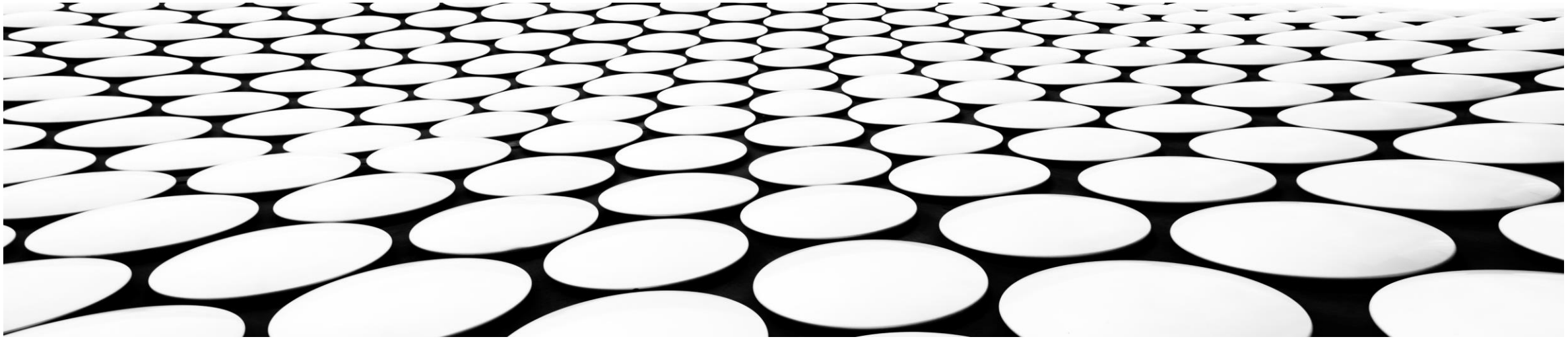
JAVASCRIPT – UNDER THE HOOD : CLASSES AND PROTOTYPES

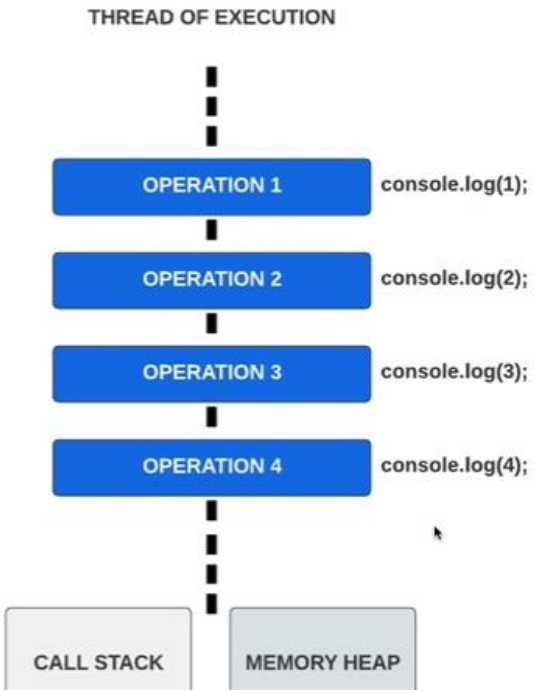


PRODEVANS

EXECUTION CONTEXT | CALL STACK | PROTOTYPE | __PROTO__ | PROTOTYPE CHAIN | FACTORIES | CONSTRUCTORS | CLASSES| ES6 AND ES5 APPROACHES

