**Order of Task Execution**

1. **Synchronous Code**
   * Runs immediately, line by line.
2. **Microtasks**
   * **In Node.js:**
     + process.nextTick (Node.js-specific, runs before other microtasks)
     + Promise handlers (.then, .catch, .finally)
   * **In Browsers:**
     + Only Promise handlers (no process.nextTick)
3. **Macrotasks**
   * Executed in event loop phases. Examples:
     + setTimeout
     + setInterval
     + setImmediate (Node.js only)
     + I/O callbacks
4. **Rendering/Other Tasks** (Browsers only)
   * Browser may update UI (not applicable in Node.js)

**Typical Node.js Event Loop Phase Order**

1. **Timers** (setTimeout/setInterval)
2. **Pending Callbacks**
3. **Idle, Prepare** (internal)
4. **Poll** (I/O callbacks)
5. **Check** (setImmediate)
6. **Close Callbacks** (e.g., socket.on('close', ...))

**After each phase**, the microtask queue is emptied (all microtasks are run before proceeding to the next phase).

**Summary Table**

| **Priority** | **Type** | **Examples** |
| --- | --- | --- |
| 1 | Synchronous | console.log('a') |
| 2 | Microtask | process.nextTick, Promises |
| 3 | Macrotask | setTimeout, setImmediate, I/O |
| 4 | Rendering | Browser paint (not Node.js) |

**Full Order in Node.js**

During each event loop tick:

1. Run all synchronous code.
2. Run all process.nextTick callbacks (microtasks, Node.js only).
3. Run all Promise microtasks.
4. Run macrotasks for the current phase.
5. After each phase, run all queued microtasks again.
6. Repeat phases until event loop is empty.

**In short:**

* Synchronous code → process.nextTick → Promises → Macrotasks (setTimeout, setImmediate, I/O, etc.)
* Microtasks always run to completion before moving to the next macrotask/event loop phase.

**Buffers**# Node.js Buffers – What, Why, and How

## 1. What is a Buffer?

A \*\*Buffer\*\* in Node.js is a special type of object for dealing with raw binary data directly.

- Unlike strings (which are sequences of characters), buffers are sequences of bytes.

- Buffers are used primarily for reading and manipulating binary data, which is common when working with files, network streams, or cryptography.

---

## 2. Why do Buffers Exist in Node.js?

### a. JavaScript Strings Are Not Binary-Safe

- JavaScript strings are for textual (character) data, and can't hold arbitrary binary data (like images, audio, or custom file formats) safely.

- You may lose or corrupt data if you try to treat all binary data as strings.

### b. Node.js Is Designed for I/O (Input/Output) Tasks

- Node.js frequently deals with binary data: reading files, sending/receiving data over networks (TCP/UDP), etc.

- Buffers provide a way to read, write, and manipulate this data efficiently.

### c. Performance

- Buffers are much faster for binary data manipulation compared to strings or arrays.

---

## 3. How Are Buffers Used?

### a. Creating Buffers

```js

const buf = Buffer.alloc(10); // allocates 10 zero-filled bytes

const bufFromStr = Buffer.from('hello'); // creates a buffer from a string

const bufFromArr = Buffer.from([1, 2, 3]); // from array of bytes

```

### b. Reading and Writing

```js

buf.write('Node.js'); // writes string as bytes into buffer

console.log(buf.toString()); // converts buffer back to string (UTF-8)

console.log(buf[0]); // reads a byte (decimal value)

```

### c. Slicing, Copying, and Concatenating

```js

const sliced = bufFromStr.slice(0, 2); // slice first 2 bytes

const combined = Buffer.concat([buf, bufFromStr]); // concatenate buffers

```

---

## 4. Where Will You See Buffers?

- \*\*File I/O:\*\* Reading or writing files (images, PDFs, videos, etc.)

- \*\*Network Streams:\*\* TCP/UDP sockets, HTTP requests/responses

- \*\*Cryptography:\*\* Hashes, encryption, etc.

- \*\*Working with data from external sources:\*\* APIs, devices, etc.

---

## 5. Key Buffer Methods & Properties

| Method/Property | Description |

|---------------------|-------------------------------------------------|

| Buffer.alloc(size) | Creates a zero-filled buffer |

| Buffer.from(x) | Creates buffer from string/array/another buffer |

| buffer.write(str) | Writes string to buffer |

| buffer.toString() | Converts buffer to string |

| buffer.length | Number of bytes in the buffer |

| buffer.slice(a, b) | Returns a sub-buffer |

| Buffer.concat([...])| Concatenates buffers |

| buffer.toJSON() | Converts buffer to JSON |

---

## 6. Summary: Why Are Buffers Needed?

- \*\*Buffers\*\* are essential for working with binary data in Node.js.

- They ensure data is handled safely and efficiently, especially for non-text files and network operations.

- If you’re building anything that interacts with files, servers, or raw protocols, you’ll use buffers.

---

\*\*Learn more:\*\* [Node.js Buffer API Documentation](https://nodejs.org/api/buffer.html)