15/08/2024

|| ॐ श्री गणेशाय नमः ||

Namaste Node.js

⇒ Make your own notes.
⇒ Proctice along with videos
⇒ Make a schedule → I will watch 1 episode a day atleast.
⇒ Let akshay, complete his lecture, code all those things which he coded

Episode-01 Introduction to Node.js

"Any application that can be written in Javascript will eventually be written in Javascript."
– Jeff Atwood, 2007
Founder, Stackoverflow

What is Node.js?

Node.js is a Javascript runtime built on Chrome's V8 Javascript engine.

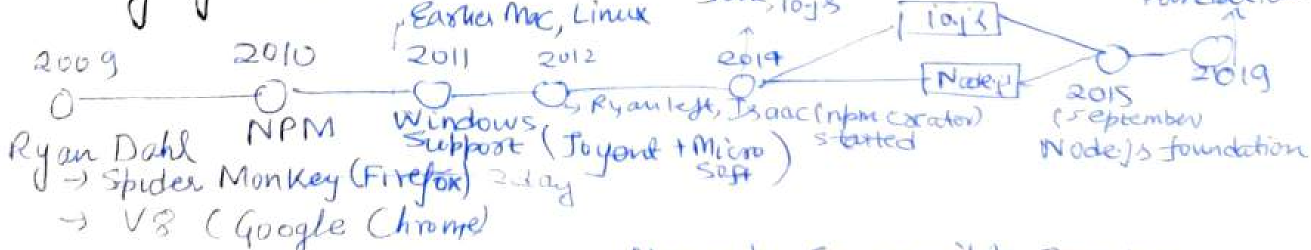
→ maintained by openJS foundation.

⇒ Node.js is a cross-platform, opensource Javascript runtime environment that can run on windows, Linux, Unix, macos and more. Node.js runs on the V8 Javascript engine, and executes Javascript outside a web browser.

→ Event-driven architecture
→ Capable of asynchronous I/O –or– Non-blocking I/O

2009 → Ryan Dahl developed Node.js

History of Node JS


2009
Ryan Dahl
→ Spider Monkey (Firefox) 2 day
→ V8 (Google Chrome)
2010
NPM
Earlier Mac, Linux
2011
Windows Support (Joyent + Microsoft)
2012
Ryan left, Isaac (npm creator) started
2014
Fedor (created fork, io.js (because slower)
io.js
Node.js
2015
(September
Node.js foundation
2019
OpenJS Foundation

wherever there is Javascript, there is Javascript engine.

Node.js is powered ignited by Google's V8 engine.

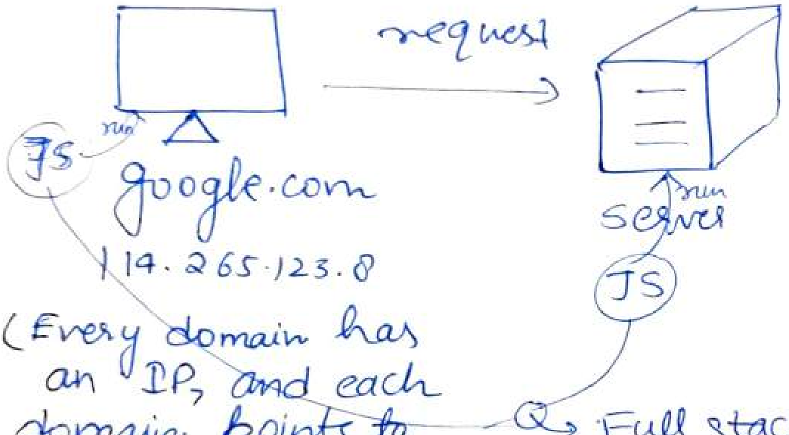
Earlier name was web.js → creating web server
renamed ↓
node.js → It can create even more.
↓
Non-Blocking Server
→ It can handle multiple server with lesser number of threads.

Apache
HTTP
Server
'Blocking'

Episode-02 | JS on Server

Node.js came with a philosphy that it can run outside the brower, primarily on server.

Server is nothing but a remote computer. One cpu that is recieving your request.



(Every domain has an IP, and each domain points to a server)

Now you just have to learn one language for frontend and backend
earlier for backend → Java, C++, python.

Node.js is C++ Code??

JS Engine - V8(Google) → C++ program?? Yes (72% C++ 25% Javascript)

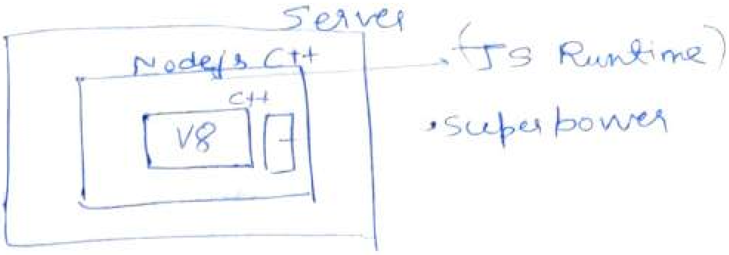
"V8 can be embedded into any C++ application"

V8 → execute JS code

→ JS → $\underset{C++}{\boxed{V8}}$ → machine level code.

→ Node.js is created using Javascript language which can be run in the desktop or create application. Node.js is also written in C++ because when the web server needs access to internal system functionality such as networking.

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



Ecma Script: is a standard for scripting languages, including JavaScript JScript and ~~Type~~ Action Script.

- standard/Rules

JS engines follow these standards.

V8 → Google
SpiderMonkey - Firefox
Chakra - Microsoft
Js Code - Safari

V8 cannot go beyond Ecma script.
So, V8 + Superpower = Node.js
↳API on server

V8 can't connect to database., can't call HTTP, can't go to file system, can't fetch images.
You need those super powers, these superpowers came in form of APIs.

⇒ Node.js is a C++ application with V8 embedded into it. along with some superpowers in form of APIs called Javascript runtime.

Because it also have a lot of Javascrip apis.

Node.js github
62% Javascript
21.9% C++

V8 is a C++ Code??

Javascript (easy to write)
C++

High Level language

Machine code

Assembly code

Binary code

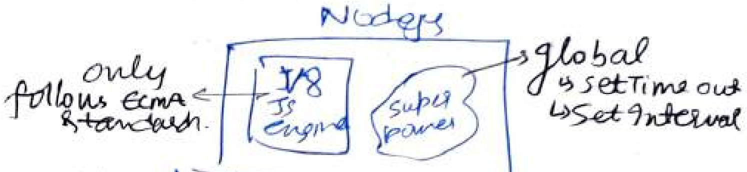
Episode-03 Let's write Code.

Where to write Nodejs code?

Node REPL → Read, Evaluate, Print, Loop

node write this into cmd.

