$$\begin{tikzcd} & \text{C} \\ \text{JS} \arrow[r] \arrow[d] & \text{lang} \\ \text{Sync} & \end{tikzcd}$$

$$\begin{tikzcd}[column sep=small] & & \text{Libuv Library} \arrow[d] & & \\ & & \text{Powers async processing in nodejs} \arrow[dl] \arrow[dr] & & \\ & \text{Sync} & & \boxed{\text{Async}} \arrow[dll] \arrow[d] & \\ \text{thread pool} & & & \boxed{\text{event loop}} & \end{tikzcd}$$

Libuv provides a lot of imp functionalities to nodejs:

1) Networking capabilities → TCP, $\boxed{\text{DNS resolution}}$, UDP etc

nodejs → net, dns, dgram }

along with Libuv, C-ares is used

2) File IO & similar ops. → $\boxed{\text{fs}}$ ✓

3) A pool of worker threads to handle thread specific cpu tasks.

$\swarrow$ worker_threads lib.

$\hookrightarrow$ By default $\rightarrow$ # $\rightarrow$ 4

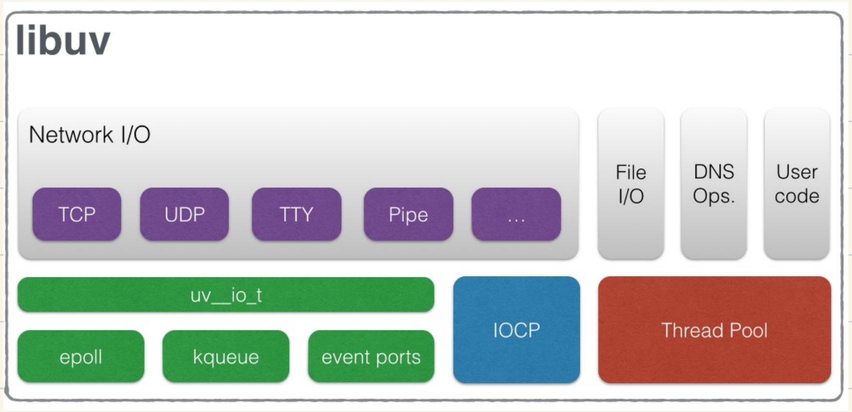
4) facilitates creation of child processes also

