# Scalable, Maintainable CSS/SCSS Architecture in Angular

**Brian Treese**
CHIEF OF USER EXPERIENCE AT SOCREATE.IT

@brianmtreese

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

# What We'll Cover

- Global Styles
- Naming Conventions
- Relative Units & Predictability
- Selectors & Overrides
- Structure & Organization
- Mixins & 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

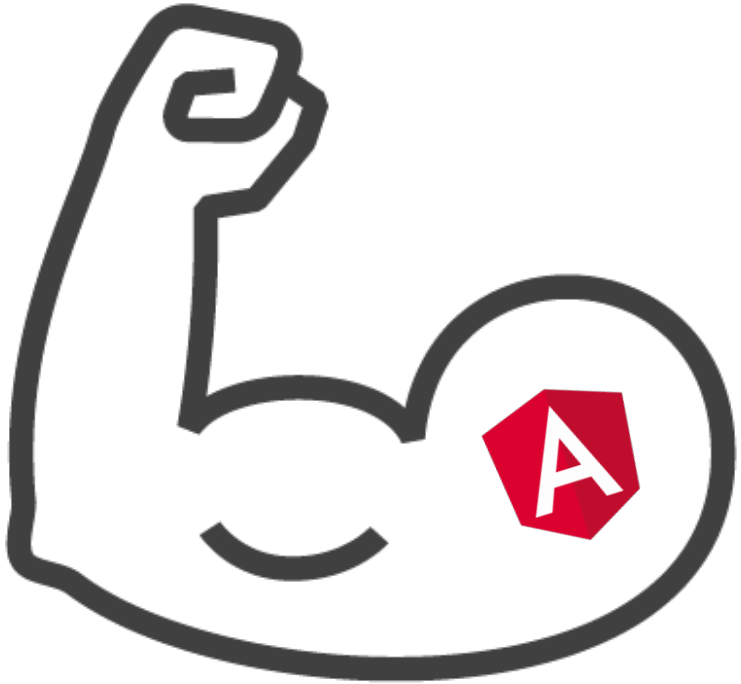
≠

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

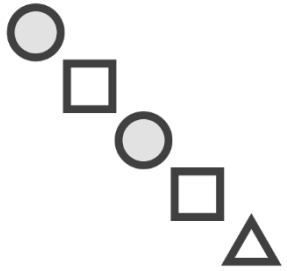
**Class Based System**

**Mixins & Variables Only**

# Similar but Different

Global

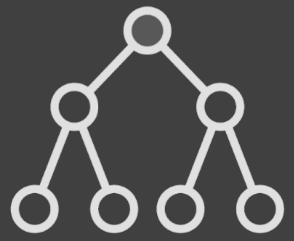
+



Local

# We Need Somewhere to Put Them


=



Modular

# How Will We Include Them?

styles.scss

Turn Off View Encapsulation

# App Component Need Scoped Styles?

**Nope**

**Change View Encapsulation Mode**

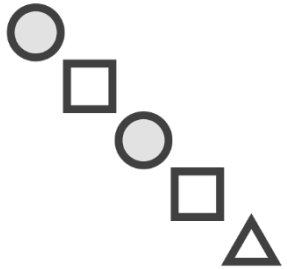
**Yep**

**Can’t Use View Encapsulation**

View Encapsulation Mode

<link> =

Global



Local

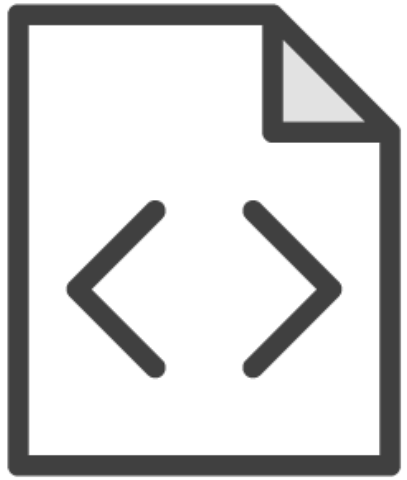
# Long, Ugly Selectors Everywhere!


