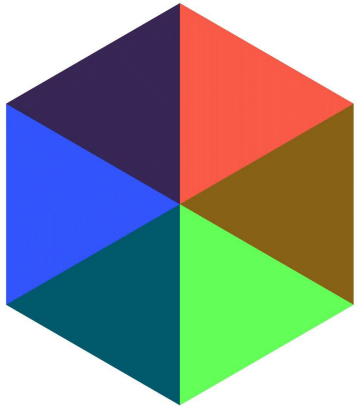
Build a better UI component library with Styled System

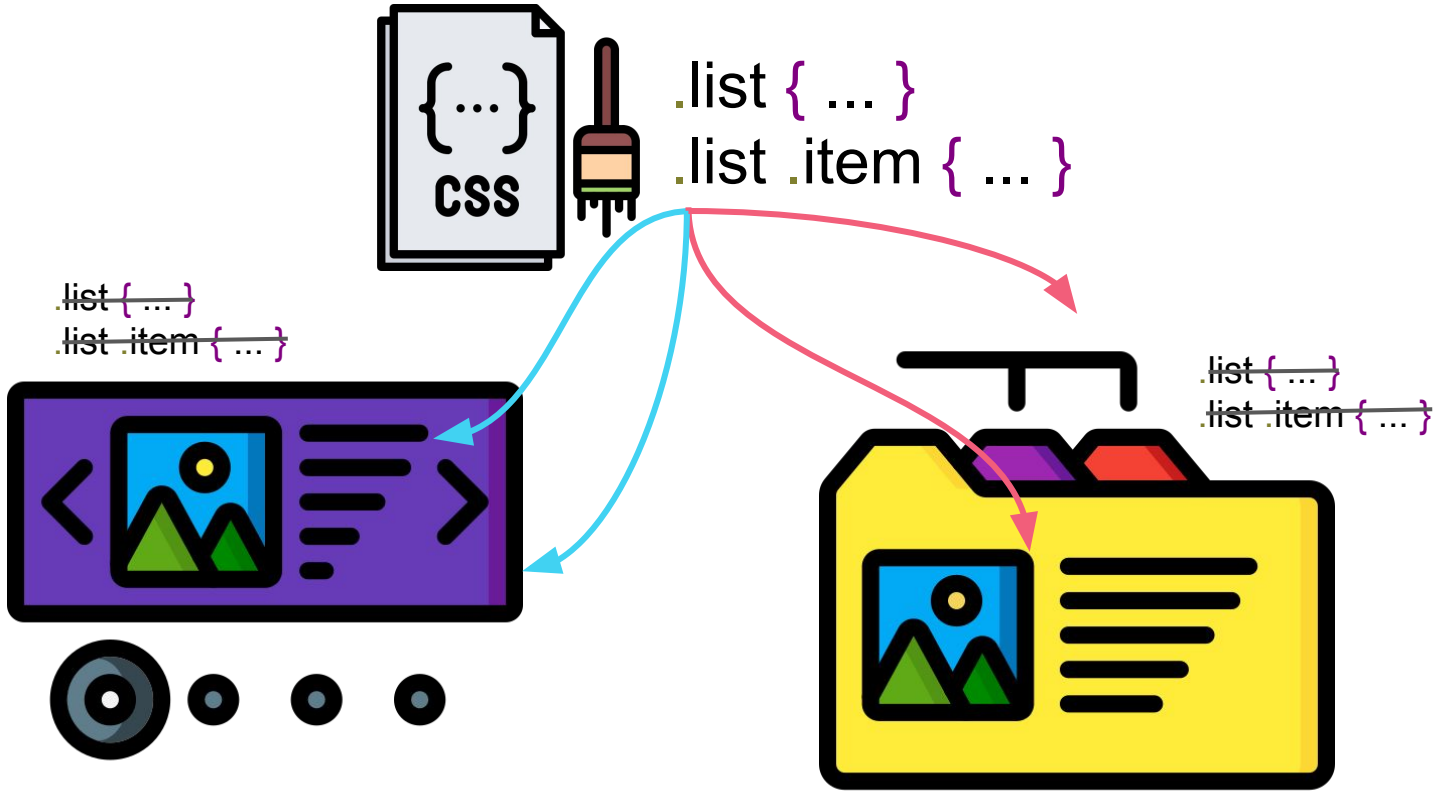


Agenda

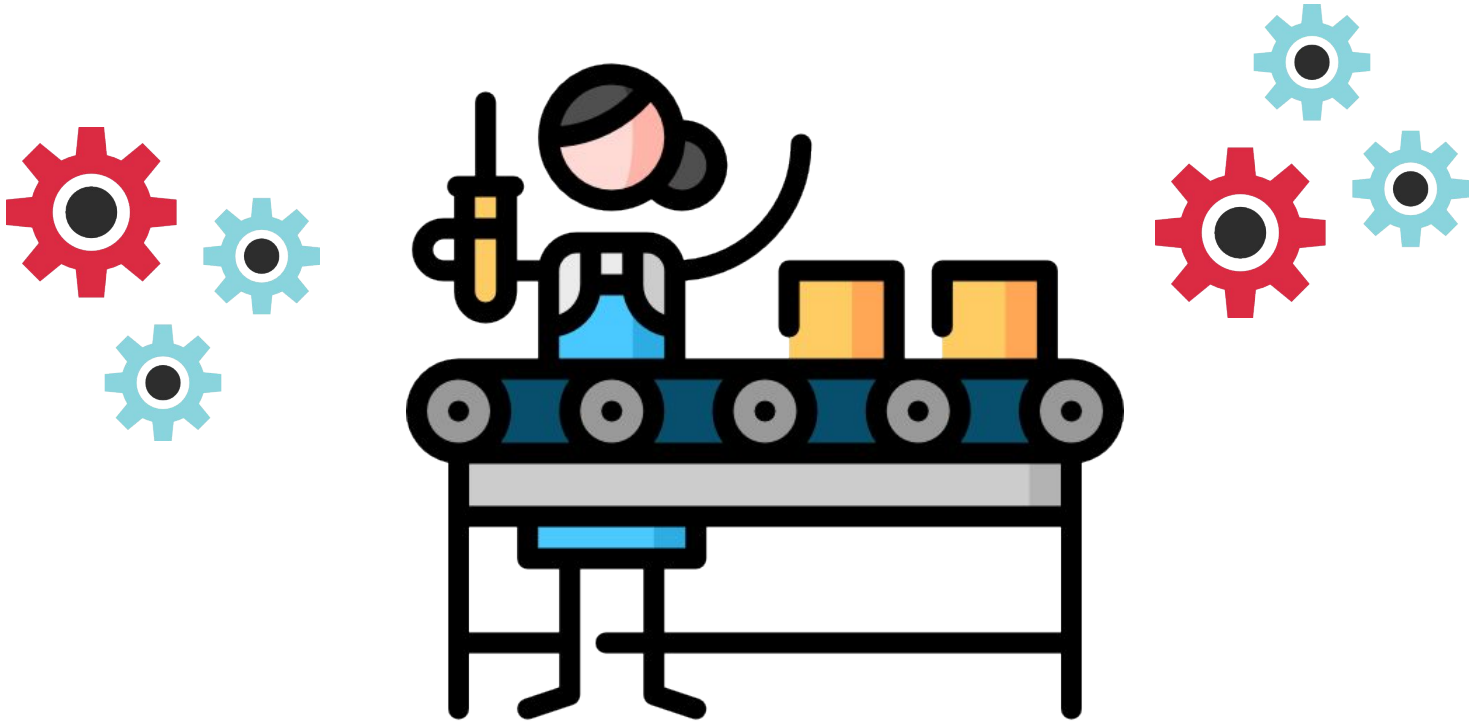
- CSS methodologies
- Styled System

Everything in CSS is global.

Name collisions



Reusability



CSS methodologies

CSS methodologies are the solution.

- OOCSS (Object-Oriented CSS)
- BEM (Block, Element, Modifier)
- SMACSS (Scalable and Modular Architecture for CSS)
- CSS Modules
- CSS in JS

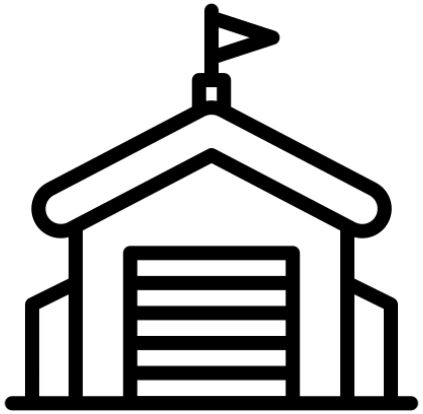
OOCSS (Object-Oriented CSS)

In OOCSS, style rules are written exclusively using CSS class selectors.

Rules

- Separation of structure from skin (結構與樣式分離)
- Separation of container and content (內容與容器分離)

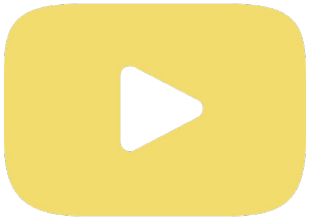
Separation of structure from skin



