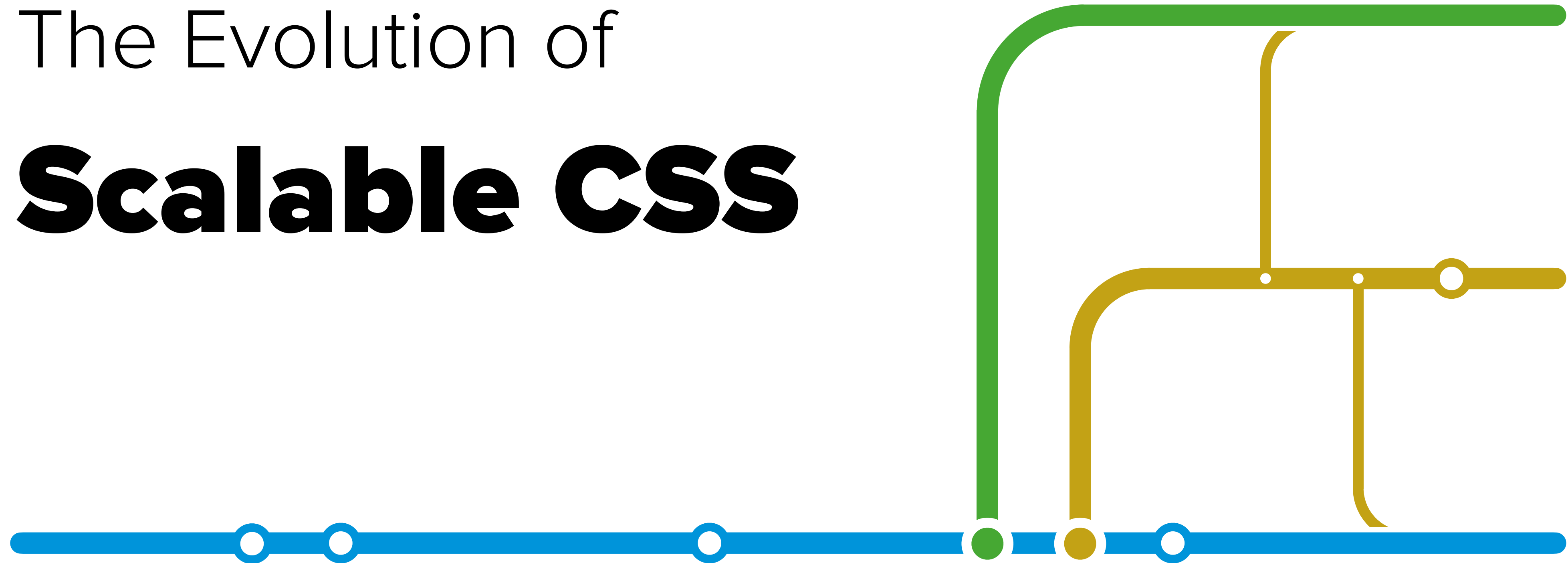



# The Evolution of **Scalable CSS**






*We have to know the **past**  
to understand the **present***

*- Carl Sagan*

A well-researched **chronicle**,  
describing how CSS **tools** and  
**techniques** have evolved over time.



A **chronicle** is a recording  
of **significant historical events**  
in the order of their occurrence,  
as seen from the **chronicler's perspective**.

- Wikipedia



# ANDREI PFEIFFER

Timișoara / RO



*Code Designer*



*Co-Organizer*



*andreipfeiffer.dev*

**<sup>s</sup>CALABLE**

**css**

**SCALABLE** **CSS**

The most problematic

# CSS Scalability Issues

- 💥 **Naming collisions**, due to global namespace
- ⚔️ **Specificity wars**, when overriding styles
- 🧟 **Zombie code**, non trivial to remove unused code





*It's time to talk about the elephant in the corner  
of the room: **stylesheet maintainability.***