# Mixins & Variables

# NEVER Override Styles!

# NEVER Override Styles!
# Ok, Almost Never

# Raw Unstyled Code



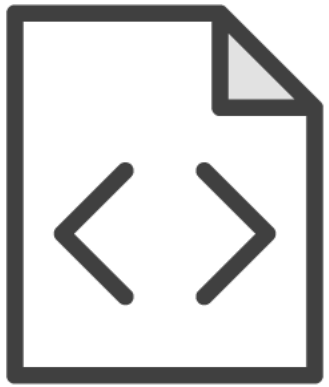
Almost

# Using a CSS Reset?

# It Should Still Be Global

# Normalize.css

## A modern, HTML5-ready alternative to CSS resets



# Raw & Unstyled

- Ultimate Freedom
- Either Use Globals or Don’t
- Easy to Opt in or Out

# And With Global Classes...

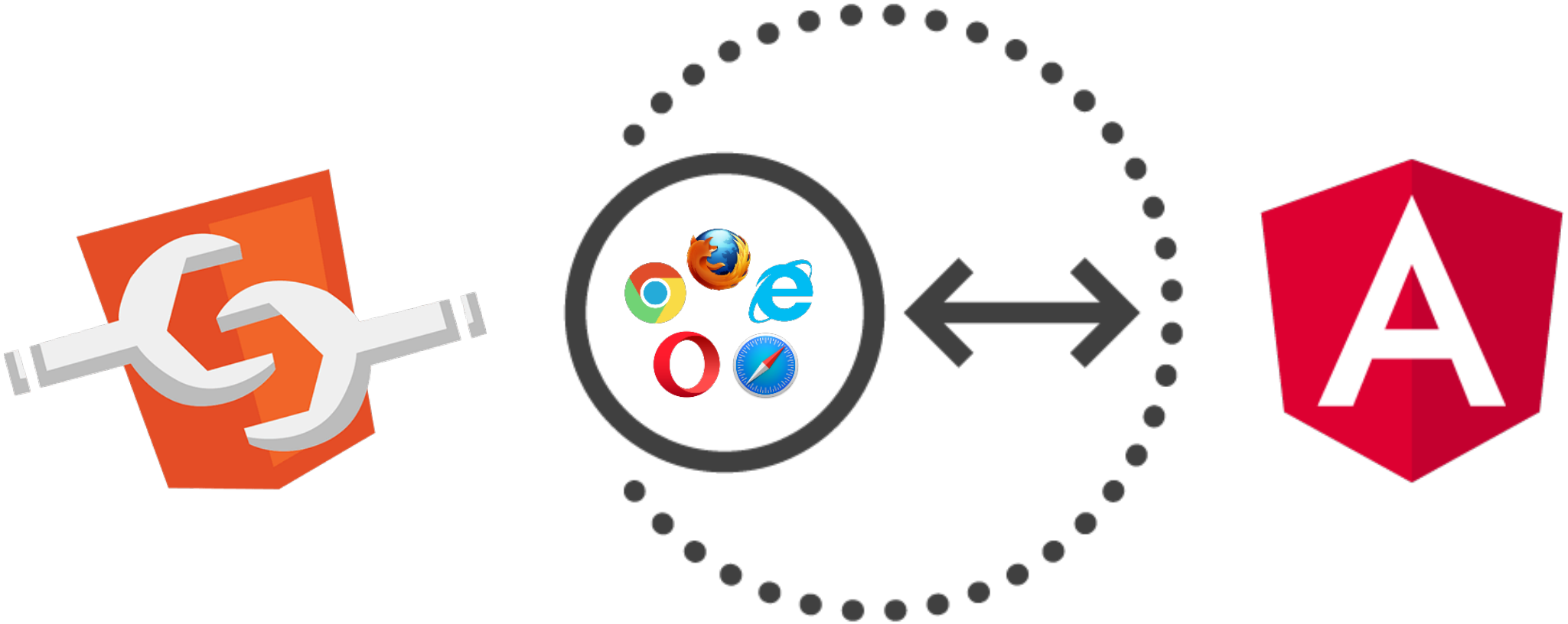
# And With Global Mixins & Variables...

# This is My Favorite, But...

# This is My Favorite, But...
# It Still Has its Own Issues


It’s Ok to
Do Things
Differently
DIAL
TYPEWRITER
SPACE

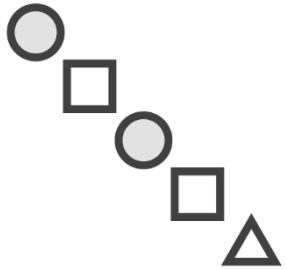
# Building for the Future



≠

Global

=



Local

# Global Mixins & 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.

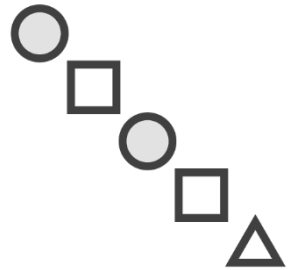
≠

Global

Absolutely Not!

