ADVANCED
HOW JAVASCRIPT WORK

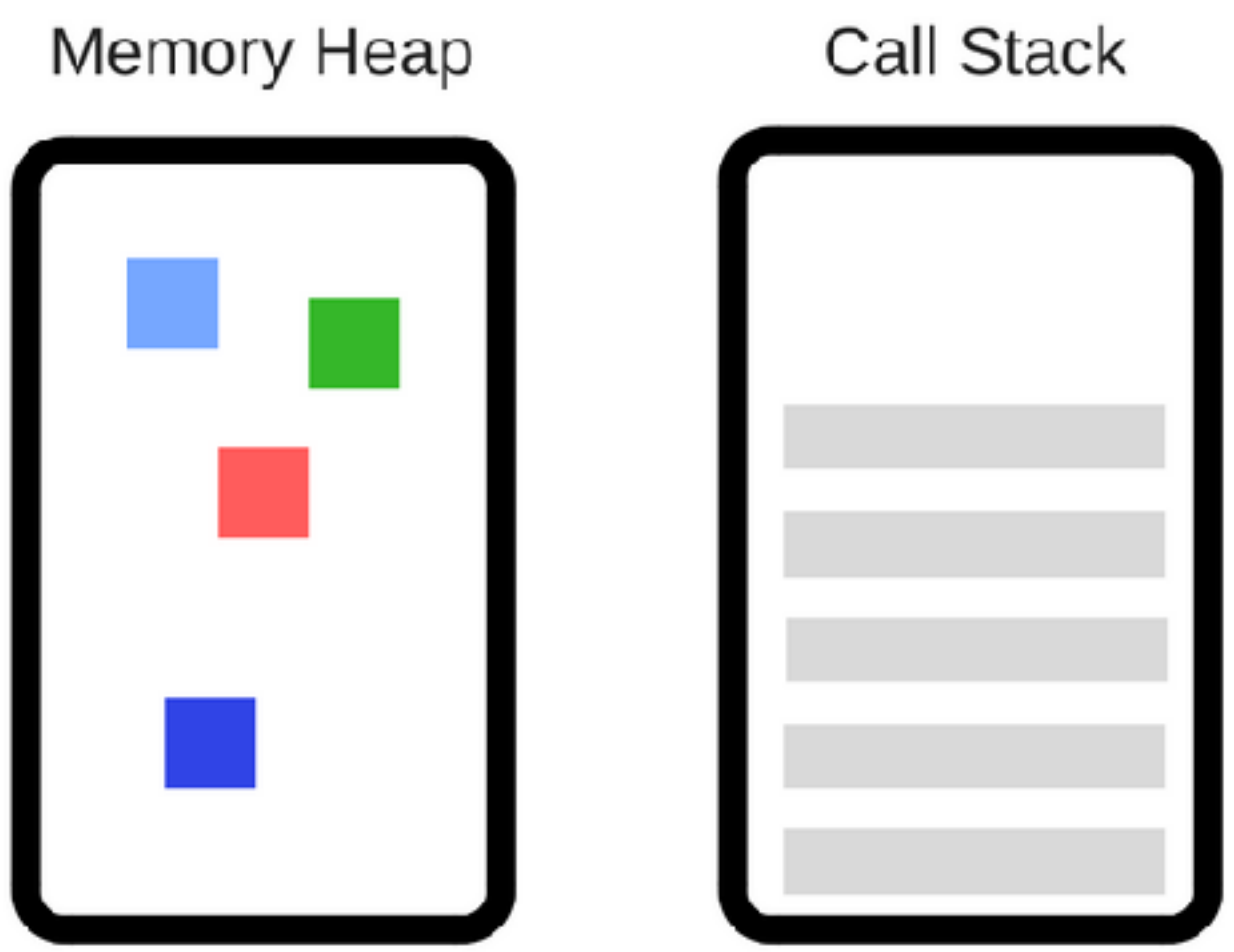
TABLE OF CONTENT

- Engine, runtime
- Call Stacks
- Event Loop
- Javascript Asynchronus
- Memory Management
- Browser and debug tool.

JS ENGINE

- Memory Heap – this is where the memory allocation happens

- Call Stack – this is where your stack frames are as your code executes

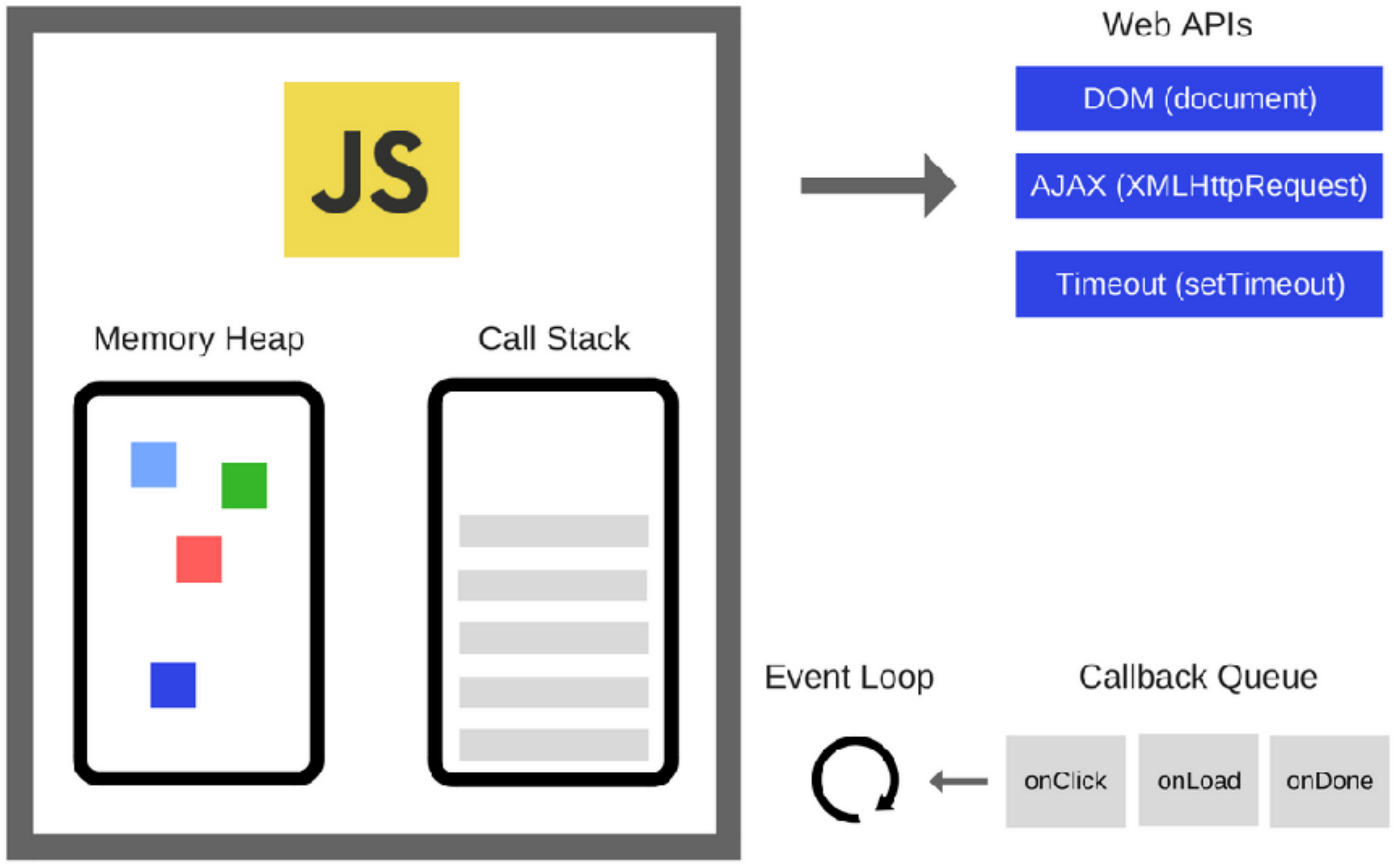


JS RUNTIME

- setTimeout()
- console.log()
- document.getElementById()
- .....

You think they are javascript,

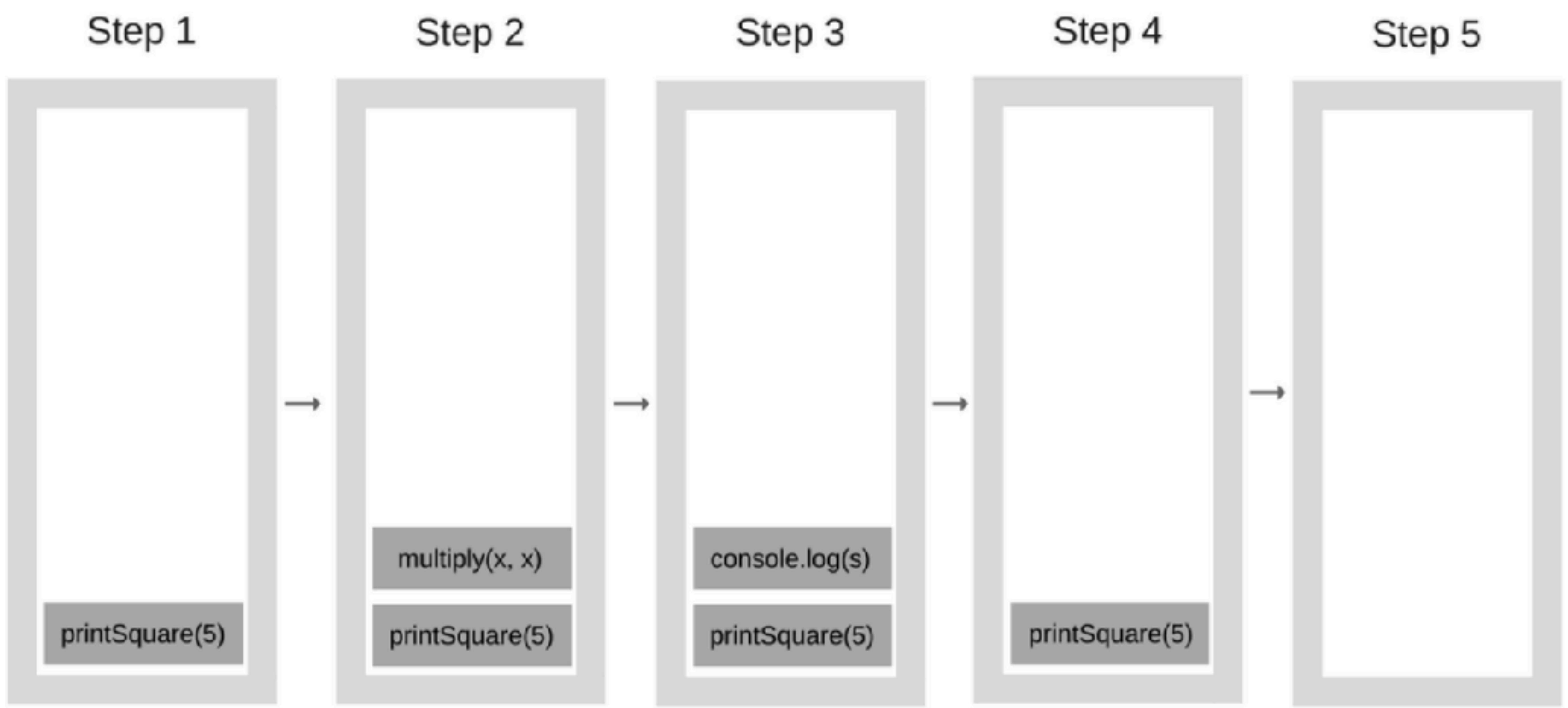
but they are not.



THE CALL STACK

- JavaScript: **single-threaded === single Call Stack**
- The Call Stack - a data structure => where we are in the program.

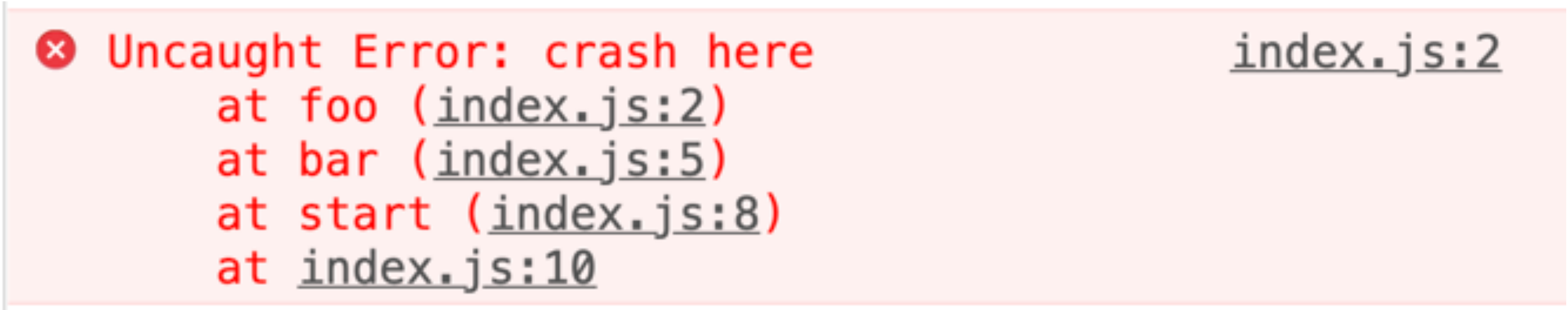
```
function multiply(x, y) {
    return x * y;
}
function printSquare(x) {
    var s = multiply(x, x);
    console.log(s);
}
printSquare(5);
```



