PART 1

MASYNCTECH
SCHOOL

# FULLSTACK WEB DEV

# NODEJS

## BACKEND DEVELOPMENT


PART 1
node JS
JS
MASYNCTECH
MASYNCTECH
www.masynctech.com

MASYNCTECH
SCHOOL

# WHAT'S NODEJS?

node JS

CORE CONCEPTS

# WHAT IS NODE.JS?

MASYNCTECH
SCHOOL

## High-Level Explanation

- Node.js: Server-side JavaScript runtime
- Uses V8 JavaScript engine
- Facilitates scalable network app development

## Basic Explanation

- Analogy: Traditional vs. Node.js server languages
- Traditional: Cashier takes one order, processes, next.
- Node.js: Cashier takes multiple orders, processes simultaneously.

## Important Rules

- Utilize Node.js's asynchronous nature.
- Modularize for maintainability.
- Effective error handling (loosely typed JS).
- Keep Node.js and dependencies updated.
- Use well-maintained, documented tools/libraries.

## Deep Dive

- Node.js unifies server-side and client-side JavaScript.
- Asynchronous, event-driven for efficiency.
- Handles multiple connections simultaneously.
- Single-threaded event loop model for concurrency.

## When to use?

- Node.js excels in real-time, collaborative apps (gaming, chat).
- Ideal for APIs, especially RESTful, due to I/O efficiency.

MASYNCTECH
SCHOOL

# IMPORTANCE IN MODERN WEB DEV

# NODEJS

CORE CONCEPTS

# IMPORTANCE IN MODERN WEB DEVELOPMENT

MASYNCTECH SCHOOL

## High-Level Explanation

- Node.js: Unified client-server JavaScript
- Facilitates real-time apps
- Promotes high performance, scalability
- Redefined modern web development

## Deep Dive

- Node.js Advantages:
  - Unified Language: JavaScript for both sides.
  - Real-Time Apps: Great for chat, gaming, collaboration.
  - High Performance: V8 engine for fast execution.
  - Scalability: Event-driven, clustering for load balancing.
  - NPM Ecosystem: Rich library/tool repository.
  - Cross-Platform: Runs on Windows, MacOS, Linux.

## Basic Explanation

- Analogy: Node.js as a Swiss Army knife
- One language (JavaScript) for all tasks
- Speedy performance
- Powers real-time apps (games, chats)
- Scalable to evolving needs
- Abundance of ready-to-use tools


## Summary

- Node.js: Essential in modern web dev
- Unifies languages, aids real-time apps
- High performance, scalability
- Rich package ecosystem
- Cross-platform versatility
- Best practices ensure secure, effective use.

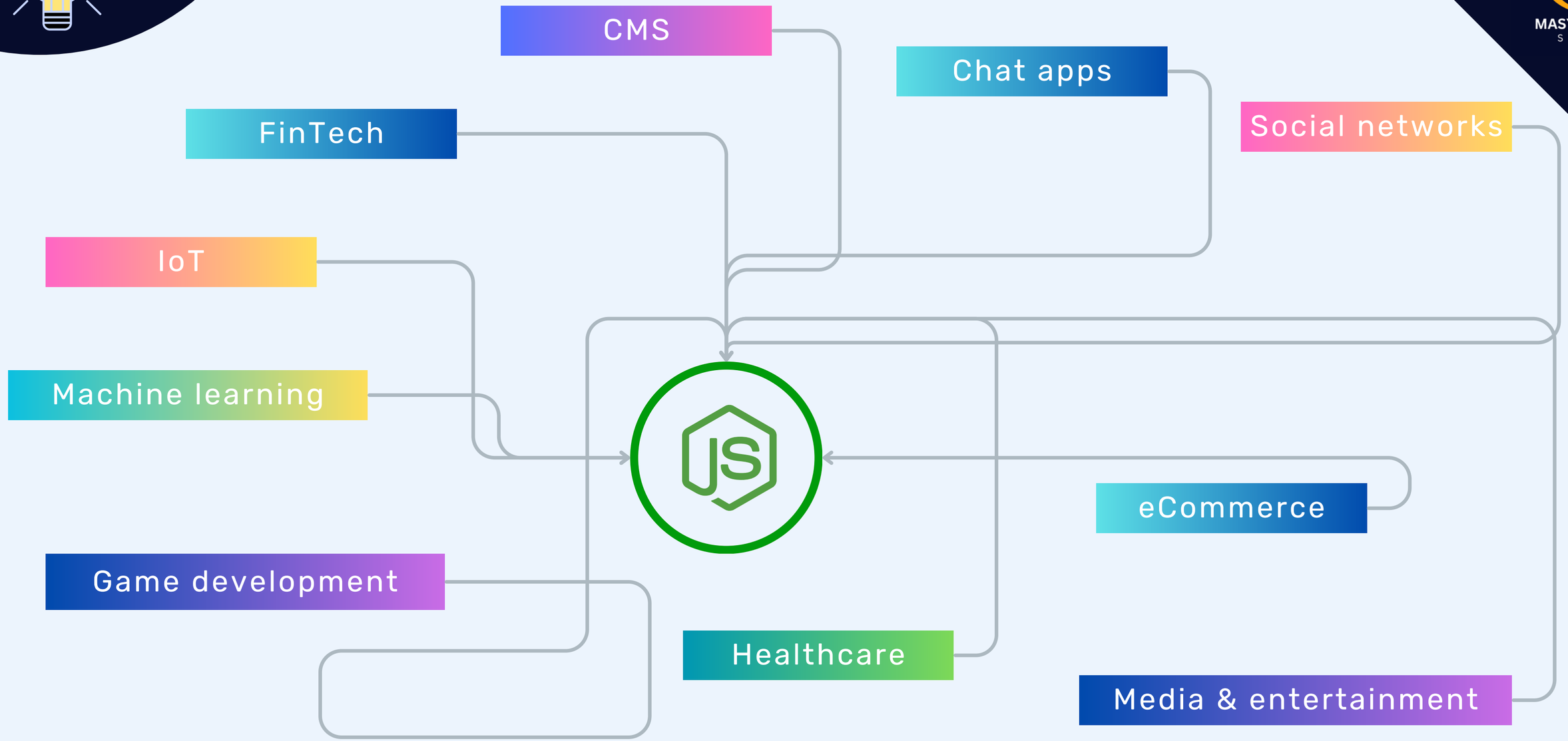
MASYNCTECH
SCHOOL

# NODEJS
# INDUSTRY USE-CASES

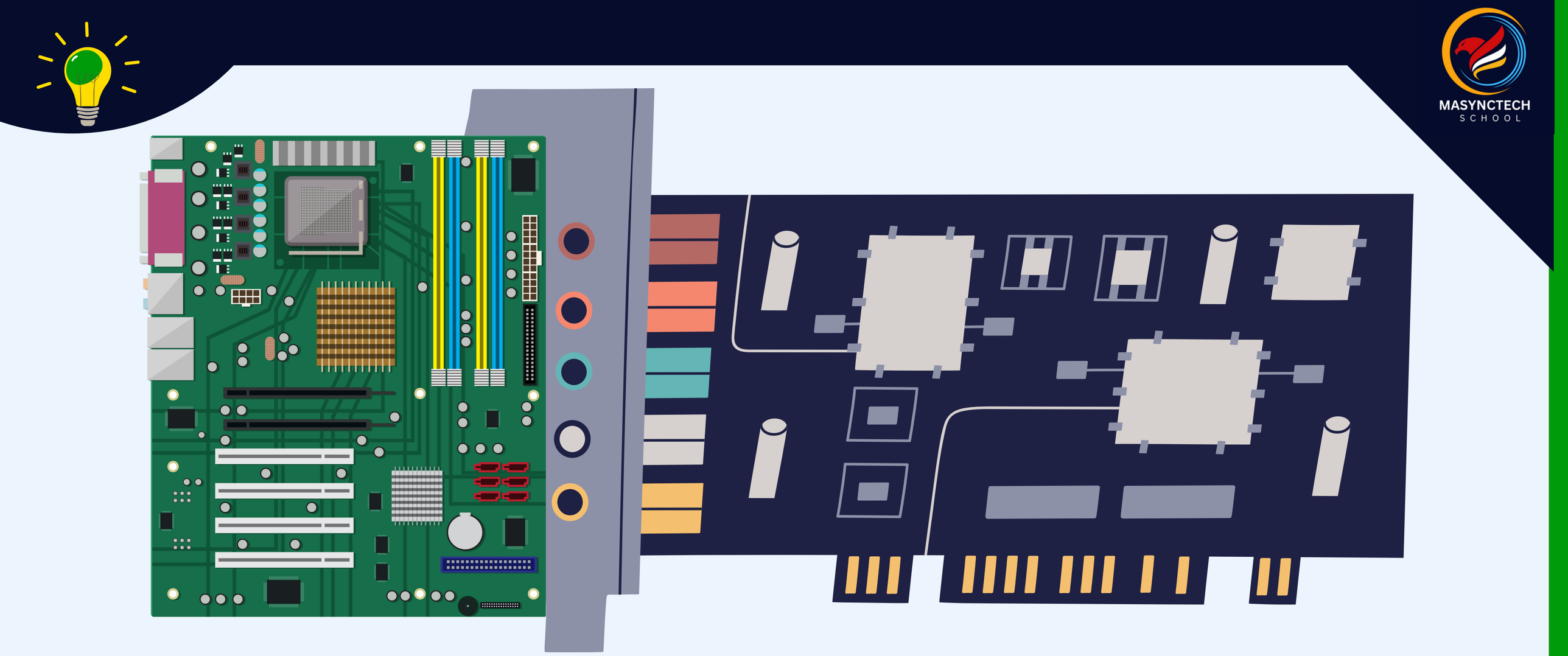
CORE CONCEPTS

# INDUSTRY USE-CASES

MASYNCTECH
SCHOOL




# CORE CONCEPTS



MASYNCTECH
SCHOOL

node JS

# CORE CONCEPTS

MASYNCTECH
SCHOOL

# COMMAND-LINE INTERFACE (CLI)


node JS

CORE CONCEPTS

# COMMAND-LINE INTERFACE

MASYNCTECH SCHOOL

## High-Level Explanation

- CLI (Command-Line Interface): Text-based computer interaction
- Uses commands typed into a prompt
- In NodeJS, CLIs automate tasks, manage files, more
- NodeJS modules like `fs`, `os`, and `process` aid CLI creation
- Contrasts with GUIs (Graphical User Interfaces)
- GUIs user-friendly, CLI commands needed
- CLIs efficient, powerful for scripting, automation

## Basic Explanation

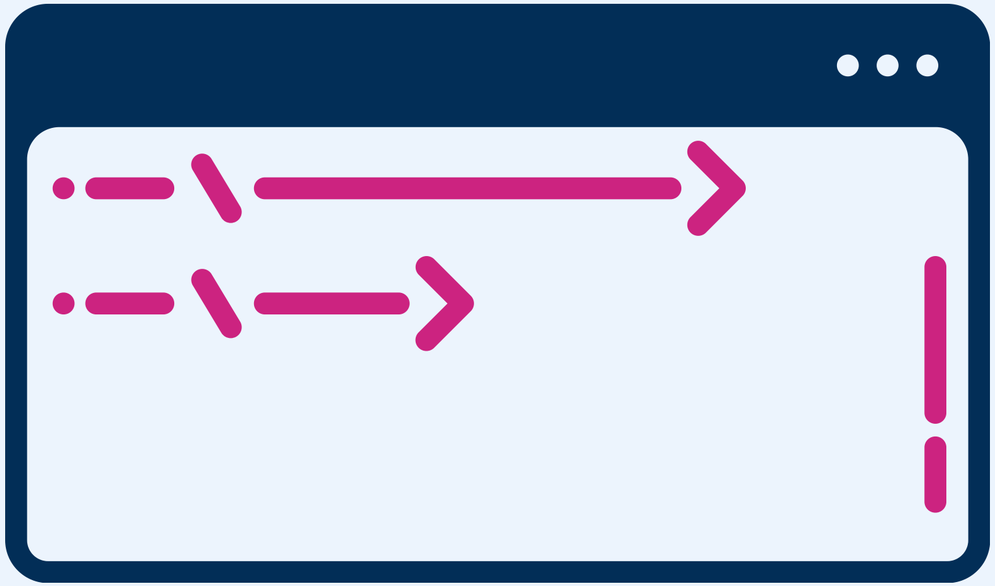
- Analogy: CLI as ordering food from a chef
- Directly communicate your specific preferences
- Describe what you want, how you want it, ingredients included

