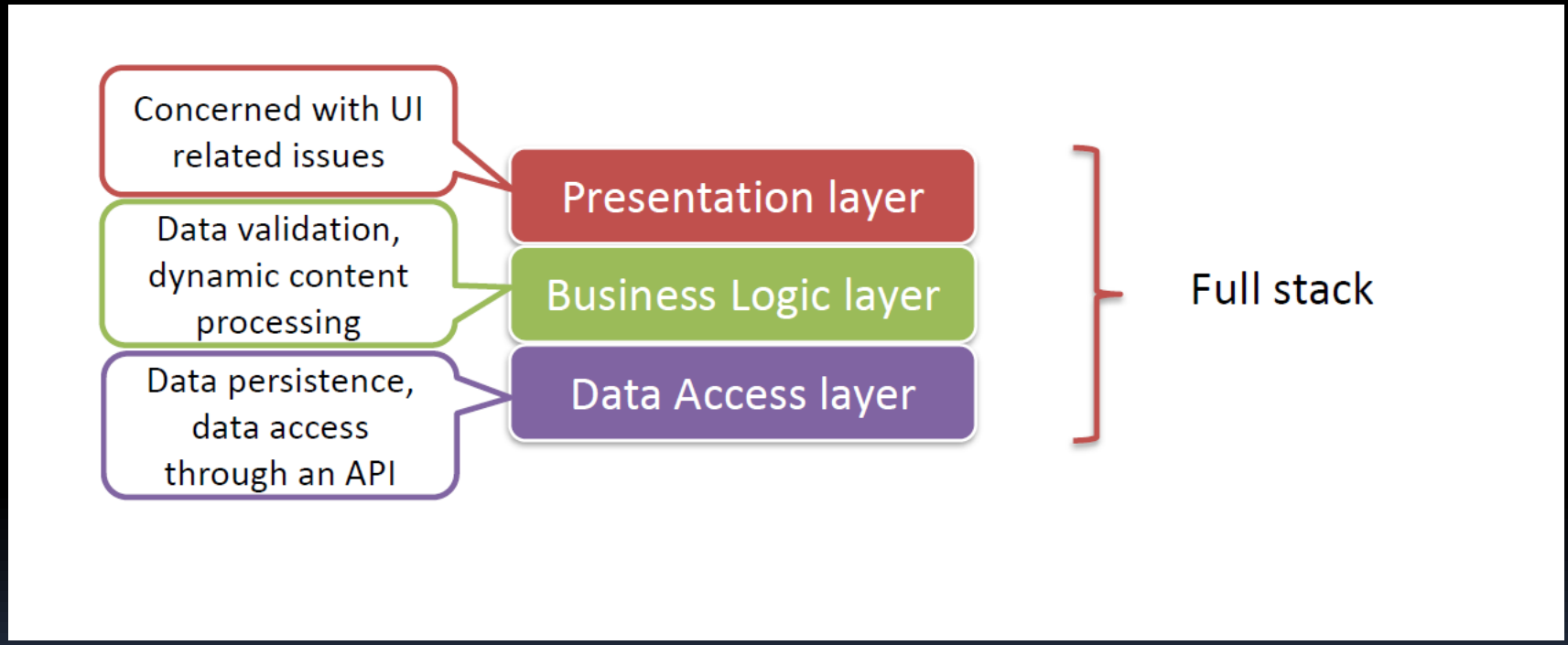
Node.js

Syllabus:

Getting started with Node.js:

- 3.1 Introduction to Node.js
- 3.2 REPL
- 3.3 Node modules: events, OS, HTTP, buffers, streams.
- 3.4 Building own API and consuming it (RESTful API).

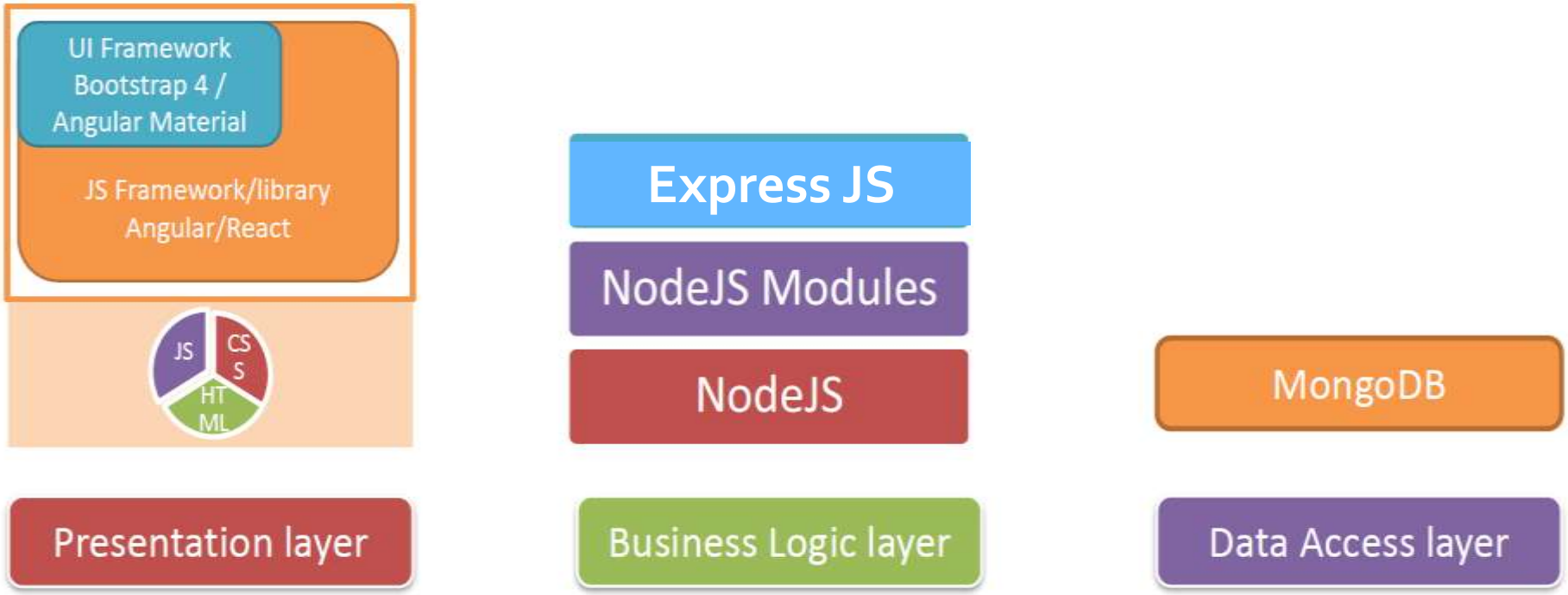
Three Tier Architecture



- Using a **Single language** to implement the entire stack (3 layers)

"JAVASCRIPT"

