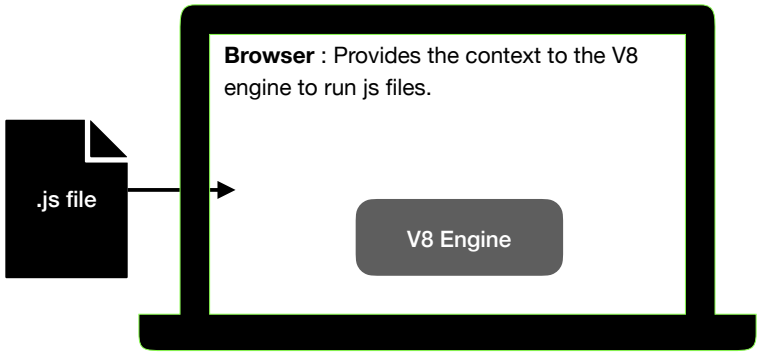
Part 1: Basics

Agenda

- ☑ Understand Node.js
- ☑ Understand and use Node.js module system
- ☑ Code organization, export and import
- ☑ WebServer using Node.js
- ☑ Writing a command line utility app.

What is Node.js??

It's a Javascript Runtime built on Chrome V8 engine. Different browser have different JS runtime which make them slow or faster than one another.
Chrome's runtime is built on V8 engine.Chrome provides the context (like window) to the engine to run the js files within the browser.



So some people thought to take the chrome V8 engine out of the browser and have it run on an OS just like any other programming languages Runtime environment like JRE for Java.

So just as they thought they took the V8 engine out of the chrome browser or Dom context and put it within a different context called NodeJS context.

The NodeJS context has the access to network apis, file system, os apis, core os context and not restricted only to DOM context.Now your js file can run all those code that other programming languages can to communicate and with OS.

So the bottom line is if can pull the V8 engine out of the browser's limited context and give it availability to all the core os apis then the small realm within which js was supposed to run become larger.That is what node js has done.

NVM: Node Version Manager

You can install node through nvm.Not only install you can juggle between different versions of node in your environment.You can do it without removing the current version and then re-installing the new. Just use the command **nvm install [version u prefer]** every time you want to change version of node.

Using the Node REPL

REPL: Read Evaluate Print Loop
It is a format of interacting with the interpreters which gives you a prompt to write.Then it reads, evaluate, print the execution result and iterate this process in a loop.

Node by default has a REPL built into it.Way to start REPL in terminal is simple just type node hit enter you will see the prompt.

When and when not to use Node.js

Node.js shines here

- ☑ Non blocking
- ☑ Event Driven
- ☑ Data Intensive
- ☑ I/O intensive

Node.js is single threaded.Although Node.js does have the concept of worker threads so this is not always true.But it helps us to understand Node.js as single threaded since there is only one “application thread” available.

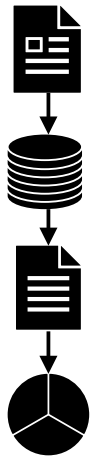
A single thread doing all the tasks might sound crazy comparing to our knowledge of other programming languages.

Code

DB

Model

View



A single thread runs through this queue of events and executes task.One by one events are handles from the queue

New tasks are queued up for execution.

The way this insertion and order management of event works is through an **event loop**.
Event loop is the loop of execution

Think about the following case :

```
for(I=0; I<10000000000;i++){
  // heavy processor intensive work
}
```

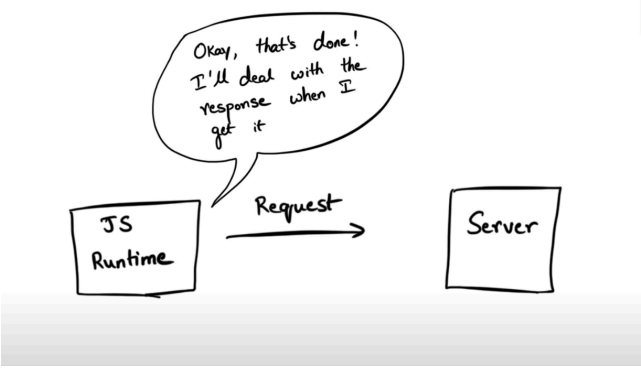
Here only one thread is busy doing a work which is heavy in terms of processor utilizations and everything else is waiting for this to end.

Well for these cases Node.js might not be a good choice where you frequently need to do processor intensive work which is synchronous and might block the main single thread.