## When to use?

- CLI advantages:
  - Simplifies repetitive tasks through scripting
  - Useful for system administration, resource management
  - Many development tools are CLI-based
  - Automation with command-line scripts
  - Efficient for troubleshooting system-level issues

MASYNCTECH
SCHOOL

# COMMAND PROMPTS



node JS

CORE CONCEPTS

# COMMAND PROMPTS

MASYNCTECH SCHOOL

## High-Level Explanation

- CLI: Interface for text commands
- "Command Prompt": Microsoft's Windows CLI application

## Deep Dive

- **CLI**: Text command interface in various OSs
- Unix-based systems use shells (e.g., Bash, Zsh)
- **Command Prompt**: Windows' native CLI
- Runs CMD commands (e.g., `dir`, `copy`)
- Uses "batch" scripting (`.bat` files)
- Coexists with more advanced interpreters like PowerShell

## Basic Explanation

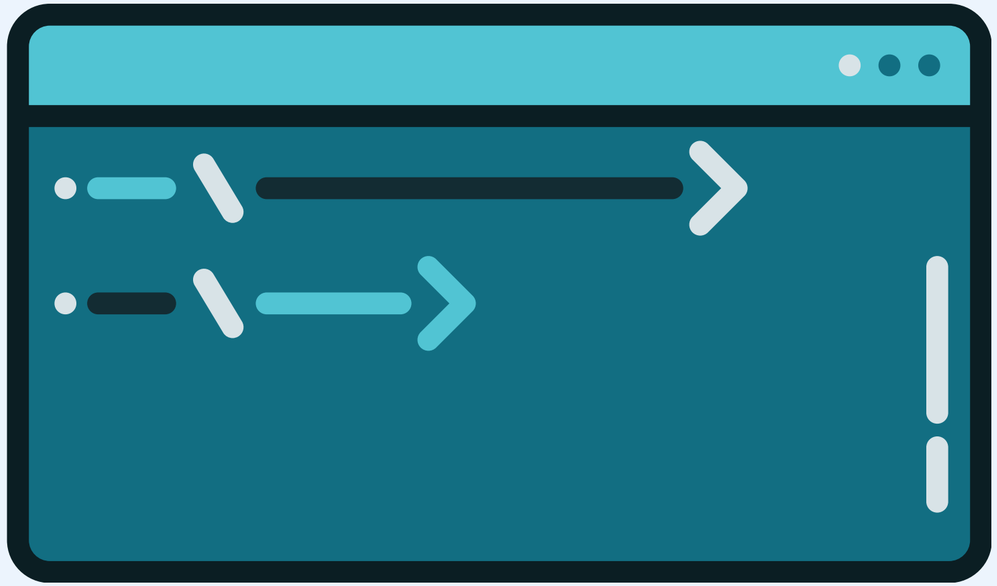
- Analogy: "CLI" as a broad category, "Command Prompt" as a specific instance
- "Cars" as a broad category, "Toyota Corolla" as a specific type
- CLI encompasses various textual interfaces
- "Command Prompt" is a specific CLI on Windows


## Summary

- "CLI" is a general text-based interface.
- "Command Prompt" is a specific Windows CLI.
- Distinction essential for clarity and cross-OS work.

MASYNCTECH
SCHOOL

# COMMON SHELL COMMANDS



node JS

CORE CONCEPTS

# COMMON SHELL COMMANDS

MASYNCTECH
SCHOOL

## High-Level Explanation

- Shell: Interface for OS services
- Unix-like systems, Linux, macOS, Windows (WSL)
- Commands entered at shell prompt for tasks

## HOW TO OPEN CLI

- Accessing shells on different OSs:
  - Windows Command Prompt: Windows + R, type cmd, or search in Start Menu
  - Windows PowerShell: Windows + X, select "PowerShell," or search in Start Menu
  - macOS Terminal: Finder → Applications → Utilities or Spotlight search
  - Linux Terminal/Console: Keyboard shortcut (Ctrl + Alt + T) or application menu search

## COMMON SHELL COMMANDS

**- Navigation:**

- `pwd`: Print current working directory.
- `cd <directory_name>`: Change to specified directory.
- `cd ..`: Navigate up one directory.
- `cd`: Navigate to home directory.

**- File Management:**

- `ls` (Linux/macOS) / `dir` (Windows): List current directory contents.
- `touch <filename>` (Linux/macOS): Create empty file.
- `mkdir <directory_name>`: Create directory.
- `rm <filename>` (Linux/macOS) / `del <filename>` (Windows): Delete file.
- `rm -r <directory_name>` (Linux/macOS) / `rmdir /s <directory_name>` (Windows): Delete directory and contents.
- `cp <source> <destination>` (Linux/macOS) / `copy <source> <destination>` (Windows): Copy files.
- `mv <source> <destination>` (Linux/macOS) / `move <source> <destination>` (Windows): Move or rename files.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# INSTALLATION
# NODEJS

CORE CONCEPTS

# INSTALLING NODE.JS

MASYNCTECH SCHOOL

## High-Level Explanation

- First step: Install Node.js for app development.
- Several methods: Package managers, official website installer.
- Installation process is straightforward.

## Deep Dive

- Methods for Node.js installation:
  - **Package Managers**: Linux users use `apt` (Ubuntu) or `yum` (Fedora) for ease of updates.
  - **Official Website:** Cross-platform, includes Node.js and NPM.
  - **Version Management Tools**: `nvm`, `n` for managing multiple Node.js versions.

## Important Rules

- Verify system requirements first.
- Install stable Node.js versions unless needed for development.
- Consider installing NPM for package management convenience.

## When to use?

- Consider package managers for Linux and command-line expertise.
- Official website installer for GUI-based simplicity.
- Use version management tools if switching Node.js versions frequently.

## Summary

- Verify installation with `node -v` and `npm -v`.
- Older projects may need specific Node.js versions.
- Version manager useful for such cases.

MASYNCTECH
SCHOOL

# CONFIGURING TEXT EDITOR

CORE CONCEPTS

MASYNCTECH
SCHOOL

# NODEJS

# YOUR FIRST APP

node JS

CORE CONCEPTS

MASYNCTECH
SCHOOL

# NODEJS

# REPL

## (READ-EVAL-PRINT-LOOP)

CORE CONCEPTS

# NODE.JS REPL (READ-EVAL-PRINT-LOOP)

MASYNCTECH SCHOOL

## High-Level Explanation

- Node.js REPL: Interactive JavaScript shell
- Run code, see immediate results
- Useful for debugging, testing, calculations

## Basic Explanation

- Node.js REPL: Smart JavaScript calculator
- Type code, it runs and shows results
- A chat with your computer for testing, fixing code

## Important Rules

- Exit with `.exit` when finished
- Note differences in behavior with async operations
- Utilize special commands like `.save` and `.load` for code management

## Deep Dive

- REPL components:
  - Read: User input
  - Eval: Evaluate JavaScript expression
  - Print: Output result
  - Loop: Await more input, create a loop
- Handles expressions, variables, functions, Node.js modules
- Interactive JavaScript playground in terminal

## When to use?

- Use cases for Node.js REPL:
  - Quick debugging/testing of JavaScript
  - Learning tool for newcomers to JavaScript/Node.js
  - Rapid calculations/logic without a full project setup

## Summary


- Starting Node.js REPL: Type `node`, no arguments, hit Enter
- Note that REPL is for temporary code; variables/functions lost on exit
- Case-sensitive, like JavaScript; `x` and `X` differ