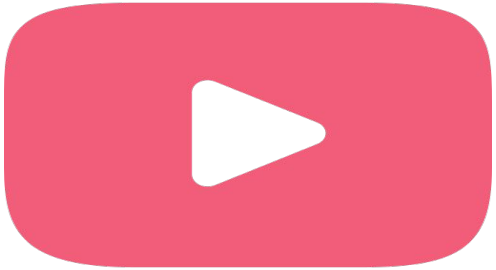
margin, padding, display, position, vertical-align, width, height

color, background, opacity, font-size

Example (1/3)



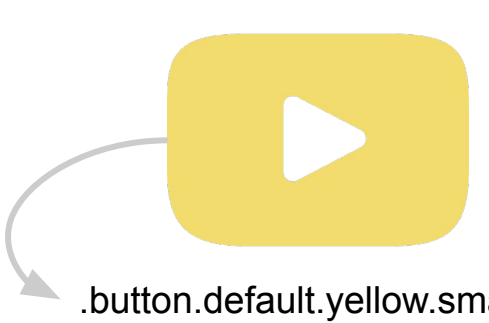
```
.button-yellow {
  width: 100px;
  height: 50px;
  margin: 10px;
  padding: 10px;
  border-radius: 5px;
  background: #f2dc6d;
  border: 1px solid #fefefe;
  color: #ccc;
}
```



```
.button-pink {
  width: 200px;
  height: 100px;
  margin: 10px;
  padding: 10px;
  border-radius: 5px;
  background: #f25e7a;
  border: 1px solid #fefefe;
  color: #ccc;
}
```

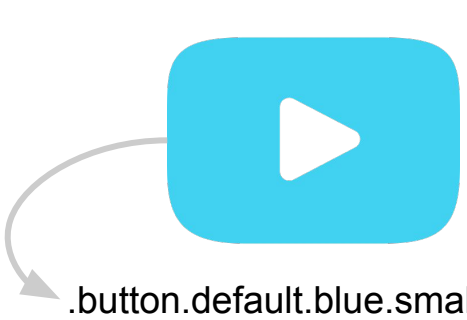
```
.button-blue {
  width: 100px;
  height: 50px;
  margin: 10px;
  padding: 10px;
  border-radius: 5px;
  background: #41d2f2;
  border: 1px solid #fefefe;
  color: #ccc;
}
```

