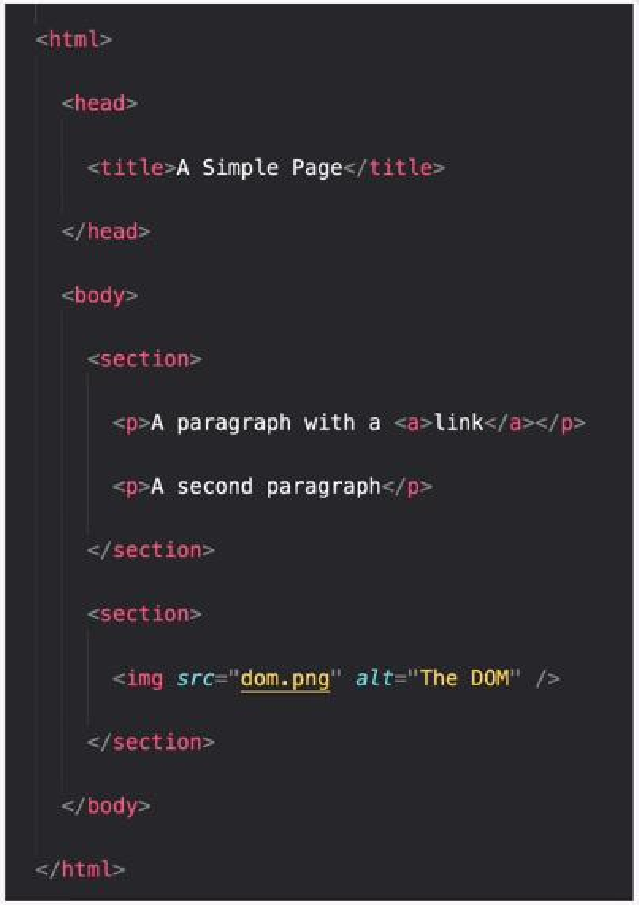
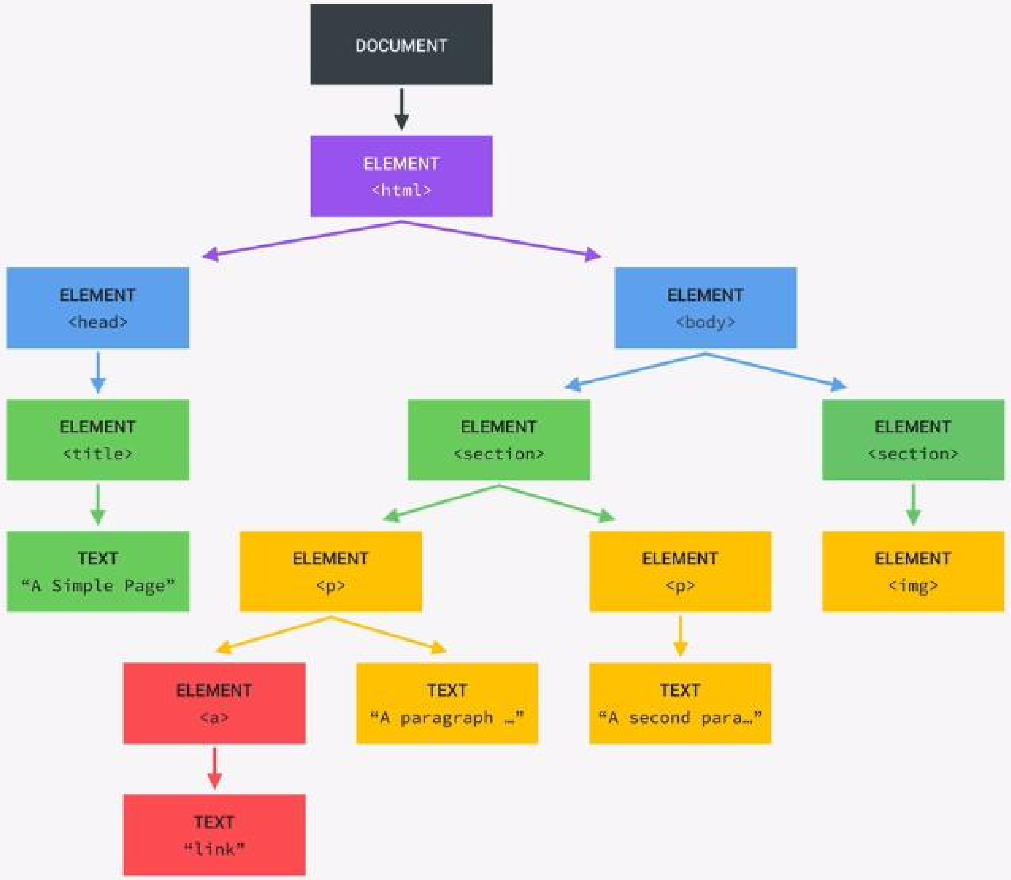
THE DOM TREE STRUCTURE



```
<html>
  <head>
    <title>A Simple Page</title>
  </head>
  <body>
    <section>
      <p>A paragraph with a <a>link</a></p>
      <p>A second paragraph</p>
    </section>
    <section>
      <img src="dom.png" alt="The DOM" />
    </section>
  </body>
</html>
```



PRIMITIVE VS. REFERENCE VALUES

👉 **Primitive** values example:

```
let age = 30;
let oldAge = age;
age = 31;
console.log(age); // 31
console.log(oldAge); // 30
```

