NITTE EDUCATION TRUST | NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

(An Autonomous Institution | Accredited with 'A+' Grade by NAAC | NBA | Approved By UGC | AICTE | Govt. of Karnataka | Affiliated to VTU, Belagavi)
Yelahanka, Bangalore - 560 064

KNOWLEDGE • CHARACTER • UNITY

Department of Computer Science and Engineering

5th Semester , Program Elective

Advanced Web Programming-I

(18CSE536), 3 Credits

Dr. Anil Kumar

Assistant Professor

Dept. of CSE, NMIT

AY 2020-2021

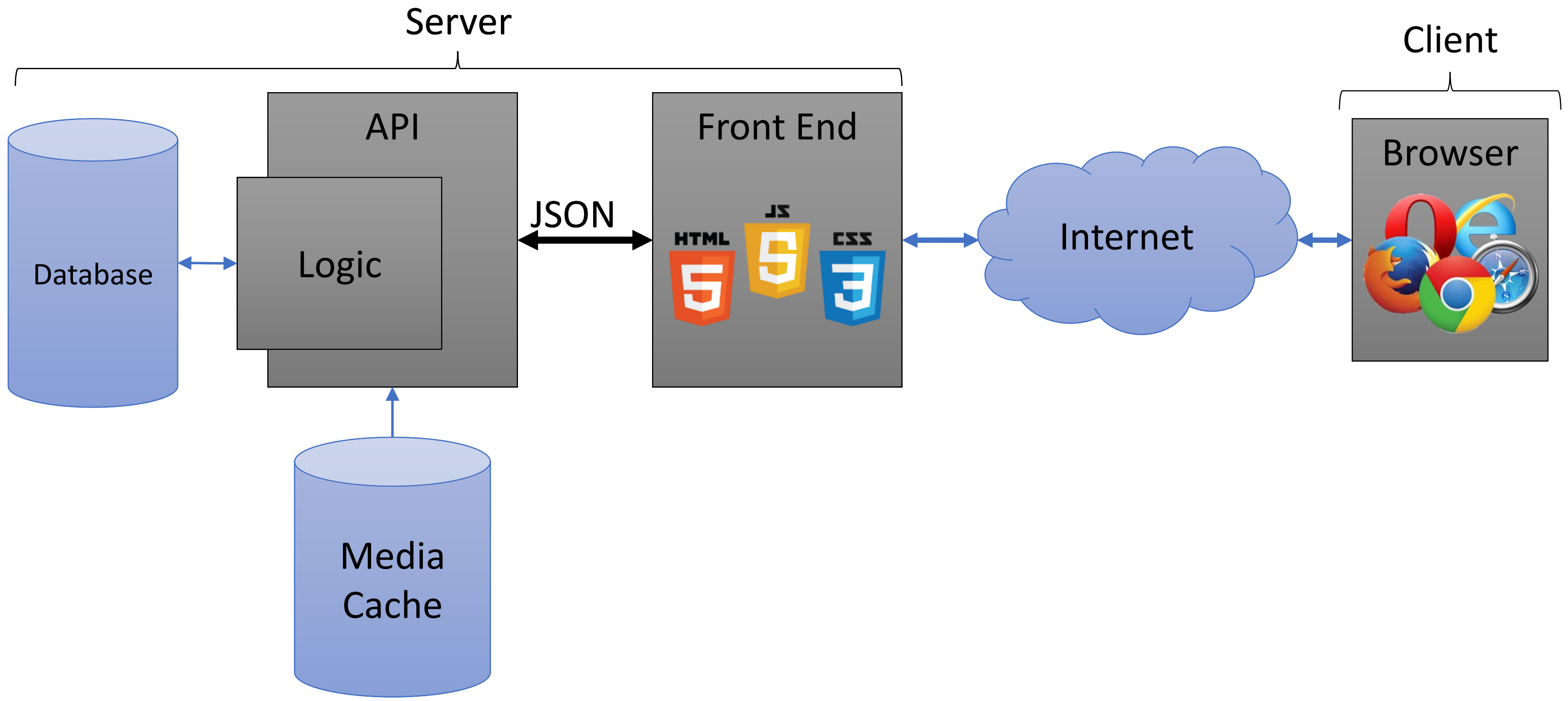
Unit-1
Introduction to Node.JS

Principles of Web Design/ Applications

- Availability
- Performance
- Reliability
- Scalability
- Manageability
- Cost

Core Components of Web Applications

- UI (Front End (DOM, Framework))
- Request Layer (Web API)
- Back End (Database, Logic)



Tools and Frameworks used for Web Development

django
Pylons
GRAILS
The Search is over
Microsoft
.NET
Framework
Flask
play!
spring
by Pivotal
Symfony
CakePHP
laravel
django
CMS
mochikit
dojo
handlebars
sitecore
Own the experience
ember
GRUNT
Knockout.
MooTools
prototype
jQuery
Joomla!
node js
npm
BACKBONE.JS
Magento
Open Source eCommerce

Back-End Frameworks

node JS™

LoopBack.io
Node.js API Framework

express

™

RAILS

django

restify

Front-End Frameworks

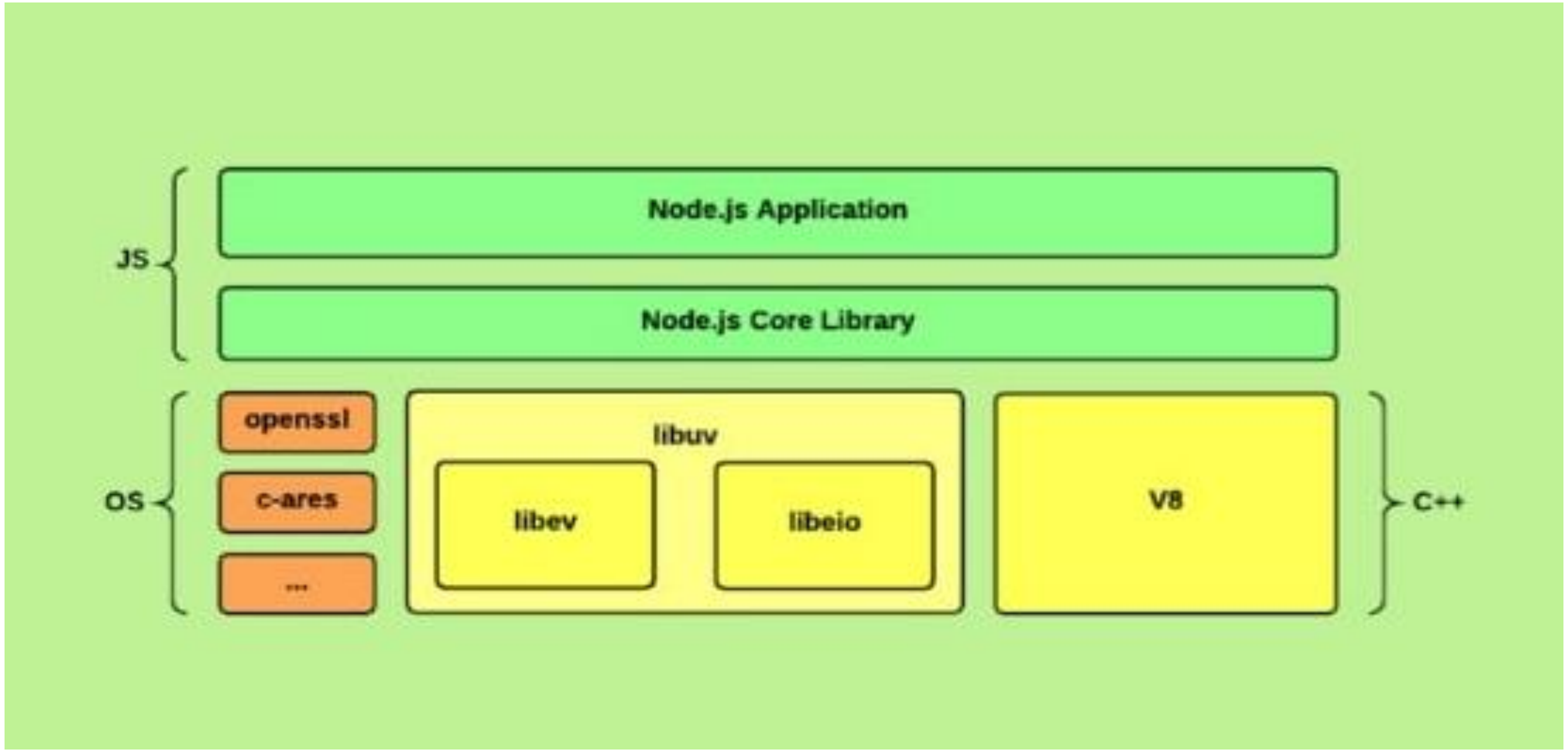
ENYO
GWT
Ractive.js
Sammy.js
dojo
canjs
mochikit
jQuery
ember
Knockout.
ext js
METEOR

WHAT IS NODE?

What is Node.js?

- Open Source, Cross Platform Development platform
- Uses V8 JavaScript engine (Google)
- A JavaScript runtime environment running Google Chrome’s V8 engine
  - a.k.a. a server-side solution for JS
- V8 is a open source JavaScript engine developed by Google.
- It is written in C++ and is used in Google Chrome Browser.
- Node.js is a command line tool.
- Runs over the command line.
- Designed for high concurrency
  - Without threads or new processes
  - Making it really fast

- Never blocks, not even for I/O
    - Created by Ryan Dahl starting in 2009
- Node.js is a platform (is not a framework)
    - Express is a framework for Node.js
- Goal is to provide an easy way to build scalable network programs.
- It makes communication between client and server happen in same language (JavaScript)
- Node is a platform for writing JavaScript applications outside web browsers.
- There is no DOM built into Node, nor any other browser capability.