Full Stack Web Development



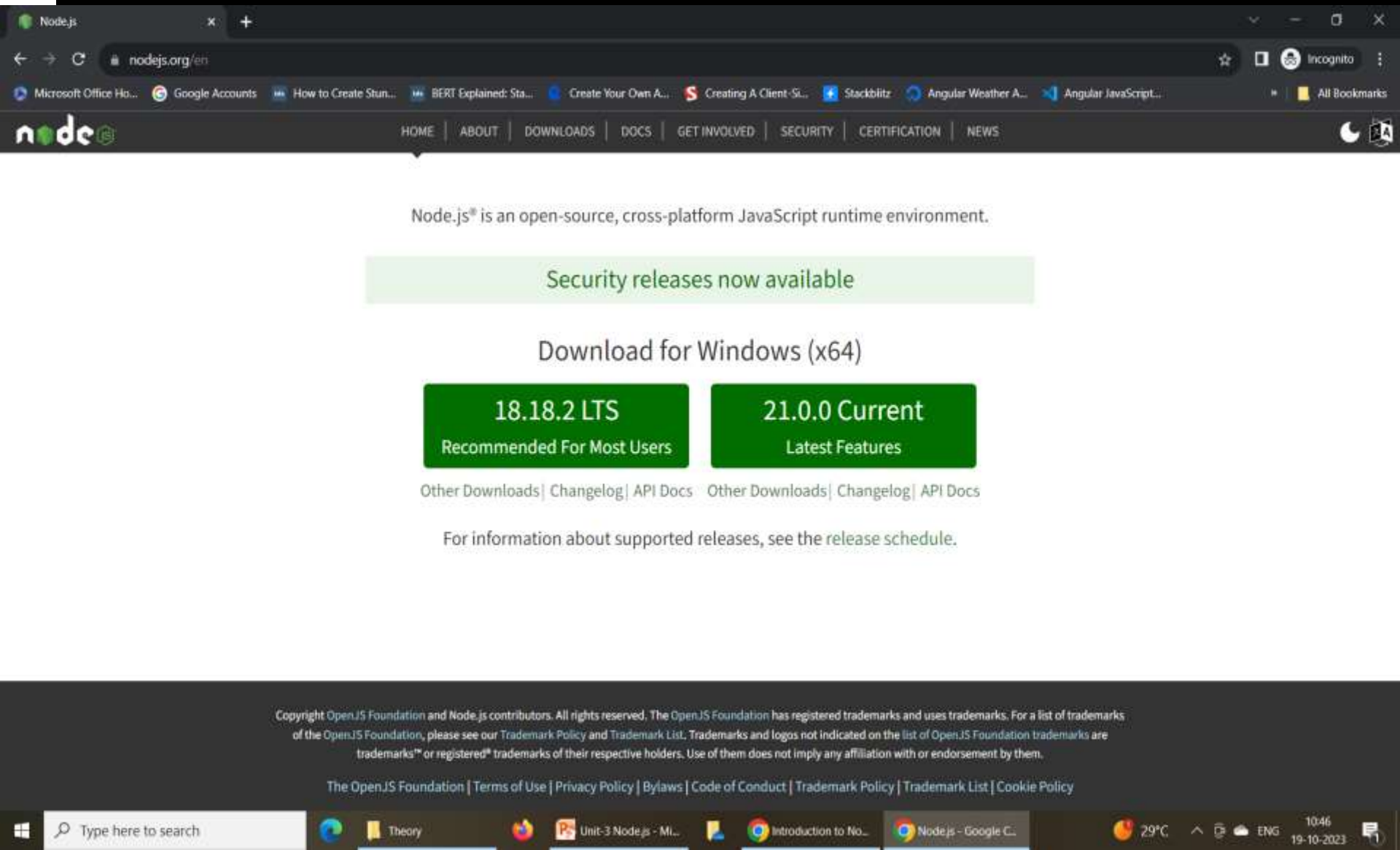
3.1 Introduction to Node.js:

- It is an **open-source and cross-platform JavaScript runtime environment**.
- allows us to **run JavaScript on the server**.
- Node.js was **written initially by Ryan Dahl** in 2009.
- Latest Version – **21.0.0** (includes npm 10.2.0)
- It is a JavaScript runtime built on **Chrome's V8 JavaScript engine**.
  - JavaScript engine that **powers Google Chrome**
  - **Provides the runtime environment in which JavaScript** executes and is independent of the browser in which it's hosted.

- It is an **open source (free) server environment**.
- runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.).
- Node (or more formally Node.js) **allows developers to create all kinds of server-side tools and applications in JavaScript**.
- **primarily used to build network programs such as Web servers**.
- **Definition:** *Node.js is an open-source, cross-platform, JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.*

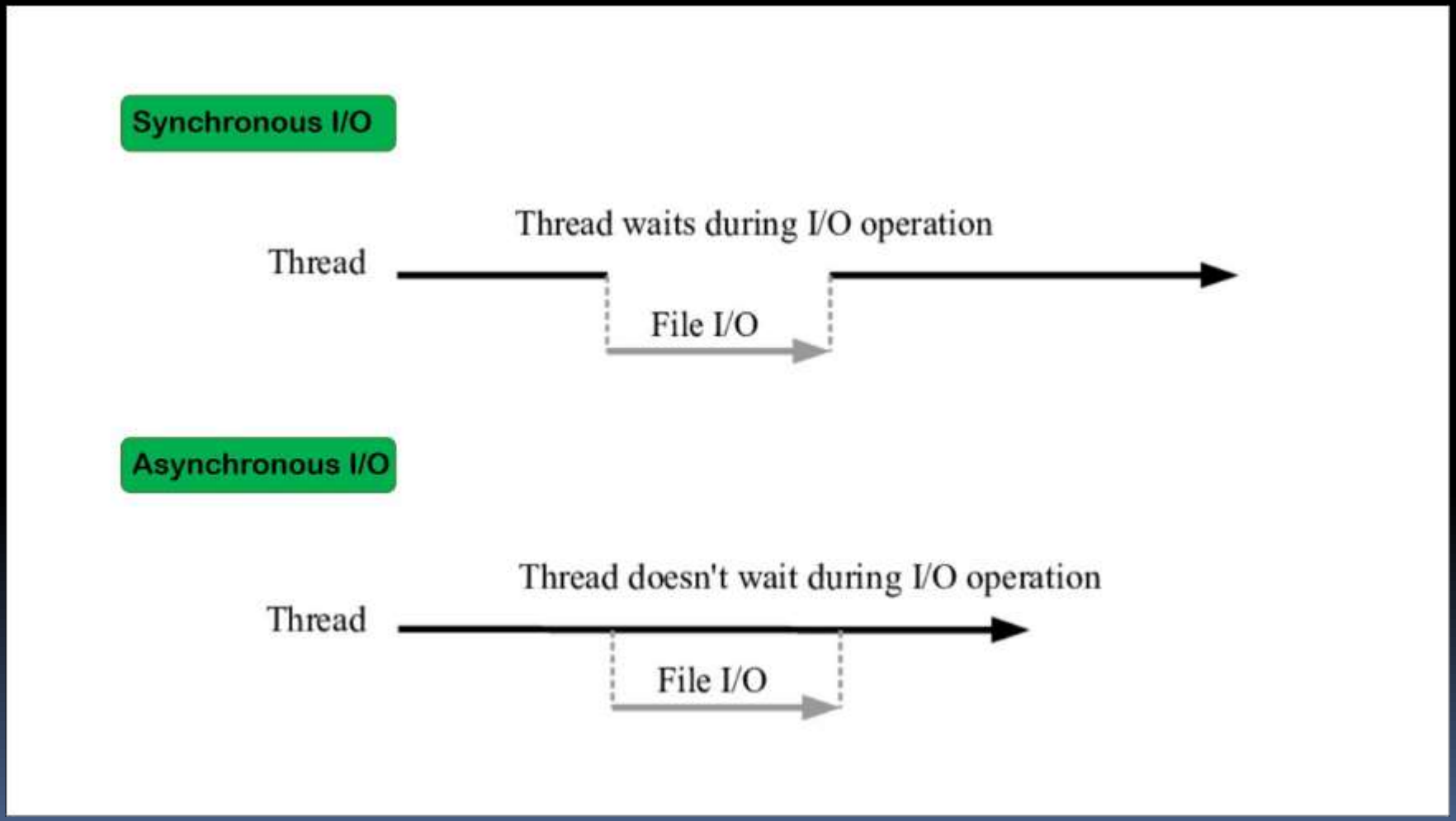
Downloading & Installing:

