Develop a Reliable Backend with Node and Express

Events and Request Management

A Day in the Life of a MERN Stack Developer

Joe is working as a MERN stack developer. His company acknowledges and appreciates his commitment and progress, as he has diligently completed all the projects assigned to him.

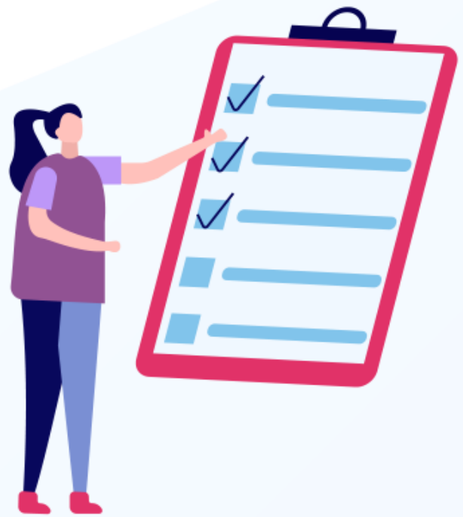
He has now been assigned to an important project and asked to export different values as modules in Node.js to maintain and manage the code base, achieving modularization.

In this lesson, Joe will learn how to solve this real-world scenario effectively and quickly.

Learning Objectives

By the end of this lesson, you will be able to:

- Discuss the features of native modules that empower developers to leverage functionalities without external dependencies
- Analyze events and event emitters to enhance application modularity
- Assess error handling techniques in Node.js, ensuring robustness and reliability in application development
- Implement routing in Node.js to handle client requests which enables the server to respond to different endpoints and HTTP methods
- Install packages using npm to streamline development by accessing a vast repository of reusable code and utilities



Exports in Node.js

Exports

Export is an object that is exposed as a module.

- Exports are used to export one or more variables or functions.
- They are not returned by the require() method.
- They are just references to the **module.exports.**

module.exports

module.exports in Node.js exports a single class, variable, or function and is returned by **require()**.

Syntax:

```
module.exports = literal | function | object
```

Here, the assigned value is exported and works as a module.

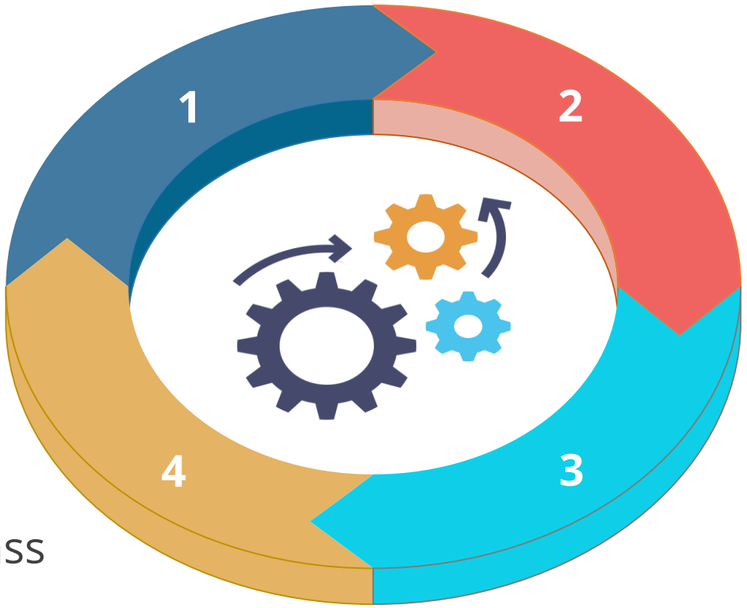
module.exports also include the JavaScript file in the Node.js applications.

module.exports

module.exports exposes different types as a module, namely:

Export literals

Export object



Export function as class

Export function

Export Literals

The steps to export a string message as a module are:

Step 1: Create a file named **application.js** and export the object with a string property using the **module.exports**

```
module.exports = 'Welcome to Node JS';
```

Step 2: Create a file named **index.js** and import the file **application.js** to print the exported literal to the console

```
const message = require ('./application');
console.log(message);
```

Export Object

The steps to attach properties or methods to an exports object are:

Step 1: Create a file named **application.js** and expose the object with a string property using **module.exports**

```
module.quote = 'Search the Candle rather than cursing the darkness';
```

Step 2: Create a file named **index.js** and import the file **application.js** to print the exported literal to the console

```
const app = require ('./application');
console.log(app.quote);
```

Export Function

The steps to attach an anonymous function to an exports object are:

Step 1: Create a file named **application.js** and expose function with a string input using **module.exports**

```
module.exports = function (msg) {
console.log(msg);
};
```

Step 2: Create a file named **index.js** and import the file **application.js** to print the exported literal to the console

```
const app = require ('./application');
console.log(app.quote);
```

Export Function as a Class

The steps to expose a function that can be used like a class are:

Step 1: Create a file named **application.js** and expose the object with a string property using **module.exports**

```
module.exports = function (name, price) {
this.name = name;
this.price = price;
this.description = function () { return this.name + '
' + this.price; }
}
```

Step 2: Create a file named **index.js** and import the file **application.js** to print the exported literal to the console

```
const Dish = require (‘./application’);
var dish1 = new Dish('Cold Coffee', 100);
console.log(dish1.description());
```

Native Modules in Node.js

Native Modules

Native modules interact with lower-level functions or libraries in Node.js.

Example:

```
const myFeature =
require('./build/Release/feature')
```

This example demonstrates how to build a native module using the name **feature** in a node application.

Native Modules

Node.js addons are native modules that are:

Node.js addons:

01 Written in C or C++

02 Dynamically linked shared objects

03 Loaded using **require()**

Events

Events

The Node.js core API has an asynchronous event-driven design in which some objects emit named events, leading to function objects being called, for instance:

net.Server	Emits an event each time a peer connects to it
fs.ReadStream	Emits an event when the file is opened
Stream	Emits an event whenever data is available

Events Module

In JavaScript, events handle user interaction.

The events module in Node.js can be used to build a similar system on the backend.

It contains the EventEmitter class, which manages events.

EventEmitter

An EventEmitter object has two functions:

Emit a named event

Attach or detach listeners to the named event

```
const EventEmitter = require('events');
const eventEmitter = new EventEmitter();
```

A user can assign event handlers to the events with the EventEmitter object.

Events

Node.js features an events module, allowing users to create, emit, and listen for custom events.

The require() method is used to include the built-in events module.

Events can be used in a code:

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

Events

Events are components of event-driven applications that trigger a callback function.

Difference between a callback function and events:

Callback functions

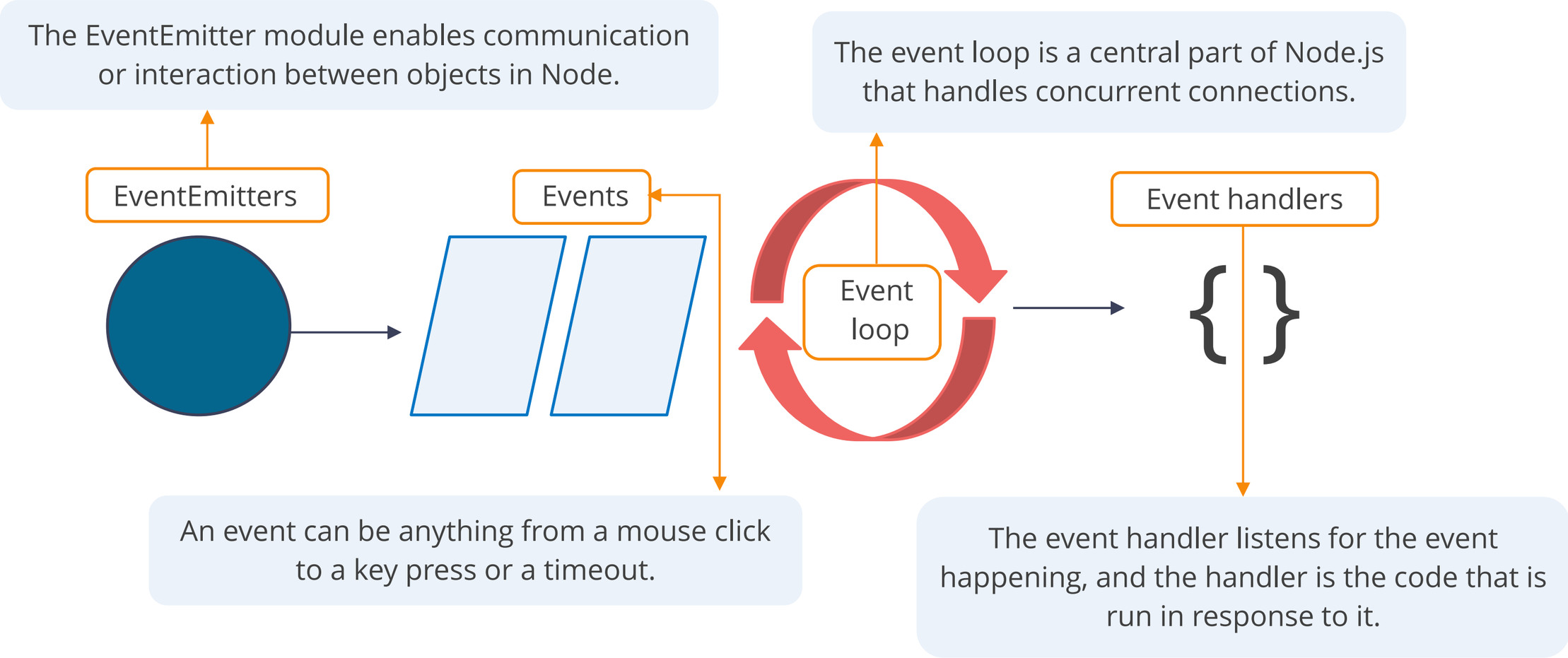
When an asynchronous function returns a result, they are called.

Events

Based on the observer pattern, they are called.

Events

An event flow diagram demonstrates how objects emit events.



Working with Events

Problem Statement: **Duration: 10 min.**

You have been assigned a task to demonstrate creation of events in Node.js.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a simple EventEmitter instance
2. Pass arguments to listeners
3. Use the listener function to switch to asynchronous mode
4. Emit the error events

Object Properties, First Class Functions, and Arrays

Object Properties

Object properties are denoted by name:values pairs.

An example of Object properties considering employees is as follows:

name	John
age	24
gender	Male
salary	50000

Example:

```
const employee = {
  name:"John",
  age:"24",
  gender:"Male",
  salary:50000
};

you can access name as:
employee.age
or
employee["name"]
or
x = "salary"; employee[x]
```

Access the Object Properties

Syntaxes to access the property of an object are as follows:

Syntax:

```
objectName.property
or
objectName["property"]
or
objectName[expression]
```

First-Class Function

A programming language has first-class functions.

First-class functions can be used in different ways:

01 Assigned to any other variable

02 Passed as an argument

03 Returned by another function

Array

An array is used to store a collection of data.

