Styling Mechanisms

The ecosystem of tooling around styling frontend applications is chaotic, especially in the React ecosystem!

Today styling it is very much entangled with frontend frameworks and component libraries.

Concerns

Scoping

Colocation

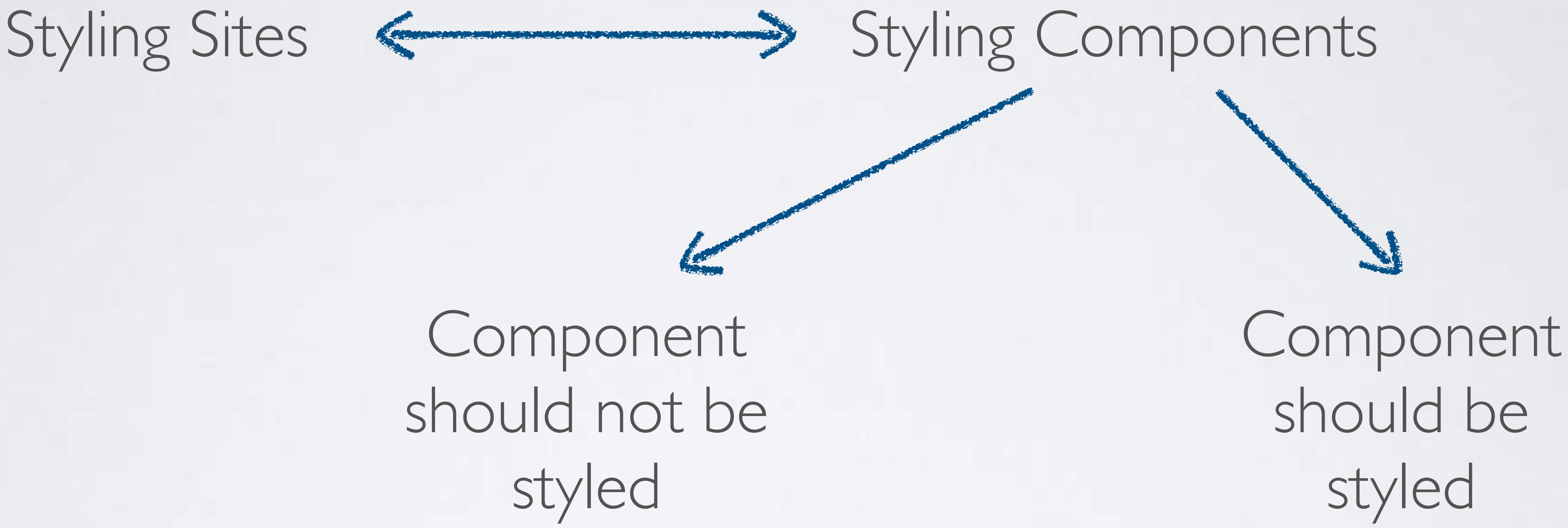
Developer Experience

Performance

Dynamic Styling

Angular, Vue, Svelte offer an integrated styling mechanism.
React, Solid, Preact do not offer a specific styling mechanism.

Styling Philosophies



Styling Sites

Styling Components

Component should not be styled

"The design of the components should be separate from their implementation."

Component should be styled

"Components should be a higher level of abstraction and solve the topic of styling (design systems)"

History

CSS Frameworks: Bootstrap, Bulma

- https://getbootstrap.com/
- https://bulma.io/

Critique: missing customizability

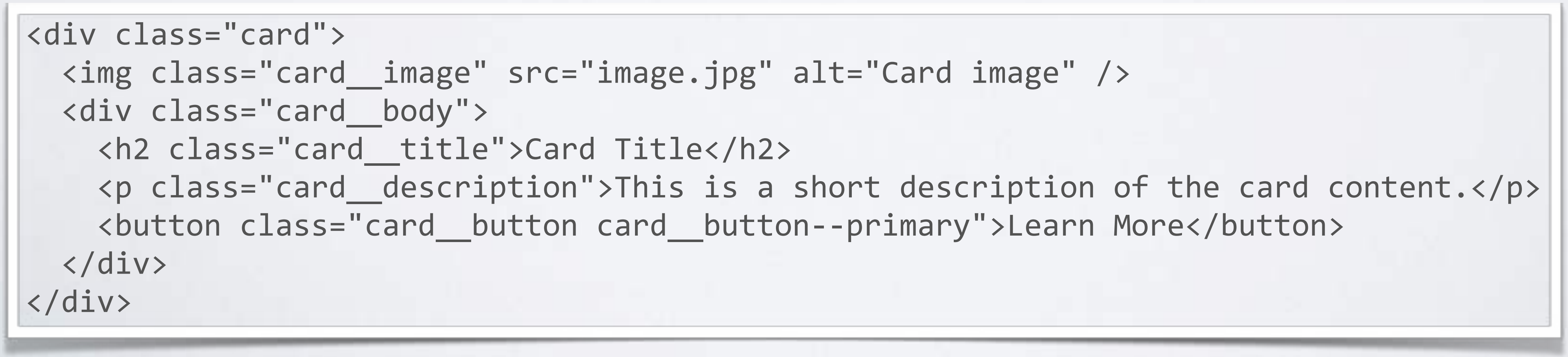
Conceptually not aligned with abstractions of component-based frameworks.

