

<WA1/>
<AW1/>
2026

Client-Server Interaction in React

Connecting React to HTTP APIs

Fulvio Corno
Luigi De Russis



Outline

- The “two servers” problem
 - Two servers + CORS → we will use this, in the course
 - Build + Express (single server)
 - Also: Understanding Build (webpack, imports, ...)



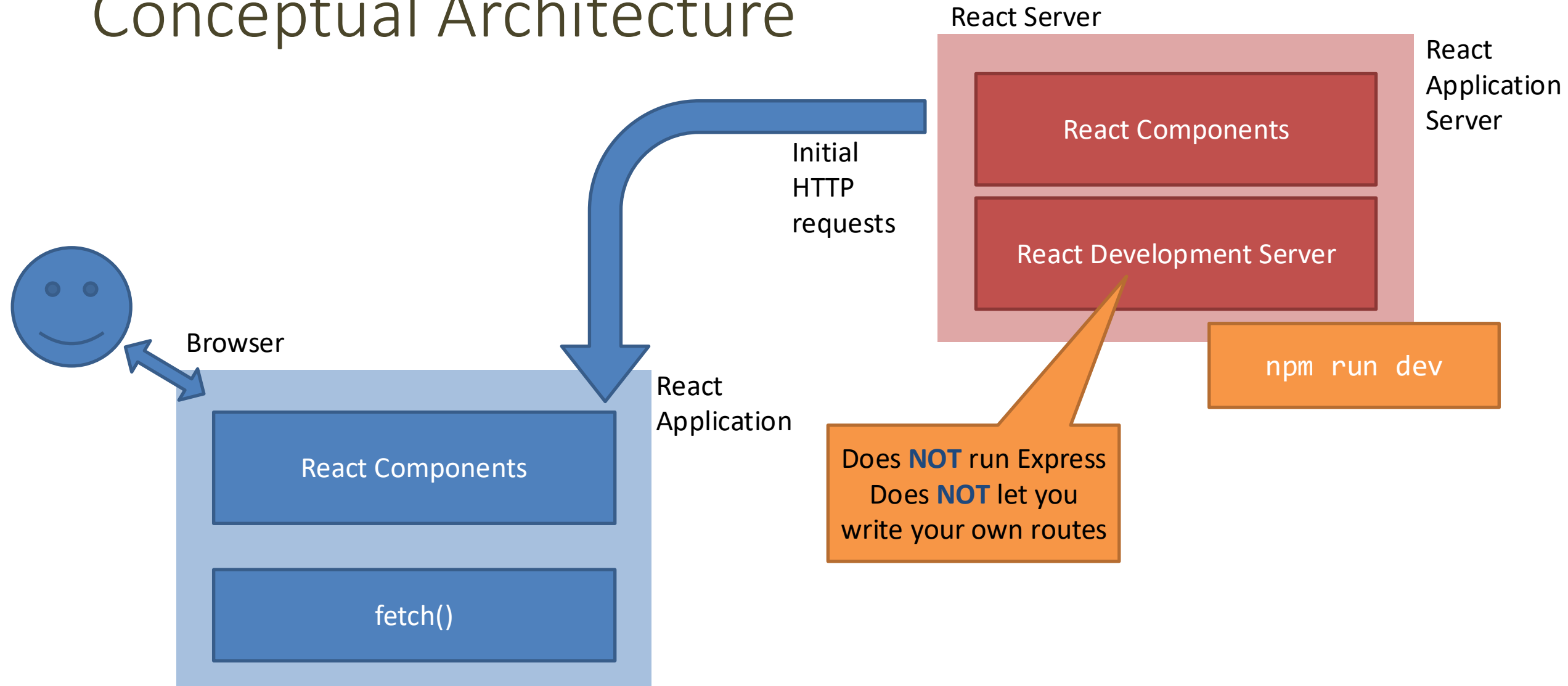
<https://www.robinwieruch.de/react-fetching-data>

Full Stack React, Chapter “Using Webpack with Create React App”