CALL STACK

- Stack Trace

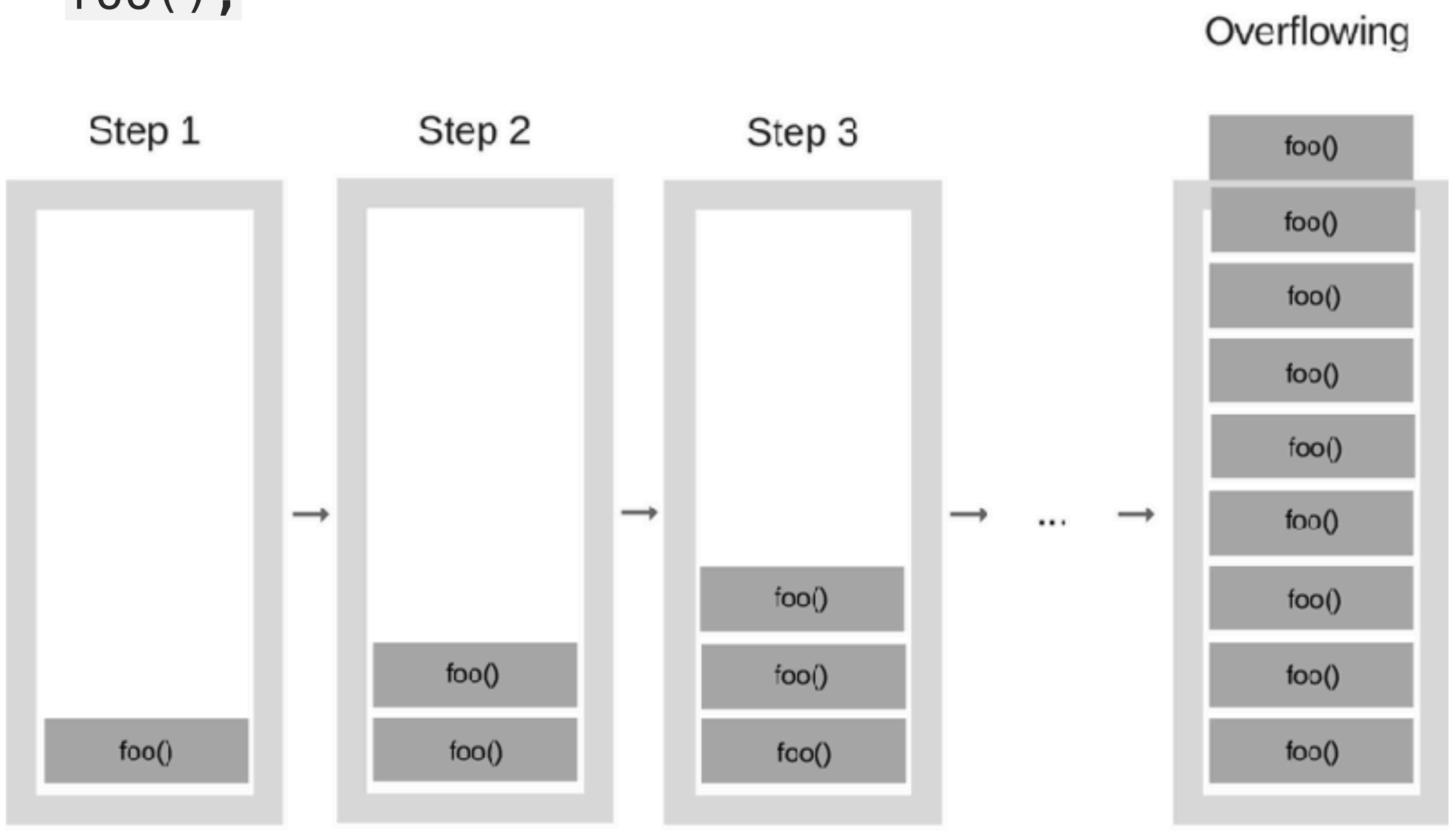
```
function foo() {
    throw new Error(‘crashed here :)');
}
function bar() {
    foo();
}
function start() {
    bar();
}
start();
```



```
Uncaught Error: crash here                    index.js:2
    at foo (index.js:2)
    at bar (index.js:5)
    at start (index.js:8)
    at index.js:10
```

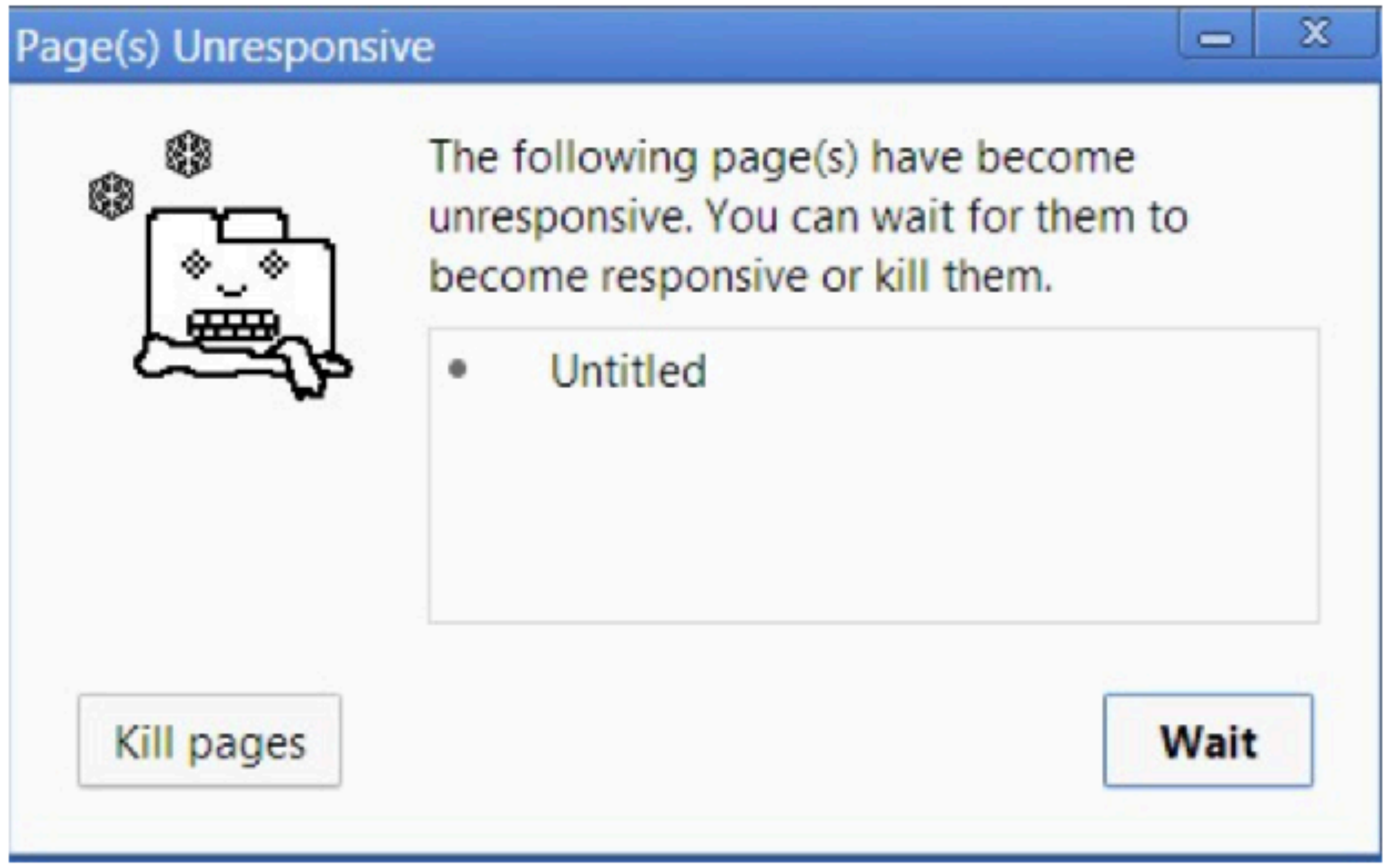
- Blowing the stack

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



CONCURRENCY

- What happens when a function calls take too much time to proceed?
  - The browser can not do anything else – it is blocked.
  - Most browsers take action by raising an error



Page(s) Unresponsive

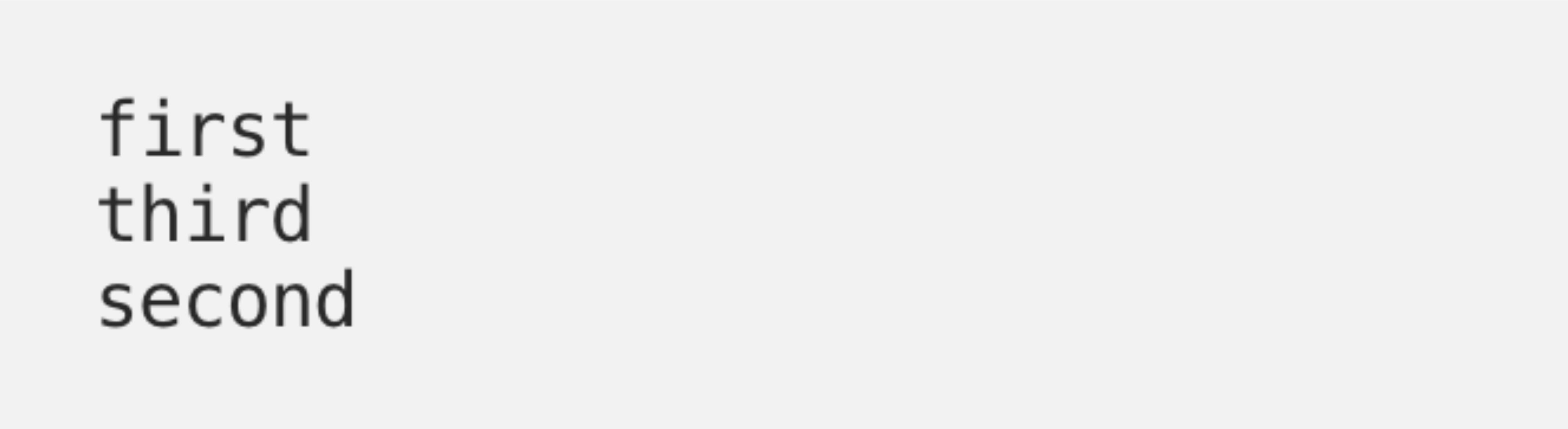
The following page(s) have become unresponsive. You can wait for them to become responsive or kill them.

- Untitled

SETTIMEOUT

```
function first() {
    console.log('first');
}
function second() {
    console.log('second');
}
function third() {
    console.log('third');
}
```

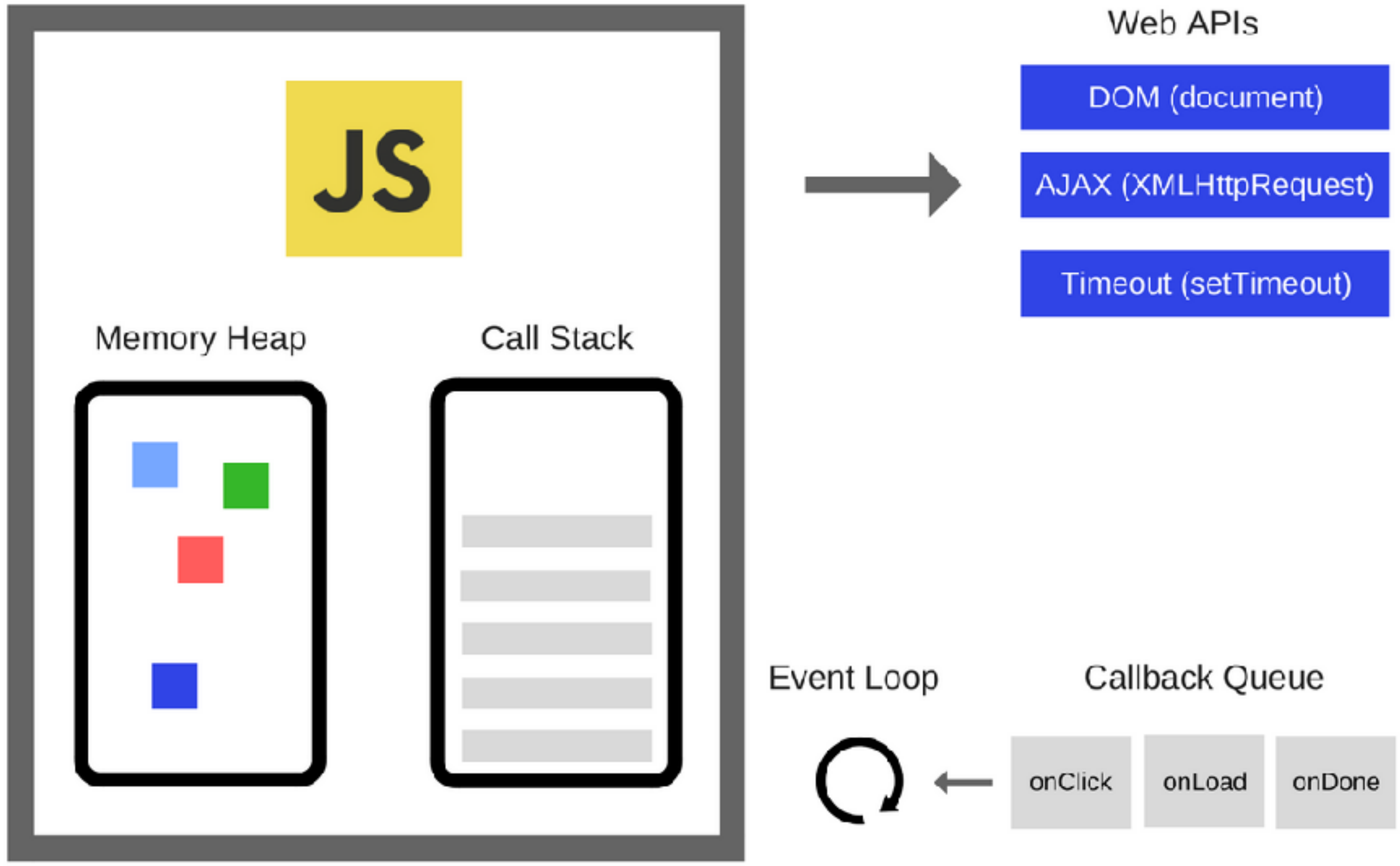
```
first();
setTimeout(second, 1000); // Invoke after 1000ms
third();
```



```
first
third
second
```

