UNIVERSITY OF
Southampton

UNIVERSITY OF Southampton

Cross-Origin Resource Sharing

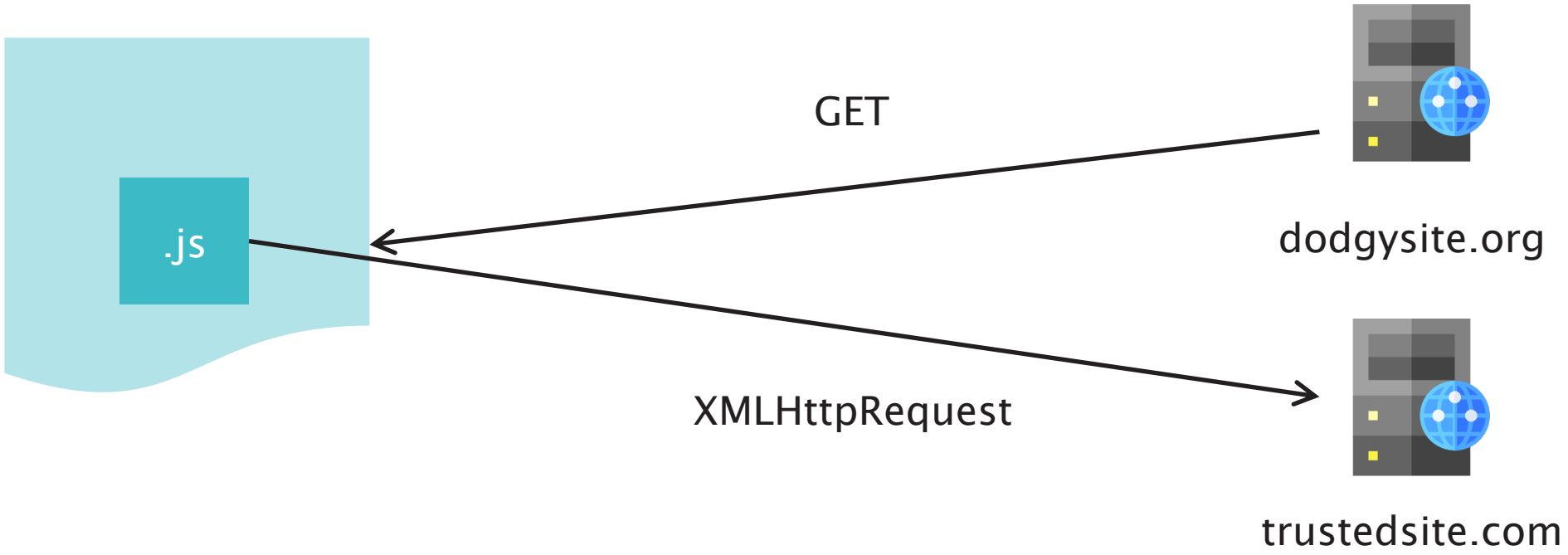
COMP3220 Web Infrastructure

Dr Nicholas Gibbins – nmg@ecs.soton.ac.uk

Cross-Site Request Forgery

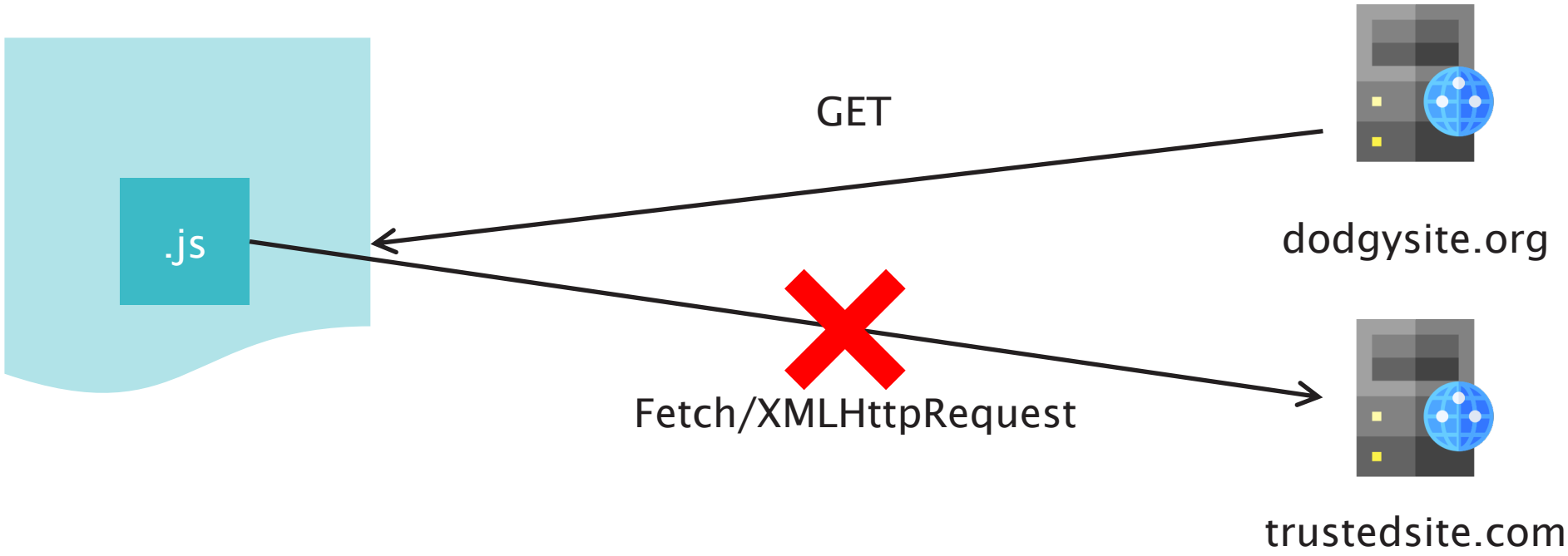
If user agents allow one origin to talk to a different origin, there may be security issues

- User agent will send cookies if available and applicable
- Privilege escalation attack – confused deputy



Same-Origin Policy

By default, browsers restrict how a resource from one origin interacts with resources from other origins



Barth, A. (2011) *The Web Origin Concept*. RFC6454. Available online at: https://tools.ietf.org/html/rfc6454

Same-Origin Policy

Two resources have the same origin if:

- Their URIs use the same protocol (i.e. no mixing of http and https)
- Their URIs have the same host
- Their URIs have the same port

(path is ignored for the purposes of determining origin)

Same-Origin Policy

All the same origin

- http://example.org/
- http://example.org:80/
- http://example.org/foo

All different origins

- http://example.org/
- https://example.org/
- http://example.org:8080/
- http://www.example.org/
- http://example.com/
- https://example.org:80/

Same-Origin Policy

By default, the SOP blocks cross-origin reads by the browser

Exception: embedded resources:

- Media (`img`/`audio`/`video`)
- external stylesheets (`<link rel="stylesheet" href="..."/>`)
- scripts (`<script src="..."></script>`)
- `@font-face` (some variability between browsers)
- `iframe`

Cross-origin POSTS that result from form submission are allowed

Cross-Origin Resource Sharing

Mechanism for selectively relaxing the Same-Origin Policy

At a protocol level:

- Adds new headers that let servers indicate which origins may make requests
- Restricts the headers which may be sent in requests (i.e. avoiding taint)
- Restricts the headers which may be received in responses

At an API level:

- CORS requests sent via `fetch()` API
- Client-side enforcement
- Disallowed request may still result in a message being sent by the browser
- Result of a disallowed request is not sent to the script by the browser
- Reason for CORS error written to browser console, but not available to script

