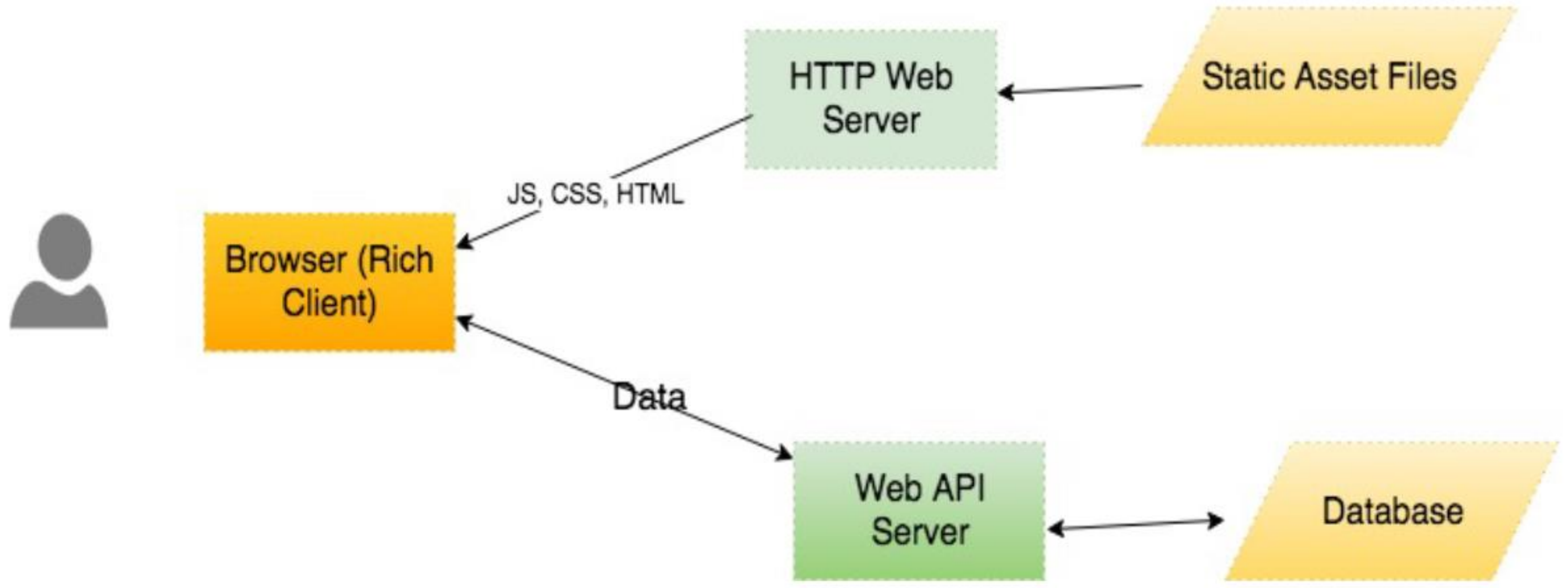
Introduction to Node.js

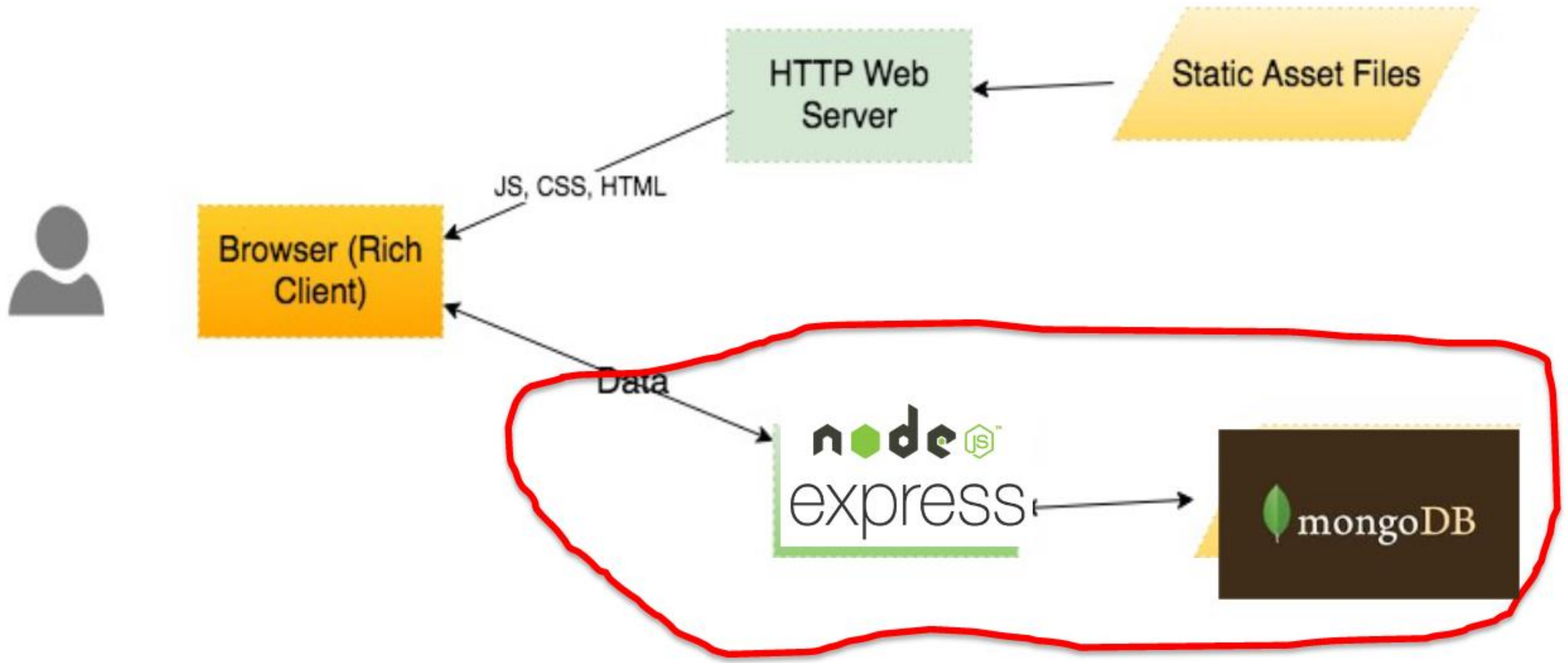
Frank Walsh
Diarmuid O'Connor

Context

Modern Web Apps - Architecture



Modern Web Apps

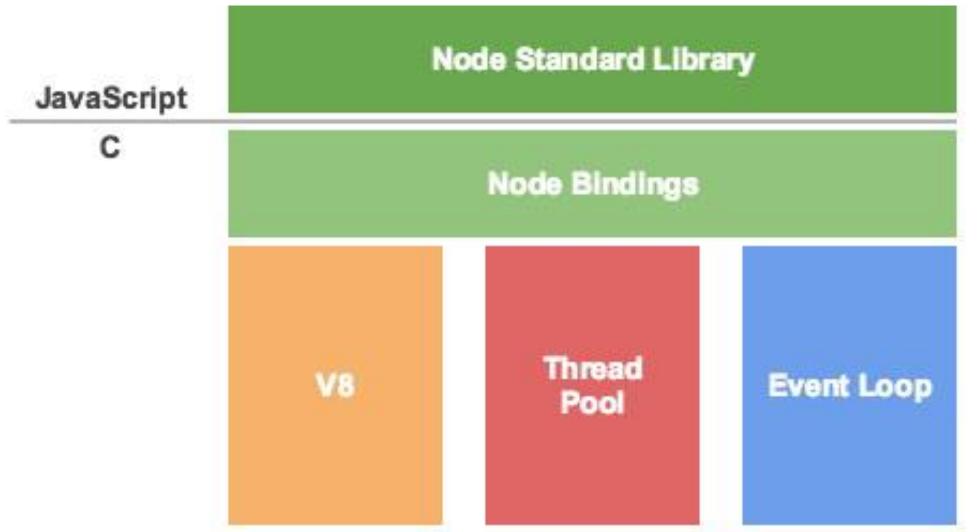


Agenda

- What is node.js
- Non Blocking and Blocking
- Event-based processes
- Callbacks in node
- Node Package Manager(NPM)
- Creating a node app

What's Node: Basics

- A Javascript runtime. “Server side JS”
- The “.js” doesn’t mean that it’s written completely in JavaScript.
  - approx. 40% JS and 60% C++
- Ecosystem of packages (NPM)
- Official site: “Node's goal is to provide an easy way to build scalable network programs”.
- Single Threaded, Event based
  - Supports concurrency using events and callbacks…

