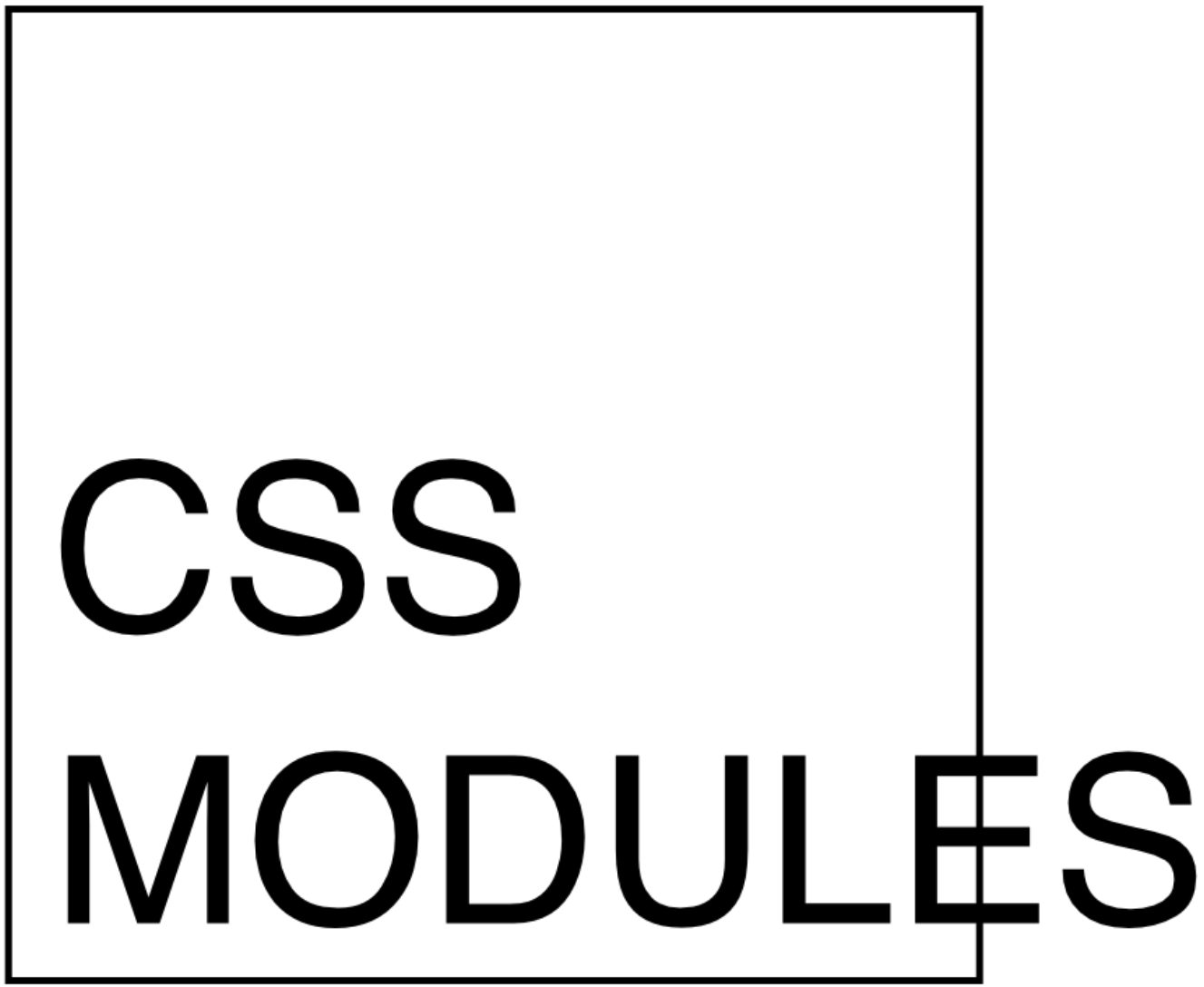

CSS
MODULES

ABOUT ME

- Veit Lehmann
- Front-end Developer at LOVOO
- github.com/levito

WHAT ARE MODULES?