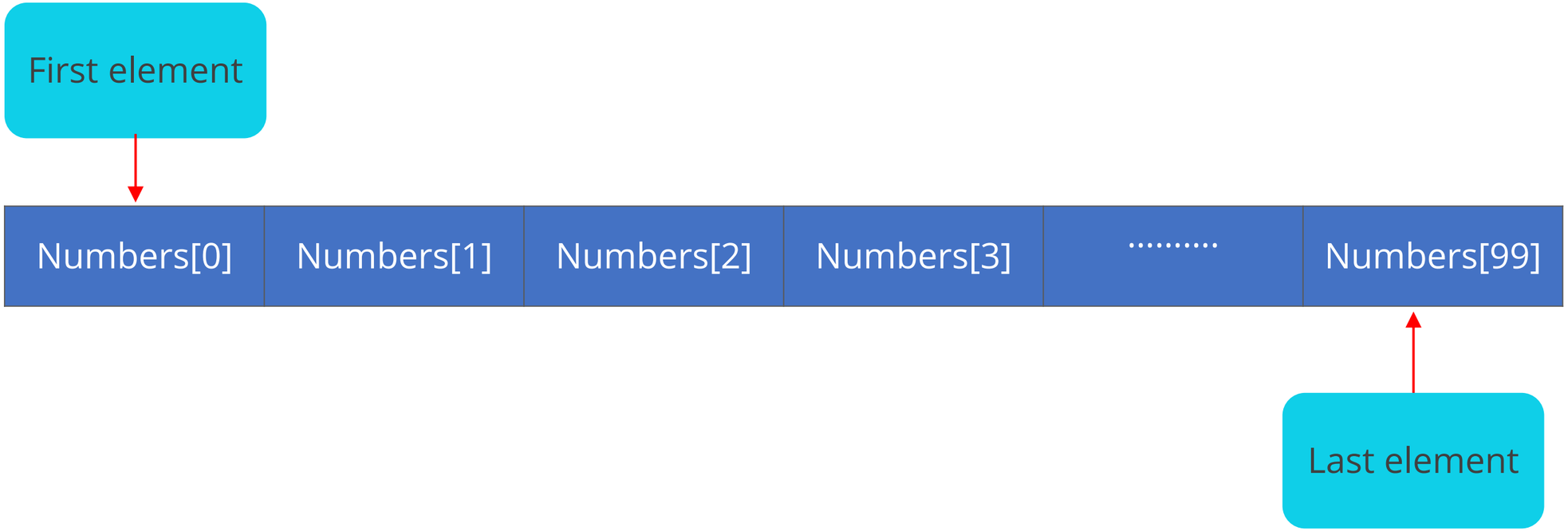
Individual variable	num0, num1,.... and num99
Array	numbers[0], numbers[1], and numbers[99]

It is a collection of same variables.

An index accesses a specific element in an array.

Array

Arrays consist of contiguous memory locations.



The lowest address corresponds to the first element and the highest address to the last element.

Creating Arrays

The user can create an array with or without the elements.

Without elements

```
var array = [ ];
```

With elements

```
var citiesArray = [ ‘Delhi’,
‘Bangalore’, ‘Hyderabad’ ];
```

Object.create()

Object.create() is a static method that helps create new objects using the available objects as prototypes.

Proto: The object that is the prototype of the object which is freshly created.

```
Object.create(proto)

Object.create(proto, propertiesObject)
```

Return value: An object created based on specified prototypes and properties.

If proto is neither null nor an object, exceptions and TypeError are thrown.

Prototype Property

A prototype property is an object that attaches methods and properties to a prototype object.

Example of creating an object using function constructor:

```
function Dish(name, description, price) {
    this.name = name;
    this.description = description;
    this.price = price;
}
console.log(Dish.prototype);
```

Javascript adds prototype property to Object.create(), allowing other objects to inherit methods and properties of this method.

Inheritance with EventEmitter

A class can extend the EventEmitter by inheriting its method and using the extends keyword.

```
const EventEmitter = require('events');
class MyEmitter extends EventEmitter {
    //..
}

const myEmitter = new MyEmitter();
myEmitter.on('event', () => {
    console.log('A New Event Occurred');
});
myEmitter.emit('event');
```

Use class inheritance to create an event emitter

Functions and Arrays

Problem Statement:

Duration: 20 min.

You are given a task to employ functions and arrays in Node.js to get the desired action of the app in a single process.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Declare a simple function in JavaScript
2. Store the function reference in a variable
3. Declare the first-class function using the arrow operator
4. Declare a simple array in JavaScript
5. Push a new value to an array
6. Remove an element from an array
7. Update the value at a particular index in an array

Pass by Reference and Pass by Value

Problem Statement: **Duration: 20 min.**

You have been assigned a task to execute events in Node.js to call function objects.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a simple EventEmitter instance
2. Pass arguments to the listeners
3. Use the listener function to switch to asynchronous mode
4. Emit the error events
5. Handle the events
6. Call and register event listeners

ES6 and Template Literals

ES6

ES6 stands for ECMAScript 2015, which is the sixth edition of the ECMAScript standard. It is the standard upon which JavaScript is based, defining the core features of the language.

ES6 introduced significant enhancements and new features to JavaScript, making it more powerful and expressive.

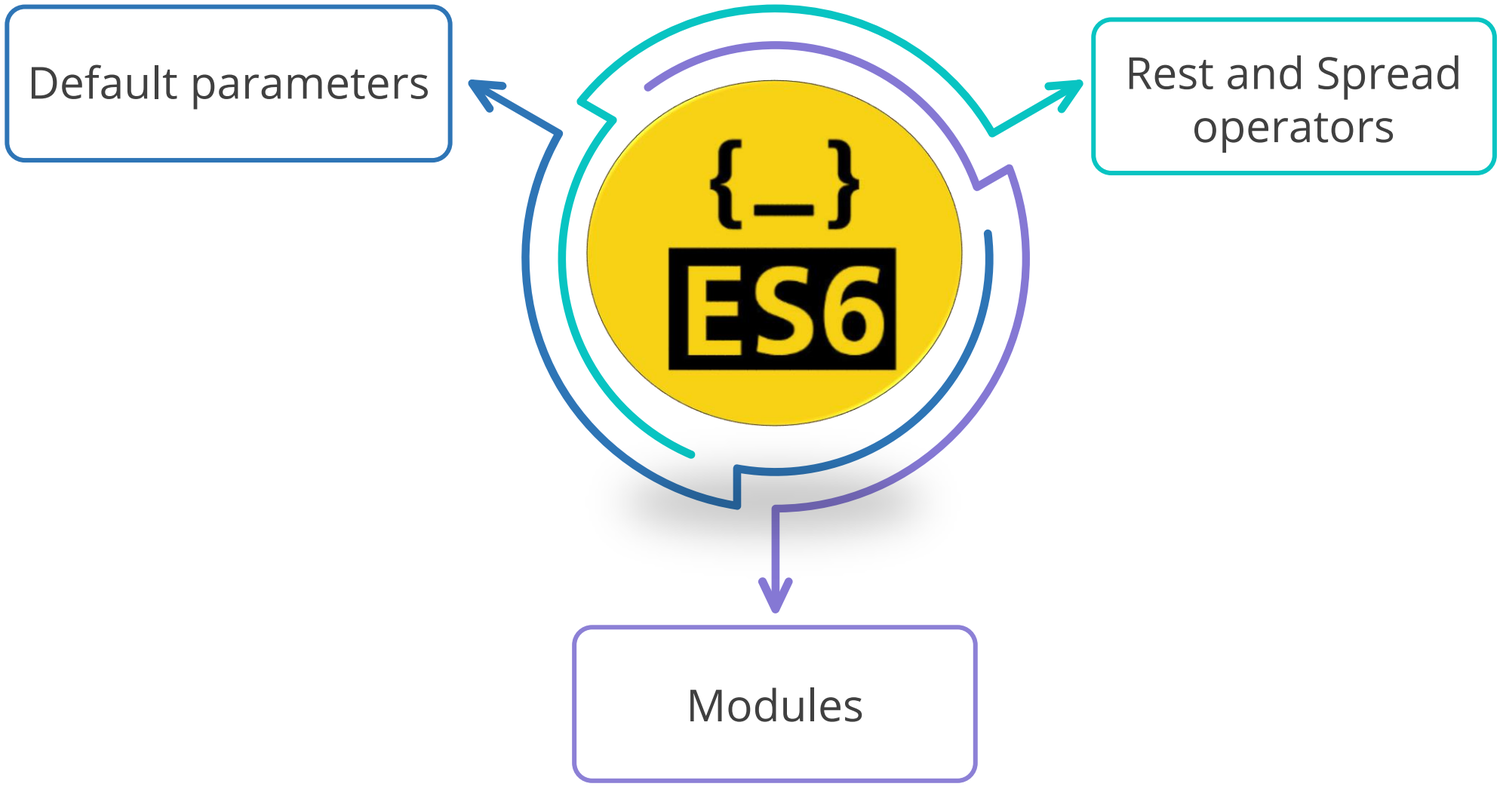
ES6: Features

ES6 (ECMAScript 2015) introduced several new features and enhancements to JavaScript. Here's some important key ES6 features:

- let and const
- Arrow functions
- Template literals
- Destructuring assignment
- Classes
- Promises

ES6: Features

ES6 (ECMAScript 2015) introduced several new features and enhancements to JavaScript. Here’s some important key ES6 features:



Template Literals

Template literals are literals delimited with backtick(`) characters, allowing multi-line strings.

- String interpolates with embedded expressions and particular constructs called tagged templates.
- They are informally called template strings as they are used for interpolation.

Template Literals: Syntax

```
`string text`
`string text line 1 string text line 2`
`string text ${expression} string text`
`tagFunction `string text ${expression} string text`
```

Call() Method

Call() method uses the owner object as an argument when calling the method.

The keyword is used to identify the function's or object's owner.

```
object.objectMethod.call( objectInstance, arguments )
```

Users invoke a method that may be applied to various objects.

Call() Method: Example

Code to call a function:

```
var dish = {
    name: 'Noodles',
    price: 200,
    getDishDescription: function() {
        var description = this.name + ' ' + this.price;
        return description;
    }
},

var dishDescription = function(restaurant) {
    console.log(this.getDishDescription() + ' is served by ' + restaurant);
};

dishDescription.call(Dish, 'Table by Basant');
```

Call() Method: Properties

Call() method has the following properties:

01

Accepts additional parameters

02

Executes the function it was called upon

03

Does not make a copy of the function it is being called on

Apply() Method

Apply() method is used to write methods that can be used on different objects.

Unlike this method, the call() method does not accept an array as an argument.

