

Namaste Node JS

ep-1 → Intro to Node.js

What is Node.js?

- Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.
- Node.js is built by Ryan Dahl in 2009.
- Node.js is maintained by the OpenJS Foundation.
- Node.js executes JavaScript code outside the web browser.

→ JavaScript is the language at the core of the web. It runs in browsers, but with Node.js, we can now run it everywhere. That is why node.js is often associated with the phrase "Run JavaScript everywhere".

- To run JavaScript, you always need a JavaScript engine.
- Everywhere you write JavaScript code, there is a "JavaScript Engine" that executes the code.
- Ryan started with the SpiderMonkey JavaScript engine, which is found in Firefox. However, within five days, he switched to V8 and never looked back.
- Eventually, Ryan was working independently, but his company name Joyent was working on something similar to Node.js. They hired Ryan to work under them, offering to fund his project - a big contribution from Joyent.
- The original name of Node.js was "web.js", but it was later named to "Node.js". because its scope extends beyond web servers.

Introduction

The Apache HTTP Server was a blocking server, so Ryan wanted to create a non-blocking server. This is why Node.js is a non-blocking I/O.

The advantage of a non-blocking server is that it can handle multiple requests with a limited number of threads.

In 2010, NPM was introduced. NPM is a package manager for Node.js. Node.js wouldn't be as successful without NPM.

Initially, Node.js was only built for macOS and Linux, but in 2011, it was also supported on Microsoft platforms.

In 2012, Isaac Z. Schlueter started maintaining Node.js; he is also the creator of NPM.

In 2014, a developer named Fedor Indutny developed a fork of Node.js named io.js,

leading to controversy within the company. A few developers began maintaining the io.js branch.

In 2015, Node.js and io.js were merged, forming the Node.js Foundation.

In 2019, the JS Foundation and Node.js Foundation merged to form the OpenJS Foundation, which currently maintains Node.js.

* Node Js - JS on Server

Node.js allows developers to use JavaScript both on the client-side and the server side, providing a unified language and ecosystem.

What is a Server?

A Server means nothing but a remote Computer.

You can assume that if it's a Computer, it's a

CPU working remotely.

You can access server's resources over a network to provide services to another Computer program.

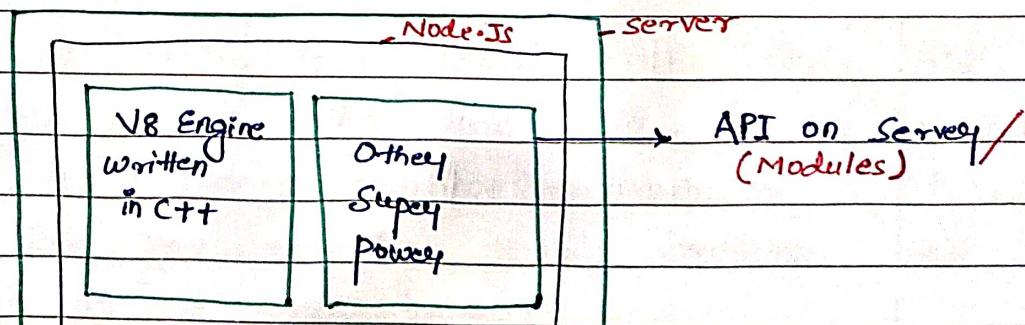
• V8 Engine - V8 (Google) → is written on C++

The Javascript engine uses C++ to execute JavaScript Code.

V8 is Google's open-source high-performance JavaScript and WebAssembly engine, written in C++. It is used in Chrome and Node.js among others. It is cross-platform.

- V8 can be embedded into any C++ application.

2 The Node.js is a C++ application with V8 embedded into it.

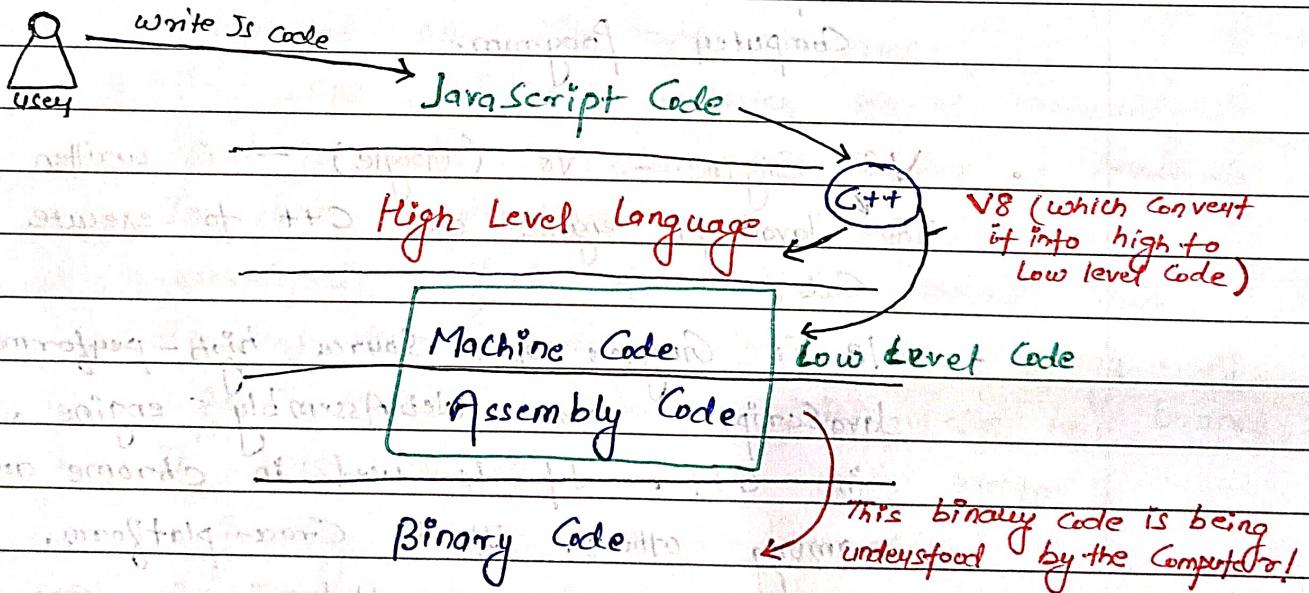


ECMA Script

- This are the Standard / Rules

[JavaScript Engine follows these standards]

- Why V8 is a C++ Code?? What is the use of it.
- Whenever we write a JavaScript Code, the JavaScript Code is processed by C++ program / JS Engine (which converts high level lang to low level) that code is converted to machine code and this is being understood by computer in the form of Binary.



Let's write Some Code

1. Installing Node.js

- First, let's start by installing Node.js on your screen.
You can download the latest version of Node.js
from the website (<https://nodejs.org/en>).

Node.js includes 'npm' (Node package Manager), which
allows you to manage and install packages for
your JavaScript projects.

2. Verification of Installation

- After installation, verify that Node.js and NPM
are installed correctly by running the following
command on your terminal.

`node -v`
`npm -v`

3. Writing Code

Using the Node REPL (Read, Evaluate, Print, Loop)

Node REPL is an interactive environment where
you can write and execute JavaScript code
directly in the terminal. It allows you to
quickly test and experiment with JavaScript
without needing to write code in a file first.

