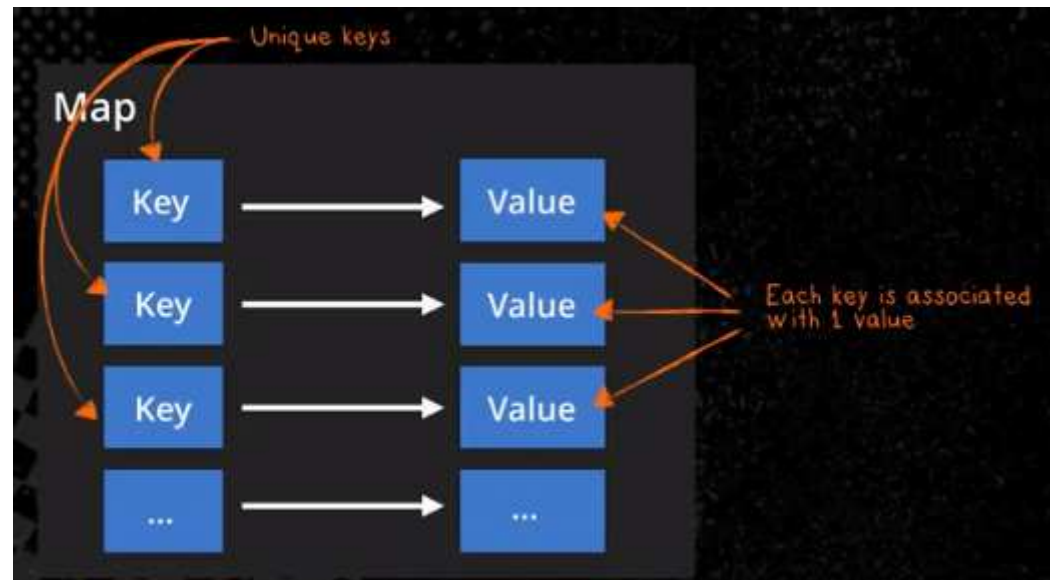


# Novedades ES6 - maps

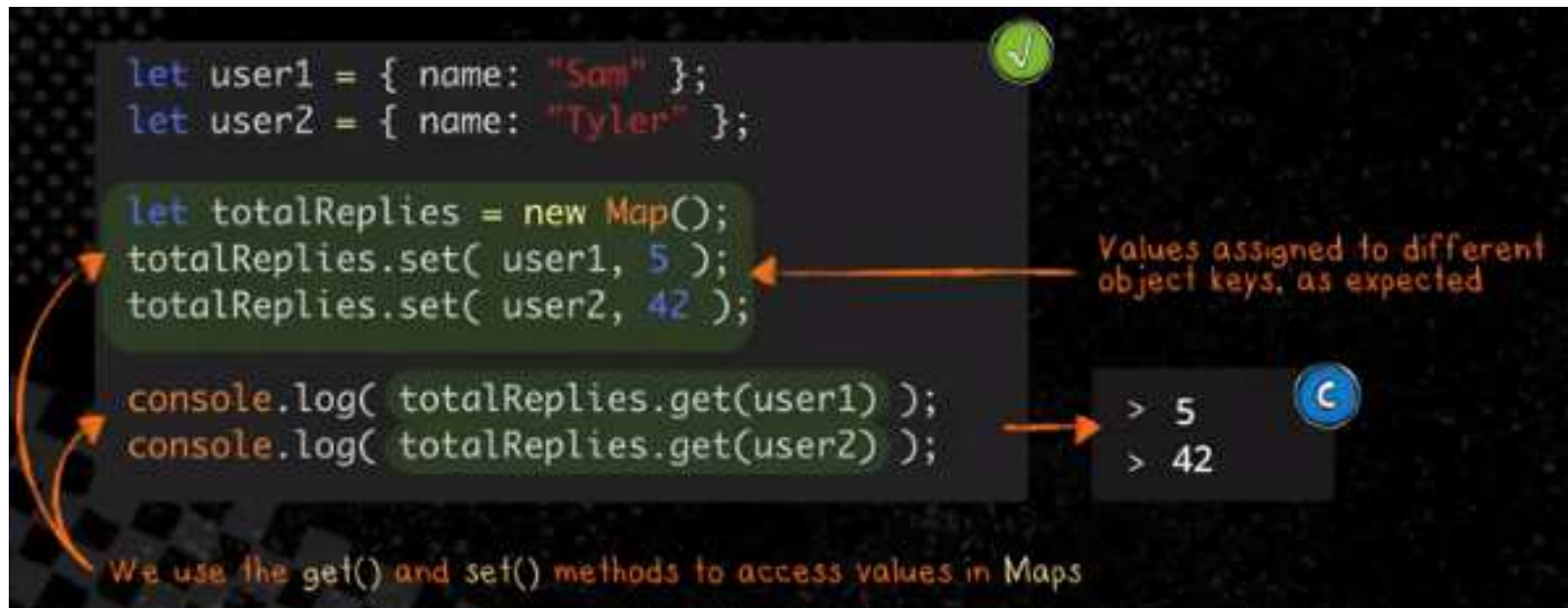
- **Maps**

- Un mapa es una estructura de datos compuesta por una colección de pares **clave / valor**.
- Son muy útiles para almacenar datos simples como valores de propiedades.



# Novedades ES6 - maps

- **Maps – declaración y métodos**
  - Cualquier valor puede ser usado como clave o valor y los objetos no son convertidos a cadenas



# Novedades ES6 - maps

- **Maps – declaración, métodos y propiedades**

- Los principales métodos son:

- Set(key, value) – añade una nueva pareja clave - valor
- Get (key) – obtiene el valor asociado a una clave
- Delete (key) – borra una pareja clave – valor a través de la clave
- Has(key) – comprueba si existe una determinada clave en el mapa

```
var map = new Map();

map.set( 'one', 1 );
map.set( 'two', 2 );
map.set( 'three', 3 );

console.log( "map.get('two') =", map.get('two') );

map.delete('three');

console.log( "map.has('three') =", map.has('three') );
```

# Novedades ES6 - maps

- **Maps – declaración , métodos y propiedades**
  - Otros métodos y propiedades son:
    - Size : contiene el número de valores en el mapa
    - forEach((value,key,map) => {}, [thisValue]): recorre todos los elementos contenidos en el mapa
    - Values(): devuelve un iterable con los valores del mapa
    - Keys(): devuelve un iterable con las claves del mapa
    - Entries(): devuelve un iterable con matrices [key,value]
    - Clear(): elimina todos los valores del mapa

# Novedades ES6 - maps

- **Maps – usos**
  - Cuando no se conoce hasta el momento de ejecución cuáles son las claves.

## Map

```
let recentPosts = new Map();  
  
createPost(newPost, (data) => {  
  recentPosts.set( data.author, data.message );  
});
```

Keys unknown until runtime, so... Map!

## Object

```
const POSTS_PER_PAGE = 15;  
  
let userSettings = {  
  perPage: POSTS_PER_PAGE,  
