node.intro(

```
function(err, ideas) {
    if (err) throw new Question(err)
    understand(ideas)
})
```

Fullstack Academy of Code

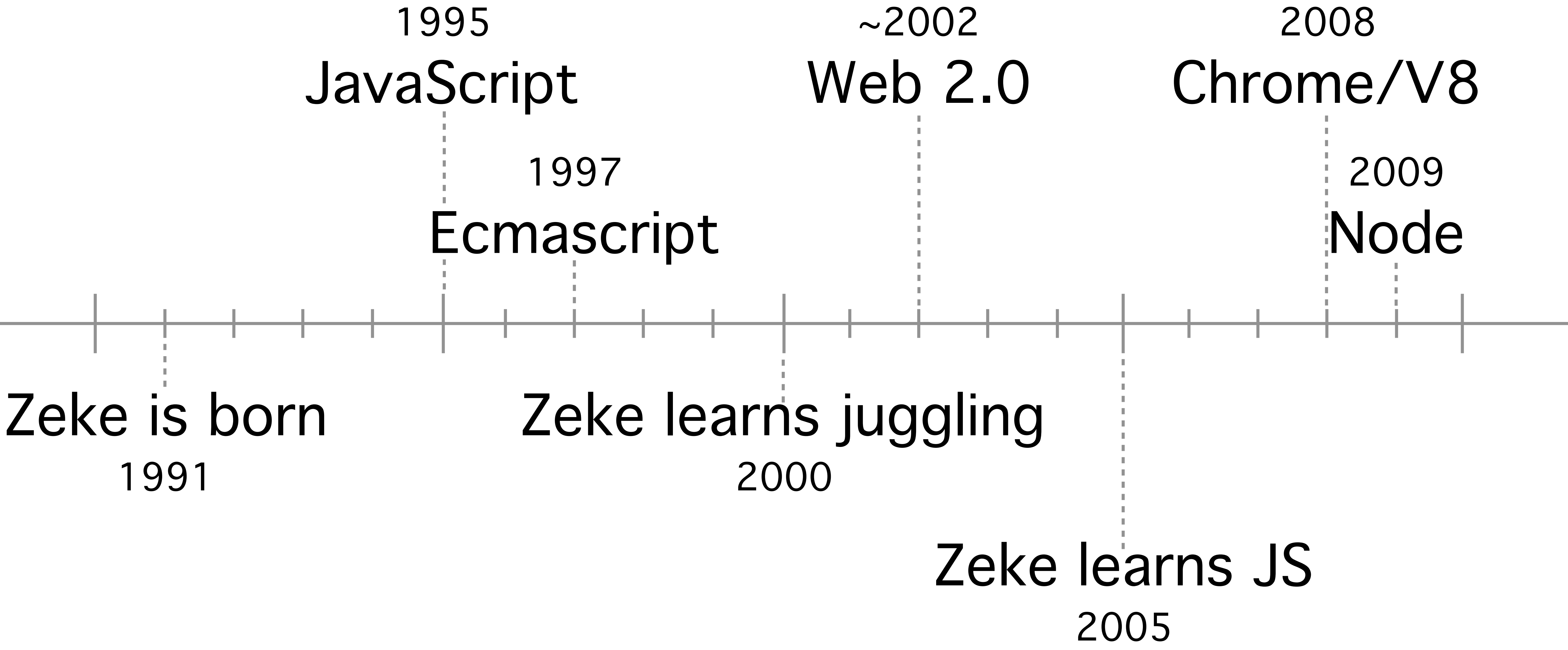
this talk

background (_)

modules (^)

asynchronicity (*)

