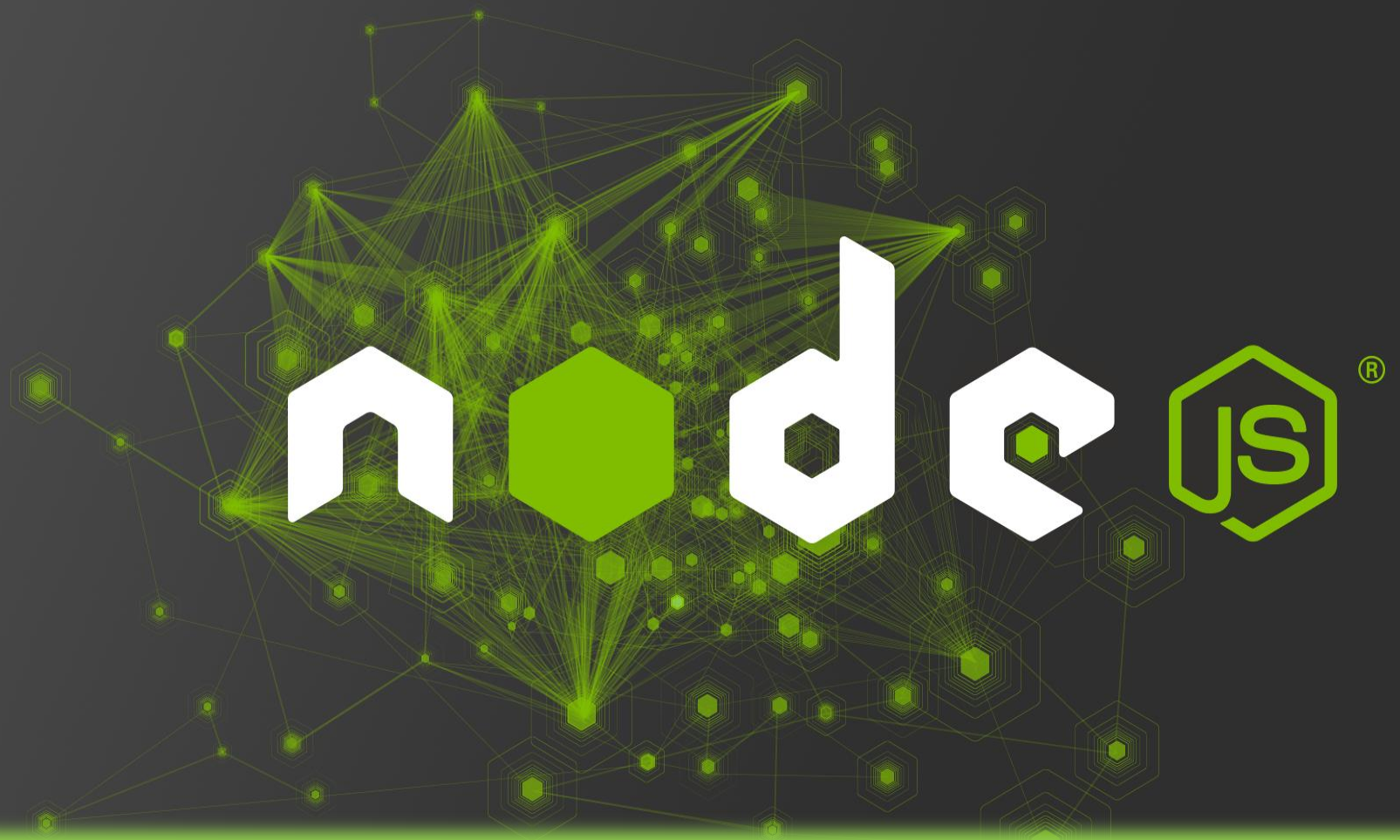


Learning server side JavaScript

Eman Fathi

Conents

Day1 : Installation ,Nodejs structure and Module Basics

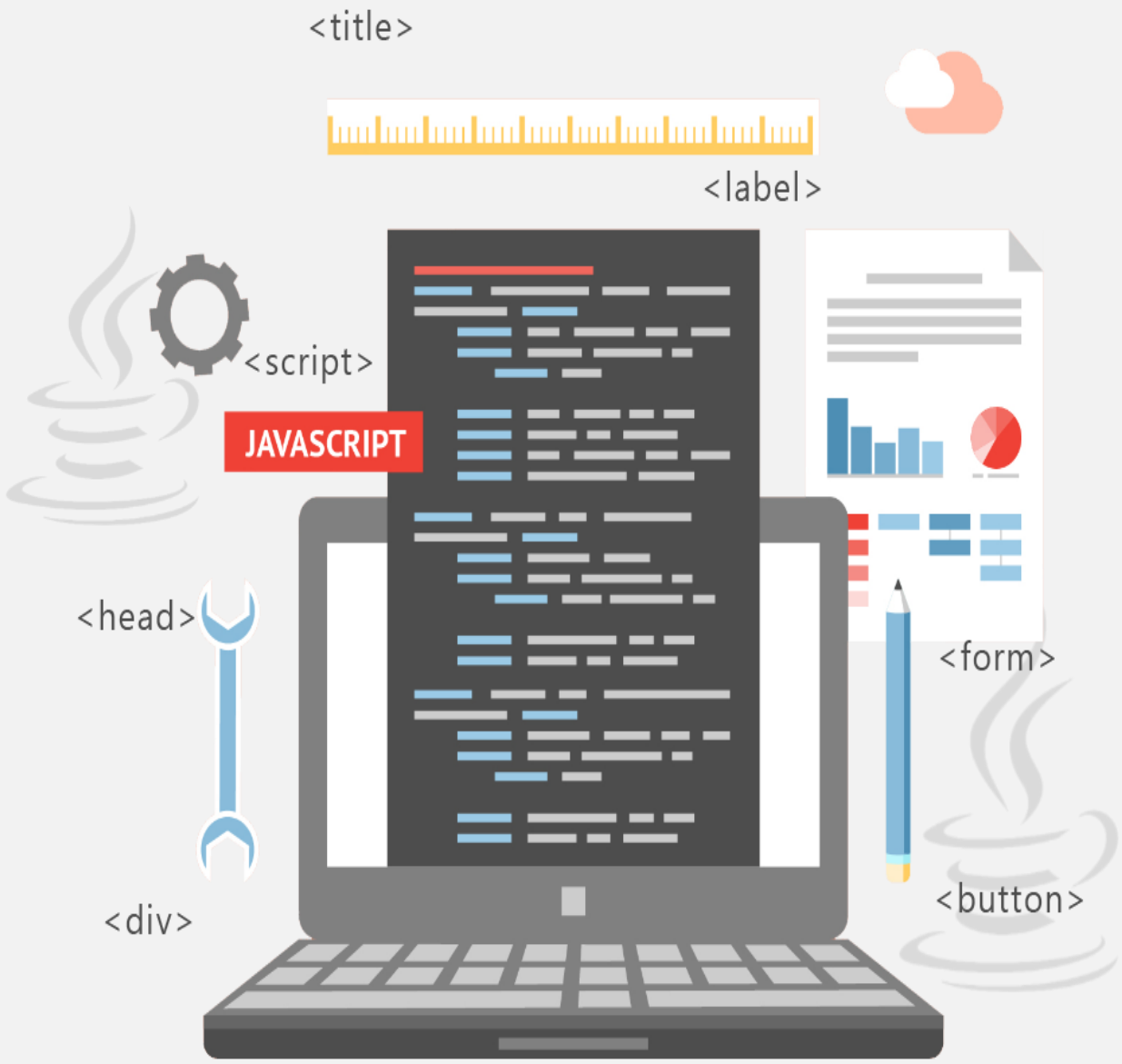
Day 2 : Express web framework with MVC (REST API)

