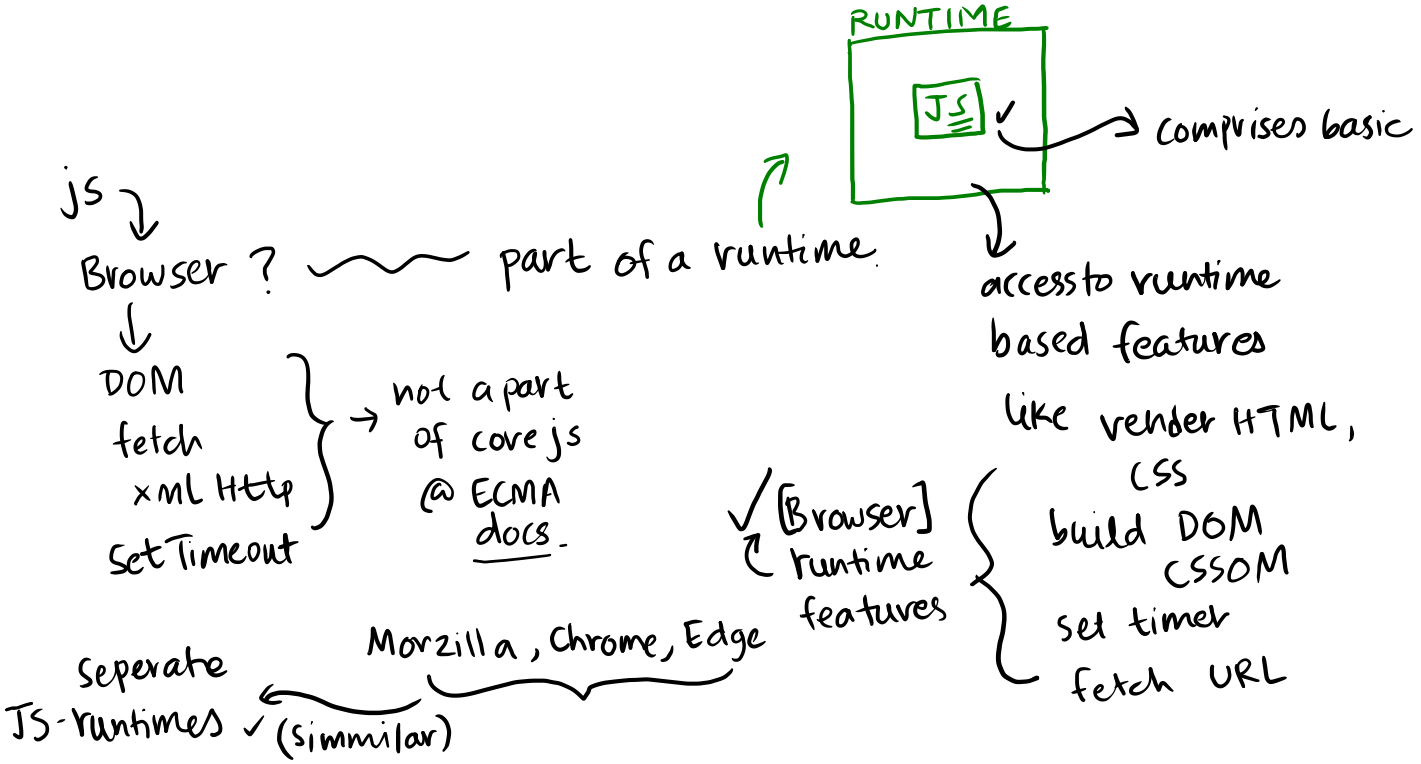
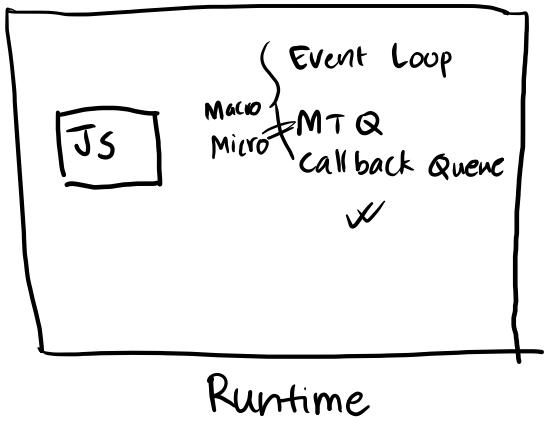
Introduction to Node.js - Lecture Notes by Milind Mishra





* for a very longtime only runtime that existed was "Browser."

