Node JS

AGENDA

What is NodeJS?

How does Node Work?

Why NodeJS?

Architecture Diagram

NodeJS vs Other Similar frameworks

What is v8 ?

Installing Node.js and NPM

Blocking(Synchronous) vs Non-Blocking(Async)

Event Loop

NPM and commands

AGENDA Conti...

- Node-modules
- Package.json
- REPL (Read-Eval-Print-Loop)
- Node Fundamentals
  - Modules
  - Callbacks
  - Events
  - Streams
  - Buffers
  - File System
  - Error handling

AGENDA Conti...

Creating a server with the HTTP module
Building Command Line Apps
Assignments

What is Node JS?

Open-source platform =Application + Server

Built on Chrome's JavaScript engine V8 to execute code

Event-driven and non-blocking I/O.

Coding in Javascript.

Single Threaded .

DIRTY(Data Intensive RealTime) Applications

Used by IBM,Microsoft,Yahoo!,Walmart,Groupon, SAP,LinkedIn, PayPal,and GoDaddy

Use-Case Problem

Restaurants Problem

Use-Case Solution 1

Serve on the Basis of order completion till that time wait in restuarent .
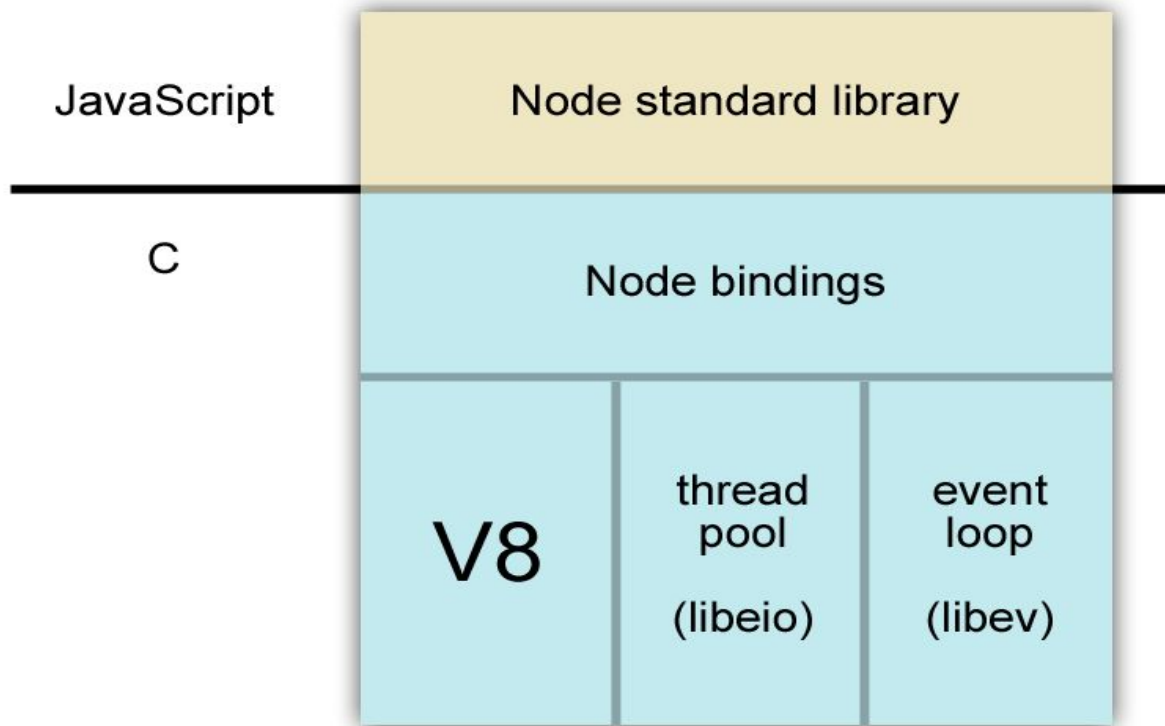
Use-Case Solution 2

Opening Many Restaurents

Why Node JS ?

Same coding standard at client and server side i.e jS .

Code Reuse .

Context switching .

