The 2-server problem

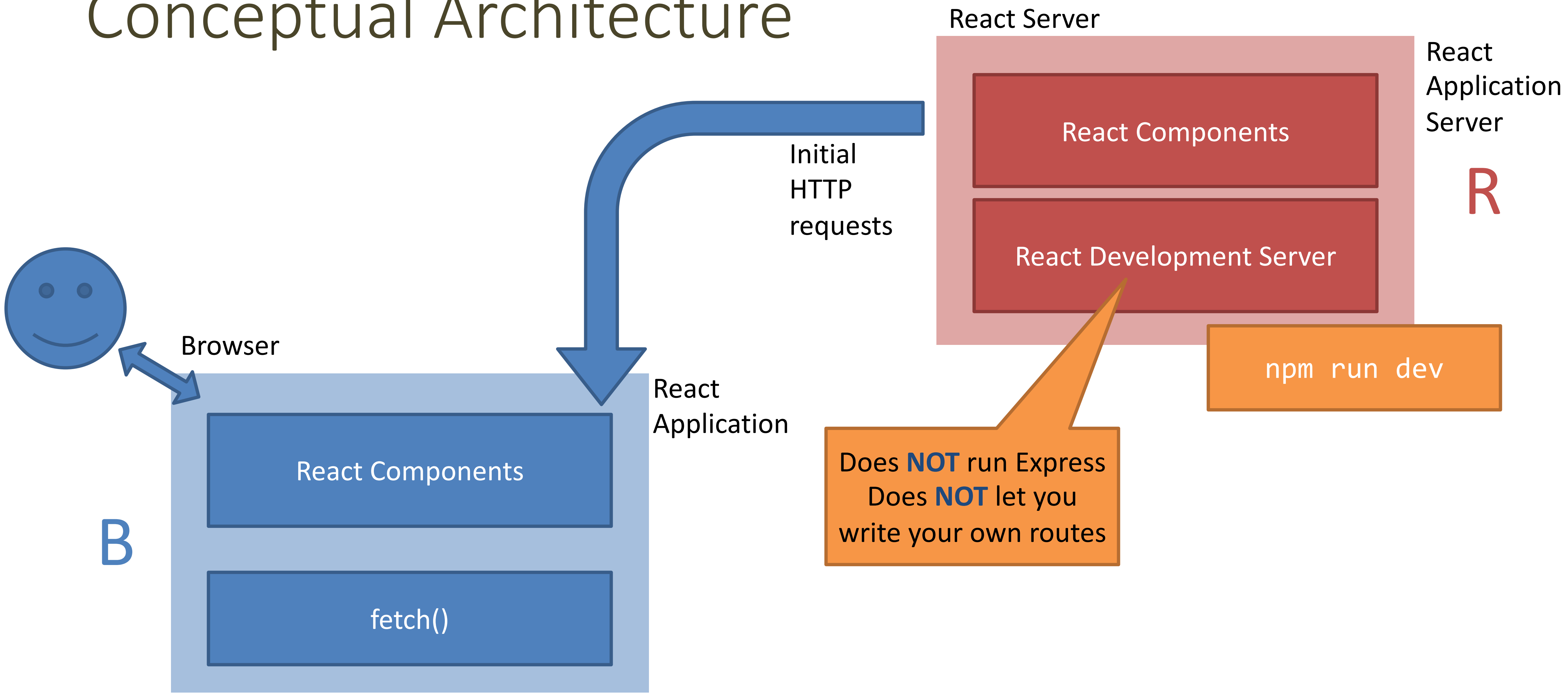
CORS and CSP

Enrico Masala
Fulvio Corno
Luigi De Russis

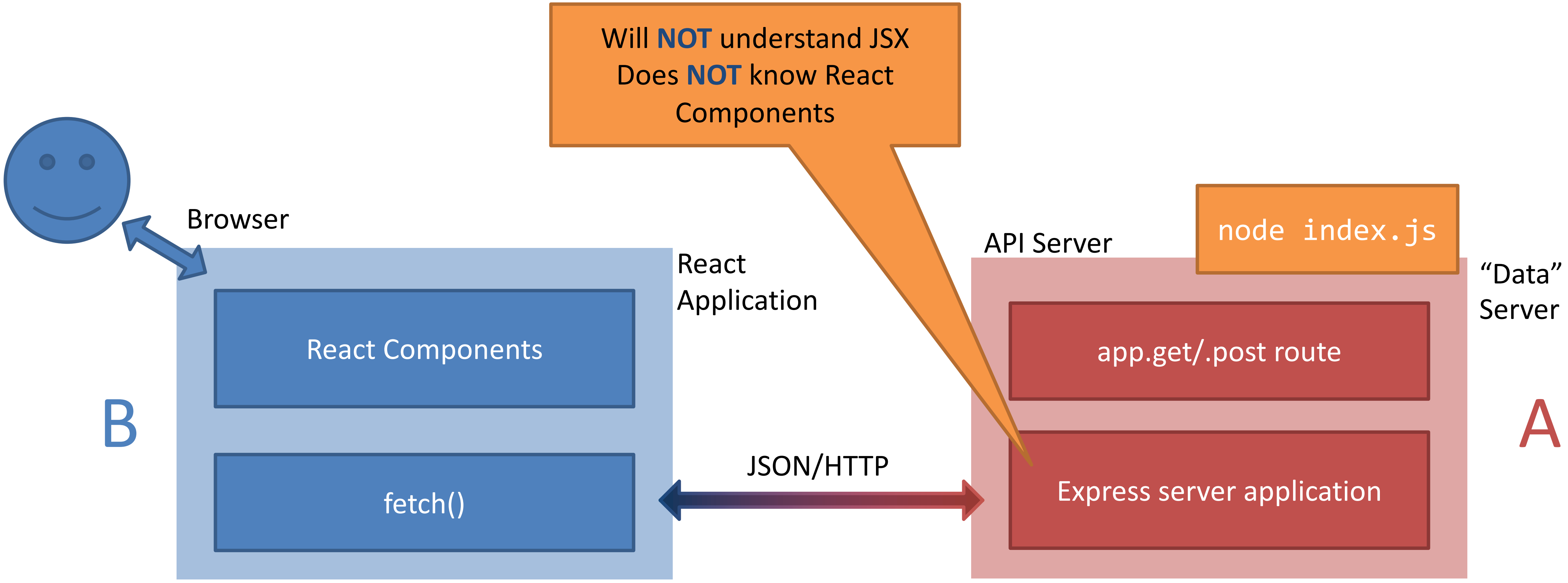
Goal

- What is an origin
- Cross-origin requests
- Why using CORS
- How CORS works
- How to enable CORS in Express

