Introduction to Node.js

Frank Walsh

Agenda

- What is Node.js
- V8 engine
- Non Blockin and Blocking
- Typical Node.js service structure

What's Node.js

- High-performance server-side JavaScript
  - Used to build scalable networked services and applications.
- Uses the Google Chrome V8 just-in-time compilation to Machine code
  - Fast because V8 is mostly C.
- Well designed module system for third party code (i.e. Node Packet Manage, NPM)

What's Node: V8 engine

- Embeddable C++ component
  - in the lab you (may have) needed to install C++
- Can expose C++ objects to Javascript
- Very fast and multi-platform
- Find out a bit about it's history here:

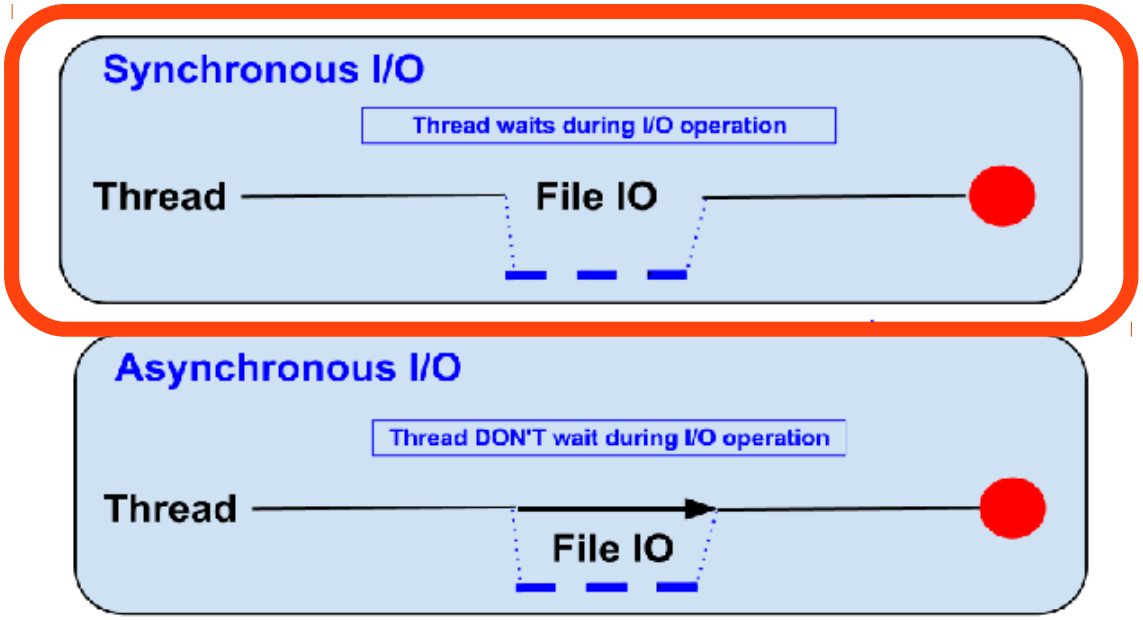
  http://www.google.com/googlebooks/chrome/big_12.html

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 (Apache, IIS)
  - Non-blocking, event-based code in single thread (NGINX, Node.js)

