Introduction to Node.js

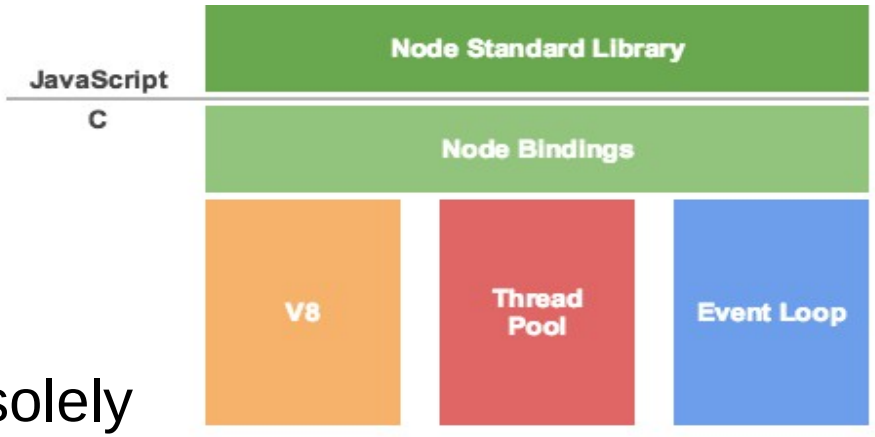
Frank Walsh
Diarmuid O'Connor

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 engine

- Embeddable C++ component
- Can expose C++ objects to Javascript
- Very fast and multi-platform
- Find out a bit about it's history here:

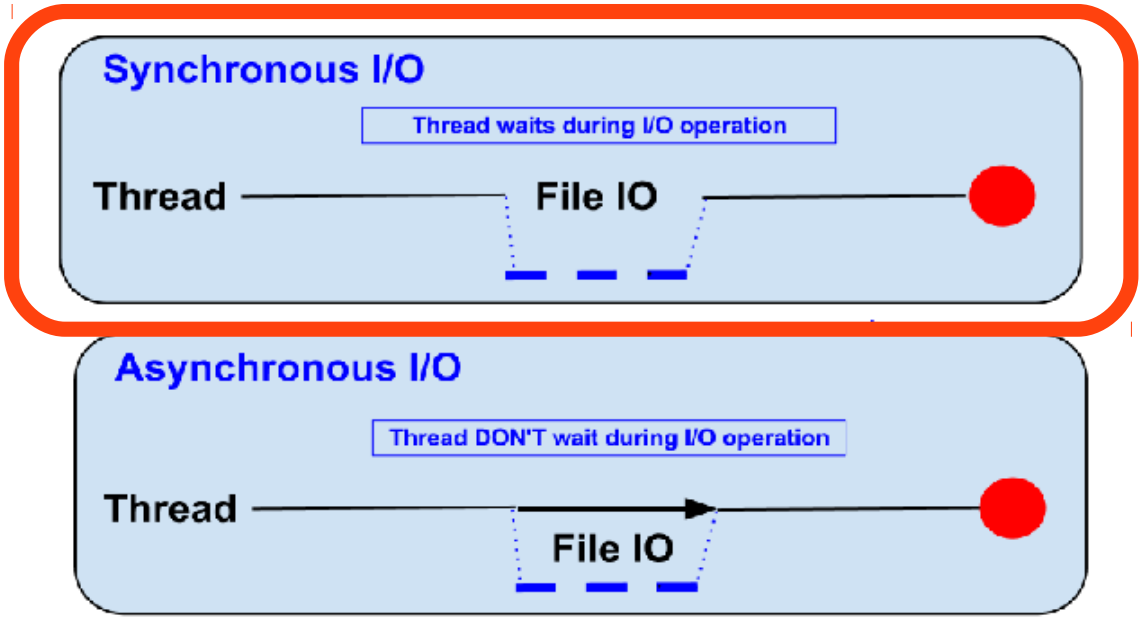
  http://www.google.com/googlebooks/chrome/big_12.html

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)

