CSS -in- JS

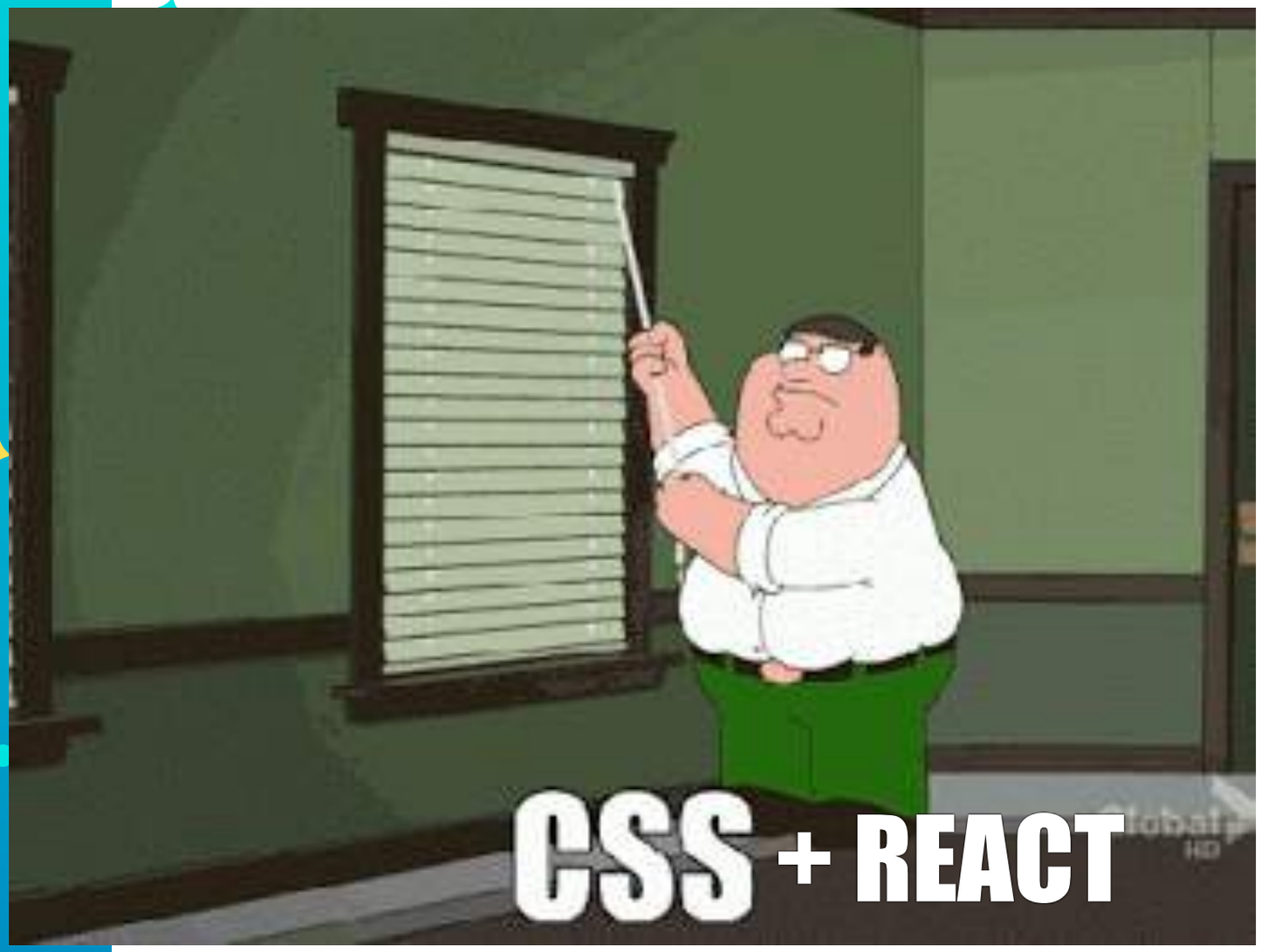
* JSS:

A new workflow that *just might actually* work.

NDEV28 // AUGUST 29, 2018

10 years (and counting) of CSS-induced swearing.