This is all running in a Node.js runtime environment.

- When you use the Node REPL, you are interacting
with the Node.js runtime environment, which is
responsible for executing your JavaScript code
outside of a web browser.

Node.js is a JavaScript runtime environment, and behind the scenes, it uses the V8 engine.

- The V8 engine, developed by Google, is the same engine that powers the Chrome browser. It takes your JavaScript code and compiles it into machine code, which your computer can execute. Node.js leverages this powerful engine to run JavaScript on the server side.

It's similar to the browser's console.

- Just like the console in your web browser, the Node REPL, allows you to write and test JavaScript code interactively. However, instead of running in the browser, it runs on your system's terminal, allowing you to interact with the underlying operating system and perform tasks like file handling, server management, and more.

However, waiting code like this for a long time can be frustrating.

- While the REPL is great for quick experiments, it's not ideal for larger projects or more complex code. The lack of features like syntax highlighting, debugging tools, and project organization makes it cumbersome for extended use.

Now, let's write some basic JavaScript code inside app.js file

```
const name = "Node.js";  
let a = 5;  
let b = 10;  
let c = a + b;  
console.log(name);  
console.log(c)
```

- Now write command node app.js inside the terminal

to run the code.

* Global object in Node.js

What are Global objects?

Global objects are objects that are available in all scopes. In Node.js, these objects provide core functionalities without needing to require them explicitly in your code.

- Difference between Browser and Node.js Global Objects.

- In a web browser, the global object is called 'window'.

The 'window' object is automatically created by the browser environment and provides access to various

browser-specific features, like the Document Object

Model (DOM), as well as methods like 'alert()',

'setTimeout()', and 'setInterval'. It's important

to note that the 'window' object is not part of

the V8 engine itself but is provided by the browser environment.

Global Object in Node.js

- In Node.js, the global is called 'global'.
- Similar to the window object in browsers, the 'global' object in Node.js provides access to globally available functionalities, such as:

'SetTimeout ()'

'SetInterval ()'

'Console'

'Process'

Write `Console.log (global);` inside your code and check the terminal.



- Global objects are not part of the V8 engine itself; they are provided by Node.js, which is built on top of V8. Node.js extends the capabilities of the V8 engine by adding these global objects, allowing it to perform tasks like file handling, interacting with the operating system, and handling asynchronous events.

Global This

This is always the global object, regardless of where it is accessed. It was introduced in ECMAScript 2020 to provide a standardized way to refer to the global object in any environment (browsers, Node.js, etc.).

- In browsers, globalThis is equivalent to window.
- In Node.js, globalThis is equivalent to global.
- It provides a consistent way to access the global object without worrying about the environment.

Ep-4 | Module.export & require

We need two diff. file, have different code that is not related to each other, so, in Node's we call them separate modules.

- How do you make two modules work together?
 - Using a **require function**

- What is the requirement function?

In Node.js, the **require()** function is a built-in function that allows you to include or require other modules into your main module.

Now, let's write our code using **require** function

With this file has definition of sum function

sum.js Let **x = "Export in Node"**;

function **CalculateSum(a, b)** {

let sum = a + b;

Console.log(sum);

// exporting

module exports = {

x : x,

CalculateSum = CalculateSum,

}

app.js -

```
const obj = require("./sum.js");
```

```
let name = "Node.js"
```

```
let a = 5;
```

```
let b = 10;
```

```
obj.CalculateSum(a,b);
```

```
console.log(obj.x); or console.log(x);
```

- When using the following import statement

```
const {x, CalculateSum} = require("%sum");
```

You can omit the .js extension, and it will still work correctly. Node.js automatically resolves the file extension for you.

- To use private variables and functions in other modules, you need to export them. This allows other parts of your application to access and utilize those variable and functions.

We have studied Common JS modules (CJS), and there is another type of module known as ES modules (ESM), which use the .mjs extension.

Common Modules (CJS)

- module.exports require()
- by Default used Node.js
- Old way
- Synchronous
- non-strict

ES Modules (ESM) (.mjs)

- import export
- By Default used in frameworks like react, angular.
- Newer way
- async
- strict

There are two major differences between these two module systems that are important to note:-

- **Synchronous and Asynchronous** - Common JS requires modules in a synchronous manner, meaning the next line of code will execute only after the module has been loaded. In contrast, ES modules load modules asynchronously, allowing for more efficient and flexible code execution. This distinction is a powerful feature and an important point to remember for interviews.

• **Strict Mode** - Another significant difference is that CommonJS code runs in non-strict mode, while ES modules execute in strict mode. This means the ES modules enforce strict parsing and error handling, making them generally safer and more reliable.

Overall, ES modules are considered better due to these advantages.

First, you need to create a new file called `package.json`.

To use ES modules, you must include the following in your `package.json`.

`package.json`

{

 "type": "module"

}

This setting indicates that your code will use ES module syntax.

What is `module.exports`?

`module.exports` is an empty object by default.

Ep5 - Diving into the Node.js Github

* How require() Works Behind the Scenes!

1. Resolving Module

Node.js will determine the path of the module. It checks whether the path is a local file (.local), JSON file (.json), or a module from the node_modules directory, among other possibilities.

2. Loading the Module

- Once the path is resolved, Node.js loads the file (Content on its type). The loading process varies depending on whether the file is JavaScript, JSON, or another type.

3. Wrapping Inside an IIFE

- The module code is wrapped in an Immediately Invoked Function Expression (IIFE). This wrapping helps encapsulate the module's scope, keeping variables and functions private to the module.

4. Code Evaluation and module.exports

- After unwrapping, Node.js evaluates the module's code. During this evaluation, module.exports is set to exports available to other files.

5. Caching (very imp)

- Importance:** Caching is crucial for performance. Node.js caches the result of the require() call so that the module is only loaded and executed once.

- Why Can't we access Variables / Functions directly?
- All the Code of a Module is wrapped inside a Function, when you all Require.

The Function is a special Function known as (IIFE)

IIFE -> Immediately Invoked Function Expression.

- How do you get access to module.exports where it comes from?

- At the end of the day all the Js Code is wrapped inside a function & gets executed (IIFE)

(Function) (module Require) { }

require ('./Path') → Given by Node

module.exports = {}

Node.js Passes Modules as a parameter to IIFE in which the Code is wrapped.

Node.js takes your code wraps it inside IIFE & sends it to V8 for execution V8 Engine known as

how to execute IIFE.

JavaScript is a Synchronous Single Threaded

→ Single Thread - It can run on a single Thread or a Single Process.

Synchronous - One after another.

- Js engine on V8 will run the Js code as soon as it sent to Js engine.

To Run Js we don't need multiple Threads we just need one thread.

