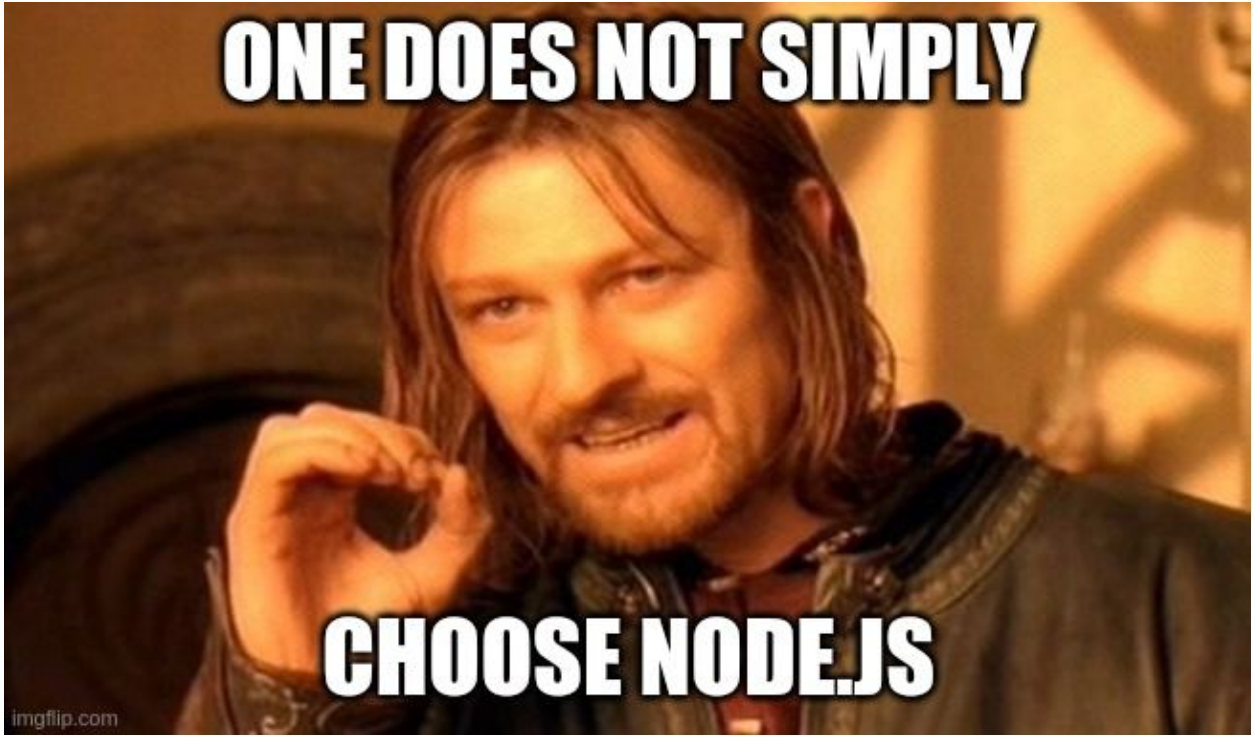
MENTORMATE™
DevCamp

Introduction to Node.js

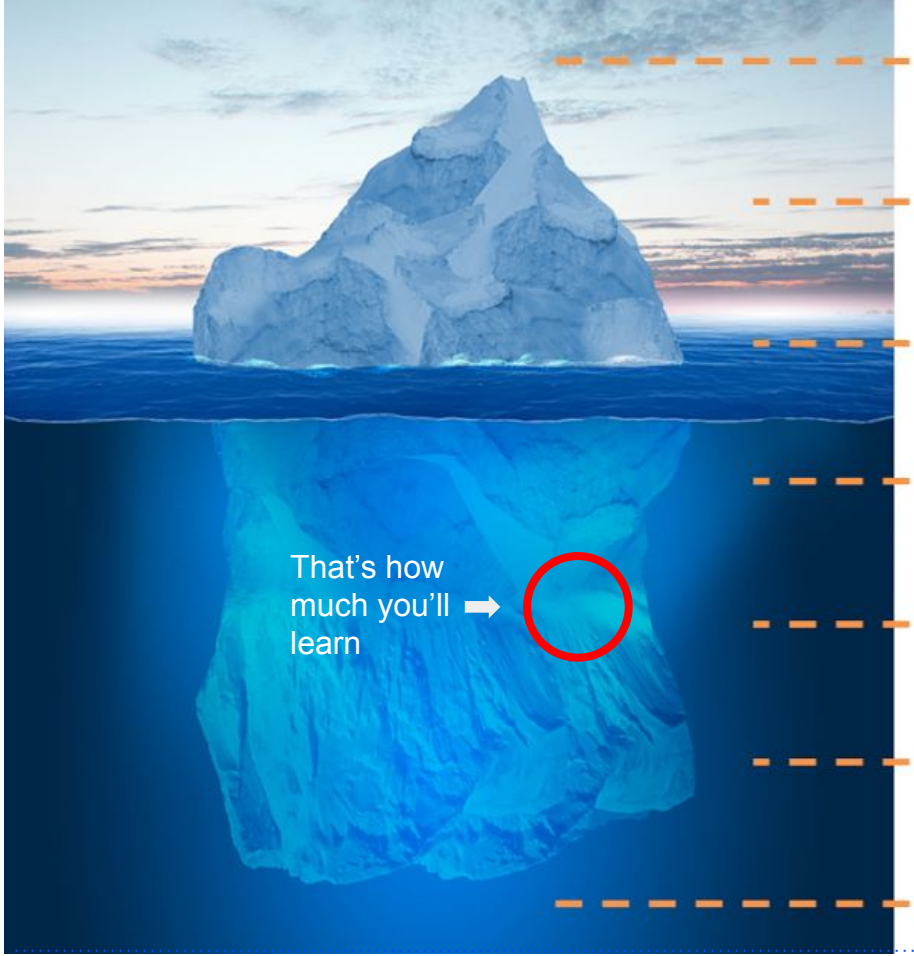
Yulia Tenincheva

Senior Cloud Engineer, MentorMate


ONE DOES NOT SIMPLY
CHOOSE NODE.JS
imgflip.com

What will you learn?

- Understanding how **Node.js Core** works
- How to **debug**, **test** and **handle errors**
- How to build a **REST API / Web Server**
- How to use a **Database** with Node.js
- How to build **Secure** applications
- How to create **real-time** applications
- What is a **Cloud** and why you should care
- Building **cloud-native** applications
- How to **deploy** and **scale** Node.js apps
