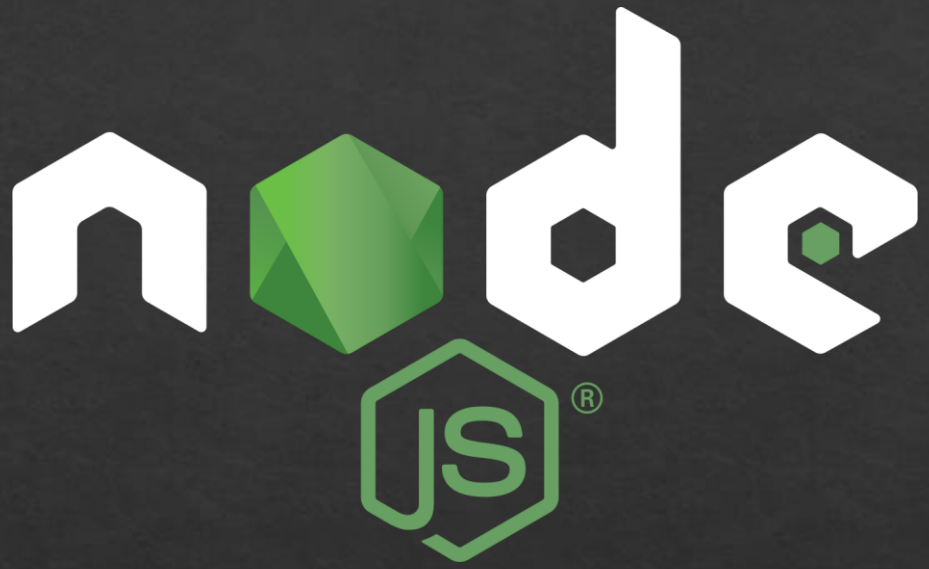
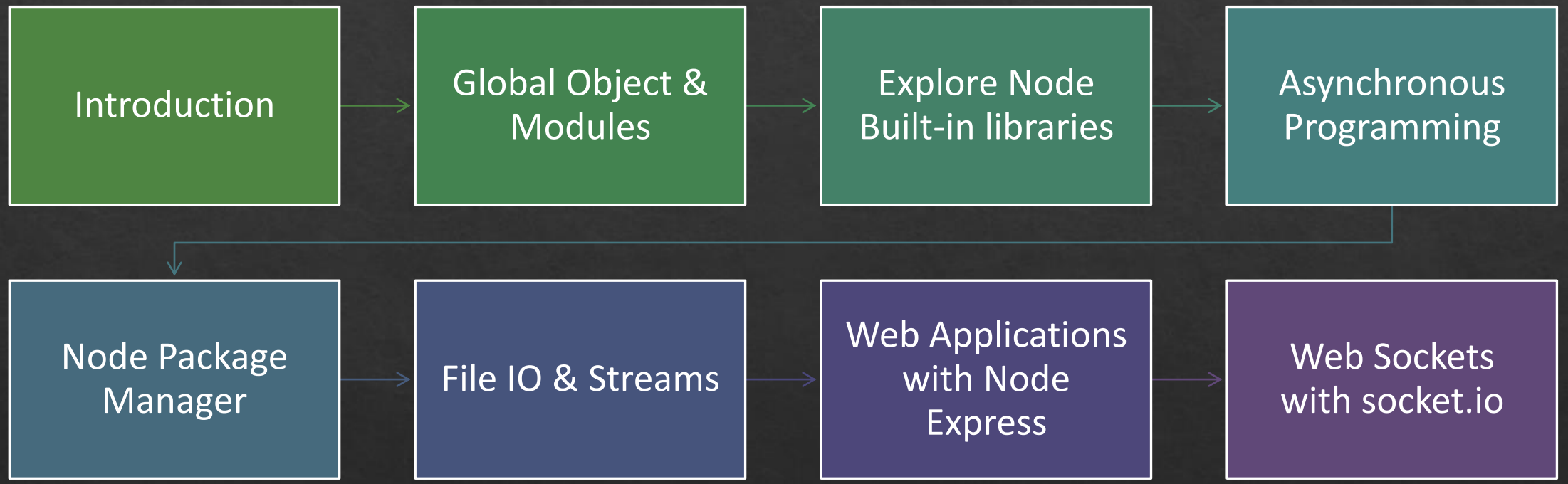


Presented By Anil Joseph(anil.jos@gmail.com)

Agenda

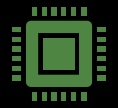


Presented By Anil Joseph(anil.jos@gmail.com)

Node.js

A server-side JavaScript platform

Created by Ryan Dahl in 2009



Provides an easy way to build scalable network applications

Built on top of Google V8 JavaScript Engine and libuv

Presented By Anil Joseph(anil.jos@gmail.com)

Node Features

Cross-Platform

Asynchronous I/O Framework

- Uses non-blocking, event-driven IO

Build fast, scalable network applications

- Capable of handling huge number of simultaneous connections with high throughput

Presented By Anil Joseph(anil.jos@gmail.com)

Node Features

Node API

- A built-in library for common functionalities

Highly Extensible

- Customize and extend Node.js as per requirements.

A community driven platform

Full stack development

- Build a complete solution using JavaScript

Presented By Anil Joseph(anil.jos@gmail.com)

Getting Started

- Install Node
  - https://nodejs.org/en/
- This install Node and NPM
- Verify by opening a command prompt and executing the command
  - node –version
- To execute a JavaScript program
  - node hello.js

Presented By Anil Joseph(anil.jos@gmail.com)

Node Command

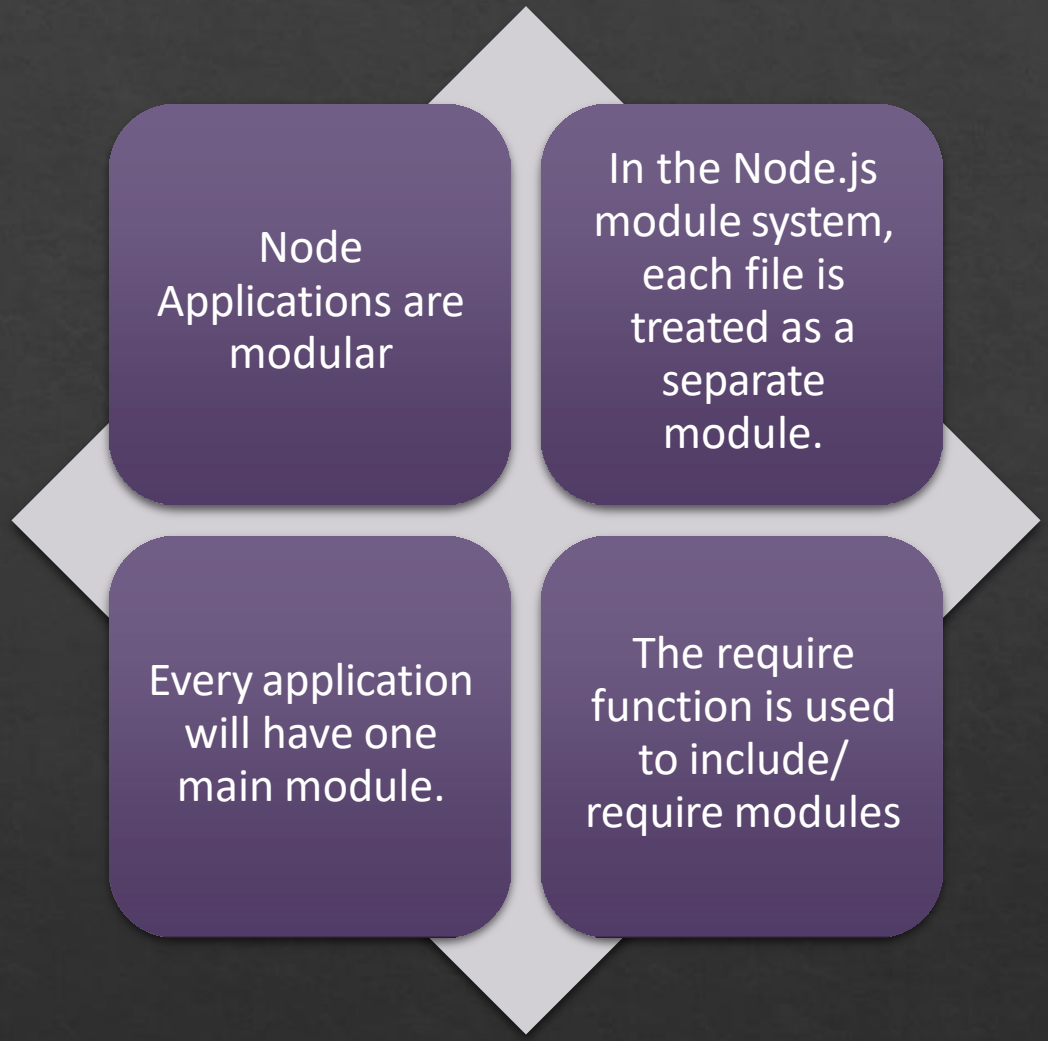
- The node command is used to start a node application.
  - Example: node app.js
- The command can be used to start the Node REPL(Read Eval Print Loop)
  - Command: node

