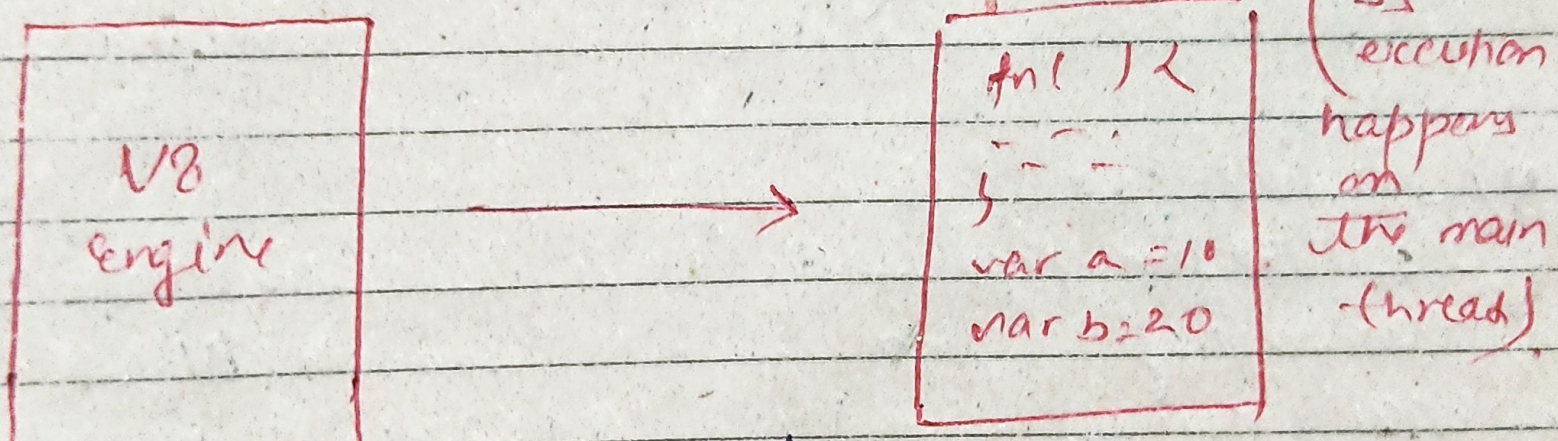
Namaste Node.js

Chapter - 06 - Libuv & Asynchronous I/O

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

→ Javascript is a) synchronous → (line by line code execution.
b) single threaded (only one thread on which V8 engine runs)



→ A thread is the smallest unit of execution inside a process.

→ A task running inside the program / process.

→ Lightweight than process.

(code gets executed in synchronous way in Javascript (i.e. before moving onto the second line, first line of code must be executed.)

→ and on single thread whole code runs.

Synchronous v/s Asynchronous

1. Execution Model

Feature	Synchronous	Asynchronous
a) Execution order	code runs line by line, blocking further execution until current task completes	code can start long tasks and move on without waiting.
b) Blocking	Yes, blocks the call stack	No blocking
c) Example	CPU-heavy loops, calculations	API fetch, timers, file I/O