- How to build **Serverless** applications



User Experience
(UI, navigation, fun factor)

Functionality
(usefulness, productivity)

Distribution
(Store, deployment, lifecycle)

Back-End Services
(logic, data, social, enterprise)

Global Infrastructure
(data centers, cloud computing)

Intelligence
(analysis, business insight)

Today's Agenda

- **What is Node.js**, advantages
- Node.js **Event Loop**
- How to learn Node.js, **Resources**
- Major environment differences
- Node.js **Module System**
- **Core modules** - overview
- **Live Exploring** (*path*, *os*, *fs*, *events*, *http)*

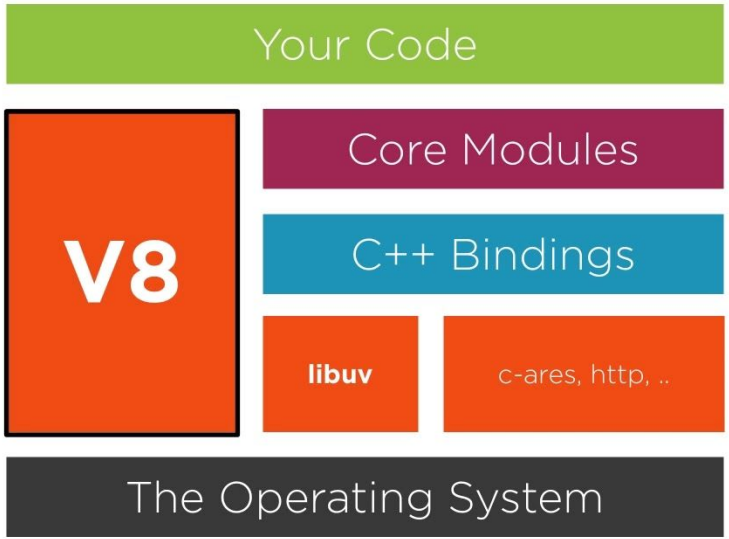
So WHAT is Node.js?

“Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.”

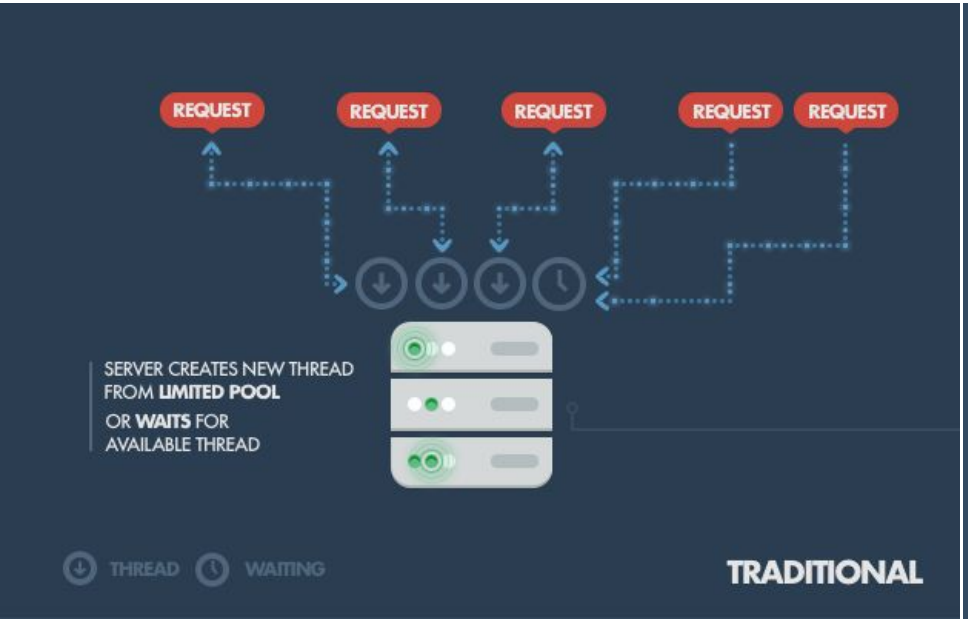
—Official Documentation

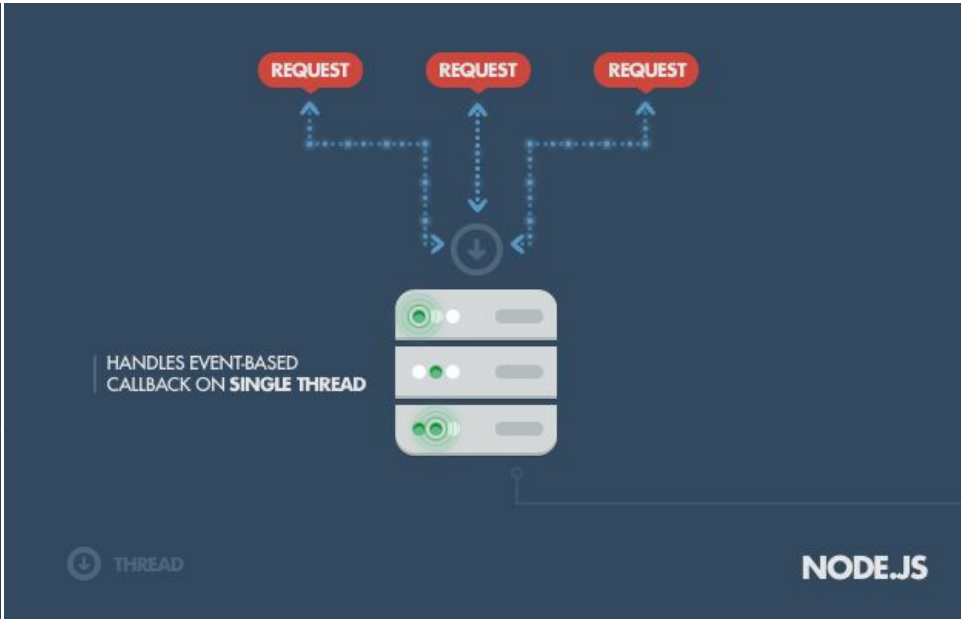
Node.js is...

