Unbundled code splitting strategy

@Seia-Soto (Ho-Jeong Go)

Bundling with Webpack

Being old but still the strategy.

- HTTP/1.1 is not made for sending multiple files at the time.
- Webpack is a solution for tying various dependencies and code, which called 'bundling'.

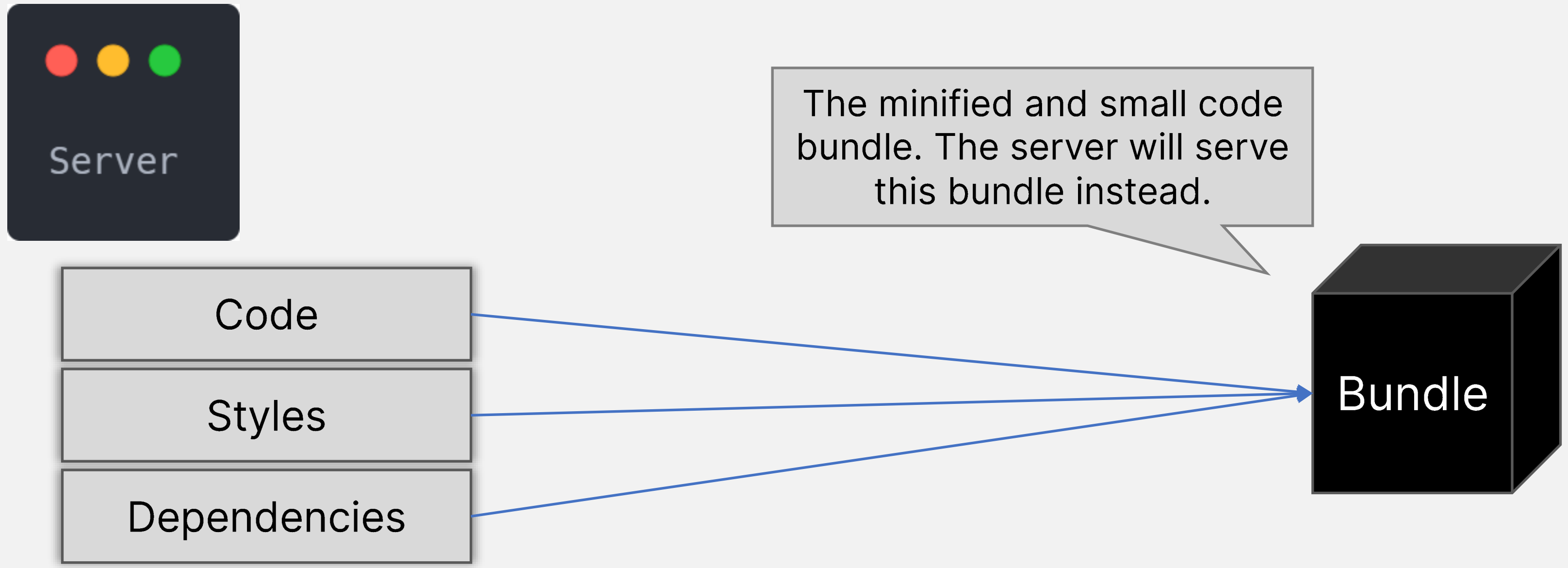
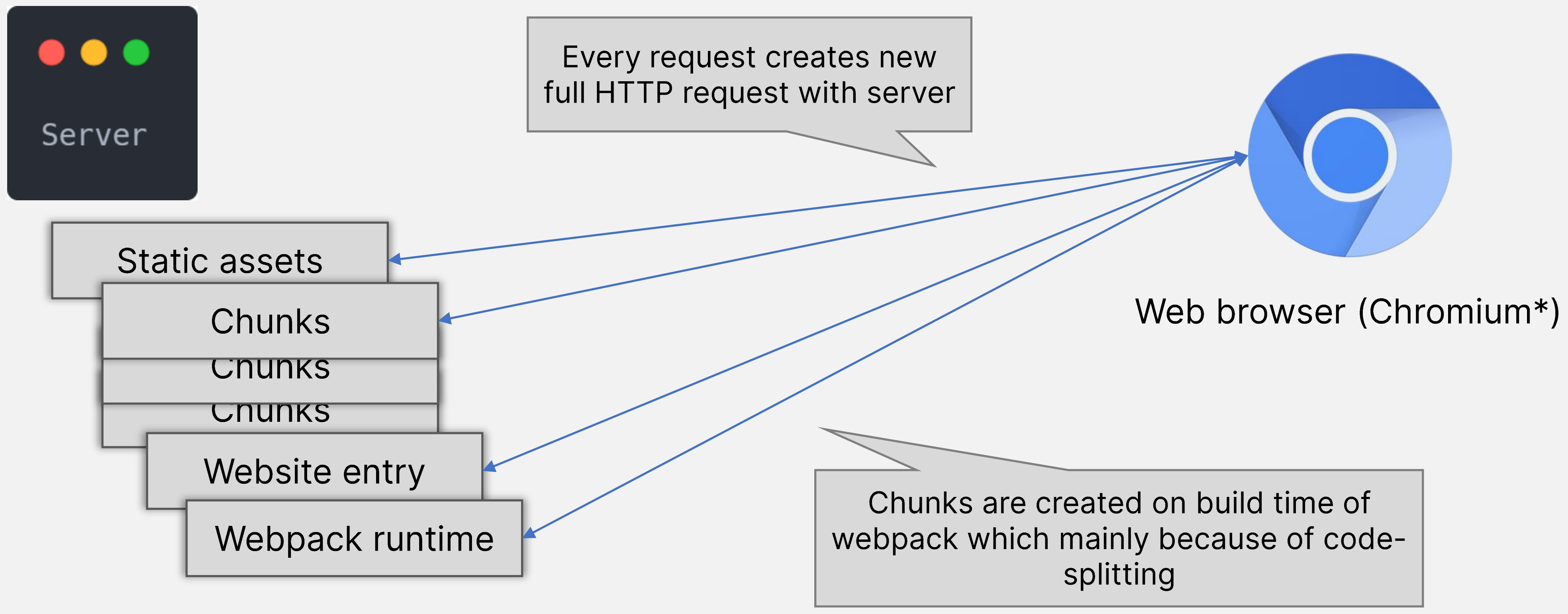