Until 2009, Ryan Dhal (creator of Node.js) made a new runtime by taking out the Chrome's V8 - Engine & made it into Node.

Node.js is ✗ NOT A FRAMEWORK, instead it's a RUNTIME

↳ Extracted features of JS ! (Core JS) out of browser, directly run it in our shell, enabled more.

Since, Nodejs was able to be run in the shell it could access OS-specific features & APIs like file system, os modules etc... ☑

Nodejs → JS in shell (terminal)
↳ RUNTIME !, but difrent from Browser runtime, it eleminated all irrelivant Browser based APIs like DOM etc.. & externally provided a new set of API's that were helpfull for backend development. Relivant CRUD operations, more OS-specific archetectur features

Essentially when we run a Nodejs based file we run it using cmd
node <file name>.js → runs on system, loads js in RAM & does everything.. @ NODE environment
↓
OS - features

NOTE:
there are more <runtime> environments as well, like deno.js, bun etc..

FEATURES OF NODE JS

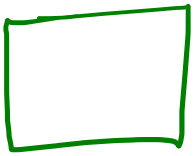
* open-source, created by Ryan Dhal realesed @ 2009.
* Brings raw JS features into our terminal so we can interact with OS-based features.

Difference b/w Runtime & application Framework:

Runtime

environment provides a set of resources for a particular program or process to run.

Framework



ex. How you cook food:

✓ buy groceries, bring home & cook

X You dont go to farm grow food & all

already done by farmer (provide resources).

{ set of industry practices already provided by a framework for us to focus on solving problems like some come w/ built in Auth etc. }

* Resources: RAM, Disk, GPU etc.. compute power (CPU)...

* features: when we install OS, os specifies a list of features to interact w/ these resources & solve some issuse provide GUI, CLI etc.. Read/write, process running in RAM ...etc.

Things we can do with Node.js, diffrence b/w Browser based APIs & Node js APIs.

Since node runs on our terminal we can,

- Make server-side applications.
- Desktop app^n
- IoT application ~> Raspberry -> Linux -> install Node -> access lots of features & solve problems in IOT. (or microcontrollers ...)
- access to file system
- Timer
- Processes & runtime variables.

* Even frontend specific libraries & framework depend on Node for their huge use-case
like React.js, Angular, Vue js, Solid, Emberjs, etc. (many ...)

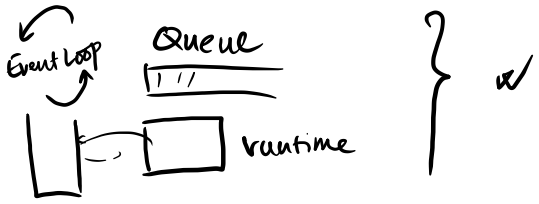
Internal features of Node.js ✓

I/O - input / output (expensive)

↓ Blocking

→ Non-blocking I/O (async) js

thread - gets blocked.



