

**Pero, ¿Qué es Node.js?**

# ¿Qué es Node.js?

Lo que todos sabemos

- Hay Javascript por alguna parte
- Backend
- ¿Algo que ver con NoSQL?
- Sirve para hacer chats

# ¿Qué es Node.js?

Lo que no está tan claro

- ¿Es un framework para Javascript?
- ¿Es una librería?
- ¿Qué tiene que ver v8 con Node.js?
- **• ¿Para qué sirve (además de los chats)?**

# ¿Qué es Node.js?

Node.js es:

- “Una plataforma de software usada para construir aplicaciones de red escalables (especialmente servidores). Node.js utiliza JavaScript como lenguaje, y alcanza alto rendimiento utilizando E/S no bloqueante y un bucle de eventos de una sola hebra”.

# ¿Qué es Node.js?

Node.js es:

- “Una plataforma de software usada para construir aplicaciones de red escalables (especialmente servidores). Node.js utiliza JavaScript como lenguaje, y alcanza alto rendimiento utilizando E/S no bloqueante y un bucle de eventos de una sola hebra”.
- **Wat?**

# ¿Qué es Node.js?

Por ejemplo, si...

- ...para **ruby** tenemos **Rails**...
- ...para **python** tenemos **Django**...
- ...para **php** tenemos **Symphony**...

# ¿Qué es Node.js?

Por ejemplo, si...

- ...para **ruby** tenemos **Rails**...
- ...para **python** tenemos **Django**...
- ...para **php** tenemos **Symphony**...
- ¿Podríamos decir que **Node.js** es el equivalente para **JavaScript**?

# ¿Qué es Node.js?

i iNO!!

# **¿Qué es Node.js?**

¿Qué es un “lenguaje de programación”?

# ¿Qué es Node.js?

¿Qué es un “lenguaje de programación”?

- Una gramática que define la sintaxis del lenguaje
- Un intérprete/compilador que lo sabe interpretar y ejecutar

# ¿Qué es Node.js?

¿Qué es un “lenguaje de programación”?

- Una gramática que define la sintaxis del lenguaje
- Un intérprete/compilador que lo sabe interpretar y ejecutar
- **Mecanismos para interactuar con el mundo exterior (llamadas al sistema)**

# ¿Qué es Node.js?

¿Qué es un “lenguaje de programación”?

- Una gramática que define la sintaxis del lenguaje
- Un intérprete/compilador que lo sabe interpretar y ejecutar
- **Mecanismos para interactuar con el mundo exterior (llamadas al sistema)**
- **Librería estándar (consola, ficheros, red, etc,...)**

# ¿Qué es Node.js?

¿Qué es un “lenguaje de programación”?

- Una gramática que define la sintaxis del lenguaje
- Un intérprete/compilador que lo sabe interpretar y ejecutar
- **Mecanismos para interactuar con el mundo exterior (llamadas al sistema)**
- **Librería estándar (consola, ficheros, red, etc,...)**
- **Utilidades (intérprete interactivo, depurador, paquetes)**

# ¿Qué es Node.js?

## v8 (JavaScript)

- Una gramática que define la sintaxis del lenguaje
- Un intérprete/compilador que lo sabe interpretar y ejecutar
- Mecanismos para interactuar con el mundo exterior (llamadas al sistema)
- Librería estándar (consola, ficheros, red, etc,...)
- Utilidades (intérprete interactivo, depurador, paquetes)

**Node.js**

# ¿Qué es Node.js?

	<b>Node.js</b>	<b>Ruby</b>	<b>Python</b>	<b>Java</b>
<b>Lenguaje</b>	JavaScript	Ruby	Python	Java
<b>Motor</b>	v8	YARV	cPython	JavaVM
<b>Entorno</b>	<b>Node.js</b>	Ruby Standard Library	Python Standard Library	Java SE
<b>Framework</b>	???	Rails	Django	Spring

# ¿Qué es Node.js?

Node.js es algo más:

- Una filosofía sobre cómo hacer las cosas
- Un modelo de ejecución singular
- Muy enfocado hacia aplicaciones de red

# ¿Qué es Node.js?

Node.js es algo más:

- **Una filosofía sobre cómo hacer las cosas**
- Un modelo de ejecución singular
- Muy enfocado hacia aplicaciones de red

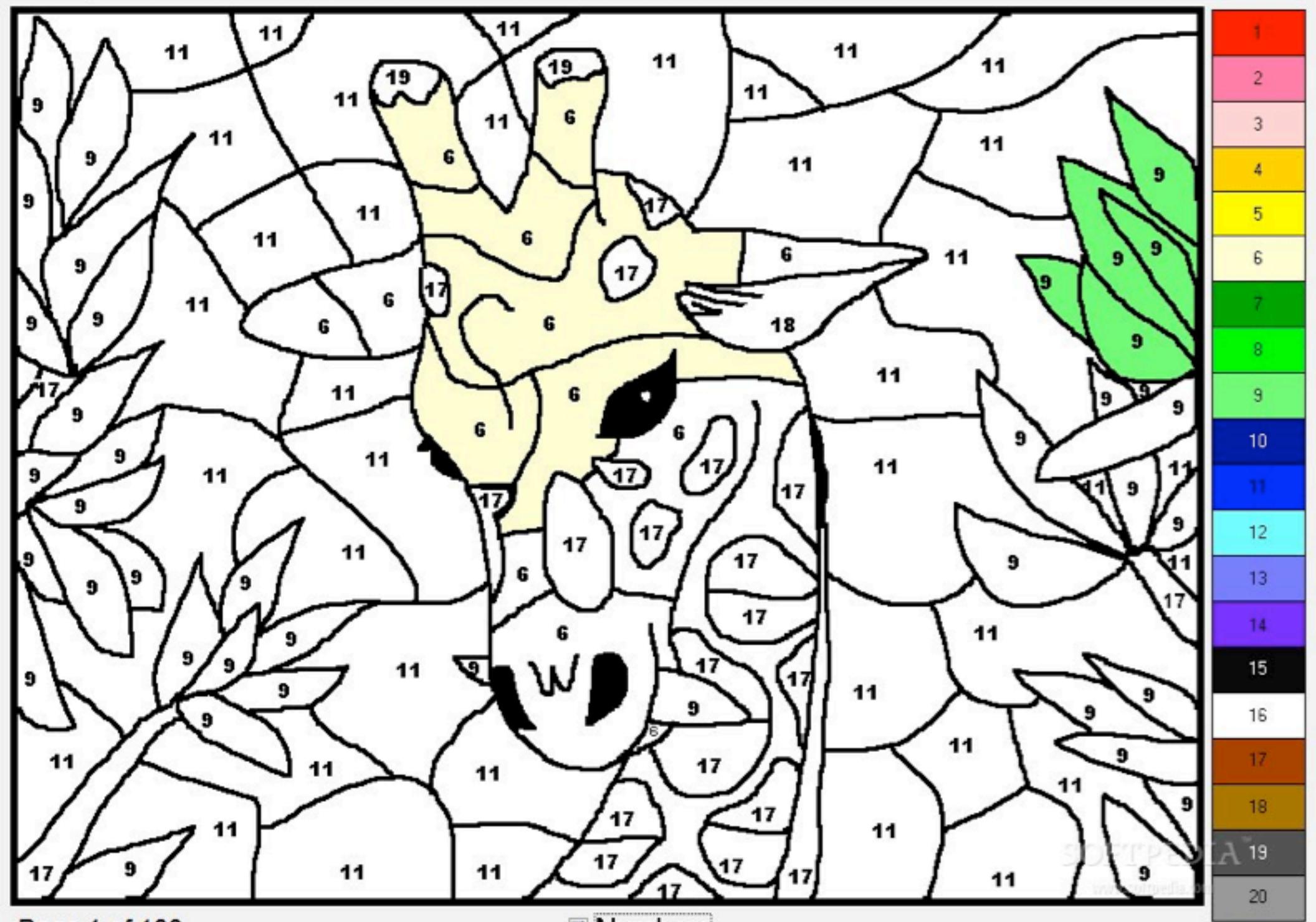
# Una filosofía

Node.js se crea con un objetivo en mente:

- Escribir aplicaciones muy eficientes (E/S) con el lenguaje dinámico más rápido (v8) para soportar miles de conexiones simultáneas
- Sin complicarse la vida innecesariamente
  - Sin paralelismo
  - Lenguaje sencillo y muy extendido
  - API muy pequeña y muy consistente
  - Apoyándose en Eventos y Callbacks













# Una filosofía

- No es la mejor opción para todos los casos
  - Si puedes hacerlo con Rails/Django/Spring, hazlo
- Evita las “soluciones totales”
  - Una necesidad, una herramienta
  - Combina diferentes herramientas simples
- Entiende lo que estás haciendo
- Flexibilidad > magia
  - Tu código es tu responsabilidad
  - Cada aplicación es un mundo







# ¿Qué es Node.js?

Node.js es algo más:

- Una filosofía sobre cómo hacer las cosas
- **Un modelo de ejecución singular**
- Muy enfocado hacia aplicaciones de red

# **Un modelo de ejecución**

Para entender Node.js, tenemos que entender dos ideas fundamentales:

- Conurrencia vs. paralelismo (asincronía)
- Eventos

# Un modelo de ejecución

Concurrencia vs. Paralelismo

- ¿Qué significa que dos cosas suceden “*a la vez*”?

# Un modelo de ejecución

¿Qué significa que dos cosas suceden “*a la vez*”?



# Un modelo de ejecución

¿Qué significa que dos cosas suceden “*a la vez*”?



# Un modelo de ejecución

Concurrencia vs. Paralelismo

- Paralelismo: varios actores realizando una acción cada uno **simultáneamente**
- Concurrencia: un solo actor, con varias tareas “activas”  
**entre las que va alternando**

# Un modelo de ejecución

Paralelismo



# Un modelo de ejecución

Concurrencia



# Un modelo de ejecución

La potencia de Node.js es (curiosamente):

- Un modelo de ejecución **concurrente**
  - Muchos clientes o tareas activas
- Pero **NO paralelo**
  - Una única hebra

# **Un modelo de ejecución**

- ¿Qué ventajas tiene evitar el paralelismo?
- ¿Qué desventajas?
- Y por tanto, ¿Cuándo es útil el modelo de Node.js?

# Un modelo de ejecución

## Patrón Reactor

- Un patrón de diseño para manejar eventos donde peticiones de servicio se transladan concurrentemente a un manejador central que se encarga de desmultiplexar las peticiones y despacharlas síncronamente mediante sus manejadores particulares asociados.

# Un modelo de ejecución

## Patrón Reactor

- Un patrón de diseño para manejar eventos donde peticiones de servicio se transladan concurrentemente a un manejador central que se encarga de desmultiplexar las peticiones y despacharlas síncronamente mediante sus manejadores particulares asociados.
- WAT??

