

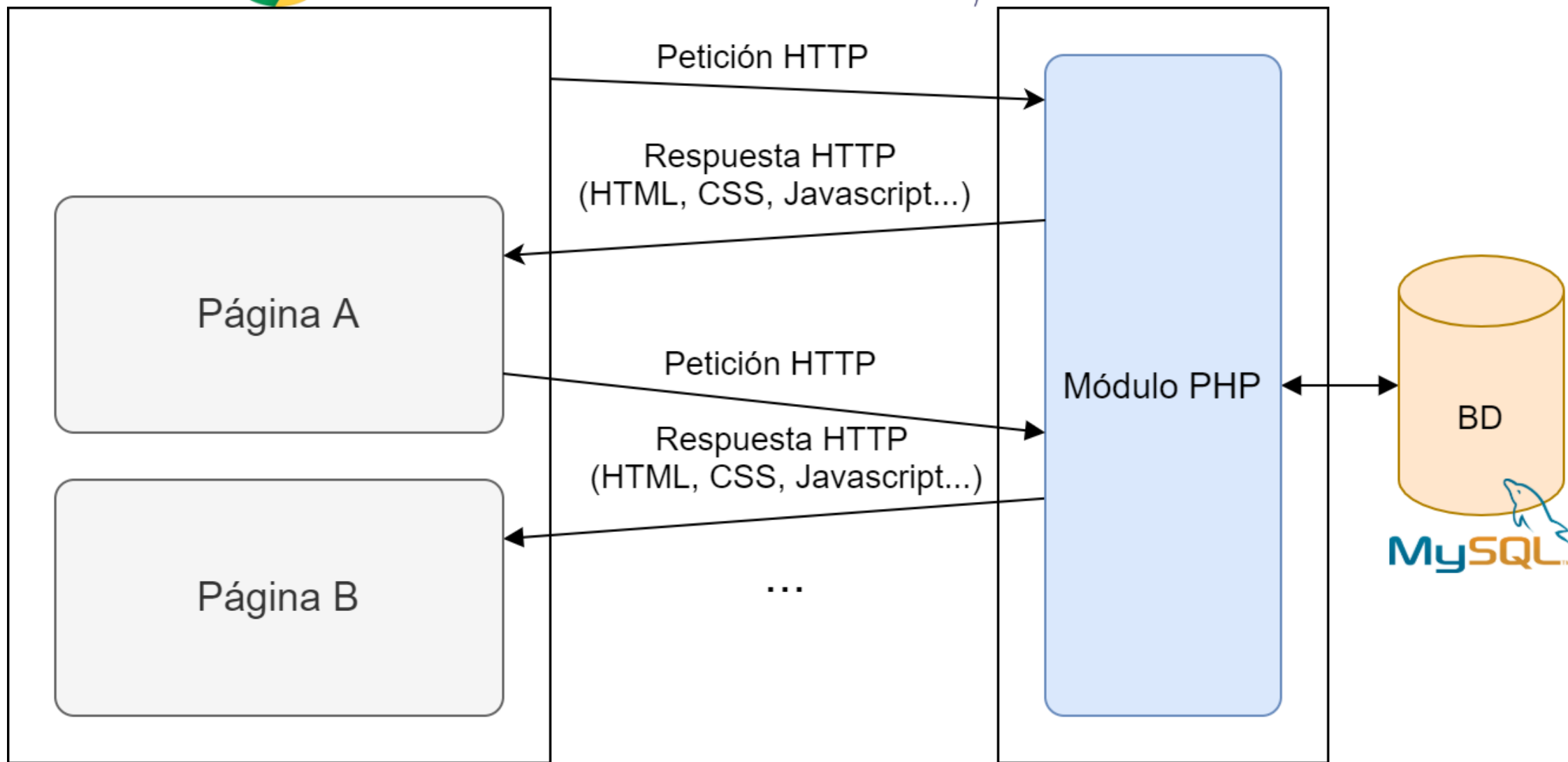
# UT5. Aplicaciones web dinámicas con AJAX

---

DESARROLLO WEB EN ENTORNO SERVIDOR

# Hasta ahora...

---



# Hasta ahora...

---

- Es necesario cargar completamente la página cuando queremos cambiar una pequeña parte.

Soluciones transitorias:



# Hasta ahora...

---

- Es necesario cargar completamente la página cuando queremos cambiar una pequeña parte.  
Soluciones transitorias: ActiveX, Flash Player, frames, iframes,...



# Hasta ahora...

---

- Es necesario cargar completamente la página cuando queremos cambiar una pequeña parte.  
Soluciones transitorias: ActiveX, Flash Player, frames, iframes,...
- Lógica de presentación (etiquetas HTML+CSS)
  - ...mezclada con la de negocio (código PHP)
  - ...mezclada con la de datos (SQL).

# Hasta ahora...

HTML

```
<html>
<head>
  <title>Bienvenido</title>
  <style>* { font-family:sans-serif; } </style>
</head>
<body>
```

CSS

PHP

```
<?php
$cadena_conexion = 'mysql:dbname=tienda;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
  $bd = new PDO($cadena_conexion, $usuario, $clave);
  $sql = 'SELECT CodProducto, Nombre, Marca, Refrigerado FROM productos';
  $productos = $bd->query($sql);
```