Day 3 : connect to mongo Database with mongoose

Day 4 : Uploading Images and JWT Authentication

Day 5 : Authorization

Day 6: Views



Server Side JavaScript

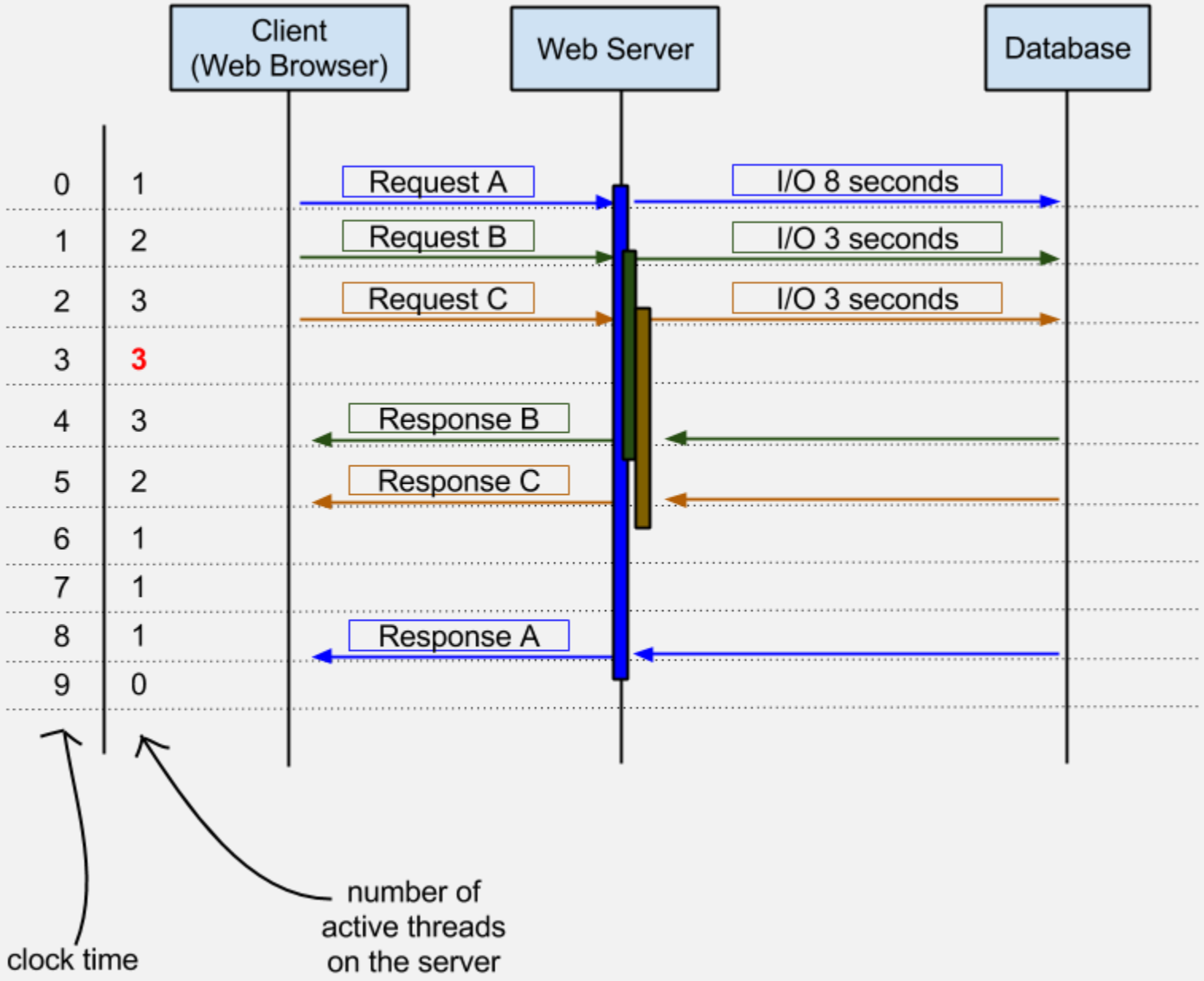
"What's a browser language doing on the server?".

JavaScript every where

JavaScript is a programming language, just like any other language, and the better question to ask is

