Looking forward: CSS Layers and the @layer at-rule

Warning:

This post is about CSS specifications that are still under development and, as far as I know, have no implementation in production browsers yet. Furthermore, the specification may change in the future.

To keep track of the progress, please follow the `Cascade Layers` section of the CSS Cascading and Inheritance Level 5 specification.

Looking at some of the current draft specifications, I found something that looks very promising and, when implemented, will be a very useful feature. The feature is called CSS Layers and it’s a way to group elements together so they will cascade as a unit.

> In the same way that cascade origins provide a balance of power between user and author styles, cascade layers provide a structured way to organize and balance concerns within a single origin. Rules within a single cascade layer cascade together, without interleaving with style rules outside the layer.
>
> Authors can create layers to represent element defaults, third-party libraries, themes, components, overrides, and other styling concerns—and are able to re-order the cascade of layers in an explicit way, without altering selectors or specificity within each layer, or relying on source-order to resolve conflicts across layers.

What I find the most intriguing about this functionality is how it groups elements together.

Layers allow you to organize styles in groups and to specify the order in which those groups cascade and get applied to a document.

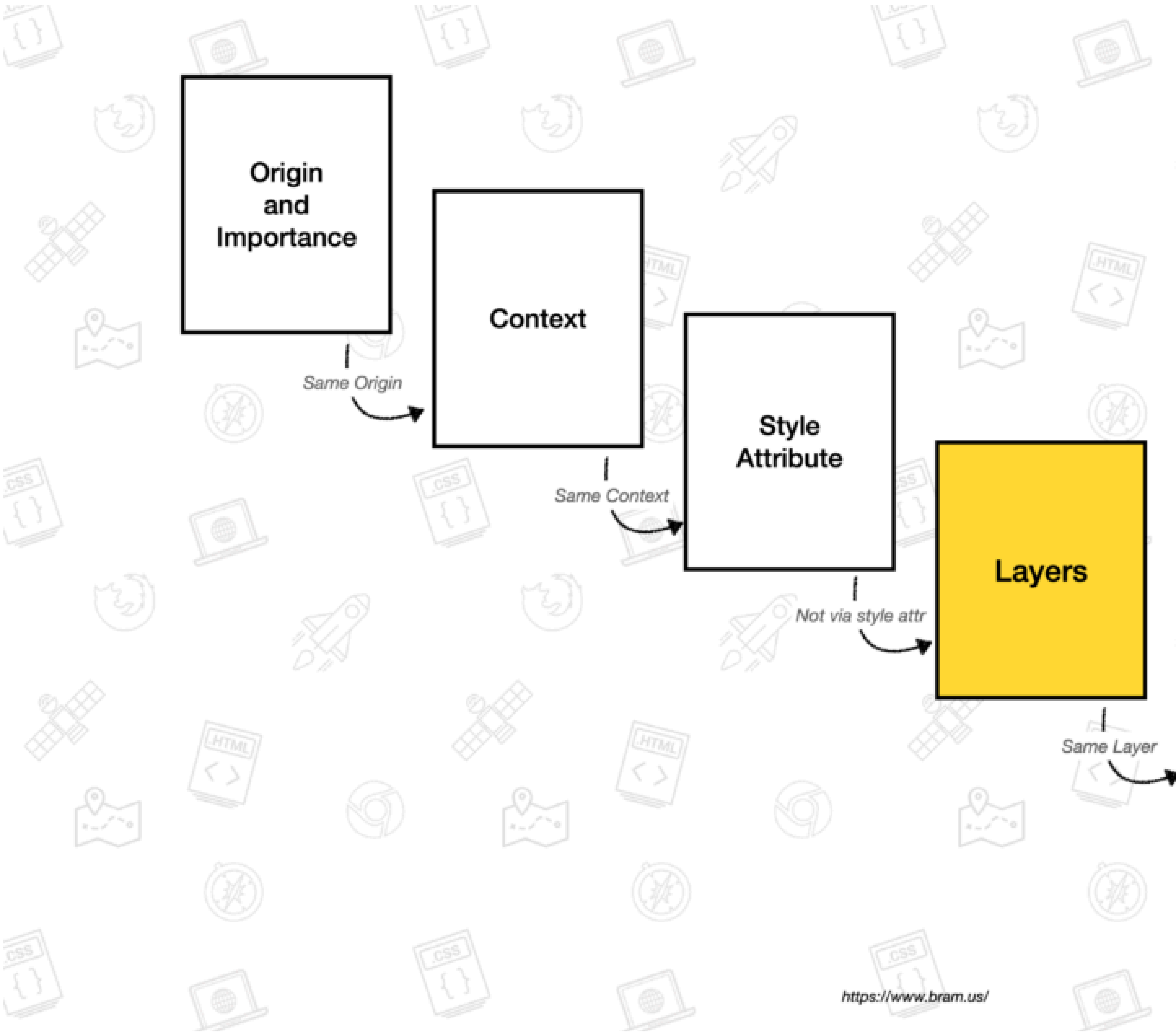

Figure 1: Where layers sit in deteriming styles. Taken from [The Future of CSS: Cascade Layers (CSS @layer)]

Layers are processed before the browser looks at specificity and the order of appearance of CSS so they help reduce some of the problems that come with specificity and the order of appearance of CSS selectors. They also provide grouping for styles; you can create layers that group styles together for specific purposes. For example, you could use a layer for reset styles, one for generic theme elements, and a different layer for specific element overrides.

This example will create a default layer, a library layer, and a theme layer. It will also define the order in which the layers cascade and are applied to the document.

The first layer declaration tells the browser the order of the layers, how they will be applied to the document.

Subsequent layer declarations define each individual layer. When rule names overlap they will cascade in the order we defined the layers in.

The document order becomes less relevant when working with layers because we've provided an explicit order to the layers.

```css
/*
