Who am I?

Humera Tariq

PhD, MS, MCS (Computer Science), B.E (Electrical)

Postdoc (Medical Image Processing, Deep Neural Networks)

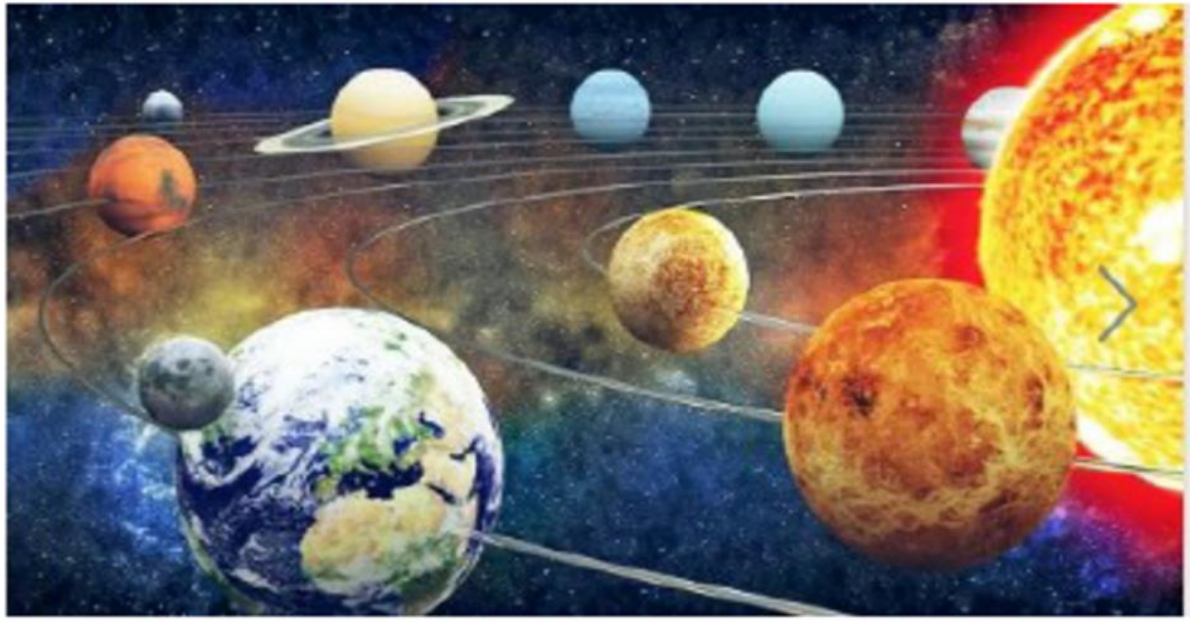
Email: humera@uok.edu.pk

Web: https://humera.pk/

Discord: https://discord.gg/xeJ68vh9

Starting in the name of Allah,

the most beneficial,
the most merciful.



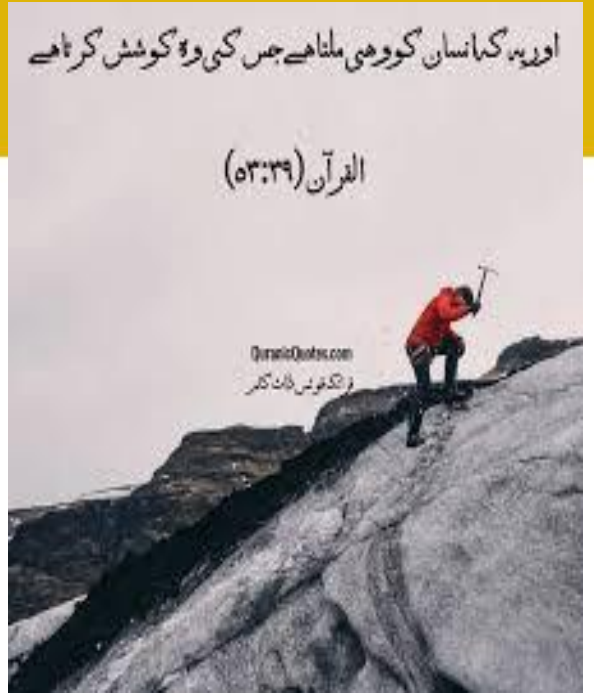
اَمۡ لِلۡاِنۡسَانِ مَا تَمَنّٰى ﴿۲۴﴾ فَلِلّٰهِ الۡاٰخِرَةُ وَالۡاُوۡلٰى ﴿۲۵﴾

کیا انسان کو ہر وہ چیز حاصل ہے جس کی اس نے تمنا کی؟ پس اللہ ہی کے لیے ہے آخرت اور دنیا

UNIVERSITY OF KARACHI

Surah An-Najm Chapter 53 Verse 39

And there is not for man except that [good] for which he strives.



UNIVERSITY OF KARACHI

Week 04

Internet Application Development

Department of Computer Science (DCS/UBIT)
University of Karachi
January 2025

Today's Agenda

- ✓ Java script (JS) run-time
- ✓ Hoisting, Scoping
- ✓ Synchronous vs Asynchronous
- ✓ JS what are you?
- ✓ V8 Engine do you know event loop, call back, http request, apis e.g. setTimeOut?

UNIVERSITY OF KARACHI

JavaScript runtime environment

Today's JS runtime environments are highly efficient and a number of them exist!

•V8 is _____ JavaScript engine (used in Chrome and other browsers).

•SpiderMonkey is _____ engine and used in Firefox.

•Chakra is _____ runtime engine. It was originally used in browser. In December 2018, _____ decided to adopt Chromium (Google's open-source browser project) as its JavaScript runtime in the browser. Chakra is still powering Windows applications that are written in HTML/CSS and JavaScript.

While the browser is the most obvious usage scenario for JavaScript runtime environments, they are also used in other areas such as _____ . Most importantly for us: the Node.js platform we cover soon is built on top of _____ .

UNIVERSITY OF KARACHI

JS engines interpret *or* compile ?

Today's JavaScript engines _____ by employing so-called **just-in-time (JIT) compilation**. This means that JavaScript code that is run repeatedly such as often-called functions is eventually _____ and no longer interpreted. This article by Lin Clark explains this in more detail for those that want to know more.

UNIVERSITY OF KARACHI

What TypeScript offers ??

It is worthwhile to know that many languages compile into JavaScript. Three of the most well-known languages are _____ , _____ and _____ ; all three fill one or more gaps of the original JavaScript language. Here is one example of what TypeScript offers: JavaScript is a _____ language , this means that you have no way of enforcing a certain **type** on a variable. Instead, a variable can hold any type, a String, a Number, an Array … but of course often you *know* what you want the type to be (for instance function parameters). It is useful to provide this knowledge upfront. TypeScript allows you to do that, by enabling _____(static/dynamic) type checking .

UNIVERSITY OF KARACHI

JavaScript

.....

- ✓ Hoisting…….. f() , variable
- ✓ Scoping……

How to avoid hoisting problems!

There are a few things you can do to avoid hoisting problems:

- ✓ Always declare your variables at the top of their scope. This will make your code more readable and easier to maintain.
- ✓ Use the let or const keywords to declare your variables instead of the var keyword. let and const variables are not hoisted, so they can only be used after they are declared.
- ✓ Use JavaScript's strict mode. Strict mode prevents you from using undeclared variables, which can help to catch hoisting problems.

Option 1: You may either believe that the JS runtime does not care about something is declared and the output will be 6 - 7

Option 2: The JavaScript runtime does indeed care and the output will be a TypeError: six is not a function

What will be the output?

```
var x = six();              //CASE 1: function declaration

function six() { return 6; }

var y = seven();            //CASE 2: function expression

var seven = function () { return 7; };

console.log(x + " - " + y);
```

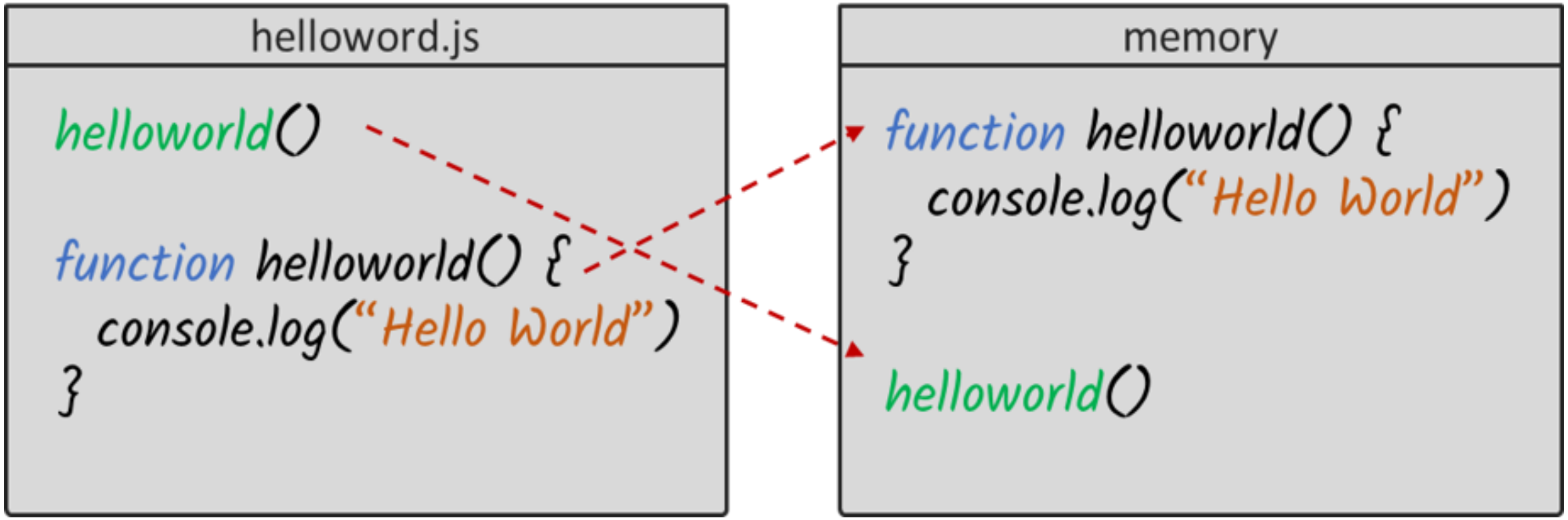
Neither of these two options are correct however, the output will be a TypeError: seven is not a function. This means that while var x = six(); works (i.e., we can call six() before declaring it), var y = seven(); does not.

function six(){...} is a **function declaration** and is defined as soon as its surrounding function or script is executed due to the **hoisting principle** declarations are processed before any code is executed.

In our example , the JavaScript runtime *hoists* the declaration of six to the top; it is processed before the remaining code is executed. **Expressions are not hoisted.**

var seven = function(){...} is a **function expression** and is only defined when that line of code is reached.

Function call / invocation before function declaration and definition



method is placed at the beginning of the code

This is not only the case for functions, variable declarations are also hoisted.

```
function f() {
    x = 5;
    y = 3;
 }

f();

console.log(x); //5
console.log(y); //3
```

Variables x and y have global scope as they are not prefixed by var or let or const so output will be 5 and 3.

JavaScript Hoisting is a process where if you declare a variable inside a scope, then it will move those declarations at the top.

```
function f() {
        a = 5;
        b = 3;
        var a, b; //variable declaration
 }
f();
 console.log(a); //ReferenceError: a is not defined
console.log(b);
```

Var a and var b declation are hoisted to the top of function so we ended up with error

helloword.js

```
var a = 10

function print() {
  console.log(a)
  var a = 20
  console.log(a)
}
print()
```

output

output

```
> undefined
> 20
```

Raise given to var a inside function print() due to hoisting. Note assignment is not raised.

helloword.js

```
var a = 10

function print() {
  console.log(a)
  var a = 20
  console.log(a)
}
print()
```

output

memory

```
var a = 10

function print() {
  var a
  console.log(a)
  a = 20
  console.log(a)
}
print()
```

JavaScript

.....

✓ Scoping.....

Scoping

Scoping is the **context in which values and expressions are** _____ . In contrast to other languages, JavaScript has very few scopes: _____ , _____and _____ .

A _____ is used to group a number of statements together with a _____(Hint: syntax) .

