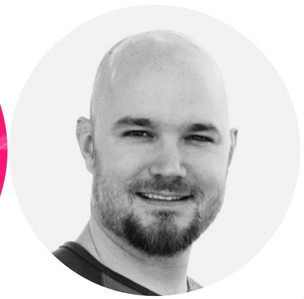
Scalable, Maintainable CSS/SCSS Architecture in Angular



Brian Treese
Chief of User Experience

www.socreate.it

Much smaller scale

No more bleed, conflict, or overriding

Many different options

With great power comes great responsibility


CSS
Clean
Organized
Structured
Scalable

CSS Preprocessors

CSS Preprocessors

What We'll Cover

- Global styles
- Naming conventions
- Relative units and predictability
- Selectors and overrides
- Structure and organization
- Mixins and variables

Global Styles: A Traditional Approach

We don’t have to change if we don’t want to

Are you...

Using a flat, low specificity approach?

Using a naming convention system like B.E.M.?

Breaking up large files into partials?

View encapsulation

~~No style encapsulation~~

~~Global styles~~


Global
CSS
VS
Local
CSS

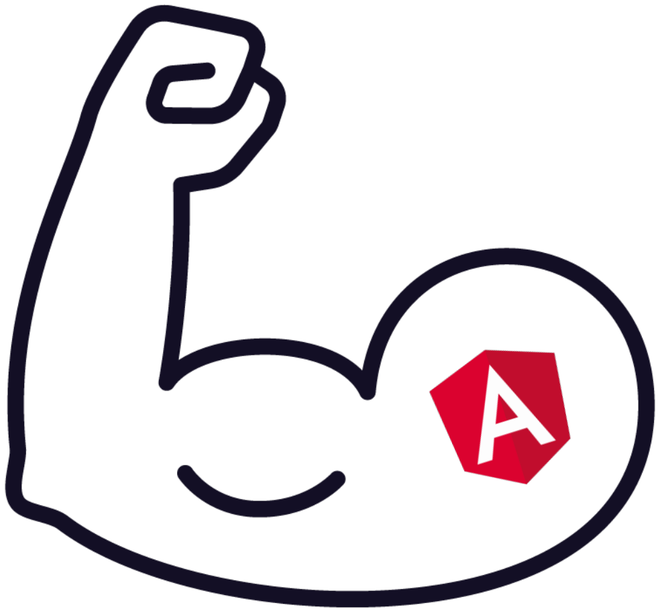
≠

Global

Global

- Browser resets
- Colors
- Typography
- Layout
- Media queries
- Utilities

Global Styles: A More Modular Approach



Style encapsulation

Not all styles need to be encapsulated

Some global styles are ok

Applying Global Styles

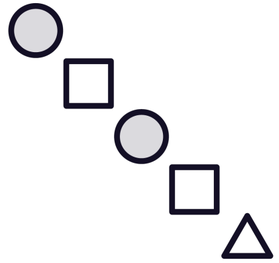
Class based system

Mixins and variables only

Similar but different

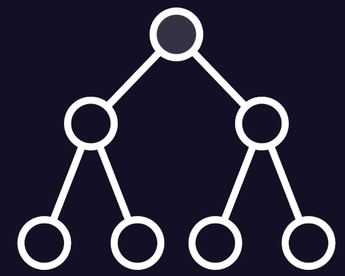
Global

+



Local

We need somewhere to put them

Sass = 

Modular

How Will We Include Them?

styles.scss

Turn off view encapsulation

App Component Need Scoped Styles?