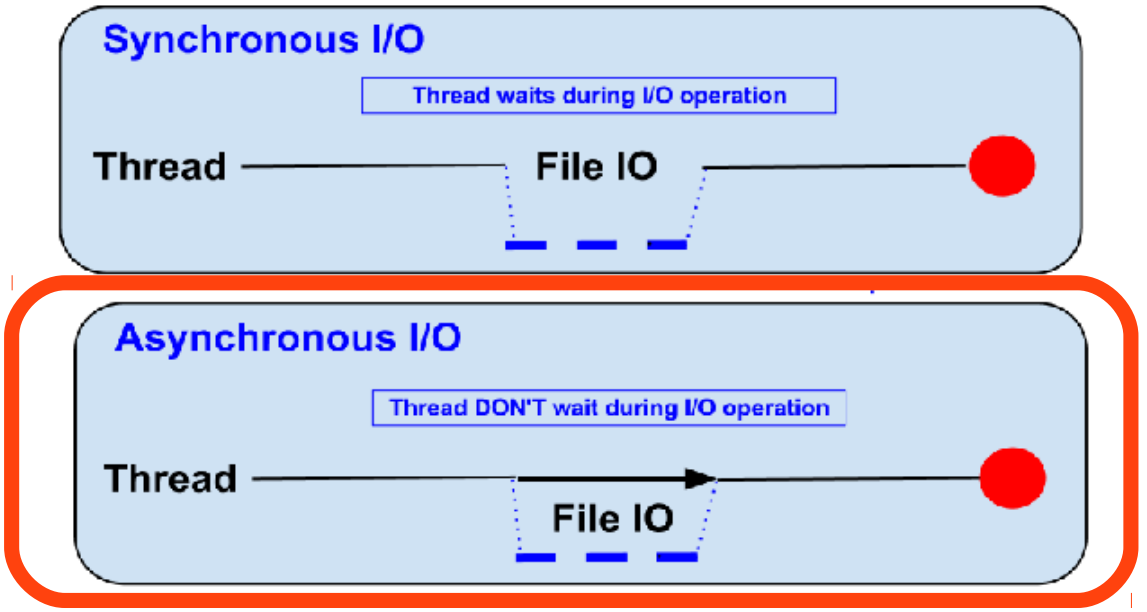
Blocking (Traditional)

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



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)



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

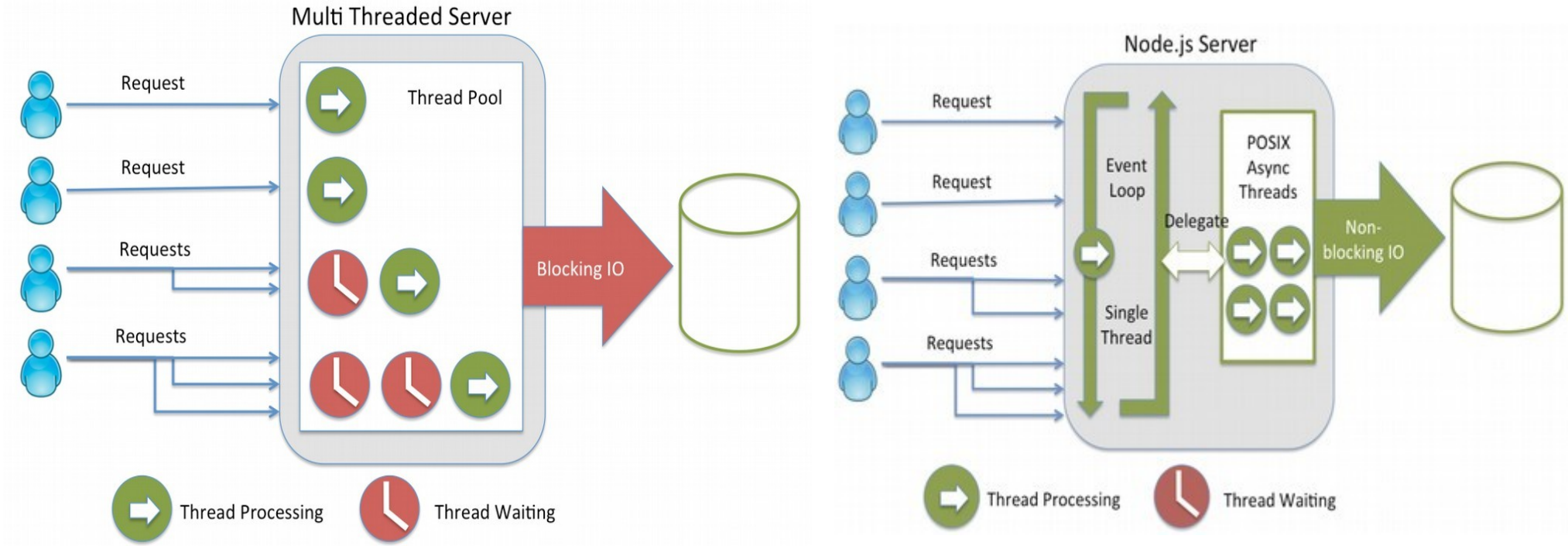
```
var contents = fs.readFileSync('/etc/hosts');
console.log(contents);
console.log('Doing something else');
```

