ON AUTO SIZES IN GRID LAYOUT

HIDDE DE VRIES, TALK.CSS, 2-10-2020

Hidde de Vries

Freelance front-end developer, accessibility specialistfrom Rotterdam, Netherlands.

hiddedevries.nl What I do About me Blog Talks Contact

Blog

Posts about HTML, CSS, JavaScript, accessibility and browsers from the perspective of a curious front-end developer.

BEST OF

Sorted by a unique and patented algorithm, these posts may interest you:

- More accessible defaults, please!
- Three ways to build Crouwel's Hiroshima poster in CSS
- How accessibility trees inform assistive tech
- Heading structures are tables of contents
- Making password managers play ball with your login form

RSS (full)
RSS (summaries)

When there is no content between headings *on 05 September 2020*

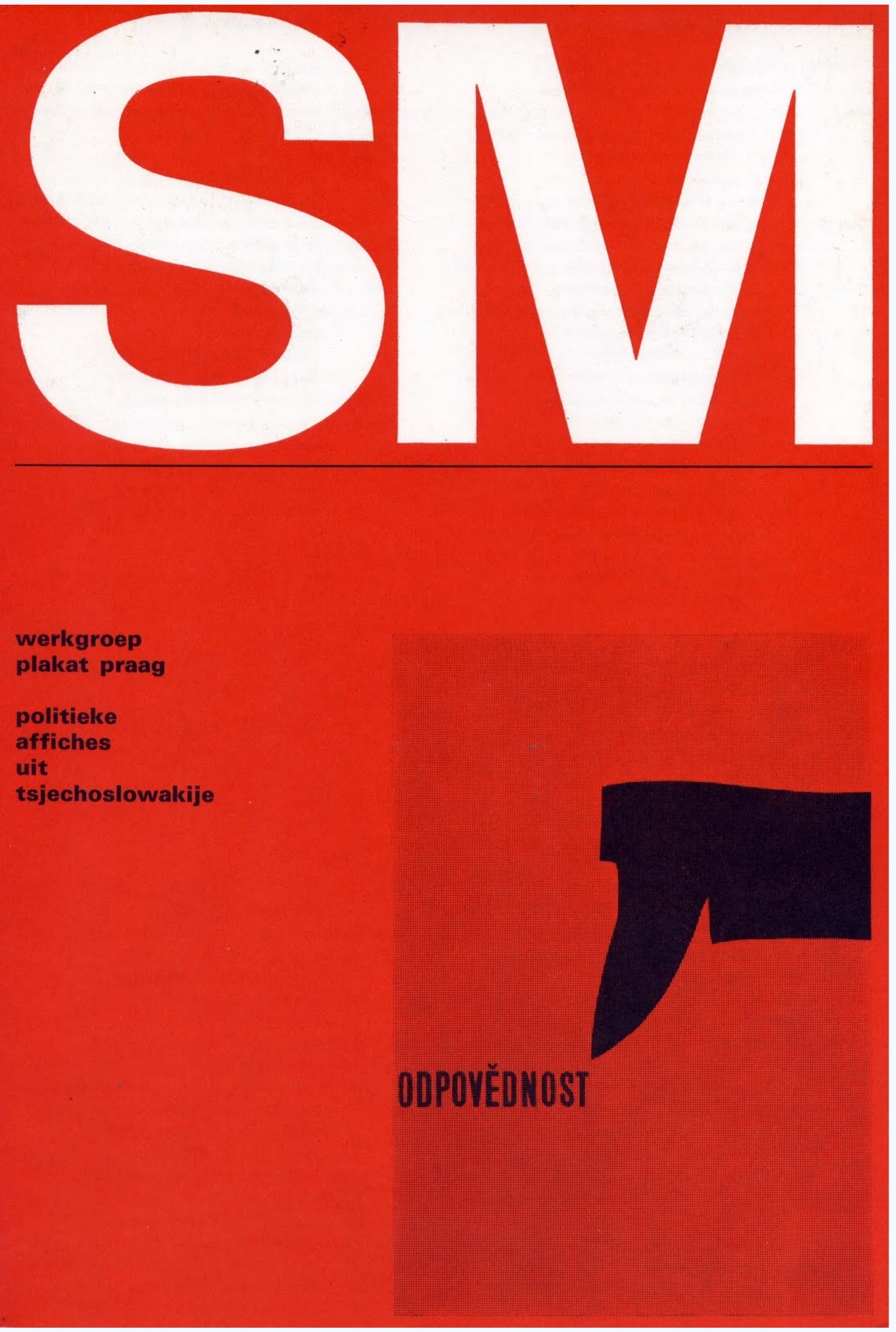
Why having no content between headings can be problematic.