UNIVERSITY OF KARACHI

Scoping : local, global , block

The difference between let and const is that _____ does not allow the reassignment or redeclaration of a variable. The originally assigned element though **can** change.

here/how	Scope
var declared within a function	?
var declared outside of a function	?
let (ES6)	?
const (ES6)	?
variable declaration without var/let/const	?

UNIVERSITY OF KARACHI

OK or not-OK ? Why or Why Not ? Write exact error

Code	COMMENT or ERROR
let a = [1, 2, 3]; const b = [4, 5, 6];	//array with 3 numbers //array with 3 numbers
a = "hello world";	?
b = "hello world";	?
b[0] = -1;	?
console.log(b);	?

UNIVERSITY OF KARACHI

What is the output ?

Before **ES6** there was no **block scope**, means there is no _____ (refer to previous scope table) we only had two scopes available: local and global. Having only two scopes available resulted in code behavior that does not always seem intuitive. Let's look at one popular example: imagine we want to print out the numbers 1 to 10. This is easy to achieve in JavaScript:

Synchronous Loop or Asynchronous loop ?

```
for (var i = 1; i <= 10; i++) {
    console.log(i);
}
```

Output

?

UNIVERSITY OF KARACHI

What is the output ?

Let's now imagine that the print outs should happen each after a delay of one second. Once you know that setTimeout(fn, delay) initiates a timer that calls the specified function fn (below: an **anonymous function**) after a delay (specified in milliseconds) you might expect the following piece of code 👇 to print out the numbers 1 to 10 with each number appearing after roughly a second:

Synchronous Loop or Asynchronous loop ?

```
for (var i = 1; i <= 10; i++) { setTimeout(function () {
                                  console.log(i);
                                }, 1000);
}
```

Output

??

What are the two issues ???

In the code above, var i has _____ scope, but we actually need it to be of _____ **scope** such that every function has its own _____ copy of it.

- printing _____ instead of _____.
- Waiting for _____ *between print outs* one by one.

UNIVERSITY OF KARACHI

https://www.youtube.com/watch?v=8aGhZQkoFbQ

What the heck is the event loop anyway? | Philip Roberts | JSConf EU

Java Script.....

- ✓ Scoping ..... Event loop
- ✓ Hoisting
- ✓ this
- ✓ Closure

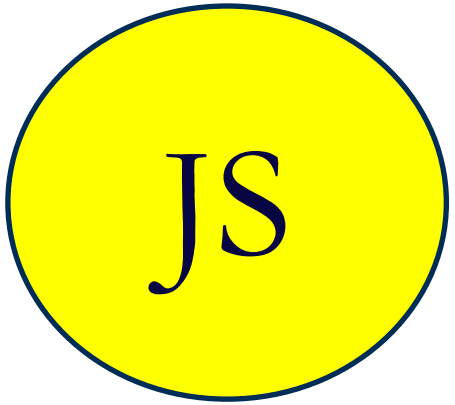
What are you ?

JS

I am a single threaded non-blocking asynchrounous concurrent language

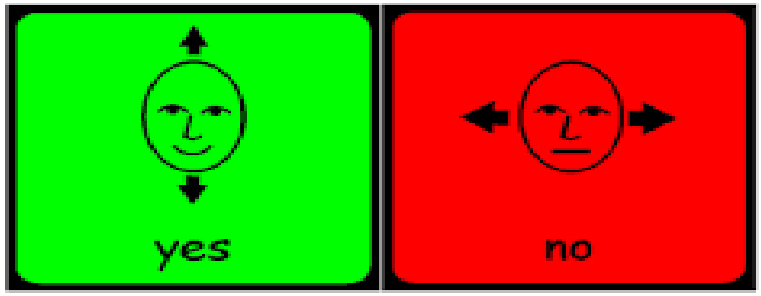
UNIVERSITY OF KARACHI

What are you ?



I have a call stack, event loop, a call back queue and some api stuff.

Hey Hey V8 !



Do you have a call back, event loop, and a call back queue ?

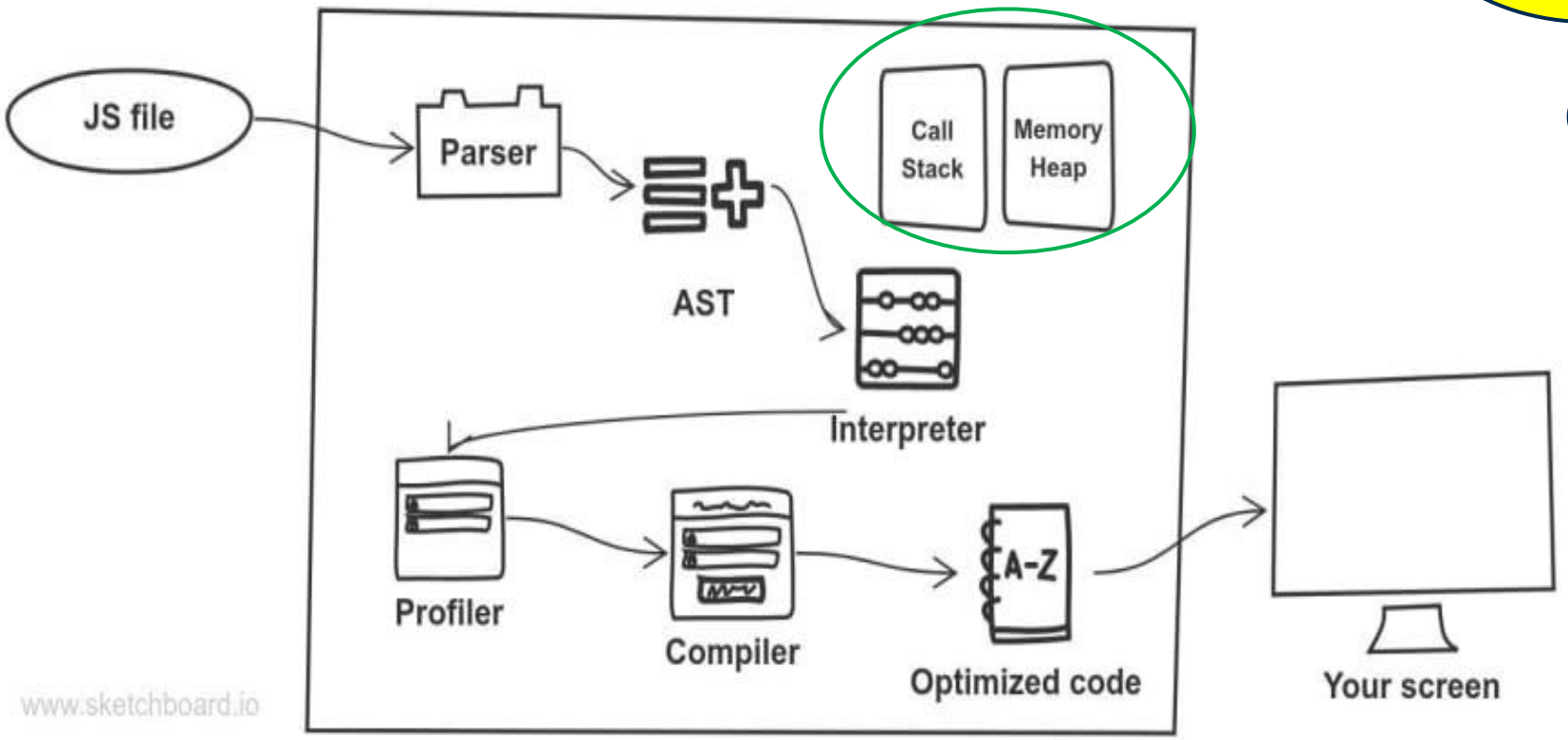
Do you have DOM , http request, setTimeout?

UNIVERSITY OF KARACHI

Chrome V8 Engine
Javascript Runtime for Node.js

I don't know call back, event loop, and a call back queue ?

Javascript Engine



Let's Recall Stack and Heap before we talk about call stack !

STACK

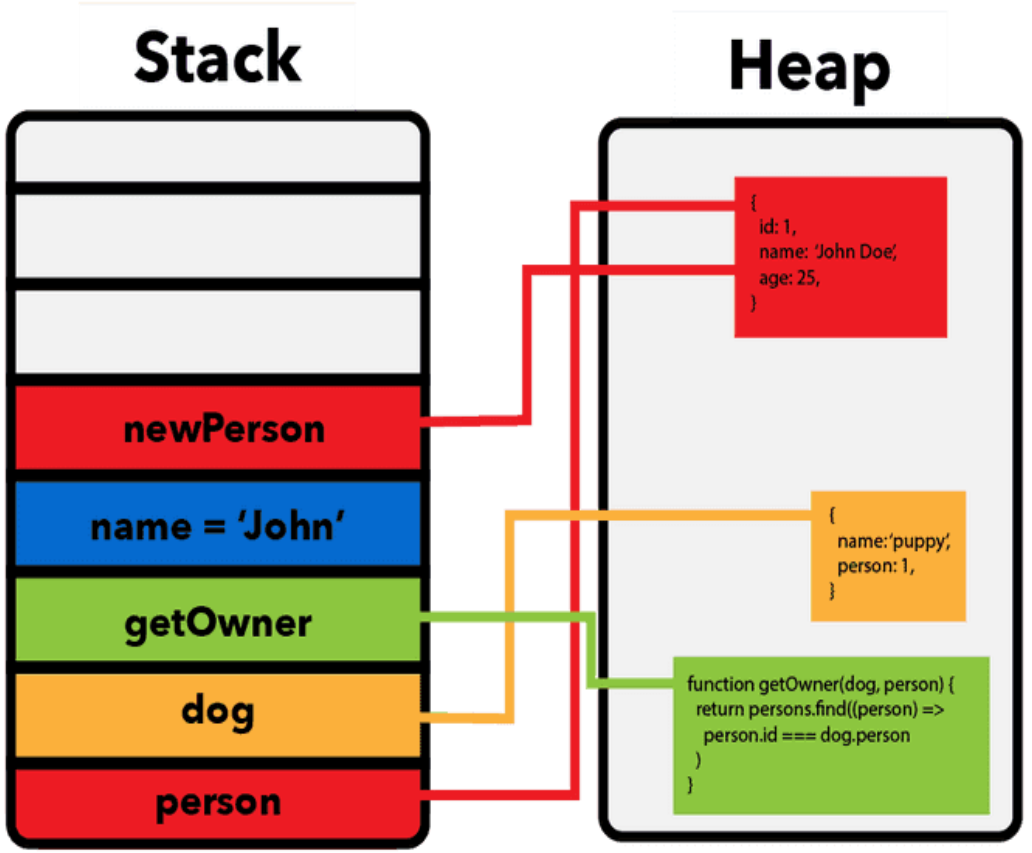
A stack is a data structure that JavaScript uses to store *static* data. Static data is data where the engine knows the size at _____ . In JavaScript, this includes **primitive values** (*strings*, *numbers*, *booleans*, *undefined*, and *null*) and **references**, which point to objects and functions.

HEAP

The heap is a different space for storing data where JavaScript stores _____ and _____ . Unlike the stack, the engine **doesn't allocate a fixed amount of memory for these objects.** Instead, more space will be allocated as needed.

All variables first point to the _____ . In case it's a non-primitive value, the stack contains a reference to the object in the _____ . The memory of the heap is not ordered in any particular way, which is why we need to keep a reference to it in the stack. You can think of references as addresses and the objects in the heap as houses that these addresses belong to.

