Agenda

What is SEM & BIO?

Briefly BEM

Deep dive into ITCSS

Ravago code examples

SEM & BIO

A combination of methodologies

S calable
E xtensible
M aintainable

B EM: Block Element Modifier
I TCSS: Inverted Triangle CSS
O OCSS: Object Oriented CSS

Focus

S calable
E xtensible
M aintainable

B EM: Block Element Modifier
I TCSS: Inverted Triangle CSS
O OCSS: Object Oriented CSS

SEM

S calable
E xtensible
M aintainable

Scalable

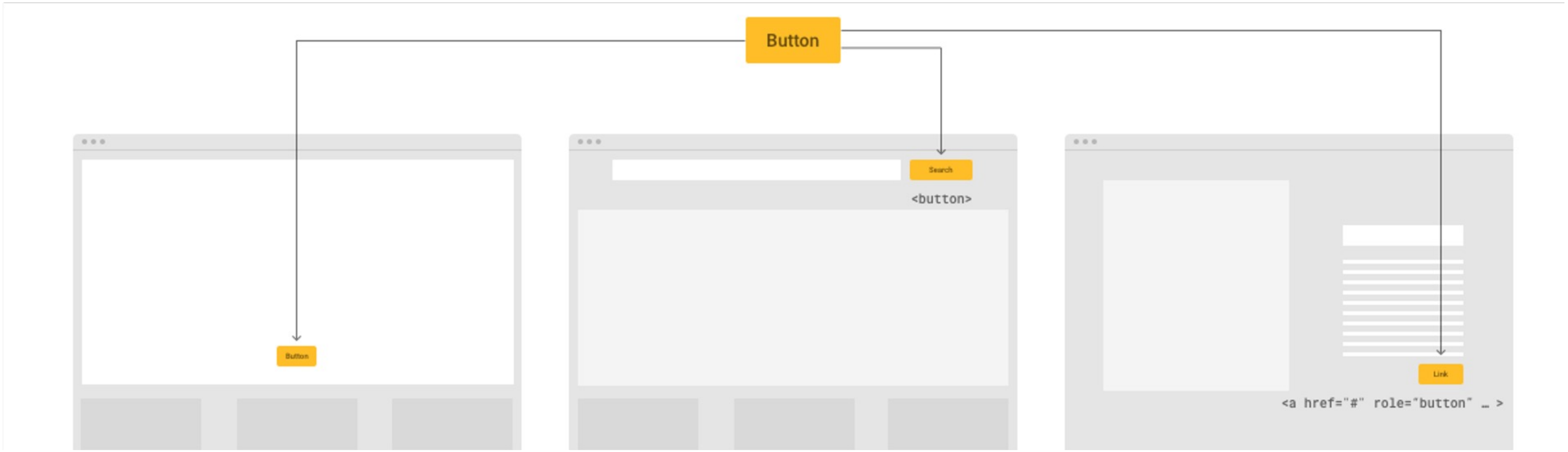
The same looking components should be used anywhere without making any coding changes

Even if the markup is different,
the styles are identical by using the same classes.

S calable

The same looking components should be used anywhere without making any coding changes.

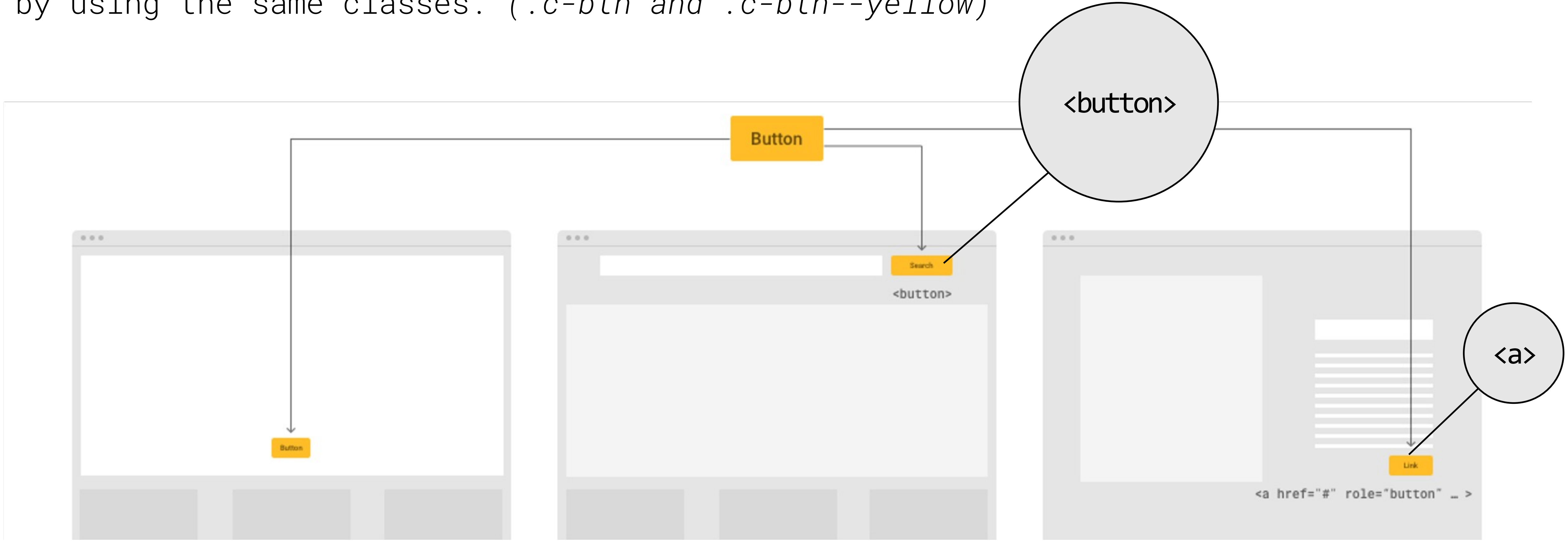
Even if the markup is different, the styles are identical by using the same classes. *(c-btn and c-btn--yellow)*



