NODE JS

Aruna S
Department of
Computer Science and Engineering

NODE JS

NodeJS Introduction

S. Aruna
Department of Computer Science and Engineering

NODE JS

NodeJS Introduction

- Node.js is an open source, cross-platform runtime environment built on Google Chrome's JavaScript Engine (V8 Engine).
- Node.js was developed by Ryan Dahl in 2009 and its latest version is v12.18.3
- Node.js is a platform built on chrome’s JavaScript runtime for easily building fast and scalable network applications.
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.
- Node.js is open source, completely free, and used by thousands of developers around the world.

NodeJS Introduction

- Node.js applications are written in JavaScript, and can be run within the Node.js runtime on mac OS X, Microsoft Windows, and Linux.
- Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Node.js = Runtime Environment + JavaScript Library

Why Node?

When you create websites with PHP for example, you associate the language with an HTTP web server such as Apache or Nginx. Each of them has its own role in the process:

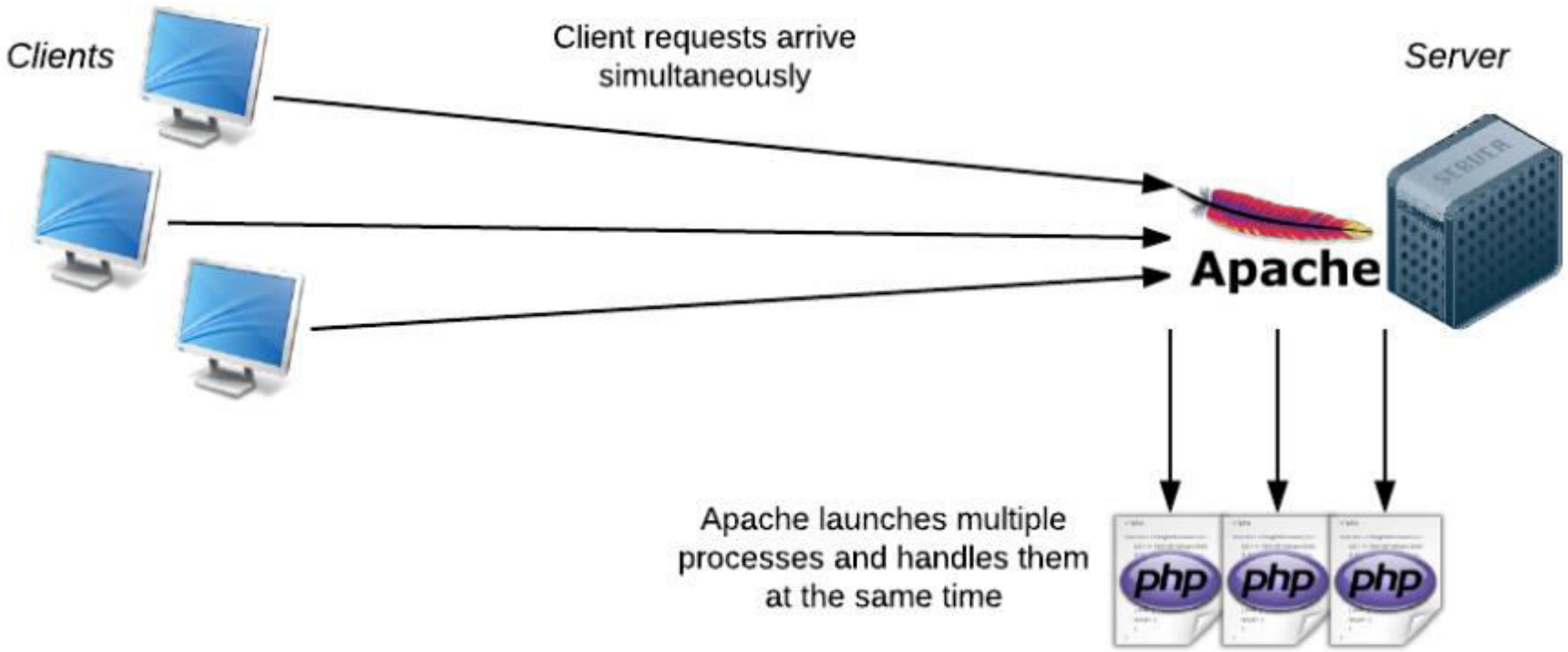
- Apache manages HTTP requests to connect to the server. Its role is more or less to manage the in/out traffic.
- PHP runs the .php file code and sends the result to Apache, which then sends it to the visitor.

As several visitors can request a page from the server at the same time, Apache is responsible for spreading them out and running different *threads* at the same time.

Each thread uses a different processor on the server (or a processor core)

NODE JS

Why Node?



The Apache server is multithread

NODE JS

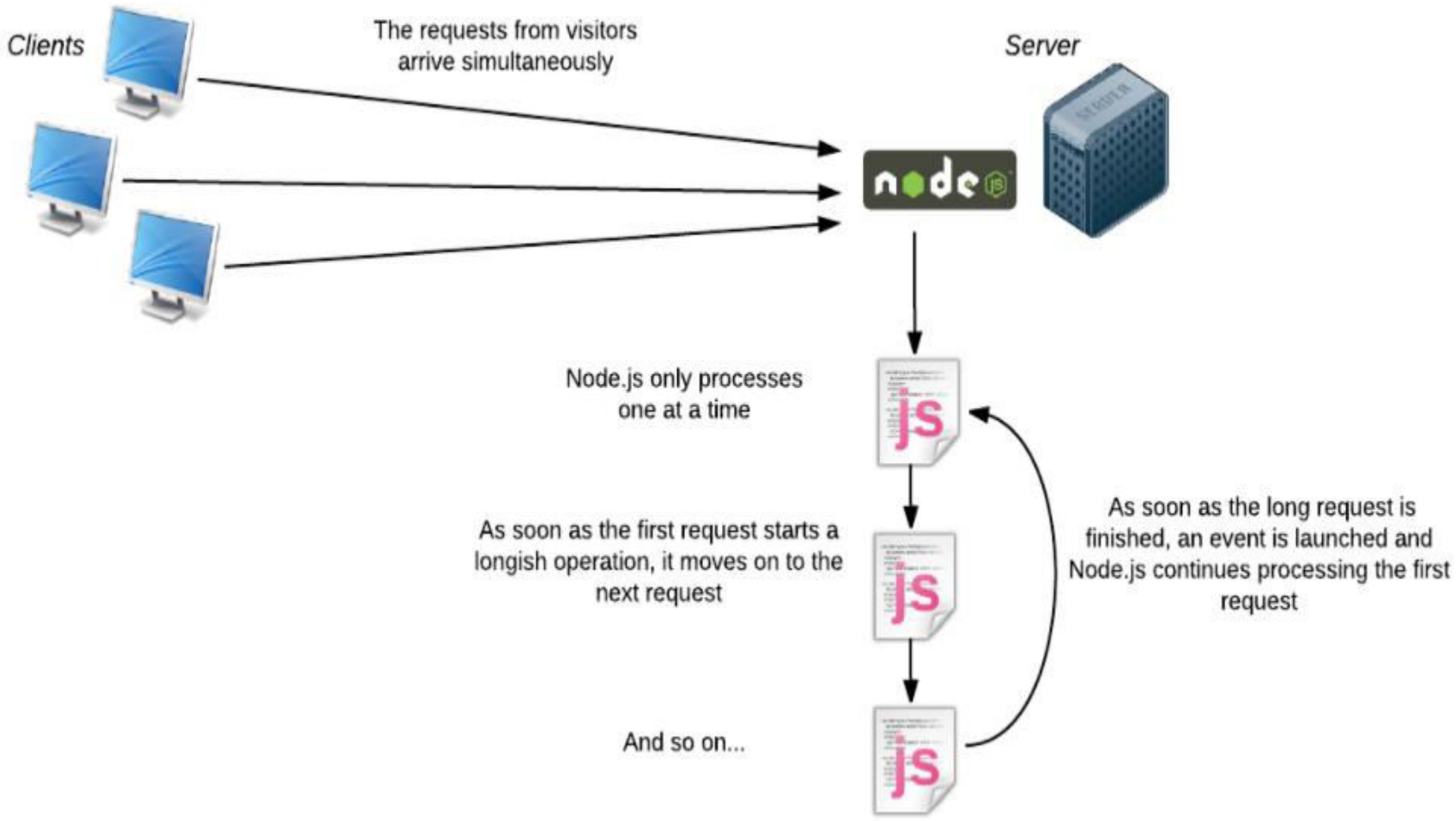
Mono-Thread – Handling multiple Requests?

- Node.js doesn't use an HTTP server like Apache. In fact, it's up to us to create the server! Isn't that great?
- Unlike Apache, Node.js is **monothread**. This means that there is only one process and one version of the program that can be used at any one time in its memory.

But I thought that Node.js was really fast because it could manage loads of requests simultaneously. If it's monothread, can it only perform one action at a time?

NODE JS

Mono-Thread Style



NODE JS

V8 Engine and C++ Bindings

- Chrome needs to run some Javascript for a particular Web page, it doesn't run the JavaScript itself. It uses the V8 engine to get that done so it passes the code to V8 and it gets the results back. Same case with Node also to run a Javascript code.
- V8 is written in C++ . Chrome and Node are largely written in C++ because they both provide bindings when they're instantiating the V8 engine.
- This facilitates to create their own JavaScript runtime with interesting and novel features.

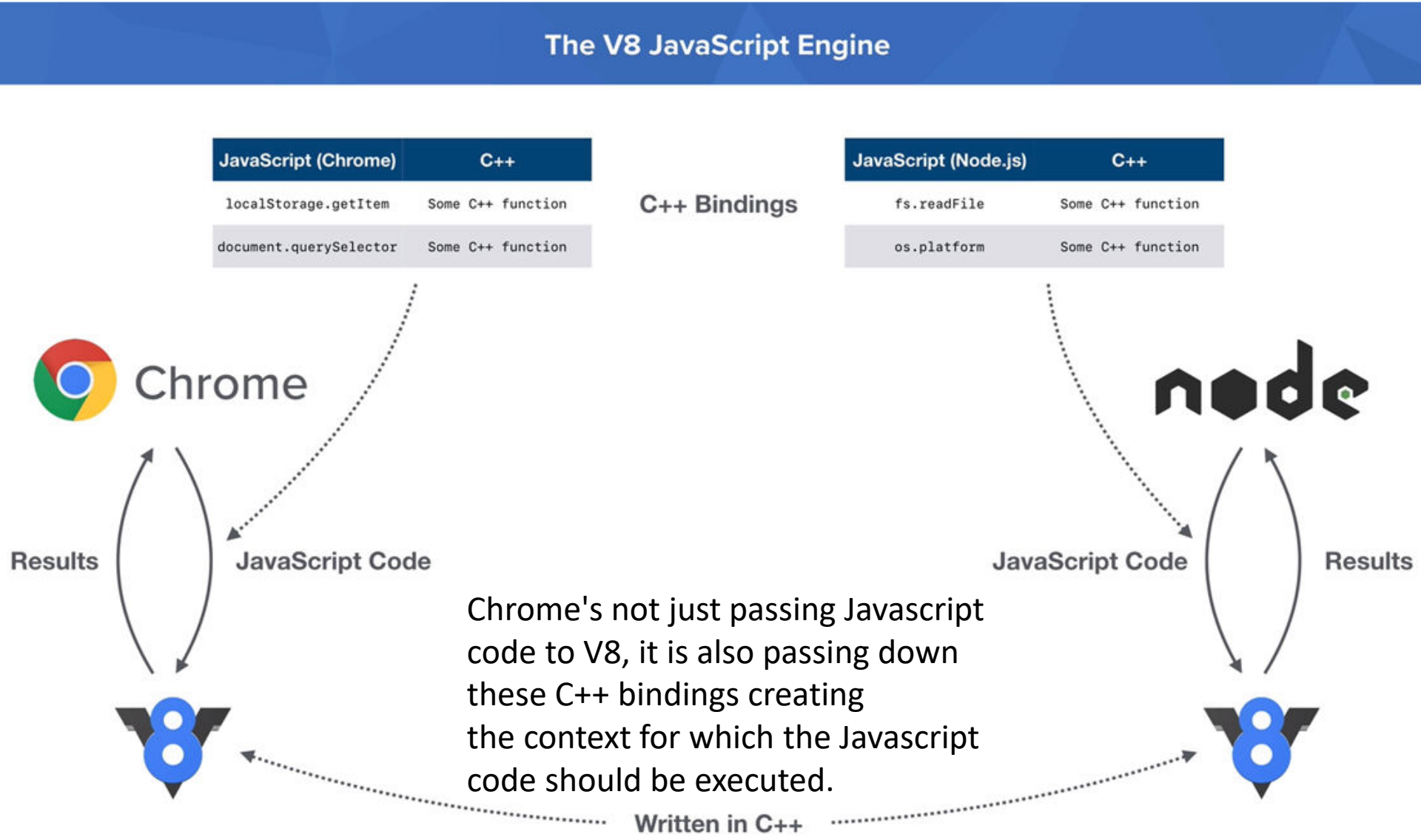
Example Instance:

Chrome to interact with the DOM when the DOM isn't part of JavaScript.

Node to interact with file system when the file system isn't part of JavaScript

NODE JS

V8 JavaScript Engine



Chrome's not just passing Javascript code to V8, it is also passing down these C++ bindings creating the context for which the Javascript code should be executed.

