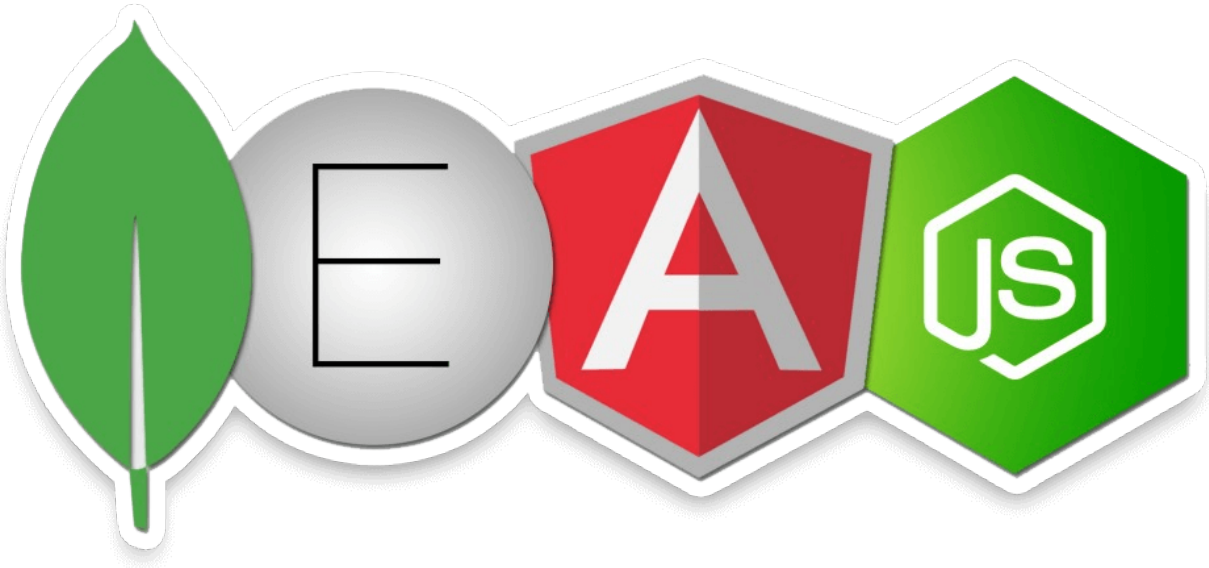
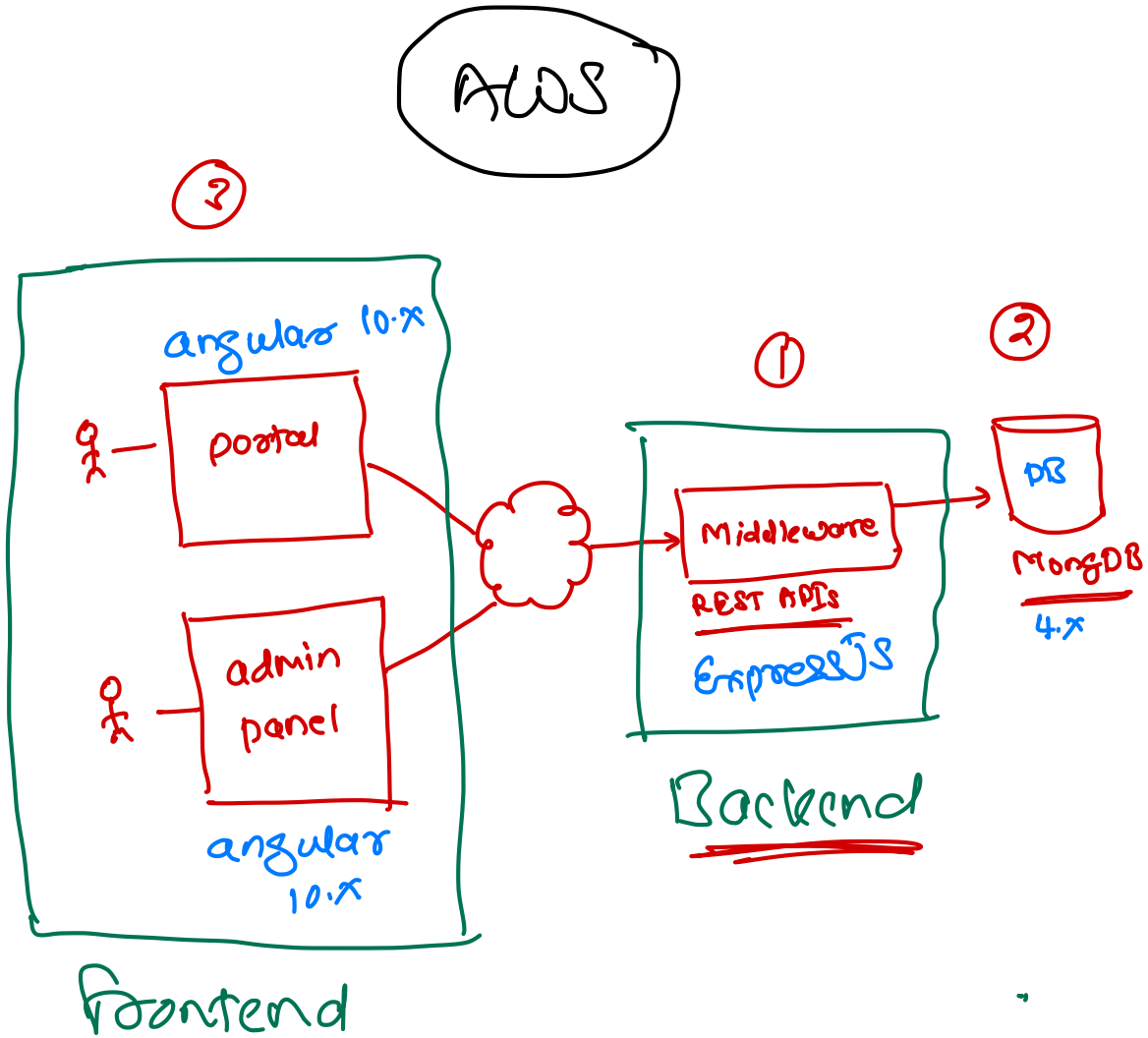

E
A
JS

Know your instructor

- 12+ years of experience
- Associate Technical Director at Sunbeam
- Freelance Developer
- Worked in various domains using different technologies
- Developed 180+ mobile applications on iOS and Android platforms
- Developed various websites using PHP, MEAN and MERN stacks
- Certifications completed
  - Certified Jenkins Engineer
  - Docker Certified Associate
  - Certified Kubernetes Application Developer

E-Commerce application

- Admin Panel
  - Product Management
  - Order Management
  - User Management
  - Reports
- Customer Portal
  - Authentication – Registration/ Login → email
  - Searching/Filtering products
  - Cart Functionality → checkout
  - Placing Order
  - Payment using Stripe
  - Tracking Product
  - Profile management
  - Product Review



Node JS + Express

JWT

Getting Started	Using ExpressJS	Authentication	Handling Payment
Web Architecture	Routes	Authorization	REST Basics
JavaScript Refresher	Advanced Routes	Sending Emails	Using Async-Await
NodeJS Basics	Node + MySQL	User Input Validation	Websockets & Socket.io
Architecture	Node + MongoDB	Error Handling	GraphQL
Sync vs Async	Using Mongoose	File Upload and Download	Testing
Modules	Sessions and Cookies	Pagination	Deployment

Angular

deploy → AWS → HTTPD / Apache2 / NGINX
application

Typescript	Getting Started	Forms	Pipes
Why Typescript?	Client Vs Server Side	User Input Validation	Consuming REST APIs
Types	Why Angular?	Services	Authentication
Namespace	Angular Architecture	Dependency Injection	Authorization
Classes	Components	Basic Routing	Optimizations
Decorators	Databinding	Advanced Routing	Testing
Modules	Observable	Directive	Deployment

MongoDB

- Getting Started
- SQL vs NoSQL
- MongDB Basics
- Basic CRUD Operations
- JSON Schema & Relations
- Working With Shells
- Advanced - Create
- Advanced - Read
- Advanced - Update
- Advanced - Delete
- Optimizing With Index
- Geospatial Queries
- Aggregation Framework
- Security
- Authentication
- Replication
- Sharding
- Transactions
- Shell To Drivers
- Deployment
- Backup & Restore

What will not be covered?

