Google

Anchor positioning Spec Updates

CSSWG F2F February 2024

February 2024

Anchor Spec Additions

01 Anchor-center alignment value

02 Grid-based syntax (inset-area)

03 Auto-positioning rewrite

04 Animating fallback positions

05 Scoping anchor names

06 Tether

Update 01

Anchor-center alignment value

Anchor-center

[spec link]

This is a new keyword addition to the `align-self` and `justify-self` property.

The new keyword is effective only if the element is absolutely-positioned and has a valid default anchor. Otherwise, it behaves the same as `center`.

If effective:

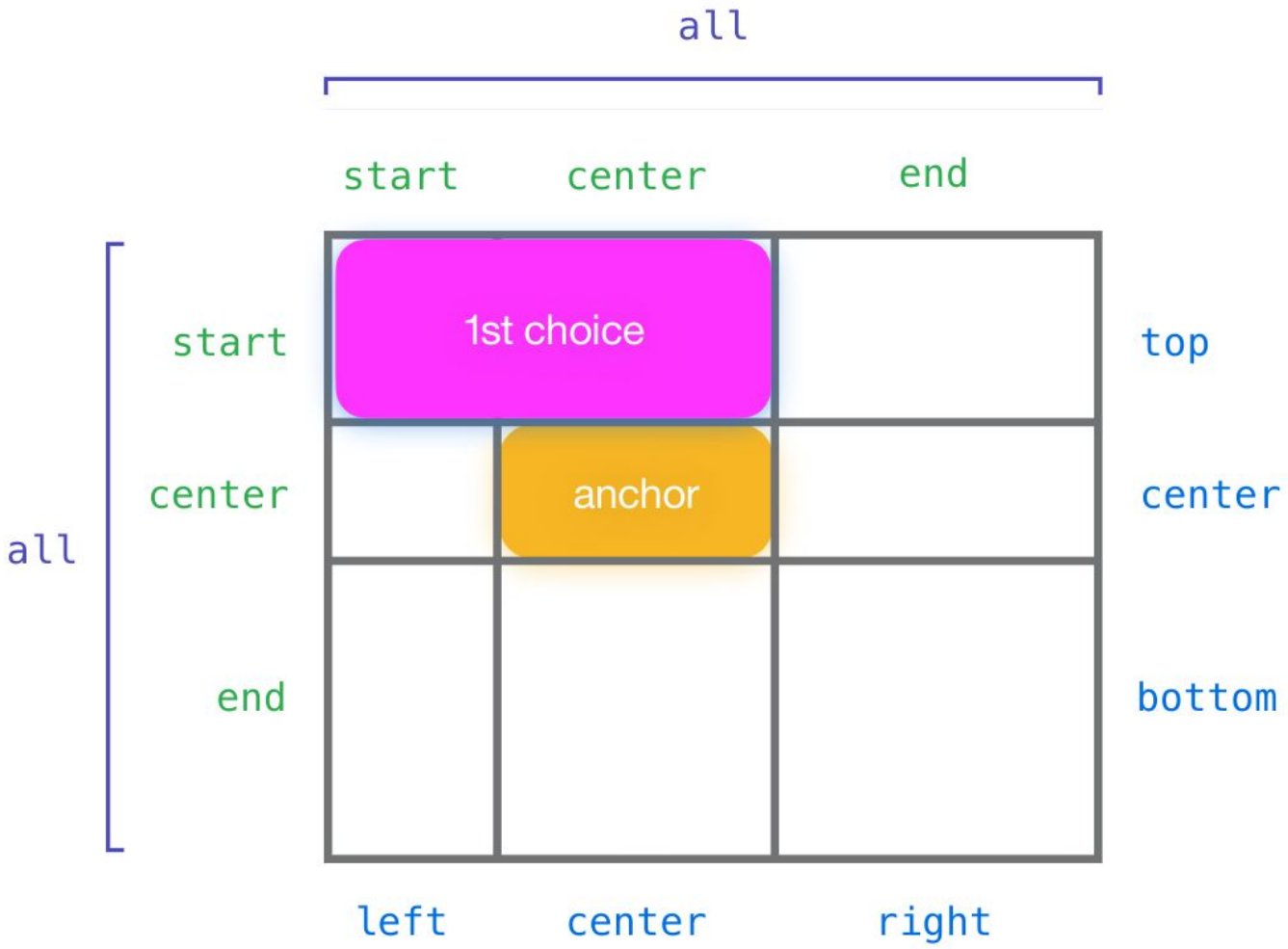
- When calculating the inset-modified containing block, and when both inset sides are `auto`, it creates an available space that is centered at the default anchor and expands as far as possible until reaching the containing block boundary.
- When placing the element, it is center-aligned with the default anchor.

