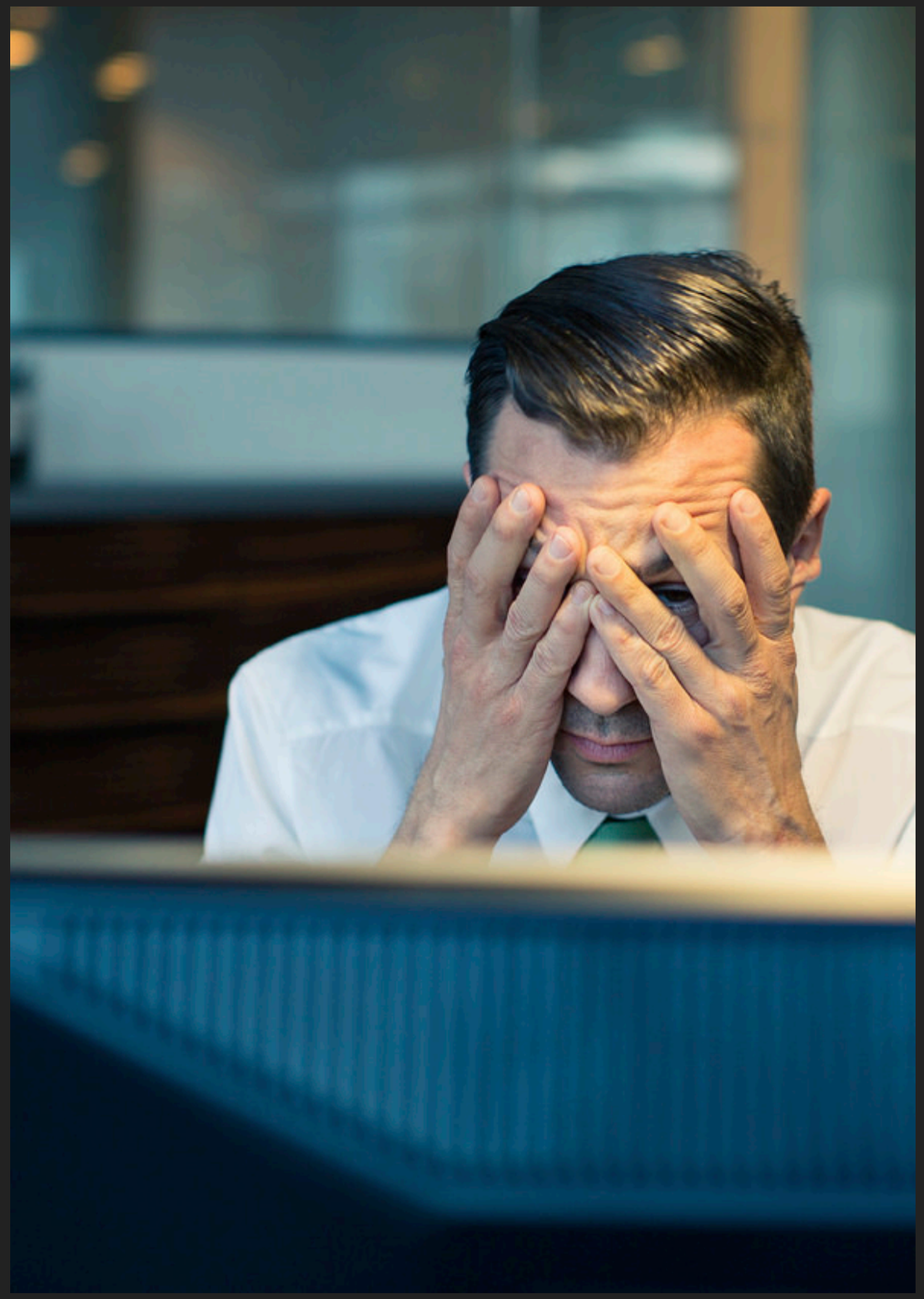
OOCSS, SMACSS, CSS-IN-JS, BEM

CSS SYSTEMS

WHY ADOPT A SYSTEM?

- Creates common expectations for how to solve problems in a project
- Creates a sense of continuity throughout a large project or between related projects
- Creates reusable code which has the potential to reduce development time
- Appeals to the organized types



WHAT SYSTEM DO I PREFER?

Everyone has their own preferences. Mine is the BEM (block-element-modifier) naming convention. I think BEM is flexible, and can be used in existing or legacy applications without requiring a refactor.