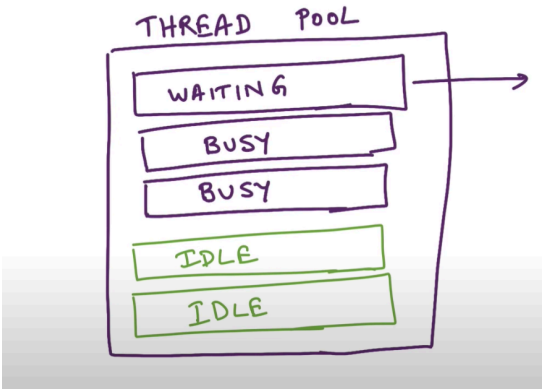
So when we should use Node.js?

Is db calling/ api request processing/ I/O operations are not heavy? Do they not block the single thread?

No.Since these operations are asynchronous, they do not block the thread from doing something else.JS runtime provides asynchronous facilities.They make a request and then continue executing the next event from the queue.When response is ready Node.js will deal with that at that point of time which is done through callback function.

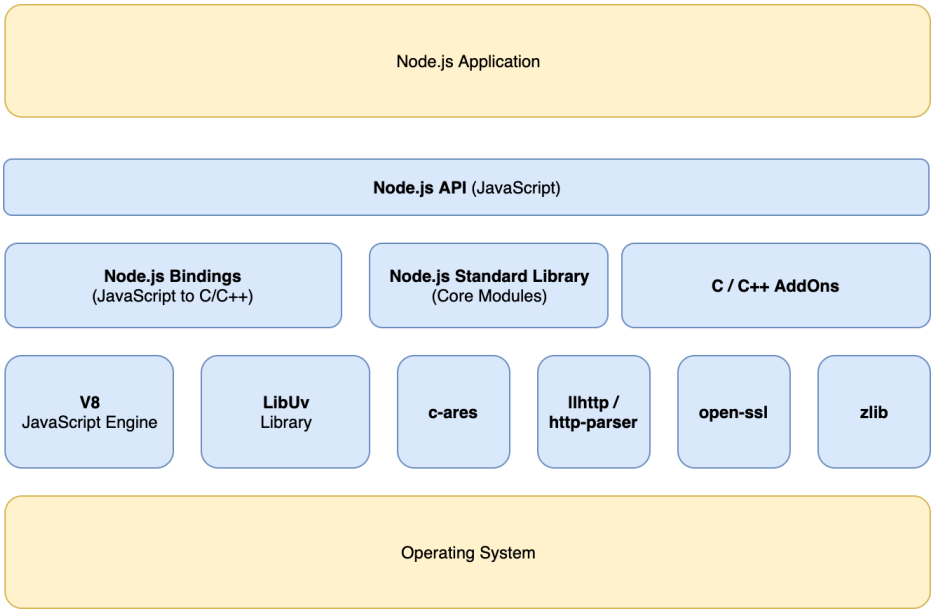


Comparing to multithreaded : Thread pull's thread waits when there is an interface with the external system. So if something else comes up another thread is assigned from the pool to serve that task.



Node.js really single threaded??

When we are saying node.js is single threaded we mean the application thread.There is one main application thread for running the whole application.The application sit onto of node runtime which is developed in C/C++ which possibly could be multithreaded.



Lets see the architecture of node.js from the below image:

WHEN WE SAY SINGLE THREADED WE MEAN THE TOP APPLICATION LAYER

So where Node.js does-not shine:

So non blocking, event-driven, data intensive or I/O intensive applications do perform better as they do not use internal resources that much.These applications mainly interface with the external resources in an asynchronous manner and serves response through call-back.

So Node.js is not good for the below applications:

- Data Calculation
- Processor Intensive
- Blocking Operation

So these are the cases where Node.js will perform poorly.As these are all processor intensive synchronous operation which will block the single thread.Node.js will not be able to pass these task to external resources and return the result as response like an asynchronous task.It actually has to do the job and then move to the next task form the event queue as a result the single thread will be blocked.

Example Apps where Node.js is suitable:

- **Web Server** : As node.js uses the callback model to serve as many request as possible from the event loop.

- **Real time server:** like web socket connection.the way node.js handles this sort of concurrent connection is using the asynchronous model, event driven model.So when there is a concurrent connection that needs attention only then it pays attention to it.
- **APIs fronting NoSql DB :** as JS has a very good way of dealing with these kind of flexible models.
- **Command Line utilities :** light processor intensive cli are ideal to build with node.js.
- **Build tooling**

Node modules intro:

Every JS file is a module of its own.
require('relative path to file') is one way of executing one JS file in another JS file.
This will execute the imported JS file but you cannot access any method or property of that imported file.
Because In JS every module is encapsulated by default.