MASYNCTECH
SCHOOL

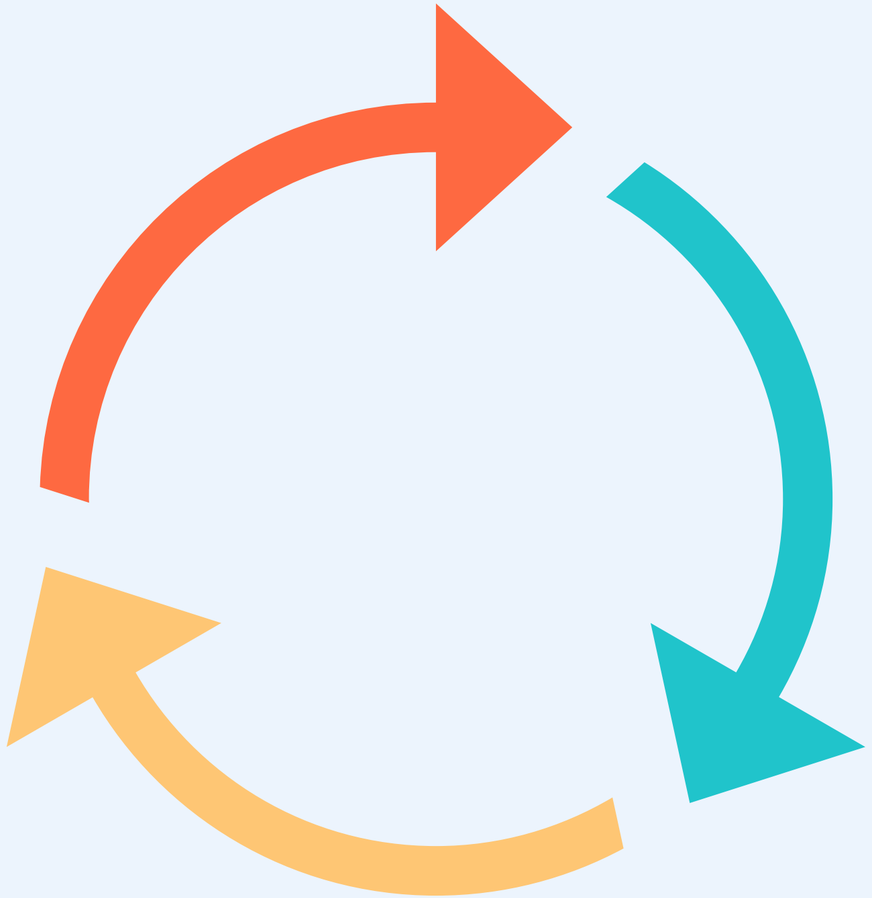
# NODEJS
# CORE CONCEPTS

CORE CONCEPTS

MASYNCTECH
SCHOOL

# NODEJS

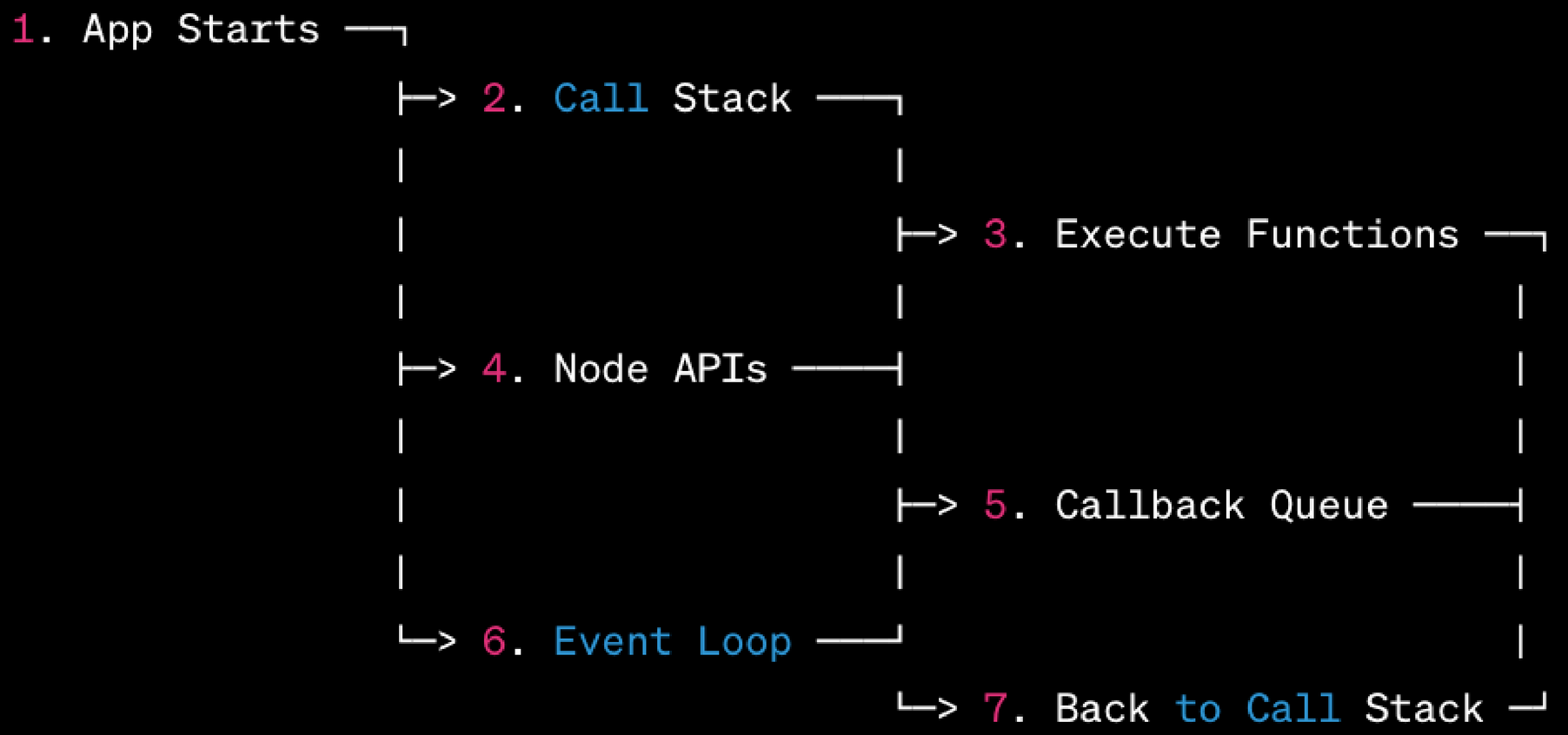
# EVENT LOOP



node JS

CORE CONCEPTS

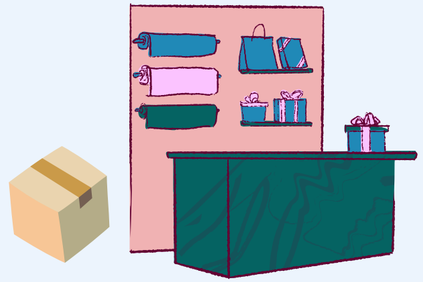
# EVENT LOOP


1. App Starts
2. Call Stack
3. Execute Functions
4. Node APIs
5. Callback Queue
6. Event Loop
7. Back to Call Stack

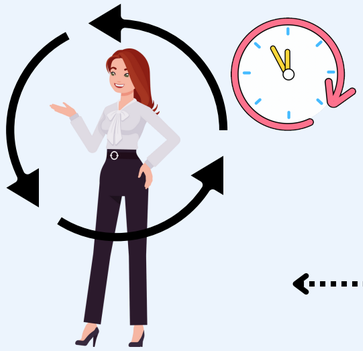
# THE NODEJS EVENT LOOP

MASYNCTECH
SCHOOL

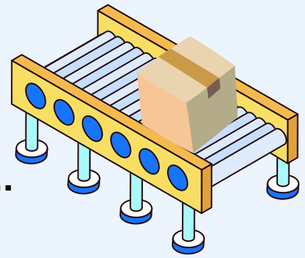
checkout counter(Call Stack)



wrapping station (NODE APIs)



A manager (Event Loop)



conveyor belt (Callback Queue)

# EVENT LOOP

MASYNCTECH SCHOOL

## High-Level Explanation

- Event Loop: Core Node.js concept
- Enables non-blocking I/O
- Handles multiple connections
- Delegates operations and subscribes to events

## Basic Explanation

- Analogy: Event Loop as a DJ at a party
- Playing a song (event), interacting (tasks)
- Playlist (queue), playing when current song ends
- Event Loop checks tasks, performs, and checks again

## Important Rules

- Avoid CPU-intensive tasks to prevent Event Loop blockage.
- Split large tasks into smaller ones to avoid blockage.
- Use asynchronous code for Event Loop efficiency.
- Careful error handling in async code to prevent unhandled exceptions.

## Deep Dive

- Event Loop enables asynchronous behavior.
- Monitors Call Stack and Callback Queue.
- When stack is empty, executes first event from queue.
- Enables non-blocking I/O in Node.js.
- Node.js is single-threaded but performs concurrent operations through Event Loop and non-blocking nature.

## When to use?

- Event Loop fundamental to Node.js
- Always active
- Especially valuable for high-concurrency apps
- Used in real-time apps, API servers, I/O-intensive apps

## Summary

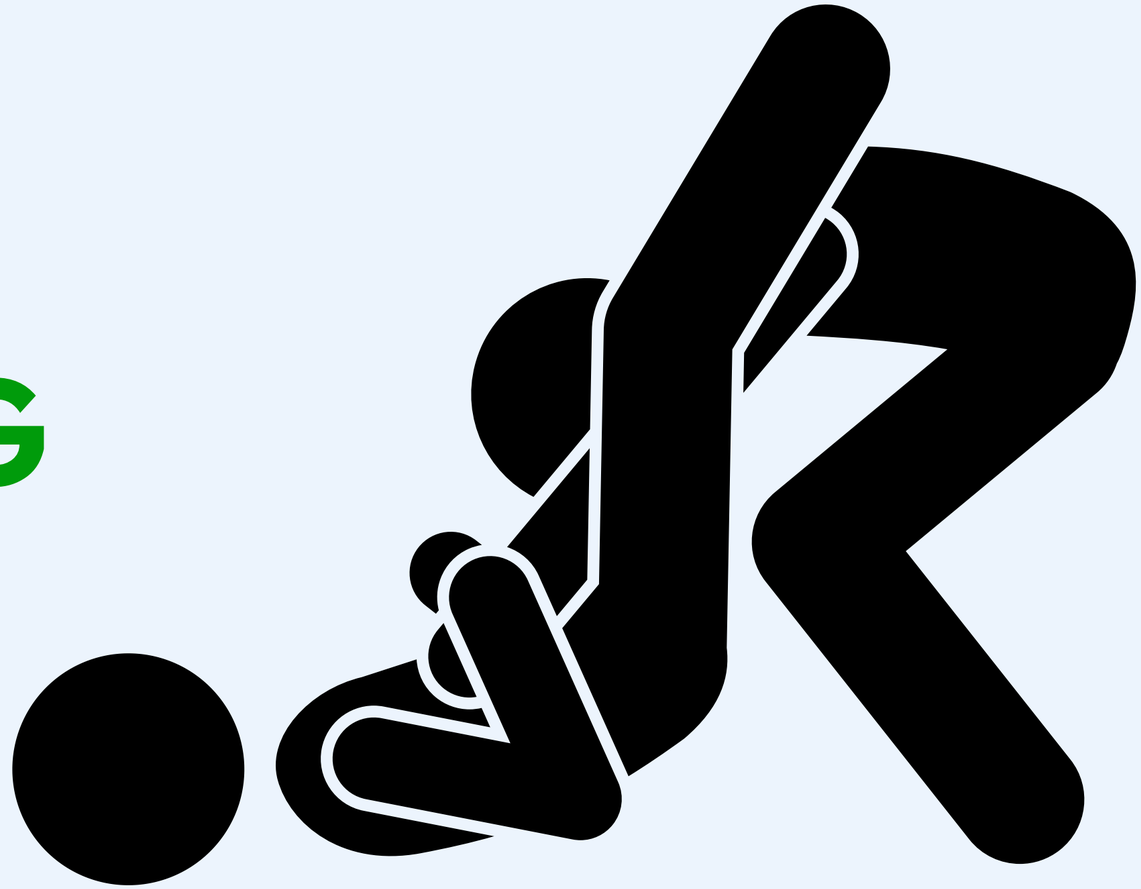

- Event Loop: Vital in Node.js for concurrent operations
- Handles without multiple threads, offloading tasks, events
- Valuable for high-concurrency apps
- Optimized through task breakdown and CPU avoidance

MASYNCTECH
SCHOOL

# NODEJS

## NON-BLOCKING

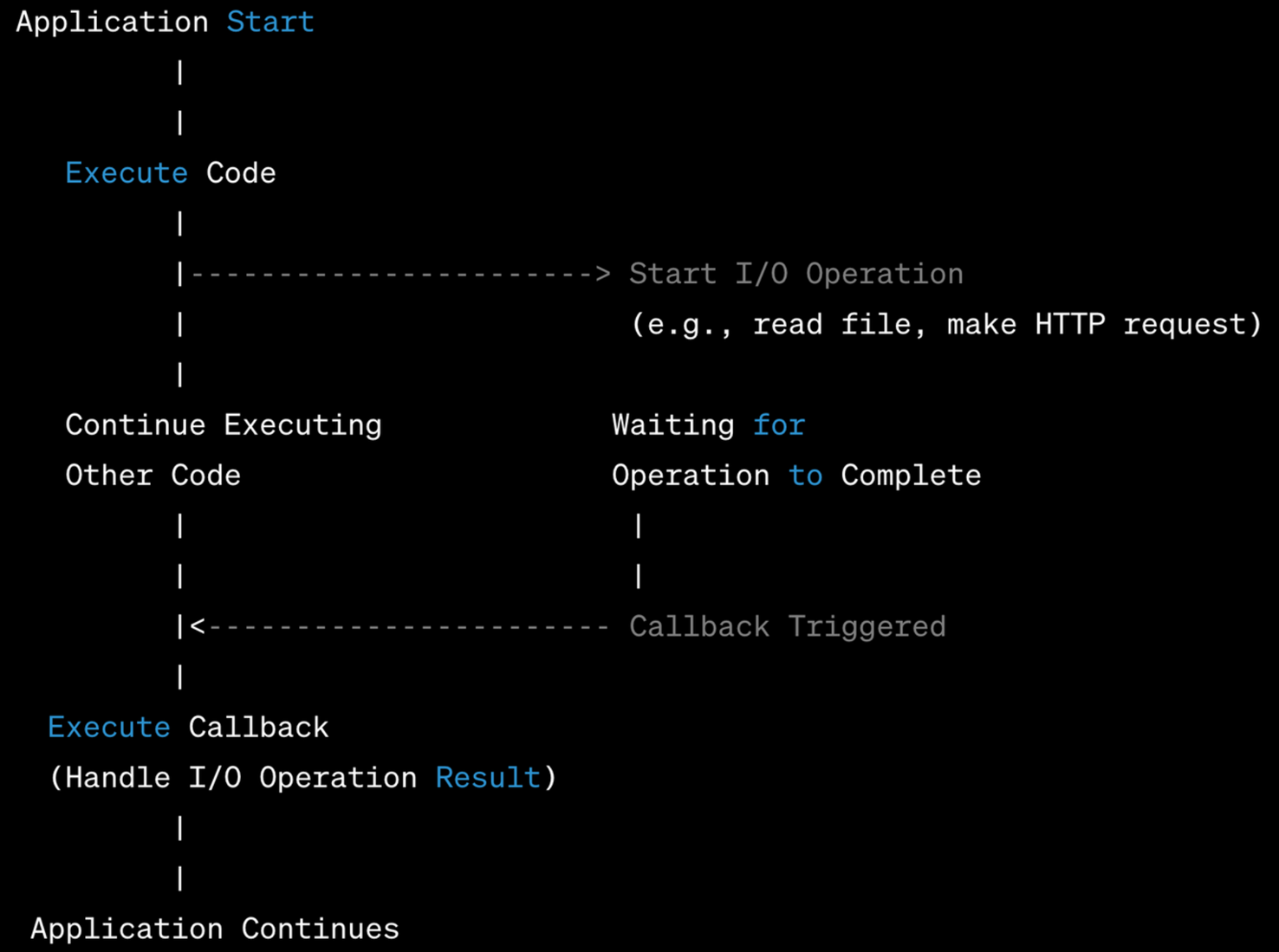
# I/O



node JS

CORE CONCEPTS

# NON-BLOCKING I/O

# NON-BLOCKING I/O

MASYNCTECH SCHOOL

## High-Level Explanation

- Allows processing to continue before transmission finishes
- Enhances efficiency and performance in Node.js