It's Fast .

Real-time .

Streaming data .
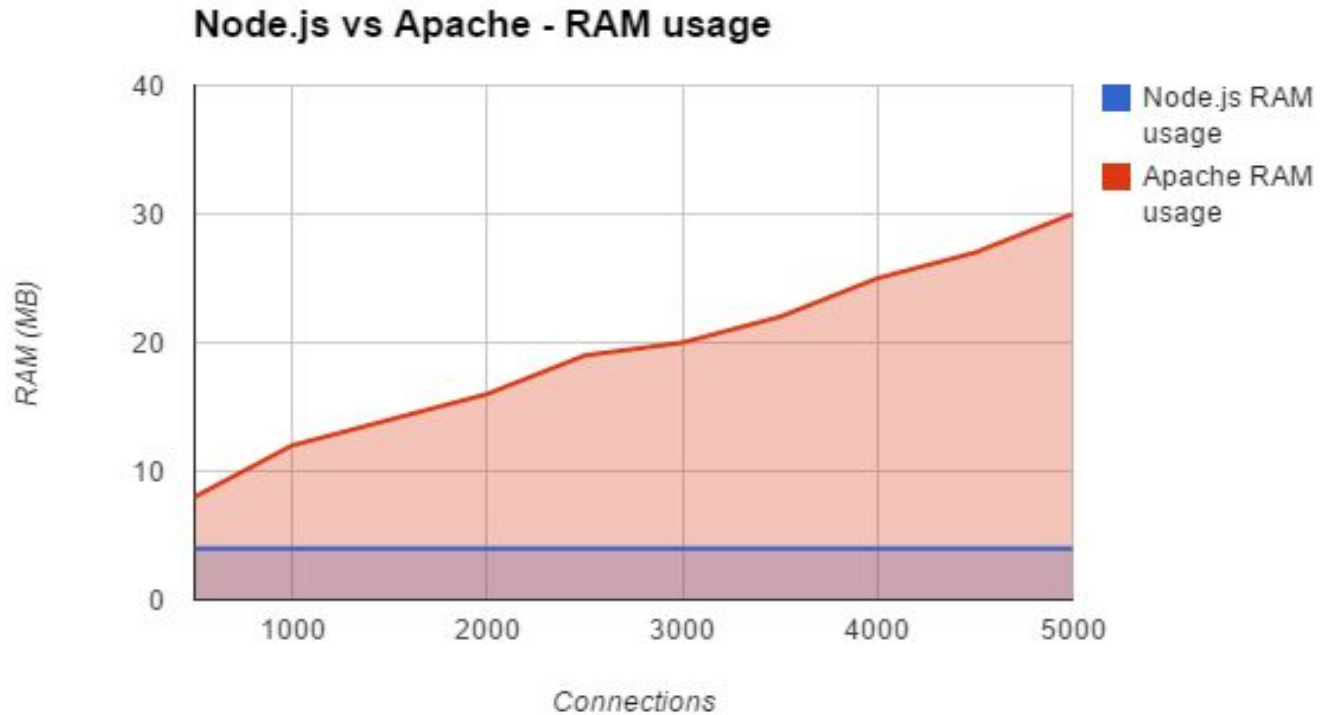
Architecture Diagram

NodeJS vs others servers

Performance

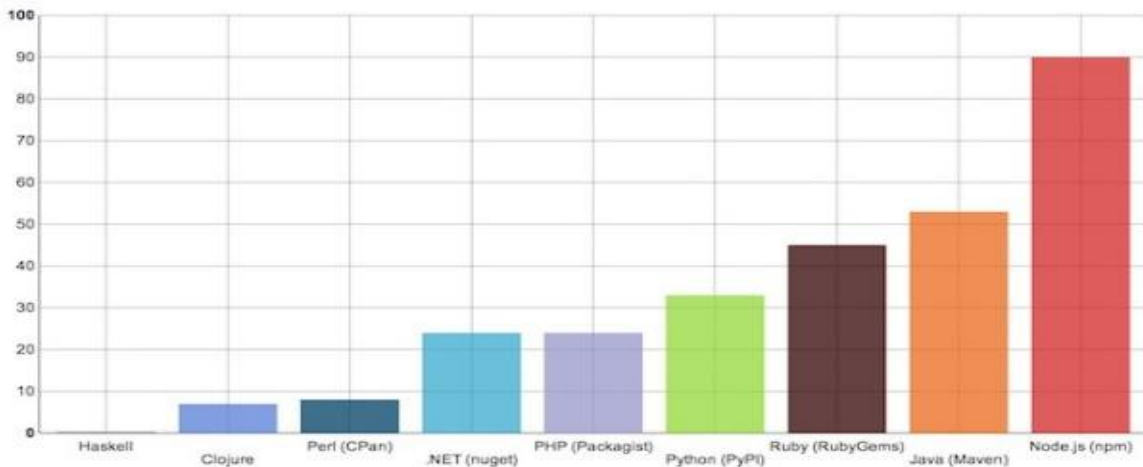| Performance		
Number of iterations	Node.JS	PHP
100	2.00	0.16
10,000	3.00	9.52
1,000,000	13.00	1117.21
10,000,000	138.00	10461.29

NodeJS vs others servers

Memory Usage

NodeJS vs others servers

No of modules (library)

node.js sec: npm modules

Comparison to other langs (mods/day):

Maciej Lasyk, node.js security

What is v8 ?

Google's open source high-performance JavaScript engine .

Written in C++ .

Used in Google Chrome .

Implements ECMAScript .

Executes javascript code .

https://developers.google.com/v8/?hl=en

Installing Node & npm

Blocking vs Non-Blocking IO

Blocking Use Case

Blocking IO is something like "you waiting for your someone to join you on a date, you wait for her indefinitely"

Blocking vs Non-Blocking IO

Non-Blocking Use Case

Non-blocking IO is like "you have asked someone to join you on a date, but you are not sure if she turns up so early, so you decide to do other works pending, or sometimes you get bored and may try asking another girl for a date".

