

# the Master Course

{C0DENATION}

# Intermediate JavaScript

## JavaScript Engines

{ CØDENATION }

# Learning Objectives

**To discover how a JavaScript engine operates**

**To be familiar with the JavaScript execution context**

**To explore JavaScript engine call stack, memory heap, event loop and callback queue**

# Intermediate JS

## JavaScript Engines

...are typically developed by **web browser** vendors



= V8 Engine



= Chakra



= SpiderMonkey

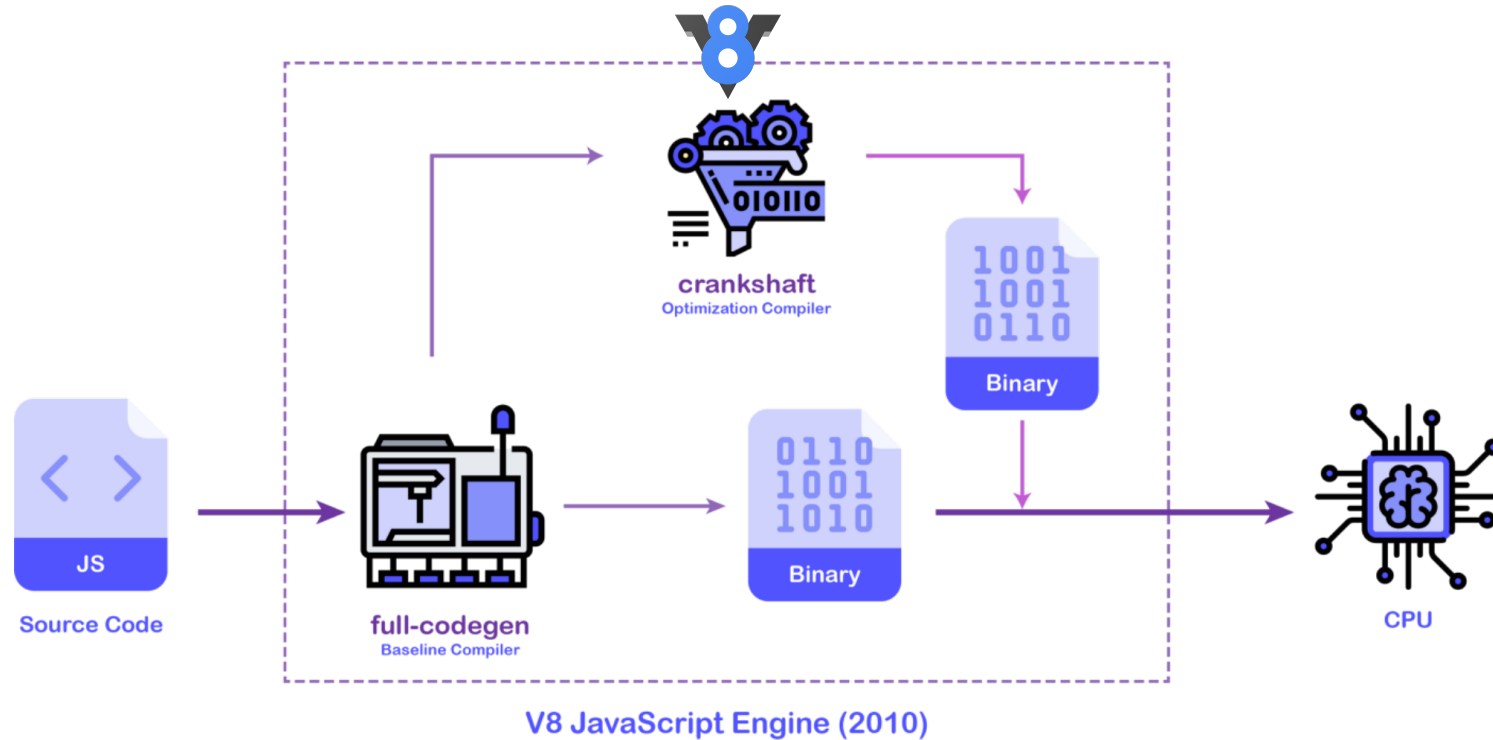


= JavaScriptCore

# Chrome V8

A JavaScript engine **executes** JavaScript code

Intermediate JS



# Intermediate JS

## Let's take a look

at JavaScript **runtime execution**  
inside the browser



# Intermediate JS

## Execution Context

**Everything** in JavaScript happens  
inside an execution context



# Intermediate JS

## Three fundamental parts of the JavaScript engine...

- Execution context
- **Memory** environment
- Thread of execution

[Javascript execution context video](#)





