CSS architectures

it's *alive!*

@carmenansio

front-end designer

at

Desigual®

@ignaciodenuevo

front-end

at

plain concepts

Cascade and **inheritance**

The order of CSS rules **matter**

Heritage

1/2

initial | inherit | unset

Example

```
h1 {
    color: black;
}

* {
    color: red;
}
```

Heritage

1/2

Example

```css
html {
  color: black;
  font-size: 16px;
}

.nav .body-text {
  color: red;
  margin-right: 1em;
}
```

Computed

```css
html {
  color: black;
  font-size: 16px;
}

.nav .body-text {
  color: red;
  font-size: 16px;
  margin-right: 1em;
}
```

Specificity

determines **which CSS rule** is applied by the browsers.

0 | 0 | 0 | **1**

element {}

::pseudo-element {}

0 | 0 | **1** | 0

.class {}

[attribute] {}

:pseudo-class {}

0 | **1** | 0 | 0

#id {}

1 | 0 | 0 | 0

property: !important;

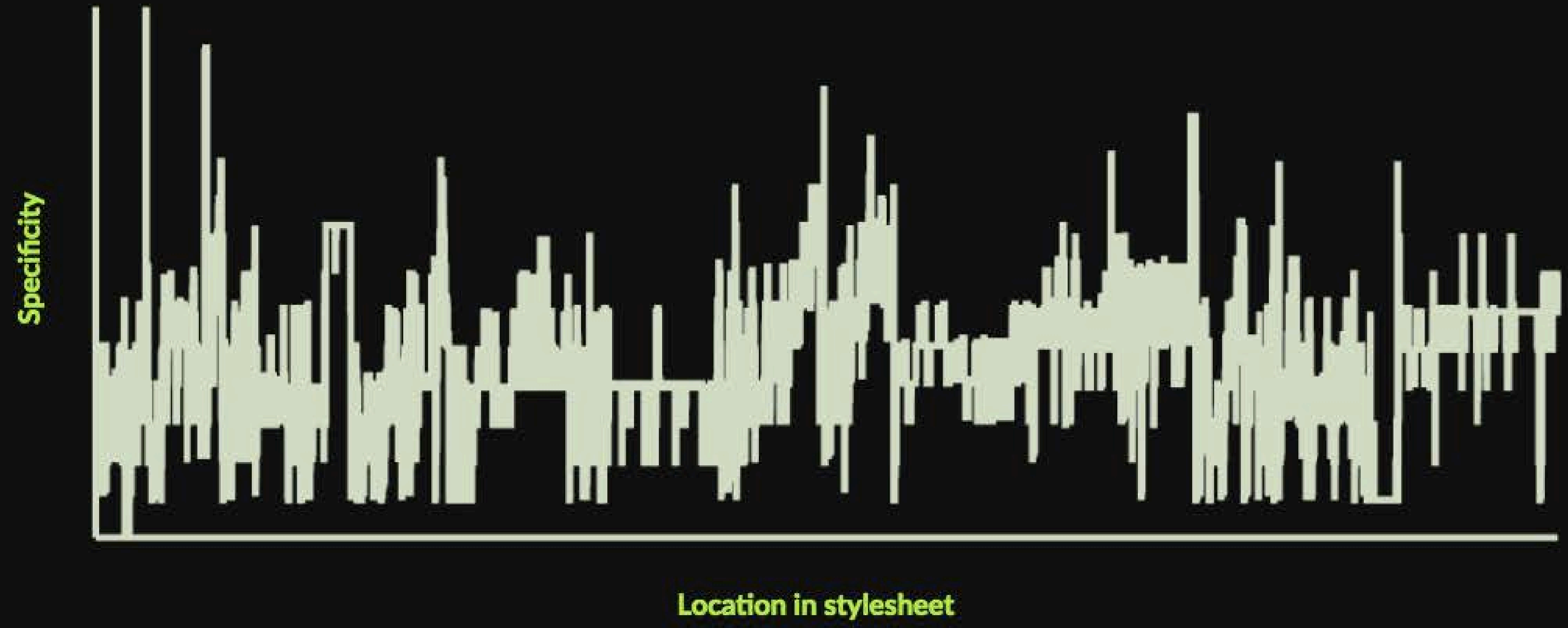
style="property: in-line;"

0 | 0 | 0 | 0

>, +, *, ~, :not()

@media

Specificity Graph



Keep **specificity** levels low

In order to **avoid** cascade problems.

