CSS Parser Extensions

bramus@google.com – 2025.08.21 (CSSWG F2F)

Bramus

🦋 @bram.us
🦣 @bramus@front-end.social
𝕏 @bramus

https://www.bram.us/

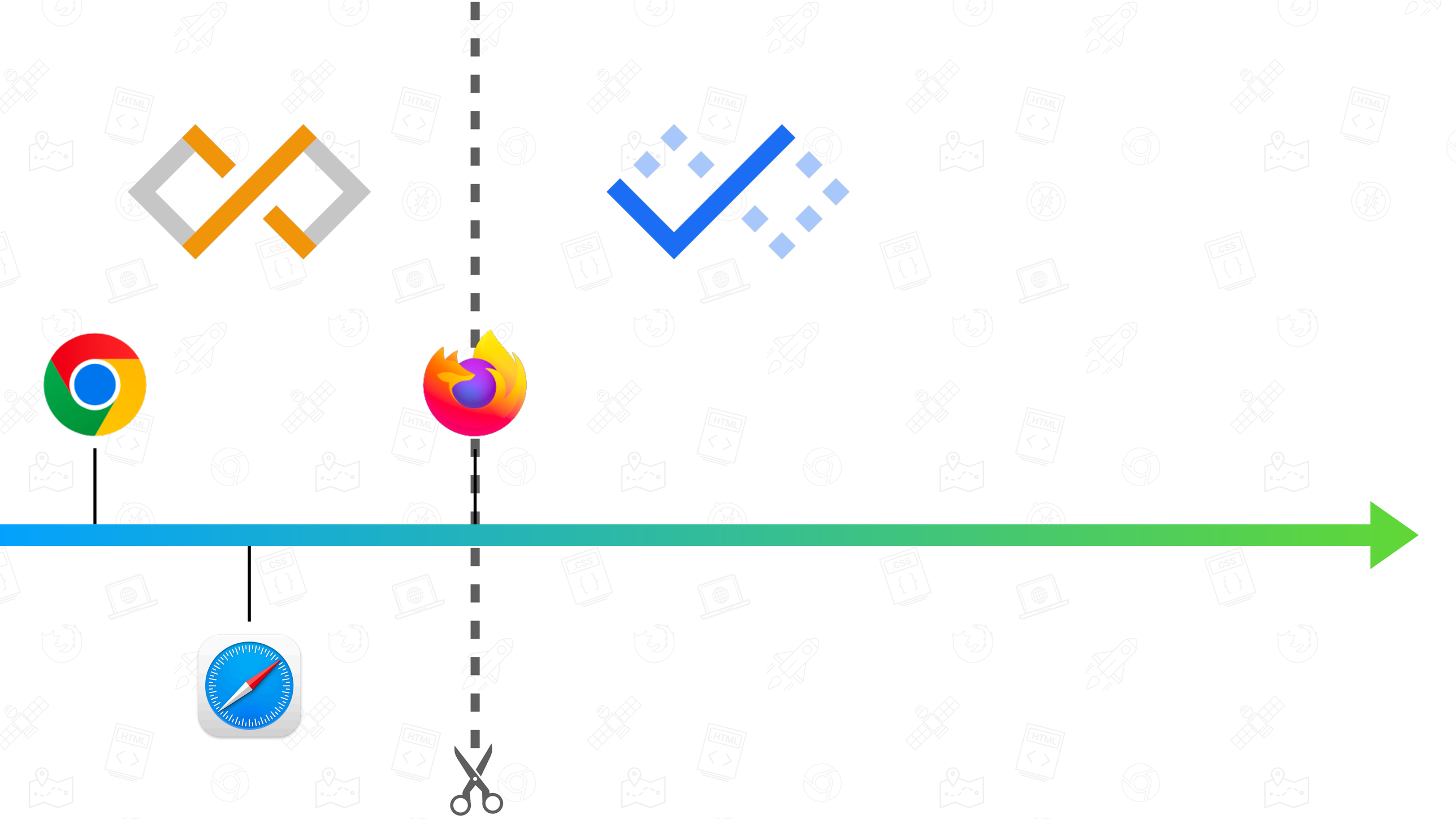
I'm with Chrome DevRel

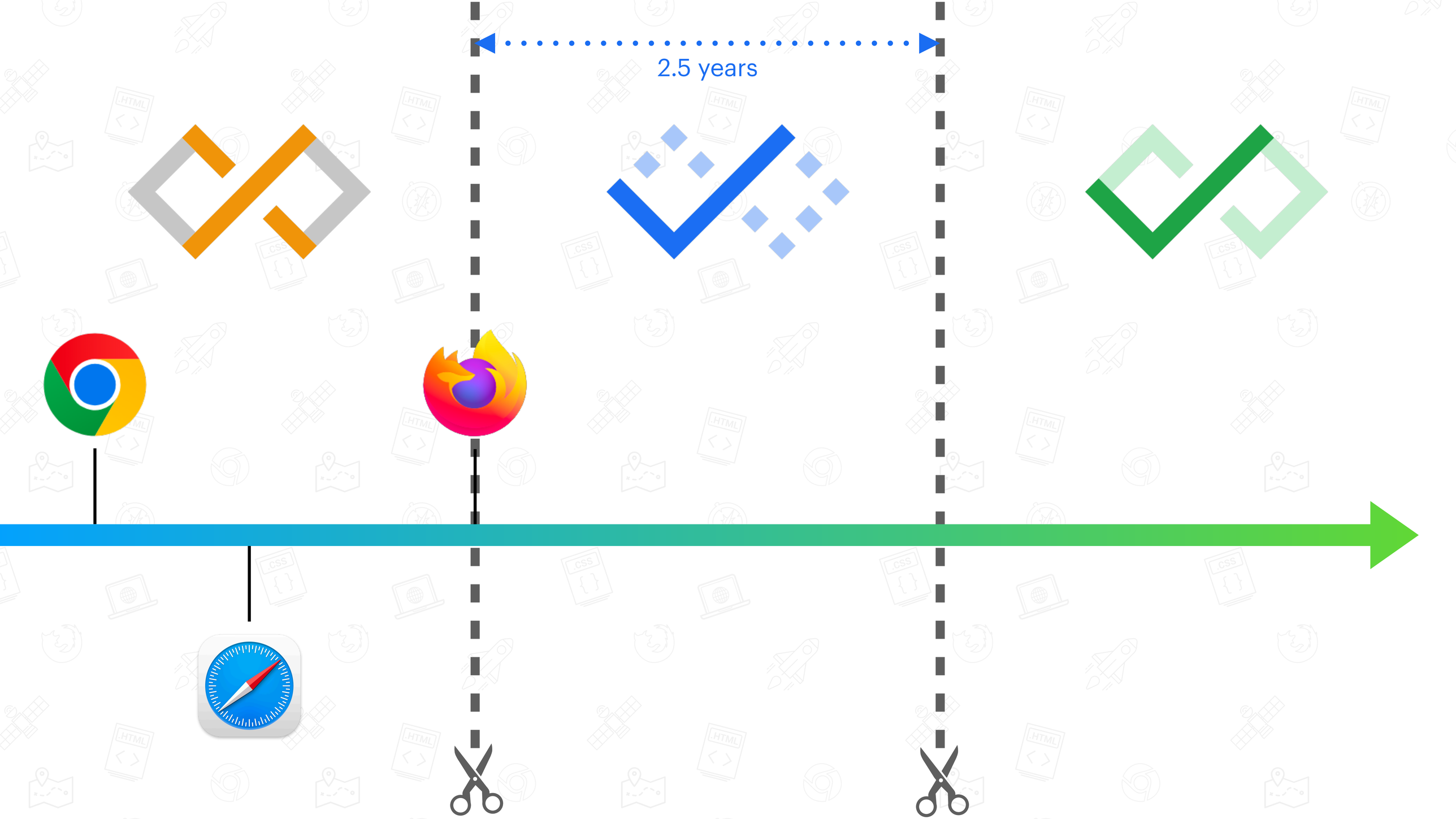
I'm with Chrome DevRel

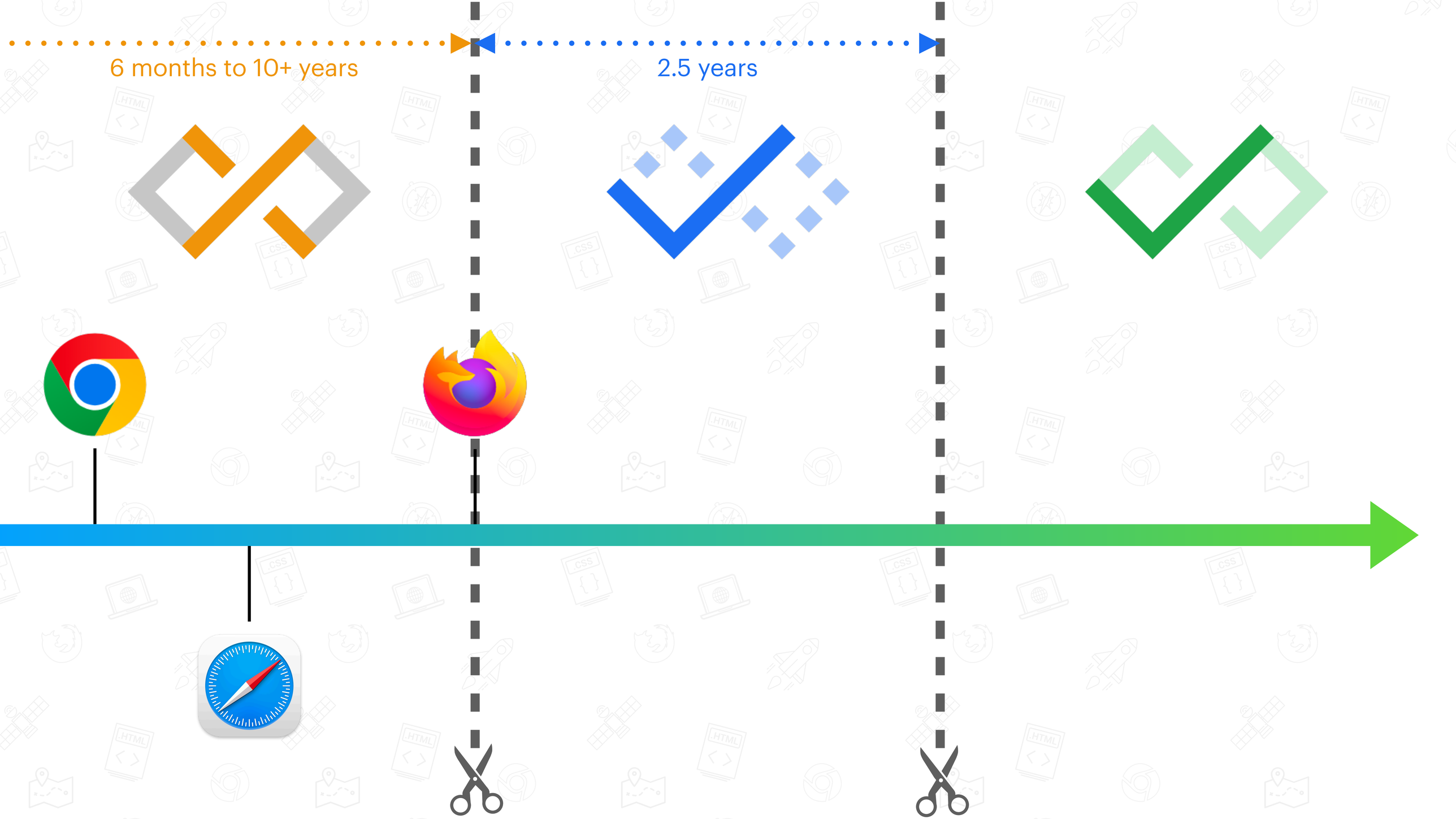
Let's start the conversation

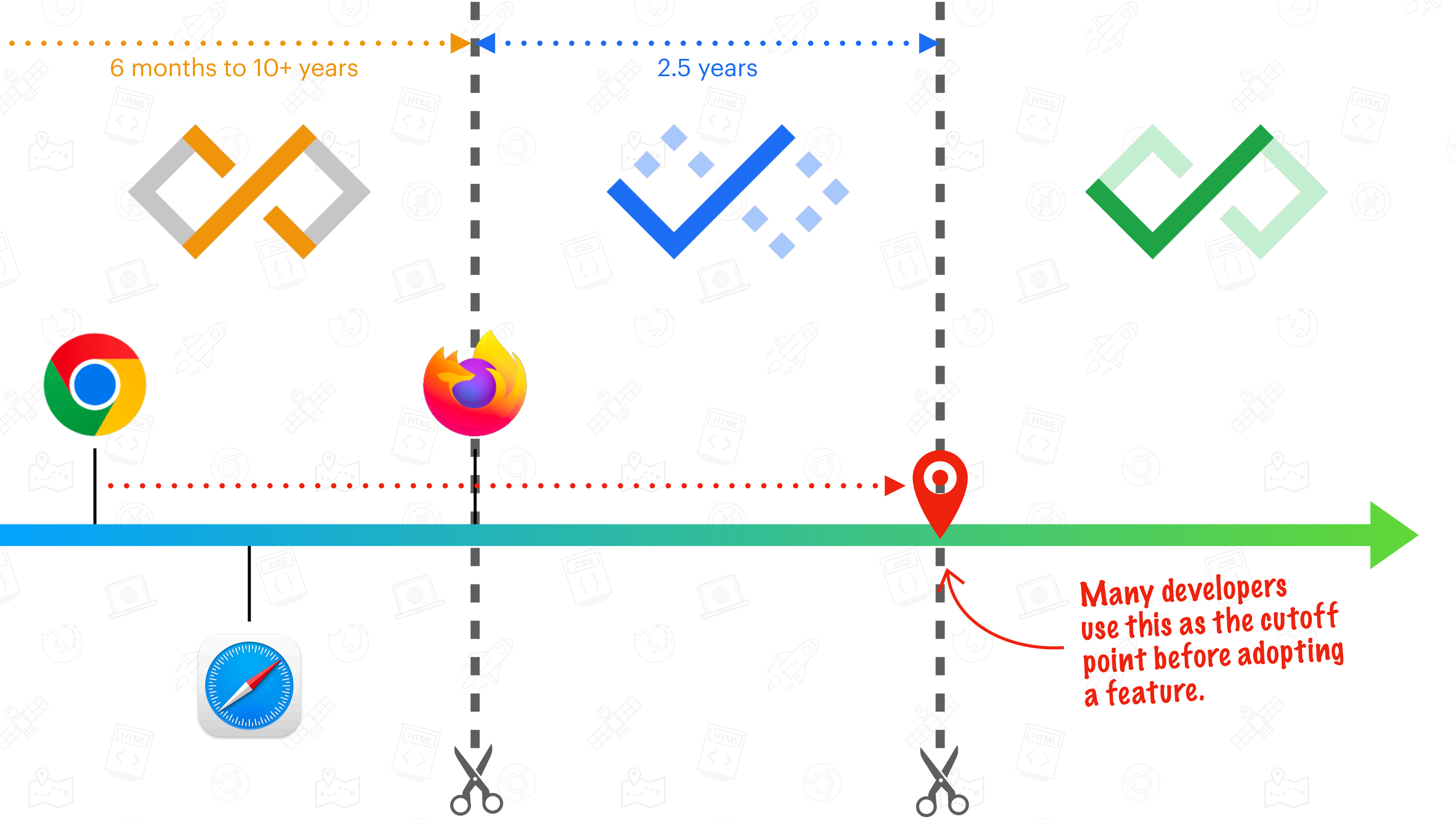
So there's this new CSS feature...

A timeline




2.5 years


6 months to 10+ years
2.5 years


6 months to 10+ years
2.5 years
Many developers use this as the cutoff point before adopting a feature.

How DevRel currently deals with this