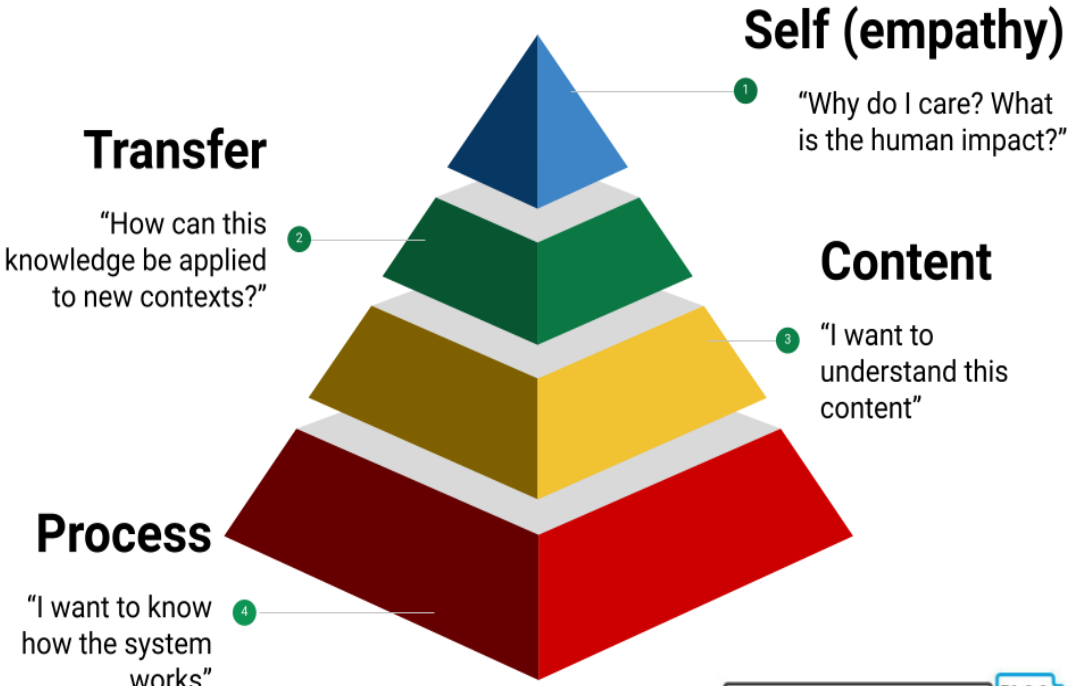
```
const person = {
  id: 1,
  name: 'John',
  age: 25,
}

const dog = {
  name: 'puppy',
  personId: 1,
}

function getOwner(dog, persons) {
  return persons.find((person) =>
    person.id === dog.person
  )
}

const name = 'John';

const newPerson = person;
```

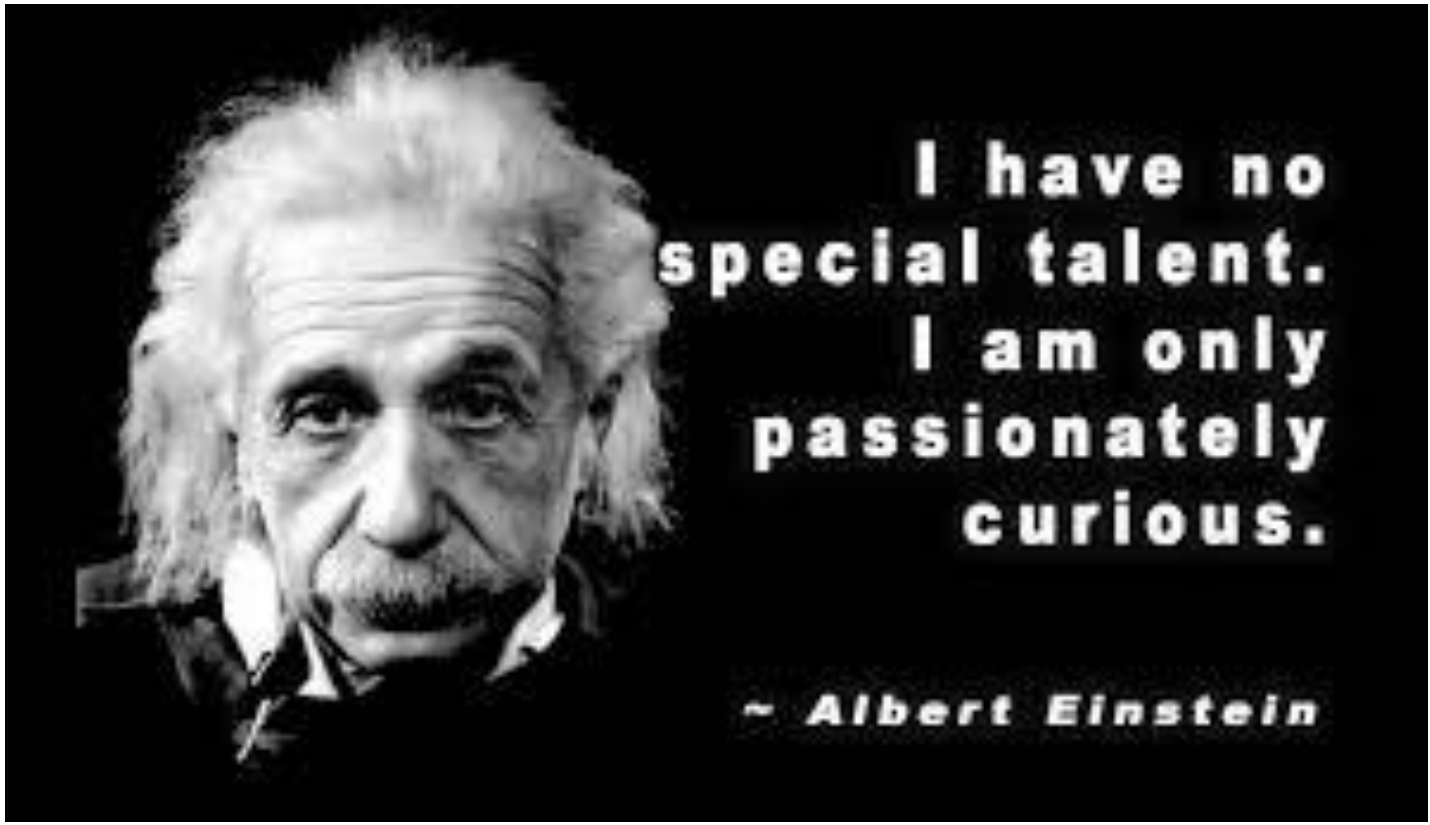


References in JavaScript

The 4 Stages of Curiosity in investigations

Based on Terry Heick's 4 stages of curiosity





Java Script……..

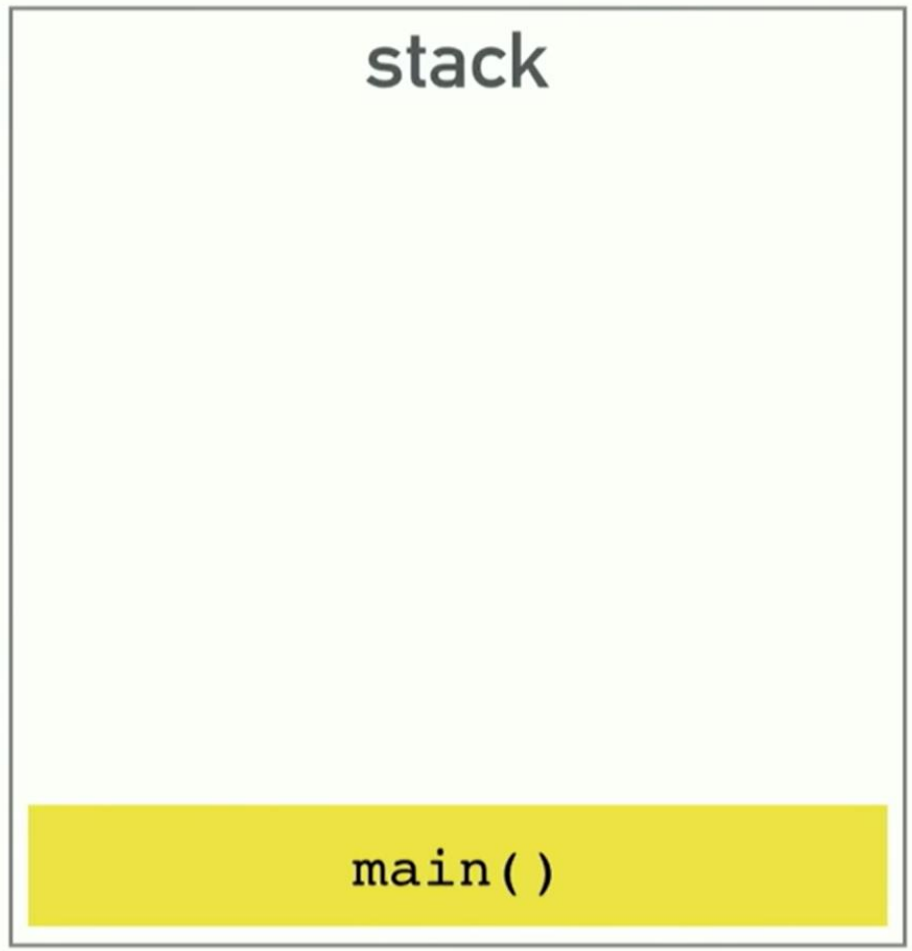
- ✓ Scoping … loops, call stack, call backs ,event loop
- ✓ Hoisting
- ✓ this
- ✓ Closure

Call Stack

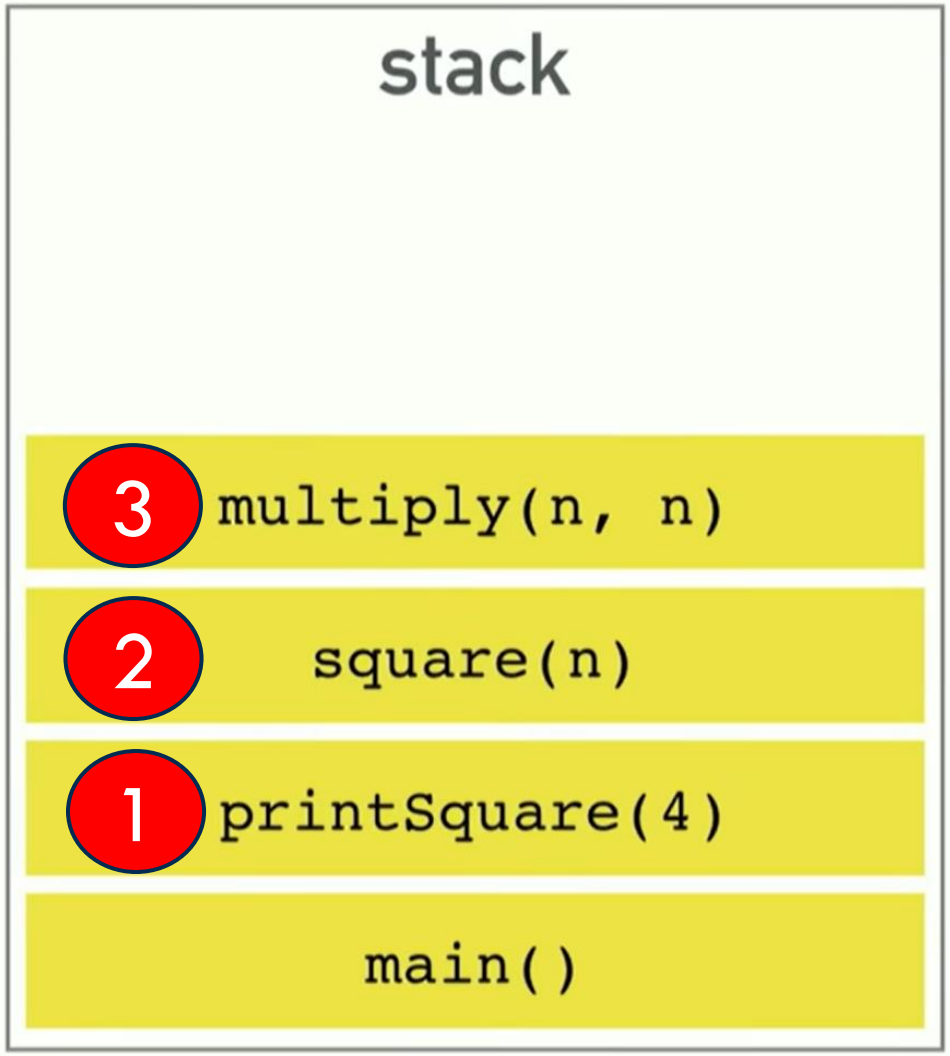
```
function multiply(a, b) {
    return a * b;
}

function square(n) {
    return multiply(n, n);
}

function printSquare(n) {
    var squared = square(n);
    console.log(squared);
}

printSquare(4);
```

