

Node Program

Node Basics



Node.js version: 7
Last updated: Nov 2016

Node.js

Introduction

Why Server-Side JavaScript?

Node was originally born out of this problem – how can you handle two things at the same time

— Ryan Dahl, The Creator of Node.js

Why Server-Side JavaScript?

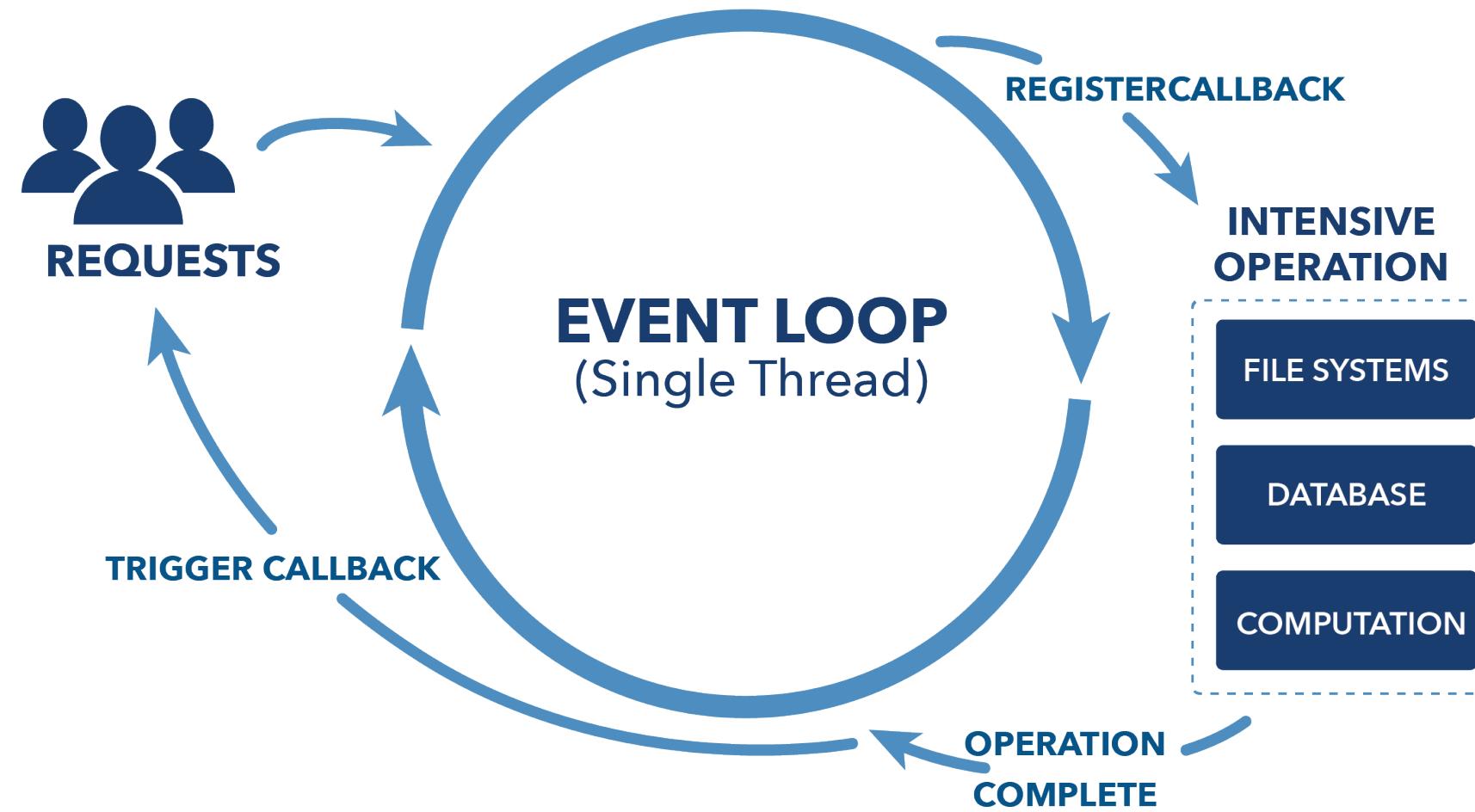
- » Non-blocking I/O: performant
- » Fast: browser arms race (V8)
- » **One language across the stack**
- » Expressive: don't waste time on setup
- » Solid standard (ECMA)

Advantages of Node.js

- >> Non-blocking I/O
- >> Super fast (V8)
- >> Vibrant ecosystem (npm)
- >> Ability to re-use code on browser and server
- >> Ability to use front-end devs for back-end and vice versa

Non-blocking I/O

It's kind of a big deal



Disadvantages of Node.js

- » Devs have to think in async and functional+prototypal
- » Frameworks and tools are not as mature as in Ruby, Java, Python (yet)
- » JavaScript "quirks" (mostly fixed in ES6!)

Node Gotcha

Don't use Node.js for CPU-intensive tasks. Hand them over to other workers.

