



Asynchronous Content

Web, Mobile and Security
Frédéric Vlummens

Agenda

- Synchronous vs asynchronous behavior
- XML HTTP
- JSON
- JSON-P
- SOP
- CORS
- Fetch API

In the beginning of the web...

- ...pages had to refresh completely to show something from the server
- Takes more time (page needs to be re-built completely)
- User sees that the entire page reloads
- You need to preserve the page state between every load

DEMO

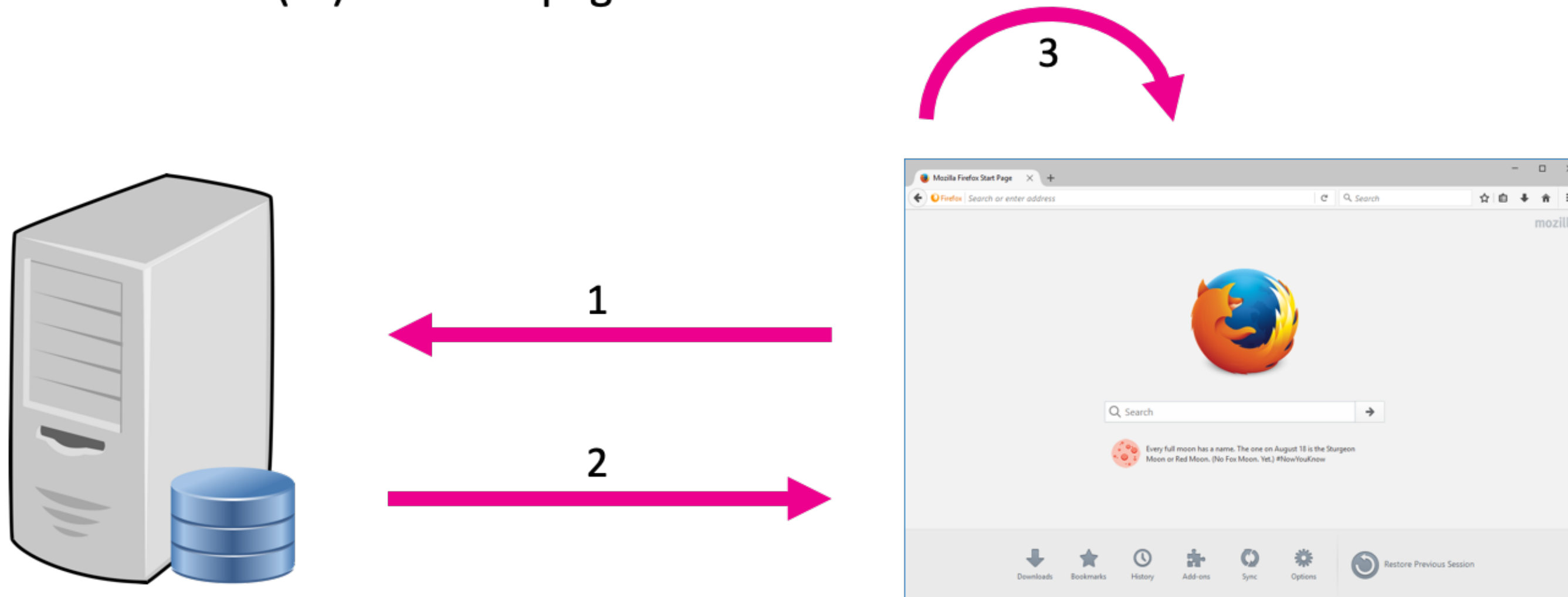
Solution: asynchronous content

- In today's web, pages load content asynchronously.
- Only parts of the pages are updated, instead of re-loading the entire page.
- Results in a better user experience for the visitor.