http://nodejs.org



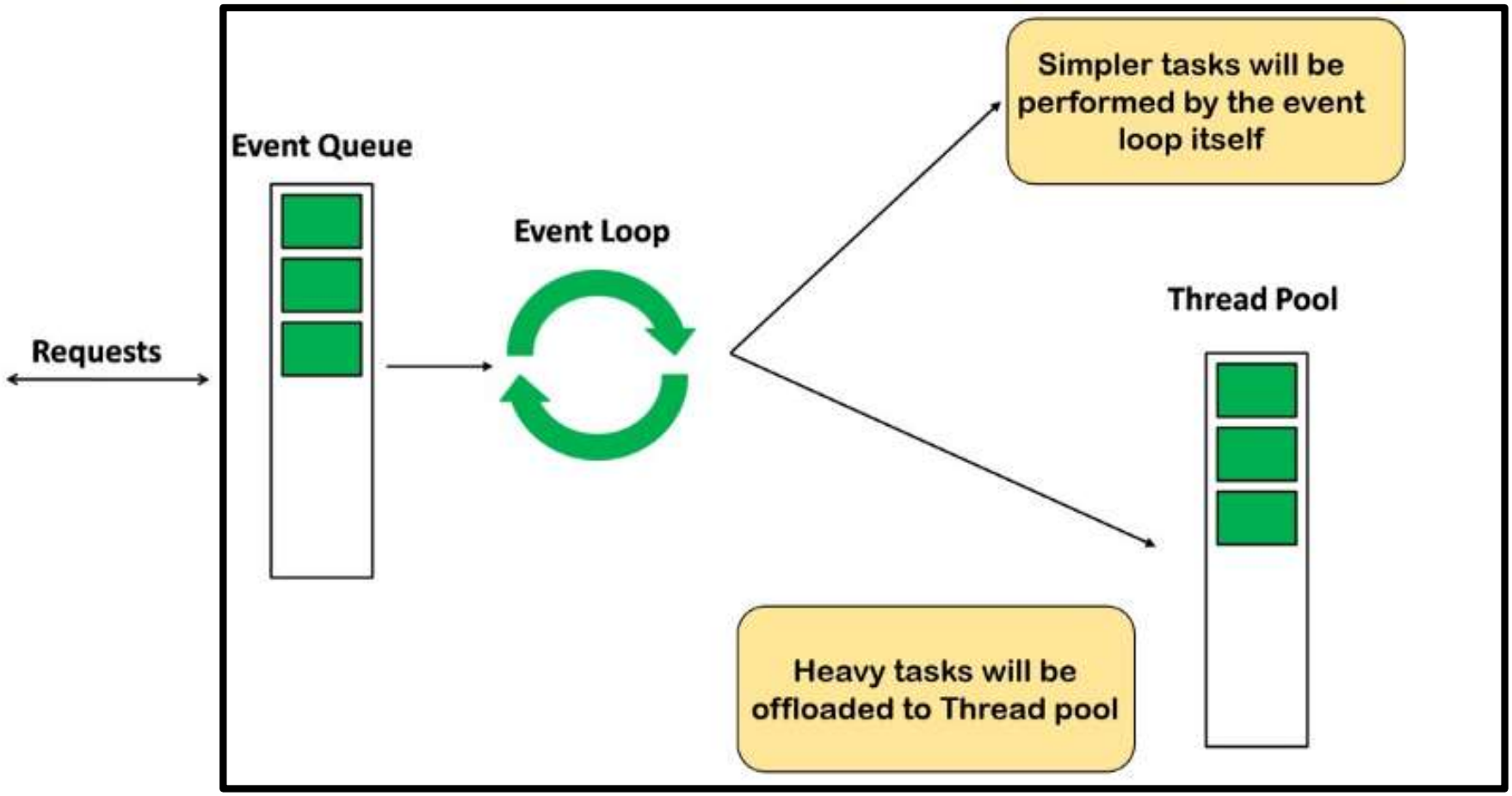
- can execute an external JavaScript file by executing the command:
  - **node fileName.js**

Asynchronous/Non blocking Model:

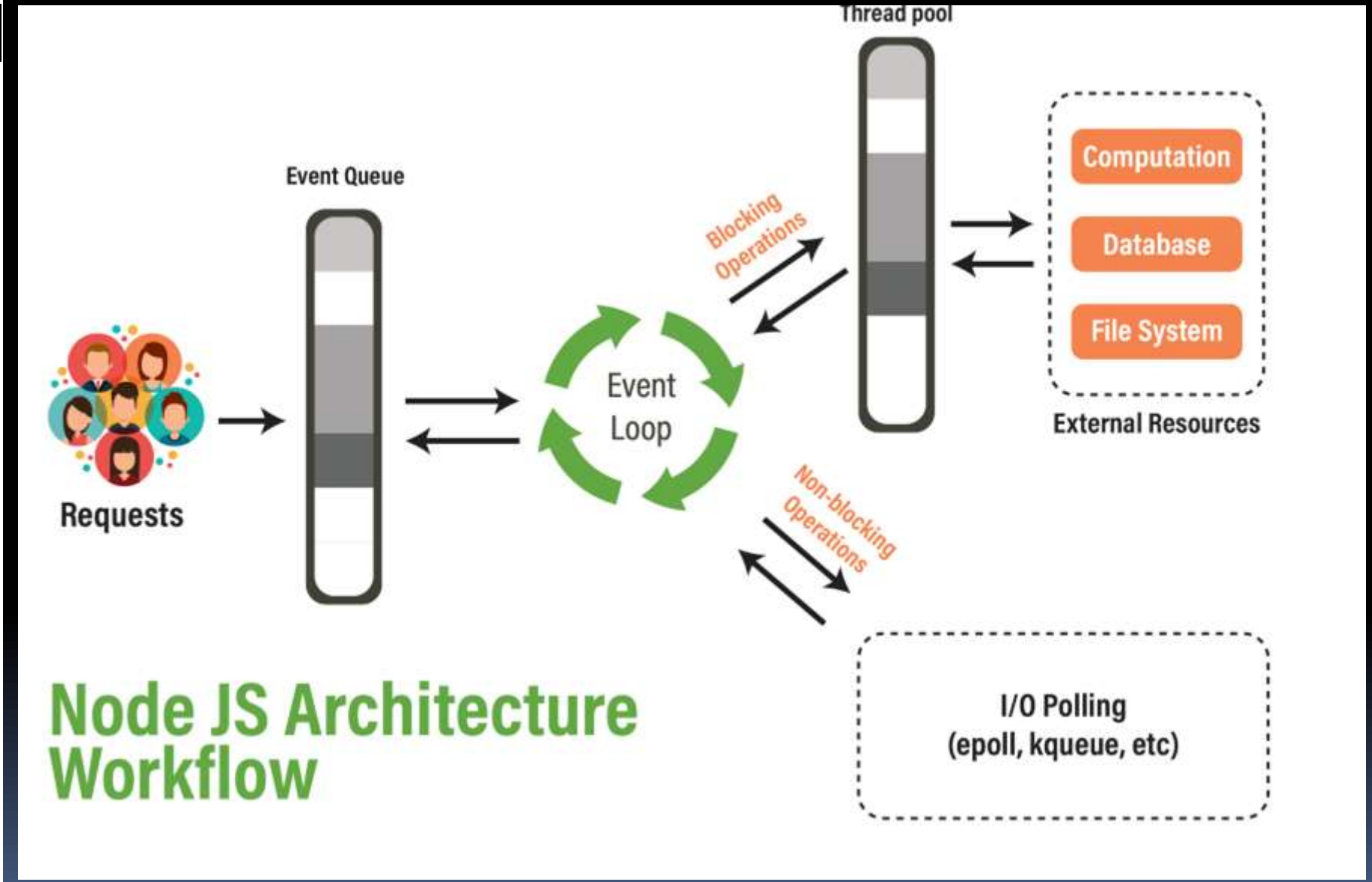


Node.js Architecture

- To manage several concurrent clients, Node.js employs a **"Single Threaded Event Loop"** design.
- The JavaScript **event-based model** and the JavaScript **callback mechanism** are employed in the Node.js Processing Model.
- It employs two fundamental concepts:
  - **Asynchronous & Non-blocking of I/O operations.**
  - **Single Threaded Event Loop Model.**


Simpler tasks will be performed by the event loop itself
Event Queue
Event Loop
Thread Pool
Requests
Heavy tasks will be offloaded to Thread pool

Node Server



Node JS Architecture Workflow

Components of the Node.js Architecture:

- **Requests:** Depending on the actions that a user needs to perform, the requests to the server can be either blocking (complex) or non-blocking (simple).
- **Node.js Server:** The Node.js server accepts user requests, processes them, and returns results to the users.
- **Event Queue:** The main use of Event Queue is to store the incoming client requests and pass them sequentially to the Event Loop.
- **Event Loop:** Event Loop receives requests from the Event Queue and sends out the responses to the clients.
- **Thread Pool:** The Thread pool in a Node.js server contains the threads that are available for performing operations required to process requests.
- **External Resources:** In order to handle blocking client requests, external resources are used. They can be of any type (computation,databases, filesystem etc).

