Unit 3 - Part 1
Node.Js

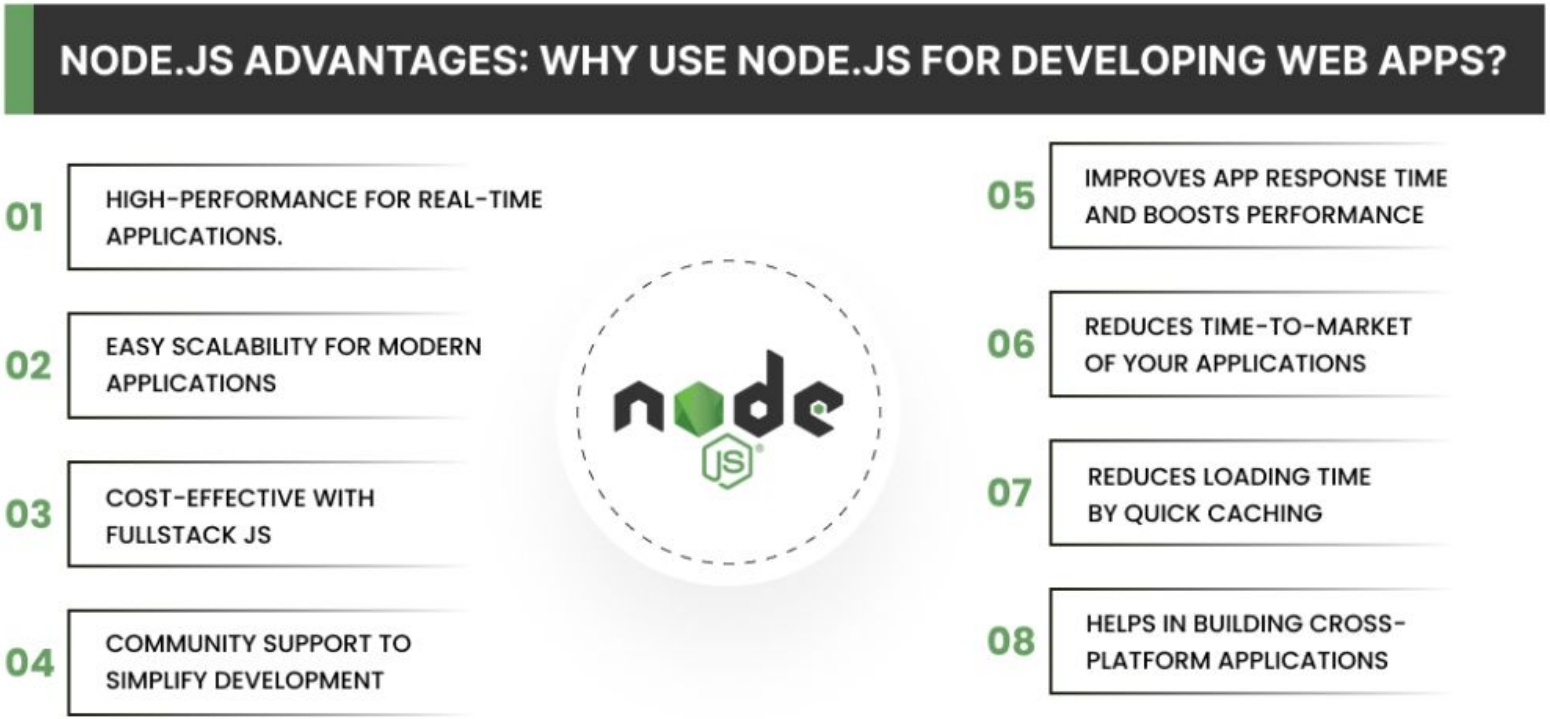
Introduction to Node.JS

- Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.
- Features
  - Asynchronous and Event Driven
  - Very Fast
  - Single Threaded but Highly Scalable
  - No Buffering
- Node.js = Runtime Environment + JavaScript Library