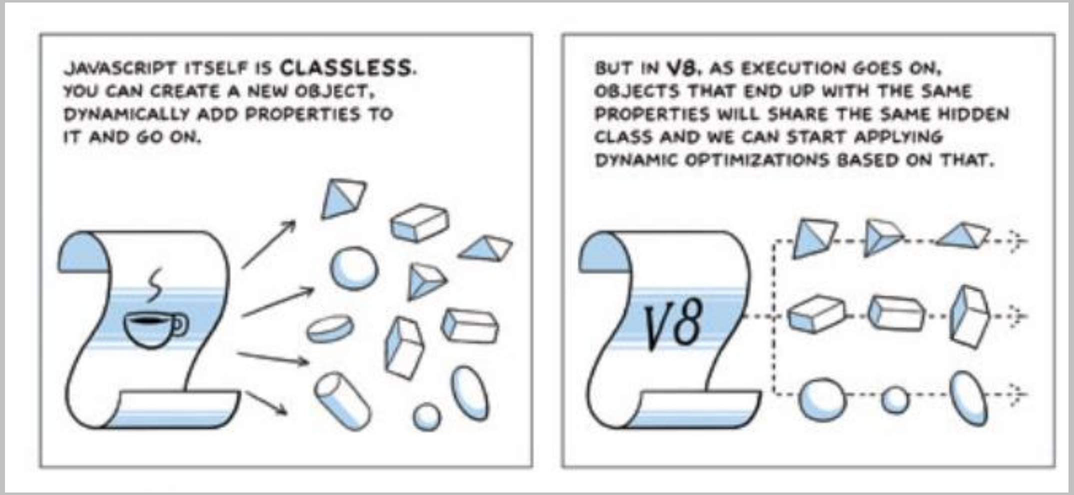


What’s Node: V8.

- Embedded C++ component
- Javascript virtual machine.
- Very fast and platform independent
- Find out a bit about it's history here:

http://www.google.com/googlebooks/chrome/big_12.html

V8 JavaScript Engine


JAVASCRIPT ITSELF IS CLASSLESS. YOU CAN CREATE A NEW OBJECT, DYNAMICALLY ADD PROPERTIES TO IT AND GO ON.
BUT IN V8, AS EXECUTION GOES ON, OBJECTS THAT END UP WITH THE SAME PROPERTIES WILL SHARE THE SAME HIDDEN CLASS AND WE CAN START APPLYING DYNAMIC OPTIMIZATIONS BASED ON THAT.
V8

What is Node.js: Event-based

- Input/Output (io) is slow.
  - Reading/writing to data store, network access.
  - Read 4K randomly from SSD* 150,000 ns ~1GB/sec SSD
  - Round trip over network within same datacenter 500,000 ns
  - Send packet US->Netherlands->US 150,000,000 ns

- CPU operations are fast.
  - L1 cache reference 0.5 ns
  - L2 cache reference 7 ns

- ***I/O operations detrimental to highly concurrent apps (e.g. web applications)***

- Solutions to deal with this are:
  - **Blocking code** combined with multiple threads of execution (e.g. Apache, IIS)
  - **Non-blocking**, **event-based code** in single thread (e.g. NGINX, Node.js)

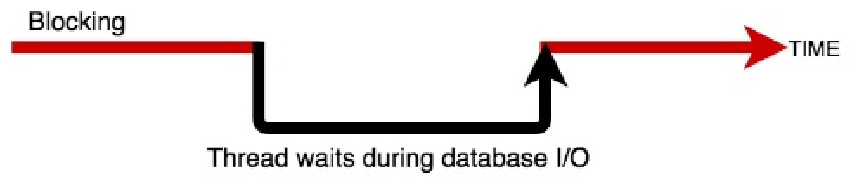
Source: https://gist.github.com/jboner/2841832

1ns = 10^{-9} s (0.000000001s)

Blocking/Non-blocking Example

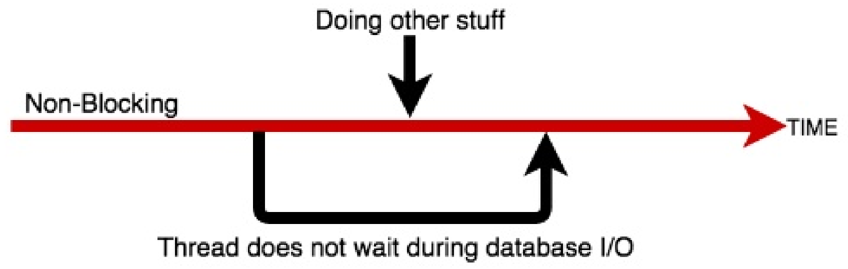
Blocking

1. Read from file and set equal to contents
2. Print Contents
3. Do other stuff...



Non-blocking

1) Read from File
   Whenever read is complete, print contents
2) Do other stuff...



Blocking/Non-blocking: JS

Blocking

```
import fs from 'fs';

const contents = fs.readFileSync('./readme.md’, ‘utf8’);
console.log(contents);
console.log('Doing something else');
```

Console output

```
Hello World……
Doing something else
```

callback

Non-blocking

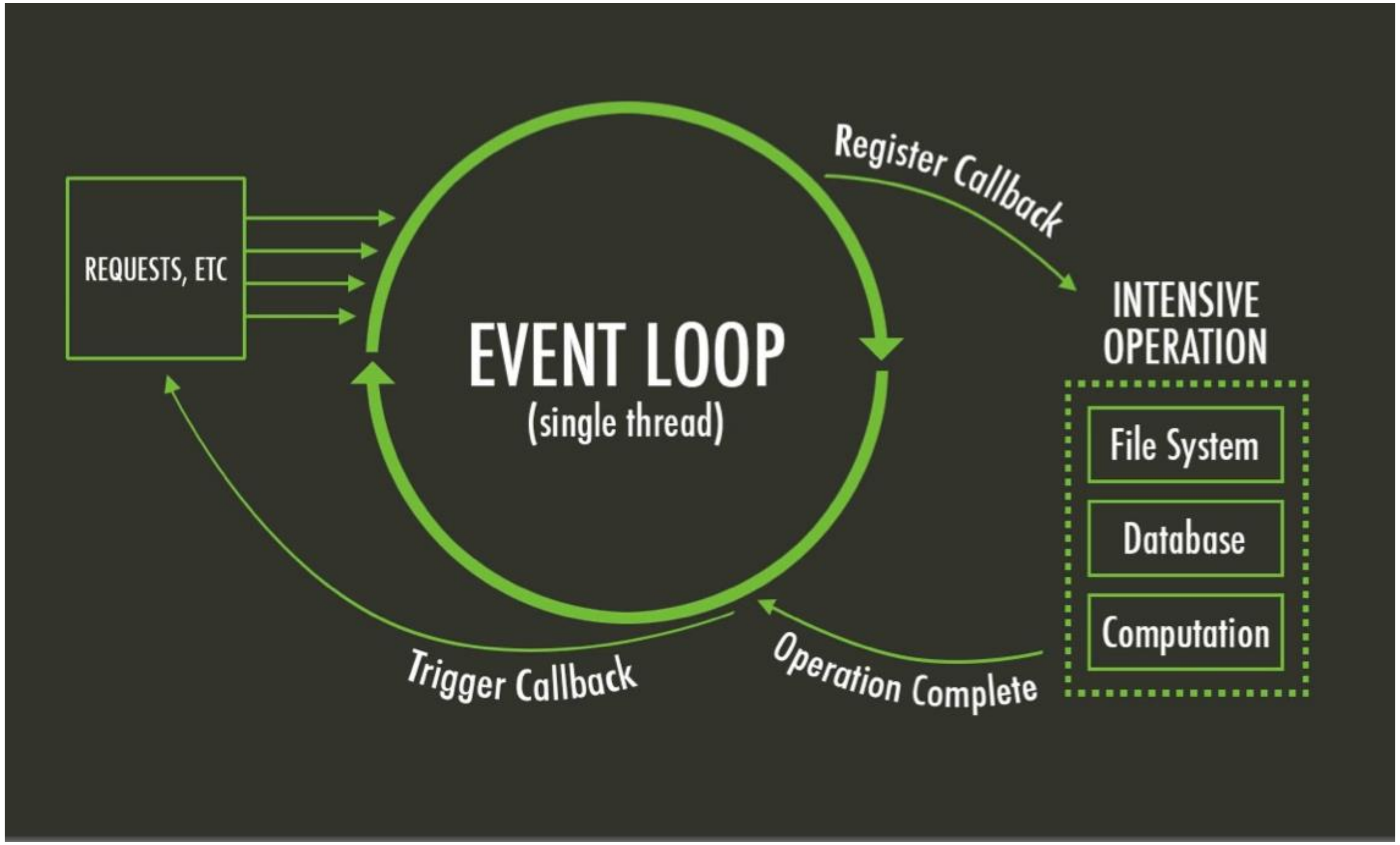
```
import fs from 'fs’;
fs.readFile('./text.txt’,’uft8’, (err, contents) => {
    console.log(contents);
});
console.log('Doing something else');
```

