CS 4220

- Current Trends in Web Design & Development -

Prof. Cydney Auman

AGENDA

01 Intro to Node.js

02 Node.js Event Loop

03 Node.js Modules

04 NPM

What is Node.js?

Node.js is an open-source, cross-platform runtime environment and allows for executing JavaScript code outside a browser. **Node.js** supports backend development and server-side development in JavaScript.

The main idea is that Node.js uses an event-driven, non-blocking I/O model to remain lightweight and efficient. I/O refers primarily to interaction with the system's disk and networking (HTTP and Database).

Node.js is *not* a framework and it's *not* a programming language.

Node.js Event Loop

There is only a single thread that executes JavaScript code and this is also where the **Event Loop runs**.

The **Event Loop** is what allows JavaScript running on Node.js to perform async or non-blocking operations . The event-driven architecture makes use of callbacks and promises.

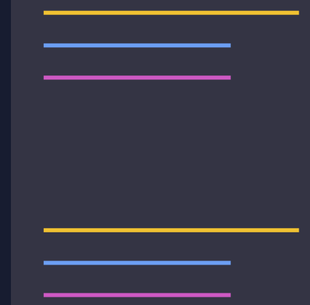
Whenever we use async functionality such as timers, file reads/write, network requests, etc - **an event will be fired** and then the associated callback will be invoked.

When Node.js starts - it automatically initializes the Event Loop. The Event Loop runs as long as the application has code that needs to be executed.

Node.js Architecture

Node.js uses an Event-Driven Architecture: it has an Event Loop for orchestration and a Thread Pool for expensive tasks.

JavaScript Code



Node.js

JS and C/C++

V8

- Interprets and executes JavaScript source code, handles memory allocation for objects, and garbage collects.
- Written in C++ and JS

libuv

- Provides Node.js access to the underlying operating system, file system, networking, and more.
- Implements two key features of Node.js - the event loop and the thread pool.
- Written in C