Single Threaded Event Loop Model Processing Steps:

- Clients Sends request to Web Server.
- Node JS Web Server receives those requests and places them into a Queue. It is known as **"Event Queue".**
- Node JS Web Server internally has a Component, known as "**Event Loop"** which uses indefinite loop to receive requests and process them.
- **Event Loop uses Single Thread only.** It is main heart of Node JS Processing Model.
- Event loop pick up one Client Request from Event Queue and starts processing that Client Request.
- If that Client Request **DOES NOT requires any Blocking IO** Operations, then process everything, prepare response and send it back to client.
- **Node JS Web Server** internally maintains a **Limited Thread pool** to provide services to the Client Requests.

- If that **Client Request requires some Blocking IO Operations** like interacting with Database, File System, External Services then it will follow different approach
  - Checks Threads availability from Internal Thread Pool
  - Picks up one Thread and assign this Client Request to that thread.
  - That Thread is responsible for taking that request, process it, perform Blocking IO operations, prepare response and send it back to the Event Loop
  - Event Loop in turn, sends that Response to the respective Client.
- Here Client Request is a call to one or more Java Script Functions.
- Java Script Functions may call other functions or may utilize its Callback functions nature. So Each Client Request looks like as shown below:

```
function(other-functioncall, callback-function)
```

Example:

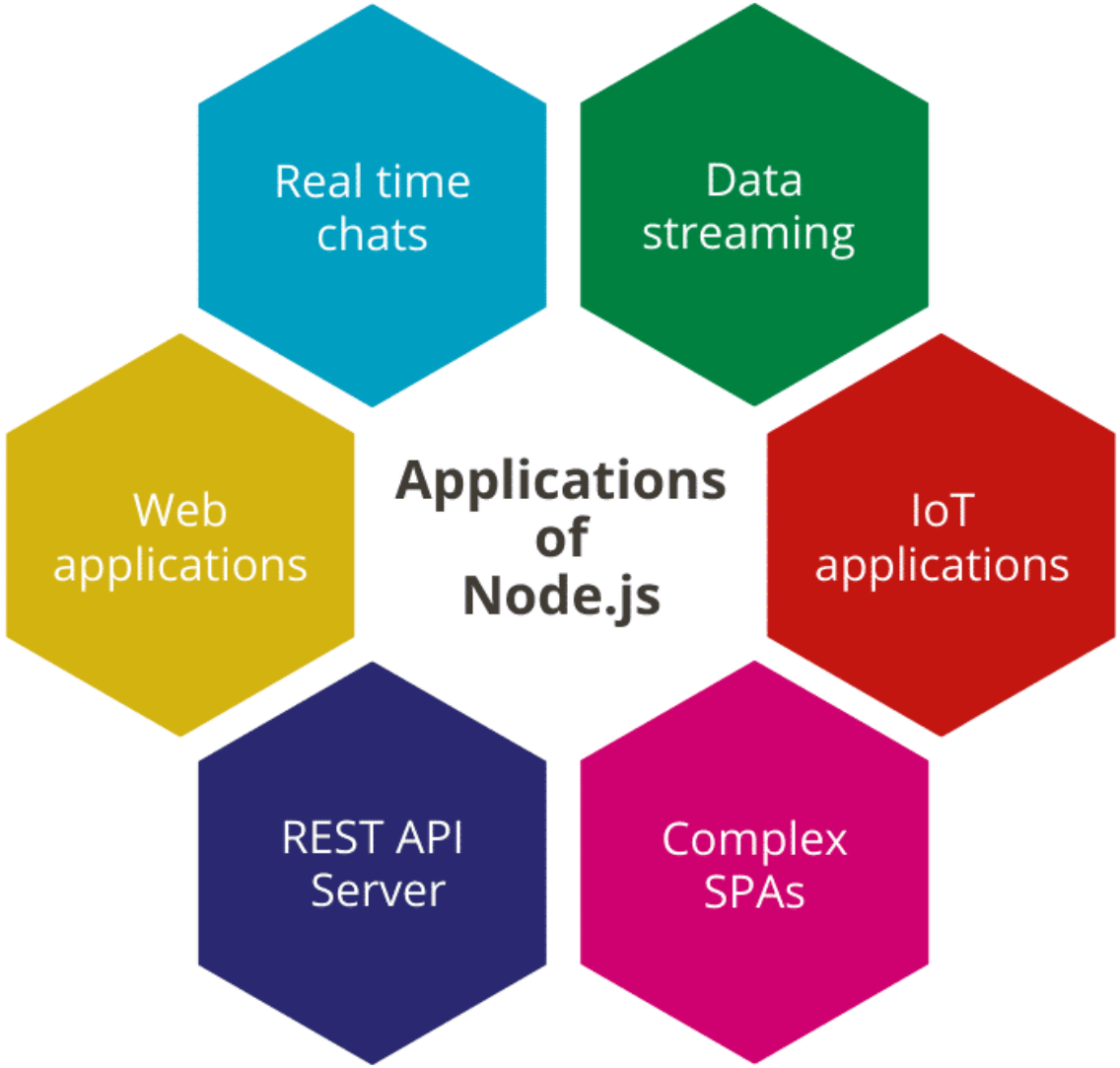
```
function1(function2,callback1);
function2(function3,callback2);
function3(input-params);
```

Node JS Architecture – "Single Threaded Event Loop" Advantages

- **Handling more and more concurrent client's request is very easy.**
- Even though our Node JS Application receives more and more Concurrent client requests, there is no need to establish multiple threads because **Event Loop processes all requests one at a time**, therefore a **single thread is sufficient**.
- Node JS application **uses less Threads so that it can utilize only less resources or memory**

Features of node.js:

- **Open Source**
- **Uses Asynchronous /Non Blocking model:**
  - eliminates the waiting, and simply continues with the next requests and will resume the operations when the response comes back.
- **Uses a** Single Threaded Event Loop Model to handle multiple concurrent clients.
  - Memory Efficient
- **Very Fast**
  - built on Google Chrome's V8 JavaScript Engine which compiles JavaScript source code to native machine code at runtime.
- **A Vast Number of Libraries:**
  - over 1,000,000 open source packages you can freely use.
- **Multi-platform**
  - Cross-platform support allows you to create complex SPA's, desktop apps, and even mobile apps, all using Node.js.



npm (Node Package Manager)

- npm is the default package manager for **Node.js**
- It consists of:
  - a **command line tool** , also called **npm**, and
  - an online database of public and paid-for private packages/modules, called the **npm registry.**
- npm is a command line tool that
  - **manages downloads of dependencies of your project**.
  - **installs**, **updates** or **uninstalls** Node.js packages/modules in your application.
- The package manager and the registry are managed by **npm, Inc.**
- **www.npmjs.com** hosts thousands of free packages to download and use.