Nope

Change view encapsulation mode

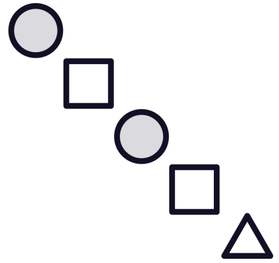
Yep

Can’t use view encapsulation

View encapsulation mode =

<link> =

Global



Local

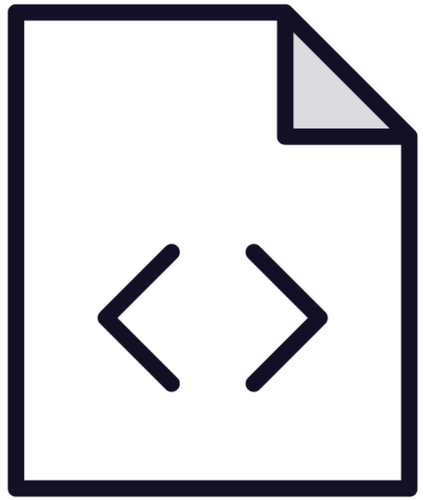
Long, ugly selectors everywhere!

Mixins and variables

NEVER override styles!

NEVER override styles!
ok, almost never

Raw Unstyled Code



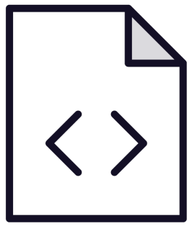
Almost

Using a CSS reset?

It should still be global

Normalize.css

A modern, HTML5-ready alternative to CSS resets

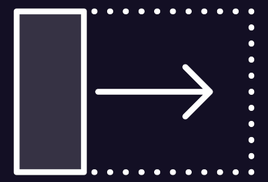


Raw and Unstyled

Ultimate freedom

Either use globals or don’t

Easy to opt in or out



Works Better With Media Queries

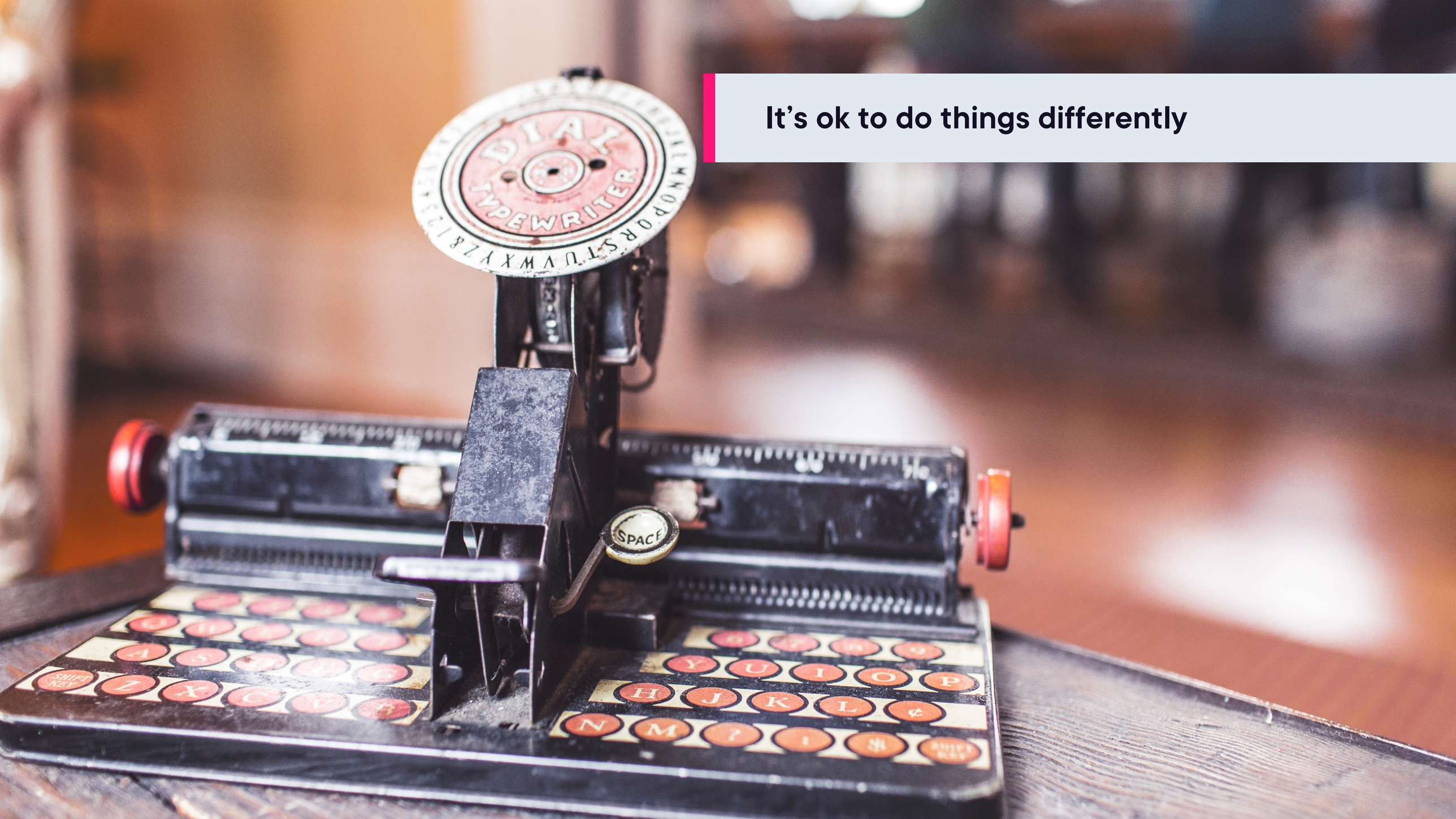
And with global classes...