# Intermediate JS

**What about functions...**

...in the execution context?

```
const sumNum = 30;

const addOne = (num) => {
  const result = num + 1;
  return result;
};

console.log('Hello World');
const newNum = addOne(4);
```

### Global Execution Context

console.log(Hello World)  
addOne(4)

#### Local Execution Context

return

#### Local Memory

num: 4  
result: 5

### Global Memory

sumNum: 30

addOne: () => {}

newNum: 5

```
const first = 'Hello';  
const second = 'Dave';  
const allTogether = `${first} ${second}`;
```

↓ console.log(allTogether);

Global Execution Context

Global Memory

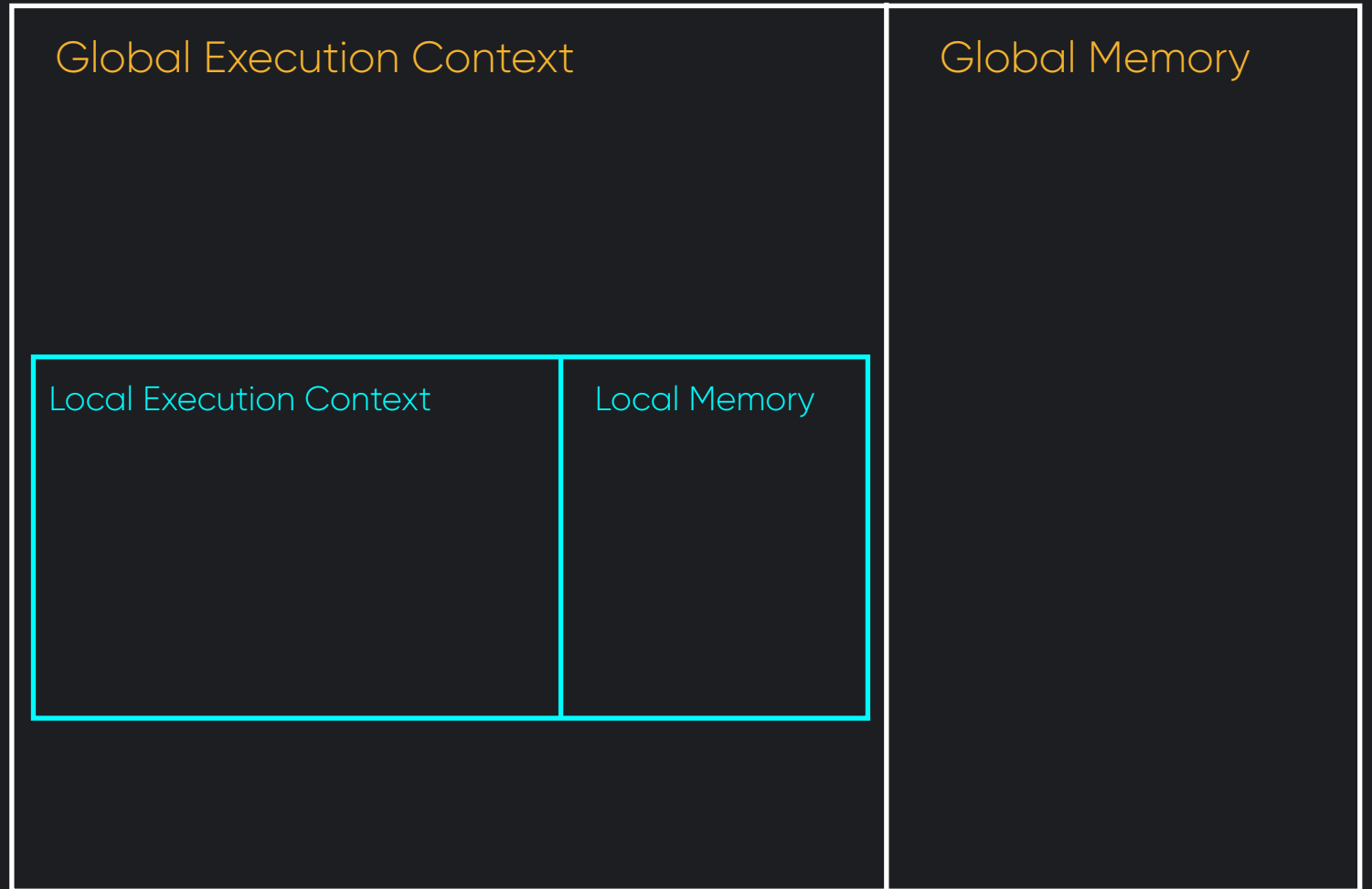
```
const words = ['hello', 'world'];  
const second = words[1];  
  
let name = 'Dave';  
name = 'Bob';  
  
const greet = () => {  
  return 'Hello';  
};
```