Consequence:

- https://react-bootstrap.netlify.app/
- https://ng-bootstrap.github.io/#/home
- https://valor-software.com/ngx-bootstrap/

Naming Conventions: most popular BEM (Block-Element-Modifier)

https://en.bem.info/



```
<div class="card">
  <img class="card__image" src="image.jpg" alt="Card image" />
  <div class="card__body">
    <h2 class="card__title">Card Title</h2>
    <p class="card__description">This is a short description of the card content.</p>
    <button class="card__button card__button--primary">Learn More</button>
  </div>
</div>
```

CSS Preprocessors

"CSS with superpowers"

SASS: Syntactically Awesome Style Sheets

https://sass-lang.com/

Alternatives: less https://lesscss.org/, stylus https://stylus-lang.com/

CSS Preprocessors have first-level integration in Vue and Angular projects:
https://vuejs.org/api/sfc-spec.html#pre-processors
https://angular.dev/guide/components/styling

However: CSS added many powerful features in recent years, that make preprocessors less needed (i.e. variables, nesting ...)

Main Features of Sass

using variables
Splitting into multiple files
Nested declarations

Native modern CSS constructs:

- custom properties:
  https://developer.mozilla.org/en-US/docs/Web/CSS/
  CSS_cascading_variables/Using_CSS_custom_properties
- CSS @import
  https://developer.mozilla.org/en-US/docs/Web/CSS/@import
- Native CSS nesting
  https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_nestin
  Using_CSS_nesting

CSS
MODULES

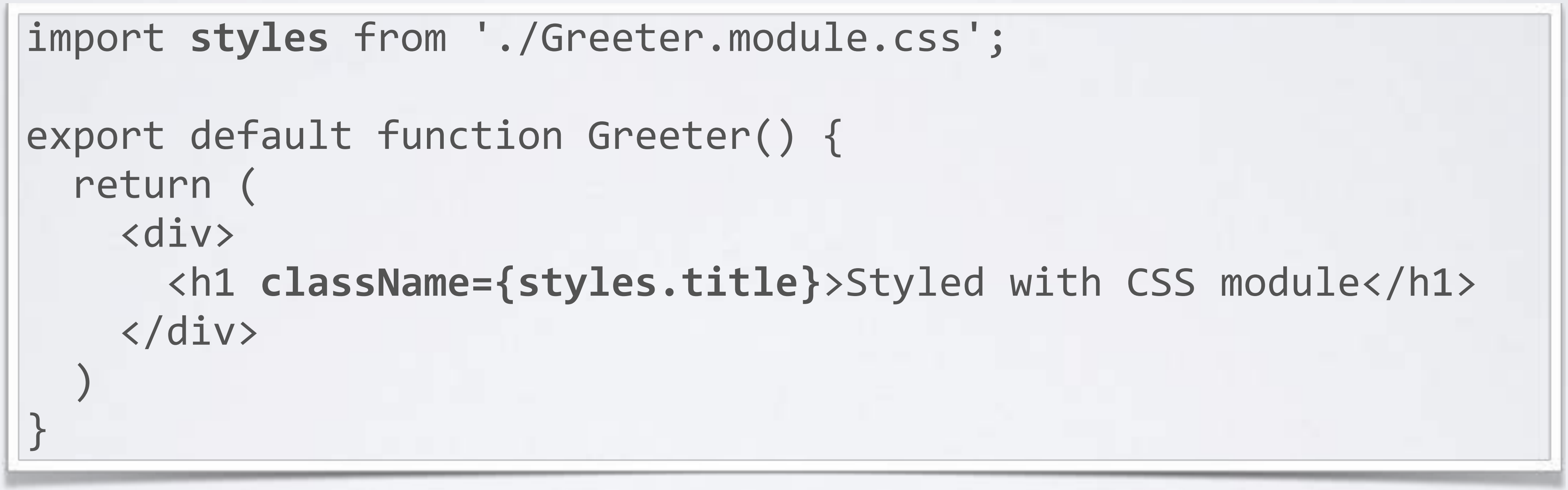
CSS Modules

https://github.com/css-modules/css-modules/

Expose scoped CSS class names to JavaScript

(can also be combined with sass)