*- 2005, Simon Willinson*

PART 1/7

# CSS Good Practices



# Avoid ID selectors

//  avoid using ID selectors in CSS

```
#contact-form {}
```

//  prefer classes instead

```
.contact-form {}
```

# Keep specificity low

//  avoid overly specific selectors

```
.main_menu ul li.item a.link {}
```

//  prefer low specificity




```
.main_menu .link {}
```

# Avoid type selectors

//  avoid type selectors as descendants  
**.card strong {}**

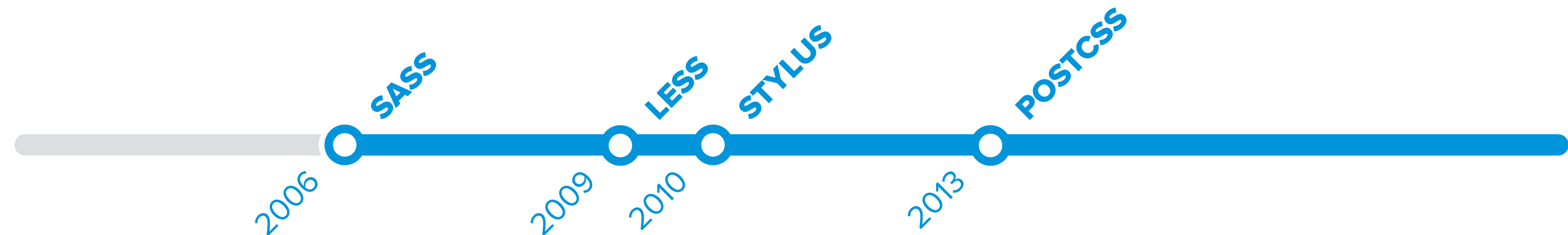
//  prefer classes for more flexibility  
**.card .keyword {}**

# Practices don't really scale

-  No official comprehensive guide
-  Cumbersome to learn and teach
-  Scaling maintenance effort

PART 2/7

# CSS Processors



# Contextual styles

Nesting + Parent selector

// 🙌 single definition

```
.title {  
  &:hover {}  
  &::after {}  
}
```

// 🙌 verbose duplication

```
.title {}  
.title:hover {}  
.title::after {}
```



# CSS Nesting Module

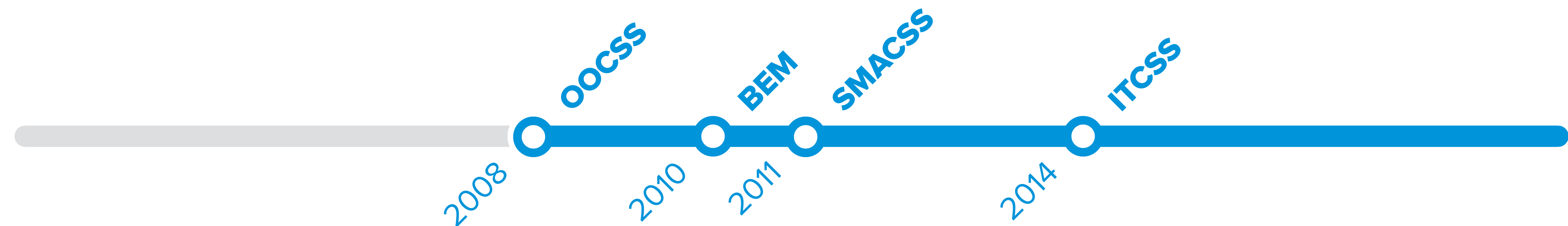
W3C First Public Working Draft, 31 August 2021