Node.js is javascript runtime environment built on Chrome's V8 Javascript engine.



Javascript is synchronous single threded language.

'window' is a global object given by Browser not by Chrome V8 engine.

In node.js the global object is global.

```
Console.log(global);
Console.log(this);   // op → {}
```

in frontend it will print window.
↳ with self,
frames.

But all browser supports globalThis

Episode-04 module.export & require

Modules protects their variables and function from leaking.

By default the modules are protected.

```
module.exports = calculatesum;

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

```
module.exports = {
        x: x,
    Calculatesum : calculateSum,
  }
```

```
require("./xyz.js");
const obj= require ("./sum.js");
 obj.calculatesum(a,b);
```

//Destructuring

```
Calculate {x, calculatesum} = require("./sum.js");
```

same as→ `module.exports = {x, calculatesum};`

CommonJS Modules (cjs)

→ modeule.exports
require()

→ by default used in Nodejs.

→ Synchronous

→ non-strict mode.

Strict mode: It provide better error checking and enforces stricter coding rules, leading to fewer bugs and better code quality.

Non-Strictmode: They may seem more flexible at first, it can lead to unexpected behavior and Security vulnerability

ES Modules (mjs)

In package.json

```
{
 "type" : "module"
}
```

```
export function - - {

   },
```

```
import {calculatesum}
        from "./sum.js";
```

→ import
export

By default used in React, Angular.

→ Async option is there.

→ strict mode.

console.log (module.exports); {}

module.exports is a empty object.

You can also write

```
module.exports.x = x;
module.exports.calculateSum = CalculateSum;
```

Always wrap inside a object and import in a {}, this a a good practice

How do you import data.json file?

```
Const data = require ("./data.json");
```

JSON.stringify(data) ⇒ ~~json~~ → object
↓
json

```
Const util = require ("node:util");
```

Episode-05 | Diving into the NodeJS github repo

What is IIFE?

When we do require('./path')

All the code of this module is wrapped inside a function (IIFE)

IIFE → Immediately invoked function expression

```
(function() {

})();
```

↑
this bracket is important.

→ Before execution, all the requires are converted into IIFE or wrapped inside IIFE.

⇒ It keeps variables & functions private and safe. The code inside IIFE will not interfere with file JS code.

⇒ Whenever you create a module, all the code that you write in a module is wrapped inside a function & then executed.

⇒ That's why you cannot access the data of the Module with exports

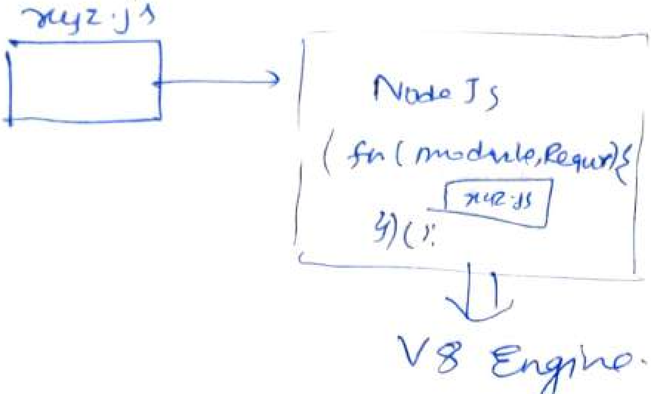
⇒ Whenever you create a module and Require it into your file, what happens is node js takes the code from that file or module and wraps it into a function & then execute it.

⇒ 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 = {}.
})();
```

NodeJs 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 knows how to execute IIFE..

→ This was all about ~~two~~ the IIFE & its working

5 Step Mechanism of Require ('./path')

1) Resolving the Module.

→ ./local path

→ .JSON

→ node:module

It sees from where the data is coming and accordingly it resolves the modules

2) Loading the Module
↳ file content is loaded according to file type.

3) Wraps inside IIFE

4) Evaluation: code is executed & returns module.exports

5) Caching: the code of require will run only once.

Episode-06 libuv & async IO

Node.js has an event driven architecture capable of asynchronous I/O

Javascript is synchronous single threaded language.

Javascript is synchronous but with the help of nodejs it becomes asynchronous.

Synchronous

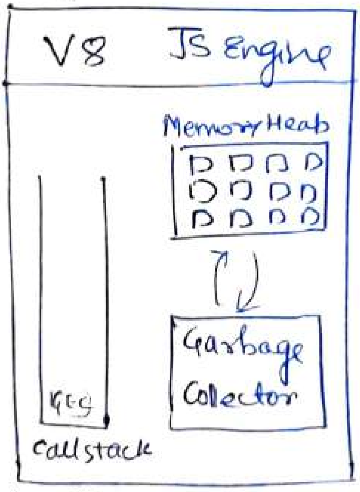
```
var a = 1078698;
var b = 20986;

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

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

Asynchronous

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

fs.readFile("./gossip.txt", "utf8", (data)=>{
   console.log("File data", data);
});
setTimeout(()=> {
     console.log("wait here for 5 seconds");
}, 5000);
```

How synchronous code is executed in javascript?

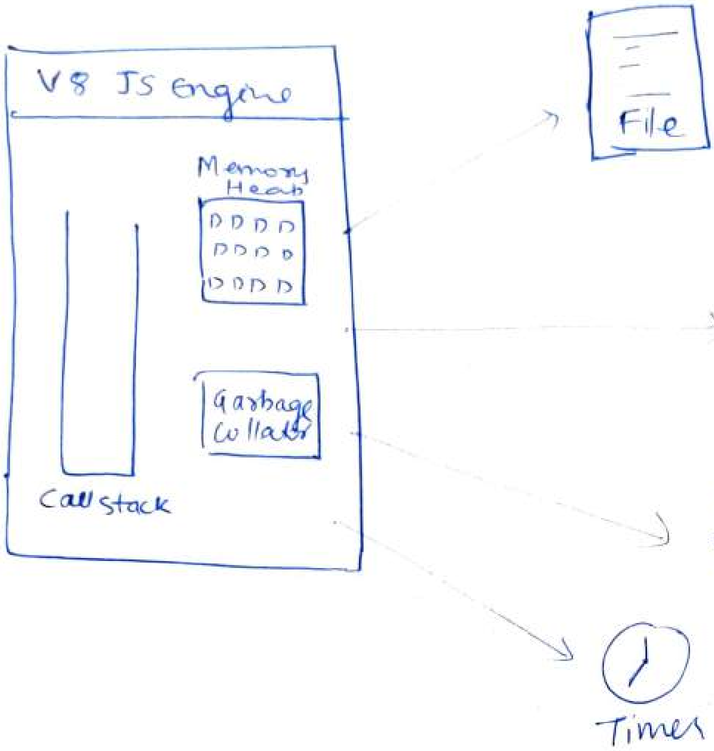
"Time, Tide & Javascript waits for none."