MODULES

- Composable, reusable pieces of code
- Do one thing well
- Loose coupling
- Easy to maintain, replace and remove

JS MODULES

- CommonJS in Node.js
- Core Node.js modules and NPM

  → **New age of JS Development**

BEFORE JS MODULES

- Everything global by default
- 3rd party scripts could break everything
- AMD/UMD for modular JS in the browser

BROWSERIFY/WEBPACK

- CommonJS Modules for the browser
- ES6 modules via Babel

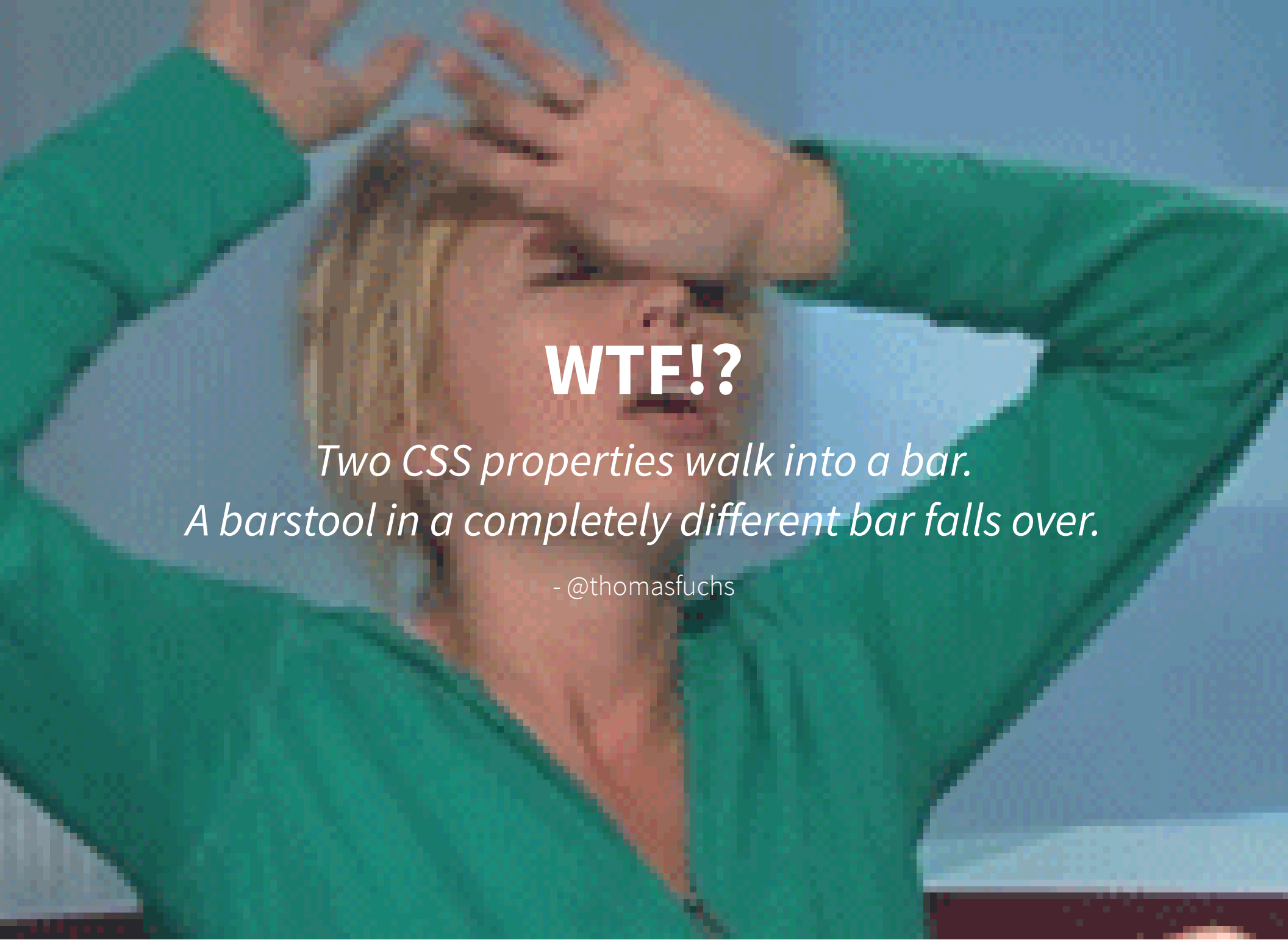
→ **Boilerplate-free modules for the browser!**