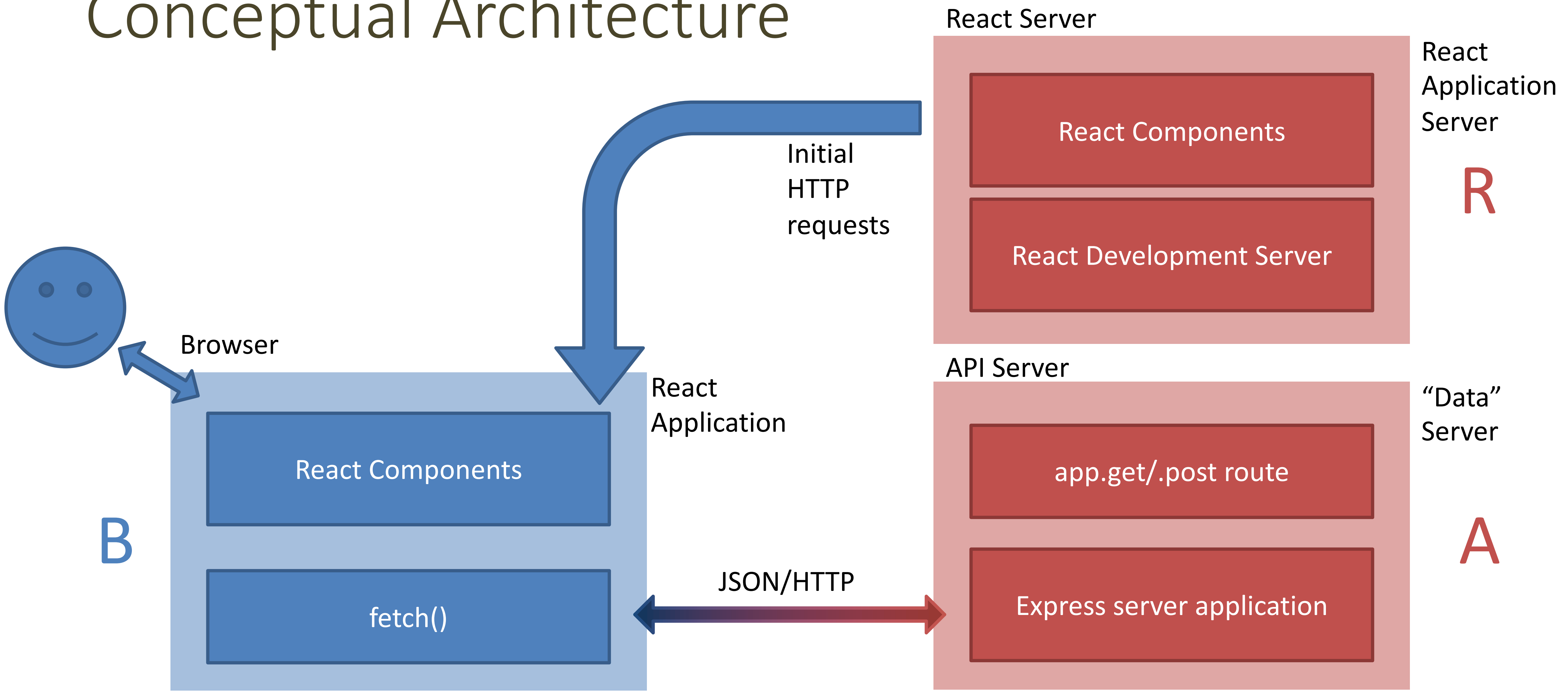
Conceptual Architecture



Conceptual Architecture



Conceptual Architecture



Issues

- Opportunities
  - Separate the load into different servers
  - Use any API Server (even 3rd party ones, even more than one)
- Deployment
  - Cross-Origin security limitations

Mozilla Developer Network:
Web technology for developers —
HTTP — Cross-Origin Resource Sharing (CORS)
https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Accessing multiple websites