- **Written in C, C++ and JavaScript**
- **Cross-Platform**
- **Asynchronous** and **Event Driven**
- **Single threaded** but **highly scalable**
- **Open Source**
- **Simple to use**



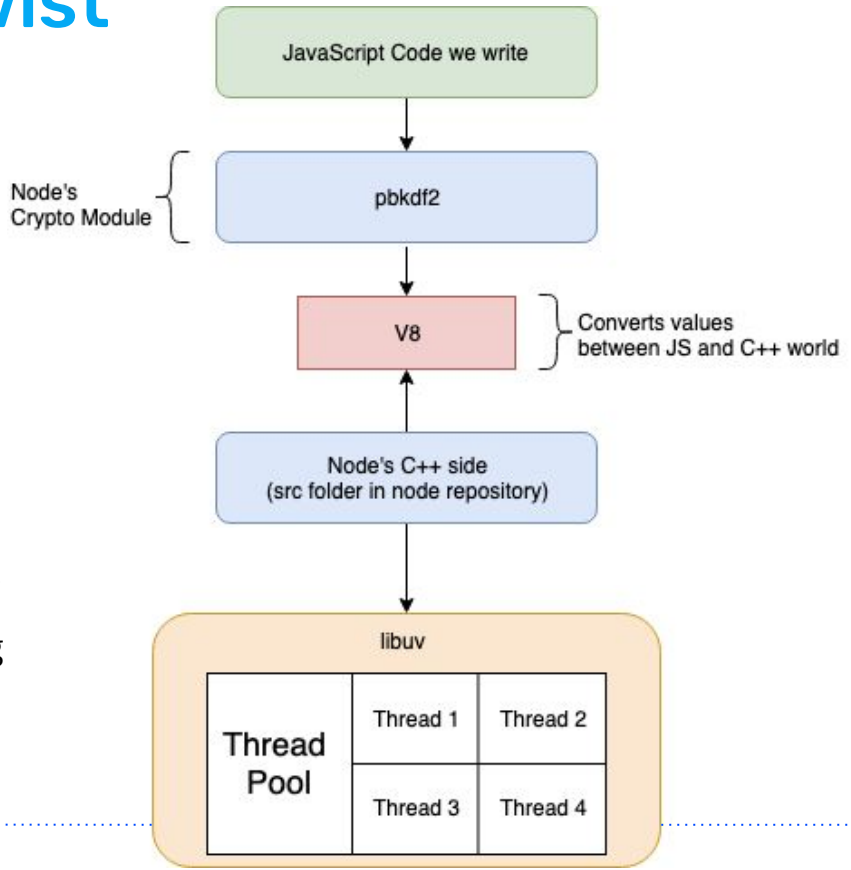
Multi-threaded vs single-threaded



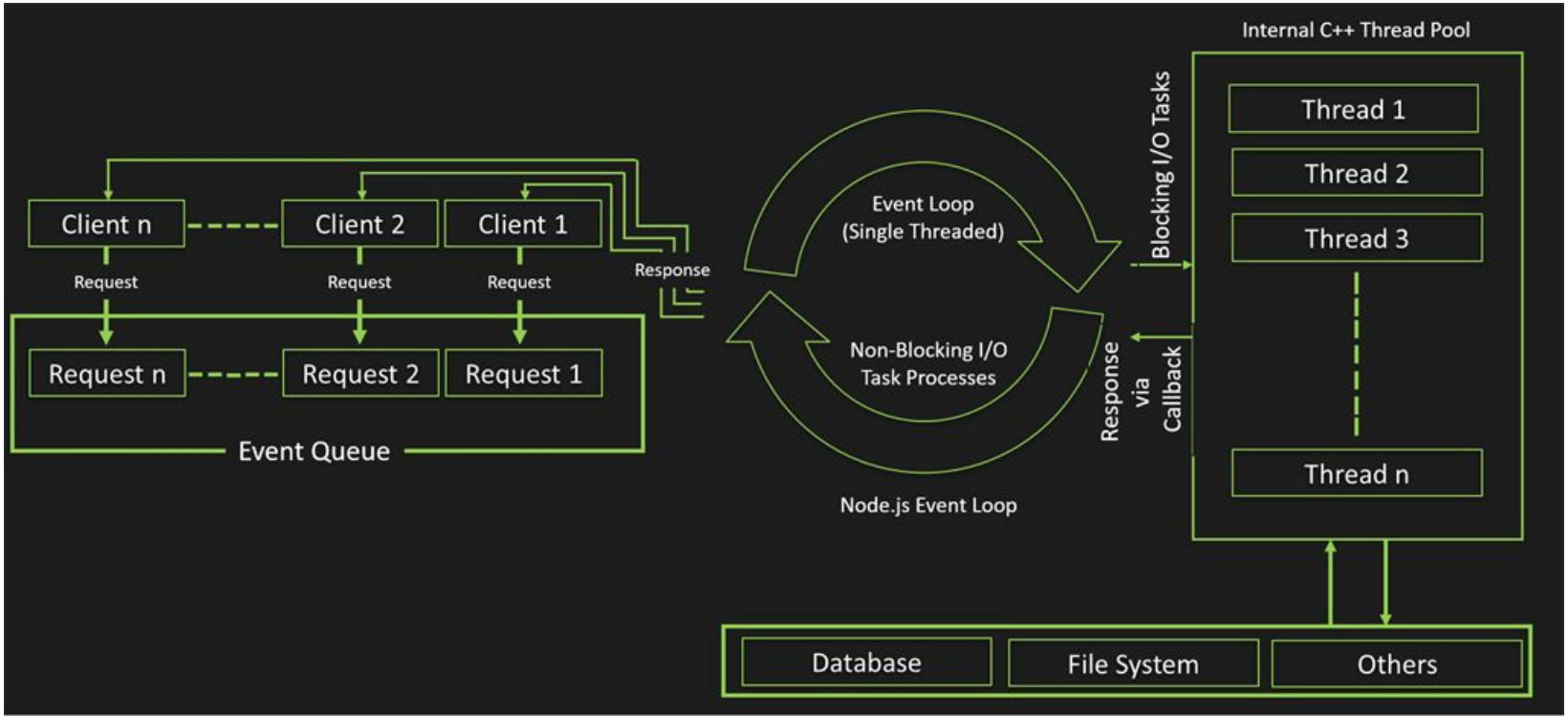


Single-Threaded with a Twist

- A Node.js application runs on single thread and the event loop also runs on the same thread.
- Node.js internally uses the **libuv** library which is responsible for handling operating system related tasks, networking, concurrency, etc.
- Libuv sets up a **thread pool** of *(Number of CPU Cores)* threads to perform OS-related operations by utilizing the power of all the CPU cores.