Mostly usded in JSX-based frameworks.



```
import styles from './Greeter.module.css';

export default function Greeter() {
  return (
    <div>
      <h1 className={styles.title}>Styled with CSS module</h1>
    </div>
  )
}
```

Implemented in bundlers Vite, Parcel, Rspack, Turbopack, Bun ...
https://github.com/css-modules/css-modules/blob/master/docs/get-started.md

Vue implements CSS modules in Single File Components:
https://vuejs.org/api/sfc-css-features#css-modules

History: CSS-in-JS

(React and JSX-based Frameworks)

CSS-in-JS was the idea to use JS to "define" the styling, like JSX for templating.

not popular any more

This was very popular in the React ecosystem ~2016-2022.
Today its rather a niche/legacy approach.
Several reasons lead to a quick demise of CSS-in-JS: general SSR, React Server Components, the rise of Tailwind, the end of the zwero-interest-rate in the US ...

💅 styled-components: in maintenance mode since march 2025
https://opencollective.com/styled-components/updates/thank-you

👩‍🎤 emotion: no support for Next.js since 2022
https://emotion.sh/docs/introduction

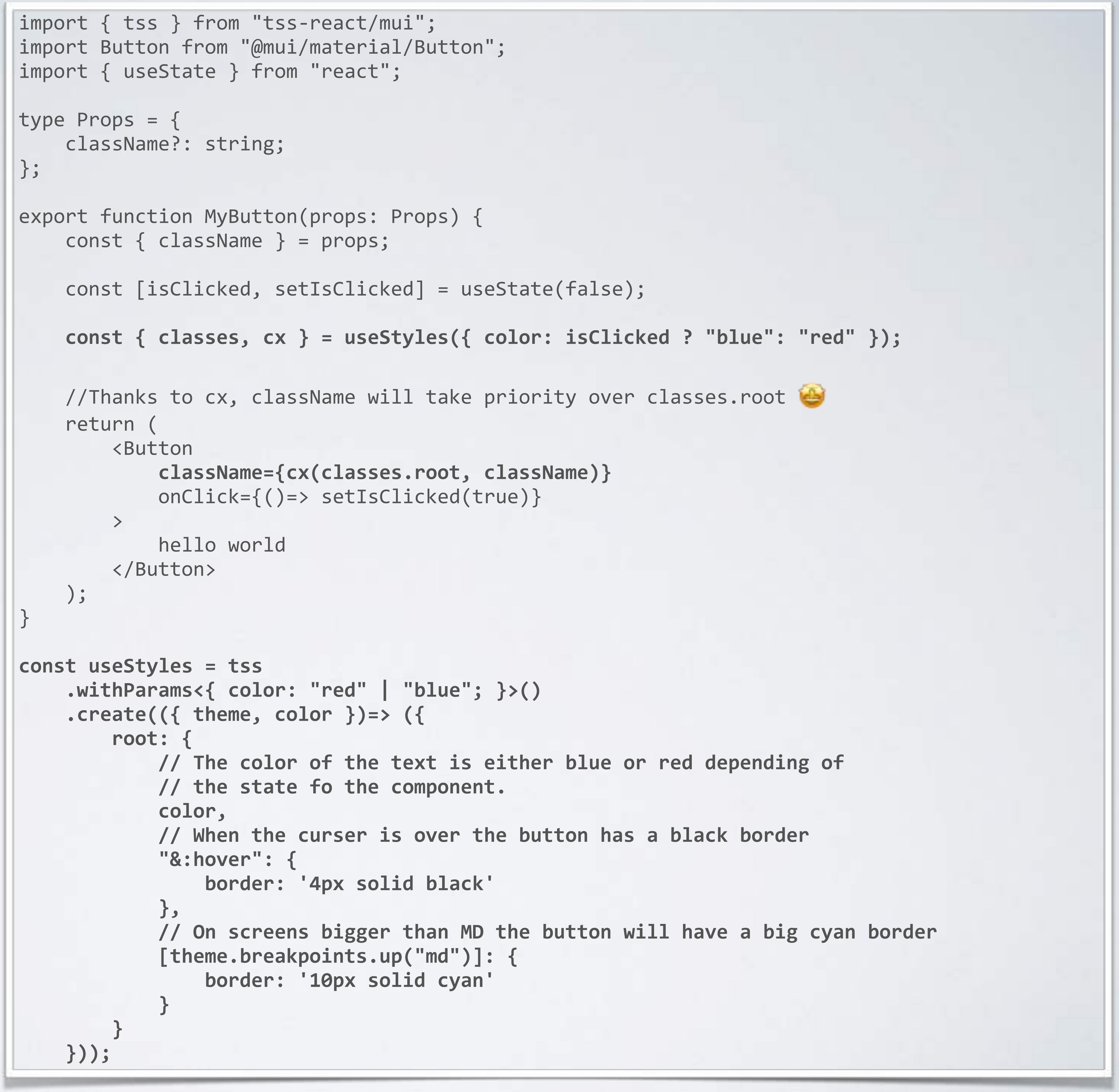
Many others: styleX, vanilla-extract, pigmentcss, Panda CSS, Linaria, tss-react, styled-jsx, jss ...