**Editors:** Tab Atkins-Bittner, Adam Argyle

<https://www.w3.org/TR/css-nesting-1/>

PART 3/7

# CSS Methodologies

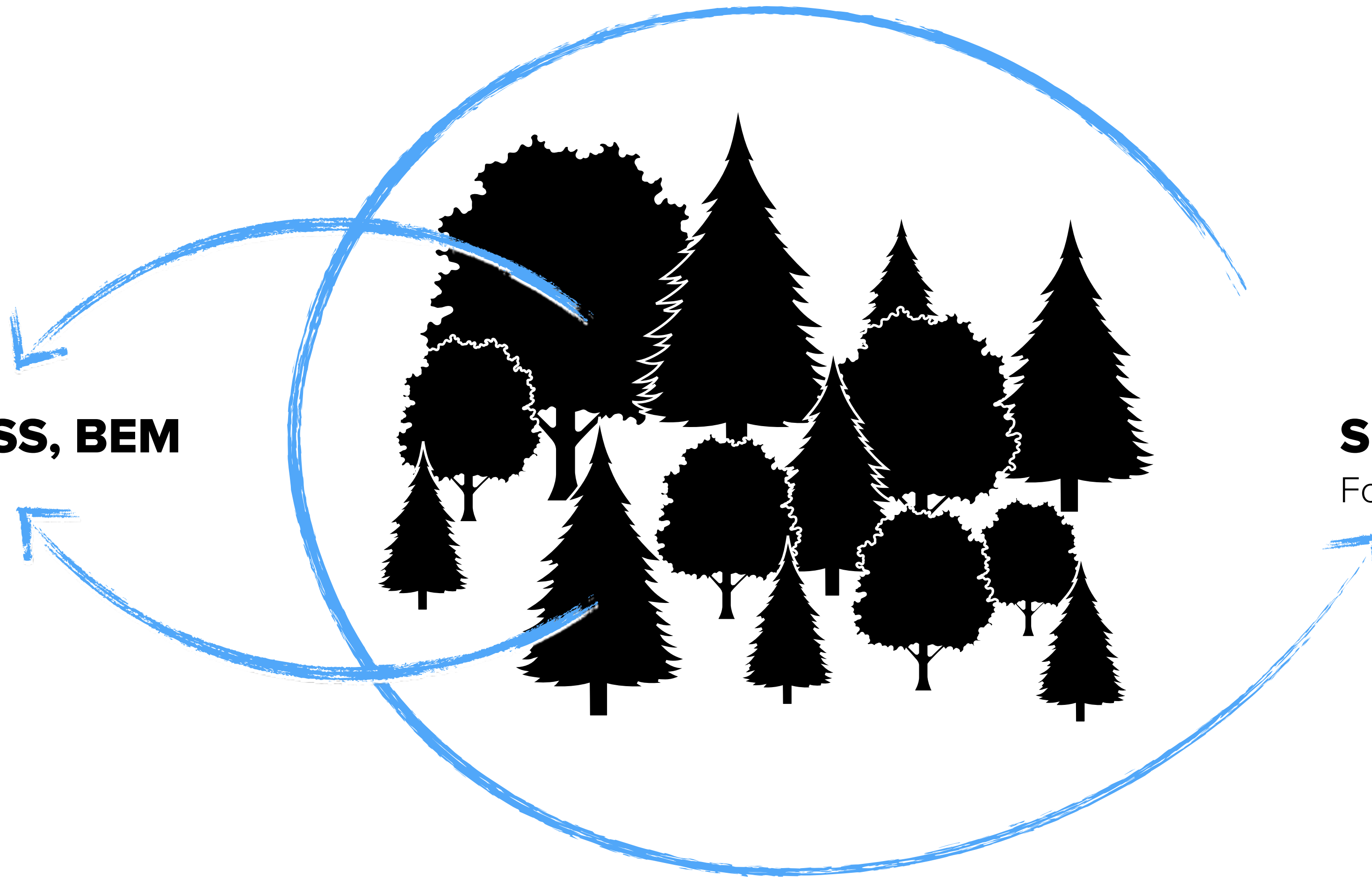


# Proto Components

- **OOCSS:** identify reusable & repeating visual patterns
- **BEM:** defines naming conventions for CSS selectors

**OOCSS, BEM**  
Trees


**SMACSS, ITCSS**  
Forest



# High-level Architecture

SMACSS, ITCSS

- **Base rules:** CSS resets, typography
- **Layout styles:** Grid system, Boxes, Sidebars
- **Application components:** abstract & domain specific
- **Utilities:** reusable helper classes for overrides



*[...] authors are encouraged to use values that describe **the nature of the content**, rather than values that describe the desired presentation.*

*- HTML Living Standard*

# Semantic CSS

//  **Semantic** (conveys meaning)

```
<nav class="main-menu"></nav>
```

//  **Non-Semantic** (conveys implementation)

```
<nav class="flex bg-dark pad-small align-center"></nav>
```