"Why should JavaScript remain trapped inside browsers?"


Ryan Dahl
Nod.js creator 2009
Software Engineer @ Joyent



Multi-Threading Server

V8 Google chrome JavaScript Engine

V8 is Google's open source high-performance JavaScript engine, written in C++.

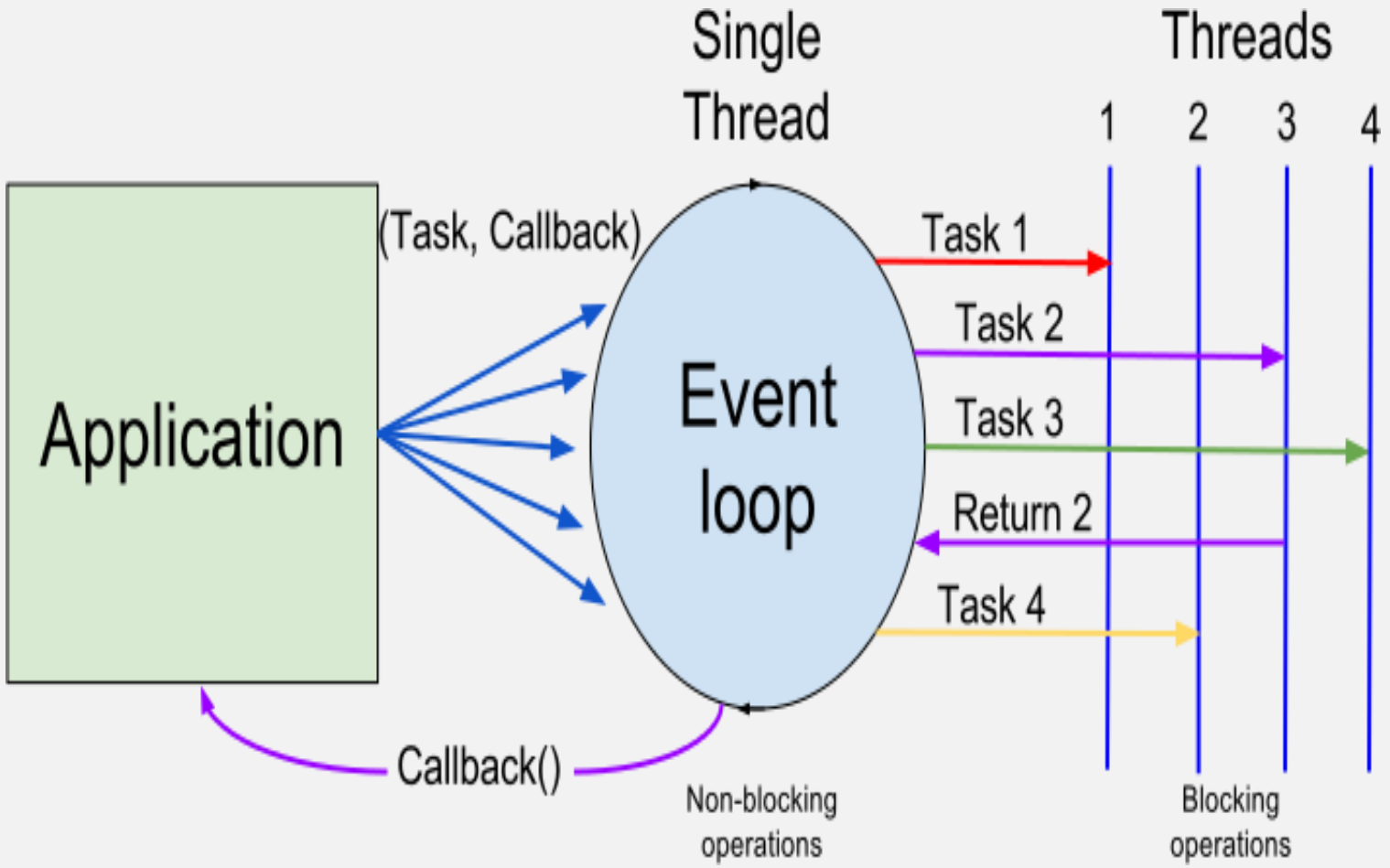
It compiles the JavaScript code into machine code at execution by implementing a JIT compiler .

