Introduction to SOP and CORS

WASA: Web and Software Architecture

Enrico Bassetti

SOP: Same-origin policy

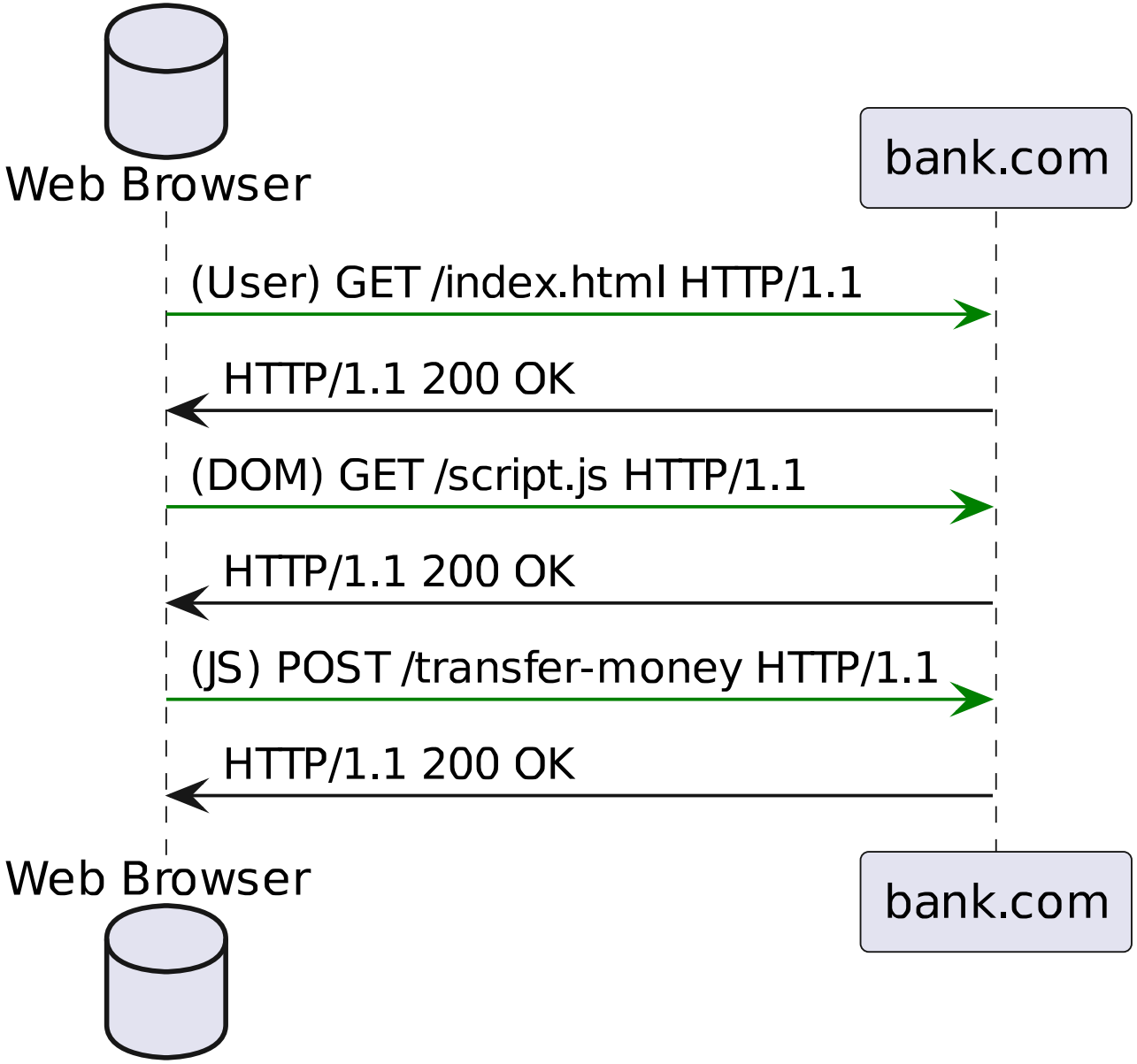
Same-origin policy

Same-origin policy is a security mechanism in **browser**. It restricts communication between scripts with different *origin*s.

https://bank.com/index.html

```
<!doctype html>
<html>
  <head><!-- ... --></head>
<body>
  <!-- ... -->
  <script src="/script.js"></script>
</body>
</html>
```

