



Sri Lanka Institute of Information Technology

APPLICATION FRAMEWORKS

NODEJS

LECTURE 04

Faculty of Computing

Department of Computer Science and Software Engineering

Module Code: SE3040

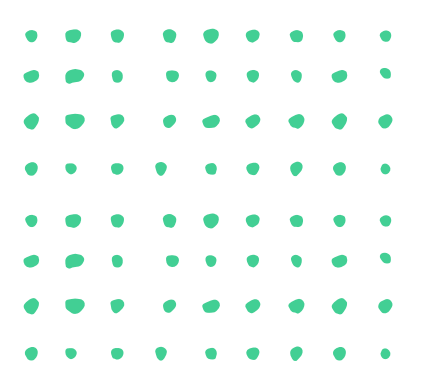
Agenda




- 1 NodeJS
- 2 Event loop
- 3 Use cases
- 4 Advantages and disadvantages
- 5 Package manager
- 6 Require

NODE JS

- Node Js is a runtime environment that allows JavaScript to be executed on the server side.
- Created by Ryan Dahl to build fast and interactive web applications that need live updates (like chat apps using WebSockets)
- NodeJS is an open source, cross platform runtime environment for server-side and networking applications.
can run on multiple operating systems (like Windows, macOS, Linux) without changes to its code.
- Uses event-driven, non-blocking I/O model which makes NodeJS lightweight and efficient.
Code continues running while slow tasks happen in the background.
Example: A chef starts chopping veggies while the oven heats up.
the program doesn't halt while waiting for input/output operations (like reading from a file or network) but instead registers a callback to be executed later, allowing it to continue processing other tasks asynchronously
- Ideal for data-intensive real-time applications that run across distributed devices.
applications that handle and process massive amounts of data quickly



Event-Driven Model

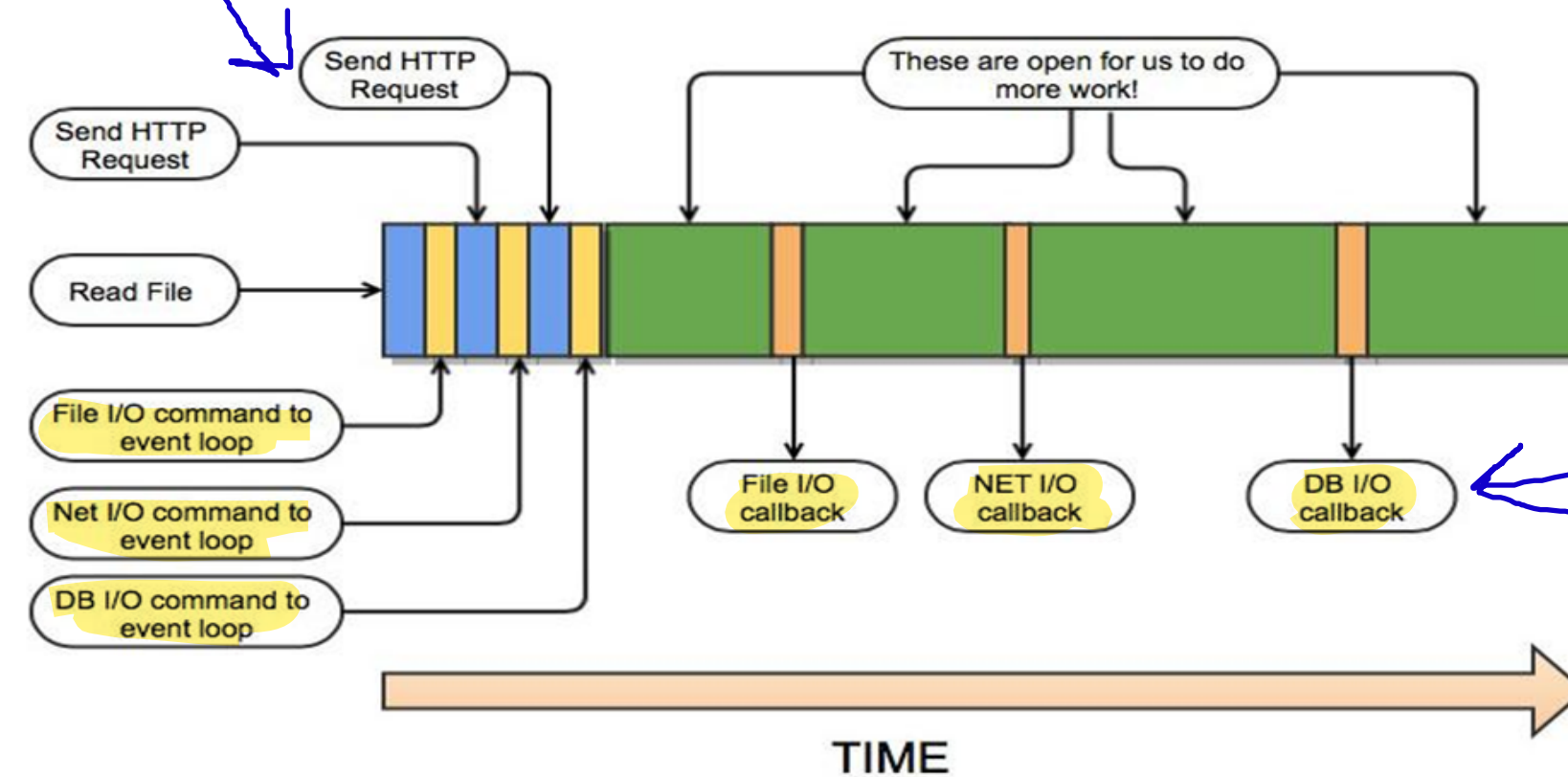
- Flow of execution is determined by events, instead of following a sequential execution. the program's actions (code execution) are triggered by external triggers, such as user actions, system events, or messages, instead of simply running through code line by line.
- How works?
 - The system listens for events and, when an event occurs, executes a callback function (event handler) without blocking execution, making event-driven systems more responsive and efficient
- Key components of event driven model:
 - **Event emitter** It is responsible for signaling that something has happened (e.g., a button click, data received, or a timer expiration). Role: It triggers events and notifies all registered listeners (event handlers) about the occurrence of the event.
 - **Event listener** / Event handler 
 - **Call back function** function that is passed as an argument to another function and is executed after a specific event occurs. Role: It defines the action to be taken when the event is triggered. The callback function is invoked by the event listener when the event is emitted.