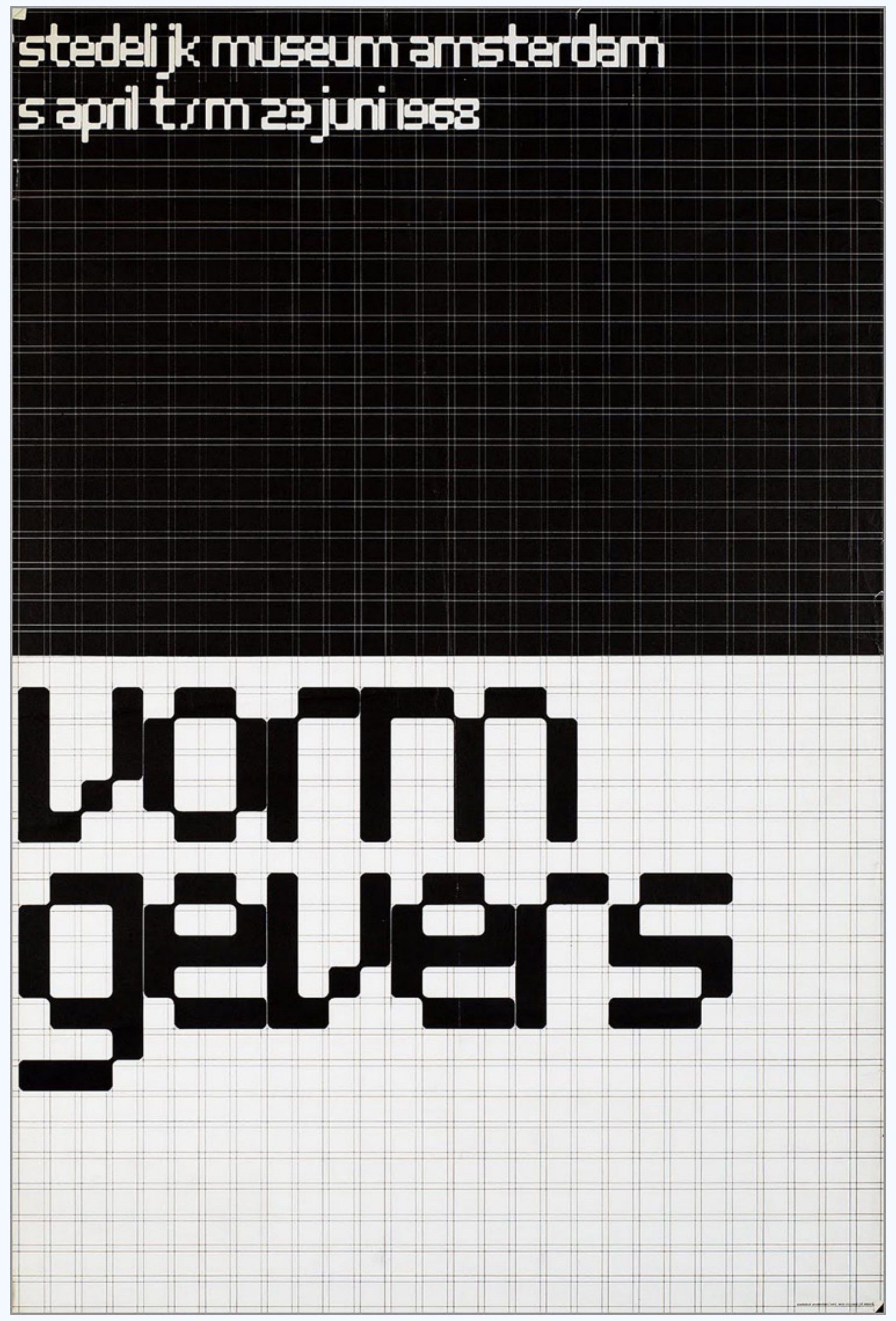
Equality, a reading list (2) *on 16 July 2020*