support and project state often unclear ...

https://nextjs.org/docs/app/guides/css-in-js
https://www.joshwcomeau.com/react/css-in-rsc/

CSS-in-JS Example

https://docs.tss-react.dev/

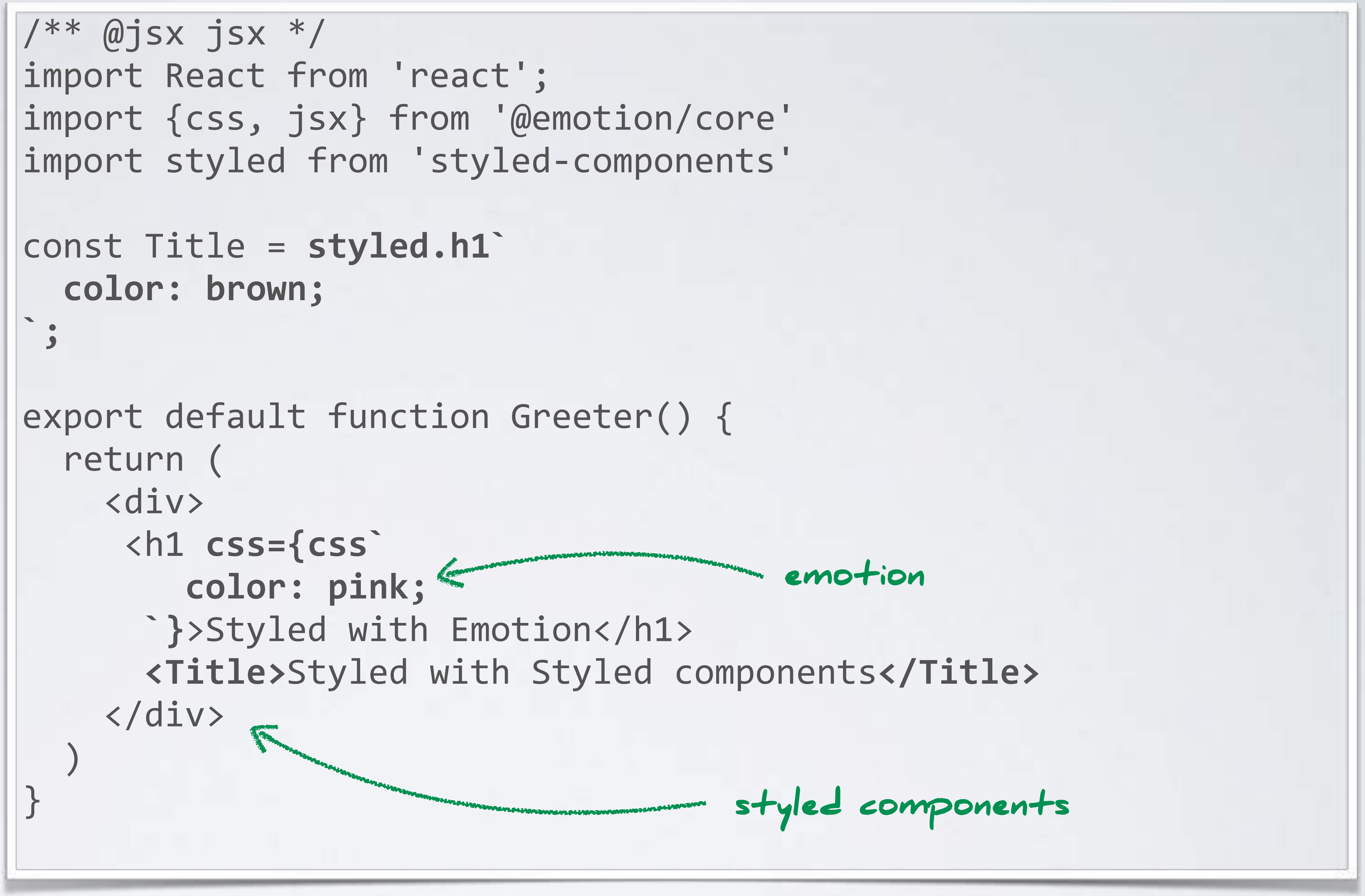


```
import { tss } from "tss-react/mui";
import Button from "@mui/material/Button";
import { useState } from "react";

type Props = {
    className?: string;
};

export function MyButton(props: Props) {
    const { className } = props;

    const [isClicked, setIsClicked] = useState(false);

    const { classes, cx } = useStyles({ color: isClicked ? "blue": "red" });

    //Thanks to cx, className will take priority over classes.root 🤩
    return (
        <Button
            className={cx(classes.root, className)}
            onClick={()=> setIsClicked(true)}
        >
            hello world
        </Button>
    );
}

const useStyles = tss
    .withParams<{ color: "red" | "blue"; }>()
    .create(({ theme, color })=> ({
        root: {
            // The color of the text is either blue or red depending of
            // the state fo the component.
            color,
            // When the curser is over the button has a black border
            "&:hover": {
                border: '4px solid black'
            },
            // On screens bigger than MD the button will have a big cyan border
            [theme.breakpoints.up("md")]: {
                border: '10px solid cyan'
            }
        }
    }));
```