S calable

The same looking components should be used anywhere without making any coding changes.

Even if the markup is different, the styles are identical by using the same classes. *(.c-btn and .c-btn--yellow)*

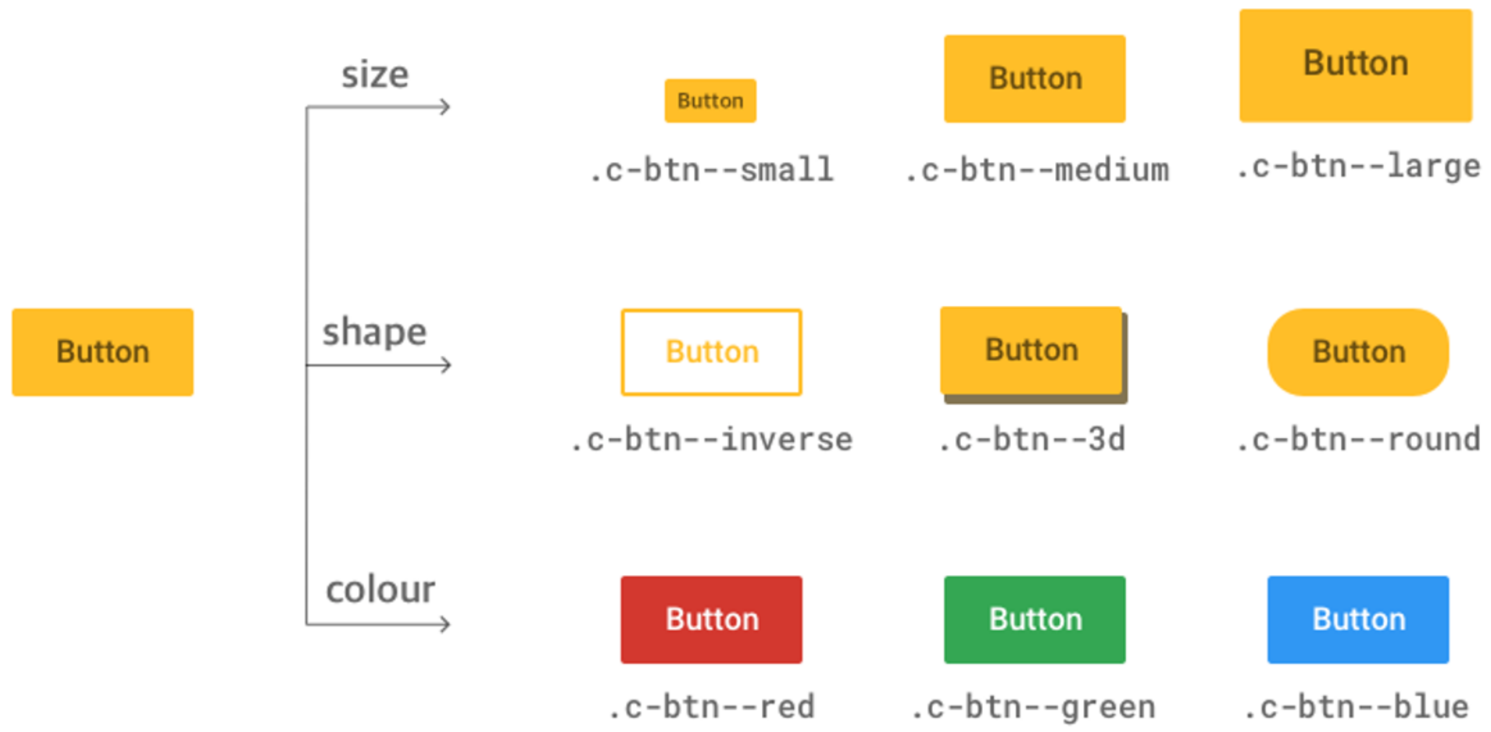


Extensible

Easily provide additional functionalities without breaking itself or having to be written from scratch.

