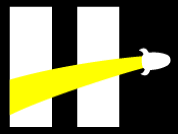
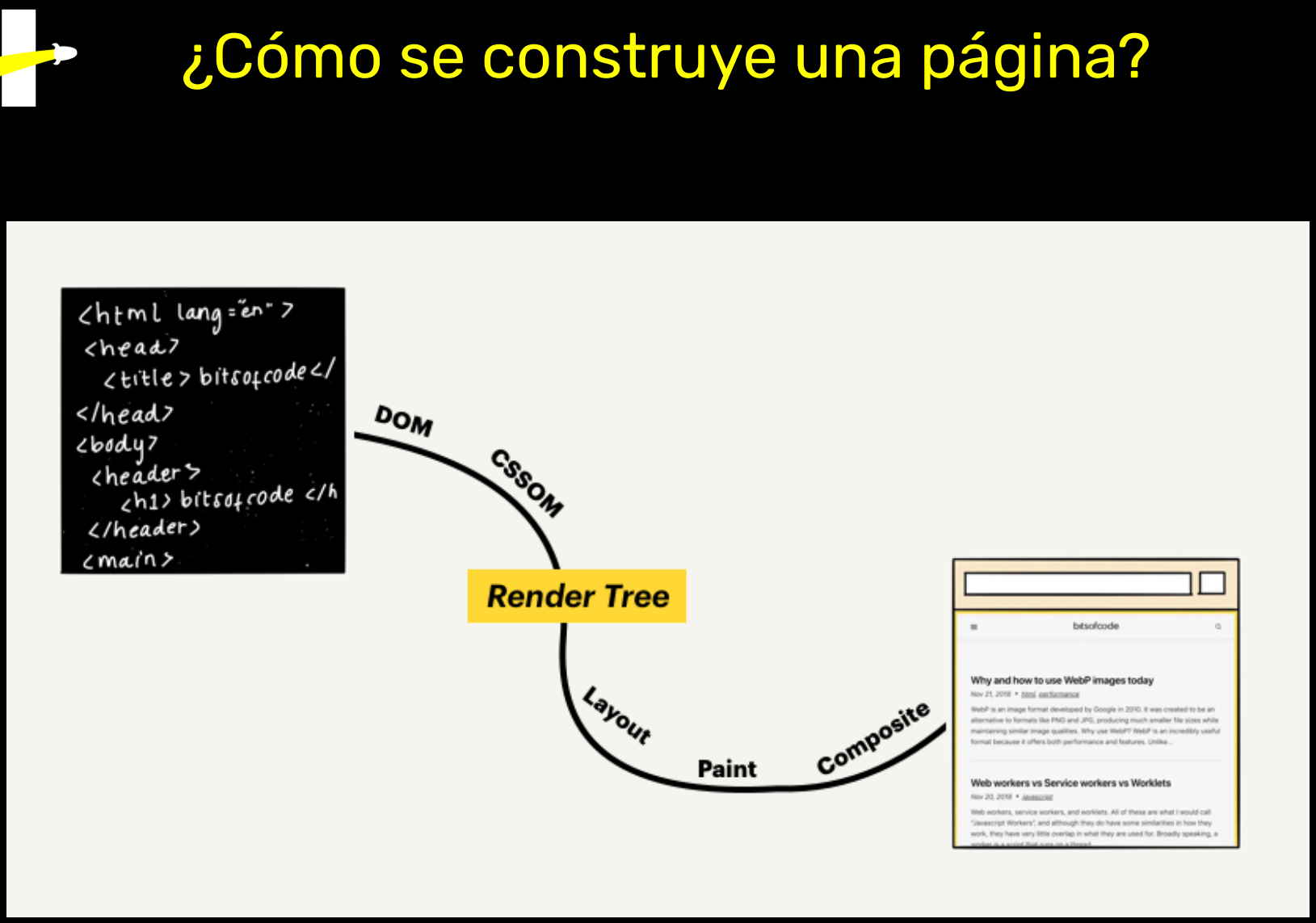
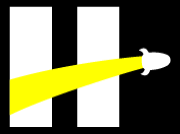

HENRY



DOM


<html lang="en">
<head>
<title> bitsofcode</
</head>
<body>
<header>
<h1> bitsofcode </h
</header>
<main>
DOM
CSSOM
Render Tree
Layout
Paint
Composite
bitsofcode
Why and how to use WebP images today
Web workers vs Service workers vs Worklets



¿Cómo se construye una página?

