

Leveraging

# Code Splitting in React Apps

 @gInnrys



**Witaj Warszawa!**



# Glenn Reyes

 @glnnrys



# Glenn Reyes

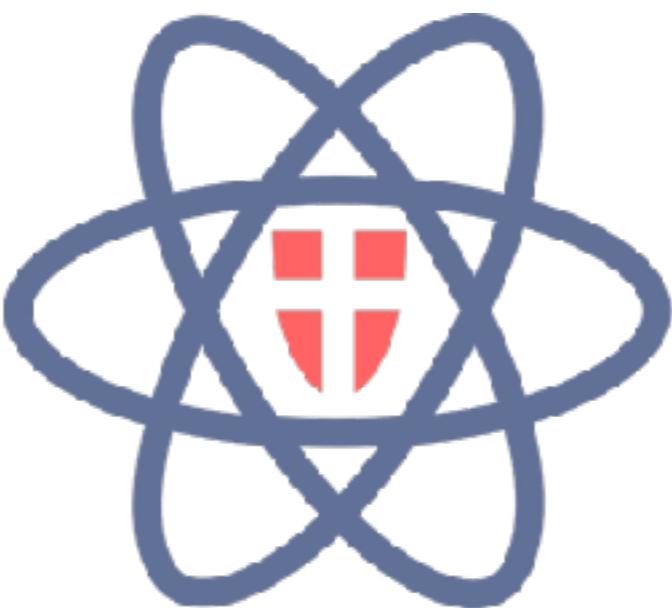


@glnnrys



*Source: Stefan Steinbauer at unsplash.com*

React



Vienna