Syntax:

```
object.objectMethod.apply(objectInstance, arrayOfArguments)
```

Apply() Method: Example

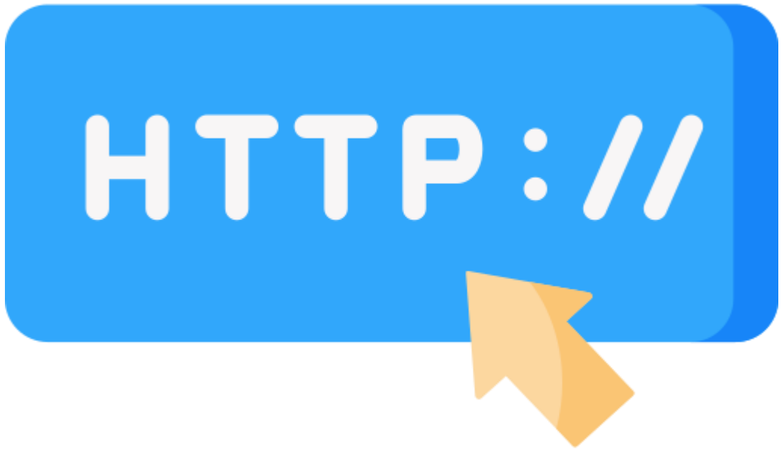
Apply() expects an array of all the parameters:

```
var dish = {
    name: 'Noodles',
    price: 200,
    getDishDescription: funciton() {
        var description = this.name + ' ' + this.price;
        return description;
    }
};

var dishDescription = function(restaurant) {
    console.log(this.getDishDescription() + 'is served by ' + restaurant);
};

dishDescription.call(dish, ['Table By Basant', 'Kylin', 'China Canteen']);
```

Introduction to Sever-Side Programming

HyperText Transfer Protocol

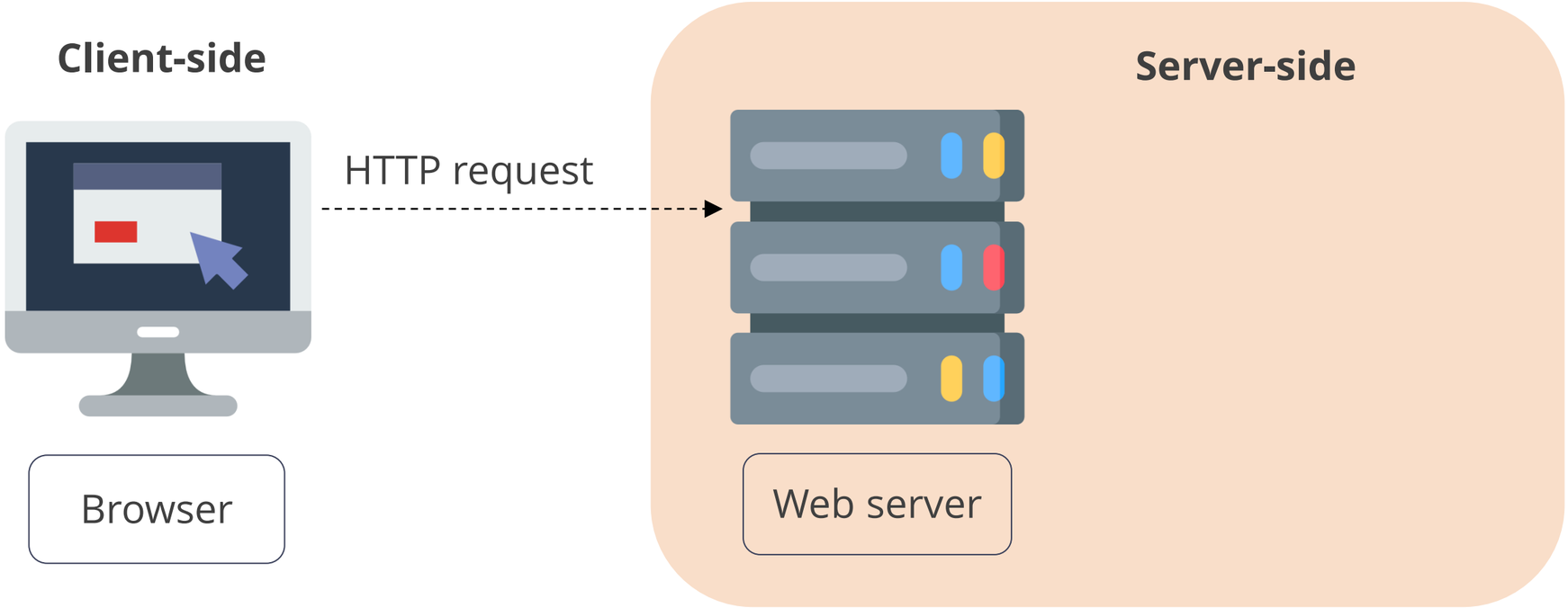
HyperText is the type of text specially coded with the help of a standard coding language called HTML.



Web browsers use the HyperText Transfer Protocol (HTTP) to communicate with web servers.

Working of HTTP

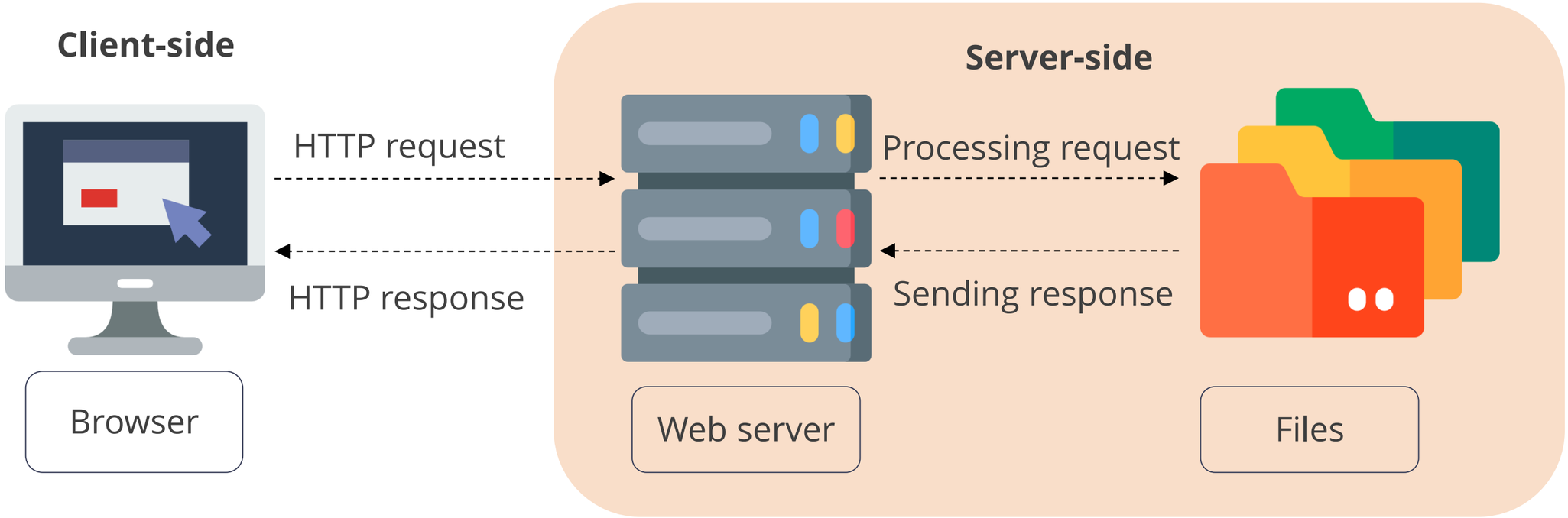
When a web page is clicked on, an HTTP request is sent from the browser to the server.



The request includes a URL identifying the affected resource and a method that defines the required action.

Working of HTTP

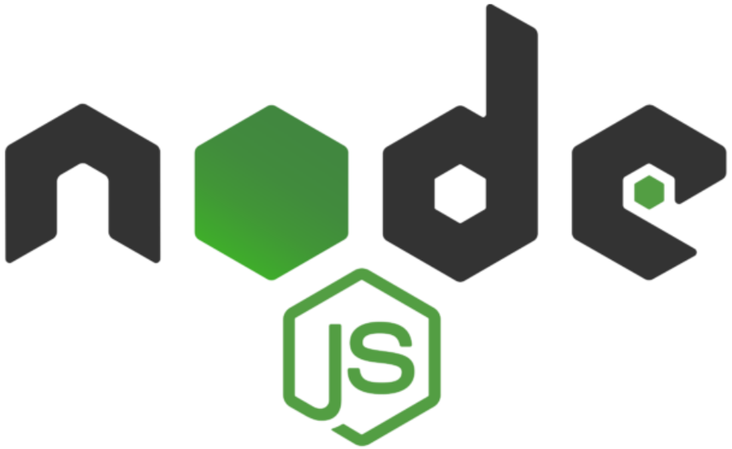
The web server processes the client request messages and sends a response to the web browser.



HTTP Requests

HTTP Requests

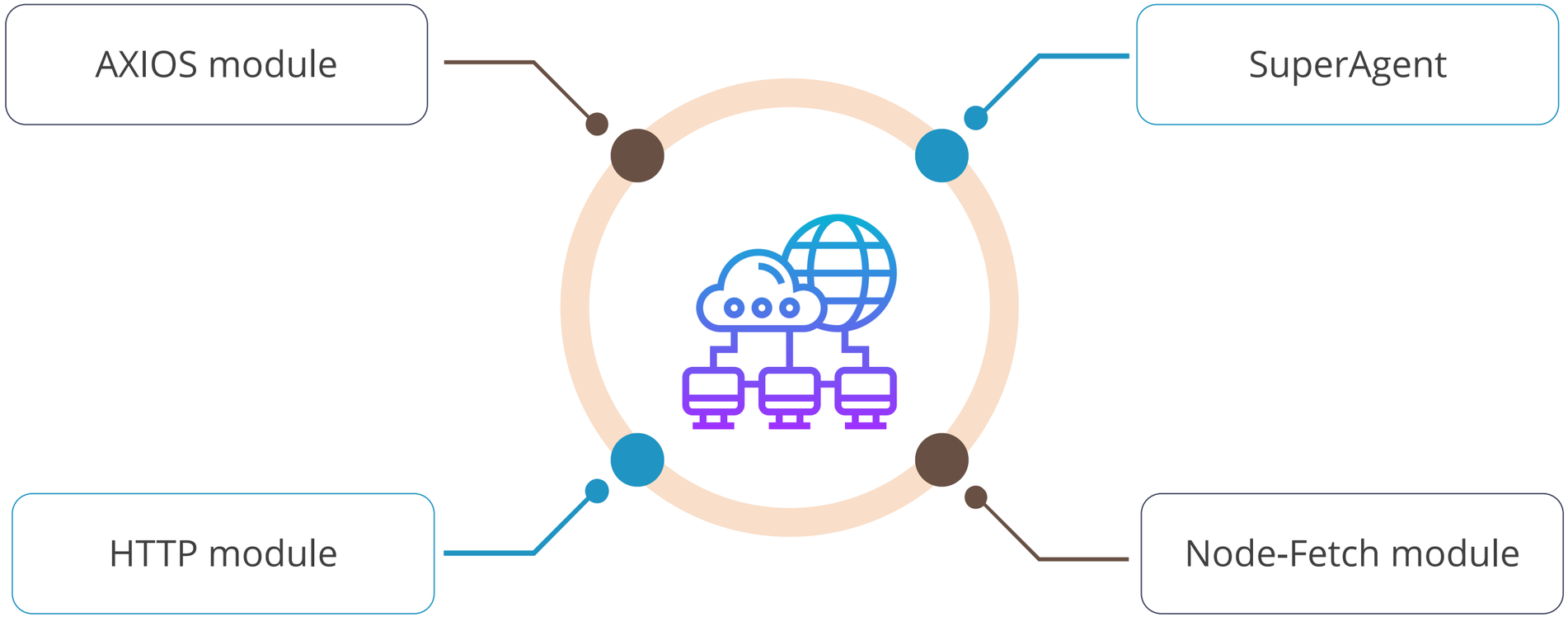
HTTP requests are a core functionality of modern technology.



HTTP Node.js module can be used to make a network request.

