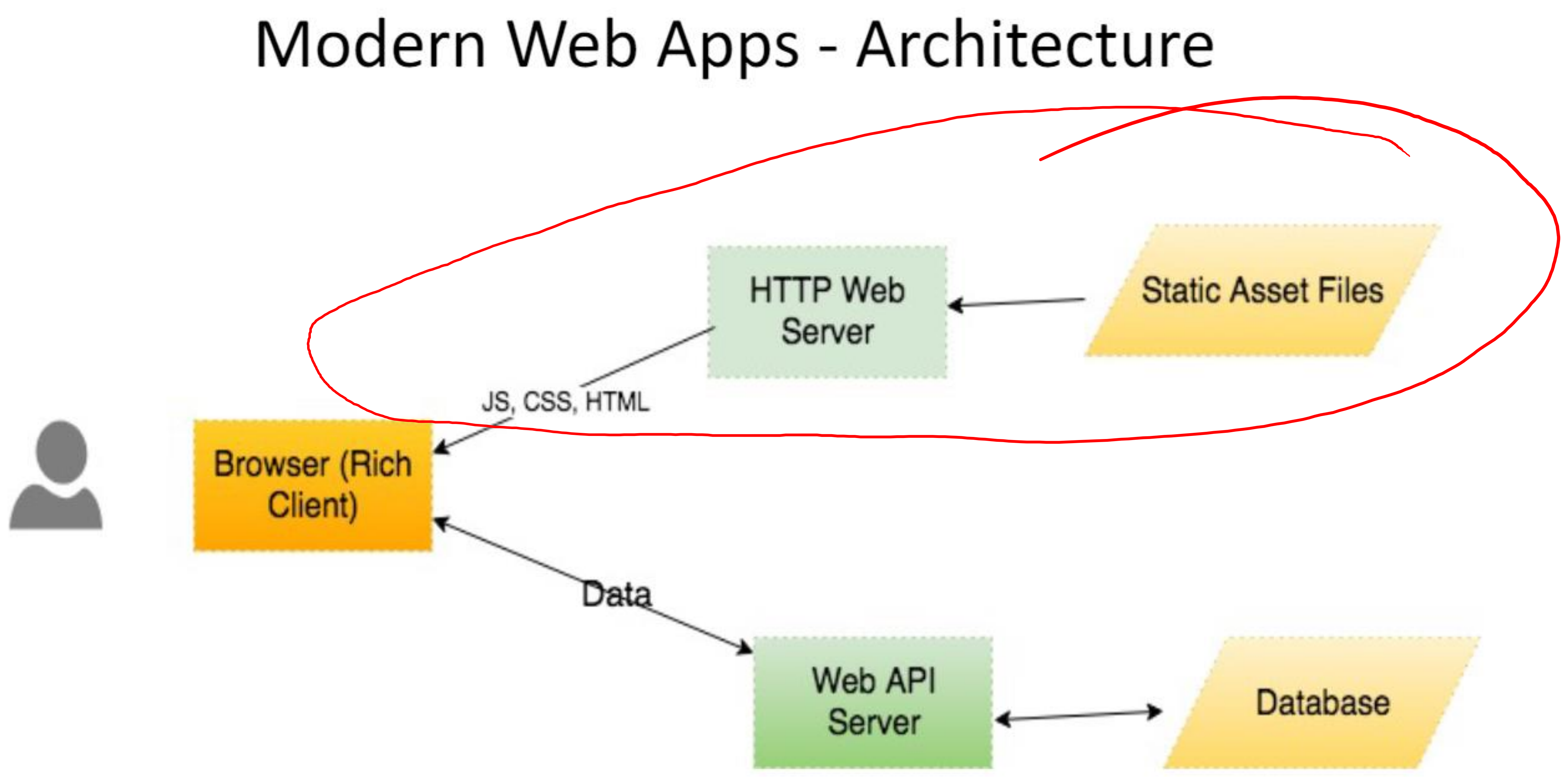
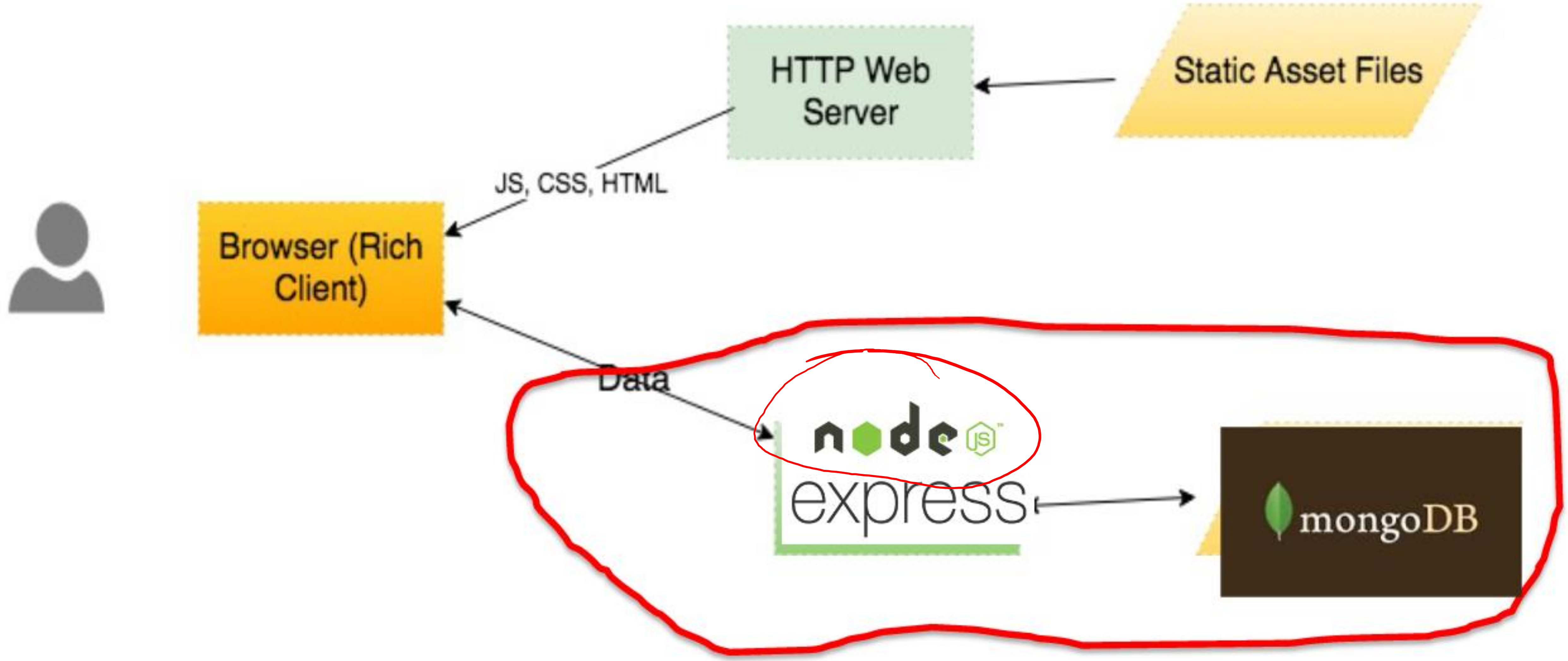
Introduction to Node.js

Frank Walsh
Diarmuid O'Connor

Context


Modern Web Apps - Architecture
HTTP Web Server
Static Asset Files
JS, CSS, HTML
Browser (Rich Client)
Data
Web API Server
Database

Modern Web Apps

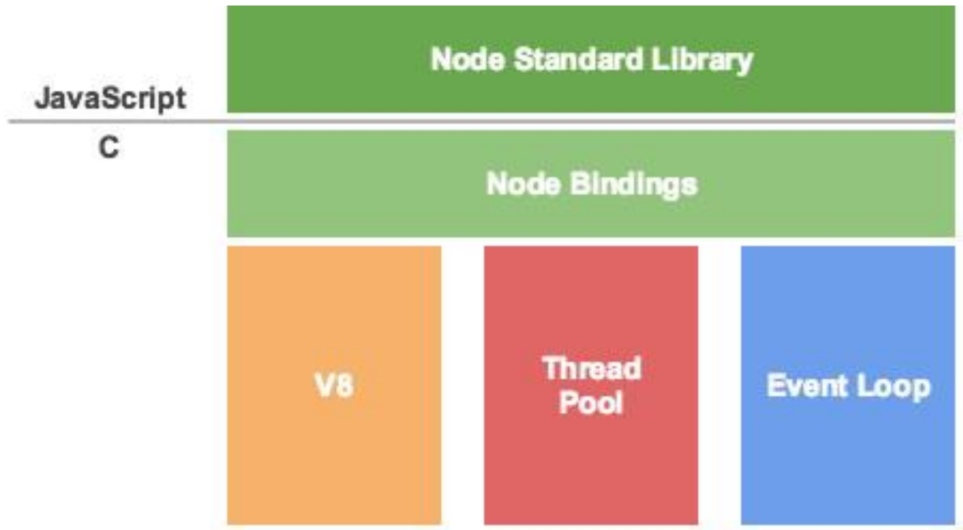


Agenda

- 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.js: 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…

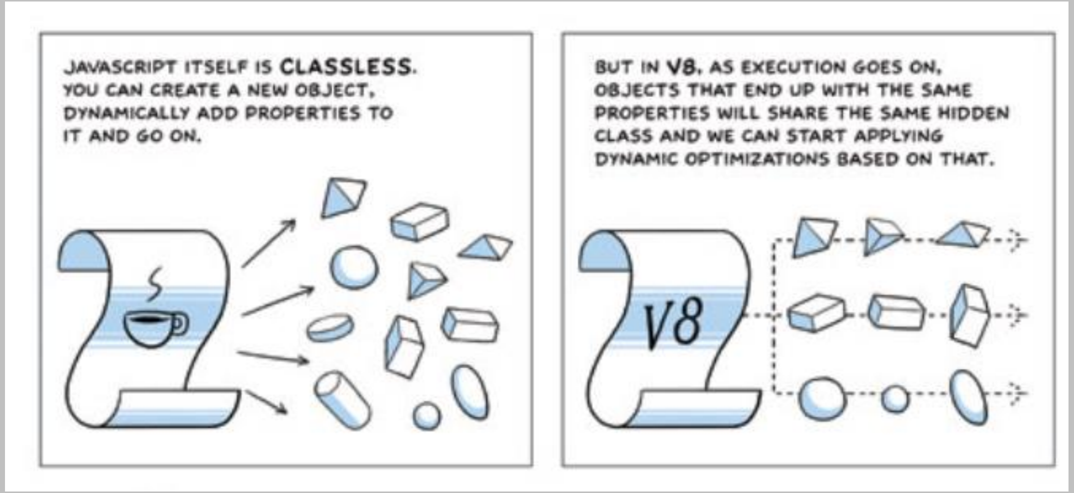


What’s Node: V8.

