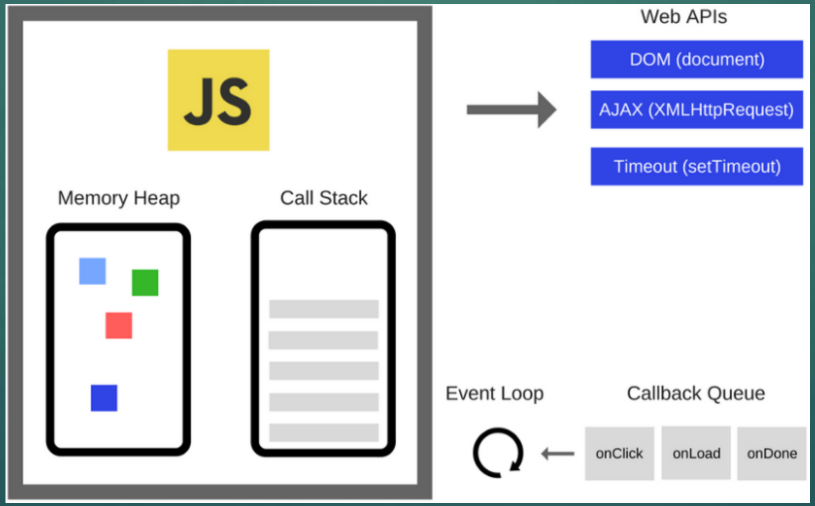
SOME BASIC CONCEPTS IN JAVASCRIPT

JAVASCRIPT ENGINE : EXECUTION CONTEXT AND THE CALL STACK



By René Logala Modia

Why is it important to know these concepts ?

- Many concepts of a programming language can be confusing, so a good knowledge of these concepts will help you master the language.
- It is often easy to create any type of application and learn any framework and library if you master the fundamentals of the language.
- And of course, as a javaScript programmer, it is also important to know these basic concepts. Because understanding the tech behind a programming language will also help you better tune your craft as a programmer.

Contents

IMPORTANT !

It should be noted that this presentation is only an overview of these three concepts among many others. For more details you will have the links at the end of the presentation so that you can deepen your knowledge on each of the concepts.

Javascript engine

What is it ?

A JavaScript engine is a kind of process virtual machine that is designed specifically to interpret and execute JavaScript code.

A JavaScript engine is a computer program that executes JavaScript code.

- The first JavaScript engines were mere interpreters, but all relevant modern engines use just-in-time compilation for improved performance.

IMPORTANT

Unlike a system virtual machine, a “process virtual machine” is less fully-functional and can run one program or process.

“Wine” is one of the examples of a process virtual machine that allows you to run Windows applications on a Linux machine, but does not provide an entire Windows OS on a Linux box.

Javascript engine

Only for browsers ?

The use of JavaScript engines is not limited to browsers. For example, the V8 engine is a core component of the Node.js and Deno runtime systems (programs that execute JS source code).

IMPORTANT

Since ECMAScript is the standardized specification of JavaScript, ECMAScript engine is another name for these engines.

Javascript engine

Who develops javascript engines ?

JavaScript engines are typically developed by web browser vendors, and every major browser has one.

Javascript engine

Some notables javascript engines

- V8 : from Google is the most used JavaScript engine. Google Chrome and the many other Chromium-based browsers.
- SpiderMonkey : developed by Mozilla for use in Firefox.
- JavaScriptCore is Apple's engine for its Safari browser. Other WebKit-based browsers also use it.
- Chakra : is the engine of the Internet Explorer browser. Originaly forked also for Edge browser, but Edge was later rebuilt as a Chromium-based browser and thus now uses V8.

Javascript engine

Brief history

- The first JavaScript engine was created by Brendan Eich in 1995 for the Netscape Navigator web browser. It was a rudimentary interpreter. (This evolved into the SpiderMonkey engine, still used by the Firefox browser.)

- The first modern JavaScript engine was V8, created by Google for its Chrome browser. V8 debuted as part of Chrome in 2008, and its performance was much better than any prior engine. The key innovation was just-in-time compilation, which can significantly improve execution times.

- Apple developed the Nitro engine for its Safari browser. Mozilla leveraged portions of Nitro to improve its own SpiderMonkey engine.

- Since 2017, these engines have added support for WebAssembly. This enables the use of pre-compiled executables.

Javascript engine

How it works ?

The creation phase

When a script executes for the first time, the JavaScript engine creates a Global Execution Context. During this creation phase, it performs the following tasks:

- Create a global object (i.e., window in the web browser or global in Node.js.)
- Create a “this” object binding which points to the global object above.
- Setup a memory heap for storing variables and function references.
- Store the function declarations in the memory heap and variables within the global execution context with the initial values as undefined.