Update 02

Grid-based syntax (inset-area)

inset-area

[spec link]



inset-area

[spec link]

Similar to `grid-area` changing the containing block for an abspos child of a Grid, `inset-area` changes the containing block to a segment of an anchor-related 3x3 grid. (Related to the same-named feature from the Apple feedback.)

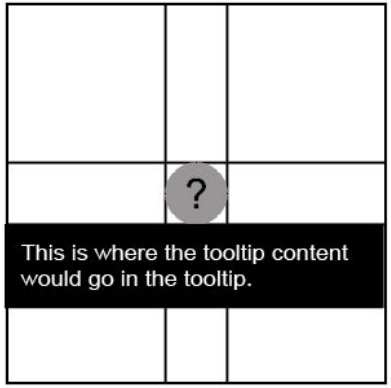
Values like:

- `top left` (into the top left corner)
- `center` (on top of the anchor)
- `center-block-start all` (in the first and second rows, across all columns)

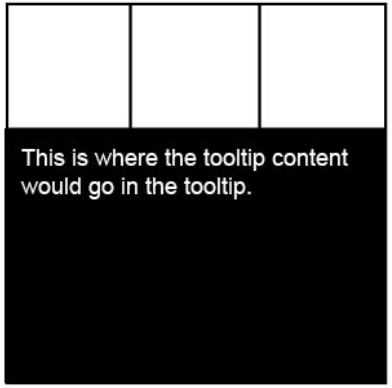
Also changes how `place-self: normal` resolves, to align it either snug to the anchor or centered (`anchor-center`) on the anchor automatically.

Because containing block changes, you can refer to the region's size directly, like:
`inset-area: bottom; max-height: 100%; overflow: auto;`
(centered on the anchor horizontally, snug against its bottom edge, content-sized unless that would overflow)

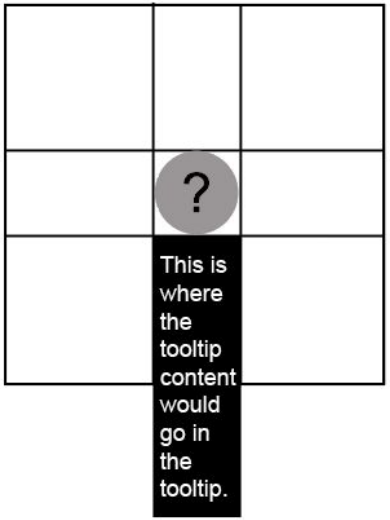
inset-area




```
inset-area: bottom;
inset-area: bottom all;
inset-area: block-end;
```




```
inset-area: center-bottom;
inset-area: center-bottom all;
inset-area: center-block-start;
```

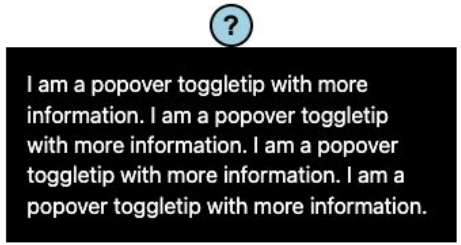



```
inset-area: bottom center;
inset-area: block-end center;
```

inset-area

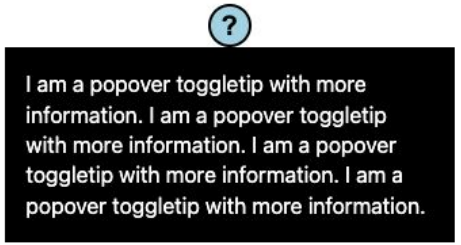
[live demo]

PREV




```
.toggletip {
  top: anchor(bottom);
  align-self: top;
  justify-self: anchor-center;
}
```

NEW




```
.toggletip {
  inset-area: bottom;
}
```

Update 03

Auto-positioning rewrite

Auto-positioning

[spec link]

Old spec: `anchor(auto)` would "magically" create fallback sets using the opposite inset.

Now: `flip-block`, `flip-inline`, `flip-start` values in `position-try-options`

Generates a fallback set inverting *all* the relevant properties - inset-area, insets, margins, alignment.