- 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 database or file
  - network access.
  - Read 4K from Solid State Disk 150,000 nanoseconds ~1GB/sec
  - Round trip over network within same datacentre 500,000 nanoseconds
  - Send IP 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)

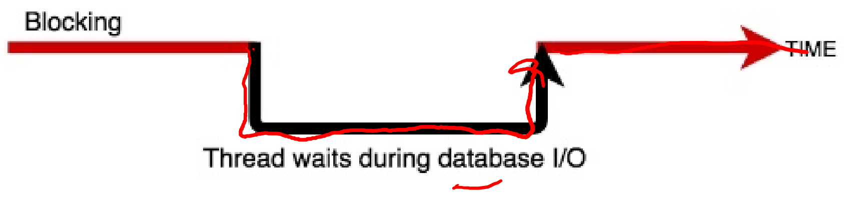
Source: https://gist.github.com/jboner/2841832

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

Blocking/Non-blocking Database Read Example

Blocking

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

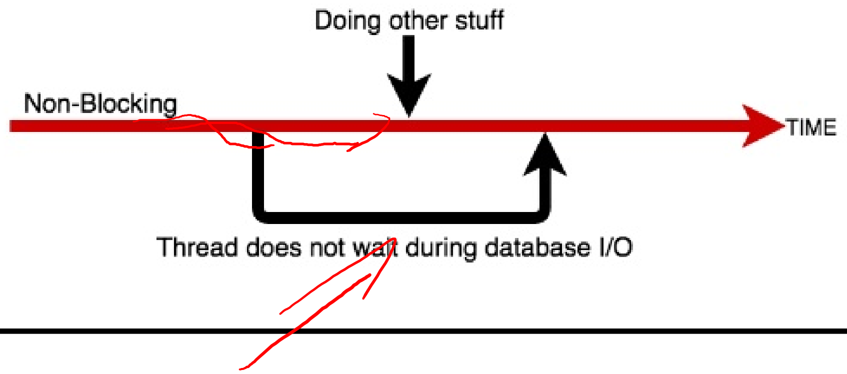


Non-blocking

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

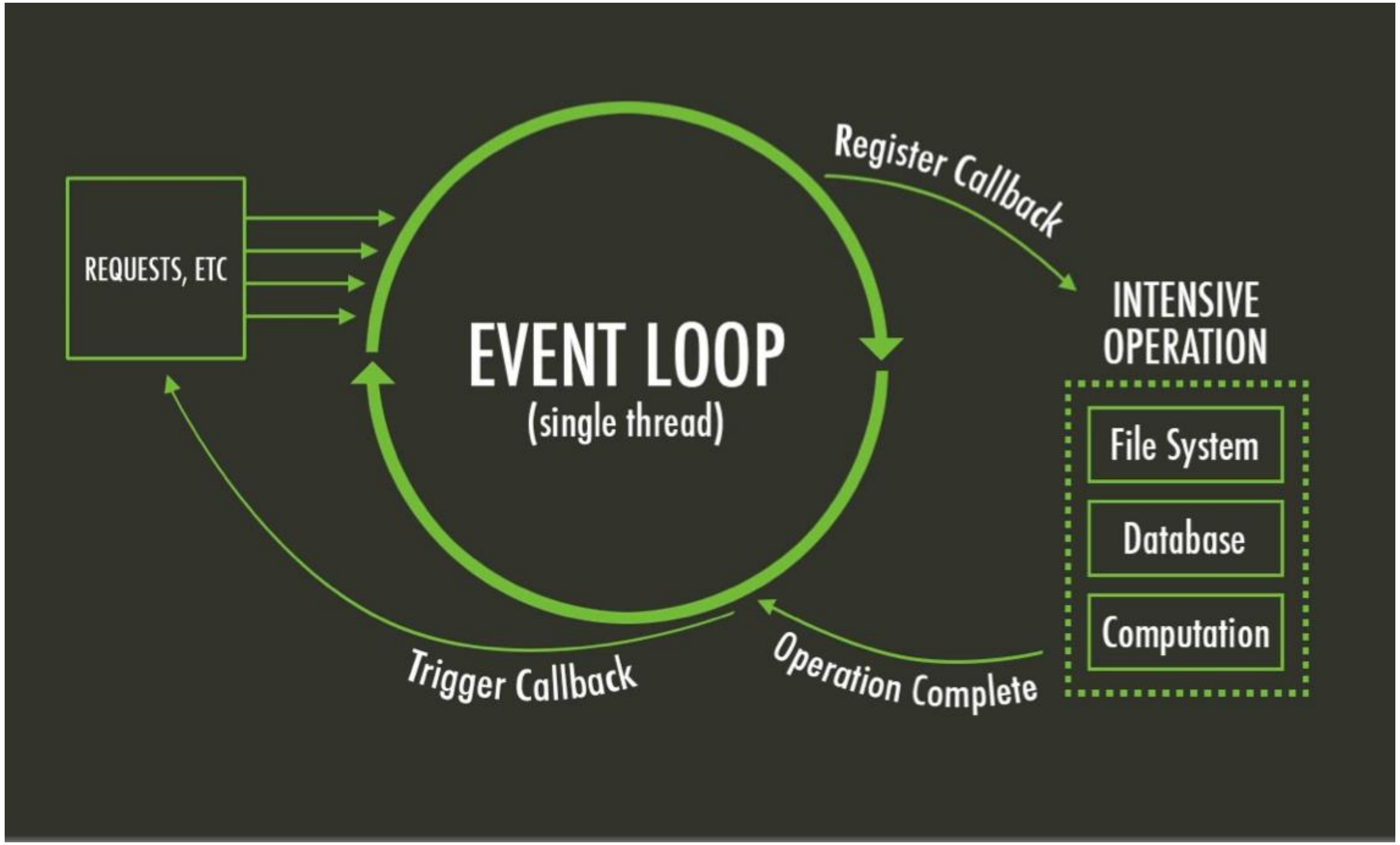
```
import fs from 'fs’;
fs.readFile('./text.txt’,’uft8’, (err, contents) => {
    console.log(contents);
});
console.log('Doing something else');
```

Console output

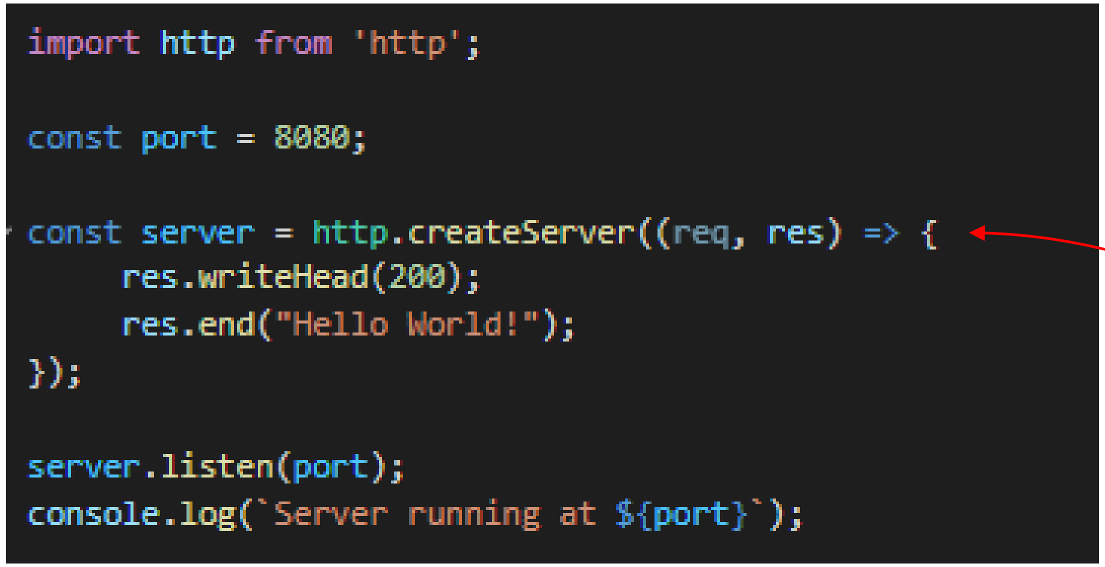
```
Doing something else
Hello World ……
```

The Node Event Loop and Callbacks

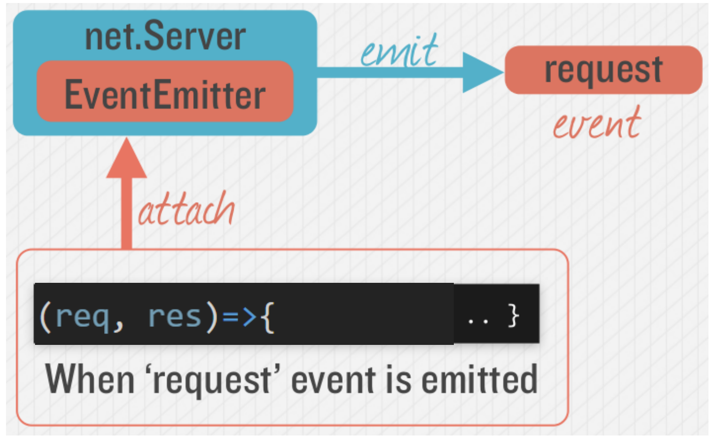
- A **Callback** is a function called at the completion of a given task. This prevents any blocking, and allows other code to be run in the meantime
- The Event Loop checks for known events, registers Callbacks and, triggers callback on completion of operation

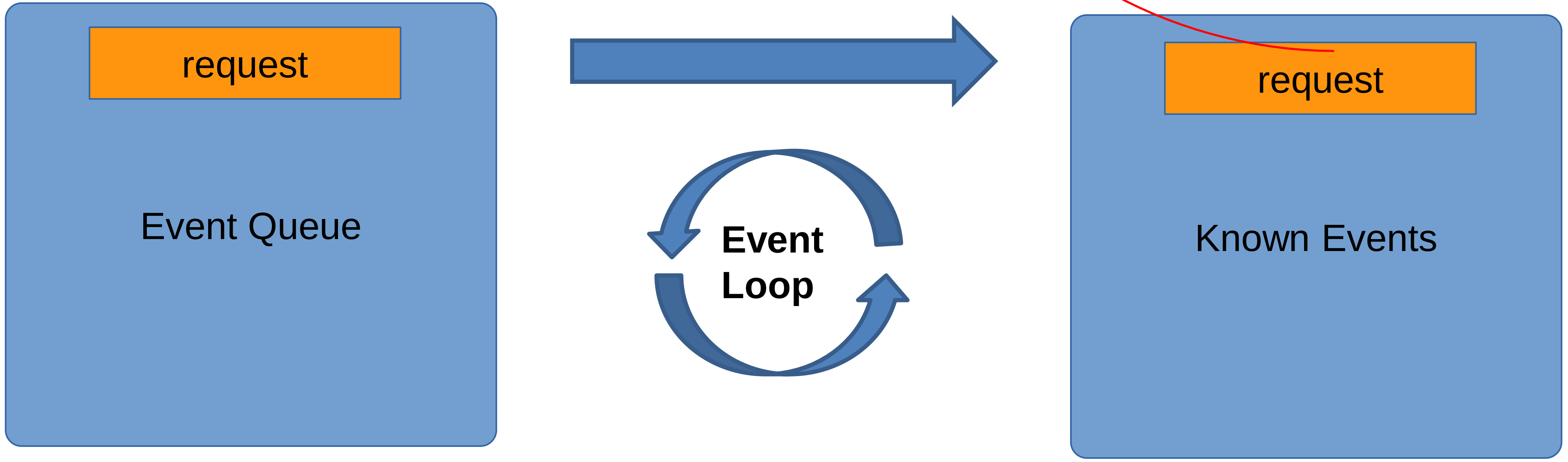


Node.js - Simple HTTP Server



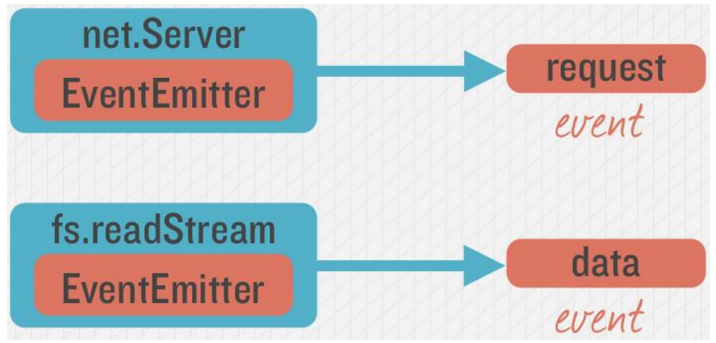
```
import http from 'http';

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}`);
```