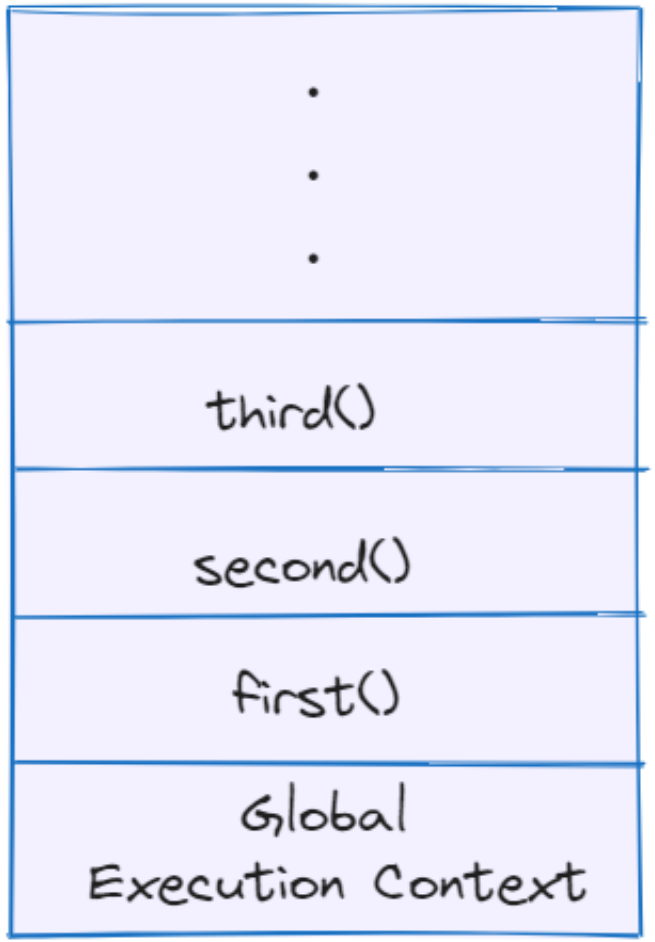
FEATURES OF JAVASCRIPT

- JavaScript is *a **single-threaded language***
- Single sequential flow of control
- JavaScript is a ***synchronous language*** with asynchronous capabilities
- A thread has a ***call stack and memory***



THE CALL STACK

- A call stack keeps track of our functions.
- It manages what we call as **Execution Context.**
- Stacks are LIFO **last in first out**



EXECUTION CONTEXT

- Whenever we run our JavaScript code, whether in browser or in NodeJS, it creates a special environment that handle the transformation and execution of code. This is called the **execution context.** It contains the currently running code and everything that aids in its execution.

- There is a global execution context as well as a function execution context for every function invoked.

