THE EVENT LOOP

libuv contains the event loop.

https://nodejs.org/en/guides/event-loop-timers-and-nexttick/

event loop → (allows) non-blocking I/O operations even though JS is single threaded

① tick. process.nextTick()

1 full iteration of event loop is a tick.
trip.

nodejs → libuv → event loop initializa^

Phases Overview

- **timers**: this phase executes callbacks scheduled by `setTimeout()` and `setInterval()`.
- **pending callbacks**: executes I/O callbacks deferred to the next loop iteration.
- **idle, prepare**: only used internally.
- **poll**: retrieve new I/O events; execute I/O related callbacks (almost all with the exception of close callbacks, the ones scheduled by timers, and `setImmediate()`); node will block here when appropriate.
- **check**: `setImmediate()` callbacks are invoked here.
- **close callbacks**: some close callbacks, e.g. `socket.on('close', ...)`.

Between each run of the event loop, Node.js checks if it is waiting for any asynchronous I/O or timers and shuts down cleanly if there are not any.

global object

process require module .... etc.

different properties to access env variables

__dirname (PWD).

env object is available in process variable.

demo1.js > ...

```
1  Promise.resolve().then(() ⇒ console.log("Printing from promise"));
2  process.nextTick(() ⇒ console.log("printing from nextTick"));
3  setTimeout(() ⇒ console.log("printing from timer"), 0);
```

```
printing from nextTick
Printing from promise
printing from timer
```

nextTick takes a callback. By calling process.nextTick we instruct node to invoke this cb fn at end of current operation before the next tick starts.

```
let a = 10;
Promise.resolve().then(() ⇒ console.log("Printing from promise"));
process.nextTick(() ⇒ console.log("printing from nextTick"));
setTimeout(() ⇒ console.log("printing from timer"), 0);
console.log(a);
```

```
10
printing from nextTick
Printing from promise
printing from timer
```

1st thing that happens → execn of nextTick callback.

multiple nextTick execute in order. Because all cb are added 1 by 1 in the nextTick queue.

For execn of next tick cb callstack must be empty. (after the current operation)

→ whatever cb are there in next tick queue, they are executed 1 by 1 (the whole Q is cleared.)

priority

nextTick q > micro task q > callback q.

No provision of hindering a blocking piece of code in JS.

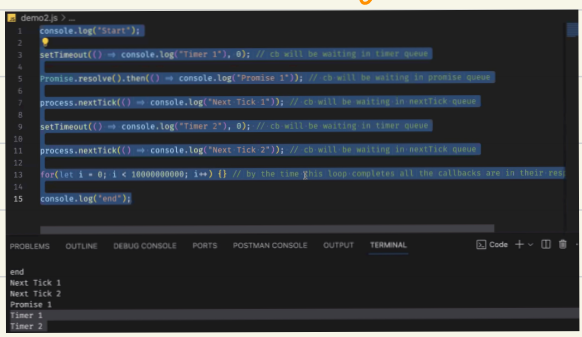
→ 1st itern of event loop starts when current set of instrns are done (main thread done) & next tick Q is empty. then next itern of event loop starts

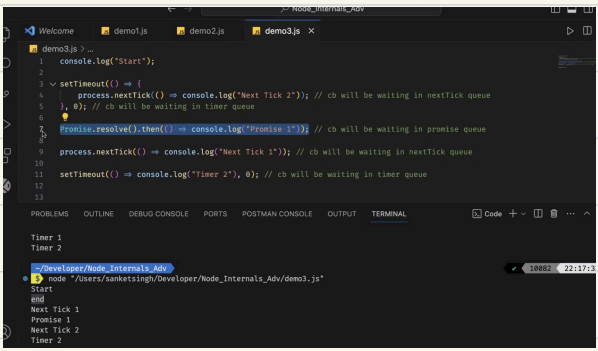
① next tick Q will be cleared

② Event loop tick (iteration starts)

③ Microtask Q / promise Q gets cleared.