  showRead: true,  
};
```

Keys are previously defined, so... Object!

# Novedades ES6 - maps

- **Maps – usos**
  - Cuando los tipos sean iguales

**Map**

```
let recentPosts = new Map();

createPost(newPost, (data) => {
  recentPosts.set( data.author, data.message );
});

// ...somewhere else in the code
socket.on('new post', function(data){
  recentPosts.set( data.author, data.message );
});
```

All keys are the same type, and all values are the same type, so Map!

**Object**

```
const POSTS_PER_PAGE = 15;

let userSettings = {
  perPage: POSTS_PER_PAGE,
  showRead: true,
};
```

Some values are numeric, others are boolean, so Object!

# Novedades ES6 - maps

- **Maps – usos**
  - Son iterables, se puede utilizar con las estructuras for .. of

```
let mapSettings = new Map();

mapSettings.set( "user", "Sam" );
mapSettings.set( "topic", "ES2015" );
mapSettings.set( "replies", [ "Can't wait!", "So Cool!" ] );

for(let [key, value] of mapSettings){
  console.log(` ${key} = ${value}`);
}
```

Remember array destructuring?

```
> user = Sam
> topic = ES2015
> replies = Can't wait!, So Cool
```

# Novedades ES6 - maps

- ¿Qué salida proporcionará el siguiente código?

```
let topicInfo = {  
  title: "New Features in JS",  
  replies: 19,  
  lastReplyFrom: "Tyler"  
};  
  
for(let [k, v] of topicInfo){  
  console.log(`${k} - ${v}`);  
}
```

☐ title - New Features in JS  
replies - 19  
lastReplyFrom - Tyler

☐ New Features in JS - title  
19 - replies  
Tyler - lastReplyFrom

☐ A `TypeError`

☐ Blank output



# Novedades ES6 - maps

- ¿Qué error tiene el siguiente código?

```
let recentPosts = {};  
  
getPost(postId, (data) => {  
  recentPosts[data.author] = data.title;  
});
```

- ☐ A regular function should be used instead of an arrow function.
- ☐ The `recentPosts` variable should be declared with `var` instead of `let`.
- ☐ `recentPosts` should be a Map, since keys are unknown until runtime.

