Mastering Node.js: A Comprehensive Crash Course

Introduction to Node.js

- **Open-source, server-side** JavaScript **runtime**.
- Allows you to run **JavaScript** on your **machine** or **servers**.
- Built on the **V8 JavaScript engine** for speed.
- Robust ecosystem with **npm**

Open Source??

Open-source means that the **source code of Node.js is publicly available.**

Anyone can:

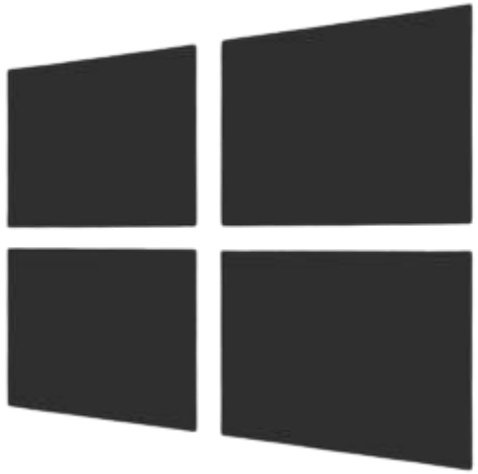
- **View**: You can see how Node.js is built under the hood.
- **Modify**: If you want to customize or improve Node.js, you can do it.
- **Contribute**: Developers around the world actively contribute to improving Node.js.

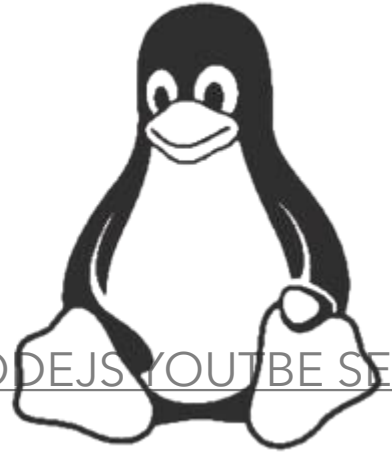
Cross-Platform??

Cross-platform means Node.js can **run on multiple operating systems** without needing major changes.

Supported platforms:

Windows **macOS** **Linux**





JavaScript Runtime Environment??

JavaScript runtime environment refers to the **environment where your JavaScript code runs.**

In browsers: JavaScript typically runs in the browser (like Chrome or Firefox) to handle frontend tasks.

With Node.js:

Node.js allows JavaScript to **run outside the browser, on the server**.

It provides tools to interact with the system, like:

File system (read/write files).

Network (handle HTTP requests).

Database (connect to databases like MongoDB or MySQL).

Why is Node.js Special as a Runtime?

V8 Engine:

Node.js uses Google Chrome's V8 engine to compile JavaScript into machine code, making it lightning-fast.

Built-in APIs:

Node.js comes with built-in APIs (like fs for file systems or http for servers) so you can build powerful applications without extra libraries.

Why is Node.js Special as a Runtime?

V8 Engine:

Node.js uses Google Chrome's V8 engine to compile JavaScript into machine code, making it lightning-fast.

Built-in APIs:

Node.js comes with built-in APIs (like fs for file systems or http for servers) so you can build powerful applications without extra libraries.

Why Do We Need Node.js?

Single Language for Full Stack Development:

- Developers can use JavaScript for both frontend and backend, reducing complexity.
- For example, a MERN stack project (MongoDB, Express, React, Node).

High Performance:

- Thanks to the V8 engine, Node.js is super fast.
- It can handle thousands of simultaneous connections with ease.

Event-Driven and Non-Blocking I/O:

- Unlike traditional servers that block a thread for each request, Node.js uses an asynchronous model, making it highly efficient for handling multiple tasks.

THAPA TECHNICAL – NODEJS YOUTBE SERIES

Scalable for Modern Applications:

Ideal for real-time applications like chat apps, streaming platforms, or online games.

Where Is Node.js Used?

Examples of Companies Using Node.js:

Netflix: To deliver faster streaming experiences.

LinkedIn: Their backend uses Node.js to handle massive amounts of data.

PayPal: To handle secure transactions and ensure scalability.

Uber: For real-time driver and rider data syncing.

Trello: For managing asynchronous updates in real-time boards.

Million Dollar Question

Is it a Language or a Framework?

Node.js is not a programming language or a framework, but rather a JavaScript runtime environment that allows developers to run JavaScript outside of a browser

History and Evolution

History and Evolution

- **JavaScript's Initial Role:** Initially, JavaScript was only used for client-side tasks in web browsers.
- **Creation of Node.js (2009):** Ryan Dahl created Node.js by taking the V8 JavaScript engine (used in Google Chrome) and running it outside the browser. This innovation allowed JavaScript to handle server-side development.
- **Introduction of Modules:** Node.js came with built-in modules like http, which made it popular for creating backend applications.

There has been lots of drama behind Node.js and its evolution. If you want to learn more, watch this [video https://www.youtube.com/watch?v=LB8KwiiUGy0](https://www.youtube.com/watch?v=LB8KwiiUGy0)

History and Evolution

- **npm (2010):** Node.js launched its package manager (npm) in 2010, revolutionizing JavaScript development by simplifying package sharing and dependency management.
- **io.js and Node.js Merger (2015):** In 2015, Node.js merged with io.js, reuniting the ecosystem and bringing in community-driven improvements.
- **Node.js Foundation (2015):** The Node.js Foundation was established to support collaboration, innovation, and sustainable development within the ecosystem.

There has been lots of drama behind Node.js and its evolution. If you want to learn more, watch this [video](https://www.youtube.com/watch?v=LB8KwiiUGy0) https://www.youtube.com/watch?v=LB8KwiiUGy0

Some Famous npm Packages

- **Express.js:** A popular framework for building web applications and APIs.
- **Mongoose:** A package for interacting with MongoDB databases using an object data modeling (ODM) approach.
- **Axios:** A promise-based HTTP client for making API requests.
- **React (and React-DOM):** Though originally developed for the browser, React can be installed via npm for building modern web applications.
- **Socket.IO:** Enables real-time communication between clients and servers.
- **Passport.js:** Authentication middleware for Node.js.

Merger of Node.js and io.js:

What happened:

- io.js was a fork of Node.js created by developers who wanted faster updates and improvements. The merger brought io.js's community-driven innovations back into the main Node.js project.
- The actual merger took place in September 2015, resulting in the release of Node.js v4.0. This version combined the best features of both projects, including V8 ES6 support from io.js, and marked the beginning of a Long-Term Support (LTS) release cycle for Node.js.
- This ensured faster progress, stability, and a unified ecosystem for developers.