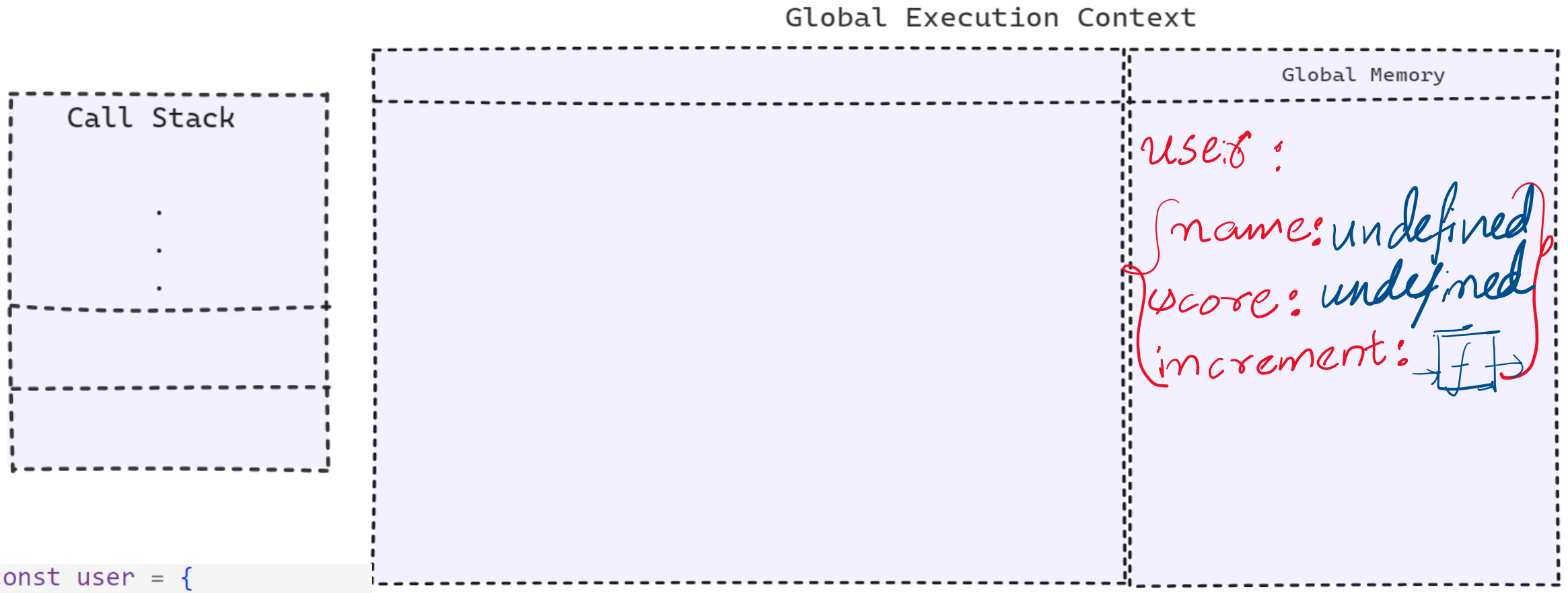
EXECUTION CONTEXT PHASES

- Memory Creation Phase:
  - Create the global object
    - Browser = window, Node.js = global
  - Create the **'this'** object and bind it to the global object.
  - Setup memory heap for storing variables and function references.
  - Store functions and variables(var) in global execution context and set it to **"undefined"**

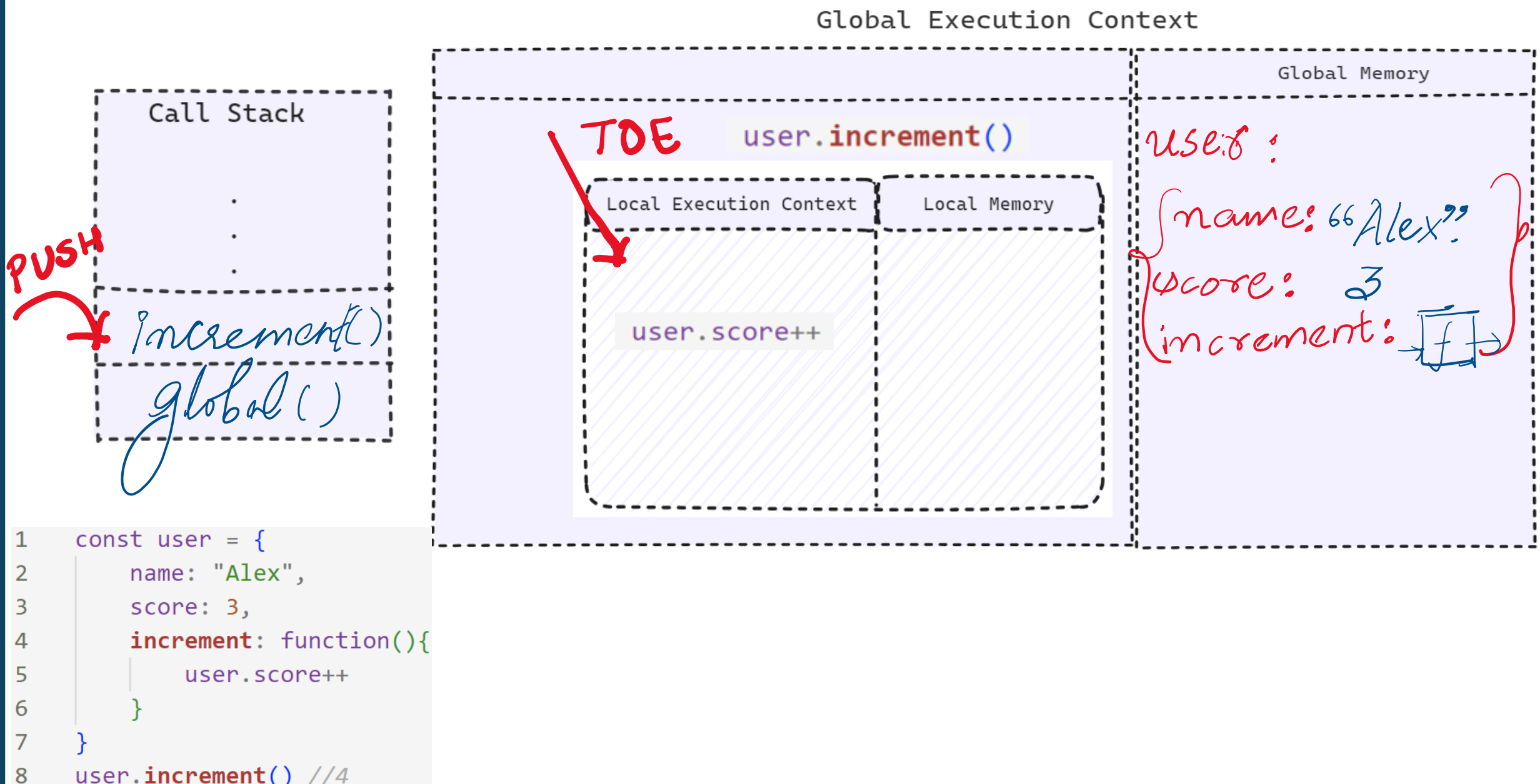
- Execution Phase:
  - Execute code line by line
  - Create a new execution context for each function call.