## Deep Dive

- Blocking (synchronous) JS: Halts until operation done
- Inefficient, especially for time-consuming I/O
- Non-blocking I/O in Node.js: Initiates, continues, returns
- Achieved with callbacks, promises, async/await
- Ideal for single-threaded Node.js, maximizes concurrency

## Basic Explanation

- Analogy: Restaurant scenario
- Blocking I/O: Order and wait idly for food
- Non-blocking I/O: Order and multitask while waiting

## When to use?

- Non-blocking I/O crucial for high concurrency needs
- Used in real-time apps, APIs, data streaming
- Keeps app active during I/O waits, avoids idle time

## Important Rules

- Leverage async code (callbacks, promises, async/await) for non-blocking I/O
- Handle errors meticulously to prevent crashes or inconsistencies
- Avoid mixing blocking and non-blocking code for clarity, performance
- Thoroughly test for high concurrency and real-world scenarios

## Summary

- Node.js relies on non-blocking I/O for efficiency and concurrency.
- Crucial in high concurrency situations.
- Requires proper async coding and error handling.
- Key to Node.js performance and scalability.

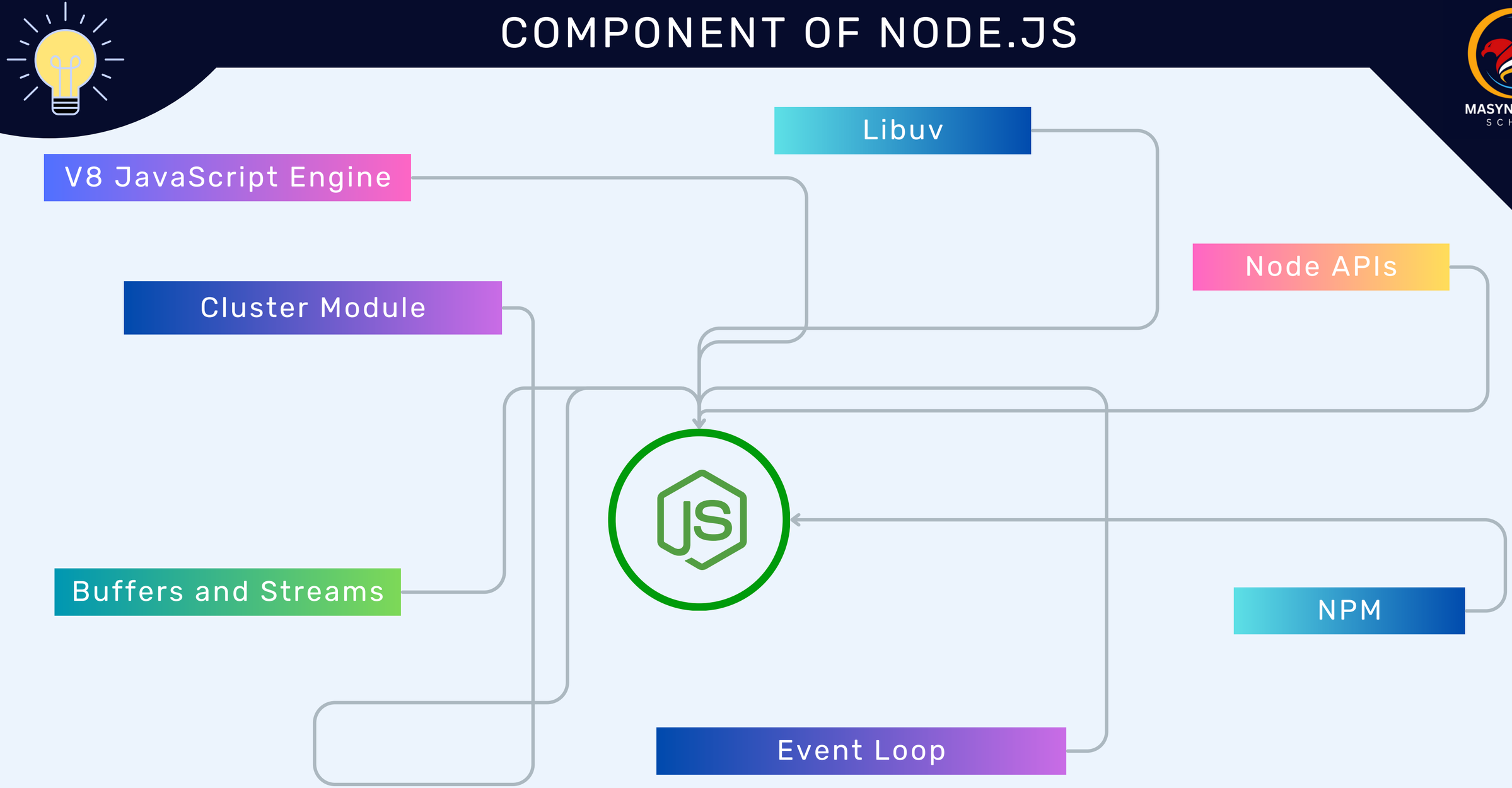
MASYNCTECH
SCHOOL

# NODEJS
# COMPONENTS

node JS

CORE CONCEPTS

# COMPONENT OF NODE.JS



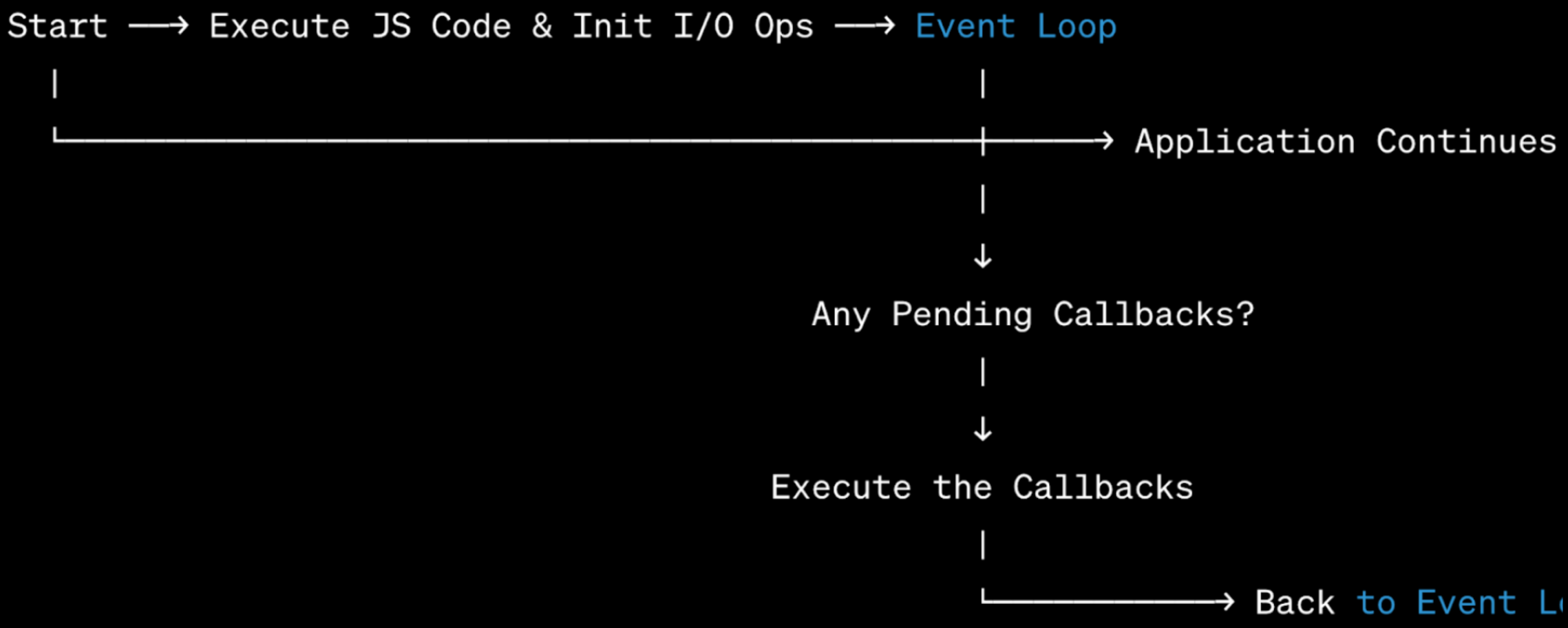
MASYNCTECH
SCHOOL

# NODEJS SINGLE THREADED

node JS

CORE CONCEPTS

# SINGLE-THREADED EVENT LOOP



# SINGLE-THREADED EVENT LOOP

MASYNCTECH SCHOOL

## High-Level Explanation

- Node.js uses single-threaded event loop, non-blocking I/O.
- Supports thousands of concurrent connections.
- Highly scalable, efficient for network apps.

## Basic Explanation

- Node.js is like a barista multitasking at a cafe.
- Takes orders (connections), starts making coffee (operations).
- Doesn't wait for coffee; takes more orders (connections).
- Efficiently handles multiple tasks simultaneously.

## Important Rules

- Avoid long, synchronous, CPU-bound tasks; block event loop.
- Split large tasks into async chunks for smooth event loop.
- Employ worker threads or child processes for CPU tasks.
- Handle errors and exceptions to prevent crashes.

## Deep Dive

- Node.js is single-threaded, uses one thread for tasks.
- Lightweight; no new thread per connection.
- CPU-bound, sync tasks run sequentially.
- Non-blocking I/O allows offloading, event handling.
- Event Loop enables concurrent connections despite being single-threaded.