Update 04
Fallback Rewrite

Fallback

[spec link]

Changed syntax model slightly to reduce amount of indirection.

Before:

- `position-fallback` to select a `@position-fallback` rule
- `@position-fallback --foo{ @try{/* styles */} @try{/* styles */} }`
- `anchor(auto)` to do a very limited auto-generation of styles

Now:

- `position-try-options` to give a list of options
- `@position-try --foo {/*styles*/}` to create *one* named style set
- `flip-block`/`flip-inline`/etc to specify an auto-generated style set

Like: `position-try-options: flip-block, --foo, --bar;`

Fallback

[spec link]

Also, `position-try-order` to sort the fallback options according to the size of the containing block. Values like `most-width`, `most-block-size`, etc. (Many anchoring JS libraries do this.)

`position-try` shorthand to set both at once.

```
.tooltip {
  position: fixed;
  position-try: most-block-size --below-anchor, --above-anchor;
```

Simple flip fallback positioning

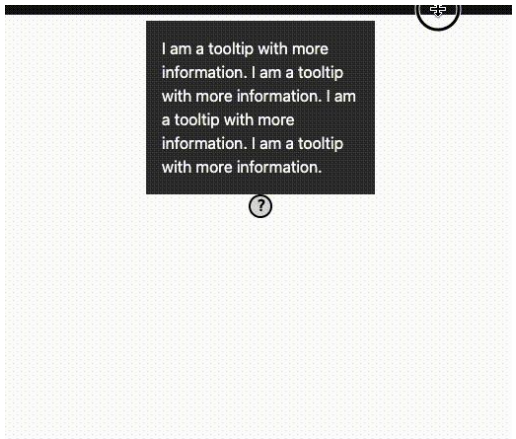
[live demo (@position-fallback)]

PREV

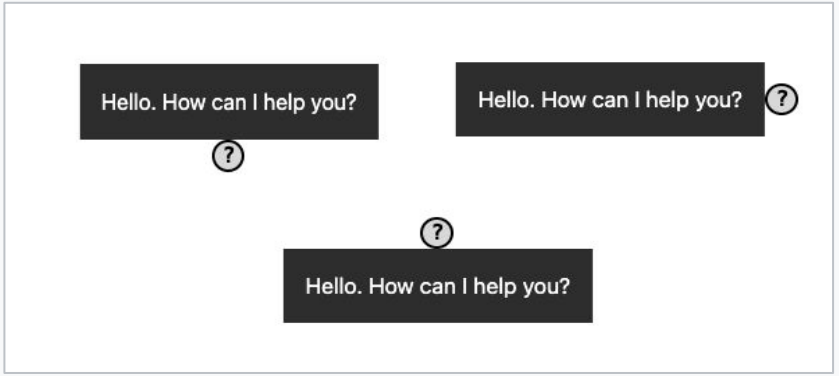
```
@position-fallback --top-then-bottom {
  @try {
    bottom: anchor(top);
    left: anchor(center);
  }

  @try {
    top: anchor(bottom);
    left: anchor(center);
  }
}

#my-tooltip {
  position-fallback: --top-then-bottom;
  /*  Centering  */
  translate: -50% 0;
}
```

NEW

```
#my-tooltip {
  inset-area: top;
  position-try-options: flip-block;
}
```



Complex fallback positioning



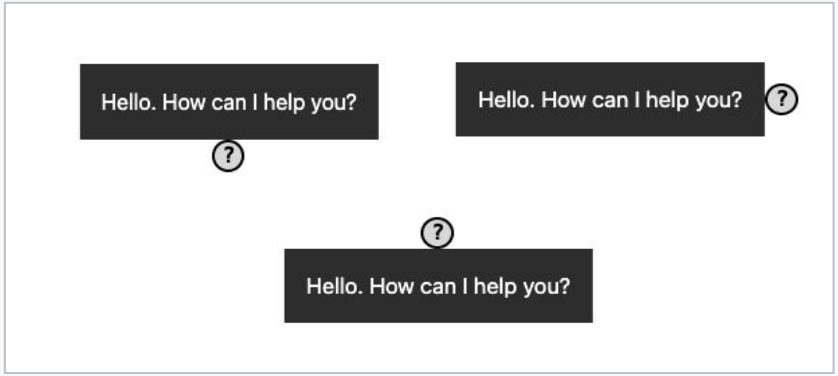
PREV

```
@position-fallback --top-left-bottom {
  @try {
    bottom: anchor(top);
    left: anchor(right);
  }

  @try {
    top: anchor(center);
    left: anchor(left);
  }

  @try {
    top: anchor(bottom);
    left: anchor(center);
  }
}

#my-tooltip {
  position-fallback: --top-left-bottom;
}
```

