Namaste Node JS

__/__/____

ep-1 → Intro to Node.js

What is Node.Js ?

→ Node.js is a java Script runtime built on Chromes V8 Java Script engine.

- Node.js is built by Ryan Dahl in 2009.
- Node.js is maintained by the Open Js Foundation.
- Node.js executes Java Script Code outside the web browser.

— Java Script is the Conguge at the Core of the web and browsers, but with Node.js, we can now run it everywhere. That is why node.js is often associated with the phrase "Run Java Script everywhere".

- To run javaScript, you always need a Java Script engine.
- Everywheye you write java Script Code, there is a "java Script Engine" that executes the Code.
- Ryan started with the Spidey Monkey javaScript engine, which is found in Firefox. Howevey, within two days, he swifched to V8 and nevey looked back.
- Intitally, Ryan was working independently, but a Company name Joyent was working on Something similay to Node.js. They hieyd Ryan to work undey them, offering to fund his project - a big Contribution fom Joyent.
- The Orignal Name of Node.js was "web.js", but is was lafey named to "Node.Js". because its scope extends beyond web serveys.

__/__/____

- The Apache HTTP servey was a blocking servey. So Ryan wanted to create a non-blocking servey. This is why Node.js is a non-blocking I/O. The advantage of a non-blocking servey is that it can handle multiple requests with a smallery numbey of threads.

- In 2010, NPM was introduced. NPM is a Package manageg 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 ~~Node.js~~ NPM.
- In 2014, a developey name Fedor Indutny developed a fork of Node.js named io.js, leading to Controveucy 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.

__/__/____

ep-2 → Js on Server

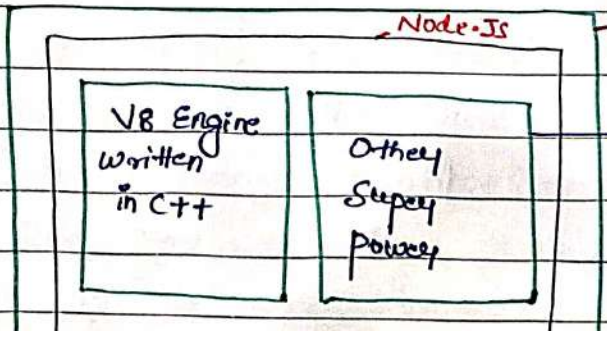
★ Node Js - Js on Server

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 it's a Computer, it's a CPU working remotely.
  - You can access servers remotely over a network to provide resources and 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.

✓ The Node.Js is a C++ application with V8 embedded into it.

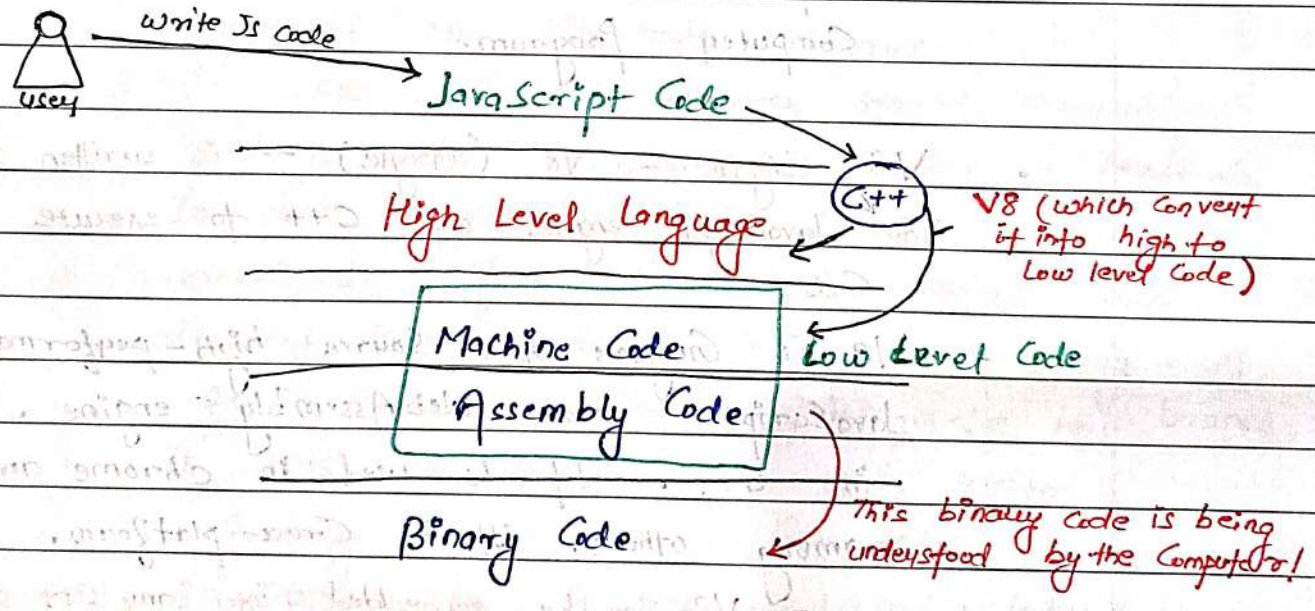


__/__/____

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 Convert high level long to low level) that Code and Converted to machine Code and this is being understood by Computer in the form of Binary.