CSS-in-JS Example

https://emotion.sh/

https://styled-components.com/



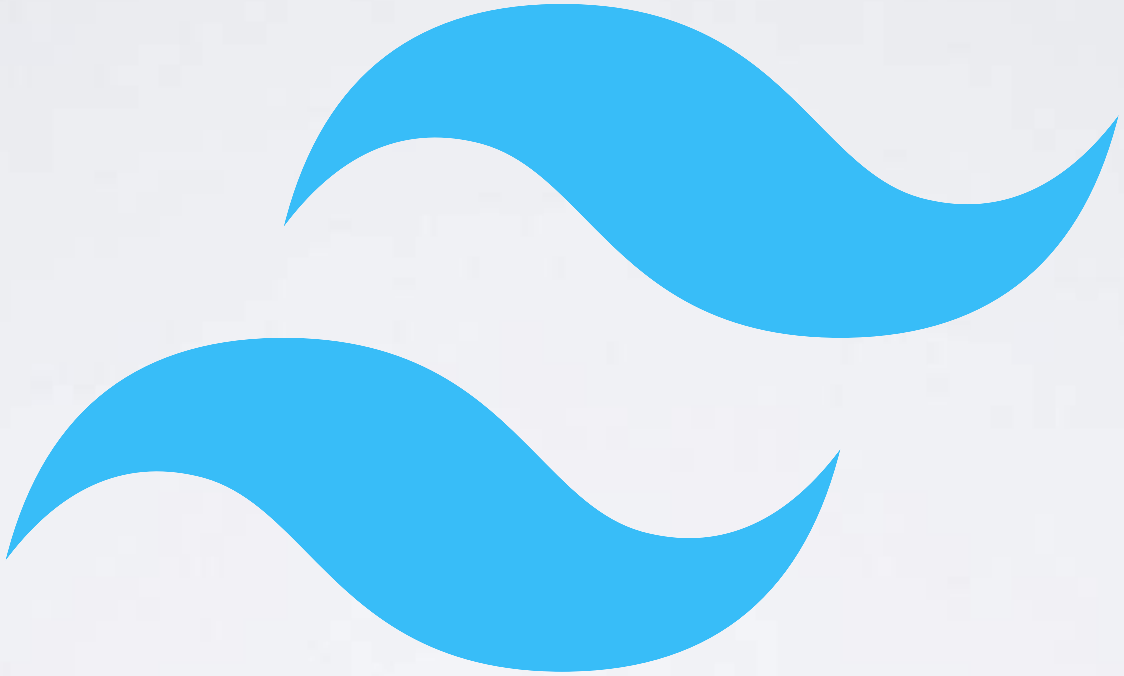
```
/** @jsx jsx */
import React from 'react';
import {css, jsx} from '@emotion/core'
import styled from 'styled-components'

const Title = styled.h1`
  color: brown;
`;

export default function Greeter() {
  return (
    <div>
     <h1 css={css`
        color: pink;
      `}>Styled with Emotion</h1>
      <Title>Styled with Styled components</Title>
    </div>
  )
}
```

emotion

styled components

Note: typically it does not make sense to use different styling libraries!

