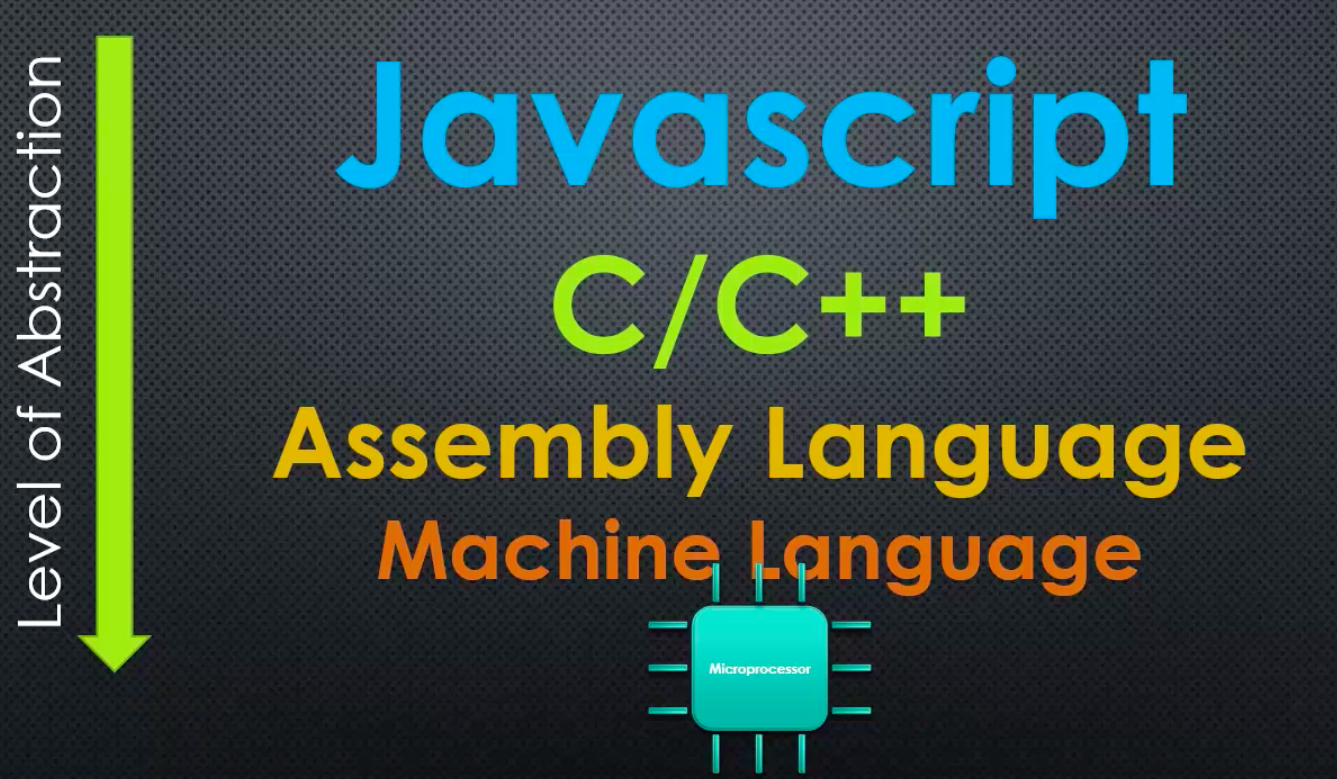
Node lets us write server code in javascript

**SECTION 2: V8 THE JAVASCRIPT ENGINE**

V8 JS engine (the heart of node.js)

You have a processor which is given instructions in machine code (programming language spoken by a computer processor)



everything written is JS is compiled through an engine that takes care of memory access and stuff, so the processor understands it

Node is written in C++!

* Because V8 (the JS engine that converts JS into machine code) is written in C++
* Node JS is built on top of V8

JAVASCRIPT ENGINES AND ECMASCRIPT SPECIFICATION

ECMASCRIPT

* The standard javascript is based on (many browsers appeared with their own engines, so a standard for JS was needed) (tells how the language should work)
* V8 makes sure the machine code that JS is compiled to works as the specification in ECMASCRIPT says

JS engine

* A program that converts JS code into machine code; in the case of V8, it should follow the ECMAscript standard

V8 is a JS engine, and it sits at the core of Node JS (node is built on top of it)

ADDING FEATURES TO JAVASCRIPT

You can embed the V8 engine into your c++ programs (since V8 is written in c++)