@lesleyannchard
@lipscripts


CSS + REACT

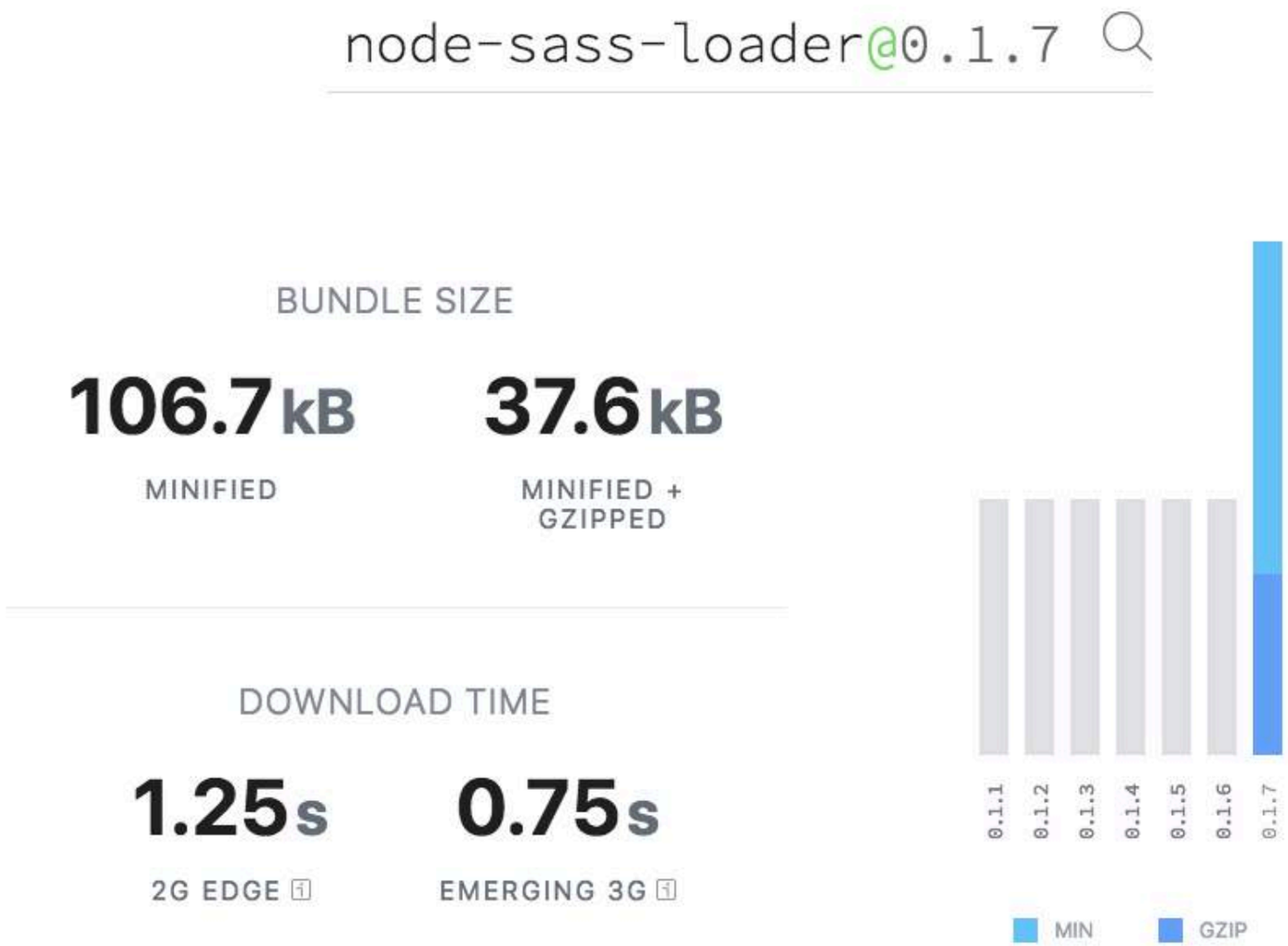
Past Environment:

1. Preprocessing
2. File organization
3. Naming conventions
4. Postprocessing

1. Preprocessing

Compile SASS or LESS to usable CSS.

- More verbose than JS
- Learning curve for a new syntax
- Compiler or loader dependency (webpack)
- Compiled to a single stylesheet



Credit: http://bundlephobia.com

2. File Organization

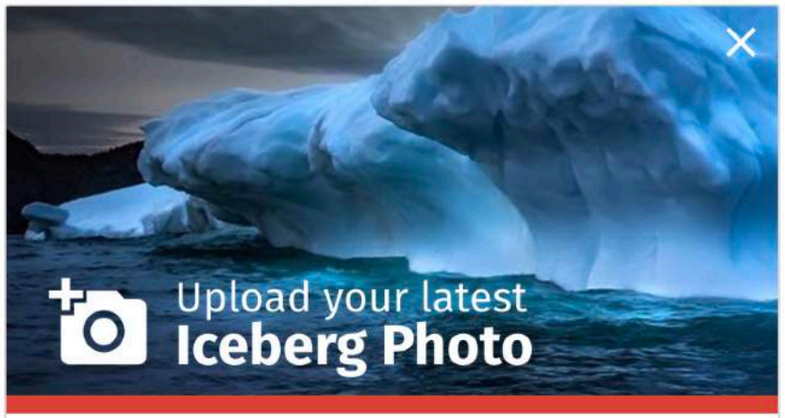
1:1 style to component file ratio with same naming and directory structure