And with global mixins and variables...

This is my favorite, but...

This is my favorite, but...

it still has its own issues



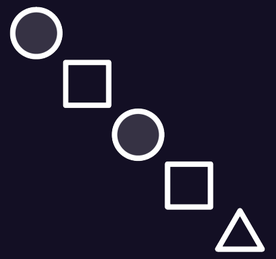
It’s ok to do things differently

Building for the Future

≠

Global

=



Local

Global mixins and variables approach still works!

Simplifying Global Imports for Preprocessors

some.component.scss

```
@import “../../../../global/scss/typography/headers”
```

Naming Conventions

Naming Is Hard

SMACSS

OOCSS

Atomic design

B.E.M.

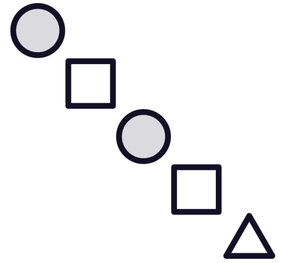
≠

Global

Absolutely not!

They're Still Beneficial!

Traditionally applied
on a global level



Can be applied
locally in Angular

Components

Functionality

Style

Markup

```
class="block__element--modifier">
```

B.E.M.

Standalone blocks
Meaningful on their own
Can be moved with little worry

B.E.M.

```
<ul class="block">
    <div class="block__element">
        I'm Normal
    </div>
    <div class="block__element block__element--modifier">
        I'm Different
    </div>
</div>
```

B.E.M.

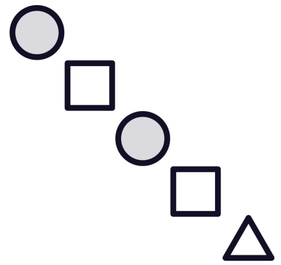
```
<ul class=“nav”>
    <li class=“nav__item”>
        Home
    </li>
    <li class=“nav__item nav__item--active”>
        About
    </li>
</ul>
```

In Traditional Web Apps...

Names are global

requiring more thought and planning

In Angular Apps...



Names are local

requiring less thought and planning

Traditional

```
<ul class=“app-navbar-header”>
</ul>
```

Angular

```
<ul class=“items”>
</ul>
```

Traditional

```
<ul class="app-navbar-header">
    <li class="app-navbar-header__item">
    </li>
    <li class="app-navbar-header__item">
    </li>
</ul>
```

Angular

```
<ul class="items">
    <li class="item">
    </li>
    <li class="item">
    </li>
</ul>
```

Traditional

```
<ul class="app-navbar-header">
    <li class="app-navbar-header__item">
        <a class="app-navbar-header__item-link">
            Home
        </a>
    </li>
    <li class="app-navbar-header__item">
        <a class="app-navbar-header__item-link">
            About
        </a>
    </li>
</ul>
```