# Novedades ES6 - maps

- En el siguiente código, ¿es apropiado usar un objeto en vez de un map?

```
const USERS_PER_PAGE = 10;

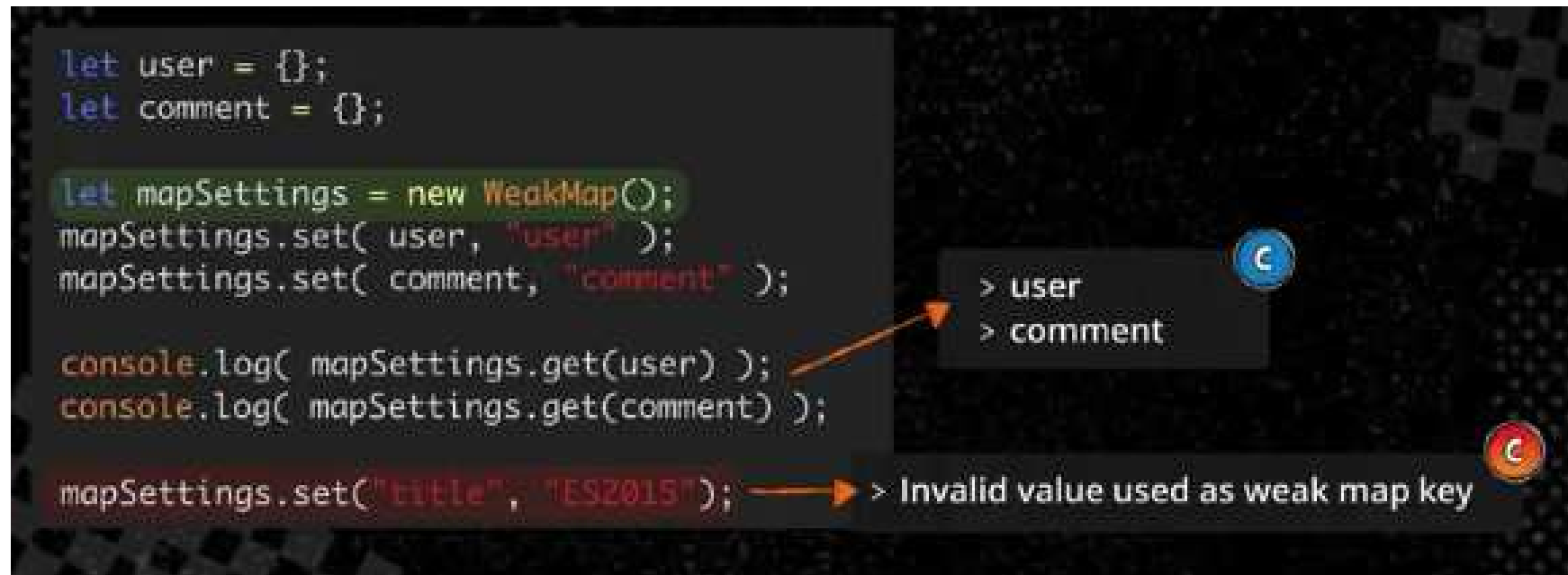
let pageSettings = {
  perPage: USERS_PER_PAGE,
  canSort: false
};
```

- ☐ Yes. Since keys like `perPage` and `canSort` are previously defined, we don't run the risk of accidentally overwriting values.
- ☐ No. We should always use maps and never ever use objects again. Ever.

# Novedades ES6 - weakmap

- **WeakMap**

- Son un tipo de Map donde las claves solo pueden ser objects. Los datos primitivos (strings, numbers, booleans,...) no están permitidos.




# Novedades ES6 - weakmap

- **WeakMap**

- Para acceder a la estructura se hace a través del objeto utilizado como clave.