When designing a system from the ground up, taking a SMACSS / ITCSS approach is very wise. However BEM can be used in together with those approaches.

Jump ahead to Slide 28 if you are not so much interested in the alternatives to BEM.

WHERE TO START?

- OOCSS
- Developed in 2008 by Nicole Sullivan
- Based on two main concepts:
  - Separation of structure from skin
  - Separation of containers and content
- Sort of a “utility class” concept
- Follows the DRY principle

```
h2 {
  margin: 10px 20px;
  border: 2px solid red;
  padding: 10px;
}

#main h2 {
  margin-top:  -20px;
  border: none;
}

#sidebar h2 {
  border: none;
  padding: 0;
}
```

WHAT DOES OOCSS LOOK LIKE?

OOCCSS uses camel-case, defines objects semantically, then uses utility classes to add the kind of styles which may be repeated many times throughout a code base.

`<div class="leftCol gMail"> ... </div>`

Downsides:

- Ugly
- Indeterminate specificity when combining classes
- Makes heavy use of abbreviations (not necessary in 2017)

SCALABLE AND MODULAR ARCHITECTURE FOR CSS

SMACSS

- Increase the semantic value of a section of html and content
- Decrease the expectation of a specific html structure
- At the very core of SMACSS is categorization. By categorizing CSS rules, we begin to see patterns and can define better practices around each of these patterns.

5 GENERAL SMACSS CATEGORIES OF CSS RULES

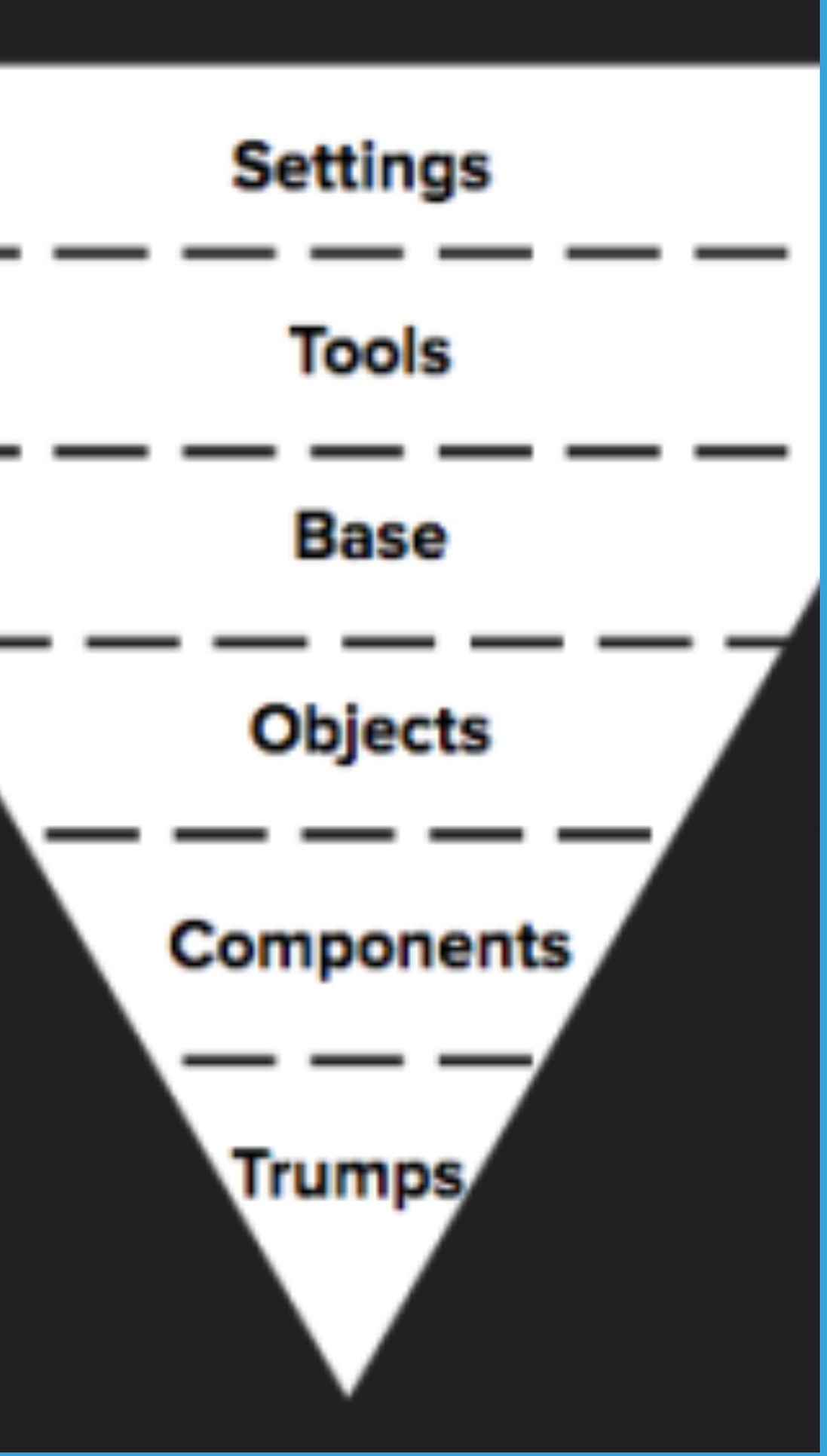
1. Base – HTML, body, h1, ul
2. Layout – Divide the page
3. Module – Reusable modular components
4. State – Same as Modifier in BEM, e.g. hidden, expanded
5. Theme – Color or typographic treatment variations that apply across the site