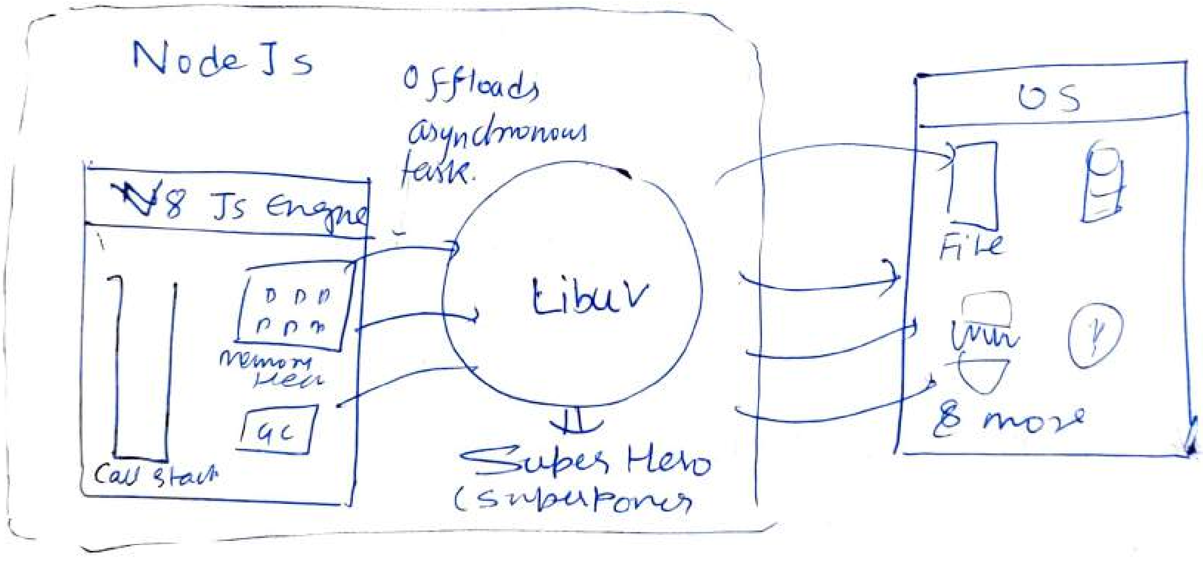


Single Thread

GEC → Global execution context.

How asynchronous code is Run?



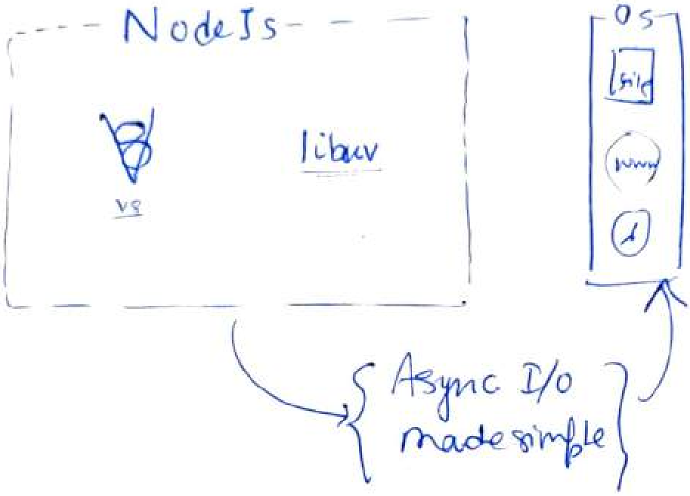


libuv → libuv is a multi-platform C library that provides support for asynchronous I/O based event loops.

Asynchronous I/o made simple.

libuv is Ginie, superhero.

Libuv is the hero, due to which Nodejs is asynchronous



Whenever V8 engine sees api call, file read operation, settimeout, it offloads this to libuv. and then libuv manages it

"Time tides and javascribt, javascribt engine waits for none."

⇒ Nodejs is Asynchronous by V8 engine is synchronous.

⇒ Nodejs cando Async I/o.

Non-Blocking I/o, → because it is not blocking our main thread.

So, Even with a single thread, it can do somany async operation together.

Episode-07 sync, async, setTimeoutZero.code

async.js

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

console.log("Hello World");

var a = 1078698;
var b = 20986;

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

setTimeOut(() => { console.log("set timeout 5 sec")}, 5000);

fs.readFile("./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);
```

O/p

```
Hello world
Multiplication result is 222607
Filedata: This is the file data
Feteted Data Succesfully
SetTimeout called after
5 Second.
```

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.

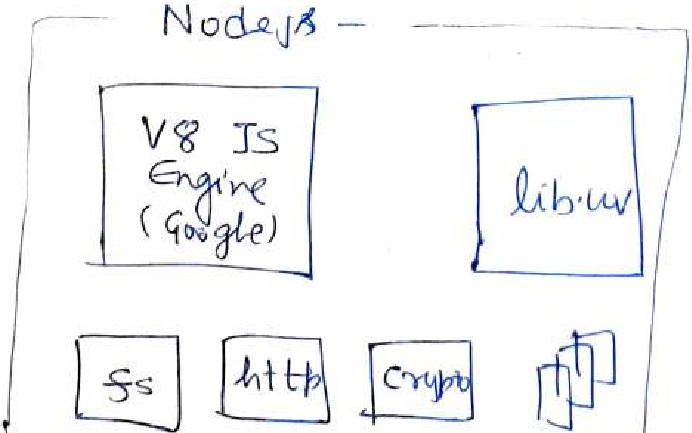
⇒ Synchronous function will block the main thread, don't use it.

