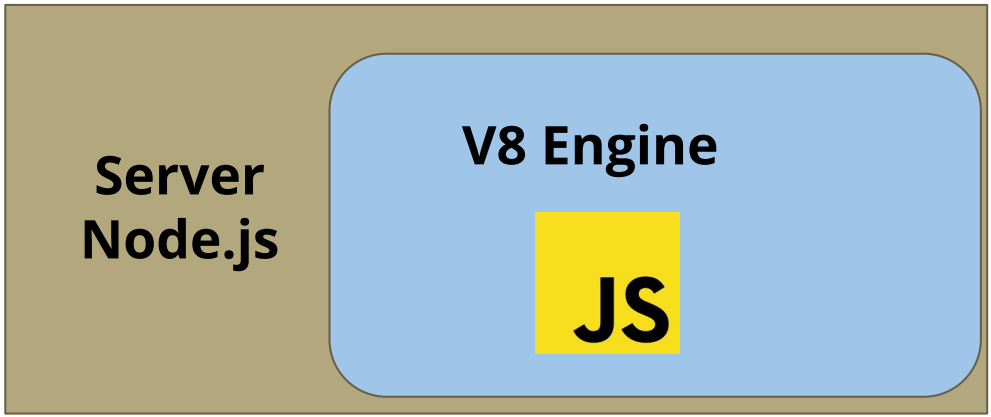
Node.js

_Anandkumar Jain, anandkumar.training@gmail.com

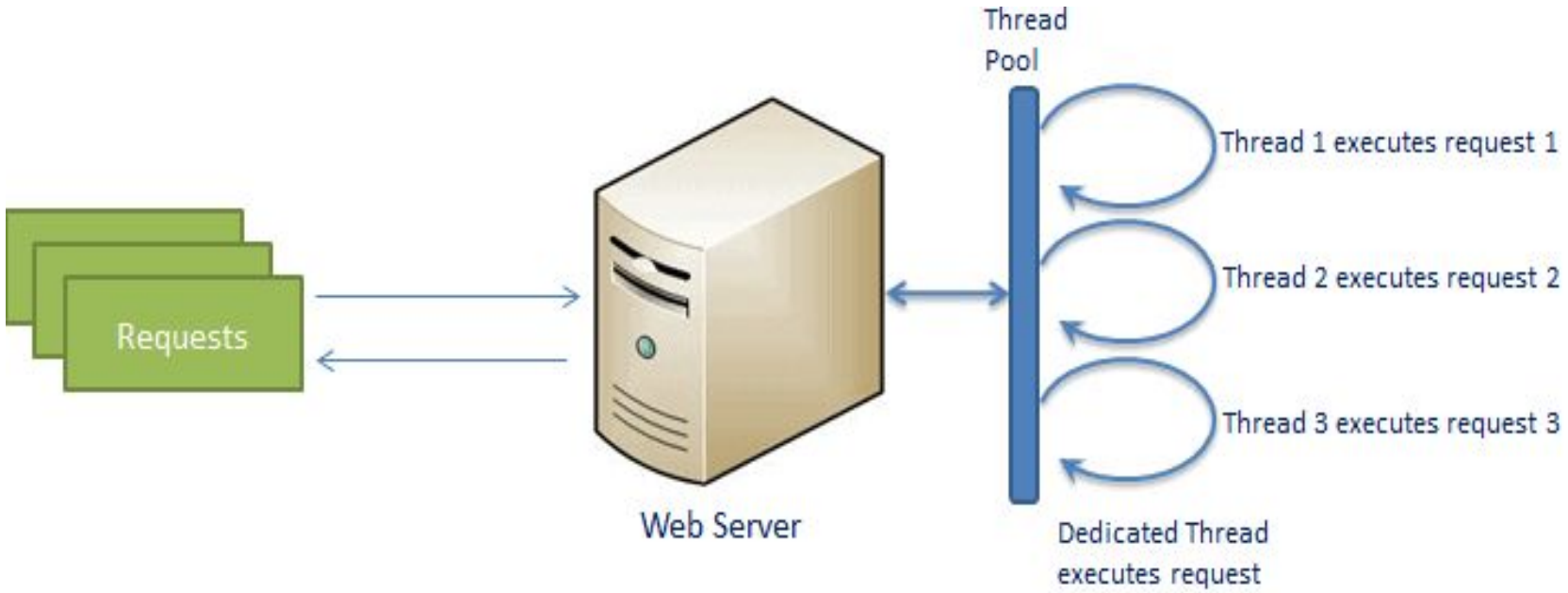
Introduction

Node.js is a **single-threaded**, **open-source**, **cross-platform** Javascript **runtime environment**, built on Google's open source **V8 Javascript Engine**, for building fast and scalable server-side and networking applications.