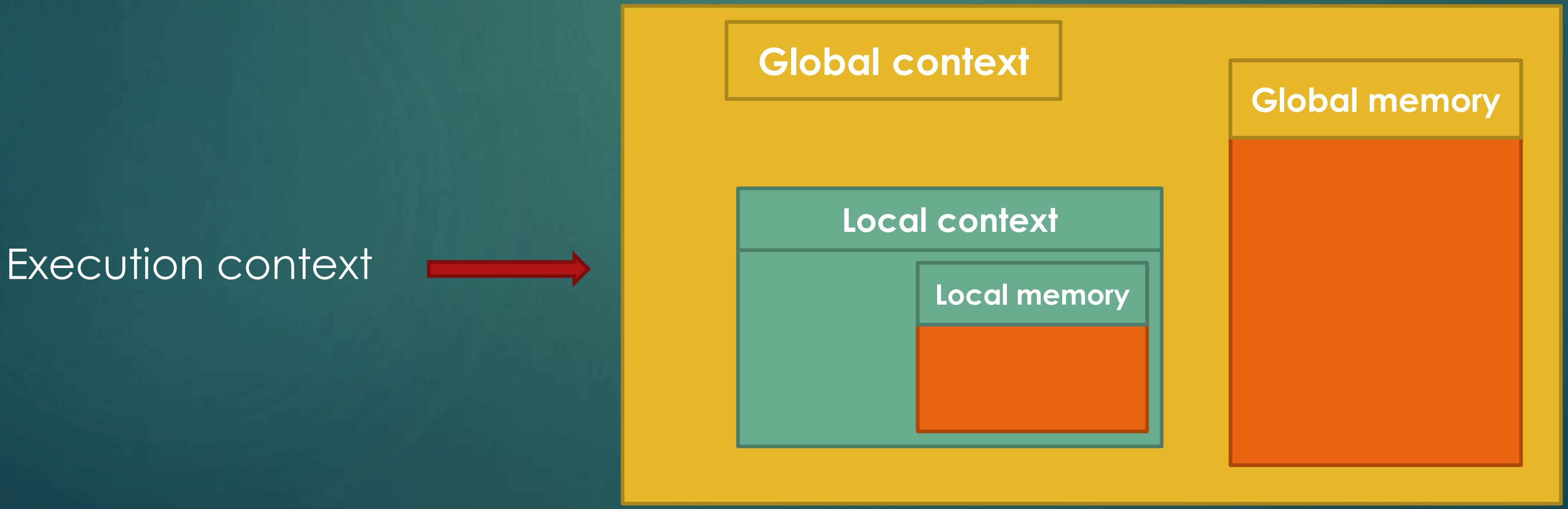
Javascript engine

Execution context

Every time you run Javascript in a browser (or in Node) the engine goes through a series of steps.

One of this step involves the creation of the Global Execution Context.

Execution Context is a fancy word for describing the environment in which your Javascript code runs.



Javascript engine

Execution context

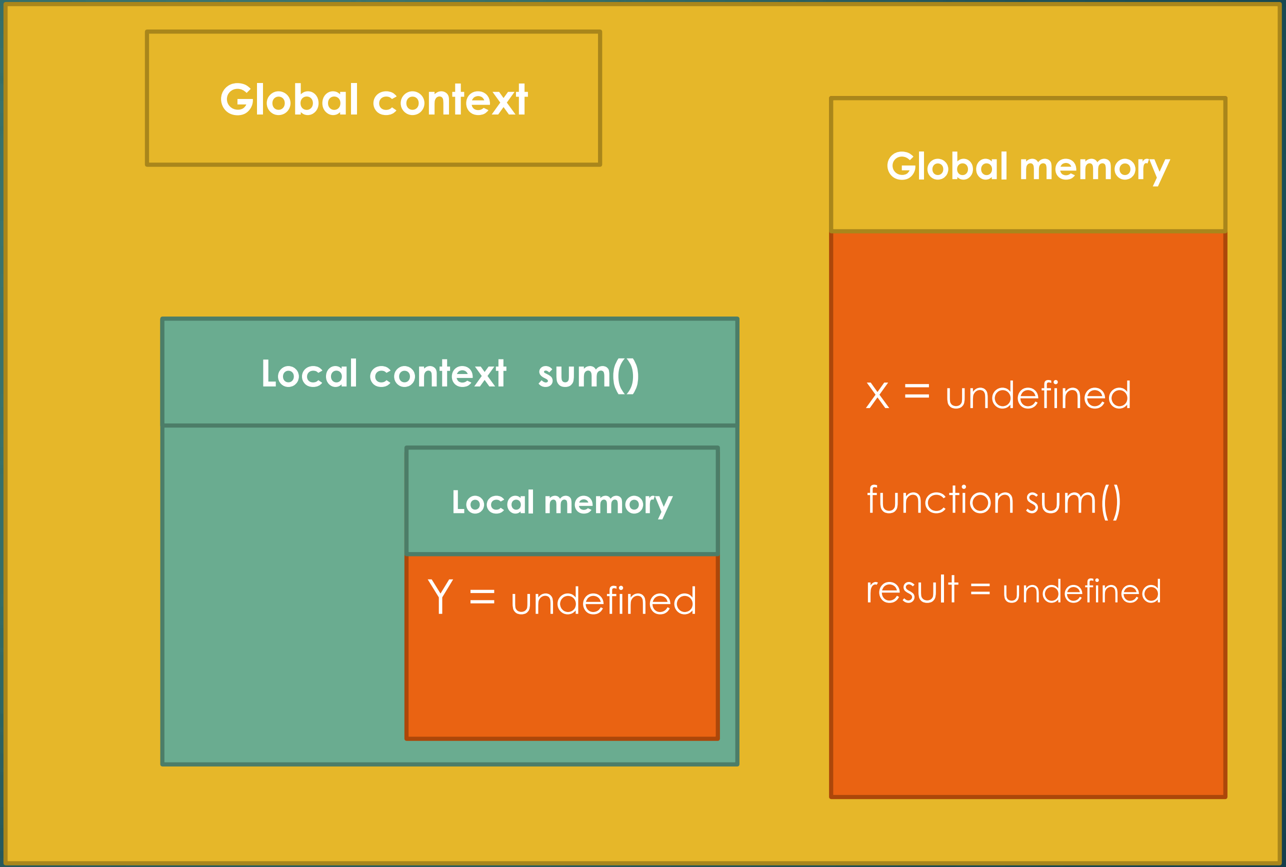
```
let x = 5;

function sum(){
  let y = 10;
  return x + y;

}

let result = sum();

console.log(result());
```