Single-Threaded with a Twist



Who uses Node.js?

- Netflix & PayPal
- Venturebeat
- Linux Foundation
- LinkedIn
- NASA
- When and how Node.js should be used

History

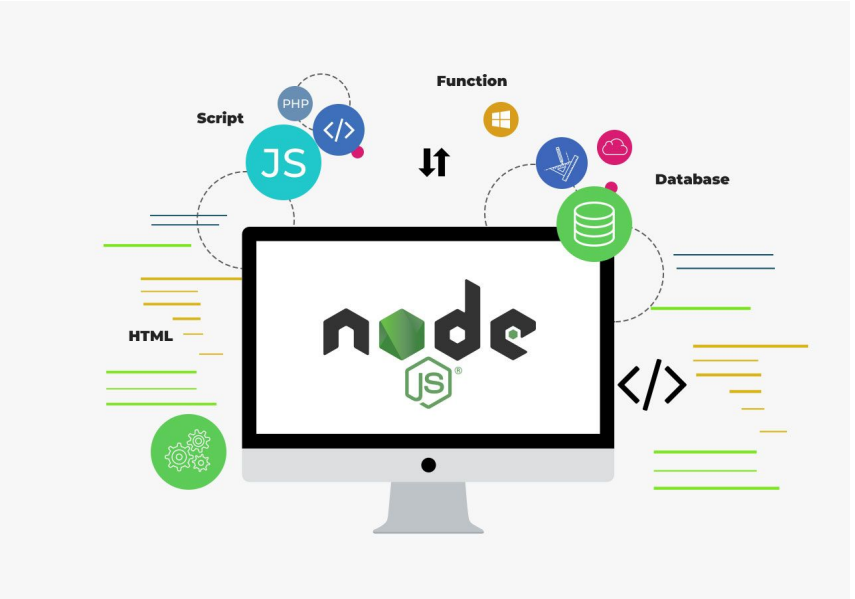
- **2009** - Node.js Created by Ryan Dahl
- 2010 - Express Framework and Socket.io created
- 2011 - NPM Created, Node.js Released for Windows
  Hapi.js Framework is created
- **2012**, 2013 - Adoption grows rapidly, Ryan Dahl steps away
- **2014** - io.js forks Node.js, *big drama*
- 2015 - Node.js Foundation is born
  Io.js is merged back into Node.js v4, *everyone is happy*