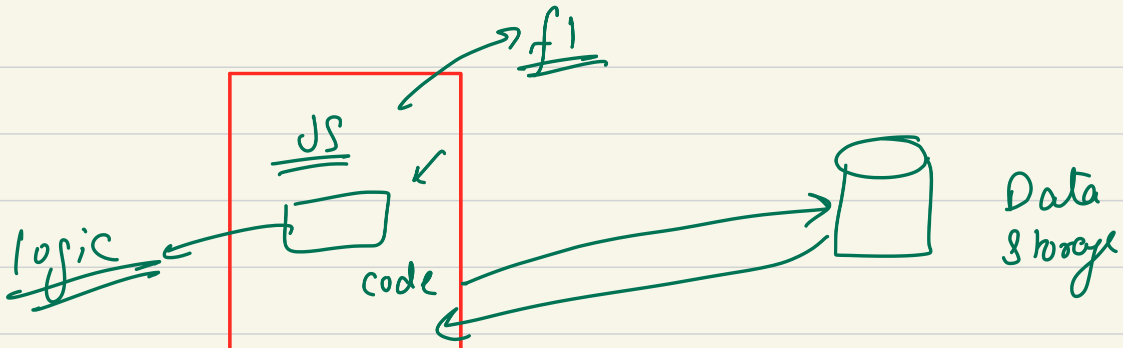
$\hookrightarrow$ child_process 2

VVI

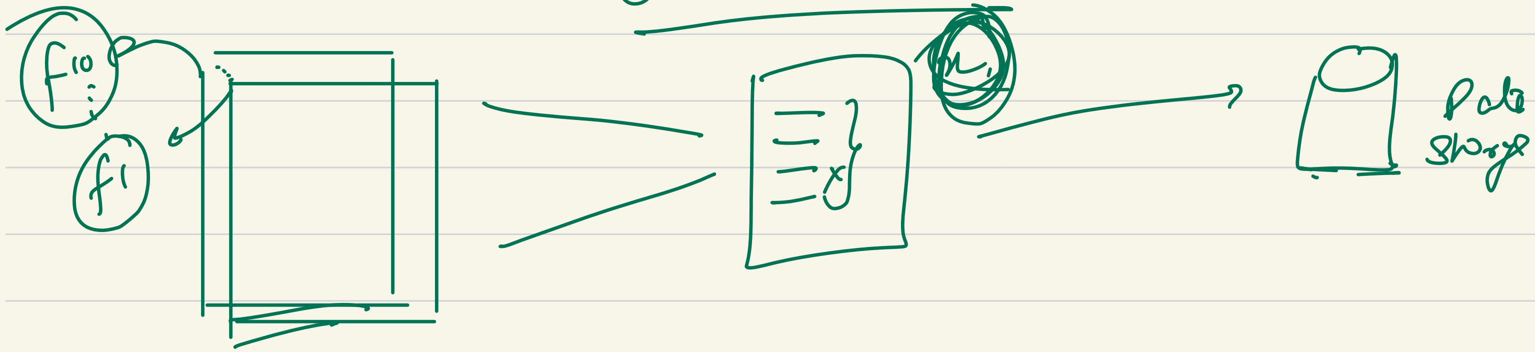
5) $\rightarrow$ fully functional event loop.

- $\rightarrow$ epoll $\rightarrow$ linux
- $\rightarrow$ kqueue $\rightarrow$ mac
- $\rightarrow$ iocp $\rightarrow$ windows


libuv
Network I/O
TCP
UDP
TTY
Pipe
...
File I/O
DNS Ops.
User code
uv__io_t
epoll
kqueue
event ports
IOCP
Thread Pool



SRP → single resp principle



Handles & Requests in libuv

handles in libuv

↳ long lived object used for async ops like TCP setup

↳ persistent object (remain in mem. until explicitly closed)

uv_close()

uv_loop_t
uv_tcp_t
uv_udp_t
⋮

} handle

Requests in libuv

↳ short lived objects (exist only till the duration of operation)

↳ reading or writing a file.

uv_write_t
uv_fs_t
⋮

$\left.\begin{array}{l} \texttt{uv\_write\_t} \\ \texttt{uv\_fs\_t} \\ \vdots \end{array}\right\}$

How the event loop in nodejs works?

Pre-requisite

- ↳ Promises ✓
- ↳ Timers ✓
- ↳ Ticks ✗
- ↳ phases of event loop ✗ }

Ticks → One full trip / iteration of event loop.

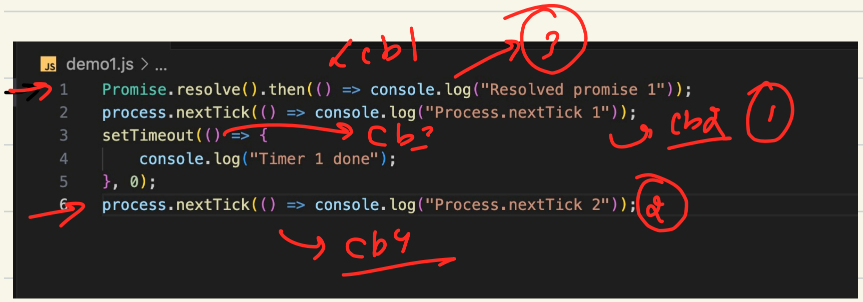
↓ Breakdown

a tick in nodejs refers to execution of a microtasks before the next iteration of the loop starts.

↓

nextTick queue / queue ← process.nextTick() → this func^n takes a callback,

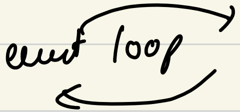
By invoking process.nextTick we instruct node to invoke this cb, at the end of current operation before the next Tick (trip/iteration) starts.