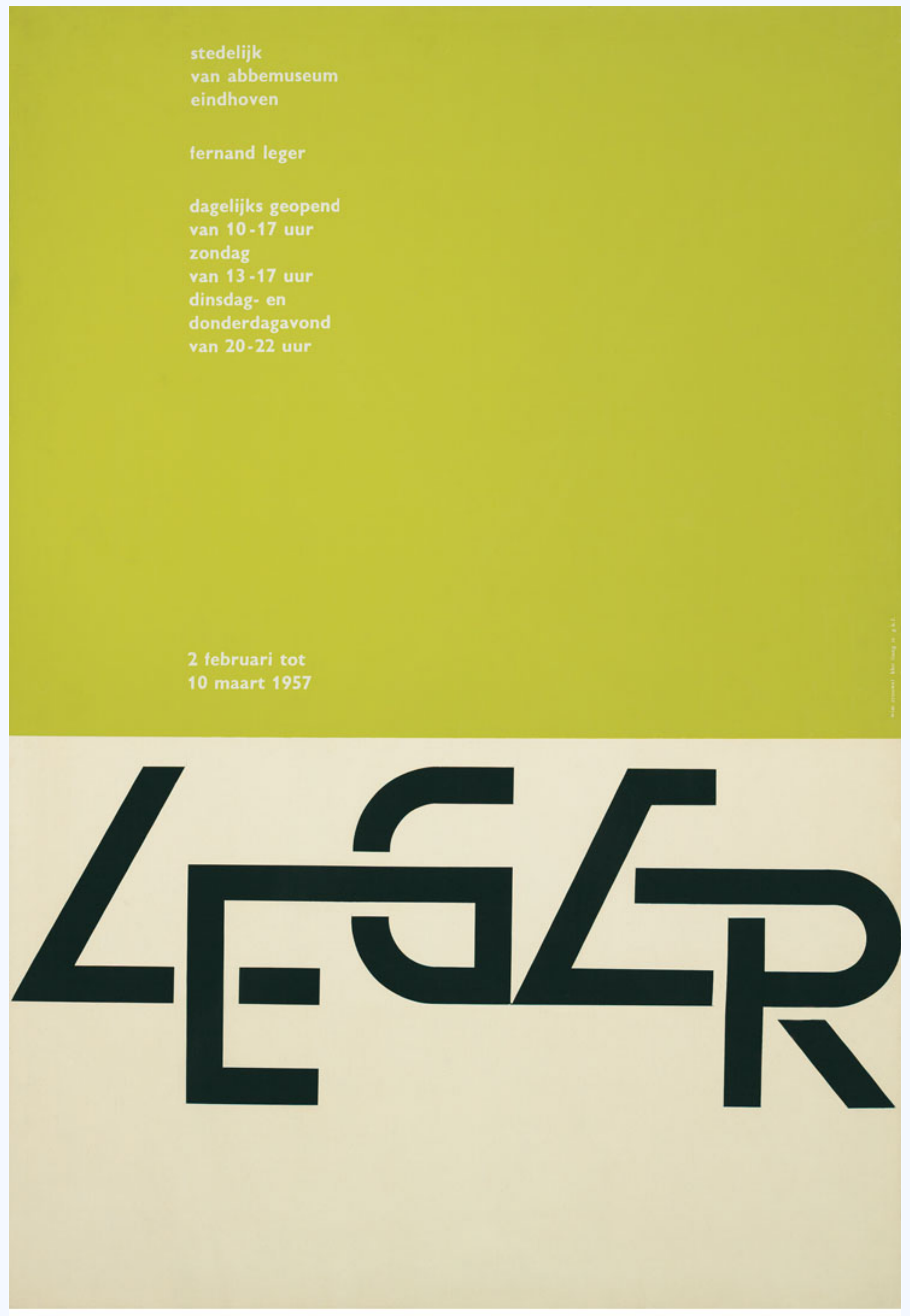
A list of books about inequality and race