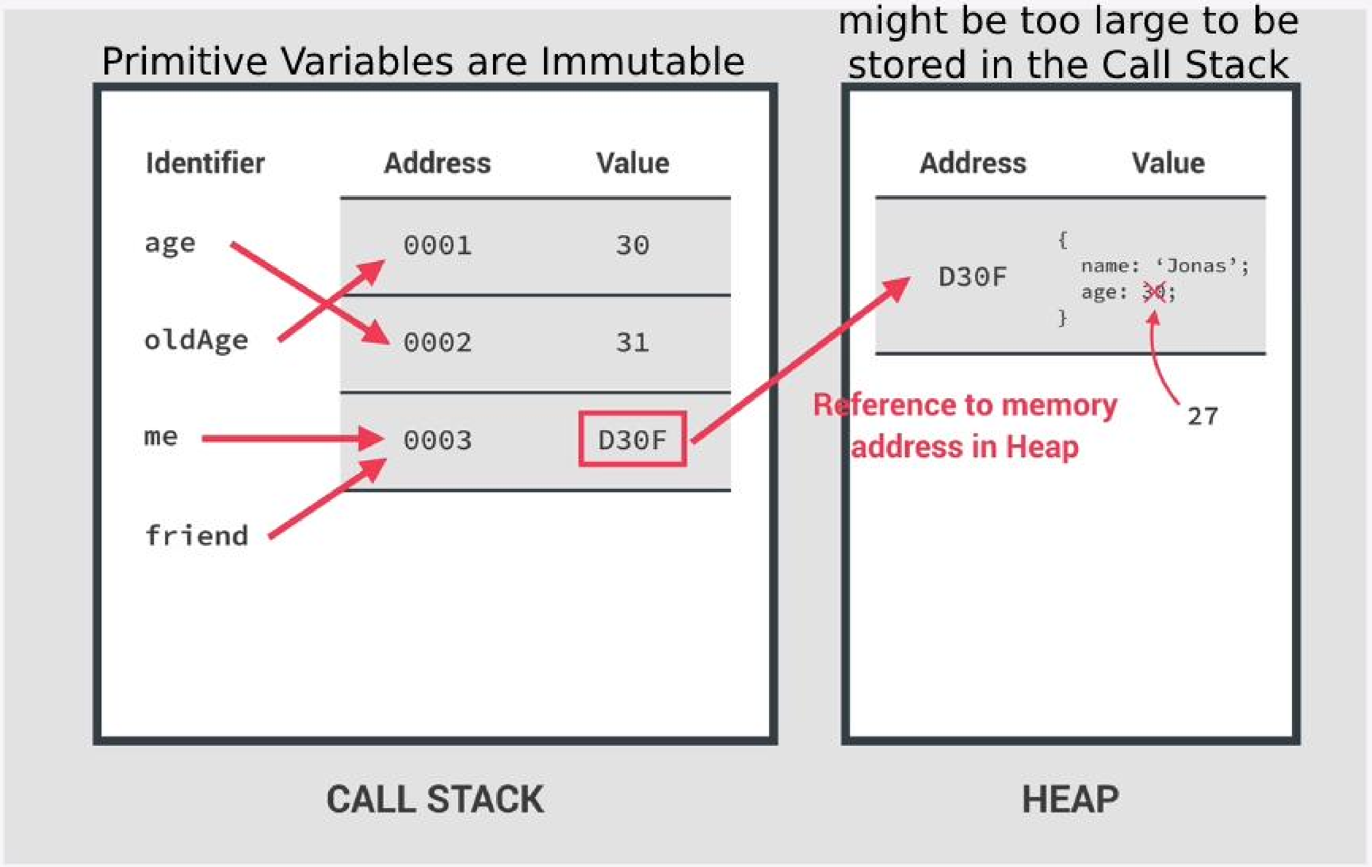
👉 **Reference** values example:

```
const me = {
  name: 'Jonas'
  age: 30
};
const friend = me;
friend.age = 27;

console.log('Friend:', friend);
// { name: 'Jonas', age: 27 }

console.log('Me:', me);
// { name: 'Jonas', age: 27 }
```

No problem, because we're NOT changing the *value* at address 0003!

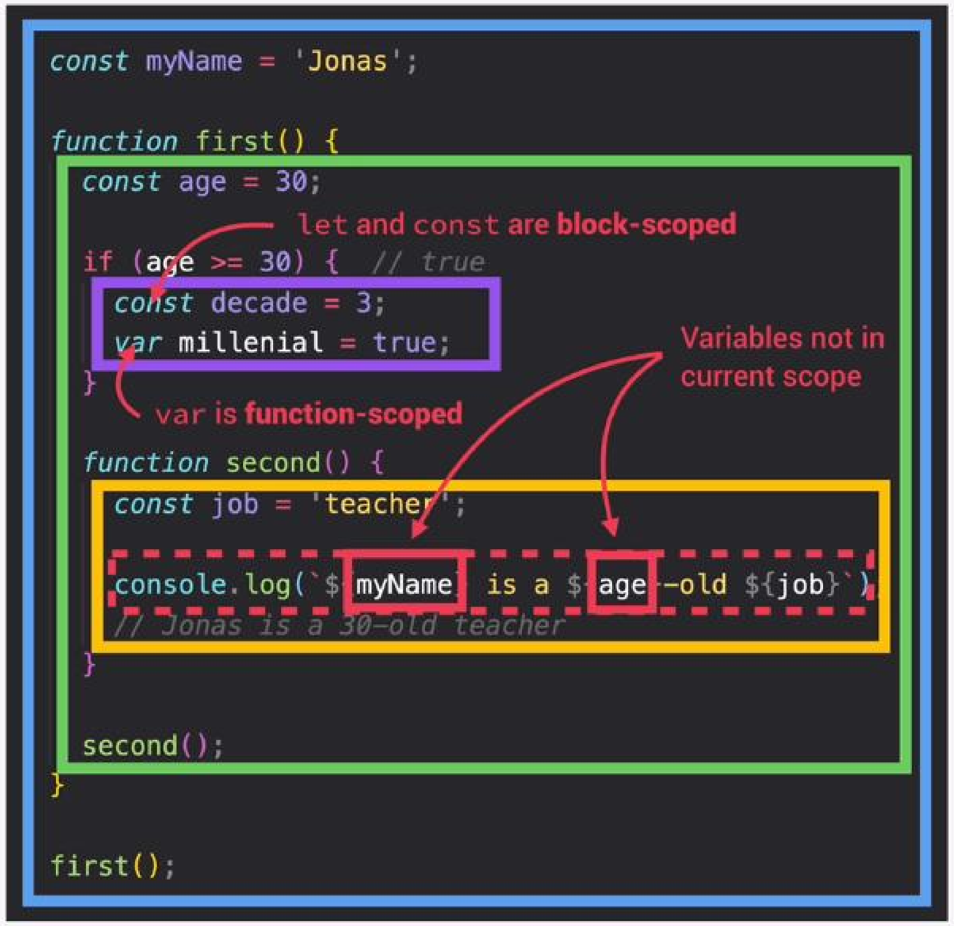
Reference Values are Stored in the Heap because they might be too large to be stored in the Call Stack

