Node js

Javascript for Web Development

Introduction

- Javascript is a cross-platform, object-oriented scripting language used to make webpages interactive (e.g., having complex animations, clickable buttons, popup menus, etc.). There are also more advanced server side versions of JavaScript such as Node.js, which allow you to add more functionality to a website.

Javascript for Web Development

History

- In September 1995, a Netscape programmer named Brandan Eich developed a new scripting language in just 10 days. It was originally named Mocha, but quickly became known as LiveScript and, later, JavaScript.

Javascript for Web Development

Applications:

- ***Client-side JavaScript*** extends the core language by supplying objects to control a browser and its *Document Object Model* (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.
- ***Server-side JavaScript*** extends the core language by supplying objects relevant to running Javascript on a server. For example, server-side extensions allow an application to communicate with a database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

Node js

Topics

- What is Node js?
- Node js Architecture
- NPM (Node Package Manager)
- Node js modules
- Basics
- File System
- Events
- Using Third Party Apis
- HTTP Module

Node js

Introduction

- Node.js is an open source, cross-platform JavaScript runtime environment for server-side and networking applications. Node.js is built on top of the Google V8 JavaScript engine, which means Node.js applications are written in JavaScript and use a similar syntax as front-end JavaScript applications, including objects, functions, and methods. Node.js was developed by Ryan Dahl in 2009.

Definition from Official Site:

- Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.
- Latest stable version available as on 10th July 2021: v14.17.3

Node js

Why Node js?

- Node.js is a JavaScript runtime built on Chrome’s V8 JavaScript engine (Very fast)
- Node.js uses an event-driven, asynchronous non-blocking I/O model
  (Asynchronous)
- Node.js operates on a single thread event loop (Highly Scalable)
- Huge ecosystem of open source packages (npm)