- HTML/CSS Programming
- Bootstrap framework
- JS Programming
- Socket programming

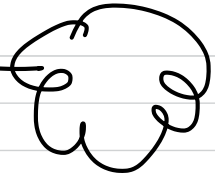
Fundamentals

Client side

- → html + CSS
- → JavaScript

Server side

PHP, C#, JavaScript –
Java, Python, Ruby, –



HTTP Request

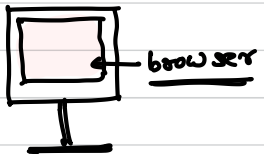
Header

path: index.html
port no: 80
method: GET
scheme: http
query string: –
⋮

Body

[Query String Param]
for method POST / PUT

http:// mydomain.com/index.html



CLIENT

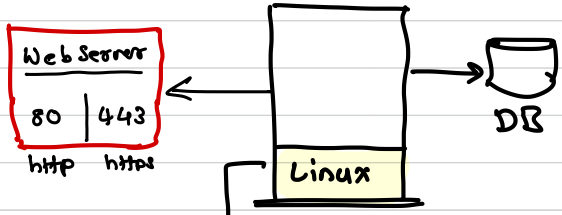
HTTP Response

header

status code: 200
content-length: 100KB
content-type: text/html
⋮

body

[contents of requested file]



SERVER
18.5.6.7
mydomain.com

→ /var/www/html

Resources
- → index.html
- → myprofile.php
- → image.jpg
- → background.mps
- ⋮

Status code

1xx – debugging / informational messages
2xx – SUCCESS (200, 201)
3xx – Redirection (300, 301)
4xx – Client Error (400, 401, 403, 404)
5xx – Server Error (500, 501)

URL

→ Uniform Resource Locator

http://google.com
https://17.5.8.3/index.html
http://mydomain.com:8080/
http://mydomain.com/mypage.php?name=person1 (Query string)
http://mydomain.com/mypage.html#top (proxy)

→ Components

1. scheme : http, https, ftp, file
2. domain name / IP address : server address
3. port number : of web server [http:80, https:443]
4. path / file name : file
5. Query string : input passed to the file
6. Proxy / Hash component : used to link parts of pages

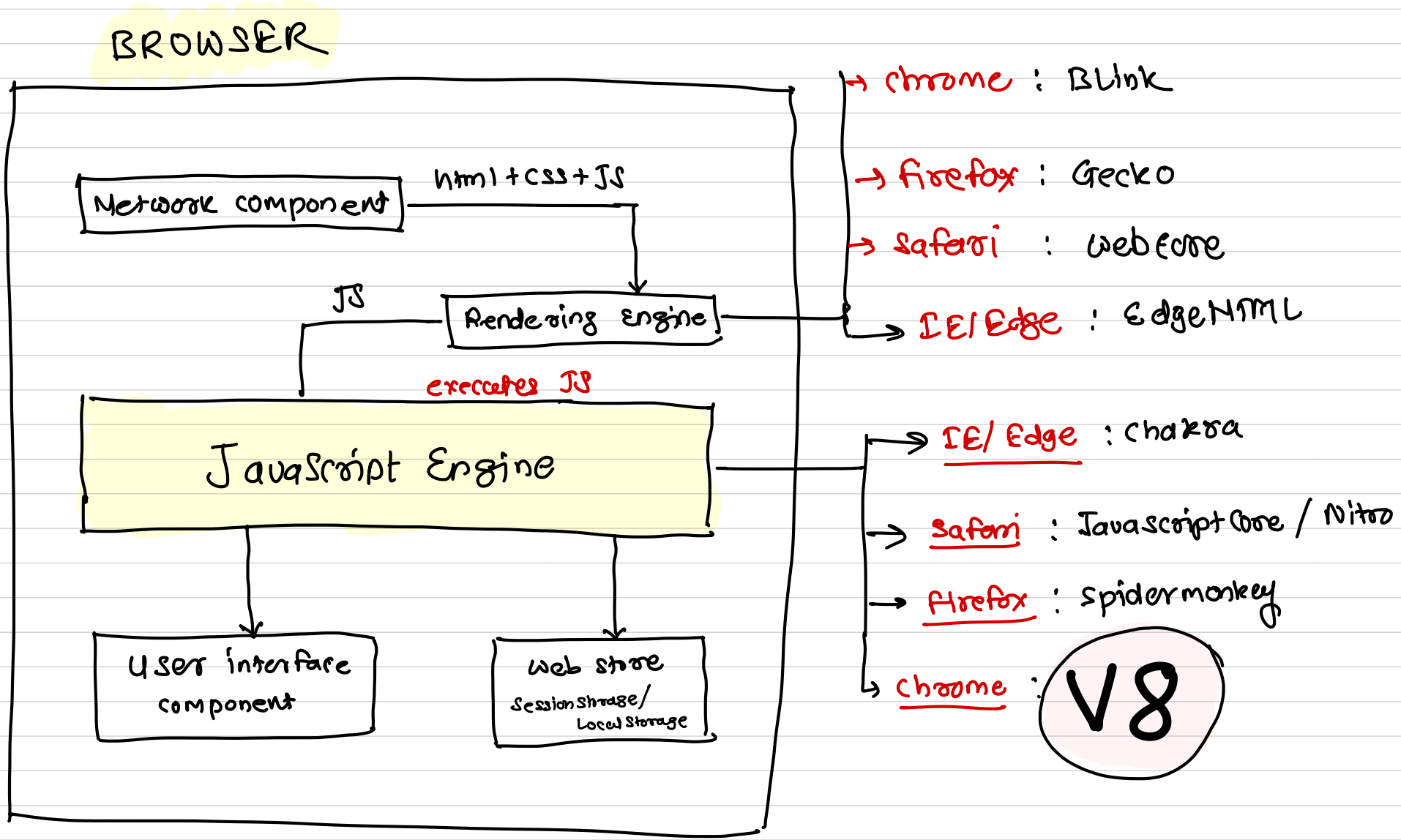
server
- process which serves user
- types