DEMO

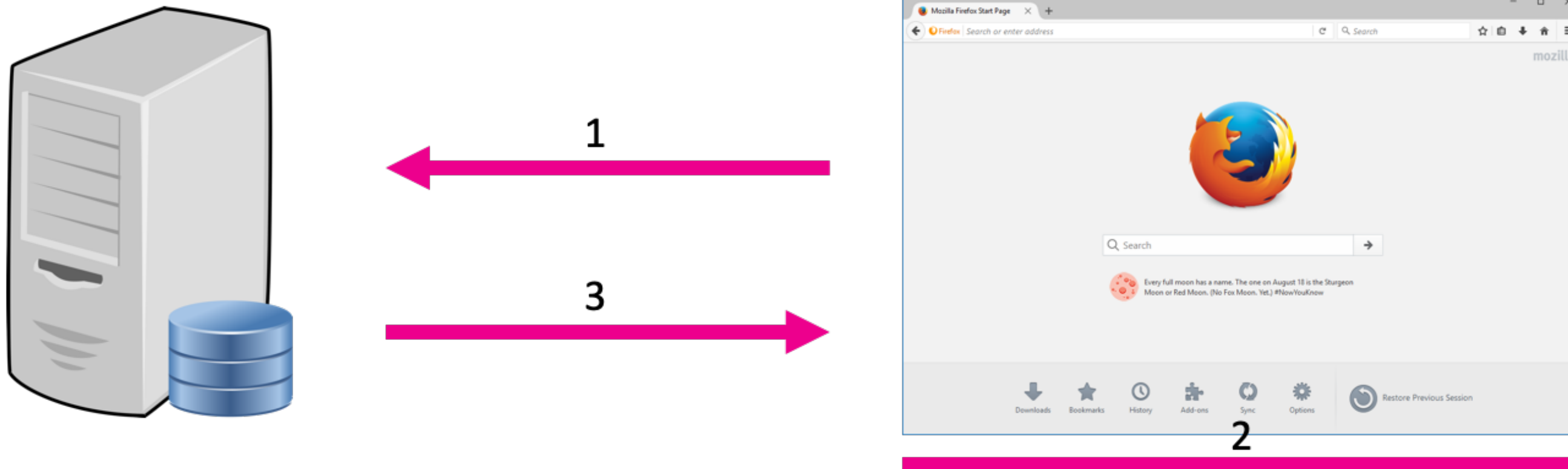
“Regular” behaviour

1. Browser issues a request
2. Browser stops responding until server sends its response
3. Browser (re)loads the page



Asynchronous behaviour

1. Browser issues a request
2. Browser can do other things while waiting for reply
3. Reply gets processed (usually resulting in something being displayed on the page)



Asynchronous behaviour

SIMPLY EXPLAINED



NO AJAX



AJAX

Source: <http://recruitmentmatters.nl/2012/01/29/wekelijkse-cartoon-201/>

Question

- Why don't we load everything asynchronously?
- Since it gives users a better experience...

Question

- Why don't we load everything asynchronously?
- Since it gives users a better experience...
- Higher complexity

How to perform asynchronous operations

- XMLHttpRequests (XHR)
 - Originally developed by Microsoft
 - Adapted by Mozilla, Apple and Google
 - Has been standardized by W3C since
 - Old style
- JSONP
 - Workaround for CORS (see later)
- Fetch
 - New style
 - Promise-based
 - Take into account CORS (see later)

XmlHttpRequests

- Requires quite a bit of code...

```
function loadData() {
    let http;

    if (window.XMLHttpRequest) {
        http = new XMLHttpRequest();
    } else {
        http = new ActiveXObject("Microsoft.XMLHTTP");
    }

    http.onreadystatechange = function() {
        if (http.readyState === 4 && http.status === 200) {
            document.getElementById("output").innerHTML
                = http.responseText;
        }
    }

    http.open("GET", "./data.txt", true);
    http.send();
}
```

XmlHttpRequests

- We will not be using this technology further on.

```
function loadData() {  
    let http;  
  
    if (window.XMLHttpRequest) {  
        http = new XMLHttpRequest();  
    } else {  
        http = new ActiveXObject("Microsoft.XMLHTTP");  
    }  
  
    http.onreadystatechange = function() {  
        if (http.readyState == 4 && http.status == 200) {  
            document.getElementById("output").innerHTML  
                = http.responseText;  
        }  
    }  
  
    http.open("GET", "./data.txt", true);  
    http.send();  
}
```

