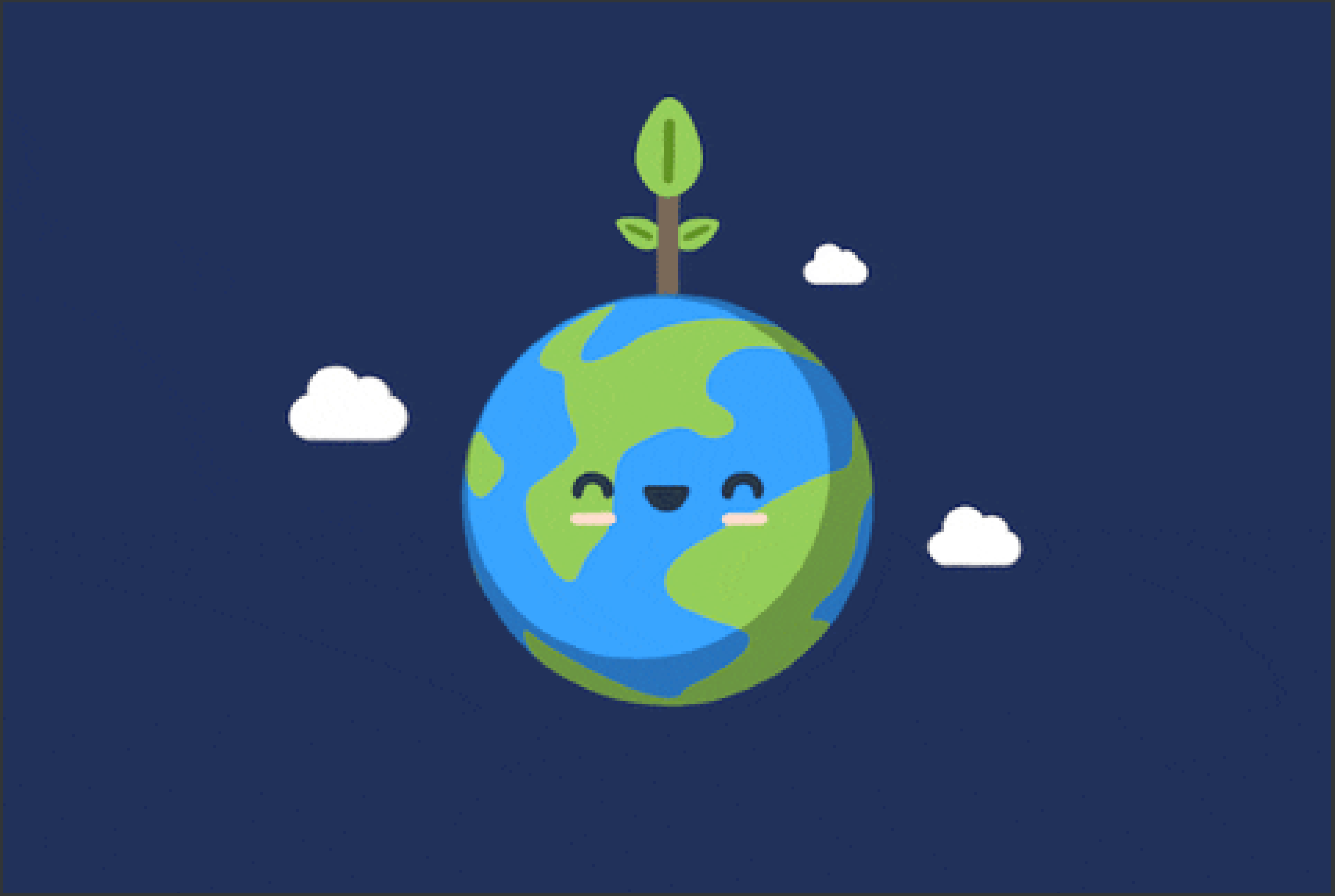
vs

If synchronous, how do we do stuff like make an api request and keep scrolling or clicking