Console output

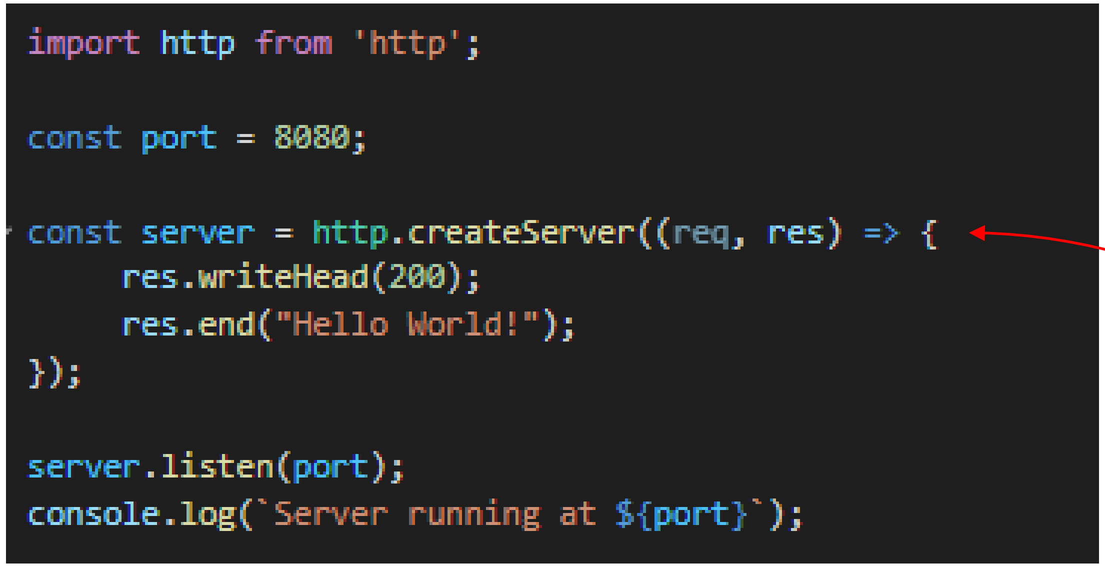
```
Doing something else
Hello World ……
```

The Node Event Loop and Callbacks

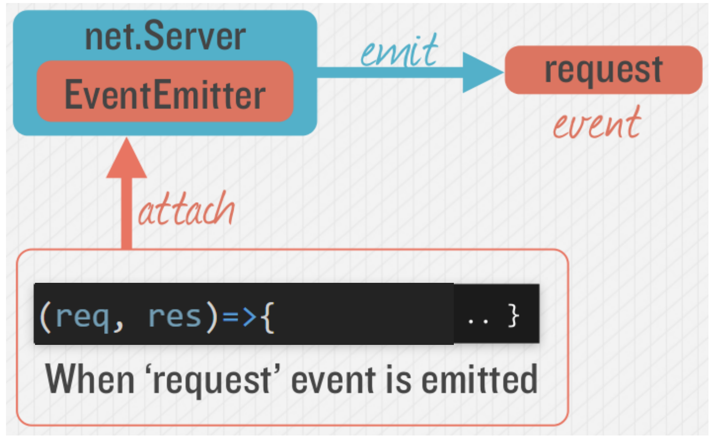
- A **Callback** is a function called at the completion of a given task. This prevents any blocking, and allows other code to be run in the meantime
- The Event Loop checks for known events, registers Callbacks and, triggers callback on completion of operation

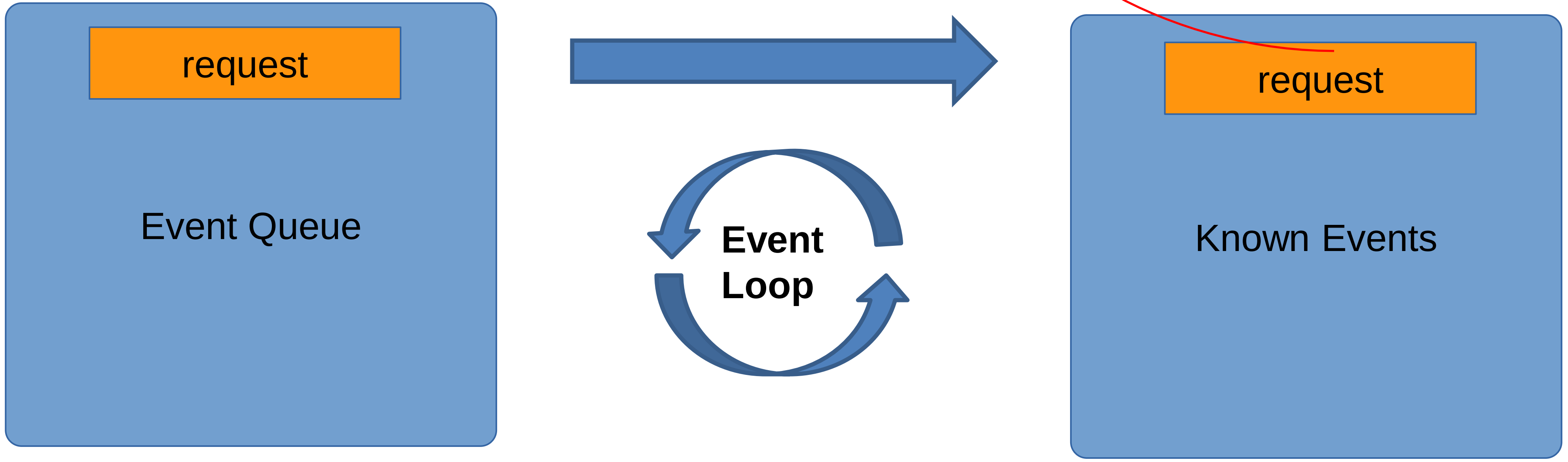


Node.js - Simple HTTP Server



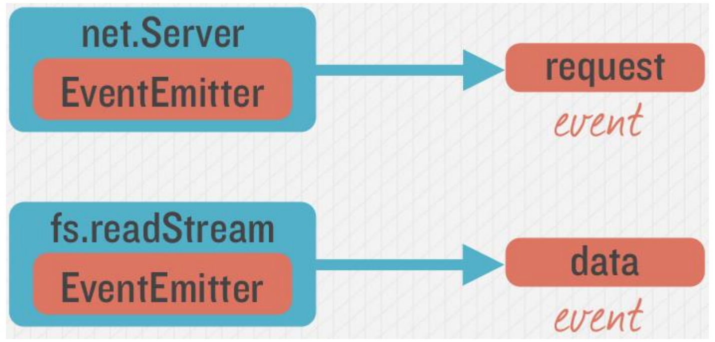
```
import http from 'http';

const port = 8080;

const server = http.createServer((req, res) => {
    res.writeHead(200);
    res.end("Hello World!");
});

server.listen(port);
console.log(`Server running at ${port}`);
```