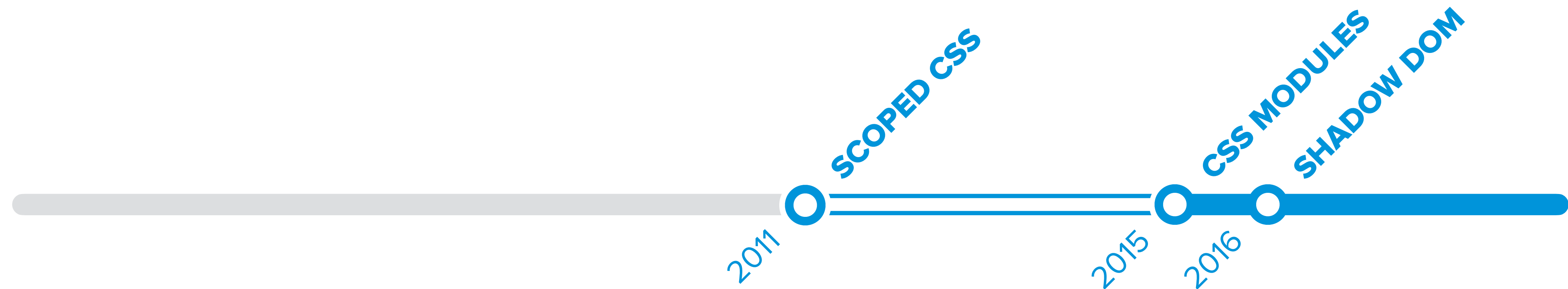
# Semantic CSS limitations

- Code repetition: `display: flex`, `font-weight: bold`, etc
- Ever growing stylesheets
- **Naming things is inherently difficult**



PART 4/7

# Styles Encapsulation





***Shadow DOM fixes CSS and DOM. It introduces  
scoped styles to the web platform, **without tools  
or naming conventions.*****

- 2016, Eric Bidelman, [web.dev](https://web.dev/)

# The End of Global CSS

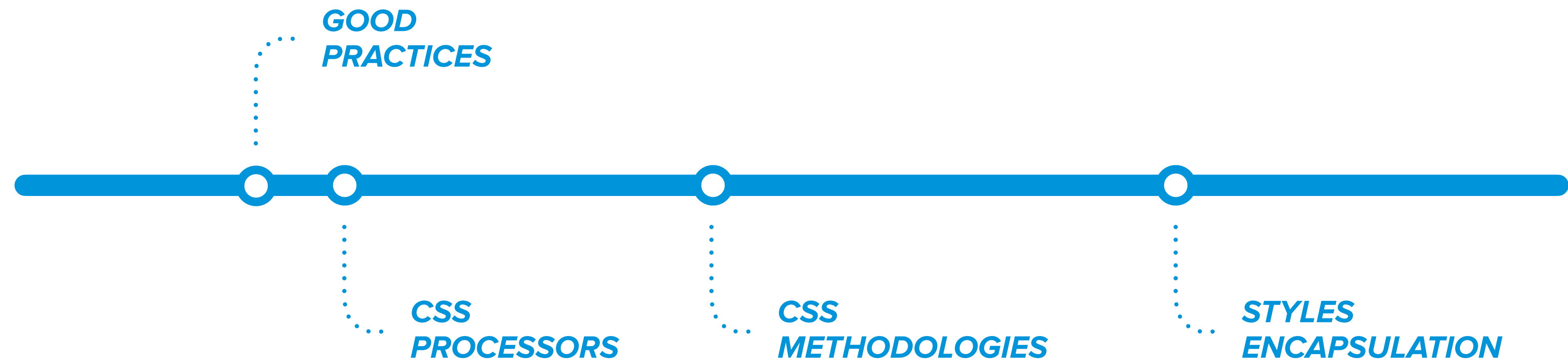
- Styles are scoped to their component
- No more naming collisions

# Industry *de facto* standard

- **Vue.js** uses the abandoned scoped attribute
- **Angular** supports Shadow DOM and Emulated
- Create **React** App has built-in support for CSS Modules
- **Next.js** and **Gatsby** support CSS Modules out of the box

# Semantic CSS

Traditional styling paradigm



PART 5/7

# Atomic CSS



# Non-Semantic classes

//  **Non-semantic** names (describing the implementation)

**.text-red {}**

**.text-green {}**

//  **Semantic** names (describing the usage & purpose)

**.text-error {}**

**.text-success {}**

# Reusability

// 🎯 specific, less reusable

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

// ♻️ highly reusable

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



# Utility CSS classes

Also used by Semantic CSS frameworks



Colors: **.text-muted** or **.text-bg-info**



Typography: **.text-center** or **.h1** or **.bold**



Spacing: **.p-3** or **.mx-auto**

# Usage

```
<!-- Example from Tailwind CSS playground -->
```

```
<img class="
  relative flex min-h-screen flex-col justify-center overflow-hidden
  bg-gray-50 py-6 sm:py-12
" />
```