## When to use?

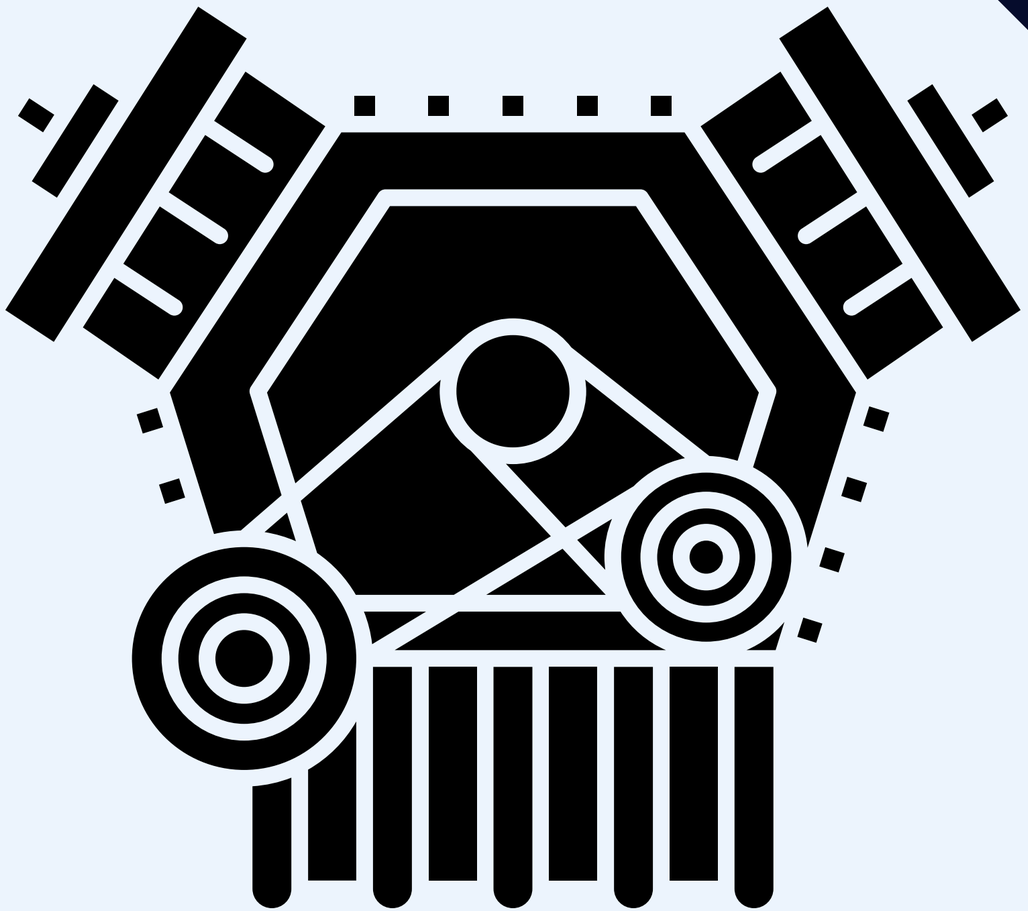
- Ideal for I/O-intensive applications
- Use cases: network, real-time, streaming, gaming, collaboration
- High concurrency and scalability needed

## Summary

- Node.js: Single-threaded event loop with non-blocking I/O
- Efficient handling of concurrent connections
- Suitable for scalable network apps
- Best practices ensure smooth operation, error handling

MASYNCTECH
SCHOOL

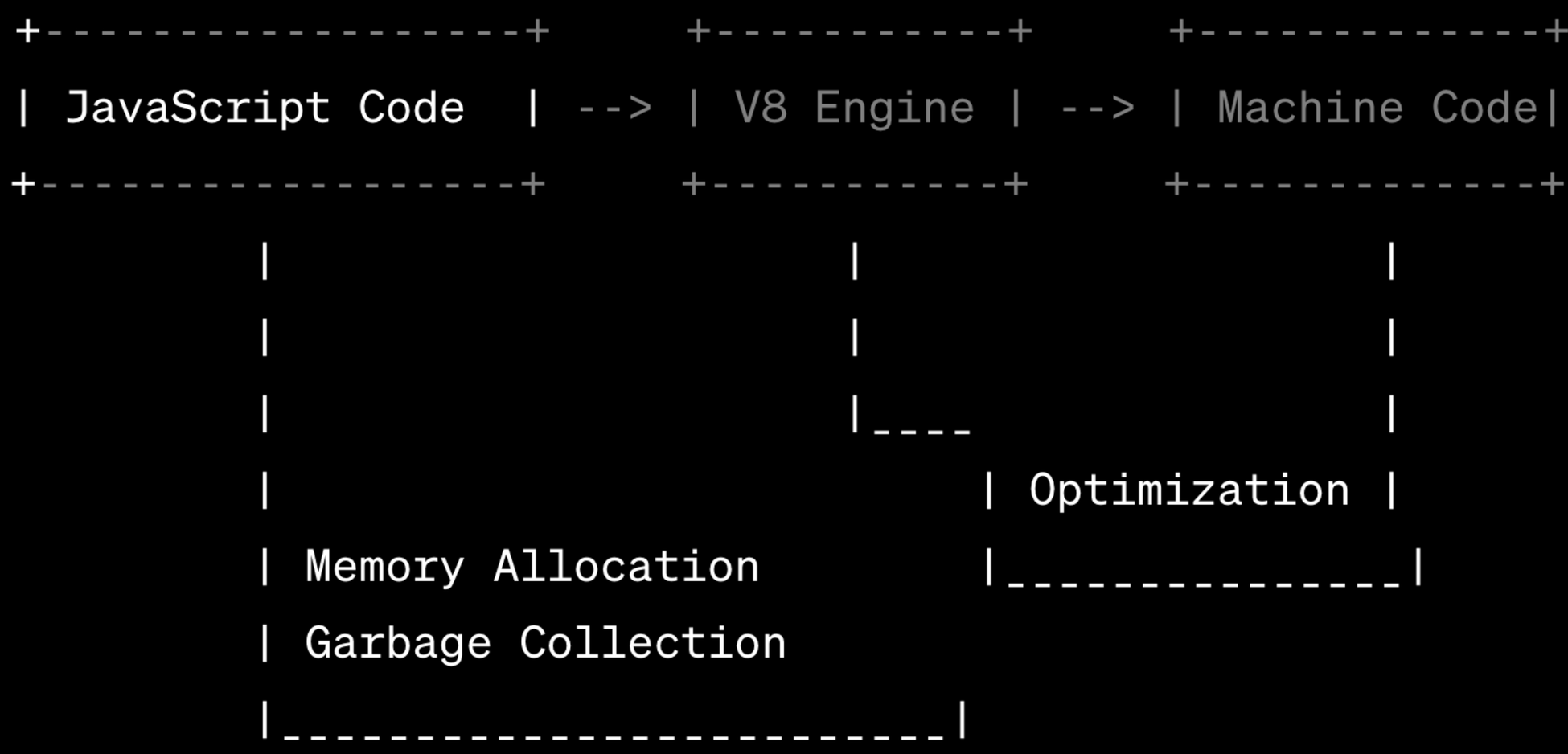
# NODEJS V8 ENGINE



node JS

CORE CONCEPTS

# HOW NODE.JS INTERPRETS JAVASCRIPT



# HOW NODE.JS INTERPRETS JAVASCRIPT

MASYNCTECH SCHOOL

## High-Level Explanation

- V8 Engine: Google's JavaScript runtime for Node.js
- Converts JavaScript to efficient machine code
- Enhances performance compared to interpretation

## Basic Explanation

- Analogy: V8 as a translator
- You speak English (JavaScript), friend speaks French (machine code)
- V8 translates quickly for computer understanding

## Important Rules

- Optimize JavaScript for V8's varied optimizations.
- Careful memory management, avoid costly object handling.
- Keep Node.js updated for V8 improvements and optimizations.

## Deep Dive

- V8 compiles JavaScript into machine code using JIT.
- Optimizes code at runtime with strategies like inline caching.
- Manages memory allocation and garbage collection for efficiency.

## When to use?

- V8 crucial for all JavaScript scenarios, notably Node.js
- Enables rapid JavaScript execution
- Ideal for real-time apps, APIs, high-performance needs

## Summary

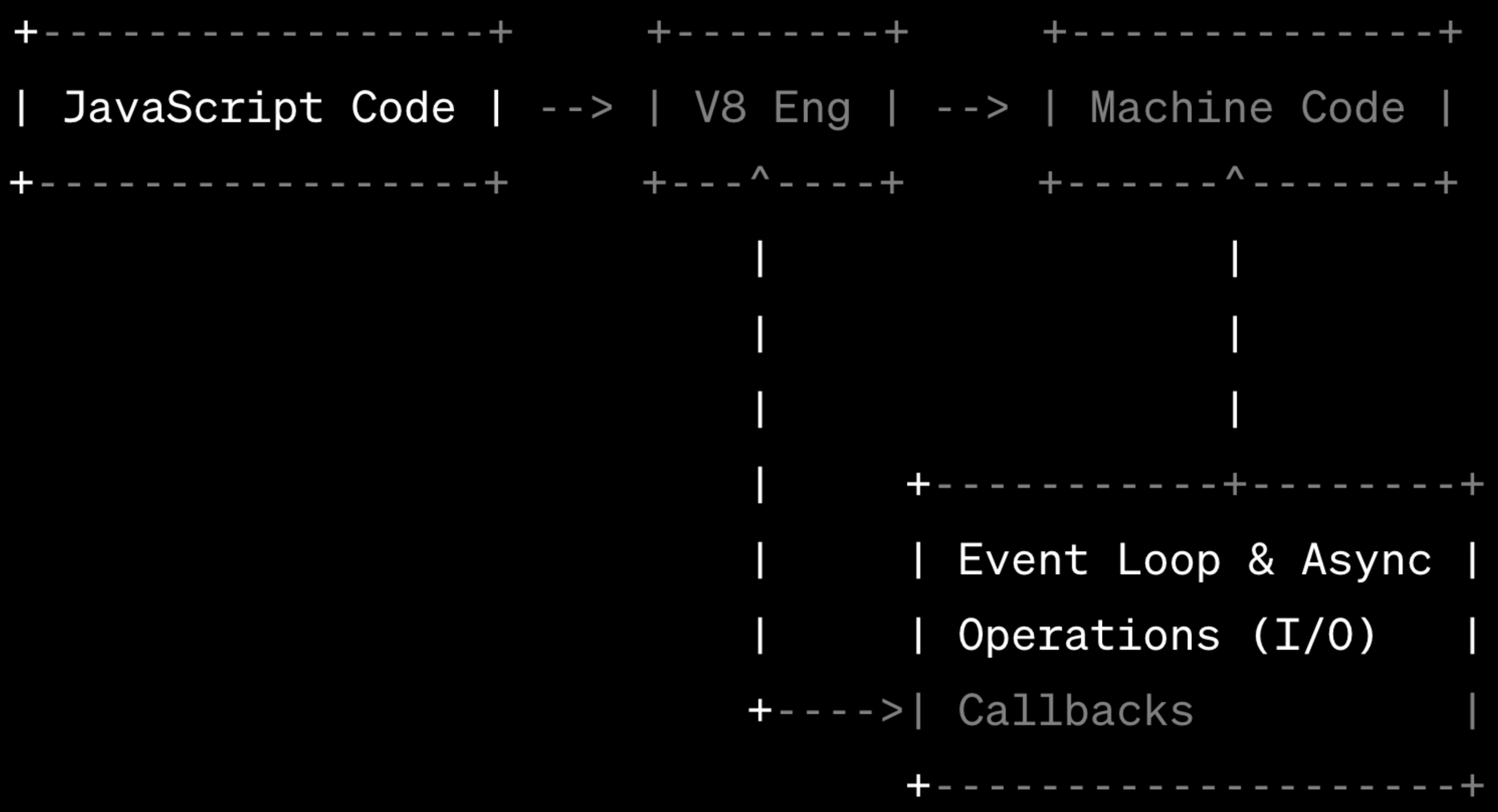
- V8 integral to Node.js for JavaScript execution.
- Converts to machine code with optimizations.
- Efficient memory management via garbage collection.
- Speedy execution in diverse applications.

MASYNCTECH
SCHOOL

# HOW NODE.JS RUNS CODE

CORE CONCEPTS

# HOW NODE.JS INTERPRETS JAVASCRIPT



# HOW NODEJS RUNS PROGRAM

MASYNCTECH SCHOOL

## High-Level Explanation

- Node.js runs programs via V8 JavaScript engine
- Event-driven, non-blocking I/O model
- Manages operations asynchronously with event loop and callbacks

## Deep Dive

- Node.js runs using V8 engine
- Event loop for non-blocking I/O in single-threaded JS
- Synchronous code executes in sequence
- Event loop checks message queue for task completion
- Enables handling thousands of concurrent connections
- Ideal for scalable network applications

## Basic Explanation

- Analogy: Node.js as an efficient chef in a busy kitchen
- Chef (event loop) handles multiple dishes (tasks)
- Non-blocking, no wait for one to finish
- Completes dishes (tasks), adds final touch (callback), serves

## When to use?

- Node.js suited for scalable network apps
- Real-time apps, APIs, microservices
- High concurrency, non-blocking crucial

## Important Rules

- Code modularity, cleanliness, maintainability
- Proper error handling to avoid crashes
- Asynchronous code for non-blocking I/O
- Keep Node.js updated for performance, security


## Summary

- Node.js uses V8 engine, event-driven, non-blocking I/O
- Event loop manages asynchronous tasks
- High concurrency for scalable network apps

MASYNCTECH
SCHOOL

# NODE.JS
# GLOBAL & PROCESS OBJECT

CORE CONCEPTS

# GLOBAL OBJECT AND PROCESS OBJECT

# GLOBAL OBJECT AND PROCESS OBJECT

MASYNCTECH SCHOOL

## High-Level Explanation

- `global` object: Global namespace, accessible everywhere
- `process` object: Global, provides info, control over Node.js process

## Basic Explanation

- Analogy: Node.js environment as a large house
- `global` object like common house features, accessible everywhere
- `process` object as a control panel, manages house functions, shows details, and provides control

## Important Rules

- **Global Object:**
  - Avoid adding properties, use modules for organization.
- **Process Object:**
  - Don't override/mutate properties, read-only except for specific methods like `process.exit()`.