# They're Still Beneficial!

**Traditionally Applied on a Global Level**



**Can Be Applied Locally in Angular**

# Components

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

# B.E.M.

Standalone Blocks

Meaningful on Their Own

Can Be Moved with Little Worry

# B.E.M.

```
<div 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.

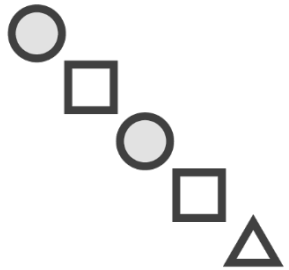
```
<ul class="nav">
    <li class="nav__item">
        Home
    </li>
    <li class="nav__item nav__item--active">
        About
    </li>
</ul>
```

# In Traditional Web Apps...

## Names Are Global

Requiring More Thought & Planning

# In Angular Apps...



## Names Are Local

Requiring Less Thought & 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
            app-navbar-header__item--selected">
        <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
            item--selected">
        <a class="item-link">
            About
        </a>
    </li>
</ul>
```

# Why Bother With B.E.M.?

## 1.

It’s Easier to Name Things

# Why Bother With B.E.M.?

**1.**

**It's Easier to Name Things**

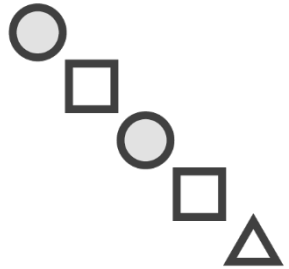
**2.**

**It's Easier as Markup Grows**

# Still Have the Same Problem Just on a Smaller Scale

**Traditional**

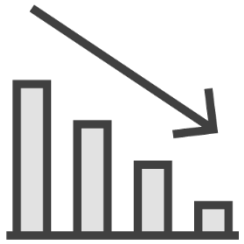
Global Scale



**Angular**

Local Scale

# Some Components Are Large



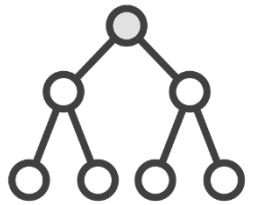
# Some Components Are Small

# We Just Don’t Know

# Structure & Organization?
# Yep, Still Important!

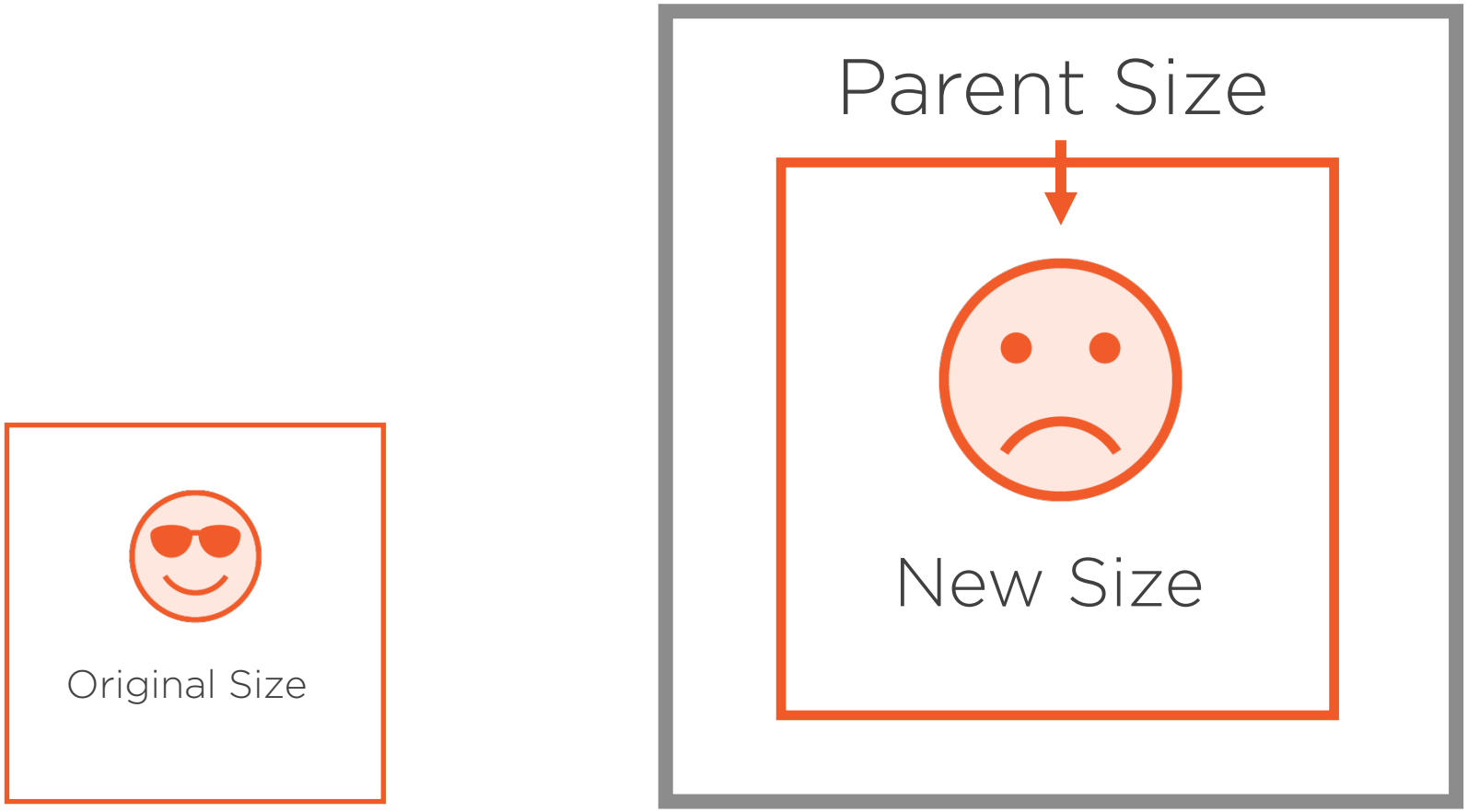
# Predictable Sizing with Relative Units