Authors have options ...

Progresive Enhancement

Progressive Enhancement

Easy to convince features

- CSS `text-wrap: pretty`
- CSS `interpolate-size: allow-keywords`
- …

Progressive Enhancement

Difficult to convince features

- Scroll-driven Animations

“Progressive Enhancement becomes a challenge when C-level folks use iPhones and can't see the feature”

[REDACTED]

Polyfills

Polyfills

Prerequisites

- The CSS feature can entirely replace a 3rd party solution
  - No two code paths to maintain

Polyfills
Requirements

- Small
- “It Just Works”
  - No special way or writing your CSS
  - No preprocessing
- Feature complete
  - Although some things can never be polyfilled

“If you’ve never tried writing a CSS polyfill yourself, then you’ve probably never experienced the pain”

Philip Walton

(*The Dark Side of Polyfilling CSS*, Dec 2016)

How to (*try to*) polyfill CSS today

A near impossible task ...

The Problem

The CSS Parser discards what it doesn't understand

The root cause of all CSS polyfill pain

```
#element {
    background-color: invalid;
    color: blue;
    size: 2rem;
}
```

How polyfill authors work around this

Polyfilling CSS

A three step approach

1. Gather all styles
2. Parse the styles
3. Apply the styles

Polyfilling CSS

1. Gather all styles

- Sources
  - `document.styleSheets`
  - `document.adoptedStylesheets`
  - Element-attached styles
- Watch for mutations

Polyfilling CSS

2. Parse the styles

- Deal with CORS
- Tokenize and parse
  - The UA already did that …
- Stay up-to-date with the CSS Syntax
  - E.g. Cascade Layers, CSS Nesting, …

Polyfilling CSS

3. Apply the styles

- Write your own Cascade
- Set up listeners
  - Media Queries, Style Queries, ...

Run their own parser and cascade

What polyfill authors currently do

CSS Parser Extensions

What if ...

