Introduction to Node.js

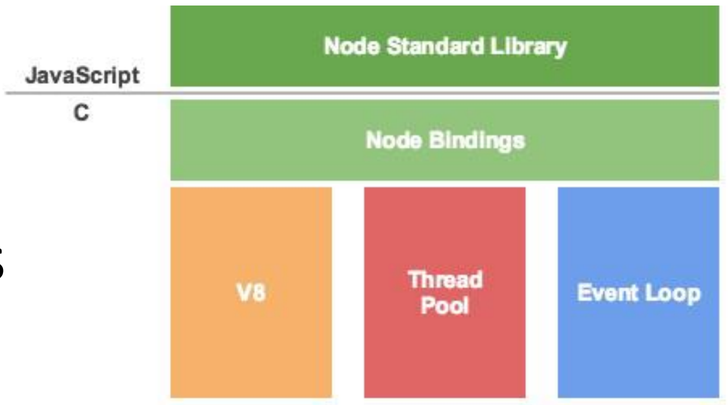
Frank Walsh
Diarmuid O'Connor

Agenda

- What is node.js
- Non Blocking and Blocking
- Event-based processes
- Callbacks in node
- Node Package Manager
- Creating a node app
- Introduction to Express

What's Node: Basics

- Put simply, Node.js is ‘server-side JavaScript’.
- More accurately, Node.js is a high performance network applications framework, well optimized for high concurrent environments.
- In ‘Node.js’ , ‘.js’ doesn’t mean that its solely written in JavaScript. It is
  - 40% JS and 60% C++.
- From the official site: ‘Node's goal is to provide an easy way to build scalable network programs.



What’s Node, V8.

- Embeddable C++ component
- Can expose C++ objects to Javascript
- Very fast and multi-platform
- Find out a bit about it's history here: http://www.google.com/googlebooks/chrome/big_12.html

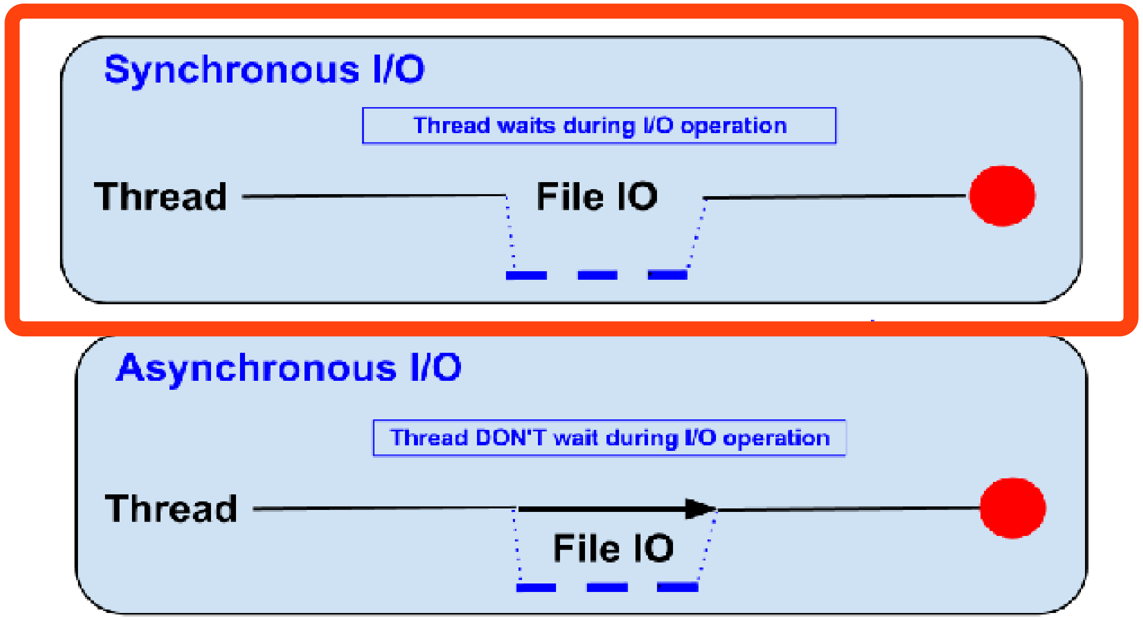
V8 JavaScript Engine

What's Node.js: Event-based

- Generally, input/output (io) is slow.
  - Reading/writing to data store, probably across a network.
- Calculations in cpu are fast.
  - 2+2=4
- Most time in programs spent waiting for io to complete.
  - In applications with lots of concurrent users (e.g. web servers), you can't stop everything and wait for io to complete.
- Solutions to deal with this are:
  - Blocking code with multiple threads of execution (e.g. Apache, IIS)
  - Non-blocking, event-based code in single thread (e.g. NGINX, Node.js)

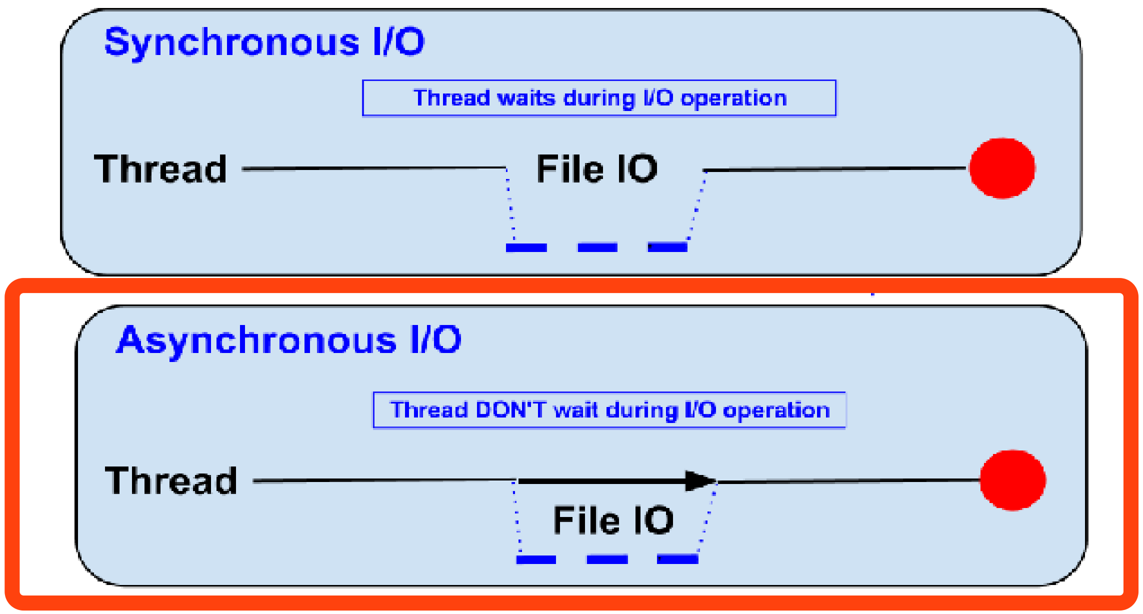
Blocking (Traditional)

- Traditional code waits for input before proceeding (Synchronous)
- The thread on a server "blocks" on io and resumes when it returns.