=



Unknown

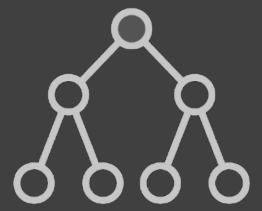
# Reliability?




Dynamic
CSS Allows
for Change

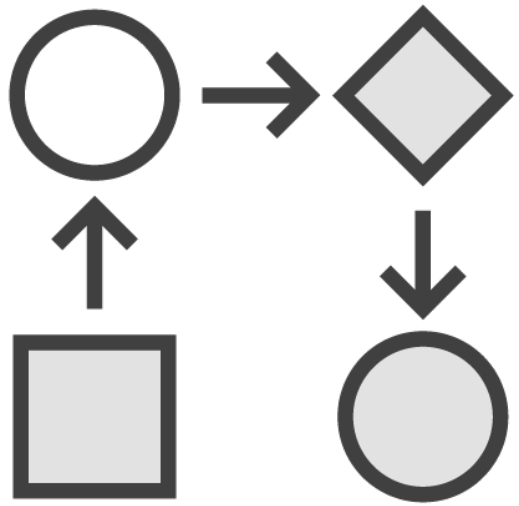
# Relative Units

=



Unknown

# We Need a System!



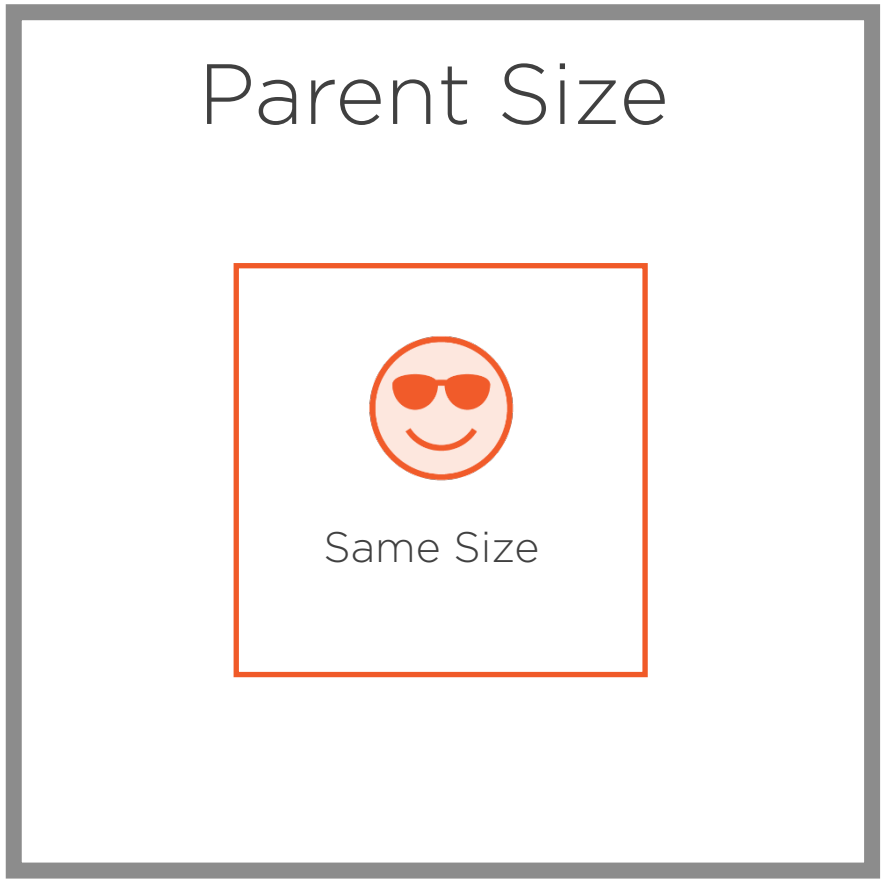
# And We Need It to Be Predictable!