Prerequisites to Learn Node.js

Prerequisites to Learn Node.js

HTML

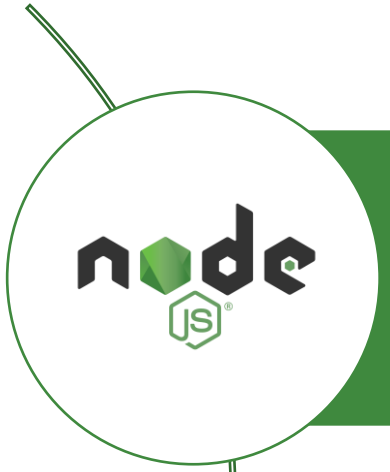
CSS

JavaScript

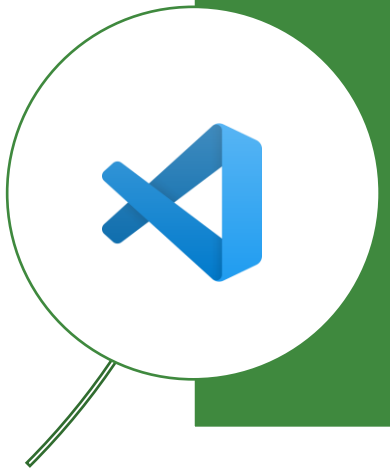
- Basic & Intermediate
- ES6 & Above

Node.JS Installation

Setting up Development Environment



Download Node.js

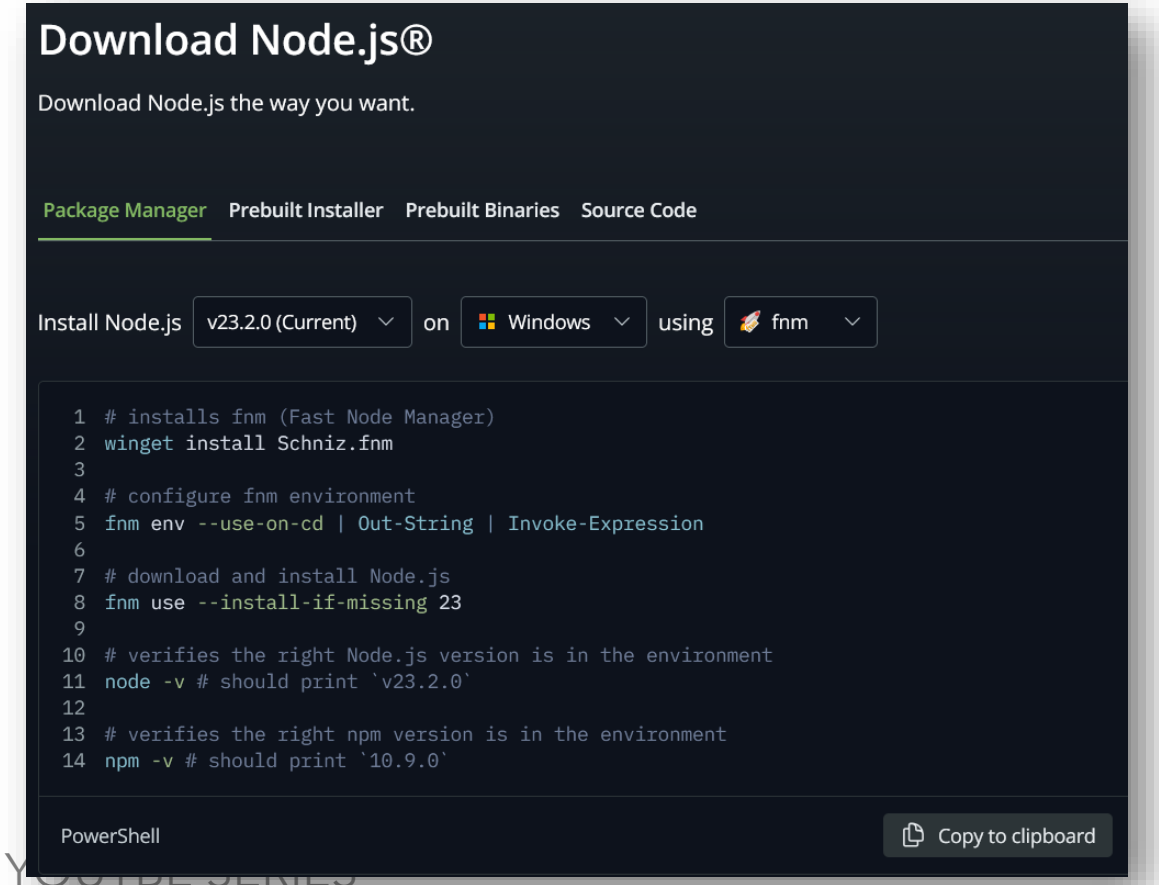


Visual Studio Code

- Prettier
  - Turn on Format on Save

Installing Node.js and npm (FNM)

- You can download Node.js from its official website. We will be using FNM in this course.
- FNM (Fast Node Manager) is a command-line utility that enables developers to easily manage and switch between different versions of Node.js on their machines.
- We are using v23.2.0 in this course, but all the content in this course will be valid for upcoming versions too.
- If you don't have winget on Windows, then you can also use chocolatey, or scoop to install FNM. Details are mentioned here.
  - Of course, you need to install chocolatey or scoop first.



REPL (Read-Eval-Print Loop)

Introduction to REPL (Read-Eval-Print Loop)

- It stands for Read-Eval-Print Loop or Read, Evaluate, Print and Loop. It's an interactive programming environment that allows you to execute JavaScript code one statement at a time.
- You can open your terminal and use node command.
- You can use it to test simple JavaScript code similar to browser devtools.
- Ctrl + C to exit out of it.

REPL: Read-Eval-Print Loop

Each part of the acronym highlights a specific step in how the REPL works:

- Read: The REPL reads the user's input (a single line or multiple lines of code) and parses it into a data structure that the JavaScript engine can understand.
- Eval (Evaluate): The parsed input is evaluated (executed) by the JavaScript engine. If the input is a valid expression, the REPL computes the result.
- Print: The result of the evaluated expression is printed back to the console so the user can see the output.
- Loop: The process then loops back, waiting for the next input, and continues until the user exits.

Node.js Basics

Understanding the Node.js Module System
Creating and Organizing Modules
Common Node.js Built-in Modules
Streams and Buffers
Introduction to package.json
ES Modules in Node.js

Creating Your First "Hello World" Application

What is CLI?

- CLI (Command Line Interface) is a way to execute predefined JavaScript files or run commands for tasks like installing packages, running scripts, or starting applications.

When to Use CLI?

- Building Applications: Execute your scripts to run applications or servers.
- Automation: Use npm scripts for repetitive tasks (e.g., building or testing code).
- Large-Scale Projects: Execute full files for consistent and reusable logic.

Windows vs Global in Node.js

Did you know that JavaScript behaves differently in the browser and in Node.js?

Node.js Context - global

- In Node.js, there's no window or document. Why? Because Node.js runs outside the browser—it doesn't deal with the DOM or browser-specific APIs.

- Instead, Node.js has a global object. It's the equivalent of window in the browser but designed for a server-side environment.

globalThis

globalThis is a new feature introduced in ECMAScript 2020 (ES11) that provides a standard way to access the global object in any JavaScript environment.

Why is it useful?

- Consistent Access: In the past, accessing the global object varied depending on the environment:
    1. Browser: window
    2. Node.js: global
    3. Web Workers: self
    4. Other Environments: Might have their own global objects

Node.js Module

Understanding the Node.js Module System

Self-contained code unit:

- Each file in Node.js is treated as a separate module.
- Variables, functions, or objects defined in one file are not accessible in another file by default unless you explicitly export them.

Encapsulation:

- Node.js uses the CommonJS module system (module.exports and require) to ensure the code in one file does not pollute or interfere with the global scope.
- This makes your code modular, maintainable, and easier to debug.

What Exactly is a Module in Node.js?

- A module in Node.js represents a file containing code that is self-contained, reusable, and encapsulated.
- Node.js uses the CommonJS module system.
- This module system came before ES Modules was introduced in JavaScript; that's why its syntax is different.
- Modules in Node.js are created by defining separate files for different functionalities.
- You must export anything you want to make accessible to other modules.

Creating and Organizing Modules

```
// math.js
const add = (a, b) => a + b;
module.exports = add;

// app.js
const addFunction = require('./math');
console.log(addFunction(2, 3)); // Outputs: 5
```

- To make variables or functions available outside the module, they must be explicitly exported using **module.exports**.
- In another module, the exported content can be imported using **require()**.

Named Export

Key Takeaways

- Avoid mixing module.exports and exports.property incorrectly.
  If you reassign module.exports, it will override any previous exports.property assignments.

- Use consistent syntax for clarity:
  - Assign everything at once with module.exports = { ... }.
  - Or assign properties individually with module.exports.property.

Path Module

Path Module - NodeJS

In Node.js, the path module provides utilities for working with file and directory paths. It's a built-in module, so you don't need to install any external packages to use it.

Special Node.js Constants

__filename

- Provides the absolute path of the currently executing file.

__dirname

- Provides the absolute directory path of the currently executing file.

*Note, these are only available in commonjs

Path Module Features

path.parse(): Returns an object with details about a given path, including root, dir, base, ext, and name.

path.join(): Joins multiple path segments into one, using the appropriate separator (\ on Windows, / on Linux/macOS).

path.resolve(): Resolves a sequence of paths into an absolute path, starting from the current directory.

path.extname(): Extracts the file extension from a given path.

path.basename(): Returns the last part of a path (e.g., file name with extension).

path.dirname(): Returns the directory part of a path.

path.sep: Returns the platform-specific path segment separator (\ for Windows, / for Linux/macOS).

OS Module

OS Module - NodeJS

1. os.platform(): Returns the OS platform (e.g., 'win32' for Windows, 'linux' for Linux, 'darwin' for macOS).

Usage: Useful for writing cross-platform applications.

2. os.arch(): Returns the CPU architecture (e.g., 'x64', 'arm').

Usage: Helps optimize code for specific architectures.

3. os.freemem(): Returns the amount of free system memory in bytes.

Usage: Useful for monitoring system performance.

4. os.totalmem(): Returns the total system memory in bytes.

Usage: Provides insights into the machine's capacity.

5. os.uptime(): Returns the system uptime in seconds.

Usage: Commonly used in logging or monitoring tools || Useful for storing temporary data.

OS Module - NodeJS

6. os.homedir(): Returns the home directory of the current user.