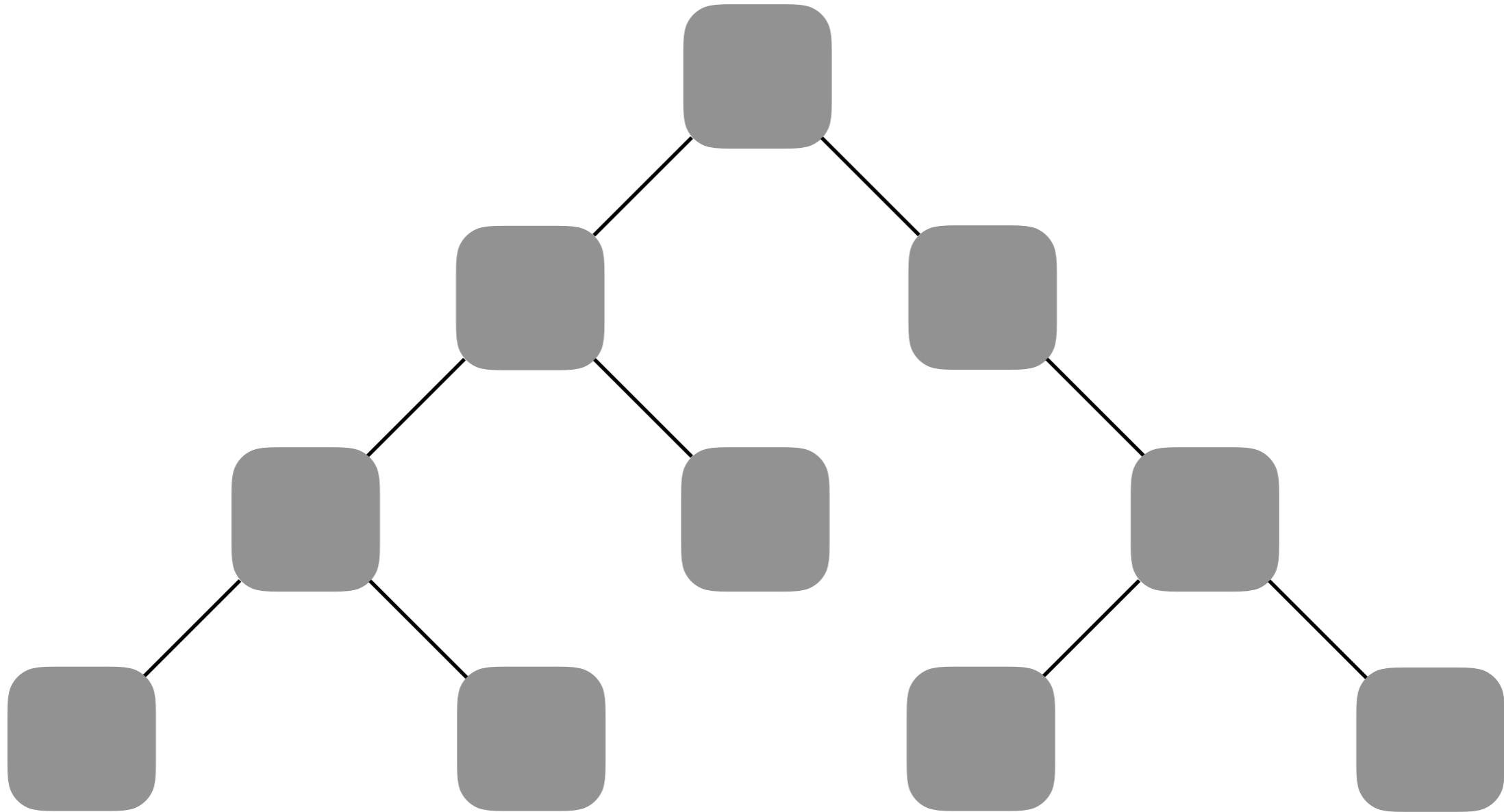
# **Code splitting**

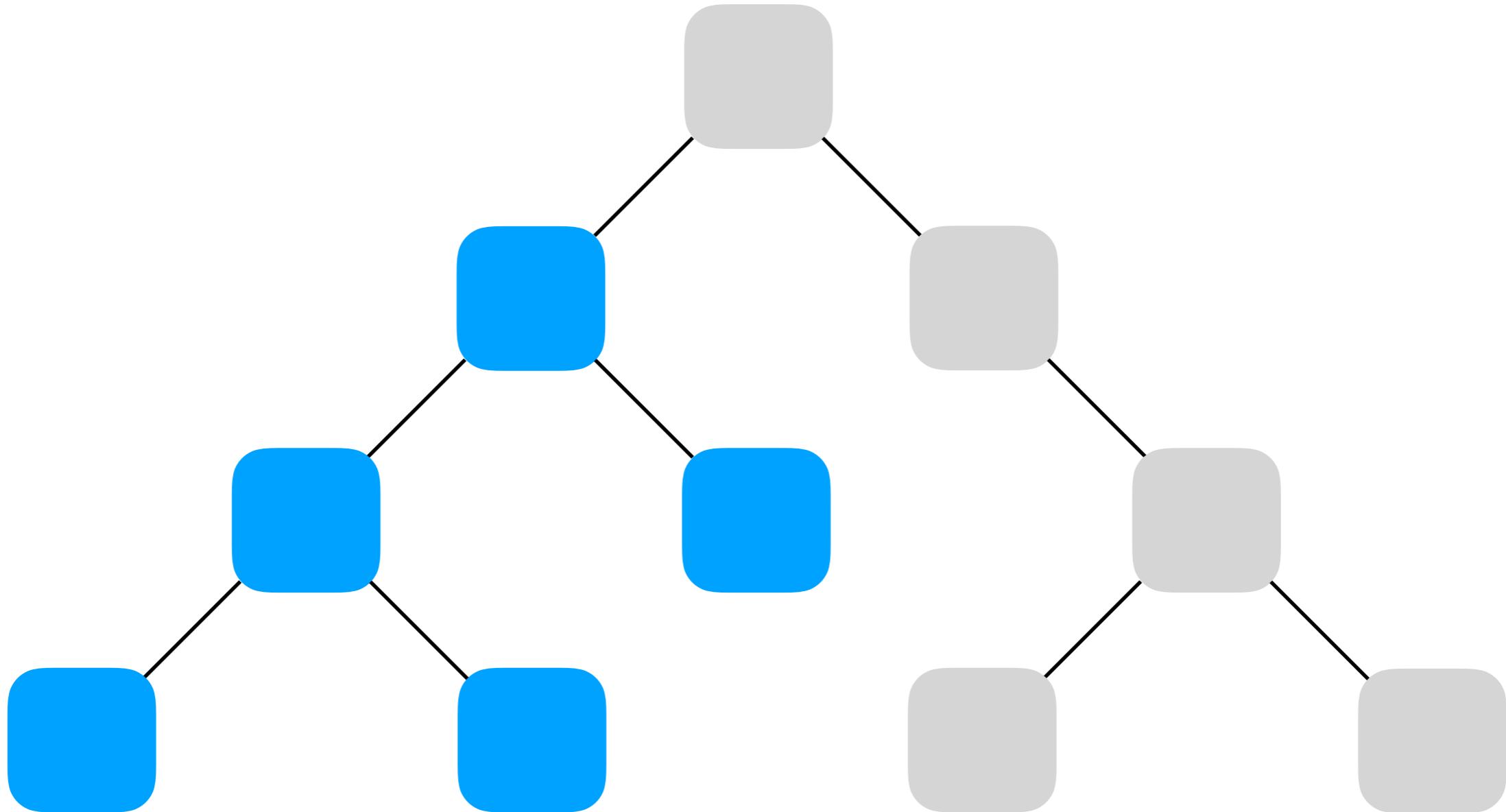
# **what is Code Splitting?**



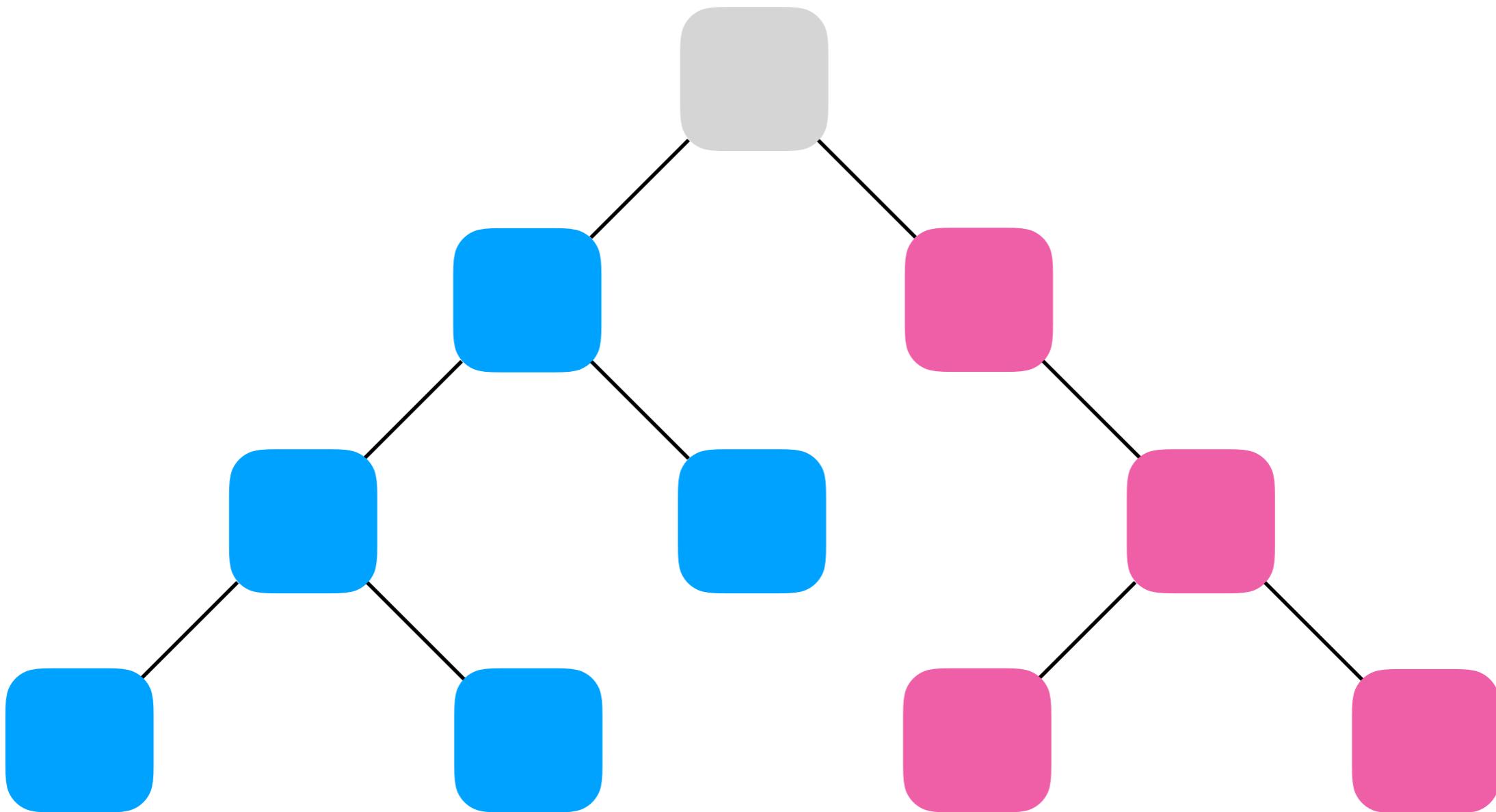
**“Splits your code into various bundles which can then be loaded on demand.”**