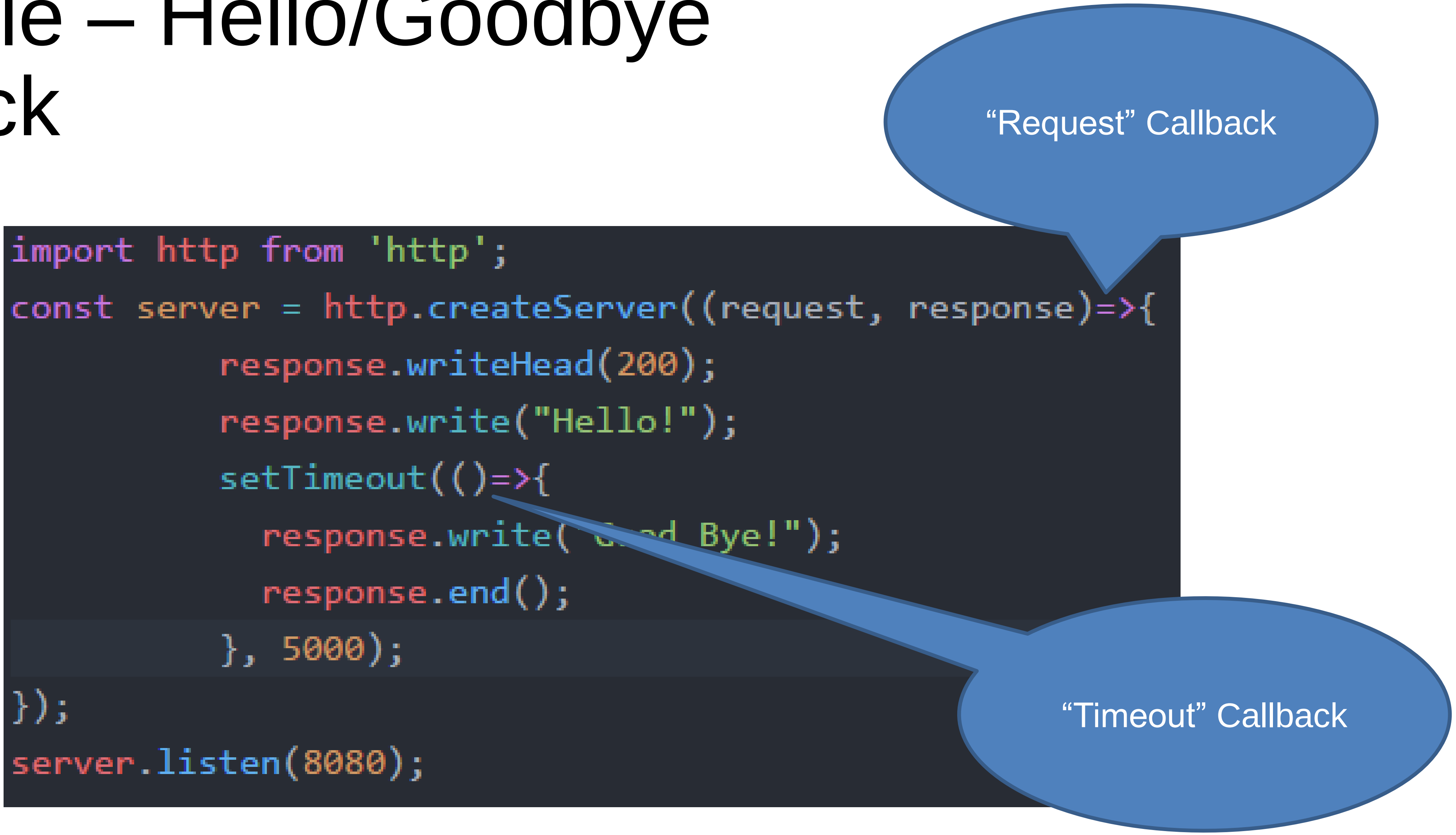


Emitting Event in Node

Many objects can emit events in node.



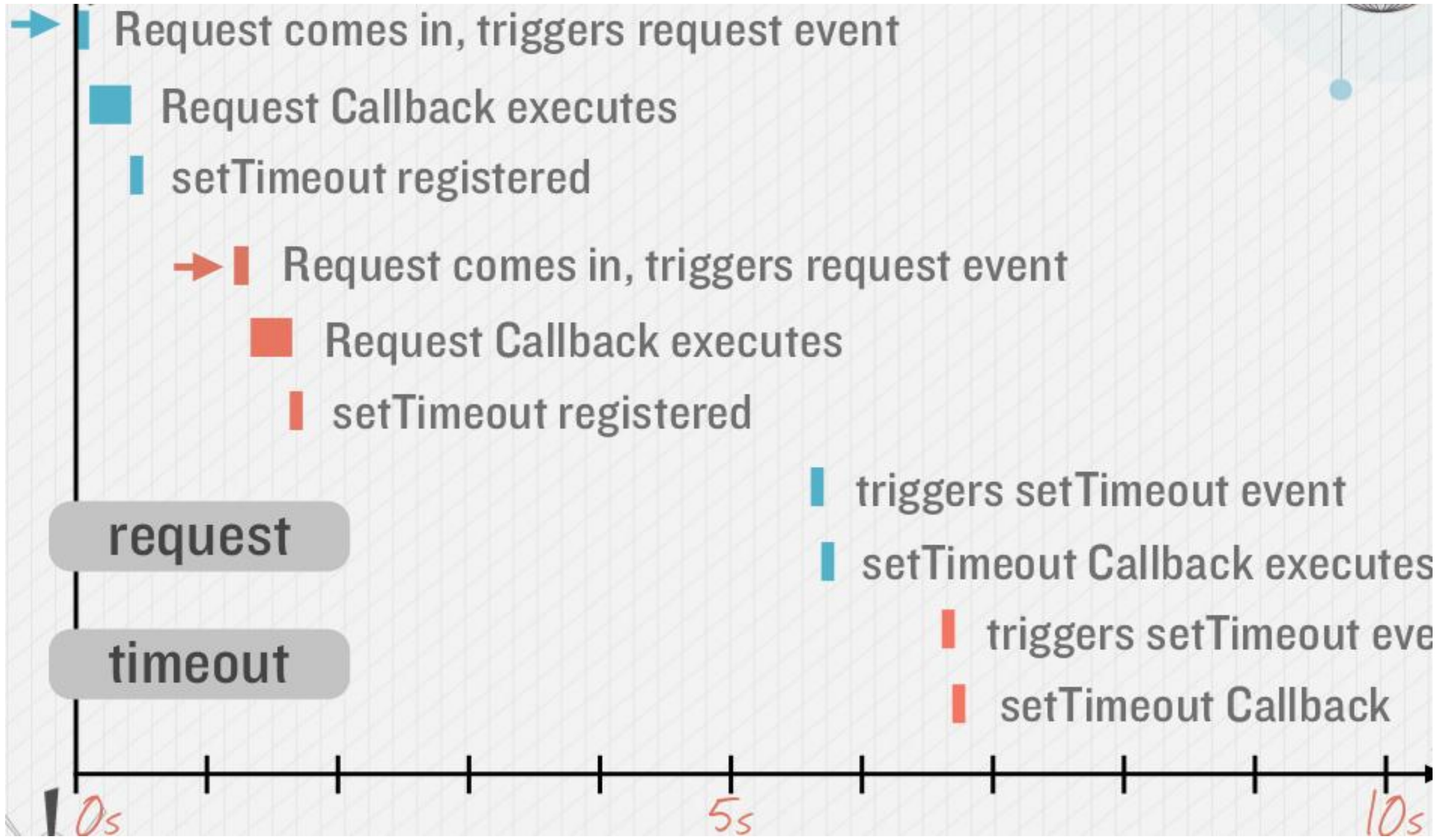
Example – Hello/Goodbye Callback



```
import http from 'http';
const server = http.createServer((request, response)=>{
          response.writeHead(200);
          response.write("Hello!");
          setTimeout(()=>{
            response.write("Good Bye!");
            response.end();
          }, 5000);
});
server.listen(8080);
```

Callback Timeline, Non Blocking

Timing example: 2 requests to web application (indicated by red and blue in diagram)


Request comes in, triggers request event
Request Callback executes
setTimeout registered
Request comes in, triggers request event
Request Callback executes
setTimeout registered
triggers setTimeout event
setTimeout Callback executes
triggers setTimeout eve
setTimeout Callback
request
timeout
0s
5s
10s

