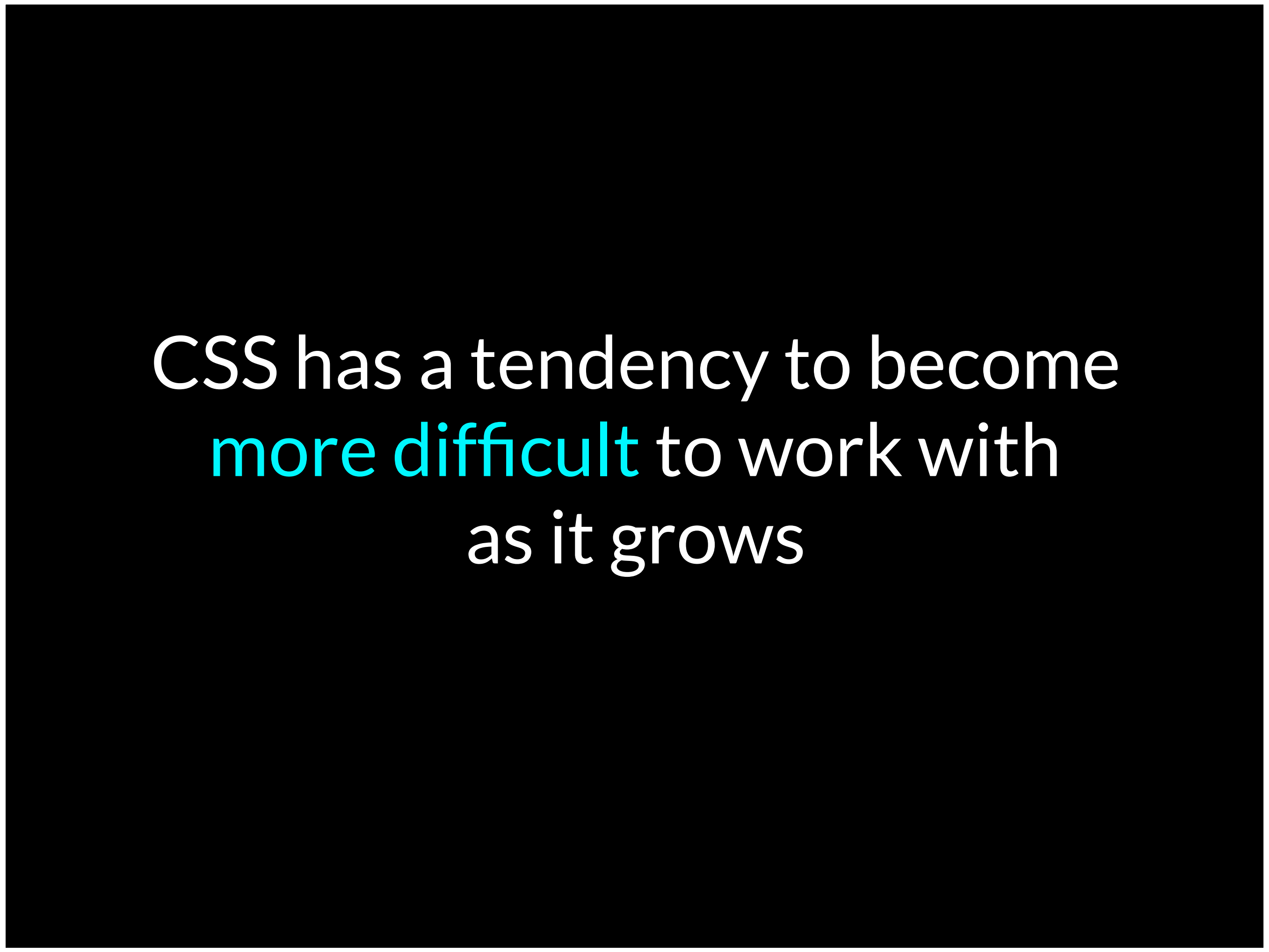
Solving CSS at scale with your own UI framework

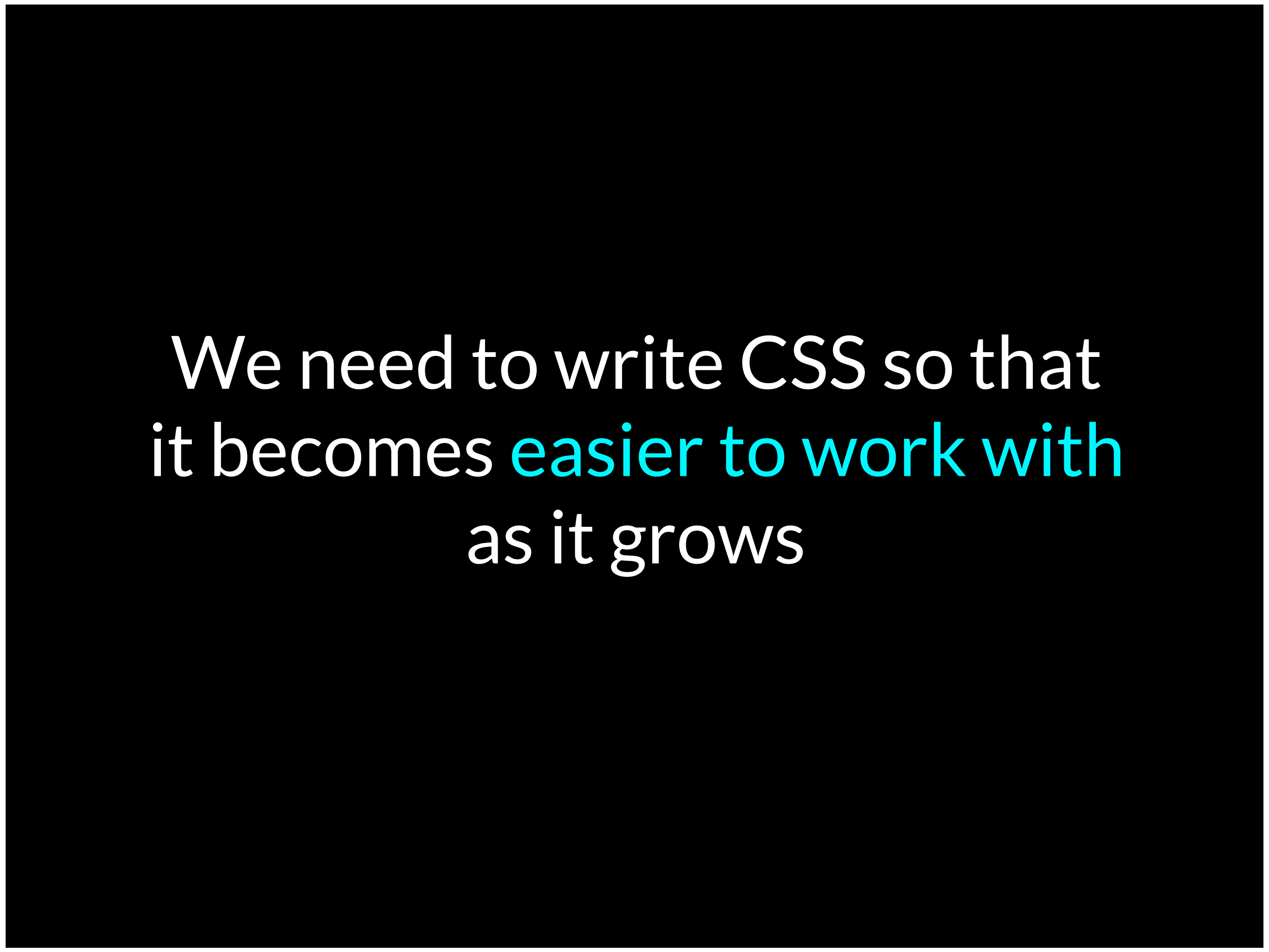
CJ Cenizal, UI Engineer @ Elastic

* Hi, I'm CJ.
* I'm here to talk about Solving CSS at scale with your own UI framework.
* When I've told people about this talk, they've asked me, "What does that mean? Solving CSS at scale?"



CSS has a tendency to become more difficult to work with as it grows

- CSS has a tendency to become MORE DIFFICULT to WORK WITH as it grows
-



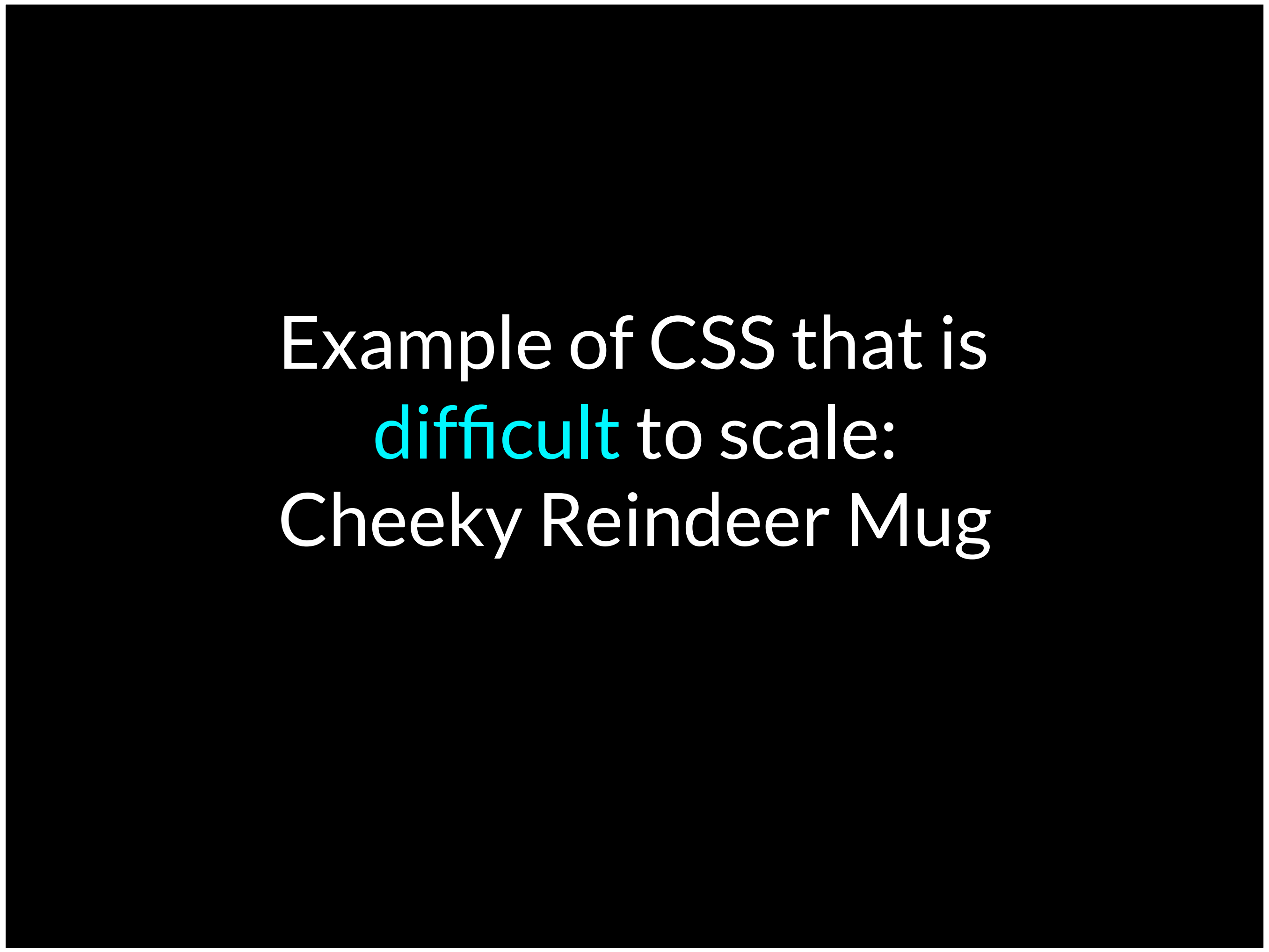
We need to write CSS so that
it becomes easier to work with
as it grows

* Solving CSS at scale means to WRITE IN A WAY so that as it grows, it becomes EASIER TO WORK WITH.

1. Examples of CSS that is difficult to scale

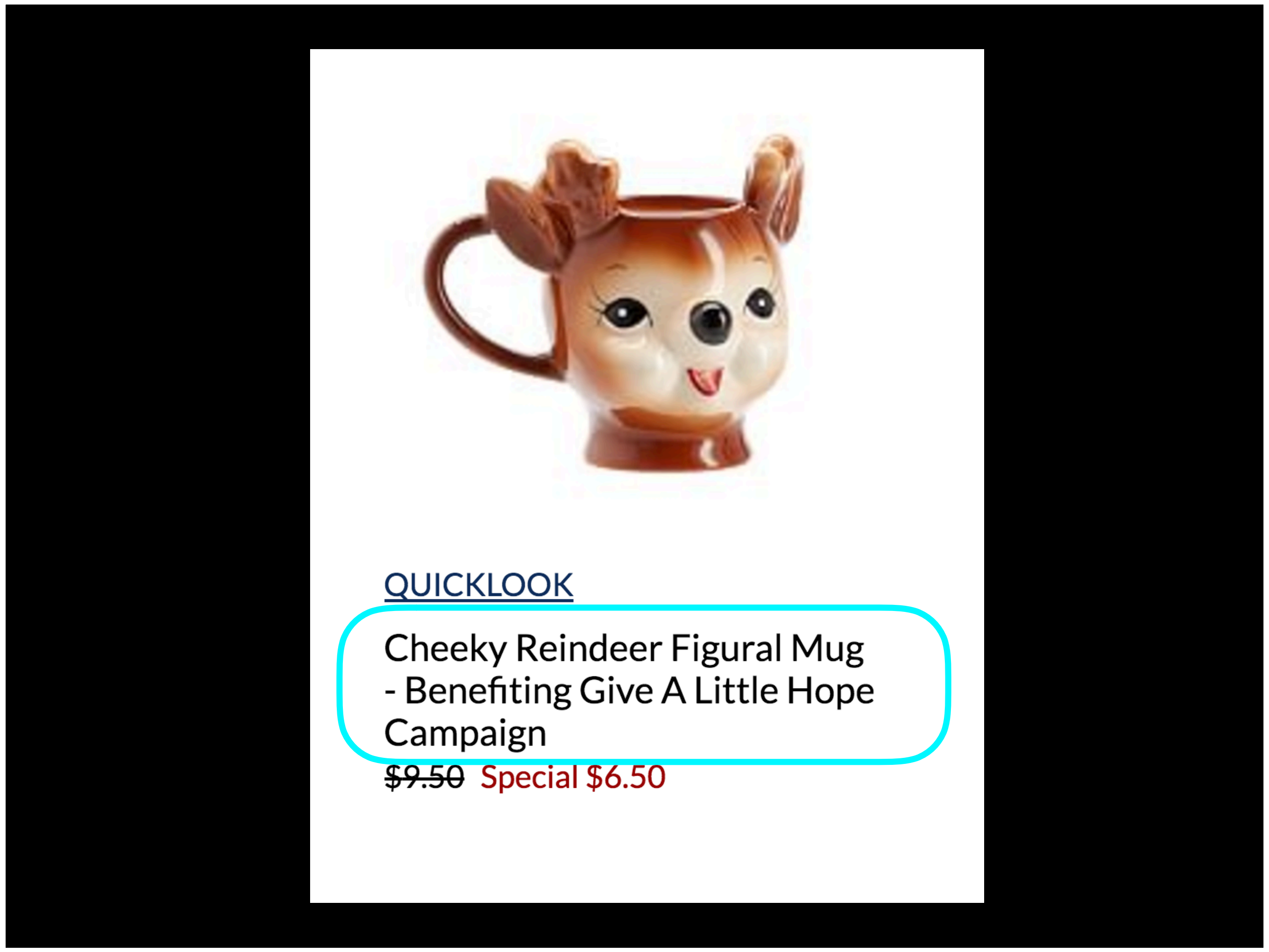
2. How to write scalable CSS

* There are TWO MAJOR PARTS to this TALK



Example of CSS that is difficult to scale: Cheeky Reindeer Mug

* Let’s start by going over an example of CSS that’s difficult to scale

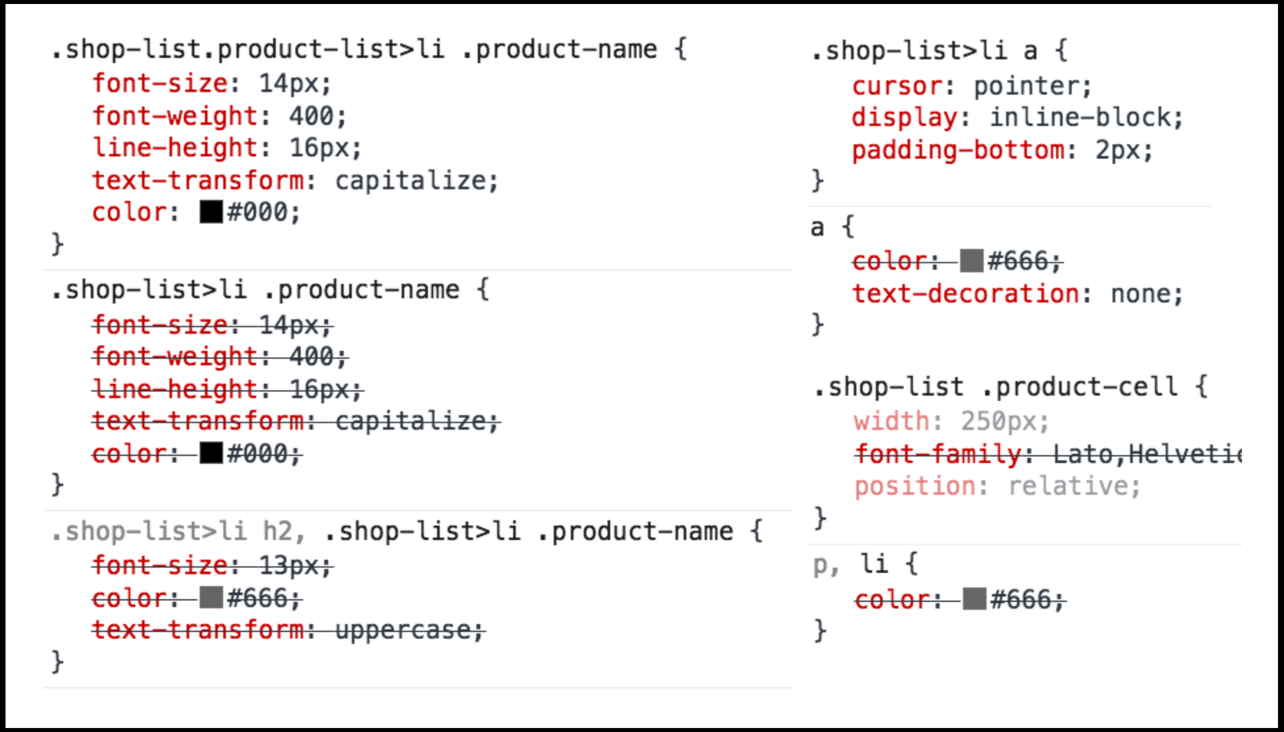


QUICKLOOK

Cheeky Reindeer Figural Mug - Benefiting Give A Little Hope Campaign

~~$9.50~~ Special $6.50

* This is from a popular retail site online. They have 13,000 LINES OF CSS.
* I found this really cool mug and wanted to SEE how they were USING CSS to style the DESCRIPTIVE TEXT of the mug.
* So I right-clicked on it and INSPECTED it in the WEB INSPECTOR.

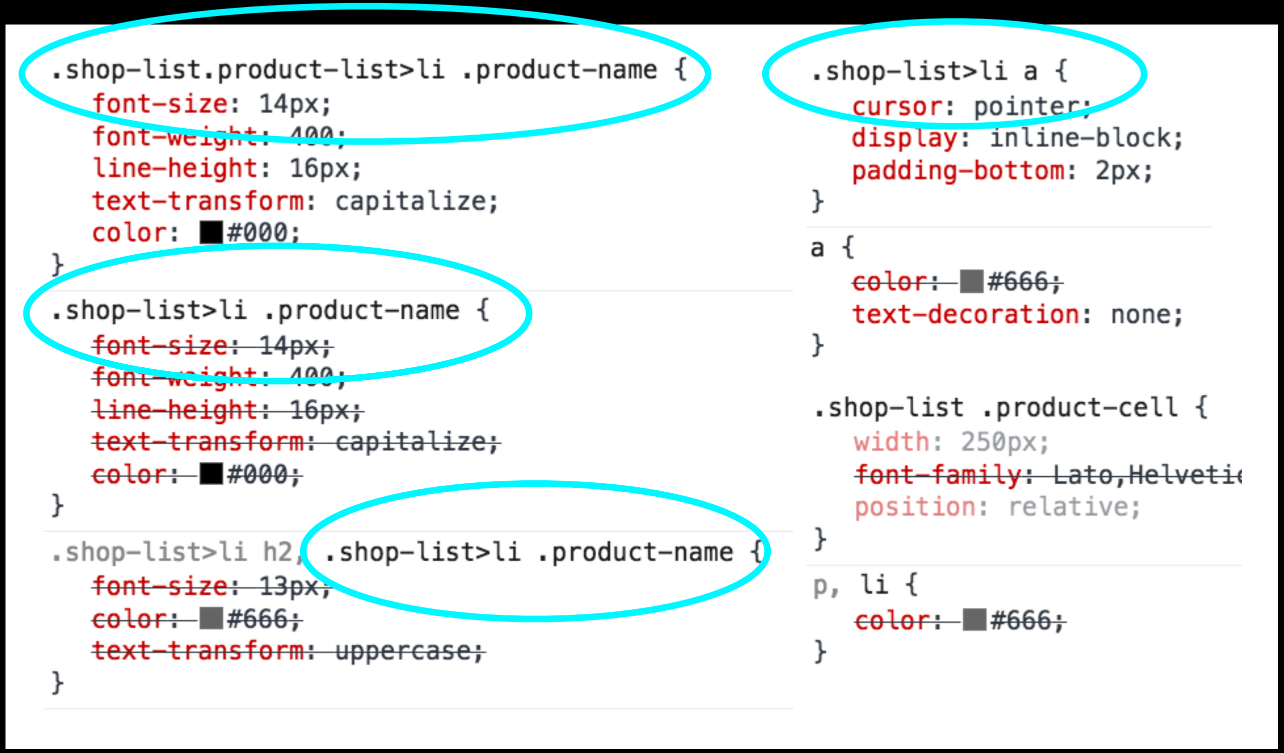


```
.shop-list.product-list>li .product-name {
   font-size: 14px;
   font-weight: 400;
   line-height: 16px;
   text-transform: capitalize;
   color: ■#000;
}
.shop-list>li .product-name {
   ~~font-size: 14px;~~
   ~~font-weight: 400;~~
   ~~line-height: 16px;~~
   ~~text-transform: capitalize;~~
   ~~color: ■#000;~~
}
.shop-list>li h2, .shop-list>li .product-name {
   ~~font-size: 13px;~~
   ~~color: ■#666;~~
   ~~text-transform: uppercase;~~
}
```

```
.shop-list>li a {
   cursor: pointer;
   display: inline-block;
   padding-bottom: 2px;
}
a {
   ~~color: ■#666;~~
   text-decoration: none;
}
.shop-list .product-cell {
   width: 250px;
   ~~font-family: Lato,Helveti~~
   position: relative;
}
p, li {
   ~~color: ■#666;~~
}
```

Let us dissect this.

- THIS is what I got.
- There’s A LOT GOING ON.
- Let’s DISSECT WHAT’S GOING ON HERE…

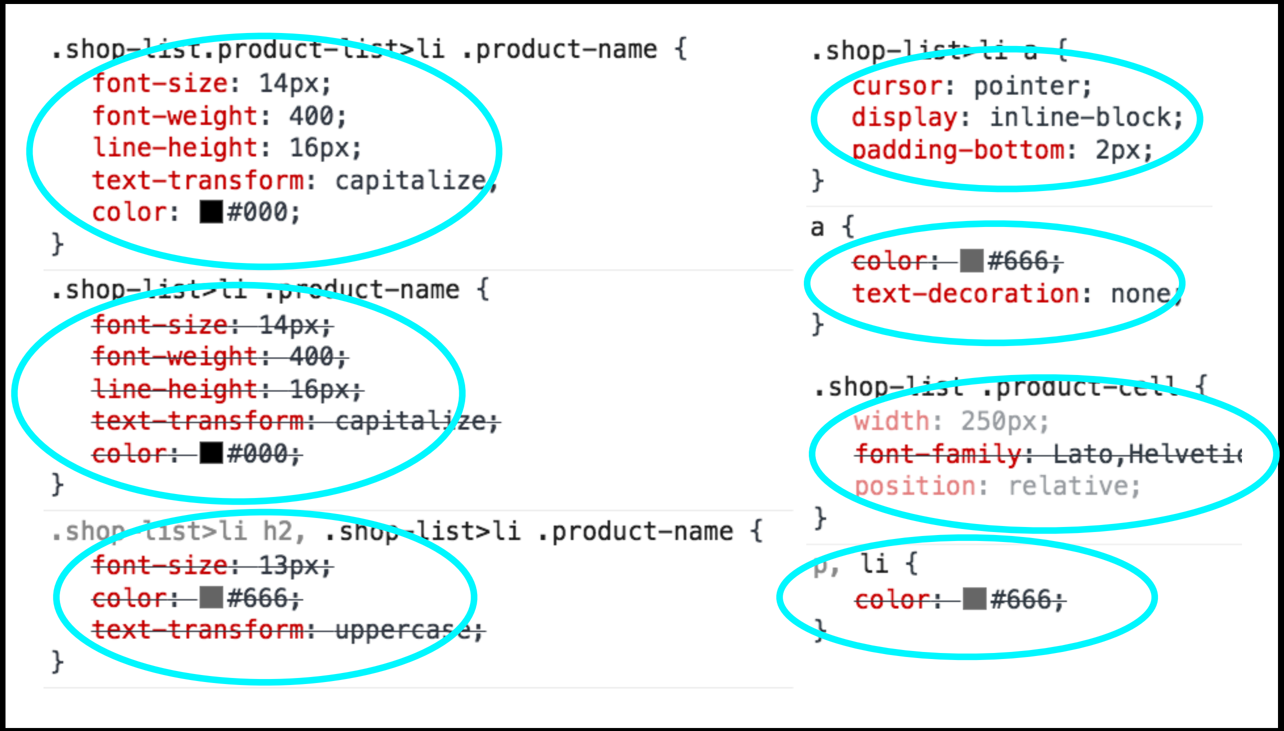


```
.shop-list.product-list>li .product-name {
    font-size: 14px;
    ~~font-weight: 400;~~
    line-height: 16px;
    text-transform: capitalize;
    color: ■#000;
}
.shop-list>li .product-name {
    ~~font-size: 14px;~~
    ~~font-weight: 400;~~
    ~~line-height: 16px;~~
    ~~text-transform: capitalize;~~
    ~~color: ■#000;~~
}
.shop-list>li h2, .shop-list>li .product-name {
    ~~font-size: 13px;~~
    ~~color: ■#666;~~
    ~~text-transform: uppercase;~~
}
```

```
.shop-list>li a {
    cursor: pointer;
    display: inline-block;
    padding-bottom: 2px;
}
a {
    ~~color: ■#666;~~
    text-decoration: none;
}
.shop-list .product-cell {
    width: 250px;
    ~~font-family: Lato,Helveti~~
    position: relative;
}
p, li {
    ~~color: ■#666;~~
}
```

Long selectors are hard to understand

* THE FIRST THING I NOTICE IS THE VERY LONG, COMPLEX NAMES.
* As a developer looking at this code for the first time,
* IT'S HARD TO TELL FROM THE NAME ALONE: what they MEAN, what they're DOING
* Let's look at what these selectors are DOING…

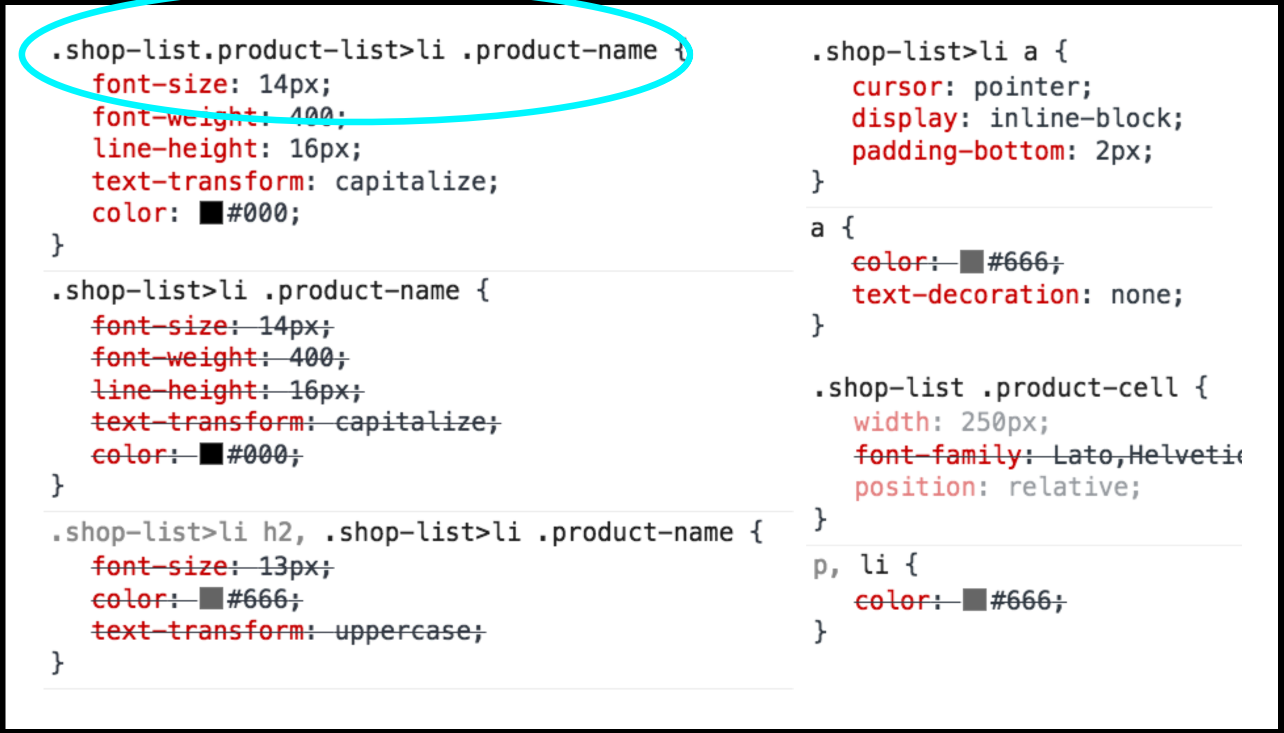


```
.shop-list.product-list>li .product-name {
    font-size: 14px;
    font-weight: 400;
    line-height: 16px;
    text-transform: capitalize;
    color: ■#000;
}
.shop-list>li .product-name {
    ~~font-size: 14px;~~
    ~~font-weight: 400;~~
    ~~line-height: 16px;~~
    ~~text-transform: capitalize;~~
    ~~color: ■#000;~~
}
.shop-list>li h2, .shop-list>li .product-name {
    ~~font-size: 13px;~~
    ~~color: ■#666;~~
    ~~text-transform: uppercase;~~
}
```

```
.shop-list>li a {
    cursor: pointer;
    display: inline-block;
    padding-bottom: 2px;
}
a {
    ~~color: ■#666;~~
    text-decoration: none;
}
.shop-list .product-cell {
    width: 250px;
    ~~font-family: Lato,Helveti~~
    position: relative;
}
p, li {
    ~~color: ■#666;~~
}
```

Many selectors fighting for control

* We have MANY SELECTORS interacting, affecting just this ONE ELEMENT
* FIGHTING OVER who gets to DICTATE the appearance of this element.
* HARD TO TELL WHICH SELECTOR IS RESPONSIBLE FOR WHAT.
* If I had to MAKE A CHANGE, I wouldn't BE SURE OF WHERE TO DO IT.

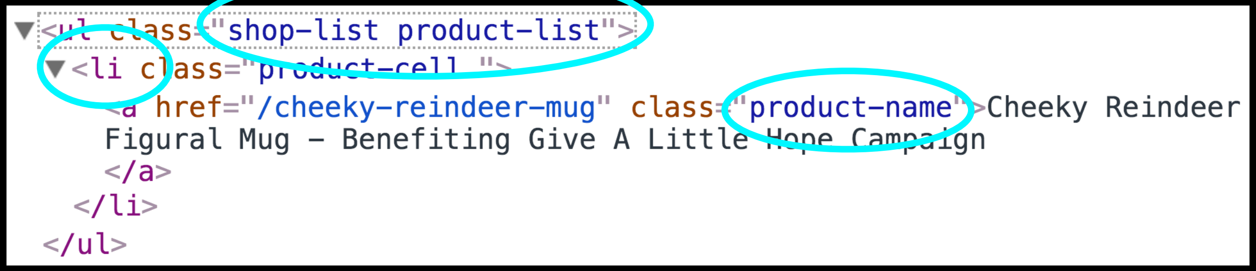


```
.shop-list.product-list>li .product-name {
   font-size: 14px;
   ~~font-weight: 400;~~
   line-height: 16px;
   text-transform: capitalize;
   color: ■#000;
}
.shop-list>li .product-name {
   ~~font-size: 14px;~~
   ~~font-weight: 400;~~
   ~~line-height: 16px;~~
   ~~text-transform: capitalize;~~
   ~~color: ■#000;~~
}
.shop-list>li h2, .shop-list>li .product-name {
   ~~font-size: 13px;~~
   ~~color: ■#666;~~
   ~~text-transform: uppercase;~~
}
```

```
.shop-list>li a {
   cursor: pointer;
   display: inline-block;
   padding-bottom: 2px;
}
a {
   ~~color: ■#666;~~
   text-decoration: none;
}
.shop-list .product-cell {
   width: 250px;
   ~~font-family: Lato,Helveti~~
   position: relative;
}
p, li {
   ~~color: ■#666;~~
}
```

Tightly coupled selectors make CSS hard to visualize

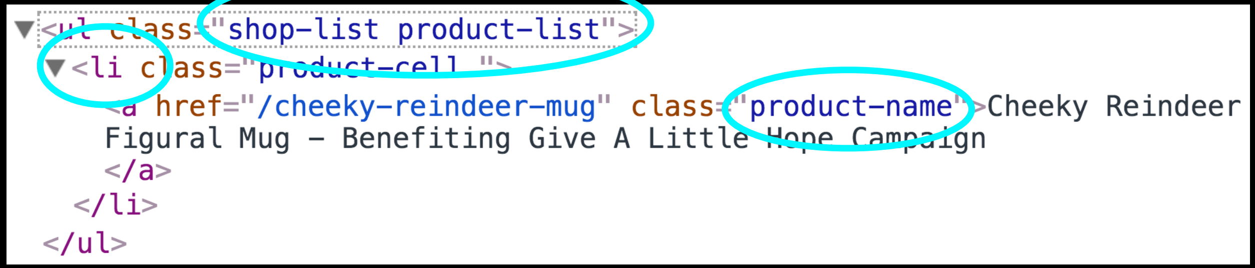
* Let’s go back to the WAY the selectors are NAMED.
* Let’s take a CLOSER LOOK at the FIRST SELECTOR.
* THE COMPLEXITY of this selector TELLS ME THAT it’s VERY TIGHTLY COUPLED with the STRUCTURE OF THE MARKUP.
* HARD to visualize HOW it will look in the browser. I have to go look at the MARKUP



```
▼<ul class="shop-list product-list">
  ▼<li class="product-cell ">
      <a href="/cheeky-reindeer-mug" class="product-name">Cheeky Reindeer
      Figural Mug - Benefiting Give A Little Hope Campaign
      </a>
    </li>
  </ul>
```

Tightly coupled selectors make markup hard to visualize, too

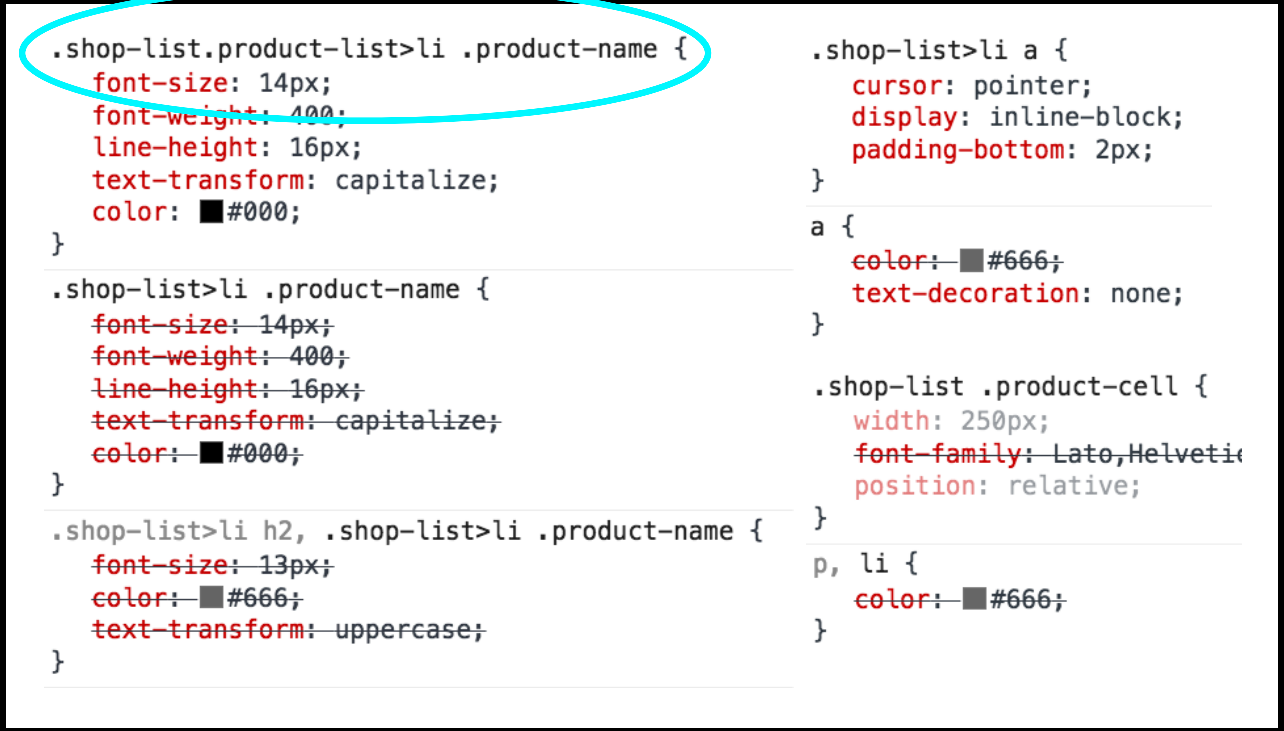
* IRONICALLY, the markup is hard to visualize TOO.
* This complex selector is DEFINED by ALL OF THESE different parts of the markup.
* It's really hard to LOOK at the MARKUP and IDENTIFY this SELECTOR.
* Without KNOWING which SELECTORS are in play, how can you tell how this MARKUP will appear in the BROWSER?



```
▼<ul class="shop-list product-list">
  ▼<li class="product-cell ">
      <a href="/cheeky-reindeer-mug" class="product-name">Cheeky Reindeer
      Figural Mug - Benefiting Give A Little Hope Campaign
      </a>
    </li>
  </ul>
```

Tightly coupled selectors create brittle relationships

* Look at these different DISCONNECTED parts of the markup.
* WE HAVE TO MAINTAIN THIS VERY STRICT, BRITTLE STRUCTURE FOR OUR CSS TO FUNCTION.
* This makes it HARD to make CHANGES to JUST the CSS or JUST the MARKUP.
* Any change to ONE will have an EFFECT UPON THE OTHER.
* So you have to VERY CONSCIOUS of BOTH CSS and the MARKUP when making any changes.



```
.shop-list.product-list>li .product-name {
   font-size: 14px;
   ~~font-weight: 400;~~
   line-height: 16px;
   text-transform: capitalize;
   color: ■#000;
}
.shop-list>li .product-name {
   ~~font-size: 14px;~~
   ~~font-weight: 400;~~
   ~~line-height: 16px;~~
   ~~text-transform: capitalize;~~
   ~~color: ■#000;~~
}
.shop-list>li h2, .shop-list>li .product-name {
   ~~font-size: 13px;~~
   ~~color: ■#666;~~
   ~~text-transform: uppercase;~~
}
```

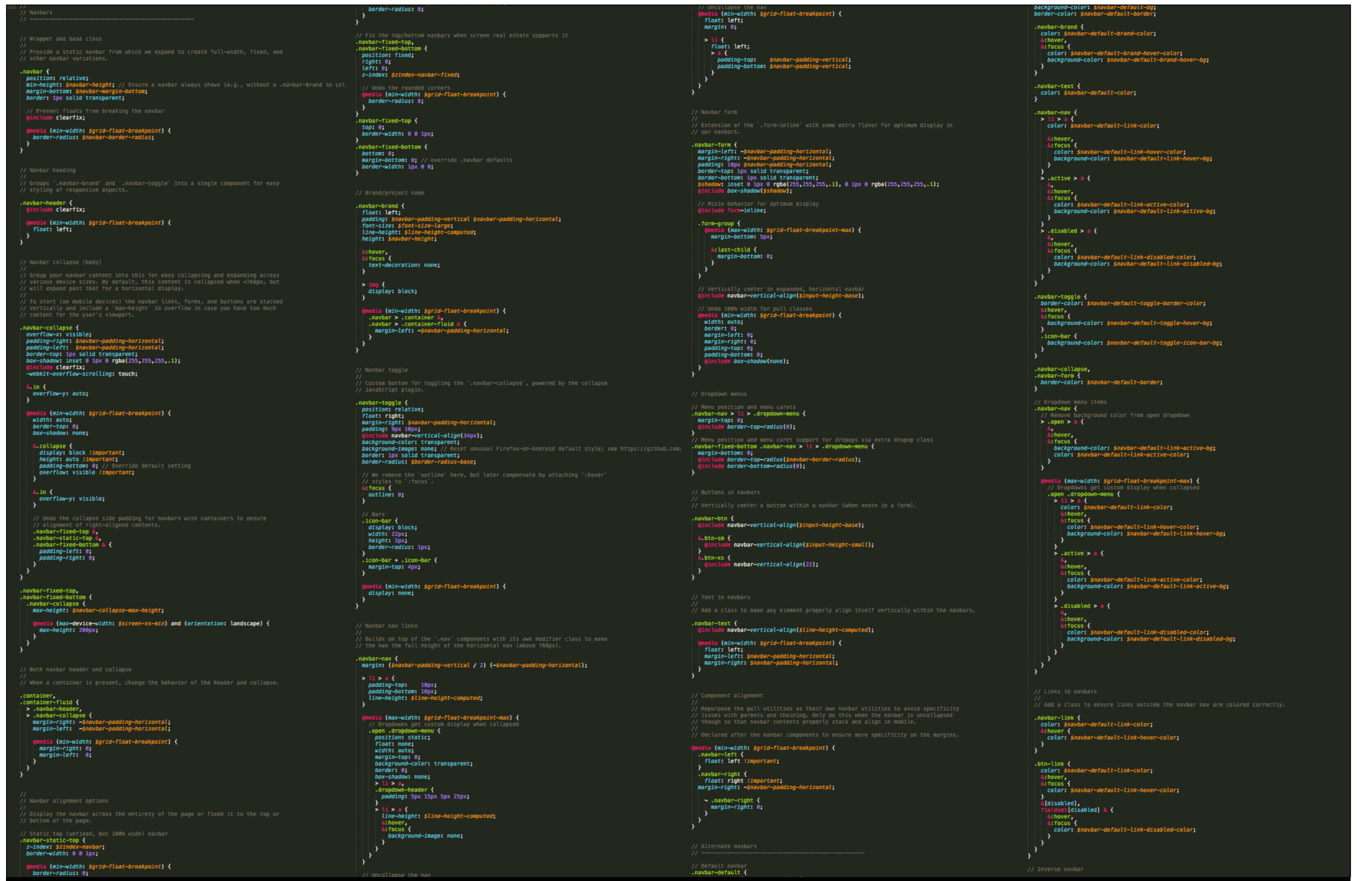
```
.shop-list>li a {
   cursor: pointer;
   display: inline-block;
   padding-bottom: 2px;
}
a {
   ~~color: ■#666;~~
   text-decoration: none;
}
.shop-list .product-cell {
   width: 250px;
   ~~font-family: Lato,Helveti~~
   position: relative;
}
p, li {
   ~~color: ■#666;~~
}
```

Complex selectors make it easy to accidentally break the UI

- COMPLEX INTERACTING SELECTORS it REALLY EASY to ACCIDENTALLY BREAK THINGS.
- We saw how hard it is to identify this kinds of selectors in the MARKUP.
- Has this ever happened to you? :D

Example of CSS that is difficult to scale: Bootstrap

* I love Bootstrap, I’m sure we all have some fond memory of using it.
* We build an MVP, but…
* It DOESN’T SCALE WELL!