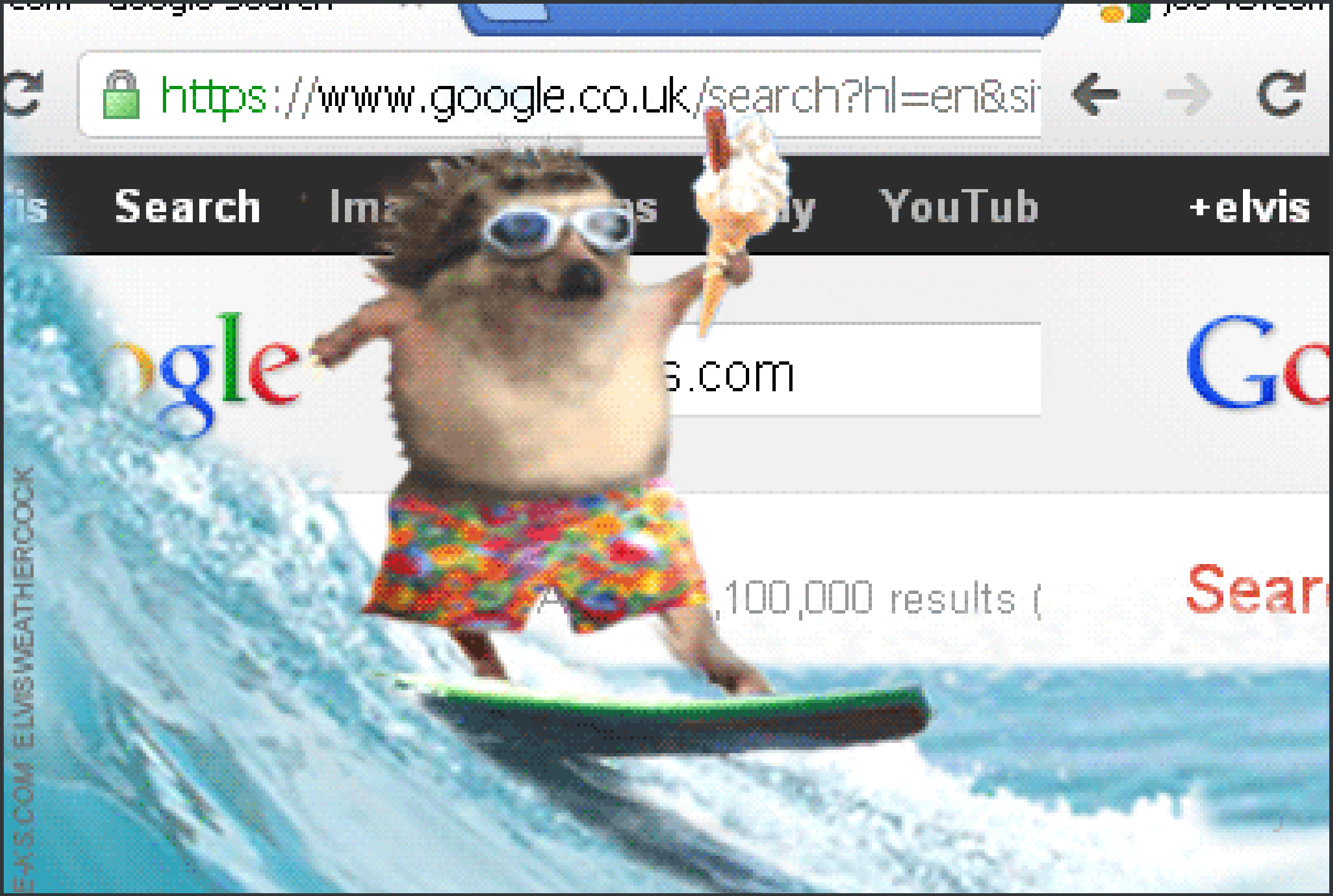
Things should block

THE ENVIRONMENT

Not This



THIS


https://www.google.co.uk/search?hl=en&si
Search
YouTub
+elvis
.com
,100,000 results (
ELVISWEATHERCOCK
E-K.S.COM

Our JS is running in a browser

Browsers have a BUNCH of APIs we can use that are async and enable us to keeping looking a cute cat photos while those operations are being processed asynchronously

Common browser APIs

DOM (Document Object Model) API

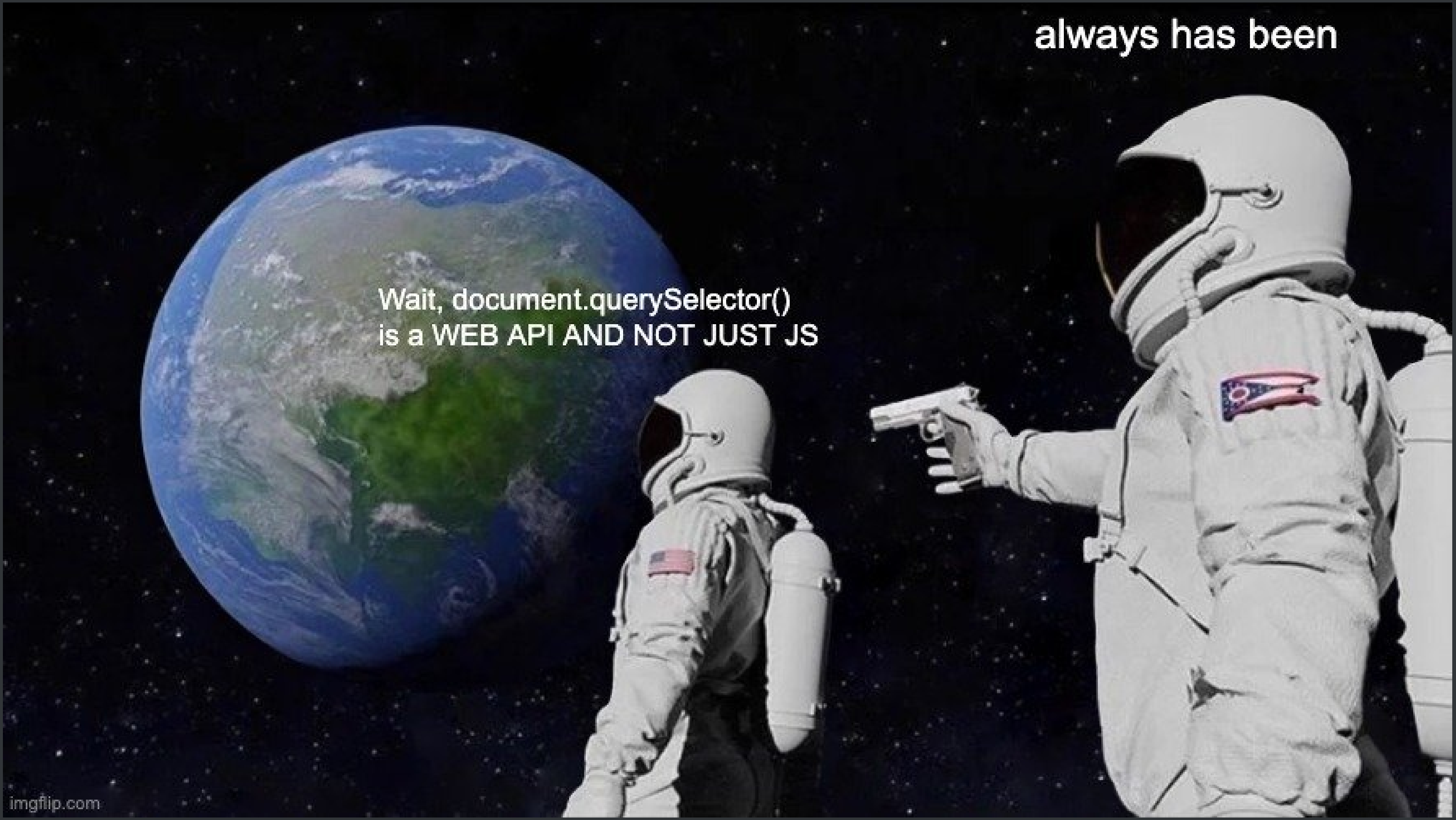
*the DOM (Document Object Model) is essentially the API one uses to manipulate an HTML (or XML) document -- usually using JavaScript

*stackoverflow from 10 years ago...

! USUALLY !

WAIT
WHAT THE FUCK

Actual words Leon said when figuring all this shit out...


Wait, document.querySelector()
is a WEB API AND NOT JUST JS
always has been
imgflip.com

So, yeah, JS can do a lot of "blocking" stuff in the browser because it is handing that stuff off to async Web APIs

BUT

We are going to need to know how to handle responses coming back from those Web APIs