Angular

```
<ul class="items">
    <li class="item">
        <a class="item-link">
            Home
        </a>
    </li>
    <li class="item">
        <a class="item-link">
            About
        </a>
    </li>
</ul>
```

Traditional

```
<ul class="app-navbar-header">
    <li class="app-navbar-header__item">
        <a class="app-navbar-header__item-link">
            Home
        </a>
    </li>
    <li class="app-navbar-header__item">
        <a
            class="
                app-navbar-header__item-link
                app-navbar-header__item-link--selected">
            About
        </a>
    </li>
</ul>
```

Angular

```
<ul class="items">
    <li class="item">
        <a class="item-link">
            Home
        </a>
    </li>
    <li class="item">
        <a
            class="
                item-link
                item-link--selected">
            About
        </a>
    </li>
</ul>
```

Why Bother with B.E.M.?

1.

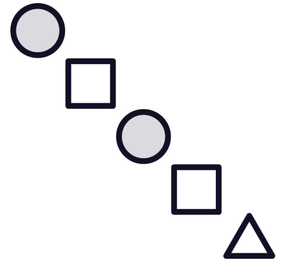
It’s easier to name things

2.

It’s easier as markup grows

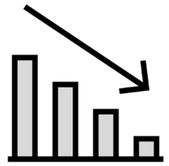
Still Have the Same Problem Just on a Smaller Scale

Traditional
Global scale



Angular
Local scale

Some components are large



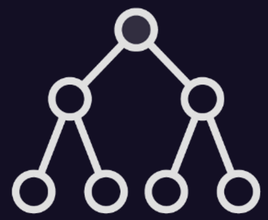
Some components are small

We just don’t know

Structure and organization?
Yep, still important!