Non-blocking

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

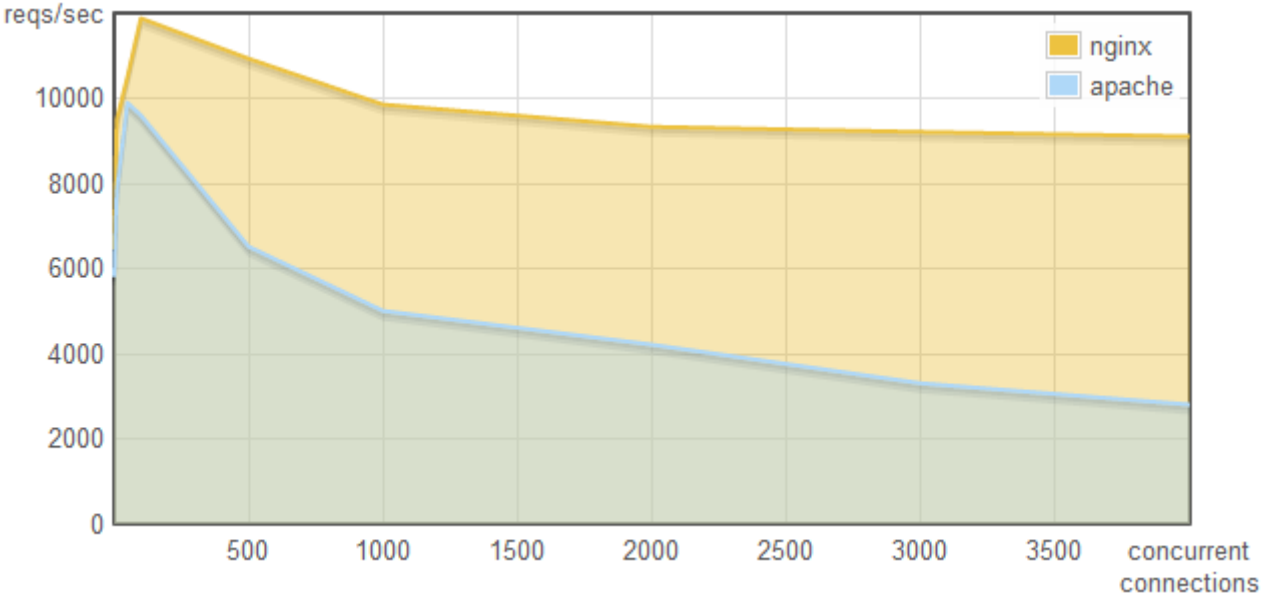
Blocking vs. Non-blocking

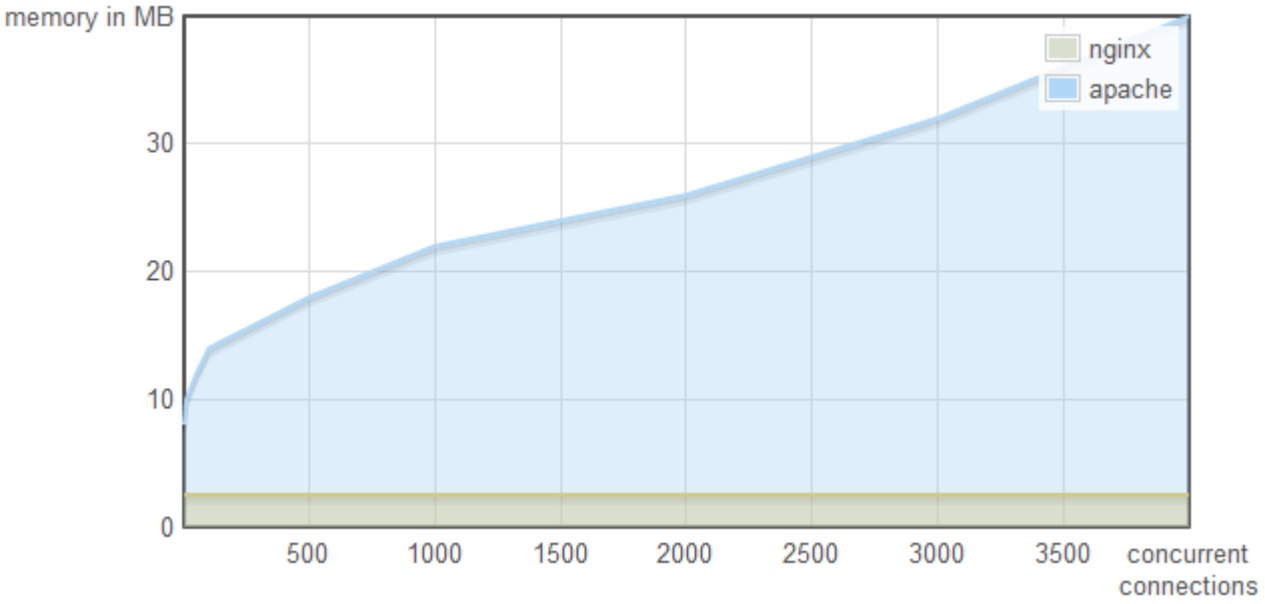
- Threads consume resources
  - Memory on stack
  - Processing time for context switching etc.
- No thread management on single threaded apps
  - Just execute “callbacks” when event occurs
  - Callbacks are usually in the form of anonamous functions.



Why does it matter...

- This is why:

