Node js is single threaded for async processing, by doing so it can achieve more performance and scalability.

**How single threaded model perform concurrency?**

Node js processig is heavily influenced by js event based model and callback system, thus nodejs can easily achieve concurrency.

Event- loop is responsible for maintaining the concurrency in nodejs

**Event Emitters-** It emits named events as soon as something important happens in the app like request hitting servers, timer expiring, file finishing to read. And these events can be picked by event listeners which fires a callback function that is attached to each listeners.

**Streams-** Used to process(read and write) data piece by piece(chunks), without completing the whole read and write operation, and therefore without keeping all the data in memory.

A **callback** is a function that is called after a given task, it allows other code to run in the meantime, and prevents blocking. It is passed as an argument to another functions. IT is commonly used to handle asynchronous ops such as

reading files, making http requests, executing db queries etc.

**Promises**

Advantages of using promises instead of callbacks

The logic of asynchronous is more specific and structured.

Built-in errors

Improved readability.

**Modules** are like js libraries that can be used to include certain set of functions.

Some core modules are http, util, fs, stream etc.

**Libraries in nodejs are**

ExpressJs- It is nodejs web application framework that provides wide set of features

Mongoose- This framework is used to connect web app to a db

**Event loop** handles async callbacks in nodejs. it is the foundation of non-blocking I/O in nodejs.

**NextTick**() postpones the execution of action untill the next pass around the event loop. or it simply calls the callback once the loops current execution is completed.

**setImmediate**() executes a callback on next cycle of the event loop and returns the control to the event loop for any IO operations.

**A Nodejs server never waits for an api to return data. Instead it moves to next API after calling an API and a notification mechanism from a Nodejs event responds to the previous API calls**.

**ASYNC/AWAIT** is a feature in js that simplifies the asynchronous code that is easier to read and write.

**ASYNC** is a function that operates asynchronously via event loop.

**AWAIT** It is a keyword and only be used inside an async function. It pauses the execution of async function until a promise is setteled.

**Promises** are objects used for handling asynchronous operations. They represent a value that is represent now or in future or never.

Async ops like reading files, making a http request or querying from DB.

Promises has three states:

Pending- Initial state.

Fulfilled- The ops is successful.

Rejected – The ops failed.

**First Class Function-** when a function can be treated as any other variable it is called a first class function. It can be passed as a param to another function(callback), or a function can return another function(higher order function) like map, filter.

**Call , Apply, Bind**

**Call**- The (call) method calls a function with a given **this** value and arguments provided individually.

Syntax - functionName.call(thisArg, arg1, arg2, ...);

**Ex - function greet(greeting, punctuation){**

**Console.log(greeting + “, “ + this.name + punctuation)**

**}**

**Let person = {name : “Alok”};**

**greet.call(person, “Hello”, “!”); // Output: "Hello, Alok!"**

**Apply-**  The apply method calls a function with given this value, but arguments are provided as an array.

Syntax- functionName.apply(thisArg, [arg1, arg2, ...]);

**Ex - function greet(greeting, punctuation) {**

**console.log(greeting + ', ' + this.name + punctuation);**

**}**

**const person = { name: 'Alice' };**

**greet.apply(person, ['Hello', '!']); // Output: "Hello, Alice!"**

**Bind** – Creates new function with a specified **this value** and optionally initial arguments, which can be called with additional arguments.

Syntax - const boundFunction = functionName.bind(thisArg, arg1, arg2, ...);

Ex - function greet(greeting, punctuation) {

console.log(greeting + ', ' + this.name + punctuation);

}

const person = { name: 'Alice' };

const greetPerson = greet.bind(person, 'Hello');

greetPerson('!'); // Output: "Hello, Alice!"