```
// This callback will only be pushed to callstack in V8
                                  once the call stack is empty.
setTimeout( () => {
          console.log("call me right now");
          }, 0);
```

⇒ This will be executed when all the synchrones code will get ~~get~~ executed and when call stack become empty.

⇒ Trust issues with setTimeout
TNC → only if the main thread is empty.

Episode-08 | Deep dive into V8 Js Engine



Code

⇓

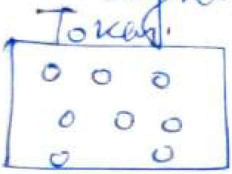
V8

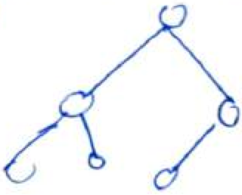
A) PARSING

1) Lexical Analysis (Tokonization)

Code → Tokens

2) Syntax Analysis (Parsing)





AST { Abstract Syntax Tree }

asterplorer.net

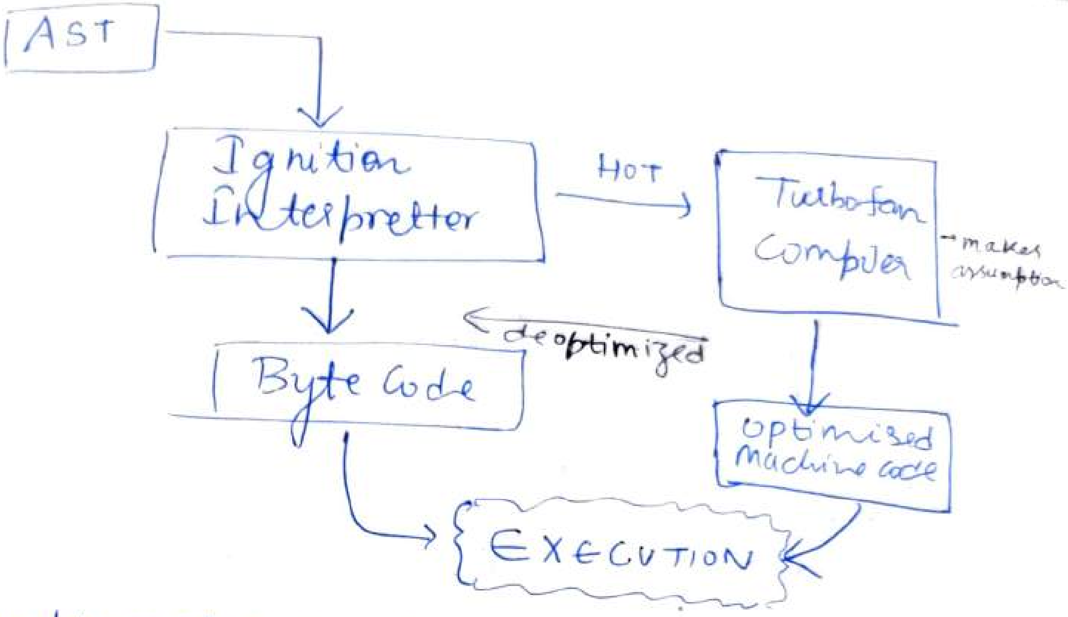
AST → INTERPRETER

2 types of languages

Interpretted	Compiled
→ line by line	→ first compilation HL code → Machine code
→ fast inital execution	→ Intially havily but executed fast
→ Interpretter	→ Compiler.

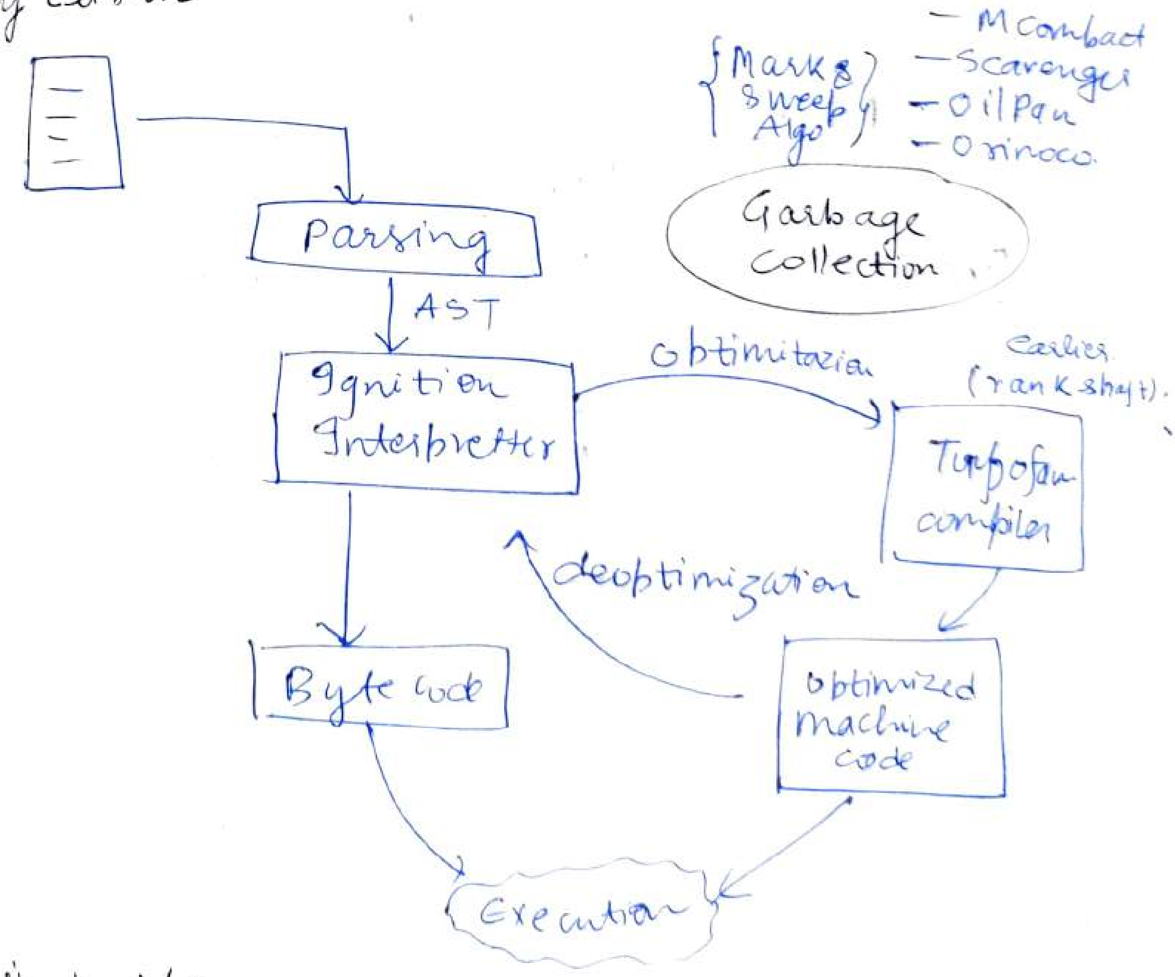
→ Javascript V8 engine uses both interpretor.
It uses
JIT compilation (Just-in-time compilation)

JIT compilation



HOT → some piece of code which is used a lot and there is some scope of optimization.

→ Inline caching
→ copy Elision

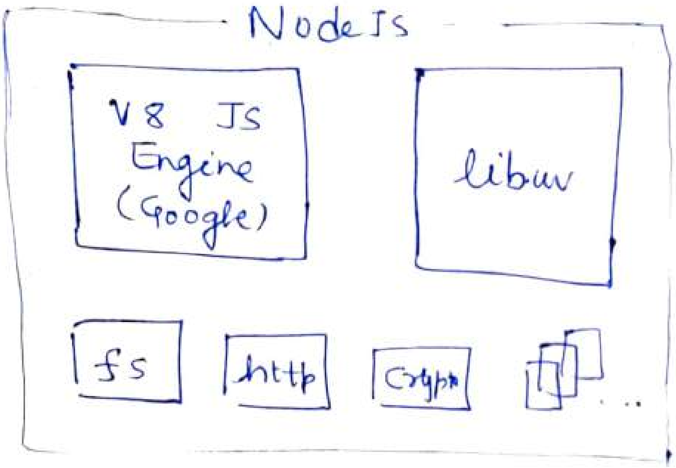


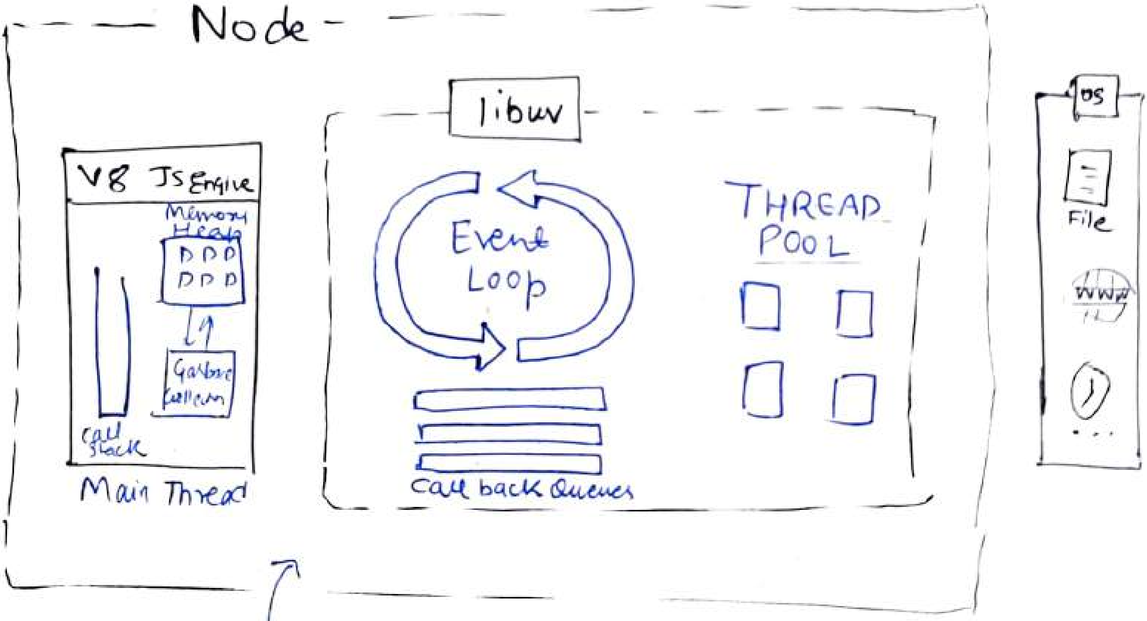
This is V8 architecture.

Just-in time compilation:

JIT compilation is a compilation during execution of a program rather than before execution.

Episode-09 | libuv & Event Loop

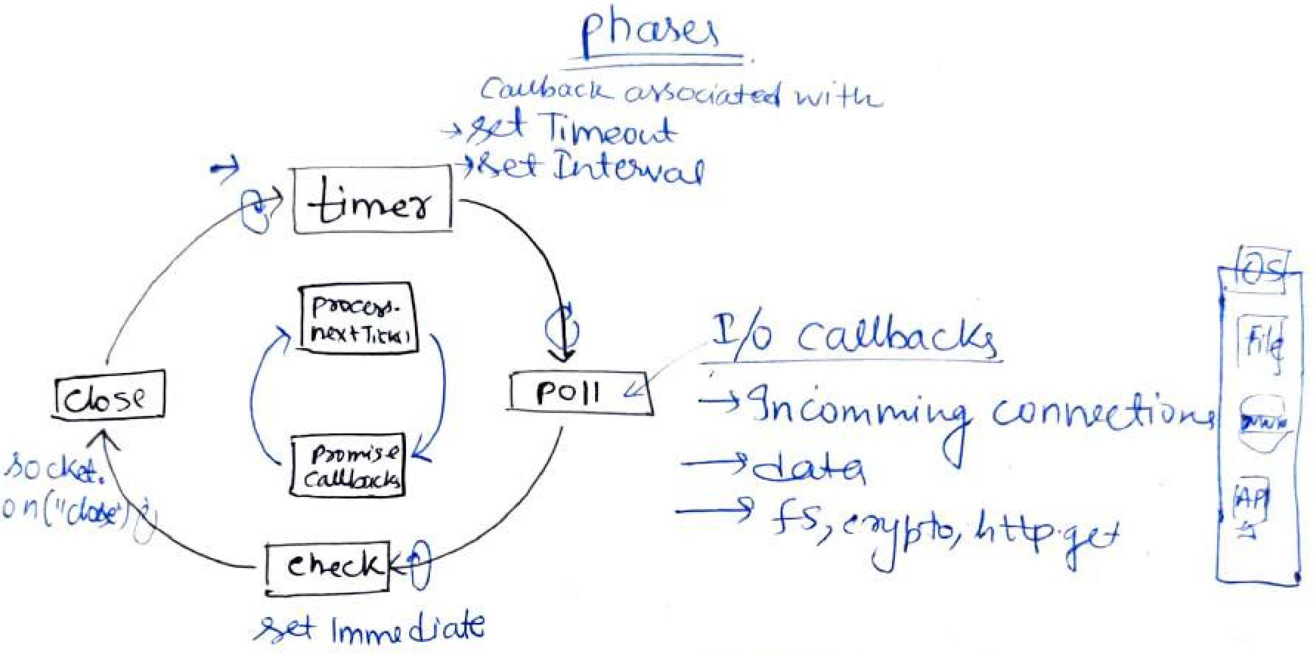




Asynchronous I/O (Non Blocking I/O)

"Javascript is synchronous single threaded language"

Event Loop



call back queues

⇒ Untill V8 Js engine callstack is not empty, libuv will push all the completed ~~fast~~ 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 manage all these things and put the correct thing in correct time and Correct order.

→ ~~E~~ Before every phase, ~~Every~~ Event loop will run inside cycle also.

→ First callback will be executed, order FIFO according to callback queue.

Code-1