In 2009, Ryan Dahl, the creator of Node.js, took the V8 engine and embedded it in an application that could execute JavaScript on the server.



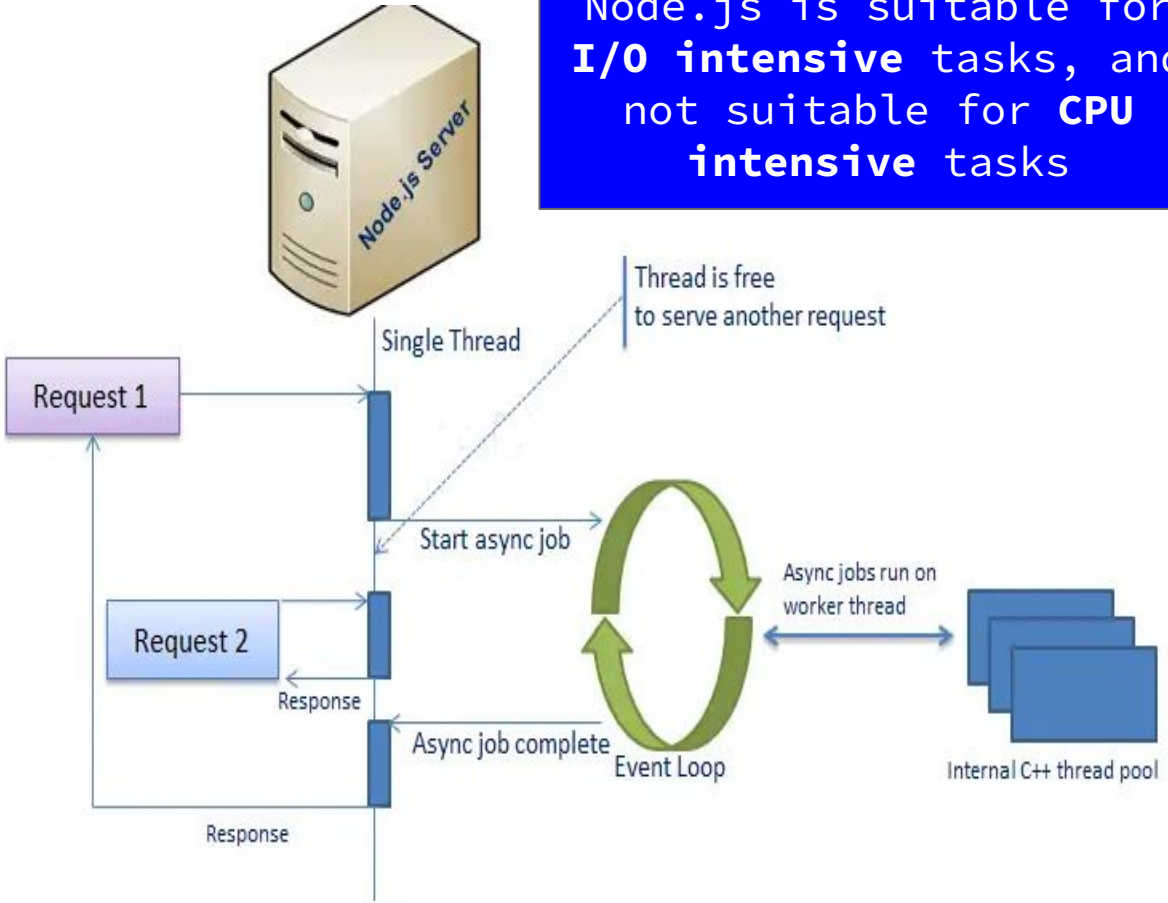
Working of Traditional Web Server Model



There is dedicated thread for every request. If there is no free thread, request has to wait.