- Styles and components stored separately
- Edge cases

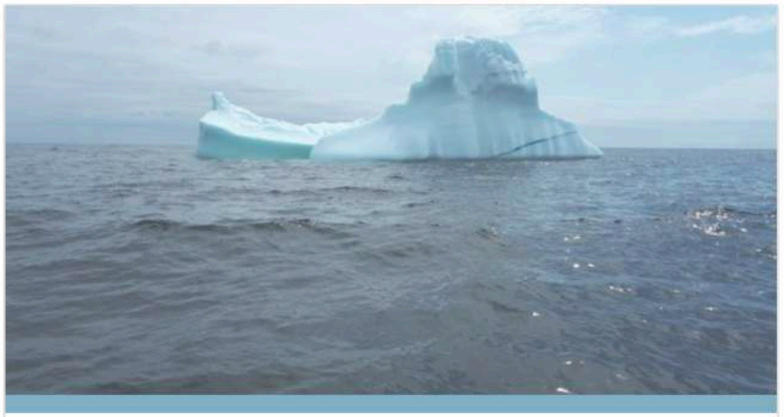
St. John's

PHOTOS ICEBERGS

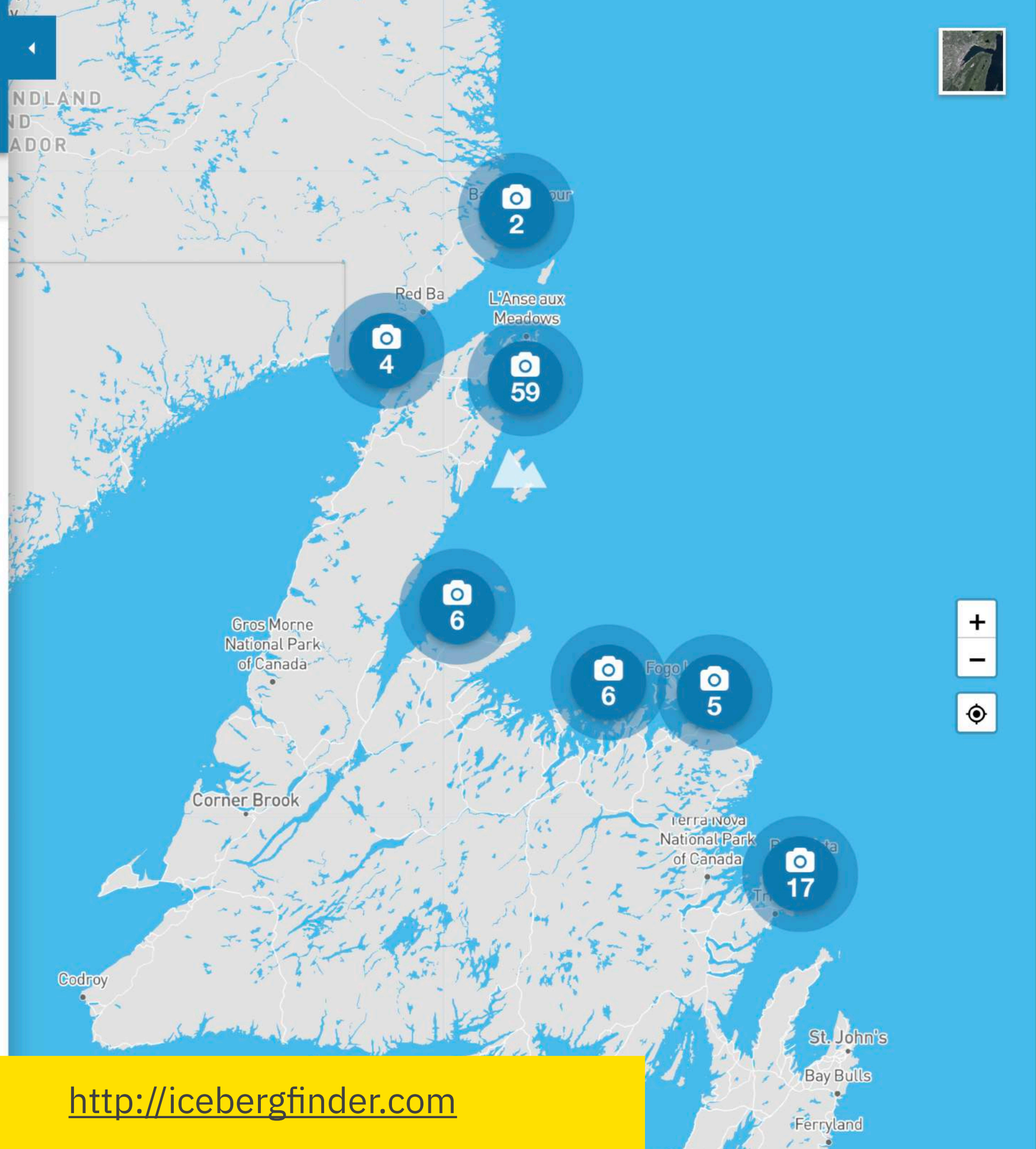
All Time



UPLOAD PHOTO DISMISS



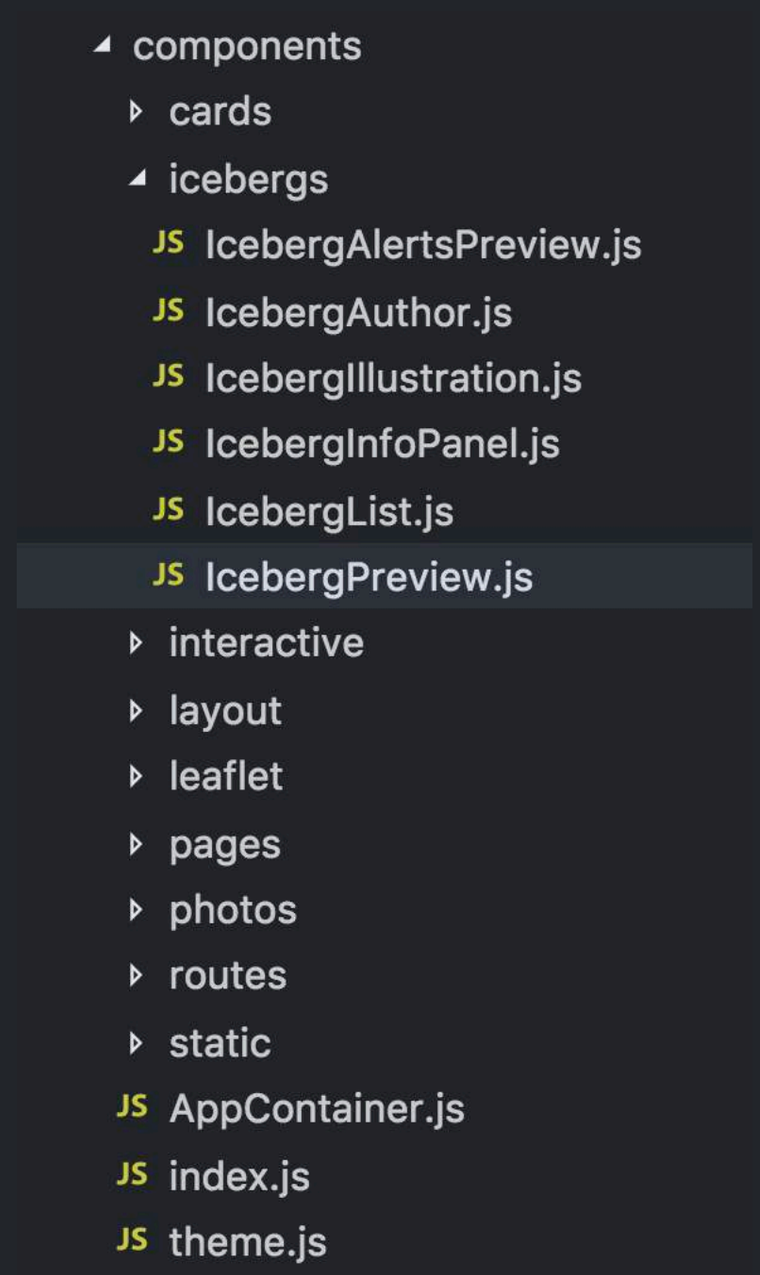
Jean Jacques Préaux 20 days ago



http://icebergfinder.com

- components
  - cards
    - CardList.js
    - Preview.js
    - Reminder.js
  - icebergs
  - interactive
  - layout
  - leaflet
  - pages
  - photos
  - routes
  - static
  - AppContainer.js
  - index.js
  - theme.js

- assets
  - fonts
  - img
  - scss
    - base
    - components
      - cards
        - __variables.scss
        - _card-list.scss
        - _preview.scss
        - _reminder.scss
      - interactive
      - layout
      - pages
      - static
    - theme
    - _import.scss
    - style.scss



_preview.scss

```scss
@include respond(max, md) {
    .preview--iceberg .preview__media {
        min-height: calc(10rem);
    }
}

@include respond(min, md) {
    .preview--photo .preview__media {
        min-height: calc(12rem);
    }
}
```