Downsides of JavaScript (Not only Node)

- » Callback Hell
- » Prototypal inheritance

JavaScript is Optional in Node.js

It's **possible** to use other languages for Node.js that compile into JavaScript, e.g., CoffeeScript, TypeScript, and ClojureScript.

Nodies are not just Silicon Valley hipsters !

NODE IS DEPLOYED BY BIG BRANDS

Big brands are using Node to power their business

Manufacturing



General Motors

Johnson Controls

SIEMENS

Financial



citigroup

Goldman Sachs

PayPal



eCommerce

amazon.com



ebay

TARGET

Zappos.com

Media



CONDÉ NAST

DOWJONES

The New York Times

SONY

Technology

salesforce.com



box



YAHOO!

IBM



airbnb

PEARSON

STAPLES

HBO

BARNES&NOBLE

redhat.

npr

ORACLE

GoDaddy.com

NETFLIX

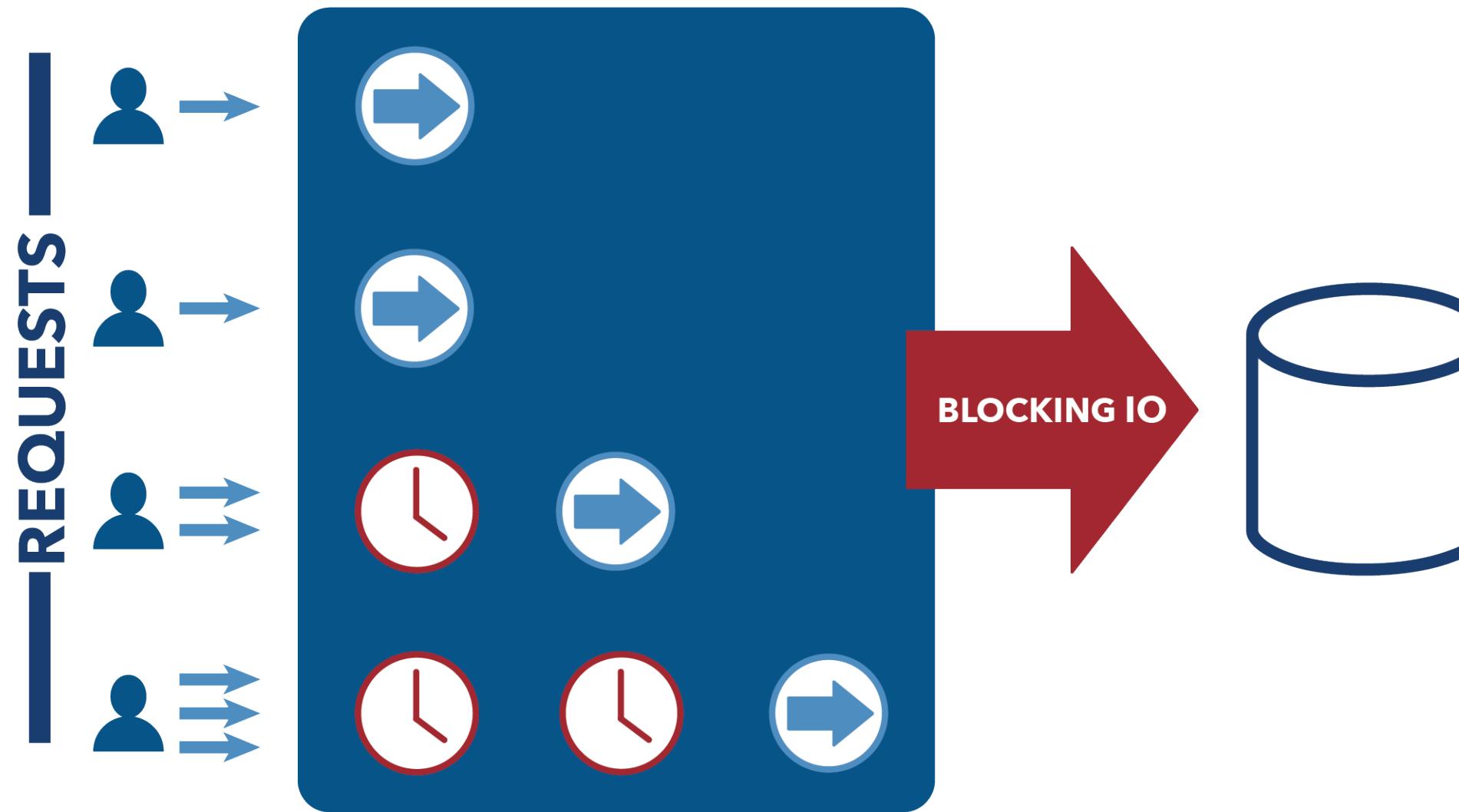
StrongLoop

Node is Single-Threaded

Node.js is single-threaded by design to make asynchronous processing simpler. Multi-threading can be very complex: racing condition, deadlocks, priority inversions...

