Asynchronous Code Execution in Node.js

```
1 console.log('First')
2 fs.readFile(__filename, () => {
3   console.log('Second')
4 })
5 console.log('Third')
```

Console

First

V8 engine

Memory heap

Call stack

global()

libuv

```
()⇒{
  console.log("second")
}
```

Few Questions

Whenever an async task completes in libuv, at what point does Node decide to run the associated callback function on the call stack?

What about async methods like setTimeout and setInterval which also delay the execution of a callback function?

If two async tasks such as setTimeout and readFile complete at the same time, how does Node decide which callback function to run first on the call stack?

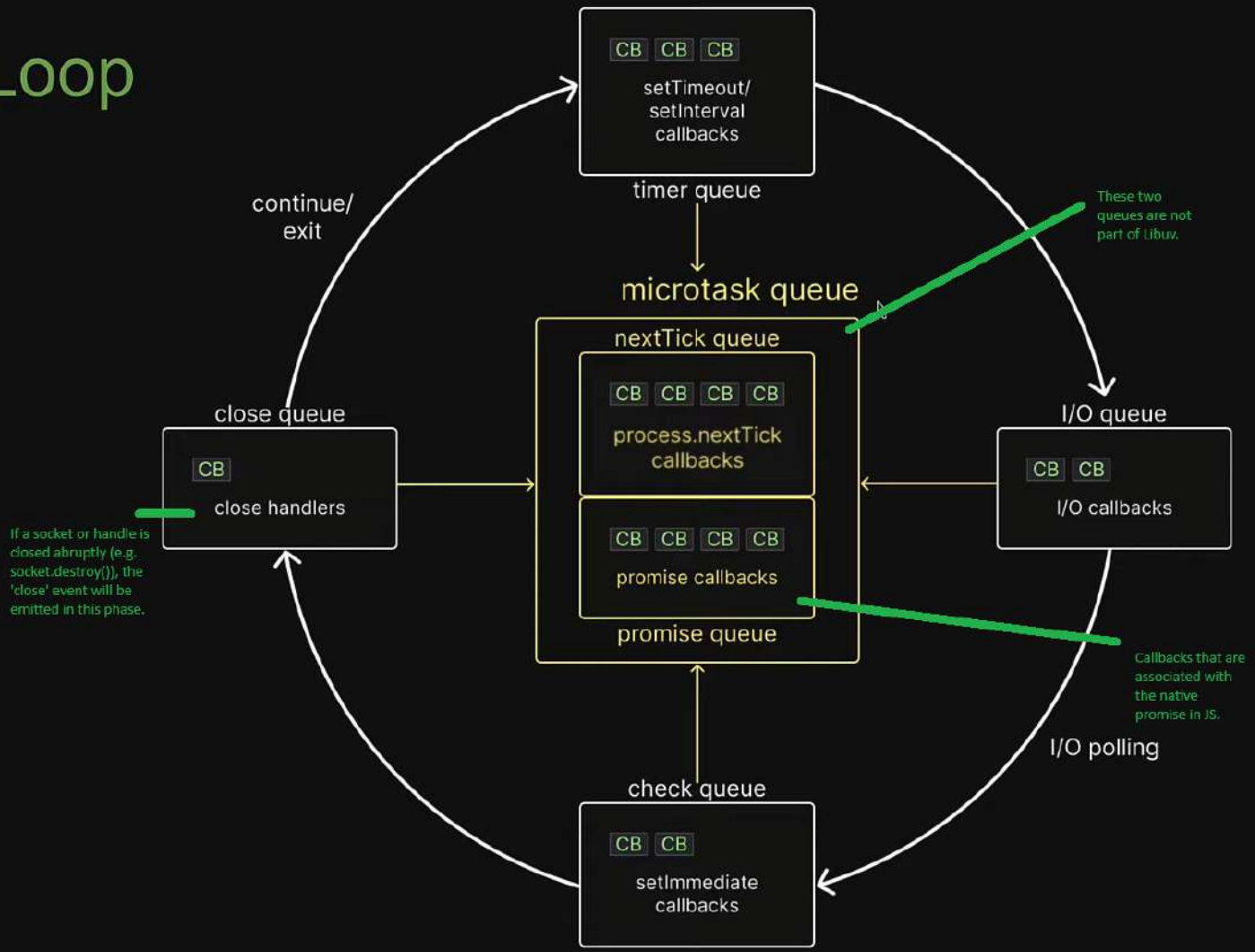
Event Loop

It is a C program and is part of libuv

A design pattern that orchestrates or co-ordinates the execution of synchronous and asynchronous code in Node.js

1. Visual representation
2. Run a few experiments

Event Loop


CB CB CB
setTimeout/
setInterval
callbacks
timer queue
continue/
exit
These two
queues are not
part of Libuv.
microtask queue
nextTick queue
CB CB CB CB
process.nextTick
callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
If a socket or handle is
closed abruptly (e.g.
socket.destroy()), the
'close' event will be
emitted in this phase.
I/O queue
CB CB
I/O callbacks
Callbacks that are
associated with
the native
promise in JS.
I/O polling
check queue
CB CB
setImmediate
callbacks

Event Loop - Execution Order

1. Any callbacks in the micro task queues are executed. First, tasks in the nextTick queue and only then tasks in the promise queue
2. All callbacks within the timer queue are executed
3. Callbacks in the micro task queues if present are executed. Again, first tasks in the nextTick queue and then tasks in the promise queue
4. All callbacks within the I/O queue are executed
5. Callbacks in the micro task queues if present are executed. nextTick queue followed by Promise queue.
6. All callbacks in the check queue are executed
7. Callbacks in the micro task queues if present are executed. Again, first tasks in the nextTick queue and then tasks in the promise queue
8. All callbacks in the close queue are executed
9. For one final time in the same loop, the micro task queues are executed. nextTick queue followed by promise queue.

Event Loop - Execution Order

If there are more callbacks to be processed, the loop is kept alive for one more run and the same steps are repeated

On the other hand, if all callbacks are executed and there is no more code to process, the event loop exits.

Few Questions

Whenever an async task completes in libuv, at what point does Node decide to run the associated callback function on the call stack?
Callback functions are executed only when the call stack is empty. The normal flow of execution fill not be interrupted to run a callback function

What about async methods like setTimeout and setInterval which also delay the execution of a callback function?
setTimeout and setInterval callbacks are given first priority

If two async tasks such as setTimeout and readFile complete at the same time, how does Node decide which callback function to run first on the call stack?
Timer callbacks are executed before I/O callbacks even if both are ready at the exact same time