WHAT DOES SMACSS LOOK LIKE?

```
.btn { ... }

.btn:hover { ... }

.btn.is-pressed { ... }

.btn.is-pressed:hover { ... }

.l-fixed #article { ... }

.l-grid > li { ... }

.pod-callout input[type=text] { ... }
```

SMACSS – GOOD, BAD, AND UGLY

- Reduces specificity wars, adds semantic meaning (not stylistic cues) to class naming style
- Most beneficial when applied *in toto* in a project in a greenfield setting - migrating existing work requires a significant refactor
- Requires conceptual modeling differentiating base, layout, and modules in a way that requires understand the whole project



ITCSS

Harry Roberts

ITCSS HIERARCHY OF CSS RULES

Take SMACSS to the next level and create a hierarchy of the following:

- Settings – Preprocessors - font and color variable definition
  - Tools – Mixins and functions - still no CSS output)
    - Generic – Reset & normalization
      - Elements – Override browser defaults
        - Objects – OOCSS-style broad object definition
          - Components – Specific UI components
            - Utilities – Final overrides

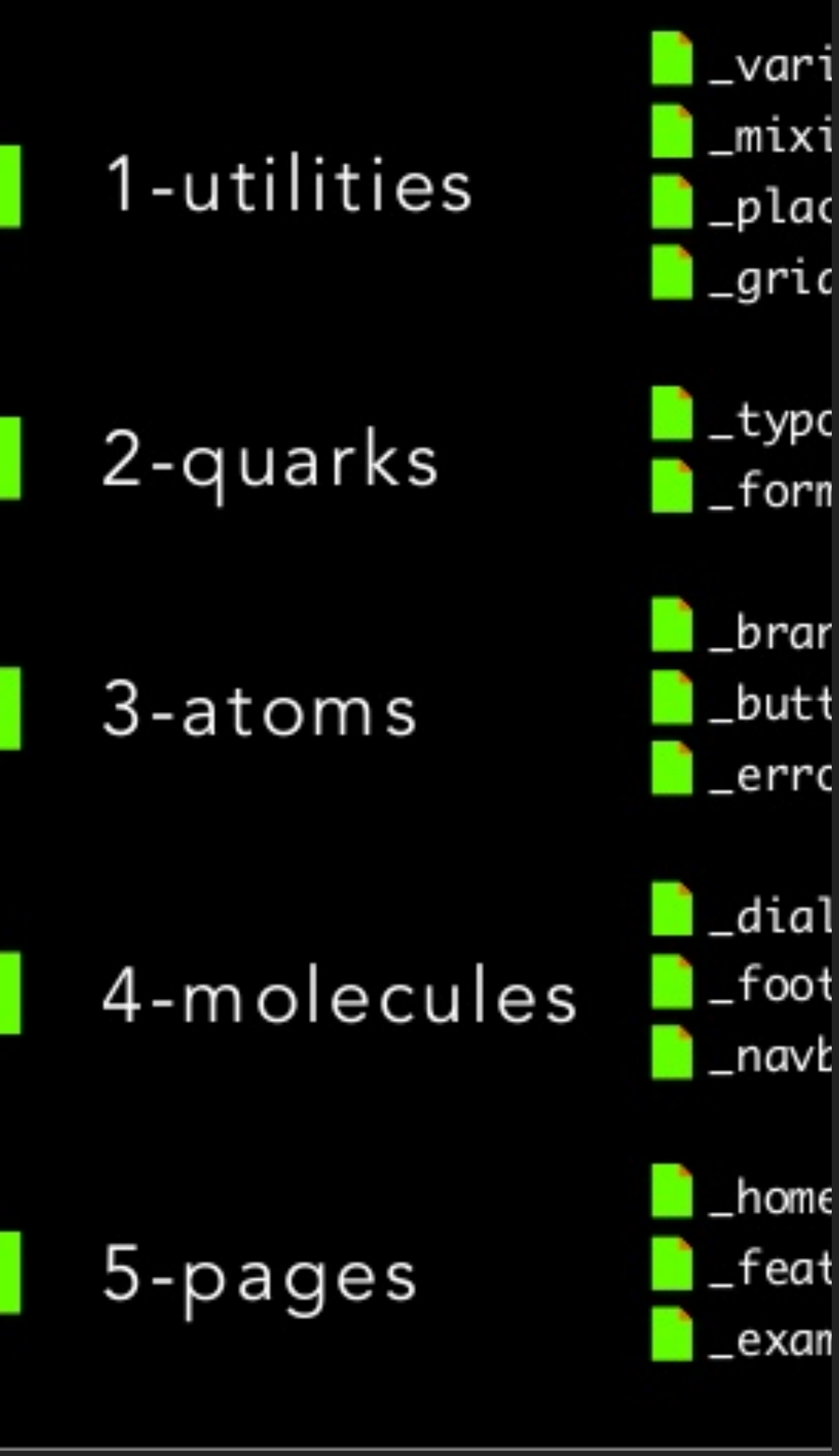
WHAT PROBLEM DOES ITCSS SOLVE?