Network Requests

Network requests can be created using:



HTTP Methods

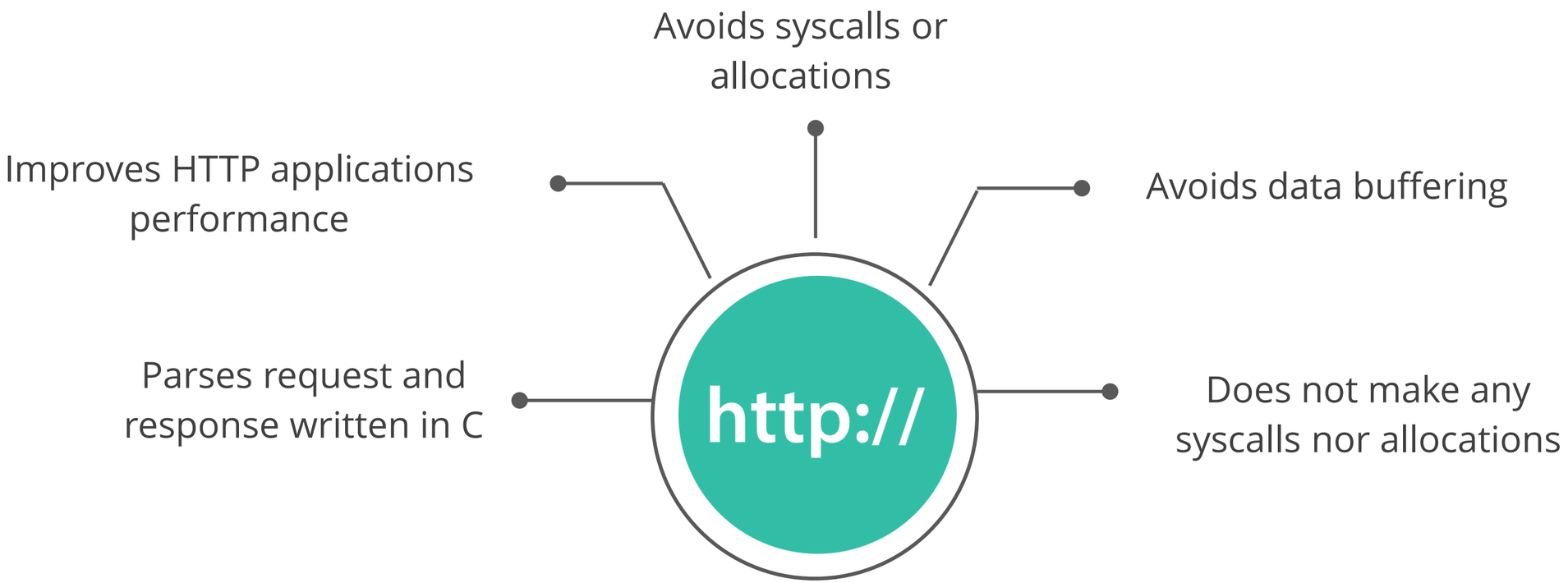
HTTP defines a set of request methods to define the actions that a user can perform on a given resource.

Method	REST API	Description
GET	/posts	Listing all resources
GET	/posts/<id>	Getting a resource
POST	/posts	Creating a resource
PUT	/posts/<id>	Updating a resource

HTTP Parser

HTTP Parser

HTTP parser is a library that parses HTTP protocol for requests and responses.



HTTP Parser: Features

01 Is independent

02 Handles persistent streams

03 Decodes chunked encoding

04 Supports upgrade

05 Defends against buffer overflow attacks

HTTP Parser: Example

HTTP parser is packaged as a standalone npm module for the **node monkeypatch** HTTP parser.

Example:

```
process.binding('http_parser').HTTPParser = require('http-parser-js').HTTPParser;
var http = require('http');
```

Handling Errors

Handling Errors

Handling errors refer to the error object in JavaScript.

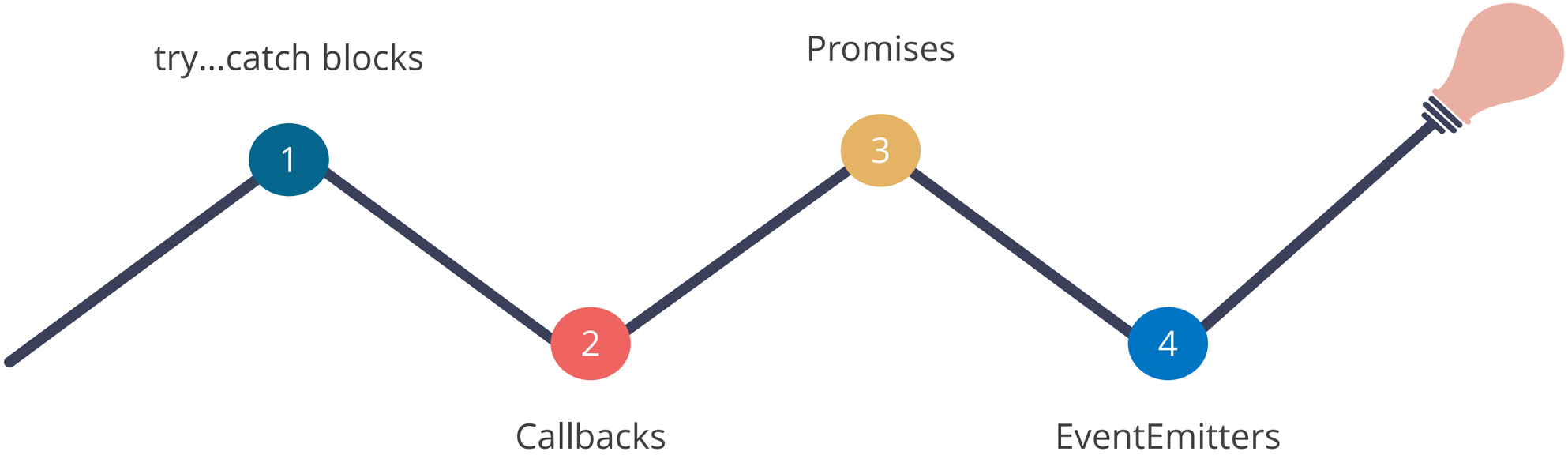
It can be constructed and thrown to other functions.

Example:

```
throw new Error('Resource Not Found'); // throwing new error
 some_function(new Error('DataBase Connection Refused')); // passing error as an argument
```

Techniques to Handle Errors

Techniques to handle errors in Node.js are:



try...catch Method

The try...catch method surrounds the code where the error(s) can occur, along with the catch block to handle exceptions.

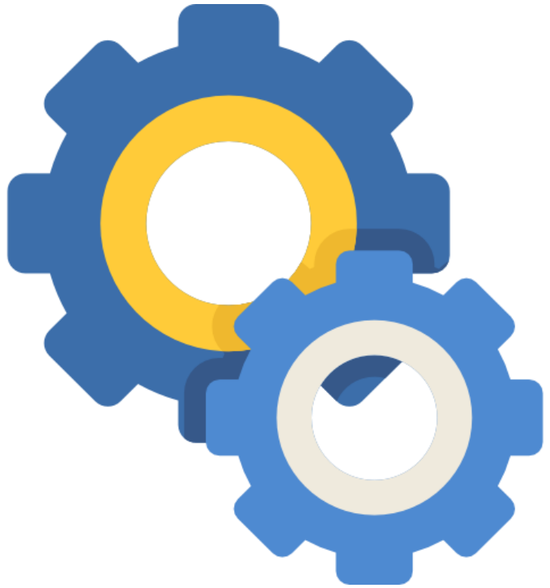
Example:

```
var fs = require('fs')
try {
const data = fs.readFileSync('/Users/Documents/dishes.csv')
} catch (error) {
  console.log(‘Something Went Wrong: ’+error)
}
```

Callback Functions

Callback Functions

Callback function is a function that is called when a task is done.



01 Prevent any kind of blocking

02 Allow other code to run

03 Perform like a synchronous function

Callback Functions: Usage

Using this, Node.js can process many requests without waiting for any function to return the result, thus increasing its scalability.

An asynchronous way to read the file using a callback:

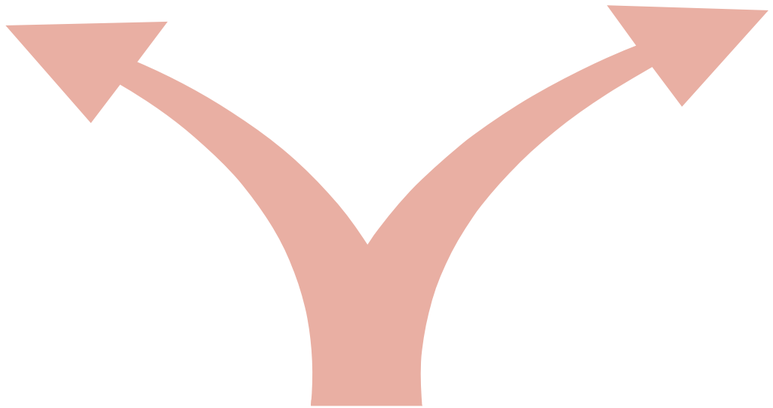
```
var fs = require("fs");

fs.readFile('dishes.csv', function (err, data) {
   if (err) return console.error(‘Something Went Wrong: ’+err);
   console.log(data.toString());
});
```

Callback Chains

Callback Chains

Callback chaining in Node.js involves nesting callbacks within each other to execute asynchronous operations sequentially.

async.parallel(tasks, callback)



async.series(tasks, callback)

The async module in JavaScript offers these two fundamental methods for controlling the flow of asynchronous functions:

Callback Chains Methods

async.parallel(tasks, callback)

- Functions run parallel through I/O switching.
- The callback function is fired if any function in the collection tasks returns an error.

async.series(tasks, callback)

- Functions in tasks run only after the previous function is completed.
- If any of the functions throw an error, the callback is triggered with the corresponding error value, and the subsequent functions are not executed.

HTTP Requests and Callback

Problem Statement:

Duration: 20 min.

You are given a task to work with HTTP Request and Callback.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a simple HTTP request in JavaScript
2. Create a callback for the HTTP request

HTTP Parsing

Problem Statement:

Duration: 20 min.

You are given a task to work with HTTP Parser.

Assisted Practice: Guidelines

Steps to be followed:

1. Create an HTTP parser for an HTTP request

TCP/IP

Protocols

Protocols are set of rules for message formats and procedures that allow machines and application programs to exchange information.

They establish the framework and message formats, enabling seamless communication and interoperability among diverse systems.

TCP/IP

Transmission Control Protocol/Internet Protocol is a set of networking protocols that allows different computers to communicate and share resources across interconnected networks.

01

It is an efficient version of the OSI model.

02

It carefully defines how information moves from sender to receiver.

Layers of TCP/IP

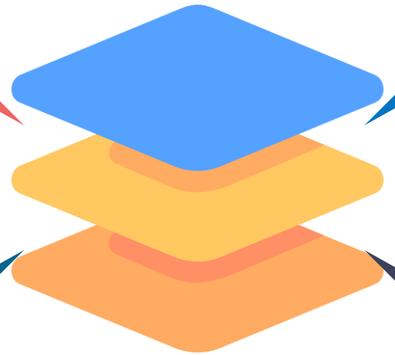
The four layers of TCP/IP are:

Network access layer
It is the combination of the data link layer and the physical layer of the OSI model.

01

02

Internet layer
It defines the protocols that help in the logical transmission of data.



Application layer
It helps in node-to-node communication and controls the user interface.

04

03

Host-to-host layer
It manages end-to-end communication and ensures that the data delivery is error-free.