SQL

# Con AJAX...

---

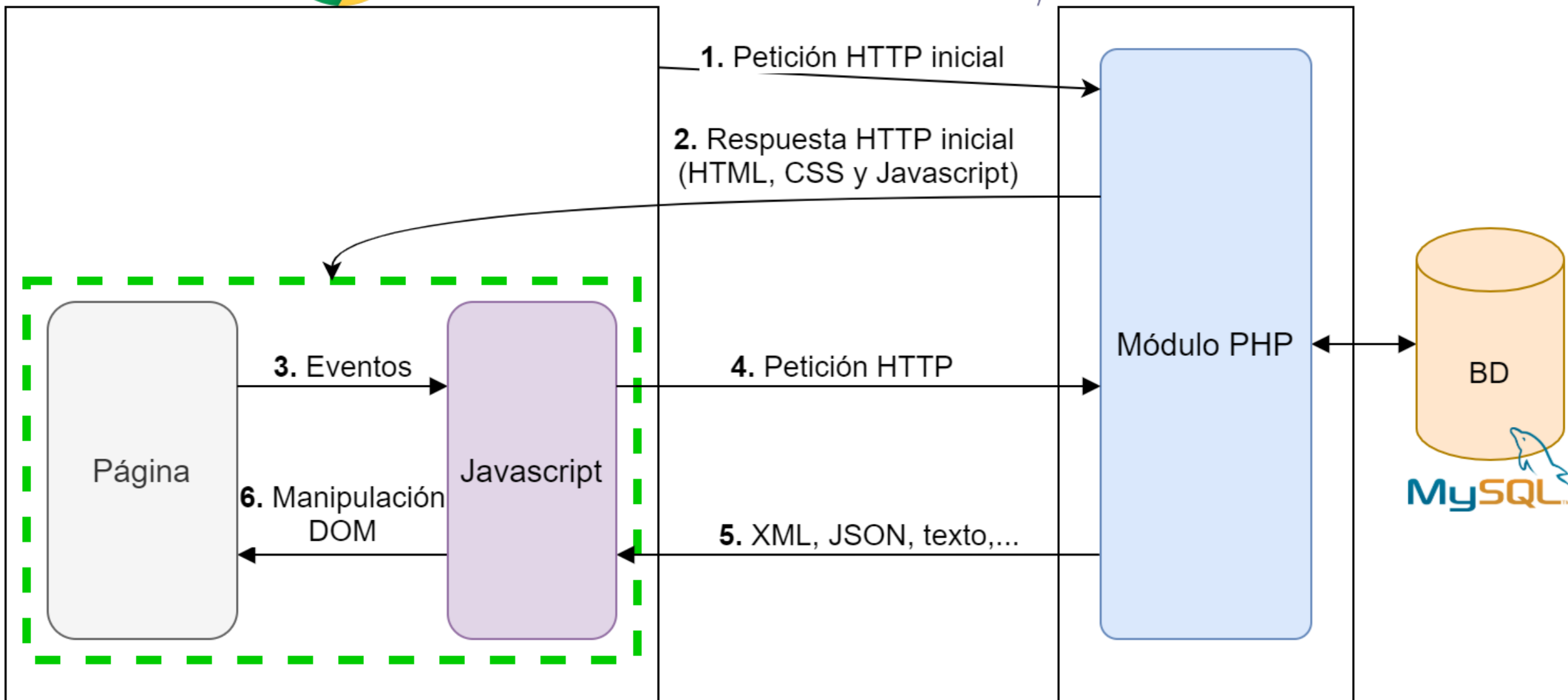


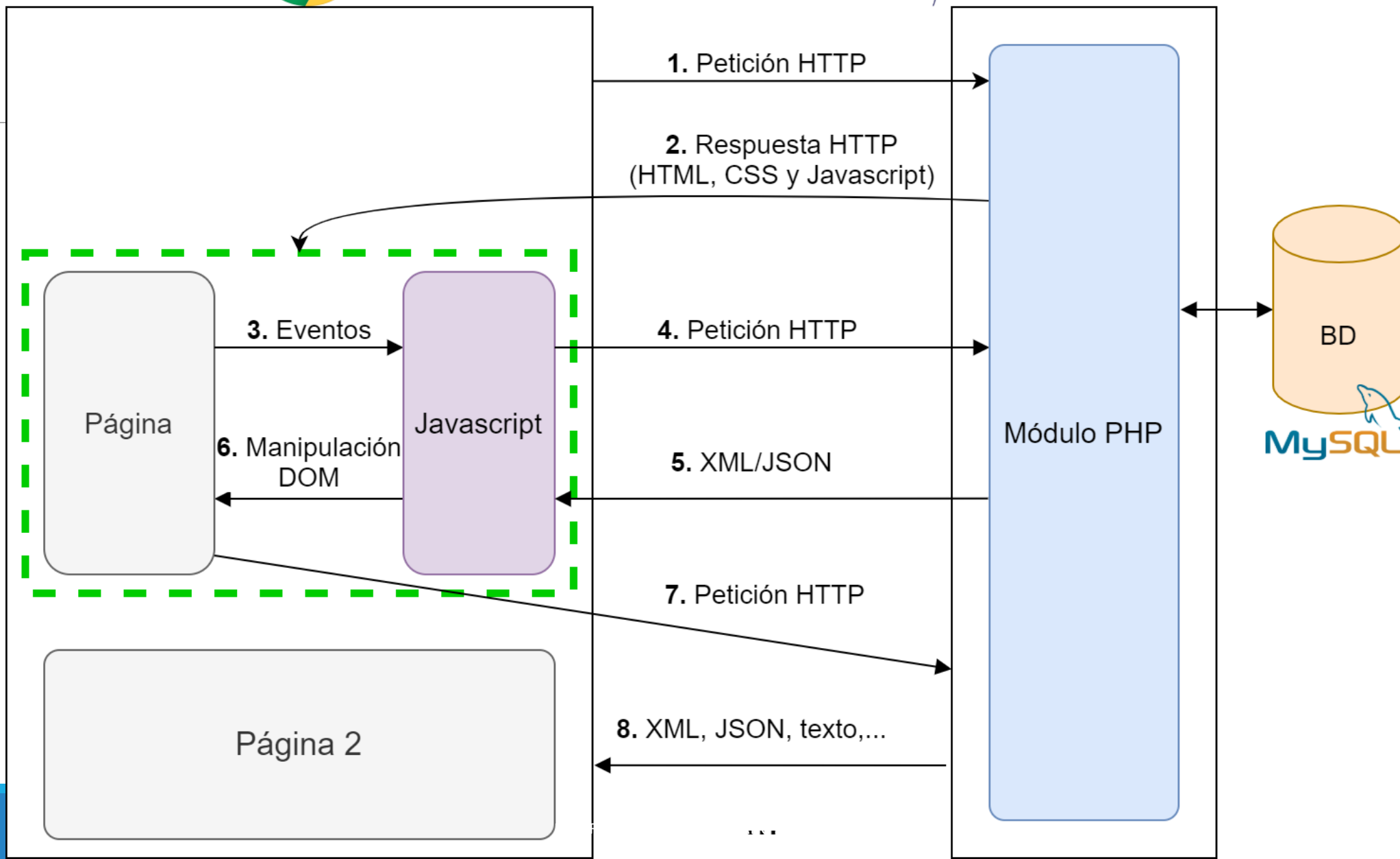
# *Asynchronous JavaScript and XML (AJAX)*

---

**Simplemente:** técnica de diseño de aplicaciones web que se basa en lanzar peticiones al servidor con JavaScript.

- El servidor ahora ya solo devuelve datos a mostrar.
- El cliente “redibuja” con JS la página en base a lo recibido.





