Date
Page 1

History of Node JS

~~2005~~ ⊙ 2009 ← mac os and linux

Ryan Dahl
- Spider monkey (firefox) 2day.
- V8 engine (Google Chrome)
- JavaScript core (Nitro) used by apple safari
- Chakra : used by microsoft edge legacy. discontinued in 2020.

Earlier name of node.js was web.js ← Create web server

↓

Node.js ← Non blocking I/O

Before this | Apache HTTP server Blocking server

* Advantage of non-blocking or Node.js is it can handle multiple request with lesser number of threads

⊙ 2010
NPM

⊙ 2011
(windows support)
(joyent + microsoft)

Production ready

⊙ 2012
Ryan dahl left the Project and Isaac took the responsibility (he also created NPM)

⊙ 2014
(-Fedor) he created fork (io.js)

Joyent → Node

Node → Node / IO.js ← Fedor

⊙ 2015

← got merged

Node.js started Committee

⊙ 2019

JS. foundation + Node.js foundation

got merged.

open js foundation

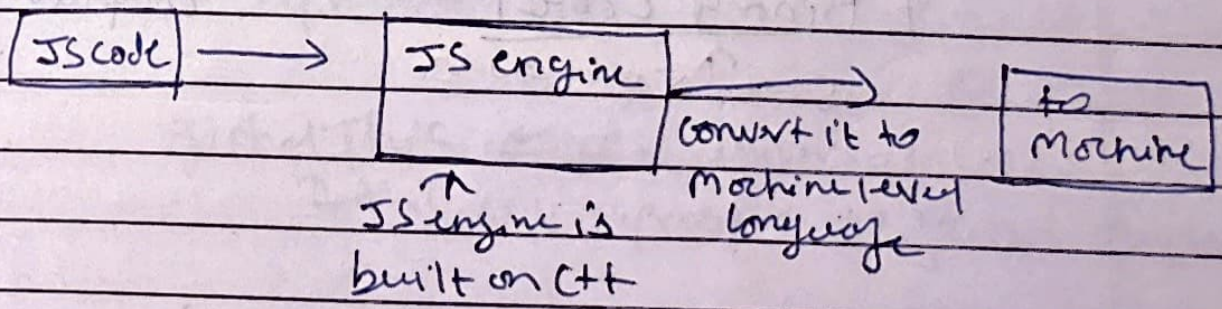
Open.js is Responsible for everything they are Managing.

Node JS - Javascript on server.

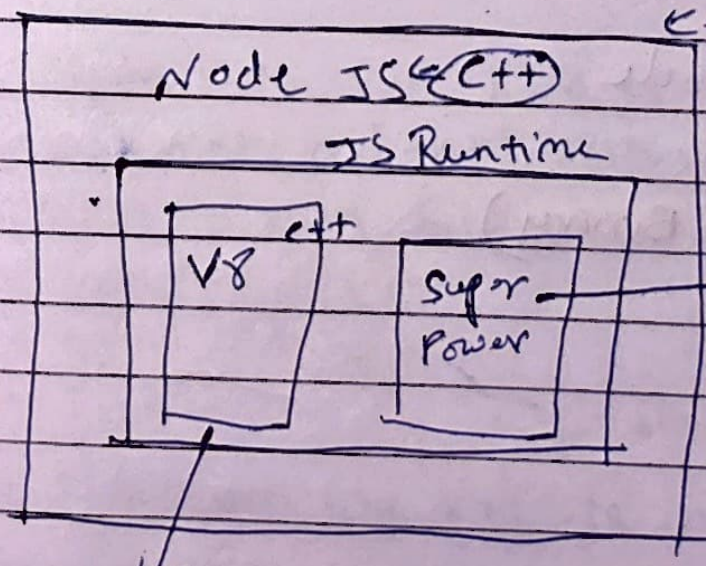
ECMA Script
- Standards / Rules

(JS engine follows this standards.)

they have to follow because they should give some output.



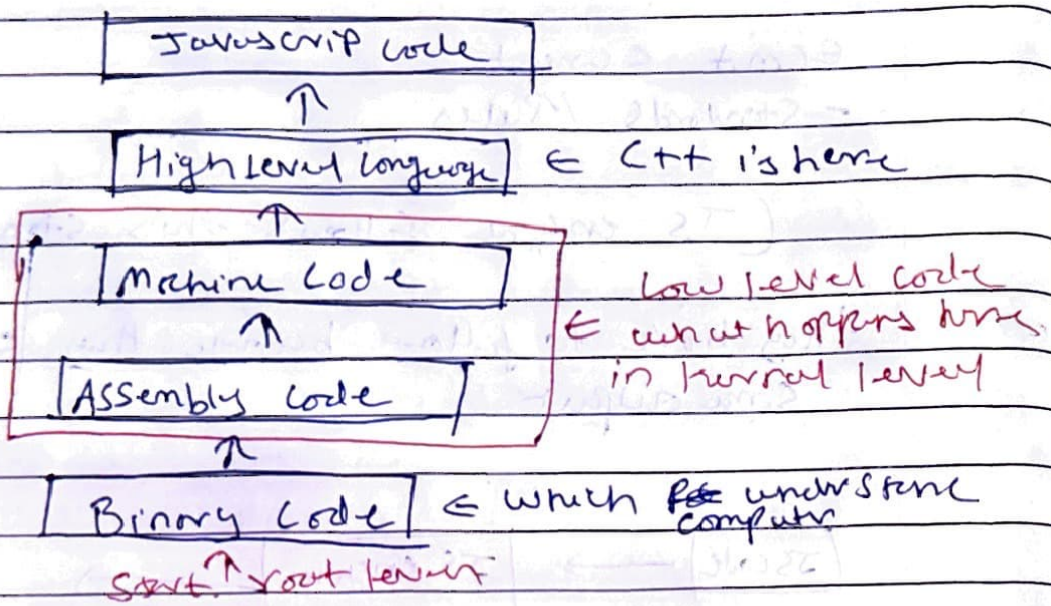
Node JS is a C++ application with V8 embedded into it.



← Servers.
← Ecmascript.

API on servers / modules.
It contains JS + V8.