# Frameworks

Complete set of CSS classes for building complex applications

- ✅ Zero CSS code duplication
- ✅ No specificity wars
- ✅ Long cache life for CSS
- 🤔 Doesn't support all CSS syntax: **descendants**, **::before**, **::after**
- 😞 Learn a new framework

PART 6/7

# CSS in JS



# CSS in JS motivation

- Unique CSS class names, like CSS Modules
- Contextual styles, like CSS Processors
- Dead code elimination
- Variable sharing

# Beyond static styling

- Dynamic styling *(state based & user defined styles)*
- Colocated styles *(Lit-Element, Vue SFC, RN)*
- Lazy loaded styles

# CSS in JS

A novel build step





*style9*



*Emotion*

*Styled JSX*



*Styled Components*



*Stitches*



*Compiled*



*TypeStyle*

*Aphrodite*



*vanilla-extract*







*style9*



*Styled JSX*



*Styled Components*



*Emotion*

*Stitches*



# ***A thorough analysis of CSS in JS***

*github.com/andreipeiffer/css-in-js*



*Compiled*



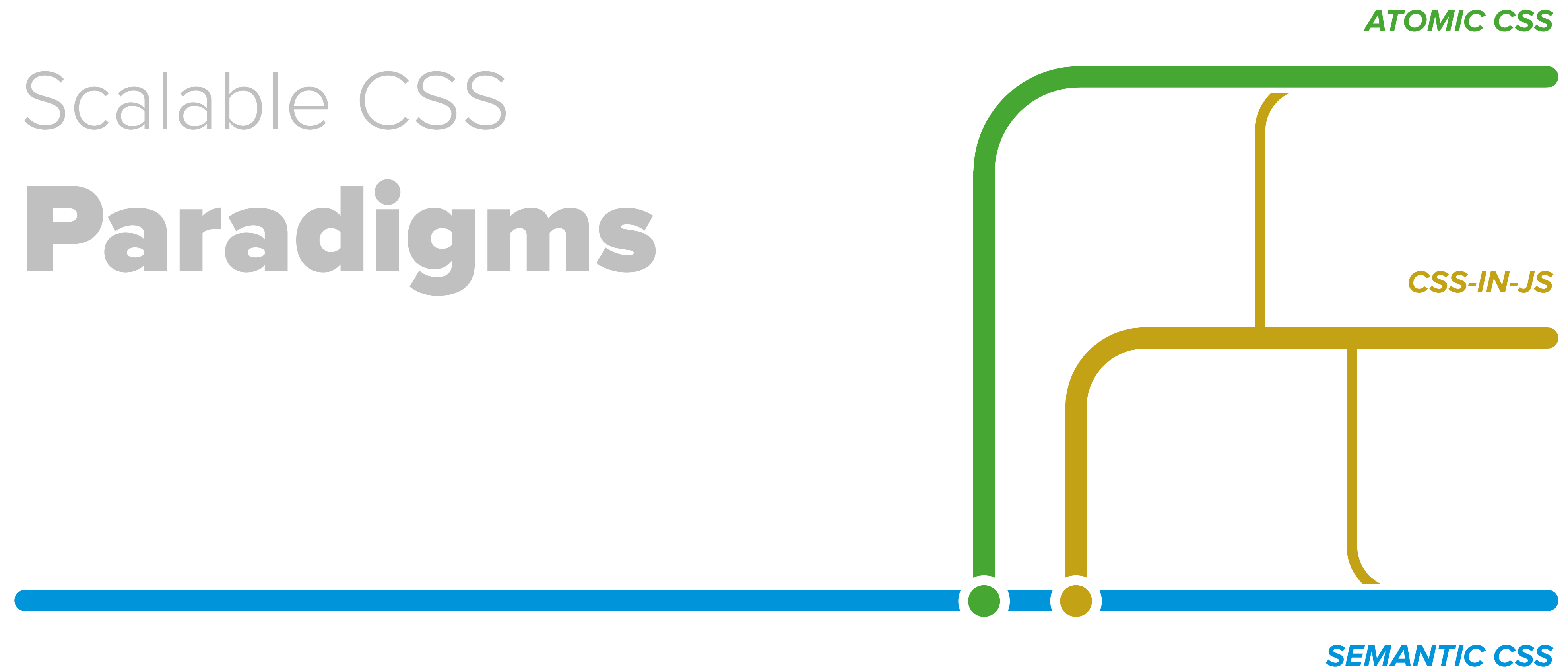
*TypeStyle*

*Aphrodite*



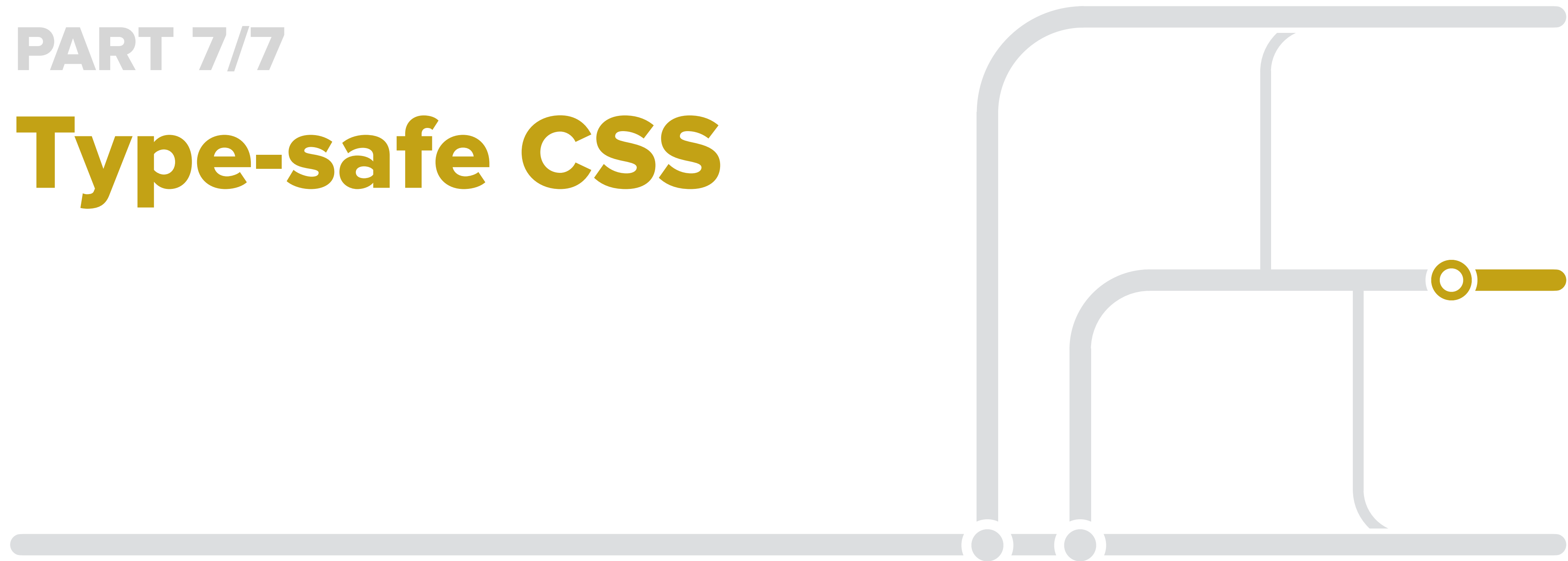
*vanilla-extract*


# Scalable CSS Paradigms



PART 7/7

# Type-safe CSS





***Type safety*** is the extent to which a programming language ***discourages or prevents type errors.***

- Wikipedia

# Lack of type safety

With standard web technologies

*Behavior*  
*JS*



⋮  
// runtime errors  
window.missing.property

*Markup*  
*HTML*



⋮  
// incorrect HTML tag nesting are auto-fixed  
<article><em> ... </article></em>

*Style*  
*CSS*



⋮  
// CSS incorrect rules are ignored  
text-align: down;