Addresses and Ports

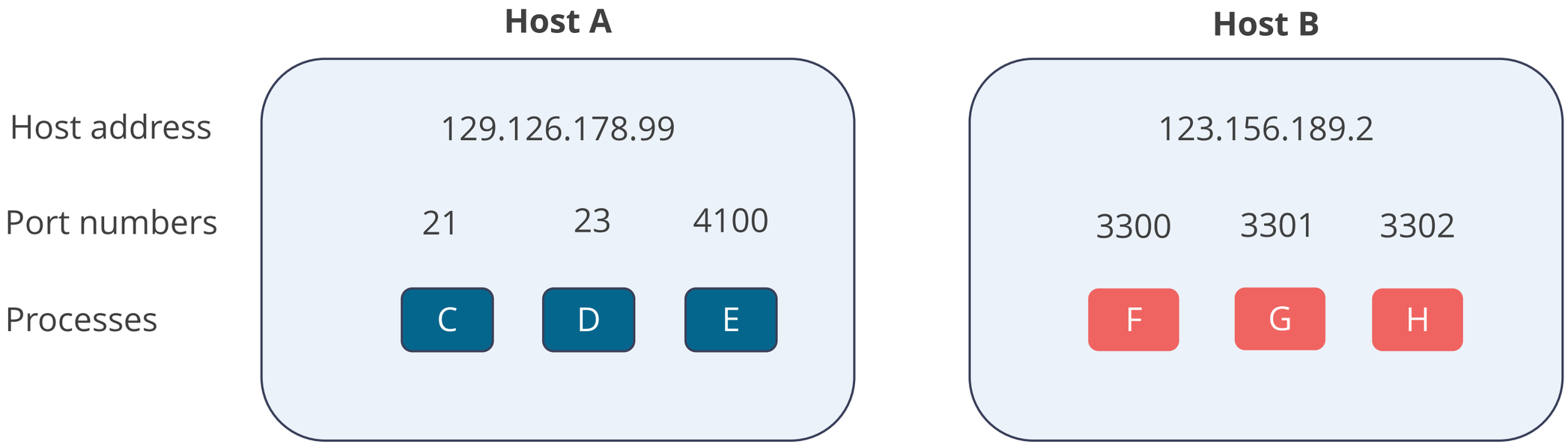
Addresses and Ports

TCP/IP provides process-to-process communication.

The host is specified by an internet address, and it is processed by a port number.

Addresses and Ports

This following figure depicts how the applications are addressed within a server:



TCP/IP Internet: Types

Each server or client is identified by a numeric IP address.

IP addresses are of two types:

IPv6 addresses

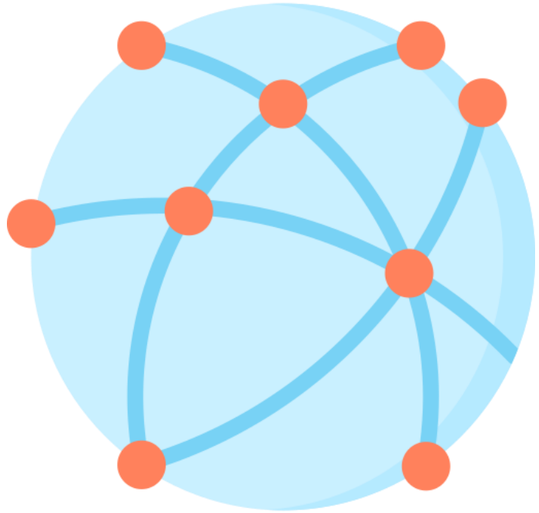
- They are 128-bit addresses, usually expressed in hexadecimal notation
- Example: 2001:db8:3333:4444:5555:6666:7777:8888

IPv4 addresses

- They are 32-bit addresses, usually expressed in dotted decimal notation
- Example: 192.0. 2.146

Port Number

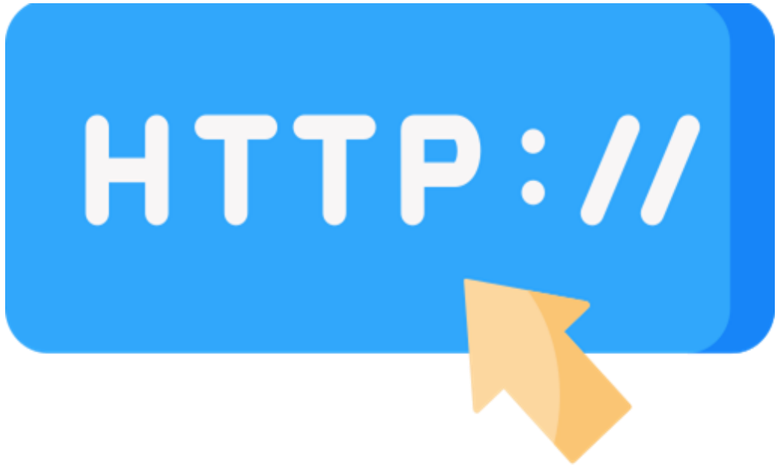
Port number helps to identify a process by which an internet or other network message gets forwarded when it arrives at a server.



All network-connected devices come equipped with standardized ports assigned by a number.

Port Number

Port numbers are reserved for certain protocols and their associated function.



Port numbers range from 0 to 65535. HTTP messages always go to port 80.

Classification of Port Numbers

Port numbers can be classified into the following types:

Port numbers for servers

1. Well-known ports (0-1023):
   - Reserved for established services
   - Examples: HTTP (port 80), HTTPS (port 443).

2. Registered ports (1024-49151):
   - Assigned for specific services but not standardized like well-known ports

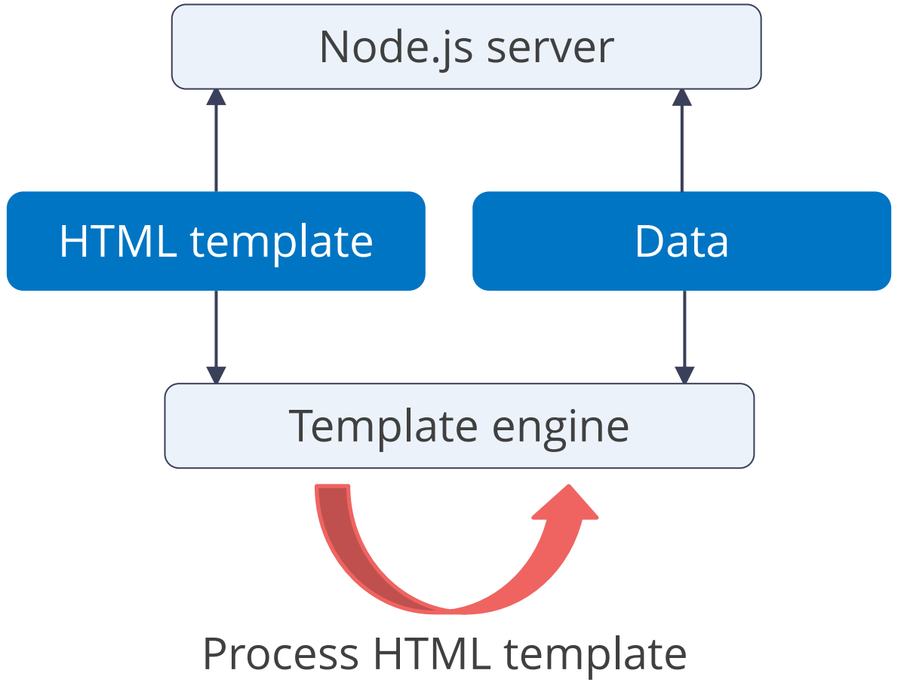
Port numbers for clients

1. Dynamic or private ports (49152-65535):
   - Available for temporary use by client programs
   - Allocated dynamically for different applications

HTML and Templates

Template Engine

Template engine helps to create HTML templates.



It injects data into the HTML template (client side) and displays the final HTML.

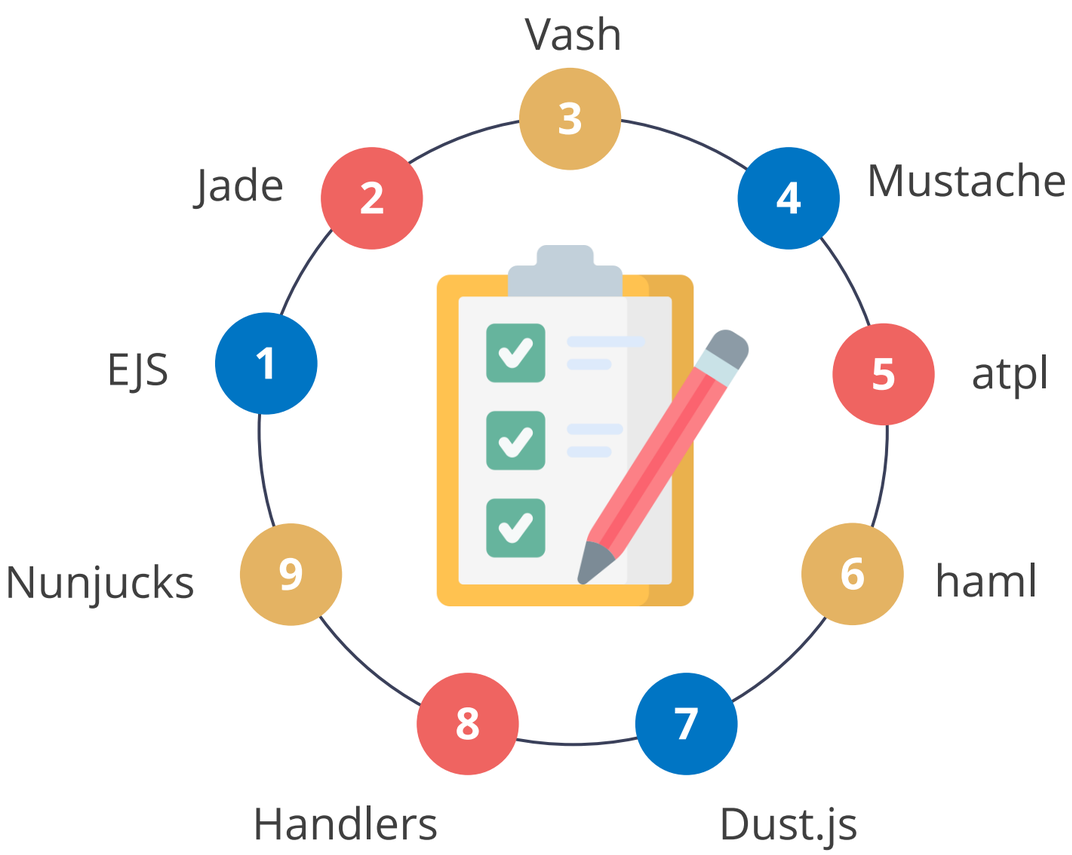
Template Engine in Node.js

Node.js offers a variety of template engines to create HTTP templates. Each template engine employs a different language.

Choosing a template engine often depends on factors like personal preference, project requirements, ease of use, and the desired balance between features and simplicity.

Template Engine in Node.js

The most used template engines in Node.js are:



Advantages of Template Engine

01 Enhances performance

02 Employs single template for multiple pages

03 Provides access to template from CDN

04 Improves productivity

05 Improves readability

06 Maximizes processing on client-side

Template Engine: Example

Here's an example of an HTML page using EJS as the template engine:

index.ejs

```
<html>
<head>
    <title>Welcome to My Page</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            background-color: #f0f0f0;
        }
        h1 {
            color: #333333;
        }
    </style>
</head>
<body>
    <h1>Welcome, <%= username %>!</h1>
    <p>This is an EJS-powered HTML page.</p>
</body>
</html>
```

When rendering this template, the <%= username %> will be replaced with the value of the username variable provided by the EJS engine.

Template Engine: Example

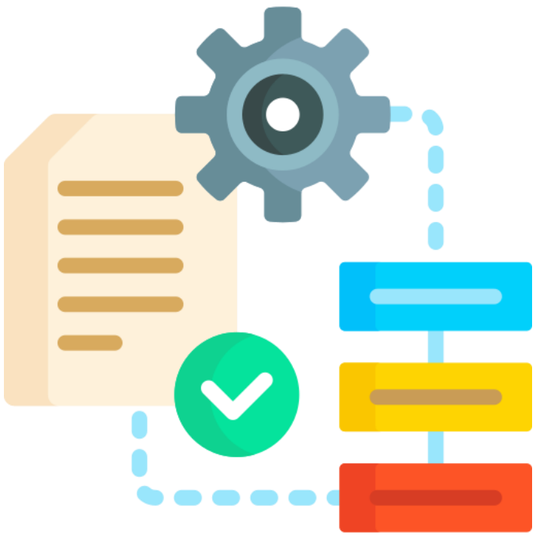
To use EJS with Node.js, pass data to this template to render it. For instance:

```
const express = require('express');
const app = express();
const ejs = require('ejs');

app.set('view engine', 'ejs');

app.get('/', (req, res) => {
    res.render('index', { username: 'Joe' });
});

app.listen(3000, () => {
    console.log('Server is running on port 3000');
});
```

Streams

Streams

Streams in Node.js handle data in chunks, improving memory efficiency. They are capable of processing large volumes of data.



