Node.js

Node.js

JavaScript on the server

Asynchronous event-driven JavaScript runtime

Lightweight

Designed to build scalable runtime applications

Designed with streaming and low latency in mind

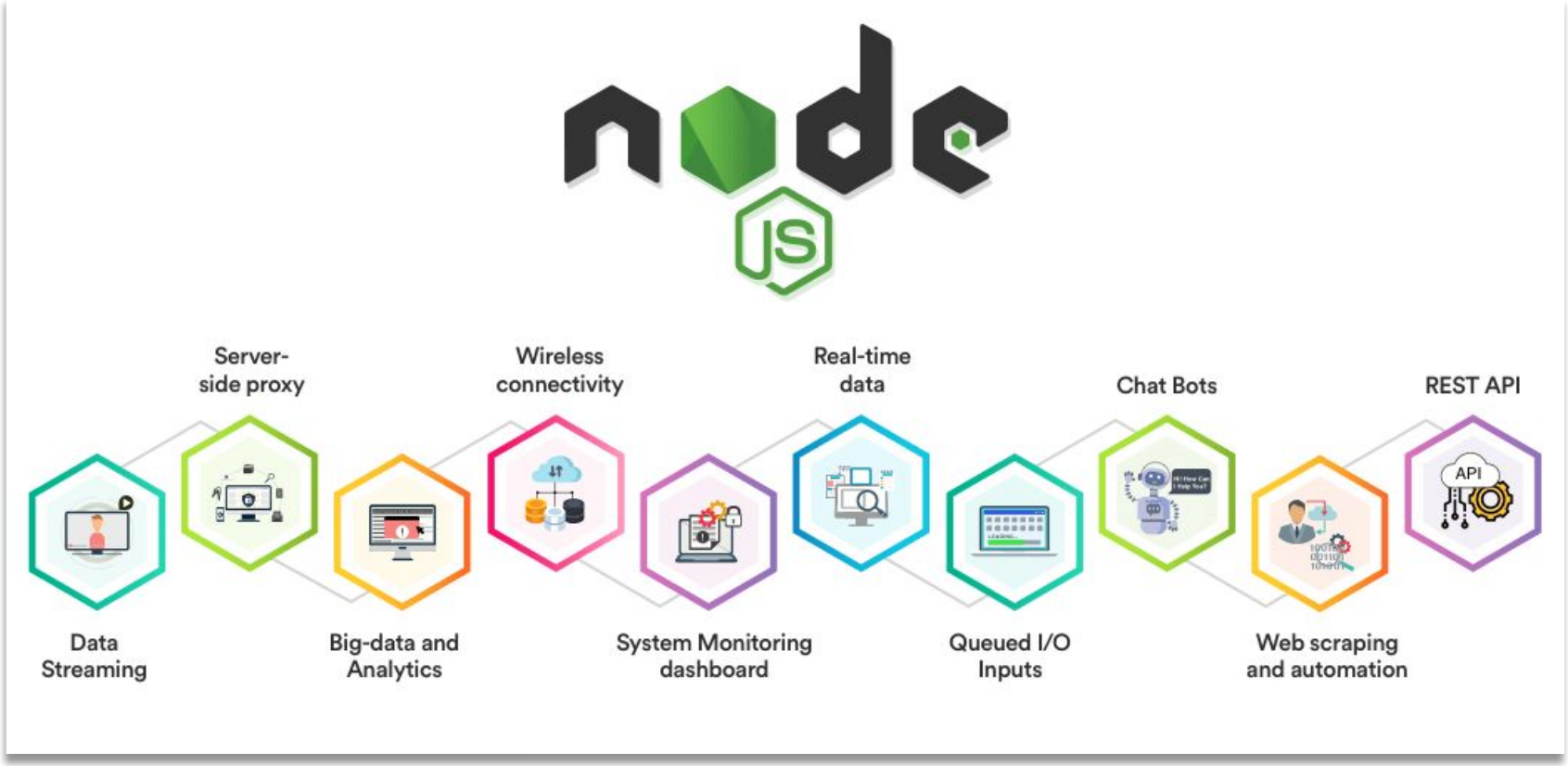
Makes Node.js as a foundation of web library or framework

Built on V8 (Google's opensource high-performance JavaScript)