E xtensible

Easily provide additional features/functionalities without breaking itself or having to be written from scratch.

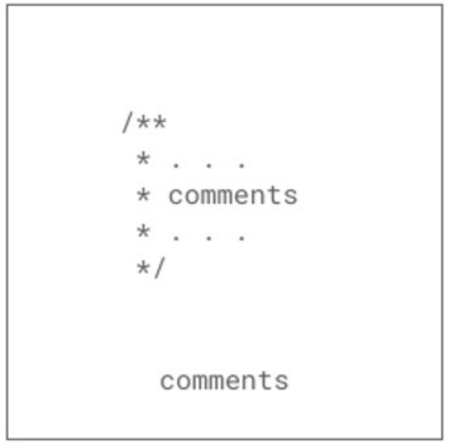


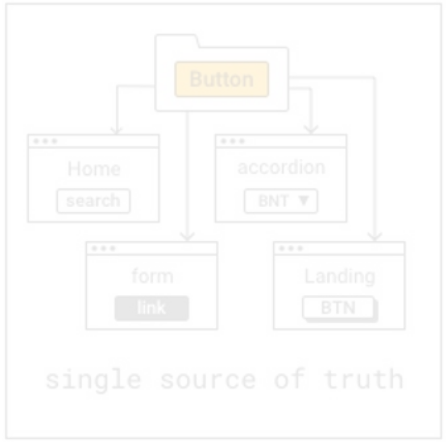
Maintainable

Keep it understandable for other developers
and for your future self

Maintainable

Keep it understandable for other developers and for your future self

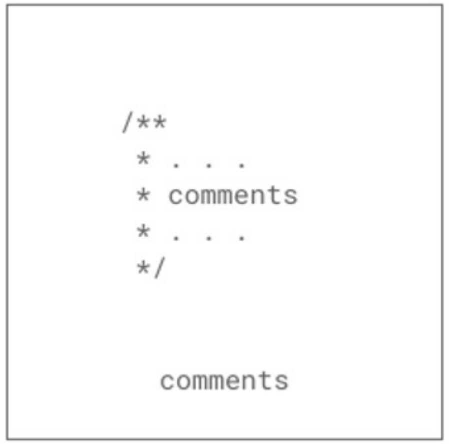


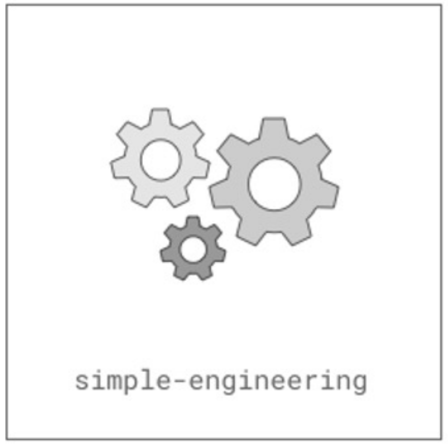


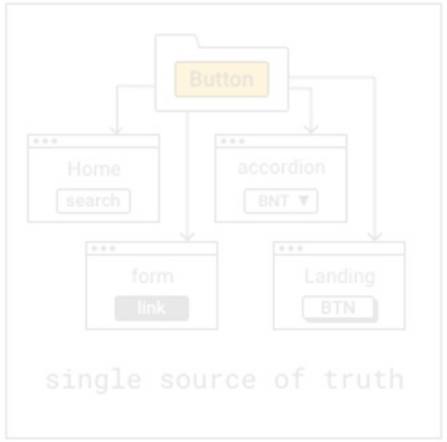
✓

M aintainable

Keep it understandable for other developers and for your future self







✓ ✓

Maintainable

Keep it understandable for other developers and for your future self

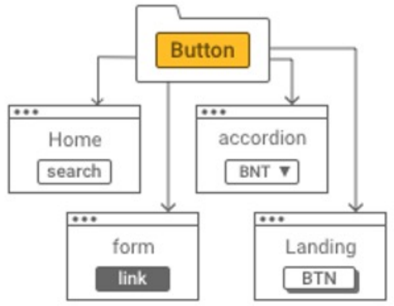
```
/**
 * . . .
 * comments
 * . . .
 */
```

comments

✓

simple-engineering

✓



single source of truth

✓

coding standard

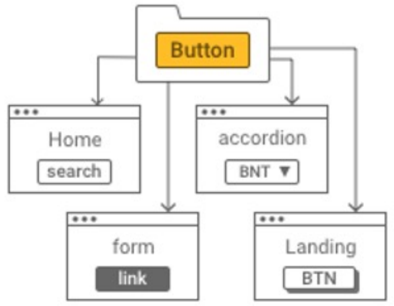
M aintainable

Keep it understandable for other developers and for your future self

```
/**
 * . . .
 * comments
 * . . .
 */
```

comments ✓

simple-engineering ✓



single source of truth ✓

coding standard ✓

Example: Comments

```
.box {
  margin: 24px;
}
```

Add comments when needed

```
.box {
  margin: 11px + 5px + 8px; // height label + margin label + padding input
}
```