JS
Node.js Application
Node.js Core Library
OS
openssl
c-ares
...
libuv
libev
libeio
V8
C++

Concurrency: The Event Loop

- Instead of threads Node uses an event loop with a stack
  - Alleviates overhead of context switching
  - Event loop as a language construct instead of as a library
  - It process each request as events
    - E.g., HTTP server doesn’t wait for the IO operation to complete while it can handle other request at the same time.
  - To avoid blocking, it makes use of the event driven nature of JavaScript by attaching callbacks to I/O requests

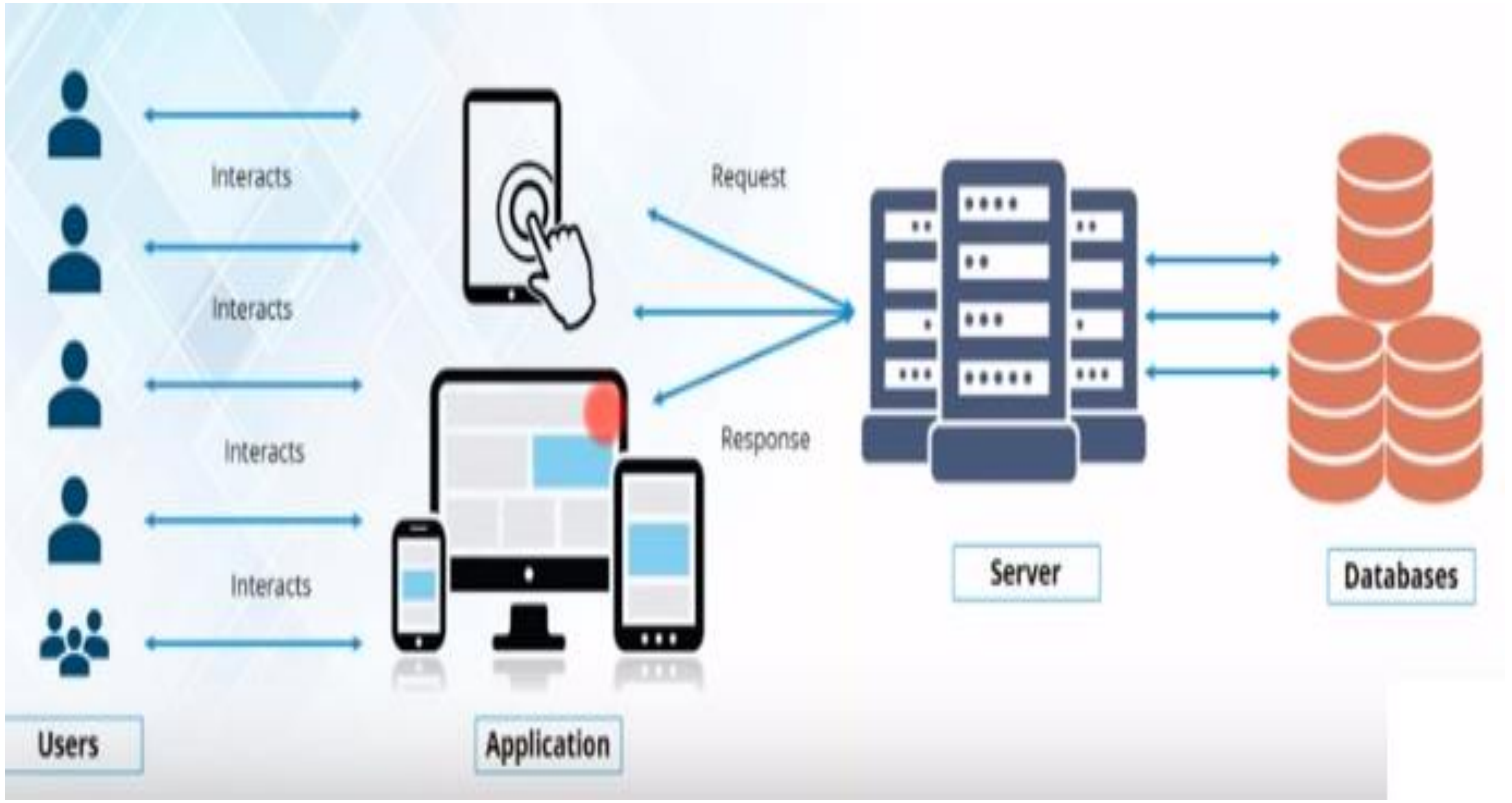
Threads VS Event-driven

Threads	Event-driven
Using incoming-request model	Using queue and then processes it
multithreaded server might block the request which might involve multiple events	manually saves state and then goes on to process the next event
Using context switching	no contention and no context switches
Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock	Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments

Node.js Architecture

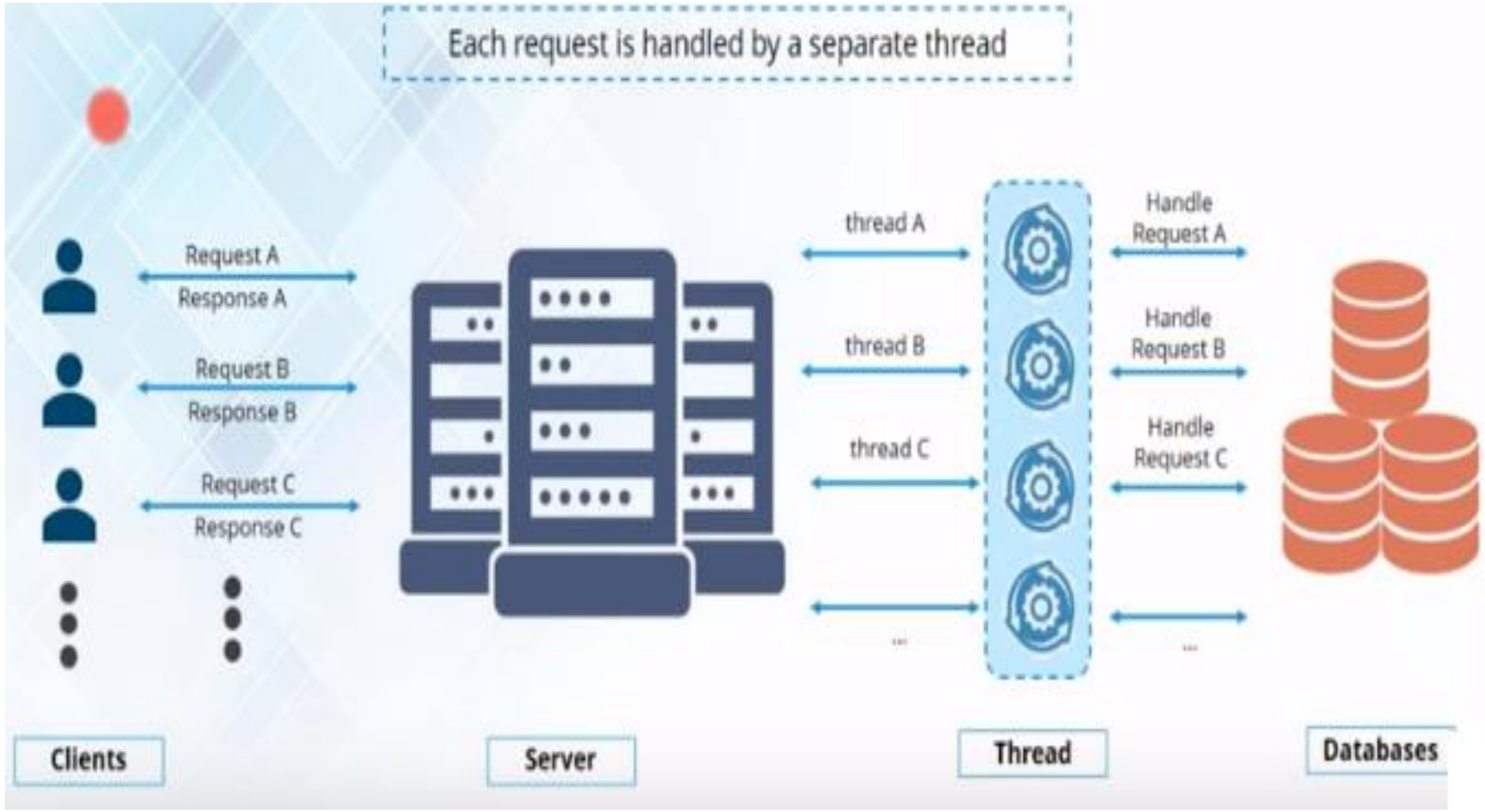
- Node.js is event-driven with Non-blocking I/O
- Uses a Event loop for Non-blocking I/O
- Node.js is single threaded

Client Server Architecture



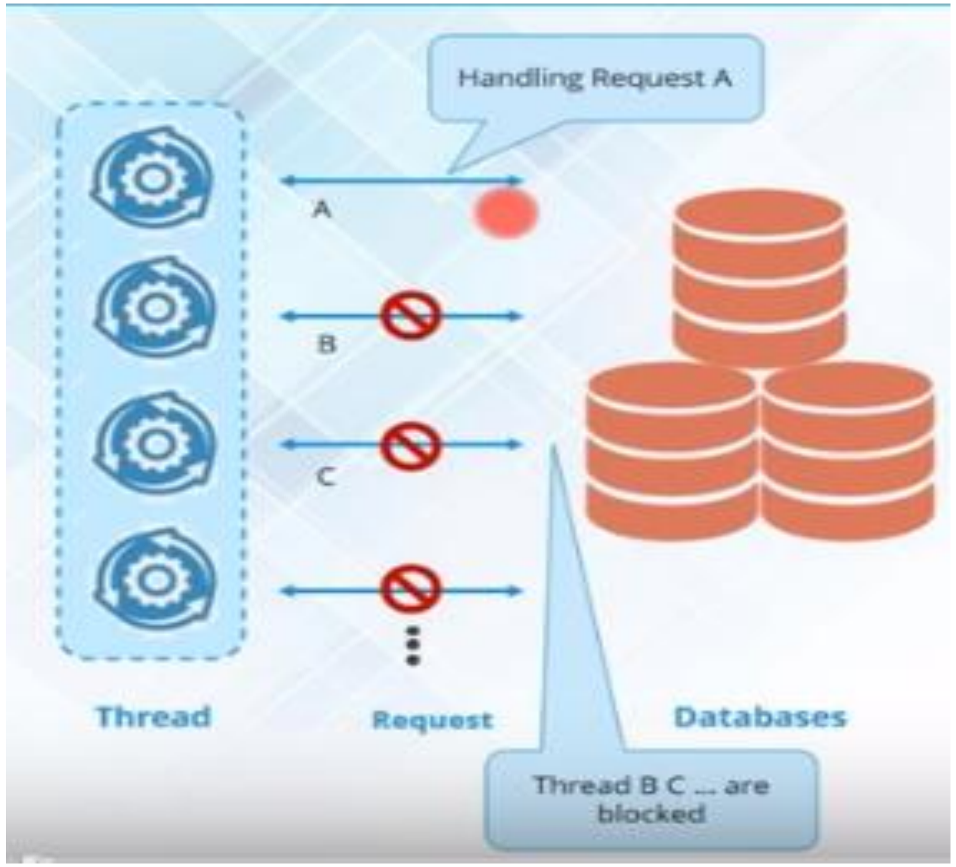
MultiThread Model

- **Limitation 1:** Threads in the Thread pool are limited and as the number of requests exceed the no of threads, threads are exhausted.