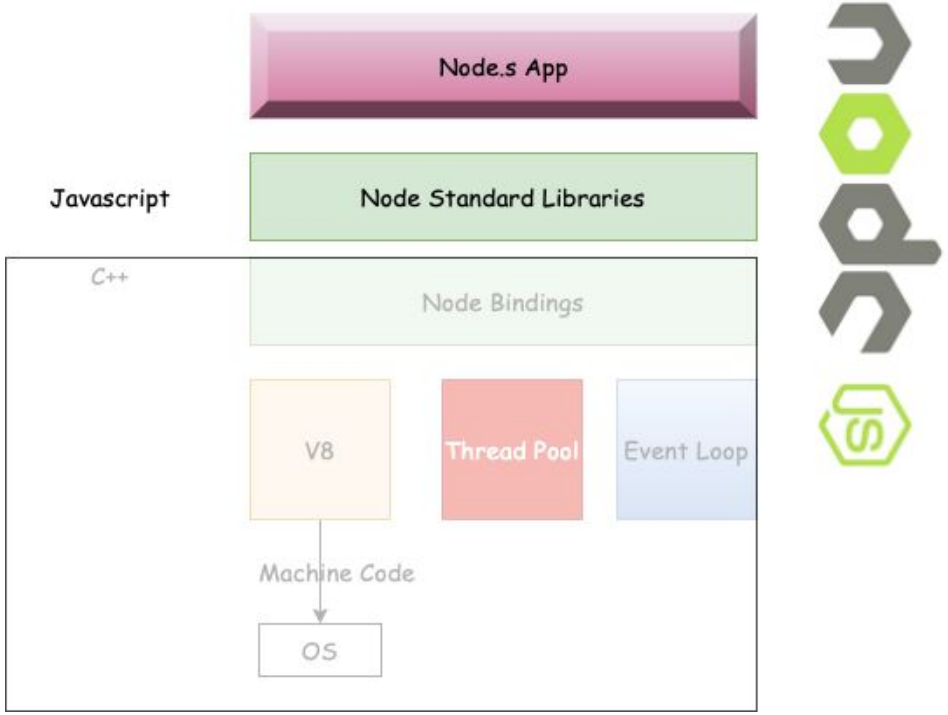
Easy to learn

Massive library support

Use-cases of Node.js

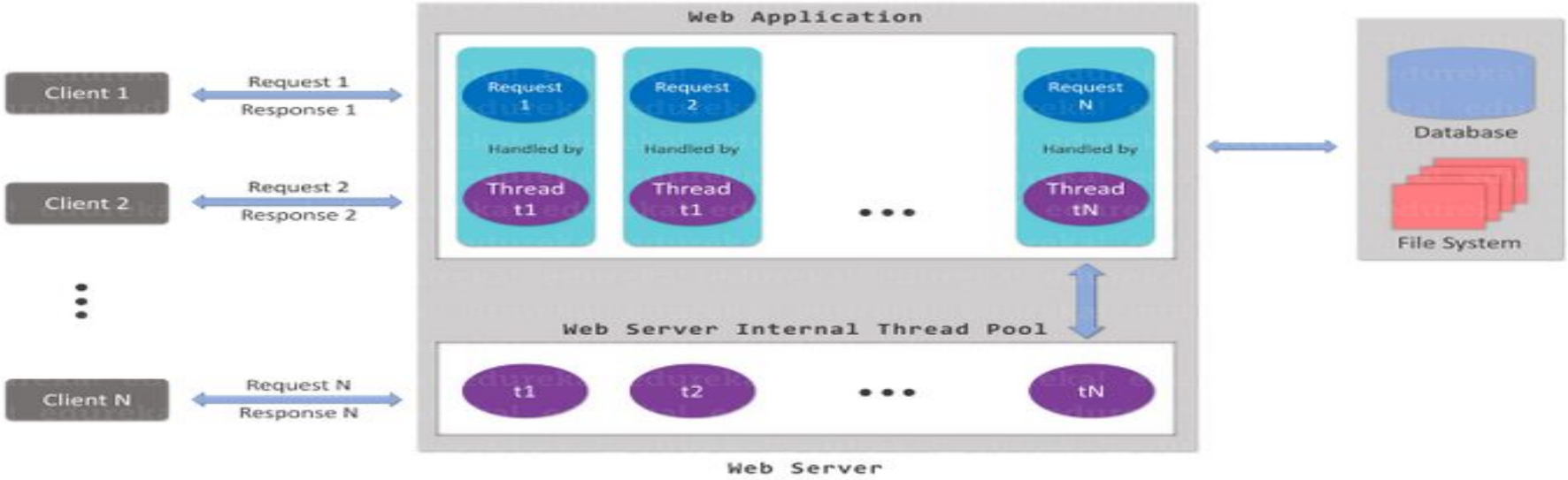


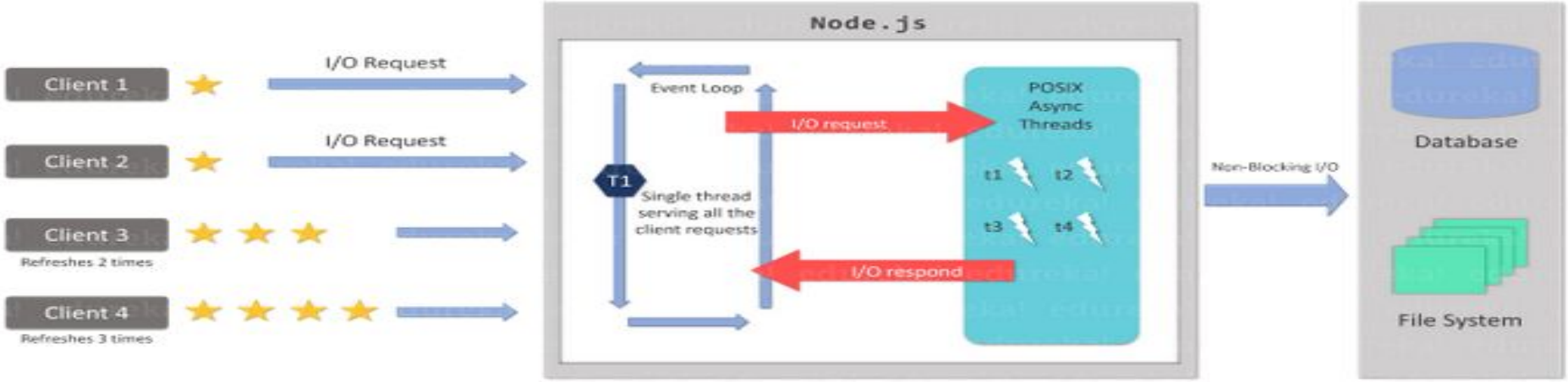
Node.js Internals



Difference between Node.js & JS

JavaScript	Node.js
Used for writing scripts on the website.	JavaScript runtime environment
JavaScript can only be run in the browsers	JavaScript on the server side (outside browser)
capable to work with HTML and play with the DOM.	Does not have the capability to add HTML tags
Can run in any browser engine as like JS core in safari, Firefox..	Can only run in V8 engine.
JavaScript is used in frontend development.	Nodejs is used in server-side development.
Some of the JavaScript frameworks are RamdaJS, TypeJS, etc.	Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm.


Web Application
Client 1
Request 1
Response 1
Client 2
Request 2
Response 2
Client N
Request N
Response N
Request 1
Handled by
Thread t1
Request 2
Handled by
Thread t1
Request N
Handled by
Thread tN
Web Server Internal Thread Pool
t1
t2
tN
Web Server
Database
File System


Node.js
Client 1
I/O Request
Client 2
I/O Request
Client 3
Refreshes 2 times
Client 4
Refreshes 3 times
Event Loop
I/O request
T1
Single thread serving all the client requests
I/O respond
POSIX Async Threads
t1
t2
t3
t4
Non-Blocking I/O
Database
File System

Install Node.js

Website

- https://nodejs.org/en/

Check

- node –version

Package/ modules

- npm install <package-name>

Initiate

- node init

Node.js Examples

```
console.log('Welcome to FSD-2 course');
```

Node.js Examples

```
var message = 'FSD Course';
console.log(message);
```

Node.js writing a function

```
function Hello() {
    console.log('Welcome to FSD-2 course');
}
Hello();
```

Node.js writing a function

```
function Hello(subject) {
    console.log('Welcome to ' + subject +' course');
}
Hello("FSD-2");
```

Node.js Modules

- Node programs can be organized as modules
- A module is a file that's exports a scope – file management
  - Contains public functions
  - Contains shared objects
- Modules are imported through the *require* function

Three types of modules

- Core Modules – Built-in modules part of the platform
- Local Modules – Application based modules
- Third-party Modules – Third party modules

Core Modules

Core Modules	Description
http	creates an HTTP server in Node.js.
assert	set of assertion functions useful for testing.
fs	used to handle file system.
path	includes methods to deal with file paths.
process	provides information and control about the current Node.js process.
os	provides information about the operating system.
querystring	utility used for parsing and formatting URL query strings.
url	module provides utilities for URL resolution and parsing.

Local Modules

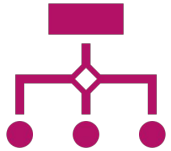
```
var logg = require('./3-1');

console.log(logg);

logg.log1('Welcome to FSD-2');
```

```
var url ='http://iiits.in/';

function log(message){
    //send http request
    console.log(message);
}

module.exports.log1 = log;
//exports.log1 = log;
module.exports.Linkurl =url;
```

Third-party Modules

Third-party modules are modules that are available online using the Node Package Manager(NPM)

These modules can be installed in the project folder or globally



Sample third-party modules

mongoose, express, angular, and react.

Node Package Manager

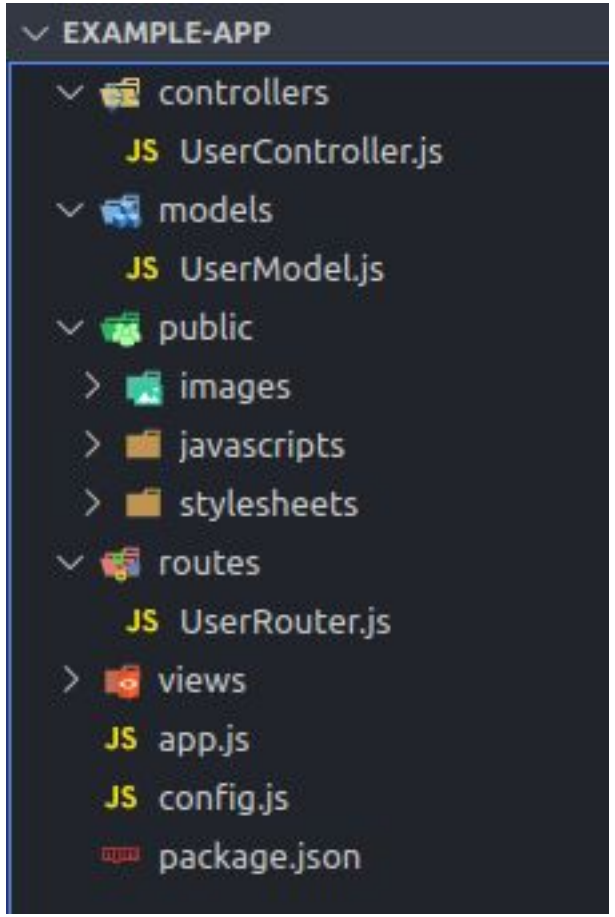
Online repository for open-source Node.js packages

Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application

The node community around the world creates useful modules and publishes them as packages in this repository

```
npm install npm –g
npm install <package name>
npm install express
```

Node.js application structure



app.js:- This file starts your web server. All your set up logic should be in this file.

Controllers:- This folder contains all the business logic of your application.

Models:- All the database models should go into the models folder.

Public:- All the public files such as images, javascript files, CSS files should go into this folder.