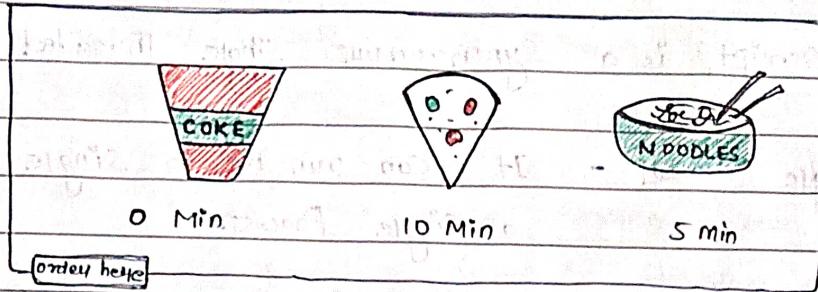
- If we are executing line 3 the line 4 will be executed after that.

- Js is only capable of running on a single piece of thread.

- In Synchronous Programming, The program execution is blocked until the current task is finished.

- Inefficient for I/O-bound tasks (e.g., file reading, network requests) as it can block entire thread until task is completed.

Diff b/w Synchronous and Asynchronous operation!



Customers	Items	Synchronous	Asynchronous
A → Coke		0 min	0 min
B → Noodles		5 min	5 min
C → Pizza		10 min	10 min
D → Coke		15 min	0 min
E → Noodles		20 min	5 min

Here, order can only be fulfilled, Once the previous order is Fullfilled.

Generally not a good way & is Blocking thread.

Whereas, in case of Asynchronous execution -

All orders with Coke will get it at 0 min
the second will get it in 10 min all the
noodle orders will be fullfilled in 5 min.
no one has to wait for any other order.

It is a non-blocking operation.

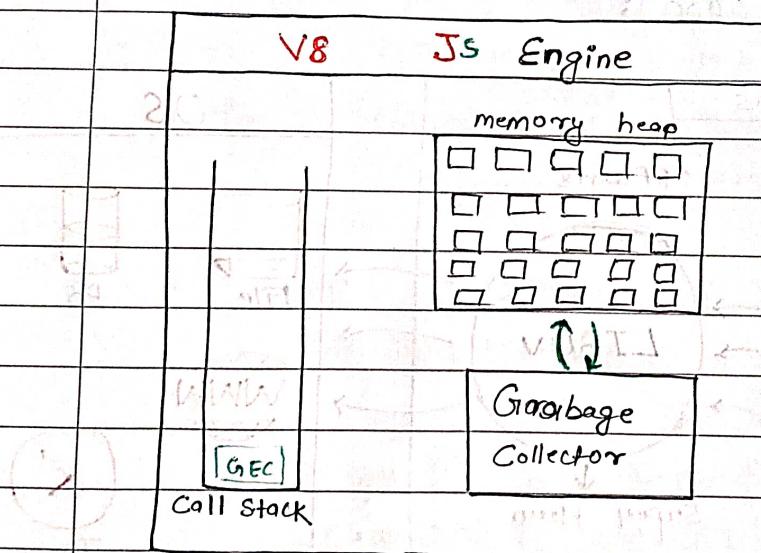
1/1

- How Synchronous Code is executed in JavaScript?

3 < Var a = 407429;
3 ()
Var b = 172410;

```
function multiplyFn (x,y) {  
    const result = x * y;  
    return result;  
}
```

Var c = multiplyFn(a,b)



Whenever you run the code, a global execution context is created and it is pushed to the call stack.

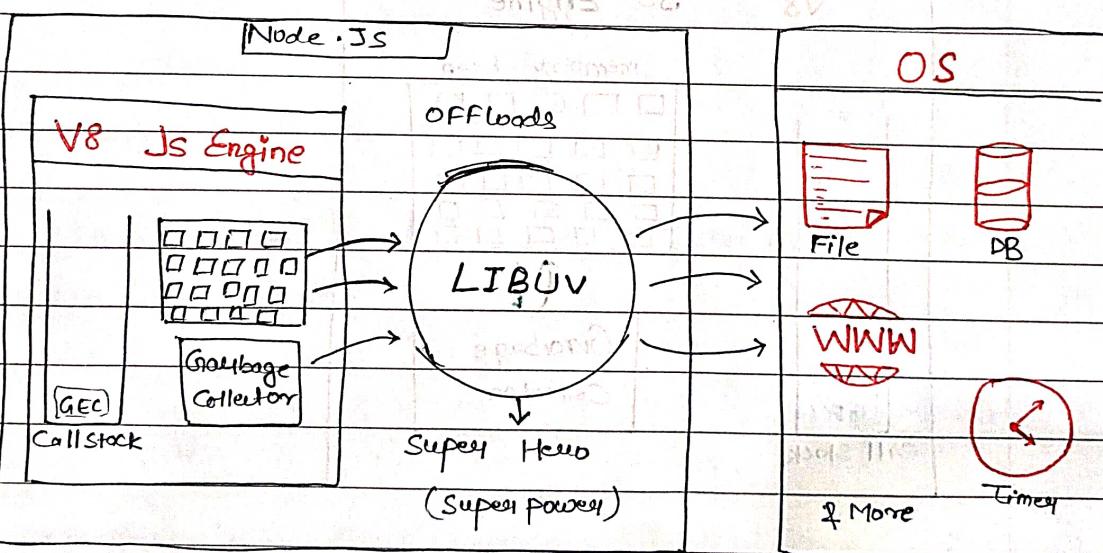
- Whenever a fn comes, it is pushed into Call Stack and all the result of the calculations will be stored into Heap. Once execution is over to result is returned to GEC (Global Execution Context) and fn moves out of the Call Stack.
- Once the whole code is executed Call Stack will become Empty!

• How Asynchronous Code is Executed in JavaScript

```
http.get ("https://api.firebaseio.com", (res) => {
    console.log ("Secret data : " + res.secret);
});
```

```
fs.readFile ("./gossip.txt", (data) => {
    console.log ("File Data", data);
});
```

```
setTimeout (() => {
    console.log ("Wait here for 5 min");
}, 5000);
```



Libuv - Libuv is a multiplatform C library that provides support for asynchronous I/O-based event loops.

Asynchronous I/O made simple

Whenever V8 engine sees API call, file load operations, setTimeout, if offloads this to Libuv, and then Libuv manages it.

- Node.js is Asynchronous by V8 engine is synchronous

- Node.js can do Asyn I/O

Non-Blocking I/O

So, even with a single thread, it can do so many async operation together.

• How a code executes while containing both sync and Async operations?

```
Var a = 427920;
```

```
Var b = 127221;
```

```
Req = https.get("https://api.fbi.com/v1/secret")
```

```
(res) => {
```

```
  console.log(res?.Secret);
```

```
};
```

```
function MultiplyFn(x, y) {
```

```
  const result = a * b;
```

```
  return result;
```

```
}
```

```
Var c = MultiplyFn(a, b);
```

```
console.log(c);
```

• How

- Firstly GEC is created inside call stack, Code inside GEC will now execute in Sync single-threaded way.

- Memory will be allocated to a, b var & Garbage Collector will work in Sync with Memory heap.

- For Api Calls, Libuv will manage the Api Call. it will Register the Api Call, and takes the callback meanwhile libuv is managing the Api Call Js engine will move to the next line.

- ~~All~~ All async task will be offloaded to libuv.

- Now Js engine will quickly execute multiply fn and new fn Context will be created & it will be executed.