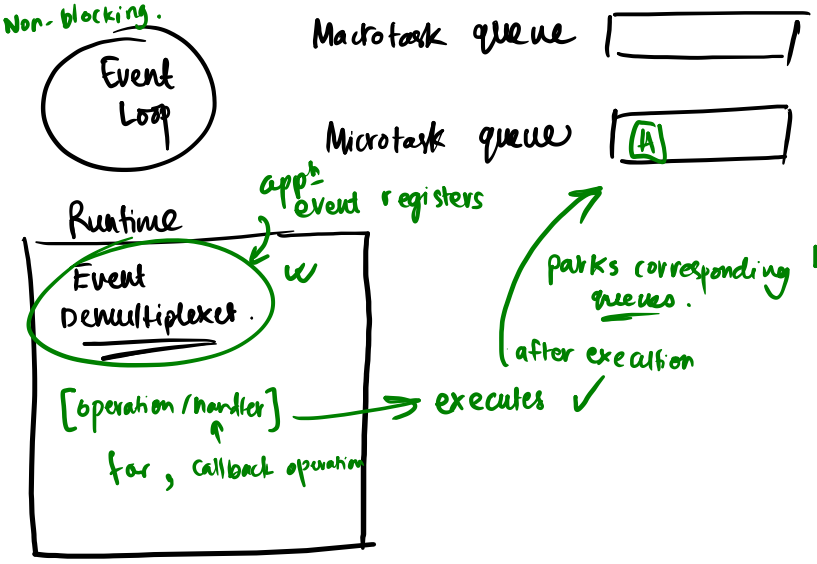
✓

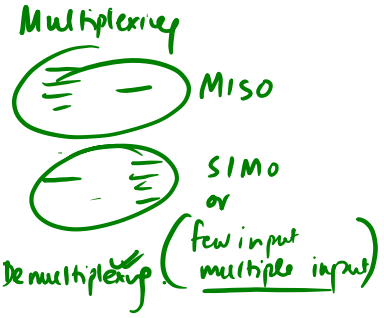
Nodejs → install LTS.
↳ version manage
↓
(nvm)

"Libuv" ?? usecase
.. node js internally uses it
.. assigns threads

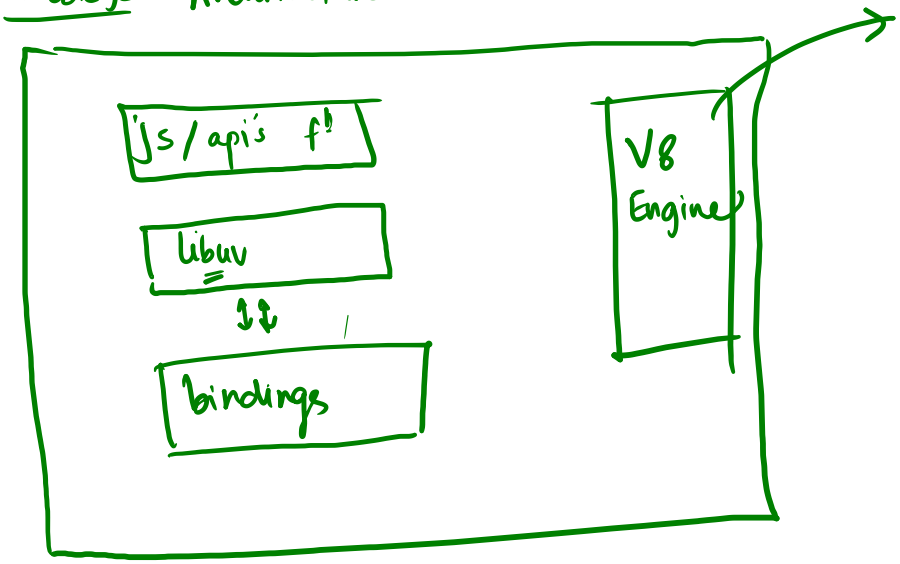
Node js architecture / mechanism :-

Non-blocking.





Node.js Architecture Introduction :-

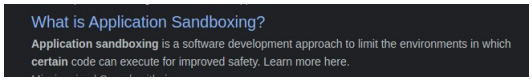


{ Js engine, developed by google for chrome }

fastest engine < for JS >

since when we ship the feature to end-to-end to a particular customer. we cant rely on their system specs.