Routes:- All your routing-related logic should go into this folder

Views:- So this folder contains all your views i.e. HTML/ejs files. Drop this folder if you are building rest API's.

config.js:- This file should contain all your configuration e.g. PORT number, secrets, keys etc.

package.json

- Manifest file for your project
- Lists all the installed packages

```
{
  "name": "package.json-mastery",
  "version": "1.0.0",
  "description": "Mastery of the package.json file",
  "main": "index.js",
  "scripts": {
    "start": "node index",
    "dev": "nodemon index",
    "test": "jest"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/Easybuoy/package.json-mastery.git"
  },
  "keywords": [
     "javascript",
     "npm”
   ],
  "author": ”Author name",
  "homepage": "https://github.com/Easybuoy/package.json-mastery#readme",
  "engines": {
     "npm": "6.10.0",
     "node": "10.14.1"
   },
   "dependencies": {
     "bcryptjs": "^2.4.3",
     "cors": "^2.8.5",
     "dotenv": "^6.1.0",
     "express": "^4.16.4"
   },
   "devDependencies": {
     "eslint": "^4.19.1",
     "mocha": "^6.2.0",
     "nodemon": "^1.19.1"
   },
   "nyc": {
     "exclude": [
       "server/app.js",
       "server/config/",
       "server/build"
     ]
   }
}
```

package-lock.json

- File listing the full *dependency tree* of your project.


```
{
"requires": true,
"lockfileVersion": 1,
"dependencies": {
"abbrev": {
"version": "1.1.1",
"resolved": "https://registry.npmjs.org/abbrev/-/abbrev-1.1.1.tgz",
"integrity":
"sha512-nne9/IiQ/hzIhY6pdDnbBtz7DjPTKrY00P/zvPSm5pOFkl6xuGrGnXn/VtTNNfNtAfZ9/1RtehkszU9qcTii0Q==",
"dev": true
},
"accepts": {
"version": "1.3.5",
"resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.5.tgz",
"integrity": "sha1-63d99gEXI6OxTopywIBcjoZ0a9I=",
"dev": true,
"requires": {
"mime-types": "~2.1.18",
"negotiator": "0.6.1"
}
}
}
}
```

Event Emitters

- **Event-driven programming** is a programming paradigm in which the flow of the program is determined by events. An event-driven program performs actions in response to events. When an event occurs it triggers a callback function.

Event Emitter Methods

EventEmitter Methods	Description
emitter.addListener(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added.
emitter.on(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. It can also be called as an alias of emitter.addListener()
emitter.once(event, listener)	Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed.
emitter.removeListener(event, listener)	Removes a listener from the listener array for the specified event. Caution: changes array indices in the listener array behind the listener.
emitter.removeAllListeners([event])	Removes all listeners, or those of the specified event.
emitter.setMaxListeners(n)	By default EventEmitters will print a warning if more than 10 listeners are added for a particular event.
emitter.getMaxListeners()	Returns the current maximum listener value for the emitter which is either set by emitter.setMaxListeners(n) or defaults to EventEmitter.defaultMaxListeners.
emitter.listeners(event)	Returns a copy of the array of listeners for the specified event.
emitter.emit(event[, arg1][, arg2][, ...])	Raise the specified events with the supplied arguments.
emitter.listenerCount(type)	Returns the number of listeners listening to the type of event.

Local Modules(ES6)

```
import {fsd} from "./10.mjs";
import {wad} from "./10.mjs";
import * as fsdc from "./10.mjs";
fsdc.fsd();
fsdc.wad();
```

```
export function fsd(){
    console.log("FSD-1 is a part of FSD")
}

export function wad(){
    console.log("WAD is not a part of FSD")
}
```

Core Modules: **path**

- filename
- dirname

path.parse() returns elements of the path.

Properties:

- dir
- root
- base
- name
- ext

Core Modules: **os**

- os.freemem()
- os.getPriority([pid])
- os.homedir()
- os.hostname()
- os.loadavg()
- os.networkInterfaces()
- os.platform()
- os.release()
- os.setPriority([pid,]priority)
- os.tmpdir()
- os.totalmem()
- os.type()
- os.uptime()

……..

console.count()

Maintains an **internal counter** and outputs to **stdout** the number of times console.count()

Third party Module: **chalk**

chalk: color your font (https://www.npmjs.com/package/chalk)

chalk.blue()

chalk.red()

……..

Third party Module: **progress**

ProgressBar: Show the progress of a task (https://www.npmjs.com/package/progress)

Readline function

Define Readline function

```
const readline = require('readline').createInterface({
    input: process.stdin,
    output: process.stdout
})
```

Taking i/p and print

```
readline.question('Course Name ', course =>{
        console.log('Course name is ' +course)
        readline.close()
})
```

Interval

Set Interval

Clear Interval

```
const interval = setInterval(()=>{
    if(2==3){

        clearInterval(interval)
        return
    }
    console.log('FSD1');
}, 10)
```

Time Out

```
setTimeout(function(){
    console.log('FSD');
}, 1000)

console.log('Welocme to');
```

What will be the output?

Event Emitter

```
const evnte = require('events');
const emt = new evnte();

// a listener
emt.on('msg',()=>{
    console.log('Welcome to FSD-1')
})

//trigger the event
emt.emit('msg');
```

Event Emitter

```
const evnte = require('events');
const emt = new evnte();

emt.on('FSD Project Submission', ()=>{
    console.log('Please submit on time');
    setTimeout(()=>{
        console.log('Last day for submission');
    }, 8000)
    setTimeout(()=>{
        console.log('Its a gentle reminder');
    }, 3000)
})
console.log('Submission starts')
console.log('Submission starts from tomorrow')
emt.emit('FSD Project Submission');
```

What will be the output?

FS Modules

```
const fs = require('fs');
//
try {
    const fd = fs.openSync('file', 'r')
} catch(err){
    console.error(err)
}
```

```
fs.readFile('file2', (err,data)=>{
    if(err) throw err;
    console.log(data.toString())
})
```

```
const content = 'FSD2 is part FSD track'
fs.writeFile('file2', content, (err)=>{
    if(err) throw err;
    console.log('Done Successfully')
})
```

```
fs.appendFileSync('file1', 'FSD3 is also part of FSD Track', (err)=>{
    if(err) throw err;
    console.log('Done Successfully')
})
```

http Modules

```
const http = require('http');
const server = http.createServer();

server.listen(3001);
console.log('server is working')
```

Express

```
app.get('/', (req, res) => {
  res.send('Hello World!')
})
```

EJS(Embedded JavaScript templates)

```
<%- include('head'); %>
<%- include('navbar'); %>
<p>FSD1 is part of FSD
```

EJS(Embedded JavaScript templates)

```
<%- include('head'); %>
<%- include('navbar'); %>
<p>FSD1 is part of FSD
```

STATIC(Middleware)

```
app.use(express.static(path.join(__dirname, 'public')));
```

Body Parser(Middleware)

Define

```
const bparser = require('body-parser')
```

```
app.use(bparser.urlencoded({extended:false}));
```

Rendering

```
res.render('abt', {fname: req.body.firstname, lname: req.body.lastname});
```

In Memory Database

```
const sqlite3 = require('sqlite3')
```

sqlite3

Database connection

```
const db_name = path.join(__dirname, "data", "fsdapp.db");
const db = new sqlite3.Database(db_name, err =>{
    if(err){
        return console.log(err.message);
    }
    console.log("FSD Database Connected")
});
```

sqlite3

Create Table

```
const ousers = `CREATE TABLE users(
    uid INTEGER PRIMARY KEY AUTOINCREMENT,
    firstname VARCHAR(100) NOT NULL,
    lastname VARCHAR(100) NOT NULL

);`;
```

```
db.run(ousers, err=>{
    if(err){
        return console.log(err.message)
    }
    console.log("FSD User table created successfully")
})
```

sqlite3

Insert Data

```
const sinsert = `INSERT INTO users (uid, firstname, lastname) VALUES
(1, 'Himangshu', 'Sarma'),
(2, 'ABC', 'DEF')`;
db.run(sinsert, err =>{
    if(err){
        return console.log(err.message)
    }
    console.log("FSD user details entered")
})
```

sqlite3

Display In a page

```
app.get("/FSD", (req, res) =>{
    const sql = "SELECT * FROM users ORDER by uid";
    db.all(sql, (err, rows)=>{
        if (err){
            return console.log(err.message);

        }
        res.render("fdata", {model: rows});
        res.render()
    })

})
```

