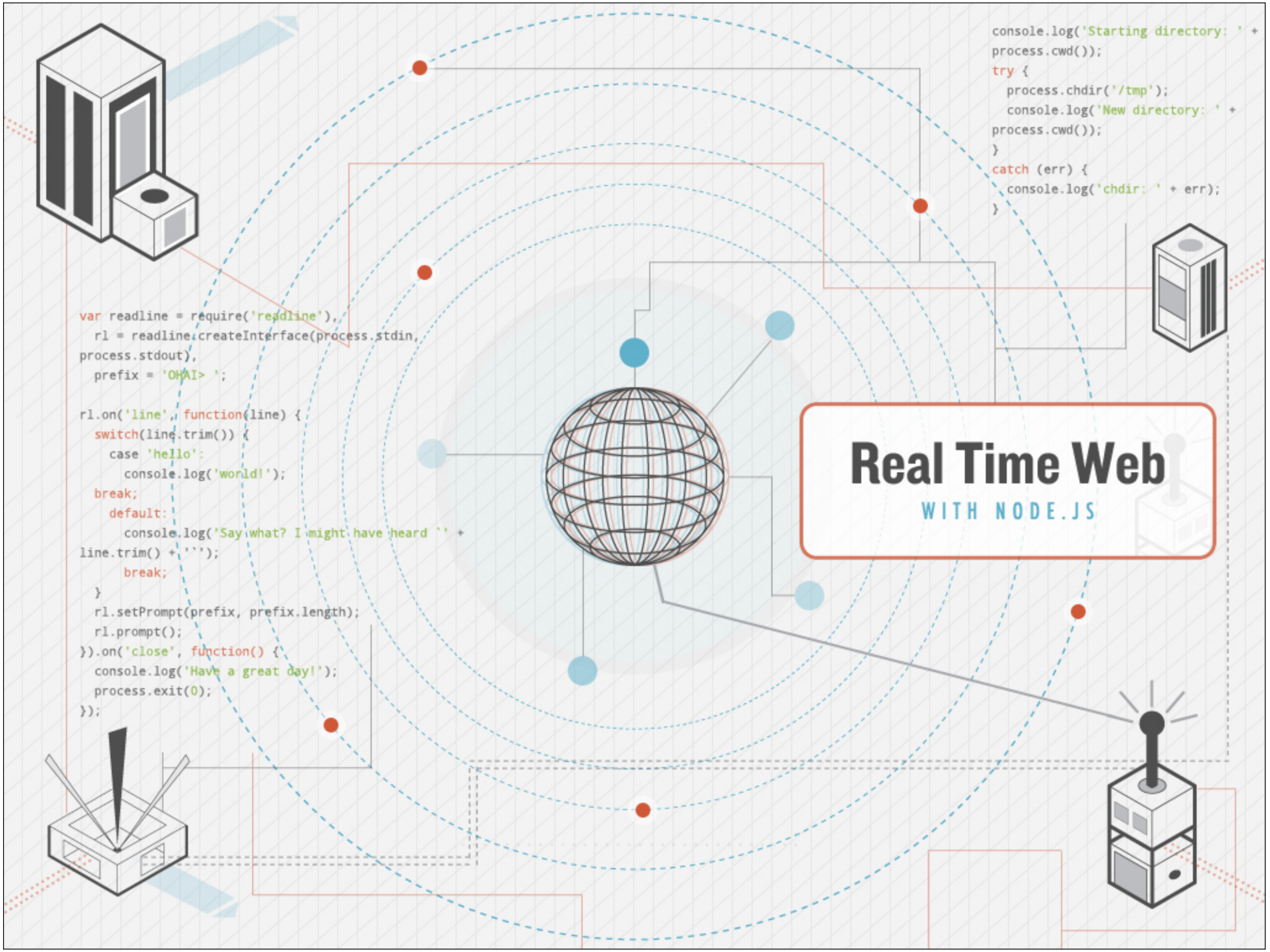
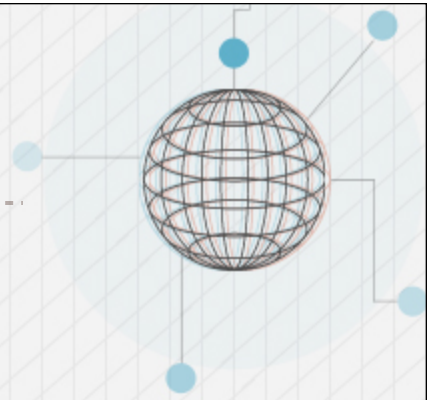

console.log('Starting directory: ' +
process.cwd());
try {
process.chdir('/tmp');
console.log('New directory: ' +
process.cwd());
}
catch (err) {
console.log('chdir: ' + err);
}
var readline = require('readline'),
rl = readline.createInterface(process.stdin,
process.stdout),
prefix = 'OHAI> ';
rl.on('line', function(line) {
switch(line.trim()) {
case 'hello':
console.log('world!');
break;
default:
console.log('Say what? I might have heard `' +
line.trim() + '`');
break;
}
rl.setPrompt(prefix, prefix.length);
rl.prompt();
}).on('close', function() {
console.log('Have a great day!');
process.exit(0);
});
Real Time Web
WITH NODE.JS

INTRO TO NODE.JS

- LEVEL ONE -

WHAT IS NODE.JS?