Javascript engine

How it works ?

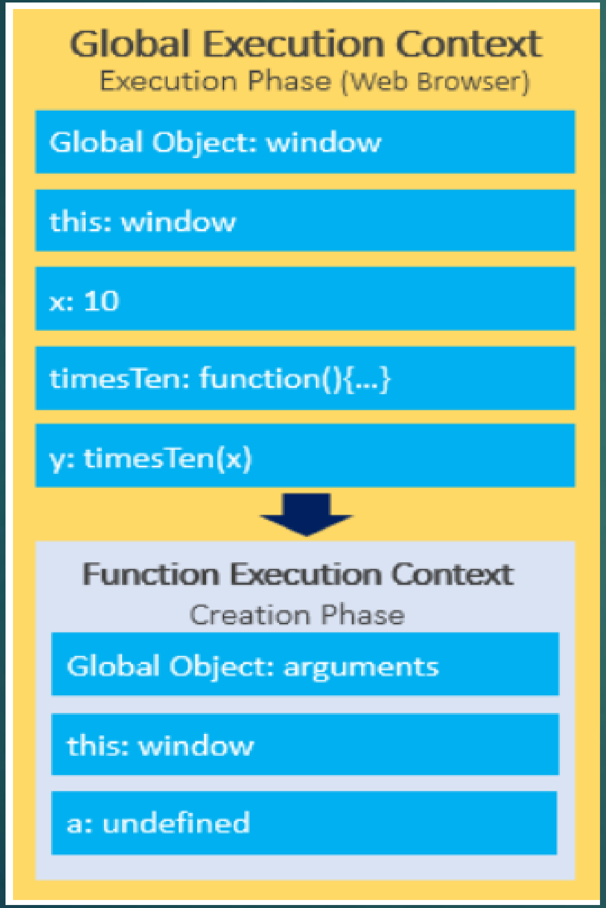
<u>The execution phase</u>

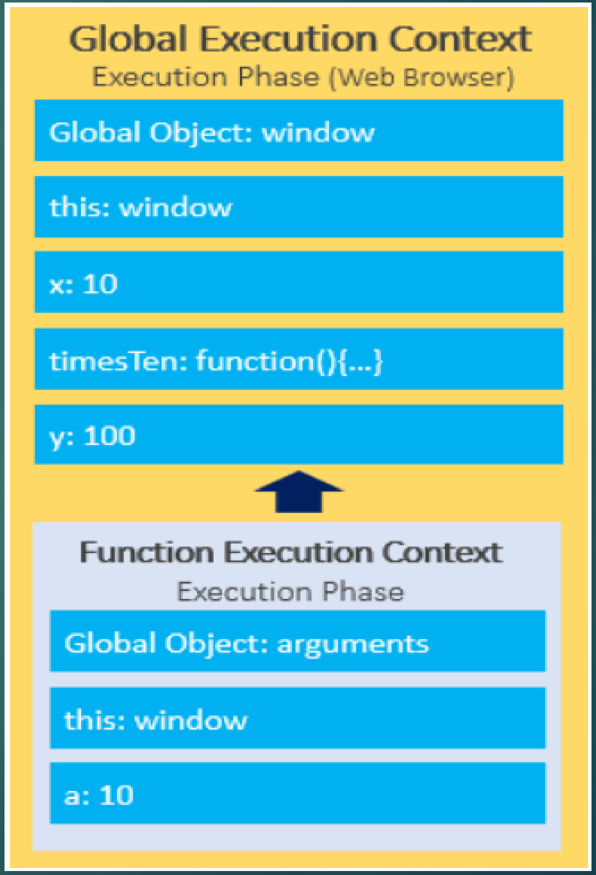
After the creation phase, the global execution context moves to the execution phase.