sqlite3

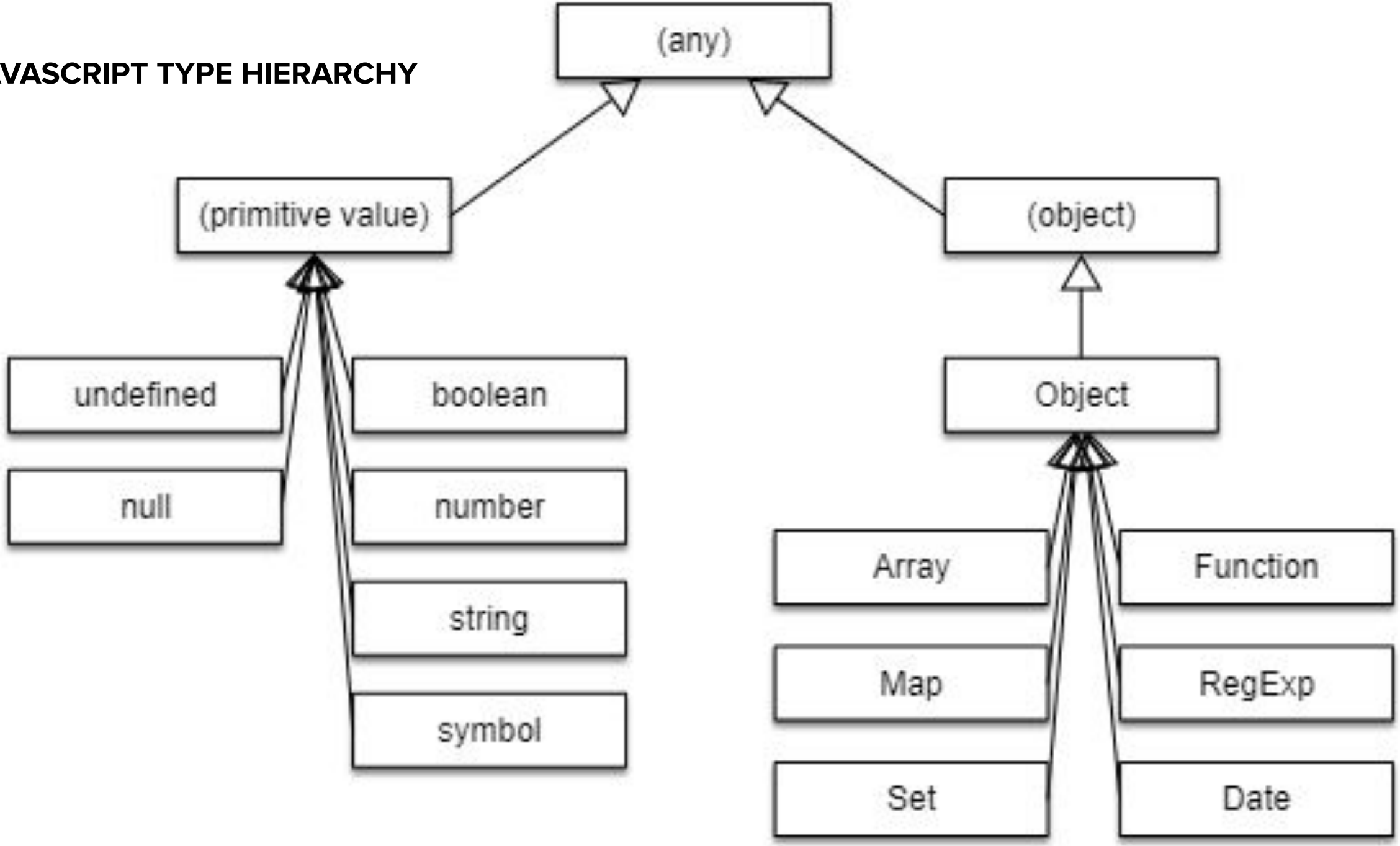
Insert data from a page

```
app.get("/Create", (req, res)=>{
    res.render("form1", {model:{}});
})


app.post("/Create", (req, res)=>{
    const sql = "INSERT INTO users (firstname, lastname)
VALUES (?, ?)"
    const book = [req.body.firstname, req.body.lastname];
    db.run(sql, book, err =>{
        if(err){
            console.log(err.message)
        }
        res.redirect("/FSD");
    })
})
```

JavaScript

The Language - Basics

JAVASCRIPT TYPE HIERARCHY



Our Checklist

- ❑ Let & Const
- ❑ Tour of types
  - ❑ Primitives
    - ❑ Boolean
    - ❑ String
    - ❑ Number
  - ❑ Reference types
    - ❑ Object
    - ❑ Arrays
    - ❑ Functions

JavaScript

Event Loop, Task Queue & Call Stack

Buzz words!

- Single threaded
- Non-blocking
- Asynchronous
- Concurrent
- Event-driven
- Dynamic
- Loosely-typed

Javascript & Friends

- We already saw JS doesn’t work alone and only works with friends (provided by the Hosted environment browser/node)
- But how ?
- How does it communicate with the friends?
  - Web APIs true! But what else
- How do they communicate back?
- What’s under the hood?

Let’s start with a recap of Call Stack (for C)

```c
#include<stdio.h>

int multiply(int n, int m){
  int res = n * m;
  return res;
}

int square(int n){
  int res = multiply(n,n);
  return res;
}

void printsquare(int n)
{
  int res = square(n);
  printf("%d",res);
}

int main() {
  int n=2;
  printsquare(n);
```

Edit this code

Stack

main
- n: int 2

printsquare
- n: int 2
- res: int ?

square
- n: int 2
- res: int ?

multiply
- n: int ?
- m: int ?
- res: int ?

Let’s start with a recap of Call Stack
(Not so different for Javascript)

```
1  function multiply(n,m){
2    let res = n * m;
3    return res;
4  }
5  function square(n){
6    let res = multiply(n,n);
7    return res;
8  }
9  function printsquare(n){
10   let res = square(n)
11   console.log(res)
12 }
13 const num=2;
14 printsquare(num)
```

Microtask Queue

Call Stack ABOUT

multiply

square

printsq

STEP

What does single threaded means?

- At any given time there can be ONLY ONE STACK
- For multithreaded languages, each thread gets its very own stack
  - ‘Main’ is the first thread
- Javascript uses the ‘event loop’ and the ‘task queue(s)’ to achieve the communication with the friends
- Lets see a simple example

Lets see a simple example

```
setTimeout(function a() {console.log('a')}, 1000);

setTimeout(function b() {console.log('b')}, 500);

setTimeout(function c() {console.log('c')}, 0);

function d() {console.log('d')}

d();
```

The `setTimeout()` method executes a block of code after the specified time. The method executes the code only once.

The commonly used syntax of JavaScript setTimeout is:

```
setTimeout(function, milliseconds);
```

Its parameters are:

- function - a function containing a block of code
- milliseconds - the time after which the function is executed

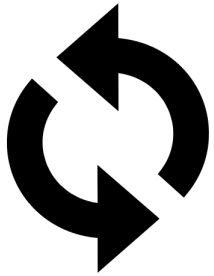
Call Stack

Console

```
subuk@Subus-Mac-mini fullstack % node javascript.js
```

Timer friend
(Node timer thread)

Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

Call Stack

Console

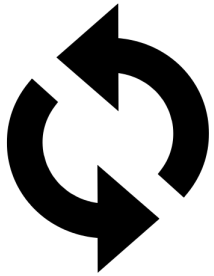
```
subuk@Subus-Mac-mini fullstack % node javascript.js
```

Timer friend
(Node timer thread)