Example (2/3)



.button.default.yellow.small

.button.default.pink.large



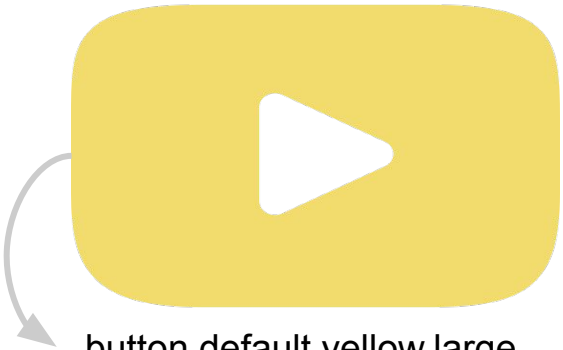
.button.default.blue.small

```
.button {
  margin: 10px;
  padding: 10px;
}
```

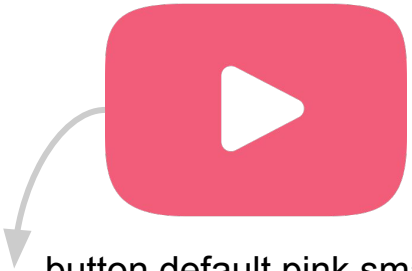
```
.button.default {
  border-radius: 5px;
  border: 1px solid #fefefe;
  color: #ccc;
}
```

```
.button.yellow { background: #f2dc6d; }
.button.pink { background: #f25e7a; }
.button.blue { background: #41d2f2; }
.button.green { background: #28efb7; }

.button.small { width: 100px; height: 50px; }
.button.large { width: 200px; height: 100px;}
```

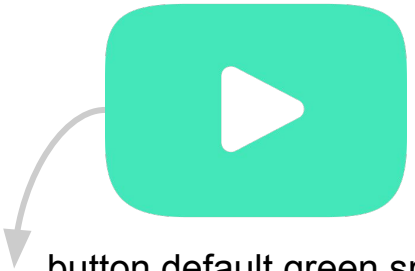
Example (3/3)