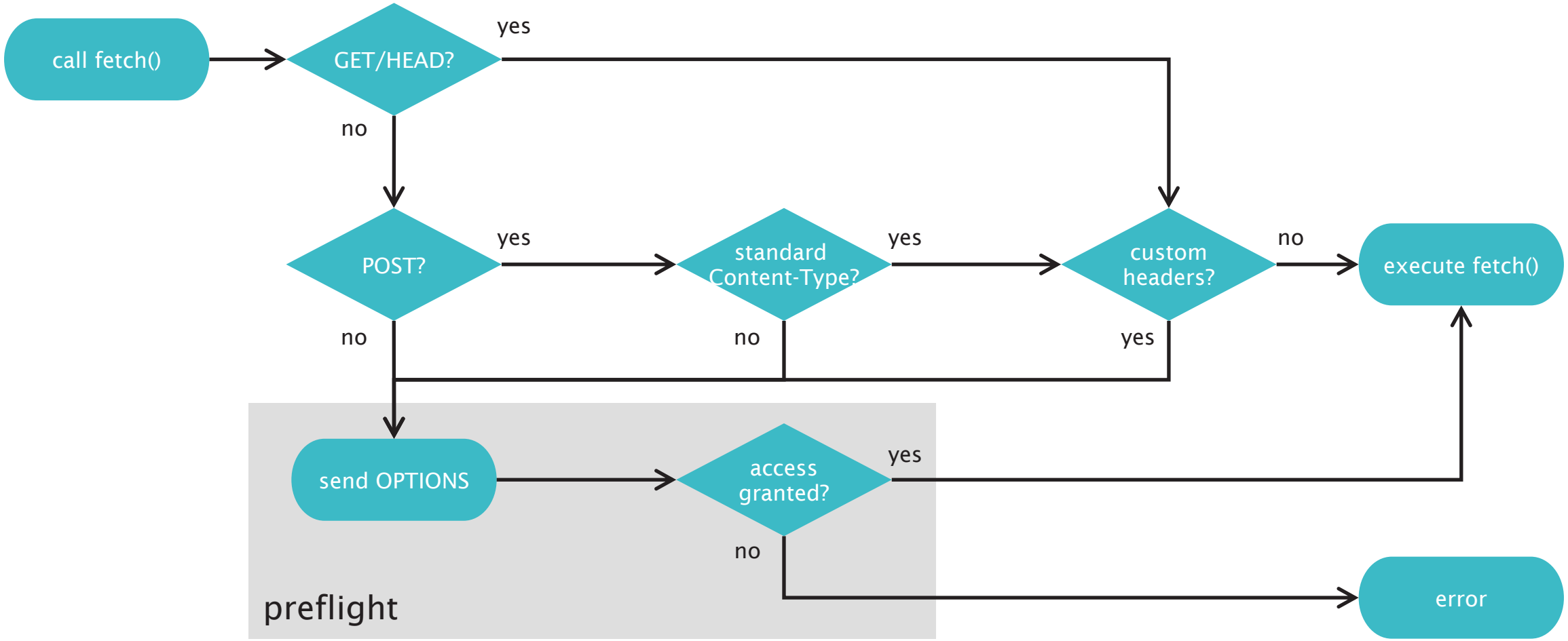
CORS requests

Simple requests satisfy **all** of the following

- `GET`, `HEAD` or `POST` only
- `Accept:`, `Accept-Language:`, `Content-Type:` or `Content-Language:` are the only headers set manually
- `Content-Type:` is one of `text/plain`, `application/x-www-form-urlencode` or `multipart/formdata` only

All other requests trigger a CORS preflight

CORS flow


call fetch()
GET/HEAD?
yes
no
POST?
yes
standard Content-Type?
yes
custom headers?
no
execute fetch()
no
no
yes
send OPTIONS
access granted?
yes
no
error
preflight