It turned out for web-based application, single-threaded asynchronous event-loop based non-blocking I/O is very performant!

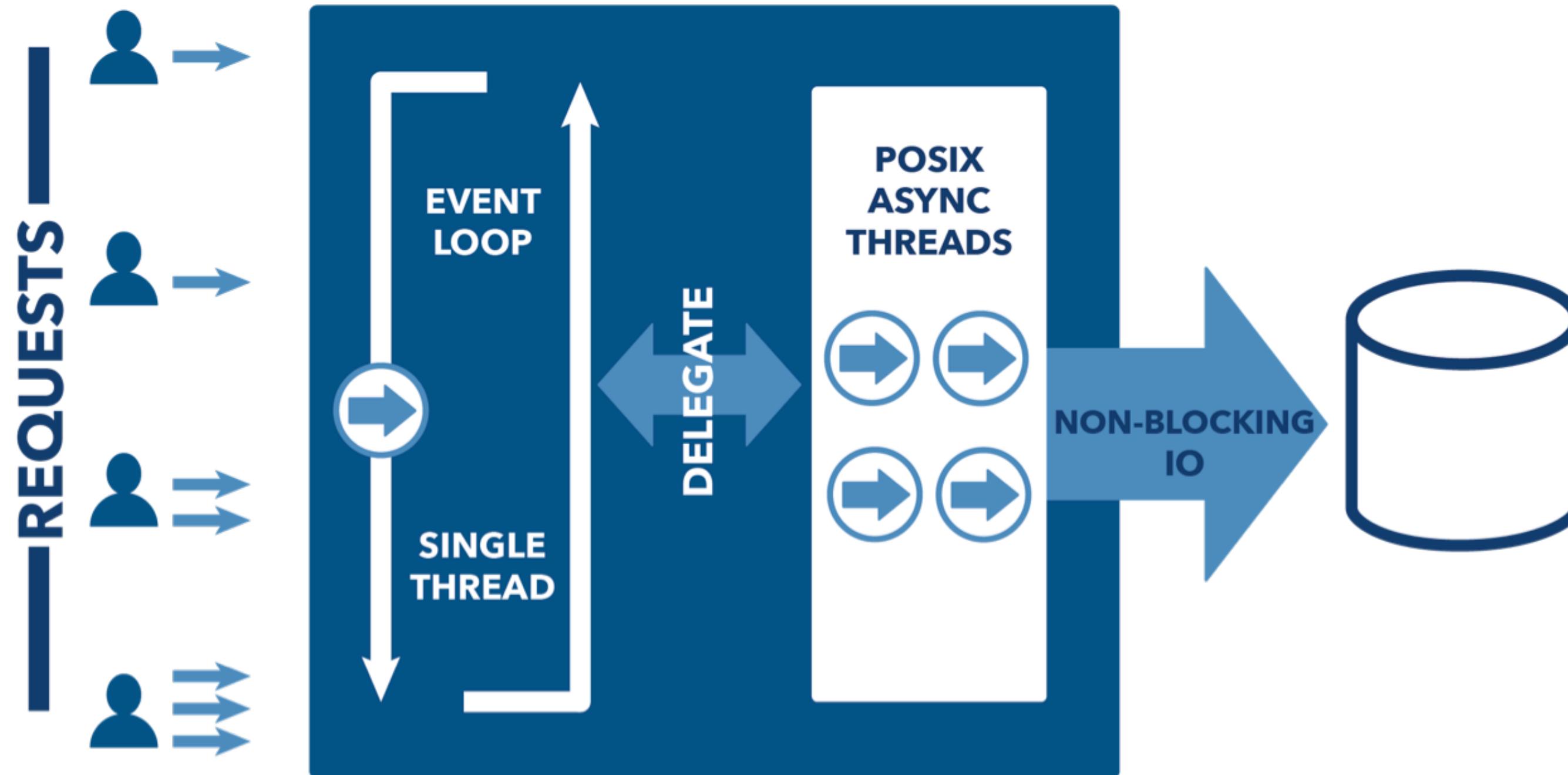
MULTI THREADED SERVER



THREAD
PROCESSING

THREAD
WAITING







Integration

>> noSQL

>> SQL

>> OAuth 1.0/2.0

>> REST

>> SOAP

Databases

- >> mySQL
- >> PostgreSQL
- >> Oracle
- >> MS SQL
- >> MongoDB
- >> Cassandra

Node + Client MVC Architecture

Single-Page Applications a.k.a. BYOC: REST API in Node + SPA

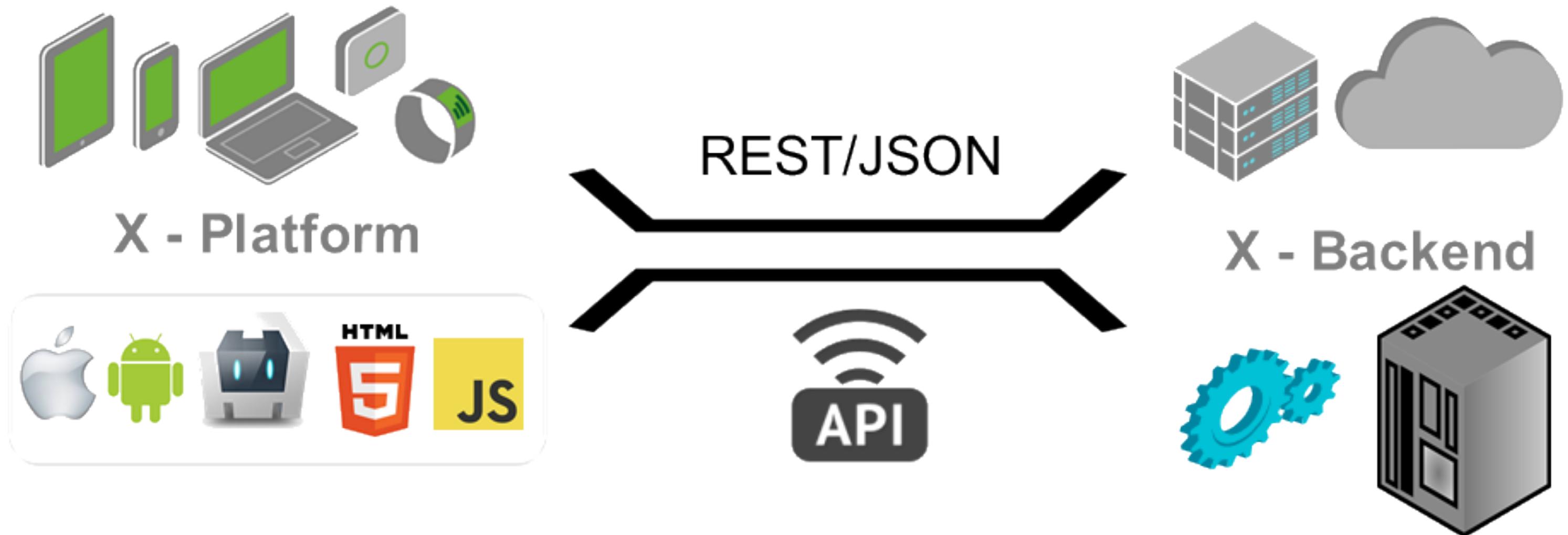
- >> Backbone
- >> Angular (e.g., M.E.A.N)
- >> Ember
- >> React
- >> MV*

Server-side Rendering

- >> Jade
- >> Handlebars
- >> EJS
- >> Hogan

Many more: <http://garann.github.io/template-chooser>

Node for SOA / REST



So what is
ECMAScript?

ES as a Language Specification

- » Browser implementations (like Chrome's V8)
- » Node builds on V8 with C++

Disclaimer: This is NOT a course on JavaScript. Please invest a few days in learning its fundamentals:

- » Eloquent JavaScript
- » JavaScript The Good Parts
- » ES6
- » JavaScript Ninja
- » Functional JavaScript

ES6 Sidenote

- » Destructuring
- » Const and let
- » Functions
- » Interpolations

More: [Top 10 ES6 Features Every Busy JavaScript Developer Must Know](#)

