Event Loop
& Code Execution

Event Execution

Referencias: en el material adjunto

builder.io - EventLoop

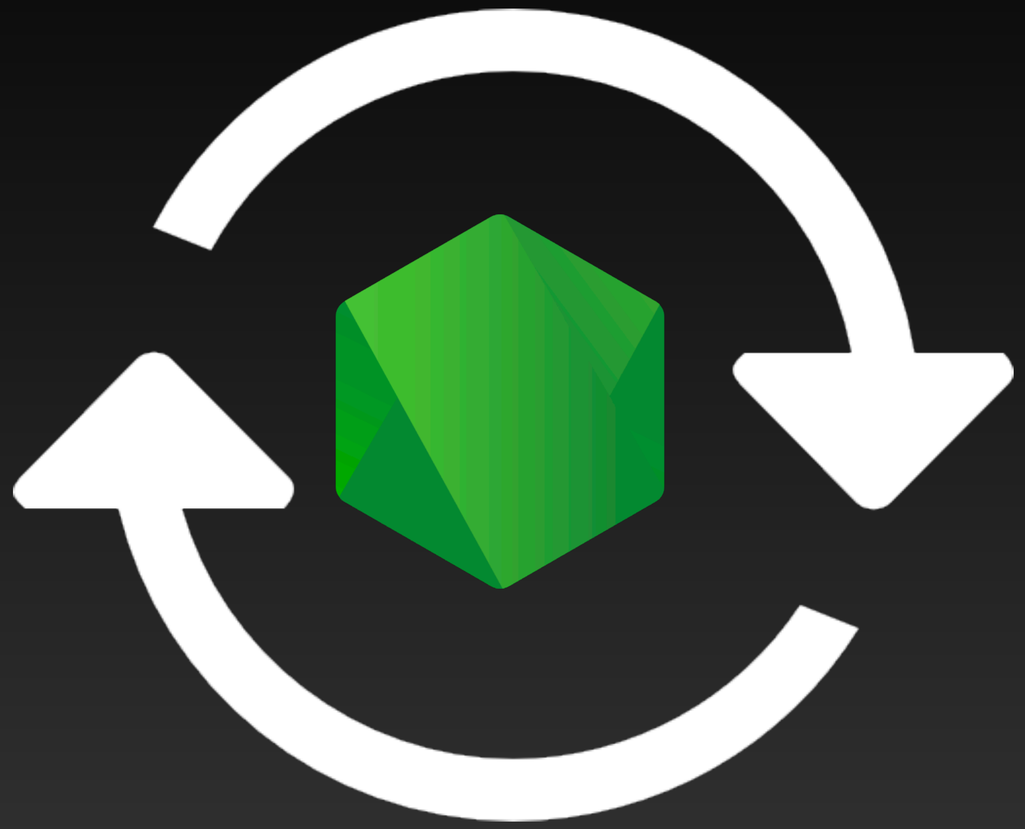
nodejs.org The Node.js Event Loop, Timers, and process.nextTick( )

nodes.org Blocking vs non-blocking

```
setTimeout( cb1 , 0);

Promise.resolve().then( cb2 );

process.nextTick( cb3 );

fs.readFile( 'file.txt' , cb4 );

setImmediate( cb5 );
```

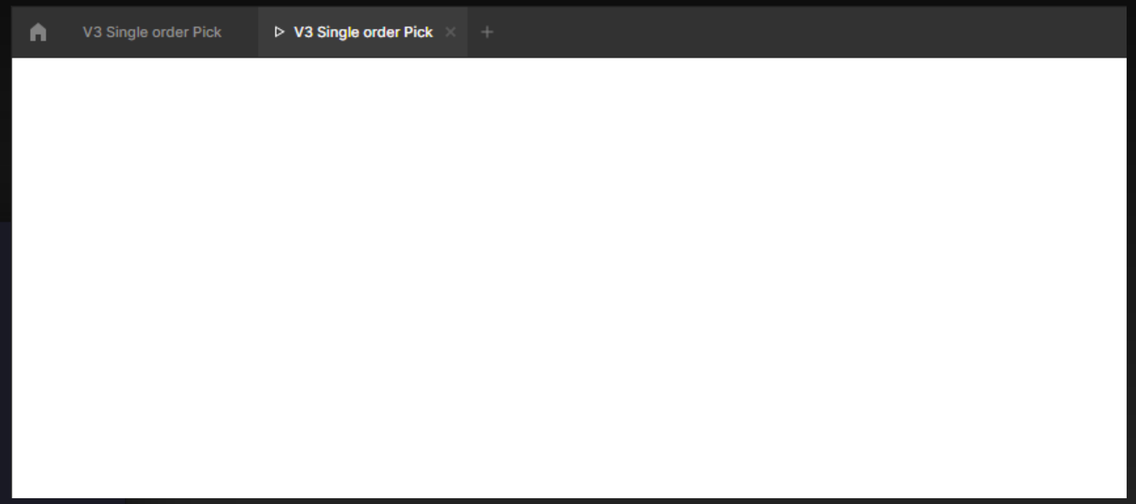
¿Quién se ejecutará primero?

¿No lo sé?



Pero es el **Event Loop** quien decide

Blocking & Single-threaded

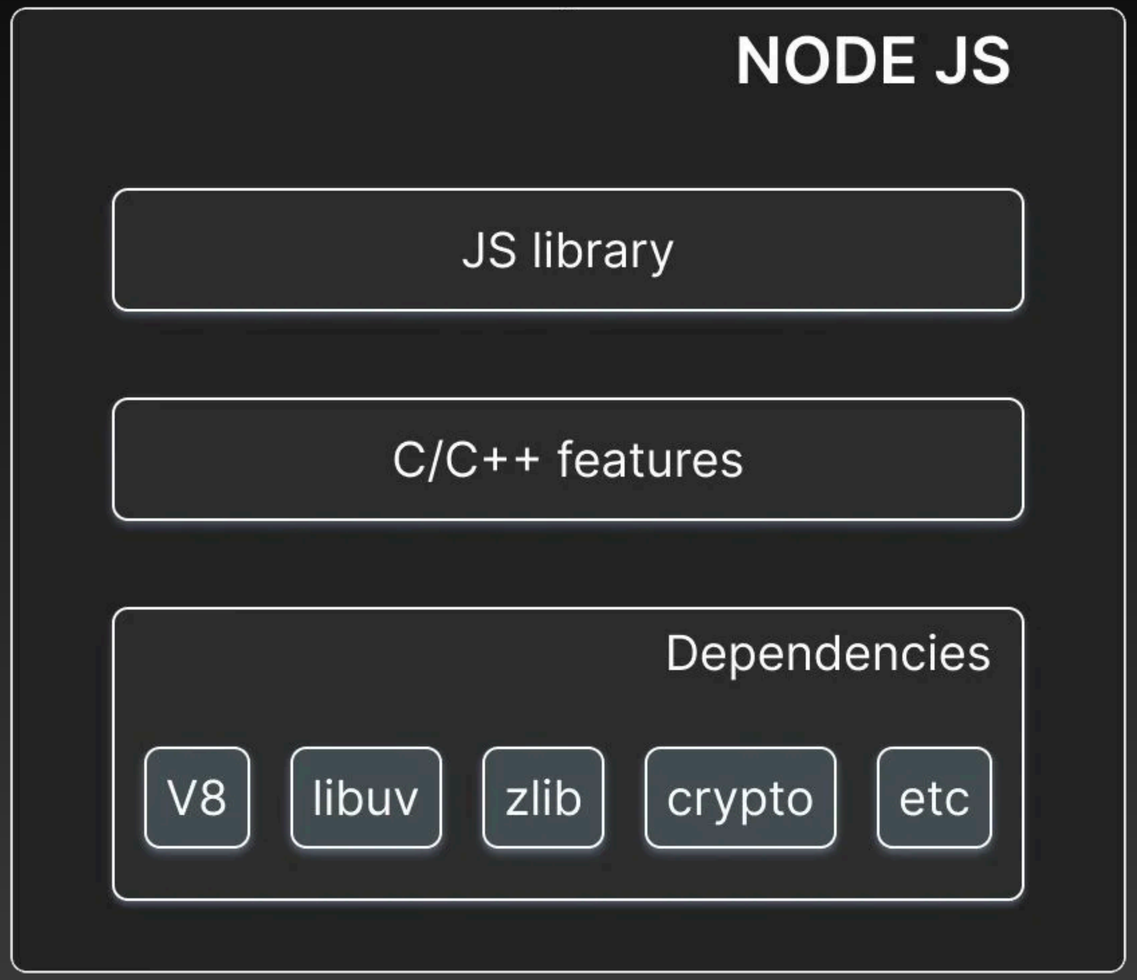



```
while (true) {
  // procedimiento
}

moreWork(); // <-- Nunca se ejecutará
```

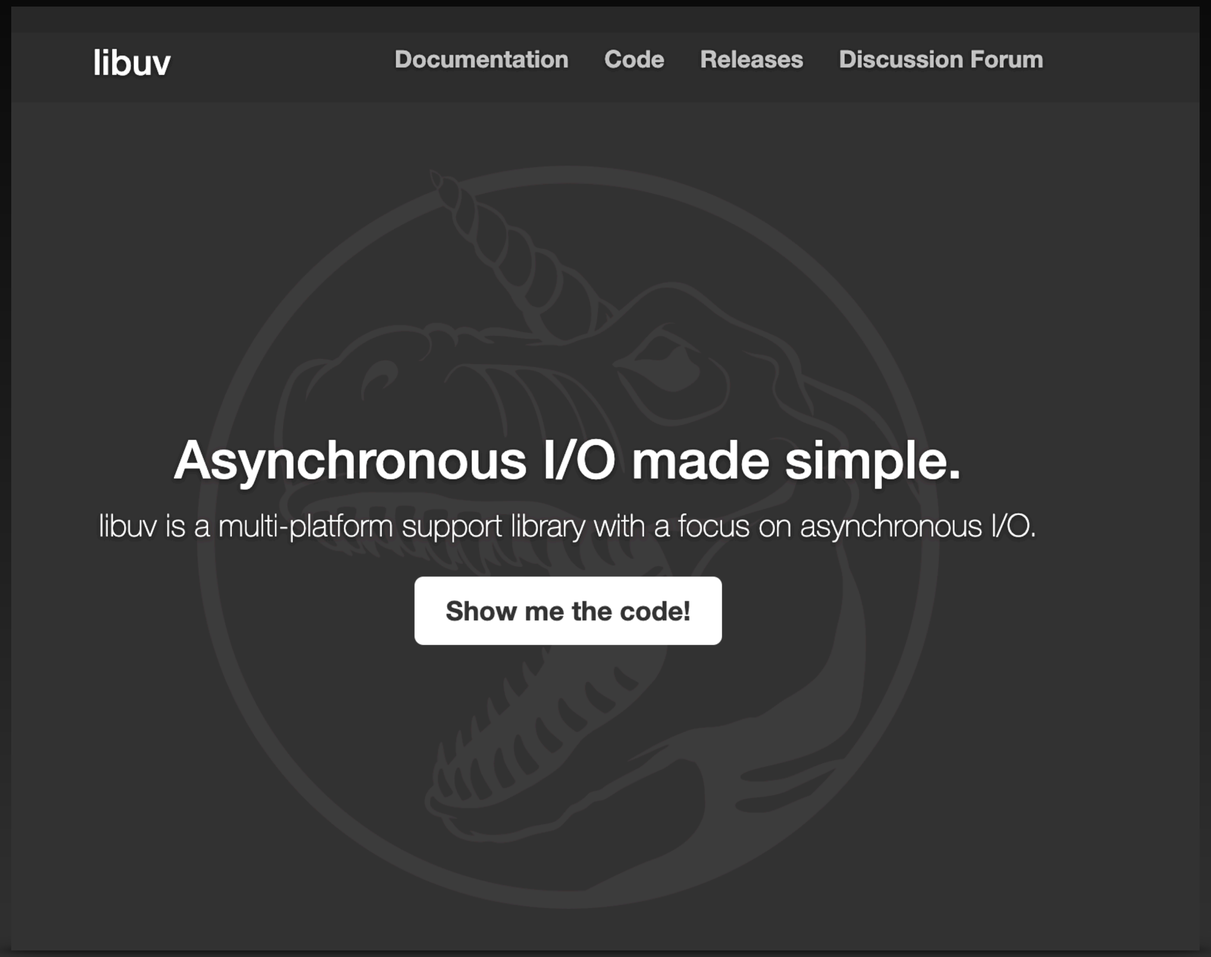
Porque el while bloquea la ejecución

3 componentes principales de Node

Tu código (app.js)



- **Dependencias externas**
- **Características de C++**
- **Librerías de JS que se conectan con C++ desde nuestro código**



Su role principal es manejar toda las operaciones asíncronas

Ejecución de código

```
const message = 'Hola mundo';

console.log(message);

console.log(message.toUpperCase());
```

Consola

```
➜  ~
```

Call Stack

Libuv

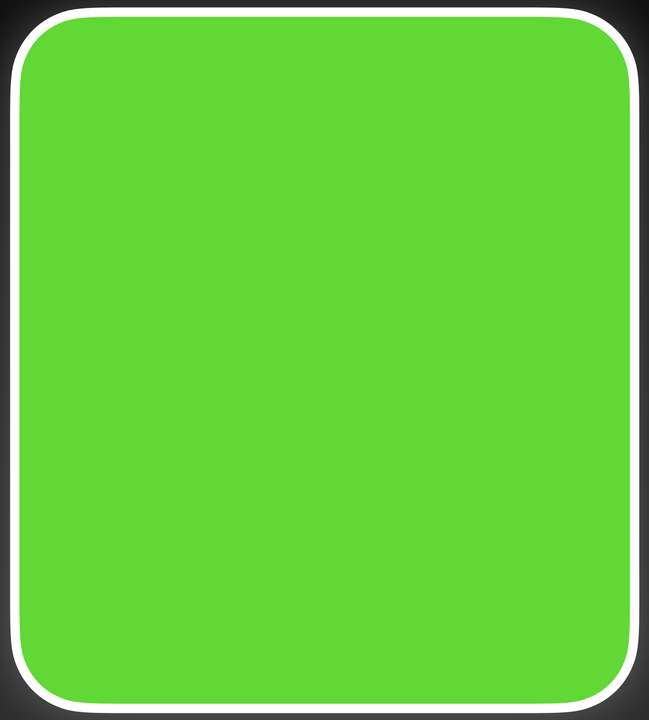
Ejecución de código

```
const message = 'Hola mundo';

console.log(message);

console.log(message.toUpperCase());
```

Consola

```
➜  ~
```

Call Stack



Libuv

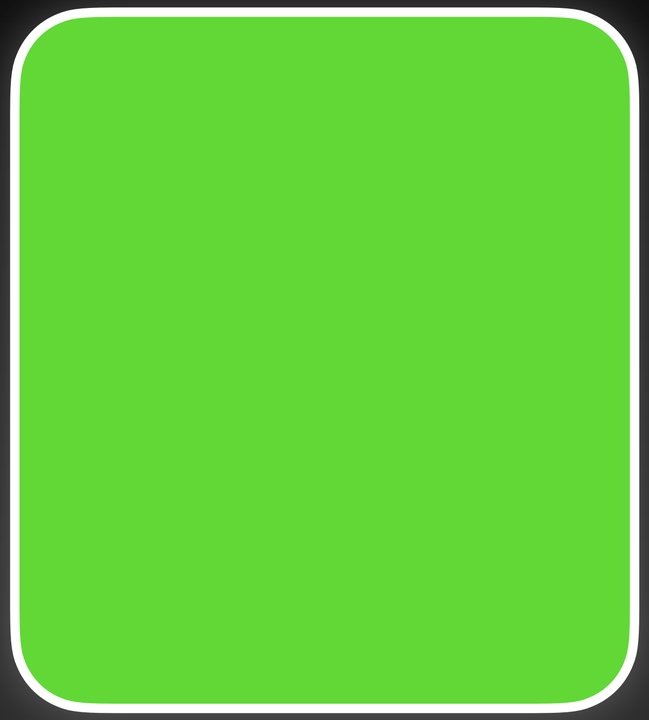
Ejecución de código

```
const message = 'Hola mundo';

console.log(message);

console.log(message.toUpperCase());
```

Consola

```
➜  ~
```

Call Stack



Libuv

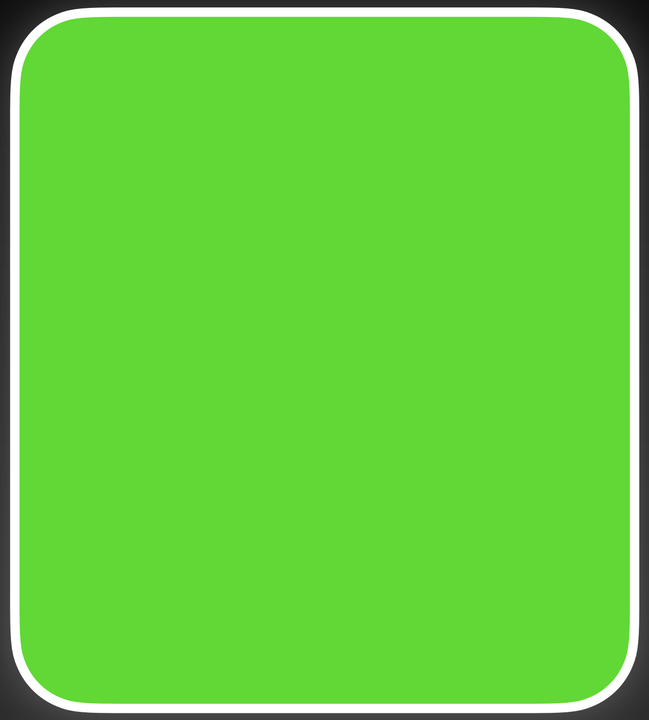
Ejecución de código

```
const message = 'Hola mundo';

console.log(message);

console.log(message.toUpperCase());
```

