NAMASTE NODE JS

SEASON-1

EPISODE-01

↳ Intro to Node.js

Node.Js

↳ It is a cross platform, open source Javascript runtime Environment built on Chorome's V8 Engine

- Node.js was built by Ryan Dahl in 2009
- Node.js is maintained by the OpenJs Foundation.
- Node.js execs JS code outside the web Browser

→ JS is a language at the core of the web and browsers, but with Node.js, we could run it Everywhere. That's why node.js is often associated with the phrase "Run Javascript Everywhere"

Success Because → Event Driven architechure
Async I/O / Non Blocking I/O

History

- 2009, First developed using Firefox's Spider Monkey JS Engine but in 2 Days Switched to Chrome's V8 Engine & Never Looked Back
- 2010, NPM was developed to support Node Js by Issac
- 2011, Node.js given a windows support (Earlier only for Macos, Linux)
- 2012, Ryan left the Node.js project & it took over by Issac
- 2014, Fedora Industry forked Node.js and called it io.js, Lead to controversy within the comp. Joyent. A few devs began maintaining the io.js branch
- 2015, Node.js & Io.js → Merged, forming the node.js foundation
- 2019, JS foundation + Node.Js foundation → Open Js foundation.

EPISODE – 02

↳ JS on Server

Server → It simply means a remote computer that provides data, resources to other computers (client) which are always on & available with stable internet connections.

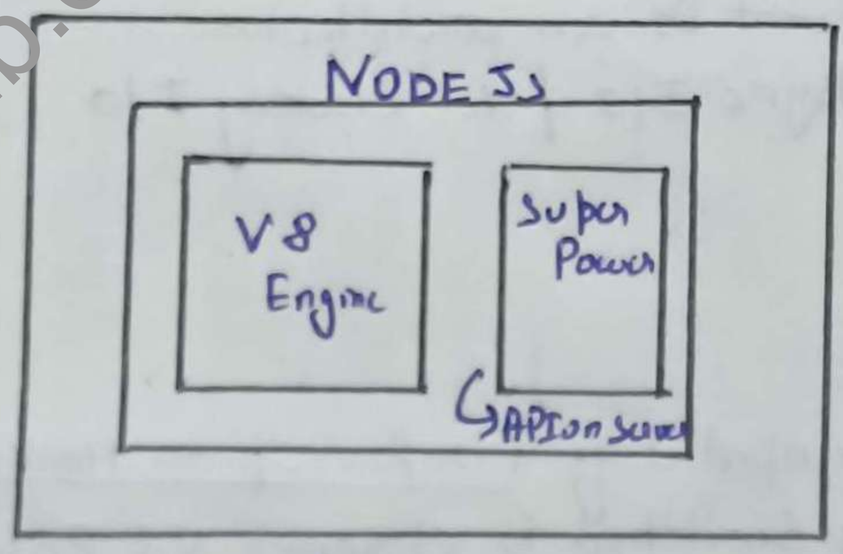
V8 Engine → It is written on C++ and can be embedded with my C++ application

- It is majorily used in chrome & NodeJS.
- It follows ECMA script standards & runs on various system

↳ Standards for scripting Language
Guidelines of writing JS, JScript, Action script etc