V8 is a C++ code



JS
↓ ∈ Processed by C++
High level ⇒ C++ (JS engine)
↓
Low level code. ⇒ Machine code
↓
Computer (Binary)

Date
Page 5

If you do consde log of global it will be global but in browser it was window

and consde log of (this) will be empty object

in browser.

#window #this #self #frames all will show you (window object.)

global object.

There should be some and single way to represent global object.

globalThis ← it's universal.

↑this will represent to global object

(globalThis === global) → true.

folder structure

whenever we run node.js Project we have entry point and that entry point we give to terminal to run. But what if we have new code in seprate file.

app.js

xyz.js ← called this module

how we will use xyz.js in app.js we can use it via require

require("Path"); ← module import.

Created new module and imported that module to our app.js.
using [require()] function.

require("./xyz.js");

Let's we have sum.js modules.

// Modules protects their variable and function from leaking.
By default
to use that or use any function to outside.
~~function~~ module. we have to export.
exports ← it have s

module.exports = CalculateSum;
and we have to import it to our app.js.

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

To export multiple file we have to make it as object.

module.exports = { x : x, calculateSum : calculateSum }
{} (module.exports) is empty object
exported two object.

To import.

const obj = required("./sm.js") ← or we can destructuring. on the fly

let obj.calculate(a,b)
(obj.x)

const {x, calculateSum} = require("./sum.js")

→ import via destructure
and export shorthand.

module.exports = {x, calculateSum};
this is ↑ this is shorthand
{x: x, calculateSum: calculateSum}

↑ we will use this shorthand and destructure only

Modules Protect their variable and function from leaking

we have created 3 files and imported and exported all the thing check it on git repo.

This pattern is known as common js modules.

common JS modules (CJS)	ES modules (ESM) (.mjs) (ES6 module)
→ module.exports (require())	to use this we have create package.json and we have to give "type": "module" ← this is different way in (.mjs) or ES modules. we export using export function
if we don't have package.json it will be default	
- by default used in Node.js	
- older way	→ import and export
- Synchronous calls	→ by default used in React, angular.
- Not strict mode	- newer way ↑ this will become standard way.
	- Async option is there. ↑ very powerfull
	- strict mode is default.

We create folder to and ~~export~~ import everything to index like calculation modules will be inside of Calculate folder.
and then we import it from there.
it is a good way.

when we call require() ← all code is wrapping up in IIFE

it wraps all codes inside a function (IIFE)

IIFE ← immediately invoked function expression.

(function () { ← anonymous function without a name.
// all codes of the modules runs inside this
})(); ← we are invoking it immediately.
↑ wrap inside above part.
and then it will give it to V8.

# IIFE → Immediately invoked function expression.
(function () {

})()
↑ immediately invoked it.

why do we need IIFE
- Immediately invoked
- It keeps your ~~function~~ variables and function (Private)

The code will not interfere to other code.
It's independent code.

Date
Page 9

Q How are variables and function private in different modules?

- because of IIFE and require statement.

require wrap modules code inside IIFE ~~and~~

module.exports = { module name };

this is given by node

when it wraps it inside a function it pass export. module over there.

Promises

(module, require) also passed if we are importing another module.

(function (){ require(path)

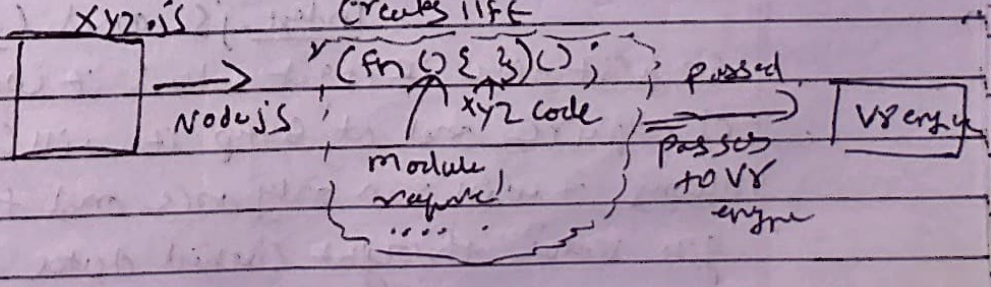
// code that module have

})(){}

module.exports → it's empty object.

How do you get access to the module.export

→ Node.js provides (modules) as a promise through IIFE



require (/path)

1 Step → Resolving the module

→ it checks whether the module is /path, .json node
sees types of data.

2 Step → Loading the module
↳ file content is loaded according to file type

3 Step → wraps inside an IIFE → immediate invoked function expression.
↳ (Compile Step)

4 Step → Code evaluation. → code executed
↳ ~~require~~, module.exports happens here
↓ returns outside then we get the data.

5 Step → Caching — Imp

~~Cache~~ Caching 5 Step

what if one module ~~file~~ is required with multiple files.

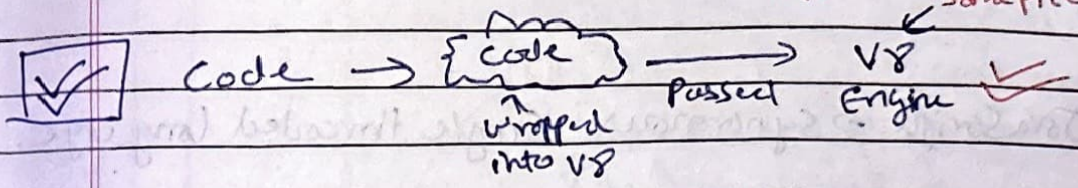
Suppose sum.js module is getting required with 2 file ① ∈ ranord cached get cached data.
index.js and calculate.js

so first time if it get called it will do all 5 Step of require and at 5 Step it will cashed it i.e means it will run only once and then it will give value through cached data (it will not do all 5 Step) it will direct get from cached.

Date
Page 11

All the code which we execute is first wrap into IIFE and it get passed to V8

V8 it's just a folder inside node repo
Some pice of C++ code

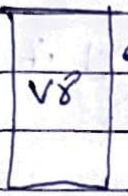


(✗) Code → V8 ∈ (its wrong it first wrapped with IIFE)
✗ wrong ✗ ✗ ✗ ✗

Libuv and async IO

Node.js has an event-driven architecture.
Capable of asynchronous I/O (Key line)

JavaScript → synchronous Single threaded language.



← one thread.

Js don't have multithreaded language

all code will run line by line.

