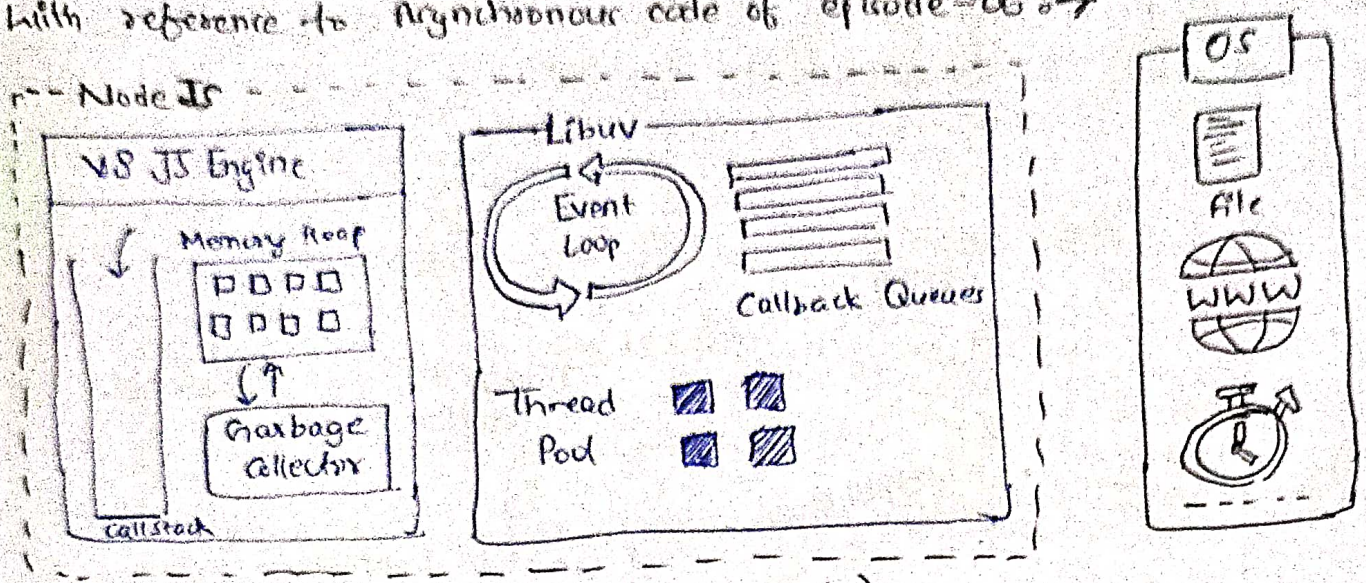
# Episode-09 → Libuv And Event Loop

When Asynchronous tasks are there, these are offloaded by Libuv to avoid blocking of main thread of V8 engine.

- NodeJS → Asynchronous I/O [Non-Blocking I/O] → mainly due to Libuv.

With reference to Asynchronous code of episode-06 :→




Asynchronous I/O (Non-Blocking I/O)

- When tasks are offloaded to Libuv, Libuv internally process a lot of things to complete this tasks
- Example: When file reading operation is there, Libuv makes call to operating system, gets the data back and then Libuv takes the callback function and give it to V8 engine
- Once Asynchronous task is completed, it is Libuv's job to take the callback function and send this to callstack.