– webpack.js.org





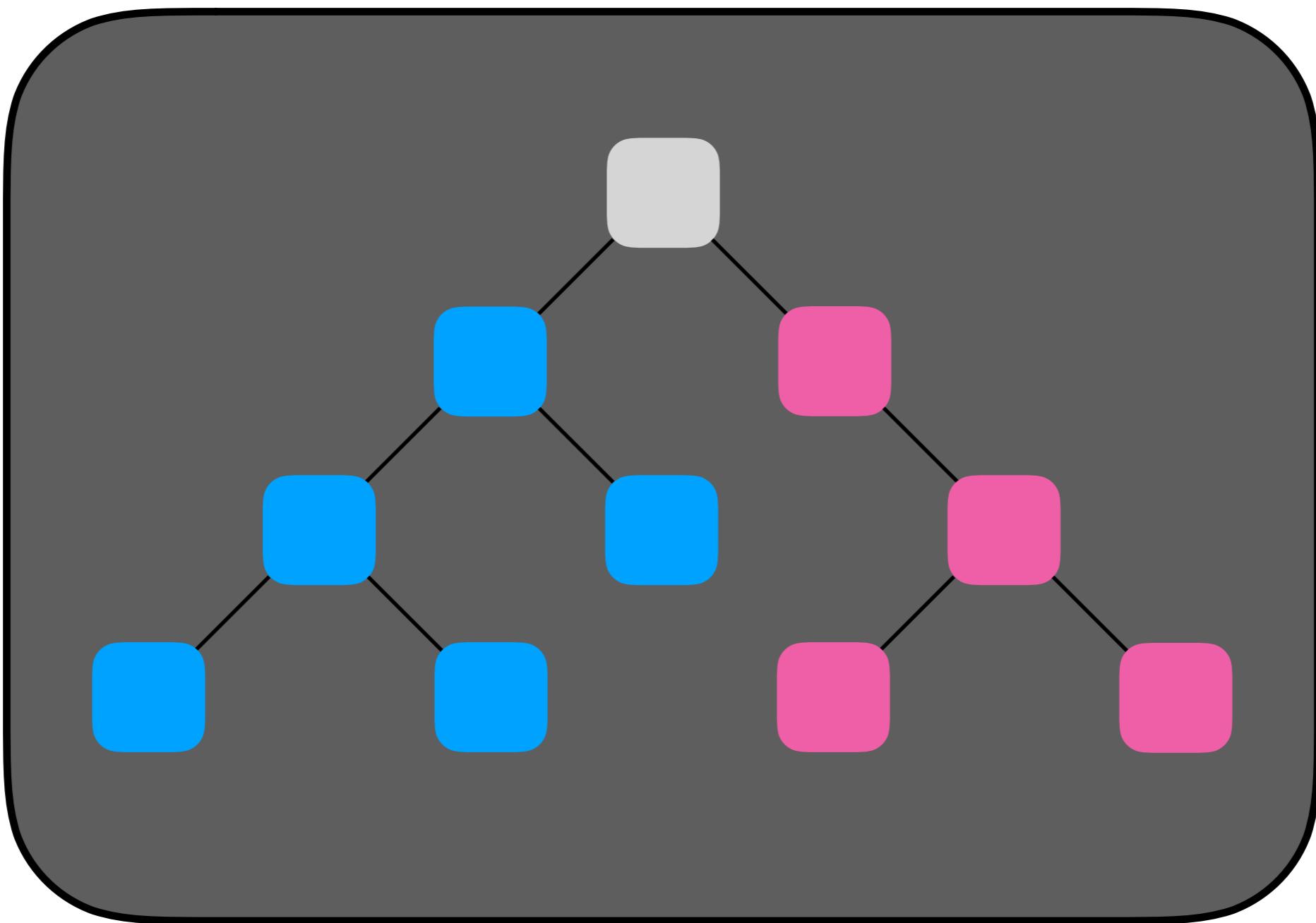
/home

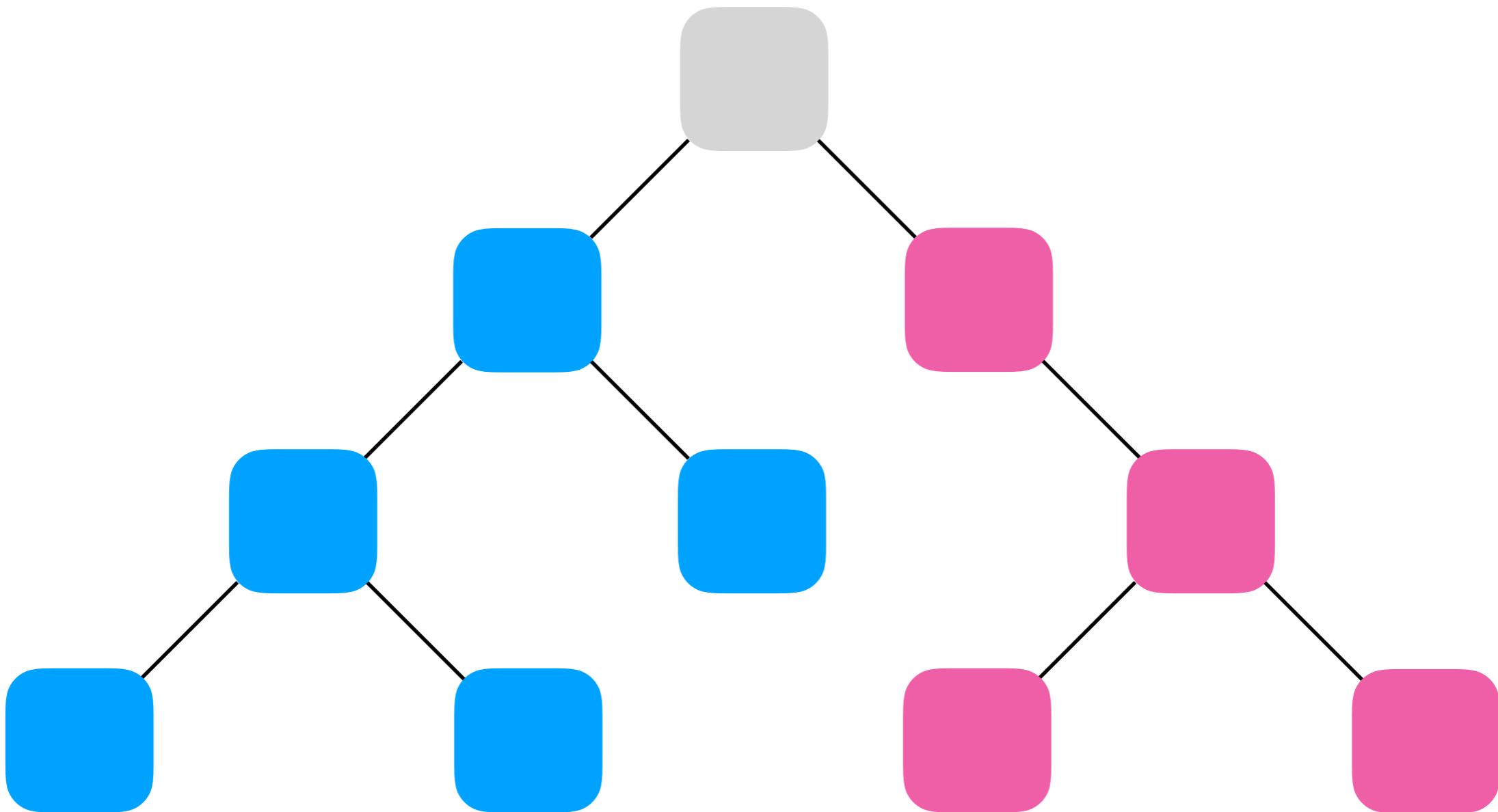


/home

/profile

# bundle.js



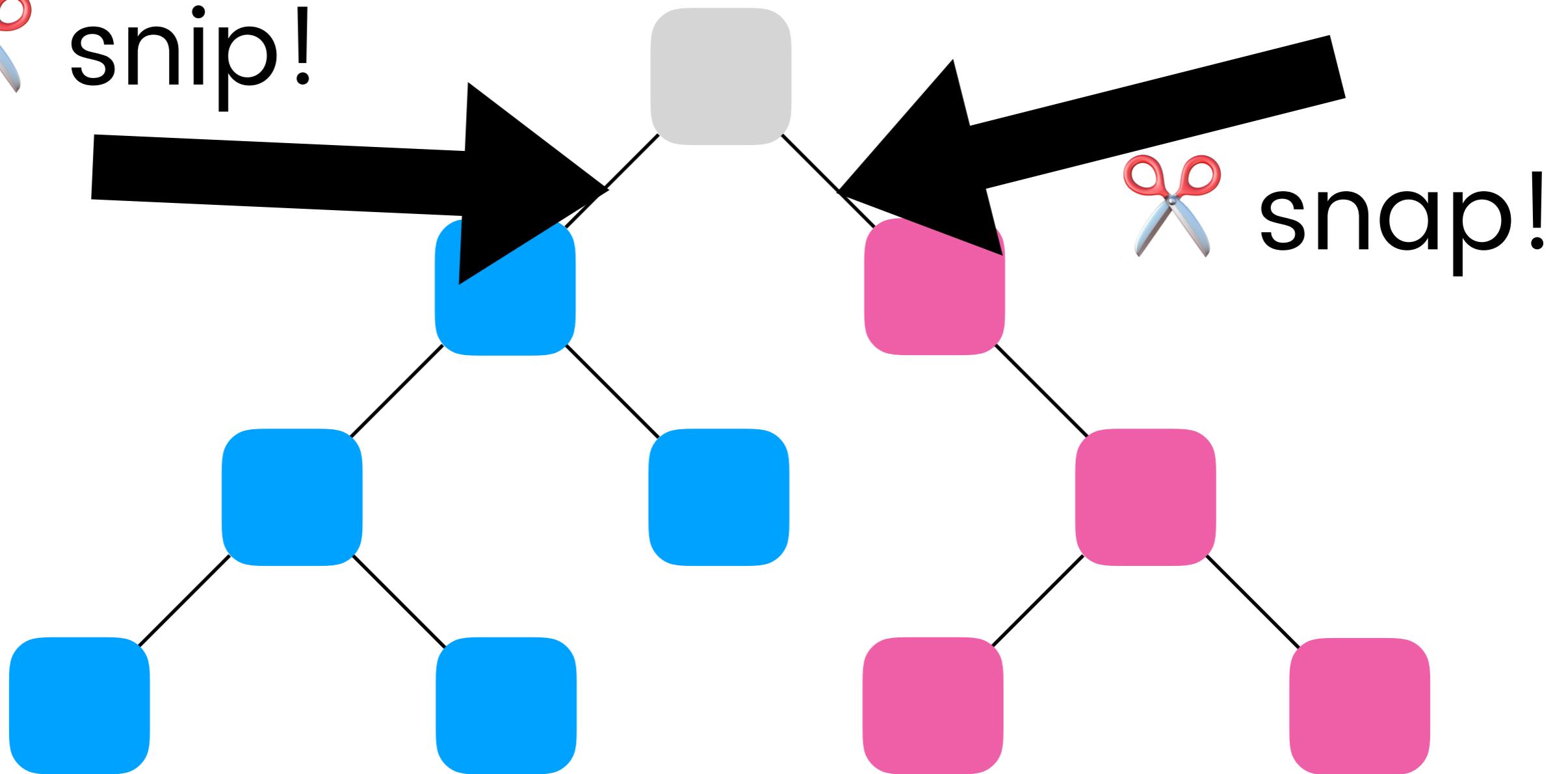


/home

/profile



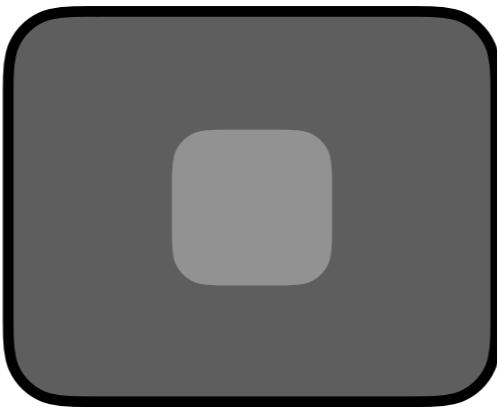
snip!