Server
Code
Styles
Dependencies
The minified and small code bundle. The server will serve this bundle instead.
Bundle

Figure 1. Webpack with Client

Code splitting on Webpack

The basic code-splitting technology.

- By lazy loading components on FE part, you can split the codes with *chunks*.
- The purpose of code-splitting is avoiding sending all codes which includes useless ones to end-user.



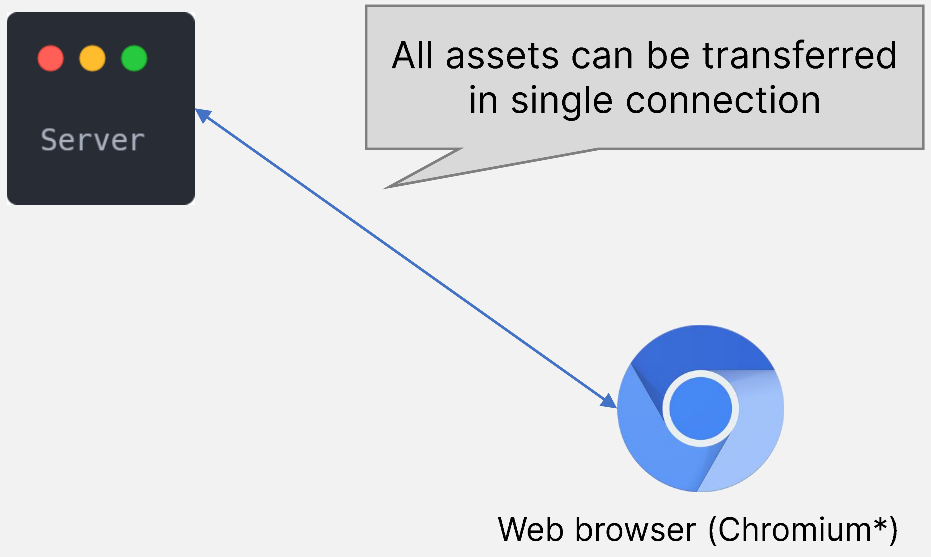
Server
Every request creates new full HTTP request with server
Static assets
Chunks
Chunks
Chunks
Website entry
Webpack runtime
Web browser (Chromium*)
Chunks are created on build time of webpack which mainly because of code-splitting

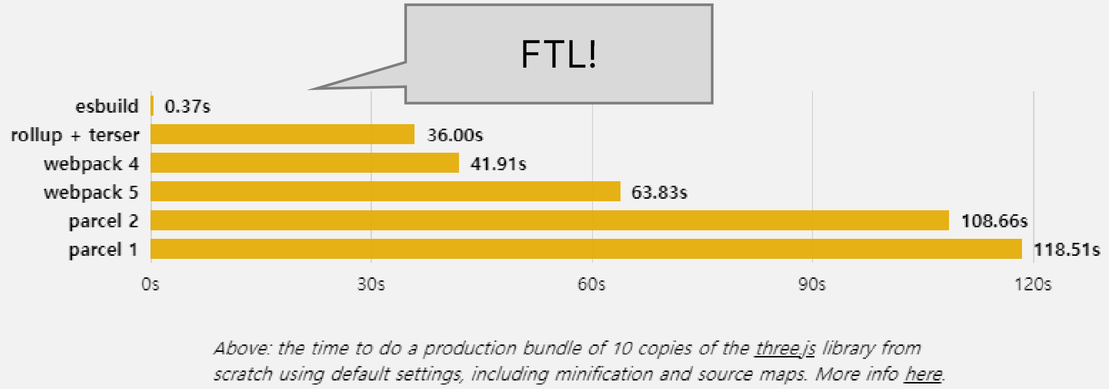
Figure 2. Webpack code-splitting

Huge changes on current web environment

A lot of new technology for developing web applications are created.

