Node.js from zero to hero

Nicola Del Gobbo – Mauro Doganieri

WHAT IS **Node.js**?

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

Node.js uses an event-driven, **non-blocking I/O** model that makes it lightweight and efficient

Node.js' package ecosystem, **npm**, is the largest ecosystem of open source libraries in the world

A **JavaScript Runtime**

Asynchronous I/O based on **event loops**

~100k LOC of **JS** and **C++**

Node **glue**

Ecosystem of **packages**

WHAT CAN I DO WITH **Node.js**?

EVERYTHING

Great thank you to Node.js **Community** and **Working Groups**

Packages	Downloads · Last Day	Downloads · Last Week
603.284	201.263.400	3.469.805.066

WHAT CAN I DO WITH **Node.js**?

express

fastify

CLI APPLICATION

DISRTRUBUTED SYSTEM

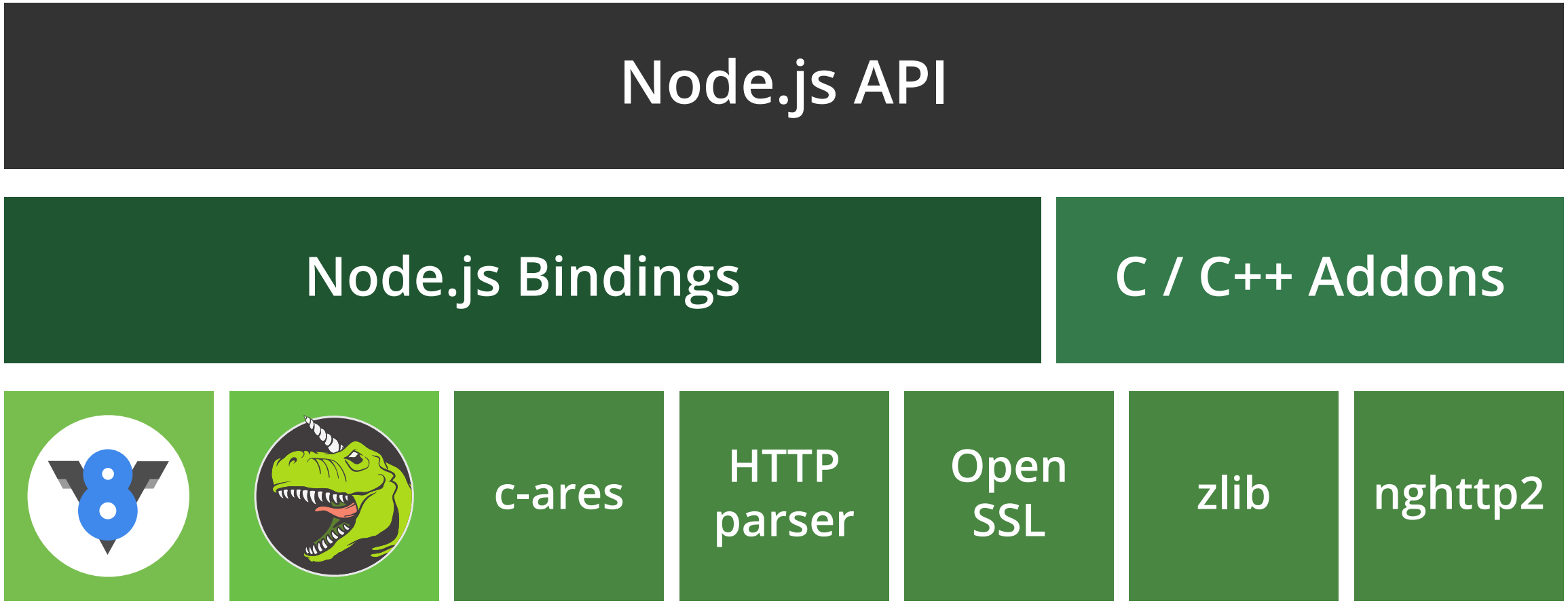
MQTT.js

socket.io

REAL TIME SYSTEM

DESKTOP APPLICATION

ARCHITECTURE



CONCURRENCY MODEL?