CROSS-ORIGIN REQUEST SHARING

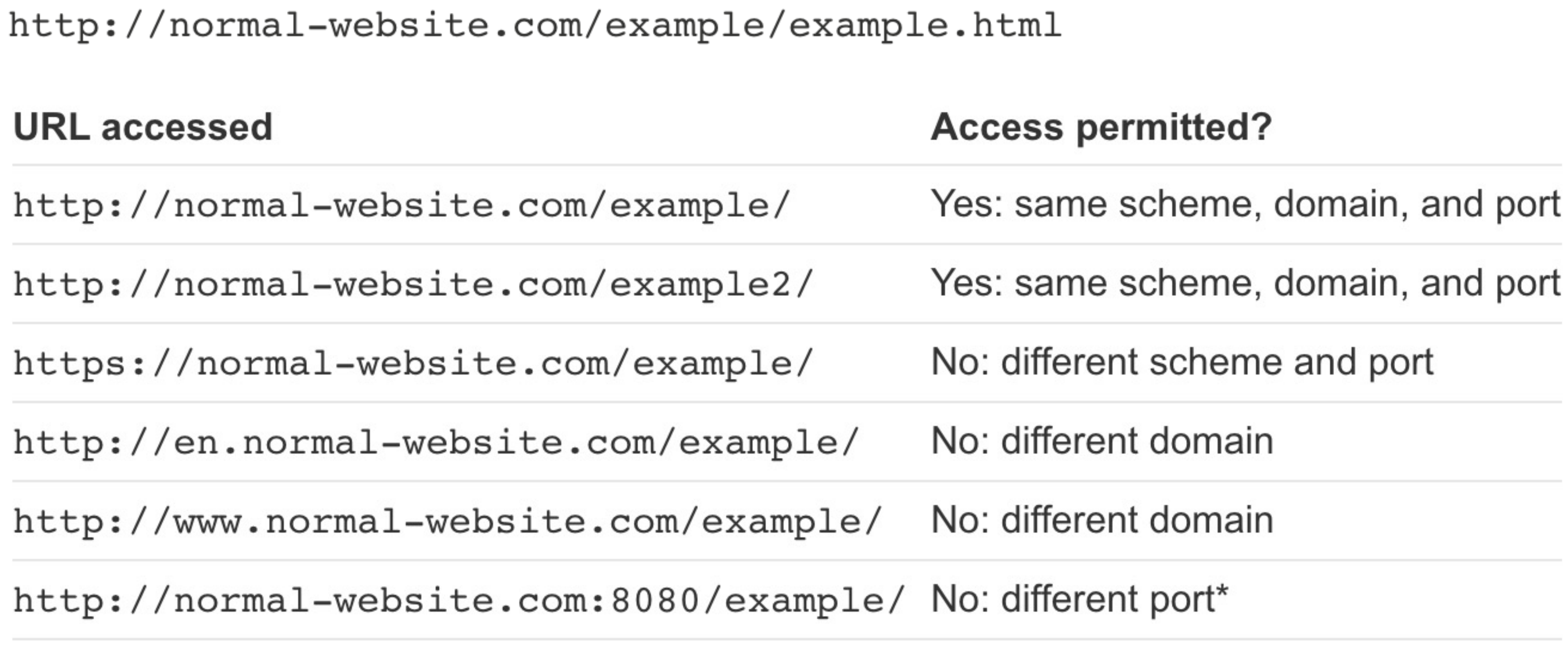
Loading a Web Page

- Loading a web page requires to load external resources (images, CSS, JS)
- They (JS, CSS) can, in turn, load other resources and generate network requests (asynchronous JS requests – e.g., via `fetch()`)
- For security reasons, network access is, by default, limited to the **same origin** (e.g., when loading other JS scripts)
- An **origin** consists of a **URI scheme**, **domain** and **port number**:

  http://**example.com**:**3456**/example/

Same-Origin Policy (SOP)

- Access only **same URI scheme, domain** and **port** number of the initial page