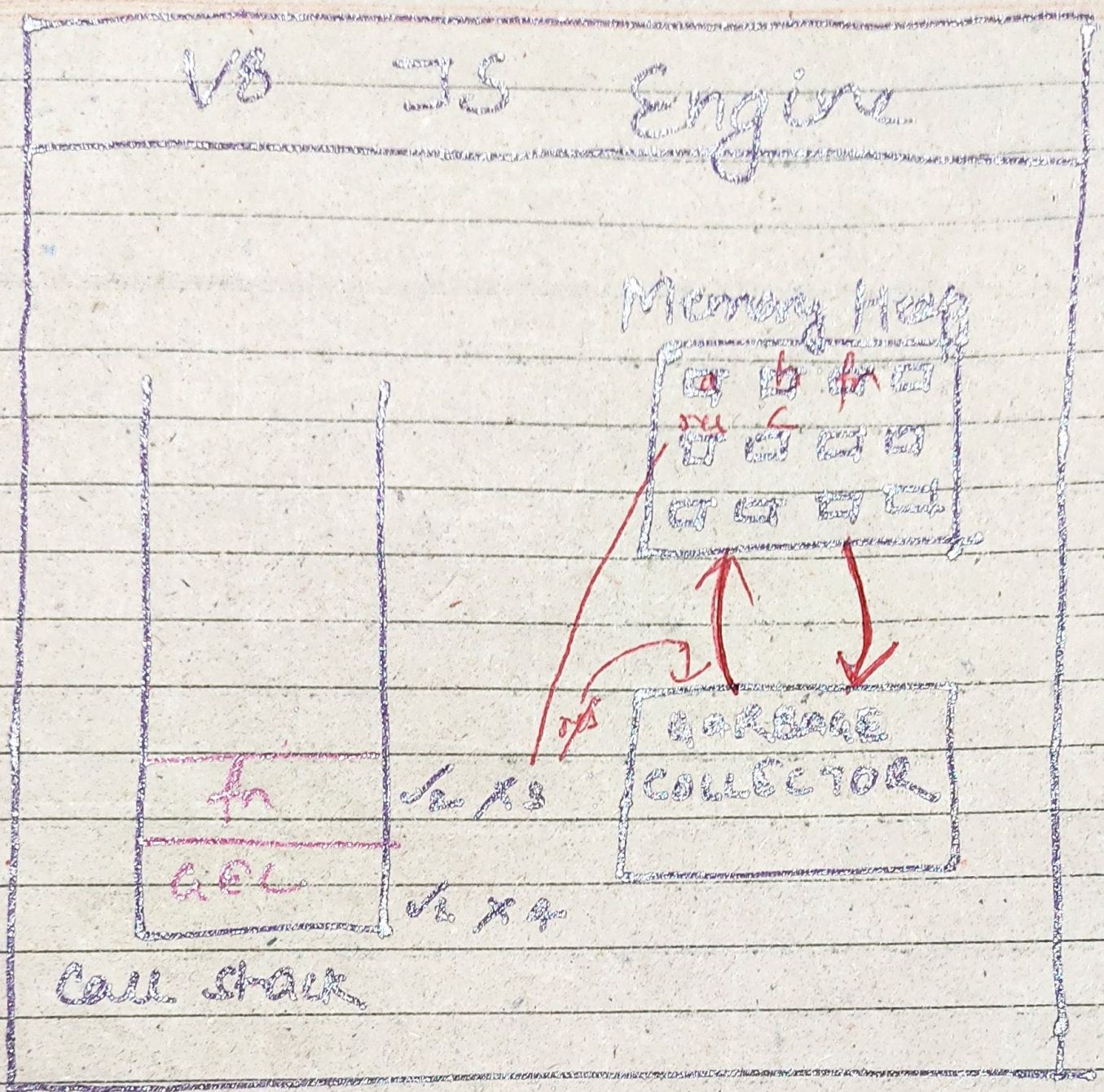
2) Impact on Performance

a) UI Responsiveness → can freeze UI → i) Keeps UI smooth
b) Concurrency → None → Allows concurrency
c) Wait times → High (because each task waits) → ii) through callbacks promises
→ iii) Lowers (tasks overlap)

3) Where Each is used

	Synchronous	Asynchronous
	a) Pure computations	a) Network calls
	b) Basic scripts	b) DB queries
	c) Immediate operations	c) Timers
	d) Blocking loops	d) File system operations

Synchronous code Running in JS Engine



→ var a = 1078698;
var b = 20986;

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

var c = multiplyFn(a, b);