If we step into a function we push it onto the stack

```
function multiply(a, b) {
    return a * b;
}

function square(n) {
    return multiply(n, n);
}

function printSquare(n) {
    var squared = square(n);
    console.log(squared);
}

printSquare(4);
```

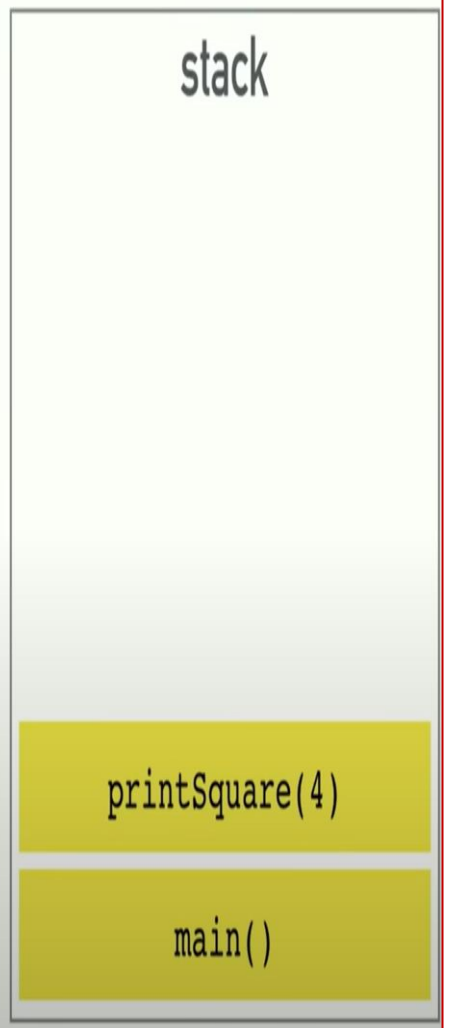


```
function multiply(a, b) {
    return a * b;
}

function square(n) {
    return multiply(n, n);
}

function printSquare(n) {
    var squared = square(n);
    console.log(squared);
}

printSquare(4);
```

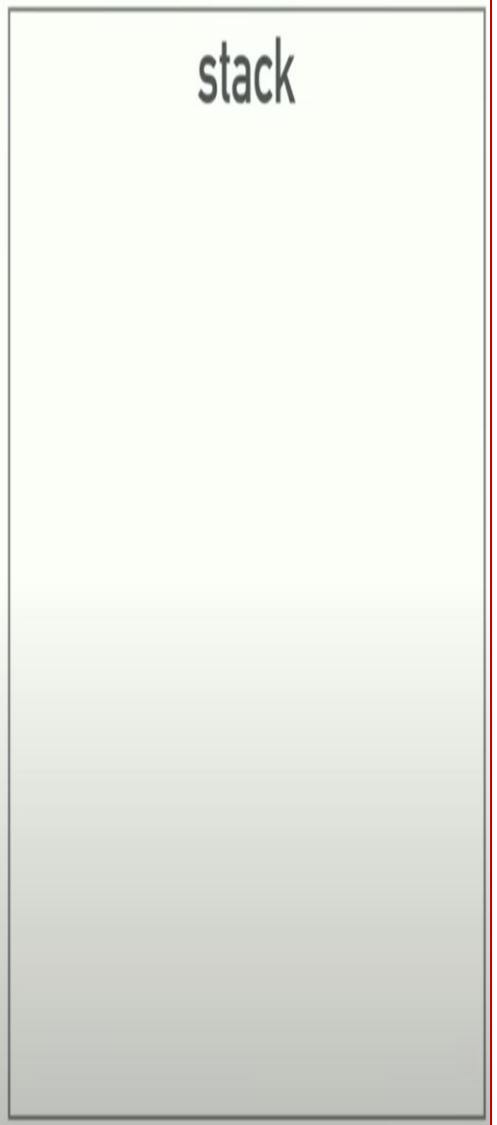


If we return from a function we pop it from the stack

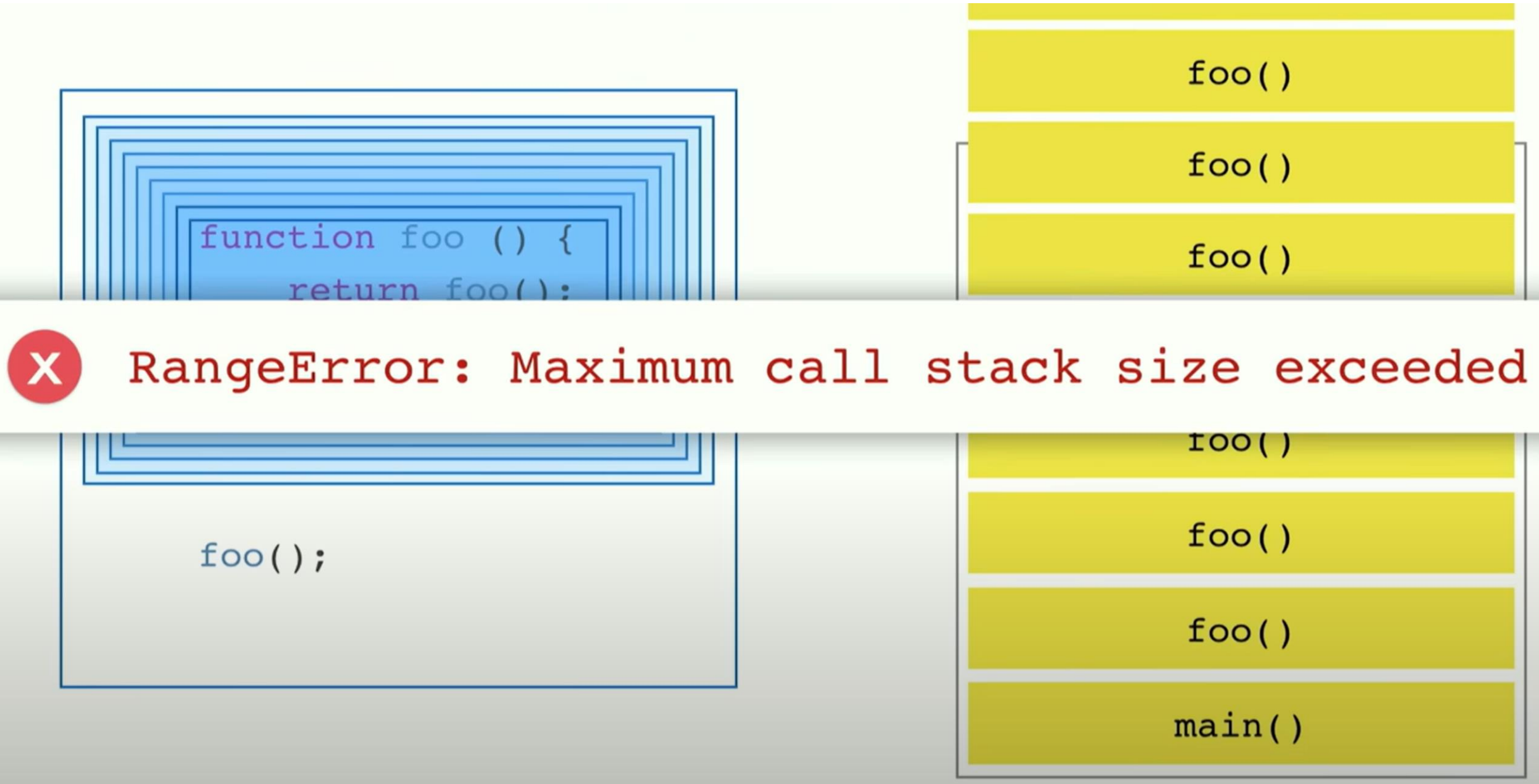
```
function multiply(a, b) {
    return a * b;
}

function square(n) {
    return multiply(n, n);
}

function printSquare(n) {
    var squared = square(n);
    console.log(squared);
}

printSquare(4);
```



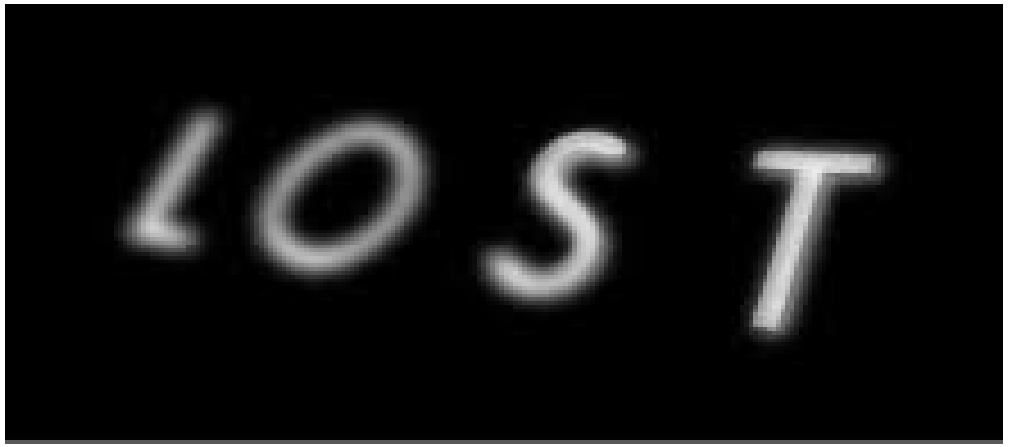

```
function multiply(a, b) {
    return a * b;
}

function square(n) {
    return multiply(n, n);
}

function printSquare(n) {
    var squared = square(n);
    console.log(squared);
}

printSquare(4);
```

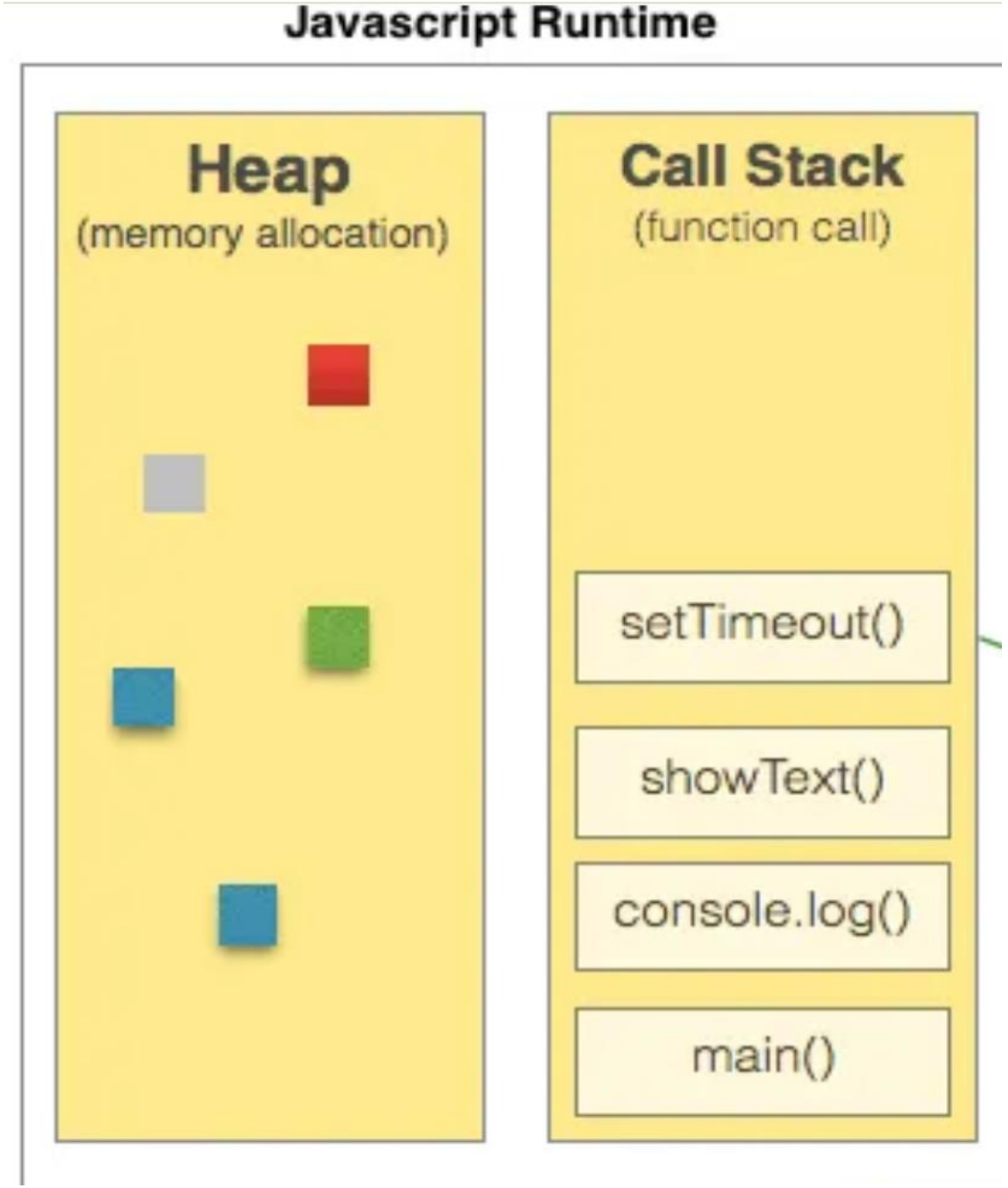


```
function foo () {
    return foo();
}

foo();
```