Example: Simple-engineering

```
.box {
  &__item {
    &--dark {
      &:hover {
        color: red;
      }
    }
  }
}
```

Keep it simple

```
.box__item--dark:hover {
  color: red;
}
```

Example: Simple-engineering

Expand when needed

```
.box {
  &__item {
    &:hover {
      color: blue;
    }

    &--dark:hover {
      color: white;
    }

    &--light:hover {
      color: black;
    }
  }
}
```

BIO

B EM: Block Element Modifier

I TCSS: Inverted Triangle CSS

O OCSS: Object Oriented CSS

BEM: Block Element Modifier

BEM is a very popular methodology to write CSS with low CSS specificity and unique class names

BEM: Block Element Modifier

```
.grid {
  .grid__item {
    .grid__item-form {
      .grid__item-content {
        // ...
      }
    }
  }
}
```

$\xrightarrow{\text{BEM}}$

```
.grid__item-content {
  // ...
}
```

High CSS specificity

Nesting is hell

Low CSS specificity

Unique class names

BEM: Explanation

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

HTML

```
<div class="card">
  <div class="card__image"></div>
  <div class="card__image card__image--small"></div>
</div>
```

SCSS

```
.card {
  // ...

  &__image {
    // ...

    &--small {
      // ...
    }
  }
}
```

BEM: Block Element Modifier

Some rules

B EM: Block

can contain other blocks

can contain elements

can have a modifiers

```
<div class="card">                                         // block
  <div class="intro">                                      // block in block
     <div class="intro__container"></div>
  </div>
  <div class="card__image"></div>                          // element
</div>

<div class="card card--small"></div>                       // block modifier
```

BEM: Element

can't contain blocks

can't contain elements

can have a modifiers

```
<div class="card">
  <div class="card__intro"></div>                   // element
  <div class="card__image"></div>                   // element
  <div class="card__image card__image--small"></div> // element modifier
</div>
```

BEM: Modifier

can't contain blocks, elements, modifiers

should always have a modified-selector + the modifier-itself

should always modify something (next slide)

```
<div class="card">
  <div class="card__image"></div>
  <div class="card__image card__image--small"></div> // element modifier
</div>

<div class="card card--small"></div>                // block modifier
```

BEM: Modifier

can't contain blocks, elements, modifiers

should always have a modified selector + the modifier itself

should always modify something

```
.card {
  &--small {
    color: gray; // modifies what?
  }

  &__image {
    &--small {
      padding: 5px; // modifies what?
    }
  }
}
```

→

```
.card {
  color: black; // default

  &--small {
    color: gray; // modifies default
  }

  &__image {
    padding: 20px; // default

    &--small {
      padding: 5px; // modifies default
    }
  }
}
```

Pointers

A few pointers

Pointer

Each CSS component is a Block itself

Because we use Angular… normally each file should have 1 parent selector

```
.hucs {
  &__list {}

  &__wrapper {}
}

.addresses {
  .k-label {}
}

:host ::ng-deep .k-grid {
  .k-text {}
}
```

→

```
.hucs {
  &__list {}

  &__wrapper {}

  .addresses {
    .k-label {}
  }

  :host & ::ng-deep .k-grid {
    .k-text {}
  }
}
```

Pointer

Because we have unique class names we can target directly, we should not unify modifiers with their element or block

```
.logs.logs--error {
}

.hucs__list.hucs__list--small {
}
```