Features of Node.js

- Asynchronous and Event Driven
- Non Blocking I/O
- Very Fast
- Single Threaded but Highly Scalable
- No Buffering
- License

NODE JS

Features of Node.js

Very Fast

Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.

Single Threaded but Highly Scalable

Uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers.

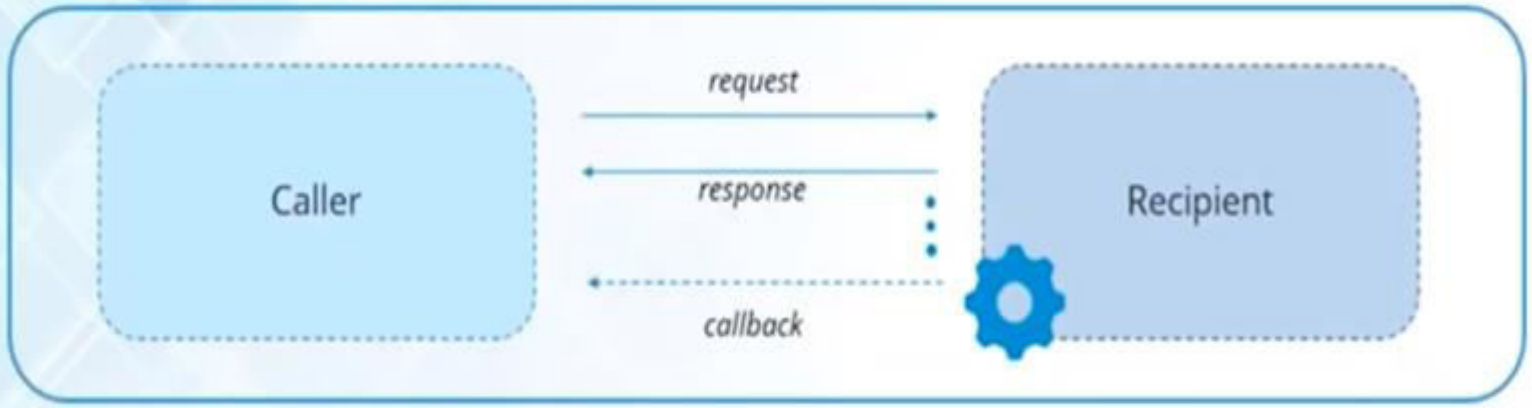
No Buffering

Node.js applications never buffer any data. These applications simply output the data in chunks.

NODE JS

Asynchronous and Event Driven

Asynchronous and Event Driven :



- When request is made to server, instead of waiting for the request to complete, server continues to process other requests
- When request processing completes, the response is sent to caller using callback mechanism

NODE JS

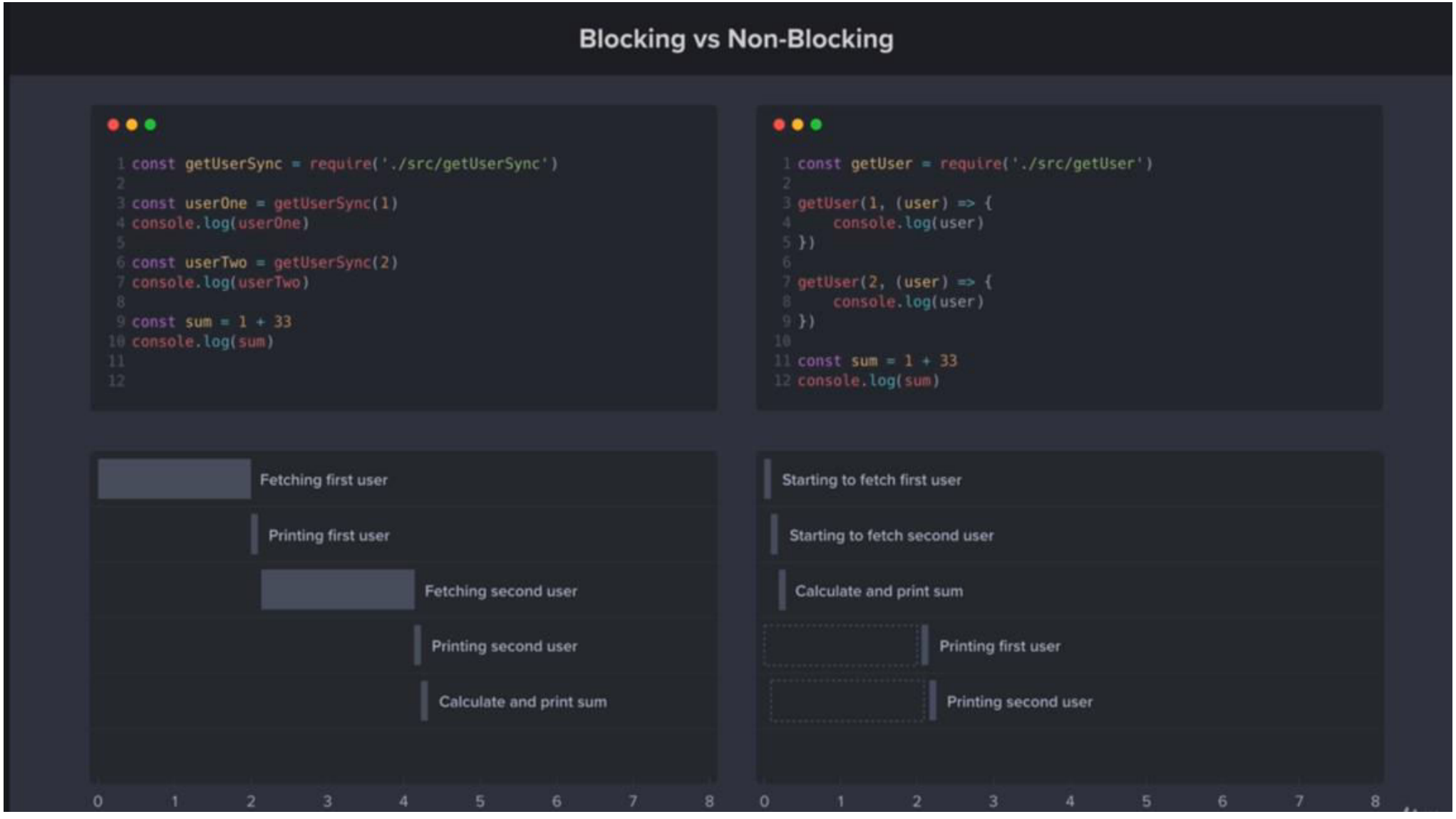
Callback Functions

- Callback is an asynchronous equivalent for a function.
- A callback function is called at the completion of a given task. Node makes heavy use of callbacks.
- All the APIs of Node are written in such a way that they support callbacks.

For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

NODE JS

Non- Blocking I/O



NODE JS

Blocking and Non- Blocking I/O

BLOCKING I/O

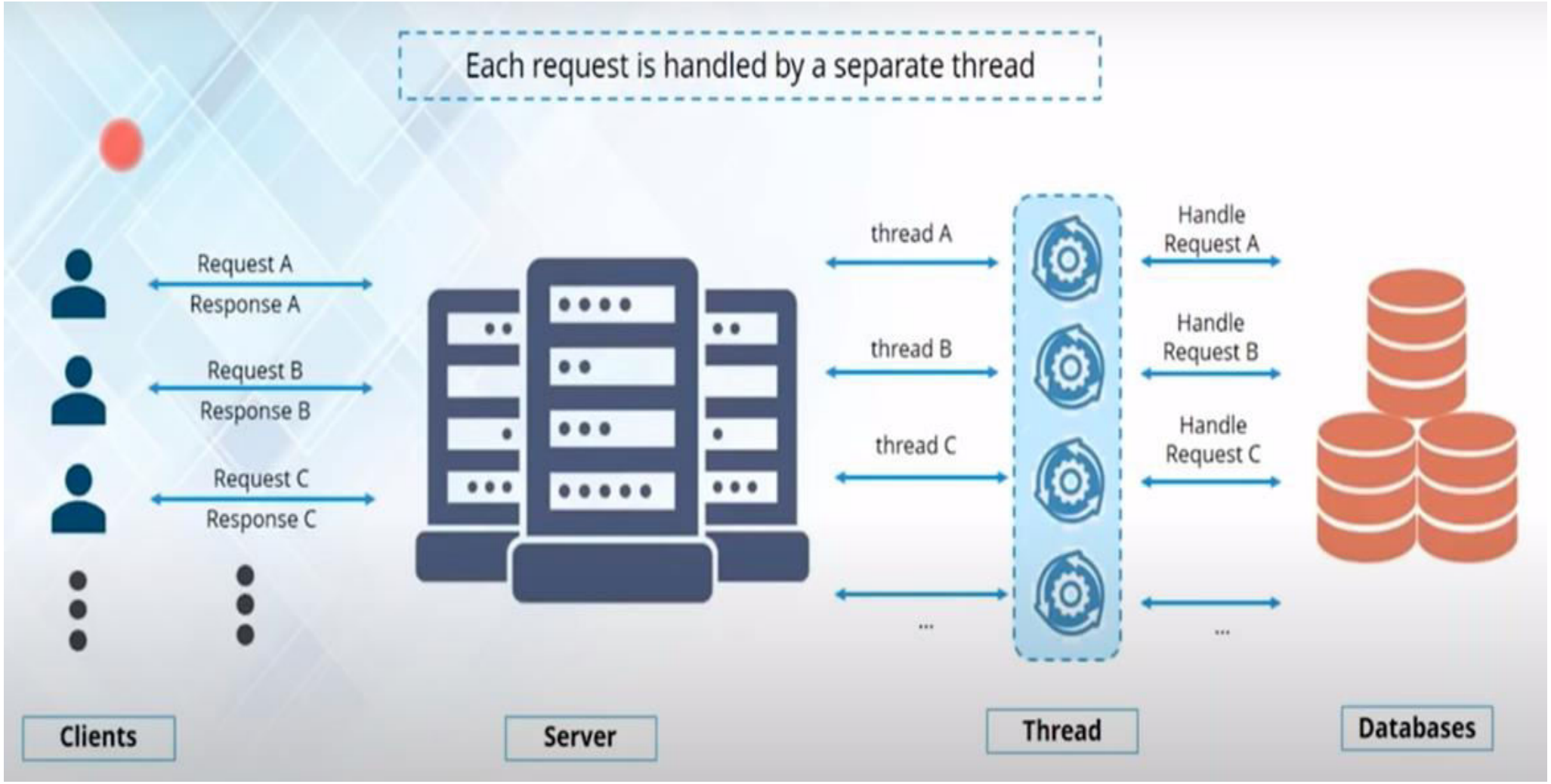
```
var fs = require('fs');
var data = fs.readFileSync('test.txt');
console.log(data.toString());
console.log('End here');
```

```
var fs = require('fs');
fs.readFile('test.txt',function(err,data){
    if(err)
    {
        console.log(err);
    }
    setTimeout(()=>{
        console.log("PES University. Display after 2 seconds")
    },2000);
});
console.log('start here');
```

NON-BLOCKING I/O

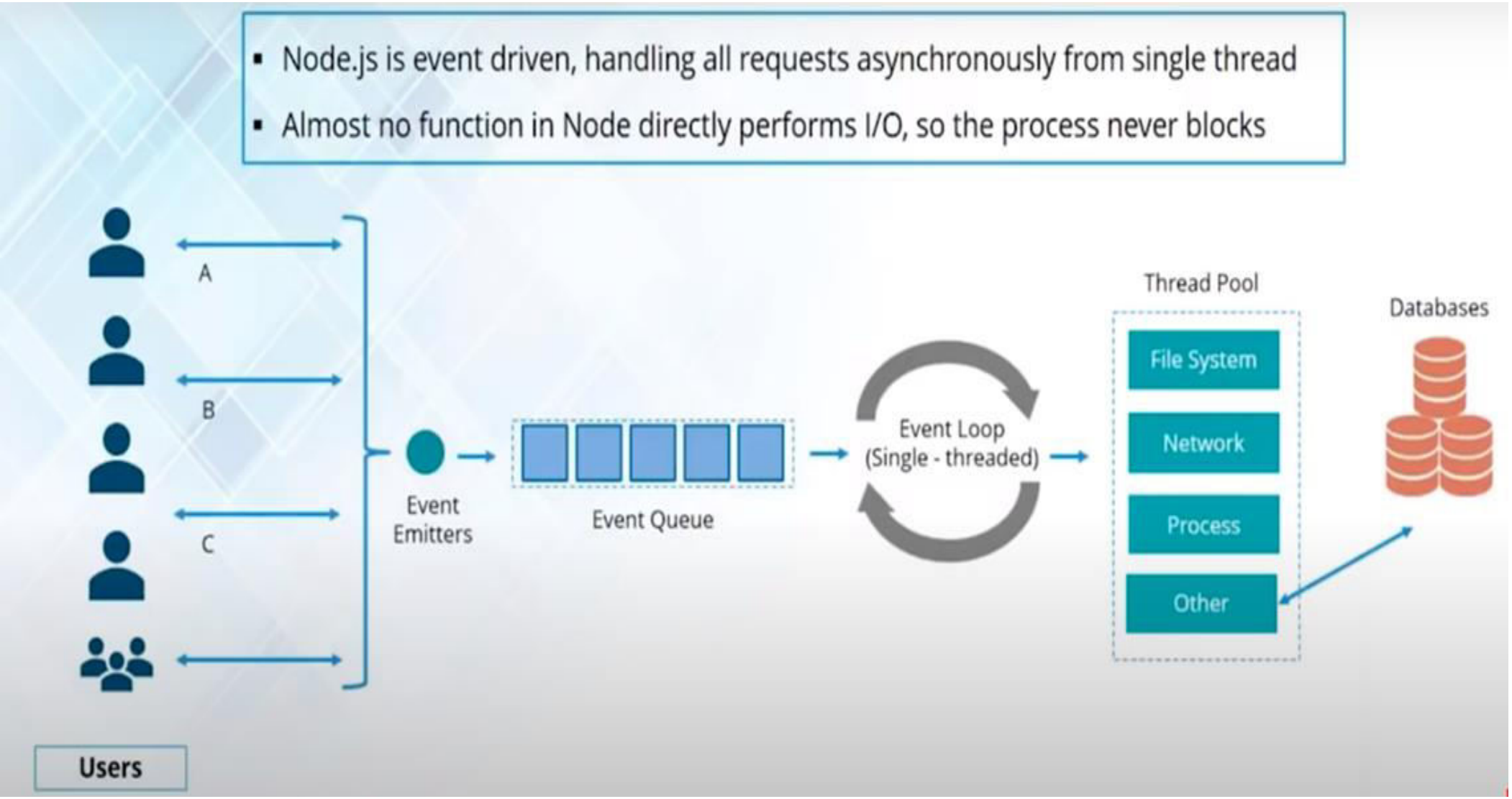
NODE JS

Multi Threaded Model



NODE JS

Single Threaded Model



NODE JS

Multi Threaded vs Asynchronous Event Driven Model

Multi-Threaded	Asynchronous Event-driven
Lock application / request with listener-workers threads	Only one thread, which repeatedly fetches an event
Using incoming-request model	Using queue and then processes it
Multithreaded server might block the request which might involve multiple events	Manually saves state and then goes on to process the next event
Using context switching	No contention and no context switches
Using multithreading environments where listener and workers threads are used frequently to take an incoming-request lock	Using asynchronous I/O facilities (callbacks, not poll/select or O_NONBLOCK) environments