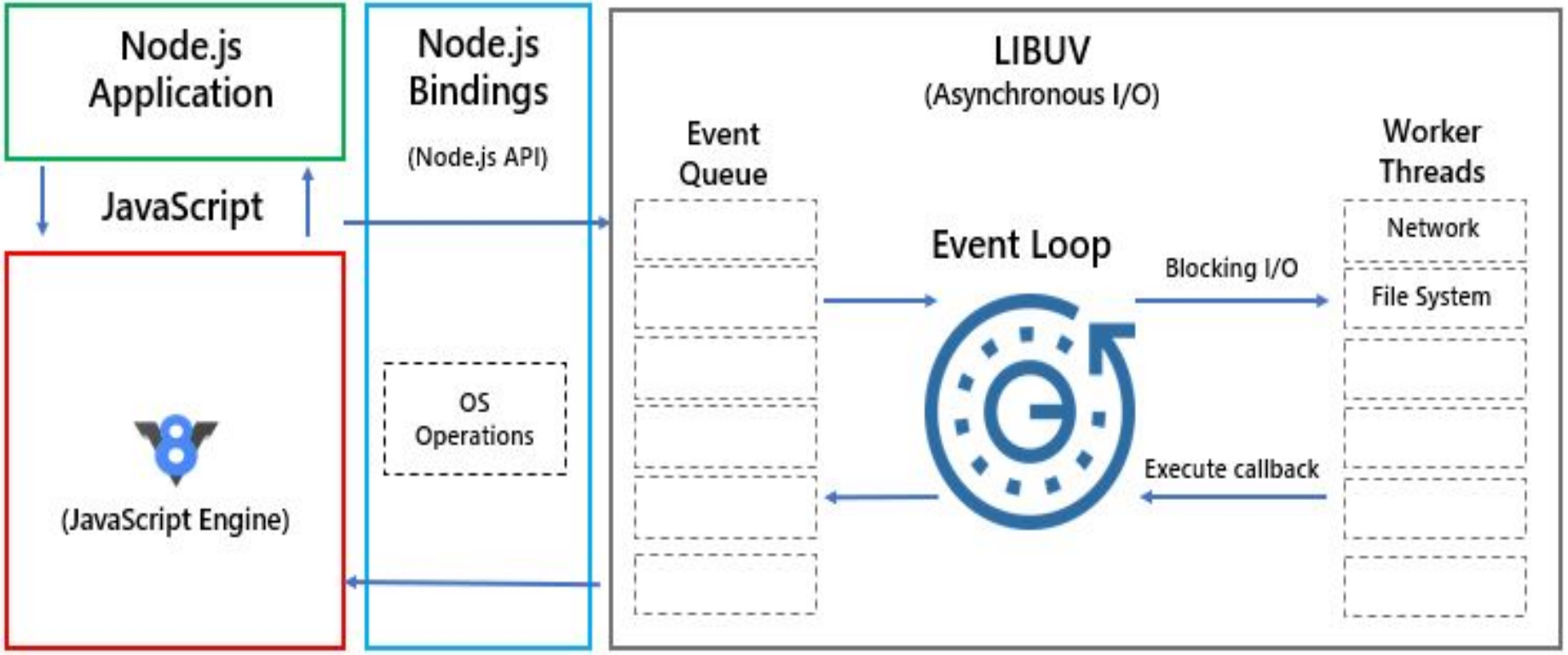
Node.js Model

Node.js is suitable for **I/O intensive** tasks, and not suitable for **CPU intensive** tasks



1) Single Thread - One request at a time
2) I/O is performed asynchronously. So thread does not have to wait for request to complete
3) An **event loop** executes callback function when the job completes
4) **libuv** is used to provide internal C++ thread pool for asynchronous I/O.

Node.js Architecture



Features of Node.js

- Node.js is Single-threaded

- Asynchronous by default. Node.js addresses blocking I/O issues by using non-blocking I/O requests. Meaning, we can continue making requests, while other tasks are going on

- Node.js is event-driven

- Lightweight framework that includes bare minimum modules. Other modules can be included as per the need of an application.

- Javascript everywhere

Installation and Setup

- Download latest LTS (Long Term Support) version of node.js from official website of node.js (https://nodejs.org/en/download/)

- After installation, open terminal in VS Code (or command prompt in Windows) and type node-v to check the version of node.js

- VS Code Extensions : ESlint, Prettier - Code Formatter, NPM Intellisense, Code Spell Checker, Auto Close Tag, Auto Rename Tag

REPL

- The Node.js Read-Eval-Print-Loop (REPL) is an interactive shell that processes Node.js expressions. The shell reads JavaScript code the user enters, evaluates the result of interpreting the line of code, prints the result to the user, and loops until the user signals to quit.

- Open the terminal and type node to open node REPL

- Press **CTRL + D** or type **.exit**, to exit REPL

- **REPL Commands**
  - Expressions
  - _ : Last result
  - Javascript file name
  - Function, Loop, Variables
  - Autocompletion
    - Math.
    - Math.sq
  - .save filename.js
  - .load filename.js
  - .help

Modules in Node.js

- **Module** in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

- **Types of Modules**
  - Core Modules
    - Provided by node itself
    - Import these modules using require()
    - Ex.: http, fs, os, path, url, querystring
  - Local Modules
    - Created locally in node application
    - We need to export the local modules, in-order to use them
  - Third Party Modules
    - The third party module can be downloaded by NPM
    - Command : npm install moduleName
    - Ex. : express, mongoose, validator

Using the Module

- Import the module → **let module = require('module_name');**
- The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.
- **Working of require()**
  - The require() function is an abstraction around an internal function, **_load()**
  - The _load function follows these steps:
    - Check **Module._cache** for a cached copy.
    - Create a new Module instance if not found in cache.
    - Save it to the cache.
    - Load contents of the module.
    - Compile the contents.
    - If there is an error, delete from the cache.
    - Return **module.exports**

Create and Export the Module (Local Module)

- Node.js treats each JavaScript file as a separate module.
- In the module (.js file), all the variables and functions are private. This means that they are invisible and cannot be used in other modules.
- To use the variables and functions of a module in another module, you need to export them at the end of the.js file.