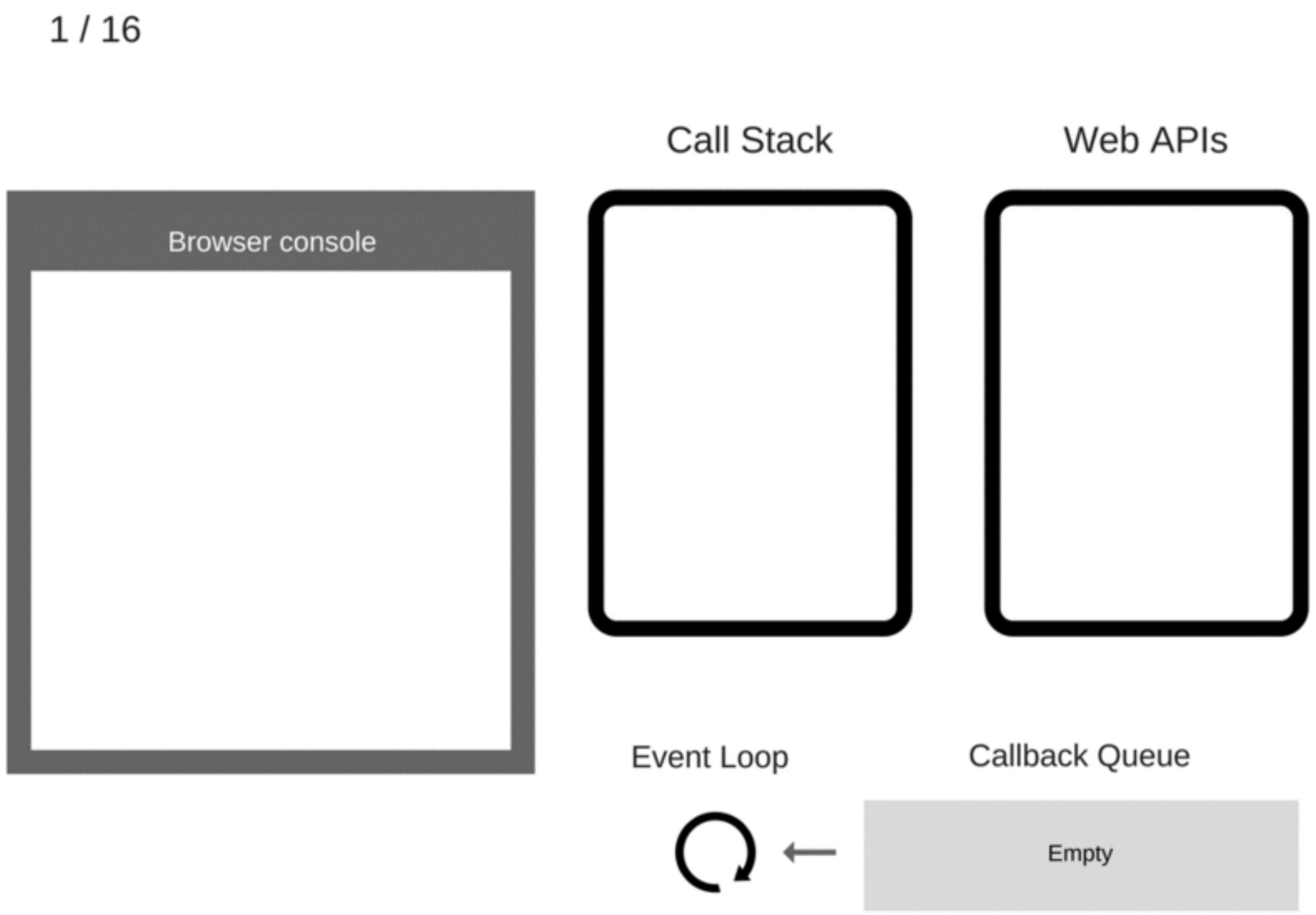
EVENT LOOP

- The JavaScript just do the synchronous code.
- JS Engine doesn't run in isolation – it runs inside a *hosting* environment.
- **the event loop:** handles the execution of multiple chunks of your program over time
- monitor the Call Stack and the Callback Queue.

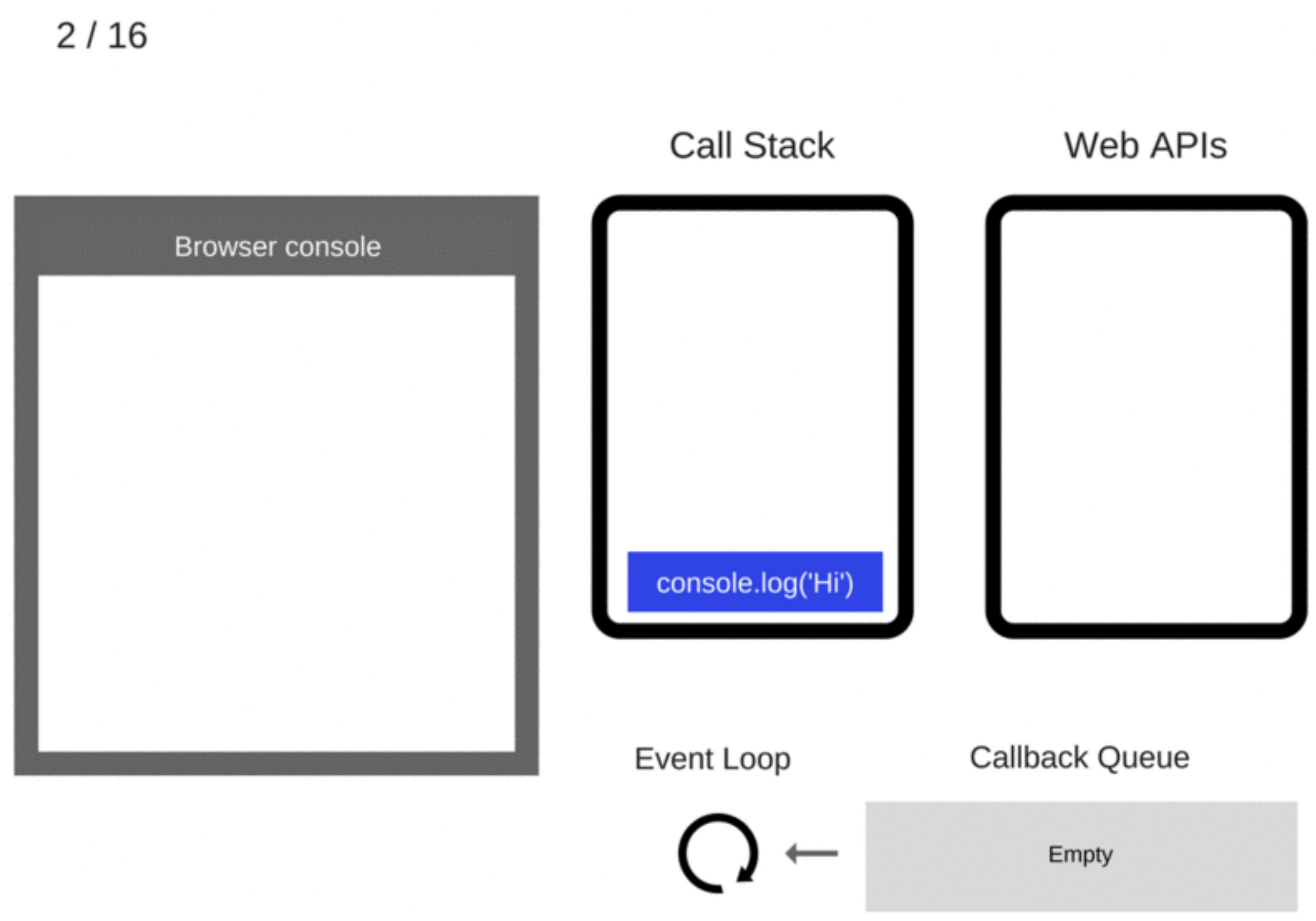


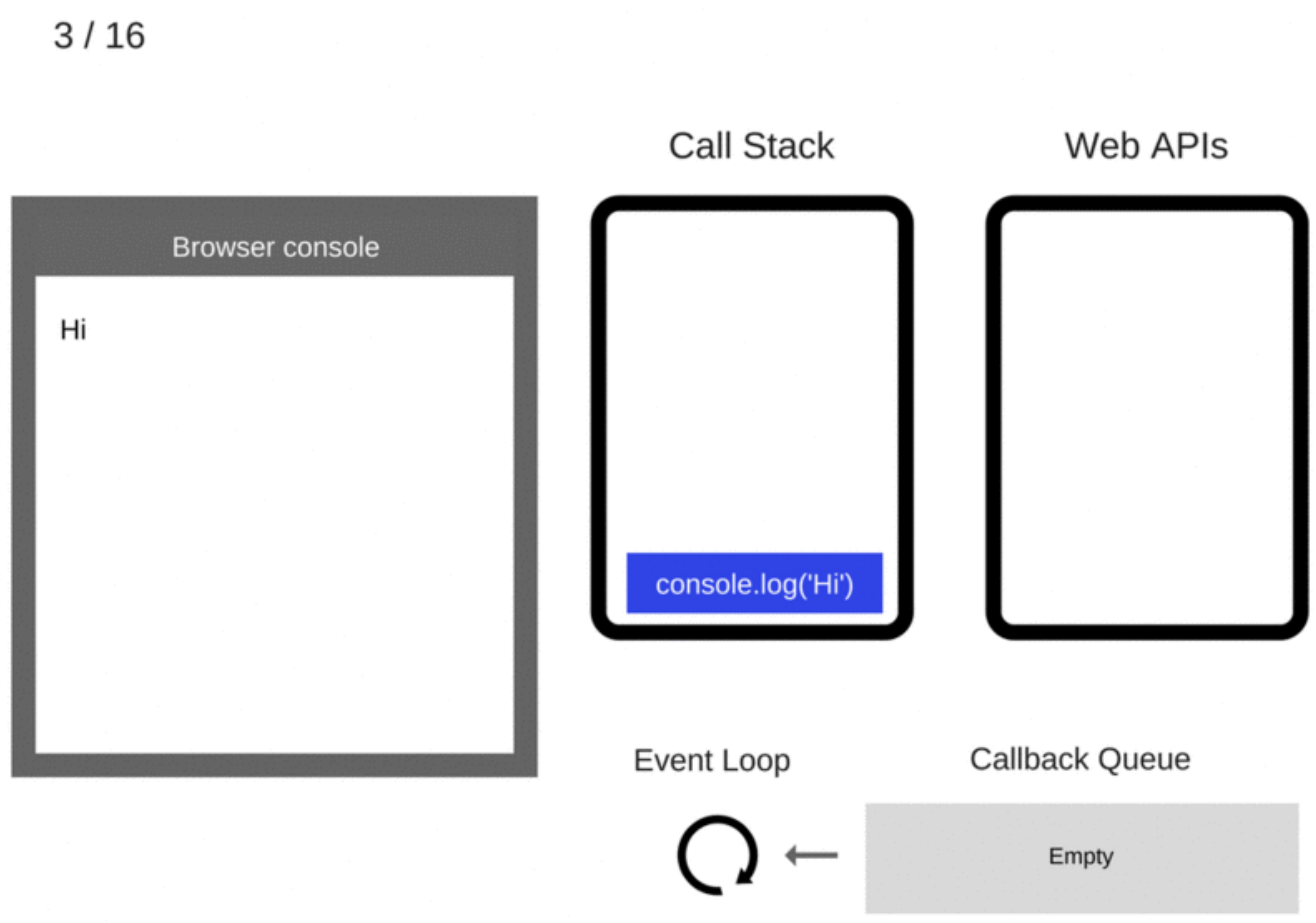
SEE WHAT HAPPEN WITH A BASIC SETTIMOUT

```
console.log('Hi');
setTimeout(function cb1() {
    console.log('cb1');
}, 5000);
console.log('Bye');
```