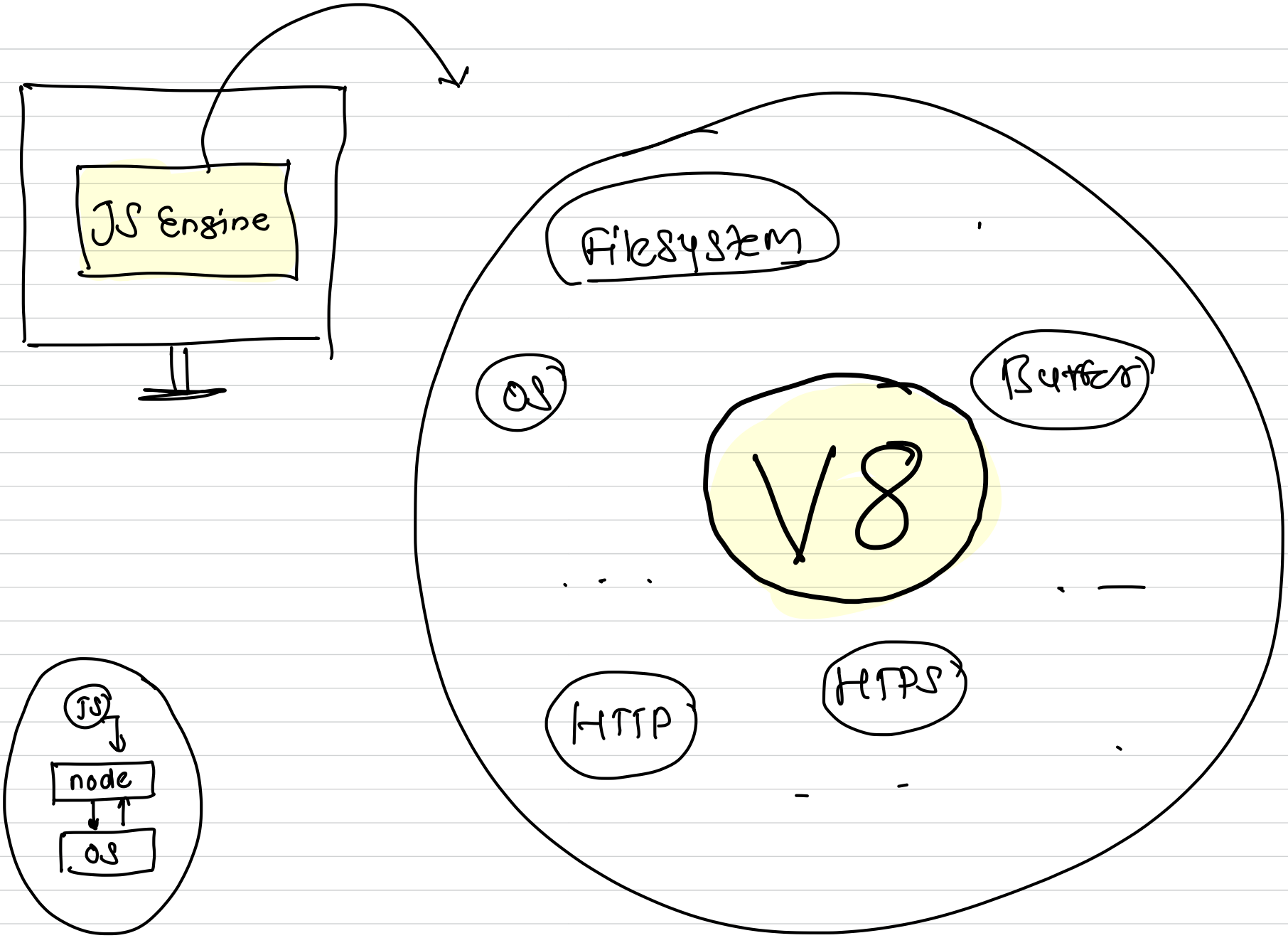
web server
- serves http Requests
- eg: Apache, NGINX, Glassfish, IIS

Database Server
- RDBMS : MySQL, Oracle, ...
→ NoSQL : MongoDB, Firebase...

File server
- NFS

Application server
- Tomcat


BROWSER
Network component
html + CSS + JS
Rendering Engine
JS
executes JS
Javascript Engine
User interface component
Web store
Session Storage/
Local Storage
chrome : Blink
firefox : Gecko
safari : webcore
IE/Edge : EdgeHTML
IE/Edge : chakra
safari : Javascript Core / Nitro
firefox : spidermonkey
chrome : V8



NodeJS

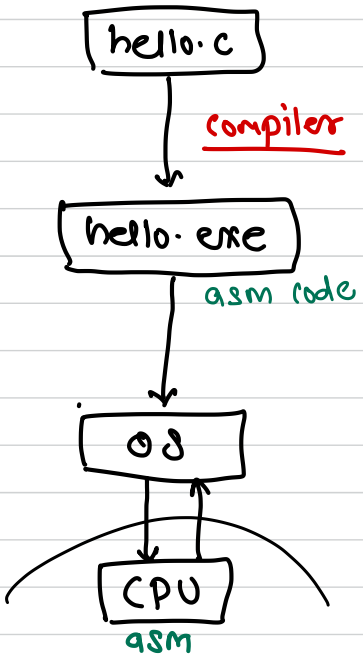
Platform : NodeJS

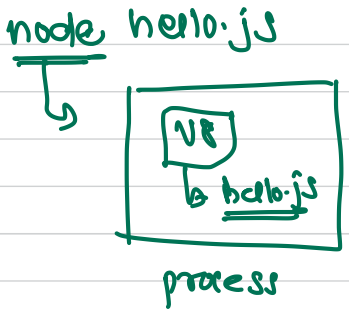
Webserver : Express

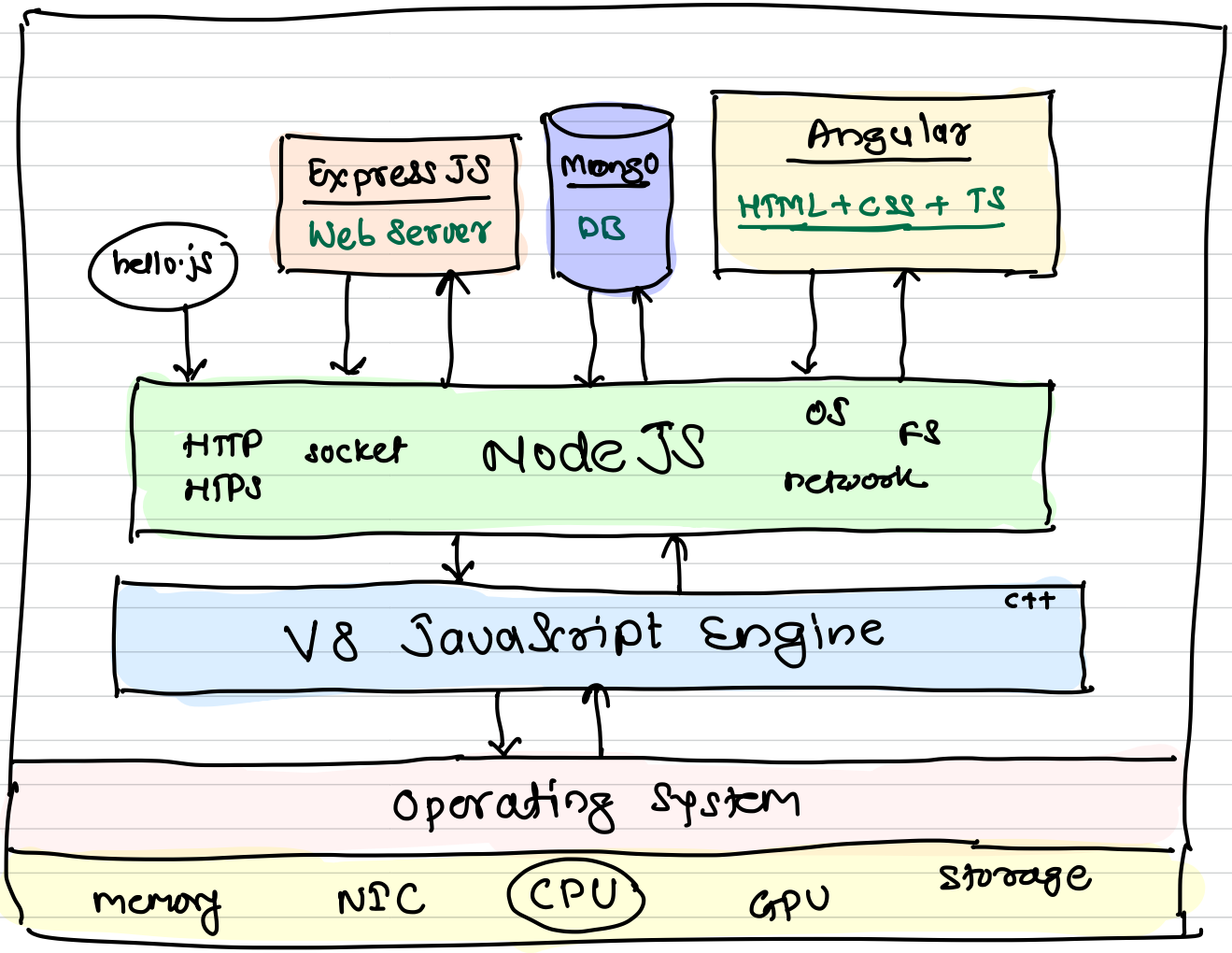
MEAN

Database: Mongo

Client side framework : Angular







SERVER

Web Architecture

- Web uses client server architecture
- Client
  - The machine which uses the browser to browse a website
  - Client machine sends a request to the server demanding the required resource
- Server
  - The machine which runs a server program
  - The server program serves a request
  - Types
    - Web server
    - File server
    - Database server