Node js

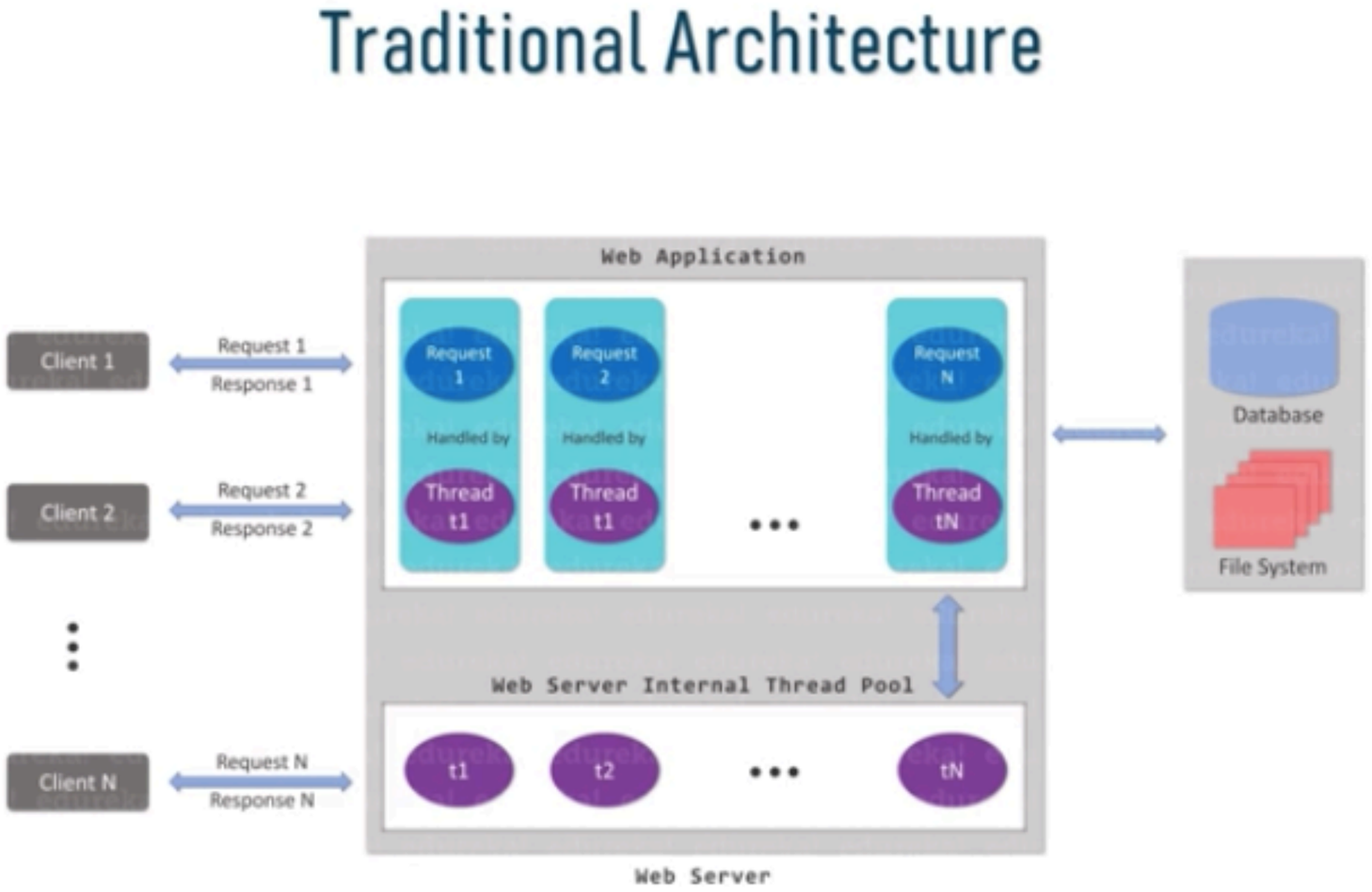
Features

- Open Source
- Simple & Fast
- Asynchronous
- Highly Scalable - facilitates micro service architecture
- Single Threaded —> Not resource intensive, yet it allows things to be done in parallel.
- No Buffering / Waiting —> JS Event Loop
- Cross Platform