THREAD OF EXECUTION

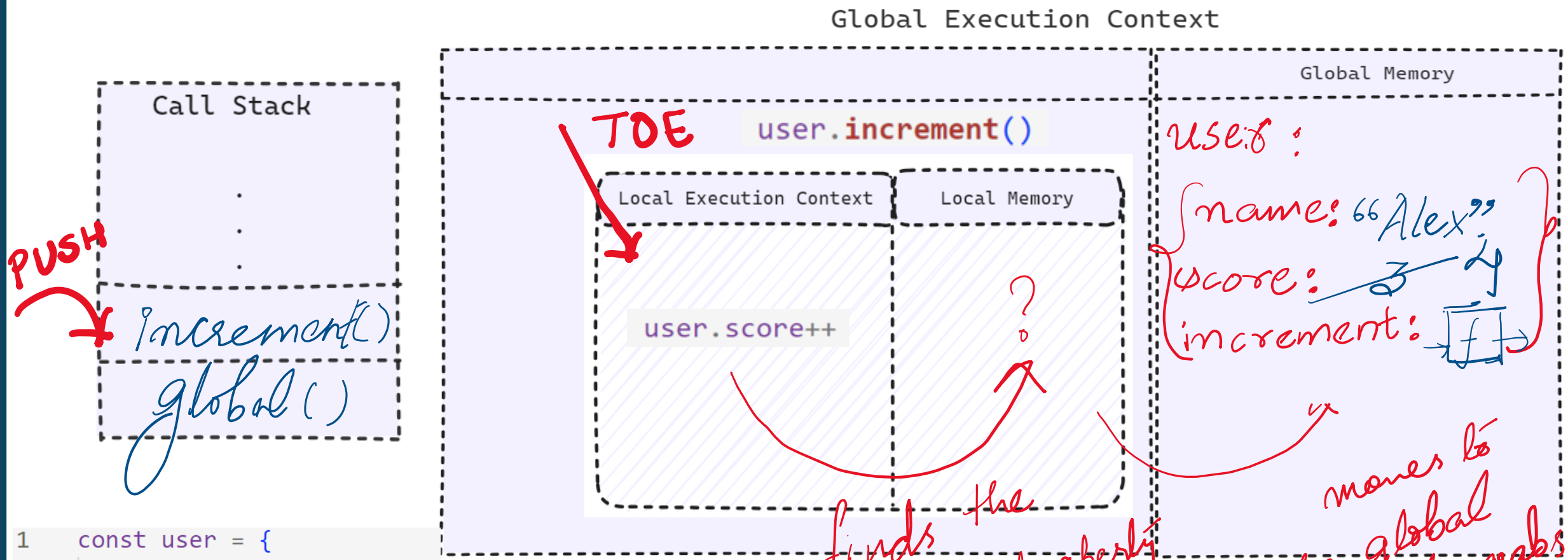
- JavaScript goes through the code (globally or in a function) line by line and does whatever the line of code says to do




```
const user = {
    name: "Alex",
    score: 3,
    increment: function(){
        user.score++
    }
}
user.increment() //4
```