.button.default.yellow.large

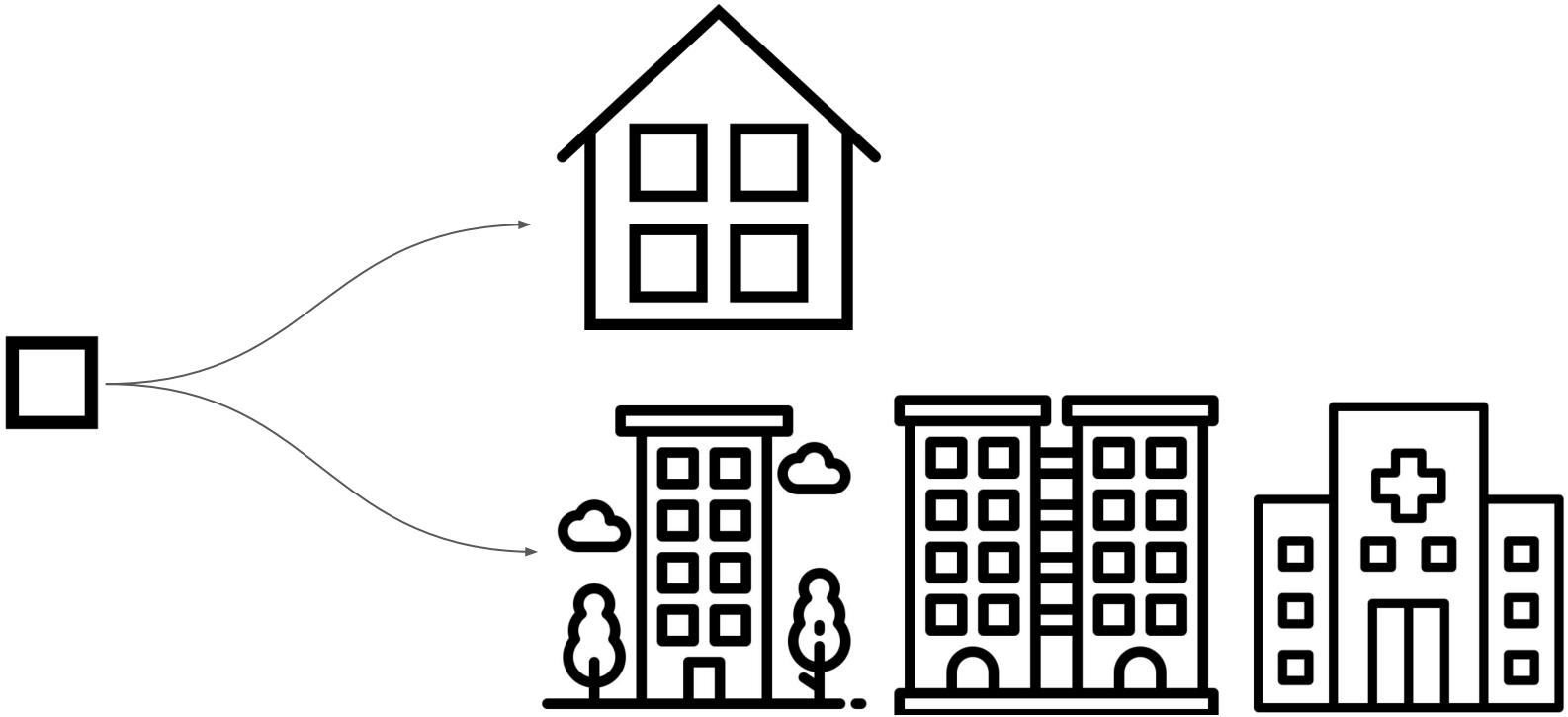


.button.default.pink.small

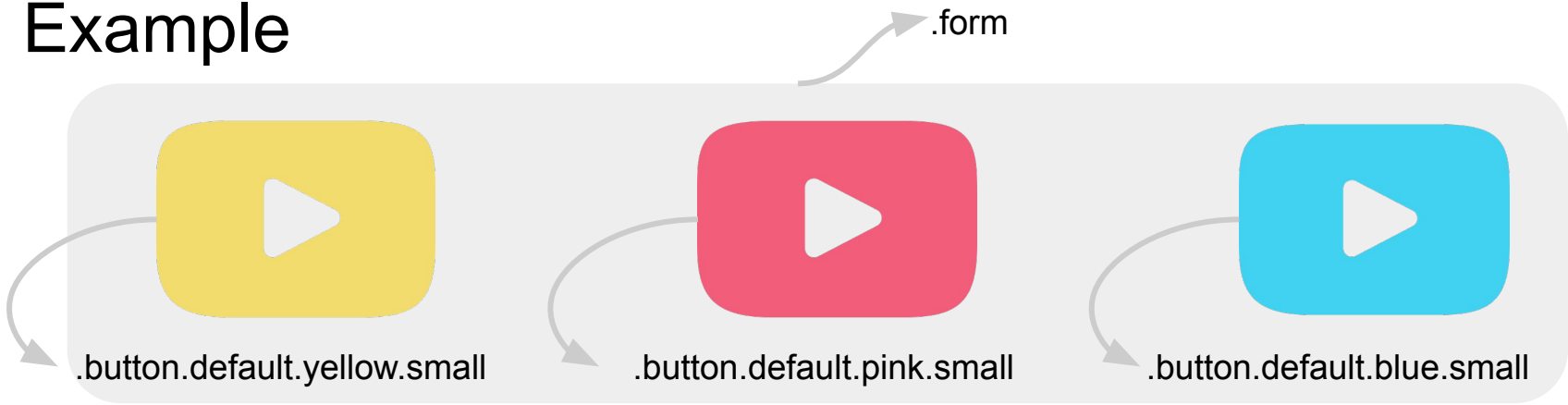


.button.default.green.small

Separation of container and content



Example

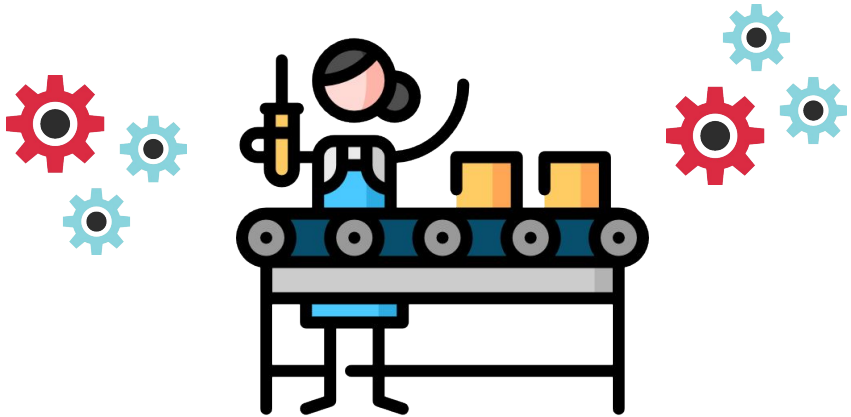



```
.form { ... }
.form .button { ... }
.form .button.yellow { ... }
.form .button.pink { ... }
.form .button.blue { ... }
```

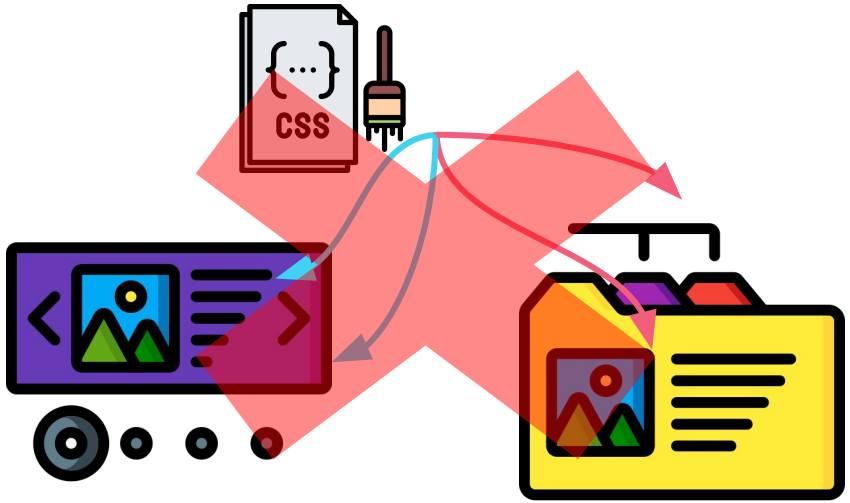
More flexible, reusable!

```
.form { ... }
.list { ... }
.button { ... }
.button.yellow { ... }
.button.pink { ... }
.button.blue { ... }
```

Pros and cons



Reusability



Name collision

BEM (Block, Element, Modifier)

Block
A block component

Element
A component of a block

Modifier
State for a block or element.