CORS headers

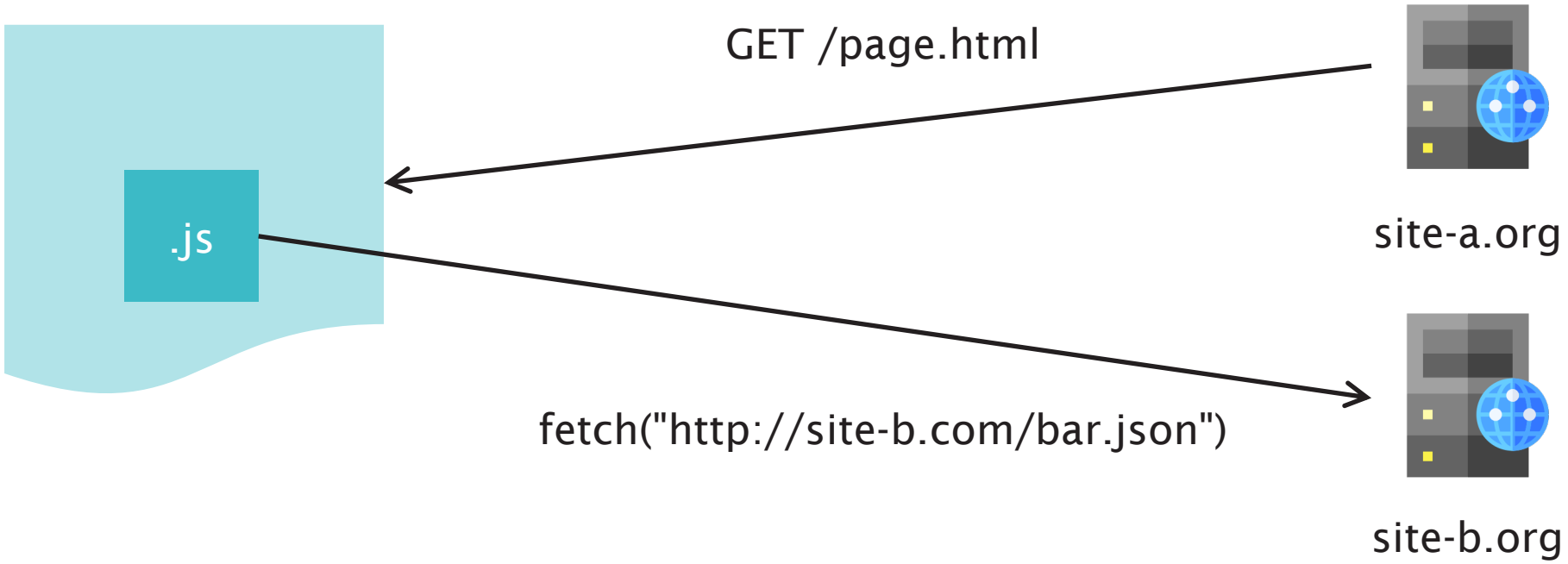
Client header

- `Origin:` – like `Referer:`, but excludes path (automatically added by browser)
- `Access-Control-Request-Method:` – used in preflight (see later)
- `Access-Control-Request-Headers:` – used in preflight (see later)

Server headers

- `Access-Control-Allow-Origin:` – which origins are accepted? (* for any)
- `Access-Control-Allow-Methods:` – which methods are accepted?
- `Access-Control-Allow-Headers:` – which headers are accepted?
- `Access-Control-Max-Age:` – for how long is a preflight check valid?
- `Access-Control-Allow-Credentials:` – include cookies (etc) in request

Simple request example



Simple request example

```
// in a script in https://site-a.org/page.html

fetch("https://site-b.org/bar.json",
      {mode: "cors",
       method: "GET"})
  .then(response => {
    if (!response.ok) {
      throw new Error("HTTP status " + response.status);
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => {
    console.error("Error: " + error);
  });
```