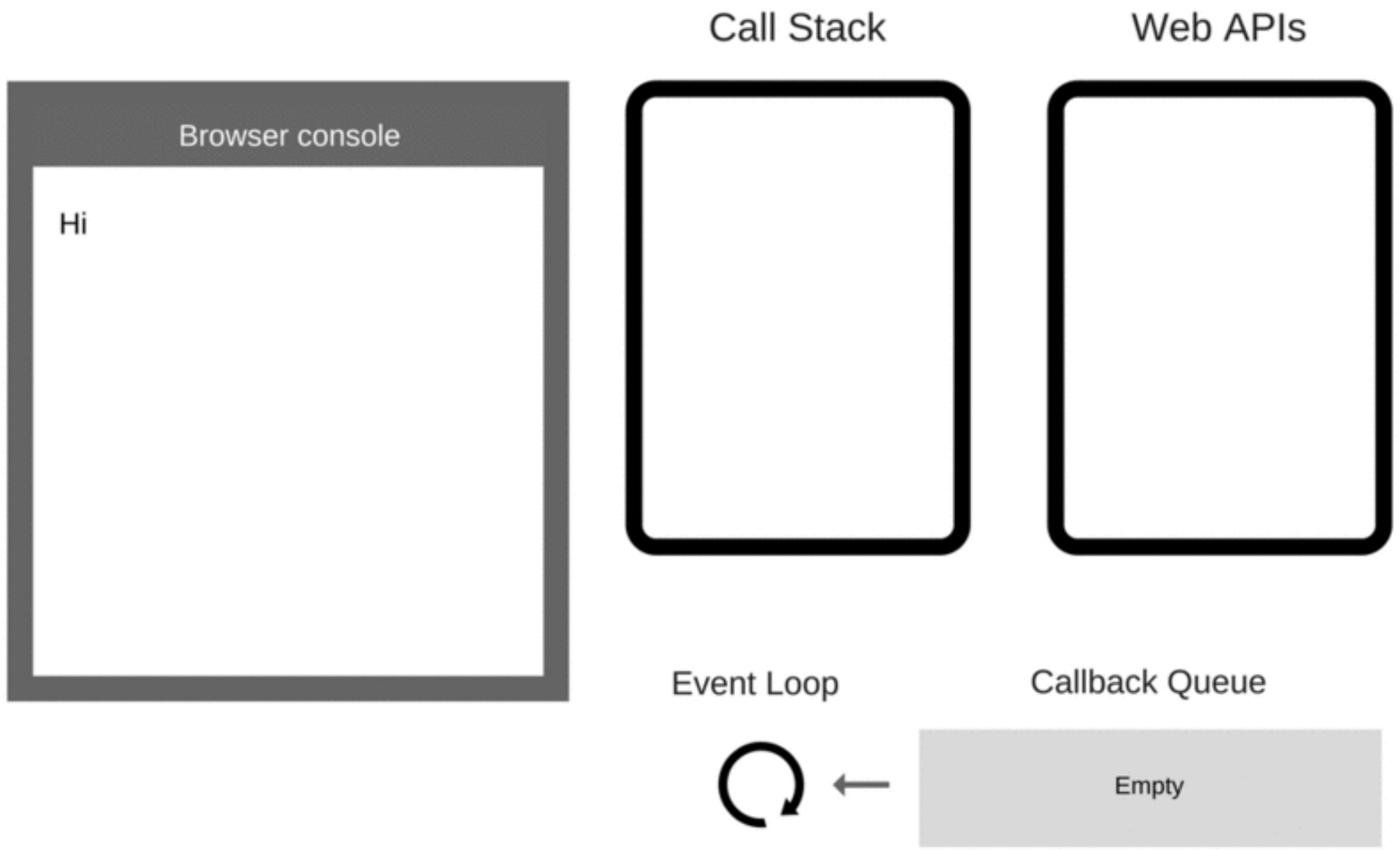
SEE WHAT HAPPEN WITH A BASIC SETTIMOUT



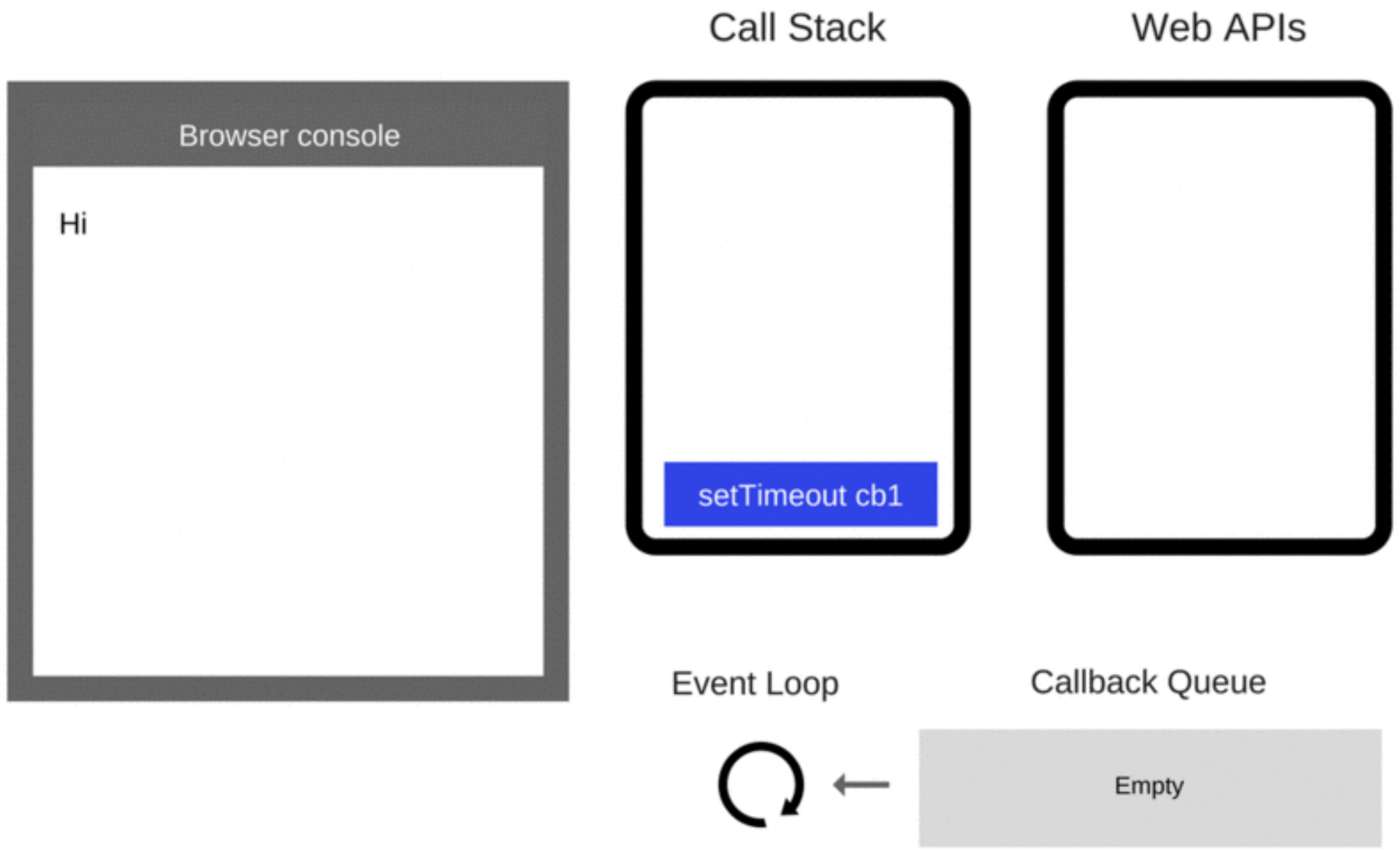


SEE WHAT HAPPEN WITH A BASIC SETTIMOUT

4 / 16

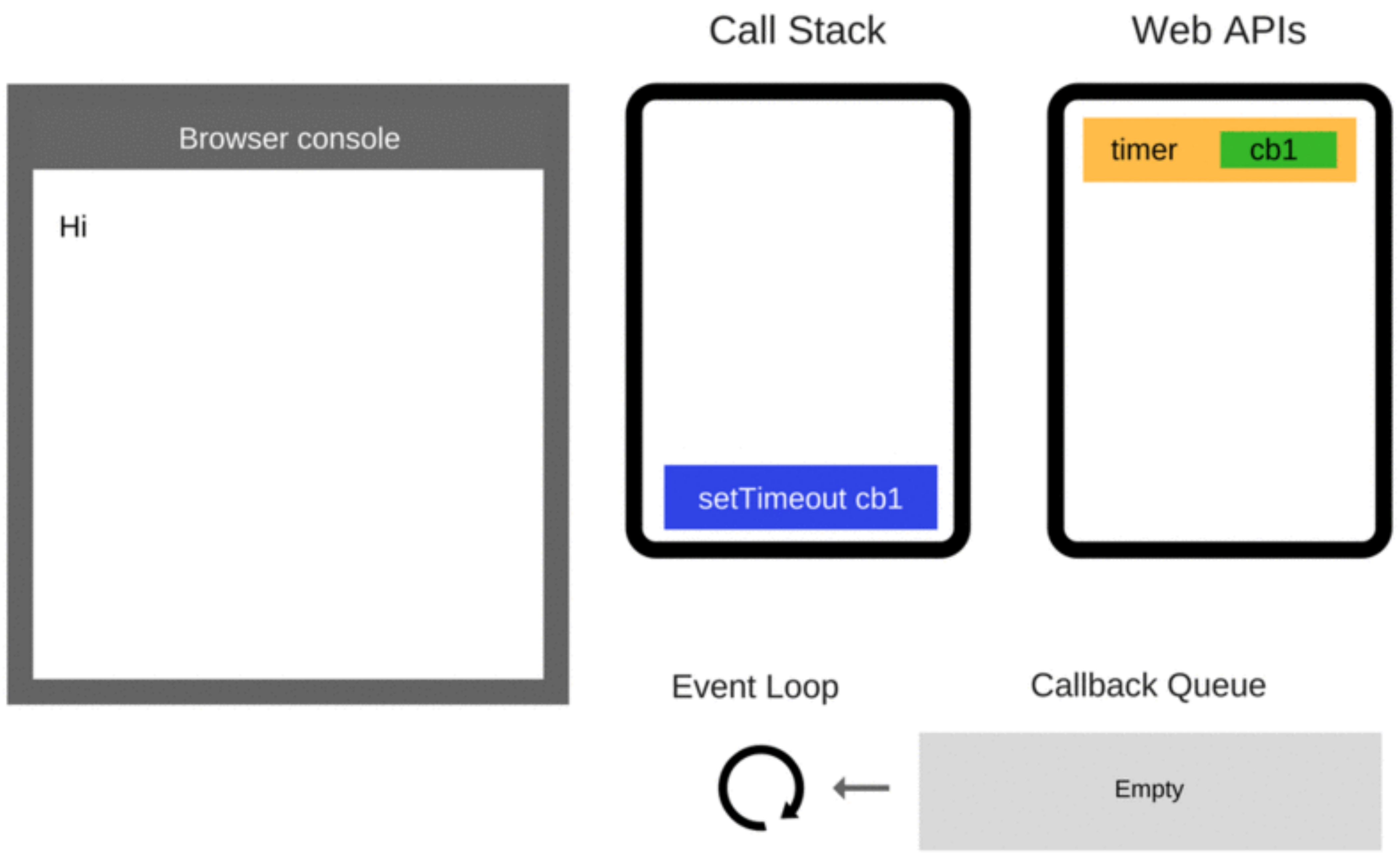


5 / 16



SEE WHAT HAPPEN WITH A BASIC SETTIMOUT

6 / 16



Call Stack

Web APIs

Browser console

Hi

setTimeout cb1

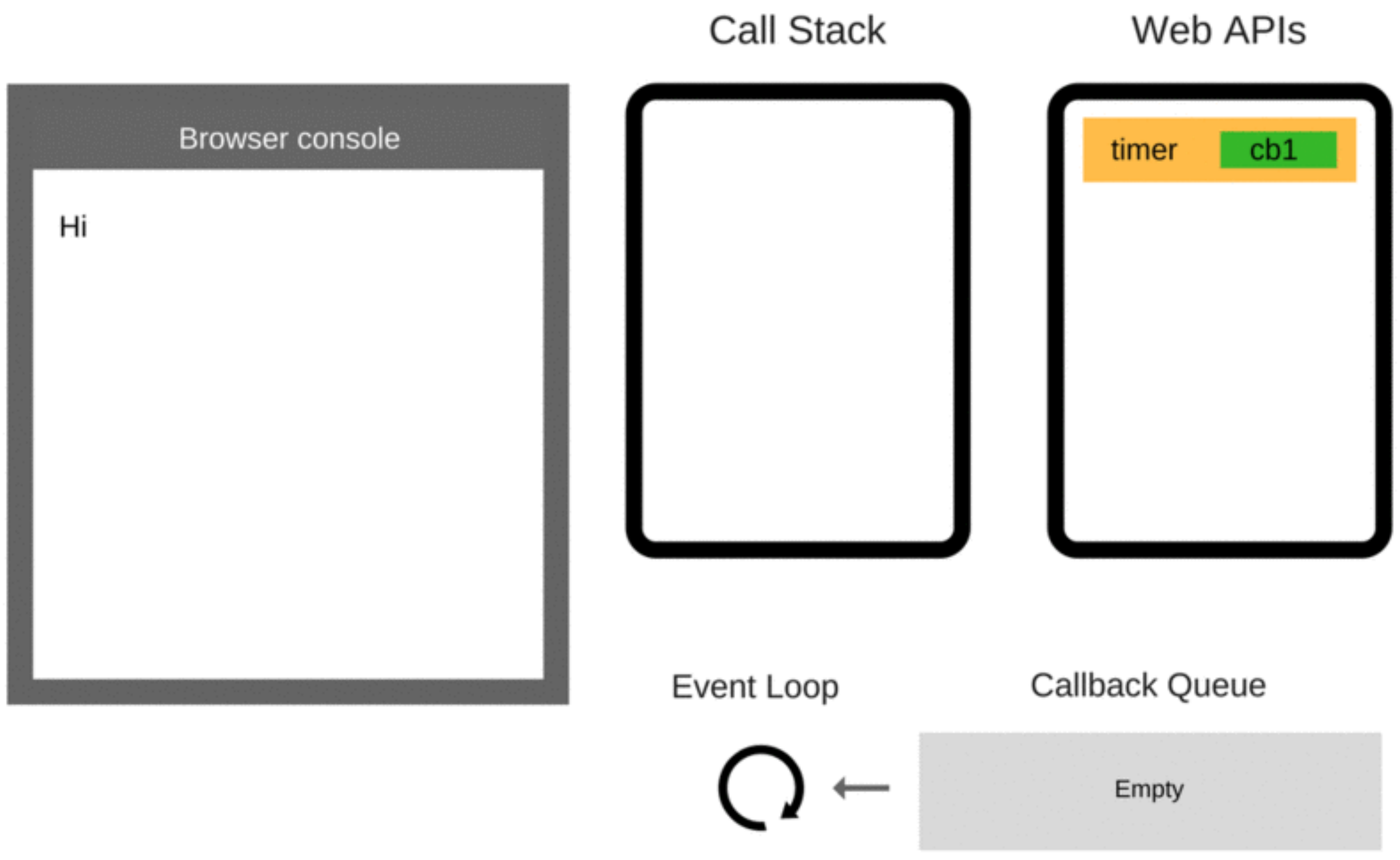
timer cb1

Event Loop