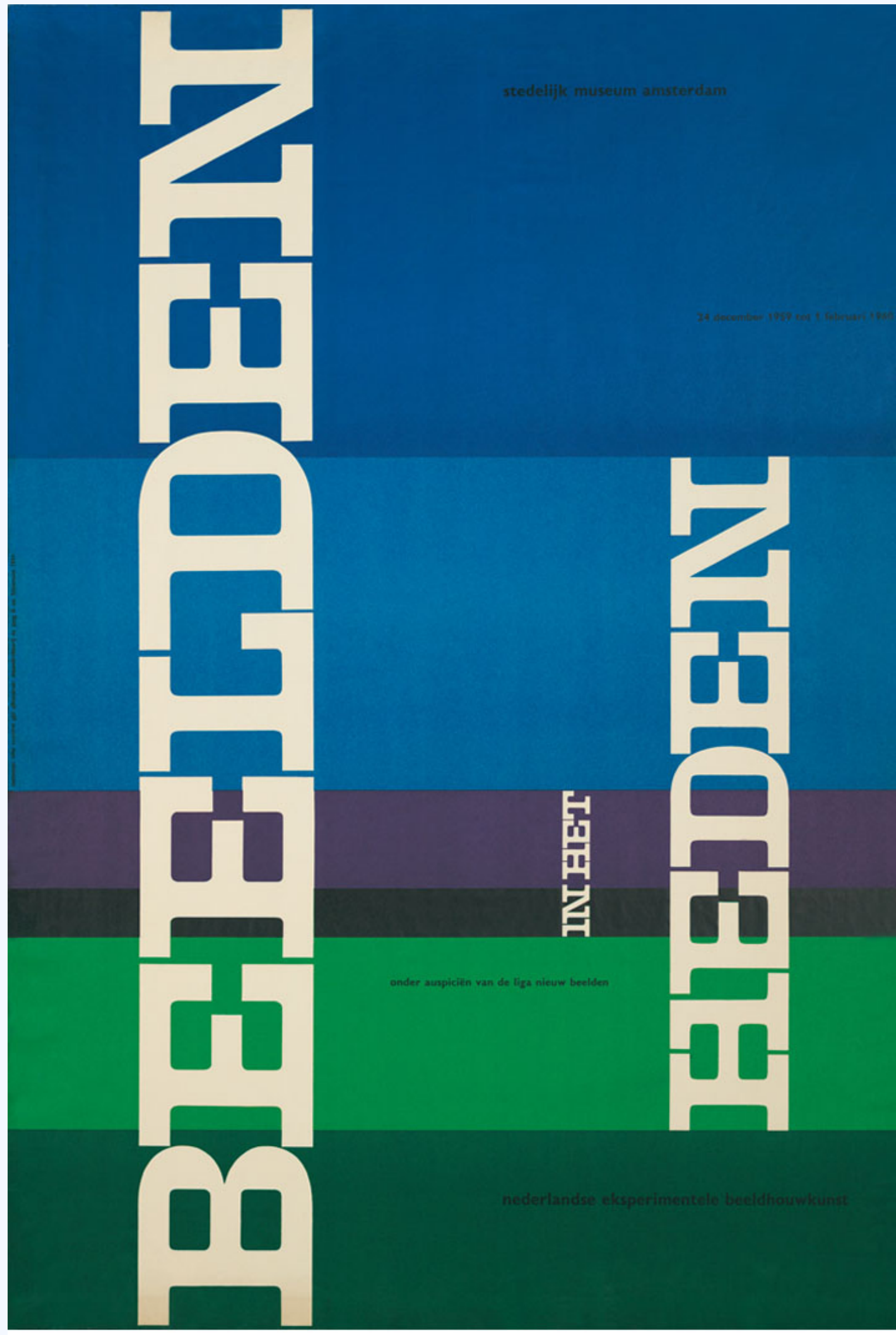
@hdv ◇ hidde.blog

FIXED CANVAS

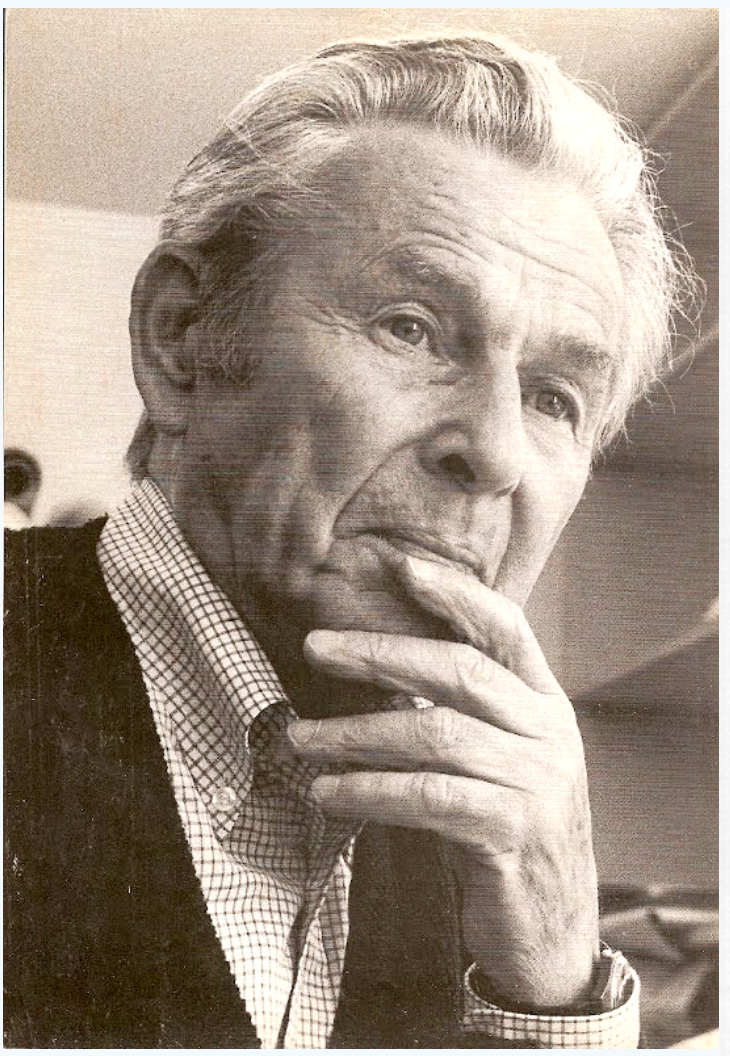


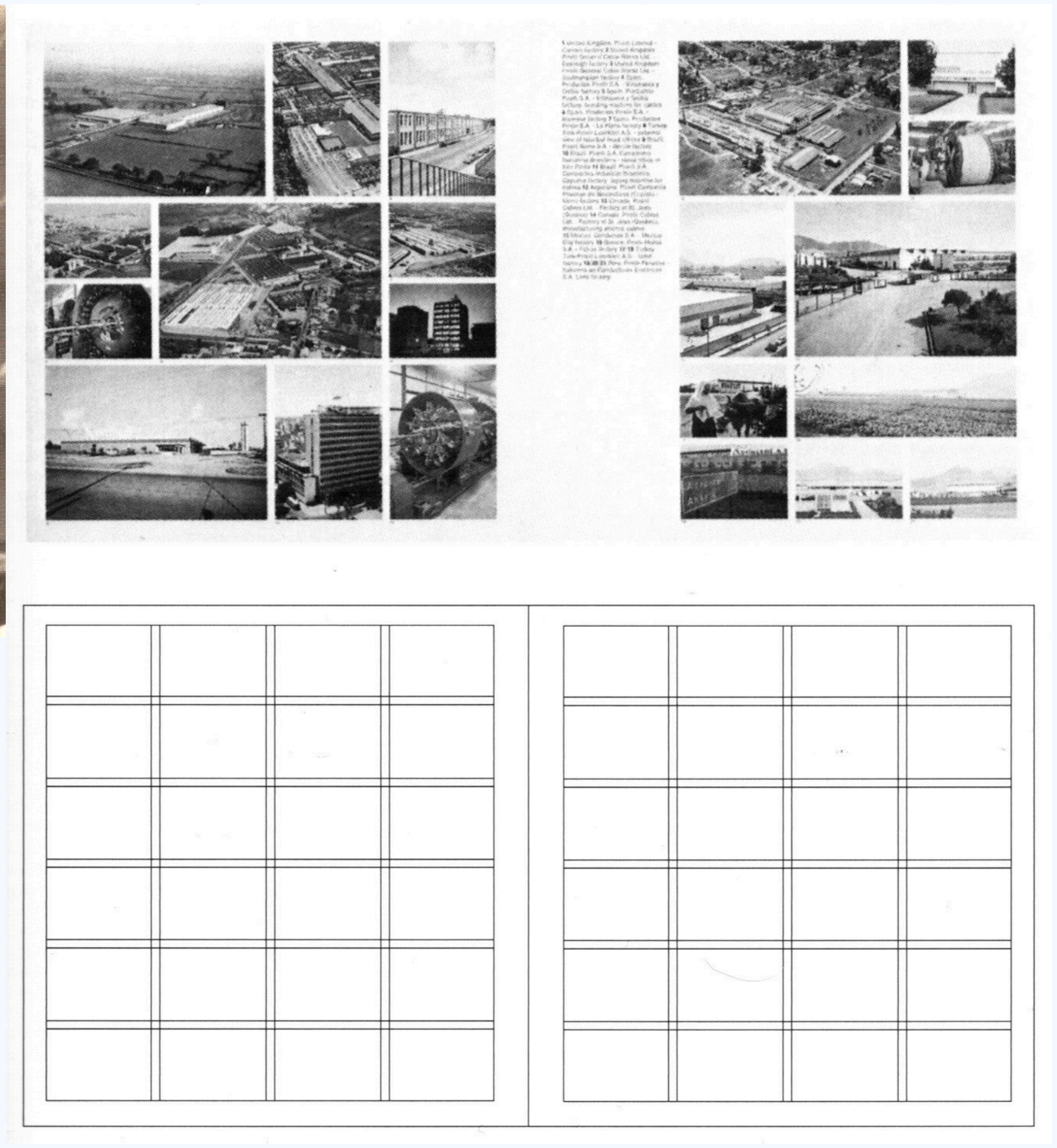





BEELDEN
stedelijk museum amsterdam
24 december 1959 tot 1 februari 1960
IN HET
HEDEN
onder auspiciën van de liga nieuw beelden
nederlandse eksperimentele beeldhouwkunst