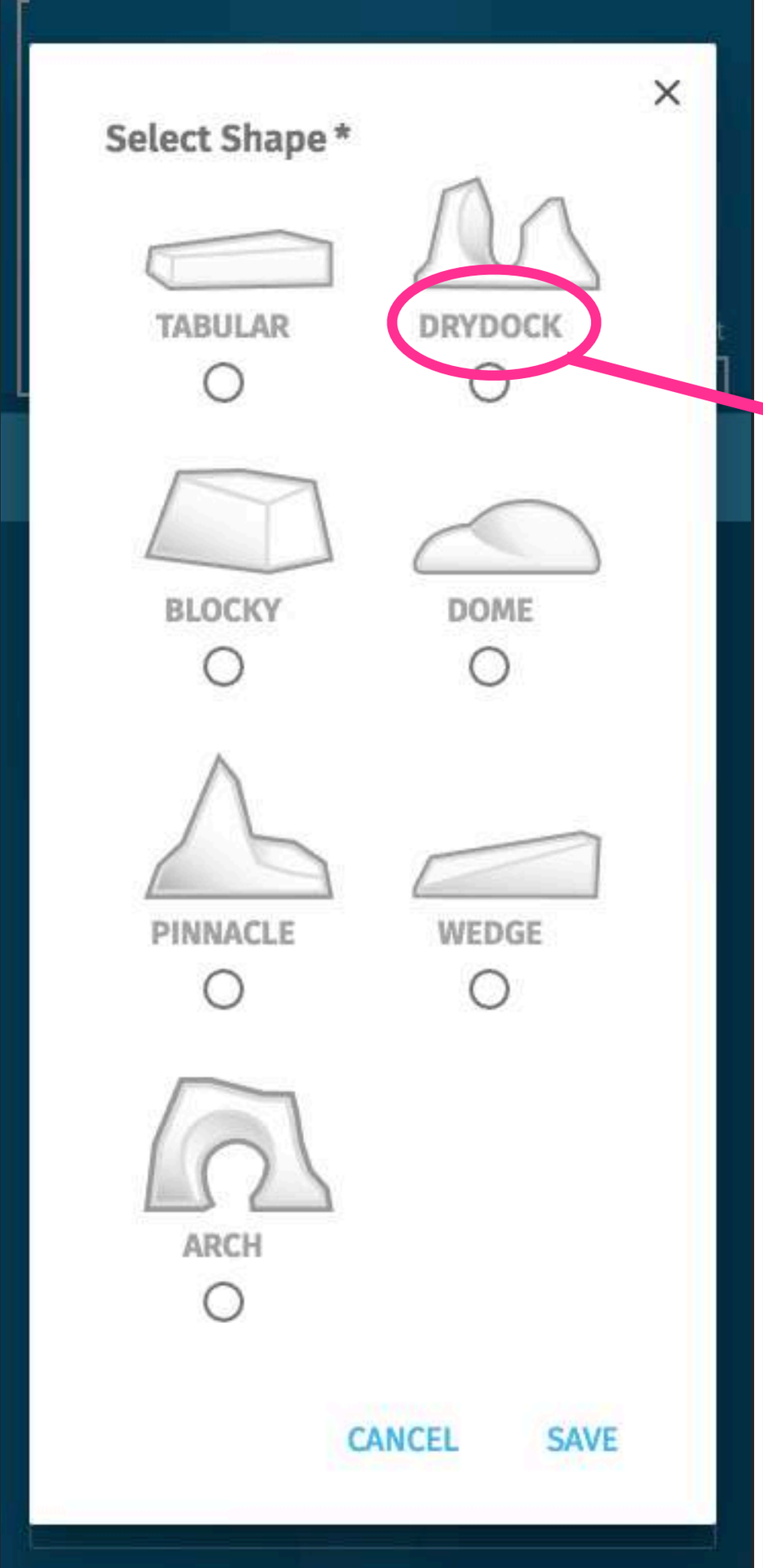
3. Naming Conventions

Local scope by extremely limited use of tag and child selectors (ex. BEM)

- Verbose and time-consuming to write
- Longer class names will negatively affect performance
- Standard is difficult to keep when collaborating

Basic BEM for React

- Apply class names to all styled components and DOM elements.
- Example: `Person.js`




```
.radio-thumbs__control-label__label {
    order: -1;
    position: relative;
    margin-bottom: -0.5em;
    color: swatch(text, 0.5);
    font-weight: font-weight(bold);

    &.active {
        color: swatch(text);
    }
}
```

4. Postprocessing

Automates functions by CSS pattern detection (ex. PostCSS)

- Compiler and loader dependencies (webpack)