`http://normal-website.com/example/example.html`

URL accessed	Access permitted?
`http://normal-website.com/example/`	Yes: same scheme, domain, and port
`http://normal-website.com/example2/`	Yes: same scheme, domain, and port
`https://normal-website.com/example/`	No: different scheme and port
`http://en.normal-website.com/example/`	No: different domain
`http://www.normal-website.com/example/`	No: different domain
`http://normal-website.com:8080/example/`	No: different port*

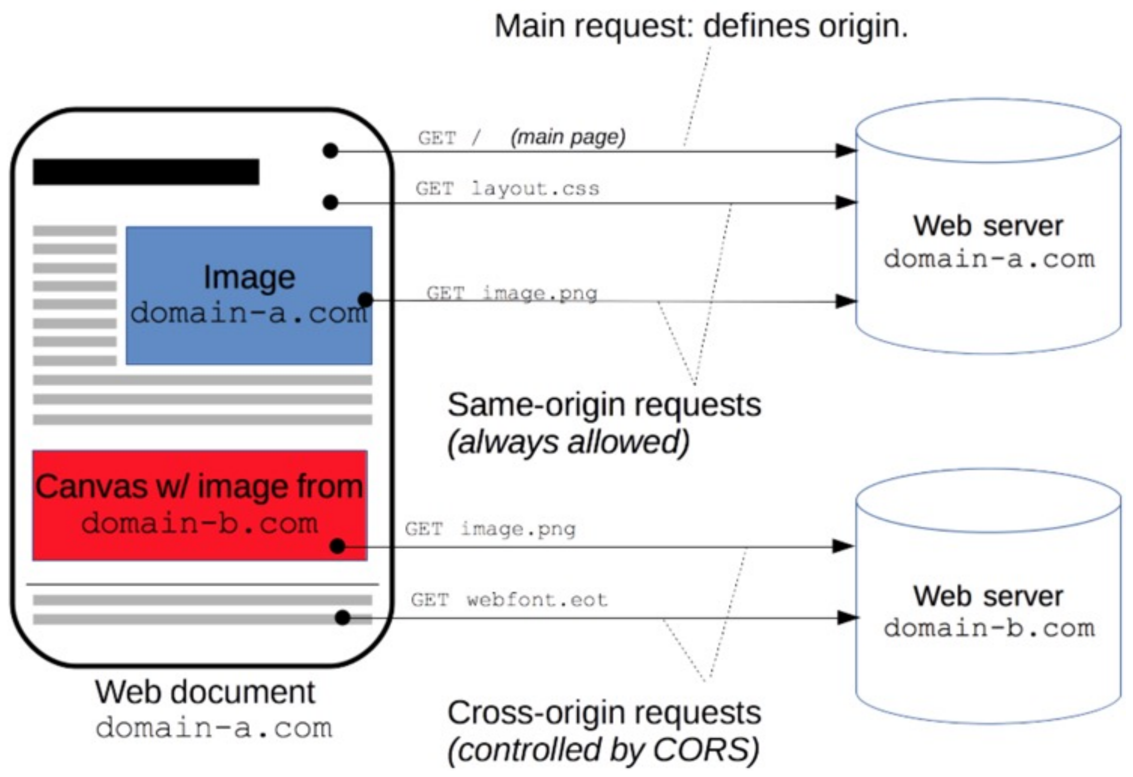
https://portswigger.net/web-security/cors/same-origin-policy

