

Chats

React Router SPA Flow

Async Code Execution Order

React State Change Flow

Open Close Expander Console

Debugging React Re-renders

JS Event Loop Breakdown

RADIO Framework for Frontend ...

Social Media Feed React

Robust Browser Detection

Logical Assignment Operators

Browser Detection Comparison

Array Key-Value Pairs

ChatGPT

Share

Explain how the JS event loop handles microtasks vs macrotasks with a real-world async example

Great — let's break down how the **JavaScript event loop** handles **microtasks vs macrotasks** using a **real-world async example** that includes:

- `setTimeout()` → macrotask
- `Promise.then()` → microtask
- `async/await` → microtasks internally
- `console.log()` → sync code

Ask anything

- Call Stack:** Where functions are executed.
- Task Queue (Macrotasks):** Things like `setTimeout`, `setInterval`, `I/O`, etc.
- Microtask Queue:** Things like `Promise.then`, `queueMicrotask`, `MutationObserver`, `async/await`.

👉 Microtasks run immediately after the current execution context completes, before any macrotask.

Example: Real-World Async Scenario

js

Copy Edit

Ask anything

```
setTimeout(() => {
  console.log("🕒 Timeout - macrotask");
}, 0);

Promise.resolve().then(() => {
  console.log("✅ Promise.then - microtask 1");
});

queueMicrotask(() => {
  console.log("🚀 queueMicrotask - microtask 2");
});

async function fetchData() {
  console.log("🔄 fetchData end - microtask 3");
}

fetchData();

console.log("🏁 App End");
```

Ask anything

Event Loop Execution Order

1. Call stack runs top-to-bottom:

Ask anything

```
"🟢 App Start"
"🚀 fetchData start"
"🏁 App End"
```

2. Microtasks queue flushes (in order):

arduino

Copy Edit

```
" Promise.then - microtask 1"
" queueMicrotask - microtask 2"
" fetchData end - microtask 3"
```

3. Macrotasks (from task queue):

Ask anything

```
" Timeout - macrotask"
```

Final Output:

sql

Copy Edit

```
App Start
fetchData start
App End
Promise.then - microtask 1
```

Ask anything

```
Timeout - macrotask
```

Key Takeaways

Task Type	Examples	Priority	When It Runs
Microtask	<code>Promise.then</code> , <code>queueMicrotask</code>	▲ High	Right after current stack execution
Macrotask	<code>setTimeout</code> , <code>setInterval</code>	▼ Lower	After microtasks and rendering

Ask anything

- Macrotasks may wait for multiple microtasks to complete first.

? Want deeper: browser rendering timing, or node.js-specific event loop phases?

Let me know how deep you want to go — I can tailor for browser vs Node.js too.

📄 🌟 🔊 🔍 ↻ ⌂

Ask anything

Perfect. Let's now **dive deeper** into how the **JavaScript event loop** behaves differently in:

Browser vs Node.js Event Loop

Although the **core concept** (call stack, microtasks, macrotasks) is the same, there are key differences in **timing**, **phases**, and **task queues**.

* Priority Order per cycle:

1. **Execute all microtasks** (resolve all `Promise.then`, `await`, etc.)
2. **Render if needed** (paint to the screen)
3. **Run one macrotask** (`setTimeout`, `setInterval`, `fetch`, etc.)

4. Repeat

📦 Browser Task Queues:

- **Macrotask queue:** `setTimeout`, `setInterval`, `postMessage`, `setImmediate`
- **Microtask queue:** `Promise.then`, `await`, `queueMicrotask`, `MutationObserver`

Ask anything

After each macrotask, the browser clears the microtask queue before the next render of macrotask.

⚙️ 2. Node.js Event Loop (Phased & More Complex)

Node.js event loop has 6 distinct phases, processed in order per tick:

Phase	What Happens
1. Timers	Runs callbacks from <code>setTimeout</code> , <code>setInterval</code>
2. Pending Callbacks	I/O error calls (e.g., TCP errors)
3. Poll	Retrieves new I/O events, executes I/O-related callbacks
5. Check	Executes <code>setImmediate()</code> callbacks
6. Close Callbacks	E.g., <code>socket.on('close', ...)</code>

📦 Microtasks:

- Run **between phases**, after each callback
- Includes `Promise.then`, `process.nextTick()` (higher priority!)

⚠️ Special Node.js Behavior:

- `process.nextTick()` runs **before all microtasks** (even before `Promise.then`)

Ask anything

👤 Real-World Node.js Code Example

```
js
setTimeout(() => {
  console.log("🕒 setTimeout");
}, 0);

setImmediate(() => {
  console.log("⚡ setImmediate");
});

Promise.resolve().then(() => {
  console.log("✅ Promise.then");
});

process.nextTick(() => {
  console.log("📦 process.nextTick");
});
```

📄 Output in Node.js:

```
arduino
📦 process.nextTick
✅ Promise.then
```

📌 Summary Table

Environment	Microtasks	Macrotasks	Unique Notes
Browser	<code>Promise.then</code> , <code>await</code>	<code>setTimeout</code> , <code>setInterval</code>	Microtasks before render
Node.js	<code>Promise.then</code> , <code>nextTick</code>	<code>setTimeout</code> , <code>setImmediate</code> , I/O ↓	<code>nextTick</code> runs before promises

Ask anything



👉 Optional Deep Dives — want any of these?

1. 📄 Write your own microtask vs macrotask scheduler to visualize the order.
2. 📅 Timeline breakdown with `requestAnimationFrame` and `setTimeout`.
3. 🧠 React-specific behavior with async effects & batching.

Let me know where you want to go next.



Ask anything



+ 🛠️ Tools

