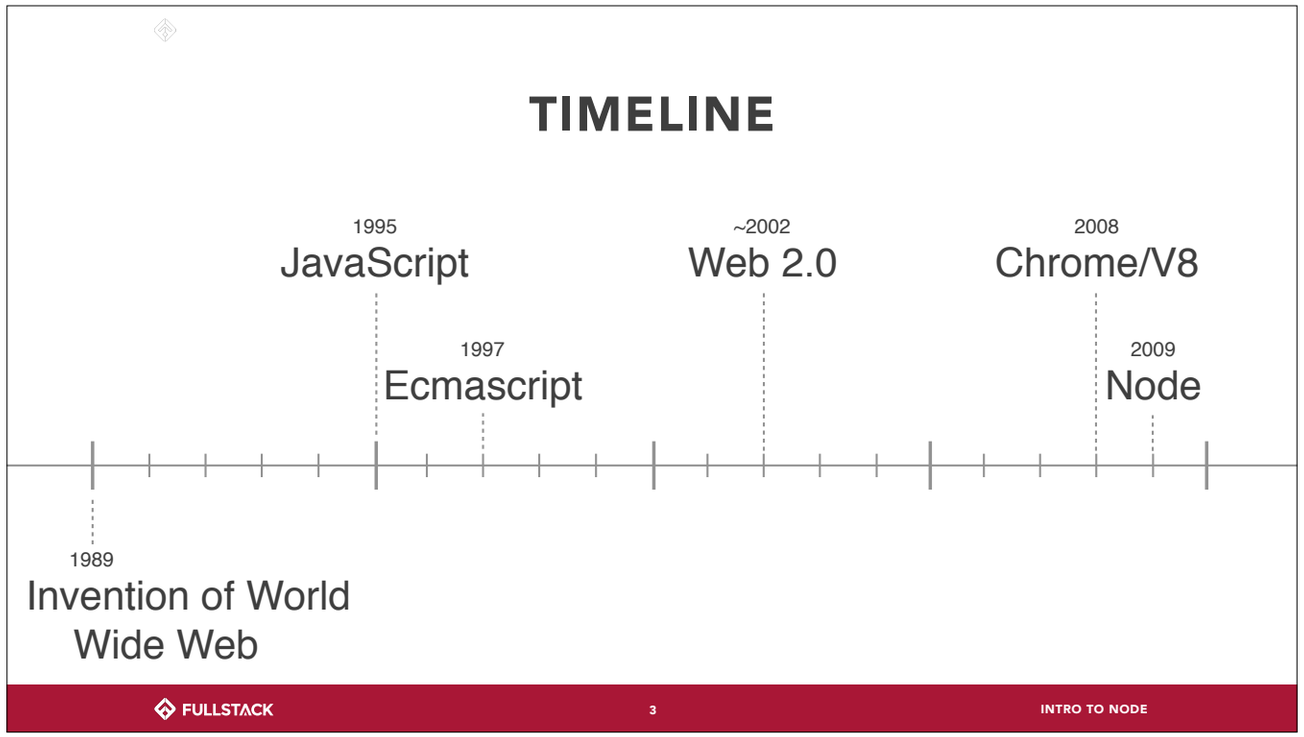
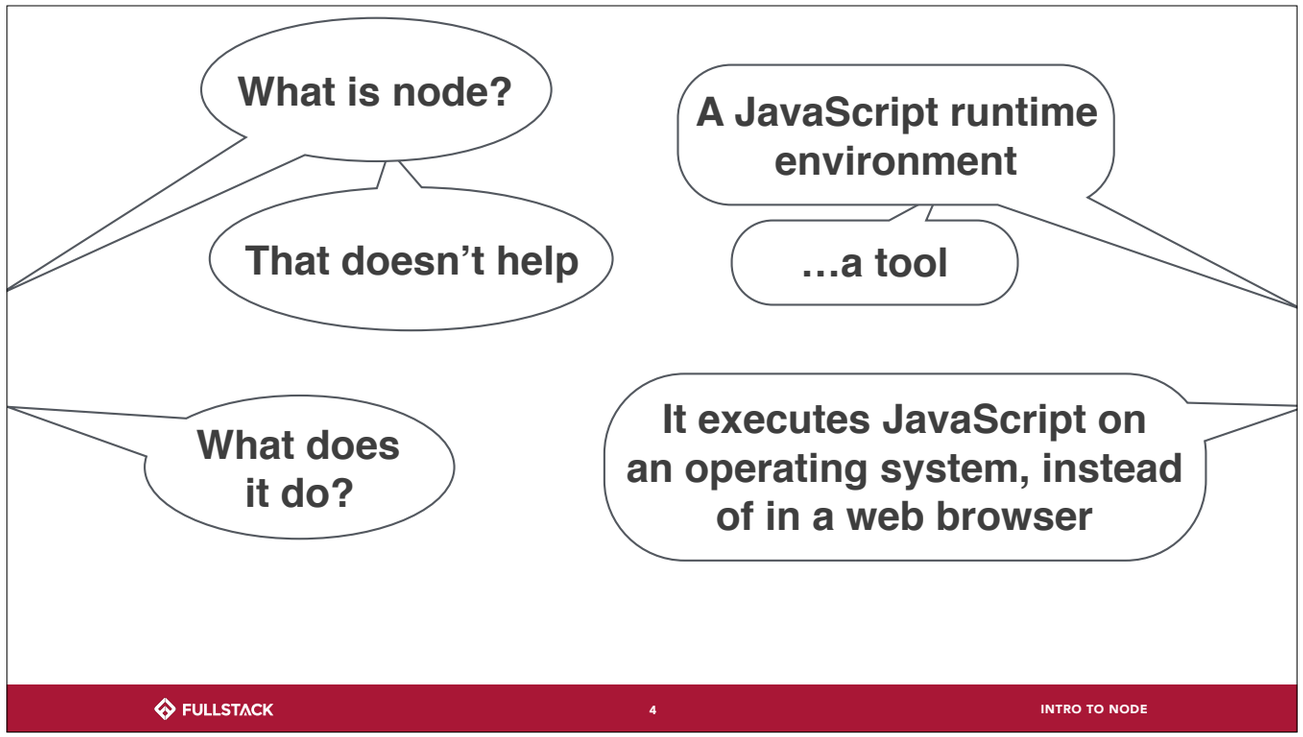
```
NODE.INTRO((err, ideas) => {
  if (err) throw new Question(err)
  else understand(ideas)
})
```