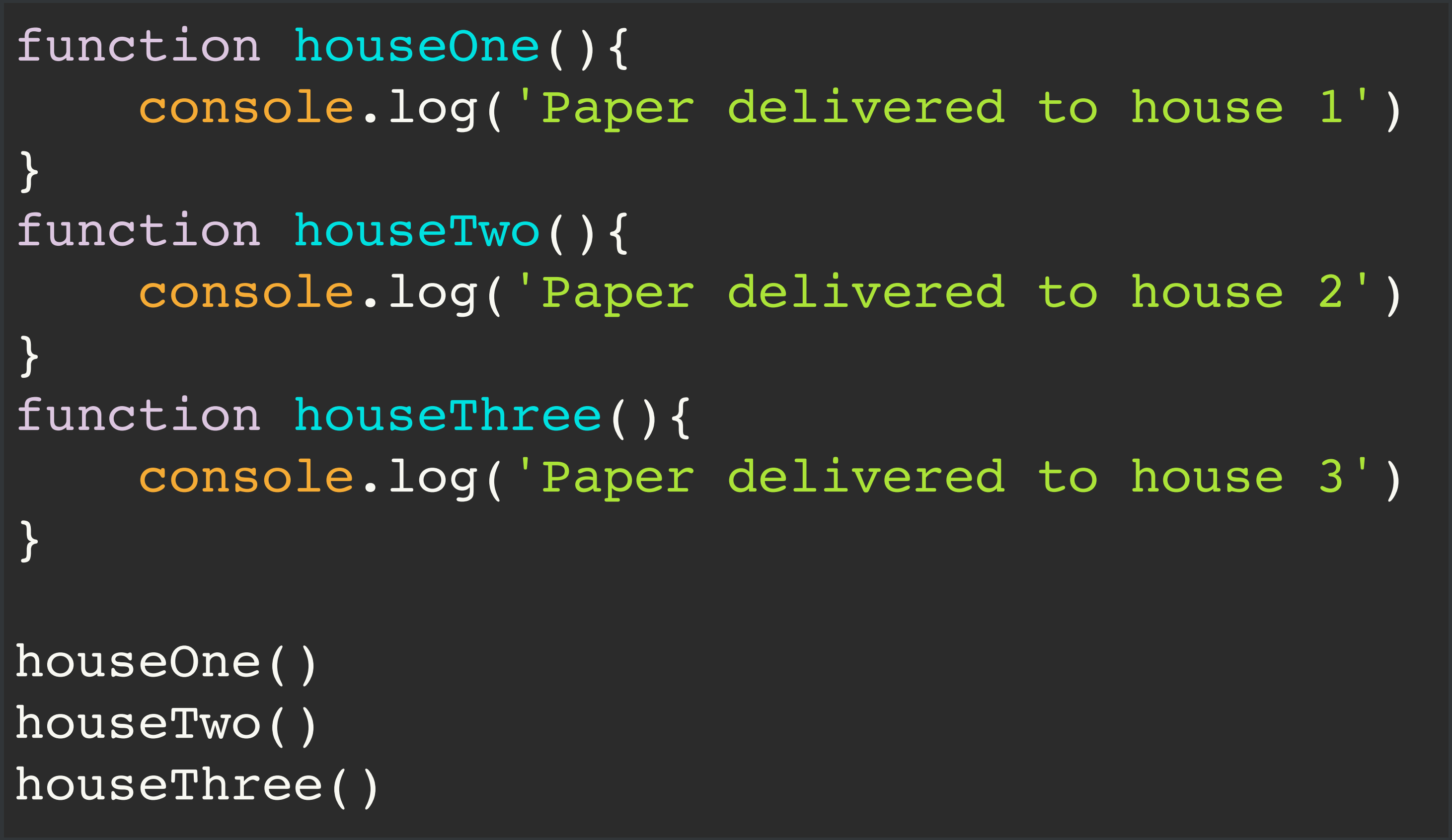
JS does this with callbacks, promises,

and eventually async/await

Call stack, Call Back Queue, Web API, Event Loop

Thursday

Let's Deliver Some Papers



```
function houseOne(){
    console.log('Paper delivered to house 1')
}
function houseTwo(){
    console.log('Paper delivered to house 2')
}
function houseThree(){
    console.log('Paper delivered to house 3')
}

houseOne()
houseTwo()
houseThree()
```

Let's Use A Web API

setTimeout()

setTimeout and setInterval are not part of the Javascript specification...

Most environments include them...

like all browsers and Node.js

Live Leon Footage

```javascript
function houseOne(){
    console.log('Paper delivered to house 1')
}
function houseTwo(){
    setTimeout(() => console.log('Paper delivered to house 2'), 3000)
}
function houseThree(){
    console.log('Paper delivered to house 3')
}
houseOne()
houseTwo()
houseThree()
```

```
function houseOne(){
    console.log('Paper delivered to house 1')
}
function houseTwo(){
    setTimeout(() => console.log('Paper delivered to house 2'), 0)
}
function houseThree(){
    console.log('Paper delivered to house 3')
}
houseOne()
houseTwo()
houseThree()
```

EVENT LOOP

Thursday

What if it is pay day?

I only want to move onto the third house after the second house has paid me

Real world this would be getting data back from an API ect...

Callbacks

The Old School Way

You can have a function that takes another function as an argument

aka Higher Order Function

You have seen this a million times

```
addEventListener('click', callback)
```

A Callback is the function that has been passed as an argument

Callbacks are not really "a thing" in JS
just a convention

Let's Get Paid

```
function houseOne(){
    console.log('Paper delivered to house 1')
}
function houseTwo(callback){
    setTimeout(() => {
        console.log('Paper delivered to house 2')
        callback()
    }, 3000)
}
function houseThree(){
    console.log('Paper delivered to house 3')
}

houseOne()
houseTwo(houseThree)
```

Callback fires when async task or another function is done

Let's Get Paid By Everyone

```
function houseOne(){
    setTimeout(() => {
        console.log('Paper delivered to house 1')
        setTimeout(() => {
            console.log('Paper delivered to house 2')
            setTimeout(() => {
                console.log('Paper delivered to house 3')
            }, 3000)
        }, 4000)
    }, 5000)
}
houseOne()
```

Welcome To Hell

Callback Hell

What if there was a more readable way to handle async code

Promise

A promise is an object that represents the eventual completion or failure of an async operation and its value

An object that MAY have a value in the future

A promise can have three possible states

- *pending*: initial state, neither fulfilled nor rejected.
- *fulfilled*: meaning that the operation was completed successfully.
- *rejected*: meaning that the operation failed.

.then()

A promise object method that runs after the promise "resolves"

`.then(value)`

Whatever value the promise object has gets passed as an argument

We've Seen This Before

APIs

Fetch Fido, Fetch!

```
fetch("https://dog.ceo/api/breeds/image/random")
    .then(res => res.json()) // parse response as JSON
    .then(data => {
      console.log(data)
    })
    .catch(err => {
        console.log(`error ${err}`)
    });
```