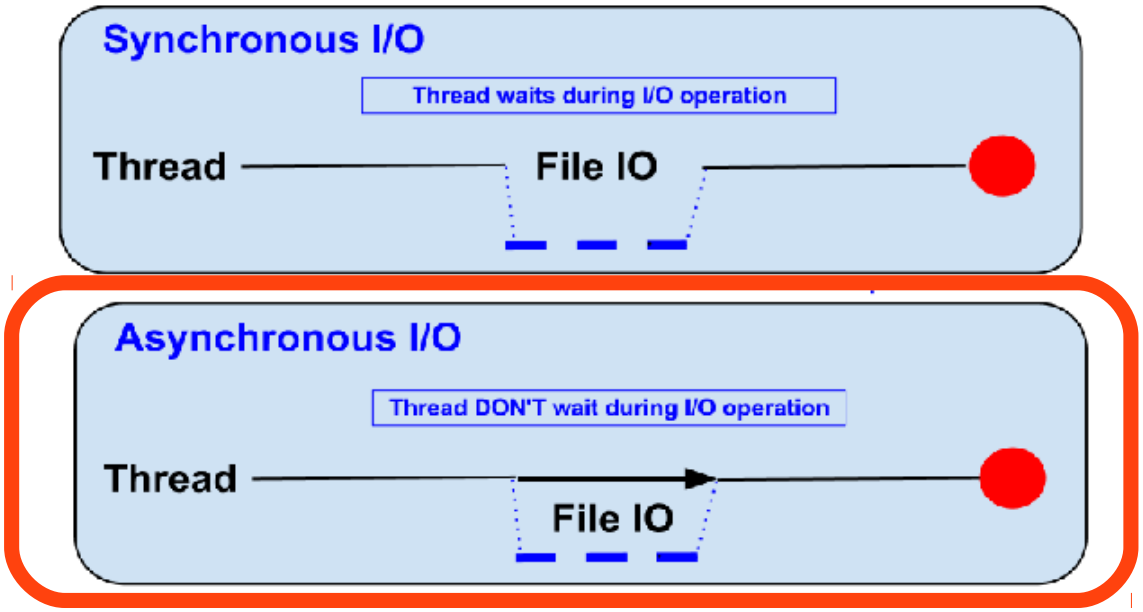
Blocking (Traditional)

- 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

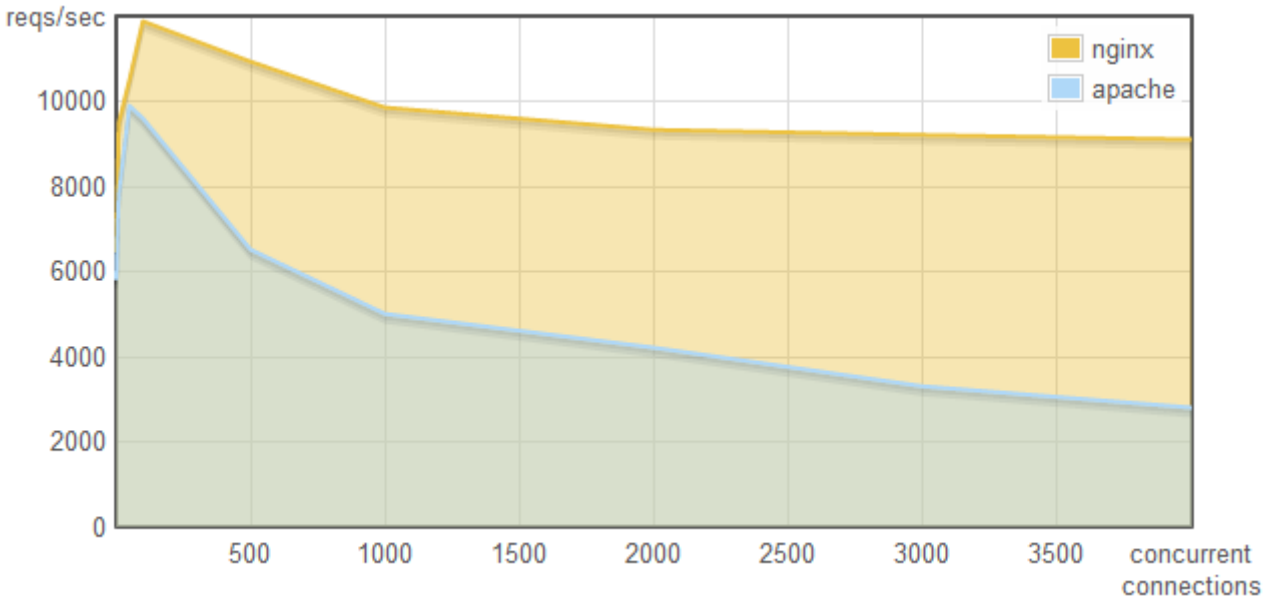
```
fs.readFile('/etc/hosts', function(err, contents) {
console.log(contents);
});
console.log('Doing something else');
```