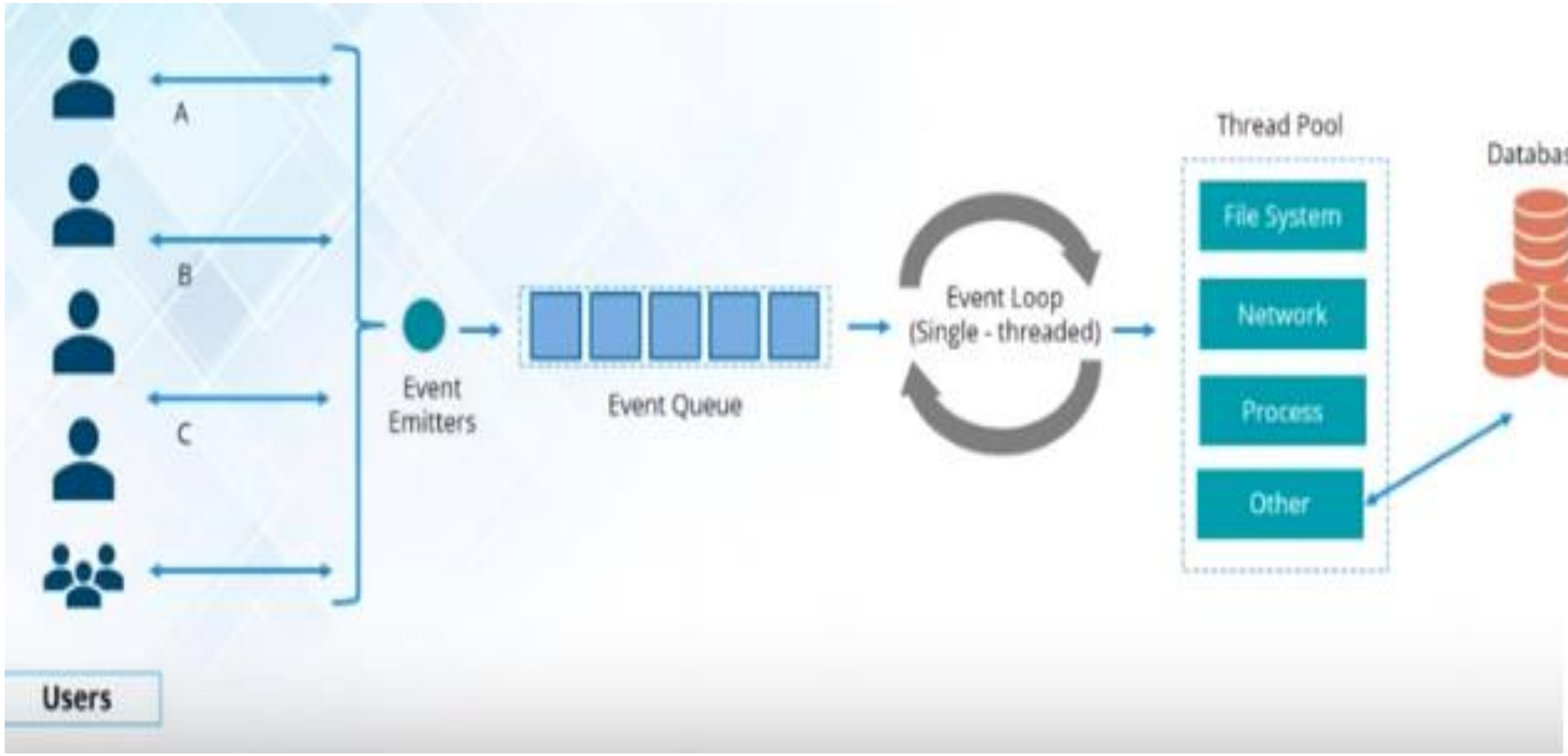
MultiThread Model

- **Limitation 2**: If a thread acquires a lock of the shared resource, it is a exclusive lock. It will block other threads from accessing the resource.
- This will affect the response time of the server.

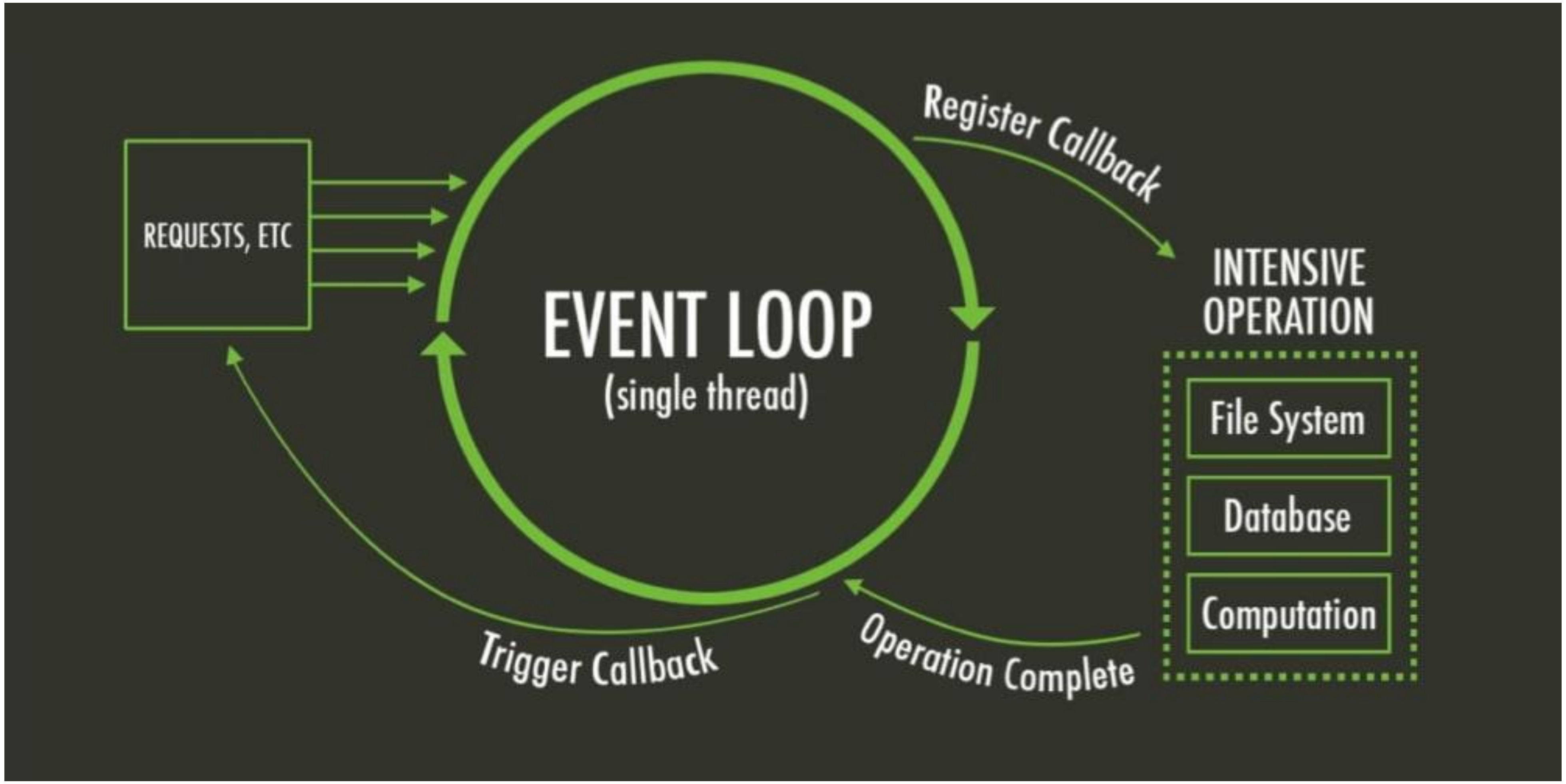


Single Threaded Model

- Node JS is event driven, handling all request asynchronously from single thread.