* In this way, you can make V8 do more things than V8 can actually do

I can write things in c++, that JS does not have, and make everything that I can do with c++ available to JS code

* JS was built for the browser, c++ was built for more lower level

You can add features to JS using c++ and extending the functionality of V8

* You send JS code to your c++ program, which then sends it to V8, then V8 calls the c++ code appropiatly for the JS code
* What you do is you tell V8 that when it encounters a certain word in JS code, V8 needs to call a specified function in c++ (which JS does not support for alone)
* This adds functionality to JS code

So V8 is built to be embedded to c++ programs so that c++ functionality is available to JS code (c++ functionality gets called through JS)

Google chrome is built in c++, it embeds V8. So when you make any kind of DOM manipulation using JS in your websites; this feature is actually not specified in the ECMAscript, it has been added by the google chrome browser. So it is adding functionality to JS.

Node JS is a c++ program, with V8 embedded that has added a bunch of c++ features in order to be a server technology

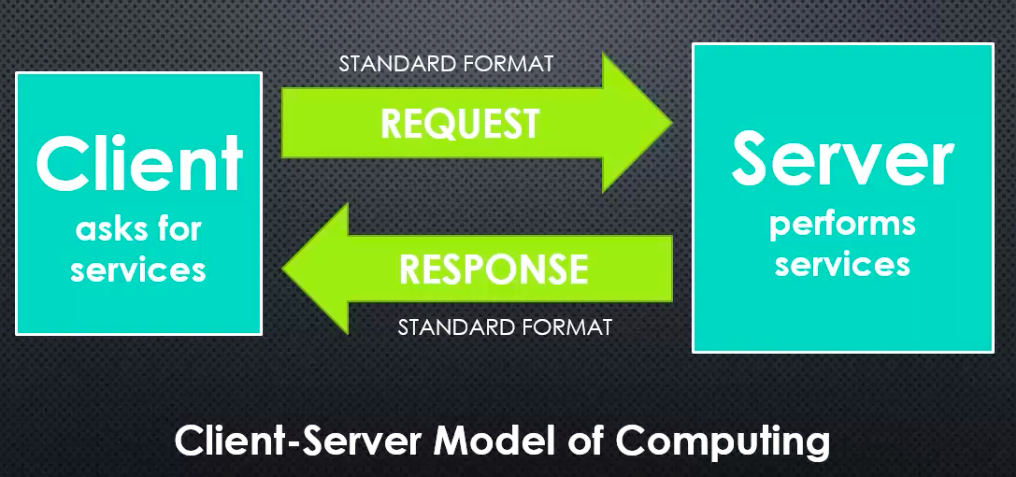
**SECTION 3: THE NODE CORE**

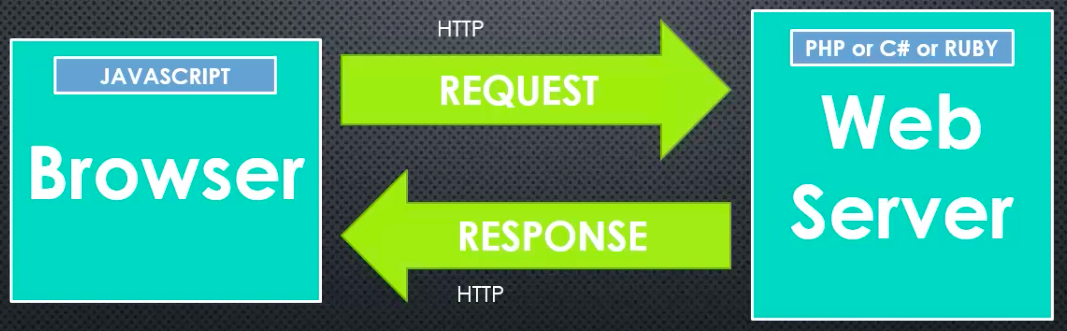
SERVERS AND CLIENTS

Node JS was designed to use JS to write server code

Front-end web development

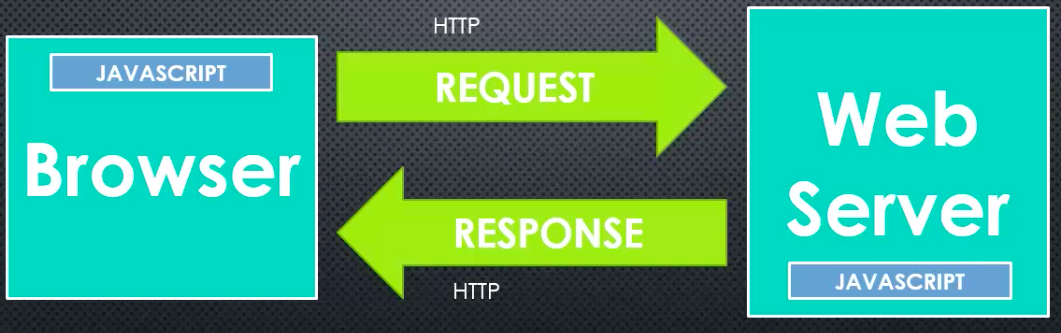
* Coding in the browser





when doing some coding in the client side, you would use JS, but when doing some coding in the server side, you had to switch to another programming language such as PHP, etc.

Therefore, NodeJS goal is to write JS on the webserver, so you don’t need to switch between programming languages; you just need to use JS.



WHAT DOES JS NEED TO MANAGE A SERVER

1. Better ways to organize our code into reusable pieces
2. Ways to deal with files
3. Ways to deal with databases
4. The ability to communicate over the internet
5. The ability to accept requests and send responses (in the standard format)
