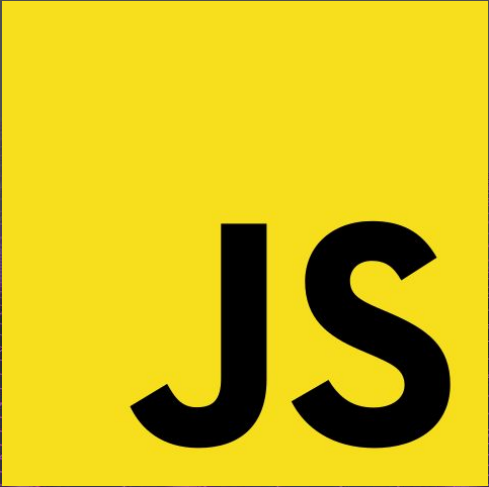


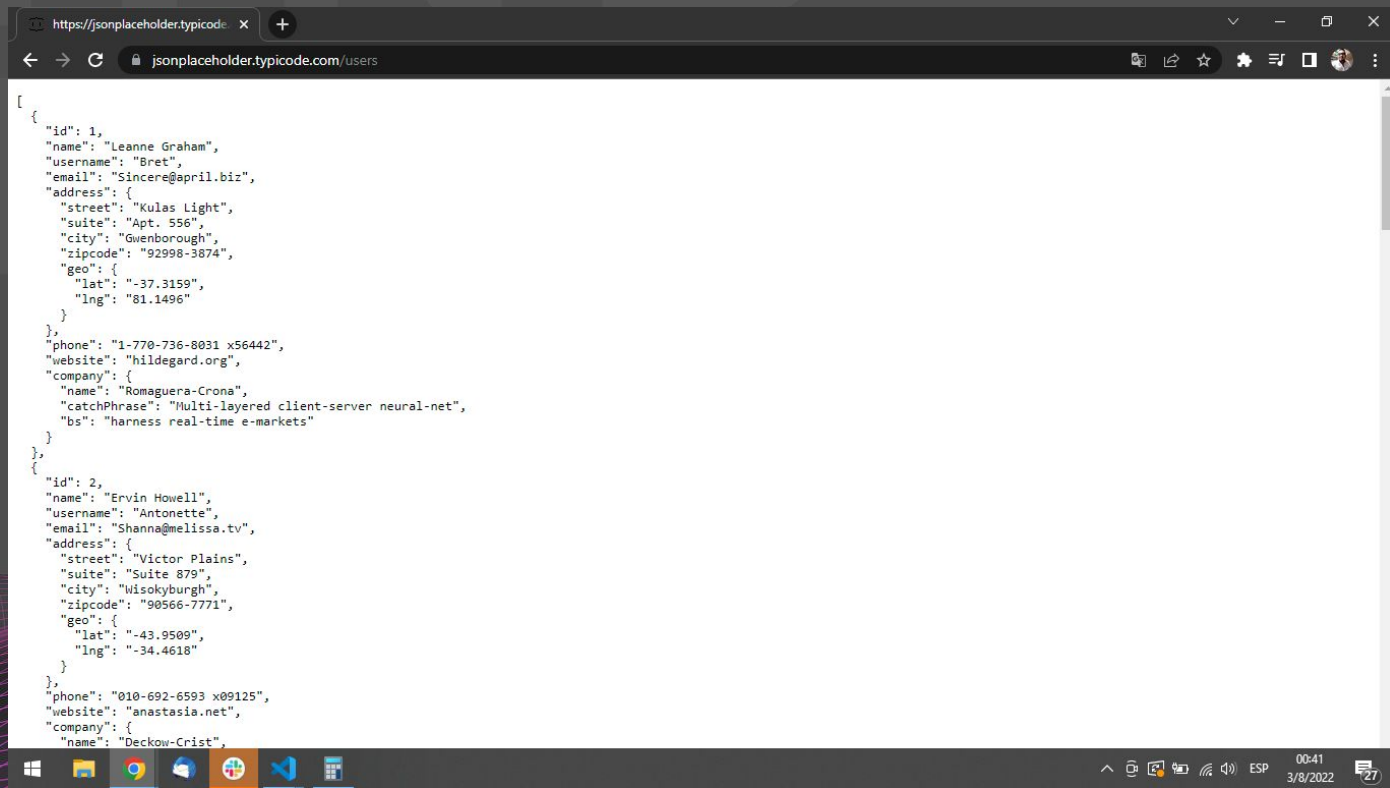
# Consumiendo la primera API

A large yellow square containing the letters 'JS' in a bold, black, sans-serif font, representing JavaScript.