global Object

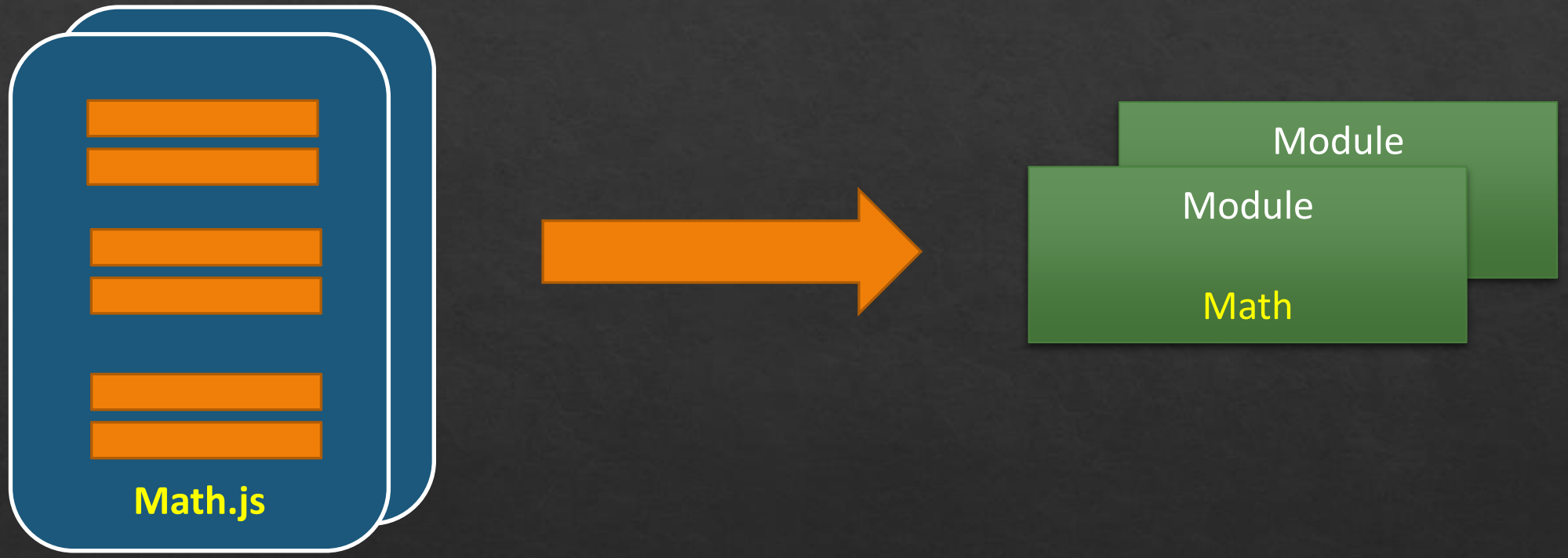
Every node application has a global object

The global object exposes objects like console, process and timer methods

Modules

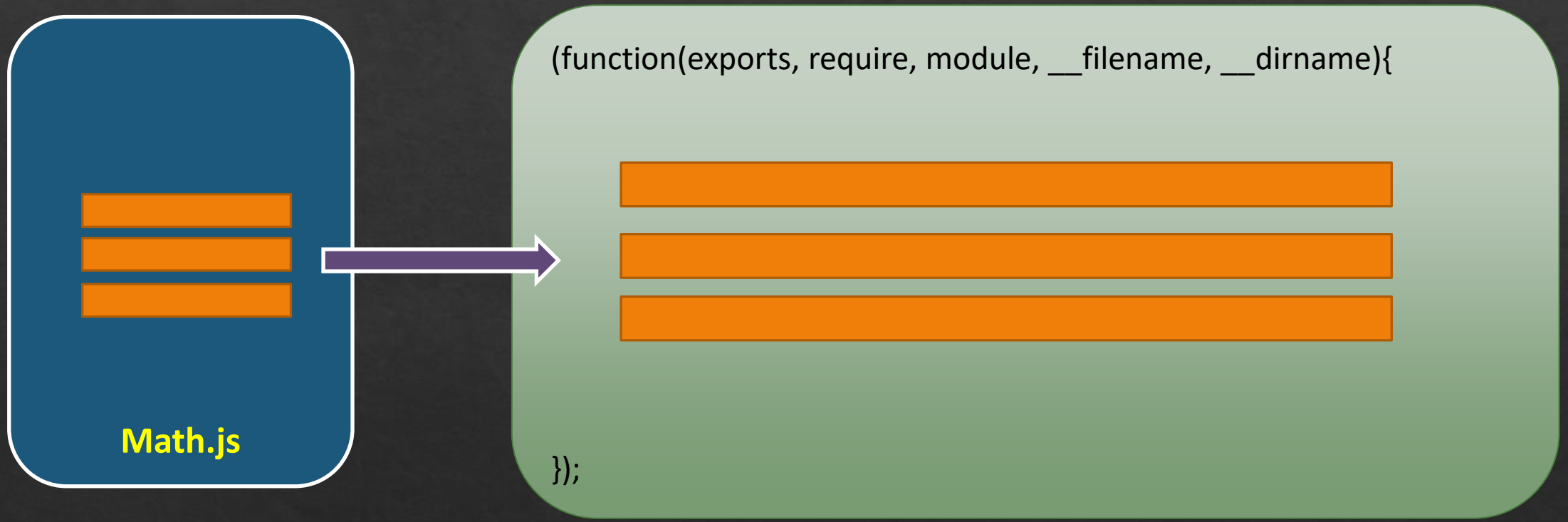

Node Applications are modular
In the Node.js module system, each file is treated as a separate module.
Every application will have one main module.
The require function is used to include/ require modules

Module



Presented By Anil Joseph(anil.jos@gmail.com)

Module Wrapper



Presented By Anil Joseph(anil.jos@gmail.com)

Sources of Modules

Built-in modules

- Comes pre-packaged with Node
- Examples fs, http, crypto, os

The project files

- Each .js file is a module
- JavaScript files can export variables, functions and objects.
- JavaScript files can be imported by providing the path.

Module folder

- A folder with a package.json file containing a main field.
- A folder with an index.js file in it.

External Modules

- Install using NPM or yarn
- Installed into node-modules folder

require

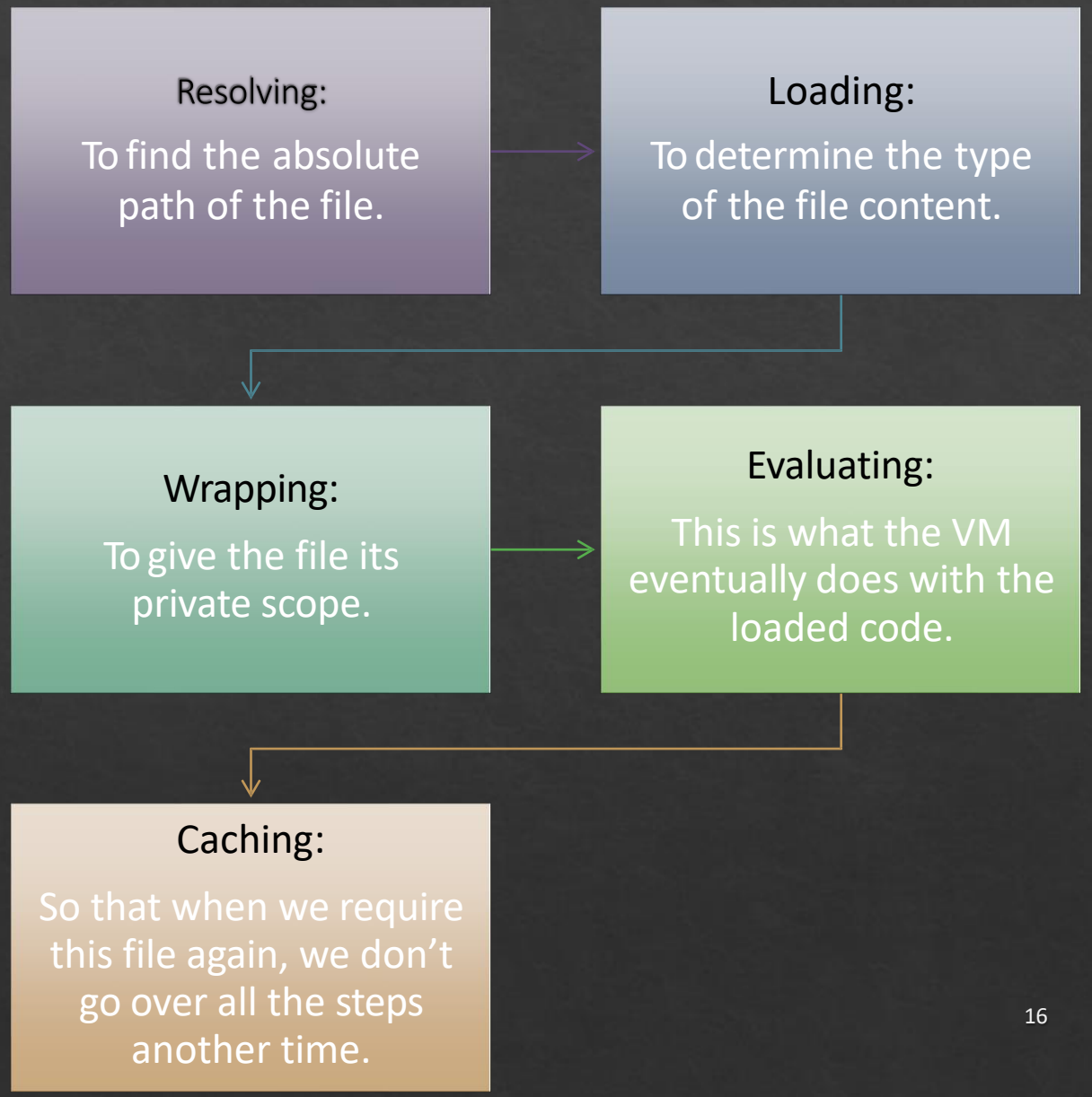
Node.js follows the CommonJS module system,

The built-in **require** function is used to include modules in separate files.