Node js

Traditional Architecture - JAVA

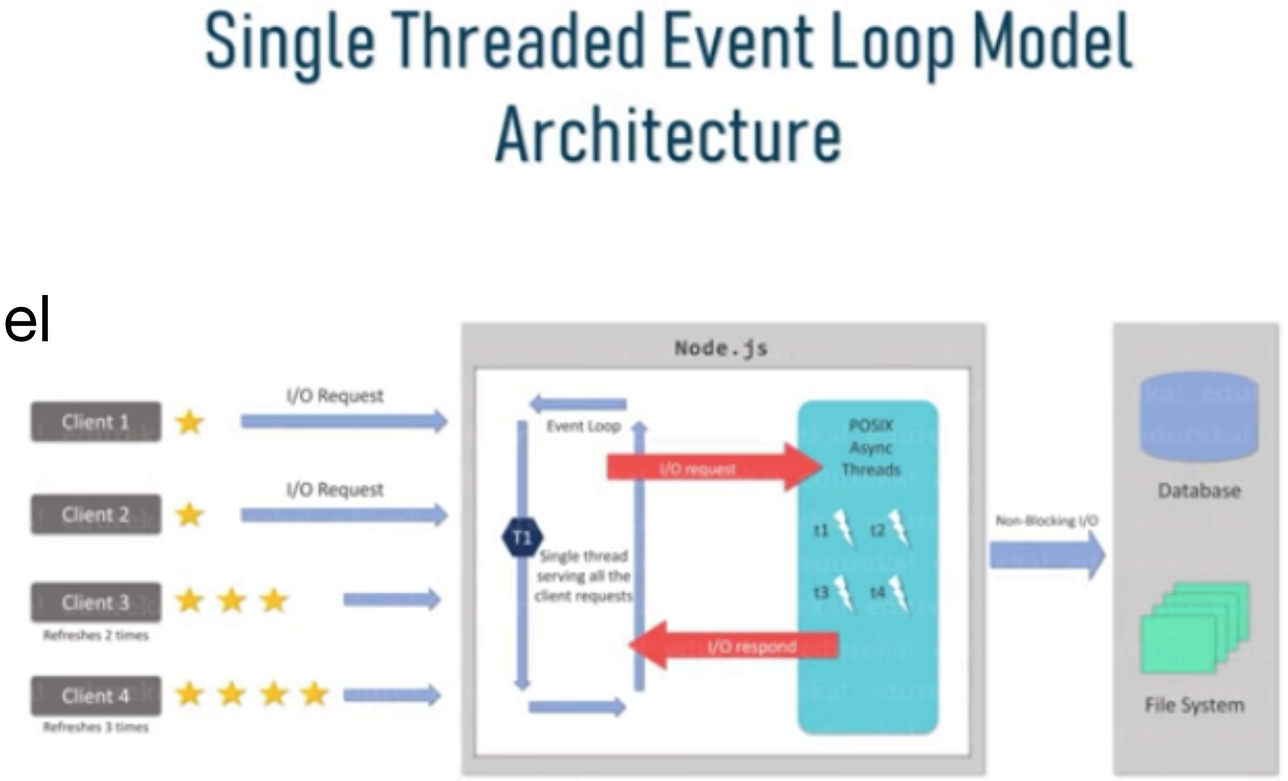
- Multi-threaded
- Resource Intensive



Node js

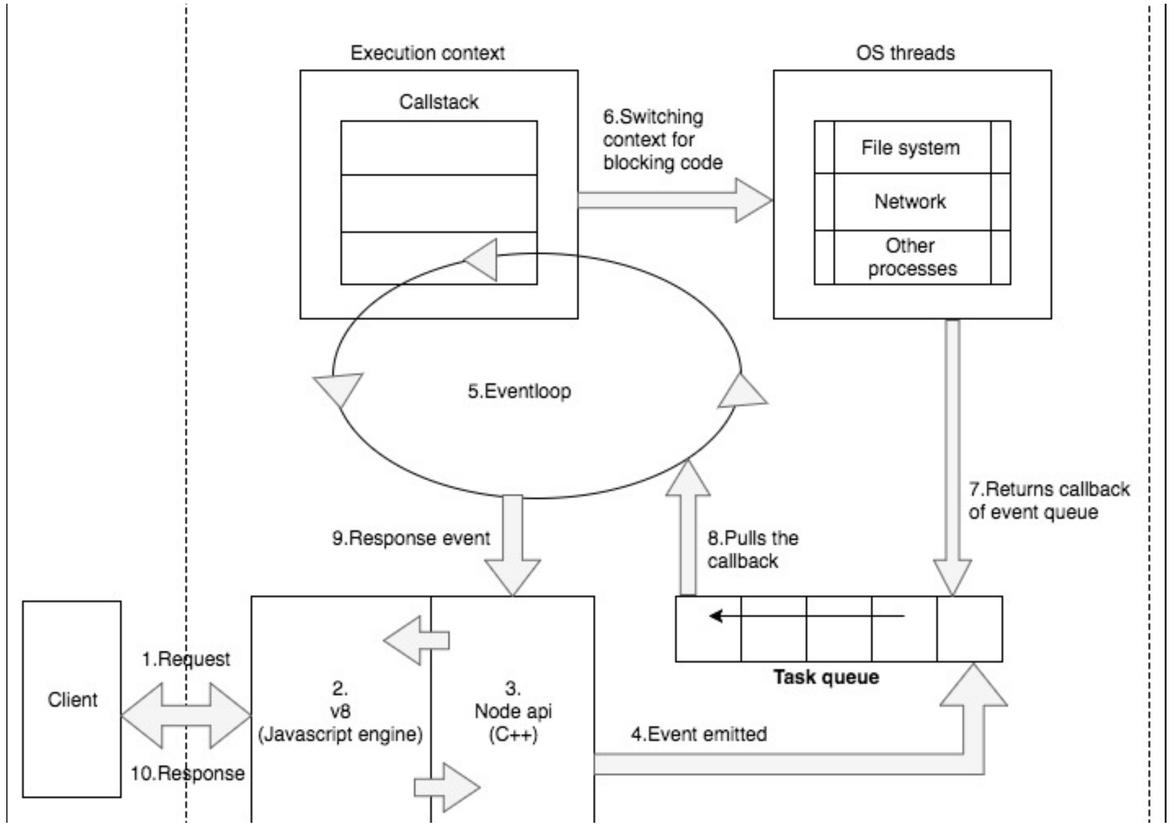
Node js Architecture

- Single threaded
- It goes through event loop
- Appears to be executed in parallel



