# nodejs



A study material for the students of GLS University

Prepared by : Ms. Harshita Maheshwari

# Introduction

**Node.js is a cross-platform runtime environment and library for running JavaScript applications outside the browser.**
**It is used  for creating server-side and networking web applications.**

Written in C++

created by Ryan Dahl starting in 2009

Built on top of Chrome's V8 engine – so pure JavaScript

Framework to build asynchronous I/O applications

Single Threaded – no concurrency bugs –no deadlock

One node process = one CPU core

It is open source and free to use.

Community and Ecosystem

# JavaScript vs. Node.js

| JAVASCRIPT | NODEJS |
|---|---|
| Javascript is a programming language that is used for writing scripts on the website. | NodeJS is a Javascript runtime environment. |
| Javascript can only be run in the browsers. | NodeJS code can be run outside the browser. |
| It is basically used on the client-side. | It is mostly used on the server-side. |
| Javascript is capable enough to add HTML and play with the DOM. | Nodejs does not have capability to add HTML tags. |
| Javascript can run in any browser engine as like JS core in safari and Spidermonkey in Firefox. | Nodejs can only run in V8 engine of google chrome. |
| Javascript is used in frontend development. | Nodejs is used in server-side development. |
| Some of the javascript frameworks are RamdaJS, TypedJS, etc. | Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm. |
| It is the upgraded version of ECMA script that uses Chrome's V8 engine written in C++. | Nodejs is written in C, C++ and Javascript. |

# Why nodejs

**Javascript everywhere**

**High performance**

**Non blocking , Asynchronous**

**Rich Eco system**

**Scalability**

**Cross platform**

**Great for Real-Time Applications**

**IoT and Embedded Systems**

# Success Stories

LinkedIn

Rails to Node
« Servers were cut to 3 from 30 »
« Running up to 20x faster in some scenarios »
« Frontend and backend mobile teams could be combined [...] »

PayPal

Java to Node
« Built almost twice as fast with fewer people »
« Double the requests per second »
« 35% decrease in the average response time »

GROUPON

Walmart

YAHOO!

UBER

# Architecture of Node.js

Node.js follows a single-threaded event loop architecture.

**Components:**

**V8 Engine** – Converts JS code into machine code.

**Event Loop** – Handles all asynchronous operations.

**Libuv Library** – Provides async I/O, thread pool, etc.

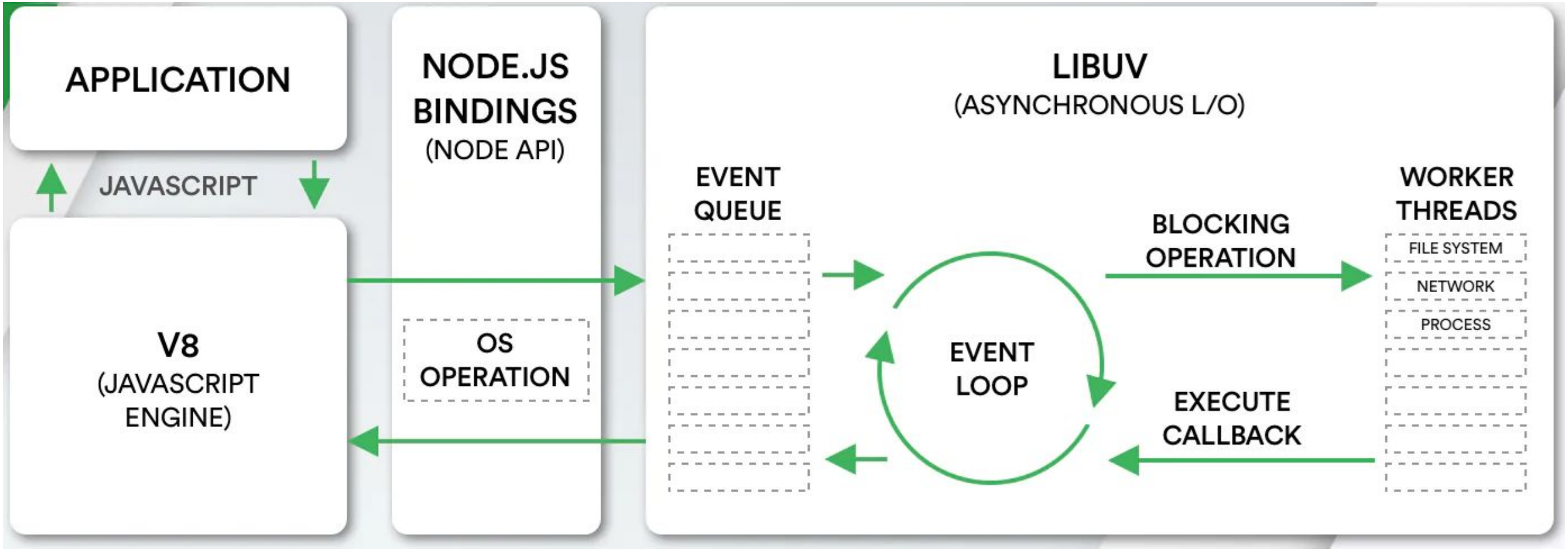
**Event Queue** – Holds incoming requests.

**Thread Pool** – Used internally for handling blocking operations.

**How it Works?**

1. A client sends a request (e.g., read a file, query DB).
2. The Event Loop receives it.
3. If non-blocking → handled immediately.
4. If blocking → sent to the Thread Pool.
5. Once done → result returned to the Event Loop.
6. The response is sent back to the client.

