Fernando Saez

Introducción a Node.js

Node.js fue creado por **Ryan Dahl** en 2009.

Es de código abierto y multiplataforma.

Se diseñó orientado a la creación de aplicaciones para Internet, principalmente Web.

Basado en Google V8 Engine (Soporta ECMAScript 2015)

Librerías o módulos nativos

Librerías o packgage de terceros.

Que hay de especial en Node.js?

- Javascript en el cliente y en el servidor
- Basado en el modelo Event Loop
- *+99,000 packages continua creciendo*
- Used by nearly all major technology companies and a rapidly increasing adoption by non-technology companies
- Utilizado por casi todas las principales empresas de tecnología.

Cuando usar Node.js

- Chat/Messaging
- Real-time Applications
- Intelligent Proxies
- High Concurrency Applications
- Communication Hubs
- Coordinators

- Web application
- Websocket server
- Ad server
- Proxy server
- Streaming server
- Fast file upload client
- Any Real-time data apps
- Anything with high I/O

Historias de Éxito

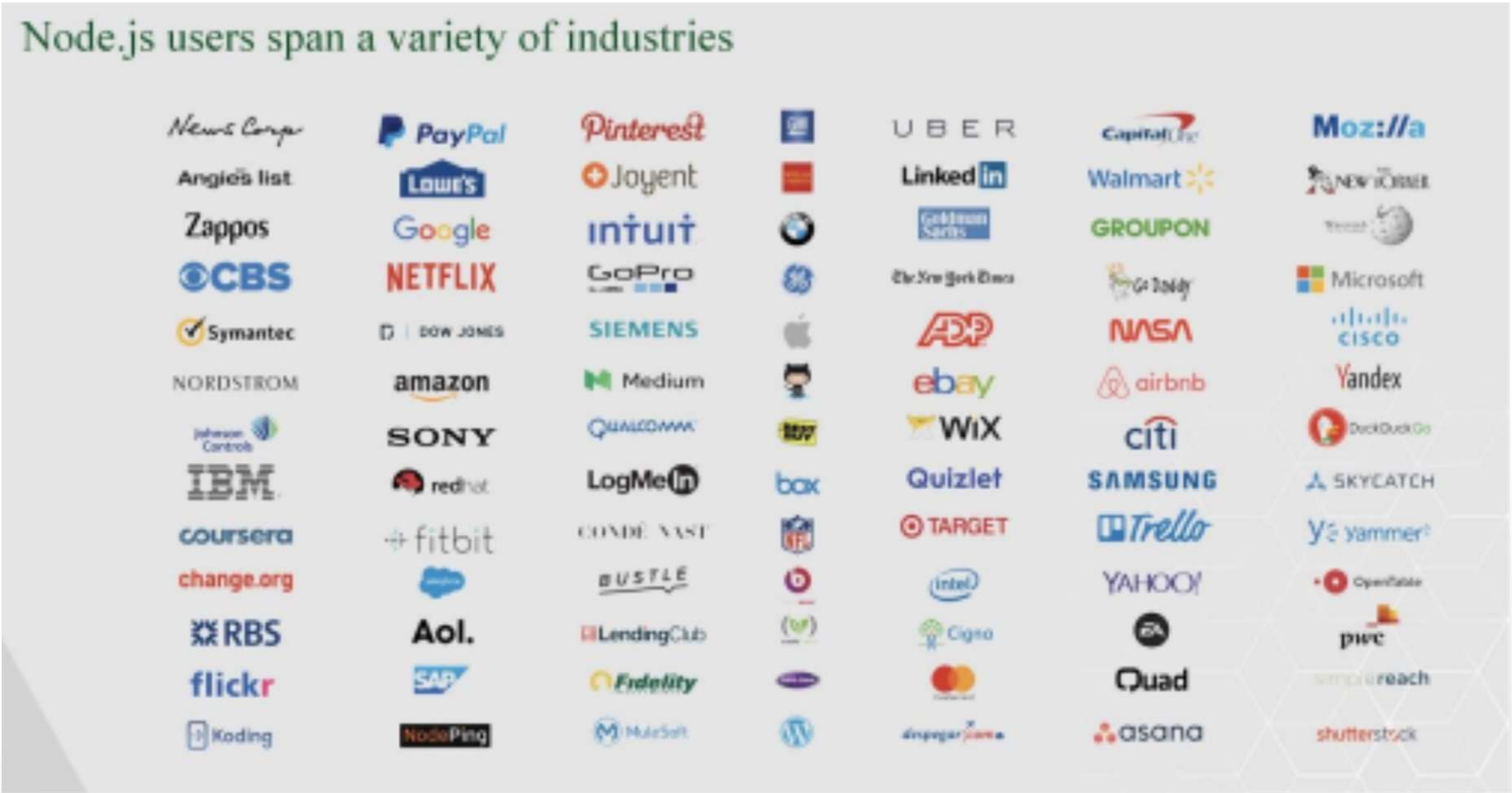
LinkedIn

Rails to Node
« Servers were cut to 3 from 30 »
« Running up to 20x faster in some scenarios »
« Frontend and backend mobile teams could be combined [...] »

PayPal™

Java to Node
« Built almost twice as fast with fewer people »
« Double the requests per second »
« 35% decrease in the average response time »

Node.js users span a variety of industries



Desarrollo para la Web

- Las aplicaciones de escritorio tradicionales tienen una interfaz de usuario conectada con lógica de fondo
  - UI basadas en Windows Forms, Jswing, WPF, Gtk, Qt, etc.
  - Dependientes del Sistema operativo

Desktop Obsoleto!!

- En la web las UI están estandarizadas
  - HTML para el contenido
  - CSS para la presentación
  - JavaScript para el contenido dinámico

Web 1.0 estáticas Infrecuentes!!

- En el pasado existían tecnologías patentadas en la web
  - Adobe Flash, ActiveX, javafx, silverlight

Apps RIA Obsoleto!!

Desarrollo para la Web

- Aplicaciones web dinámicas
  - Tecnologías del lado del servidor incluyen PHP, JSP, ASP.net, Rails, Django, y también node.js

Base para las Nuevas tendencias

- Clientes Fuertes (SPA) React, Vue y Angular.
  - Web components
  - API, REST, GraphQL
- Aplicaciones web de tiempo real
  - Push notifications, pooling, websockets
- Aplicaciones Web progresivas (PWA)
- Aplicaciones Móviles (Web, Nativas e híbridas)

Actualidad!!

Actualidad!!

Actualidad!!