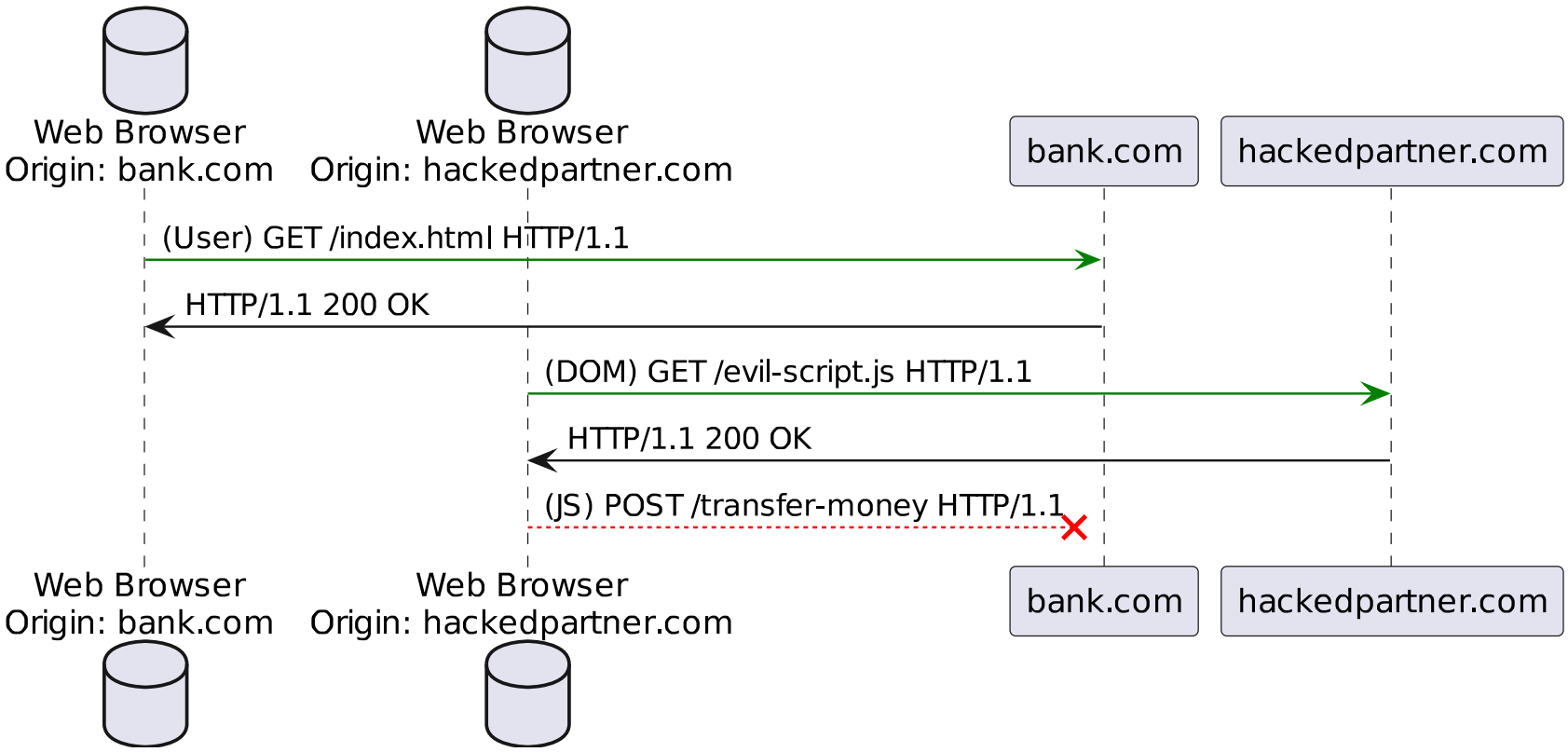
Script sequence diagram



https://bank.com/index.html

```
<!doctype html>
<html>
  <head><!-- ... --></head>
<body>
  <!-- ... -->
  <script
    src="https://hackedpartner.com/evil-script.js">
  </script>
</body>
</html>
```

Malicious script, different origin



What SOP protects?

- Cookies
- HTTP Authentication
- IndexedDB
- Web Storage
- DOM

SOP rules

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same scheme, host and port
http://www.example.com/dir2/other.html	Success	Same scheme, host and port
http://username:password@**www.example.com**/dir2/other.html	Success	Same scheme, host and port
http://www.example.com:**81**/dir/other.html	Failure	Same scheme and host but different port
https://www.example.com/dir/other.html	Failure	Different scheme
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

Table courtesy of Wikipedia - CC BY-SA 3.0

SOP interactions

Cross-origin requests can be:

- **Writes**: typically allowed but limited, CORS may be needed
- **Embedding**: typically allowed
- **Reads**: typically denied

SOP is not enough...