Event Loop in Node.js



Blocking code vs Nonblocking code

Blocking Code

```
Read the file from the File System, when done reading the contents,
        Print the contents.
Do something else
```

NonBlocking Code

```
Read the file from the File System,
        Whenever  you are complete, Print the contents.
Do something else
```

This is a callback function

Blocking Code

```
var contents = fs.readFileSync(“filename”);
Console.log(contents);
console.log(“Do something else”);
```

Non Blocking Code

```
fs.readFile(“filename”, function (err, contents){
        console.log(contents);
}
console.log(“DO something else”);
```

Node.js/HTTP vs. Apache

- Node.js/HTTP
- It's fast
- It can handle tons of concurrent requests
- It's written in JavaScript (which means you can use the same code server side and client side)Platform

Platform	Number of request per second
PHP (Via Apache)	318727
Static (Via Apache)	296651
Node.Js	556930

COMPANIES THAT USE NODE.JS

Walmart	ebay	PayPal	DOWJONES
intuit.	NETFLIX	Linked in	The New York Times
Microsoft	Uber	YAHOO!	Kingfisher
Bank of America	NBC	STAPLES	NASA
BEST BUY	Cigna.	Bloomberg	Alibaba.com
TARGET	FANDANGO	PG&E	WELLS FARGO
Symantec.	Go Daddy	macy's	DIRECTV

Installation of Node JS

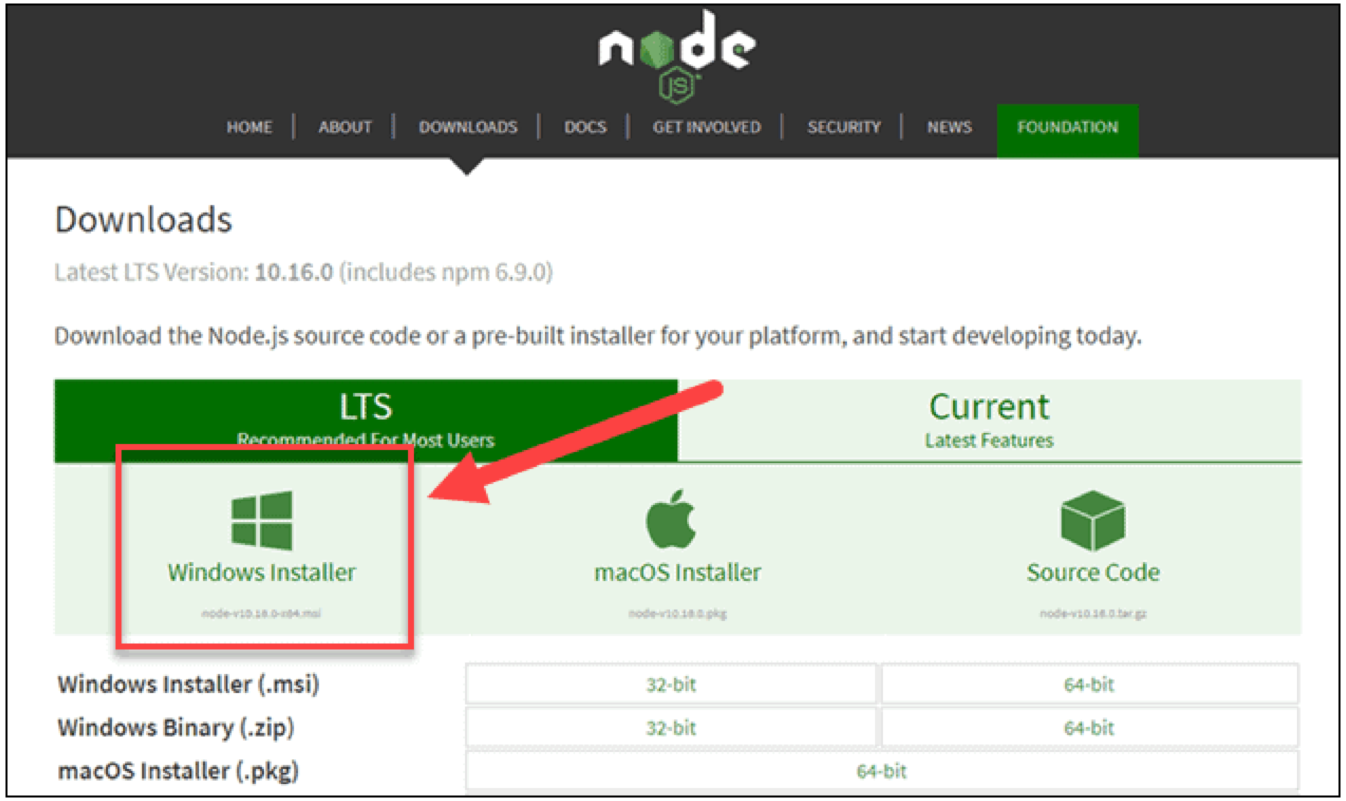
- Download from official site
- https://nodejs.org/en/download/

- Node is command line, on successful installation open cmd and check version using the command

node –v

Steps required to install Node.js and NPM on Windows

- **Step 1: Download Node.js Installer**
  - In a web browser, navigate to https://nodejs.org/en/download/. Click the Windows Installer button to download the latest default version. At the time this article was written, version 10.16.0-x64 was the latest version. The Node.js installer includes the NPM package manager.



Downloads

Latest LTS Version: **10.16.0** (includes npm 6.9.0)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

node-v10.16.0-x64.msi

node-v10.16.0.pkg

node-v10.16.0.tar.gz

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit	

- **Step 2: Install Node.js and NPM from Browser**
  - 1. Once the installer finishes downloading, launch it. Open the **downloads** link in your browser and click the file. Or, browse to the location where you have saved the file and double-click it to launch.
  - 2. The system will ask if you want to run the software – click **Run**.
  - 3. You will be welcomed to the Node.js Setup Wizard – click **Next**.
  - 4. On the next screen, review the license agreement. Click **Next** if you agree to the terms and install the software.
  - 5. The installer will prompt you for the installation location. Leave the default location, unless you have a specific need to install it somewhere else – then click **Next**.
  - 6. The wizard will let you select components to include or remove from the installation. Again, unless you have a specific need, accept the defaults by clicking **Next**.
  - 7. Finally, click the **Install** button to run the installer. When it finishes, click **Finish**.
- **Step 3: Verify Installation**
  - Open a command prompt (or PowerShell), and enter the following:
    - node -v
  - The system should display the Node.js version installed on your system. You can do the same for NPM:
    - npm -v

Getting Started with Node JS: First Node JS program

- A Node.js application consists of the following three important components
  - **Import required modules** − We use the **require** directive to load Node.js modules.
  - **Create server** − A server which will listen to client's requests similar to Apache HTTP Server.
  - **Read request and return response** − The server created in an earlier step will read the HTTP request made by the client which can be a browser or a console and return the response.

Creating Node.js Application

- Step 1 - Import Required Module
  - We use the require directive to load the http module and store the returned HTTP instance into an http variable as follows
    - var http = require("http");

- Step 2 - Create Server
  - We use the created http instance and call http.createServer() method to create a server instance and then we bind it at port 8081 using the listen method associated with the server instance. Pass it a function with parameters request and response. Write the sample implementation to always return "Hello World".

```
http.createServer(function (request, response) {
   // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
   response.writeHead(200, {'Content-Type': 'text/plain'});
   // Send the response body as "Hello World"
   response.end('Hello World\n');
}).listen(8081);
// Console will print the message
console.log('Server running at http://127.0.0.1:8081/’);
```

- The above code is enough to create an HTTP server which listens, i.e., waits for a request over 8081 port on the local machine.

- Step 3 - Testing Request & Response
  - Let's put step 1 and 2 together in a file called hello_world.js and start our first HTTP server.
  - Now execute the hello_world.js to start the server as follows
    - $ node main.js
  - Verify the Output. Server has started.
    - Server running at http://127.0.0.1:8081/

Hello World.js

```
var http = require('http');
 http.createServer(function (req, res) {
   res.writeHead(200, {'Content-Type': 'text/html'});
   res.end('Hello World!');
 }).listen(8080);
```

When to use Node.js?

- Node.js allows the creation of **web servers** and **networking tools**, using **JavaScript** and a collection of **modules** that handle various core functionality.
- **Modules** handle file system I/O, networking, binary data (buffers), cryptography functions, data streams, etc.
- Node.js is **not limited to the server-side**, it provides a number of tools for **frontend development workflows**.

Node.js Modules

- Consider modules to be the same as JavaScript libraries.
- A set of functions you want to include in your application.
- Built-in Modules

    -Node.js has a set of built-in modules which you can use without any further installation.

What Can You Do With Node.js

- Web Applications
  - Real-time applications
  - Event-based web sites
- REST (Representational State Transfer) Web Services
- IOS/Android
- Desktop Applications
  - Linux/Windows/OS X

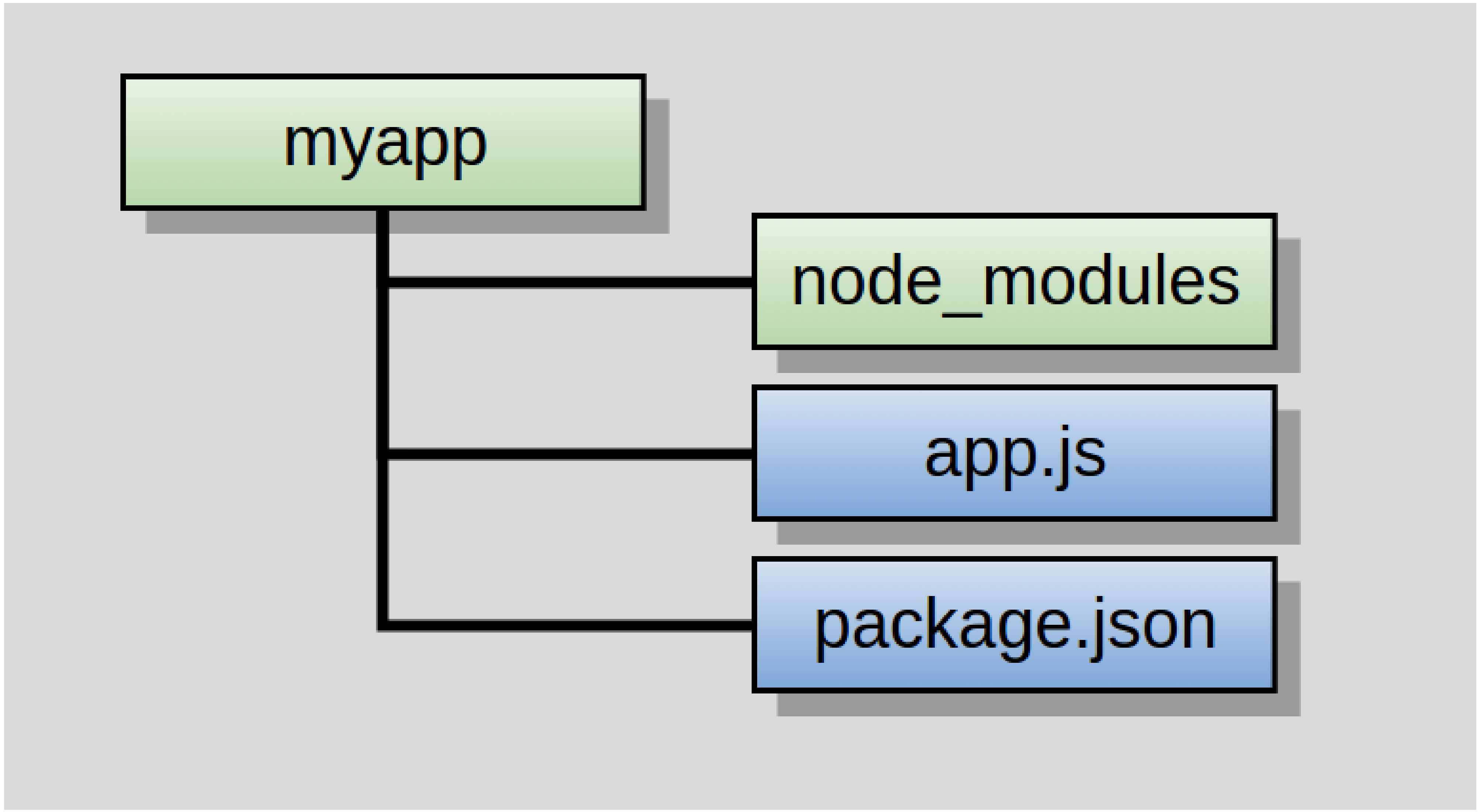
assert	Provides a set of assertion tests
buffer	To handle binary data
child_process	To run a child process
cluster	To split a single Node process into multiple processes
crypto	To handle OpenSSL cryptographic functions
dgram	Provides implementation of UDP datagram sockets
dns	To do DNS lookups and name resolution functions
Domain Deprecated.	To handle unhandled errors
events	To handle events
fs	To handle the file system
http	To make Node.js act as an HTTP server
https	To make Node.js act as an HTTPS server.
net	To create servers and clients
os	Provides information about the operation system
path	To handle file paths
querystring	To handle URL query strings
readline	To handle readable streams one line at the time
stream	To handle streaming data
string_decoder	To decode buffer objects into strings
timers	To execute a function after a given number of milliseconds
tls	To implement TLS and SSL protocols
url	To parse URL strings
util	To access utility functions
V8	To access information about V8 (the JavaScript engine)
vm	To compile JavaScript code in a virtual machine
zlib	To compress or decompress files