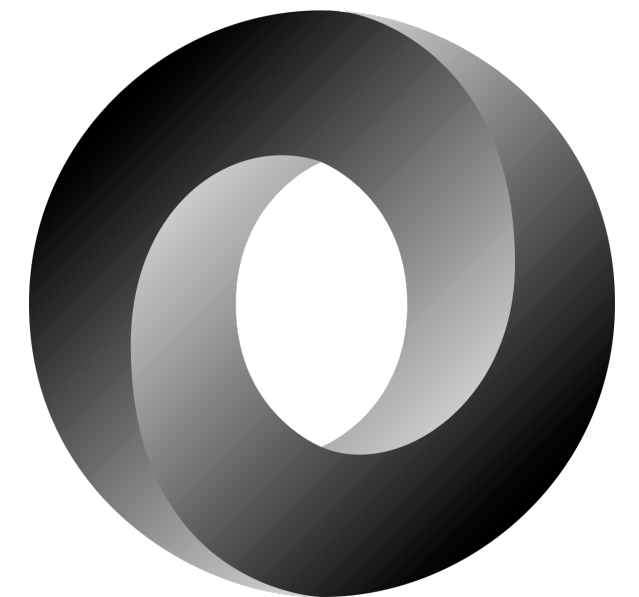
JSON-P

- Second mechanism
- Abbreviation stands for: JSON with Padding
- But, what is JSON again?

JSON

- JavaScript Object Notation
- Standard to represent JavaScript objects as strings
- Can be easily transferred between systems, over the network, ...

```
{ "empinfo" :  
  {  
    "employees" : [  
      {  
        "name" : "James Kirk",  
        "age" : 40,  
      },  
      {  
        "name" : "Jean-Luc Picard",  
        "age" : 45,  
      },  
      {  
        "name" : "Wesley Crusher",  
        "age" : 27,  
      }  
    ]  
  }  
}
```



JavaScript: convert from object to JSON

```
let lecturer = {
  lastName: "Vlummens",
  firstName: "Frédéric",
  age: 39
};

console.log(lecturer);
console.log("The type is:", typeof lecturer);

console.log("-----");

let lecturerAsJson = JSON.stringify(lecturer);

console.log(lecturerAsJson);
console.log("The type is:", typeof lecturerAsJson);
```

```
▶ {lastName: "Vlummens", firstName: "Frédéric", age: 39}
The type is: object
-----
{"lastName":"Vlummens","firstName":"Frédéric","age":39}
The type is: string
```

DEMO

JavaScript: convert from JSON to object

```
let json = '{"lastName":"Vlummens","firstName":"Frédéric","age":39}';  
console.log(json);  
console.log("The type is:", typeof json);  
  
console.log("-----");  
  
let lecturer = JSON.parse(json);  
console.log(lecturer);  
console.log("The type is:", typeof lecturer);
```

```
{"lastName":"Vlummens","firstName":"Frédéric","age":39}  
The type is: string  
-----  
▶ {lastName: "Vlummens", firstName: "Frédéric", age: 39}  
The type is: object
```