∵ we cant depend on OS specific features so libuv is sandboxed into Node for OSspecific features & fast



ex: "fs" in Linux is blocking i/o, but node is non-blocking i/o ... etc.

library → "libuv" prepared by Node.js core team

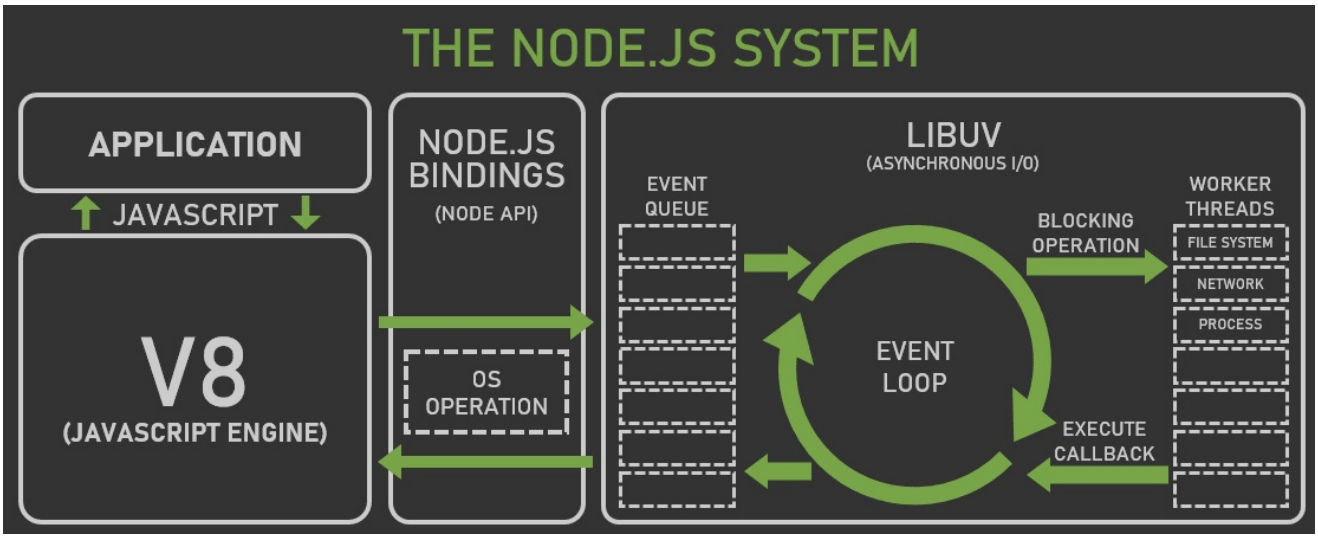
written in C++ → makes, node compatible w/ all O.S.

very exciting features into node. ✓

libuv – I/O bindings for node, low level I/O based implementations for node (consistent apis) ✓

bindings – special program that helps node to access the "libuv".

Node js works on "Event Driven Architecture".



→ Globals ✓

→ Module Pattern → Create our own, use 3rd party & make project... ✓

→ Streams ✓

Globals in Node.js

↳ Global variables available everywhere in node

→ process { helps access process variables, current process, env. var }

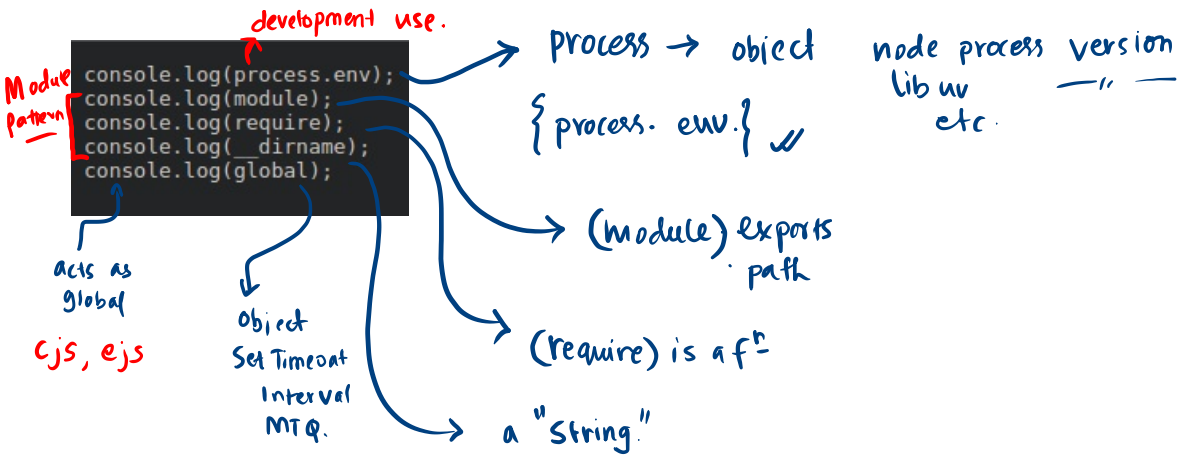
→ __dirname { access current dir. }

→ require → (other module) or DOM.