c-ares | llhttp | openssl | zlib

Event Loop, Call Stack & Queue

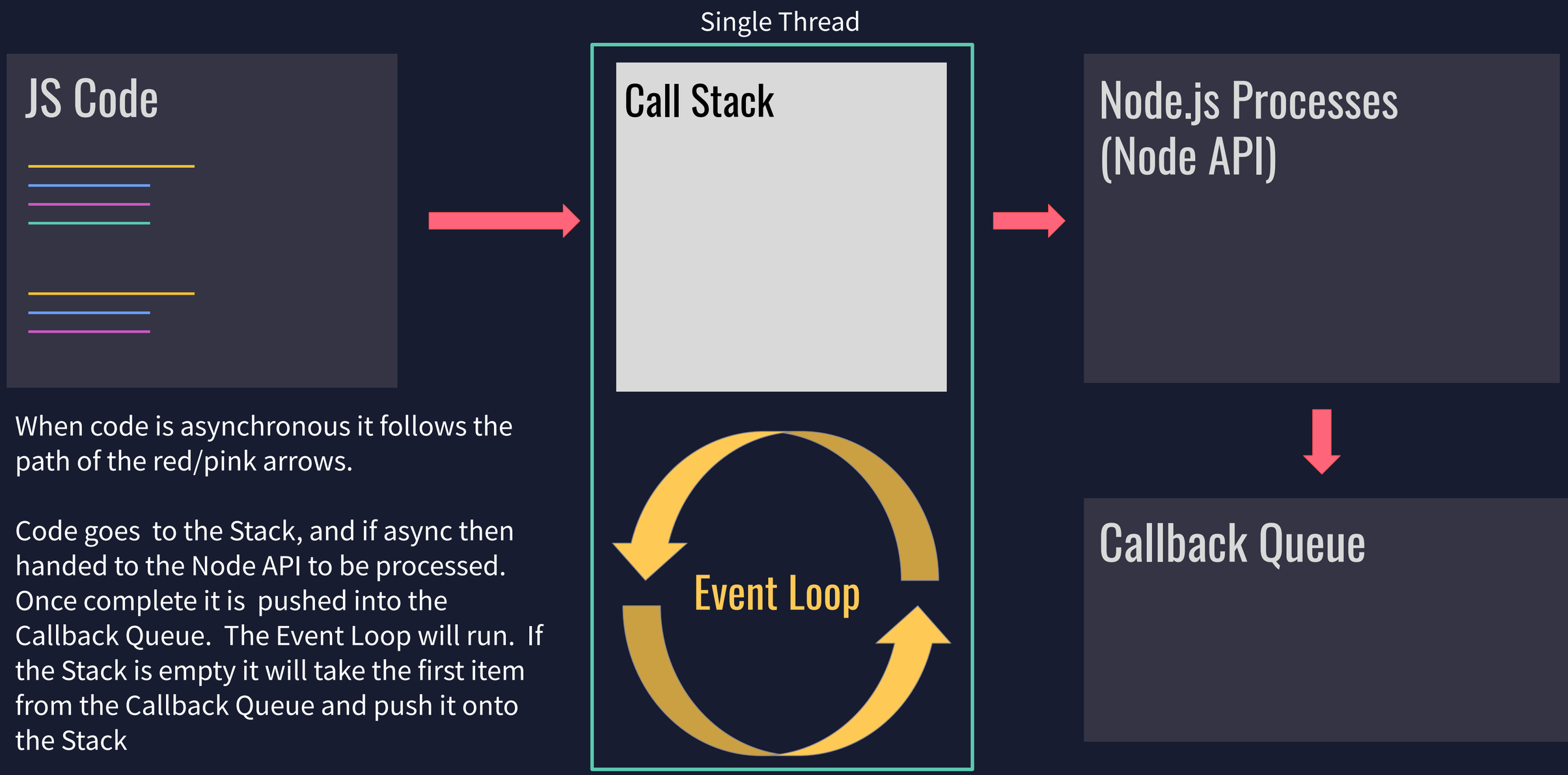
The **Event Loop** works with the Call Stack and the Callback Queue.

JavaScript has a single **Call Stack** in which it keeps track of what functions are currently executing and what function needs to be executed next.

When executing an asynchronous function with Node.js this function is added to the Call Stack then it gets moved to be handled by **Node.js background processes or thread pool**.

When a async process has finished, such as a network request or file read. It is then placed into the **Callback/Event Queue**. Only after the Call Stack is empty, then items in the Callback/Event Queue are processed by the Event Loop - it is popped from the Queue and pushed to Stack.

Visualizing the Event Loop



When code is asynchronous it follows the path of the red/pink arrows.

Code goes to the Stack, and if async then handed to the Node API to be processed. Once complete it is pushed into the Callback Queue. The Event Loop will run. If the Stack is empty it will take the first item from the Callback Queue and push it onto the Stack

Do NOT Block the Event Loop

As we learned the Event Loop is on the main thread. And asyc "expensive" tasks are off-loaded to a Threadpool. If the Event Loop is **taking a long time to execute a callback** then we refer to this as "blocked". When blocked it cannot handle requests from any other clients.

Blocking code is generally considered a long operation that runs more than a few seconds to complete. However, it is important to understand that a **long response time does not always mean you have a blocking code**. Response times can take on the order of seconds when code is related to database access, external API requests and etc.

Try to understand and differentiate long operations and operations that will slow down or block the Event Loop. Operations that can be handled asynchronously should be - meaning always try to use the asynchronous version of all Node.js methods.

What Node.js Achieves

Node.js is *not* the solution for every development situation. It is a platform that fills a particular need and understanding Node.js strengths is essential. You would not want to use Node.js for intensive operations or computations.

Node.js addresses one of the more difficult problems when writing applications that communicate over a network and this is managing input and output — meaning the reading and writing of data to and from the network, the hard drive, databases and etc. Plus, it has the added benefit is that it has become a way to unify the stack by having both JavaScript server side and client side.

Who actually uses Node.js?

Since it’s launch in 2009 Node.js has continued to grow with the modern development practices and thus rise in popularity.