Synchronous I/O
Thread waits during I/O operation
Thread
File IO
Asynchronous I/O
Thread DON'T wait during I/O operation
Thread
File IO

Non-blocking (Node)

- Node.js code runs in a Non-blocking, event-based Javascript thread
  - No overhead associated with threads
  - Good for high concurrency (i.e. lots of client requests at the same time)


Synchronous I/O
Thread waits during I/O operation
Thread
File IO
Asynchronous I/O
Thread DON'T wait during I/O operation
Thread
File IO

Blocking/Non-blocking Example

Blocking

- Read from file and set equal to contents
- Print Contents
- Do Something Else...

Non-blocking

- Read from File
  - Whenever read is complete, print contents
- Do Something Else...

Blocking/Non-blocking Example

Blocking

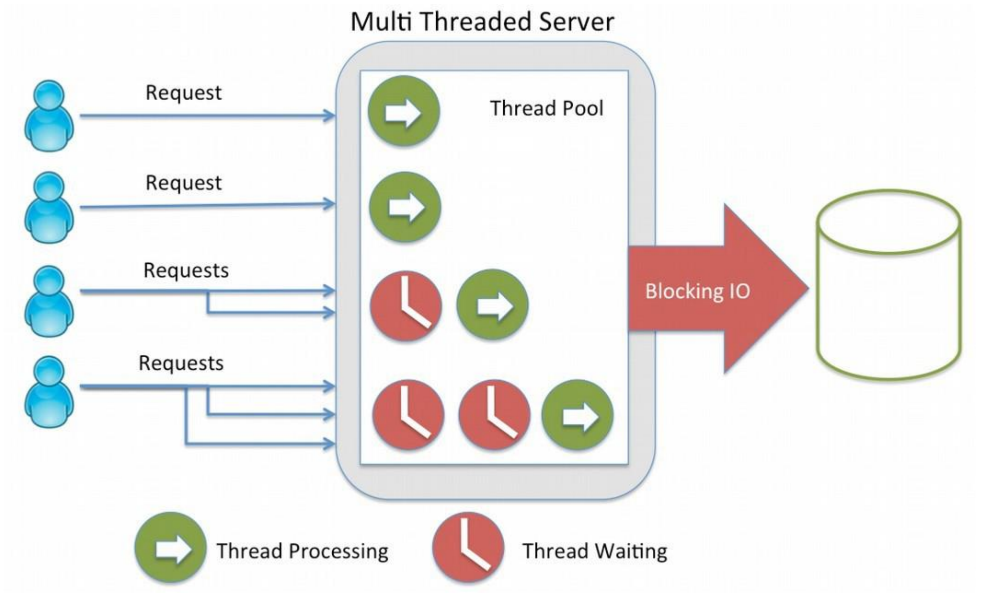
```
const contents = fs.readFileSync('./text.txt');
console.log(contents);
console.log('Doing something else');
```

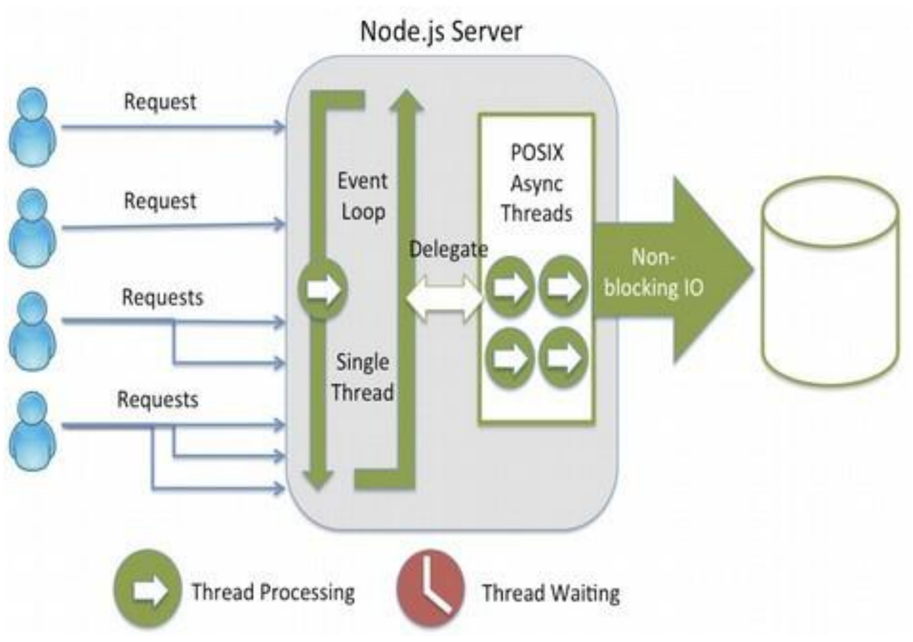
Non-blocking

```
fs.readFile('/etc/hosts', function(err, contents) { console.log(contents); });
console.log('Doing something else');
```