# *Asynchronous JavaScript and XML (AJAX)*

---

## **Ventajas:**

- Navegación fluida, sin recargas de página.
- Código más fácil de mantener y modificar.

# Asynchronous JavaScript and XML (AJAX)

---

## Ventajas:

- Código del servidor reutilizable para diferentes clientes (mismo servidor, varias apps).



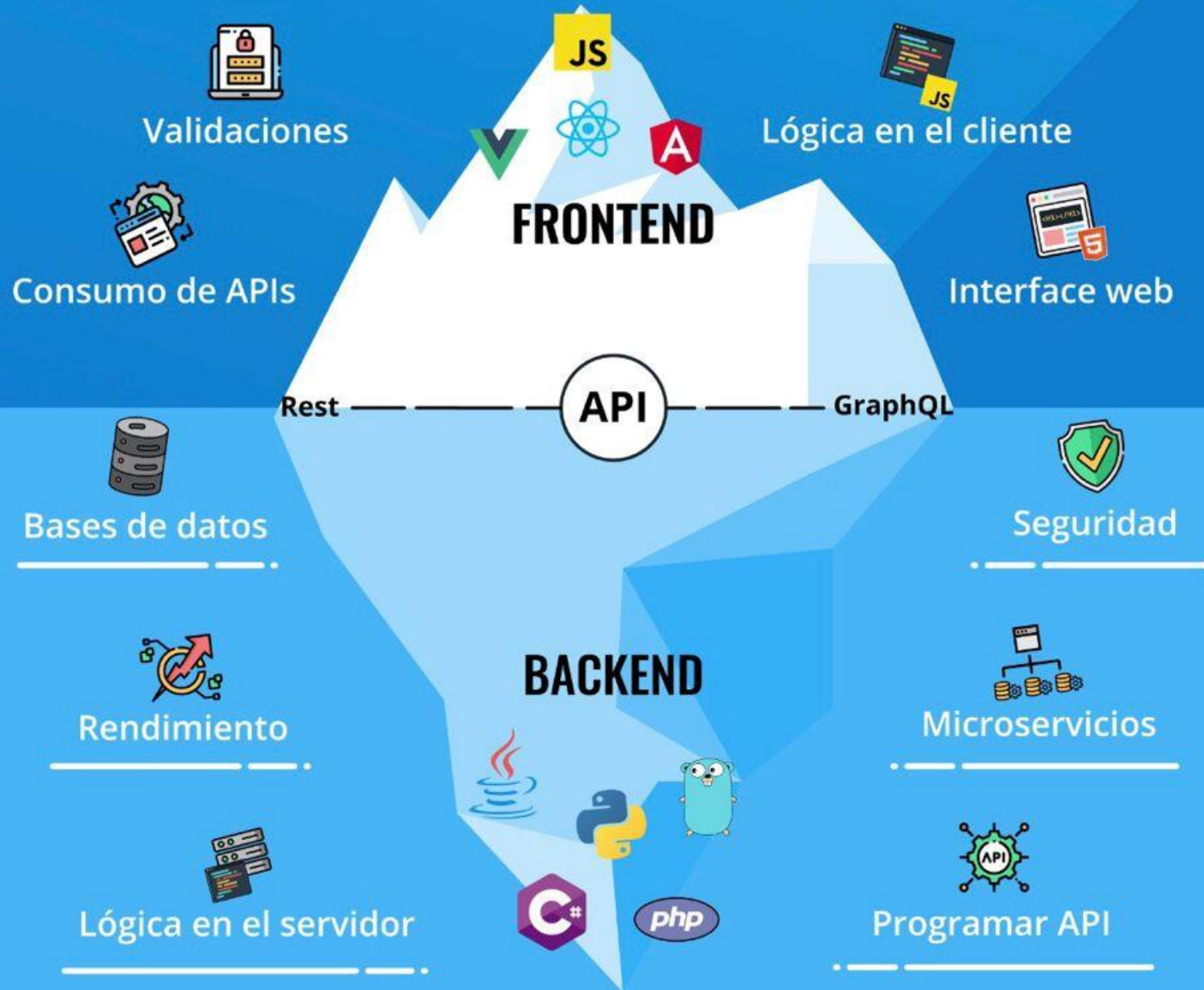
# *Asynchronous JavaScript and XML (AJAX)*

---

## **Ventajas:**

- Separación de equipos de trabajo:
  - Uno trabajando en el cliente → **frontend**.
  - Otro en el servidor → **backend**.