Synchronous

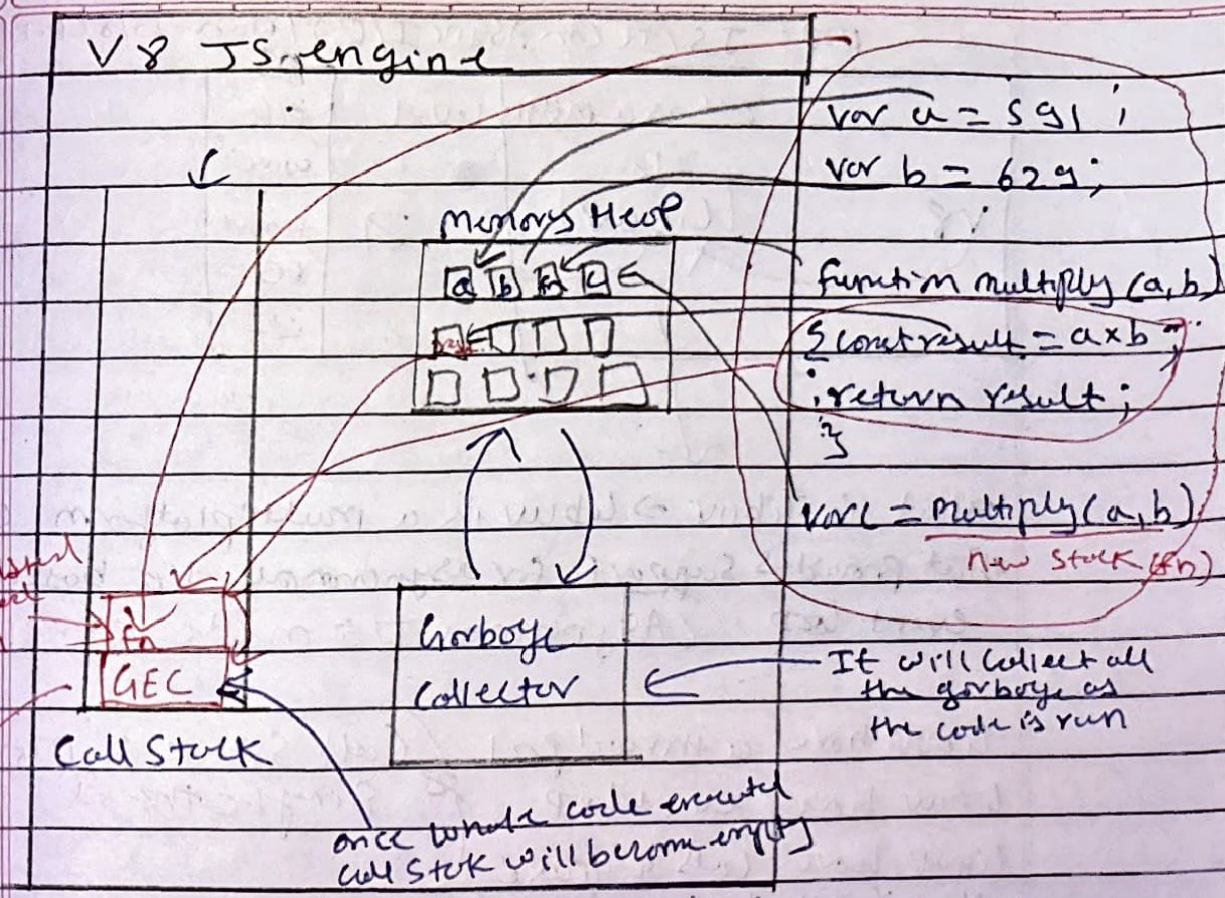
```
var a = 10256792;
var b = 5927625;
const multiply ((a,b) => {
   const total = a * b;
   return total;
}
multiply(a,b);
```

↑
this task are very fast
it can execute quickly

Asynchronous

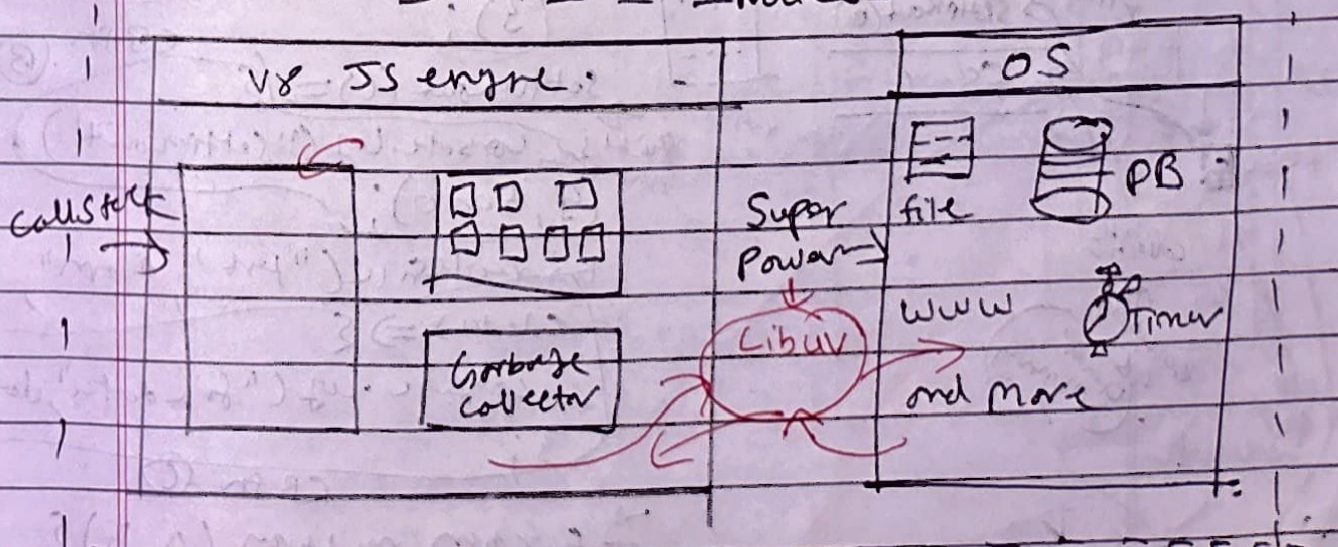
```
http.get("link", (res) => {
   const.log("data" + res.data);
})
fs.readfile("name", "utf8",
(data) => {const.log("data", data)
});

setTimeout(() => {
   const.log("call for 5 second");
}, 5000);
```

{ this task take time to execute.