The basic functionality of require is

- **Reads a JavaScript file**
- **Executes the file**
- **Return the exports object.**

require steps



Asynchronous Programming

Callbacks

- A function invoked to notify the result of the async call

Promises

- Promise represents the result of asynchronous function.
- Promises can be used to avoid chaining of callbacks.

Events

- Node.js provides **EventEmitter** module that can help you to program using events.

Generators

- Generators provides you the ability to convert asynchronous code to synchronous one.

Presented By Anil Joseph(anil.jos@gmail.com)

Timeout Methods

- setTimeout
- setImmediate
- Process.nextTick
- setInterval

Presented By Anil Joseph(anil.jos@gmail.com)

Event Loop

Node is single threaded, and event driven

Event-driven programming is when the application flow is driven by events and changes in state

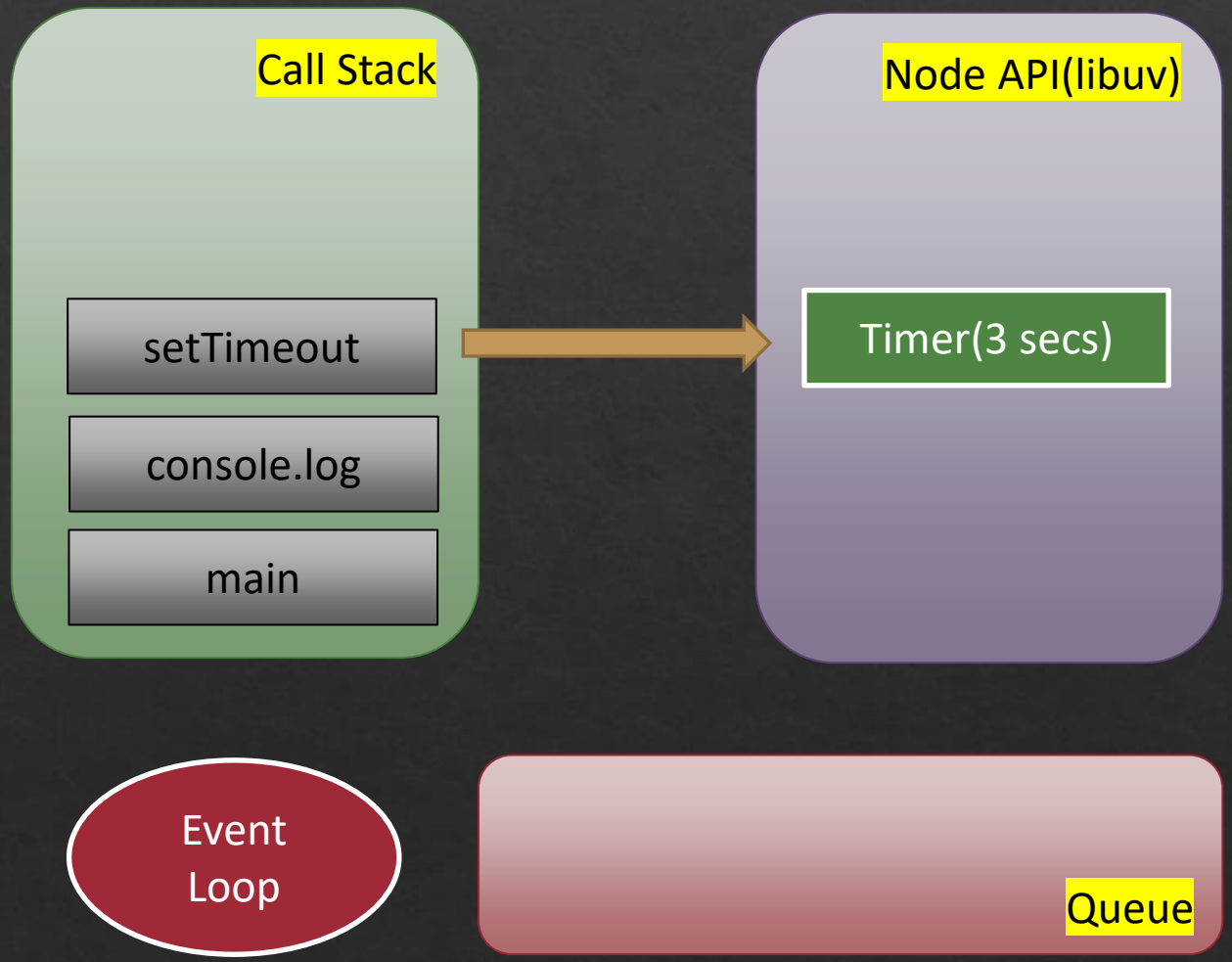
Node runs a central event loop which

- Listens for events
- Invokes callback functions

Presented By Anil Joseph(anil.jos@gmail.com)

Asynchronous Execution

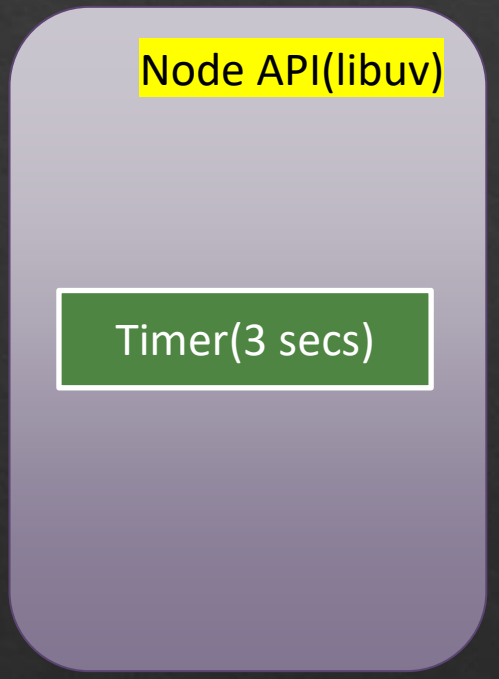
```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```



Presented By Anil Joseph(anil.jos@gmail.com)

Asynchronous Execution

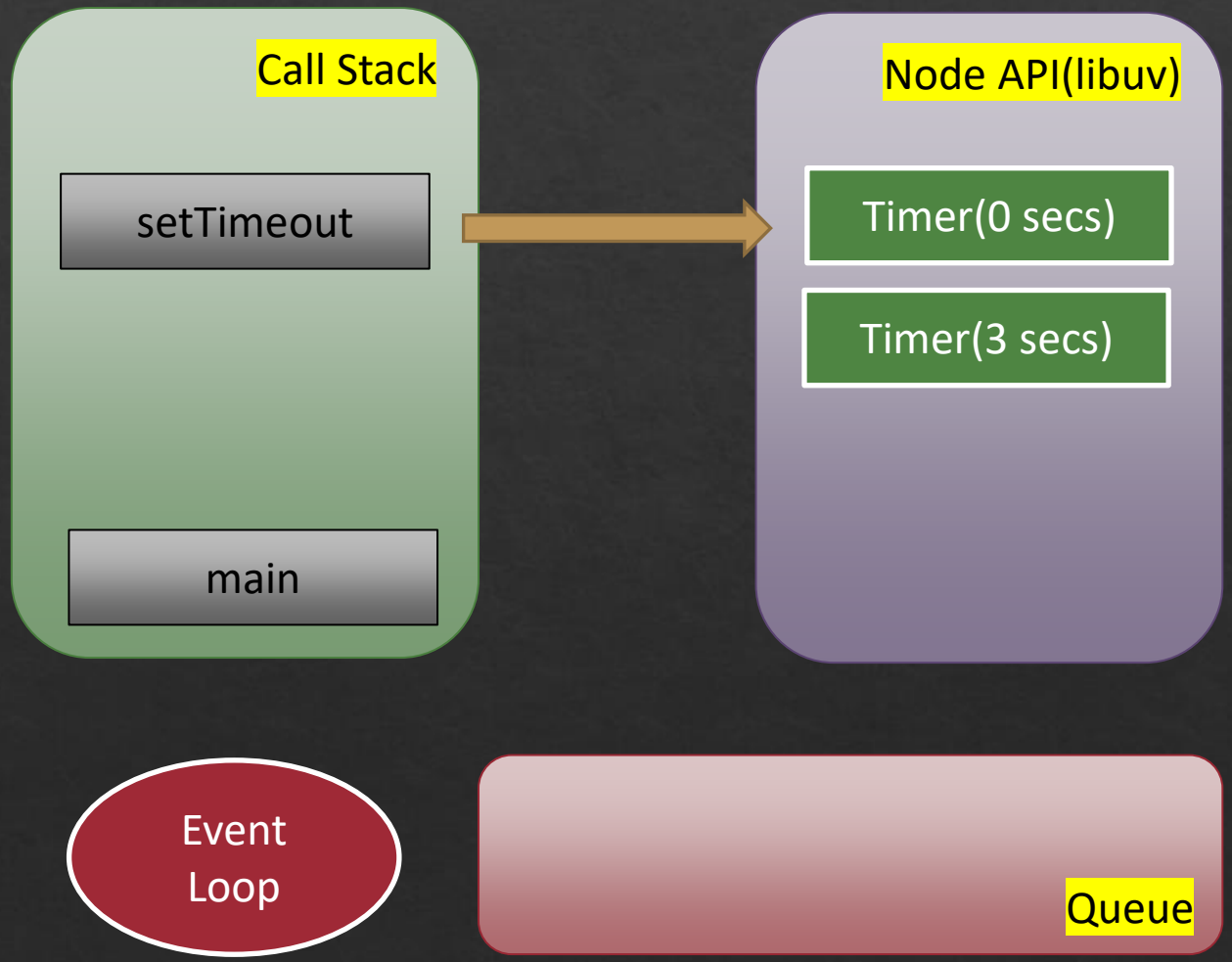
```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```