Unfortunately, SOP is not enough: CSRF (*Cross-site request forgery*) may be executed via cross-origin writes.

CSRF token + SOP can mitigate that (SOP will disallow reading the CSRF token).

In a Single-page application, the token is in the SPA website origin, and we use a header for that - no need for a CSRF token (see CORS).

CORS: Cross-Origin Resource Sharing

Cross-Origin Resource Sharing

Sometimes you need to share resources between *origins*.

CORS is an HTTP-header-based mechanism for configuring *Same-origin policy*.

Type of requests

CORS defines two different types of requests:

- **Simple requests**
  - Always sent to the server
  - The server decides which origin can read the response
- **Preflighted requests**
  - Before sending the request, the browser sends a preflight request
  - The server decides which origin can send requests and what can be sent
  - The server decides which origin can read the response

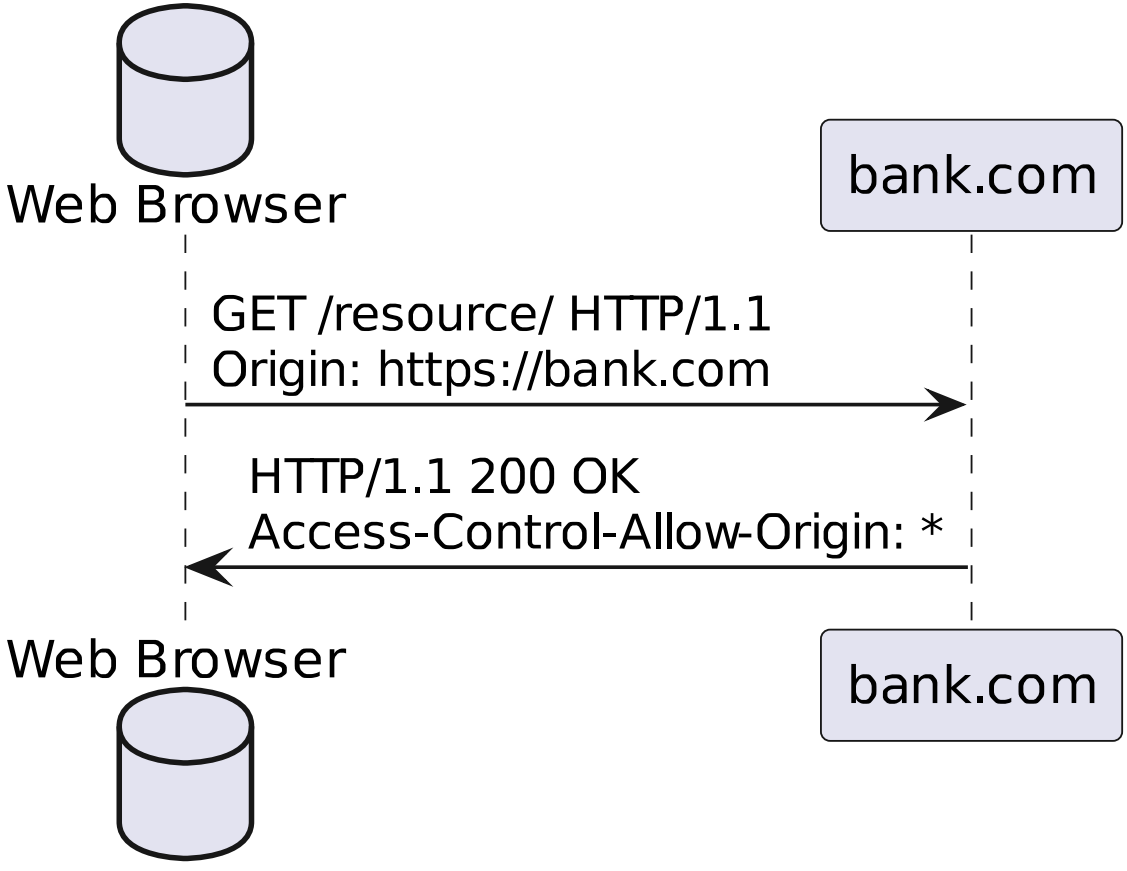
Simple requests

A **simple request** must meet all the following conditions:

