ES6 Modules vs CommonJS in Node.js

Node.js supports two main module systems:

1. **CommonJS (CJS)** – Default in older Node.js versions
2. **ES6 Modules (ESM)** – Modern standard, used in frontend and supported natively in recent Node.js versions

---

1. CommonJS

- **File Extension:** `.js`
- **Import Syntax:** `require()`
- **Export Syntax:** `module.exports` or `exports`

Example:

```
// math.js
function add(a, b) {
  return a + b;
}
module.exports = { add };
```

```
// app.js
const math = require('./math');
console.log(math.add(2, 3));
```

---

2. ES6 Modules

- **File Extension:** `.mjs` or `.js` with `"type": "module"` in `package.json`

- **Import Syntax:** `import`
- **Export Syntax:** `export` / `export default`

Example:

```
// math.mjs
export function add(a, b) {
  return a + b;
}
```

```
// app.mjs
import { add } from './math.mjs';
console.log(add(2, 3));
```

Key Differences

Feature	CommonJS (CJS)	ES6 Modules (ESM)
Syntax	`require`, `module.exports`	`import`, `export`
File extension	`.js`	`.mjs` or `.js` with `"type": "module"`
Loading style	Synchronous	Asynchronous
Support	Default in Node.js	Modern Node.js (14+)
Top-level await	Not supported	Supported
Used in	Node.js traditionally	Both frontend and backend

When to Use What

- Use **CommonJS** if you're working on older Node.js projects or need compatibility with legacy packages.
- Use **ES6 Modules** for new projects to align with modern JavaScript standards and browser support.

Mixing CJS and ESM

Mixing module types can be tricky:

- You can't use `require()` to import an ES6 module.
- You can't use `import` to load a CommonJS module unless it's default-exported.

Conclusion

Both module systems help organize and reuse code, but **ES6 modules are the future**, offering cleaner syntax and better interoperability across frontend and backend.