- HTTP/2 can ship many materials such as your code with single *connection*.
- ESBuild is faster than any existing bundlers such as webpack or rollup.





Above: the time to do a production bundle of 10 copies of the three.js library from scratch using default settings, including minification and source maps. More info here.

Figure 3. New technologies

See experience with bundlers

It’s not only you want, avoid using it all even in useless condition.

- Bundlers contains its runtime, also has the impact on your website.
- The build time is really *longer* than ESBuild which I introduced in prior page.
- Webpack says that it bundles all your assets but it isn’t good always.

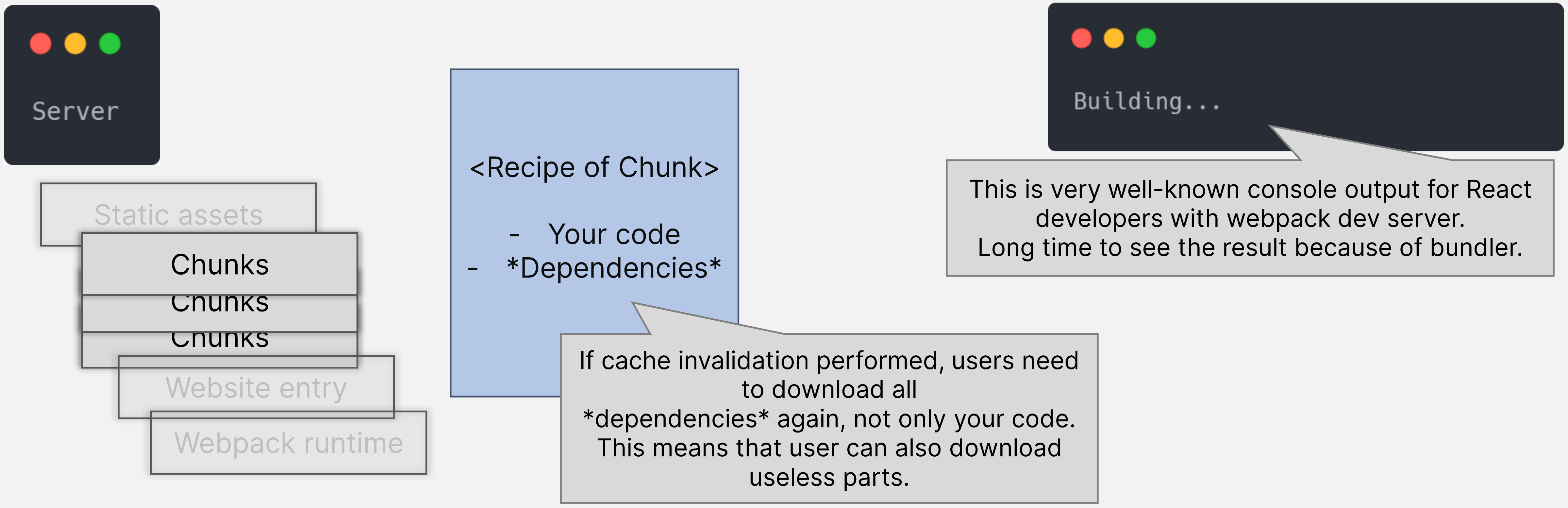
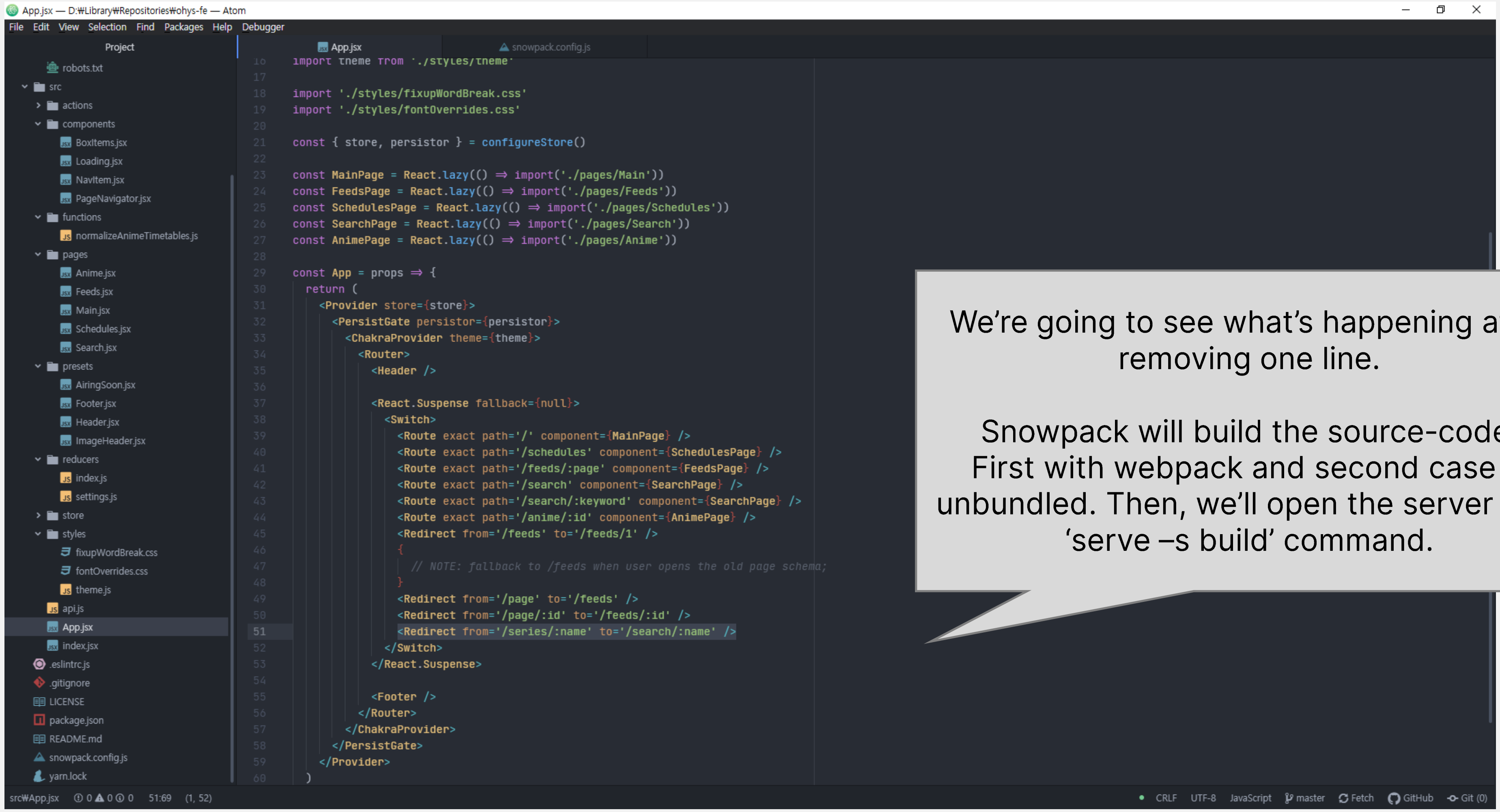