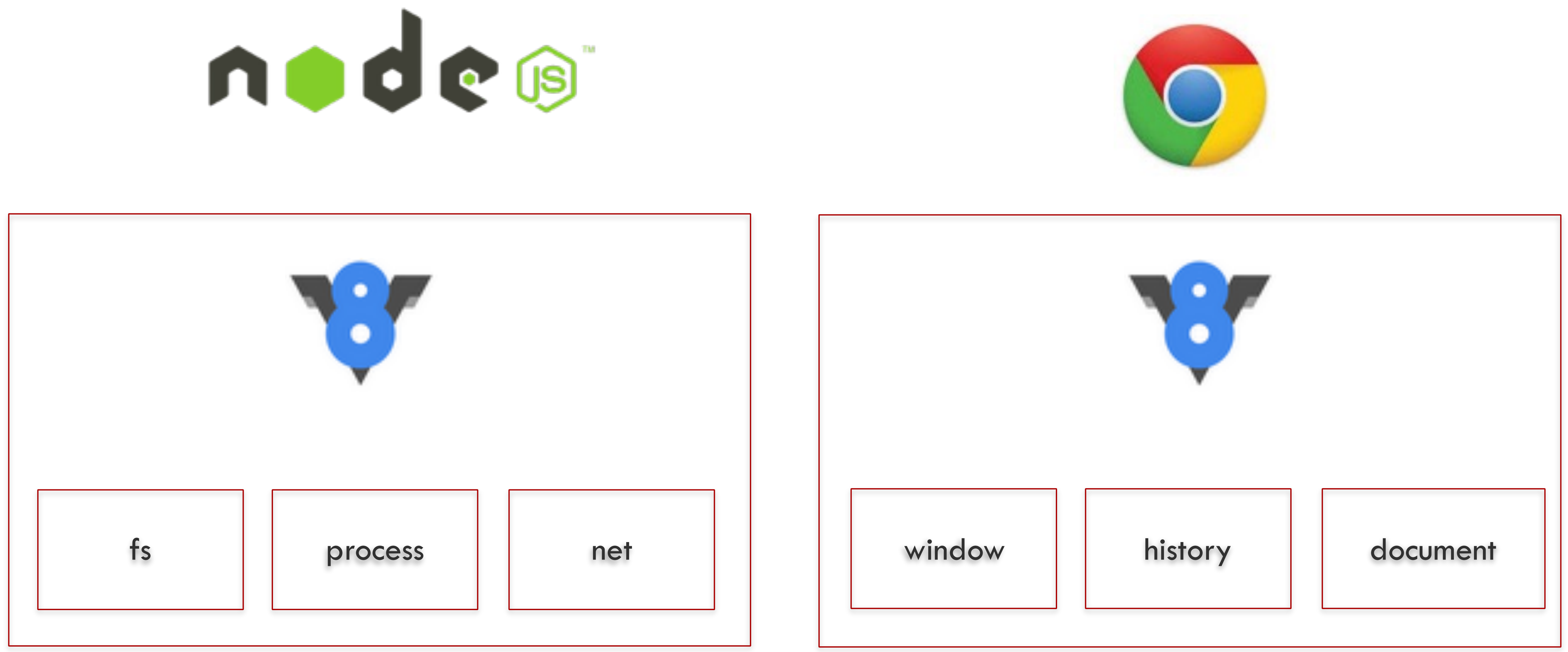
timeline


1995
JavaScript
1997
Ecmascript
~2002
Web 2.0
2008
Chrome/V8
2009
Node
Zeke is born
1991
Zeke learns juggling
2000
Zeke learns JS
2005

node.js

Executes JavaScript on an operating system
...instead of in a web browser

files (e.g. `app.js`) `<script></script>`s



why care?

If you want to create a server and know JavaScript

why create a server?

If you want to create a website or webapp

server

A program running on a computer connected to the internet

Serves content requested by remote clients

cooking metaphor

If programming were cooking...

