

Backend · concept board

Node.js

Runtime, event loop, modules, streams, scaling and npm

1. What Node.js is

4. Core modules, streams and errors

2. The Node event loop phases

5. Scaling and production

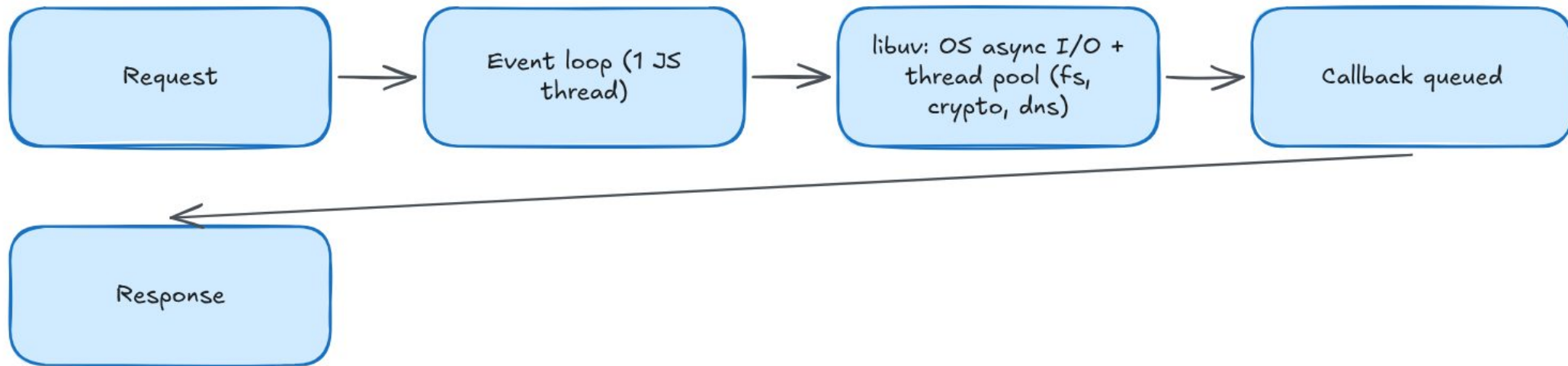
3. Modules and npm

Draw on any slide, then Present (arrow keys) or export the board as a PDF.

1. What Node.js is

Definition

Node.js is a JavaScript runtime built on V8 + libuv. It runs JS outside the browser with a single-threaded event loop and non-blocking I/O, so one process can serve many concurrent connections.



- Great for I/O-heavy work: APIs, real-time apps, proxies.
- Bad at long CPU work on the main thread (image processing, big loops) — it blocks every request. Use `worker_threads` or a separate service.

2. The Node event loop phases

1. timers — setTimeout / setInterval callbacks
2. pending callbacks — some deferred system callbacks
3. poll — wait for and run I/O callbacks
4. check — setImmediate callbacks
5. close — socket.on("close") etc.

Ordering (in a Node script)

```
setTimeout(() => console.log('timeout'), 0)
setImmediate(() => console.log('immediate'))
Promise.resolve().then(() => console.log('promise'))
process.nextTick(() => console.log('nextTick'))
console.log('sync')
// sync → nextTick → promise → timeout / immediate
// (timeout vs immediate order varies at top level; inside an I/O callback, immediate runs first)
```

Tip

process.nextTick runs before promise microtasks. Too many nextTicks can starve I/O.

3. Modules and npm

CommonJS	ES Modules
<code>require()</code> / <code>module.exports</code>	<code>import</code> / <code>export</code>
Loaded synchronously at runtime	Static, analysable, top-level await
Default for .js without "type": "module"	.mjs, or "type": "module" in package.json

- package.json: scripts, dependencies vs devDependencies, engines.
- Semver MAJOR.MINOR.PATCH: ^1.2.3 allows minor + patch, ~1.2.3 allows patch only.
- Commit package-lock.json; use npm ci in CI for exact, repeatable installs.
- npx runs a package binary without installing it globally.

4. Core modules, streams and errors

Read a big file without loading it all into memory

```
import { createReadStream } from 'node:fs'
import { createInterface } from 'node:readline'

const rl = createInterface({ input: createReadStream('access.log') })
let errors = 0
for await (const line of rl) {
  if (line.includes(' 500 ')) errors++
}
console.log('5xx responses:', errors)
```

- Core modules: fs/promises, path, http, crypto, events, stream, os, child_process, worker_threads.
- Streams process data chunk by chunk (Readable, Writable, Duplex, Transform) — use pipeline() for errors and back-pressure.
- EventEmitter: on, once, emit — the base of streams and servers.
- Always handle promise rejections; log unhandledRejection / uncaughtException, then exit and let the process manager restart.

Common mistake

Sync APIs in request handlers (readFileSync, bcrypt.hashSync, big JSON.parse loops) block every other user. Use the async versions.

5. Scaling and production

Tool	What it does
cluster / PM2	one process per CPU core behind a shared port
worker_threads	run CPU-heavy work off the main thread
Load balancer + stateless app	scale horizontally across machines
Redis	shared cache, sessions, rate limits, queues (BullMQ)
Graceful shutdown	on SIGTERM stop accepting, finish in-flight requests, close DB

- Config from environment variables (12-factor); never commit secrets.
- Structured logs (pino), health checks (/health), metrics.
- Cron jobs: node-cron in a single instance or a queue scheduler to avoid duplicates when scaled.

Revise

Single-threaded event loop + libuv for non-blocking I/O. nextTick → promises → timers/check. Don't block the loop; use streams for big data, worker_threads for CPU, cluster/PM2 + stateless design to scale.

Rapid-fire Q&A (1/2)

Is Node.js single-threaded?

JavaScript runs on one thread (the event loop), but libuv uses the OS and a thread pool for I/O, crypto and DNS, so I/O doesn't block.

What blocks the event loop?

Long synchronous work: CPU-heavy loops, sync file/crypto APIs, huge JSON parsing. Every request waits.

process.nextTick vs setImmediate vs setTimeout?

nextTick runs before other microtasks; setImmediate runs in the check phase after I/O; setTimeout(0) runs in the timers phase.

What are streams and why use them?

Chunk-by-chunk processing of data (files, HTTP bodies) with low memory and back-pressure handling.

CommonJS vs ES Modules?

CommonJS uses require/module.exports and loads synchronously; ESM uses import/export, is statically analysable and supports top-level await.

How do you scale a Node app?

cluster/PM2 to use all cores, stateless servers behind a load balancer, Redis for shared state, worker_threads or queues for CPU work.

Rapid-fire Q&A (2/2)

dependencies vs devDependencies?

dependencies are needed at runtime;
devDependencies only for building and testing.

^ vs ~ in package versions?

^1.2.3 allows minor and patch updates;
~1.2.3 allows patch updates only.

What is graceful shutdown?

On SIGTERM: stop accepting connections, finish in-flight requests, close DB connections, then exit.