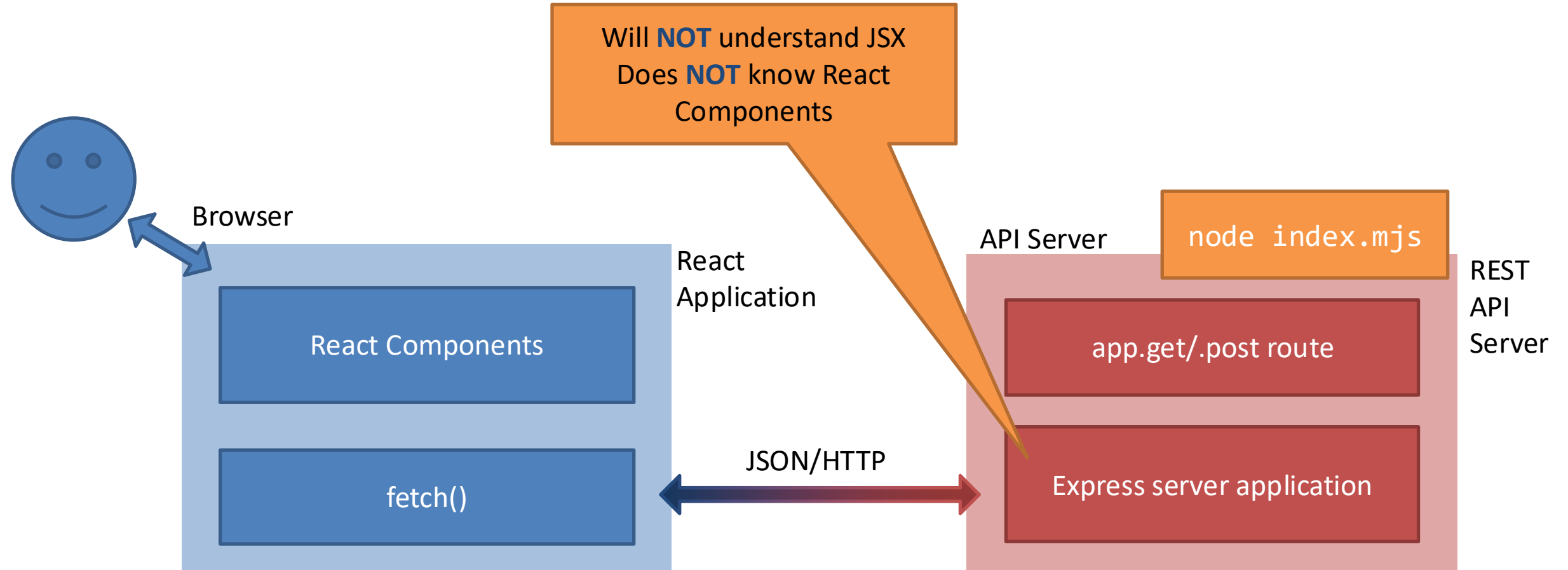
A Client and a Server walk into a bar...