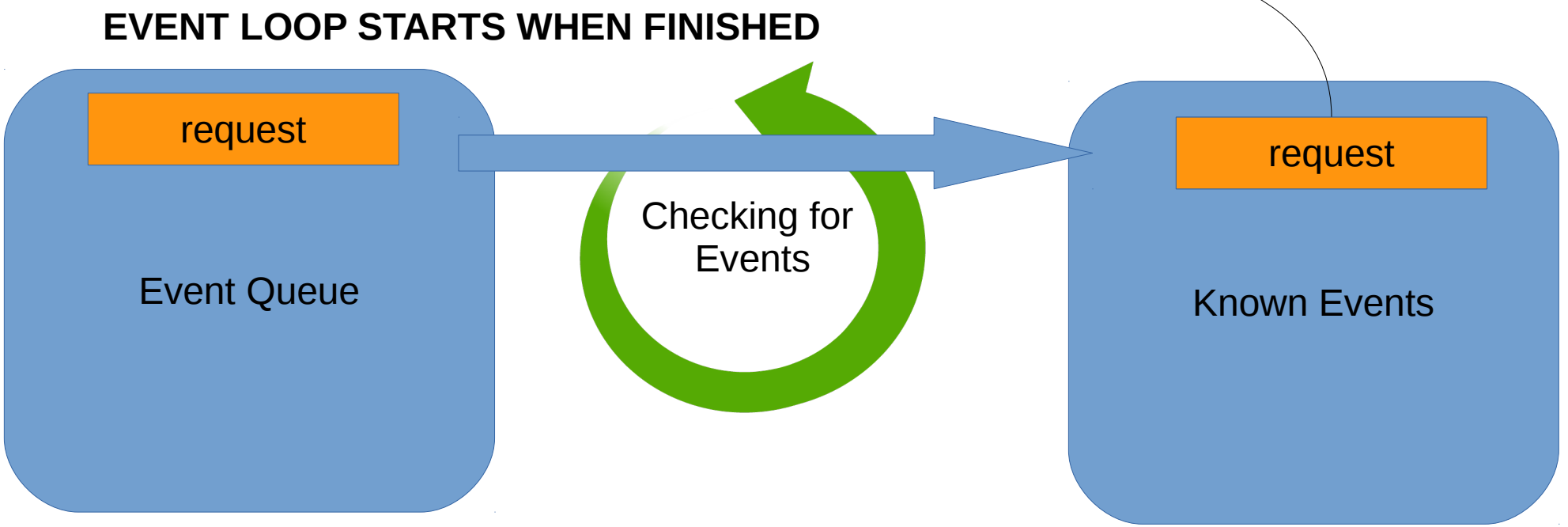




http://blog.webfaction.com/a-little-holiday-present

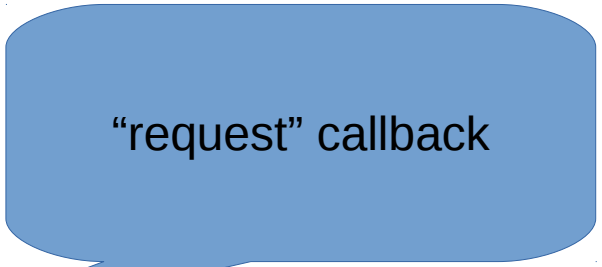
Node.js Event Loop

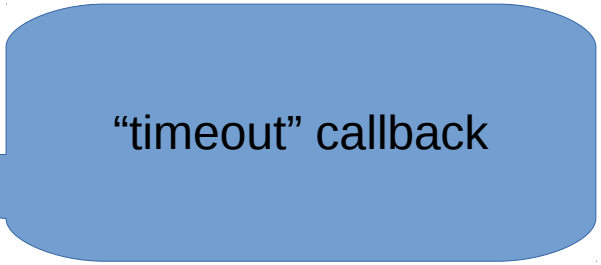
```
var http = require('http');
var server = http.createServer(function (request, response) {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.end("Hello World\n");
});
server.listen(8080);
console.log("Server running at http://127.0.0.1:8080/");
```



