Laboratorio III
fetch

Clase 3 - p2

Maximiliano Neiner

Temas a Tratar

- Fetch
- Async/await
- CORS

Fetch: peticiones asíncronas

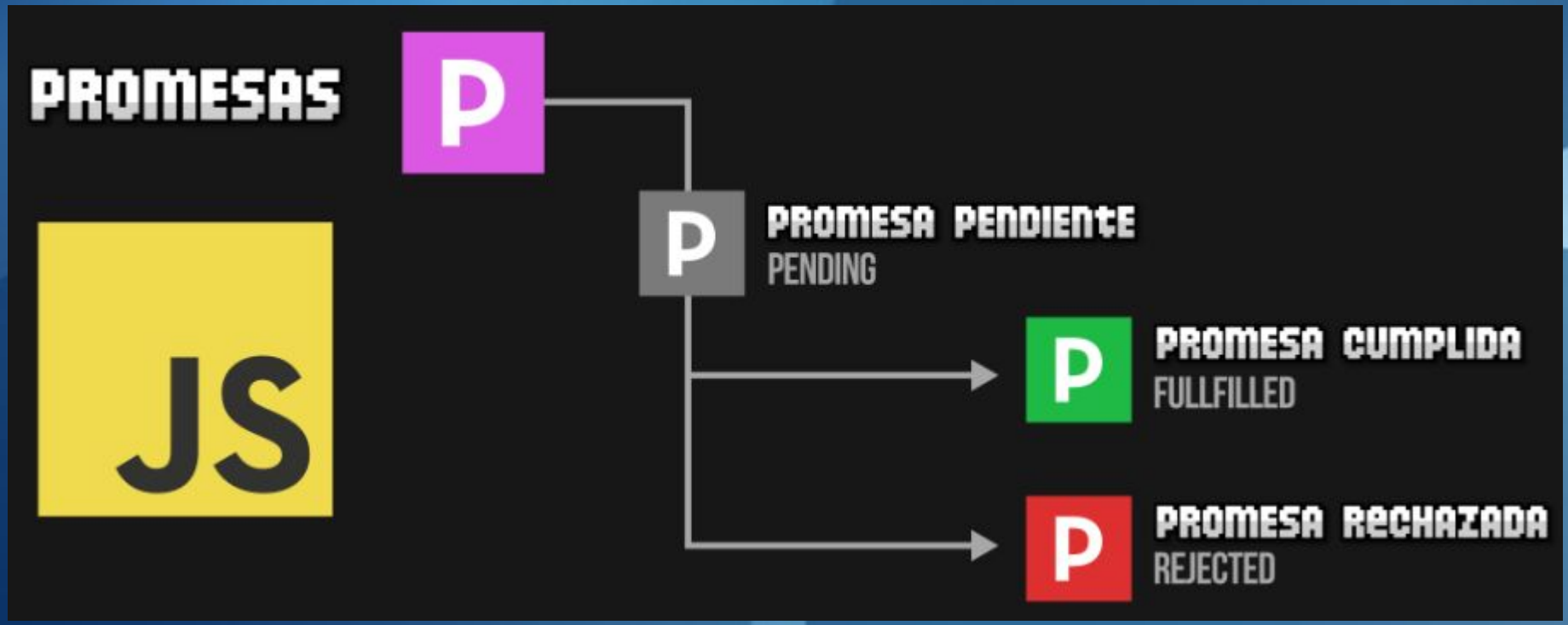
- Fetch es el nombre de una nueva API para javascript con la cual podemos realizar peticiones HTTP asíncronas utilizando **promesas** y de forma que el código sea un poco más sencillo y menos verbose.

```
fetch("Url")
.then(response => {
   //código aquí
});
```

- El fetch() retornará una **promesa** que deberá ser aceptada cuando reciba una respuesta y solo será rechazada si hay un fallo.

Promesas en Javascript (1/2)

- Las promesas se representan mediante un objeto y cada promesa estará en un estado concreto: **pendiente**, **aceptada** o **rechazada**.