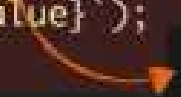
```
let user = {};  
  
let mapSettings = new WeakMap();  
mapSettings.set( user, "ES2015" );  
  
console.log( mapSettings.get(user) );  
console.log( mapSettings.has(user) );  
console.log( mapSettings.delete(user) );
```



```
> ES2015  
> true  
> true
```

- No son estructuras iterables, no podemos utilizar for..of

```
for(let [key,value] of mapSettings){  
  console.log( `${key} = ${value}` );  
}
```



```
> mapSettings[Symbol.iterator] is not a function
```

# Novedades ES6 - weakmap

- **WeakMap – métodos y propiedades**
  - Get(key)
  - Set(key, value)
  - Has(key)
  - Delete(key)

# Novedades ES6 - weakmap

- **WeakMap vs Map**

- Son más eficientes con la gestión de memoria
- Con los mapas normales se mantienen las referencias a los objetos clave, impidiéndole ser recolectados. Con WeakMap los objetos clave pueden ser recolectados en caso de que no haya otras referencias al objeto

```
let user = {};  
  
let userStatus = new WeakMap();  
userStatus.set( user, "logged" );  
  
//...  
someOtherFunction( user );
```

All objects occupy memory space

Object reference passed as key to the WeakMap

Once it returns, user can be garbage collected

WeakMaps don't prevent the garbage collector from collecting objects currently used as keys, but that are no longer referenced anywhere else in the system



# Novedades ES6 - set

- **Sets**

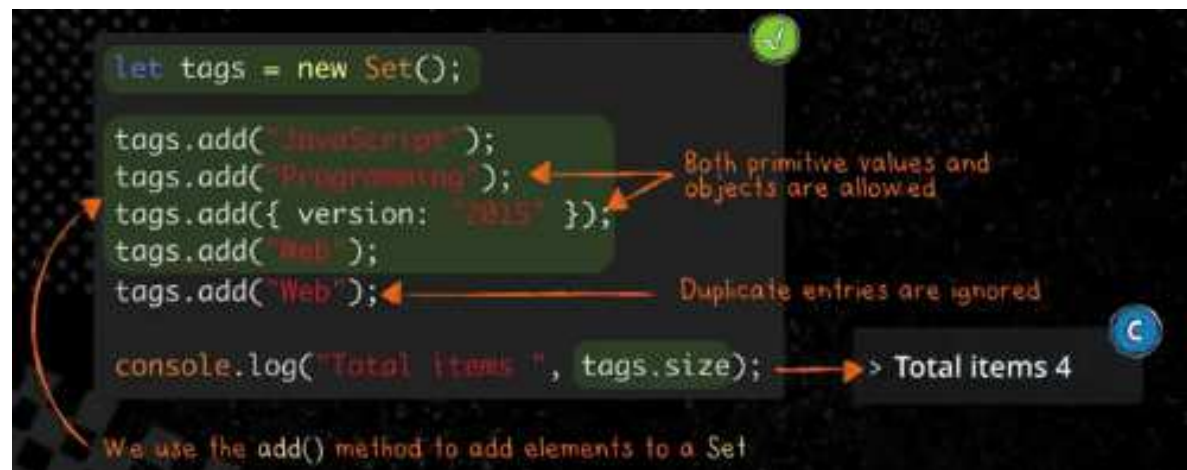
- Arrays pueden tener entradas duplicadas



```
let tags = [];  
tags.push( "JavaScript" );  
tags.push( "Programming" );  
tags.push( "Web" );  
tags.push( "Web" );  
  
console.log( "Total items ", tags.length );
```

The screenshot shows a code editor with a dark background. The code defines an array `tags` and pushes four elements: "JavaScript", "Programming", "Web", and "Web". A red arrow points to the second "Web" entry with the text "Duplicate entry". Below the code, the console output is shown as `> Total items 4`. There are red 'x' and 'c' icons in the top right corner of the code editor window.