- When you install Node.js, NPM is also installed.
- **Commands:**
  - **Verify NPM installation**
    - npm -v
  - **Update it to the latest version**
    - npm install npm -g
  - **To access NPM help**
    - npm help

Packages

- A package **consists of one or more modules and all the files you need for a module**.
- A **package** is a file or directory that is described by a **package.json** file.
- A package must contain a package.json file in order to be published to the npm registry.

- **package.json:**
  - All **npm packages contain a file called package.json** this file holds various **metadata relevant to the project**.
  - add a package.json file to your package to make it easy for others to manage and install.
  - This file is used to give information to **npm** that allows it to identify the application/package as well as handle its dependencies.
  - Metadata includes – **name, version** , **author, description, main(entry point** for the application) **dependencies**(list of npm packages installed as dependencies), browserslist **etc.,**

Sample package.json file:

```
{
    "name" : "test-project",
    "version" : "1.0.0",
     "description " : “A node.js application "
}
```

Creating a default package.json file:

```
npm init  --yes
```

Creating package.json file:

```
npm init
```

Installing a package using npm

- **npm install <package-name>**
  - the package is installed in the **current file tree**, under the **node_modules** subfolder. (local install)
- **npm install -g <package-name>**
  - npm won't install the package under the local folder, but instead, it will **use a global location**. (global install)
  - **npm root -g** command will tells us the exact location on our machine.

- To install specific version of a package
  - **npm install <package-name> @ <package-version>**
- Updating a package using npm
  - **npm update <package-name>**
  - **npm update –g <package-name>**
- To remove an installed package
  - **npm uninstall <package-name>**
  - **npm uninstall –g <package-name>**

3.2 REPL

- **Read-Eval-Print-Loop**
- **easy-to-use command-line tool, used for processing Node. js expressions**.
- 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.
- **It captures the user's JavaScript code inputs, interprets, and evaluates the result of this code.**
- **It displays the result to the screen, and repeats the process till the user quits the shell.**
- ***Definition: "It is a programming language environment (basically a console window) that takes single expression as user input and returns the result back to the console after execution."***

A REPL has:

- A **read function**, which accepts an expression from the user and parses it into a data structure in memory.
- An **eval function**, which takes the data structure and evaluates the expression.
- A **print function**, which prints the result.
- A **loop function**, which runs the three commands above until termination

- **multi line JavaScript expression or functions** - press **Enter** as a continuation of your code. The REPL terminal will display three dots **(...)**, it means you can continue on next line.

REPL Commands

- **node -** To launch the REPL
- **.help:** list of all commands.
- **.break:** Exit from multiline expression.
- **.clear:** **Alias for .break**
- **.exit:** **Exit the REPL**
- **.save:** Save **all evaluated commands in this** current REPL session to a file: > .save ./file/to/save.js
- **.load:** Load JS from a file into the current REPL session. > .load ./file/to/load.js
- **.editor:** Enter editor mode (Ctrl+D to finish, Ctrl+C to cancel).

- **ctrl + c** – terminate the current command.
- **ctrl + c twice** – terminate the Node REPL.
- **ctrl + d** – terminate the Node REPL.
- **Up/Down Keys** – see command history and modify previous commands.
- **tab Keys** – current commands auto completion options.

3.3 MODULES

- Modules are like **JavaScript libraries** that can be used in a Node.js application
- ***Includes a set of functions.***
- Modules **can be reused throughout the Node.js application**.
- In the Node.js module system, each file is treated as a separate module i.e **each module can be placed in a separate .js file.**
- Node.js implements CommonJS modules standard. CommonJS is a group of volunteers who define JavaScript standards for web server, desktop, and console application.

Types of modules

1. **Core Modules** –
   - that are part of the platform and comes with Node.js installation.
   - can be loaded into the program by using the **require** function.
   - Examples : http, os, fs, os, util
2. **Local Modules/file-based**
   - define the Node module within a file, within our application and
   - we make use of it within our application using **require** function

3. Third Party Modules –

- that are developed by Node developers, and
- then made available through the Node ecosystem.
- These modules can be installed in the project folder or globally using **NPM**.
- Some of the popular third-party modules are mongoose, express, angular, and react.
- Example: **npm install express**

- The following list contains some of the important core modules in Node.js:

Core Modules	Description
http	creates an HTTP server in Node.js.
buffer	To handle binary data
fs	used to handle file system.
path	includes methods to deal with file paths.
process	provides information and control about the current Node.js process.
os	provides information about the operating system.
querystring	utility used for parsing and formatting URL query strings.
stream	To handle streaming data
url	module provides utilities for URL resolution and parsing.

Creating modules (Local)

- To create a module in Node.js, you will need the **exports** keyword.
- This keyword tells Node.js that the function can be used outside the module.
- **Syntax:**
- (1) module.exports = function / literal / object
- (2) exports.function_name = **function**(arg1, arg2, ….argN) { // function body };

Including Your Own Module

- To include a module in a Node.js application, use the **require()** function with the parenthesis containing the name of the module.
- Syntax:
  - **require('./myfirstmodule')**

./ is used to locate the module, that means that the module is located in the same folder as the Node.js file.

OS Module:

- The OS module provides **information about the underlying operating system** and the computer the program runs on.
- Some methods:

Method	Description
arch()	Returns the operating system CPU architecture
cpus()	Returns an array containing information about the computer's CPUs
freemem()	Returns the number of free memory of the system
hostname()	Returns the hostname of the operating system
networkInterfaces()	Returns the network interfaces that has a network address
platform()	Returns information about the operating system's platform
release()	Returns information about the operating system's release
tmpdir()	Returns the operating system's default directory for temporary files
totalmem()	Returns the number of total memory of the system
type()	Returns the name of the operating system
userInfo()	Returns information about the current user

HTTP Module:

- The HTTP core module is **a key module to Node.js networking.**
- **provides a way of making Node.js transfer data over HTTP** (Hyper Text Transfer Protocol).
- The HTTP module **creates an HTTP server** that listens to server ports and gives a response back to the client.
- **Syntax:**
  - var http = require('http');

Method	Description
createServer()	Creates an HTTP server
get()	Sets the method to GET, and returns an object containing the user's request
globalAgent()	Returns the HTTP Agent
request()	Returns an object containing the user's request

Events Module:

- Node.js has an **event-driven architecture** which can perform asynchronous tasks.
- Node.js has a built-in module, called "**events**", where you can **create-, fire-, and listen** for- your own events.
- **'events'** module **emits named events** that can cause corresponding functions or callbacks to be called.
- Functions(Callbacks) register to a particular event to occur and when that event triggers, all the callbacks subscribed to that event are fired one by one in order to which they were registered.
- **Syntax:** **var e = require('events');**

- The EventEmitter Object:
  - The events module provides us the EventEmitter class, which is **key to working with events in Node.js**.
  - We **can assign event handlers** to your own events with the EventEmitter object.
  - An EventEmitter object has two main functions:
    - **Emit a named event.**
    - **Attach and detach one or more event listeners to the named event.**
  - **Syntax:**
  - var events = require('events');
    var eventEmitter = new events.EventEmitter();

- Some EventEmitter Methods:
  - eventEmitter.on(event, listener)
  - eventEmitter.off(event,listener)
  - eventEmitter.emit(event, [arg1], [arg2], [...])
  - eventEmitter.once(event, listener)
  - eventEmitter.removeAllListeners([event])
  - eventEmitter.listenerCount(event)
  - eventEmitter.setMaxListeners(number)
  - eventEmitter.getMaxListeners()

Steps:

- Create an event handler
- Assign the event handler to an event
  - using eventEmitter.on(event, listener)
- Fire the event
  - Using eventEmitter.emit(event)

Stream Module:

- Streams are a way to handle reading/writing files, network communications, or any kind of end-to-end information exchange in an efficient way.
- **For example:**
  - In the traditional way, when you tell the program to read a file, the file is read into memory, from start to finish, and then you process it.
  - Using streams you read it piece by piece, processing its content without keeping it all in memory.
- **Advantages of streams:**
  - **Memory efficiency**
  - **Time efficiency**

- **The Stream module provides a way of handling streaming data.**
- In Node.js, there are four types of streams –
  - **Readable** – Stream which is used for read operation.
    - **Example:** **fs.createReadStream()**
  - **Writable** – Stream which is used for write operation.
    - **Example:** **fs.createWriteStream()**

  Duplex – Streams that are both, Writable as well as Readable.
  - **Transform** – A type of duplex stream that can modify or transform the data as it is written and read.