Web Architecture

- It uses request-response pattern
- Request
  - An object created by the client (browser)
  - Contains all the information required by the server to send the response
  - Header contains
    - HTTP Method, url, port, required resource, user-agent etc
  - Body contains the body parameters
- Response
  - An object created by the server which contains the requested resource
  - Header contains
    - Status code, server type, content-type, content size
  - Body contains the requested resource

Browser

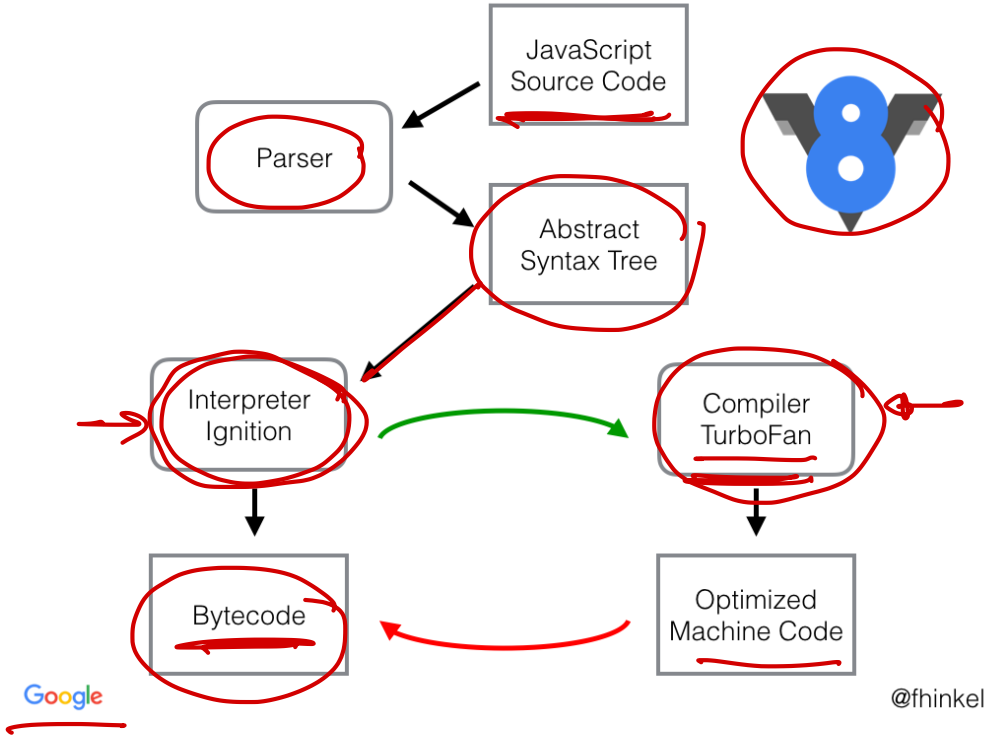
- A special application used to browse the websites
- It contains parsers for HTML, XML, CSS, Text etc. which are used to parse HTML and CSS and can produce the UI
- It executes the JS code using JS engine
- Any browser contains
  - User interface component
  - Browser Engine (JS Engine)
  - Rendering engine
  - Networking component
  - JS interpreter
  - Data persistence component

JavaScript Engine

- E.g.
  - Browser: Chakra, JavaScriptCore, SpiderMonkey, V8
  - Node JS: V8, Chakra, SpiderNode
  - Electron: V8
  - IoT: Duktape, JerryScript

V8

- V8 is Google's open source high-performance JavaScript and WebAssembly engine
- It is written in C++
- It is used in Chrome and Node JS
- Built on top of ECMAScript standard



Node JS

What is Node ?

- Node.js is a JavaScript runtime built on Google's V8 JavaScript engine
  - The same V8 engine also powers the chrome
- It is a platform to develop server sided and networking applications in JS
- Developed Ryan Dahl in 2009
- It is free and open source
- It can run on various Operating Systems like macOS, Linux and Windows