ASINCHRONOUS I/O

SINGLE THREADED

SINCHRONOUS MODEL?

```
var result = db.query('select ...');
// use result
```

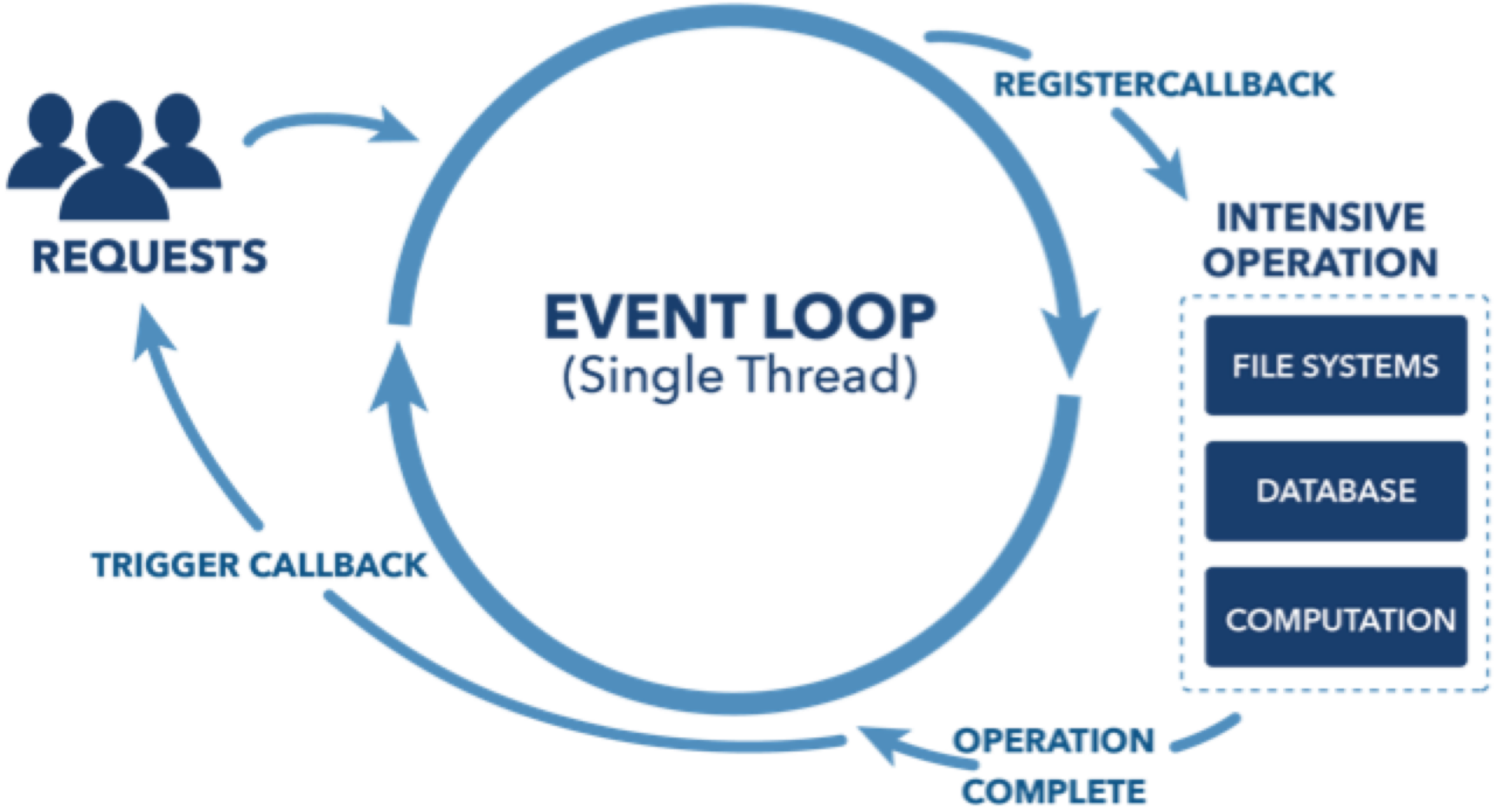
Blocking the whole process or you need to have **multiple execution stacks**

ASINCHRONOUS MODEL?

```
db.query('select ...', function (result) {
    // use result
})

// Execution continue
console.log('Do other stuff ...')
```

The **main** process is **never blocked**. No strategy is required to handle competing requests

EVENT **LOOP**


REQUESTS
EVENT LOOP
(Single Thread)
REGISTERCALLBACK
INTENSIVE OPERATION
FILE SYSTEMS
DATABASE
COMPUTATION
OPERATION COMPLETE
TRIGGER CALLBACK