# ¿QUÉ ES BACKEND Y FRONTEND?



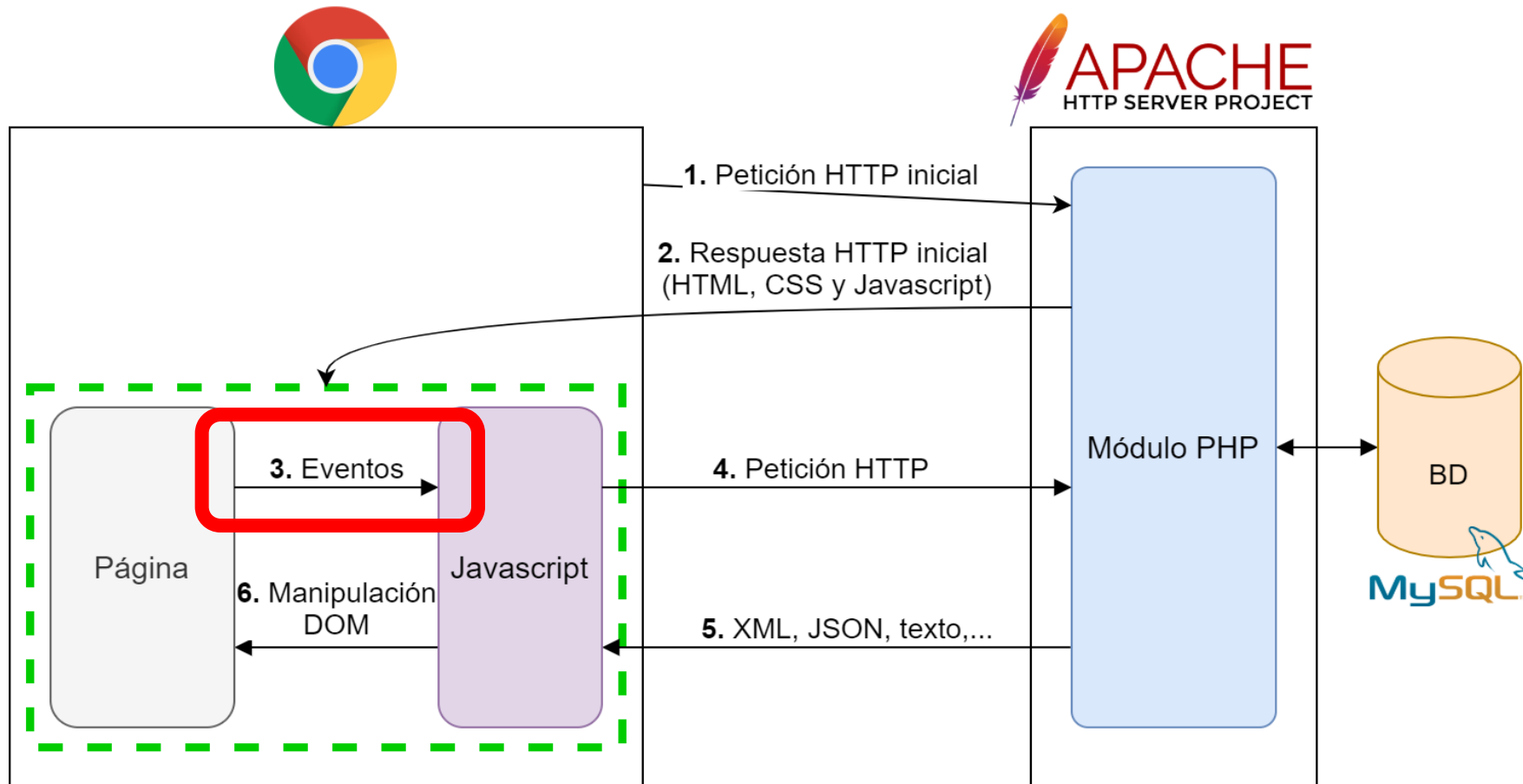
## Backend vs frontend

- La lógica de presentación estará en cliente.
- La de negocio y en el servidor.
- <https://ecdisis.com/que-es-frontend-y-backend/>

# Elementos de AJAX: eventos



# Eventos



# Eventos

---

¿Cuándo hacemos una petición al servidor con AJAX?

- Al cargar la página.

```
window.onload = funcionQueLanzaAjax;
```

- Cada 5 segundos.

```
setInterval(funcionQueLanzaAjax, 1000);
```

- Al pulsar un botón.

```
<button onclick='funcionQueLanzaAjax()'>
```

**Cuando nos convenga en nuestra aplicación.**

# Eventos

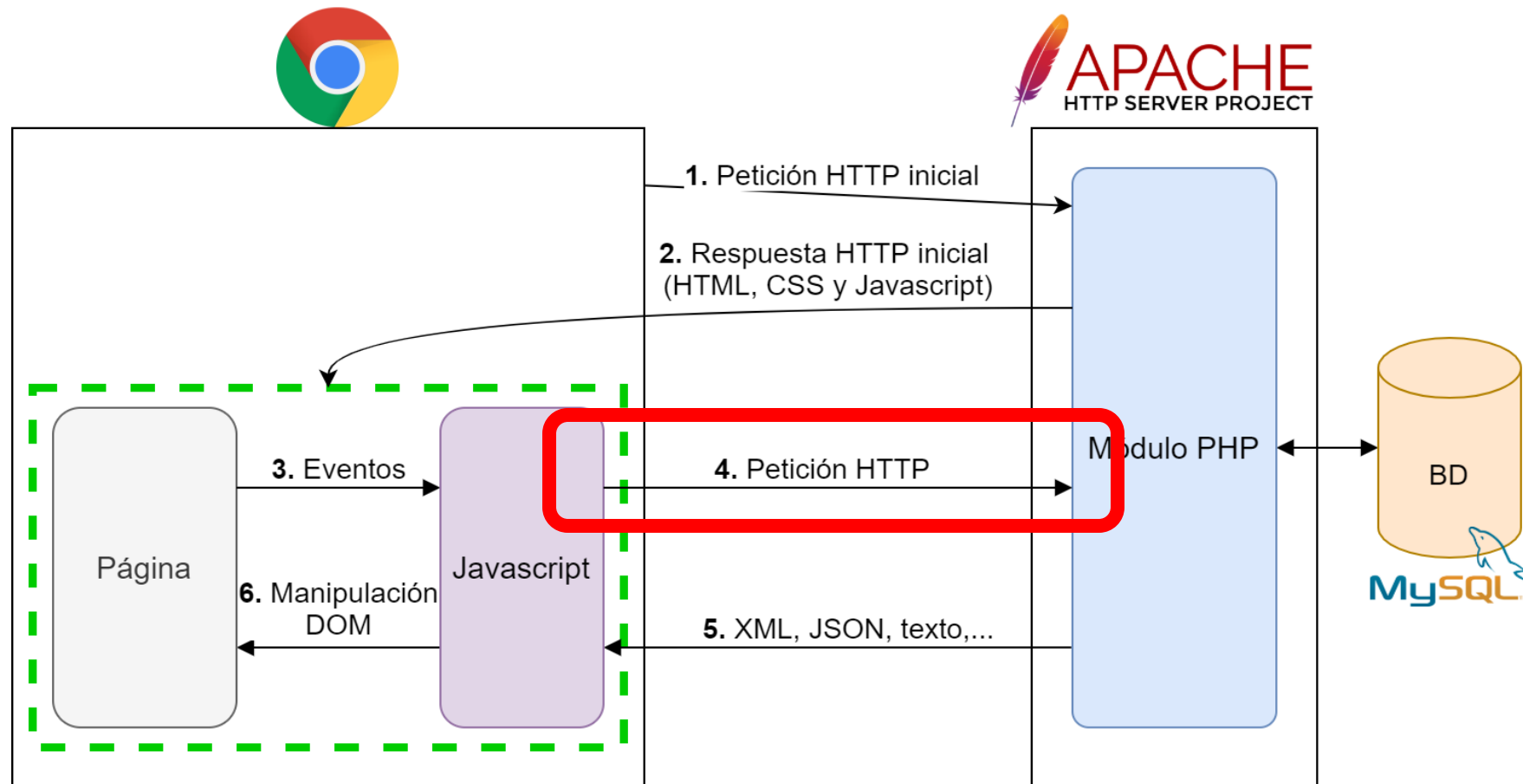
---

Lo habitual es cuando suceda algún **evento**:

- `onclick`
- `onsubmit`
- `onmouseover`
- `onfocus`
- ...