RangeError: Maximum call stack size exceeded


LOST

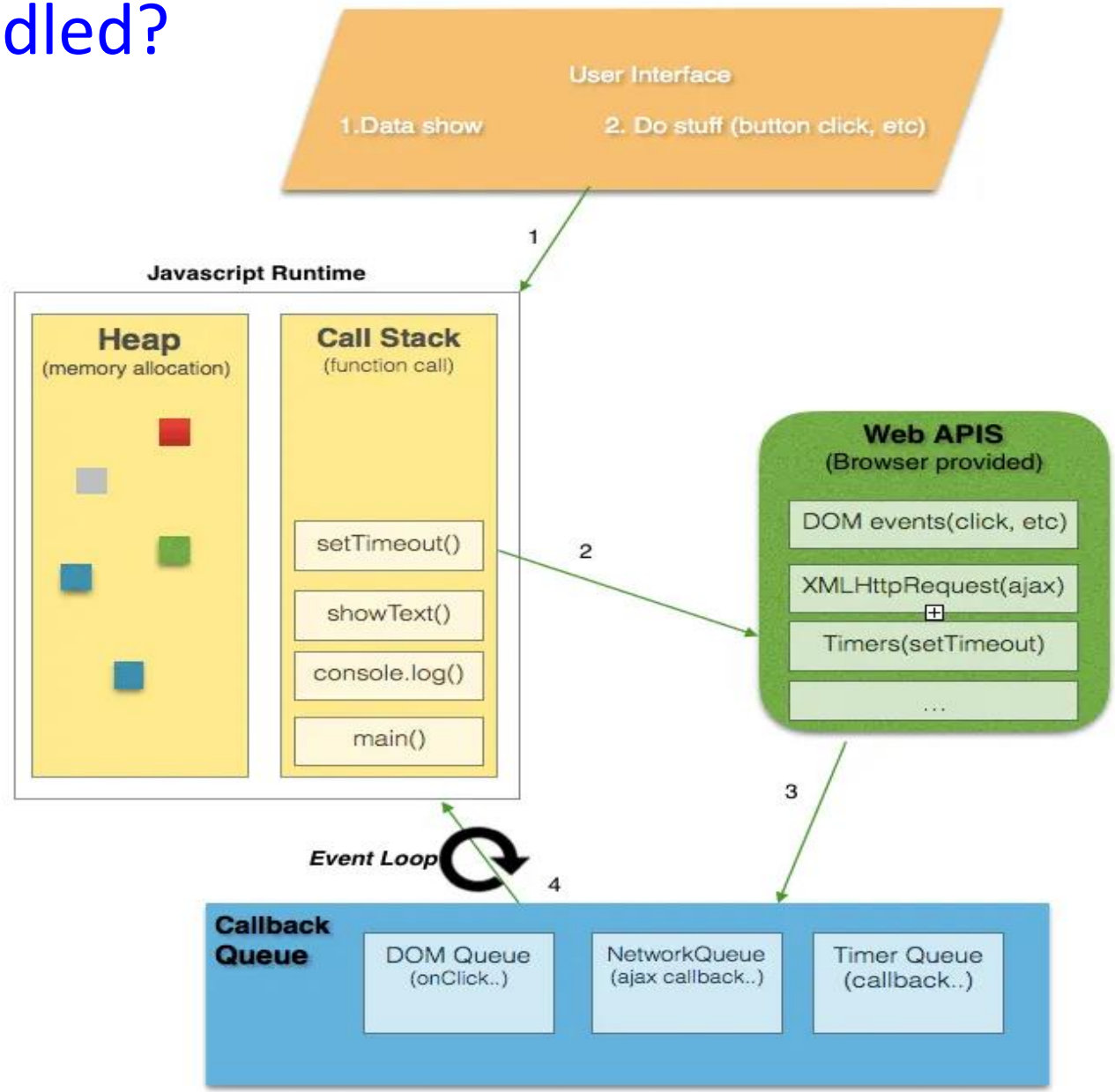

Javascript Runtime
Heap
(memory allocation)
Call Stack
(function call)
setTimeout()
showText()
console.log()
main()

Node is all about _____function execution. All these _____ callbacks doesn't run _____ and are going to run some time _____ , so can't be pushed immediately inside the _____unlike _____ functions like console.log(), mathematical operations.

Synchronous

Asynchronous

Where the hell these Async call backs go and how they are handled?



What is the output?

Call Stack

Async JS code

```
setTimeout(function greet(){
    console.log("Welcome to GeeksforGeeks!");
}, 2000);

console.log("Hi!");
```

Browser

window

Web APIs

*setTimeout

* fetch

*console

greet(){
..
}

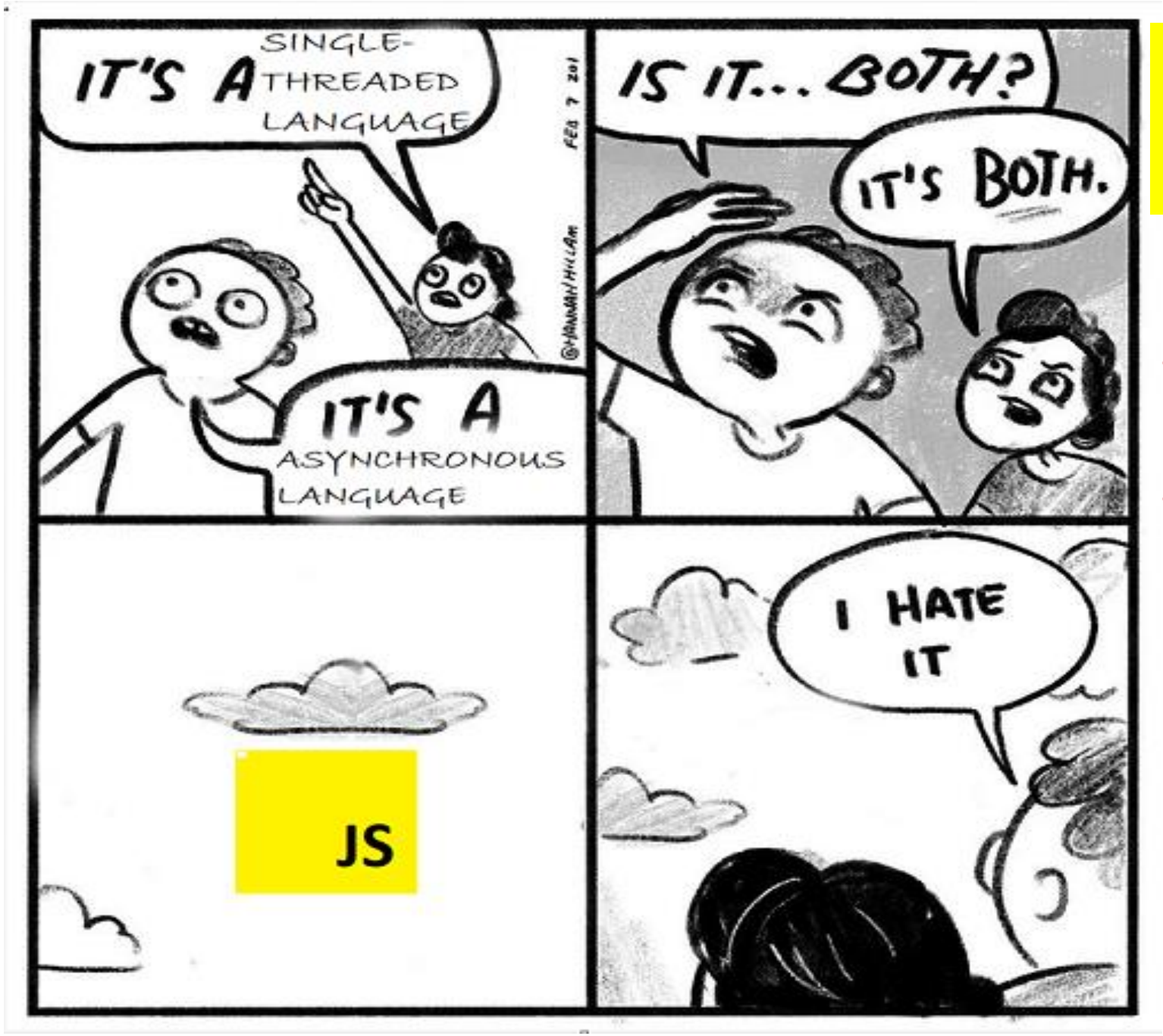
After 2 s

Callback Queue

greet(){
..
}

Event Loop

greet(){
..
}



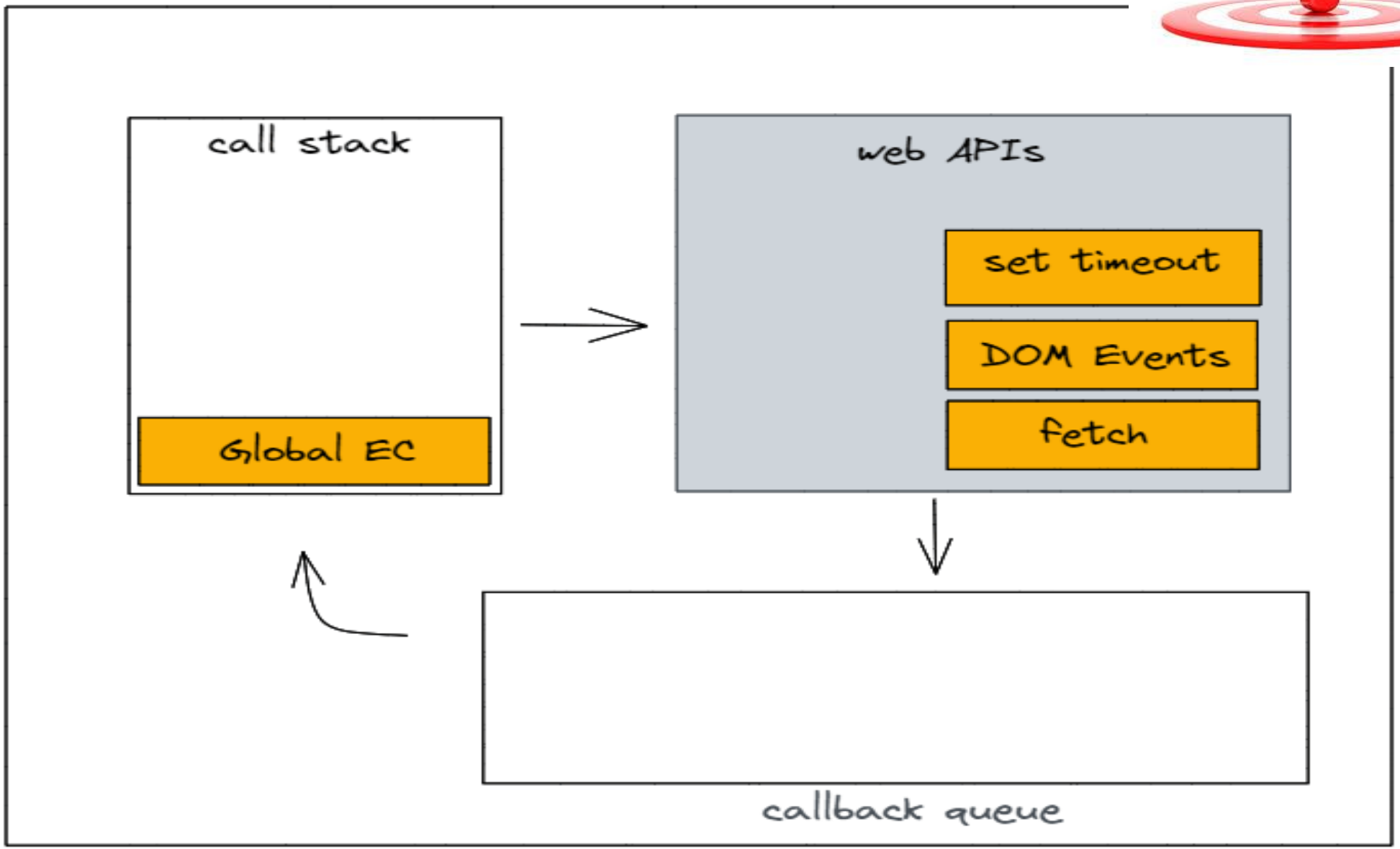
"Concurrency in JS— One Thing at a Time, except not Really, Async Callbacks"

No browser will allow the JavaScript of a single page to run concurrently. However, JavaScript does support asynchronous functions like the setTimeOut function, and while this allows some sort of scheduling, fake concurrency, and starting and stopping of threads, it is not true multi-threading.