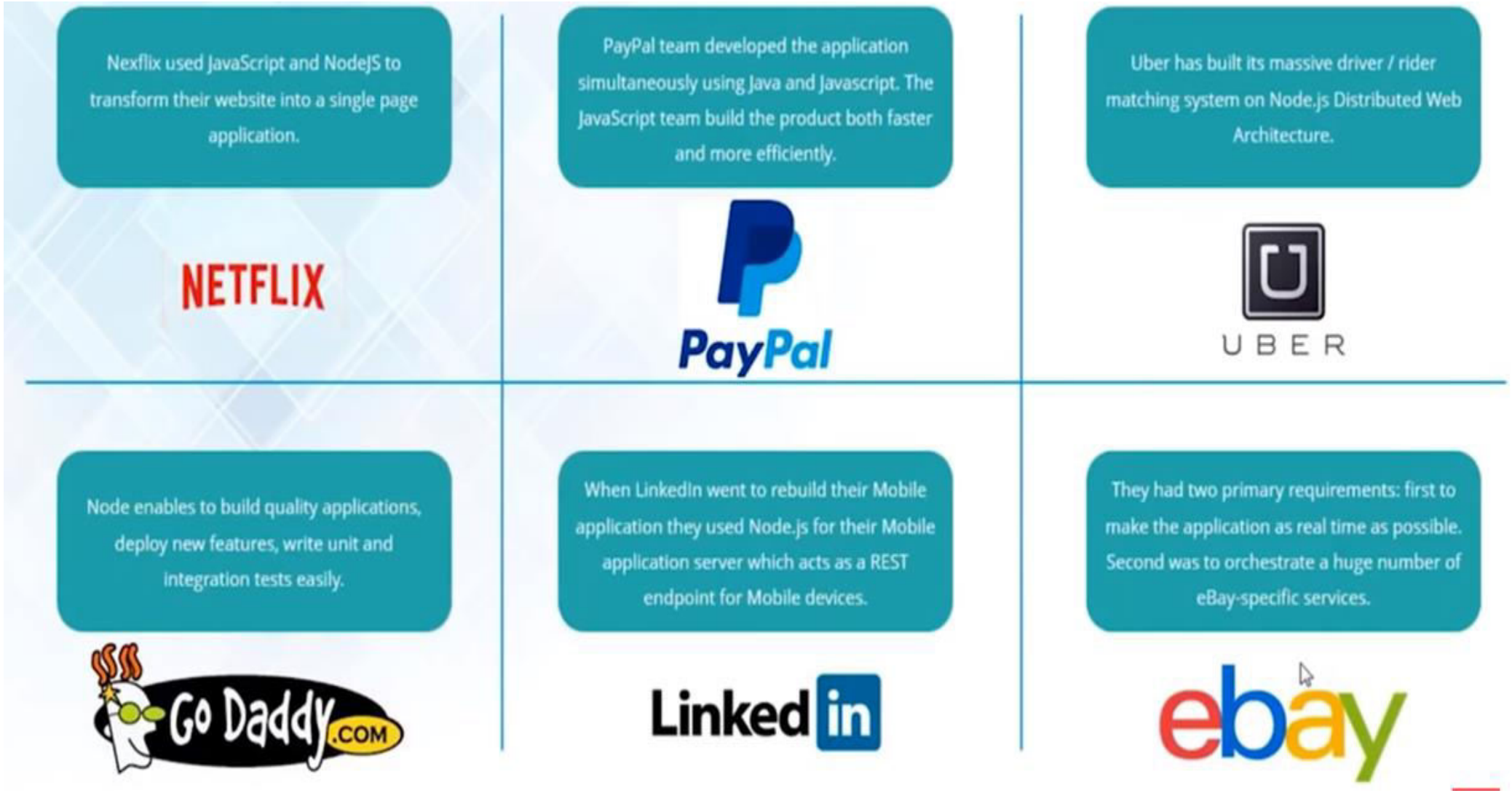
Applications of Node.js

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications
- Not for CPU intensive applications.

Link: https://www.youtube.com/watch?v=8aGhZQkoFbQ

NODE JS

Node JS Success Stories



UBER Story

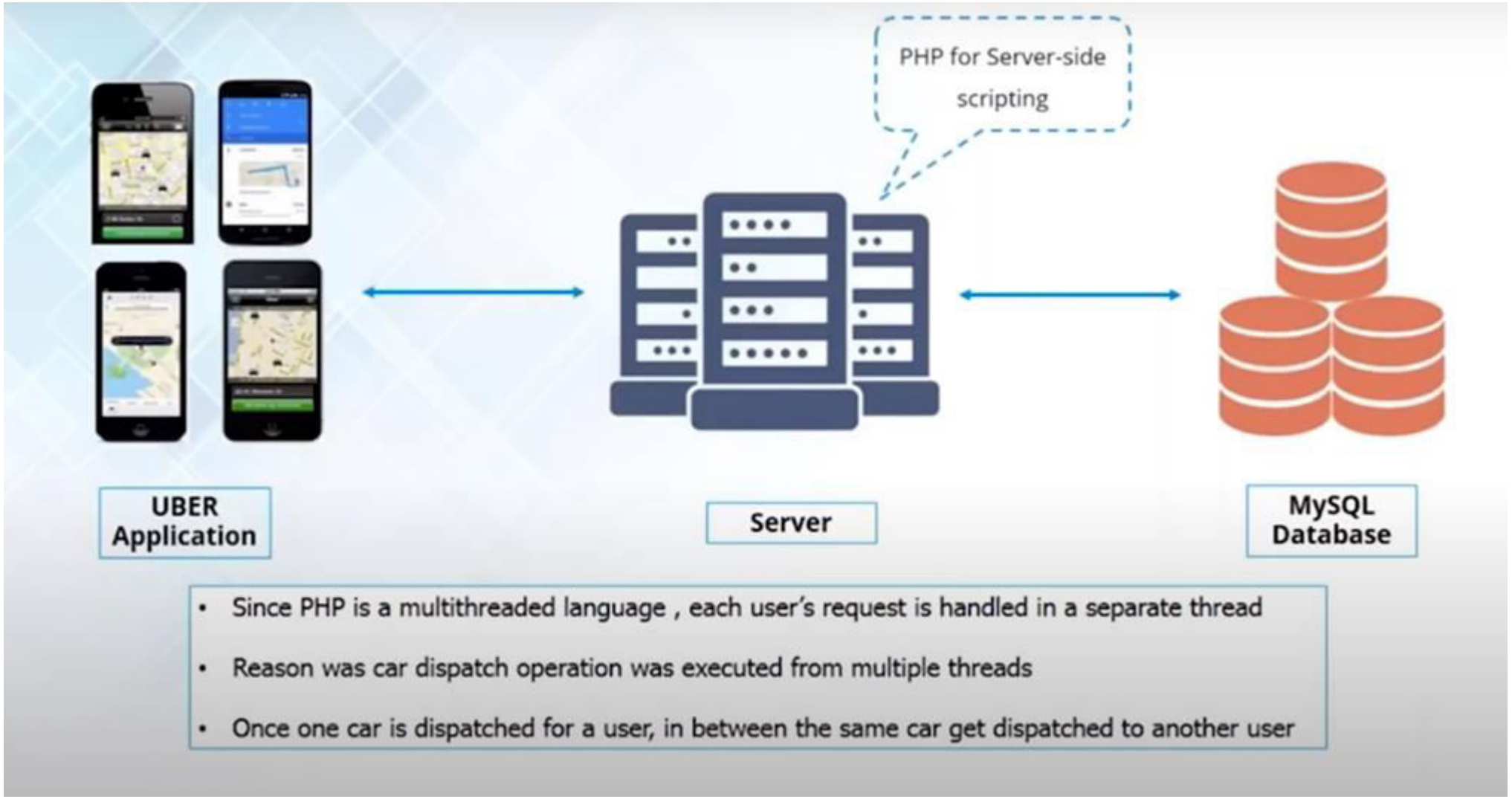
- Customers interact with Uber applications and generating request to book a cab.
- Requests are sent to the Uber server to check in the geospatial databases for the cab.
- Server send back the card details or driver details back to the customer in form of response.

Disadvantage of Multi Threaded model in this case:

- Every request is assigned a new thread from thread pool and it gets exhausted. Scalability is very poor
- Whenever a thread is working on a shared resource. It acquires a lock on that resource.

NODE JS

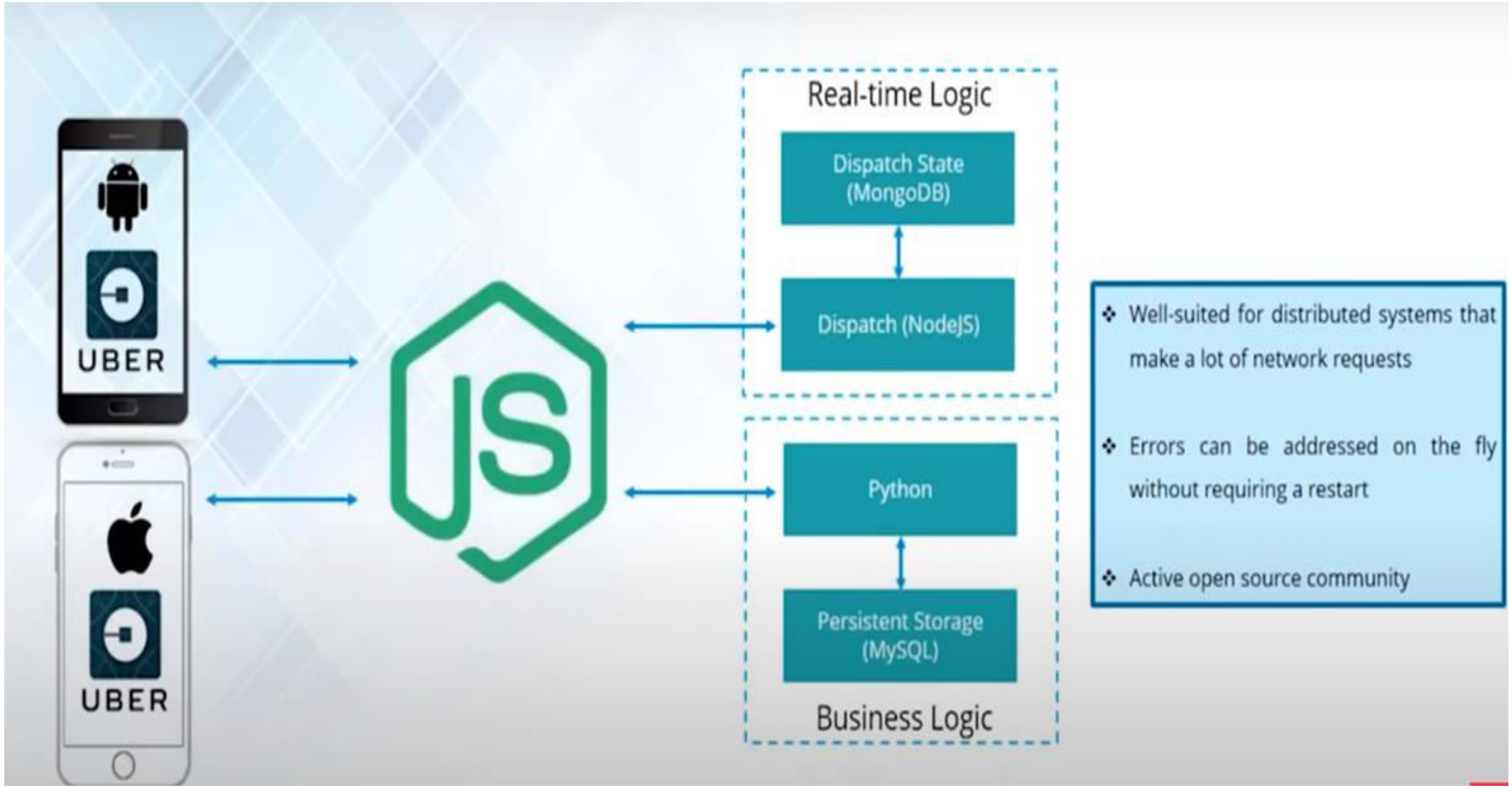
UBER Story – Old Architecture



- Since PHP is a multithreaded language , each user's request is handled in a separate thread
- Reason was car dispatch operation was executed from multiple threads
- Once one car is dispatched for a user, in between the same car get dispatched to another user

NODE JS

UBER Story – New Architecture



NODE JS

UBER Story – Single threaded Model

- User performs an activity or event is generated, each new request is taken as an event.
- Event Emitter emit those events and then those events reside inside the event queue in the server.
- Events are executed using event Loops,which is a single thread mechanism
- A worker thread present inside the thread pool is assigned for each request.
- Only one thread in this event Loop will be handling the events directly and process will never get Blocked.

