
SENPAI
ACADEMY

Node.js

Asynchronous

SENPAI
ACADEMY

Node.js - Asynchronous

Node.js es una librería asíncrona por definición, está basado en eventos de entrada y salida (E/S), por lo que usa constantemente los llamados ***callbacks***

En un programa sincrónico podríamos hacer algo así:

```
1  // Seudocodigo
2
3  function procesarData () {
4    var data = fetchData ();
5    data += 1;
6    return data;
7  }
```

Lo mismo pero en Node.js se vería de esta forma:

```
1  function processData (callback) {
2    fetchData(function (err, data) {
3      if (err) {
4        console.log("Ocurrió un error");
5        return callback(err);
6      }
7      data += 1;
8      callback(data);
9    });
10 }
```

SENPAI ACADEMY

Node.js - Callbacks

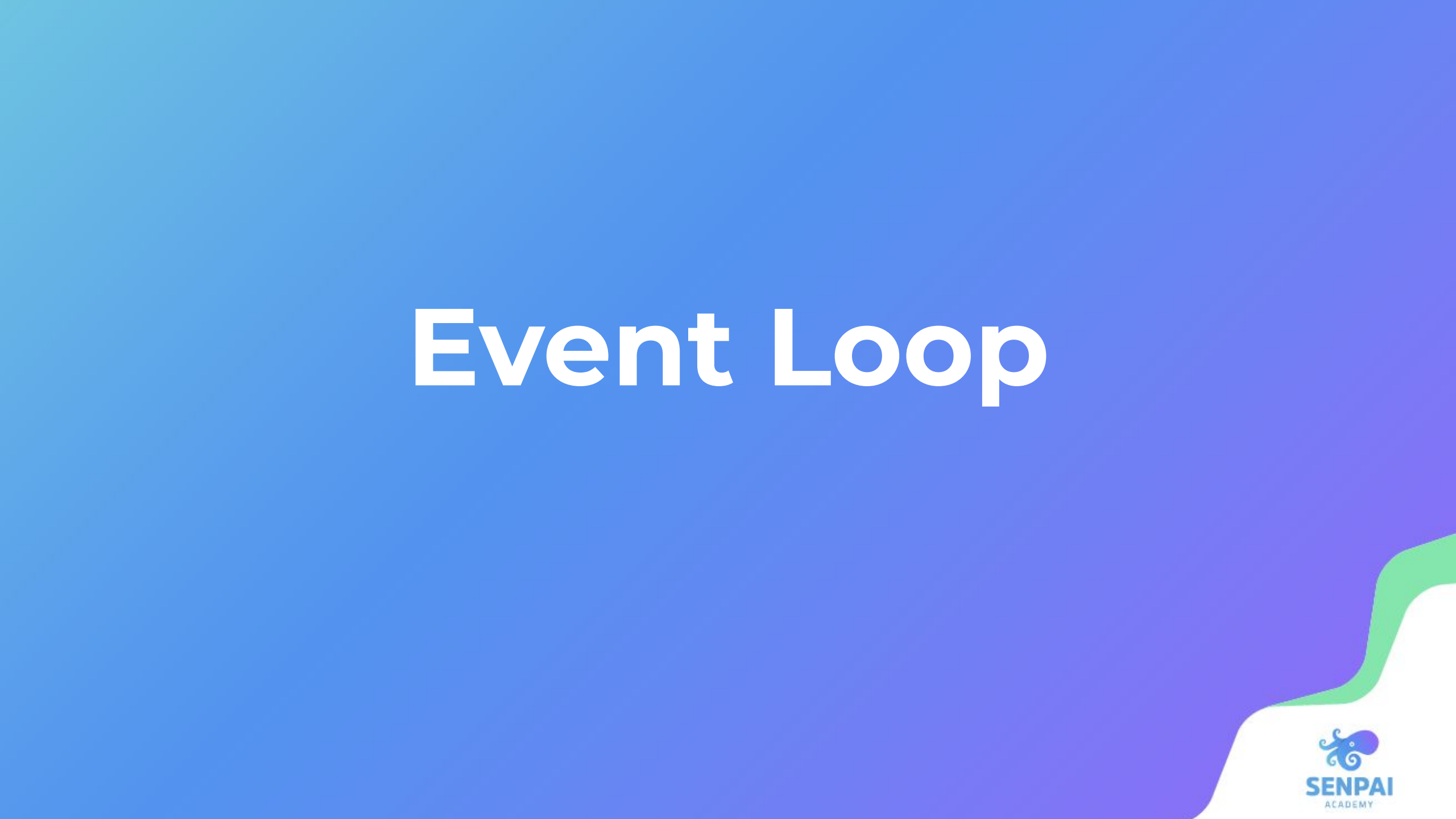
Los ***callbacks*** son funciones que le damos a Node.js para que ejecute al finalizar determinada operación (que puede llevar mucho o poco tiempo).