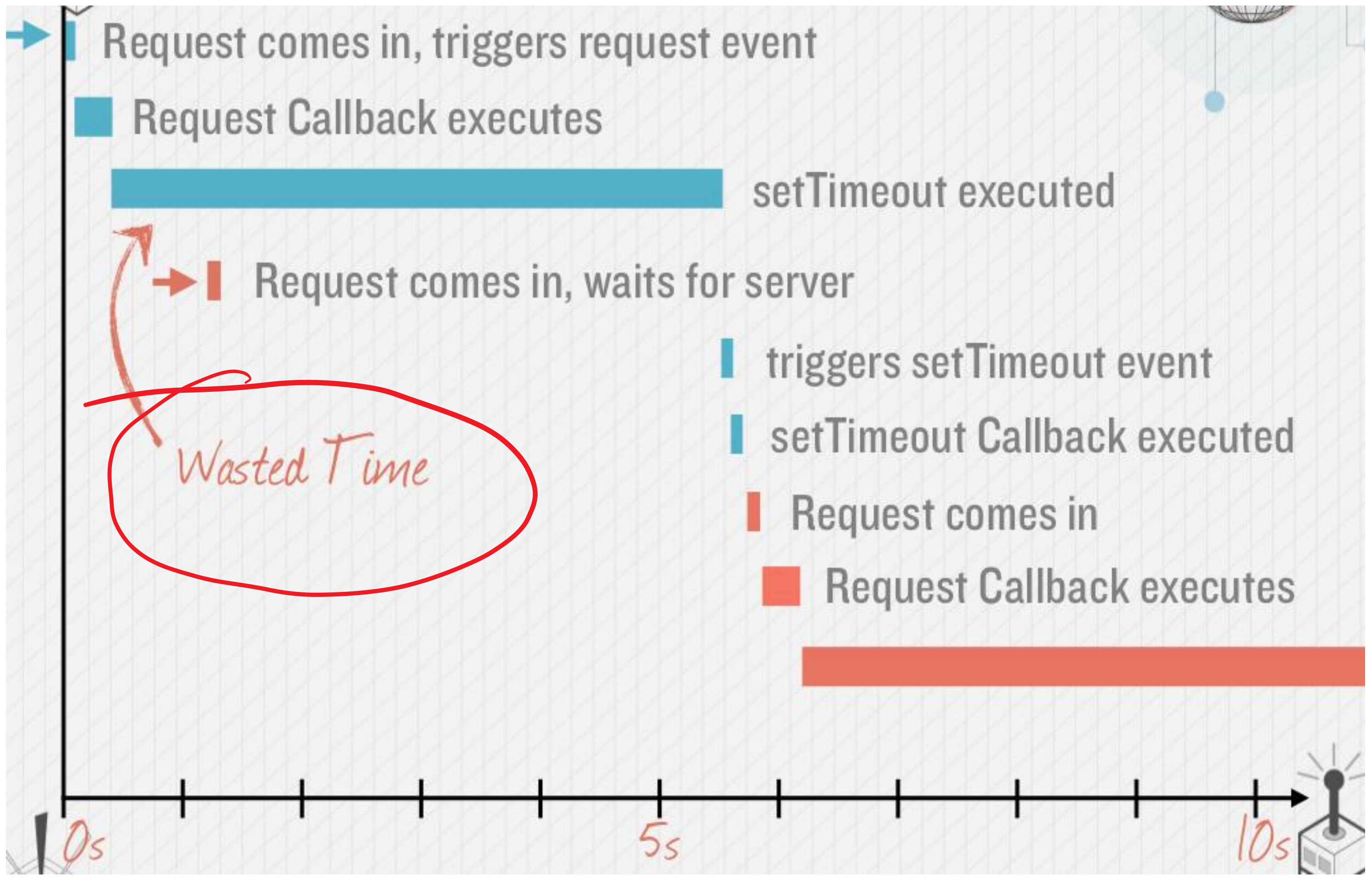
Avoid Blocking Calls in Node.js apps

- setTimeout in previous slide is an example of an asynchronous, non-blocking call.
- Avoid potential blocking/ synchronous calls
- **Activity likely to be blocking should be called asynchronously.**

Examples:

- Calls to 3rd party Web Services
- Database queries
- Computationally expensive operations (image file processing)

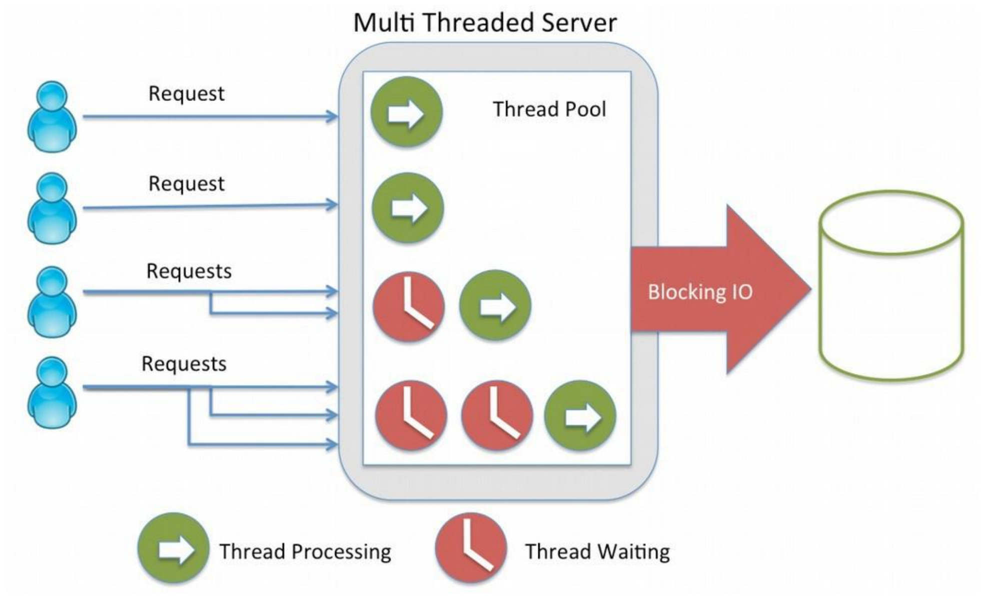
What if setTimeout() blocked…


Request comes in, triggers request event
Request Callback executes
setTimeout executed
Request comes in, waits for server
triggers setTimeout event
setTimeout Callback executed
Request comes in
Request Callback executes
Wasted Time
0s
5s
10s

Blocking vs. Non-blocking: Web Servers

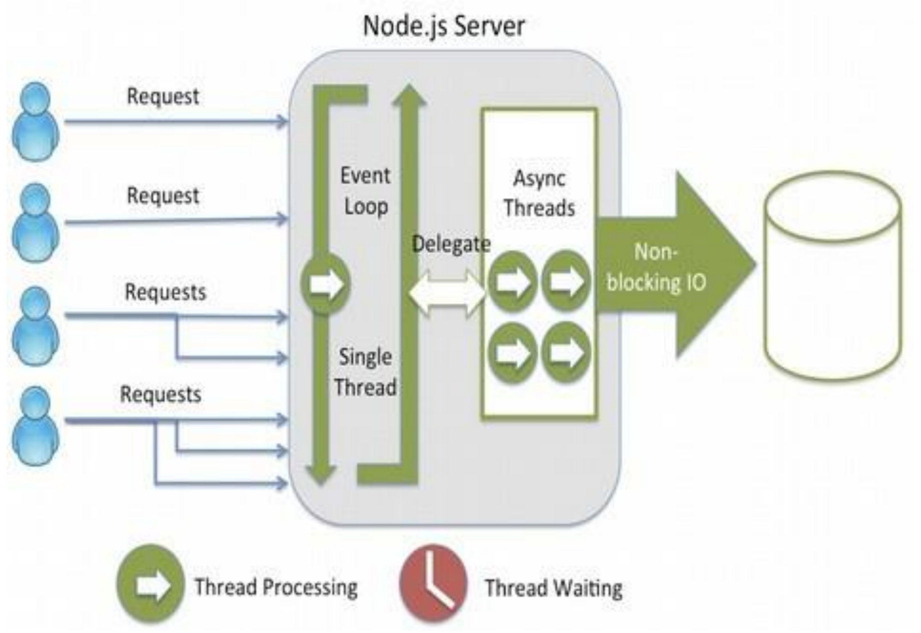
Threads consume resources

- Memory on stack
- Processing time for context switching etc.



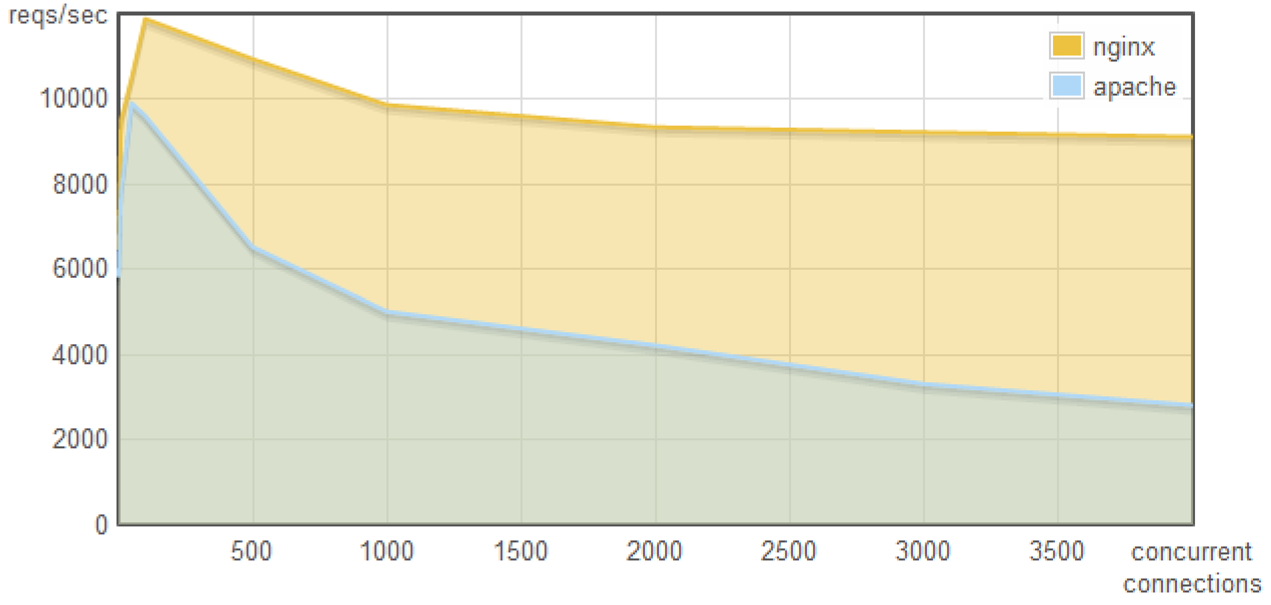
No thread management on single threaded apps