NODE JS

Node JS installation

- Go to https://nodejs.org/en/
- Look for the Latest Stable Version and download it
- After Installing, Verify the installation by giving the command as

```
C:\Users\DELL>node -v
v12.18.3
```

- Install a editor like Visual Code, Sublime Text or any suitable editors for running Node JS Applications

THANK YOU

Aruna S
Department of
Computer Science and Engineering
arunas@pes.edu

NODE JS: Set up Node JS app

Aisha Begam
Department of Computer Science and Engineering

Node JS installation

- Step 1: **Go to the NodeJS website and download NodeJS**
  - Go to the NodeJS website https://nodejs.org/en/ and download NodeJS.
  - Look for the Latest Stable Version and download it
- Step 2: **Make sure Node and NPM are installed and their PATHs defined**
  - After Installing, Verify the installation by giving the command as
    - *node –v //should return the version number*
    - *npm –v // should return the version number of npm package*
  - Returns 'node' is not recognized as an internal or external command, operable program or batch file. Nodejs not installed properly check for path.
  - Install a editor like Visual Code, Sublime Text or any suitable editors for running Node JS Applications

Node JS installation

- Step 3: **Create a New Project Folder**
- Step 4: **Start running NPM in your project folder**
  - open a terminal.
  - change directories until you are in your project folder.
  - run the command npm init in the terminal

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (simple-node-server)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author: Kris Hill
license: (ISC)
About to write to C:\Users\krist\OneDrive\Documents\Blog Posts\Example Projects\simple-node-server\package.json:

{
  "name": "simple-node-server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Kris Hill",
  "license": "ISC"
}


Is this OK? (yes)
```

Node JS installation

- Step 4: **Start running NPM in your project folder**
  - a file called package.json in your project

```
{
  "name": "u4_bsection",
  "version": "1.0.0",
  "description": "demo",
  "main": "NewU4L2.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
```

Node JS installation

- Step 5: **Install Any NPM Packages. (covered in later classes)**
  - there are two parts to connecting a package to our app:
    - Download/install the package from NPM
    - Save the package name and version number under “Dependencies” in your package.json
- Step 6: **Create an HTML file**
  - Add a file to your project folder with the extension ‘.html’.
- Step 7: **Create a Node/JavaScript file in the project folder**
  - Add a file to your project folder with the extension ‘.js’.

Node JS installation

- Step 8: **Start the Node Server**
  - by opening up a terminal and running the command:
  - *node filenae.js*

- Step 9: **Visit Your (Local) Site!**

Node JS: Examples

- Create a file with .js extension and run it using the command “node filename.js”
  - console.log("Hello World")
  - const name1=“John";
  - console.log('Hello,',name1);

- Non Blocking I/O

```
setTimeout(()=>{
    console.log("Timer Stopped“)},2000);
console.log("Timer Started")
```

Node JS: Examples

- Fetching a file: To include a module, use the require() function with the name of the module:

```
const fs=require('fs')
fs.stat('U4L2.js', (err,stats)=>{
    if (err) throw err;
    console.log('Stats of U4L2.js',JSON.stringify(stats))
})
```

- Renaming a file

```
const fs=require('fs')
fs.rename('U4L2.js',"NewU4L2.js",(err)=>{
    console.log("Rename Succesfull")
})
fs.stat('NewU4L2.js', (err,stats)=>{
    if (err) throw err;
    console.log('Stats of newer.js',JSON.stringify(stats))
})
```

Node JS: Examples

- Non Blocking I/O

```
const fs=require('fs')
fs.readFile('NewU4L2.js','UTF-8' ,(err,data)=>{
    if(err) throw err
    console.log("Contents:",data)
})
console.log("Reading the Contents");
```

- Blocking I/O

```
const fs=require('fs')
const data=fs.readFileSync("NewU4L2.js",'UTF-8')
console.log("Reading the file contents...")
console.log("data:",data)
```

THANK YOU

Aisha Begam
Department of
Computer Science and Engineering
aisha.b@pes.edu

NODE JS

Aruna S
Department of
Computer Science and Engineering

NODE JS

Module System

S. Aruna
Department of Computer Science and Engineering

Node JS Modules

- https://nodejs.org/dist/latest-v12.x/docs/api/

  To see all the available modules

- Few modules are inbuilt globally available.

  Ex: Console module, Timer Module

- Many modules need to be explicitly included in our application

  Ex: File System module

Such modules need to be required at first in the application

NODE JS

Node JS Timer Module

- This module provides a way for functions to be called later at a given time.
- The Timer object is a global object in Node.js, and it is not necessary to import it

Method	Description
clearImmediate()	Cancels an Immediate object
clearInterval()	Cancels an Interval object
clearTimeout()	Cancels a Timeout object
ref()	Makes the Timeout object active. Will only have an effect if the Timeout.unref() method has been called to make the Timeout object inactive.
setImmediate()	Executes a given function immediately.
setInterval()	Executes a given function at every given milliseconds
setTimeout()	Executes a given function after a given time (in milliseconds)
unref()	Stops the Timeout object from remaining active

NODE JS

NPM and installing modules

- NPM is a package manager for Node.js packages, or modules if you like.
- www.npmjs.com hosts thousands of free packages to download and use.
- The NPM program is installed on your computer when you install Node.js

  A package in Node.js contains all the files you need for a module.

  Modules are JavaScript libraries you can include in your project.

Example:

```
D:\nodejs>npm install validator
```

- validator package is downloaded and installed. NPM creates a folder named "node_modules", where the package will be placed.
- To include a module, use the **require()** function with the name of the module

```
var val = require('validator');
```

NODE JS

Importing your own modules

- The module.exports is a special object which is included in every JavaScript file in the Node.js application by default.
- The module is a variable that represents the current module, and exports is an object that will be exposed as a module.
- So, whatever you assign to module. exports will be exposed as a module. It can be
  - Export Literals
  - Export Objects
  - Export Functions
  - Export Function as a class

NODE JS

Importing your own modules

You can create your own modules, and easily include them in your applications. The following example creates a module that returns a date and

```
exports.myDateTime = function () {
    return Date();
  };
```

Use the exports keyword to make properties and methods available outside the module file.

```
var date = require('./myfirstmodule.js');
console.log(date.myDateTime());
```

```
PS C:\Users\DELL\Desktop\WTII\Node Examples\notes-app> node app.js
Sat Sep 12 2020 12:03:34 GMT+0530 (India Standard Time)
```

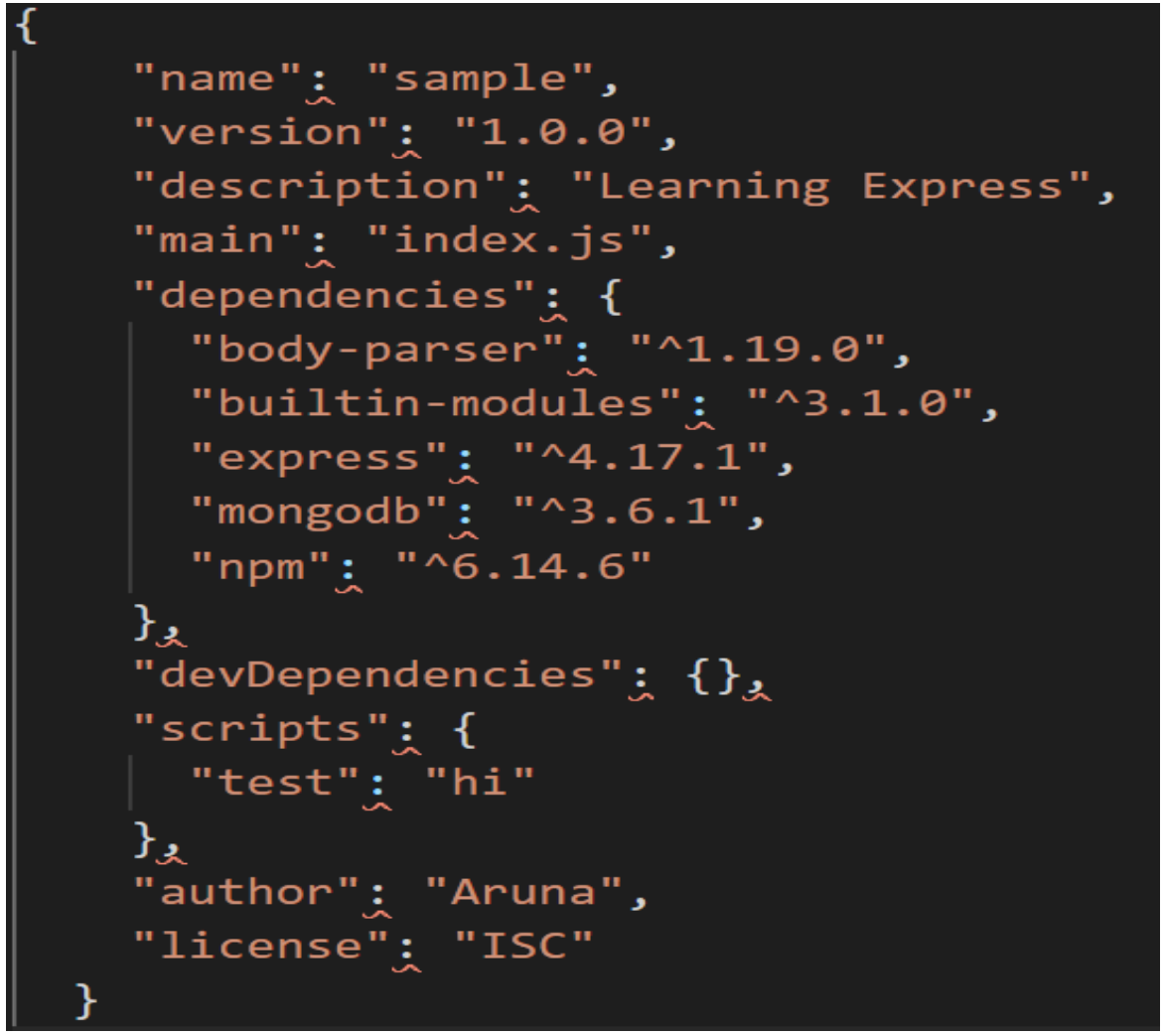
NODE JS

Package.json file

- All npm packages contain a file, usually in the project root, called package.json
- This file holds various metadata relevant to the project.
- It is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.
- It can also contain other metadata such as a project description, the version of the project in a particular distribution, license information, even configuration data - all of which can be vital to both npm and to the end users of the package.
- The package.json file is normally located at the root directory of a Node.js project.

NODE JS

A Sample Package.json file



```
{
    "name": "sample",
    "version": "1.0.0",
    "description": "Learning Express",
    "main": "index.js",
    "dependencies": {
      "body-parser": "^1.19.0",
      "builtin-modules": "^3.1.0",
      "express": "^4.17.1",
      "mongodb": "^3.6.1",
      "npm": "^6.14.6"
    },
    "devDependencies": {},
    "scripts": {
      "test": "hi"
    },
    "author": "Aruna",
    "license": "ISC"
  }
```

Demo for Modules

- **Installation of validator module:**
- You can install this package by using this command.
  - *npm install validator*
- After installing validator module you can check your validator version in command prompt using the command.
  - *npm version validator*

NODE JS

Demo for Modules

```
const validator = require('validator')

// Check whether given email is valid or not
var email = 'testmail@gmail.com'
console.log(validator.isEmail(email)) // true
email = 'testmail@'
console.log(validator.isEmail(email)) // false

// Check whether string is in lowercase or not
var name = 'john'
console.log(validator.isLowercase(name)) // true
name = 'JOHN'
console.log(validator.isLowercase(name)) // false

// Check whether string is empty or not
var name = '    '
console.log(validator.isEmpty(name)) // true
name = 'Smith'
console.log(validator.isEmpty(name)) // false

// Other functions also available in  isBoolean(), isCurrency(), isDecimal(), isJSON(),
isJWT(), isFloat(), isCreditCard(), etc.
```

Demo for Modules

Create Modules in Node.js

- To create a module in Node.js, use **exports** keyword tells Node.js that the function can be used outside the module.
- **Create a file that you want to export**

JS calc.js > ...