This code will run in synchronous single thread.
it will be executed one by one.

This was synchronous way. but what if we make asynchronous. Let's see.



libuv is like a genie for us.

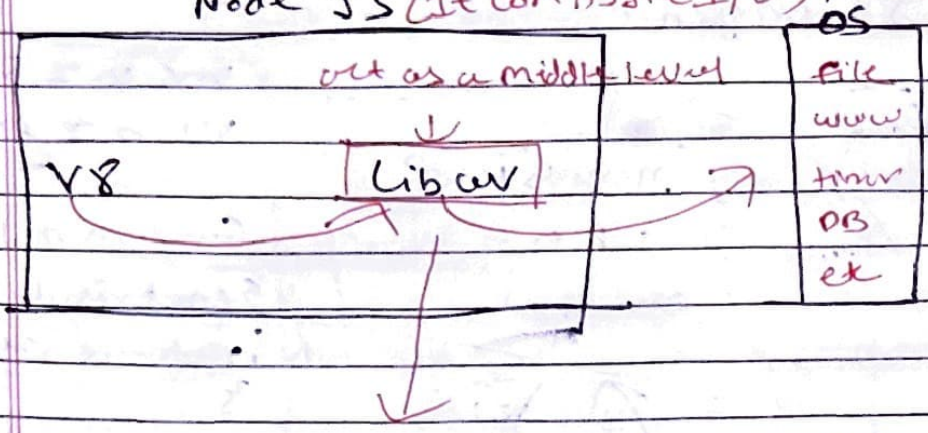
V8 engine → libuv → OS
req res

V8 engine ← libuv ← OS
req res

Node JS is Asynchronous → using power of libuv
V8 engine is Synchronous.

Date
Page 14

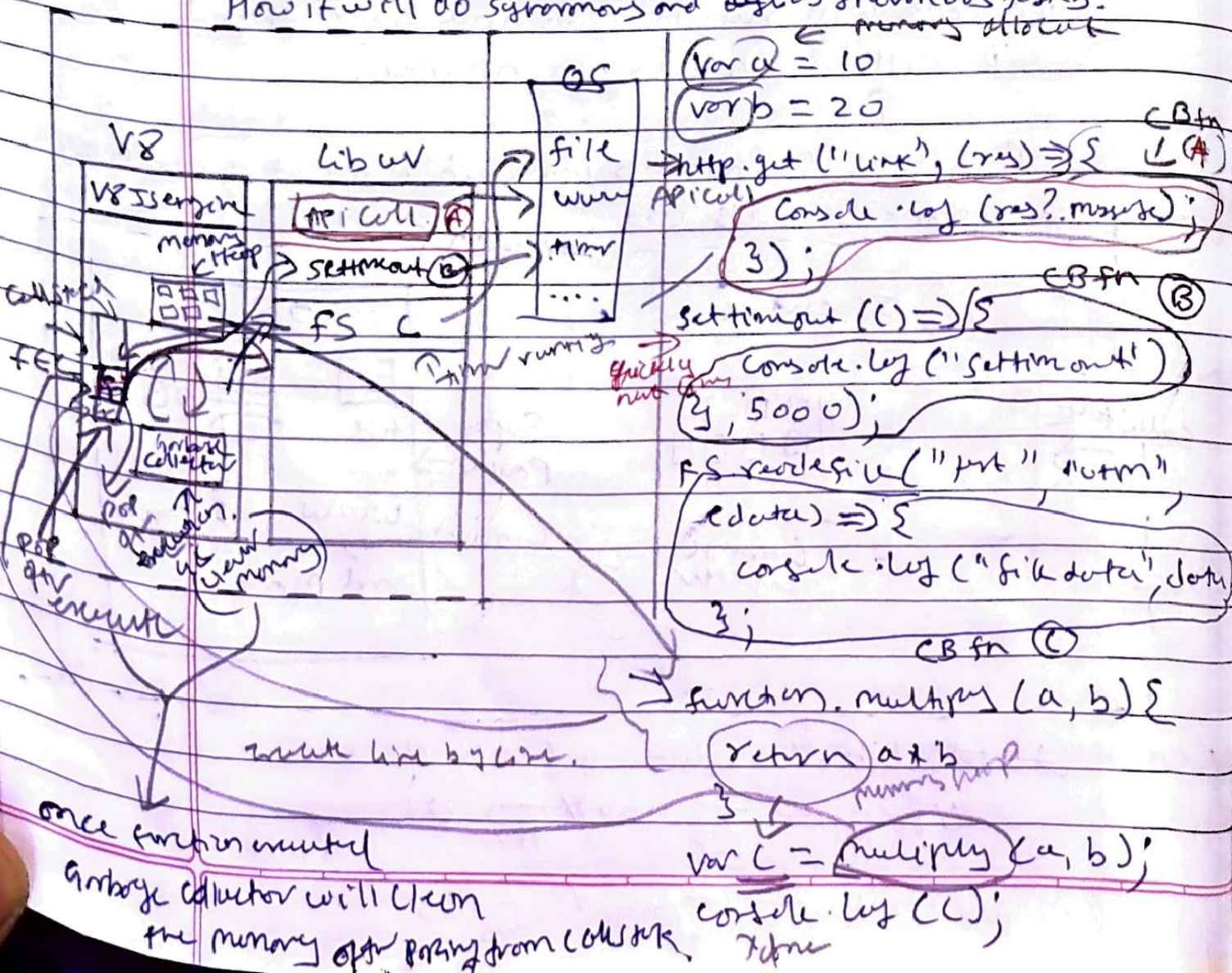
Node JS (It can Async I/O) / non-blocking I/O



What is libuv → libuv is a multiplatform C library that provides support for asynchronous I/O based on event loop (Asynchronous I/O made Simple)