Love it or hate it:

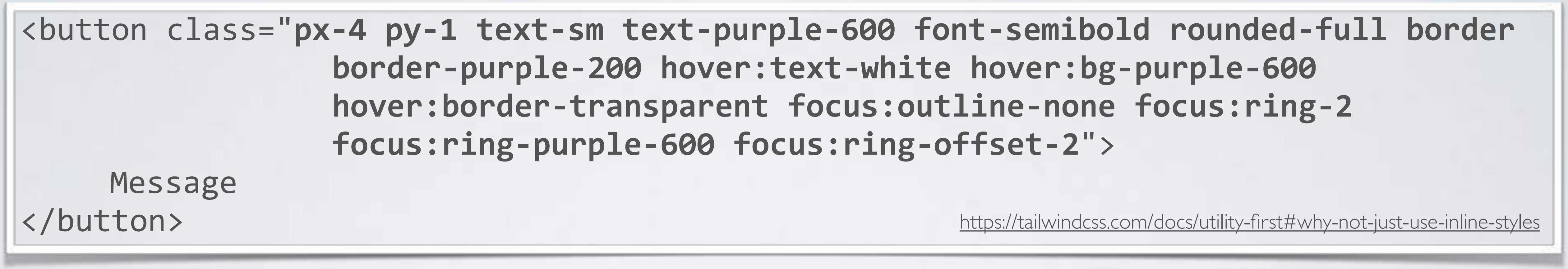


TailwindCSS

... but it might become a defacto-standard :-)

TailwindCSS

utility-first CSS framework



```
<button class="px-4 py-1 text-sm text-purple-600 font-semibold rounded-full border
                 border-purple-200 hover:text-white hover:bg-purple-600
                 hover:border-transparent focus:outline-none focus:ring-2
                 focus:ring-purple-600 focus:ring-offset-2">
     Message
</button>
```

https://tailwindcss.com/docs/utility-first#why-not-just-use-inline-styles

Framework agnostic but especially popular in React and other JSX-based frameworks:
Installation for any framework:
https://tailwindcss.com/docs/installation/framework-guides

Tailwind is traditionally strong for styling raw html elements. Typically it can't be used to style a traditional component library.
But the rise of headless component libraries open a new usage-scenario for Tailwind.

https://tailwindcss.com/

Tailwind is very controversial

Tailwind CSS is the worst:

https://www.youtube.com/watch?v=lHZwlzOUOZ4

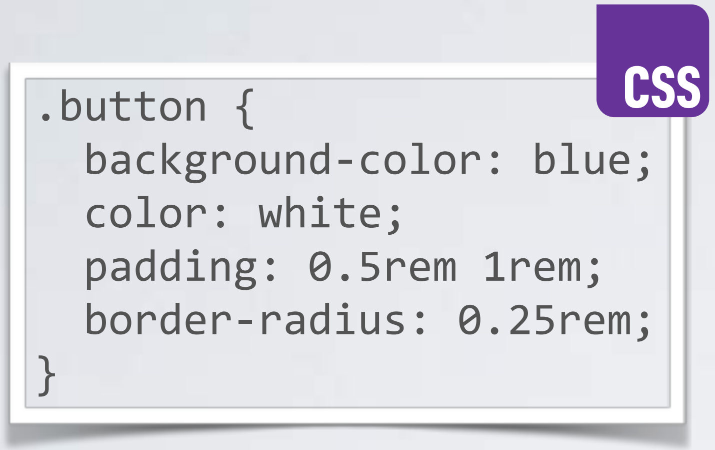
The Tailwind CSS Drama Your Users Don't Care About

https://www.builder.io/blog/the-tailwind-css-drama-your-users-don't-care-about

Why I don't like Tailwind:

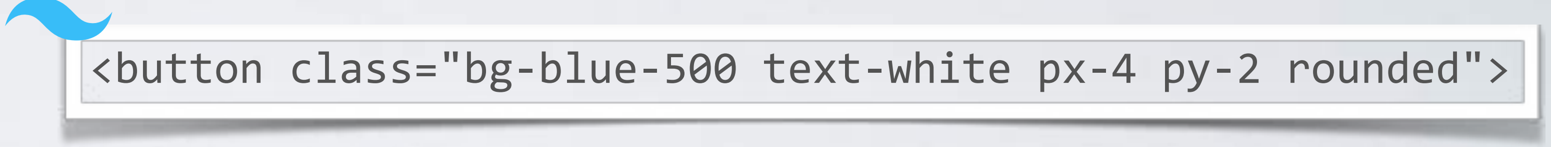
https://www.aleksandrhovhannisyan.com/blog/why-i-dont-like-tailwind-css/

Tailwind Conteroversy



```
.button {
  background-color: blue;
  color: white;
  padding: 0.5rem 1rem;
  border-radius: 0.25rem;
}
```