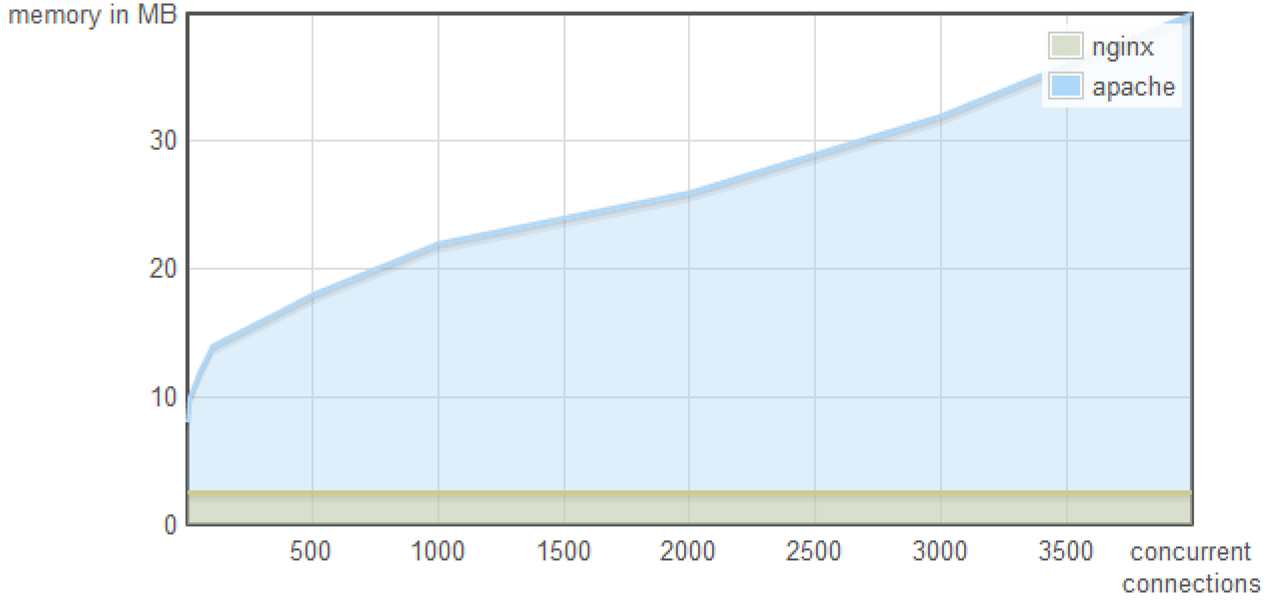
- Just execute “callbacks” when event occurs



Why does it matter...

- This is why:

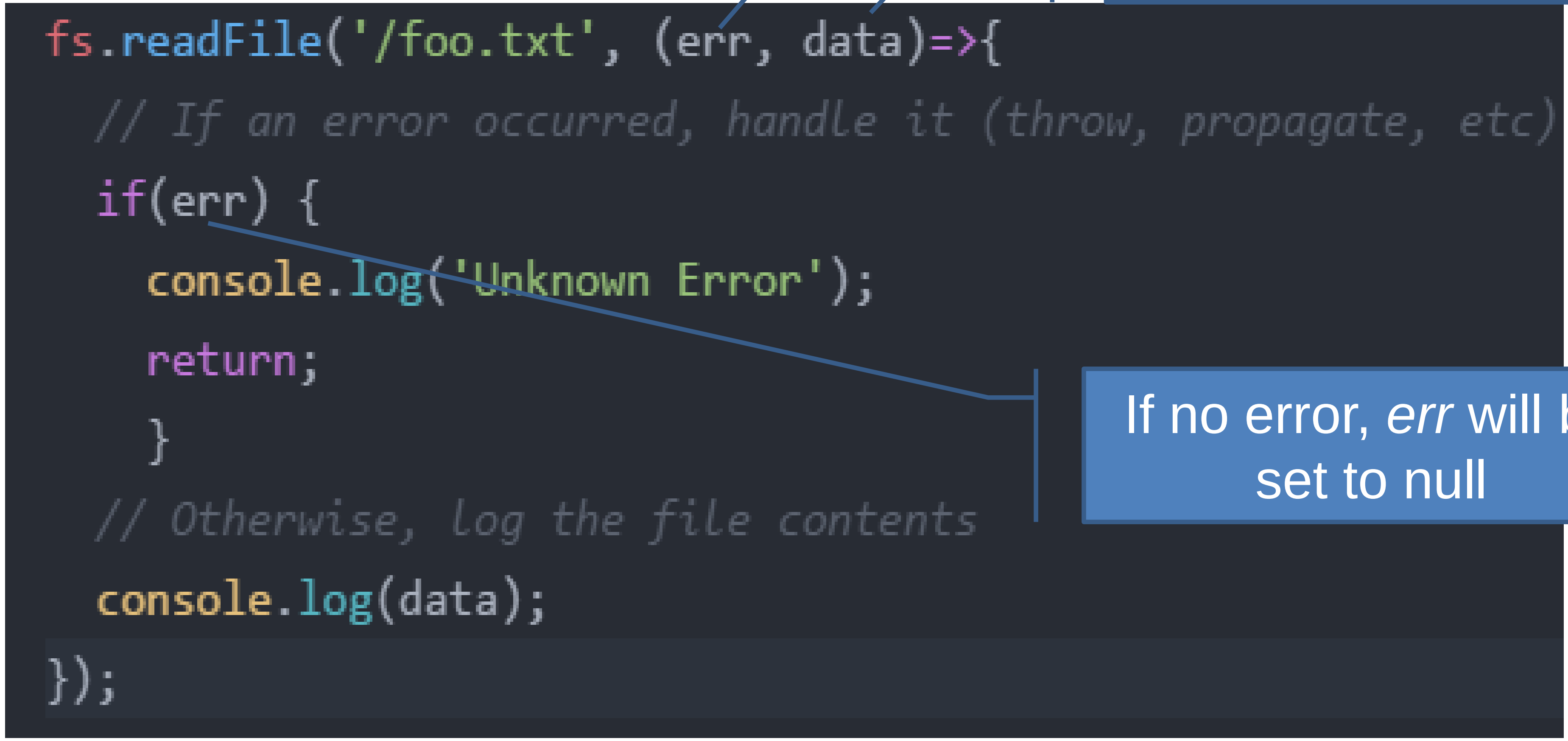




http://blog.webfaction.com/a-little-holiday-present

Node “Error First” Callbacks

The “error-first” callback (or “node-style callback”) is a standard convention for many Node.js callbacks.