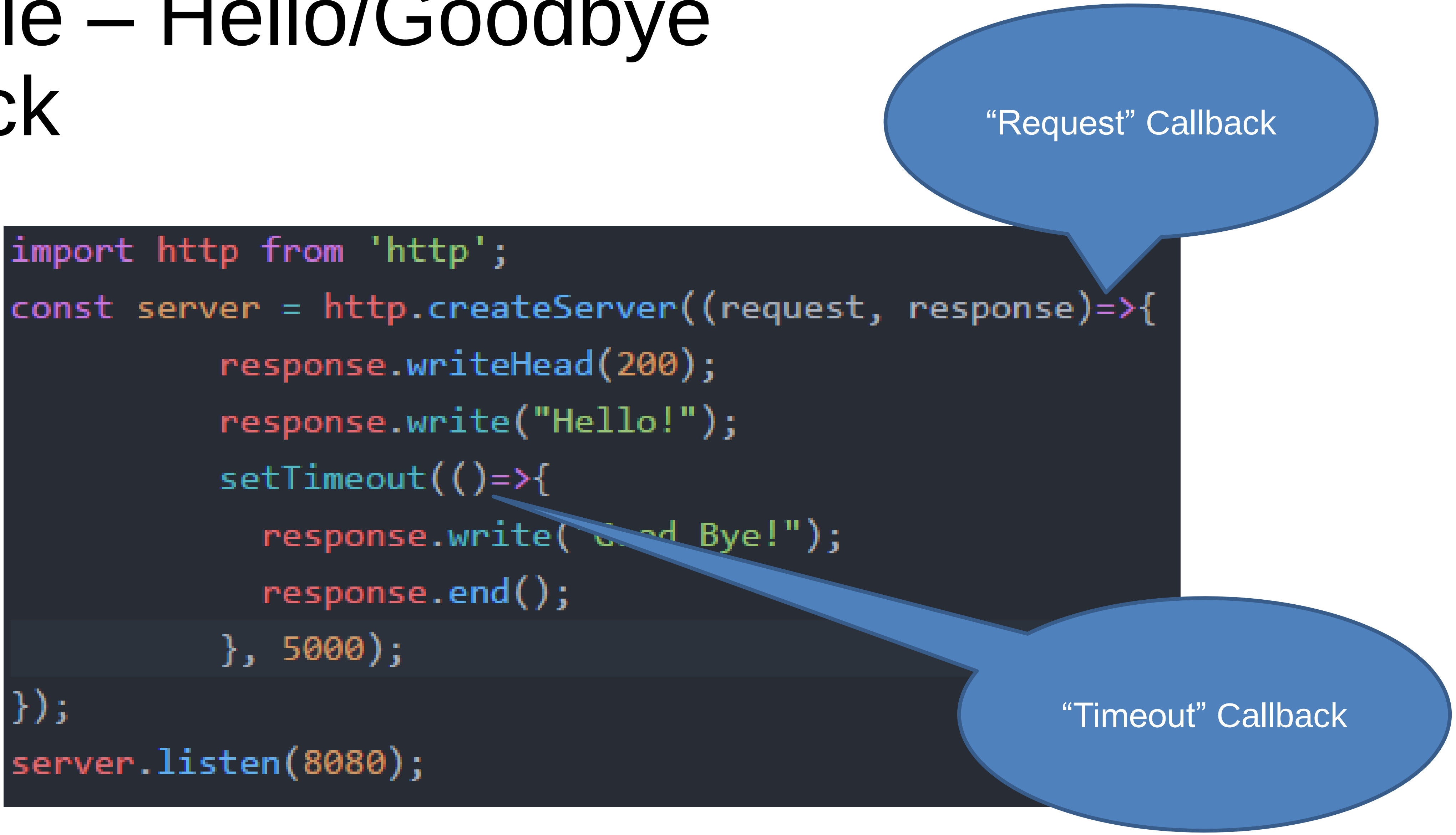


Emitting Event in Node

Many objects can emit events in node.