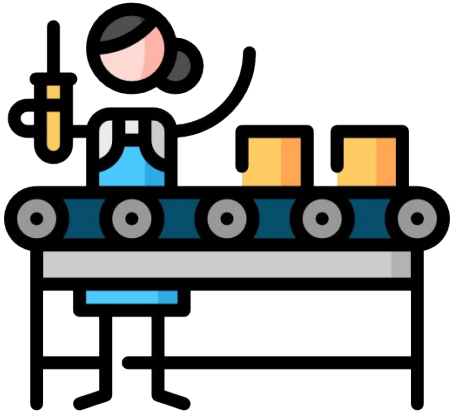
Example

Block
.card-list

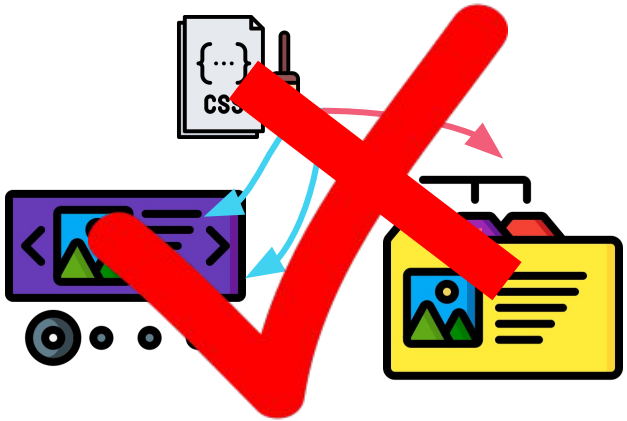
Element
.card-list__item

Modifier
.card-list__item--highlight

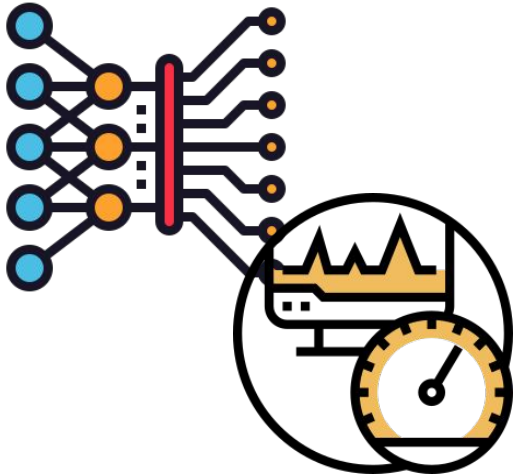
Pros and cons



Reusability



Name collision



Selector nesting for css specificity and rendering performance

SMACSS (Scalable and Modular Architecture for CSS)

Layout
structural layout

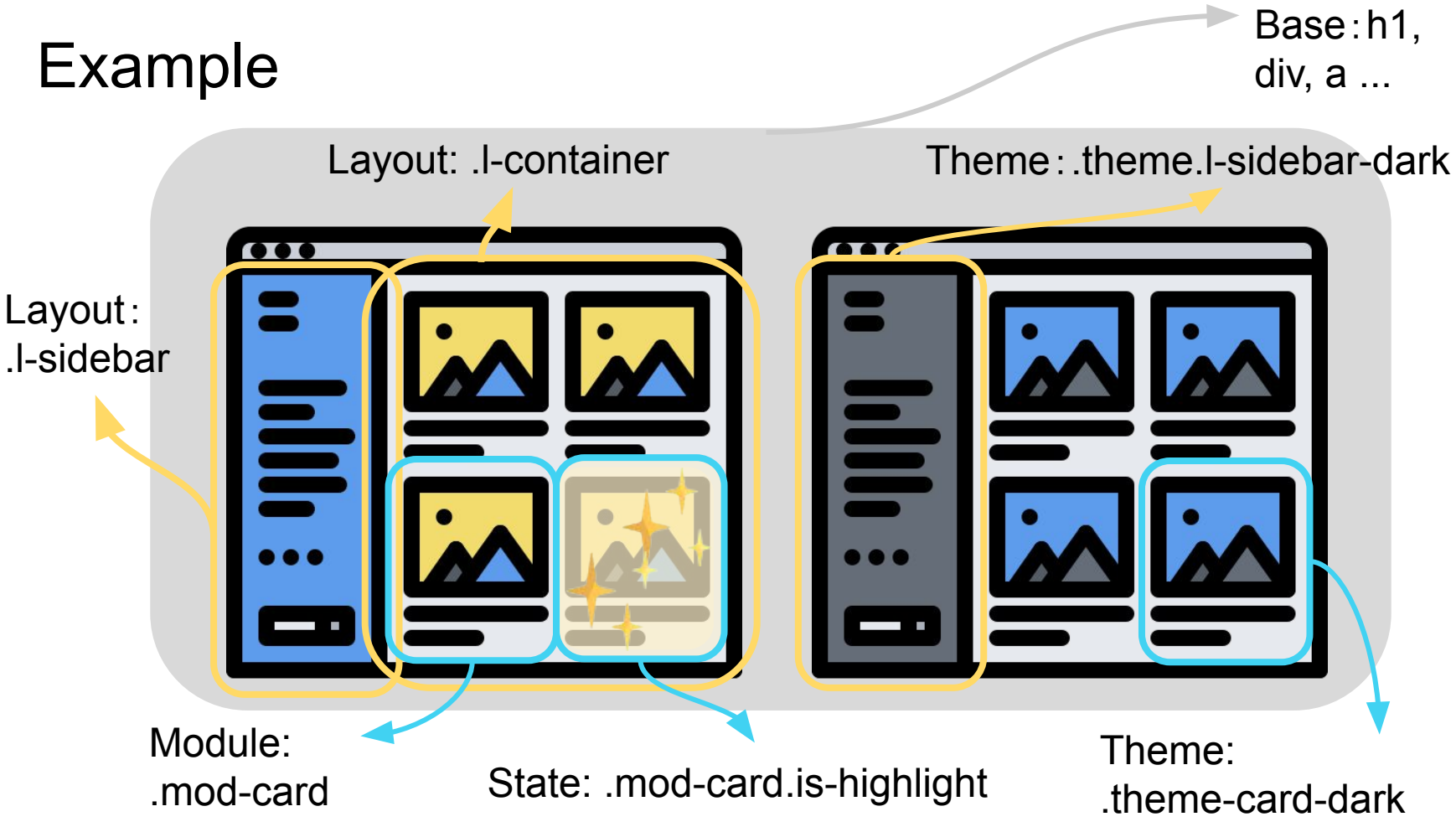
Base
set the default CSS properties for the whole site