**JS**

**MIND  
HUB.**

# PRIMER PEDIDO A UNA API



```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "username": "Antonette",
    "email": "Shanna@melissa.tv",
    "address": {
      "street": "Victor Plains",
      "suite": "Suite 879",
      "city": "Wisokyburgh",
      "zipcode": "90566-7771",
      "geo": {
        "lat": "-43.9509",
        "lng": "-34.4618"
      }
    },
    "phone": "010-692-6593 x09125",
    "website": "anastasia.net",
    "company": {
      "name": "Deckow-Crist",

```

# PINTAMOS EL DOM

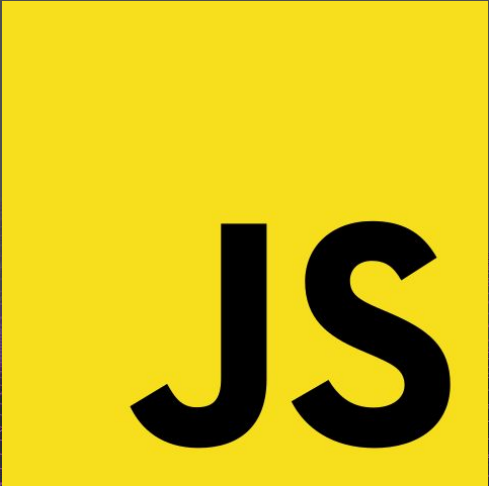


Utilizando Fetch, hicimos un pedido  
GET a

<https://jsonplaceholder.typicode.com/users>

Luego, procesamos la información  
que obtuvimos de la API y la  
pintamos en DOM por medio de JS

# Asincronismo

A large yellow square containing the letters 'JS' in a bold, black, sans-serif font, representing JavaScript.

**JS**



# Antes de empezar: setTimeout

El método `setTimeout` es (ni más ni menos) un temporizador. Este método permite ejecutar un callback luego de que haya transcurrido un determinado tiempo. Los dos parámetros más importantes a tener en cuenta son: la `function` a ejecutar y el `time` (en ms) que debe esperar para ejecutarse

# Sintaxis de setTimeout

```
setTimeout(function (){  
    // Do something  
}, time)
```

In arrow function format

```
setTimeout(() => {  
    //Do Something  
}, time)
```

# Asincronismo

---

Aprender a trabajar con asincronismo en JavaScript nos permitirá ejecutar tareas tengan un tiempo diferido respecto al tiempo de ejecución hasta que se consideren finalizadas.  
Pero ¿por qué necesitamos saber esto?

JavaScript  $\Rightarrow$  Lenguaje Monohilo (Single-thread).

Significa que solo puede hacer una tarea a la vez  
(secuencial)



# Asincronismo

---

Aunque no sea multitarea (mejor llamado multi-thread), puede delegar la ejecución a otros procesos.