Esto nos permite tener todas las operaciones de E/S que el sistema operativo soporte ocurriendo a la misma vez.

Por ejemplo, en un servidor web con cientos o miles de requests pendientes con múltiples consultas bloqueadas, nos permite seguir respondiendo a las nuevas requests, y a medida que se resuelvan las bloqueadas, responder esas también.

SENPAI
ACADEMY

Event Loop



Node.js - Event Loop

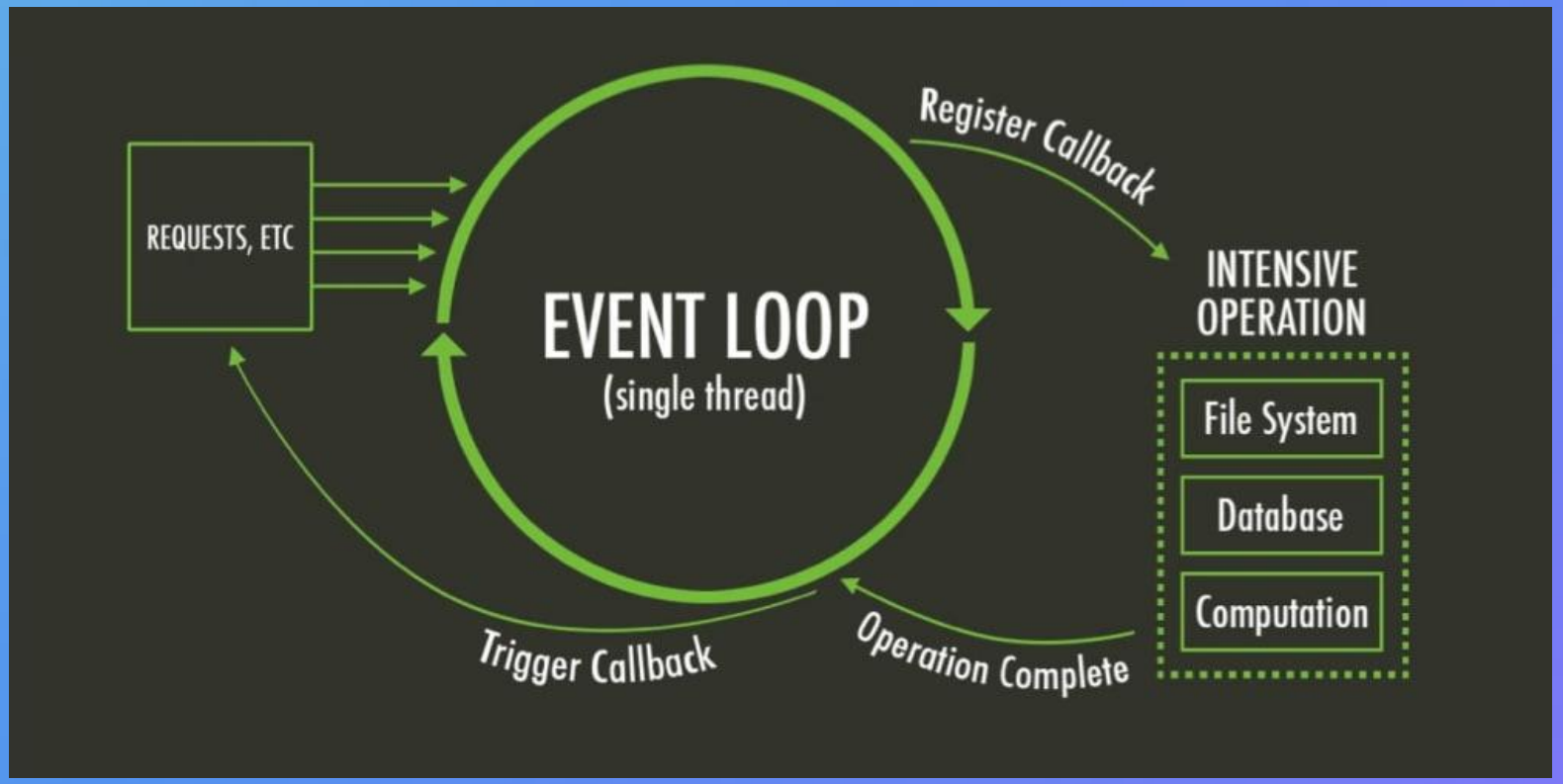
El “Event Loop” o “Bucle de Eventos” es lo que permite que Node.js sea asíncrono, y que no se bloquee con operaciones de E/S.