NEW

```
@position-try --left {
  inset-area: inline-start;
}

#my-tooltip {
  inset-area: top;
  position-try-options: --left, flip-block;
}
```

Complex fallback positioning



PREV

```
@position-fallback --top-left-bottom {
  @try {
    bottom: anchor(top);
    left: anchor(right);
  }

  @try {
    top: anchor(center);
    left: anchor(left);
  }

  @try {
    top: anchor(bottom);
    left: anchor(center);
  }
}

#my-tooltip {
  position-fallback: --top-left-bottom;
}
```

NEW

```
#my-tooltip {
  inset-area: top;
  position-try-options: flip-start, flip-block;
}
```

Update 04

Animating fallback position

Computed(ish) instead of used value timing

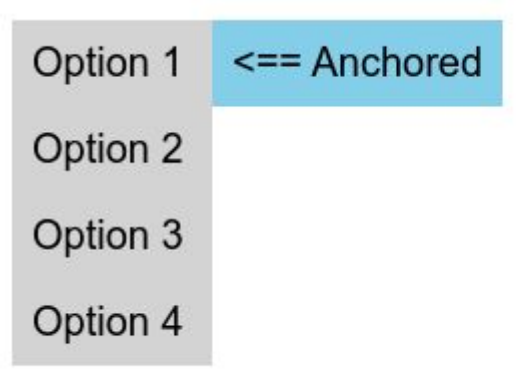
Animating

Properties affected by container queries, and cq units themselves, operate as "computed value"-ish.

Behavior is not yet specified, but Chrome and WebKit, at least, both end up with similar results - the result of the container changing can trigger computed-value changes, which kick off transitions. We're provisionally calling this "style interleaving".

Plan (currently sketched in the spec) is to do the same with the allowed `@position-try` properties.

This allows elements to smoothly animate when their anchor changes, when their alignment changes, etc, none of which are possible if you transition the individual properties based on their "normal" computed value.



Update 05
Scoping anchor names

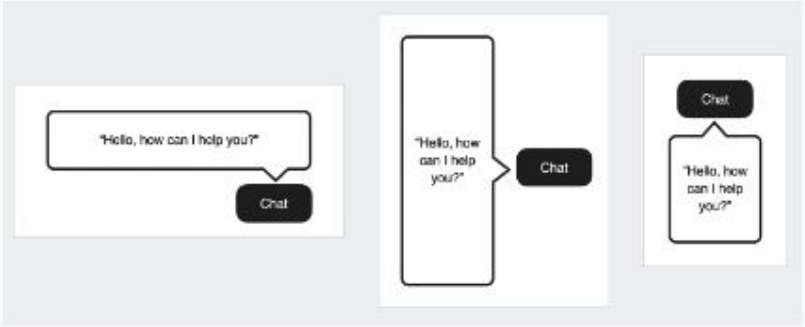
Style containment already scopes anchor names. `anchor-scope` property isn't in the spec yet, but I plan to add it soon, to trigger this functionality more intentionally.

Related problem, tho, with shadow DOM. Strong requests from internal teams (and we believe reflect general author need) to somehow let an element position against an anchor hidden away in shadow DOM. Need more research on this.

Update 06
Tether

(anchor-level-2)

::tether [css-anchor-position-2]



When an element is anchor-positioned with a valid default anchor, it also creates two pseudo elements with the following tree structure:

CB
```
+- anchor-positioned element
+- ::tether-container
   +- ::tether
```

Where `::tether-container` is an absolutely positioned box inserted as a sibling after the anchor-positioned element, and its insets are set by the UA style to match the “tether region” between the element and its anchor. The UA style and the box’s layout is similar to this demo.
`::tether` is an absolutely positioned child box of `::tether-container` and can be styled arbitrarily.

The main use case is to create an arrow that points to the default anchor element.

Last year, we resolved to keep it in the level 1 spec, punting if eventually necessary.

I propose more strongly that we go ahead and punt it, despite its desirability.

Non-trivial versions of the feature will just require fundamentally new (non positioning-related) functionality, like border shaping. I'm not working on that in the near future, and if nobody else is committing to it, I don't want to keep an aspirational section in the spec.

Questions?