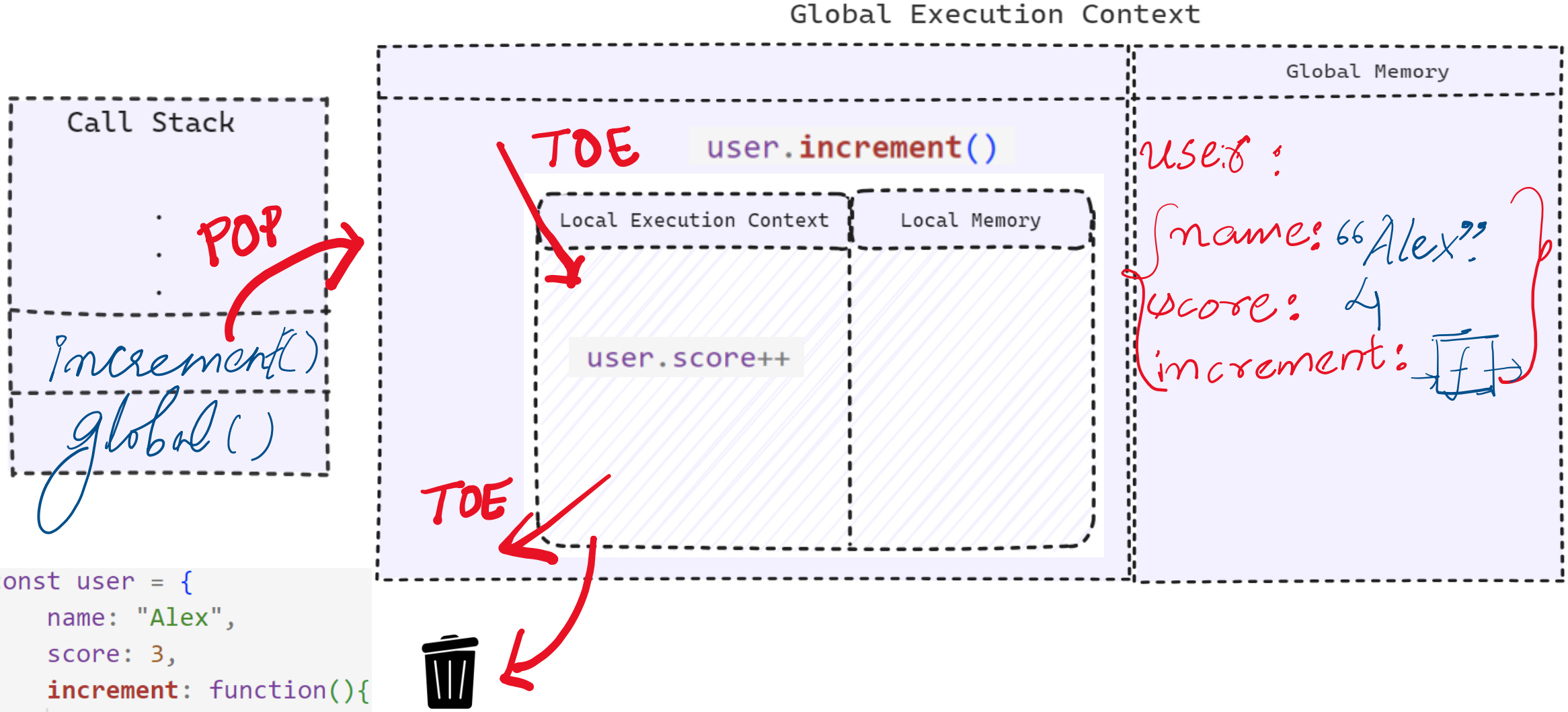



```
1   const user = {
2       name: "Alex",
3       score: 3,
4       increment: function(){
5           user.score++
6       }
7   }
8   user.increment() //4
```




```
1   const user = {
2       name: "Alex",
3       score: 3,
4       increment: function(){
5           user.score++
6       }
7   }
8   user.increment() //4
```

function call




```
1   const user = {
2       name: "Alex",
3       score: 3,
4       increment: function(){
5           user.score++
6       }
7   }
8   user.increment() //4
```