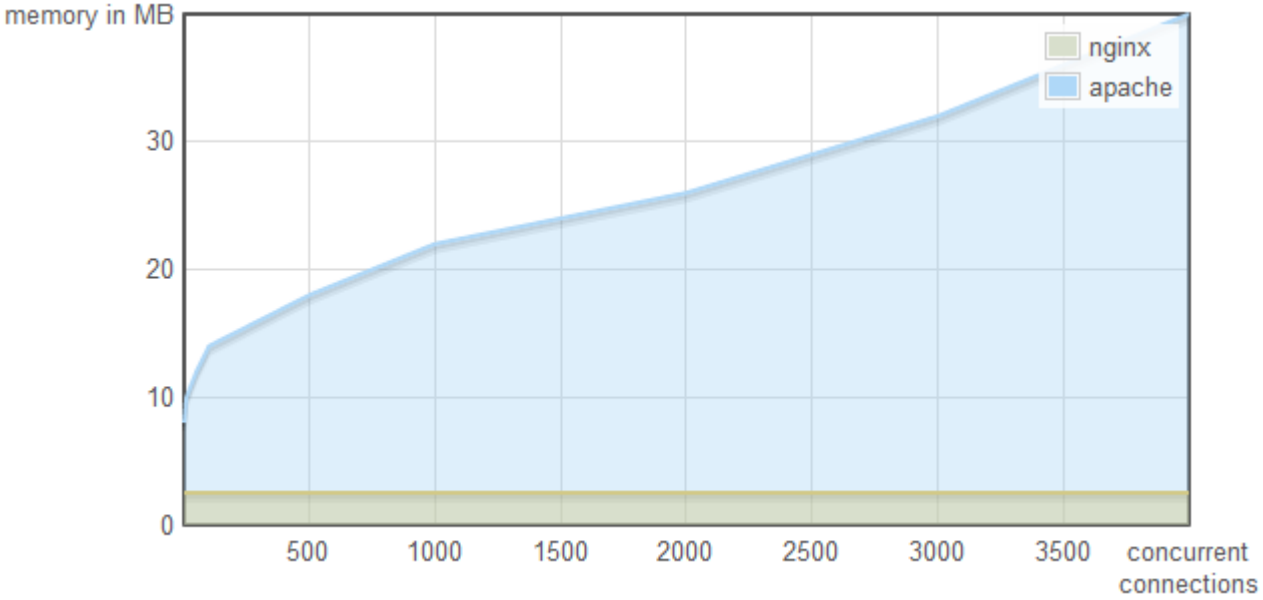
Blocking vs. Non-blocking

- 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:





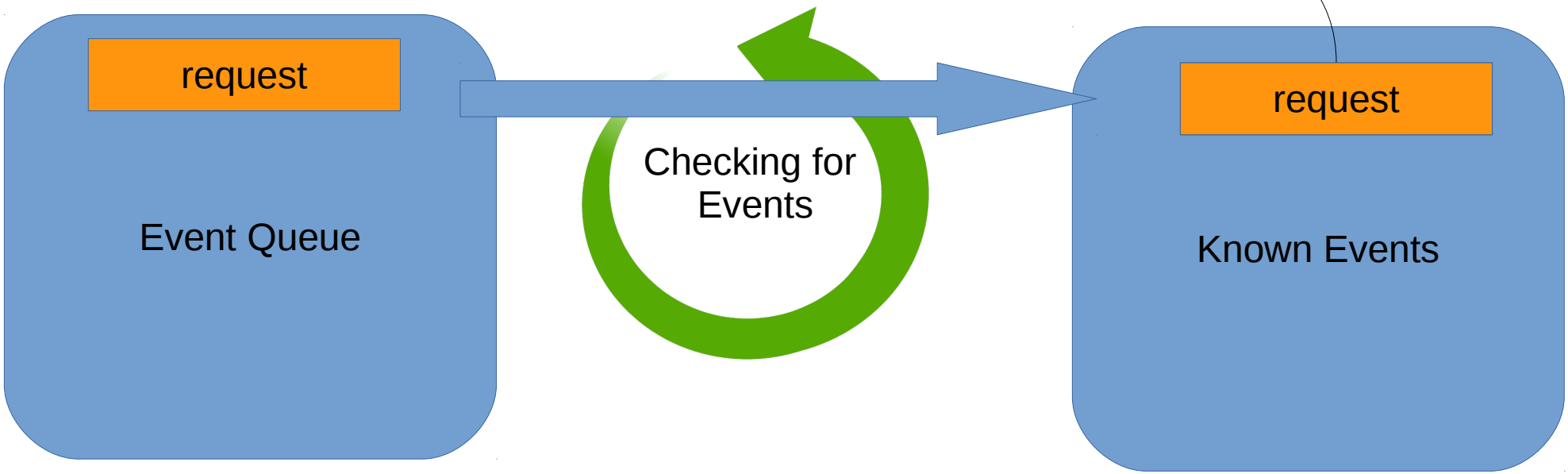
http://blog.webfaction.com/a-little-holiday-present