Let’s talk about Node. This slide may not be that funny yet, but it will be very shortly.

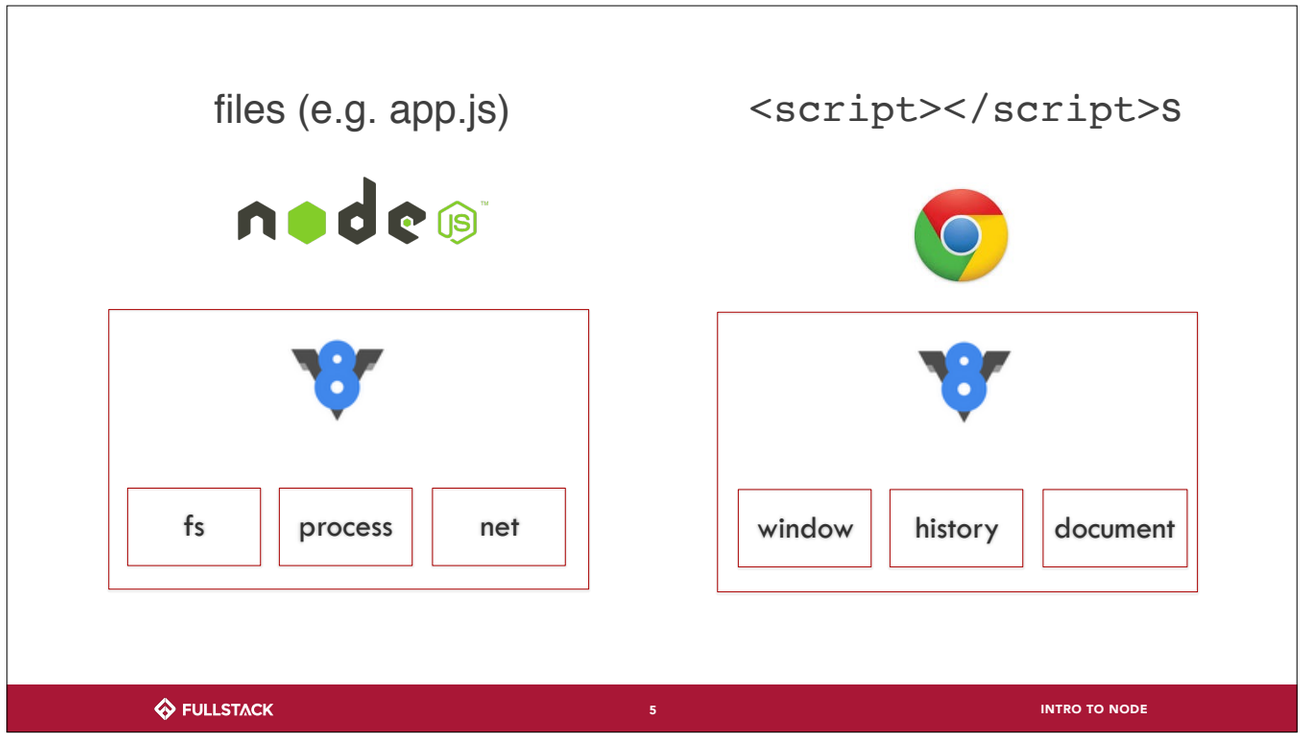
BACKGROUND



JavaScript was first born in 1995 as a scripting language for the Netscape browser, written by Brendan Eich, and by 1997 it was codified in a specification called Ecmascript, so that other browsers could implement JavaScript as well. So for a couple of years, JavaScript was really just used for doing things like adding some interactivity to these otherwise largely static web pages. Then in 2002, browsers begin to support this thing called AJAX - async javascript and xml - which we'll talk about later in this course, which enables web pages to become way more sophisticated than pages. This is where web apps, like Gmail, start to appear. Still, throughout this whole time, JS is a language that needs a browser to run (like Chrome) to run it. In 2009, we got this cool idea - what if we took the V8 "engine" in Chrome - that is, the software in Chrome that executes JavaScript - and worked it around so that it can execute JavaScript on your operating system, the same way python or java might be. Node is the result of that project.