EXAMPLE - 1

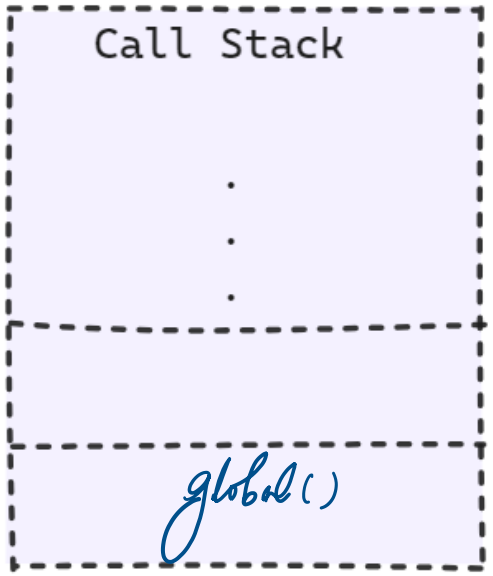
JS example.js > ...

```
1   function userCreator(name, score){
2       const newUser = {};
3       newUser.name = name;
4       newUser.score = score;
5       newUser.increment = function(){
6           newUser.score++;
7       };
8       return newUser;
9   }
10
11  var user1 =userCreator("Annaya", 4);
12  var user2 = userCreator("Sanjana", 8);
13
14  user1.increment();
```