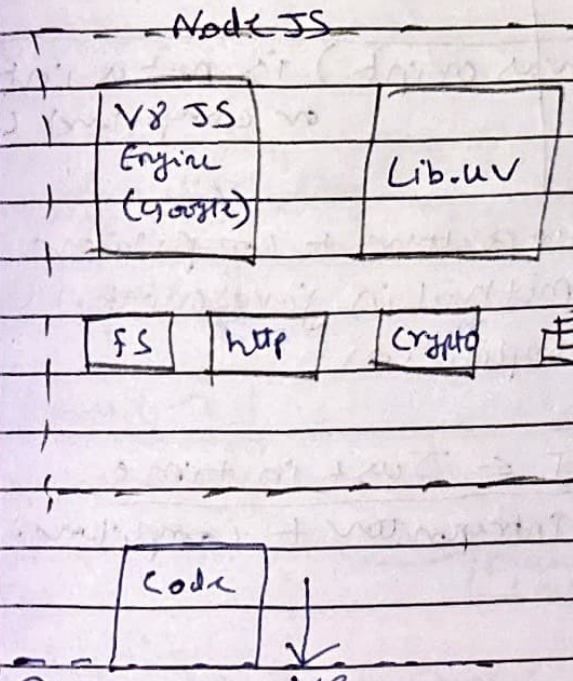
Libuv have a thread pool (Call Stack is main only Single thread)
Libuv have event loop.
Libuv have lots of queue

How it will do synchronous and asynchronous tasks.



Date
Page 15

Dive Deep into V8 Engine



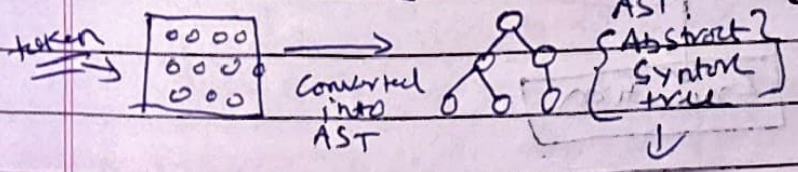
A- 1st Step - Parsing Phase

① lexical analysis (tokenization)

Code → tokens ← Code is broken down into tokens (a) (=) (let) (10)

② Syntax Analysis (Parsing) - (AST Explorer.net)

← translate code into AST




```
var x = 10;
```

kind — identifier — literal

why it or why we get Syntax error because it cannot generate abstract syntax tree

B. Second Step [INTERPRETTER] ~~is~~ is JavaScript is interpreter or compiled

AST → interpreter. (Passed to)

There are two type of languages

interpreted	compiled
→ Code read line by line	→ first compilation (whole code is converted into machine code) and then machine code is executed
→ fast initial execution	→ initial very but executed fast.

- interpreter	→ compilers.
	(Javascript) is not a interpreter or compiled language

Javascript uses interpreter + compiler
The compilation method in javascript is known as JIT compilation

~~J → Just~~ JIT ← Just in time
(JIT) mix of interpreter + compiler.