__/__/____

ep-3 → Let's write Some Code!

Let's write some Code

1. Installing Node.js
   - First, lets 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 mange and install packages for your JavaScript projects.

2. Verification the Installation
   - After installition, Verify that Node.js and NPM are installed Correctoly by running the following Command on your terminal.

     node -v

     npm -v

3. Writing Code
   - Using the Node REPL (Read, Evalute, Print, Loop)
   - Node REPL is an interactive enviroment 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 ~~see~~ 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 the 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 browser 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 intract with the underlying operating System and perform tasks like file handling, Server management, and more.

However, writing 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 Projects Orginization makes it Cumbersome for extended use.

__/__/____

Now, lets write some basic javaScript Code inside app.js file

```
let 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 teuminal to run the Code.

★ Global object in Node.js

- What are Global objects?
  - Global objects are objects that are avilable in all scopes. In Node.js, these objects provide Core functionaties without needing to require them explicity in your Code.

- Difference between Browsey and Node.js Global Objects.
  - In a web browsey, the global object is Called 'window'. The 'window' object is automatically Created by the browsey enviroment and provides access to various browsey-specific features, like the Document object Model (DOM), as well as methods like 'alert()', 'settimeout()', and 'setInteival'. Its important to note that the 'window' object is not part of the V8 engine itself but is provided by the browsey environment.

__/__/____

Global Object in Node.js

- In Node.js, the global it called 'global'.
- Similar to the window object in browsers, the 'global' object in Node.js Provides access to globally aviloble functionalities, such as:

'SetTimeout()'
'SetInteval()'
'Console'
'Process'

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

★★★

- Global objects are not pault 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 peuform tosks like file handling, interacting with the opeuoting System, and handling asynchronous events.

Global 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 browseys 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 Nodejs we call them seprate 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 modules.

Now, let's write our Code using require function

sum.js -

```
Let x = "export in React exports in Node";
function CalculateSum (a, b) {
    let Sum = a+b;
    Console.log (Sum);
}

// exporting

module exports = {
    x : x,
    Calculate Sum = CalculatSum,
};
```

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)	Es Modules (Esm) (mjs)
• module.exports require()	• import export
• by Default used Node.js	• By Default used in frameworks like react, angulary.
• Oldey way	• Newey way
• Synchronous	• async
• non-strict	• strict

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

- Synchronous and Asynchronous - CommonJs requires modules in a synchronous manney, meaning the next line of code will execute only aftey 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 rembey 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 stricter 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.s github

* How require() works Behind the Scenes!

1. Resolving Module

- Node.js determines the path of the module. It checks whether the path is a local file (./local), a 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 is an Immediately Invoked Function Expression (IIFE). This wrapping helps encapsulate the module's scope. keeping variables and function private to the module.

4. Code Evaluation and module.exports

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

5. Caching (very imp)

- Importance: Coaching 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 does 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')
    module.exports = {}
})();
```

Given by Node

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 how to execute IIFE.

__/__/____

ep-6 | Libuv & async IO

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 on thread.

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

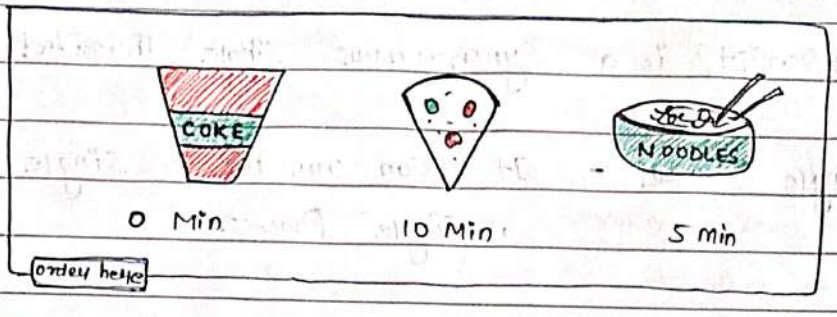
- Js is only capable of Running on a single Piece of thread.

- 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.

__/__/____

Diff b/w Synchronous and Asynchronous operation!



Customeys	items	Synchronous	Asynchronous
(A) →	Coke	0 min	0 min
(B) →	Noodles	5 min	5 min
(C) →	Pizza	15 min	10 min
(D) →	Coke	15 min	0 min
(E) →	Noodles	20 min	5 min

↓

Here, ordey can only be fulfilled, Once the Previous ordey is Fullfiled.

Generally not a good way it is Blocking thread.

Whereas, in Asynchronous exeution.
All ordeys with Coke will get it at 0min the second will get it in 10min all the noodle ordeys will be fullfiled in 5 min. no one has to wait for any othey ordey.

It an non-blocking operation.