Consola

```
➜  ~
```

Call Stack



Libuv

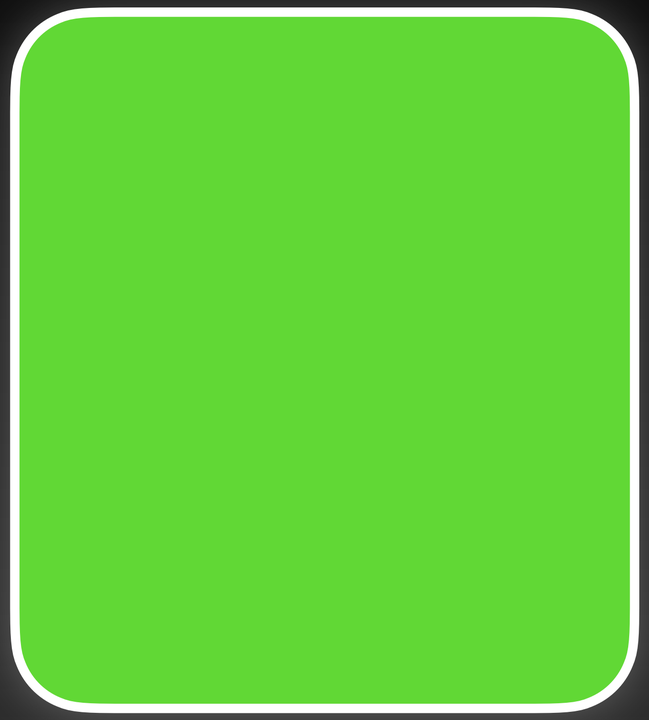
Ejecución de código

```
const message = 'Hola mundo';

console.log(message);

console.log(message.toUpperCase());
```

Consola

```
➜  ~
```

Call Stack



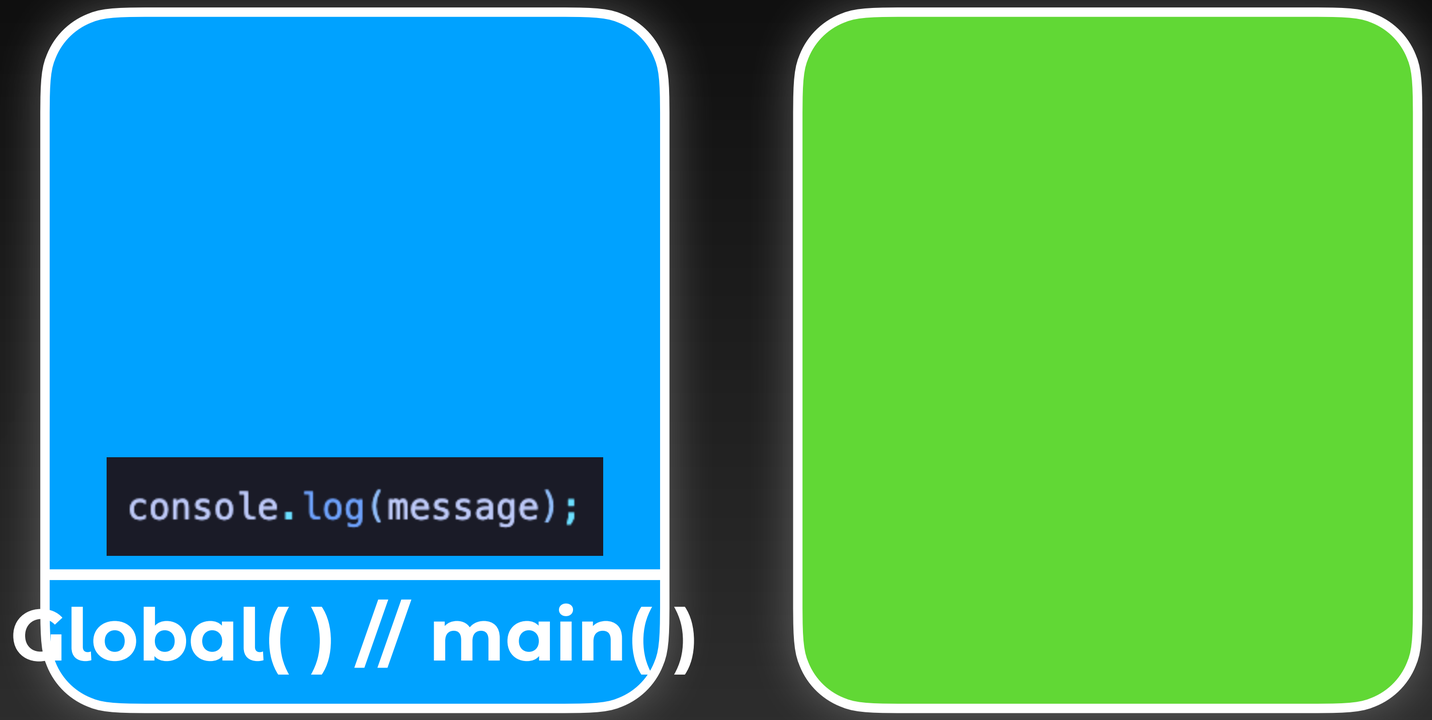
Libuv

Ejecución de código

```
const message = 'Hola mundo';

console.log(message);

console.log(message.toUpperCase());
```

Consola

```
→ ~
Hola Mundo
```



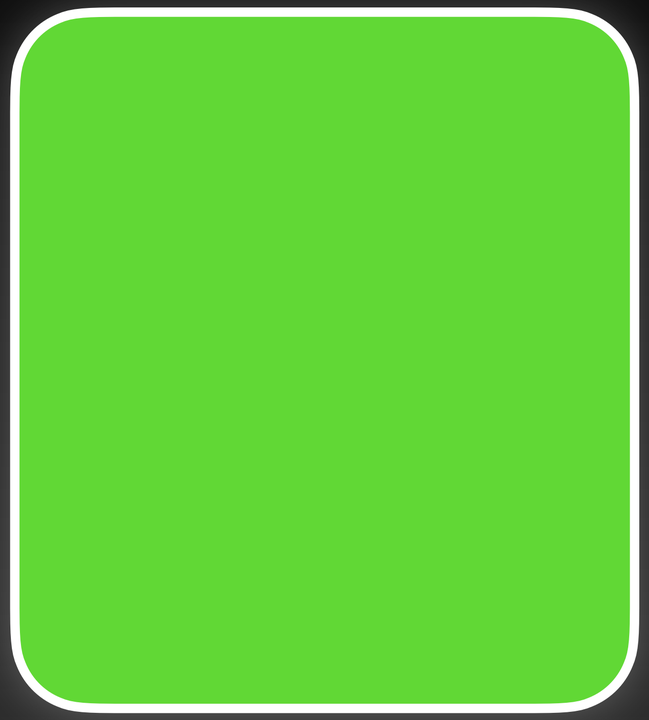
Ejecución de código

```
const message = 'Hola mundo';

console.log(message);

console.log(message.toUpperCase());
```

Consola

```
→ ~
Hola Mundo
```

Call Stack



Libuv

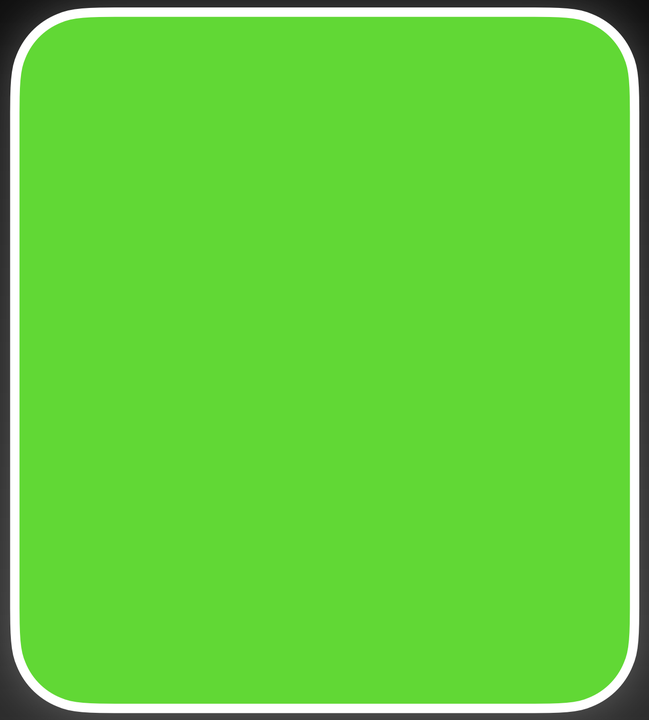
Ejecución de código

```
1
2  const message = 'Hola mundo';
3
4  console.log(message);
5
6  console.log(message.toUpperCase());
7
8
```

Consola

```
→  ~
Hola Mundo
```

Call Stack



Libuv

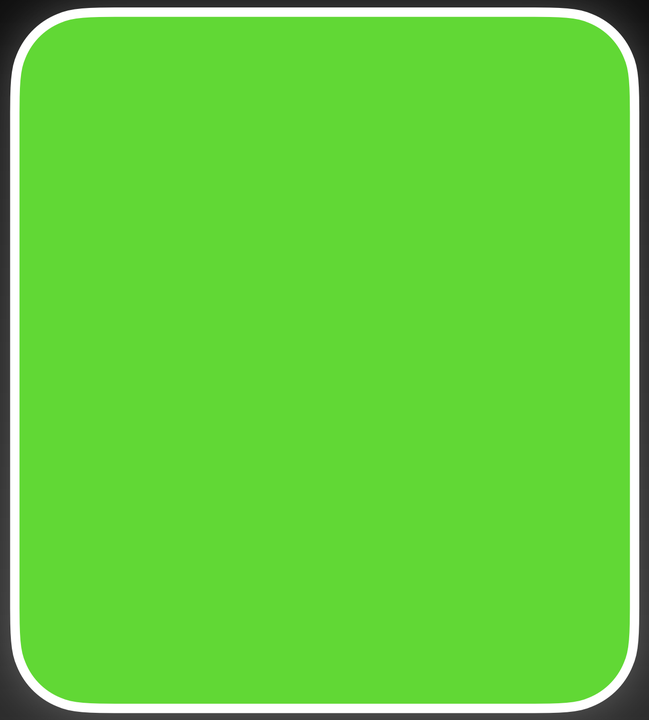
Ejecución de código

```
1
2  const message = 'Hola mundo';
3
4  console.log(message);
5
6  console.log(message.toUpperCase());
7
8
```

Consola

```
→  ~
Hola Mundo
HOLA MUNDO
```

Call Stack



Libuv

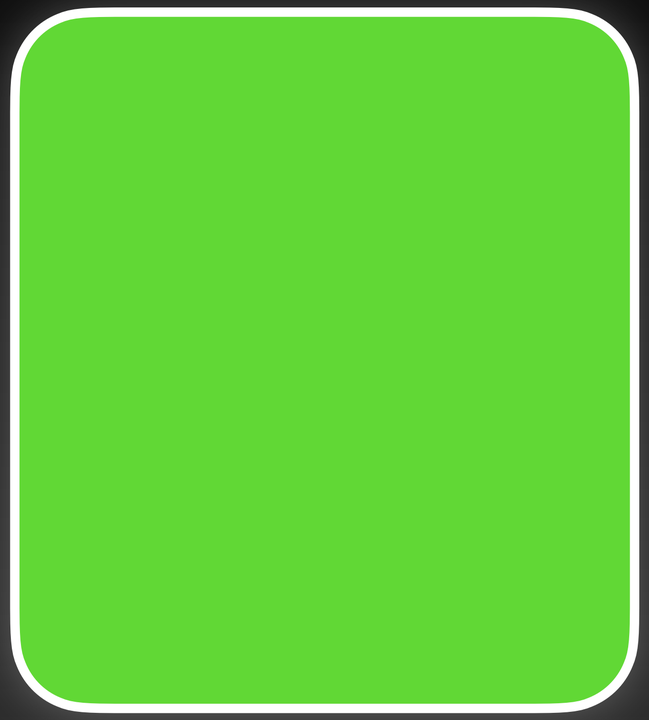
Ejecución de código

```
const message = 'Hola mundo';

console.log(message);

console.log(message.toUpperCase());
```

Consola

```
→ ~
Hola Mundo
HOLA MUNDO
```

Call Stack



Libuv

Ejecución de código

```
1
2  const message = 'Hola mundo';
3
4  console.log(message);
5
6  console.log(message.toUpperCase());
7
8
```

Consola

```
➜  ~
Hola Mundo
HOLA MUNDO
```

Call Stack

Exit code 0

Libuv

Ejecución de código

Ejemplo #2

```
1
2  const fs = require("fs");
3
4  fs.readFile('file.md', (err, data) => {
5    console.log(data);
6  });
7
8  console.log('Fin de programa');
```

Consola

```
➜  ~
```

Call Stack

Libuv

Ejecución de código

Ejemplo #2

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
➜  ~
```

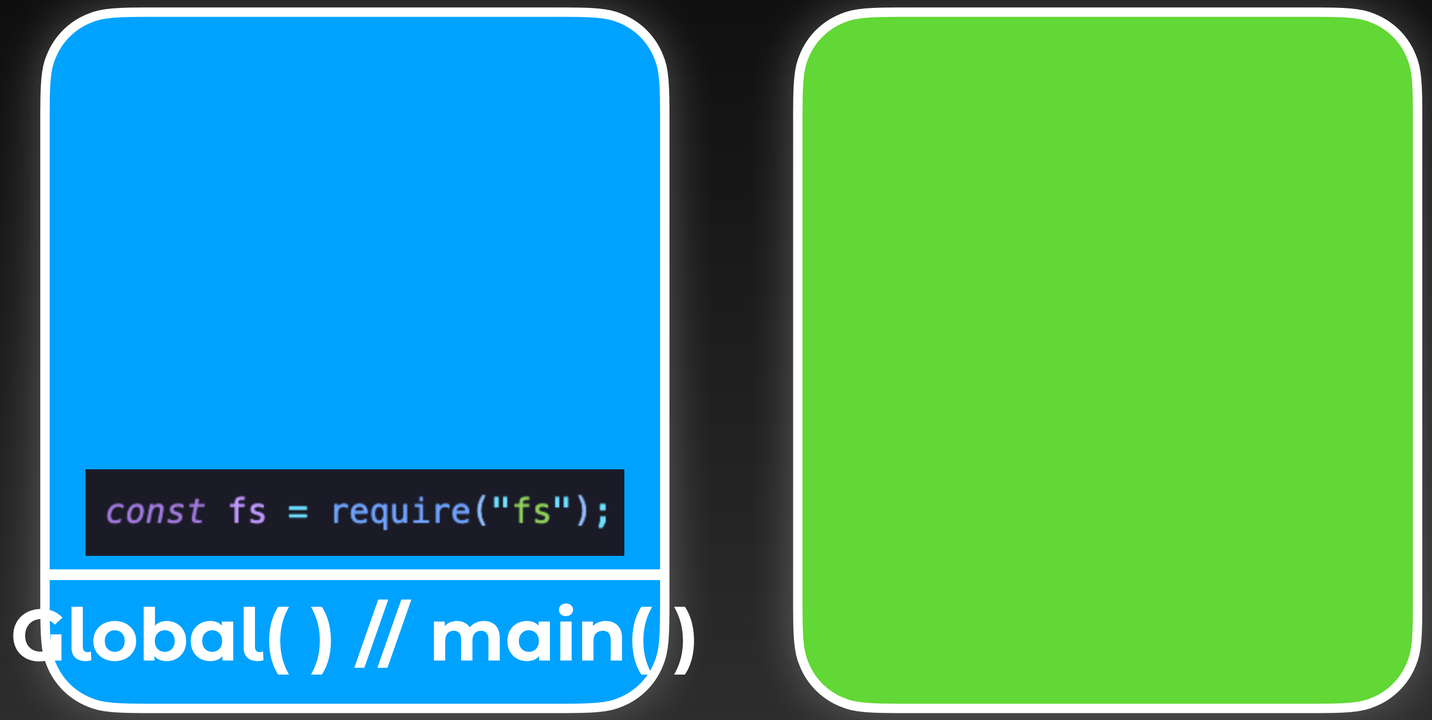
Ejecución de código

Ejemplo #2

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
➜  ~
```



Ejecución de código

Ejemplo #2

```
1
2  const fs = require("fs");
3
4  fs.readFile('file.md', (err, data) => {
5    console.log(data);
6  });
7
8  console.log('Fin de programa');
```