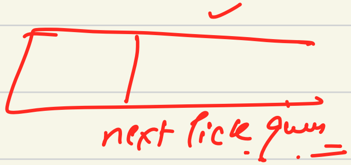


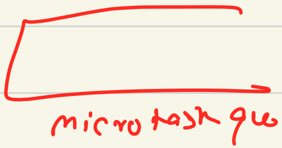
demo1.js > ...

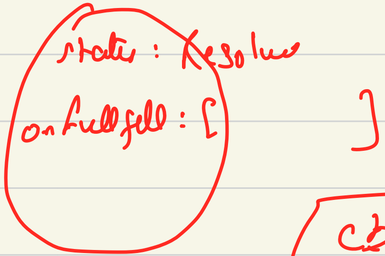
```javascript
Promise.resolve().then(() => console.log("Resolved promise 1"));
process.nextTick(() => console.log("Process.nextTick 1"));
setTimeout(() => {
    console.log("Timer 1 done");
}, 0);
process.nextTick(() => console.log("Process.nextTick 2"));
```

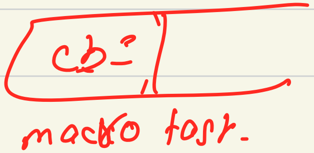
Timer → 0ms

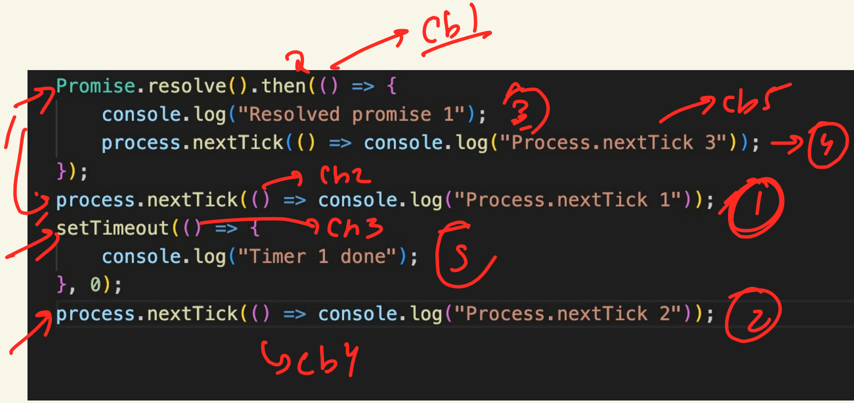




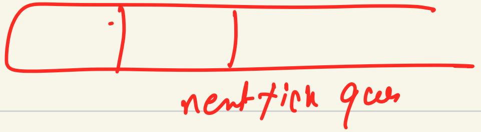


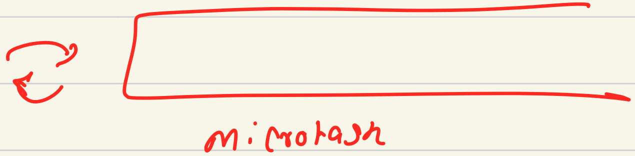


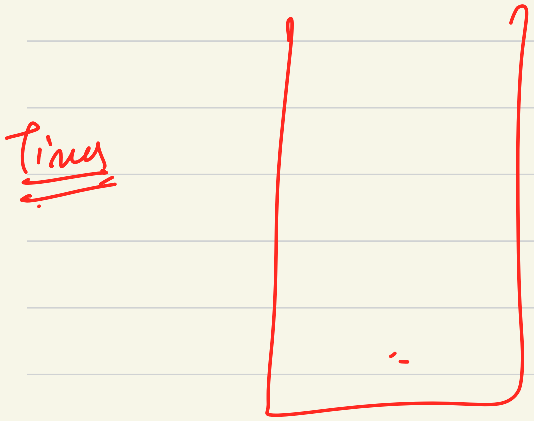