Usage: Useful for locating user-specific files.

7. os.hostname(): Returns the hostname of the system.

Usage: Useful for logging or identifying machines in networks.

8. os.networkInterfaces(): Returns an object with details about the network interfaces.

Usage: Helps in network diagnostics or configuration.

9. os.cpus(): Returns details about each logical CPU/core.

Usage: Helps optimize code for multi-core processing.

10. os.tmpdir(): Returns the default directory for temporary files.

Crypto Module

Crypto Module - NodeJS

1: crypto.randomBytes(size)

Purpose: Generates cryptographically strong random data.

Usage: Useful for creating tokens or unique IDs.

2. crypto.createHash(algorithm)

Purpose: Creates a hash for a given input using algorithms like sha256.

Usage: Ensures data integrity (e.g., verifying file changes).

crypto.randomBytes(size)

Real-Life Examples:

1. Token Generation for Password Reset
    - When users request a password reset, generate a secure random token.
    - Example:
    - A user clicks "Forgot Password" → A random token is sent via email → The token is verified during the reset process.

1. API Keys and Secret Generation
    - Generate secure API keys for third-party integrations.
    - Example:
    - When creating a developer account on platforms like Twitter or GitHub, a unique API key is generated.

crypto.createHash(algorithm)

Real-Life Examples:

1. Password Hashing (Login Systems)
    - Hash user passwords before storing them in a database to enhance security.
    - Example:
    - A user's password is hashed with sha256 and stored. During login, the entered password is hashed and compared.

Common Node.js Built-in Modules

- fs (File System): Handles file operations like reading, writing, and deleting files.
- path: Simplifies working with file and directory paths.
- os (Operating System): Provides information about the operating system and hardware.
- crypto: Offers tools for encryption, hashing, and secure token generation.
- http/https: Allows creation of web servers and handling of HTTP/HTTPS requests.
- events: Supports an event-driven programming model with emitters and listeners.

THAPA TECHNICAL – NODEJS YOUTBE SERIES

You can also checkout Node.js official website to see all built-in modules.

FS Module

Thapa Technical

FS Module - Node.js

- The fs (File System) module in Node.js is a core module that allows you to work with the file system, enabling you to read, write, update, delete, and watch files.

Let’s see how to URUD operation in a Synchronous way

FS Module – CRUD Operations

Create (fs.writeFile)

- Writes content to the file. If the file does not exist, it creates one.

Read (fs.readFile)

- Reads the content of the file asynchronously and logs it.

Update (fs.appendFile)

- Appends new content to the file without overwriting the existing content.

Delete (fs.unlink)

- Deletes the file completely.

fs/promises Module - Node.js

The fs/promises module provides a promise-based API for interacting with the filesystem, allowing asynchronous operations to be handled using modern JavaScript features like async/await or .then() chains. It is part of the fs module in Node.js but designed for those who prefer promises over callback-based or synchronous methods.

Purpose: Simplifies asynchronous file operations by using promises, making the code more readable and modern compared to traditional callback-based approaches.

fs/promises Module - Node.js

Advantages:

No need for manual callbacks.

Cleaner and more intuitive asynchronous workflows using async/await.

Suitable for modern JavaScript applications.

Use Cases

Reading Files asynchronously in a non-blocking way.

Writing or Appending Data to files without blocking the event loop.

Performing Multiple File Operations sequentially or concurrently with promise chaining.

Handling Errors Gracefully with .catch() or try..catch blocks.

Event Module

Thapa Technical

Event Module - Node.js

EventEmitter is a core module in Node.js used to create and handle custom events.

It is part of the events module and is often used for building event-driven systems in Node.js.

Key Methods

1. emit(eventName, [args])

Purpose: Emits (or triggers) an event with the specified eventName. You can also pass arguments that will be consumed by the listeners.

It's like calling a function, but instead, it triggers all listeners (functions) attached to the specified event.

2. on(eventName, listener)

Purpose: Attaches a listener (a function) to a specific eventName. This listener will execute when the event is emitted.

Event Module - Node.js

Think of it as defining custom events and then triggering them:

Defining a Listener (on):

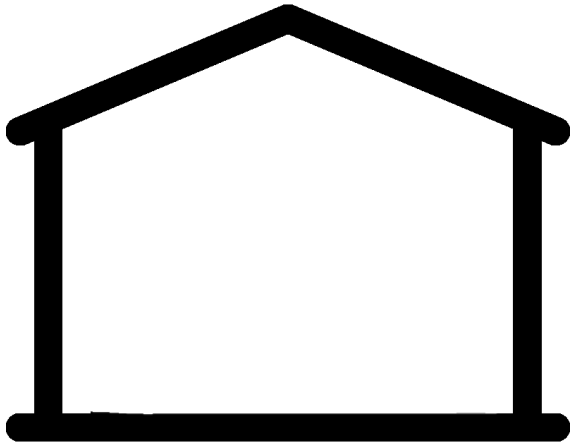
It's like defining a function for an event.

The listener will execute when the corresponding event is triggered.

Emitting an Event (emit):

It's like calling that event's listener.

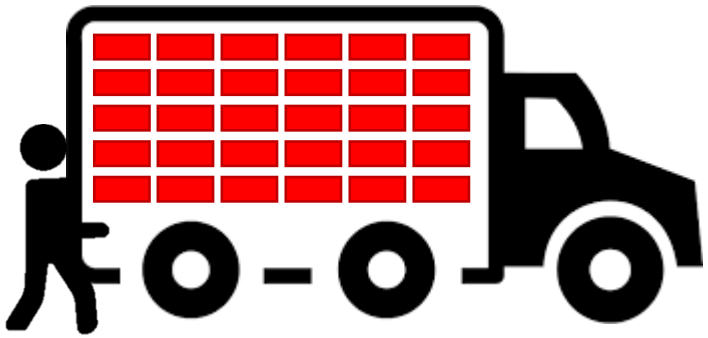
Streams and Buffers

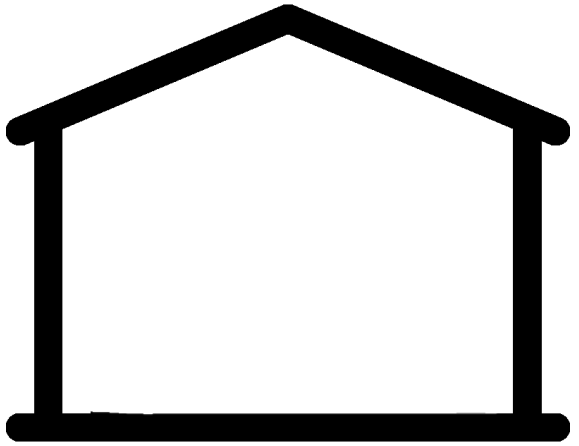


THAPA TECHNICAL – NODEJS YOUTBE SERIES

Imagine, you have a truck full of bricks that you want to carry.

Streams and Buffers





THAPA TECHNICAL – NODEJS YOUTBE SERIES

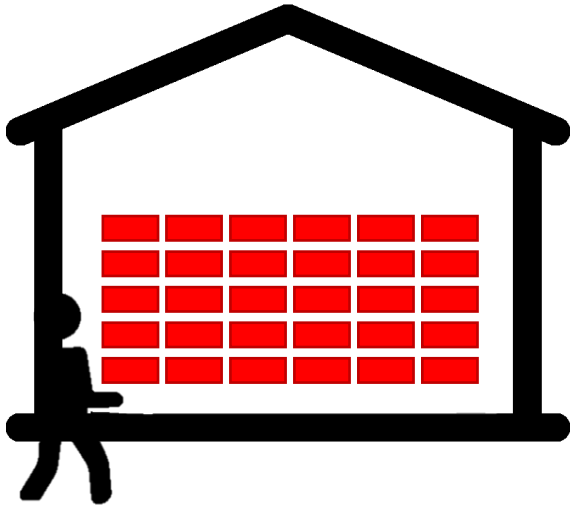
You can try to carry everything at once

Streams and Buffers

THAPA TECHNICAL – NODEJS YOUTBE SERIES

You can try to carry everything at once

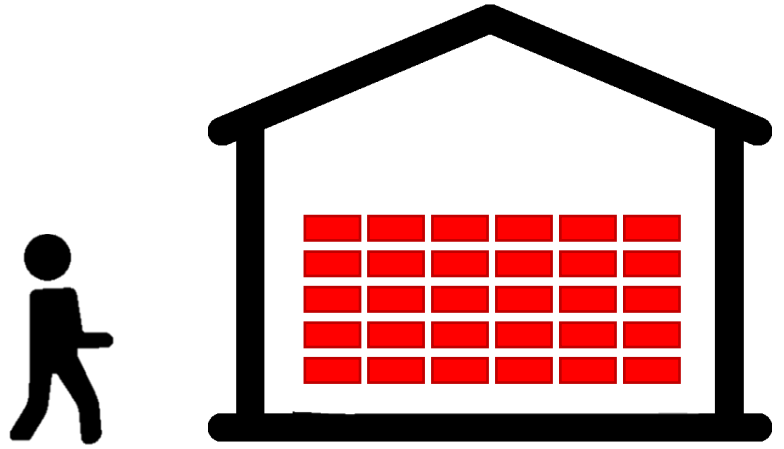
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