$\longrightarrow$

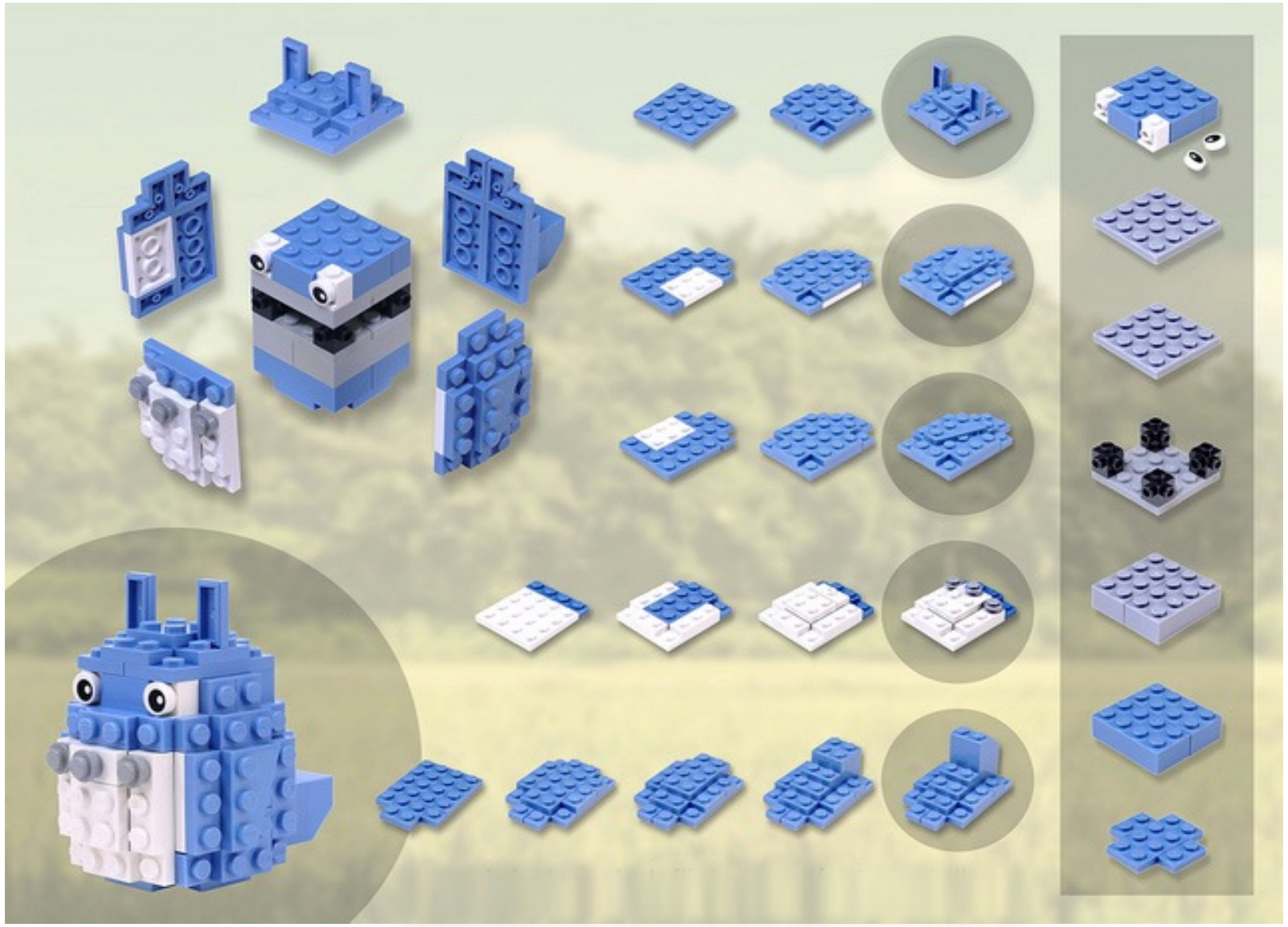
```
.logs--error {
}

.hucs__list--small {
}
```