- Once the call stack get empty, all the memory will be cleaned by Garbage Collector.

- Once libuv is done with all the tasks and if sees that the call stack is empty.

- early Api Call is success it will put the CB() to call stack.

- Call stack will execute all the stuff inside it quickly.

Code :-

async.js

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

console.log("Hello world");

https.get("https://dummyjson.com/1", (res) => {
    console.log("Fetched Data Successfully");
});
```

```
setTimeout(() => {
    console.log("setTimeout called after 5 sec");
}, 5000);
```

```
fs.readFile("./file.txt", "utf8", (err, data) => {
    console.log("File Data:", data);
});
```

```
function multiplyFn (a, b) {
    const result = a * b;
    return result;
}
```

```
var c = multiplyFn (a, b);
```

```
console.log ("multiplication result is: " + c);
```

Output -

- Hello world
- ~~Multiplication result is = 924728104291~~
- File Data : This is the file Data of Home
- Fetched Data successfully
- SetTimeout called after 5sec

UTF-8 - is a variable length character encoding for electronic communication. Defined by unicode standard, the name is derived from the Unicode Transformation Format - 8 bit.

Synchronous function - will block the main thread don't use it.

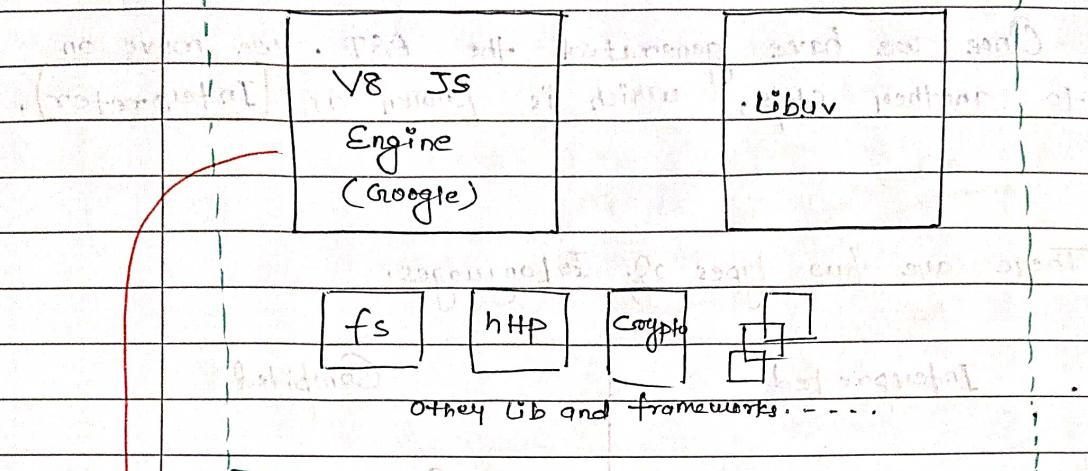
// This callback will only be executed to once the call stack is empty.

