# Node 101

Backend Foundations

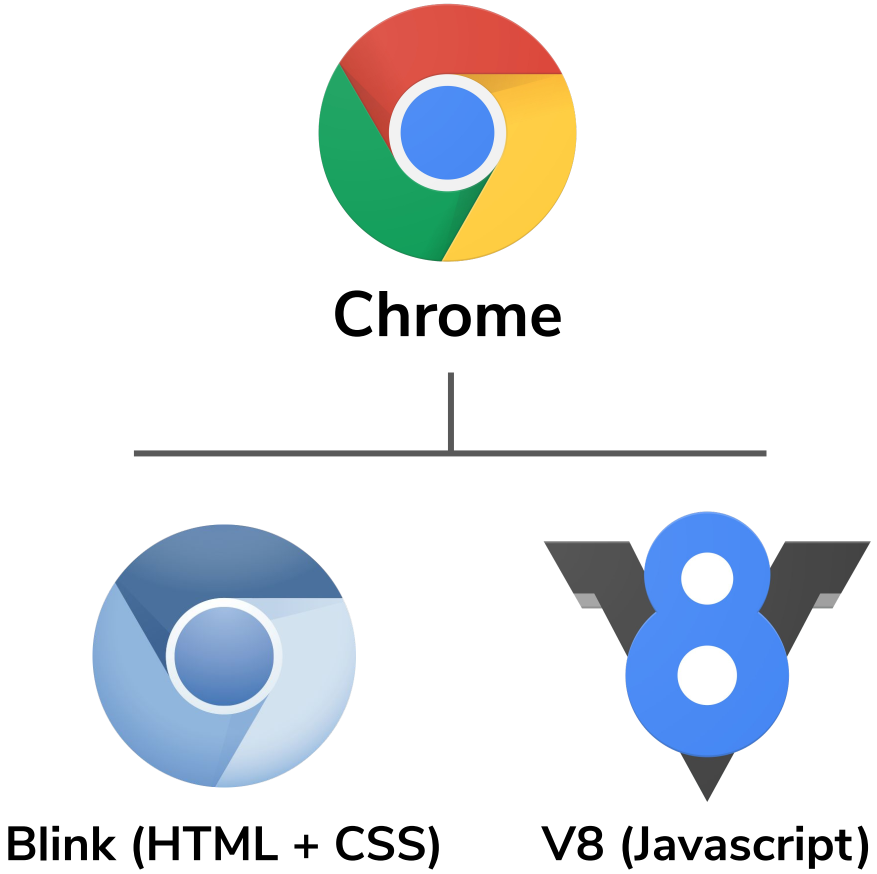
## Browser Engines

Chrome has two engines. Blink and V8. Blink handles the HTML and CSS, while V8 handles the Javascript.

Both of these are open source and you can see the source code and do what you like with it (so long as you follow the licence).

Blink is used in a number of other browsers including Opera, Vivaldi and now Edge.

The V8 engine is also used in a number of other database and server applications.



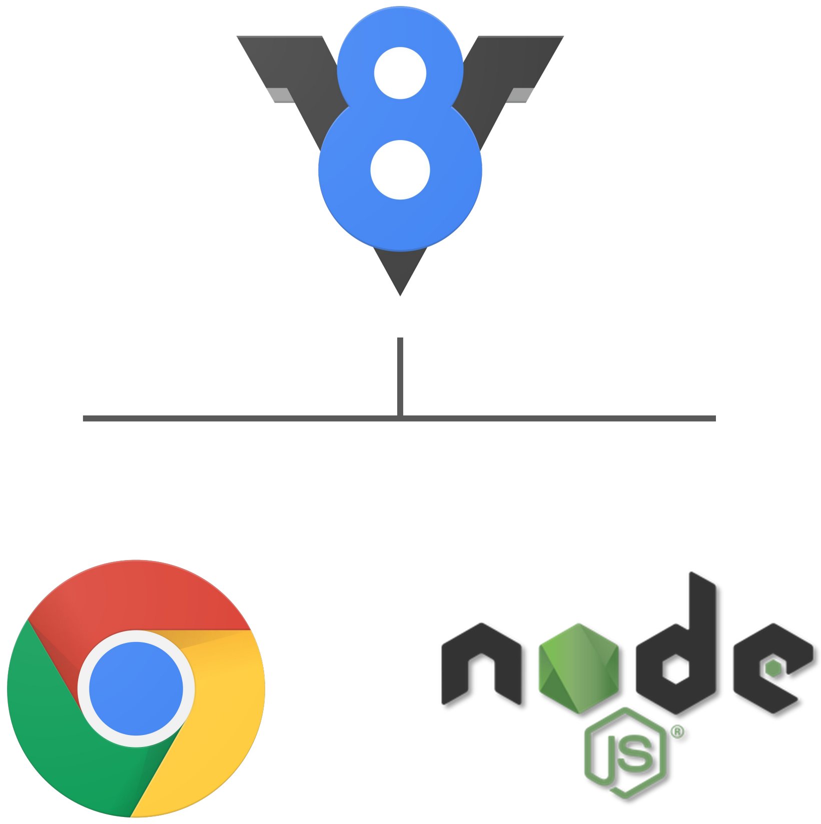
# Node.js

Most notably, V8 is also used in Node.js.