API returns a JSON object that we can use within our apps

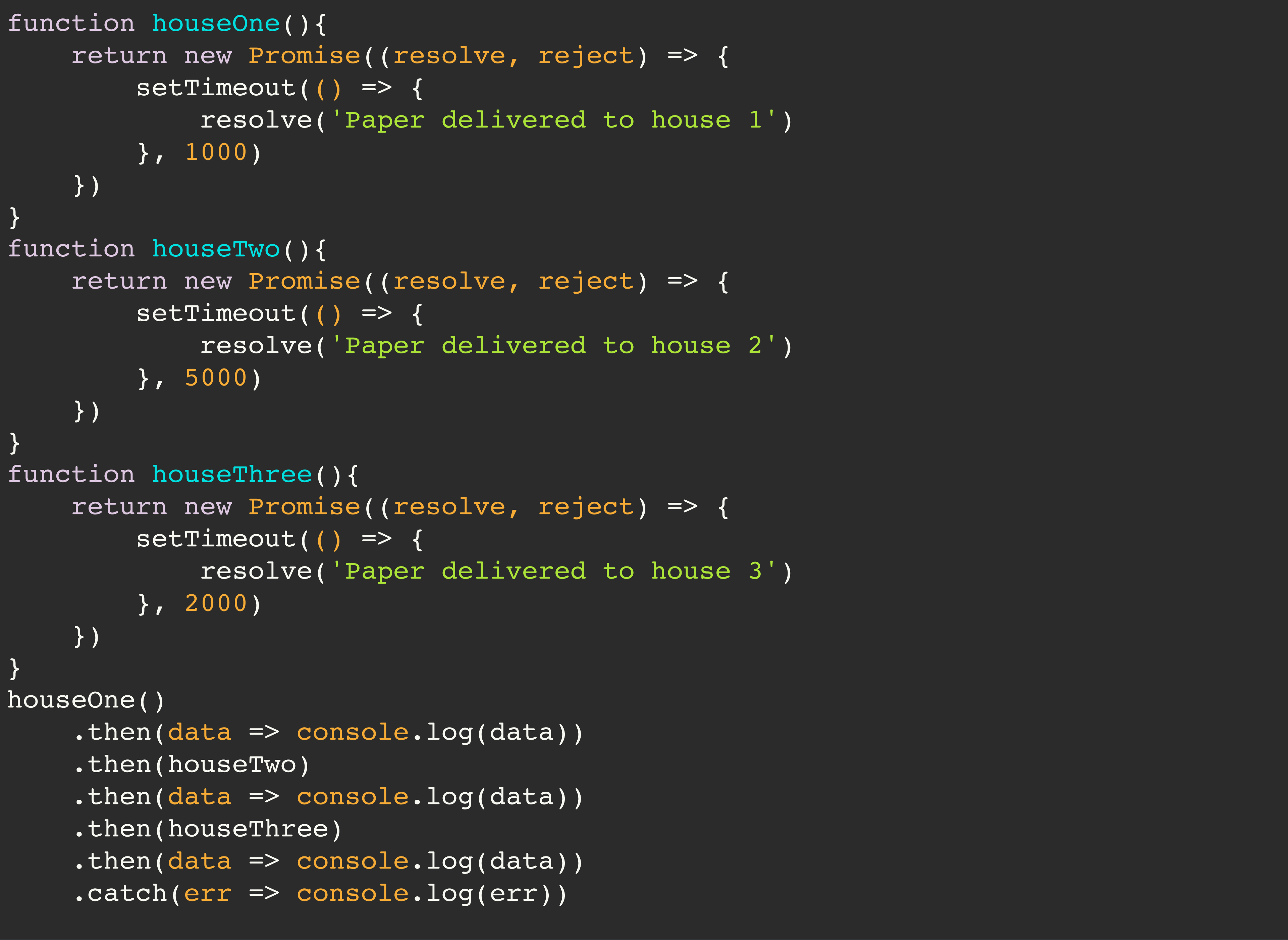
Fetch returns a Promise

Like a bunch of Web APIs running async code

Let's see those three states

```
const promise = new Promise((resolve, reject) => {
    const error = false
    if(!error){
        resolve('Promise has been fullfilled')
    }else{
        reject('Error: Operation has failed')
    }
})
console.log(promise)
promise
    .then(data => console.log(data))
    .catch(err => console.log(err))
```

Let's Get Paid By Everyone



```javascript
function houseOne(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Paper delivered to house 1')
        }, 1000)
    })
}
function houseTwo(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Paper delivered to house 2')
        }, 5000)
    })
}
function houseThree(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Paper delivered to house 3')
        }, 2000)
    })
}
houseOne()
    .then(data => console.log(data))
    .then(houseTwo)
    .then(data => console.log(data))
    .then(houseThree)
    .then(data => console.log(data))
    .catch(err => console.log(err))
```