# REM Units

# some.component.scss

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

# Reliability? ✓

Global Styles

Name & Structure Classes

Apply Relative Units

Specificity & Overrides

# CSS Selectors & Style Overrides

# Good CSS Practices


# 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 > a {
    color: #F15B2A;
}
```

# Creates More Consistent Code

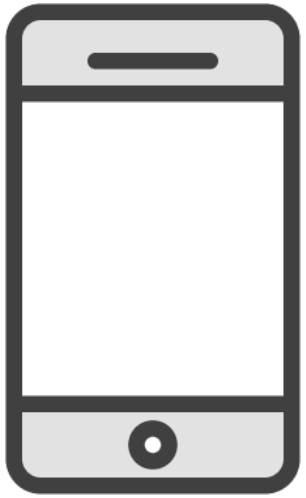

EASY

+


IDs

# Overriding Styles

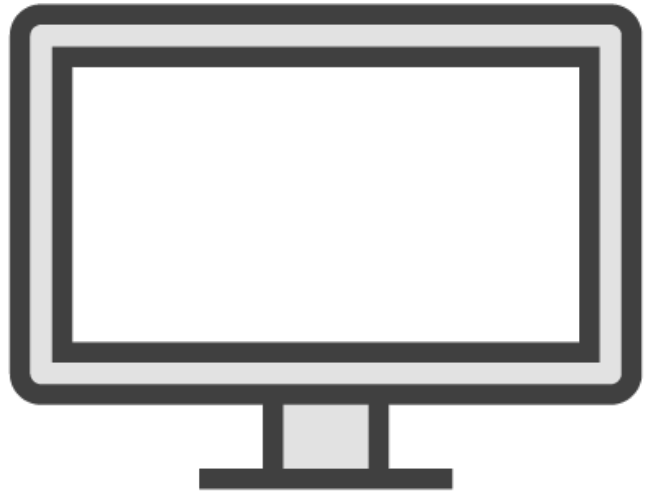
# Responsive Design



## Mobile First

Overrides