Blocking vs Non-Blocking IO

```
var fs = require('fs');
var contents =
fs.readFileSync('users','utf8');
console.log(contents);
console.log("Hello Node\n");
```

```
var fs = require('fs');
var contents = fs.readFile('./users','utf8',
function(err,contents){
    console.log(contents);
});
console.log("Hello Node\n");
```
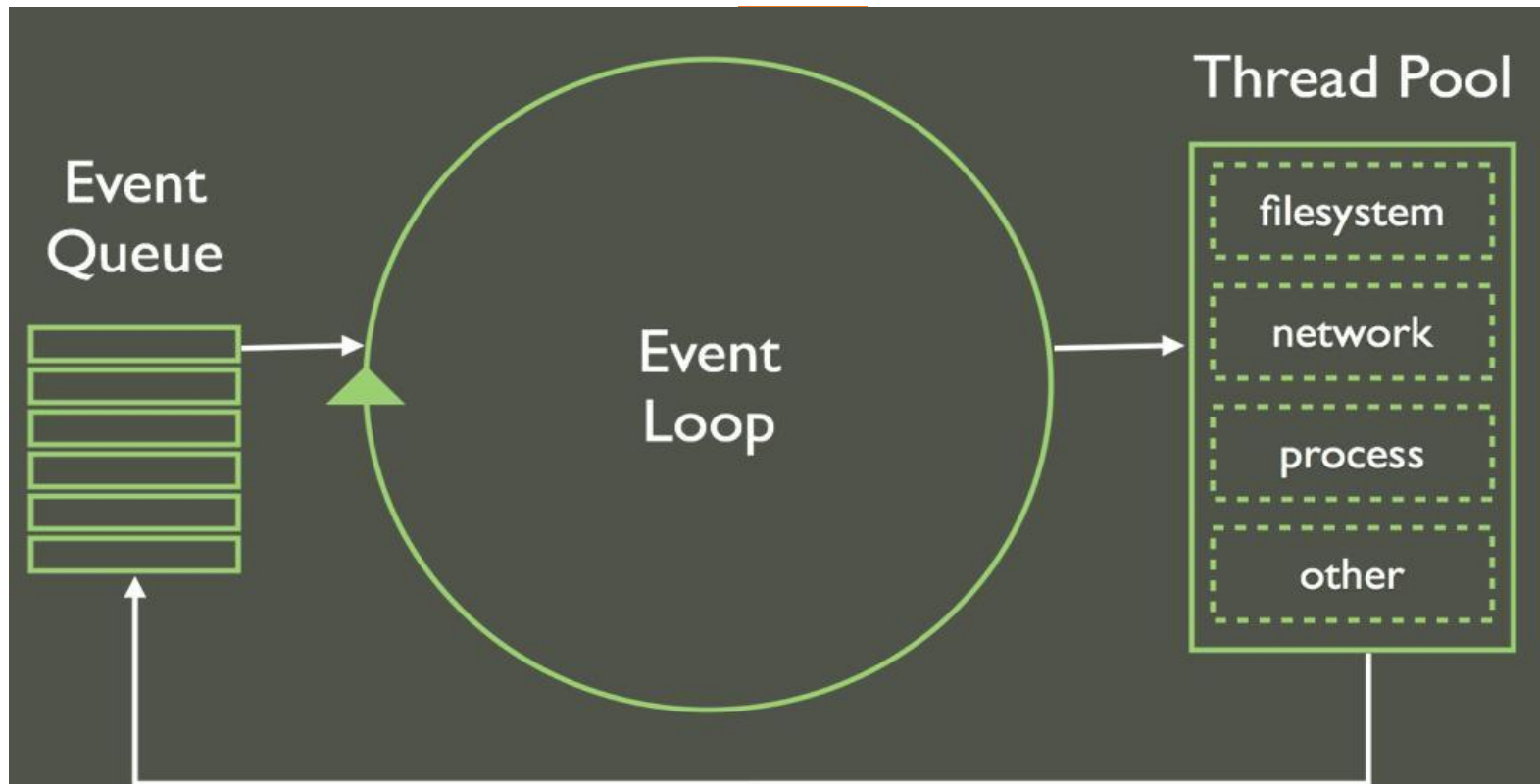
Event Loop

Event loop is single-threaded .

Event loops is core of javascript , all events , requests are handled by event loop.

Works on event driver framework.

Handle highly concurrent requests .

Blocking the event loop can have catastrophic effects on the Node application.

Event Loop

NPM commands

NPM is an online registry for open-source node.js projects, modules, resources, etc
Provides a command line interface (CLI) for interacting with the registry.
NodeJS package manager : install , update all node module (3rd party libraries)
Getting help: **npm help**
Installing stuff: **npm install [-g] [package-name]**
Showing all node modules : **npm ls**
Updating packages : **npm update [-g] [package-name]**
Making a package.json file**.**
Use **npm init [--force]** to generate package.json

Node modules

The core modules are defined within Node.js's source and are located in the lib/ folder.
Core modules are always preferentially loaded
Few core node-modules:
    1)Cluster
    2)Console
    3)Crypto
    4)DNS
    5)Errors
    6)Events
    7)File System
    8)Globals
    9)HTTP

Package.json

All npm packages contain a file, usually in the project root, called package.json
Holds various metadata relevant to the project

```json
{
  "name" : "underscore",
  "description" : "JavaScript's functional programming helper library.",
  "author" : "Jeremy Ashkenas <jeremy@documentcloud.org>",
  "dependencies" : [],
  "repository" : {"type": "git", "url": "git://github.com/documentcloud/underscore.git"},
  "main" : "underscore.js",
  "version" : "1.1.6"
}
```

REPL

Read-Eval-Print-Loop (REPL) is available both as a standalone program and easily includable in other programs

The REPL provides a way to interactively run JavaScript and see the results .

Type "node" in your terminal. Try following code now

    1)1+2

    2)console.log("hello world");

    3)var name='vishnu';

    4)console.log(name);

Node Fundamentals -Modules

Modules

Building block of a node application

Seprate our components based on business logic.

Modules can be single files or directories containing one or more file.