- Sass nesting.
- Class concatenation.
- Using ID's.
- Inline styles.
- !important.

Keep **specificity** levels low

1|1|2|4

```
body {
  &.header {
    &.nav {
      ul {
        li {
          a#link {
            color: red !important;
          }
        }
      }
    }
  }
}

body.header.nav ul li a#link {
  color: red !important;
}
```

Specificity **shame**

HTML

```
<div>
  <div>
    <div>
      <div>
        <div>
          <div>
            <div>
              <div>
                <div>
                  <div>
                    <div>
                      <div class="frankestein">
                        Igor
                      </div>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

CSS

```
.frankestein {
  color: green;
}

div div div div div div div div div div div div {
  color: red;
}
```

class specificity

0 | 0 | 1 | 0

div's specificity

0 | 0 | 0 | 12

!important

Adding **!important** to a declaration **is the same** as **declaring all** of its sub-properties as **!important**.

```
.card {
  margin: 1em !important;
}
```

=

```
.card {
  margin-bottom: 1em !important;
  margin-left: 1em !important;
  margin-right: 1em !important;
  margin-top: 1em !important;
}
```

Naming conventions

Naming **CSS** is **really hard**

Naming **CSS** classes

1/4

Separation of Concerns: **CSS** that **depends on HTML**.

- Naming your classes **based on your content** .author-bio treats your HTML as a dependency of your CSS.
- **In this model, your HTML is restyleable, but your CSS is not reusable.**

Naming CSS classes

2/4

Mixing Concerns: HTML that **depends on CSS**.

- Naming your classes in a **content-agnostic** way after the repeating patterns in your UI

  `.media-card`

  treats your CSS as a dependency of your HTML.
- In this model, your **CSS is reusable** , but your HTML is not restyleable.

Naming CSS classes

3/4

```
.card
.btn, .btn--primary, .btn--secondary
.badge
.card-list, .card-list-item
.img--round
.modal-form, .modal-form-section
```

✗ `.nav ul li a img {}`   ✓ `.user-card__portrait {}`

Naming **CSS** classes

4/4

Functional class: names (best ones) **based on its content** or **on its presentation**.

```
.positive-button {}
```

Content-based: class names as your site grows they're **not good for style reuse**.

```
.submit-button {}
```

Presentational: class names are very **self-describing**.

```
.green-button {}
```

Naming Sass variables

- Avoid using **color names**
- Use **six hexadecimal** values
- Use **lowercase** hexadecimal

BEM

Bloc | Element | Modifier

- Communicates **purpose or function**.
- Communicates **component structure**.
- Sets a consistent **low-level specificity** for styling selectors

✗