```
console.log("Call it straight now!");
```

```
, 0);
```

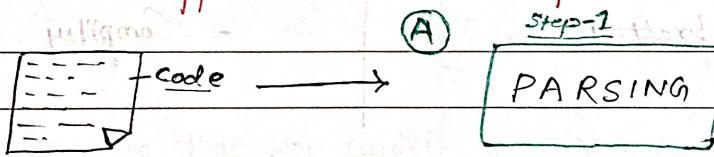
This will be executed when all the synchronous code will get executed and call stack become empty.

|- Node.js



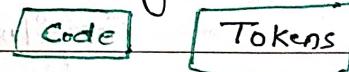
→ We are going to deep inside V8 JS Engine

#1 what happens when we write a Code

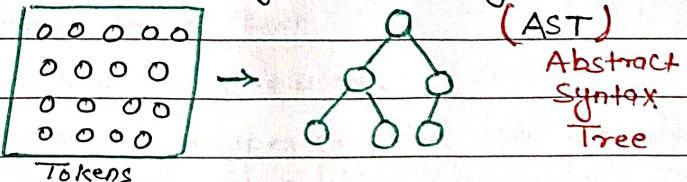


↓ Step-2

Lexical Analysis (Tokenization)



Syntax Analysis (Parsing)



- So, whenever you give any Piece of Code to Javascript V8 Engine, it goes through the first step known as PARSING state.

The Parsing state itself do the lexical analysis, code is broken down into tokens.

Now, the Syntax analysis happens to this tokens.
and the Tokens is converted to a AST.

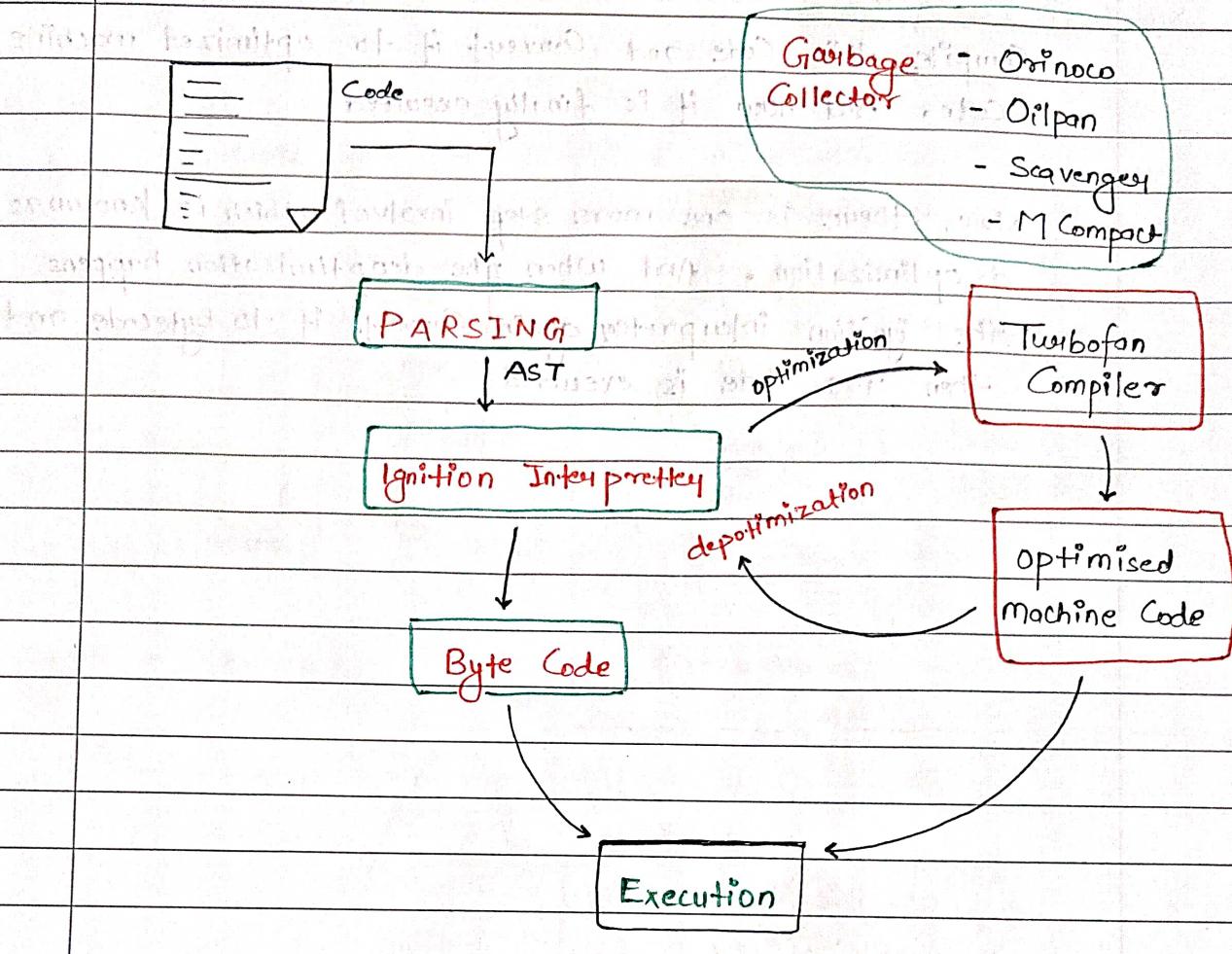
Once we have generated the AST. we move on
to another step, which is known as Interpreter.

There are two types of Languages.

<u>Interpretted</u>	<u>Compiled</u>
- Line by Line	- first Compilation Hc Code → Mc Code
- Fast initial execute	- Initially slowly but executed fast
- Interpreter	- Compiler

∴ JavaScript V8 engine uses both interpreted,
If was JIT Compilation (Just in time
Compilation) JIT Compiler.

V8 Architecture



The Code that you write ex - var A = 10

- A lot of things happens behind the scenes are as follows :-

- The Code is broken down into tokens
- Then these tokens are converted into AST
- The AST goes to the Interpreter.
- Interpreter converts it to byte.
- Then the byte code is executed.

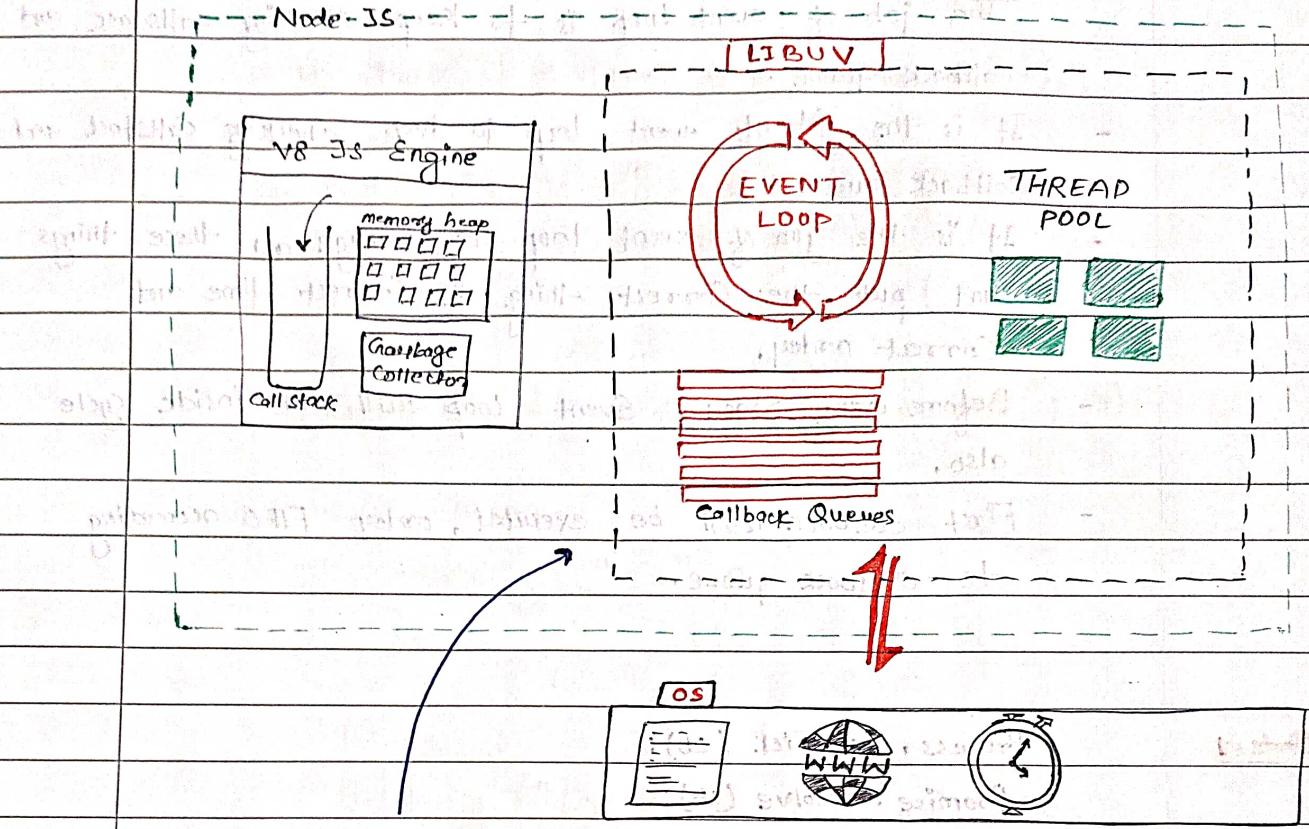
and in b/w Compiler is happening, In b/w garbage collection is happening. This is how V8 Engine works.

- Now in b/w of this Ignition Interpreter, if it finds the opportunity to optimize the code. It will pass that to Turbofan Compiler. The step there is known as optimization. And now, Turbofan Compiler will

Compile this Code and Convert it to optimized machine code. And now it is finally executed.

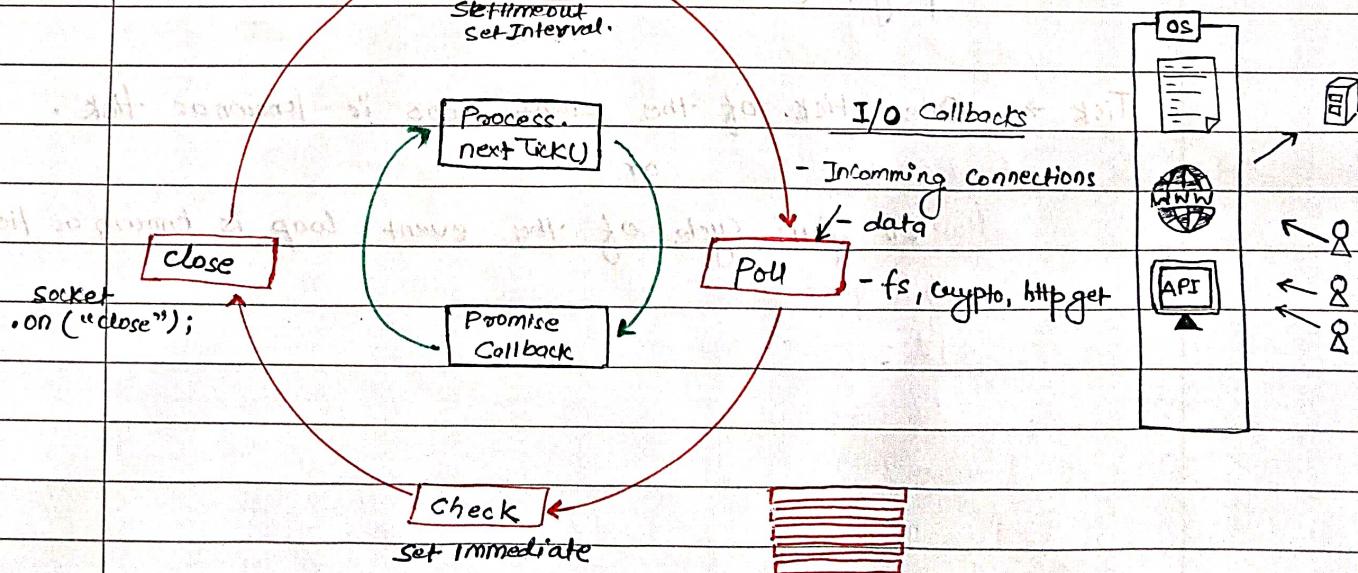
Now, there is one more step involved which is known as deoptimization. And when the deoptimization happens the ignition interpreter again converts it to bytecode and then the code is executed.

ep-9 / libuv & Event Loop



Asynchronous I/O (Non-blocking I/O)

"JavaScript is (Synchronous) single-threaded language."



Event Loop

- until V8 JS Engine callstack is not empty, libuv will push all the completed async task in Callback Queue.
- The job of event loop is to keep checking callstack and callback queue.
- It is the job of event loop to keep checking callstack and callback queue.
- It is the job of event loop to manage all these things and put the correct thing in correct time and correct order.
- Before every phase, Event loop will run inside cycle also.
- First callback will be executed, order FIFO according to callback queue.

Code ex:
process.nextTick(cb);

Promise.resolve(cb);

(or) setTimeout(cb, 0);

setImmediate(cb);

fs.readFile("./file.txt", cb);

https.get("URL", cb);

Tick → One tick of the event loop is known as tick.

or

One full cycle of the event loop is known as tick.

Code 1 -

