Theming for Web Components

Two Types of Theming Customers

Customer A: Large eng orgs, design systems, ...

- Encapsulation is critical
- Components must be able to evolve without breaking users
- Want to define a tightly controlled styling API

Customer B: App components, npm/OSS packages, ...

- Encapsulation is a hindrance
- Need broad styling capabilities
- Either
  - Are their own user
  - Use versioning to manage evolution

Customer A

- Large eng organizations
- Design systems
-

Customer B Solutions

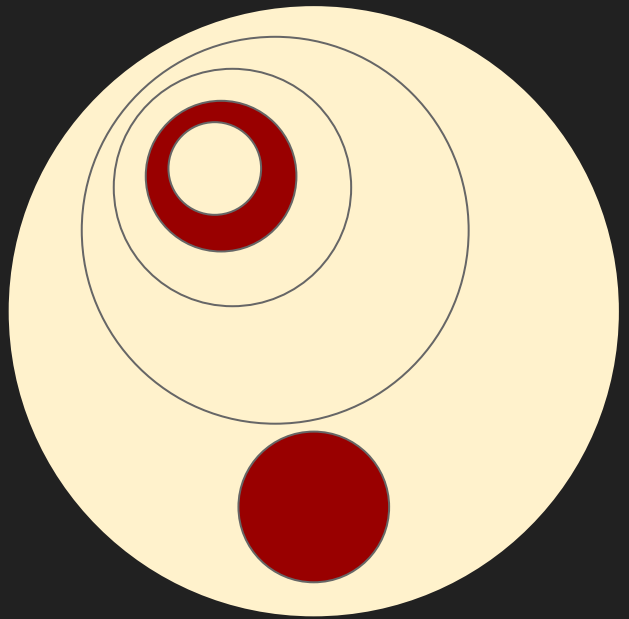
- ::theme()
- "open styleable" ShadowRoot mode

Open Styleable Shadow Roots

Shadow Roots are open for styling from any scope above it

Open non-styleable shadow roots block styling as normal

How do selectors match? Would we need /deep/?



Requirements, Level 1: Pierce Shadow Roots

- Affect styling in a shadow root
- Inherit down multiple levels of scopes
- Component opts-in to styling

✔️ Both ::part() and CSS properties affect styling in another root

✔️ Both ::part() and CSS properties require the component to opt-in

✔️ CSS properties inherit

❌ ::part() doesn't inherit

Requirements, Level 2: Targeting

- Control what properties are styleable
- Allow styling many properties, if desired

✔️ CSS properties control exactly which properties are styleable

❌ CSS properties make it cumbersome to allow styling for large number of properties

❌ ::part() does not allow easy control over what properties are styleable

✔️ ::part() does make it easy to allow arbitrary properties to be styleable

Requirements, Level 3: Abstraction

- Parameterize theming with high-level parameters
- Transform high-level parameters to concrete CSS properties

✔️ CSS properties let you define new parameter names

❓ ::part() sort of lets you define a parameter as a pseudo-element

❌ CSS properties do not effectively let you define new value types

❌ calc() is too limited for transformations to low-level properties

Custom A Solutions

- ?
- Evolve CSS itself more
  - Use CSS Custom Properties as style parameters
  - Expand CSS's ability to implement transforms from parameters to styles
  - == standardize runtime versions of Sass features?

Expressiveness

Abstraction

Modularity

Extensibility

Expressiveness

- calc()
  - Operators: ternary, comparison, logical, numeric
  - Functions: string, math, color…
  - Literals: true, false, string
- Lookup Tables
- Container Queries

Expressiveness in calc() allows components to implement richer, higher-level custom properties.

Abstraction

- Custom Functions
- Mixins

These might be mainly used by components to implement transforms from custom properties to shadow styles.

They allow sharing that implementation across many components.

Modularity

- CSS References
  - Export CSS entities (Rulesets, Mixins, Variables, etc)
  - Import CSS entities into CSS, JS, HTML
- Module-like import facility (@use?)

Modularity allows sharing of custom functions and mixins.

Developers largely want lexical scoping. Using CSS properties to refer to entities is dynamic scoping.

Extensibility

- Style Observers
- Custom CSS Features in JS: http://tabatkins.github.io/specs/css-aliases/

```
@mixin buttonType($type) {
  border-radius: $type == 'fab' ? 5px : 0;
}

#button {
  /* --button-type == 'fab', 'submit', or 'basic' */
  @include(buttonType(var(--button-type, 'basic')));
}
```