Node.js is an open-source, cross-platform runtime environment used for development of server-side web applications.

It uses:

- Event-driven architecture
- Non-blocking Input/Output API



## Why use Node?

- Asynchronous event driven IO helps concurrent request handling
  - One of the biggest selling points
    - This means that if a request is received by Node for some Input/Output operation, it will execute the operation in the background and continue with processing other requests.
  - Other languages, Input/Output blocks. When files are read in, the entire program stops.

## Why use Node?

- Handling of concurrent requests
  - Because Node is event based, concurrent requests don't add overhead, they just go onto the call stack until they can be processed
- It's JavaScript
  - You already know JavaScript. That's a big plus.
- Lots of developers know JavaScript
  - There is an active and vibrant community for the Node.js framework

## Why use Node?

- Handling of concurrent requests
  - Because Node is event based, concurrent requests don’t add overhead, they just go onto the call stack until they can be processed
- It’s JavaScript
  - You already know JavaScript. That’s a big plus.
- Lots of developers know JavaScript
  - There is an active and vibrant community for the Node.js framework

# When should I use Node?

Node.js is best for usage in streaming or event-based real-time applications.

**High Concurrency**

- Chat applications
- Game Servers
- Streaming Servers
- Collaboration Servers

**Low CPU time**

- Static Sites
- Advertisement servers
- Brochure-ware

## When should I ***not*** use Node?

Node is structured to be single threaded. Heavy, CPU-intensive calculations will prevent Node from processing other requests.

## What is different about Node vs Browser JS?

- You can access the filesystem and other system level libraries.
  - Modify files, access databases, include low-level libraries.
- No Page
  - There is no “page” to display. This means there is no `document` or `window` available in the global scope.
- Core Modules
  - Node comes with modules that add extra functionality

## Modules

## What is a Module?

- A module in Node.js is a reusable piece of functionality organized in separate JavaScript file/s that can be reused throughout the Node.js application.

- Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope.

- Modules can be placed in separate .js file/s under a separate folder.

# The Node Module Pattern

`require()`

and

`module.exports =`

## require()

First you need to require the module

In this example, the ‘http’ module exports an object with all of its methods included so we are assigning this exported object to the constant variable ‘http’.

Most modules typically export an object, but they can export anything you can assign to a variable; strings, functions, arrays, numbers, booleans, etc.

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

# module.exports

When creating our own module, we need to explicitly tell Node what we want to become available through exports.

On the right, this example is exporting two functions, the first is `multiplyNumbers` and it is being exported on the object property `multiply`. The second is `addNumbers` exported as `add`.

```
module.exports = {
  multiply: multiplyNumbers,
  add: addNumbers,
};
```

# Using it!

If the file in the last example was called “math.js” and it was in the same folder as out main application, we would import it like this.

```
const math = require('./math');
```

Once it is required, we can use it in our file like this example shows.

```
const newNumber = math.add(4, 4);
```

## Types of Modules

- Core Modules
    - This are the modules that come with Node
- Local Modules
    - These are the modules we write ourselves
- Third Party Modules
    - These are the modules that others have written and we have installed

## Writing a simple module

This example creates an object with three functions as properties; info, warning, and error.

It then exports the log object for use.

```
const log = {
 info: function (info) {
   console.info('Info: ' + info);
 },
 warning:function (warning) {
   console.warning('Warning: ' + warning);
 },
 error:function (error) {
   console.error('Error: ' + error);
 }
};

module.exports = log;
```

## Using a local module

Referring to the previous slide, this example imports the module from the `log.js` file as `myLogModule` and then uses the `info` method to send a message.

To use local modules in your application, you need to specify the path of JavaScript file of the module.

```
const log = require('./log.js');

log.info('Node.js started');

// Info: Node.js started
```

## Core Modules

Node.js is a lightweight framework. The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary distribution and load automatically when Node.js process starts.

To use this functionality, you need to import the core module into your application.

The Node Documentation has the best detail on all of these core modules: https://nodejs.org/docs/latest/api/index.html

## Core Modules

- DNS
- FileSystem (fs)
- Path
- Readline

# Third Party Modules

The JavaScript Community is massive. They have solved a lot of problems.

You can use them.

They are installed from a Package Repository like:

- NPM
- Open-Registry.dev
- GitHub Packages

```
// from terminal, inside project folder
npm install cowsay

// inside app.js
const cowsay = require('cowsay');
```

- Set of command line tools for managing packages (AKA modules)
- Keep it DRY
- Publish to npmjs.org

## Use it!

When you create a new node project, you should set it up with the init command!

This will ask you a bunch of questions to set up the project and create a `package.json` file

You can skip all these questions and use the default answers by running `npm init -y`

```
npm init
```

```
npm init -y
```

## Use it!

You can install new modules by running the command `npm install` followed by the module name

```
npm install module-name
```

You can install packages globally with the `-g` option. You only really need to do this with command line tools.

```
npm install -g module-name
```