Back to topic

AST →
[interpreter] ← none of this interpreter is ignition interpreter

convert code into Byte code ↓ AST converted to Byte code

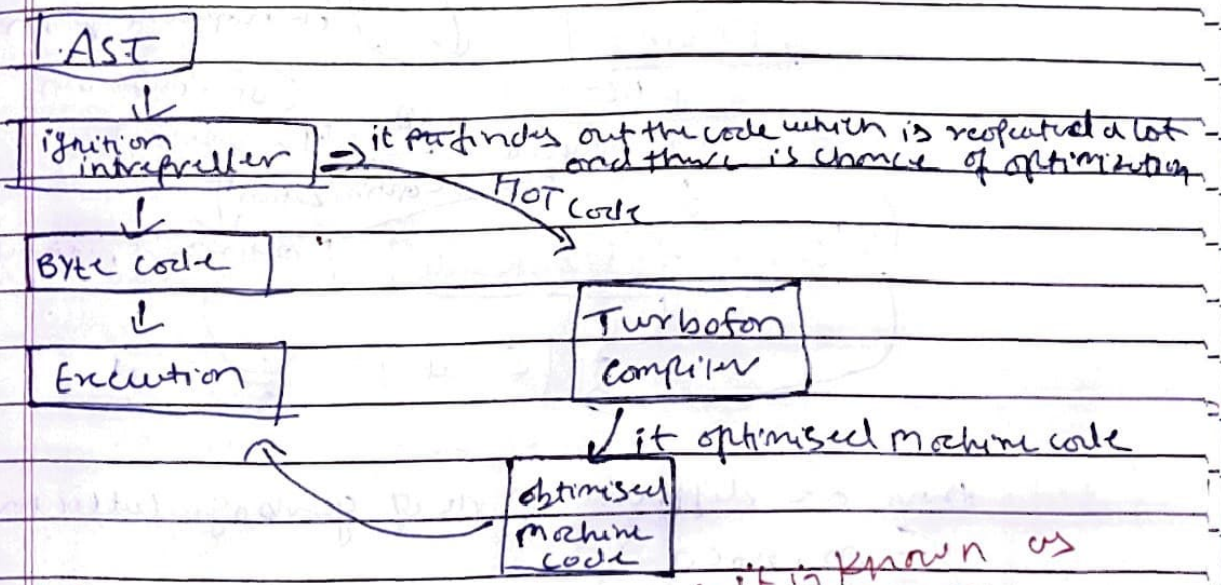
Byte code

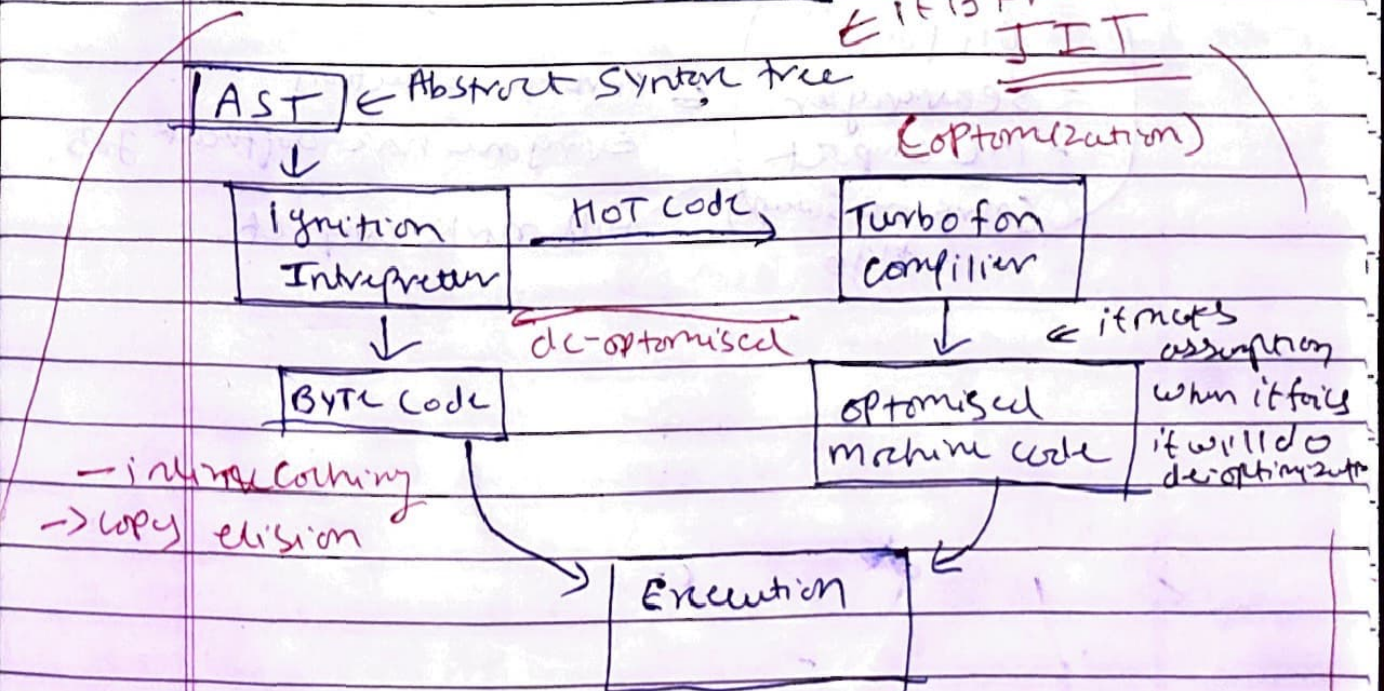
Execution

where compiler comes in.

Compiler → None of the compiler is Turbofan compiler.

2nd Step without explanation.





- inlining
- Caching
- -> copy elision

de-optimised (Suppose we have (sum function) now JS see that is has called to many time and then it ignition interpreter → Sends that code to turbo fan Compiler

Turbofan Compiler

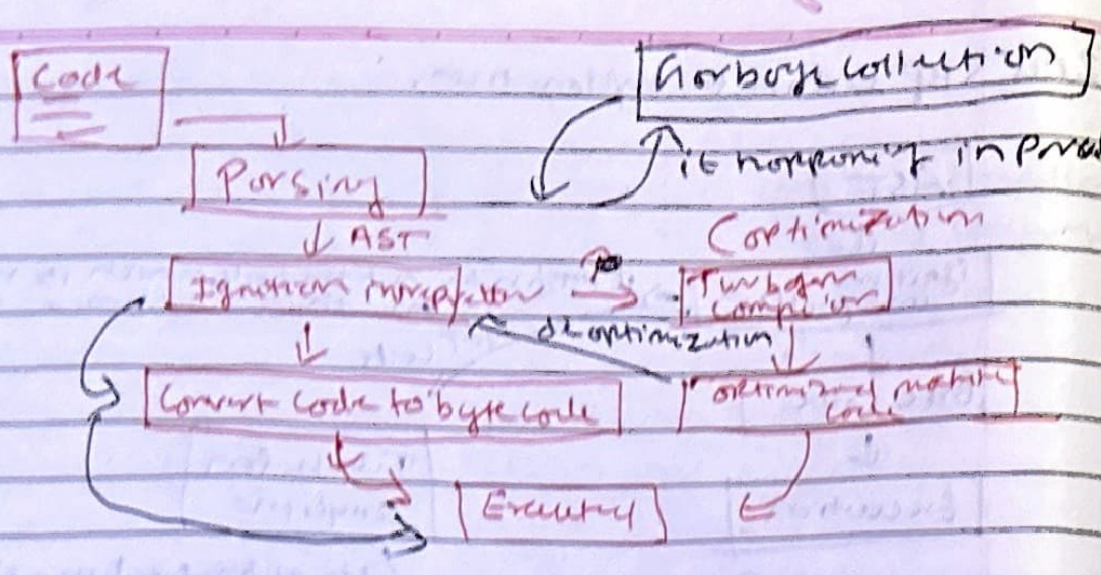
(Sum(a,b)) ← have compiled fails.

it will de optimized the code and interpreter will convert into byte code and this will execute the code

But we pass (2nd String)

But while optimizing it makes assumption → Sum-(10,15) it assumpt that it will be number not string

it ran very fast → Sum (2,3)



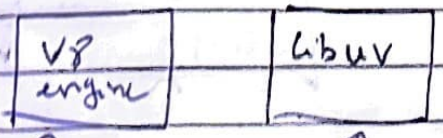
There are different type of garbage collector.

- orinoco
- oilpan
- scavenger
- MC compact (mark and sweep) Algo

← Garbage collection
Everyone have different job.
full mark compact.

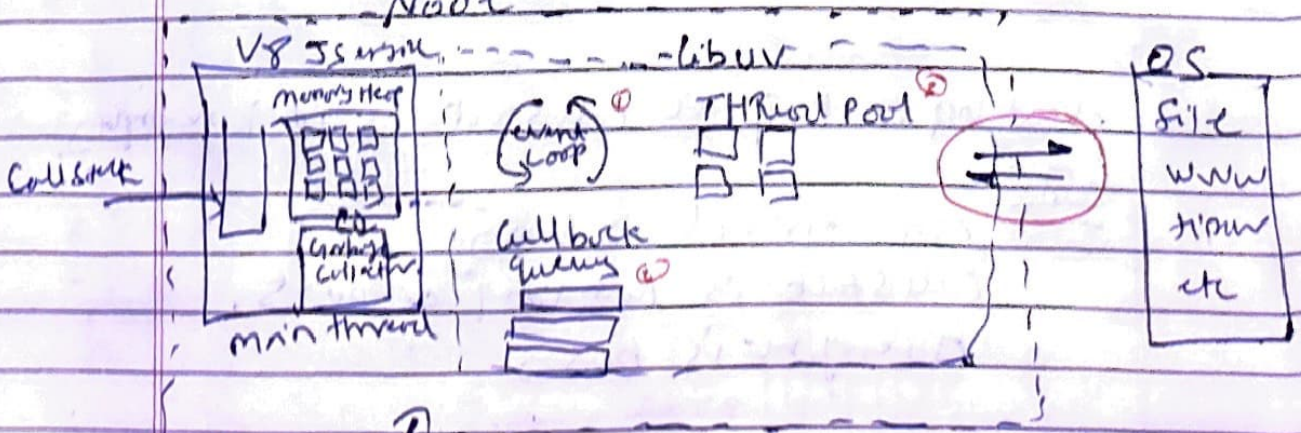
Libuv and Event Loop.