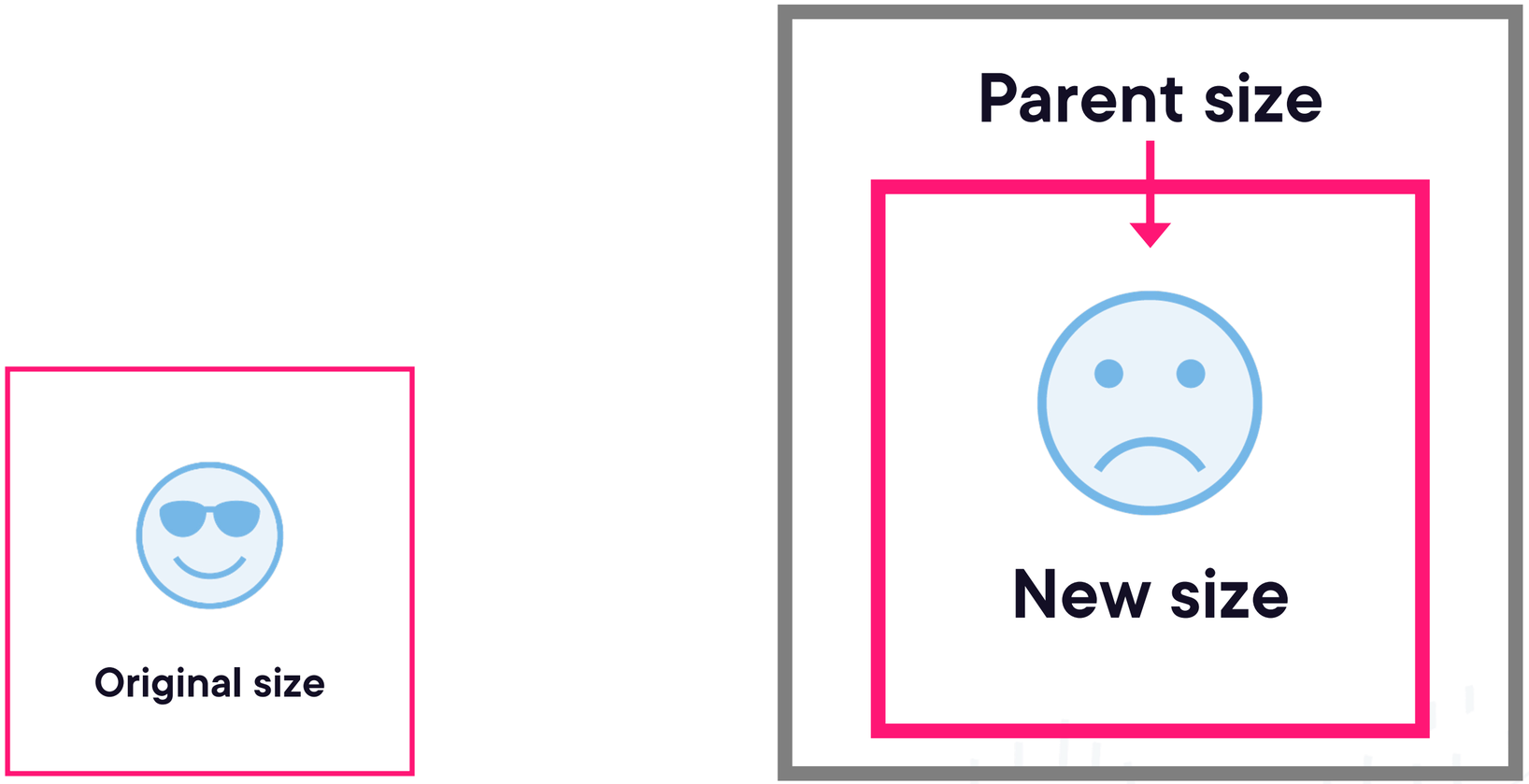
Predictable Sizing with Relative Units

=



Unknown

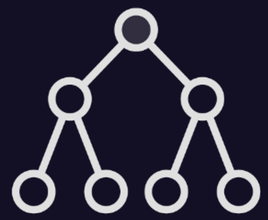
Reliability?



Dynamic CSS allows for change

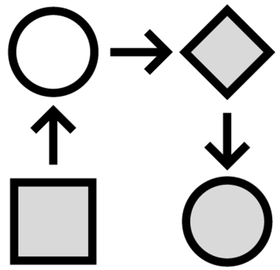
Relative units

=



Unknown

We Need a System!



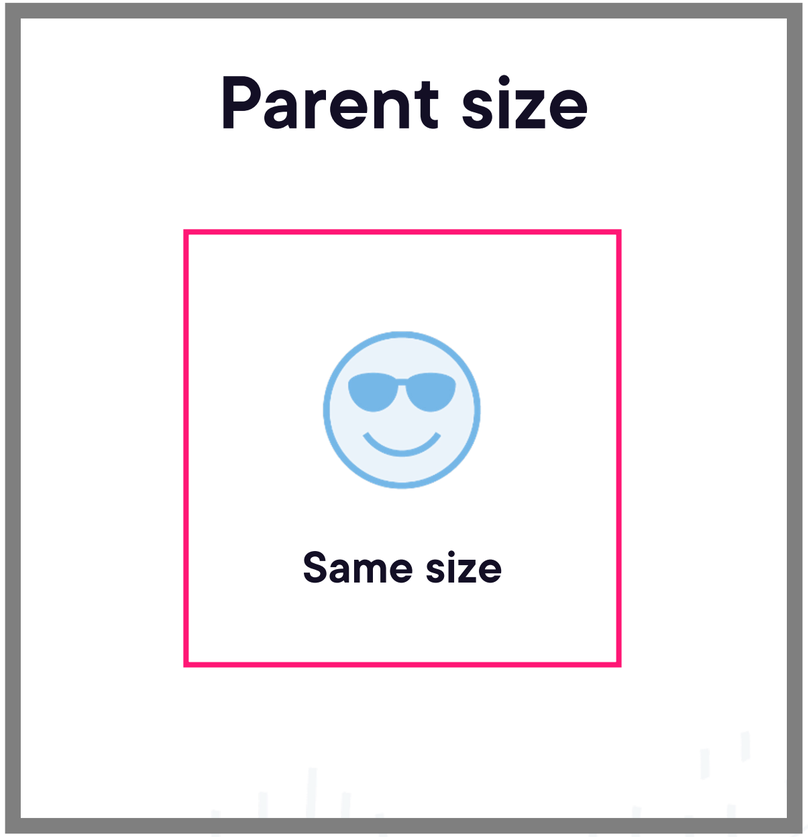
And we need it to be predictable!