# **Un modelo de ejecución**

Patrón Reactor



**Bucle Principal**

# Un modelo de ejecución

Patrón Reactor

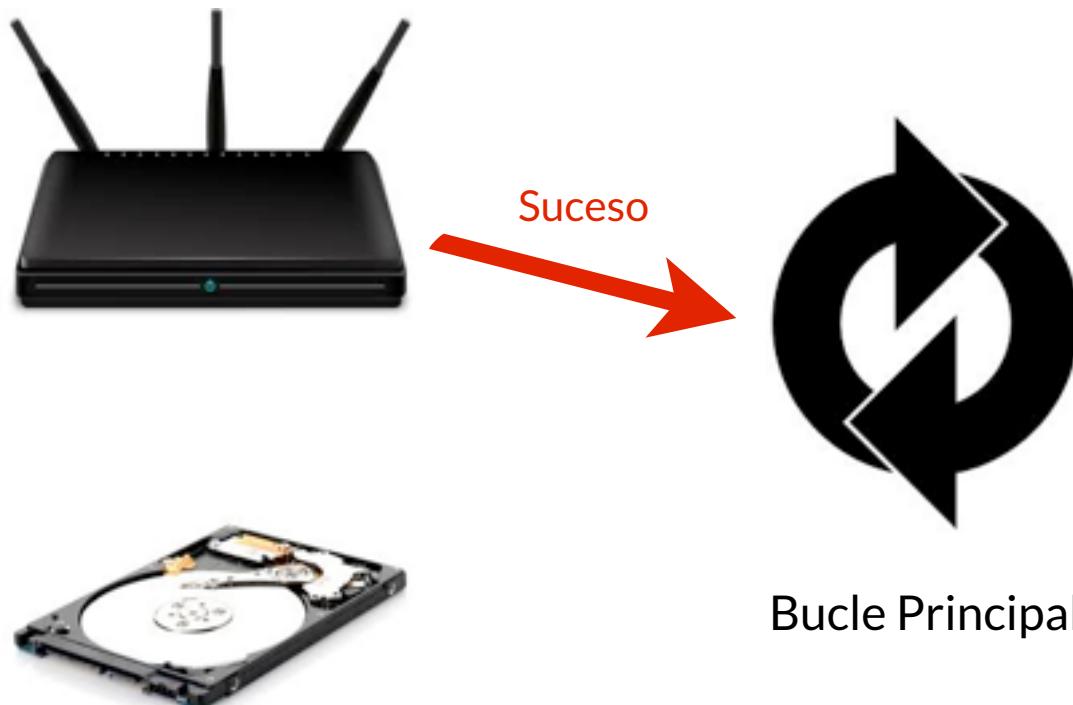


Bucle Principal

Mundo Exterior

# Un modelo de ejecución

Patrón Reactor



Mundo Exterior

# Un modelo de ejecución

Patrón Reactor



Mundo Exterior

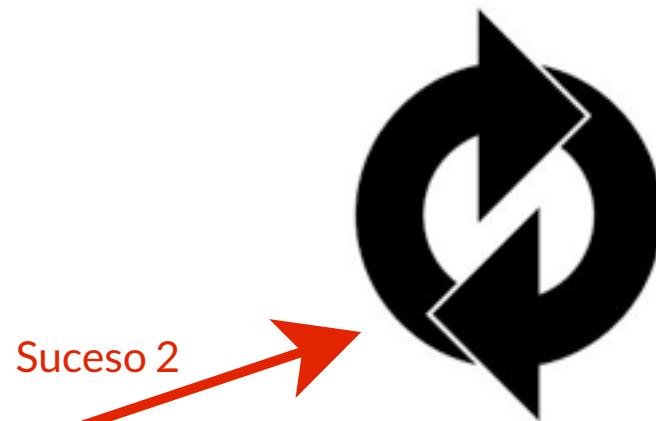


Bucle Principal

Suceso

# Un modelo de ejecución

Patrón Reactor



Bucle Principal

Suceso

Mundo Exterior

# Un modelo de ejecución

Patrón Reactor



Mundo Exterior



Bucle Principal

Suceso  
Suceso 2

# Un modelo de ejecución

Patrón Reactor



Mundo Exterior



Bucle Principal

Suceso  
Suceso 2

# Un modelo de ejecución

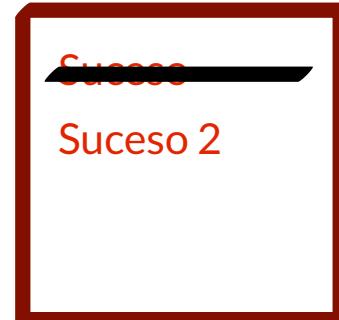
Patrón Reactor



Mundo Exterior



Bucle Principal



# Un modelo de ejecución

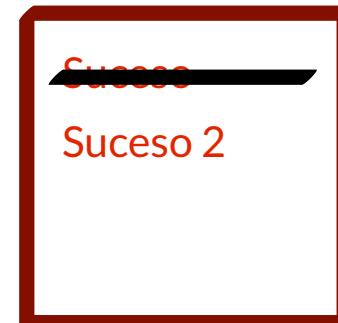
Patrón Reactor



Mundo Exterior



Bucle Principal



Manejadores

# Un modelo de ejecución

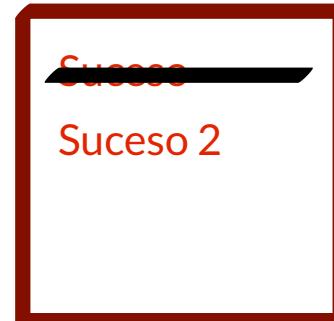
Patrón Reactor



Mundo Exterior



Bucle Principal



Manejadores

# Un modelo de ejecución

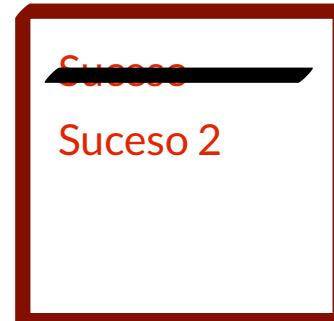
Patrón Reactor



Suceso 3



Bucle Principal



Mundo Exterior



Manejadores

# Un modelo de ejecución

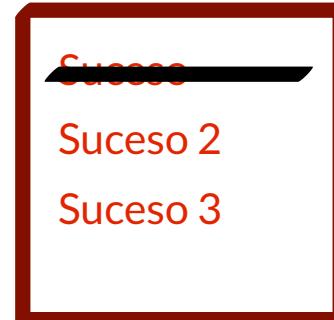
Patrón Reactor



Mundo Exterior



Bucle Principal



Manejadores

# Un modelo de ejecución

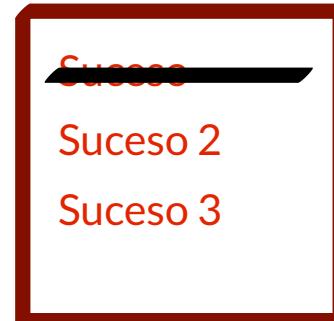
Patrón Reactor



Mundo Exterior



Bucle Principal



Manejadores

# Un modelo de ejecución

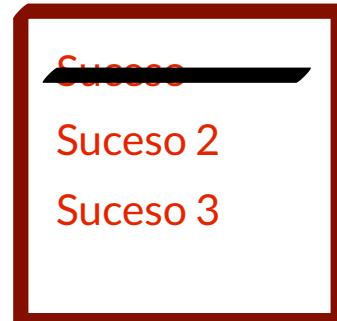
Patrón Reactor



Mundo Exterior



Bucle Principal



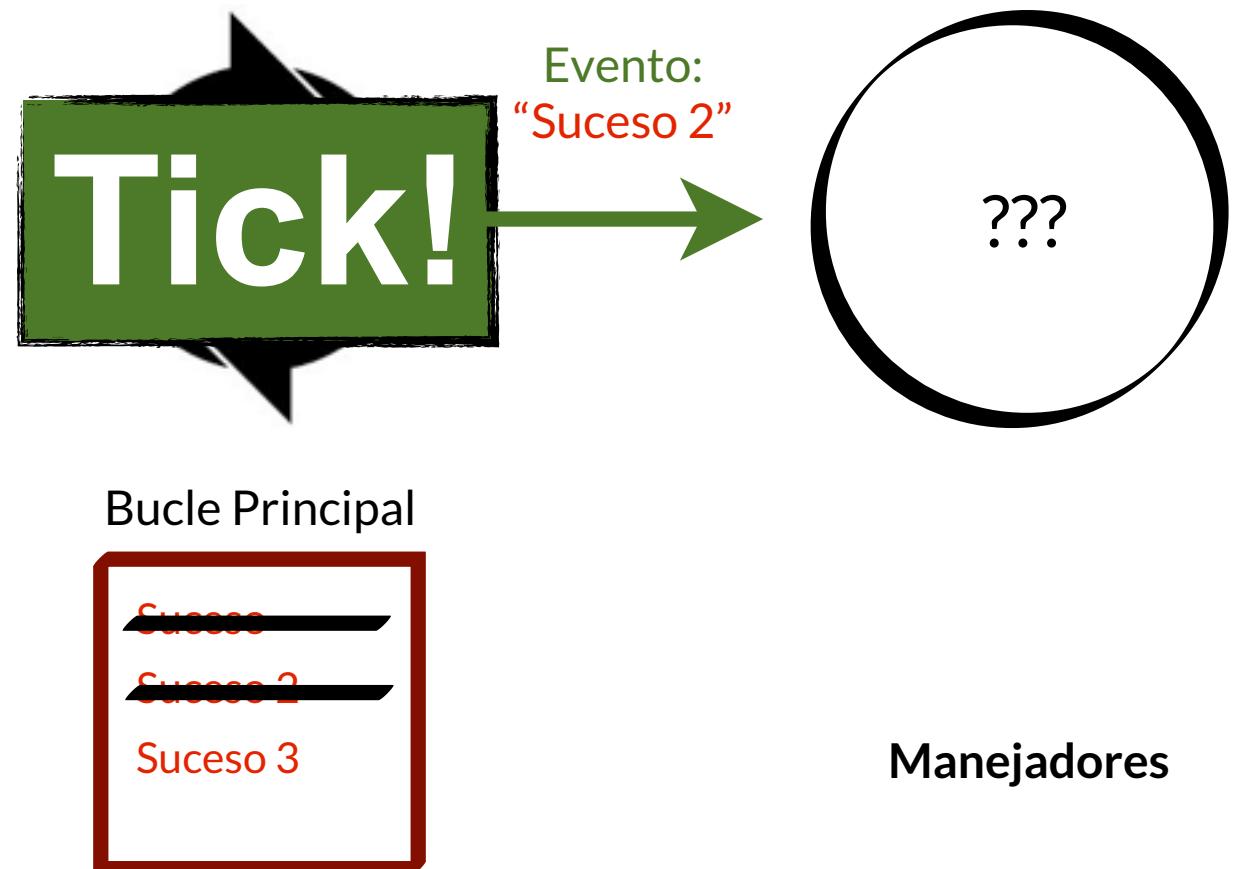
Manejadores

# Un modelo de ejecución

Patrón Reactor



Mundo Exterior



# Un modelo de ejecución

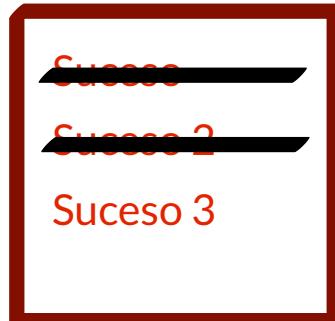
Patrón Reactor



Mundo Exterior



Bucle Principal



Manejadores

# Un modelo de ejecución

## Patrón Reactor

- Programación contra eventos
- Una vez puestos los manejadores, se pueden ejecutar en cualquier orden
  - El orden lo determina el orden en que aparezcan sucesos
- La ejecución de los manejadores **bloquea la hebra**
- Nunca hay dos manejadores ejecutándose al mismo tiempo
- Muy eficiente... cuando E/S >> ejecución del manejador

# ¿Qué es Node.js?

Node.js es algo más:

- Una filosofía sobre cómo hacer las cosas
- Un modelo de ejecución singular
- **Muy enfocado hacia aplicaciones de red**

# **¿Qué es Node.js?**

Muy enfocado hacia aplicaciones de red

- ¿Por qué?