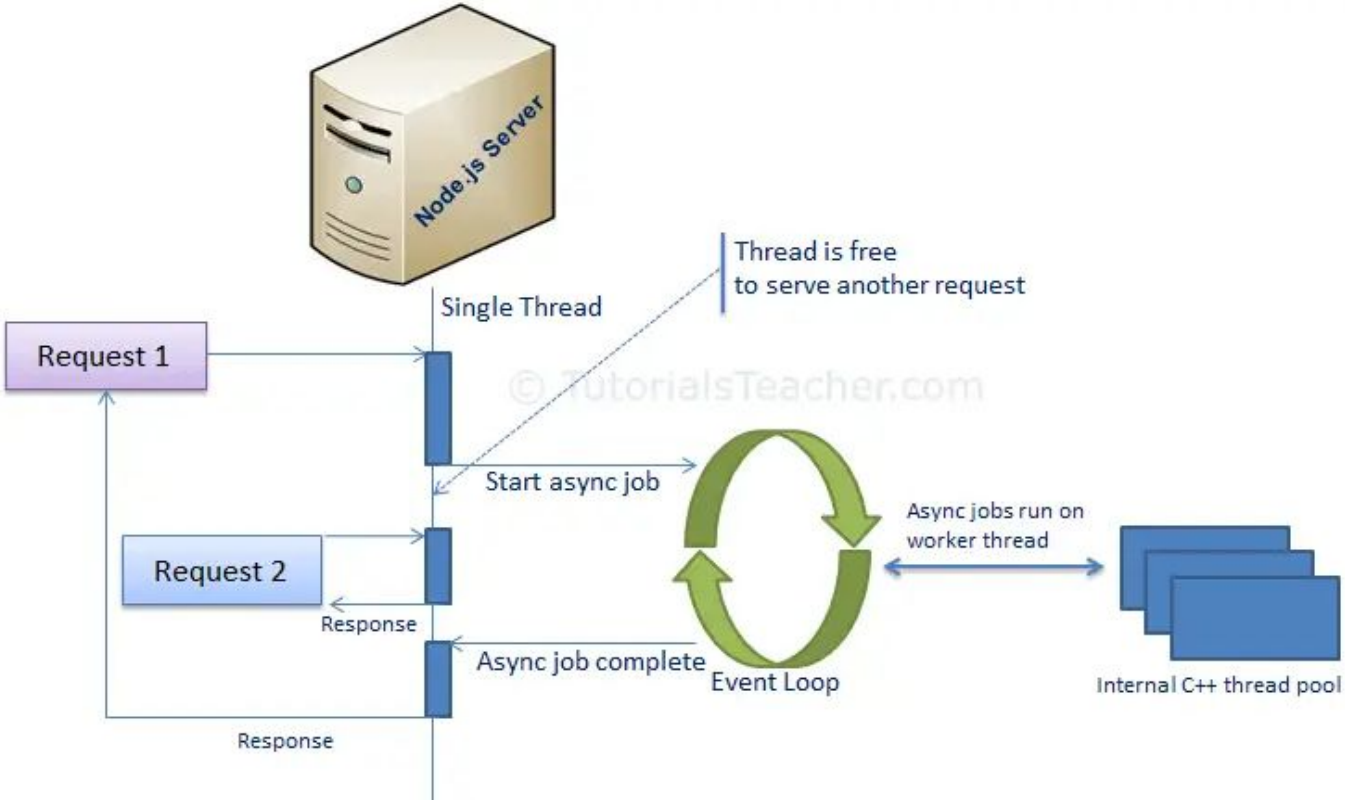
Where to use and where to not

- Where to Use Node.js?
  - I/O bound Applications
  - Data Streaming Applications
  - Data Intensive Real-time Applications (DIRT)
  - JSON APIs based Applications
  - Single Page Application
- Where Not to Use Node.js?
  - It is not advisable to use Node.js for CPU intensive applications.