```
Const a = 100
```

```
SetImmediate(() => console.log("SetImmediate"));
```

```
fs.readFile("./file.txt", "utf", () => {
    console.log("File reading CB");
});
```

```
setInterval(() => console.log("Timer expired"), 0);
```

```
((function() {
    function PrintA() {
        console.log("a =", a);
    }
    PrintA();
}))
```

```
Console.log("Last Line of the file");
```

- output : ("a=100" as edit field)

a = 100

Last Line of the file

Timed expired

SetImmediate

File reading CB

Code 2 -

Const a = 100

setImmediate(() => console.log("SetImmediate"));

:Promise.resolve(() => console.log("Promise"));

fs.readFile("./file.txt", "utf8", () => {

console.log("File Reading CB");
});

:() { SetTimeout(() => console.log("Timer expired"), 0);

process.nextTick(() => console.log("Process.nextTick"));

function PointA() {
 console.log("a =", a);
}

PointA();

console.log("Last line of the file");

- Output

a = 100

Last line of the code

Process.nextTick

Timer expired

Set Immediate

File reading CB

Code-3

```
SetImmediate () => console.log ("SetImmediate");
```

```
setTimeout () => console.log ("Timer expired"), 0
```

```
: file Promise.resolve () => console.log ("Promise"));
```

```
fs.readFile ("./file.txt", "utf8", () => {
```

```
setTimeout () => console.log ("2nd Timer"), 0);
```

```
process.nextTick () => console.log ("2nd nextTick));
```

```
SetImmediate () => console.log ("2nd SetImmediate));
```

```
console.log ("File reading CB");
```

```
);
```

```
process.nextTick () => console.log ("nextTick));
```

```
console.log ("Last line of the file");
```

- Output

Last line of the file

nextTick

Promise

Timer expired

SetImmediate

File Reading CB

2nd nextTick

2nd Set Immediate

2nd Timer

Code -4

```
const fs = require("fs");
SetImmediate(() => console.log("SetImmediate"));
setTimeout(() => console.log("Timed expired"), 0);
Promise.resolve(() => console.log("Promise"));
fs.readFile("./file.txt", "utf8", () => {
    console.log("File Reading CB");
});

process.nextTick(() => {
    process.nextTick(() => console.log("inney nextTick"));
    console.log("nextTick");
});