NODE PACKAGE MANAGER(NPM)

- Provides online repositories for node.js packages / modules. Provides command line utility to install Node.js packages.

NODE PACKAGE REGISTRY

- The Node modules have a managed location called the Node Package Registry where packages are registered. This allows you to publish your own packages in a location where others can use them as well as download packages that others have created.
- The Node Package Registry is located at https://npmjs.com.


myapp
node_modules
app.js
package.json

What is package.json?

- package.json is a plain JSON(Java Script Object Notation) text file which contains all metadata information about Node JS Project or application.
- Every Node JS Package or Module should have this file at root directory to describe its metadata in plain JSON Object format.
- We should use same file name with same case “package” and same file extension “*.json”.
- Why that filename is “package”: because Node JS platform manages every feature as separate component. That component is also known as “Package” or “Module”.

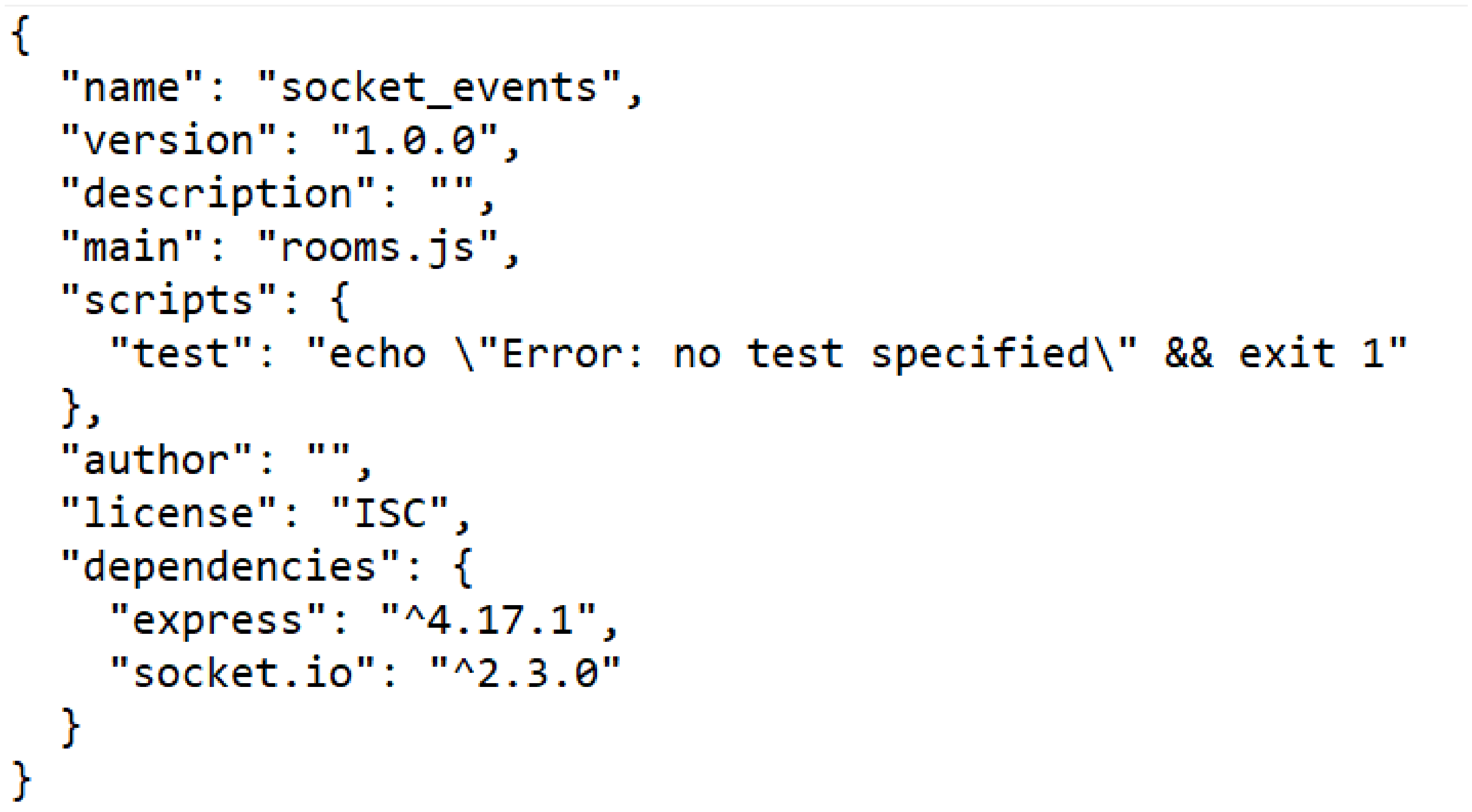
Who uses package.json file?

- NPM (Node Package Manager) uses this package.json file information about Node JS Application information or Node JS Package details.
- package.json file contains a number of different directives or elements. It uses these directives to tell NPM “How to handle the module or package”.
- The package.json files contains information such as the project description, the version of the project in a particular distribution, license information, and configuration data.
- The package.json file is normally located at the root directory of a Node.js project

Directives used in the **`package.json`** file

Directive	Description	Example
`name`	Unique name of package.	`"name": "camelot"`
`preferGlobal`	Indicates this module prefers to be installed globally.	`"preferGlobal": true`
`version`	Version of the module.	`"version": 0.0.1`
`author`	Author of the project.	`"author": "arthur@???.com"`
`description`	Textual description of module.	`"description": "a silly place"`
`contributors`	Additional contributors to the module.	`"contributors": [ { "name": "gwen", "email": "gwen@???.com"}]`
`bin`	Binary to be installed globally with project.	`"bin: { "excalibur": "./bin/excalibur"}`
`scripts`	Specifies parameters that execute console apps when launching node.	`"scripts" { "start": "node ./bin/excalibur", "test": "echo testing"}`
`main`	Specifies the main entry point for the app. This can be a binary or a .js file.	`"main": "./bin/excalibur"`

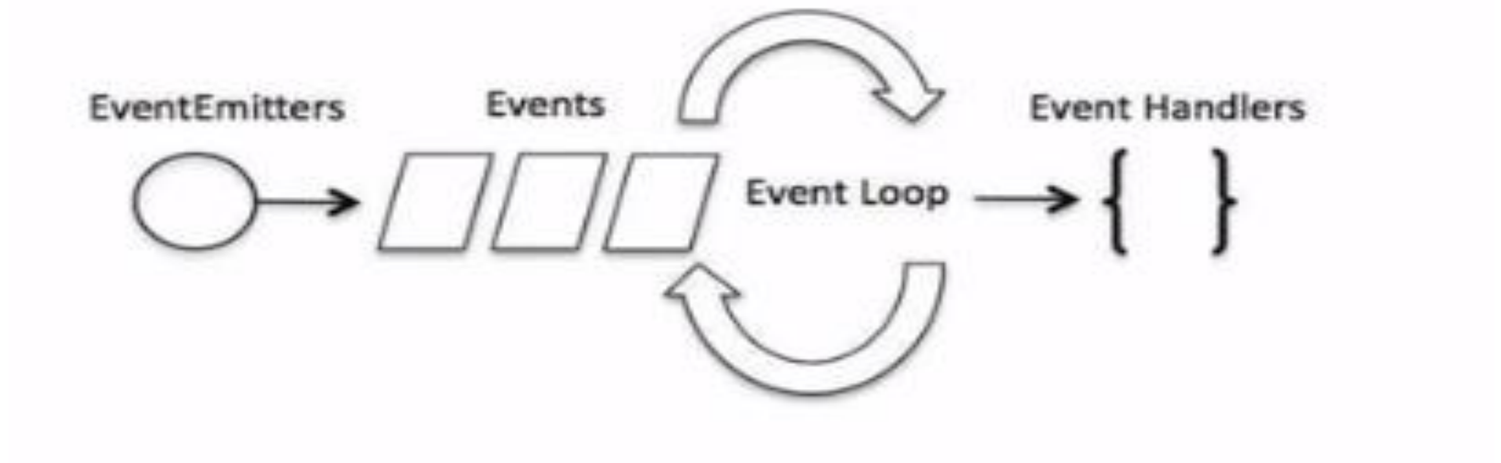
Example of Package.Json



```json
{
  "name": "socket_events",
  "version": "1.0.0",
  "description": "",
  "main": "rooms.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "socket.io": "^2.3.0"
  }
}
```

NODE JS Events

- Single threaded.
- Supports concurrency via events and callbacks. EventEmitter class is used to bind events and event listeners.