- Set almacena valores **únicos** y de **cualquier tipo**



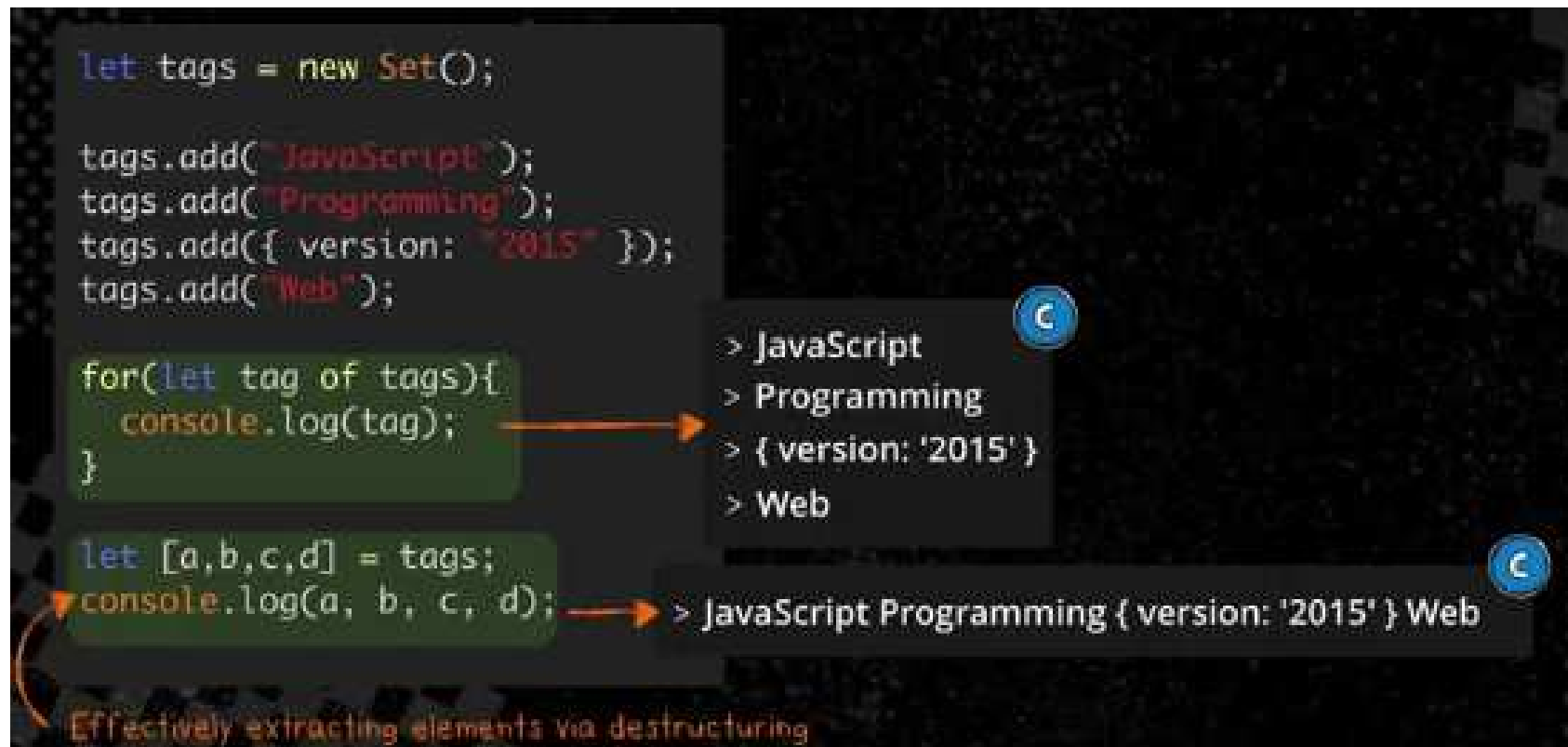
```
let tags = new Set();  
tags.add( "JavaScript" );  
tags.add( "Programming" );  
tags.add( { version: "2015" } );  
tags.add( "Web" );  
tags.add( "Web" );  
  
console.log( "Total items ", tags.size );
```

The screenshot shows a code editor with a dark background. The code defines a `Set` object `tags` and adds five elements: "JavaScript", "Programming", an object `{ version: "2015" }`, "Web", and "Web". A green arrow points to the first three elements with the text "Both primitive values and objects are allowed". A red arrow points to the second "Web" entry with the text "Duplicate entries are ignored". Below the code, the console output is shown as `> Total items 4`. There are green and blue 'x' and 'c' icons in the top right corner of the code editor window. A red arrow points from the text "We use the add() method to add elements to a Set" at the bottom to the `tags.add()` calls in the code.

# Novedades ES6 - set

- **Sets**

- Son objetos iterables, por tanto podemos usar for..of y destructuring





# Novedades ES6 - set

- **Sets – métodos y propiedades**
  - Size
  - Add(value)
  - Has(value)
  - Delete(value)
  - Entries()
  - Values()