ASINCHRONOUS IN LOW LEVEL

```
'use strict'

const gs = require('ghostscript4js')

let cmd = '-sDEVICE=pngalpha -o my.png -sDEVICE=pngalpha -r144 my.pdf'
gs.execute(cmd, function (err) {
  if (err) {
    console.log("Ooops... something wrong happened")
  }
})
```

```
class GhostscriptWorker : public AsyncWorker
{
  public:
    GhostscriptWorker(Callback *callback, string RAWcmd)
        : AsyncWorker(callback), RAWcmd(RAWcmd), res(0) {}
    ~GhostscriptWorker() {}

    void Execute()
    {
        res = 0;
        vector<string> explodedCmd;
        istringstream iss(RAWcmd);
        for (string RAWcmd; iss >> RAWcmd;)
            explodedCmd.push_back(RAWcmd);
        int gsargc = static_cast<int>(explodedCmd.size());
        char **gsargv = new char *[gsargc];
        for (int i = 0; i < gsargc; i++)
        {
            gsargv[i] = (char *)explodedCmd[i].c_str();
        }
        try
        {
            GhostscriptManager *gm = GhostscriptManager::GetInstance();
            gm->Execute(gsargc, gsargv);
            delete[] gsargv;
            res = 0;
        }
        catch (exception &e)
        {
            delete[] gsargv;
            msg << e.what();
            res = 1;
        }
    }
```

This code will be executed on the Worker Thread

```
    void HandleOKCallback()
    {
        Nan::HandleScope();
        Local<Value> argv[1];
        if (res == 0)
        {
            argv[0] = Null();
        }
        else
        {
            argv[0] = Error(Nan::New<String>(msg.str()).ToLocalChecked());
        }
        callback->Call(1, argv);
    }
```

When execute method end its computation the resuls will be reassembled and passed to JS context

```
  private:
    string RAWcmd;
    int res;
    stringstream msg;
};
```

```
NAN_METHOD(Execute)
{
    Callback *callback = new Callback(info[1].As<Function>());
    Local<String> JScmd = Local<String>::Cast(info[0]);
    AsyncQueueWorker(new GhostscriptWorker(callback, *String::Utf8Value(JScmd)));
    info.GetReturnValue().SetUndefined();
}
```

Return immediately

ASINCHRONOUS IN LOW LEVEL

```
'use strict'

const gs = require('ghostscript4js')

let cmd = '-sDEVICE=pngalpha -o my.png -sDEVICE=pngalpha -r144 my.pdf'
gs.execute(cmd, function (err) {
  if (err) {
    console.log("Ooops... something wrong happened")
  }
})
```

JavaScript

"The world's most misunderstood programming language"

Douglas Crockford

Strict mode

ECMAScript 5's strict mode is a way to *opt in* to a restricted variant of JavaScript. Strict mode isn't just a subset: it *intentionally* has different semantics from normal code.

Strict mode – how to

Strict mode for scripts

```
// Whole-script strict mode syntax
'use strict'
var v = 'Hi! I\'m a strict mode script!'
```

Strict mode for functions

```
function strict() {
  // Function-level strict mode syntax
  'use strict'
  function nested() {
    return 'And so am I!'
  }
  return 'Hi! I\'m a strict mode function! ' + nested()
}

function notStrict() {
  return 'I\'m not strict.'
}
```

Scope

Where your variables are actually created depends on

- how you declare them, using **var**, **let** or **const**
- if you are in *strict mode* or not
- if you are explicitly declaring them or not

Scope – **var** declaration and hoisting

- Declared variables are constrained in the execution context in which they are declared
- Declared variables are created before any code is executed

```
function test(condition) {
  // value exists here with a value of undefined
  if (condition) {
    var value = 7
    // value exists here with a value of 7
  } else {
    // value exists here with a value of undefined
  }
  // value exists here with a value of 7 or undefined based on condition
}

test(true)

// value is not defined outside of test (ReferenceError)
```

Scope – undeclared variables

- Undeclared variables are always global
- Undeclared variables do not exist until the code assigning to them is executed

```
function test() {
  // value doesn't exist here
  console.log(value) // ReferenceError
  value = 7
  // value exists here with a value of 7
}

test()

// value exists here with a value of 7
// value is global!
```

```
'use strict'

function test() {
  // Throws ReferenceError: value is not defined
  value = 7
}

test()
```