Chaining Don't Read Good

I want my asynchronous code to look sychronous

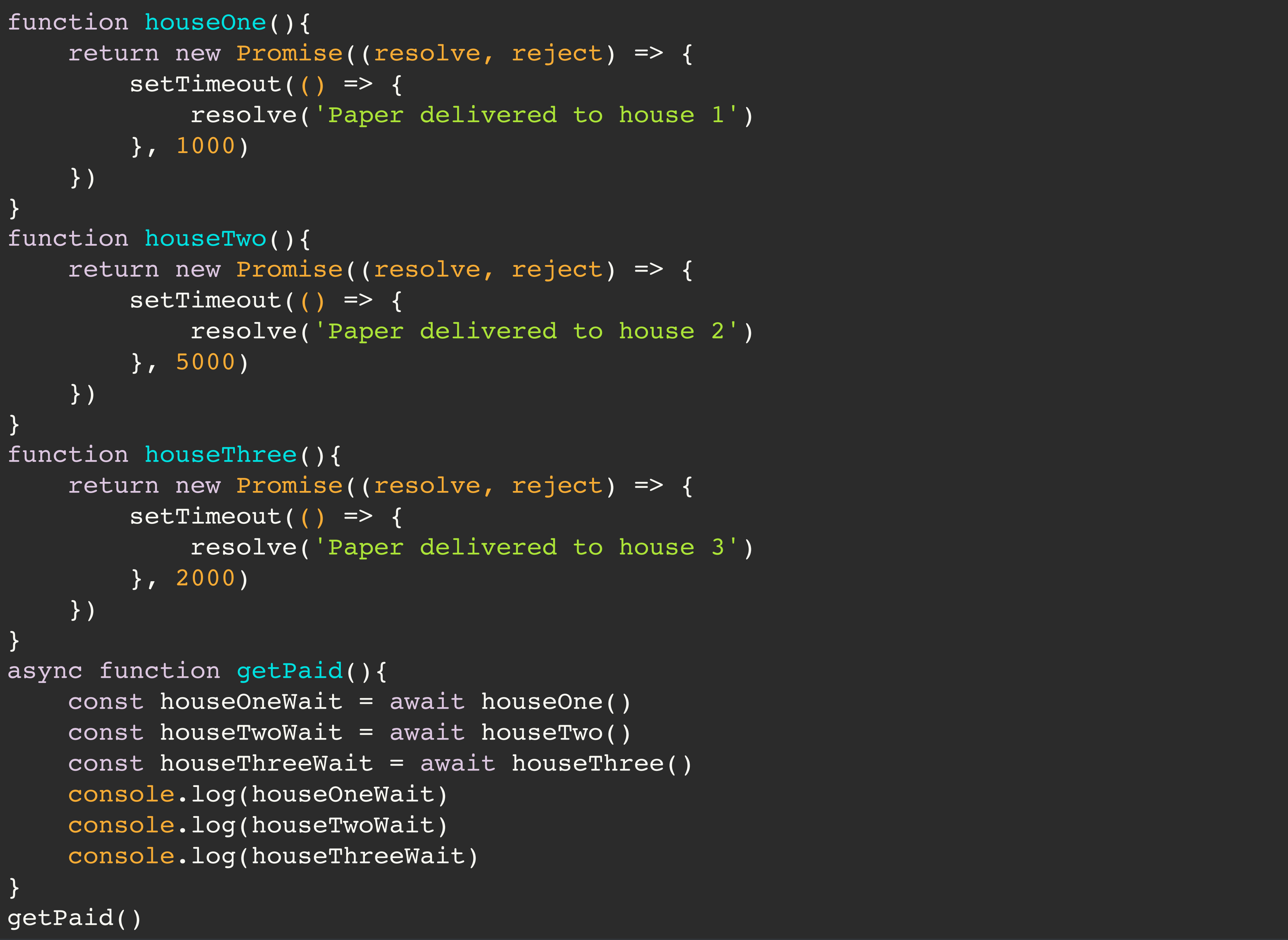
Async / Await

A way to handle async responses

Promises Under The Hood

Syntactic sugar on top of promises, making asynchronous code easier to write and to read afterwards

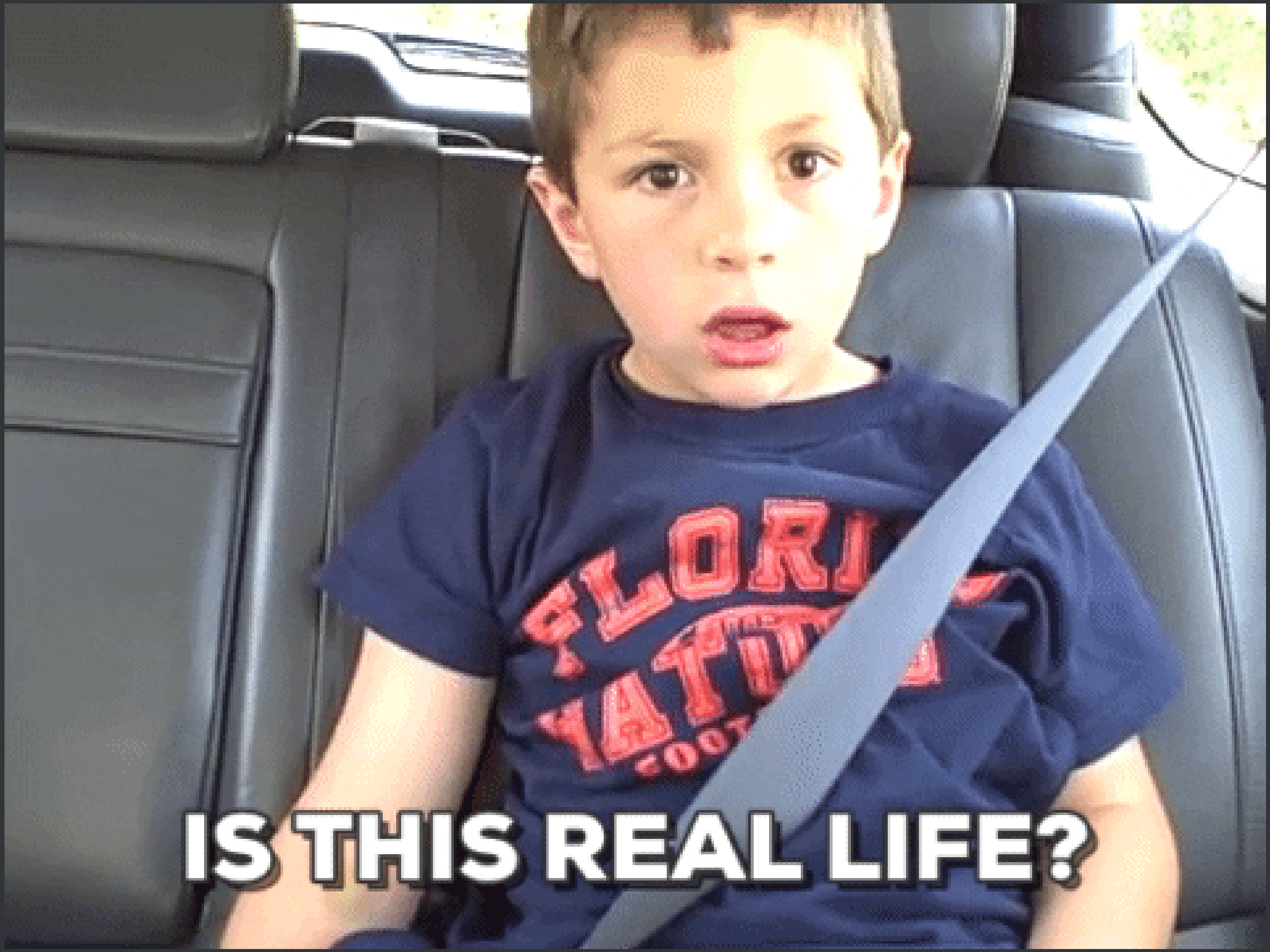
Await waits for an async process to complete inside an Async Function



```
function houseOne(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Paper delivered to house 1')
        }, 1000)
    })
}
function houseTwo(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Paper delivered to house 2')
        }, 5000)
    })
}
function houseThree(){
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Paper delivered to house 3')
        }, 2000)
    })
}
async function getPaid(){
    const houseOneWait = await houseOne()
    const houseTwoWait = await houseTwo()
    const houseThreeWait = await houseThree()
    console.log(houseOneWait)
    console.log(houseTwoWait)
    console.log(houseThreeWait)
}
getPaid()
```

```javascript
async function getPaid(){
    const houseOneWait = await houseOne()
    const houseTwoWait = await houseTwo()
    const houseThreeWait = await houseThree()
    console.log(houseOneWait)
    console.log(houseTwoWait)
    console.log(houseThreeWait)
}
getPaid()
```