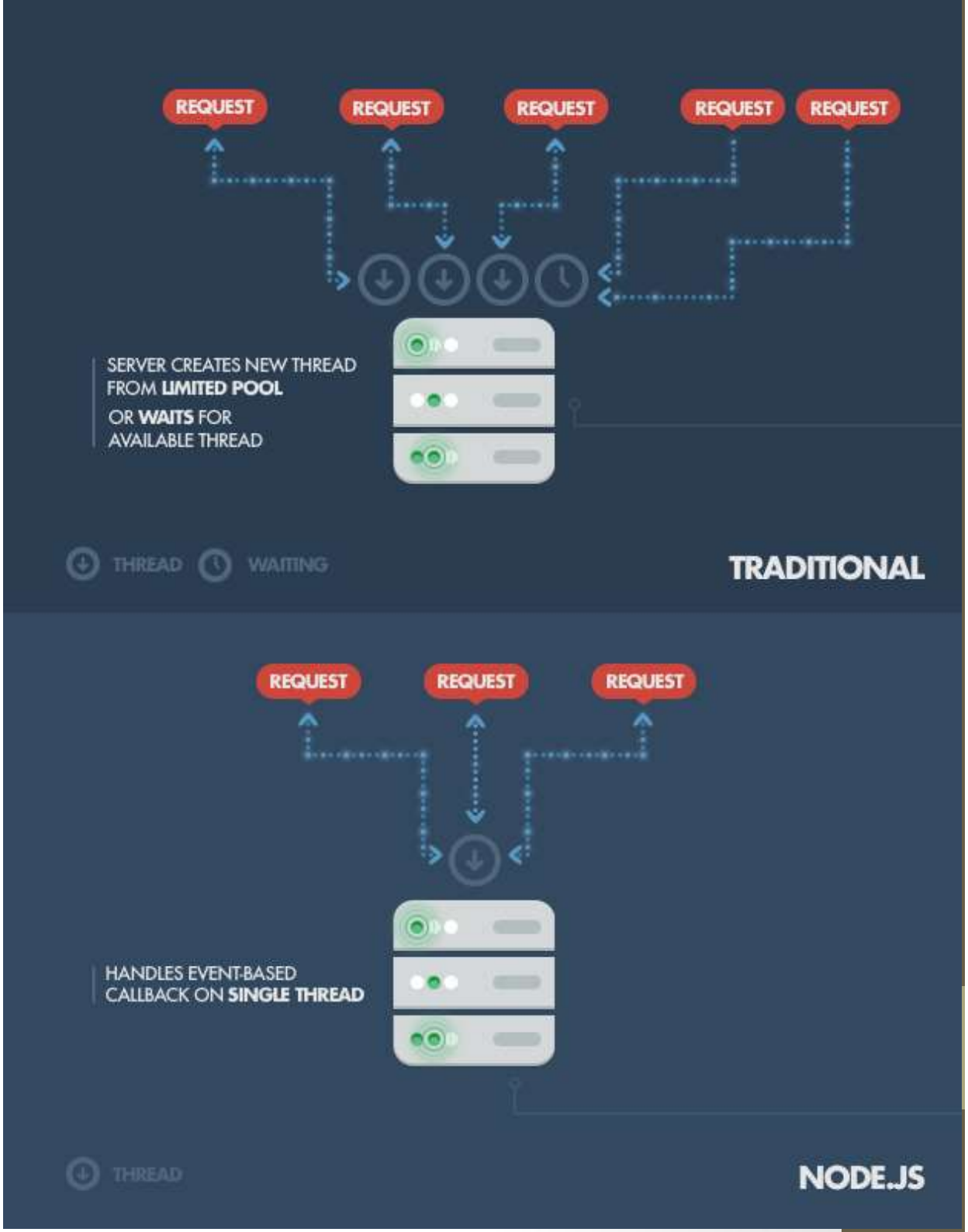
Actualidad!!

Porque usar Node.js ?

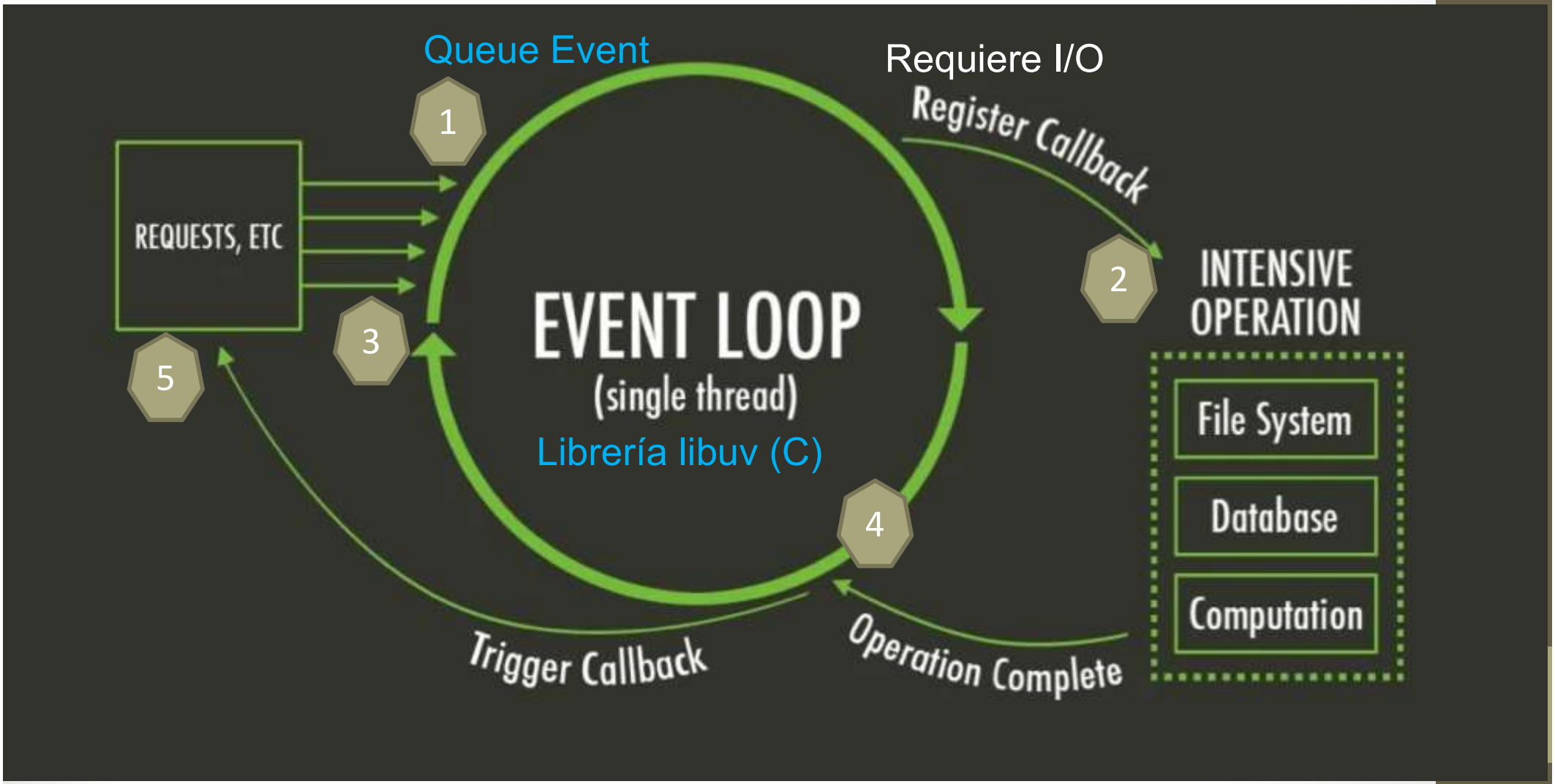
- La meta de Node's es proveer un fácil desarrollo de rápidas, escalables aplicaciones de red.
- Sin Esperas
- Naturaleza Asíncrona
- Monoproceso
- Lucha por los recursos

¿Como funciona?