Scope – declaration using **let**

Variables declared by **`let`** have as their scope the block in which they are defined, as well as in any contained sub-blocks

```
function test(condition) {
  // value doesn't exist here
  if (condition) {
    let value = 7
    // value exists here with a value of 7
    if (condition) {
      // value exists here with a value of 7
    }
  }
  // value doesn't exist here
}

test(true)

// value is not defined outside of test
```

Scope – no redeclaration using **let**

Redeclaring the same variable within the same function or block scope raises a `SyntaxError`

```
let value = 5
let value = 3 // SyntaxError: Identifier 'value' has already been declared
```

Scope – redeclaration **var** vs **let**

```
{
  let value = 5
  {
    let value = 3 // different varible
    // value is equal to 3
  }
  // value is equal to 5
}
```

```
{
  var value = 5
  {
    var value = 3 // same variable
    // value is equal to 3
  }
  // value is equal to 3
}
```

Functions

In JavaScript, functions are first-class objects, because they can have properties and methods just like any other object

Functions – how to represent an object

```
function Person(firstName, lastName) {
  this.firstName = firstName
  this.lastName = lastName

  this.getName = function () {
    return this.firstName + ' ' + this.lastName
  }
}

let heisenberg = new Person('Walter', 'White')

heisenberg.getName() // Walter White
```

Functions – how to represent an object

```
function Person(firstName, lastName) {
  this.firstName = firstName
  this.lastName = lastName