Usage:

1- chrome browser

2- V8 is the core of node.js

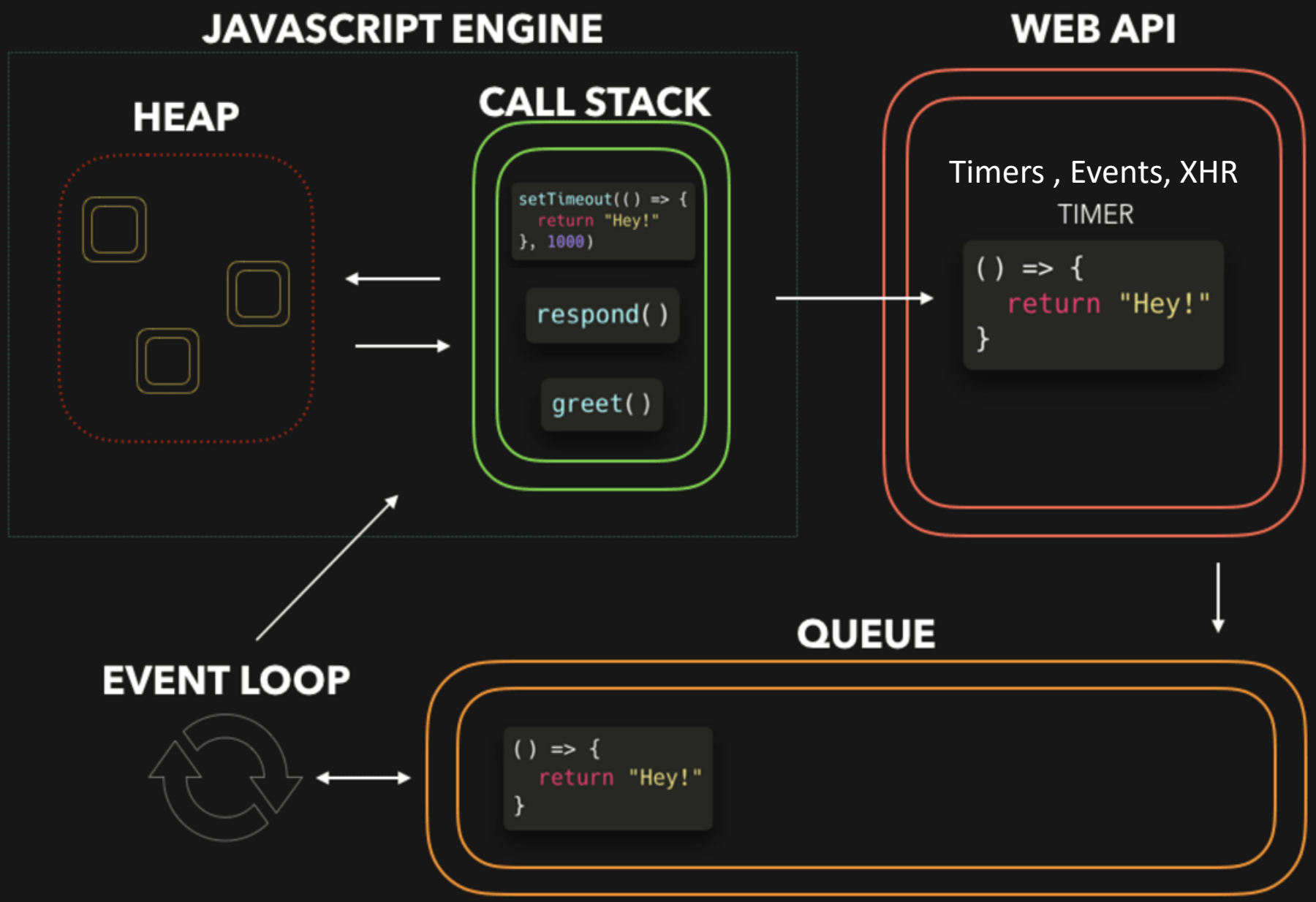
3-Underlying component for ATOM and Visual Studio code text editors