6. A way to deal with work that takes a long time

NodeJS takes care of all of these requirements

THE C++ CORE

deps folder

* Things that were built outside of NodeJS, but that are part of NodeJS

src folder

* The c++ source code

NodeJS is not javascript; NodeJS accepts Javascript; but at the same time and allows us to have more features when using JS

THE JAVASCRIPT CORE

lib folder

* The JS source code
* The most part are wrappers, they are JS libraries that wrap c++ features

**SECTION 4: MODULES, EXPORTS, AND REQUIRES**

Module

* A reusable block of code whose existence does not accidentally impact other code
* It is independent from the other code, but works with it
* You can re-use it
* A way to structure your code; to organize it as it grows

Common JS modules

* An agreed upon standard for how code modules should be structured

First class functions

* Everything you can do with other types, you can do with functions
* You can use functions like strings, numbers, pass them around, et variables equal to them, put them in arrays, etc.
* Allow us to write function expressions on the fly

LET’S BUILD A MODULE

Modules are independent, they do not affect code in other files, so when you specify a “require” in a file, and specify the path for another file; this will not give you access to the variables and functions of the file you are requiring, hence it does not intervene with your code. What this will do is that any code that is supposed to run in the file that is being required, will run.

* So, you can have variables and functions with the same name in each file, this does not matter, since modules do not affect the code in other files

To make the variables and functions of a JS file available to other files, then we need to specify a “module.exports = …”, and selecting the things that we want to make publicly available.

* This will make vars and funcs available to use, outside of the module

The “require” and “module.exports” functions are inside the JS core of the nodeJS code