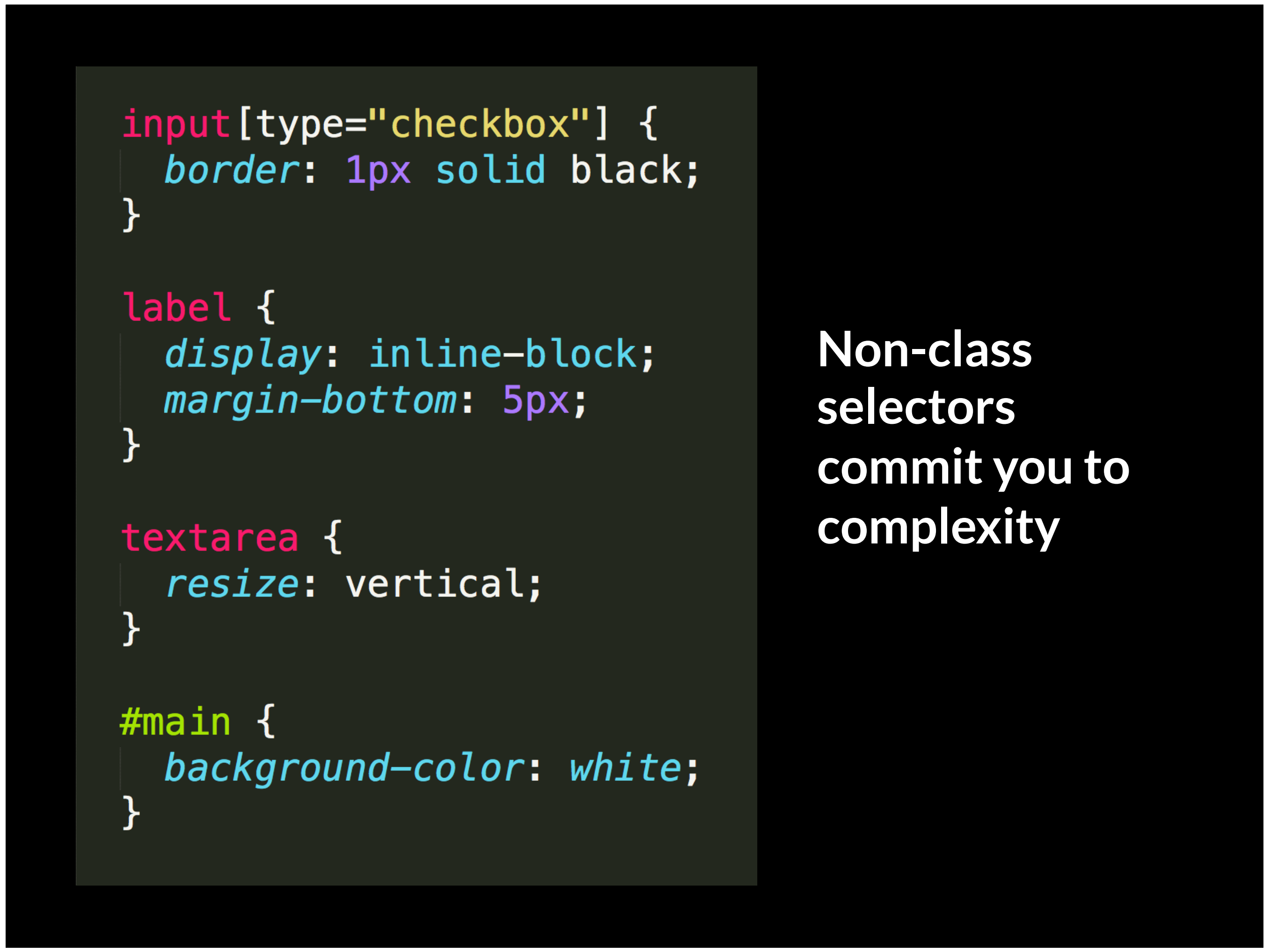
```scss
//
// Navbars
// --------------------------------------------------


// Wrapper and base class
//
// Provide a static navbar from which we expand to create full-width, fixed, and
// other navbar variations.

.navbar {
  position: relative;
  min-height: $navbar-height; // Ensure a navbar always shows (e.g., without a .navbar-brand in col
  margin-bottom: $navbar-margin-bottom;
  border: 1px solid transparent;

  // Prevent floats from breaking the navbar
  @include clearfix;

  @media (min-width: $grid-float-breakpoint) {
    border-radius: $navbar-border-radius;
  }
}


// Navbar heading
//
// Groups `.navbar-brand` and `.navbar-toggle` into a single component for easy
// styling of responsive aspects.

.navbar-header {
  @include clearfix;

  @media (min-width: $grid-float-breakpoint) {
    float: left;
  }
}


// Navbar collapse (body)
//
// Group your navbar content into this for easy collapsing and expanding across
// various device sizes. By default, this content is collapsed when <768px, but
// will expand past that for a horizontal display.
//
// To start (on mobile devices) the navbar links, forms, and buttons are stacked
// vertically and include a `max-height` to overflow in case you have too much
// content for the user's viewport.

.navbar-collapse {
  overflow-x: visible;
  padding-right: $navbar-padding-horizontal;
  padding-left:  $navbar-padding-horizontal;
  border-top: 1px solid transparent;
  box-shadow: inset 0 1px 0 rgba(255,255,255,.1);
  @include clearfix;
  -webkit-overflow-scrolling: touch;

  &.in {
    overflow-y: auto;
  }

  @media (min-width: $grid-float-breakpoint) {
    width: auto;
    border-top: 0;
    box-shadow: none;

    &.collapse {
      display: block !important;
      height: auto !important;
      padding-bottom: 0; // Override default setting
      overflow: visible !important;
    }

    &.in {
      overflow-y: visible;
    }

    // Undo the collapse side padding for navbars with containers to ensure
    // alignment of right-aligned contents.
    .navbar-fixed-top &,
    .navbar-static-top &,
    .navbar-fixed-bottom & {
      padding-left: 0;
      padding-right: 0;
    }
  }
}

.navbar-fixed-top,
.navbar-fixed-bottom {
  .navbar-collapse {
    max-height: $navbar-collapse-max-height;

    @media (max-device-width: $screen-xs-min) and (orientation: landscape) {
      max-height: 200px;
    }
  }
}


// Both navbar header and collapse
//
// When a container is present, change the behavior of the header and collapse.

.container,
.container-fluid {
  > .navbar-header,
  > .navbar-collapse {
    margin-right: -$navbar-padding-horizontal;
    margin-left:  -$navbar-padding-horizontal;

    @media (min-width: $grid-float-breakpoint) {
      margin-right: 0;
      margin-left:  0;
    }
  }
}


//
// Navbar alignment options
//
// Display the navbar across the entirety of the page or fixed it to the top or
// bottom of the page.

// Static top (unfixed, but 100% wide) navbar
.navbar-static-top {
  z-index: $zindex-navbar;
  border-width: 0 0 1px;

  @media (min-width: $grid-float-breakpoint) {
    border-radius: 0;
  }
}

// Fix the top/bottom navbars when screen real estate supports it
.navbar-fixed-top,
.navbar-fixed-bottom {
  position: fixed;
  right: 0;
  left: 0;
  z-index: $zindex-navbar-fixed;

  // Undo the rounded corners
  @media (min-width: $grid-float-breakpoint) {
    border-radius: 0;
  }
}
.navbar-fixed-top {
  top: 0;
  border-width: 0 0 1px;
}
.navbar-fixed-bottom {
  bottom: 0;
  margin-bottom: 0; // override .navbar defaults
  border-width: 1px 0 0;
}


// Brand/project name

.navbar-brand {
  float: left;
  padding: $navbar-padding-vertical $navbar-padding-horizontal;
  font-size: $font-size-large;
  line-height: $line-height-computed;
  height: $navbar-height;

  &:hover,
  &:focus {
    text-decoration: none;
  }

  > img {
    display: block;
  }

  @media (min-width: $grid-float-breakpoint) {
    .navbar > .container &,
    .navbar > .container-fluid & {
      margin-left: -$navbar-padding-horizontal;
    }
  }
}


// Navbar toggle
//
// Custom button for toggling the `.navbar-collapse`, powered by the collapse
// JavaScript plugin.

.navbar-toggle {
  position: relative;
  float: right;
  margin-right: $navbar-padding-horizontal;
  padding: 9px 10px;
  @include navbar-vertical-align(34px);
  background-color: transparent;
  background-image: none; // Reset unusual Firefox-on-Android default style; see https://github.com/
  border: 1px solid transparent;
  border-radius: $border-radius-base;

  // We remove the `outline` here, but later compensate by attaching `:hover`
  // styles to `:focus`.
  &:focus {
    outline: 0;
  }

  // Bars
  .icon-bar {
    display: block;
    width: 22px;
    height: 2px;
    border-radius: 1px;
  }
  .icon-bar + .icon-bar {
    margin-top: 4px;
  }

  @media (min-width: $grid-float-breakpoint) {
    display: none;
  }
}


// Navbar nav links
//
// Builds on top of the `.nav` components with its own modifier class to make
// the nav the full height of the horizontal nav (above 768px).

.navbar-nav {
  margin: ($navbar-padding-vertical / 2) (-$navbar-padding-horizontal);

  > li > a {
    padding-top:    10px;
    padding-bottom: 10px;
    line-height: $line-height-computed;
  }

  @media (max-width: $grid-float-breakpoint-max) {
    // Dropdowns get custom display when collapsed
    .open .dropdown-menu {
      position: static;
      float: none;
      width: auto;
      margin-top: 0;
      background-color: transparent;
      border: 0;
      box-shadow: none;
      > li > a,
      .dropdown-header {
        padding: 5px 15px 5px 25px;
      }
      > li > a {
        line-height: $line-height-computed;
        &:hover,
        &:focus {
          background-image: none;
        }
      }
    }
  }

  // Uncollapse the nav
  @media (min-width: $grid-float-breakpoint) {
    float: left;
    margin: 0;

    > li {
      float: left;
      > a {
        padding-top:    $navbar-padding-vertical;
        padding-bottom: $navbar-padding-vertical;
      }
    }
  }
}


// Navbar form
//
// Extension of the `.form-inline` with some extra flavor for optimum display in
// our navbars.

.navbar-form {
  margin-left: -$navbar-padding-horizontal;
  margin-right: -$navbar-padding-horizontal;
  padding: 10px $navbar-padding-horizontal;
  border-top: 1px solid transparent;
  border-bottom: 1px solid transparent;
  $shadow: inset 0 1px 0 rgba(255,255,255,.1), 0 1px 0 rgba(255,255,255,.1);
  @include box-shadow($shadow);

  // Mixin behavior for optimum display
  @include form-inline;

  .form-group {
    @media (max-width: $grid-float-breakpoint-max) {
      margin-bottom: 5px;

      &:last-child {
        margin-bottom: 0;
      }
    }
  }

  // Vertically center in expanded, horizontal navbar
  @include navbar-vertical-align($input-height-base);

  // Undo 100% width for pull classes
  @media (min-width: $grid-float-breakpoint) {
    width: auto;
    border: 0;
    margin-left: 0;
    margin-right: 0;
    padding-top: 0;
    padding-bottom: 0;
    @include box-shadow(none);
  }
}


// Dropdown menus

// Menu position and menu carets
.navbar-nav > li > .dropdown-menu {
  margin-top: 0;
  @include border-top-radius(0);
}
// Menu position and menu caret support for dropups via extra dropup class
.navbar-fixed-bottom .navbar-nav > li > .dropdown-menu {
  margin-bottom: 0;
  @include border-top-radius($navbar-border-radius);
  @include border-bottom-radius(0);
}


// Buttons in navbars
//
// Vertically center a button within a navbar (when *not* in a form).

.navbar-btn {
  @include navbar-vertical-align($input-height-base);

  &.btn-sm {
    @include navbar-vertical-align($input-height-small);
  }
  &.btn-xs {
    @include navbar-vertical-align(22);
  }
}


// Text in navbars
//
// Add a class to make any element properly align itself vertically within the navbars.

.navbar-text {
  @include navbar-vertical-align($line-height-computed);

  @media (min-width: $grid-float-breakpoint) {
    float: left;
    margin-left: $navbar-padding-horizontal;
    margin-right: $navbar-padding-horizontal;
  }
}


// Component alignment
//
// Repurpose the pull utilities as their own navbar utilities to avoid specificity
// issues with parents and chaining. Only do this when the navbar is uncollapsed
// though so that navbar contents properly stack and align in mobile.
//
// Declared after the navbar components to ensure more specificity on the margins.

@media (min-width: $grid-float-breakpoint) {
  .navbar-left {
    float: left !important;
  }
  .navbar-right {
    float: right !important;
  margin-right: -$navbar-padding-horizontal;

    ~ .navbar-right {
      margin-right: 0;
    }
  }
}


// Alternate navbars
// --------------------------------------------------

// Default navbar
.navbar-default {
  background-color: $navbar-default-bg;
  border-color: $navbar-default-border;

  .navbar-brand {
    color: $navbar-default-brand-color;
    &:hover,
    &:focus {
      color: $navbar-default-brand-hover-color;
      background-color: $navbar-default-brand-hover-bg;
    }
  }

  .navbar-text {
    color: $navbar-default-color;
  }

  .navbar-nav {
    > li > a {
      color: $navbar-default-link-color;

      &:hover,
      &:focus {
        color: $navbar-default-link-hover-color;
        background-color: $navbar-default-link-hover-bg;
      }
    }
    > .active > a {
      &,
      &:hover,
      &:focus {
        color: $navbar-default-link-active-color;
        background-color: $navbar-default-link-active-bg;
      }
    }
    > .disabled > a {
      &,
      &:hover,
      &:focus {
        color: $navbar-default-link-disabled-color;
        background-color: $navbar-default-link-disabled-bg;
      }
    }
  }

  .navbar-toggle {
    border-color: $navbar-default-toggle-border-color;
    &:hover,
    &:focus {
      background-color: $navbar-default-toggle-hover-bg;
    }
    .icon-bar {
      background-color: $navbar-default-toggle-icon-bar-bg;
    }
  }

  .navbar-collapse,
  .navbar-form {
    border-color: $navbar-default-border;
  }

  // Dropdown menu items
  .navbar-nav {
    // Remove background color from open dropdown
    > .open > a {
      &,
      &:hover,
      &:focus {
        background-color: $navbar-default-link-active-bg;
        color: $navbar-default-link-active-color;
      }
    }

    @media (max-width: $grid-float-breakpoint-max) {
      // Dropdowns get custom display when collapsed
      .open .dropdown-menu {
        > li > a {
          color: $navbar-default-link-color;
          &:hover,
          &:focus {
            color: $navbar-default-link-hover-color;
            background-color: $navbar-default-link-hover-bg;
          }
        }
        > .active > a {
          &,
          &:hover,
          &:focus {
            color: $navbar-default-link-active-color;
            background-color: $navbar-default-link-active-bg;
          }
        }
        > .disabled > a {
          &,
          &:hover,
          &:focus {
            color: $navbar-default-link-disabled-color;
            background-color: $navbar-default-link-disabled-bg;
          }
        }
      }
    }
  }


  // Links in navbars
  //
  // Add a class to ensure links outside the navbar nav are colored correctly.

  .navbar-link {
    color: $navbar-default-link-color;
    &:hover {
      color: $navbar-default-link-hover-color;
    }
  }

  .btn-link {
    color: $navbar-default-link-color;
    &:hover,
    &:focus {
      color: $navbar-default-link-hover-color;
    }
    &[disabled],
    fieldset[disabled] & {
      &:hover,
      &:focus {
        color: $navbar-default-link-disabled-color;
      }
    }
  }
}

// Inverse navbar
```

Massive files are hard to understand and change

- They have files that are hundreds of lines long.
- This is the HALF of the SCSS file for their NAVBAR. The file is 663 lines.
- If you wanted to make changes to this NAVBAR, it would be hard to find the right lines to make changes in this massive file.
- This kind of file/folder structure is really hard to SCALE.
- So you usually end up adding OVERRIDING styles.

Example of CSS that is difficult to scale:

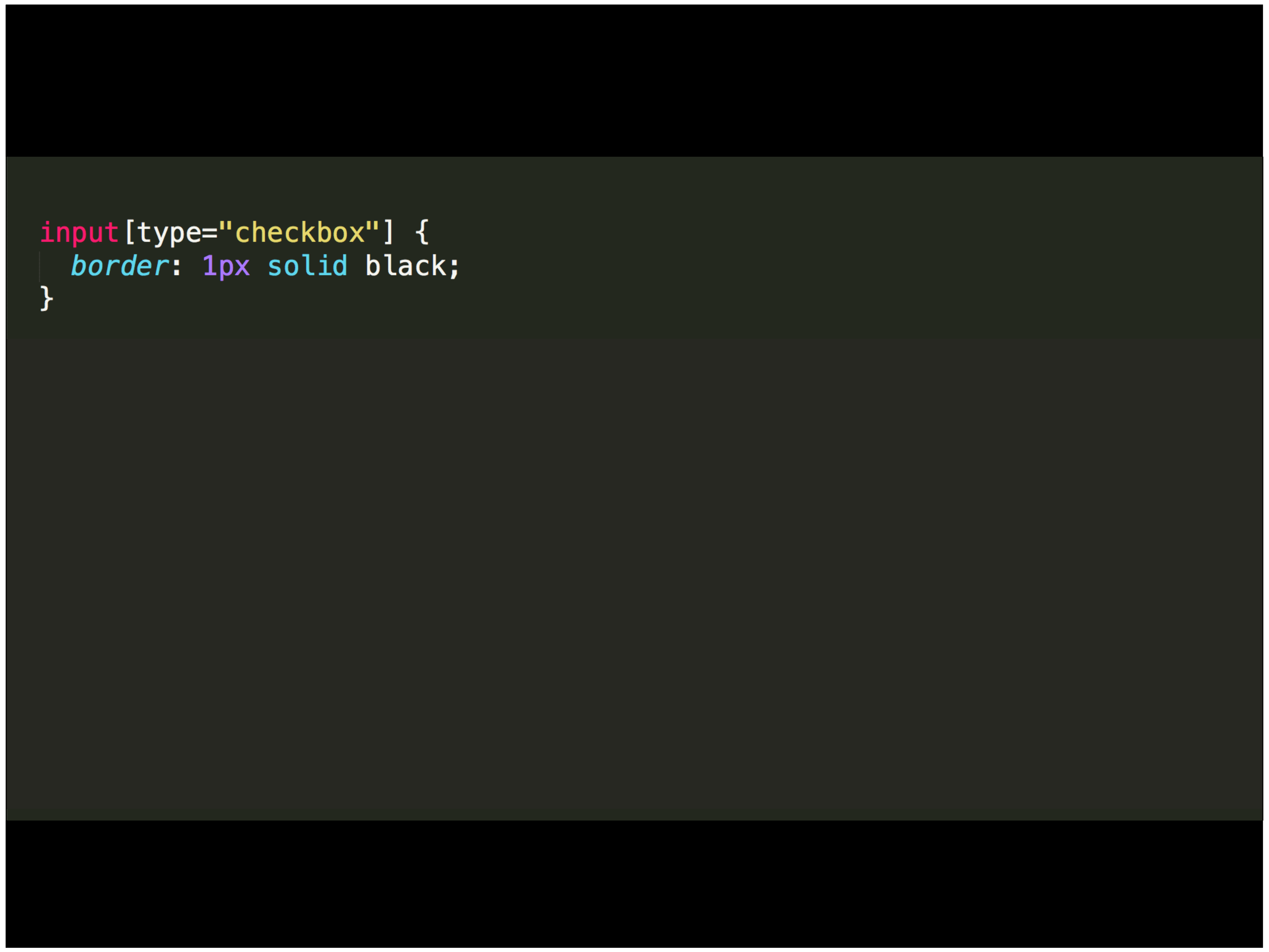
My own



```css
input[type="checkbox"] {
  border: 1px solid black;
}

label {
  display: inline-block;
  margin-bottom: 5px;
}

textarea {
  resize: vertical;
}

#main {
  background-color: white;
}
```

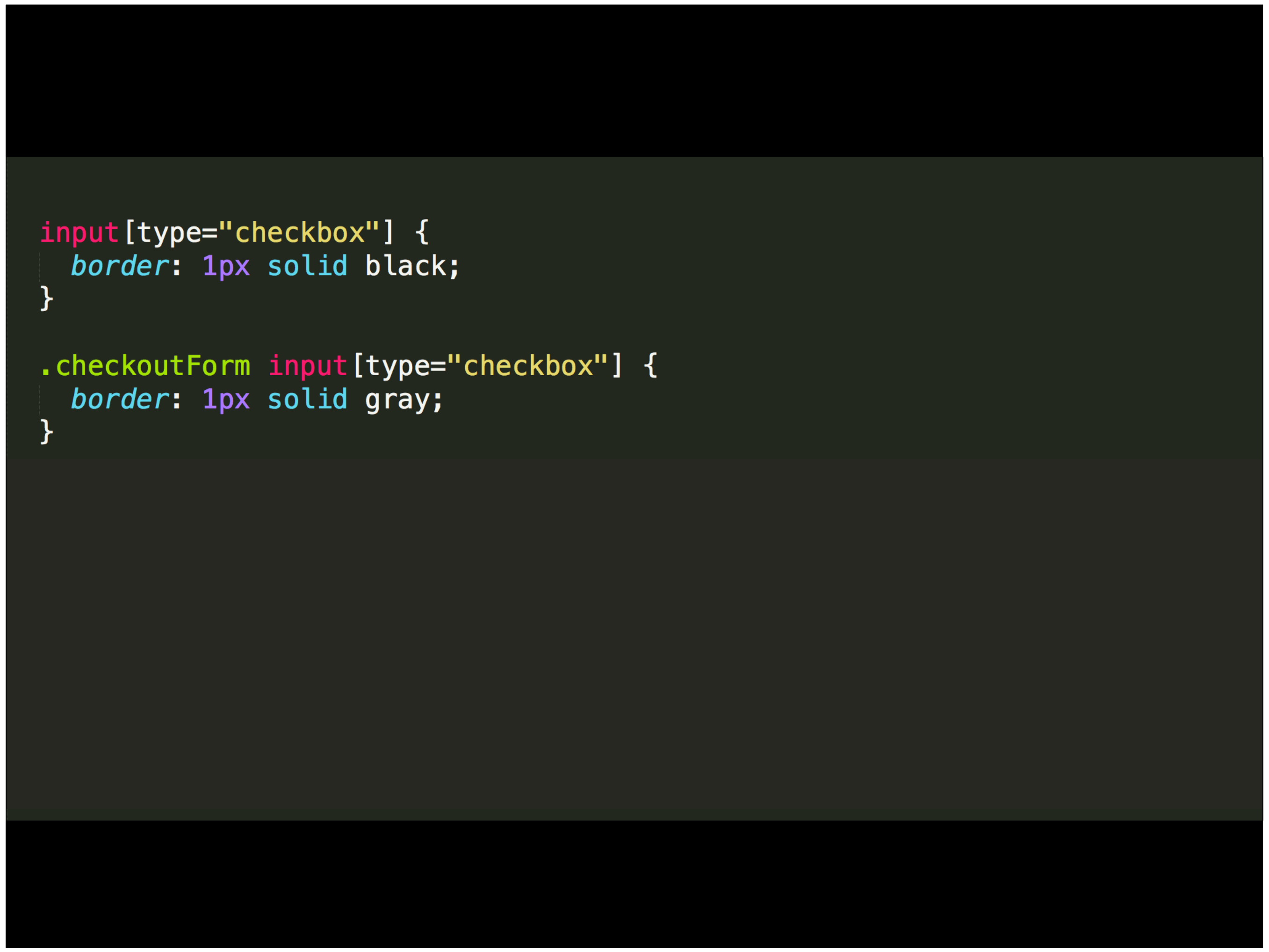
Non-class selectors commit you to complexity

* An IMPORTANT IDEA I want to share is that NON-CLASS SELECTORS commit you to COMPLEXITY.
* NON-CLASS selectors using ELEMENTS, ATTRIBUTES and IDS.
* That means if you're writing selectors like these, are red flags that YOU'RE ALREADY writing UNSCALABLE CSS.
* In my experience, here's how this happens…



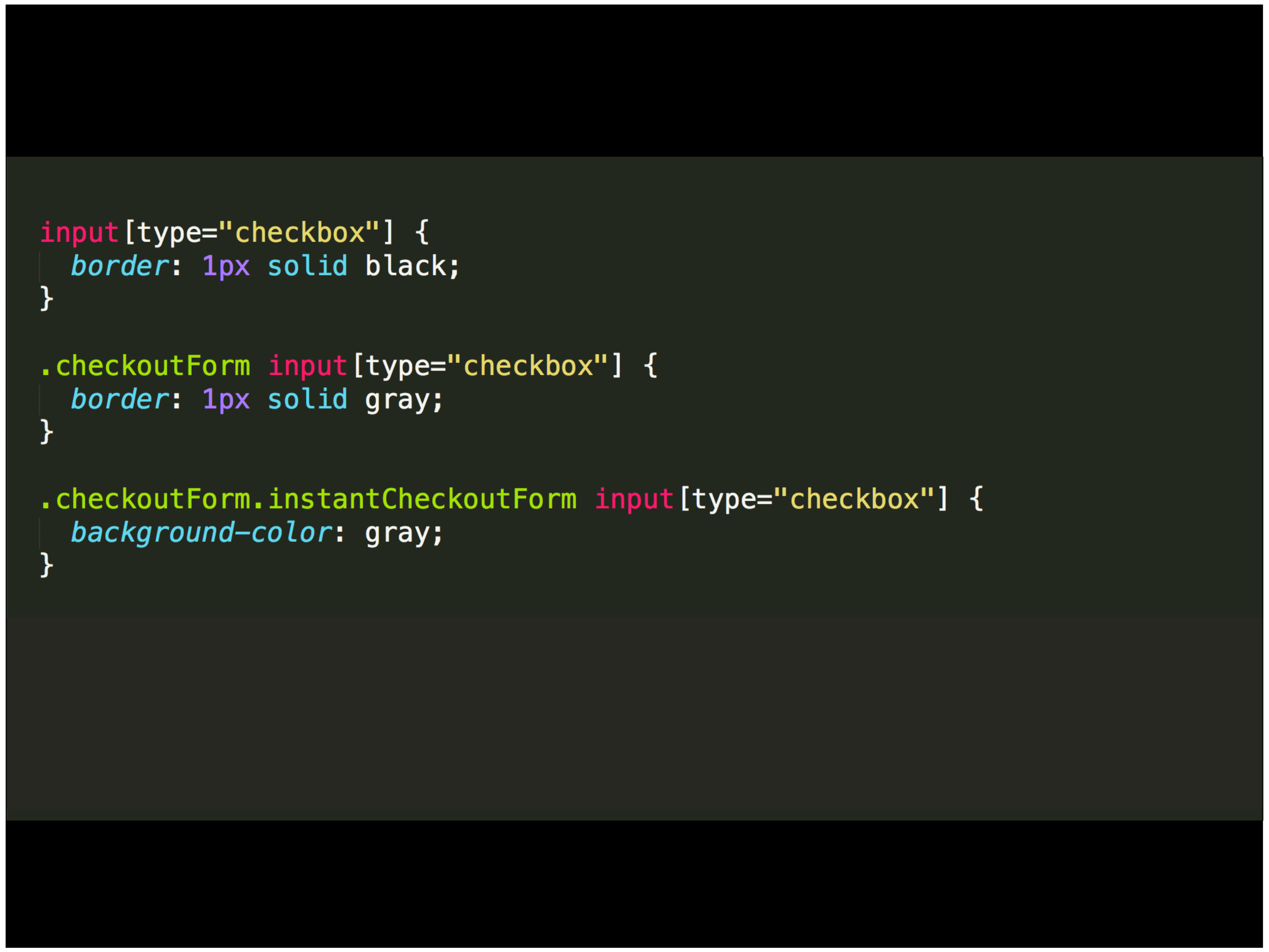
```css
input[type="checkbox"] {
  border: 1px solid black;
}
```

- I generally need to style something, like maybe a checkbox.
- And things would start out fine.
- I have a CHECKBOX. Makes sense right?



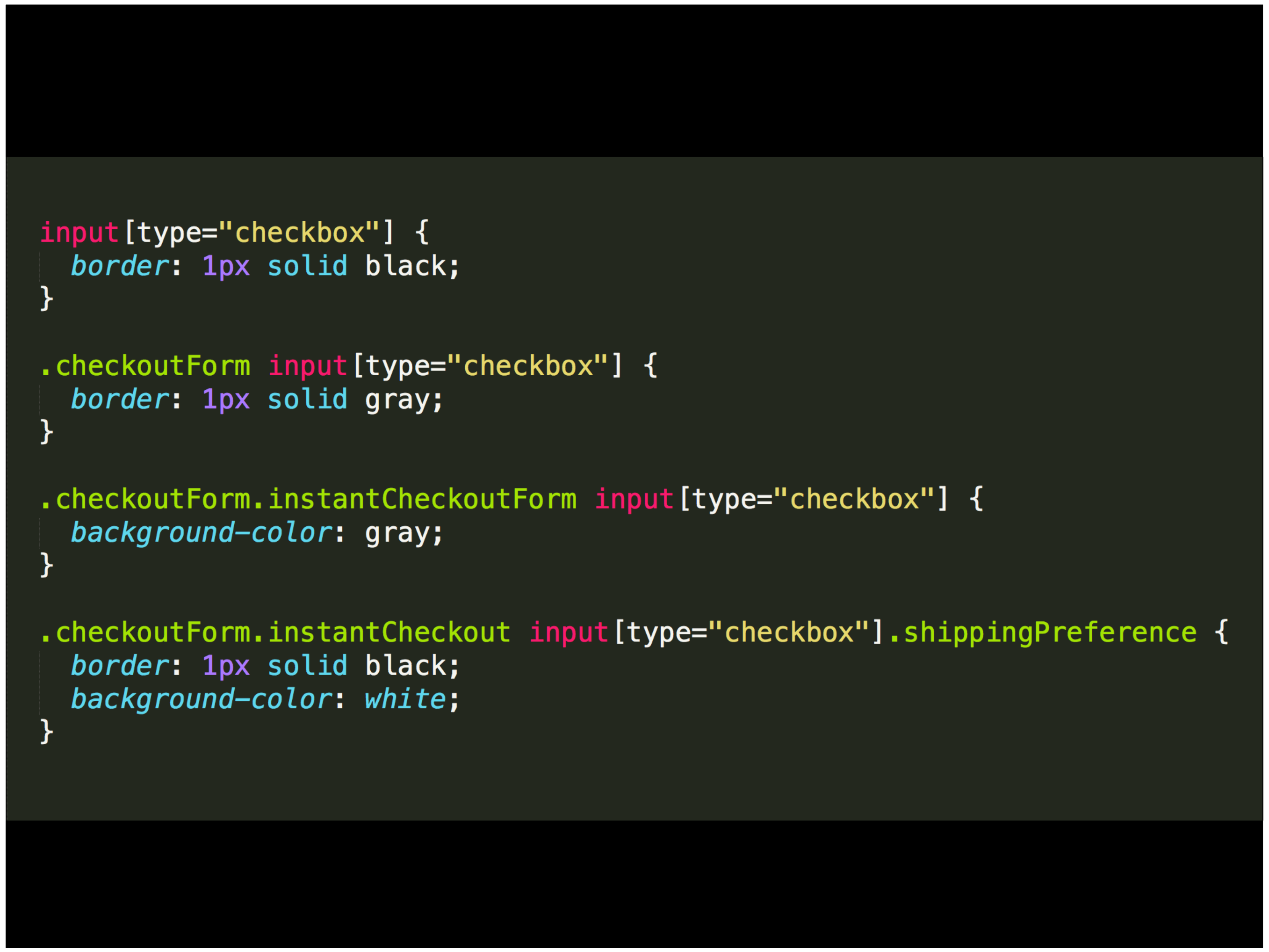
```css
input[type="checkbox"] {
  border: 1px solid black;
}

.checkoutForm input[type="checkbox"] {
  border: 1px solid gray;
}
```

- But then there’s a new requirement in the UI. The designer wants a different KIND of checkbox.
- Except now the checkbox input already has default styles.
- Well OK… no problem. I’ll just make a little scoping selector to add a little specificity boost to override these styles.



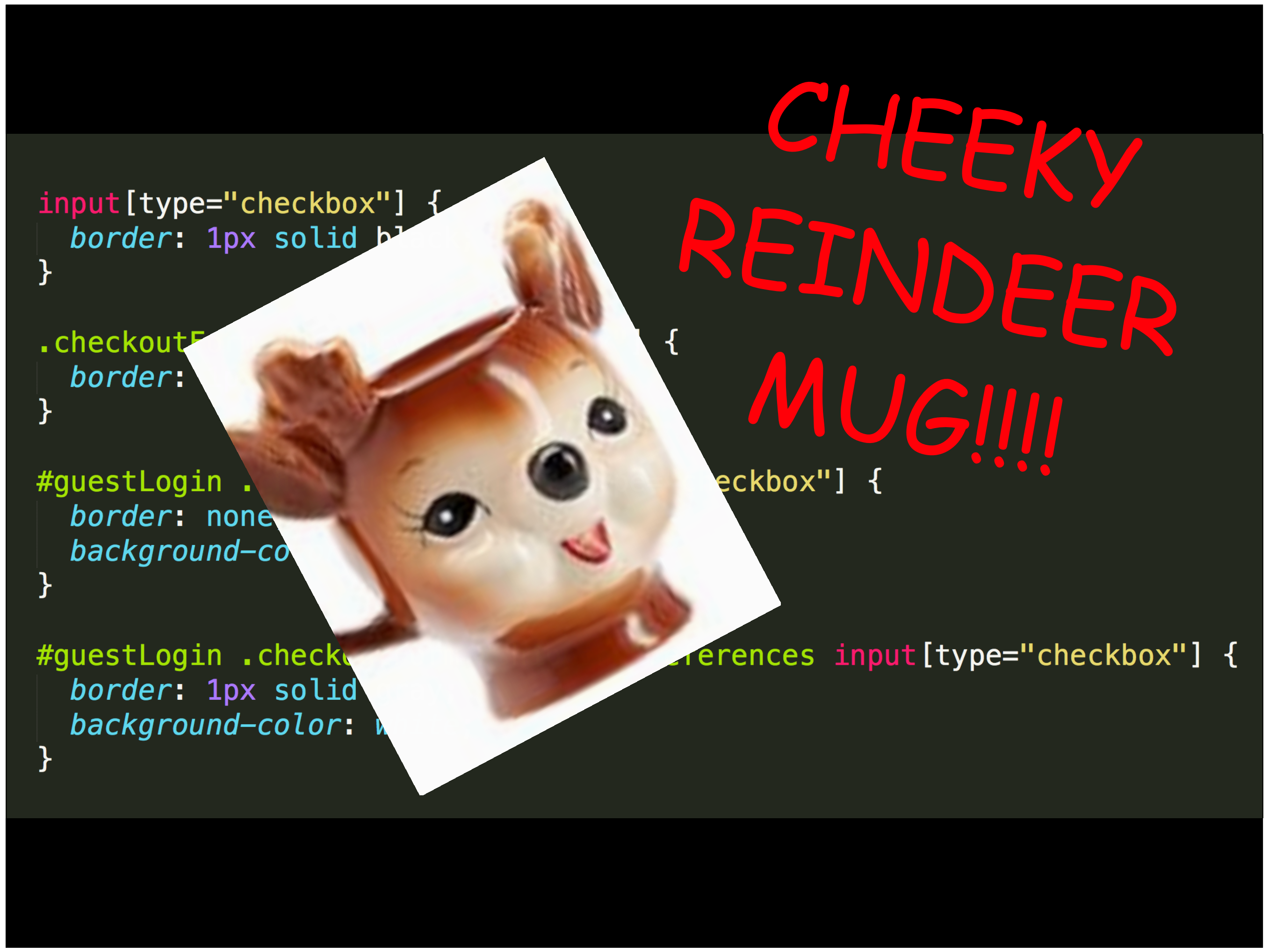
```css
input[type="checkbox"] {
  border: 1px solid black;
}

.checkoutForm input[type="checkbox"] {
  border: 1px solid gray;
}

.checkoutForm.instantCheckoutForm input[type="checkbox"] {
  background-color: gray;
}
```

* But time goes on. And new UI requirements come up.
* We need different KINDS of checkboxes.
* I need to create MORE SPECIFIC selectors to override the ones that already exist.