<Do a comparison of chrome dev tools vs. node repl>



Something you may find challenging is telling the difference between Node JavaScript and Browser JavaScript. Here's kind of a side-by-side:
We have Node on the left and a browser (Chrome), on the right. Both use the V8 program to execute JavaScript, but there are some differences in the environment that JavaScript lives in, and in the way they're packaged up. In Browser JavaScript, all of our javascript goes into script tags, which are put in an html document, and then executed when that html page is loaded. We also have some of these hopefully familiar global variables like window, document, history, etc. In Node, JS files are referred to as modules which can be kept in separate js files, and we execute them with Node from our command line. This gives us access to a very different set of global environment variables, with names like fs (for filesystem), process and net.

WHY CARE?

If you want to create a server and know JavaScript

WHY CREATE A SERVER?

If you want to create a custom website or webapp

SERVER

- A program running on a computer connected to the internet
- Serves content requested by remote clients

A server is just a “role” that a program connected to the internet plays

IF PROGRAMMING WERE COOKING...

What does it mean for a program to be running?

Program vs. Process

"recipe"

- **Program is data**
  - machine code (pre-compiled)
  - bytecode (re-compiled by a VM)
  - text file (can be interpreted)
- **Inert — not doing anything**
- **Ready to be run as a process**

- **Process is execution** "cooking"
  - memory allocated
  - CPU performing steps
- **"Live"**
- **Produces results**
- **Interactive**
- **Can be started/stopped**
- **Multiple processes from one program…**

Example: pre-compiled Chrome binary in Applications folder; "launching" that program means starting a Chrome process.

Bytecode - compiled object code that is re-compiled for a specific machine’s processor by a VM

COOKING METAPHOR

	(term)	(metaphor)
`log('hi');`	program	recipe
JavaScript	programming language	recipe language
V8	engine/VM/interpreter	chef
Node	runtime environment	kitchen
Sierra	operating system	building (restaurant?)

MODULES AND THE NODE ENVIRONMENT

Demo:

- modules vs. script tags
  - module.exports and require
- module scoping
  - npm install (use express)
  - node_modules folder
- require with ./ for files, not for node modules

GLOBAL VARIABLES

Every module in Node has access to the same set of global variables

```
process
global
console
setTimeout/clearTimeout
setInterval/clearInterval
```


The same way “window” or “document” are available to any browser js file
global is like “window”, process is like “document”

“MODULE” VARIABLES

- **Every module in Node has its OWN set of “module” variables that are available in the default scope**

```
__dirname
__filename
module
require
```

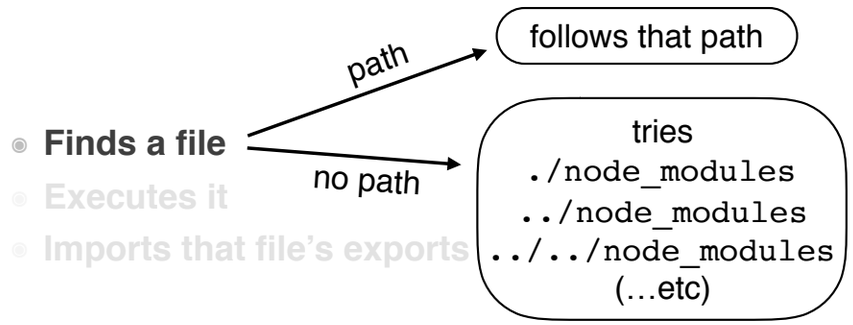
`module`

- **Object**
- **Represents the module itself**
- **Most importantly, has a property called `exports`**

`module.exports`

- **Initially an empty object**
- **Assign it the data you want to expose**
- **A `require` of this file ("module") will return its `module.exports`**

require



- **Finds a file**
- Executes it
- Imports that file's exports

- You can leave off the .js in a file path
- A path to a directory will look for index.js

DEMO

NPM

- node package manager
- Command line tool
- Can find libraries of code online
- Downloads them locally or globally (into `node_modules` directory)
- Keeps list of project dependencies in `package.json`

Go back to demo:

package json

npm install —save

It can be very easy to take something like ‘npm’ for granted, but dependency management is a hard problem that `npm` solves quite well. Having a strong system like `npm` behind it is one of the reasons Node is as popular as it is today - it’s really easy to share projects and collaborate on projects.

package.json

Describes your project, e.g. its dependencies…

- **Collaboration within your team**
- **Sharing within the node community**

SUMMARY

- **Node allows for server-side JavaScript**
- **`require` pulls in what `module.exports` exposes**