## Modelo de concurrencia

Dos o más tareas progresan simultáneamente

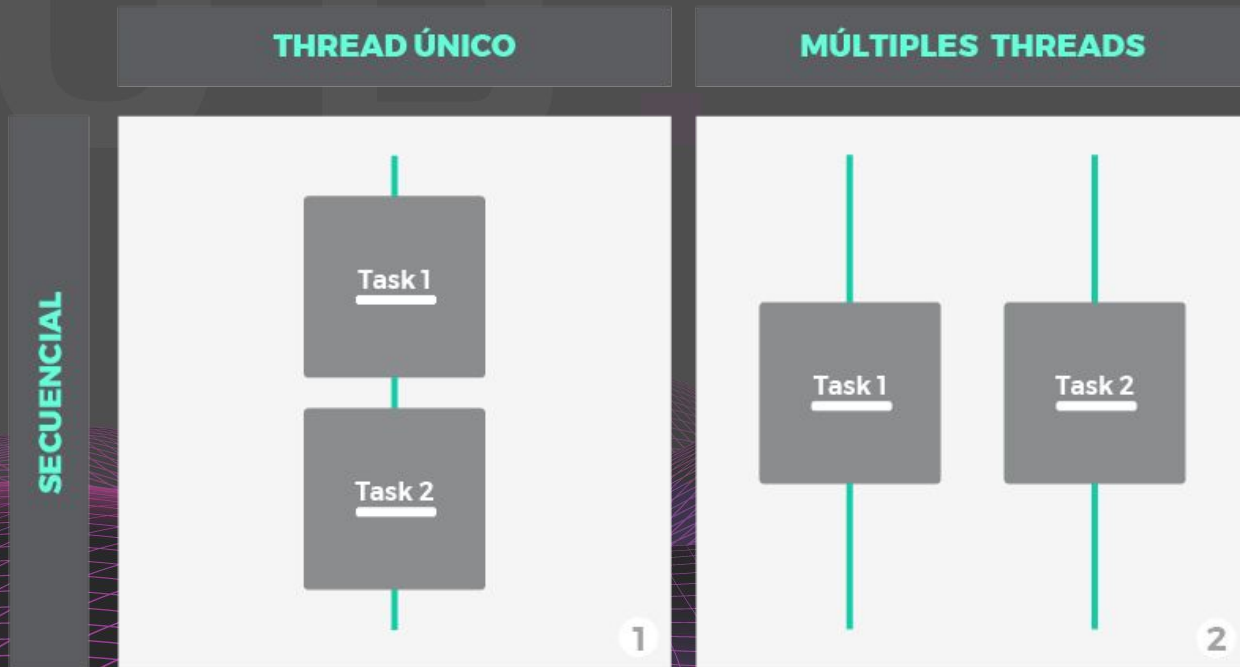
VS

## Modelo de Paralelismo

Dos o más tareas ocurren simultáneamente



# Concurrencia vs Paralelismo

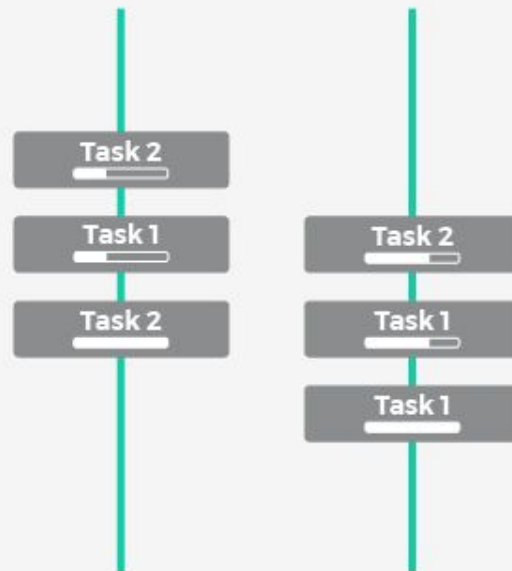


# Concurrencia vs Paralelismo

ENTRELAZADO



3



4

ND  
HUB.

# Entonces ¿redefinimos JS?

---

JavaScript es un lenguaje concurrente, asíncrono, no bloqueante, interpretado, de alto nivel, monohilo.

**Monohilo:** Tiene un único hilo de ejecución

**Concurrente:** Permite el avance de tareas de forma simultánea

**No bloqueante:** Permite derivar procesos para seguir ejecutándose

**Asíncrono:** Ejecuta código que tarda más tiempo "en otro lado"

**MIND  
HUB.**



# Event Loop y Call Stack

---

¿Cómo maneja el asincronismo, la concurrencia y el no-bloqueo si es single-thread?

## Mecanismo Event Loop

JavaScript posee una pila de ejecución llamada **Call Stack** donde coloca las llamadas a funciones según el orden en que deban ejecutarse

Cada línea de ejecución se lee de forma secuencial pero, cuando una función llama a otra, entonces esa tarea se agrega a la pila hasta que termina de ejecutarla y luego la elimina de la pila