Node.js Callbacks

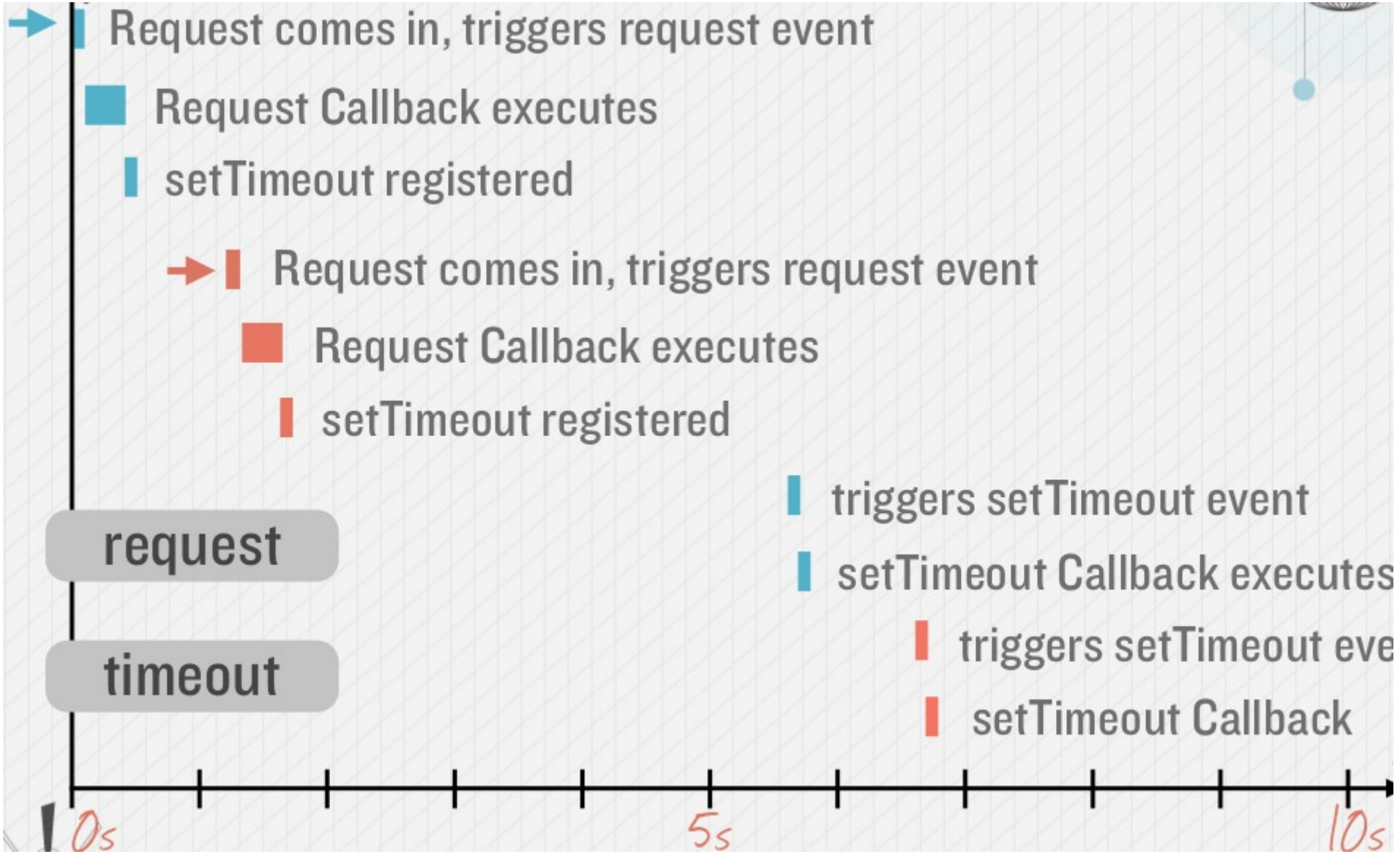
Example of 2 callbacks



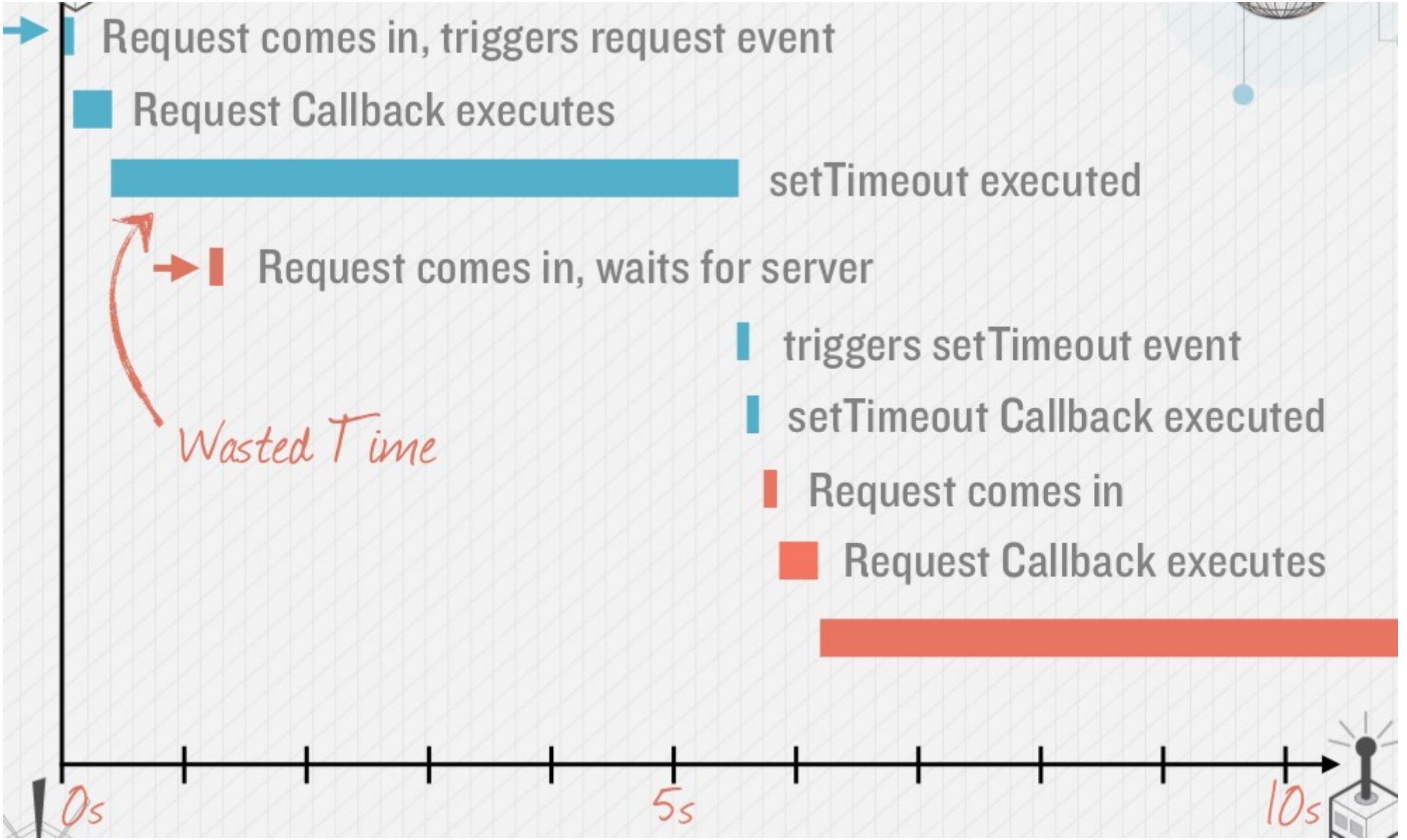



```
var http = require('http');
var server = http.createServer(function(request, response) {
   response.writeHead(200);
   response.write("Hello!");
   setTimeout(function(){
        response.write("Good Bye!");
        response.end();
    }, 5000);
});
server.listen(8080);
```