Allows you to build scalable network applications using JavaScript on the server-side.

Node.js

V8 JavaScript Runtime

It's fast because it's mostly C++ code

WHAT COULD YOU BUILD?

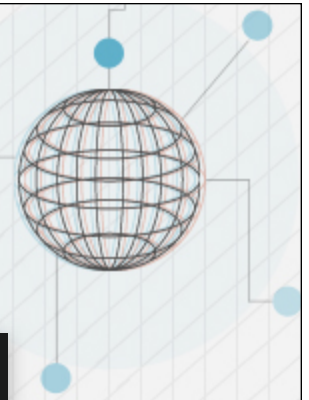
- Websocket Server *Like a chat server*
- Fast File Upload Client
- Ad Server
- Any Real-Time Data Apps

WHAT IS NODE.JS NOT?

- A Web Framework
- For Beginners *It's very low level*
- Multi-threaded

  You can think of it as a single threaded server

OBJECTIVE: PRINT FILE CONTENTS



- Blocking Code

```
Read file from Filesystem, set equal to “contents”
Print contents
Do something else
```

- Non-Blocking Code

```
Read file from Filesystem
  whenever you’re complete, print the contents
Do Something else
```

This is a "Callback"

BLOCKING VS NON-BLOCKING

- Blocking Code

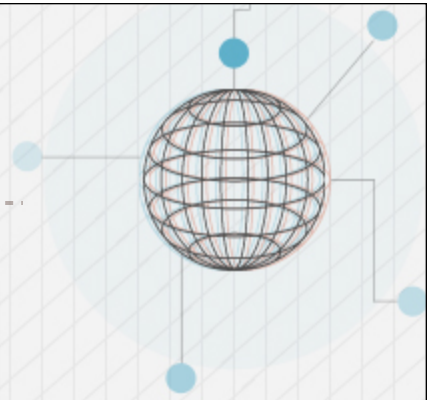
```
var contents = fs.readFileSync('/etc/hosts');
console.log(contents);
console.log('Doing something else');
```

Stop process until complete

- Non-Blocking Code

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

CALLBACK ALTERNATE SYNTAX



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

Same as

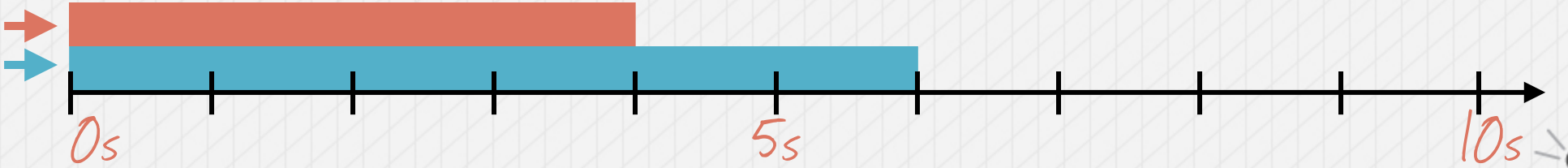
```
var callback = function(err, contents) {
  console.log(contents);
}

fs.readFile('/etc/hosts', callback);
```

BLOCKING VS NON-BLOCKING

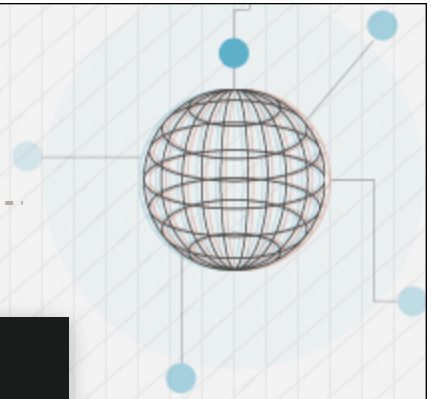
```
var callback = function(err, contents) {
  console.log(contents);
}

fs.readFile('/etc/hosts', callback);
fs.readFile('/etc/inetcfg', callback);
```

blocking

non-blocking



NODE.JS HELLO DOG



hello.js

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

http.createServer(function(request, response) {
  response.writeHead(200);
  response.write("Hello, this is dog.");
  response.end();
}).listen(8080);
console.log('Listening on port 8080...');
```

How we require modules

Status code in header

Response body

Close the connection

Listen for connections on this port

```
$ node hello.js
```

Run the server

----> Listening on port 8080...