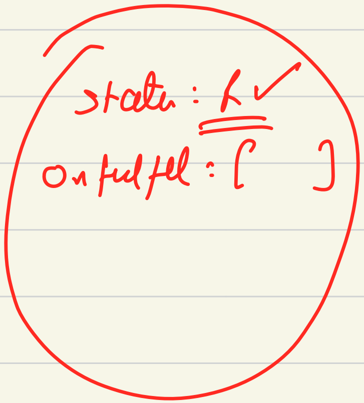


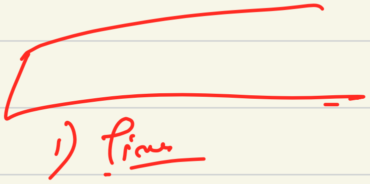
```
Promise.resolve().then(() => {
    console.log("Resolved promise 1");
    process.nextTick(() => console.log("Process.nextTick 3"));
});
process.nextTick(() => console.log("Process.nextTick 1"));
setTimeout(() => {
    console.log("Timer 1 done");
}, 0);
process.nextTick(() => console.log("Process.nextTick 2"));
```











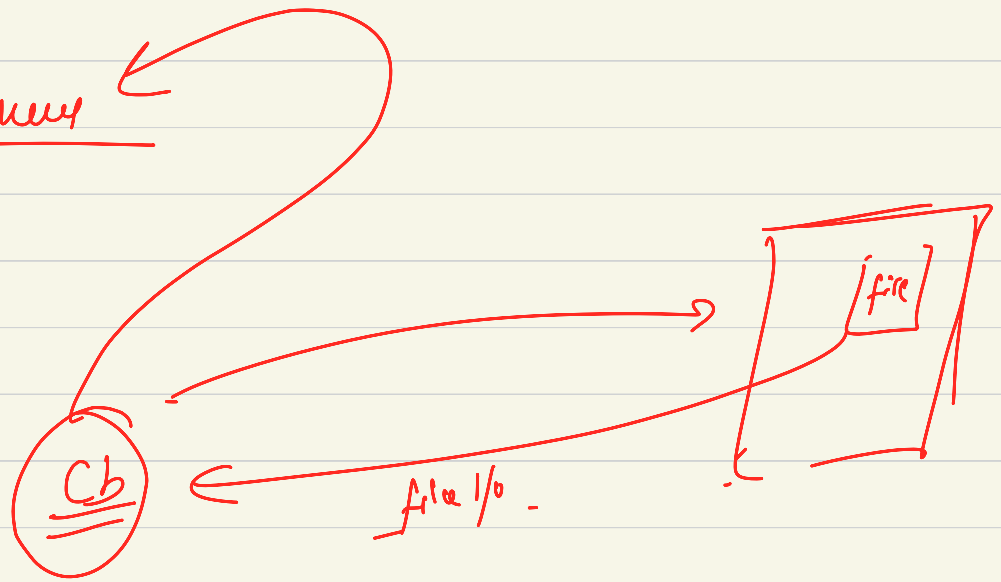
$\longrightarrow$ macrotask contains multiple queue $\rightarrow$

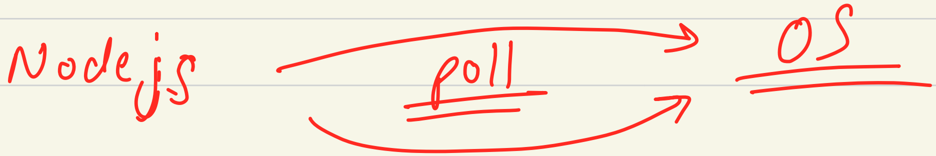
↳ working of macrotask is defined by phases of event loop.