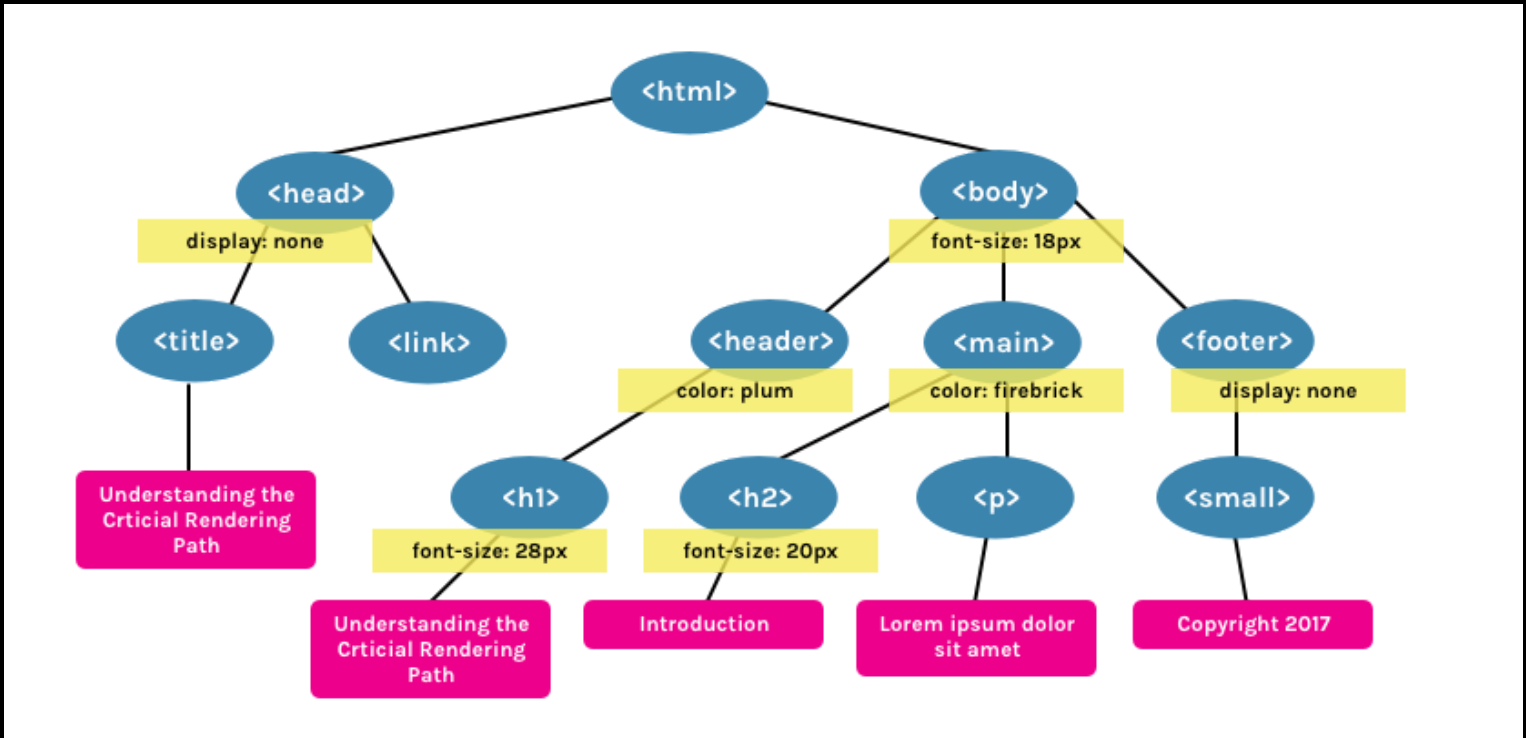
1. Construcción del DOM
2. Construcción del CSSOM
3. Ejecuta JavaScript
4. Creación del Render Tree
5. Generación del Layout
6. Painting

Construcción del DOM

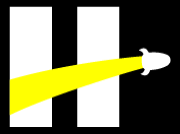
```
1  <!doctype html>
2  <html lang="en">
3   <head>
4     <title>My first web page</title>
5    </head>
6   <body>
7      <h1>Hello, world!</h1>
8      <p>How are you?</p>
9    </body>
10 </html>
```

El DOM (Document Object Model) es una representación en un Objeto de la página HTML parseada.