- **Each type of Stream is an EventEmitter instance and throws several events** at different instance of times.
- Some of the commonly used events are –
- **data** – This event is fired when there is data is available to read.
- **end** – This event is fired when there is no more data to read.
- **error** – This event is fired when there is any error receiving or writing data.
- **finish** – This event is fired when all the data has been flushed/written to underlying system.

- Some methods **returns a readable/writable stream object**, like **http.createServer()**, and if that is the case, you do not have to include the stream module.
- Otherwise, the syntax for including the Stream module in your application:
- var stream = require('stream');

Buffer Module:

- A buffer is an **area of memory**.
- It represents a **fixed-size chunk of memory (can't be resized) allocated outside of the V8 JavaScript engine.**
- can think of a buffer **like an array of integers**, which each represent a byte of data.
- **The buffers module provides a way of handling streams of binary data.**
- The Buffer object is a **global object in Node.js**, and it is **not necessary to import** it using the require keyword.

- **Creating a Buffer:**
  - var buf = Buffer.alloc(size)
  - var buf = Buffer.from(string/array/buffer);

Writing to Buffers:

- Buffer.write(String)

Reading contents of a Buffer:

- Buffer.toString()

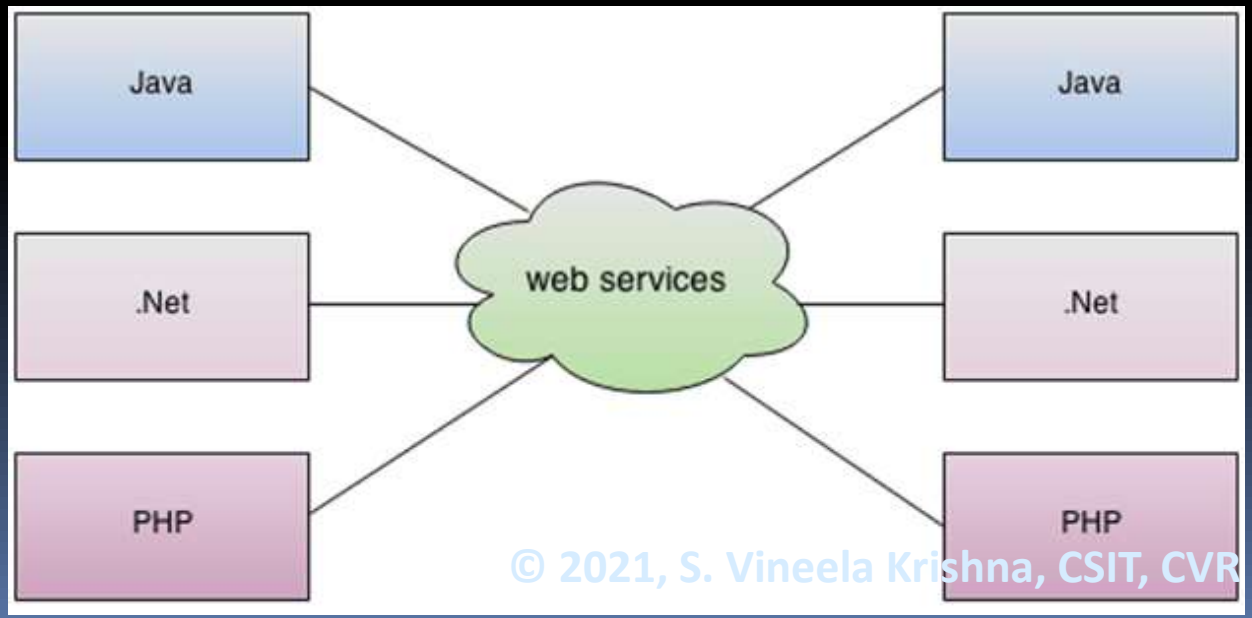
- **Some Buffer module methods:**
  - Buffer.concat(list)
  - Buffer.compare(otherBuffer)
  - Buffer.slice([start][, end])
  - Buffer.toJSON()
  - Buffer.length -> property

3.4 Building own RESTful API and consuming it

REST/RESTful API:

- REST stands for **RE**presentational **S**tate **T**ransfer.
- Definition: It **defines an architectural style for building web services that interact via an HTTP protocol**.
- **defines a set of constraints** to be used for creating web services.
- **"A web API that obeys the REST constraints is described as RESTful API. "**
- RESTful web APIs are typically **based on HTTP methods** to access resources via URL-encoded parameters and the use of JSON or XML to transmit data.

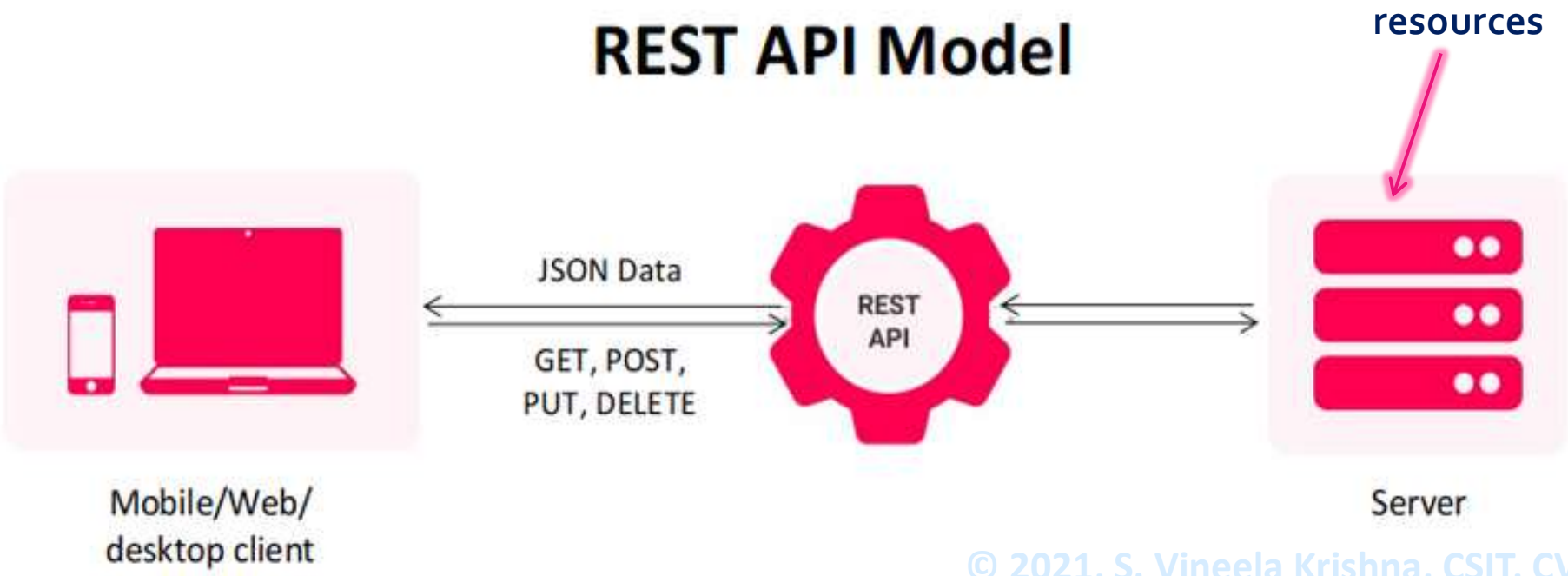
- **Web service (WS) :**
  - **Is a service offered by an electronic device to another electronic device**, communicating with each other via the World Wide Web, or
  - **A server running on a computer device, listening for requests at a particular port over a network, serving web documents** (HTML, JSON, XML, images).
  - It is a **software system for the interoperable machine to machine communication**.
  - It is a **collection of standards or protocols for exchanging information between two devices or application**.