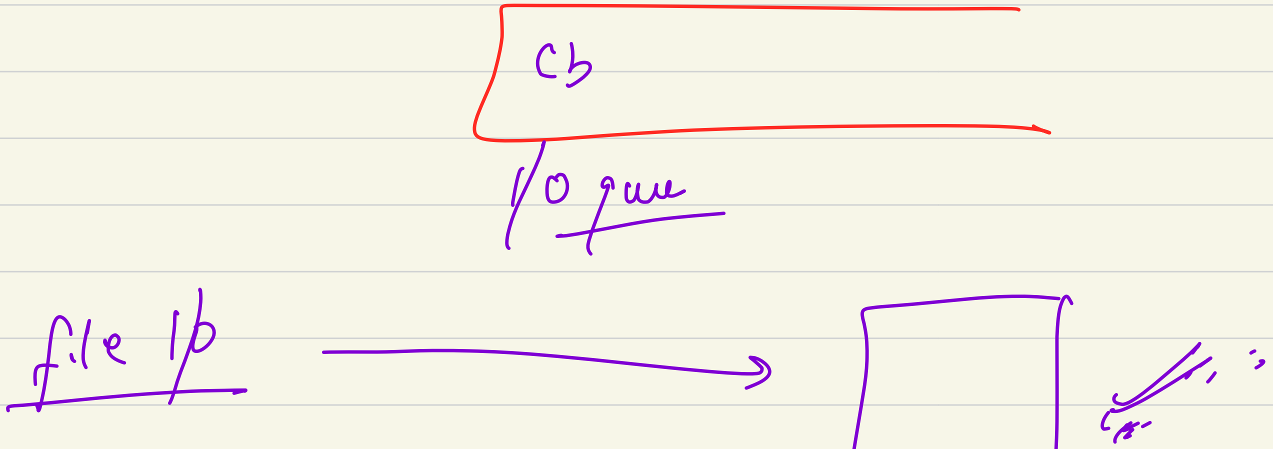
(1) Timer queue $\longleftarrow$ cb of setTimeout / setInterval
(min heap) $\rightarrow$

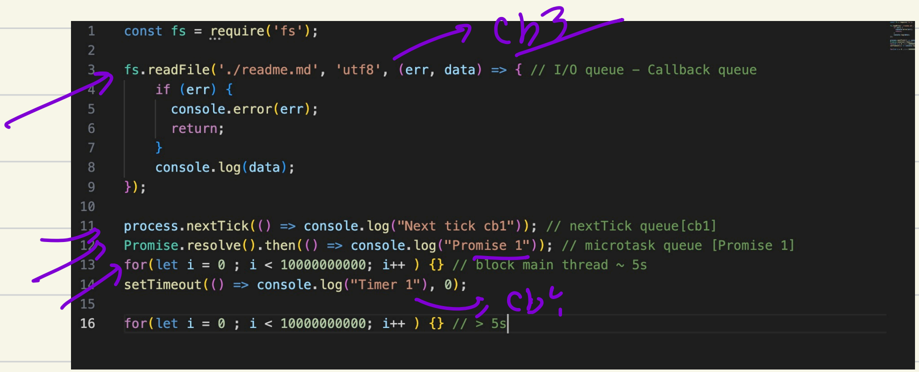
Note $\rightarrow$ 'Callbacks in microtask queue are executed after every timer callback in timer queue.' & even before microtask queue we execute the next tick queue cb.