Presented By Anil Joseph(anil.jos@gmail.com)

Asynchronous Execution

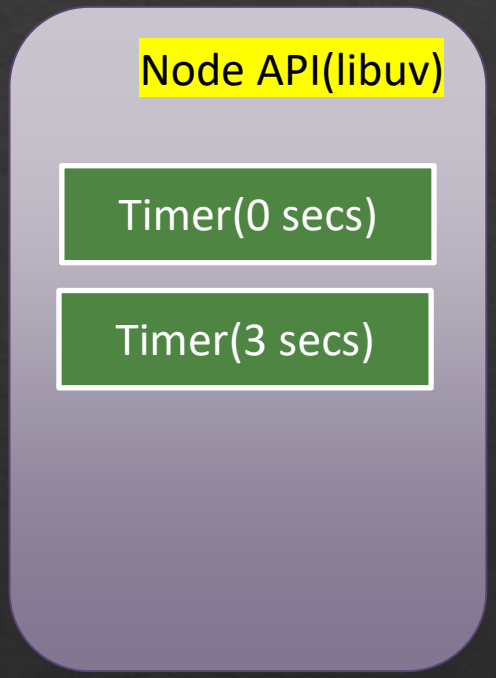
```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```

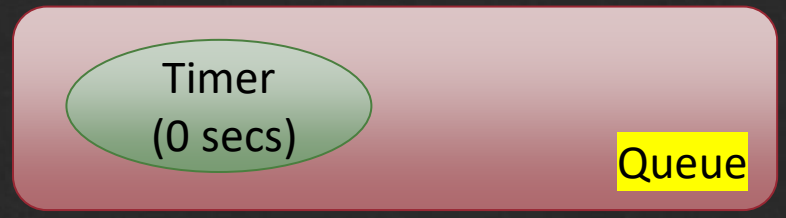


Presented By Anil Joseph(anil.jos@gmail.com)

Asynchronous Execution

```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```

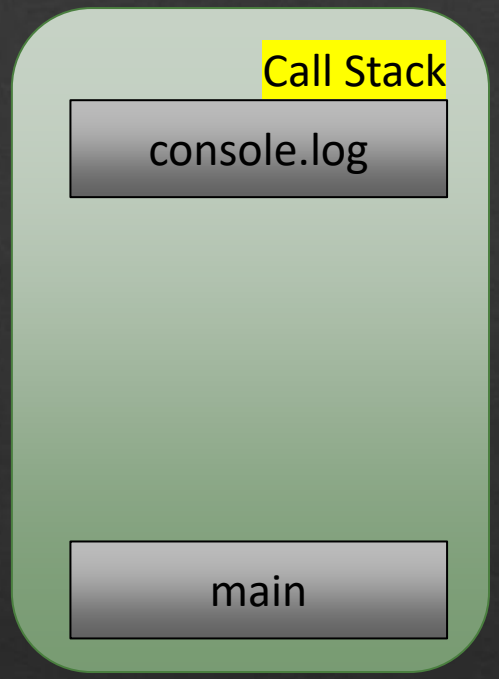


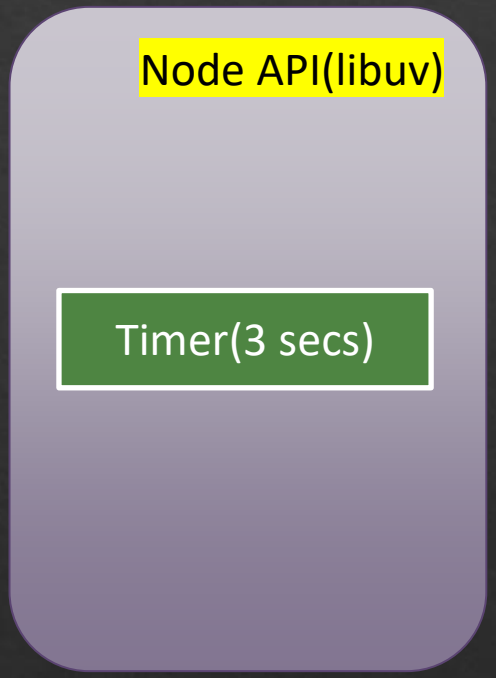


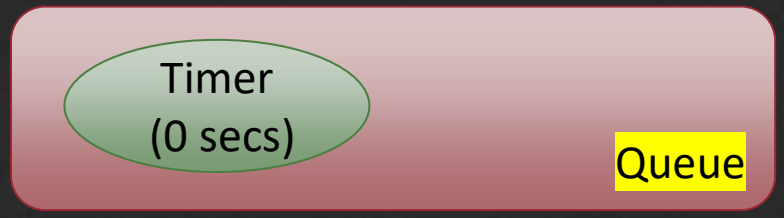
Presented By Anil Joseph(anil.jos@gmail.com)

Asynchronous Execution

```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```



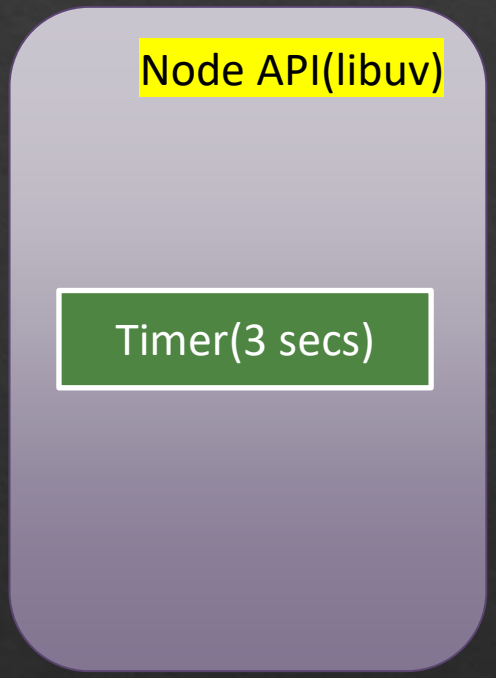


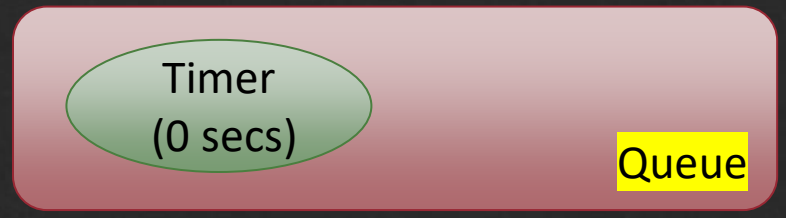


Presented By Anil Joseph(anil.jos@gmail.com)

Asynchronous Execution

```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```

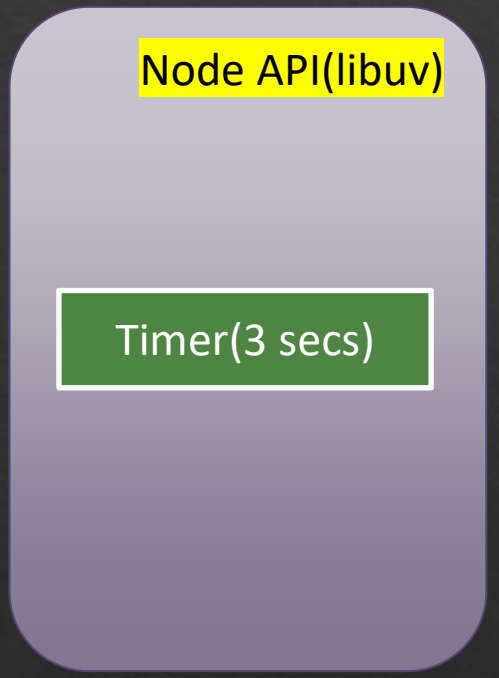




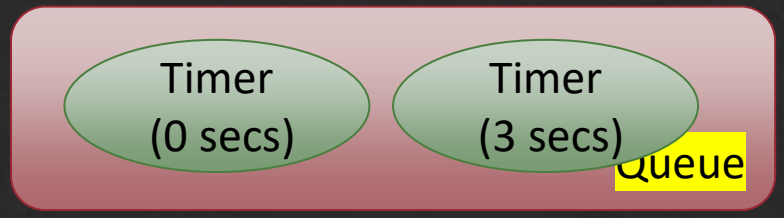
Presented By Anil Joseph(anil.jos@gmail.com)