__/__/____

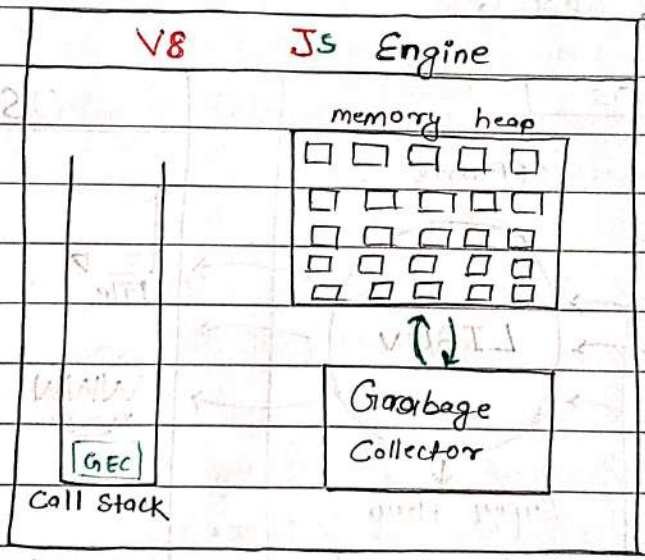
- How Synchronous Code is executed in Java Script ?

-

```
Var a = 407429 ;
Var b = 172410 ;

function multiplyFn (x,y) {
    Const result = a * b ;
    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 into CallStack 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!

__/__/____

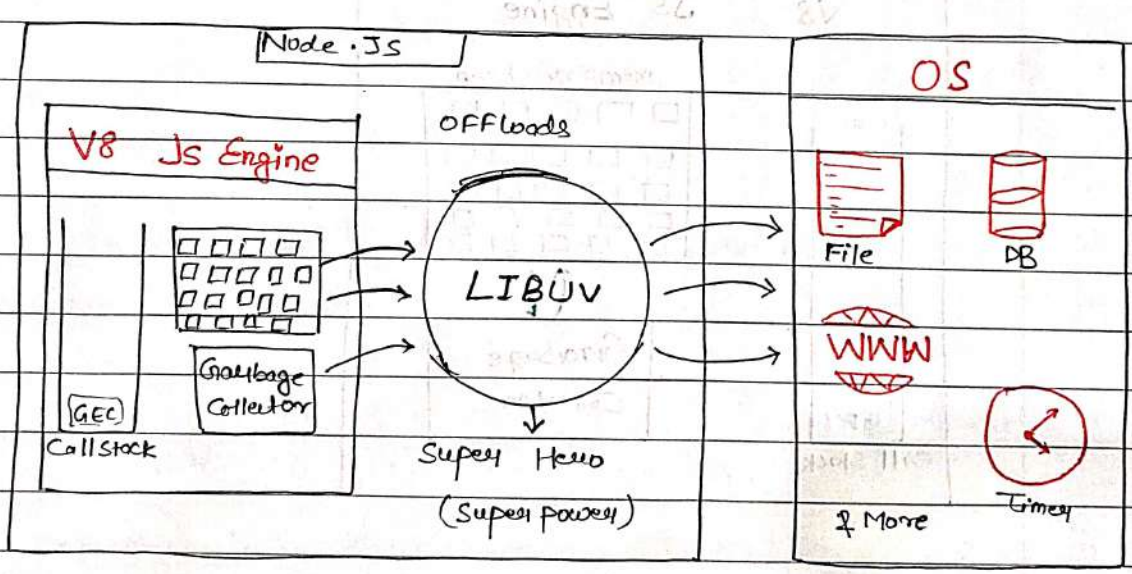
• How Asynchronous Code is Executed in Java Script

-

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

fs.readFile ("./gossip.text", (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, it 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 executed while Containg both Sync and Async operations ?

```
Var a = 427920 ;
Var b = 127221 ;

https.get ("https://api.fbi.com",
  (res) => {
    Console.log (res?.secret);
  });

function MultiplyFn (x,y) {
  Const result = a * b;
    return result;
  }
Var c = MultiplyFn (a,b);
Console.log (c);
```

__/__/____

• How ↓

1. 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.

- ~~For~~ 8. ~~How~~ All async task will be offlooded 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 it 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.

__/__/____

ep-7 / Sync, async SetTimeout Zero

Code :-

async.js

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

Console.log ("Hello world");

    Var a = 1074298;
    Var b = 254670;

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

SetTimeout (() => {
        Console.log ("setTimeout Called after 5 ~~min~~ sec");
        }, 5000);

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

    function multiplyFn (x, y) {
        Const result = a * b;
        return result;
        }

    Var c = multiplyFn (a, b);

    Console.log ("multiplication result is: ", c);
```

__/__/____

Output -

- Hello world
- Multiplication result is = 92472810429!
- File Data : This is the file Data
- 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 Formate - 8 bit.

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

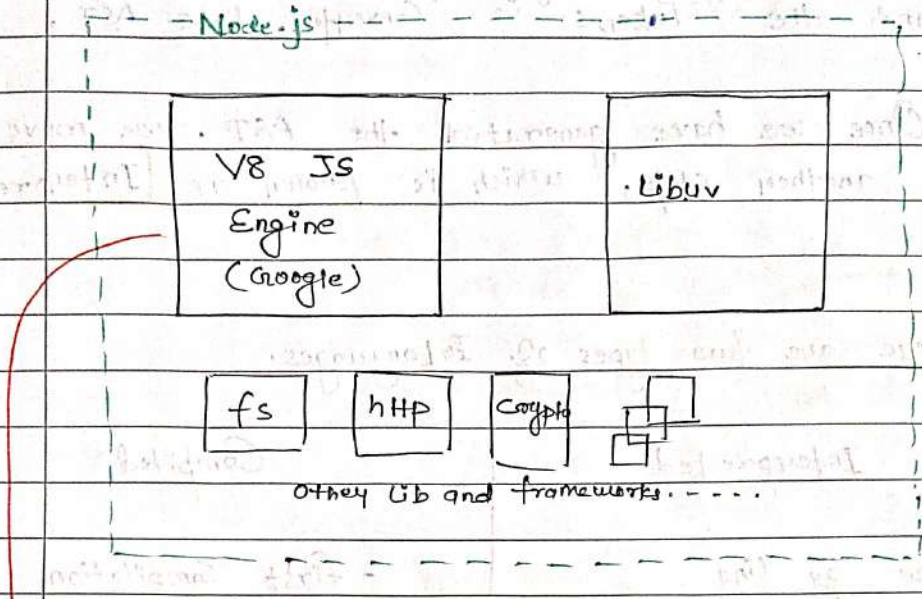
// This callback will only be executed once the Call stack is empty.

```
Console.log ("Call it right now!");
}, 0);
```