```
1   exports.add = function (a, b) {
2       return a + b;
3   };
4
5   exports.sub = function (a, b) {
6       return a - b;
7   };
8
9   exports.mult = function (a, b) {
10      return a * b;
11  };
12    exports.div = function (a, b) {
13      return a / b;
14  };
```

- **Use the 'require' keyword to import the file**

JS U4L2.js > ...

```
14
15  var a = 50, b = 20;
16
17  console.log("Addition of 50 and 20 is "
18                          + calculator.add(a, b));
19
20  console.log("Subtraction of 50 and 20 is "
21                          + calculator.sub(a, b));
22
23  console.log("Multiplication of 50 and 20 is "
24                          + calculator.mult(a, b));
25
26  console.log("Division of 50 and 20 is "
27                          + calculator.div(a, b));
```

THANK YOU

Aruna S
Department of
Computer Science and Engineering
arunas@pes.edu

NODE JS

Aruna S
Department of
Computer Science and Engineering

NODE JS

File System Module

S. Aruna
Department of Computer Science and Engineering

NODE JS

File System Module

- Node implements File I/O using simple wrappers around standard POSIX functions.
- The Node File System (fs) module can be imported using the following syntax –

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

Synchronous vs Asynchronous

- Every method in the fs module has synchronous as well as asynchronous forms.
- Asynchronous methods take the last parameter as the completion function callback

and the first parameter of the callback function as error.

- It is better to use an asynchronous method instead of a synchronous method, as the

former never blocks a program during its execution, whereas the second one does.

NODE JS

Fs module operations

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

Fs File Open

Syntax

Following is the syntax of the method to open a file in asynchronous mode −

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

Parameters

Here is the description of the parameters used −

path − This is the string having file name including path.

flags − Flags indicate the behavior of the file to be opened. All possible values have been mentioned below.

mode − It sets the file mode (permission and sticky bits), but only if the file was created. It defaults to 0666, readable and writeable.

callback − This is the callback function which gets two arguments (err, fd).

Flags for read/write operations are − r,r+,rs,rs+,w,wx,w+,wx+,a,ax,a+,ax+

Fs File Write

Syntax

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

This method will over-write the file if the file already exists. If you want to write into an existing file then you should use another method available.

Parameters

path – This is the string having the file name including path.

data – This is the String or Buffer to be written into the file.

options – The third parameter is an object which will hold {encoding, mode, flag}. By default. encoding is utf8, mode is octal value 0666. and flag is 'w'

callback – This is the callback function which gets a single parameter err that returns an error in case of any writing error.

Syntax

fs.read(fd, buffer, offset, length, position, callback) This method will use file descriptor to read the file. If you want to read the file directly using the file name, then you should use another method available.

Parameters

fd – This is the file descriptor returned by fs.open().

buffer – This is the buffer that the data will be written to.

offset – This is the offset in the buffer to start writing at.

length – This is an integer specifying the number of bytes to read.

position – This is an integer specifying where to begin reading from in the file. If position is null, data will be read from the current file position.

callback – This is the callback function which gets the three arguments, (err, bytesRead, buffer).

Unlinking a File

Use fs.unlink() method to delete an existing file.

fs.unlink(path, callback);

Closing a File

fs.close(fd, callback)

fd – This is the file descriptor returned by file fs.open() method.

callback – This is the callback function No arguments other than a possible exception are given to the completion callback.

Truncate a File

fs.truncate(fd, len, callback)

fd – This is the file descriptor returned by fs.open().

len – This is the length of the file after which the file will be truncated.

callback – This is the callback function No arguments other than a possible exception are given to the completion callback.

NODE JS

Fs Module – Other Important Methods

Method	Description
fs.readFile(fileName [,options], callback)	Reads existing file.
fs.writeFile(filename, data[, options], callback)	Writes to the file. If file exists then overwrite the content otherwise creates new file.
fs.open(path, flags[, mode], callback)	Opens file for reading or writing.
fs.rename(oldPath, newPath, callback)	Renames an existing file.
fs.chown(path, uid, gid, callback)	Asynchronous chown.
fs.stat(path, callback)	Returns fs.stat object which includes important file statistics.
fs.link(srcpath, dstpath, callback)	Links file asynchronously.
fs.symlink(destination, path[, type], callback)	Symlink asynchronously.
fs.rmdir(path, callback)	Renames an existing directory.
fs.mkdir(path[, mode], callback)	Creates a new directory.
fs.readdir(path, callback)	Reads the content of the specified directory.
fs.utimes(path, atime, mtime, callback)	Changes the timestamp of the file.
fs.exists(path, callback)	Determines whether the specified file exists or not.
fs.access(path[, mode], callback)	Tests a user's permissions for the specified file.
fs.appendFile(file, data[, options], callback)	Appends new content to the existing file.

THANK YOU

Aruna S
Department of
Computer Science and Engineering
arunas@pes.edu

NODE JS

Aruna S
Department of
Computer Science and Engineering

NODE JS

Buffers and Streams

S. Aruna
Department of Computer Science and Engineering

NODE JS

Buffers Introduction

- Pure JavaScript is Unicode friendly, but it is not so for binary data.
- Working with TCP streams or the file system, it's necessary to handle octet streams.
- Node provides Buffer class which provides instances to store raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap.
- Buffer class is a global class that can be accessed in an application without importing the buffer module.

NODE JS

Buffer Operations

- Creating Buffers
- Writing to Buffers
- Reading from Buffers
- Concatenate Buffers
- Copy Buffers
- Compare Buffers

Creating Buffers

- The **Buffer.alloc() method** is used to create a new buffer object of the specified size.
- This method is slower than **Buffer.allocUnsafe() method** but it assures that the newly created Buffer instances will never contain old information or data that is potentially sensitive.

Syntax

Buffer.alloc(size, fill, encoding)

size: It specifies the size of the buffer.
fill: It is an optional parameter and specifies the value to fill the buffer. Its default value is 0.
encoding: It is an optional parameter that specifies the value if the buffer value is a string. Its default value is **'utf8'**.
Return Value: This method returns a new initialized Buffer of the specified size.
A TypeError will be thrown if the given size is not a number.

Writing to Buffers

Syntax

buf.write(string[, offset][, length][, encoding]) Parameters

string – This is the string data to be written to buffer.
offset – This is the index of the buffer to start writing at. Default value is 0.
length – This is the number of bytes to write. Defaults to buffer.length.
encoding – Encoding to use. 'utf8' is the default encoding.

Return Value

This method returns the number of octets written. If there is not enough space in the buffer to fit the entire string, it will write a part of the string.

Reading from Buffers

Syntax

buf.toString([encoding][, start][, end]) Parameters

encoding – Encoding to use. 'utf8' is the default encoding.
start – Beginning index to start reading, defaults to 0.
end – End index to end reading, defaults is complete buffer.

Return Value

This method decodes and returns a string from buffer data encoded using the specified character set encoding.

NODE JS

Compare Buffers

buf.compare(target[, targetStart[, targetEnd[, sourceStart[, sourceEnd]]]])

- target <Buffer> | <Uint8Array> A Buffer or Uint8 Array with which to compare buf.
- targetStart <integer> The offset within target at which to begin comparison. Default: 0.
- targetEnd <integer> The offset within target at which to end comparison (not inclusive). Default: target.length.
- sourceStart <integer> The offset within buf at which to begin comparison. Default: 0.
- sourceEnd <integer> The offset within buf at which to end comparison (not inclusive). Default: buf.length.
- Returns: <integer>

Compares buf with target and returns a number indicating whether buf comes before, after, or is the same as target in sort order.

0 is returned if target is the same as buf

1 is returned if target should come before buf when sorted.

-1 is returned if target should come after buf when sorted.

NODE JS

Copy Buffers

buf.copy(target[, targetStart[, sourceStart[, sourceEnd]]])#

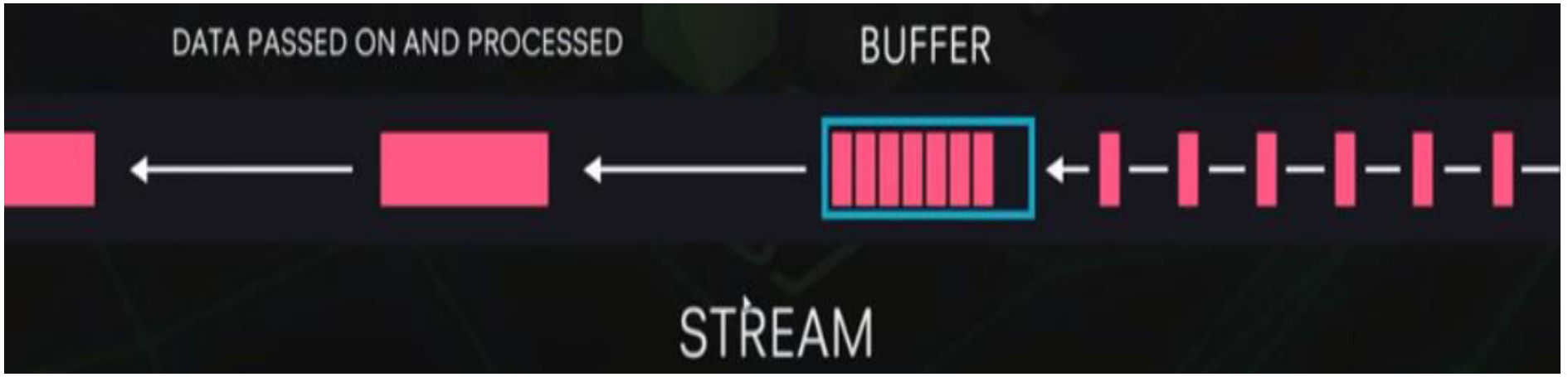
- target <Buffer> | <Uint8Array> A Buffer or Uint8Array to copy into.
- targetStart <integer> The offset within target at which to begin writing. Default: 0.
- sourceStart <integer> The offset within buf from which to begin copying. Default: 0.
- sourceEnd <integer> The offset within buf at which to stop copying (not inclusive).

  Default: buf.length.
- Returns: <integer> The number of bytes copied.

Copies data from a region of buf to a region in target, even if the target

memory region overlaps with buf.

Buffers and Streams in Action – YouTube Example



NODE JS

Streams

- Streams are one of the fundamental concepts that power Node.js applications.
- They are data-handling method and are used to read or write input into output sequentially.
- Streams are a way to handle reading/writing files, network communications, or any kind of end-to-end information exchange in an efficient way.
- A program reads a file into memory **all at once** like in the traditional way, whereas streams read chunks of data piece by piece, processing its content without keeping it all in memory.
- This makes streams really powerful when working with **large amounts of data**, for example, a file size can be larger than your free memory space, making it impossible to read the whole file into the memory in order to process it.
- Streams also give us the power of 'composability' in our code

For Example "streaming" services such as **YouTube or Netflix**

Why Streams

Streams basically provide two major advantages compared to other data handling methods:

Memory efficiency: you don’t need to load large amounts of data in memory before you are able to process it

Time efficiency: it takes significantly less time to start processing data as soon as you have it, rather than having to wait with processing until the entire payload has been transmitted

Types of streams in Node Js

There are 4 types of streams in Node.js:

Writable: streams to which we can write data. For example, fs.createWriteStream() lets us write data to a file using streams.

Readable: streams from which data can be read. For example: fs.createReadStream() lets us read the contents of a file.

Duplex: streams that are both Readable and Writable. For example, net.Socket

Transform: streams that can modify or transform the data as it is written and read. For example, in the instance of file-compression, you can write compressed data and read decompressed data to and from a file.

In HTTP server, **request is a readable stream and response is a writable stream.**

Streams as Events

Each type of Stream is an **EventEmitter** instance and throws several events at different instance of times.

For example, 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 to underlying system.

NODE JS

Native Nodejs Objects - streams

Readable Streams	Writable Streams
HTTP responses, on the client	HTTP requests, on the client
HTTP requests, on the server	HTTP responses, on the server
fs read streams	fs write streams
zlib streams	zlib streams
crypto streams	crypto streams
TCP sockets	TCP sockets
child process stdout and stderr	child process stdin
process.stdin	process.stdout, process.stderr

THANK YOU

Aruna S
Department of
Computer Science and Engineering
arunas@pes.edu

NODE JS

Aruna S
Department of
Computer Science and Engineering

NODE JS

HTTP Module

S. Aruna
Department of Computer Science and Engineering

NODE JS

Web Server

- A Web Server is a software application which handles HTTP requests sent by the HTTP client, like web browsers, and returns web pages in response to the clients.
- Web servers usually deliver html documents along with images, style sheets, and scripts.
- Most of the web servers support server-side scripts, using scripting languages or redirecting the task to an application server which retrieves data from a database and performs complex logic and then sends a result to the HTTP client through the Web server.
- Apache web server is one of the most commonly used web servers. It is an open source project.

NODE JS

Web Application Architecture

A Web application is usually divided into four layers −

Client − This layer consists of web browsers, mobile browsers or applications which can make HTTP requests to the web server.

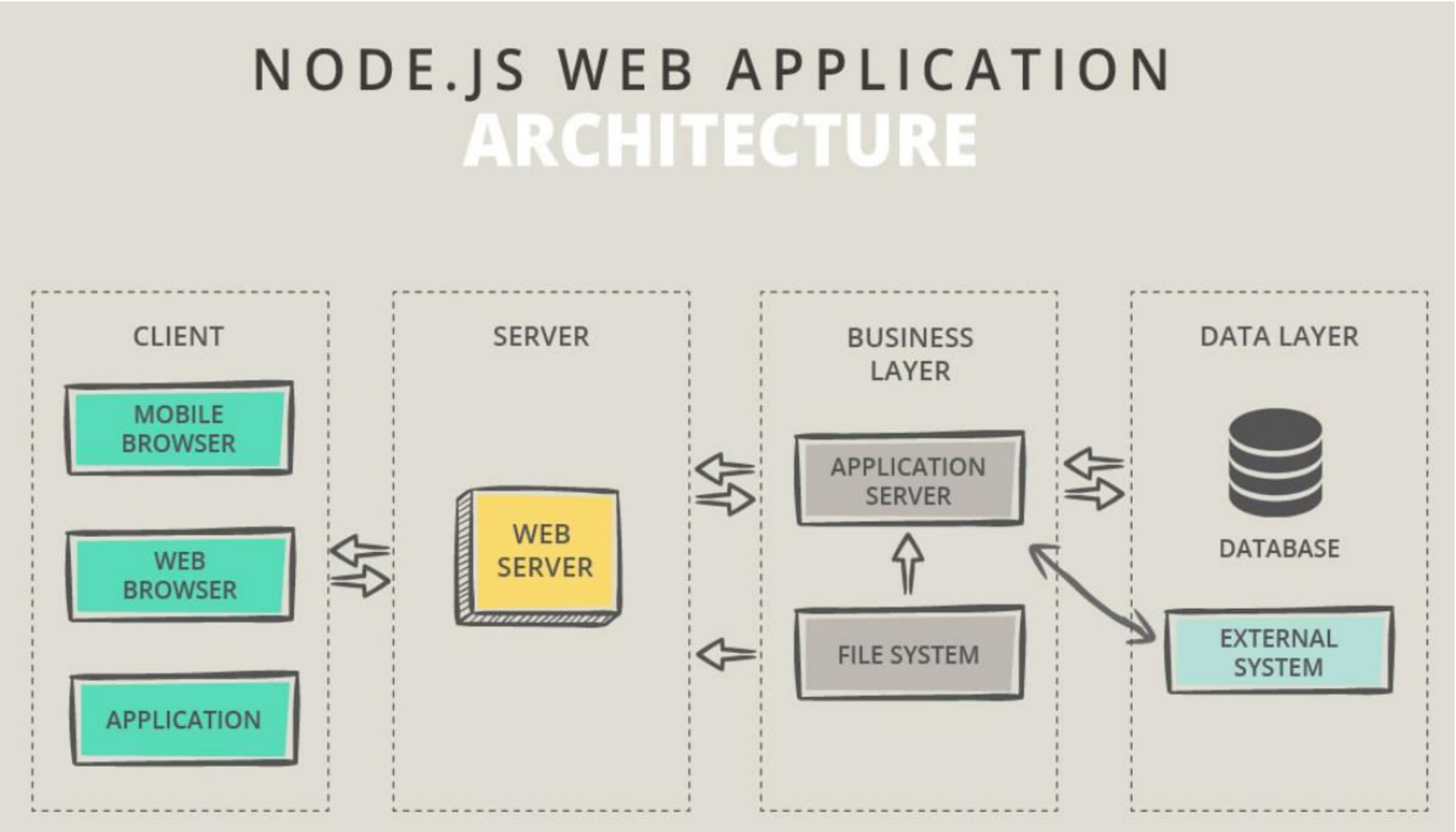
Server − This layer has the Web server which can intercept the requests made by the clients and pass them the response.

Business − This layer contains the application server which is utilized by the web server to do the required processing. This layer interacts with the data layer via the database or some external programs.

Data − This layer contains the databases or any other source of data.

NODE JS

Web Application Architecture



NODE JS

HTTP Module – Web Server using Node

The components of a Node.js application.

Import required modules – We use the **require** directive to load Node.js modules.

Create server – A server which will listen to client's requests similar to Apache HTTP Server. Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

Read request and return response – The server created in an earlier step will read the HTTP request made by the client which can be a browser or a console and return the response.

Creating Node.js Application

Step 1 - Import Required Module

We use the **require** directive to load the http module and store the returned HTTP instance into an http variable as follows