Module
modular, reusable components

State
specify the current state of something in the interface

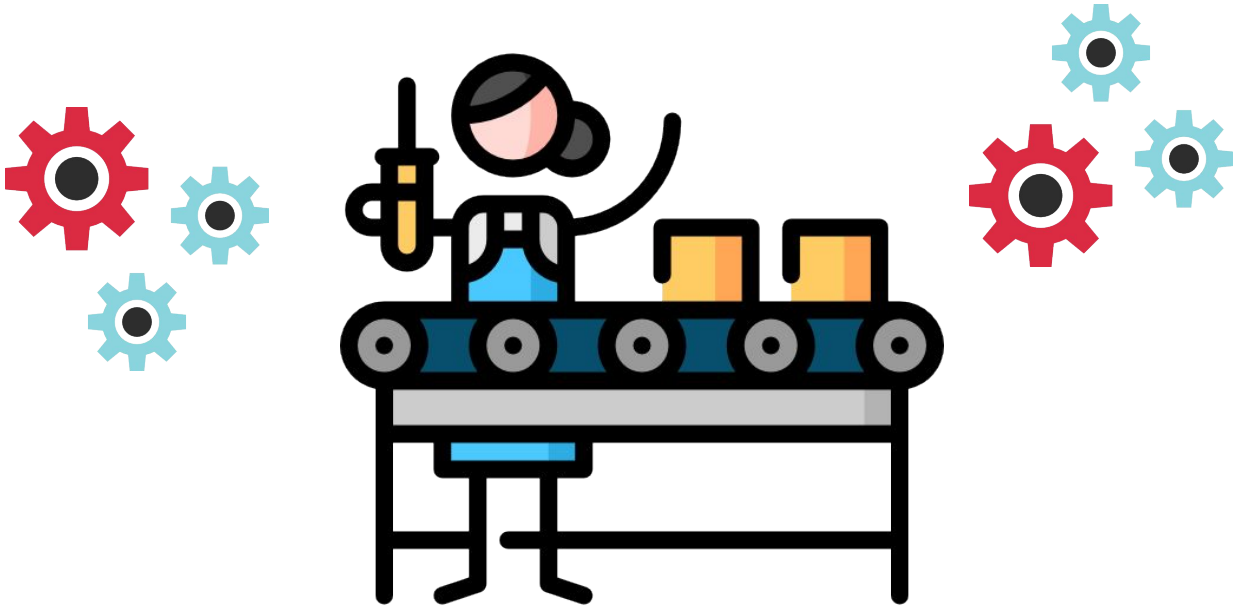
Theme
affect layouts and modules, triggered by user events

Example


Base : h1,
div, a ...
Layout: .l-container
Theme : .theme.l-sidebar-dark
Layout :
.l-sidebar
Module:
.mod-card
State: .mod-card.is-highlight
Theme:
.theme-card-dark

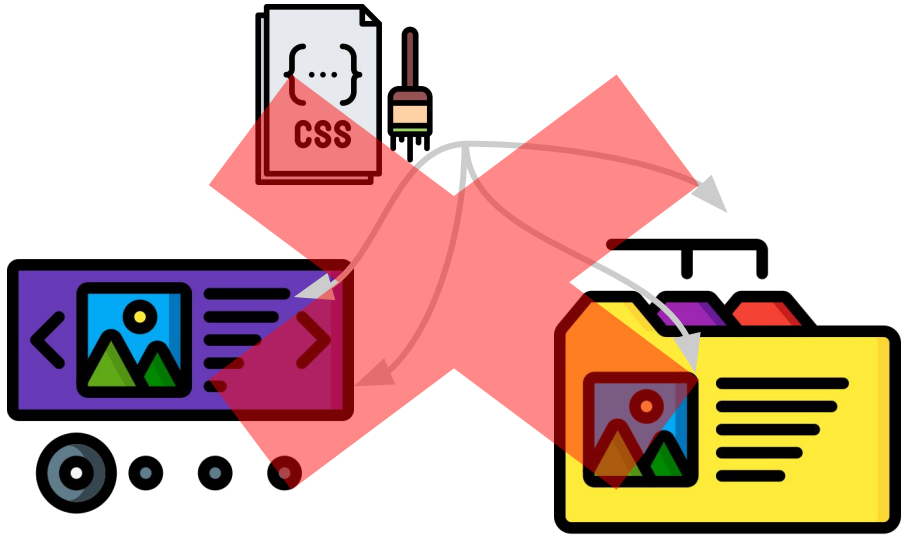
What problems do OOCSS, BEM and SMACSS solve?

DRY stylesheets, more flexible, modular, reusable!



Global scope is still a problem!

Name collisions still happen in the global scope.



CSS Modules

Scope CSS rules locally by using tools to transform the class name with a hash.