Server
Static assets
Chunks
Chunks
Chunks
Website entry
Webpack runtime
<Recipe of Chunk>
- Your code
- *Dependencies*
If cache invalidation performed, users need to download all *dependencies* again, not only your code. This means that user can also download useless parts.
Building...
This is very well-known console output for React developers with webpack dev server. Long time to see the result because of bundler.

Figure 4. Problems with bundler

ESC

Testing with actual project

Ohys-FE.



App.jsx — D:₩Library₩Repositories₩ohys-fe — Atom

```jsx
import theme from './styles/theme'

import './styles/fixupWordBreak.css'
import './styles/fontOverrides.css'

const { store, persistor } = configureStore()

const MainPage = React.lazy(() => import('./pages/Main'))
const FeedsPage = React.lazy(() => import('./pages/Feeds'))
const SchedulesPage = React.lazy(() => import('./pages/Schedules'))
const SearchPage = React.lazy(() => import('./pages/Search'))
const AnimePage = React.lazy(() => import('./pages/Anime'))

const App = props => {
  return (
    <Provider store={store}>
      <PersistGate persistor={persistor}>
        <ChakraProvider theme={theme}>
          <Router>
            <Header />

            <React.Suspense fallback={null}>
              <Switch>
                <Route exact path='/' component={MainPage} />
                <Route exact path='/schedules' component={SchedulesPage} />
                <Route exact path='/feeds/:page' component={FeedsPage} />
                <Route exact path='/search' component={SearchPage} />
                <Route exact path='/search/:keyword' component={SearchPage} />
                <Route exact path='/anime/:id' component={AnimePage} />
                <Redirect from='/feeds' to='/feeds/1' />
                {
                  // NOTE: fallback to /feeds when user opens the old page schema;
                }
                <Redirect from='/page' to='/feeds' />
                <Redirect from='/page/:id' to='/feeds/:id' />
                <Redirect from='/series/:name' to='/search/:name' />
              </Switch>
            </React.Suspense>

            <Footer />
          </Router>
        </ChakraProvider>
      </PersistGate>
    </Provider>
  )
```