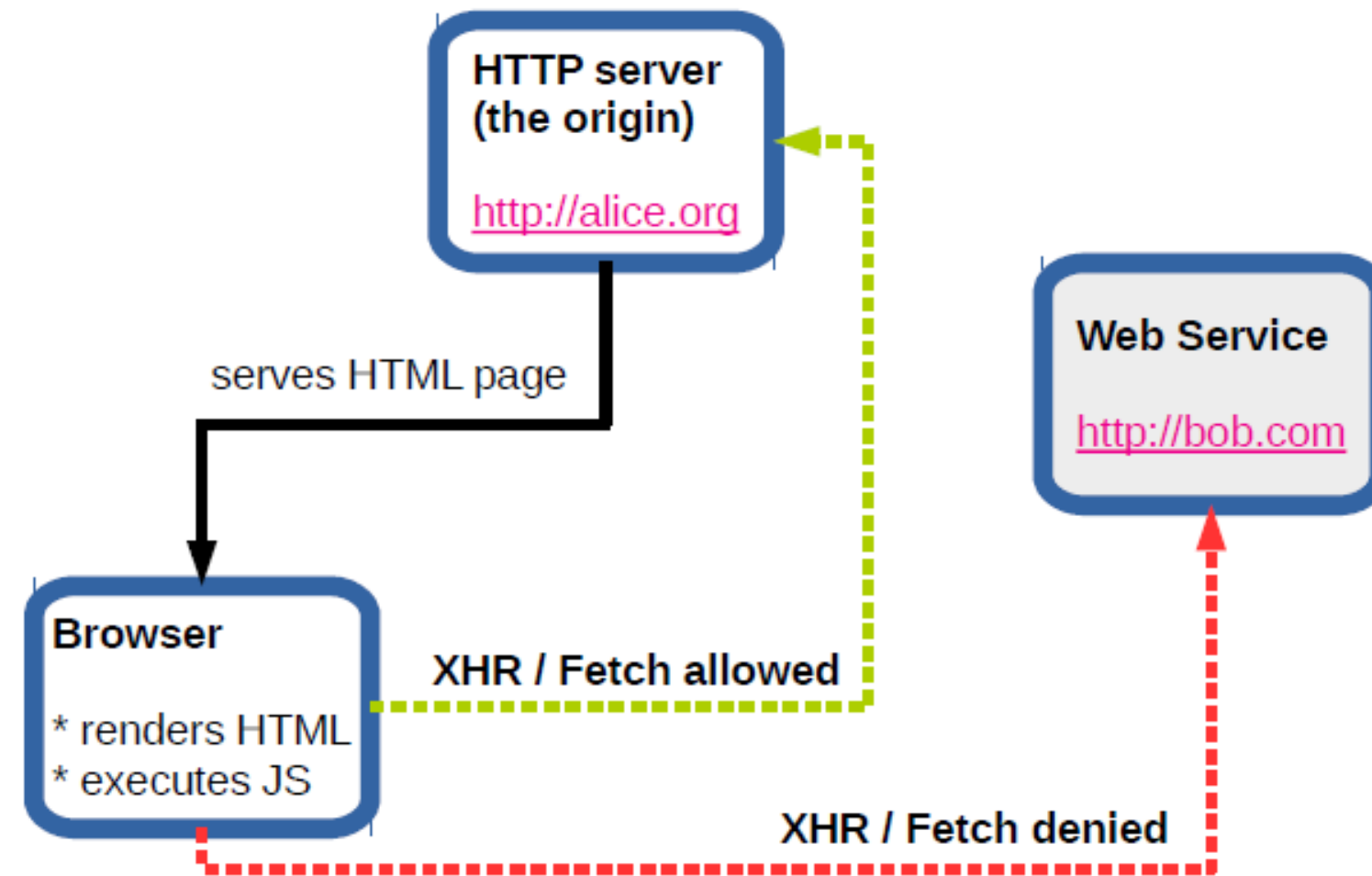
DEMO

JSON-P

- Second mechanism we will look at to transmit data asynchronously
- JSON with Padding
- JavaScript is used to load data by executing a script tag
- Allows to bypass something called the Same Origin Policy (SOP)
- Before the adoption of CORS (Cross-Origin Resource Shares), this was one of the only ways to get around SOP
- A word about SOP and CORS...

The Same Origin Policy

- Security mechanism
- Restricts how document/script loaded from origin A can access/interact with resources from origin B



Origin

- Consists of protocol, port and host.
- Example:
Comparison of different URLs to <http://app.howest.be/acs/index.html>

| URL | Status | Reason |
|-------------------------------------------------------------------------------------------------|--------|--------------------------|
| http://app.howest.be/ti/index.html | | Only different path |
| http://app.howest.be/acs/sem4/index.html | | Only different path |
| https://app.howest.be/ti/index.html | | Different protocol |
| http://app.howest.be:81/acs/index.html | | Different port (80 ⇔ 81) |
| http://www.howest.be/ti/index.html | | Different host |

CORS = Cross Origin Resource Sharing

- HTTP header based
- Tells browser to give web app at origin A access to resources at origin B
- Cross-origin HTTP request: when requesting a resource with a different origin