- During the execution phase, the JavaScript engine executes the code line by line.
- It assigns values to variables and executes the function calls.
- For every function call, the JavaScript engine creates a new Function Execution Context (local context). The Function Execution Context is similar to the Global Execution Context, but instead of creating the global object, it creates the arguments object that contains a reference to all the parameters passed into the function

Javascript engine

Execution context / Execution phase





Javascript engine

The Call stack

To keep track of all the execution contexts, including the Global Execution Context and Function Execution Contexts, the JavaScript engine uses a data structure named call stack.

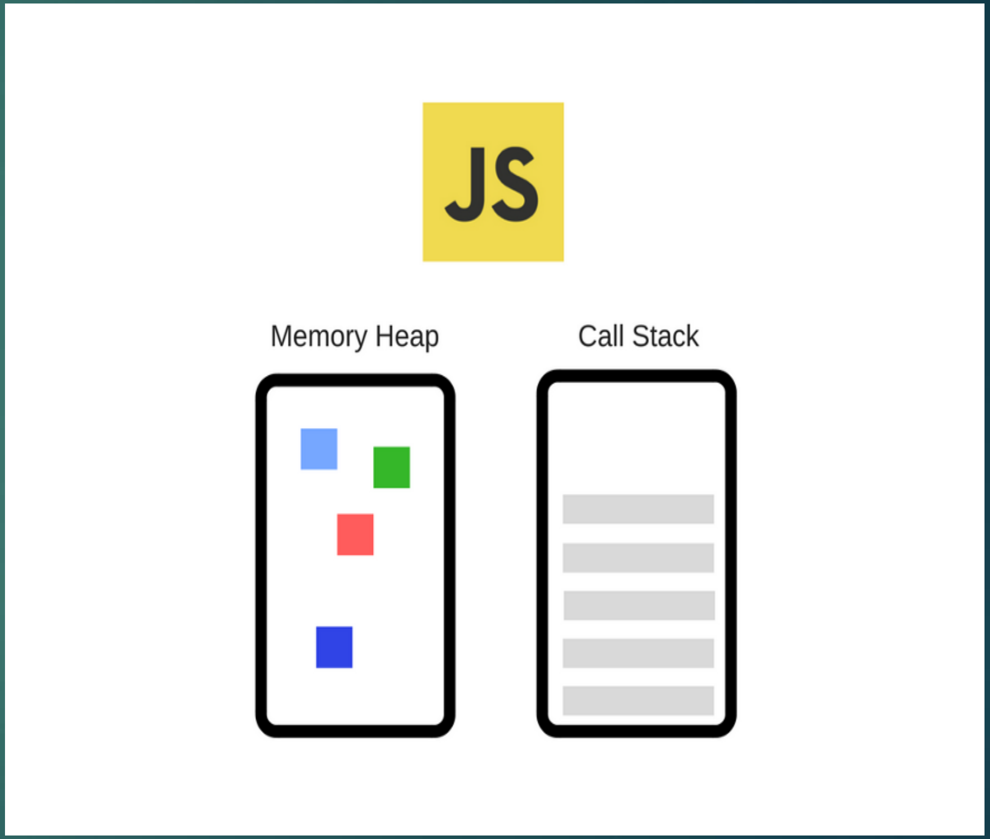
JavaScript is a single-threaded programming language, which means it has a single Call Stack. Therefore it can do one thing at a time.

The Call Stack is a data structure which records basically where in the program we are. If we step into a function, we put it on the top of the stack. If we return from a function, we pop off the top of the stack.