2 I/O queue





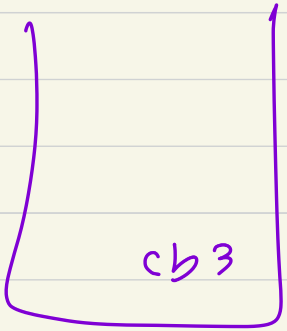

cb
I/O que
file I/p

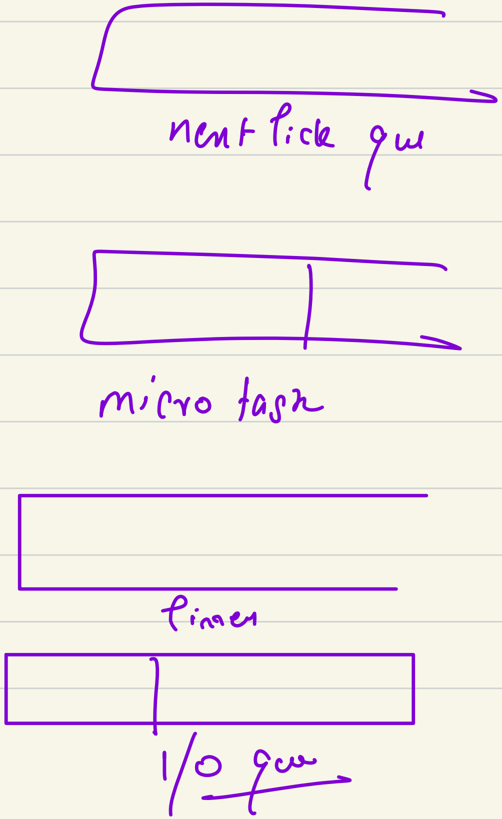


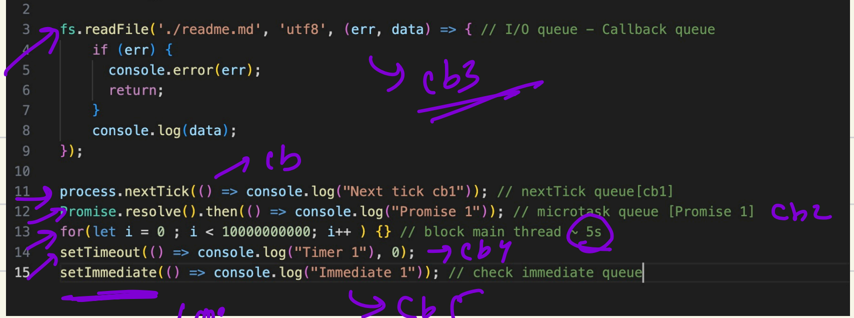
```javascript
const fs = require('fs');

fs.readFile('./readme.md', 'utf8', (err, data) => { // I/O queue - Callback queue
    if (err) {
      console.error(err);
      return;
    }
    console.log(data);
});

process.nextTick(() => console.log("Next tick cb1")); // nextTick queue[cb1]
Promise.resolve().then(() => console.log("Promise 1")); // microtask queue [Promise 1]
for(let i = 0 ; i < 10000000000; i++ ) {} // block main thread ~ 5s
setTimeout(() => console.log("Timer 1"), 0);

for(let i = 0 ; i < 10000000000; i++ ) {} // > 5s
```

Started file I/O

assume cb of I/O automatically gets in I/O queue







```
fs.readFile('./readme.md', 'utf8', (err, data) => { // I/O queue - Callback queue
    if (err) {
      console.error(err);
      return;
    }
    console.log(data);
});

process.nextTick(() => console.log("Next tick cb1")); // nextTick queue[cb1]
Promise.resolve().then(() => console.log("Promise 1")); // microtask queue [Promise 1]
for(let i = 0 ; i < 10000000000; i++ ) {} // block main thread ~ 5s
setTimeout(() => console.log("Timer 1"), 0);
setImmediate(() => console.log("Immediate 1")); // check immediate queue
```

