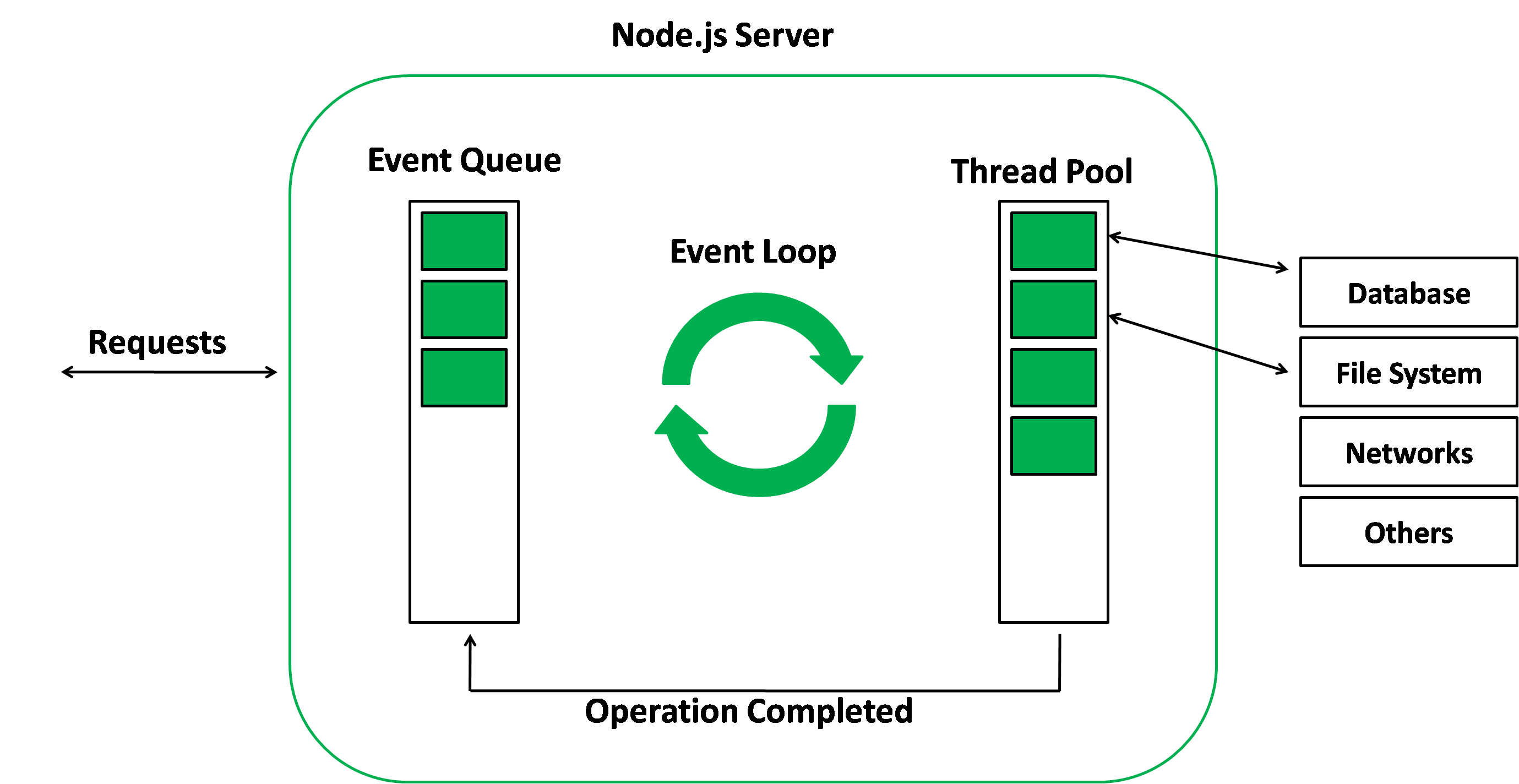
**Synchronous Operations –**

* Definition:
  + Synchronous operations are executed one after the other, in the exact order they appear in the code.
  + Each line of code must finish executing before the next one runs.
* Characteristics:
  + Each operation waits for the previous one to complete.
  + Runs in a single thread (main thread).
  + Simple and predictable.

### Asynchronous Operations

* Definition:
  + Asynchronous operations allow certain tasks (like network requests, timers, or file reading) to run in the background without blocking the execution of the rest of the code.
* Characteristics:
  + Non-blocking: The next line can execute without waiting for the current one to finish.
  + Uses Web APIs (in browser) or background threads (in Node.js) to handle delays.
  + Makes JavaScript more efficient and responsive.
  + Common Asynchronous Tools :
    - Promises
    - async/await
    - fetch



## What is Promise () -

1. A Promise () is an Object that represents the eventual failure of an asynchronous operation.
2. States of Promise -
   1. Pending - Neither rejected nor fulfilled
   2. Fulfilled - operation is completed
   3. Rejected - operation failed
3. Once the promise is fulfilled or rejected, it's considered as settled.

**Promise States**

A Promise is always in one of three mutually exclusive states:

**Pending**: The initial state. The asynchronous operation is still in progress.

**Fulfilled**: The operation completed successfully. The promise now has a resulting value.

**Rejected**: The operation failed. The promise has a reason for the failure, typically an error object.

Once a promise is fulfilled or rejected, it's considered **settled**. A settled promise can't change its state.

**How Promises Work**

When you create a new promise, you pass it a function called the **executor**. The executor takes two arguments: resolve and reject.

* resolve(value):
  + Call this function when the asynchronous operation is successful. This fulfills the promise with the given value.
* reject(reason):
  + Call this function when the operation fails. This rejects the promise with the given reason (usually an Error object).

You handle the result of a promise using the **.then(), .catch(), and .finally()** methods.

* .then(onFulfilled, onRejected):
  + This method is used to schedule a callback to be executed when the promise is fulfilled or rejected.
  + The first argument, onFulfilled, is a function that runs on success, receiving the fulfillment value.
  + The second argument, onRejected, is a function that runs on failure, receiving the rejection reason. It's common practice to use .catch() for handling rejections.
* .catch(onRejected):
  + This is a shorthand for .then(null, onRejected). It's a clean way to handle errors for promises.
* .finally(onFinally):
  + This method schedules a callback to be executed whether the promise is fulfilled or rejected. It's useful for cleanup code, like hiding a loading spinner, as it runs regardless of the outcome.

## async and await

* async and await are modern JavaScript keywords that provide a cleaner, more readable syntax for working with promises.
* They allow you to write asynchronous code that looks and behaves like synchronous code.

async -

* async is used to declare an asynchronous function.
* An async function always returns a promise.
* If the function returns a non-promise value, JavaScript automatically wraps it in a fulfilled promise.

await -

* await can only be used inside an async function.
* It pauses the execution of the async function until a promise is settled (either fulfilled or rejected).
* It then returns the fulfillment value of the promise.
* If the promise is rejected, await throws an error, which you can handle with a try...catch block.

Dummy api link – [click here](https://jsonplaceholder.typicode.com/users)

https://jsonplaceholder.typicode.com/users