# **Strict HTML Doctypes**

Abandoned in 2007, in favor of HTML5

# **ES4 static types**

Abandoned in 2008, in favor of ES5

# Static type checkers

Non-standard tools for detecting compile time errors

- **TypeScript**, in 2012
- **Flow**, in 2015

# Static type checkers

Non-standard tools for detecting compile time errors

*Behavior*  
*TS*



// Property 'missing' does not exist on type 'Window'  
window.missing.property

*Markup*  
*JSX / TSX*



// JSX element 'em' has no corresponding closing tag  
<article><em> ... </article></em>

*Style*  
*CSS-in-TS*



// Type 'down' is not assignable to type 'TextAlign'  
text-align: down;



# Typed Interfaces

Auto-complete and type checking for consumers

```
interface NotificationProps {  
  // 🤔 too permissive  
  color: string;  
}
```

# Typed Interfaces

Auto-complete and type checking for consumers

```
interface NotificationProps {  
  // ✅ restricted to specific colors  
  color: "success" | "error";  
}
```

# Safe refactorings

Manual or automatic

- Eliminates the fear of changing code
- Refactoring tools: *Rename*, *Extract*, etc

# The missing puzzle

For type-safe Front-End development



# Towards a type-safe future

## CSS Typed Object Model

Introduced in 2018

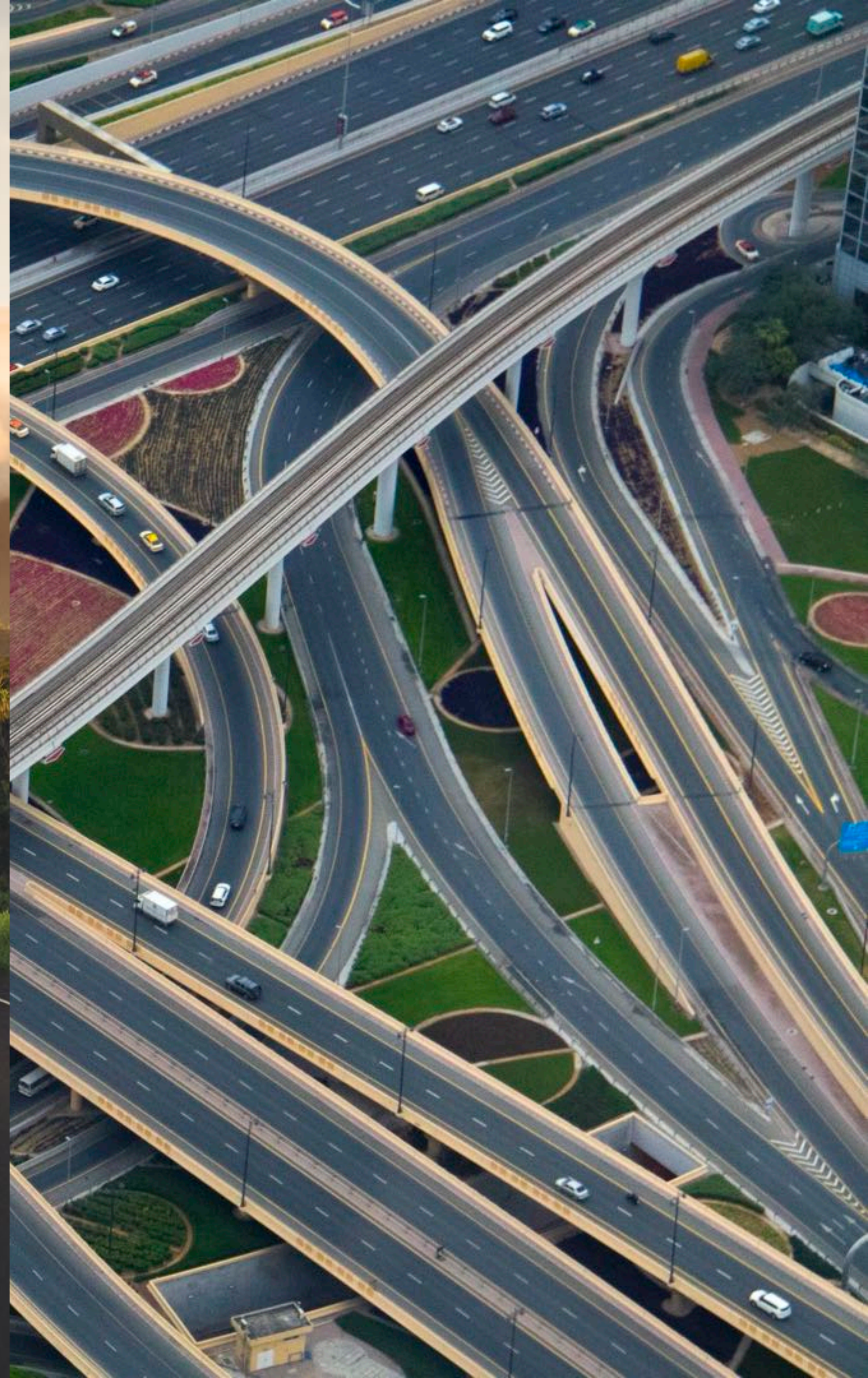
```
element.attributeStyleMap.set("font-size", CSS.em(3));
```