Consola

```
➜  ~
```

Call Stack

Libuv

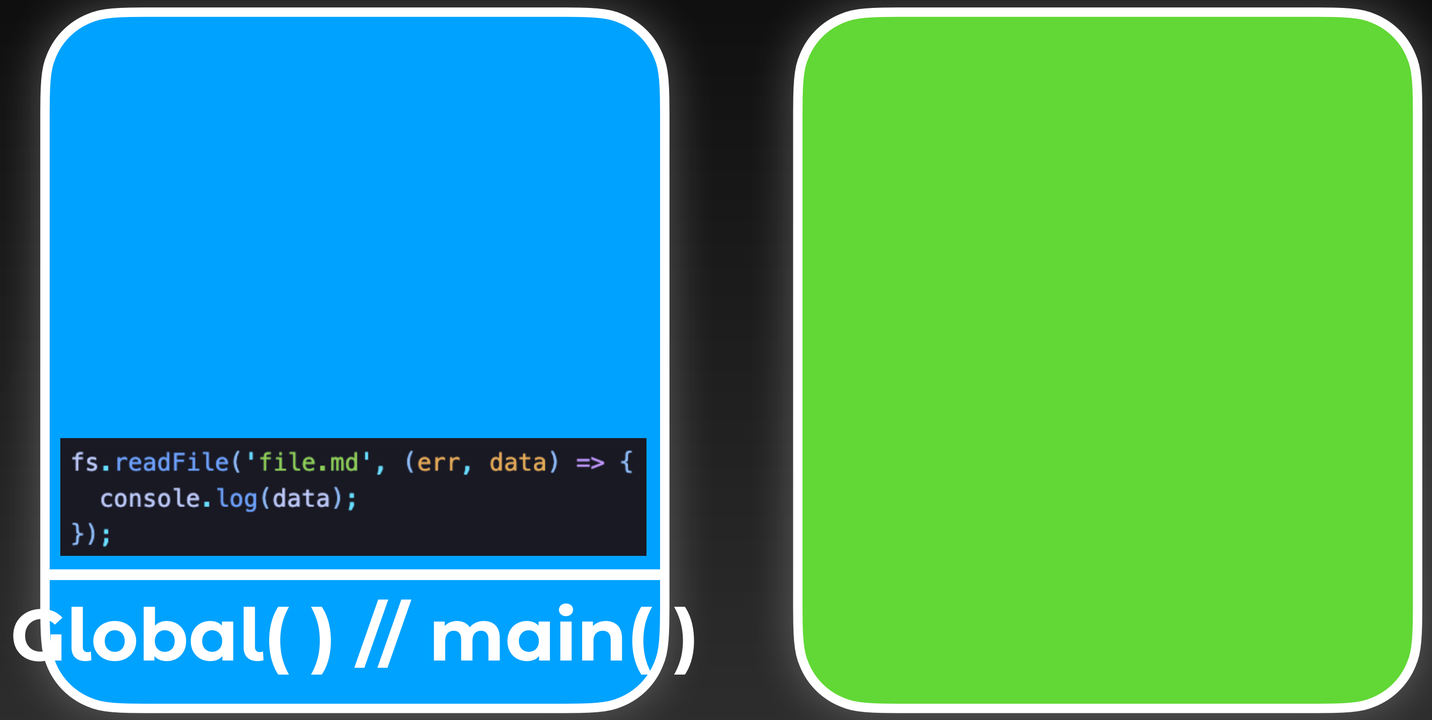
Ejecución de código

Ejemplo #2

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
➜  ~
```



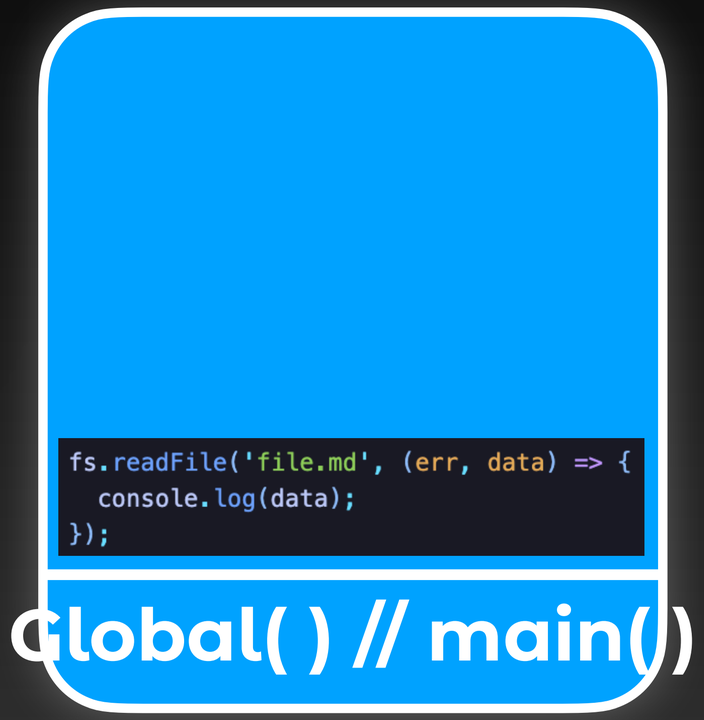
Ejecución de código

Ejemplo #2

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
➜  ~
```



Call Stack

Libuv

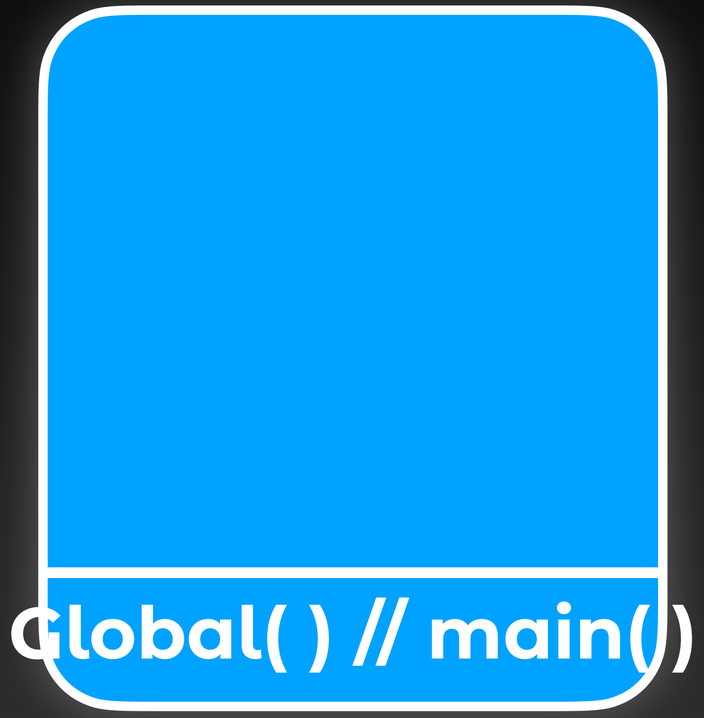
Ejecución de código

Ejemplo #2

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
➜  ~
```



Call Stack

```
(err, data) => {
  console.log(data);
}
```

Libuv

Ejecución de código

Ejemplo #2

node

cución síncrona primero

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
➜  ~
```

```
console.log('Fin de programa');
```

Global( ) // main( )

Call Stack

```
(err, data) => {
  console.log(data);
}
```

Libuv

Ejecución de código

Ejemplo #2

node

cución síncrona primero

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
→ ~
Fin de programa
```

```
console.log('Fin de programa');
```

Global( ) // main( )

Call Stack

```
(err, data) => {
  console.log(data);
}
```

Libuv

Ejecución de código

Ejemplo #2

node

cución síncrona primero

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
→ ~
Fin de programa
```

Global( ) // main( )

Call Stack

```
(err, data) => {
  console.log(data);
}
```

Libuv

Ejecución de código

Ejemplo #2

node

cución síncrona primero

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
→ ~
Fin de programa
```

```
(err, data) => {
  console.log(data);
}
```

Global( ) // main( )

Call Stack

Libuv

Ejecución de código

Ejemplo #2

node

cución síncrona primero

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
→  ~
Fin de programa
....data
```

```
(err, data) => {
  console.log(data);
}
```

Global( ) // main( )

Call Stack

Libuv

Ejecución de código

Ejemplo #2

node

cución síncrona primero

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
→  ~
Fin de programa
 ....data
```

Global( ) // main( )

Call Stack

Libuv

Ejecución de código

Ejemplo #2

node

cución síncrona primero

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
→  ~

Fin de programa
 ....data
```

Call Stack

Libuv

Ejecución de código

Ejemplo #2

node

cución síncrona primero

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

fs.readFile('file.md', (err, data) => {
  console.log(data);
});

console.log('Fin de programa');
```

Consola

```
→  ~
Fin de programa
....data
```

Call Stack

Exit code 0

Libuv

Ejecución de código

Ejemplo #3

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
➜  ~
```

Consola

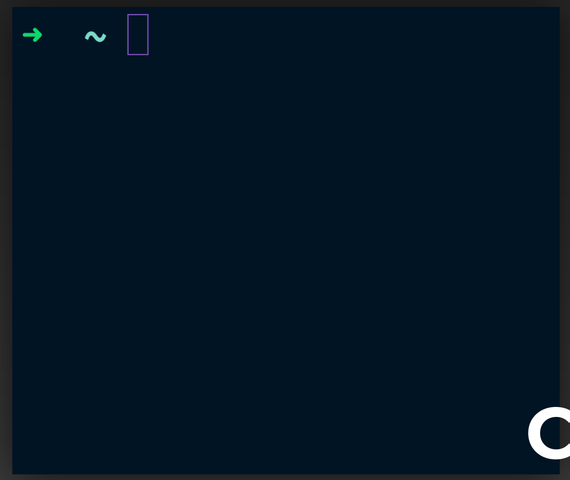
Call Stack

Libuv

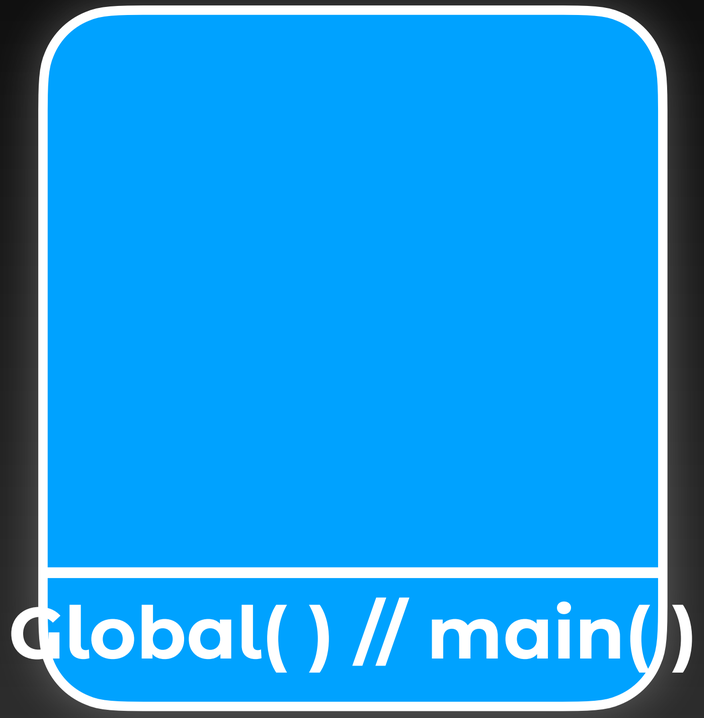
Ejecución de código

Ejemplo #3

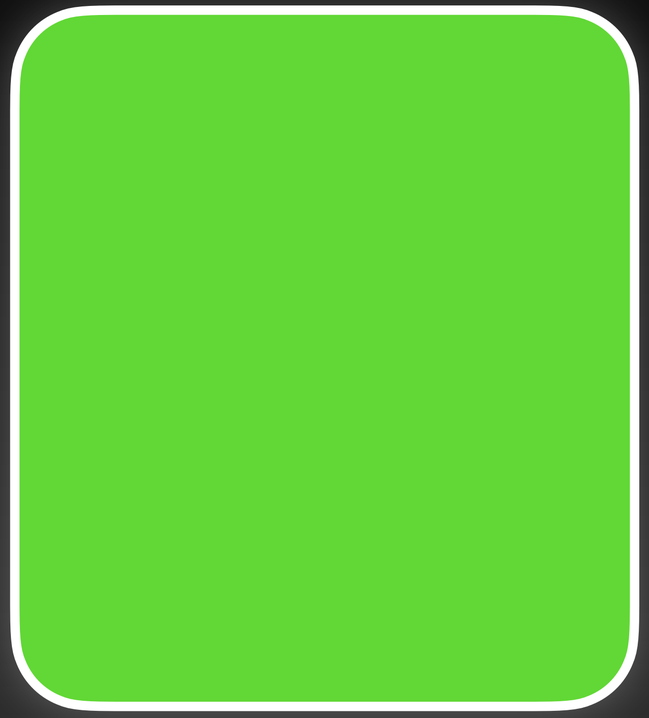
```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```



Consola



Call Stack



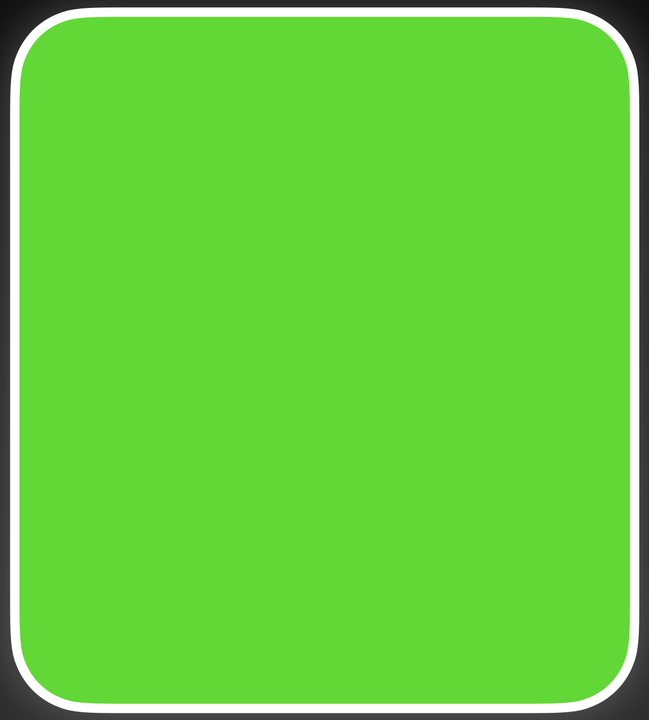
Libuv

Ejecución de código

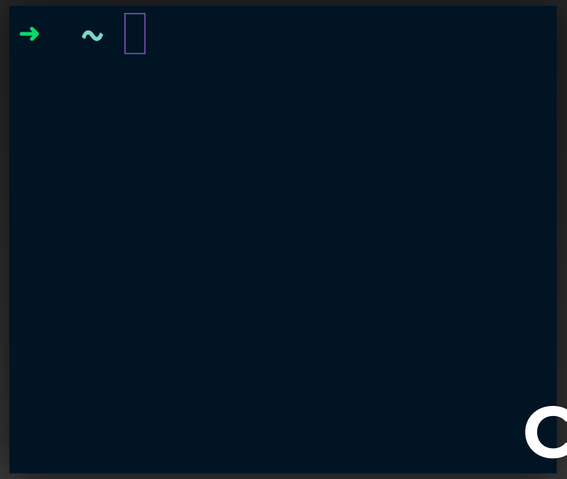
Ejemplo #3

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

Call Stack



Libuv



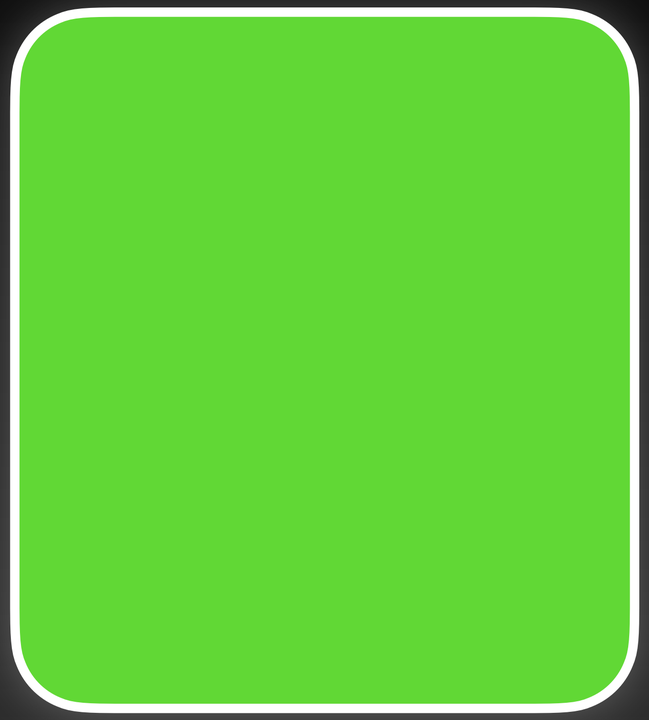
Consola

Ejecución de código

Ejemplo #3

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

Call Stack



Libuv

```
→  ~
Inicio de programa
```

Consola

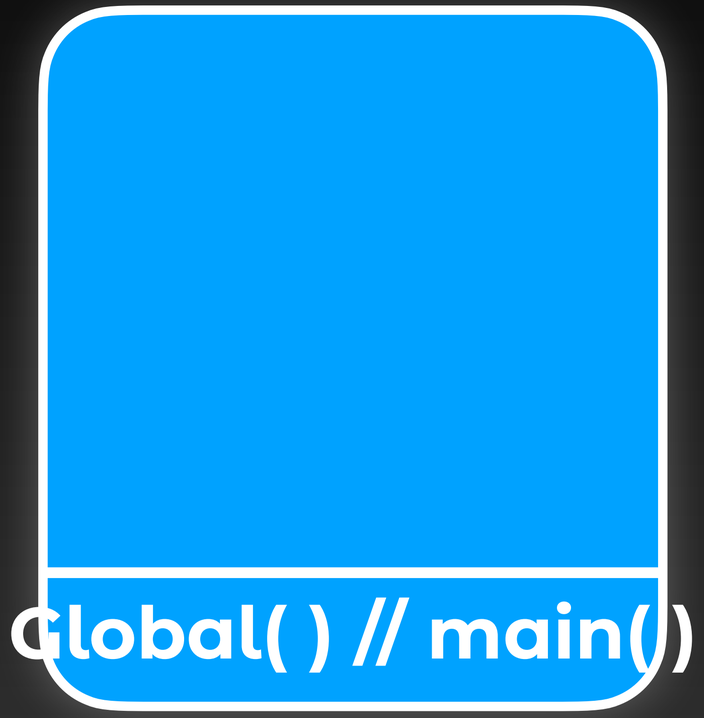
Ejecución de código

Ejemplo #3

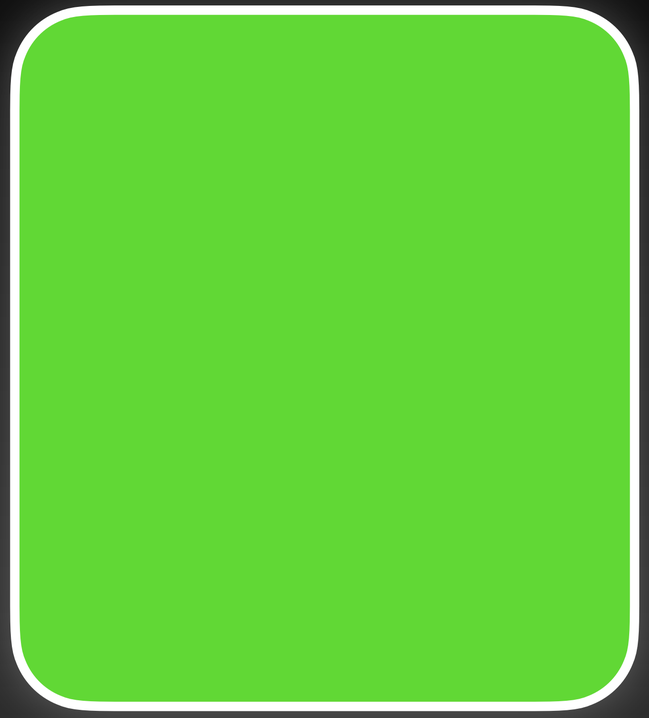
```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
```

Consola



Call Stack



Libuv

Ejecución de código

Ejemplo #3

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );
```