- Un solo thread
- I/O asincrónica (No bloqueante)
- Event Loop



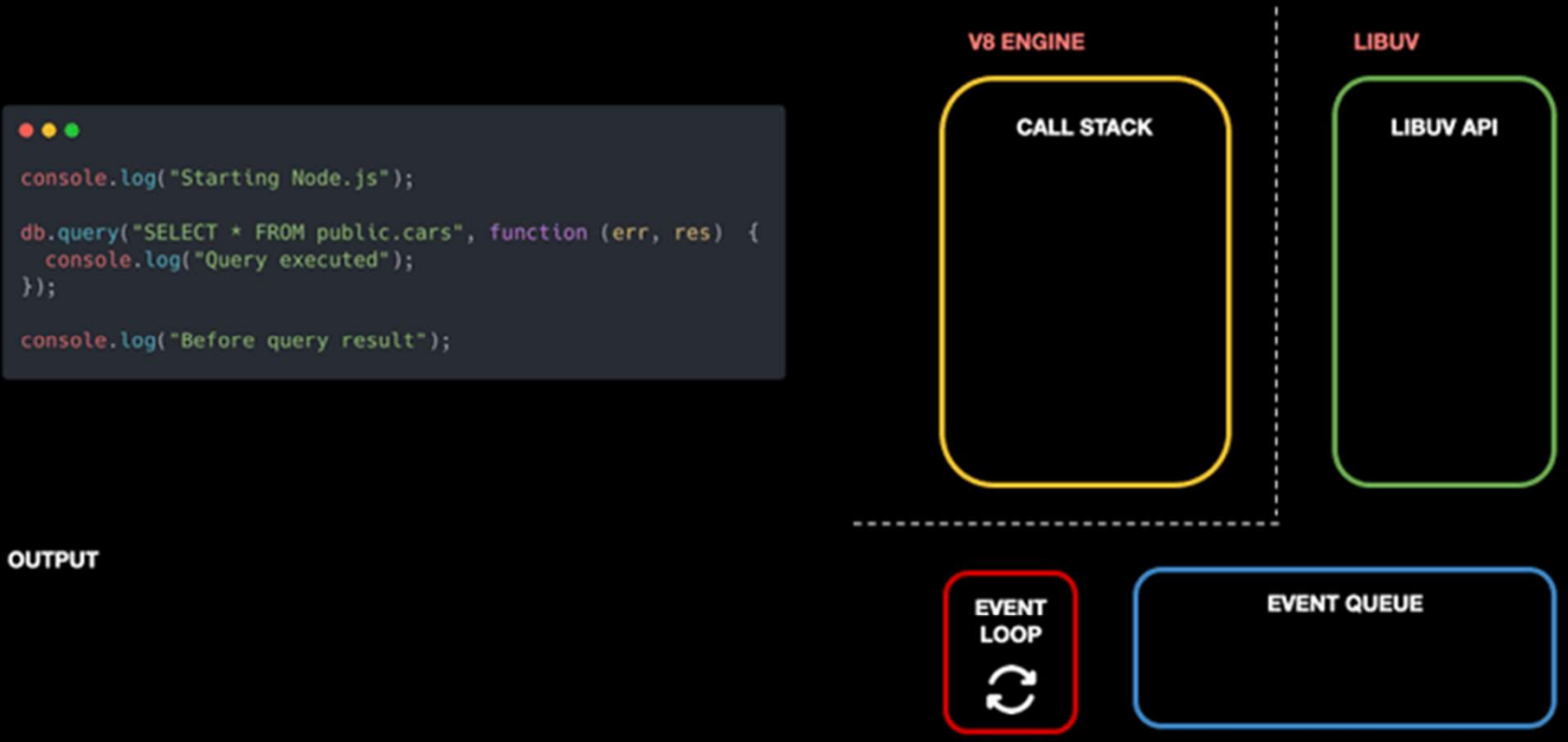
Event Driven



Event Loop

An invoked function is added to the call stack. Once it returns a value, it is popped off.

```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res)  {
  console.log("Query executed");
});

console.log("Before query result");
```



Made with ❤️ by @FabriLallo and @AndrewHu368

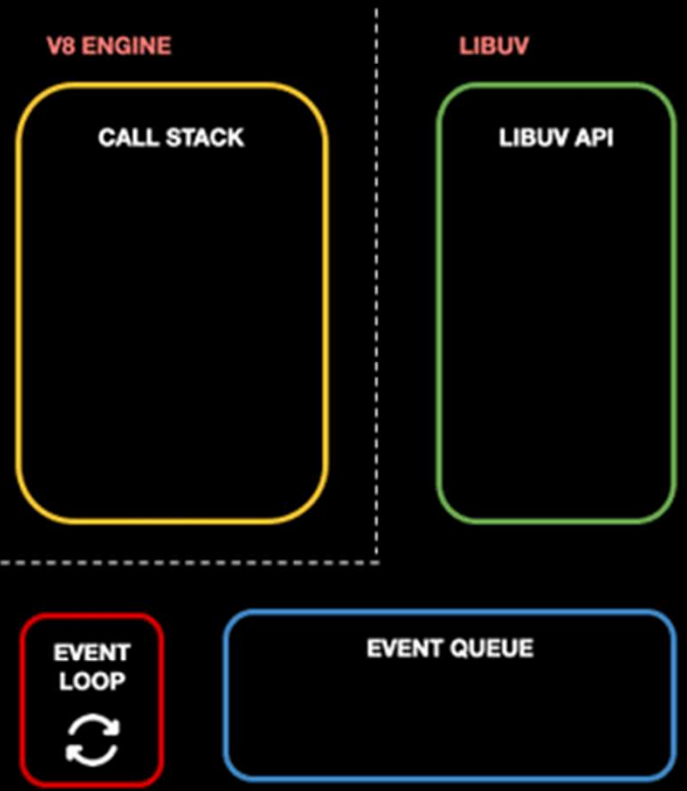
Event Loop

Database queries or other I/O ops do not block Node.js single thread because Libuv API handles them.

```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res)  {
  console.log("Query executed");
});

console.log("Before query result");
```