Demo

code/node/lang

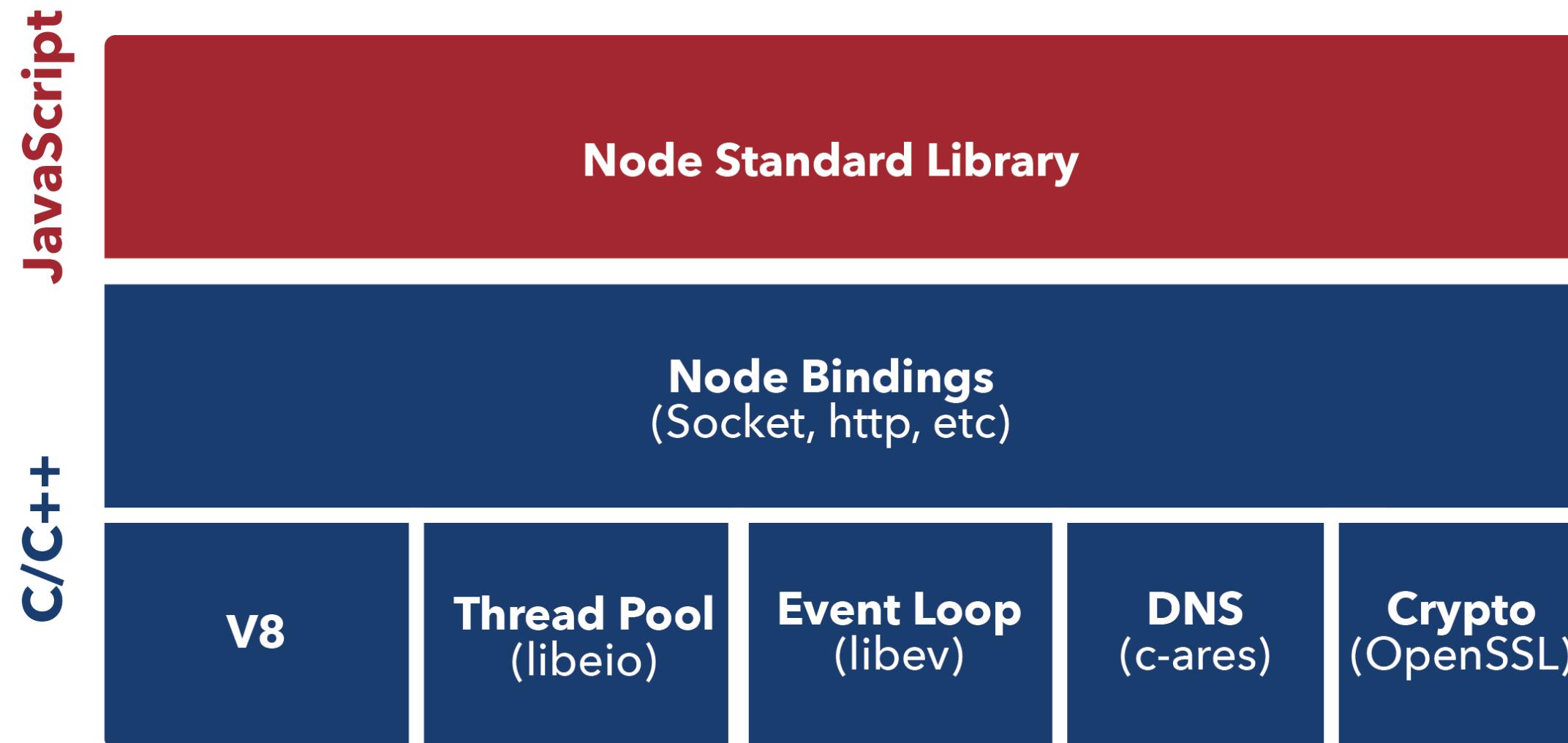
Browser JS != Node

- >> Modules
- >> Scopes
- >> window vs. global and process
- >> fs and other modules

