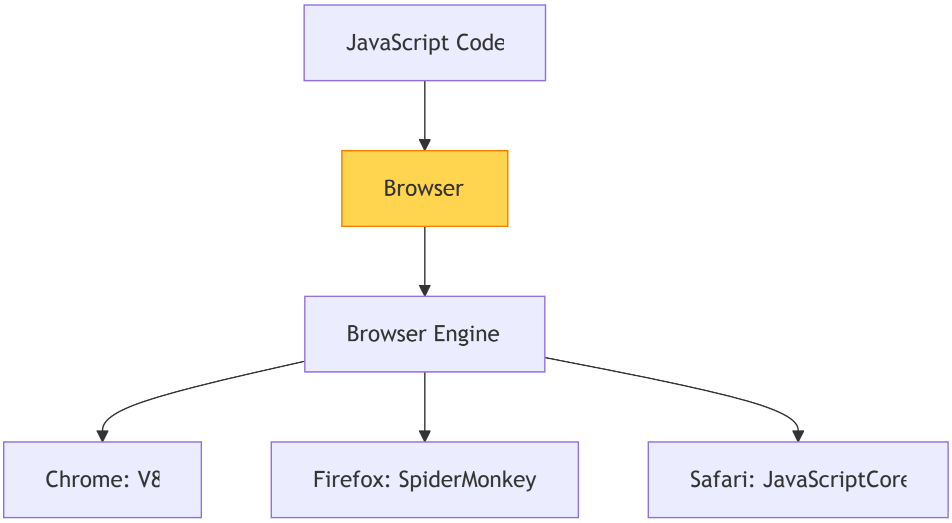
01-Intro/03-nodejs-architecture.md

⚙️ Node.js Architecture & How It Works

🕰️ Before Node.js (Pre-2009)

JavaScript in the Browser

JavaScript was **only** meant to run inside web browsers.



Different browsers = Different JavaScript engines!

Browser-Specific Objects

```
// Only available in browsers
document.getElementById('myElement');
window.location.href;
console.log(window);
```

🎉 2009: Ryan Dahl's Revolution

The Birth of Node.js

Ryan Dahl embedded **Chrome's V8 JavaScript Engine** into a **C++ program**

`V8 Engine (JavaScript) + C++ Program = Node.js`

Result: JavaScript can now run **anywhere**! 🚀

🏗️ Node.js vs Browser

Different Objects, Same Language

🌐 **Browser**	🟢 **Node.js**

`// Browser-specific`
`document.getElementById('id');`
`window.location;`
	`// Node-specific`
	`fs.readFile('file.txt');`
	`http.createServer();`

📦 What is Node.js?

Architecture Components

```
Node.js = V8 JavaScript Engine + Additional Modules
```

Additional Modules Include:

- 📁 File System (fs)
- 🌐 Networking (http, https)
- 🔐 Cryptography
- 🗜 Compression
- 🎯 Much more...

> ⚠️ **Important:** Node.js is **NOT a framework**!
> It's different from ASP.NET, Rails, or Django.

🚀 How Does Node Work?