## ECMAScript Type Annotations

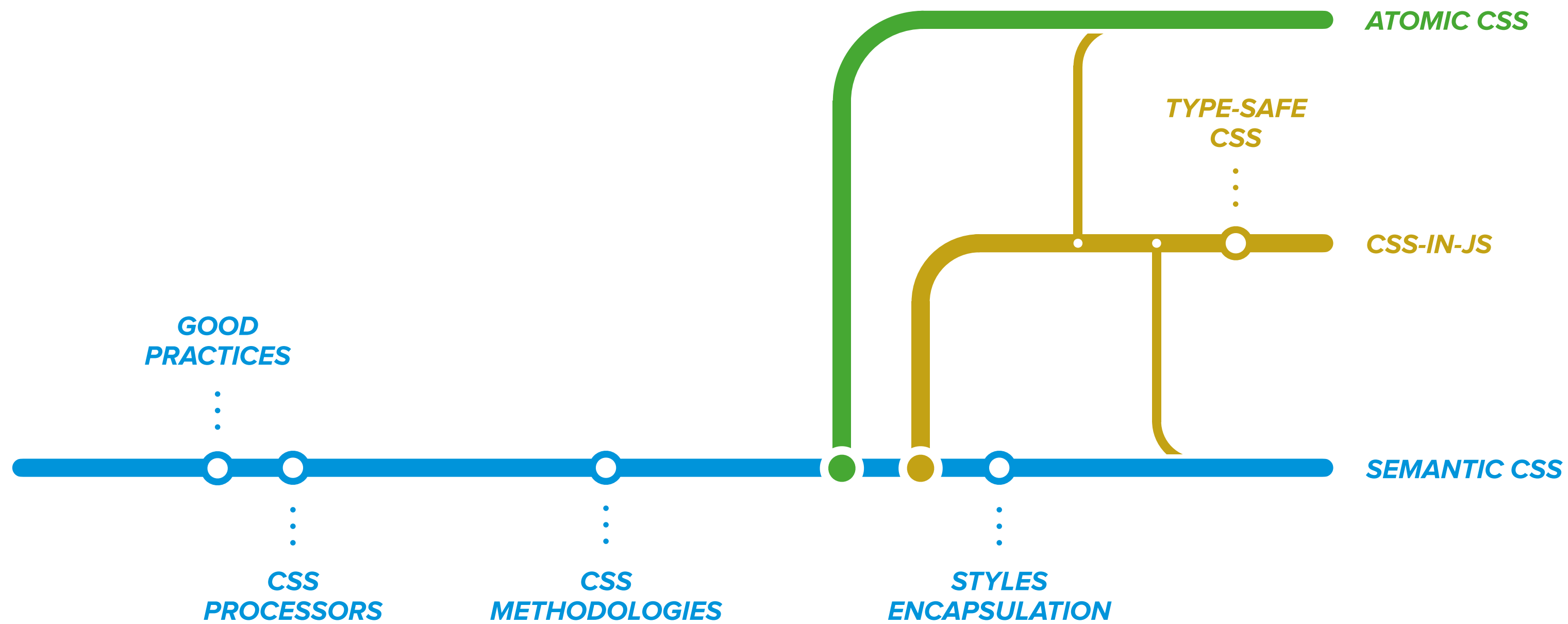
Proposed in 2022

```
function sort(list: Array<Item>, order: "asc" | "desc") {}
```









---

# ***The evolution of scalable CSS***

*andreipfeiffer.dev/blog*

# THANK YOU



*andreipfeiffer.dev*