Callback Queue

Empty

7 / 16



Call Stack

Web APIs

Browser console

Hi

timer cb1

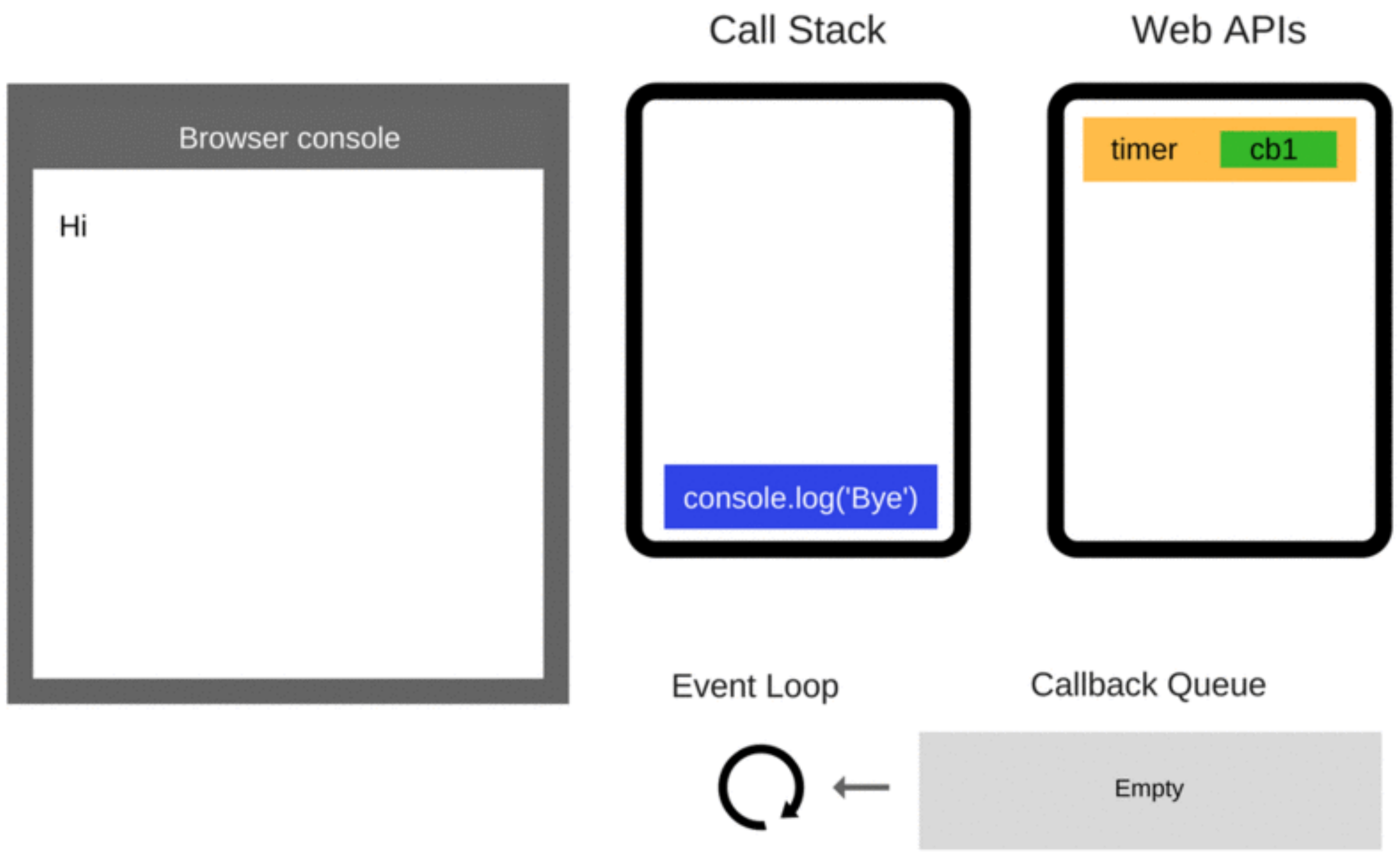
Event Loop

Callback Queue

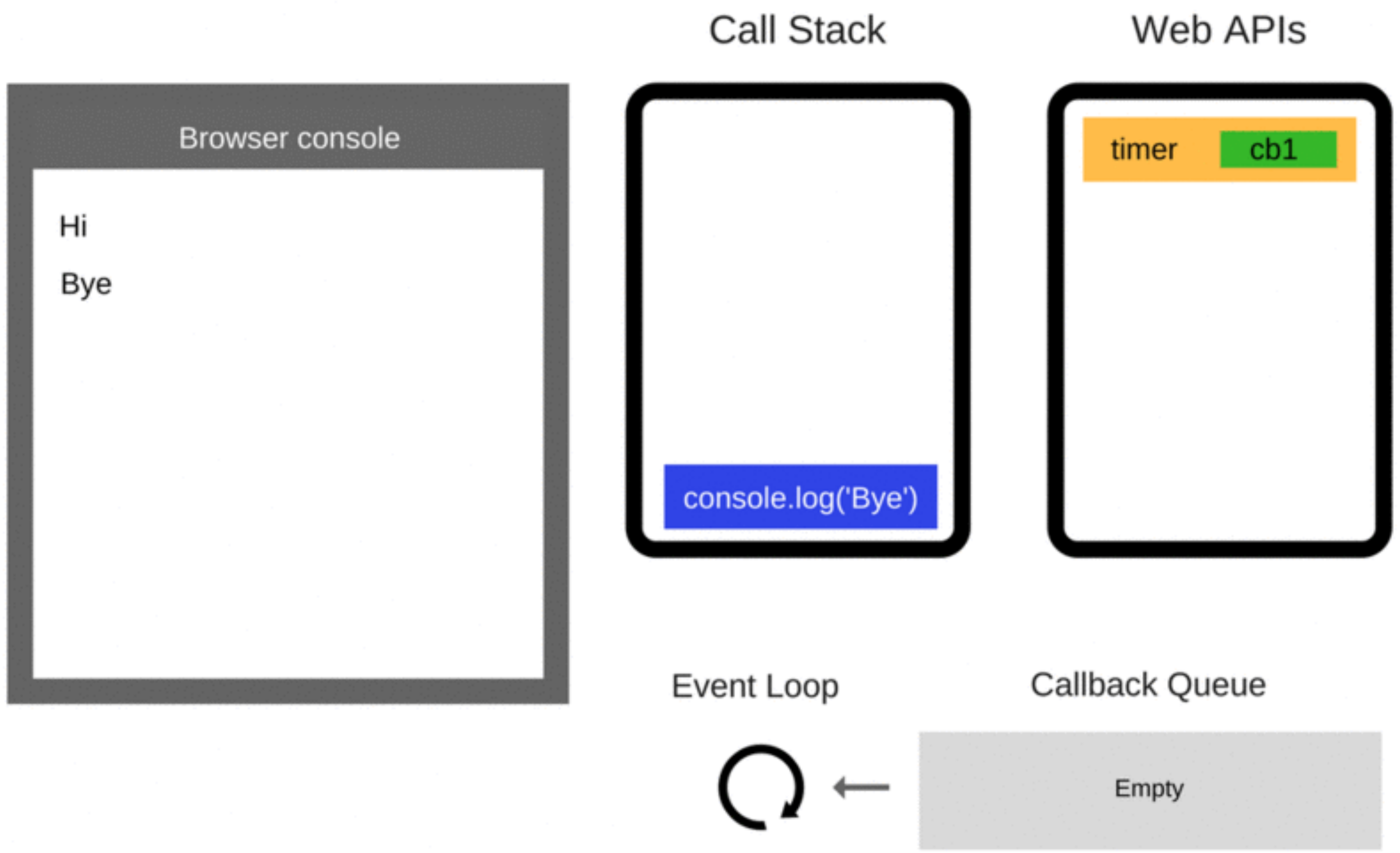
Empty

SEE WHAT HAPPEN WITH A BASIC SETTIMOUT

8 / 16

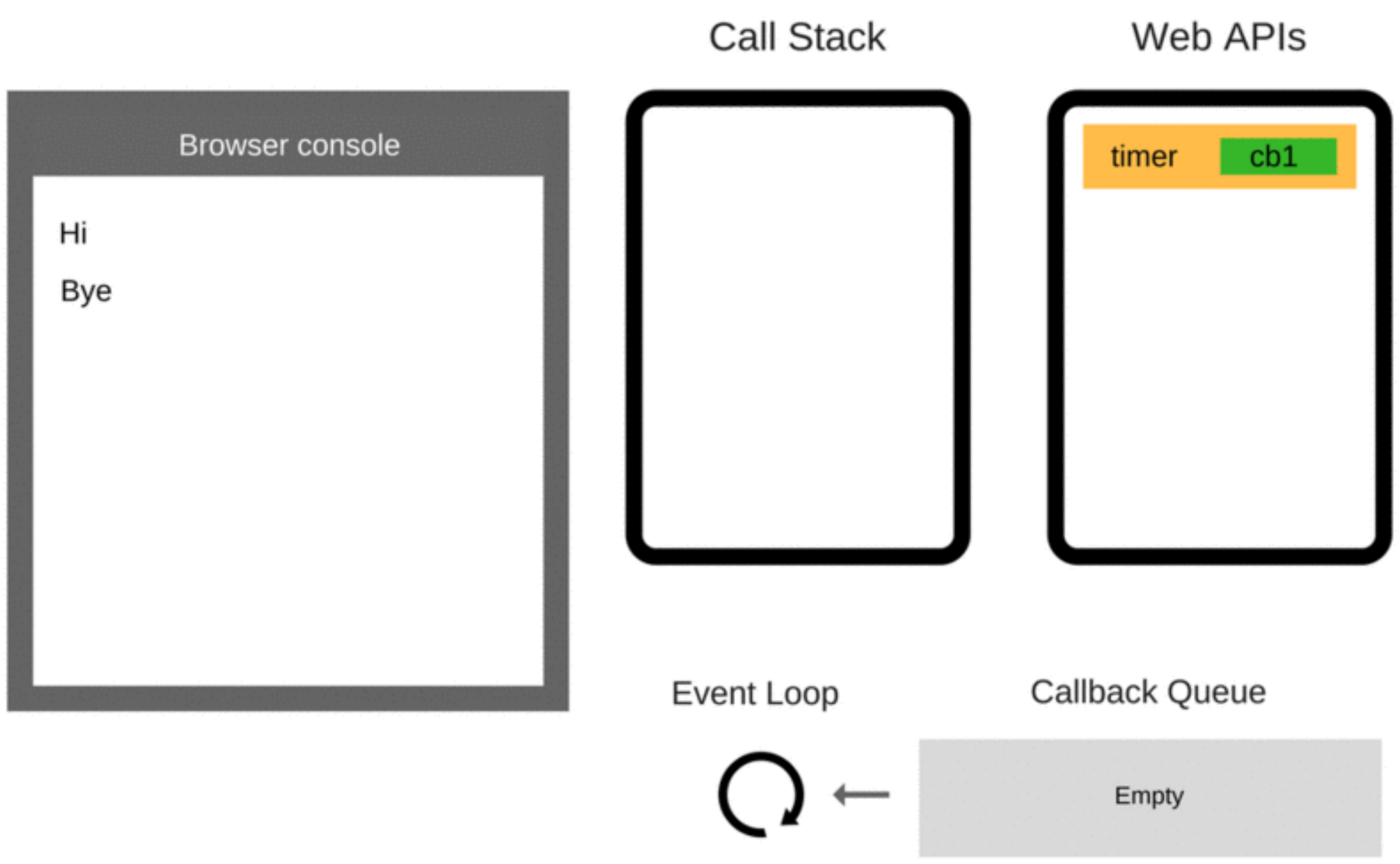


9 / 16

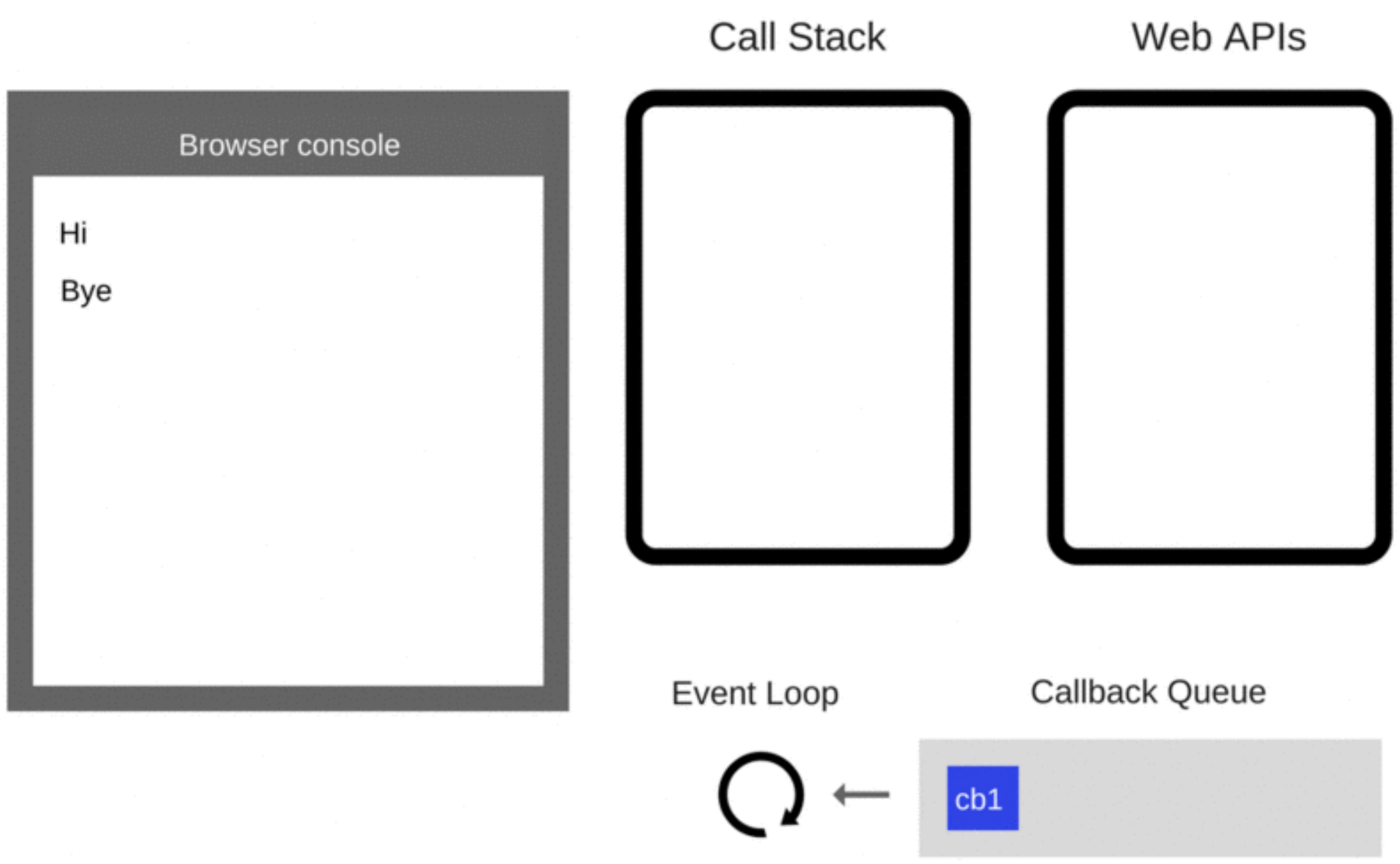


SEE WHAT HAPPEN WITH A BASIC SETTIMOUT

10 / 16