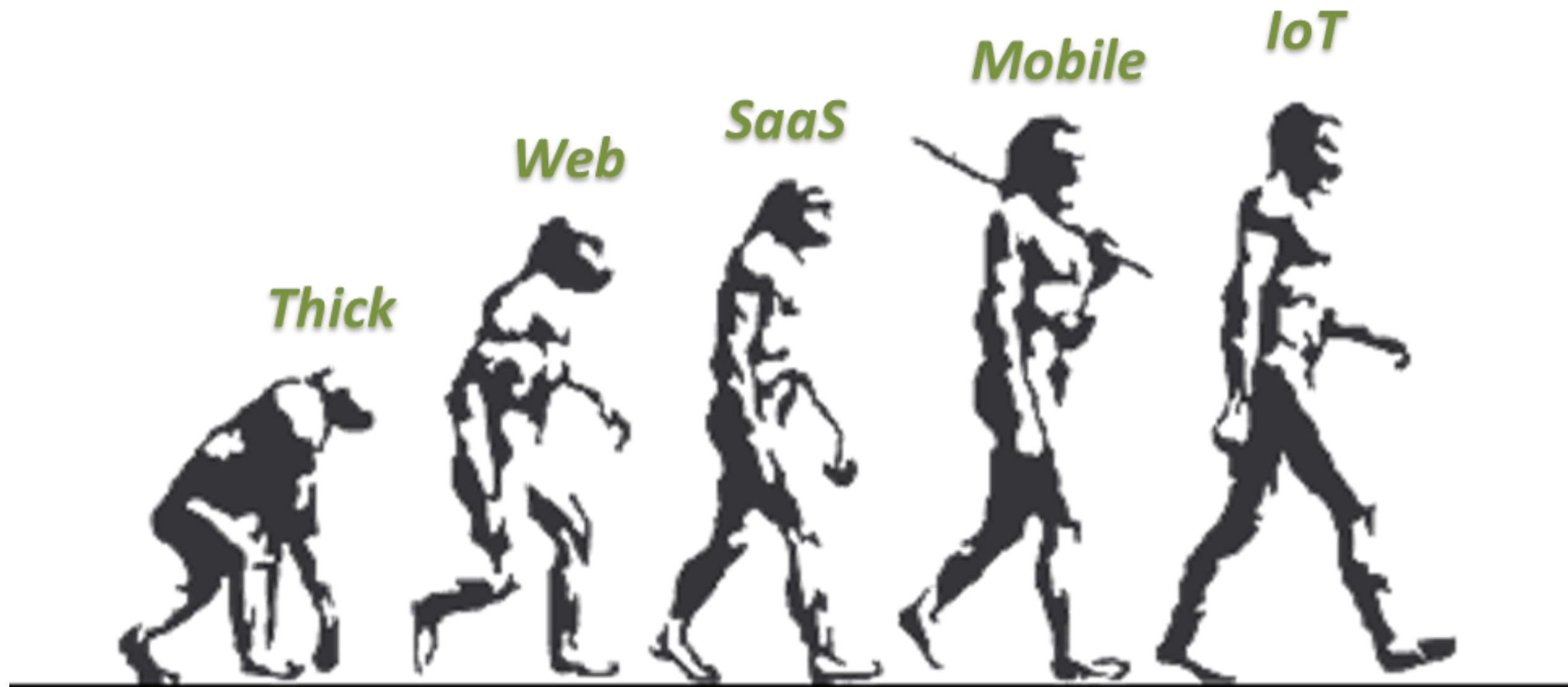
Node Core: V8, libev, and libeio

- » Libev: The event loop
- » LibEio: Async I/O
- » LibUv: Abstraction on libEio, libev, c-ares (for DNS) & ioCP (for Windows)

Node Core Architecture



Patterns evolve to serve market needs



Framework Categories

- » KISS Servers: small core, small modules
- » Convention: follow the leader, steep learning curve
- » Configuration: open path, manual effort for advanced
- » ORM & Isomorphic: model-driven, shared code, steep learning

Framework Examples

- >> KISS Servers: Node core
- >> Convention: Restify, Total.js
- >> Configuration: Hapi, Kraken, Express
- >> ORM & Isomorphic: LoopBack, Sails, Meteor*

"Everything is a callback... in Node everything is non-blocking and so it doesn't allow you to just sit there and then return the response. "

— Ryan Dahl <http://www.infoq.com/interviews/node-ryan-dahl>

Node Language

Node is not JavaScript, but both JavaScript and Node are ECMAScript.

Demo

Ways to run Node:

1. Node REPL
2. Node Eval (node -e)
3. Node Command
4. Other tools and scripts, e.g., bash script, Makefile, nodemon, node-dev, supervisor, pm2, etc.

global Object

The global object is similar to window object in the browser
JavaScript.

global.process same as process

global.console same as console

global.require same as require

global.module.exports same as module.exports

process Object

In Node, interaction with the current Node process takes place via the `process` object.

Environment Variables

Environment variables can be accessed via the `env` attribute:

```
console.log(process.env)
```

```
{ SHELL: '/bin/bash',
  USER: 'jordan',
  HOME: '/home/jordan',
  ...
}
```

Command-Line Arguments

Shell commands accept arguments to alter their behavior:

```
$ echo "Hello World!" // "Hello World" is the argument here
```

Node makes these arguments accessible via `process.argv`

Command-Line Options

The `argv` property is an array.

The first two elements are 'node' and the application's name:

```
$ node app.js arg1 arg2 arg3=val3
```

```
/*
process.argv => [
  'node', 'app.js', 'arg1',
  'arg2', 'arg3=val3'
]
*/
```

Exiting a Process

To exit a process, use the `exit` function

`process.exit()`

Exiting with Errors

Exit codes can also be specified

```
// this process exits successfully  
process.exit(0)
```

```
// this process failed  
process.exit(1)
```

```
// this process failed with a different code  
process.exit(129)
```

Exiting with Errors

Note:

- » Different failure codes can be used to differentiate types of failure
- » Knowing how an application failed allows the developers the means to program an appropriate response

Modules

Modules in Browser

Don't exist natively until ES6, i.e., no built-in module support!

Modules in Browser Workarounds

>> <script>

>> CommonJS

>> AMD (requirejs)

>> ES6

Module Loaders in Browser

- » SystemJS
- » RequireJS
- » Browserify
- » es6-module-loader

More info: <http://mzl.la/1Ieu8zM> and <http://mzl.la/1Ieu7vz>

Modules in Node

Built-in modules with require a CommonJS notation! 

Node Require Example:

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

Requiring Modules

Modules can live different places with JavaScript. They can be on local machines, virtual machines, servers, remote URI locations, or anywhere really.

Loading Node Modules

These modules can be loaded with module loaders like require or via inversion of control patterns.

```
var filesystem = require('fs'),  
    databaseConfigs = require('./configs/database.json'),  
    routes = require('../routes'),  
    server = require('./boot/server.js')
```

Creating a Module with a "Class" Example:

```
function UserController() {  
  var username, password  
  function doLogin(user, pw) {  
    username = user;  
    password = pw  
    // do the rest of the login work  
  }  
  var publicAPI = {  
    login: doLogin  
  }  
  return publicAPI  
}  
  
// create a `UserController` instance  
var ctrl = UserController()  
ctrl.login( "fred", "12Battery34!" )
```

Node Patterns for Module Exports

```
>> module.exports = function(ops) {...}
```

```
>> module.exports = {...}
```

```
>> exports.methodA = function(ops) {...}
```

```
>> exports.obj = {...}
```