EXAMPLE - 2

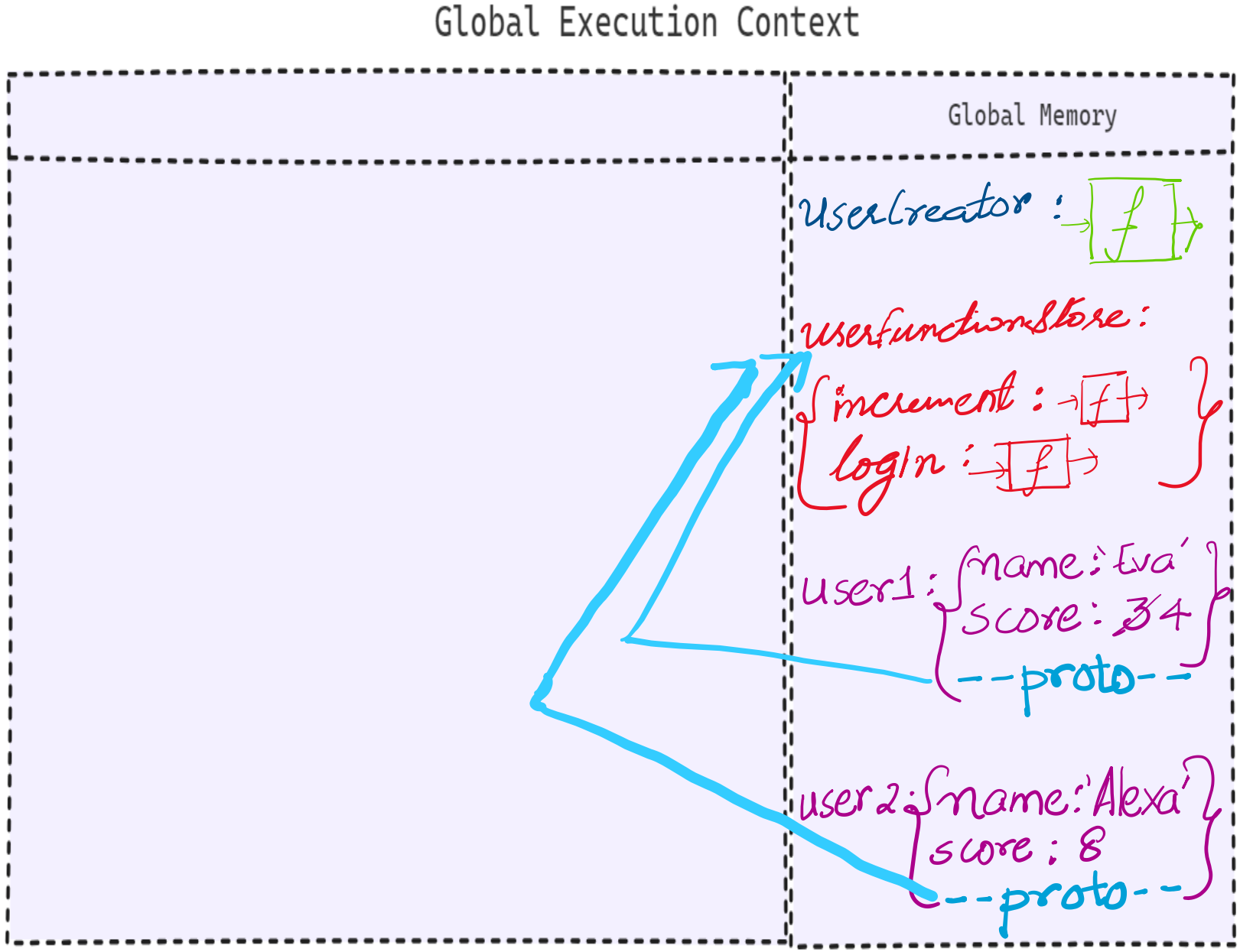
example.js > ...

```
1   function userCreator(name, score){
2       const newUser = Object.create(userFunctionStore);
3       newUser.name = name;
4       newUser.score = score;
5       return newUser;
6   };
7
8   const userFunctionStore = {
9       increment: function() {this.score++;},
10      login: function() {console.log("Logged In");}
11  };
12
13  const user1 =userCreator("Eva", 4);
14  const user2 = userCreator("Alexa", 8);
15  user1.increment();
```



example.js > ...

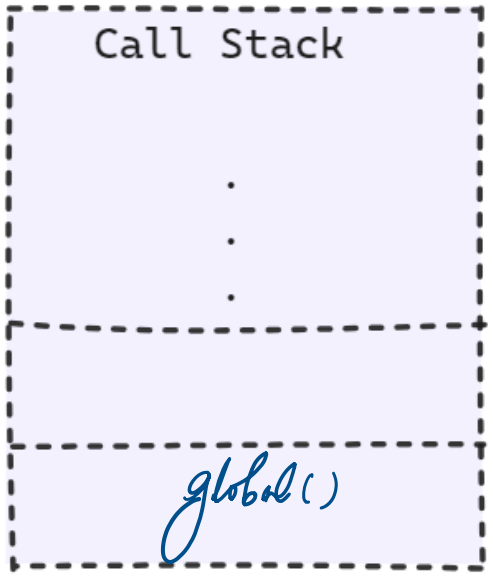
```
1   function userCreator(name, score){
2       const newUser = Object.create(userFunctionStore);
3       newUser.name = name;
4       newUser.score = score;
5       return newUser;
6   };
7
8   const userFunctionStore = {
9       increment: function() {this.score++;},
10      login: function() {console.log("Logged In");}
11  };
12
13  const user1 =userCreator("Eva", 4);
14  const user2 = userCreator("Alexa", 8);
15  user1.increment();
```