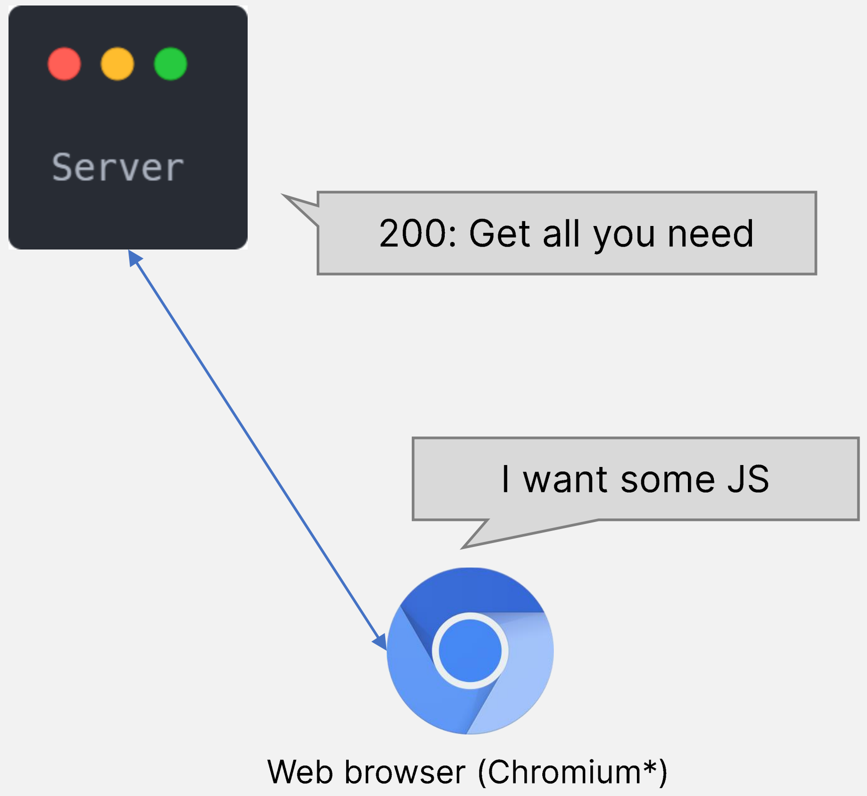
We're going to see what's happening after removing one line.

Snowpack will build the source-code. First with webpack and second case is unbundled. Then, we'll open the server with 'serve –s build' command.

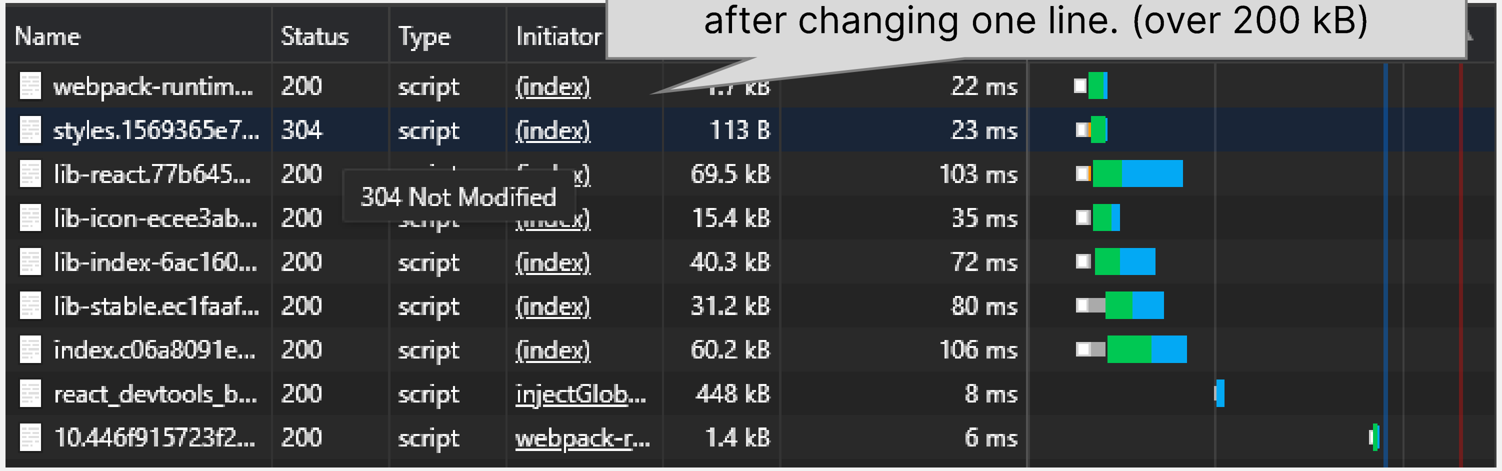
Figure 5. Creating test

Useless requests are made

Lazy loading components are not enough with bundlers.