THE “TWO SERVERS” PROBLEM

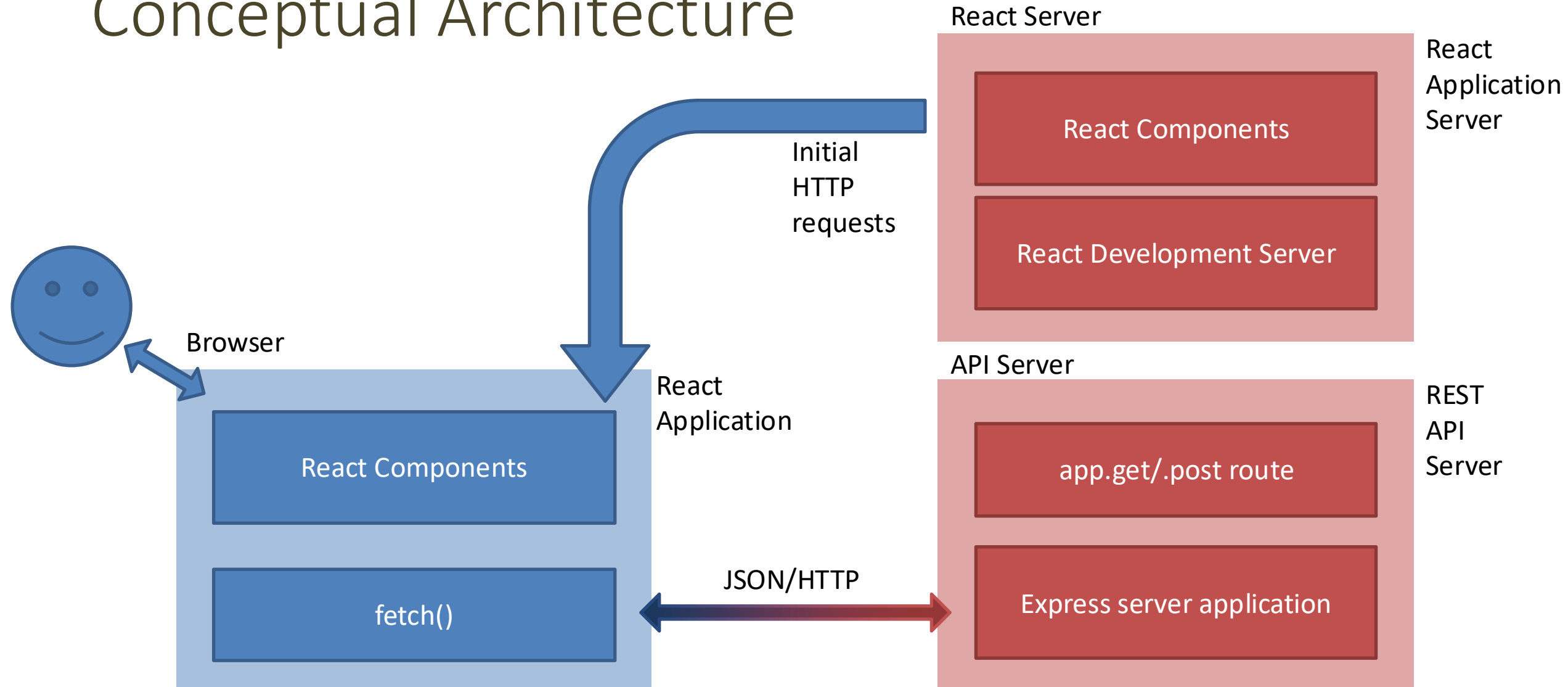
Conceptual Architecture



Conceptual Architecture



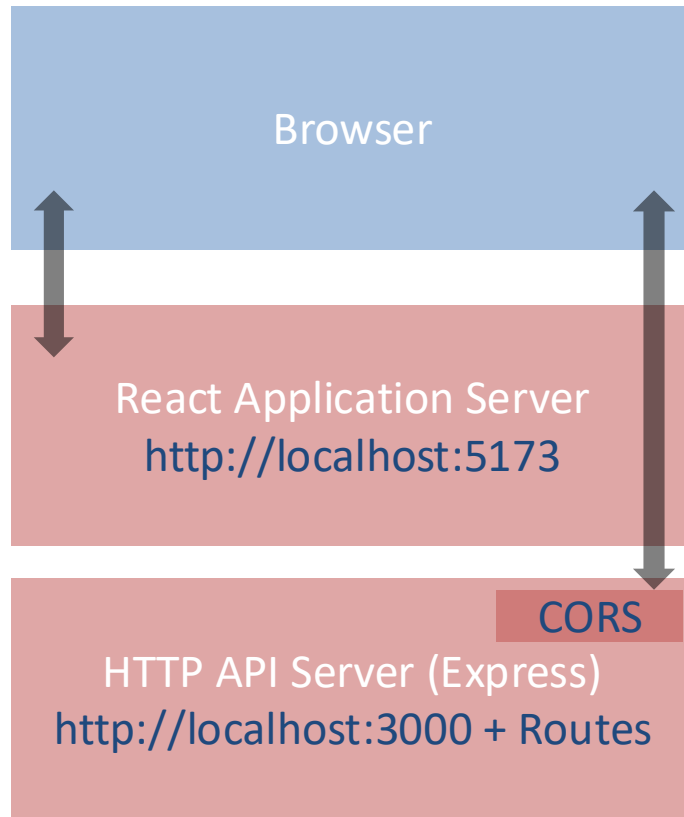
Conceptual Architecture



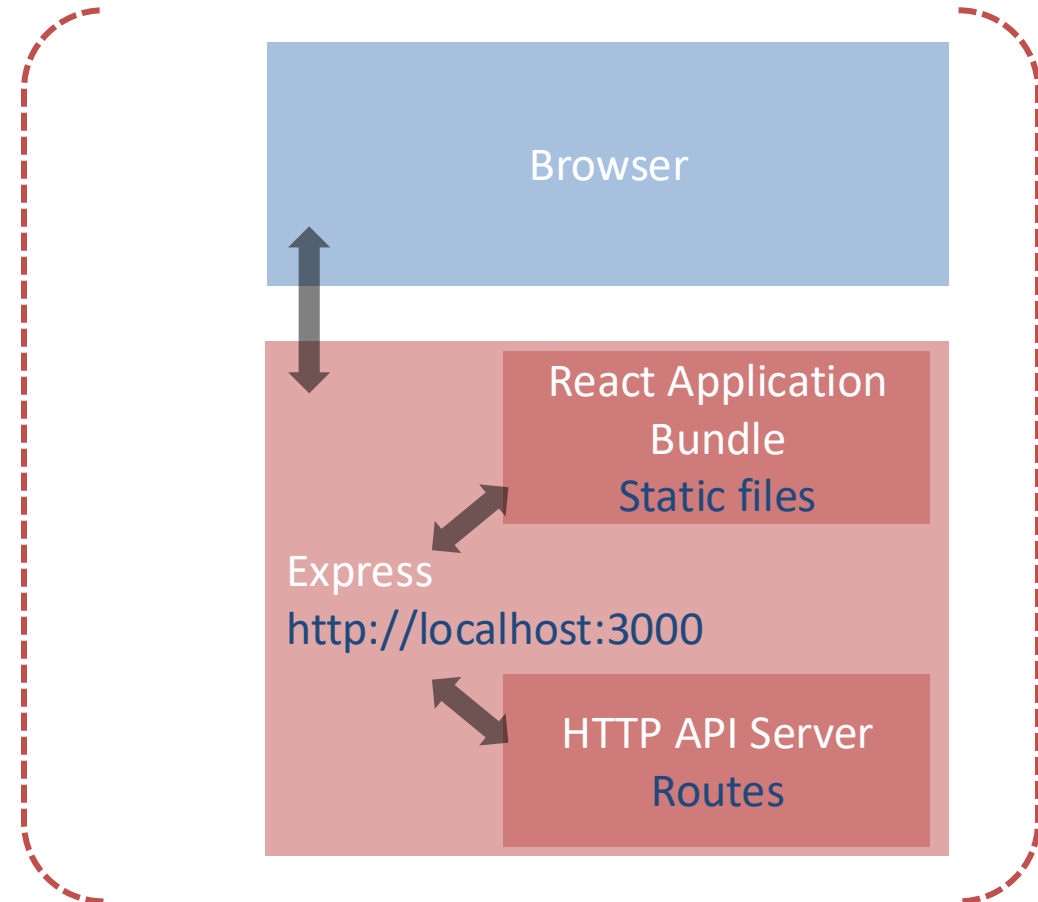
Issues

- Deployment
 - One-server-does-all or two-separate-servers?
 - Development vs. Production trade-off
 - convenience/debug/turnaround time vs performance/security
 - Cross-Origin security limitations
- Opportunities
 - Separate the load
 - Use any API Server (even 3rd party ones)

Two Possible Solutions



Two independent servers +
CORS configuration



Build a production bundle
and host it in a web server

We will use this, in
the course



<https://www.newline.co/fullstack-react/articles/using-create-react-app-with-a-server/>