I Need Something Real



Let's Code

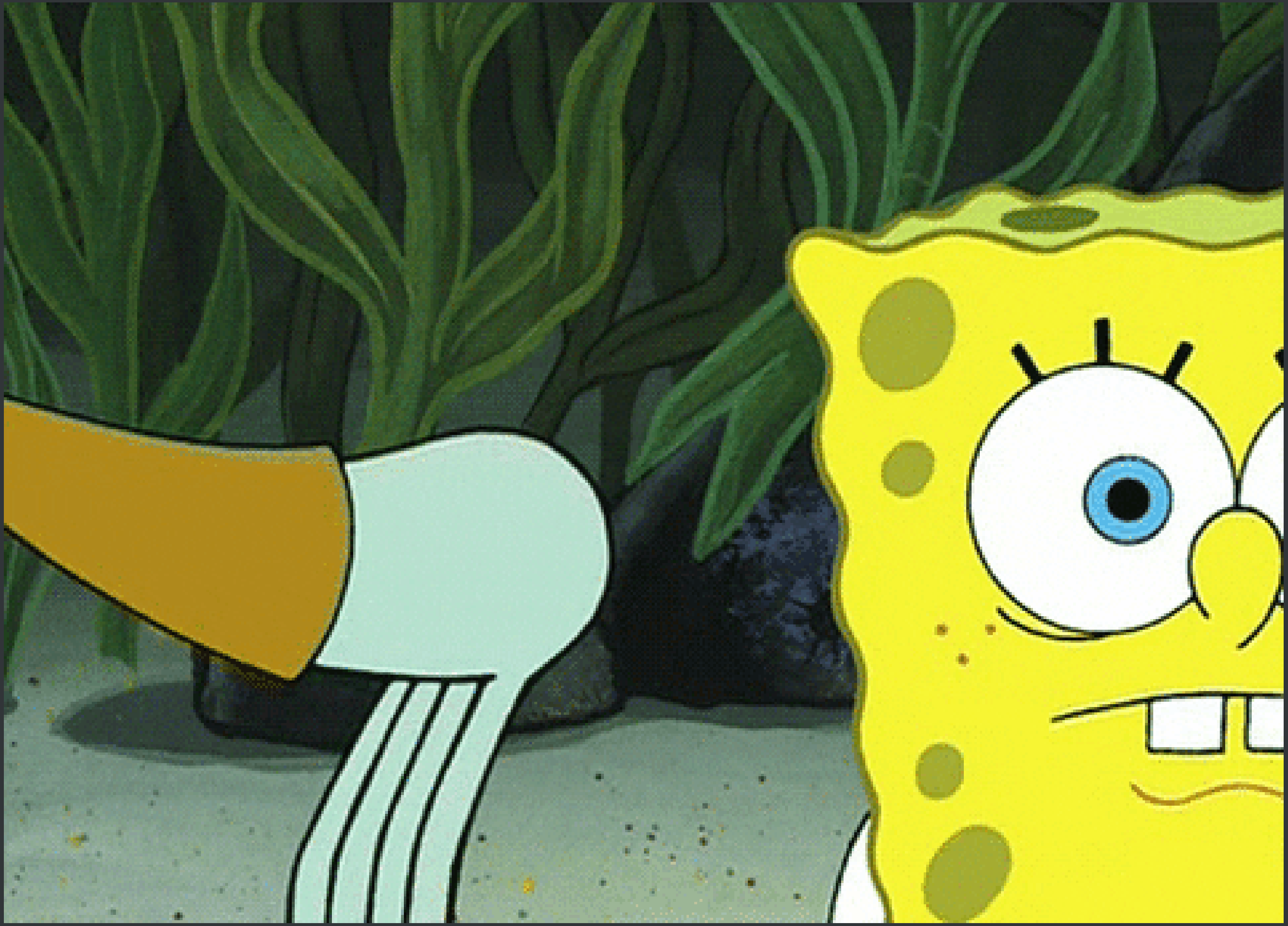
An API request using Async/Await

APIs

Fetch Fido, Fetch!

```
async function getACuteDogPhoto(){
    const res = await fetch('https://dog.ceo/api/breeds/image/random')
    const data = await res.json()
    console.log(data)
}
getACuteDogPhoto()
```

Backend BABY



Does Javascript have access to the DOM natively (built in)?

Javascript needed Web APIs to handle async and a bunch of stuff in the Browser

JS is sandboxed in the browser

JS is a language that can only do what the hosting environment allows

What Do Servers Need?

Disk Access

(hardrive/ssd)