```
var http = require('http');
```

Step 2 - Create Server

- We use the created http instance and call **http.createServer()** method to create a server instance.
- Then we bind it at port 8088 using the **listen** method associated with the server instance.

```
http.createServer(function (request, response) {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end('Hello PES University\n');
 }).listen(8088);
  console.log('Server running at http://127.0.0.1:8088/')
```

Step 3 - Testing Request & Response

$node app.js

Verify the Output. Server has started.

Server running at http://127.0.0.1:8088/

NODE JS

URL Module

The URL module splits up a web address into readable parts.
To include the URL module, use the require() method:

```
var url = require('url');
```

Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

Node Examples > JS app.js > ...

```
1   var url = require('url');
2   var adr = 'http://localhost:8080/pesu.htm?year=2020&month=September';
3   var q = url.parse(adr, true);
4
5   console.log(q.host); //returns 'localhost:8080'
6   console.log(q.pathname); //returns '/pesu.htm'
7   console.log(q.search); //returns '?year=2020&month=September'
8
9   var qdata = q.query; //returns an object: { year: 2020, month: 'september' }
10  console.log(qdata.month); //returns 'september'
```

NODE JS

Web Client Using Node

A web client can be created using **http** module. A Screenshot of the example is below

```
var http = require('http');
var options = {
   host: 'localhost',
   port: '8081',
   path: '/index.htm'
};
var callback = function(response) {
   var body = '';
   response.on('data', function(data) {
      body += data;
   });

   response.on('end', function() {
      console.log(body);
   });
}
var req = http.request(options, callback);
req.end();
```

Node JS and MongoDB Connectivity

- Node.js can be used in database applications.
- One of the most popular NoSQL database is MongoDB.
- Download a free MongoDB database at https://www.mongodb.com.
- Node.js can use this module to manipulate MongoDB databases

require('mongodb');

Creating a Database

- To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database.

- MongoDB will create the database if it does not exist, and make a connection to it.

Creating a Collection

- To create a collection in MongoDB, use the Collection() method
- In MongoDB, a collection is not created until it gets content.

Insert a single document Into Collection

- To insert a record, or document as it is called in MongoDB, into a collection, use the insertOne() method.
- A document in MongoDB is the same as a record in MySQL
- The first parameter of the insertOne() method is an object containing the name(s) and value(s) of each field in the document you want to insert.
- It also takes a callback function where you can work with any errors, or the result of the insertion

Select the documents from collection:

- In MongoDB use the find and findOne methods to find data in a collection.
- Just like the SELECT statement is used to find data in a table in a MySQL database.

Find:

- To select data from a table in MongoDB, we can also use the find() method.
- The find() method returns all occurrences in the selection.
- The first parameter of the find() method is a query object.

THANK YOU

Aruna S
Department of
Computer Science and Engineering
arunas@pes.edu

NODE JS

Aruna S
Department of
Computer Science and Engineering

NODE JS

HTTP Module

S. Aruna
Department of Computer Science and Engineering

NODE JS

Web Server

- A Web Server is a software application which handles HTTP requests sent by the HTTP client, like web browsers, and returns web pages in response to the clients.
- Web servers usually deliver html documents along with images, style sheets, and scripts.
- Most of the web servers support server-side scripts, using scripting languages or redirecting the task to an application server which retrieves data from a database and performs complex logic and then sends a result to the HTTP client through the Web server.
- Apache web server is one of the most commonly used web servers. It is an open source project.

NODE JS

Web Application Architecture

A Web application is usually divided into four layers −

Client − This layer consists of web browsers, mobile browsers or applications which can make HTTP requests to the web server.

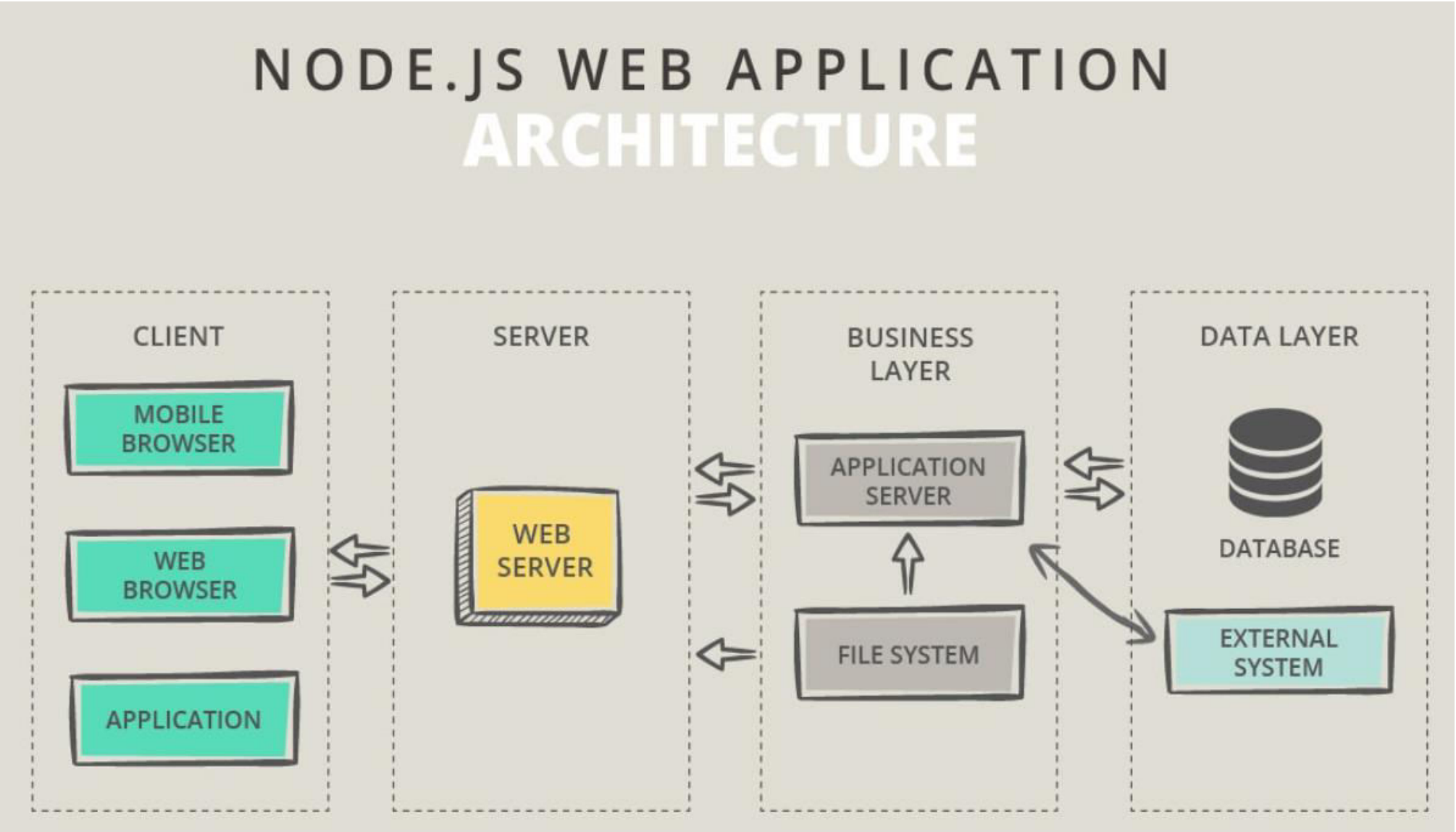
Server − This layer has the Web server which can intercept the requests made by the clients and pass them the response.

Business − This layer contains the application server which is utilized by the web server to do the required processing. This layer interacts with the data layer via the database or some external programs.

Data − This layer contains the databases or any other source of data.

NODE JS

Web Application Architecture



NODE JS

HTTP Module – Web Server using Node

The components of a Node.js application.

Import required modules – We use the **require** directive to load Node.js modules.

Create server – A server which will listen to client's requests similar to Apache HTTP Server. Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

Read request and return response – The server created in an earlier step will read the HTTP request made by the client which can be a browser or a console and return the response.

Creating Node.js Application

Step 1 - Import Required Module

We use the **require** directive to load the http module and store the returned HTTP instance into an http variable as follows

```
var http = require('http');
```

Step 2 - Create Server

- We use the created http instance and call **http.createServer()** method to create a server instance.
- Then we bind it at port 8088 using the **listen** method associated with the server instance.

```
http.createServer(function (request, response) {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end('Hello PES University\n');
 }).listen(8088);
  console.log('Server running at http://127.0.0.1:8088/')
```

Step 3 - Testing Request & Response

$node app.js

Verify the Output. Server has started.

Server running at http://127.0.0.1:8088/

NODE JS

URL Module

The URL module splits up a web address into readable parts.
To include the URL module, use the require() method:

```
var url = require('url');
```

Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

Node Examples > JS app.js > ...