# Elementos de AJAX: la petición HTTP

# La petición HTTP con JavaScript



# La petición HTTP con JavaScript

---

¿Cómo podemos hacer una?

- trae una clase para esto: `XMLHttpRequest`.
- A pesar de su nombre, no implica trabajar con datos XML.
- (Muy) incómoda de usar.

# La petición HTTP con JavaScript – tipos

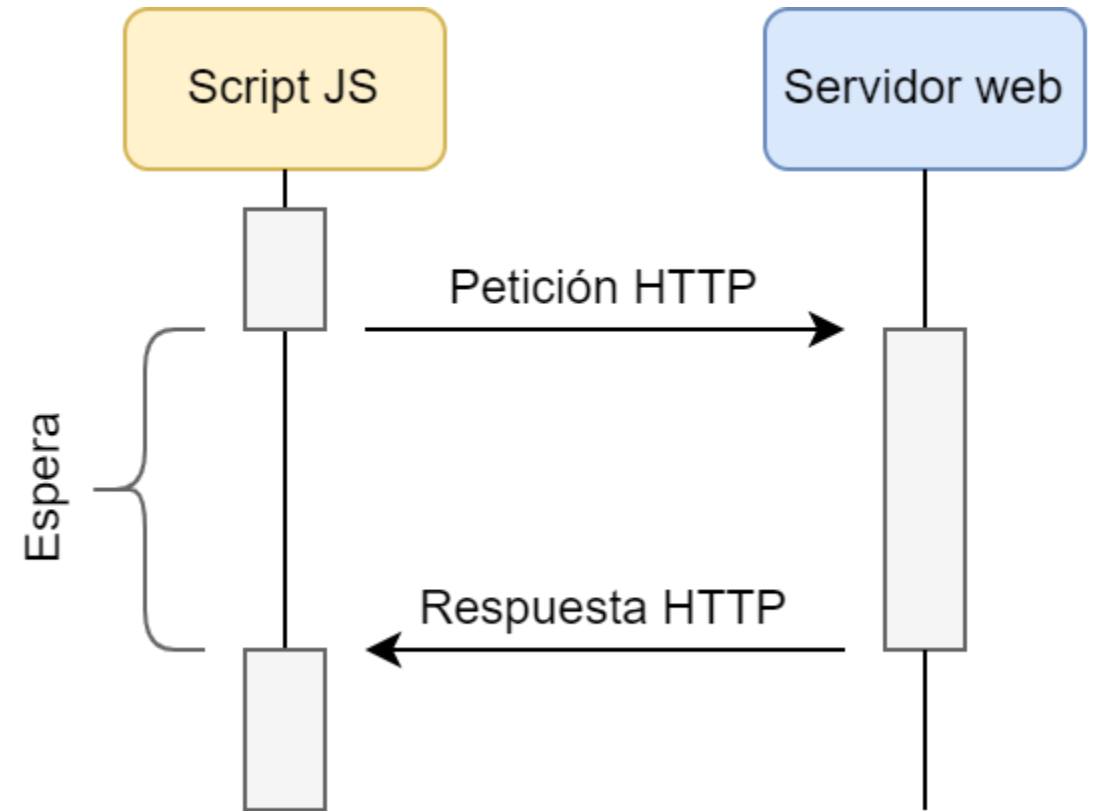
---

- **GET.**
- **POST.**

Además, pueden llevar parámetros o no.

# La petición HTTP con JavaScript – tipos

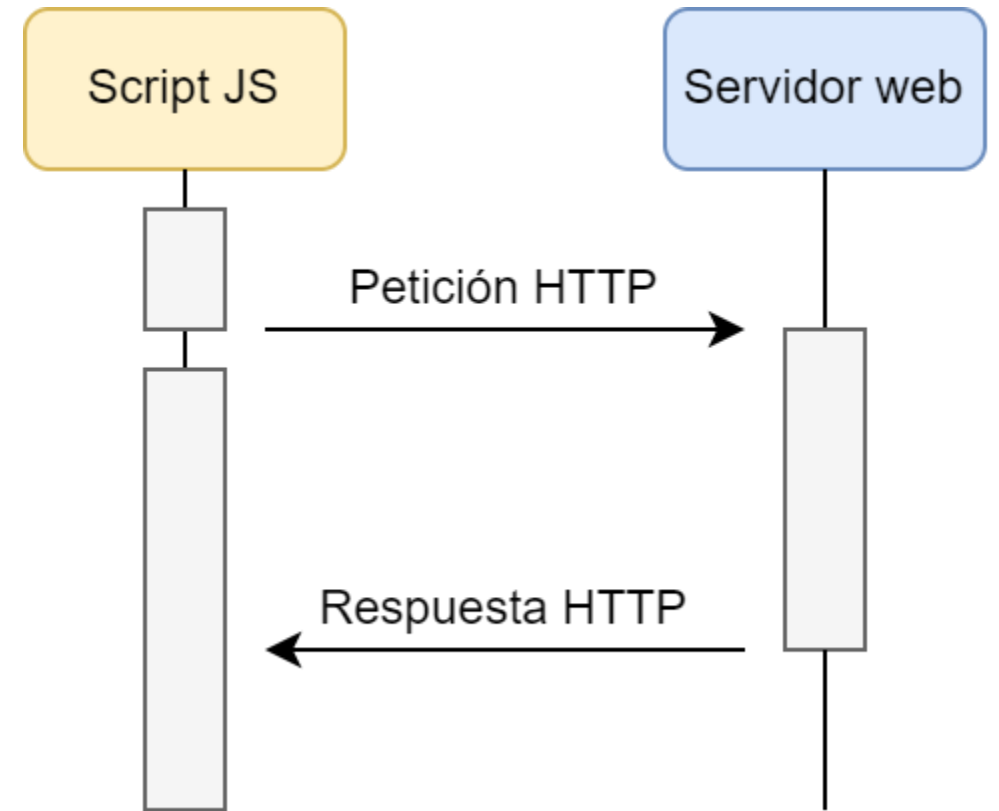
- **Síncronas:** el script se bloquea hasta que se reciba una respuesta.