Node JS



↑ we have Studied this

↑ now this.



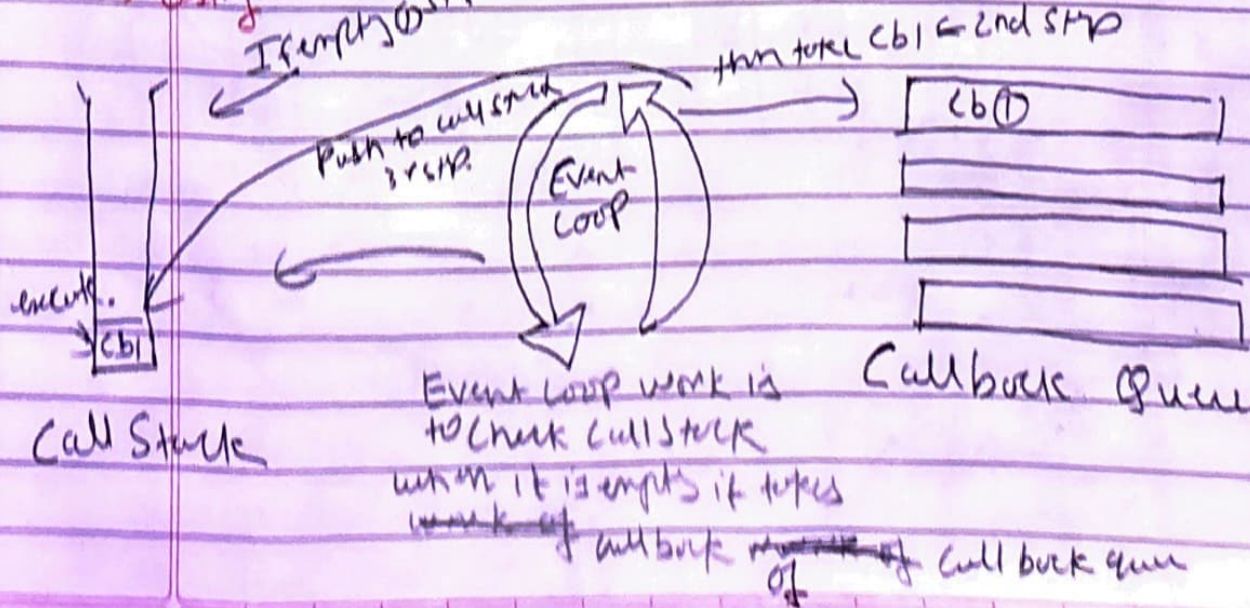
Asynchronous I/O (Non Blocking I/O)

Can only be done with the help of libuv

V8 JS ~~is engine~~ → ~~single~~ Synchronous single threaded language.

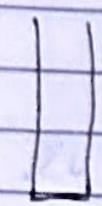
⚠ Java Script is Synchronous single threaded language

→ using



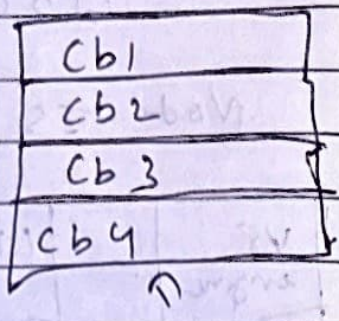
and Push to call STACK.

IS race condition.



Call Stack.

event loop



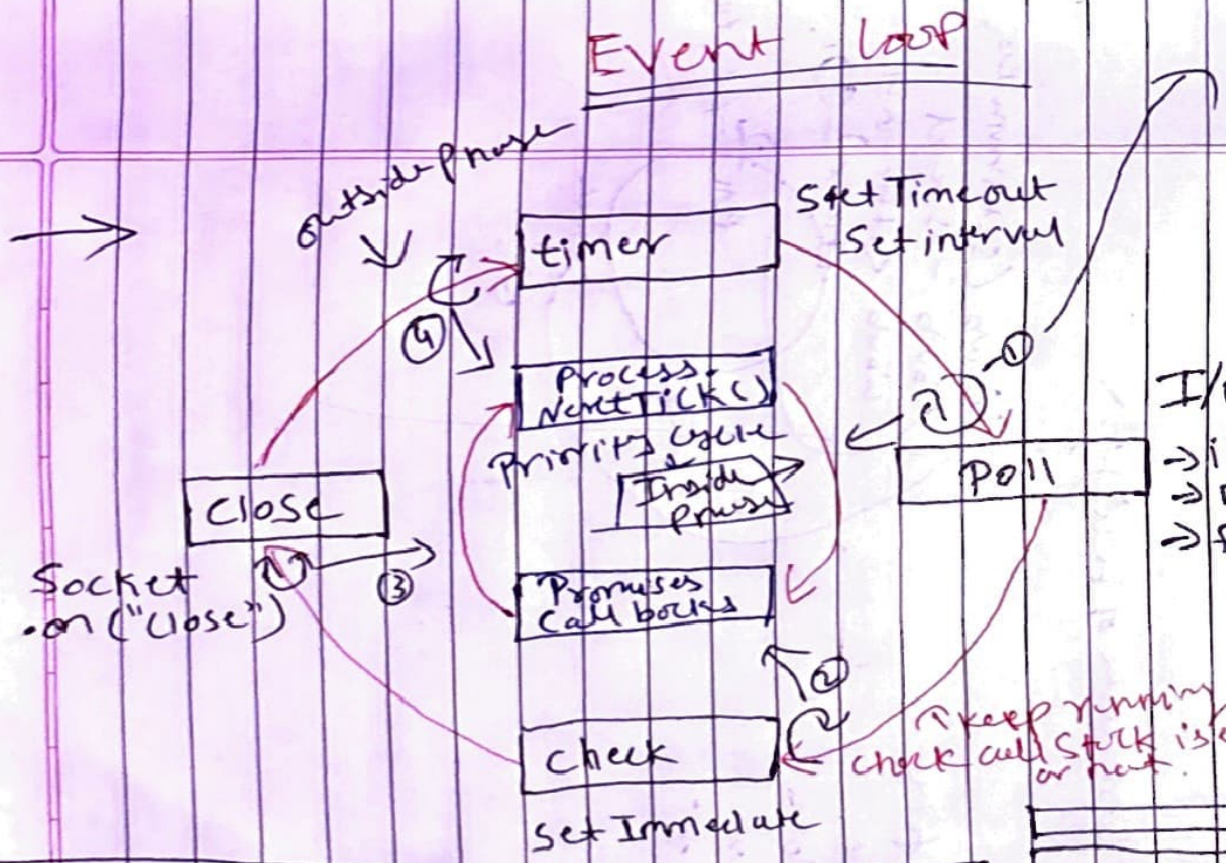
if there is so much cb are waiting the event loop will prioritize the task.