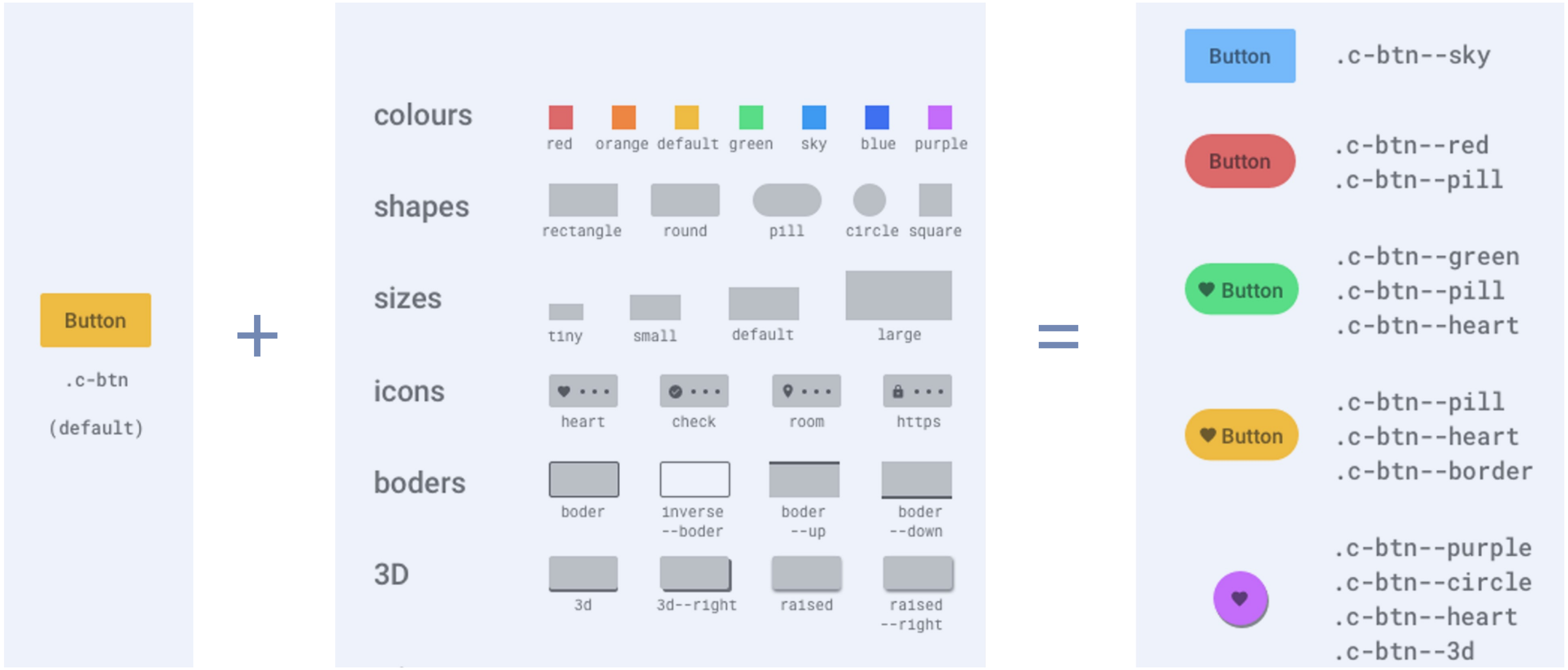
O OCSS: Object Oriented CSS

Create individual parts separately and
construct them together to build components

0 OCSS: Your LEGO childhood all over again



O OCSS: Object Oriented CSS



Example

```
<button class="c-btn">
```

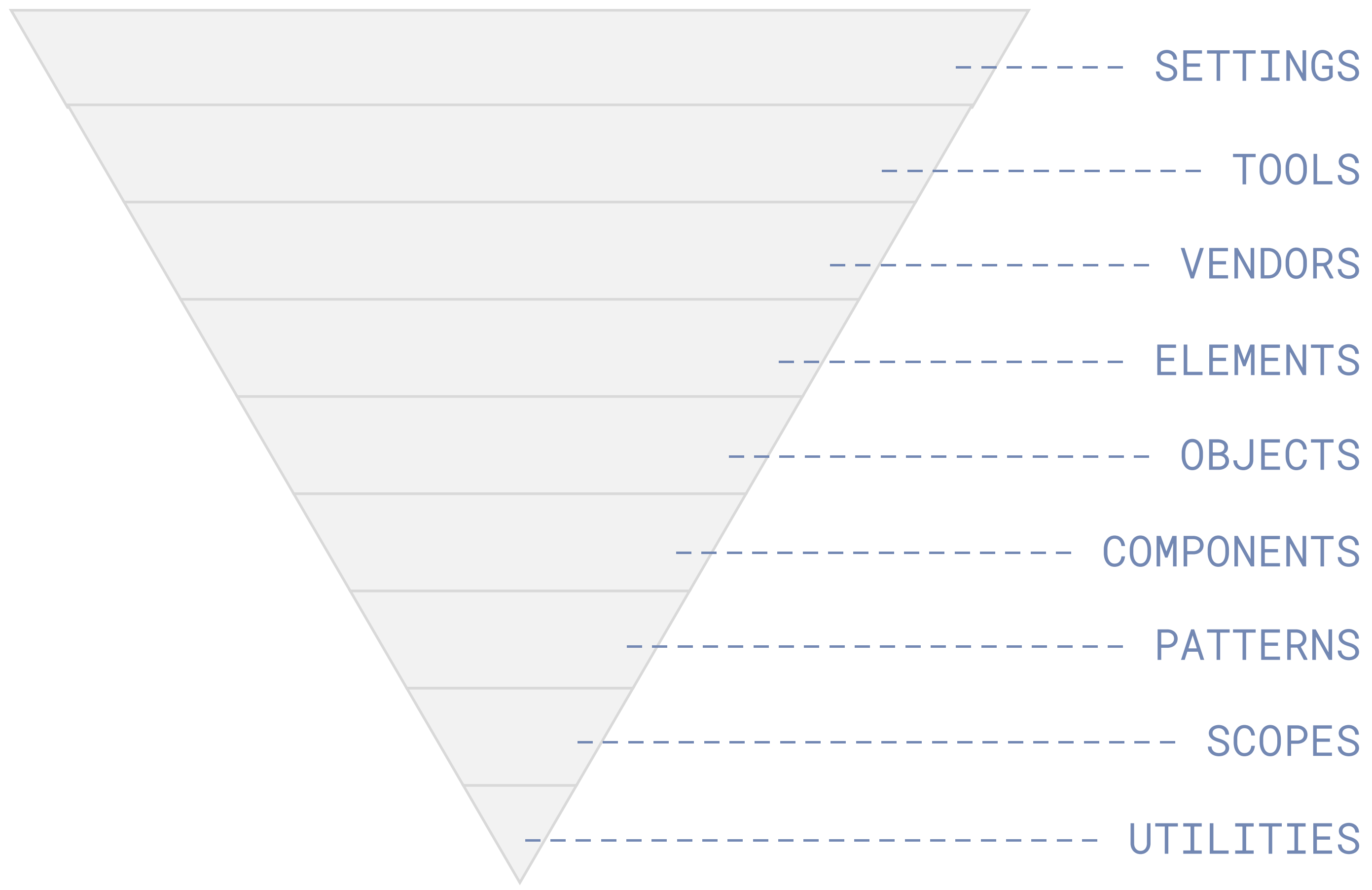
```
<button class="c-btn c-btn--sky">
```

```
<button class="c-btn c-btn--purple c-btn--circle c-btn--heart c-btn--3d">
```