Node.js Single-Thread Server

Sever side JavaScript is:
Single-Threaded
Non-blocking
Asynchronous Language

- ✓ JavaScript is synchronous (one process at a time)
- ✓ JavaScript can run as asynchronous :
  - Callback functions
  - Promises
  - Async/await




```
function greet() {
  return "Hello!"
}

function respond() {
  return setTimeout(() => {
    return "Hey!"
  }, 1000))
}

greet()
respond()
```

Nodejs Creation

- ✓Sever side JavaScript
- ✓Built on Google's V8 JavaScript Engine
- ✓Single-threading, Event loop, non-blocking IO System
- ✓CommonJs Module System
- ✓Written in C++ & javascript

NodeJs Creation

2009: Node.js was originally written by Ryan Dahl
supported only Linux and Mac OS

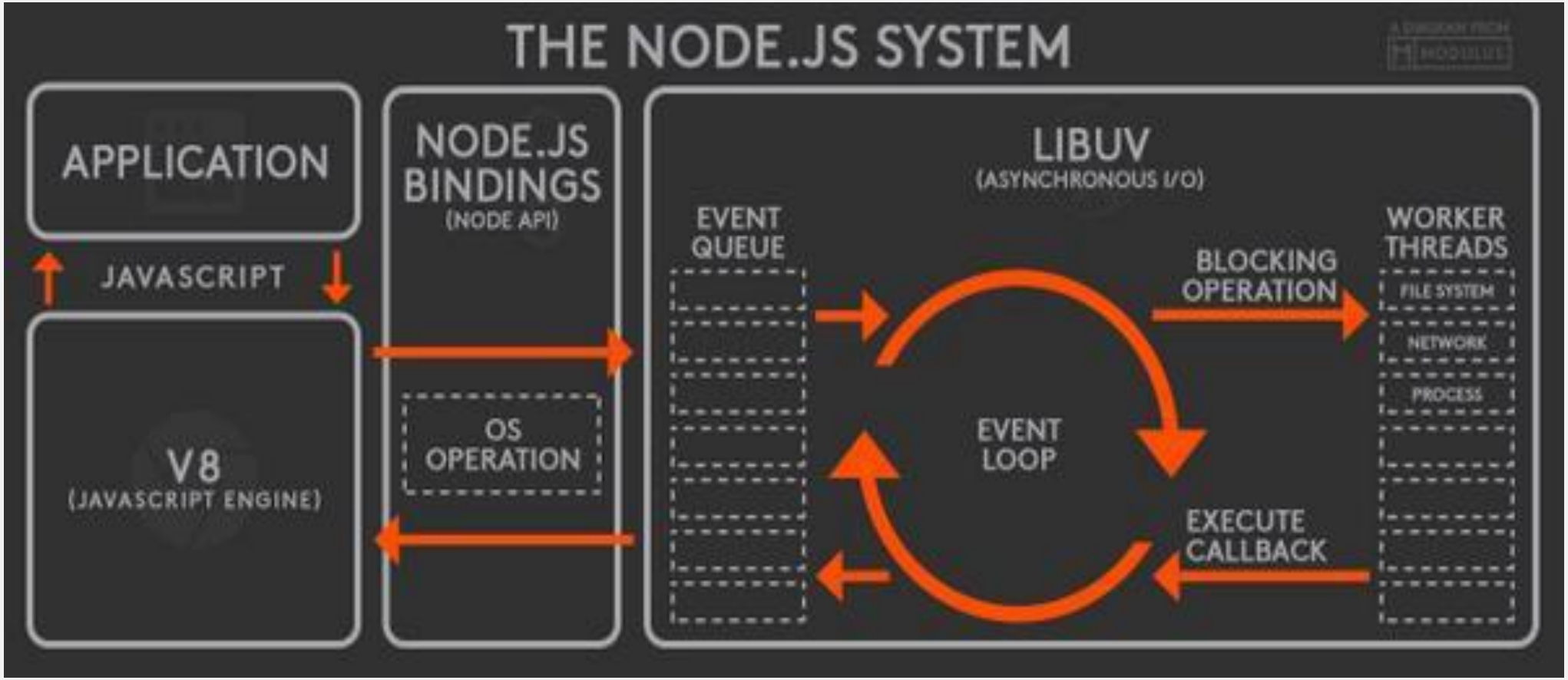
2011: Node Package Manager (NPM) created
and first windows version of nodejs released