Asynchronous Execution

```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```

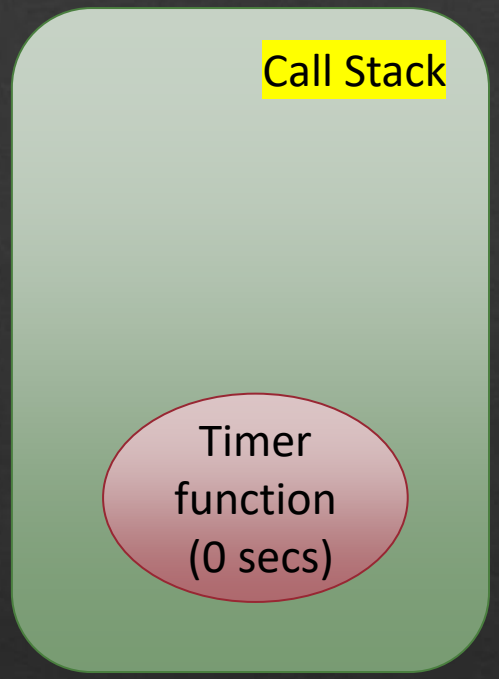


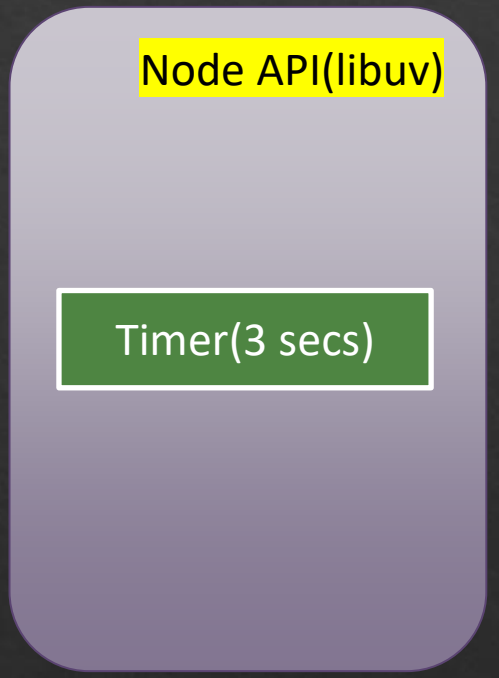
Presented By Anil Joseph(anil.jos@gmail.com)

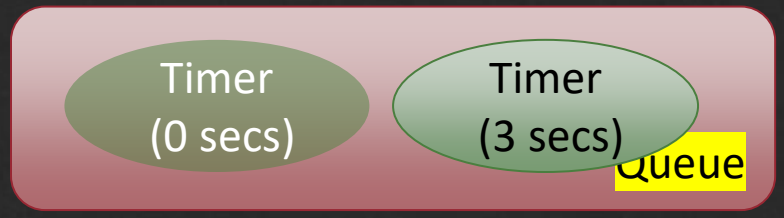


Asynchronous Execution

```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```



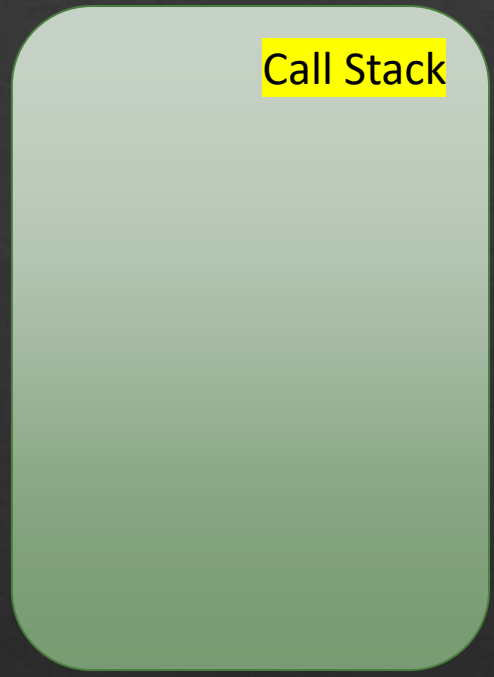


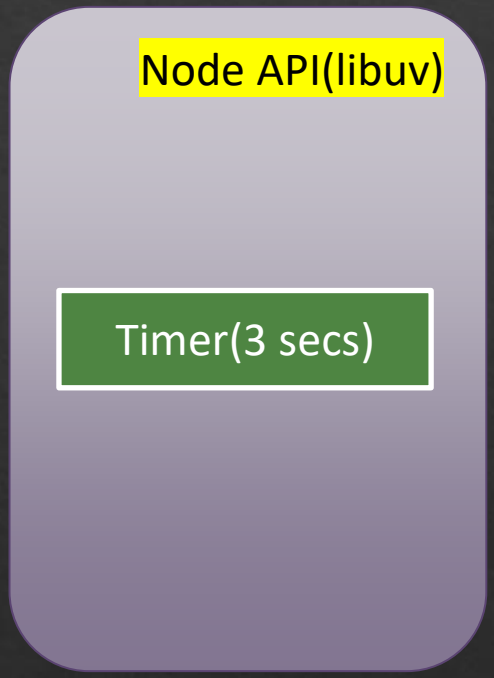


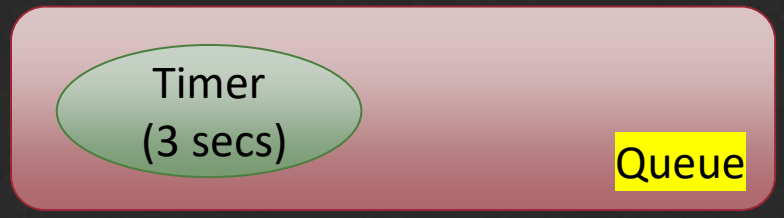
Presented By Anil Joseph(anil.jos@gmail.com)

Asynchronous Execution

```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```



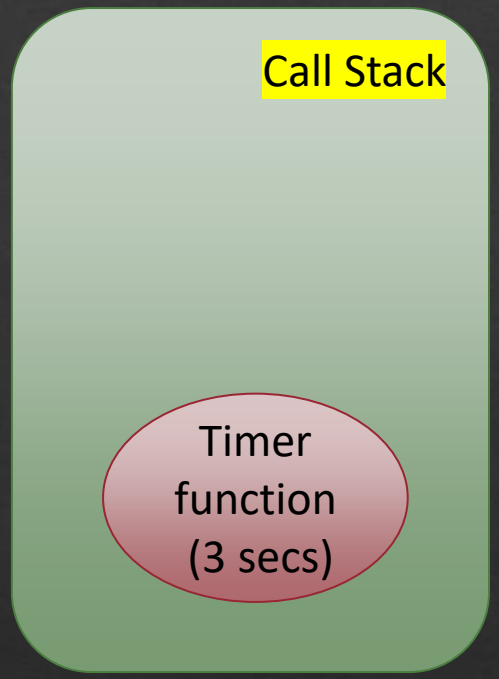


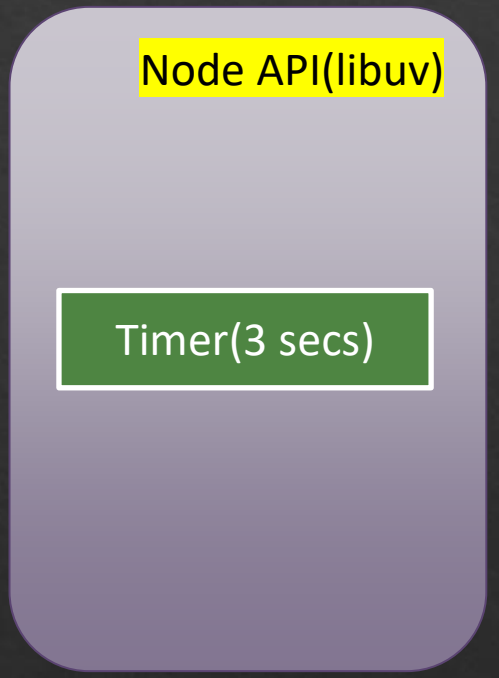


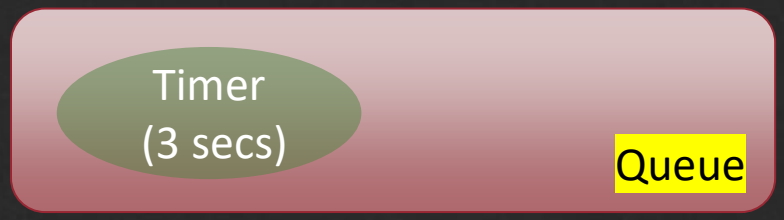
Presented By Anil Joseph(anil.jos@gmail.com)

Asynchronous Execution

```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```



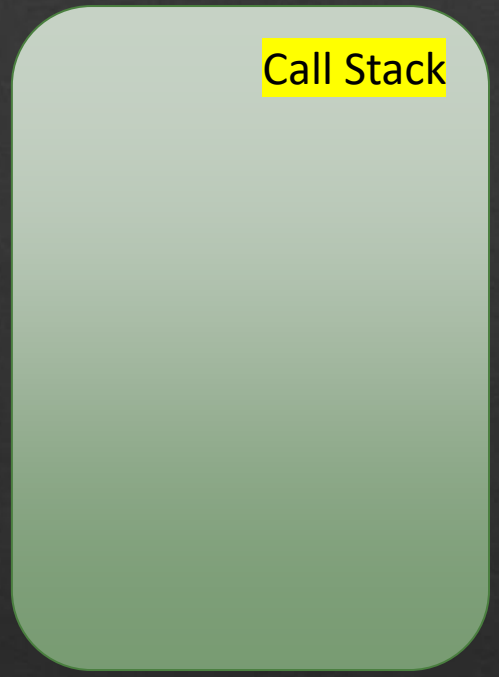


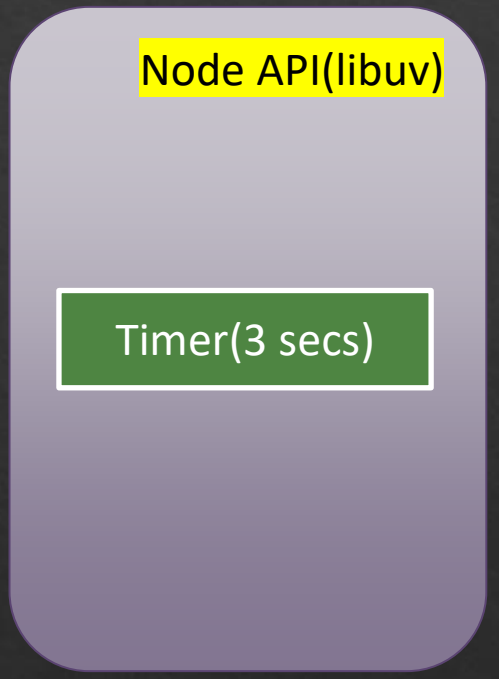


Presented By Anil Joseph(anil.jos@gmail.com)