Construcción del CSSOM



EL CSSOM (CSS Object Model) es un Objeto representando los estilos asociados al DOM.

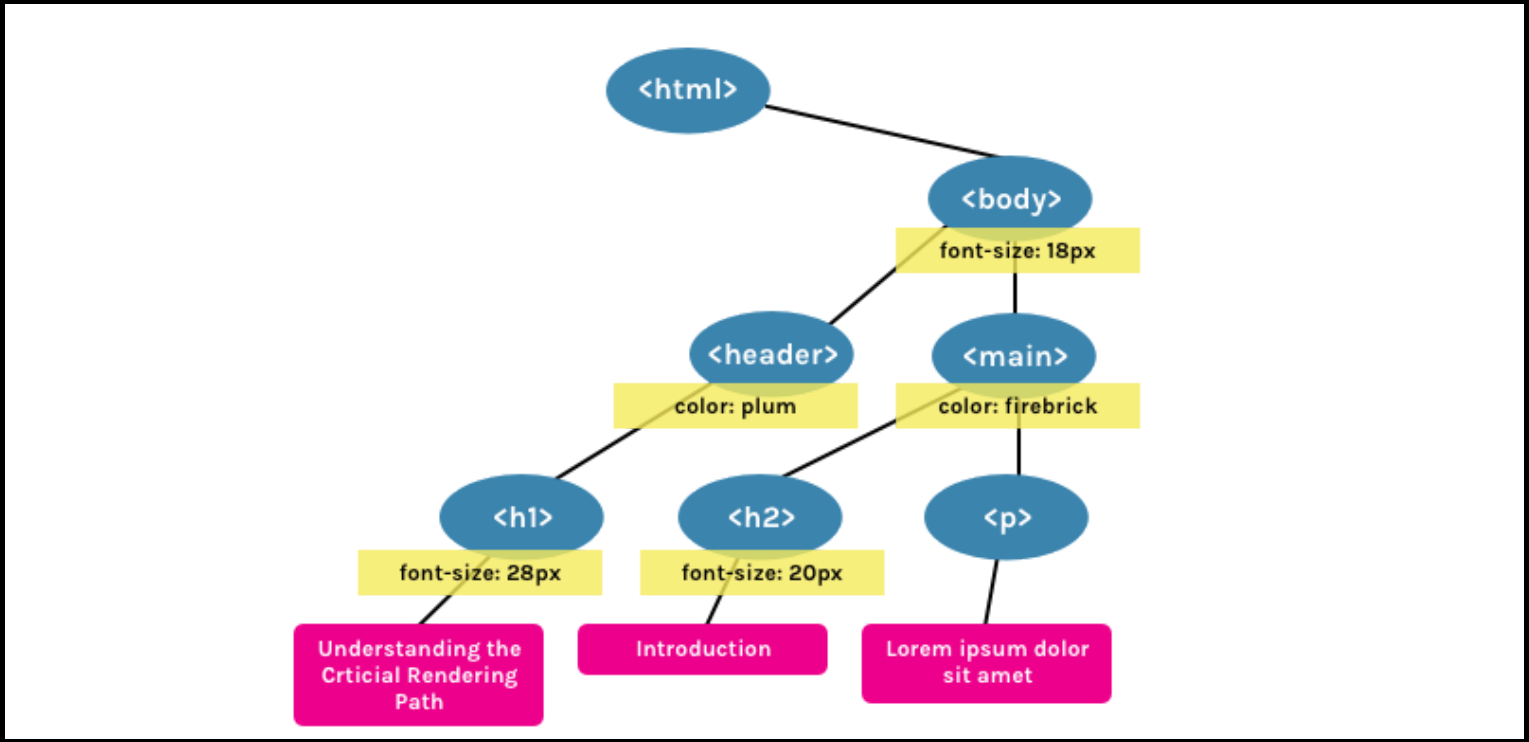


Ejecutando JS

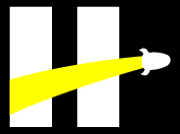
JavaScript bloquea el parseo del DOM. Cuando el parser llega a un tag <script /> frena para poder traer ese recurso y ejecutarlo.

Por eso, cuando agregamos scripts de JS, lo hacemos al final del documento HTML

Creando el render Tree



El render Tree es una combinación del DOM y el CSSOM, en donde se deja sólo los elementos visibles.