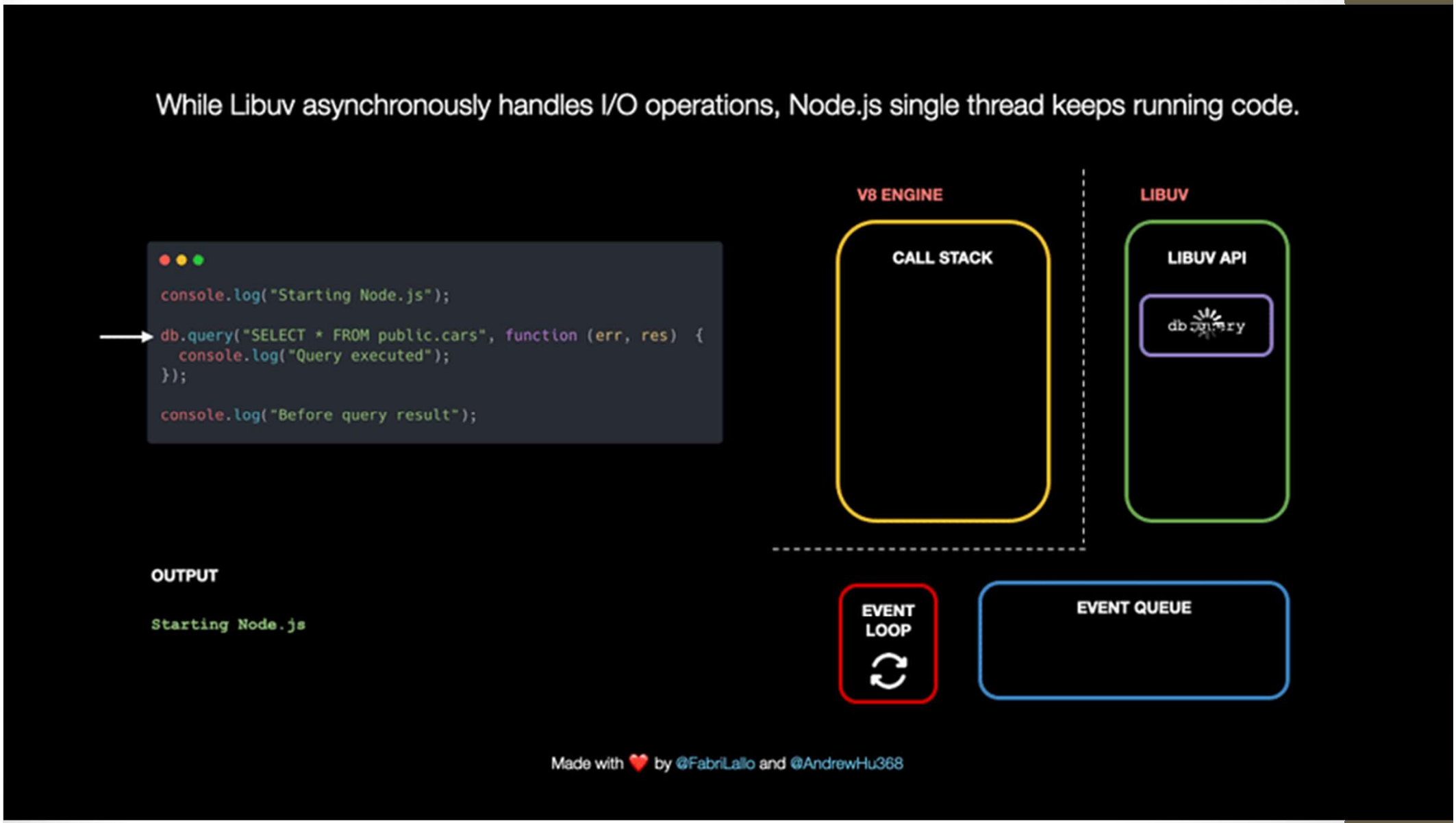
OUTPUT

```
Starting Node.js
```



Made with ❤️ by @FabriLallo and @AndrewHu368

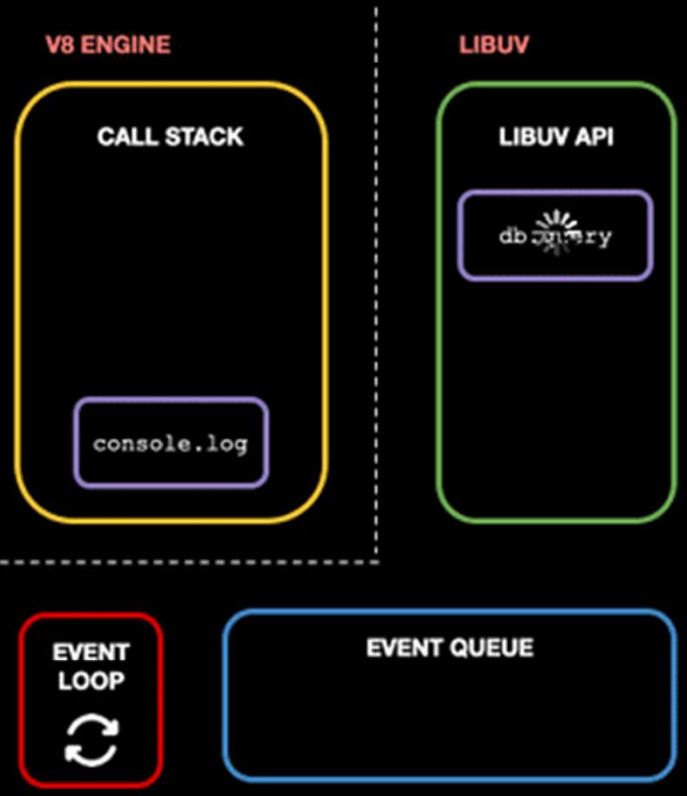
Event Loop


While Libuv asynchronously handles I/O operations, Node.js single thread keeps running code.
console.log("Starting Node.js");
db.query("SELECT * FROM public.cars", function (err, res) {
console.log("Query executed");
});
console.log("Before query result");
OUTPUT
Starting Node.js
V8 ENGINE
CALL STACK
LIBUV
LIBUV API
EVENT LOOP
EVENT QUEUE
Made with ❤️ by @FabriLallo and @AndrewHu368

Event Loop

Callbacks of completed queries are moved to the event queue. If the call stack is empty, the event loop checks for callbacks and transfers the first.

```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res)  {
  console.log("Query executed");
});

console.log("Before query result");
```



OUTPUT

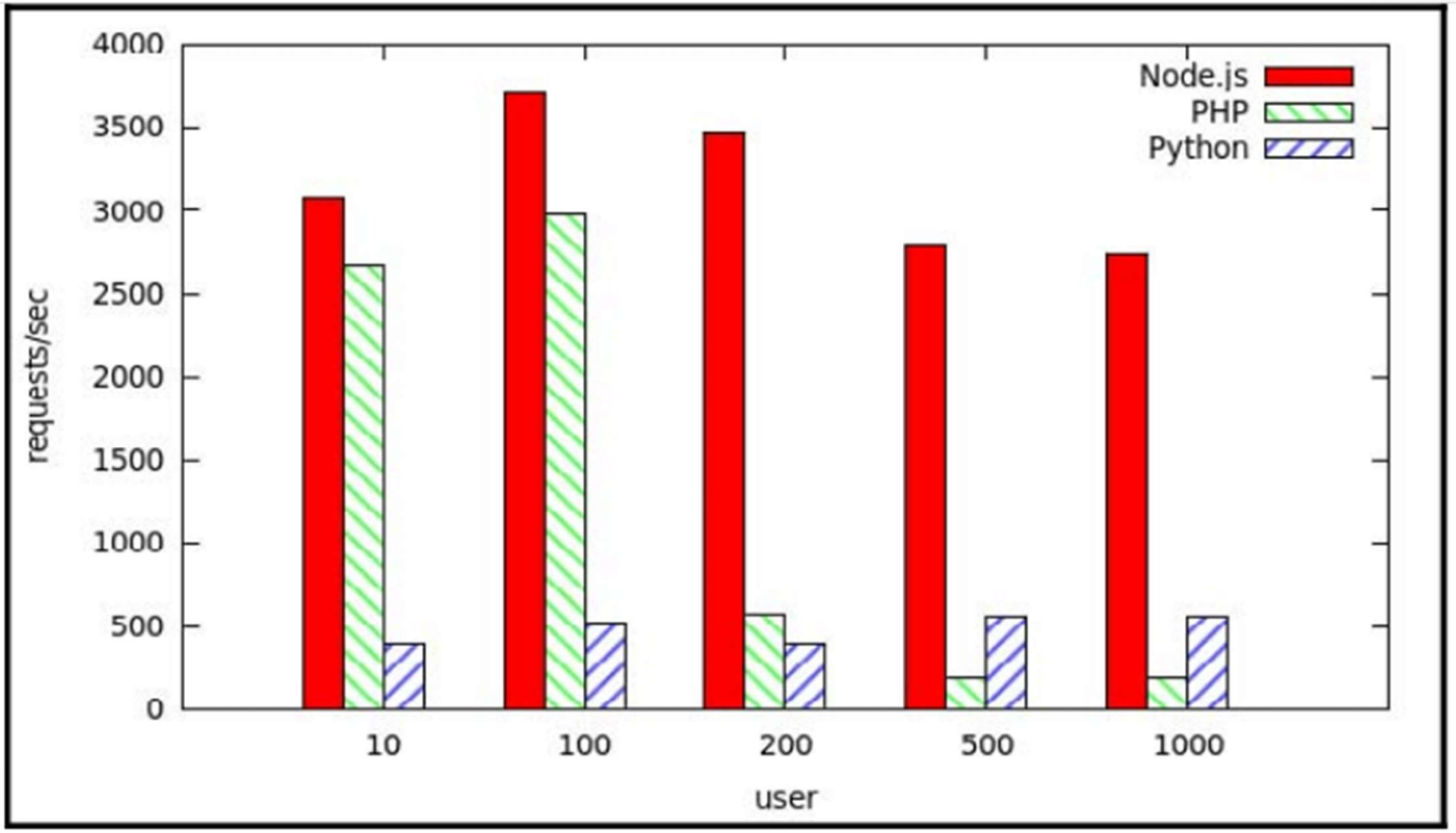
```
Starting Node.js

Before query result
```