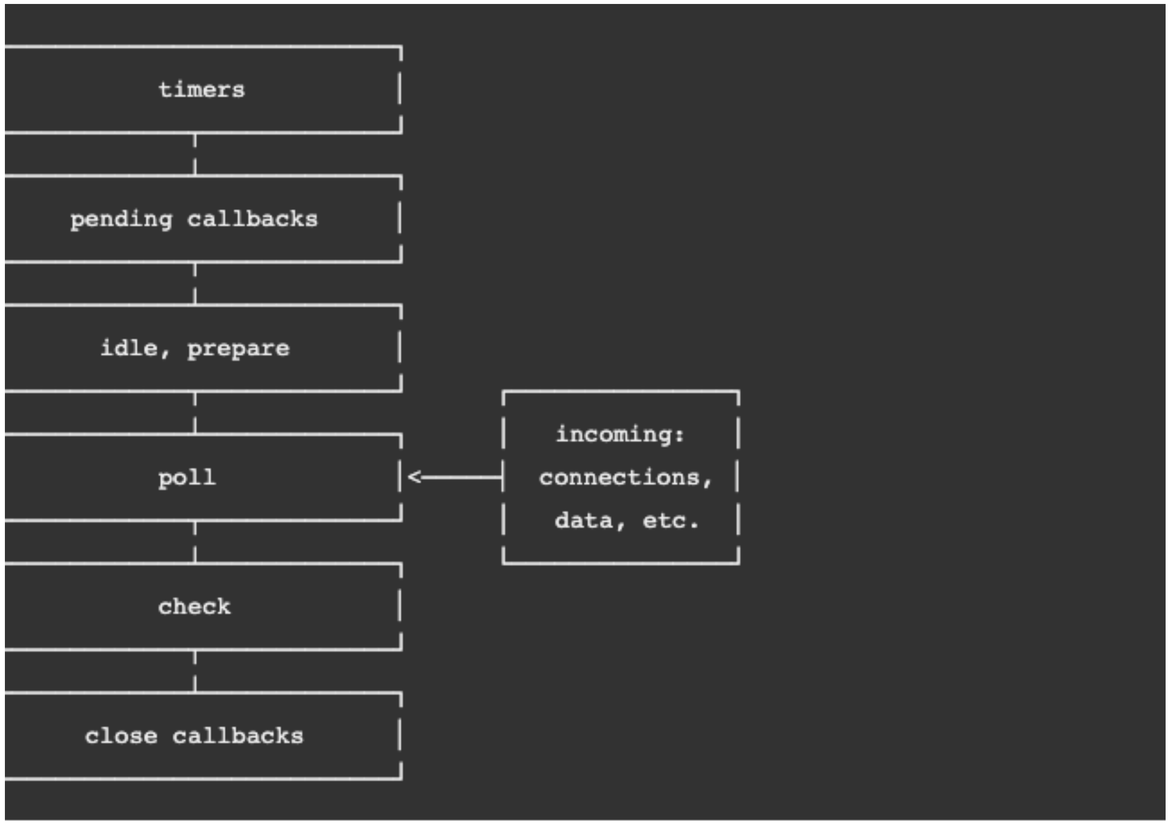
Node js

Node js Event Loop



Node js

Node js Event Loop



Node js

Installing Node js

- Node js org
- NVM - Node Version Manager

Node js

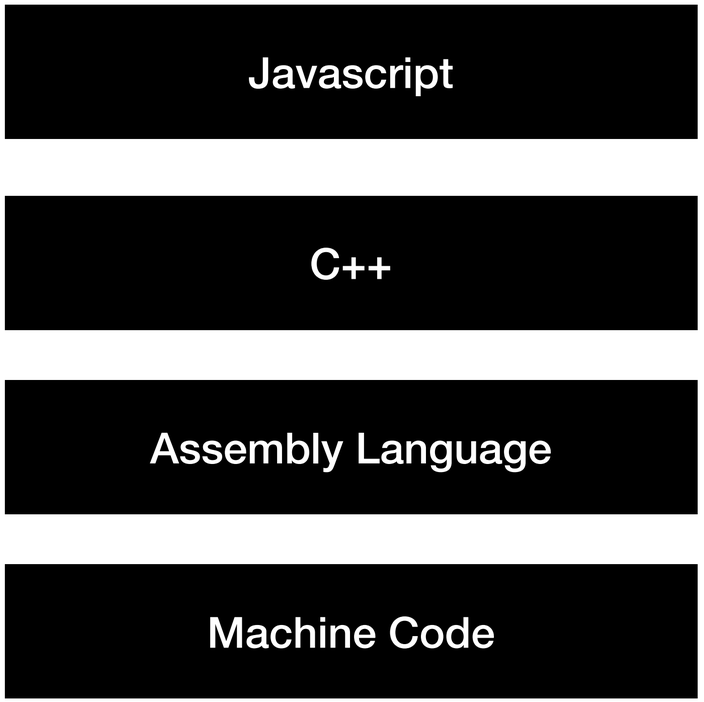
V8 Engine

V8 is the name of the JavaScript engine that powers Google Chrome. It's the thing that takes our JavaScript and executes it while browsing with Chrome. V8 provides the runtime environment in which JavaScript executes. The DOM, and the other Web Platform APIs are provided by the browser.

- Javascript is not understood by computers
- It needs to be converted to machine code

Node js

V8 Engine



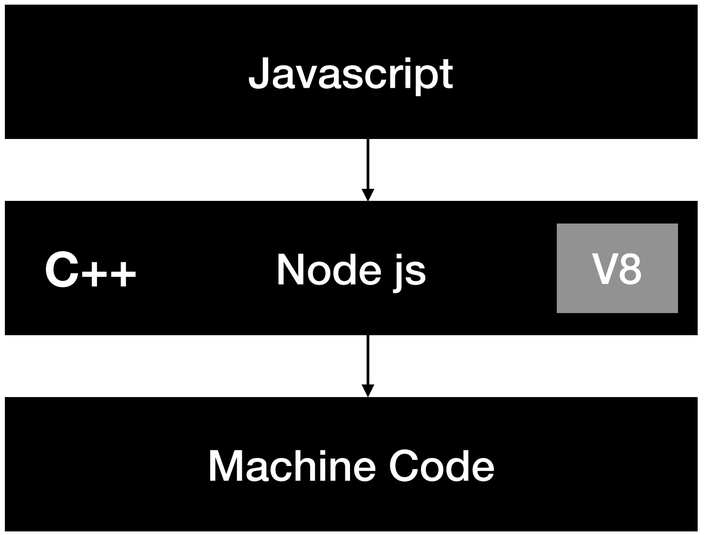
Node js

V8 Engine

- Node js is written in C++
- At the core, it is V8 engine
- V8 engine converts javascript to machine code
- https://developers.google.com/v8

Node js

V8 Engine



Node js

Initialising a node project

- npm init

Node js

NPM - Node Package Manager

npm is the standard package manager for Node.js.

In January 2017 over 350000 packages were reported being listed in the npm registry, making it the biggest single language code repository on Earth, and you can be sure there is a package for (almost!) everything.

It started as a way to download and manage dependencies of Node.js packages, but it has since become a tool used also in frontend JavaScript.
There are many things that npm does

Node js

NPM - Node Package Manager

- Core Modules - Comes with node js - fs, path, os, http etc…
- Local modules - which are written by a developer - functions written and exported by a developer
- Third Party Modules - NPM repository

Node js

Install a npm package

- Globally —> npm install --global <package> / npm install -g <package>
- Locally —> npm install --save <package> / npm install -s <package>
- Dev dependency —> npm install --save-dev <package>
- Uninstall —> npm uninstall <package>
    - Use -S to remove it from dependencies,
    - Use -D to remove from dev dependencies
    - Use -g to uninstall package globally.

Other commands:

- npm install
- npm update

Node js

package.json - JSON File

- heart of the node js application
- Manifest file that contains the metadata of the project

Node js

package-lock.json - JSON File

- In version 5, npm introduced the package-lock.json file.

Node js

Semantic Versioning using npm

The Semantic Versioning concept is simple: all versions have 3 digits: x.y.z.