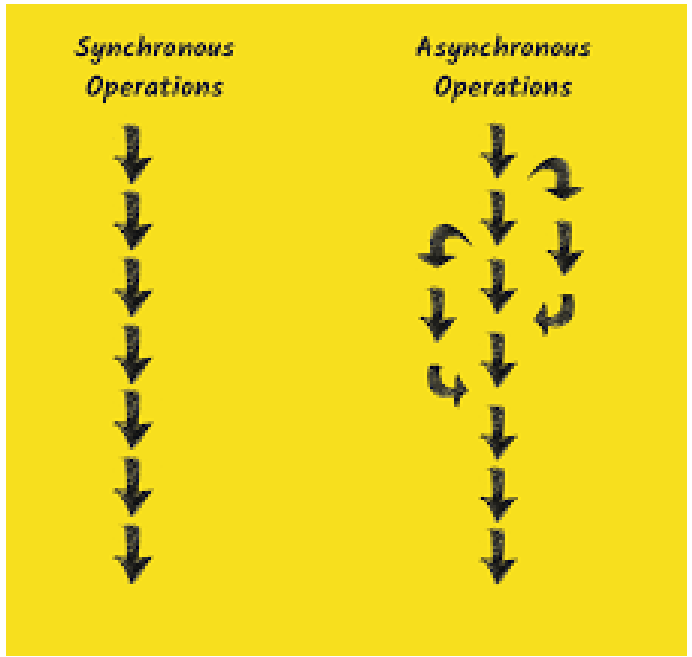
What is the output?

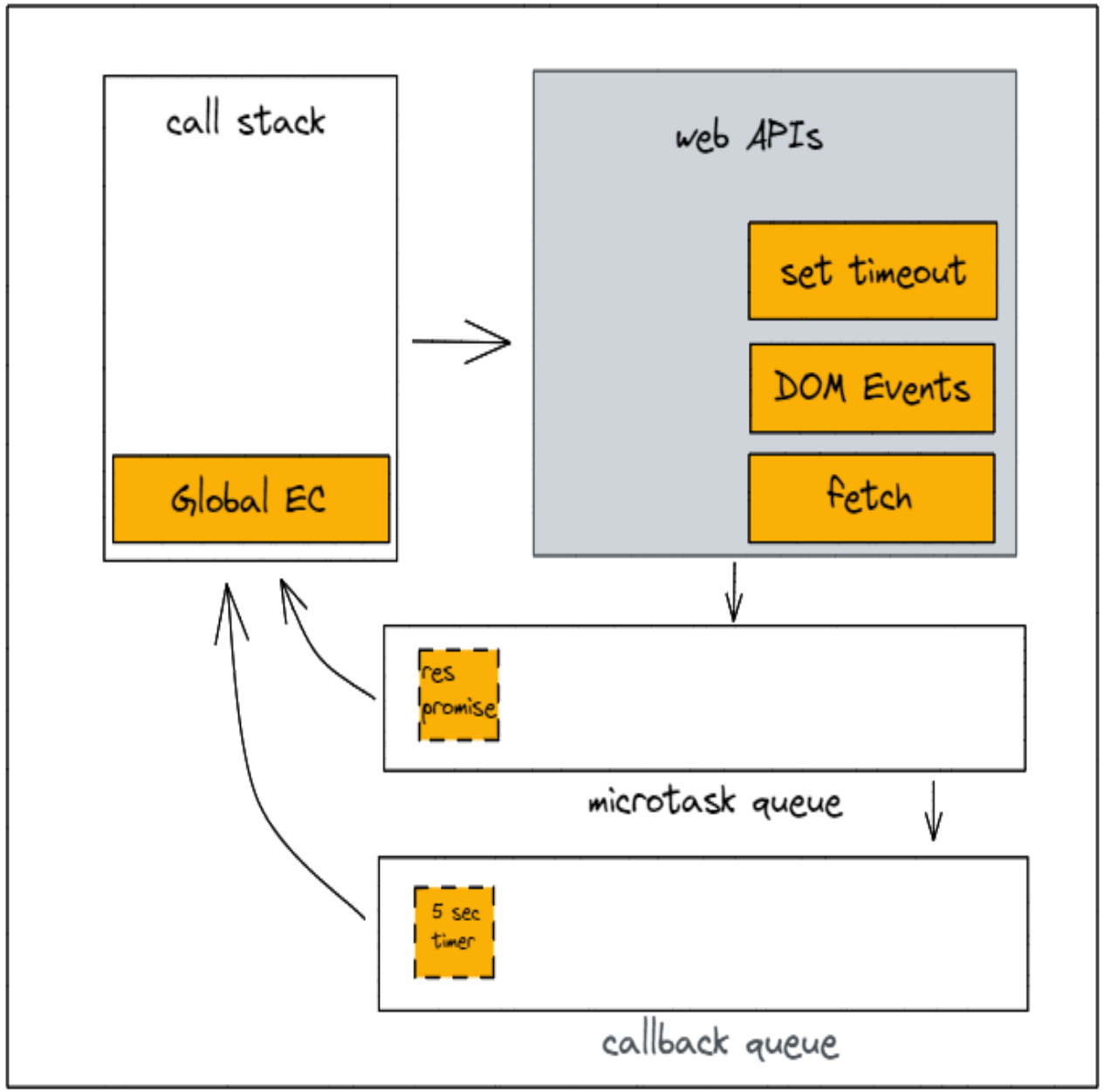
```
console.log('start’);

setTimeout(() => {
      console.log(`understand asynchronous javascript 2`);}, 5000);

setTimeout(() => {
     console.log(`understand asynchronous javascript 1`);}, 0);

console.log('end');
```

What is the output?




What is New?


Synchronous Operations
Asynchronous Operations


call stack
Global EC
web APIs
set timeout
DOM Events
fetch
res promise
microtask queue
5 sec timer
callback queue

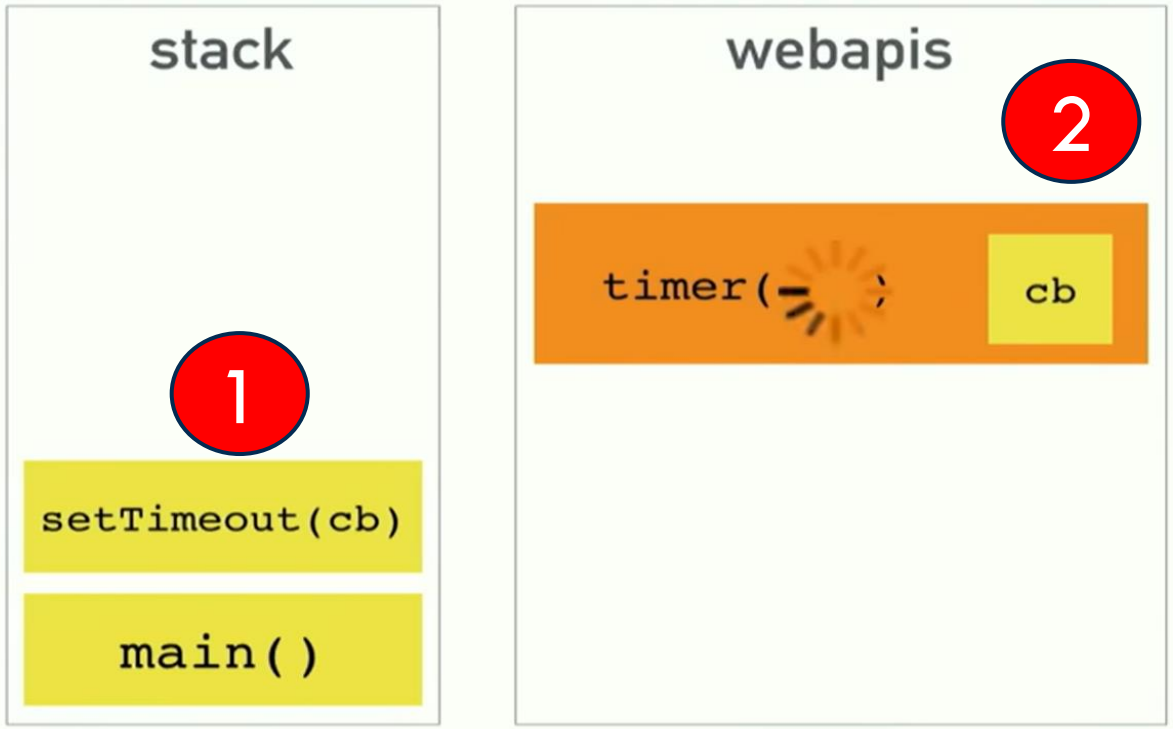
What is the output?

```
console.log('start’);

setTimeout(() => {

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

new Promise((resolve, reject) => {   resolve('Promise resolved’);  } )

      .then((res) => console.log(res))

      .catch((err) => console.log(err));

console.log('end');
```

```
console.log('Hi');

setTimeout(function cb() {
    console.log('there');
}, 5000);

console.log('JSConfEU');
```

Console

```
Hi
```



event loop

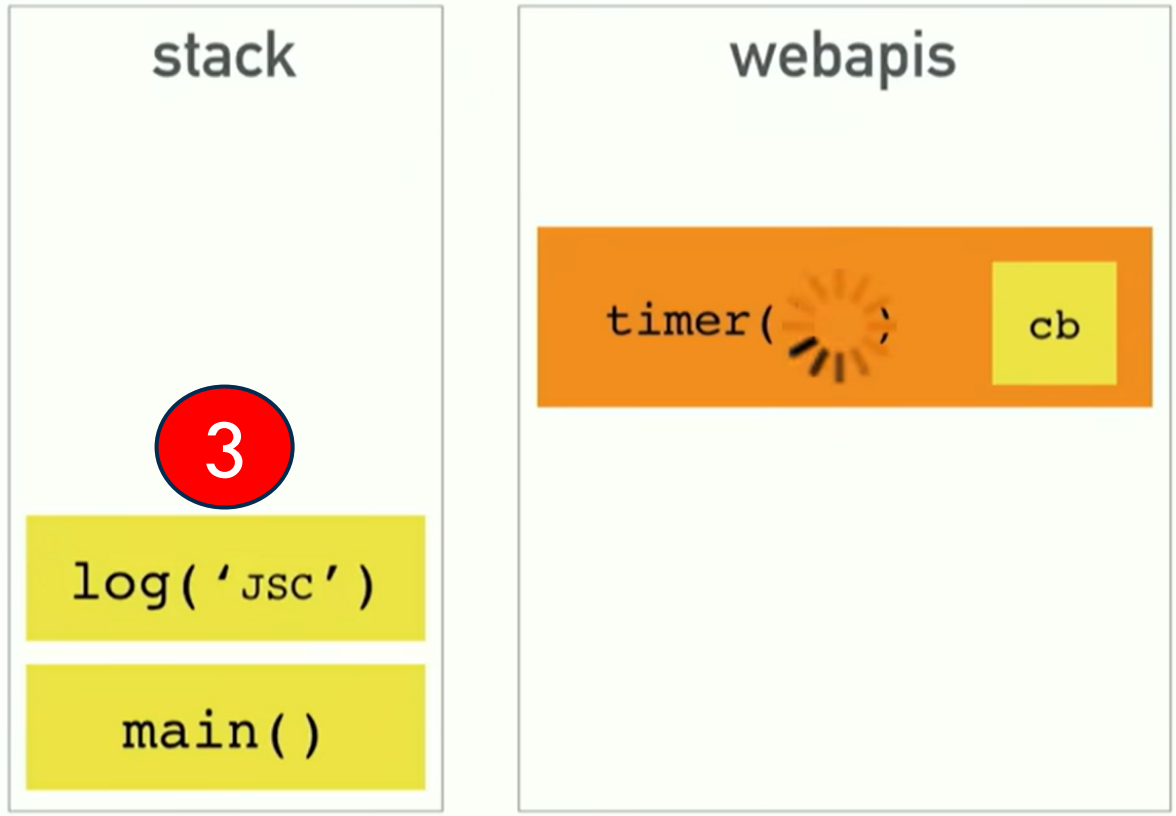
task queue

```
console.log('Hi');

setTimeout(function cb() {
    console.log('there');
}, 5000);

console.log('JSConfEU');
```