Three types of modules

1 . Core node modules (installed using npm cmd)

2.   Thirdy party node modules(available in node core libraries i.e default module)

3.   Custom modules. (we make these modules).

Node Fundamentals-Modules

How to make a custom modules .

Use these steps

1)Create a file named "demo-module.js"

2)Write these lines into this file :

exports.myFun = function() {

console.log("demo-module data");

}

3)Create a new file name "main.js" :

1)Write these lines in it :

2)**var demoModule=require("./demo-module");**

3)**demoModule.myFun();**

Node Fundamentals-Modules

Exports vs module.exports

  Exports : if you want to expose more than one function or variable, then we can set property of object called **"exports".**

  Module.exports : if you want to return single function/variable/object.

  KEY POINT TO NOTE :

1) node perform syncronous lookup in order to locate file contents of module

2) When you use require("filename.js") , extention ".js"is optional .

3) If you create module that populates both **"exports"** and **"module.exports"**

4),then **"module.exports"** will be returned and exports will be ignored

Node Fundamentals-Modules

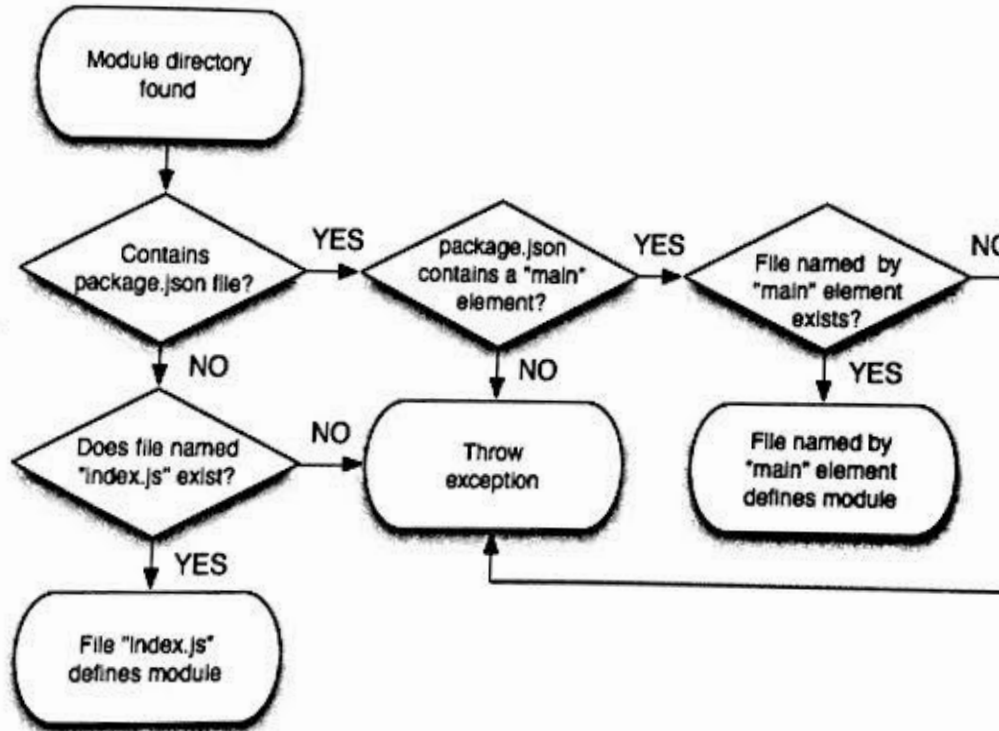Exports is only set on object/functions not on class .

For eg :

```
var Currency=function(dollar){
this.dollar=dollar;
}
Currency.protoype.roundTwoDecimals=function(amount){
return Math.round(amount*100)/100;
}
exports=Currency  //This line will give you error instead of this use "module.exports"
```

Node Fundamentals-Modules

Steps to find a module :

Node Fundamentals-Callbacks

Callbacks : a function which is passed as an argument to a async function.
It is main concept behind asyncronous programming.
For example :

```
function myFun(name, cb){
    console.log("myfun");
    cb();
}

myFun("Mayank",function(){
    console.log('Callback called');
})
```

Node Fundamentals-Callbacks

In Node file handling :

```
var fs=require('fs');
fs.readFile('filename',myFun);
function myFun(err, content){
    if(err)
    console.log(err);
    console.log(content);
}
```

Node Fundamentals-Events

Events : change in state of an object.