You can try to carry everything at once

Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

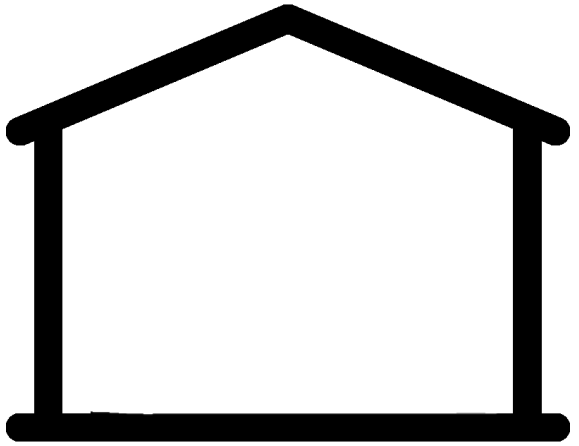
You can try to carry everything at once

Streams and Buffers

THAPA TECHNICAL – NODEJS YOUTBE SERIES

But is this possible? Maybe, if you are a gorilla, but it’s not possible for a normal human being.

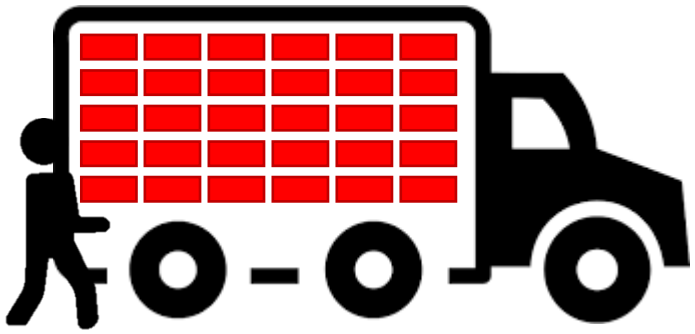
Streams and Buffers

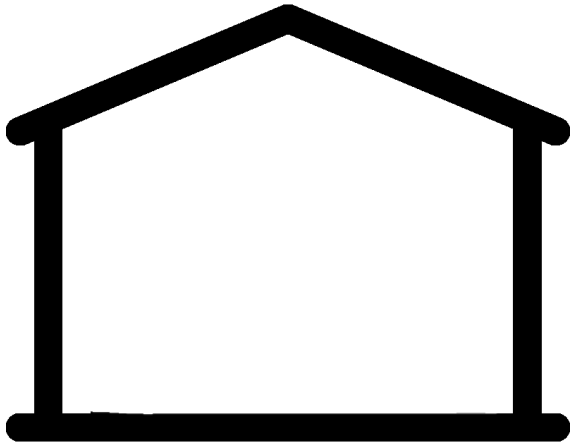


THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers

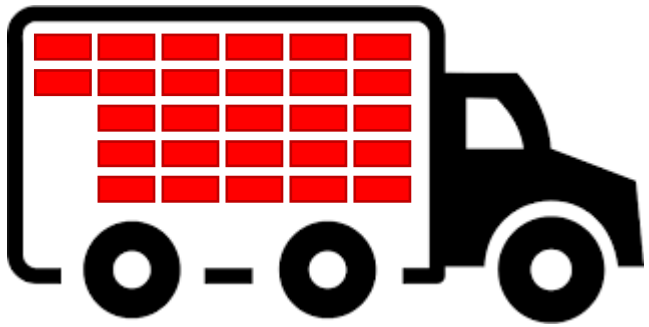


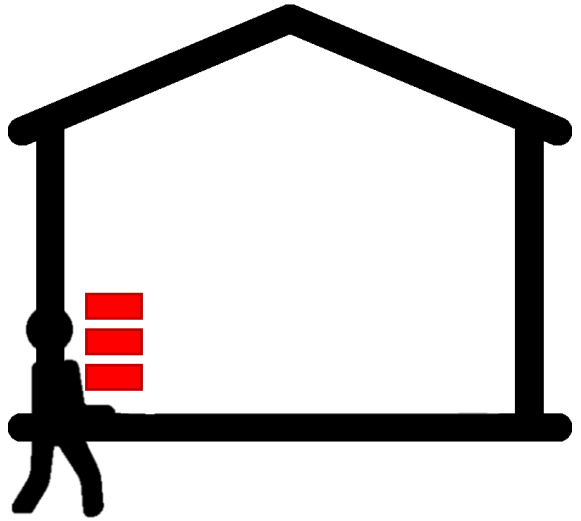


THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers

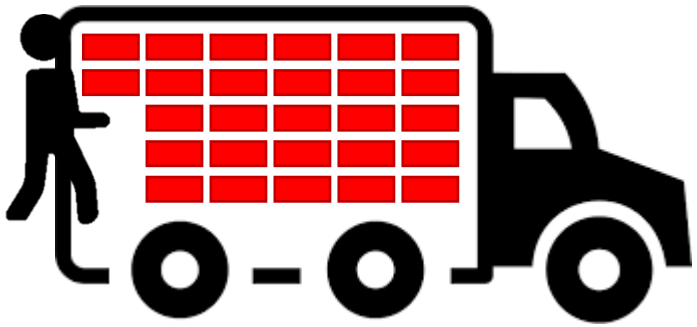


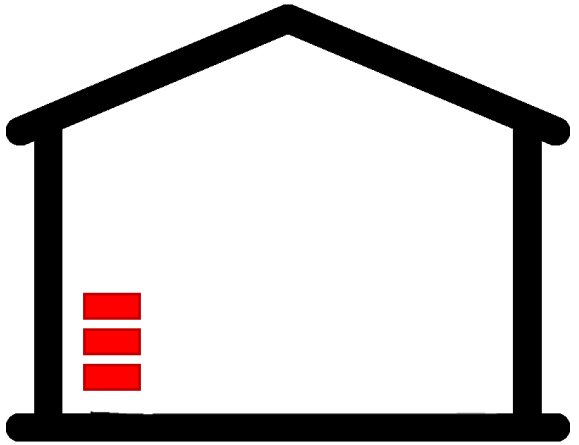


THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers

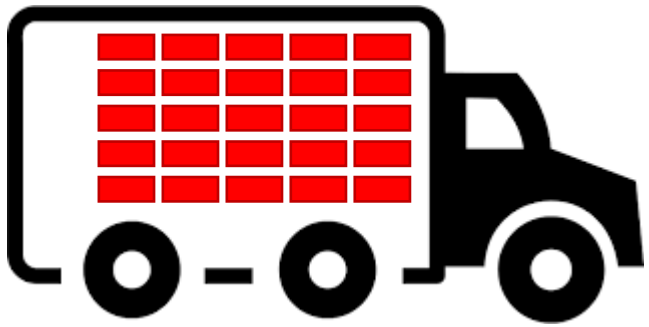


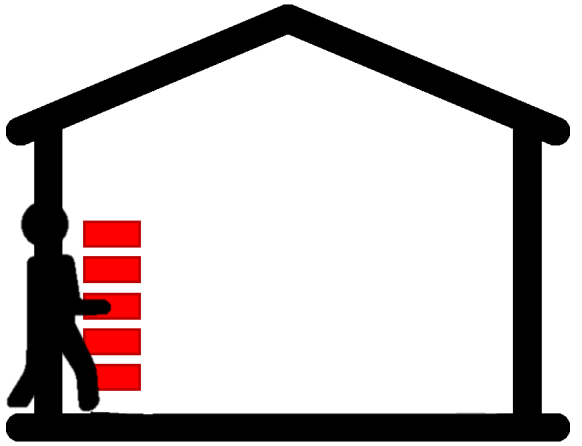


THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers

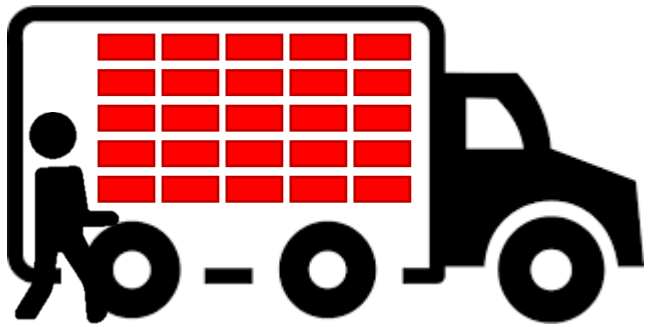