- Much of the Node.js core API is built around an idiomatic asynchronous event-driven architecture in which certain kinds of objects (called "emitters") emit named events that cause Function objects ("listeners/event handlers") to be called.

NODE JS Events

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

//Create an event handler:
var myEventHandler = function () {
  console.log(‘Message logged in');
}

//Assign the event handler to an event:
eventEmitter.on('log', myEventHandler);

//Fire the 'log' event:
eventEmitter.emit('log');
```

Run program event.js

Import the event class

Create an object of the Emitter class

Response to an event.

Trigger an event

*****on** property is used to bind a function with the event.
emit is used to fire an event.

Event.js

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
//Create an event handler:
var myEventHandler = function () {
  console.log(‘Message logged in');
}

//Assign the event handler to an event:
eventEmitter.on('log', myEventHandler);

//Fire the 'log' event:
eventEmitter.emit('log');
```

Node.js EventEmitter

- Node.js allows us to create and handle custom events easily by using events module.
- EventEmitter is a class that helps us create a publisher-subscriber pattern in NodeJS
- EventEmitter class is responsible to generate events. Generating events is also known as Emitting. That’s why this class is named as EventEmitter.
- Event module includes EventEmitter class which can be used to raise and handle custom events.
- Example: Raise and Handle Node.js events
- we first import the 'events' module and then create an object of EventEmitter class.
- We then specify event handler function using on() function. The on() method requires name of the event to handle and callback function which is called when an event is raised.

```
// get the reference of EventEmitter class of events module
var events = require('events');
//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();
//Subscribe for FirstEvent
em.on('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});
// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.’);
```

- The emit() function raises the specified event. First parameter is name of the event as a string and then arguments.
- An event can be emitted with zero or more arguments. You can specify any name for a custom event in the emit() function.

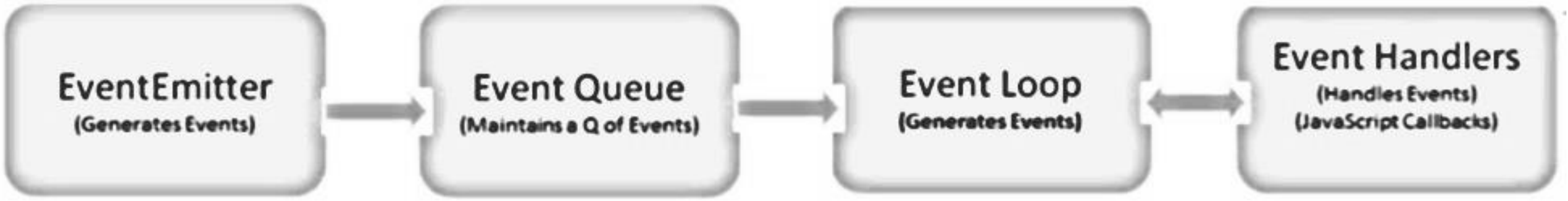
EventEmitter Methods	Description
emitter.addListener(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added.
emitter.on(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. It can also be called as an alias of emitter.addListener()
emitter.once(event, listener)	Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed.
emitter.removeListener(event, listener)	Removes a listener from the listener array for the specified event. Caution: changes array indices in the listener array behind the listener.
emitter.removeAllListeners([event])	Removes all listeners, or those of the specified event.
emitter.setMaxListeners(n)	By default EventEmitters will print a warning if more than 10 listeners are added for a particular event.
emitter.getMaxListeners()	Returns the current maximum listener value for the emitter which is either set by emitter.setMaxListeners(n) or defaults to EventEmitter.defaultMaxListeners.
emitter.listeners(event)	Returns a copy of the array of listeners for the specified event.
emitter.emit(event[, arg1][, arg2][, ...])	Raise the specified events with the supplied arguments.
emitter.listenerCount(type)	Returns the number of listeners listening to the type of event.

Class Methods

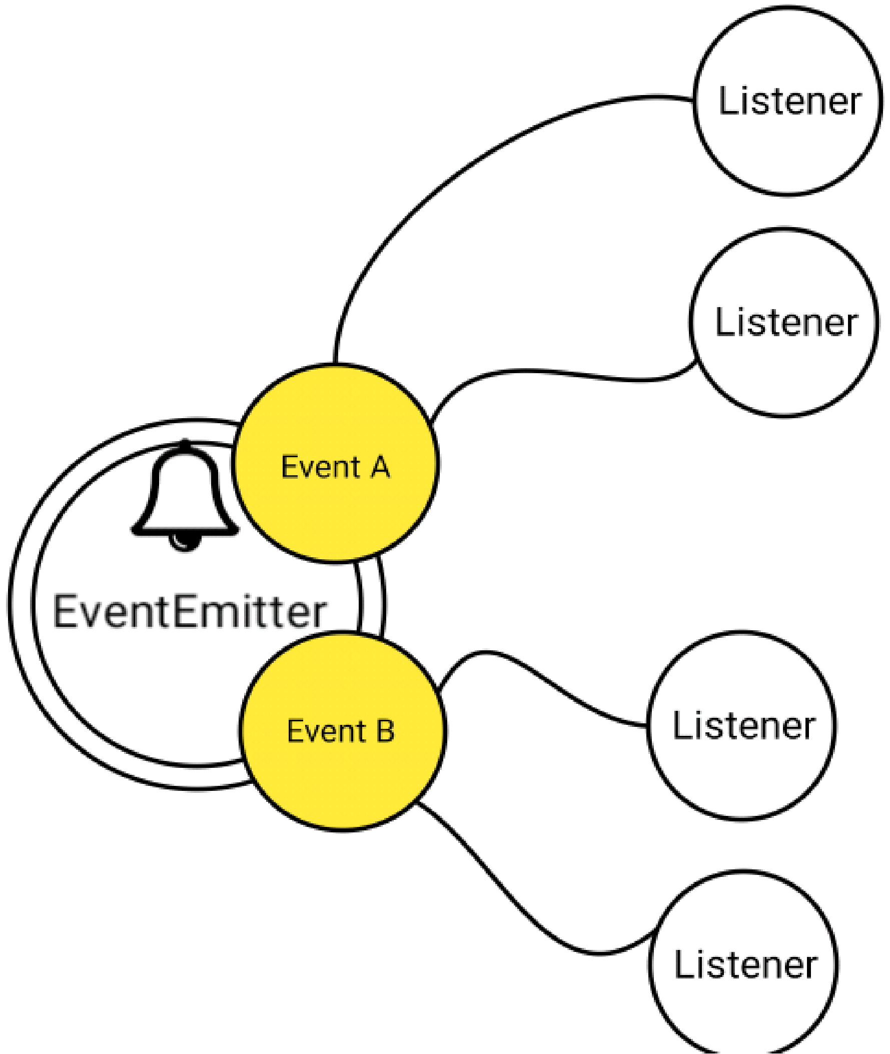
Sr.No.	Method & Description
1	**listenerCount(emitter, event)** Returns the number of listeners for a given event.

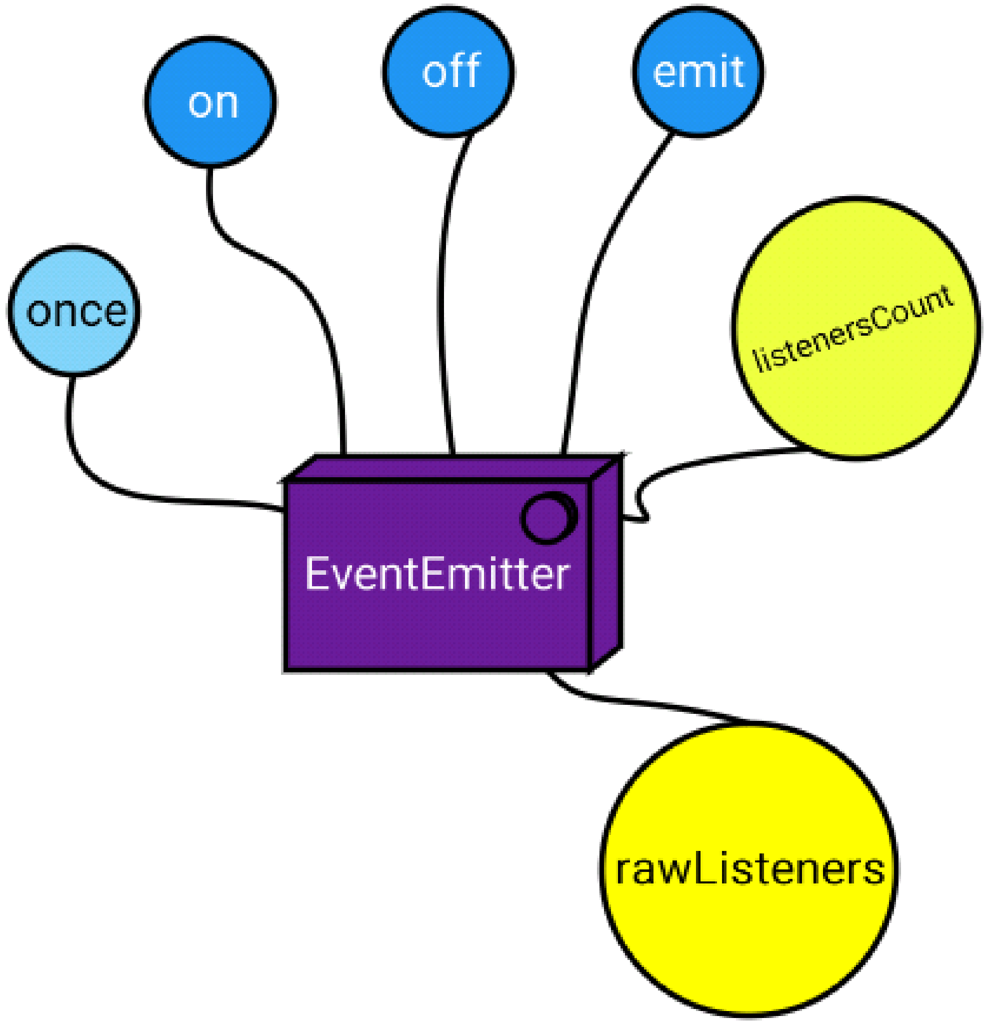
Events

Sr.No.	Events & Description
1	**newListener** •**event** − String: the event name •**listener** − Function: the event handler function This event is emitted any time a listener is added. When this event is triggered, the listener may not yet have been added to the array of listeners for the event.
2	**removeListener** •**event** − String The event name •**listener** − Function The event handler function This event is emitted any time someone removes a listener. When this event is triggered, the listener may not yet have been removed from the array of listeners for the event.