## Deep Dive

- **Global Object:**
  - In Node.js, `global` is the global namespace.
  - Contains some methods/props (e.g., `global.setTimeout()`).
  - Not all expected global props (e.g., `console`) are part of it but are still global.
- **Process Object:**
  - An instance of `EventEmitter`, globally available.
  - Offers methods/props for Node.js process management.
  - Useful for handling app-level configurations and behaviors.

## When to use?

- **Global Object:**
  - Declare app-wide variables, functions.
- **Process Object:**
  - Interact with app's environment.
  - Read env vars, handle termination, command-line input.

## Summary

- `global` object: Global namespace, accessible everywhere.
- `process` object: Offers insights and controls over Node.js process, crucial for application-level interactions and configurations.

MASYNCTECH
SCHOOL

# NODE.JS
# GLOBAL OBJECTS METHODS

CORE CONCEPTS

# GLOBAL OBJECT METHODS

MASYNCTECH
SCHOOL

## High-Level Explanation

- `global`: Reference to the global object itself.
- `__dirname`: String representing the directory name of the current module.
- `__filename`: String representing the file name of the current module.
- `clearInterval()`: Clears intervals.
- `clearTimeout()`: Clears timeouts.
- `setInterval()`: Sets a function to be called repeatedly after a set period.
- `setTimeout()`: Sets a function to be called after a set period.
- `console`: Reference to the console module for stdout and stderr output.
- `process`: Provides info and control over the current Node.js process.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

CORE CONCEPTS

MASYNCTECH
SCHOOL

# NODE.JS
# PROCESS OBJECT METHODS

CORE CONCEPTS

# PROCESS OBJECT METHODS

## High-Level Explanation

- `process.exit()`: Terminate Node.js process.
- `process.cwd()`: Get current working directory.
- `process.chdir(directory)`: Change working directory.
- `process.nextTick(callback)`: Execute callback on the "next tick".
- `process.hrtime()`: Get high-resolution timestamp.
- `process.memoryUsage()`: Retrieve memory usage info.
- `process.uptime()`: Get process runtime in seconds.
- `process.kill(pid, [signal])`: Send signal to a process (default is SIGTERM).

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# MODULES AND REQUIRE

node JS

MODULES

# MODULES AND REQUIRE OVERVIEW

MASYNCTECH SCHOOL

## High-Level Explanation

- Modules: Units of code with related functionalities
- In Node.js, each file is a module
- Require Function: Node.js built-in function
- Imports and uses modules in other modules

## Basic Explanation

- Analogy: Assembling a puzzle
- Puzzle pieces represent "modules"
- "Require" function akin to picking and fitting pieces

## Important Rules

- **Modules:**
  - Small, single-functionality modules.
  - Prevent cyclic dependencies.
- **Require Function:**
  - Place `require` at file top.
  - Avoid dynamic `require` for code clarity.

## Deep Dive

- **Modules:**
  - Organize and maintain code, separate logic.
  - Promote "Separation of Concerns" principle.
  - Module-level context, prevents global variable pollution.
- **Require Function:**
  - Imports and executes JavaScript files, caches results.
  - Avoids re-evaluation for multiple requires.
  - Imports core Node.js modules, NPM modules, custom modules.

## When to use?

- **Modules:**
  - Scalable, maintainable app development.
  - Organizing database operations, API requests, etc.
- **Require Function:**
  - Accessing functionalities from other modules.
  - Using external libraries, packages.

## Summary


- Node.js modules: Organize, maintain code.
- `require` function: Import core, third-party, custom modules.
- Vital for scalable, structured applications.

MASYNCTECH
SCHOOL

# COMMONJS
# VS
# ES MODULES

node JS

MODULES

# COMMONJS VS ES MODULES

MASYNCTECH SCHOOL

## High-Level Explanation

- **CommonJS (CJS): Node.js module system**
  - Each file is a module.
  - `module.exports` exports values.
  - `require()` imports values.
- **ECMAScript Modules (ESM): ES6 standard format**
  - Uses `export`/`import` syntax.

## Basic Explanation

- Analogy: LEGO sets
- CommonJS like traditional LEGO, mixable, dynamic
- ESM like new LEGO, specific order, less mixable, complex designs

## Important Rules

- **CommonJS:**
  - Avoid cyclic dependencies.
  - Caching with `require()` reduces reloads.
- **ES Modules:**
  - Top-level `import`/`export`.
  - Use file extensions in Node.js (`import x from './module.mjs'`).

## Deep Dive

- **CommonJS (CJS):**
  - Synchronous `require()`.
  - `module.exports` and `exports` for multiple exports.
  - Dynamic `require()` placement.
- **ECMAScript Modules (ESM):**
  - Static `import`/`export`.
  - Top-level `import` hoisting.
  - File-based modules in browsers.

## When to use?

- CommonJS: Server-side, sync loading.
- ES Modules: Server-side, client-side, consistency, tree-shaking for smaller bundles in frontend.

## Summary


- CommonJS: Historical Node.js module system.
- ES Modules: Unified system for browser and server.
- Choice based on project, environment, preference.
- Transitioning to ES Modules in modern apps for advantages and ECMAScript alignment.

MASYNCTECH
SCHOOL

# EXPORTING MODULES

node JS

MODULES

# EXPORTING MODULES

MASYNCTECH
SCHOOL

## High-Level Explanation

- **Commo**nJS: `module.exports` and `exports`.
- **ES Modules:** `export` keyword, including named and default exports.

## Deep Dive

- **CommonJS:**
  - Default export with `module.exports`.
  - Named exports with `exports`.
- **ES Modules:**
  - Named exports for multiple functionalities.
  - Default export for a primary module value.

## Basic Explanation

- CommonJS: One lunchbox for everything.
- ES Modules: Separate items or a main dish with sides.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# IMPORTING MODULES

node JS

MODULES

# IMPORTING MODULES

MASYNCTECH SCHOOL

## High-Level Explanation

- **CommonJS**: `require()` for imports.
- **ES Modules:** `import` for imports.

## Basic Explanation

- CommonJS: Bookstore, ask cashier, get whole book.
- ES Modules: Magazine stand, request whole magazine (default export) or specific articles (named imports) using `import`.

## Important Rules

- **CommonJS:**
  - Avoid cyclic dependencies.
  - Caching with `require()` reduces reloads.
- **ES Modules:**
  - Top-level `import`/`export`.
  - Use file extensions in Node.js (`import x from './module.mjs'`).

## Deep Dive

- **CommonJS:**
  - Default export, access directly.
  - Multiple exports, accessed as object properties.
- **ES Modules:**
  - Named imports for specific functionalities.
  - Default exports without curly braces.
  - Star imports for everything as an object.

## When to use?

- CommonJS: Server-side, sync loading.
- ES Modules: Server-side, client-side, consistency, tree-shaking for smaller bundles in frontend.

## Summary

- CommonJS: Historical Node.js module system.
- ES Modules: Unified system for browser and server.
- Choice based on project, environment, preference.
- Transitioning to ES Modules in modern apps for advantages and ECMAScript alignment.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# MODULE CACHING

node JS

MODULES

# MODULE CACHING

MASYNCTECH
SCHOOL

## High-Level Explanation

- Node.js caches modules, preventing re-execution.
- Caching reduces redundancy, boosts performance, and ensures singleton-like behavior.

## Deep Dive

- Module loading in Node.js:
  1. Code execution.
  2. Exported content stored in cache.
  3. Content returned to caller.
- Subsequent calls fetch module from cache.
- Cache key is resolved filename; different paths = same module.

## Basic Explanation

- Analogy: Node.js module caching like a puzzle
- Keeps used modules in a side pile (cache)
- Faster and efficient retrieval
- Avoids searching every time


## NOTE

- Node.js module caching: Enhances performance
- Saves exported module content after initial load
- Prevents redundant code execution on subsequent imports
- Speeds up module loading, preserves state like singletons

MASYNCTECH
SCHOOL

# FS

## FILE SYSTEM

node JS

NODE CORE APIS

# FILE SYSTEM (FS) API

MASYNCTECH SCHOOL

## High-Level Explanation

- Node.js FS API: Core module for file system interaction
- Offers sync and async methods for file/directory operations

## Basic Explanation

- Analogy: FS API as a paper drawer helper
- Fetch a paper (read a file)
- Add a new paper (create a file)
- Modify paper content (update a file)
- Dispose of a paper (delete a file)
- All based on your instructions

## Important Rules

- Check file/dir existence before operations.
- Handle errors, especially in async methods.
- Properly close file streams to prevent issues.
- Opt for streams for efficient frequent I/O ops.

## Deep Dive

- **Node.js FS API features:**
  - Create, read, update, delete files
  - Manage directories, streams, file permissions
  - Comprehensive for OS-level file operations

## When to use?

- Use Node.js FS API for file operations in your app.
- Examples: Read configs, log errors, process uploads, and more.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# OS

## OPERATING SYSTEM

node JS

NODE CORE APIS

# OS (OPERATING SYSTEM) API

MASYNCTECH SCHOOL

## High-Level Explanation

- Node.js `os` module: Interact with the OS.
- Access OS info: type, platform, uptime, and more.

## Deep Dive

- `os` module: Bridge between Node.js and OS.
- Gather system info for diagnostics, logging.
- Useful for runtime decisions, environment adaptation.

## Basic Explanation

- Analogy: `os` module as artist's canvas info
- Provides system type, size, available resources
- Helps adapt code to the system environment

## When to use?

- Adapt app behavior for different OSes.
- Log system details for debugging or audit.
- Dynamically handle path differences between OSes.

## Important Rules

- Use `os` module for info, not critical decisions.
- Handle OS differences in cross-platform apps.


## NOTE

- `os` module for info, not OS manipulation.
- Be aware of platform-specific method variations.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# STREAM API

node JS

NODE CORE APIS

# STREAM API

MASYNCTECH SCHOOL

## High-Level Explanation

- Node.js streams: Data processing channels like pipes.
- Read from source, write to destination continuously.

## Deep Dive

- Node.js Stream API: Vital for efficient data handling.
- Four main types: Readable, Writable, Duplex, Transform.
- Used in reading/writing files, network, HTTP, and more.

## Basic Explanation

- Analogy: Streams like filling a pool with buckets.
- Start processing data as it arrives, not waiting for all.
- Efficiency in data handling and processing.

## When to use?

- **Use streams for:**
  - Large, memory-intensive datasets
  - On-the-fly processing (compress, transform, encrypt)
  - I/O operations (files, network, multimedia)

## Important Rules

- Stream event handling ('error', 'data', 'end', 'finish')
- `pipe` for chaining streams (read to write)
- Manage backpressure if read stream is faster than write

## NOTE


- Streams are event-driven, rely on emitting/listening to events.
- Buffering crucial for rate mismatches (fast source, slow destination).

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# BUFFER API

node JS

NODE CORE APIS

# BUFFER API

## High-Level Explanation

- Node.js Buffer module: Manage raw memory outside V8 heap
- Purpose: Handle binary data efficiently in Node.js

## Deep Dive

- JavaScript struggles with binary data.
- Node.js I/O operations often require binary data handling.
- Buffer module permits byte-level data interaction.