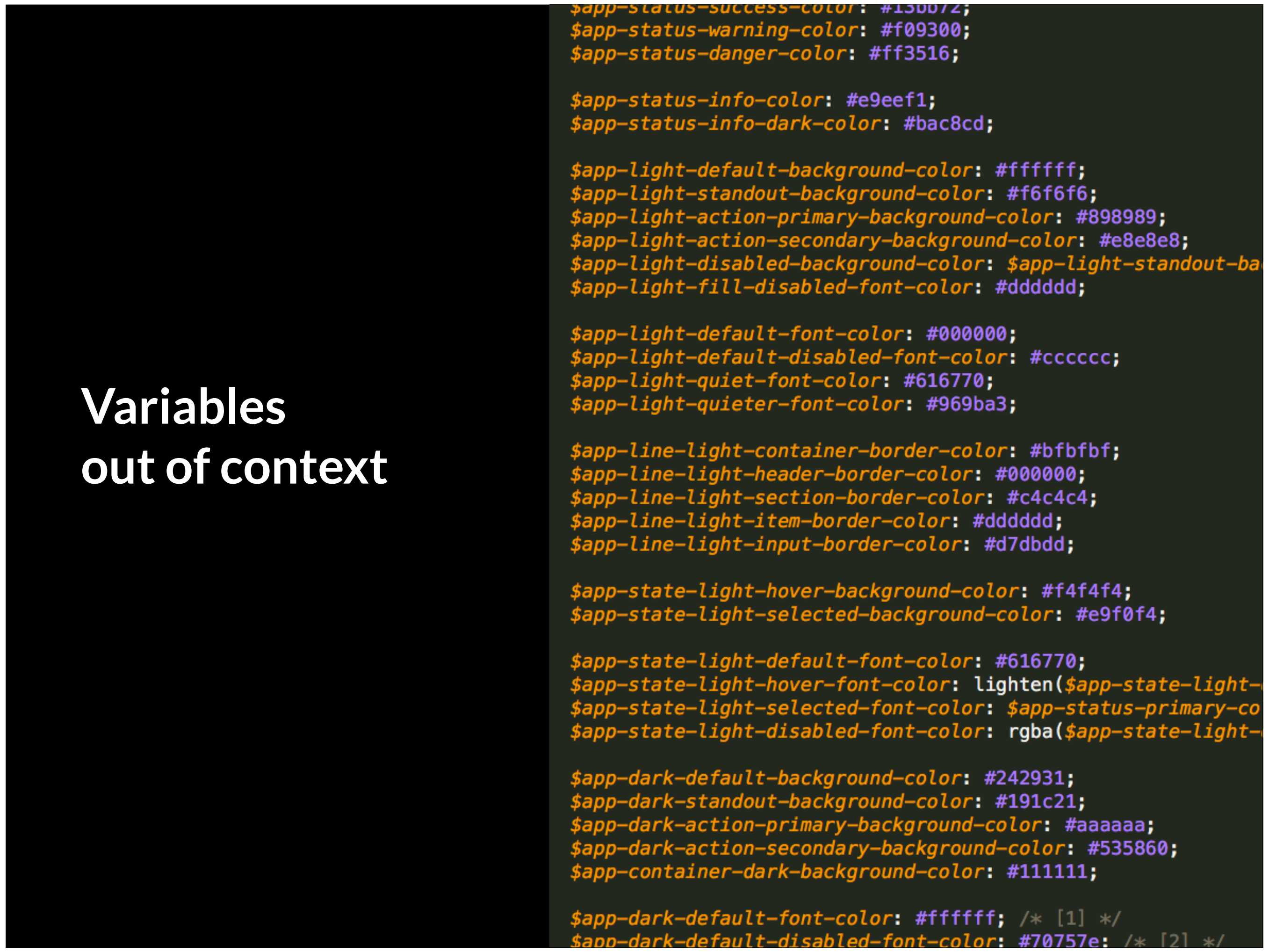
```css
input[type="checkbox"] {
  border: 1px solid black;
}

.checkoutForm input[type="checkbox"] {
  border: 1px solid gray;
}

.checkoutForm.instantCheckoutForm input[type="checkbox"] {
  background-color: gray;
}

.checkoutForm.instantCheckout input[type="checkbox"].shippingPreference {
  border: 1px solid black;
  background-color: white;
}
```

* I'm COMMITTED NOW.
* New UI requirements? NO PROBLEM! I know what to do!
* I'll just LAYER ON
* MORE AND MORE SPECIFIC SELECTORS!
* MORE AND MORE LAYERS OF COMPLEXITY!



```
input[type="checkbox"] {
  border: 1px solid bl
}

.checkoutF                        ] {
  border:
}

#guestLogin .                      eckbox"] {
  border: none
  background-co
}

#guestLogin .check                ferences input[type="checkbox"] {
  border: 1px solid
  background-color: w
}
```

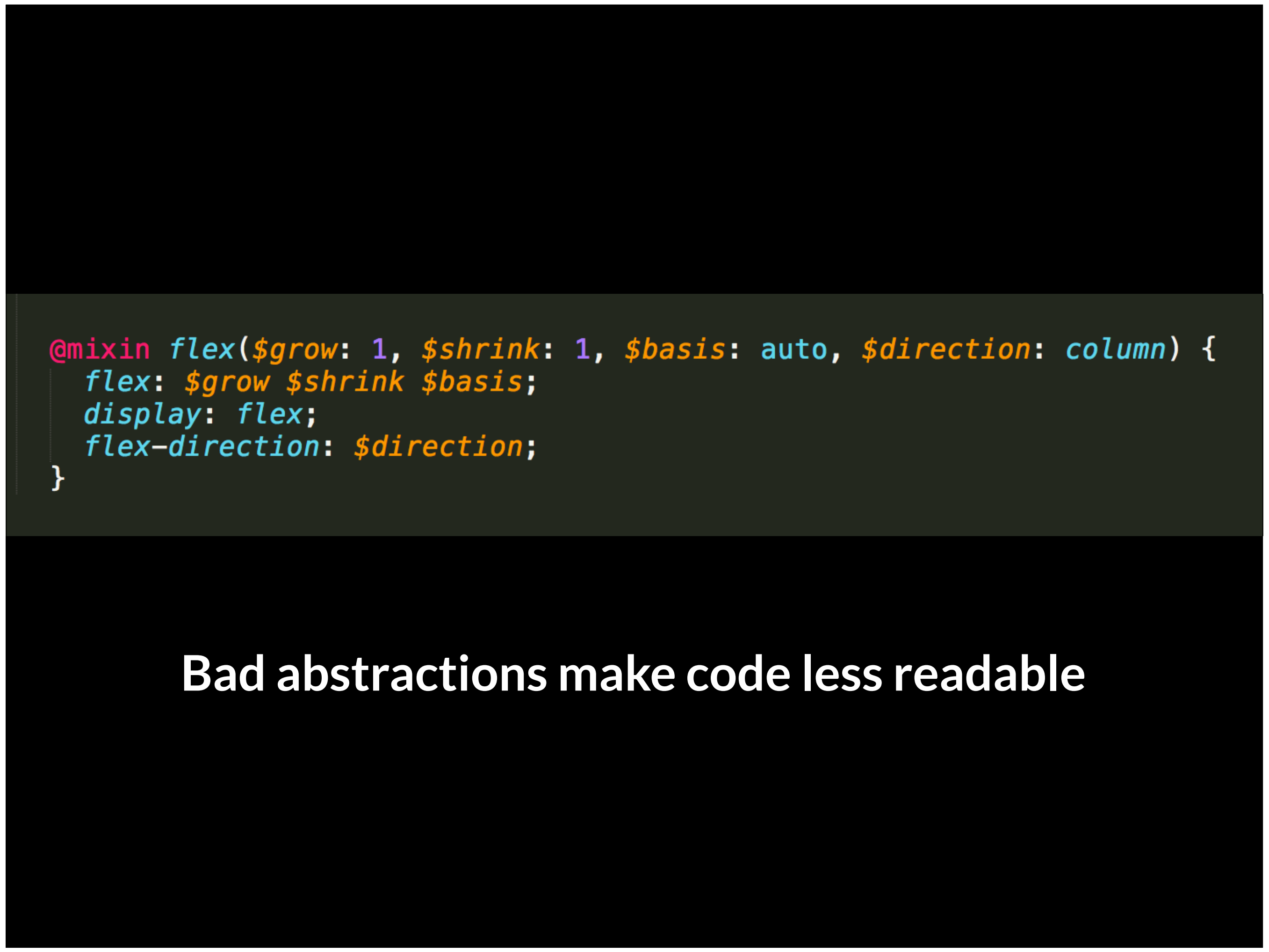
- Until eventually I end up with CHEEKY REINDEER MUG CSS!
- By STARTING OUT with NON-CLASS selectors
- COMMITTED to scaling by adding specificity.
- To scaling by adding complexity.
- It can happen to any of us. It can happen YOU!



Variables
out of context

```
$app-status-success-color: #13bb72;
$app-status-warning-color: #f09300;
$app-status-danger-color: #ff3516;

$app-status-info-color: #e9eef1;
$app-status-info-dark-color: #bac8cd;

$app-light-default-background-color: #ffffff;
$app-light-standout-background-color: #f6f6f6;
$app-light-action-primary-background-color: #898989;
$app-light-action-secondary-background-color: #e8e8e8;
$app-light-disabled-background-color: $app-light-standout-ba
$app-light-fill-disabled-font-color: #dddddd;

$app-light-default-font-color: #000000;
$app-light-default-disabled-font-color: #cccccc;
$app-light-quiet-font-color: #616770;
$app-light-quieter-font-color: #969ba3;

$app-line-light-container-border-color: #bfbfbf;
$app-line-light-header-border-color: #000000;
$app-line-light-section-border-color: #c4c4c4;
$app-line-light-item-border-color: #dddddd;
$app-line-light-input-border-color: #d7dbdd;

$app-state-light-hover-background-color: #f4f4f4;
$app-state-light-selected-background-color: #e9f0f4;

$app-state-light-default-font-color: #616770;
$app-state-light-hover-font-color: lighten($app-state-light-
$app-state-light-selected-font-color: $app-status-primary-co
$app-state-light-disabled-font-color: rgba($app-state-light-

$app-dark-default-background-color: #242931;
$app-dark-standout-background-color: #191c21;
$app-dark-action-primary-background-color: #aaaaaa;
$app-dark-action-secondary-background-color: #535860;
$app-container-dark-background-color: #111111;

$app-dark-default-font-color: #ffffff; /* [1] */
$app-dark-default-disabled-font-color: #70757e; /* [2] */
```

* This is SCSS, a CSS preprocessor language, but it has most of the SAME SYNTAX.
* In my first UI framework, I had a SINGLE COLOR VARIABLES file.
* 450 lines of color variables
* I made fun of Bootstrap for doing this, just a second ago, but I did the same thing.
* Variables were too far from where they were used
* By removing them SO FAR, I made it REALLY DIFFICULT to MAKE EFFECTIVE CHANGES



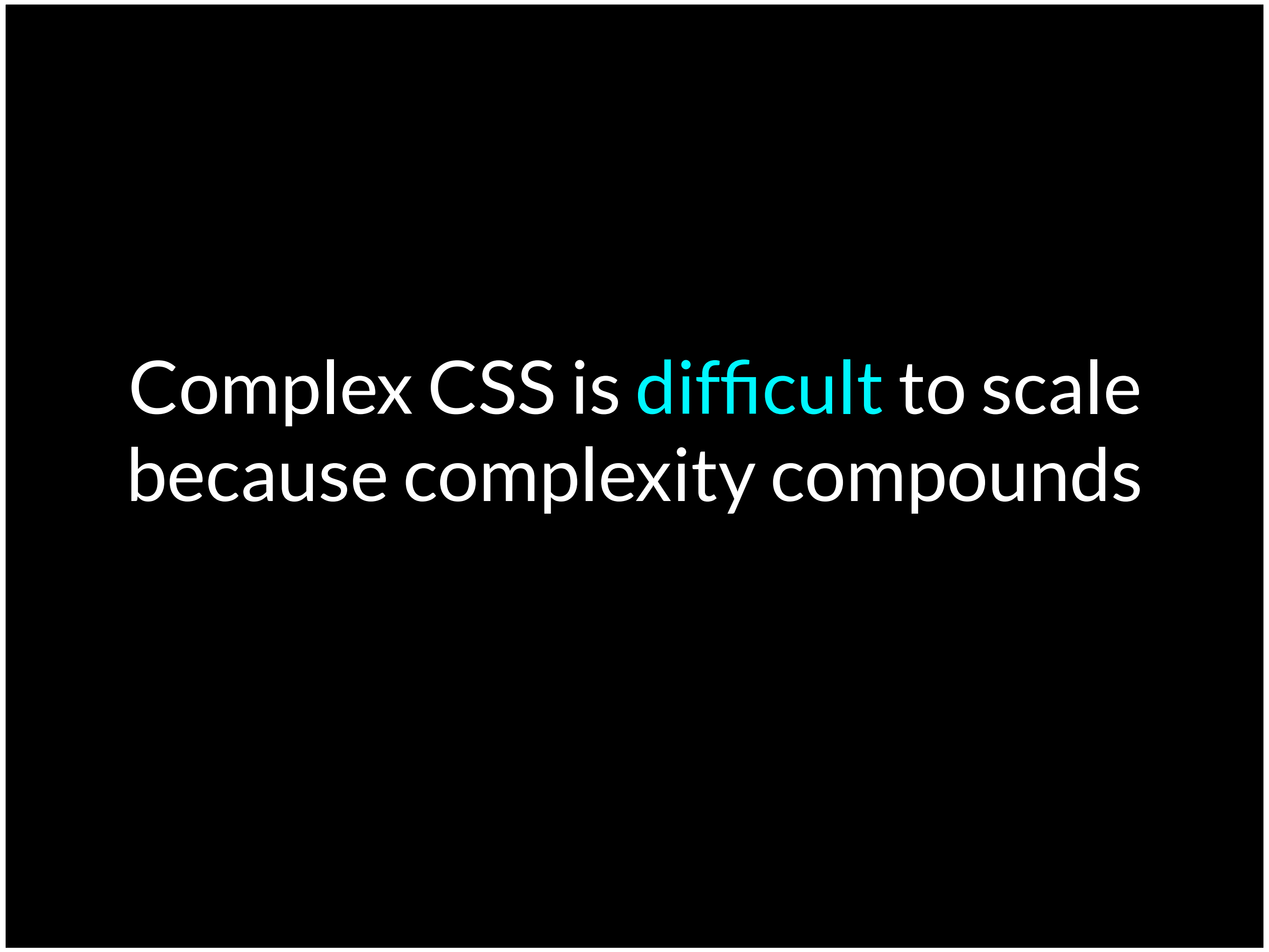
```scss
@mixin flex($grow: 1, $shrink: 1, $basis: auto, $direction: column) {
  flex: $grow $shrink $basis;
  display: flex;
  flex-direction: $direction;
}
```

Bad abstractions make code less readable

- In SCSS, mixins can be used to create abstractions around the CSS.
- Mixin was supposed to make it easier to use flexbox. You can call it with some arguments and it will apply the right flexbox properties.
- But, to use it you still need to know flexbox
- So you end up having to learn a PSEUDO-FLEXBOX INTERFACE that you can’t even use without LEARNING FLEXBOX in the first place
- This wrong abstractions just made the CSS HARDER TO UNDERSTAND.

CSS is difficult to scale when it has:

- Complex selectors
- Many interacting selectors
- Selectors coupled with markup
- Poor file/folder structure
- Confusing abstractions


Complex CSS is difficult to scale because complexity compounds

* When you notice yourself ADDING SPECIFICITY to a selector to OVERRIDE styles.
* You know you’re in trouble.
* You are COMMITTED to scaling your codebase by ADDING MORE COMPLEXITY on top of complexity.

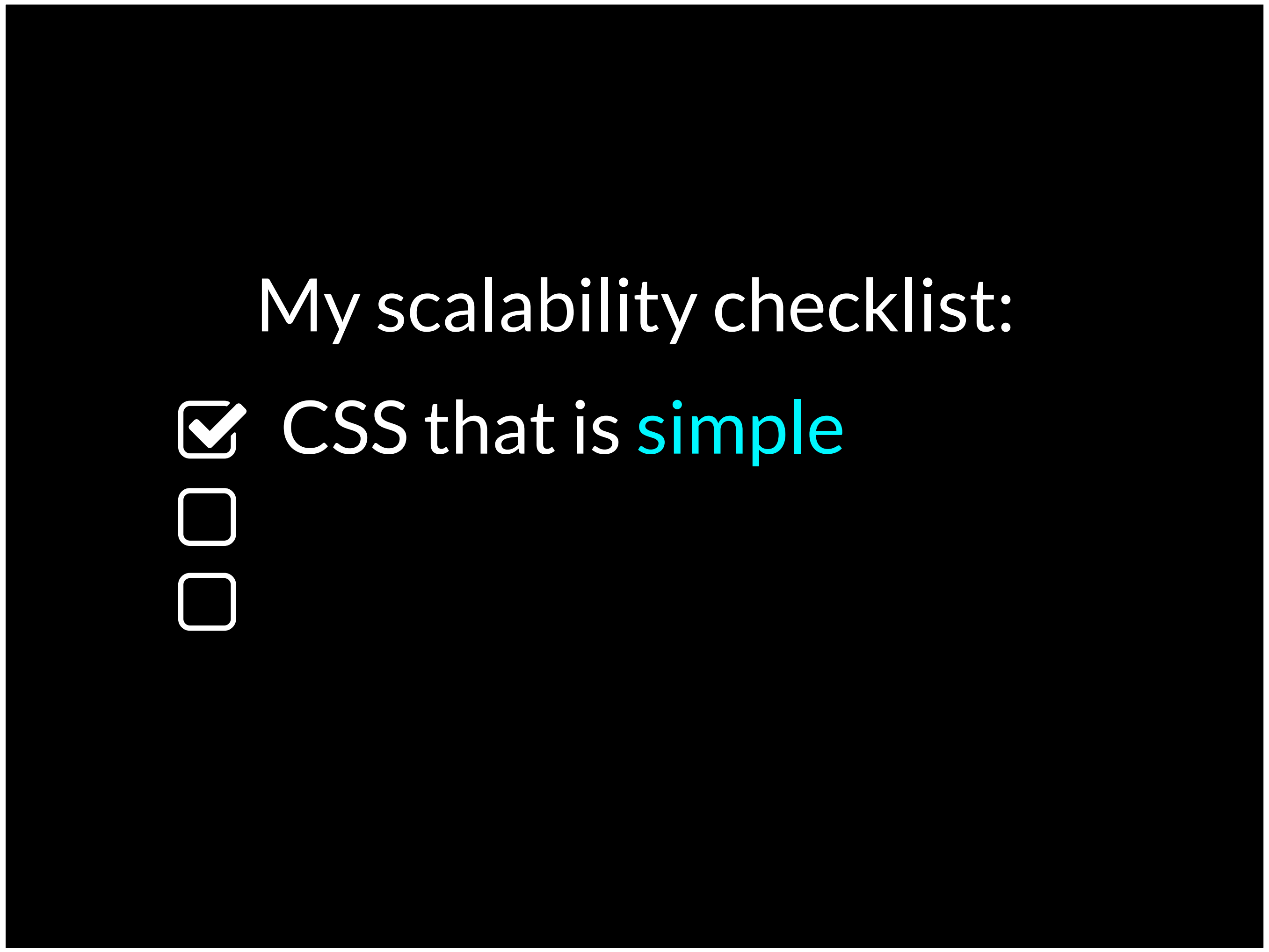
My scalability checklist:

☐

☐

☐

* I have a SCALABILITY CHECKLIST.



My scalability checklist:

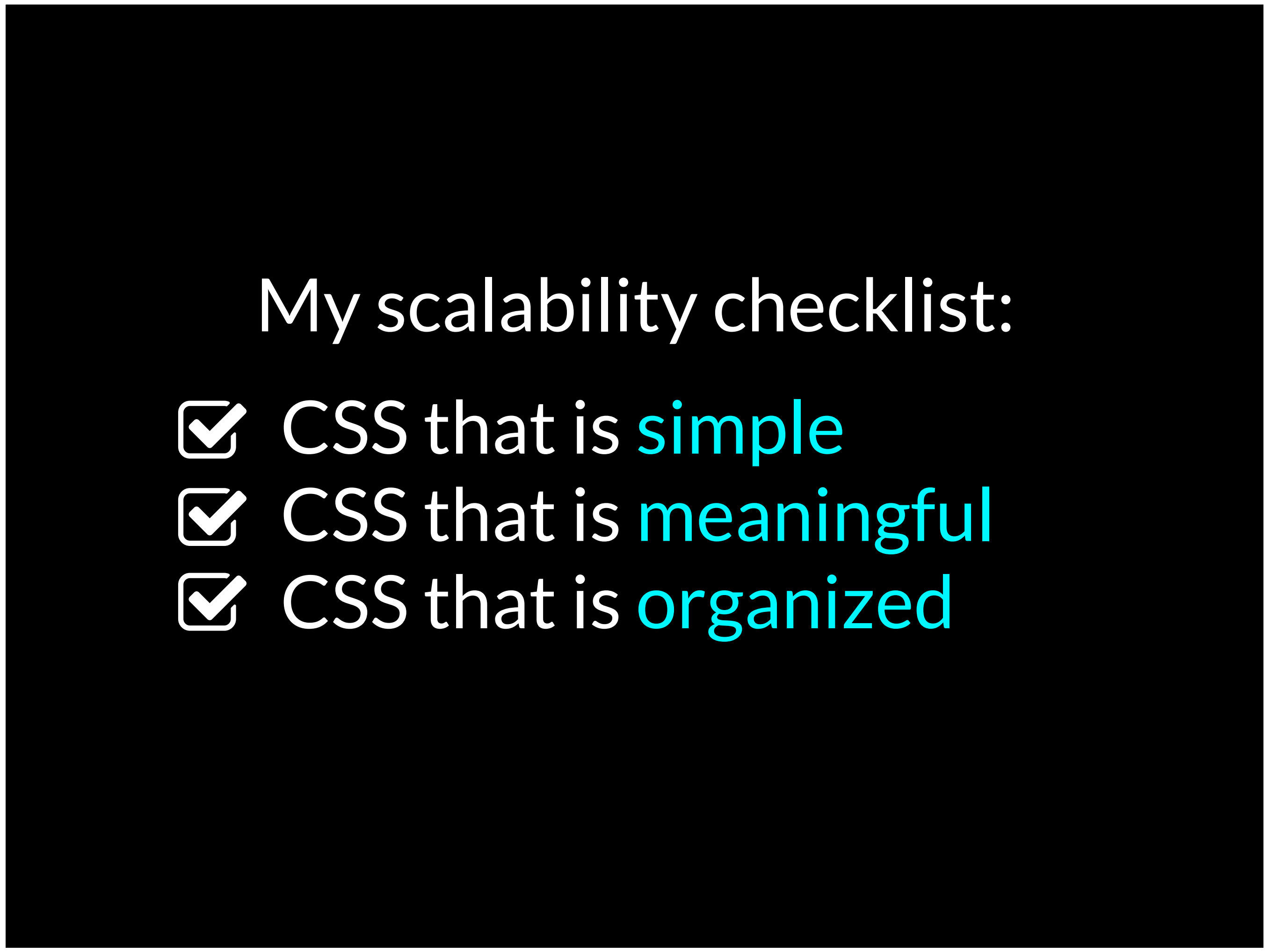
- ✓ CSS that is simple
- ☐
- ☐

* CSS can scale if it's SIMPLE.
*

My scalability checklist:

- ✓ CSS that is simple
- ✓ CSS that is meaningful
- ☐

* CSS can scale if it’s MEANINGFUL.



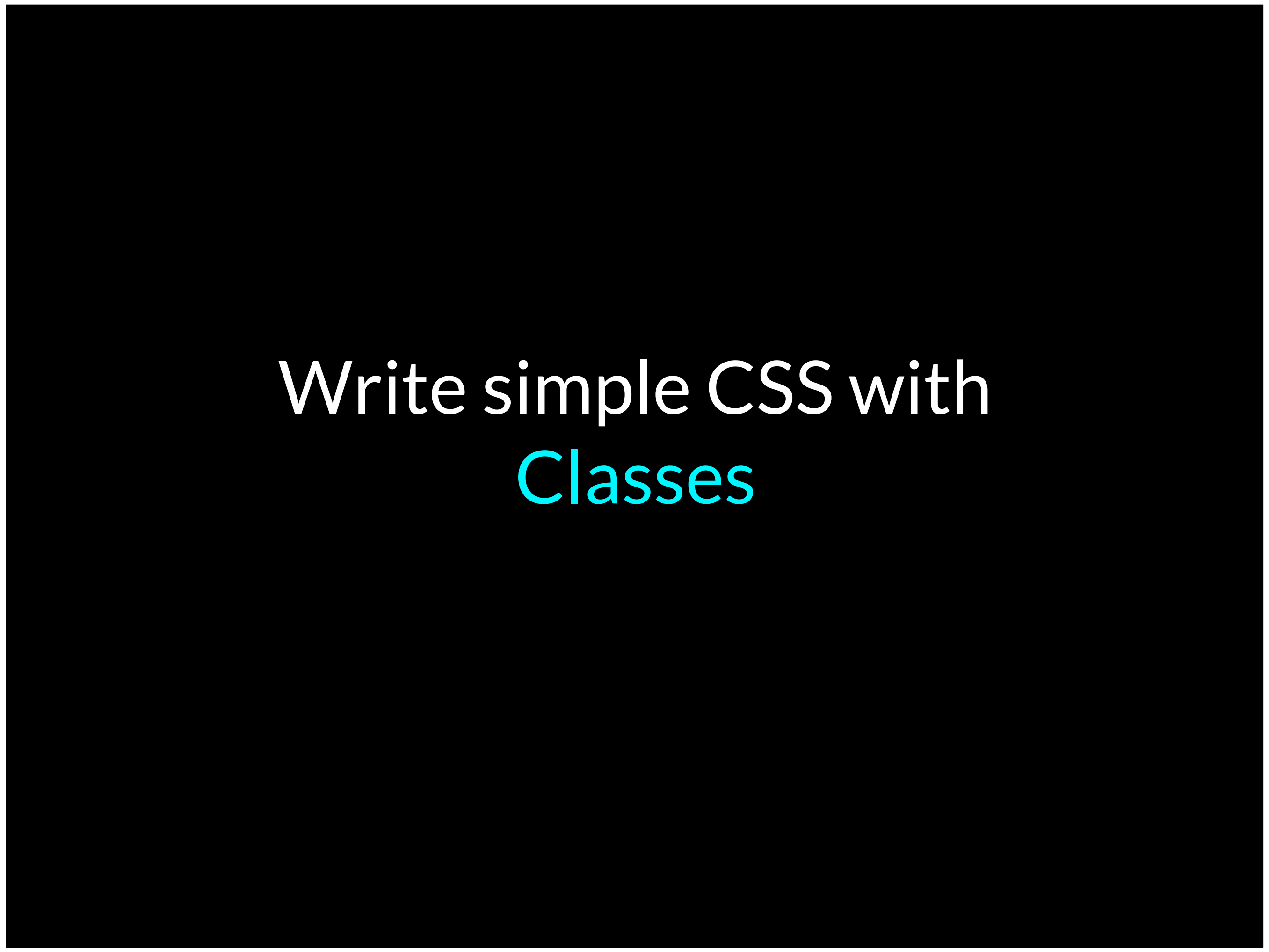
My scalability checklist:

- ☑ CSS that is simple
- ☑ CSS that is meaningful
- ☑ CSS that is organized

* CSS can scale if it’s ORGANIZED.
*

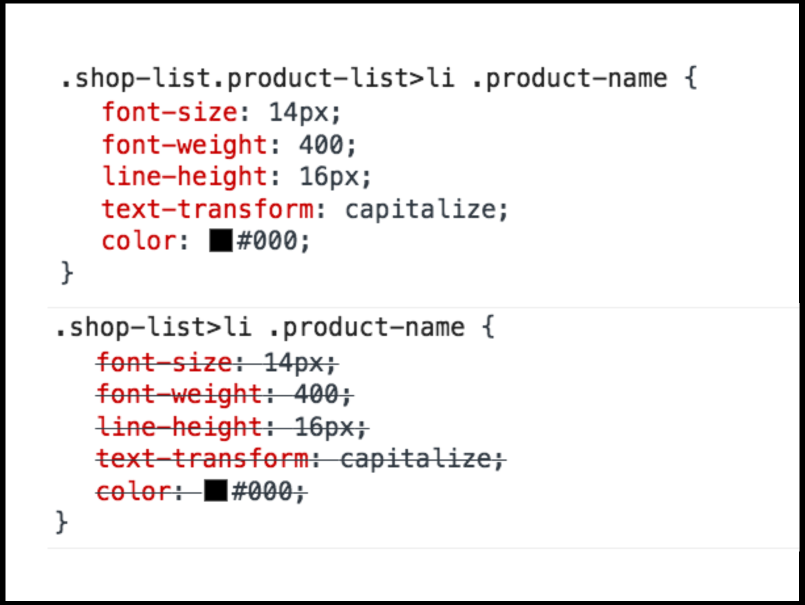
Apply the scalability checklist:

- Simplify CSS with classes
- Give it meaning with components
- Organize it with a UI framework



Write simple CSS with Classes

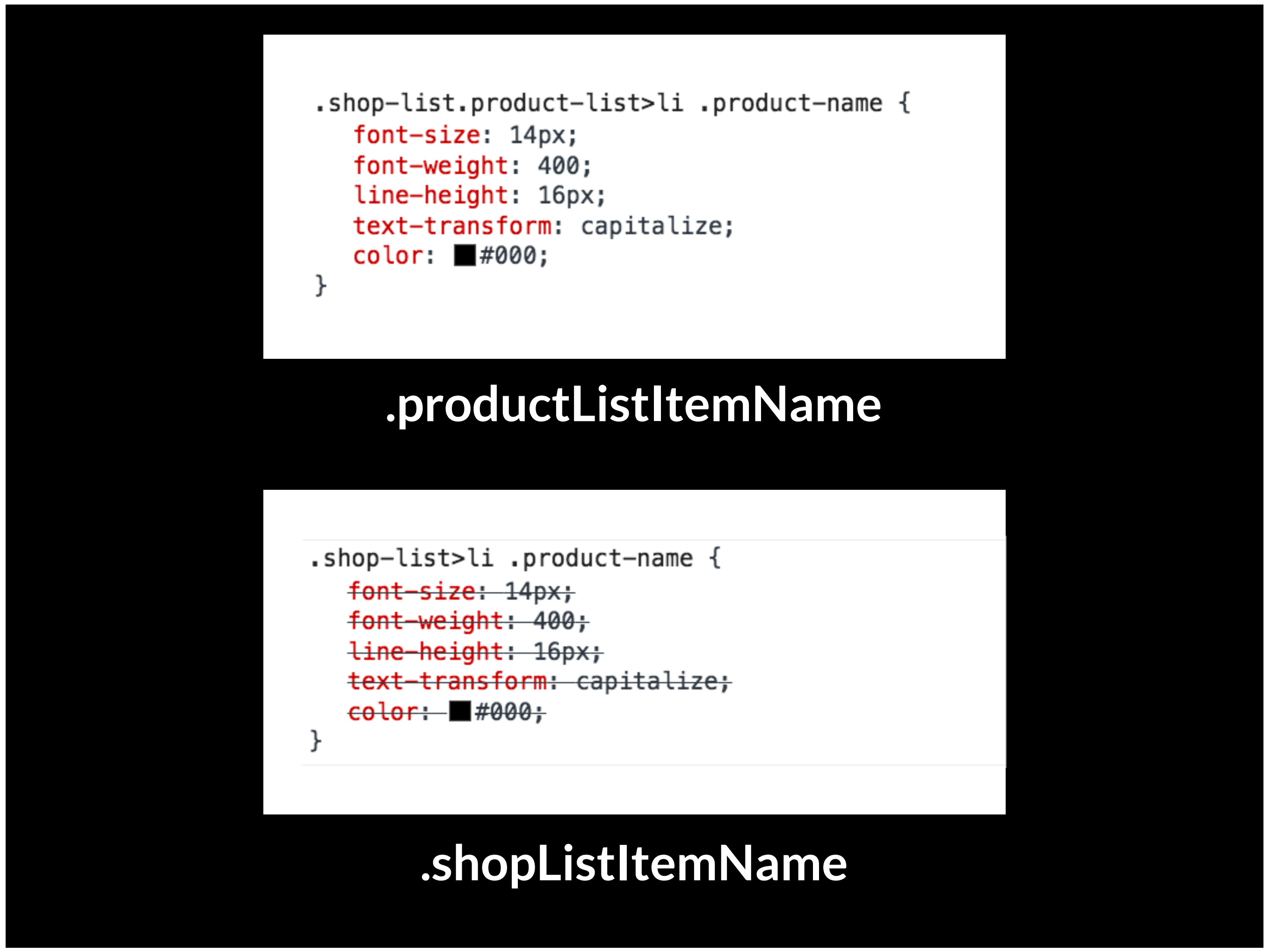
* Long, complex, interacting selectors make it difficult to scale CSS
* Solution is short, simple selectors
* We only need classes to do this



```
.shop-list.product-list>li .product-name {
   font-size: 14px;
   font-weight: 400;
   line-height: 16px;
   text-transform: capitalize;
   color: ■#000;
}
```

.shop-list>li .product-name {
   ~~font-size: 14px;~~
   ~~font-weight: 400;~~
   ~~line-height: 16px;~~
   ~~text-transform: capitalize;~~
   ~~color: ■#000;~~
}

* Let's look AGAIN at this EARLIER EXAMPLE.
* BOTH selectors style the PRODUCT NAME inside of a LIST ITEM
* The DIFFERENCE is that the top one is for .PRODUCT-LIST and the bottom one is for .SHOP-LIST
* But these are very COMPLEX selectors which involve both CLASSES and ELEMENTS.



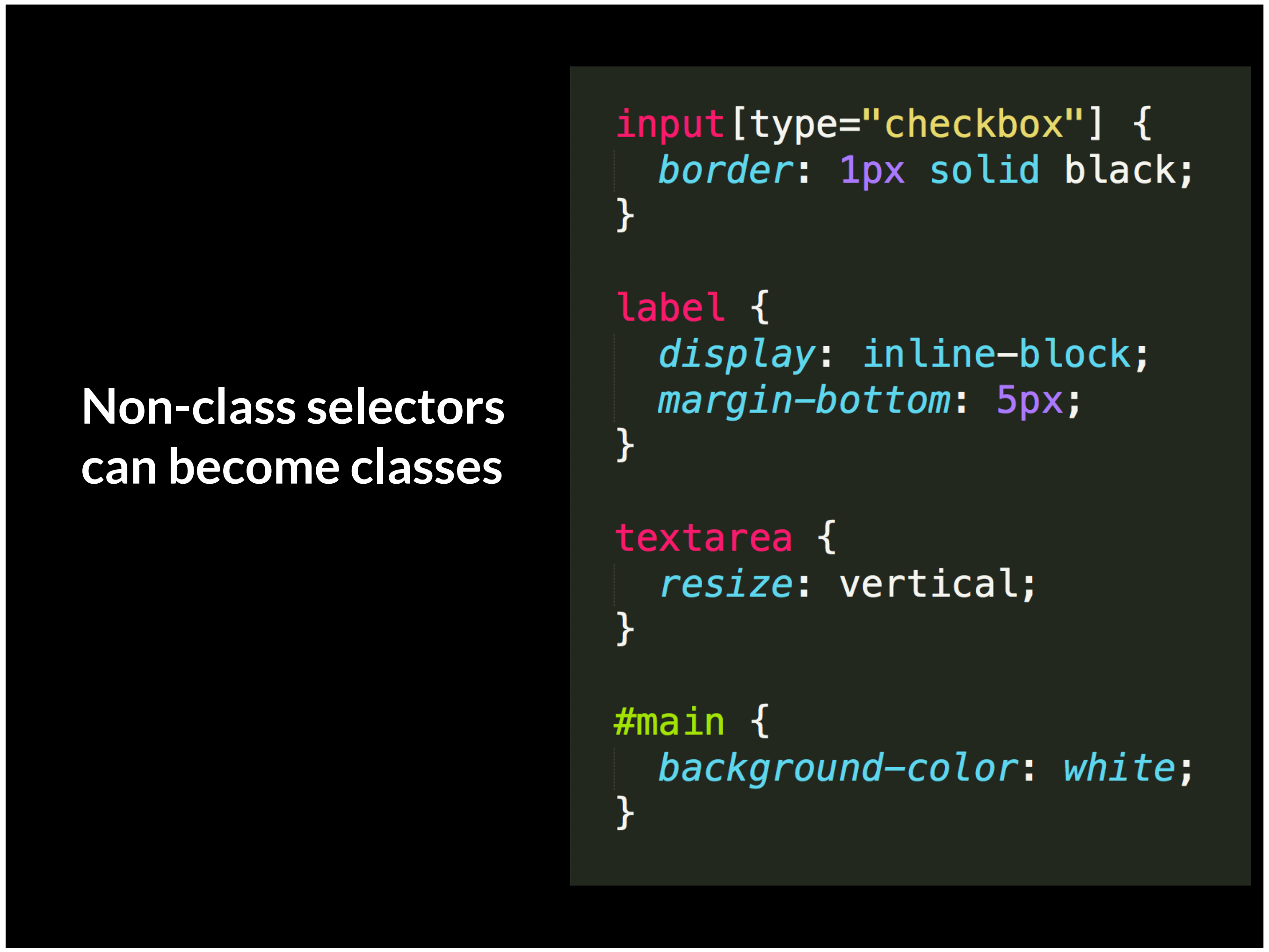
```
.shop-list.product-list>li .product-name {
   font-size: 14px;
   font-weight: 400;
   line-height: 16px;
   text-transform: capitalize;
   color: ■#000;
}
```

.productListItemName

```
.shop-list>li .product-name {
   ~~font-size: 14px;~~
   ~~font-weight: 400;~~
   ~~line-height: 16px;~~
   ~~text-transform: capitalize;~~
   ~~color: ■#000;~~
}
```

.shopListItemName

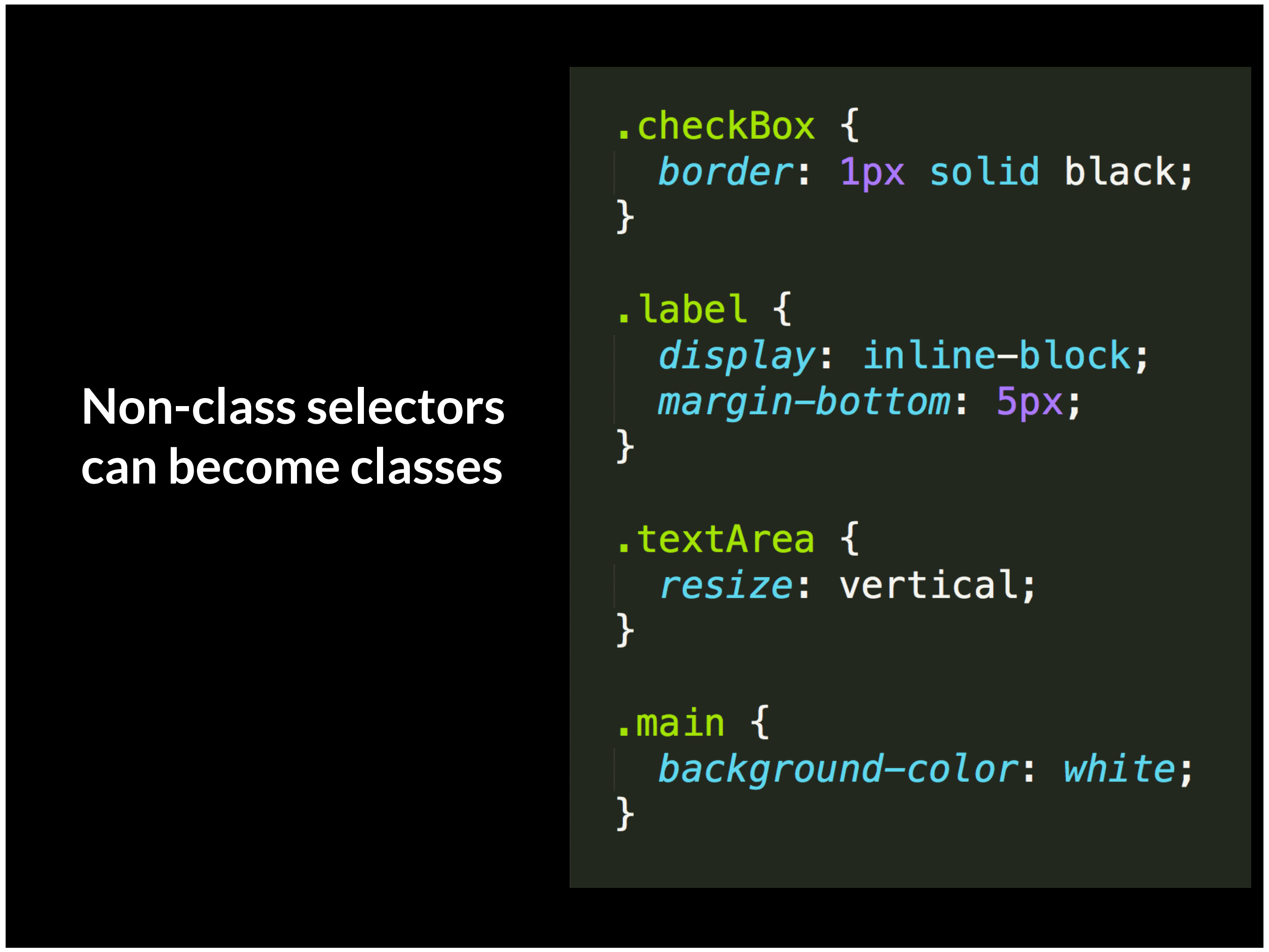
- I would just create simple classes to indicate each one
- I think the verbosity is fine… it’s worth the tradeoff:
- You get a CLEAR and SIMPE name
- And they would NO LONGER INTERACT — so the way the STYLES affect the element becomes simple
- ASIDE: I just wanted to point out that I personally love using camel-case in CSS. I don’t think it’s really caught on yet, but I definitely urge you to GIVE IT A SHOT. You might like it too!