Primitive Variables are Immutable



Reference Variables Properties can be changed because it doesn't change its Address in the Call Stack

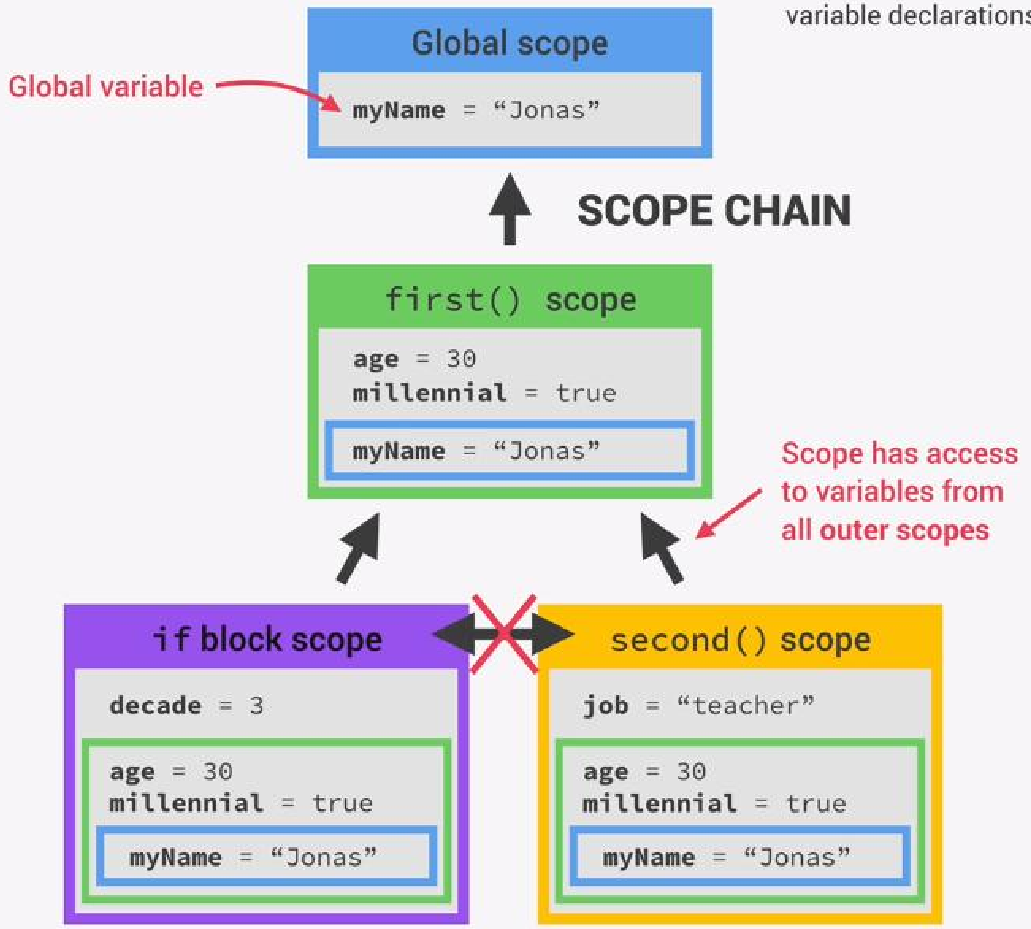
THE SCOPE CHAIN

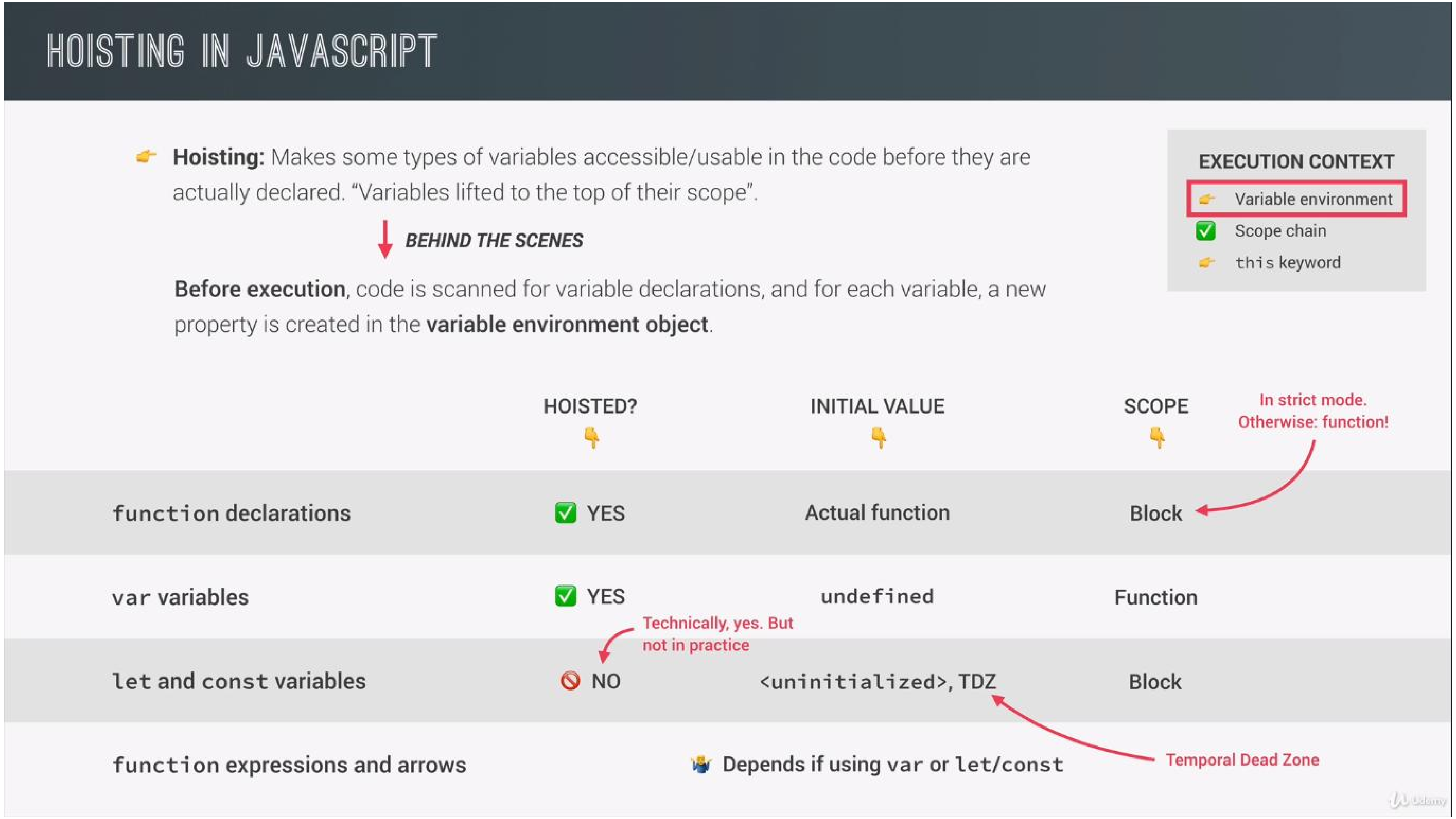


```
const myName = 'Jonas';

function first() {
  const age = 30;

  if (age >= 30) { // true
    const decade = 3;
    var millenial = true;
  }

  function second() {
    const job = 'teacher';

    console.log(`${myName} is a ${age}-old ${job}`);
    // Jonas is a 30-old teacher
  }

  second();
}

first();
```

let and const are block-scoped