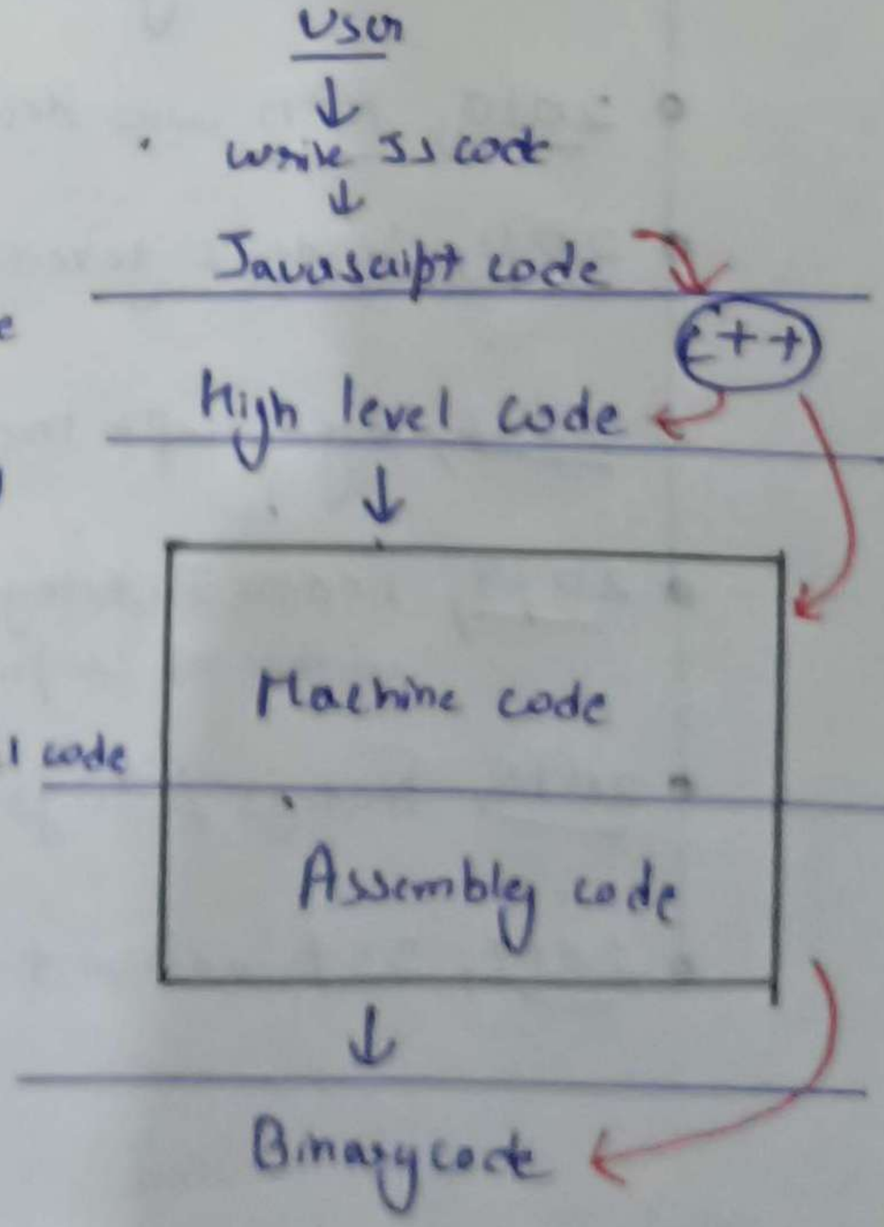
- It takes JS code & convats it into a machine code that computer can execute

Note → Nodejs is a C++ Application with V8 Engine embedded into it



Why V8 Engine is a C++ code?

Whenever we write a JS code. The JS code is processed by C++ program/ JS Engine (which converts JS code (high level) to low level) and converted to machine code and this is being understood by computer in the form of Binary



EPISODE - 03

↳ Lets write code

1) Install NodeJs

- Go to the website of NodeJs and download the latest version of nodejs
- Verify → node -v
  npm -v

* Node REPL (Read, Evaluate, Print, Loop)

- Node REPL is an interactive environment where we can write and Execute JS code directly into the terminal
- It is the easiest way of writing program

* Global obj in Node → global, it is one of the super power that nodejs gives us and it is not a part of V8 Engine

→ setInterval, setTimeout, clearInterval → Part of global obj not JS

→ 'this' Keyword is not points to global obj rather it is an empty obj (code in Repo)

EPISODE - 04

↳ Module.export & Require

- In Nodejs each files are called 'Modules'
- In every Nodejs project there is only one entry point (a file where the program start). if we have multiple files then we have to import those file into the entry point file using require(path)
- We couldn't simply import and use variables & fn from one module to another. Bcz modules protects their var & fn from leaking by default
- It access var&fn we have to export them.