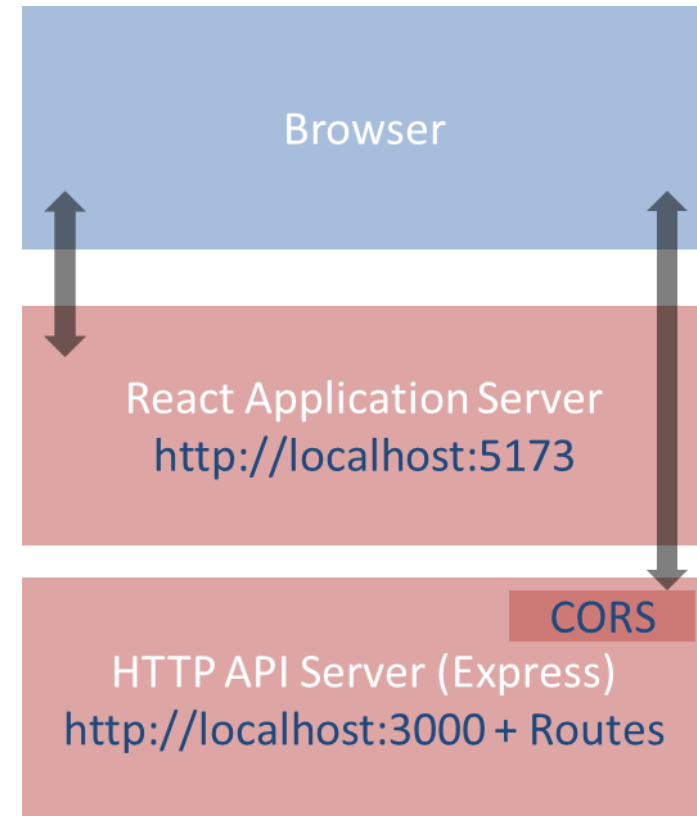
Full Stack React, Chapter “Using Webpack with Create React App / Using Create React App with an API server”

Side-by-side deployment

RUNNING TWO SEPARATE SERVERS

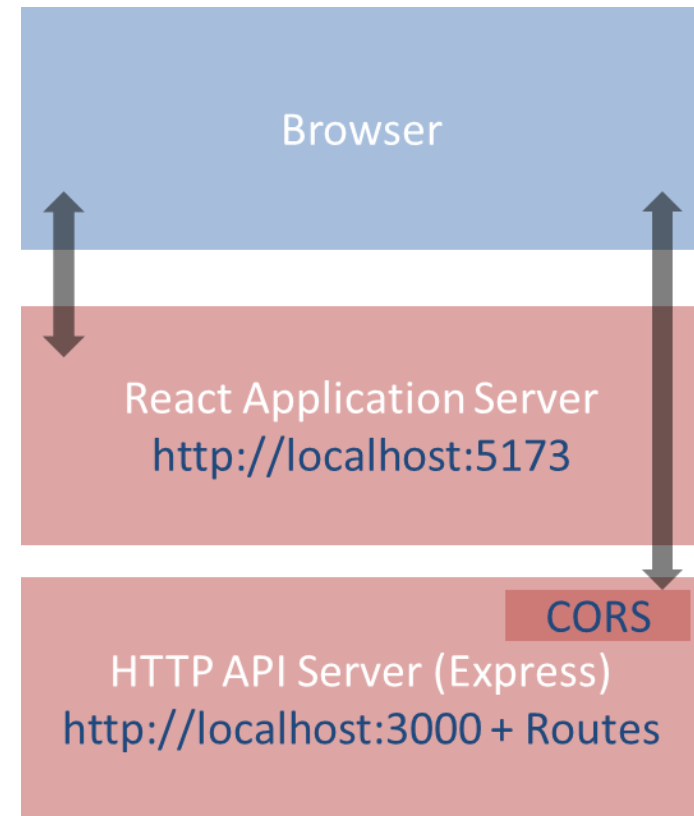
Double-Server Setup

- React Web Server and HTTP API server are hosted separately
 - Different hosts, and/or
 - Different ports
- The browser:
 - Receives the React application
 - Directs the API requests to the API server