## Callback Queues:

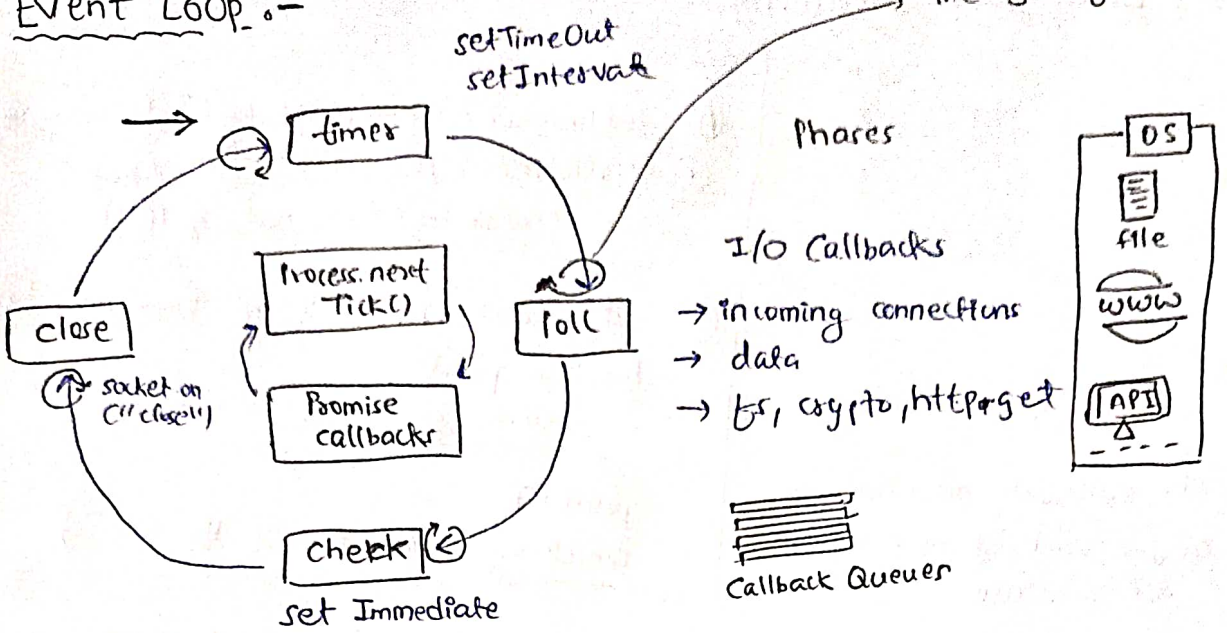
- When Asynchronous task is complete but call stack is not empty (due to large code), then the callback functions have to wait in their respective Callback Queue. There are separate Queue for API calls, timers etc.
- Something from callback Queue can only be pushed inside callstack when the callstack is empty.

## Event Loop:

- Event Loop keeps checking the callstack and callback Queue. As soon as the callstack is empty, one of the tasks from callback Queue will be pushed into callstack by event Loop.
- Job of event loop is to manage all the callback Queues and push this task into the callstack at correct time in correct order.

Call stack is empty ≡ V8 engine Idle ≡ main Thread Not Blocked → All these lingos mean same

# Event Loop :-




## # cycle of Event Loop (Prioritising of callbacks)

**Phase-1: Timer** all the callbacks from setTimeOut, setInterval will be pushed to callstack form callback Queue. and will be executed.

**Phase-2: Poll** all the I/O callbacks such as incoming connections, data and fs, crypto, http.get are executed in this phase.
Most of the callbacks are executed in this phase.

**Phase-3: Check** all the callbacks which are scheduled from setImmediate and are waiting in the callback Queue will be executed in this phase

**Phase-4: Close** all the close and cleanup operations callback are executed in this phase

**Note:** The cycle of process.nextTick() and promise callback will be executed before each phase. After executing this cycle only, the phase will begin.

All of these phases have their-own queues. These phases will begin only when the callstack is empty.

## #Example:

① process.nextTick (cb)
② Promise.resolve (cb)
③ setTimeOut (cb, 0)
④ setImmediate (cb)
⑤ fs.readfile ("file.txt", cb)
⑥ https.get ("URL", cb)

① & ② will be executed first after which in Timer phase, the cb for setTimeout will be executed Then process.nextTick and promise.resolve will be checked if their callbacks are waiting, since they were already executed, Then ⑤ & ⑥ will be executed in Poll phase, after ④'s callback ~~phase~~ will be executed And Nothing callback will be available in close phase.