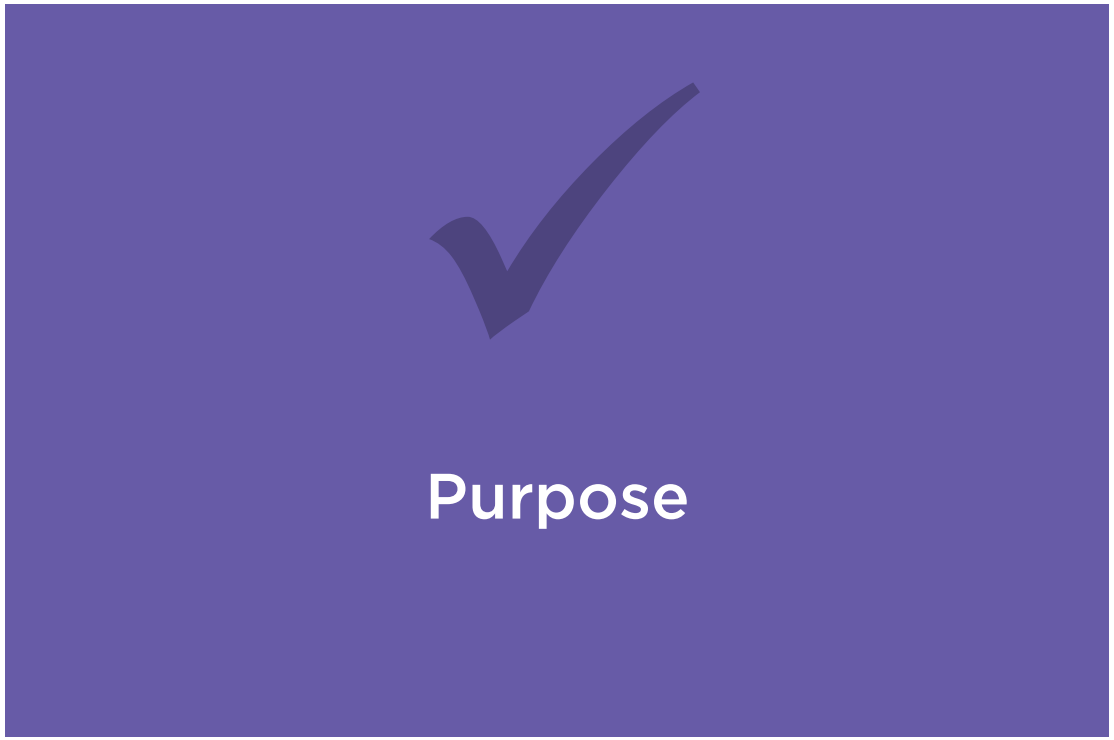
Tablet Laptop Desktop



## Desktop First

Overrides

Phone Tablet Laptop


Purpose


Reliability

# Component CSS Structure & Organization

# Organization

Important in Traditional Web Apps

Even More Important in Angular

Keep Files Small

Make Them Easy to Work With

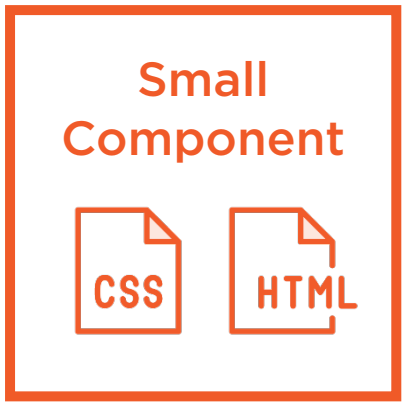
# CSS Preprocessors & Frameworks

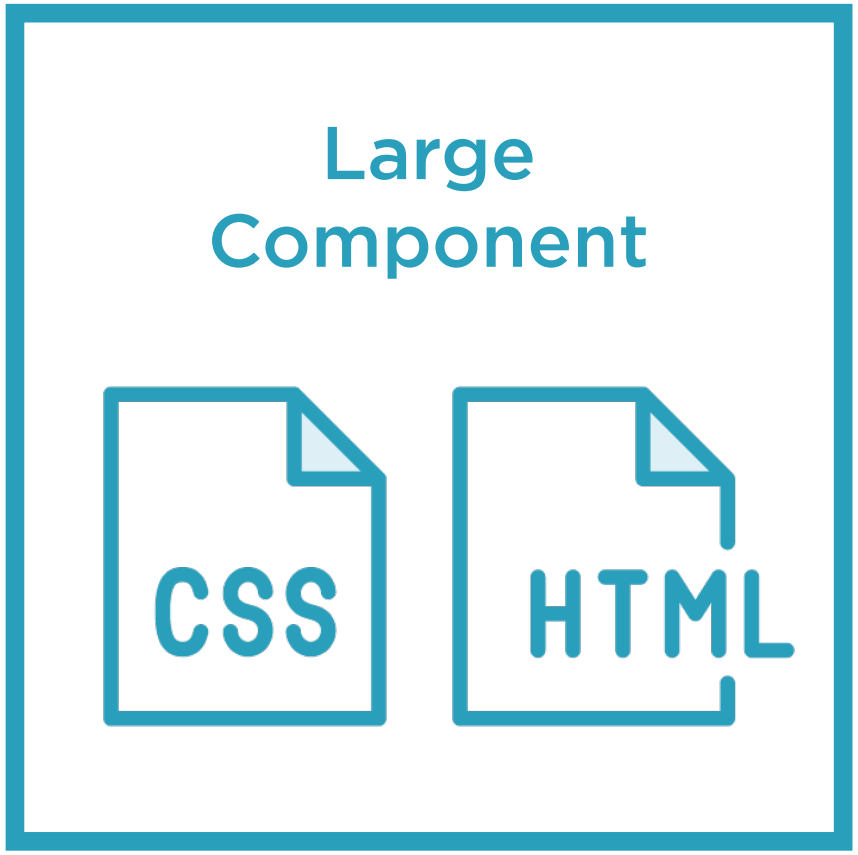


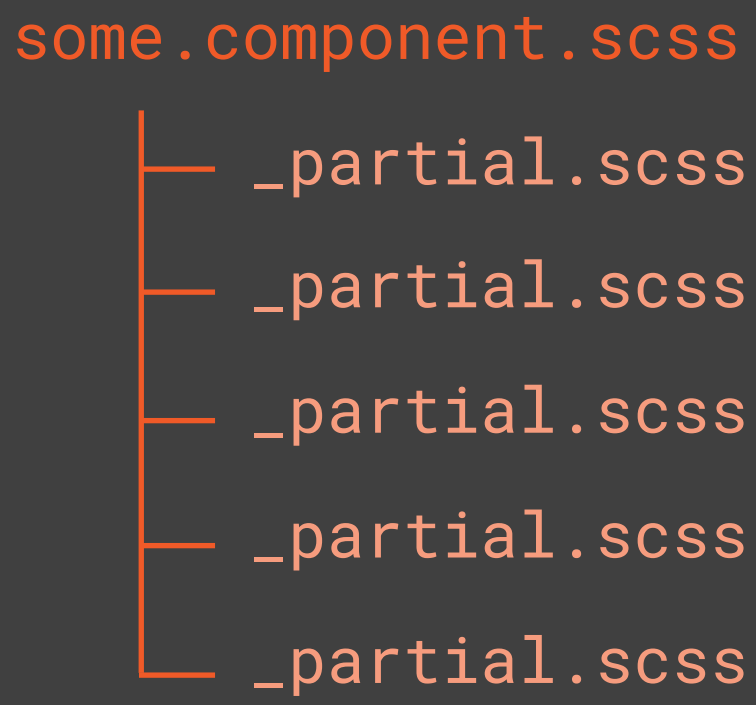


# We Can Do This in Angular!

# 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 @imports