Vs

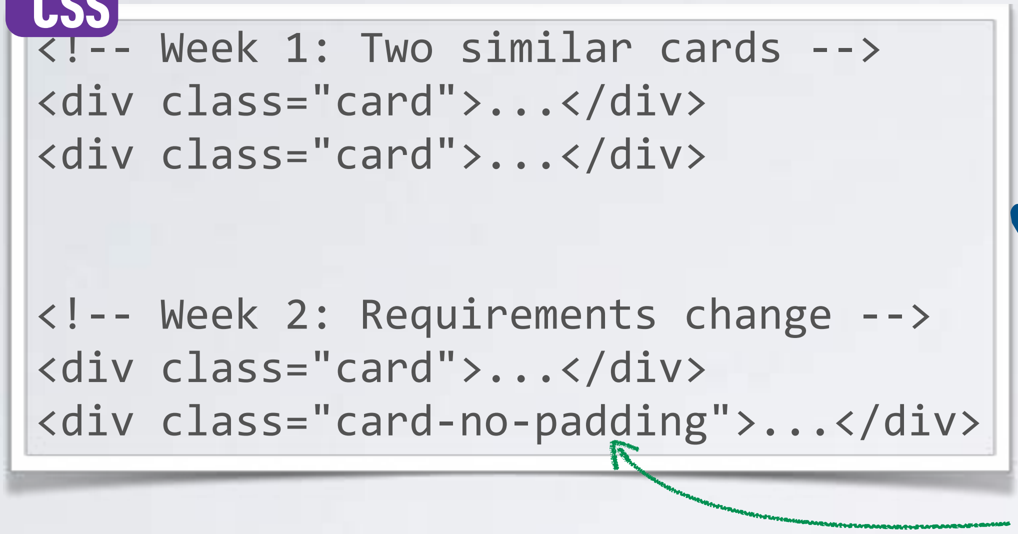


```
<button class="bg-blue-500 text-white px-4 py-2 rounded">
```

Tailwind arguments:
faster development speed, no CSS naming debates, easier refactoring (just delete the HTML, no orphaned CSS), and built-in design constraints that keep things consistent.

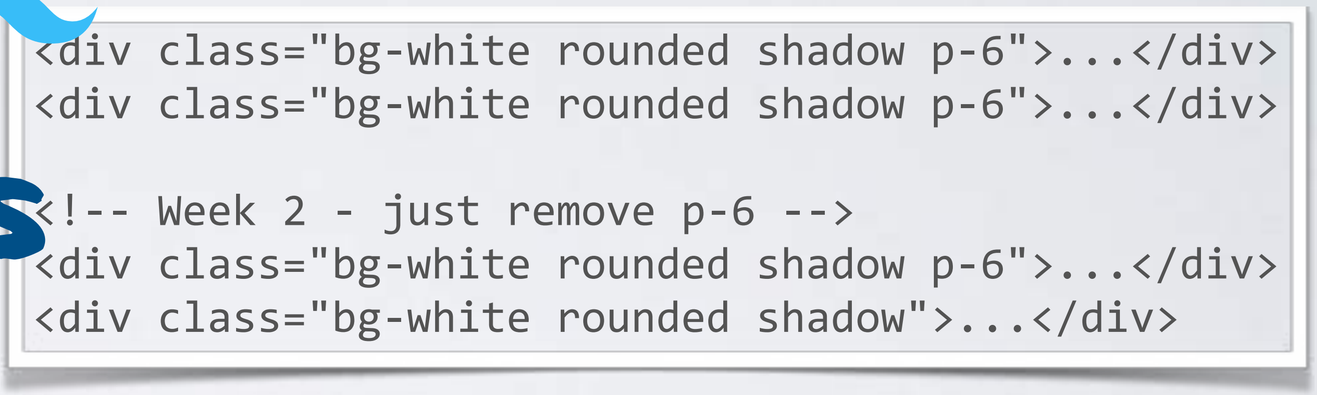
Critisim:
Duplication/Not DRY
Counter Argument:
-> in a "CSS-approach", the duplication is often in CSS
-> can be addressed with components

Tailwind Conteroversy



```
<!-- Week 1: Two similar cards -->
<div class="card">...</div>
<div class="card">...</div>


<!-- Week 2: Requirements change -->
<div class="card">...</div>
<div class="card-no-padding">...</div>
```

vs



```
<div class="bg-white rounded shadow p-6">...</div>
<div class="bg-white rounded shadow p-6">...</div>

<!-- Week 2 - just remove p-6 -->
<div class="bg-white rounded shadow p-6">...</div>
<div class="bg-white rounded shadow">...</div>
```

probably more duplication in css

The Real Question

The debate isn't really about duplication - it's about where complexity lives;

Traditional CSS:

Complexity in CSS (class names, specificity, cascading)
Simple HTML
Hidden relationships

vs

Tailwind:

Complexity in HTML (longer class lists)
Simple CSS (just utilities)
Explicit relationships