console.log("Last line of the file");
```

- Output

Last line of the code

nextTick

inney nextTick

Promise

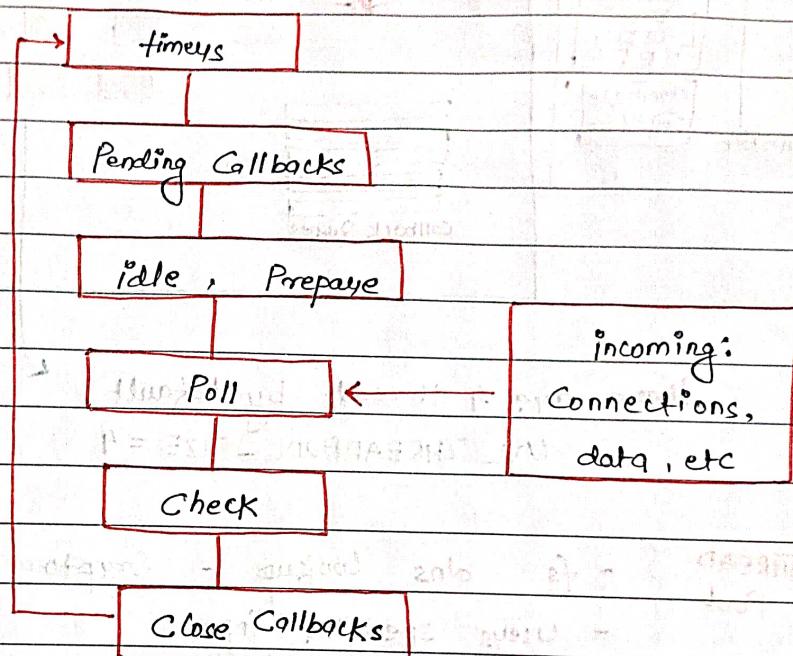
Timed expired

SetImmediate

File reading CB

Event Loop

- When Node.js starts, it initializes the event loop, processes the provided input script (or drops into the REPL, which is not covered in this document) which may make API calls, schedule timers, or call `process.nextTick()`, then begins processing the event loop.



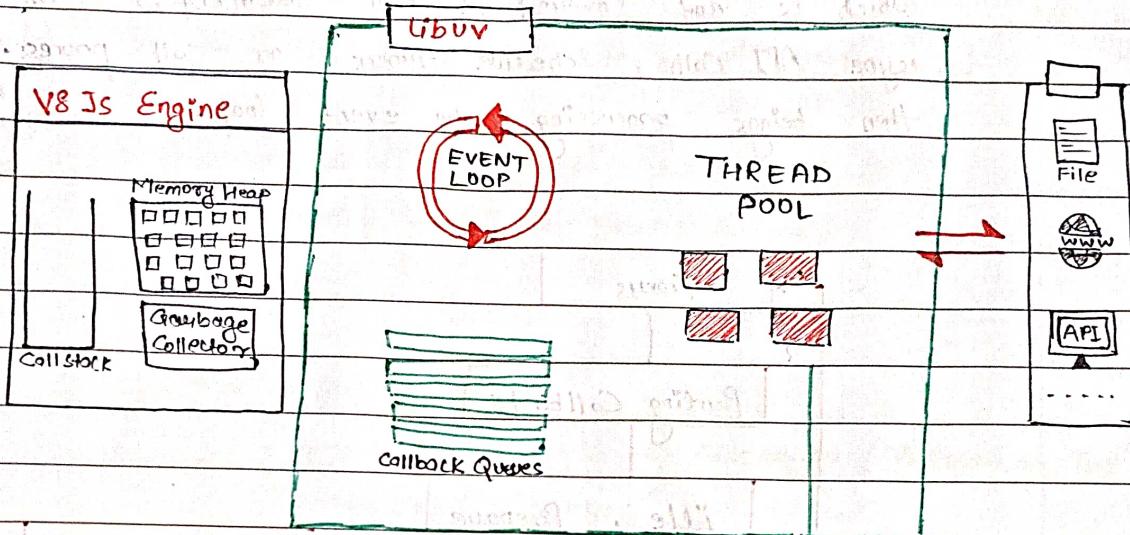
Each box will be referred to as a "Phase" of the event loop.

- **timers**: this phase executes callbacks scheduled by `SetTimeout()` and `SetInterval()`.
- **Pending Callbacks**: Executes I/O cb deferred to the next loop iteration.
- **idle, prepae**: Only used internally.
- **Poll**: retrieves new I/O events, executes I/O related cb (almost always with the exception of close cb, the one scheduled by timers, and `SetImmediate()`; node will block here when appropriate).
- **Check**: `SetImmediate()` callbacks are invoked here.
- **Close callbacks**: Some close callbacks, e.g. `socket.on('close')`.

Thread Pool

{ Process. UV_THREADPOOL_SIZE }

Is Node.js Single-threaded or Multi-threaded?



There are 4 threads by default!

UV_THREADPOOL_SIZE = 4

THREAD POOL
- fs dns lookup - Crypto
- Uses specified input

Is Node.js Single-threaded or Multi-threaded?

→ If you're giving it an synchronous task then Node.js is single-threaded but if you're giving some asynchronous task if used UV Threadpool then Node.js is multithreaded.

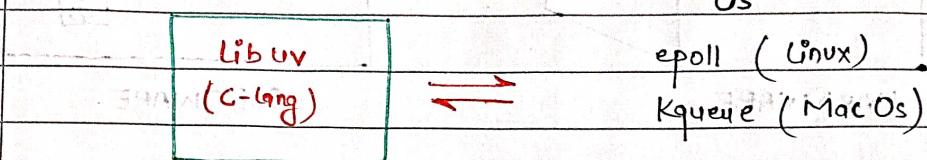
So, the answer should be it depends! what task you're going to perform.

You can change the number of threads by:

- `Process.env.UV_THREADPOOL_SIZE = 8;`

whatever size you want you just need to add the no.

All the networking happens on Socket, API calls does not
uses thread pool.



Libuv uses processes (Scalable I/O event Notification Mechanism)

Main Teachings -

"Don't Block the main Thread"

→ Sync methods - Heavy JSON objects

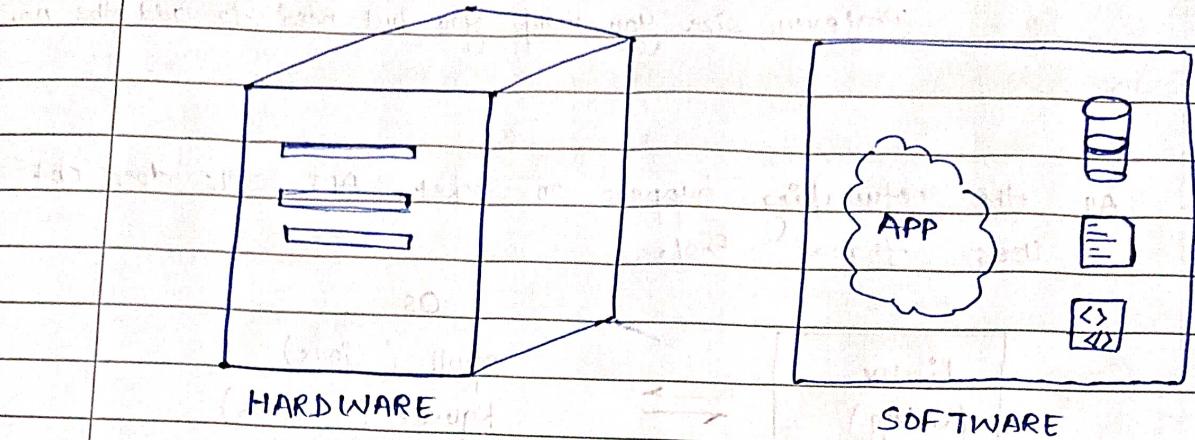
→ Complex Regex - Complex Calculations / Loops

"Data Structure is important"

"Naming is very important"

Ep-11 / Creating a Server

Server



We can use our computers as servers, we don't need AWS, But there are some limitation.

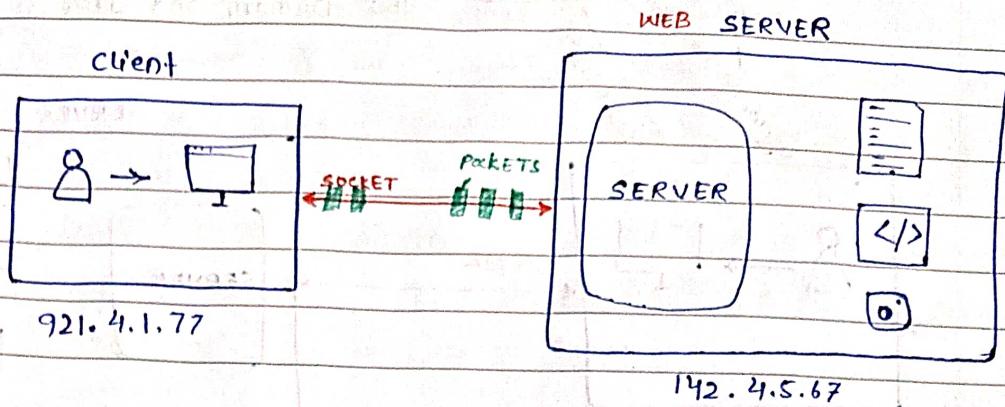
- Limited RAM
- Always up and running
- We use a local Internet Provider, they don't guarantee your IP. But AWS guarantee you a IP.

AWS has big data centers in multiple regions.

You are creating a HTTP server using Node.js

You are creating an application that can handle user requests.

Client - Server Architecture



TCP / IP

Types of Servers:-

HTTP Server - Hyper Text transfer Protocol

SMTP Server - Simple Mail transfer Protocol

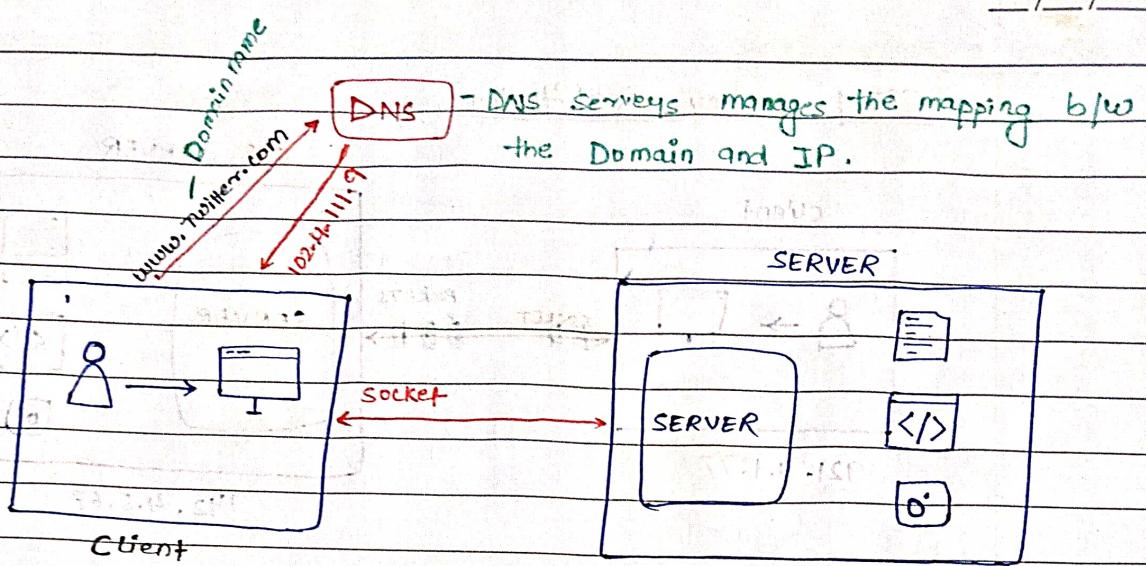
FTP Server - File transfer protocol

When we talk about Node, we talk about HTTP Server in web server.

Server listens to the requests

Packets - Data are sent in small chunks called Packets.

- We can create multiple of ^{HTTP} Server inside a Server, but you would need a different Port number (localhost:3000)
 - Port Number is needed so that other people can connect to your applications.



- A server can talk/connect to another server also.

Socket	Web Sockets
<ul style="list-style-type: none"> It socket, you make a connection, complete your task and close the connection. It takes less resources. 	<ul style="list-style-type: none"> When a user makes a connection, it stays for a long time. It takes more resources.

Server.js