# Novedades ES6 - set

- **Sets vs Arrays**

<b>Array/indexOf</b>	<b>Set/has</b>
0.50ms	0.05ms

<b>Array/push</b>	<b>Set/add</b>
0.13ms	0.04ms

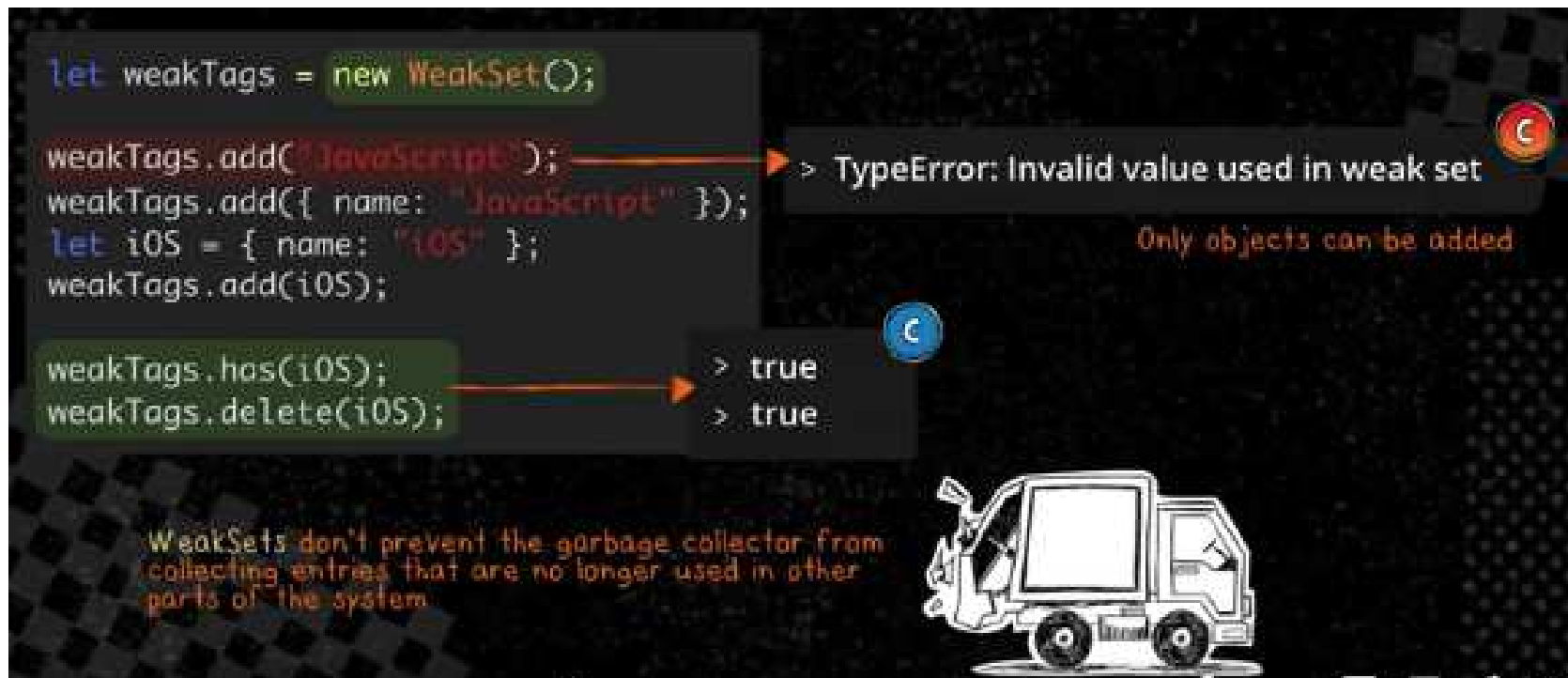
<b>Array/deleteFromArr</b>	<b>Set/remove</b>
15.55ms	0.04ms

	<b>Diferencia</b>	<b>Intersección</b>
<b>Array</b>	0.75ms	0.37ms
<b>Set</b>	0.30ms	0.24ms

# Novedades ES6 – weakset

- **WeakSet**

- Al igual que los WeakMaps son más eficientes con la memoria cuando se almacenan objetos



- No son iterables, no se pueden utilizar con `for...of`, no se pueden consultar sus valores

# Novedades ES6 – weakset

- **WeakSet**
  - ¿Cuándo son útiles estas estructuras?



# Novedades ES6 – weakset

- WeakSet

```
let allPosts = new WeakSet();

let post1 = { title: "ES2015" };
let post2 = { title: "CoffeeScript" };
let post3 = { title: "TypeScript" };

allPosts.add( post1 );
allPosts.add( post2 );
allPosts.add( post3 );
```

¿Cómo podríamos consultar **allPosts** para determinar si tiene el objeto **post2**?

- ☐ `allPosts.get( post2 );`
- ☐ `allPosts.contains( post2 );`
- ☐ `allPosts.has( post2 );`