- Four basic design principles of REST: –
  - **Use HTTP methods explicitly**
  - **Stateless** - Every request is a new request from the client and doesn't maintain any information about clients
  - **Provide access to the resources through URLs**
  - **Transfer using XML, JavaScript Object Notation (JSON), or both**

REST System consists of a :

- **Client who requests for the resources.**
- **Server who has the resources.**
- A request is sent from client to server in the form of a web URL with methods such as HTTP GET or POST or PUT or DELETE.
- A response from the server is a resource represented in the form of a JSON/XML.



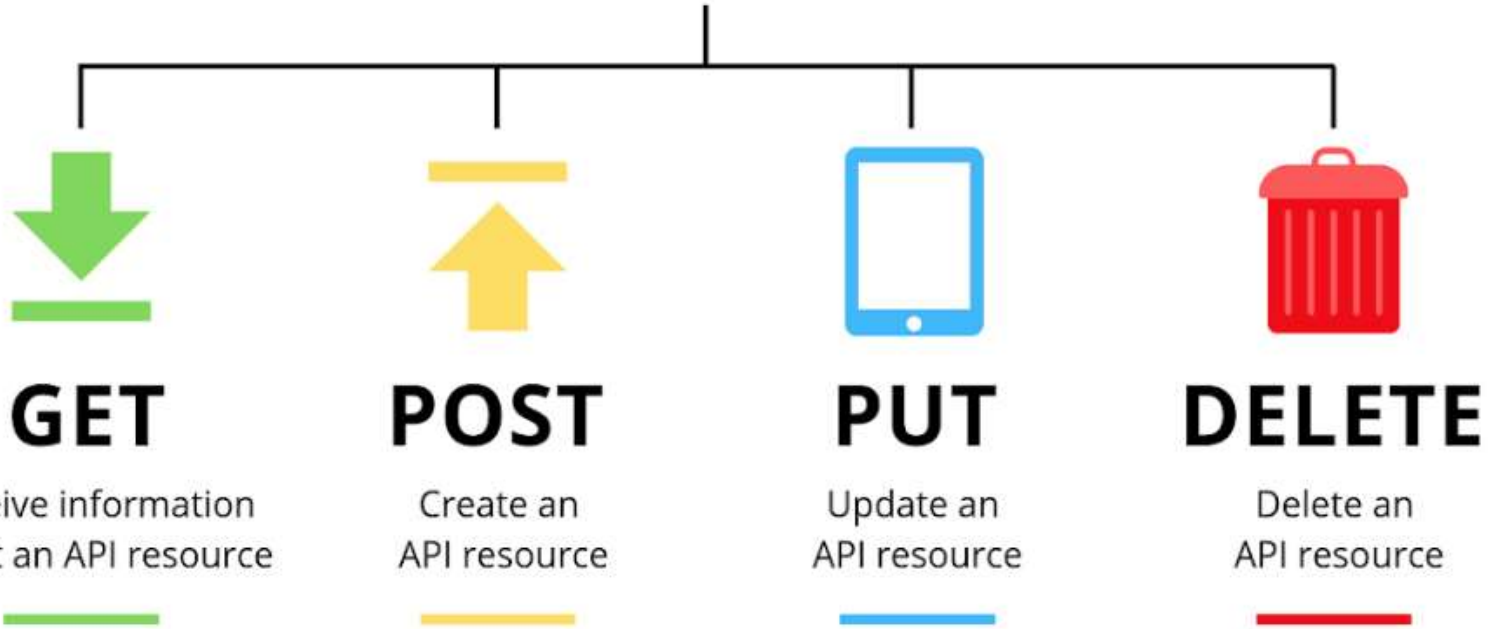
key elements of the REST/RESTful API paradigm:

1. **Resources:**
   - The key abstraction of information in REST is a resource.
   - which is any piece of content that the server can provide to the client (for example, a video or a text file).
   - Clients can only access resources using **URIs**
2. **Representation:**
   - How data is represented or returned to the client for presentation
   - The client requests a resource using a URI and the server responds with a **representation** of the resource.
   - **Two main formats: –**
     - JavaScript Object Notation **(JSON) & XML**

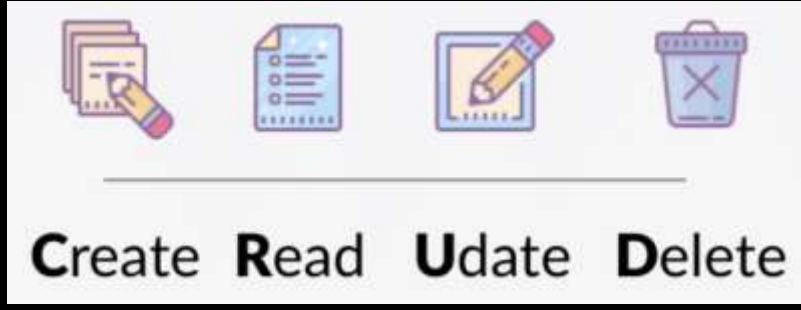
3. HTTP Methods:

- **REST is built upon HTTP Protocol**
- To get access to a resource, the client sends an **HTTP request**.
- In return, the server generates an **HTTP response** with encoded data on the resource.
- An **HTTP method** describes what is to be done with a *resource (*POST, GET, PUT, PATCH, and DELETE)

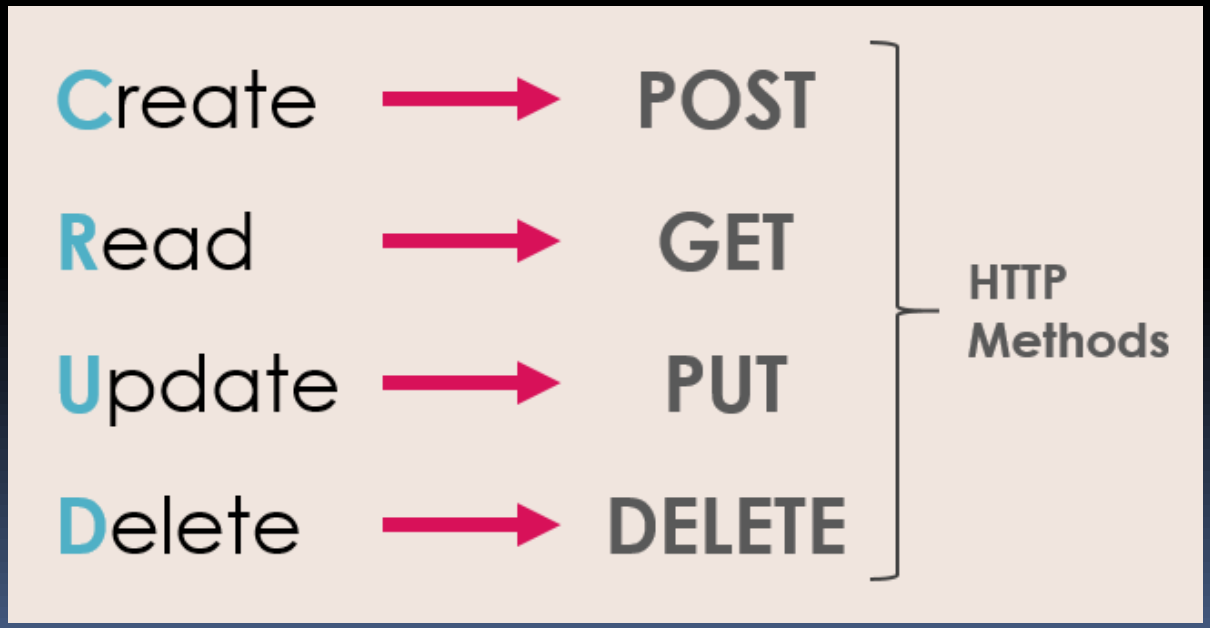
REST API Methods



CRUD Operations:



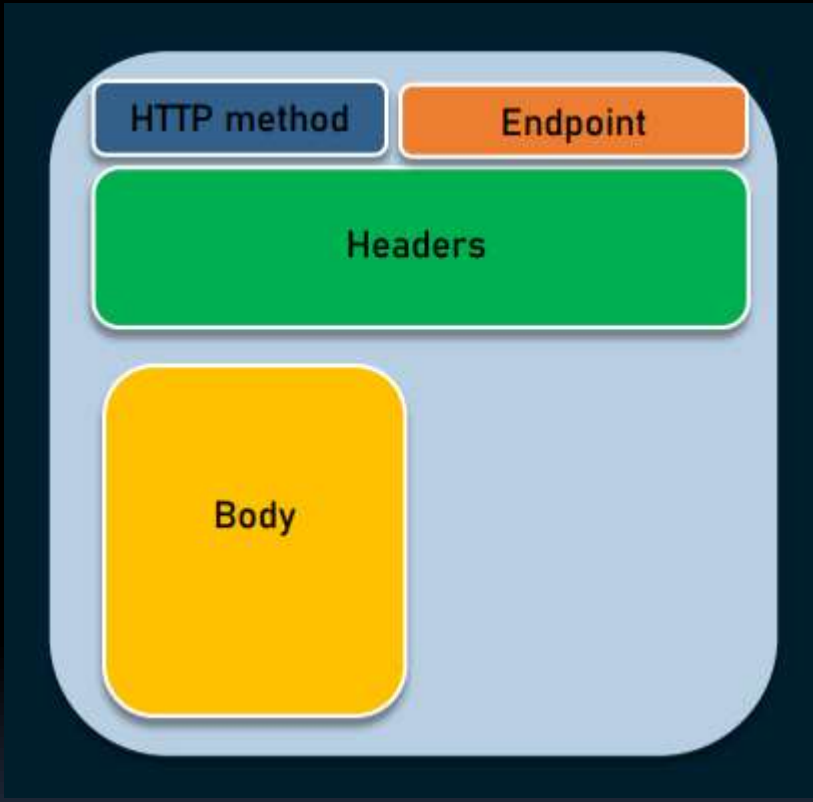
- **HTTP Methods** or REST API Methods **corresponds to the CRUD operations to be performed on a resource.**
- **CRUD operations – operations on Data Base that stores resources on the server side.**



- **HTTP POST -> CREATE**
  - Used to create new resource
- **HTTP GET -> READ**
  - Used to read (or retrieve) a representation of a resource.
- **HTTP PUT -> UPDATE**
  - Used to update an existing resource
- **HTTP DELETE -> DELETE**
  - Used to remove the resource identified by the URI

REST request structure

- Any REST request includes four essential parts:
  - **HTTP method**,
  - **Endpoint**
  - **Headers**, and
  - **Body**

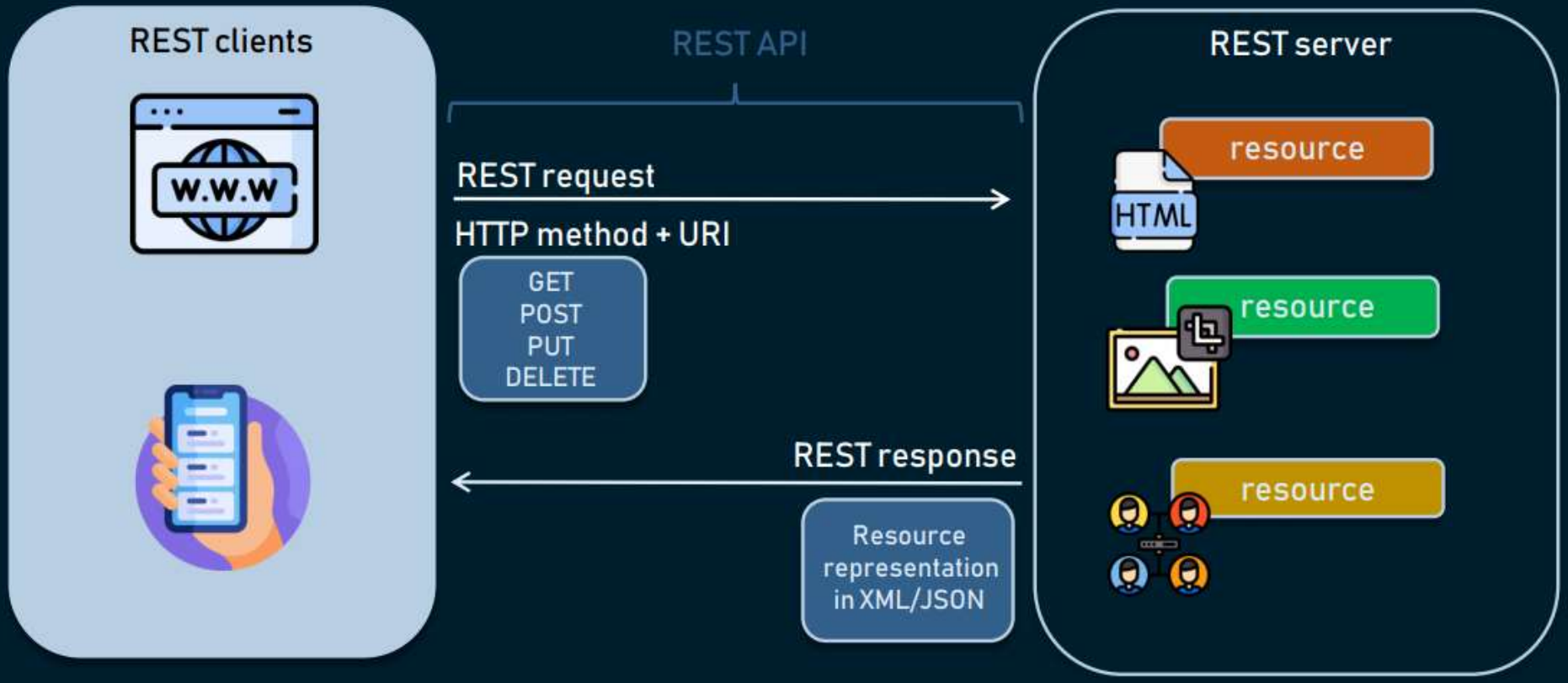


- An **HTTP method** describes what is to be done with a *resource*.
- An **endpoint** contains a *Uniform Resource Identifier (URI)* indicating where and how to find the resource on the Internet.
- **Headers** store information relevant to both the client and server.
  - Mainly, headers provide authentication data — such as
    - an API key,
    - the name or IP address of the computer where the server is installed,
    - and the information about the response format.
- A **body** is used to convey additional information to the server. For instance, it may be a piece of data you want to add or replace.

REST response structure

- In response, the server sends not the sought-for resource itself, but its **representation** —
  - a machine-readable description of its current state.
  - most popular representations are **XML and JSON**.

REST API IN ACTION



Building REST/RESTful API:

1. Install Node.js and NPM
2. Create a new Node.js project
3. Create package.json file
4. Install necessary dependencies/packages
5. Create a file called "Server.js"
6. Start and test the server using nodemon
7. Define API URL endpoints for various HTTP Requests.
8. Testing API with Postman/Thunder Client