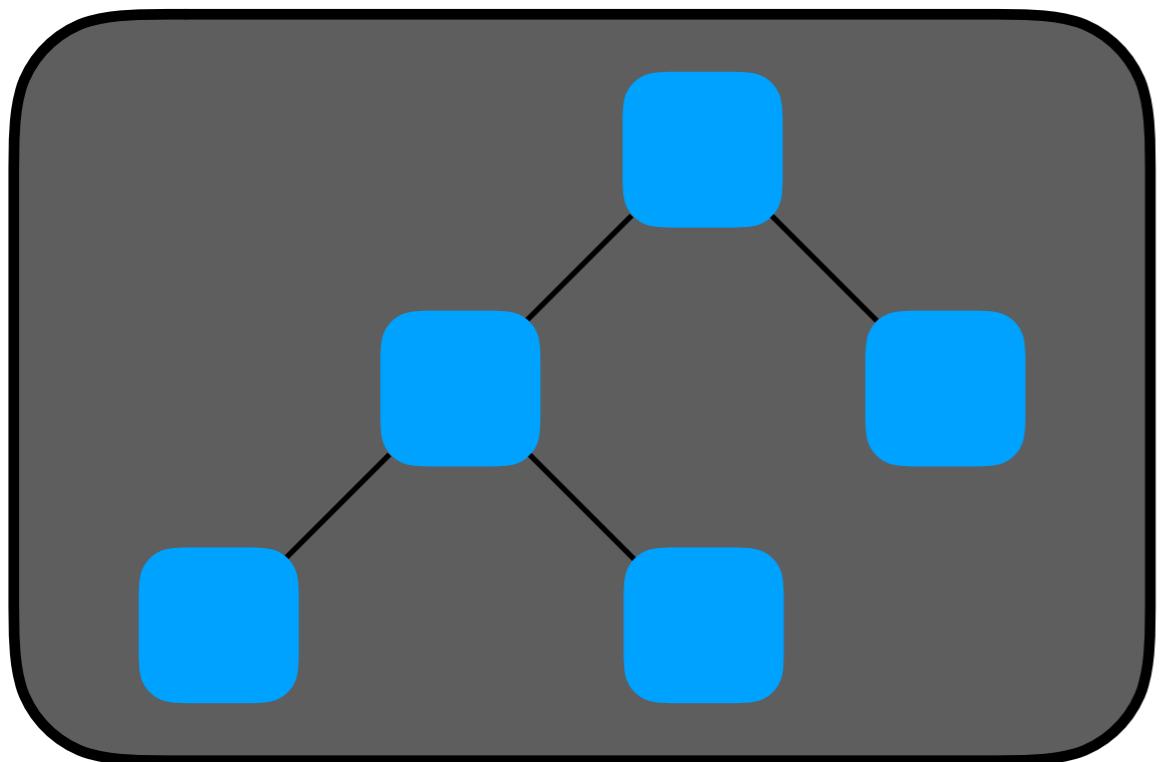
/home

/profile

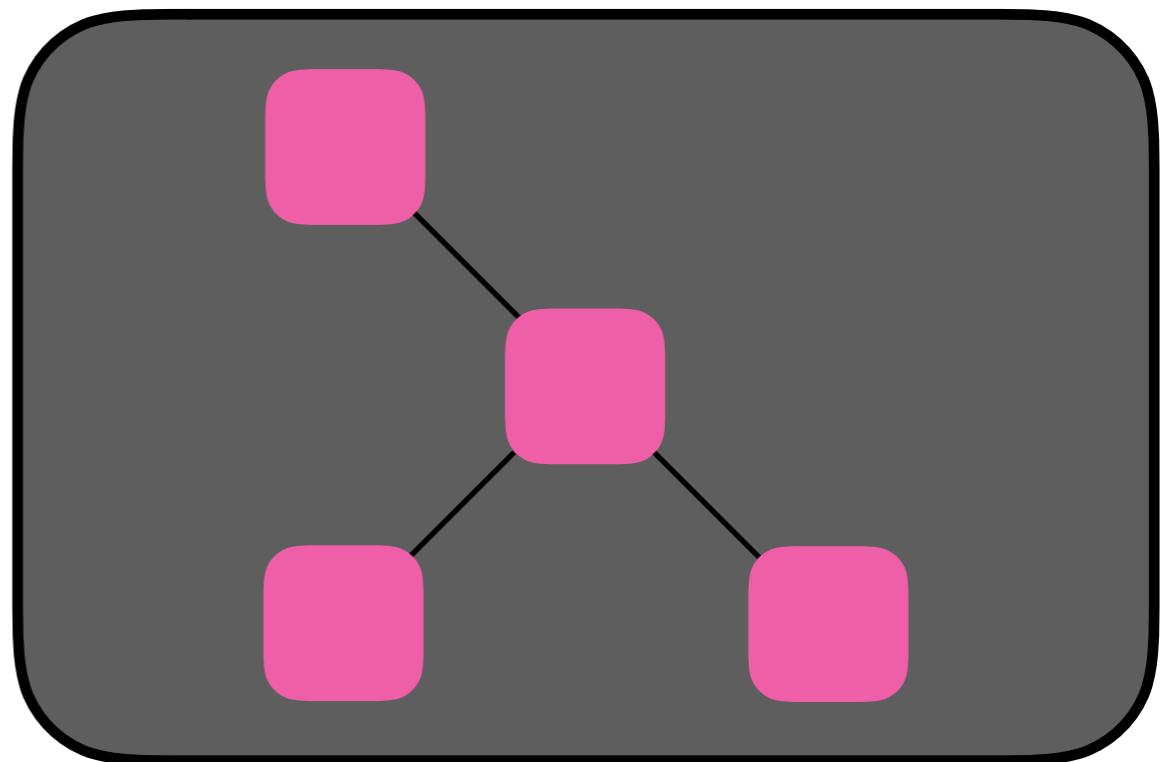
# bundle.js

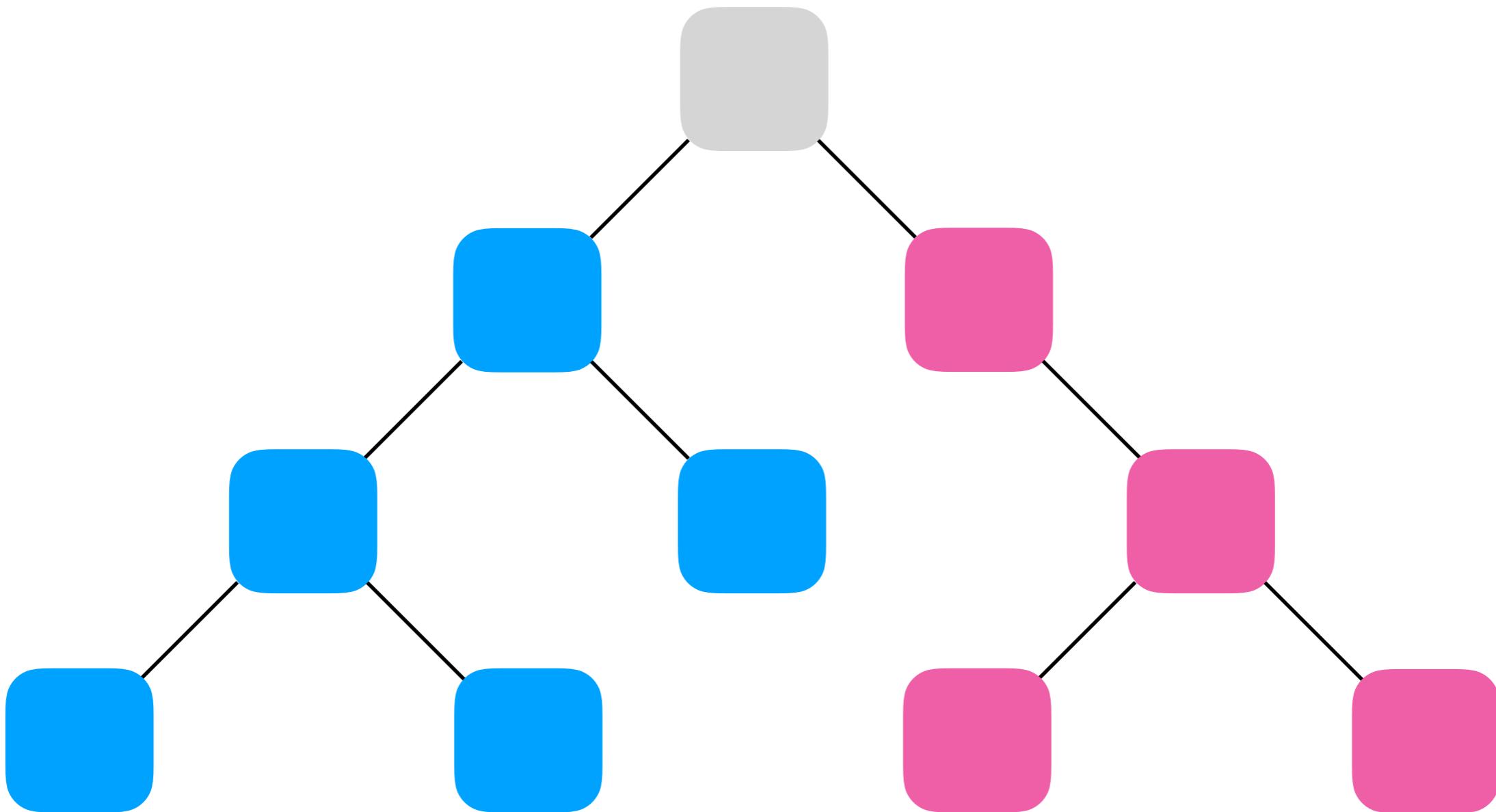


## home.js



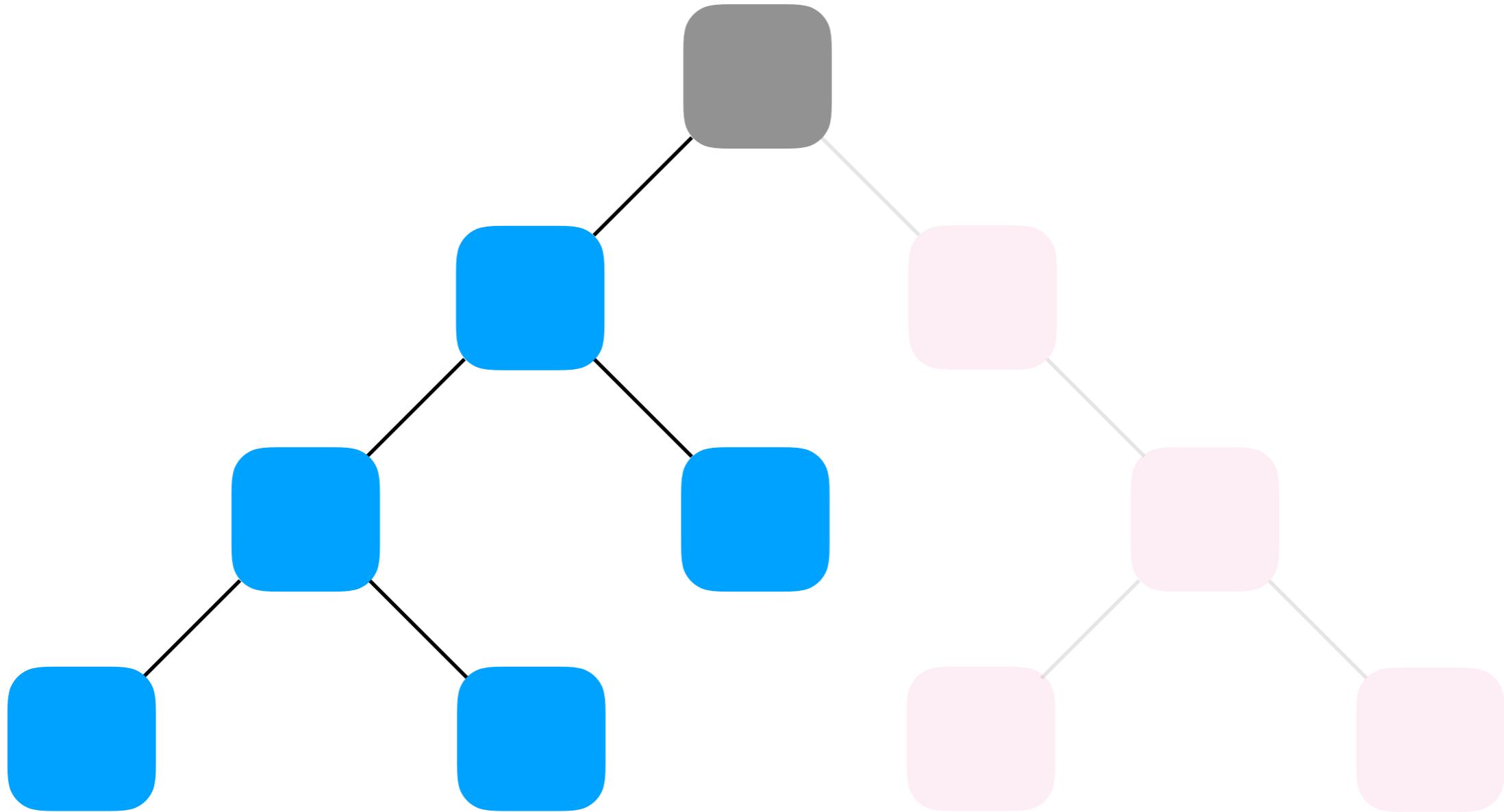
## profile.js





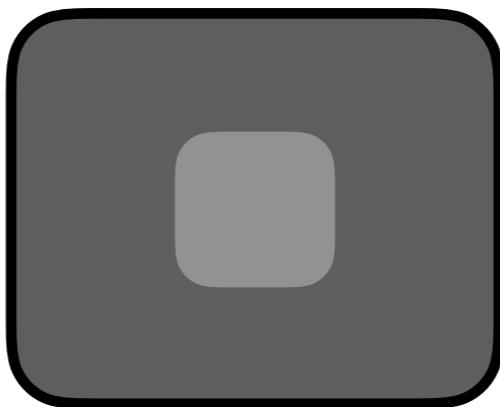
/home

/profile

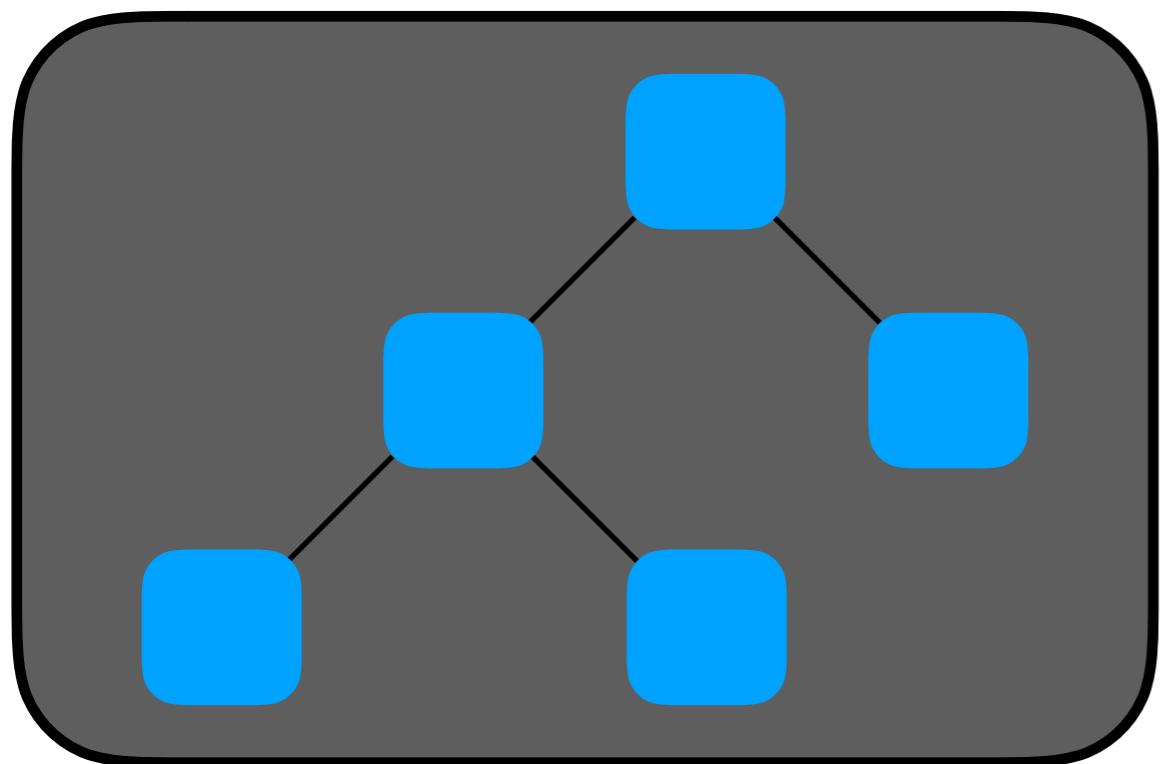


/home

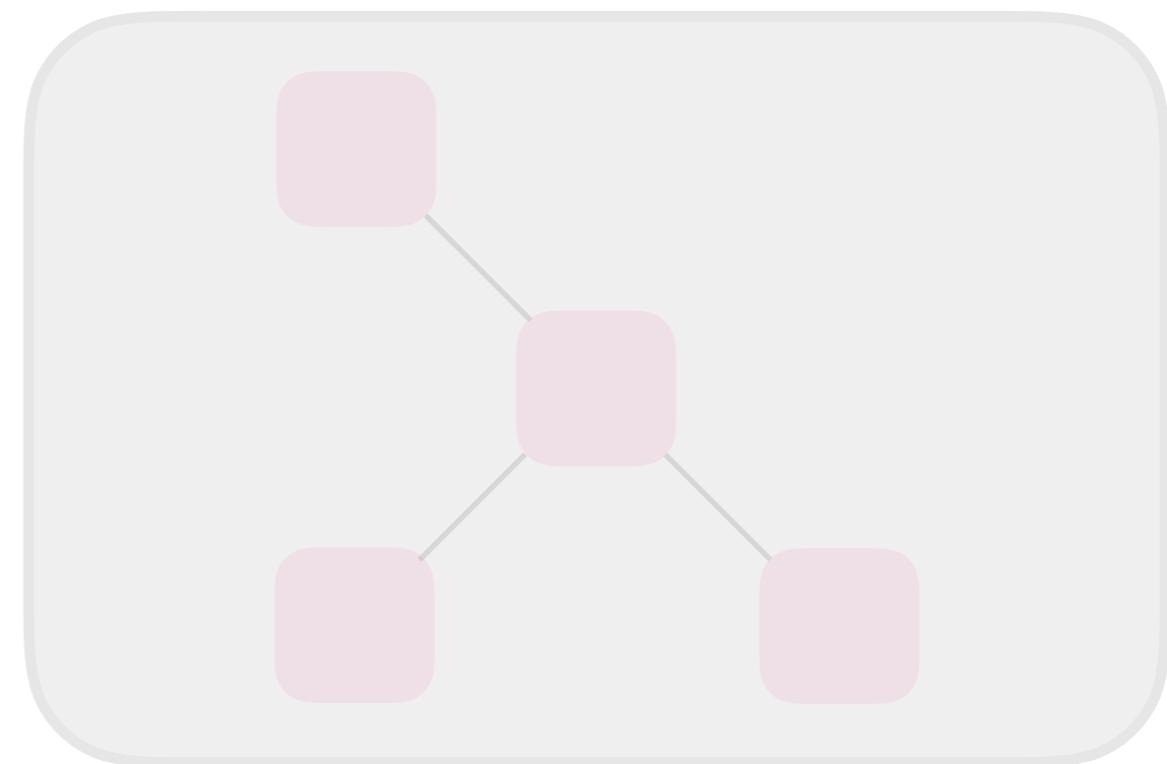
# bundle.js



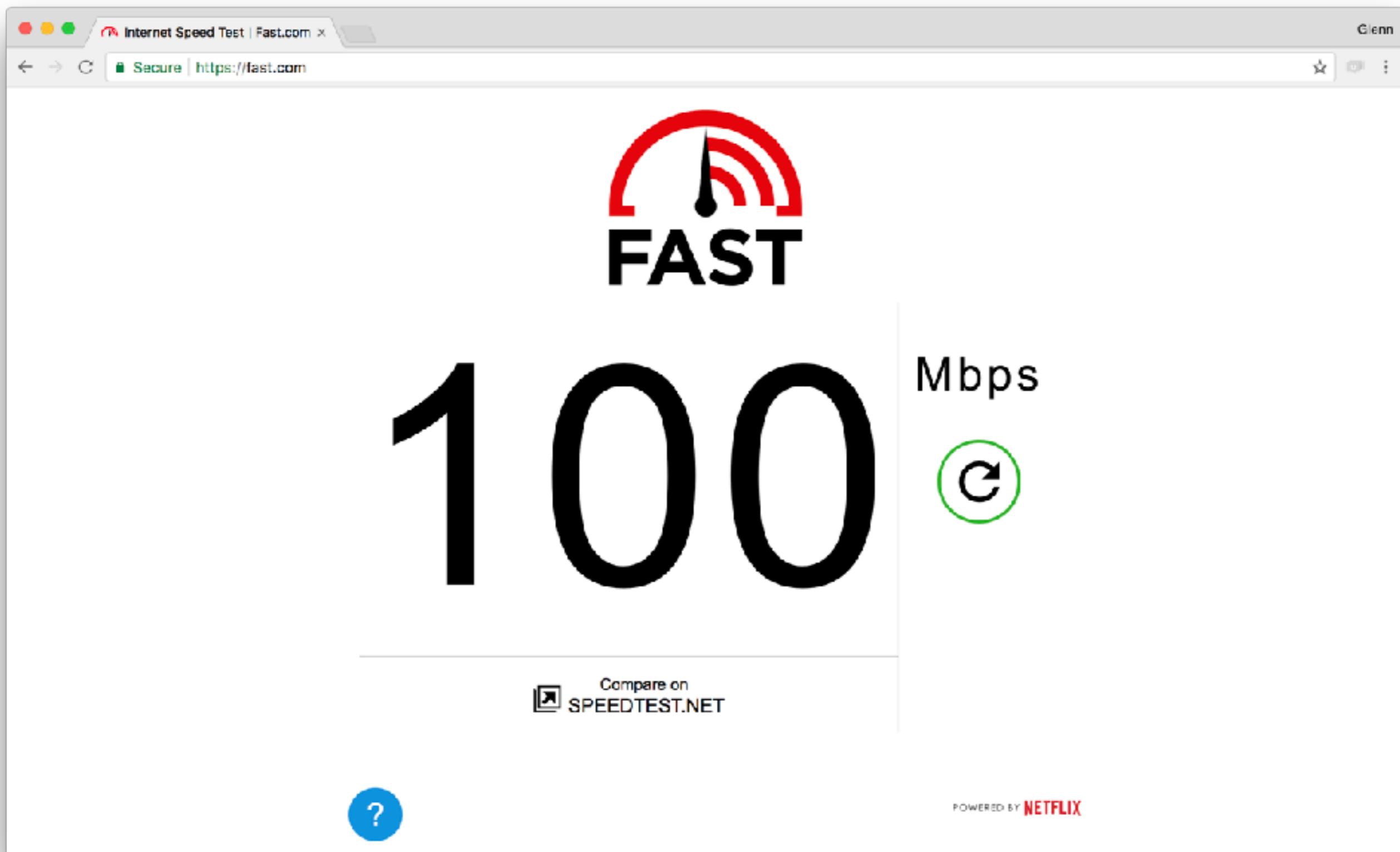
## home.js



## profile.js



**why should I  
care?**









**why should I  
care?**

it's simple

# **How does Code Splitting increase performance?**

# Caching

Load code  
**asynchronously**

**Load code  
on demand**

**Load code  
as needed**

# **How does it work?**

# glenreyes/code-splitting-example

Code Splitting example

localhost:3000

Home Profile

Home page content

Hover here to load async content

Elements

```
<!DOCTYPE html>
...<html lang="en"> == $0
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="shortcut icon" href="/favicon.ico">
    <title>Code Splitting example</title>
    ><style type="text/css">...</style>
    <script type="text/javascript" charset="utf-8" async src="/static/js/home.chunk.js">
    </script>
    ><style type="text/css">...</style>
  </head>
  <body>
    ><div id="root">...</div>
    <script type="text/javascript" src="/static/js/bundle.js"></script>
  </body>
</html>
```