postcss@7.0.2

BUNDLE SIZE

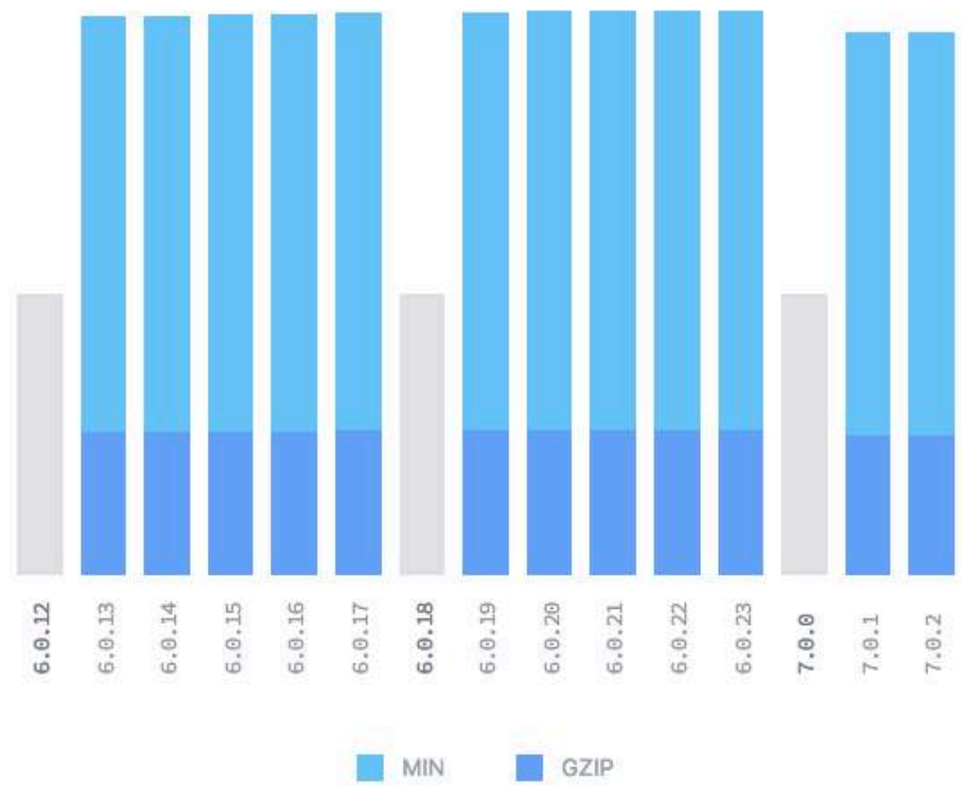
75.9 kB MINIFIED

19.5 kB MINIFIED + GZIPPED

DOWNLOAD TIME

0.65 s 2G EDGE

390 ms EMERGING 3G



Credit: http://bundlephobia.com

CSS-in-JS

Present Day CSS-in-JS

- Fresh takes on an old concept
- Component-centred styles
- Locally scoped CSS
- Better theming

styled components

glamorous

JSS

... and many more.

Why Choose JSS?

- Simple, familiar syntax (JSON)
- Small bundle size (~6kb minified, gzipped)
- Active, passionate community
- **Used by Material UI React**

JSS Crash Course

http://cssinjs.org/

How Does it Work?

1. Install via npm package
2. Import package and set up any additional presets
    - Camelcase keys, nested JSON selectors, default preset
3. Write styles
4. Create a stylesheet and pass in styles
5. Attach styles to DOM or render them server-side

```
import jss from 'jss'
import preset from 'jss-preset-default'

jss.setup(preset())

// Create your style.
const style = {
  myButton: {
    color: 'green'
  }
}

// Compile styles, apply plugins.
const sheet = jss.createStyleSheet(style)

// If you want to render on the client, insert it into DOM.
sheet.attach()

// If you want to render server-side, get the css text.
sheet.toString()
```

JSS with vanilla JavaScript

What’s Compiled?

- JSS transforms JSON to CSS styles.
- Class names are compiled into non-deterministic strings to prevent collisions.
- You can create as many stylesheets as you want or need.
- Stylesheets can be injected at specific locations using insertion points.

Stylesheets injected via HTML comment

index.html (snippet)

```
<head>
  <title>JSS</title>
  <!-- custom-insertion-point -->
</head>
```

index.js (snippet)

```
jss.setup({insertionPoint: 'custom-insertion-point'})
```

Stylesheets injected via DOM element

index.html (snippet)

```
<head>
  <title>JSS in body</title>
</head>
<body>
  <div id="insertion-point">
    This might be any DOM node of your choice which can serve as an insertion point.
  </div>
</body>
```

index.js (snippet)

```
jss.setup({
  insertionPoint: document.getElementById('insertion-point')
})
```

JSS Object → **Compiled CSS**

```
1  export default {
2    myButton: {
3      color: 'green',
4      backgroundColor: '#fff',
5    },
6  };
7
```