Promesas en Javascript (2/2)

- Las promesas poseen los siguientes métodos.

Métodos	Descripción
`.then( FUNCTION resolve)`	Ejecuta la función callback `resolve` cuando la promesa se cumple.
`.catch( FUNCTION reject)`	Ejecuta la función callback `reject` cuando la promesa se rechaza.
`.then( FUNCTION resolve, FUNCTION reject)`	Método equivalente a las dos anteriores en el mismo `.then()`.
`.finally( FUNCTION end)`	Ejecuta la función callback `end` tanto si se cumple como si se rechaza.

```
fetch("Url")
.then(response => response.text())
.then(data => console.log(data))
.catch(err => console.log(err.message))
.finally (()=> console.log("terminado..."));
```

Opciones de fetch()

- La función fetch() recibe opcionalmente un segundo parámetro de opciones, es un *object* con opciones de la petición HTTP.

Campo	Descripción
STRING method	Método HTTP de la petición. Por defecto, GET. Otras opciones: HEAD, POST, etc...
OBJECT headers	Cabeceras HTTP. Por defecto, {}.
OBJECT body	Cuerpo de la petición HTTP. Puede ser de varios tipos: String, FormData, Blob, etc...
STRING credentials	Modo de credenciales. Por defecto, omit. Otras opciones: same-origin e include.

```
const opciones = {
   method: "POST",
   headers: { "Content-Type": "application/json" },
   body: JSON.stringify(data)
};
```

El objeto Response (1/2)

- El objeto *Response* encapsula la respuesta que llega desde el servidor web y contiene una serie de propiedades y métodos.

Propiedad	Descripción
NUMBER .status	Código de error HTTP de la respuesta (100-599).
STRING .statusText	Texto representativo del código de error HTTP anterior.
BOOLEAN .ok	Devuelve `true` si el código HTTP es `200` (o empieza por 2).
OBJECT .headers	Cabeceras de la respuesta.
STRING .url	URL de la petición HTTP.

El objeto Response (2/2)

- Métodos

Método	Descripción
STRING `.text()`	Devuelve una promesa con el texto plano de la respuesta.
OBJECT `.json()`	Idem, pero con un objeto `json`. Equivale a usar `JSON.parse()`.
OBJECT `.blob()`	Idem, pero con un objeto `Blob` (binary large object).
OBJECT `.arrayBuffer()`	Idem, pero con un objeto `ArrayBuffer` (buffer binario puro).
OBJECT `.formData()`	Idem, pero con un objeto `FormData` (datos de formulario).
OBJECT `.clone()`	Crea y devuelve un clon de la instancia en cuestión.
OBJECT `Response.error()`	Devuelve un nuevo objeto `Response` con un error de red asociado.
OBJECT `Response.redirect(url, code)`	Redirige a una `url`, opcionalmente con un `code` de error.

Promesas con async/await

- Usar async/await no es más que lo que se denomina **azúcar sintáctico** (utilizar algo visualmente más agradable que por debajo hace la misma tarea).

- Regla: un *await* solo se puede ejecutar si está dentro de una función definida cómo *async*.