- the first digit is the major version
- the second digit is the minor version
- the third digit is the patch version

When you make a new release, you don't just up a number as you please, but you have rules:

- you up the major version when you make incompatible API changes
- you up the minor version when you add functionality in a backward-compatible manner
- you up the patch version when you make backward-compatible bug fixes

Node js

Semantic Versioning and npm update

- `^` : It will only do updates that do not change the leftmost non-zero number. If you write `^0.13.0`, when running `npm update`, it can update to `0.13.1`, `0.13.2`, and so on, but not to `0.14.0` or above. If you write `^1.13.0`, when running `npm update`, it can update to `1.13.1`, `1.14.0` and so on, but will not update to `2.0.0` or above.
- `~` : if you write `~0.13.0`, when running `npm update` it can update to patch releases: `0.13.1` is ok, but `0.14.0` is not.
- `>` : you accept any version higher than the one you specify
- `>=` : you accept any version equal to or higher than the one you specify
- `<=` : you accept any version equal or lower to the one you specify
- `<` : you accept any version lower to the one you specify
- `=` : you accept that exact version
- `-` : you accept a range of versions. Example: `2.1.0 - 2.6.2`
- `||` : you combine sets. Example: `< 2.1 || > 2.6`

Node js

Command line

- REPL (Read Eval Print Loop)
- Running a script in node js
- process (core module)
- Access environment variables
- Command line arguments
- npx command

Node js

Global Object

- Its the window object in case of a browser - provides different methods like console, alert, setTimeout, setInterval, clearInterval
- Node js also provides global objects —> https://nodejs.org/docs/latest-v13.x/api/globals.html

Node js

Modules and require

- A module is any js file
- module.exports —> require
- export —> import (ESM)

Node js

File System

- var fs = require(‘fs’);
- Synchronous and Asynchronous methods
- Create / Delete files, dirs etc..

Node js

Events

- Emit and handle events
- util module —> inherits

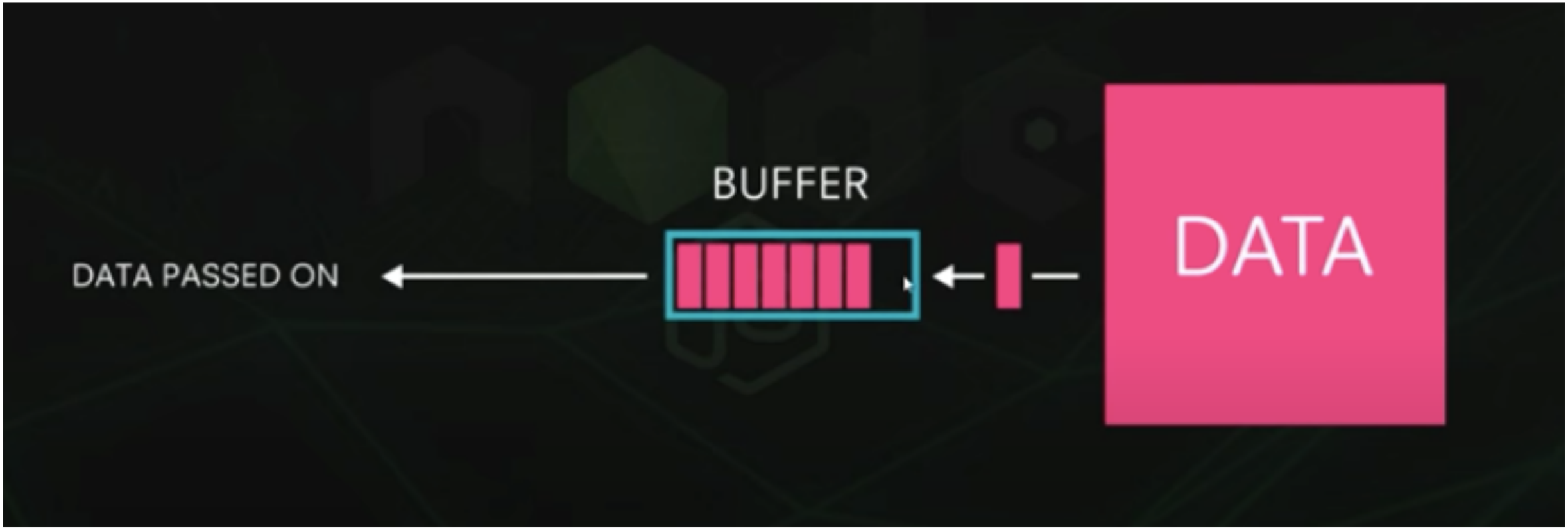
Node js

Streams

- Buffers
  - Temporary storage spot for a chunk of data that needs to be transferred from one place to other
  - Transfer small chunks of data at a time

Node js

Streams



Node js

Streams

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

Node js

Streams

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.