Advantages of Node.JS



Process model of Node.JS



Node.JS

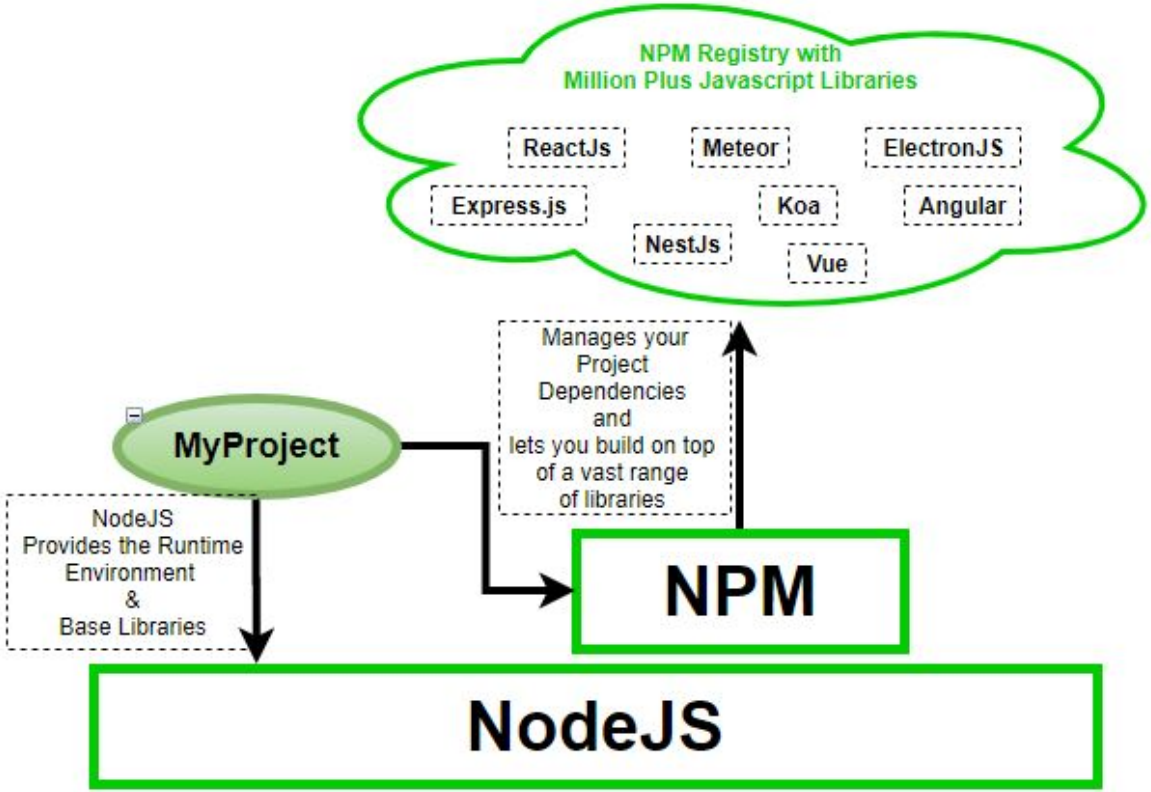
- Sample program
- Parts of a Node.JS program
  - Import the models
  - Create server
  - Read request, return response
- Sample programs to create server

Node.JS Built In modules

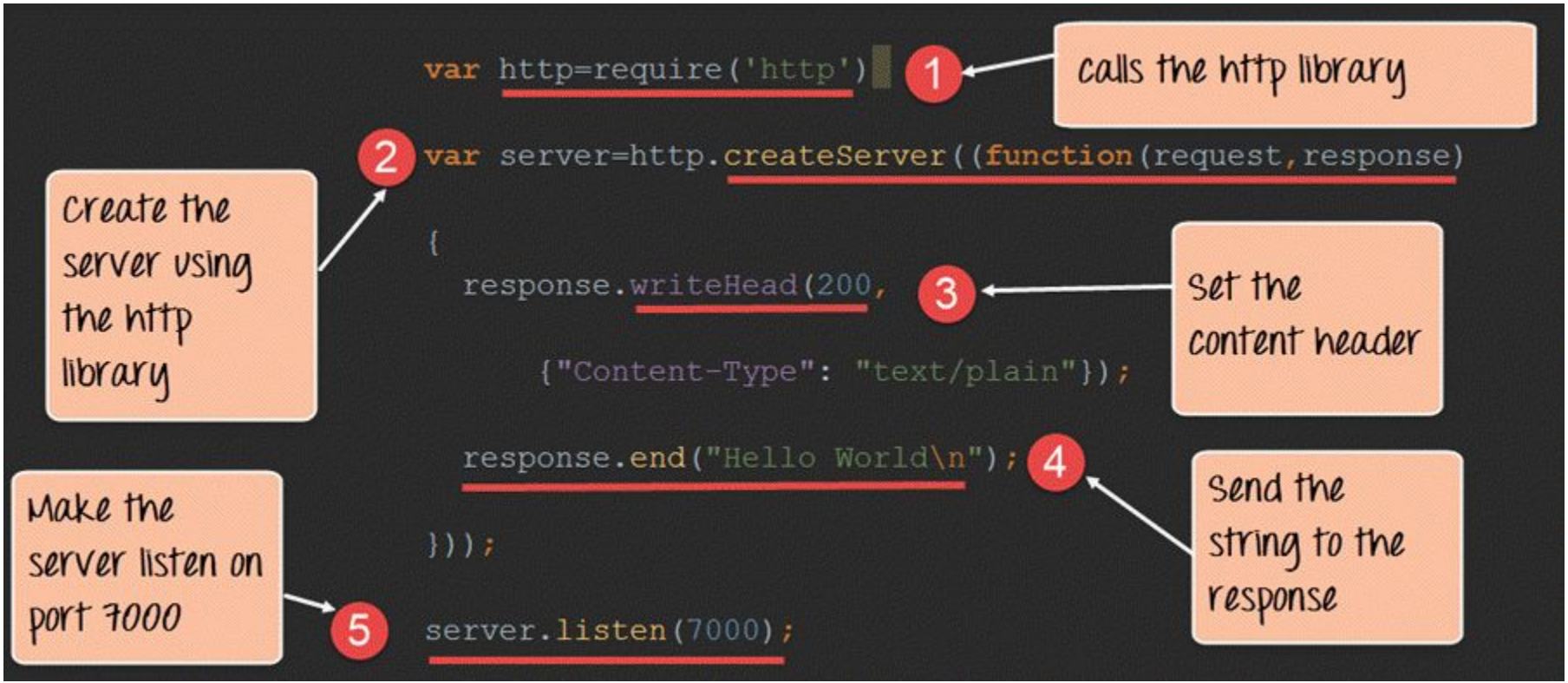
- These are the modules that can be used without any installation