EVENT LOOP

- 1) Client Request: A client (e.g., a web browser) sends a request to the Node.js server (e.g., an HTTP request).
- 2) Event Loop: Node.js uses a single-threaded event loop to handle incoming requests. The event loop is the core of Node.js's non-blocking I/O model. It continuously checks for new events (e.g., incoming requests, completed I/O operations) and processes them.
- 3) When a request involves I/O operations (e.g., reading a file, querying a database), Node.js delegates these tasks to the system kernel or external libraries. Instead of waiting for the I/O operation to complete, Node.js continues processing other requests.
- 4) Callback Function: Once the I/O operation is complete, a callback function is triggered to handle the result. The callback function is executed asynchronously, allowing the server to remain responsive.
- 5) Response to Client: After processing the request, Node.js sends a response back to the client.

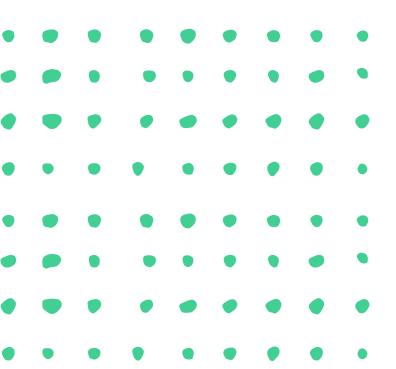
Node.js (non-blocking) Event Loop



The **event loop** is a core mechanism in Node.js that **allows it to handle multiple operations without blocking execution**, making it **fast and efficient** for **real-time applications**.

USE CASES

situation in 'node js' be used.



- **Not the best** platform for CPU intensive heavy computational applications.
- Ideal for building fast and scalable network applications.
- NodeJS is capable of handling a huge number of simultaneous connections with high throughput.
- For each connection NodeJS does not spawn new Thread causing max out of memory instead handle all in single thread using non-blocking I/O model.
- NodeJS has achieved over 1 Million concurrent connections.
- Bubbling errors up to NodeJS core event loop will cause crashing the entire program.