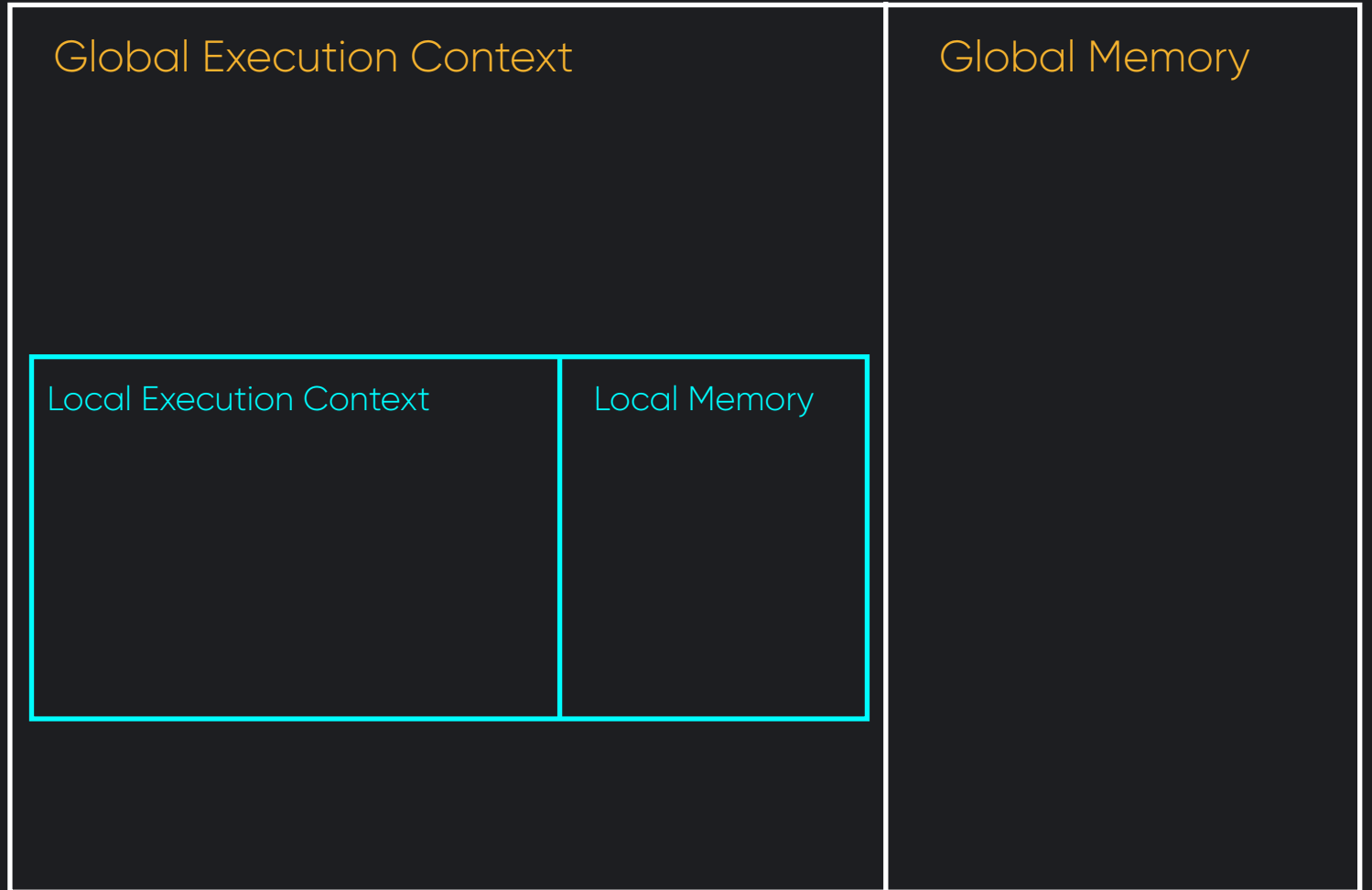
Global Execution Context

Global Memory

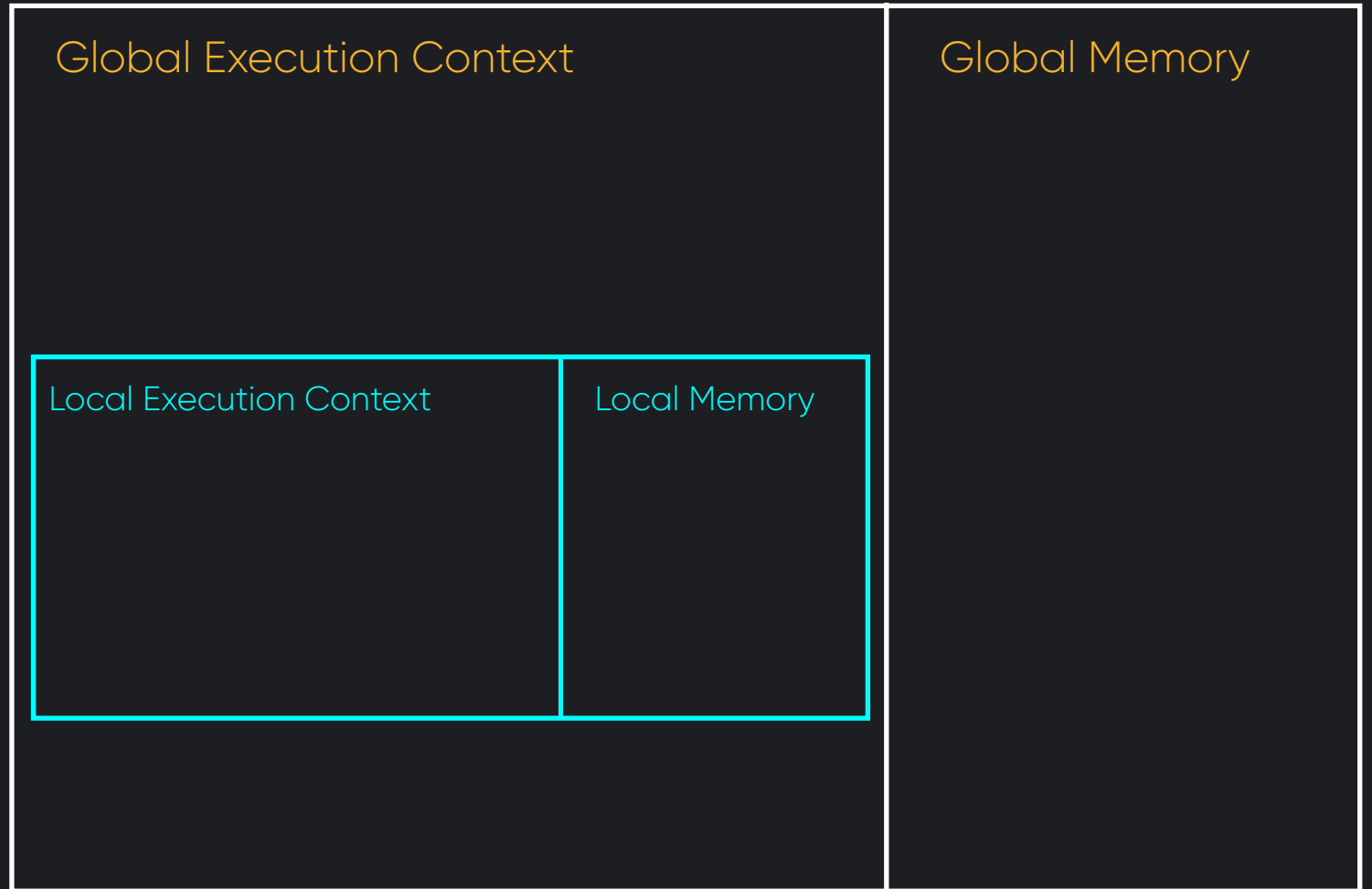
```
let name = 'Dave';  
  
const greet = (person) => {  
  return `Hello ${person}`;  
};  
  
console.log('I like pizza');  
const result = greet(name);  
  
console.log(result);
```



```
const multiply = (num1, num2) => {  
  const result = num1 * num2;  
};  
  
const newNum = multiply(2, 3);  
  
console.log(newNum);
```