  establish the layer order, so the
  "override" layer takes precedence
*/
@layer framework, override;

@layer override {
  @keyframes slide-left {
    from { translate: 0; }
    to { translate: -100% 0; }
  }
}

@layer framework {
  @keyframes slide-left {
    from { margin-left: 0; }
    to { margin-left: -100%; }
  }
}

.sidebar { animation: slide-left 300ms; }
```

We can also assign imported blocks to layers using the `layer` attribute in the import.

In the following example, the imported rules in `headings.css` and `links.css` are assigned to the `default` layer.

```css
@import url(headings.css) layer(default);
@import url(links.css) layer(default);

@layer default {
  audio[controls] {
    display: block;
  }
}
```

Details and gotchas

Some things that are worth keeping in mind about the at-rule.

Unlayered content

Unlayered content takes precedence over layered content, even if it wouldn’t under normal rules.

```css
audio {
  display: flex;
}

@layer reset {
  audio[controls] {
    display: block;
  }
}
```

In the previous example, the unlayered audio declaration will take precedence, even though the declaration of the `audio[controls]` element has a higher specificity and the layer is defined later in the declaration.

Naming layers is optional, but…

Naming your layers is not required but if you don’t they automatically become “anonymous” layers that are not accessible from anywhere else in the stylesheet; you can’t append to them and you can’t combine them with other layers.

This will create two different layers, not append the content of the second layer declaration to the first.

```css
@layer { /* layer 1 */}
@layer { /* layer 2 */}
```

Layer Nesting

You can also nest layer declarations. In the next example, the declaration of a base layer inside the `framework` layer is different than the `base` layer at the top level of the stylesheet.

Nested layers can be expressed using a `dot` notation. The base layer inside the framework layer can be referenced as **`framework.base`**.

The layers from the example below can be referenced as:

1. `base`
2. `framework.base`
3. `framework.theme`

```css
@layer base {
  p { max-width: 70ch; }
}

@layer framework {
  @layer base {
    p { margin-block: 0.75em; }
  }

  @layer theme {
    p { color: #222; }
  }
}
```

Caveats and things to keep in mind

While the spec is still in development and we don’t know if they will stay this way, there are still some things to keep in mind.

This section will get very technical so it’s ok if you skip it and jump to browser support :)

Cascade Layers and the use of !important

`!important` is a very powerful feature that will definitely help and mess up your code on equal measures.