* Types of Export & Import

Common JS Modules (CJS)	ES Modules (ESS)
1) Import : require()	Import : import {fn name} from "path"
2) Export : module.exports	Export : Export fn name/var
3) Default used in nodejs	Default used in React, Angular etc
4) synchronous & non-strict mode	option for async, strict mode is Default

EPISODE - 05

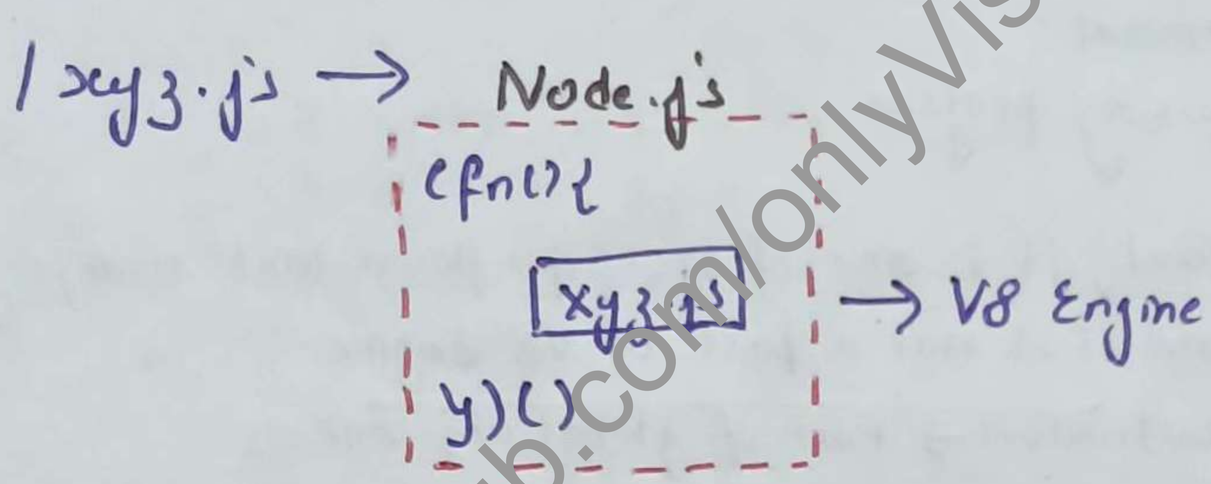
↳ Diving into the nodejs github repo

IIFE (Immediately Invoked fn Expression)

↳ Modules are wrapped in IIFE, called when require() statement runs
↳ Used to create private scope

```
(function (module, required, ....) {
// module
```

y)()



5 step Mechanism of Require

i) Resolving the module → check whether the file path is
- ./localpath
- ./.json
- /node! module

ii) Loading the Module → file content is loaded according to file type

iii) Wrapping Inside an IIFE (compile) → module code wrapped inside an IIFE, helps encapsulate the module scope. Keeps var & fn private.

iv) Evaluation → module.exports is set to exports available to other file

v) Caching → Node caches the required modules only once & made it available in my file whenever it required it without above steps

