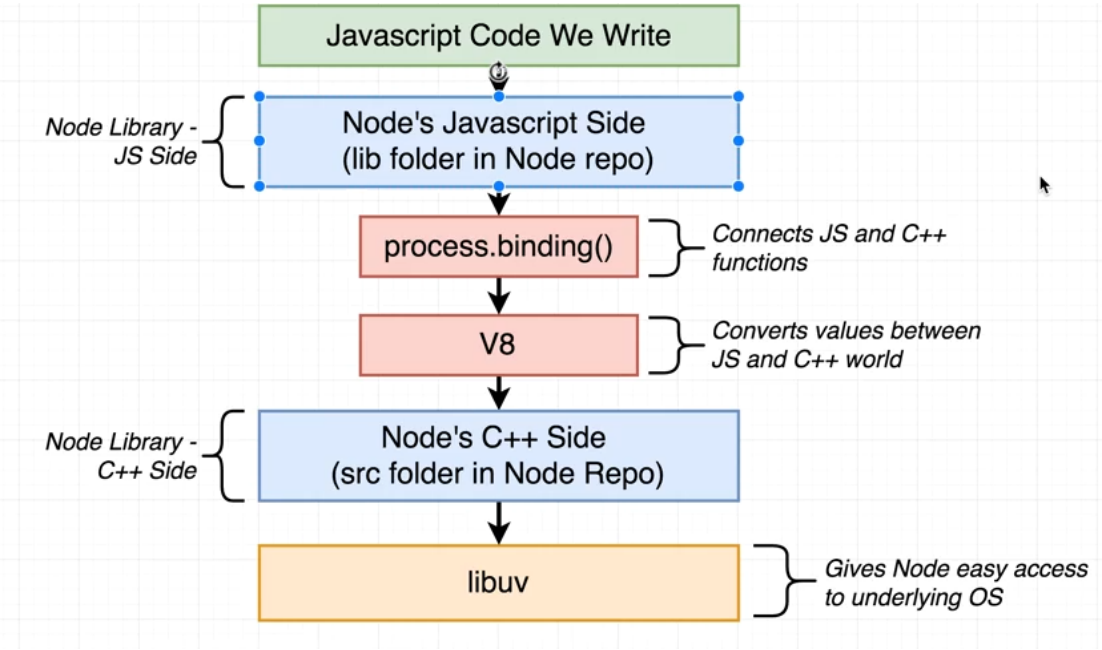
NODE.JS

Node.JS architecture

- V8
  - V8 is an open-source JavaScript engine developed by Google. It is primarily **written 100% in C++** and is used to execute JavaScript code in web browsers and other environments. V8 is designed for high-performance execution of JavaScript and is known for its speed and efficiency.
  - It effectively translates the high-level JavaScript syntax into lower-level C++ code that can be executed efficiently by the underlying hardware. This process allows web browsers to run JavaScript code and render web pages with interactive features.
  - **process.binding( )**: It is like a bridge that allows JavaScript code to interact with lower-level functionality implemented in C++.

  - **<u>JS code way to execution:</u>**
    - **Parsing**: The JavaScript source code is parsed by V8's parser, creating an abstract syntax tree (AST) that represents the code's structure.
    - **Compilation to C++**: V8's compiler translates the AST into optimized C++ code, tailored for the specific JavaScript code.
    - **Optimizations**: The generated C++ code undergoes various optimizations to improve performance, such as function inlining and loop optimizations.
    - **Just-In-Time (JIT) Compilation**: At runtime, V8's JIT compiler converts the optimized C++ code into machine code, specialized for the current execution context.
    - **Execution**: The machine code is executed by the computer's processor, producing the intended behavior of the original JavaScript code.
- LibuV

- LibuV is a critical component of the Node.js runtime **written in C++ and JS** . It provides the underlying infrastructure that enables Node.js to be asynchronous and event-driven.
- LibuV is also responsible for Event Loop and Thread Pool.



- The node.js program start with a single thread includes the event loop, then the event loop start to read the node.js file and check all functions start with pending time functions (setTime, setInterval) then pending OS tasks (server listening to port) and finally the pending operations (fs module)
- some pending OS tasks done out the event loop and OS take care of it and pending operations sends to the Thread pool
- by default the Thread pool contain 4 threads available and they can scale up and down using the following code

```
process.env.UV_THREADPOOL_SIZE = 5;
```

- increasing thread pool size not always benefits it can lead to:
  - diminishing returns:
    - introduce more overhead due to thread management and context switching.
  - Performance Degradation:

        - When you increase the thread pool size too much or without proper consideration, it can lead to performance degradation. This means that the overall performance of your application could actually become worse as a result of the increased thread pool size.
    - when a thread take a task it may complete it to finish or let it for some reason and back to it later or other thread back to it
    - like starting FS reading file, first the thread goes to hard drive to get some information about the file and while the hard drive get these information the thread goes to do another thing and back later to take the information that hard came with.

- Improve node.js performance
    - Node Clustering
        - *SECTION 2 IN "NODE.JS ADVANCED TOPICS" COURSE.*
    - Worker Threads
        - *SECTION 2 IN NODE.JS ADVANCED TOPICS COURSE.*
- *In case of uploading image*
    - *the traditional way is the image is going to the server then server sends it back to the database for example S3, and this is not the ideal way to deal with uploading process.*
      because of the uploading process is a heavy process and take much CPU and RAM resources.
      and you don't need the image to cross over the server first because you will not use it anyway.
      so the best way to upload an image is to send it directly from the client to the database and only
      thing you need is a metadata about the image from the client. so how to do that ??
      the image below discuss that..

- we can override the mechanism of converting object to json by adding a func called toJSON to the object
    - ex: person = {name:"Ahmed", toJson(){return 1}}
      anytime you called JSON.stringfy(person) ==> it will return 1

- if you want to check if session is defined in header and if JWT is defined in header in session field
    - you can do that in this statement if(!req.session || !req.session.jwt)
      but you can do it like this if(!req.session?.jwt)