  this.getName = function () {
    return this.firstName + ' ' + this.lastName
  }
}

let heisenberg = new Person('Walter', 'White')
heisenberg.profession = 'chemist'

Person.prototype.getProfession = function () {
  return this.profession
}

heisenberg.getProfession() // chemist
```

Functions – the **class** keyword

```
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName
    this.lastName = lastName
  }

  getName() {
    return this.firstName + ' ' + this.lastName
  }
}

let heisenberg = new Person('Walter', 'White')

heisenberg.getName() // Walter White
```

Functions – arrow functions

An **arrow function expression** has a shorter syntax than a function expression and does not have its own *this*, *arguments*, *super*...

These function expressions are best suited for non-method functions, and they cannot be used as constructors

Functions – arrow functions, an example

```
let materials = [
  'Hydrogen',
  'Helium',
  'Lithium',
  'Beryllium'
]

materials.map(function (material) {
  return material.length
}) // [8, 6, 7, 9]

materials.map((material) => {
  return material.length
}) // [8, 6, 7, 9]

materials.map(material => material.length) // [8, 6, 7, 9]
```

Functions - closures

A *closure* is the combination of a function and the lexical environment within which that function was declared

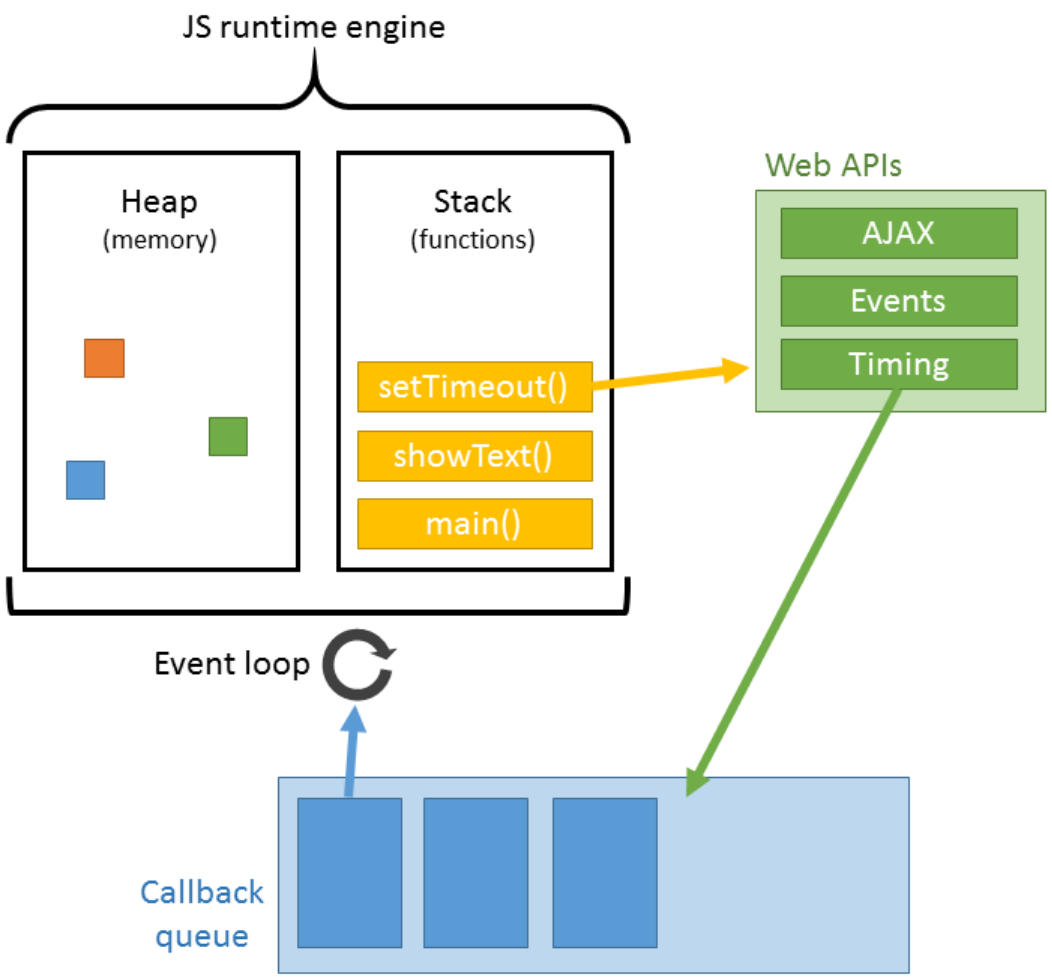
Functions – closures, an example

```
function makeAdder(x) {
  return function (y) {
    return x + y
  }
}

let add5To = makeAdder(5)
let add10To = makeAdder(10)

console.log(add5To(6)) // 11
console.log(add10To(3)) // 13
```