# ¿Qué es Node.js?

Muy enfocado hacia aplicaciones de red

- Mucha E/S
  - Por tanto, mucho tiempo con la CPU inactiva
- Para aprovechar ese tiempo, necesitas otros clientes que lo puedan aprovechar
- Similar a un camarero en un bar
  - Es un “Patron Reactor” del mundo real
  - Para aprovecharlo, tiene que haber varios clientes!
  - Un cliente no termina más deprisa por dedicarle un camarero sólo a él

# **Toma de contacto**

# Ahora en serio: ¿Qué es Node.js?

Necesitas:

- Un ordenador
- Un editor de texto
- Saber abrir una consola/terminal
- Tener node instalado (mejor si es v. 0.10)
  - Asegúrate de tener también npm
  - Como alternativa: <http://c9.io>
- Saber manejarte con JavaScript

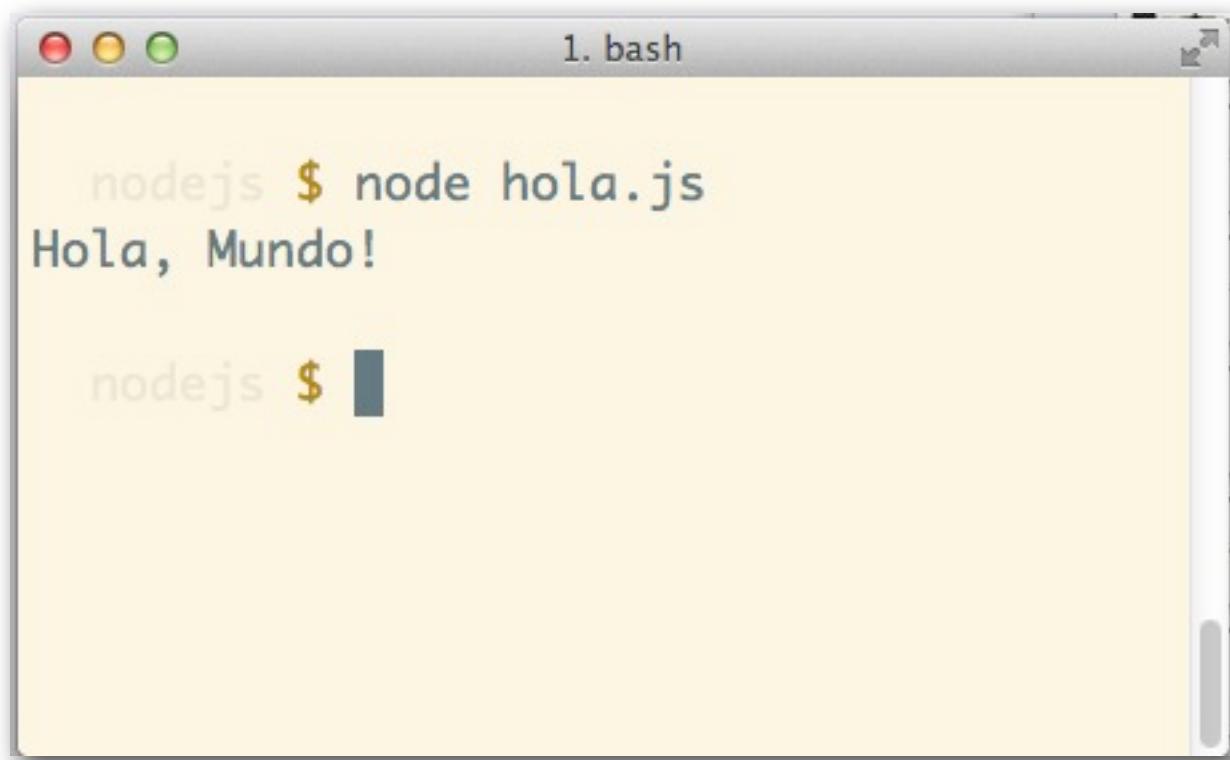
# Ahora en serio: ¿Qué es Node.js?

```
console.log("Hola, Mundo!");
```

# Ahora en serio: ¿Qué es Node.js?

```
$ node hola.js
```

# Ahora en serio: ¿Qué es Node.js?



```
nodejs $ node hola.js
Hola, Mundo!

nodejs $
```

# Ahora en serio: ¿Qué es Node.js?

Pero...

- ¿Y ese rollo del patrón Reactor?
- ¿Por qué termina el programa en vez de quedarse escuchando sucesos en el bucle principal?

# Ahora en serio: ¿Qué es Node.js?

Lo que pasa en realidad:

- Node.js ejecuta todo tu código del tirón
- Coloca los manejadores que hayas definido
- Si no hay **ningún manejador que se pueda ejecutar en el futuro**, el programa termina!

# Ahora en serio: ¿Qué es Node.js?

```
setTimeout(function() {  
  console.log("Hola, Mundo del futuro!");  
}, 1000);
```

# Ahora en serio: ¿Qué es Node.js?

```
setInterval(function() {  
  console.log("Hola otra vez, Mundo del futuro!");  
, 1000);
```

# Ahora en serio: ¿Qué es Node.js?

```
setTimeout(function() {  
  while (true);  
}, 100);  
  
setInterval(function() {  
  console.log("Hola otra vez, Mundo del futuro!");  
}, 1000);
```

# Ahora en serio: ¿Qué es Node.js?

```
var start = Date.now();

setTimeout(function() {
  for (var i=Number.MAX_VALUE; i--; ) {
    Math.pow(12345, 123455);
  }
}, 100);

setInterval(function() {
  var now = Date.now();
  console.log("Han pasado", now - start, "ms");
  start = now;
  console.log("Hola otra vez, Mundo del futuro!");
}, 1000);
```

# Ahora en serio: ¿Qué es Node.js?

```
var start = Date.now();

setTimeout(function() {
    var timesLeft = Number.MAX_VALUE,
        r;
    (function unPoquitoMas() {
        if (timesLeft-- > 0) { r = Math.pow(12345, 123455); }
        setTimeout(unPoquitoMas, 0);
    })();
}, 100);

setInterval(function() {
    var now = Date.now();
    console.log("Han pasado", now - start, "ms");
    start = now;
    console.log("Hola otra vez, Mundo del futuro!");
}, 1000);
```

# Ahora en serio: ¿Qué es Node.js?

Nos surgen problemas curiosos...

- ¿Excepciones?

# Ahora en serio: ¿Qué es Node.js?

Nos surgen problemas curiosos...

- ¿Excepciones?

```
try {
    throw new Error("Peté!");
} catch(e) {
    console.log("Excepción!");
}
```

# Ahora en serio: ¿Qué es Node.js?

Nos surgen problemas curiosos...