④ Timers Q (contains all callbacks of setTimeout/setInterval) start executing. ref phases image.





as 1 callback is executed from timer Q the event loop checks the next tick Q again.

① clear next tick Q

② clear promise Q

③ call cb in timer Q 1by1 but after every cb execution go to step ①.

④ Pending callbacks → callback Q / i/o Q

Any kind of cb apart from timers go to the i/o Q.

poll phase

the moment file reading is done the cb is pushed to the callback Q. Does not happen.

When file reading happens. I/O polling happens.

in I/O polling you check from the OS & you poll it if the task is done.

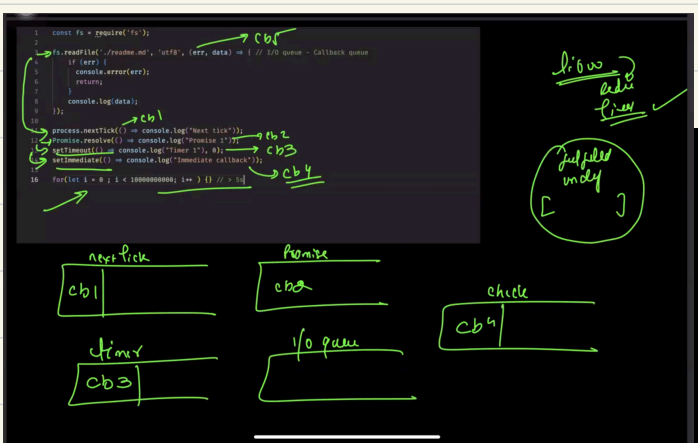
event loop polls the OS to check if pending I/O operations are done.

next tick → promise → timer → i/o Q → polling → check Q (cb of setImmediate only)

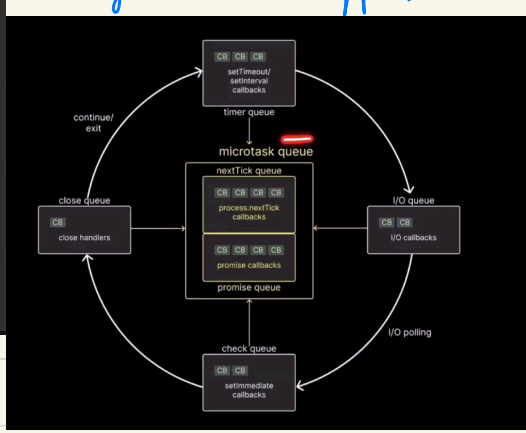
```
Welcome | demo1.js | demo2.js | demo3.js | demo4.js | demo5.js
demo5.js > ...
1  const fs = require('fs');
2
3  fs.readFile('./readme.md', 'utf8', (err, data) => { // I/O queue - Callback queue
4      if (err) {
5        console.error(err);
6        return;
7      }
8      console.log(data);
9  });
10
11 process.nextTick(() => console.log("Next tick"));
12 Promise.resolve(() => console.log("Promise 1"));   add .then
13 setTimeout(() => console.log("Timer 1"), 0);
14 setImmediate(() => console.log("Immediate callback"));
15
16 for(let i = 0 ; i < 10000000000; i++ ) {} // > 5s

PROBLEMS  OUTLINE  DEBUG CONSOLE  PORTS  POSTMAN CONSOLE  OUTPUT  TERMINAL

~/Developer/Node_Internals_Adv
$ node "/Users/sanketsingh/Developer/Node_Internals_Adv/demo5.js"
Next tick
Timer 1
Immediate callback
Please donot touch me

~/Developer/Node_Internals_Adv
$
```

O/P

```
$ node "/Users/sanketsingh
Next tick
Promise 1
Timer 1
Immediate callback
Please donot touch me
```



for cb 5 to go into i/o Q polling needs to happen



There is space for context switching when process interacts with the OS.
So whenever a process waits for the OS a context switching situation might occur

ex. fs read & timer with 0 ms. (since I/O polling is involved)