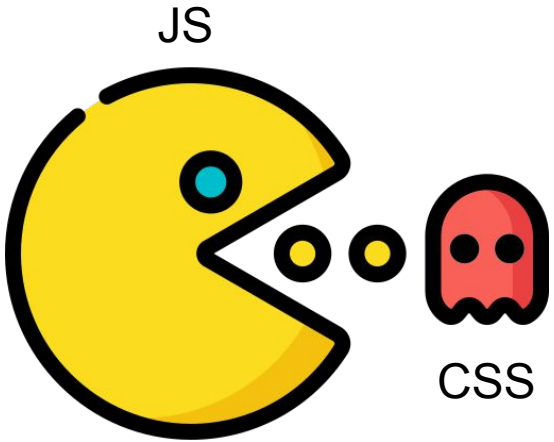
.button.default.blue.small

.button { … }
.default { … }
.small { ... }
.blue { ... }

View in browser inspector

._3zyde4l1yATCOkgn-DBWEL.ucdIPRB M8dj46yVBF3bcu._1Jf-igm_Q7n33cjbU1 HWU.UwRmwF8HGfUEMqBxpndWg

._3zyde4l1yATCOkgn-DBW { … }
.ucdIPRBM8dj46yVBF3bcu { … }
._1Jf-igm_Q7n33cjbU1HWU { ... }
.UwRmwF8HGfUEMqBxpndWg { ... }



CSS in JS

Everything is constructed by using components.
Write CSS in components.

Styled Components 

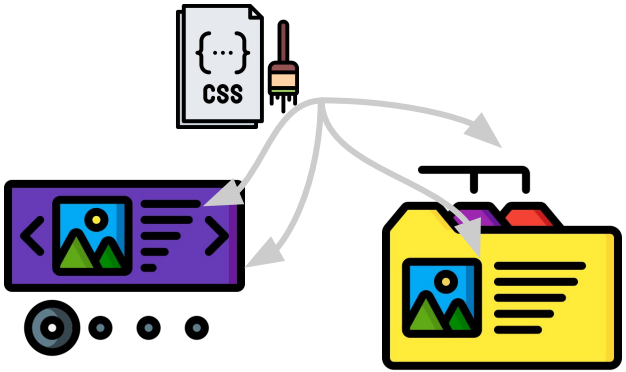
JSX

```
<Button />
```

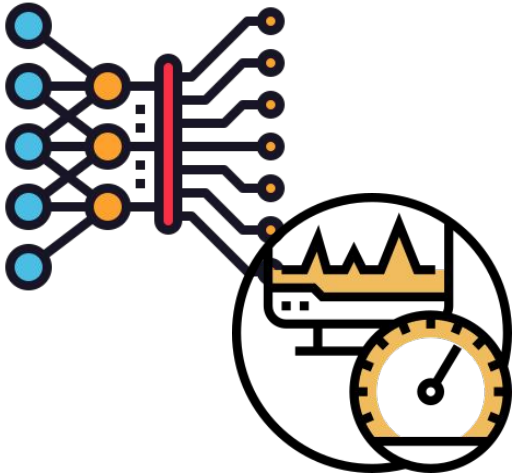
CSS in JS

```
import styled from 'styled-components';

const Button = styled.button`
  margin: 0 10px;
  padding: 10px;
  background: #fefefe;
  border-radius: 3px;
  border: 1px solid #ccc;
  color: #525252;
`
```

What problems does CSS in JS solve?



Name collision

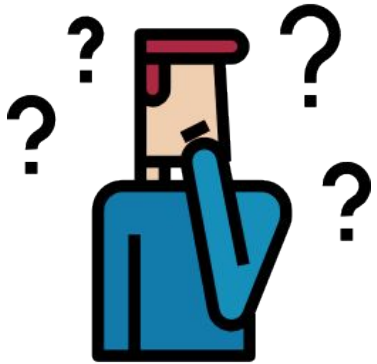


Selector nesting for css specificity and rendering performance

Refactoring

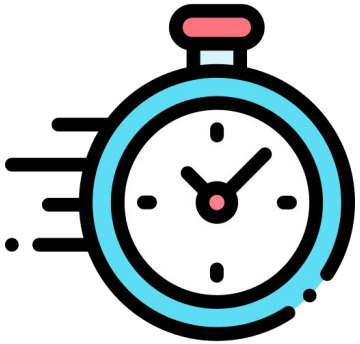
Still have some problems...

- Components with **inconsistent props**.
- How to efficiently implement **multiple themes**? Like SMASCC…
- How to efficiently implement **responsive** styles for different devices?
- How to **reduce redundant codes** for setting media queries or mapping props to css properties?



Build a better component library with **Styled System**

Responding to Change



Respond to changing requirements quickly by using **utility functions**

Utility functions (1/2)


```
<Box color='#000 bg='tomato' />
```

```
const Box = styled.div`
  margin: 15px 0;
  padding: 15px;
  color: ${(props) => props.color};
  background: ${(props) => props.bg};
  border-radius: 10px;
`;
```

```
const getStyles = ({ color, bg }) => ({
  color,
  background: bg,
});

const Box = styled.div`
  ${getColor};
  margin: 15px 0;
  padding: 15px;
  border-radius: 10px;
`;
```

Utility functions (2/2)

```
const getStyles = ({ color, bg }) => ({
  color,
  background: bg,
});

const Box = styled.div`
  ${getColor};
  margin: 15px 0;
  padding: 15px;
  border-radius: 10px;
`;
```

Styled System


```
import { color } from 'styled-system';

const Box = styled.div`
  ${color}
  margin: 15px 0;
  padding: 15px;
  border-radius: 10px;
`;
```