Web browser doesn't know the exact location of dependencies and all things are re-cached after changing one line. (over 200 kB)



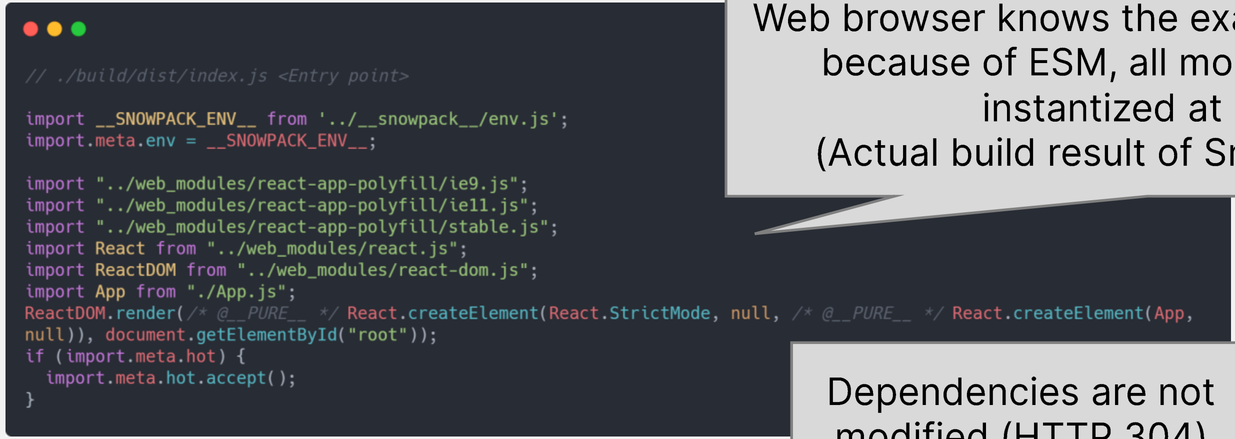
Name	Status	Type	Initiator		
webpack-runtim...	200	script	(index)	1.7 kB	22 ms
styles.1569365e7...	304	script	(index)	113 B	23 ms
lib-react.77b645...	200	script	(index)	69.5 kB	103 ms
lib-icon-ecee3ab...	200	script	(index)	15.4 kB	35 ms
lib-index-6ac160...	200	script	(index)	40.3 kB	72 ms
lib-stable.ec1faaf...	200	script	(index)	31.2 kB	80 ms
index.c06a8091e...	200	script	(index)	60.2 kB	106 ms
react_devtools_b...	200	script	injectGlob...	448 kB	8 ms
10.446f915723f2...	200	script	webpack-r...	1.4 kB	6 ms

304 Not Modified

Figure 6. Browser doesn't know webpack

Code-splitting becomes easy without bundlers

Lazy loading component is enough. No more.



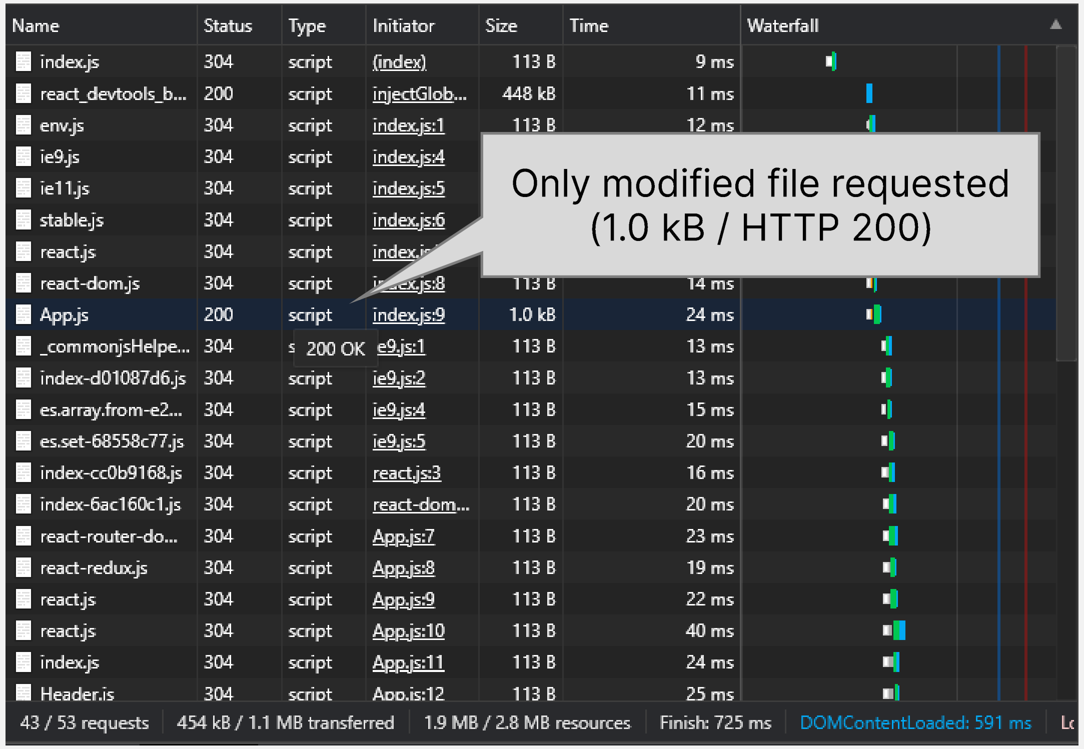
```
// ./build/dist/index.js <Entry point>

import __SNOWPACK_ENV__ from '../__snowpack__/env.js';
import.meta.env = __SNOWPACK_ENV__;

import "../web_modules/react-app-polyfill/ie9.js";
import "../web_modules/react-app-polyfill/ie11.js";
import "../web_modules/react-app-polyfill/stable.js";
import React from "../web_modules/react.js";
import ReactDOM from "../web_modules/react-dom.js";
import App from "./App.js";
ReactDOM.render(/* @__PURE__ */ React.createElement(React.StrictMode, null, /* @__PURE__ */ React.createElement(App,
null)), document.getElementById("root"));
if (import.meta.hot) {
  import.meta.hot.accept();
}
```