event loop check if the call stack is idel or empty.

Ex:-

main thread is busy
Call Stack is not idel or busy.
JS engine is busy.

Event Loop



There are many Phases in event loop
But main 4 Phases ① timer ② Poll ③ Check ④ Close

① Timer → all the timer (set timeout, Set interval) Suppose timer was expired and now it will wait call back. ② Priority

② Poll Phase → I/O call back executed in Poll Phase (most of callback executed in Poll Phase) (VI Phase)

③ Check Phase → Set immediate (cb) all the callback which are set with Set immediate will be get executed.

④ Close Phase → all the close or clean up things in close phase.

Before every Phase

it will run inside Process
① Process.NextTick ② Process
② Promise call back.

Before every outer Phase run it will run inside Phase first then. outside Phase will run.

First inside call back will get executed i.e Process.nextTick and Promises.
and then it will run outside Phase callback.

Date
Page 21

const fs = require("fs");
const a = 11
libuv ← Set immediate (() => console.log("Set immediate"));

fs.fileread

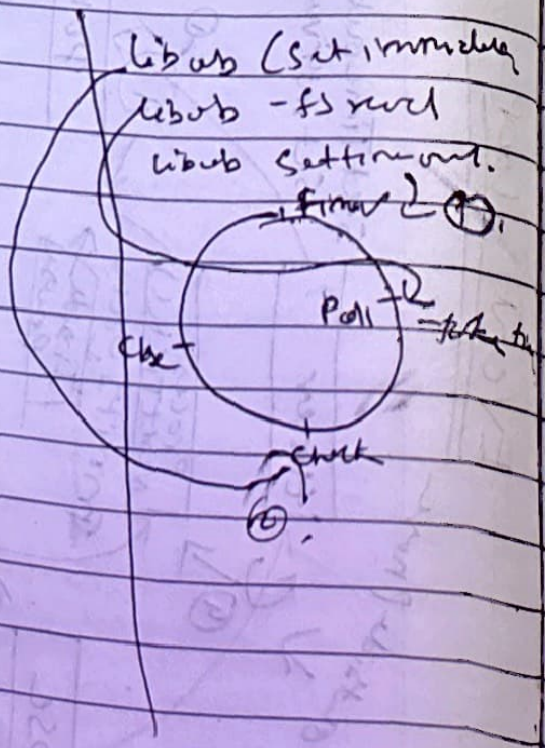
libuv ← fs.readFile("./file.txt", 'utf8', () => {
    console.log("file reading");
});

libuv ← setTimeout(() => { console.log("time expire"); }, 0);
(1)

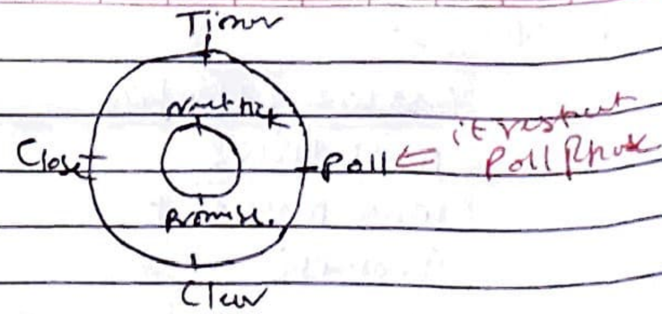
← function PrintA () {
    console.log("a", a);
}

call ← PrintA();
console.log("last line of file");

a 11
last line of file
time expire
Set immediate
file reading



Libuv -



What i expect

O/P	a=100	Sync code
	Last line of the file	
	Process.nextTick	
	Promise	
	Time expired	
	Set immediate	it depends the takes time
	File reading CB	

once callstack is empty
Event loop will goes to queue and
throw callback to the callback according priority.

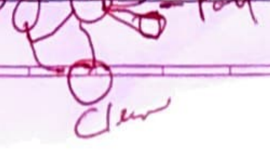
Que callback
eg.
- Set immediate
- Promise.resolve — P Queue
- fs read file (Priority queue)
- Settimeout
- Process.next tick — P queue

[Next tick, Promises] have priority queue.

Event loop waits on the poll Phase (Semi infinite loop)

Last line of code	Last line of code
~~Timer expired~~ Promise	next tick
timer expired	Promise
Set immediate	Time expired — didnt see
	Set immediate → file reading CB
	2nd next tick
	2nd timer
	2nd set immediate

Wrong Because event loop waits at poll phase → and it starts from plate phase →



what if the I/O

Last line of the file
next tick.
inner next tick.
Promise
Time expired
Set immediate.
file reading CB

we will go to Poll Phase but there is nothing because it is empty

file CB

what if

Last line of code. (1)
(3) inside next interval -> nextTick
data + file readed. (2)
ax2 value. (4) inner next tick (5) No timers to set timeout.
Promise.
time
time.

Last line of code.
null file readed.
inside next interval.
inner next tick.
1090
time.