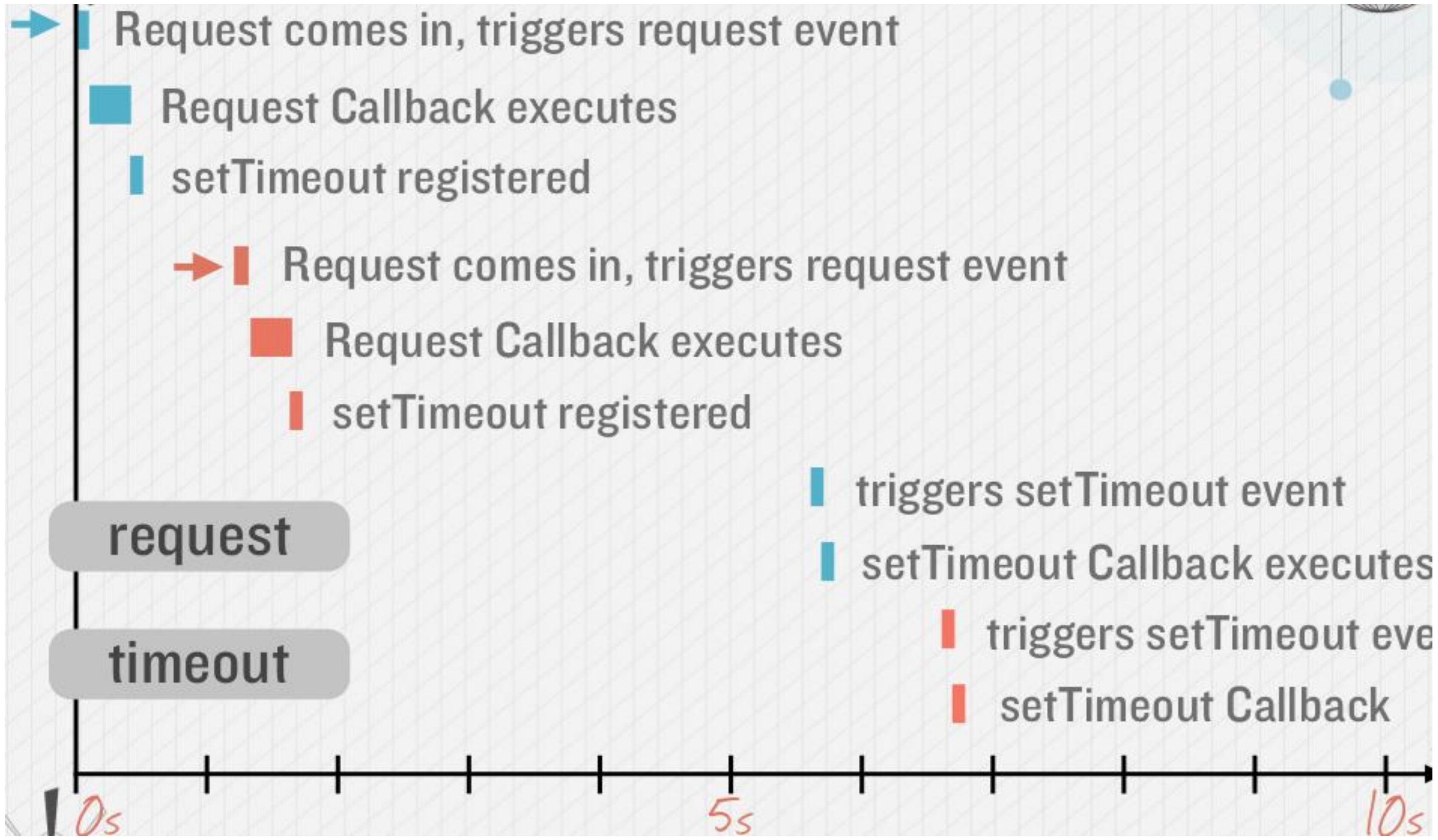
Example – Hello/Goodbye Callback



```
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

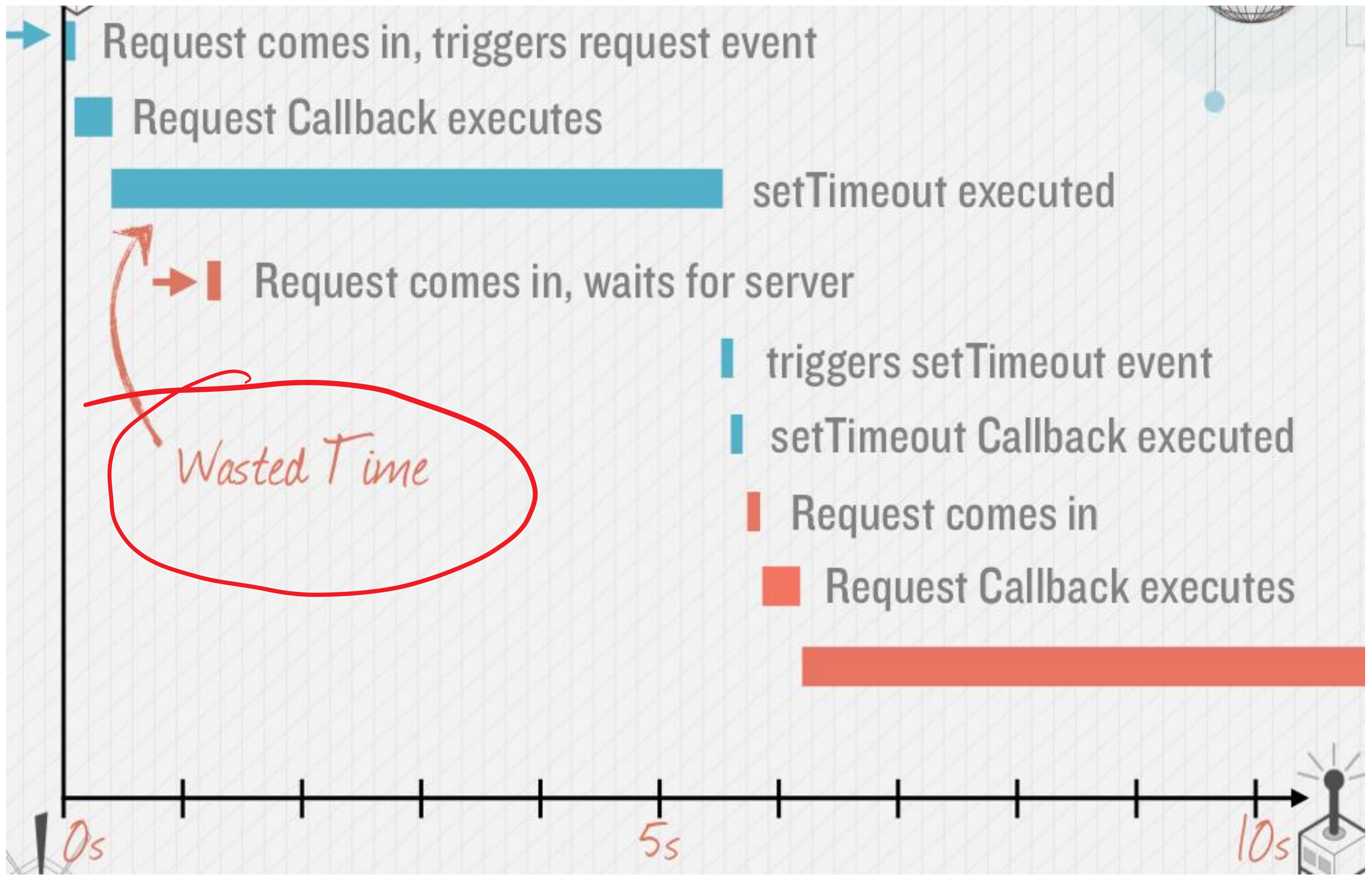
Avoid Blocking Calls in Node.js apps

- 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)

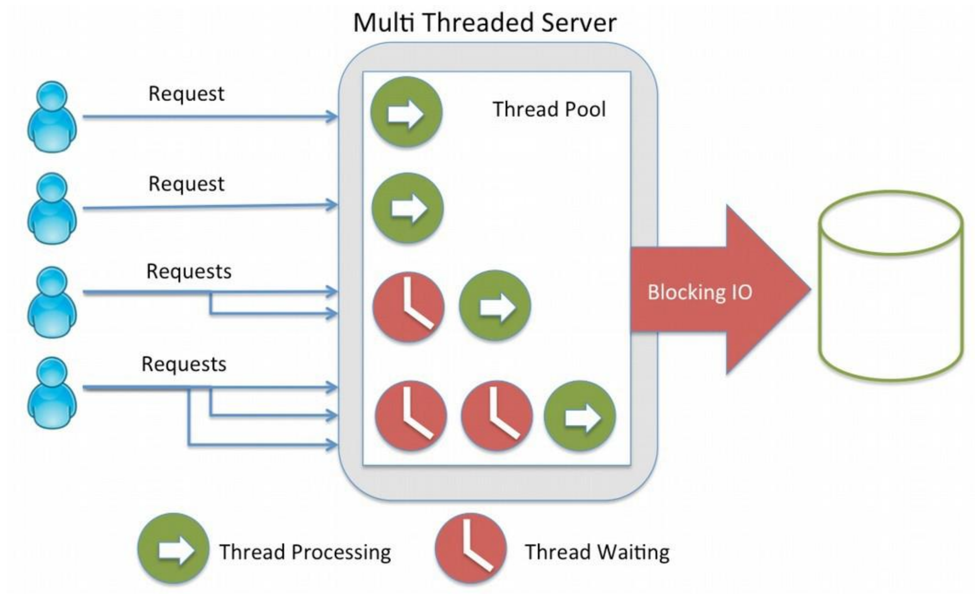
What if setTimeout() blocked…


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

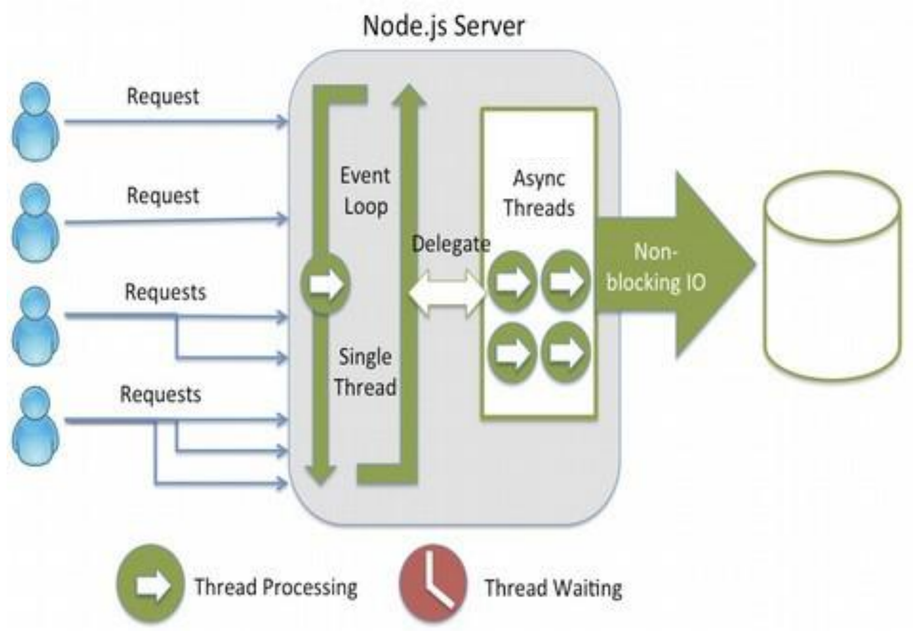
Threads consume resources

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



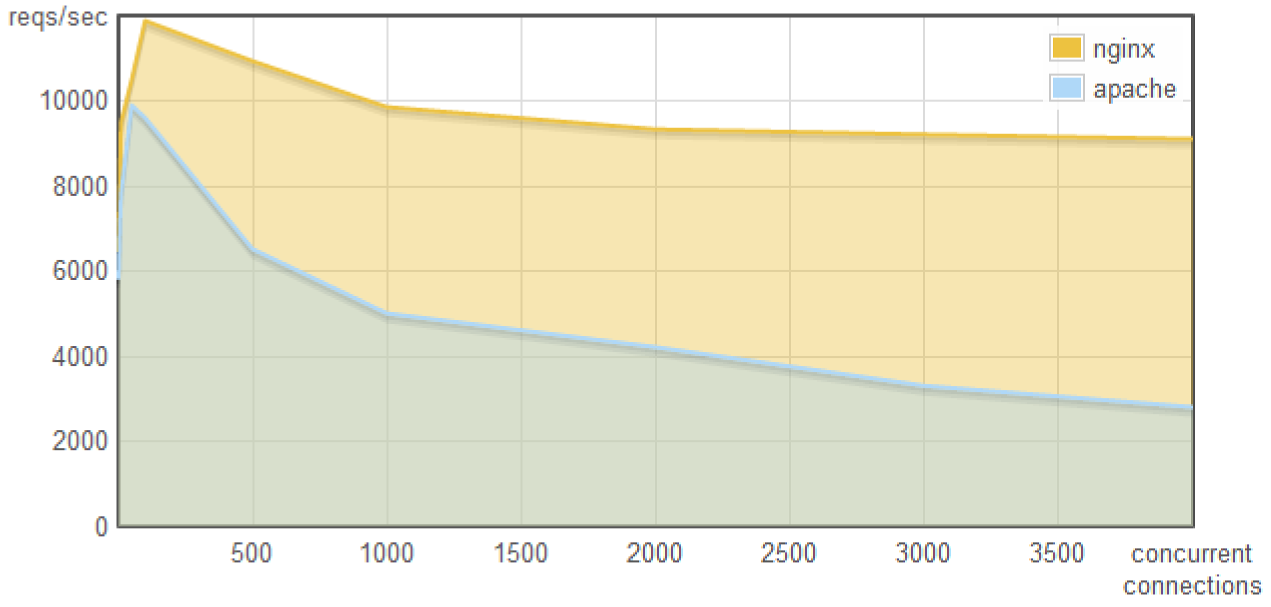
No thread management on single threaded apps

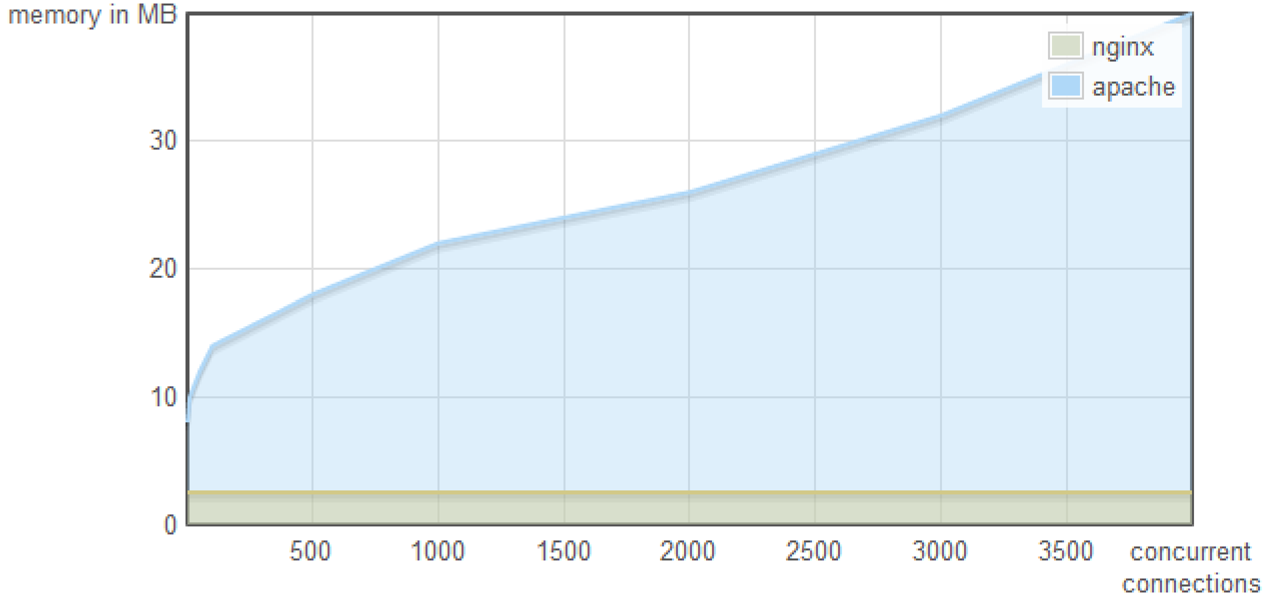
- Just execute “callbacks” when event occurs



Why does it matter...

- This is why:

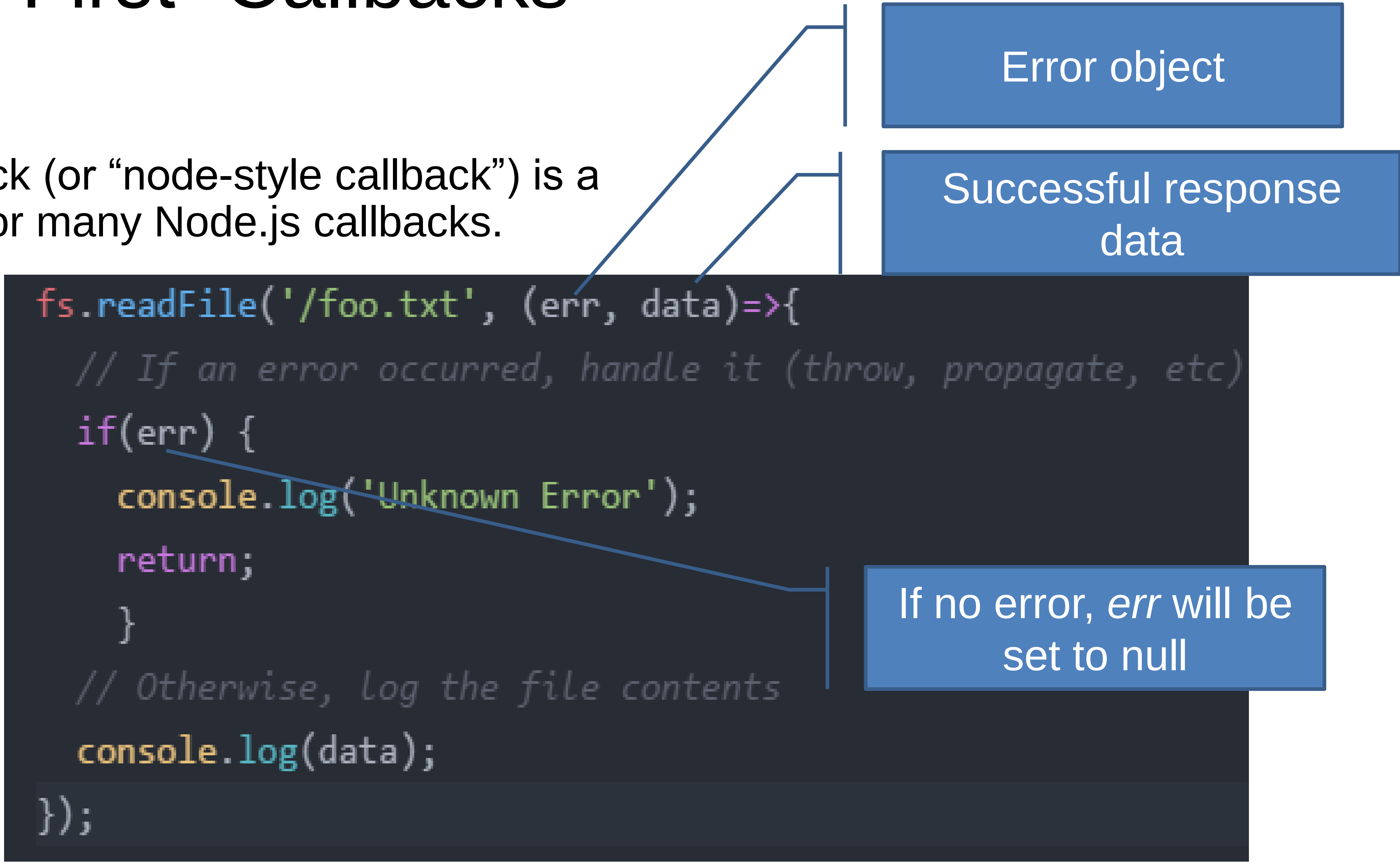




http://blog.webfaction.com/a-little-holiday-present

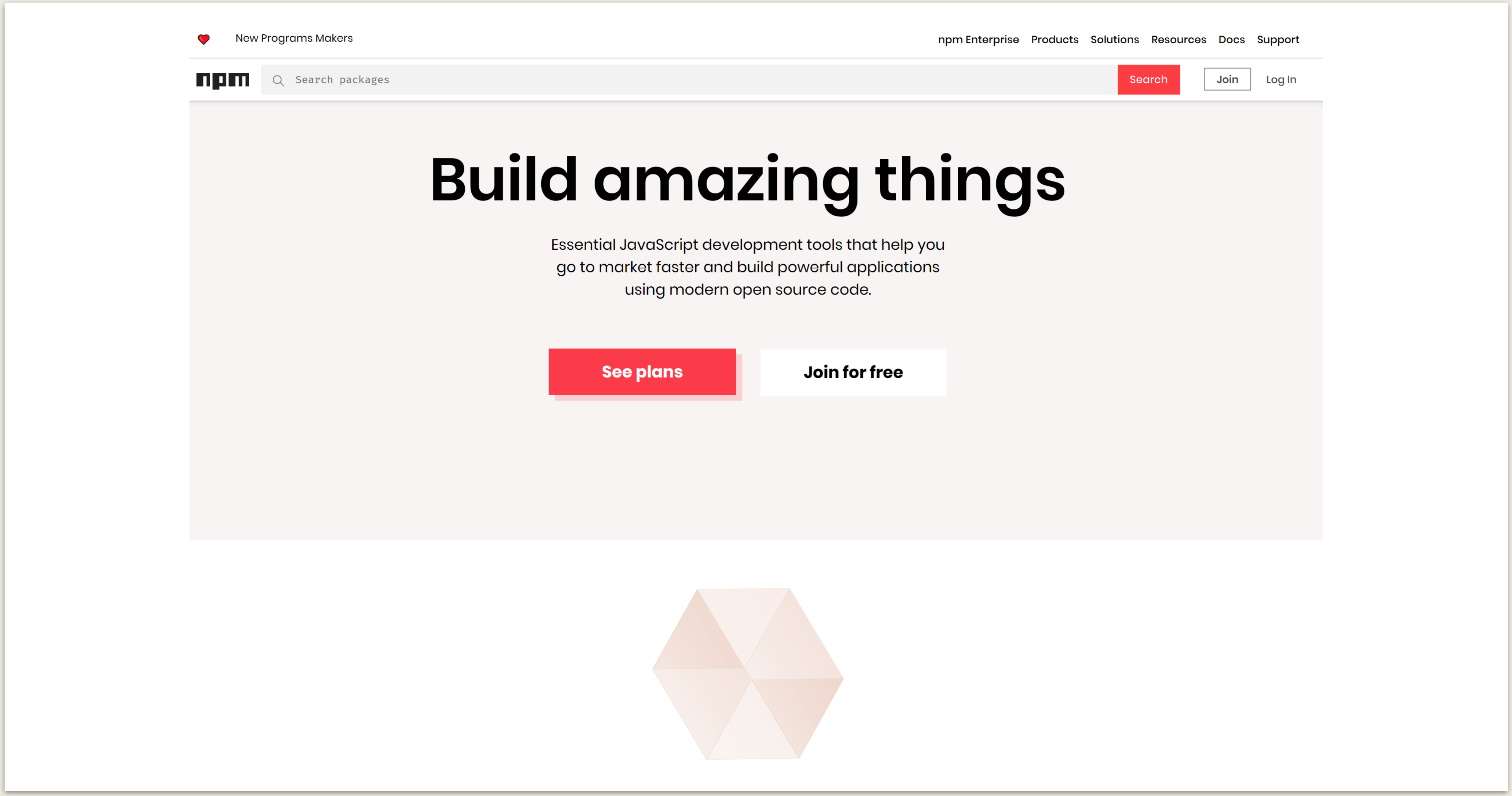
Node “Error First” Callbacks

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);
});
```