- This will be executed when all the Synchronous Code will get executed and call stack become empty.

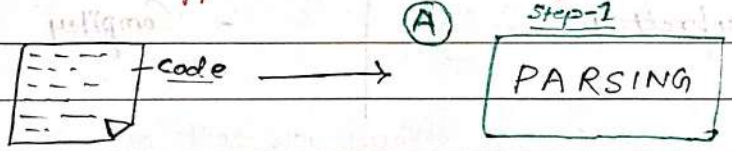
__/__/____

ep-8/ Deep dive into V8 Js engine



We are going to deep inside V8 Js Engine

What happens when we write a Code

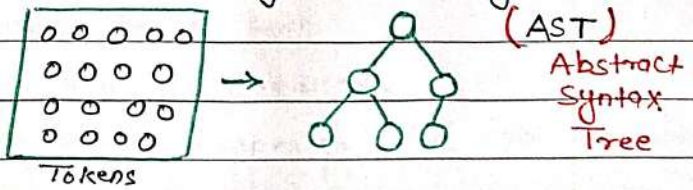


Step-2

Lexical Analysis (Tokenization)

Code → Tokens

Syntax Analysis (Parsing)



- So, whenever you gives 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**.

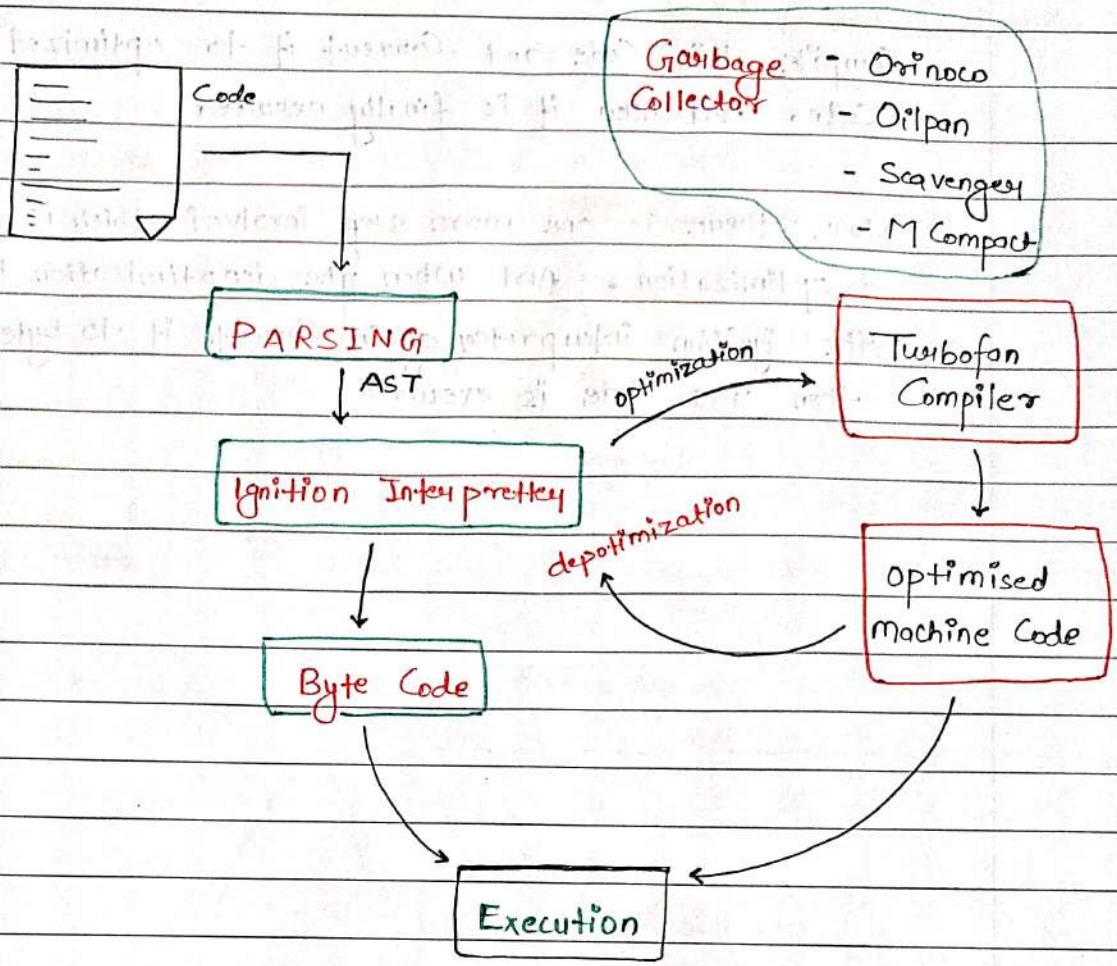
There are two types of Languages.