The long hard road out of *callback hell*

From *callbacks* to *promises* and *async/await*, trying to understand the *event loop*

The event loop



MODULARITY

CommonJS Module

ES6 Module from v8.5.0

“Se non hai provato Node.js non sai cos’è la **modularità** e il **riuso**”

Matteo Collina

MODULARITY

```
'use strict'

function Person (opts) {
    this._firstName = opts.firstName || ''
    this._lastName = opts.lastName || ''
}

Person.prototype.toString = function toString() {
    return `First name: ${this._firstName}
    Last name: ${this._lastName}`
}

module.exports = Person
```

```
'use strict'

function isPrime(p) {
    const upper = Math.sqrt(p)
    for(let i = 2; i <= upper; i++) {
        if (p % i === 0 ) {
            return false
        }
    }
    return true
}

exports.isPrime = isPrime
```

MODULARITY

```
'use strict'

const Person = require('./Person')

const p = new Person({
    firstName: 'Nicola',
    lastName: 'Del Gobbo'
})

console.log(p.toString())
```

```
'use strict'

const isPrime = require('./isPrime')

console.log(isPrime(3))
```

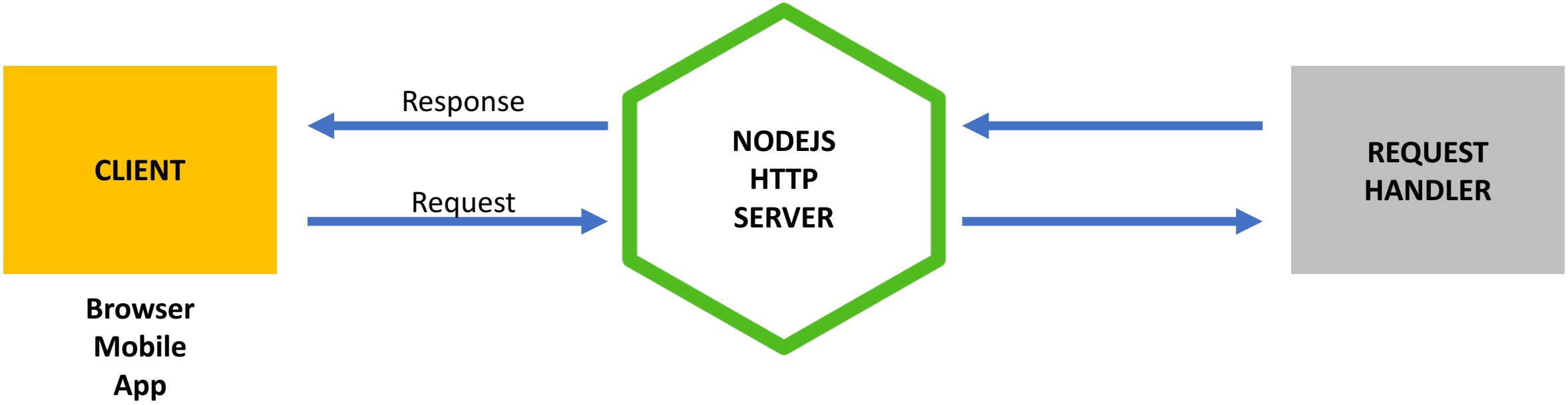
EVENT EMITTER

EventEmitter is a very important class in Node.js. It provides a **channel for events to be dispatched** and **listeners notified**. Many objects you will encounter in Node.js inherit from EventEmitter

EVENT EMITTER

```
'use strict'

const EventEmitter = require('events').EventEmitter

class User extends EventEmitter {

    constructor () {
        // do dome initialization tasks
    }

    addUser(user) {
        // register user on db
        // emit event
        this.emit('user:added', {userName: 'NickNaso', password: 'nodejs'})
    }

}

User.on('user:added', (data) => {
    // do something with data
})
```

WEB APPLICATION



http://your-web-application

~40k req/s

Hard to maintain

```
'use strict'

const http = require("http")

const url = require("url")

function onRequest(request, response) {

    let pathname = url.parse(request.url).pathname
    console.log("Request for " + pathname + " received.")

    if (pathname === "/start") {
        response.writeHead(200, { "Content-Type": "text/plain" })
        response.write("Hello")
        response.end()

    } else if (pathname === "/finish") {
        response.writeHead(200, { "Content-Type": "text/plain" })
        response.write("Goodbye")
        response.end()
    } else {
        response.writeHead(404, { "Content-Type": "text/plain" })
        response.end("404 Not Found")
    }
}


http.createServer(onRequest).listen(5000)


console.log("Server has started.")
```

Import http module

Listener for the incoming request

Route handler

FRAMEWORKS

express

EXPRESS

expressjs / express

Unwatch 1,736 | Unstar 34,685 | Fork 6,261

Code | Issues 104 | Pull requests 42 | Projects 0 | Wiki | Insights

Fast, unopinionated, minimalist web framework for node. https://expressjs.com

javascript nodejs express server

npm

Stats

788.780 downloads in the last day

4.310.178 downloads in the last week

18.740.809 downloads in the last month

EXPRESS

- Minimalist
- Unopinionated
- Fast (about **21k** req/sec)
- Simple (**do one thing well** philosophy from Unix world)
- Wrapper of **http** core module

```
'use strict'

const http = require("http")
const express = require('express')
```

Import and create

```
const app = express()
```

```
app.get('/start', (req, res) => {
    res.status(200).send('Hello')
})
```

Route handler

```
app.get('/finish', (re, res) => {
    res.status(200).send('Goodbye')
})

http.createServer(app).listen(5000)
console.log("Server has started.")
```