Javascript engine

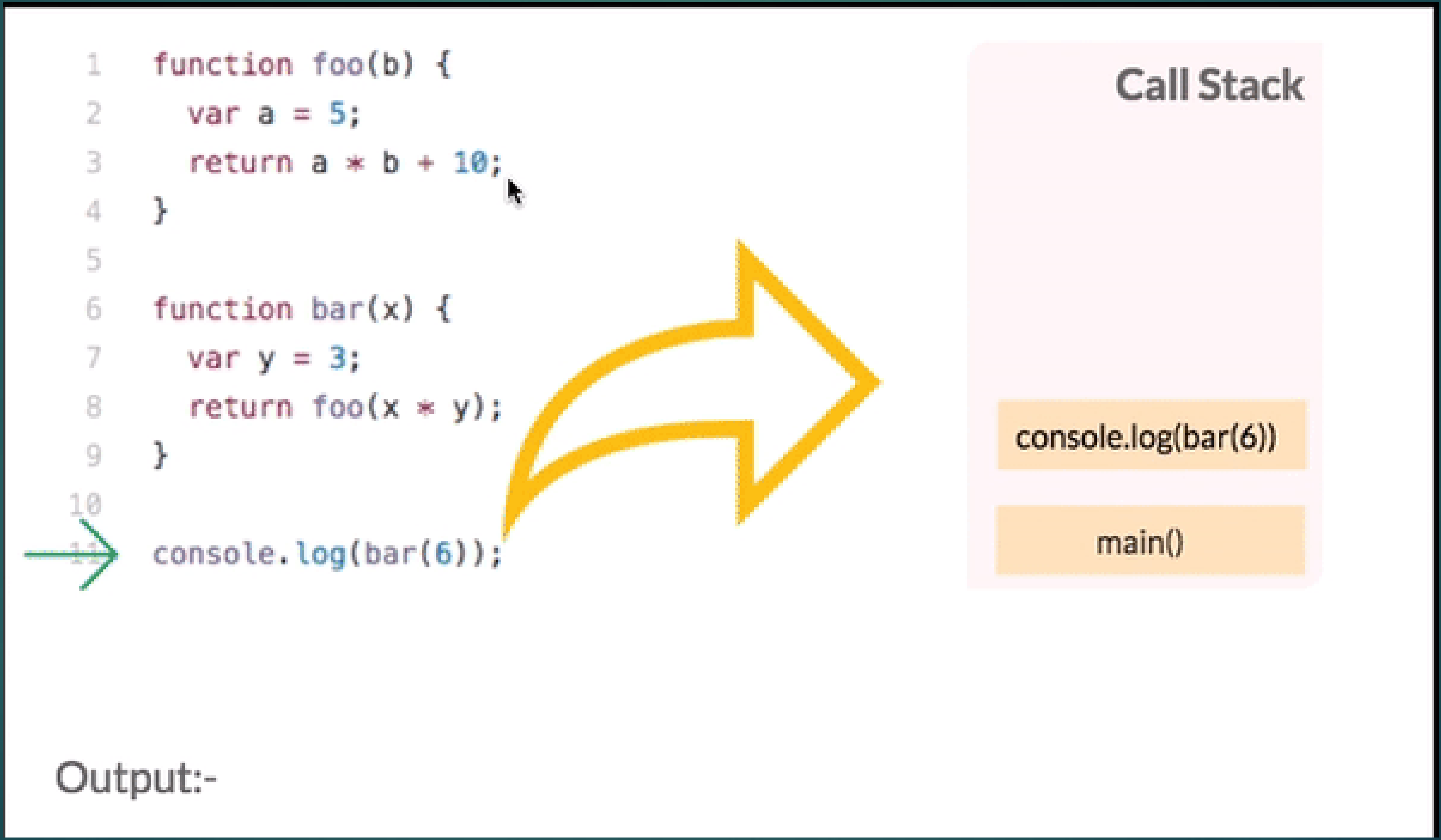
The Call stack

- Every function call gets pushed into the Call Stack
- The first thing that gets pushed is main() (or global()), the main thread of execution of your Javascript program.
- When a function ends executing it gets popped from the Call Stack



Javascript engine

The Call stack : how it works ?



```
function foo(b) {
  var a = 5;
  return a * b + 10;
}

function bar(x) {
  var y = 3;
  return foo(x * y);
}

console.log(bar(6));
```