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/");
```

Callback

EVENT LOOP STARTS WHEN FINISHED



Callbacks

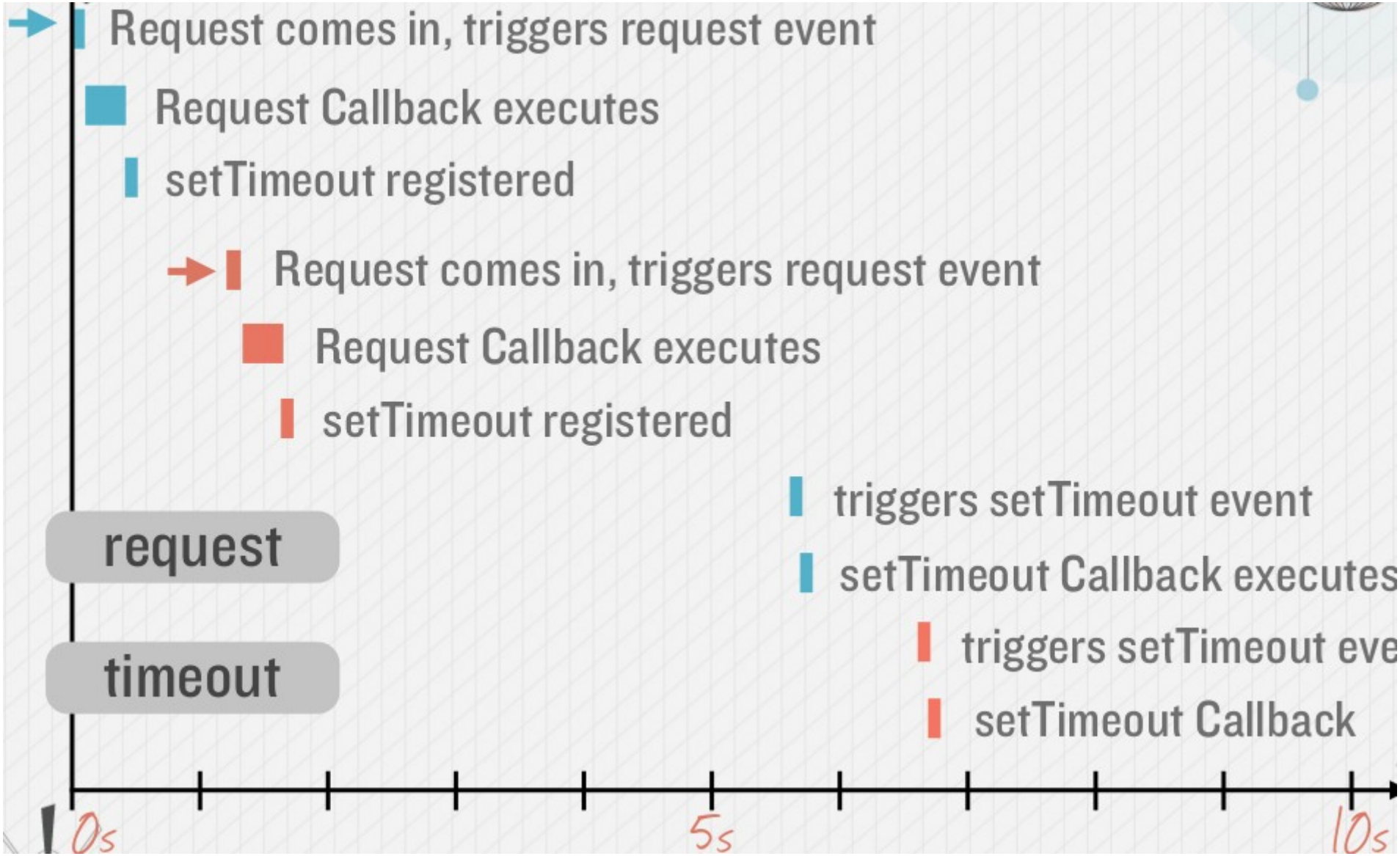
Example of 2 callbacks

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

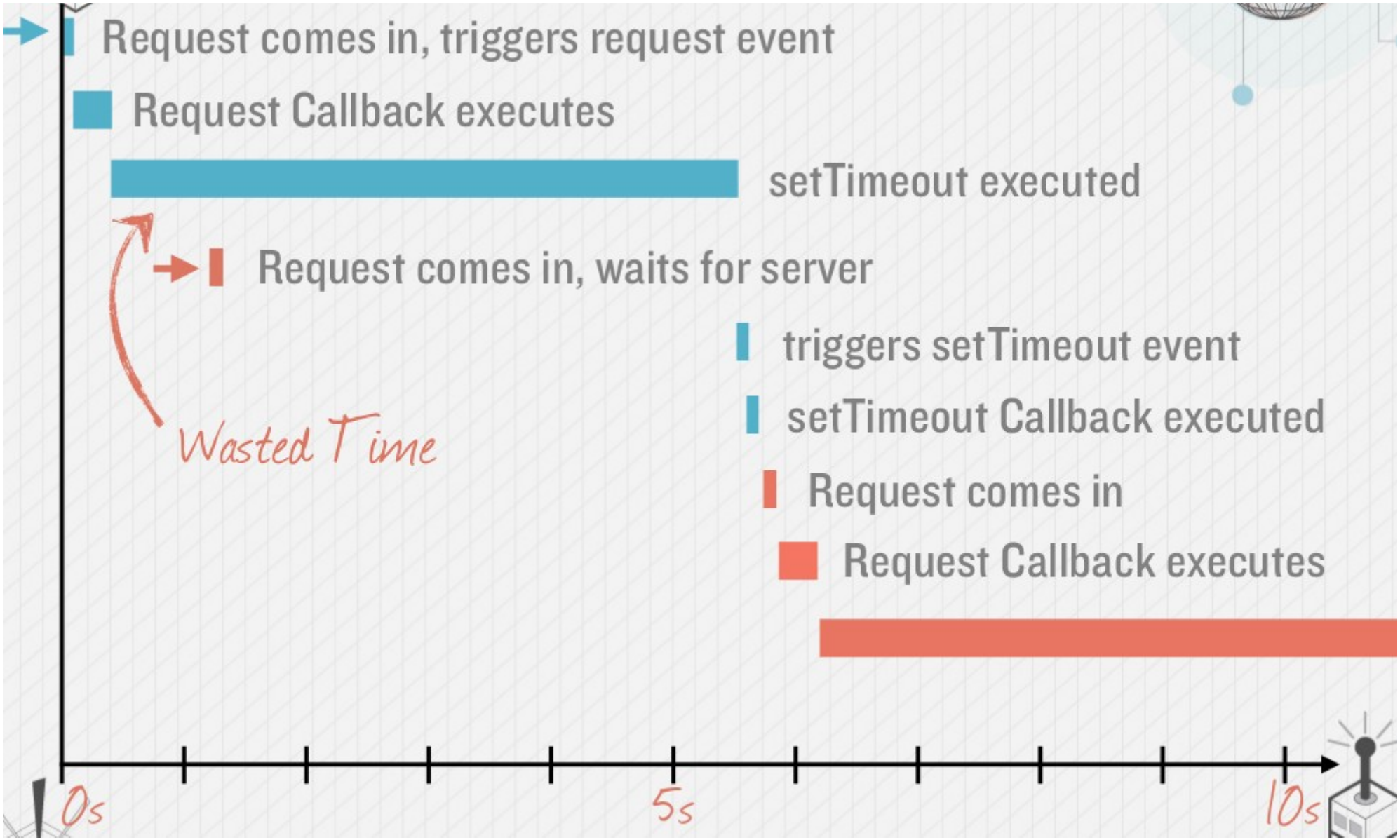
“request” callback

“timeout” callback

Callback Timeline, Non Blocking

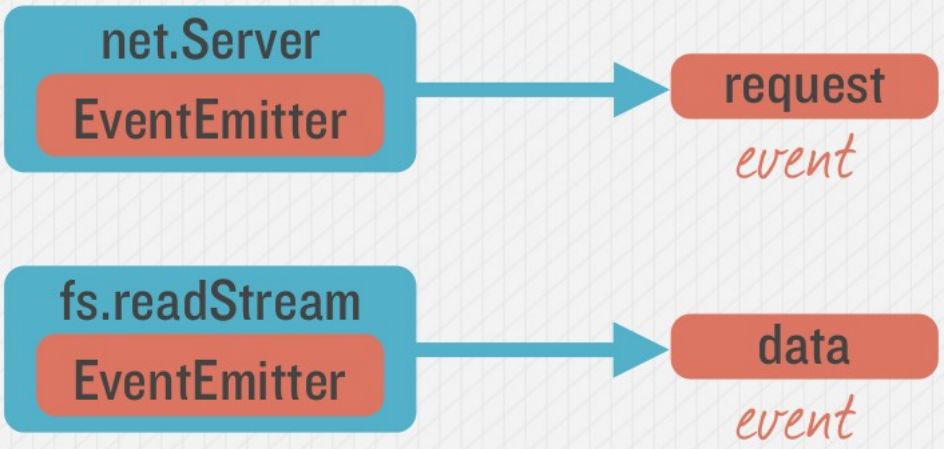


Callback Timeline, Blocking

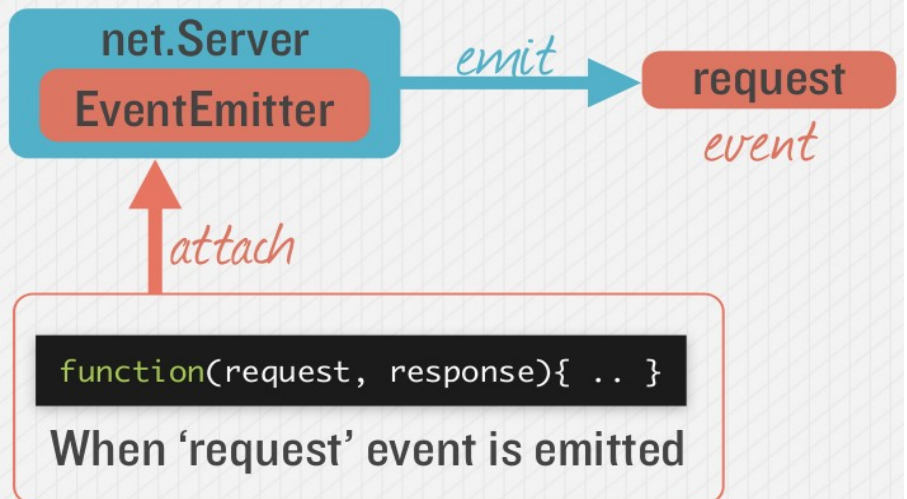


Emitting Event in Node

- Many objects can emit events in node.



- See here for a description of how HTTP Server works



Node 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:
- Navigate to the application folder and run:

```
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.

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;
```

Export defines what require returns

- 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')
```

Node Applications Structure

Structuring Node Services

- Node Server Code needs to be structured
  - Manage code base
  - Keeps code maintainable
- Typical structure for Node.js service
  - common code
  - Main server code
  - Api implementation code
  - Helper code

Example Approach:

- Use a “node” folder as the top level to contain all node.js files
  - Run npm in this folder to ensure just one node_modules folder
  - Use a lib folder within the node folder for your code

-node
--->lib
--->node_modules

common.js

- Can define a “node/lib/common.js” for common code

```
// build-in modules
exports.fs = require('fs')
// npm modules
exports.connect = require('connect')
// utilities
exports.zeropad = function(num){
	return num < 10 ? '0'+num : ''+num
}
```

- Use require to load the common.js file. Anything exported by common.js can be used in the calling script:

```
var common = require('./common')
console.log( common.zeropad(1) )
var server = common.connect.createServer()
common.fs.open( '/etc/passwd', ... )
```