var is function-scoped
Variables not in current scope

(Considering only variable declarations)





HOISTING IN JAVASCRIPT

👉 **Hoisting:** Makes some types of variables accessible/usable in the code before they are actually declared. "Variables lifted to the top of their scope".

↓ ***BEHIND THE SCENES***

Before execution, code is scanned for variable declarations, and for each variable, a new property is created in the **variable environment object**.

EXECUTION CONTEXT

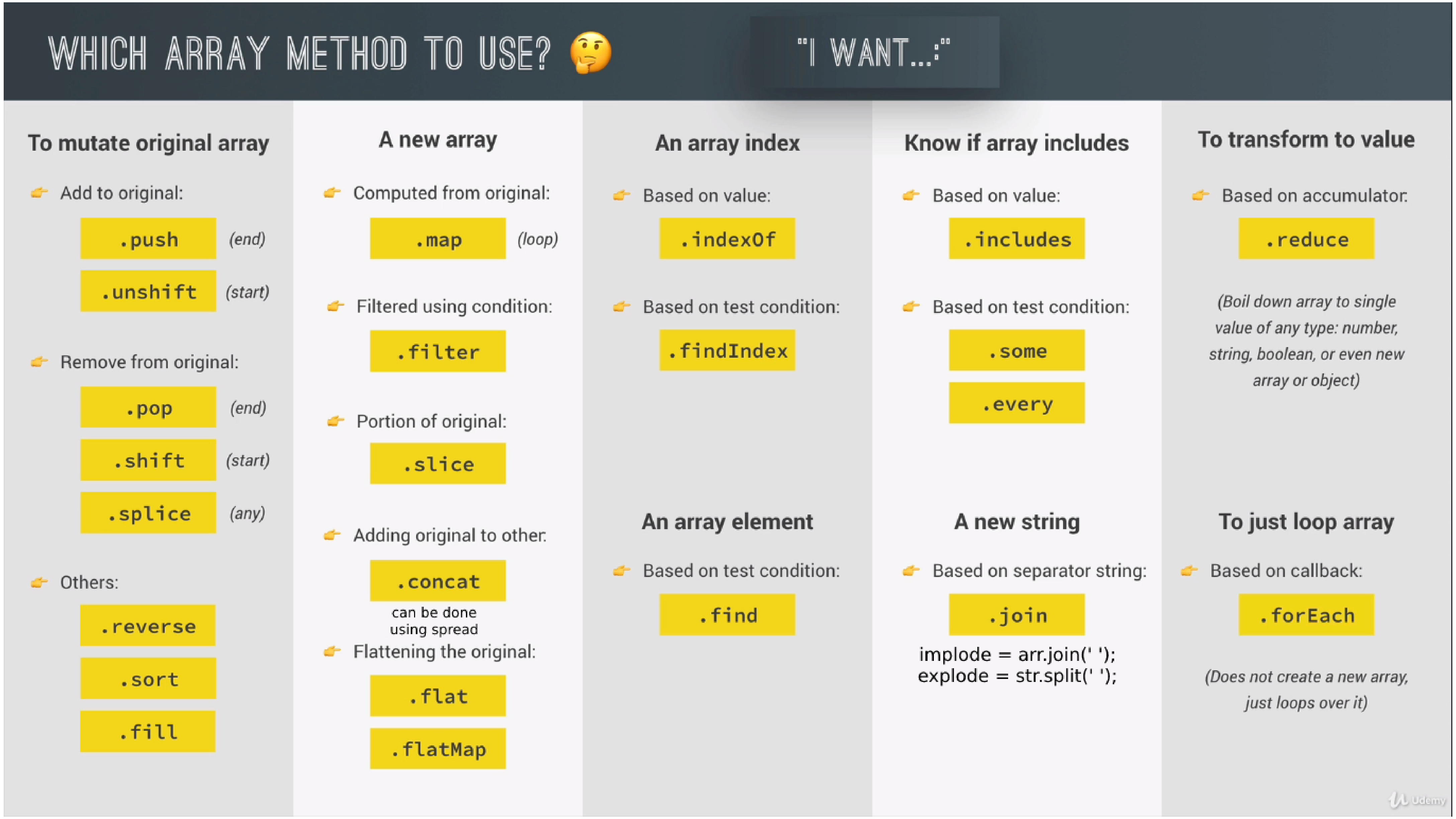
- 👉 Variable environment
- ✅ Scope chain
- 👉 `this` keyword