```
$ curl http://localhost:8080
```

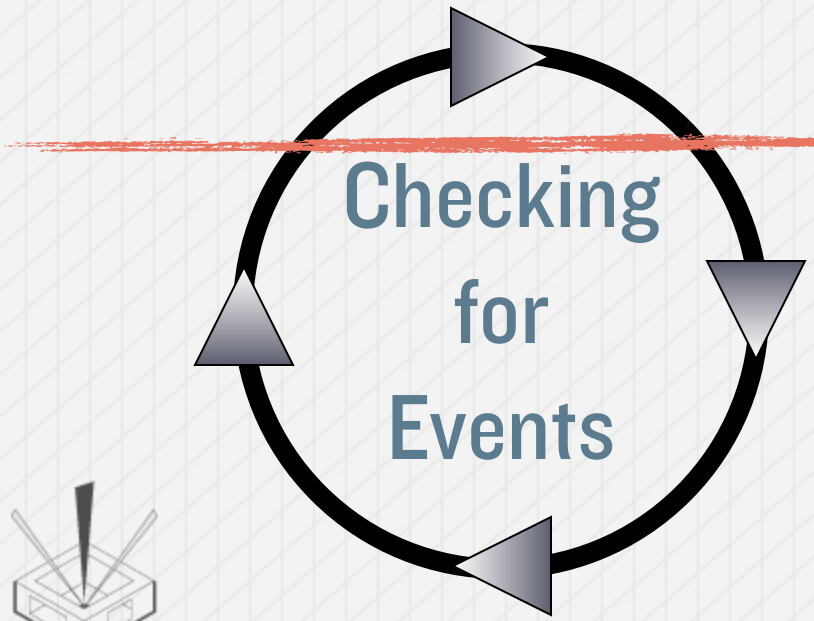
----> Hello, this is dog.

THE EVENT LOOP

```
var http = require('http');
http.createServer(function(request, response) {
  ...
}).listen(8080);
console.log('Listening on port 8080...');
```

Starts the Event Loop when finished

Run the Callback



Known Events

request

WHY JAVASCRIPT?

"JavaScript has certain characteristics that make it very different than other dynamic languages, namely that it has no concept of threads. Its model of concurrency is completely based around events."

- Ryan Dahl

THE EVENT LOOP

Event Queue

- close
- request

Known Events

- request
- connection
- close

Events processed one at a time

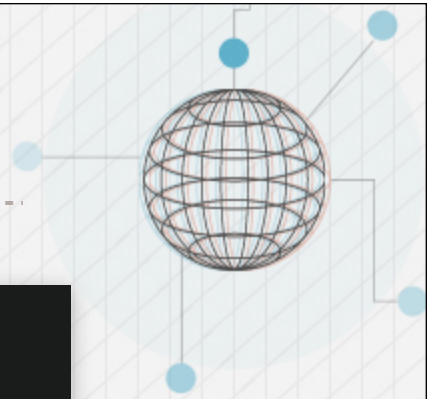
WITH LONG RUNNING PROCESS

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

http.createServer(function(request, response) {
  response.writeHead(200);
  response.write("Dog is running.");
  setTimeout(function(){
    response.write("Dog is done.");
    response.end();
  }, 5000);
}).listen(8080);
```