Blocking vs. Non-blocking

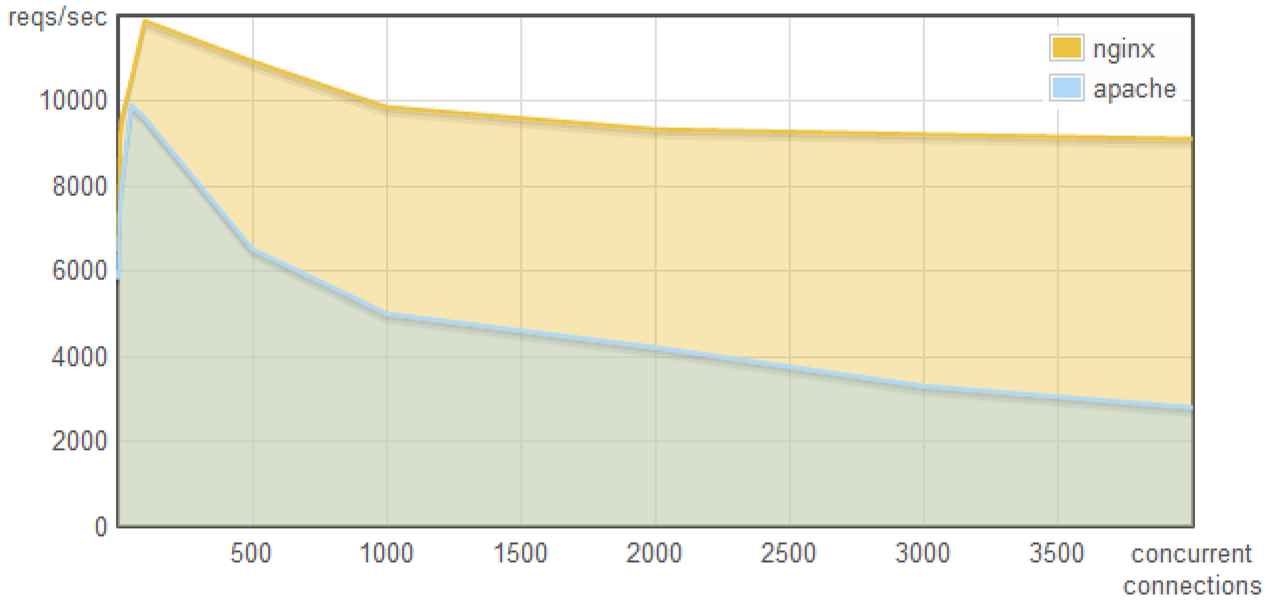
- Threads consume resources
  - Memory on stack
  - Processing time for context switching etc.
- No thread management on single threaded apps
  - Just execute “callbacks” when event occurs
  - Callbacks are usually in the form of anonamous functions.

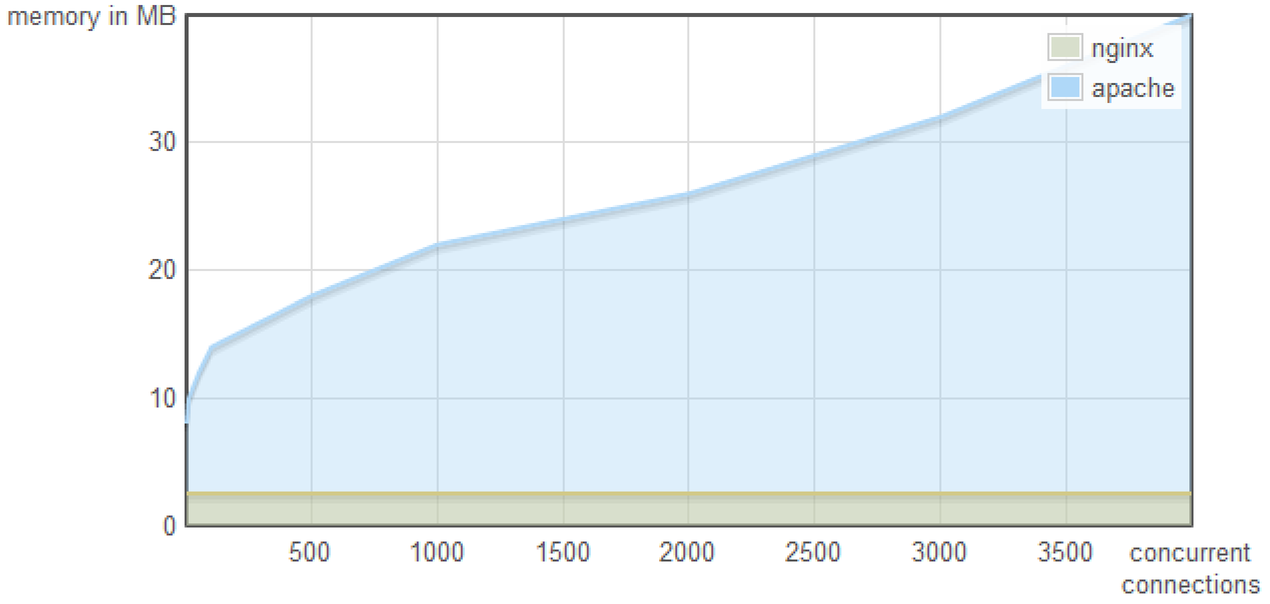




Why does it matter...

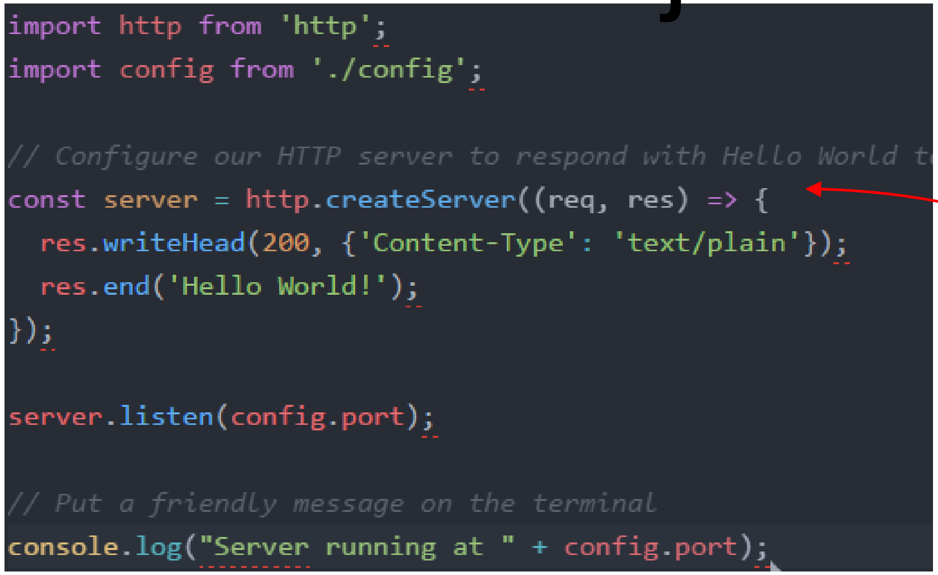
- This is why:





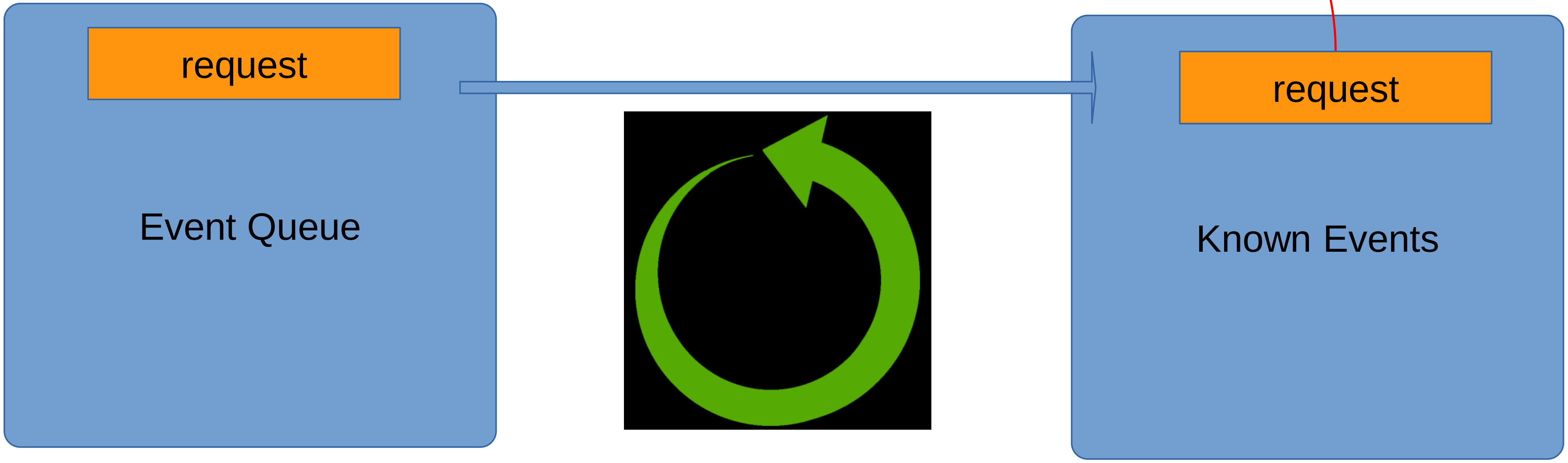
http://blog.webfaction.com/a-little-holiday-present

Node.js Event Loop

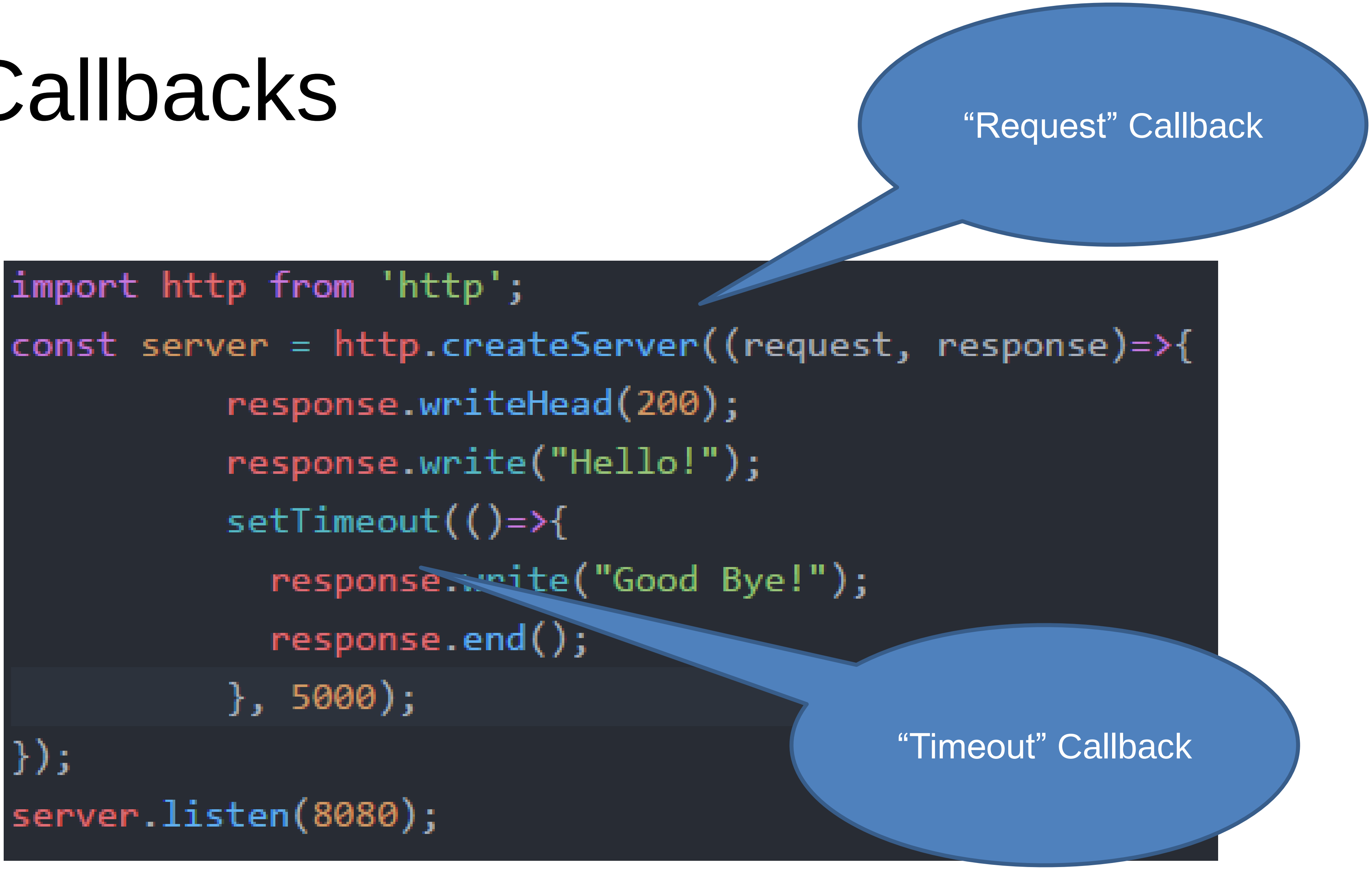


```
import http from 'http';
import config from './config';

// Configure our HTTP server to respond with Hello World t
const server = http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World!');
});

server.listen(config.port);

// Put a friendly message on the terminal
console.log("Server running at " + config.port);
```