ITCSS PRINCIPLES AND BENEFITS

- Generic to explicit
- Low specificity to high specificity
- Far-reaching to localized
- Each layer of CSS has a similar level of specificity
- Explicitly creates a structure that avoids specificity wars
- Very clear structure dictating where different things go

ITCSS DOWNSIDES

- Benefits are most pronounced when the entire codebase follows this structure
- Hard to incorporate this practice into an existing codebase without a massive refactor
- Requires considerable forethought and decision-making at each step in order to divide styles in the right way
- Can be used without a preprocessor but is designed with preprocessors in mind
- Relies on use of prefixes for namespacing levels `o-media__img@md`



METAPHOR-BASED

WITH INTERNAL LOGIC

ATOMIC CSS

ATOMIC CSS

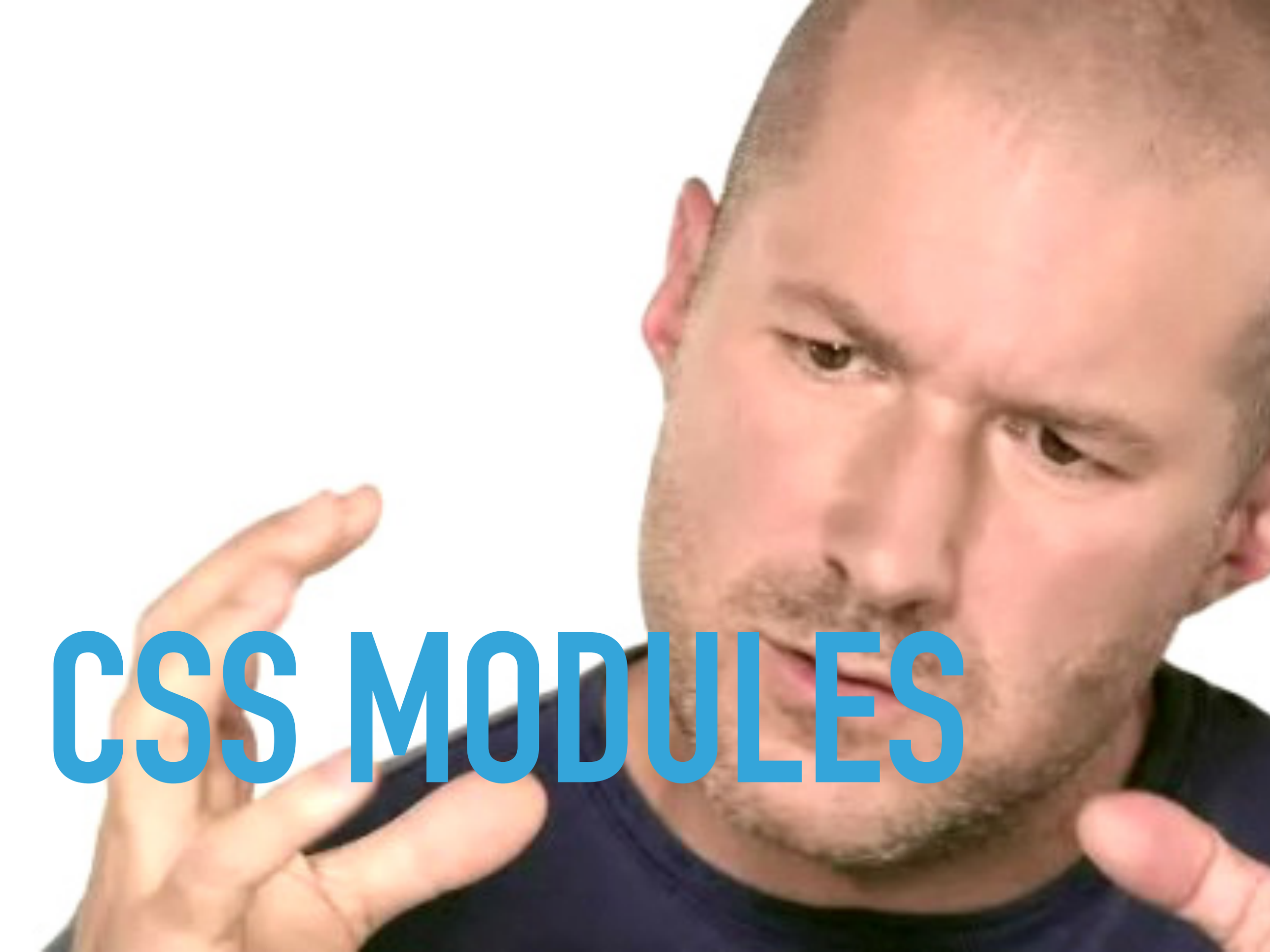
- Declare your styles as functions passed to element class attributes
- Reduces CSS file size
- Single responsibility principle
- One class, one style
- Scope is limited - no reliable on cascading
- Avoids the problem of naming altogether - no class name creation