→ global → { just like window object in browser runtime }

→ module → access module pattern. ✓ of node.js

* there are cases where you cases when we can't access specific globals
ex: can't access __dirname everywhere / everytime during certain config^n.



* NOTE: never update a global var → might just be a catastrophy for others working in the project.

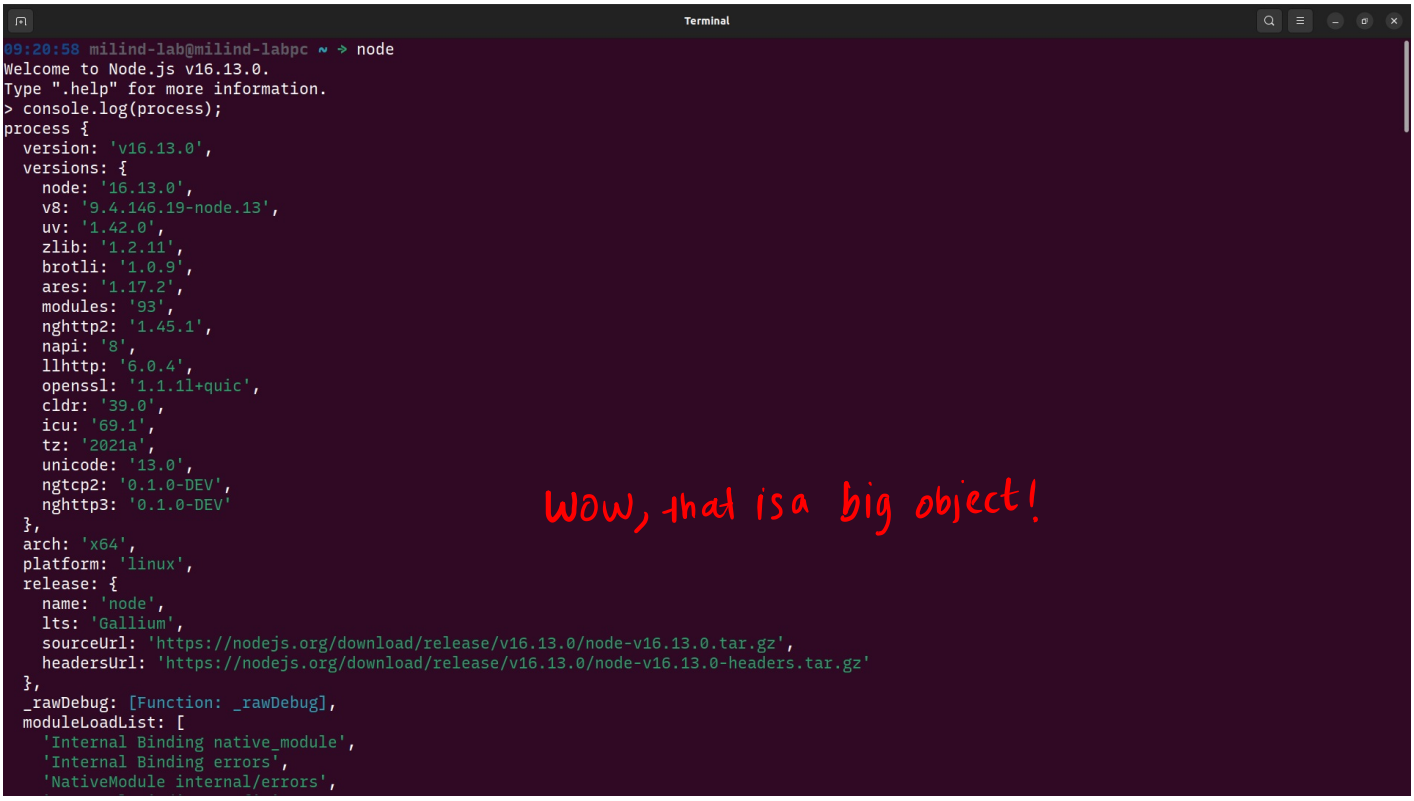
** changing globals hampers others execution of code as theres code dependencies on global objects / vars etc..