```
1   var url = require('url');
2   var adr = 'http://localhost:8080/pesu.htm?year=2020&month=September';
3   var q = url.parse(adr, true);
4
5   console.log(q.host); //returns 'localhost:8080'
6   console.log(q.pathname); //returns '/pesu.htm'
7   console.log(q.search); //returns '?year=2020&month=September'
8
9   var qdata = q.query; //returns an object: { year: 2020, month: 'september' }
10  console.log(qdata.month); //returns 'september'
```

NODE JS

Web Client Using Node

A web client can be created using **http** module. A Screenshot of the example is below

```
var http = require('http');
var options = {
   host: 'localhost',
   port: '8081',
   path: '/index.htm'
};
var callback = function(response) {
   var body = '';
   response.on('data', function(data) {
      body += data;
   });

   response.on('end', function() {
      console.log(body);
   });
}
var req = http.request(options, callback);
req.end();
```

Node JS and MongoDB Connectivity

- Node.js can be used in database applications.
- One of the most popular NoSQL database is MongoDB.
- Download a free MongoDB database at https://www.mongodb.com.
- Node.js can use this module to manipulate MongoDB databases

require('mongodb');

Creating a Database

- To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database.

- MongoDB will create the database if it does not exist, and make a connection to it.

Creating a Collection

- To create a collection in MongoDB, use the Collection() method
- In MongoDB, a collection is not created until it gets content.

Insert a single document Into Collection

- To insert a record, or document as it is called in MongoDB, into a collection, use the insertOne() method.
- A document in MongoDB is the same as a record in MySQL
- The first parameter of the insertOne() method is an object containing the name(s) and value(s) of each field in the document you want to insert.
- It also takes a callback function where you can work with any errors, or the result of the insertion

Select the documents from collection:

- In MongoDB use the find and findOne methods to find data in a collection.
- Just like the SELECT statement is used to find data in a table in a MySQL database.

Find:

- To select data from a table in MongoDB, we can also use the find() method.
- The find() method returns all occurrences in the selection.
- The first parameter of the find() method is a query object.

THANK YOU

Aruna S
Department of
Computer Science and Engineering
arunas@pes.edu

MongoDB

Vinay Joshi
Department of
Computer Science and Engineering

MongoDB

File Based Systems

- Program defines and manages it’s own data

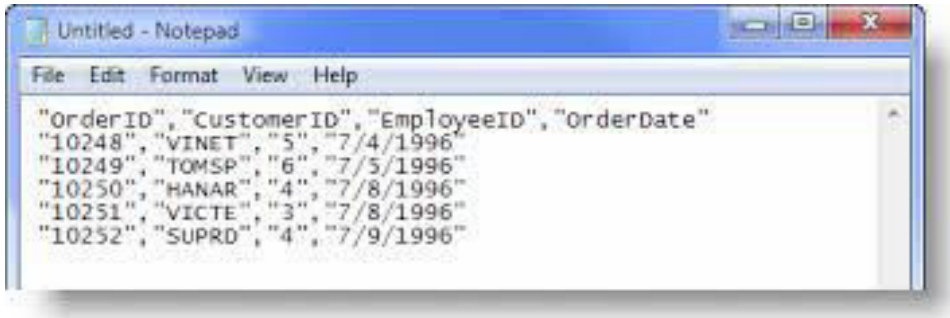
Limitations:

- Separation and isolation
- Duplication
- Program & data dependence
- Fixed queries
- Proliferation of application programs

MongoDB

File Based Systems

File (Typically a CSV file)



Database

Emp_name	Emp_Id	Emp_addr	Emp_desig	Emp_Sal
Prasad	100	"Shubhodaya", Near Katariguppe Big Bazaar, BSK II stage, Bangalore	Project Leader	40000
Usha	101	#165, 4th main Chamrajpet, Bangalore	Software engineer	10000
Nupur	102	#12, Manipal Towers, Bangalore	Lecturer	30000
Peter	103	Syndicate house, Manipal	IT executive	15000

MongoDB

Introduction

- Name comes from “Humongous” & huge data
- Written in C++, developed in 2009
- Creator: 10gen, former doublick

- Definition: MongoDB is an open source, document-oriented database designed with both scalability and developer agility in mind
- Instead of storing your data in tables and rows as you would with a relational database, in MongoDB you store JSON-like documents with dynamic schemas (schema-free, schemaless)

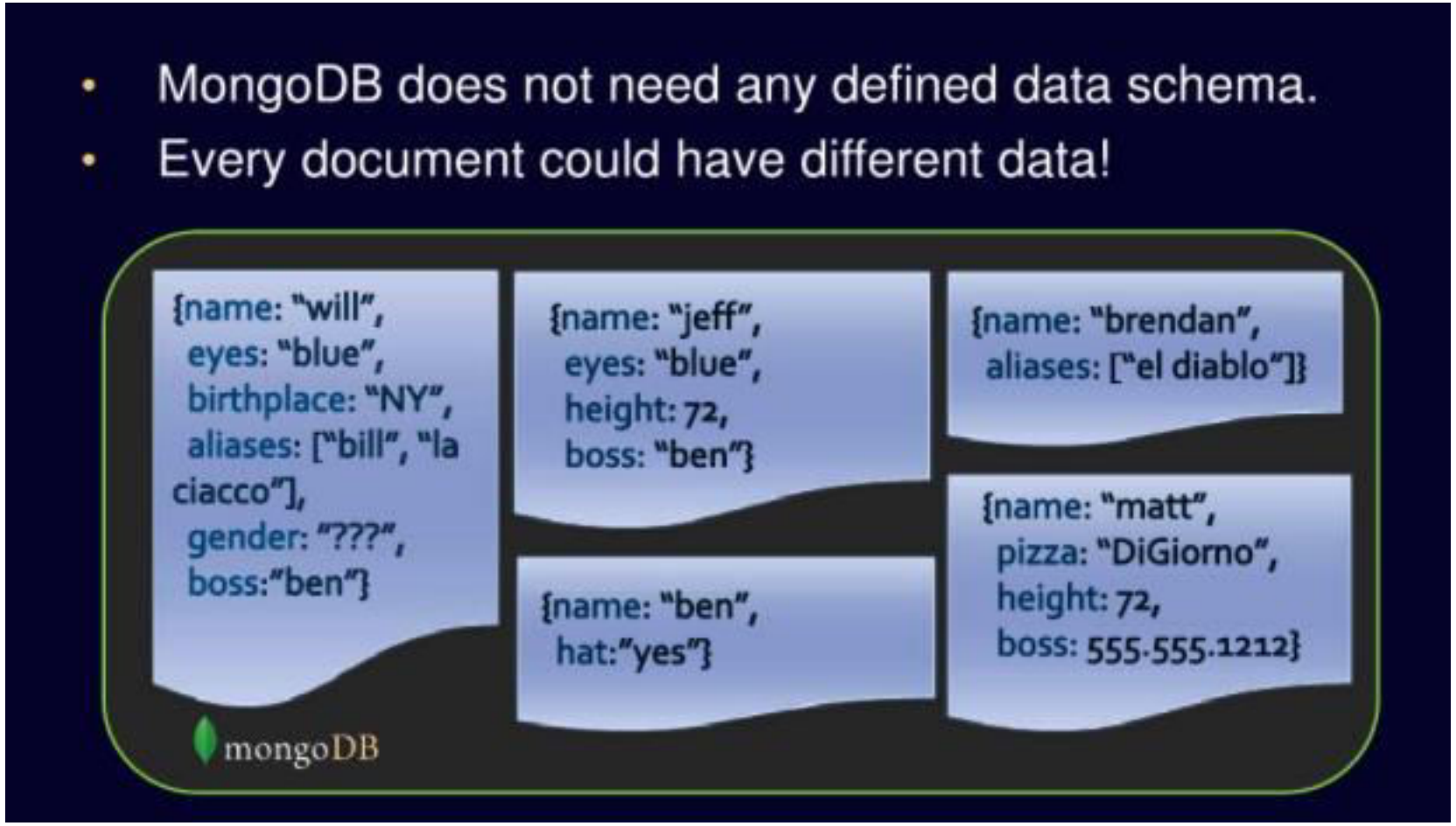
MongoDB

NoSQL Database

- Stands for **N**ot **O**nly **SQL**??
- Class of non-relational data storage systems
- Usually do not require a fixed table schema nor do they use the concept of joins to derive data from different tables

MongoDB – a NoSQL Database

- No Defined Schema (Schema Free or Schema Less)



MongoDB

Terms Mapping (DB vs. MongoDB)

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default _id key provided by mongodb)

MongoDB

Data Model – BSON Format

- BSON format (binary JSON)
- Developers can easily map to modern object-oriented languages without a complicated ORM layer.

```
{ author: 'joe',
  created : new Date('03/28/2009'),
  title : 'Yet another blog post',
  text : 'Here is the text...',
  tags : [ 'example', 'joe' ],
  comments : [
              { author: 'jim',
                comment: 'I disagree'
              },
              { author: 'nancy',
                comment: 'Good post'
              }
  ]
}
```

Remember it is stored in binary formats

```
"\x16\x00\x00\x00\x02hello\x00
 \x06\x00\x00\x00world\x00\x00"

"1\x00\x00\x00\x04BSON\x00&\x00
 \x00\x00\x020\x00\x08\x00\x00
 \x00awesome\x00\x011\x00333333
 \x14@\x102\x00\xc2\x07\x00\x00
 \x00\x00"
```

MongoDB

MongoDB Data Model

One ***document*** (e.g., one tuple in RDBMS)

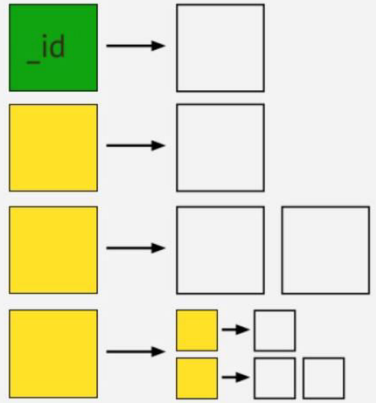
```
{
  name: "sue",          <-- field: value
  age: 26,              <-- field: value
  status: "A",          <-- field: value
  groups: [ "news", "sports" ]   <-- field: value
}
```

One ***Collection*** (e.g., one Table in RDBMS)

- **Collection** is a group of similar documents
- Within a collection, each document must have a unique Id

MongoDB Document

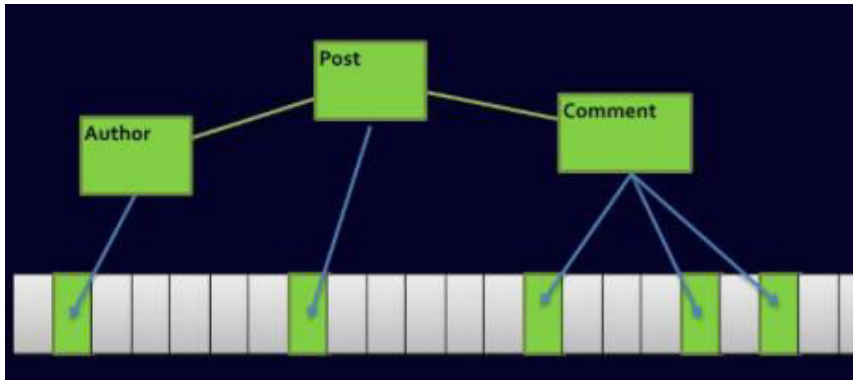
- **N-dimensional** storage
- Field can contain **many** values and **embedded** values
- Query on **any field & level**
- **Flexible** schema
- Optimal data locality requires fewer **indexes** and provides better **performance**



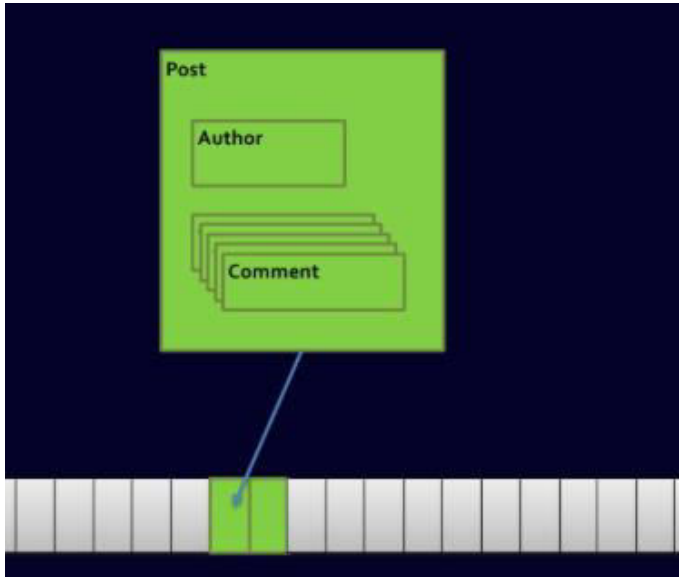
MongoDB

Complex Join Queries

Relational DBs



MongoDB



MongoDB

Getting Started…

Install it

Practice simple stuff

Move to complex stuff

Install it from here: http://www.mongodb.org

Manual: http://docs.mongodb.org/master/MongoDB-manual.pdf
(Focus on Ch. 3, 4 for now)

Dataset: http://docs.mongodb.org/manual/reference/bios-example-collection/

MongoDB

Data Operations

Create

db.collection.insert(<document>)

db.collection.save(<document>)

db.collection.update(<query>, <update>, { upsert: true })

Read

db.collection.find(<query>, <projection>)

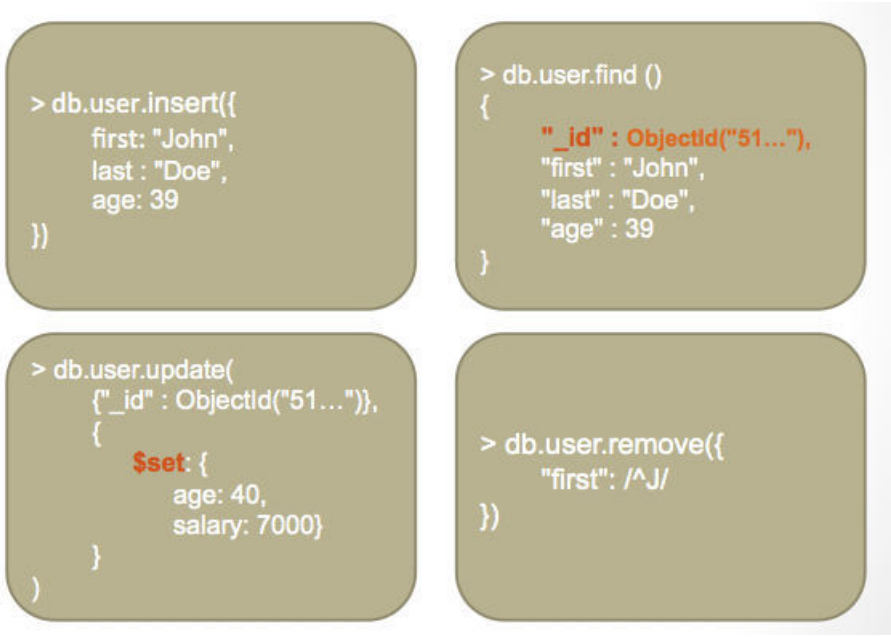
db.collection.findOne(<query>, <projection>)

Update

db.collection.update(<query>, <update>, <options>)

Delete

db.collection.remove(<query>, <justOne>)



MongoDB

Example Operations – Creation and Deletion

In RDBMS