Non-class selectors can become classes

```css
input[type="checkbox"] {
  border: 1px solid black;
}

label {
  display: inline-block;
  margin-bottom: 5px;
}

textarea {
  resize: vertical;
}

#main {
  background-color: white;
}
```

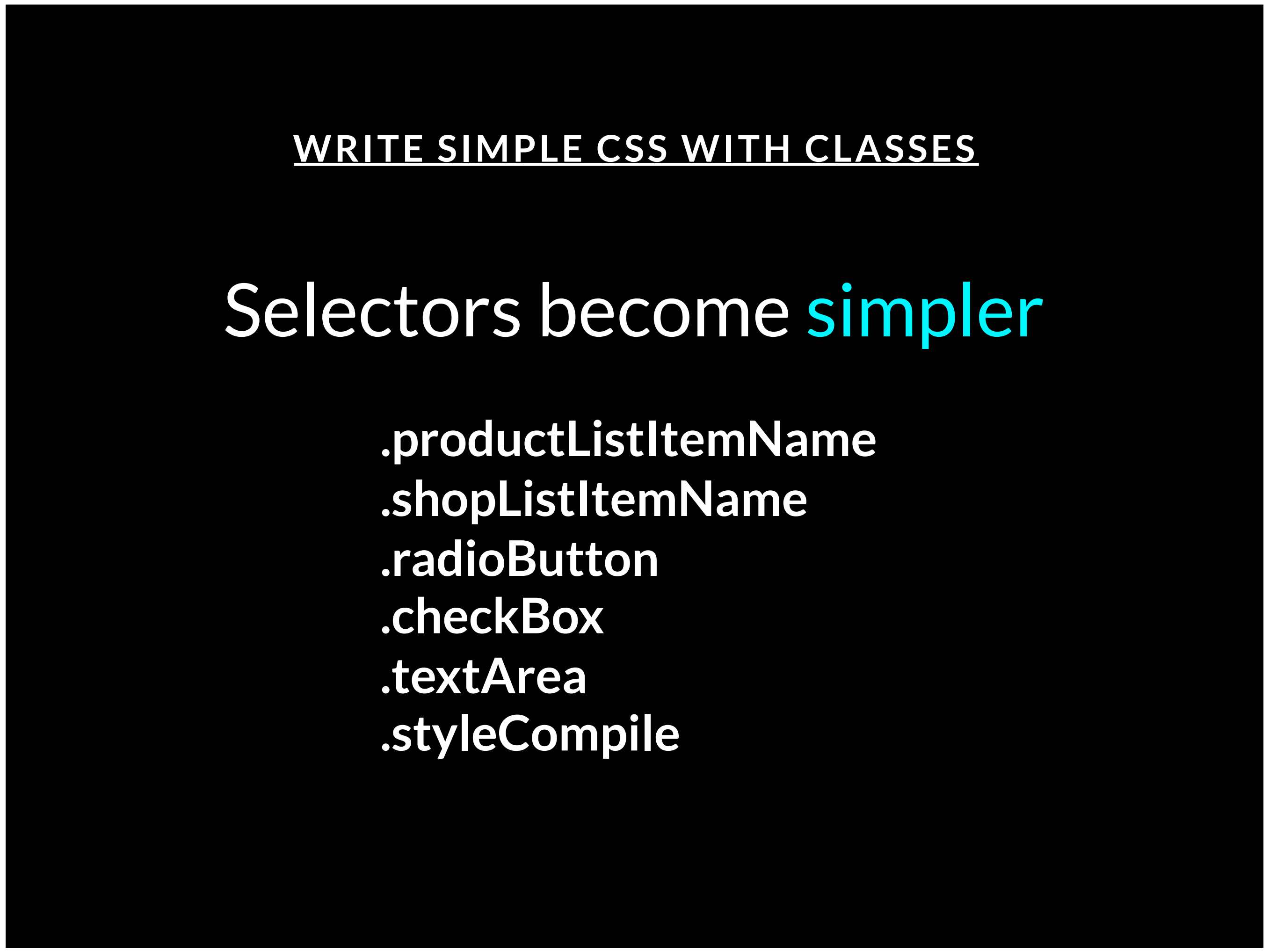
We can apply this same pattern…



Non-class selectors can become classes

```
.checkBox {
  border: 1px solid black;
}

.label {
  display: inline-block;
  margin-bottom: 5px;
}

.textArea {
  resize: vertical;
}

.main {
  background-color: white;
}
```

- We can apply the SAME RULE and replace these selectors with CLASSES.
- What's great about this is that now you gain FLEXIBILITY.
- Now if you wanted to have DIFFERENT TYPES of checkBoxes, you wouldn't need to OVERRIDE any styles applied to the ELEMENT.
- You could just CREATE A NEW CHECKBOX CLASS.

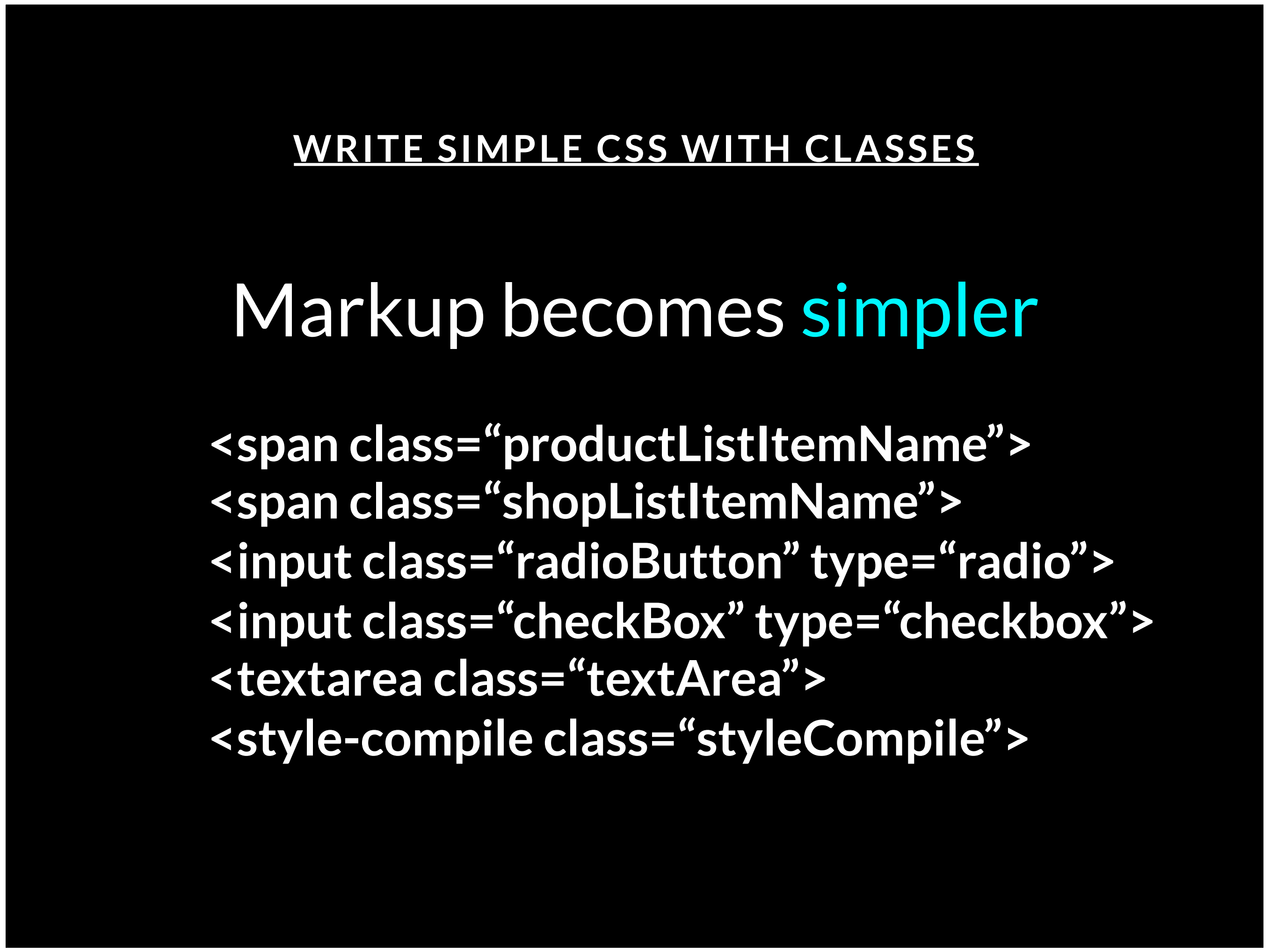


WRITE SIMPLE CSS WITH CLASSES

Selectors become simpler

.productListItemName
.shopListItemName
.radioButton
.checkBox
.textArea
.styleCompile

- Selectors become simpler: short, straightforward, clear

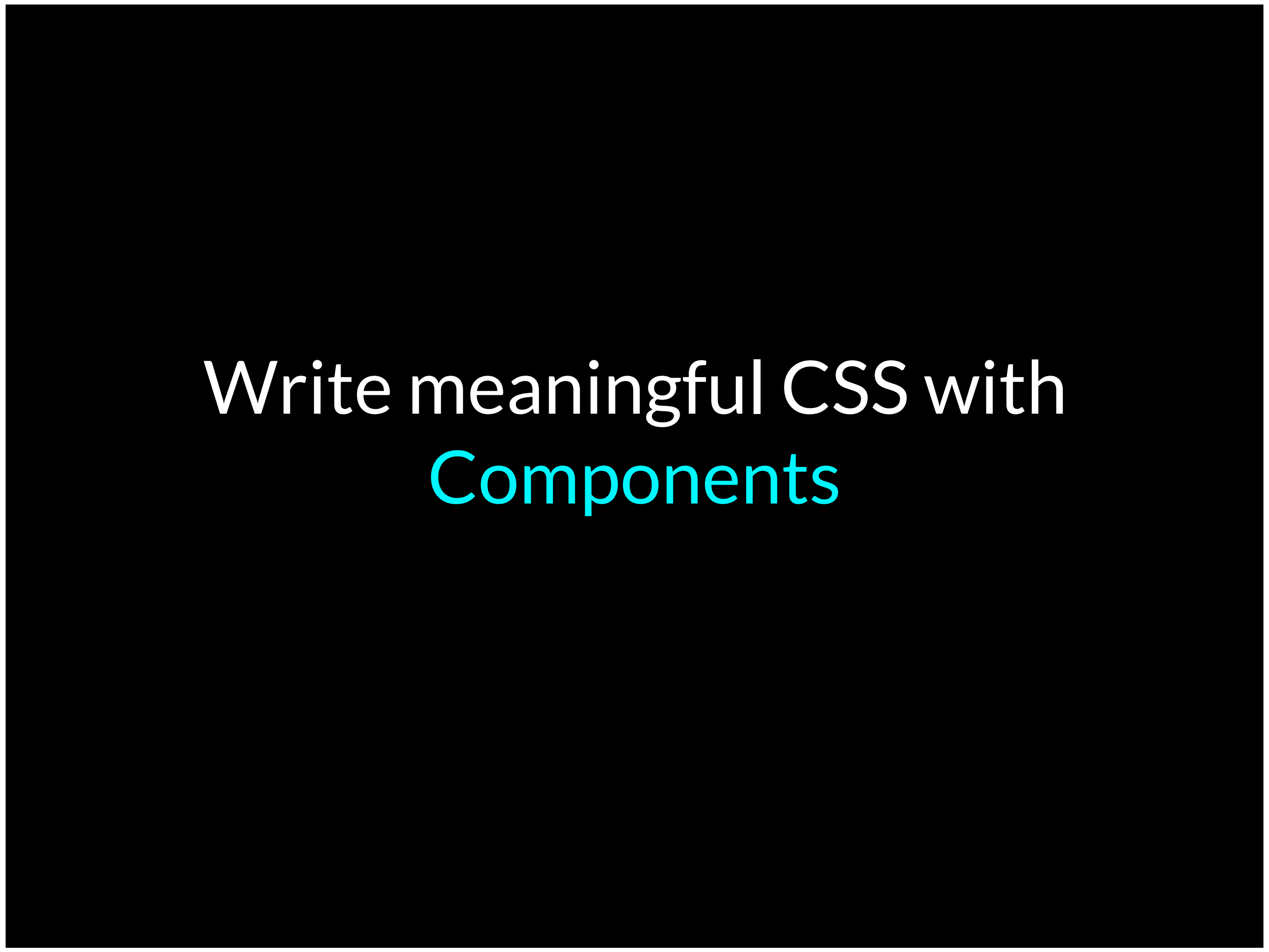


WRITE SIMPLE CSS WITH CLASSES

Markup becomes simpler

```
<span class="productListItemName">
<span class="shopListItemName">
<input class="radioButton" type="radio">
<input class="checkBox" type="checkbox">
<textarea class="textArea">
<style-compile class="styleCompile">
```

* Markup becomes simpler too



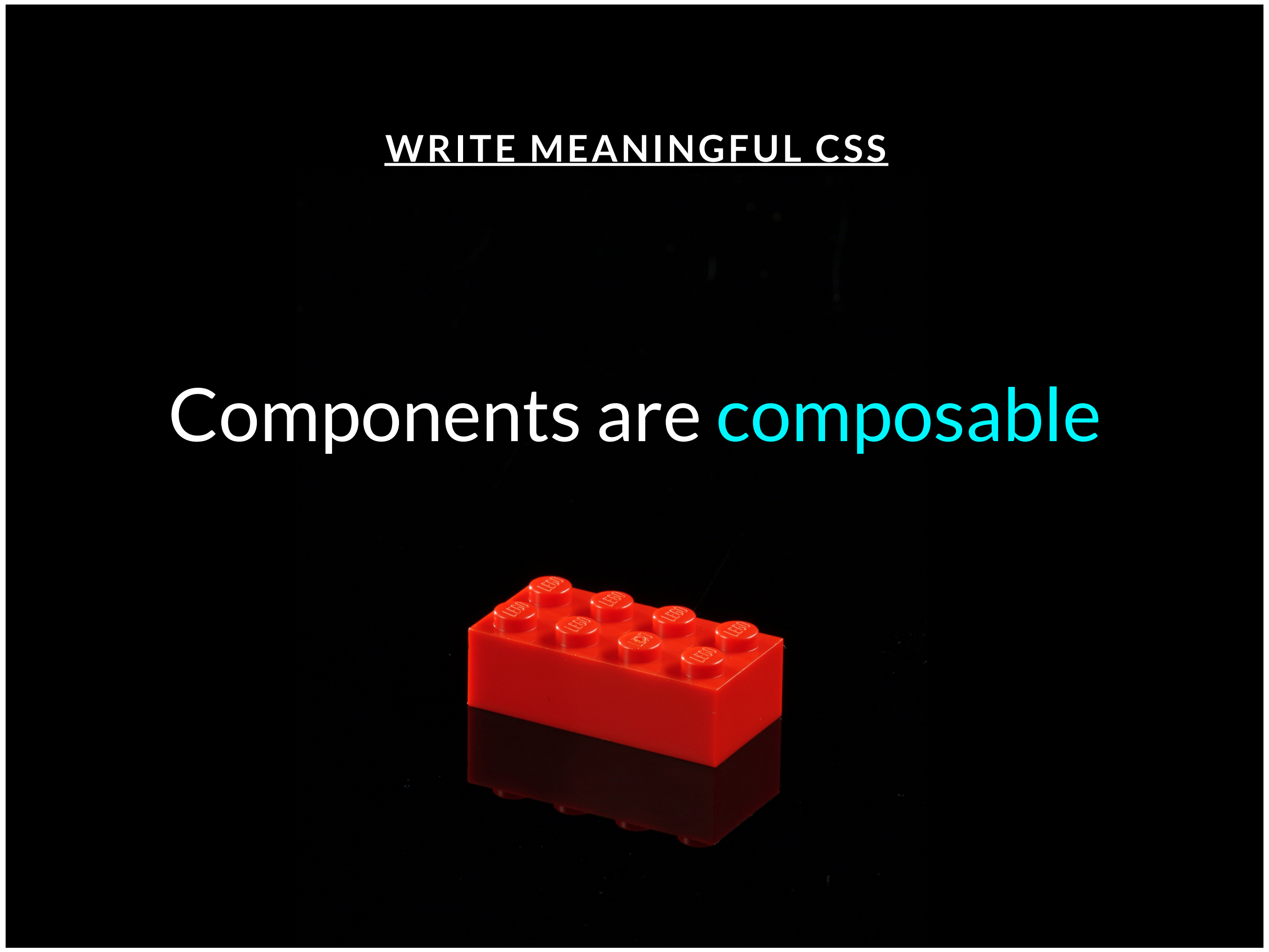
Write meaningful CSS with Components

* Meaningful selectors have clear ROLES in the UI
* We can make our CSS meaningful by thinking of UI and writing CSS in terms of COMPONENTS

WRITE MEANINGFUL CSS

Components have a single responsibility

* Single Responsibility Principle: a specific functionality in software is represented by a single unit of code
* In CSS, this means that a single requirement of the UI should be implemented by a single component



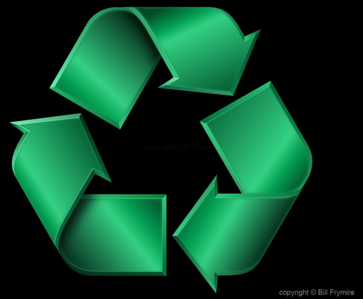
WRITE MEANINGFUL CSS

Components are composable

- small, fine-grained, simple components like Lego blocks
- we can put them together THIS WAY to create larger, more complex components
- This means that often times we want Components to be able to act as CONTAINERS for other components. And those components will in turn CONTAIN even more components. It's this ability to compose layers and layers of components that allows you to build very complex user interfaces from a relatively simple library of components.

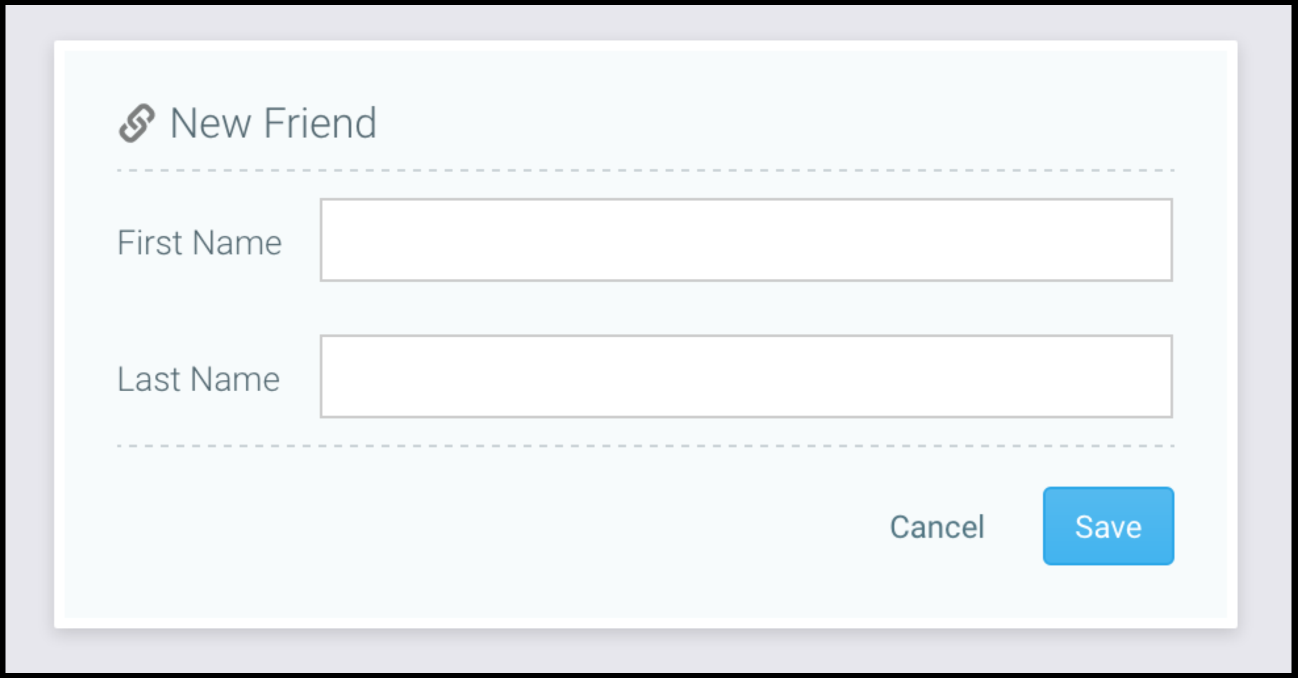
WRITE MEANINGFUL CSS

Components are reusable



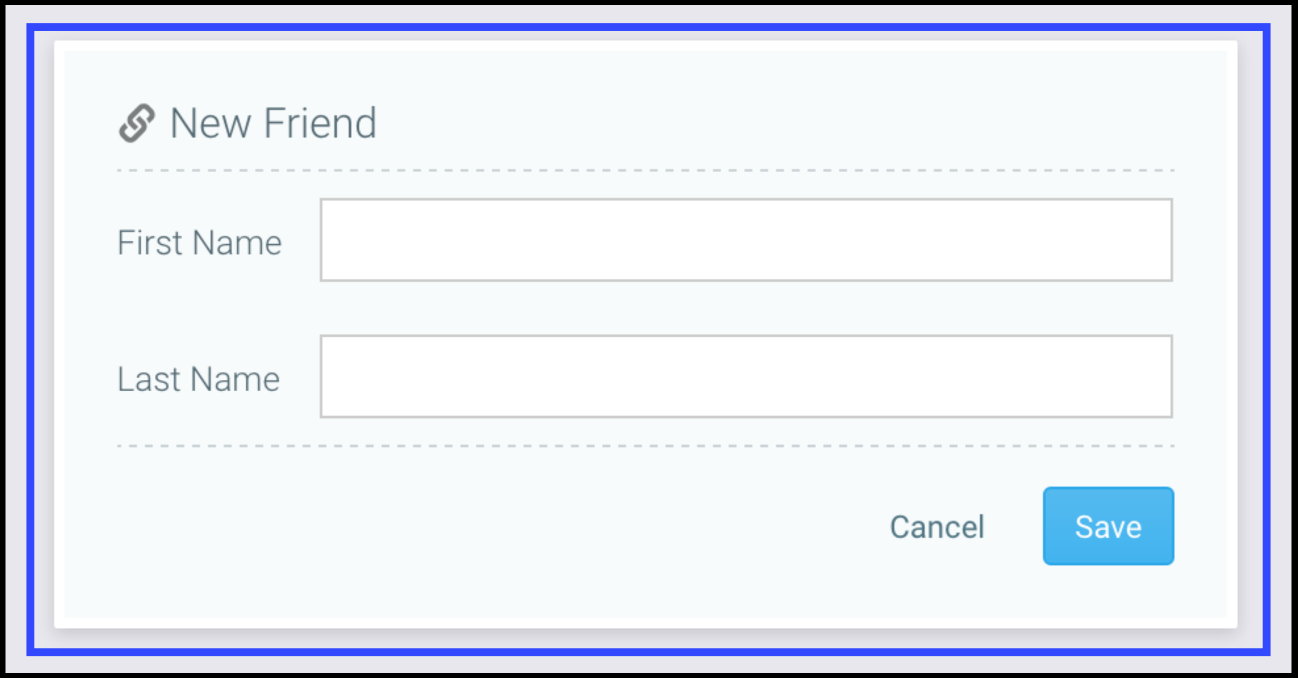
* We don’t want to design components to only work in specific scenarios. We want them to be as REUSABLE as possible, to maximize their utility.
* If we design components to function in any context, then we can reuse them over and over to build new user interfaces, even if the user interfaces have REQUIREMENTS we didn’t anticipate when building the components.

Example component: Modal



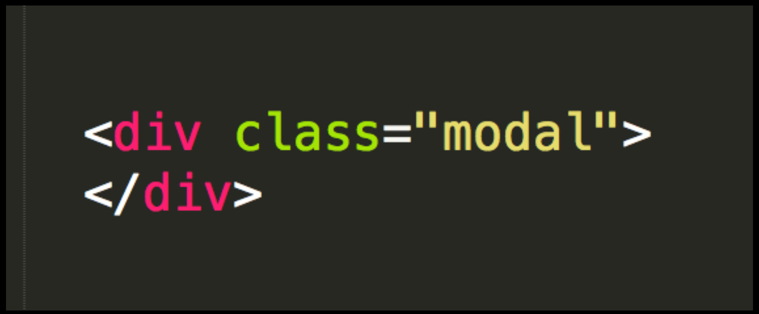
New Friend Modal

* Mockup from DESIGNER
* New friend modal
* Title, form fields, buttons
* Well what's start by asking ourselves: reusable about this?
* We'll start by working outside-in, which is an idea Brad Westfall brought up in his talk at last year's CSSDAY.io.



Generic “modal”

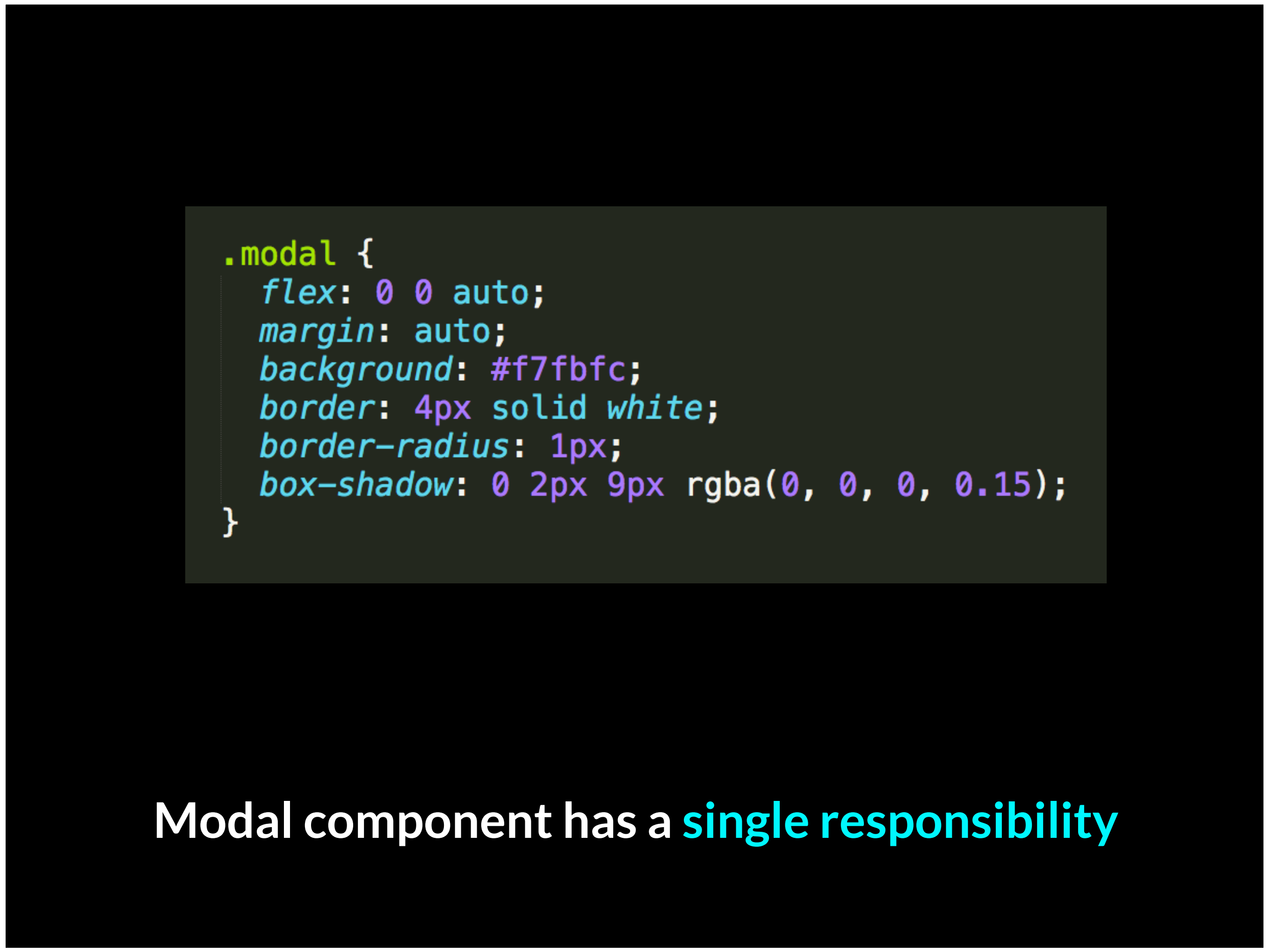
- The outermost component here is simply the idea of a generic “modal”.
- It seems like this could be used for ALL KINDS of MODALS, not just this one.
- So we can make a MODAL COMPONENT



```html
<div class="modal">
</div>
```

Clear role in UI

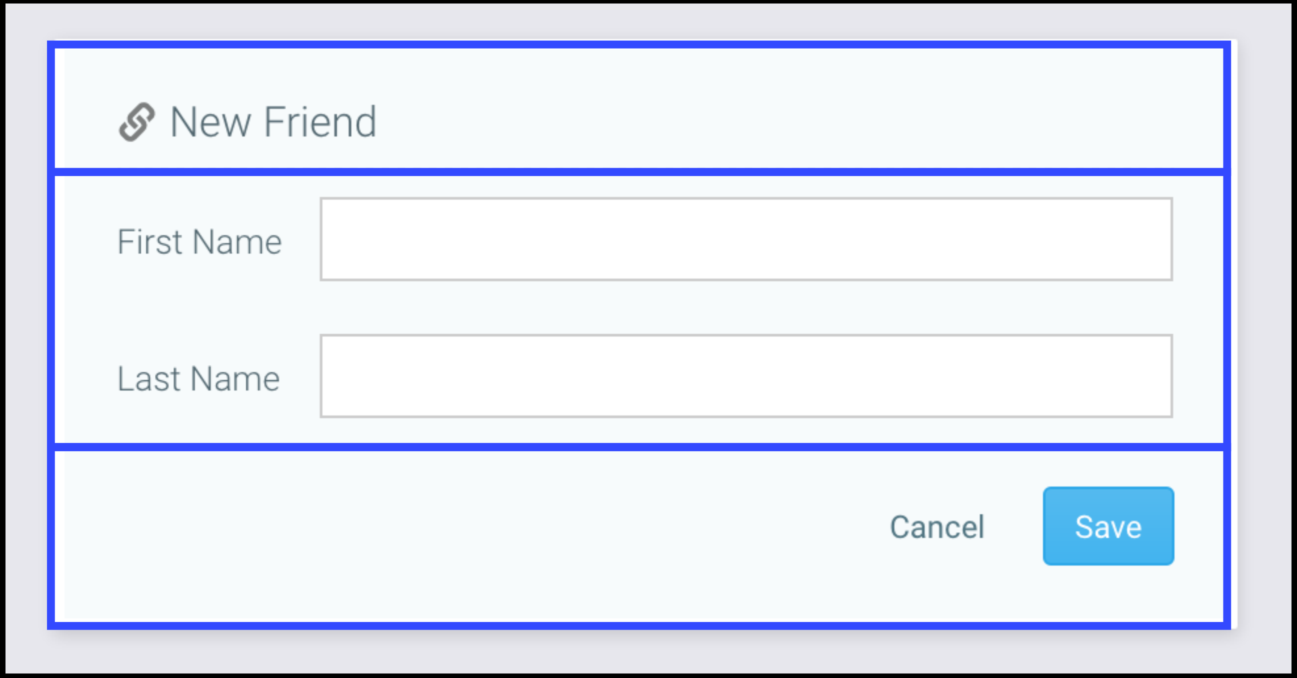
* This will basically just be a simple DIV.
* You just need to add the MODAL class to it.
* And from reading this MARKUP, the role in UI is clear. This DIV is a MODAL.



```css
.modal {
  flex: 0 0 auto;
  margin: auto;
  background: #f7fbfc;
  border: 4px solid white;
  border-radius: 1px;
  box-shadow: 0 2px 9px rgba(0, 0, 0, 0.15);
}
```

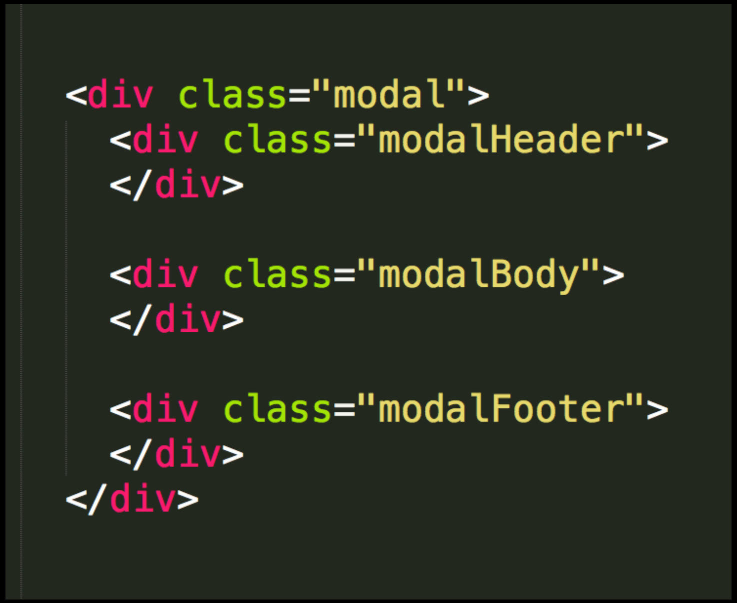
Modal component has a single responsibility

* Looking at the STYLES:
* It HAS background color and border, positioning and flex layout
* DOESN’T HAVE font-size, line-height, or anything else that can be INHERITED
* This means TWO THINGS.
* 1: This modal component has a SINGLE RESPONSIBILITY: look like a modal. Be a DUMB CONTAINER. You can literally put anything in this modal. Maybe some things won’t make sense, but it will still CONTAIN and present it.
* 2: Selectors for the CHILDREN inside of the modal won’t need to OVERRIDE anything.
* WHICH IS IMPORTANT, because as a DUMB CONTAINER, we’re going to FILL THIS MODAL WITH OTHER COMPONENTS. And we don’t want them to have to care about their CONTEXT.



.modalHeader
.modalBody
.modalFooter

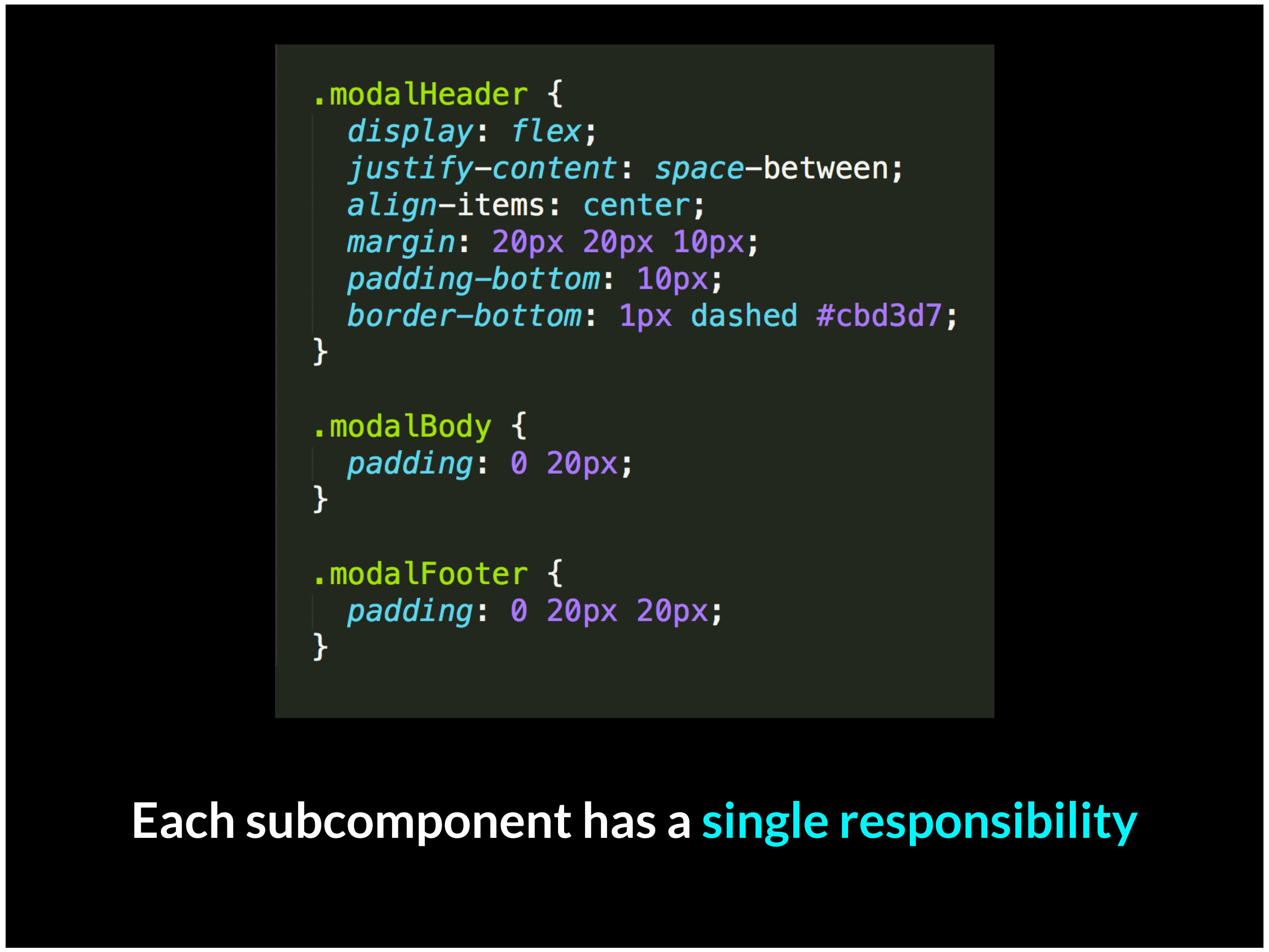
* Continuing on, we can identify MORE CONTAINERS.
* modalHeader, modalBody, modalFooter
* We can can KEEP BREAKING THIS MODAL DOWN into SUBCOMPONENTS



```
<div class="modal">
  <div class="modalHeader">
  </div>

  <div class="modalBody">
  </div>

  <div class="modalFooter">
  </div>
</div>
```

Modal component
is composed of modal subcomponents

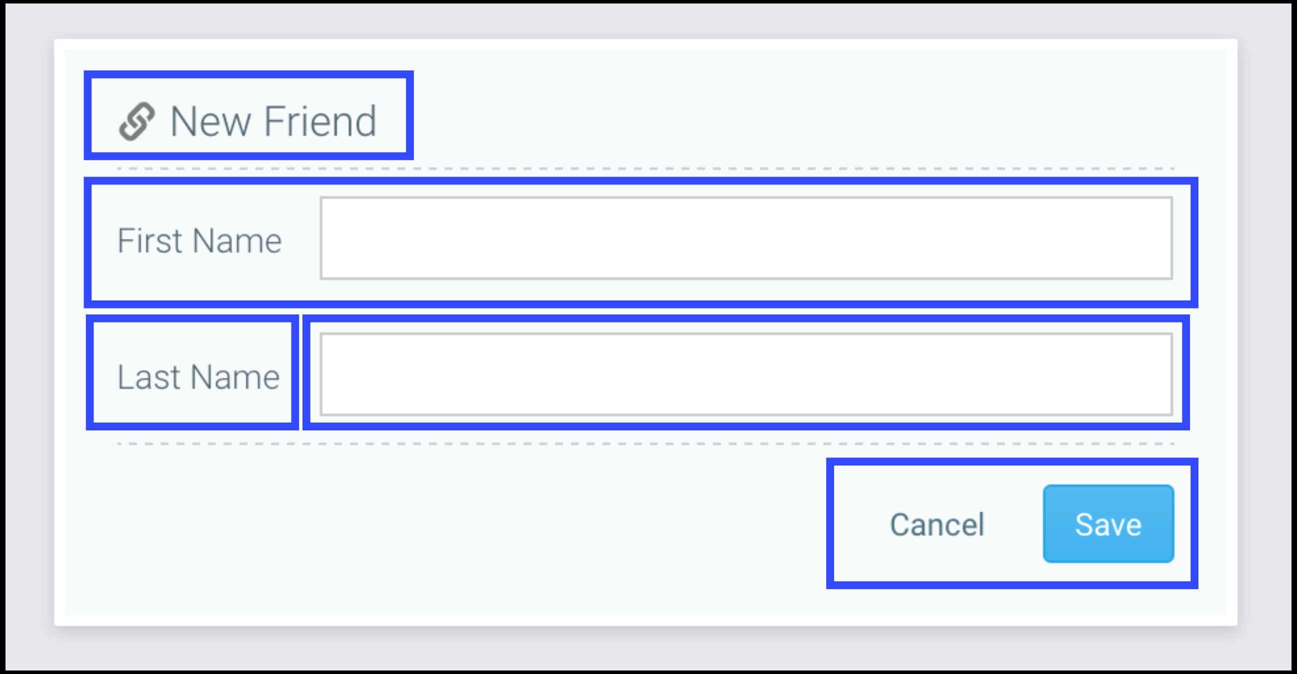
- We can see that it’s COMPOSED of subcomponents
- And what’s GREAT is that the CONNECTION between markup and CSS is clear
- We can read the markup and see that we have a modal comprised of a header, body, and footer



```
.modalHeader {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin: 20px 20px 10px;
  padding-bottom: 10px;
  border-bottom: 1px dashed #cbd3d7;
}

.modalBody {
  padding: 0 20px;
}

.modalFooter {
  padding: 0 20px 20px;
}
```

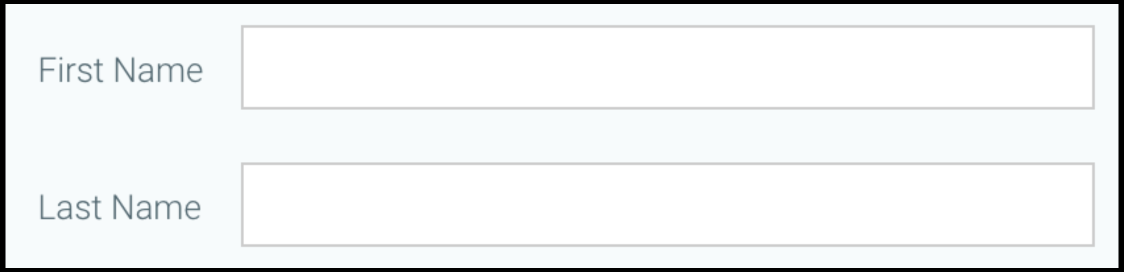
Each subcomponent has a single responsibility

* THEIR STYLES WON'T BE INHERITED BY THEIR CHILDREN
* Children don't need to worry about overriding any styles
* These are MORE DUMB CONTAINERS that we're going to FILL WITH CHILDREN
* And each subcomponent has a SINGLE RESPONSIBILITY (talk about them)