Posters by Wim Crouwel





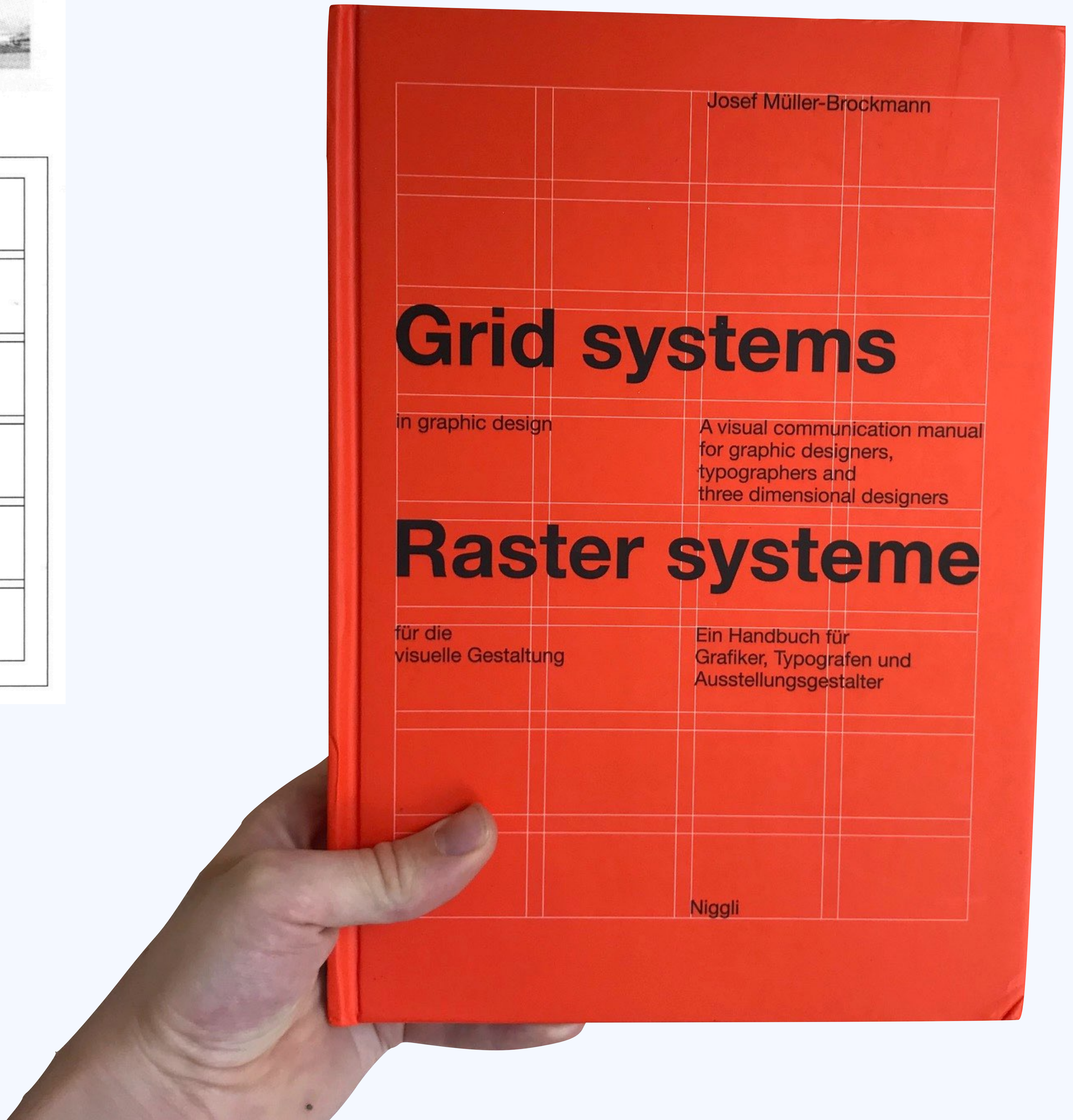
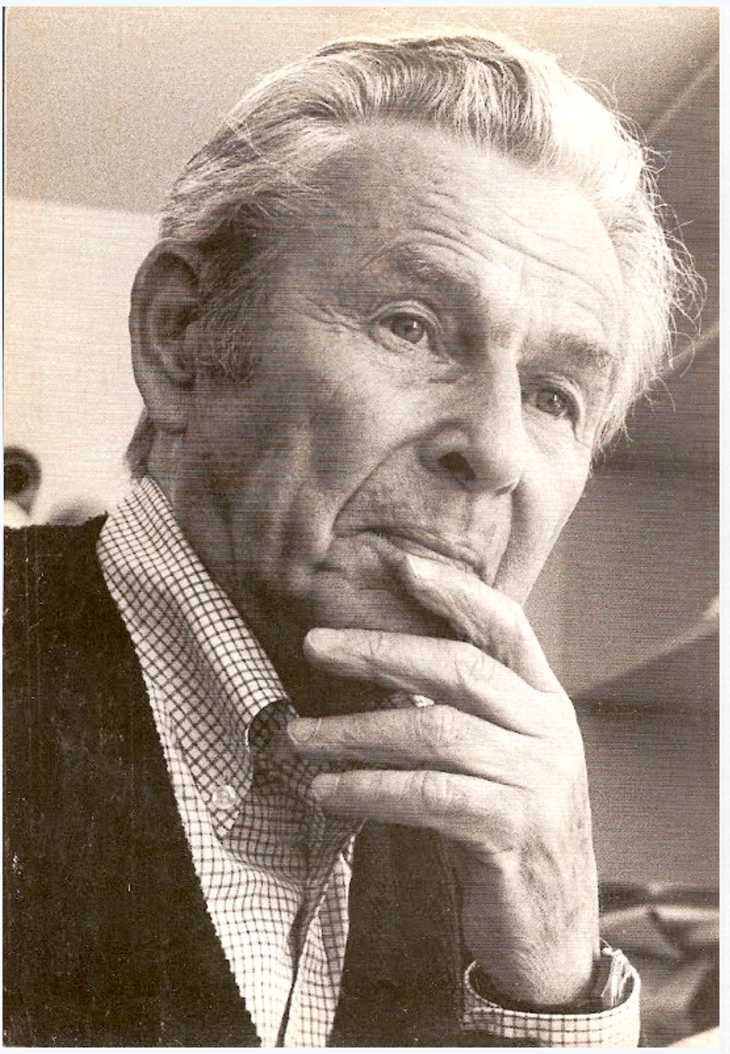
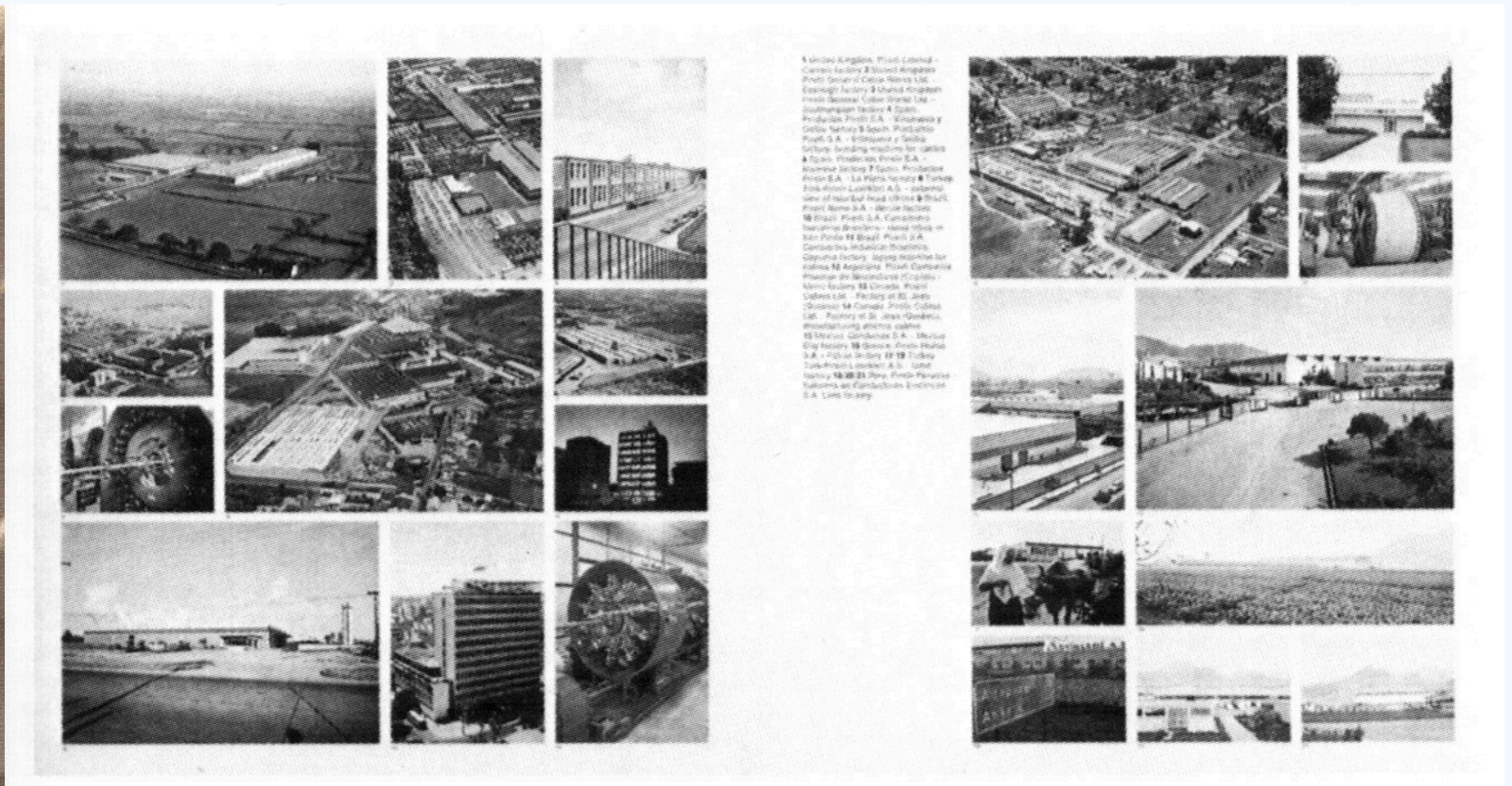