Made with ❤️ by @FabriLallo and @AndrewHu368

Threads VS Event-driven

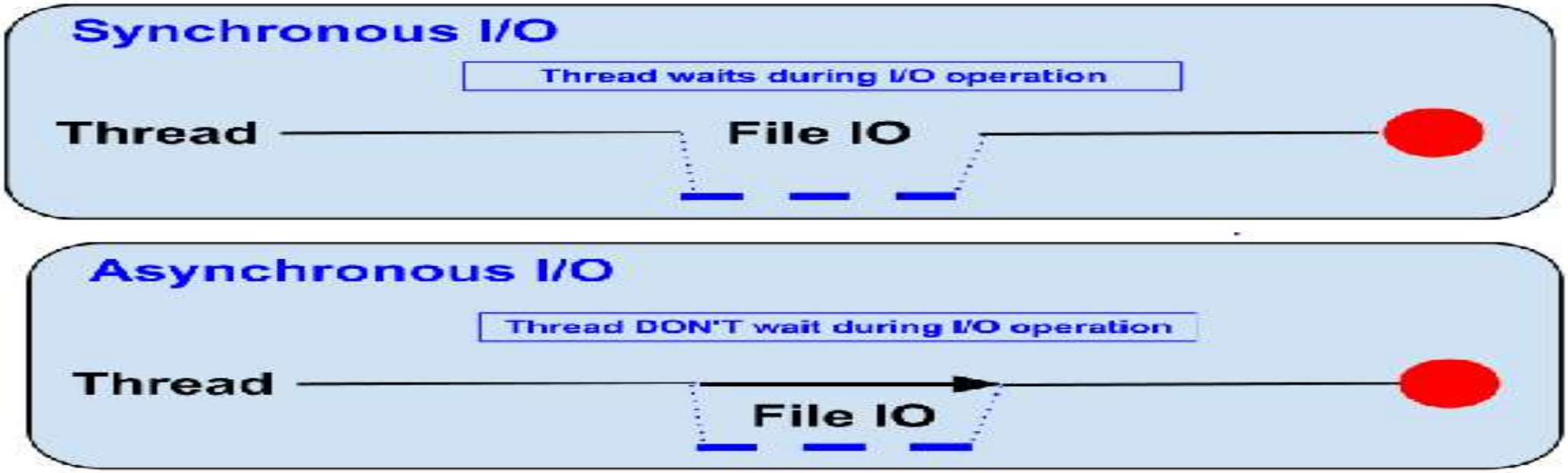
Multi-Threads	Asynchronous Event-driven
Un Thread por cada solicitud	Un solo thread para todas las solicitudeds
Consume más Memoria	Menos Memoria
Solicitudes con i/o son bloqueadas.	I/o no bloqueada. Procesada por evento loop
Usa context switching	No usa context switching
Los thread listener-workers se utilizan con frecuencia para tomar un bloqueo de solicitud entrante	El uso de I/O asíncrono es facilitado por (callbacks, not poll/select or O_NONBLOCK)

Node.js vs Apache vs Python



Operaciones bloqueantes vs No bloqueantes

Ejemplo: Leer un dato desde un archivo y mostrarlo

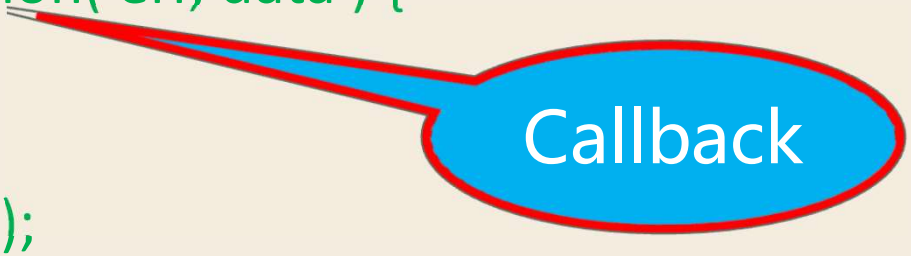


Operación bloqueante

```
var data = fs.readFileSync( “test.txt” );
console.log( data );
console.log( “otras tareas” );
//datos del archivo
//otras tareas
```

Operación No bloqueante

```
fs.readFile( “test.txt”, function( err, data ) {
console.log(data);
});
console.log( “otras tareas” );
//otras tareas
//datos del archivo
```



¿Que es NPM?

+1.000.000 Paquetes

NPM es automáticamente instalado cuando instalas Node.js.

Es común decir que hay un paquete npm para casi todo lo que se necesite.

Puedes publicar tus propios paquetes.

http://npmjs.com

Package

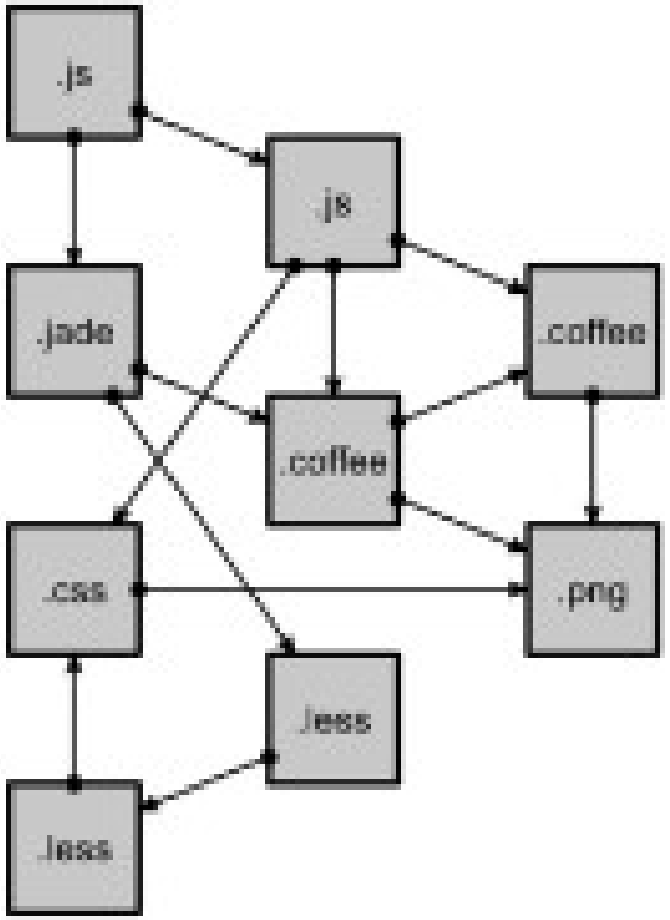
¿Que es un paquete?