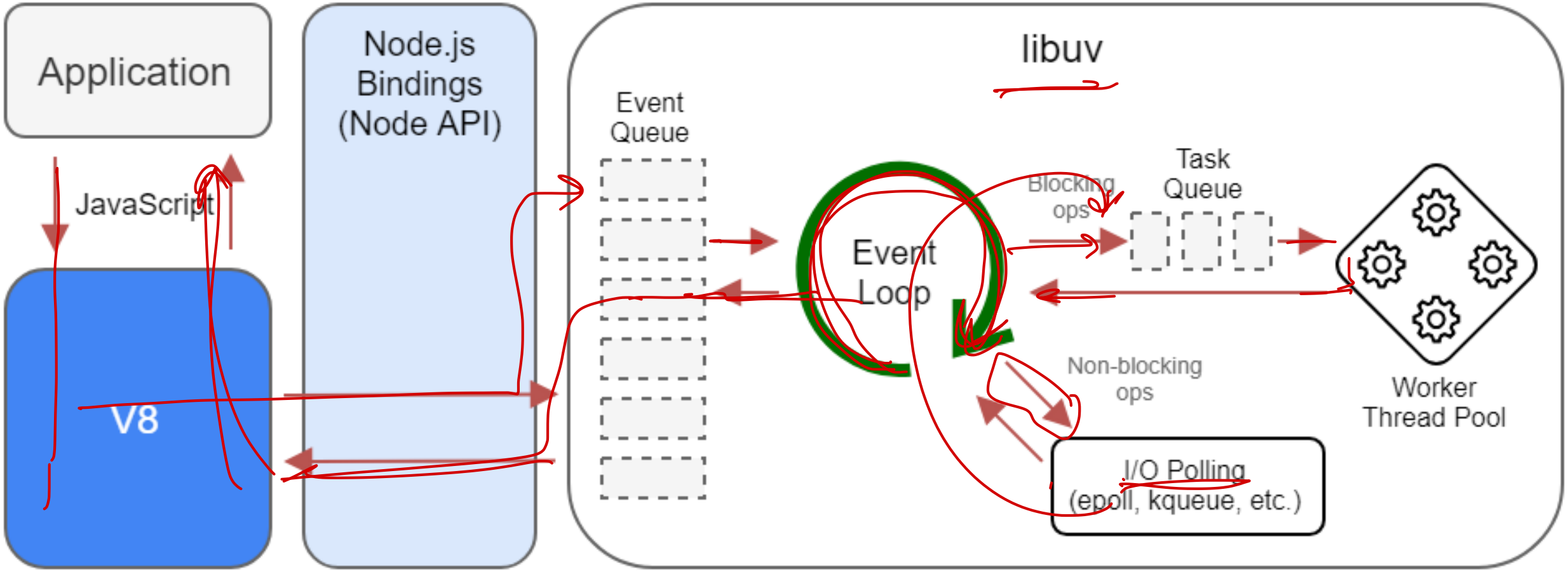
Node JS Components

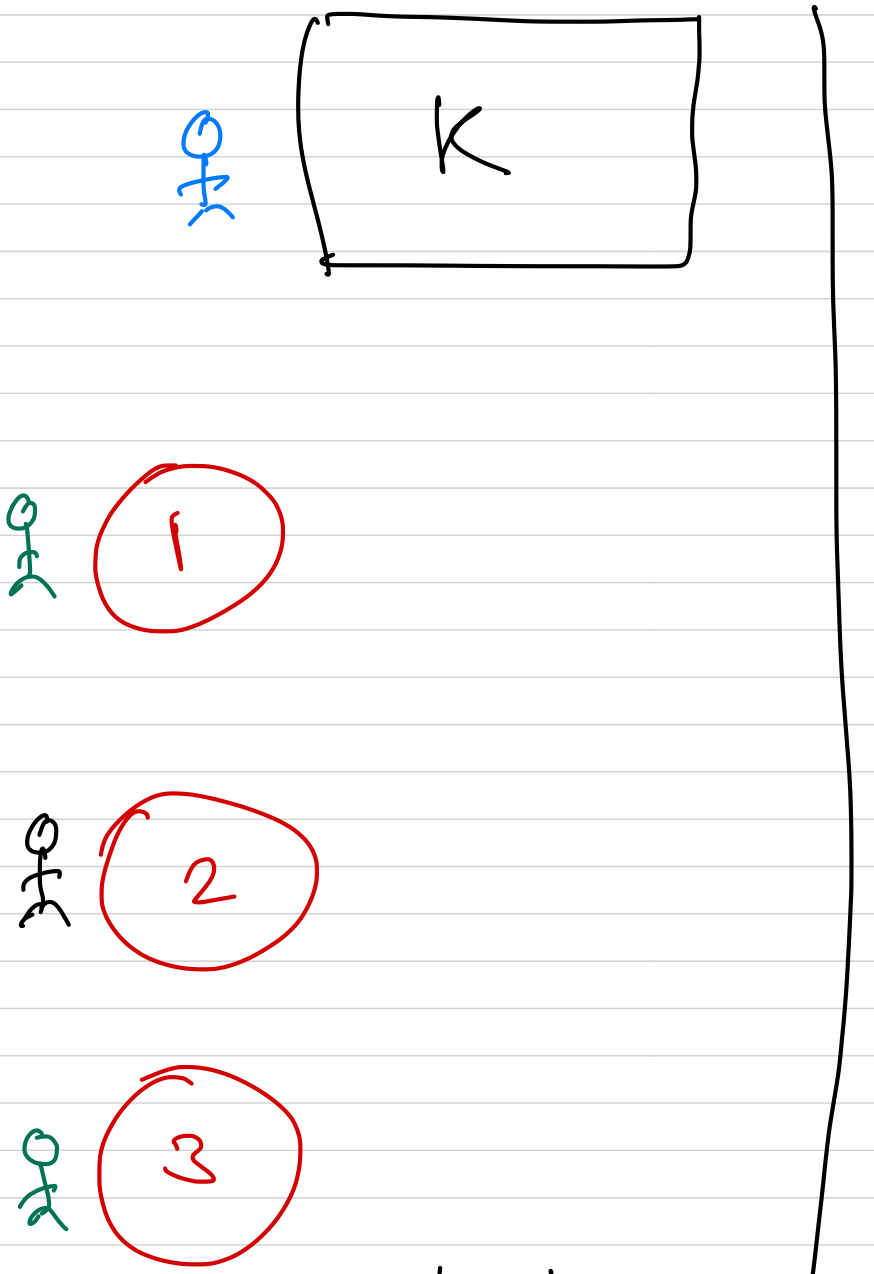
- JavaScript Runtime: V8 (you can use different engine)
  chakra
  SpinderNode
- Event Loop: Libuv
- Standard Library
  - Filesystem Access
  - Crypto
  - TCP/UDP
  - HTTP
  - Buffer
  - etc

Misconceptions

- Node JS is single threaded
  - Not all parts of NodeJs are single threaded
  - Event Loop and V8 run in single thread. It has single main call stack.
  - Your code has single threaded. If you run every expensive application then you block event loop
- Node JS uses a thread pool for all IO
  - Threads need to block while we do the IO so Node does not use Thread Pool for IO
  - It uses thread pool
    - when there is no OS async API that allows it do the non-blocking API
    - For crypto work
    - For DNS lookup
- The Event Loop runs in one thread and V8 runs in separate thread