- ¿Excepciones?

```
try {
    setTimeout(function() {
        throw new Error("Peté!");
    }, 0);
} catch(e) {
    console.log("Excepción!");
}
```

# **EventEmitter**

Nuestro código va a estar dirigido por eventos

- Node.js tiene su propio “estándar”
- Trae una implementación del patrón Observador (o Pub/Sub): EventEmitter
- Todas sus librerías (y casi todos los paquetes) siguen este modelo

# EventEmitter

```
var EventEmitter = require("events").EventEmitter;  
  
var pub = new EventEmitter();  
  
pub.on("ev", function(m) {  
    console.log("[ev]", m);  
});  
  
pub.once("ev", function(m) {  
    console.log("(ha sido la primera vez)");  
});  
  
pub.emit("ev", "Soy un Emisor de Eventos!");  
pub.emit("ev", "Me vas a ver muy a menudo...");
```

# EventEmitter

```
var EventEmitter = require("events").EventEmitter;
```

```
var pub = new EventEmitter();
```

```
pub.on("ev", function(m) {  
    console.log("[ev]", m);  
});
```

```
pub.once("ev", function(m) {  
    console.log("(ha sido la primera vez)");  
});
```

```
pub.emit("ev", "Soy un Emisor de Eventos!");  
pub.emit("ev", "Me vas a ver muy a menudo...");
```

# **require/exports**

`require(<paquete o ruta>)`

- Importar módulos (paquetes, otros ficheros)
- Garantía: una única vez
- Devuelve el módulo!

# **require/exports**

`exports.propiedadPublica = <valor>`

- El otro lado del mecanismo
- Se puede exportar cualquier valor

# require/exports

codigo.js

```
var lib = require("./libreria");
console.log(lib.propiedad);
```

libreria.js

```
console.log("una vez");
exports.propiedad = "Pública";
```

# Para que te confíes...

Haz un módulo “reloj”

- Que exporte una clase Reloj
- Emite eventos “segundo”, “minuto” y “hora”

```
var Reloj = require("./reloj").Reloj;
var reloj = new Reloj();

reloj.on("segundo", function(fecha) {
  console.log("Un segundo! son las:", fecha);
  reloj.removeAllListeners("segundo");
});
```

# Para que te confíes...

Un truco:

- `require("util").inherits`
- `inherits(constructor, superConstructor)`

```
var inherits = require("util").inherits;

function MiClase() {
    // ...
}

function MiSubClase() {
    // ...
}
inherits(MiSubClase, MiClase);
```

# JavaScript y el Universo

El JavaScript del navegador vive en un mundo ideal

- No hay SO con el que lidiar
- No hay datos binarios
- Todo es accesible con objetos y valores primitivos
- Apenas hay E/S, siempre con valores simples (strings)
- Un único usuario

# JavaScript y el Universo

En Node.js, las cosas son de otra manera...

- Llamadas al sistema
- Mucha E/S
- Datos binarios (ficheros, sockets, etc)
- Descriptores de ficheros
- Puertos
- ...

# JavaScript y el Universo

Tenemos que añadir nuevos conceptos a nuestro JS

- Streams (lectura, escritura o duplex)
- Buffers (representación de datos binarios)
- Procesos
- Rutas
- <http://nodejs.org/api/index.html>

# JavaScript y el Universo

## Buffers

- Una tira de bytes (datos binarios)
- Similar a un array de enteros
- Tamaño fijo
- Manipular datos directamente
  - Sockets
  - Implementar protocolos complejos
  - Manipulación de ficheros/imágenes
  - Criptografía
  - ...

# JavaScript y el Universo

## Buffers

```
var buf = new Buffer(100);
buf.write("abcd", 0, 4, "ascii");

console.log(buf.toString("ascii"));
```

# JavaScript y el Universo

Buffers

Tamaño del buffer



```
var buf = new Buffer(100);
buf.write("abcd", 0, 4, "ascii");

console.log(buf.toString("ascii"));
```

# JavaScript y el Universo

Buffers

Posición

Datos

Longitud

Codificación

```
var buf = new Buffer(100);
buf.write("abcd", 0, 4, "ascii");

console.log(buf.toString("ascii"));
```

# JavaScript y el Universo

## Buffers

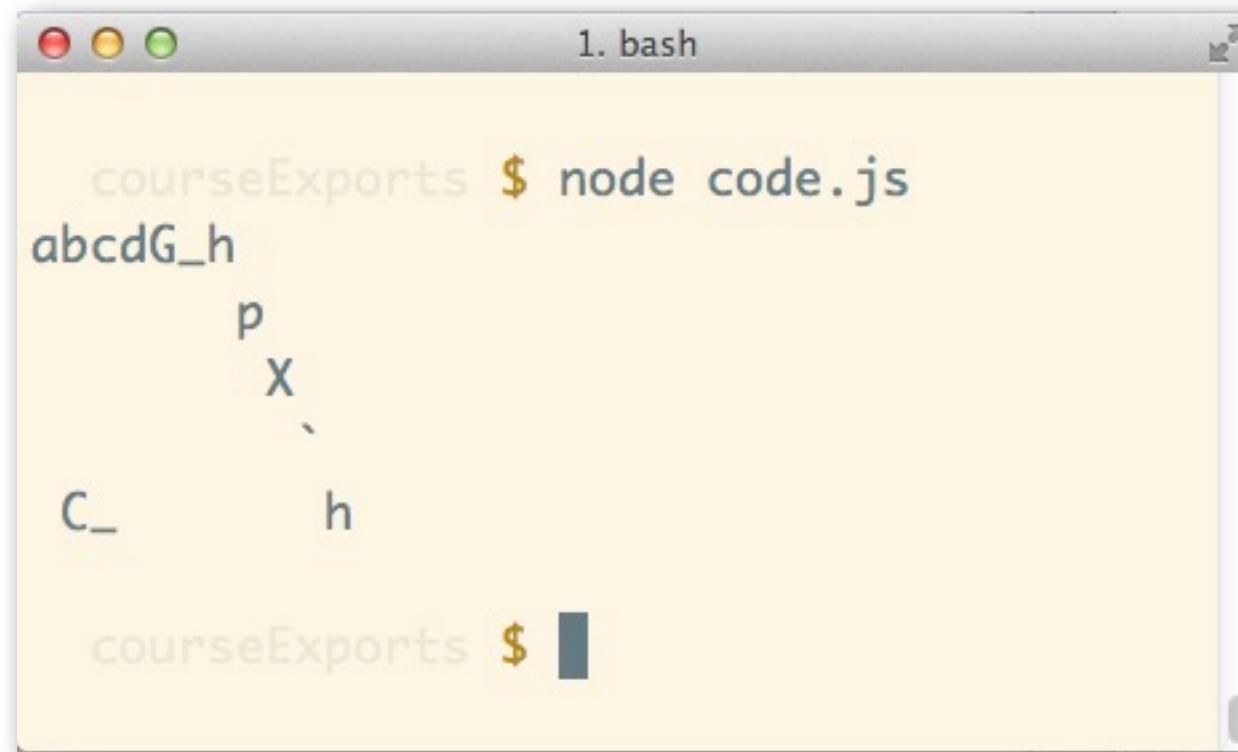
```
var buf = new Buffer(100);
buf.write("abcd", 0, 4, "ascii");

console.log(buf.toString("ascii"));
```



# JavaScript y el Universo

## Buffers



A screenshot of a macOS terminal window titled "1. bash". The window has the standard OS X title bar with red, yellow, and green buttons. The main pane displays the following text:

```
courseExports $ node code.js
abcdG_h
p
x
`_
C_      h
courseExports $
```