¿Como se identifican?

"name": "@username/nombredelpaquete“

MAJOR.MINOR.PATCH

npm install @vue/cli@5.4.2



modules
with dependencies

Paquetes

npm init → package.json

Npm install

Local (por defecto)
Global (opción –g)

npm uninstall jquery

npm update jquery

~1.0.4	1.0.7
^1.0.4	1.1.0
*	2.1.3

Package.json

```
{
  "name": "app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "start": "nodemon index.js",
    "test": "echo \"Error: no test specified\" &&
exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.3",
    "strike-a-match": "^1.2.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.15"
  }
}
```

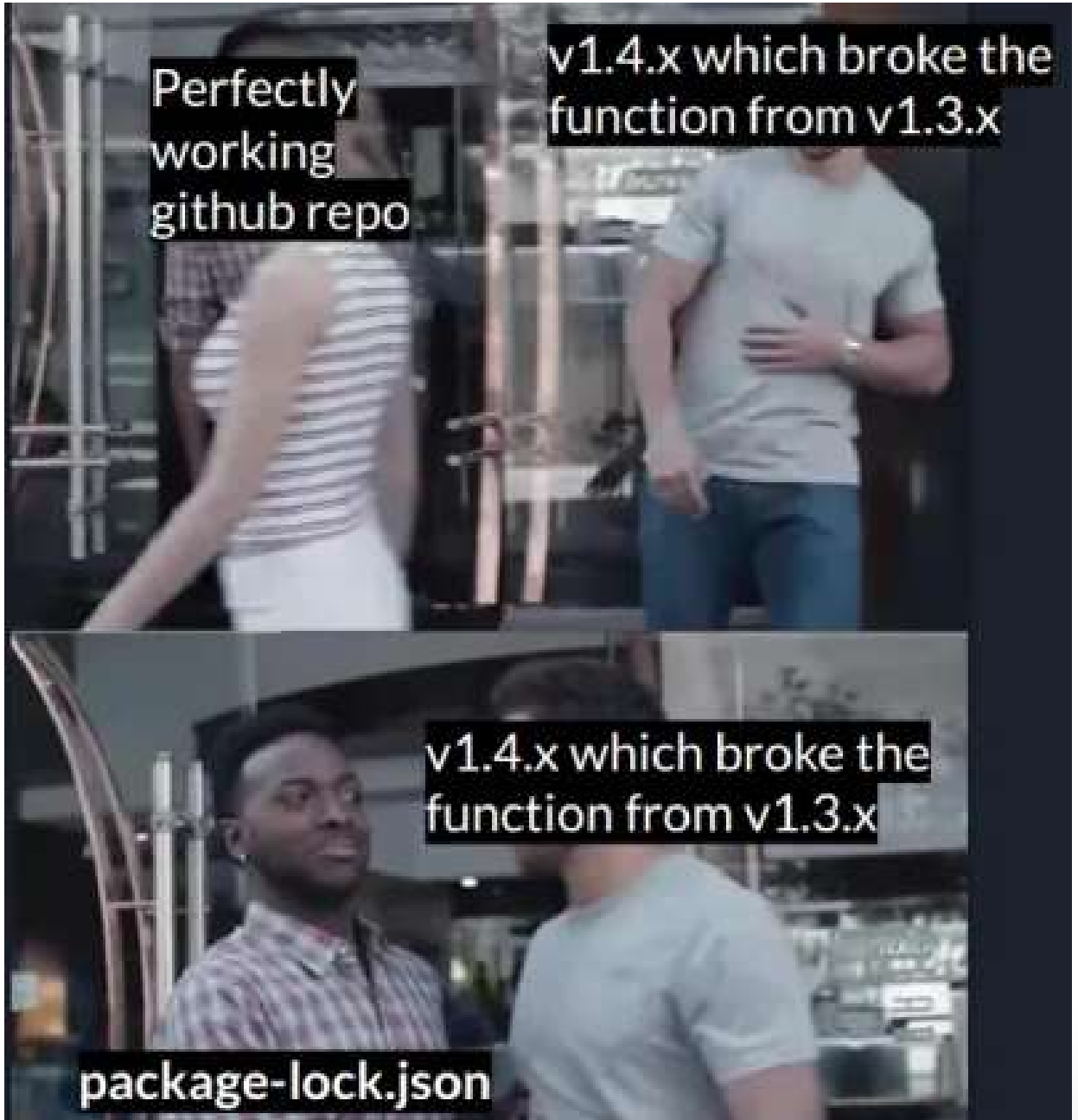
npm start

npm run test

Ej. npm install express

Ej. npm install nodemon -D

Package-lock.json



App Hola Mundo

Paso 1: crear directorio, llamar a npm init y seguir instrucciones

>mkdir myapp

>cd myapp

- Usar npm init para crear un package.json

  >npm init

- Completar el nombre de la aplicación y versión.
- RETURN para aceptar valores por defecto.

Paso 2 Editar el archive app.js y agregar

Console.log('Hola Mundo');

Paso 3 Ejecute la app con el siguiente comando:

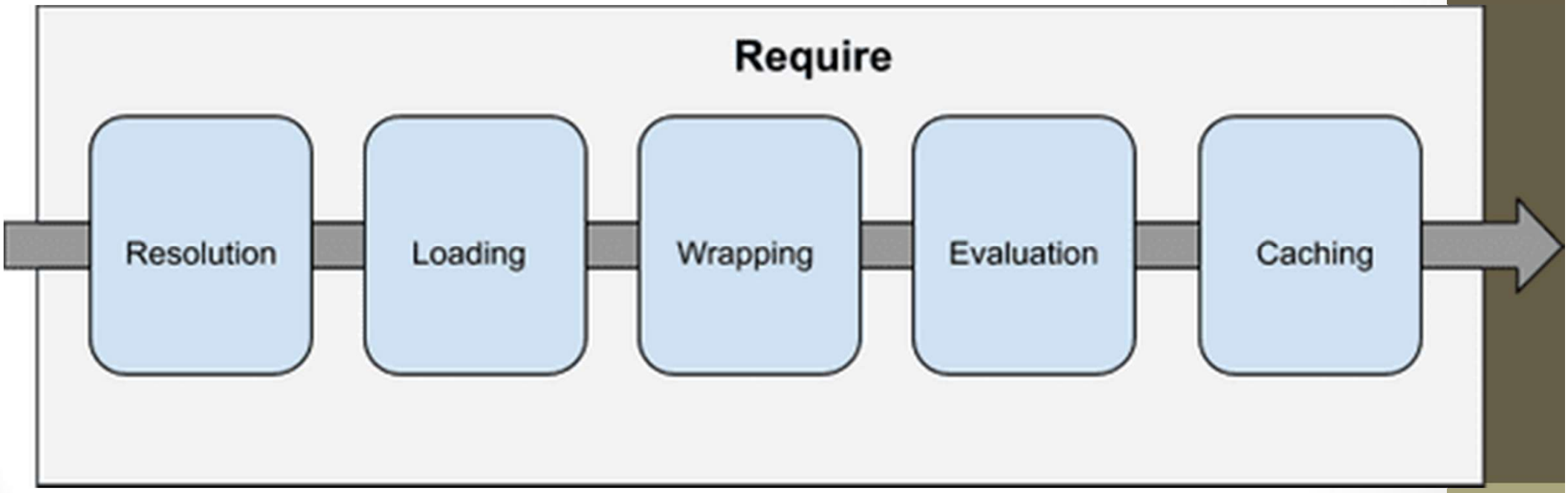
>node app.js

Package.json

```
{
"name": "holamundo",
"version": "1.0.0",
"description": "simple hola
mundo",
"main": "app.js",
"scripts": {
"test": "echo \"Error: no test
specified\" && exit 1"
},
"author": "fsaez",
"license": "ISC",
"dependencies": {
}
}
```