Console

Hi

JSConfEU



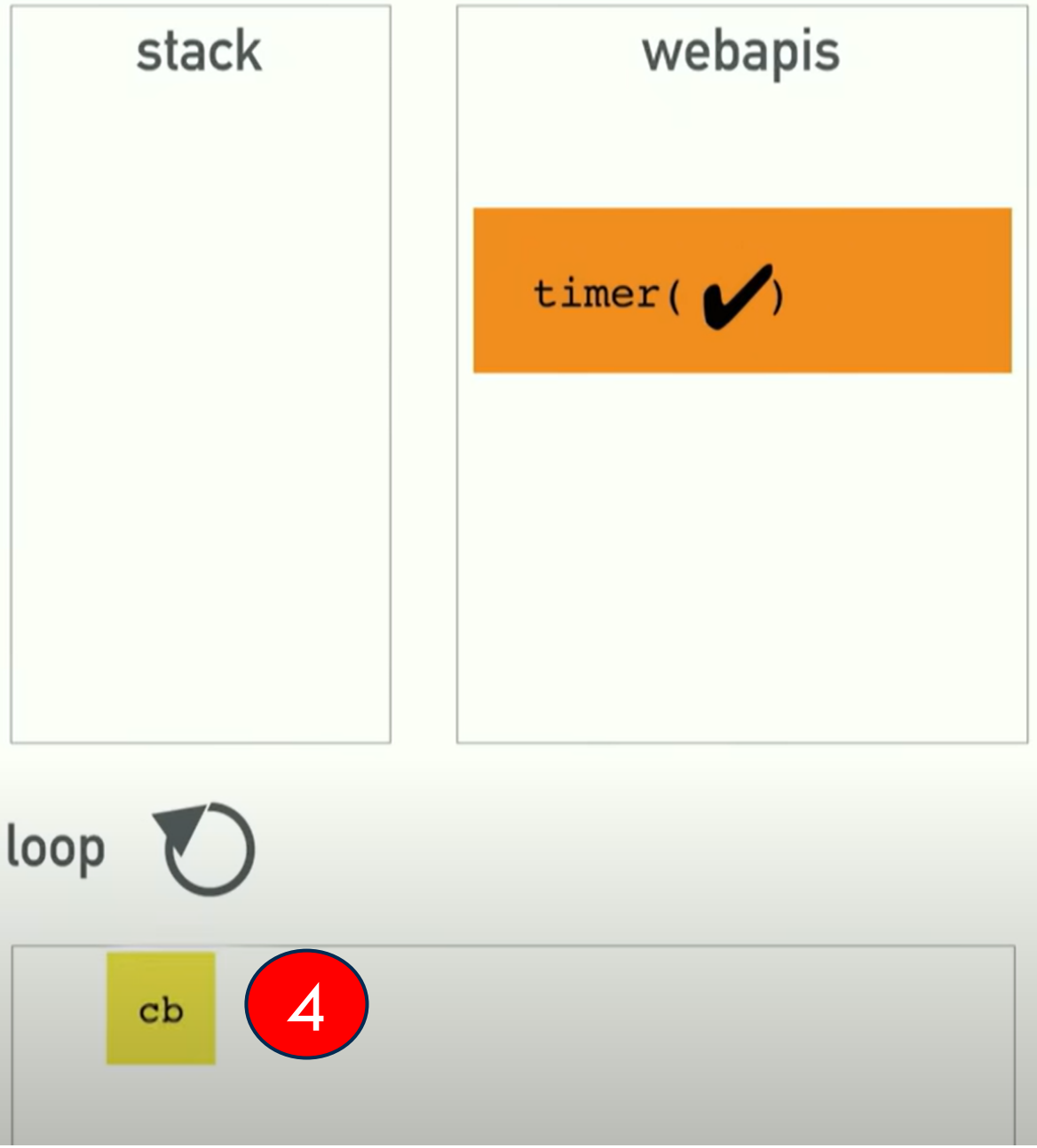
event loop

task queue

```
console.log('Hi');

setTimeout(function cb() {
    console.log('there');
}, 5000);

console.log('JSConfEU');
```

Console

```
Hi

JSConfEU
```



```
console.log('Hi');

setTimeout(function cb() {
    console.log('there');
}, 5000);

console.log('JSConfEU');
```

stack

log('there')

cb

5

webapis

event loop

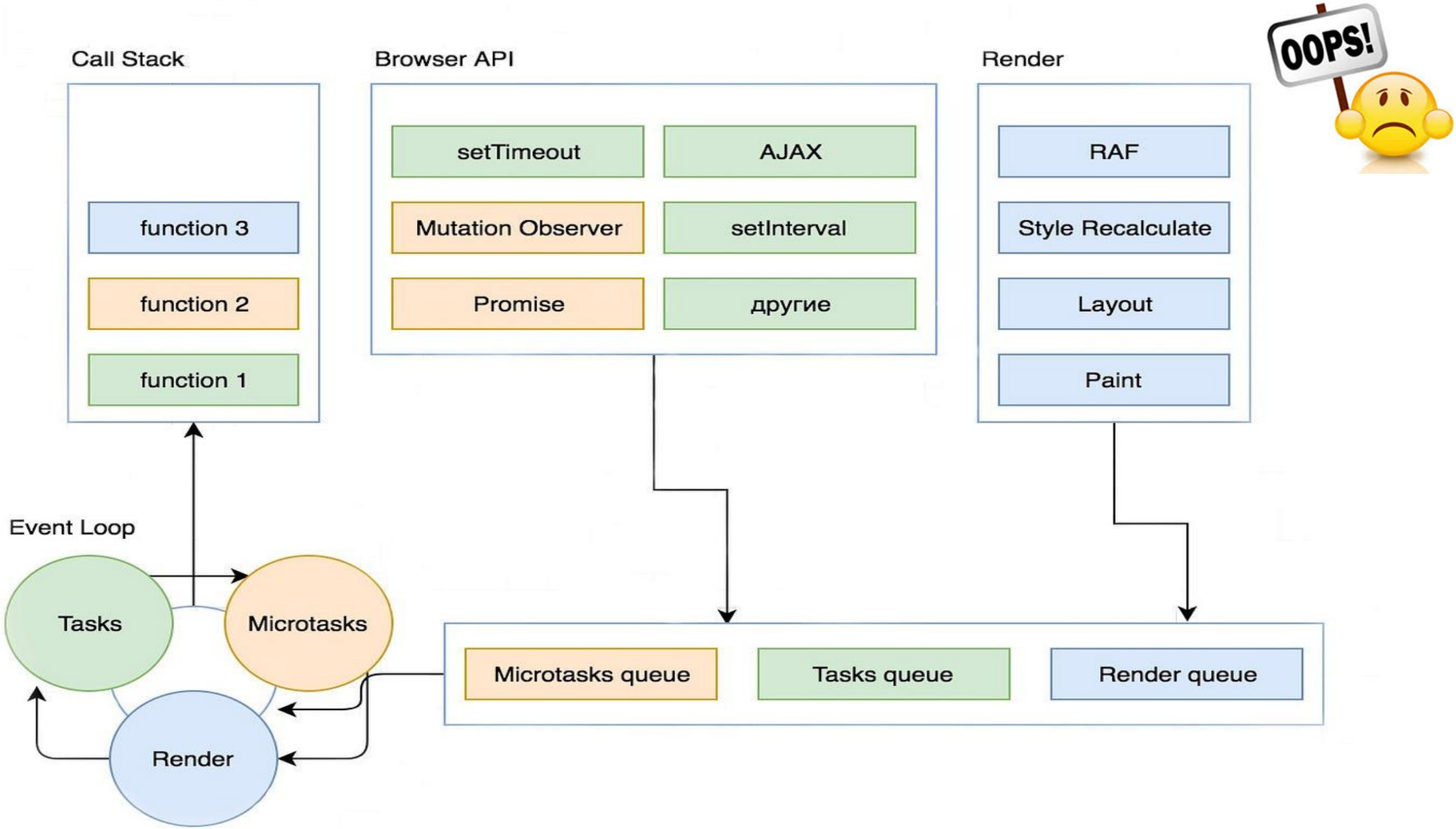
task queue

Console

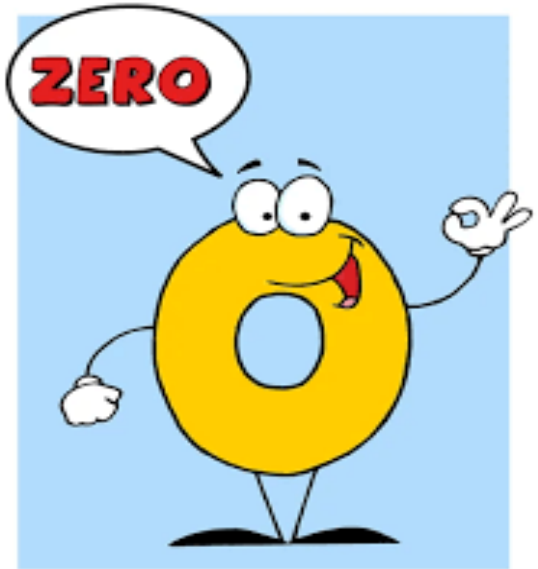
Hi

JSConfEU

there


Call Stack
function 3
function 2
function 1
Browser API
setTimeout
AJAX
Mutation Observer
setInterval
Promise
другие
Render
RAF
Style Recalculate
Layout
Paint
OOPS!
Event Loop
Tasks
Microtasks
Render
Microtasks queue
Tasks queue
Render queue

```
console.log('Hi');

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

console.log('JSConfEU');
```



```
console.log('Hi');

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

console.log('JSConfEU');
```