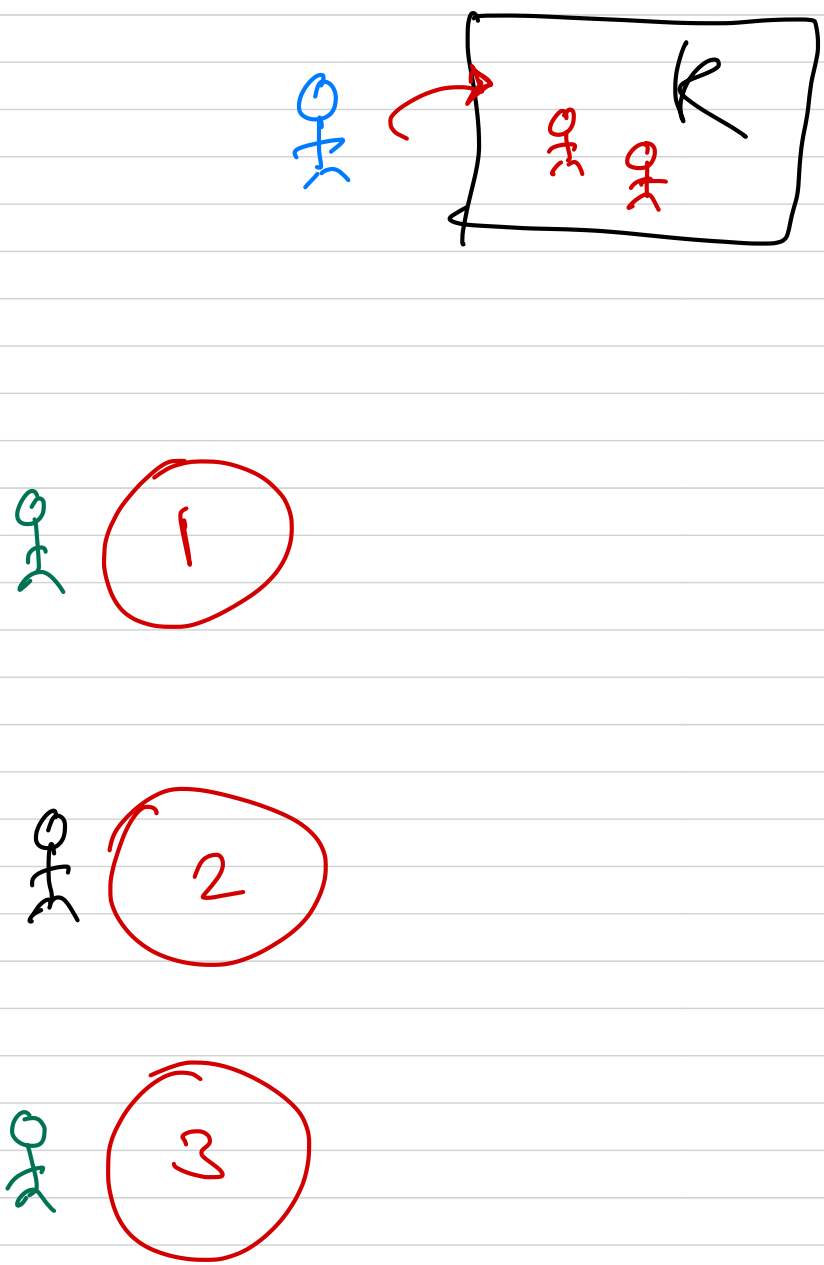
Architecture





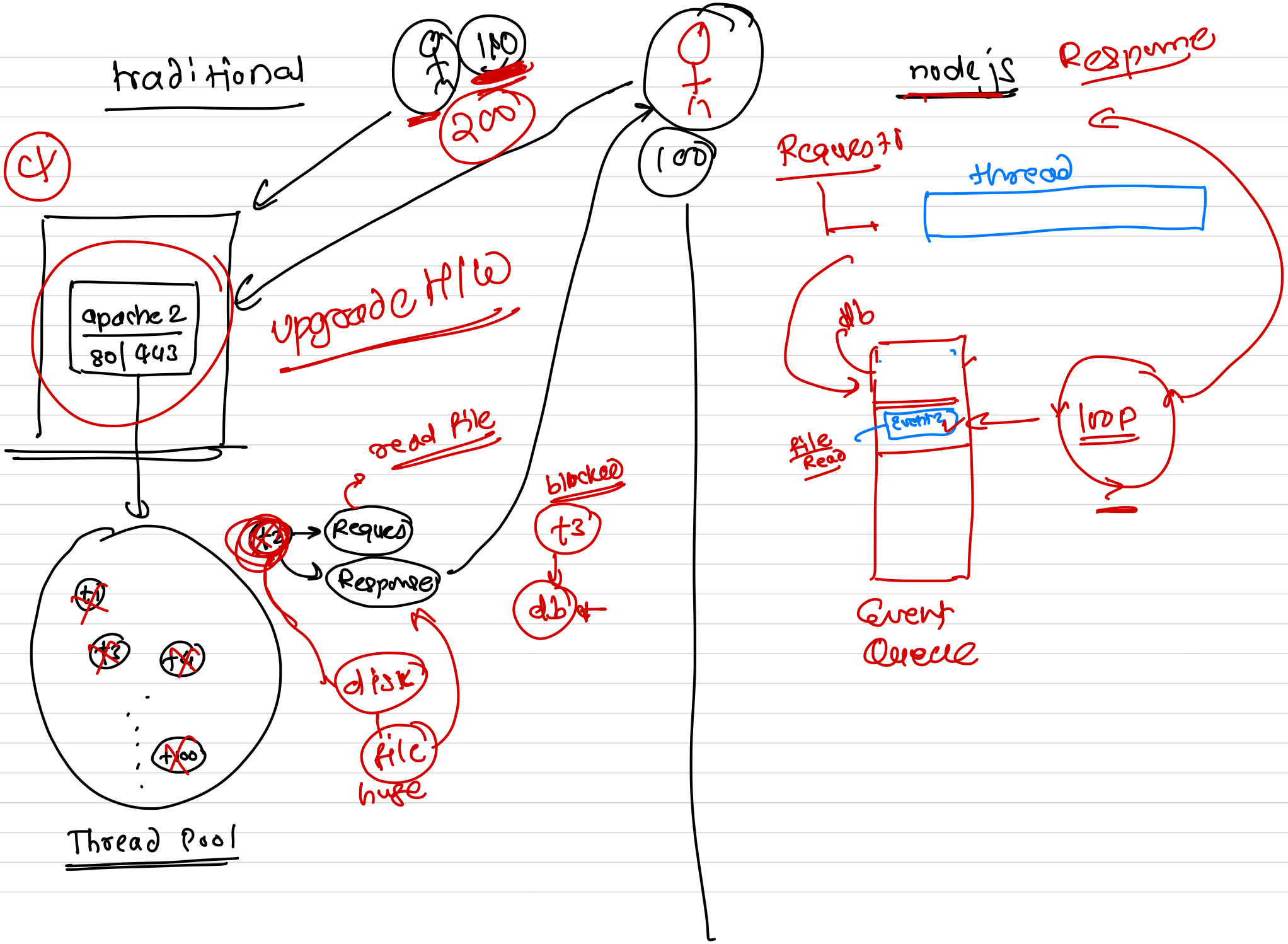
Blocking

Sync



Non-blocking

Asynchronous


traditional
100
200
100
node js
Response
Request
thread
apache 2
80 | 443
upgrade H/W
read file
Request
Response
blocked
t3
db
disk
file
huge
Thread Pool
db
file Read
event 2
loop
Event Queue

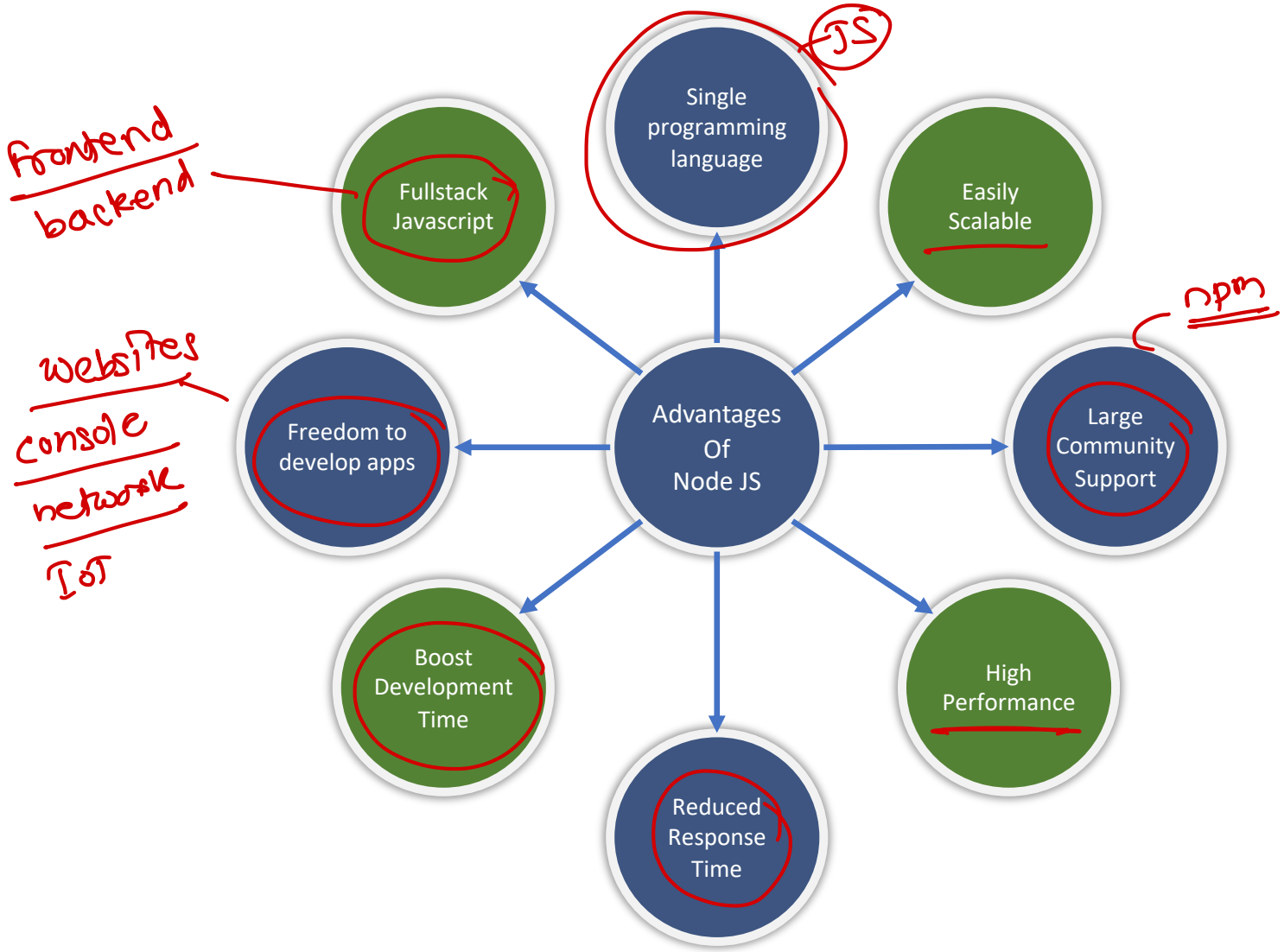
NodeJs Features

- It uses event-driven, non-blocking I/O model
- It has a single threaded but highly scalable architecture
- It is a lightweight and efficient
- It has its own package ecosystem which has thousands of packages for different purpose npm
- It never buffers the data. Rather it simply outputs the data in chunks

NodeJs Usecases

- Where to use NodeJs
  - I/O bound applications
  - Data streaming applications
  - IoT applications
  - JSON API based applications
  - Single Page Applications
- Where not to use NodeJs
  - CPU intensive applications
  - Heavy server sided processing

Advantages of Node JS



JavaScript

JavaScript

- Weekly Typed Language
  - No explicit type assignment
  - Data types can be switched dynamically
- Data types are inferred
  - Data types are assigned by JS by inspecting value in the variable
- Object Oriented language
  - Data can be organized in logical objects
  - Primitive and reference types
- Versatile language
  - Runs in browser as well as directly on desktop
  - Can perform variety of tasks