Global( ) // main( )

Call Stack

Libuv

```
→  ~
Inicio de programa
```

Consola

Ejecución de código

Ejemplo #3

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
```

Consola

```javascript
setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );
```

Global( ) // main( )

Call Stack

```javascript
() => {
  console.log('Primer Timeout');
}
```

Libuv

Ejecución de código

Ejemplo #3

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
➜  ~
Inicio de programa
```

Consola

Global( ) // main( )

Call Stack

```javascript
() => {
  console.log('Primer Timeout');
}
```

3s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );
```

Global( ) // main( )

Call Stack

```
() => {
  console.log('Primer Timeout');
}
```

3s

Libuv

```
→  ~
Inicio de programa
```

Consola

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );
```

Global( ) // main( )

Call Stack

```
() => {
    console.log('Segundo Timeout');
}
```

0s

```
() => {
    console.log('Primer Timeout');
}
```

3s

Libuv

```
→  ~
Inicio de programa
```

Consola

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→ ~
Inicio de programa
```

Consola

Global( ) // main( )

Call Stack

```javascript
() => {
    console.log('Segundo Timeout');
}
```

0s

```javascript
() => {
  console.log('Primer Timeout');
}
```

3s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→ ~
```

Inicio de programa

Consola

```
setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );
```

Global( ) // main( )

Call Stack

```
() => {
    console.log('Segundo Timeout');
}
```

0s

```
() => {
    console.log('Primer Timeout');
}
```

3s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );
```

Global( ) // main( )

Call Stack

```
() => {
    console.log('Tercer Timeout');
}
```

0s

```
() => {
    console.log('Segundo Timeout');
}
```

0s

```
() => {
  console.log('Primer Timeout');
}
```

3s

Libuv

```
→  ~
Inicio de programa
```

Consola

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→ ~
Inicio de programa
```

Consola

Global( ) // main( )

Call Stack

```javascript
() => {
    console.log('Tercer Timeout');
}
```

0s

```javascript
() => {
    console.log('Segundo Timeout');
}
```

0s

```javascript
() => {
  console.log('Primer Timeout');
}
```

3s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
```

Consola

```
console.log('Fin de programa');
```

Global( ) // main( )

Call Stack

```
() => {
    console.log('Tercer Timeout');
}
```

0s

```
() => {
    console.log('Segundo Timeout');
}
```

0s

```
() => {
  console.log('Primer Timeout');
}
```

3s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
console.log('Fin de programa');
```

Global( ) // main( )

Call Stack

```
() => {
    console.log('Tercer Timeout');
}
```

0s

```
() => {
    console.log('Segundo Timeout');
}
```

0s

```
() => {
    console.log('Primer Timeout');
}
```

3s

Libuv

```
→ ~
Inicio de programa
Fin de programa
```

Consola

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
```

Consola

Global( ) // main( )

Call Stack

```javascript
() => {
    console.log('Tercer Timeout');
}
```

0s

```javascript
() => {
    console.log('Segundo Timeout');
}
```

0s

```javascript
() => {
  console.log('Primer Timeout');
}
```

3s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
```

Consola

Global( ) // main( )

Call Stack

```javascript
() => {
    console.log('Tercer Timeout');
}
```

0s

```javascript
() => {
    console.log('Segundo Timeout');
}
```

0s

```javascript
() => {
    console.log('Primer Timeout');
}
```

2s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→ ~
Inicio de programa
Fin de programa
```

Consola

Global( ) // main( )

Call Stack

First in / First Out

(completadas)

```javascript
() => {
    console.log('Tercer Timeout');
}
```

0s

```javascript
() => {
    console.log('Segundo Timeout');
}
```

0s

```javascript
() => {
  console.log('Primer Timeout');
}
```

2s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

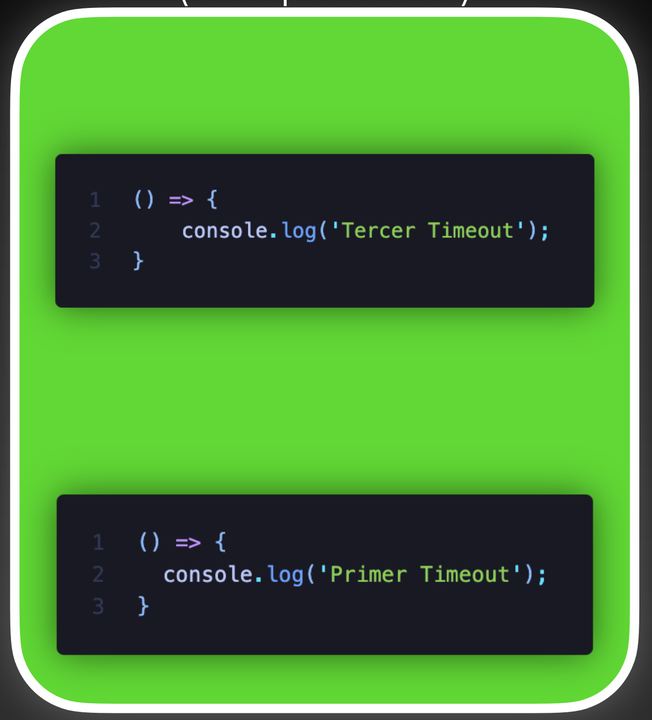
```
→ ~
Inicio de programa
Fin de programa
```

Consola

Call Stack

First in / First Out

(completadas)



Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

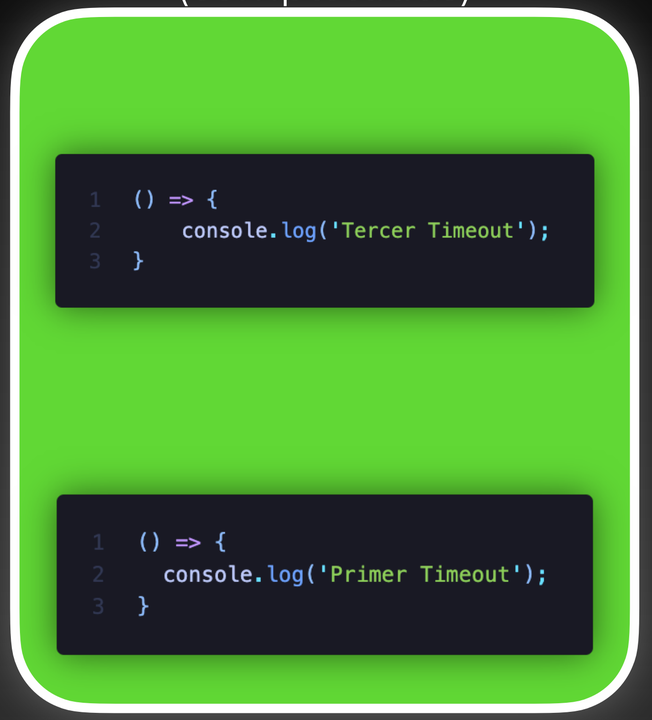
```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
```

Consola

Call Stack

First in / First Out

(completadas)



Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
```

Consola

Global( ) // main( )

Call Stack

First in / First Out

(completadas)

```
() => {
    console.log('Tercer Timeout');
}
```

0s

```
() => {
    console.log('Primer Timeout');
}
```

2s

Libuv

Ejecución de código

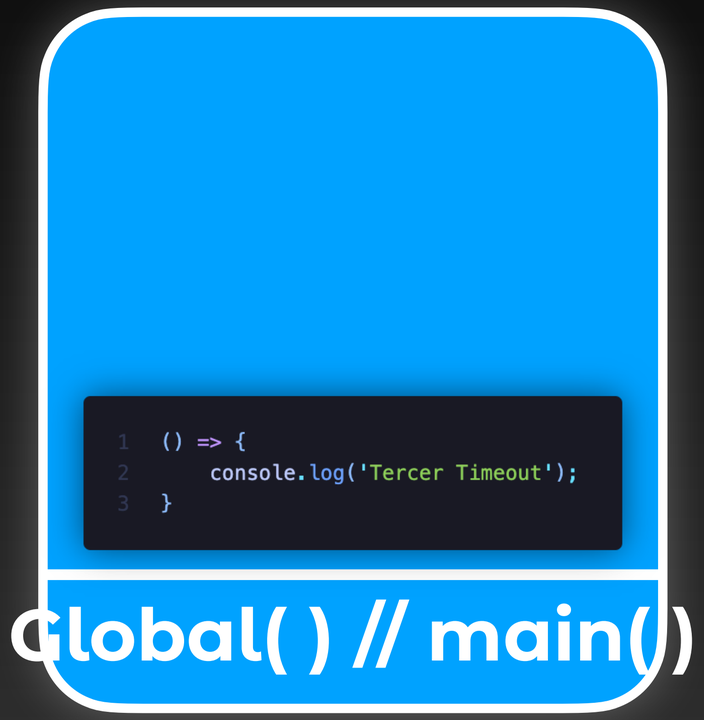
Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
```

Consola



Call Stack

First in / First Out

(completadas)

Libuv

Ejecución de código

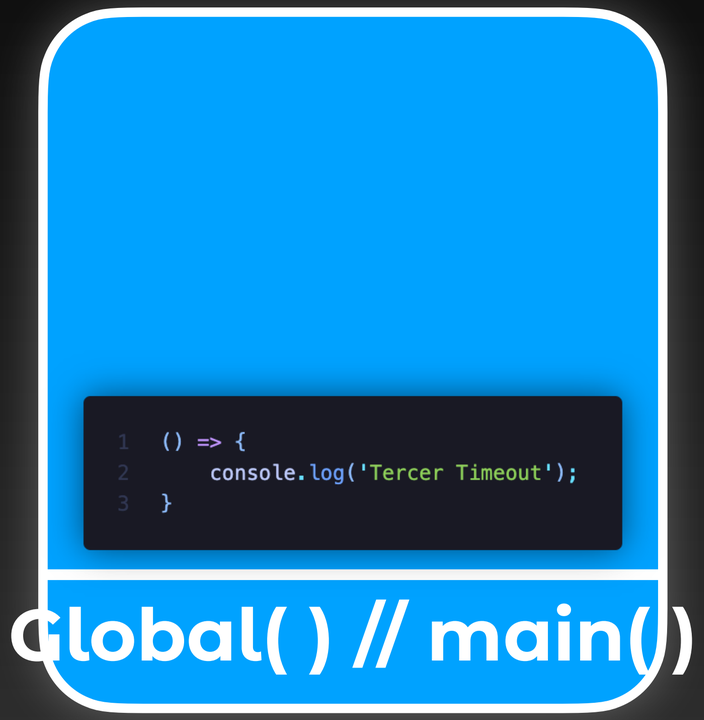
Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
Tercer Timeout
```

Consola



Call Stack

First in / First Out

(completadas)

1s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
Tercer Timeout
```

Consola

Global( ) // main( )

Call Stack

First in / First Out

(completadas)

```javascript
() => {
  console.log('Primer Timeout');
}
```

1s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
Tercer Timeout
```

Consola

Call Stack

First in / First Out

(completadas)

0s

Libuv

Ejecución de código

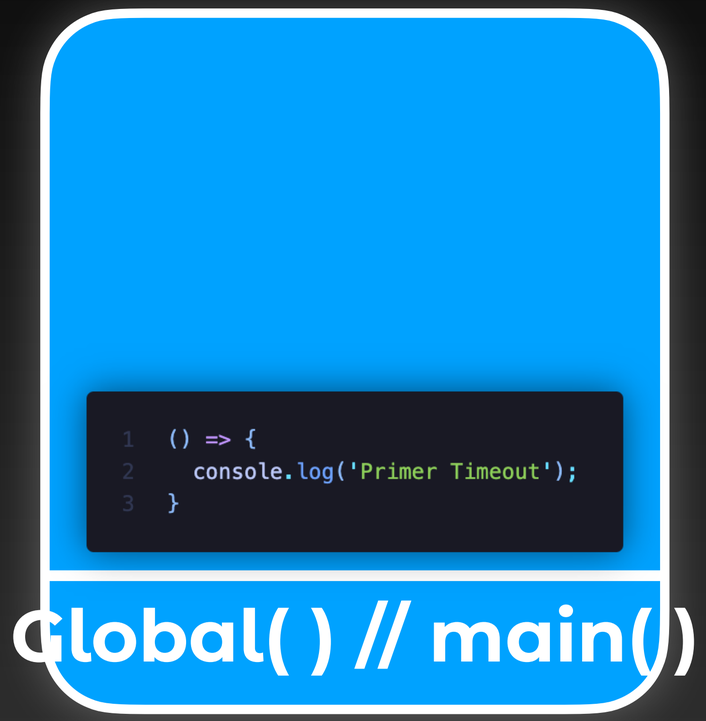
Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→ ~
Inicio de programa
Fin de programa
Segundo Timeout
Tercer Timeout
```

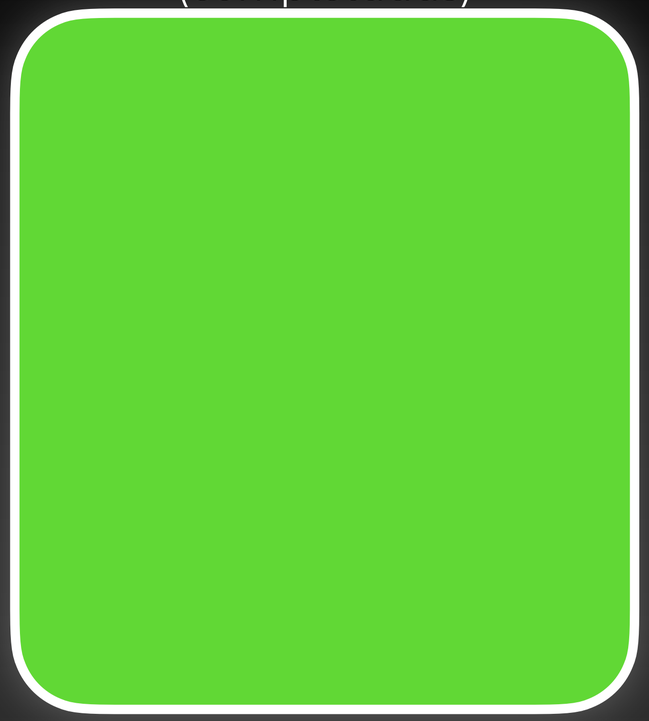
Consola



Call Stack

First in / First Out

(completadas)



Libuv

Ejecución de código

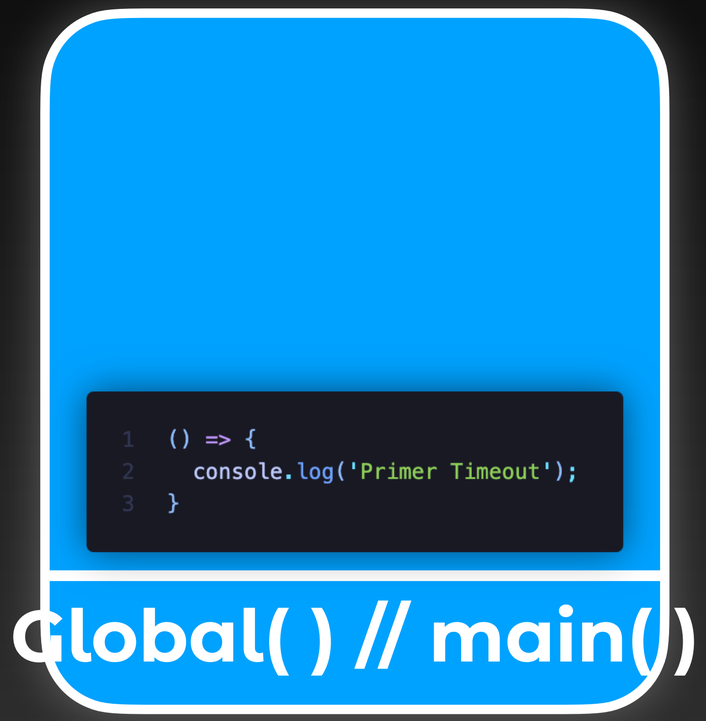
Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
Tercer Timeout
Primer Timeout
```