For web sockets bind express to http module

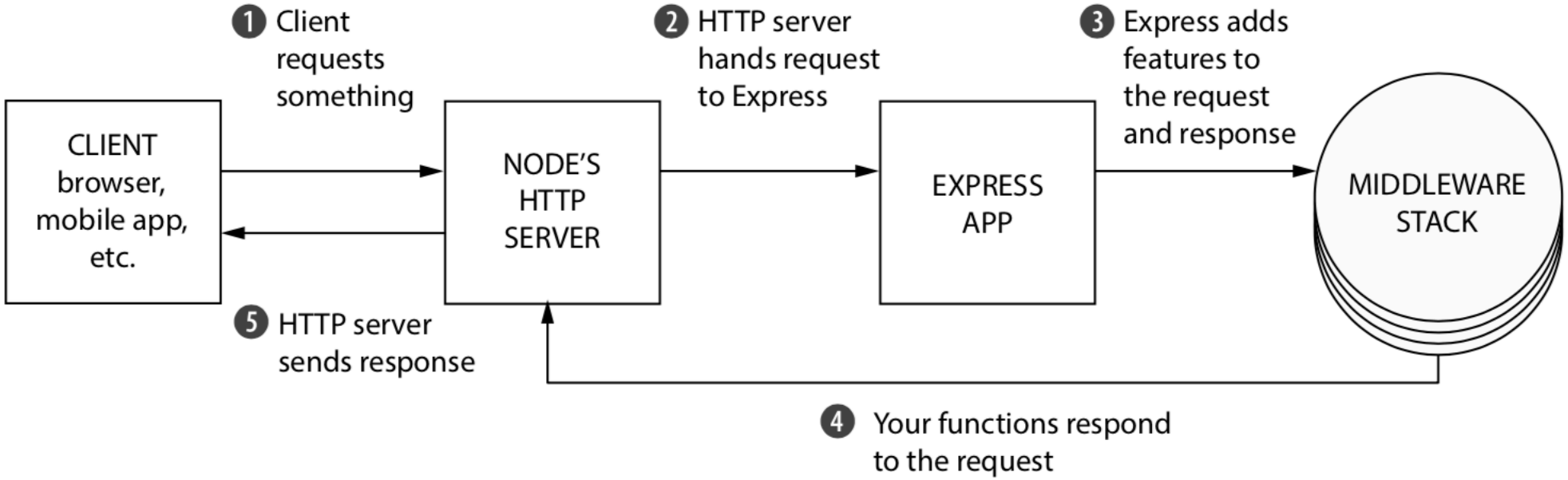
EXPRESS **MILESTONES**

- Middleware
- Error handler
- Router
- Views / template engine

MIDDLEWARES

It’s always a question to manipulate the **Request** and **Response** object

MIDDLEWARES



MIDDLEWARES

```
'use strict'

const path = require('path')
const http = require("http")
const express = require('express')
const morgan = require('morgan')
const serve = require('express').static

const app = express()
```

Attach middleware to Express

```
app.use(morgan('combined'))
app.use(serve(path.join(__dirname, 'public')))
```

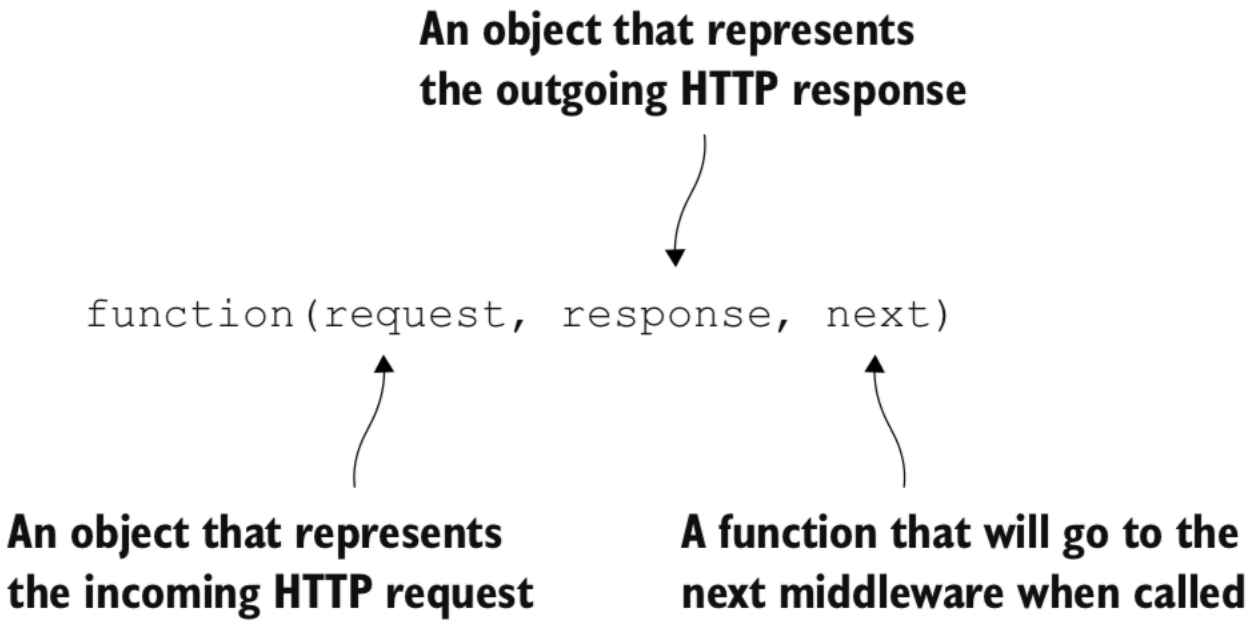
```
app.get('/', (req, res) => {
    res.status(200).sendFile('/index.html')
})

http.createServer(app).listen(5000)
console.log("Server has started.")
```