Asynchronous Execution

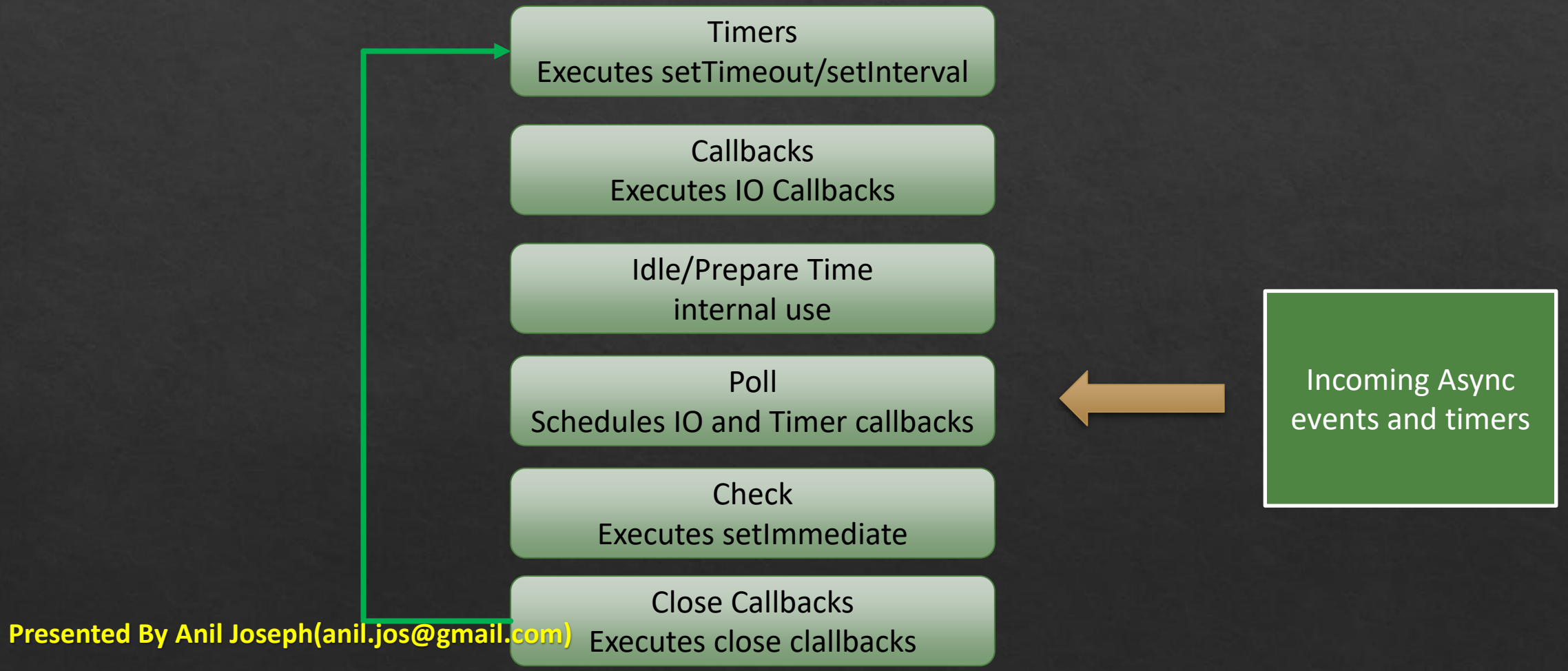
```
console.log("App started");

setTimeout(function(){
    console.log("timeout after 3 sec")
}, 3000);

setTimeout(function(){
    console.log("timeout after 0 sec")
}, 0);

console.log("App Completed");
```





Presented By Anil Joseph(anil.jos@gmail.com)

Event Loop



Presented By Anil Joseph(anil.jos@gmail.com)

Events

- Node.js core API is built around an idiomatic asynchronous event-driven architecture
- Events work with emitters that emit the event and listeners(function objects) called to process the event.
- All objects that emit events are instances of the EventEmitter class.
- These objects expose an eventEmitter.on() function that allows one or more functions to be attached to named events emitted by the object.

Presented By Anil Joseph(anil.jos@gmail.com)

Streams

Readable stream

- An abstraction for a source from which data can be consumed

Writable stream

- An abstraction for a destination to which data can be written

Duplex streams

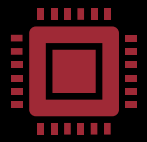
- Both Readable and Writable

Transform stream

- A duplex stream that can be used to modify or transform the data as it is written and read.

All streams are instances of EventEmitter.

Node: Options to perform CPU intensive applications



child_process

cluster

worker_threads

Presented By Anil Joseph(anil.jos@gmail.com)

Workers

- The worker_threads module enables the use of threads that execute JavaScript in parallel.
- Workers (threads) are useful for performing CPU-intensive JavaScript operations.
- They will not help much with I/O-intensive work.
  - Node.js's built-in asynchronous I/O operations are more efficient than Workers can be.
- Workers can share memory
  - Transfer ArrayBuffer instances
  - Sharing SharedArrayBuffer instances.

Presented By Anil Joseph(anil.jos@gmail.com)

Workers

- Standard Process
  - One Process
  - One thread
  - One event loop
  - One V8 Engine instance

- Process with Worker Thread
  - One Process
  - Multiple threads
  - One event loop per thread
  - One V8 Engine instance per thread

Presented By Anil Joseph(anil.jos@gmail.com)

Child Process

- Node.js is single-threaded by design.
- The child_process module allows us to spawn processes that can be used for any task to be run in parallel.
- Child_process is a built-in module

Child Process Methods

spawn

- Initiates a new process in the background asynchronously without blocking the main thread.
- Designed to run **system commands**

fork

- Built on top of spawn() will spin up a fresh V8 instance to be used to run any Node-based code
- Useful to execute CPU intensive tasks in parallel or to scale an application.

exec

- Similar to spawn but returns a buffer while spawn return a data stream.

Scaling

Cloning

Decomposing

Spilitting

HTTP Server

Node.js provides an HTTP module to transfer data using the Http protocol.

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

Usage **var http = require('http');**

Node Express

Web Application Framework for Node.js

Unopinionated Framework

Provides a Robust set of features for Web and Mobile Applications

Provides a myriad of HTTP utility methods and middleware to create REST API's

Many popular frameworks are based on Express.

Presented By Anil Joseph(anil.jos@gmail.com)

Node Express

- Install
  - npm install express
- Usage
  - const express = require('express')
  - const app = express()
  - app.listen(5000, () => console.log(`Example  app listening at http://localhost5000`))

Presented By Anil Joseph(anil.jos@gmail.com)

Routing

- Routing refers to determining how an application responds to a client request to a endpoint.
- app.METHOD(PATH, HANDLER)
  - app is an instance of express.
  - METHOD is an HTTP request method, in lowercase.
  - PATH is a path on the server.
  - HANDLER is the function executed when the route is matched.
- Example

```
app.get('/', function (req, res) {
  res.send('Hello World!')
})
```

Presented By Anil Joseph(anil.jos@gmail.com)

Express Templating Engines

Template engine helps us to create an HTML template with minimal code.

EJS : Embedded JavaScript templates.

Handlebars.js

Pug (also known as jade)

Mustache

Presented By Anil Joseph(anil.jos@gmail.com)

What is REST?

- Representational State Transfer (REST) is a style of architecture.
  - Describes how networked resources are defined and addressed.
- These principles were first described in 2000 by Roy Fielding
- REST has proved to be a popular choice for implementing Web Services.

Principles of REST

Client-Server

Cacheable

Stateless

Uniform Interface

Layered

Code on demand

Principles of REST

Client–server

- The client and the server both have a different set of concerns.
- The server stores and/or manipulates information and makes it available to the user in an efficient manner.
- The client takes that information and displays it to the user and/or uses it to perform subsequent requests for information.

Stateless

- A communication between the client and the server always contains all the information needed to perform the request.
- There is no session state in the server, it is kept entirely on the client's side.

Cacheable

- The client, the server and any intermediary components can all cache resources in order to improve performance.

Principles of REST

Uniform Interface

- Provides a **uniform interface** between components.
- Requests from different clients look the same

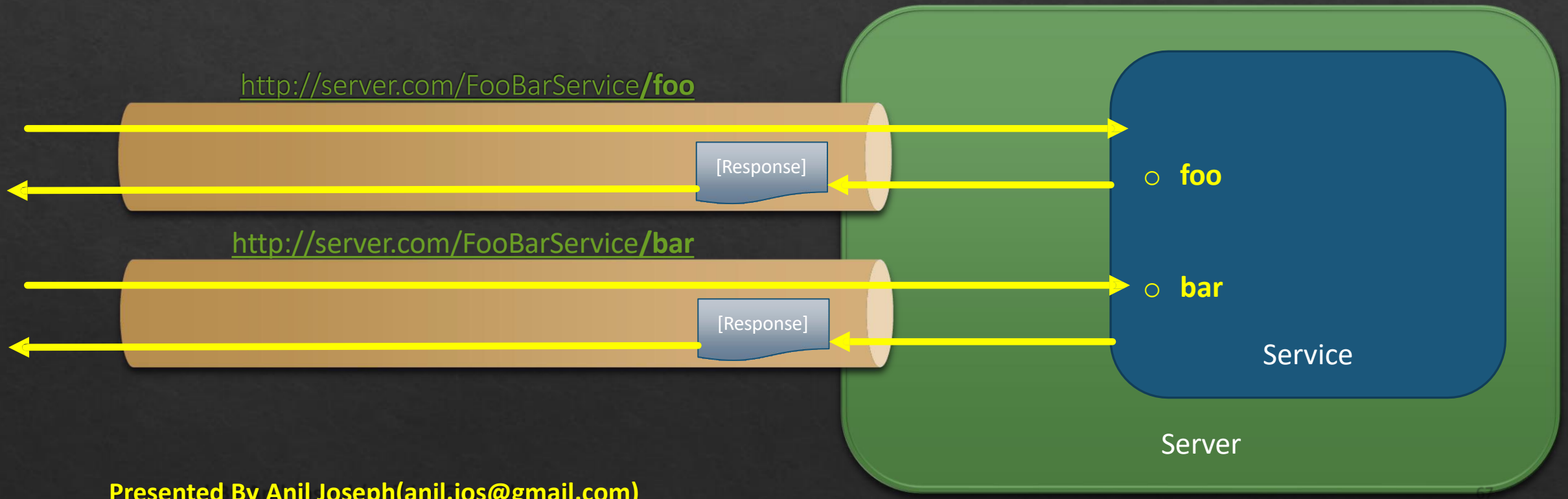
Layered System

- Between the client requests and the server response there can be a number of components/servers in the middle.
- The layers can provide security, caching, load-balancing or other functionality.
- The client is agnostic as to how many layers.