```
Const a= 100;
setImmediate(()=>console.log("setImmediate"));

fs.readFile("./file.txt", "utf8', ()=>{
                console.log("File Reading callback");
    });

setTimeout(()=>console.log("Timer expired"),0);
function printA(){
    console.log("a=",a);

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

Console

```
a = 100
Last line of the file
Timer expired
setImmediate
FileReading Callback.
```

Code-2

```
Const a= 100;
setImmediate(()=> console.log("setImmediate"));
~~Promise.resolve(()=> console.log "Promise"));~~
  Promise.resolve("Promise").then(console.log());
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"I
```

Console:

```
a =100
Last line of the file
process.next Tick
Promise
Timer expired
setImmediate
File Reading CB
```

⇒ Event loop waits at poll phase, when the is idle.

In browsers, the event loop Keep on Keeps on running while in node.js Event loop rests/waits at poll phase when it has nothing to do.

Code-3

```
setImmediate(() => console.log("setImmediate"));
set Timeout (() => 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 set immediate"));

        console.log("File Reading CB");
});
Process.nextTick(() => console.log("nextTick"));
Console.log("Last Line of the file");
```

Console:

```
Last line of the file
nextTick
Promise
Timer Expired
Set Immediate
File Reading CB
2nd next Tick
2nd set immediate
2nd Timer.
```

Code.4

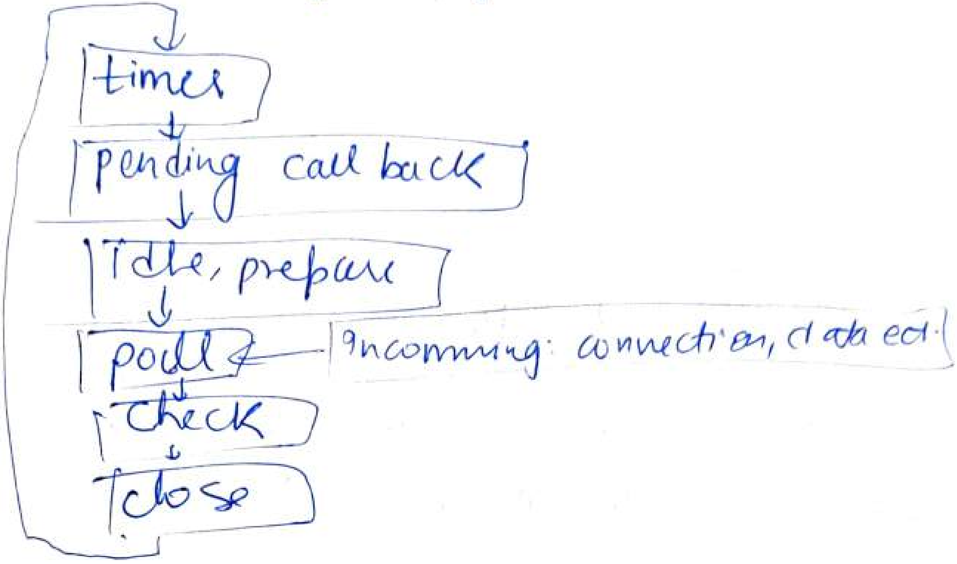
```
const fs = require("fs");
Set Immediate (() => Console.log ("set Immediate"));
set Timeout(() => console.log("Timer expired");0);

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

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

Process.nextTick(() => {
        Process.nextTick(() => console.log("inner nextTick"));
        Console.log("nextTick");
    });

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

Console:

```
Last line of the file
nextTick
inner nextTick
Promise
Timer expired
~~inner nextTick~~
Set Immediate
File Reading CB
```

Episode-10 Thread Pool in libuv

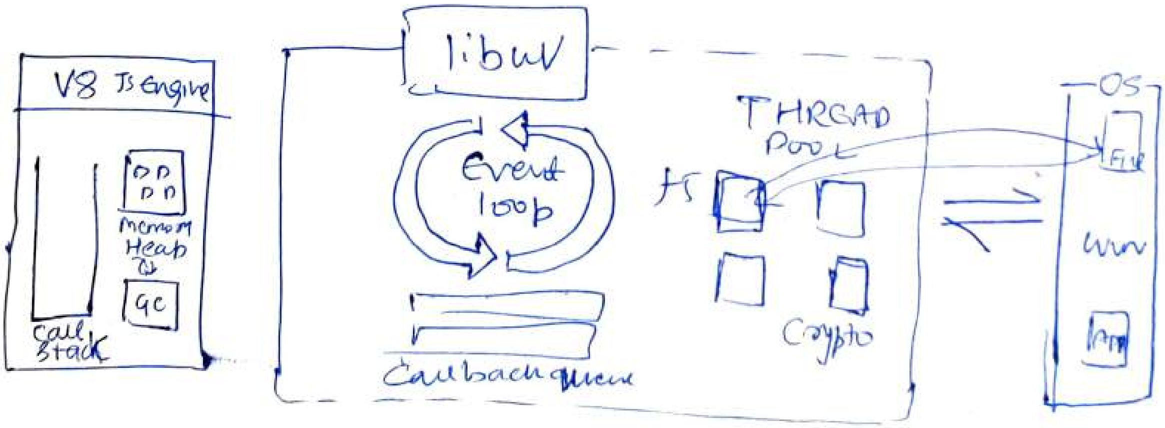
Tick→ one full cycle of event loop.



Thread Pool

{process.UV_THREADPOOL_SIZE}

Is NodeJs singlethreaded or multithreaded?



There are 4 threads by default

UV_THREADPOOL_SIZE = 4

THREAD POOL { - fs - dns.lookup - crypto
- user specified input

Q Is NodeJs single-threaded or multithreaded?

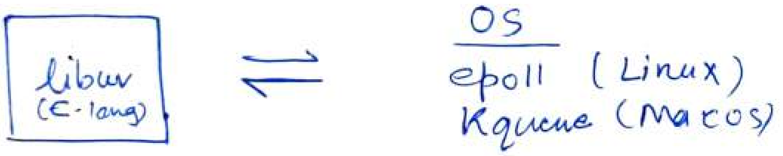
Ans If your are giving synchronous task then, it is single threaded, but if you are giving some asynchronous task it used UV Threadpool in libuv library which has 4 threads.

Answer is it depends.

⇒ You can change the number of threads by:

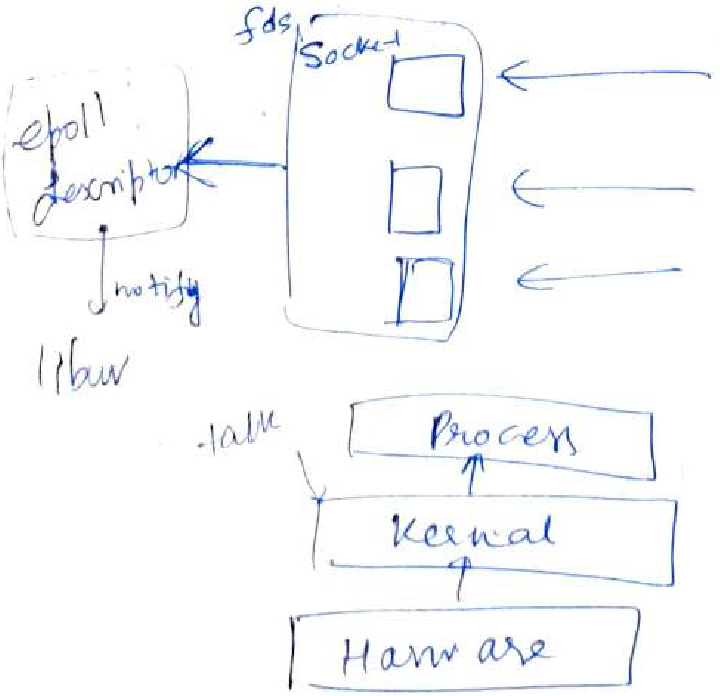
```
process.env.UV_THREADPOOL_SIZE = 2;
```

⇒ All the networking happens on socket API calls does not uses thread pool.



[scalable I/O event Notification mechanism)

1 thread per connection is not a good idea.



That's why it's called event driven architecture.

epoll behind the seen uses Red black tree

Main teachings

"Don't Block the main thread".

→ sync methods - Heavy JSON objects

- Complex Regex - Complex calculations/loops

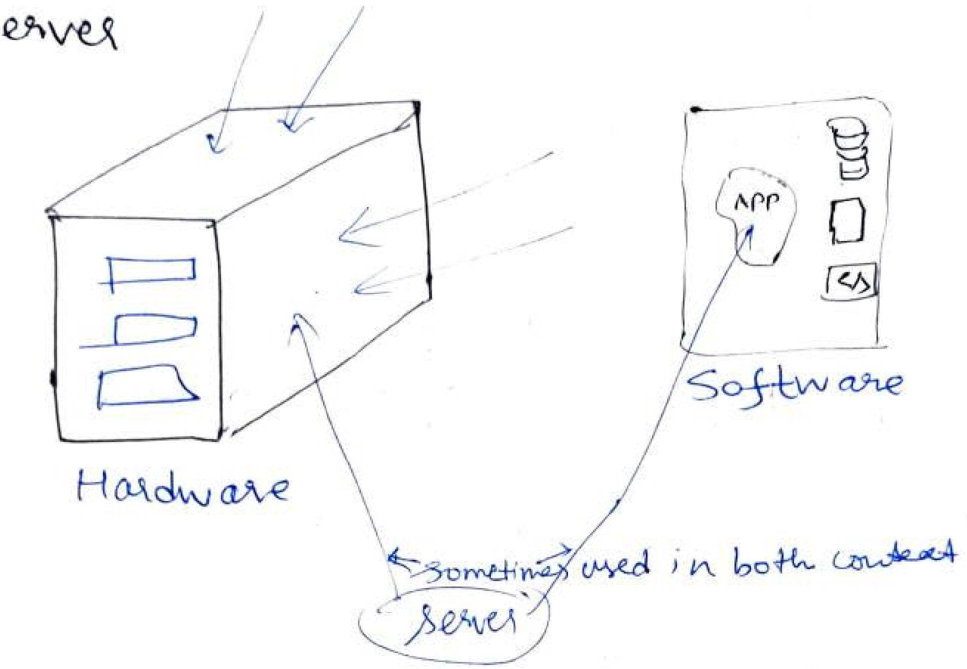
"Data Structures is important".

" Naming is very important".

"There's a lot to learn".

Episode-11 Creating a Server

Server



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

ec2 instance → resource of AWS you suscribed

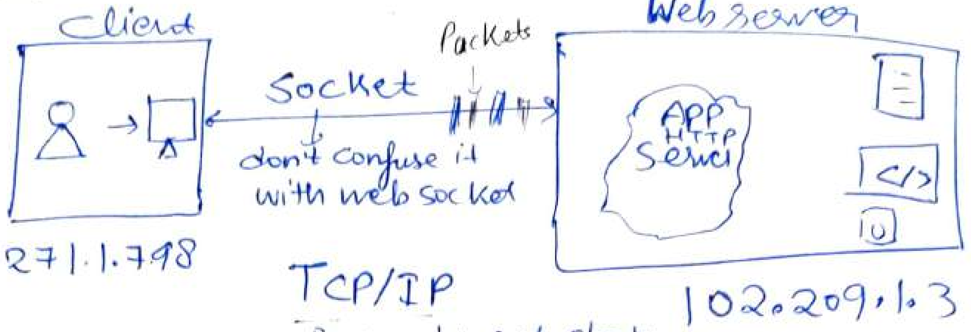