Component Libraries

Component Libraries

- Angular Material
  https://material.angular.io/
- Taiga UI
  https://taiga-ui.dev/
- Ng-Zorro / Ant Design
  https://ng.ant.design/docs/introduce/en
- PrimeNG
  https://www.primefaces.org/primeng/#/
- Kendo UI
  https://www.telerik.com/kendo-angular-ui
- Clarity Design System
  https://clarity.design/
- agGrid
  https://www.ag-grid.com/

- ng-bootstrap
  https://ng-bootstrap.github.io
- ngx-bootstrap
  https://valor-software.com/ngx-bootstrap
- Wijmo:
  https://www.grapecity.com/en/angular
- Infragistics:
  https://www.infragistics.com/products/ignite-ui-angular
- Syncfusion:
  https://www.syncfusion.com/angular-ui-components
- jQWidgets:
  https://www.jqwidgets.com/angular/angular-grid/
- More:
  https://angular.io/resources

Component Libraries

Material UI

https://mui.com/

- Mantine:
  https://mantine.dev/
- Chackra UI:
  https://chakra-ui.com/
- Ant Design of React
  https://ant.design/docs/react/introduce
- Semantic UI
  https://react.semantic-ui.com/

And more:
Rainbow UI, Cloudscape Design System, react-bootstrap, reactstrap ...

- KendoReact
  https://www.telerik.com/kendo-react-ui/
- PrimeReact
  https://www.primefaces.org/primereact/
- Infragistics / Ignite UI:
  https://www.infragistics.com/products/ignite-ui-react
- DevExtreme
  https://js.devexpress.com/
- Syncfusion:
  https://www.syncfusion.com/react-ui-components
- jQWidgets:
  https://www.jqwidgets.com/react/react-js-components.htm
- agGrid
  https://www.ag-grid.com/

https://github.com/brillout/awesome-react-components

Component Libraries

https://vuetifyjs.com/en/

QUASAR

https://quasar.dev/

Buefy

https://buefy.org/

PRIMEVUE

https://primefaces.org/primevue/

https://www.telerik.com/kendo-vue-ui

Design Systems

Design Systems

aka: "Widget Libraries"

Design Systems typically consists of pure presentation components, that should be widely useable in different applications.

Examples:

- SBB component library: https://angular.app.sbb.ch/
- AXA Patterns Library: https://axa-ch-webhub-cloud.github.io/plib-feature/develop

You have to decide what is the lowest common denominator that you will force on the consuming applications:

- framework/library
- framework/library versions

In the short-lived JavaScript ecosystem it might be a good idea to only rely on the web platform: html, javascript, css
WebComponents might be a good fit today.

Headless Components

Headless Components

Components with minimal or no UI.

Radix https://www.radix-ui.com/

React Aria https://react-spectrum.adobe.com/react-aria/index.html

headless UI https://headlessui.com/

base ui https://ark-ui.com/

https://base-ui.com/

Often combined with Tailwind: https://tailwindcss.com/

shadcn/ui https://ui.shadcn.com/

TanStack Table: https://tanstack.com/table

TanStack Form: https://tanstack.com/form

ReactRanger: https://github.com/tannerlinsley/react-ranger

Unstyled Components + Tailwind = shadcn

shadcn/ui https://ui.shadcn.com/

shadcn/vue: https://www.shadcn-vue.com/

"Code generator for components"
"Components are included as source code not as npm packages"

Web Components

WebComponents is a series of browser standards for creating reusable custom elements:

Shadow DOM	Encapsulation
HTML templates	markup that is not initially rendered and can be instantiated.
Custom elements	JavaScript API to define custom elements that can be used in html and their behavior

Custom Elements API:

JavaScript

```
customElements.define('my-component',  class extends HTMLElement { ... })
```

html:

```
<div> <my-component></my-component> <div>
```

Framework integrations:

- Angular elements
- Vue & React wrappers

Supported in all modern browsers today.

Framework support for Web Components:
https://custom-elements-everywhere.com/

https://developer.mozilla.org/en-US/docs/Web/Web_Components
https://developer.mozilla.org/en-US/docs/Web/API/Window/customElements

WebComponent Frameworks

stencil https://stenciljs.com/

Lit https://lit.dev/

Support for Web Components in Frameworks

https://custom-elements-everywhere.com/

WebComponent Libraries

Fast (Microsoft)	https://www.fast.design/
Spectrum Web Components (Adobe)	https://opensource.adobe.com/spectrum-web-components/
Lightning Web Components (Salesforce)	https://developer.salesforce.com/docs/platform/lwc/guide
Shoelace	https://shoelace.style/
Vaadin Web Components	https://github.com/vaadin/web-components

Directory: https://www.webcomponents.org/

Web Components Out of the Box are missing

- Reactivity & State Management
- Rich Templating & Data Binding
- Powerful composability
- Developer Experience
- Server-Side Rendering