Module exports in Node.js explained

Exporting through module.exports from area_module.js

```
function areaOfRectangle(length, height) {
    return length * height;
}
function areaOfCylinder(radius, height) {
    return 2 * Math.PI * radius * height + 2 * Math.PI * radius;
}
function areaOfTriangle(base, height) {
    return (base * height) / 2;
}
function areaOfCircle(radius) {
    return Math.PI * Math.pow(radius, 2);
}
//Recommended Way of exporting
module.exports.areaOfCircle = areaOfCircle
//Another way of exporting
module.exports.areaOfPlanes = {
    areaOfTriangle,
    areaOfRectangle
}
//Short hand for module.exports is exports
exports.areaOfCylinder = areaOfCylinder;
```

Importing area_module.js in index.js file Using require

```
let area_module = require('./area_module');
console.log('Area of circle is', area_module.areaOfCircle(10));
console.log('Area of triangle is',
area_module.areaOfPlanes.areaOfTriangle(10, 10));
console.log('Area of rectangle is',
area_module.areaOfPlanes.areaOfRectangle(10, 10));
console.log('Area of Cylinder is',area_module.areaOfCylinder(10,
10));
```

module.exports can be replaced with **exports** keyword which is a short-hand for module.exports
You must use . Property to make a key and assign something property to it when using **exports**.

In modern js you can construct any object by using {} bracket notation but you do not necessarily have to use key value. JS will make the key same name as property name like below:

```
module.exports.areaOfPlanes = {
    areaOfTriangle,
    areaOfRectangle
}
```

You can destruct or decompose the object like below

```
let { areaOfTriangle, areaOfRectangle } = require('./
area_module').areaOfPlanes;
```

Require with node apis

https://nodejs.org/docs/latest/api/ this side holds all the apis that comes with node.

What is npm ???

The letters npm stand for "**node package manager**". When you are working on a JavaScript project, you can use npm to install other people's code packages into your own project. ... npm is a CLI tool you install on your computer.

You might need useful third party packages/libraries to be integrated in your project to make your work simpler.The question is what is the ideal way to do it in JS/Node projects.

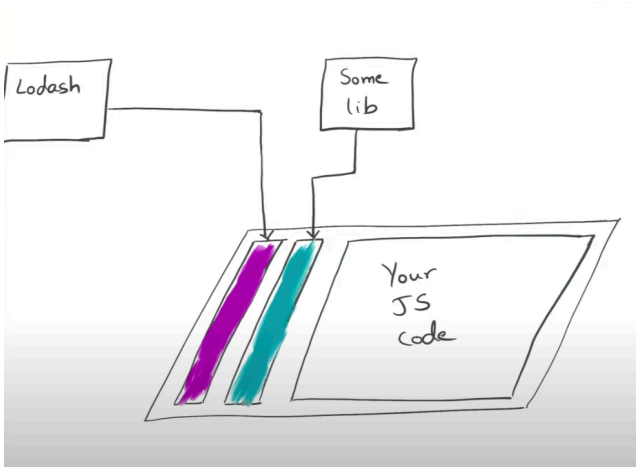
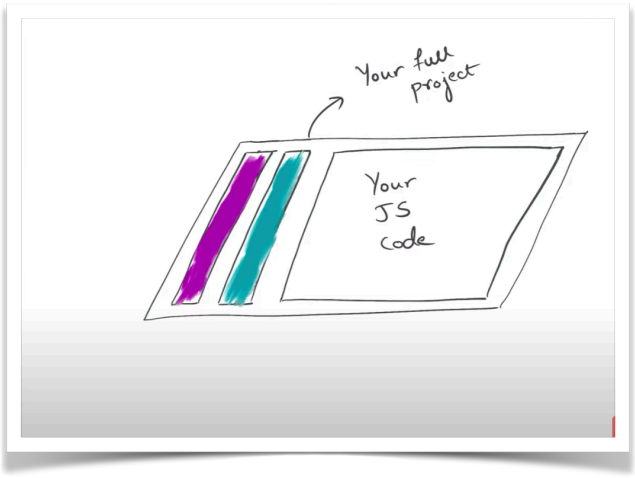
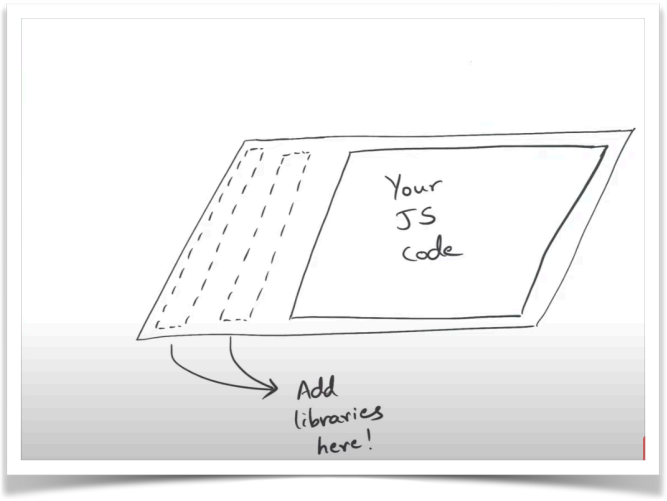