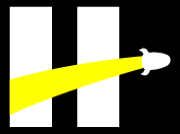


Generando el Layout

El Layout es lo que determina el tamaño del viewport, cree da el contexto necesario para calcular los estilos que depende de él, por ejemplo: % o vw units.

El View port se puede configurar a traves del tag meta viewport

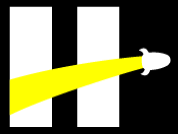
```
1 <meta name="viewport" content="width=device-width,initial-scale=1">
```



Painting

Finalmente, la parte visible de la página se convierte en pixels que se muestran en la pantalla.

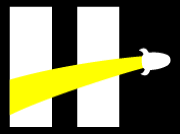
El tiempo de pintado depende del tamaño del DOM, como los estilos que se le aplican.



<Script />

```
1
2 <html>
3   <head>
4       <script>
5         alert('Inyectando código Javascript');
6       </script>
7     </head>
8 </html>
```

```
1
2 <html>
3   <head>
4     <script type="text/javascript" src="./index.js" async></script>
5   </head>
6 </html>
```

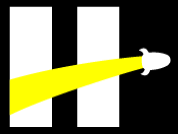


DOM API

El *browser* nos da una *API* para interactuar con el DOM usando JavaScript

Esta *API* nos permite

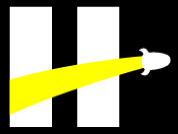
- inspeccionar los elementos y la estructura del documento
- modificar la *estructura*: agregar, modificar o eliminar elementos HTML ó atributos
- modificar el *contenido* del documento
- modificar los *estilos* (CSS)
- agregar o eliminar eventos
- reaccionar a determinados eventos
- etc...



Objeto *document*

Mediante la ejecución de Javascript tenemos la posibilidad de acceder a un objeto global denominado **document** que contiene las propiedades del **DOM** y métodos de su prototipo que nos permiten acceder a los elementos del DOM y manipularlos.

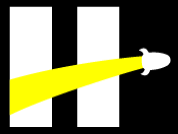
```
document
HTMLDocument https://github.com/atralice/caronte/tree/M2-ajax/M2/00-DOM
  URL: "https://github.com/atralice/caronte/tree/M2-ajax/M2/00-DOM"
  activeElement: <body class="logged-in env-production min-width-lg">
  alinkColor: ""
  all: HTMLAllCollection { 0: html , 1: head , 2: meta , … }
  anchors: HTMLCollection { length: 0 }
  applets: HTMLCollection { length: 0 }
  baseURI: "https://github.com/atralice/caronte/tree/M2-ajax/M2/00-DOM"
  bgColor: ""
  body: <body class="logged-in env-production min-width-lg">
  characterSet: "UTF-8"
  charset: "UTF-8"
  childElementCount: 1
  childNodes: NodeList [ <!DOCTYPE html>, html ]
  children: HTMLCollection { 0: html , length: 1 }
  compatMode: "CSS1Compat"
  contentType: "text/html"
  cookie: "_ga=GA1.2.899944563.1515683265;
```



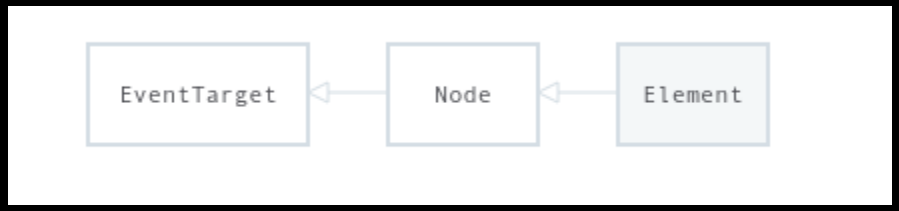
Selectores

```
document.getElementById('user-content-document-selectors')
a#user-content-document-selectors.anchor
    accessKey: ""
    accessKeyLabel: ""
    assignedSlot: null
  ▶ attributes: NamedNodeMap(4) [ id="user-content-document-
  selectors", class="anchor", aria-hidden="true", … ]
    baseURI: "https://github.com/atralice/caronte/tree/M2-ajax/M2
    /00-DOM"
    charset: ""
    childElementCount: 1
  ▶ childNodes: NodeList [ svg.octicon.octicon-link ]
  ▶ children: HTMLCollection { 0: svg.octicon.octicon-link ,
  length: 1 }
  ▶ classList: DOMTokenList [ "anchor" ]
    className: "anchor"
    clientHeight: 24
    clientLeft: 0
    clientTop: 0
    clientWidth: 20
    contentEditable: "inherit"
    contextMenu: null
    coords: ""
  ▶ dataset: DOMStringMap(0)
```

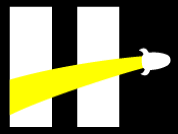
Los **selectores** nos permitiran buscar y recuperar un elemento del DOM. (como cuando buscabamos un elemento en un árbol de búsqueda), sólo que ahora el elemento retornado es un objeto JS que representa una entidad HTML.