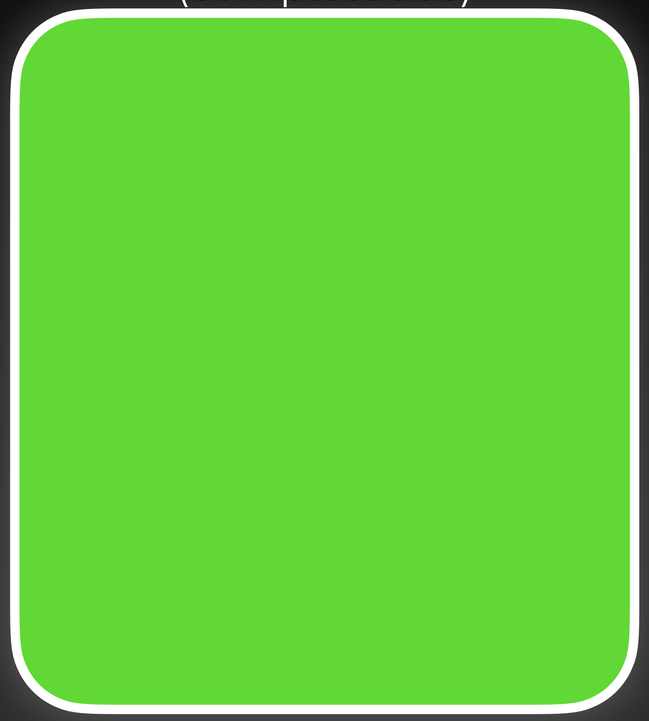
Consola



Call Stack

First in / First Out

(completadas)



Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

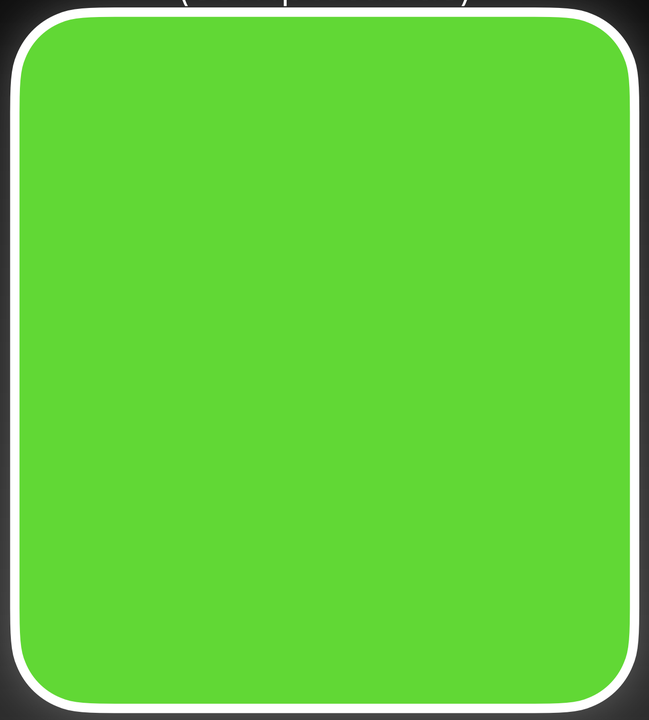
```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
Tercer Timeout
Primer Timeout
```

Consola

Call Stack

First in / First Out

(completadas)



Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
Tercer Timeout
Primer Timeout
```

Consola

Call Stack

First in / First Out

(completadas)

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```javascript
console.log('Inicio de programa');

setTimeout( () => {
    console.log('Primer Timeout');
}, 3000 );

setTimeout( () => {
    console.log('Segundo Timeout');
}, 0 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
Tercer Timeout
Primer Timeout
```

Consola

First in / First Out

(completadas)

Call Stack

Exit code 0

Libuv

```javascript
setTimeout( () => {
    console.log('Segundo Timeout');
}, 1 );

setTimeout( () => {
    console.log('Tercer Timeout');
}, 0 );
```

Ejecución síncrona primero

Ejecución de código

Ejemplo #3

```
console.log('Inicio de programa');

setTimeout(cb, 3000);

setTimeout(cb, 1);

setTimeout(cb, 0);

moreWork();

console.log('Fin de programa');
```

First in / First Out

(completadas)

```
() => {
    console.log('Tercer Timeout');
}
```

0ms

```
() => {
    console.log('Segundo Timeout');
}
```

1ms

```
() => {
  console.log('Primer Timeout');
}
```

3s

Global( ) // main( )

Call Stack

Libuv

```
→ ~
Inicio de programa
Fin de programa
```

Consola

Ejecución síncrona primero

Ejecución de código

Ejemplo #3

```
console.log('Inicio de programa');

setTimeout(cb, 3000);

setTimeout(cb, 1);

setTimeout(cb, 0);

moreWork();

console.log('Fin de programa');
```

First in / First Out

(completadas)

```
moreWork();
```

Global( ) // main( )

Call Stack

```
() => {
    console.log('Tercer Timeout');
}
```

0ms

```
() => {
    console.log('Segundo Timeout');
}
```

1ms

```
() => {
  console.log('Primer Timeout');
}
```

3s

Libuv

```
→ ~
Inicio de programa
Fin de programa
```

Consola

Ejecución síncrona primero

Ejecución de código

Ejemplo #3

```
console.log('Inicio de programa');

setTimeout(cb, 3000);

setTimeout(cb, 1);

setTimeout(cb, 0);

moreWork();

console.log('Fin de programa');
```

```
moreWork();
```

Global( ) // main( )

Call Stack

First in / First Out

(completadas)

```
() => {
    console.log('Tercer Timeout');
}
```

0s

```
() => {
    console.log('Segundo Timeout');
}
```

0s

```
() => {
  console.log('Primer Timeout');
}
```

2s

Libuv

```
→ ~
Inicio de programa
Fin de programa
```

Consola

Ejecución síncrona primero

Ejecución de código

Ejemplo #3

```
console.log('Inicio de programa');

setTimeout(cb, 3000);

setTimeout(cb, 1);

setTimeout(cb, 0);

moreWork();

console.log('Fin de programa');
```

First in / First Out

(completadas)

```
() => {
    console.log('Tercer Timeout');
}
```

0s

```
() => {
    console.log('Segundo Timeout');
}
```

0s

```
() => {
  console.log('Primer Timeout');
}
```

2s

Global( ) // main( )

Call Stack

Libuv

```
→  ~
Inicio de programa
Fin de programa
```

Consola

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout(cb, 3000);

setTimeout(cb, 1);

setTimeout(cb, 0);

moreWork();

console.log('Fin de programa');
```

```
→ ~
Inicio de programa
Fin de programa
Segundo Timeout
```

Consola

Global( ) // main( )

```
() => {
    console.log('Segundo Timeout');
}
```

Call Stack

First in / First Out

(completadas)

```
() => {
    console.log('Tercer Timeout');
}
```

0s

```
() => {
  console.log('Primer Timeout');
}
```

2s

Libuv

Ejecución de código

Ejemplo #3

Ejecución síncrona primero

```
console.log('Inicio de programa');

setTimeout(cb, 3000);

setTimeout(cb, 1);

setTimeout(cb, 0);

moreWork();

console.log('Fin de programa');
```

```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
```

Consola

Global( ) // main( )

Call Stack

First in / First Out

(completadas)

```
() => {
    console.log('Tercer Timeout');
}
```

0s

```
() => {
  console.log('Primer Timeout');
}
```

2s

Libuv

Ejecución síncrona primero

Ejecución de código

Ejemplo #3

```
console.log('Inicio de programa');

setTimeout(cb, 3000);

setTimeout(cb, 1);

setTimeout(cb, 0);

moreWork();

console.log('Fin de programa');
```

First in / First Out

(completadas)

```
() => {
    console.log('Tercer Timeout');
}
```

Global( ) // main( )

Call Stack

```
() => {
  console.log('Primer Timeout');
}
```

2s

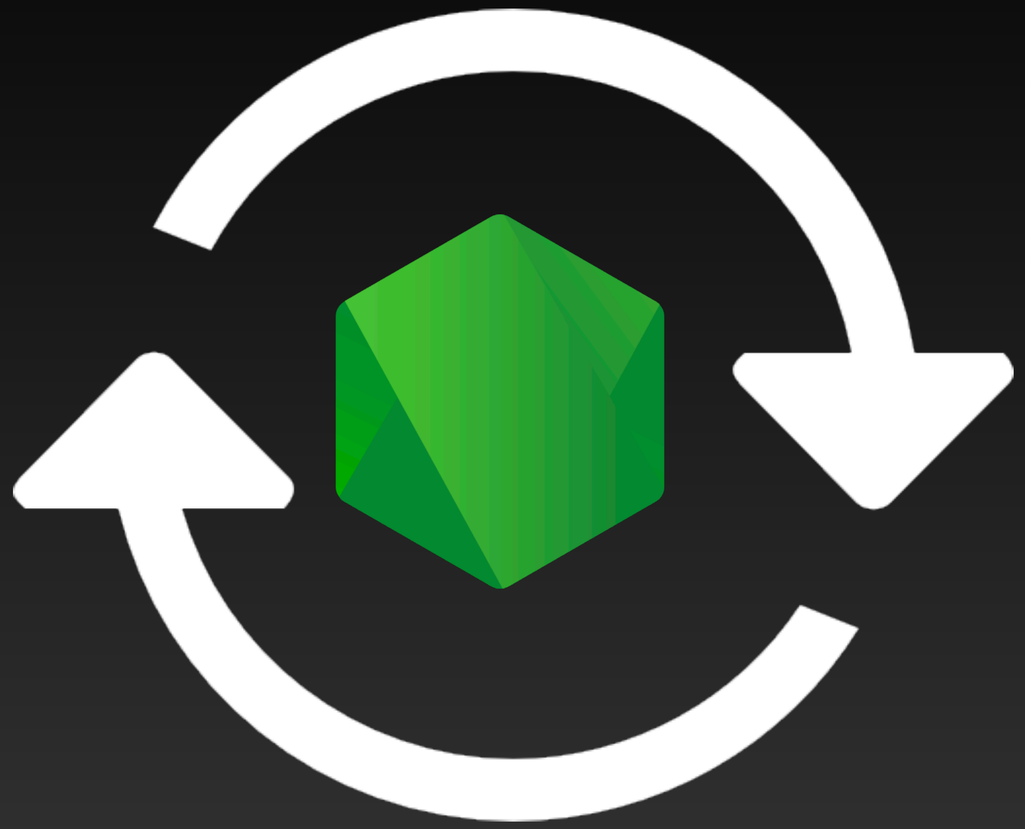
Libuv