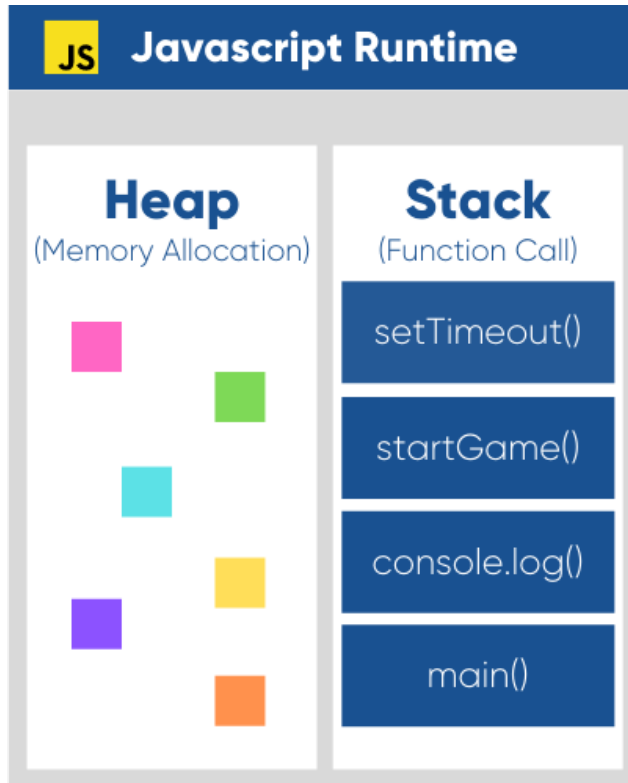


```
let name = 'John';  
  
function subtract(num1) {  
    return num1 - 4;  
}  
  
console.log(name);  
const result = subtract(4);  
  
console.log(result);
```