```
CREATE TABLE users (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
```

```
DROP TABLE users
```

In MongoDB

Either insert the 1st docuement

```
db.users.insert( {
    user_id: "abc123",
    age: 55,
    status: "A"
 } )
```

Or create "Users" collection explicitly

```
db.createCollection("users")
```

```
db.users.drop()
```

MongoDB

Example Operations – Removal or Deletion

- You can put condition on any field in the document (even ***_id***)

```
db.users.remove(          <-- collection
   { status: "D" }        <-- remove criteria
)
```

The following diagram shows the same query in SQL:

```
DELETE FROM users         <-- table
WHERE   status = 'D'      <-- delete criteria
```

db.users.remove ( ) → Removes all documents from *users* collection

MongoDB

Example Operations – Update

```
db.users.update(                    <-- collection
   { age: { $gt: 18 } },            <-- update criteria
   { $set: { status: "A" } },       <-- update action
   { multi: true }                  <-- update option
)
```

Otherwise, it will update only the 1st matching document

Equivalent to in SQL:

```
UPDATE users          <-- table
SET    status = 'A'   <-- update action
WHERE  age > 18       <-- update criteria
```

MongoDB

Example Operations – Replace a document

New doc

```
db.inventory.update(
    { item: "BE10" },
    {
      item: "BE05",
      stock: [ { size: "S", qty: 20 }, { size: "M", qty: 5 } ],
      category: "apparel"
    }
)
```

Query Condition

For the document having item = “BE10”, replace it with the given document

MongoDB

Example Operations – Insert or Update?

```
db.inventory.update(
    { item: "TBD1" },
    {
      item: "TBD1",
      details: { "model" : "14Q4", "manufacturer" : "ABC Company" },
      stock: [ { "size" : "S", "qty" : 25 } ],
      category: "houseware"
    },
    { upsert: true }
)
```

The ***upsert*** option

If the document having item = “TBD1” is in the DB, it will be replaced
Otherwise, it will be inserted.

THANK YOU

Vinay Joshi
Department of
Computer Science and Engineering
vinayj@pes.edu

MongoDB Connectivity

Vinay Joshi
Department of
Computer Science and Engineering

Node JS and MongoDB Connectivity

- Node.js can use this native module to manipulate MongoDB databases
  require('mongodb');
- Alternately, use more sophisticated third party module like 'mongoose' that provides Object Data Modeling capabilities

Creating a Database

- To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database.

- MongoDB will create the database if it does not exist, and make a connection to it.

Creating a Collection

- To create a collection in MongoDB, use the Collection() method
- In MongoDB, a collection is not created until it gets content

Insert a single document Into Collection

- To insert document into a collection, use the insertOne() method
- A document in MongoDB is the same as a record in MySQL
- The first parameter of the insertOne() method is an object containing the name(s) and value(s) of each field in the document you want to insert.
- It also takes a callback function where you can work with any errors, or the result of the insertion
- To insert multiple documents at once, use insertMany() method

Select the documents from collection:

- In MongoDB use the find() and findOne() methods to find data in a collection.
- Just like the SELECT statement is used to find data in a table in a MySQL database.

Find:

- To select data from a table in MongoDB, we can also use the find() method.
- The find() method returns all occurrences in the selection.
- The first parameter of the find() method is a query object.

THANK YOU

Vinay Joshi
Department of
Computer Science and Engineering
vinayj@pes.edu

NODE JS

Aruna S
Department of
Computer Science and Engineering

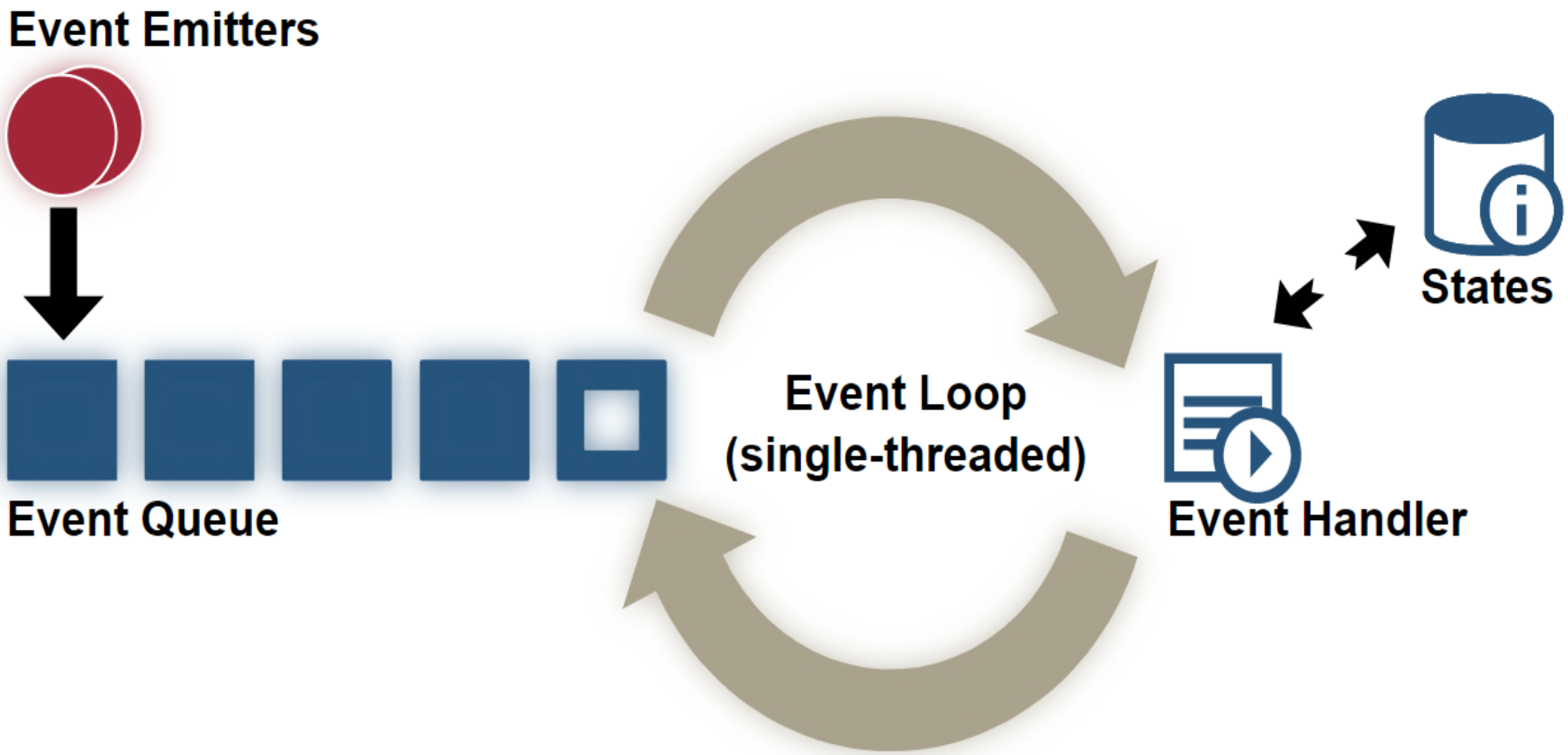
NODE JS

Event Loop and Event Emitter

S. Aruna
Department of Computer Science and Engineering

NODE JS

Events Architecture flow



NODE JS

Events

- Node.js is a single-threaded application, but it can support concurrency via the concept of **event** and **callbacks**.
- Every API of Node.js is asynchronous and being single-threaded, they use **async function calls** to maintain concurrency.
- Node uses observer pattern. Node thread keeps an event loop and whenever a task gets completed, it fires the corresponding event which signals the event-listener function to execute.

NODE JS

Event Driven Programming

- Node.js uses events heavily and it is also one of the reasons why Node.js is pretty fast compared to other similar technologies.
- As soon as Node starts its server, it simply initiates its variables, declares functions and then simply waits for the event to occur.
- In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.

NODE JS

EventEmitter Class

- The functions that listen to events act as **Observers**.
- Whenever an event gets fired, its listener function starts executing. Node.js has multiple in-built events available through events module and EventEmitter class which are used to bind events and event-listeners

```
var events = require('events');

// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();
```

EventEmitter Class

EventEmitter provides multiple properties like **on** and **emit**. **on** property is used to bind a function with the event and **emit** is used to fire an event.

Methods:

- addListener(event, listener)
- on(event, listener)
- once(event, listener)
- removeListener(event, listener)
- removeAllListeners([event])
- setMaxListeners(n)
- listeners(event)
- emit(event, [arg1], [arg2], [...])

Class Methods:

- listenerCount(emitter, event)

Events:

- newListener

event − String: the event name

listener − Function: the event handler function

This event is emitted any time a listener is added. When this event is triggered, the listener may not yet have been added to the array of listeners for the event.

- removeListener

This event is emitted any time someone removes a listener. When this event is triggered, the listener may not yet have been removed from the array of listeners for the event.

THANK YOU

Aruna S
Department of
Computer Science and Engineering
arunas@pes.edu

React and Node Integration

Vinay Joshi
Department of
Computer Science and Engineering

Node JS and React JS integration

There are two main approaches to create React applications on Node

- HTML approach
  - Write react code in HTML
  - Return the HTML file in response when a request URL contains .html
  - The react code will be executed by the browser
- React App approach
  - Execute react code at server end
  - Steps to follow:
    1. npx create-react-app <myapp> (myapp is the app name to be given)
    2. cd <myapp>
    3. npm start
    4. Modify src/App.js to create your own app

React Router

- SPA (Single Page Applications) need fewer reload of pages, faster page speeds so that it leads to better User Experience
- Traditionally, regular applications used the <a> tag to enable navigation from one page to another. This resulted in server requests and hence called server side routing
- With React Router, we achieve client side routing using the <Link> and <Route> component

React Router - Syntax


```
import {BrowserRouter as Router, Route, Link} from 'react-router-dom';

<Router>
    <Link to="/">Home</Link>
    <Link to="/contact">Contact</Link>
    <Link to="/about">About Us</Link>
    <Route path="/" component={Home} />
    <Route path="/about" component={About} />
    <Route path="/contact" component={Contact} />
</Router>
```

React Router – Styling the links

App.js

```
import {StyledLink} from ‘./styles.js’;
<Router>
    <StyledLink to=“/”>Home</StyledLink>
    …
</Router>
```

styles.js

```
import styled from ‘styled-components’;
import {NavLink} from ‘react-router-dom’;

export const StyledLink = styled(NavLink)`
    margin-right: 5px;
`;
export default StyledLink;
```

THANK YOU

Vinay Joshi
Department of
Computer Science and Engineering
vinayj@pes.edu

React and Node Integration

Vinay Joshi
Department of
Computer Science and Engineering

Node JS and React JS integration

There are two main approaches to create React applications on Node

- HTML approach
  - Write react code in HTML
  - Return the HTML file in response when a request URL contains .html
  - The react code will be executed by the browser
- React App approach
  - Execute react code at server end
  - Steps to follow:
    1. npx create-react-app <myapp> (myapp is the app name to be given)
    2. cd <myapp>
    3. npm start
    4. Modify src/App.js to create your own app

React Router

- SPA (Single Page Applications) need fewer reload of pages, faster page speeds so that it leads to better User Experience
- Traditionally, regular applications used the <a> tag to enable navigation from one page to another. This resulted in server requests and hence called server side routing
- With React Router, we achieve client side routing using the <Link> and <Route> component

React Router - Syntax


```
import {BrowserRouter as Router, Route, Link} from 'react-router-dom';

<Router>
    <Link to="/">Home</Link>
    <Link to="/contact">Contact</Link>
    <Link to="/about">About Us</Link>
    <Route path="/" component={Home} />
    <Route path="/about" component={About} />
    <Route path="/contact" component={Contact} />
</Router>
```

React Router – Styling the links

App.js

```
import {StyledLink} from ‘./styles.js’;
<Router>
    <StyledLink to=“/”>Home</StyledLink>
    …
</Router>
```

styles.js

```
import styled from ‘styled-components’;
import {NavLink} from ‘react-router-dom’;

export const StyledLink = styled(NavLink)`
    margin-right: 5px;
`;
export default StyledLink;
```

THANK YOU

Vinay Joshi
Department of
Computer Science and Engineering
vinayj@pes.edu