# How it works


APPLICATION
JAVASCRIPT
V8
(JAVASCRIPT
ENGINE)
NODE.JS
BINDINGS
(NODE API)
OS
OPERATION
LIBUV
(ASYNCHRONOUS L/O)
EVENT
QUEUE
EVENT
LOOP
BLOCKING
OPERATION
EXECUTE
CALLBACK
WORKER
THREADS
FILE SYSTEM
NETWORK
PROCESS

# Nodejs setup

## Windows Setup -

1. Download: Go to nodejs.org→ Download LTS version (.msi).
2. Install: Run installer → Accept defaults → Finish.
3. Verify: node -v
   npm -v

## Ubuntu Setup -

sudo apt update
sudo apt install -y nodejs npm
Verify: node -v
npm -v

# Running Node.js

You can run Node.js in two ways:

**REPL (Console)** – type node in terminal and run JavaScript directly.

**Script file** – create a file (e.g., app.js) and run: node app.js

# Nodejs console - REPL

REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode. Node.js or Node comes bundled with a REPL environment. It performs the following tasks −

Read − Reads user's input, parses the input into JavaScript data structure, and stores in memory.

Eval − Takes and evaluates the data structure.

Print − Prints the result.

Loop − Loops the above command until the user presses ctrl-c twice.

The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.

# REPL Commands

| Command | Description |
| --- | --- |
| `.help` | List all commands |
| `.exit` | Exit the console |
| `.save filename` | Save current session |
| `.load filename` | Load JS file |

# Command Utilities

| Command | Description |
| --- | --- |
| `node -v` | Check Node.js version |
| `npm -v` | Check npm version |
| `npm init` | Create a new Node project |
| `npm install <package>` | Install a module |
| `npm list` | List installed packages |
| `node <filename>` | Run a JavaScript file |

# modules

In simple terms, a module is a piece of reusable JavaScript code. It could be a .js file or a directory containing .js files. You can export the content of these files and use them in other files.

Modules help developers adhere to the DRY (Don't Repeat Yourself) principle in programming. They also help to break down complex logic into small, simple, and manageable chunks.

# Types of modules

**Core module/Built-in modules** : Node.js comes with some modules out of the box. These modules are available for use when you install Node.js.

e.g. http, path, url, os, fs, net, etc.

**Third-party module:** Modules that you install from node package manager. We use these modules to accomplish or simplify any existing task. For example, to simplify our web API development we use express, or to deal with date and time we use moment.

**Local module:** These are the modules that we create for our own use.

These modules basically consist of core business logic of our code.

# Create a local module

Creating a module in Node is very simple, just create a javascript file .
Let's say we defined one file, named sum.js, with the following content:

**greet.js**

```
export function sayHello(name) {
  return `Hello, ${name}!`;
```

**app.js**

```
import { sayHello } from './greet.js';
console.log(sayHello("Students"));
```

# HTTP module

The http module allows Node.js to create web servers and handle HTTP requests and responses — no external library needed.

It’s a core module (built-in).

1. **Import the module -**

```
import http from 'http';
```

# HTTP module (Create a Basic Web Server)

```
import http from 'http';

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, World! Welcome to Node.js Server.');
});


server.listen(3000, () => {
  console.log('Server running at http://localhost:3000/');
});
```

# HTTP module (with routing)

```
import http from 'http';
const server = http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/html'});
  if (req.url === '/')
    res.end('<h1>Home Page</h1>');
  else if (req.url === '/about')
    res.end('<h1>About Us</h1>');
  else {
    res.writeHead(404);
    res.end('<h1>404 Not Found</h1>');
  }
});
server.listen(4000, () => console.log('Server running on port 4000'));
```

# URL module

The URL module splits up a web address into readable parts.

```
const url = require('url’);
let adr = 'http://localhost:1000/default.htm?id=2&name=ram’;
let q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:1000’
console.log(q.pathname); //returns '/default.htm’
console.log(q.search); //returns '?id=2&name=ram'

var qdata = q.query; //returns an object: { id: 2, name: ‘ram' }
console.log(qdata.name); //returns ‘ram'
```

# fs module

The fs module allows Node.js to interact with the file system — reading, writing, deleting, and managing files and directories.

It's a core (built-in) module — no installation required.

## Common File Operations -

| Operation | Method | Example |
|---|---|---|
| **Write File** | `fs.writeFile()` | Create or overwrite a file |
| **Read File** | `fs.readFile()` | Read file contents |
| **Append File** | `fs.appendFile()` | Add content to an existing file |
| **Delete File** | `fs.unlink()` | Remove a file |
| **Rename File** | `fs.rename()` | Change file name |

# Reading File

**Reading File Asynchronously:**

```
import fs from 'fs';
fs.readFile('data.txt', 'utf-8', (err, data) => {
  if (err) throw err
  else console.log('Async Read Output:', data);
});
```

**Reading File Synchronously:**

```
import fs from 'fs';
try {
  const data = fs.readFileSync('data.txt', 'utf-8');
  console.log('Sync Read Output:', data);
} catch (err) { console.error('Error reading file:', err);}
```

# Writing and append data into file

Creating & Writing File:

```
import fs from 'fs';

fs.writeFile('example.txt', 'Hello Node.js!',

(err) => {

  if (err) throw err;

  console.log('File created successfully!');

});
```

Append File Content:

```
import fs from 'fs';

fs.appendFile('example.txt',

'\nWelcome to FS module.', (err) => {

  if (err) throw err;

  console.log('Data appended!');

});
```

Delete File -

```
import fs from 'fs';
fs.unlink('example.txt', (err) => {
  if (err) throw err;
  console.log('File deleted!');
});
```

Rename File -

```
import fs from 'fs';
fs.rename('example.txt', 'newfile.txt', (err) => {
  if (err) throw err;
  console.log('File renamed!');
});
```

# Thank you !!