Node.js primarily manages its asynchronous operations through several types of callback queues, which are part of its event loop mechanism. The main callback queues are:

Next Tick Queue: This queue is for callbacks scheduled with process.nextTick(). These callbacks are executed after the current operation completes and before the event loop continues to the next phase. They have the highest priority among all callbacks.

```
process.nextTick(() => console.log('nextTick callback'));
```

Microtask Queue: This queue is for promise callbacks and other microtasks (like async/await). These are executed after the next tick queue is emptied and before the event loop continues to the next phase. Microtasks are executed in the order they were added.

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

Timers Queue: This queue is for callbacks scheduled with setTimeout() and setInterval(). The event loop checks this queue after completing the microtask queue, executing callbacks that have reached their specified time.

```
setTimeout(() => console.log('setTimeout callback'), 1000);
```

I/O Callbacks Queue: This queue is for callbacks related to I/O operations, such as file system operations, network calls, etc. These callbacks are executed after the timers queue and are meant for most types of I/O events.

```
const fs = require('fs');
fs.readFile('file.txt', () => console.log('File read callback'));
```

SetImmediate Queue: This queue is for callbacks scheduled with setImmediate(). These callbacks are executed after I/O callbacks have been processed in the current cycle of the event loop. It's designed to execute callbacks as soon as the event loop is idle.

```
setImmediate(() => console.log('setImmediate callback'));
```

Close Callbacks Queue: This queue is for callbacks related to closing events, such as closing a server or a socket. These are executed after the setImmediate queue.

```
const server = require('http').createServer();
server.on('close', () => console.log('Server close callback'));
server.close();
```

Example:

```
1  console.log('start');
2
3  process.nextTick(() => console.log('nextTick callback'));
4
5  Promise.resolve().then(() => console.log('Promise callback'));
6
7  setTimeout(() => console.log('setTimeout callback'), 0);
8
9  setImmediate(() => console.log('setImmediate callback'));
10
11 console.log('end');
```

Output Order:

```
1  start
2  end
3  nextTick callback
4  Promise callback
5  setTimeout callback
6  setImmediate callback
```

1. start and end are logged immediately.
2. nextTick callback is executed next because process.nextTick() callbacks have the highest priority.
3. Promise callback is executed after all next tick callbacks, as part of the microtask queue.
4. setTimeout callback and setImmediate callback are somewhat interchangeable depending on the phase of the event loop when they are scheduled. However, in this

scenario, the setTimeout callback is likely to execute before the setImmediate callback because the timer has a delay of 0 ms, and it's scheduled before setImmediate in the code.

5. This demonstrates how different types of asynchronous operations are managed in Node.js through various queues, ensuring a structured and predictable execution order based on the event loop phases.