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
```

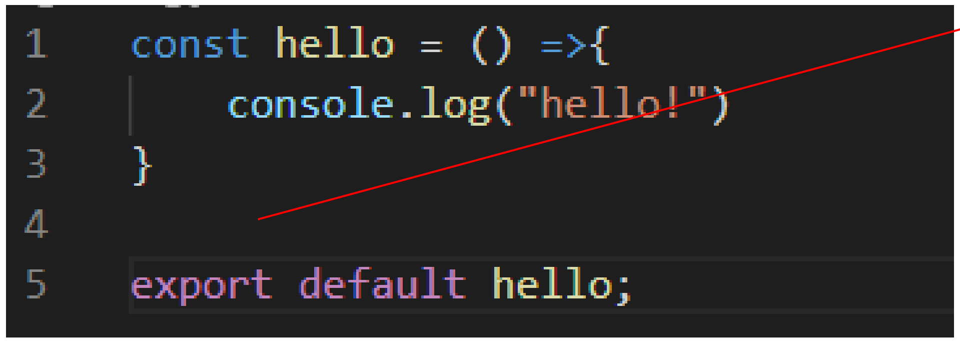
- This puts the **“grunt”** command in the system path, allowing it to be run from any directory.

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

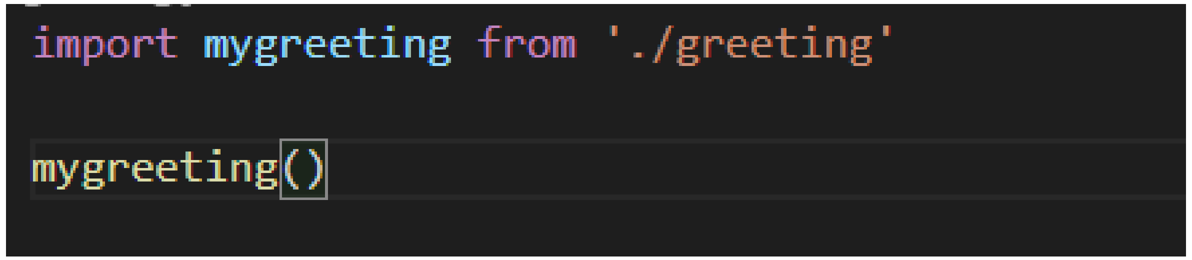
- We want to create the following module called **greeting.js:**



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

export default hello;
```

Export defines what import returns

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

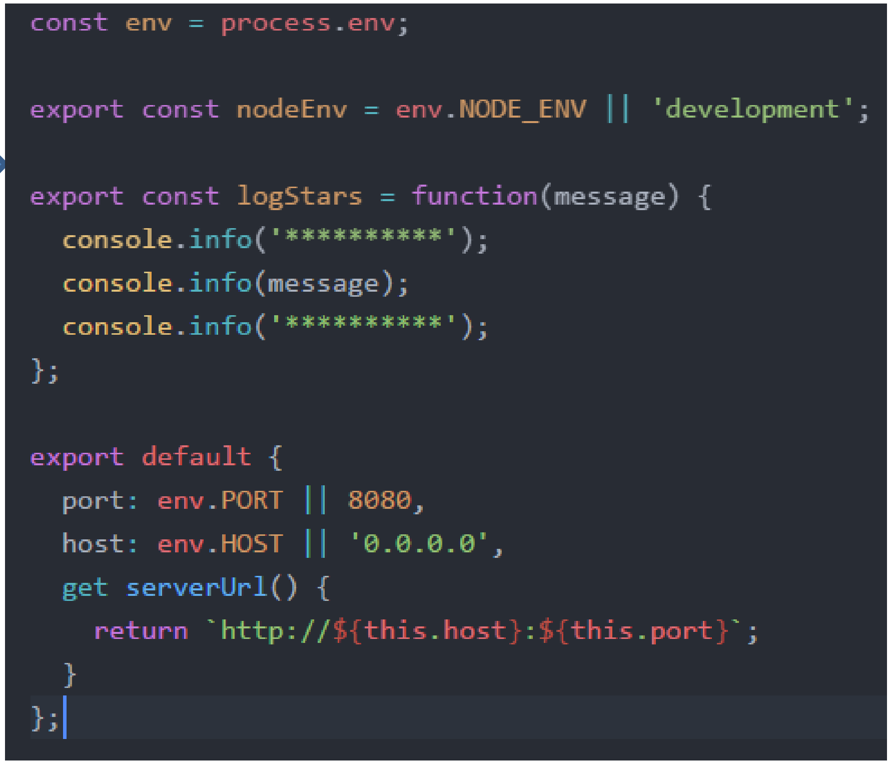


```
import mygreeting from './greeting'

mygreeting()
```

Creating your own Node Modules

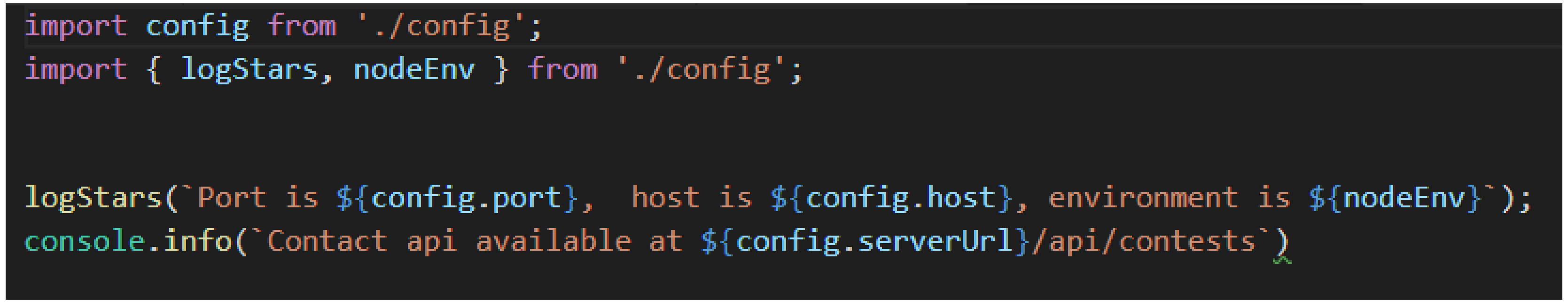
- Exporting Multiple Properties

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}`;
  }
};
```

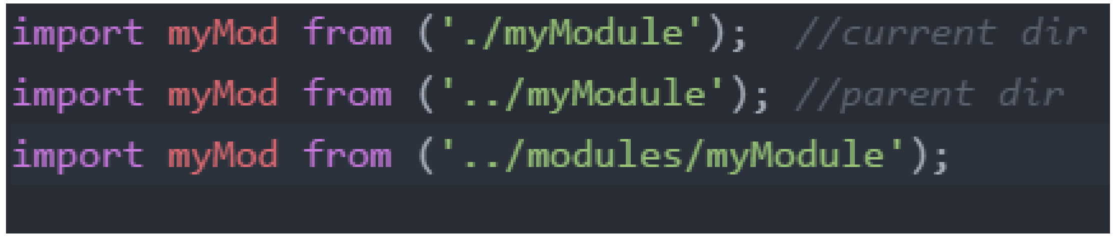
- Accessing in other scripts



```
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`)
```

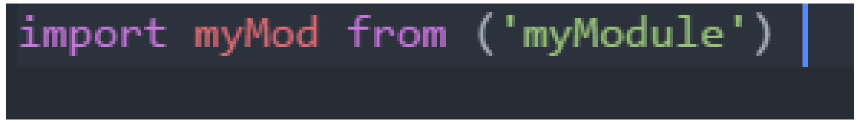
The import search

- 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')
```