```
<button class="btn--secondary">
  Purchase
</button>

.btn--secondary {
  display: inline-block;
  color: green;
}
```

✓

```
<button class="btn btn--secondary">
  Purchase
</button>

.btn {
  display: inline-block;
  color: blue;
}

.btn--secondary {
  color: green;
}
```

BEM

Bloc | Element | Modifier

Advantages

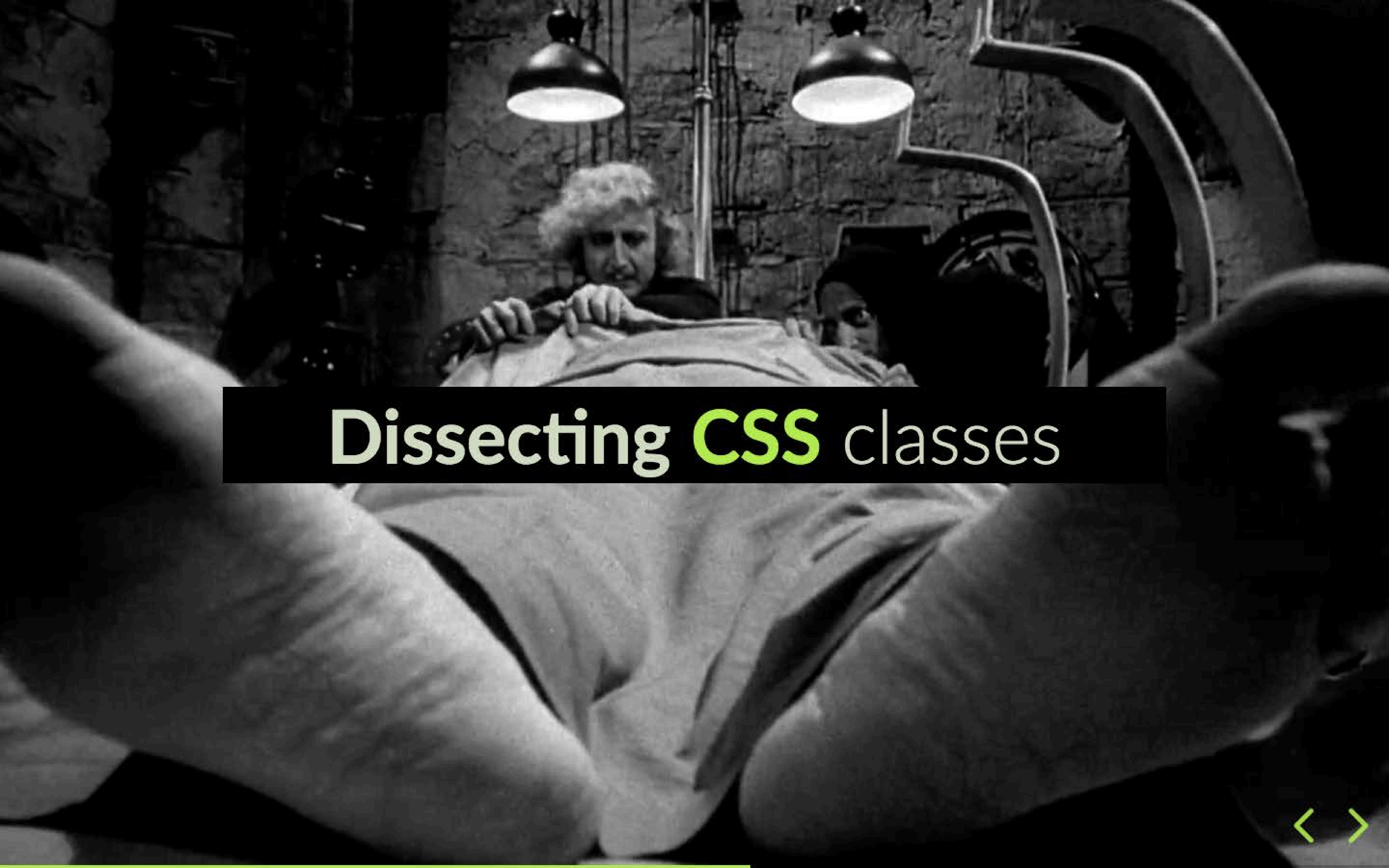
Low specificity

Disadvantages

Dirty HTML

Namespaces

- **o-**: object
- **c-**: component
- **u-**: utility class
- **t-**: theme
- **s-**: context or scope
- **is-**, **has-**: state (SMACSS)
- _: hack
- **js-**: JavaScript
- **qa-**: quality assurance



Dissecting CSS classes

.frnk-js-btn--primary {}

- **.frnk-:** brand prefix {}
- **js-:** JavaScript behavior {}
- **.btn:** class {}
- **--primary:** modifier {}

Architectures

Preprocessor **agnostic**

Why use an **Architecture?**

- **Scalability**
- Lack of **documentation**
- Lack of structure, **quality assurance**
- Lack of **knowledge** (about CSS or the project itself)
- Different styles, preferences, **ways of working** (linterns)
- Adding new styles to the **end of stylesheets**

OOCSS object oriented CSS

Good: reducing the amount of code by reusing it **(DRY principle)**.

Bad: complex support.

When you change the style of a particular element, you will most likely have to change not only CSS (because most classes are common), but also add classes to the markup.

OOCSS object oriented CSS

Separate structure and skin

✗