ATOMIC CSS DOWNSIDES

- According to the docs: “With Atomic CSS, developers don't create bloat because they don't write the selectors, instead they mostly re-use existing classes. This can greatly simplify the learning curve for inexperienced developers.”
- From my opinion - use of abbreviated function names makes this system look like gibberish.
- Adds a level of abstraction on top of CSS that requires knowledge of the Atomic CSS system.
- Styles are not reusable in any project that does not use ACSS

WHAT DOES ATOMIC CSS LOOK LIKE? 😟

```
<div class="Row">
    <div class="Fl(start) W(1/2) Bgc(#0280ae.5) H(90px)"></div>
    <div class="Fl(start) W(1/2) Bgc(#0280ae) H(90px)"></div>
</div>
<div class="D(tb) W(100%)" role="presentation">
    <div class="D(tbc) Bgc(#0280ae) H(90px)"></div>
    <div class="D(tbc) Bgc(#0280ae.5) H(90px)"></div>
</div>
   <div class="IbBox W(50%) Bgc(#0280ae.5) H(90px)"></div>
   <div class="IbBox W(50%) Bgc(#0280ae) H(90px)"></div>
<div class="D(f)">
    <div class="Flxg(1) Bgc(#0280ae) H(90px)"></div>
    <div class="Flxg(1) Bgc(#0280ae.5) H(90px)"></div>
</div>
```