file IO ✓

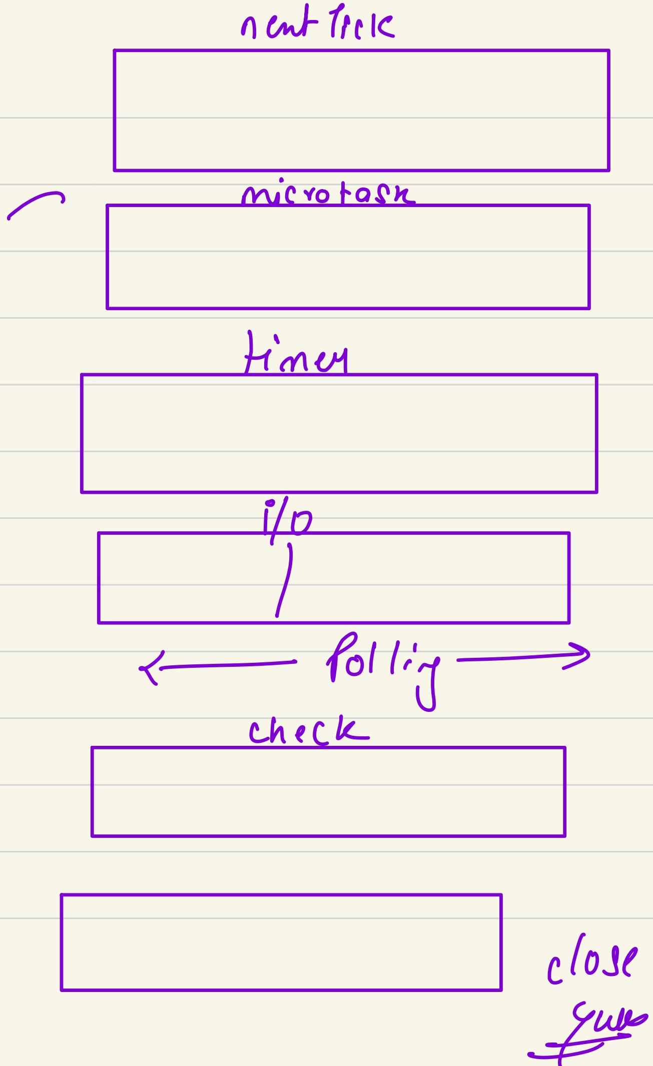
cb1 ✓

cb2 ✓

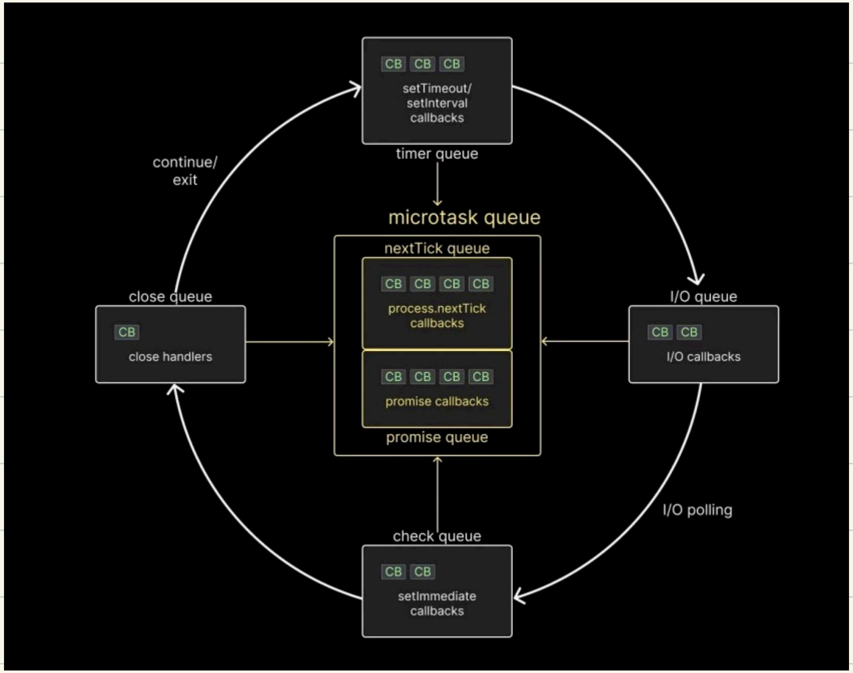
cb4 ✓

cb5 ✓

cb3 ✓



Event loop
Nodejs


CB CB CB
setTimeout/ setInterval callbacks
timer queue
continue/ exit
microtask queue
nextTick queue
CB CB CB CB
process.nextTick callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
I/O queue
CB CB
I/O callbacks
I/O polling
check queue
CB CB
setImmediate callbacks