# Event loop in node.js

Node.js is a single-threaded event-driven platform that is capable of running non-blocking, asynchronous programming. These functionalities of Node.js make it memory efficient. The **event loop** allows Node.js to perform non-blocking I/O operations even though JavaScript is single-threaded. It is done by assigning operations to the operating system whenever and wherever possible.

## Features of Event Loop:

- Event loop is an endless loop, which waits for tasks, executes them, and then sleeps until it receives more tasks.
- The event loop executes tasks from the event queue only when the call stack is empty i.e. there is no ongoing task.
- The event loop allows us to use callbacks and promises.
- The event loop executes the tasks starting from the oldest first.

### Example:

```
console.log("This is the first statement");

setTimeout(function(){
    console.log("This is the second statement");
}, 1000);

console.log("This is the third statement");
```

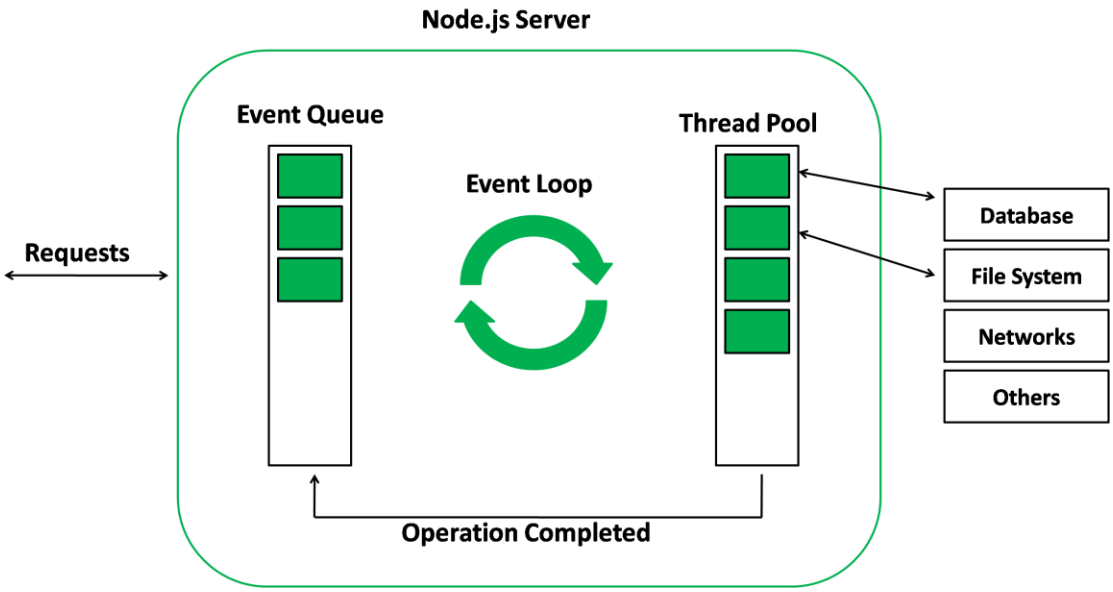
### Output:

```
This is the first statement

This is the third statement

This is the second statement
```

**Explanation:** In the above example, the first console log statement is pushed to the call stack, and “This is the first statement” is logged on the console, and the task is popped from the stack. Next, the setTimeout is pushed to the queue and the task is sent to the Operating system and the timer is set for the task. This task is then popped from the stack. Next, the third console log statement is pushed to the call stack, and “This is the third statement” is logged on the console and the task is popped from the stack.
When the timer set by the setTimeout function (in this case 1000 ms) runs out, the callback is sent to the event queue. The event loop on finding the call stack

empty takes the task at the top of the event queue and sends it to the call stack. The callback function for the setTimeout function runs the instruction and “This is the second statement” is logged on the console and the task is popped from the stack.

The following diagram is a proper representation of the event loop in a Node.js server:




**Phases of the Event loop:** The following diagram shows a simplified overview of the event loop order of operations:

- **Timers:** Callbacks scheduled by setTimeout() or setInterval() are executed in this phase.
- **Pending Callbacks:** I/O callbacks deferred to the next loop iteration are executed here.
- **Idle, Prepare:** Used internally only.
- **Poll:** Retrieves new I/O events.
- **Check:** It invokes setIntermediate() callbacks.
- **Close Callbacks:** It handles some close callbacks. Eg: socket.on(‘close’, …)

## References:

https://www.geeksforgeeks.org/node-js-event-loop/