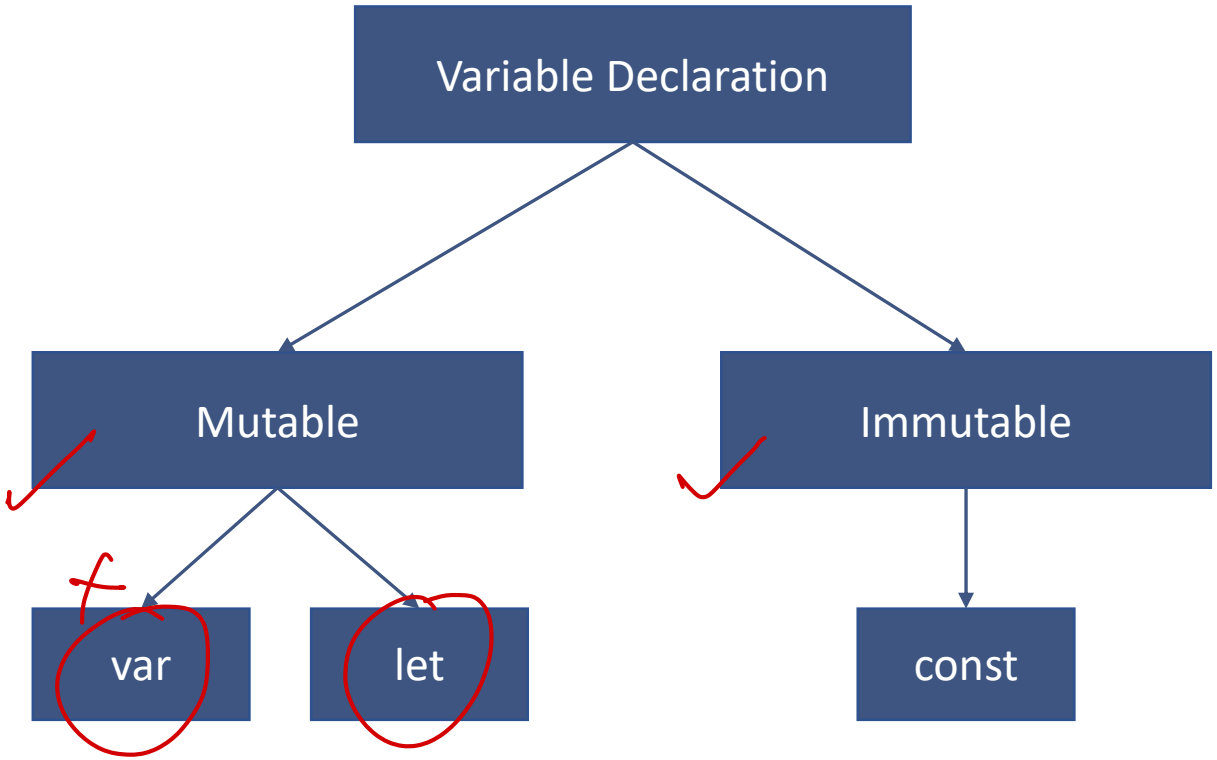
"steve" – string

100 – number

Core Language Features

- Variables
- Constants
- Object, Properties and methods
- Function
- Array
- Loop

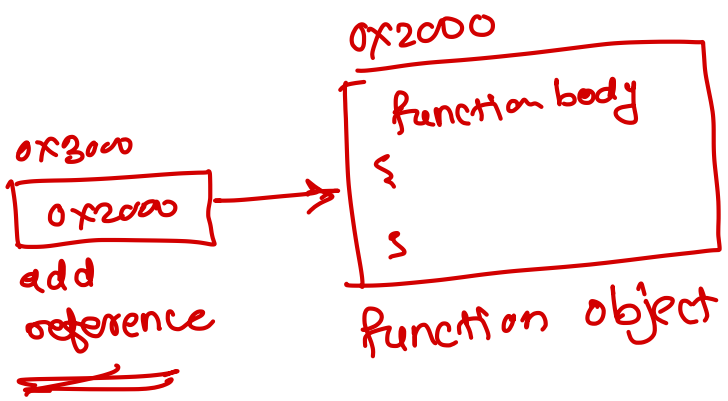
Variables and Constants




```
// function scoped
// redeclaration allowed
var num = 100

// block scoped
// redeclaratoin is not allowed
let firstName = 'steve'

// constant
// once declared, can not change
const address = pune
```

Functions

```
// named function
function add(p1, p2) {
  const addition = p1 + p2
  console.log(`addition = ${addition}`)
}

add(10, 20)
```




```
// arrow function
const add = (p1, p2) => {
  const addition = p1 + p2
  console.log(`addition = ${addition}`)
}

add(10, 20)
```

Objects

- Reference type
- Collection of Key-Value pairs
- Key must be string
- Value can be of any type
  - String
  - Number
  - Object
  - Boolean
  - Array
  - Function
- Object can be created using
  - JSON
  - Object
  - Constructor function

```
// person is a reference
const person = {
  name: "person 1",
  address: "Pune",
  age: 30,
  married: true,
  canVote: function () {
    if (this.age >= 18) {
      console.log("can vote")
    } else {
      console.log("can not vote")
    }
  }
}

// calling function
person.canVote()
```

Array

- Collection of values
- [] is used to create an array
- Array can hold similar or dissimilar objects
- Provides methods like
  - push
  - pop
  - insert
  - splice
  - includes
  - indexOf
  - map
  - filter
  - reduce
  - join

```
// collection of numbers
const numbers = [10, 20, 30, 40]

// collection of strings
const names = ["person1", "person2"]

// collection of mixed values
const mixed = [10, "test", true]

// collection of objects
const persons = [
  { name: "person1", age: 20 },
  { name: "person2", age: 22 },
  { name: "person3", age: 23 }
]
```

Loops

```
const numbers = [10, 20, 30, 40]

// for..of loop
for (const value of numbers) {
  console.log(`number = ${value}`)
}
```

```
const numbers = [10, 20, 30, 40]

// for..each loop
numbers.forEach(number => {
  console.log(`number = ${number}`)
});
```

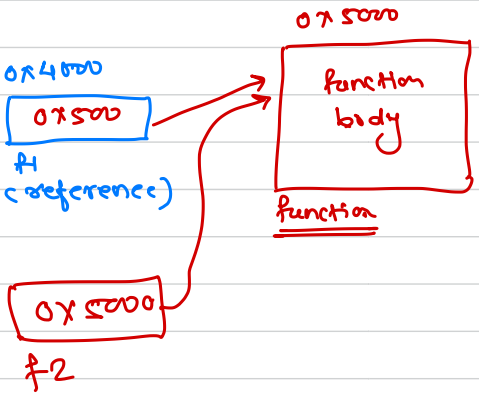
```
const numbers = [10, 20, 30, 40]

// traditional for loop
for (let index = 0; index < numbers.length; index++) {
  const number = numbers[index];
  console.log(`number = ${number}`)
}
```

Reference type

function f1 () {

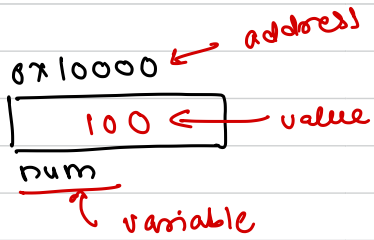
}

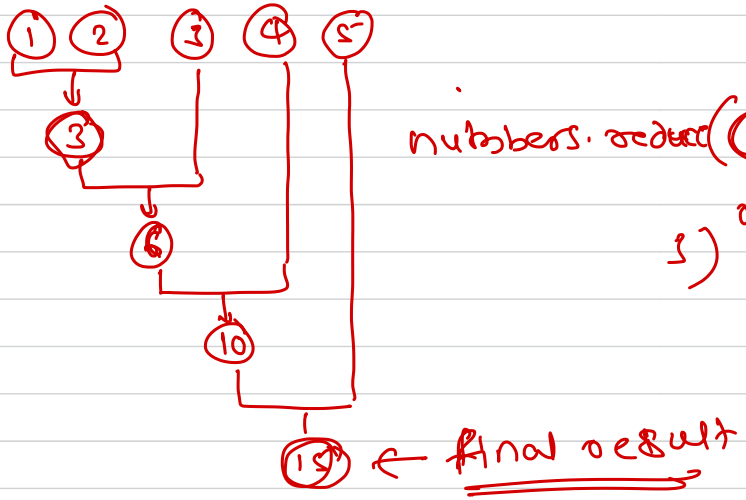
f2 = f1



value type

num = 100





numbers.reduce((x, y) ⇒ {
return x + y
})