REM units

some.component.scss

```scss
:host {
   font-size: 1rem;
}
```

Reliability?



Global styles

Name and structure classes

Apply relative units

Specificity and overrides

CSS Selectors and Style Overrides

Good CSS Practices

Keep specificity low

Avoid style overrides

Only Use Classes

```
.class {
   color: blue;
}
```

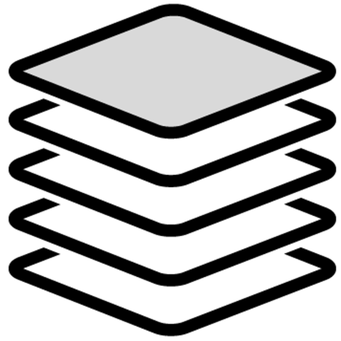
Only Use Classes

```
.class:hover {
   color: blue;
}
```

Only Use Classes

```
.class + .sibling {
   color: blue;
}
```

Avoid Specificity Wars



Yuck!

```
ul > li:first-child + li > a {
   color: #2A9FBC;
}
```

Yuck!

```
#navbar-primary > ul > li:first-child + li > a {
    color: #F15B2A;
}
```

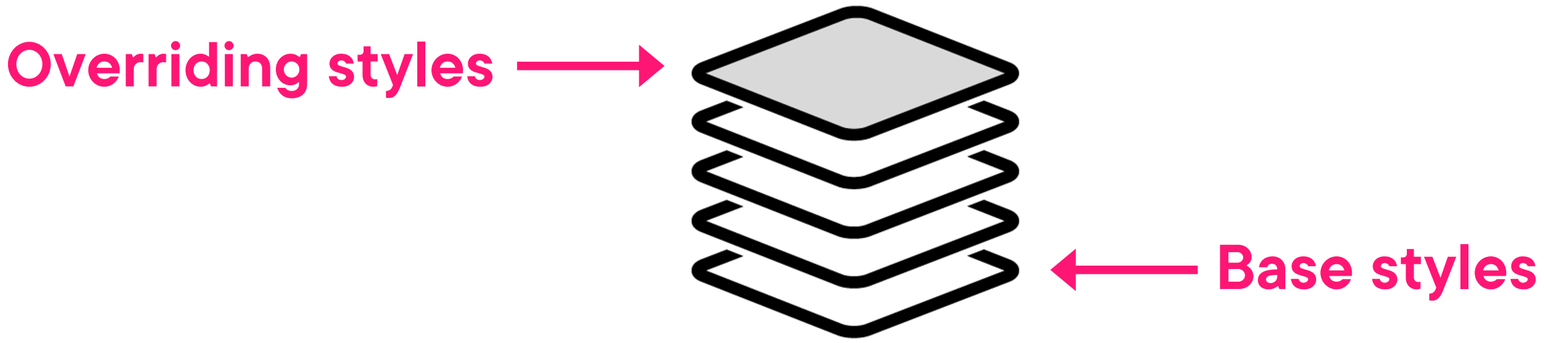
Creates More Consistent Code


EASY

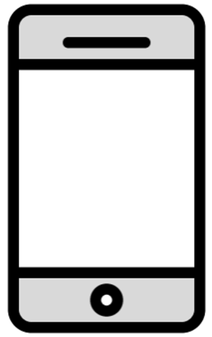
+


IDs

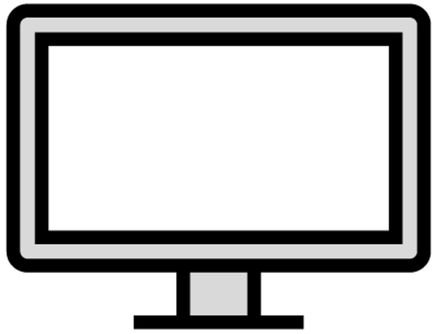
Overriding Styles



Responsive Design



Mobile first



Desktop first

Overrides →

Tablet Laptop Desktop

← Overrides

Phone Tablet Laptop

Purpose

Reliability

Component CSS Structure and Organization

Organization

Important in traditional web apps

Even more important in Angular

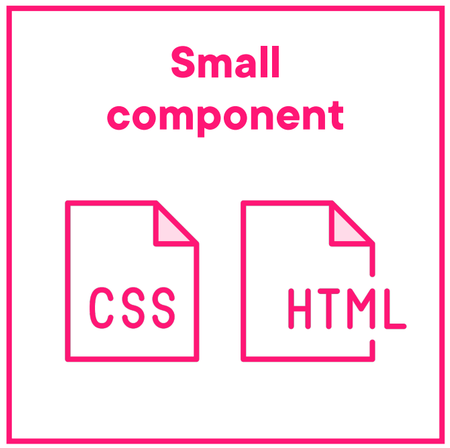
Keep files small

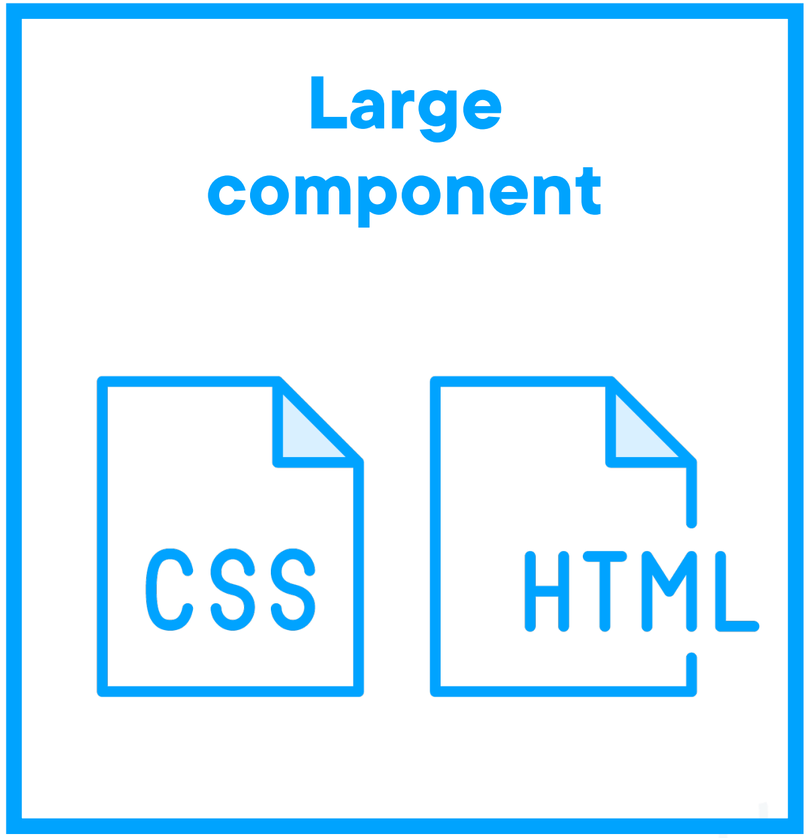
Make them easy to work with

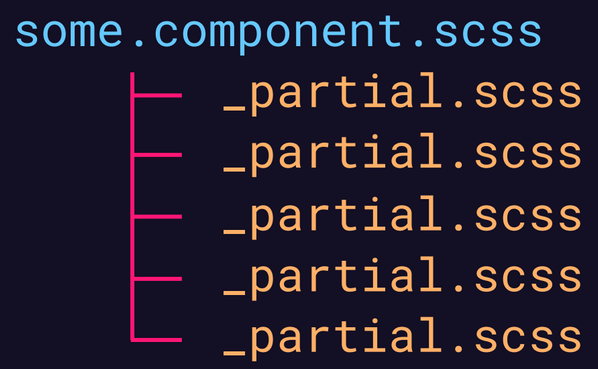
CSS Preprocessors and Frameworks

We Can Do This in Angular!