Web browser knows the exact location of each JS and because of ESM, all modules won't loaded and instantized at initiating time.
(Actual build result of Snowpack app: Ohys-FE)

Dependencies are not modified (HTTP 304)



Name	Status	Type	Initiator	Size	Time	Waterfall
index.js	304	script	(index)	113 B	9 ms	
react_devtools_b...	200	script	injectGlob...	448 kB	11 ms	
env.js	304	script	index.js:1	113 B	12 ms	
ie9.js	304	script	index.js:4			
ie11.js	304	script	index.js:5			
stable.js	304	script	index.js:6			
react.js	304	script	index.j			
react-dom.js	304	script	index.js:8	113 B	14 ms	
App.js	200	script	index.js:9	1.0 kB	24 ms	
_commonjsHelpe...	304	s 200 OK	e9.js:1	113 B	13 ms	
index-d01087d6.js	304	script	ie9.js:2	113 B	13 ms	
es.array.from-e2...	304	script	ie9.js:4	113 B	15 ms	
es.set-68558c77.js	304	script	ie9.js:5	113 B	20 ms	
index-cc0b9168.js	304	script	react.js:3	113 B	16 ms	
index-6ac160c1.js	304	script	react-dom...	113 B	20 ms	
react-router-do...	304	script	App.js:7	113 B	23 ms	
react-redux.js	304	script	App.js:8	113 B	19 ms	
react.js	304	script	App.js:9	113 B	22 ms	
react.js	304	script	App.js:10	113 B	40 ms	
index.js	304	script	App.js:11	113 B	24 ms	
Header.js	304	script	App.js:12	113 B	25 ms	

43 / 53 requests | 454 kB / 1.1 MB transferred | 1.9 MB / 2.8 MB resources | Finish: 725 ms | DOMContentLoaded: 591 ms | Lo

Only modified file requested
(1.0 kB / HTTP 200)