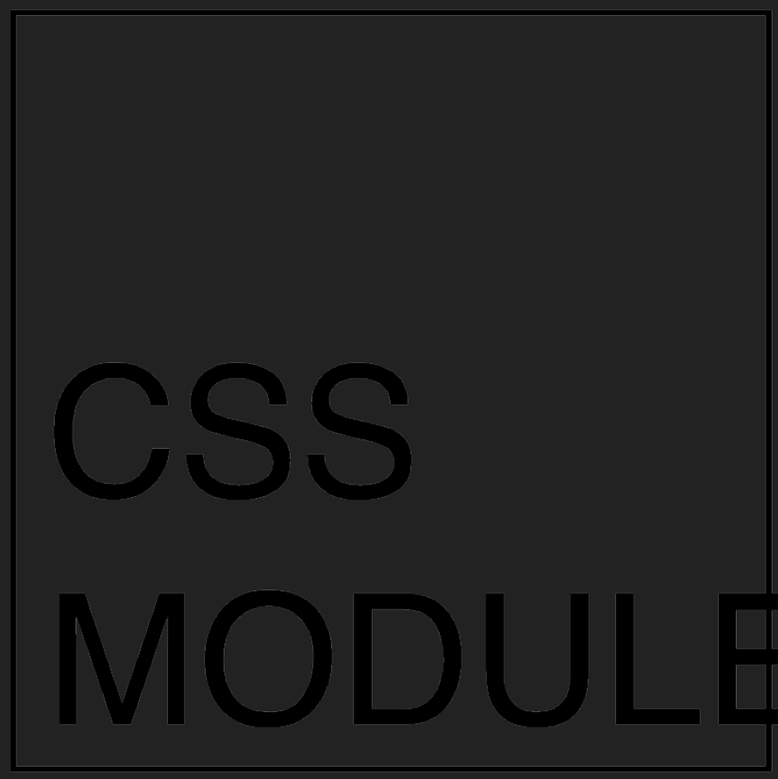

CSS MODULES

CSS MODULES

```
import styles from "./styles.css";

element.innerHTML =
  `<h1 class="${styles.title}">
     An example heading
   </h1>`;
```

=>

```
<h1 class="_styles__title_309571057">
  An example heading
</h1>

._styles__title_309571057 {
  background-color: red;
}
```



Problems with CSS at Scale

1. Global Namespace
2. Dependencies
3. Dead Code Elimination
4. Minification
5. Sharing Constants
6. Non-deterministic Resolution
7. Isolation

CSS MODULES

- Only possible to share styles if you use a preprocessor

```
.element {
  composes: large from "./typography.css";
  composes: dark-text from "./colors.css";
  composes: padding-all-medium from "./layout.css";
  composes: subtle-shadow from "./effect.css";
}
```

- Have many similar problems to CSS-in-JS generally
- Not sharable with projects that don’t use this
- Level of abstraction only valuable in the context of itself

CSS IN JAVASCRIPT

An Idea by Javascript Devs

CSS IN JAVASCRIPT

- With React, now we're putting HTML in the JavaScript.
- "Why not do the same with CSS." :/
- Write CSS in JavaScript objects and pass styles to components by referencing keys of the objects
- Works *excellently* for encapsulated UI components and for handling all related concerns in a single file
- Reduces development time by colocating styles with components

WHAT DOES CSS-IN-JS LOOK LIKE?