EVENT LOOP STARTS WHEN FINISHED

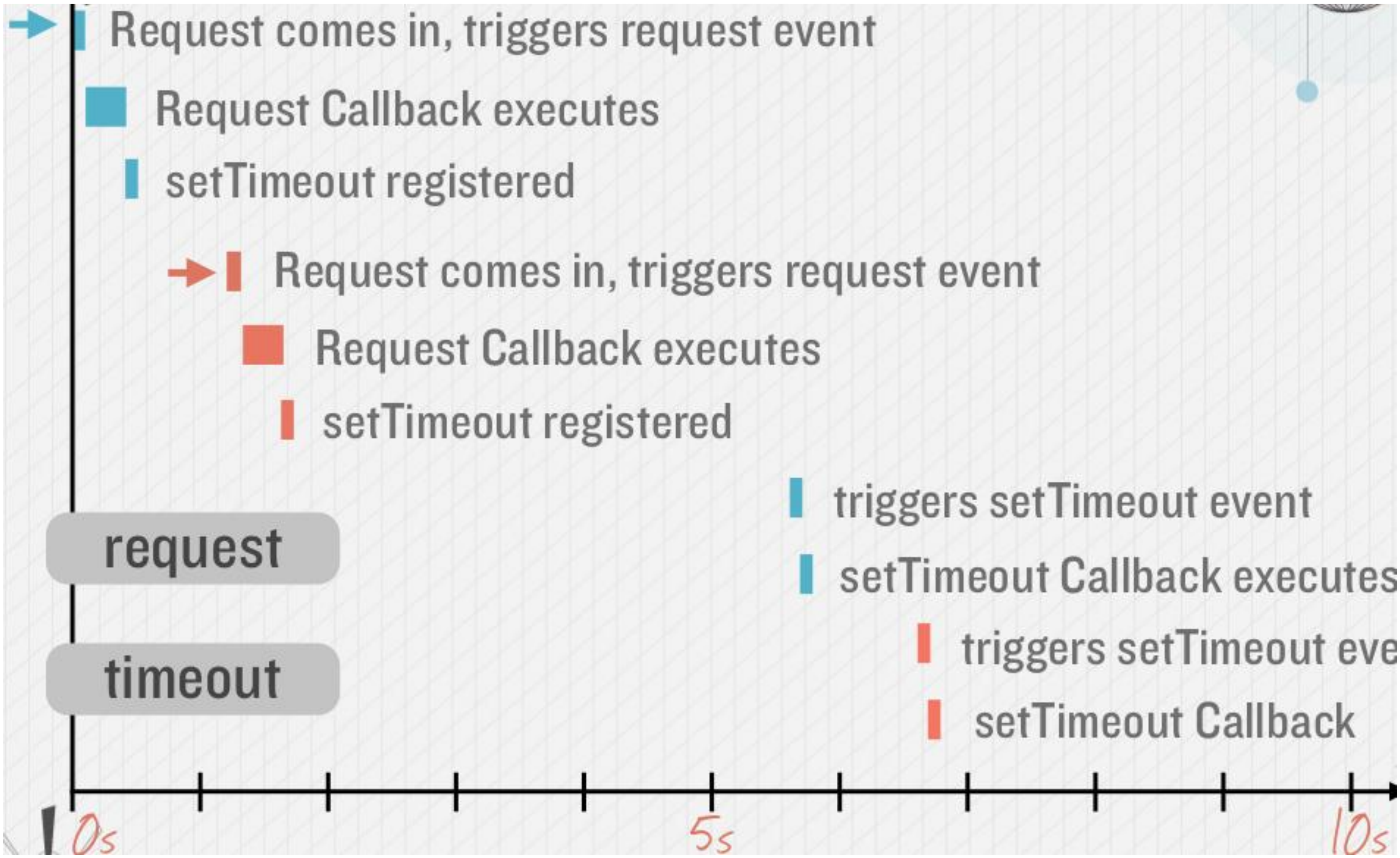


Callbacks

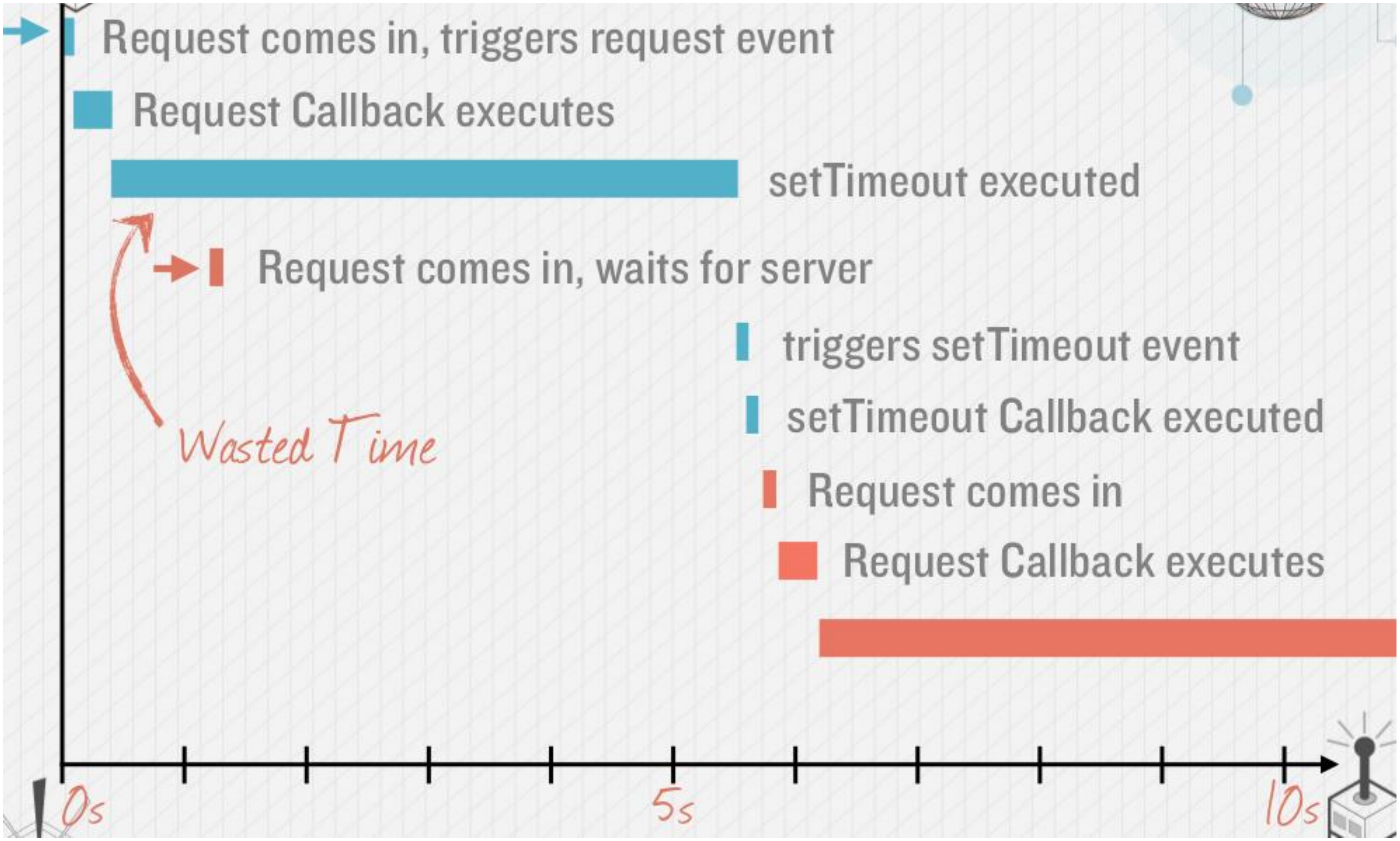


```
import http from 'http';
const server = http.createServer((request, response)=>{
          response.writeHead(200);
          response.write("Hello!");
          setTimeout(()=>{
            response.write("Good Bye!");
            response.end();
          }, 5000);
});
server.listen(8080);
```