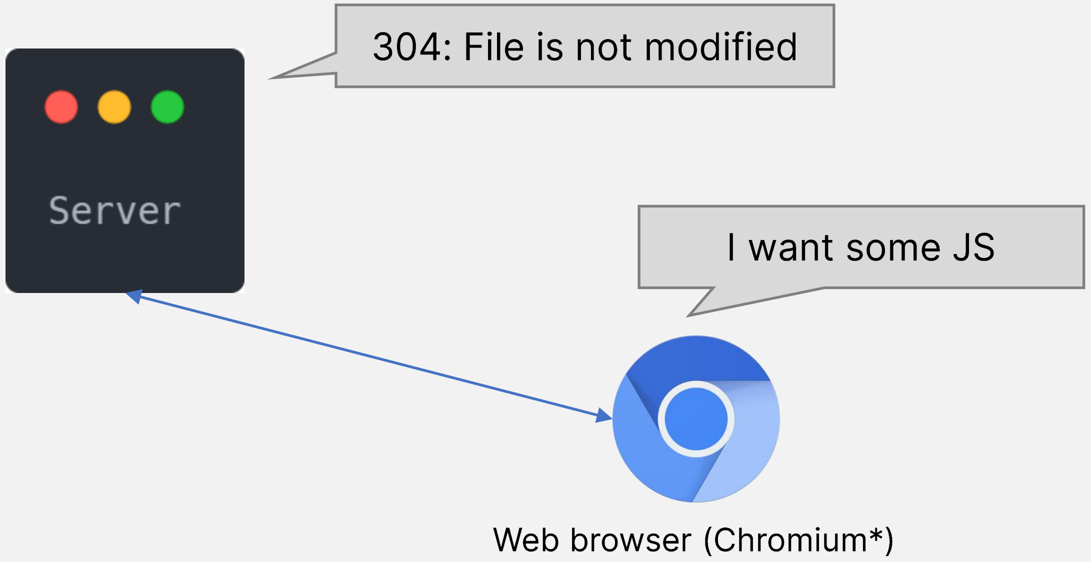


Figure 7. Web browser know this application

Traffic controlling via CDN on micro-frontends

All core assets are shared via CDN, this make apps centralized but divided at the time.

- Thanks to **zhoukekestar**, similar ideas are already formed nicely.
- Optimization of traffic will be reached by CDN providers.
- Each micro-services (or micro-frontends) are not required to worry about duplicated assets.

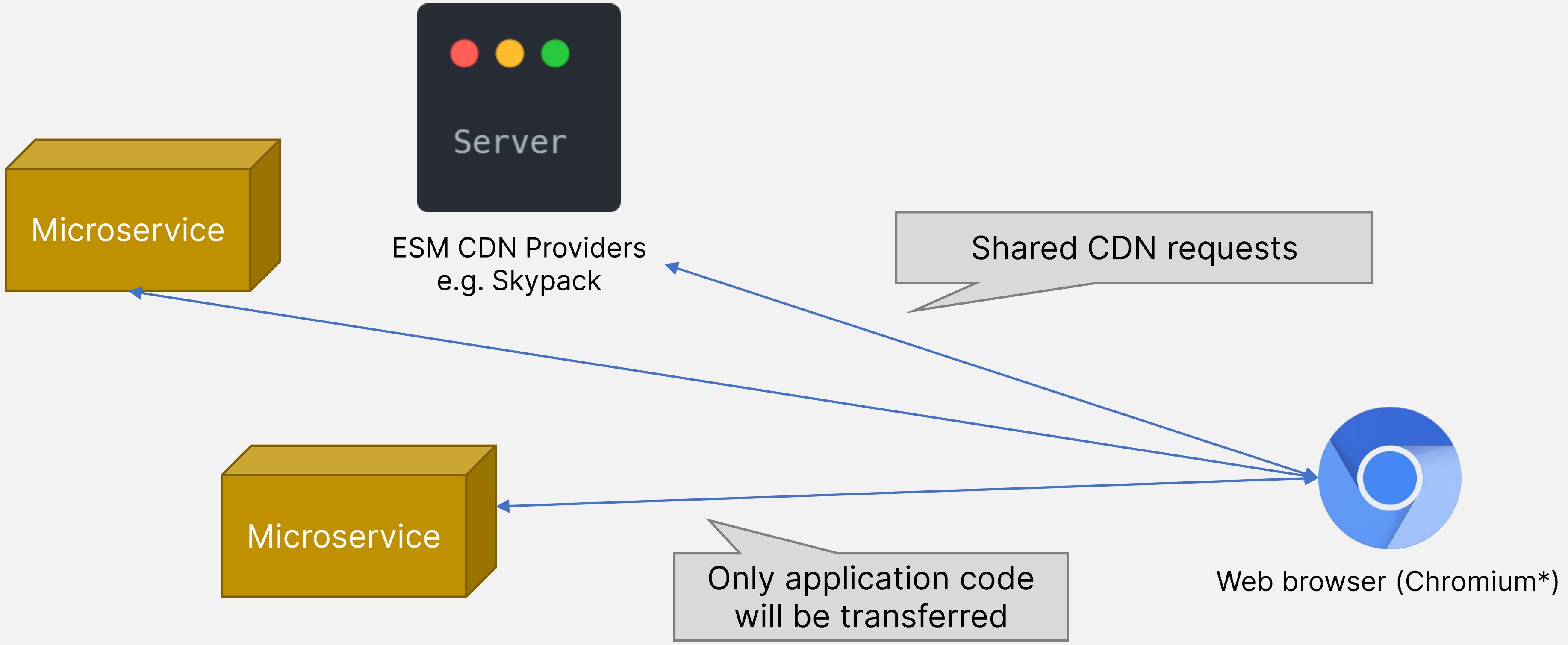


Figure 8. Micro-Frontend situation

Future workarounds to ‘MOVE THE WEB FORWARD’

Web ecosystems are all progressive but need to think backward compatibility.

- Implementation of HTTP/2 and this requires HTTPS by default.
- Still mobile network is not that reliable.
- Client may disable JavaScript and still there are Internet Explorer users.

Thank you

Font: Inter.

- GitHub: @Seia-Soto
- Twitter: @Seia-Soto, @fluentAroma