Great candidates for reuse

* Now that we get to the CHILDREN inside the MODAL HEADER, BODY, and FOOTER
* We see they’re not really modal-specific
* Definitely great candidates for REUSE

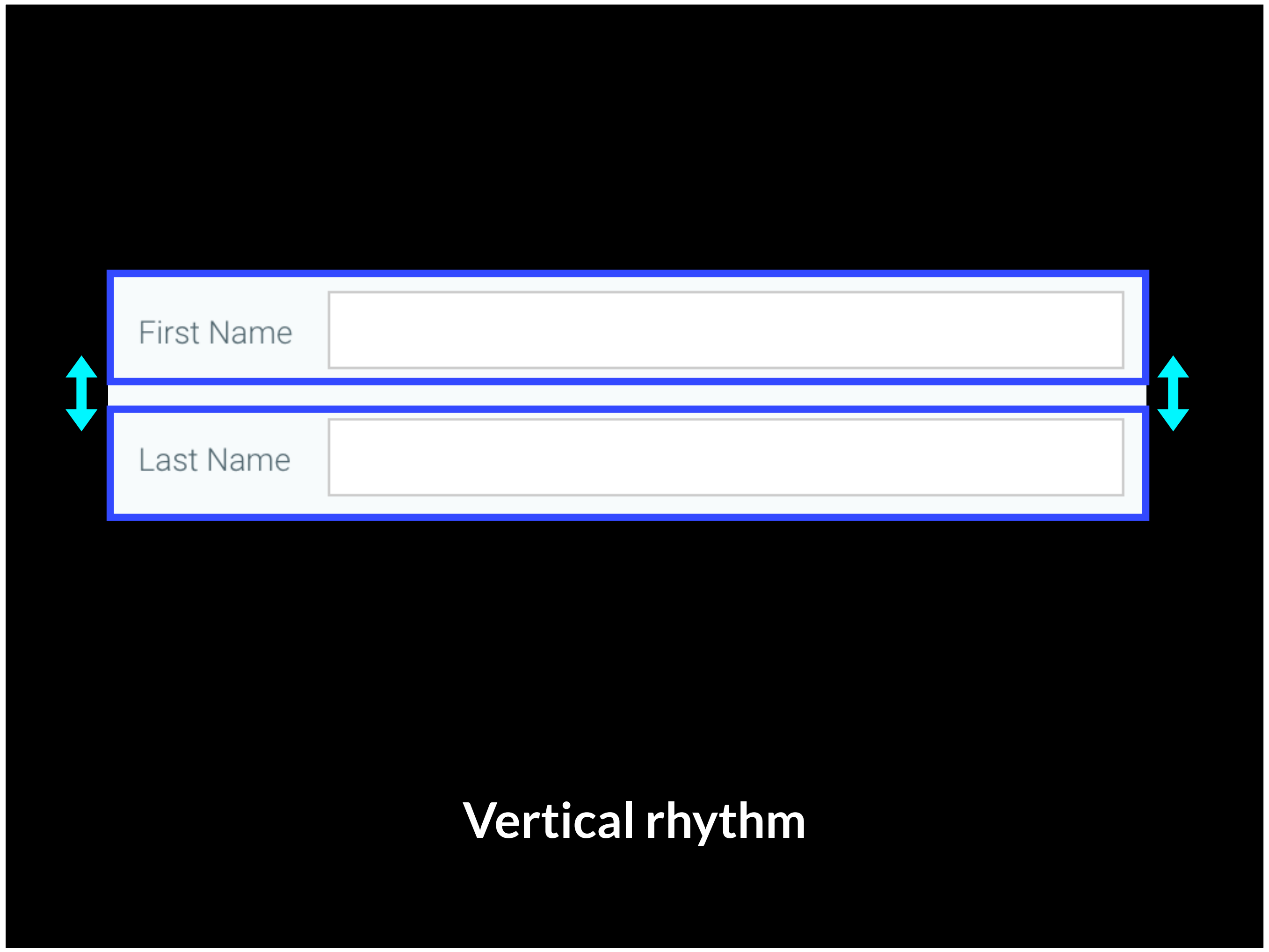


First Name

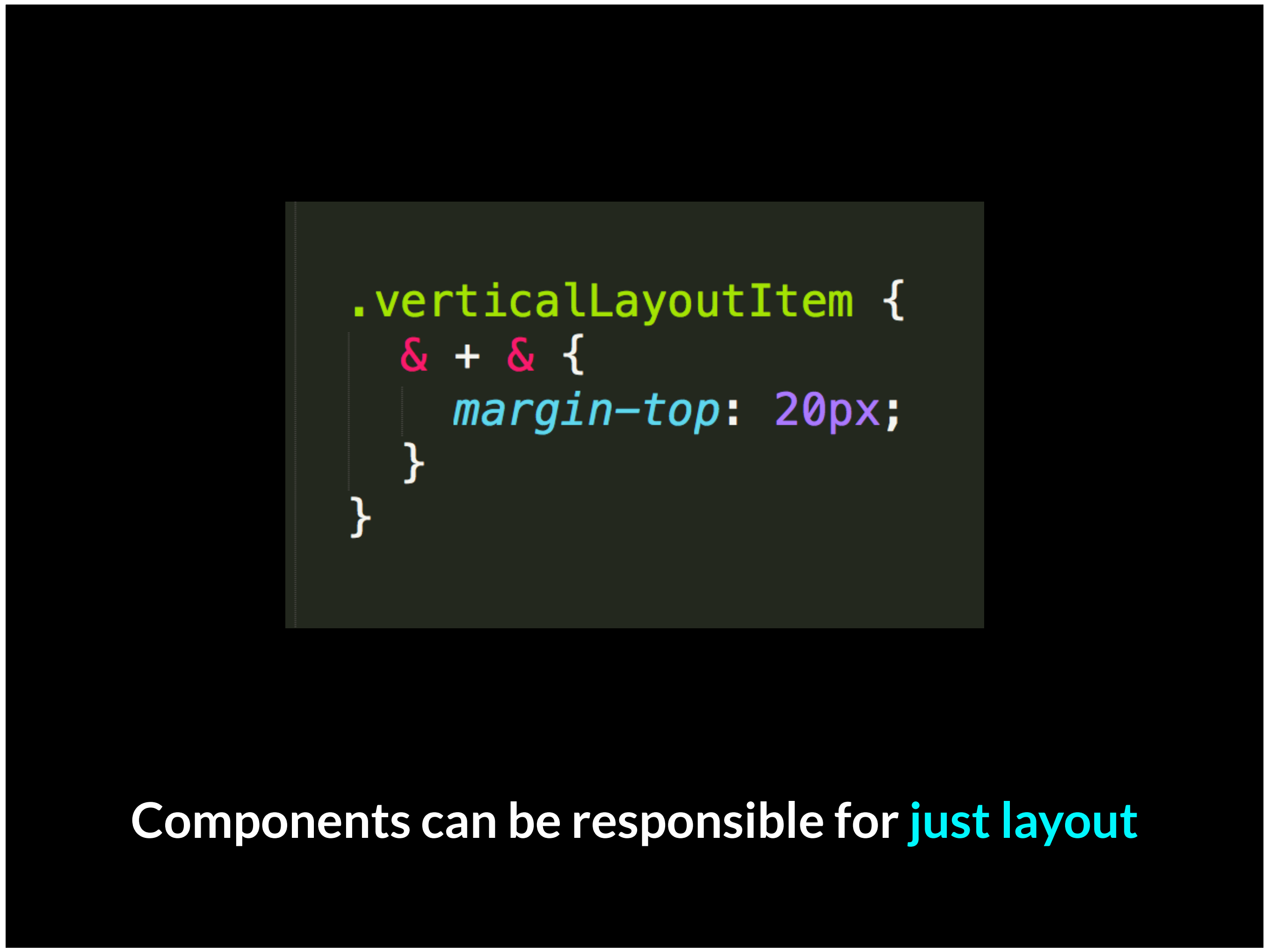
Last Name

What’s reusable about this?

- Let’s look at the FORM.
- What’s REUSABLE about this.
- Sure there’s a label, a text input… those can definitely be components.
- But there’s something else here.
- Something invisible that isn’t related to forms…



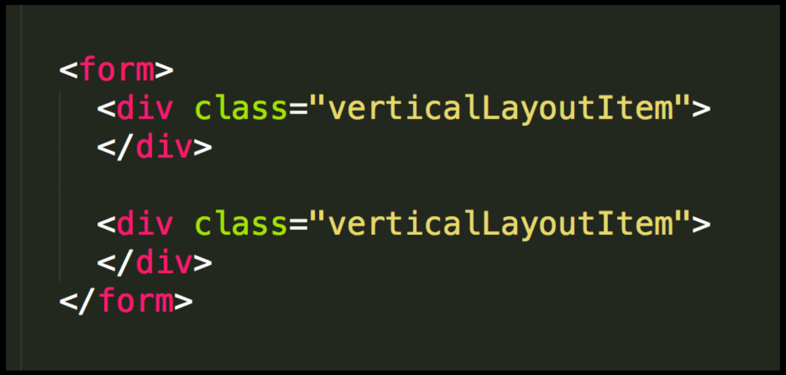
- It's the LAYOUT of these form elements.
- Vertical rhythm between them
- We don't want the labels and text input to be responsible for having their own vertical rhythm. We'd have to add a margin-bottom to them, and that means we'd have to add a margin-bottom to EVERY component that needs to support vertical rhythm.
- Instead we can create a custom component that will do this for us.
-



```
.verticalLayoutItem {
  & + & {
    margin-top: 20px;
  }
}
```

Components can be responsible for just layout

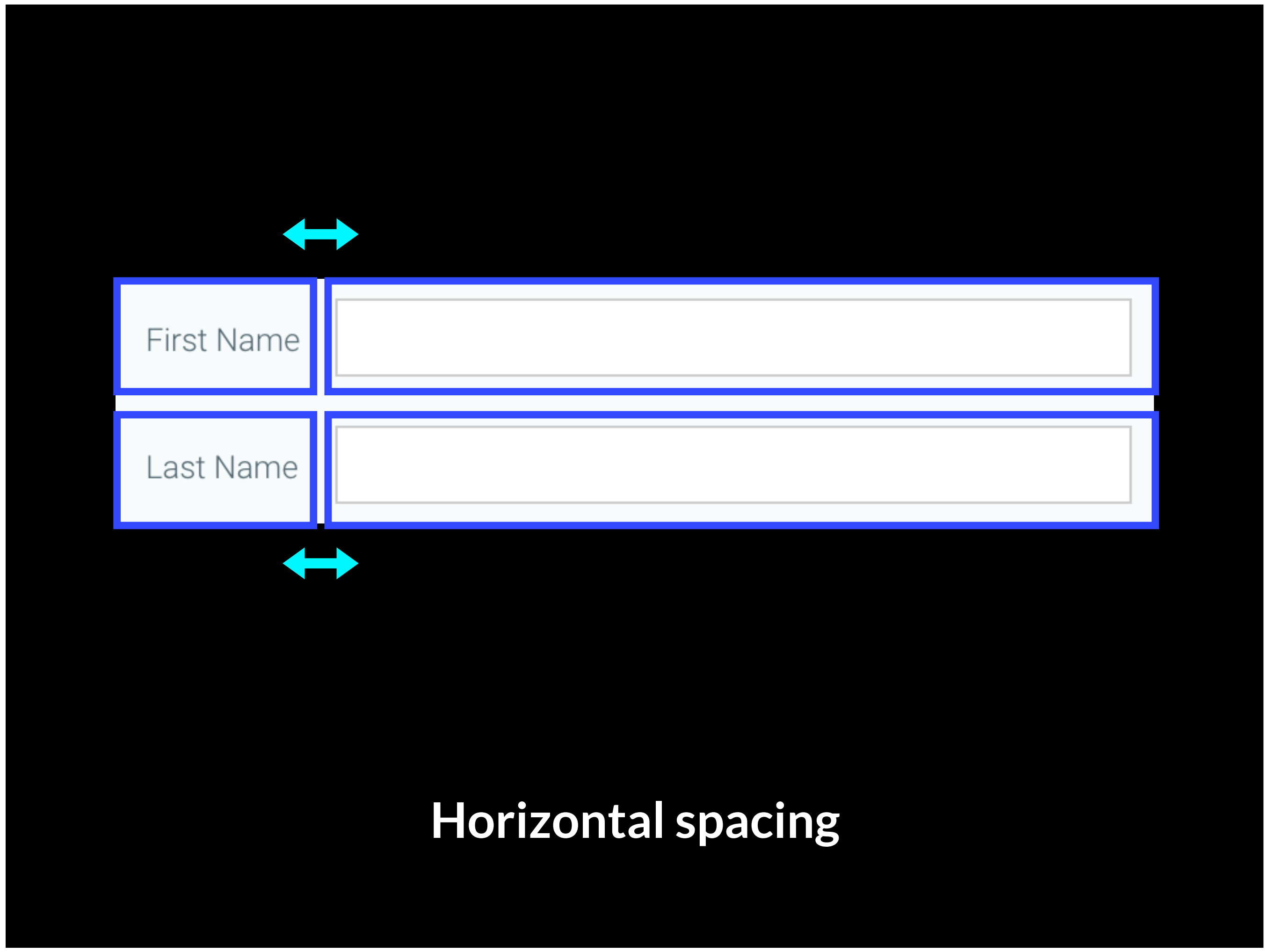
* verticalLayoutItem component
* ALL IT DOES: create vertical rhythm by using margin-bottom.
* THAT'S IT!
* This can be reused in forms everywhere, or to space out BODIES of text
* Very clear role in UI. All it's responsible for is LAYOUT
* And it's about the dumbest container you can HAVE. Can't even SEE it. But it can contain any kind of content you can imagine.



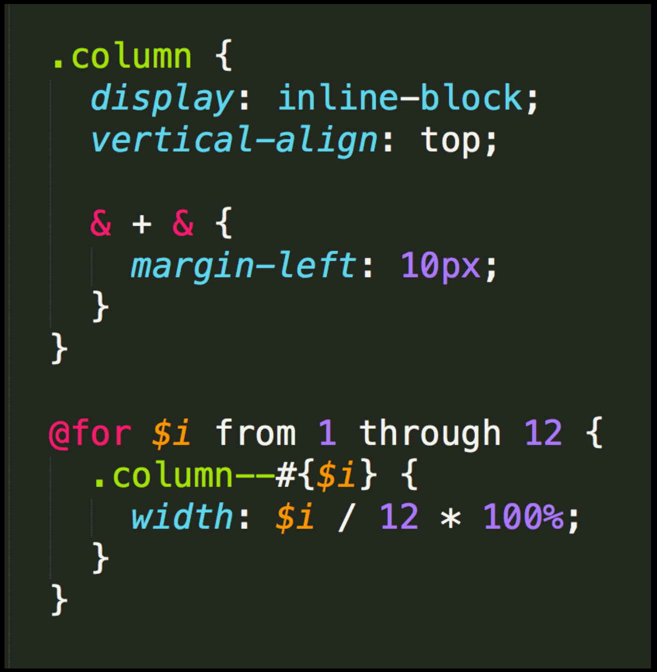
```
<form>
  <div class="verticalLayoutItem">
  </div>

  <div class="verticalLayoutItem">
  </div>
</form>
```

The form is composed
of verticalLayoutItem components

* The form is composed of verticalLayoutItems
* Role in UI is clear
*



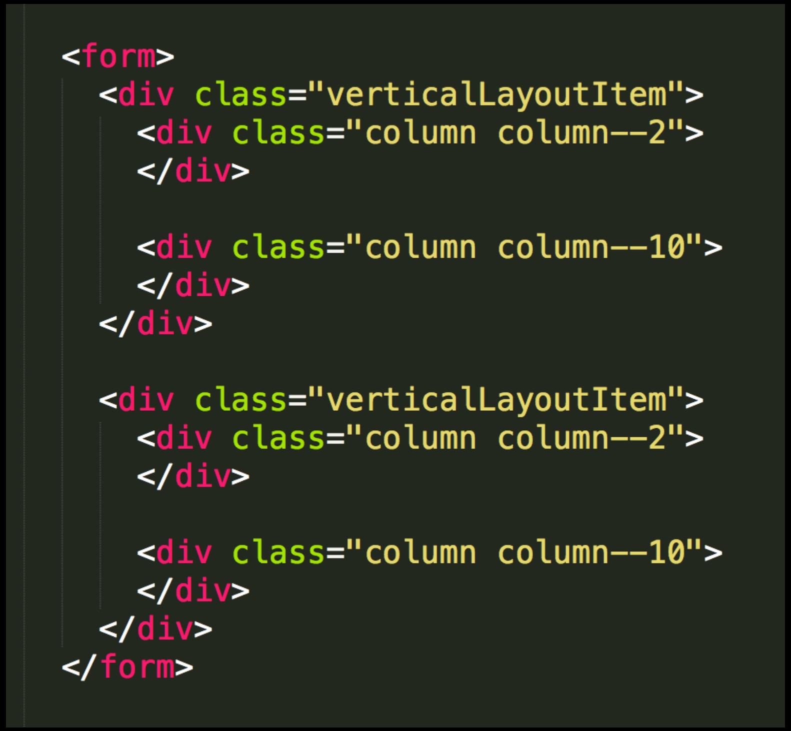
- Inside of each verticalLayoutRhythm div
- Horizontal spacing



```scss
.column {
  display: inline-block;
  vertical-align: top;

  & + & {
    margin-left: 10px;
  }
}

@for $i from 1 through 12 {
  .column--#{$i} {
    width: $i / 12 * 100%;
  }
}
```

Responsible for just layout

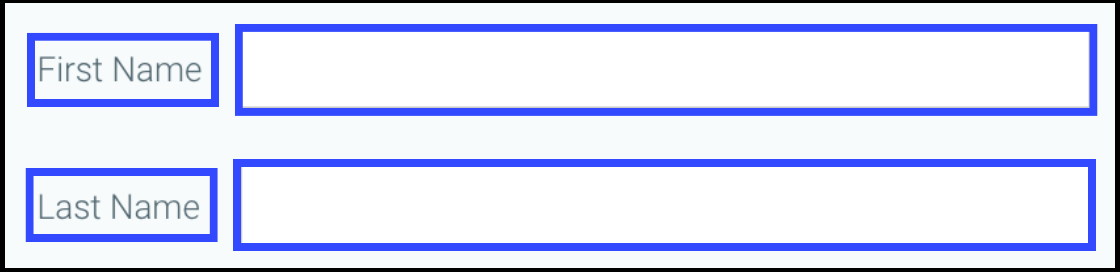
* Column classes that define certain widths and a spacing between them
* 12 different column classes, each representing a different width
* 10px between them
* Column component, which essentially substitutes for a grid system.



```
<form>
  <div class="verticalLayoutItem">
    <div class="column column--2">
    </div>

    <div class="column column--10">
    </div>
  </div>

  <div class="verticalLayoutItem">
    <div class="column column--2">
    </div>

    <div class="column column--10">
    </div>
  </div>
</form>
```

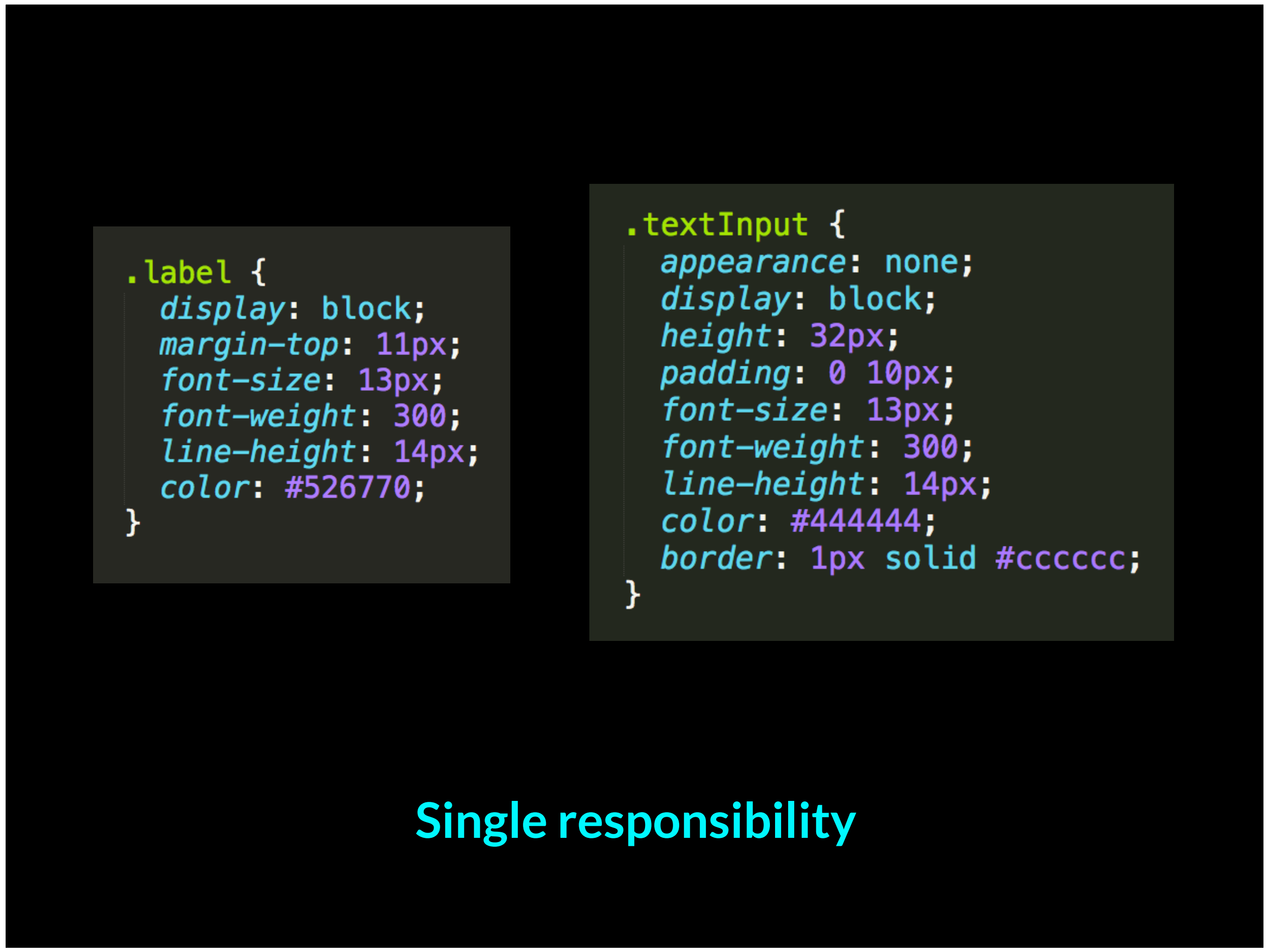
verticalLayoutItem components are composed of column components

* The form is composed of verticalLayoutItems,
* Which are composed of columns
* MORE DUMB CONTAINERS inside of DUMB CONTAINERS
* Just layout, now we can fill this layout with our content



Label and TextInput components

* Label and textInput components
* LEAF COMPONENTS in the COMPONENT TREE
* They don’t contain anything

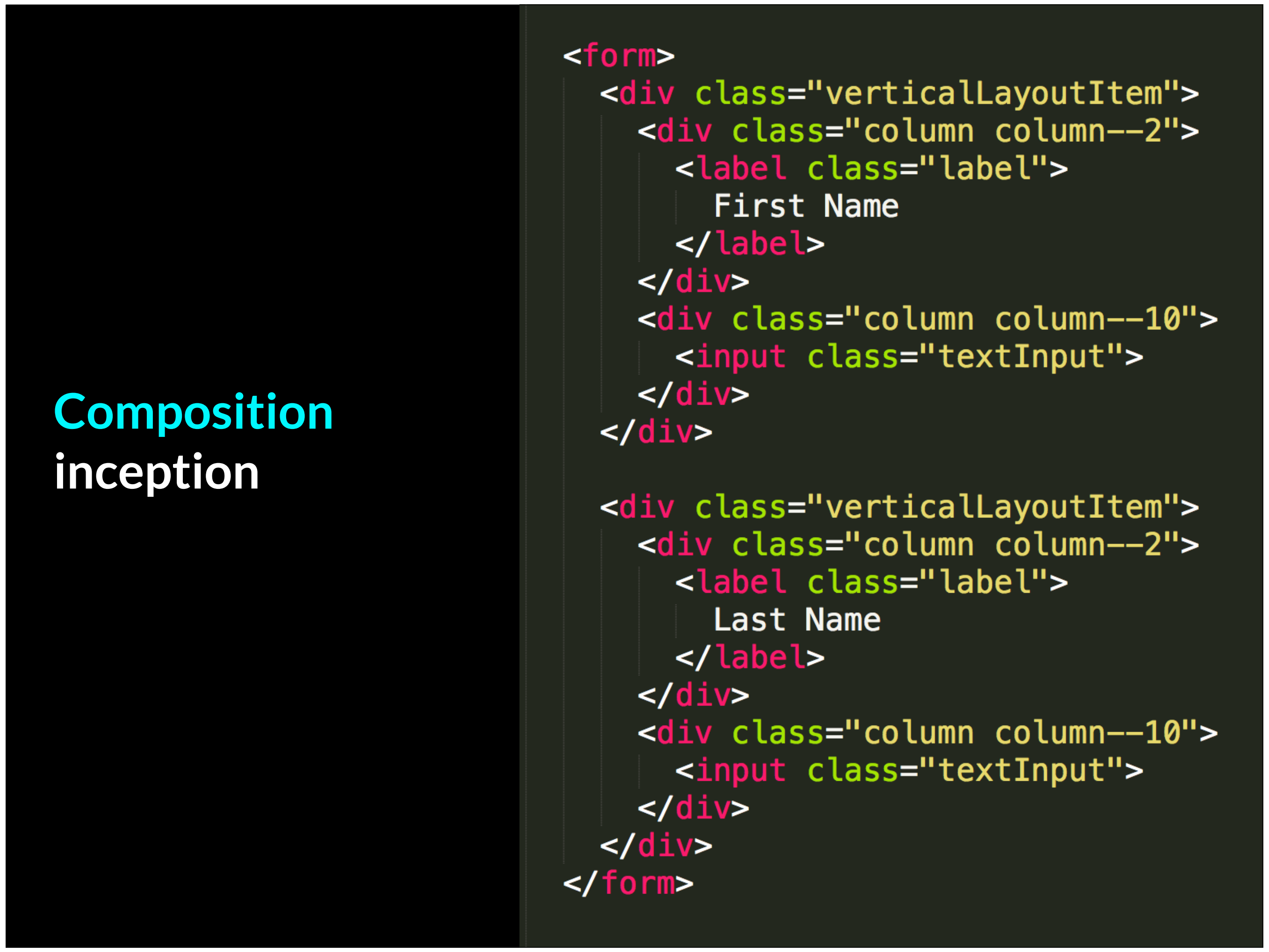


```
.label {
  display: block;
  margin-top: 11px;
  font-size: 13px;
  font-weight: 300;
  line-height: 14px;
  color: #526770;
}
```

```
.textInput {
  appearance: none;
  display: block;
  height: 32px;
  padding: 0 10px;
  font-size: 13px;
  font-weight: 300;
  line-height: 14px;
  color: #444444;
  border: 1px solid #cccccc;
}
```

Single responsibility

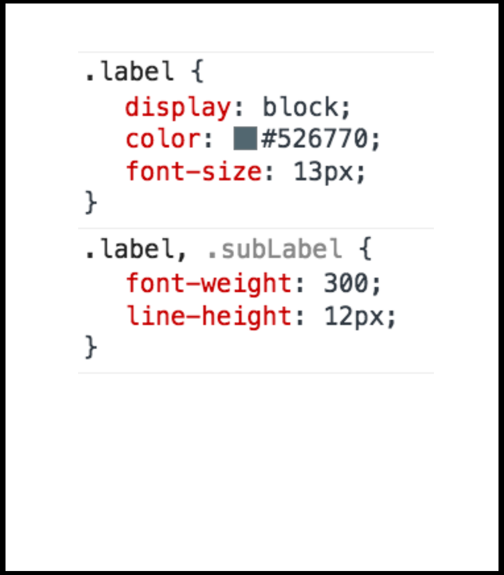
* Styles that govern appearance, and NOT LAYOUT.
* Font-size, color, font-weight, and line-height



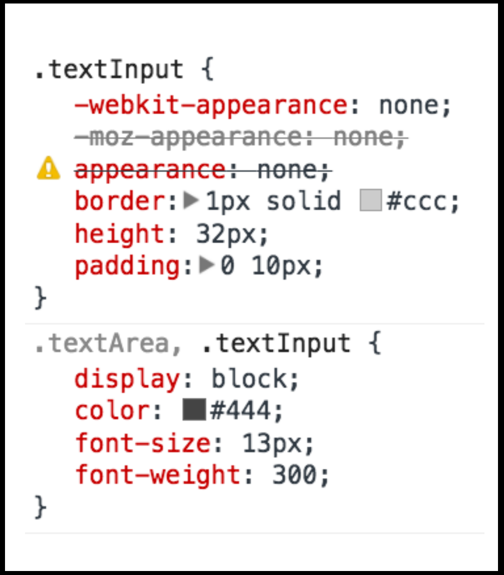
Composition
inception

```
<form>
  <div class="verticalLayoutItem">
    <div class="column column--2">
      <label class="label">
        First Name
      </label>
    </div>
    <div class="column column--10">
      <input class="textInput">
    </div>
  </div>

  <div class="verticalLayoutItem">
    <div class="column column--2">
      <label class="label">
        Last Name
      </label>
    </div>
    <div class="column column--10">
      <input class="textInput">
    </div>
  </div>
</form>
```

* When we read the markup
* Layer and layer of composition
* Each component is a CONTAINER, composed of more components
* Until we get down to our lead nodes, the content that makes our layout a true user interface



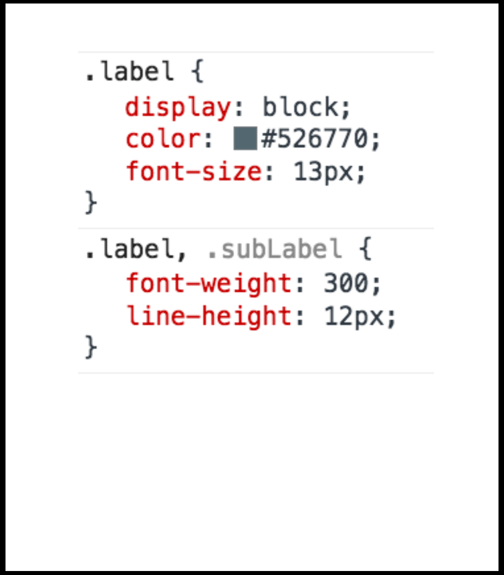
```
.label {
   display: block;
   color: ■#526770;
   font-size: 13px;
}
.label, .subLabel {
   font-weight: 300;
   line-height: 12px;
}
```



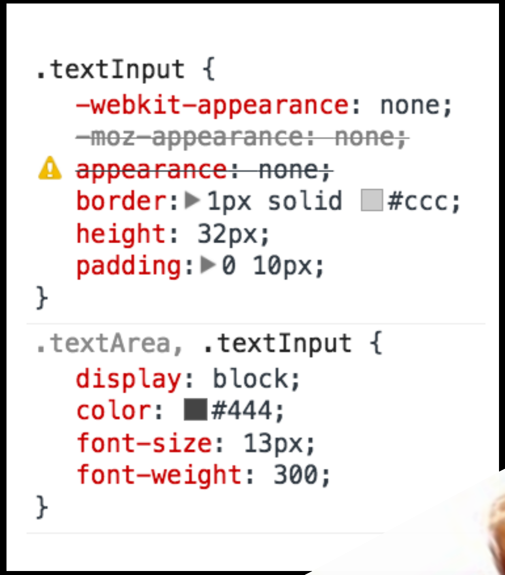
```
.textInput {
   -webkit-appearance: none;
   ~~-moz-appearance: none;~~
⚠ ~~appearance: none;~~
   border:▶1px solid ■#ccc;
   height: 32px;
   padding:▶0 10px;
}
.textArea, .textInput {
   display: block;
   color: ■#444;
   font-size: 13px;
   font-weight: 300;
}
```

Selectors have clear roles in UI
No overridden styles

* Take a look in the inspector at the leaf elements in our component tree
* Looking at each selector, it's obvious what it does. You know exactly what it's role is in the UI
* Isn't it refreshing to see no overriding styles? MAN that's great.



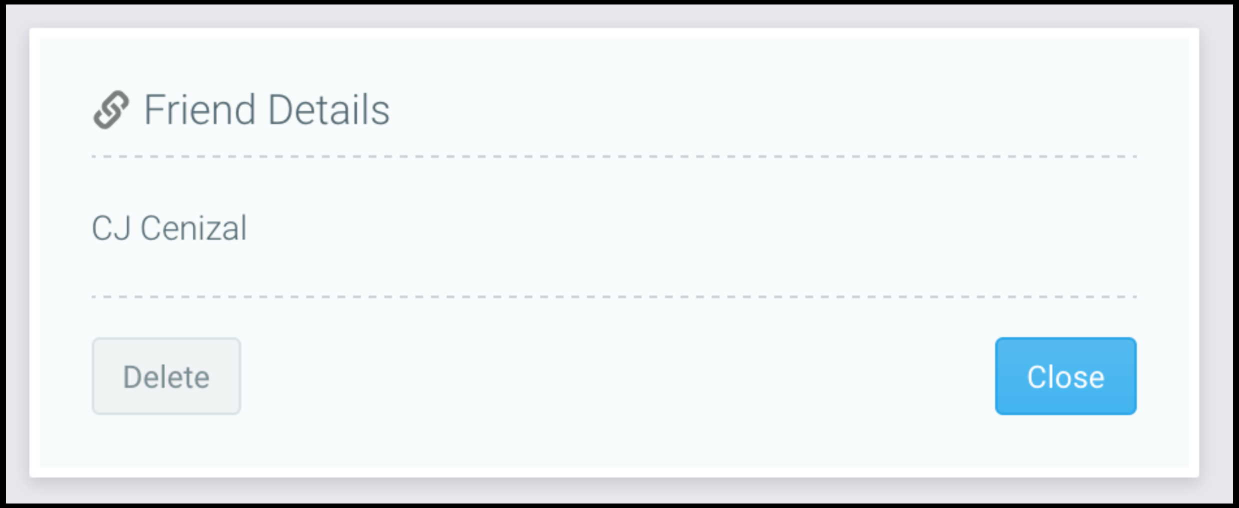
```
.label {
   display: block;
   color: ■#526770;
   font-size: 13px;
}
.label, .subLabel {
   font-weight: 300;
   line-height: 12px;
}
```



```
.textInput {
   -webkit-appearance: none;
   ~~-moz-appearance: none;~~
⚠  ~~appearance: none;~~
   border:▶1px solid □#ccc;
   height: 32px;
   padding:▶0 10px;
}
.textArea, .textInput {
   display: block;
   color: ■#444;
   font-size: 13px;
   font-weight: 300;
}
```

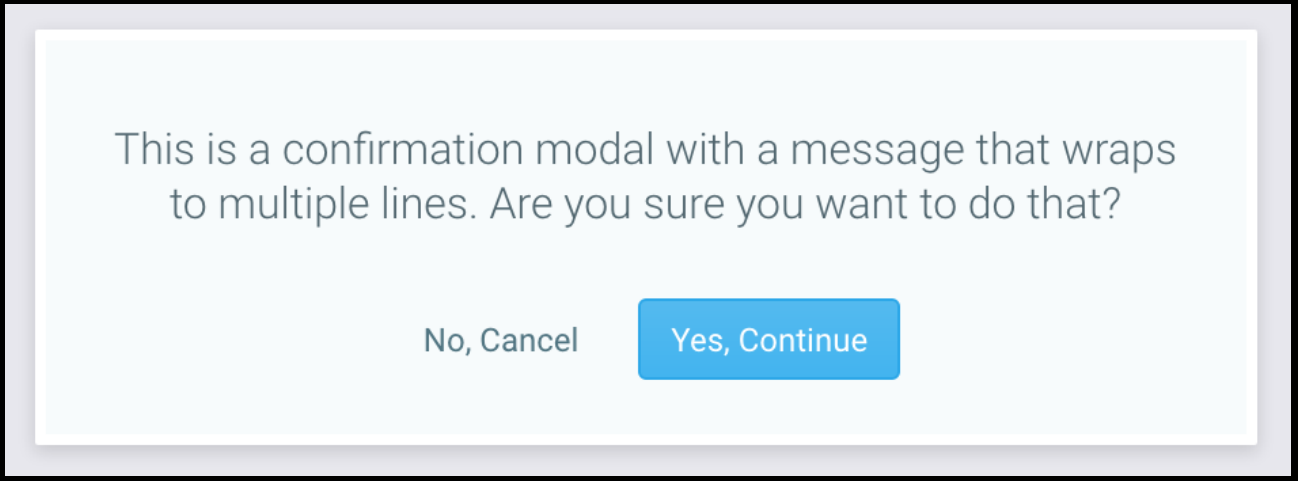
No place for you here,
cheeky reindeer mug!

* SO much easier to understand compared to the CHEEKY REINDEER MUG CSS!



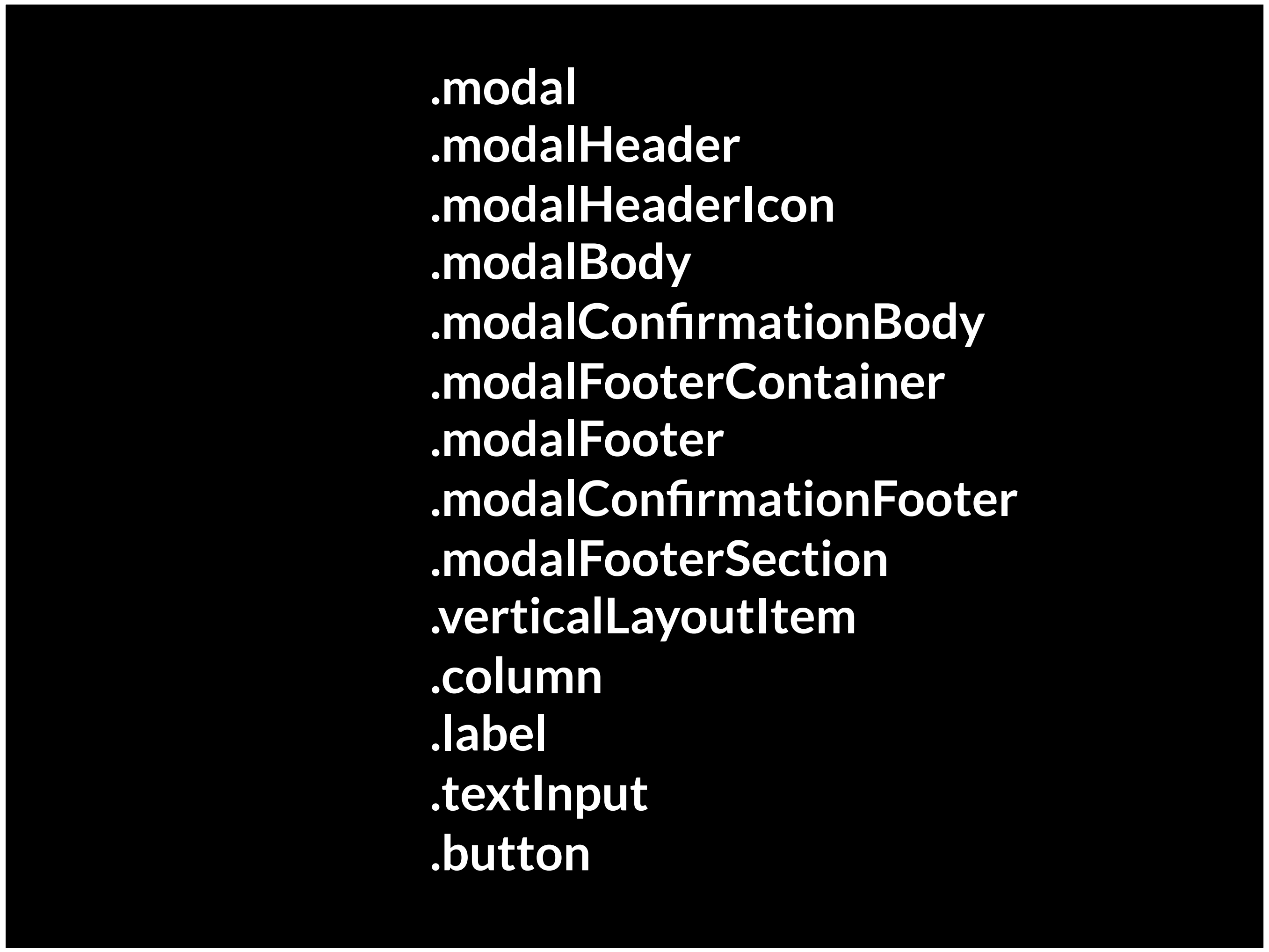
Components can be reused for a Friend Details modal

Components can be used to create different kinds of modals


This is a confirmation modal with a message that wraps to multiple lines. Are you sure you want to do that?