```
@import ‘scss/globals’;
@import ‘scss/header’;
@import ‘scss/content’;
@import ‘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**

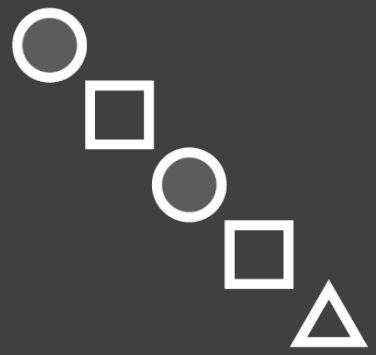
**Use SASS Imports**

# Component Styles

# Local Mixins & Variables

Global Styles


Mixins & Variables



Local Styles


Mixins & Variables

# What Can We Do?


Create Local Mixins & 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;
```

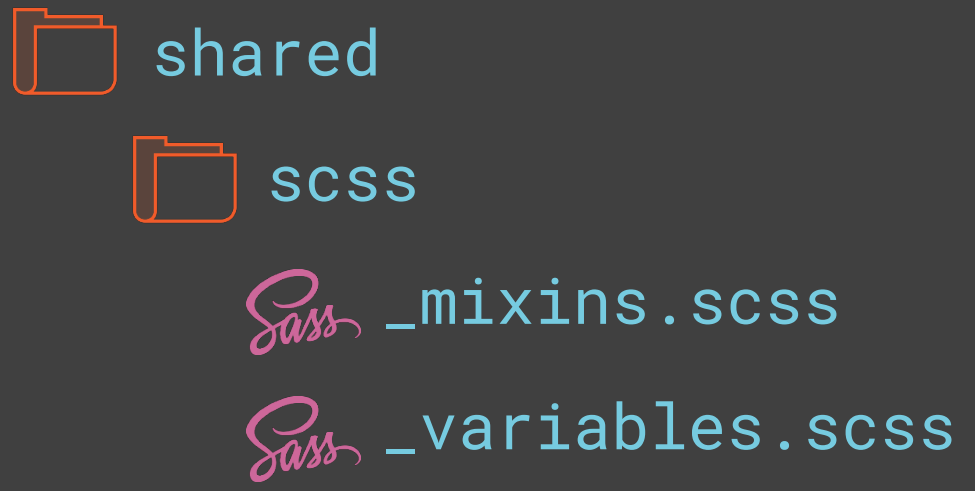
# Mixins

On a File Level

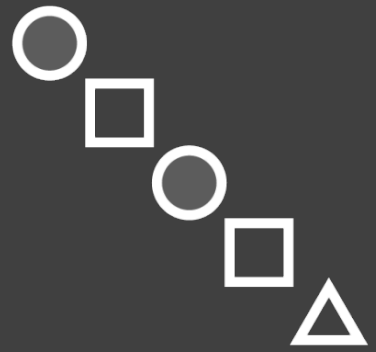
```
// Mixins
@mixin component__box {
    border: solid 0.2em #ccc;