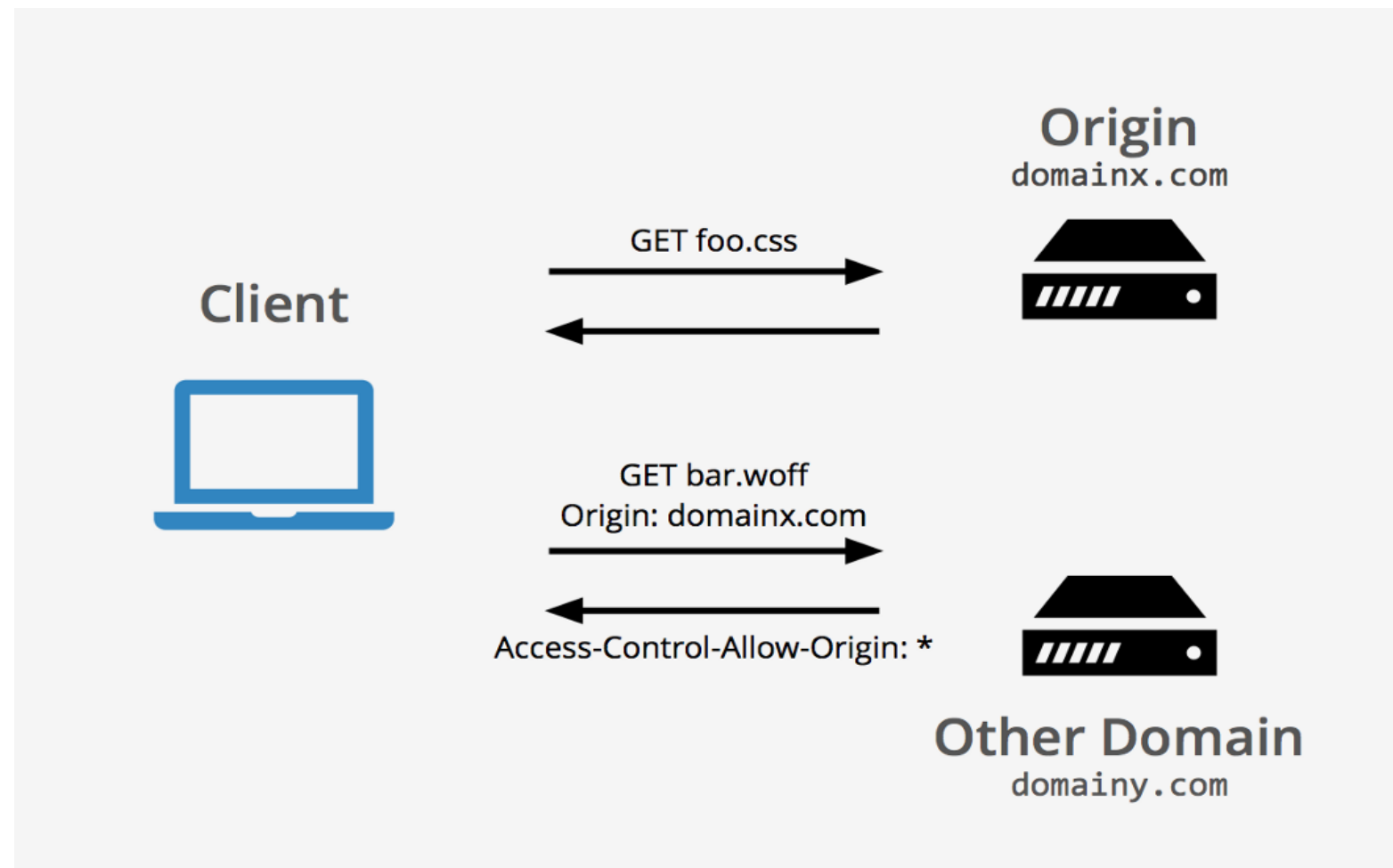


Image source: <https://medium.com/@buddhiv/what-is-cors-or-cross-origin-resource-sharing-eccbfcaaaa30>

CORS: three categories

- Cross-origin **writes**
 - Typically allowed
 - Examples: links, redirects and form submissions
- Cross-origin **reads**
 - Typically not allowed
 - Often worked around using embedding
- Cross-origin **embedding**
 - Typically allowed

```
2 Failed to load resource: the server responded with a status of 403 (Forbidden) www.fredericvlummens.be/:1
✖ Access to fetch at 'https://www.fredericvlummens.be/' from origin 'null' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled. index.html:1
✖ Uncaught (in promise) TypeError: Failed to fetch index.html:1
>
```

CORS: embedding content

- JavaScript: `<script>`
 - Error messages only available for same-origin scripts
- CSS: `<link>`
 - Cross-origin CSS requests require correct Content-Type header
- Images: ``
- Media files: `<video>` and `<audio>`
- Plug-ins: `<object>`, `<embed>` and `<applet>`
- Fonts: `@font-face` (browser-dependant)
- Frames: `<frame>` and `<iframe>`
 - Anything can be embedded
 - Use X-Frame-Options header in order to prevent

JSON-P

- Second mechanism we will look at in order to transmit data asynchronously
- JSON with Padding
- Works by “abusing” the embedded JavaScript rule
- How?
 - Create a <script> tag using JavaScript
 - Set the src to the URL you want to retrieve data from
 - Include callback to process JSON that is retrieved
 - Inject in page so it will load
 - Process the JSON data in the callback process