Callback Timeline, Non Blocking

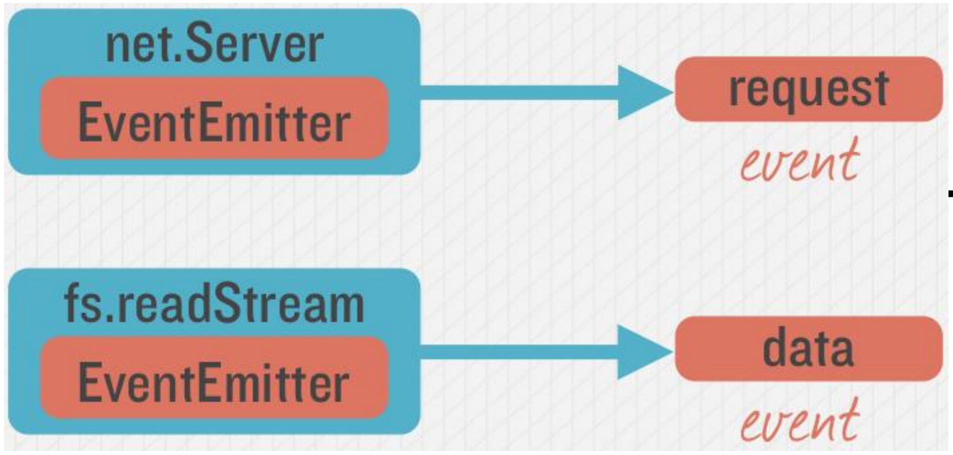

Request comes in, triggers request event
Request Callback executes
setTimeout registered
Request comes in, triggers request event
Request Callback executes
setTimeout registered
triggers setTimeout event
setTimeout Callback executes
triggers setTimeout eve
setTimeout Callback
request
timeout
0s
5s
10s

Callback Timeline, Blocking

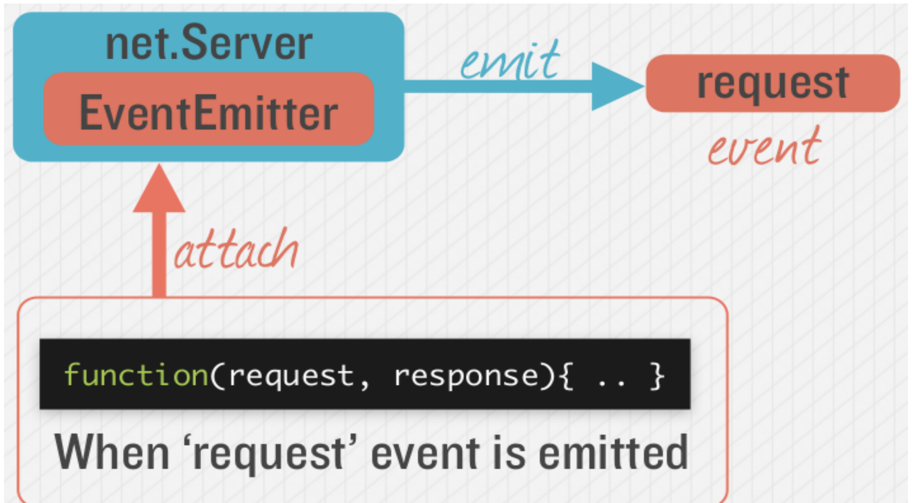

Request comes in, triggers request event
Request Callback executes
setTimeout executed
Request comes in, waits for server
triggers setTimeout event
setTimeout Callback executed
Request comes in
Request Callback executes
Wasted Time
0s
5s
10s

Emitting Event in Node

- Many objects can emit events in node.



- See here ITTP Server works



```
function(request, response){ .. }
```

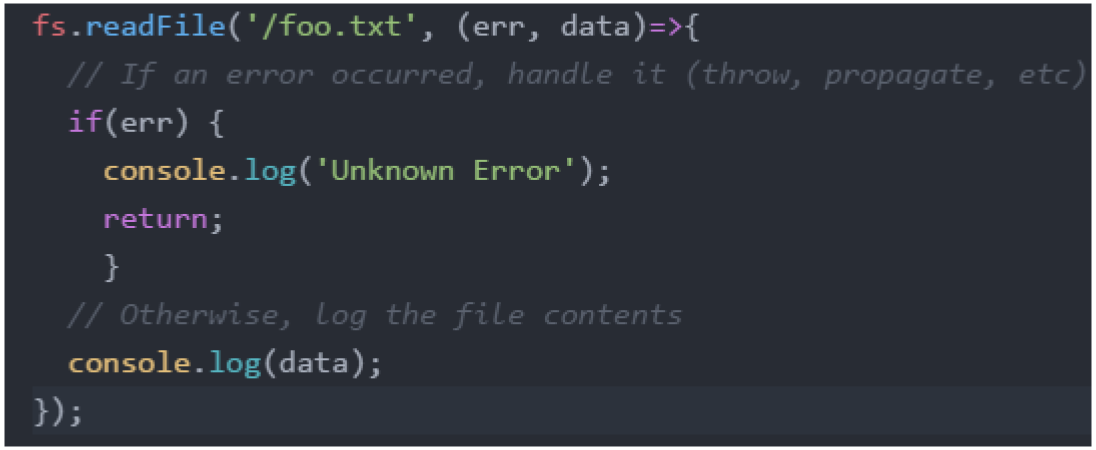
When 'request' event is emitted

Node Callbacks

- “If Google’s V8 Engine is the heart of your Node.js application, then callbacks are its veins”.
- They enable a balanced, non-blocking flow of asynchronous control across modules and applications.
  - But for callbacks to work at scale you need a common, reliable protocol.
  - The “error-first” callback (also known as an “errorback”, “errback”, or “node-style callback”) was introduced to solve this problem, and is a standard for Node.js callbacks.
- A callback is basically a function called at the completion of a given task. This prevents any blocking, and allows other code to be run in the meantime.