FIG: ONE WAY IS BY DOWNLOADING THE SRC FILES BUT IT HAS PROBLEMS

You can obviously download the src code and require it into your JS file and use it but that process will be much difficult to evolve and sustain as there are concepts like documentations, handling new updates or version to original code etc.



What if you have many dependency src files and you want to handover the project to some other developer.You have to copy all of them and make them a part of your project and handover them along with the src files.



Or
You can say these are the libraries I have go download them.

But it this process will be almost impossible to maintain as there are issues like versioning, updating src file, security check etc.

NPM is a rescuer in this scenario.

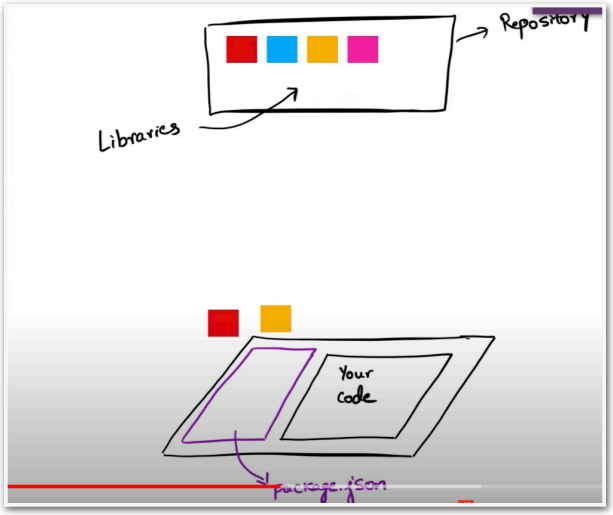
NPM provides a standard way of using third party libraries into your JS projects.

It has a repository online containing various third party libraries and manages by a company called **npm, Inc**

Using npm when you create a project npm will add file to your project called **package.json.**

package.json has many other utilities but it's main job is to manage all the dependencies (third party libraries/package) of your project.

Now your project is mobile and if you want to share your project you just need to share the **package.json** along with the code and others can download them from npm online repo.



So npm is managing multiple things at once like
Versions, libraries, making sure this versioning is consistent and development process is easy.

So npm is many things in one like
It is a CLI through which you can create, install and manage dependencies,
It is an online repo of libraries,
The organization npm inc which manages this whole infrastructure.

https://www.npmjs.com/about see this.

Starting a new project using npm

NPM comes with node installation. Check with npm -v.
We will see how we can create a project with npm.We will create a command line utility which will convert your current time to a different timezone. Codes will. Be found under code_files/timezone folder for the timezone project.

To start a project run commands

```
mkdir timezone
npm init.
```

To obstruct the command node filename.js we will create an alias command in the **package.json** file under script object.Our script object will look like below:

```
"scripts": {
        "test": "echo \"Error: no test specified\" && exit 1",
        "start": "node index.js”,
        "my-custom-script": "ls"
}
```

These scripts you see here are special.They are called **life-cycle script** so you can run them by **npm script-name.**
But to run any custom script you have to type **npm run script-name.**
E.g : npm run my-custom-script.

Installing and using libraries with npm

Let's add moment.js library to our timezone project.
Run this command **npm install moment —save**
To use the moment module we have to import the module by using require function.
The convention for node apis and third party apis is to use the name of the module in require function.

```
let moment = require('moment');
```

So whatever that module exports will be saved into the local variable moment.

Writing command line script in Node.js from scratch :

You can find the code under **code_files/timezone** folder.

Understanding callbacks using Node.js :

Synchronous operations are very costly for single threaded model like node js.If a synchronous operation takes a while all the other instructions after that operation needs to wait to be executed.As a result the whole program is halted for that one synchronous operation to be completed.Which is not a good idea.

In these cases you would want to use async operations.Callbacks are used to support the asynchronous way of execution.

Whenever you are passing a callback function to an api its a clear clue that this is an async api.

Example of how callback works is provided in the code files in **callback_examples.js** file under **code_files** folder.