Interpretted	Compiled
- Line by Line - Fast initial execute - Interpretter	- first Compilation Hc Code → Mc Code - Intially havily but executed fas - Compiler

∴ JavaScript V8 engine uses both interpretted & If was JIT Compilation (Just in time Compilation) JIT Compilor.

V8 Architecture

__/__/____



The Code that you write ex- var A = 10

- A lot of things happings behind the scences are as follows :-
  - The Code is broken down into tokens
  - Then this tokens are Converted into AST
  - The AST goes to the Interpretor.
  - Interpretor 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 ingition interpretor, if it finds the oppurtunity to optimize the Code. It will pass that to turbofan Compiler. The step there is known as optimization. And Now, Turbofan Compiler will

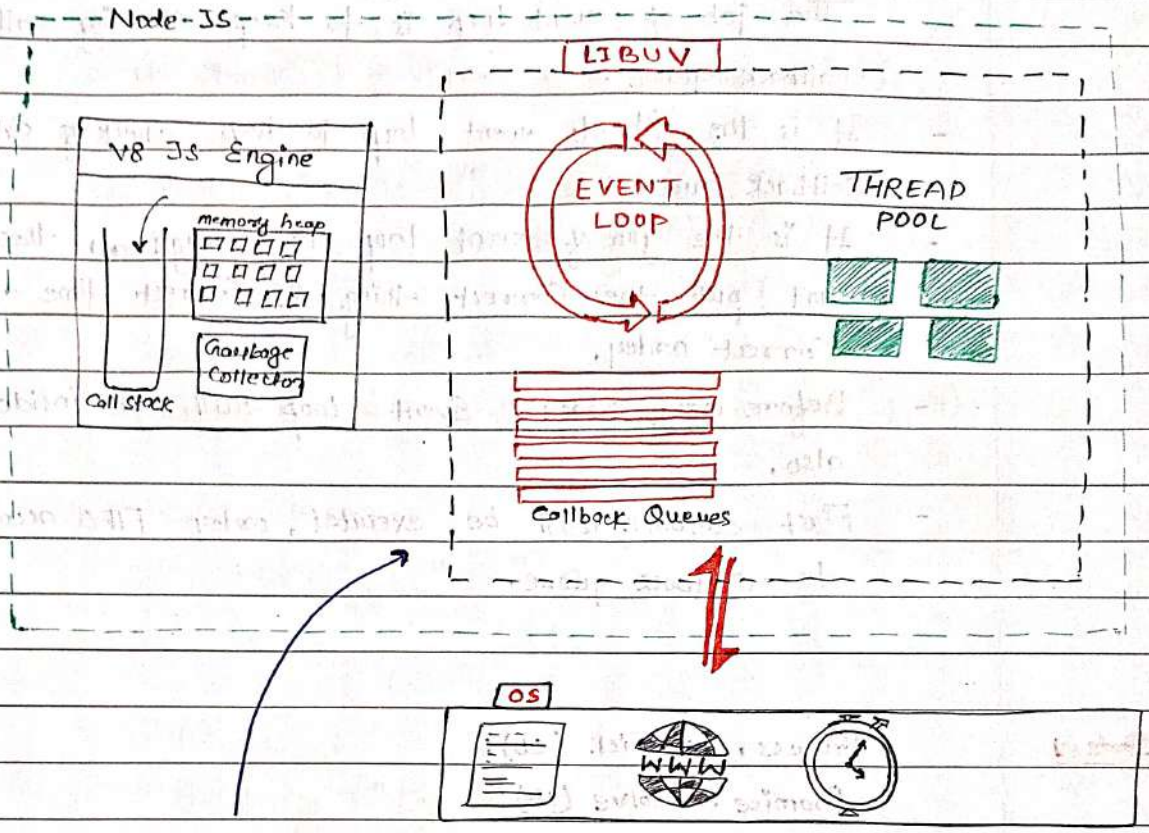
__/__/____

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.