EPISODE-06

↳ libuv & async IO

Javascript is a synchronous single Threaded

↳ one after other

↳ can run on single Thread or single process

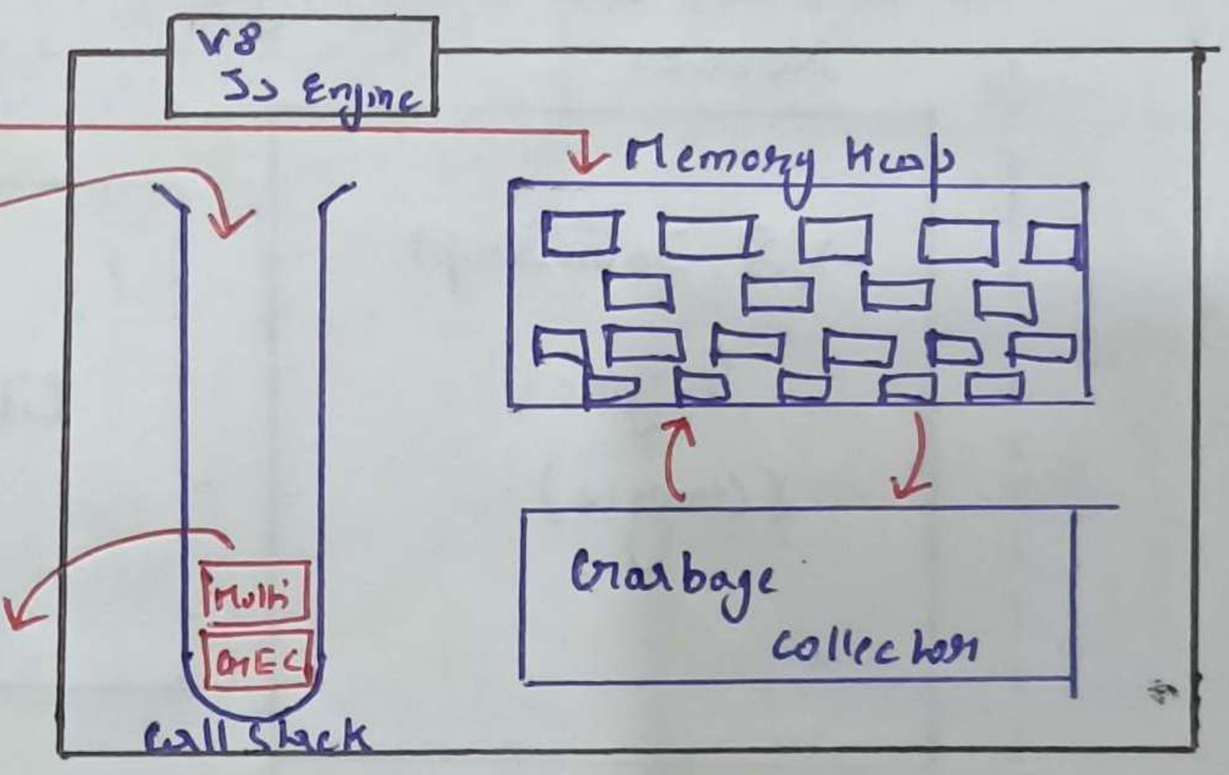
NODE Async JS

* Synchronous

Js code exwte line by line by default which mans one task has to wait for the previous task to get completed only after that the next task will get Exwted. It is Blocking

* Exwtion of synchronous code

```
var a = 1278698;
var b = 20986;
function multi(x,y){
    const result = x*y;
    return result;
}
var c = multi(a,b);
```