Cross-Origin Security Risks

- Loading page resources (images, CSS, JS) from different origins (cross-origin requests) without restrictions is a **huge security risk**
  - Browsers always send any cookies relevant to the domain with any request
  - If valid authentication/session cookies are sent, a request could operate from one origin (**example.com**) but as authenticated in another (**bank.com**)
- However, sometimes it is useful to load resources from other origins
  - Other subdomains/ports than the original one, e.g., “static-content.example.com”
  - Other domains: content delivery networks – “CDNs” (common libraries, etc.), public services information (weather, news, stock values, etc.), content provided by third parties (advertisement, etc.)
  - API servers, either in your network (but different server) or publicly accessible

Solving the Cross-Origin Problem: CORS

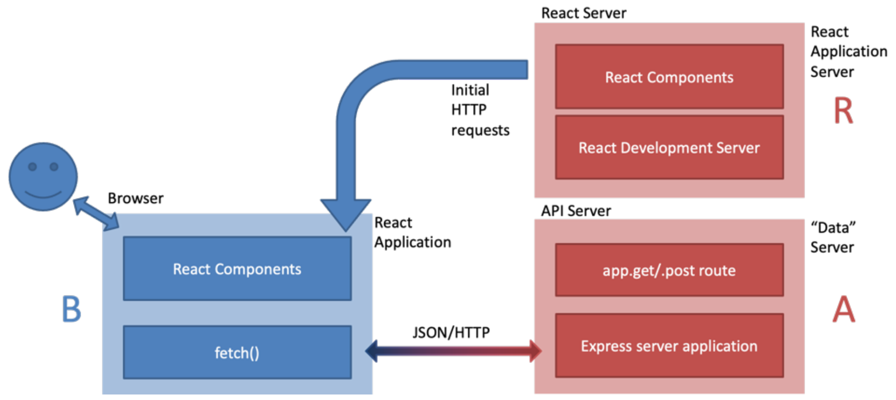
- Cross-Origin Resource Sharing (**CORS**): a **standard** mechanism to **implement cross-domain requests**
- CORS defines a **set of HTTP headers** that allow the browser and server to communicate about which requests are (or are not) allowed
- The **server** defines which origins are accepted for any request


Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image domain-a.com
Web server domain-a.com
Same-origin requests (always allowed)
Canvas w/ image from domain-b.com
GET image.png
GET webfont.eot
Web server domain-b.com
Web document domain-a.com
Cross-origin requests (controlled by CORS)

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

https://fetch.spec.whatwg.org/#http-cors-protocol

CORS in Practice



- The browser knows that the request is addressed to a different origin ("A"), i.e., it is a CORS request
- The browser allows to send the request, but it includes the origin "R" in the CORS request via a specific header:

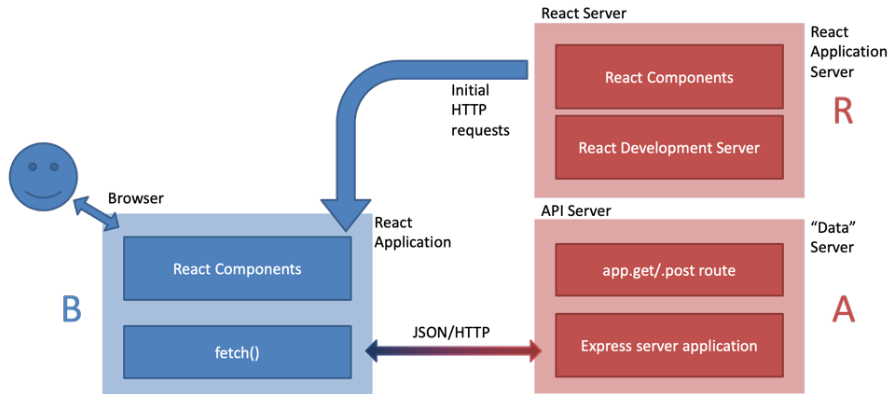
```
Origin: https://foo.example
```

- The *Response* from the receiving server "A" includes, via another header, which origins ("R") can do the request to "A":

```
Access-Control-Allow-Origin: https://foo.example
```

- If the two match, the **browser** allows the script to access the response, otherwise the content, even if received by the browser, will not be passed to the script: the request will appear as "failed to load" for the script

CORS usefulness in the browser



- CORS DOES NOT APPLY when making requests outside browsers
  - curl, wget, REST Client, etc. do not care about the extra headers
  - origin = null in this case (i.e., as seen from the server)
- CORS works as an agreement between server and browser, where the server tells the browser for which origin it is safe to use its content
  - A way to overcome the limitations of the Same Origin Policy (SOP) when server "A" is trusted by applications served from "R"
  - A way to avoid that the scripts in browser "B", loaded from another origin "X" (instead of "R"), can use information coming from server "A" (as if they were "R")

CORS Preflight Requests

- CORS requests, by definition, interact with a different origin
- Requests might leak *private information* if sent to untrusted receivers
- For this reason, before sending requests with *private/sensitive information*, the browser checks if such a request is safe to send by means of the HTTP “OPTION” method
  - Example: when doing POST, PUT, or requests with special (e.g. cookie) headers
- Such cross-site requests are said "*preflighted*"
- This is **decided and performed automatically** by the browser
  - Need to know because it might impact application performance (each time two HTTP transactions happen instead of one)

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

CORS Preflight example

```
// fetch https://api.com/the/resource/you/request with method POST

OPTIONS /the/resource/you/request
Access-Control-Request-Method: POST
Access-Control-Request-Headers: origin, x-requested-with, accept
Origin: https://your-origin.com
```

```
// Response from https://api.com/the/resource/you/request

HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://your-origin.com
Access-Control-Allow-Methods: POST, GET, OPTIONS, DELETE
```

https://flaviocopes.com/express-cors/

CORS with Authentication

- By default, fetch requests do not send credentials (e.g., cookies)
- If needed, `fetch` has an option in the `init` object to include them
- Values: `'omit'` (default), `'same origin'` (send only in requests to the same origin), `'include'`

```
fetch('https://example.com', {
  …
  credentials: 'include'
  …
});
```

CORS with <scripts> tag

- Scripts loaded via `<script>` tag from other origins run with the same privileges of the other scripts in the web application
- Only load scripts you trust, and always include integrity check to prevent malicious code injection
  - To make the integrity attribute work, the `crossorigin` attribute MUST be set (to “anonymous”), otherwise the resource will be blocked
    (beware: disable browser cache when testing such things)
  - Note that the server must have used: `Access-Control-Allow-Origin: *`

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
crossorigin="anonymous"></script>
```

Loading modules via script tag

- Modules loaded via `<script>` are subject to CORS rules
- Unfortunately, modules loaded from the file system have **origin `null`** so browsers prevent module loading **even from local file system** (file:// URI)
- Solution: serve content from a (local) web server as static files

```
<body>
 ...
 <script type="module" src="main.js"></script>
 <script type="module" src="index.js"></script>
</body>
```

```
// In index.js:
import * as jsdom from 'main.js';  // import requires the .js be loaded as a module
```

https://github.com/expressjs/cors

https://flaviocopes.com/express-cors/

Controlling Allowed Origins in your API Server

CORS ON THE SERVER SIDE

Enabling CORS in Express application

- Use the middleware `cors`
  - http://expressjs.com/en/resources/middleware/cors.html
  - `npm install cors`

```
const express = require('express');
const cors = require('cors');
const app = express();

app.use(cors());  // Careful: enables all origins
```

Simple Usage

- Enable **CORS for all** requests (for the app server)