Error First Callbacks

- The first argument of the callback is reserved for an error object. If an error occurred, it will be returned by the first err argument.
- The second argument of the callback is reserved for any successful response data. If no error occurred, err will be set to null and any successful data will be returned in the second argument.



```
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: Caution!

- In node, event callback functions should execute fast
  - Do not perform computationally expensive operations in the event callback function.
- Always use callback functions. Better for high concurrency.

Node Modules

Node Modules

- Node has a small core API
- Most applications depend on third party modules
- Thisd 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 write 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

- Installing a NPM Module is easy:
- Navigate to the application folder and run:

  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:
  - var express = require('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 globaly 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*

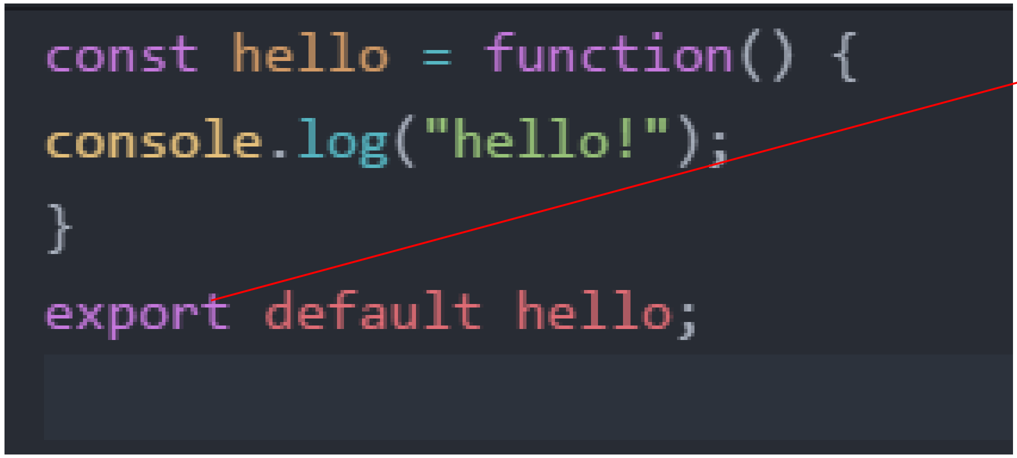
– **npm install** *install packages listed in package.json*

– **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 = function() {
console.log("hello!");
}
export default hello;
```

Export defines what import returns

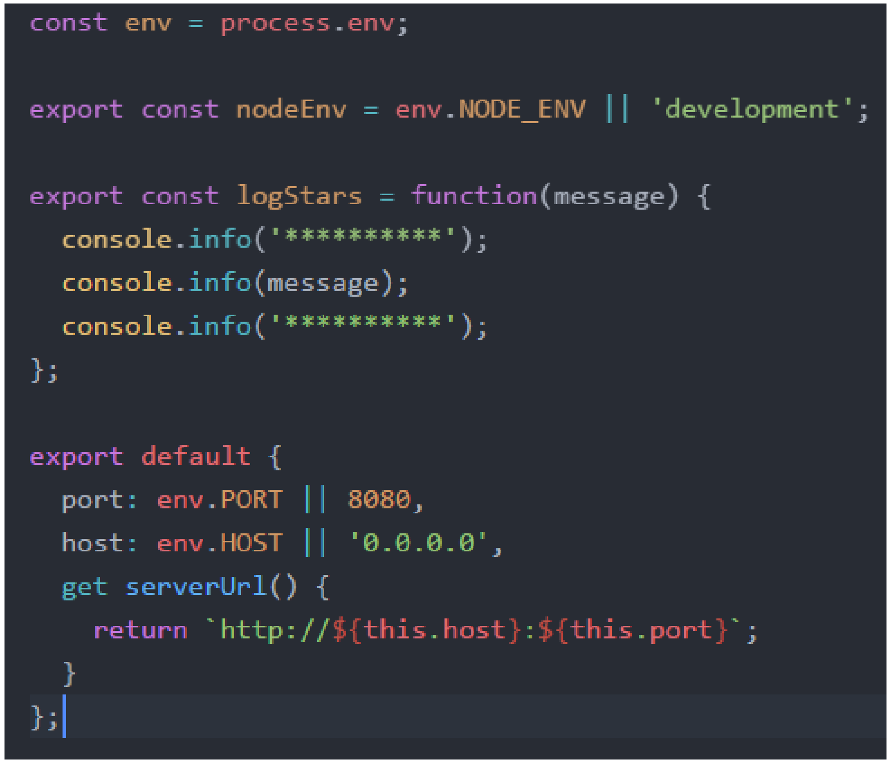
- To access in our application, **app.js:**

```
import hello from './custom_hello';
hello();
```

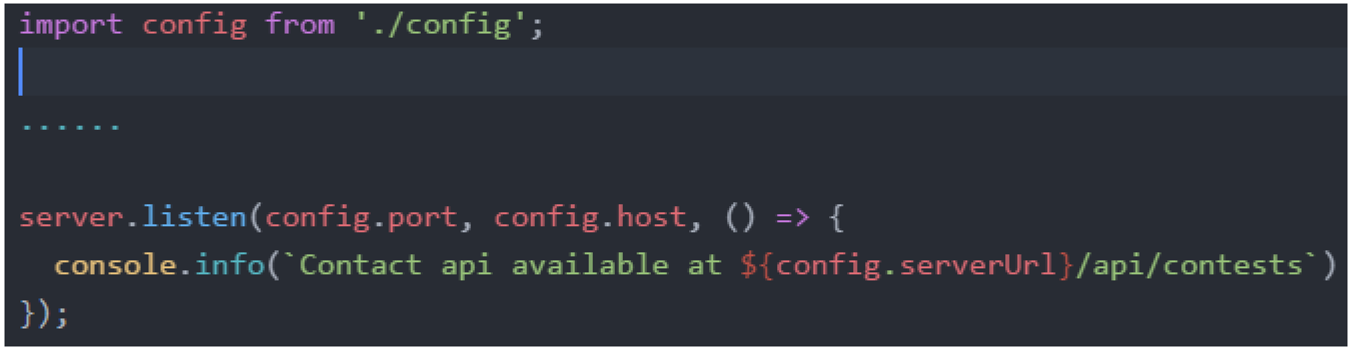
Creating your own Node Modules

- Exporting Multiple Properties

- Accessing in other scripts



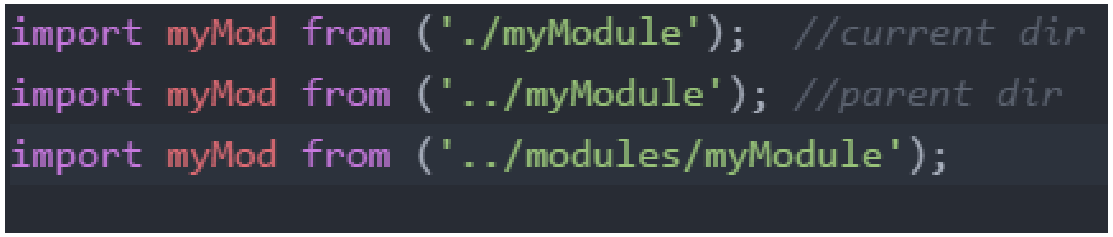
```
const env = process.env;

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}`;
  }
};
```



```
import config from './config';

......

server.listen(config.port, config.host, () => {
  console.info(`Contact api available at ${config.serverUrl}/api/contests`)
});
```