MIDDLEWARES

```
'use strict'

const bannedIps = ['192.168.0.1', '192.168.0.2', '192.168.0.3']

function myMiddleware (req, res, next) {

    // ... DO SOMETHING WITH REQUEST AND RESPONSE
    if (bannedIps.includes(req.ip)) {
        const err = new Error('Your IP is banned go away')
        next(err)
    } else {
        next()
    }

}
```

```
'use strict'

function myMiddleware (bannedIps) {
    // ... DO SOMETHING WITH BANNED IPs
    return function (req, res, next) {
        // ... DO SOMETHING WITH REQUEST AND RESPONSE
        if (bannedIps.includes(req.ip)) {
            const err = new Error('Your IP is banned go away')
            next(err)
        } else {
            next()
        }
    }
}
```

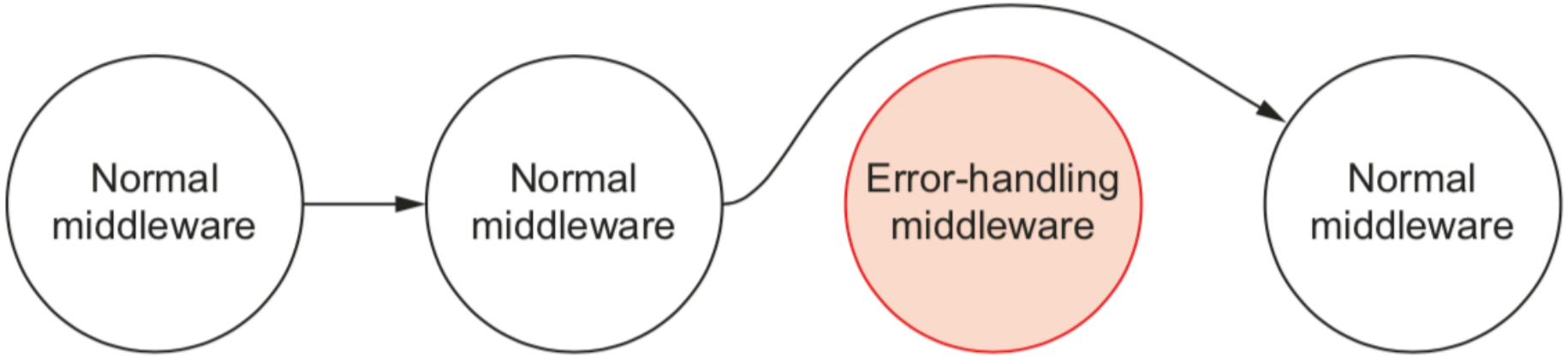
MIDDLEWARES



http://expressjs.com/en/resources/middleware.html

ERROR HANDLER

```
'use strict'

function errorHandler () {
    return function (err, req, res, next) {
        // ... PARSE YOUR ERROR
        // ... DO SOMETHING WITH REQUEST AND RESPONSE
        // ... IDENTIFY STATUS CODE AND MESSAGE FOR YOUR RESPONSE
    }
}
```



ROUTING

```
app.get('/songs', (req, res, next) => {
    // ... DO SOMETHING ON YOUR ROUTE
})

app.get('/songs/:title', (req, res, next) => {
    // ... DO SOMETHING ON YOUR ROUTE
})

app.post('/songs', (req, res, next) => {
    // ... DO SOMETHING ON YOUR ROUTE
})

app.put('/songs/:title', (req, res, next) => {
    // ... DO SOMETHING ON YOUR ROUTE
})

app.delete('/songs/:title', (req, res, next) => {
    // ... DO SOMETHING ON YOUR ROUTE
})
```

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method

VALIDATE YOUR **INPUT**

Ajv

The fastest JSON Schema validator for Node.js and browser

npm install ajv

https://github.com/epoberezkin/ajv

VIEWS / TEMPLATE ENGINE

Pug - Mustache - Dust - Nunjuks - EJS

```
app.set('view engine', 'ejs')
```

Set the engine

```
app.engine('ejs', require('ejs').__express)
```

Register the engine

```
app.set('views', path.join(__dirname, 'views'))
```

Set views folder

SECURITY

Helmet helps you secure your Express apps by setting various HTTP headers

npm install helmet

https://github.com/helmetjs/helmet

LOGGER

Log everything that happens in your application

Pay attention there is a **cost** for logging

npm install winston

https://github.com/winstonjs/winston

DOMANDE?

GRAZIE

https://github.com/NickNaso/nodejs-tsw-2017