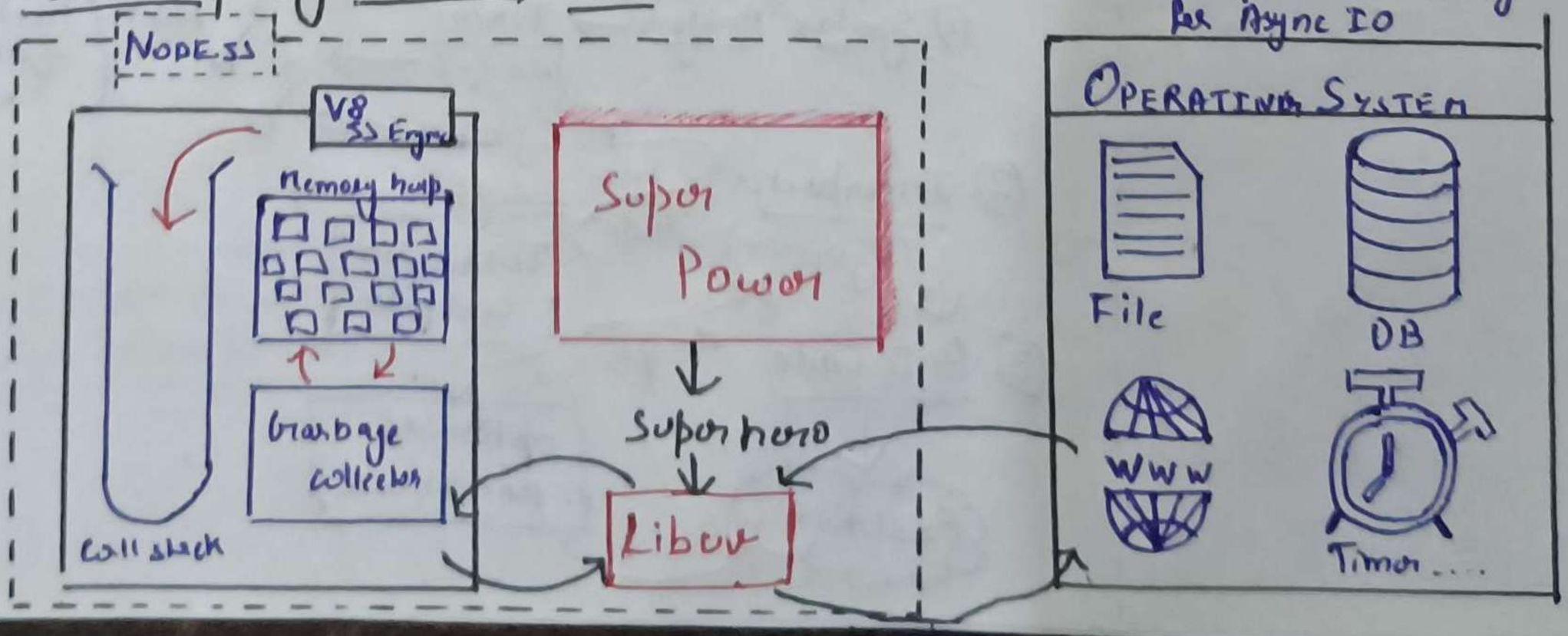
GEC → Global Exwtion context

* Asynchronous

one task does not have to wait for other task to get completed/exwted. Each task perform seperately. Though Js is synchronous & single treaded in default. This async operation is possible by of various ways like callback, promises, async/await, eventloop, setTimeout, setInterval & so on. It is non-Blocking

* Exwtion of Asynchronous code

Libuv = Multiplatform c library for Async IO



EPISODE - 01

↳ sync, async, setTimeoutZero - code

* Synchronous (Blocking IO)

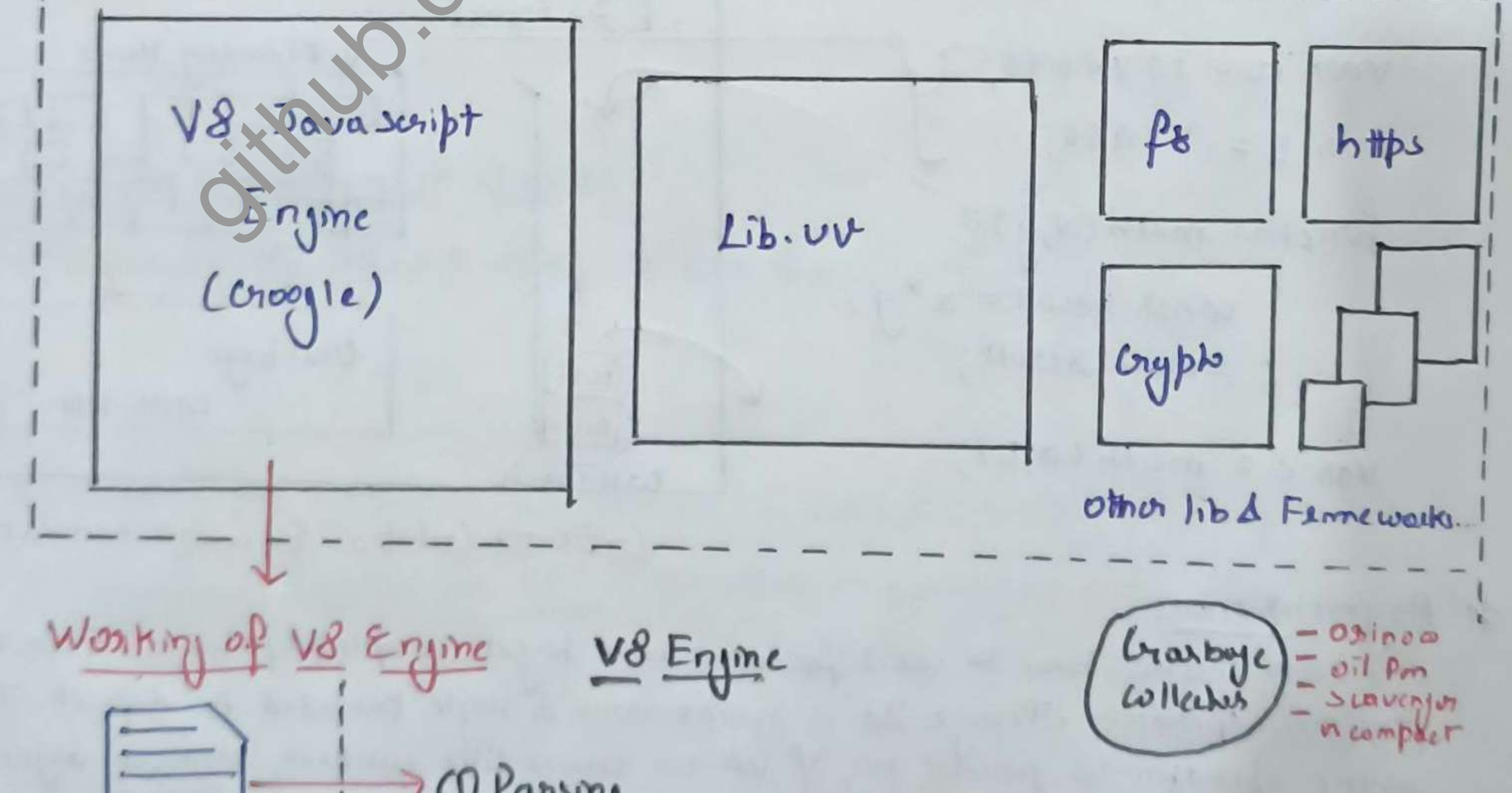
V8 Engine → Node.js c++ binding → OS Task → data returned →
C++ bindings → V8 Engine