Represent long running process

5000ms = 5 seconds

TWO CALLBACKS HERE



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

http.createServer(function(request, response) {
  response.writeHead(200);
  response.write("Dog is running.");
  setTimeout(function(){
    response.write("Dog is done.");
    response.end();
  }, 5000);
}).listen(8080);
```

request

timeout

TWO CALLBACKS TIMELINE

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

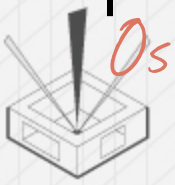
setTimeout Callback executes

triggers setTimeout event

setTimeout Callback

request

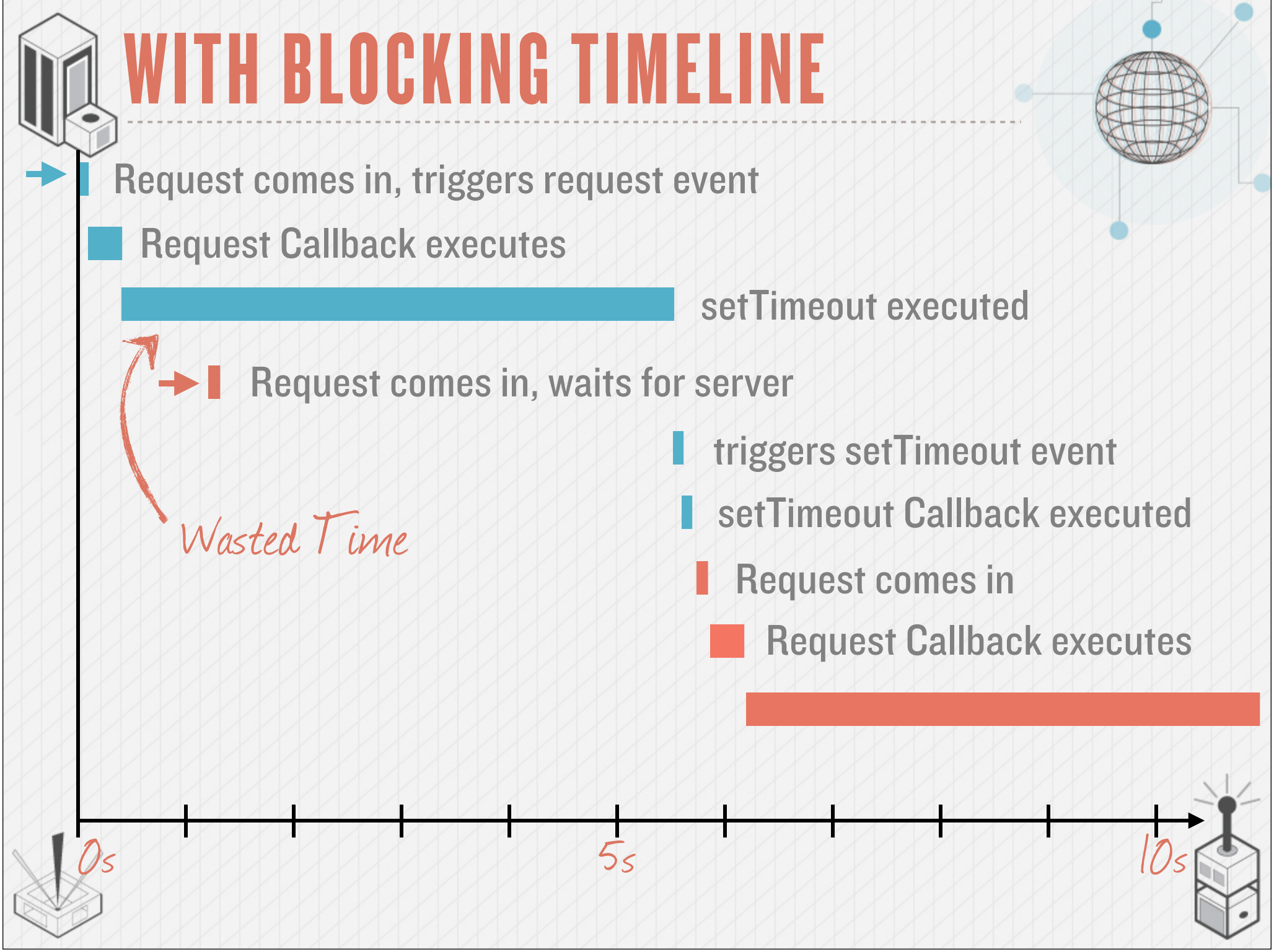
timeout



0s

5s

10s


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

TYPICAL BLOCKING THINGS

- Calls out to web services
- Reads/Writes on the Database
- Calls to extensions

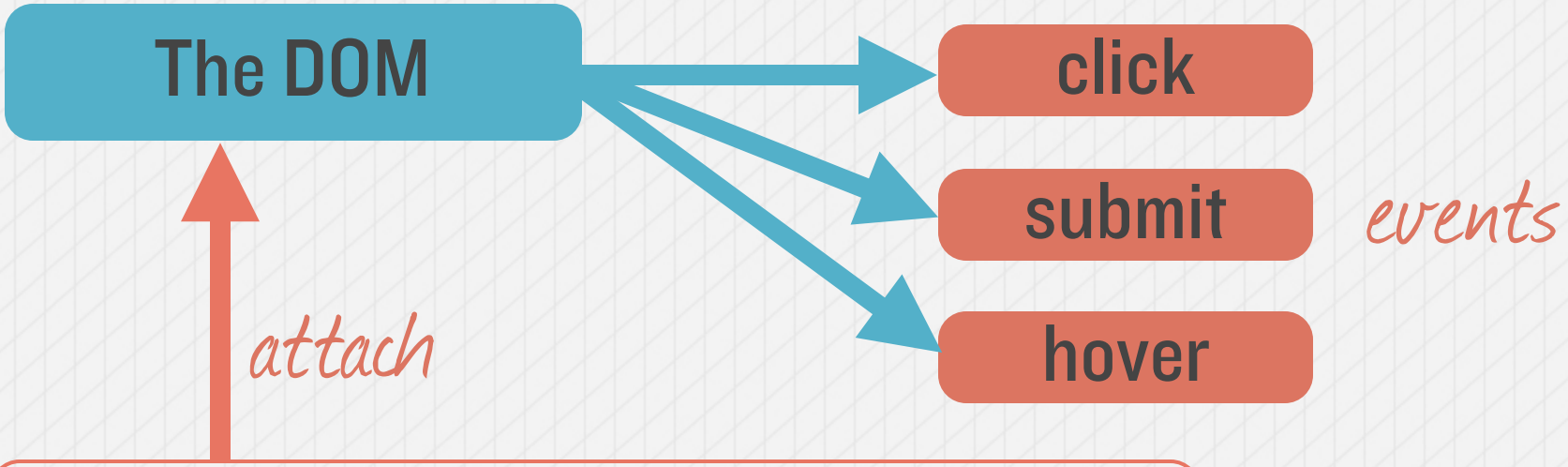
EVENTS

- LEVEL TWO -

EVENTS IN THE DOM

The DOM triggers Events

you can listen for those events

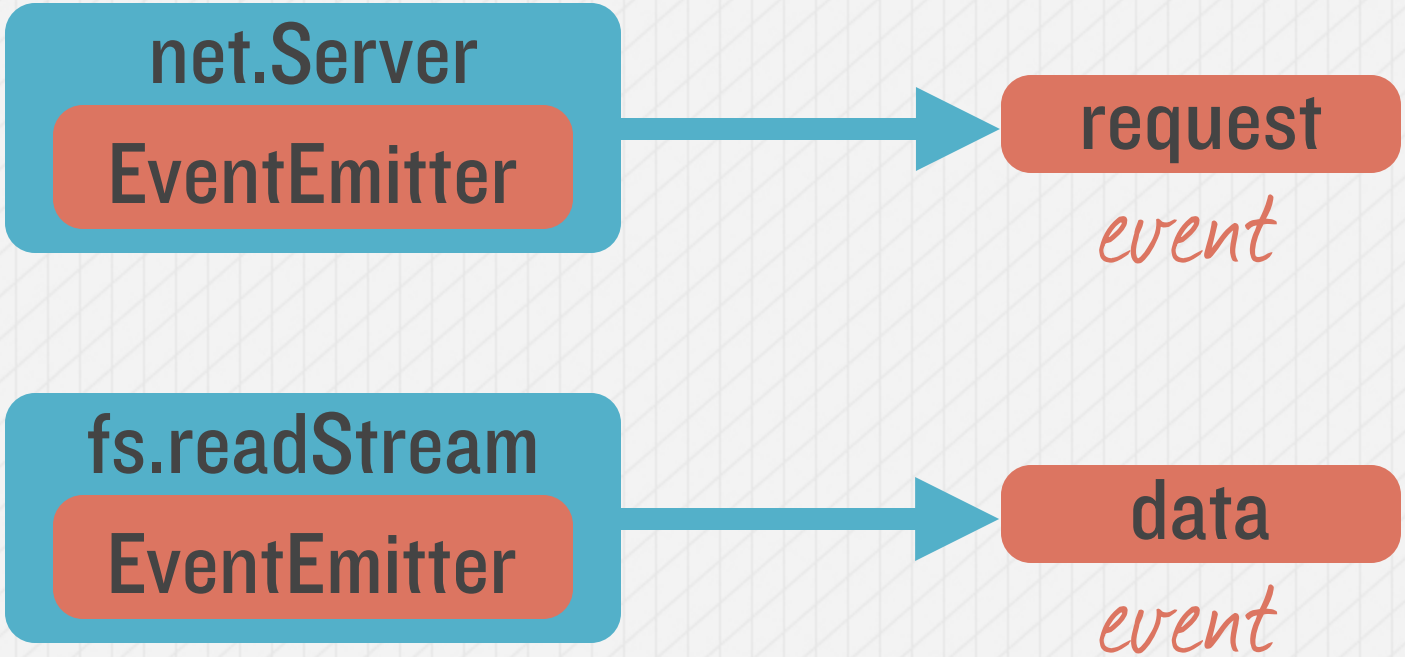