I TCSS: Iverted Triangle CSS

Organise CSS by applying a structure to determine how specific to get with a specific component

ITCSS: Inverted Triangle CSS



Settings

Settings are generally a collection of variables that do not generate CSS, but are applied to classes

Examples: Base, Colour, Typography, Animation

Tools

Tools also will not produce any CSS yet. They are typically preprocessor functions that help write or extend properties on classes

Examples: Functions, Placeholders, Mixins, Media Queries

Vendors

Vendors are third-party styles that are being used in a project.
They are high in the structure so we can override them.
We also can add overrides at the end of this layer

Examples: reset.css, normalise.css, bootstrap, bootstrap-overrides

Elements

Namespace e-

Elements are the HTML native elements which we would not style base on the class names. We should provide default styles to a <a> element rather the styling a .link class.

Examples: body, h1, span, body:not(.admin-page), section

Objects

Namespace o-

Objects are used for design patterns, such as layouts, where items are being arranged rather than decorated.

Object classed are used across all pages, so if you make any change to an object class, you should be aware this affects all usages throughout the application

Examples: o-body, o-main, o-section, o-container, o-inner-container

Components

Namespace c-

A component is a small feature that makes up a part of the app.
Buttons, accordions, sliders, modal dialogs,…

Each component is fully functional by itself and does not rely on other components

Examples: c-btn, c-icon, c-masonry, c-link-underlined

Components (continued)

Namespace c-

If you use a component based framework like Angular or React, this layer will be empty because the components are already isolated and encapsulated

Patterns

Namespace p-

Patterns are components which are not scalable.
Because they are too specific and not reusable, they can't be created as components.

Example **component**: c-accordion
This can be used everywhere

Example **pattern**: p-header
This can be used on all pages, but not in the content, sidebar, …

Scopes

Namespace s-

The purpose of scope is to give us the highest specificity so we can override any styles for a specific purpose.

This layer should stay pretty empty. If you start to write a lot here, you probably have to refactor your CSS structure

Example:

```
.c-accordion.s-homepage & {
  background-color: light-blue; // only for the homepage
}
```

Utilities

Namespace u-

Sometimes you want to make changes only for a certain style in a specific place. In that case utility classes can help without changing the whole CSS structure

Because this is the last layer containing styles who should always win, you are allowed to use !important here

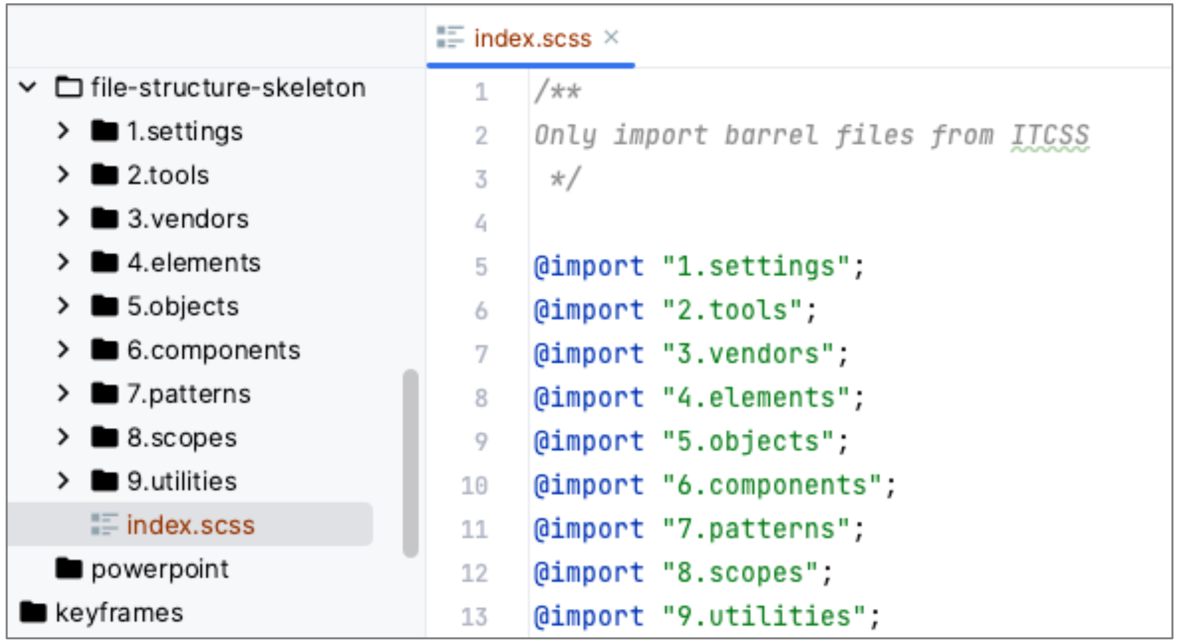
Like scope classes, if you are using too many utility classes, you should check with designer if the design can be more consistent across the app