Node JS Event Driven Programming





Timers module

- The Timers module in Node.js contains various functions that allow us to execute a block of code or a function after a set period of time.
- The Timers module is global, hence it do not need to use require() to import it.
- Timers module has divided in to two parts

  1)Scheduling Timers

  2)Cancelling Timers

Scheduling functions:

- **setImmediate()**: It is used to execute setImmediate.
- **setInterval()**: It is used to define a time interval.
- **setTimeout()**: It is used to execute a one-time callback after delay milliseconds.

Cancelling Functions:

- **clearImmediate(immediateObject)**: It is used to stop an immediateObject, as created by setImmediate
- **clearInterval(intervalObject)**: It is used to stop an intervalObject, as created by setInterval
- **clearTimeout(timeoutObject)**: It prevents a timeoutObject, as created by setTimeout

Callbacks in Node.js

- Callback is an asynchronous equivalent for a function.
- A callback function is called at the completion of a given task. Node makes heavy use of callbacks.
- All the APIs of Node are written in such a way that they support callbacks.
- Asynchronous functions are also known as non-blocking functions since they do not block the thread on which they are running on. Node.js relies heavily on asynchronous functions.
  - For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed.
  - Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So there is no blocking or wait for File I/O.
  - This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

Callback functions in Node.js

- NodeJS has asynchronous callbacks and commonly supplies two parameters to your functions sometimes conventionally called err and data. An example with reading a file text.

```
const fs = require("fs");
fs.readFile("./test.txt", "utf8", function(err, data) {
    if(err) {
        // handle the error
    } else {
        // process the file text given with data
    }
});
```

- UTF-8 is a variable-width character encoding used for electronic communication.

Blocking and Non-Blocking Code

Blocking Code

```
var fs = require("fs");
var data = fs.readFileSync('input.txt');

console.log(data.toString());
console.log("Program Ended");
```

Non-Blocking Code

```
var fs = require("fs");
fs.readFile('input.txt', function (err, data) {
   if (err) return console.error(err);
   console.log(data.toString());
});
console.log("Program Ended");
```

The above two examples explain the concept of blocking and non-blocking calls.

- The first example shows that the program blocks until it reads the file and then only it proceeds to end the program.
- The second example shows that the program does not wait for file reading and proceeds to print "Program Ended" and at the same time, the program without blocking continues reading the file.
- Thus, a blocking program executes very much in sequence. From the programming point of view, it is easier to implement the logic but non-blocking programs do not execute in sequence.

- Async functions on the other hand give us a way of doing tasks in parallel thereby minimizing execution time. This intern makes servers run faster, providing a good user experience.
- Callbacks are to be used when we *don’t know* when something will be done. Again, think of something like an API call, fetching data from a database or I/O with the hard drive.
- All of these will take time, so we want our callback to be called when the event we are waiting for is done. Hence the term **event-driven programming.**

Input and Output

- Computers have many senses -- keyboard, mouse, network card, camera, joystick, etc. Collectively, these are called **INPUT**.
- Computers can also express themselves in many ways -- text, graphics, sound, networking, printing, etc. Collectively, these are called **OUTPUT**.
- Input and Output together are called **I/O**.
- <u>**Memory vs I/O**</u>
- Performing *calculations* and accessing *memory* is **very fast**
- ...but reading and writing to I/O devices is **slow**
  - (at least as far as the CPU is concerned)
  - I/O operations can take *seconds* or *milliseconds*; CPU operations take *nanoseconds*
- Every time you ask JavaScript to do an I/O operation, it *pauses your program*
  - this allows the CPU to spend time doing other things, not just sitting idle waiting for a key to be pressed or a file to be written

- Node.js has the native module to read files that allows us to read line-by-line. It was added in 2015 and is intended to read from any Readable stream one line at a time.
- This fact makes it a versatile option, suited not only for files but even command line inputs like process.stdin.
- Readline Module in Node.js allows the reading of input stream line by line. This module wraps up the process standard output and process standard input objects.
- Readline module makes it easier for input and reading the output given by the user.
- To use readline, include the following lines in the top of your source file:

```
const readline = require('readline');
```

- The readline module in NodeJS provides you with a way to read data stream from a file or ask your user for an input.

- Interaction with the user: For the interaction, we will first create an interface for the input and output. For creating an interface, write the following code:

```
var readline = require('readline’);
var rl = readline.createInterface(
process.stdin, process.stdout);
```

- Here, the **createInterface()** method takes two arguments. The first argument will be for the standard input and the second one will be for reading the standard output.

```
rl.question('What is your age? ', (age) => {
console.log('Your age is: ' + age);
});
```

- Here, **rl.question()** method is used for asking questions from the user and reading their reply (output). The first parameter is used to ask the question and the second parameter is a callback function which will take the output from the user as the parameter and return it on the console.

- Here, the problem is it will not exit the application and it will keep asking for the inputs. To resolve this issue, **rl.close()** method is used.

- We can also use a setPrompt() method is used to set the particular statement to the console. prompt() method for displaying the statement which is set in setPrompt() Method.

```
var readline = require('readline’);
var rl = readline.createInterface(
process.stdin, process.stdout);
rl.setPrompt(`What is your age? `);
rl.prompt()
```

- This code will take input from the user. Now, we need a listener for reading the input from the user and displaying it to the console.
- For this purpose, **Readline Module** has a listener method called **on** that will take two parameters.
- The first parameter will the event and the second parameter will be a callback function that will return the output to the console.

```
rl.on('line', (age) => {
console.log(`Age received by the user: ${age}`);
rl.close();
});
```

- **rl.on()** method takes the first argument as **line** event. This event is invoked whenever the user press **Enter** key.
- **Events:** Along with the listener, readline module also comes with the events properties. Let us see the various events.
  - **close:** This event is invoked when either **rl.close()** method is called or when the user presses **ctrl + c** to close the interface.
  - **line:** This event is invoked whenever the user presses **Enter** or **return** keys. This event is called in the listener function.
  - **pause:** This event is invoked when the **input** stream is paused.
  - **resume:** This event is invoked whenever the **input** is resumed.

code	**explanation**
const readline = require('readline');	load the readline package and name it readline
const readlineInterface = readline.createInterface({...})	create an interface to readline using the following settings:
process.stdin,	for input, use the standard input stream (i.e. terminal keyboard input)
process.stdout	for output, use the standard output stream (i.e. terminal console output)
function ask(questionText) {...}	a function named ask that uses the Promise API to asynchronously ask a question and wait for a reply

Callback Hell

- **What is Synchronous Javascript?**
  - In Synchronous Javascript, when we run the code, the result is returned as soon as the browser can do. Only one operation can happen at a time because it is single-threaded. So, all other processes are put on hold while an operation is executing.

- **What is Asynchronous Javascript?**
  - Some functions in Javascript requires AJAX because the response from some functions are not immediate. It takes some time and the next operation cannot start immediately. It has to wait for the function to finish in the background. In such cases, the callbacks need to be asynchronous.

- **What is a callback?**
  - Callbacks are nothing but functions that take some time to produce a result.
  - Usually these async callbacks (async short for asynchronous) are used for accessing values from databases, downloading images, reading files etc.
  - As these take time to finish, we can neither proceed to next line because it might throw an error saying unavailable nor we can pause our program.
  - So we need to store the result and call back when it is complete.

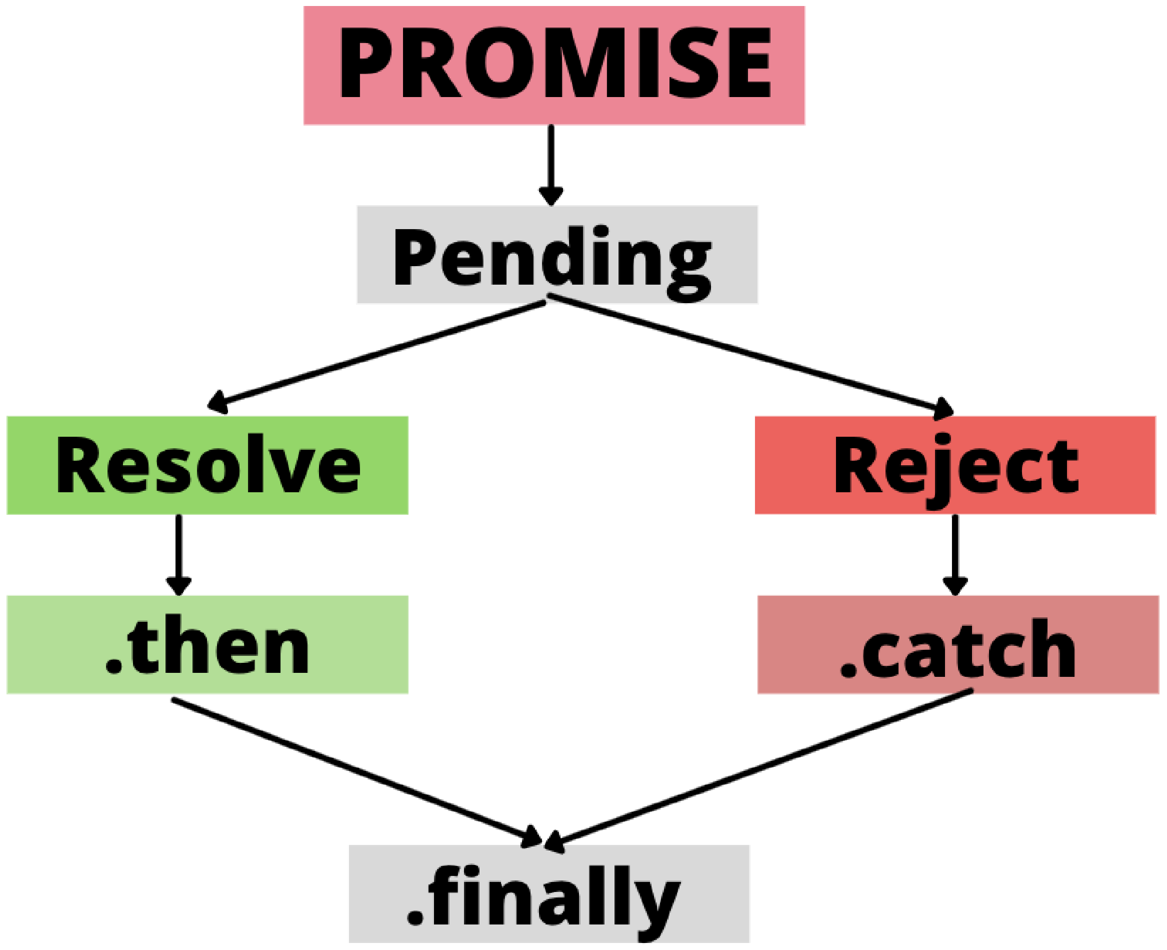
- **What is callback hell?**
  - This is a big issue caused by coding with complex nested callbacks. Here, each and every callback takes an argument that is a result of the previous callbacks. In this manner, The code structure looks like a pyramid, making it difficult to read and maintain. Also, if there is an error in one function, then all other functions get affected.

```
setTimeout(() => {console.log('Sunday’)
setTimeout(() => {console.log('Monday')
setTimeout(() => { console.log('Tuesday')
setTimeout(() => {console.log('Wednesday')
setTimeout(() => {console.log('Thursday')
setTimeout(() => {console.log('Friday')
setTimeout(() => {console.log('Saturday')
setTimeout(() => {console.log('Wow! week is over')
}, 2000)}, 2000)}, 2000)}, 2000)}, 2000)}, 2000)}, 2000)}, 2000)
```