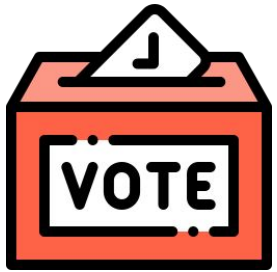
Consistency

- Style consistently with a global theme
- Components with inconsistent props

Theming

Utilizing `<ThemeProvider>` and pass the theme object from root node to provide global theme.

```
<ThemeProvider theme={theme}>
  <Box color='black' bg='tomato'/>
</ThemeProvider>
```



```
const theme = {
  color: {
    black: '#333',
  },
  bg: {
    tomato: 'tomato',
  },
};
```

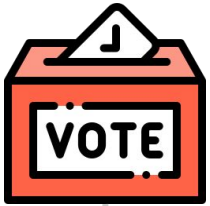
Define component styles in theme object

```
<Button variant="danger" size="large" />


const buttonStyle = variant({ key:
'buttons'});
const buttonSizeStyle = variant({
prop: 'size', key: 'buttons.size' });

const Button = styled.div`
  ${buttonStyle}
  ${buttonSizeStyle}
  padding: 15px;
`;
```

```
const theme =  {
  buttons: {
    danger: {
      color: 'white',
      background: '#f25e7a'
    },
    size: {
      default: { height: 50 },
      large: { height: 100 }
    }
  }
};
```

Variants

Utilizing variants to define component styles.




```
<Box variant='primary' />
```


```
<Box variant='secondary'/>
```

```
import { variant } from 'styled-system';

const Box = styled('div')(
  variant({
    variants: {
      primary: { color: 'black', bg: 'tomato' },
      secondary: { color: 'black', bg: 'yellow' },
    },
  }),
);
```

Inconsistent props

```
<Button color='black'>Click</Button>
```

Use color utility function in Styled System

```
<Label fontColor='white'>$</Label>
```

```
<Button color='black'>Click</Button>
```

```
<Label color='white'>$</Label>
```

Mobile-First

Create mobile-first responsive layouts with ease by using **array syntax**

Responsive styles

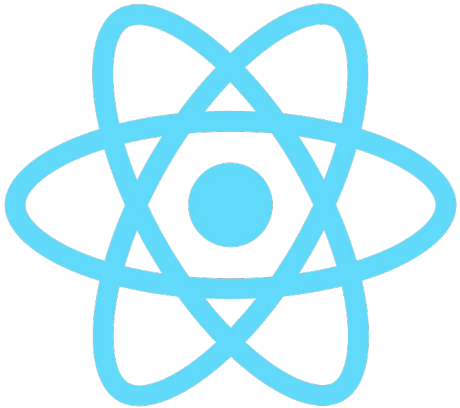
Create mobile-first responsive layouts with ease by using an array syntax.

```
.thing {
  font-size: 16px;
  width: 100%;
}

@media screen and (min-width: 40em) {
  font-size: 20px;
  width: 50%;
}

@media screen and (min-width: 52em) {
  font-size: 24px;
}
```

Styled System

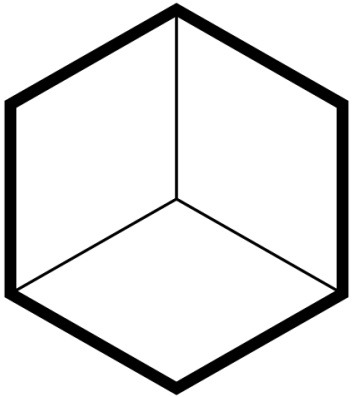

```
<Thing
  fontSize={[ 16, 20, 24 ]}
  width={[1, 1/2]}
/>
```

Best Solution 



React.js

Styled
Components



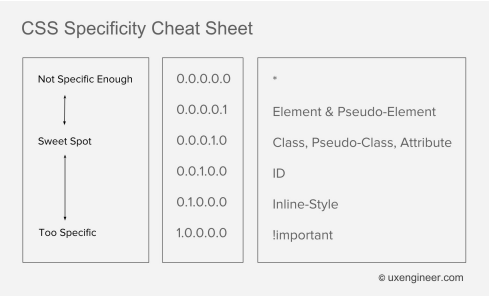
Styled
System


Q
A

Traditional CSS rules feat. Styled Components

#root **div** { color: red; } /* in site.css, score: 100 + 1 = 101 */

.jqouBD { color: black; } /* in styled component, score: 10 */



source

#root **div** { color: red; } /* in site.css, score: 100 + 1 = 101 */

.jqouBD { color: black **!important**; } /* in styled component, score: 10000 */

Class name without a hash for end-to-end testing

Use a chaninable method “attrs” to attach props to the styled component.

```
<Button>Click me!</Button>
```

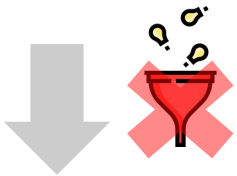
```
const Button = styled.button.attrs({ className: 'button-submit' })`...`;
```


View in browser inspector

```
<button class="sc-EHOje button-submit">Click me!</button>
```

Do not filter out results when passing props to child components

```
<WrapperComponent color="blue" />

const WrapperComponent = styled(InnerComponent)`...`;

const InnerComponent = props => <div {...props}>Inner</div>;
```



```
<div color="blue" class="sc-TFwJa HJEpQ">Inner</div>
```

Demo

Simple Example

http://bit.ly/35x16cL

References 

- Styled System https://styled-system.com/
- We need a better UI component library - Styled System http://bit.ly/2roJEsq
- The Three Tenets of Styled System http://bit.ly/35ygrcV
- Styled System Example http://bit.ly/35x16cL
- Styled System: Pseudo selectors in Variant http://bit.ly/35yqGhy
- How to create responsive UI with styled-components http://bit.ly/34lhxJ3
- CSS 實戰心法 http://bit.ly/2shXUn4