The Pitch

The CSS Parser discards what it doesn't understand

The root cause of all CSS polyfill pain

~~Run their own parser and cascade~~

What polyfill authors currently do

What if we could teach the CSS parser some new tricks?

CSS.parser

JavaScript access to the CSS Parser

Teaching the parser new tricks

Braindump

- Registration of CSS language features
  - `CSS.parser.registerSyntax(name, syntax);`
  - `CSS.parser.registerKeyword(keyword, options);`
  - `CSS.parser.registerFunction(syntax, options);`
  - `CSS.parser.registerProperty(name, options);`
  - …
- Define how they compute
- Define what to do when matching/unmatching an element

```
CSS.parser
  .registerKeyword('random:<number>', {
    caching_mode: CSS.parser.caching_modes.PER_MATCH,
    invalidation: CSS.parser.invalidation.NONE,
  })
  .computeTo((match) => {
    return Math.random();
  });
;
```

```
CSS.parser
  .registerFunction(
    'light-dark(light:<color>, dark:<color>):<color>',
    { invalidation: ['color-scheme'] }
  )
  .computeTo((match, args) => {
    const { element,  property, propertyValue } = match;
    const colorScheme =
      CSS.parser.getSpecifiedStyle(element)
      .getPropertyValue('color-scheme');

    if (colorScheme == 'light') return args.light;
    return args.dark;
  })
);
```

```
CSS.parser
  .registerFunction('at-rule(keyword:<string>):<boolean>', {
    caching_mode: CSS.parser.computation_modes.GLOBAL,
  })
  .computeTo((match, args) => {
    switch (args.keyword) {
      case '@view-transition':
        return ("CSSViewTransitionRule" in window);
      case '@starting-style':
        return ("CSSStartingStyleRule" in window);
      // …
      default:
        return false;
    }
  })
;
```

```
CSS.parser
  .registerProperty('size', {
      syntax: '[<length-percentage [0,∞]> | auto]{1,2}',
      initialValue: 'auto',
      inherits: false,
      percentages: 'inline-size'
      animatable: CSS.parser.animation_types.BY_COMPUTED_VALUE,
  })
  .computeTo(…)
  .onMatch((match, computedValue) => {
    const { element, specifiedValue } = match;
    return {
      'width': computedValue[0],
      'height': computedValue[1] ?? computedValue[0],
    };
  });
;
```

```
CSS.parser.registerProperty('scroll-timeline', { … });

CSS.parser
  .matchProperty('scroll-timeline')
  // No .computeTo … so it would just return the declared value
  .onMatch(parserMatch => {
    const resizeObserver = new ResizeObserver((entries) => {
        // …
    });
    resizeObserver.observe(parserMatch.element);
    parserMatch.data.set('ro', resizeObserver);
  })
  .onElementUnmatch(parserMatch => {
    const resizeObserver = parserMatch.data.get('ro');
    resizeObserver.disconnect();
  })
;
```

```
CSS.parser
  .registerSyntax(
    '<single-animation-timeline>',
    'auto | none | <dashed-ident> | <scroll()> | <view()>'
  )
;
```

Benefits

CSS Parser Extensions

Benefits for polyfill authors

- Easier to author polyfills
- Smaller polyfills
  - `css-anchor-positioning` -85%
  - `container-query-polyfill` -64%
  - `scroll-timeline-polyfill` -48%
- More performant polyfills
- More robust polyfills
  - Half of the open `scroll-timeline-polyfill` issues are parsing issues