CSS BEFORE MODULES

- Everything global
- How to deal with it?

  → **Naming Conventions**

WHAT HAPPENS WITH SLOPPY CSS?


WTF!?
Two CSS properties walk into a bar.
A barstool in a completely different bar falls over.
- @thomasfuchs

CSS NAMING CONVENTIONS

DEEP NESTING

- CSS Selectors mirror the DOM structure
- Popular in the 2000s

DEEP NESTING PROS

- Isolation
- Clean, semantic markup

DEEP NESTING CONS

- No CSS reusability
- CSS bloat (append-only code)
- Lots of overwriting CSS
- Specifity wars
- Visual inconsistencies

BEM

- Block__Element--Modifier
- Add more classes to the markup
- Reuse lots of CSS rules
- Short selector chains
- Naming convention to avoid name clashes
- e. g. `button button--danger`, `button__icon button__icon--after`

BEM PROS

- Easy to write and reason about in markup
- No CSS bloat but reuse
- Naming convention makes it safe

BEM CONS

- Lots of discipline needed to write CSS
- Fine-tuning is hard
- Verbose and annoying to write
- Markup bloat
- Corner cases hard to name

ATOMIC CSS

- Combine single purpose classes
- Aim: No need to write new CSS once implemented
- e. g. `padding-small color-red text-center`

ATOMIC CSS PROS

- Easy to write and reason about in markup
- Even more CSS reuse than BEM
- Fine-tuning is easier

ATOMIC CSS CONS

- Rebrushing is hard
- Even more markup bloat than BEM
- Not component-oriented but huge pile of globals
- Feels like inline-styling
- Fine-tuning has its limits
- Pseudo selectors/attributes hard to style
- Responsive styling is painful

INLINE STYLING IN JS (JSS)

- Came up in the React community
- No external stylesheets
- Dynamic style attributes instead

JSS PROS

- Everything is in the component
- No class name clashes
- No specifity wars
- Code sharing between JS and Style
- Dynamic styling via JS opens possibilities

JSS CONS

- No pseudo elements → markup needed
- No @media queries and pseudo selectors
  → reimplement with JS
- Responsive styling is really painful
- Styles in objects → undefined order of properties
  → `margin` might overwrite `margin-top` defined later
- Awkward syntax
- Browsers are not optimized for that

STATE-OF-THE-ART TODAY

- Combine BEM with *trumps* for corner-case adjustments
- Trumps are like Atomic CSS, but `!important` to always win
- Visual-semantic naming:
  - not `button--checkout`, but `button--primary`
  - not `text--lightgray`, but `text--weak`
- Thoughtful use of Pre-/Postprocessing