SetTimeout

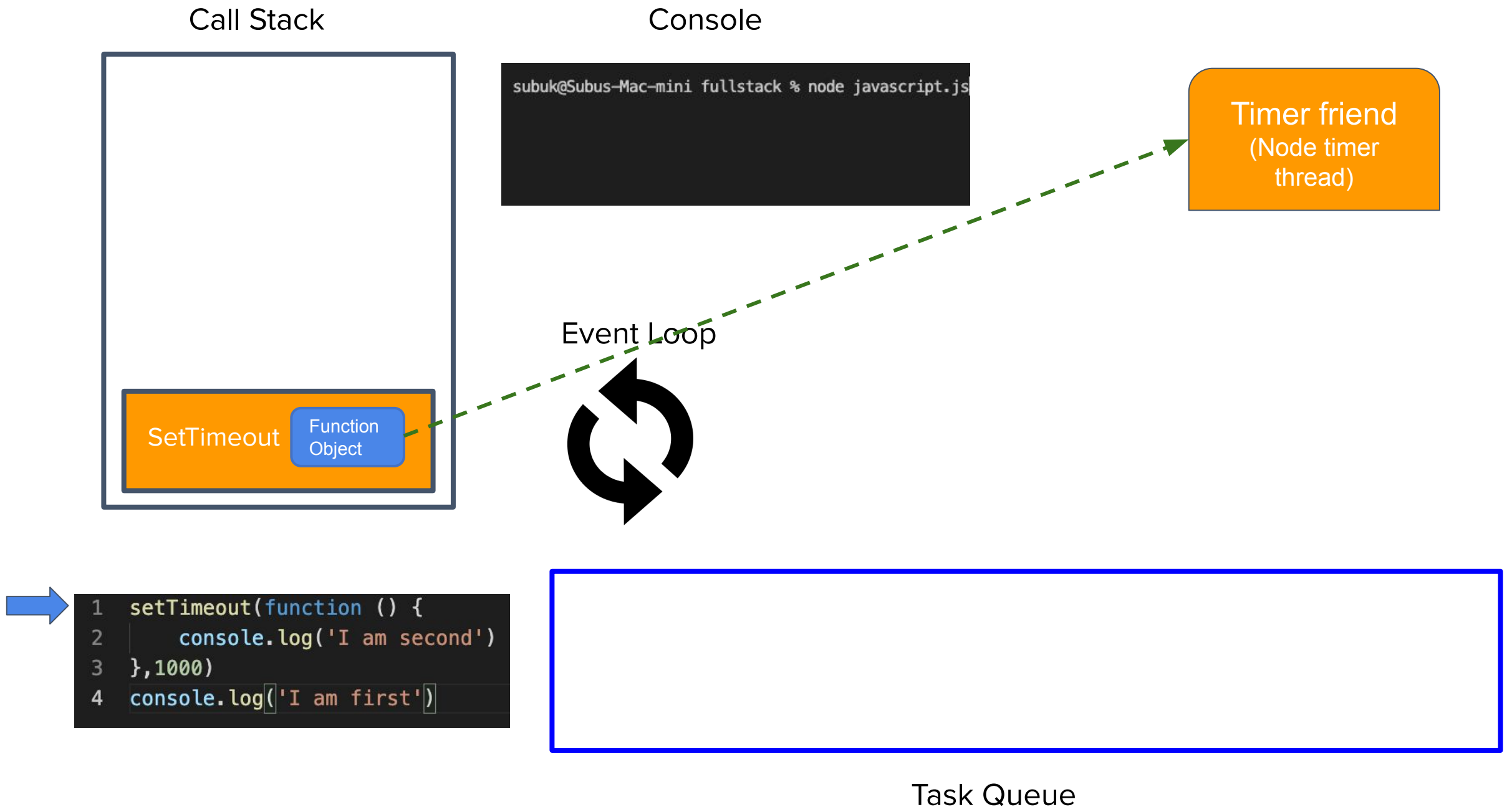
Function Object

Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue


Call Stack
Console
subuk@Subus-Mac-mini fullstack % node javascript.js
Timer friend
(Node timer thread)
Event Loop
SetTimeout
Function Object
setTimeout(function () {
console.log('I am second')
},1000)
console.log('I am first')
Task Queue

Call Stack

Console

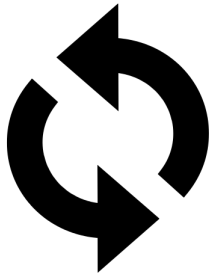
```
subuk@Subus-Mac-mini fullstack % node javascript.js
```

Function Object

Timer friend
(Node timer thread)

1000

Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

Call Stack

console.log

String Object

Console

```
subuk@Subus-Mac-mini fullstack % node javascript.js
```

Function Object

Timer friend
(Node timer thread)

200

Event Loop

```
1  setTimeout(function () {
2      console.log('I am second')
3  },1000)
4  console.log('I am first')
```

Task Queue

Call Stack

Console

```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
```

Function Object

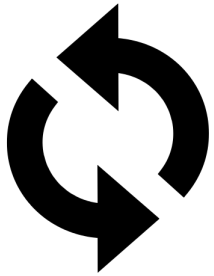
Timer friend
(Node timer thread)

100

console.log

String Object

Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

Call Stack

Console

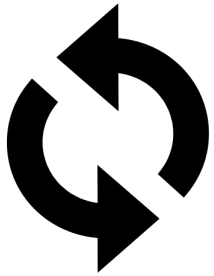
```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
```

Function Object

Timer friend
(Node timer thread)

0

Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
```

Function Object

Timer friend
(Node timer thread)

0

Event Loop

```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

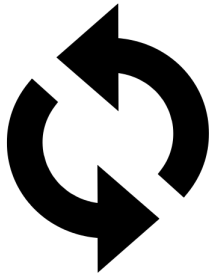
Call Stack

Console

```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
```

Timer friend
(Node timer thread)

Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

Call Stack

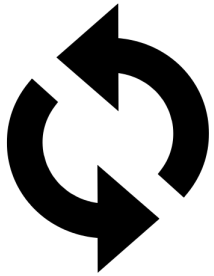
Anonymous function

Console

```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
```

Timer friend
(Node timer thread)

Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

Call Stack

console.log

String Object

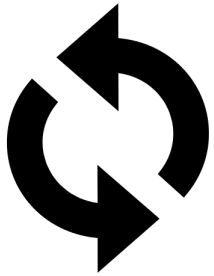
Anonymous function

Console

```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
```

Timer friend
(Node timer thread)

Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

Call Stack

Console

```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
I am second
subuk@Subus-Mac-mini fullstack %
```

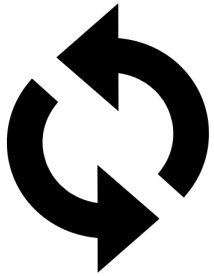
Timer friend
(Node timer thread)

console.log

String Object

Anonymous function

Event Loop



```
setTimeout(function () {
    console.log('I am second')
},1000)
console.log('I am first')
```

Task Queue

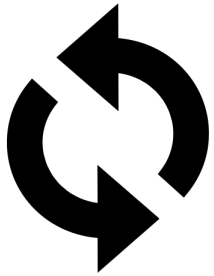
Call Stack

Console

```
subuk@Subus-Mac-mini fullstack % node javascript.js
I am first
I am second
subuk@Subus-Mac-mini fullstack %
```

Timer friend
(Node timer thread)