CSS Parser Extensions

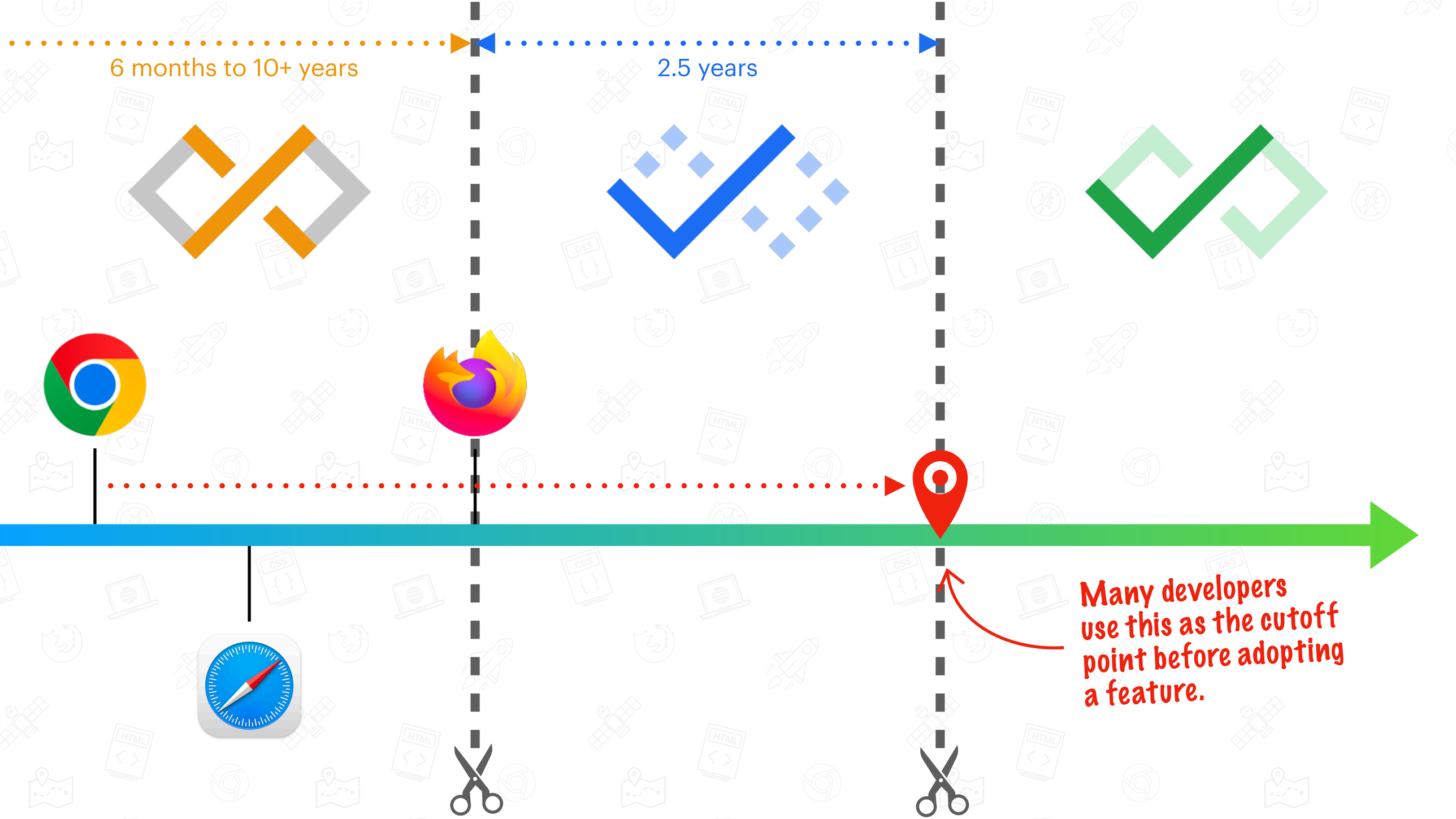
Benefits for authors

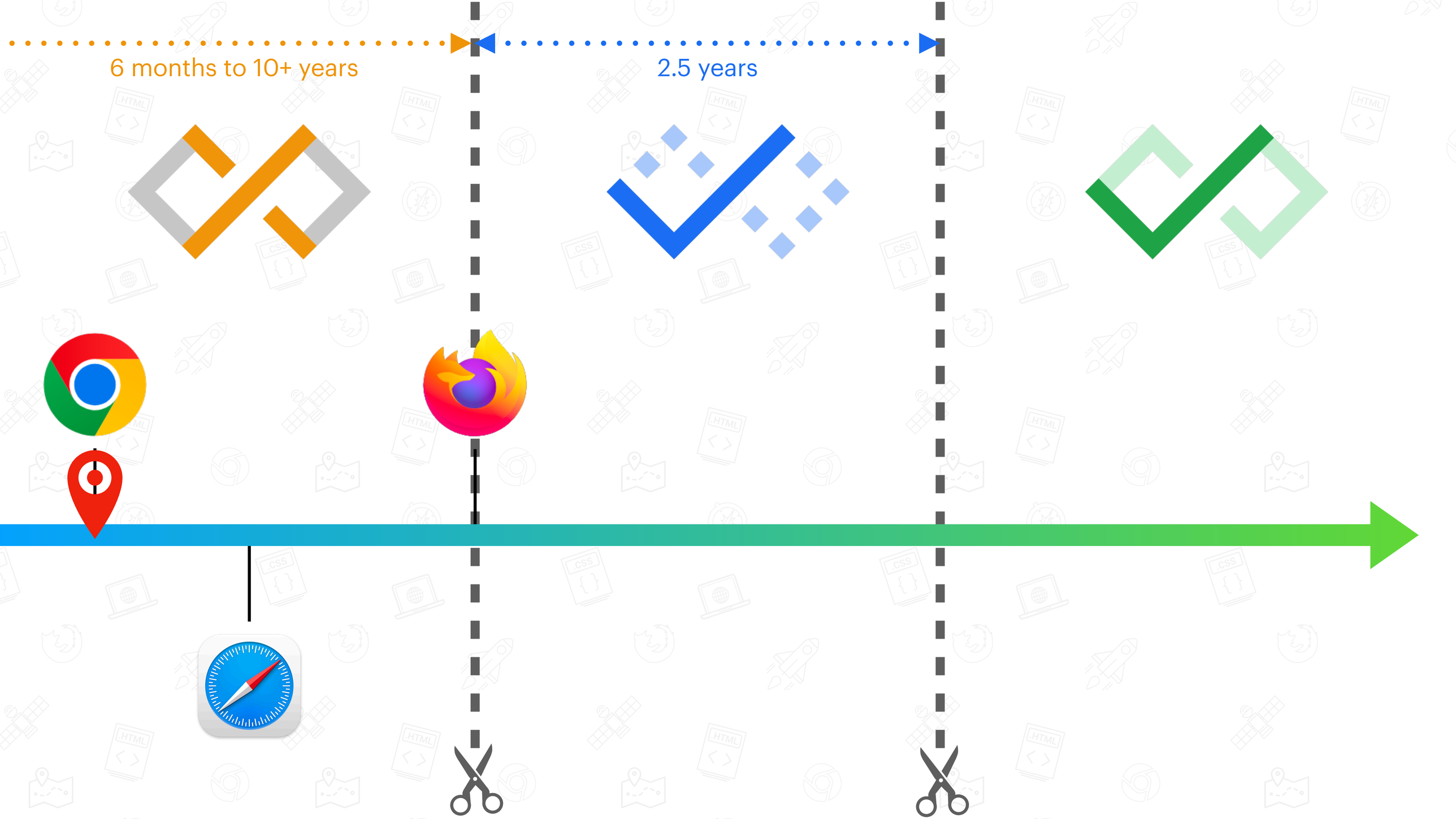
- Drop-in polyfills that work with the native (future) syntax
- Features work as if they are supported natively