## Basic Explanation

- Analogy: Buffer as a special tool for unique, old-language book
- Helps read and understand binary data, like decoding the book.

## When to use?

**Buffer module useful when:**

- Managing TCP streams or file system data.
- Interacting with processes/systems using different data formats.
- Processing binary data from sources like images, audio files, etc.

## Important Rules

- Avoid `new Buffer()`, use `Buffer.from()` or `Buffer.alloc()`.
- Be cautious of unintended modifications via memory manipulation.
- Always specify encoding when converting Buffers to strings.

## NOTE

- Buffers are mutable; content can change.
- Buffer size fixed upon creation; not resizable.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# PATH API

node JS

NODE CORE APIS

# THE PATH API

## High-Level Explanation

- Node.js Path module: Manage file/directory paths.
- Handles paths efficiently, OS-independent.

## Basic Explanation

- Analogy: File paths as city addresses.
- Path module provides consistency like a universal address system.
- Understand and locate paths across different "cities" (OS).

## Important Rules

- Use Path methods for cross-platform compatibility.
- Detect path type with `path.isAbsolute()`.

## Deep Dive

- Path module abstracts OS differences in file paths.
- Ensures code consistency across platforms.
- Functions for path parsing, joining, extracting names, etc.

## When to use?

- **Use Path module:**
  - Manipulating file/directory paths
  - Cross-platform apps for path consistency
  - Extract path components (dir, base, extension).

## NOTE

- Path module manipulates path strings, not filesystem access.
- For platform-specific paths, use `path.win32` or `path.posix`.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# EVENT API

node JS

NODE CORE APIS

# EVENT API

## High-Level Explanation

- Node.js Event module: `EventEmitter` class.
- Essential for event handling in Node.js.
- Simplifies creation and management of custom events.

## Basic Explanation

- Analogy: `EventEmitter` as a radio broadcaster.
- App parts tune to channels (events) and react to broadcasts.

## Important Rules

- Limit event listeners to prevent memory leaks.
- Handle `error` event in `EventEmitter` instances.
- Avoid blocking operations in event listeners.

## Deep Dive

- Node.js core modules use `EventEmitter`.
- Event-driven programming central to Node.js.
- Publisher-subscriber model for reacting to events.

## When to use?

- Use `EventEmitter` for indirect component communication.
- Create custom event-driven architectures.

## NOTE


- Default max: 10 listeners per event, adjustable.
- `EventEmitter` is synchronous; listeners called one after the other.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

# BUILDING RESTFUL API

MASYNCTECH SCHOOL

node JS

BUILDING RESTFUL API

MASYNCTECH
SCHOOL

# OVERVIEW OF RESTFUL API

node JS

BUILDING RESTFUL API

# BRIEF OVERVIEW OF RESTFUL APIS

MASYNCTECH SCHOOL

## High-Level Explanation

- RESTful APIs: Follow HTTP conventions for web services.
- Use HTTP for CRUD operations (Create, Read, Update, Delete) on data.

## Basic Explanation

- Analogy: Library for RESTful API.
- Request librarian (server) for books (data).
- Librarian actions: Read (give), Create (take), Update (replace), Delete (discard).
- Standardized requests (HTTP methods) for predictability.

## Deep Dive

- REST by Roy Fielding (2000): Web as resource-centric.
- Key principles:
  - **Statelessness**: Requests self-contained; server sessionless.
  - **Client-Server:** UI (client) separate from data processing (server).
  - **Cacheability:** Responses cacheable for performance.
  - **Layered System:** Intermediaries for scalability.
  - **Uniform Interface:** Consistent, simplified interactions.

# BRIEF OVERVIEW OF RESTFUL APIS

# BRIEF OVERVIEW OF RESTFUL APIS

MASYNCTECH SCHOOL

## High-Level Explanation

- RESTful APIs: Follow HTTP conventions for web services.
- Use HTTP for CRUD operations (Create, Read, Update, Delete) on data.

## Deep Dive

- REST by Roy Fielding (2000): Web as resource-centric.
- Key principles:
  - **Statelessness**: Requests self-contained; server sessionless.
  - **Client-Server:** UI (client) separate from data processing (server).
  - **Cacheability:** Responses cacheable for performance.
  - **Layered System:** Intermediaries for scalability.
  - **Uniform Interface:** Consistent, simplified interactions.

## Basic Explanation

- Analogy: Library for RESTful API.
- Request librarian (server) for books (data).
- Librarian actions: Read (give), Create (take), Update (replace), Delete (discard).
- Standardized requests (HTTP methods) for predictability.

## When to use?

- Use REST:
  - Stateless client-server interaction.
  - Scalable, cacheable, and consistent web services.
  - Simple, integrable, cross-platform API.

MASYNCTECH
SCHOOL

# WHAT IS AN API?

node JS

BUILDING RESTFUL API

# BRIEF OVERVIEW OF RESTFUL APIS

# WHAT IS AN API?

MASYNCTECH SCHOOL

## High-Level Explanation

- API: Rules and protocols for app communication.
- Specifies methods, data formats for info exchange.

## Deep Dive

- APIs central in software development.
- Integrate systems, modularize, reuse components.
- Used in web apps, mobile-cloud data, hardware, OS services.

## Basic Explanation

- API like a restaurant menu.
- Lists dishes (operations) with descriptions.
- Specify items, kitchen (system) serves without knowing how it works.

## When to use?

- **Use APIs when:**
  - Components need to communicate or share data.
  - Third-party interaction without exposing internals.
  - Building modular, scalable systems with replaceable components.

MASYNCTECH
SCHOOL

# CLIENT-SERVER COMMUNICATION

# LIFE CYCLE

node JS

BUILDING RESTFUL API

# CLIENT-SERVER COMMUNICATION LIFE CYCLE

MASYNCTECH SCHOOL

## High-Level Explanation

- Client-server communication: Steps for client-server interaction.
- Client (user's device) communicates with server (centralized system).

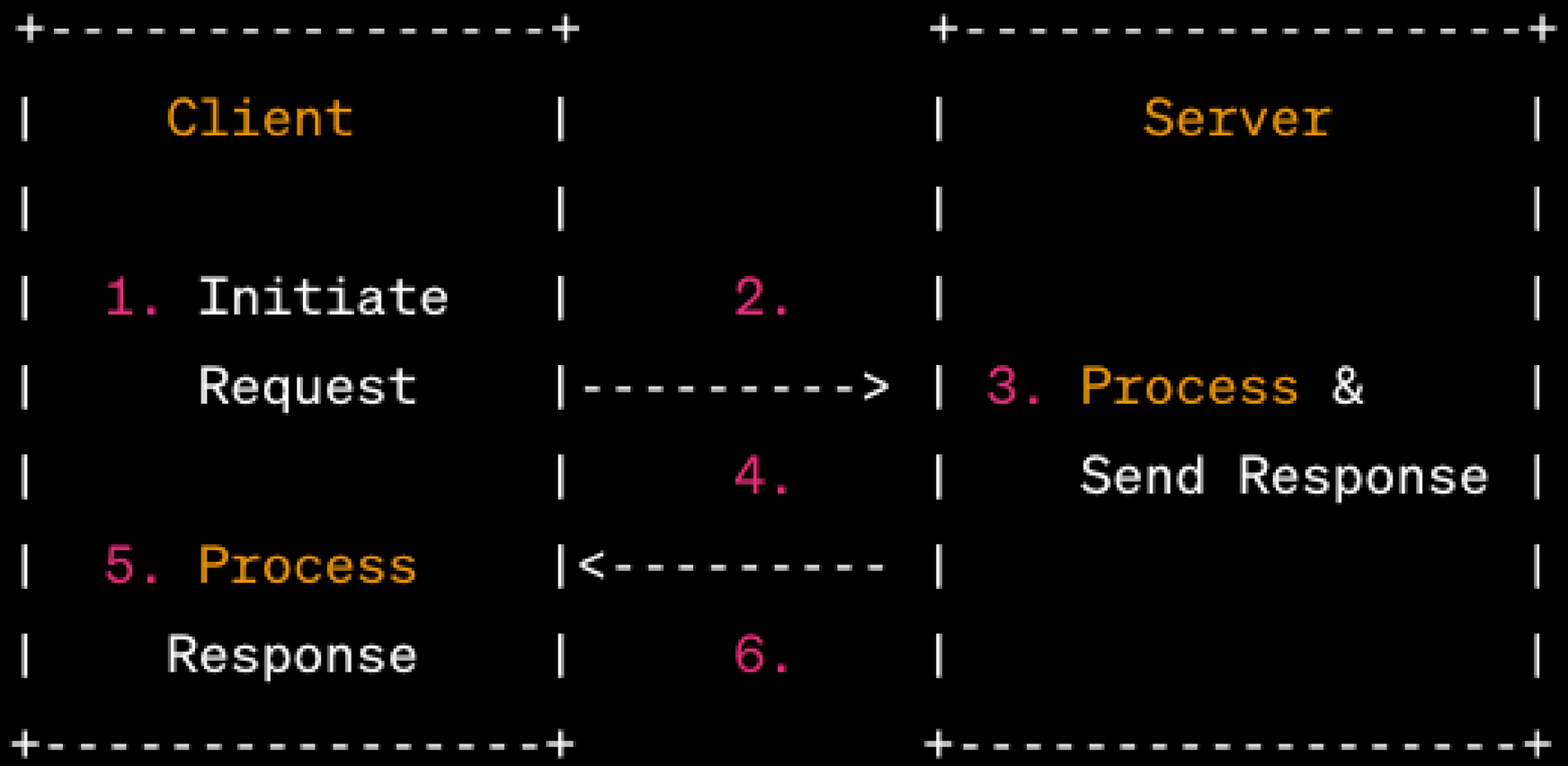
## Deep Dive

- **Client-server communication life cycle:**
  1. **Request Initiation**: Client initiates a request.
  2. **Connection Establishmen**t: Establish connection (TCP/IP).
  3. **Request Sent**: Client sends request (e.g., HTTP).
  4. **Processing on Server**: Server processes request.
  5. **Response Sent**: Server sends response.
  6. **Client Processes Response**: Client handles data (e.g., web rendering).
  7. **Connection Termination**: Terminate or keep alive (HTTP/1.1).

## Basic Explanation

- Analogy: Restaurant visit for client-server flow.
- You (client) place order (request) with waiter (protocol).
- Waiter goes to kitchen (server) for food prep.
- Waiter serves food (response) to your table.
- You leave (connection termination).

# CLIENT-SERVER COMMUNICATION LIFE CYCLE



# CLIENT-SERVER COMMUNICATION LIFE CYCLE

Client

Request

Response

Server

MASYNCTECH
SCHOOL

# UNDERSTANDING

# REST AND

# HTTP METHODS

node JS

BUILDING RESTFUL API

# UNDERSTANDING REST AND HTTP METHODS

MASYNCTECH SCHOOL

## High-Level Explanation

- REST: Architectural style for networked apps.
- Uses HTTP methods, primarily for web services.
- CRUD operations via HTTP requests.

## Basic Explanation

- Analogy: REST as a library visit.
- You (client) request books (GET), return (DELETE), add (POST), or update (PUT).
- Librarian (server) handles requests, no need to remember you (Statelessness).
- Get suggestions based on recent and popular books (Cacheable).

## Deep Dive

- **REST by Dr. Roy Fielding (2000): Use HTTP simplicity.**
- REST principles:
  - Statelessness: Request self-contained, no server session.
  - Client-Server: Client UI, server data and ops.
  - Cacheable: Responses cacheable.
  - Layered System: Components limited to their layer.
  - Code on Demand (Optional): Servers extend client.
  - Uniform Interface: Simplifies, modularizes, and scales.

- **REST HTTP Methods:**
  - **GET**: Retrieve resource(s).
  - **POST**: Create new resource.
  - **PUT**: Update existing resource.
  - **DELETE**: Remove resource.
  - **PATCH**: Partial update to resource.
  - **HEAD**: Fetch headers.
  - **OPTIONS**: Retrieve supported methods.