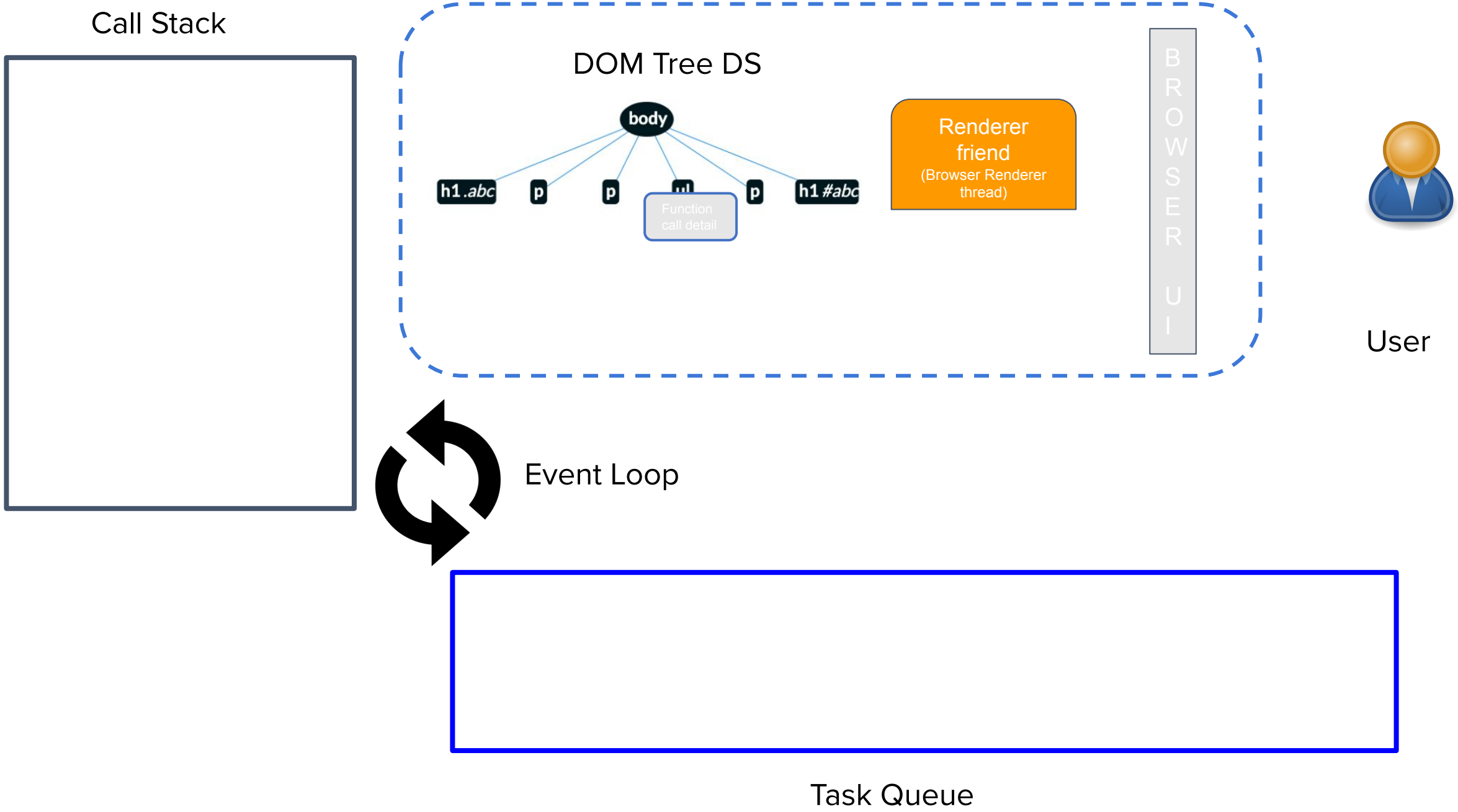
Event Loop

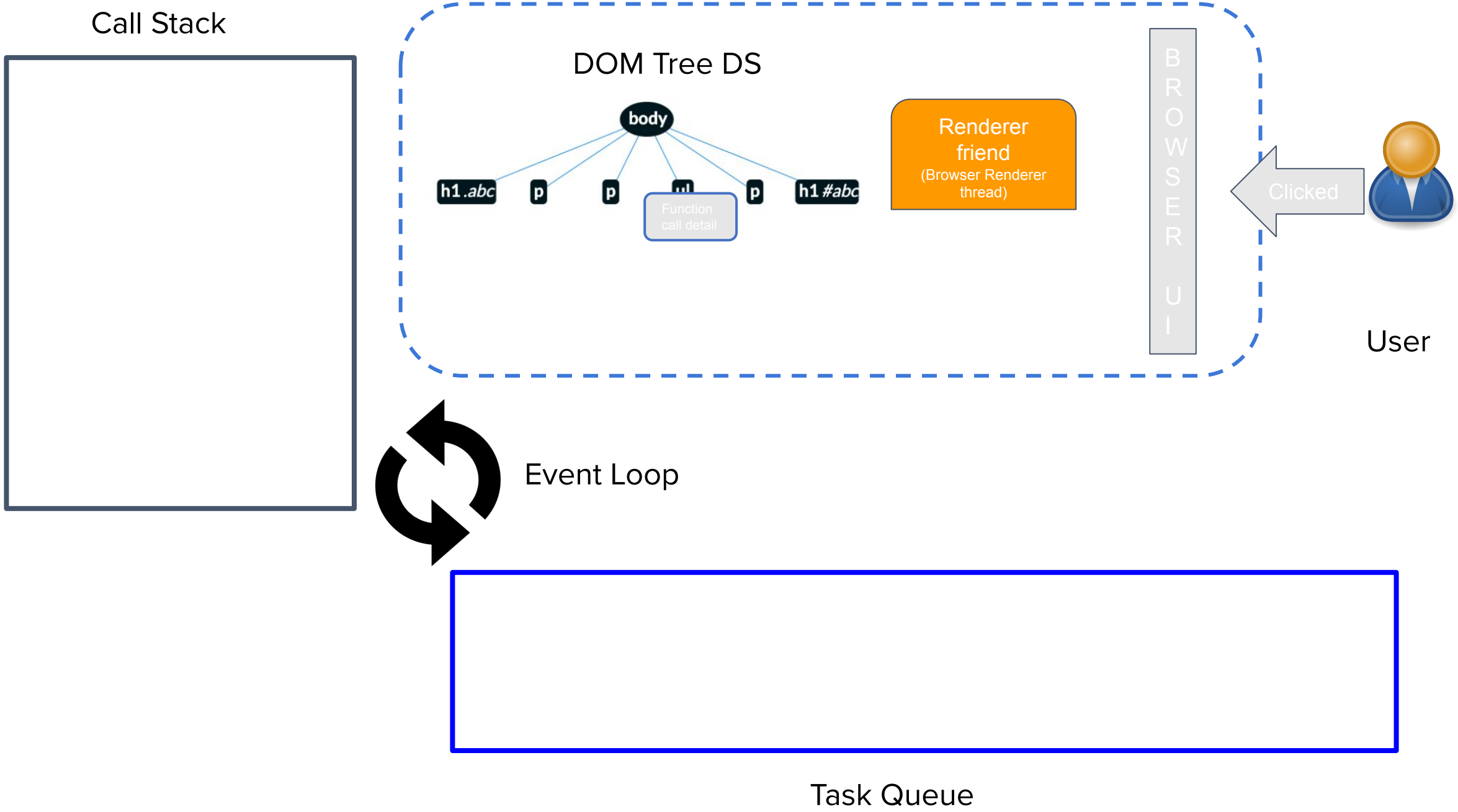


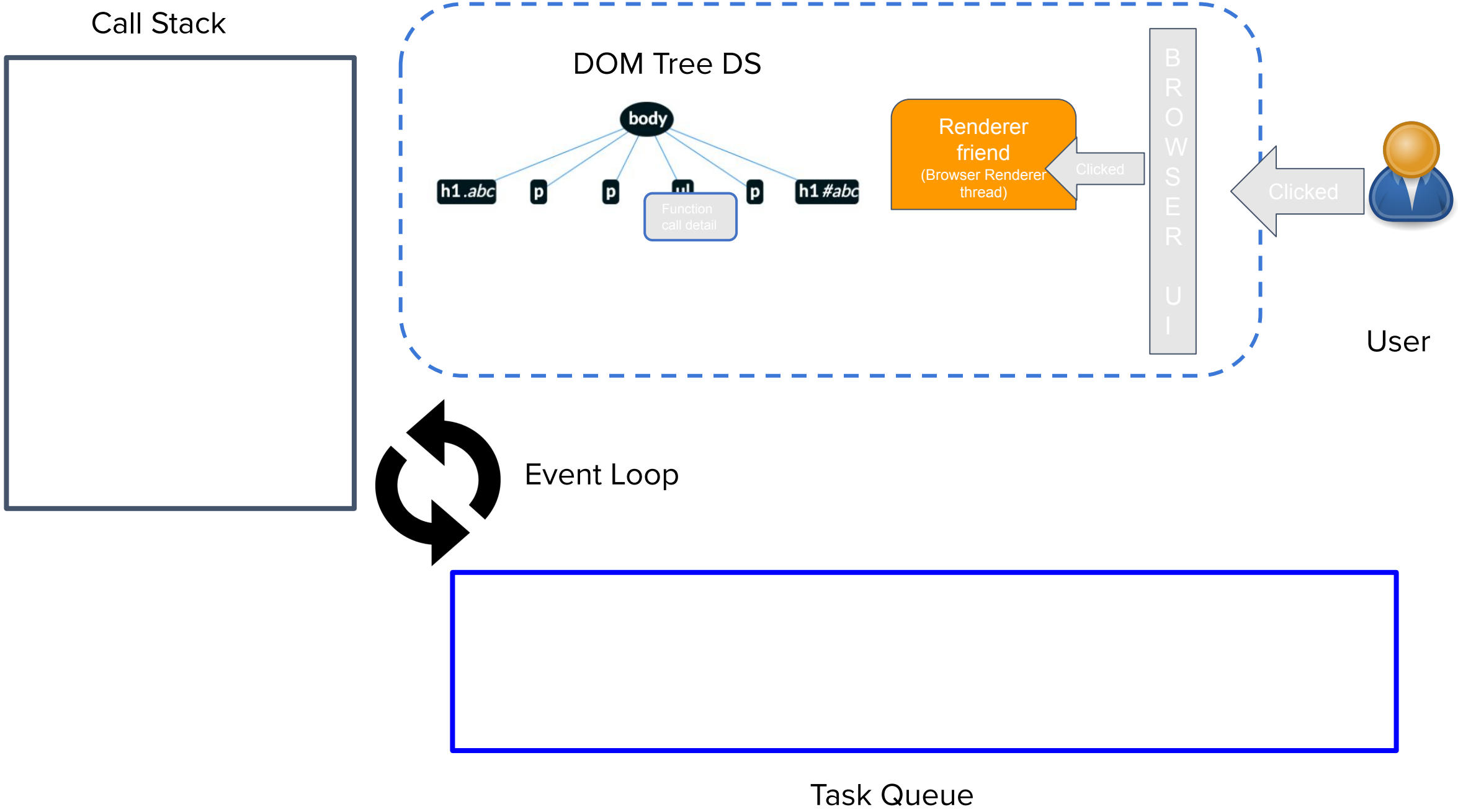
```
1  setTimeout(function () {
2      console.log('I am second')
3  },1000)
4  console.log('I am first')
```