Callback Timeline, Non-Blocking

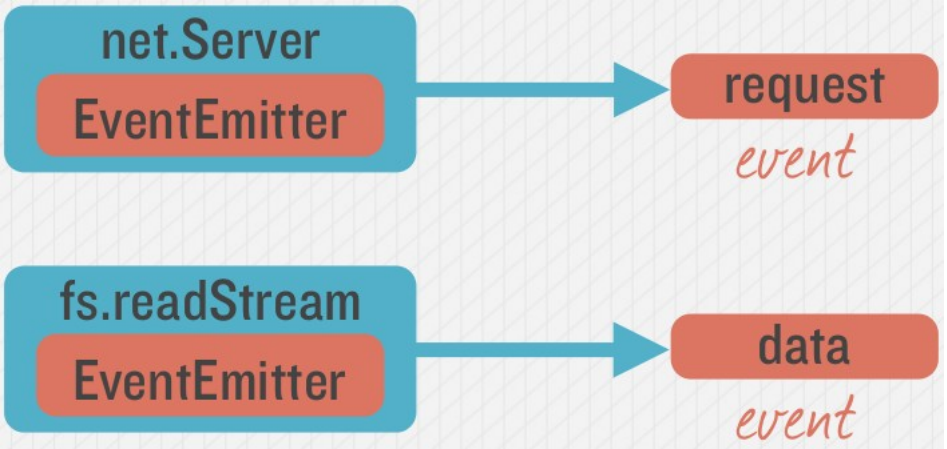


Callback Timeline, Blocking

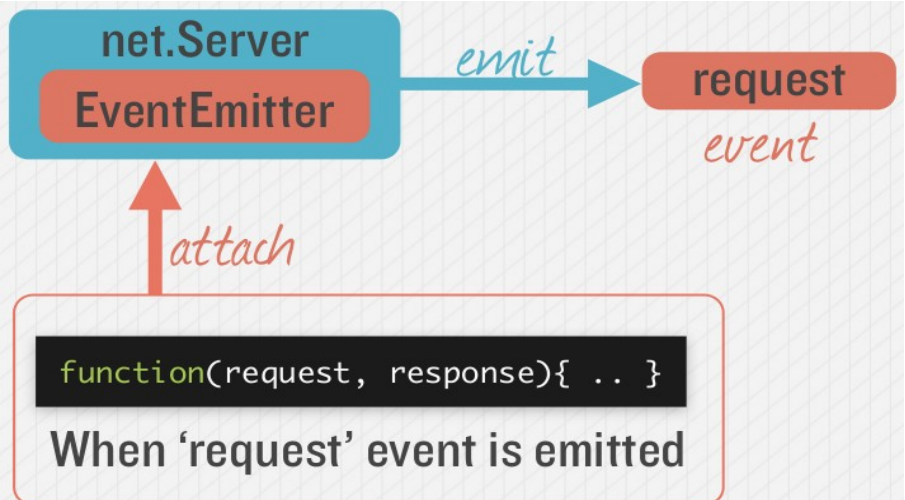


Emitting Event in Node

- Many objects can emit events in node.



- See here for a description of how HTTP Server works



Node:Caution

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

Node: Callbacks

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 has since become the 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.

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 has since become the 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.

Defining 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', function(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

Node Modules

- Node has a small core API
- Most applications depend on 3rd party modules
- 3rd 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.

Node Modules

- Installing a NPM Module is easy:
- Libraries in Node.js are called packages and they can be installed by typing:

```
npm install express
```

- This installs into a “**node_module**” folder in the current folder.
- To use the module in your code, use:

```
var express = require('express');
```

- This loads express from local **node_modules** folder.

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*

Global Node Modules

- Sometimes you may want to access modules from the shell/command line.
- You can install modules that will execute glovbaly 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.

Creating your own Node Modules

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

```
var hello = function() {
    console.log("hello!");
}
exports = hello;
```

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

```
var hello = require('./custom_hello');
hello();
```

Creating your own Node Modules

- Another example **custom_goodbye.js:**

```
exports.goodbye = function() {
    console.log("Bye!");
}
```