SIDE NOTE: PREPROCESSORS

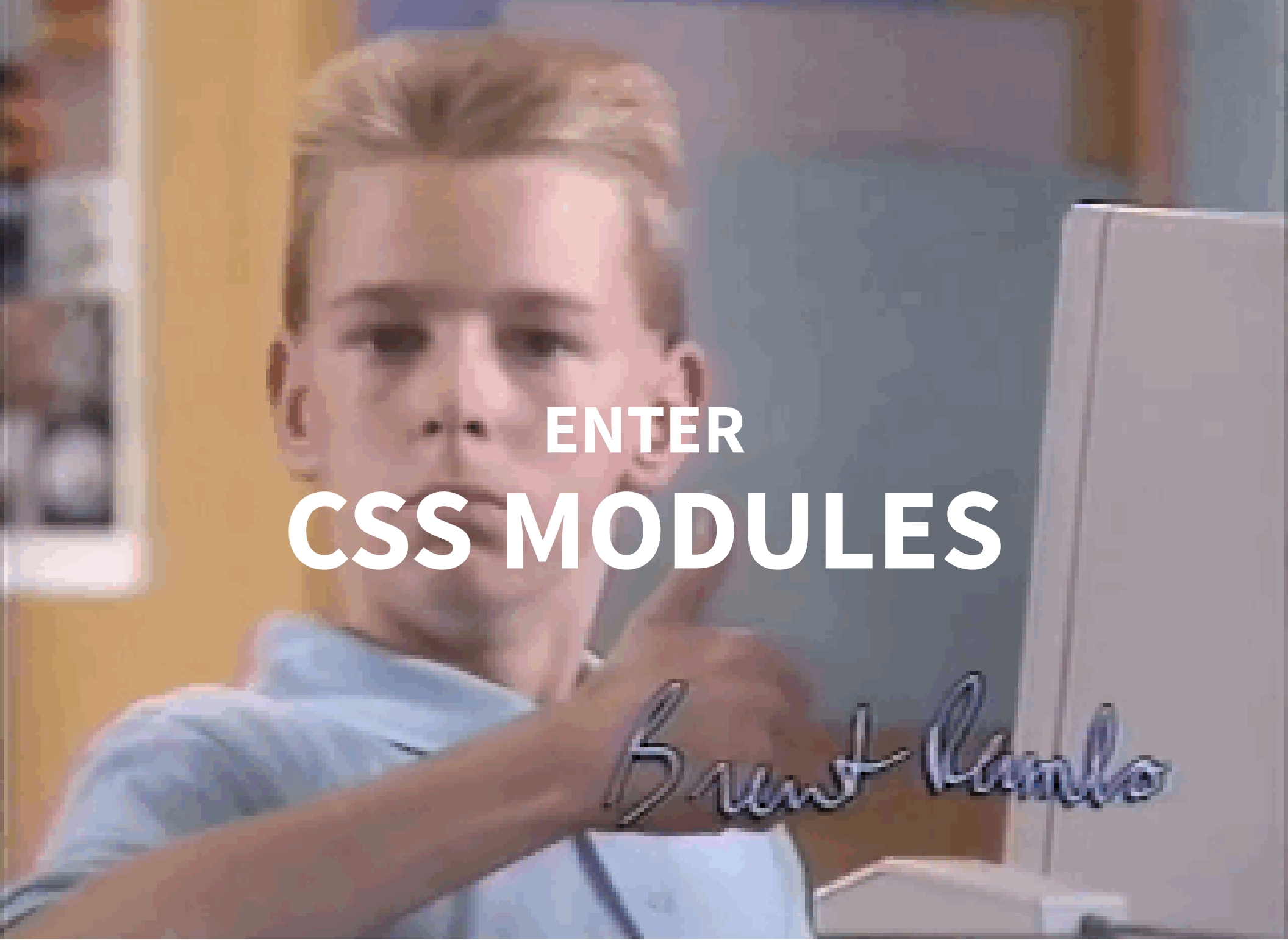
- Variables, mixins, extends, functions and nesting
  → DRY CSS authoring
- **But: Markup is unaffected**

WHAT IF WE COULD

- Ensure *non-conflicting class names*?
- Keep source *markup clean* like with deep-nesting?
- Keep CSS *small and maintainable*?
- Make CSS pieces easily *removable/replacable*?