Lecture 2:
Environment/Structure for Labs
Web APIs using Express.js

Tools and Technologies

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

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

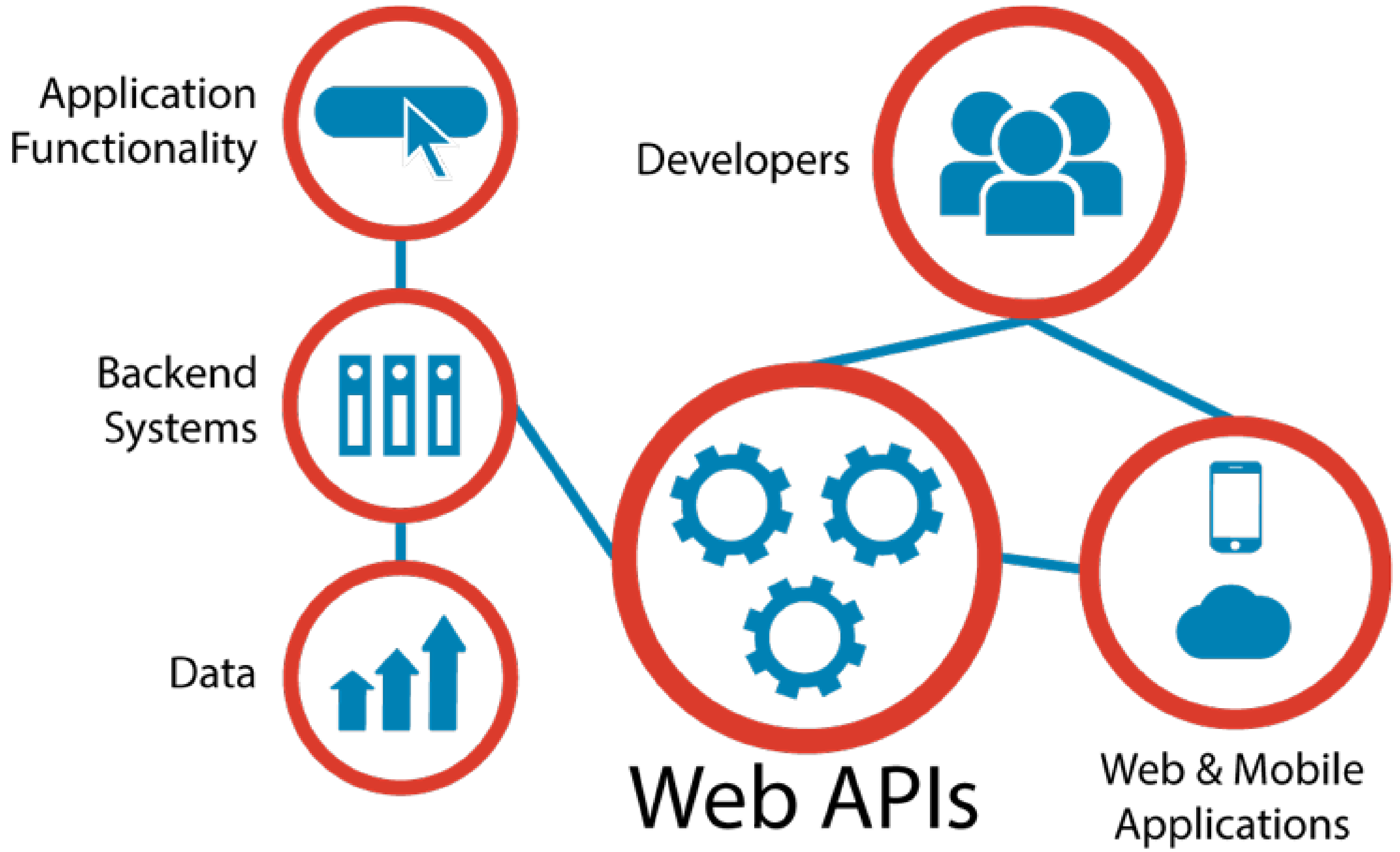
Basic Node App Structure

```
/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. DO
    api/               NOT CHANGE OR MODIFY
```

Web APIs

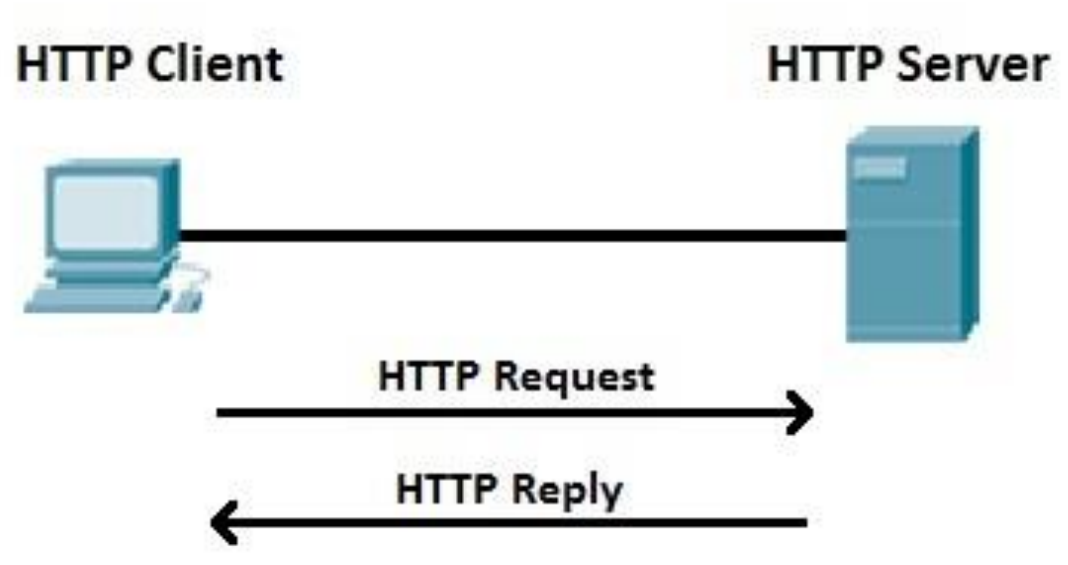
What is a Web Application Programming Interface?

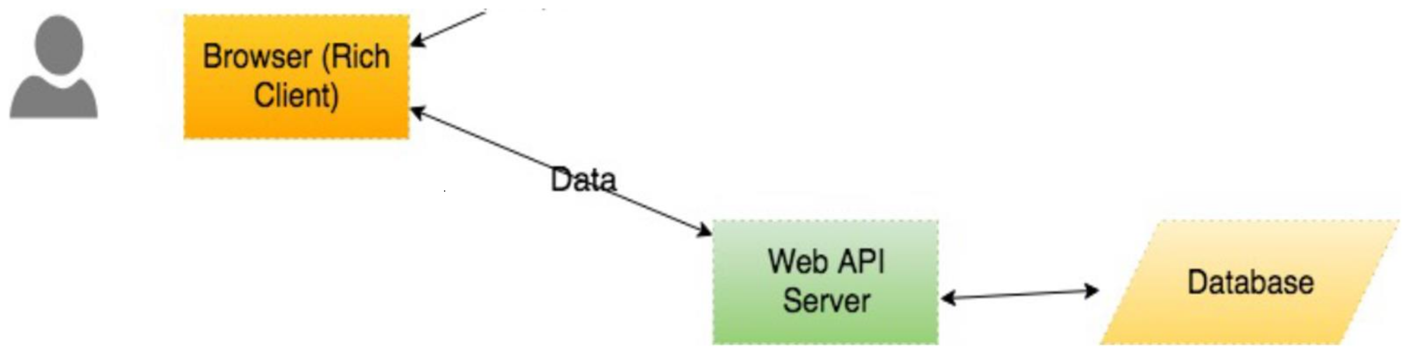
- Interface exposed via the Web
  - Usually via a URL
- Accessed over the web using the **HTTP**
- Uses open standards for data representation
  - JSON
  - XML
- Typical use:
  - Expose application functionality via the web
  - Machine to machine communication
  - Distributed systems



What is HTTP

- HyperText Transfer Protocol
- Your browser communicates using HTTP (HTTP Client)
  - Transfers HTML
- Communicate with APIs using HTTP
- Simple, ubiquitous.
- We will be writing Node.js Apps that listen for and process HTTP requests
- We can test using Postman, a HTTP Client





URL

- A URL (Uniform Resource Locator) uniquely identifies a resource over the web. *protocol://hostname:port/path-and-resource*
- *There* are 4 parts in a URL:
  - *Protocol*: The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
  - *Hostname*: The DNS domain name (e.g., www.nowhere123.com) or IP address (e.g., 192.128.1.2) of the server.
  - *Port*: The TCP port number that the server is listening for incoming requests from the clients.
  - *Path-and-resource-name*: The name and path of the requested resource
- Example, for http://www.myserver.com:8080/api/movies
  - the communication protocol is HTTP
  - The host is www.myserver.com.
  - The port number is TCP port 8080.
  - The path and resource name is “api/movies".

HTTP Protocol (Request)

- HTTP clients (e.g. a browser) translates a URL into a request message according to the specified protocol; and sends the request message to the server.
- For example, a html client may translated the URL *http://www.myserver.com:8080/api/movies* into the following HTTP request message:

```
GET /api/movies/ HTTP/1.1
User-Agent: PostmanRuntime/7.26.5
Accept: application/json
Cache-Control: no-cache
Host: www.myserver.com:8080
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

HTTP Protocol (Response)

- When this request message reaches the server, the server can take either one of these actions:
  1. The server interprets the request received, maps the request into a program kept in the server, executes the program, and returns the output of the program to the client.
  2. The request cannot be satisfied, the server returns an error message.

An example of the HTTP response message is below:

```
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 2019 07:16:26 GMT
Content-Length: 22
Connection: close
Content-Type: application/json