# UNDERSTANDING REST AND HTTP METHODS

MASYNCTECH
SCHOOL

# SETTING UP BASIC HTTP SERVER

node JS

BUILDING RESTFUL API

MASYNCTECH
SCHOOL

# NODEJS SERVER

# ROUTING

BUILDING RESTFUL API

# ROUTING IN NODE.JS

MASYNCTECH
SCHOOL

## High-Level Explanation

- Routing: Handling client requests based on URI and HTTP method. - In raw Node.js (no frameworks), route manually by checking URL and method. - Simple for basic apps, but cumbersome for complex ones.

## Drawbacks

- Node.js for simple routing: Manually check request URL, method.
- Works for basic apps or learning.
- Cumbersome, inefficient for complex apps.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# ROUTE PARAMS

node JS

BUILDING RESTFUL API

# ROUTE PARAMS

MASYNCTECH
SCHOOL

## Syntax

```
`https://example.com/products/:productId`:
```

- `https://`: URL protocol, secure HTTP.
- `example.com`: Domain or hostname, server location.
- `/products/`: Path or endpoint, resource/page.
- `:productId`: Route parameter, captures URL segment.

# ROUTE PARAMS

MASYNCTECH SCHOOL

## High-Level Explanation

- Route parameters: Named URL segments, capture values.
- Handle URL variations without multiple routes.

## Deep Dive

- Node.js native HTTP module lacks built-in route parameter support.
- Common in web frameworks, starts with colon (:), captures values.
- Enables handling dynamic content and actions from URL.

## Basic Explanation

- Analogy: Route parameters like a versatile map.
- Instead of separate maps for places, use one with "PLACE."
- Friends replace "PLACE" with the location they want.
- Route parameters serve various locations/data by changing the value, like changing a destination on a map.

## When to use?

- Use route parameters:
  - Capture dynamic URL values (e.g., user IDs, product names).
  - Create flexible, scalable routing (no multiple routes).
  - Retrieve data based on user actions or requests.

## SUMMARY

- Route parameters: Dynamic URL segments, handle data with one route.
- Vital for scalable routing.
- Wise use, validation, and clarity are key.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# PARAMETERS

# QUERY

node JS

BUILDING RESTFUL API

# EXTRACTING QUERY PARAMETERS

MASYNCTECH SCHOOL

## High-Level Explanation

- Query Parameters: Key-value pairs in URL after "?", used in HTTP requests.
- Role: Enable dynamic URL variations without route change.
- Common Uses: Filtering, searching, pagination, and more.
- Distinct: After "?" and use "&" to combine, unlike path parameters in URL path.

## Basic Explanation

- Analogy: Ordering a customized burger at a fast-food counter.
- Website requests can ask for specific details, like burger toppings.
- Query parameters are like customizations in the web address after `?`.
- Each parameter (e.g., `pickles=extra`) is a specific request.
- Extracting query parameters in Node.js is like understanding and fulfilling a custom order.

```
`www.burgerplace.com/menu?pickles=extra&onions=none&add=bacon`.
```

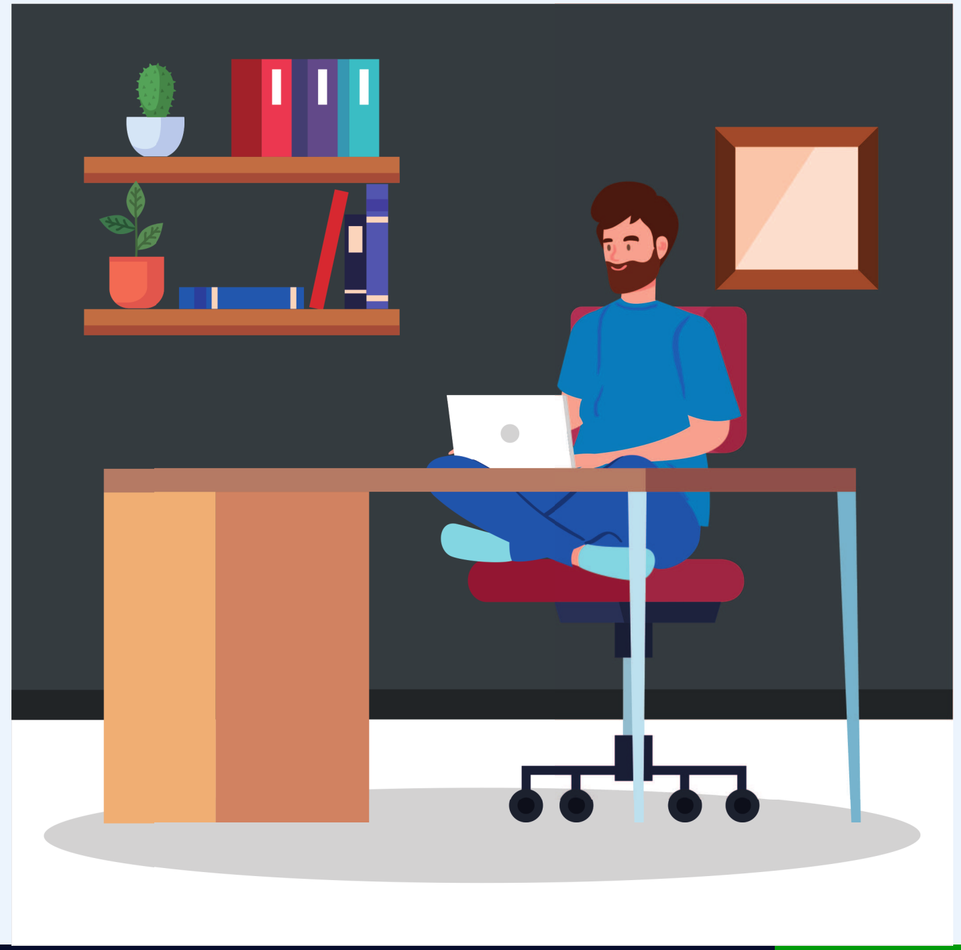
MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# SENDING JSON RESPONSE



node JS

BUILDING RESTFUL API

# SENDING JSON RESPONSES

MASYNCTECH
SCHOOL

## High-Level Explanation

- To send JSON responses in Node.js using `http`:
  1. Convert object/array to JSON string with `JSON.stringify()`.
  2. Set response header Content-Type to `application/json`.
  3. Send JSON string as response.

MASYNCTECH
SCHOOL

# CODE DEMO

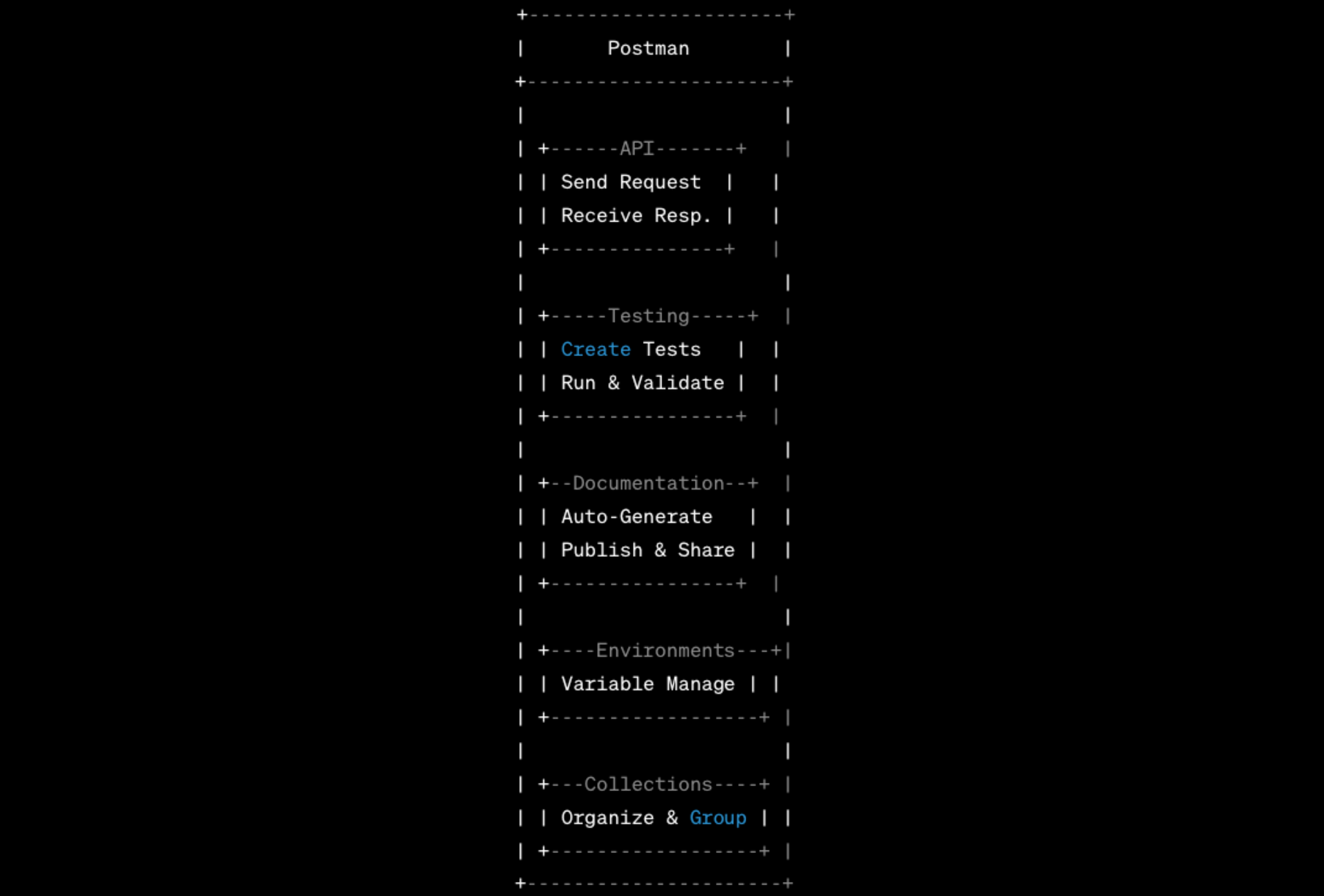
node JS

MASYNCTECH
SCHOOL

# INTRODUCING POSTMAN CLEINT

BUILDING RESTFUL API

# INTRODUCING POSTMAN



# INTRODUCING POSTMAN

MASYNCTECH SCHOOL

## High-Level Explanation

- Postman: Developer tool for API tasks.
- Build, test, document APIs.
- Send requests, view responses for development and testing.

## Basic Explanation

- Postman: Your digital mail carrier's toolbelt.
- Organizes, sends, checks digital messages (requests) to APIs.
- Ensures efficient deliveries and responses.

## Important Rules

- Organize requests in collections.
- Use env variables for dynamic values.
- Write tests, set up monitors for critical endpoints.
- Share collections, env settings for team collaboration.

## Deep Dive

- Started as Chrome extension in 2012.
- Now standalone app for macOS, Windows, Linux.
- Builds API requests, configures headers, params, payloads.
- Supports automated testing, mock servers, monitoring, docs.

## When to use?

- Use Postman for:
  - Testing new API endpoints.
  - Debugging or troubleshooting existing APIs.
  - Documenting APIs.
  - Creating mock servers.
  - Automated endpoint testing.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# HANDLING POST REQUESTS

node JS

BUILDING RESTFUL API

# HANDLING POST REQUESTS

## High-Level Explanation

- Raw Node.js for POST: Manual setup needed.
- Node.js lacks high-level request body parsing.
- Frameworks simplify POST handling.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS

MASYNCTECH
SCHOOL

# PROJECT EMPLOYEES API

node JS

BUILDING RESTFUL API

# HANDLING POST REQUESTS

## High-Level Explanation

- Raw Node.js for POST: Manual setup needed.
- Node.js lacks high-level request body parsing.
- Frameworks simplify POST handling.

MASYNCTECH
SCHOOL

# CODE DEMO

node JS