THE MISSING PIECE

- Rewrite the *application* of styles in the *markup*
- i. e. rewrite `class="..."`


ENTER
CSS MODULES

CSS MODULES

- Put CSS *into the component*
- *Import* it into the view
- Class names remain *local* to the component
- @media queries, pseudo elements/classes, animations etc. still work

HOW TO USE
CSS MODULES

:LOCAL BY DEFAULT

```
/* my-component.css */
.foo { color: red; }
```

```
// my-component.js
import styles from 'my-component.css';

export `<div class="${styles.foo}">
            Styles of .foo are applied
        </div>`;
```

OUTPUT

```
.s0m3_rnd-Ha5h { color: red; }
```

```
<div class="s0m3_rnd-Ha5h">
    Styles of .foo are applied
</div>
```

:LOCAL BY DEFAULT

- Prevents naming clashes like BEM
- *... but without its problems*
  - No verbose naming
  - No accidental clashes
  - No head-scratching in corner cases

:GLOBAL STYLES

```
:global .foo { color: red; }
.bar { font-weight: bold; }
.foo :global .baz { text-decoration: underline; }
```

```
export `<div class="${styles.foo}">
            <span class="${styles.bar}">
                Lorem <span class="${styles.baz}">ipsum</span>
            </span>
        </div>`;
```

OUTPUT

```
.foo { color: red; }
.DFGDFG { font-weight: bold; }
.DFGDFG .baz { text-decoration: underline; }
```

```
<div class="foo">
    <span class="DFGDFG">
        Lorem <span class="baz">ipsum</span>
    </span>
</div>
```

:GLOBAL STYLES

- When you need to break out
  - like content from a RTE
  - 3rd party code like date-pickers etc.

COMPOSITION

```
.foo { color: green; }
.baz {
    composes: foo;
    composes: bar from '../some-shared.css';
    font-weight: bold;
}
```

```
export `<div class="${styles.baz}">
            Styles of .foo, .bar and .baz are applied
        </div>`;
```

OUTPUT

```
.ASDASD { color: green; }
.QWEQWE { font-weight: bold; }
```

```
<div class="ASDASD YXCYXC QWEQWE">
    Styles of .foo, .bar and .baz are applied
</div>
```

COMPOSITION

- Code reads like Sass `@extend`s
- *... but without its problems*
- Output like Atomic CSS
  - No massive selector chains
  - No attribute duplication
  - But more classes on elements

VALUES/VARIABLES

```
/* colors.css */
@value blue: #0c77f8;
@value red: #d00020;
@value green: #aaf200;
```

```
@value colors: "./colors.css";
@value blue, red, green from colors;

.button { color: blue; }
```

OUTPUT

```
.BTN987XY { color: #0c77f8; }
```

WHAT HAVE WE GAINED?

- *Clean source* markup (like deep nesting)
- *Non-conflicting* class names (like BEM)
- *Slim CSS* via composition (like Atomic CSS)
- *DRY Code* with variables etc. (like preprocessors)

USAGE ADVICES

PREPROCESSORS?

- Possible, but not needed
  - you don’t want to nest
  - variables built-in
  - better alternatives to mixins and extends

ALTERNATIVE

- More syntax needed? Add PostCSS plugins
- Awesome companion: CSSnext
- But remember: KISS

GOOD FIT

- SPAs, expecially React and Angular
- Complex, modular UIs
- Sustaining projects
- Bigger frontend teams

MAYBE NOT

- CMS projects
- Simple campaign pages

CAN I USE IT?

- Webpack: *css-loader* in modules mode
- Browserify: *css-modulesify*
- JSPM: *jspm-loader-css-modules*
- Node.js: *css-modules-require-hook*
- Other: *postcss-modules* (generates mapping json)

USEFUL RESOURCES

- glenmaddern.com/articles/css-modules
- github.com/css-modules/css-modules
- medium.com/seek-ui-engineering/the-end-of-global-css-90d2a4a06284
- medium.com/@webPapaya/css-architectures-in-2016-fab5d14c9b6e

THAT'S IT

THANKS!