 {“result”:”It Works!”}
```

HTTP Methods

- GET
  - Request objects without sending data
- POST
  - Modify objects with data that you are sending
- PUT
  - Create new objects with data that your are sending
- DELETE
  - Delete objects without sending data

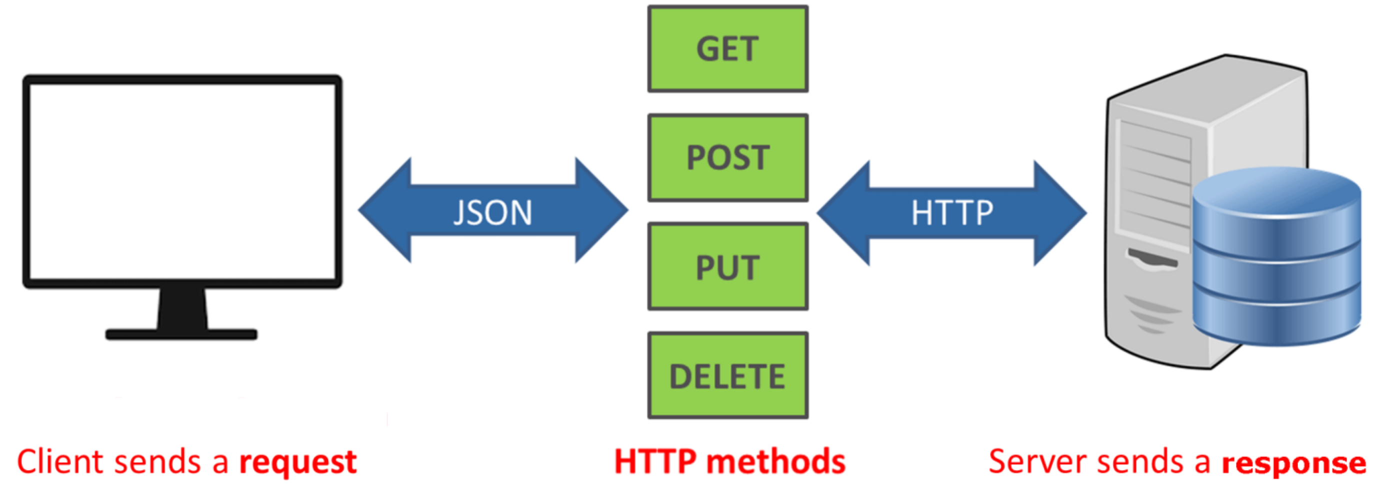
REST

- Short for **Representational State Transfer**
- Set of Principles for how web should be used
- Coined by Roy Fielding
  - One of the HTTP creator

•A set of principles that define how Web standards(HTTP and URIs) can be used.

Key REST Principles

1. Every "thing" has an identity
   - URL
2. Link things together
   - Hypermedia/Hyperlinks
3. Use standard set of methods
   - HTTP GET/POST/PUT/DELETE
   - Manipulate resources through their representations
4. Resources can have multiple representations
   - JSON/XML/png/...
5. Communicate stateless
   - Should **not** depend on server state.

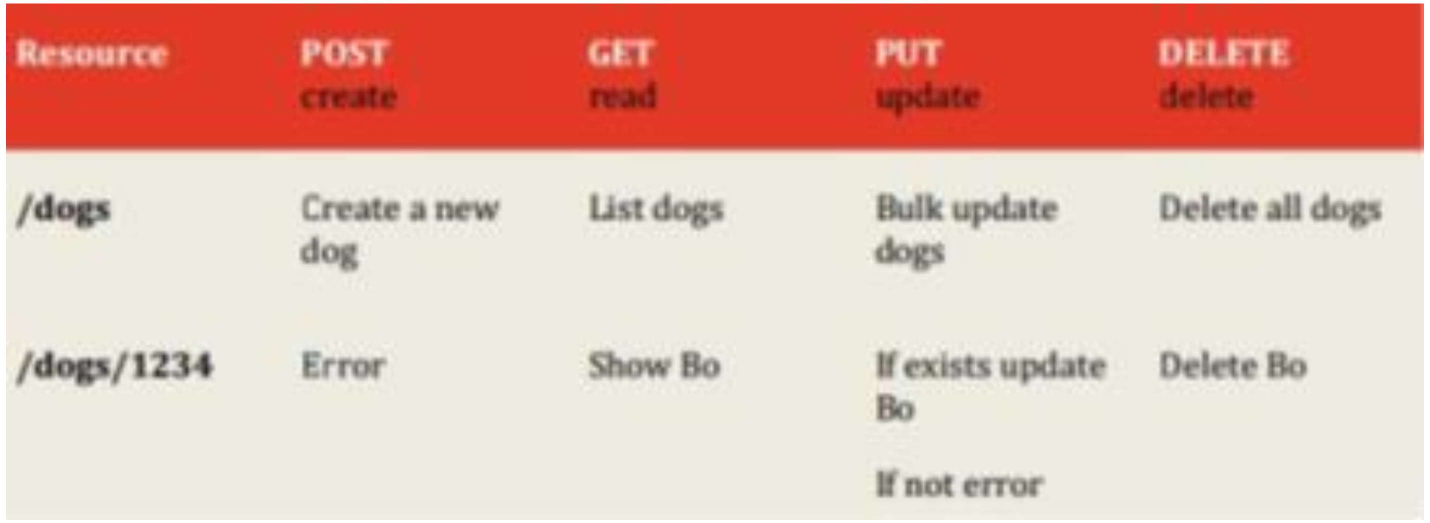


API Design

- Use principle of “developer-first”
  - put target developers’ interests ahead of other considerations
  - Strive for a better developer experience
- Commit to RESTful APIs
- Take a “grammatical” approach to the functionality
- Keep interface simple and intuitive
- Optional: Can use a Interface Description Language & Tools like:
  - RESTful API Markup Language (RAML)
  - Swagger

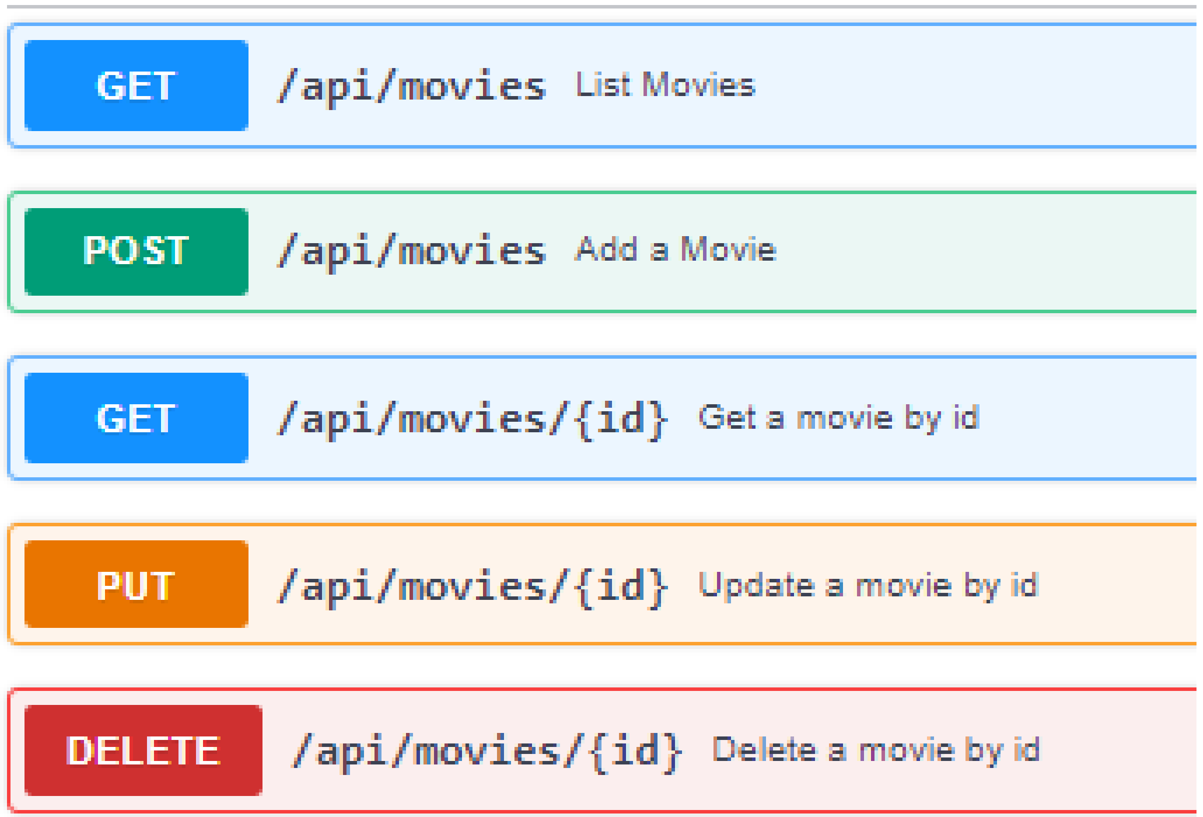
API Design

- In Rest, everything is based around resources
  - the “things” you’re working with are modelled as resources described by URI paths--like /users, /groups, /dogs
  - Notice they are **nouns .**
  - **Verbs in URLs are BAD**
- The things that you do on these things (or nouns) are characterised by the fixed set of HTTP methods
  - What GET,POST,PUT does is something that the designer/developer gets to put into the model.
- The metadata (the adjectives) is usually encoded in HTTP headers, although sometimes in the payload.
- The responses are the pre-established HTTP status codes and body. (200, 404, 500 etc.)
- The representations of the resource are found inside the body of the request and response



Resource	POST create	GET read	PUT update	DELETE delete
/dogs	Create a new dog	List dogs	Bulk update dogs	Delete all dogs
/dogs/1234	Error	Show Bo	If exists update Bo If not error	Delete Bo

API Design Demo: Movies API



GET /api/movies List Movies

POST /api/movies Add a Movie

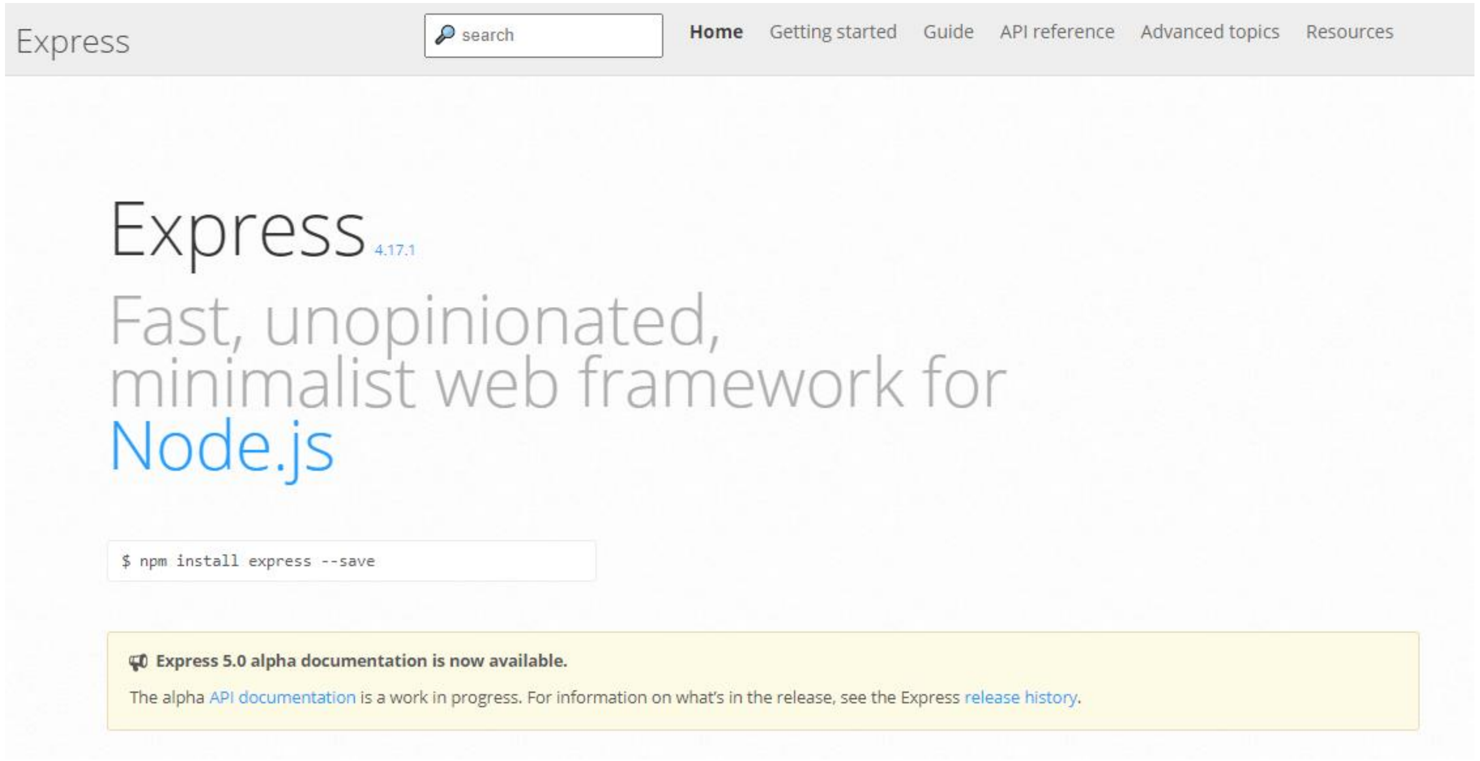
GET /api/movies/{id} Get a movie by id

PUT /api/movies/{id} Update a movie by id

DELETE /api/movies/{id} Delete a movie by id

The Express Package

What is Express?



Express | search | Home | Getting started | Guide | API reference | Advanced topics | Resources

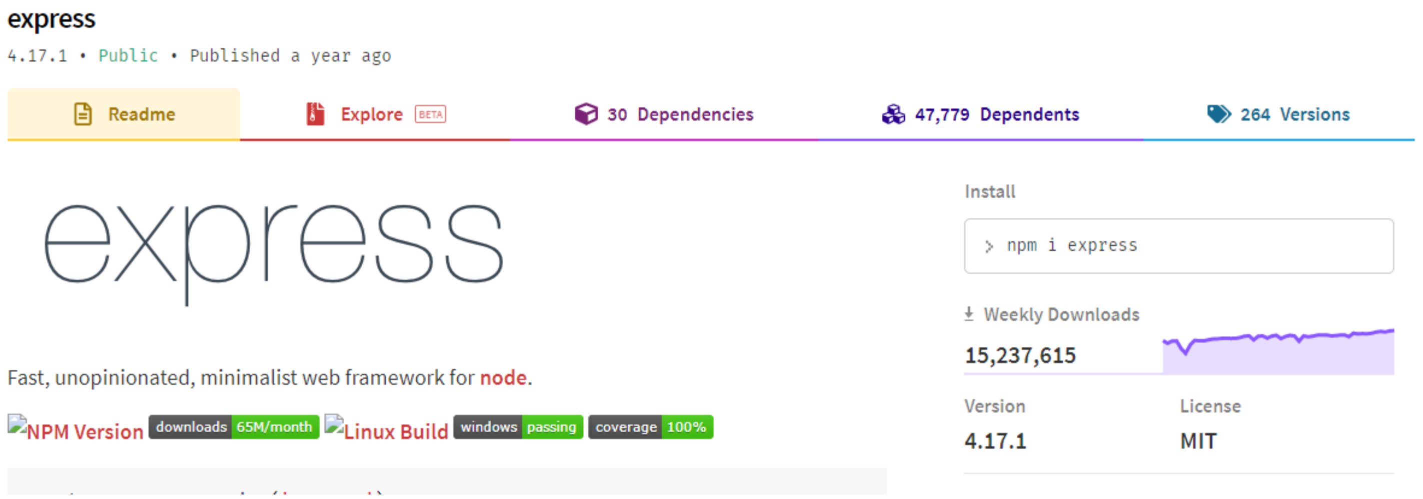
Express 4.17.1

Fast, unopinionated, minimalist web framework for Node.js