```
.nav {
  border-style: solid;
  border-color: red;
  border-width: 1px;
  margin-left: auto;
  margin-right: auto;
  padding-bottom: 1em;
  padding-top: 1em;
}

.foo {
  border-style: solid;
  border-color: red;
  border-width: 1px;
  margin-left: .5em;
  margin-right: .5em;
  padding-bottom: 1.2em;
  padding-top: 1.2em;
}
```

✓

```
.nav {
  margin-left: auto;
  margin-right: auto;
  padding-bottom: 1em;
  padding-top: 1em;
}

.foo {
  margin-left: .5em;
  margin-right: .5em;
  padding-bottom: 1.2em;
  padding-top: 1.2em;
}

.gmail {
  border-style: solid;
  border-color: red;
  border-width: 1px;
}
```

OOCSS object oriented CSS

Separate container and content

✗

```
.nav h3 {
  color: red;
  font-family: 'Lato';
  font-size: 16px;
  font-weight: 900;
}

.foo h3 {
  color: red;
  font-family: 'Lato';
  font-size: 16px;
}
```

✓

```
h3 {
  color: red;
  font-family: 'Lato';
  font-size: 16px;
}

.text-bold {
  font-weight: 900;
}
```

SMACSS

Scalable and Modular Architecture for CSS

Base: applies to HTML, no class/ID selectors.

Layout: big page sections.

`.header, .sidebar, .footer`

Module: encapsulation modules, re-usable.

State: overrides defaults.

`.is-opened, .is-active`

Theme

7-1 pattern

1/2