- One of: *`GET`*, *`HEAD`* or *`POST`*
- Apart from automatic headers, allowed headers are: *`Accept`*, *`Accept-Language`*, *`Content-Language`*, *`Content-Type`*, *`Range`*
- *`Content-Type`*, if set, is one of: *`application/x-www-form-urlencoded`*, *`multipart/form-data`*, or *`text/plain`*
- If *`XMLHttpRequest`* is used, no *`XMLHttpRequest.upload`* listeners
- No *`ReadableStream`* object used

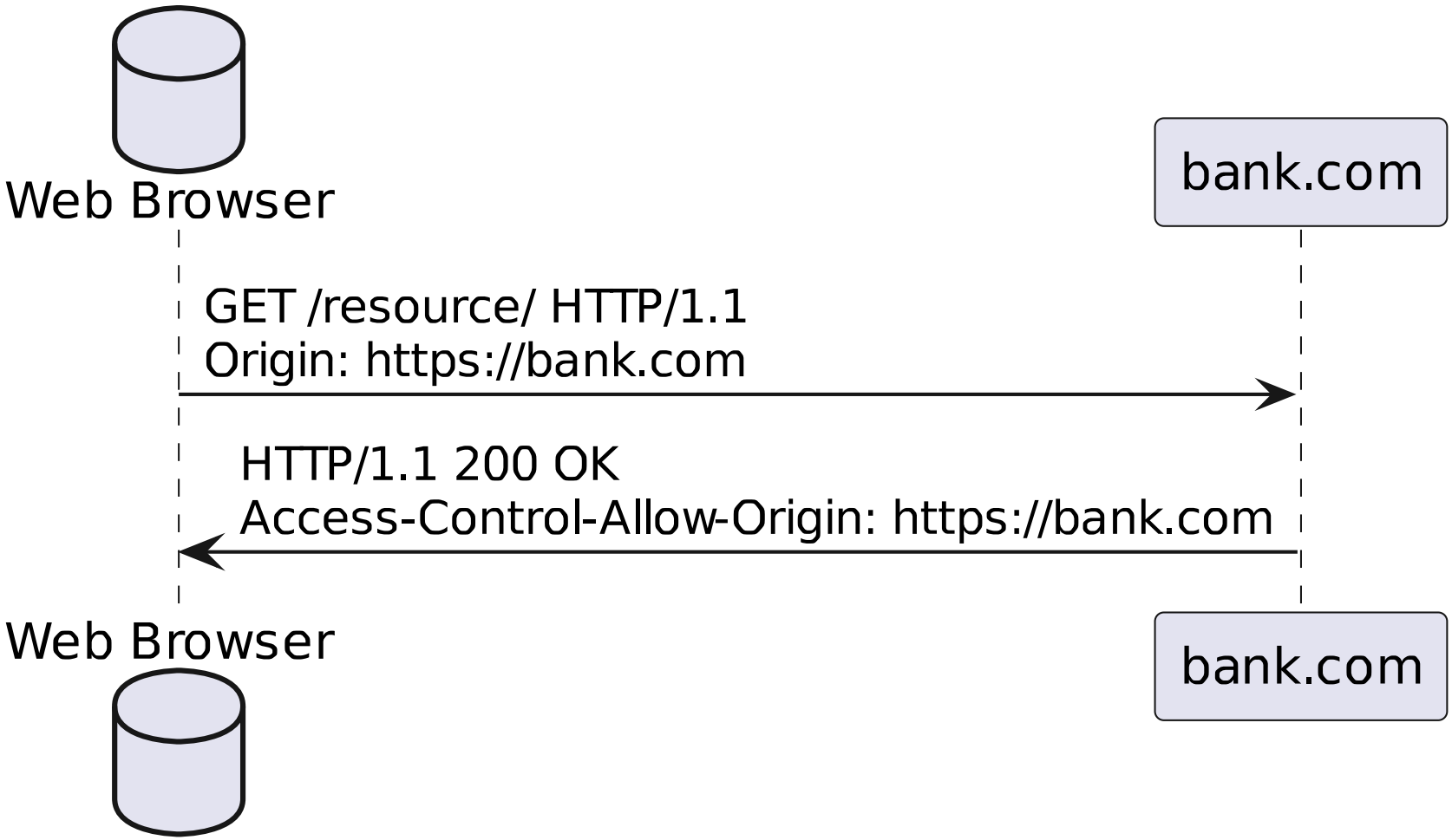
Browsers may have additional restrictions.