html

Styles Event Listeners DOM Breakpoints Properties

Filter :hov .cls +

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="shortcut icon" href="/favicon.ico">
    <title>Code Splitting example</title>
    ▶<style type="text/css">...</style>
    <script type="text/javascript" charset="utf-8" async src="/static/js/home.chunk.js">
    </script>
    ▶<style type="text/css">...</style>
  </head>
  <body>
    ▶<div id="root">...</div>
    <script type="text/javascript" src="/static/js/bundle.js"></script>
  </body>
</html>
```

---

# bundle + home

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="shortcut icon" href="/favicon.ico">
    <title>Code Splitting example</title>
    ▶<style type="text/css">...</style>
    <script type="text/javascript" charset="utf-8" async src="/static/js/home.chunk.js">
      </script>
    ▶<style type="text/css">...</style>
  </head>
  <body>
    ▶<div id="root">...</div>
    <script type="text/javascript" src="/static/js/bundle.js"></script>
  </body>
</html>
```

Code Splitting example

localhost:3000

Home Profile

Home page content

Hover here to load async content

Elements

```
<!DOCTYPE html>
...<html lang="en"> == $0
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="shortcut icon" href="/favicon.ico">
    <title>Code Splitting example</title>
  ><style type="text/css">...</style>
  <script type="text/javascript" charset="utf-8" async src="/static/js/home.chunk.js">
  </script>
  ><style type="text/css">...</style>
</head>
  <body>
    ><div id="root">...</div>
    <script type="text/javascript" src="/static/js/bundle.js"></script>
  </body>
</html>
```

html

Styles Event Listeners DOM Breakpoints Properties

Filter :hov .cls +

The screenshot shows a browser window titled "Code Splitting example" at the URL "localhost:3000/profile". The main content area displays a dark header with a logo and the title, followed by a pink section labeled "Profile page content". A tooltip at the bottom left says "Hover here to load async content". The browser's developer tools are open, specifically the "Elements" tab, which shows the rendered HTML structure. The DOM tree includes the following code:

```
<!DOCTYPE html>
...<html lang="en"> == $0
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="shortcut icon" href="/favicon.ico">
    <title>Code Splitting example</title>
    ><style type="text/css">...</style>
    <script type="text/javascript" charset="utf-8" async src="/static/js/home.chunk.js">
    </script>
    ><style type="text/css">...</style>
    <script type="text/javascript" charset="utf-8" async src="/static/js/profile.chunk.js">
    </script>
    ><style type="text/css">...</style>
  </head>
  <body>
    ><div id="root">...</div>
    <script type="text/javascript" src="/static/js/bundle.js"></script>
  </body>
</html>
```

The "Styles" tab in the DevTools is selected, showing a search bar with the filter ":hov .cls +".

# bundle + home + profile

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="shortcut icon" href="/favicon.ico">
    <title>Code Splitting example</title>
    ▶<style type="text/css">...</style>
    <script type="text/javascript" charset="utf-8" async src="/static/js/home.chunk.js">
    </script>
    ▶<style type="text/css">...</style>
    <script type="text/javascript" charset="utf-8" async src="/static/js/profile.chunk.js">
    </script>
    ▶<style type="text/css">...</style>
  </head>
  <body>
    ▶<div id="root">...</div>
    <script type="text/javascript" src="/static/js/bundle.js"></script>
  </body>
</html>
```



# Before

```
SystemJS.import('./profile.js');

System.import('./profile.js');

require('./profile.js');

require.ensure(['./profile.js'], cb);

MyCustomLoader('./profile.js');
```



# Dynamic import

`import();`

# Function-like syntactic form

```
import('./chart');
```

# tc39/proposal-dynamic-import

The screenshot shows the GitHub repository page for `tc39/proposal-dynamic-import`. The page includes the repository name, a star count of 583, a fork count of 16, and links for Watch, Unstar, and Fork. It also shows 3 issues and 0 pull requests. The README.md file is displayed, featuring a large heading `import()` and a description of the proposal. The repository is described as being in stage 3 of the TC39 process and having 4 contributors.

tc39 / proposal-dynamic-import

Code Issues 3 Pull requests 0 Projects 0 Insights

Branch: master proposal-dynamic-import / README.md

aendrew Fixes broken HTML integration link in README 23ct15a on 26 Jun

4 contributors

138 lines (96 sloc) | 12 KB

## import()

This repository contains a proposal for adding a "function-like" `import()` module loading syntactic form to JavaScript. It is currently in stage 3 of the TC39 process. Previously it was discussed with the module-loading community in [whatwg/loader#149](#).

You can view the in-progress spec draft and take part in the discussions on the [issue tracker](#).

### Motivation and use cases

The existing syntactic forms for importing modules are static declarations. They accept a string literal as the module specifier, and introduce bindings into the local scope via a pre-runtime "linking" process. This is a great design for the 90% case, and supports important use cases such as static analysis, bundling tools, and tree shaking.

The logo for Babel, consisting of the word "BABEL" written in a bold, yellow, hand-drawn style font.

**syntax-dynamic-plugin**



# webpack

<https://webpack.js.org/guides/code-splitting>

<https://survivejs.com/webpack/building/code-splitting>

# **Sync vs async imports**

# Synchronous import

```
import Chart from './Chart';  
  
console.log(Chart);
```



**ES6 IMPORTS**

**ME**

**COMMONJS IMPORTS**

# Asynchronous import

```
import('./Chart').then(module => {
  console.log(module.default);
});
```

# Async functions and import();

```
async () => {  
  const module = await import('./Chart');  
  console.log(module.default);  
}
```



# **How with React?**

```
1 import React, { Component } from 'react';
2
3 class App extends Component {
4     state = {
5         AsyncComponent: () => <div>Nothing loaded</div>,
6     };
7
8     async componentDidMount() {
9         const module = await import('./AsyncComponent');
10        const AsyncComponent = module.default;
11
12        this.setState({ AsyncComponent });
13    }
14
15    render() {
16        const { AsyncComponent } = this.state;
17
18        return <AsyncComponent />;
19    }
20}
21
22 export default App;
23
```

```
1 import React, { Component } from 'react';
2
3 class App extends Component {
4     state = {
5         AsyncComponent: () => <div>Nothing loaded</div>,
6     };
7
8     async componentDidMount() {
9         const module = await import('./AsyncComponent');
10        const AsyncComponent = module.default;
11
12        this.setState({ AsyncComponent });
13    }
14
15    render() {
16        const { AsyncComponent } = this.state;
17
18        return <AsyncComponent />;
19    }
20}
21
22 export default App;
23
```

```
1 import React, { Component } from 'react';
2
3 class App extends Component {
4     state = {
5         AsyncComponent: () => <div>Nothing loaded</div>,
6     };
7
8     async componentDidMount() {
9         const module = await import('./AsyncComponent');
10        const AsyncComponent = module.default;
11
12        this.setState({ AsyncComponent });
13    }
14
15    render() {
16        const { AsyncComponent } = this.state;
17
18        return <AsyncComponent />;
19    }
20}
21
22 export default App;
23
```