Node.js is a high-performance, event-driven JavaScript runtime built on **Chrome's V8 engine**, designed to handle massive concurrency (thousands of simultaneous connections) with high throughput (fast data processing).

Node.js doesn't spawn a new thread for each connection

One main thread handles all connections
Uses non-blocking I/O: delegates operations (DB calls, file reads) to the OS

When an error bubbles up to the Node.js event loop without being caught, it crashes the entire process.

```
setTimeout(() => {  
  throw new Error('Uncaught exception!'); // CRASHES Node.js  
, 1000);
```

USE CASES - Activity

Discuss and write down a real-world success story of companies or projects that have effectively utilized Node.js?

Netflix

Challenges Netflix Faced -

Scalability Issues: Netflix's legacy Java-based backend struggled to handle the increasing number of users and the growing complexity of its services.

Slow Startup Time: The Java-based system had a slow startup time, which delayed the deployment of new features and updates.

Complexity in Development: The monolithic architecture made it difficult for developers to work on different parts of the application independently.

Why Netflix Chose Node.js -

Lightweight and Fast: Node.js is built on Google's V8 JavaScript engine, which is known for its speed and efficiency.

Non-Blocking I/O: Node.js's event-driven, non-blocking I/O model made it ideal for handling a large number of concurrent connections, which is crucial for a streaming platform like Netflix.

Modularity: Node.js allowed Netflix to break down its monolithic architecture into smaller, more manageable microservices.

Developer Productivity: JavaScript is widely used, and Node.js enabled Netflix's frontend and backend developers to use the same language, improving collaboration and reducing development time.

How Netflix Utilized Node.js

Microservices Architecture: Netflix transitioned from a monolithic architecture to a microservices-based architecture using Node.js. Each microservice was responsible for a specific function (e.g., user authentication, recommendation engine, video streaming), making the system more modular and scalable.

API Layer: Node.js was used to build the API layer that connects the frontend (user interface) with the backend (microservices). This improved the performance of the application by reducing latency and enabling faster data exchange.

Real-Time Updates: Node.js's event-driven architecture allowed Netflix to deliver real-time updates to users, such as personalized recommendations and notifications.

Improved Startup Time: Node.js reduced the startup time of the application from 40 minutes to less than 1 minute, enabling faster deployment of new features and updates.

Results and Benefits

Improved Performance: Node.js helped Netflix reduce the startup time and improve the overall performance of the application. The platform could now handle millions of concurrent users without any performance degradation.

Scalability: The microservices architecture enabled Netflix to scale individual components independently, ensuring that the platform could handle growing user demand.

Faster Development: By using JavaScript for both frontend and backend development, Netflix improved developer productivity and reduced the time required to build and deploy new features.

Enhanced User Experience: Real-time updates and personalized recommendations improved the overall user experience, leading to higher customer satisfaction and retention.

USE CASES

- I/O bound applications.
- Data streaming applications.
- Data intensive real-time applications.
- JSON APIs based applications.
- Single page applications



UBER



NETFLIX



PayPal



eBay

ADVANTAGES

JavaScript can be used for both frontend and backend development.

- Ability to use single programming language from one end of the application to the other end.

horizontally (adding more machines) and vertically (adding more resources to a single machine)

- NodeJS applications are easy to scale both horizontally and vertically.
- Delivers improved performance since V8 engine compile the JS code into machine code directly.
- Performance increased via caching modules into memory after the first use.
- Easily extensible.
- Support for common tools like unit testing.
- Well build 'npm' package manager and it's large number of reusable modules.

DISADVANTAGES

- Even though there are number of libraries available, the actual number of robust libraries is comparatively low.
- Not suitable for computationally intensive tasks. CPU-heavy tasks (e.g., image processing, complex math, machine learning).
- Asynchronous programming model is complex than synchronous model.

Exercise:

1. What are robust libraries?
2. What are the available robust libraries?
3. Why is the Number of Robust Libraries Low?

What are Robust Libraries?

Robust libraries are software libraries that are designed to be reliable, efficient, and maintainable, making them suitable for use in production environments. These libraries are built with a focus on stability, performance, and scalability, ensuring that they can handle real-world use cases without failing or causing issues.