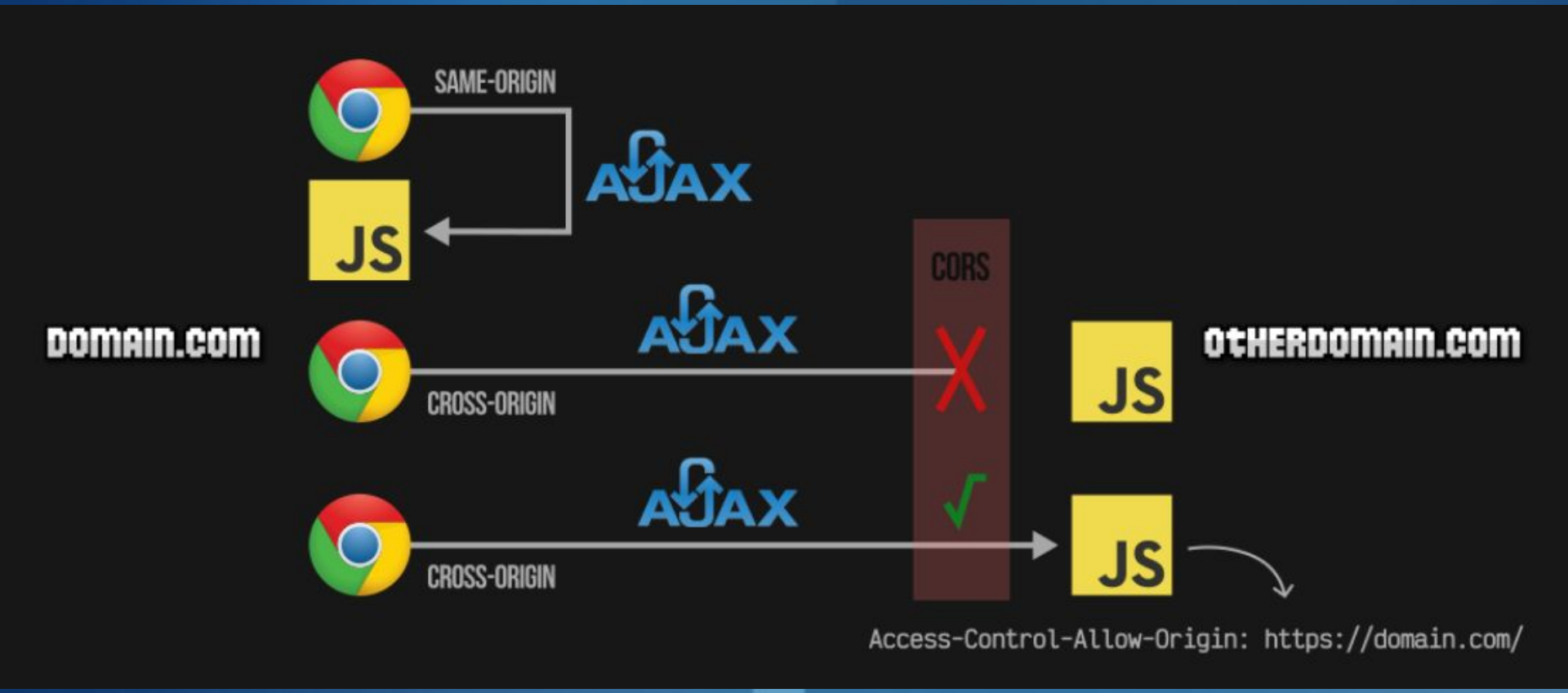
```
const request = async (url) => {
   const response = await fetch(url);
   if( ! response.ok){ throw new Error("error"); }
   return await response.text();
}

const resultado = await request("Url");
const otro_resultado = await request("Url2");
```

Demo

¿Qué es CORS?

- CORS (Cross-Origin Resource Sharing) es una política de seguridad que permite controlar las peticiones HTTP **asíncronas** que se pueden realizar desde un navegador a un servidor con un dominio diferente de la página cargada originalmente.
- Los recursos situados en dominios distintos al de la página actual **no** están permitidos (por defecto) desde peticiones asíncronas (AJAX o fetch).
- A esto se lo suele denominar **protección de CORS**.

¿Qué es CORS?


SAME-ORIGIN
AJAX
JS
CORS
AJAX
DOMAIN.COM
X
JS
OTHERDOMAIN.COM
CROSS-ORIGIN
AJAX
√
JS
CROSS-ORIGIN
Access-Control-Allow-Origin: https://domain.com/

Access-Control-Allow-Origin

- Para evitar la protección de CORS se puede agregar la cabecera Access-Control-Allow-Origin en la respuesta de la petición asincrónica, dónde se debe indicar el dominio al que se quiere dar permiso.

```
Access-Control-Allow-Origin: https://domain.com/
```

- De esta forma el navegador comprobará dicha cabecera y si coinciden con el dominio de origen que realizó la petición, esta se permitirá.

Nota: con el valor * se permitirá cualquier dominio.

Ejercitación