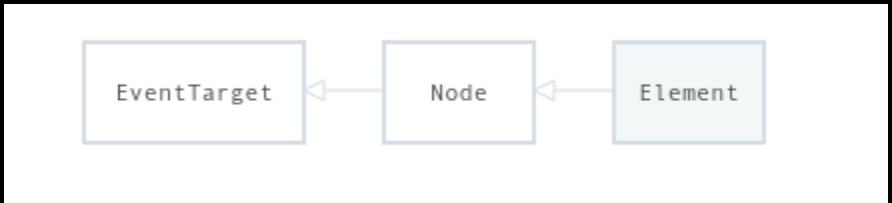
Elements



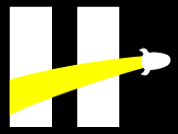
- EventTarget: es una interfaz implementada por los objetos que pueden recibir eventos y pueden tener escuchadores para los objetos.
- Node: es una interfaz en la cuál un número de objetos de tipo DOM API heredan. Esta interfaz permite que esos objetos sean tratados similarmente.
- Element: Representa un elemento HTML.



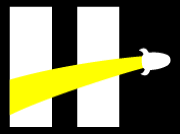
API




```
1
2
3 const divs = document.getElementsByClassName('divClass');
4
5 const div = document.getElementById('divId');
6
7 const div = document.querySelector('.divId');
8
9 const divs = document.querySelectorAll('.divId');
```