&&

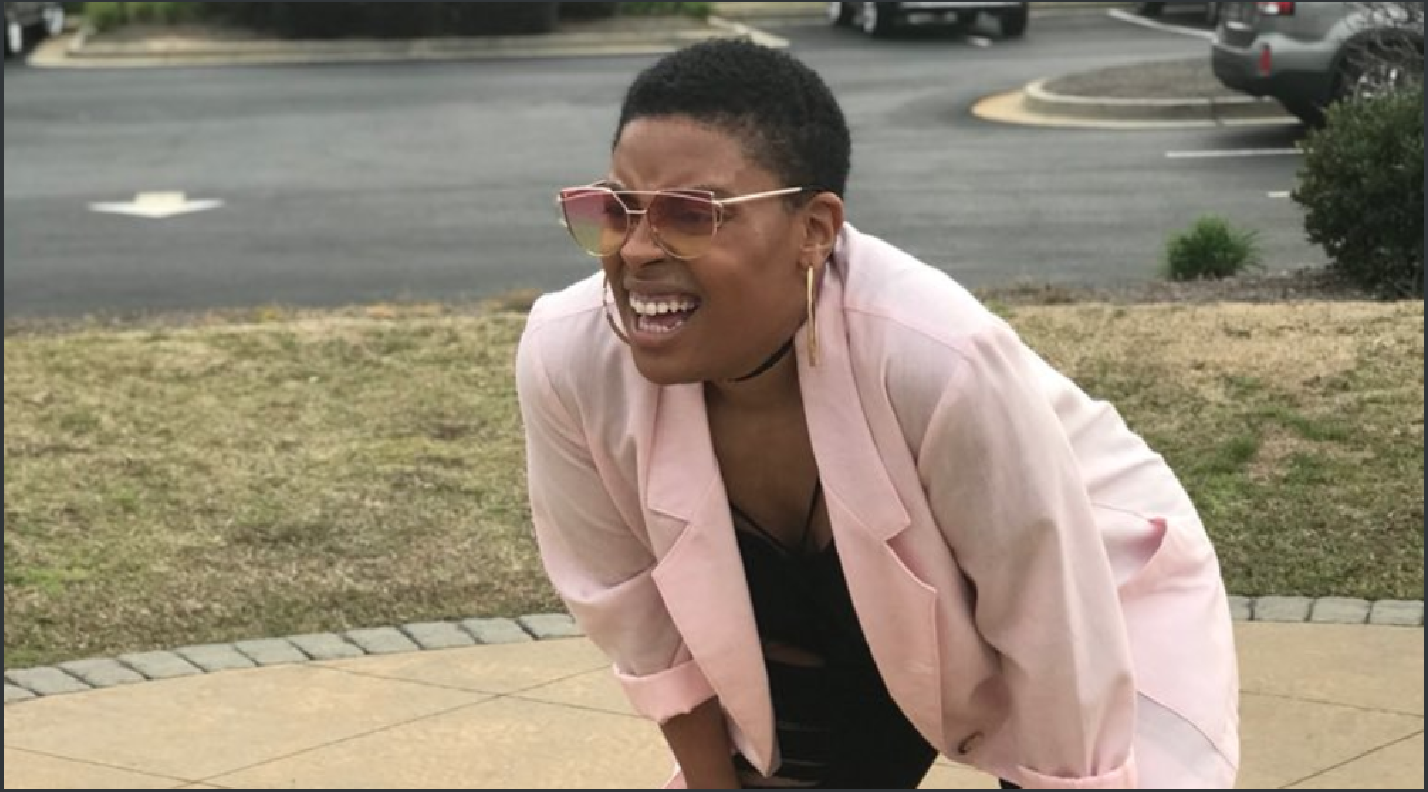
Network Access

(internet, request / responses)

What if there was a hosting environment that allowed JS to have disk and network access

NODE.js BABY

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.



The same shit that lets you run JS in the browser can now be used to run JS on Servers, Desktops, and elsewhere

True Story

V8 Engine Does All The Heavy Lifting

And just like the browser's Web APIs Node come with a bunch of stuff

Built in Modules

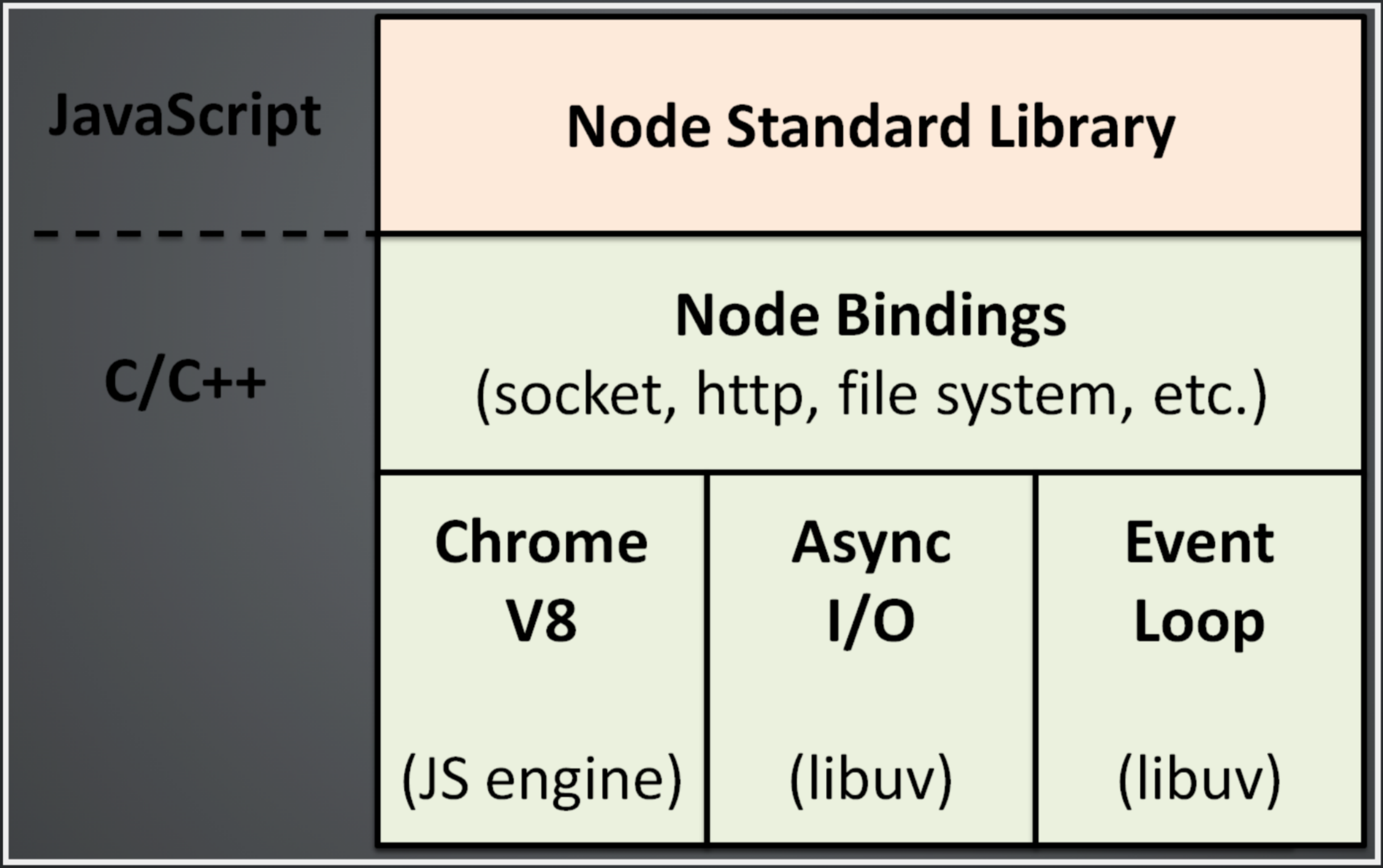
(libraries or collections of functions)

HTTP (network access)

FS (file system access)

Access to millions of packages via NPM

(groupings of one or more custom modules)