Módulos en NODE.js

```
const http = require('http')
const mifuncion = require('../mimodulo/mifuncion.js')
```



Usando los módulos

```
// utility.js
const add = (a, b) => { return a + b; } const
subtract = (a, b) => { return a - b; }


module.exports.add = add;
module.exports.subtract = subtract;
```

Para utilizar el **módulo utility** debemos importarlo.

```
// index.js
const utility = require('./utility.js')
console.log(utility.add(4, 5)) //9
```

```
// index2.js
const add = require('./utility.js').add
console.log(add(4, 5)) //9
```

Usando los módulos

logger.js

```
const error = 'ERROR';
const warning = 'WARNING';
const info = 'INFO';

function log(message, level = info) {
    console.log(`${level}: ${message}`);
}

module.exports.log = log;
module.exports.error = error;
module.exports.info = info;
module.exports.warning = warning;
```

app.js

```
const {
    log,
    error,
    info,
    warning
} = require('./logger');


log('Node.js module demo 1');
log('Node.js module demo 2', warning);
```

Usar ECMAScript modules

1) Agregar la clave {type: module} en package.json

2) Hacer que el archive del modulo tenga extension .mjs

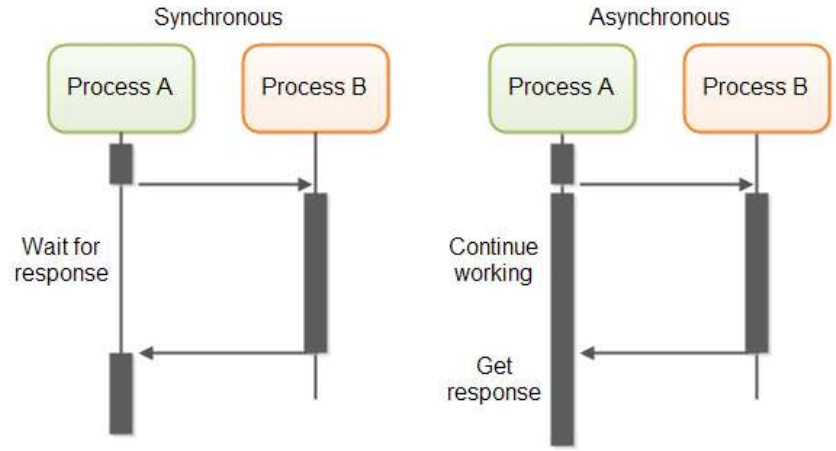
```
// month.mjs (ES Module)
import fs from ‘fs';
import { monthFromDate } from './month-from-date.mjs';
const dateString = process.argv[2] ?? null;
console.log(monthFromDate(dateString));
```

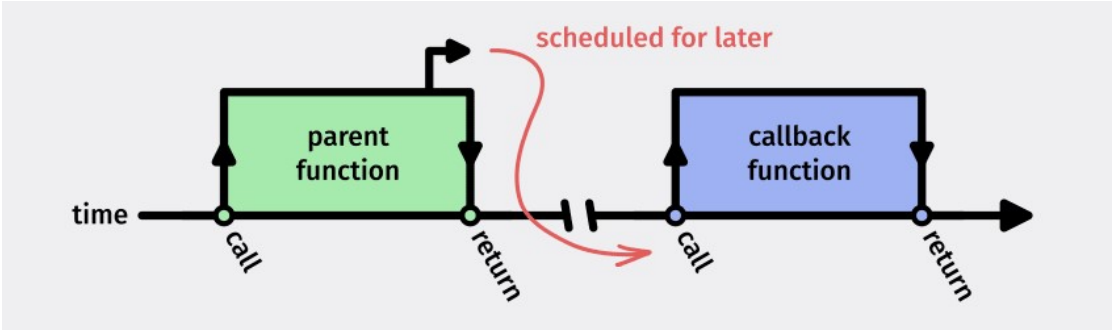
Mézclando formato de módulos

```
// ES module
import defaultComponent from './module.commonjs.js';
// ...
```

```
// CommonJS module
async function loadESModule() {
  const {
    default: defaultComponent,  component1
  } = await import('./module.es.mjs');
  // ...
}
loadESModule();
```

Asincronismo y Callbacks






```
function hacerAlgo() {
  numberOfClicks++;
}
const button = document.getElementById('action-button’);

button.addEventListener("click", hacerAlgo);
```

Módulo para administrar el sistema de archivos: fs

fs: módulo que viene implementado en Node.js por defecto y nos permite acceder al sistema de archivos para poder leer sus contenidos y crear otros archivos o carpetas.

```
const fs=require('fs');
fs.writeFile('./archivo1.txt', 'línea 1\nLínea 2', error => {
  if (error)
    console.log(error);
  else
    console.log('El archivo fue creado');
});
console.log('última línea del programa');
```

Módulo para administrar el sistema de archivos: fs

```
const fs=require('fs');
fs.readFile('./archivo1.txt',leer);

console.log('última línea del programa');

function leer(error,datos){
    if (error)
      console.log(error);
    else
      console.log(datos.toString());
}
```