```
app.use(cors())
```

  By default, **all origins** will be enabled for all HTTP methods

  Also, enabling it as application-level middleware (i.e., `app.use(…)`) automatically handles <u>preflight</u> requests <u>for all routes</u>

- Enable CORS for a **Single Route**

```
app.get('/products/:id', cors(), function (req, res, next) {
  …
  res.json({msg: 'This is CORS-enabled for a Single Route'})
})
```

http://expressjs.com/en/resources/middleware/cors.html#enabling-cors-pre-flight

Configuration options

- The `cors(options)` call accepts a JS configuration object
- Always specify the allowed origins (as a string, function, regexp, array)
  - E.g., `"origin": "https://appwebsite.com"`
- Specify the allowed methods
- Fine-tune allowed headers and credentials *(more on this later)*

Default configuration options

```
{
  "origin": "*",
  "methods": "GET,HEAD,PUT,PATCH,POST,DELETE",
  "preflightContinue": false,
  "optionsSuccessStatus": 204
}
```

https://expressjs.com/en/resources/middleware/cors.html

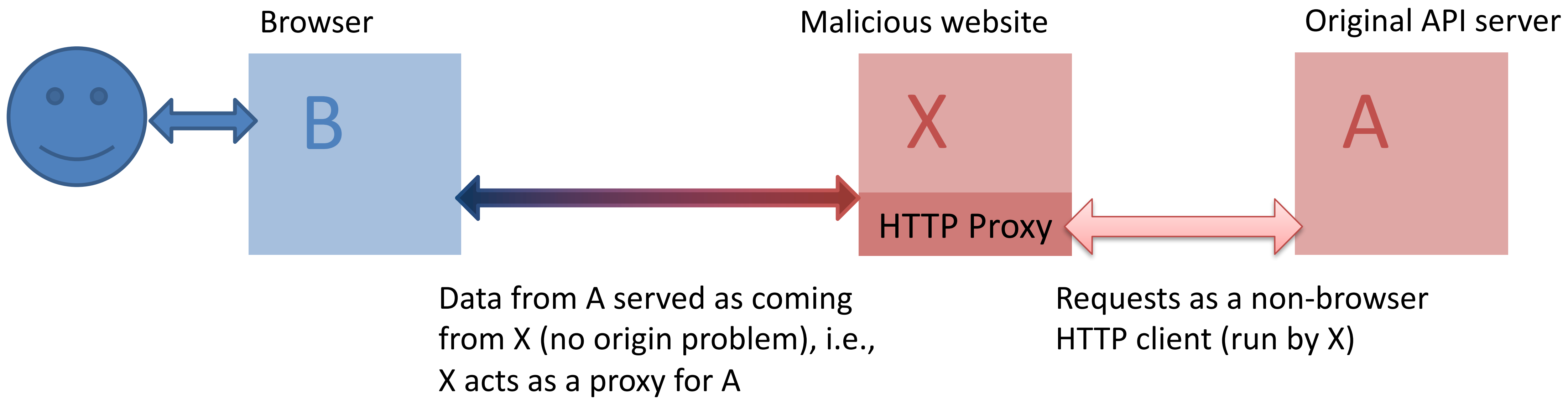
CORS Summary

- Loading a resource from another site (i.e., origin) <u>that you trust</u>
  - What is **their** CORS policy?

- Another site wants to load data from your API server
  - What is **your** CORS policy (i.e., sites you trust they do not mishandle your data)?

CORS and API Security

- CORS is just a mitigation strategy
- A malicious attacker X can always act as a proxy for API server requests
- API security must rely on other mechanisms, e.g. authorization!


Browser
Malicious website
Original API server
B
X
A
HTTP Proxy
Data from A served as coming from X (no origin problem), i.e., X acts as a proxy for A
Requests as a non-browser HTTP client (run by X)

https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny

References

- A tutorial on CORS
  - https://auth0.com/blog/cors-tutorial-a-guide-to-cross-origin-resource-sharing/
- https://portswigger.net/web-security/cors
- https://github.com/expressjs/cors
- https://flaviocopes.com/express-cors/
- https://owasp.org/www-community/attacks/CORS_OriginHeaderScrutiny
- https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

License