They enhance performance and scalability of applications by facilitating asynchronous, non-blocking I/O operations.

Readable Stream

Data is read from a readable stream by listening to data events emitted by the stream. Here's an example demonstrating how data can be read from a readable stream by listening to data events emitted by the stream:

Example:

```
const fs = require("fs");

const readableStream = fs.createReadStream("./dishes.csv", { highWaterMark: 20 });

readableStream.on("data", (chunk) => {
  console.log(`Chunk Size: ${chunk.length} bytes`);
  console.log(`Data in Chunk: ${chunk.toString()}`);
});
```

Writable Stream

Data is sent using the write() method, enabling the transmission of chunks to the stream destination. Here's an example demonstrating how data can be written to a writable stream:

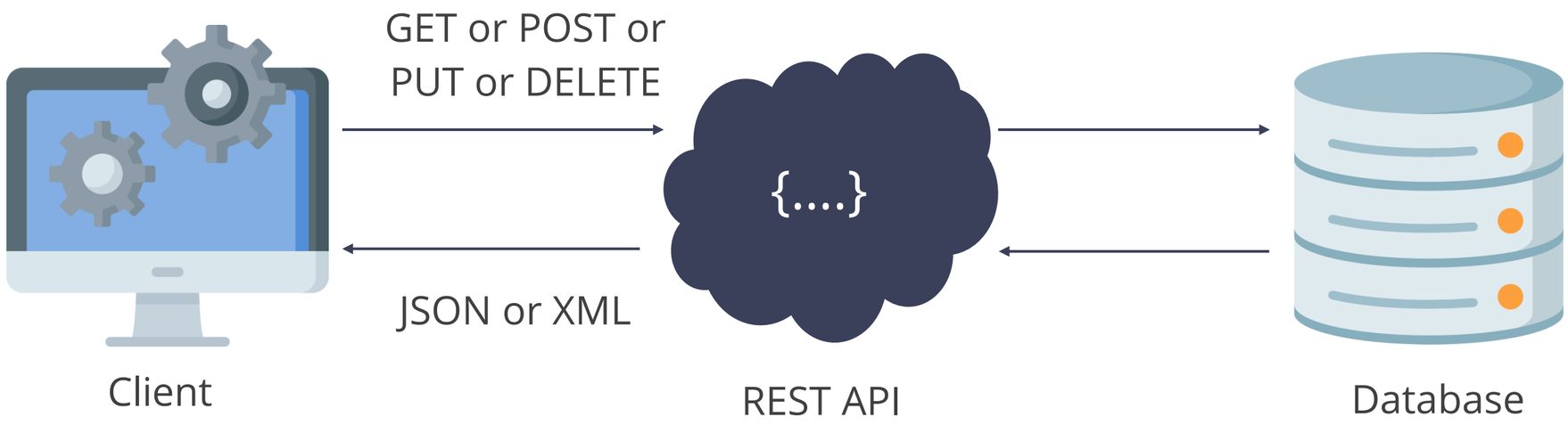
Example:

```
const { Writable } = require('stream’);

const writableStream = new Writable({
  write(chunk, encoding, callback) {
    console.log(`Writing: ${chunk.toString()}`);
    callback(); // Signal that the chunk has been processed
  }
});
writableStream.end(() => {
  console.log('Stream ended.');
});
```

APIs and Endpoints

Application Programming Interface (API)

API helps computer programs to communicate with each other.



It sends requests for information from a web application or a web server and receives a response.

API Endpoints

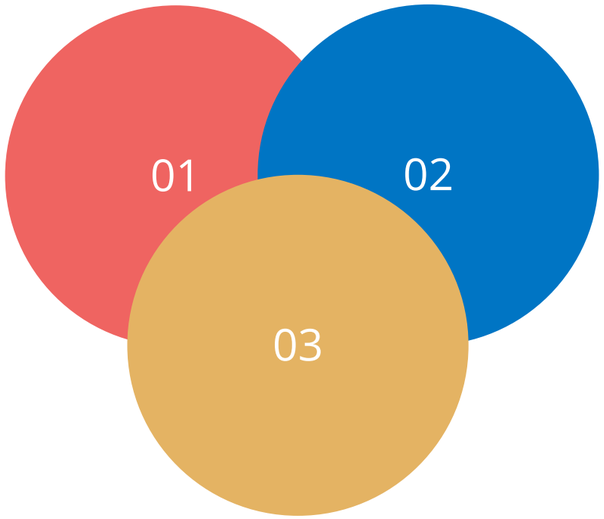
Endpoints are specific URLs within an API that perform functions or provide access to resources.

An API's performance depends on its capacity to communicate successfully with API endpoints.

API Endpoints: Requirements

They enable communication between different systems, allowing data exchange and functionality access. For an effective request to be processed by the endpoint, the client must provide:

Uniform Resource Locator (URL)



Method

List of headers and a body

APIs in Node.js

Problem Statement:

Duration: 20 min.

You are given a task with work with APIs in Node.js.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Install the fetch API package
2. Create a Node.js app to fetch users

Routing

Routing

Routing defines how the application endpoints handle client requests. It is essential for organizing and managing web application navigation and functionality.

Ways to implement routing in Node.js:

With framework

ex

Using a framework like Express.js simplifies routing and provides more features to handle requests and responses efficiently. It has an **app object** that corresponds to HTTP.

Routing

Ways to implement routing in Node.js:

Without framework

When working without a framework, users can manually handle HTTP requests and route them to the appropriate handlers based on the URL and HTTP method.

HTTP Routing

Problem Statement:

Duration: 15 min.

You are given a task to perform HTTP Routing.

Assisted Practice: Guidelines

Steps to be followed:

1. Create HTTP routes to handle incoming requests

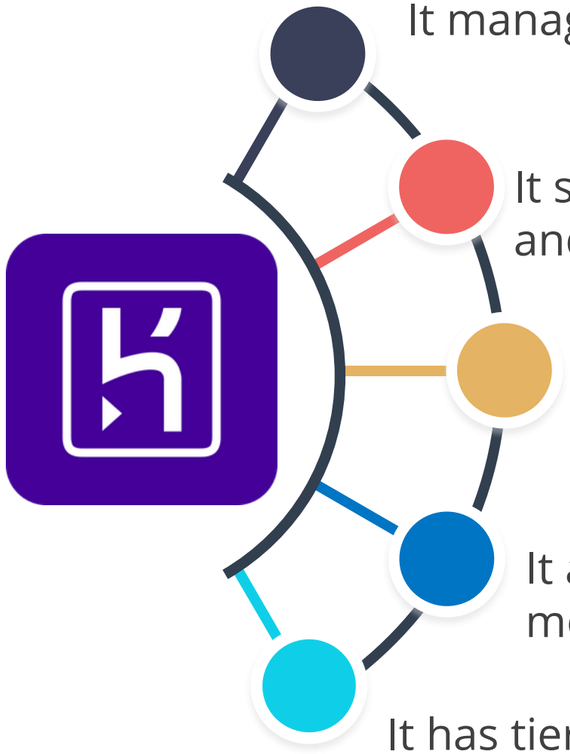
Heroku

Heroku

Heroku is a cloud-based platform for deploying, managing, and scaling applications. It is an AWS-based service provider which is simpler to use than Elastic Compute Cloud.

- It is popular because of its add-on capabilities such as alerts and management tools.
- It is owned by Salesforce.com.

Features of Heroku



It manages hardware and servers by providing Platform-as-a-Service (PaaS).

It supports Node.js applications, providing a platform to host and run them.

It offers integration with GitHub, enabling automatic deployment of code changes pushed to specific branches.

It allows scaling the application dynamically based on traffic and provides monitoring tools to track application performance.

It has tiered service packages to address the needs of complex business.

Advantages and Disadvantages of Heroku

Some of the benefits and limitations of Heroku are:

Advantages

01 Easy to setup

02 Easier to scale

03 Great plugin support

Disadvantages

01 Pricey

02 Low performance

03 Single point of failure

Git and GitHub

Git

Git is an open-source distributed version control system created by Linus Torvalds in 2005.

Steps to install Git:

- Install Git on Linux
  - From the user shell, install Git using apt-get:

```
$ sudo apt-get update
$ sudo apt-get install git
```

- Verify the installation by using the below command:

```
$ git --version
```

- Configure user's Git username and email using the following commands:

```
$ git config --global user.name "John Watson"
$ git config --global user.email "john@example.com
```

Features of Git

Some of the important features of Git are:

- It stores the entire history of the codebase developed by the developers.
- It allows easy branching and merging of the codes.

GitHub

GitHub is a cloud-based service that helps to track, control, and manage codes.

Some basic terminologies are:

Repository(Repo)	It is the database that stores files.
Server	It is the computer that stores the repo.
Client	It is the computer that connects to the repo.
Working Set/Copy	It is the local directory of files.
Trunk/Main	It is the primary location for code in the repo.

Advantages of GitHub

Some of the basic advantages of GitHub are:

Constant notifications

Portfolio showcase

Cross-platform compatibility

Offline work

Version tracking

GitHub

Version Control System

Version Control System

Version control system is a software that helps the developer team to:

02

Track what changes have been made to the source code, and who made them

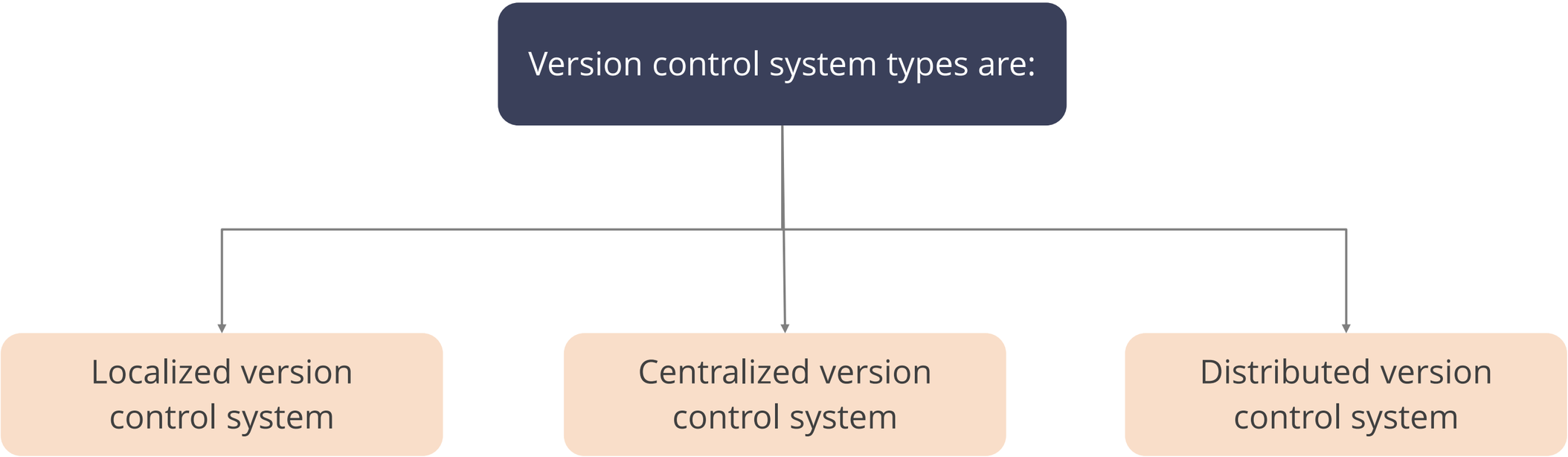
01

Record changes in the source code

03

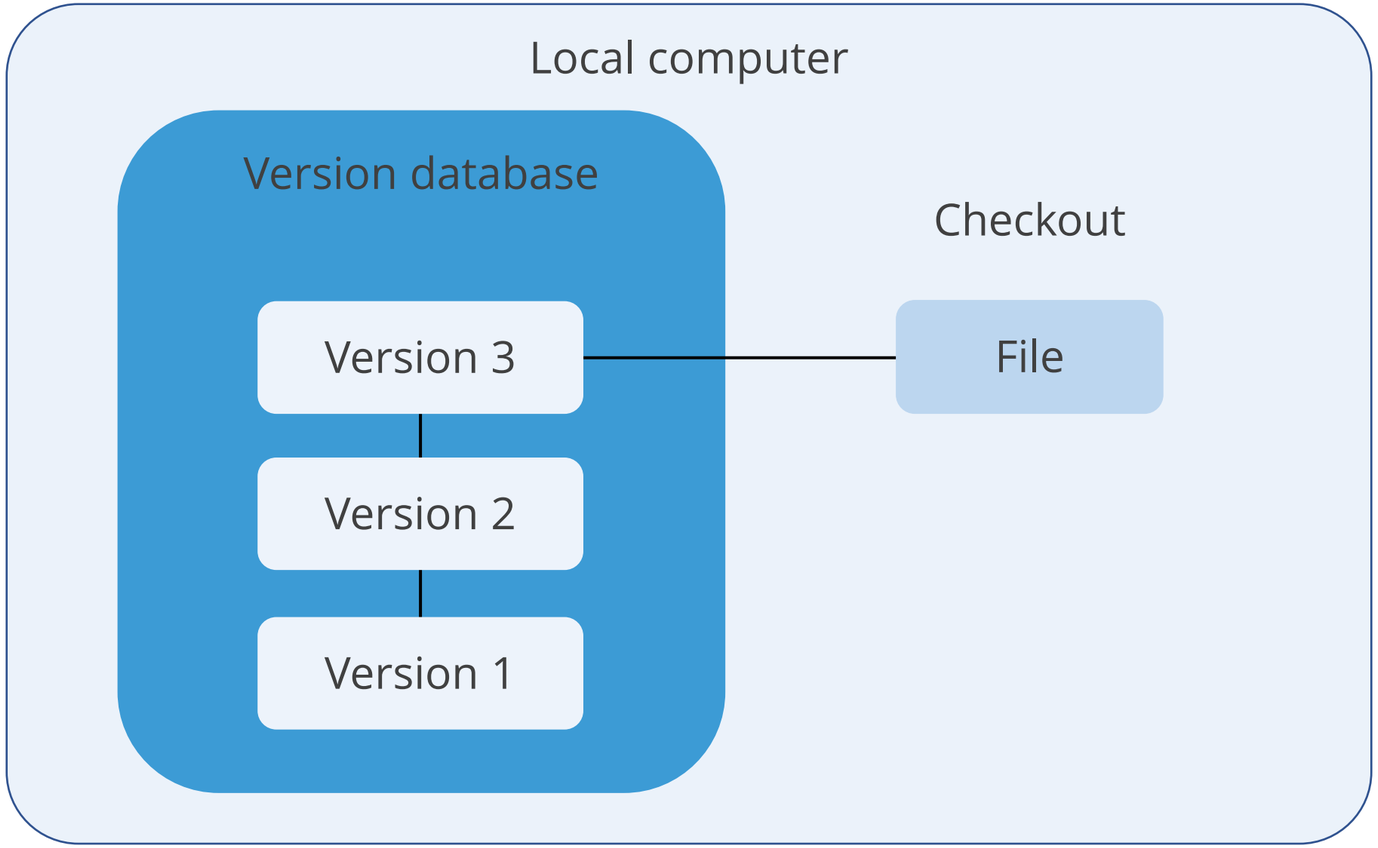
Communicate efficiently across the team

Types of Version Control System



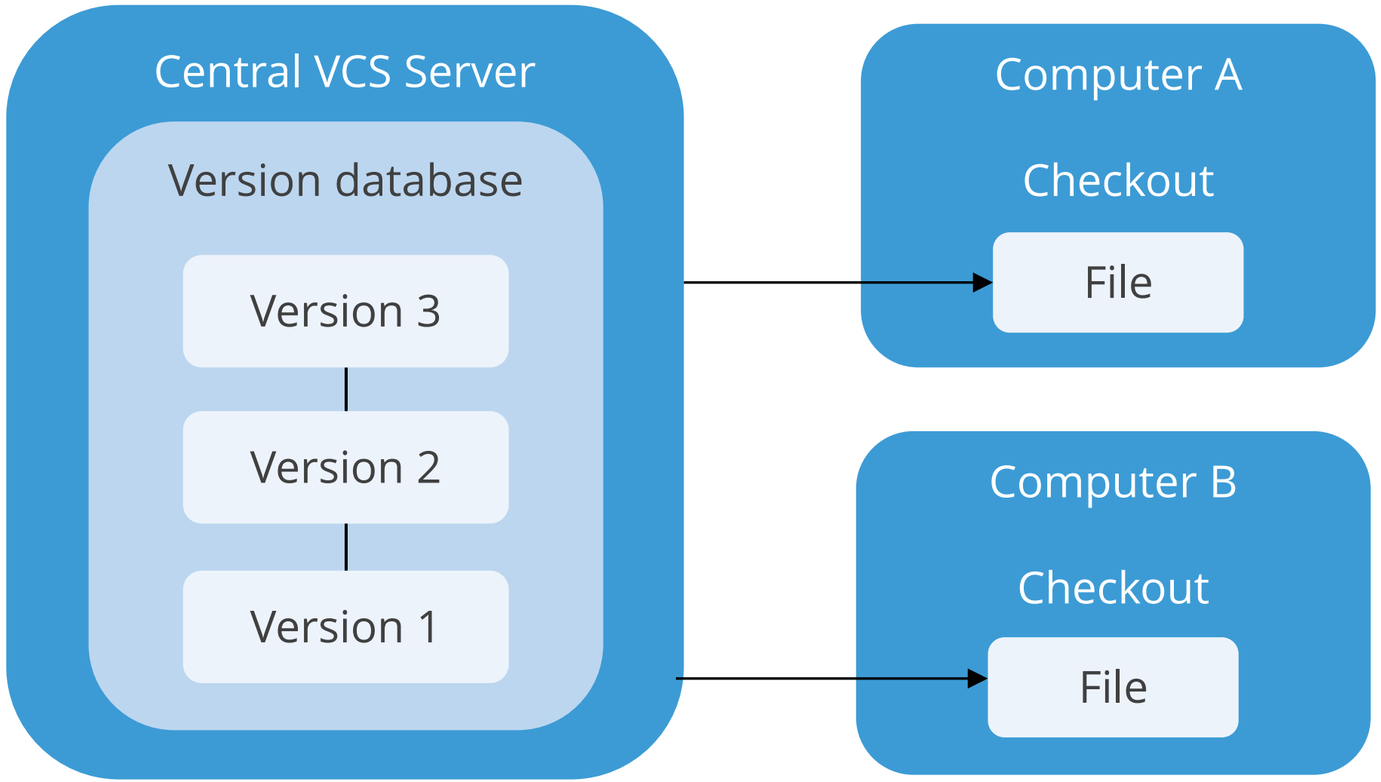
Localized Version Control Systems

A localized version control system is a local database in which every file change is stored as a patch.



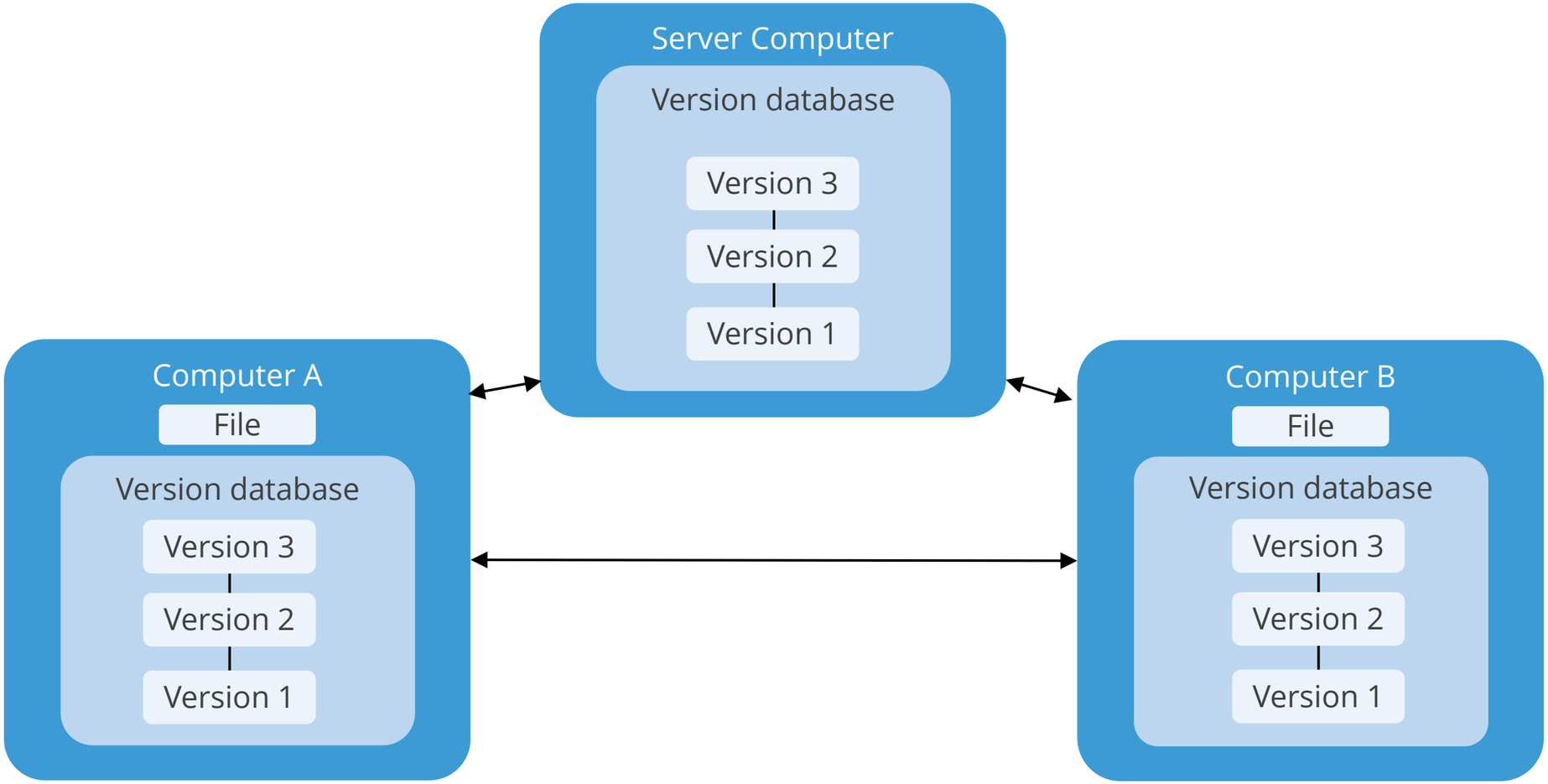
Centralized Version Control Systems

A centralized version control system enables multiple clients to access files simultaneously as it contains all file versions in a single server.