    margin: 2em;
    padding: 1em;
}
```

# Mixins & Variables

Shared Across Components

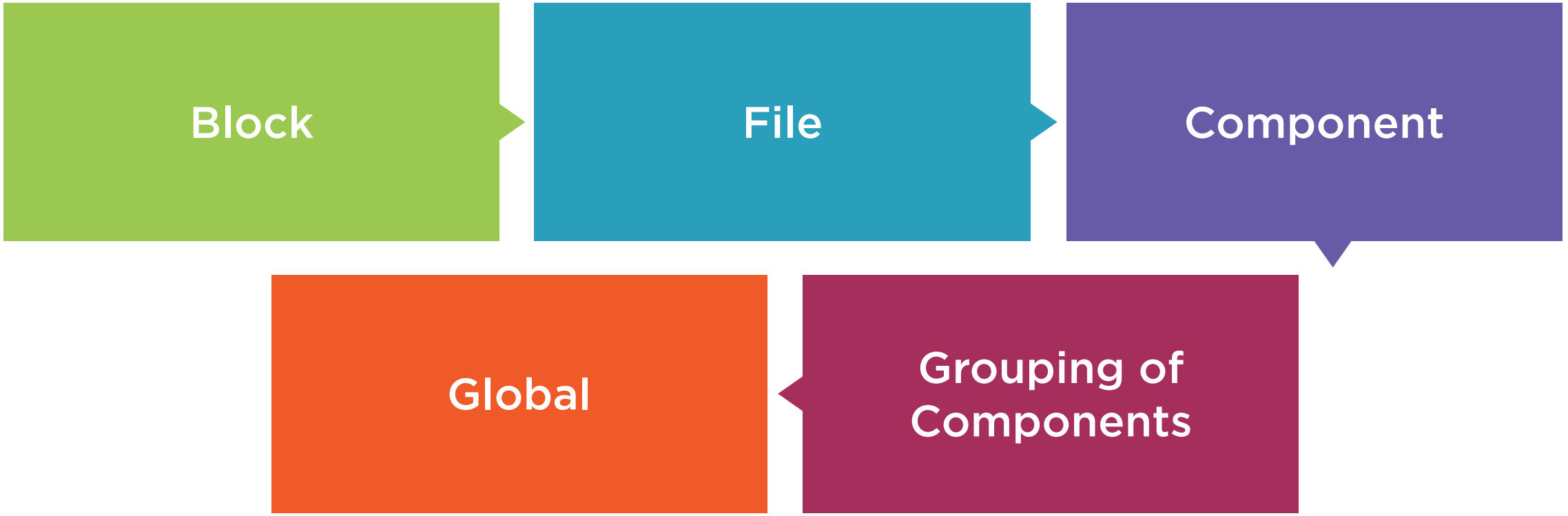



Mixins & Variables



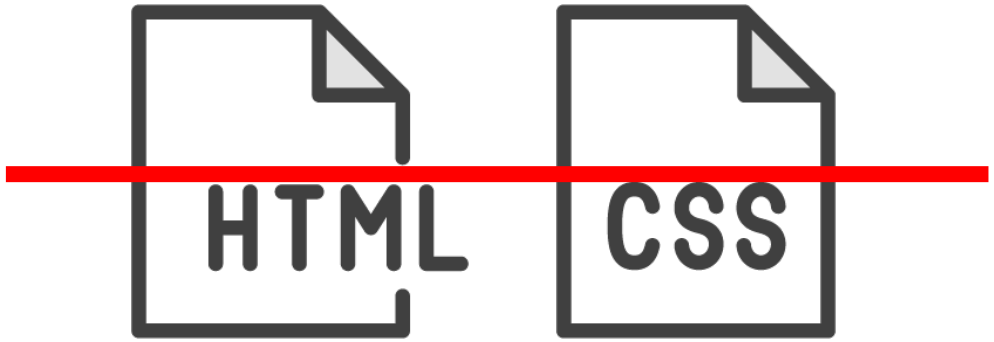
As Local as Possible

# Mixins & Variables



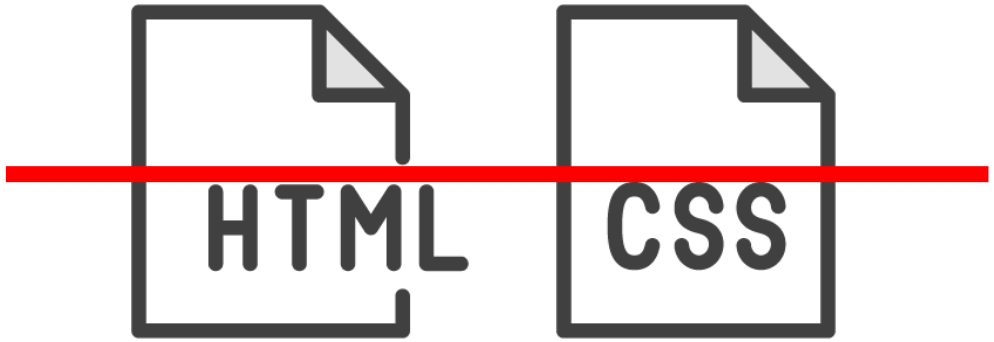
# Styling the Pre-boostrap Loading Screen

# Add Style & 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, & Maintain Styles
- Use SASS to Create Better Code
- Global Styles as a Global Stylesheet
- Global Styles as Mixins & 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 & Simple App Loading Screen

# Component Theming