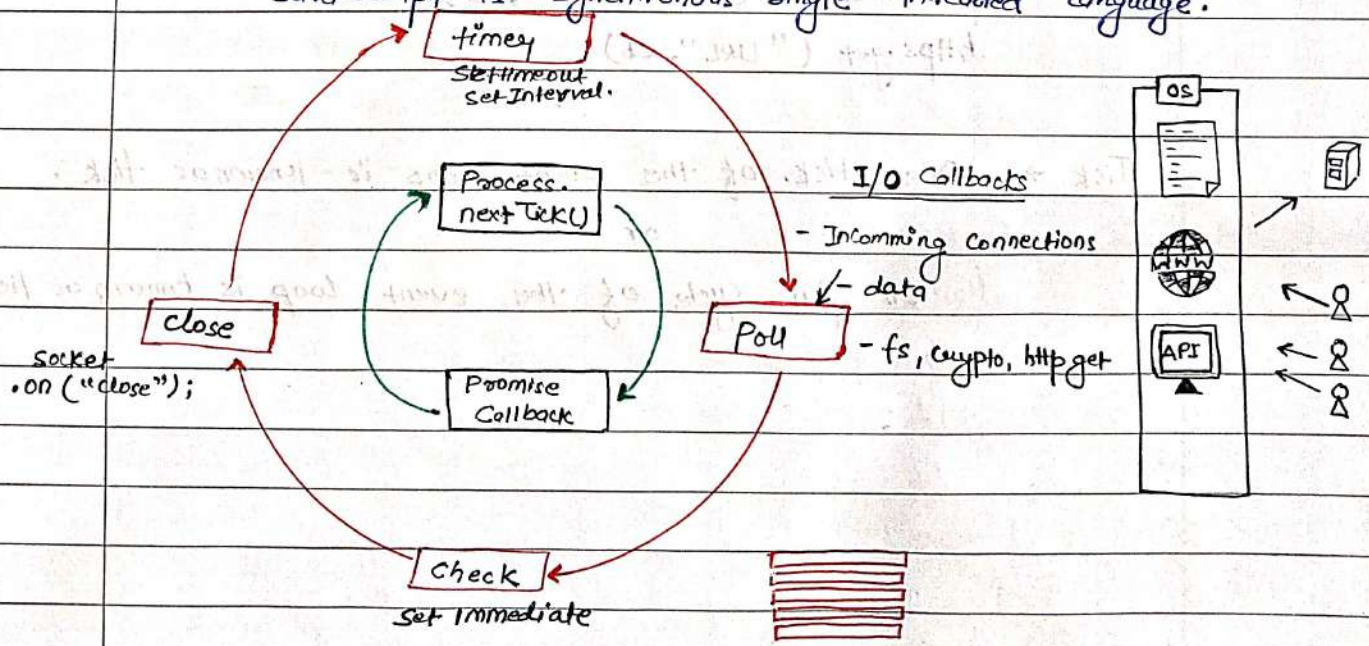
__/__/____

ep-9 / Libuv & Event Loop



Asynchronous I/O (NonBlocking I/O)

" Java Script 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 ek

```
Process.nextTick (cb);
Promise.resolve (cb);
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

On 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");
});

SetTimeout (() => Console.log ("Timey expired"), 0);

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

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

- output

a = 100
Last Line of the file
Timey 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 PrintA () {
    Console.log ("a=", a);
}

PrintA ();
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

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 setImmediate
2nd Timer

__/__/____

Code - 4

```
Const fs = require ("fs");

Set Immediate (() => Console.log ("Set Immediate "));
Set TimeOut (() => Console.log ("Timey expired "), 0);

Promise . resolve (() => Console.log (" Promise "));

fs . read File ("./file .txt", "utf8", () => {
        Console.log (" File Reading CB ");
    });

Process. next Tick (() => {
        Process . next Tick (() => Console.log ("inney next Tick "));
        Console . log (" next Tick ");
    });

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

- Output

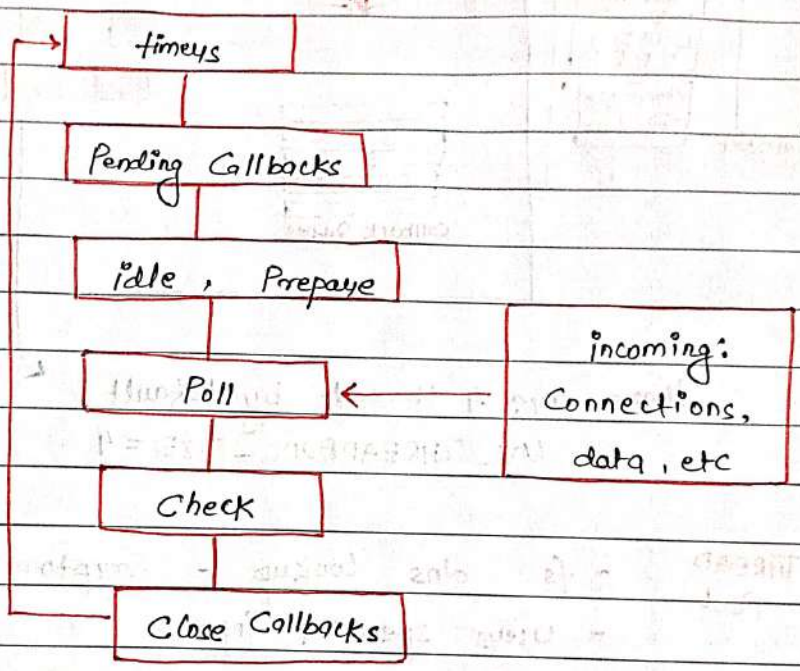
Last line of the Code
next Tick
inney next Tick
Promise
Timey expired
Set Immediate
File reading CB