* Asynchronous (Non-Blocking IO)

V8 Engine → libuv → OS Task → Data returned → libuv → V8 Engine
(Thread Pool) (call back Execution)

EPISODE - 08

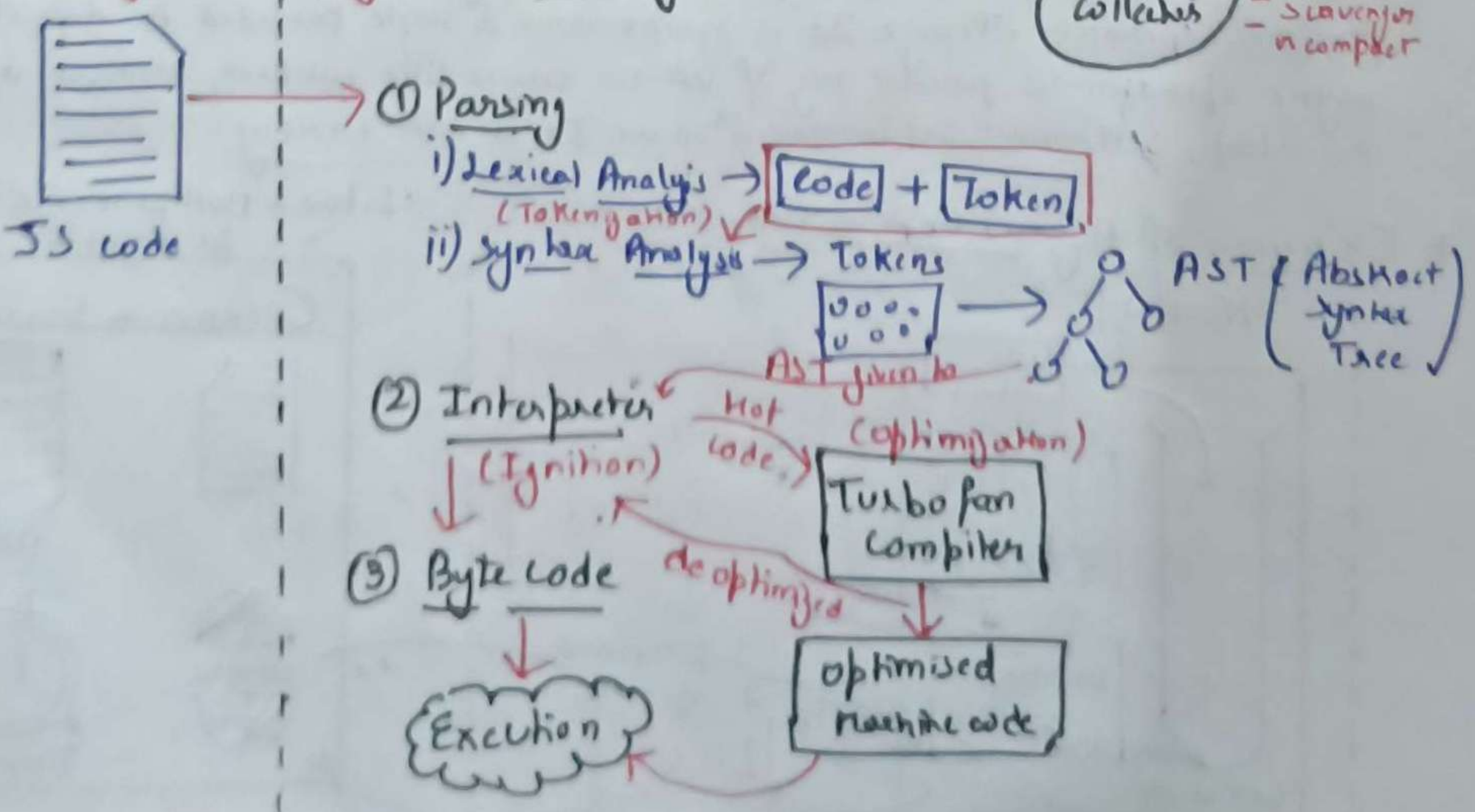
↳ Deep dive into V8 JS Engine



Working of V8 Engine

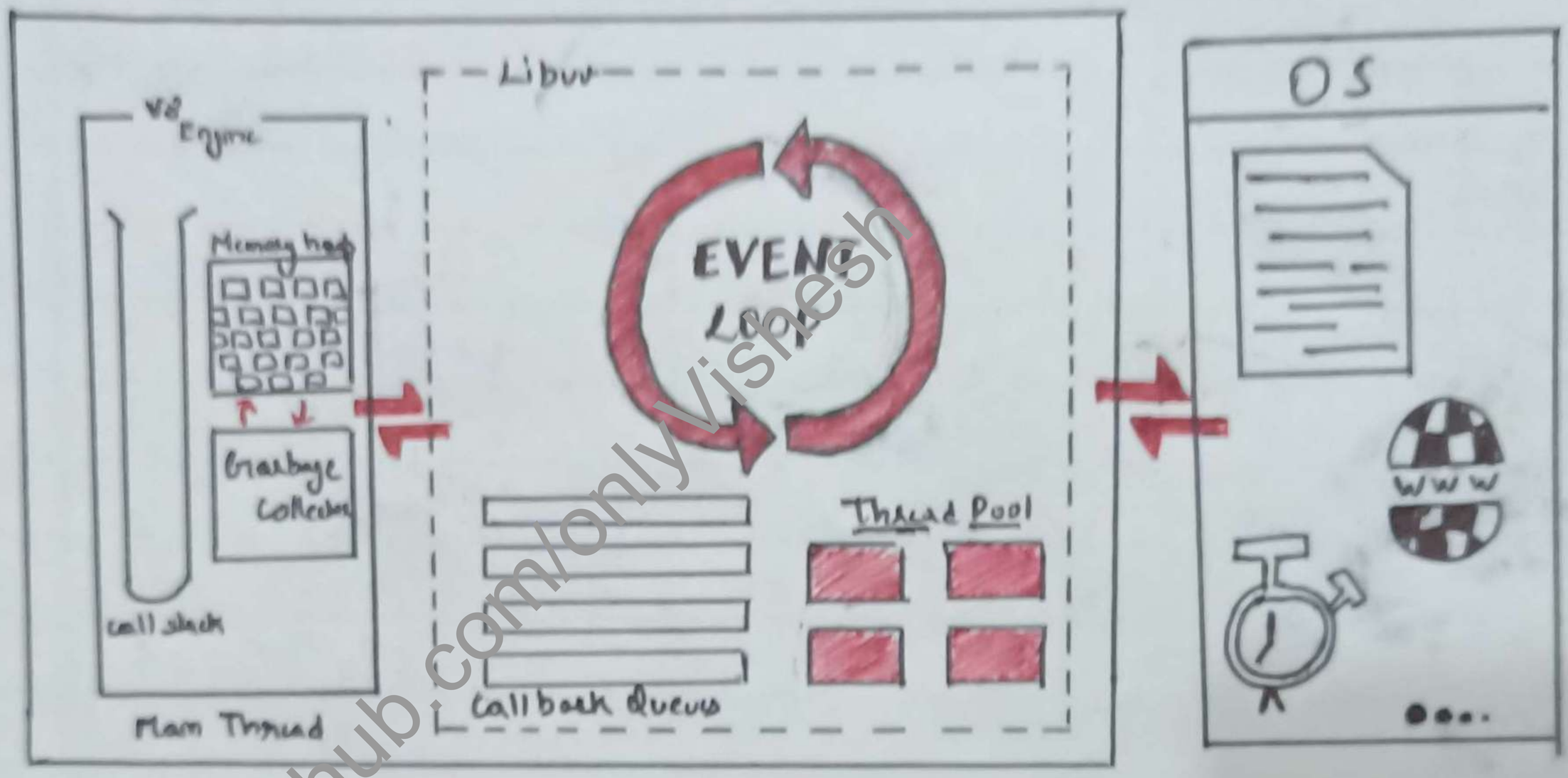
V8 Engine

Garbage Collector
- Orinoco
- Oil Pan