# Find the Output { Event Loop Question }

```
const a=100;
(A) setImmediate(() => console.log("setImmediate"))
(C) fs.readFile(". /file.txt", "utf8", ()=>{
      console.log("File Reading CB")
    })
(B) setTimeOut(()=> console.log("Timer expired"), 0)
    function print(){
      console.log("a", a)
    }
    print();
    console.log("Last line of the file")
```

O/P

a = 100
Last line of the File
Timer expired
setImmediate
file Reading CB

Explanation :

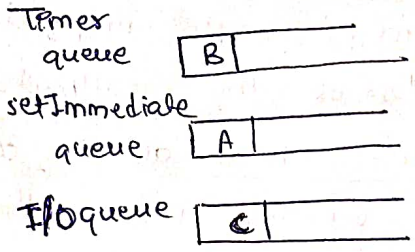
(A) is executed and its callback received to libuv is pushed in setImmediate callback queue by event loop.

(C) is offloaded to libuv which will read file data with the help of OS and get back the response.

(B) is also offloaded to libuv and since it has zero delay, it will be immediately pushed to callback queue (that is callback fn will be pushed to callback queue for Timer)

Now event loop working →

- In Timer phase, the event loop will find (B) in timer queue and execute the callback function as soon as callstack becomes empty.
- setImmediate callback is executed before the "fs.readFile" callback because I/O operation have not completed by the time event loop reaches the poll phase
- In next cycle's poll phase, the event loop will execute the callback of I/O queue (if completed/ready).



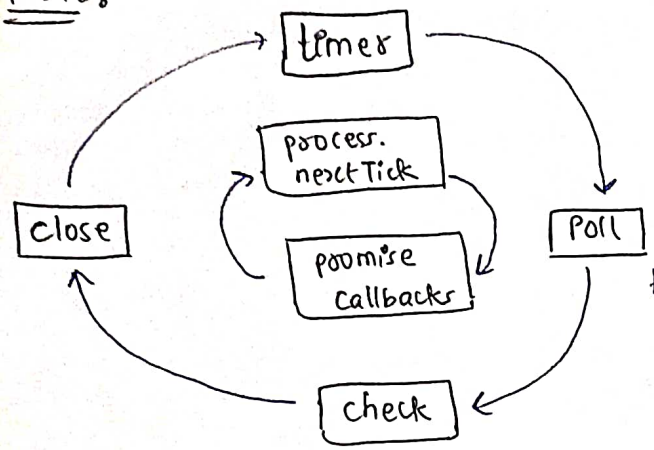

O/P

a = 100
Last line of the file
Process.nextTick()
Promise
Timer expired
setImmediate
fileReading CB

```
const a=100;
(A) setImmediate(()=> console.log("setImmediate"))
(B) Promise.resolve(()=> console.log("Promise"))
(C) fs.readFile(". /file.txt", "utf8", ()=>{
      console.log("fileReading CB")
    })
(D) setTimeOut(()=> console.log("Timeexpired"),
(E) process.nextTick(()=> console.log("process.nextTick"
(F) function printA(){
      console.log("a:", a)
    }
    printA()
    console.log("last line of the file")
```

Explanation → The o/p is same except that now, process.NextTick and Promise will be printed/executed before the phases like Timer, Poll, check, close even starts.

Note:




Event loop waits

When the callstack is empty and when the callback queue is empty, Then the event loop waits in the Poll phase that is when it has nothing to do.

Hence called semi infinite loop.

While in Browser, the event loop keeps running, never waits.

Find the O/p

Code:

```
setImmediate(() => console.log("setImmediate"));
setTimeout(() => console.log("Timer Expired"));
Promise.resolve(() => console.log("Promise"))
fs.readFile("./file.txt", "utf8", () => {
    setTimeout(() => console.log("2nd Timer"), 0)
    process.nextTick(() => console.log("2nd setImmediate"))
    console.log("File Reading CB")
})
process.nextTick(() => console.log("nextTick"))
console.log("Last line of the file")
```

O/p

```
Last line of the file
nextTick
Promise
Timer Expired
setImmediate
File Reading CB
2nd nextTick
2nd setImmediate
2nd Timer
```

Explanation → same as before until setImmediate is printed, Now when file reading operation completes and its callback is executed, we will be in Poll phase.

Within the Callback →
- synchronous code "File Reading CB" is logged.
- The second "setTimeout" is scheduled to run in the next Timers phase coz right now we are in poll phase
- process.nextTick is queued to run immediately after the callback before the check phase begins.
- In check phase, setImmediate callback is scheduled and executed.