Simple requests



- The request always contains the *`Origin`* header
- The server can control which origins can access the response via *`Access-Control-Allow-Origin`*

Simple requests



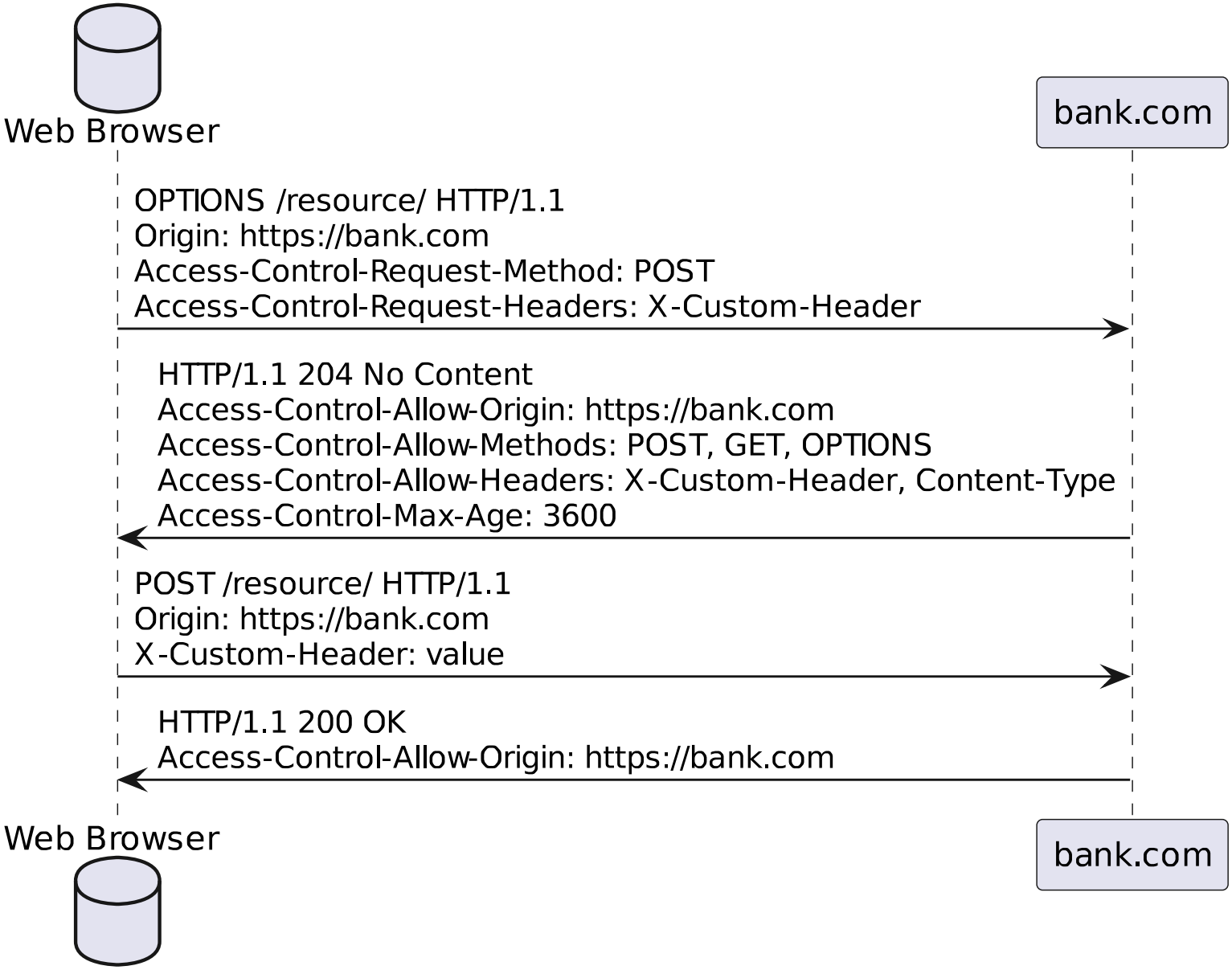
- Here, only *`https://bank.com`* can access the response