```
$("p").on("click", function(){ ... });
```

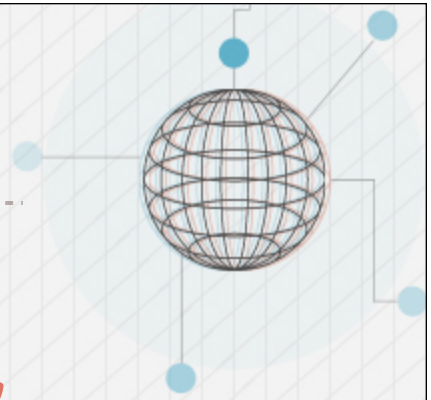
When ‘click’ event is triggered

EVENTS IN NODE

Many objects in Node emit events



CUSTOM EVENT EMITTERS



```
var EventEmitter = require('events').EventEmitter;
```

```
var logger = new EventEmitter();
```

events

error warn info

```
logger.on('error', function(message){
  console.log('ERR: ' + message);
});
```

listen for error event

```
logger.emit('error', 'Spilled Milk');
```

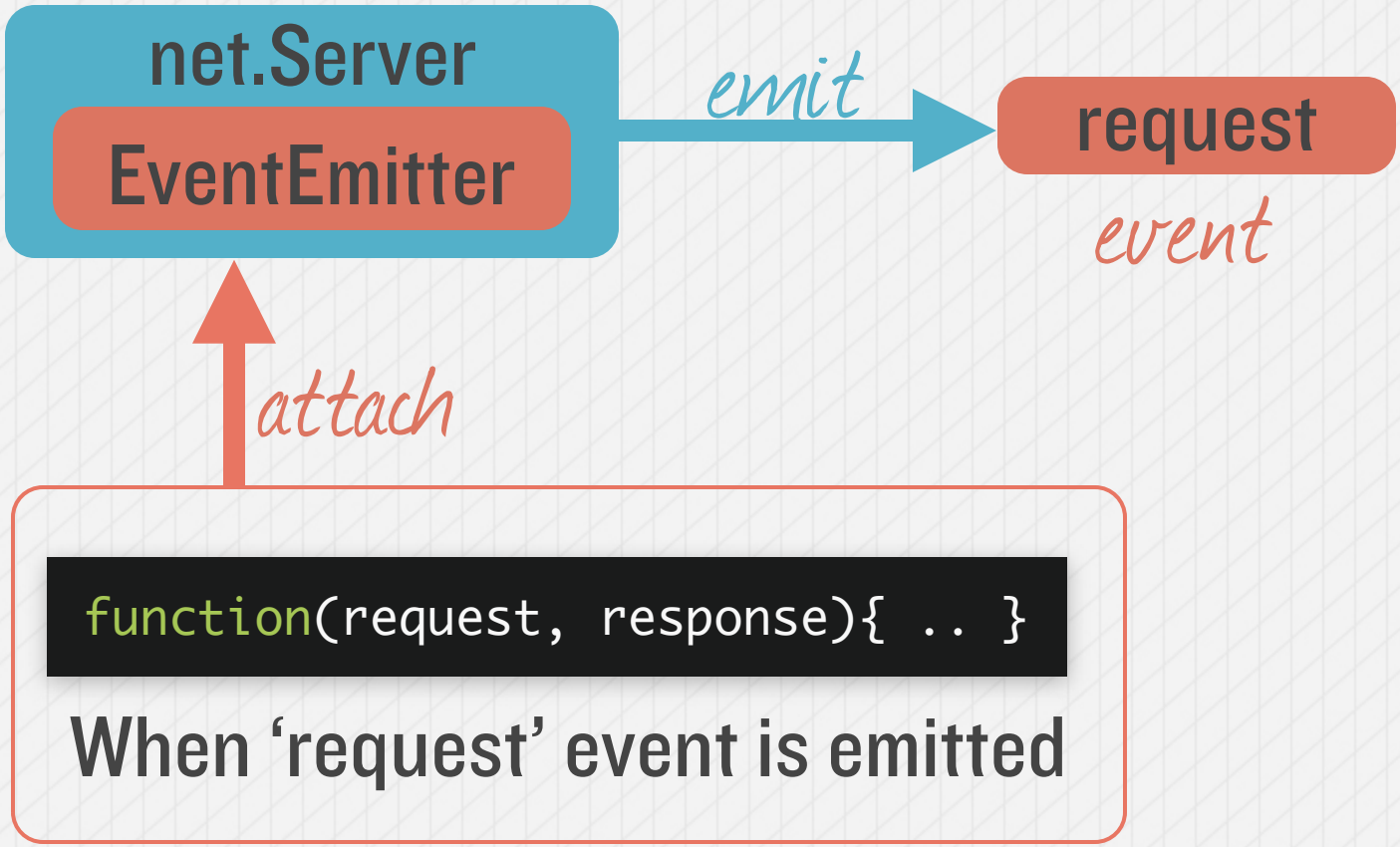
-> ERR: Spilled Milk

```
logger.emit('error', 'Eggs Cracked');
```

-> ERR: Eggs Cracked

EVENTS IN NODE

Many objects in Node emit events



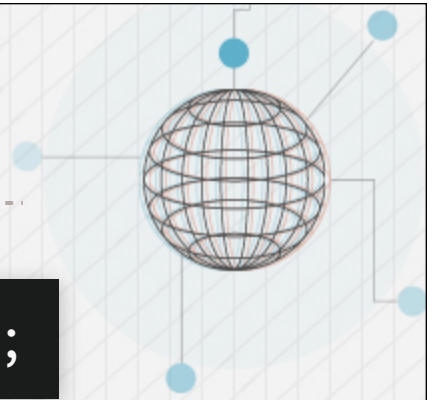
HTTP ECHO SERVER

```
http.createServer(function(request, response){ ... });
```

But what is really going on here?

http://nodejs.org/api/

BREAKING IT DOWN



```
http.createServer(function(request, response){ ... });
```

http.createServer([requestListener])

Returns a new web server object.

The `requestListener` is a function which is automatically added to the `'request'` event.

Class: http.Server

This is an `EventEmitter` with the following events:

Event: 'request'

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

Emitted each time there is a request.

ALTERNATE SYNTAX

```
http.createServer(function(request, response){ ... });
```

Same as

```
var server = http.createServer();
server.on('request', function(request, response){ ... });
```

This is how we add
add event listeners

Event: 'close'

```
function () { }
```

Emitted when the server closes.