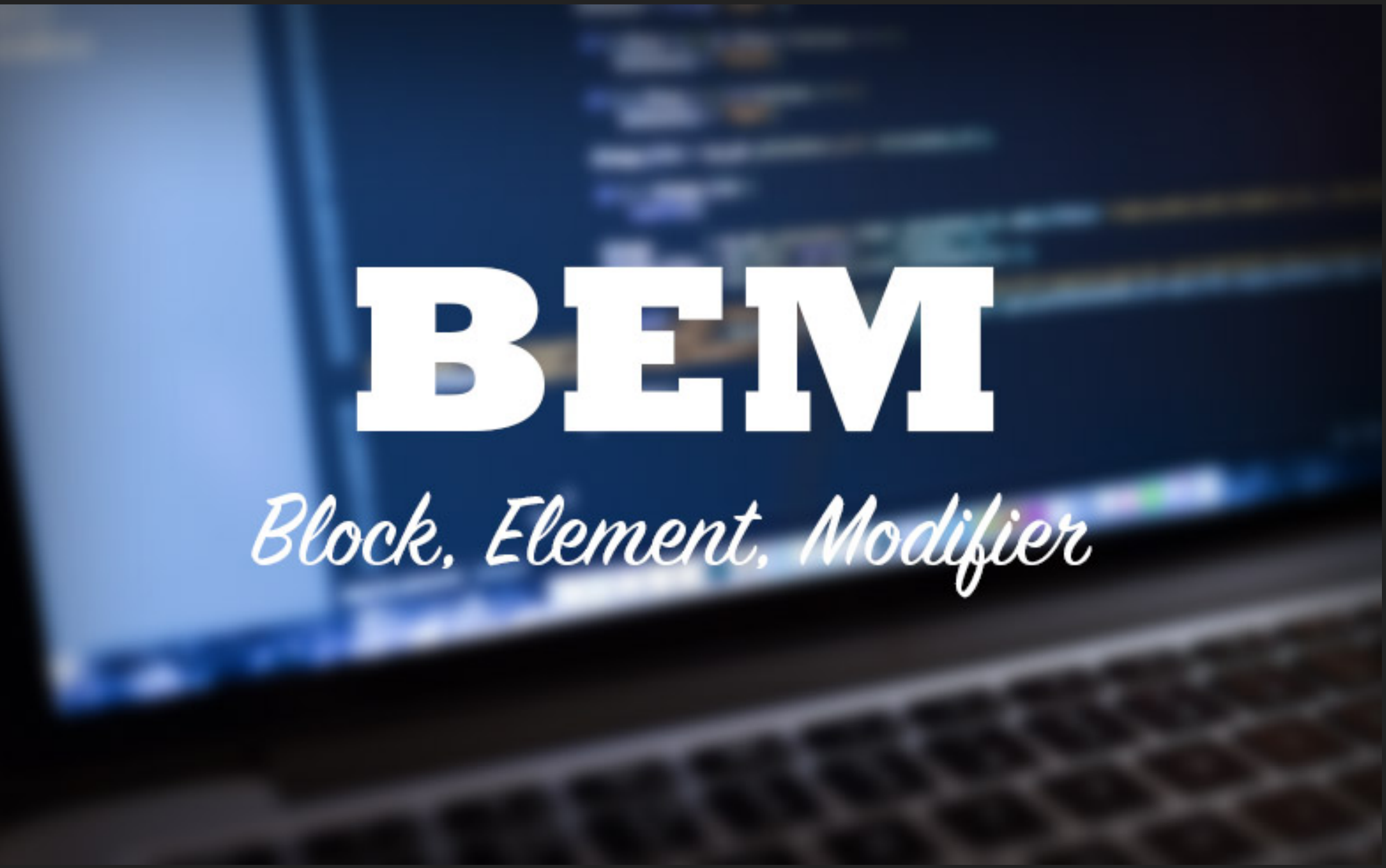
```
const styles = StyleSheet.create({
  button: {
    background: gradient,
    borderRadius: '3px',
    border: 0,
    color: 'white',
    height: '48px',
    textTransform: 'uppercase',
    padding: '0 25px',
    boxShadow: '0 3px 5px 2px rgba(255, 105, 135, .30)',
  },
});
```

CSS-IN-JS DOWNSIDES

- Changes how CSS is declared syntactically in a way that makes it less portable between projects,
  i.e. `border-radius` becomes `borderRadius`
- All styles are *inlined* which means that if you are using a library that uses CSS-in-JS your ability to change styles depends on their component API implementation,
  e.g. the Material UI React library doesn't provide style API
- Lose the ability to cache CSS in the browser, resulting in possibly longer loading times
- Possibility for divergent styles between components if care is not taken
- **Does not handle adaptive / responsive design well**

CSS SYSTEMS

- JavaScript devs *love* CSS-in-JS and see it as the next evolution of CSS following CSS Modules
- It serves some specific use cases very well, such as modularized UI components
- IMHO it is not appropriately suited to most large projects or multiple projects with multiple devs
- Because inline styles cannot be overridden in CSS, it is not easily incorporated into projects that don’t use it overall


BEM
Block, Element, Modifier

BLOCK-ELEMENT-MODIFIER

- BEM is a class naming convention
- Accomplishes several things:
  - Semantically describes the UI
  - Provides a framework for making naming decisions
  - Avoids collisions through namespacing
  - Eliminates the need to nest classes
  - Keeps specificity war and avoids specificity wars

BLOCK-ELEMENT-MODIFIER

What is a block?

> A block represents an object. It could be a registration form, a hero section on a splash page, or a footer.

What is an element?

> An element is defined in relation to its block. For instance, the main image of a hero section.

What is a modifier?