Create and Export the Module (Local Module)

logger.js

```
const error='ERROR';
const warning="WARNING";
const info='INFO';

function log(message, level=info)
{
    console.log(`${level} :
${message}`);
}

module.exports = {log, error,
warning, info};
```

importmodule.js

```
const logger =
require('../logger');

logger.log("This is demo of
Local Module");

logger.log("Think of using ES6
Modules",logger.warning);
```

Using Core Modules

- **path Module :** The path module is one of the core modules in Node.js, designed to handle file paths and directory paths in a platform-independent way.

- **path Module Methods**
  - path.basename(path, [,ext])
  - path.dirname(path)
  - path.extname(path)
  - path.format(pathObj)
  - path.isAbsolute(path)
  - path.join(...path)
  - path.normalize(path)
  - path.parse(path)
  - path.relative(from, to)
  - path.resolve(...path)

- **path Module Properties**
  - path.sep
  - path.delimiter

Using Core Modules

- **os Module :** The os module in Node.js provides a set of operating system-related utility methods and properties. It allows you to access information about the operating system on which the Node.js process is running.

- **os Module Methods**
  - os.arch()
  - os.cpus()
  - os.endianness()
  - os.freemem()
  - os.homedir()
  - os.hostname()
  - os.platform()
  - os.release()
  - os.tmpdir()
  - os.totalmem()
  - os.type()
  - os.userInfo([options])

Using Core Modules

- **events Module :** Node.js provides a built-in events module that facilitates event-driven programming by allowing you to create custom events, emit events, and handle them asynchronously.
- **events Module Methods**
  - on(eventName, listener): Adds a listener function to the specified event.
  - off(eventName, listener): Removes a listener function of the specified event.
  - emit(eventName[, ...args]): Emits the specified event, triggering the invocation of all listener functions attached to that event.
- **EventEmitter Class:** The events module in Node.js provides the EventEmitter class, which serves as the foundation for event-driven programming. You can create instances of this class to emit events and handle them.

Using Core Modules - http Module (Creating Server)

- **The http module** in Node.js is a built-in module that provides functionality for creating HTTP servers and making HTTP requests. It allows you to interact with the HTTP protocol, enabling you to build web servers, handle incoming HTTP requests, and make outgoing HTTP requests to other servers.

- **Creating HTTP Servers:** You can create HTTP servers using the http.createServer() method. This method takes a request listener callback function as an argument, which is called each time the server receives an HTTP request.

- **Handling HTTP Requests:** When an HTTP request is received by the server, the request listener callback function is invoked with two arguments: req (the request object) and res (the response object). You can use these objects to handle the incoming request and send back an appropriate response.

Using Core Modules - http Module

- **HTTP Response Codes:** It also provides constants for HTTP response status codes such as 200 for OK, 404 for Not Found, 500 for Internal Server Error, etc.

- **Making HTTP Requests:** The http module allows you to make outgoing HTTP requests to other servers using the http.request() method. You can specify the request method, URL, headers, and request body.

- **Handling HTTP Responses:** When making an HTTP request, you can handle the response using event listeners such as 'response', 'data', and 'end'. These listeners allow you to process the response data as it arrives and handle the end of the response.

Third Party Modules

- **The third party module** can be downloaded by NPM (Node Package Manager). These types of modules are developed by others and we can use that in our project. Some of the best third party module examples are listed as follows: express, gulp, lodash, async, socket.io, mongoose, underscore, pm2, bower, q, debug, react, mocha etc. Third party modules can be installed inside the project folder or globally.

- **Example:**
  - npm install express
  - npm install mongoose
  - npm install -g @angular/cli

- **NPM and package.json**

Module Wrapper Function

- In Node.js, a module wrapper function is a function that wraps every module within a function scope. This function helps encapsulate the module's code and provides a set of variables and functions that are available within the module's scope but are not accessible from outside.

- The module wrapper function typically looks like this:

```
(function(exports, require, module, __filename, __dirname)
{
 // Module code actually lives in here
});
```

Module Wrapper Function

- exports: This is an object that is used to define exports from the module. Anything assigned to exports becomes accessible from outside the module when it is required elsewhere.
- require: This function is used to import other modules or files into the current module.
- module: This is an object that represents the current module. It has properties like module.exports, which can be used to export values from the module.
- __filename: A string representing the filename of the current module.
- __dirname: A string representing the directory name of the current module.

By wrapping the module code within this function, Node.js ensures that variables and functions declared within the module do not pollute the global scope.