Function Pattern

module.js:

```
module.exports = function(options) {
  var limit = 100
  if (options.type === 'foobar') {
    limit = 200;
  }
  return {
    name: 'request',
    limit: limit,
    type: options.type,
    method: function(data) { return data; }
  }
}
```

Functional Pattern

main.js:

```
var mod = require('./module.js')
var request = mod({
  type: 'foobar'
})
request.method({
  x: 10,
  y: 20
})
```

Demo

>> code/node/module-basic

>> code/node/module-greetings v2 and v3

Observer Pattern

```
var Job = function() {}

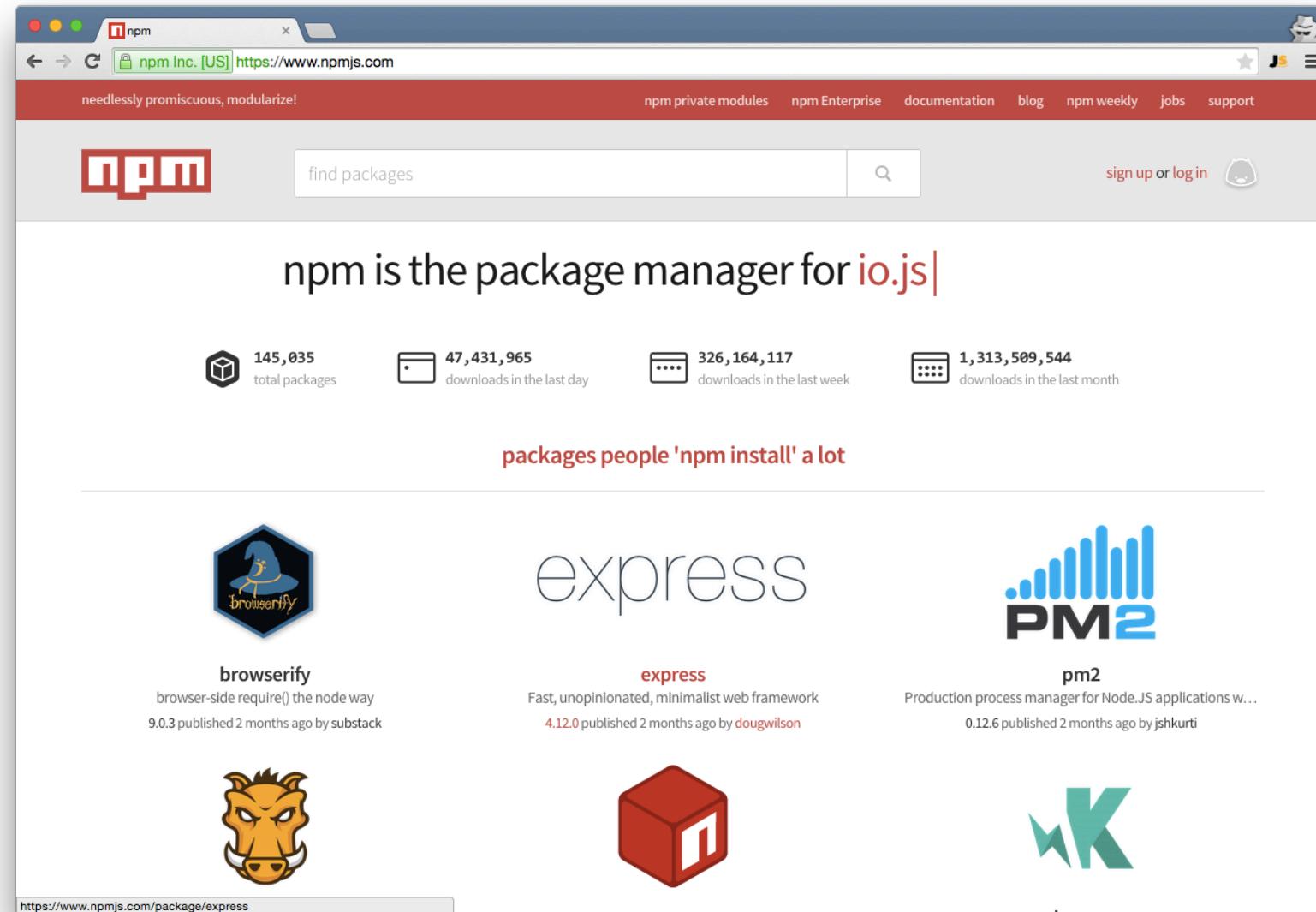
require('util').inherits(Job, require('events').EventEmitter)
job = new Job()

job.on('done', function(timeDone){
  console.log('Job was pronounced done at', timeDone)
})

job.emit('done', new Date())
job.removeAllListeners()
```

npm

Meet the beast!



What is npm?

A package manager for Node.