DEMO

JSON-P

```
let BASE_URL = "https://api.flickr.com/services/feeds/photos_public.gne?" +  
    "jsoncallback=processJSON&format=json&tags=";  
  
document.addEventListener("DOMContentLoaded", init);  
  
function init() {  
    document.querySelector("form").addEventListener("submit", processForm);  
}  
  
function processForm(e) {  
    e.preventDefault();  
    let search = document.getElementById("search").value;  
    performFlickrSearch(search);  
}
```

JSON-P

```
function performFlickrSearch(tag) {
    let url = BASE_URL + tag;
    let script = document.createElement("script");
    script.src = url;

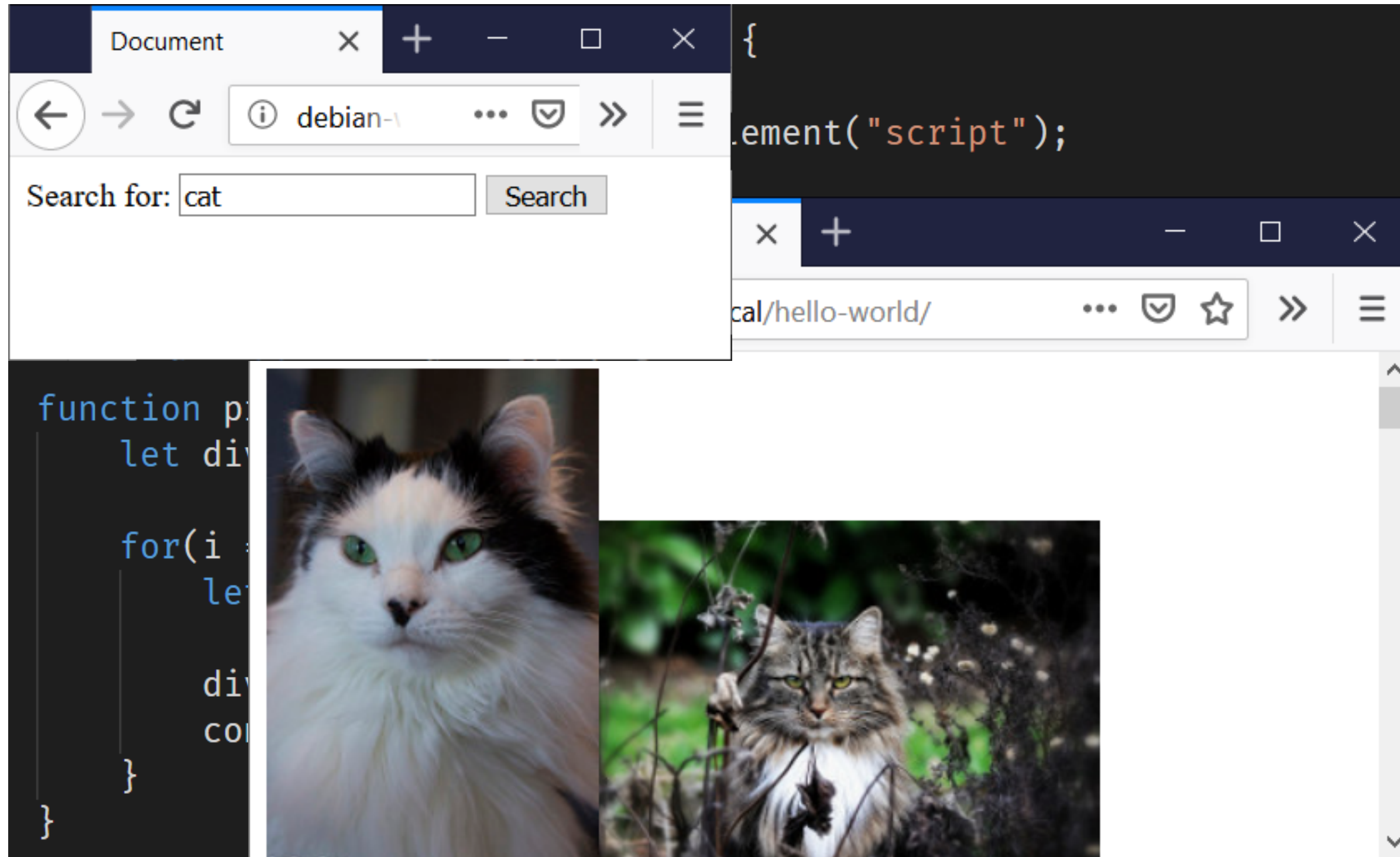
    document.getElementsByTagName("body")[0].appendChild(script);
}

function processJSON(json) {
    let div = document.querySelector("div");

    for(i = 0; i < json.items.length; i++) {
        let item = json.items[i].media.m;

        div.innerHTML += ``;
        console.log(item);
    }
}
```