In strict mode. Otherwise: function!

	HOISTED? 👇	INITIAL VALUE 👇	SCOPE 👇
`function` declarations	✅ YES	Actual function	Block
`var` variables	✅ YES	`undefined`	Function
`let` and `const` variables	🚫 NO	`<uninitialized>`, TDZ	Block
`function` expressions and arrows	🤷 Depends if using `var` or `let/const`		

Technically, yes. But not in practice

Temporal Dead Zone



WHICH ARRAY METHOD TO USE? 🤔

"I WANT...:"

To mutate original array

- Add to original:
  - `.push` *(end)*
  - `.unshift` *(start)*
- Remove from original:
  - `.pop` *(end)*
  - `.shift` *(start)*
  - `.splice` *(any)*
- Others:
  - `.reverse`
  - `.sort`
  - `.fill`

A new array

- Computed from original:
  - `.map` *(loop)*
- Filtered using condition:
  - `.filter`
- Portion of original:
  - `.slice`
- Adding original to other:
  - `.concat`

  can be done using spread
- Flattening the original:
  - `.flat`
  - `.flatMap`

An array index

- Based on value:
  - `.indexOf`
- Based on test condition:
  - `.findIndex`

An array element

- Based on test condition:
  - `.find`

Know if array includes

- Based on value:
  - `.includes`
- Based on test condition:
  - `.some`
  - `.every`

A new string

- Based on separator string:
  - `.join`

implode = arr.join(' ');
explode = str.split(' ');

To transform to value

- Based on accumulator:
  - `.reduce`

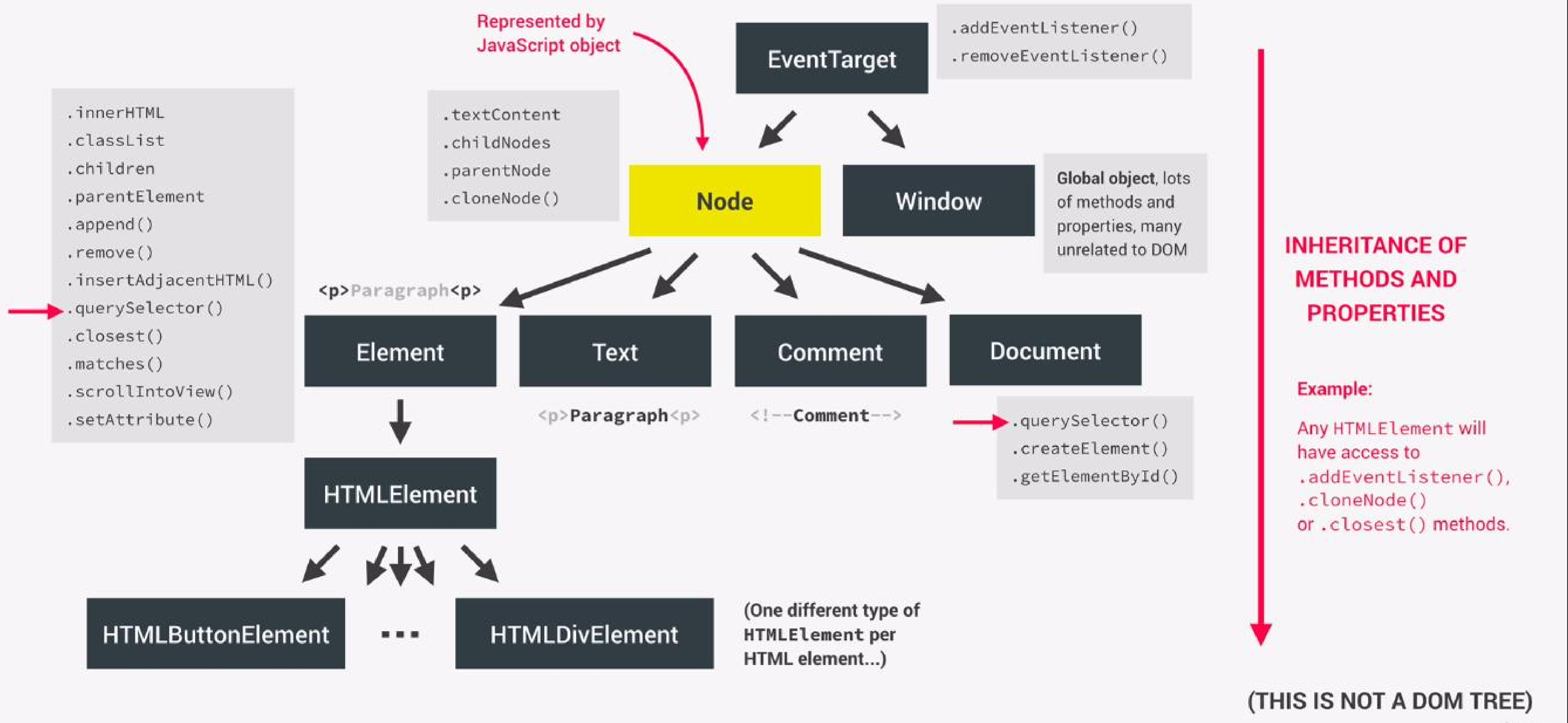
(Boil down array to single value of any type: number, string, boolean, or even new array or object)