Simple request example

Simple request example

Simple request example

Preflight requests

Unsafe requests (POST, PUT, DELETE) require a preflight check

Client sends an OPTIONS message to determine if the intended request may be sent

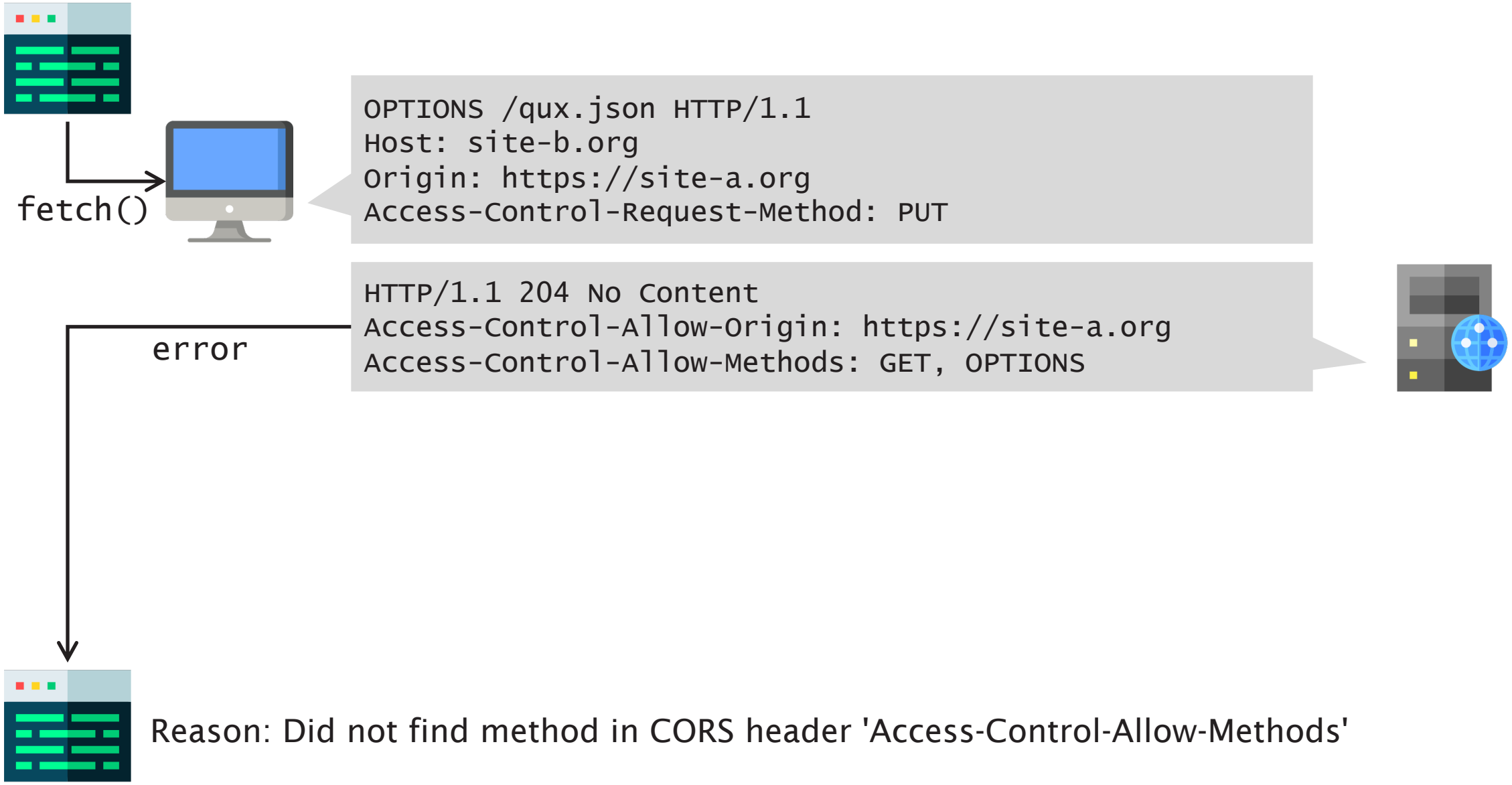
- `Access-Control-Request-*`: headers used to express intended request
- Server responds with Access-Control-Allow-*: headers to express permissions

Preflight request example

```
// in a script in https://site-a.org/page.html

fetch("https://site-b.org/qux.json",
      {mode: "cors",
       method: "PUT",
       body: <...>})
  .then(response => {
    if (!response.ok) {
      throw new Error("HTTP status " + response.status);
    }
    return response.json();
  })
  .then(data => console.log(data))
  .catch(error => {
    console.error("Error: " + error);
  });
```


fetch()
OPTIONS /qux.json HTTP/1.1
Host: site-b.org
Origin: https://site-a.org
Access-Control-Request-Method: PUT
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: https://site-a.org
Access-Control-Allow-Methods: GET, PUT, OPTIONS
PUT /qux.json HTTP/1.1
Host: site-b.org
Origin: https://site-a.org
response
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://site-a.org
Access-Control-Allow-Methods: GET, PUT, OPTIONS



Reason: Did not find method in CORS header 'Access-Control-Allow-Methods'

Further reading

Barth, A. (2011) *The Web Origin Concept*. RFC6454

https://tools.ietf.org/html/rfc6454

CORS for developers

https://w3c.github.io/webappsec-cors-for-developers/

HTML5 Fetch API

https://fetch.spec.whatwg.org/

Cross-Origin Resource Sharing at MDN

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

UNIVERSITY OF Southampton

Next Lecture: Content Security Policy