Preflighted requests

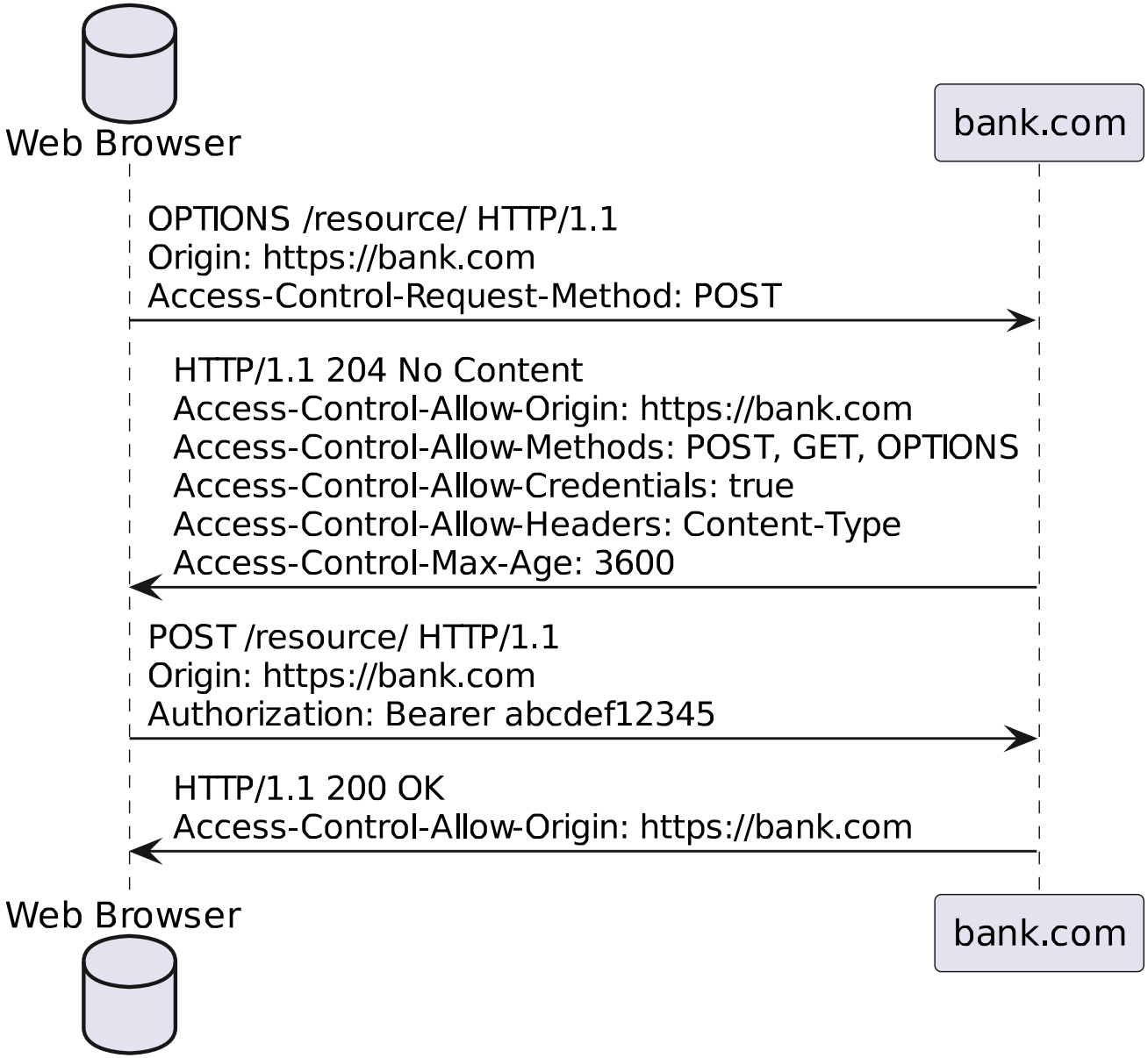
A **preflighted request** is a request that does not match the conditions for **simple requests**.

It requires an HTTP *`OPTIONS`* request before the actual request (handled automatically by browsers) to agree on the SOP with the server.

Preflighted requests



Preflighted requests with authentication



No need for CSRF token

We use the *`Authorization`* header for authentication/authorization in the WASA project.

Given that *`Authorization`* will trigger a preflight request, there is no need for a CSRF token — just configure CORS correctly.

Links

- https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- https://www.youtube.com/watch?v=KaEj_qZgiKY
- https://en.wikipedia.org/wiki/Same-origin_policy
- https://en.wikipedia.org/wiki/Cross-origin_resource_sharing