- We can notice the nested callbacks is look like a pyramid that makes it difficult to understand.
  - It’s hard to read or poor readability.
  - Hard to understand the code.
  - It’s even harder to debug
  - Difficult to add error handling.

Promises

- “Promises” as designed to help us from “callback hell” and more importantly it helps us to better handle our code and tasks.
- Promises has few states to go through.
  - **Pending** : This is just an initial state. Example : Just consider a customer is visiting your shop and going through your menu before making an order.
  - **Resolved** : It’s like customer made an order and they got the order delivered to their hand and they are very happy.
  - **Rejected** : It’s like customer ordered something which is not available and he left the shop without getting anything.



- Lets see an example.
- We just saw a promise that can take the "**Resolved**" state when Printer was ON and it took a "**Rejected**" state when Printer was OFF. And irrespective of the state we got the **finally** block also executed.

Node.js as a File Server

- The Node.js file system module allows you to work with the file system on your computer.
- To include the File System module, use the **require()** method:

```
var fs = require('fs’);
```

- Common use for the File System module:
  - Read files
  - Create files
  - Update files
  - Delete files
  - Rename files

Reading File

- Use fs.readFile() method to read the physical file asynchronously.

  fs.readFile(fileName [,options], callback)

- Parameter Description:
  - **filename**: Full path and name of the file as a string.
  - **options:** The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".
  - **callback:** A function with two parameters err and fd. This will get called when readFile operation completes.

Opening a File

- You can open a file in your computer using the syntax below.

  fs.open(path, flags[, mode], callback)

- **path** – is the string having file name including path.
- **flags** – Flags indicate the behavior of the file to be opened. All possible values have been mentioned in the table below.
- **mode** – sets the file mode (permission and sticky bits). But only if the file was created. The defaults are 0666, readable and writeable.
- **callback** – is the callback function to call after a successful operation. It gets two arguments (err, fd).

Node.js Flags for Read/Write

Flag	Description
r	open file for reading. an exception occurs if the file does not exist.
r+	open file for reading and writing. an exception occurs if the file does not exist.
rs	open file for reading in synchronous mode.
rs+	open file for reading and writing, telling the os to open it synchronously. see notes for 'rs' about using this with caution.
w	open file for writing. the file is created (if it does not exist) or truncated (if it exists).
wx	like 'w' but fails if path exists.
w+	open file for reading and writing. the file is created (if it does not exist) or truncated (if it exists).
wx+	like 'w+' but fails if path exists.
a	open file for appending. the file is created if it does not exist.
ax	like 'a' but fails if path exists.
a+	open file for reading and appending. the file is created if it does not exist.
ax+	open file for reading and appending. the file is created if it does not exist.

Getting File Information

- Node.js allows you to retrieve the properties of a file in your system. This is done using the stat() method of fs object.

fs.stat(path, callback)

- **path** – In the path to the file
- **callback** – is the callback function to execute. It gets two arguments (err, stats) where **stats** is an object of fs.Stats type.

- Other methods available in the fs.Stats class in given below. This method is used to get additional metadata of the file

SN.	Method and brief description
1	**stats.isFile()** Returns true if file of a simple file type.
2	**stats.isDirectory()** Returns true if the specified path is type of a directory.
3	**stats.isBlockDevice()** Returns true if file type of a block device, otherwise, it returns false.
4	**stats.isCharacterDevice()** Returns true if file type of a character device, otherwise, it returns false.
5	**stats.isSymbolicLink()** Returns true if file type of a symbolic link, otherwise, it returns false.
6	**stats.isFIFO()** Returns true if file type of a FIFO, otherwise, it returns false.
7	**stats.isSocket()** Returns true if file type of asocket, otherwise, it returns false.

Creating File

- The File System module has methods for creating new files:
  - fs.appendFile()
  - fs.open()
  - fs.writeFile()

- The fs.appendFile() method appends specified content to a file. If the file does not exist, the file will be created.
- The fs.open() method takes a "flag" as the second argument, if the flag is "w" for "writing", the specified file is opened for writing. If the file does not exist, an empty file is created.
- The fs.writeFile() method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created.

Writing to a File

- You can write to a file using the syntax below:

fs.writeFile(filename, data[, options], callback)

- If the file already exists, then the method will overwrite the file.
- The parameters are explained as follows:
- **path** – is the string having the file path.
- **data** – is the String or Buffer to be written into the specified file.
- **options** – The third parameter is an object which will hold {encoding, mode, flag}. By default. encoding is utf8, mode is octal value 0666. and flag is ‘w’
- **callback** – is the callback function which gets a single parameter err that returns an error in case of any writing error.

- The **fs.writeFile()** and **fs.writeFileSync()** methods are used to write content to a File. If a file with similar name already exists, it overwrites the existing data or creates a new file and writes the content into that file.
- The **fs.open()** method can be used to create and write the content in a document. However, the fs.open() and fs.openSync() methods are used to open the specific files and document.
- Moreover, fs.open() method also allows the users to read, write and append something in the document, but uses different flags to serve different purposes.
- Apart from the fs.writeFile() method, the **fs.appendFile()** method is also used to create new files. The fs.appendFile() and fs.appendFileSync() methods are mostly used to append or add the specified content to a file. However, if the file does not exist, the new file will be created.

- fs.access(): check if the file exists and Node.js can access it with its permissions
- fs.appendFile(): append data to a file. If the file does not exist, it's created
- fs.chmod(): change the permissions of a file specified by the filename passed. Related: fs.lchmod(), fs.fchmod()
- fs.chown(): change the owner and group of a file specified by the filename passed. Related: fs.fchown(), fs.lchown()
- fs.close(): close a file descriptor
- fs.copyFile(): copies a file
- fs.createReadStream(): create a readable file stream
- fs.createWriteStream(): create a writable file stream
- fs.link(): create a new hard link to a file
- fs.mkdir(): create a new folder
- fs.mkdtemp(): create a temporary directory
- fs.open(): set the file mode
- fs.readdir(): read the contents of a directory
- fs.readFile(): read the content of a file. Related: fs.read()

- fs.readlink(): read the value of a symbolic link
- fs.realpath(): resolve relative file path pointers (., ..) to the full path
- fs.rename(): rename a file or folder
- fs.rmdir(): remove a folder
- fs.stat(): returns the status of the file identified by the filename passed. Related: fs.fstat(), fs.lstat()
- fs.symlink(): create a new symbolic link to a file
- fs.truncate(): truncate to the specified length the file identified by the filename passed. Related: fs.ftruncate()
- fs.unlink(): remove a file or a symbolic link
- fs.unwatchFile(): stop watching for changes on a file
- fs.utimes(): change the timestamp of the file identified by the filename passed. Related: fs.futimes()
- fs.watchFile(): start watching for changes on a file. Related: fs.watch()
- fs.writeFile(): write data to a file. Related: fs.write()

Deleting a file

- The File System module has already defined the methods that allow the users to delete or truncate a file. These methods are:
  - fs.unlink()
  - The fs.unlink() and its synchronous version, fs.unlinkSync(), are used to remove a file or a symbolic link.
  - fs.ftruncate()
  - fs.ftruncate() However, it doesn’t help us to delete or remove any file from the directory; instead, the fs.ftruncate() is used to modify the file’s size; it means, either we can increase the file size or decrease it.
  - The file is truncated to a specified length, if len is shorter than the current length of the file at the path. The file length can also be padded by appending the null bytes (x00) until len is reached, if it is more significant than the file’s length.

HTTP Services

- Express is the most popular Node web framework, and is the underlying library for a number of other popular Node web frameworks. It provides mechanisms to:
  - Write handlers for requests with different HTTP verbs at different URL paths (routes).
  - Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
  - Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
  - Add additional request processing "middleware" at any point within the request handling pipeline.

HTTP Methods

- REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol.
- Following four HTTP methods are commonly used in REST based architecture.
  - **GET** – This is used to provide a read only access to a resource.
  - **PUT** – This is used to create a new resource.
  - **DELETE** – This is used to remove a resource.
  - **POST** – This is used to update a existing resource or create a new resource.

RESTful Web Services

- A web service is a collection of open protocols and standards used for exchanging data between applications or systems.
- Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer.

Sr.No.	**URI**	**HTTP Method**	**POST body**	**Result**
1	listUsers	GET	empty	Show list of all the users.
2	addUser	POST	JSON String	Add details of new user.
3	deleteUser	DELETE	JSON String	Delete an existing user.
4	:id	GET	empty	Show details of a user.