cooking metaphor

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

modules

`module.exports`

`require`

npm

`module.exports`

Assign it the data you want to expose

A `require` of the file will import its `module.exports`

^

`require`

finds a file

executes it

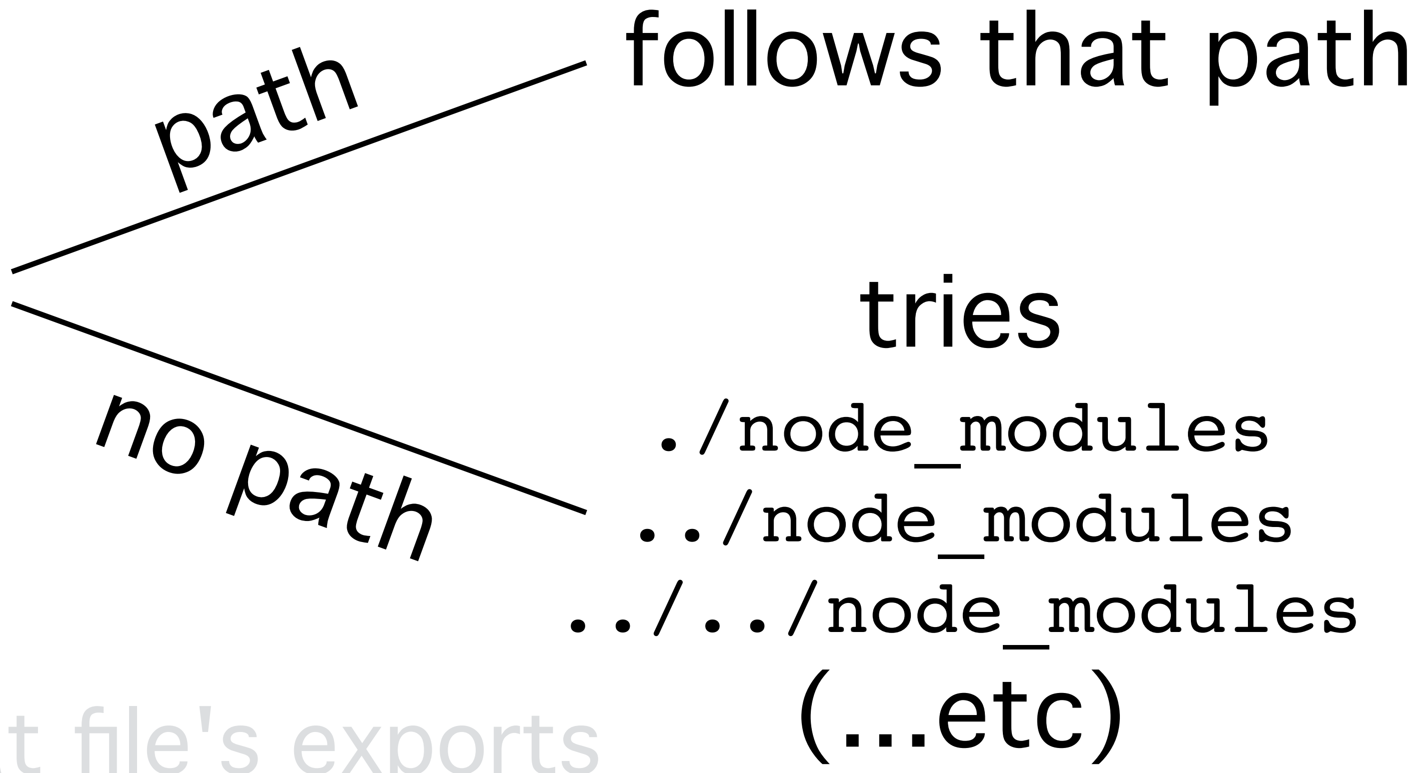
imports that file's exports

require

finds a file

executes it

imports that file's exports



npm

node package manager

command line tool

can find libraries of code online

download them locally

keeps a list of dependencies in `package.json`

package.json

Describes your project, e.g. its dependencies

collaboration

...within your team

sharing

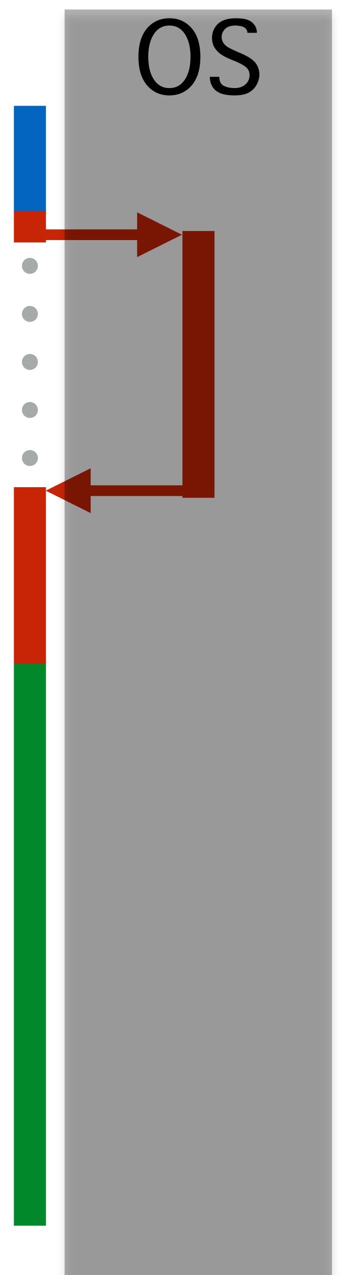
...within node community

*

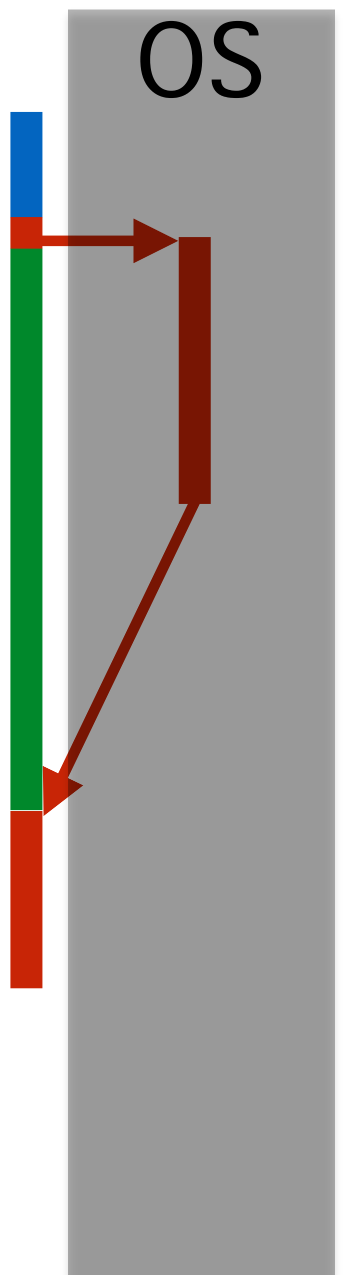
(skit)

concurrency

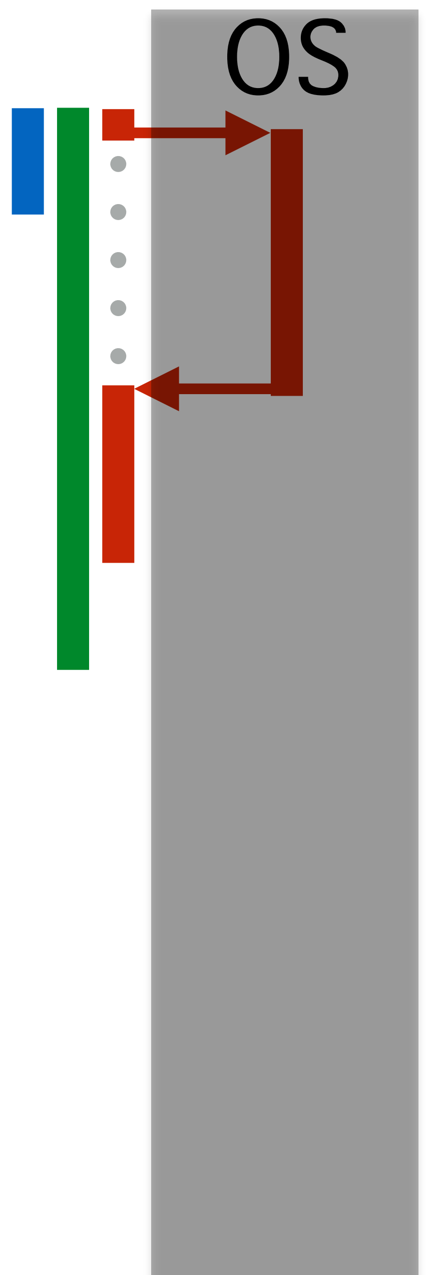
blocking



non-blocking



parallel (blocking)



*

concurrency

JavaScript is single threaded, its runtime environment is not

The runtime environment exposes tools that you can trigger

*

async

(Code is asynchronous if) the execution order is independent of the command order

event based

If it helps, think of an asynchronous function as something that...

1. kicks off some external process
2. registers an event handler for that process finishing (callback)

*

what happens?

```
var start = new Date;
setTimeout(function(){
  var end = new Date;
  console.log('Time elapsed:', end - start, 'ms');
}, 500);

while (new Date - start < 1000) {};
```

```
=> Time elapsed: 1000 ms
```

why?

```
var start = new Date;
setTimeout(function(){ //starts up a timeout
  var end = new Date;
  console.log('Time elapsed:', end - start, 'ms');
}, 500);

while (new Date - start < 1000) {}; //idles for 1000 ms
//...meanwhile, the timeout finishes
//however, the queued callback only executes once the
//callstack is empty
```

*

gotcha

You can't know just by looking at a function's name whether or not it executes asynchronously, you have to look at the documentation

summary

Node allows for server-side JavaScript

`require` pulls in what `module.exports` puts out

JavaScript is single-threaded but its friends are not

A callback executes when its async event finishes