```
→  ~
Inicio de programa
Fin de programa
Segundo Timeout
```

Consola

- ¿Qué pasa si dos Timeouts terminan al mismo tiempo?
- ¿Qué pasa si una promesa termina al mismo tiempo que otro callback?

¿En qué orden se ejecutarán?



El Event Loop quien decide

Event Loop

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...

Event Loop

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...

Event Loop

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...

Cola de tareas

Event Loop

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...

Event Loop

Cola de tareas

Libuv

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...

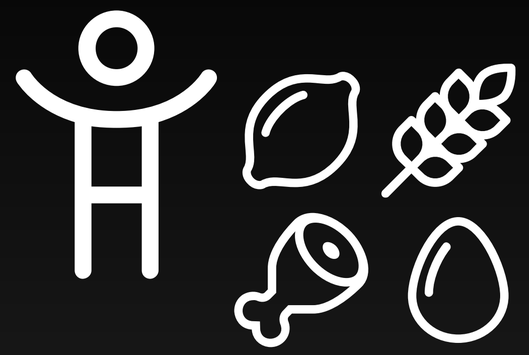
Event Loop

Cola de tareas

Libuv

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...



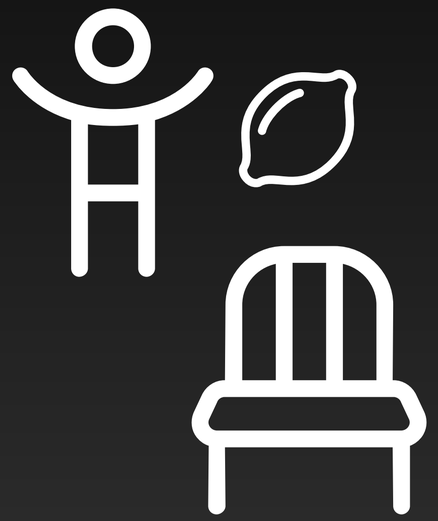
Cola de tareas

Libuv

Event Loop

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...



Cola de tareas

Libuv

Event Loop

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...

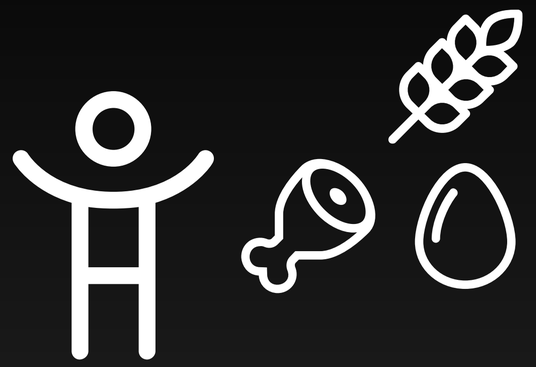
Event Loop

Cola de tareas

Libuv

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...



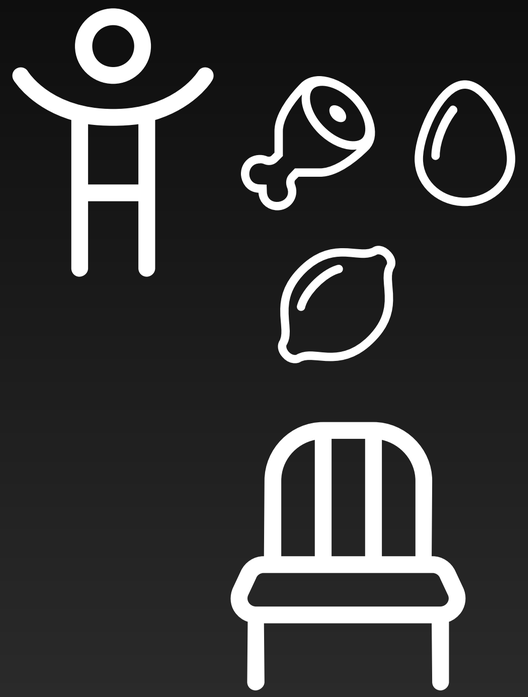
Event Loop

Cola de tareas

Libuv

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...



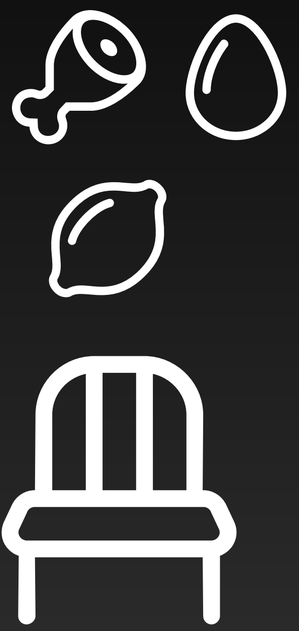
Cola de tareas

Libuv

Event Loop

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...



Cola de tareas

Libuv

Event Loop

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...

Cola de tareas

Libuv

Event Loop

Pila de ejecución (CallStack)

Arroz
Papa
Carne
Zanahoria
Etc...

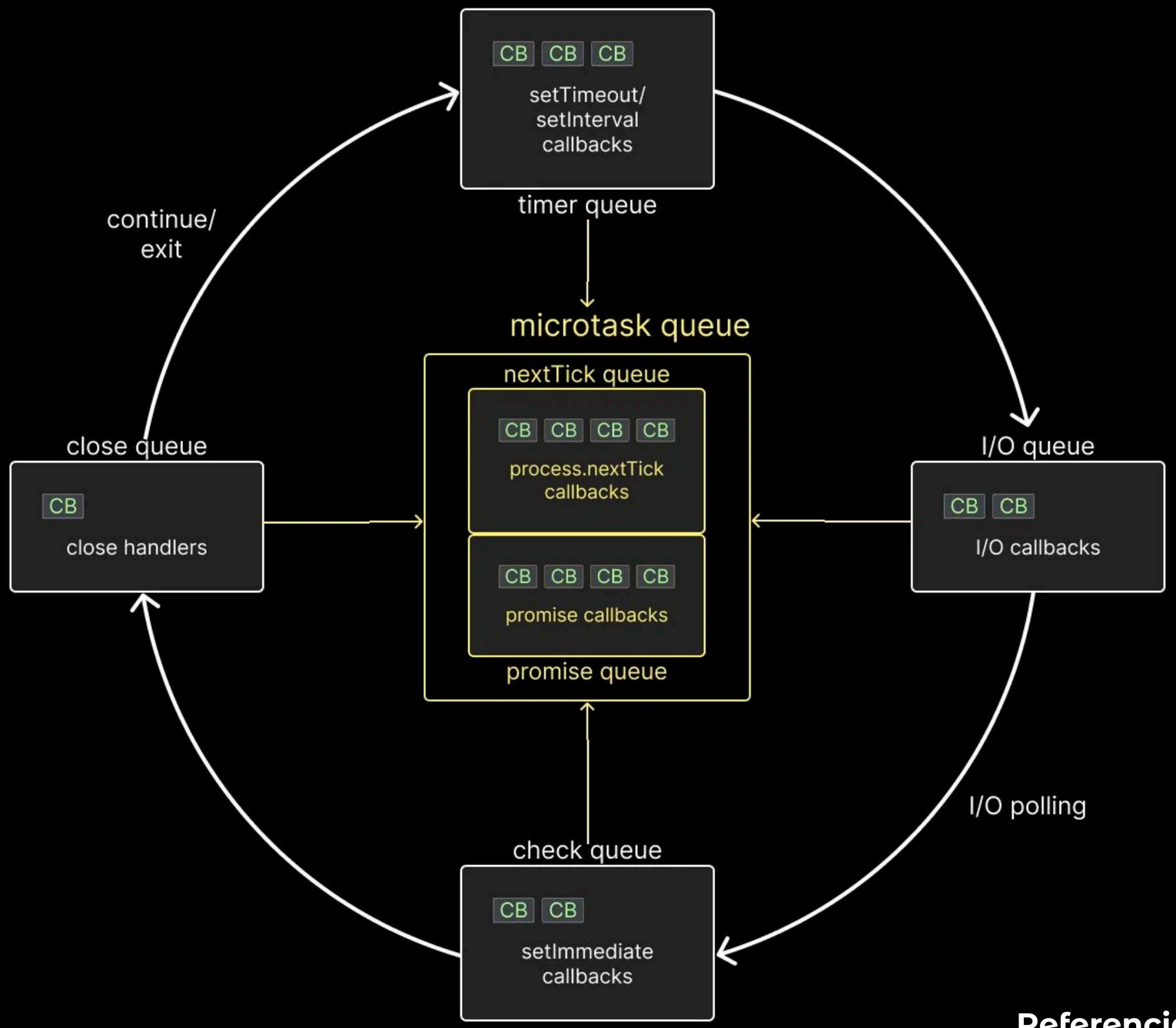
Event Loop

Cola de tareas

Libuv

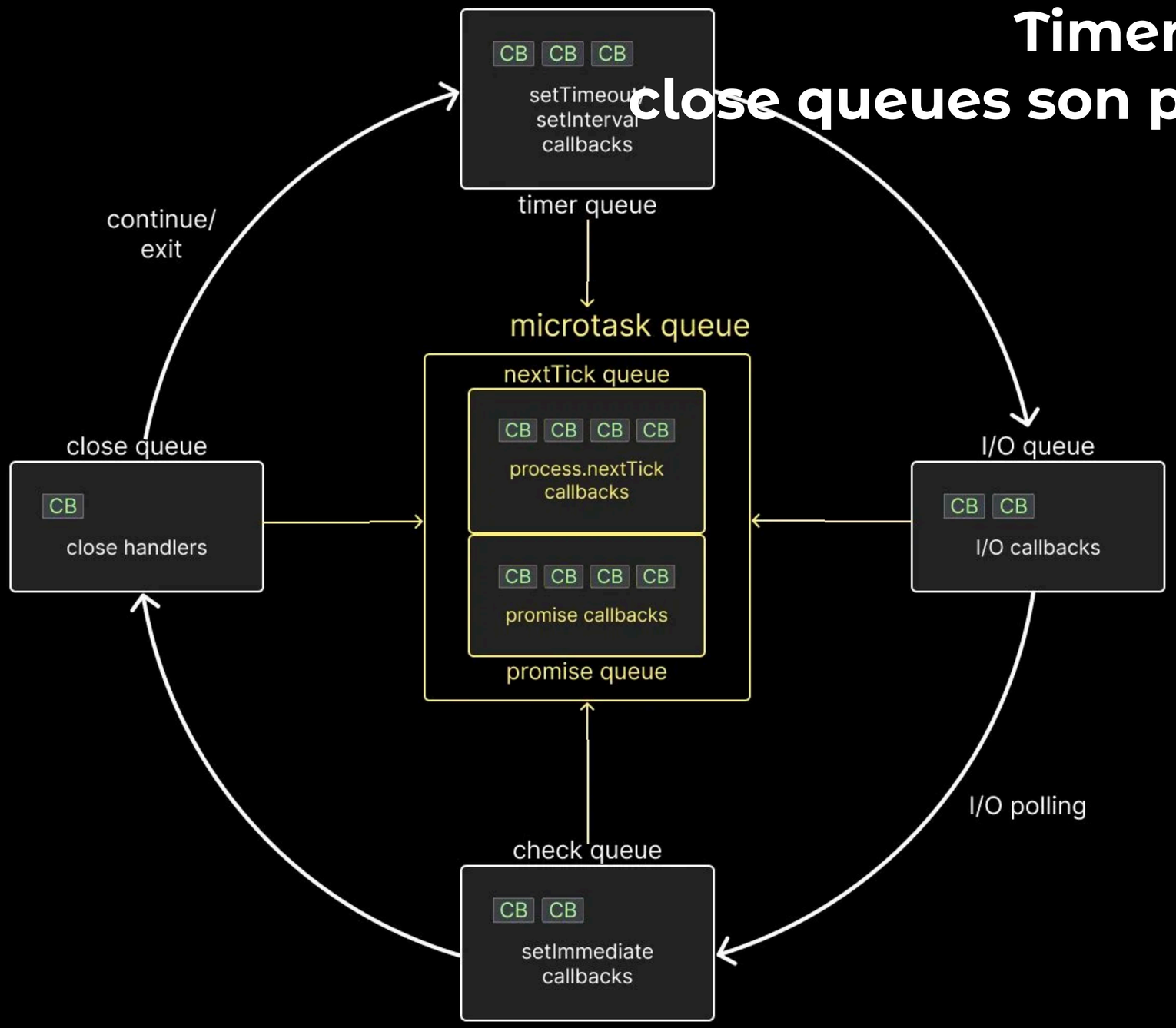
- ¿Qué pasa si dos Timeouts terminan al mismo tiempo?
- ¿Qué pasa si una promesa termina al mismo tiempo que otro callback?

¿En qué orden se ejecutarán?

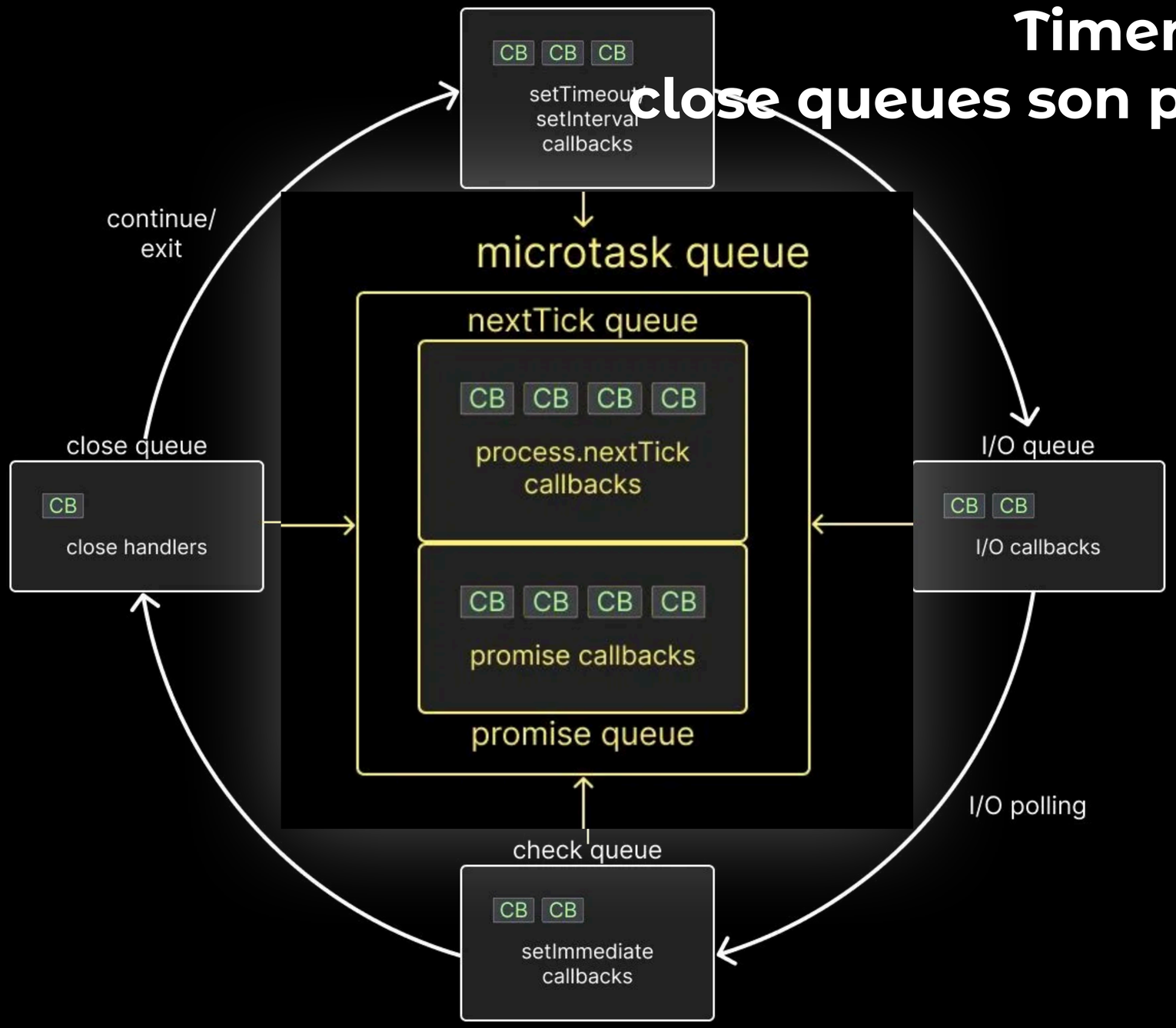


Referencias en el material adjunto

Timer, I/O, checks, close queues son parte de libuv

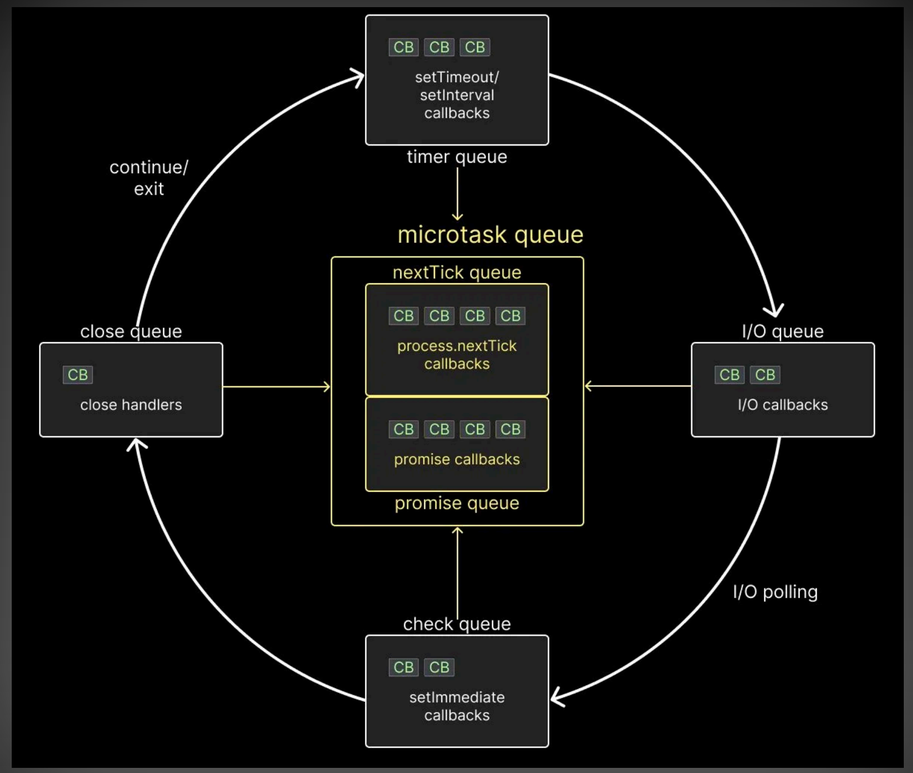


Timer, I/O, checks, close queues son parte de libuv



El Event Loop sigue contando

3. Callbacks en el microtask queue (si hay) se ejecutan después de los callback timers, primero tareas en el nextTick queue y luego tareas en el promise queue.
4. Callbacks de I/O se ejecutan.
5. Callbacks en el microtask queue se ejecutan (si hay), y luego promise queue (si hay)
6. Todos los callbacks en el check queue se ejecutan.
7. Callbacks en el microtask se ejecutan después de cada callback en el check queue. (Siguiendo el mismo orden anterior, nextTick y luego promise)
8. Todos los callbacks en el close



node

Pregunta:

Cuando se completa una tarea asíncrona en libuv, ¿En qué momento decide Node ejecutar la función callback asociada en la pila de llamadas (callstack)?

Respuesta:

Callbacks son ejecutados sólo cuando el callstack está vació

node

Pregunta:

¿Node espera a que la pila de llamadas esté vacía antes de ejecutar un callback o interrumpe el flujo normal de ejecución para ejecutar el callback?

Respuesta:

El flujo normal de ejecución no se interrumpirá para ejecutar una función de devolución de llamada.

Pregunta:

¿Qué pasa con otros métodos asíncronos como setTimeout y setInterval, que también retrasan la ejecución de un callback?

Respuesta:

Los callbacks de setTimeouts y setIntervals tienen prioridad, pero no interrumpen el flujo normal.

Pregunta:

Si dos tareas asíncronas como setTimeout y readFile se completan al mismo tiempo,

¿cómo decide Node qué callback ejecutar primero en la pila de llamadas? ¿Uno tiene prioridad sobre el otro?

Respuesta:

Los callbacks de temporizador se ejecutan antes que las devoluciones de llamada de I/O, incluso si ambas están listas exactamente al mismo tiempo.

node

node

Introducción
Pruebas unitarias
y de integración

¿Qué son?

Lineamientos

¿Para qué sirven?

Características

Mitos

{dev/talles}

No son una pérdida de tiempo

{dev/talles}

Unitarias y de Integración

Unitarias: Enfocadas en pequeñas funcionalidades.

Integración: Enfocadas en cómo reaccionan varias piezas en conjunto

{dev/talles}

Unitarias

Integración

{dev/talles}

Características de las pruebas

1. Fáciles de escribir
2. Fáciles de leer
3. Confiables
4. Rápidas
5. Principalmente unitarias

AAA

- Arrange (Arreglar)
- Act (Actuar)
- Assert (Afirmar)

AAA

Preparamos el estado inicial

- Inicializamos variables
- Importaciones necesarias

Arrange - Arreglar

Aplicamos acciones o estímulos

- Llamar métodos
- Simular clicks
- Realizar acciones sobre el paso anterior

Act - Actuar

Observar el comportamiento resultante

- Son los resultados esperados
- Ej: Que algo cambie, algo incremente o bien que nada suceda

Assert - Afirmar

mitos

- Hacen que mi aplicación no tenga errores
- Las pruebas no pueden fallar
- Hacen más lenta mi aplicación
- Es una pérdida de tiempo
- Hay que probar todo

node

noc

(Network Operation Center)

NOC-App

{dev/talles}

noc

(Network Operation Center)

NOC-App

- **Ejecutar código en momentos específicos**
  - Por hora
  - Al final del día
  - Todos los lunes
  - Etc

Este “código” será un caso de uso

node

noc

(Network Operation Center)

NOC-App

- Monitorear un API
- Crear tus propios procesos de monitoreo
- Enviar correos

{dev/talles}

noc

(Network Operation Center)

NOC-App

- **Grabar los LOGs en:**
  - FileSystem
  - MongoDB
  - PostgreSQL

Usando arquitectura limpia

NOC-App

Logs

FileSystem

MongoDB

PostgreSQL

{dev/talles}

NOC-App

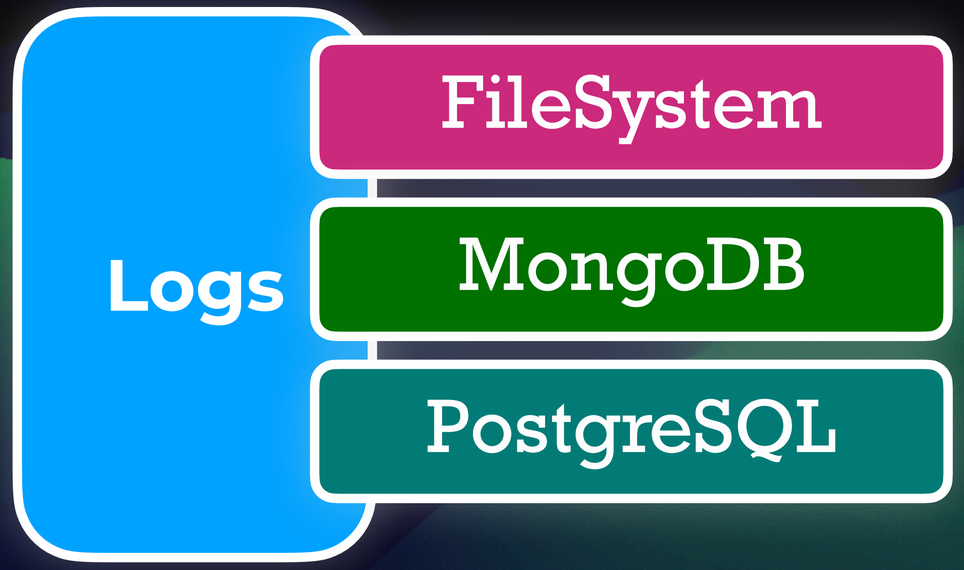
NOC-App

{dev/talles}

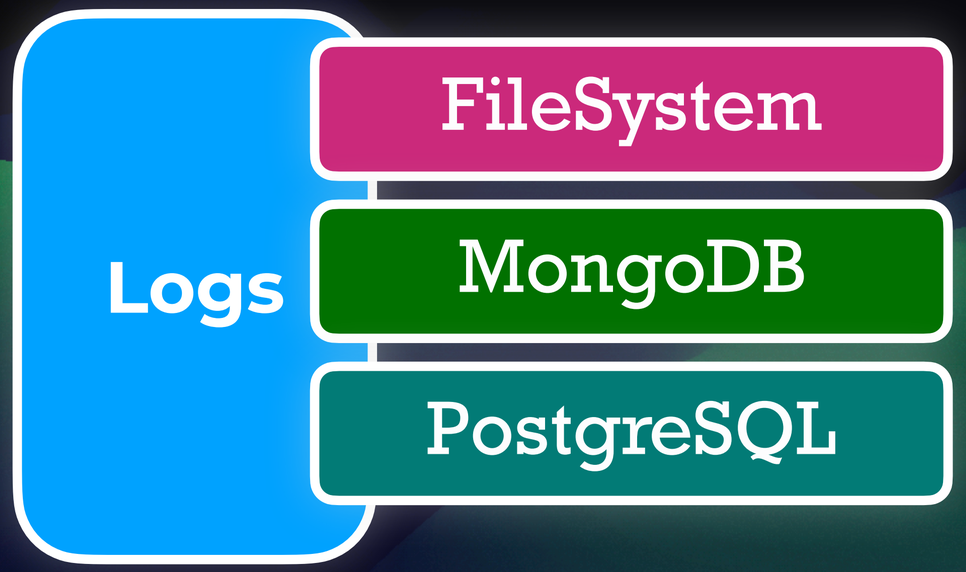
NOC-App

NOC-App

NOC-App



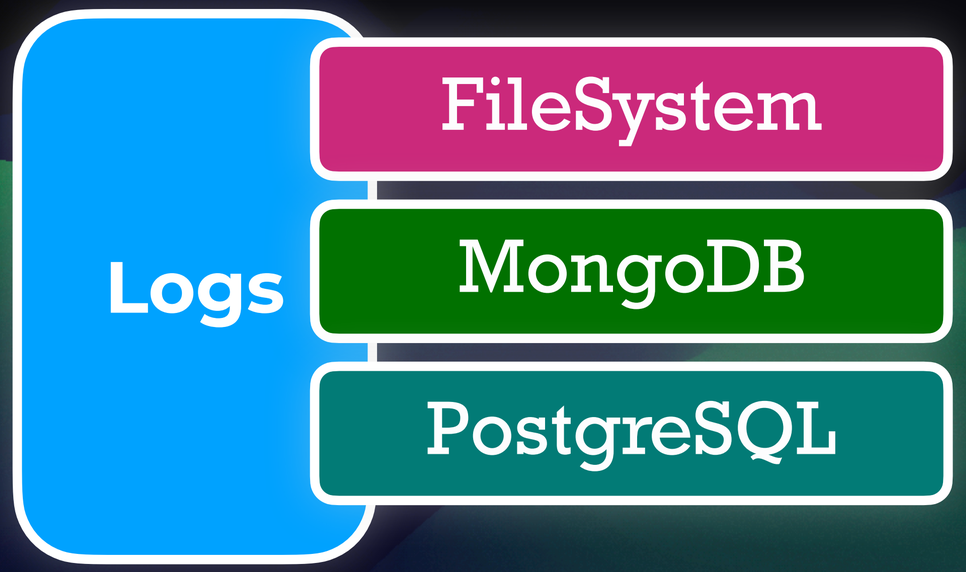
NOC-App



Enviar Email

- Plano
- HTML
- Con attachements

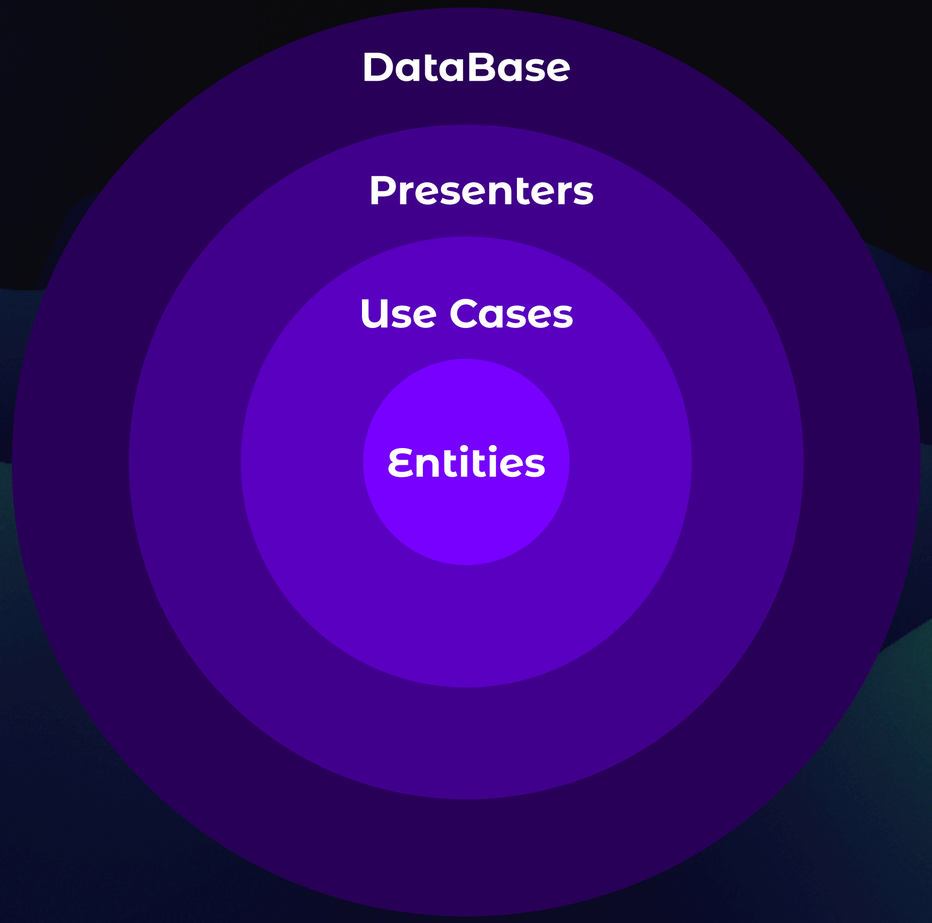
NOC-App



Enviar Email

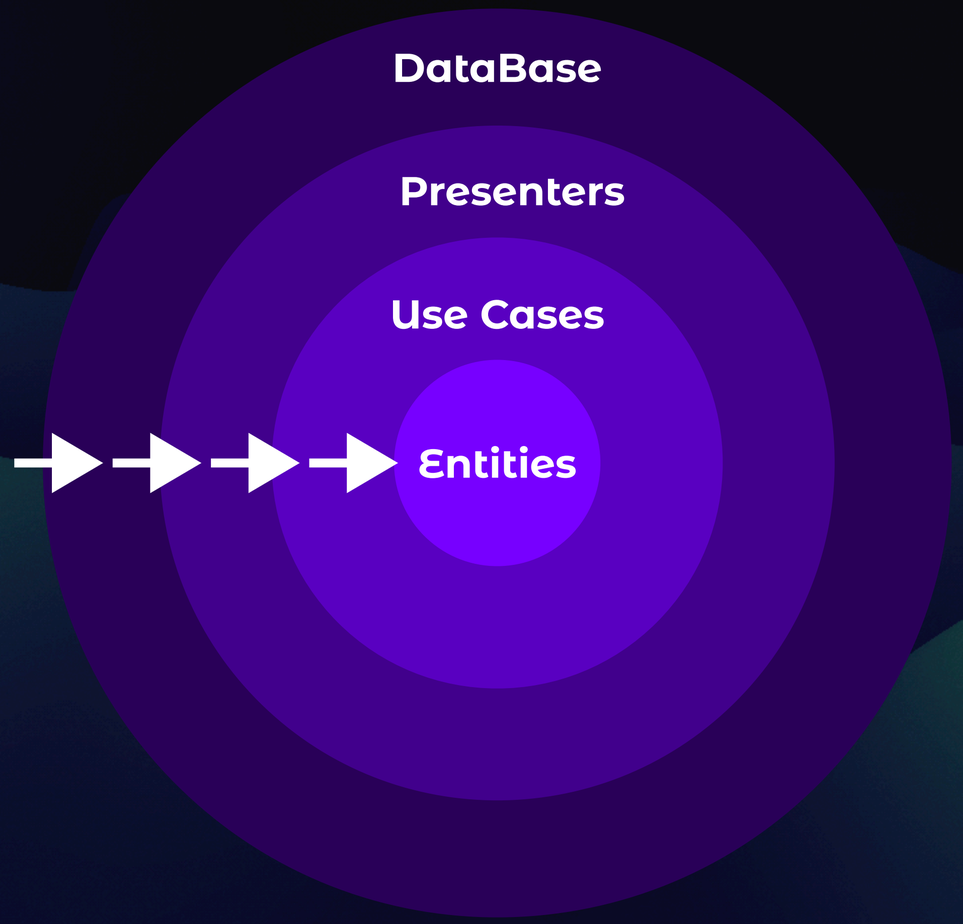
- Plano
- HTML
- Con attachements

NOC-App



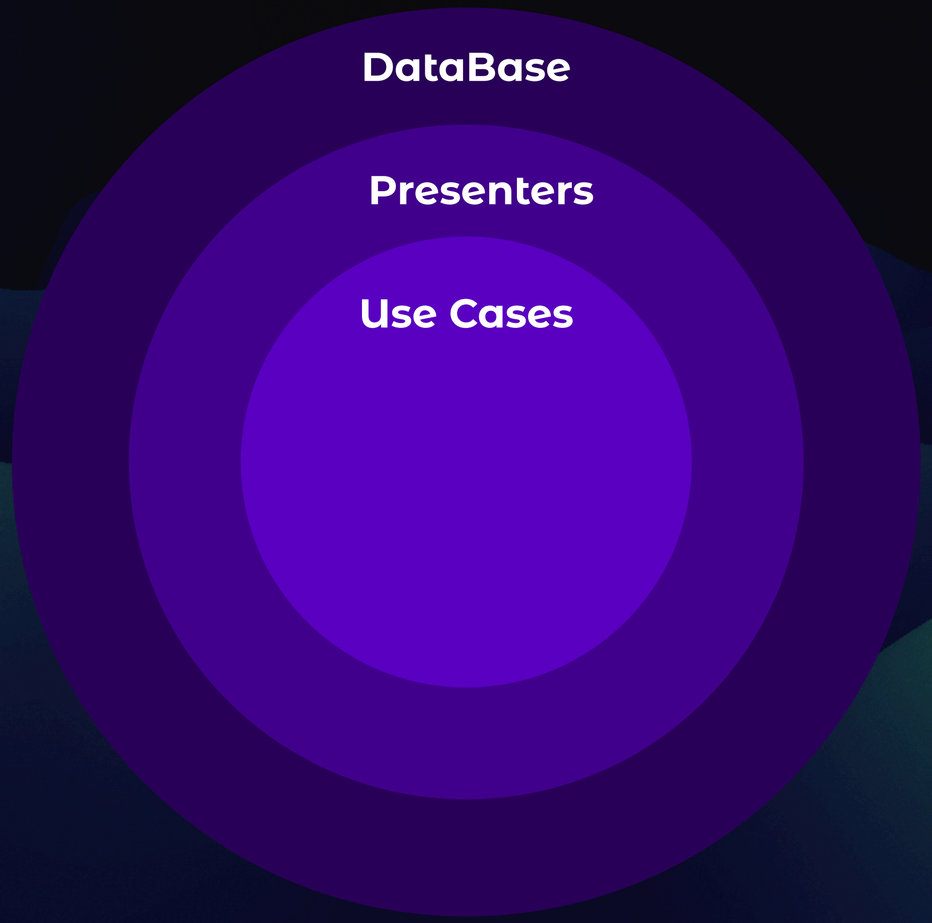
{dev/talles}

NOC-App



{dev/talles}

NOC-App

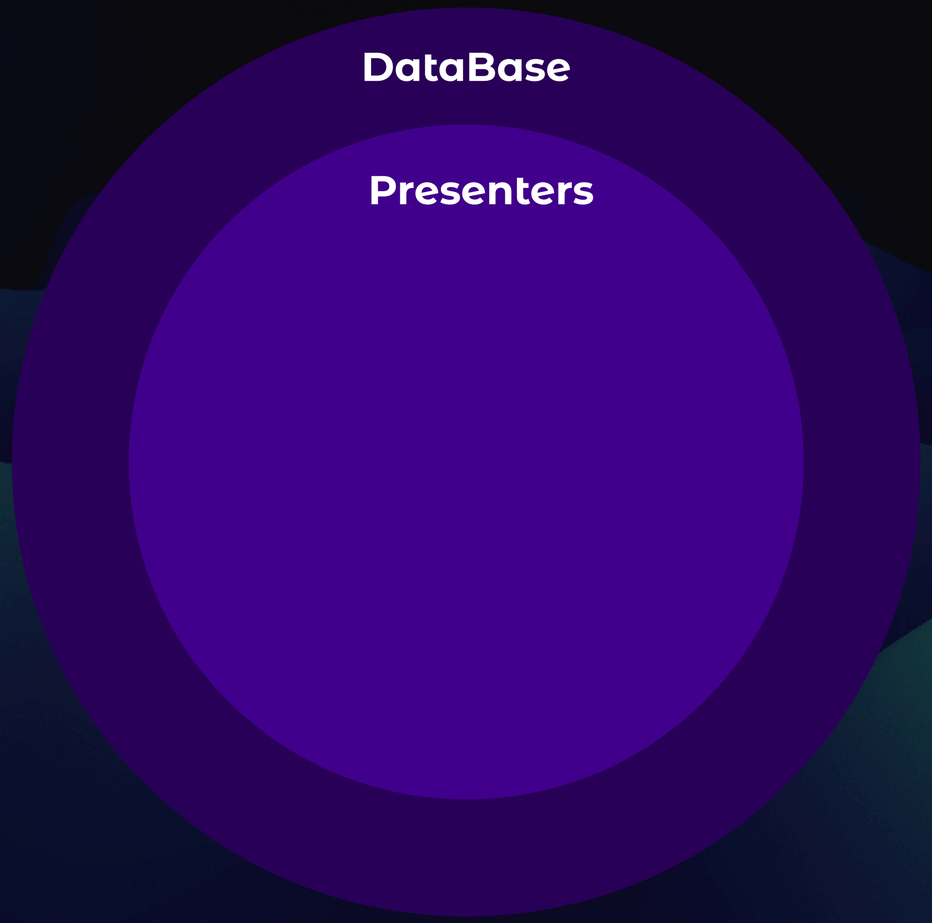


Entities

LogEntity

- Nivel de severidad
- Mensaje del suceso
- ¿Cuándo paso?

NOC-App



- Grabar logs
- Leer logs por severidad
- Enviar email

NOC-App

DataBase

Presenters

- Consola

{dev/talles}

NOC-App

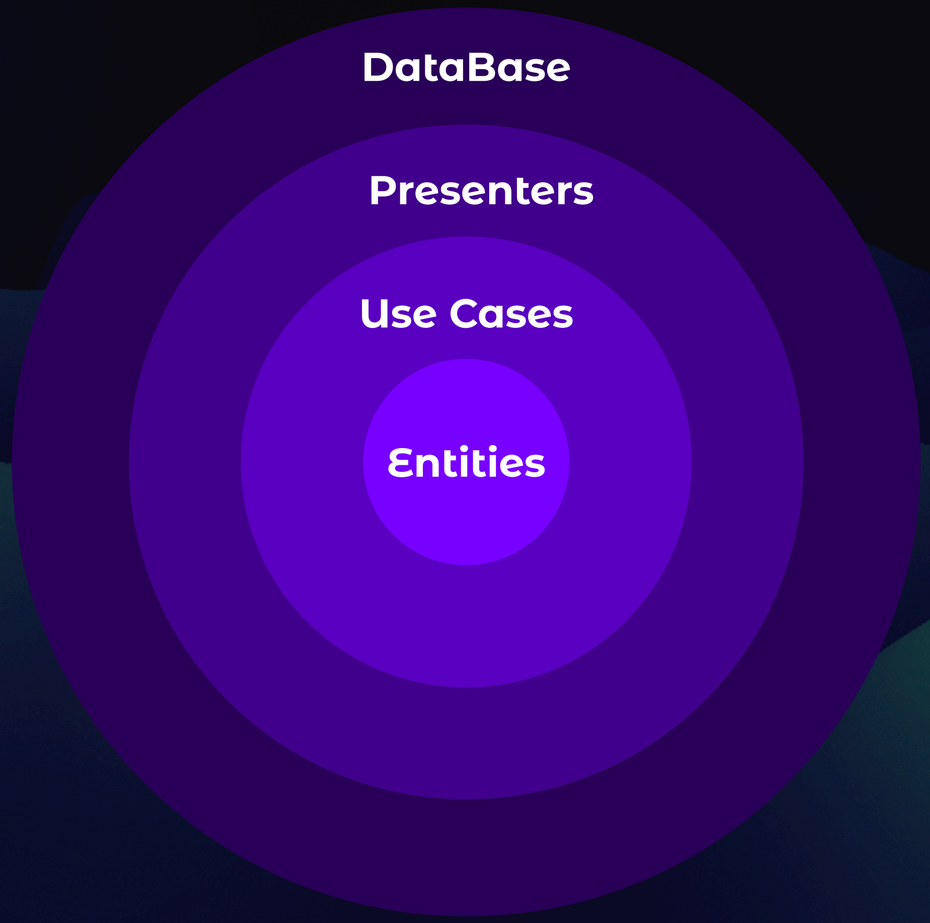
DataBase

- FileSystem
- MongoDB
- PostgreSQL

{dev/talles}

Consideraciones

NOC-App



No debería de afectar si:

- Cambiamos la base de datos
- Cambiamos el motor de correos
- Añadimos o eliminamos tareas
- Queremos trabajar con múltiples origines de datos

¿Cómo lo logramos?

Tenemos que crear más archivos y directorios que en otras circunstancias no serían necesarios.

node

Entity