__/__/____

ep-10 | Thread Pool & Libuv

Event Loop

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



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

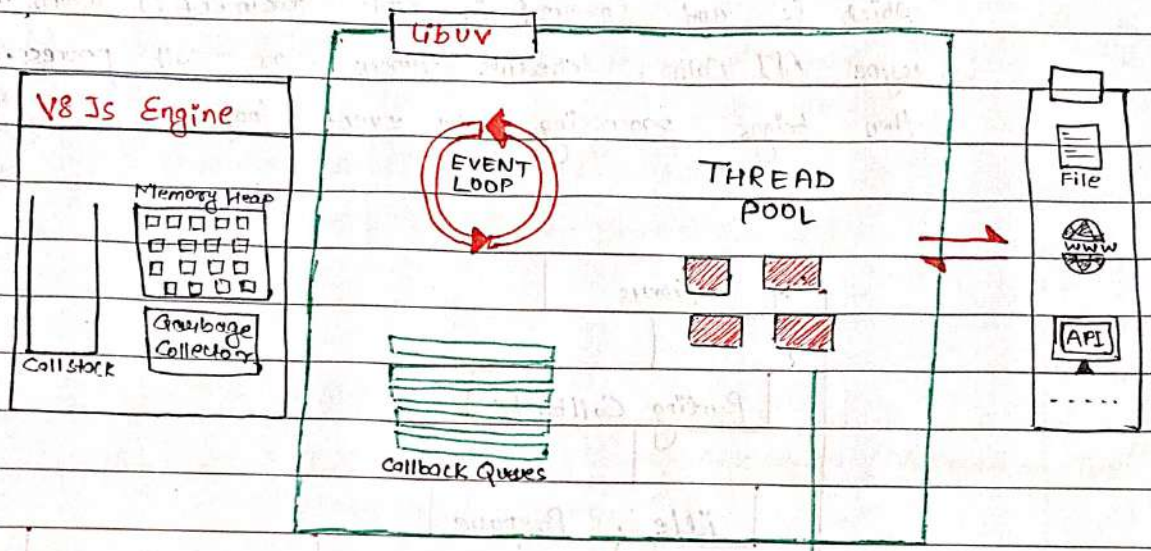
- timers : this phase executes callbacks scheduled by setTimeOut() and setInterval().
- Pending Callbacks: Executes I/O cb deffered to the nxt loop iteration
- idle, prepare : Only used internally.
- Poll : retrive new I/O events, executes I/O related cb (almost all with the exception of close cb, the one scheduled by timers, and setImmediate()); node will block here when appropriate.
- Check : setImmediate() Callbacks are invoked here.
- Close Callbacks - Some close callbacks, eg 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
- Users specified input

Is Node.js is Single threaded or Multi threaded ?

→ If you're giving it an Synchronous task then it is Single threaded but if you're giving some asynchronous task it used UV Threadpool in Libuv library which has 4 thread and act as Multi threaded.

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

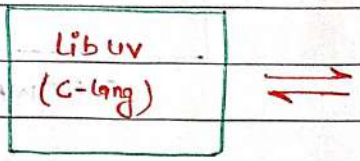
__/__/____

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.



OS
epoll (Linux)
Kqueue (Mac Os)

(Scalable I/O event Notification Mechanism)

Main Teachings -

"Don't Block the main Thread"

→ Synch methods - Heavy JSON objects

→ Complex Regex - Complex Calculations / Loops

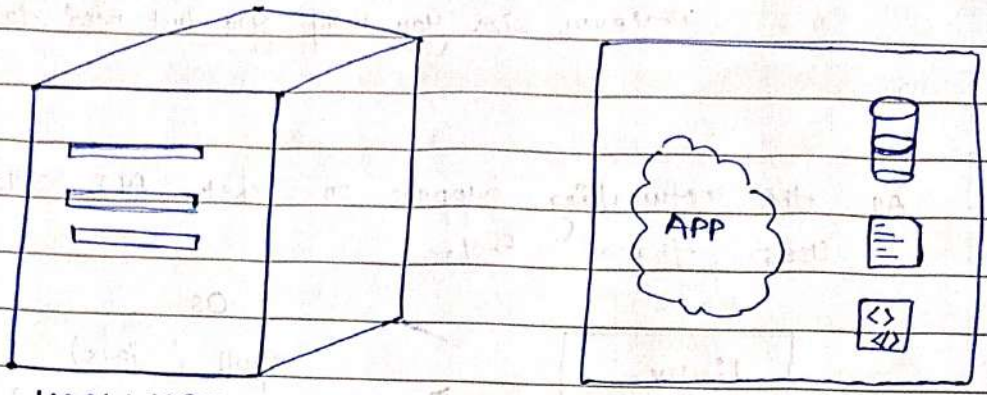
"Data structure is important"

"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.

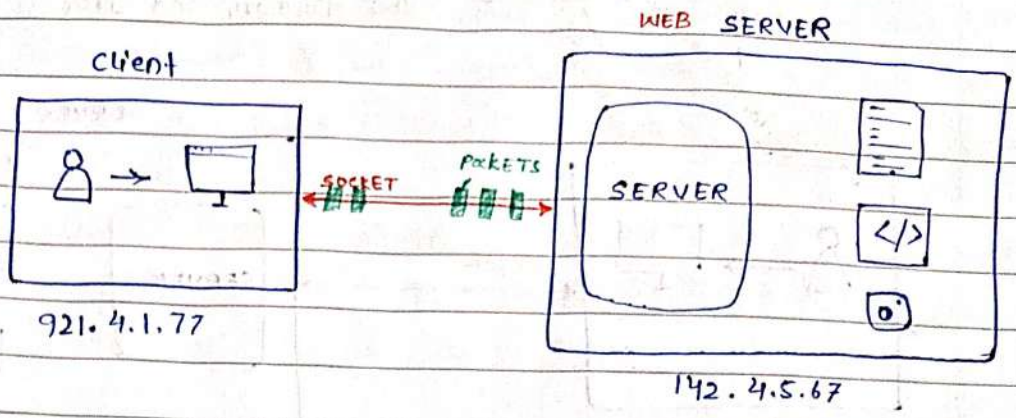
- Limited RAM
- Always up and running
- We use a local Internet Provider, they don't guarantee your IP. But AWS guarantee you a IP.