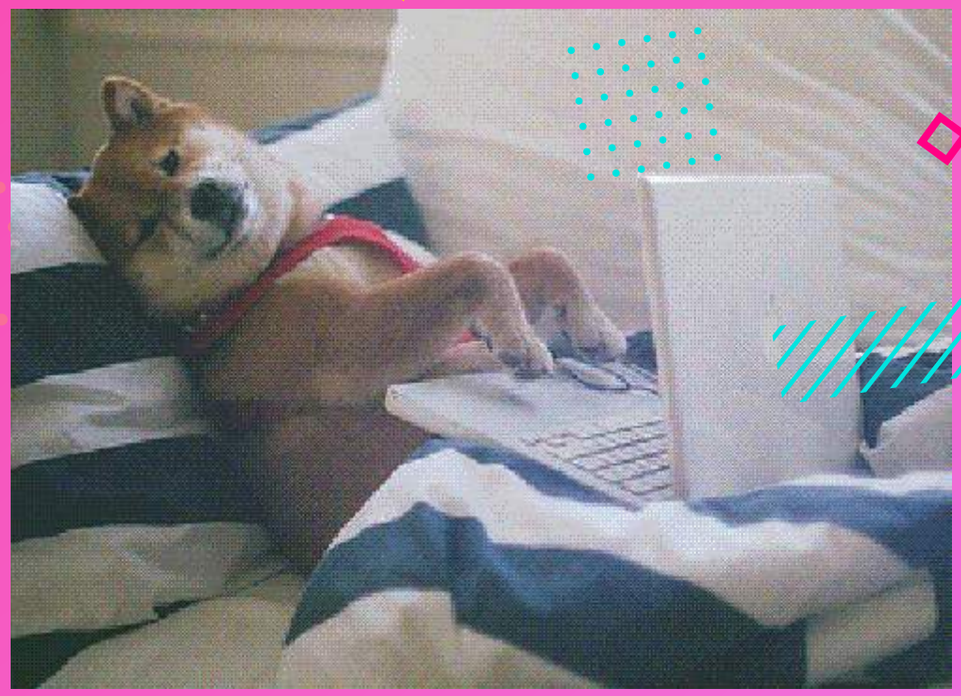
```
1  .myButton-0-1-26 {
2    color: green;
3    background-color: #fff;
4  }
```

Via JSS Playground: http://cssinjs.org/repl

How Does it Work with React?

- At its core, JSS is framework agnostic.
- There's an npm for that:
  `react-jss`

LET'S *code* STUFF



https://bit.ly/2ooiPiR

JSS in the Wild

Material UI

- Component libraries keep your UI and app functionality separate
- Reusable across any React app
- MUI is a model example
- Uses JSS for all theming and styling
- Extends and exposes the JSS API

Material UI Advantages

- ✅ Follows Google’s Material Design patterns
- ✅ Highly interactive components at your fingertips
- ✅ Fully tested
- ✅ Actively developed, maintained
- ✅ Easy to theme
- ✅ Completely customizable with CSS or JSS

Bluedrop’s UI Library

JSS in the Wild

Library Specs

- Built on top of Material UI
- Uses JSS for theming and custom styles
- Contains Material UI abstractions and custom components
- Deployed as a private NPM package
- Imported wherever we need it

Why we love it:

- 💖 Stored and maintained separately to our apps core functions
- 💖 Developed in parallel to core functionality
- 💖 Reusable and portable
- 💖 Recognizable brand across all of our products
- 💖 Standards are easily kept
- 💖 **100% test coverage** 🎉

CSS *vs.* JSS

Past Environment

1. Preprocessing

CSS

(ex. SASS/LESS)

- Verbose
- Learning curve for new syntax
- Compiler or loader dependency
- Compiled to a single stylesheet

vs.

JSS

- Written in JavaScript
- Simple JSON API
- Main dependency (`jss`, `react-jss`, and plugins)
- Multiple stylesheets, only loaded when needed