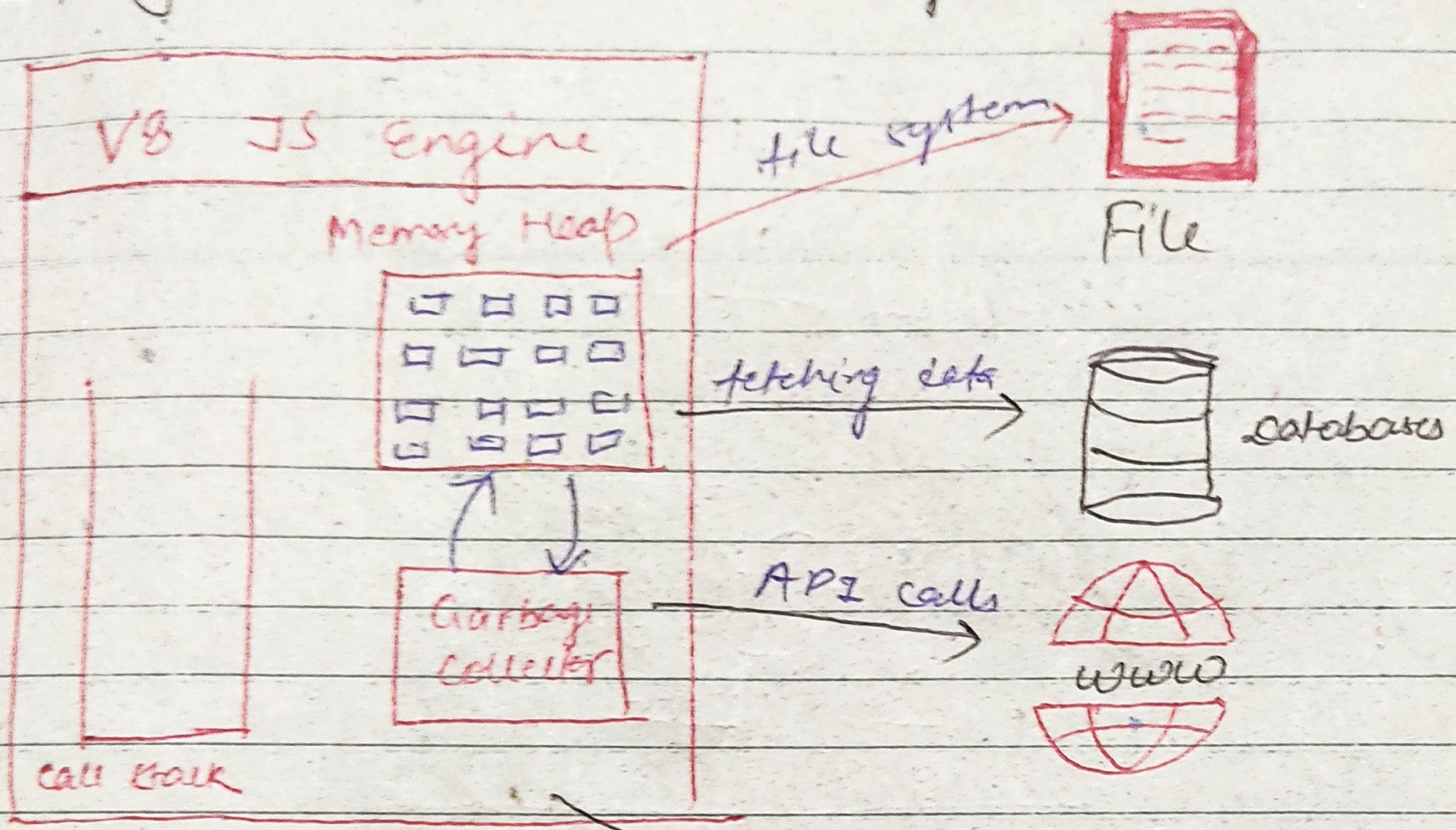
i) all the functions and variables get stored inside the memory heap.

ii) whenever a file runs, a GEC gets created onto call stack.

iii) If a function gets called, its FEC gets pushed onto the stack.

iv) once everything gets executed, EC get destroyed, var are Garbage Collected.