- → limited RAM
- → Can't maintain Always up and running
- → We have a local Internet provider, they don't guarantee your IP. But AWS gdarantees you a ~~A~~ IP.

AWS has a big Data Centers in multible regions.

You are creating a 'HTTP server using Nodejs

⇒ You are creating a application that can handle user requests.

Client - Server Architecture



TCP/IP

Protocol : set of rules defined to communicate ~~using~~ between computers.

HTTP server (Hyper Text transfer protocol)
SMT Server (simple mail transfer protocol)
FTP Server (File transfer protocol)

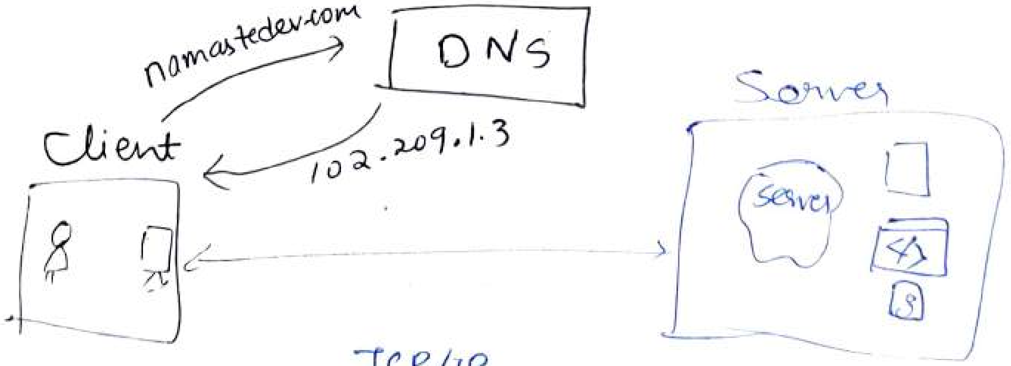
When we talk about Node, we talk about HTTP server in web server.

Server listens to the requests.

Data are sent in small chunks called 'Packets'

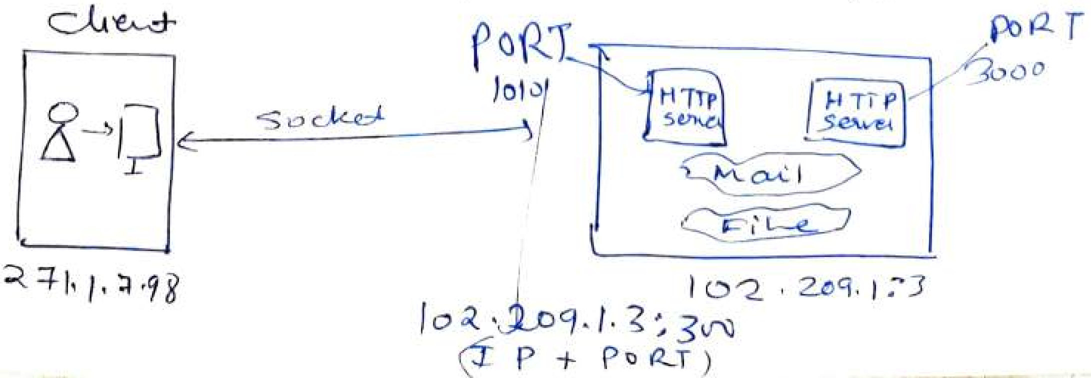
It's like stream of Data.

Stream & Buffers



TCP/IP

Can I create multiple HTTP server?? → Yes



Port number is needed so that other people can connect to you.

Some port no. are reserved like 80

Domain Name

⇅

IP + PORT + PATH

⇅

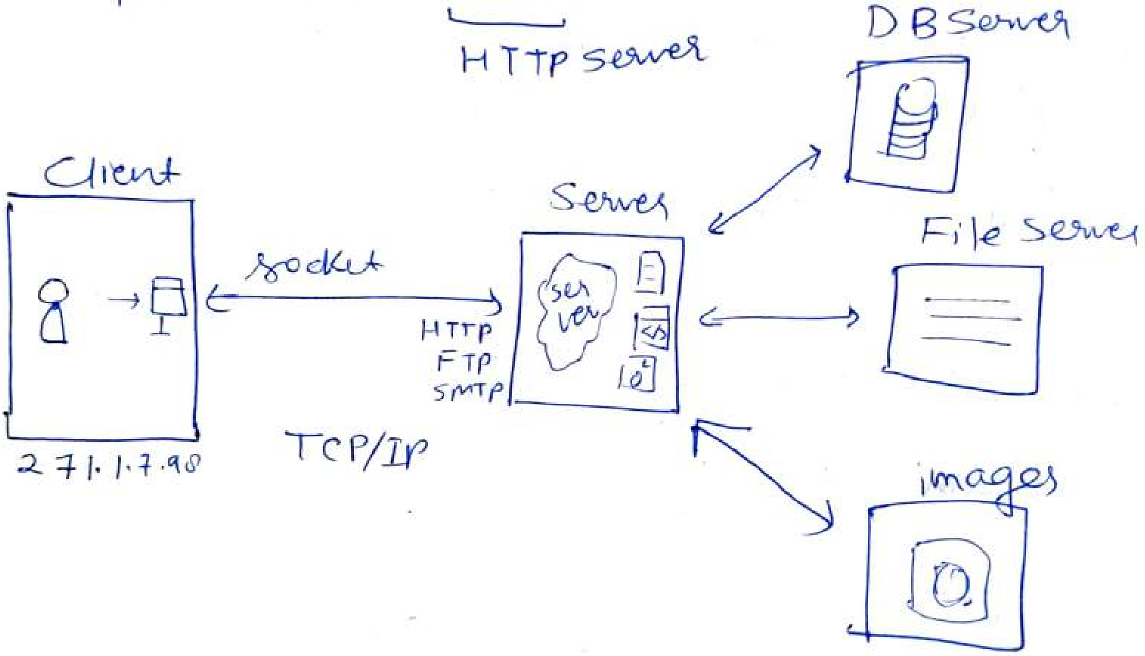
API

namastedev.com

123.4.5.6 : 3000

/api/getUserInfo

HTTP server



A server can talk to another server also.

Socket vs Web Sockets

- → In socket, you made a connection, complete your task and close the connection.
- → It takes less resources

- → When a user makes a connection, it stays for a long time.
- → It takes more resources

Server.js

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

Const ~~htt~~ server = http.createserver();

Server.listen(7777);
```