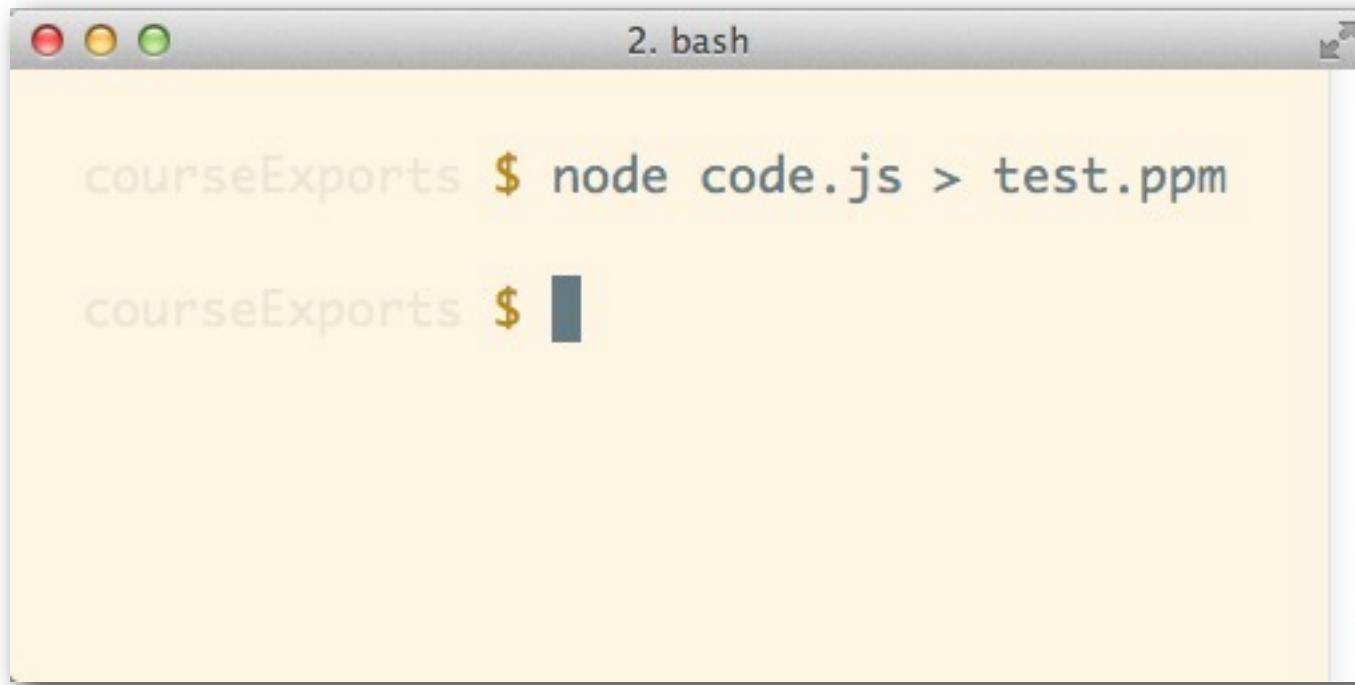
The text is displayed in a light gray font on a white background. The cursor is positioned at the end of the last line, indicated by a small blue square.

# JavaScript y el Universo

```
function Bitmap(w, h) {
  this.width = w;
  this.height = h;
  this.header = "P6\n" + w + " " + h + "\n255\n";
  this.buffer = new Buffer(w*h*3+this.header.length);
  this.buffer.write(this.header, 0, this.header.length, "ascii");
}
Bitmap.prototype = {
  putPixel: function(x, y, color) {
    var pos = this.header.length + (y*this.width*3) + x*3;
    this.buffer.write(color, pos, 3, "hex");
  },
  fill: function(color) {
    this.buffer.fill(255, this.header.length);
  },
  render: function() {
    process.stdout.write(this.buffer);
  }
};
```

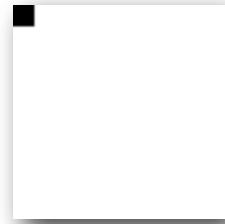
# JavaScript y el Universo

```
var bitmap = new Bitmap(1, 1);
bitmap.putPixel(0, 0, "000000");
bitmap.render();
```



# JavaScript y el Universo

```
var bitmap = new Bitmap(10, 10);
bitmap.fill("ffffff");
bitmap.putPixel(0, 0, "000000");
bitmap.render();
```



# JavaScript y el Universo

## Streams

- “Chorros” de información
  - Lectura / Escritura / Duplex
- Detrás de muchos mecanismos de Node.js
  - stdin/stdout
  - request HTTP
  - sockets
  - etc...
- Instancias de **EventEmitter**
- Acceso asíncrono

# JavaScript y el Universo

## Streams

- Es raro crear streams directamente
- Pero muchos recursos nos ofrecen este interfaz

# JavaScript y el Universo

## Streams de lectura

- Entrada de datos
- Eventos:
  - **readable**: hay datos para leer
  - **data**: se ha leído un trozo y está disponible
  - **end**: se agotó el stream
  - **close**: se cerró el stream
  - **error**: algo malo sucedió leyendo los datos

# JavaScript y el Universo

## Streams de lectura

```
var fs = require("fs");

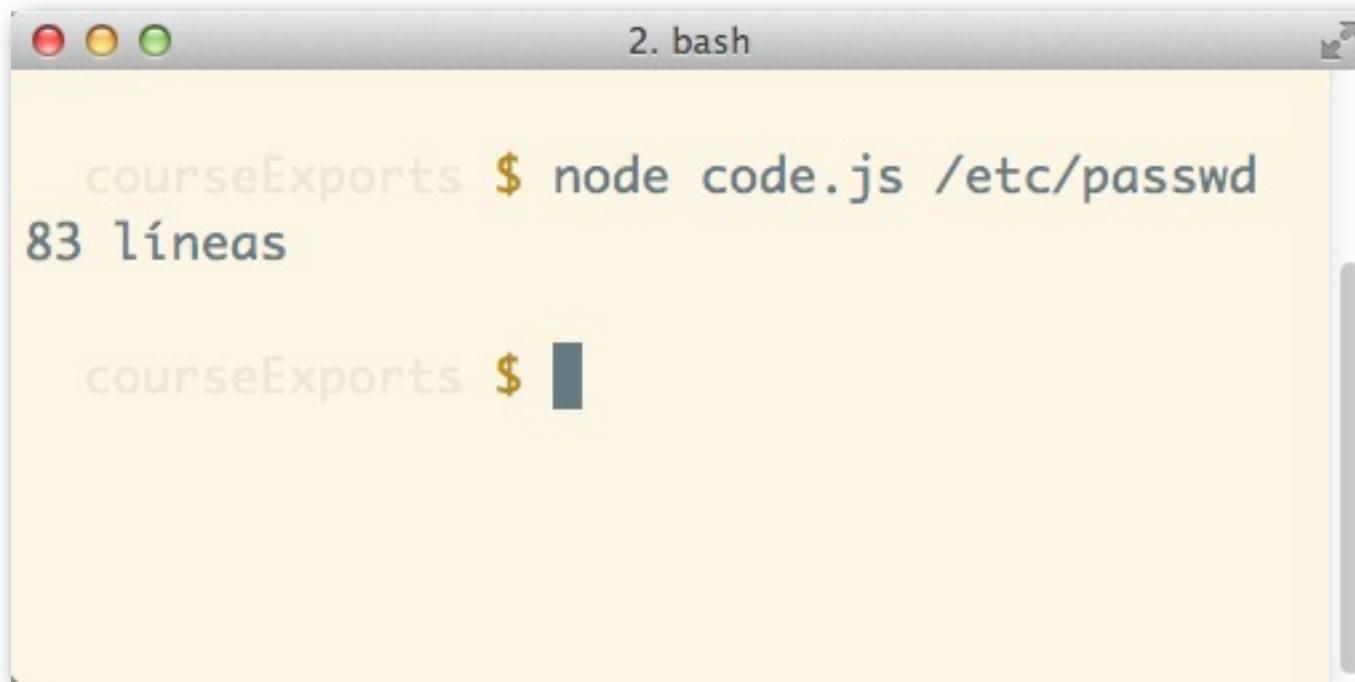
var readStream = fs.createReadStream("/etc/passwd", {
  flags: "r",
  encoding: "ascii",
  autoClose: true
});

readStream.on("data", function(chunk) {
  console.log("He leído:", chunk.length);
});

readStream.on("end", function() {
  console.log("ya está!");
});
```