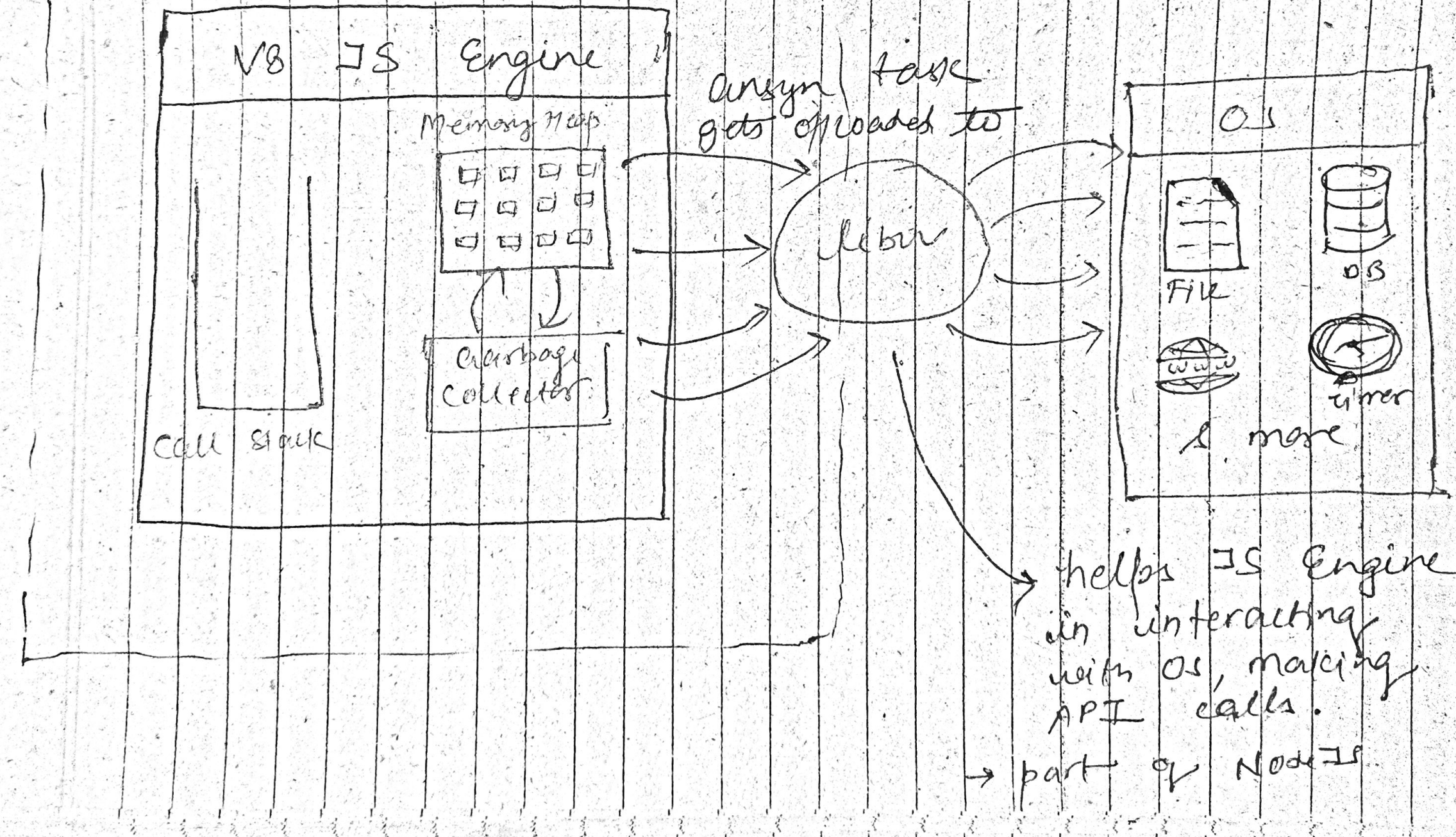
Asynchronous Code Running in JS Engine



→ these all other functionalities are not part of core JS (e.g., timers), but provided by its runtime environment.

→ all the asyn operation gets offloaded to libuv.

NodeJS


V8 JS Engine
Memory Heap
Call Stack
Garbage Collector
async task gets offloaded to
libuv
OS
File
OS
Timer
& more
helps JS Engine in interacting with OS, making API calls.
→ part of NodeJS

Asynchronous I/O made simple

3) What is libuv ?

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

ii) It supports epoll(4), Kqueue(2), Windows IOCP, Solaris event ports and Linux io-uring.

iii) It is primarily designed for use in Node.js but it is also used by the other software projects.

iv) Node.js is an asynchronous because of libuv (because it contains Event Loop, ThreadPool ----).

v) It also interacts with operating systems. (works as a communicator between Node.js and operating system). (its part of Node.js)

Q) How the Javascript file runs?

A) i) A global execution context is created and pushed onto the call stack. (all code runs inside it in single ~~sync~~ threaded sync manner.)

ii) Variables and functions gets stored inside the aEC in (key, value) pairs and empty spaces in memory heaps.

iii) Garbage collector works in sync with the memory heap. It collects and clean the garbage. (destroy those (free memory from them) variables which doesn't have any references (they are not in use now).).

iv) Whenever JS engine encounters any asynchronous operation (API calls, timers, file read/write, etc.) these thing happens
a) it calls libuv to execute those asynchronous calls.
b) libuv registers this asynchronous call and it will also take the callback (store it)
c) JS engine moves to the next line and handles

synchronous (execute it itself) and asynchronous operations (offloads to libuv) accordingly. (Main thread doesn't get blocked.)

d) If it encounters any function calling, a functional execution context gets created and pushed onto the call stack. If the whole code (function) gets executed, it moves out of the call stack and garbage collector will free up all the memory occupied by this function. (synchronous)

e) When the whole js file finished executing, GEC popped off the call stack & Garbage collector will free up all the memory spaces occupied by the whole js file. (call stack becomes empty).

*) Behind these all things, libuv is managing and handling all the async operations,
Suppose an async operation (fileread) gets completed and it has acquired the required data then it gives the

callback function (associated with fsreadFile) and gives it to V8.

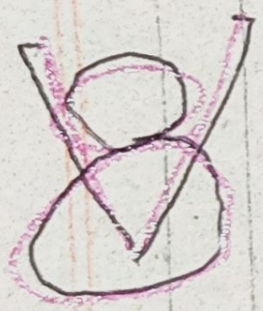
g) V8 engine → call stack pushes the callback function to get it executed in single threaded synchronous manner, gets executed and popped off the call stack.