We never know


Small
component
CSS
HTML


Large
component
CSS
HTML


some.component.scss
_partial.scss
_partial.scss
_partial.scss
_partial.scss
_partial.scss

SASS @use

```
@use ‘scss/globals’;
@use ‘scss/header’;
@use ‘scss/content’;
@use ‘scss/footer’;
```

```
styleUrls: [‘file-01.scss,file-02.scss,file-03.scss’];
```

It’s a matter of preference

Mind Having Multiple Style Blocks?

Nope

Use styleUrls!

Yep

Use SASS partials

Component Styles

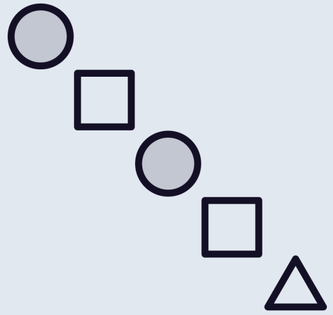
Structured

Well organized

Local Mixins and Variables

Global styles

Mixins and variables



Local styles

Mixins and variables

What Can We Do?

Create local mixins and variables

Variable Scope

```
.class {
    $pad: 1em;
    $padding-top: $pad;
    $padding-right: $pad;
}
```

Not available outside .class {}

Variables

On a file level

```
// Settings
$component__pad: 1em;
$component__border: 0.5em;
$component__spacing: 2em;
```

Mixins

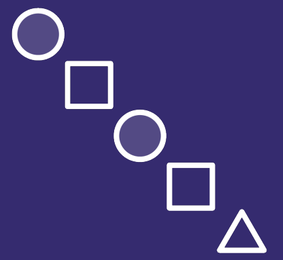
On a file level

```
// Mixins
@mixin component__box {
    border: solid 0.2em #ccc;
    margin: 2em;
    padding: 1em;
}
```

Mixins and Variables

Shared across components

Mixins and Variables



As local as possible

Mixins and Variables

Block

File

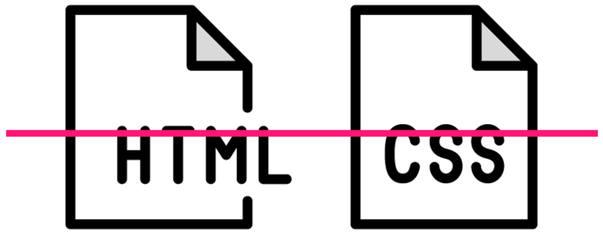
Component

Grouping of components

Global

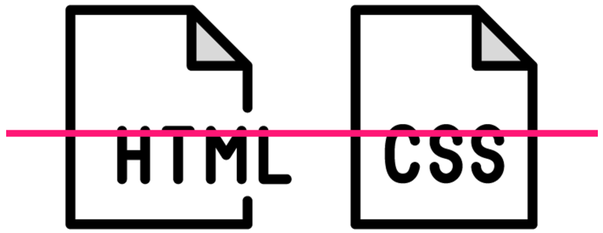
Styling the Pre-bootstrap Loading Screen

Add Style and Markup



It’ll be removed when the app is loaded!

It’s All in One Spot!



It won’t hang around!

Summary

- How to use, structure, and maintain styles
- Use SASS to create better code
- Global styles as a global stylesheet
- Global styles as mixins and variables
- Naming conventions are still important
- Use relative units predictably
- Use low specificity class-based selectors
- Avoid overriding styles
- Create small, single purpose, files
- Use a local first approach
- Clean and simple app loading screen

Component Theming