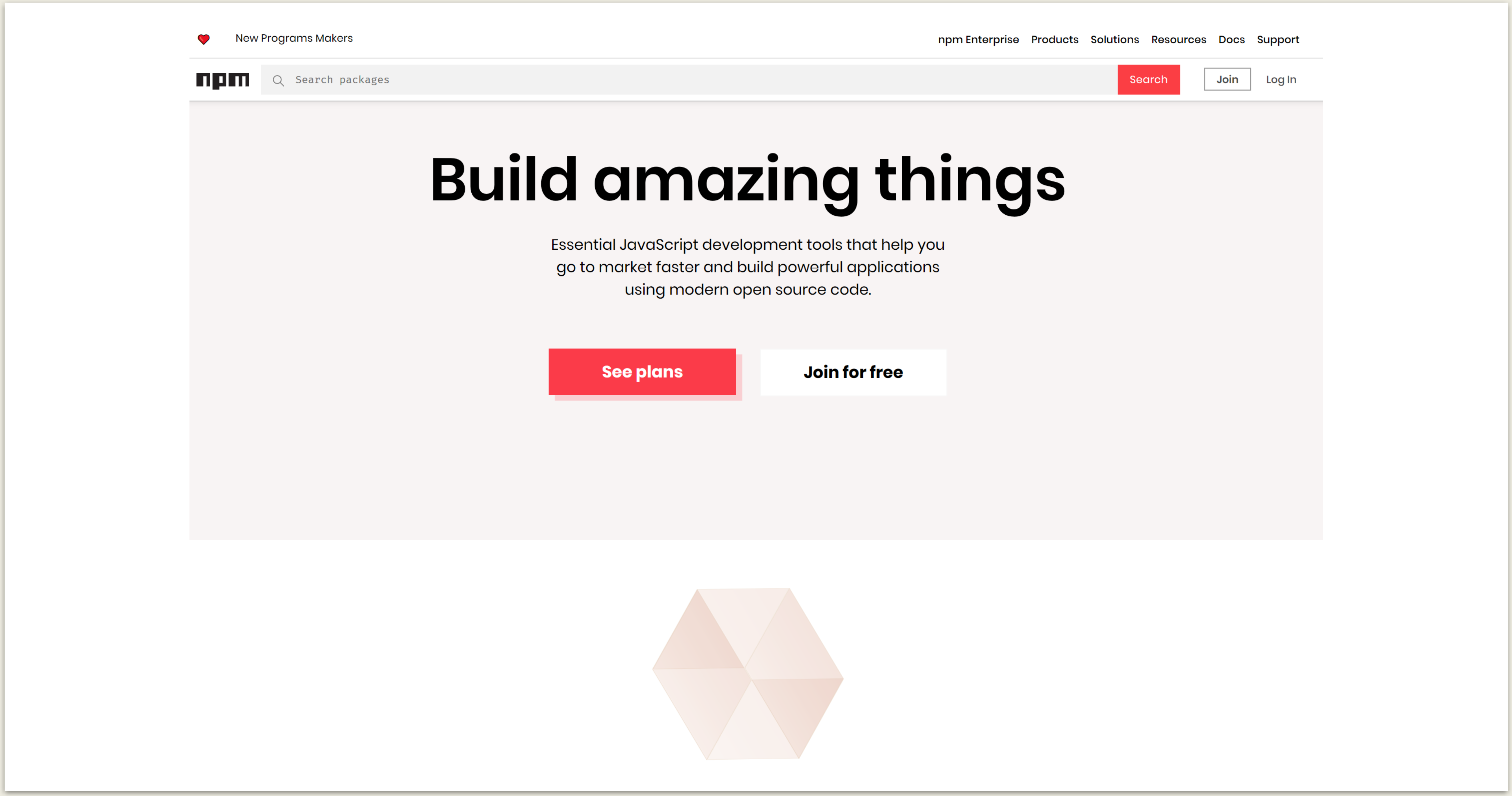
```
fs.readFile('/foo.txt', (err, data)=>{
  // If an error occurred, handle it (throw, propagate, etc)
  if(err) {
    console.log('Unknown Error');
    return;
    }
  // Otherwise, log the file contents
  console.log(data);
});
```

Error object

Successful response data

If no error, *err* will be set to null

Node Modules



New Programs Makers

npm Enterprise Products Solutions Resources Docs Support

Search packages

Search

Join

Log In

Build amazing things

Essential JavaScript development tools that help you go to market faster and build powerful applications using modern open source code.

See plans

Join for free

Node Modules

- Node has a small core API
- Most applications depend on third party modules
- Curated in online registry called the Node Package Manager system (NPM)
- NPM downloads and installs modules, placing them into a **node_modules** folder in your current folder.

NPM init

- You can use NPM to manage your node projects
- Run the following in the root folder of your app/project:

```
npm init
```

- This will ask you a bunch of questions, and then create a package.json for you.
- It attempts to make reasonable guesses about what you want things to be set to, and then writes a package.json file with the options you've selected.

Node Modules

- To install NPM modules, navigate to the application folder and run “npm install”. For example :

  npm install express --save
- This installs into a “**node_module**” folder in the current folder.
- The **--save** bit updates your package.json with the dependency
- To use the module in your code, use:

  ```
  import express from 'express';
  ```
- This loads express from local **node_modules** folder.

Global Node Modules

- Sometimes you may want to access modules from the shell/command line.
- You can install modules that will execute globally by including the **'-g'.**
- Example, **Grunt** is a Node-based software management/build tool for Javascript.

```
npm install -g grunt-cli
```

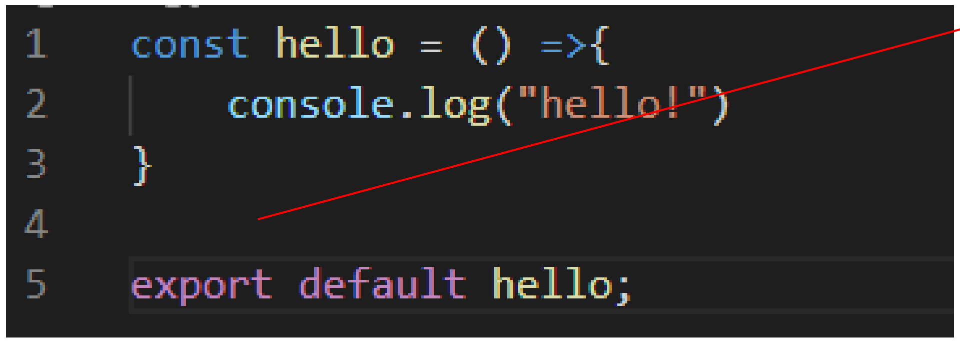
- This puts the **“grunt”** command in the system path, allowing it to be run from any directory.

NPM Common Commands

Common npm commands:
– **npm init** *initialize a package.json file*
– **npm install <package name> -g** *install a package, if – g option is given package will be installed globally,* ***--save*** *and* ***--save-dev*** *will add package to your dependencies*
– **npm install** *install packages listed in package.json*
– **npm ls –g** *listed local packages (without –g) or global packages (with –g)*
– **npm update <package name>** *update a package*

Creating your own Node Modules

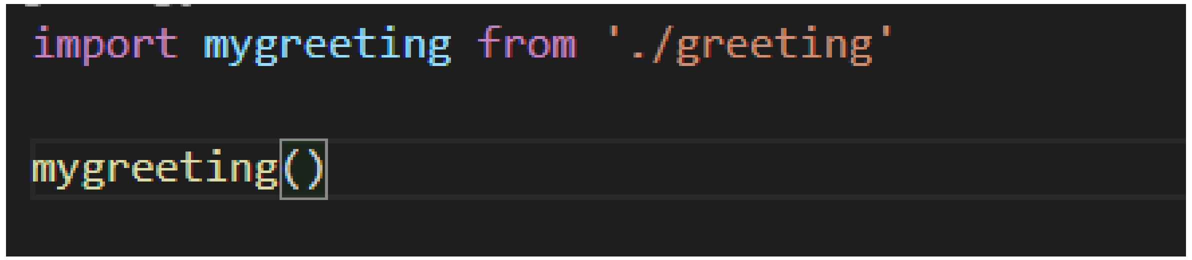
- We want to create the following module called **greeting.js:**



```javascript
const hello = () =>{
    console.log("hello!")
}

export default hello;
```