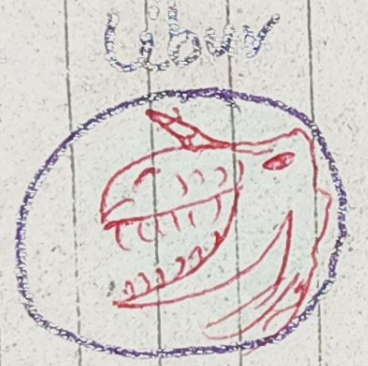
h) Suppose many other async operations gets resolved by libuv, then same thing will be done (pushed to js engine and to get executed) for them.

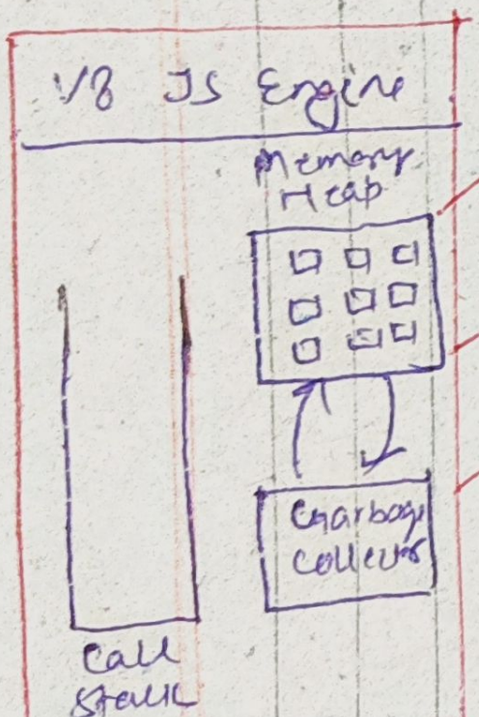
NOTE :: Node.js in Asynchronous Javascript Engine is synchronous.

- In these manner, Node.js able to perform asynchronous I/O without blocking the main thread and being single threaded.

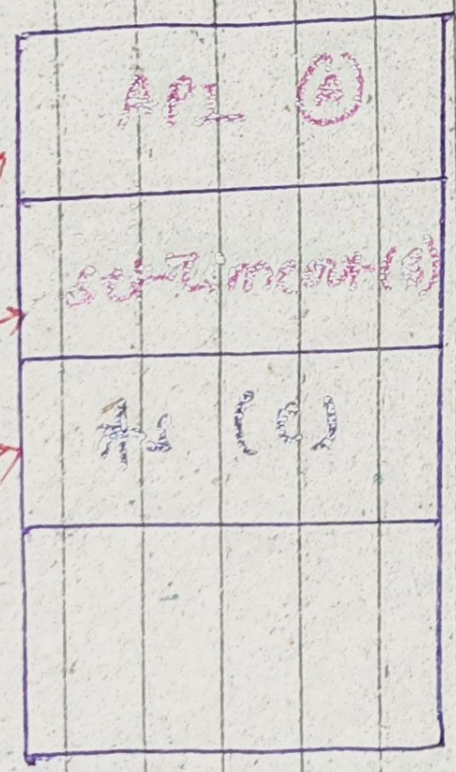
NODE.js (Asynchronous)



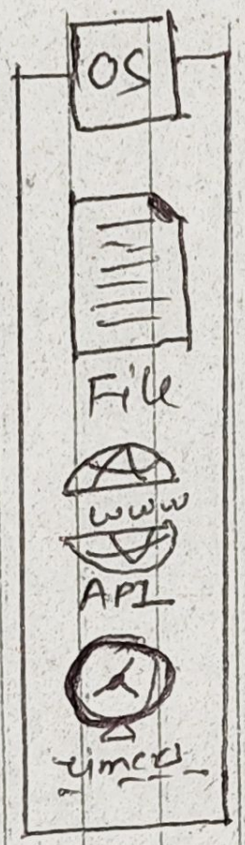




(Synchronous)



(libuv library)



All async offloade to libuv making main thread free

JS code

```
var a = 10;
var b = 20;

https.get("https://api.mcn.com",
(res) => {
  console.log("res?.sources");
});

setTimeout(() => {
  console.log("setTimeout");
}, 5000);

fs.readFile("./gossip.txt", "utf8",
(data) => {
  console.log("File data", data);
});

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

var c = multiplyFn(a, b);
console.log(c);
```