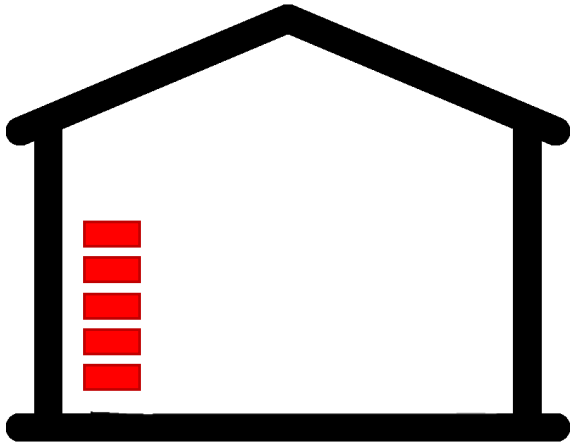


THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers

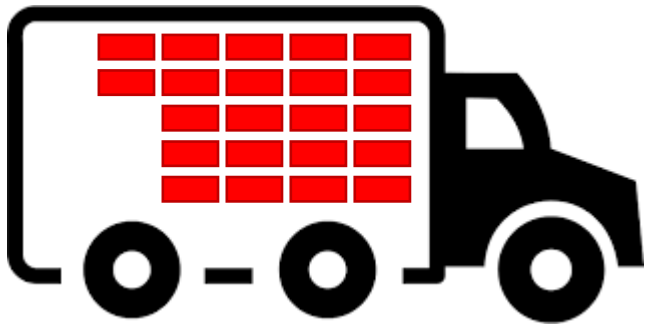




THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

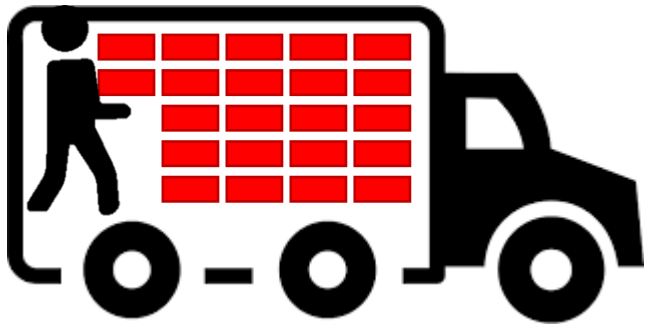
Streams and Buffers

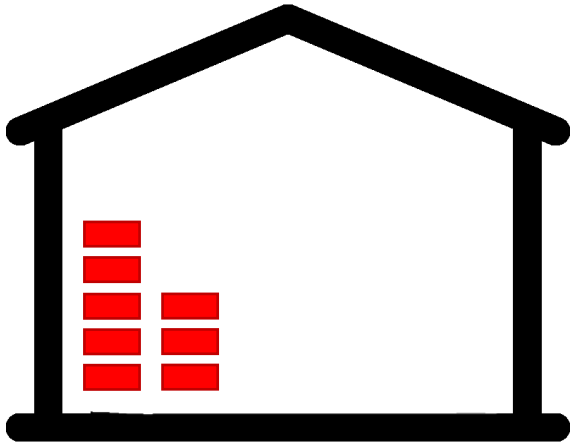


THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers

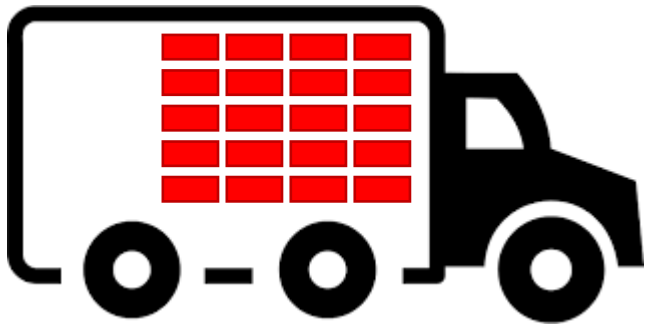




THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

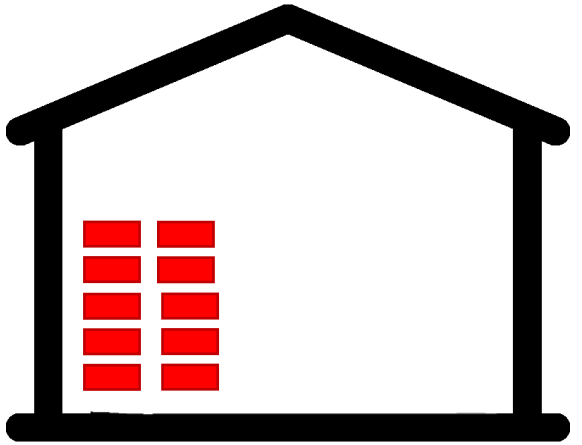
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

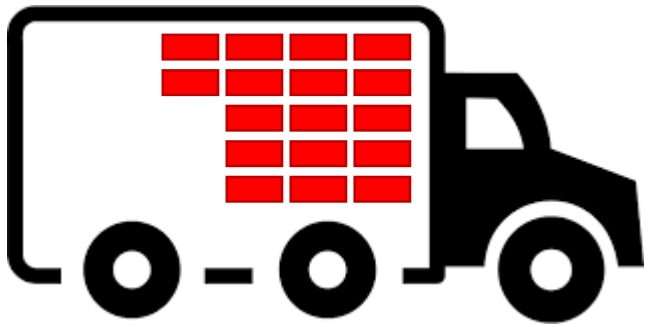
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

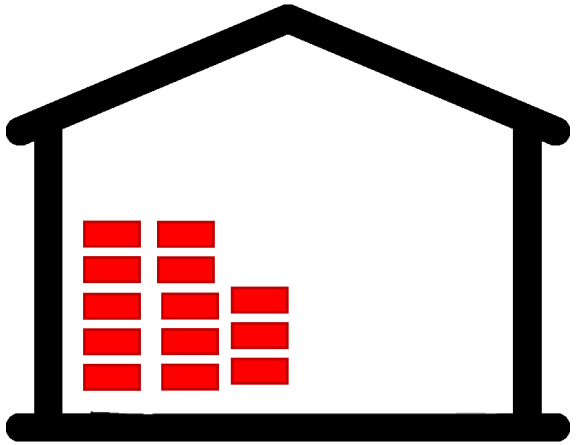
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

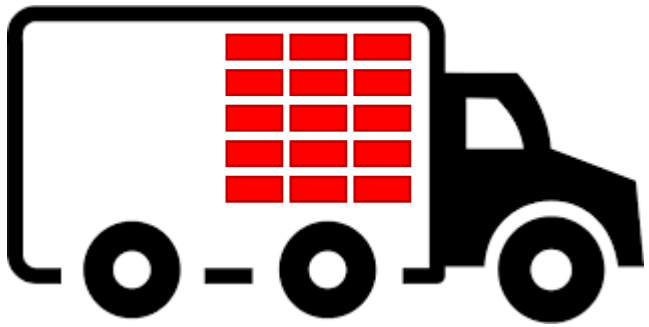
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

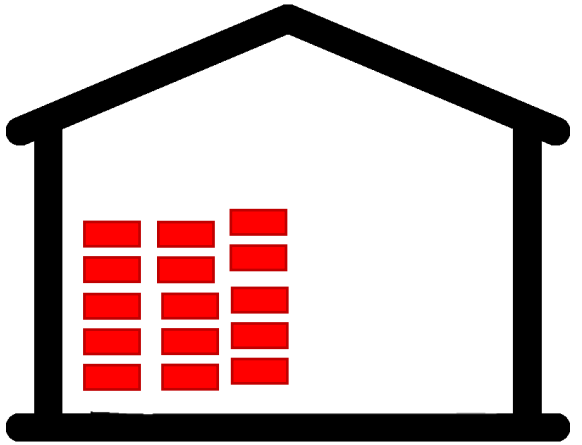
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers

THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers

THAPA TECHNICAL – NODEJS YOUTBE SERIES

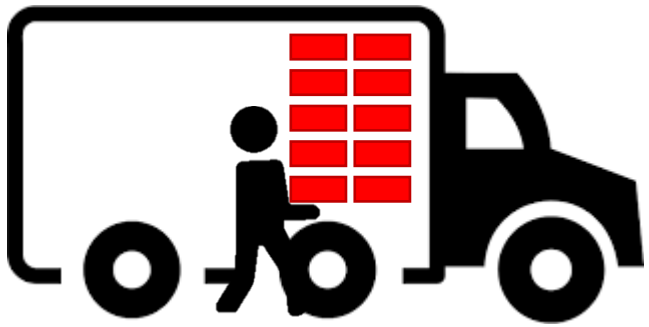
So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers

THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

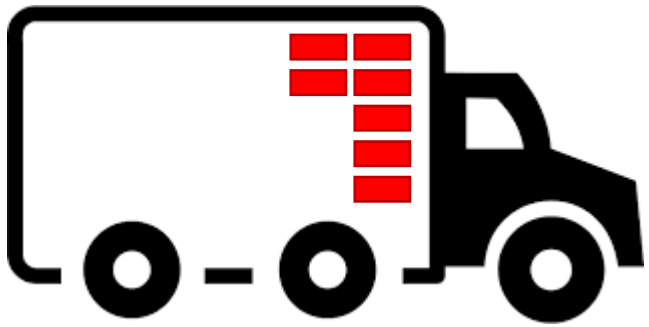
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

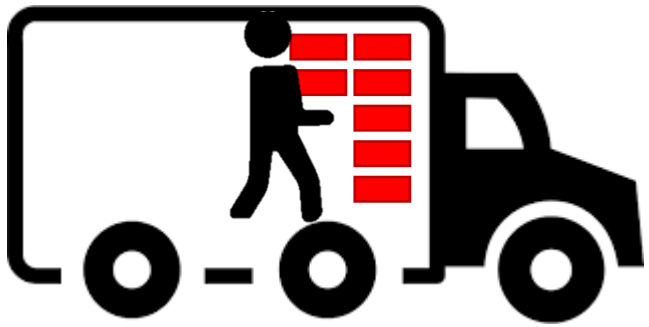
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

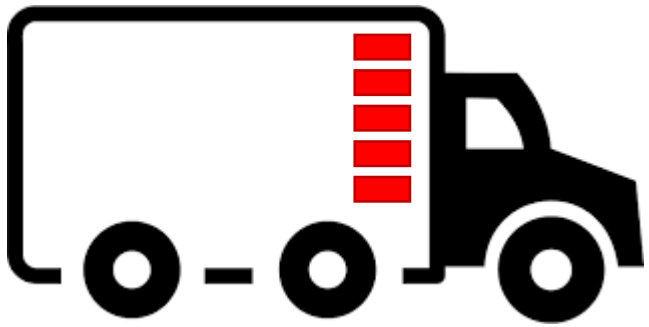
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

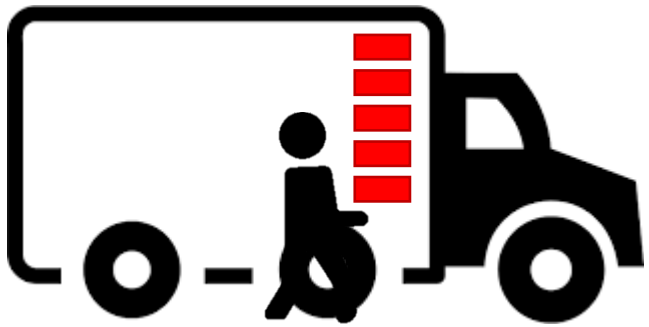
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

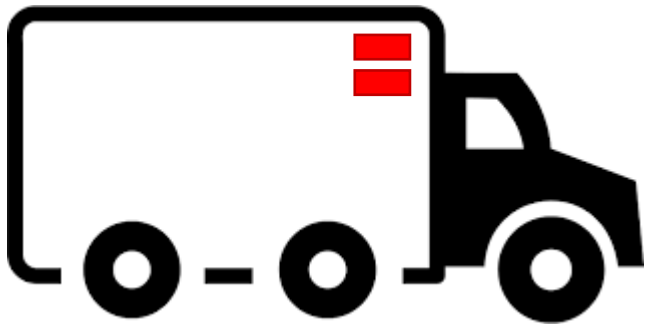
Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

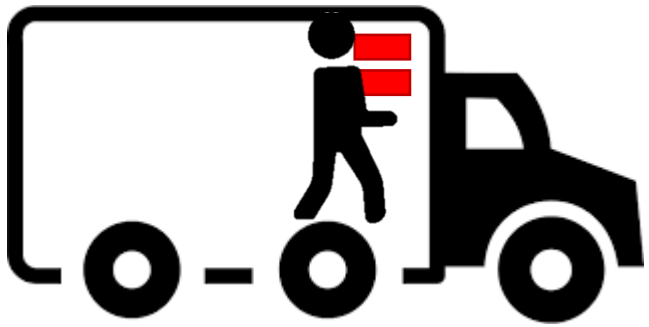
Streams and Buffers

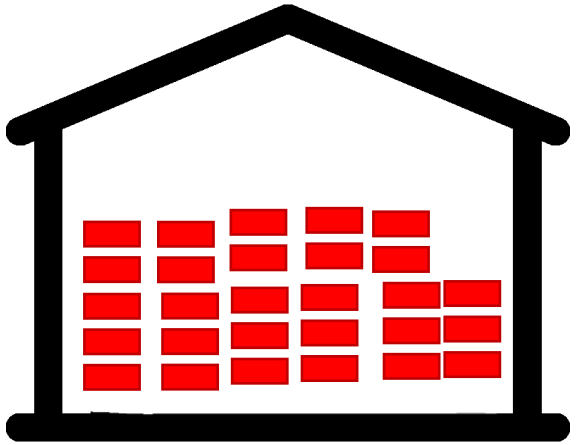


THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers





THAPA TECHNICAL – NODEJS YOUTBE SERIES

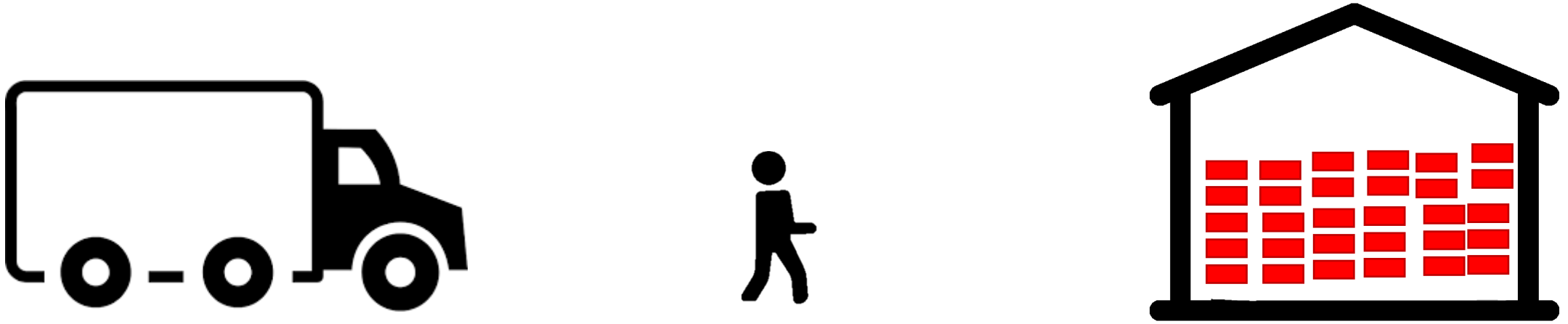
So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers

THAPA TECHNICAL – NODEJS YOUTBE SERIES

So, what you will typically do is, take small chunk of bricks one by one and take it.

Streams and Buffers



THAPA TECHNICAL – NODEJS YOUTBE SERIES

Finally completed, but how is this related to Node.js?

Streams and Buffers

8 GB RAM

16 GB RAM

THAPA TECHNICAL – NODEJS YOUTBE SERIES

Let's, change the example to client-server relation.
Let's imagine a client has 8 GB RAM, and server has 16 GB RAM.

Streams and Buffers

8 GB RAM

2TB File

16 GB RAM

THAPA TECHNICAL – NODEJS YOUTBE SERIES

If a client wants to upload a 2 TB file to the server, then what it can do is give whole 2 TB file to server, Then server can store it in disk. But is that possible?