```
const http = require ('node:http');
```

```
const server = http.createServer();
```

```
server.listen (7777);
```

```
const server = http.createServer (function (req,res){
    res.end ("Hello World!");
});
```

Express - Node.js Web application framework.

Q. What is database? what is DBMS?

- A database is an organized collection of data, or a type of data store based on the use of a database management system.
- DBMS is a software that interacts with end users, applications and the database itself to capture, store and analyze the data.

Types of Databases

1. Relational DB - MySQL, PostgreSQL
2. NoSQL DB - MongoDB
3. In-memory DB - Redis
4. Distributed SQL DB - Cockroach DB
5. Time Series DB - Influx DB
6. OO DB - db4o
7. Graph DB - Neo4j
8. Hierarchical DB - IBM IMS
9. Network DB - IDMS
10. Cloud DB - AMAZON RDS

Every database has its own server.

RDBMS (MySQL, PostgreSQL) - Oracle is managing MySQL

NoSQL (MongoDB) - "Not Only SQL"

Some time Node.js Comes.

MongoDB Comes in 2009

RDBMS (MySQL)

- Table, Rows, Columns
- Structure Data
- Fixed Schema
- SQL
- Tough horizontal scaling
- Relationships = Foreign keys + joins
- Read heavy apps, transaction workloads
- Ex - Banking apps

NOSQL (Mongo DB)

- Collection, document, field
- Unstructured Data
- flexible schema
- Mongo (MQL), Neo4j (cypher)
- Easy to scale horizontally & vertically
- Nested (Relationship)
- Real time, Big Data
- distributed Computing
- Ex - Real time analytics, Information and Social media.

In NOSQL (Mongo DB)

- No need for joins
- No need for Data normalization

Database ↔ Database

Table ↔ Collection

Row ↔ Documents

Column ↔ Fields

Ep-13 | Creating a database & MongoDB

How to connect mongo db to your project.

npm i mongodb

database.js

```
const { MongoClient } = require('mongodb');
const url = "Your Connect String";
const client = new MongoClient(url);
const dbName = 'HelloWorld';

async function main() {
    await client.connect();
    console.log("Connected successfully to server");
    const db = client.db(dbName);
    const collection = db.collection("user");
    return "done";
}
```

Main()

- then (console.log)
- Catch (console.error)
- finally (() => client.close());

// Read

```
const findResult = await collection.find({}).toArray();
```

```
console.log("found document =>", findResult);
```

// Insert documents

```
const data = {
    firstname : "Virat",
    lastname : "Kohli",
    city : "Delhi",
    Phone no. : "9242321202",
};
```

```
const insertResult = await collection.insertMany([data]);  
console.log("Inserted document =>", insertResult);
```

11 Count Document

```
const countResult = await collection.countDocuments({});
```

```
console.log("Count of document in the user collection =>", countResult);
```

```
return "done";
```

```
}
```

if (err) return "error";

else if (docs.length > 0)

if ("name" in docs[0]) info += "Name: " + docs[0].name;

: (constant) (E. g., name = did) + " does not have a name";

if ("post" in docs[0]) info += "Post: " + docs[0].post;

: "Posts" number

else if (docs[0].post.length > 0)

for (let i = 0; i < docs[0].post.length; i++)

info += "Post " + (i + 1) + ": " + docs[0].post[i];

else if (docs[0].post.length == 0)

info += "No posts found";

else if (docs[0].post.length < 0)

info += "Negative post length";

else if (docs[0].post.length > 1000)

info += "Post length exceeds limit";

else if (docs[0].post.length < 0)

info += "Post length is negative";

else if (docs[0].post.length > 10000)

info += "Post length exceeds limit";

else if (docs[0].post.length < 0)

info += "Post length is negative";

else if (docs[0].post.length > 100000)

info += "Post length exceeds limit";

else if (docs[0].post.length < 0)

info += "Post length is negative";

(1) `prefixed`: ({}) + " - prefix after each document"; good

(2) `labeled`: ({}, "label": "label"); bad, slow

(3) `unlabeled`: ({}) + " - label"; good

(4) `labeled`: ({}, "label": "label"); good

(5) `unlabeled`: ({}) + " - label"; good

(6) `labeled`: ({}, "label": "label"); good

(7) `unlabeled`: ({}) + " - label"; good

(8) `labeled`: ({}, "label": "label"); good

(9) `unlabeled`: ({}) + " - label"; good