jss@9.8.7

BUNDLE SIZE

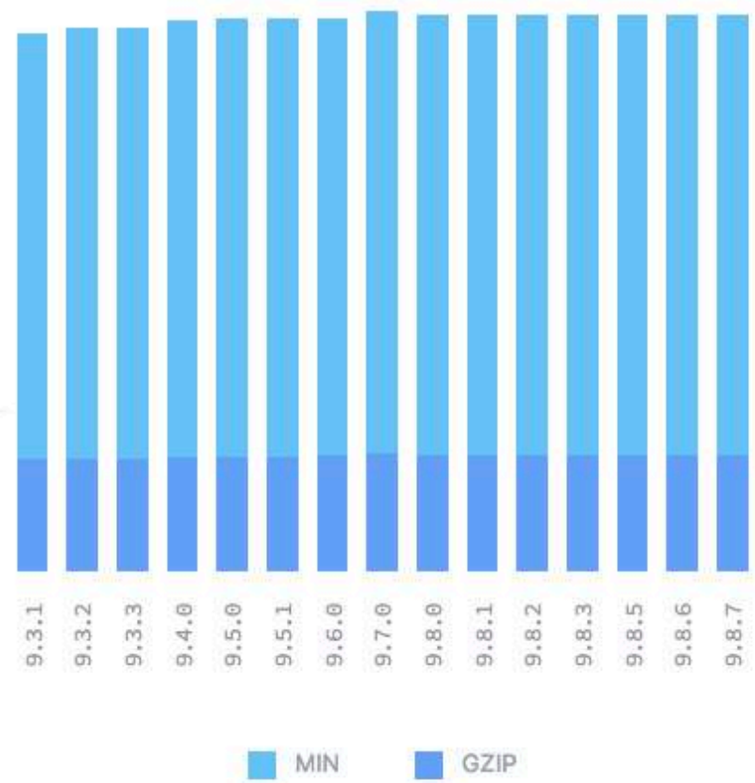
32.6 kB MINIFIED

6.9 kB MINIFIED + GZIPPED

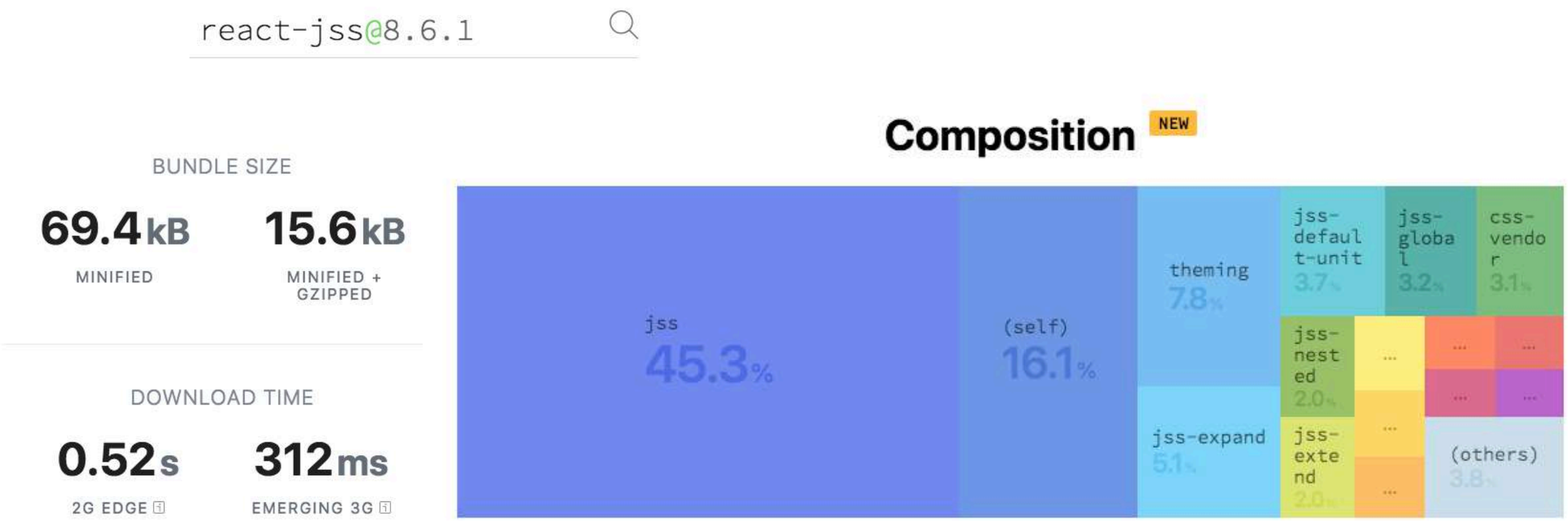
DOWNLOAD TIME

228 ms 2G EDGE

137 ms EMERGING 3G



Credit: http://bundlephobia.com



Credit: http://bundlephobia.com

Past Environment

1. Preprocessing

CSS

(ex. SASS/LESS)

- Verbose
- 😭 LOSER
- Compiler or loader dependency
- Compiled to a single stylesheet

VS.

JSS

- Written in JavaScript
- Simple JSON API
- 🏆 WINNER
- Composability (`jss`, `react-jss`, and plugins)
- Multiple stylesheets, only loaded when needed

Past Environment

2. File Organization

CSS

(ex. 1:1 component to styles)

- Styles and components stored separately
- Edge cases

vs.

JSS

- Styles and components stored together
- No known edge cases

Past Environment

2. File Organization

CSS

(ex. 1:1 component to styles)

- Styles and components stored separately

😭 LOSER

vs.

JSS

- Styles and components stored together

🏆 WINNER

Past Environment

3.Naming Conventions

CSS

(ex. BEM)

- Verbose and time-consuming to write
- Longer class names will negatively affect performance
- Standard is difficult to keep when collaborating

vs.

JSS

- Simple JSON key names
- Class names by default are minified in production
- Disobeyed standards are easily refactored

Past Environment

3.Naming Conventions

CSS

(ex. BEM)

- Verbose and time-consuming to write
- [illegible] negatively affect performance
- Standard is difficult to keep when collaborating

😭 **LOSER**

vs.

JSS

- Simple JSON key names
- [illegible] production
- Disobeyed standards are easily refactored

🏆 **WINNER**

Past Environment

4.Postprocessors

CSS

(ex. PostCSS)

- Compiler and loader dependencies

vs.

JSS

- Plugin dependencies
- Custom JavaScript functions

Past Environment

4.Postprocessors

CSS

(ex. PostCSS)

- Compiler and loader dependencies

🏆 WINNER

vs.

JSS

- Plugin dependencies

😭 LOSER

Questions?