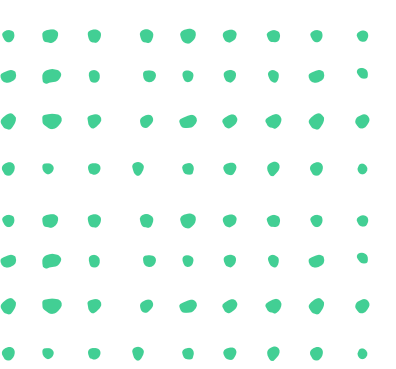
What are the Available Robust Libraries?

Express.js, Lodash, Axios, Mongoose, Sequelize, Jest, Mocha, Socket.IO

Why is the Number of Robust Libraries Low?

Rapid Growth of the Ecosystem, Lack of Maintenance, Insufficient Testing

NODE PACKAGE MANAGER



- Reusable NodeJS components easily available through online repository.
- Build in **dependency management**, **version** and **scripting mechanism**.
- Global installations will be available throughout the system while local installations will only be available for that particular application.
- By default all the dependencies will get installed to 'node_modules' directory.
- 'package.json' contains all information related to the NodeJS application. The file be placed in the root directory of the application.
- 'package.json' will contain name, version, author, repository, required node version, scripts and dependencies etc.

all metadata about the application.



NODE PACKAGE MANAGER... (CNT)

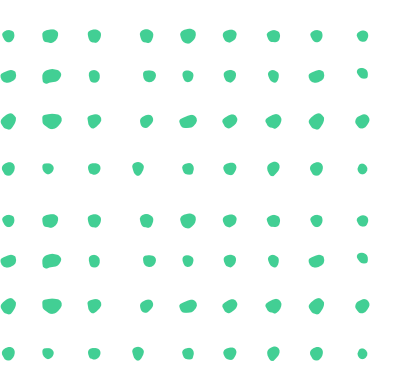
- To denote the compatible version numbers

Operator	Example	Meaning
<code><=</code>	<code>"express": "<=4.18.2"</code>	Installs version 4.18.2 or lower
<code>>=</code>	<code>"express": ">=4.18.2"</code>	Installs version 4.18.2 or higher
<code>~</code>	<code>"express": "~4.18.2"</code>	Allows only patch updates (4.18.x but NOT 4.19.0)
<code>^</code>	<code>"express": "^4.18.2"</code>	Allows minor & patch updates (4.x.x but NOT 5.0.0)
<code>*</code>	<code>"express": "*"</code>	Installs any available version (not recommended)
<code>1.2.x</code>	<code>"express": "1.2.x"</code>	Installs any patch version (1.2.0, 1.2.5, but NOT 1.3.0)
<code>latest</code>	<code>"express": "latest"</code>	Installs the latest available version from npm

Only the last digit (patch version)
Minor and patch versions can increase.

- There are two types of dependencies,
 - Dependencies**: needed in runtime (e.g., `express`, `mongoose`)
 - devDependencies**: only for development (e.g., `jest`, `nodemon`)

NODE REQUIRE



- `require()` is a built-in function in Node.js used to import modules and files.
- Require modules get loaded synchronously and will be cached after the first use.
- A **core module** in Node.js is a built-in module that comes with Node.js out of the box

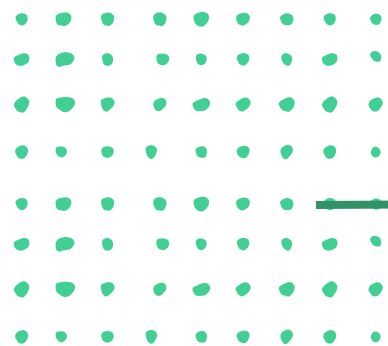
- **Core modules** i.e → `const http = require('http');`

Core modules are built-in modules that come pre-installed with Node.js. These modules provide essential functionality and **do not require any additional installation**. They are part of the Node.js runtime and can be imported using the **require function**.

- Module start with `./`, `../`, or `/` treat as local file/module

- **Local module** i.e → `const add = require('./math');`

Local modules are **user-defined modules** created by developers for their specific applications. These modules are stored in separate files and can be imported into other files using the **require function**. Local modules are identified by paths that start with `./`, `../`, or `/`.





THAT'S ALL FOLKS !

ANY QUESTIONS ?