Core Modules	Description
http	Includes classes, methods and events to create Node.js http server
util	Includes utility functions useful for developers
fs	Includes events, classes, and methods to deal with file I/O operations
url	Includes methods for URL parsing
querystring	Includes methods to work with query string
stream	Includes methods to handle streaming data
zlib	Includes methods to compress or decompress files

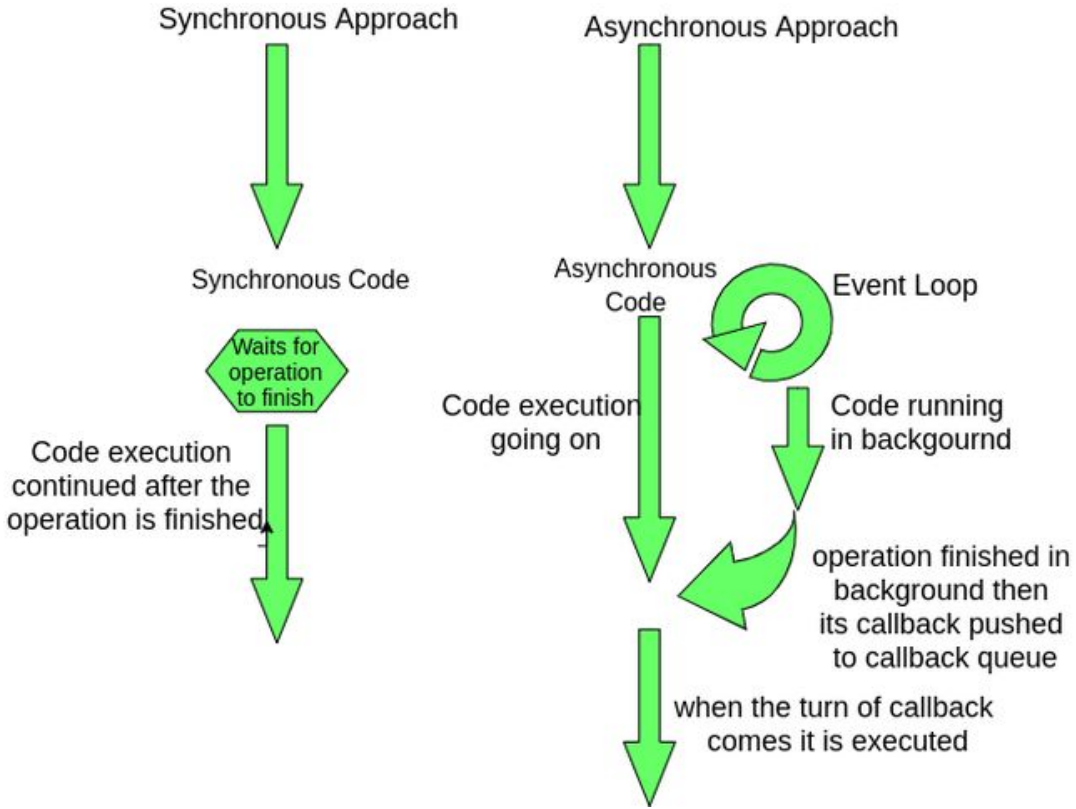
NodeJS NPM


NPM Registry with
Million Plus Javascript Libraries
ReactJs
Meteor
ElectronJS
Express.js
Koa
Angular
NestJs
Vue
Manages your Project Dependencies and lets you build on top of a vast range of libraries
MyProject
NodeJS Provides the Runtime Environment & Base Libraries
NPM
NodeJS