Code on demand

- This constraint is optional
- The client can request code from the server, and then the response from the server will contain some code

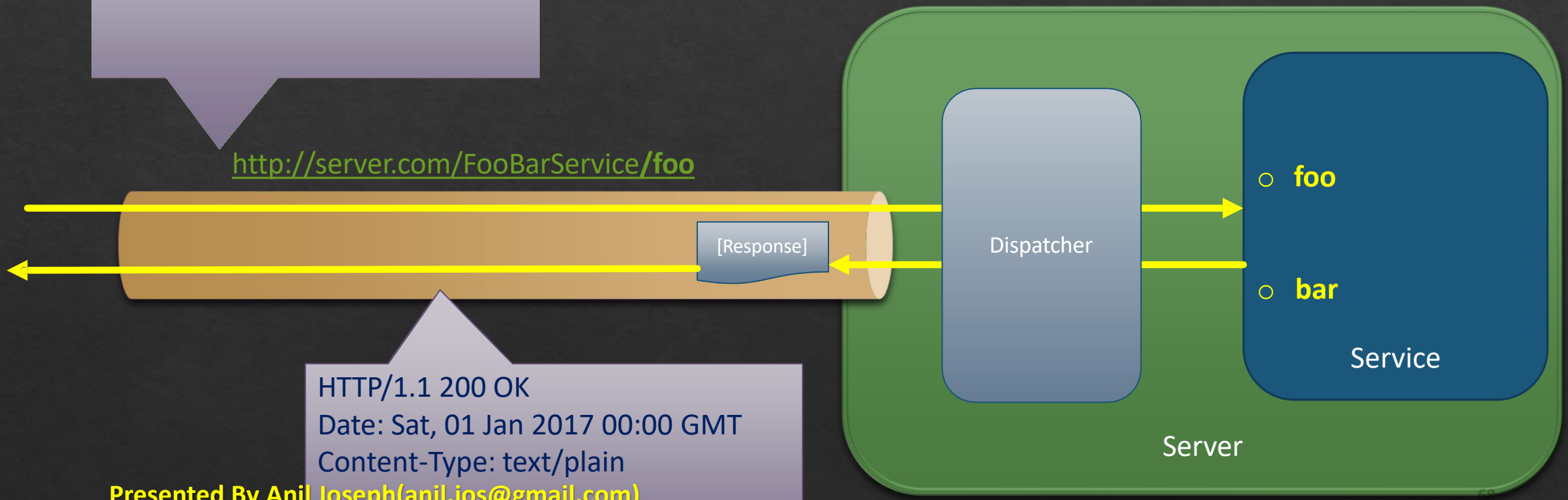
REST Service



Presented By Anil Joseph(anil.jos@gmail.com)

REST Service

HTTP/1.1 200 OK
Date: Sat, 01 Jan 2017 00:00 GMT
Content-Type: text/plain



HTTP/1.1 200 OK
Date: Sat, 01 Jan 2017 00:00 GMT
Content-Type: text/plain

{“name”: “anil joseph”}

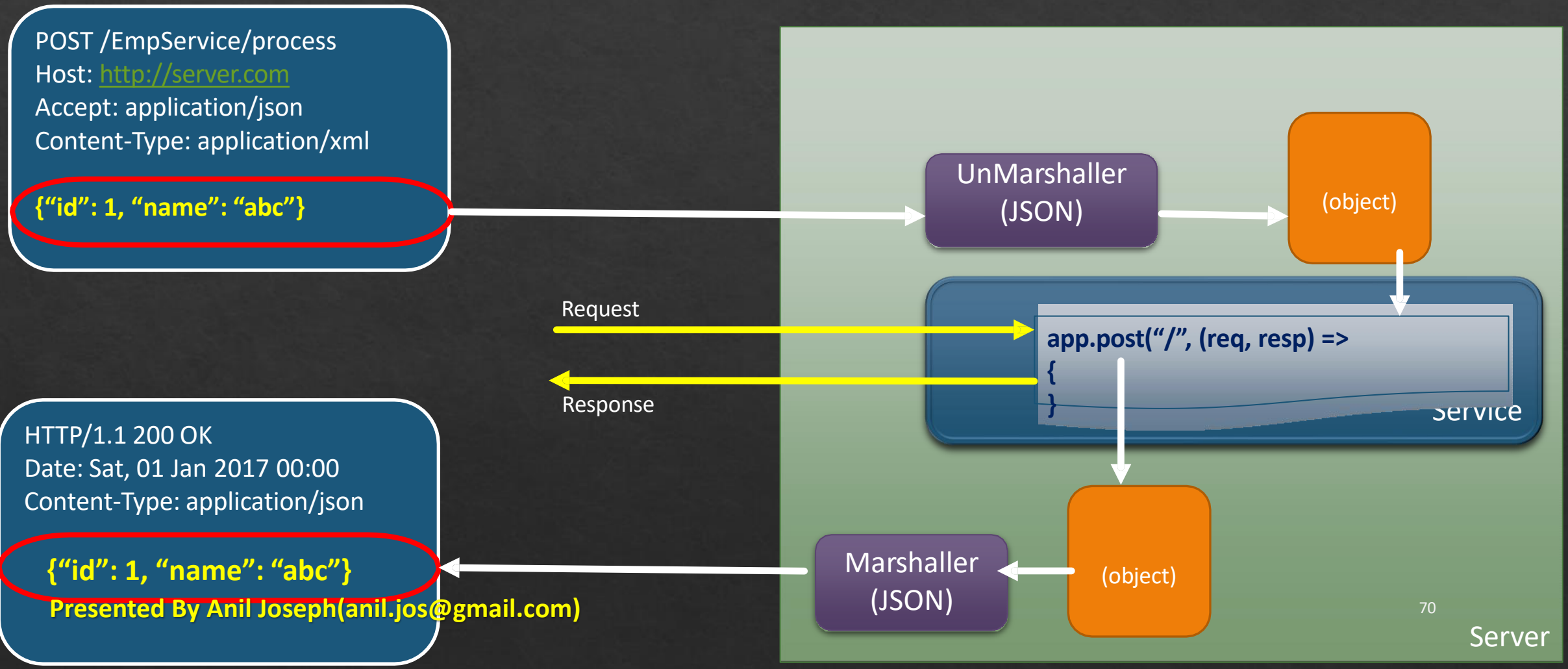
Presented By Anil Joseph(anil.jos@gmail.com)

Http Methods

Get	To retrieve or read a resource.
Post	To create a new resource.
Delete	To delete/remove an existing resource.
Put	To update an existing resource.
Patch	Used to update a slice of the resource

Presented By Anil Joseph(anil.jos@gmail.com)

JSON Marshalling & Unmarshalling



Cluster Module

- “Cluster” was introduced to scale an application execution on multiple processor cores by creating worker processes.
- Worker processes share a single port, therefore, requests are routed through a single port.
- Each worker process is called a Node.

Presented By Anil Joseph(anil.jos@gmail.com)

Cluster Module

Pros

- Spawned process that terminate can be started instantly
- Utilize all available cores for application execution, increases application performance.
- Easy to implement since all work is managed by NodeJs module.

Cons

- Session management is not available
- IPC is a tedious job to manage an application,

Presented By Anil Joseph(anil.jos@gmail.com)

Socket.io

- Socket.IO enables real-time, bidirectional and event-based communication.
- Cross-platform(on both browsers and devices)
- Based on Web Socket protocol
- Applications
  - Binary Streaming
  - Real-time analytics
  - Messaging and Chat