Export defines what import returns

- To access in our application, **index.js:**

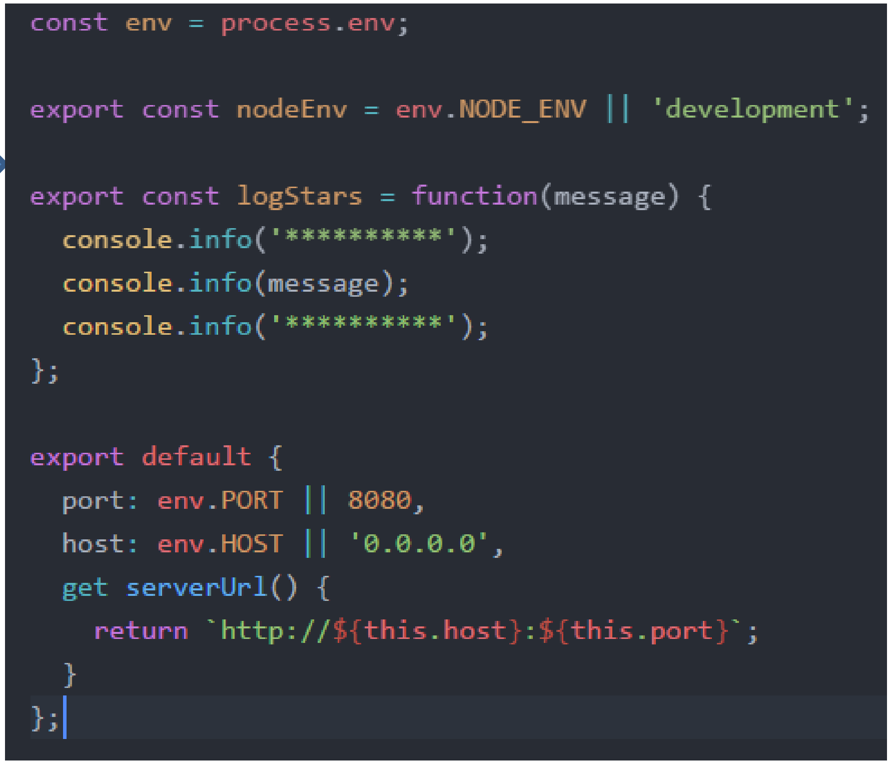


```javascript
import mygreeting from './greeting'

mygreeting()
```

Creating your own Node Modules

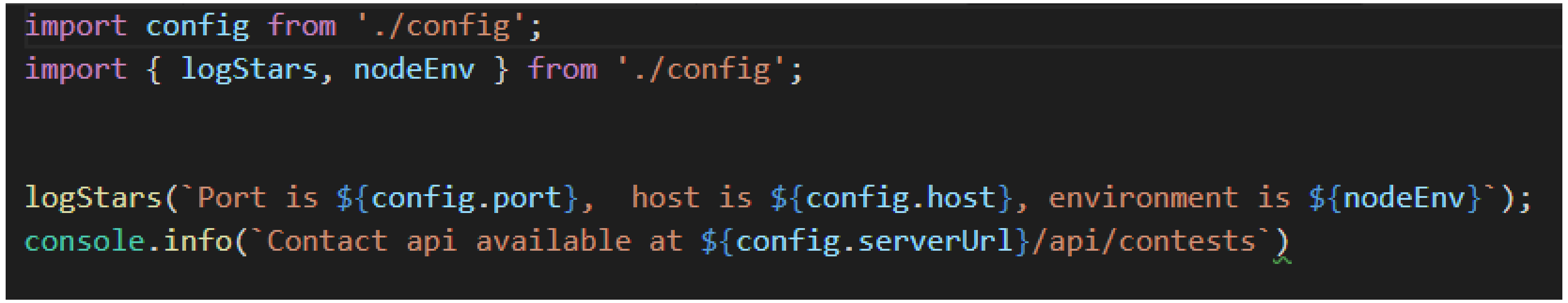
- Exporting Multiple Properties

Config.js



```
const env = process.env;

export const nodeEnv = env.NODE_ENV || 'development';

export const logStars = function(message) {
  console.info('**********');
  console.info(message);
  console.info('**********');
};

export default {
  port: env.PORT || 8080,
  host: env.HOST || '0.0.0.0',
  get serverUrl() {
    return `http://${this.host}:${this.port}`;
  }
};
```

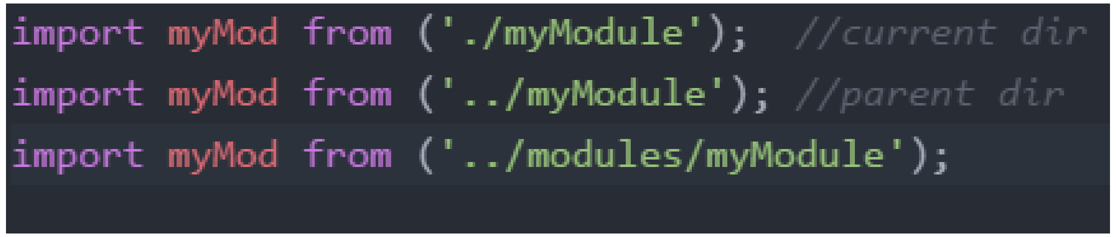
- Accessing in other scripts



```
import config from './config';
import { logStars, nodeEnv } from './config';


logStars(`Port is ${config.port},  host is ${config.host}, environment is ${nodeEnv}`);
console.info(`Contact api available at ${config.serverUrl}/api/contests`)
```

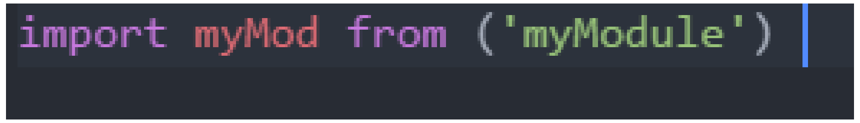
The import search

- Import searches for modules based on path specified:



```
import myMod from ('./myModule');  //current dir
import myMod from ('../myModule'); //parent dir
import myMod from ('../modules/myModule');
```

- Just providing the module name will search in **node_modules** folder



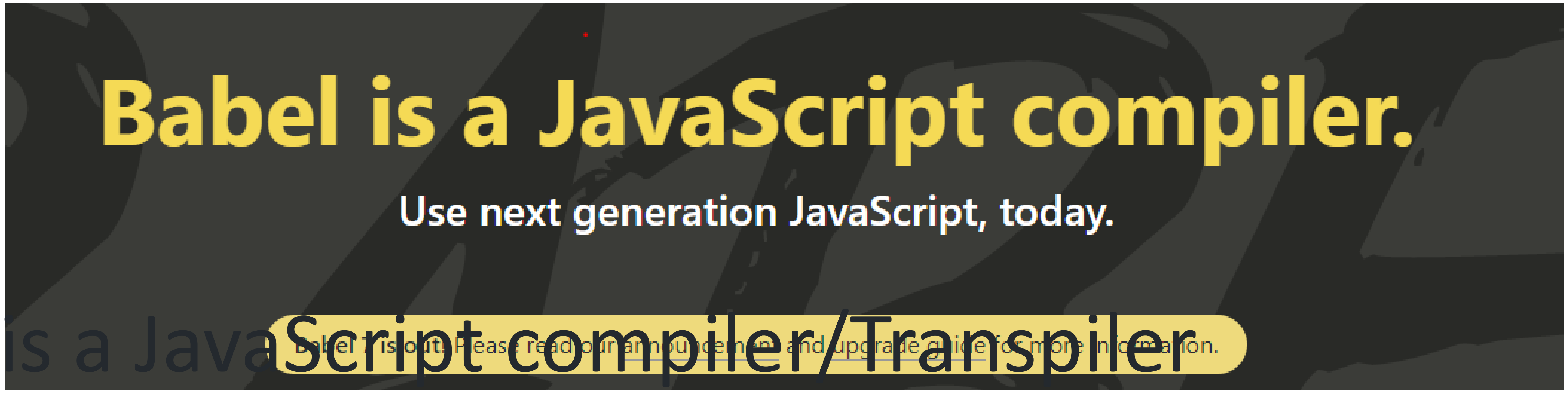
```
import myMod from ('myModule')
```

Environment/Structure for Labs

Tools and Technologies

- Tools:
- VS Code
- Postman (or equivalent)
- Technologies
- Node v12.18.4 or closer
- Express.js
- Mongo
- JSON Web Tokens

Babel


Babel is a JavaScript compiler.

Use next generation JavaScript, today.

Babel 7 is out! Please read our announcement and upgrade guide for more information.

Babel is a JavaScript compiler/Transpiler

Convert the latest versions of Javascript code into a backwards compatible version of JavaScript in current and older browsers or environments(e.g. Node.js v12.18.4)

Set it up as part of our Node project: see the lab!

Structuring Node Apps

- Node Server Code needs to be structured
  - Manage code base
  - Keeps code maintainable
  - Nodes packaging system supports this approach
- Typical Node.js application code:
  - main app code
  - api implementation code
  - helper code

Example Approach:

- Use a “project root” folder is the top level and contains the “entry point” or main server code
  - Always run npm in this folder to ensure just one node_modules folder
  - Use a **public** folder within the node folder for any static content

Basic Node App Structure

```
/projectroot/            → Root of your actual application
    package.json         → Tells Node and NPM what packages are required
    readme.md
    index.js             → The main entry point for the Express application
    .env                 → Environment variables
    .babelrc             → Bable Transpiler Config
    public/
            /images           Static content (if you need it)
            /stylesheets
            /scripts
            index.html
    node_modules/        → Output directory for all NPM installations
    api/
```