* NOTE: diffrent runtimes can implement similar feature of js diffrently.
like setTimeout in Node spits out an object & in browser it spits a number.

All you can do is just read through description of thes APIs. ✓

(wierd?)
No!
Because they're just diffrent RUNTIMES

In REPL (browser) or (Node) if nothing is returned @ current line it returns "undefined"




```
> console.log(require)
[Function: require] {
  resolve: [Function: resolve] { paths: [Function: paths] },
  main: undefined,
  extensions: [Object: null prototype] {
    '.js': [Function (anonymous)],
    '.json': [Function (anonymous)],
    '.node': [Function (anonymous)]
  },
  cache: [Object: null prototype] {}
}
undefined
>
```

```
> console.log(global)
<ref *1> Object [global] {
  global: [Circular *1],
  clearInterval: [Function: clearInterval],
  clearTimeout: [Function: clearTimeout],
  setInterval: [Function: setInterval],
  setTimeout: [Function: setTimeout] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
  queueMicrotask: [Function: queueMicrotask],
  performance: Performance {
    nodeTiming: PerformanceNodeTiming {
      name: 'node',
      entryType: 'node',
      startTime: 0,
      duration: 1189741.7345920056,
      nodeStart: 0.12087000906467438,
      v8Start: 0.8238469958305359,
      bootstrapComplete: 17.60751999914646,
      environment: 9.584022998809814,
      loopStart: 34.761327996850014,
      loopExit: -1,
      idleTime: 1189338.285553
    },
    timeOrigin: 1680960060071.907
  },
  clearImmediate: [Function: clearImmediate],
  setImmediate: [Function: setImmediate] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  }
}
undefined
>
```

NOTE:

Like discussed, the runtimes are diffrent & acts diffrently and hence "this." acts diffrently in browser vs Node. as well

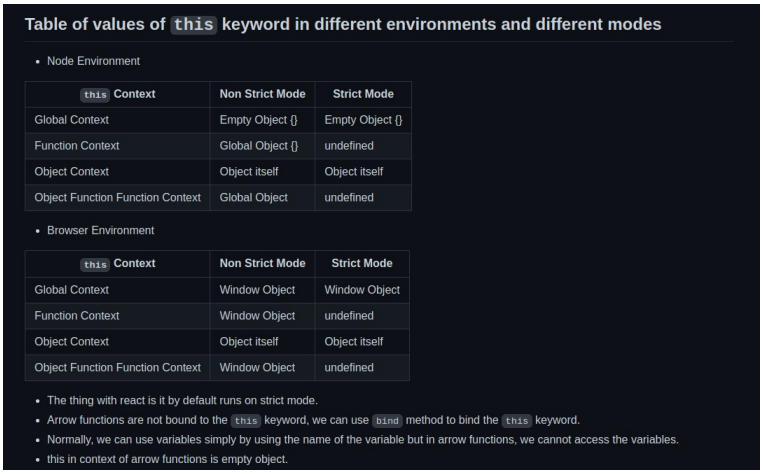


Table of values of `this` keyword in different environments and different modes

- Node Environment

`this` Context	Non Strict Mode	Strict Mode
Global Context	Empty Object {}	Empty Object {}
Function Context	Global Object {}	undefined
Object Context	Object itself	Object itself
Object Function Function Context	Global Object	undefined

- Browser Environment

`this` Context	Non Strict Mode	Strict Mode
Global Context	Window Object	Window Object
Function Context	Window Object	undefined
Object Context	Object itself	Object itself
Object Function Function Context	Window Object	undefined

- The thing with react is it by default runs on strict mode.
- Arrow functions are not bound to the `this` keyword, we can use `bind` method to bind the `this` keyword.
- Normally, we can use variables simply by using the name of the variable but in arrow functions, we cannot access the variables.
- this in context of arrow functions is empty object.