sorry, don't remember the source

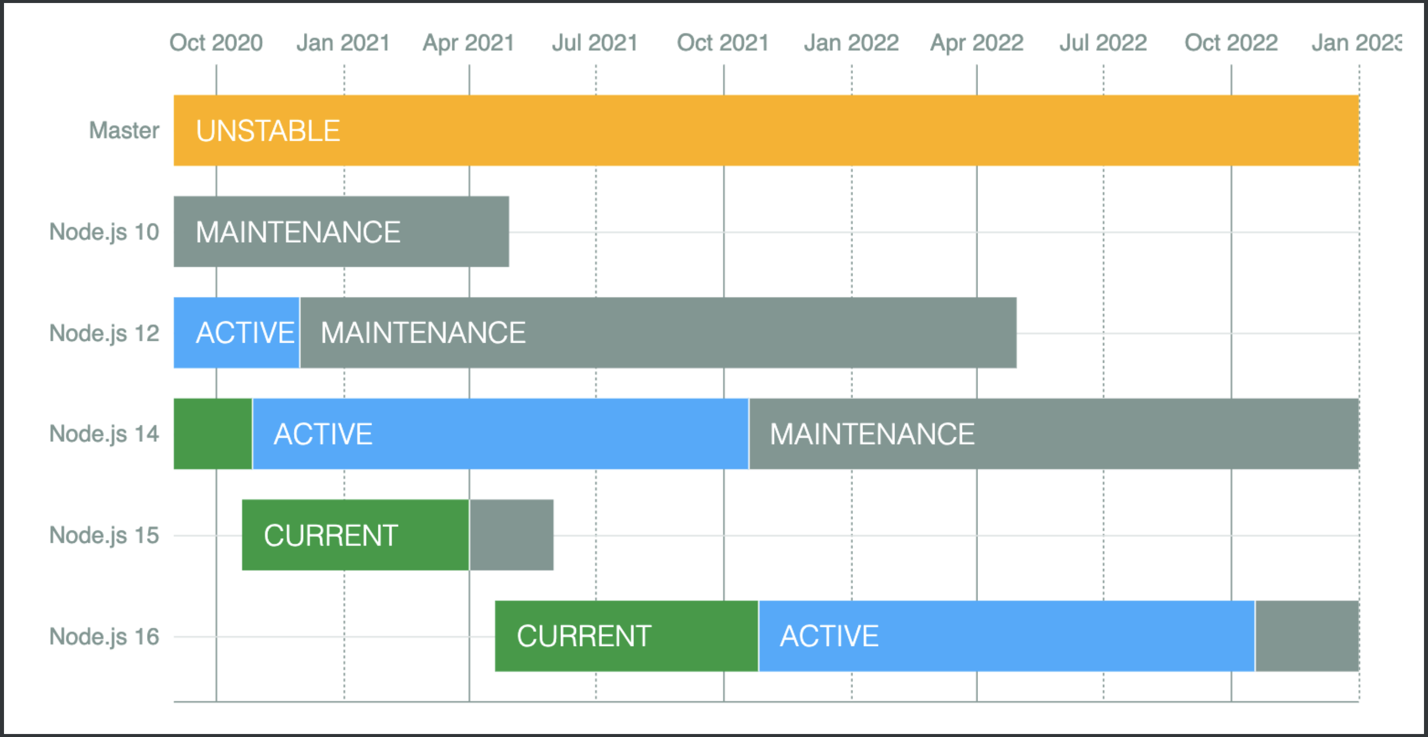
Call stack, Call Back Queue, Node Modules, Event Loop

Thursday

Install Node

Releases?

LTS, Current, Nightly?



Let's Code

Simple Node Server

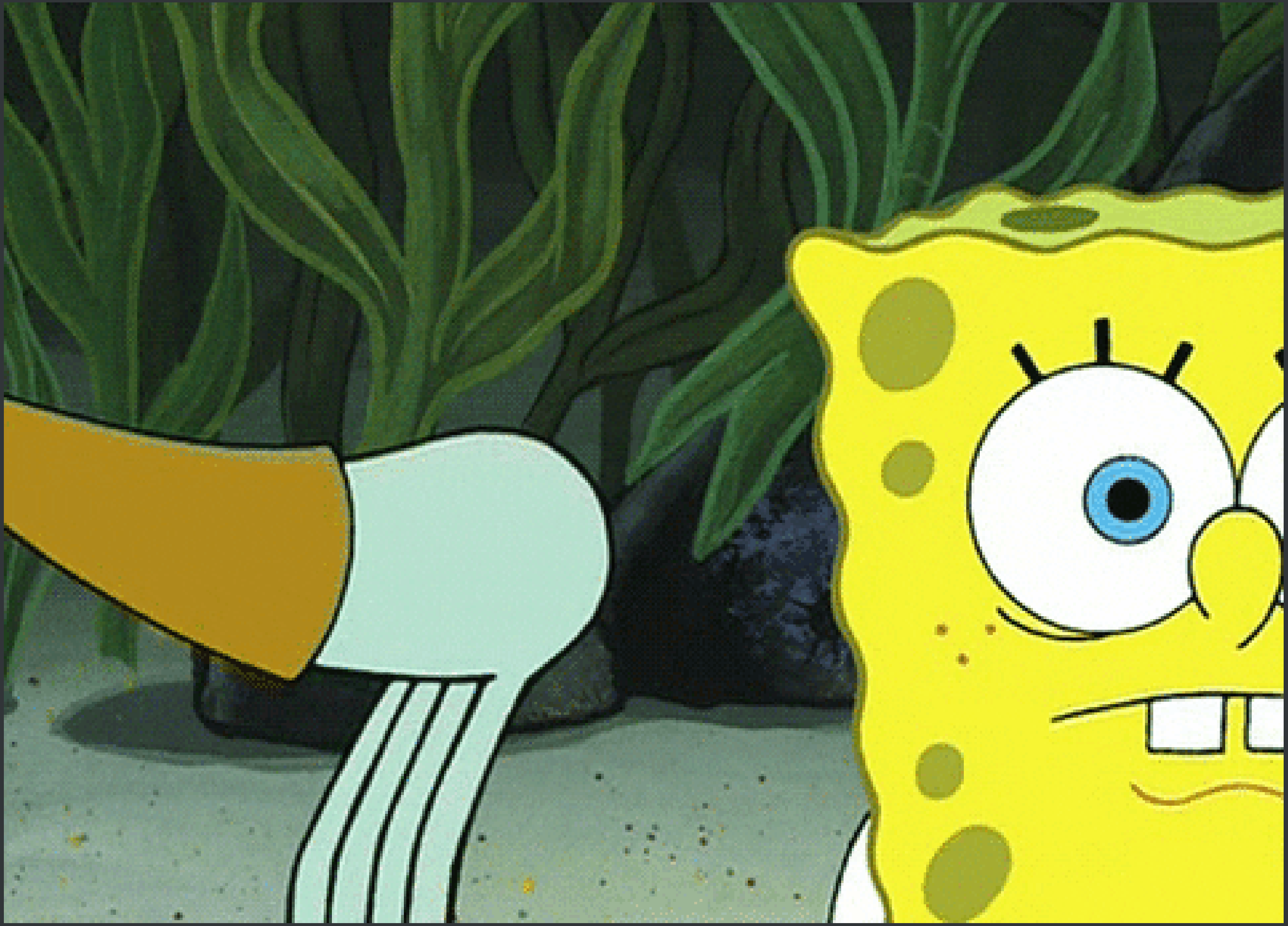
Just
HTTP & FS

```
const http = require('http')
const fs = require('fs')
http.createServer((req, res) => {
  fs.readFile('demofile.html', (err, data) => {
    res.writeHead(200, {'Content-Type': 'text/html'})
    res.write(data)
    res.end()
  })
}).listen(8000)
```

You are now a Software Engineer that can build Fullstack Web Applications

Let's Look

More Complex Backend



How could we clean this up?

Homework

Do: Start prepping THE BANK

Do: Complete Your Professional Links

Do: Make node-backend-simple-json more readable

Do: Make a coinflip game where the randomization happens server side