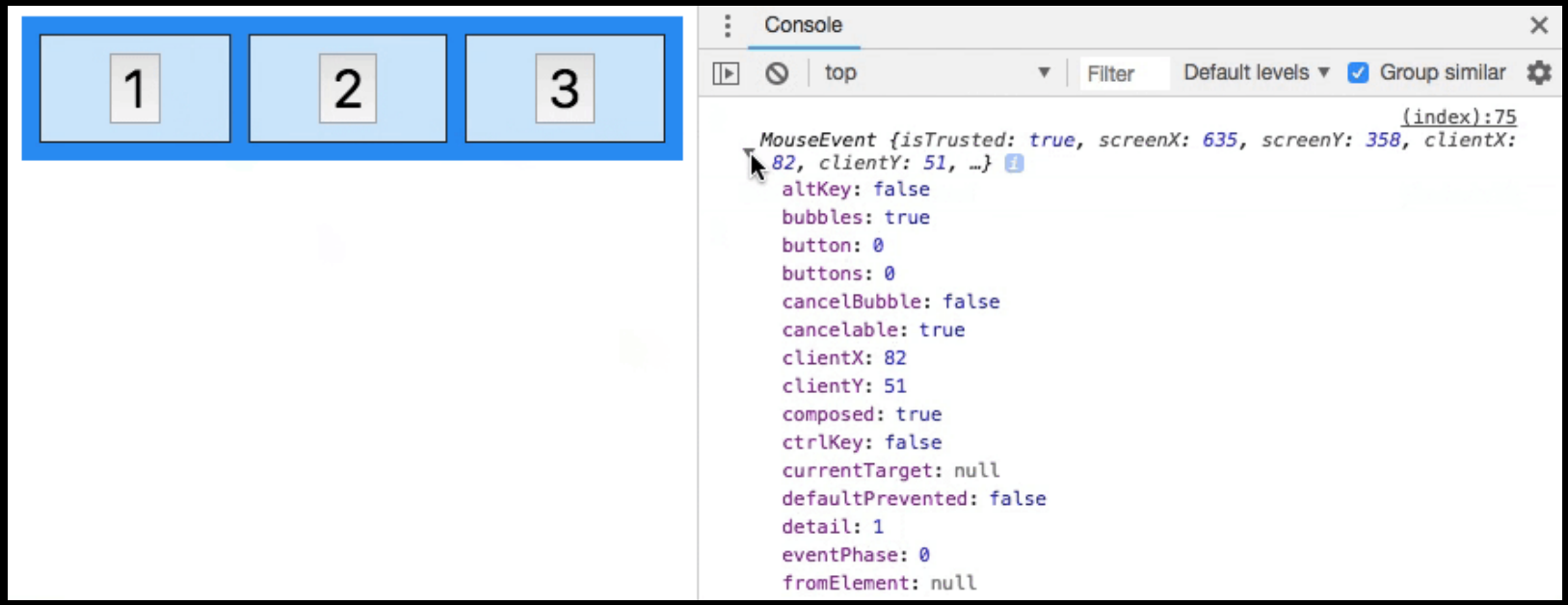
Eventos



Eventos

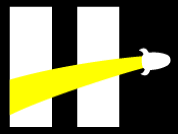
Un *evento* es una señal que algo sucedió. Todos los nodos del DOM pueden generar estas señales.

Un *Event Listener* es el encargado de escuchar por esas señales y hacer *algo.*

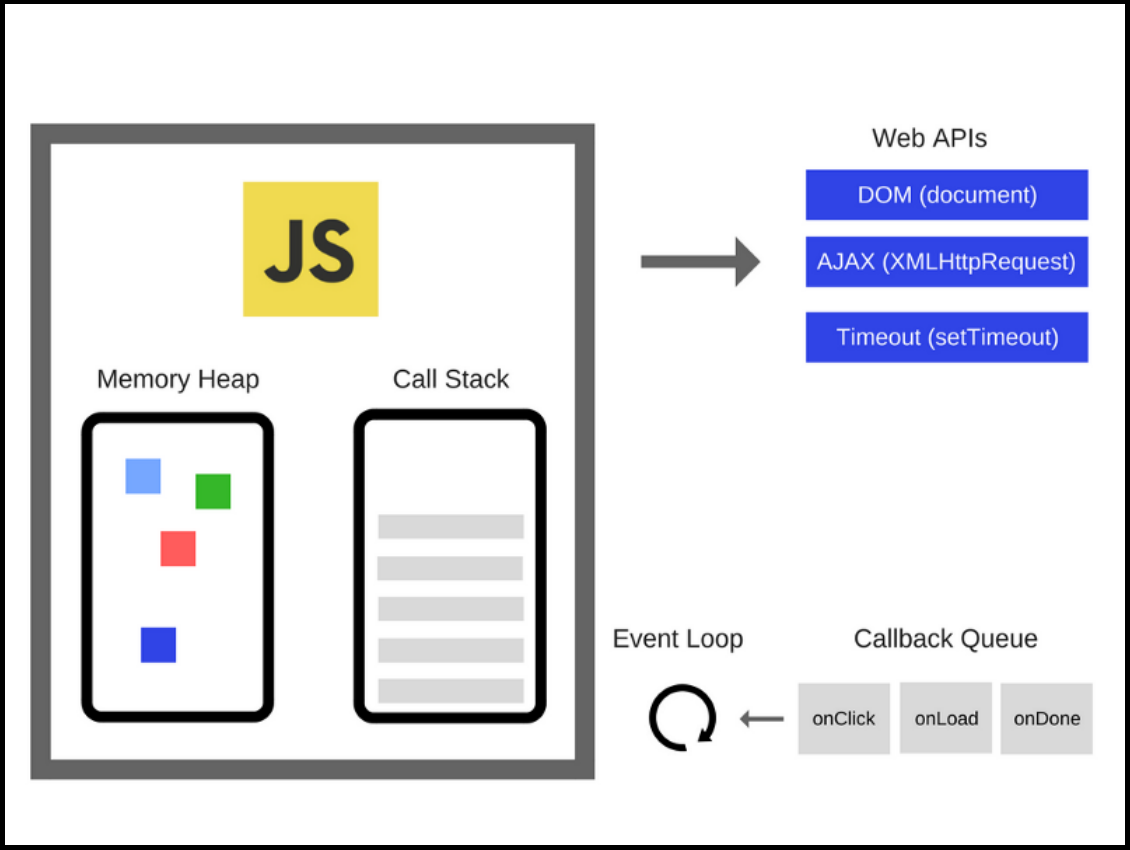


Eventos

Propiedades de los eventos



Event Loop


JS
Memory Heap
Call Stack
Web APIs
DOM (document)
AJAX (XMLHttpRequest)
Timeout (setTimeout)
Event Loop
Callback Queue
onClick
onLoad
onDone