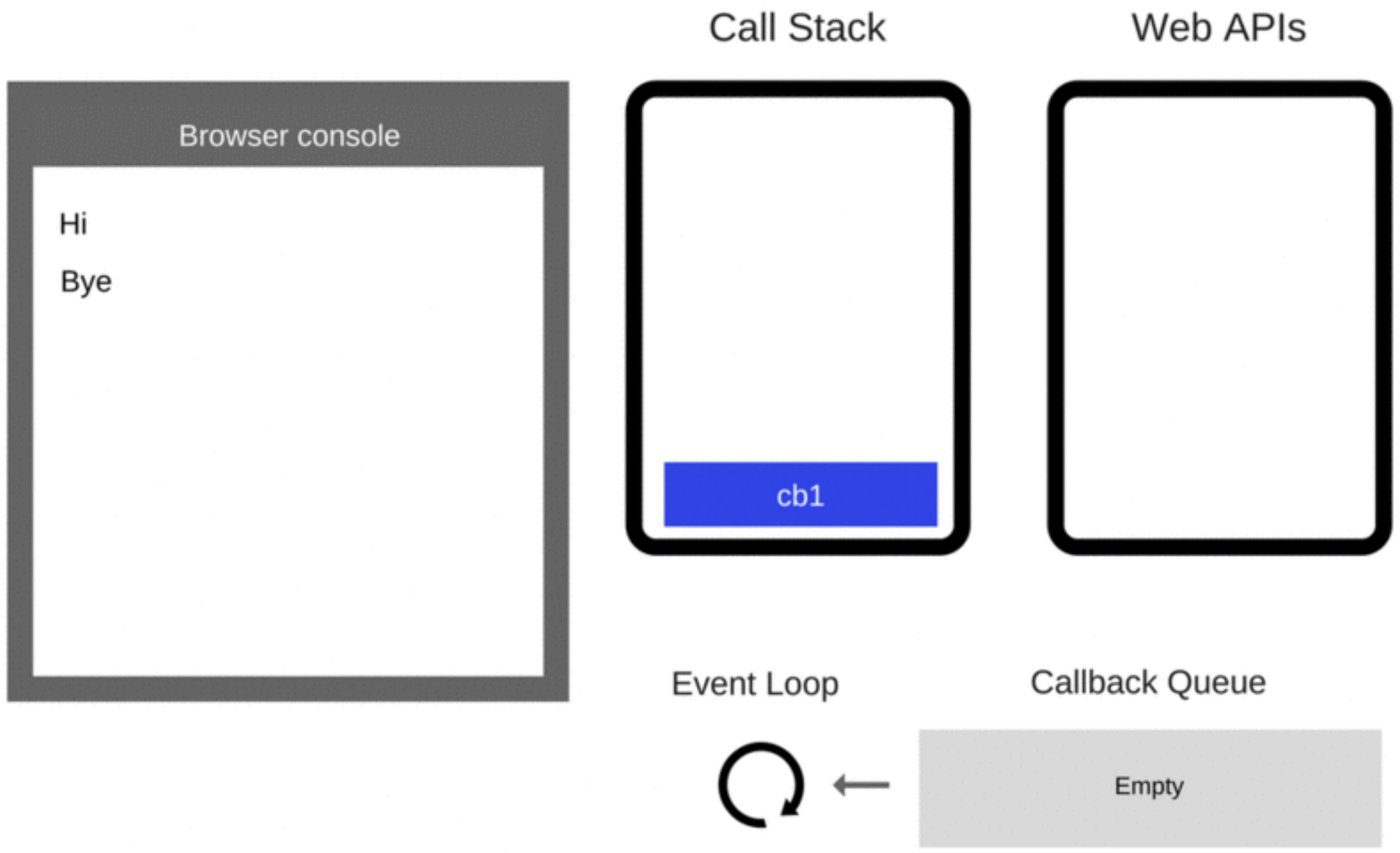


11 / 16



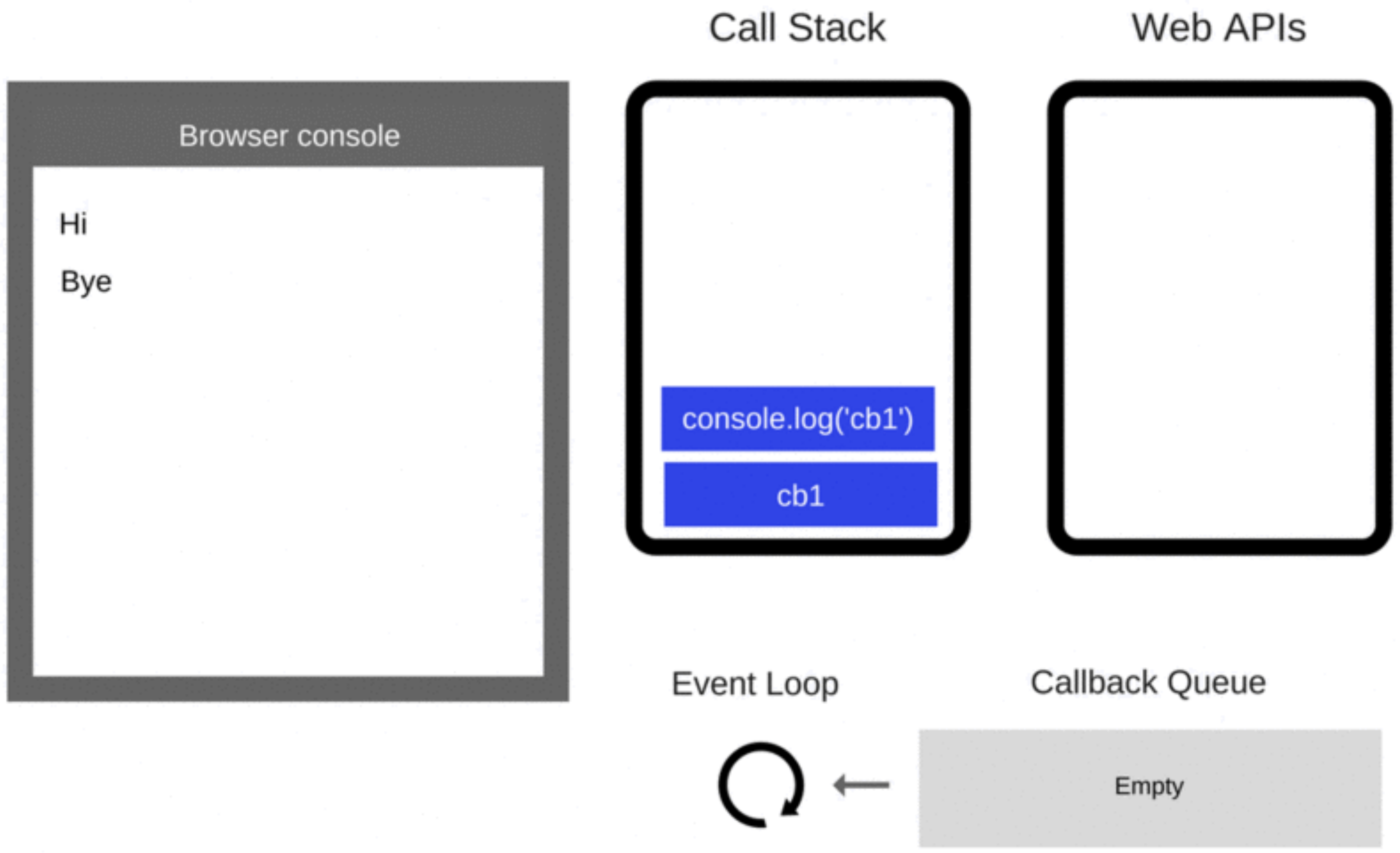
SEE WHAT HAPPEN WITH A BASIC SETTIMOUT

12 / 16



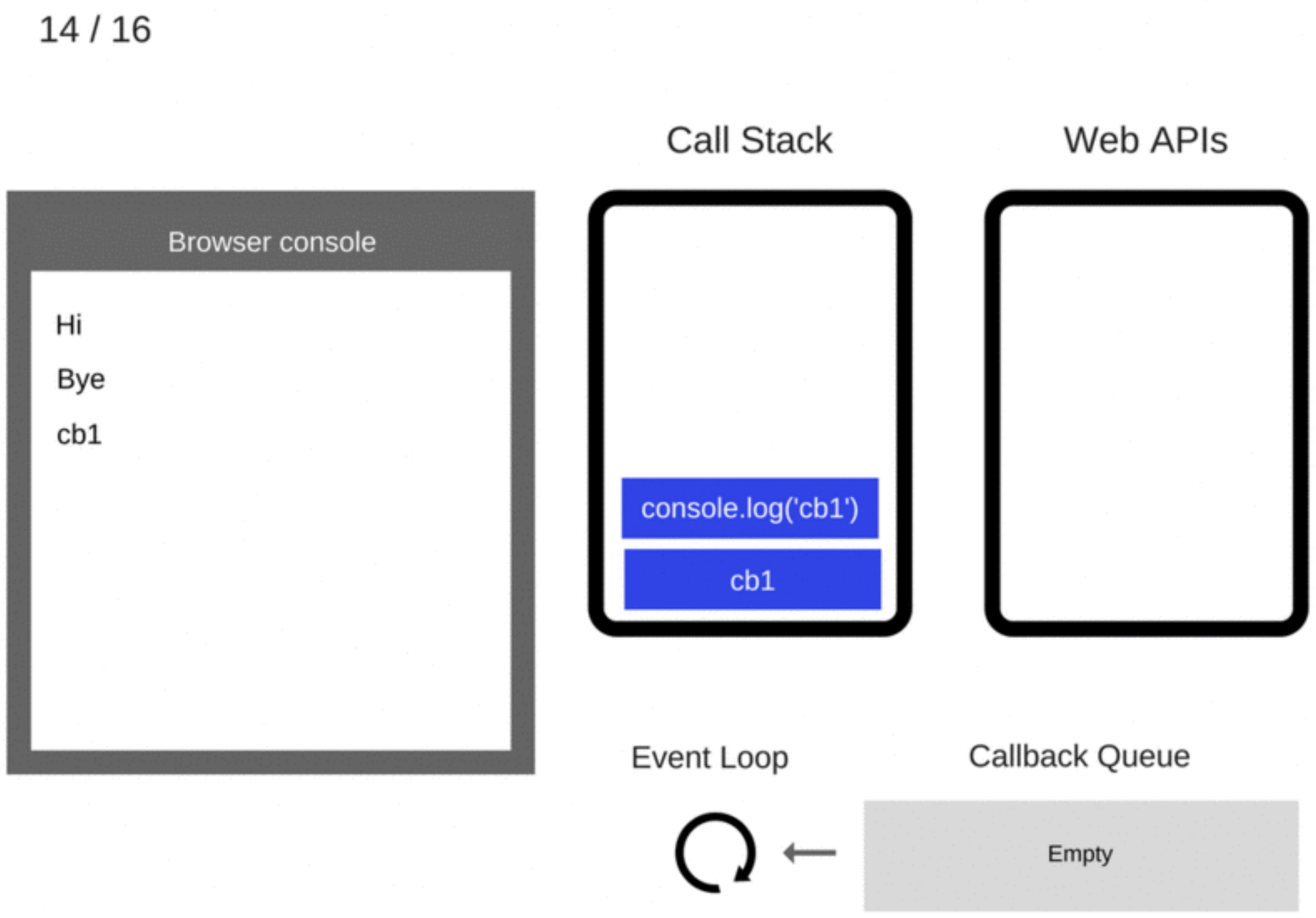
```
Hi
Bye
```

13 / 16

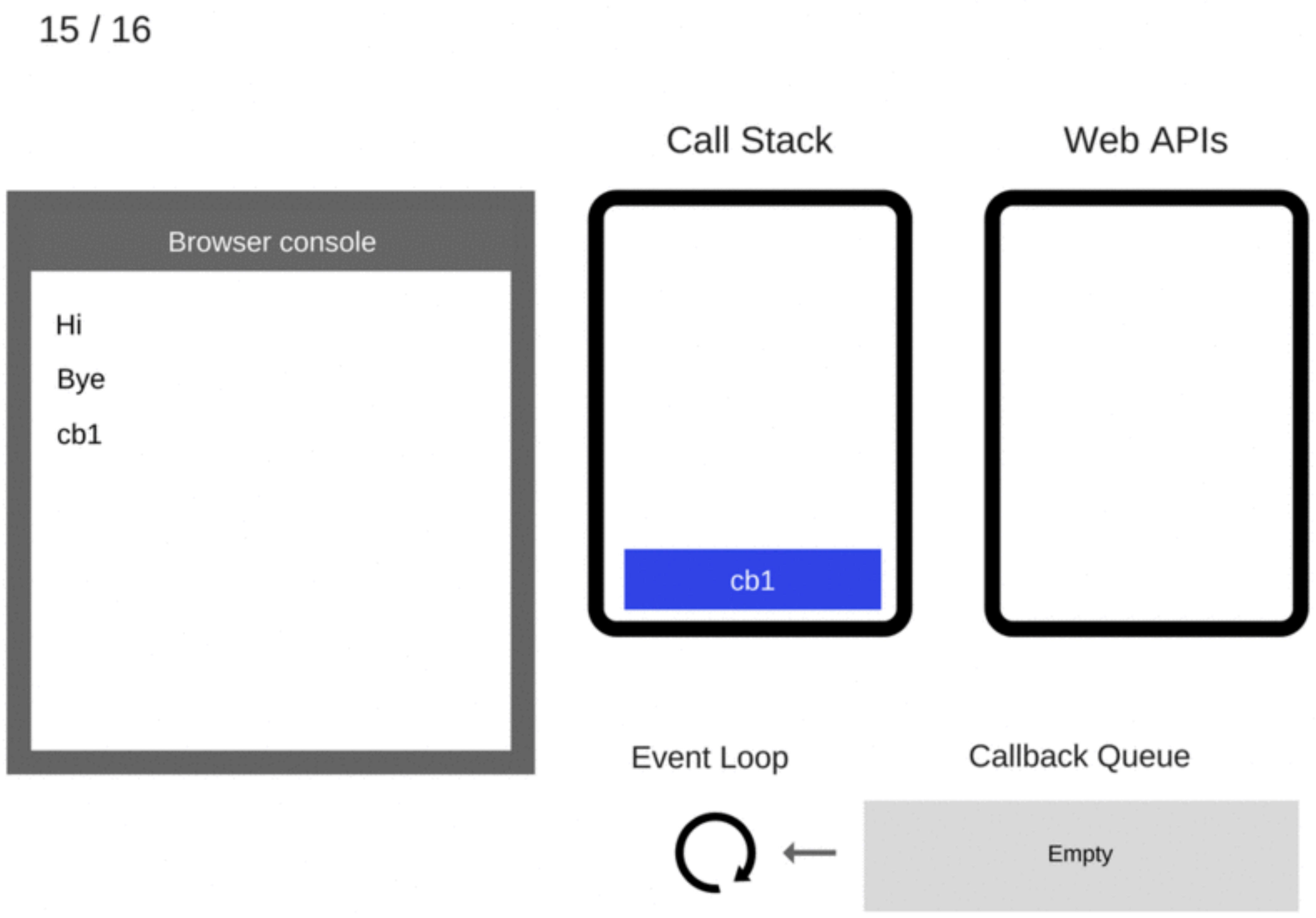


```
Hi
Bye
```

SEE WHAT HAPPEN WITH A BASIC SETTIMOUT

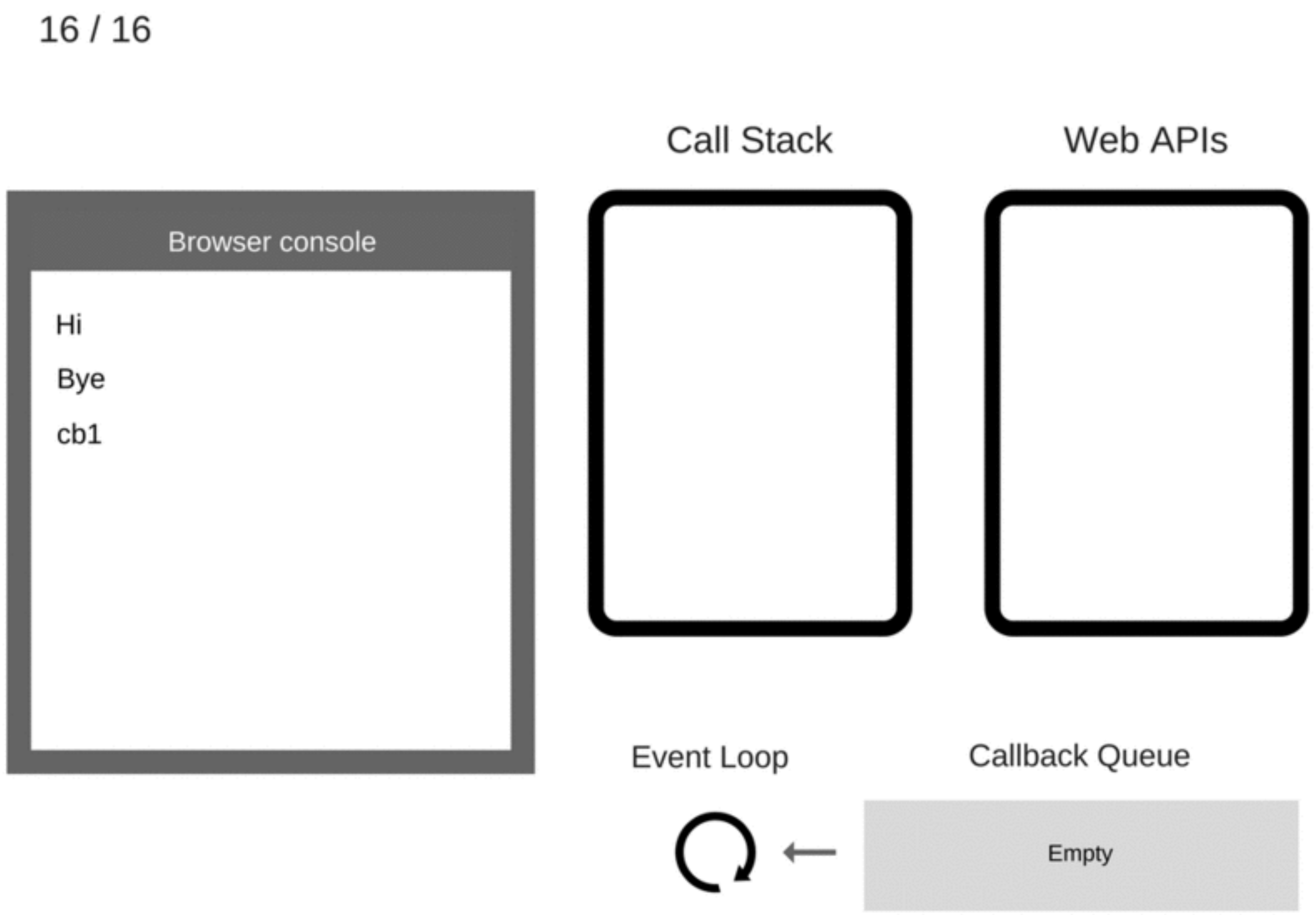


```
Hi
Bye
cb1
```

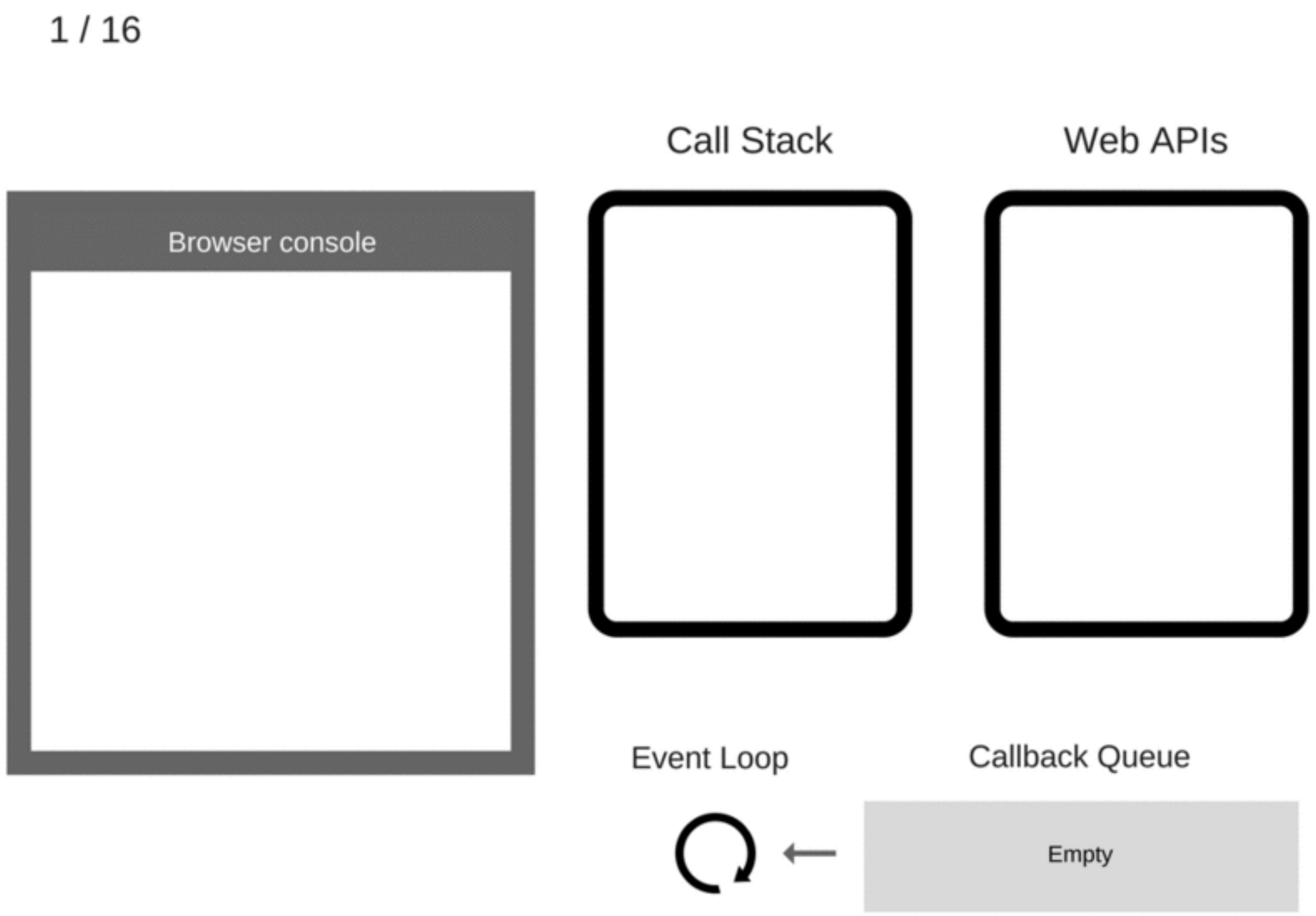


```