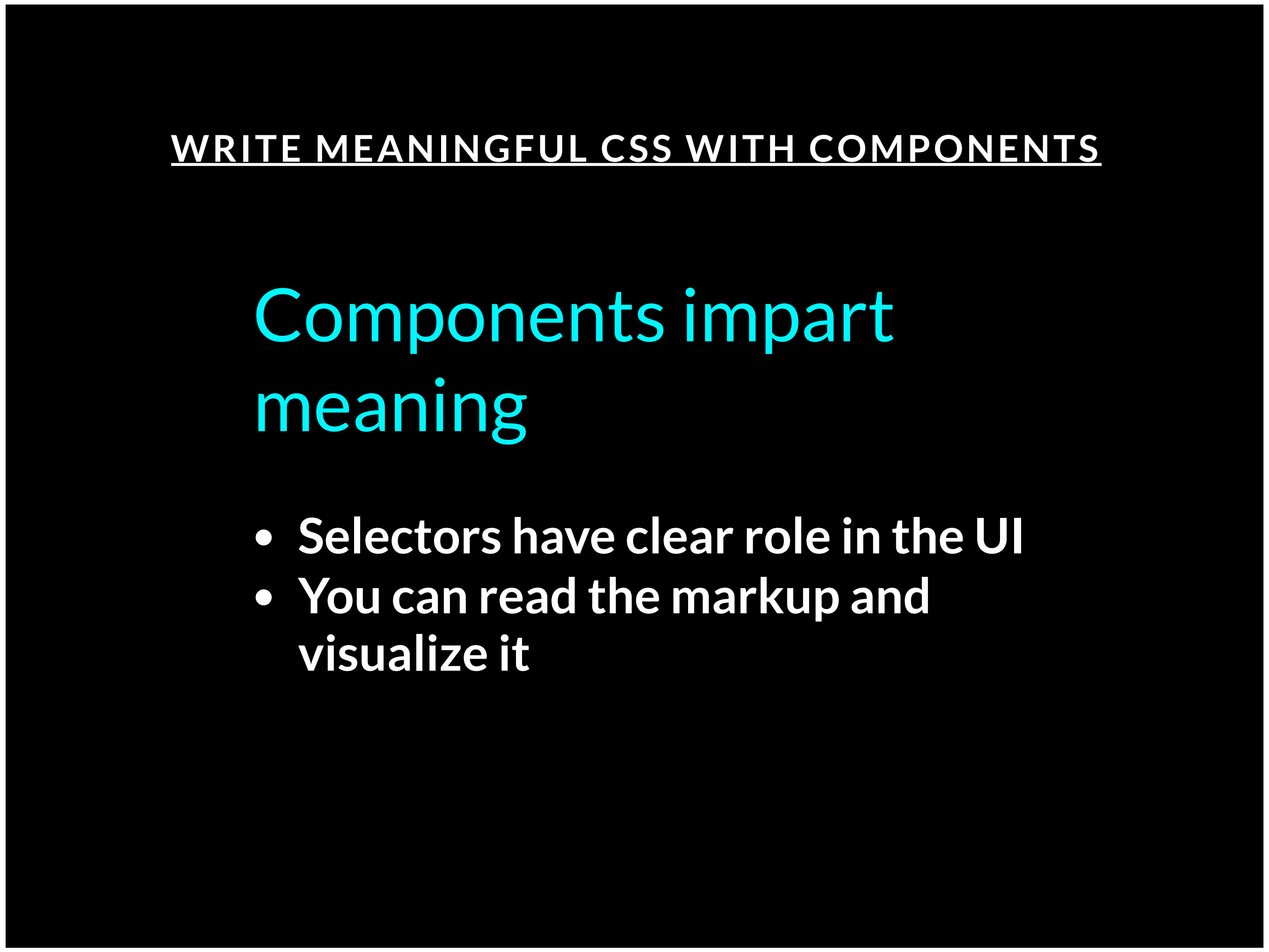
Components can be reused for a Confirmation modal

* Can also be used outside of modals
*



.modal
.modalHeader
.modalHeaderIcon
.modalBody
.modalConfirmationBody
.modalFooterContainer
.modalFooter
.modalConfirmationFooter
.modalFooterSection
.verticalLayoutItem
.column
.label
.textInput
.button

* Component-based approach results in a clear library of building blocks for your UI
* Forms, for navigation, for any column-based layouts.

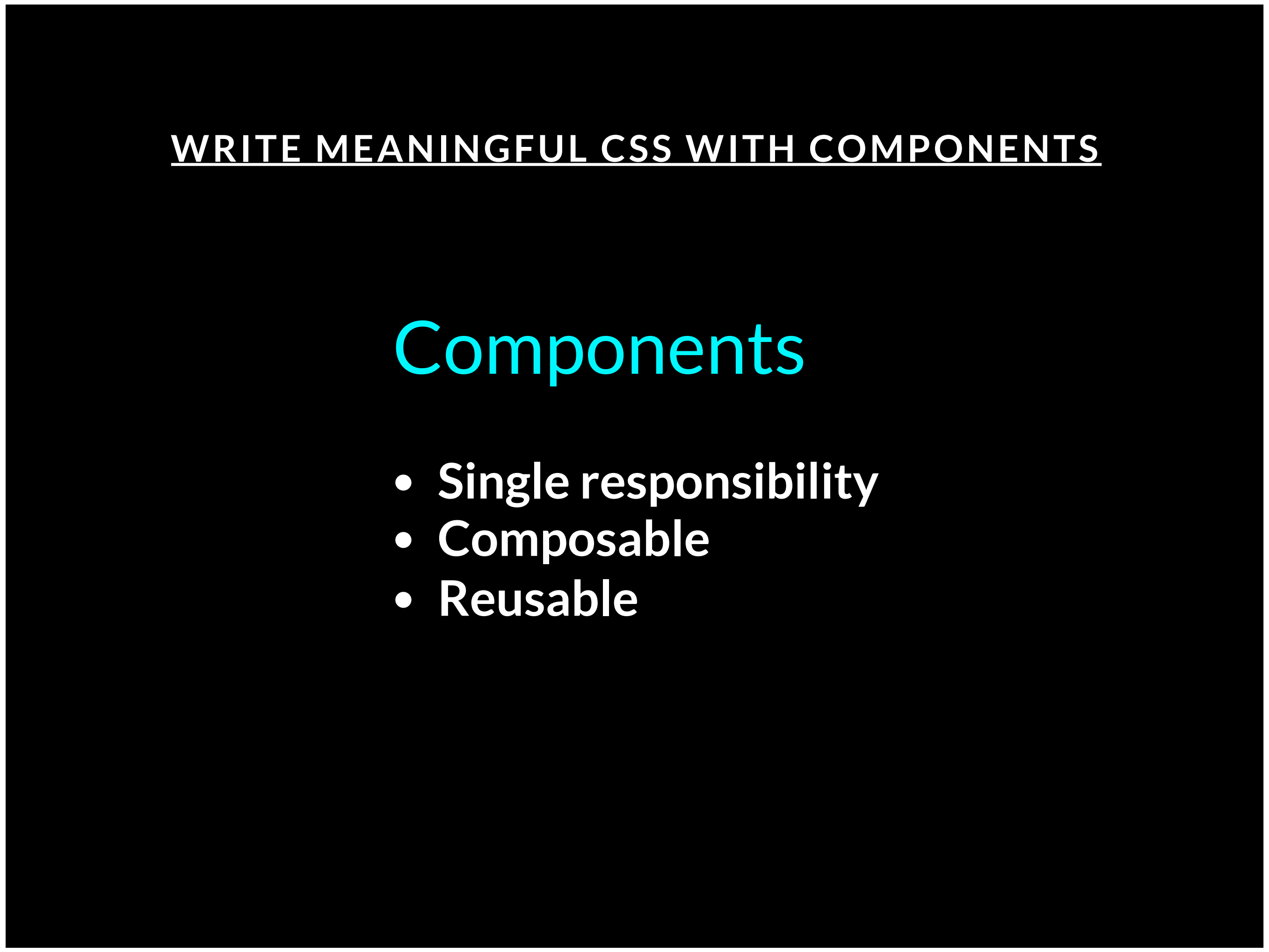


WRITE MEANINGFUL CSS WITH COMPONENTS

Components impart meaning

- **Selectors have clear role in the UI**
- **You can read the markup and visualize it**

* Selectors have clear role in the UI
* You can read the markup, see the components, and visualize how the UI will look without needing to actually open up the browser

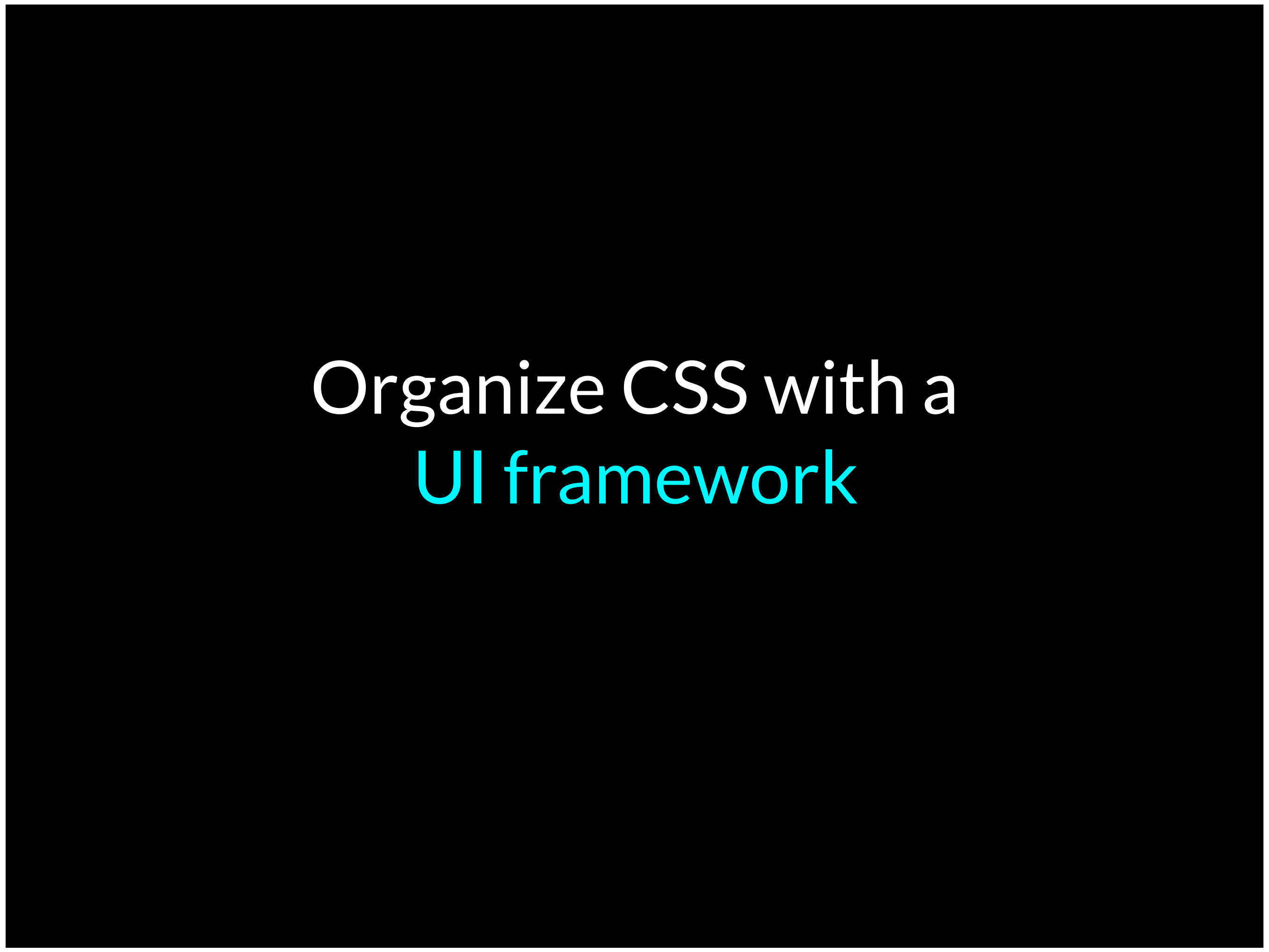


WRITE MEANINGFUL CSS WITH COMPONENTS

Components

- Single responsibility
- Composable
- Reusable

* And when you write your components, remember that each component
* should have a single responsibility
* be composable
* and reusable.



Organize CSS with a UI framework

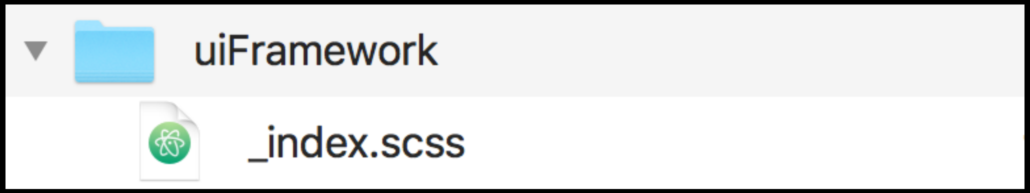
- Once you’ve written CSS to be simple using classes and meaningful using components
- You need a way to organize it
- That’s what the purpose of the UI framework is for
- It’s your body of CSS, organized so that it’s easy to know where to find things, where and how to add things

ORGANIZE CSS WITH A UI FRAMEWORK

- File and folder structure
- Class naming convention
- CSS preprocessor tools

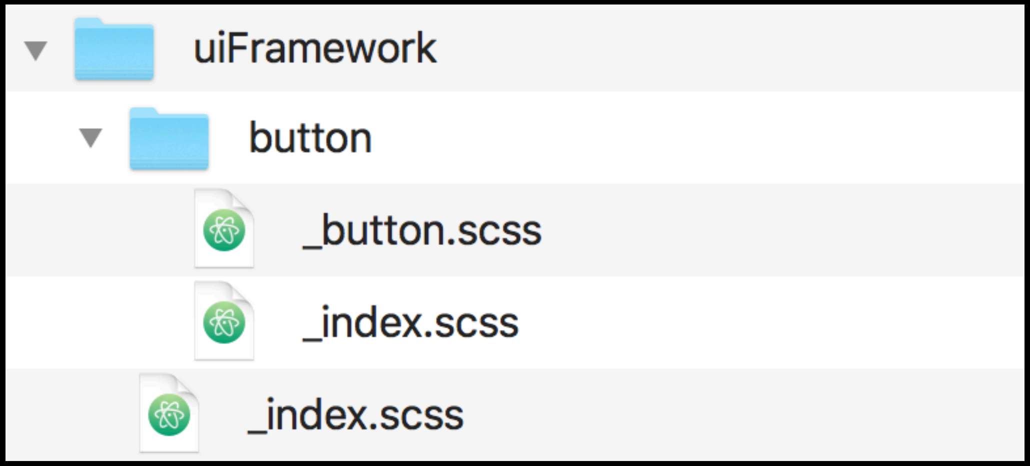
ORGANIZE CSS WITH A UI FRAMEWORK

1. File and folder structure



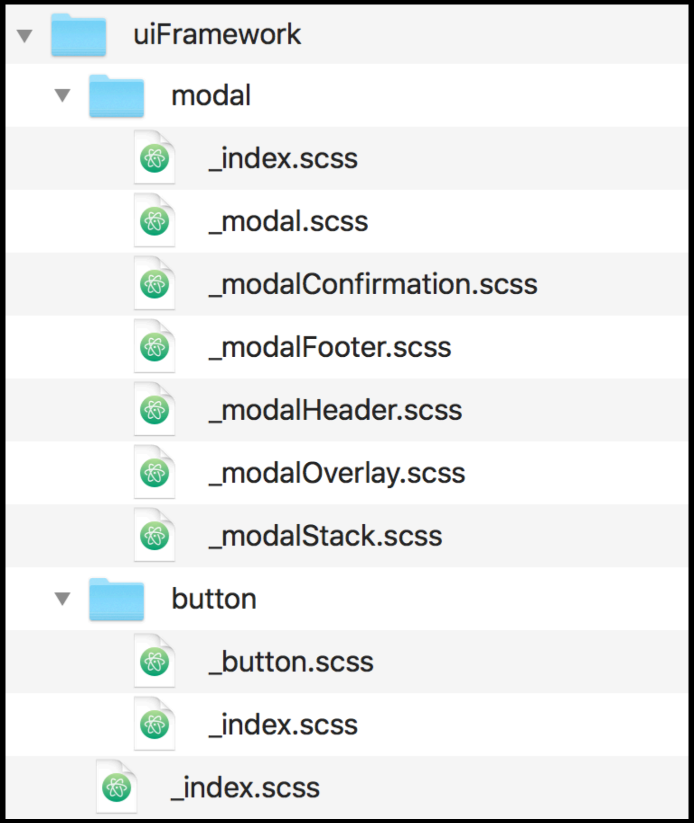
UI framework folder is easy to find

- I recommend putting your CSS into a single UI framework folder, even if it’s a child folder inside of a project that USES the UI framework.
- This helps separate the REUSABLE, COMPOSABLE components from any project-specific CSS which may not yet be written in the same ORGANIZED STYLE as the UI framework.
- An index SCSS file at the root level. This will IMPORT all of the components you’ll put in the folder.


```
▼ uiFramework
    ▼ button
          _button.scss
          _index.scss
      _index.scss
```

Each component gets its own folder

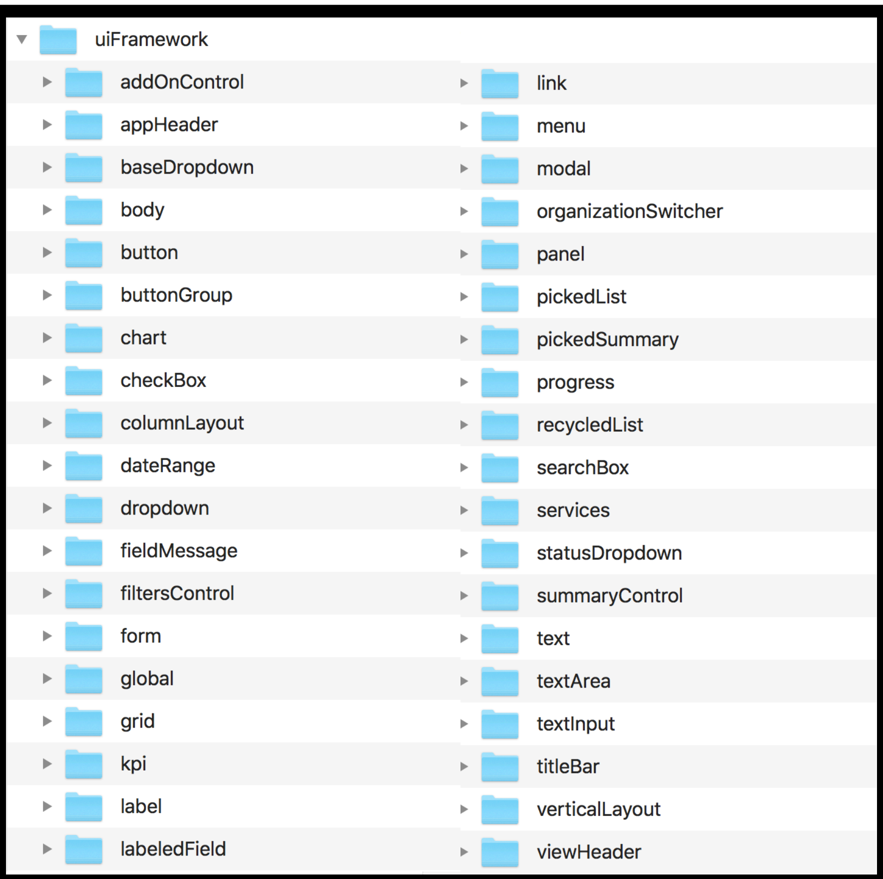
* I like to organize my components into folders. Each folder is named after the core component.
* Each component gets its own SCSS file.
* An index SCSS file imports all of the components in the folder.
* The root index SCSS file imports all of the component index.scss files.
* May seem like a lot of BOILERPLATE for a little component like the Button, but…



```
uiFramework
    modal
        _index.scss
        _modal.scss
        _modalConfirmation.scss
        _modalFooter.scss
        _modalHeader.scss
        _modalOverlay.scss
        _modalStack.scss
    button
        _button.scss
        _index.scss
    _index.scss
```

Folders scale as components grow

- …when you add more INVOLVED components, like the Modal
- it’s useful to be able to ENCAPSULATE all of these RELATED component files in ONE folder.

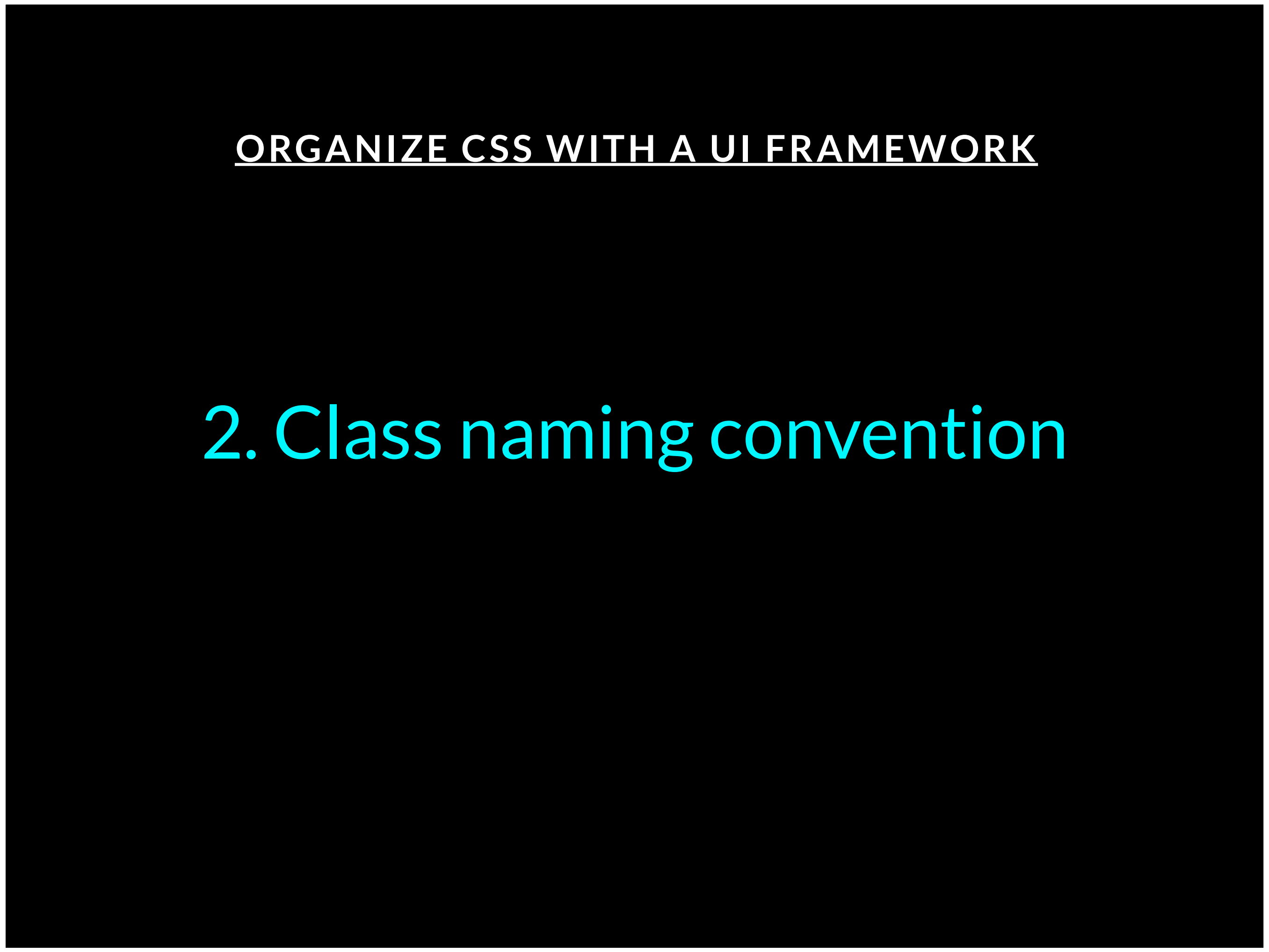


uiFramework

- addOnControl
- appHeader
- baseDropdown
- body
- button
- buttonGroup
- chart
- checkBox
- columnLayout
- dateRange
- dropdown
- fieldMessage
- filtersControl
- form
- global
- grid
- kpi
- label
- labeledField
- link
- menu
- modal
- organizationSwitcher
- panel
- pickedList
- pickedSummary
- progress
- recycledList
- searchBox
- services
- statusDropdown
- summaryControl
- text
- textArea
- textInput
- titleBar
- verticalLayout
- viewHeader

Life is good!

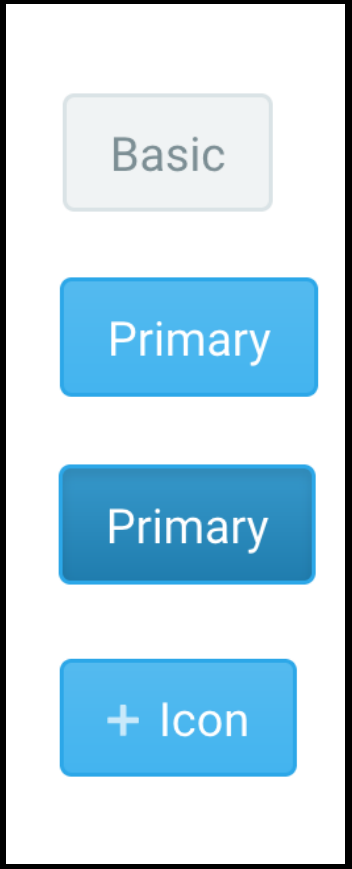
- And check out how COOL it is to have a UI framework full of AWESOME components!
- As a developer, I LOVE having a TOOLKIT of components available that make it easy for me to CONSTRUCT a UI.
- It’s a GREAT FEELING.


ORGANIZE CSS WITH A UI FRAMEWORK

2. Class naming convention

- A way to define TYPES of classes and how they're supposed work together
- STORY: The convention I like to use is BASICALLY BEM
- There are others like OOCSS, SMACSS, SUITCSS, even Bootstrap
- They share some SIMILARITIES.
- I've tried to keep mine as SIMPLE as possible

Example component: Button



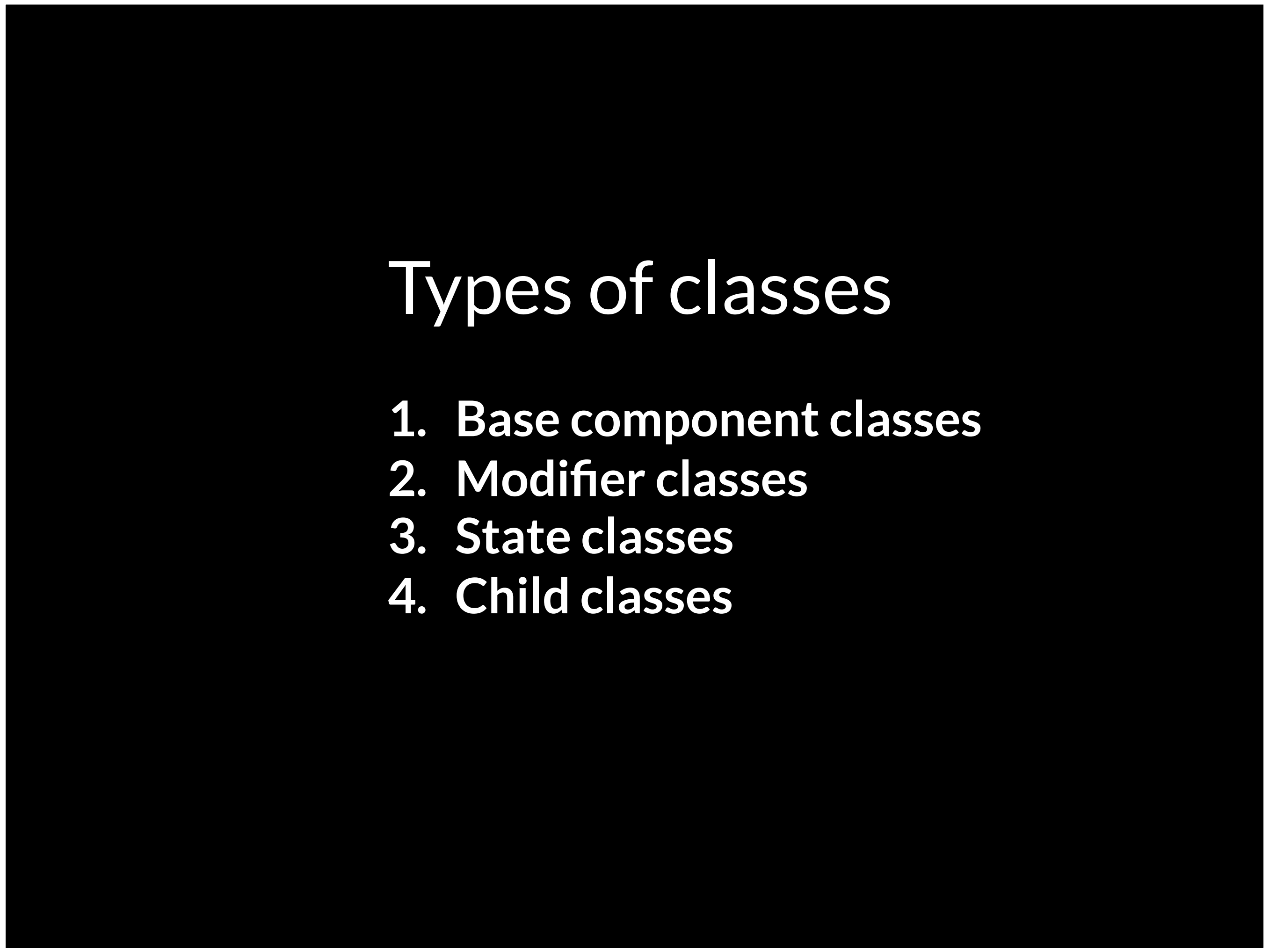
Basic button

Primary button

“Pressed” state

Optional icon

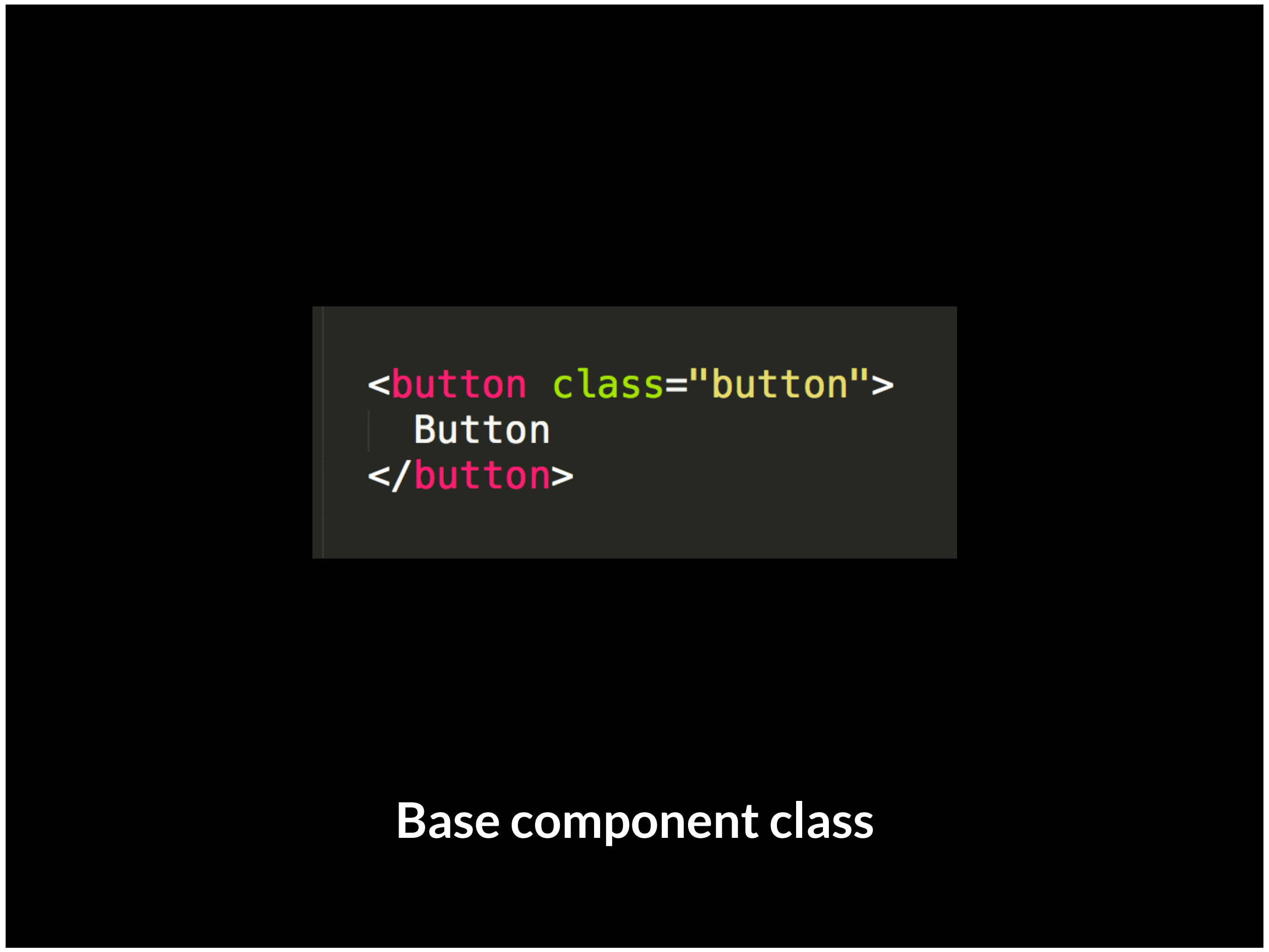
* Two different TYPES of button
* PRESSED state
* OPTION to display an icon INSIDE the button



Types of classes

1. **Base component classes**
2. **Modifier classes**
3. **State classes**
4. **Child classes**

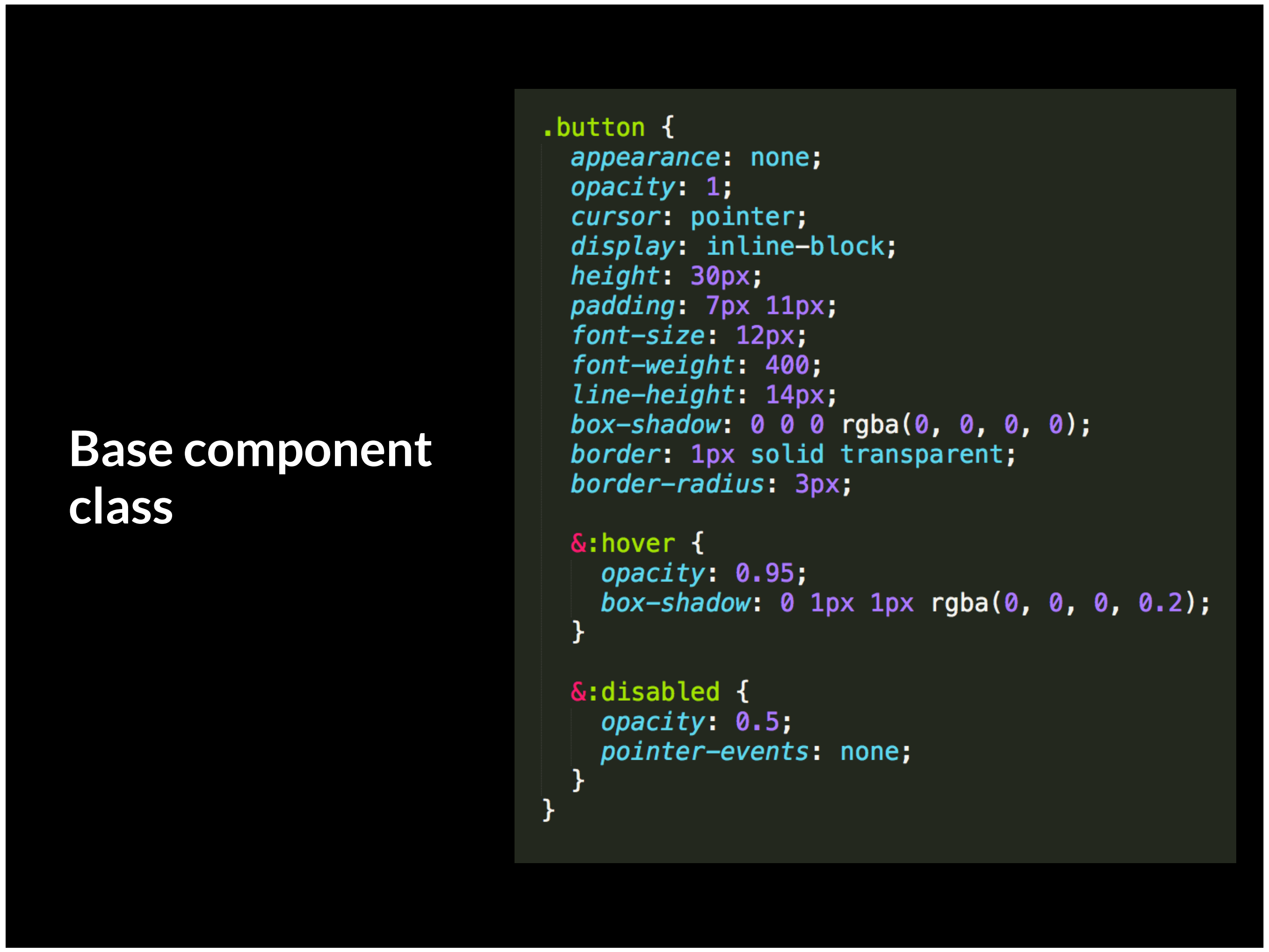
- We’re going to explore FOUR different types of classes
- Base component classes, modifier classes, state classes, and child classes
- I’ll show you how the naming convention helps CLEARLY IDENTIFY the role of each type of class



```html
<button class="button">
  Button
</button>
```

Base component class

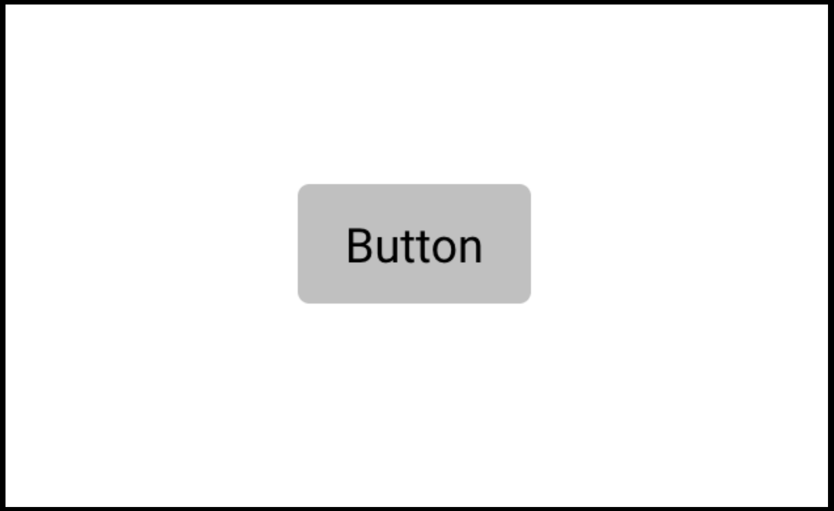
* BASE COMPONENT CLASS
* We’ll just call it button
* Nice and simple



Base component class

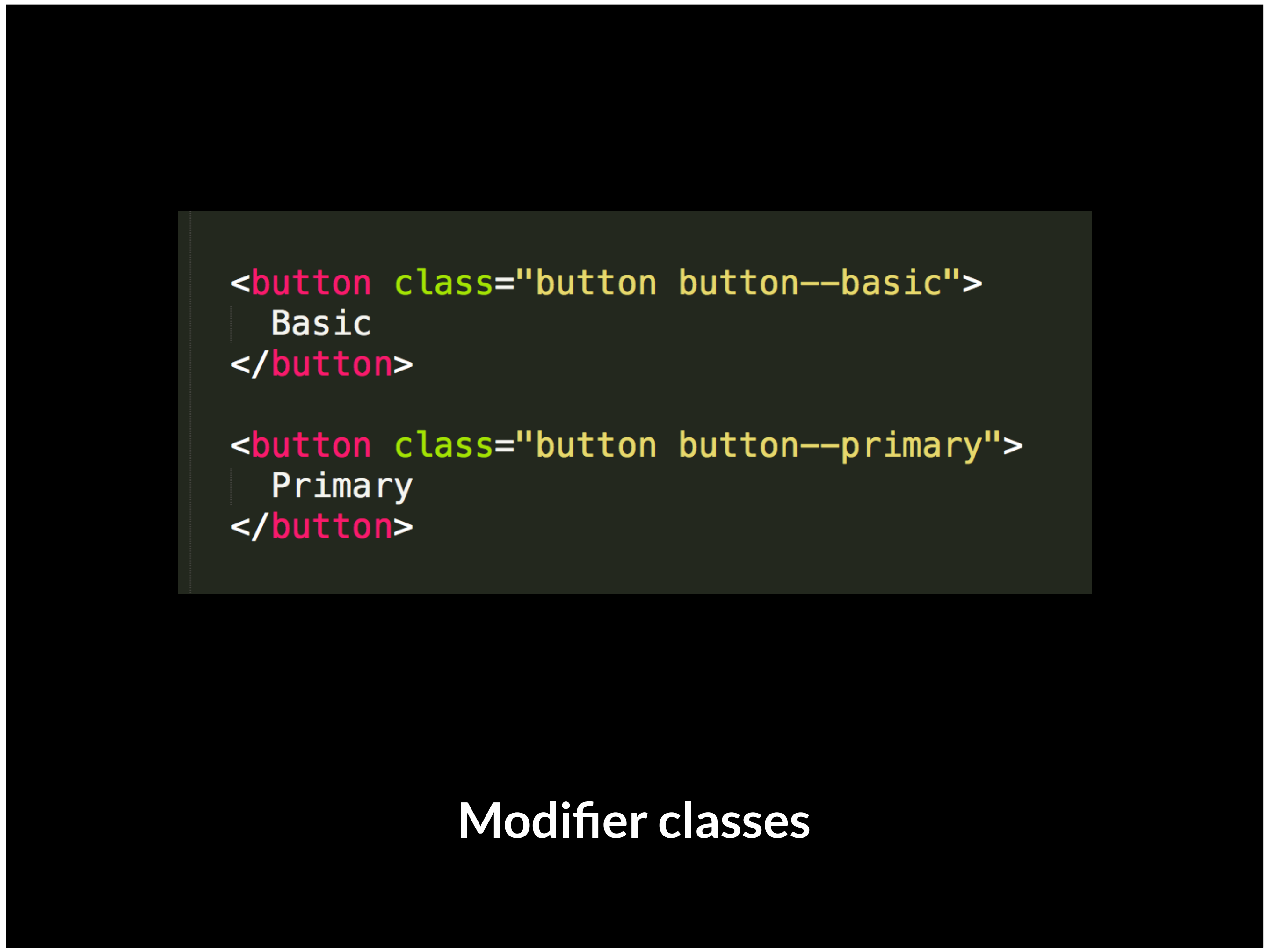
```
.button {
  appearance: none;
  opacity: 1;
  cursor: pointer;
  display: inline-block;
  height: 30px;
  padding: 7px 11px;
  font-size: 12px;
  font-weight: 400;
  line-height: 14px;
  box-shadow: 0 0 0 rgba(0, 0, 0, 0);
  border: 1px solid transparent;
  border-radius: 3px;

  &:hover {
    opacity: 0.95;
    box-shadow: 0 1px 1px rgba(0, 0, 0, 0.2);
  }

  &:disabled {
    opacity: 0.5;
    pointer-events: none;
  }
}
```