Server.js

```
const http = require("node:http");
const server = http.createServer(function(req, res){
            res.end("Hello World!");
     });
server.listen(7777);
```

```
if(req.url=== "/getSecretData"){
    res.end("There is no secretdata");
```

It is a tough and tedious way especially in routing. We have ~~may~~ many issues in Creating servers using http modules. So we use Express

Express - Node.js web application framework

Episode-12 | Databases - SQL & NoSQL

Q What is database? What is DBMS?

Ans A database is an organized collection of data, or a type of data store based on the use of a database management system.

DBMS ⇒

DBMS is a software that interacts with end users, applications and the database itself to capture and analyze the data.

Types of Databases

1. Relational DB – MySQL, PostgresSQL
2. NoSQL DB – MongDB
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. Hierarchial DB – IBM IMS
9. Network DB – IDMS
10. Cloud DB – AmaZon RDS

Every database has its own server.

RDBMS (MySQL, PostgreSQL), oracle)

oracle is managing MySQL

NoSQL (MongoDB) (Not-only SQL)

* Document DB * Keyvalue DB * Graph DB

* Wide-Column DB * Multimodel

MongoDB is Document DB

MongoDB comes in 2009 (2000s)

↓

same time Node.js come

In NOSQL (MongoDB)

→ No needs for joins

→ No need for data normalization

Cluster (deploy a cluster)

Database ⟷ database

Table ⟷ collections

Row ⟷ document

Column ⟷ fields

RDBMS (My SQL)	No SQL (MongoDB)
- Table, Rows, Columns	- Collection, document; field
- Structure Data	- Unstructured / semistructure data
- Fixed schema	- flexible schema
- SQL	- Mongo (MQL), Neo4J (Cypher
- Tough horzontal scaling	- Easy to scale horizontally & vertically
- Relationships - foreign keys + joins	- Nested [Relationships]
- Read heavy app, transaction workloads.	- Real Time, Big data, distribute computing
- Ex. Banking apps	- Ex. Real Time analytics Social media.

19/08/2024

Episode-13| Creating a database & 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("User");
        // Write code for CRUD operations here.
        return "done.";
}

  main()
        .then (console.log)
        .catch (console.error)
        .finally(() => client.close());


//Read
Const findResult = await collection.find({}).toArray();
console.log('Found documents =>', findResult);
```

```
// insert a document
Const data={
      firstname: "Deepika",
       lastname: "Padukone",
       City: "Mumbai",
      phoneNumber: "9875432100";

  };

Const insertResult = await collection.insertMany([data]);
Console.log ('Inserted documents=>', insertResult);
```

But while making project, we will not use this 'mongodB' package.

We will use Mongoose Package.
Mongoose is a MongoDB object modelling tool designed to work in an asynchronous environment.
Mongoose supports Node.j's and Deno (alpha).

"See you all in Season-02"

19/08/2024
19:42:00