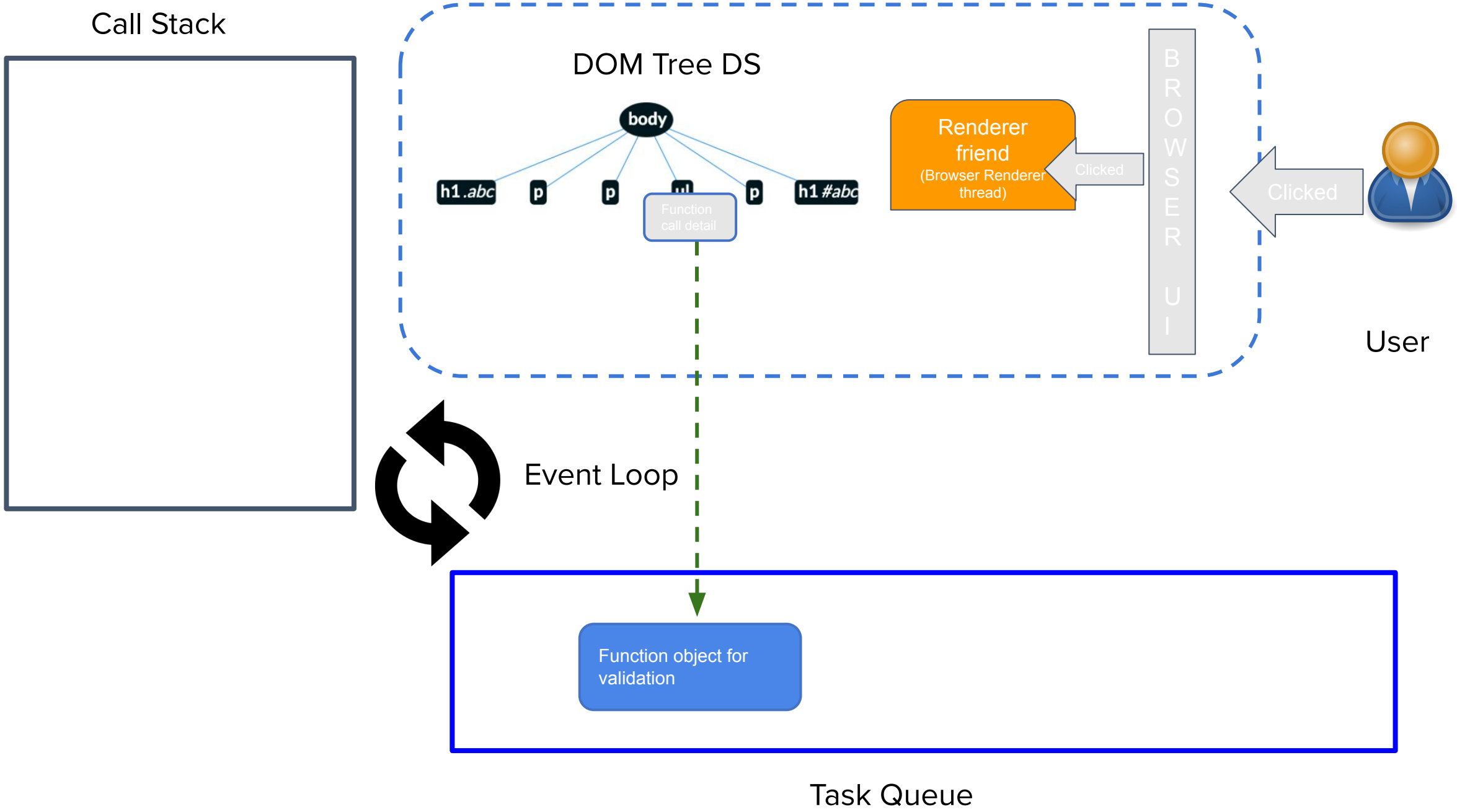
Task Queue

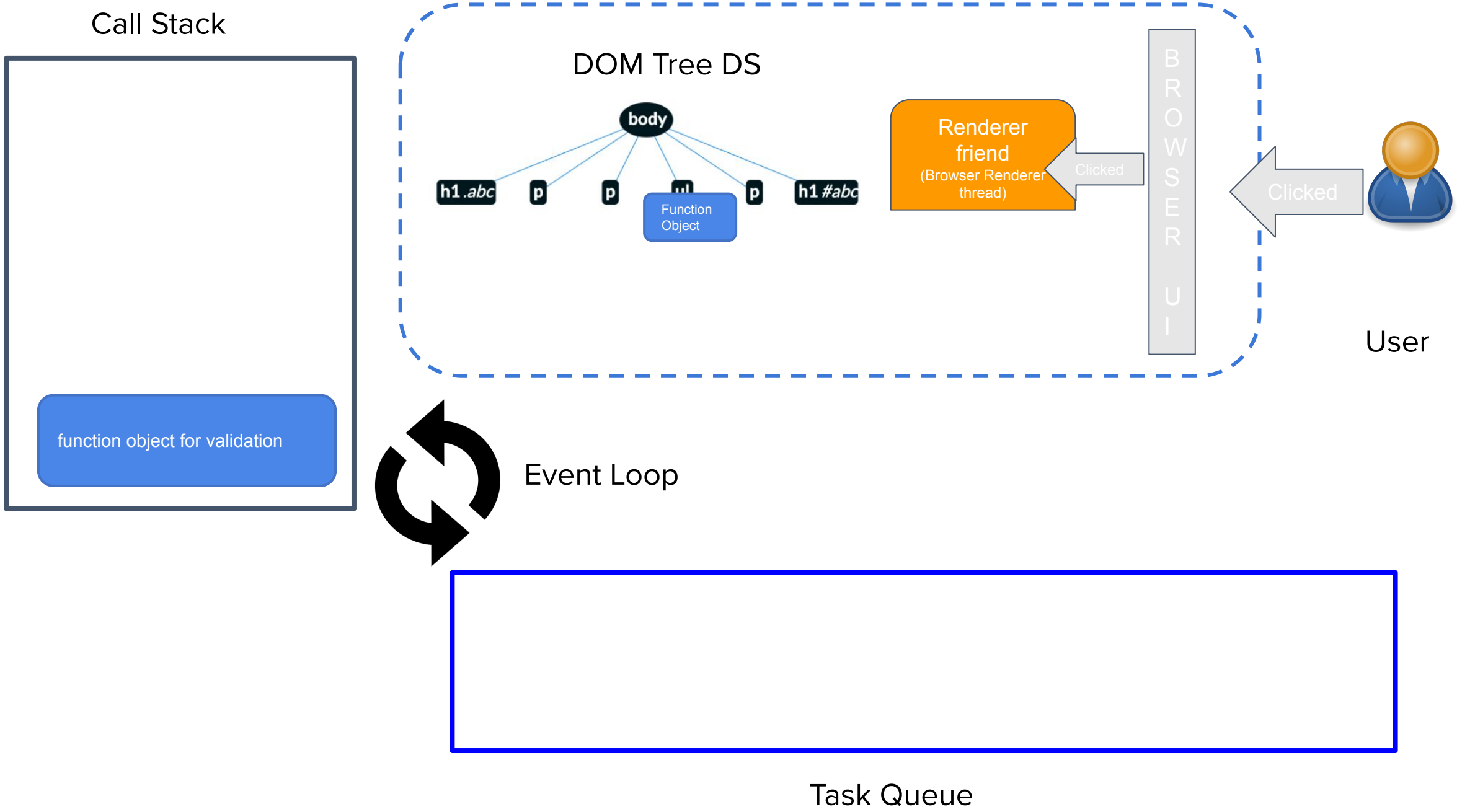
What about the browser?