Core Characteristics

📈 **Very Scalable** Handle thousands of connections efficiently

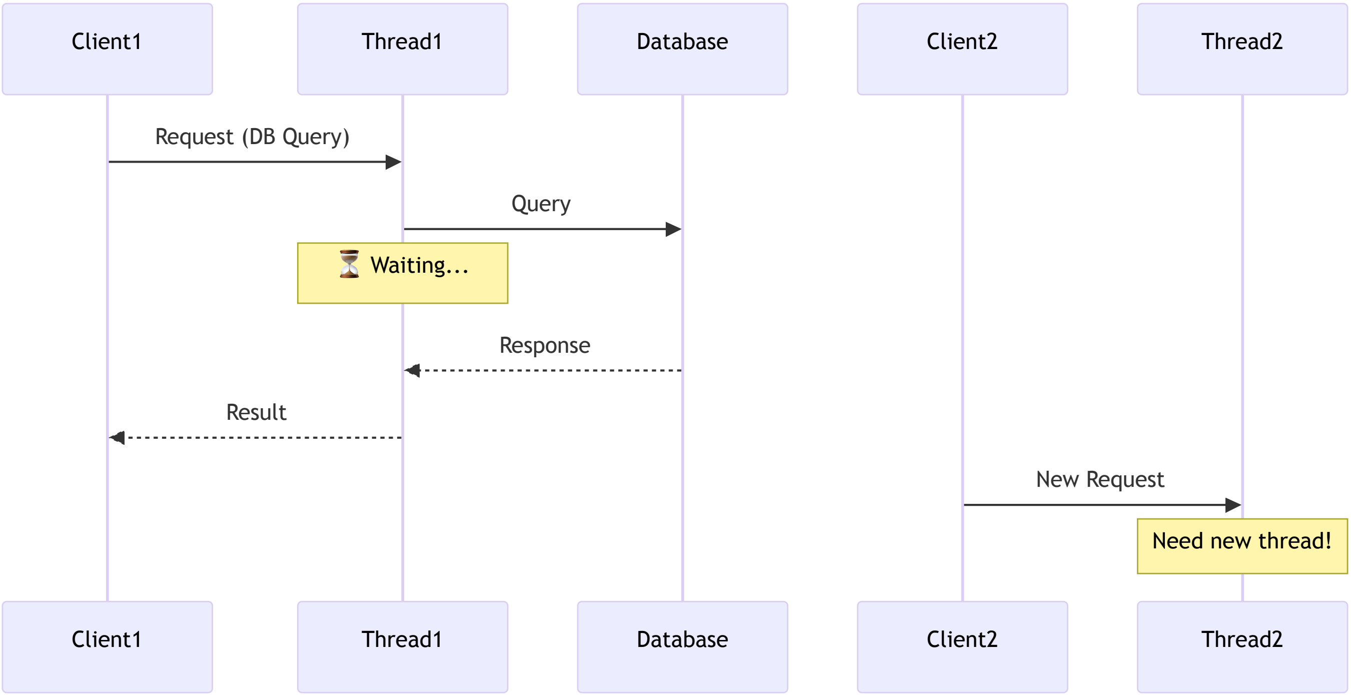
⚡ **Real-time** Perfect for live applications

🔄 **Non-blocking** Asynchronous by nature

💛 **JavaScript** Use the language you know

🔄 Synchronous vs Asynchronous

❌ Synchronous (Traditional Approach)

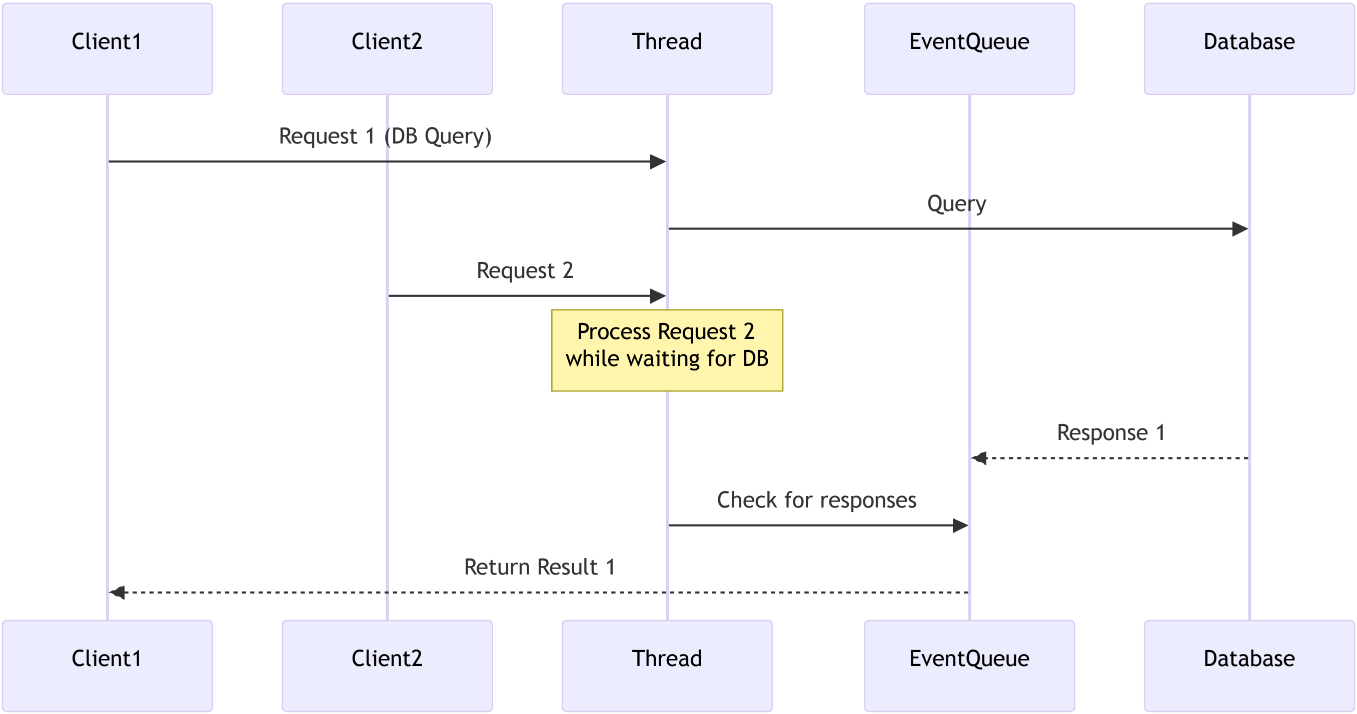


Problems:

- ❌ Each request needs a new thread
- ❌ Threads wait idly during I/O operations
- ❌ With many clients, threads get exhausted
- ❌ Need to deploy more hardware
- 💰 **Expensive to scale**

Example: Default ASP.NET behavior

✅ Asynchronous (Node.js Approach)



Benefits:

- ✅ **Single thread** handles all requests
- ✅ No waiting during I/O operations
- ✅ **Event Queue** manages responses
- ✅ Thread serves next request immediately
- 🚀 **Highly efficient**

🍽️ Restaurant Analogy

Asynchronous (Node.js Way) ✅

```
👨‍🍳 1 Waiter (Thread) → Multiple Tables

1. Takes order from Table 1
2. Sends order to kitchen
3. Immediately takes order from Table 2
4. Takes order from Table 3
5. Delivers ready orders as they come
```

Efficient: One waiter handles multiple tables!

Synchronous (Traditional Way) ❌

```
👨‍🍳 Waiter takes order from Table 1
⏳ Waits in kitchen until food is ready
🚶 Delivers food to Table 1
🔄 Only then takes order from Table 2
```

Inefficient: Need multiple waiters (threads)!

✅ Node.js is GOOD For

📊 I/O Intensive Applications

Applications with **heavy disk or network access:**

- 🌐 Web APIs
- 💬 Chat applications
- 📊 Real-time analytics
- 📁 File processing
- 🔄 Data streaming
- 📡 Microservices

Why? Node.js doesn't block while waiting for I/O operations!

❌ Node.js is NOT Good For

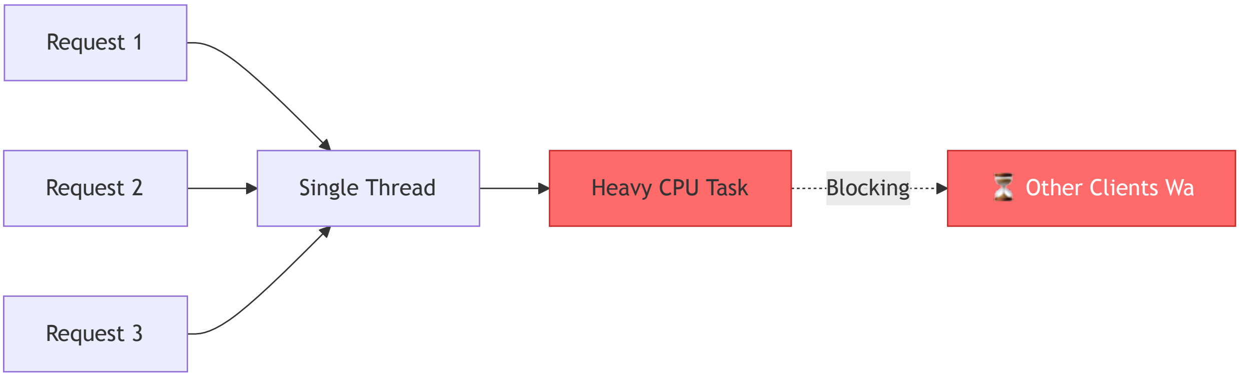
🖥️ CPU Intensive Applications

Applications requiring **heavy calculations:**

- 🎬 Video encoding
- 🖼️ Image processing
- 🧮 Complex mathematical computations
- 🎨 3D rendering
- 🔐 Encryption/Decryption of large files

Why? Single thread gets blocked by heavy CPU operations!

⚠️ The Problem with CPU Intensive Tasks



> When the CPU is busy with intensive calculations, other clients must wait until the thread is free again!

🎯 Summary

Concept	Key Takeaway
Architecture	V8 Engine + C++ + Additional Modules
Threading	Single-threaded with Event Loop
Execution	Asynchronous & Non-blocking
Best For	I/O Intensive Applications
Avoid For	CPU Intensive Applications

🔜 Next Steps

Now that you understand the architecture, let's get Node.js **installed and running** on your machine!

🏠 Course Home | 📘 Chapter 1 Home

← Previous: What is Node.js | Next: Installation & Setup →