# The Memory Heap and Call Stack

## Intermediate JS



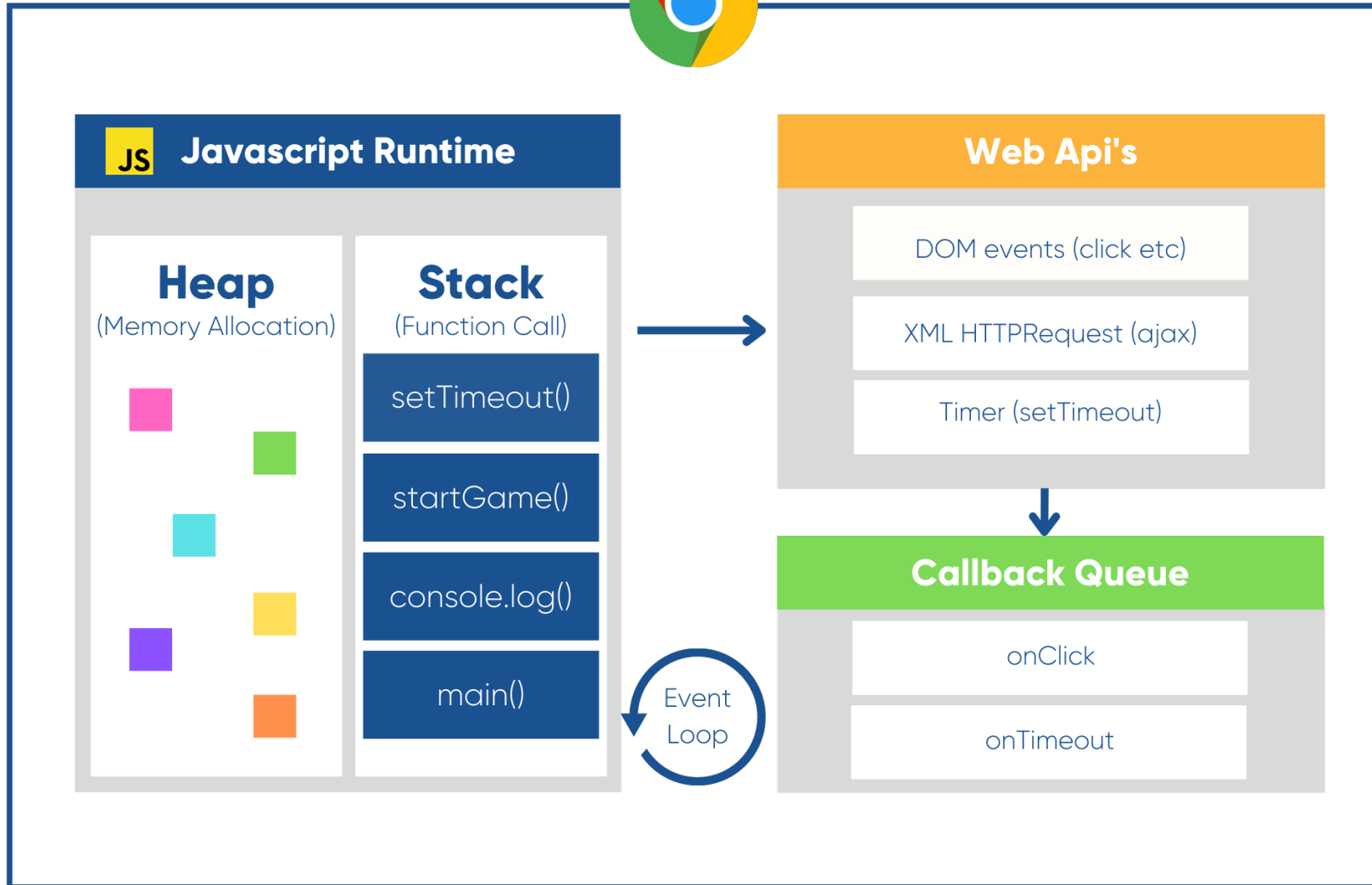
The call **stack** is **responsible for keeping the flow of execution** for our application. Without it, JavaScript wouldn't know what to call or when.

The **heap** is responsible for storing our data. This is where the **memory allocation** happens.

...Let's take a closer look at what happens in the browser



# Intermediate JS





# Intermediate JS

## JavaScript...

...is always synchronous and  
single-threaded

**...but what about pieces of code that take time to execute?**



# Intermediate JS

## Asynchronous functions

... such as `setTimeout()` are provided by browser webAPI's  
... we'll look at asynchronous functions in more detail later

[Javascript engine operation video](#)

# Learning Objectives

**To discover how a JavaScript engine operates**

**To be familiar with the JavaScript execution context**

**To explore JavaScript engine call stack, memory heap, event loop and callback queue**