Call Stack
DOM Tree DS
body
h1 .abc
p
p
p
h1 #abc
Function call detail
Renderer friend
(Browser Renderer thread)
BROWSER UI
User
Event Loop
Task Queue


Call Stack
DOM Tree DS
body
h1 .abc
p
p
p
h1 #abc
Function call detail
Renderer friend
(Browser Renderer thread)
BROWSER UI
Clicked
User
Event Loop
Task Queue


Call Stack
DOM Tree DS
body
h1 .abc
p
p
ul
p
h1 #abc
Function call detail
Renderer friend
(Browser Renderer thread)
Clicked
BROWSER UI
Clicked
User
Event Loop
Task Queue


Call Stack
DOM Tree DS
body
h1 .abc
p
p
p
h1 #abc
Function call detail
Renderer friend
(Browser Renderer thread)
Clicked
BROWSER UI
Clicked
User
Event Loop
Function object for validation
Task Queue


Call Stack
function object for validation
DOM Tree DS
body
h1 .abc
p
p
Function Object
p
h1 #abc
Renderer friend
(Browser Renderer thread)
Clicked
BROWSER UI
Clicked
User
Event Loop
Task Queue

Summary

- The infrastructure around JS engines + Hosted environment allows for an effortless asynchronous solution.
- The components involve
  - Call Stack
  - Event Loop
  - Task Queue

Primitive Types

```
let x= 5
let y= 5.0
let z= ‘FSD2’
let a= null
let b
let c = true
```

JavaScript is *Object-Based*

- Everything that JavaScript manipulates, it treats as an object – e.g. a window or a button
- An object has properties – e.g. a window has size, position, status, etc.
- Objects are modified with methods that are associated with that object – e.g. a resize a window with resizeTo(150, 200)

not a true object-oriented language like C++ or Java

Reference Type

- Object
- Array

Object

```
let person = {
  firstName : "John",
  lastName  : "Doe",
  age     : 50,
  eyeColor  : "blue"
};
```

Object

```
let person = {
  firstName : "John",
  lastName  : "Doe",
  age     : 50,
  eyeColor  : "blue"
};
 console.log(person)
```

Object

```
let person = {
  firstName : "John",
  lastName  : "Doe",
  age     : 50,
  eyeColor  : "blue"
};
 console.log(person)
console.log(person.firstName)
```

Object

```
let person = {
  firstName : "John",
  lastName  : "Doe",
  age     : 50,
  eyeColor  : "blue"
};
 console.log(person)
console.log(person.firstName)
console.log(person[‘firstName’])
```

```
let fsd2 = {
    sub: "FSD 2",
    faculty: "HS"

}

console.log(fsd2)
console.log(fsd2['faculty'])
```

```
let fsd2 = {
    sub: "FSD 2",
    faculty: "HS",
    section: {
        secA: 'SK',
        secB: 'BH'
    }
}

console.log(fsd2)
console.log(fsd2['faculty'])
```

```
let fsd2 = {
    sub: "FSD 2",
    faculty: "HS",
    section: {
        secA: 'SK',
        secB: 'BH'
    },
    printd: function(){
        return this.sub+' taught by ' +this.faculty
    }
}

console.log(fsd2)
console.log(fsd2['faculty'])
console.log(fsd2.printd())
```

```
let fsd2 = {
    sub: "FSD 2",
    faculty: "HS",
    section: {
        secA: 'SK',
        secB: 'BH'
    },
    printd: function(){
        return this.sub+' taught by ' +this.faculty
    }
}

console.log(fsd2)
console.log(fsd2['faculty'])
console.log(fsd2.printd())
```

```
let check = 'printd'
console.log(fsd2[check]())
```

```
function sum(x,y){
    let res = x+y;
    return res;
}
```

```
function sum(x,y){
    let res = x+y;
    return res;
}
```

```
let sum1 = function sum(x,y){
    let res = x+y;
    return res;
}
```

```
function sum(x,y){
    let res = x+y;
    return res;
}
```

```
let sum1 = function sum(x,y){
    let res = x+y;
    return res;
}
```

```
let sum2 =(x,y) => x+y
```

Arrays

```
let cars = ["Saab", "Volvo", "BMW"];
let ncar = cars
ncar[0] = ‘Mahindra’

console.log(cars)
```

Objects using Constructor

```
function constructor(arg1, arg2) {
  this.a = arg1;
  this.b = arg2;
  this.display = function() {
    console.log(`Running Text`);
  };
}
```

Prototype

```
function fsd() {
    this.cname = 'FSD-1';
    this.dept = 'CSE';
}

let fsdObj1 = new fsd();
fsdObj1.cre = 4;
alert(fsdObj1.cre);

let fsdObj2 = new fsd();
alert(fsdObj2.cre);
```

Prototype

```
function fsd() {
    this.cname = 'FSD-1';
    this.dept = 'CSE';
}

fsd.prototype.cre = 4;

let fsdObj1 = new fsd();
alert(fsdObj1.cre);

let fsdObj2 = new fsd();
alert(fsdObj2.cre);
```

Prototype

```
function fsd(subName, cName){
    this.name = subName
    this.cnam = cName
}

//include new prototype
fsd.prototype.chName = function nfs(){
    return this.cnam;
}

let c1 = new fsd("FSD1", "CSE")
console.log(c1)
console.log(c1.chName())
```

Class

```
class CLASS {
    constructor(arg1, arg2) {
        this.a = arg1;
        this.b = arg2;
    }
```

Class

```
class CLASS {
    constructor(arg1, arg2) {
        this.a = arg1;
        this.b = arg2;
    }

    //  Methods
    msg(){
        return `Text running`;
    }
    // Static method
    static add(a, b){
        return a + b;
    }
}
```

Class

```
class CLASS {
    constructor(arg1, arg2) {
        this.a = arg1;
        this.b = arg2;
    }

    //  Methods
    msg(){
        return `Text running`;
    }
    // Static method
    static add(a, b){
        return a + b;
    }
}
```

```
obj1 = new CLASS("value1",
"value2");
console.log(obj1.msg());
console.log(CLASS.add(34, 5))
```

Inheritance

```
class CSE extends dept{
    constructor(subject, coursecode, credit, elective, track){
        super(subject, coursecode, credit);
        this.ele = elective;
        this.tr = track;
    }
}
```

JavaScript Callbacks

```
function Calc(num1, num2) {
   return sum = num1 + num2;
 }

 let a = Calc(5,5)
 console.log("Output is " +a)
```

JavaScript Callbacks

A callback is a function passed as an argument to another function.

JavaScript Asynchronous

Used to execute callback function on specific time

JavaScript Asynchronous

```
setTimeout(myFunction, 3000);
function myFunction() {
   console.log("FSD 2 is running");
}
```

JavaScript Promises

Two arguments

- Resolve
- Reject

```
let promise = new Promise(function(resolve, reject) {
  );
```

JavaScript Promises

Three States

- Pending
- Fulfilled
- Rejected

JavaScript Promises

Skeleton

```
let promise = new Promise(function(resolve, reject) {
   resolve();
   reject();
  );
promise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

JavaScript Promises

```
let promise = new Promise(function(resolve, reject) {
    const x = 1;
    const y = "1";
    if(x == y) {
      resolve();
    } else {
      reject();
    } });

  promise.then(function(){
  console.log('Both grades are same');}).catch(function () {
  console.log('Both grades are not same');});
```

JavaScript Async/ Await

```
async function FSD2() {
    const x = 1;
    const y = "1";
    if(x == y) {
        await console.log("Correct");
    } else {
      await console.log("Error");
    } };

    console.log("MSG 2");
    console.log(FSD2());
    console.log("MSG 3");
```

AJAX(Asynchronous JavaScript And XML)

- Asynchronous JavaScript And XML
- Update web pages asynchronously
- Update a web page without reloading the whole page
- XMLHttpRequest: browser built-in object (Request Data)
- Print/Display: JavaScript and HTML DOM

AJAX(Asynchronous JavaScript And XML)

```
// Create an XMLHttpRequest object
const DisplayContent = new XMLHttpRequest();

// Define a callback function
DisplayContent.onload = function() {
 // Here you can use the Data
}

// Send a request
DisplayContent.open("GET", "fsd.txt");
DisplayContent.send();
```

XMLHttpRequest Methods

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(method, url, async, user, psw)	Specifies the request method: the request type GET or POST url: the file location async: true (asynchronous) or false (synchronous) user: optional user name psw: optional password
send()	Sends the request to the server Used for GET requests
send(string)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

XMLHttpRequest Object Properties

Property	Description
onload	Defines a function to be called when the request is received (loaded)
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

onreadystatechange Properties

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")