El código JavaScript corre en un hilo único (conocido como “single-thread”), lo que quiere decir que solo una sentencia se está ejecutando cada vez.

Esta limitación es en realidad muy útil, ya que nos ahorramos los problemas de concurrencia que surgen cuando programamos en entornos “multi-thread”.

SENPAI
ACADEMY

Node.js - Event Loop



SENPAI ACADEMY

Promesas

SENPAI
ACADEMY

Promesas

Las promesas son objetos de JavaScript que nos sirven para representar un valor que puede estar disponible **ahora**, en el **futuro**, o **nunca**.

Cuando una función o método nos retorna una promesa, esa promesa se va a encontrar en uno de estos estados:

- pendiente (pending): estado inicial
- cumplida (fulfilled): significa que la operación se completó con éxito.
- rechazada (rejected): significa que la operación falló.

Cuando una promesa se completa, o se rechaza, se invoca el método asociado ***then***, y si ocurre una excepción, se invoca el método asociado ***catch*** (es importante controlar los errores)

SENPAI ACADEMY

async / await

Node.js - async / await

async / **await** es una sintaxis que nos va a facilitar el uso de las Promesas.

Si ponemos la palabra clave ***async*** delante de la definición de una función, automáticamente la transformamos en una función asincrónica, permitiendo que dentro utilicemos ***await*** para invocar código asíncrono.

La palabra clave ***await*** se puede poner delante de cualquier función que retorne una promesa, y esto pausa nuestro código para esperar el resultado, en esa misma línea:

```
async function main () {
  const resultado = await multiplicarPositivos(15, 3)
  console.log('resultado', resultado);
}

main();
```

SENPAI ACADEMY

Node.js - async / await

Cuando usamos **async** / **await**, no estamos bloqueando la ejecución del programa, sino que simplemente le indicamos a Node.js que las siguientes líneas de ejecución precisan esperar el resultado de la operación asíncrona.

Para manejar los errores, osea, que la promesa sea rechazada o falle, podemos envolver nuestro bloque de código con una estructura de ***try ... catch***

```
async function main () {

  try {
    const resultado = await multiplicarPositivos(15, -3)
    console.log('resultado', resultado);
  }
  catch (exception) {
    console.log('Ocurrió un error', exception);
  }

}

main();
```

SENPAI ACADEMY

SENPAI
ACADEMY

gustavguez

gustavguez

GUSTAVO RODRIGUEZ

FULL STACK DEVELOPER
SOLCRE