To just loop array

- Based on callback:
  - `.forEach`

(Does not create a new array, just loops over it)

HOW THE DOM API IS ORGANIZED BEHIND THE SCENES



Represented by JavaScript object

```
.addEventListener()
.removeEventListener()
```

```
.textContent
.childNodes
.parentNode
.cloneNode()
```

Global object, lots of methods and properties, many unrelated to DOM

```
.innerHTML
.classList
.children
.parentElement
.append()
.remove()
.insertAdjacentHTML()
.querySelector()
.closest()
.matches()
.scrollIntoView()
.setAttribute()
```

```
.querySelector()
.createElement()
.getElementById()
```

(One different type of `HTMLElement` per HTML element...)

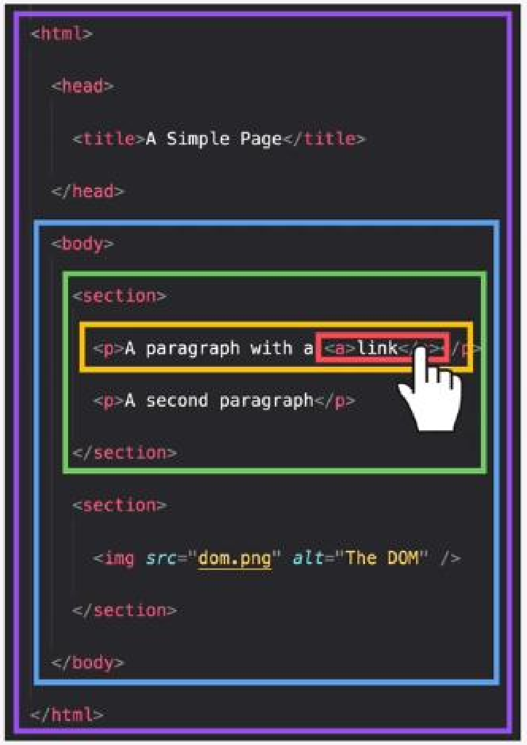
INHERITANCE OF METHODS AND PROPERTIES

Example:

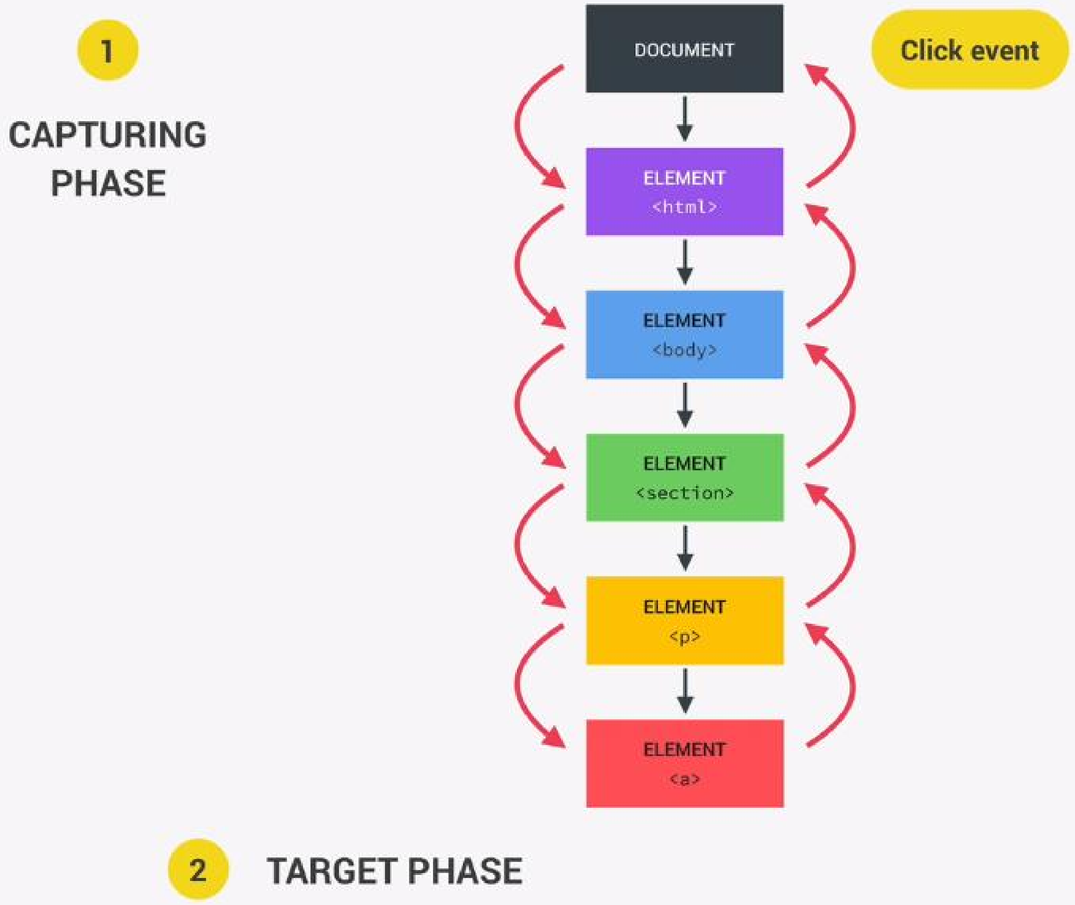
Any `HTMLElement` will have access to `.addEventListener()`, `.cloneNode()` or `.closest()` methods.