```
$ npm install express --save
```

Express 5.0 alpha documentation is now available.

The alpha API documentation is a work in progress. For information on what's in the release, see the Express release history.



express

4.17.1 • Public • Published a year ago

Readme | Explore BETA | 30 Dependencies | 47,779 Dependents | 264 Versions

express

Fast, unopinionated, minimalist web framework for node.

NPM Version | downloads 65M/month | Linux Build | windows passing | coverage 100%

Install

```
> npm i express
```

Weekly Downloads

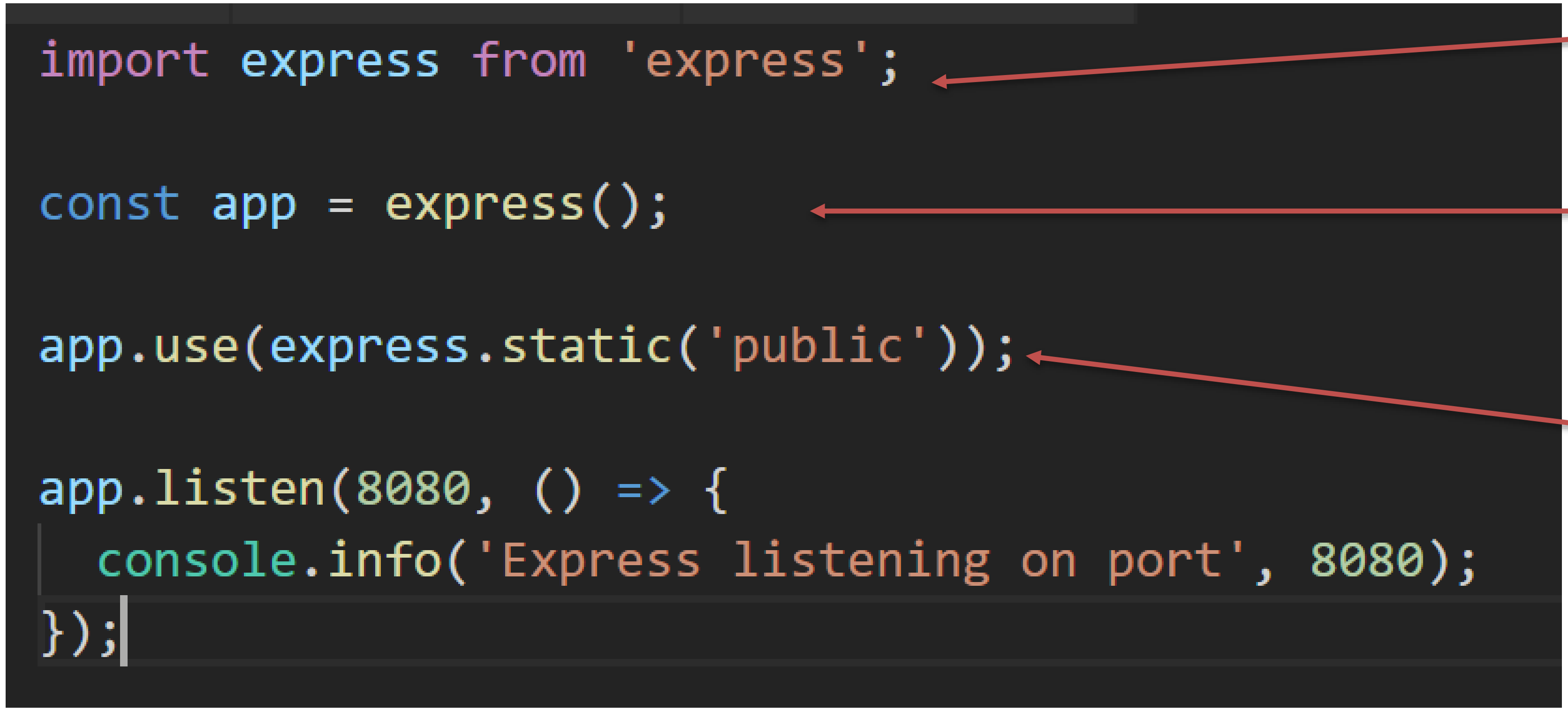
15,237,615

Version	License
4.17.1	MIT

What Express Gives Us...

- Parses arguments and headers
- Easy Routing
  - Route a URL to a callback function
- Sessions
- File Uploads
- Middleware…

Simple Express App (index.js)



```
import express from 'express';

const app = express();

app.use(express.static('public'));

app.listen(8080, () => {
  console.info('Express listening on port', 8080);
});
```

Loads Express module

Instantiates Express server

Define static content for HTTP GET

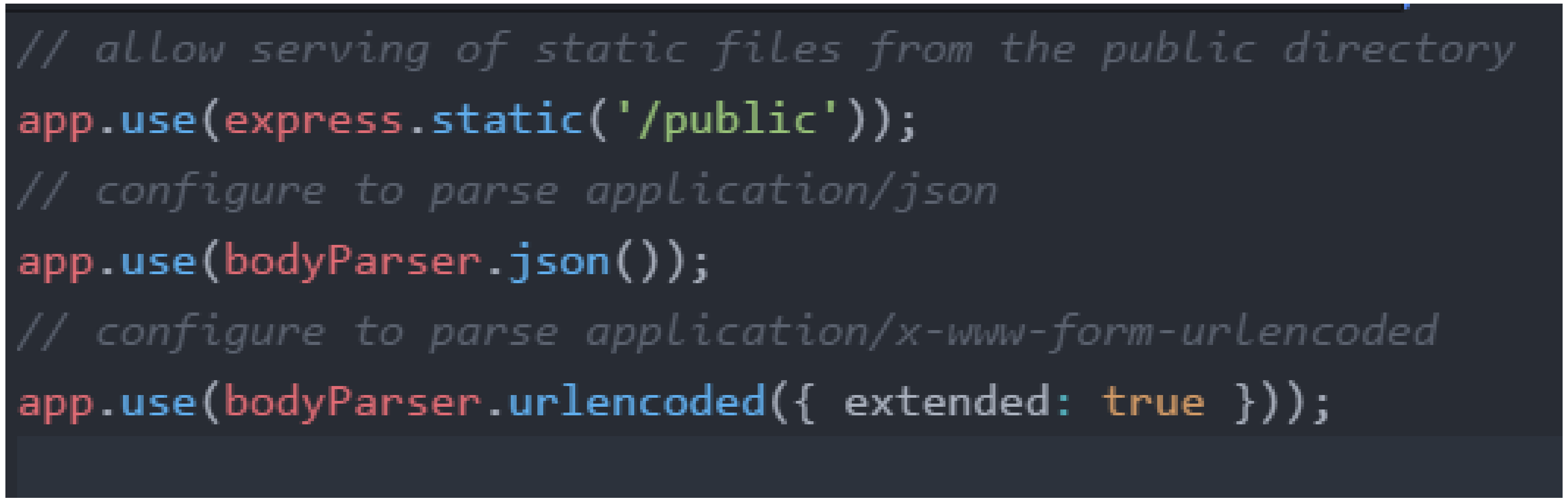
Getting Started with Express

- Installing Express

```
[local install] C:\> npm install express --save
[global install] C:\> npm install express -g
```

Express Configuration

Express allows you to easily configure your web app behaviour...



```
// allow serving of static files from the public directory
app.use(express.static('/public'));
// configure to parse application/json
app.use(bodyParser.json());
// configure to parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));
```

Express: Routing

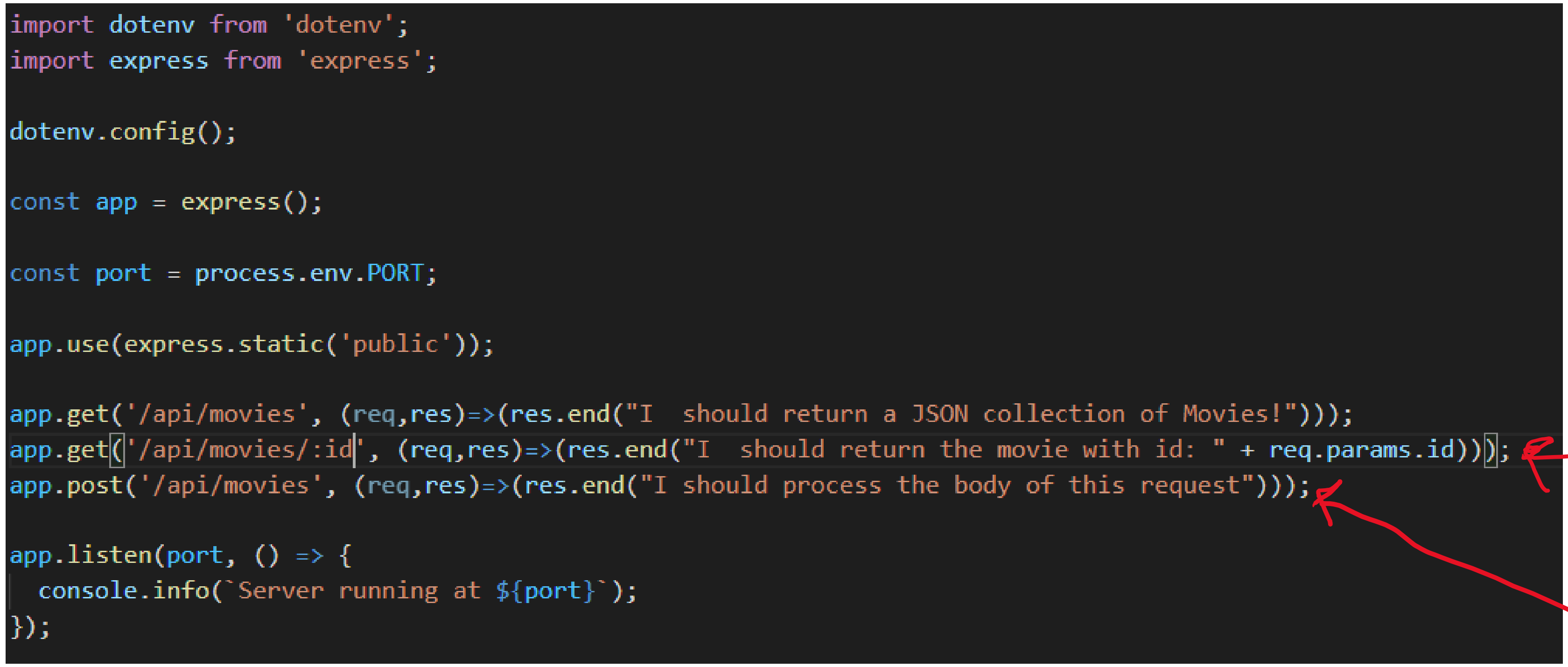
- Routing refers to determining how an application responds to a client request
- The **path** and **HTTP request method** (e.g. GET, POST) are used to **“route”** the request.
- Each route can have **one or more** handler functions, which are executed when the route is matched.

(/*)?

Routing Example

Syntax follows the pattern:

app.[http_verb](path, (req,res)=>{});



```
import dotenv from 'dotenv';
import express from 'express';

dotenv.config();

const app = express();

const port = process.env.PORT;

app.use(express.static('public'));

app.get('/api/movies', (req,res)=>(res.end("I  should return a JSON collection of Movies!")));
app.get('/api/movies/:id', (req,res)=>(res.end("I  should return the movie with id: " + req.params.id)));
app.post('/api/movies', (req,res)=>(res.end("I should process the body of this request")));

app.listen(port, () => {
  console.info(`Server running at ${port}`);
});
```

```
app.post('/api/movies', …);
app.get('/api/movie/:id', …)
```

Handles HTTP POST requests on path /api/movies

Parametised URL. Accepts :id route argument. Access using req.id