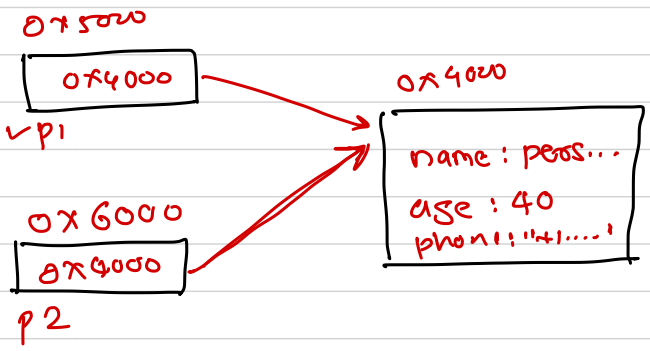
1 2

Spread and Rest Operators

```
// original array
const numbers = [10, 20, 30, 40, 50]

// copy of old array
const newArray = [...numbers]

// original object
const person = {
  name: "person 1",
  age: 20,
  address: "pune"
}

// copy of old object
const newPerson = {
  ...person,
  phone: "+911234123412"
}
```

```
// rest operator
function add(...args) {
  let addition = 0
  for (const value of args) {
    addition += value
  }

  console.log(`addition = ${addition}`)
}

add(10, 20, 30, 40)
```

```
const p1 = {
    name : 'person1',
    age : 40
}

const p2 = p1
p2.phone = "+91....."
```



Destructuring

```
const person = {
  name: "person 1",
  age: 20,
  address: "pune"
}

// destructuring object into variables
const { name, age, address } = person
console.log(`name: ${name}`)
console.log(`age: ${age}`)
console.log(`address: ${address}`)
```

const [n1, n2, n3] = [10, 20, 30]

```
const numbers = [10, 20, 30, 40]

// destructuring array
const [n1, n2, n3, n4] = numbers
console.log(`n1 = ${n1}`)
console.log(`n2 = ${n2}`)
console.log(`n3 = ${n3}`)
console.log(`n4 = ${n4}`)
```

Modules

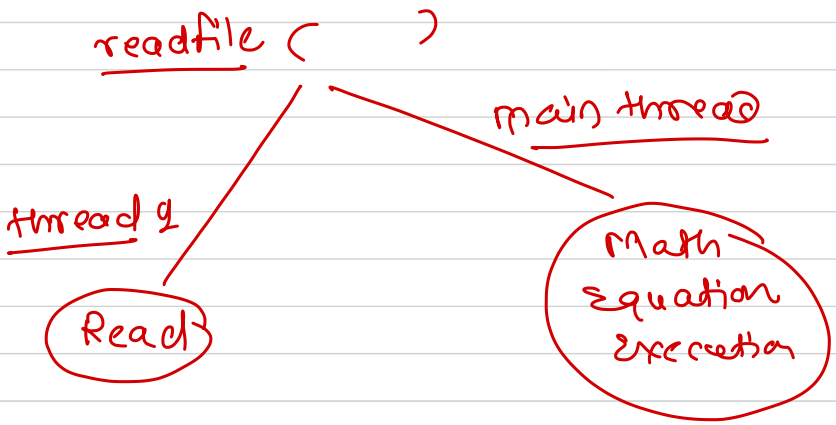
Module (library) .js

- Collection of functions
- JavaScript libraries which can be shared with multiple applications
- Types
  - Built-in modules
    - HTTP
    - FileSystem
    - Buffer
    - OS
  - User defined modules

* Every module has an object add by node ⇒ module

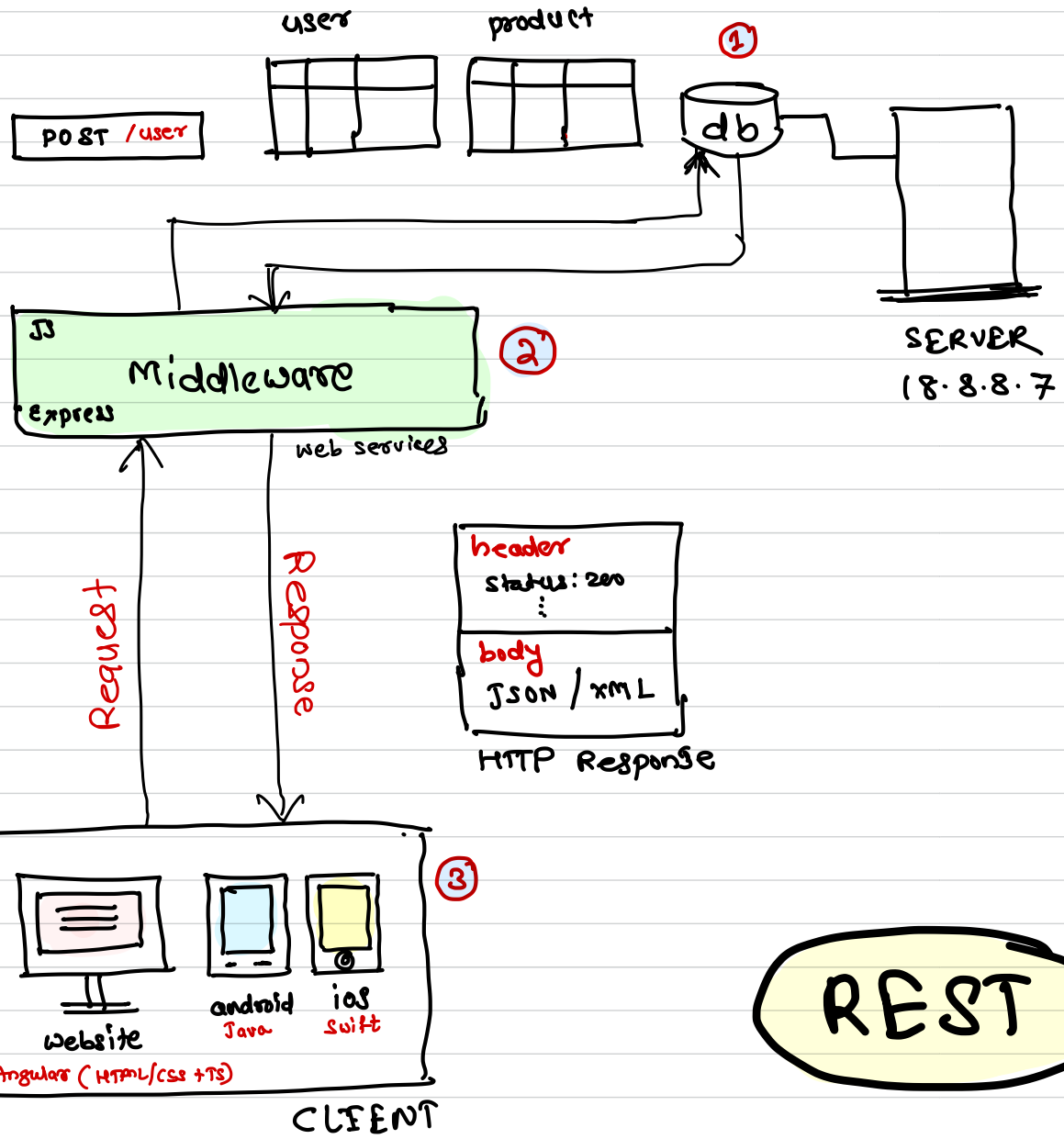
```
// math.js

function add(p1, p2) {
  console.log(`addition = ${p1 + p2}`)
}

// export all the entities need to
// be exported from this module
module.exports = {
  add: add
}
```

```
// importing math module
const math = require('./math.js')

// calling add function exported
// from math module
const result = math.add(10, 20)
console.log(`result = ${result}`)
```


readfile ( )
main thread
thread 1
Read
Math
Equation
Execution

path = <path>

Method	operation
POST	insert into <table>...
GET	select * from <table>...
PUT	update <table> .....
DELETE	delete from <table>.....



url (path) = <path>

Requirement	Method
Create record	POST
Read records	GET
Update record	PUT
Delete record	DELETE

HTTP Request

REST