# La petición HTTP con JavaScript – tipos

- **Asíncronas:** el script continúa, y se ejecutará una función de *callback* cuando llegue la respuesta.



# La petición HTTP con JavaScript – tipos

---

Nos salen 4 posibilidades:

	POST	GET
Síncronas	Síncrona con POST	Síncrona con GET
Asíncronas	Asíncrona con POST	Asíncrona con GET

¿Cuál usamos?

# La petición HTTP con JavaScript – tipos

---

Nos salen 4 posibilidades:

	POST	GET
Síncronas	Síncrona con POST	Síncrona con GET
Asíncronas	Asíncrona con POST	Asíncrona con GET

¿Cuál usamos?

# Antes de nada...

---

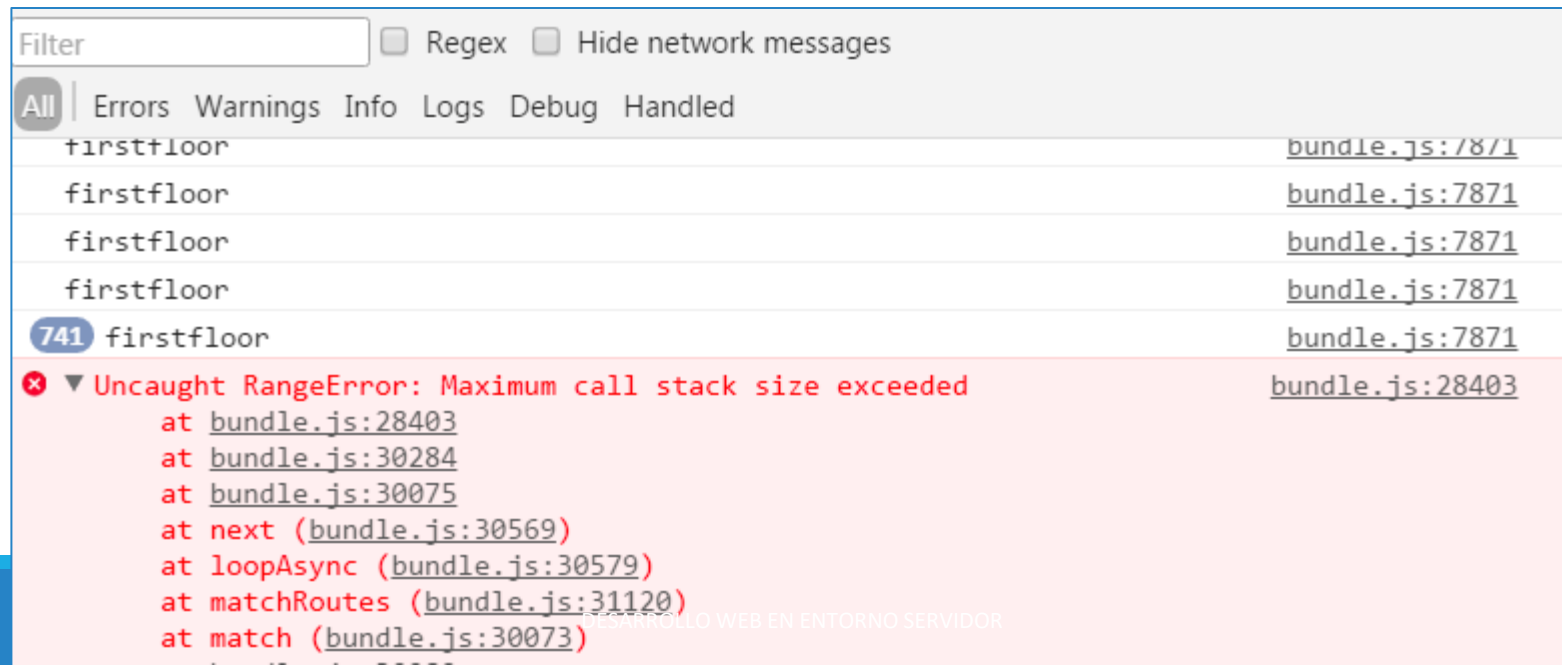
El navegador no avisa si JavaScript falla... ¿cómo lo depuramos?

# Antes de nada...

---

El navegador no avisa si JavaScript falla... ¿cómo lo depuramos?


- Con la consola del navegador.
- Con `console.log()`.



# La clase XMLHttpRequest

Asíncrona GET (sin parámetros):


```
var xhttp = new XMLHttpRequest();  
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200)  
    {  
        alert(this.response);  
    }  
};  
xhttp.open("GET", "time.php", true);  
xhttp.send();
```



# La clase XMLHttpRequest

Asíncrona GET (sin parámetros):

```
var xhttp = new XMLHttpRequest();  
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200)  
    {  
        alert(this.response);  
    }  
};  
xhttp.open("GET", "time.php", true);  
xhttp.send();
```

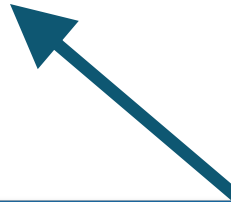


Función de *callback* a ejecutar cuando cambie el estado del objeto

# La clase XMLHttpRequest

Asíncrona GET (sin parámetros):

```
var xhttp = new XMLHttpRequest();  
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200)  
    {  
        alert(this.response);  
    }  
};  
xhttp.open("GET", "time.php", true);  
xhttp.send();
```



Si la respuesta del servidor es 200 y el objeto ha cambiado al estado 4 → todo OK



# La clase XMLHttpRequest

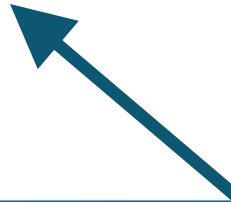
- `xhttp.onreadystatechange` indica **qué función** queremos que se ejecute cuando se reciba la respuesta.
- `xhttp.readyState`: indica **el estado interno de la petición**:

Valor	Nombre	Descripción
0	UNSET	No se se ha llamado a open
1	OPENED	Se ha llamado a open, pero no a send
2	HEADERS_RECEIVED	Se han recibido las cabeceras de la respuesta
3	LOADING	Se está recibiendo el cuerpo de la respuesta
4	DONE	Se ha recibido la respuesta

# La clase XMLHttpRequest

Asíncrona GET (sin parámetros):

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200)
    {
        alert(this.response);
    }
};
xhttp.open("GET", "time.php", true);
xhttp.send();
```



Si la respuesta del servidor es 200 y el objeto ha cambiado al estado 4 → todo OK

# La clase XMLHttpRequest

Asíncrona GET (sin parámetros):

```
var xhttp = new XMLHttpRequest();  
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200)  
    {  
        alert(this.response);  
    }  
};  
xhttp.open("GET", "time.php", true);  
xhttp.send();
```



`this.response` contiene el  
HTML que devuelve el servidor.

# La clase XMLHttpRequest

Asíncrona GET (sin parámetros):

```
var xhttp = new XMLHttpRequest();  
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200)  
    {  
        alert(this.response);  
    }  
};  
xhttp.open("GET", "time.php", true);  
xhttp.send();
```



Inicializa la petición, pero no implica lanzarla.

# La clase XMLHttpRequest

Asíncrona GET (sin parámetros):

```
var xhttp = new XMLHttpRequest();  
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200)  
    {  
        alert(this.response);  
    }  
};  
xhttp.open("GET", "time.php", true);  
xhttp.send();
```

Este `true` indica a JavaScript que la petición es asíncrona.

Inicializa la petición, pero no implica lanzarla.

# La clase XMLHttpRequest

Asíncrona GET (sin parámetros):

```
var xhttp = new XMLHttpRequest();  
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200)  
    {  
        alert(this.response);  
    }  
};  
xhttp.open("GET", "time.php", true);  
xhttp.send();
```



Se envía aquí

# La clase XMLHttpRequest

Asíncrona GET (con parámetros):

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200)
    {
        alert(this.response);
    }
};
xhttp.open("GET", "time.php?id=4&usuario=Laura", true);
xhttp.send();
```

# La clase XMLHttpRequest

Asíncrona POST (sin parámetros):

```
var xhttp = new XMLHttpRequest();  
xhttp.onreadystatechange = function() {  
    if (this.readyState == 4 && this.status == 200)  
    {  
        alert(this.response);  
    }  
};  
xhttp.open("POST", "time.php", true);  
xhttp.send();
```

Basta con cambiar el método



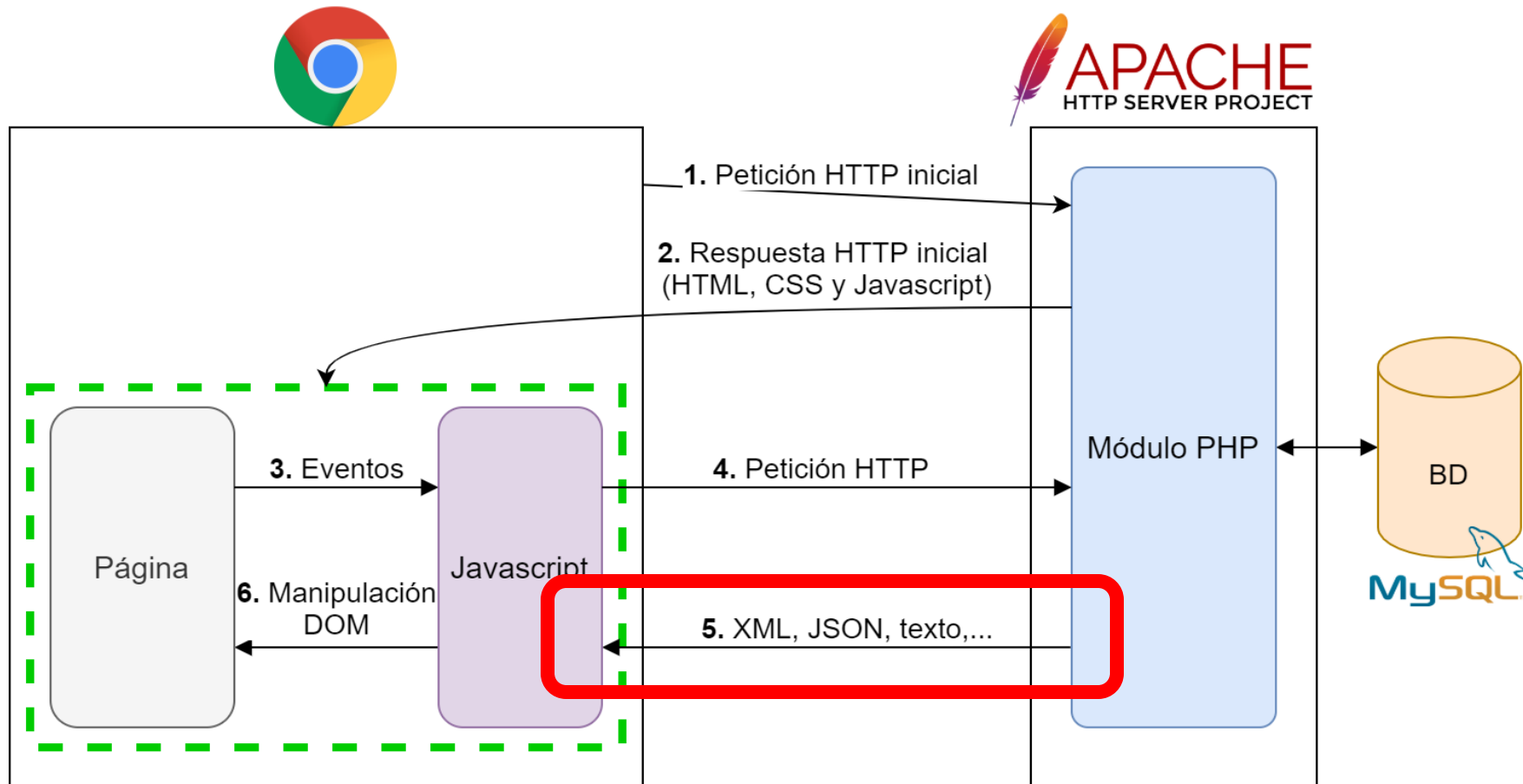
# La clase XMLHttpRequest

Asíncrona POST (con parámetros):