(THIS IS NOT A DOM TREE)

BUBBLING AND CAPTURING

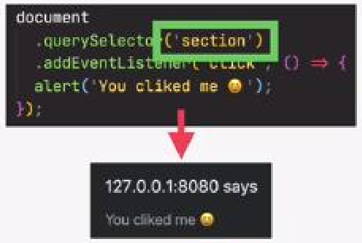


```
<html>
  <head>
    <title>A Simple Page</title>
  </head>
  <body>
    <section>
      <p>A paragraph with a <a>link</a></p>
      <p>A second paragraph</p>
    </section>
    <section>
      <img src="dom.png" alt="The DOM" />
    </section>
  </body>
</html>
```



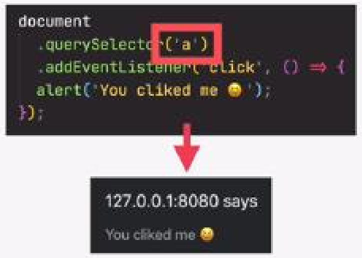
3

BUBBLING PHASE



```
document
  .querySelector('section')
  .addEventListener('click', () => {
  alert('You cliked me 😀');
});
```

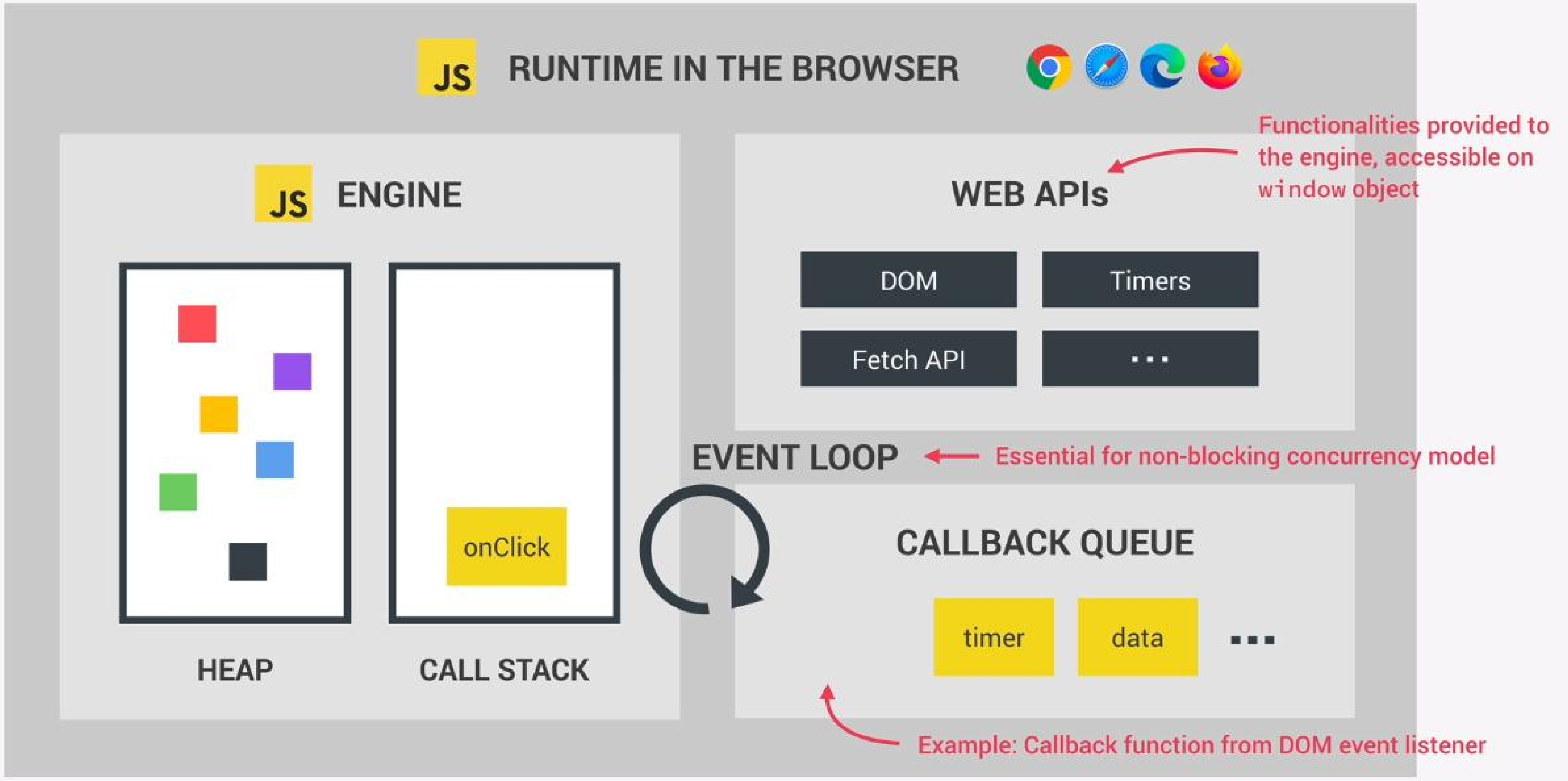
```
127.0.0.1:8080 says
You cliked me 😀
```