History

- 2016 - The leftpad incident, yarn is born, Node v6
- **2017** - NPM focuses more on security, Node v8 & v9
  HTTP/2 support, 3 billion npm downloads every week
  V8 becomes an official target for the JS Engine
- 2018 - Node v10 & v11, ES modules experimental support
- 2019 - Node v12 & v13, deno created by Ryan Dahl
- 2020 - Node v14 & v15, Github acquired NPM
- Interactive History

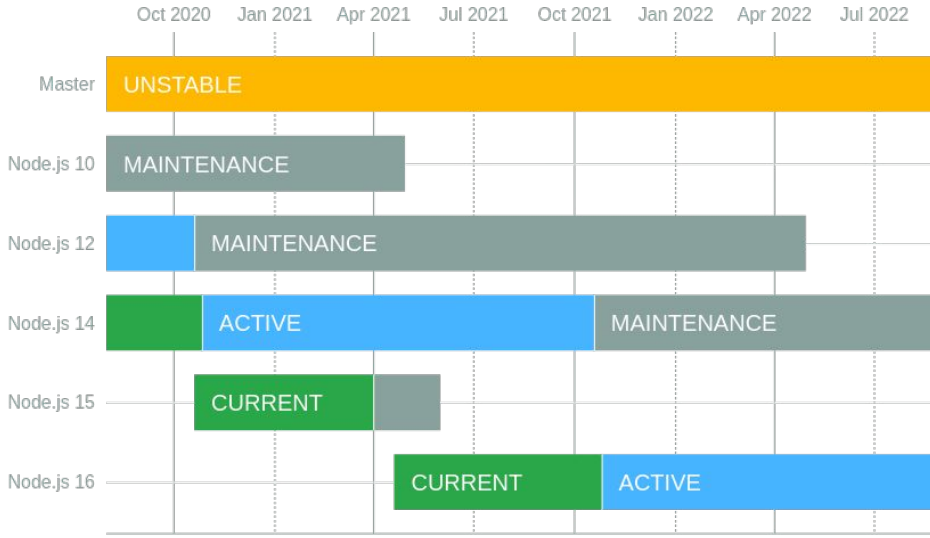
How to Learn Node.js

- Node.js Official Documentation
- Node Weekly Newsletter
- Node.js best practices guide
- Hands on
- Nodetuts - Series of talks
- 10 Things I Regret About Node.js by Ryan Dahl - Insightful talk by the creator of Node.js about some of its limitations.



Development Environment

- Install Node.js v14.9.0 (Current) instead of LTS
- Install Postman
- JSON View (or similar) plugin
- Install AWS CLI Tool
- Install PostgreSQL Database



“Talk is cheap. Show me the code!”

—Linus Torvalds

Global Scope

- **Global** scope, **global** object compared to browser's **window** object
  - What is "**this**" (context) in Node.js ?
  - global is the **global namespace object**, such as window, but the scope is not the same
  - In Node.js the **top-level scope is not the global scope**.