Javascript engine

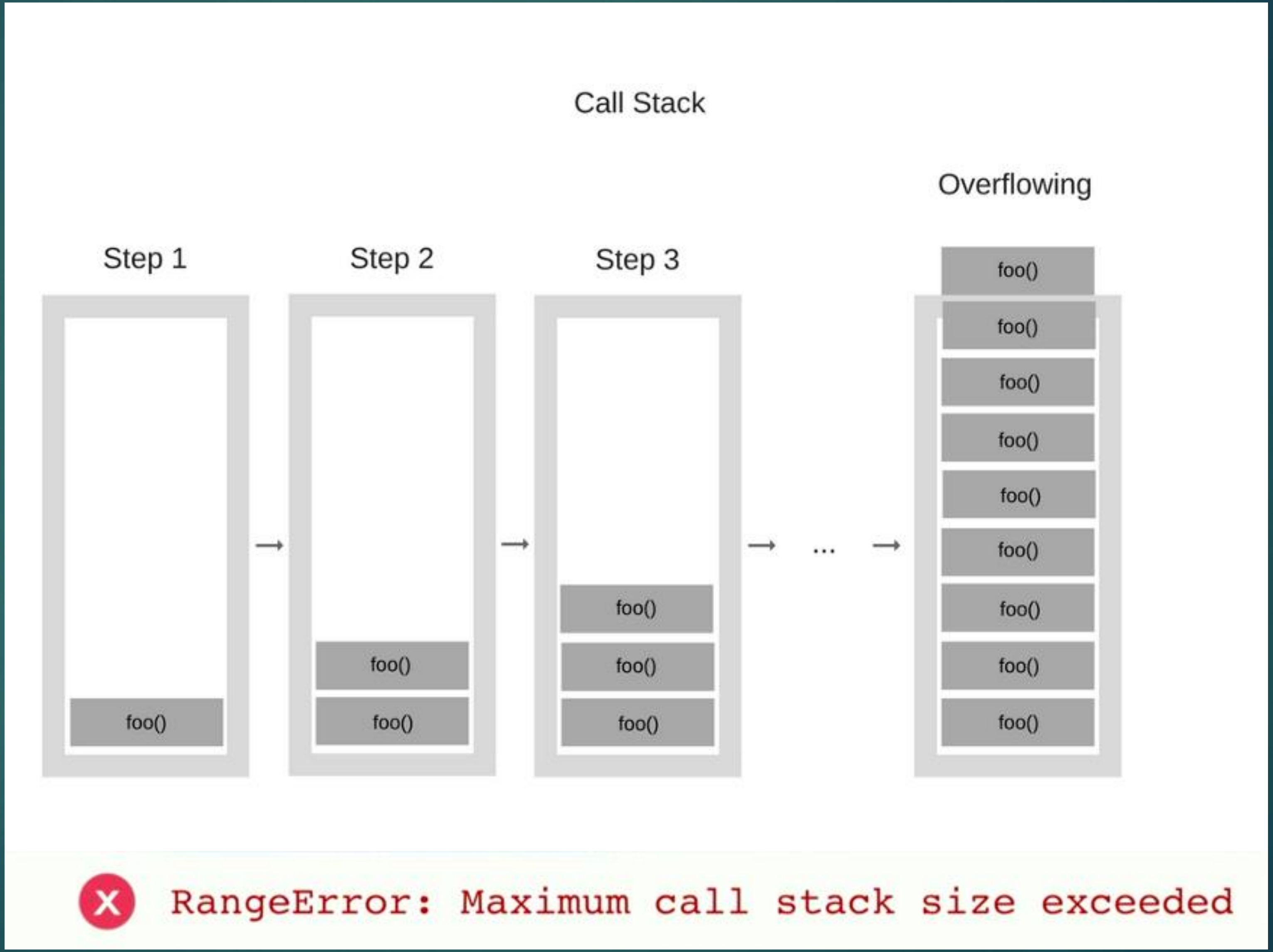
The Call stack : Stack overflow

This happens when you reach the maximum Call Stack size. And that could happen quite easily, especially if you're using recursion without testing your code very extensively.

For Chrome browser, there is a limit on the size of the stack which is 16,000 frames.

The Call stack : Stack overflow

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



RangeError: Maximum call stack size exceeded

Javascript engine

The Call stack and Event Loop

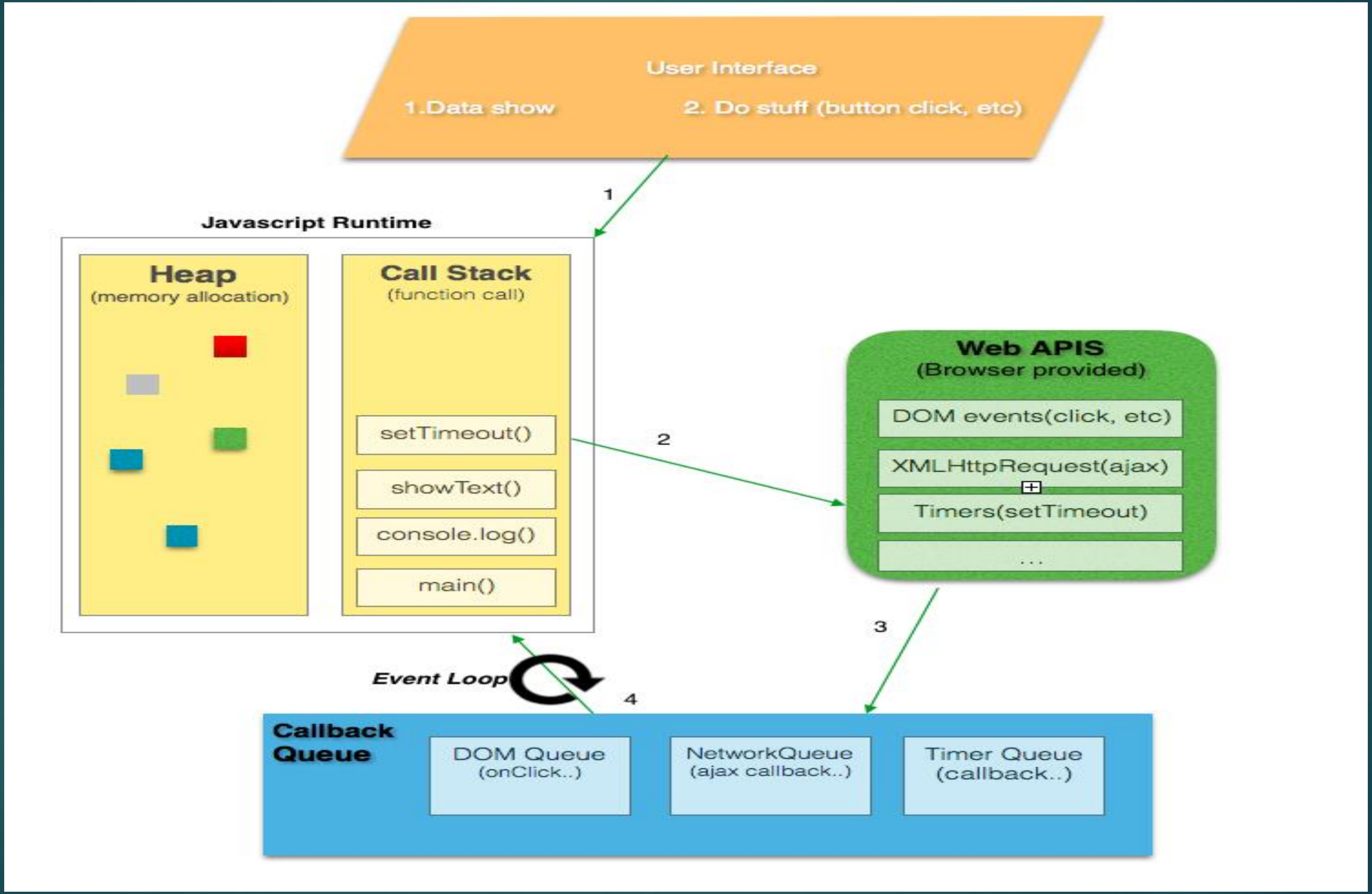
Event loop basic job is to look both at the stack and the task queue, pushing the first thing on the queue to the stack when it see stack as empty,