- Google - Google Product Manager Interview on Node.js
- Netflix
- PayPal/Venmo
- NASA
- Twitter
- Uber
- Mozilla
- Disney/Hulu

Node.js and the Module System

Node.js attempts to remain as lightweight as possible. So, Node.js puts little functionality in the global scope.

If you want to access other built-in functionality, you have to use the **module system** for it. Node.js utilizes the CommonJS module system, based on the **require** function.

This `require()` system is built into Node.js and can be used to load in various items into our application.

`require(String)` accepts a string argument which can represent the name of a Node module or relative path to JSON and JavaScript files.

Node.js Core API

Information on the entire Node.js API and it's core module system can be found on the Node.js site.
Node.js API (**https://nodejs.org/dist/latest-v16.x/docs/api/**

Global Environment Keyword

- `__dirname` - provides the absolute path to the directory that holds the file

Path

- Utilities for working with file and directory paths.

```
const path = require('path')
```

File System

- Handles File I/O

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

Node.js Core API

Information on the entire Node.js API and it’s core module system can be found on the Node.js site.
Node.js API (**https://nodejs.org/dist/latest-v16.x/docs/api/**)

HTTP/HTTPS

- Interfaces designed to support features of the http or https protocol.

```
const http = require('http')
const https = require('https')
```

Node Package Manager (NPM)

NPM is two things:

(1) An online service registry where one can download/upload packages. These package can be optionally shared with the JavaScript community. (https://www.npmjs.com/)

(2) A CLI program (bundled with Node.js) that helps you install and manage the packages This CLI is how most developers interact with npm to install or remove packages.

A package is what **NPM** use to refer to re-usable code. Another name used to refer to these packages are called modules and sometimes libraries. These words are interchangeable.

NPM package.json

All npm packages and Node.js projects contain a file called **package.json.** This file holds various metadata relevant to the package and/or project.

The **package.json** is used to provide information such as the project’s name, authors, and most importantly the project's dependencies.

It can also contain **additional 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.

NPM Basic Commands

`npm init`
- assist in creating the package.json.

`npm install`
- installs all modules in the package.json - installed in the node_modules directory.

`npm install --save <module-name>`
- installs the module by name and auto-magically adds it to package.json.

`npm uninstall --save <module-name>`
- removes the module by name and auto-magically removes it to package.json.

NPM Node Modules

Node.js modules are code libraries that **utilize the built-in modules** in the Core API of Node.js to provide a more complete solution with enhanced functionality.

Not all Node.js modules are “good”. Evaluating a module before using it is important. One way to evaluate the module is by looking at the various statistics on NPM.

- Build and Test Flags
- Documentation and Github Repo
- Dependencies
- Last Updated & *Weekly Downloads**

Superagent

Progressive HTTP request library/module that works both with Node.js and inside the browser.
Superagent is essentially a user-friendly wrapper built on top of the Node.js Core API.

Superagent on NPM

- https://www.npmjs.com/package/superagent

Superagent on Github

- https://github.com/visionmedia/superagent (Source Code)
- https://visionmedia.github.io/superagent/ (Documentation)

Resources

Node.js Github

https://github.com/nodejs/node

Node.js Docs

https://nodejs.org/en/docs/meta/topics/dependencies/

https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

https://nodejs.org/en/docs/guides/dont-block-the-event-loop/

What The Heck Is the Event Loop? - JS EU Conference YouTube

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

Async Demo

Mozilla Developer Network

https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

Review and Prep

Review

- Review Slides
- Review and Run Code Examples
- Watch “JS EU Conference - What The Heck Is the Event Loop?”
    - Pairs with Slide 7
    - Slide 20 Link to Video

Homework 2

- Open on Canvas - Wednesday 03/01
- Due on Canvas - Wednesday 03/08 at 11:59pm PST
- Async + Node.js topics from 2/20 and 2/27
- 2x Lab Time to work on Homework 2

```
console.log('Week 06');
console.log('Code Examples');
```