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

```
var gb = require('./custom_goodbye');

gb.goodbye();
```

Export defines what require returns

Creating your own Node Modules

- Exporting Multiple Functions, **my_Module.js:**

```
exports.hello = function() {
    console.log("Hello!");
}
exports.goodbye = function() {
    console.log("Bye!");
}
```

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

```
var myMod = require('./my_Module.js');
myMod.hello();
myMod.goodbye();
```

Export defines what require returns

The require search

- Require searches for modules based on path specified:

```
var myMod = require('./myModule') //current dir
var myMod = require('../myModule') //parent dir
var myMod = require('../modules/myModule')
```

- Just providing the module name will search in node_modules folder

```
var myMod = require('myModule')
```

The Express Package

What is Express?

- Web application framework for
  - Based on “Sinatra” is a DSL for quickly creating web applications in Ruby with minimal effort. http://www.sinatrarb.com/intro.html
- 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)

```
var express = require('express');
var app = express()


// allow serving of static files from the public directory
app.use(express.static(__dirname + '/public'));

var server = app.listen(3000, function () {
  var host = server.address().address
  var port = server.address().port
  console.log('Example app listening at http://%s:%s', host, port)
})
```

Loads Express module

Instantiates Express server

Root route for HTTP GET

Listen on on port 3000

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(__dirname +
'/public'));
// configure to parse
application/json
app.use(bodyParser.json());
// configure to parse application/x-
www-form-urlencoded
```

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, function(req,res), [function(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 your

Suggested Project Structure