NodeJS HTTP


var http=require('http')
1
calls the http library
2
var server=http.createServer((function(request,response)
{
response.writeHead(200,
3
{"Content-Type": "text/plain"});
response.end("Hello World\n");
4
}));
server.listen(7000);
5
Create the server using the http library
Set the content header
Send the string to the response
Make the server listen on port 7000

NodeJS FS


Synchronous Approach
Synchronous Code
Waits for operation to finish
Code execution continued after the operation is finished
Asynchronous Approach
Asynchronous Code
Event Loop
Code execution going on
Code running in backgournd
operation finished in background then its callback pushed to callback queue
when the turn of callback comes it is executed

NodeJS FS Open a file

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


// Asynchronous - Opening File
console.log("Going to open file!");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
});
```

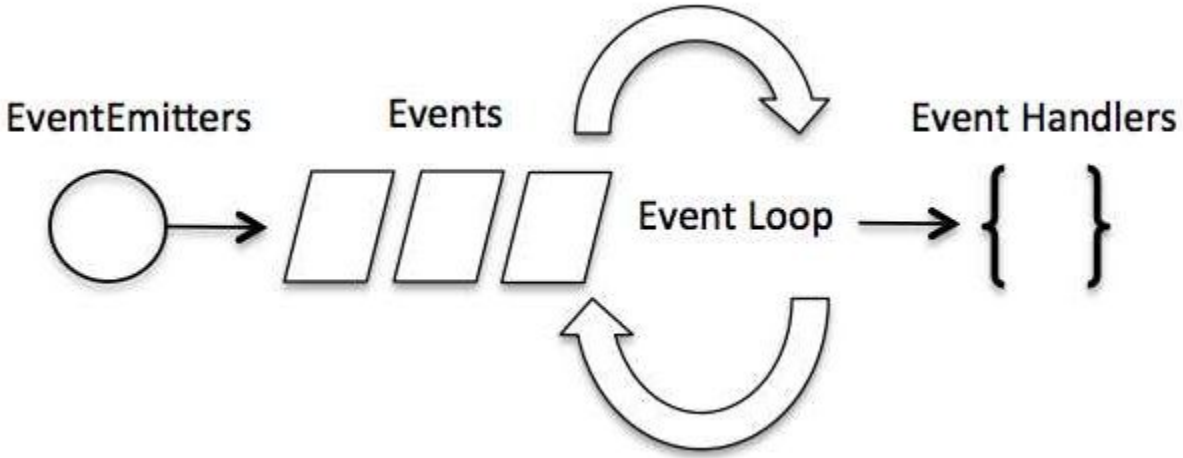
NodeJS FS Stats

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

console.log("Going to get file info!");

fs.stat('input.txt', function (err, stats) {
   if (err) {
      return console.error(err);
   }
   console.log(stats);
   console.log("Got file info successfully!");
   // Check file type
   console.log("isFile ? " + stats.isFile());
   console.log("isDirectory ? " + stats.isDirectory());
});
```

NodeJS FS Writing a file

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

console.log("Going to write into existing file");

fs.writeFile('input.txt', 'Simply Easy Learning!', function(err) {

  if (err) {

     return console.error(err); }

  console.log("Data written successfully!");

  console.log("Let's read newly written data");

  fs.readFile('input.txt', function (err, data) {

     if (err) {

        return console.error(err);}

 console.log("Asynchronous read: " + data.toString());

  });

});
```

NodeJS Events



NodeJS Events example

```
// Import events module

var events = require('events');

// Create an eventEmitter object

var eventEmitter = new events.EventEmitter();

// Create an event handler as follows

var connectHandler = function connected() {

  console.log('connection succesful.');

  // Fire the data_received event

  eventEmitter.emit('data_received');

}

// Bind the connection event with the handler

eventEmitter.on('connection', connectHandler);

// Bind the data_received event with the anonymous function

eventEmitter.on('data_received', function() {

  console.log('data received succesfully.');

});

// Fire the connection event

eventEmitter.emit('connection');

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