> A modifier represents the state of an element, e.g. `hidden`

WHAT DOES BEM LOOK LIKE?

```
.block__element--modifier { … } OR

.block-element--modifier { … }
```

```
<body>
    <header class="header">
        <div class="header__logoContainer">
            <img class="logoContainer__logo" src="assets/images/img.jpg">
        </div>
        <nav class="header__nav">
            <ul>
                <li class="nav__item nav__item--selected">home</li>
                <li class="nav__item">about me</li>
                <li class="nav__item">skills</li>
                <li class="nav__item">portfolio</li>
                <li class="nav__item">contact me</li>
            </ul>
        </nav>
    </header>
```

BLOCK-ELEMENT-MODIFIER BENEFITS

- Better CSS Performance - Rendering engines evaluate CSS selectors from right to left. The less they have to evaluate, the faster it renders (not the biggest perf concern)
- Using single classnames makes it easier to rework an existing CSS code base
- Does not need global styles like SMACSS, doesn’t need resets or normalization
- Encourages semantic thinking
- Integrates well with design processors (OOUX)

BLOCK-ELEMENT-MODIFIER DOWNSIDES

- Long classnames are descriptive, not pithy
- Increases bytes in CSS and HTML
  - This is counteracted by minification and compression
- May have more duplication than with utility-based OOCSS-type models
  - This can be mitigated with the use of mixins with preprocessors

USING BEM

- Only use a block name once
- Describe components semantically, not in terms of their style specifics
- It is not necessary to follow the conventional naming convention, and I prefer not to. My pref:

`block-element--modifier` **not**

`block__element--modifier`

EXAMPLE

```
.countries {
   font-family: "Helvetica Neue";
   font-size: 18px;
}

.country-form-wrapper {
   margin-bottom: 24px;
   margin-top: 8px;
}

.country-form-label {
   font-size: 24px;
}

.country-form-input {
   border: 1px solid #9ea7af;
   font-size: 24px;
   height: 40px;
   margin-left: 16px;
   padding-left: 8px;
}
```

```
.country-form-submit,
.country-form-submit--disabled {
   background-color: #ebebeb;
   border: 1px solid #9ea7af;
   border-radius: 4px;
   font-size: 24px;
   margin-left: 8px;
   padding: 8px 24px;
   transition: background-color ease
150ms;
}

.country-form-submit--disabled {
   background-color: #ddd;
   cursor: not-allowed;
}

.country-form-error-text {
   color: #ff0033;
}
```

SOURCES

- OOCSS
  - https://www.smashingmagazine.com/2011/12/an-introduction-to-object-oriented-css-oocss/
  - http://oocss.org/
  - https://appendto.com/2014/04/oocss/
  - https://github.com/stubbornella/oocss/wiki
- SMACSS
  - https://smacss.com/
  - http://vanseodesign.com/css/smacss-introduction/

SOURCES

- ITCSS
  - http://www.creativebloq.com/web-design/manage-large-css-projects-itcss-101517528
  - https://medium.com/@pistenprinz/css-at-trivago-part-1-structure-and-itcss-52f63ed557ca
  - https://medium.com/@jordankoschei/how-i-shrank-my-css-by-84kb-by-refactoring-with-itcss-2e8dafee123a
  - https://csswizardry.com/2015/08/bemit-taking-the-bem-naming-convention-a-step-further/
- Atomic CSS
  - https://acss.io/

SOURCES

- CSS Modules
  - https://glenmaddern.com/articles/css-modules
  - https://css-tricks.com/css-modules-part-1-need/
- CSS In Javascript
  - https://www.kirupa.com/html5/setting_css_styles_using_javascript.htm
  - https://medium.freecodecamp.org/css-in-javascript-the-future-of-component-based-styling-70b161a79a32

SOURCES

- BEM
  - https://medium.com/@GarrettLevine/bem-helps-beginners-learn-html-css-3721a2091f1c
  - https://www.sitepoint.com/bem-smacss-advice-from-developers/
  - https://www.slideshare.net/BobDonderwinkel/bem-presentation-40907446
  - https://www.smashingmagazine.com/2014/07/bem-methodology-for-small-projects/
  - http://nicolasgallagher.com/about-html-semantics-front-end-architecture/
  - https://csswizardry.com/2013/01/mindbemding-getting-your-head-round-bem-syntax/
  - https://mattstauffer.co/blog/organizing-css-oocss-smacss-and-bem
  - http://frontendbabel.info/articles/bem-with-css-preprocessors/