```
1 import React, { Component } from 'react';
2
3 class App extends Component {
4     state = {
5         AsyncComponent: () => <div>Nothing loaded</div>,
6     };
7
8     async componentDidMount() {
9         const module = await import('./AsyncComponent');
10        const AsyncComponent = module.default;
11
12        this.setState({ AsyncComponent });
13    }
14
15    render() {
16        const { AsyncComponent } = this.state;
17
18        return <AsyncComponent />;
19    }
20}
21
22 export default App;
23
```

## App.js

```
3  class App extends Component {
4      state = {
5          components: [],
6      };
7
8      async componentDidMount() {
9          const components = await Promise.all([
10              import('./Home').then(m => m.default),
11              import('./Profile').then(m => m.default),
12          ]);
13
14          this.setState({ components });
15      }
16
17      render() {
18          return (
19              <div>
20                  {this.state.components.map((AsyncComponent, index) => (
21                      <AsyncComponent key={index} />
22                  )));
23              </div>
24          );
25      }
}
```



ReactionGIFs

# **react-loadable**

<https://github.com/thejameskyle/react-loadable>

# sync

```
<Route exact path="/" component={Home} />
```

# async

```
<Route exact path="/" component={Loadable({  
  loader: () => import('./Home'),  
  loading: Loading,  
})} />
```

**Only 3 more lines  
of code**

# **create-react-app**

<https://github.com/facebookincubator/create-react-app>

# **What about SSR?**

# Next.js

<https://github.com/zeit/next.js>



Guillermo Rauch

@rauchg

Following



It works for zeit.co with thousands of components and hundreds of code-splitting entry points.

5:07 PM - 8 Aug 2017 from San Francisco, CA

2 Retweets 12 Likes



1



2



12



# **Where to code split?**

#1

**Split app and vendor**

Applause from A. Sharif, James Gillmore, and 321 others



Andrey Okonetchnikov

Freelance UI-Developer, co-creator of @colorsnapper, @ReactVienna co-organizer, co-owner of @...

Jul 11, 2015 · 5 min read

# Long-term caching of static assets with Webpack

Webpack is a great way to package all your static resources such as JavaScript, CSS or even images, but to effectively use generated assets in production one should leverage long-term caching. Documentation on this topic is scattered across different resources and it is not quite easy to get it right. The aim of this article is to guide front-end developers through the complete setup.

Kent C. Dodds

## tl;dr

To enable long-term caching of static resources produced by webpack:

1. Use `[chunkhash]` to add a content-dependent cache-buster to each file.

Applause from Sean T. Larkin, Dan Abramov, and 339 others



Tim Sebastian [Follow](#)

Something, something javascript at Atlassian // twitter @schnibl // github.com/timse // bitbucket.org...

May 20 · 6 min read

# Predictable long term caching with Webpack

Getting long-term caching right with Webpack is a problem that never really got a final answer.

There is, however, this open [issue on Github](#):

- it has 162 comments
- is soon to have its 2 birthday
- a lot of suggestions that often make things worse
- and probably quite some google hits.

Arguably there is an easy answer, you could use the [RecordsPlugin](#) (given the documentation on this is a little sparse. But that requires you to keep track of

#2

**Split at route level**

# webpack chunk names

```
import(/* webpackChunkName: "home" */ './Home'),  
  
import(/* webpackChunkName: "profile" */ './Profile'),
```



/home.360c0ce7.chunk.js

/profile.8b26492e.chunk.js

#3

**Split at  
component level**

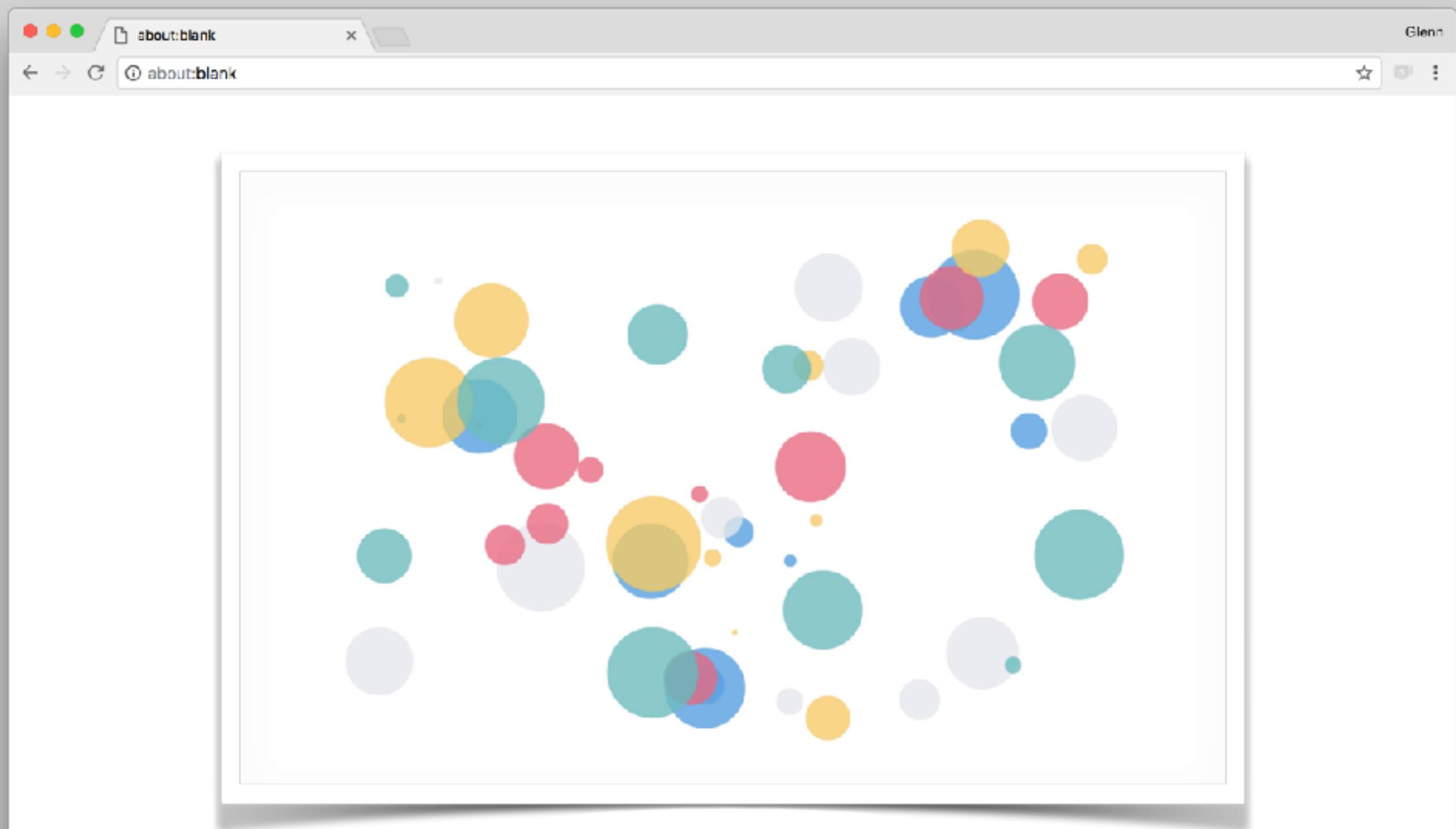


Chart.js

**Predict user intent  
and prefetch**

# Where to code split

 **App and vendor**

 **Routes**

 **Components**

# Good reads

- Straightforward code splitting with React and Webpack  
<https://hackernoon.com/straightforward-code-splitting-with-react-and-webpack>
- ES proposal: import() – dynamically importing ES modules  
<http://2ality.com/2017/01/import-operator.html>
- Tree Shaking vs Code Splitting: A Real-World Benchmark  
<https://medium.com/outreach-engineering/tree-shaking-vs-code-splitting-a-real-world-benchmark-bbbf36245db3>
- Progressive Web Apps with React.js: Part 2 – Page Load Performance  
<https://medium.com/@addyosmani/progressive-web-apps-with-react-js-part-2-page-load-performance-33b932d97cf2>
- The PRPL Pattern  
<https://developers.google.com/web/fundamentals/performance/prpl-pattern/>

T-MOB

ARENA

# Recap

- ✨ import();
- ✨ Load code as needed
- ✨ Up to 5x faster page load
- ✨ Split vendor, at route & component
- ✨ create-react-app & Next.js



glenreyes



@glnnrys