* The styles are used to establish the FOUNDATION of this component
* font, font-size, border width, border-radius
* BUT NO: colors, background-color, font color, no border color
* We know that DIFFERENT KINDS of buttons will need to CHANGE these PROPERTIES



Base component class

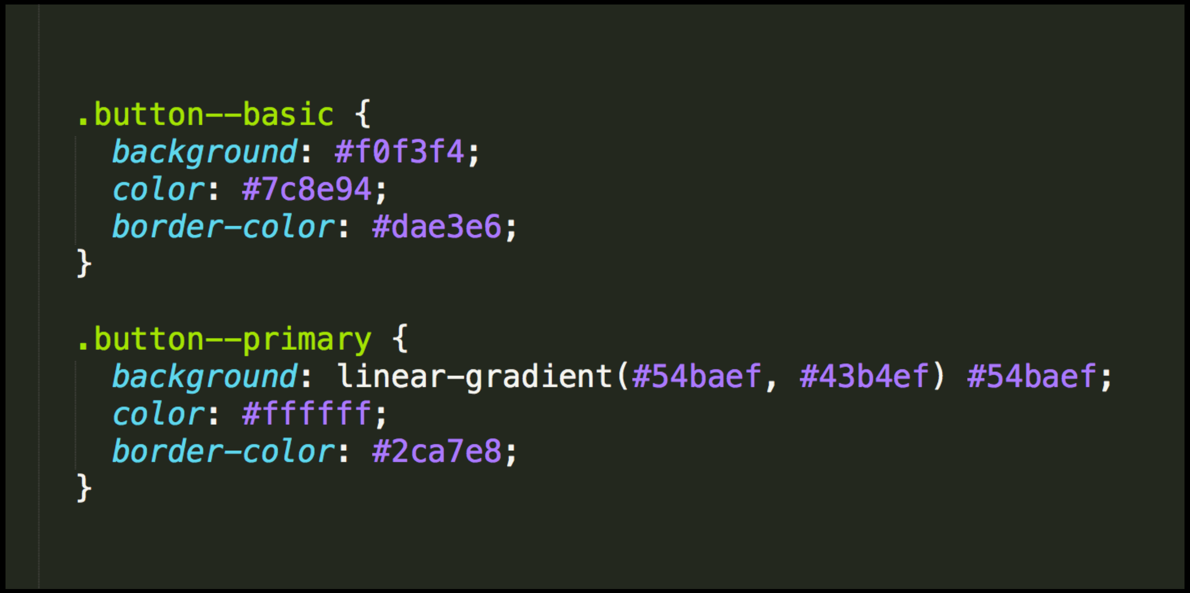
* And if we LOOK at the button in the BROWSER
* It looks PRETTY BORING
* But that’s OK…



```
<button class="button button--basic">
  Basic
</button>

<button class="button button--primary">
  Primary
</button>
```

Modifier classes

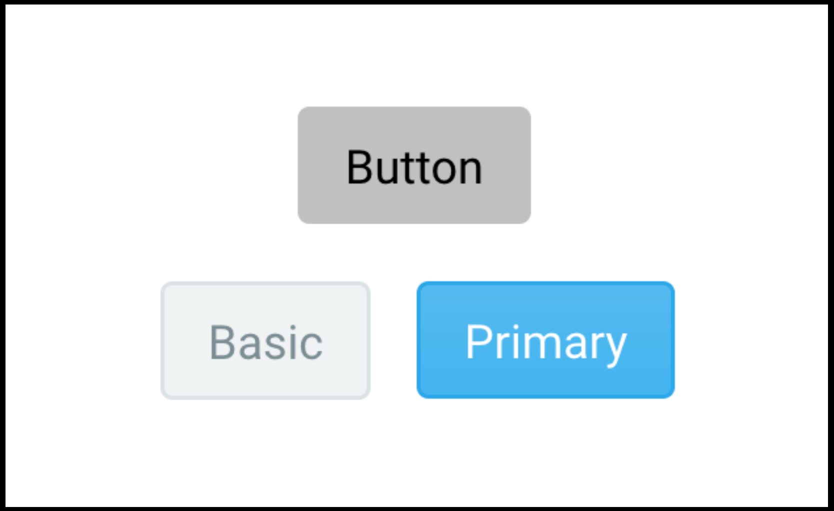
* …because we'll use MODIFIER CLASSES to make our BUTTONS INTERESTING.
* These button--basic and button--primary classes are the MODIFIER CLASSES.
* The DOUBLE DASHES make them EASY TO IDENTIFY as modifiers.
* You can see how they WORK WITH the base component class.



```css
.button--basic {
  background: #f0f3f4;
  color: #7c8e94;
  border-color: #dae3e6;
}

.button--primary {
  background: linear-gradient(#54baef, #43b4ef) #54baef;
  color: #ffffff;
  border-color: #2ca7e8;
}
```

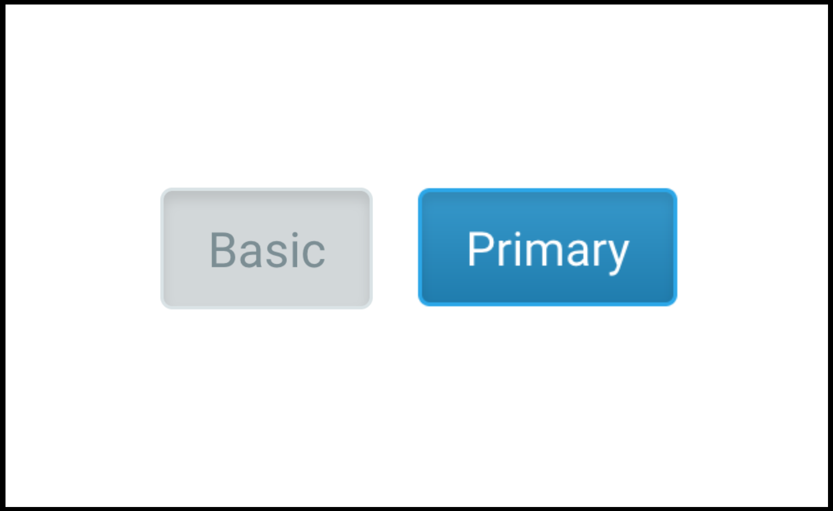
Modifier classes

* Looking at the styles, we see that they FILL THE HOLES left by the base component class.
* They define the PROPERTIES that make these different types of buttons UNIQUE.



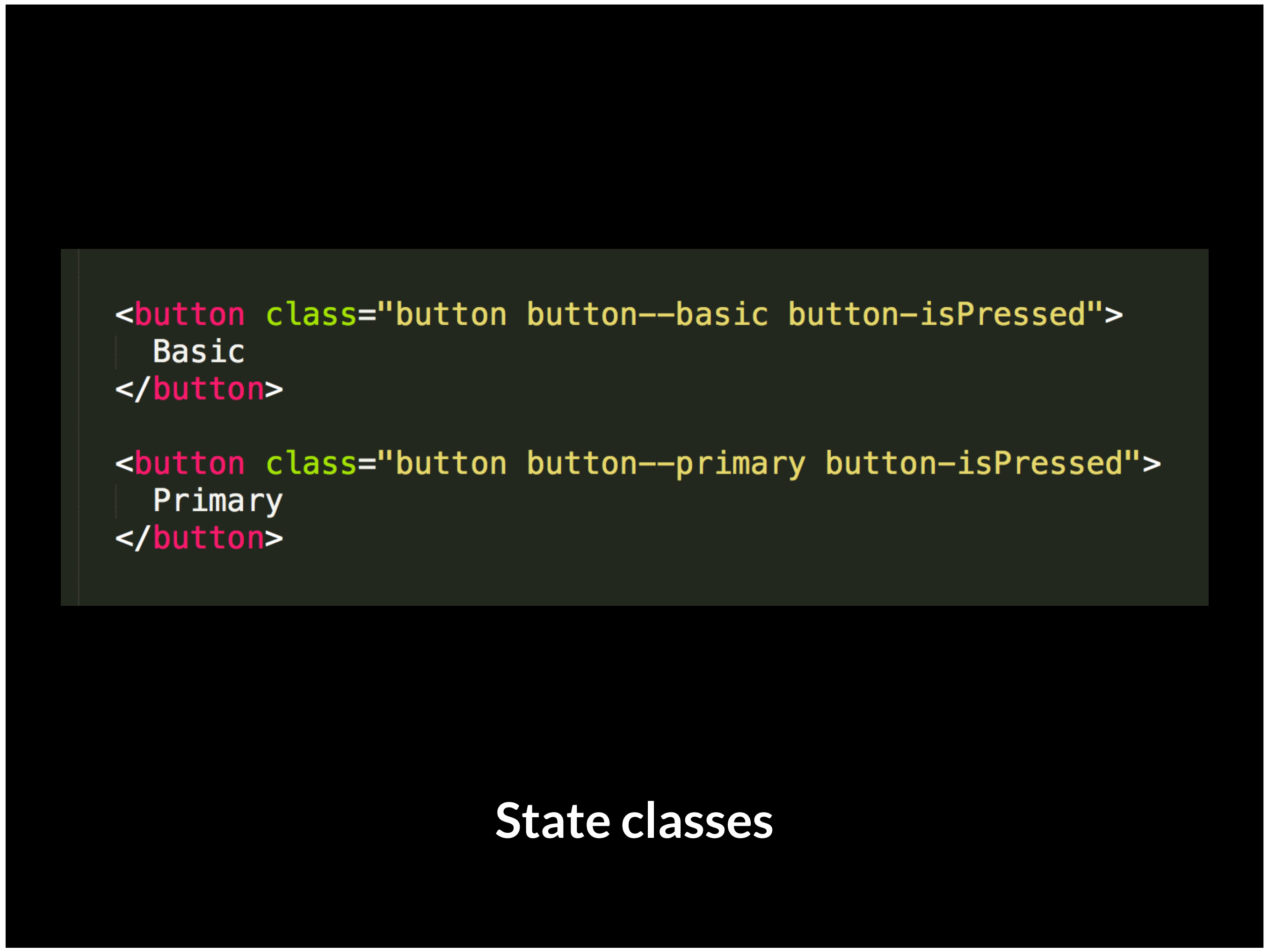
Modifier classes

* And here in the BROWSER, you can see how they’ve CUSTOMIZED the appearance of the button with just the base component class applied.



State classes

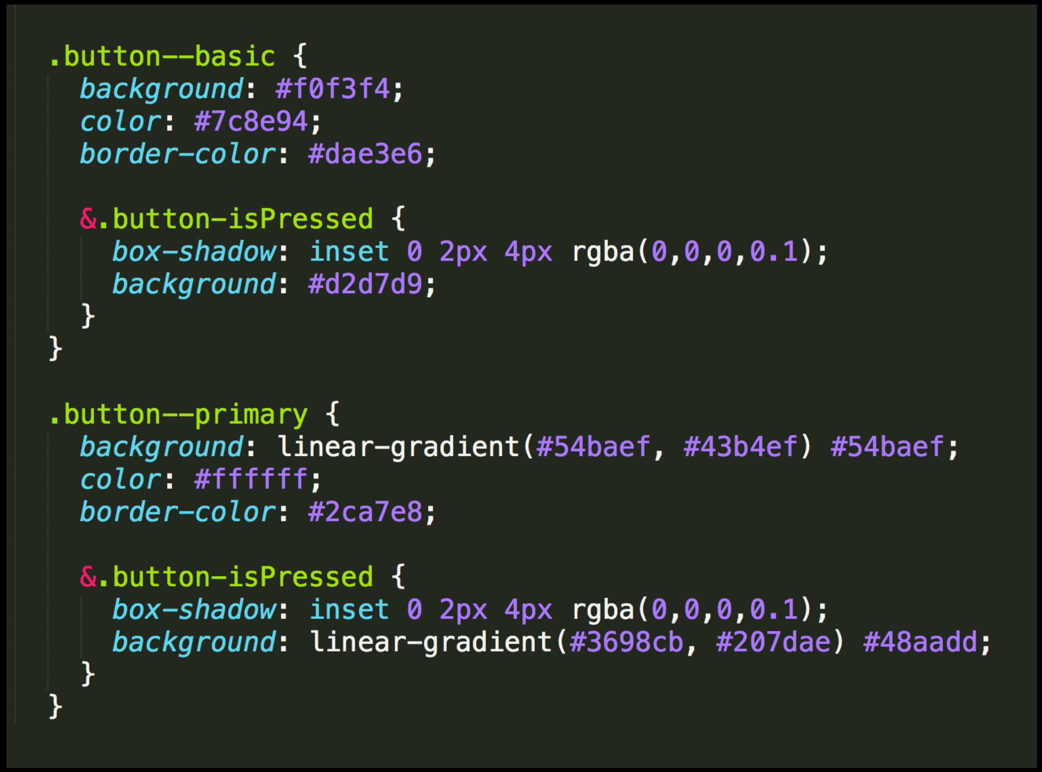
- Now let’s say we want each of these button types to CHANGE APPEARANCE when they’ve been PRESSED.
- This is where STATE CLASSES come in.
- You can create STATE classes for all kinds of states: DISABLED, SELECTED, SAVING
- But in this case we only care about the PRESSED state



```
<button class="button button--basic button-isPressed">
  Basic
</button>

<button class="button button--primary button-isPressed">
  Primary
</button>
```

State classes

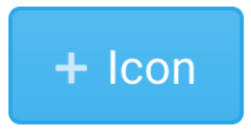
- You can TELL what's a state class by the use of a single dash and the word "IS" inside of the name.
- STORY:
- People sometimes ask what's the DIFFERENCE between a state and a modifier
- Modifier classes live on an element for the element's ENTIRE LIFETIME
- State classes are APPLIED and REMOVED by JAVASCRIPT and can be applied and remove MANY TIMES over the COURSE of the element's lifetime.
-



```
.button--basic {
  background: #f0f3f4;
  color: #7c8e94;
  border-color: #dae3e6;

  &.button-isPressed {
    box-shadow: inset 0 2px 4px rgba(0,0,0,0.1);
    background: #d2d7d9;
  }
}

.button--primary {
  background: linear-gradient(#54baef, #43b4ef) #54baef;
  color: #ffffff;
  border-color: #2ca7e8;

  &.button-isPressed {
    box-shadow: inset 0 2px 4px rgba(0,0,0,0.1);
    background: linear-gradient(#3698cb, #207dae) #48aadd;
  }
}
```

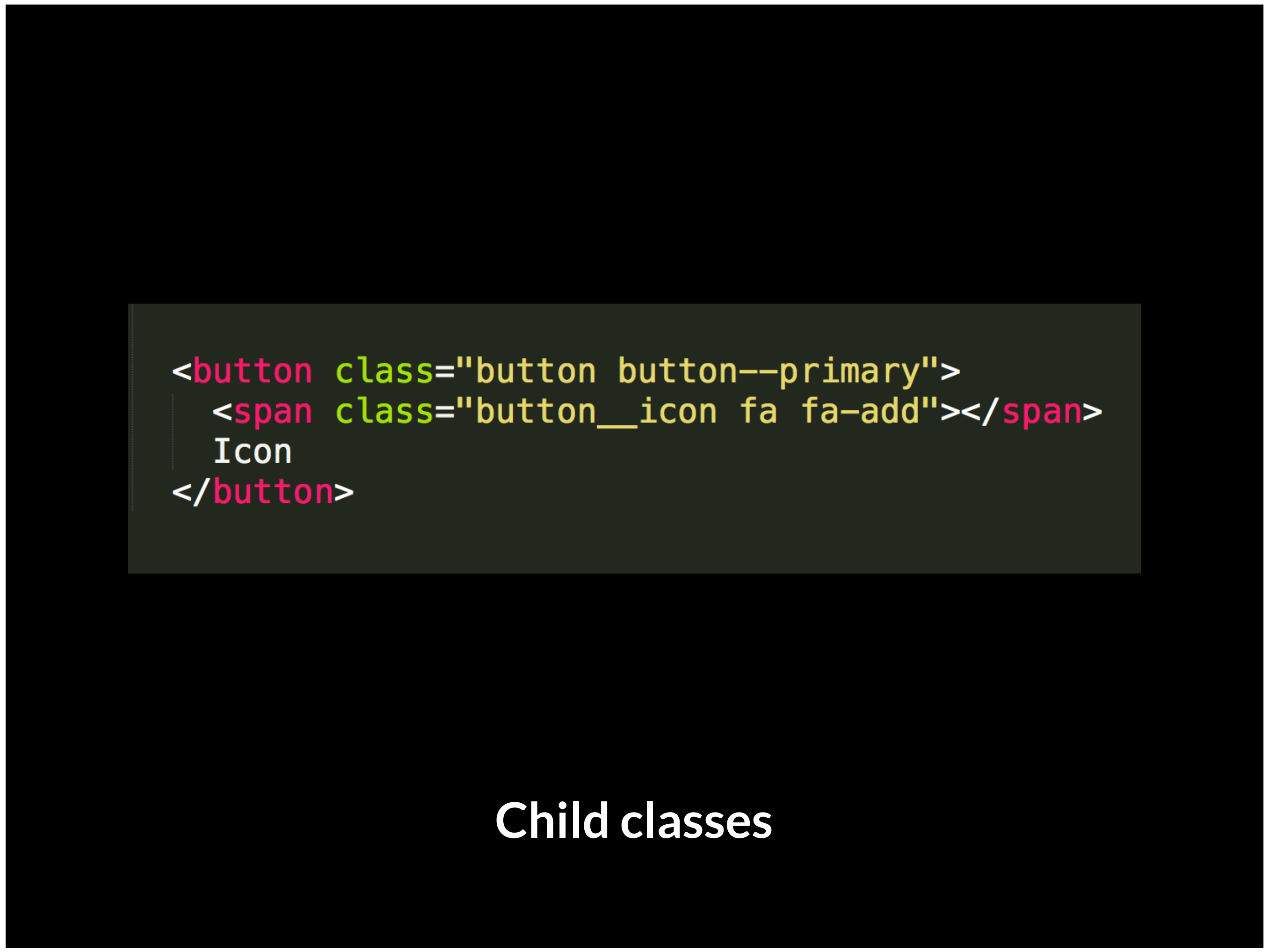
State classes

- We can see the button-isPressed state class but SPECIFIC to each MODIFIER.
- They just alter the BACKGROUND COLOR and add an inset box-shadow.
- STORY:
- When this is compiled to CSS, these selectors will have higher SPECIFICITY, because they'll ONLY APPLY to elements that have BOTH the modifier class and the state class.
- In the case of STATE CLASSES, this is OK, because we can be fairly confident that we WON'T apply ADDITIONAL CLASSES to these buttons
- that would CHANGE their appearance IN THIS STATE.



Child classes

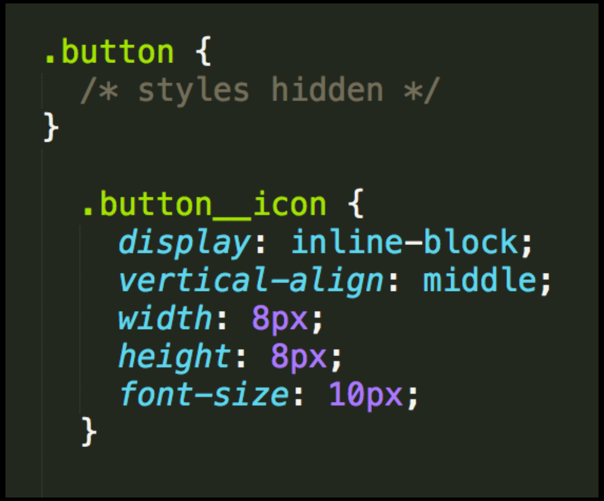
* FINALLY, we get to CHILD CLASSES.
* CHILD CLASSES are classes for elements which are TIGHTLY COUPLED to a base component.
* In this case, the ICON only makes SENSE within the context of the BUTTON.



```
<button class="button button--primary">
  <span class="button__icon fa fa-add"></span>
  Icon
</button>
```

Child classes

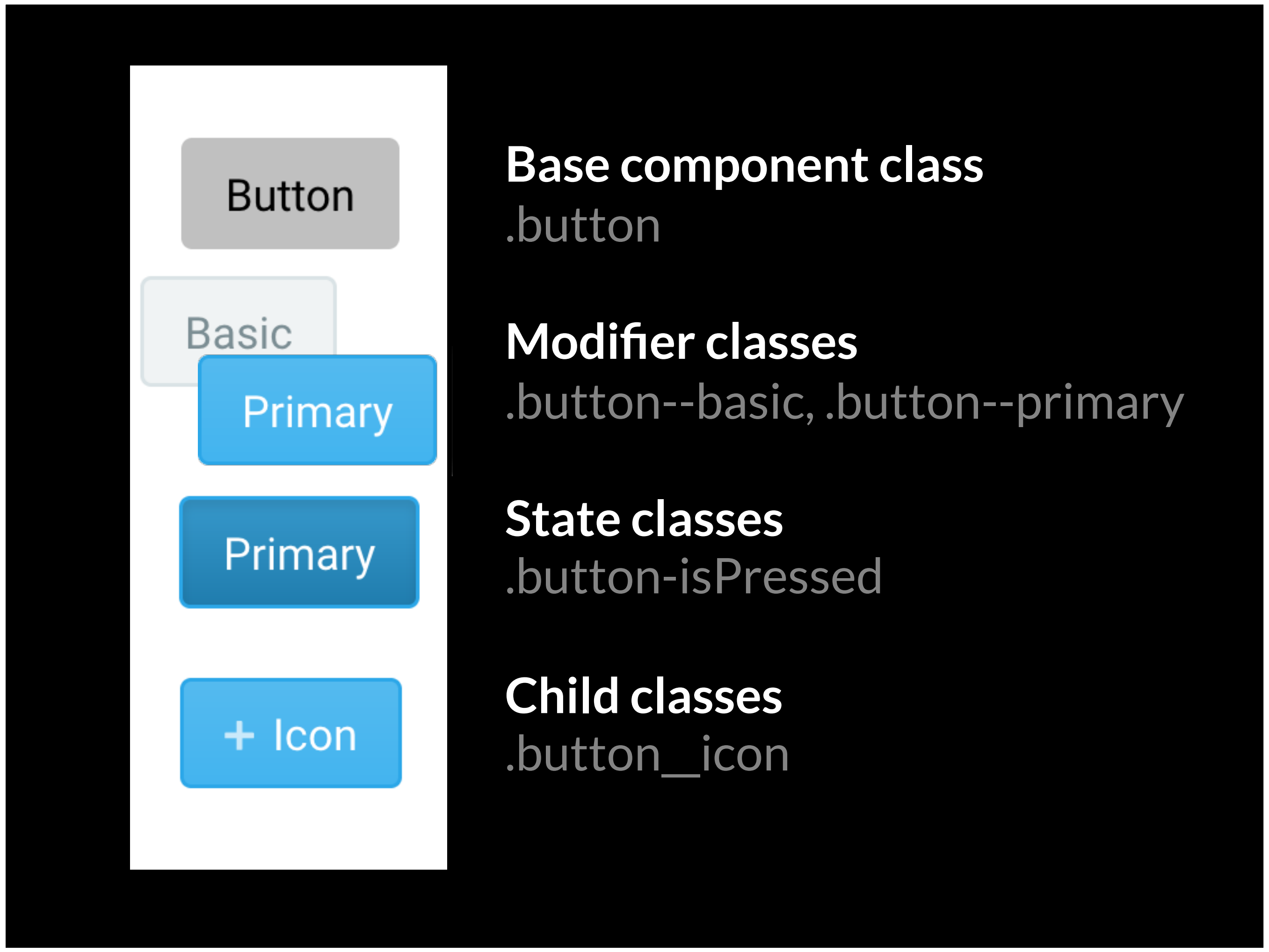
- Let’s take a LOOK. Here we have an element that’s a CHILD of the button, with the CHILD CLASS button__icon.
- Child classes have the SAME NAME as the base component, followed by double underscores and then a UNIQUE NAME.
- The double underscores give child classes a UNIQUE SHAPE
- Makes it easy to SCAN the MARKUP and SPOT them
- And identify their RELATIONSHIP to the BASE COMPONENT in the MARKUP.



```
.button {
  /* styles hidden */
}

  .button__icon {
    display: inline-block;
    vertical-align: middle;
    width: 8px;
    height: 8px;
    font-size: 10px;
  }
```

Child classes

* In the SCSS, child classes are INDENTED beneath the base component class, but NOT NESTED inside of it.
* By doing this we GAIN TWO THINGS
* Indentation VISUALLY COMMUNICATES to us the parent-child relationship
* But we AVOID the SPECIFICITY we would gain if we actually nested them.



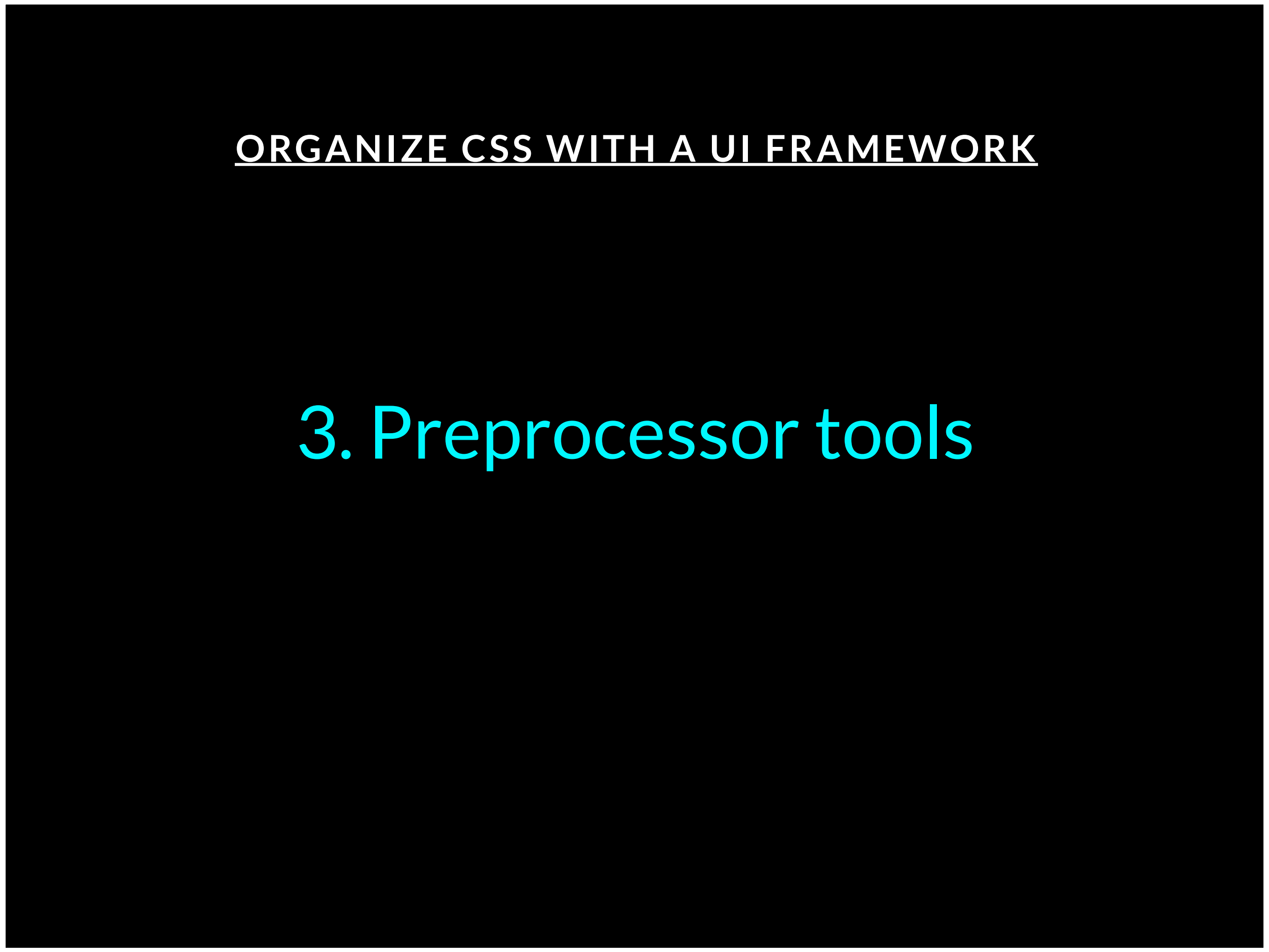
Base component class
.button

Modifier classes
.button--basic, .button--primary

State classes
.button-isPressed

Child classes
.button__icon

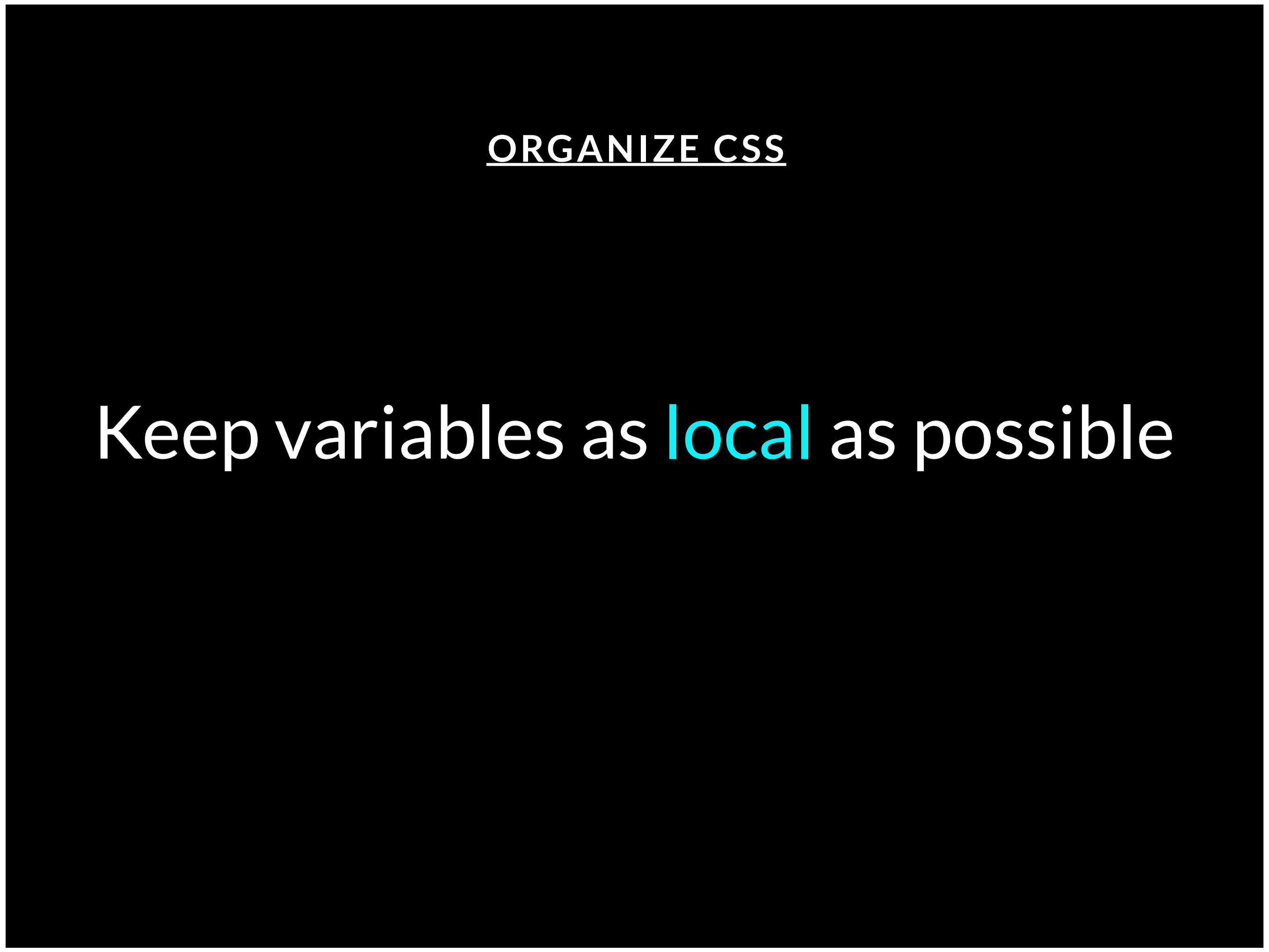
- RECAP:
- Naming convention consists of 4 types of class names
- Base component class, etc
- Each one of these has a DIFFERENT NAME SHAPE.
- which helps IDENTIFY its ROLE and DIFFERENTIATE it from the others.



ORGANIZE CSS WITH A UI FRAMEWORK

3. Preprocessor tools

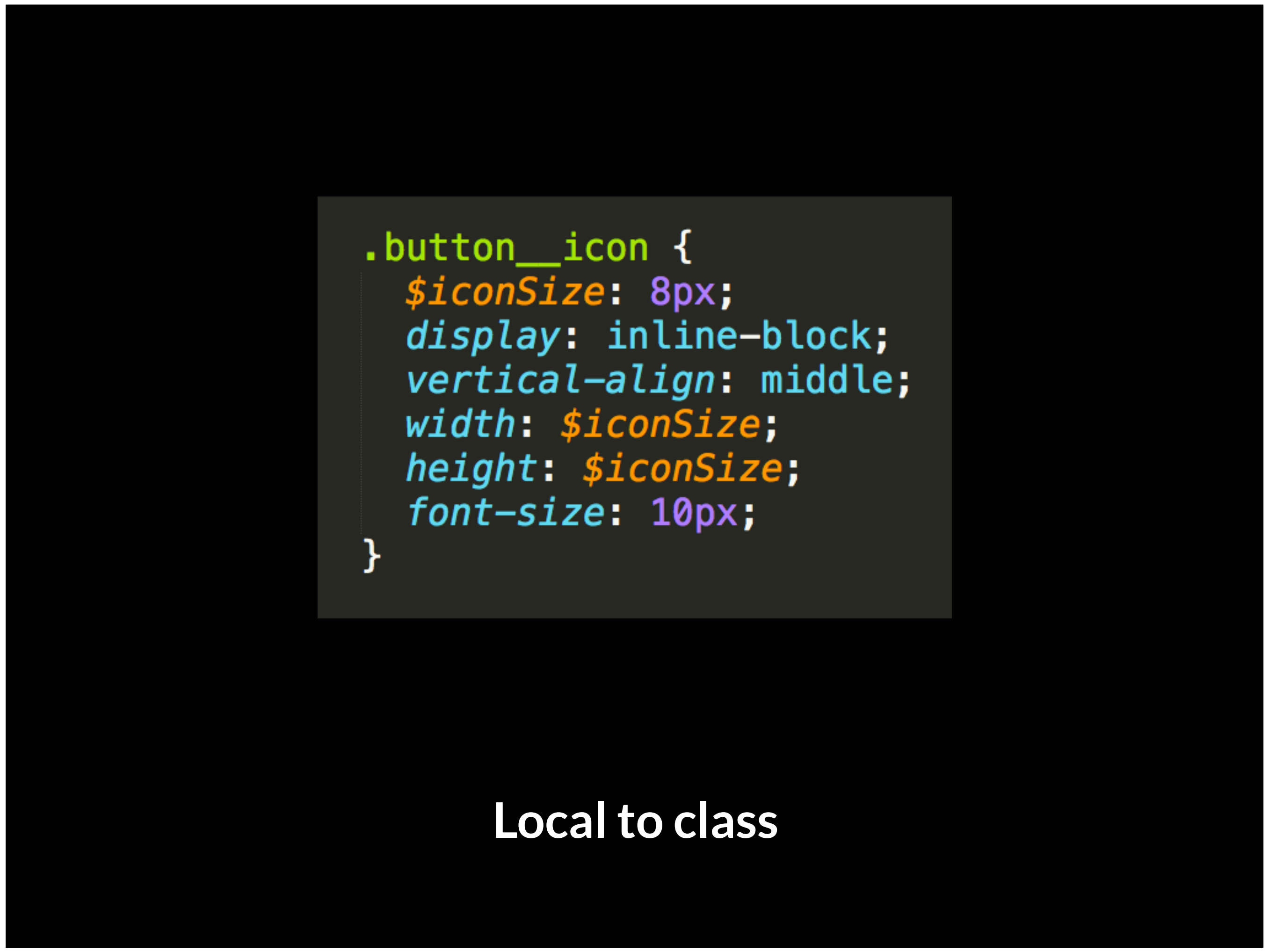
* Preprocessors like SCSS and LESS provide TOOLS like VARIABLES and MIXINS
* Which let you create ABSTRACTIONS around your STYLES and the VALUES of various PROPERTIES.
* If we use them wisely, we can USE them to help us SCALE our CSS.



ORGANIZE CSS

Keep variables as local as possible

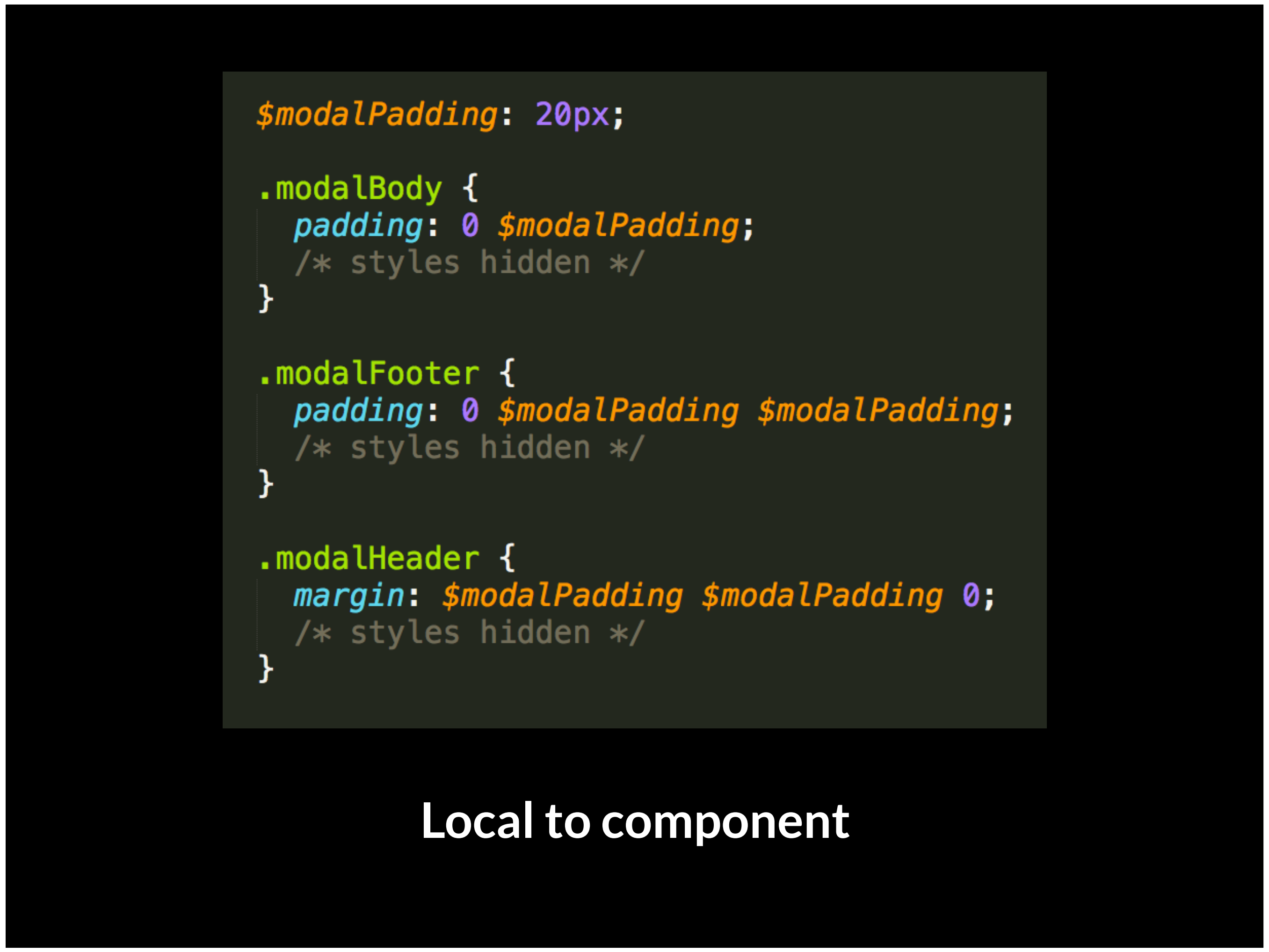
* STORY:
* Remember that COLORS VARIABLE FILE I showed before. 450 lines. Totally UNUSABLE.
* When you PREMATURELY generalize like that, you end up introducing complexity that’s hard to SCALE and slows you down. Better to globalize GRADUALLY, when NECESSARY.



```
.button__icon {
  $iconSize: 8px;
  display: inline-block;
  vertical-align: middle;
  width: $iconSize;
  height: $iconSize;
  font-size: 10px;
}
```

Local to class

* This is about as local as you get in SCSS.
* This variable can only be used inside of this class declaration.
* You can change this variable with CONFIDENCE because you know the change will be limited in scope. You don’t need to worry about UNEXPECTED SIDE EFFECTS somewhere else in the code.