Hi
Bye
cb1
```

SEE WHAT HAPPEN WITH A BASIC SETTIMOUT



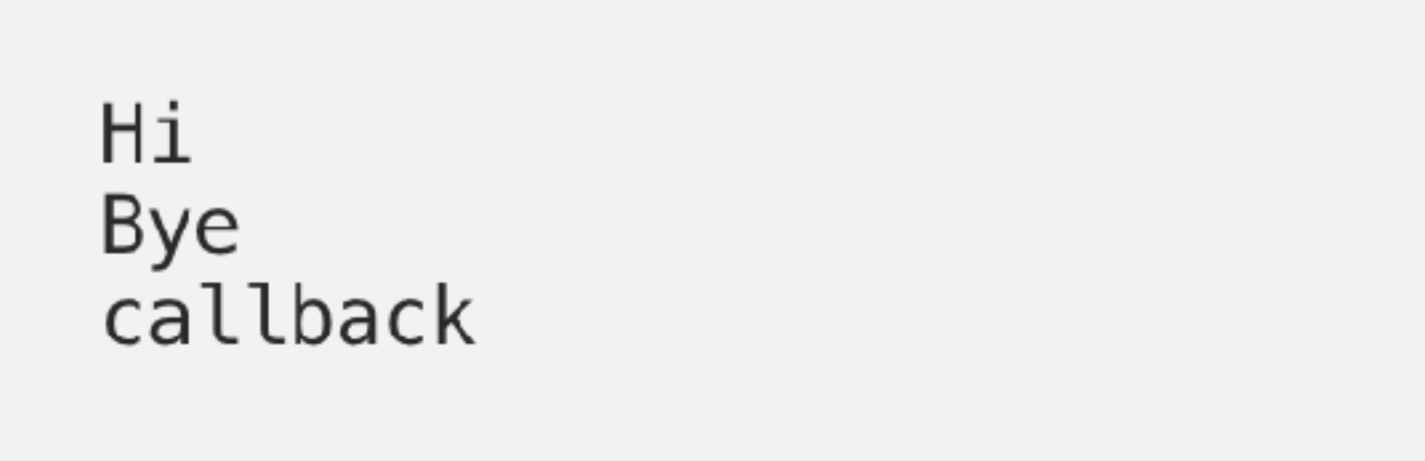
```
Hi
Bye
cb1
```



HOW SETTIMEOUT WORK?

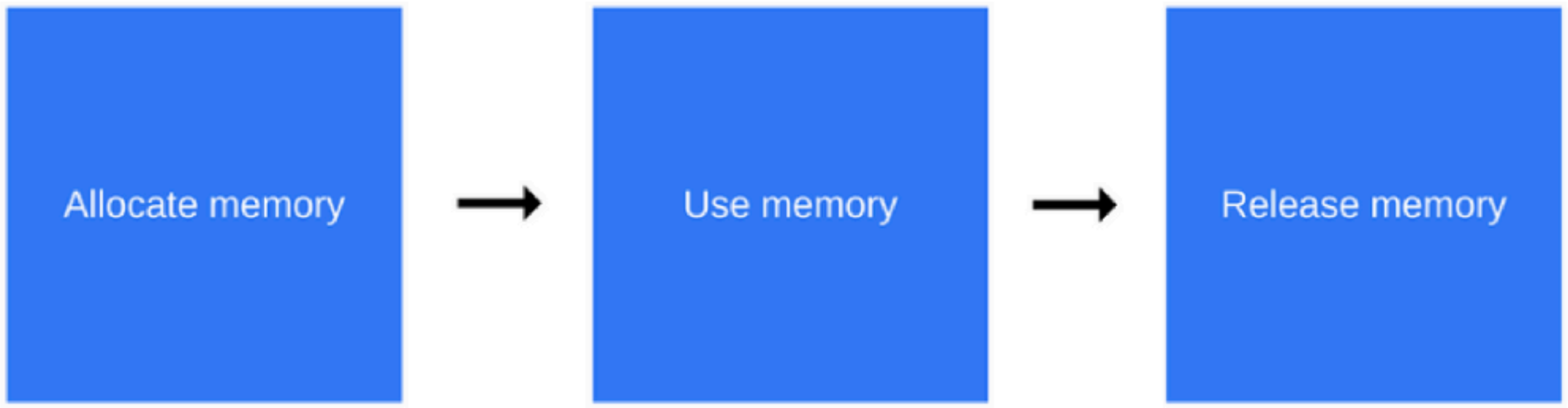
- 1. setTimeout do not put callback to the event loop queue.
- 2. It sets up a timer.
- 3. When the timer expires, the environment places your callback into the event loop
- 4. The callback will be picked up and execute.