- » Website: <https://www.npmjs.com/>
- » Command-line tool: npm
- » Registries: public and private

Introduction to NPM

Two ways to install a module:

>> Locally: most of your projects' dependencies, e.g., express, request, hapi

```
npm install module-name
```

>> Globally: command-line tools only (mostly), e.g., mocha, grunt, slc

```
npm install -g module-name
```

Installing packages

```
$ npm install express  
$ npm install express@4.2.0  
$ npm install express@latest  
$ npm install express --save  
$ npm install mocha --save-dev  
$ npm install grunt -g  
$ sudo npm install grunt -g
```

Package.json

Creating package.json: Run init action to interactively create a package.json

```
$ npm init
```

```
This utility will walk you through creating a package.json
file. It only covers the most common items, and tries to
guess sane defaults.
```

```
See `npm help json` for definitive documentation on these
fields and exactly what they do.
```

```
Use `npm install <pkg> --save` afterwards to install a package
and save it as a dependency in the package.json file
```

```
Press ^C at any time to quit
name: (my-package-name)
```

Package.json

```
{  
  "name": "my-cool-app",  
  "version": "0.1.0",  
  "description": "A gret new application",  
  "main": "server.js",  
  "dependencies": {  
    "express": "~4.2.0",  
    "ws": "~0.4.25"  
  },  
  "devDependencies": {  
    "grunt": "~0.4.0"  
  }  
}
```

npm

When running `npm install NAME` in a folder:

- >> npm looks for `node_modules` or `package.json`
- >> if nothing is found it goes up the tree

Therefore, in an empty folder, create `package.json` or `node_modules` dir first.

Demo



Sample Code:

code/node/11-npm-cli.txt

Public Modules & Registries

Set config values...

```
$ npm set init.author.name "Your Name"
```

```
$ npm set init.author.email "you@example.com"
```

```
$ npm set init.author.url "http://yourblog.com"
```

Sign up on the npm website and add yourself:

```
$ npm adduser
```

Publishing your module

Add package.json (maybe try npm init)

Then Publish!

```
$ cd my-cool-app
```

```
$ npm publish
```

Private Registries

- » Hosted by npmjs
- » Hosted by you

Advantages of private:

Code is not exposed to outside and no external dependencies (if self-hosted)

(There are strategies for deployment, e.g., tar file)

To list currently installed npm modules, use the `ls` action
`ls` lists out modules local to the current Node project

```
$ npm ls  
/home/johndoe/node-app  
|__ q@1.0.1
```

To list globally installed modules, add the `-g` flag

```
$ npm ls -g  
/usr/lib  
|__ bower@1.3.11  
    |__ abbrev@1.0.5  
    |__ archy@0.0.2  
    |__ semver@4.0.0
```

Remove a module

To remove an npm module

```
$ npm rm mysql
```

To remove a global module

```
$ npm rm mysql -g
```

Packaging

Module packaging in Node is done using a package.json file
There are many options that can be configured:

- >> name
- >> version number
- >> dependencies
- >> etc

Private Modules

The private attribute prevents accidental publishing

```
{  
  "name" : "my-private-module",  
  "version": "0.0.1",  
  ...  
  "private": true,  
  ...  
}
```

When to use -g?

A: Only for command-line tools. They usually have bin in package.json:

```
{  
  "name": "stream-adventure",  
  "version": "4.0.4",  
  "description": "an educational stream adventure",  
  "bin": {  
    "stream-adventure": "bin/cmd.js"  
  },  
  "dependencies": {  
    ...  
  }
```

Hello World

Web Content

Types of web content

- >> Static
- >> Dynamic

Node is mostly for Dynamic content

The server, request, response objects

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'})
  res.end('Hello World\n')
}).listen(1337, '127.0.0.1')

console.log('Server running at http://127.0.0.1:1337/')
```

Running the App

Run with:

```
$ node server.js
```

Send requests:

```
$ curl http://localhost:1337
```

Or

<http://localhost:1337>

HTTP Object

Http object:

<https://nodejs.org/api/http.html>

https://nodejs.org/api/http.html#http_class_http_server

```
var server = http.createServer([requestListener])  
server.listen(port[, hostname][, backlog][, callback])
```

HTTP Response

```
response.writeHead(200, {  
  'Content-Length': body.length,  
  'Content-Type': 'text/plain' })
```

Demo

Sample Code (code/node):

1. code/node/08-nodejs-app.js
2. code/node/09-nodejs-app2.js
3. code/node/static-app-project

Testing

assert:

```
const assert = require ('assert')
```

Chai Should

```
const should = require('chai').should()
```

code/node/escape.test.js

Starting learnyounode

Install:

```
$ sudo npm install learnyounode -g
```

Start:

```
$ learnyounode
```

Verifying learnyounode

Verify solution with:

```
$ learnyounode verify program.js
```

learnyounode Workshop

1. Pick the first problem
2. Read instructions
3. Solve the problem (e.g., create `program.js`)
4. Verify
5. Pick the next problem