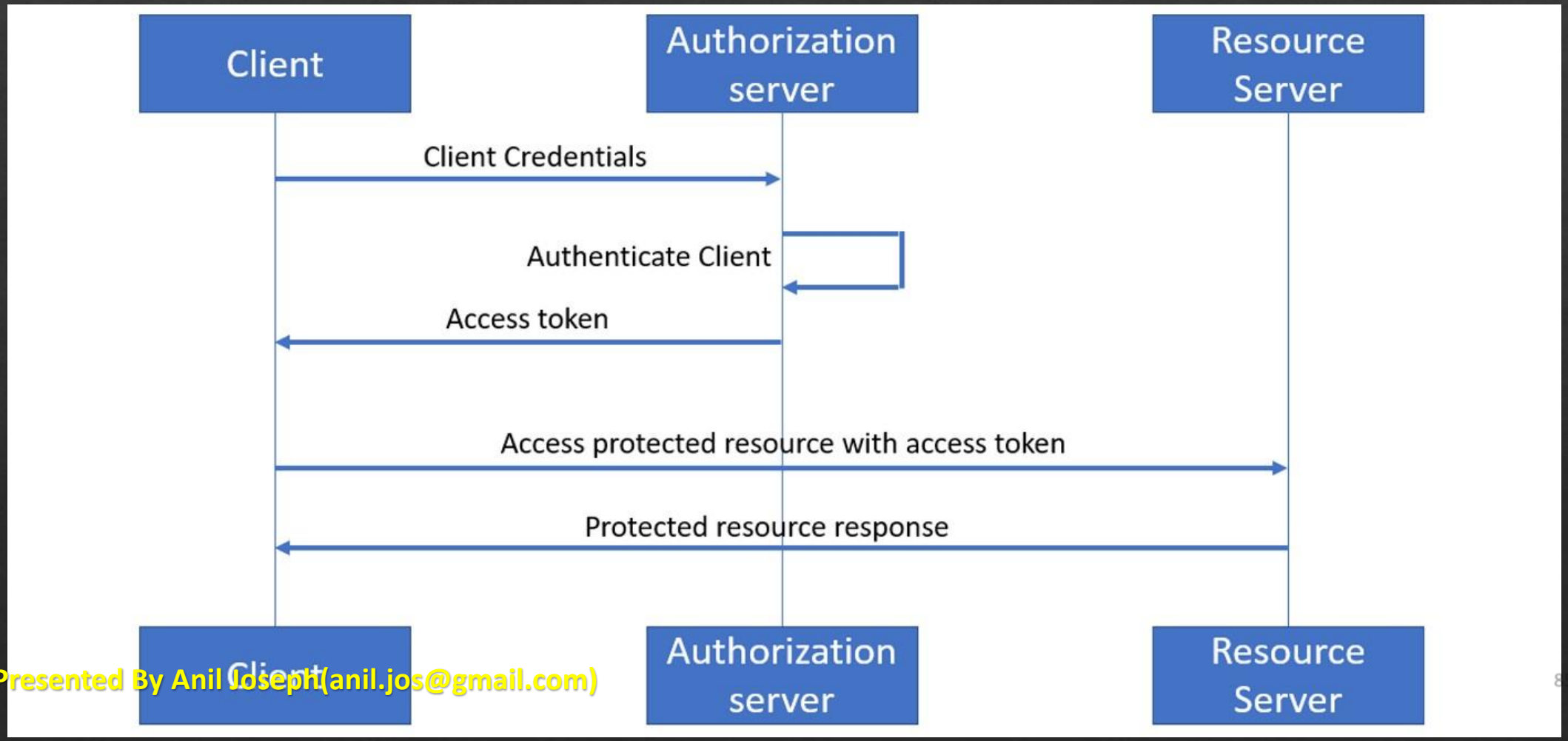
Presented By Anil Joseph(anil.jos@gmail.com)

Authentication & Authorization(REST)

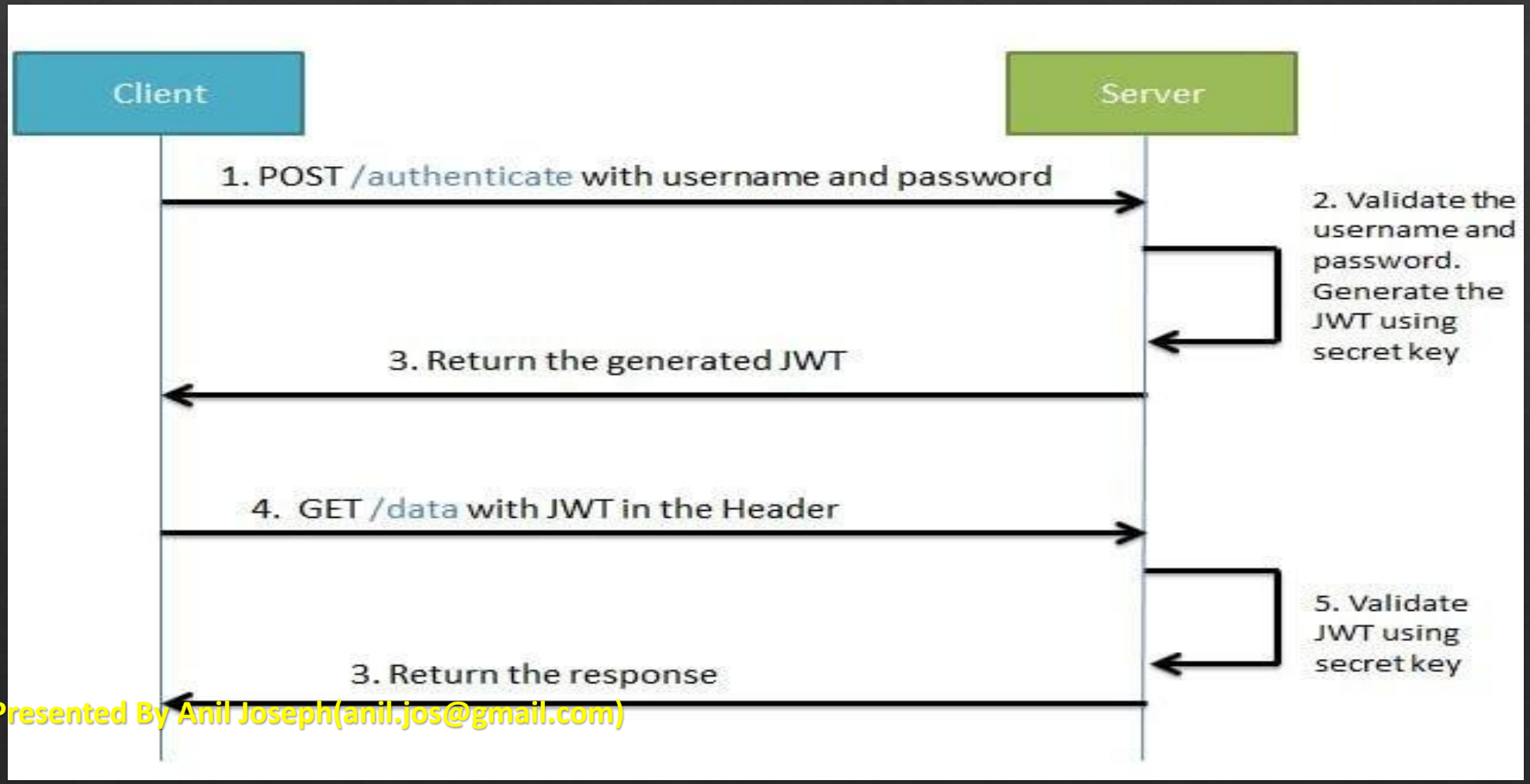
- Json Web Tokens
- oAuth2
- Api Keys

Presented By Anil Joseph(anil.jos@gmail.com)

oAuth2


Client
Authorization server
Resource Server
Client Credentials
Authenticate Client
Access token
Access protected resource with access token
Protected resource response
Client
Authorization server
Resource Server

JWT


Client
Server
1. POST /authenticate with username and password
2. Validate the username and password. Generate the JWT using secret key
3. Return the generated JWT
4. GET /data with JWT in the Header
5. Validate JWT using secret key
3. Return the response

NPM(Node Package Manager)

- NPM is a package manager for the JavaScript programming language.
- Allows users to consume and distribute JavaScript modules that are available on the registry
- The default package manager for Node.
- Comprises of
  - Command line client, also called npm
  - An online database called the npm registry.
    - Public
    - Paid-for private packages
  - NPM website
- The package manager and the registry are managed by npm, Inc.

NPM Packages

- Packages are reusable code published in the npm registry.
- A directory with one or more files a metadata file called package.json.
- A package is a building block that solves a specific problem.
- An application generally depends on many packages.
- Packages can be used on
  - Server side. Example: Express, Request
  - Client Side. Example Angular, React
  - Command based. Typescript, Angular CLI, JSLint

Modules revisited

- A module is anything that can be loaded with require() in a Node.js program.
- Examples
  - A folder with a package.json file containing a main field.
  - A folder with an index.js file in it.
  - A JavaScript file.
- Most npm packages are modules

NPM Commands

- NPM is installed with Node
- Check the version
  - npm –v
- Update npm
  - npm install npm@latest –g
- Find the path to npm's directory:
  - npm config get prefix
- Help
  - npm –h
  - npm [command] –h
  - npm help [command]
  - npm help-search [key]

package.json

- A metadata file for the project
- Used to track dependencies
- Create Scripts
- Command to create package.json
  - npm init
- Setting defaults
  - npm set init-author-name 'Anil Joseph'
  - npm get init-author-name
  - npm config delete init-author-name

Installing Packages

- Install to a project
  - npm install express
- Install and save to dependencies
  - npm install angular --save
- Install and save to development dependencies
  - npm install express --save –dev
- Install globally
  - npm install gulp -g

Install Packages with version

- Specific version
  - npm install underscore@1.8.2
- Latest Version
  - npm install underscore@1.8.x
  - npm install underscore@1.8
  - npm install underscore@1.8.2
  - npm install underscore@1.x
  - npm install underscore@1
  - npm install underscore
- Other Options
  - npm install underscore@”>=1.4.x < 1.6.x”

Listing & Removing Package

- npm list
  - List all packages
  - Shows all dependencies of packages
- npm list --depth 1
  - List packages show dependency depth to 1 level
- npm list --global true
  - List global packages
- npm list --global true --depth 0
  - List global packages show dependency depth to 0 level

Listing & Removing Package

- npm list --dev true
  - List development dependiecies
- npm list --prod true
  - List Production dependencies
- npm ls
  - ls as shortcut
- npm uninstall underscore/ npm rm underscore/ npm un underscore
  - Removes/Uninstall a package
- npm uninstall underscore --save
  - Uninstalls and updates the package.json
- npm uninstall underscore -g
  - Unstall global package

Anil Joseph

anil.jos@gmail.com

WhatsApp : +91 98331 69784

Thank You