Console

```
Hi
```

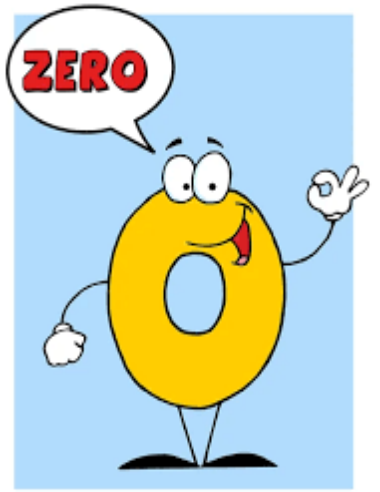
stack

1

setTimeout(cb)

main()

webapis



event loop

task queue

cb

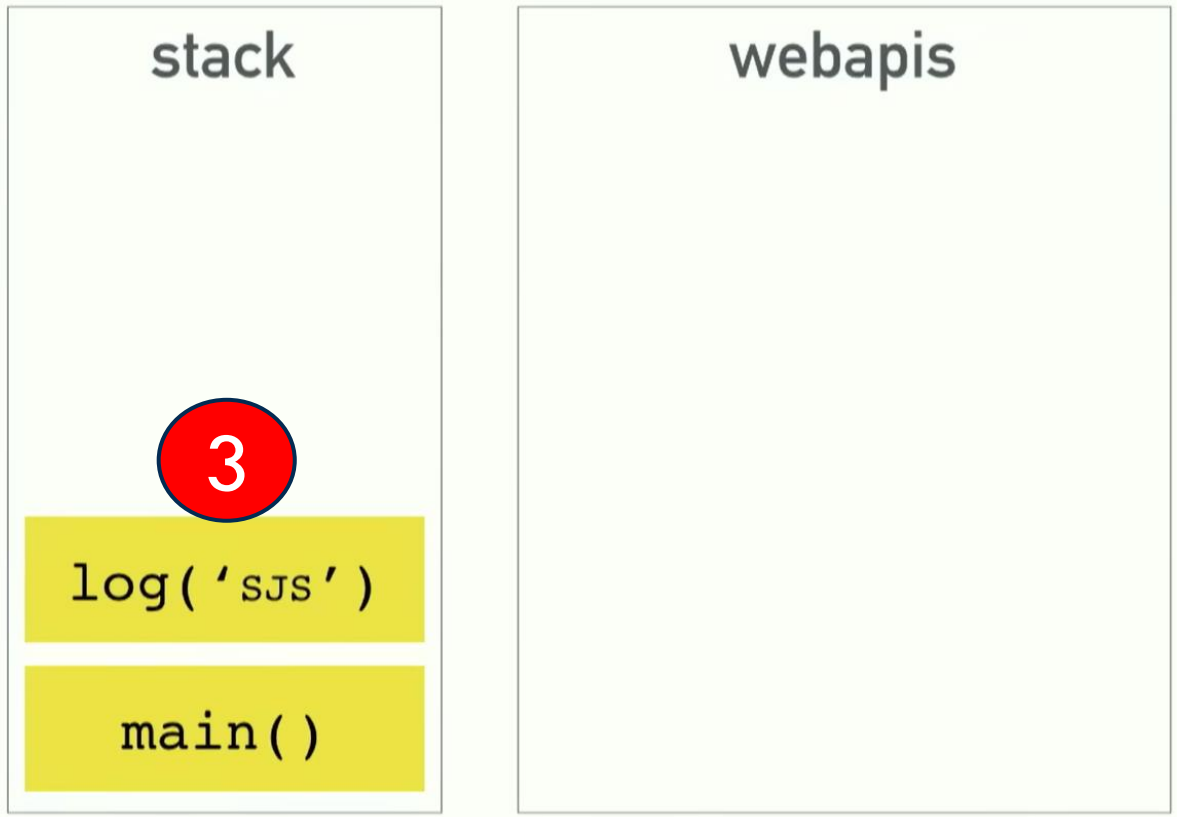
2

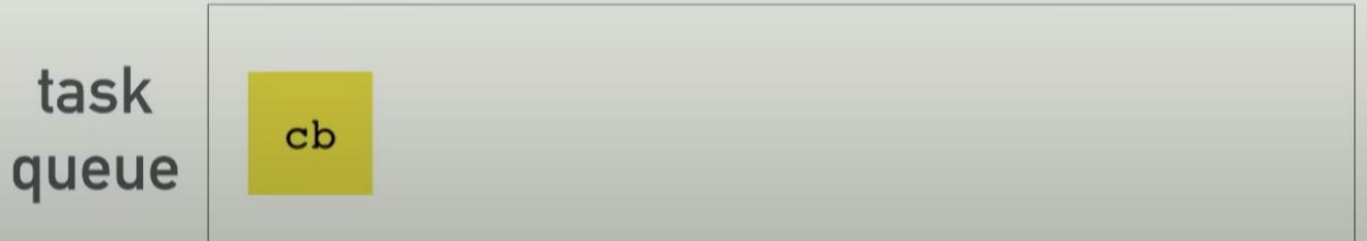
But call stack needs to be clear so stack continues to run !

```
JS nsole.log('Hi');

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

console.log('JSConfEU');
```

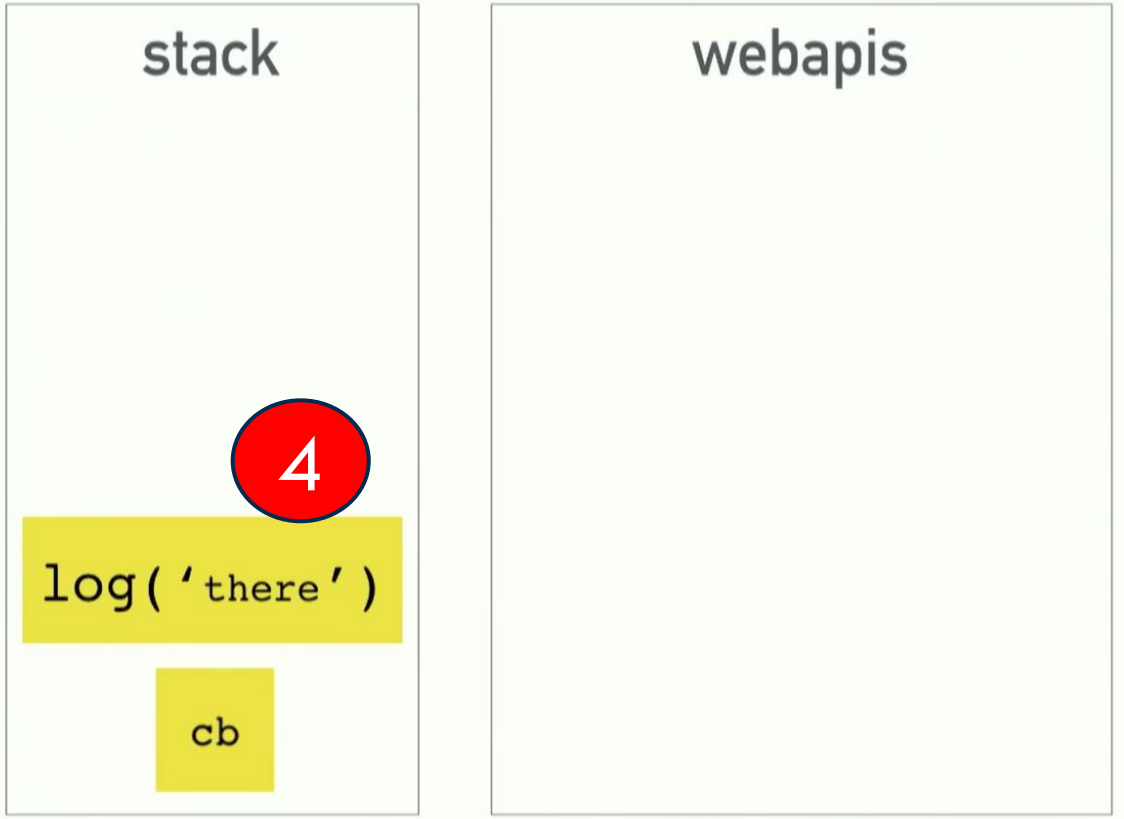
Console

```
Hi

JSConfEU
```





```
console.log('Hi');

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

console.log('JSConfEU');
```



Console

```
Hi
JSConfEU
there
```

event loop

task queue

All the webapis work the same way !

via

Asynchronous communication and Keeps the UI responsive while waiting for server data.

Lets now think about resolving the two issues ???

- printing _____ instead of _____.
- Waiting for _____ *between print outs* one by one.

```
1     for (var i = 1; i <= 10; i++) {
2       setTimeout(function () {
3         console.log(i);
4       }, 1000);
5     }
```

```
1    for (var i = 1; i <= 10; i++) {
2      setTimeout(function () {
3        console.log(i);
4      }, 1000);
5    }
```

setup before ES6

```
function fn(i) {
    setTimeout(function () { console.log(i); }, 1000 * i);
}
for (var i = 1; i <= 10; i++) fn(i);
```

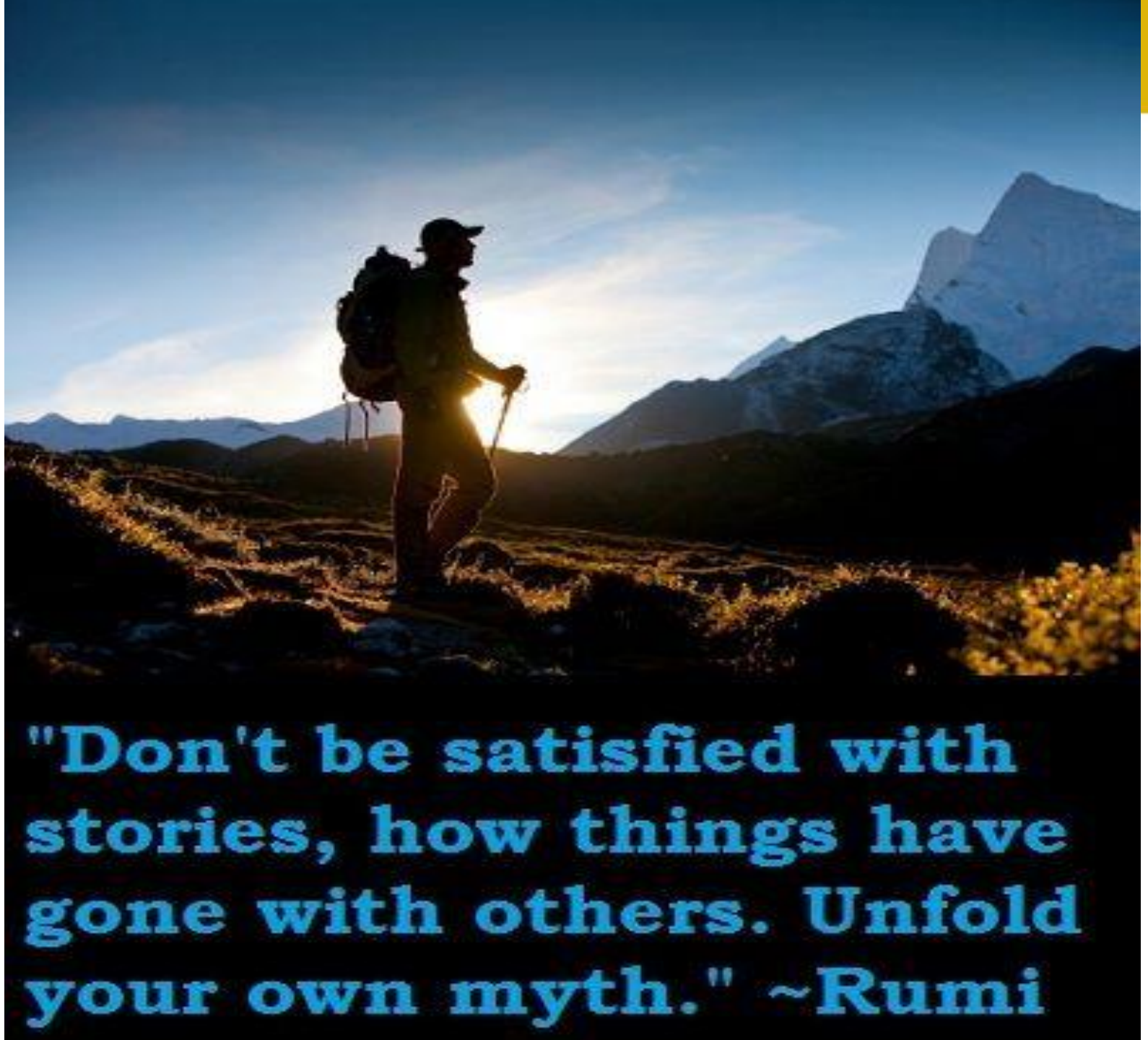
The IIFE (Immediately Invoked Function Expression) captures the current value of i and assigns it to j, creating a unique copy for each callback.

setup after ES6

```
for (let i = 1; i <= 10; i++)
    setTimeout(function () { console.log(i); }, 1000 * i);
```

Each iteration creates a new block-scoped i, preserving its value for the callback.

UNIVERSITY OF
KARACHI



UNIVERSITY OF KARACHI

Department of Compute Science (UBIT Building), Karachi, Pakistan.

1200 Acres (5.2 Km sq.)
53 Departments
19 Institutes
25000 Students

UNIVERSITY OF KARACHI

My Homeland Pakistan