AWS has a big data Centeres in multiple regions.

You are creating a HTTP server using Node.js
- you are creating a application that can handle users requests.

__/__/____

Client - Server Architecture



TCP/IP

Types of Servers:-

HTTP Server - Hypey Text transfey Protocol
SMT Server - Simple Mail transfey Protocol
FTP Server - File transfey protocol

When we talk about NOde, we talk about HTTP servey in web servey.

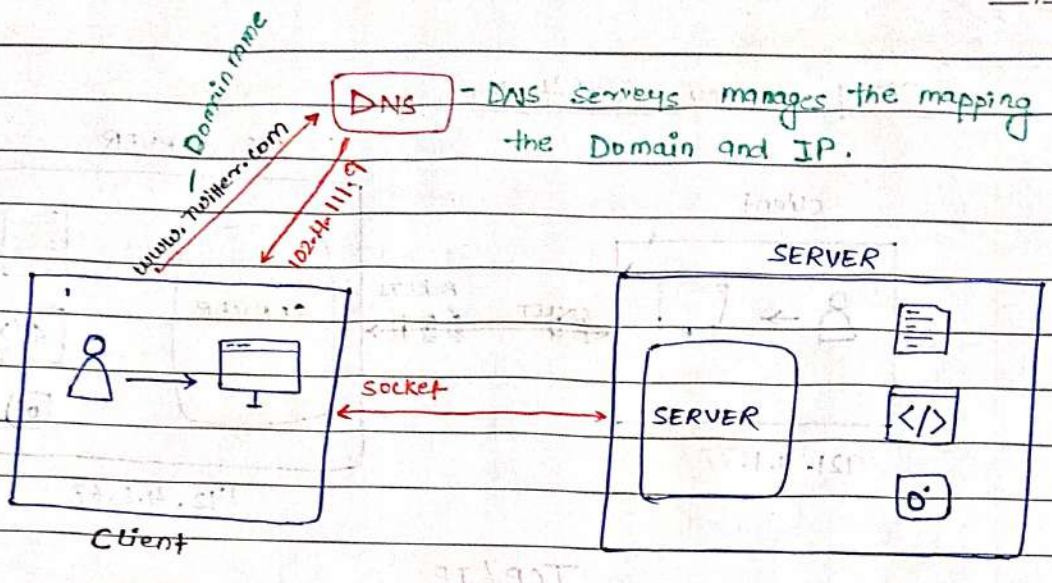
Servey listents to the requests

Pockets - Data are sent in small chunks called Packets.

- We can create multiple of HTTP Servey inside a Servey, but you would need a different Port numbey (localhost:3000)

  - Port Numbey is needed so that othey people can connect to youy applications.

__/__/____



- DNS servers manages the mapping b/w the Domain and IP.

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

Socket	Web Sockets
- In socket, you made a connections, complete your task and close the connection	- When a user makes a connection, it stays for a long time.
- It takes less resources	- It takes more resources

Code

Server.js

```
Const http = require ('node:http');
Const server = http.createserver ();
server.listen (7777);

Const server = http.CreateServer (function (req, res){
res.end ("Hellow world!");
});
```

Express - Node.js web application framework.

__/__/____

ep-12 | Database - SQL & NOSQL

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 an software that interact with end users, applications and the database itself to capture 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. Hierachial DB - IBM IMS
9. Network DB - IDMS
10. Cloud DB - AMAZON RDS

Every database has its own servers.

RDBMS (MySQL, PostgreSQL) - Oracle is mangging MySQL

NOSQL (MongoDB) - "Not only SQL"

MongoDB Comes in 2009 — same time Node.js Comes.

__/__/____

RDBMS (MySQL)	NOSQL (Mongo DB)
- Table, Rows, Columns	- Collection, document, field
- Structure Data	- Un Structured Data
- Fixed Schema	- flexible schema
- SQL	- Mongo (MQL), Neo4j (cypher)
- Tough horizontal scaling	- Easy to scale horizontal & Vertically
- Relationships - Foreign keys + joins	- Nested (Relationship)
- Read heavy apps, transaction workloads	- Real time, Big Data distributed Computing
- Ex - Banking apps.	- Ex - Real time analytics and Soial media.

In NOSQL (Mongo DB)

→ No need for joins

→ No need for Data normalization

Database	⟷	Database
Table	⟷	Collection
Row	⟷	Documents
Columns	⟷	fields

__/__/____

ep-13| Creating a database in Mongodb

How to Connect mongodb 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 succesfully to server");
    Const db = client.db (dbName);
    Const Collection = db.Collection ("users");
    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);

// Count Document

Const countResult = await collection.countDocuments({});
Console.log("Count of document in the users collection =>", countResult);

return "done";
}
```