Streams and Buffers

8 GB RAM

16 GB RAM

2TB File

THAPA TECHNICAL – NODEJS YOUTBE SERIES

It’s not possible because client first has to take whole 2 TB file in their RAM, then upload it to server, then server should initially store whole 2 TB file in RAM, then store it in disk.

Streams and Buffers

8 GB RAM

2TB File

16 GB RAM

THAPA TECHNICAL – NODEJS YOUTBE SERIES

Neither client has 2 TB RAM, nor the server can handle it. In that case, we have to split the file Into chunks and server should take the file in chunk and write into disk at the same time.

Streams and Buffers

8 GB RAM

2TB File

16 GB RAM

THAPA TECHNICAL – NODEJS YOUTBE SERIES

For that, we use streams and buffers in Node.js

Streams and Buffers

- Streams in Node.js are a way to handle continuous flow of data.
- They enable reading or writing data piece by piece instead of loading the entire data into memory.
- A Buffer in Node.js is a temporary storage area for binary data.
- Buffers work as chunks of data, enabling efficient data manipulation in streams.
- It helps to decrease memory consumption of your web server.
- There are four types of streams in Node.js: Writeable, Readable, Duplex, and Transform. But we won't cover everything in detail, as that would take a separate course if we wanted. We will teach what is needed for now.

HTTP MODULE

HTTP Module – Node.JS

The http module in Node.js allows developers to create an HTTP server and handle client requests and server responses. It provides methods and properties to work with HTTP requests and responses, enabling the creation of REST APIs, web pages, and other networked applications.

A web server is software or hardware that serves web content (HTML, CSS, JavaScript, etc.) to clients (usually browsers) over the internet or an intranet. It uses protocols like HTTP/HTTPS to handle requests and responses.

THAPA TECHNICAL – NODEJS YOUTBE SERIES

HTTP Module – Node.JS

Step 1: Importing the http Module

- The http module is built into Node.js. Use require('http') to import it into your script.

Step 2: Using http.createServer()

- The createServer method initializes an HTTP server.
- It takes a callback function as an argument with two parameters:
- req: the incoming request object.
- res: the outgoing response object.

Step 3: Handling Requests

- The req.url property provides the path requested by the client.
- Based on req.url, send different responses using res.end().

Step 4: Starting the Server

- Use the listen() method to specify the port the server will run on.
- Optionally, pass a callback to run code when the server starts (e.g., console.log()).

Introduction to package.json

- **package.json** is a configuration file used in Node.js projects.
- It contains metadata about the project and information on project dependencies.
- Go to your project folder and use **npm init** to initialize the project or to create **package.json**
- Name, Version, Description ("name", "version", "description"):
  - Specifies the name and version of the project.
  - Helps uniquely identify and version the project.
- Entry Point ("main"):
  - Specifies the main entry point file for the application.
  - The file executed when the application starts.

Introduction to package.json

- Scripts:
  - Contains custom scripts for various tasks (e.g., running tests, starting the server).
  - Provides shortcuts for common development tasks.
- Author, License, and More:
  - Information about the project's author, license, repository, and other metadata.
  - Useful for documentation and collaboration.
- You can also use **npm init -y** to create package.json quickly.

ES Modules in Node.js

- ES Modules (ECMAScript Modules) allow you to use import and export syntax. They have been available in Node.js since version 12.
- To enable ES Modules, you can either:
  - Name your file with the .mjs extension, or
  - Set "type": "module" in your package.json. (Recommended)
- Use import and export instead of require and module.exports.
- After Node.js v14.8, you can use top-level await when ES Modules are enabled.
- We'll use ES Modules in this course as it's part of the ECMAScript standard and a more modern approach. Don't worry, even if a package uses CommonJS, you can import it into ES Modules. However, importing ES Module packages into CommonJS may cause issues, as some packages have switched exclusively to ES Modules.

ES Modules in Node.js

Feature	ES Modules (`import/export`)	CommonJS (`require/module.exports`)
Default Export	`export default ...`	`module.exports = ...`
Named Export	`export const func = () => {};`	Not supported (must destructure manually).
Import Syntax	`import ...`	`const ... = require(...)`

Export Type	Term	Use Case
Named Export	"Named Export"	When you need to export multiple functions, constants, or variables.
Default Export	"Default Export"	When a module has one primary export (e.g., a single function or class).
Aggregated Export	"Export Aggregation"	When grouping multiple exports into one statement for modularity.

npm

Introduction to npm

Introduction to npm

- npm is a popular package manager which comes bundled with Node.js.
- It is a CLI tool used to install, update, and remove external packages.
- You can also create your own package and publish it on npmjs.com registry.
- Do not confuse npm CLI with npmjs.com, as it's a registry where most of the packages of Node.js are saved.
  - There are alternative Node.js package managers registry like JSR, which we will discuss in future.
- Fun facts:
  - npm shouldn't be written in capitalized form unless you have everything as capital.
  - npm doesn't stand for Node Package Manager even though many people refer to it as that. It is a recursive bacronymic abbreviation for "npm is not an acronym".

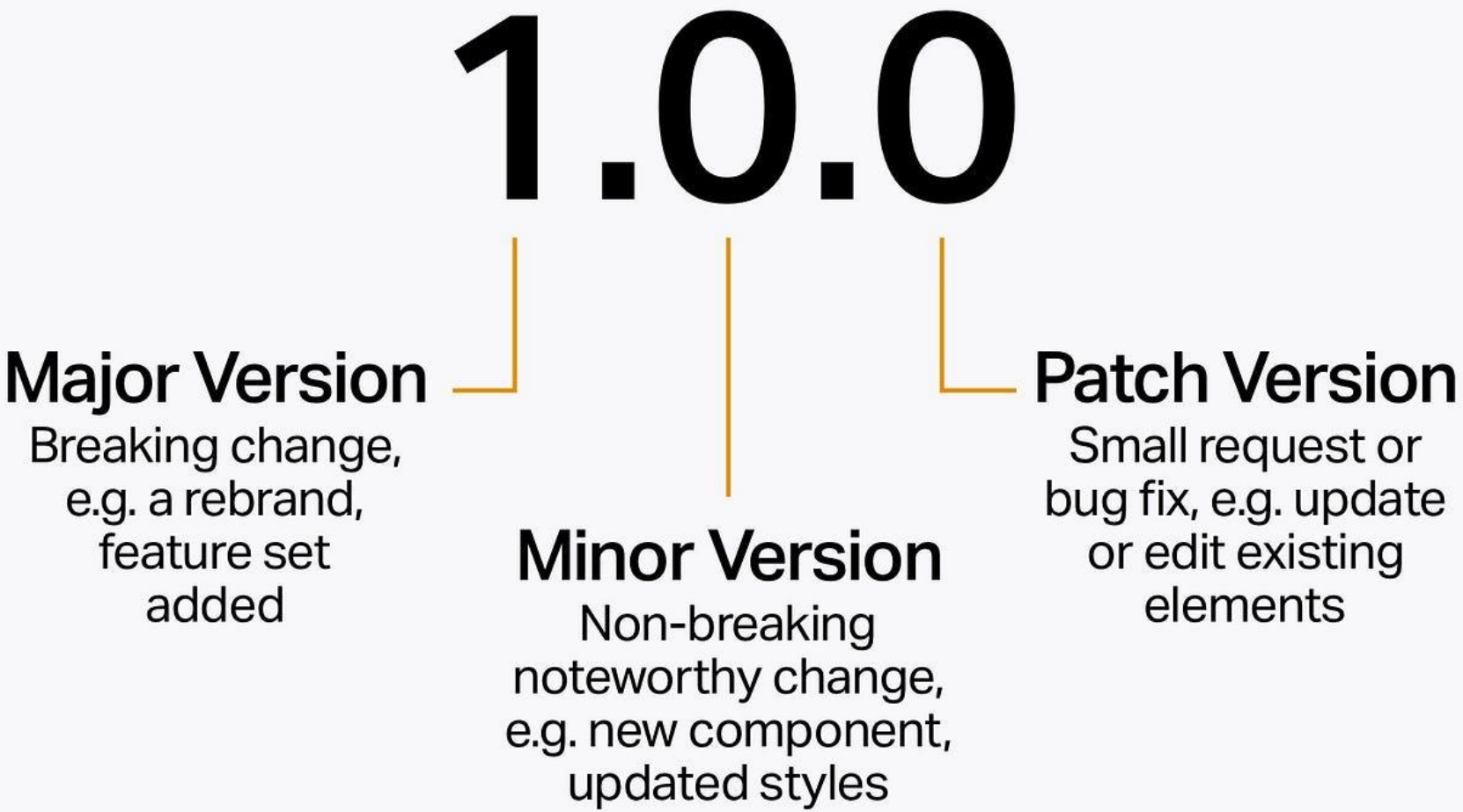
npm commands

- Before following these commands, make sure you initialized your project.
- npm install <package-name>
  - Alternatively, you can use **npm i**
- After installation, you will notice a **node_modules** folder and **package-lock.json**
  - node_modules is what stores all the installed packages. It's usually heavy, so make sure to include it in .gitignore so that it won't get pushed on version control and **avoid it while sharing with others.**
  - You will notice that there are some packages which you didn't install, it's because the package that you installed depend on those third-party packages.
  - package-lock.json includes exact version of all packages that you install. It makes sure to prevent breaking changes in newer versions of package.
- Now, you can use the package by importing normally as you do with core modules.

