Callback Execution Priority in Node.js

Node.js runs callbacks in a specific order, based on **event loop phases** and **task queues**. The main ones to know are:

1️⃣ `process.nextTick()` Queue

- **Highest priority** — runs **immediately after the current operation finishes**, before the event loop moves to the next phase.
- Even higher priority than Promises (microtasks).
- Overusing it can block the event loop.

2️⃣ Microtask Queue (`Promise.then()`, `queueMicrotask`)

- Runs right after `process.nextTick()` and before any timers or I/O callbacks.
- Example:

Js > Copy

```js
Promise.resolve().then(() => console.log('Promise microtask'));
```

3️⃣ Timers Phase (`setTimeout`, `setInterval`)

- Executes callbacks scheduled by timers **if their delay has expired**.
- Example:

Js > Copy

```js
setTimeout(() => console.log('timeout'), 0);
```

4️⃣ I/O Callbacks Phase

- Handles completed I/O operations (like file reads, network responses).

5️⃣ Check Phase (`setImmediate`)

- Runs after I/O callbacks.
- If `setTimeout(..., 0)` and `setImmediate()` are scheduled in the same tick, `setImmediate()` usually runs **after** the timeout.

6️⃣ Close Callbacks

- Runs cleanup callbacks like `socket.on('close', ...)`.

🔍 Example to See the Order

Js Copy

```js
console.log('start');

process.nextTick(() => console.log('nextTick'));
Promise.resolve().then(() => console.log('promise'));
setTimeout(() => console.log('timeout'), 0);
setImmediate(() => console.log('immediate'));

console.log('end');
```

Expected output:

Code Copy

```
start
end
nextTick
promise
timeout
immediate
```

✅ **Summary of Priority:**
`process.nextTick()` → Microtasks (Promises) → Timers → I/O → `setImmediate()` → Close callbacks.