JSON-P



Fetch API

- Easier to use than XHR
- Less housekeeping code to write
- Promise-based:
 - “Promises represent eventual completion (or failure) of an asynchronous operation and its resulting value”
 - Cleaner and easier code
 - Most modern APIs are promise-based


Fetch: a first example

```
function loadPokemon() {  
  ⚡ fetch( "https://pokeapi.co/api/v2/pokemon" ).then(function(response) {  
    console.log(response);  
  }).catch(function(err) {  
    console.error("Something went wrong:", err);  
  });  
}
```

`function(response) { ... }` is executed upon successful completion of the fetch.

DEMO

Fetch: a first example

```
function loadPokemon() {  
  fetch( "https://pokeapi.co/api/v2/pokemon" ).then(function(response) {  
    console.log(response);  
     }).catch(function(err) {  
    console.error("Something went wrong:", err);  
  });  
}
```

`function(err) { ... }` is executed when something goes wrong.

Fetch: a first example

```
function loadPokemon() {  
  fetch("https://pokeapi.co/api/v2/pokemon").then(function(response) {  
    console.log(response);  
  }).catch(function(err) {  
    console.error("Something went wrong:", err);  
  });  
}
```

Successful completion,
but we get the response object, not JSON data→

```
▼ Response  
  ▶ body: ReadableStream { locked: false }  
    bodyUsed: false  
  ▶ headers: Headers { }  
    ok: true  
    redirected: false  
    status: 200  
    statusText: "OK"  
    type: "cors"  
    url: "https://pokeapi.co/api/v2/"  
  ▶ <prototype>: ResponsePrototype { clone:  
    clone(), arrayBuffer: arrayBuffer(), blob:  
    blob(), ... }
```

Fetch: a first example

```
function loadPokemon() {  
  fetch( "https://pokeapi.co/api/v2/pokemon" ).then(function(response) {  
    return response.json();  
  }).then(function(json) {  
    console.log(json);  
  }).catch(function(err) {  
    console.error("Something went wrong:  
  });  
}
```