```
sass/
|
|– abstracts/
|   |– _variables.scss    # Sass Variables
|   |– _functions.scss    # Sass Functions
|   |– _mixins.scss       # Sass Mixins
|   |– _placeholders.scss # Sass Placeholders
|   …
|
|– base/
|   |– _reset.scss        # Reset/normalize
|   |– _typography.scss   # Typography rules
|   …
|
|– components/
|   |– _buttons.scss      # Buttons
|   |– _carousel.scss     # Carousel
|   |– _cover.scss        # Cover
|   |– _dropdown.scss     # Dropdown
|   …
|
```

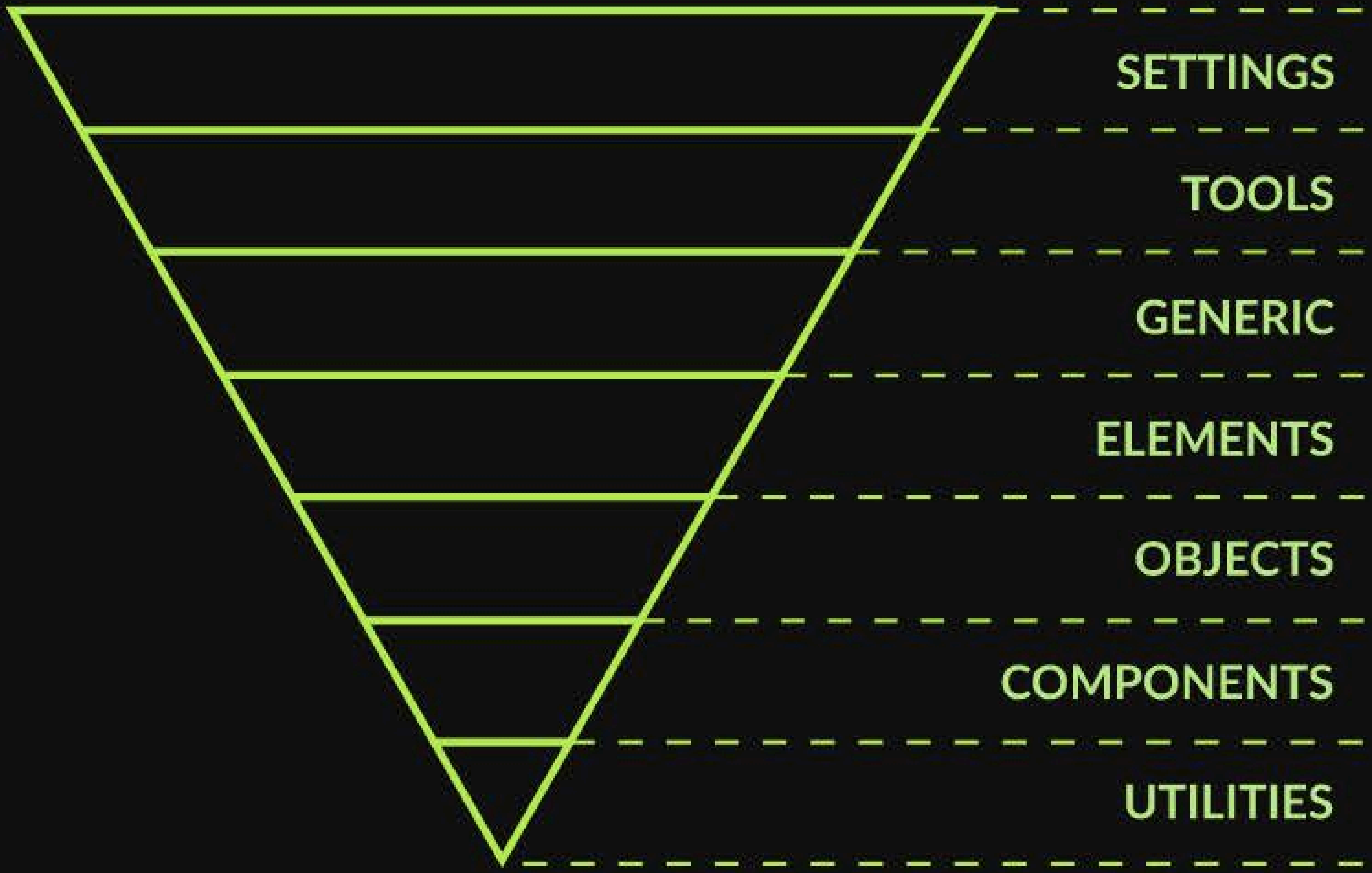
7-1 pattern

2/2

```
|
|– layout/
|   |– _grid.scss         # Grid system
|   |– _header.scss       # Header
|   |– _footer.scss       # Footer
|   …
|
|– pages/
|   |– _home.scss         # Home specific styles
|   |– _contact.scss      # Contact specific styles
|   …
|
|– themes/
|   |– _theme.scss        # Default theme
|   |– _admin.scss        # Admin theme
|   …
|
|– vendors/
|   |– _bootstrap.scss    # Bootstrap
|   |– _jquery-ui.scss    # jQuery UI
|   …
|
`– main.scss              # Main Sass file
```

ITCSS inverted triangle CSS

1/3



ITCSS inverted triangle CSS

2/3