2013: MEAN stack by *Valeri Karpov*

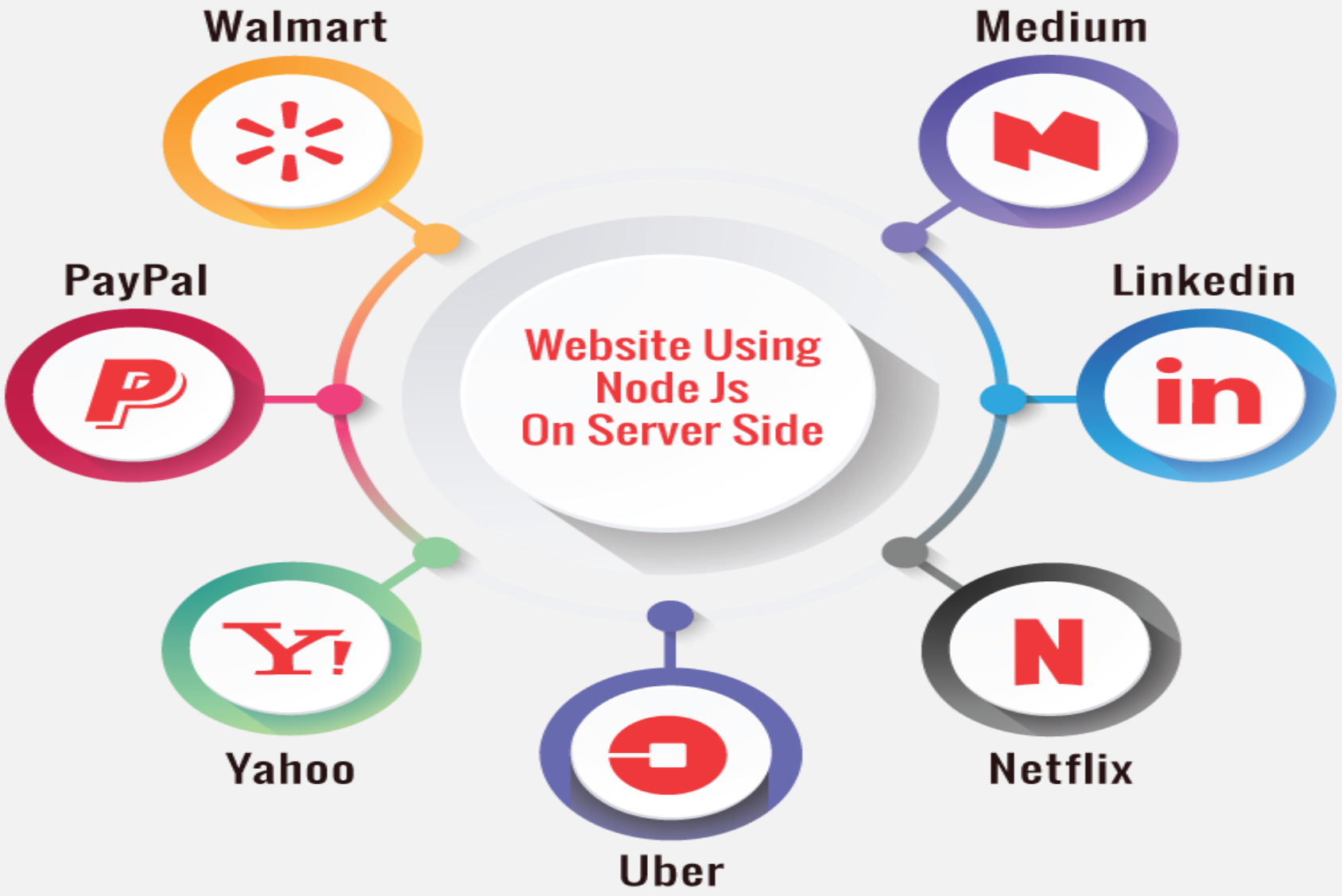
2014 : io.js project forked

2015: Node.js 4.0 released including ES6 features

LIBUV


THE NODE.JS SYSTEM
APPLICATION
JAVASCRIPT
V8
(JAVASCRIPT ENGINE)
NODE.JS
BINDINGS
(NODE API)
OS
OPERATION
LIBUV
(ASYNCHRONOUS I/O)
EVENT
QUEUE
EVENT
LOOP
BLOCKING
OPERATION
EXECUTE
CALLBACK
WORKER
THREADS
FILE SYSTEM
NETWORK
PROCESS