OBJECTS AND OBJECT LITERALS

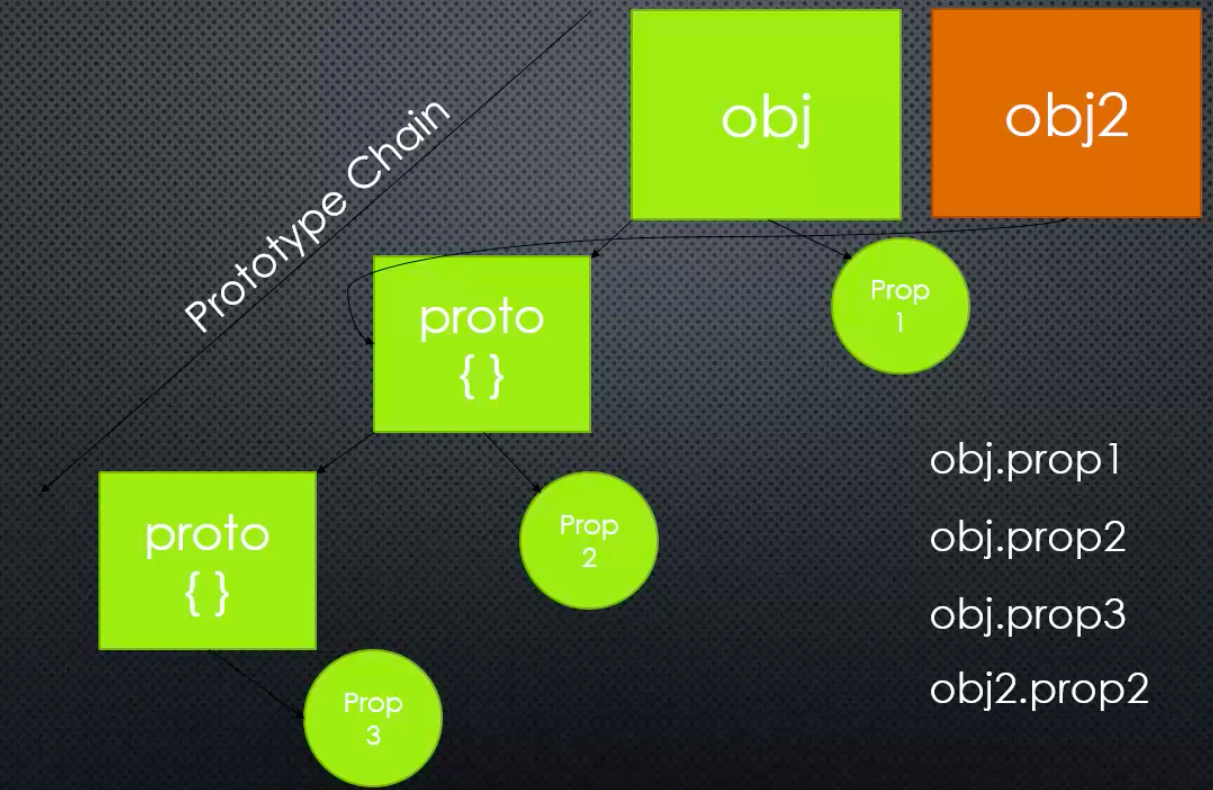
Object in JS

* A collection of name/value pairs

Object literal

* Name/value pairs separated by commas and surrounded by curly braces
* Is a quick way of creating JS objects

PROTOTYPAL INHERITANCE AND FUNCTION CONSTRUCTORS



Function constructor

* A normal function that is used to construct objects
* The “this” variable points to a new empty object, which is returned from the function