Distributed Version Control Systems

A distributed version control system allows user to have local copy of a repository and facilitates branching and merging of files.



Features of Version Control System

1. Allows the users to have **versions** of a project
2. Allows the user to backtrack if necessary and undo the changes
3. Compares two versions or reverses the changes
4. Changes can be saved just in time
5. Stores all version on a central server

Benefits of Version Control System

01

Enhances the project development speed by providing efficient collaboration

02

Leverages productivity through better communication and assistance

03

Helps contributors to contribute irrespective of the different geographical locations

04

Reduces the possibilities of errors and conflicts by tracking more minor changes

Deployment Node.js Applications Using Heroku

Problem Statement:

Duration: 20 min.

You are asked to employ Heroku to develop and deploy node.js applications.

Assisted Practice: Guidelines

Steps to be followed:

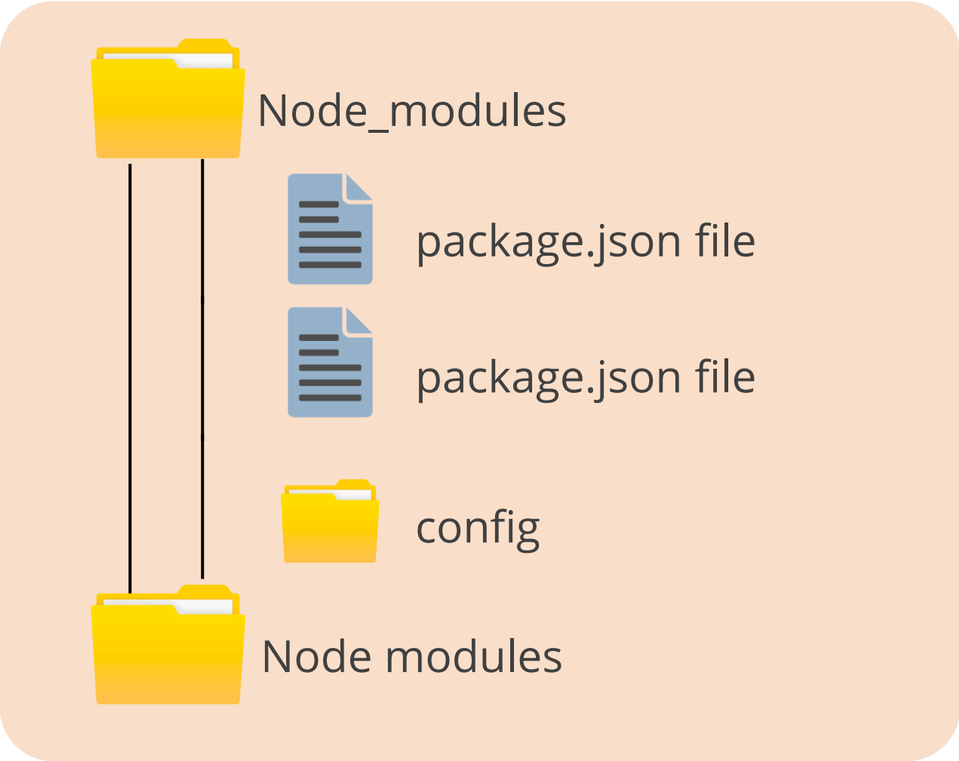
1. Create a Node.js app on a local machine
2. Push the code into the GitHub remote repository
3. Deploy the app to Heroku

Packages and Package Managers

Packages

A package in Node.js contains all the files needed by a user for a module. They are pre-built code modules or libraries that offer specific functionalities.

Following is a folder tree described by a **package.json** file:



package.json

package.json file resides in the center of the Node.js system.

It contains the metadata of any Node.js project.

Structure of the file:

```
{
   "name": "MyApplication",
   "version": "0.0.1",
   "description": "MyApplication",
   "main": "app.js",
   "author": {
     "name": "John",
     "email":  "john@example.com"
   },
   "dependencies": {
     "express": "^4.13.3"
   }
```

Information in package.json

The metadata information in package.json are:

Functional
metadata properties

Identifying
metadata properties

Package Managers

Package managers are used to automate the process of:

01 Installing programs

02 Upgrading programs

03 Configuring programs

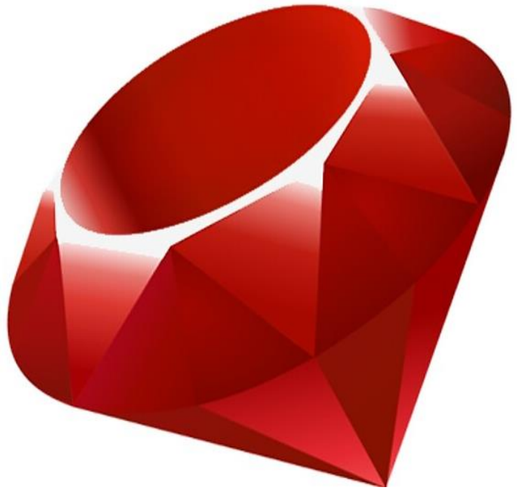
04 Removing programs

They are available for Unix or Linux-based systems.

Package Managers

Package managers are also used to install and manage modules for languages such as:

Python



Ruby

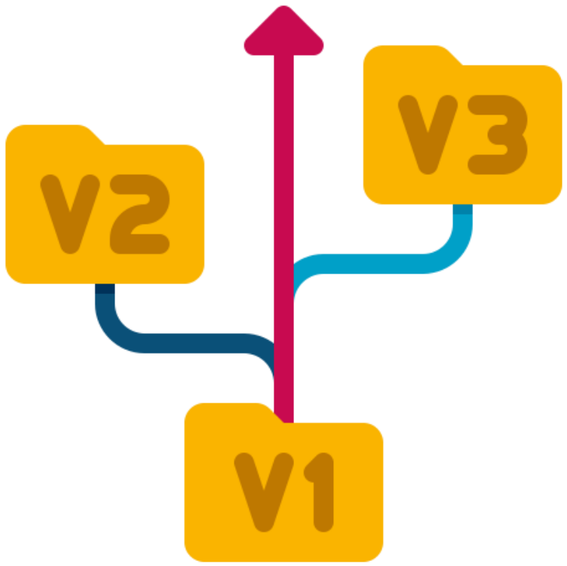
Functions of a Package Manager

- Extracts package archives through interaction with file archivers
- Examines digital certificates and checksums to ensure the integrity and authenticity of the package
- Uses an app store to install and update existing applications
- Reduces confusion by grouping packages by function
- Avoids dependency by ensuring a package is installed with all the necessary components

Semantic Versioning (semver)

Semantic Versioning (semver)

It is a versioning system specifying how version numbers convey meaning. **semver** is used when a new package version is published with a revised version number in the package.json file.



It helps to track the extent of changes made to the code in a version and modify them.

Incrementing Semantic Versioning

This type of versioning can be implemented in published packages:

Code Status	Stage	Rule	Example Version
First release	New product	Start with 1.0.0	1.0.0
Backward compatible bug fixes	Patch release	Increment the third digit	1.0.1
Backward compatible new features	Minor release	Increment the middle digit and reset last digit to zero	1.1.0

npm and npm Registry

npm

npm stands for node package manager for Node.js packages/modules.

01 npm is an online repository to publish Node.js projects.

02 It is a command-line service that helps in package installation and version management.

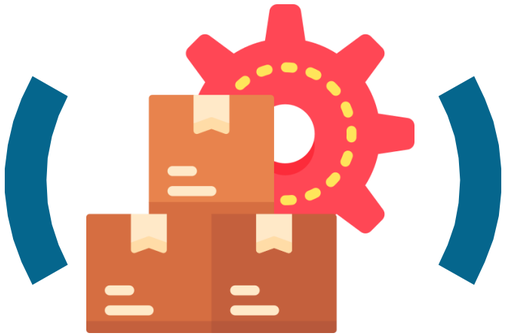
03 It can install the required package with a single-line command.

Public npm Registry

It is a public repository hosting packages for use in Node.js projects, accessible via npm. A public npm registry is a database of JavaScript packages.

It includes software and metadata that helps to:

Contribute packages



Download packages

npm view command is used to view the registry of specific packages.

init and nodemon

init

init is used to create **package.json** for the project.

Syntax to set up a new or existing npm package:

```
npm init <initializer>
```

The init command is transformed to a corresponding **npm exec** operation as follows:

Syntax:

```
npm init myapp -> npm exec create-myapp
```

Nodemon

Nodemon is a tool that aids in the automatic development of Node.js apps.

Nodemon relaunches the node application when the change in the file is detected in the directory.

It does not make changes to the code or development process.

It serves as the node's replacement wrapper.

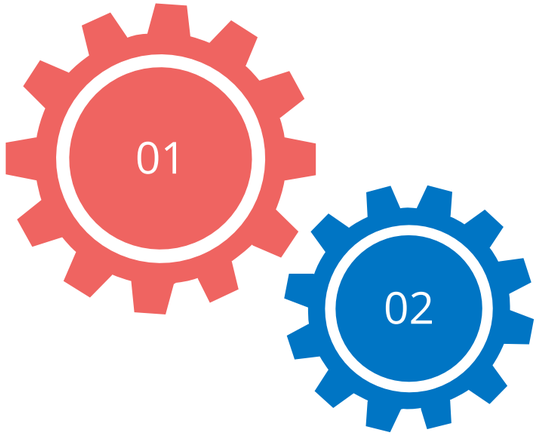
To use it, replace the word node on the command line while executing the script.
Example: nodemon your_script.js

npm Global Installation

npm Global Installation

npm global installation is the process of installing packages from the Node Package Manager (npm) into the global environment.

The global packages are installed in the **/user/local/lib/node_modules** folder.



-g is used to install commands in global package.

Example:

```
npm install -g express
```

Install Node Package Manager

Problem Statement:

Duration: 10 min.

You are given a task to install Node Package Manager.

ASSISTED PRACTICE

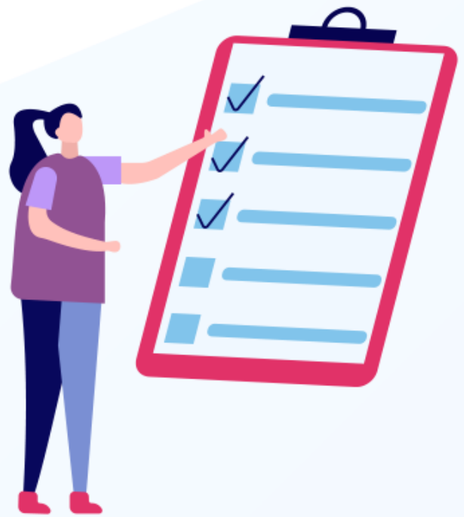
Assisted Practice: Guidelines

Steps to be followed:

1. Download Node.js
2. Install NPM

Key Takeaways

- The module.exports are used to export any literal, function, or object as a module.
- Exports are not returned by the require() function.
- A module is the object reference that gets returned from the require() calls.
- A protocol is a set of rules defining how data is transmitted and received, ensuring standardized communication between devices.
- TCP/IP addresses uniquely identifiable devices, while ports specify service endpoints, enabling effective data exchange in computer networks.
- Semantic Versioning(SemVer), is a versioning scheme for software that aims to convey meaning about the underlying changes in a consistent way.



Thank You