```
server.on('close', function(){ ... });
```

MODULES

- LEVEL THREE -

REQUIRING MODULES

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

--------> http.js

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

--------> fs.js

How does ‘require’ return the libraries?

How does it find these files?

LETS CREATE OUR OWN MODULE

custom_hello.js

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

exports = hello;
```

custom_goodbye.js

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

exports defines what require returns

app.js

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

hello();
gb.goodbye();
```

```
require('./custom_goodbye').goodbye();
```

If we only need to call once

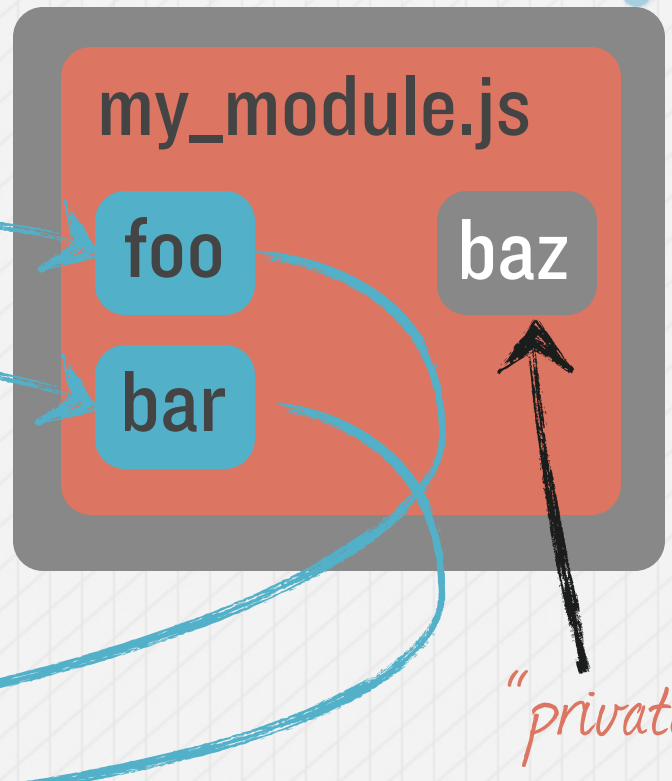
EXPORT MULTIPLE FUNCTIONS

my_module.js

```
var foo = function() { ... }
var bar = function() { ... }
var baz = function() { ... }

exports.foo = foo
exports.bar = bar
```

app.js

```
var myMod = require('./my_module');
myMod.foo();
myMod.bar();
```



MAKING HTTP REQUESTS

app.js

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

var message = "Here's looking at you, kid.";
var options = {
  host: 'localhost', port: 8080, path: '/', method: 'POST'
}

var request = http.request(options, function(response){
  response.on('data', function(data){
    console.log(data); logs response body
  });
});
request.write(message); begins request
request.end(); finishes request
```

ENCAPSULATING THE FUNCTION

app.js

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

var makeRequest = function(message) {
  var options = {
    host: 'localhost', port: 8080, path: '/', method: 'POST'
  }

  var request = http.request(options, function(response){
    response.on('data', function(data){
      console.log(data);
    });
  });
  request.write(message);
  request.end();
}

makeRequest("Here's looking at you, kid.");
```

CREATING & USING A MODULE

make_request.js

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

var makeRequest = function(message) {
   ...
}

exports = makeRequest;
```

app.js

```
var makeRequest = require('./make_request');

makeRequest("Here's looking at you, kid");
makeRequest("Hello, this is dog");
```

Where does require look for modules?

REQUIRE SEARCH

```
var make_request = require('./make_request')
var make_request = require('../make_request')
var make_request = require('/Users/user/nodes/make_request')
```

look in same directory

look in parent directory

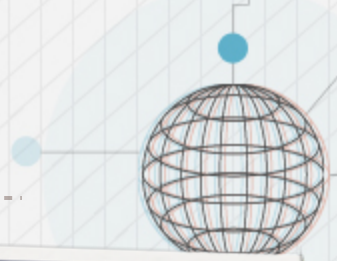
/Home/user/my_app/app.js

Search in node_modules directories

```
var make_request = require('make_request')
```

- /Home/user/my_app/node_modules/
- /Home/user/node_modules/make_request.js
- /Home/node_modules/make_request.js
- /node_modules/make_request.js

NPM: THE USERLAND SEA



Package manager for node

- Comes with node
- Module Repository
- Dependency Management
- Easily publish modules
- “Local Only”

“Core” is small. “Userland” is large.

INSTALLING A NPM MODULE

In /Home/my_app

```
$ npm install request
```

--> https://github.com/mikeal/request

Installs into local node_modules directory

```
Home / my_app / node_modules / request
```

In /Home/my_app/app.js

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

Loads from local node_modules directory

LOCAL VS GLOBAL

Install modules with executables globally

```
$ npm install coffee-script -g
```

global

```
$ coffee app.coffee
```

Global npm modules can't be required

```
var coffee = require('coffee-script');
```