- Global objects, specific to Node.js
  - `__dirname`
  - `__filename`
  - `process`
  - `exports`
  - `module`
  - `require()`

```
> global
<ref *1> Object [global] {
  global: [Circular *1],
  clearInterval: [Function: clearInterval],
  clearTimeout: [Function: clearTimeout],
  setInterval: [Function: setInterval],
  setTimeout: [Function: setTimeout] {
    [Symbol(nodejs.util.promisify.custom)]: [Function (anonymous)]
  },
  queueMicrotask: [Function: queueMicrotask],
  clearImmediate: [Function: clearImmediate],
  setImmediate: [Function: setImmediate] {
    [Symbol(nodejs.util.promisify.custom)]: [Function (anonymous)]
  }
}
```

Timers

- Timers guide & process.nextTick() guide

- *“When I say so"* Execution ~ setTimeout(), clearTimeout()
  will execute as close to N milliseconds as possible, but no earlier

- *“Right after this"* Execution ~ setImmediate(), clearImmediate()
  will execute code at the end of the current event loop cycle

- *“Infinite Loop"* Execution ~ setInterval(), clearInterval()
  will execute an infinite number of times with a given ms delay

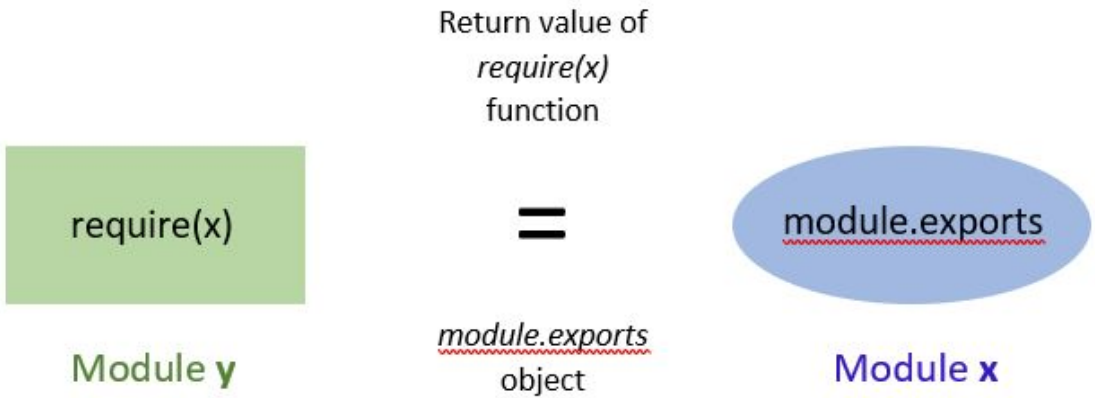
- *“More immediate"* Execution ~ process.nextTick()
  will run before any Immediates or any scheduled I/O, non-clearable

process object

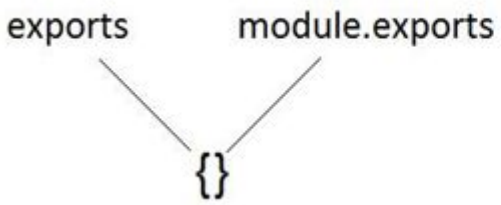
- Provides information about, and control over, the current Node.js process. Documentation
- Events
  - Event: 'beforeExit'
  - Event: 'disconnect'
  - Event: 'exit'
  - 'SIGINT', ‘SIGTERM’
- process.argv

Node.js Module System

- CommonJS Spec
- How require() works
- module.exports vs exports
- module object
- Wrapper function
- ES6 Modules Support



Module.exports vs exports



Both of them are references to the same (empty) object at the beggining. (But only module.exports will be returned!)

exports is a variable and **module.exports** is an attribute of the module object.

```
exports.msg = 'hi';
console.log(module.exports === exports); // true

exports = 'yo';
console.log(module.exports === exports); // false

exports = module.exports;
console.log(module.exports === exports); // true

module.exports = 'hello';
console.log(module.exports === exports); // false

module.exports = exports;
console.log(module.exports === exports); // true
```

require();

- How `require()` works - pseudocode
- Caching
  Modules are cached after the first time they are loaded. This means (among other things) that every call to require('foo') will get exactly the same object returned, if it would resolve to the same file.
- module.exports vs exports
- Circular `require()` calls
- Folders as modules
- The module wrapper