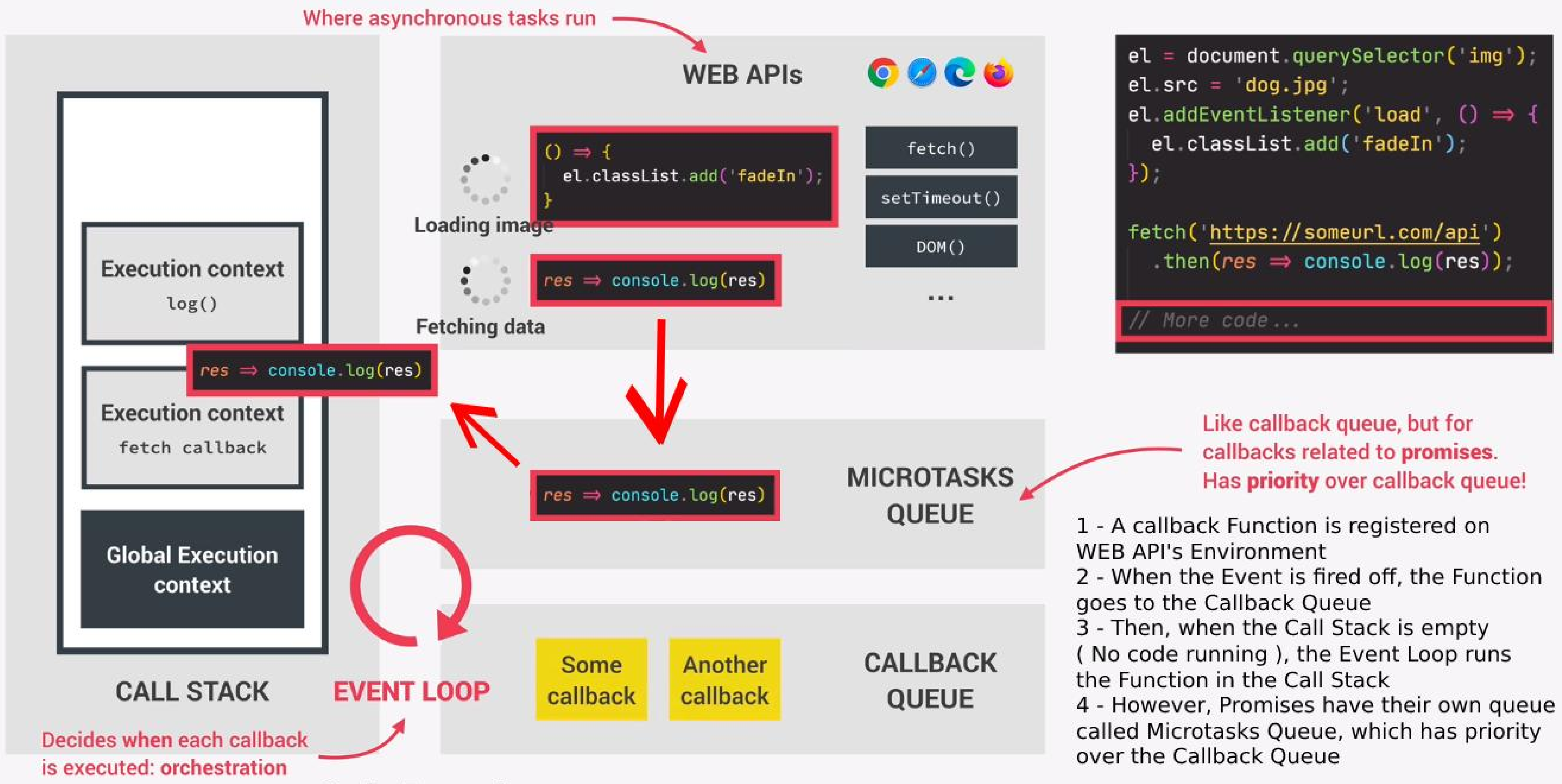
```
document
  .querySelector('a')
  .addEventListener('click', () => {
  alert('You cliked me 😀');
});
```

```
127.0.0.1:8080 says
You cliked me 😀
```

THE BIGGER PICTURE: JAVASCRIPT RUNTIME


JS RUNTIME IN THE BROWSER
JS ENGINE
HEAP
onClick
CALL STACK
WEB APIs
Functionalities provided to the engine, accessible on window object
DOM
Timers
Fetch API
...
EVENT LOOP
Essential for non-blocking concurrency model
CALLBACK QUEUE
timer
data
...
Example: Callback function from DOM event listener

HOW ASYNCHRONOUS JAVASCRIPT WORKS BEHIND THE SCENES



```
el = document.querySelector('img');
el.src = 'dog.jpg';
el.addEventListener('load', () ⇒ {
  el.classList.add('fadeIn');
});

fetch('https://someurl.com/api')
  .then(res ⇒ console.log(res));
// More code ...
```

1 - A callback Function is registered on WEB API's Environment
2 - When the Event is fired off, the Function goes to the Callback Queue
3 - Then, when the Call Stack is empty (No code running), the Event Loop runs the Function in the Call Stack
4 - However, Promises have their own queue called Microtasks Queue, which has priority over the Callback Queue

Code Example:
1º - console.log('Start');
4º - setTimeout(() => ... , 0); //Will be delayed due lack of priority
3º - Promise.resolve('Resolved promise 1').then(...takes 5 seconds...);
2º - console.log('End');