# Una fácil

Haz un programa que cuente las líneas de un fichero



The screenshot shows a terminal window with a light gray background and a dark gray title bar. The title bar has three colored window control buttons (red, yellow, green) on the left and a close button on the right. The title itself is "2. bash". The main area of the terminal contains the following text:

```
courseExports $ node code.js /etc/passwd
83 líneas

courseExports $
```

The text is displayed in a monospaced font. The first line starts with "courseExports" followed by a dollar sign and "node code.js /etc/passwd". The second line contains the number "83" followed by the Spanish word "líneas". The third line starts with "courseExports" followed by a dollar sign.

# Una fácil

Haz un programa que cuente las líneas de un fichero

- Tienes los argumentos con los que se ha llamado al programa en `process.argv`

# JavaScript y el Universo

## Streams de escritura

- Salida de datos
- Operaciones:
  - `write(chunk, [encoding], [callback])`
  - `end([chunk], [encoding], [callback])`
- Eventos:
  - **drain**: el buffer del stream está vacío (puedes escribir más)
  - **finish**: se ha terminado de escribir toda la info y se ha cerrado el stream

# JavaScript y el Universo

## Streams de escritura

```
var fs = require("fs");

var writeStream = fs.createWriteStream(process.argv[2], {
  flags: "w",
  encoding: "utf-8"
});

for (var i=100; i--;) {
  writeStream.write(i + " líneas más para terminar...\\n");
}
writeStream.end("FIN");

writeStream.on("finish", function() {
  console.log("Listo!");
});
```

# **¿Preguntas?**

Un buen momento para despejar dudas antes de seguir...

# **HTTP**

(por fin...)

# HTTP

Node.js trae un servidor web estupendo

- Asíncrono
  - No bloquea la hebra
  - Cada cliente conectado consume muy poquitos recursos
  - Genial para miles de conexiones simultáneas
- Relativamente rápido
- Interfaz sencilla
- HTTP puro y duro, sin adornos
- Basado en streams y eventos

# HTTP

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

var server = http.createServer();

server.on("request", function(req, res) {
  res.end("Hola, Mundo!");
});

server.listen(3000);
```

# HTTP

El módulo



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

var server = http.createServer();

server.on("request", function(req, res) {
  res.end("Hola, Mundo!");
});

server.listen(3000);
```

# HTTP

```
var http = require("http");
var server = http.createServer();
server.on("request", function(req, res) {
  res.end("Hola, Mundo!");
});
server.listen(3000);
```

Eventos!



Respuesta  
(stream)



Request  
(objeto)



# HTTP

## El servidor HTTP

- Eventos:
  - connection
  - request
- Operaciones:
  - `createServer([requestCallback])`
  - `listen(puerto, [hostname], [backlog], [callback])`
  - `close([callback])`

# HTTP

## `http.IncomingMessage` (parametro “req”)

- Representa la petición HTTP del cliente
- Propiedades:
  - `req.headers`: cabeceras de la petición
  - `req.method`: verbo HTTP
  - `req.url`: url de la petición
  - `req.connection.remoteAddress`: ip del cliente

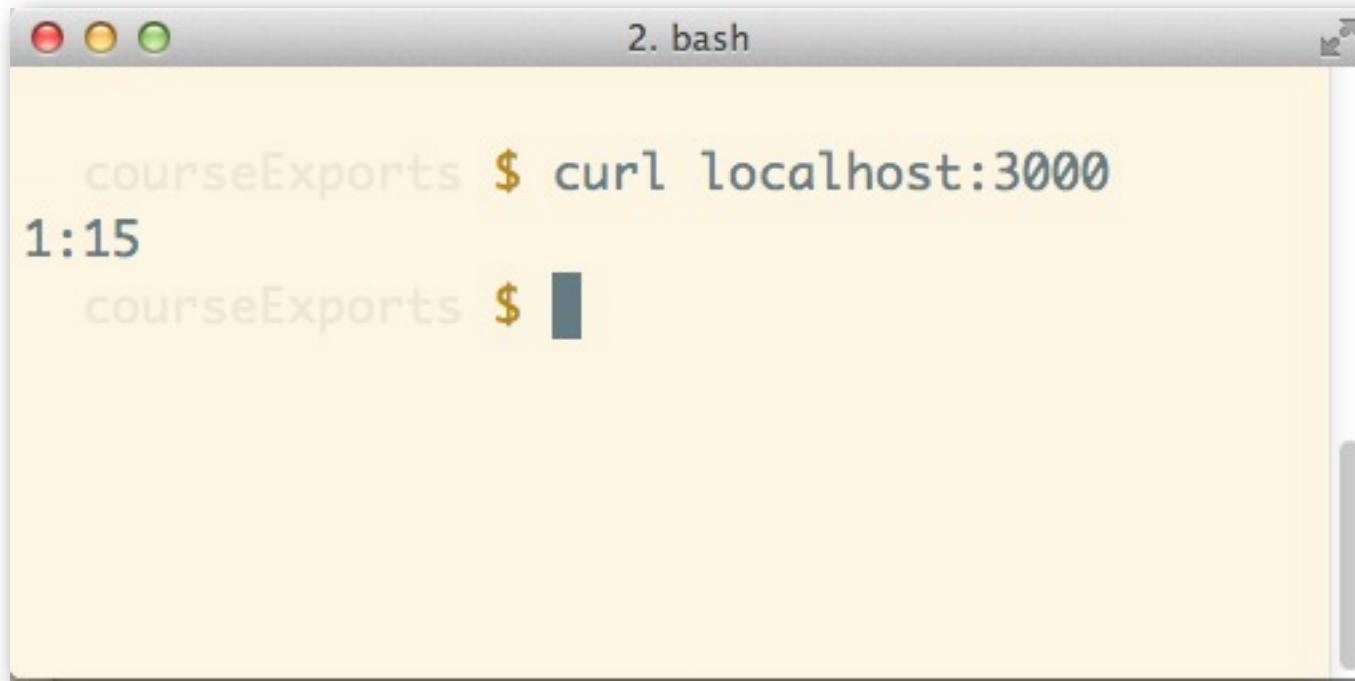
# HTTP

## `http.ServerResponse` (parametro “res”)

- Representa la respuesta del servidor
- Stream de escritura
- Operaciones adicionales:
  - `res.writeHead(statusCode, [headers])`: código HTTP y cabeceras de la respuesta
  - `res.statusCode`: [propiedad] Otra forma de establecer el código HTTP de la respuesta
  - `res.setHeader(name, value)`: Otra forma de establecer las cabeceras, de una en una

# Manos a la obra

Escribe un servidor web que devuelva la hora



```
courseExports $ curl localhost:3000
1:15
courseExports $ █
```

# Un consejo: nodemon

Para mejorar el flow de trabajo:

- Editar, matar el proceso, volver a lanzarlo, probar... muy tedioso!