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200)
    {
        alert(this.response);
    }
};
xhttp.open("POST", "time.php", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("id=4&usuario=Laura");
xhttp.send();
```

# Elementos de AJAX: la respuesta

# La respuesta del servidor



# La respuesta del servidor

---

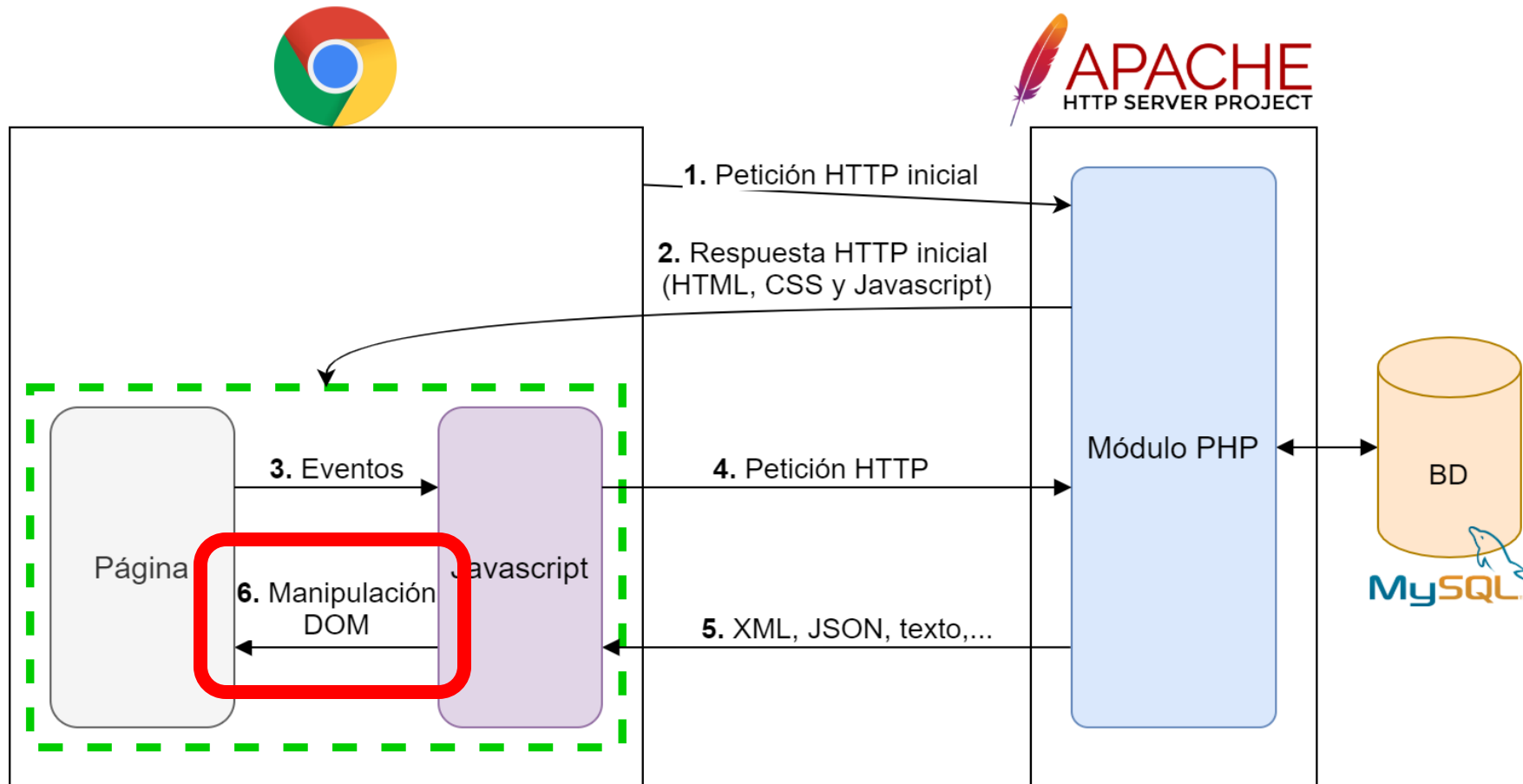
¿Qué puede devolver el servidor?

- Código HTML.
- Texto plano.
- Texto marcado con XML, JSON,...
- La foto de un gato.
- ...

En definitiva → cualquier cosa que pueda imprimir con PHP.

# Elementos de AJAX: manipulando el DOM

# El DOM



# El DOM

---

La **interfaz** que permite manipular un documento HTML.

Un documento HTML es:

- Un árbol.
- En el que los nodos son los elementos del HTML.
- Hay una relación jerárquica entre ellos (padre, hijo, descendiente, antecesor,...)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ejemplo DOM</title>
  </head>
  <body>
  </body>
</html>
```

# El DOM

---

Búsqueda de elementos:

Creación de elementos:



# El DOM

---

## Búsqueda de elementos:

Función	Descripción
<code>document.getElementById(id)</code>	Devuelve el elemento con el ID dado
<code>document.getElementsByClassName(clase)</code>	Devuelve array de elementos con la clase dada
<code>document.querySelector(selector)</code>	Devuelve el primer elemento encontrado usando selectores tipo CSS

## Creación de elementos:

# El DOM

---

## Búsqueda de elementos:

Función	Descripción
<code>document.getElementById(id)</code>	Devuelve el elemento con el ID dado
<code>document.getElementsByClassName(clase)</code>	Devuelve array de elementos con la clase dada
<code>document.querySelector(selector)</code>	Devuelve el primer elemento encontrado usando selectores tipo CSS

## Creación de elementos:

Función	Descripción
<code>document.createElement(etiqueta)</code>	Crea un nuevo elemento
<code>elemento.setAttribute(nombre, valor)</code>	Añade a elemento un nuevo atributo
<code>elemento.innerHTML = "contenido"</code>	Modifica el contenido de elemento

# El DOM

manipular\_DOM.html

## Inserción de elementos:

Función	Descripción
<code>elemento.appendChild(elem_hijo)</code>	Añade a elemento el hijo elem_hijo
<code>elemento.insertAdjacentElement(posición, elem)</code> <code>elemento.insertAdjacentHTML(posición, "&lt;br&gt;")</code>	Añade el elemento elem a elemento en base a lo que indique posición ('beforebegin', etc.)