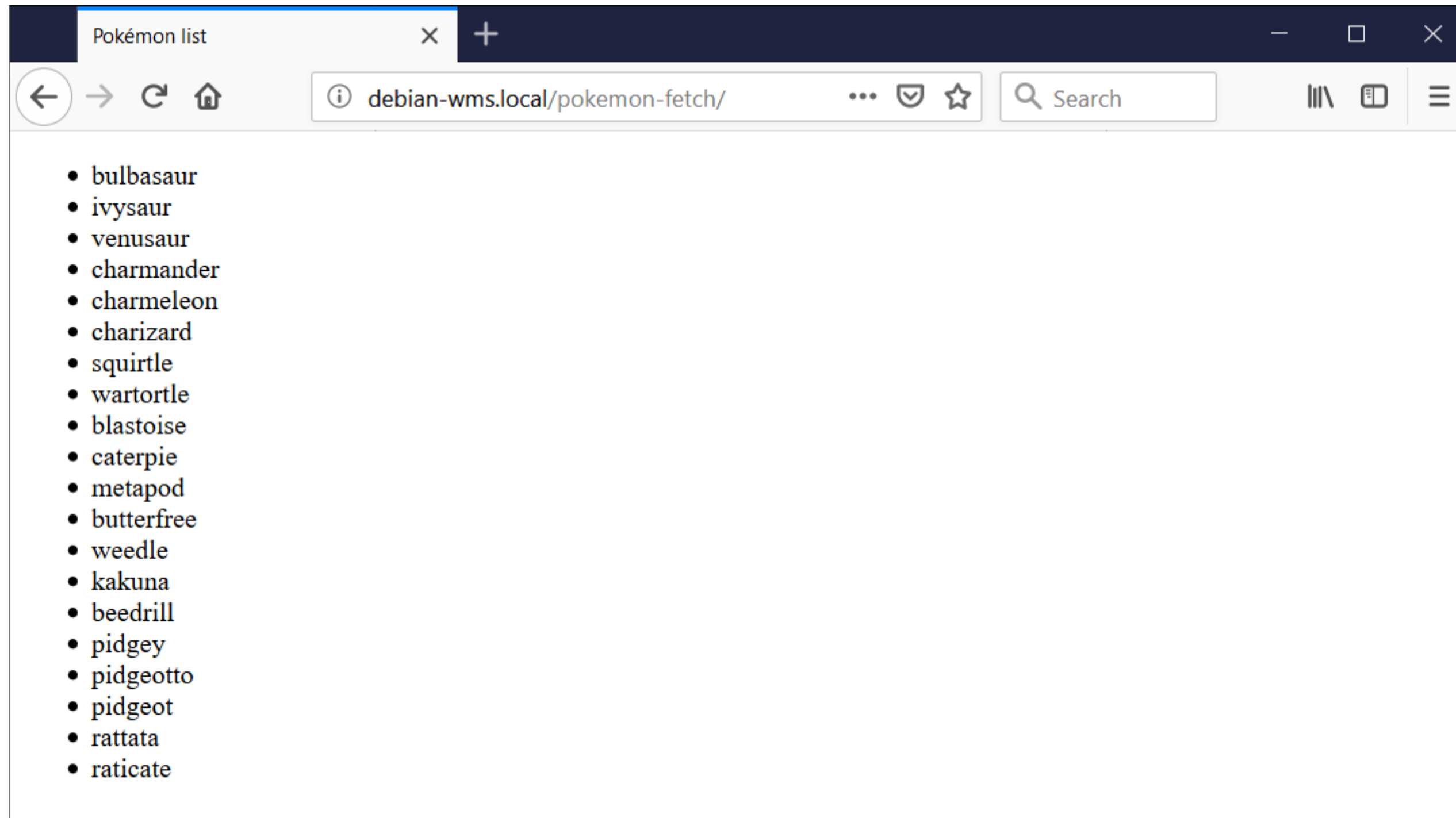
We go one step “deeper”
and we find our data →

```
{...} script  
count: 964  
next: "https://pokeapi.co/api/v2/  
/pokemon?offset=20&limit=20"  
previous: null  
results: (20) [...]  
  ▶ 0: Object { name: "bulbasaur", url:  
    "https://pokeapi.co/api/v2/pokemon/1/" }  
  ▶ 1: Object { name: "ivysaur", url:  
    "https://pokeapi.co/api/v2/pokemon/2/" }  
  ▶ 2: Object { name: "venusaur", url:  
    "https://pokeapi.co/api/v2/pokemon/3/" }  
  ▶ 3: Object { name: "charmander", url:  
    "https://pokeapi.co/api/v2/pokemon/4/" }  
  ▶ 4: Object { name: "charmeleon", url:  
    "https://pokeapi.co/api/v2/pokemon/5/" }
```

Fetch: completing our first example

```
function loadPokemon() {  
  fetch("https://pokeapi.co/api/v2/pokemon").then(function(response) {  
    return response.json();  
  }).then(function(json) {  
    let pokemonList = json.results;  
  
    let output = document.querySelector("ul");  
  
    pokemonList.forEach(function(pokemon) {  
      output.innerHTML += `<li>${pokemon.name}</li>`;  
    });  
  }).catch(function(err) {  
    console.error("Something went wrong:", err);  
  });  
}
```

Fetch: completing our first example



Fetch: more uses

Retrieving response body as text:

```
fetch("https://example.com/hello-world.txt").then(function(response) {  
    return response.text();  
}).then(function(text) {  
    console.log(text);  
}).catch(function(err) {  
    console.error(err);  
});
```

Sending options and (in this case) performing a POST instead of a standard GET, including the data of the form with ID “the-form”:

```
fetch("https://example.com/submit-entry.php", {  
    method: "post",  
    body: new FormData(document.getElementById("the-form"))  
});
```

Fetch: more uses

Sending options and (in this case) performing a POST instead of a standard GET, submitting JSON string representation of object with 2 properties (name and address):

```
fetch("https://example.com/submit-json.php", {  
  method: "post",  
  body: JSON.stringify({  
    name: document.getElementById("name").value,  
    address: document.getElementById("address").value  
  })  
});
```

Fetch: more information

- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

Questions?