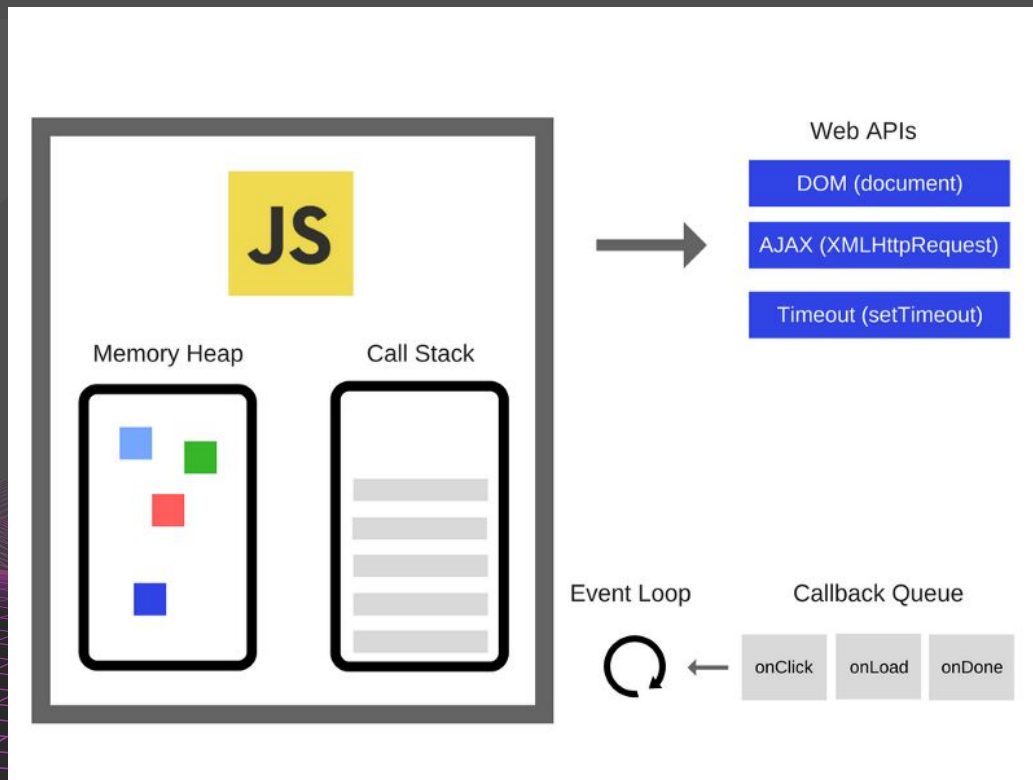
# Callback

---

Un **Callback** es una función que recibe como parámetro a otra función

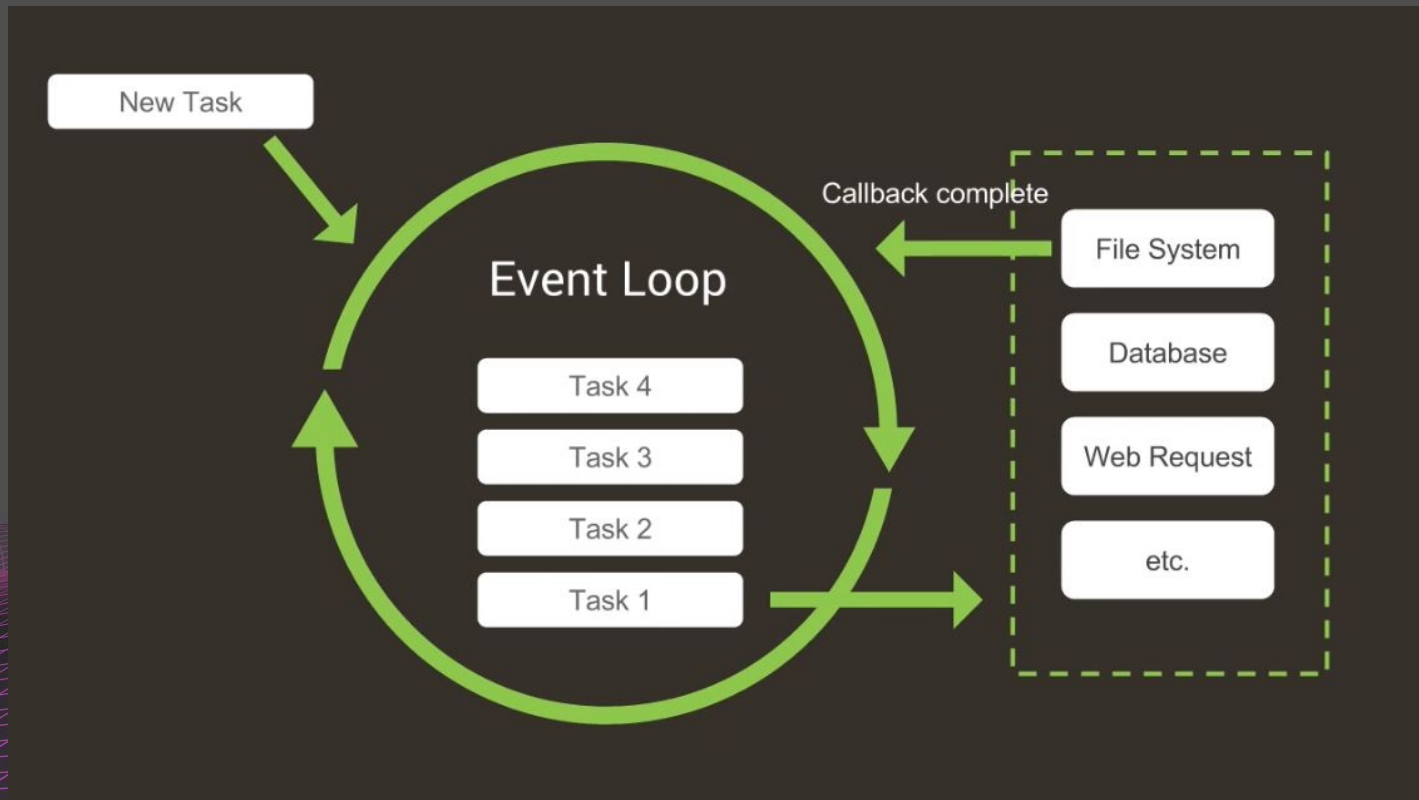
Cuando una función llama a otra función y esta última se resuelve, será agregada al **Callback Queue**. Esto indica que JS sigue ejecutando todas las demás tareas sincrónicas y, cuando se quede sin funciones a ejecutar en la pila de ejecución, allí agregará la información obtenida desde la cola de tareas

# Event Loop





# Event Loop



# Asincronismo en acción

## LET'S GO TO THE CODE