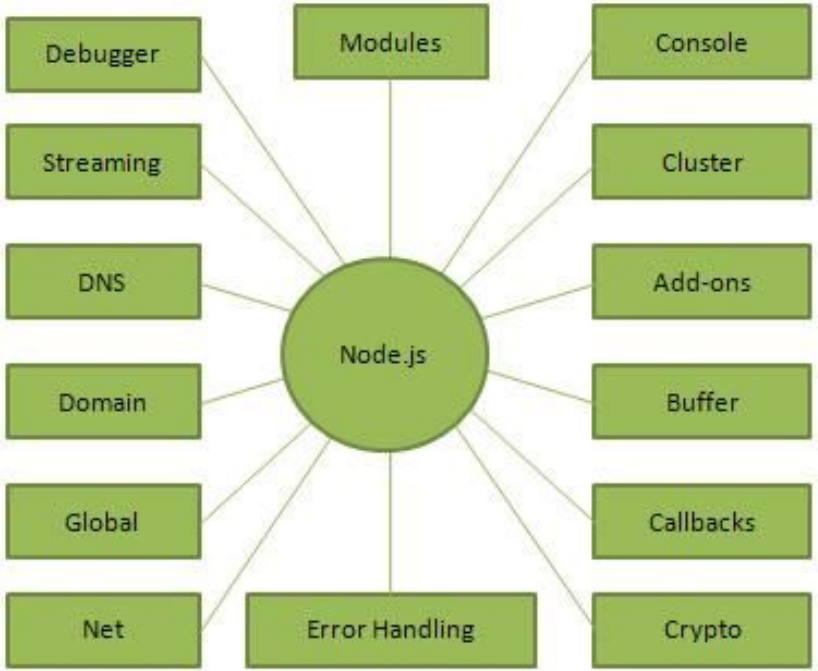
```
require('something');
```

Node.js Core Modules

- ```
  require("module").builtinModules
  ```

- ```
  require("module").builtinModules.length; // 56
  ```

- ```
  require("module").builtinModules.filter(x => {
    return !x.startsWith('_')
  }).length; // 56
  ```


  ```

Node.js Core Modules

Module	Description
assert	Provides a set of assertion tests
buffer	To handle binary data
child_process	To run a child process
cluster	To split a single Node process into multiple processes
crypto	To handle OpenSSL cryptographic functions
dns	To do DNS lookups and name resolution functions
events	To handle events
fs	To handle the file system
http/s	To make Node.js act as an HTTP/s server
net	To create servers and clients
os	Provides information about the operation system
path	To handle file paths
querystring	To handle URL query strings
readline	To handle readable streams one line at the time
stream	To handle streaming data
url	To parse URL strings
util	To access utility functions

Let's build a simple logger

EventEmmiter example with ES6 Classes

Homework

1. **Testing Framework**

   Choose any suitable 3 tasks from JavaScript Basics homeworks.

   - Wrap them into Node.js modules. Each module should export one function so that it can be tested from another module.
   - Create another file (e.g “test.js”) and by using the “assert” node.js built-in module, test the expected behavior versus the actual result.
   - Add more sample inputs. Try with different data types.

2. Coffee Bar

Create an **Event-Driven** (using the EventEmitter), asynchronous model of a coffee bar. On a random **interval** (between 1 and 5 seconds), a new client comes in and orders one or more coffees (1 to 5 on random). The type of the coffee is also random. Each coffee takes some **time** to prepare and it has a **price**.

- "*Espresso*" takes ~500ms to prepare, costs 1$
- "*Cappuccino*" takes ~1 second, costs 3.50$
- "*Latte*" takes ~1.5 seconds, costs 4.30$
- "*Americano*" takes ~700ms, costs 1.50$

Let your program work for 45 seconds. Create a **break** after 12 seconds that lasts for 5 seconds. No orders should be requested, accepted or processed at this interval. At the end of the day (45sec), your program should stop receiving new orders, but the existing ones should be fulfilled. Calculate the total profit for the day and print it before the program exits. Throughout the day there is a small (10%)

Bonus requirement: Throughout the day there is a small (10%) chance for a **VIP client** to show up. Their orders should be processed with a **priority** (coffee preparation time stays the same). They leave a **30% tip** for the coffee.