At Node Side : Event Module

```
// Import events module
var events = require('events');
// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();
// Bind the connection event with the listner1 function
eventEmitter.addListener('connection', listner1);
// Bind the connection event with the listner2 function
eventEmitter.on('connection', listner2);
// Fire the connection event
eventEmitter.emit('connection');
```

Node Fundamentals-Events

Event module Methods :

    1)addListener(event, listener)

    2)on(event, listener)

    3)once(event, listener)

    4)removeListener(event, listener)

    5)removeAllListeners([event])

    6)emit(event, [arg1], [arg2], [...])

Node Fundamentals-Streams

Streams are objects that let you read data from a source or write data to a destination in continous fashion.

In Node.js, there are four types of streams.

1)**Readable -** Stream which is used for read operation.
2)**Writable -** Stream which is used for write operation.
3)**Duplex -** Stream which can be used for both read and write operation.
4)**Transform -** A type of duplex stream where the output is computed based on input.

Node Fundamentals-Streams

Readable stream :

```
var fs=require('fs');
    // Create a readable stream
    var readerStream = fs.createReadStream('input.txt');
    // Set the encoding to be utf8.
    readerStream.setEncoding('UTF8');
    // Handle stream events --> data, end
    var data="";
    readerStream.on('data', function(chunk) {
        data += chunk;
    });
    readerStream.on('end',function(){
      console.log(data);
    });
```

Node Fundamentals-Streams

Writable stream :

```
var fs = require("fs");
var data = 'Simply Easy Learning';
// Create a writable stream
var writerStream = fs.createWriteStream('output.txt');
// Write the data to stream with encoding to be utf8
writerStream.write(data,'UTF8');
// Mark the end of file
writerStream.end();
```

Node Fundamentals-Buffer

Buffer : similar to an array of integers but corresponds to a raw memory allocation outs the V8 heap.

Buffer class is a **global class** and can be accessed in application without

importing buffer module.

Creating Buffers :

```
var buf = new Buffer(100);
//writing into a buffer
len = buf.write("Simply Easy Learning");
console.log("Octets written : "+  len);
  //reading from a buffer
  buf.toString([encoding][, start][, end])
```

Node Fundamentals-File handling

Node File System (fs) module can be imported using following syntax:

var fs = require("fs")

Few methods of fs

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

    fs.stat(path, callback)

        stats.isFile() Returns true if file type of a simple file.

        stats.isDirectory()   Returns true if file type of a directory.

        stats.isBlockDevice() Returns true if file type of a block device.

        stats.isCharacterDevice() Returns true if file type of a character device.

        stats.isSymbolicLink()Returns true if file type of a symbolic link.

        stats.isFIFO()   Returns true if file type of a FIFO.

        stats.isSocket()   Returns true if file type of asocket.

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

fs.read(fd, buffer, offset, length, position, callback)

fs.close(fd, callback)

Node Fundamentals-Error handling

Error handling is a pain .

Easy to get by for a long time in Node.js without dealing with many errors correctly .

Building robust Node.js apps requires dealing properly with errors

Use **try catch** block: to handle errors.

Throw error : to throw customise error

process.on('uncaughtException')-To handle unhandled exception which stops your node server.

Node Fundamentals-HTTP SERVER

http://www.sitepoint.com/creating-a-http-server-in-node-js/

Node Fundamentals-GLOBAL OBJECTS

- __dirname .
- __filename
- setTimeout(cb, ms).

Node Fundamentals-Command line app

```
console.log(process.argv);
var userArgs = process.argv.slice(2);
```

Assignment

- Get the single line comments in different file.

- Read package.json file
  - Find the count of dependencies and devdependencies.
  - Find the common dependencies btw dependencies and devdependencies.
  - Write the above results in another file.

Package.json

Thank you