Double-Server Setup

- Must run two web servers
 - React project: `npm run dev`
 - Express project: `node index.js`
 - Two projects, in two different directories (or different servers)
- Problem: handle **CORS**
 - *Cross-Origin Resource Sharing*
 - Default security policy prevents loading data from other servers
 - Details not discussed here



Advantages and Disadvantages

- Servers are easy to deploy
- Scalable solution: requests are sent to the appropriate server
- Only possible configuration if the HTTP API is provided by a third party
 - Public APIs
- Need to configure cross-origin resource sharing (CORS) on API server
- Requires using absolute URLs to access APIs
- Wrongly configured CORS might be a security risk (undesired access to APIs from e.g., mock websites)

How To Configure

- Configure CORS on API server for development

```
// index.mjs (node express server)
import cors from 'cors'; // npm install cors

//Enable All CORS Requests (for this server)
app.use(cors());
//Use ONLY for development, otherwise restrict domain
```

- In production mode, use different domains for React and API servers, NEVER allow CORS requests from any origin, always specify origin
 - See also <https://github.blog/security/application-security/localhost-dangers-cors-and-dns-rebinding/>

Example

API.mjs in the React Application

```
const APIURL=new URL('http://localhost:3000');

async function getCourses() {
  return fetch(new URL('/courses', APIURL))
    .then((response)=>{
      if(response.ok) {
        return response.json() ;
      } else {
        throw response.statusText;
      }
    })
    .catch((error)=>{
      throw error;
    });
}
```

Called in useEffect()

index.mjs for the API Server

```
import express from 'express';
import cors from 'cors';

const app = express();
const port = 3000;
app.use(cors());

app.get('/courses', (req, res) => {
  dao.listCourses()
    .then((courses) => res.json(courses))
    .catch((dbErrorObj)=>
      res.status(503)
        .json(dbErrorObj));
});

app.listen(port, () => console.log(`Example app
listening at http://localhost:${port}`));
```

Calls dao.mjs



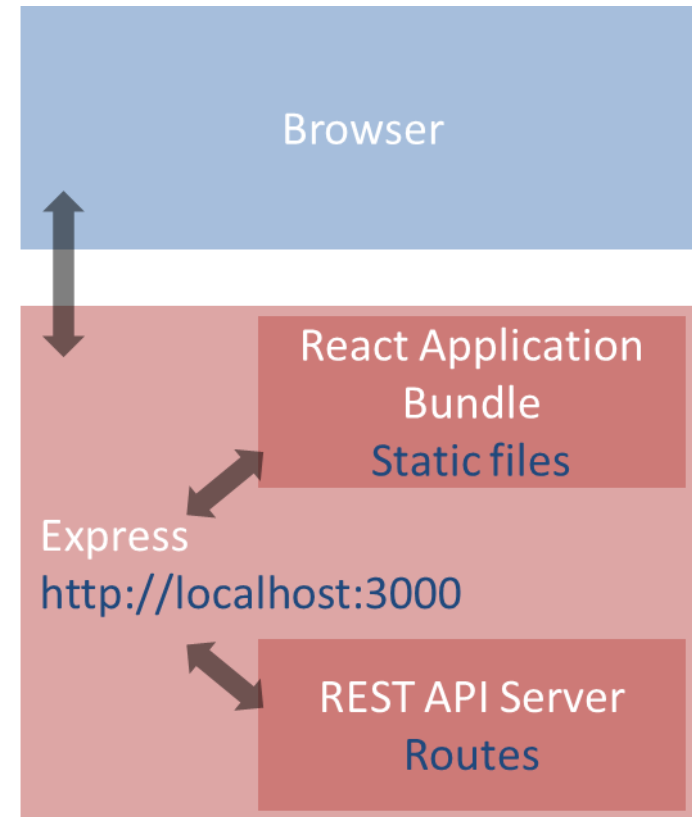
<https://vitejs.dev/guide/static-deploy.html>

Packing and moving the React application into any web server

DEPLOYING A BUILD INSIDE A SERVER

Deploying the React Bundle

- React does not need to run in the development server
- `npm run build` will create a “production bundle” with all the contents needed to run the application
- This bundle is composed of static files (html, js, assets) and may be served by *any webserver* (including Apache, nginx, express, php, ...)