If a function takes a long time to run, you cannot interact with the web browser while the function is running because the page crashes.

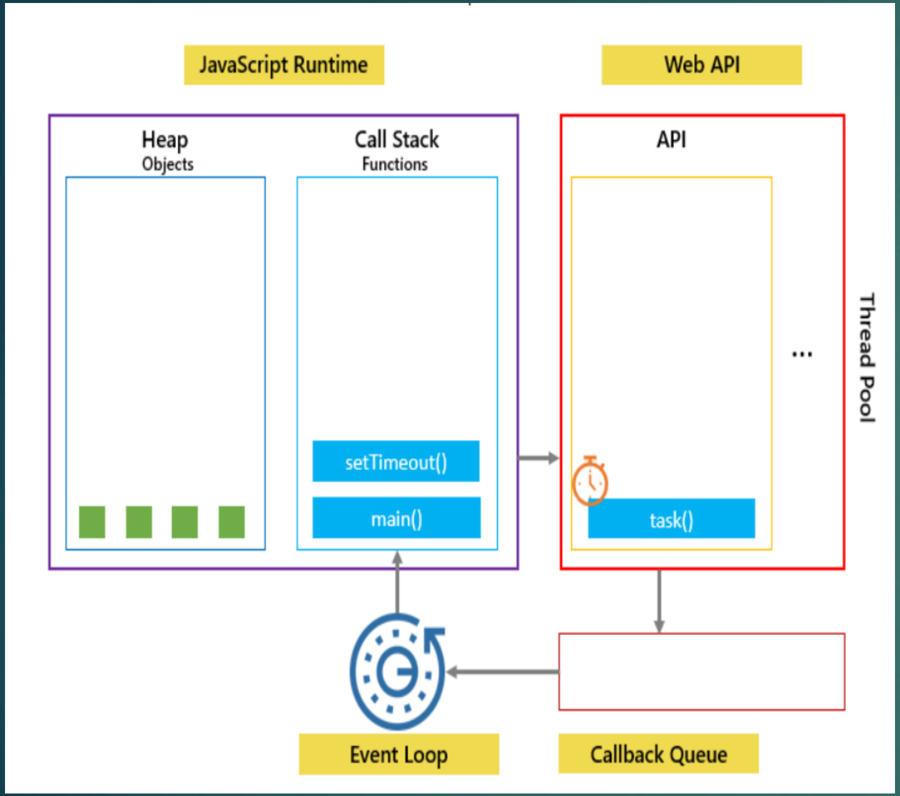
A function that takes a long time to complete is called a blocking function. Technically, a blocking feature blocks all web page interactions, such as the mouse click.