- Instálate **nodemon**

```
npm install -g nodemon
```

- Reinicia el servidor cada vez que cambia el fichero

```
$ nodemon <fichero.js>
```

# HTTP

Node.js trae un módulo para parsear URLs

```
var http = require("http"),
    url = require("url"),
    inspect = require("util").inspect;

var server = http.createServer();

server.on("request", function(req, res) {
    var urlData = url.parse(req.url, true);
    res.end(inspect(urlData, {colors: false}));
});

server.listen(3000);
```

# HTTP

Node.js trae un módulo para parsear URLs

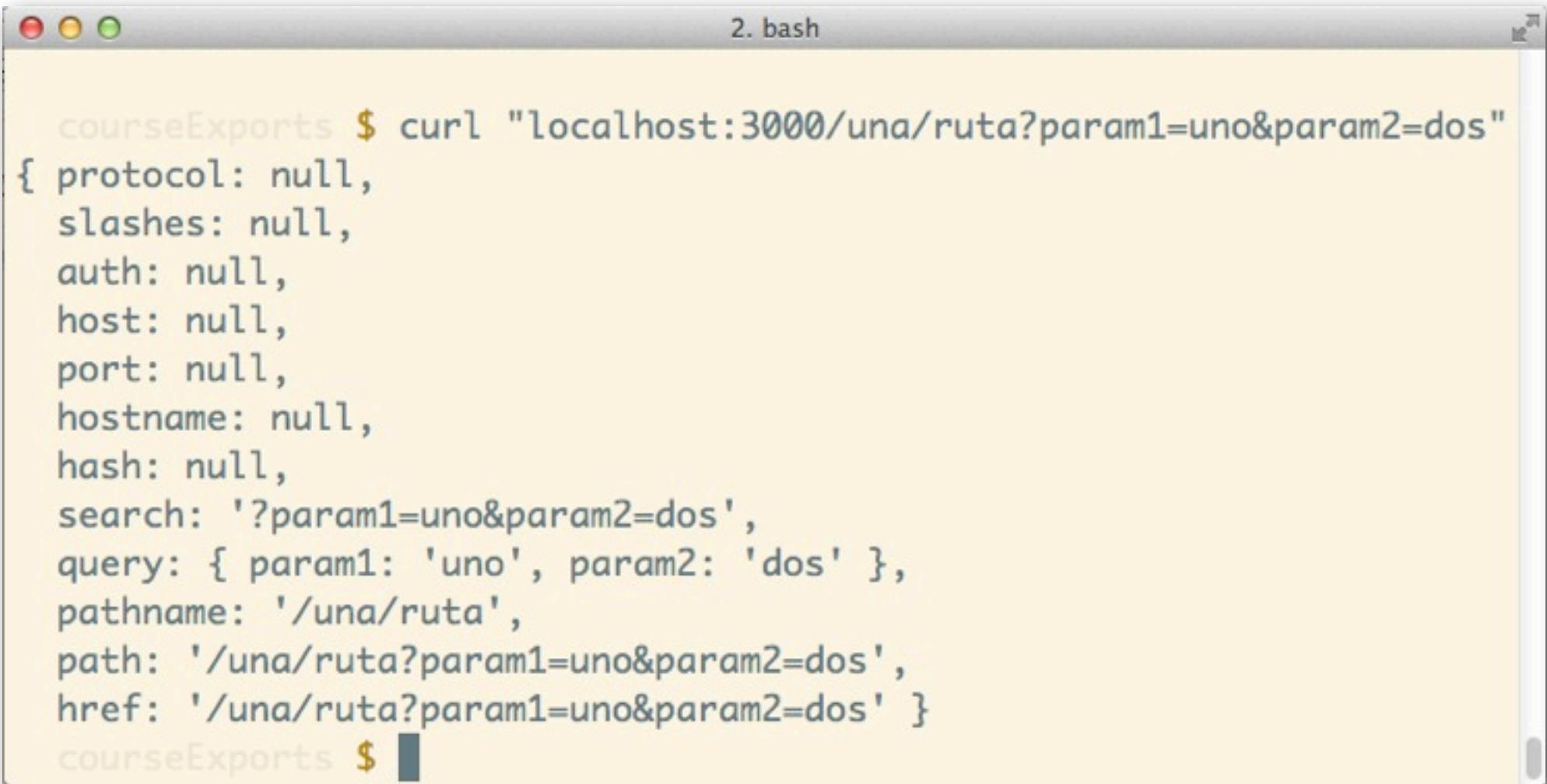
```
var http = require("http"),
    url = require("url"),
    inspect = require("util").inspect;

var server = http.createServer();

server.on("request", function(req, res) {
    var urlData = url.parse(req.url, true);
    res.end(inspect(urlData, {colors: false}));
});

server.listen(3000);
```

# HTTP



The screenshot shows a terminal window with a light gray background and a dark gray title bar. The title bar has three colored window control buttons (red, yellow, green) on the left and a close button on the right. The main area of the terminal is titled '2. bash'. The terminal displays the following text:

```
courseExports $ curl "localhost:3000/una/ruta?param1=uno&param2=dos"
{ protocol: null,
  slashes: null,
  auth: null,
  host: null,
  port: null,
  hostname: null,
  hash: null,
  search: '?param1=uno&param2=dos',
  query: { param1: 'uno', param2: 'dos' },
  pathname: '/una/ruta',
  path: '/una/ruta?param1=uno&param2=dos',
  href: '/una/ruta?param1=uno&param2=dos' }
courseExports $
```

# **Un poco más difícil**

Escribe un servidor de ficheros

- Lee el pathname de la URL
- Busca un fichero con esa ruta dentro de ./public
- Si existe, lo sirve
- Si no existe, devuelve 404

# Un poco más difícil: notas

`fs.exists(filePath, callback)`

- Llama al callback con `true` si el fichero filePath existe
- O con `false` si no existe

```
var fs = require("fs");

fs.exists("./hola.txt", function(exists) {
  console.log(exists);
});
```

# Un poco más difícil: notas

`fs.readFile(filePath, callback)`

- Otra manera de leer ficheros
- Intenta leer filePath e invoca a callback con dos parámetros:
  - `err`: null si todo ha ido bien o, si hubo error, el error
  - `data`: todo el contenido del fichero (si fue posible leerlo)

```
var fs = require("fs");

fs.readFile("./hola.txt", function(err, data) {
  if (err) { /* error! */ }
  console.log(data);
});
```

# **Un poco más difícil: epílogo**

¿Cómo podríamos añadir un caché para no leer los ficheros del disco duro más de una vez?

¿Cómo podríamos hacer que los ficheros cacheados se liberaran después de x minutos?

¿Cómo podríamos escribir un registro de acceso?

# **Un poco más difícil: variaciones**

Haz un contador de aperturas de emails

- Sirviendo un .gif de 1x1 y contando cuantas veces lo sirves
- Mejor aún, cuenta solo a cuántas IPs lo sirves

Haz un servicio de avatares que cambie según:

- La hora del día
- La frecuencia con que es pedido (popularidad)

# **Un poco más difícil: variaciones**

Haz un “servidor hellban”

Si la IP está en la lista negra, todas las peticiones tienen un delay aleatorio que se va incrementando con cada petición consecutiva

# **Servidor A/B Testing**

Cada vez que un cliente pide un recurso:

- Se comprueba si ya tiene caso asignado (por IP) o se le asigna uno
- Se construye la ruta del recurso según el caso asignado y se sirve
- Si pide “success.png”, se marca su caso como exitoso y se le sirve la imagen
- /stats devuelve un JSON con info sobre los casos (visitas totales y visitas exitosas por cada caso)

# **Servidor A/B Testing**

Tenéis maqueta y recursos en :

`/tema1/abtesting`