.property function

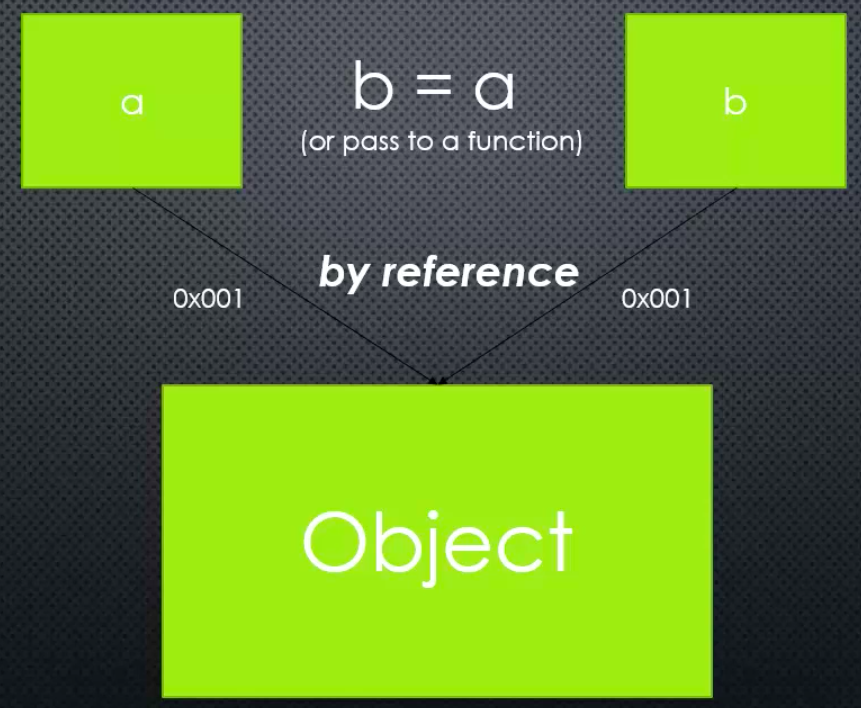
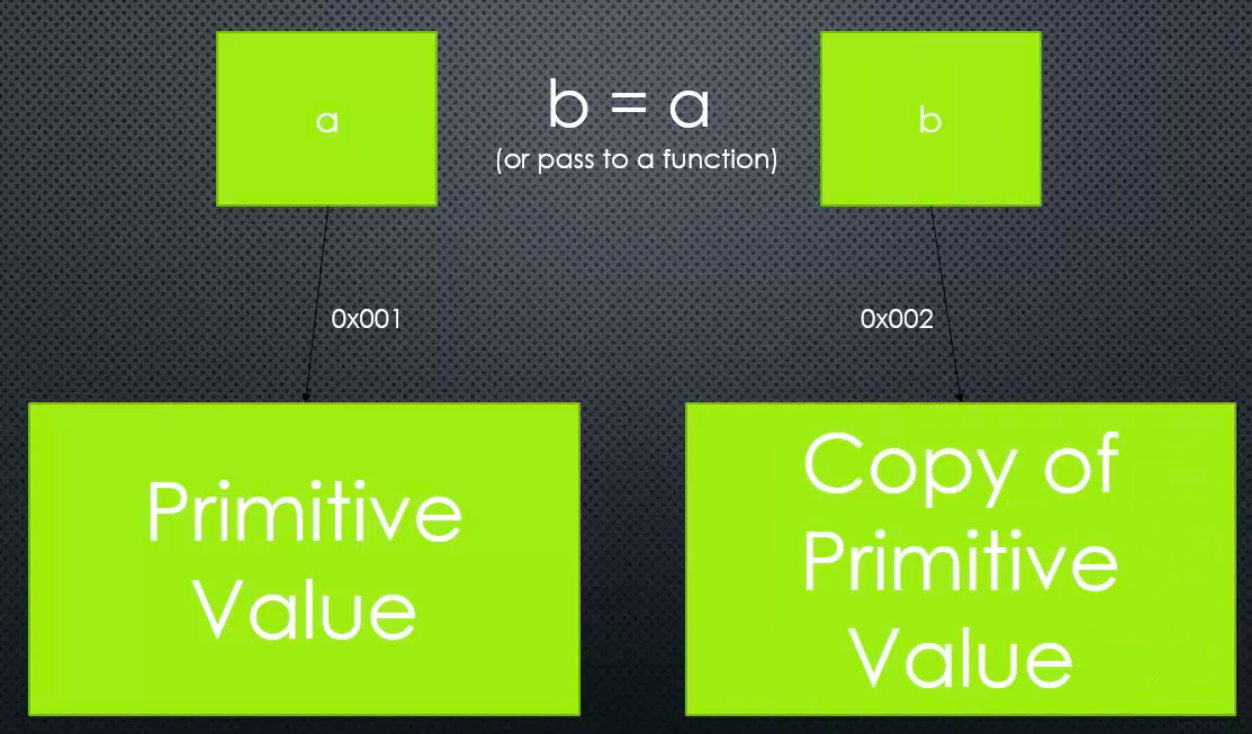
* We are adding some methods/properties to be available to all objects created from a function constructor

BY REFERENCE AND BY VALUE

Primitive

* A type of data that represents a single value (not an object)

Objects are always passed by reference



IMMEDIATELY INVOKED FUNCTIONS

The scope of the variables sits within the IIFE, you can have two variables with the same name in a file as long as one of them sits inside an IIFE

* IIFE’s therefore behave somewhat similar as modules, their code does not affect other code; IIFE’s wrap code inside

HOW DO NODE MODULES REALLY WORK? MODULE.EXPORTS AND REQUIRE

Your JS code get passed to node, which in turn wraps it inside a function expression, then your code runs, and module.exports is returned

The keyword require returns an object, which references the value of module.exports for a given file

JSONS

Standard for structuring data that is inspired by JS object literals

* JS engines are built to understand it; they can convert JS objects into JSON and JSON into JS objects
* In a .JSON file, you don’t need to specify a module.exports, when the require function is made, the engine will convert the JSON text into JS code

MODULE PATTERNS

“require” caches the results of the require function for any particular filename

* If you simply return an object from “require”, that will be cached, and any other call to that same file via the require function will get you back the same object everywhere

the revealing module pattern

* Exposing only the properties and methods you want via a returned object

Un callback es una función que se llama de manera asynchrona y se utiliza como parámetro en otra función

Un event handler es un callback asociado a un evento, tiene un event handler (que es el callback)

When you execute something synchronously, you wait for it to finish before moving on to another task. When you execute something asynchronously, you can move on to another task before it finishes.