THAPA TECHNICAL – NODEJS YOUTBE SERIES

- While importing, first Node.js checks for core modules, then files or folders, and at last looks inside node_modules.

Semantic Versioning System



Most of the npm packages use Semantic Versioning System or SemVer. Note: Some packages like typescript, react-native don't follow it.

Symbols in dependency versions

Symbol	Meaning	Example	Resolves To
^ (Caret)	Minor and patch updates allowed	^4.17.1	4.18.0, not 5.0.0
~ (Tilde)	Only patch updates allowed	~4.17.1	4.17.2, not 4.18.0
Exact	Fixed version	4.17.1	4.17.1 only
>	Greater than	>4.17.1	4.18.0, 5.0.0
<	Less than	<4.17.1	4.16.0, not 4.17.1
>=	Greater than or equal to	>=4.17.1	4.17.1, 5.0.0
<=	Less than or equal to	<=4.17.1	4.17.1, 4.16.0
*	Any version	*	4.0.0, 5.0.0, etc.
Range	Acceptable range	4.16.0 - 4.17.1	4.16.0, 4.17.1
x	Wildcard for minor/patch versions	4.x	4.16.0, 4.18.1

Note: **^4.17.1** is equivalent to **4.x**, and **~4.17.1** is equivalent to **4.17.x**.

npm commands

- npm install
  - Even if you delete node_modules, you can use this command to install all of them again. It uses version specified in package-lock.json or package.json
- npm list
  - Because of symbols, the versions specified in package.json might not be installed.
  - To see the exact versions of all packages installed in your project.
  - Use **-a** flag to see whole list.
- npm view <package-name>
  - To see details of a package that you installed including version, license, author, and so on.
- npm view <package-name> <package.json-property>
  - Example: npm view express version
  - You can use it to view any property from package.json of a package that you installed.
- npm view <package-name> versions
  - You can use it to see all versions of a package.

npm commands

- npm install <package-name>@<version>
  - Examples:
    - npm install express@4.0.0 // ^4.0.0
    - npm install express@4.0.0 --save-exact // To install exact version.
    - npm install express@~4.0.0
    - npm install express@4.2.x
  - You can use any symbols while specifying version.
- npm outdated
  - It shows outdated packages in your project.
  - Current: The version of the package currently installed in your project.
  - Wanted: The latest version that satisfies the version range defined in your package.json. It basically uses the symbols to get it.
  - Latest: Absolute latest version of the package.
- npm remove <package-name>

npm commands

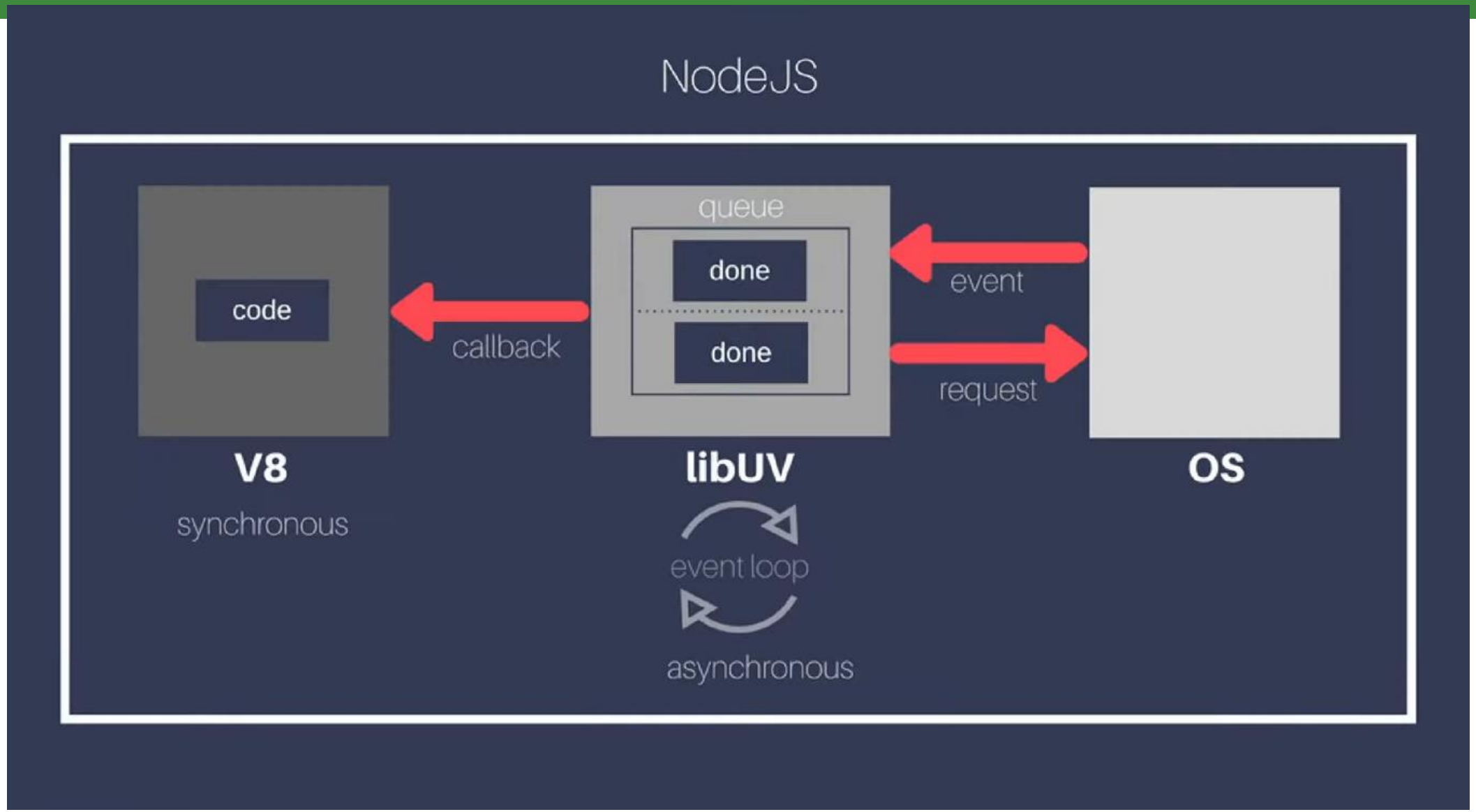
- npm update
  - This updates all the packages in your project, but it follows the range defined in package.json. It doesn't update to absolute latest version.
- npx npm-check-updates
  - npx is a CLI tool that comes with npm.
  - It is used to execute a package without requiring you to install globally or locally.
  - It is useful for temporary usage of a package.
  - npm-check-updates is a package which you can use to upgrade your packages to absolute latest versions.
  - Use **-u** flag at the end to update the packages after reviewing.
  - This only updates package.json, then you can use **npm install** to update the packages.

Global packages

- Up until now, we have been installing packages locally.
- To install package globally,
  - npm install -g npm-check-updates
  - On Linux, and macOS, you might have to prefix it with **sudo**
  - This package gets installed globally, and you can access from anywhere.
  - You can try running **npm-check-updates** command anywhere in any projects without using **npx**.
  - **npm-check-updates** also have an alias named **ncu**, which you can also use.
- Fun fact: npm is itself a global package which means if you need to update npm, you will do: **npm install -g npm**
- npm outdated -g
  - To see outdated global packages.
- npm update –g <package-name>
- npm remove -g <package-name>

Development dependency

- Development dependencies are the packages that aren't needed for functioning of your project in production.
- This can be for formatting, linting, testing, and so on.
- If you use **npm install --production**, those packages won't be installed, but they will be installed if you don't use production flag.
- npm install -D eslint
  - This will install eslint as a development dependency.
  - You will see it in separate property inside package.json named "**devDependencies**"


NodeJS
code
V8
synchronous
callback
queue
done
done
libUV
event loop
asynchronous
event
request
OS