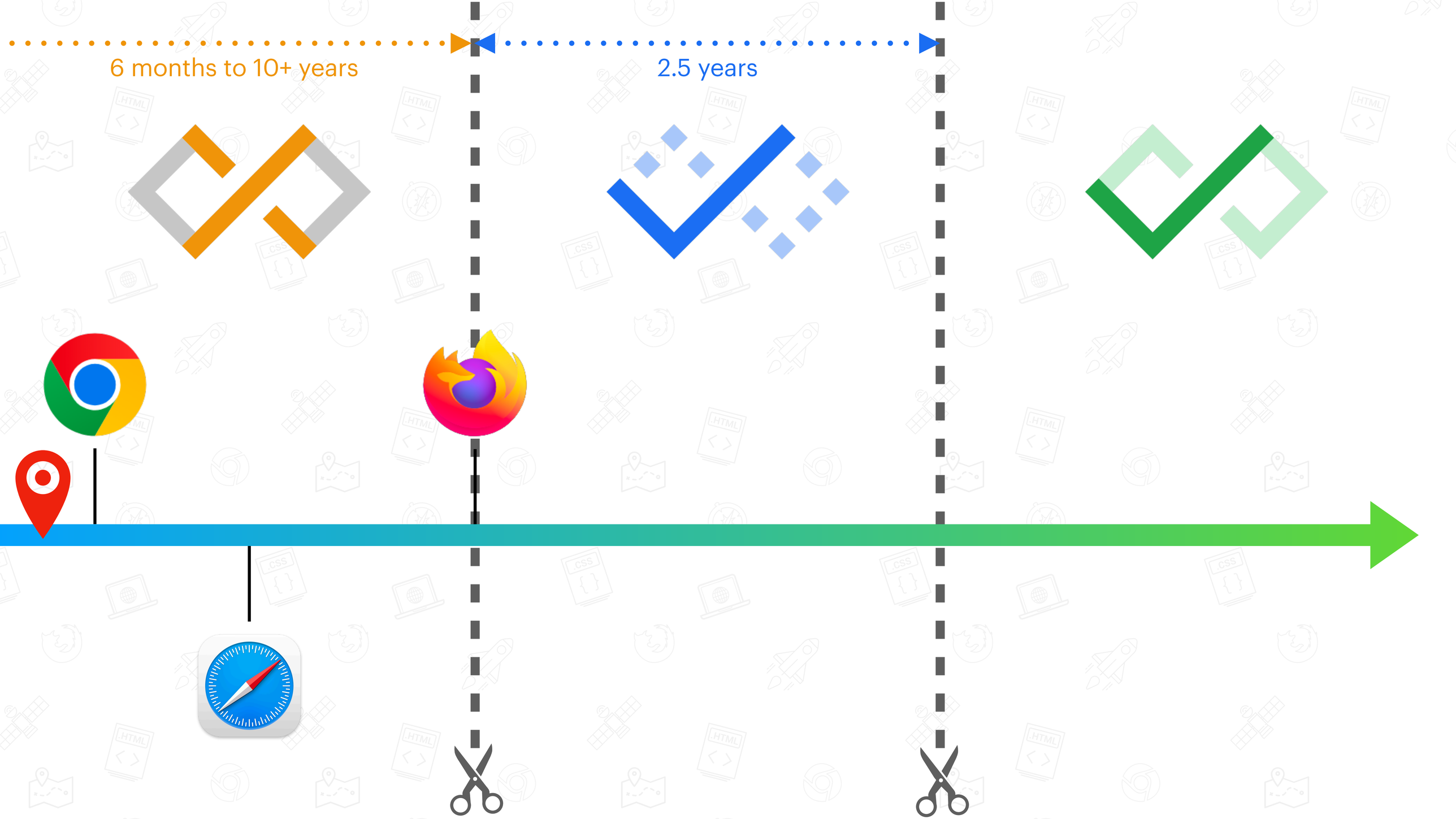
CSS Parser Extensions

Benefits for browser vendors

- Faster adoption of CSS features
- Ability to prototype in the browser before doing so in your engine
- Ability to get early feedback from authors


6 months to 10+ years
2.5 years
Many developers use this as the cutoff point before adopting a feature.


6 months to 10+ years
2.5 years


6 months to 10+ years
2.5 years

Risks / Caveats

CSS Parser Extensions

Risks / Caveats

- Need to carefully think about the timing of all this
  - Modeled after ResizeObserver?
- We don’t want another SmooshGate
  - Registrations always win
- This falls or stands with buy-in from all browser vendors
- Not every CSS feature can be polyfilled

That’s it

Time for discussion

(Extra slides)

CSS Parser Extensions

Caching Modes

- NONE
  - The value is not cached and needs to be recomputed every time an invalidation happens
- PER_MATCH *(default)*
  - The computed value is cached for the remainder of the lifecycle of the page.
- PER_RULE
  - The computed value is cached for the rule that matched the element. I.e. two occurrences in the same rule have the same value.
- PER_ELEMENT
  - The computed value is cached for the element. I.e. multiple rules that use the same keyword will have the same value.
- GLOBAL
  - The computed value is computed once and the result is cached globally. All other or successive computations yield the same result

CSS Parser Extensions

Invalidation Sources

- PROPERTIES
  - List of of properties to track
- STATE
  - List of IDL attributes to track
- DOM_MUTATION
  - List of Content Attributes to track
- ...

CSS Parser Extensions

Utility Functions

- CSS.computeLength
  - To compute a <length> to its actual pixel values
  - Needs the element to be passed along with so, so that it can auto-get the necessary stuff for percentages, em's, etc.
- CSS.isAbsoluteLength and friends
- CSS.parser.getSpecifiedStyle (or window.getSpecifiedStyle)
  - See w3c/csswg-drafts#10002

```
CSS.parser.registerProperty('animation-range-start', {
    syntax: '<single-animation-range>',
    initialValue: 'none',
    inherits: false,
    percentages: customPercentageComputationFunction,
    animatable: CSS.parser.ANIMATABLE_DISCRETE,
});
CSS.parser.registerProperty('animation-range-end', { … });
CSS.parser.registerProperty('animation-range', {
    syntax: "[ <'animation-range-start'> <'animation-range-end'>? ]#",
    shorthand: true,
});
```

```
CSS.parser
  .matchProperty(['animation-range-start', 'animation-range-end'])
  .computeTo(match => {
    const { element, specifiedValue } = match;
    const animationTimeline = CSS.parser.getSpecifiedStyle(element)
      .getPropertyValue('animation-timeline');

    // Parse the range to pixel values based on the the timeline
    const toReturn = parseRange(specifiedValue, animationTimeline);

    // The computed value is the declared one
    return toReturn;
  })
;
```