Sever side JavaScript is Single-Threaded Non-blocking Asynchronous Language

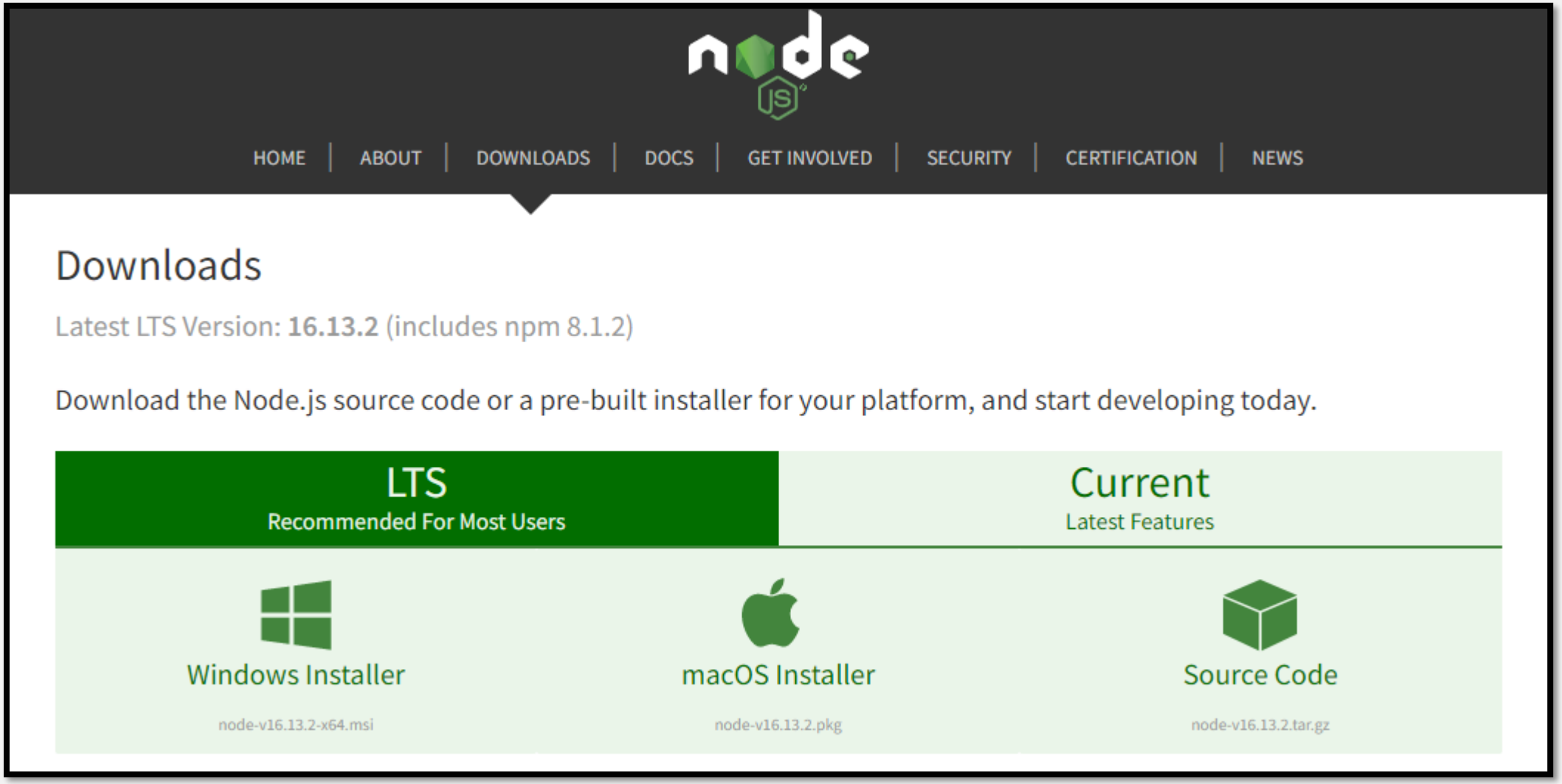

Walmart
Medium
PayPal
Website Using
Node Js
On Server Side
Linkedin
Yahoo
Uber
Netflix

What Does JavaScript need to run as Server?

- ✓Organize code into reusable code.
- ✓Dealing with the requests over the internet .
- ✓Ability to accept request and send responses back to the client.
- ✓Ability to deal with files(html,xml..)
- ✓Ability to deal with databases.

Nodejs Server Installation

Installing Node.js



Running Windows power Shell (cmd)

```
C:\Users\ITI>node --version
v16.13.1

C:\Users\ITI>npm --version
8.1.2
```

Node.js REPL

Node.js comes with virtual environment called **REPL** (Node shell). REPL stands for **R**ead-**E**val-**P**rint-**L**oop. It is a quick and easy way to test simple Node.js/JavaScript code.