Examples: u-clr-black, u-text-left, u-top-10, u-font-h2

Demo

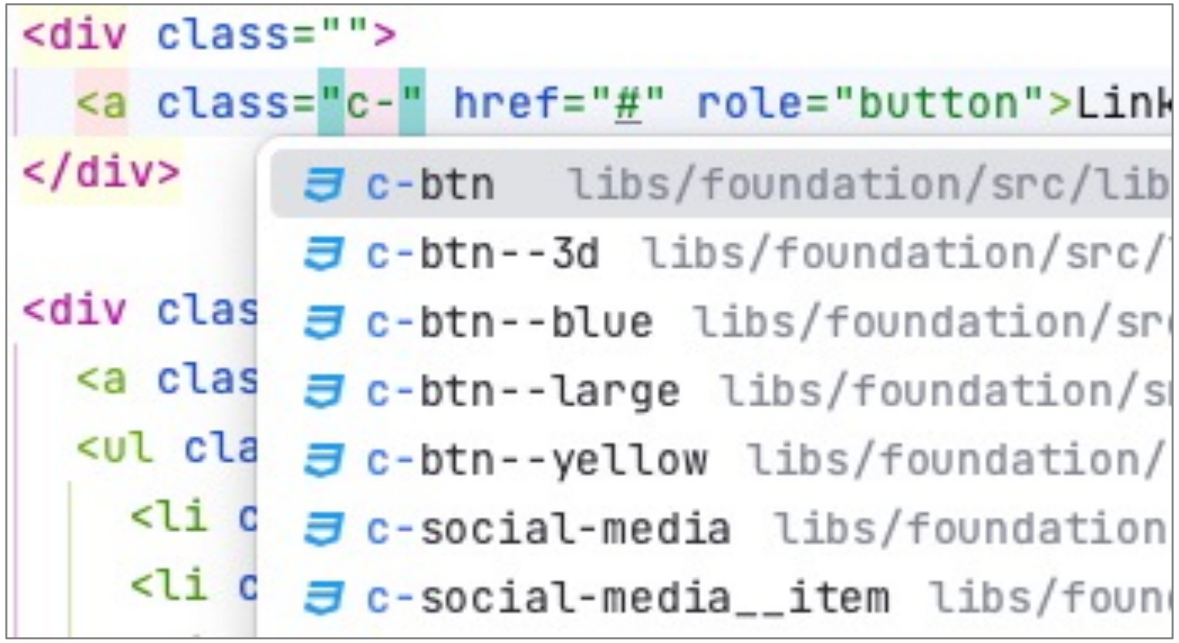
Time to see these layers in action!

ITCSS: BENEFITS



```scss
/**
Only import barrel files from ITCSS
 */

@import "1.settings";
@import "2.tools";
@import "3.vendors";
@import "4.elements";
@import "5.objects";
@import "6.components";
@import "7.patterns";
@import "8.scopes";
@import "9.utilities";
```



```html
<div class="">
  <a class="c-" href="#" role="button">Link
</div>


<div clas
  <a clas
  <ul cla
    <li c
    <li c
```

c-btn libs/foundation/src/lib
c-btn--3d libs/foundation/src/
c-btn--blue libs/foundation/sr
c-btn--large libs/foundation/s
c-btn--yellow libs/foundation/
c-social-media libs/foundation
c-social-media__item libs/foun

By using layers:

You have a better overview of overrides

You can decide more easily where styles should go

You have reusable styles (like utilities)

By using the namespaces:

You know better what you're doing

Your IDE gives you auto-suggestions