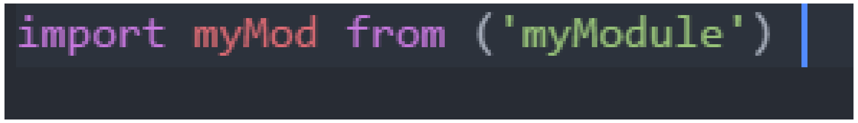
The import search

- 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')
```

The Express Package

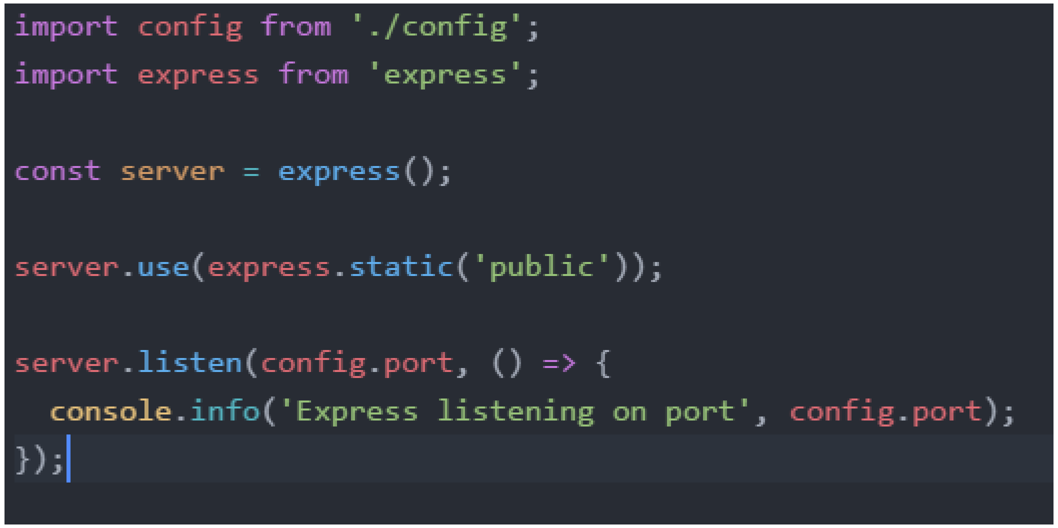
What is Express?

- Web application framework for Node
  - Built on the Connect middleware package
  - It's popular because it‘s
    - Minimalist,
    - Fast
    - Simple

What Express Gives Us...

- Parses arguments and headers
- Easy Routing
  - Route a URL to a Javascript callback function
- Views
  - Partials
  - Layouts
- Environment-based Configuration
- Sessions
- File Uploads

Simple Express App (server.js)



```
import config from './config';
import express from 'express';

const server = express();

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

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

Loads Express module

Instantiates Express server

Define static content for HTTP GET

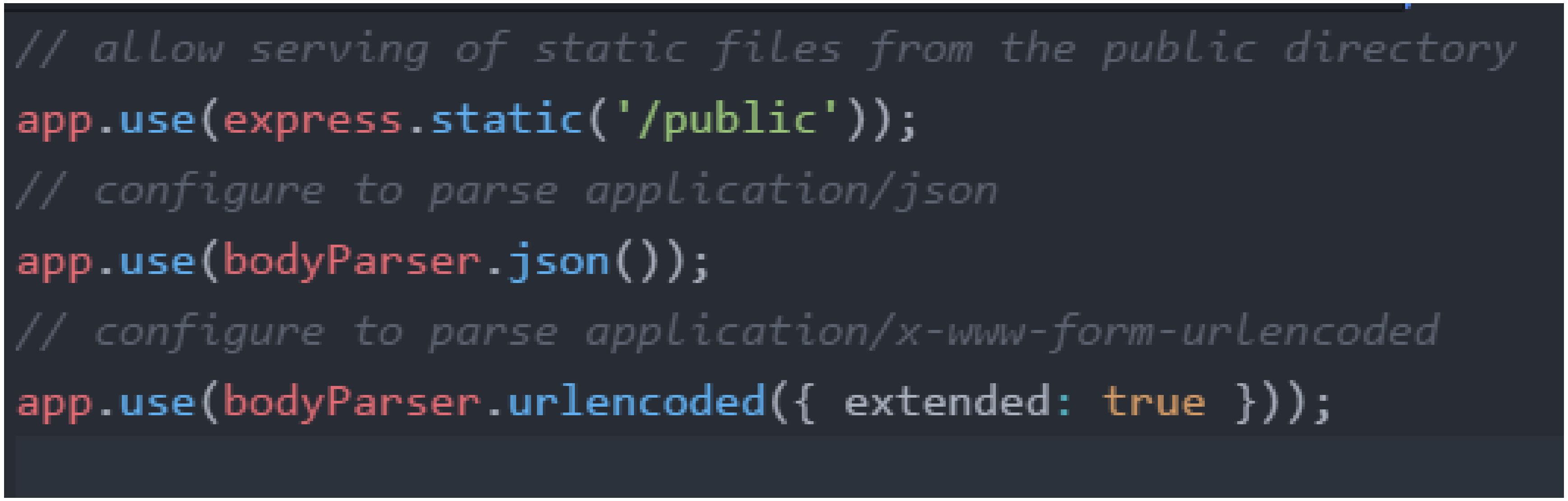
Getting Started with Express

- Installing Express

```
[local install] C:\> npm install express
[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 }));
```

Routing Examples

```
//Catch-all
app.all('/app(/*)?', requiresLogin);

// Routes
app.get('/', routes.index);
app.get('/about', routes.about);
app.get('/contact', routes.contact);
app.get('/app/list', routes.listapps);
app.get('/app/new', routes.newapp);
app.post('/app/new', routes.saveapp);
app.get('/app/:app', routes.getapp);
app.get('/app/:app/edit', routes.editapp);
```

Catch-all – works for all HTTP verbs

HTTP GET request

HTTP POST request

Accepts :app route argument

Syntax follows the pattern:

App.[verb](path, (req,res)=>{});

Node Applications Structure

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 server 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 static content

Basic Project Structure

```
/projectroot/          → Root of your actual application
      package.json     → Tells Node and NPM what packages are required
      readme.md
      server.js        → The main entry point for the Express application
      public/
            /images
            /stylesheets
            /scripts
            index.html
      node_modules/
      api/
```

Static content

Output directory for all NPM installations