```
sass/
|
|— settings/
|    |— _config.scss # Project-level config
|    |— _core.scss # Core setup
|    |— _global.scss # Variables
|    …
|
|— tools/
|    |— _font-size.scss # Baseline mixin
|    |— _clearfix.scss # Clearfix mixin
|    |— _hidden.scss # Hidding mixin
|    …
|
|— generic/
|    |— _box-sizing.scss # Default `box-sizing`
|    |— _normalize.scss # Normalize.css vendor
|    |— _reset.scss # A tiny reset
|    …
|
|— elements/
|    |— _page.scss # Global `font-size` & `line-height`
|    |— _headings.scss # Default styles for headings
|    |— _images.scss # Default images styles
|    …
|
```

ITCSS inverted triangle CSS

3/3

```
|
|— objects/
|    |— wrapper.scss # Page constraint object
|    |— layout.scss # Generic layout module
|    |— media.scss # Content side by side
|    …
|
|— components/
|    |— _buttons.scss # Default buttons styles
|    |— _nav.scss # Default nav styles
|    …
|
|— vendor/ (this folder is not mean to be used in ITCSS)
|    |— _bootstrap.scss # Bootstrap
|    …
|
|— utilities/
|    |— _widths.scss # Widths helper classes
|    |— _headings.scss # Headings helper classes
|    |— _spacings.scss # Spacings helpe classes
|    …
|
`_all.scss
```

Tips & Tricks

Principles

Software Design Principles

- **DRY:** Don't repeat yourself
- **KISS:** Keep it Simple Stupid
- **YAGNI:** You "Ain't Gonna Need It"
- **SR:** Single Responsibility
- **OS:** Open-Close objects are open to extension but closed to modification)

Recommendations

- The **Broken Window Theory**.
- **Don't** leave **automatizations** decide for you.
- **@imports** order are !important.
- Use of **token variables**.

Reduce, Reuse & Recycle

- **Reduce** means writing the shortest chain of elements possible in selectors.
- **Reusing** involves creating generic classes instead of overly specific ones.
- **Recycling** involves better leveraging the cascade to cut down on redundant style declarations.

CSS order

Comments

- **Place comments on a new line** above their subject.
- Keep **line-length to a sensible maximum** , e.g., 80 columns.
- Make liberal use of comments to break CSS code into discrete sections.
- Use **"sentence case"** and consistent **text indentation**.

```
/// Make a context based selector a little more friendly
/// @author Hugo Giraudel
/// @param {String} $context
@mixin when-inside($context) {
  #{$context} & {
    @content;
  }
}
```

CSS order

Declarations

- When multiple classes uses the same property, **use one line for each**.

```
.frnk-js-btn--primary,
.frnk-js-btn--secondary,
.frnk-js-btn--tertiary {
    font-family: 'Lato';
}
```

CSS order

Alphabetically

```
.body-text {
  border-color: red;
  border-style: solid;
  border-width: 1px;
  padding-left: 20px;
  padding-right: 20px;
  z-index: 666;
}
```

- @extend
- @include
- Media queries
- Modifiers
- Parent selectors
- States

By type

- Positioning
- Display & Box Model
- Color
- Text
- Other
- @extend
- @include
- Media queries
- Modifiers
- Parent selectors
- States

CSS order

```
// Buttons can be applied to any HTML element that is used to trigger a user
// action (e.g. following a call to action link, submitting a form).
//
// 1. Line differently sized buttons up a little nicer.
.btn,
.link {
    // Avoid @extends: creates unnecesary CSS
    @extend .text;
    @include fluid-font();
    color: red;
    vertical-align: middle; /* [1] */

    @media (min-width: 64em) {
        color: pink;
    }

    &--secondary {
        width: 100%;
    }

    .nav & {
        margin-left: auto;
    }

    &.is-disable {
        opacity: .5;
    }
}
```

Architecture **smells**

- Using a large number of **font-size**.
- !important should only ever be used **proactively**, not reactively.
- CSS should be **location** independent.
- Styling **HTML elements**.
- **Qualified** selectors.
- **Nesting** more than 3 levels.
- **Undoing** styles.
- **Magic** numbers (27px).
- **@extend**
- Limit string **concatenation** for classes

Resources

Codemotion 2017

Thanks

Any **Question?**