- Scavenger
- Compactor

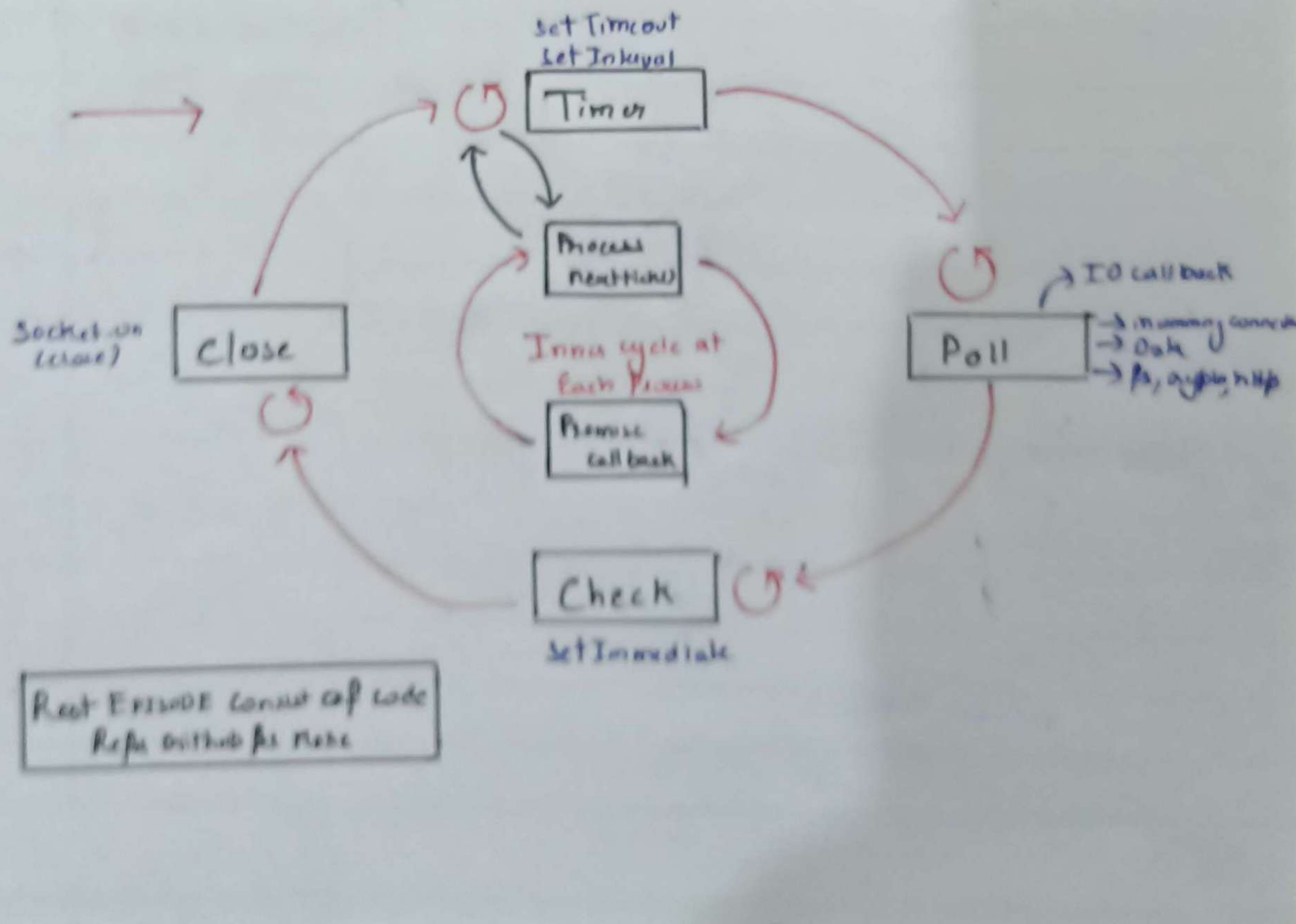


EPISODE -09

↳ Libuv & Event loop



* Event Loop



Next EPISODE consist of code
Refer github for more