EXAMPLE - 3

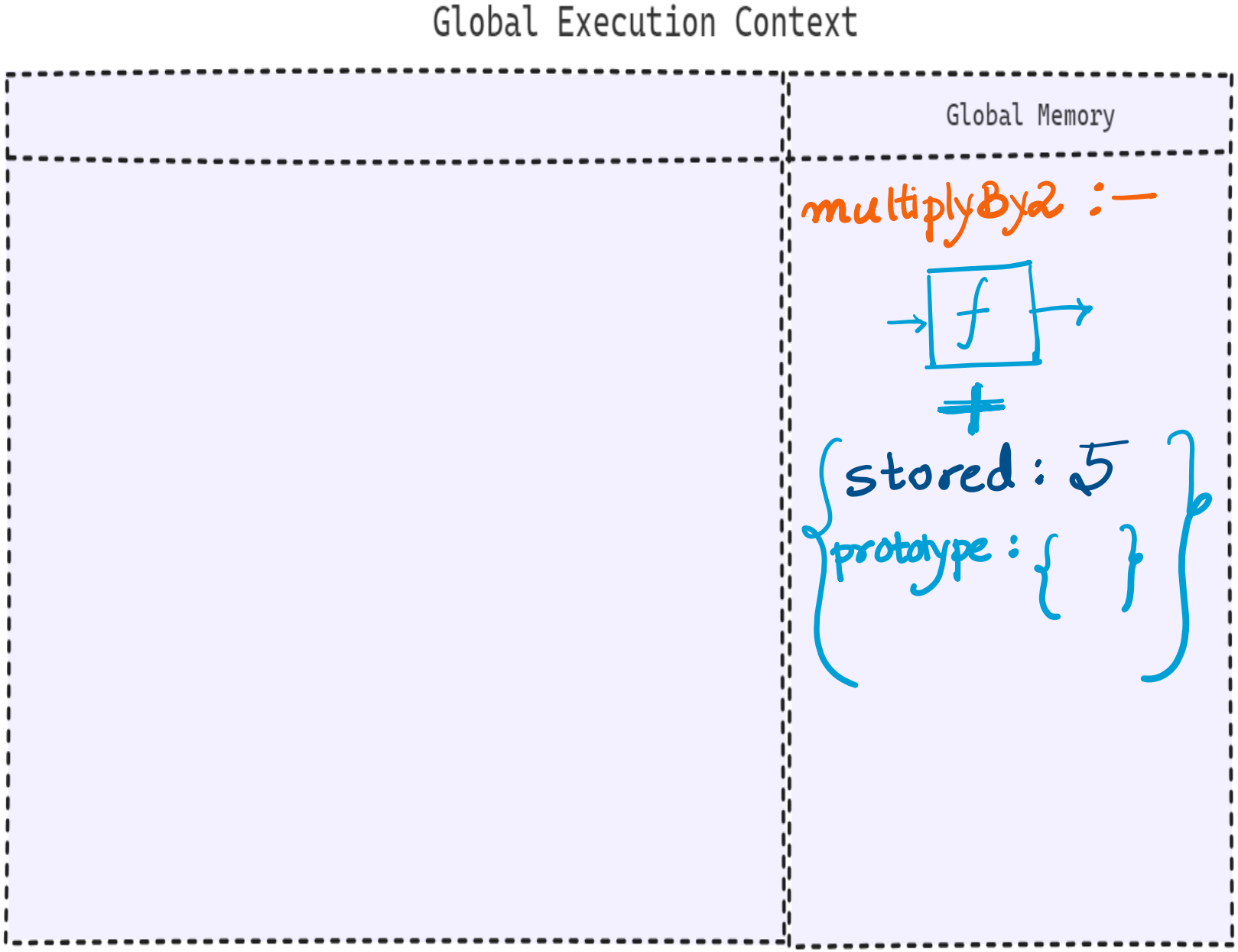
JS example.js > ...

```
1   function multiplyBy2(num){
2       return num*2;
3   };
4
5   multiplyBy2.stored = 5;
6   multiplyBy2(3);
7
8   multiplyBy2.stored; // 5
9   multiplyBy2.prototype; // {}
```



example.js > ...

```
1   function multiplyBy2(num){
2       return num*2;
3   };
4
5   multiplyBy2.stored = 5;
6   multiplyBy2(3);
7
8   multiplyBy2.stored; // 5
9   multiplyBy2.prototype; // {}
```



REFERENCES :

- Object.create() - JavaScript | MDN (mozilla.org)