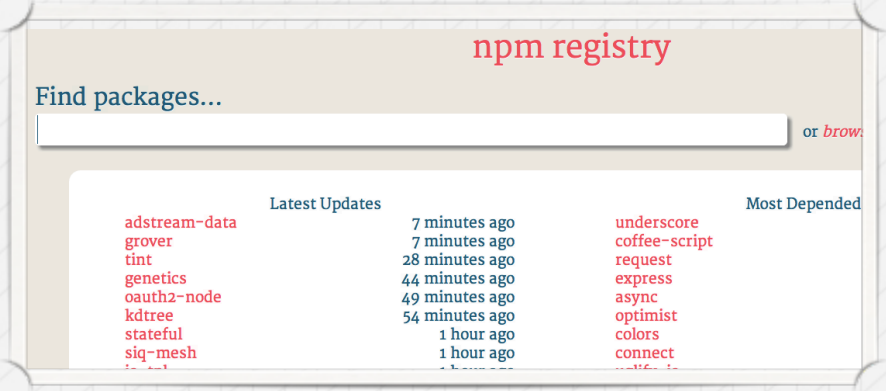
```
$ npm install coffee-script
```

Install them locally

```
var coffee = require('coffee-script');
```

FINDING MODULES

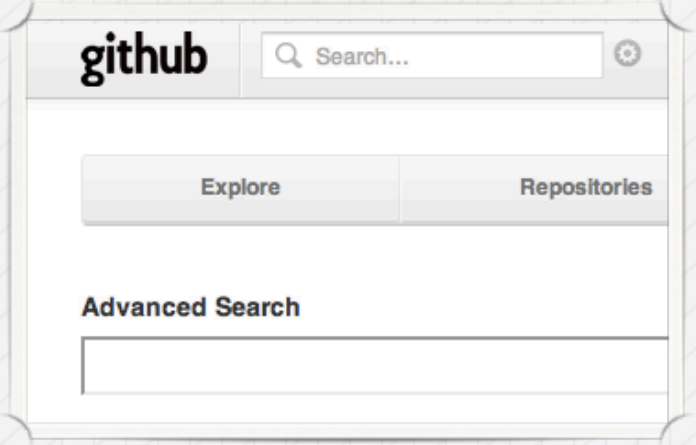
npm registry



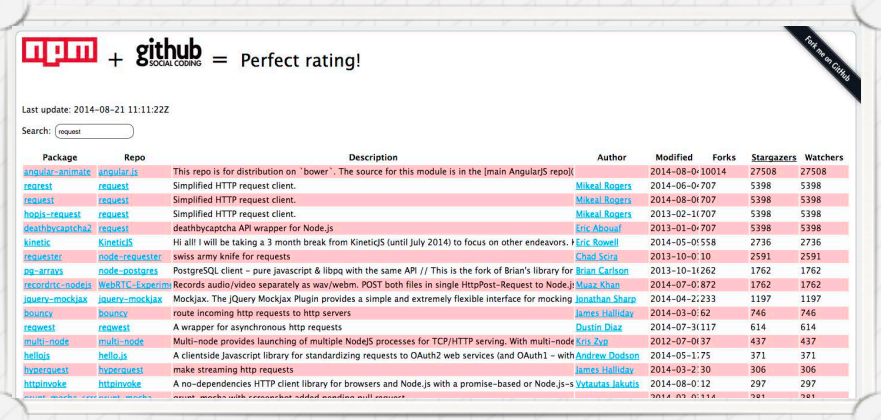
npm command line

```
$ npm search request
```

github search



eirikb.github.io/nipster



DEFINING YOUR DEPENDENCIES

my_app/package.json

```
{
  "name": "My App",
  "version": "1",
  "dependencies": {
    "connect": "1.8.7"
  }
}
```

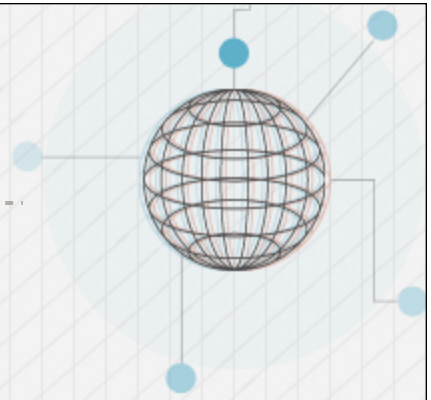
version number

```
$ npm install
```

Installs into the node_modules directory

my_app / node_modules / connect

DEPENDENCIES



my_app/package.json

```
"dependencies": {
  "connect": "1.8.7"
}
```

No conflicting modules!

Installs sub-dependencies

my_app / node_modules / connect

connect / node_modules / qs

connect / node_modules / mime

connect / node_modules / formidable

SEMANTIC VERSIONING

Major Minor Patch

```
"connect": "1.8.7"
```

1 . 8 . 7

Ranges

`"connect": "~1"`	-->	`>=1.0.0 <2.0.0`
`"connect": "~1.8"`	-->	`>=1.8 <2.0.0`
`"connect": "~1.8.7"`	-->	`>=1.8.7 <1.9.0` **Safest**
`"connect": "^1.8.7"`	-->	`>=1.8.7 <2.0.0` **Default**

http://semver.org/