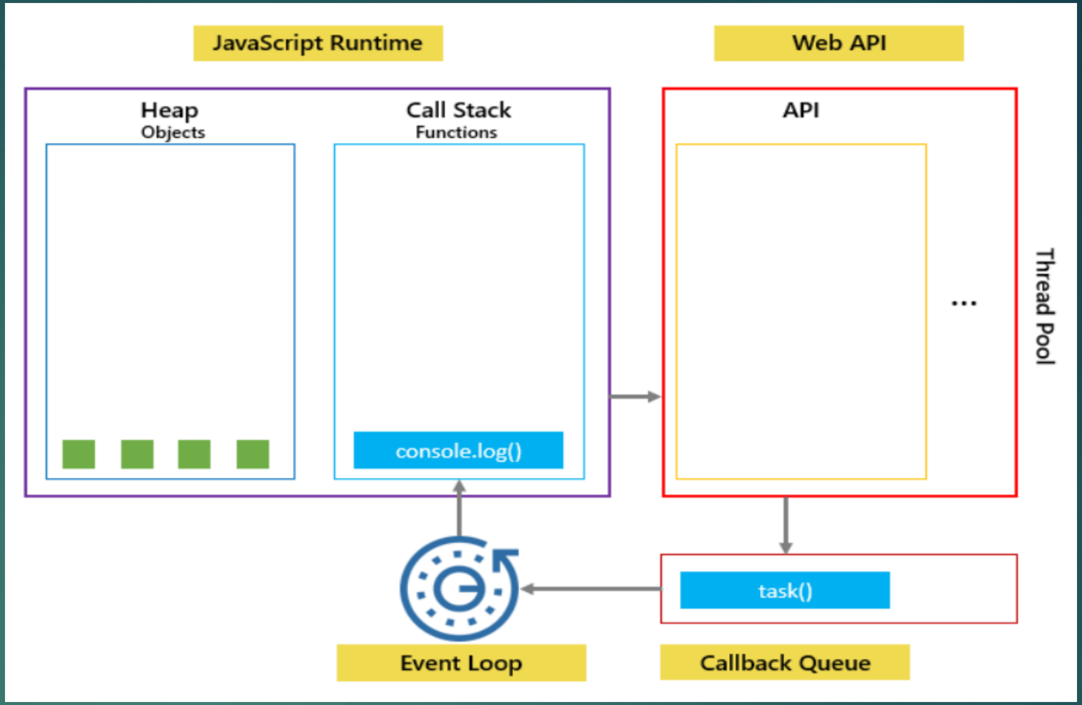
To prevent a blocking function from blocking other activity, you usually put it in a callback function for later execution and this is when the event loop comes in.

The Call stack and Event Loop

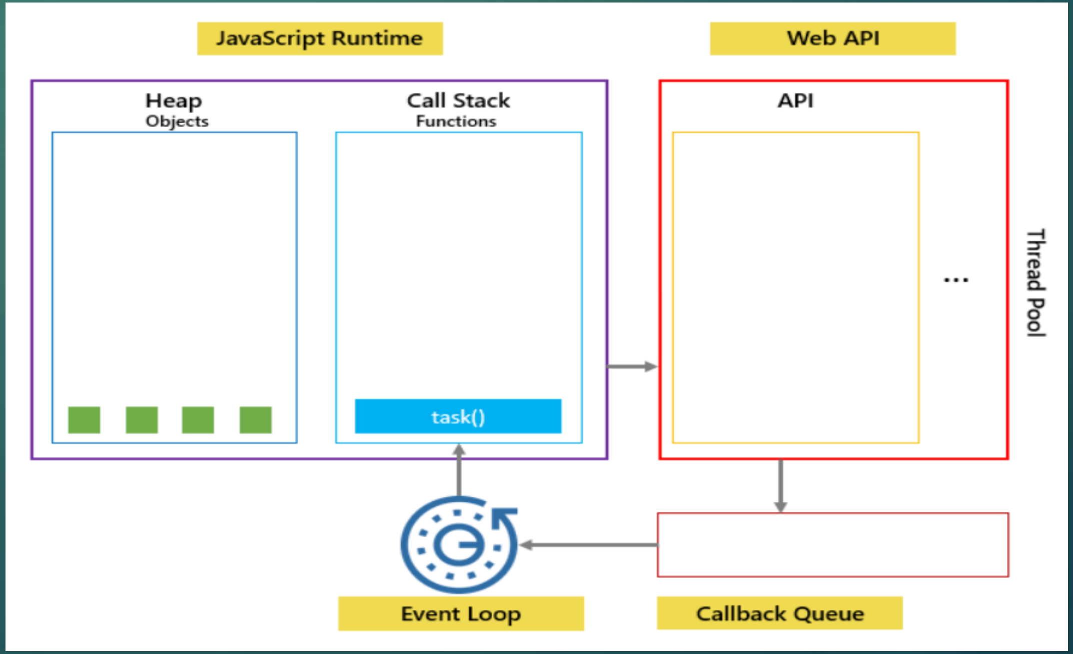


1


JavaScript Runtime
Web API
Heap
Objects
Call Stack
Functions
setTimeout()
main()
API
task()
...
Thread Pool
Event Loop
Callback Queue


JavaScript Runtime
Web API
Heap
Objects
Call Stack
Functions
console.log()
API
...
Thread Pool
task()
Event Loop
Callback Queue

2


JavaScript Runtime
Web API
Heap
Objects
Call Stack
Functions
task()
API
...
Thread Pool
Event Loop
Callback Queue

3

Some referral sources

- https://en.wikipedia.org/wiki/JavaScript_engine
- https://web.archive.org/web/20181208123231/http://developer.telerik.com/featured/a-guide-to-javascript-engines-for-idiots/
- https://github.com/leonardomso/33-js-concepts
- https://www.javascripttutorial.net/javascript-execution-context/
- https://medium.com/@gaurav.pandvia/understanding-javascript-function-executions-tasks-event-loop-call-stack-more-part-1-5683dea1f5ec

Thank you for your attention !