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

setTimeout(function() {

    console.log('callback');

}, 0);

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



```
Hi
Bye
callback
```

MEMORY MANAGEMENT

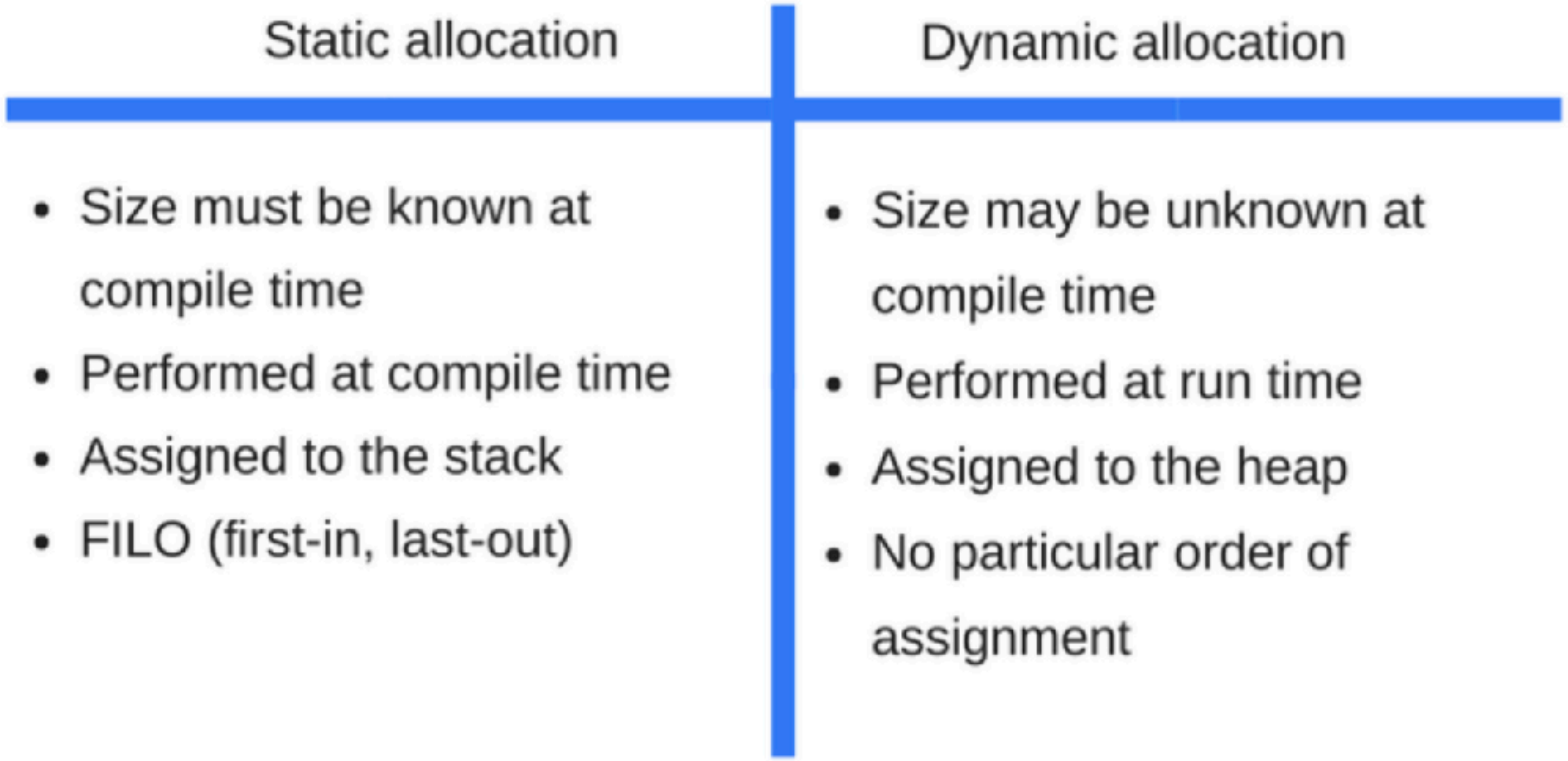


Allocate memory → Use memory → Release memory

- In low-level languages, developer need to handle it.
- In high-level languages, the languages will help you
- But if you choose not to care about it, **this was a big mistake**

ALLOCATION IN JAVASCRIPT

- JavaScript relieves developers from the responsibility to handle memory allocations
- JavaScript does it by itself, alongside declaring values.



Static allocation	Dynamic allocation
• Size must be known at compile time • Performed at compile time • Assigned to the stack • FILO (first-in, last-out)	• Size may be unknown at compile time • Performed at run time • Assigned to the heap • No particular order of assignment

ALLOCATION IN JAVASCRIPT

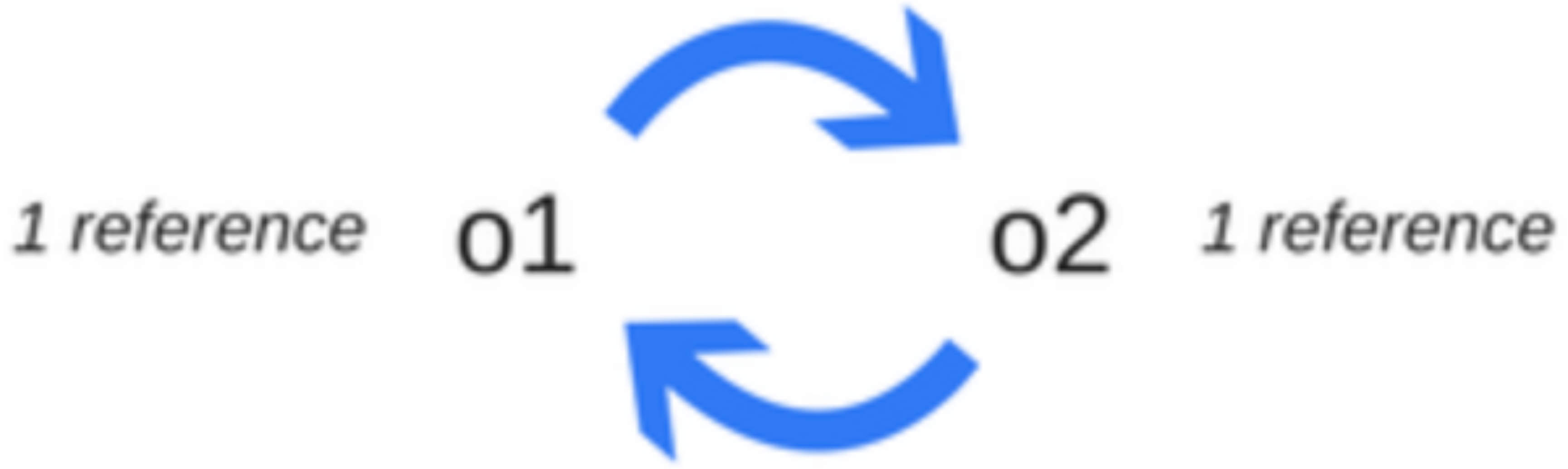
```
var n = 374; // allocates memory for a number
var s = 'sessionstack'; //allocates memory for a string
var o = {
  a: 1,
  b: null
};
 // allocates for object and its properties
var a = [1, null, 'str'];
// (like object)
// allocates for array and its elements
function f(a) {
  return a + 3;
} // allocates a function (which is a callable object)

// // function expressions also allocate an object
someElement.addEventListener('click', function() {
  someElement.style.backgroundColor = 'blue';
}, false);
```

```
var s1 = 'sessionstack';
var s2 = s1.substr(0, 3);
//Since strings are immutable,
//JavaScript may decide to not allocate memory,
// but just store the [0, 3] range.
var a1 = ['str1', 'str2'];
var a2 = ['str3', 'str4'];
var a3 = a1.concat(a2);
// new array with 4 elements being
// the concatenation of a1 and a2 elements
```

RELEASE MEMORY IN JAVASCRIPT

- **Garbage collector:** track memory allocation and use in order to find when a piece of allocated memory is not needed any longer in which case, it will automatically free it.
- **Reference-counting garbage collection:** the object that there are no references pointing to it.

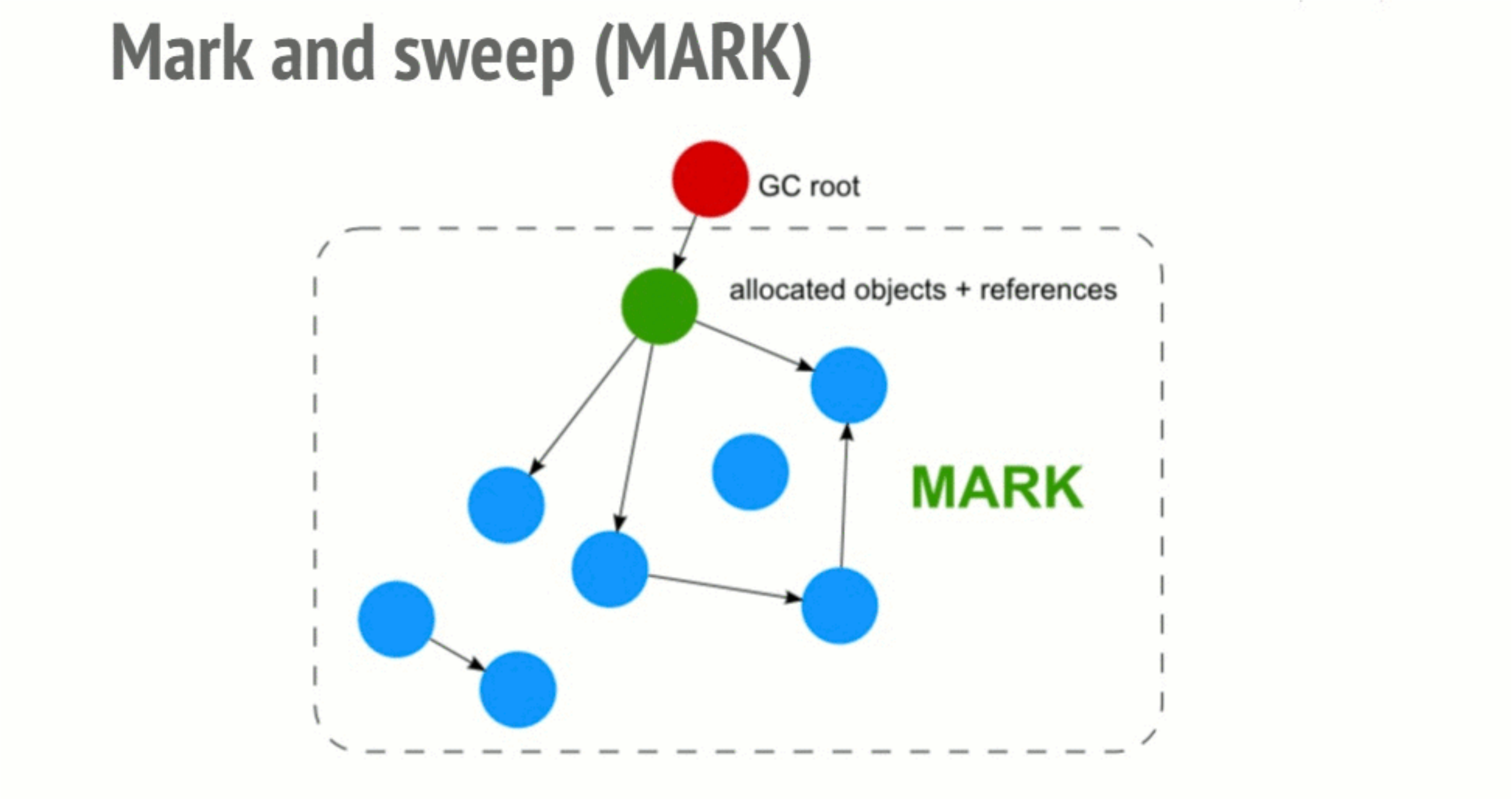
Cycles are creating problems



MARK-AND-SWEEP ALGORITHM

- 1.**Roots:** global variables which get referenced in the code. (window in browser and global in nodes)
- 2. inspects **all roots and their children** and marks them as active (meaning, they are not garbage)
- 3. Finally, the garbage collector frees all memory pieces that are not marked as active and returns that memory to the OS.

MARK-AND-SWEEP ALGORITHM

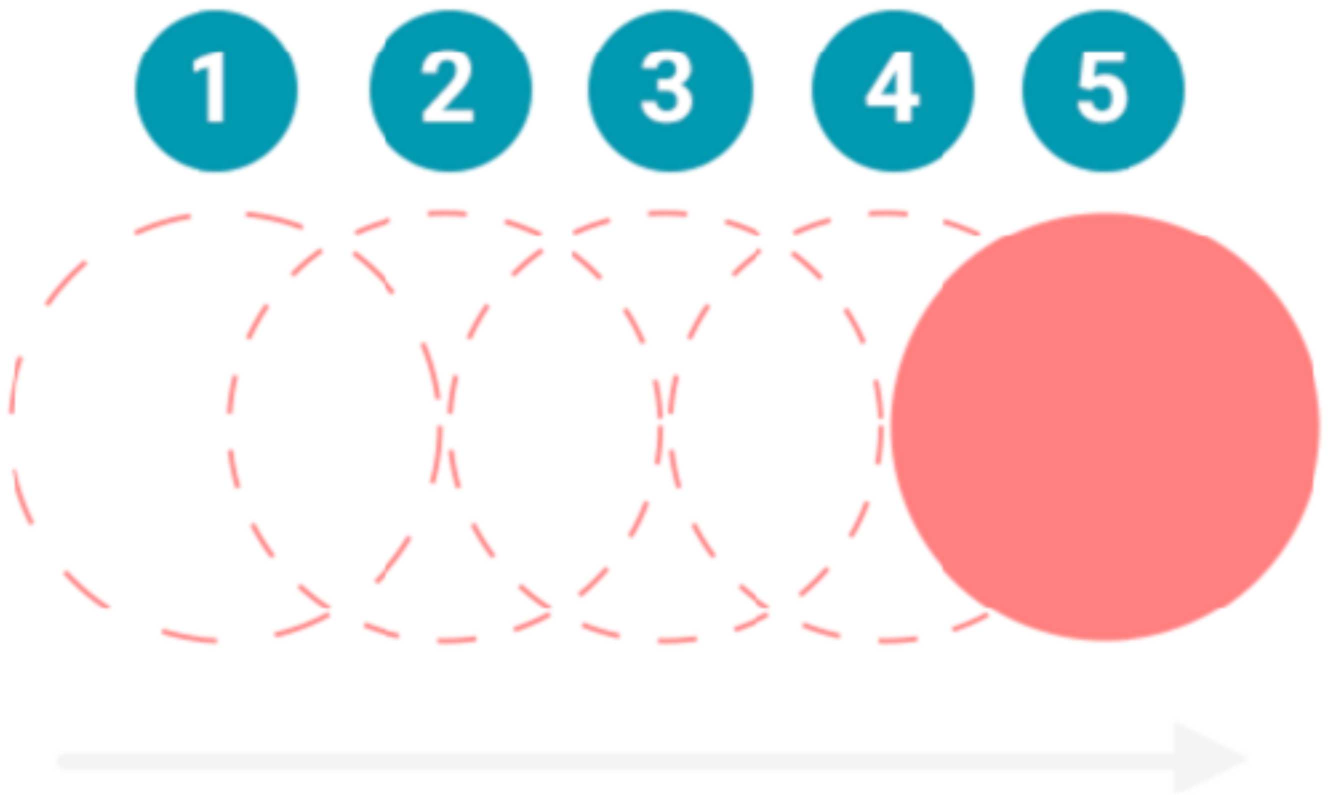


COMMON JAVASCRIPT MEMORY LEAKS

- 1: Global variables
- 2: Timers or callbacks that are forgotten
- 3: Closures
- 4: Out of DOM references

PRATICE: CREATING A GAMELOOP – THE BASE OF ANIMATION

```
function update(progress) {
  // Update the state of the world for
// the elapsed time since last render
}

function draw() {
  // Draw the state of the world
}

function loop(timestamp) {
  var progress = timestamp - lastRender

  update(progress)
  draw()

  lastRender = timestamp
  window.requestAnimationFrame(loop)
}
var lastRender = 0
window.requestAnimationFrame(loop)
```



```
// A bad game loop
while (running) {
    draw();
}
```

```
// Another bad game loop
setInterval(gameLoop, 16);

function gameLoop() {
    draw();
}
```