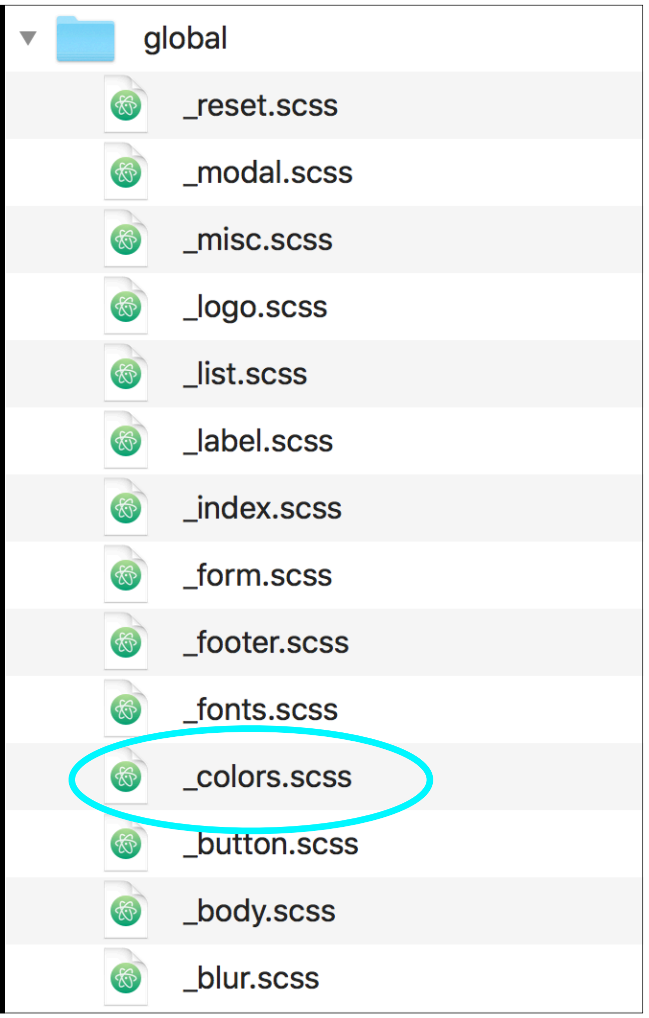


```
$modalPadding: 20px;

.modalBody {
  padding: 0 $modalPadding;
  /* styles hidden */
}

.modalFooter {
  padding: 0 $modalPadding $modalPadding;
  /* styles hidden */
}

.modalHeader {
  margin: $modalPadding $modalPadding 0;
  /* styles hidden */
}
```

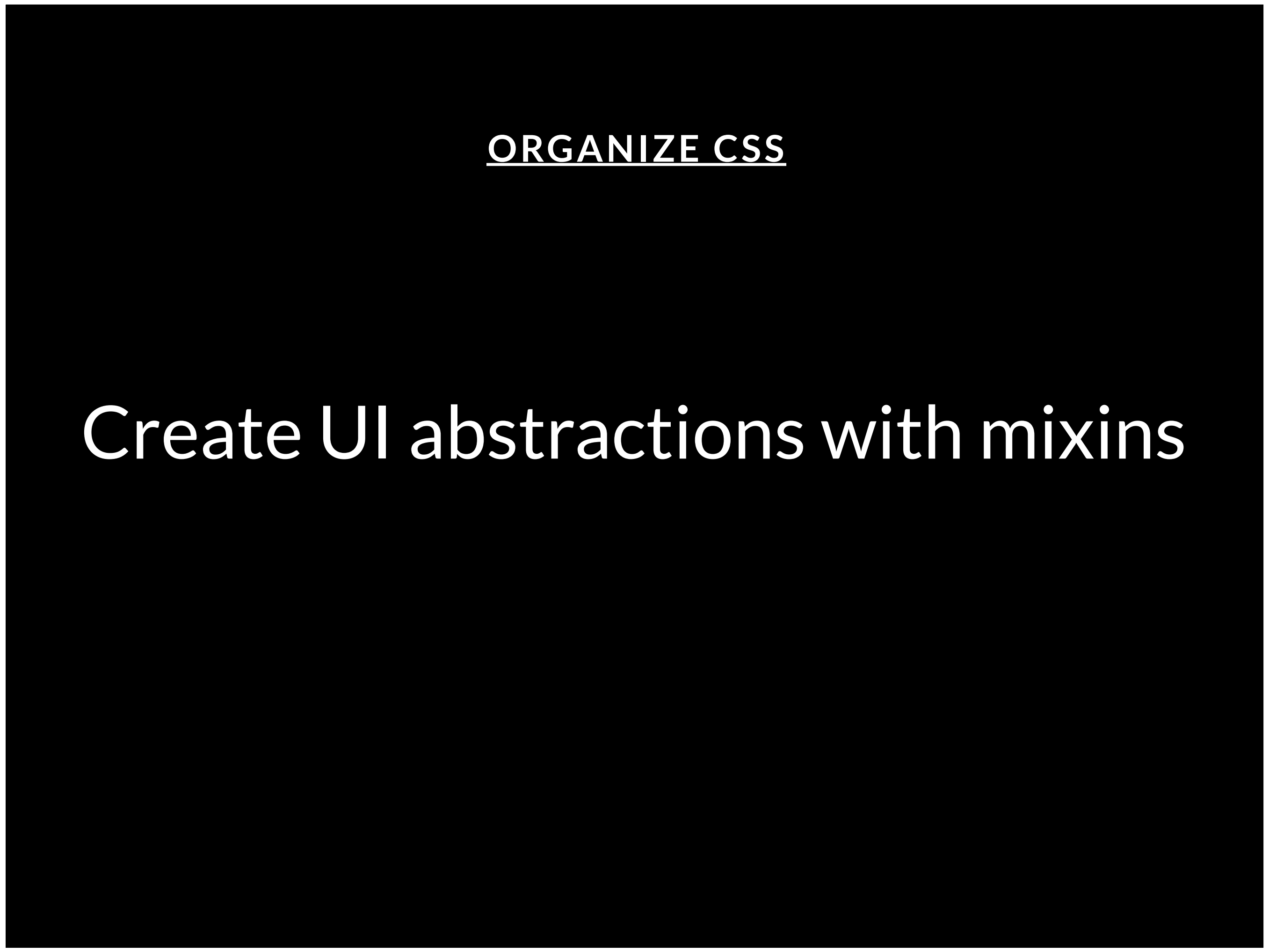
Local to component

- If you need to make a variable more global, try limiting the scope to just the COMPONENT.
- Just be sure to NAME them meaningfully and UNIQUELY
- Make it clear that the variable is TIED to the component
- Because now the variable technically lives in a GLOBAL scope and you want to avoid NAMING COLLISIONS

“global” folder as last resort


global
_reset.scss
_modal.scss
_misc.scss
_logo.scss
_list.scss
_label.scss
_index.scss
_form.scss
_footer.scss
_fonts.scss
_colors.scss
_button.scss
_body.scss
_blur.scss

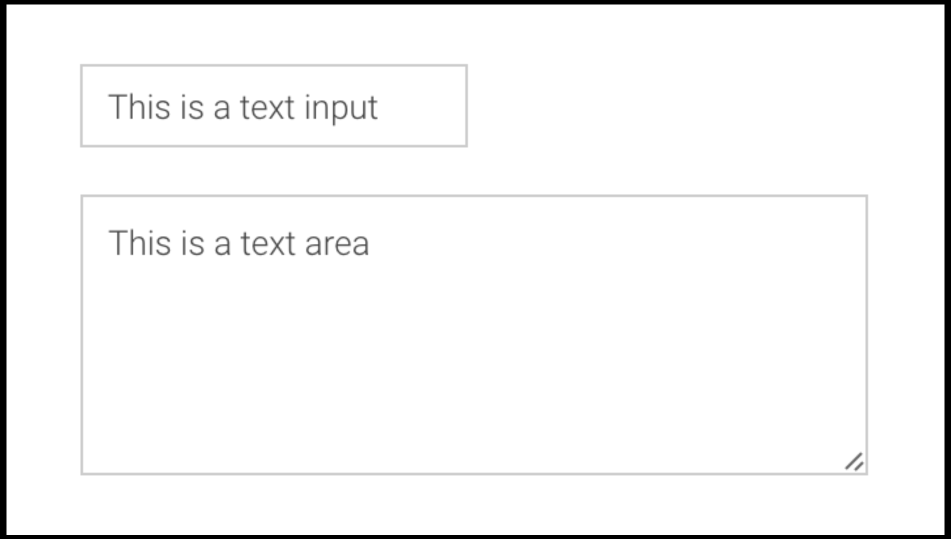
- If you really need GLOBAL variables, I suggest putting them in a “global” folder inside the “uiFramework” folder.
- But only do this out of NECESSITY. This prevents PREMATURE GENERALIZATION, and you’ll know that these variables have a GOOD REASON to be global.
- See that “colors” file? By following this pattern, this time my colors file was only 45 lines long.



ORGANIZE CSS

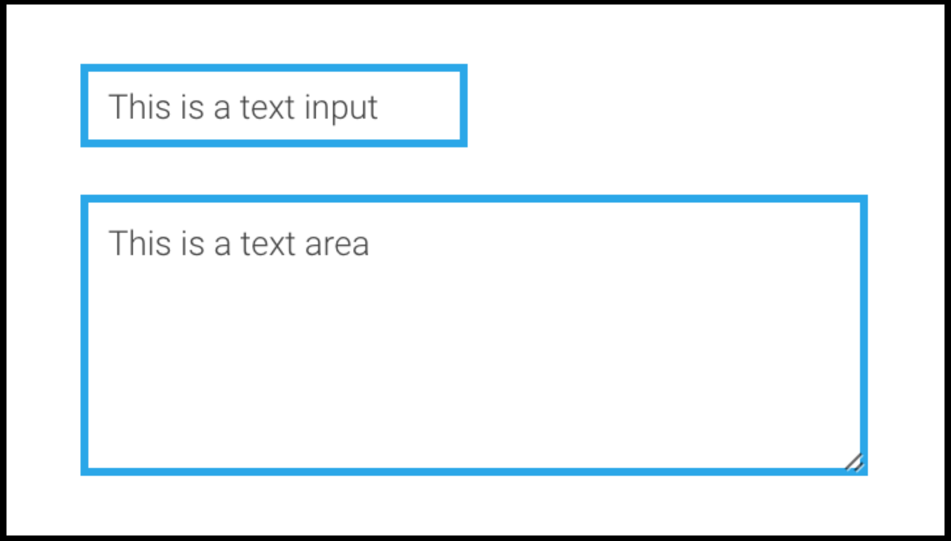
Create UI abstractions with mixins

- Mixins are ANOTHER invaluable tool that PREPROCESSORS offer.
- I’ve found that the BEST way to use them, is to USE them to create ABSTRACTIONS around REPEATING patterns in the UI.



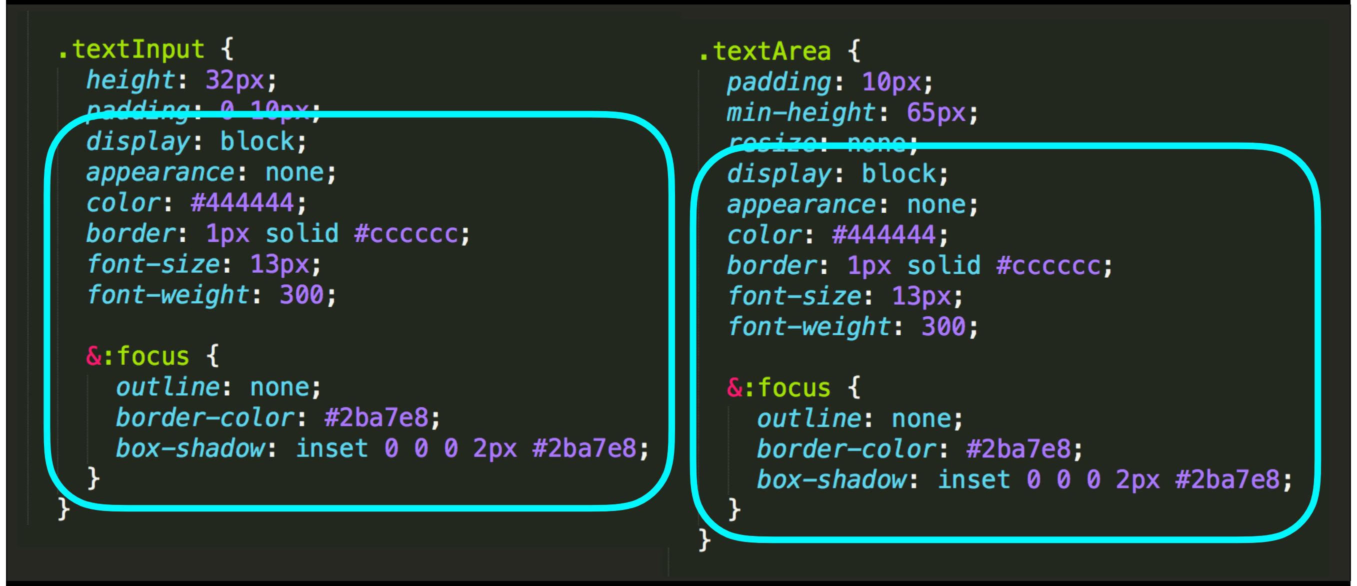
TextInput and TextArea components

* For EXAMPLE, take a look at these TextInput and TextArea components.
* STORY:
* In a USER INTERFACE, you want to use the APPEARANCE of controls to communicate their FUNCTION.
* In this case, these are both FORM FIELDS, which the user can ENTER TEXT into.
* So it makes sense that they both LOOK VERY SIMILAR



Receiving focus

* Even their appearance when RECEIVING FOCUS is alike
* Both use a BLUE BORDER

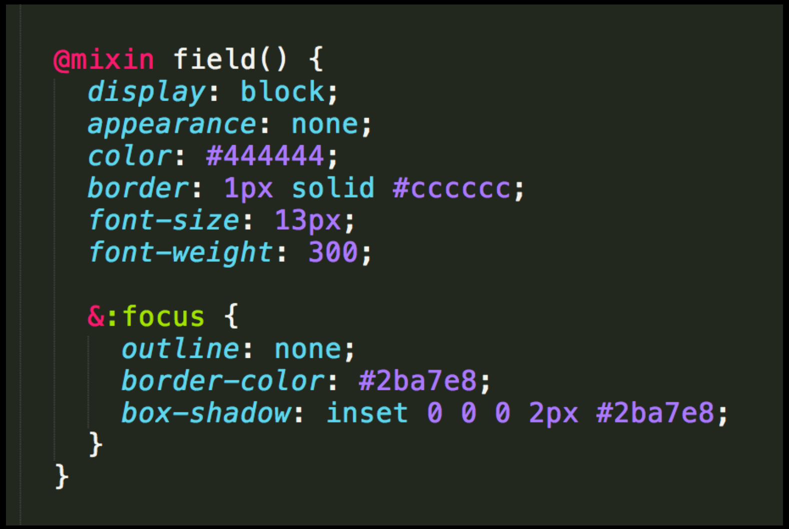


```
.textInput {
  height: 32px;
  padding: 0 10px;
  display: block;
  appearance: none;
  color: #444444;
  border: 1px solid #cccccc;
  font-size: 13px;
  font-weight: 300;

  &:focus {
    outline: none;
    border-color: #2ba7e8;
    box-shadow: inset 0 0 0 2px #2ba7e8;
  }
}
```

```
.textArea {
  padding: 10px;
  min-height: 65px;
  resize: none;
  display: block;
  appearance: none;
  color: #444444;
  border: 1px solid #cccccc;
  font-size: 13px;
  font-weight: 300;

  &:focus {
    outline: none;
    border-color: #2ba7e8;
    box-shadow: inset 0 0 0 2px #2ba7e8;
  }
}
```

Common styles are UI patterns

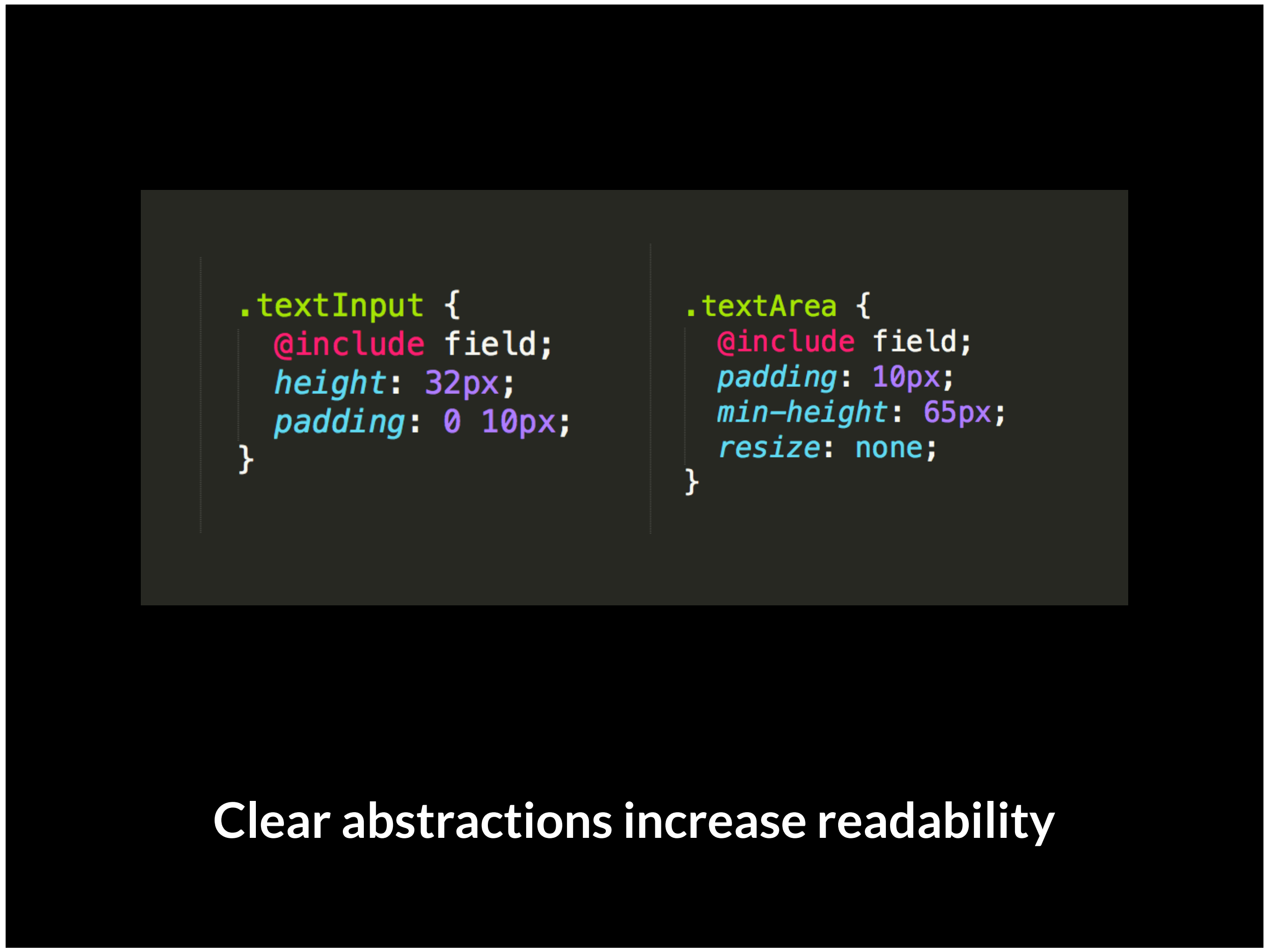
* If we look at the STYLES, we can IMMEDIATELY SPOT where they OVERLAP.
* STORY:
* If we want to define the CONCEPT of a “FORM FIELD” within the VISUAL LANGUAGE of our UI, then these styles are that DEFINITION.



```
@mixin field() {
  display: block;
  appearance: none;
  color: #444444;
  border: 1px solid #cccccc;
  font-size: 13px;
  font-weight: 300;

  &:focus {
    outline: none;
    border-color: #2ba7e8;
    box-shadow: inset 0 0 0 2px #2ba7e8;
  }
}
```

Abstracted UI pattern

- We can use a MIXIN to ENCAPSULATE that definition.
- Now if we wanted to CHANGE that definition, we could do it HERE in ONE PLACE
- instead of in MULTIPLE places throughout the CSS



```
.textInput {
  @include field;
  height: 32px;
  padding: 0 10px;
}
```

```
.textArea {
  @include field;
  padding: 10px;
  min-height: 65px;
  resize: none;
}
```

Clear abstractions increase readability

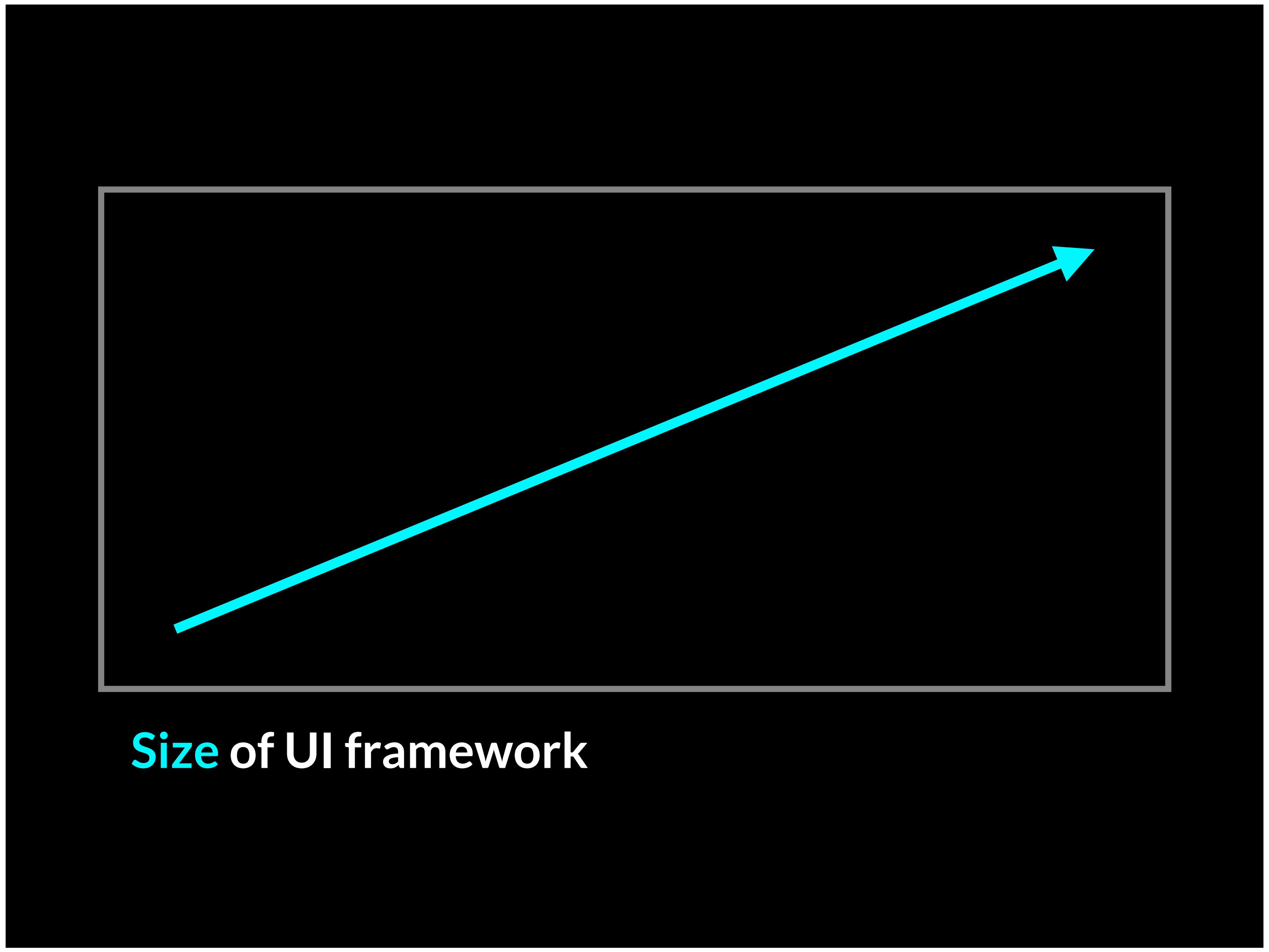
- STORY:
- Now, when we LOOK at the TEXTINPUT and TEXTAREA classes, we can SPOT the use of these ABSTRACTIONS.
- The clear NAME of the mixin adds INFORMATION for us
- We see it and can say, “AHH… this is a FORM FIELD”, and immediately understand HOW it will FUNCTION in the UI

ORGANIZE CSS WITH A UI FRAMEWORK

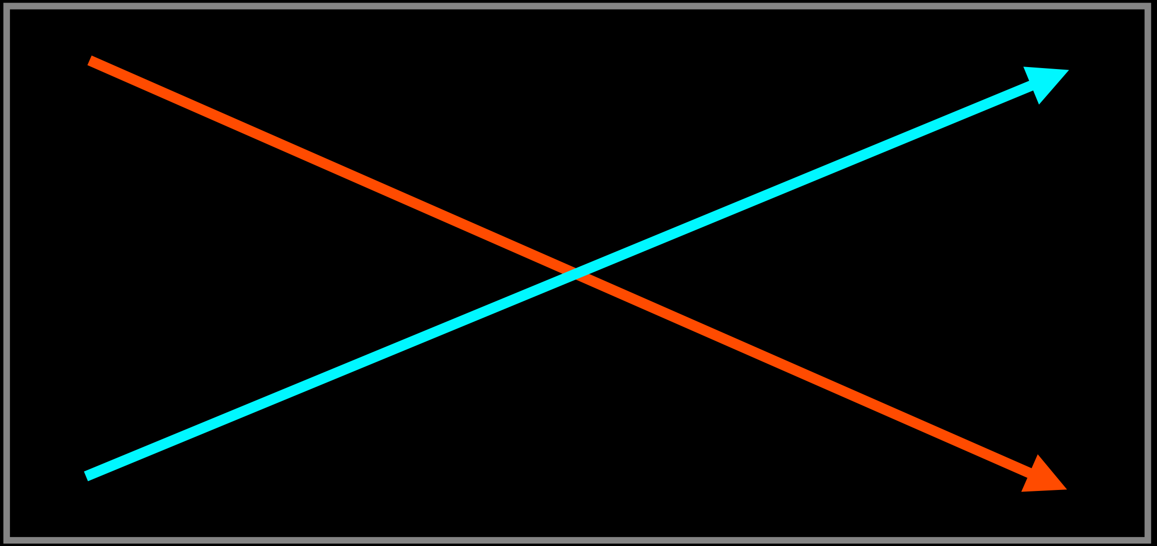
- A clear file and folder structure
- Consistent naming convention
- Effective use of preprocessor tools

This is an investment

* STORY:
* We’re getting to the END NOW, and I don’t know about YOU but I’m really looking FORWARD to the AFTER PARTY.
* So let me LEAVE you with a THOUGHT.
* It may seem like a lot of WORK to apply these IDEAS… you may have to refactor some code.
* And it’s tough to challenge and change existing practices for writing your CSS. If you decide to apply the ideas I talked about, it will take time and hard work! :)
* But it’s an investment… and I promise you it will PAY OFF.



* As you INVEST TIME in the IDEAS I talked about TODAY, you’ll see the size of your UI framework grow
* And it will grow STEADILY, because the WAY you’re WRITING the code will be SCALABLE

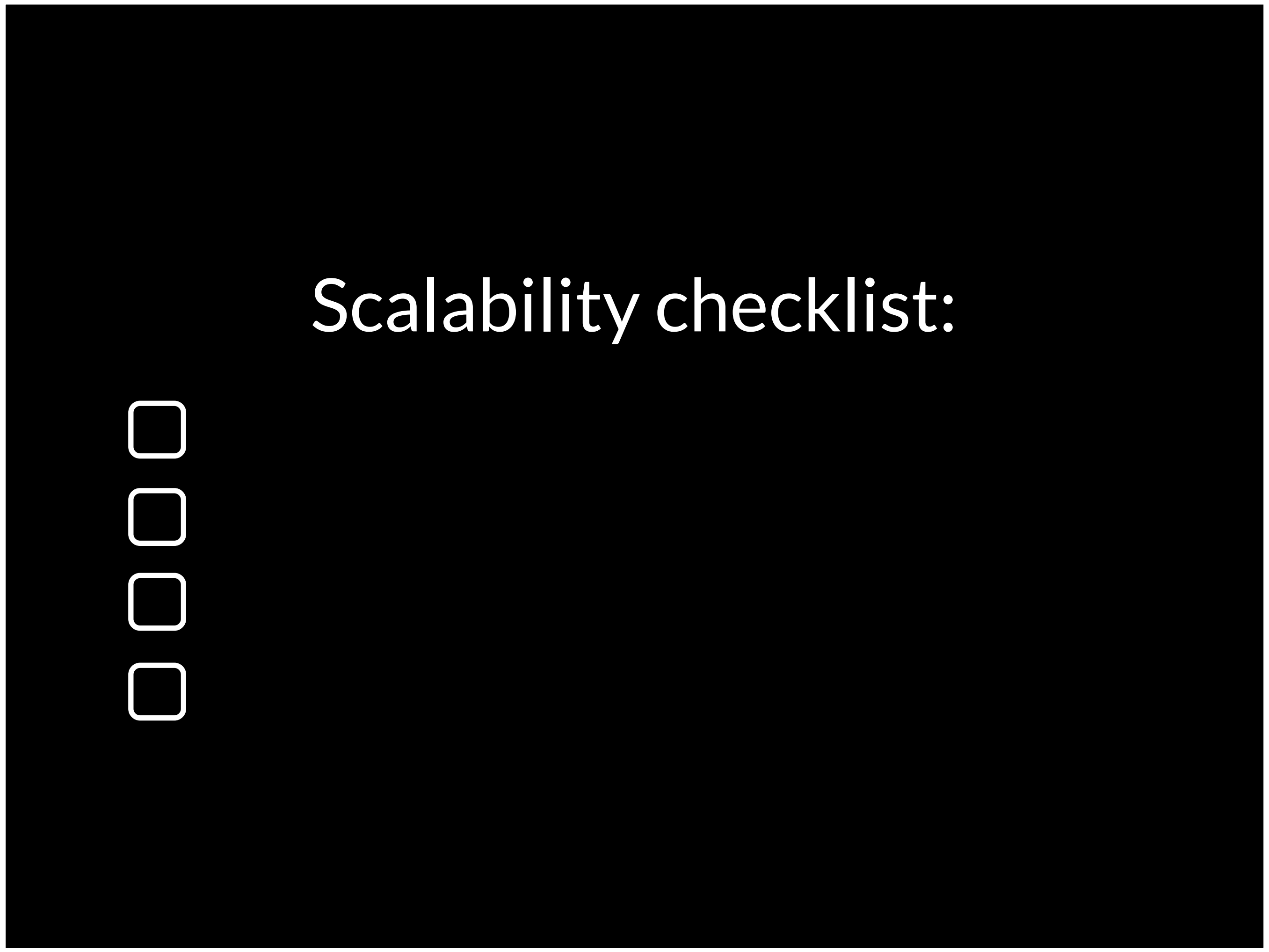


Size of UI framework
Time spent building UI

* As your UI framework GROWS, you’ll find yourself able to build user interfaces more QUICKLY and more CONFIDENTLY.
* You’ll spend less time thinking about your CSS and more time thinking about the UI and UX itself.

Scalable CSS = Faster, better UI development

* Scalable CSS gives you faster better UI development
* THIS is the PAYOFF of your INVESTMENT



Scalability checklist:

☐
☐
☐
☐

* Recap of my SCALABILITY CHECKLIST

Scalability checklist:

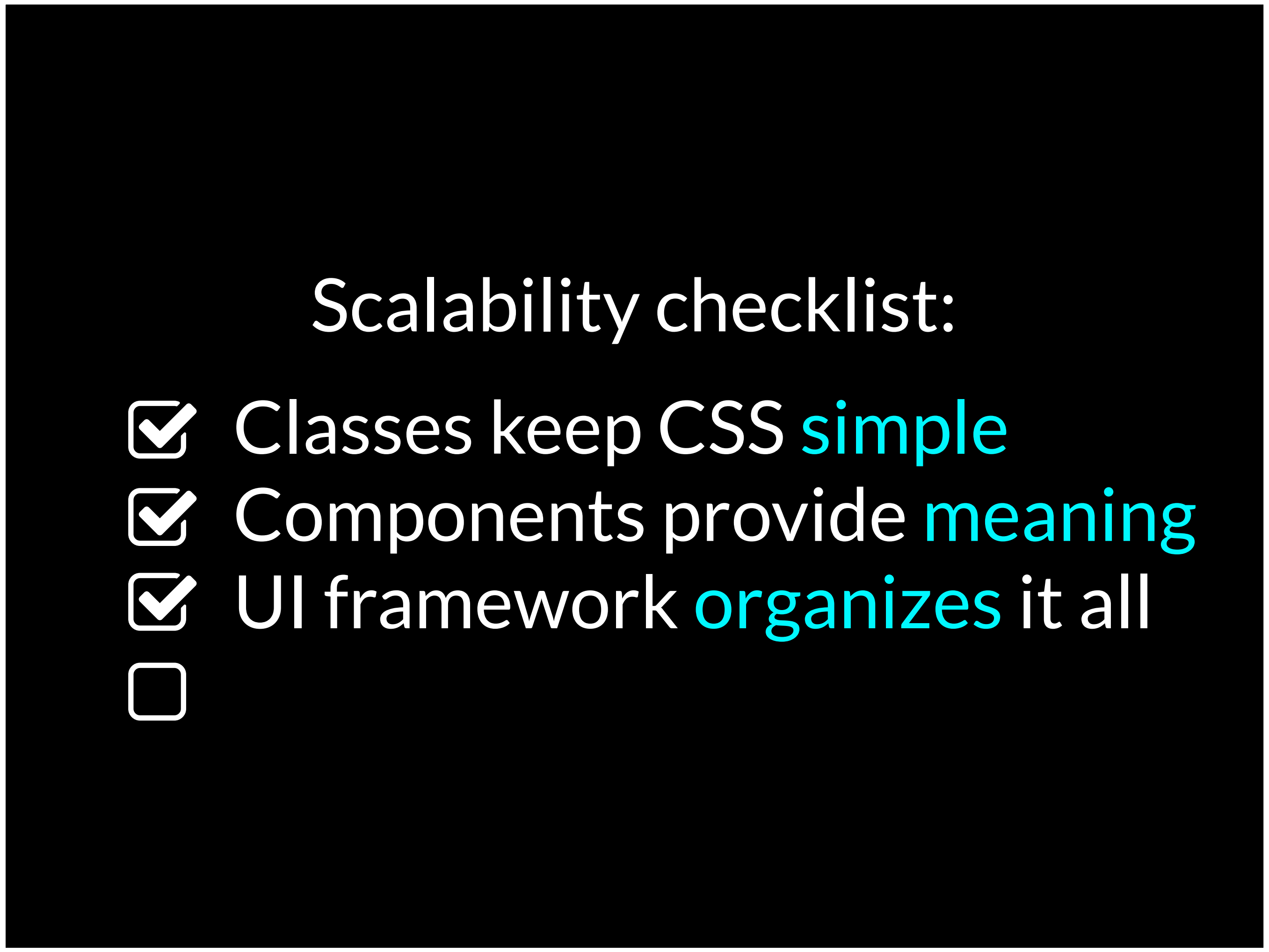
- ✓ Classes keep CSS simple
- ☐
- ☐
- ☐

* Use classes to create short, simple selectors.
* Give the classes NAMES that are EASY to UNDERSTAND.
* Classes also help you keep your specificity LOW so you can avoid the DOOMSDAY scenario where you're adding on LAYER upon LAYER of complex SELECTORS.
*

Scalability checklist:

- ☑ Classes keep CSS simple
- ☑ Components provide meaning
- ☐
- ☐

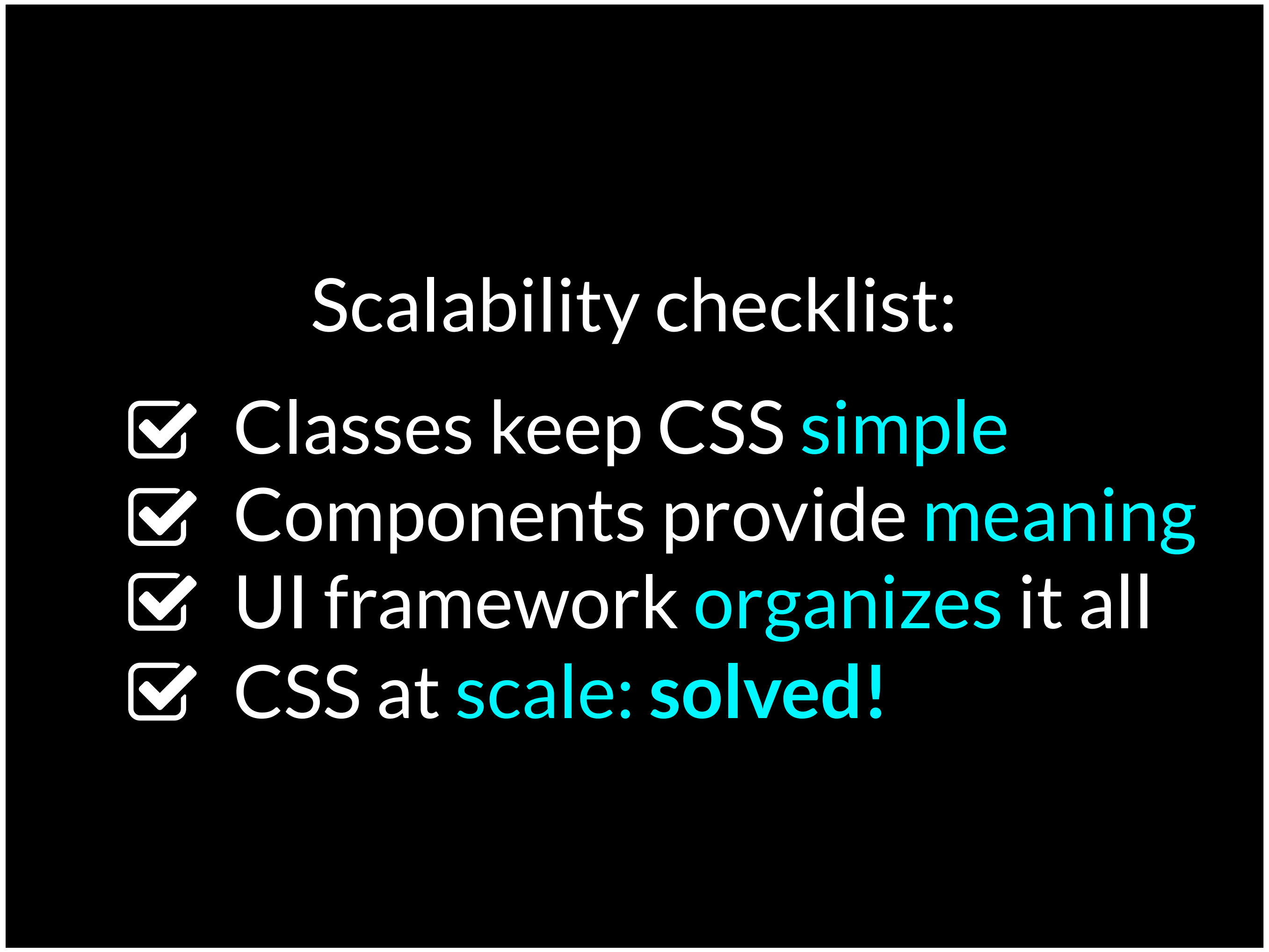
* Use components to add meaning to your CSS
* Components have clear ROLES in the UI. You can look at them and tell WHAT they DO and HOW you can USE them to build your UI.
* Once you’re familiar with the components, you can READ THE MARKUP and form a CLEAR MENTAL IMAGE of how it will look in the browser.



Scalability checklist:

- ☑ Classes keep CSS simple
- ☑ Components provide meaning
- ☑ UI framework organizes it all
- ☐

* Organize everything within a UI framework
* Your file and folder structure helps you find and organize your CSS.
* Your naming convention helps make both the CSS and MARKUP more readable. You can READ the markup and understand the relationships between the different ELEMENTS because their CLASSES are named in a way the suggest roles and relationships.
* And you're using preprocessor tools like mixins and variables to create USEFUL abstractions that help make your CSS more understandable and maintainable.



Scalability checklist:

- ✓ Classes keep CSS simple
- ✓ Components provide meaning
- ✓ UI framework organizes it all
- ✓ CSS at scale: **solved!**

And THIS is how you SOLVE CSS AT SCALE.

Thank you!

@TheCJCenizal

Work with me @ elastic.co