```
1  export enum LogSeverityLevel {
2    low = 'low',
3    medium = 'medium',
4    high = 'high'
5  }
```

```
1  export class LogEntity {
2    public level: LogSeverityLevel;
3    public message: string;
4    public createdAt: Date;
5
6    constructor(message: string, level: LogSeverityLevel) {
7      this.message = message;
8      this.level = level;
9      this.createdAt = new Date();
10   }
```

{dev/talles}

Datasource

```
1
2  export abstract class LogDatasource {
3    abstract saveLog( log: LogEntity): Promise<void>;
4    abstract getLogs( severityLevel: LogSeverityLevel ): Promise<LogEntity[]>;
5  }
```

Establece cómo los orígenes de datos se comportarán

Repository

```
1
2  export class LogRepositoryImpl implements LogRepository {
3
4    constructor(
5      private readonly logDatasource: LogDatasource
6    ) { }
7
```

Habla con el DataSource, esto permite cambiarlo fácilmente por:

- FileSystemDatasource
- MongoDBDatasource
- PostgreSQLDatasource

Casos de uso

```
1
2  export class CheckService implements CheckServiceUseCase {
3
4    constructor(
5      private logRepository: LogRepository,
6      ) {}
```

Los casos de uso hablan reciben el repositorio(s), el cual ya llega configurado para que llame sus métodos

Al caso de uso, no le interesa si es MongoDB, PostgreSQL o FileSystem.

No se preocupen por comprender todo

No pretendo que lo hagan, pero mi idea es que los conceptos mencionados, se queden en su subconsciente.

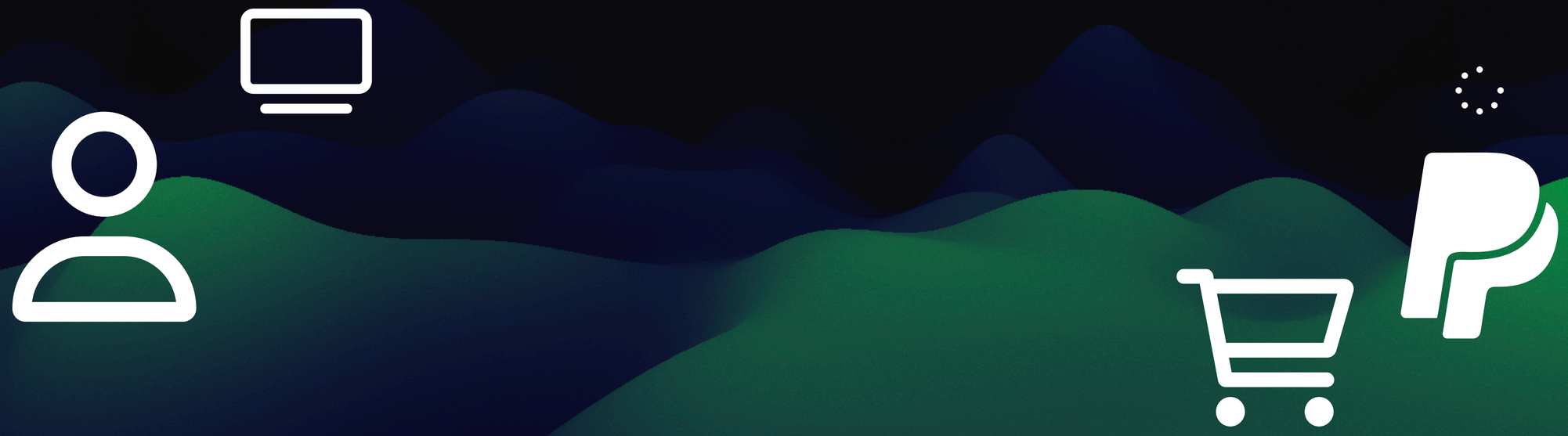
Regresaremos a ellos cuando sea necesario.

{dev/talles}

WebHooks

Sin WebHooks

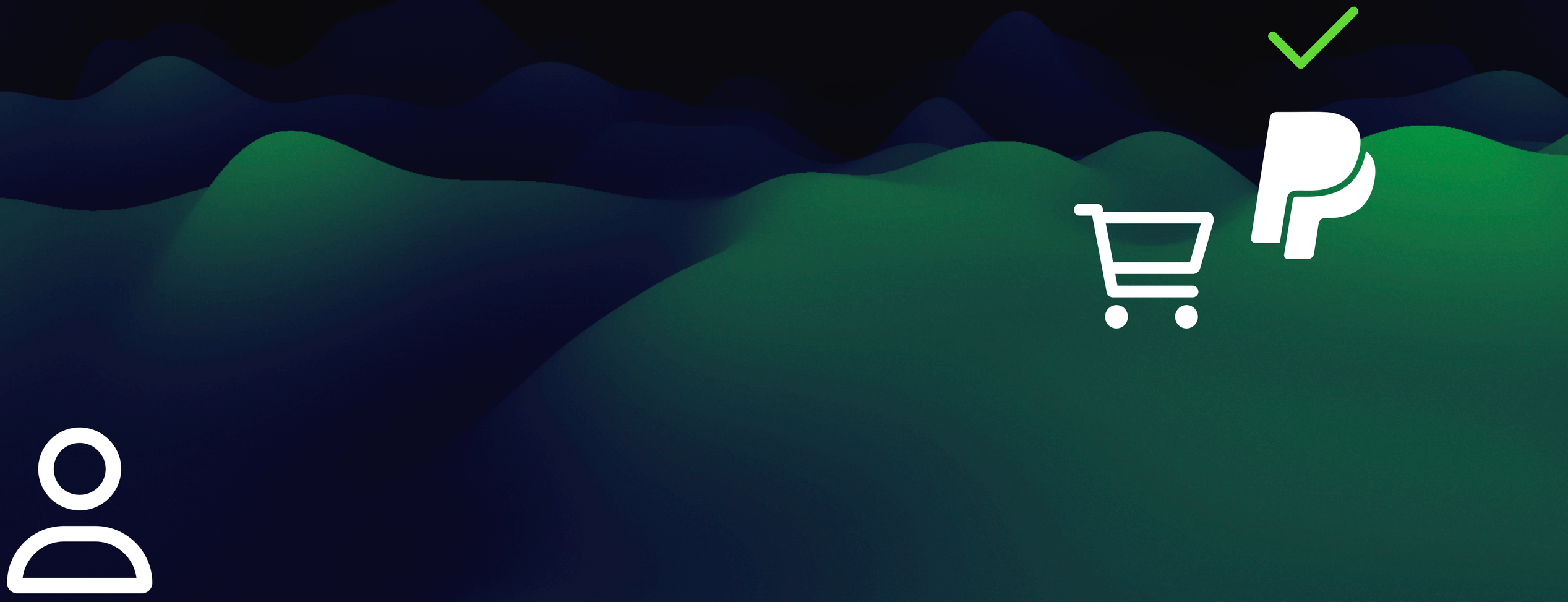
Sin WebHooks



Sin WebHooks

Cierra el navegador

Sin WebHooks



Sin WebHooks

Sin WebHooks

Entonces podríamos hacer esto

Sin WebHooks

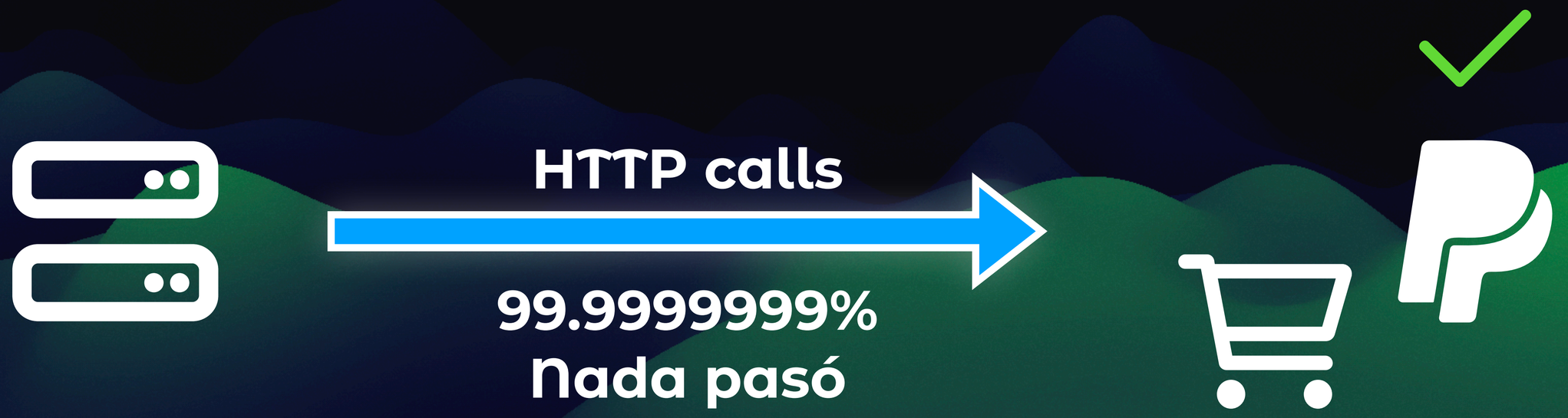
Entonces podríamos hacer esto

Sin WebHooks

Entonces podríamos hacer esto

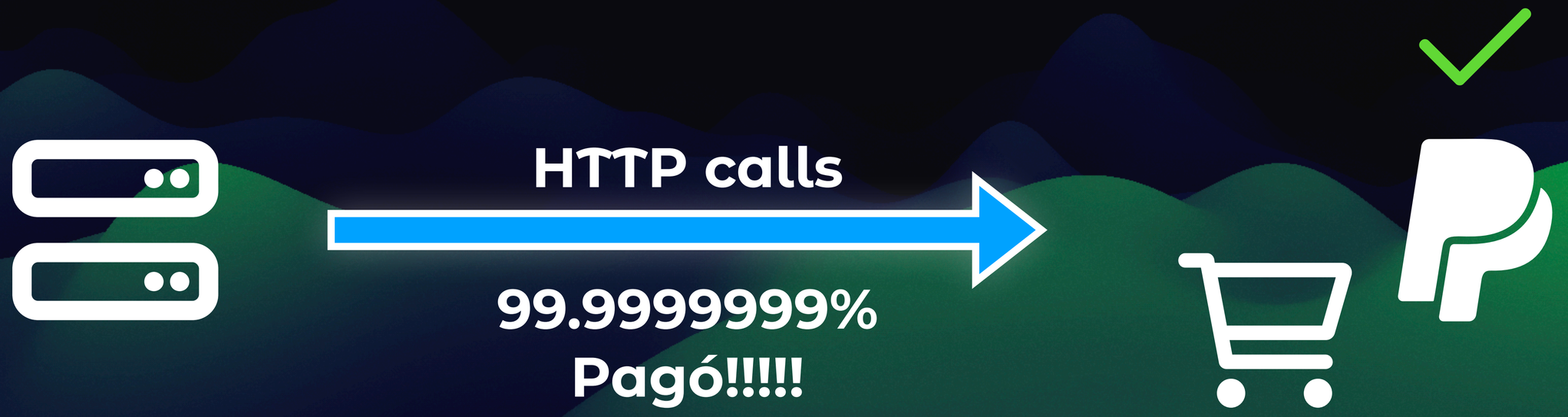
Sin WebHooks

Entonces podríamos hacer esto



Sin WebHooks

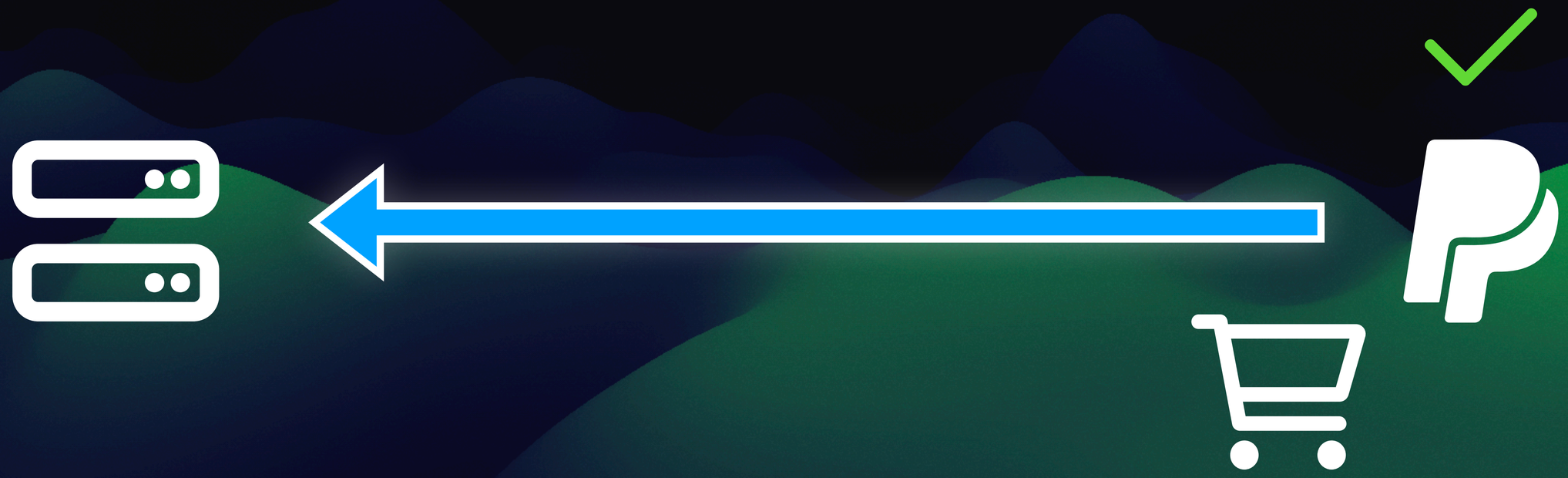
Entonces podríamos hacer esto



WebHooks

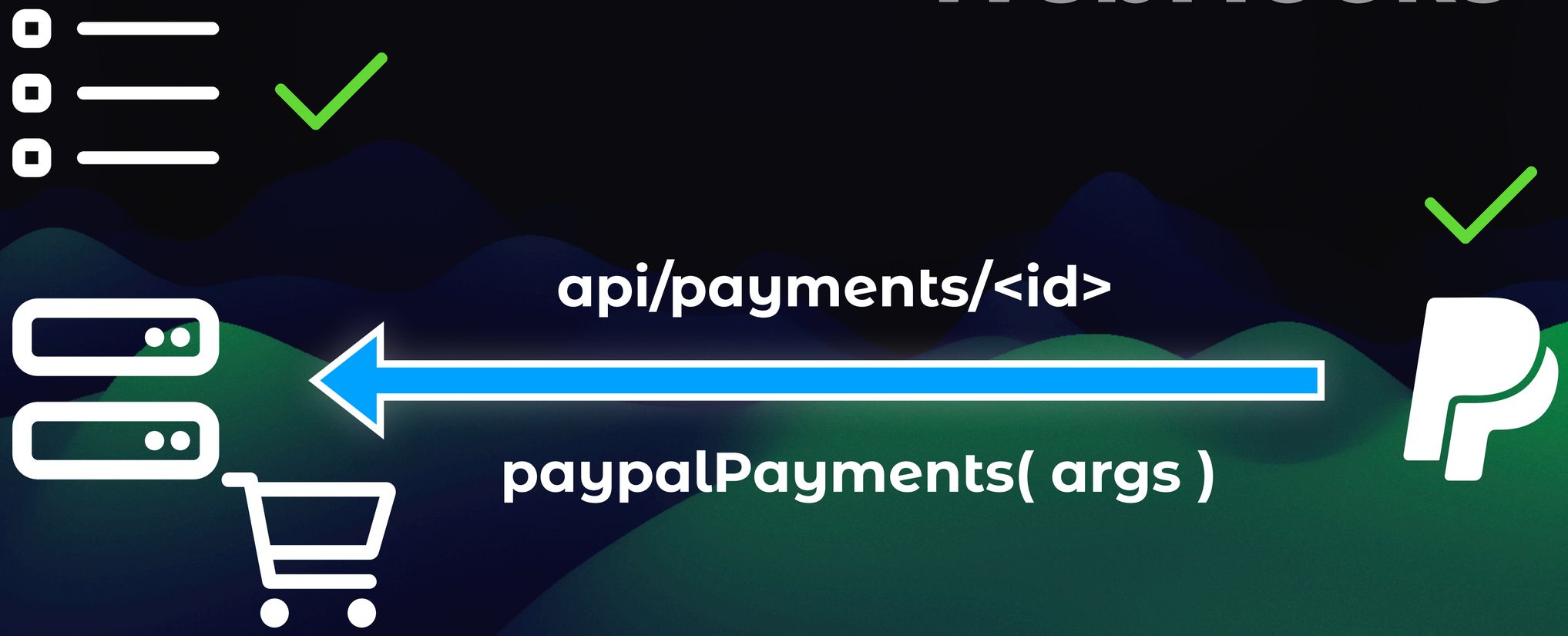
WebHooks

WebHooks



WebHooks

WebHooks



node

¿Simple no?

¿Qué pasa si alguien sabe el endpoint?

api/payments/<id>

paypalPayments(args)

¿Qué pasa si alguien sabe el endpoint?

api/payments/<id>

paypalPayments(args)

node

Secret Token

53babb8e097b4fd948c665a8ce2b9cce3bcc25bf

node

Secret

53babb8e097b4fd948c665a8ce2b9cce3bcc25bf

Token

Secret

53babb8e097b4fd948c665a8ce2b9cce3bcc25bf

Token

Secret

53babb8e097b4fd948c665a8ce2b9cce3bcc25bf

Token

Ok!

Secret

53babb8e097b4fd948c665a8ce2b9cce3bcc25bf

Secret

53babb8e097b4fd948c665a8ce2b9cce3bcc25bf

Token?

Secret

53babb8e097b4fd948c665a8ce2b9cce3bcc25bf

Token?

¿llamadas fallidas?
Ban IP

WebHooks

Puntos a considerar

- Nunca compartas los “secrets”.
- Deben de ser aleatorios seguros.
- Sigue las recomendaciones de la documentación.
- Nunca confíes en que el anonimato.
- La idea es la misma entre servicios*.