Build Command

```
npm run build
```

```
[luigi@meletta react-qa]$ npm run build
> react-qa@0.0.0 build
> vite build

vite v4.2.1 building for production...
✓ 340 modules transformed.
dist/index.html                                0.39 kB
dist/assets/bootstrap-icons-cfe45b98.woff2    121.34 kB
dist/assets/bootstrap-icons-999550fa.woff     164.36 kB
dist/assets/index-4e55b3b0.css                 274.30 kB | gzip: 40.19 kB
dist/assets/index-b55e27f4.js                 210.40 kB | gzip: 69.73 kB
✓ built in 1.02s
```

Creates everything
under `./dist`

<https://vitejs.dev/guide/static-deploy.html>

What Does “build” Do?

- Most of the work in “building” the static application is done by Babel and Webpack
 - Babel translates all JSX (and new JS syntax) into basic JS (according to the ‘production’ property in package.json)
 - Webpack packs and minimizes all JS code into a single file
 - Prepares an index.html that loads all the JS code
- The content of the “dist” folder is self-contained and may be moved to the deployment server
- All debugging capabilities are removed

Check the Build Results

- You may test the built app by running `npm run preview`
- The vite's preview command will launch a local static web server
 - serving the files from “dist” at <http://localhost:4173>

Hosting The Build in Express

- `cd express-api-server`
- `cp -r ../react-app/dist .`
- Define a static route in `server.js`

```
app.use(express.static('./build'));  
  
app.get('/', (req, res)=> {res.redirect('/index.html')} );
```

- In the application, you may call APIs locally
 - `fetch('/api/questions')...`

Hosting the Build in Online Services

- Different online services allow free hosting of static websites, e.g.,
 - GitHub Pages, GitLab Pages, Firebase, Vercel, etc.
- Some of them are free or have a free tier.
- To host the build on such services, refer to the guide at <https://vitejs.dev/guide/static-deploy.html>.

Pros and Cons

- Simple to deploy the final application (anywhere)
- May include the application inside the API server (in production, too)
- The JS code runs on every browser (thanks to polyfills and transpiling)

- The build cannot be directly modified
- Need a save/build/copy/reload cycle for every modification

Other “Magic” By Webpack

- Packing of all imported modules
- Bundling of Assets
 - Images
 - CSS files
- CSS Modules

In Development Mode...

- `npm run dev` runs the “Webpack development server” (WDS)
- All our code is transpiled and packed into a `bundle.js` that is automatically inserted into `index.html`
 - Contains all our code, plus React, plus imported modules
 - Also handles imports of non-JS files
- `bundle.js` does not exist – it’s kept in-memory by the WDS
- Sets up hot-reloading and synchronized error messages (via websockets)

Imports in Webpack

- `import logo from './logo.svg';`
- `import logo from './logo.png';`
 - Will include the image reference inside the bundle (placed under static/media)
 - Small files are rendered inline
- `import './Button.css';`
 - This component will use these CSS declarations
 - All CSS will be concatenated into a single file, but here we are stating the dependency
- `import styles from './Button.module.css';`
 - Files ending with `.module.css` are CSS modules
 - Styles may be applied with `className={styles.primary}`
 - Class names are *renamed to be unique*: no conflict with other Components' styles

Why Use Imports

- Scripts and stylesheets get minified and bundled together to avoid extra network requests.
- Missing files cause compilation errors instead of 404 errors for your users.
- Result filenames include content hashes, so you do not need to worry about browsers caching their old versions.
- They are an optional mechanism. “Traditional” loading (with link) still works, if you save your files in the public directory



License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
 - **Share** — copy and redistribute the material in any medium or format
 - **Adapt** — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
 - **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - **NonCommercial** — You may not use the material for [commercial purposes](#).
 - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.
 - **No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