Node.js REPL important commands

Command	Description
.save filename	Save current Node REPL session to a file.
.load filename	Load the specified file in the current Node REPL session.
ctrl + c	Terminate the current command.
ctrl + c (twice)	Exit from the REPL.
ctrl + d	Exit from the REPL.
.break	Exit from multiline expression.
.clear	Exit from multiline expression.

Node.js Global Objects

Objects that are available globally on the **global** namespace. We can use those objects immediately on JavaScript file without requiring anything.

- ✓ console
- ✓ __dirname
- ✓ __filename
- ✓ exports
- ✓ module
- ✓ setInterval
- ✓ setTimeout
- ✓ process

Node.js Global Objects

In browsers, the top-level scope is the global scope. This means that within the browser **var X** will define a new global variable. In Node.js this is different. The top-level scope is not the **global** scope; **var X** inside a Node.js module will be **local** to that module.

```
console.log("Hello") === global.console.log("Hello"); //true

let instructorName="Eman"; //local to the module
course="NodeJs"; //added to global object

console.log(global.course); //"NodeJs"
console.log(global.instructorName); //undefined
```

Node.js Global Objects

One important object that is available globally is **process** object. It contains functionality that allows dealing with current process instance (get environment information, environment variables, communicate with terminal or parent processes) .
By using process object , any information can be available when application starts.

```
process.env ;// all environment information
process.env.lang ;// 'en_US.UTF-8'
process.exit(-1) //exit current process
```

- ✓Organize code into reusable code.
- ✓Dealing with the requests over the internet .
- ✓Ability to accept request and send responses back to the client.
- ✓Ability to deal with files(html,xml..)
- ✓Ability to deal with databases.

NodeJs Modules

Node.js Modules

Before writing Node.js applications, you must learn about Node.js **modules** and **packages**. Modules and packages are the building blocks for breaking down your application into smaller pieces. Every JavaScript file used in Node.js is itself a module

"Modules are the basic building block for constructing Node.js applications."

Node.js's module implementation is strongly inspired by, but not identical to, the **CommonJS** module specification.

Node.js does not support ES2015 modules but **Babel** supports transpiling ES2015 modules to Node.js modules. Starting with version **8.5.0**, Node.js supports ES modules natively, behind a command line option.

Defining a module

There are essentially two elements to interact with modules: **require** and **exports**.

The **require** function searches for modules, loading the module definition into the Node.js runtime, and makes its functions available. Anything assigned to a field of the **exports** object is available to other pieces of code, and everything else is hidden. The exports object is what's returned by **require('./module')**.

Defining a module

```
let ProjectId=30;
let projectDuration="20 Weeks";
exports.projectDuration= projectDuration;
exports.projectSupervisor="Eman fathi";
exports.projectData=function(){ return {ProjectId} }
```

Module1.js

```
let module1Values=require("./module1");
let InstructorName=module1Values.projectSupervisor;
let ProjectNumber=module1Values.projectDuration;
module1Values.projectData();
```

Module2.js

node.js™

+

npm

npm – Node Package Manager

Its' purpose is publishing and distributing Node.js packages over the Internet using a simple command-line interface. With npm, you can quickly **find** packages to serve specific purposes, **download** them, **install** them, and **manage** packages you've already installed.

The **npm package** defines a **package format** for Node.js largely based on the CommonJS package specification. It uses the same **package.json** file that's supported natively by Node.js, but with additional fields to build in additional functionality.

Website :https://npmjs.org

NPM, Inc. [US] | https://www.npmjs.com

Not Parents' Money

npm Enterprise Features Pricing Docs Support

npm

Search packages

log in or sign up

Build amazing things

npm is the package manager for JavaScript and the world's largest software registry. Discover packages of reusable code — and assemble them in powerful new ways.

Sign up

Installing an npm package

In Dependencies:

npm install express –save

In DevDependencies:

npm install express –save-dev

The named module is installed in node_modules in the current directory.

Globally:

If you instead want to install a module globally, add the -g option:

npm install noedmon -g

The npm package format – Package.json

An **npm** package is a directory structure with a **package.json** file describing the package.

A basic package.json file is as follows:

```
{  "name": "packageName",
   "version": "1.0",
   "main": "mainModuleName.js",
   "scripts": {
              "start": "node mainModuleName.js",
              }
}
```

The file is in JSON format, which, as a JavaScript programmer, you should be familiar with.

Module identifiers and path names

There are three types of module identifiers: relative, top-level and core

- ✓ **Relative module identifiers:** These begin with ./ or ../ That is, a module identifier beginning with ./ is looked for in the current directory; whereas, one starting with ../ is looked for in the parent directory.
- ✓ **Top-level module identifiers:** These begin with none of those strings and are just the module name. These must be stored in a node_modules directory, and the Node.js runtime has a nicely flexible algorithm for locating the correct node_modules directory.
- ✓ **Core modules :** exists where nodejs installed

Thank You!