Without going into details, which I’m not sure I fully understand, the `!important` keyword will invert the order of the rules as it increases the importance of the rule it’s applied to.

using these layers

```
@layer reset, base, theme, utilities;
```

Then the normal order would be:

- Normal reset
- Normal base
- Normal theme
- Normal utilities

Important declarations in these layers will go in the “Important User” Origin, and will be ordered in reverse:

- Important utilities
- Important theme
- Important base
- Important reset

Because “Normal Unlayered Styles” implicitly go last, this also means that “Important Unlayered Styles” will go first.

For more information see Bramus’ The Future of CSS: Cascade Layers (CSS

@layer)

Cascade Layers and conditional CSS

When a layer is inside a conditional CSS statement like a Media Query or a and the conditional evaluates to false, the `@layer` will not be included in the layer order.

If the media query later evaluates to true the `@layer` order will be recalculated and your CSS may change as a result.

In the example below the following things may happen:

- If none of the media queries match then there `@layer` array will be empty
- If the width media query matches but the color scheme doesn't then the only the `layout` @layer will apply
- Likewise, if the `prefers-color-scheme` query matches but the size doesn't then the `theme` @layer will apply
- If both queries match then the @layer order will be `layout, theme`

```
@media (min-width: 30em) {
  @layer layout {
    .title { font-size: x-large; }
  }
}

@media (prefers-color-scheme: dark) {
  @layer theme {
    .title { color: white; }
  }
}
```

What happens if you reuse a @layer

If you reuse a named @layer, the rules in the second and subsequent declarations will be appended to the first one.

The following example, while it's a little contrived, presents a case where we reuse the `base` @layer.

```
@layer base {
  p {
    color: rebeccapurple;
  }
}

@layer base {
  /* Will append to the base layer */
  p {
    font-size: 1.25em;
  }
}

@layer base {
  /* Will append to the base layer */
  p {
    line-height: 1.3;
  }
}
```

Cascade Layers vs. “Name-Defining Rules”

Name defining rules like `@keyframes`, `@scroll-timeline`, `@font-face`, and others follow Layer Order.

This example, a repetition of an earlier example, provides two layers, each defining a different set of rules.

Since we declared the order of the layers, the last layer in the order we specified will take precedence.

```
/* establish the layer order, so the "override" layer takes precedence */
@layer framework, override;

@layer override {
  @keyframes slide-left {
    from { translate: 0; }
    to { translate: -100% 0; }
  }
}

@layer framework {
  @keyframes slide-left {
    from { margin-left: 0; }
    to { margin-left: -100%; }
  }
}

.sidebar { animation: slide-left 300ms; }
```

No Interleaving of @import/@namespace and @layer

The CSS Working Group decided to not allow interleaving of @import and @namespace and @layer at-rules.

```
@layer default;
@import url(theme.css) layer(theme);
@layer components; /*  This @layer statement
here will make all subsequent @import rules
be ignored. In this example however
there are none */

@layer default {
  audio[controls] {
    display: block;
  }
}
```

Instead, you should do the following:

- define your layers
- group all your @import and @namespace rules
- add your @layer rules

```
@layer default;
@import url(theme.css) layer(theme);
@import url(default.css) layer(default);

@layer components;
@layer default {
  audio[controls] {
    display: block;
  }
}
```

Browser support

The support for the CSS Cascade Layers is wider than I thought it would be, even though it's still under development.

All major browsers support the @layer at-rule in their development versions and will release sometime between now and summer 2022 (or whenever Apple releases the next version of the operating system).

Browser	**Support**
Chromium (Blink)	Available in Chrome 96+ (current Canary) via the `#enable-cascade-layers` feature flag in `chrome://flags/`.
Firefox (Gecko)	Available in Firefox 94+ (current Canary) by setting `layout.css.cascade-layers.enabled` to true via `about:config`.
Safari (WebKit)	Available in Safari Technology Preview 133. To enable it, choose ***Experimental Features*** → ***CSS Cascade Layers*** from the Develop menu.

Use this Codepen from Miriam to test if your browser supports Layers.