Callback hell issue

```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

ES7

ES6

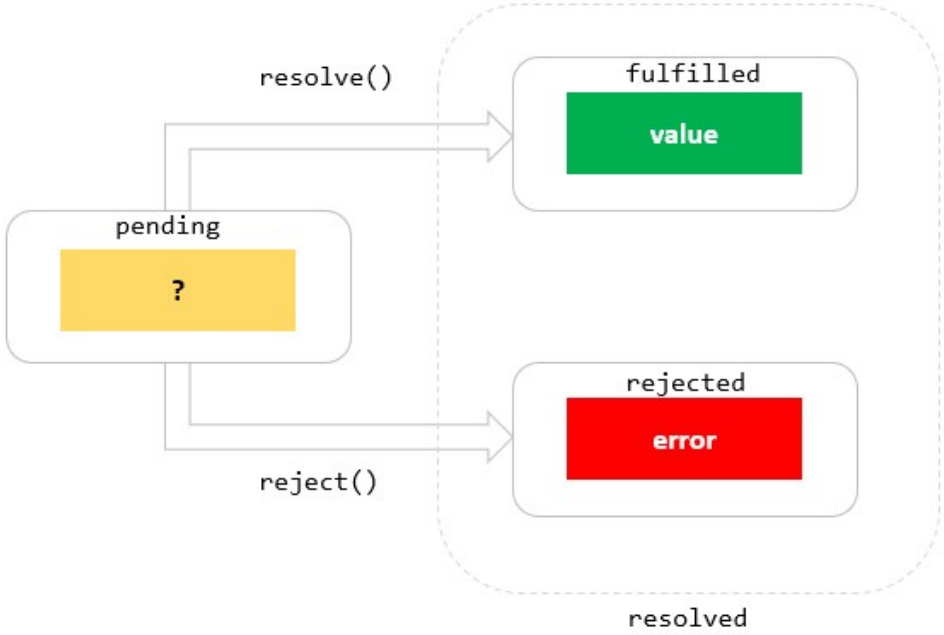
Promise

ES5

Promesas

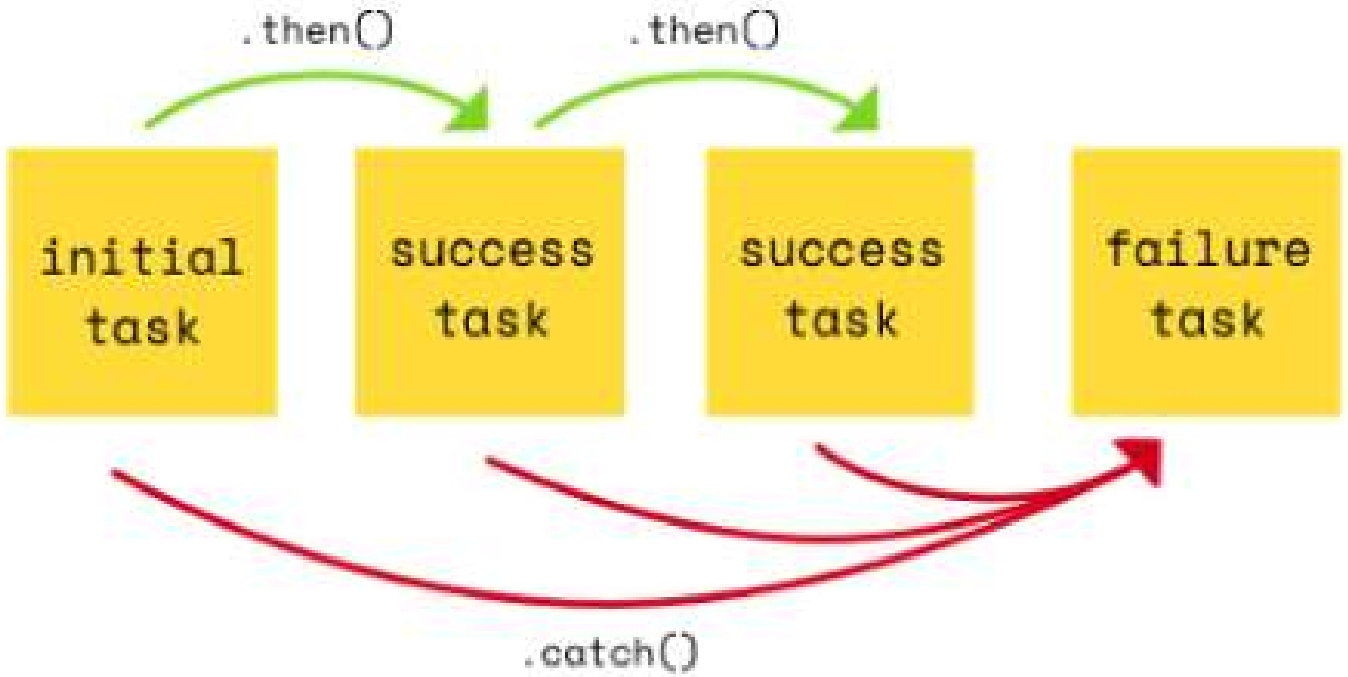
Disponible desde ES6

Una promesa se define como una función no-bloqueante y asíncrona cuyo valor de retorno puede estar disponible justo en el momento, en el futuro o nunca.



Promesas

Cuando consumimos una promesa, usamos callbacks para los estados cumplidos y rechazados de esa promesa.



Fetch con Promesas

```
fetch(`https://seriousnews.com/api/headlines`)
  .then(response => response.json())
  .then(headlines => {
    console.log(headlines));
    //or do something with the nicely
    //formatted json hash of headlines
  }).catch(error => console.log(error));
```

Fetch con Promesas

```
const fetchPromise = fetch('https://mdn.github.io/learning-area/javascript/apis/fetching-data/can-store/products.json');

fetchPromise.then((response) => {
  const jsonPromise = response.json();
  jsonPromise.then((data) => {
    console.log(data[0].name);
  });
});
```

Como crear nuestras Promesas

```
let completed = true;
let learnJS = new Promise(function (resolve, reject) {
    setTimeout(() => {
        if (completed) {
            resolve(“Promesa resuelta"); }
        else {
            reject(“Retorno promesa rechazada como Error"); }
     }, 8 * 1000); });
```

```
//consumiento promesa
learnJS
    .then(success => console.log(success)) // Promesa resuelta
    .catch(reason => console.log(reason))
    .finally(() => console.log('finally'));
```

Promise.all( )

```
var p1 = Promise.resolve(3);
var p2 = 1337;
var p3 = new Promise((resolve, reject) => {
  setTimeout(resolve, 100, "foo");
});

Promise.all([p1, p2, p3]).then(values => {
  console.log(values); // [3, 1337, "foo"]
});
```

Async Await

```
function helloWorld() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('Hello World!');
    }, 2000);
  });
}

async function msg() {
  const msg = await helloWorld();
  console.log(‘Mensaje:', msg);
}

msg(); // Mensaje: Hello World! <-- después de 2 segundos
```

Async Await

```
function resolveAfter2Seconds(x) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(x);
    }, 2000);
  });
}
```

```
async function add1(x) {
  const a = await resolveAfter2Seconds(20);
  const b = await resolveAfter2Seconds(30);
  return x + a + b;
}
add1(10).then(v => {
  console.log(v);  // prints 60 after 4 seconds.
});
```

```
async function add2(x) {
  const p_a = resolveAfter2Seconds(20);
  const p_b = resolveAfter2Seconds(30);
  return x + await p_a + await p_b;
}

add2(10).then(v => {
  console.log(v);  // prints 60 after 2 seconds.
});
```

Reescritura de Promesas en Async Await

```
function getProcessedData(url) {
  return downloadData(url) // returns a promise
    .catch(e => {
      return downloadFallbackData(url)  // returns a promise
    })
    .then(v => {
      return processDataInWorker(v); // returns a promise
    });
}
```

JAVASCRIPT PROMISES

```
async function getProcessedData(url) {
  let v;
  try {
    v = await downloadData(url);
  } catch(e) {
    v = await downloadFallbackData(url);
  }
  return processDataInWorker(v);
}
```

async/await

Usar fetch con Async Await

```
async function fetchMovies() {
  const response = await fetch('/movies');
  // espera hasta que request se complete...
  console.log(response);
}
```

```
async function fetchMoviesJSON() {
  const response = await fetch('/movies');
  const movies = await response.json();
  return movies;
}

fetchMoviesJSON().then(movies => {
  movies; // fetched movies
});
```

Async Await y Promise.all( )

1
```
const fetcher = async(path) => {
  let response = await fetch('https://jsonplaceholder.typicode.com' + path);
  return await response.json()
}
```
2
```
async function getData() {
  try {
    let [users, todos] = await Promise.all([fetcher('/users'), fetcher('/todos')]);

    let user = users.map(item => item.name);
    let todo = todos.map(item => item.title);
    console.log(user);
    console.log(todo);
  } catch (error) {
    console.log('error: ' + error);
  }
};
getData();
```


¡YO TENGO UNA PREGUNTA!
¡PREGUNTALE A SAN GOOGLE!
¿ALGUNA PREGUNTA?