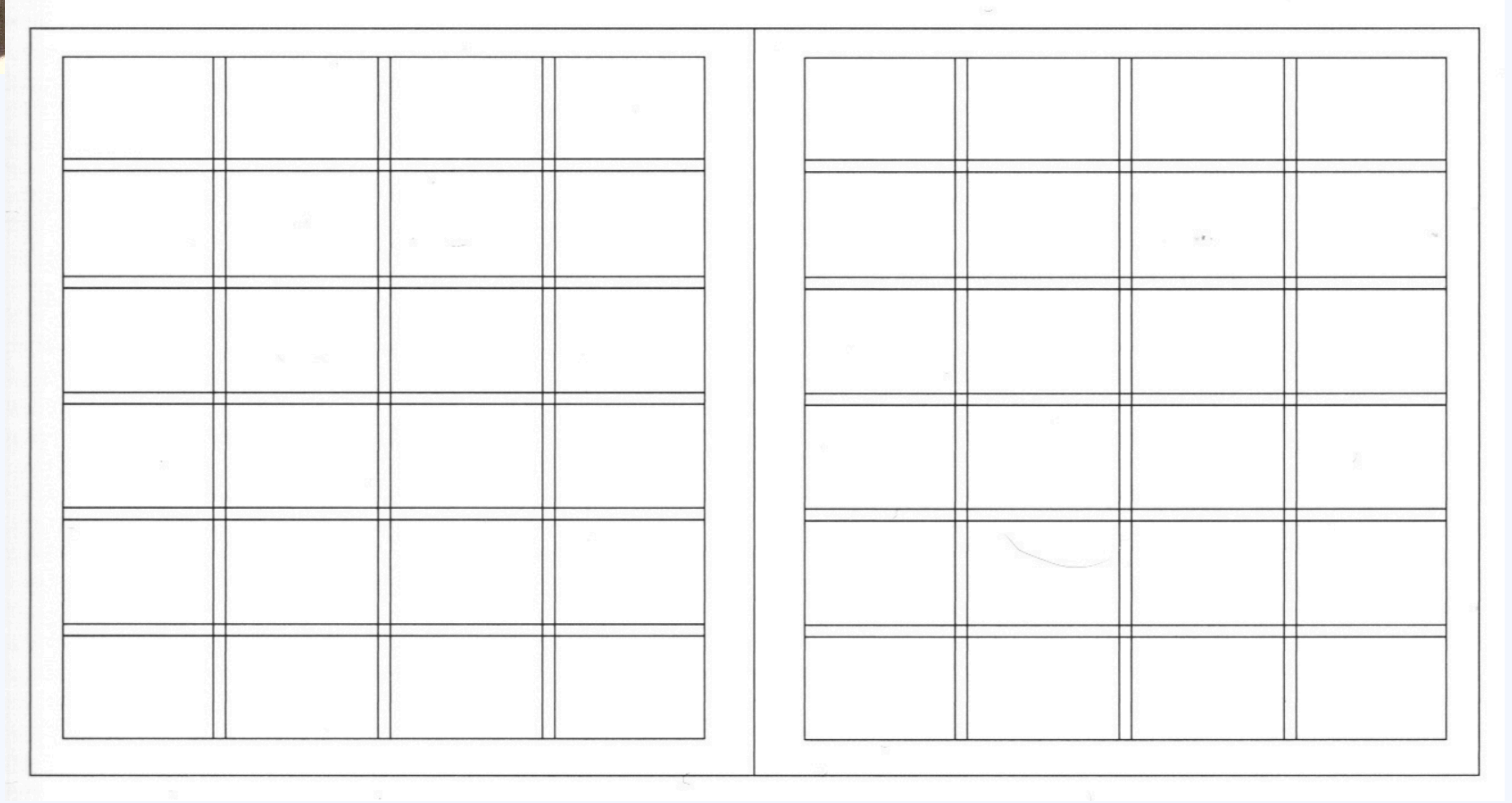

Josef Müller-Brockmann
Grid systems
in graphic design
A visual communication manual for graphic designers, typographers and three dimensional designers
Raster systeme
für die visuelle Gestaltung
Ein Handbuch für Grafiker, Typografen und Ausstellungsgestalter
Niggli

Photo JMB: MoMA

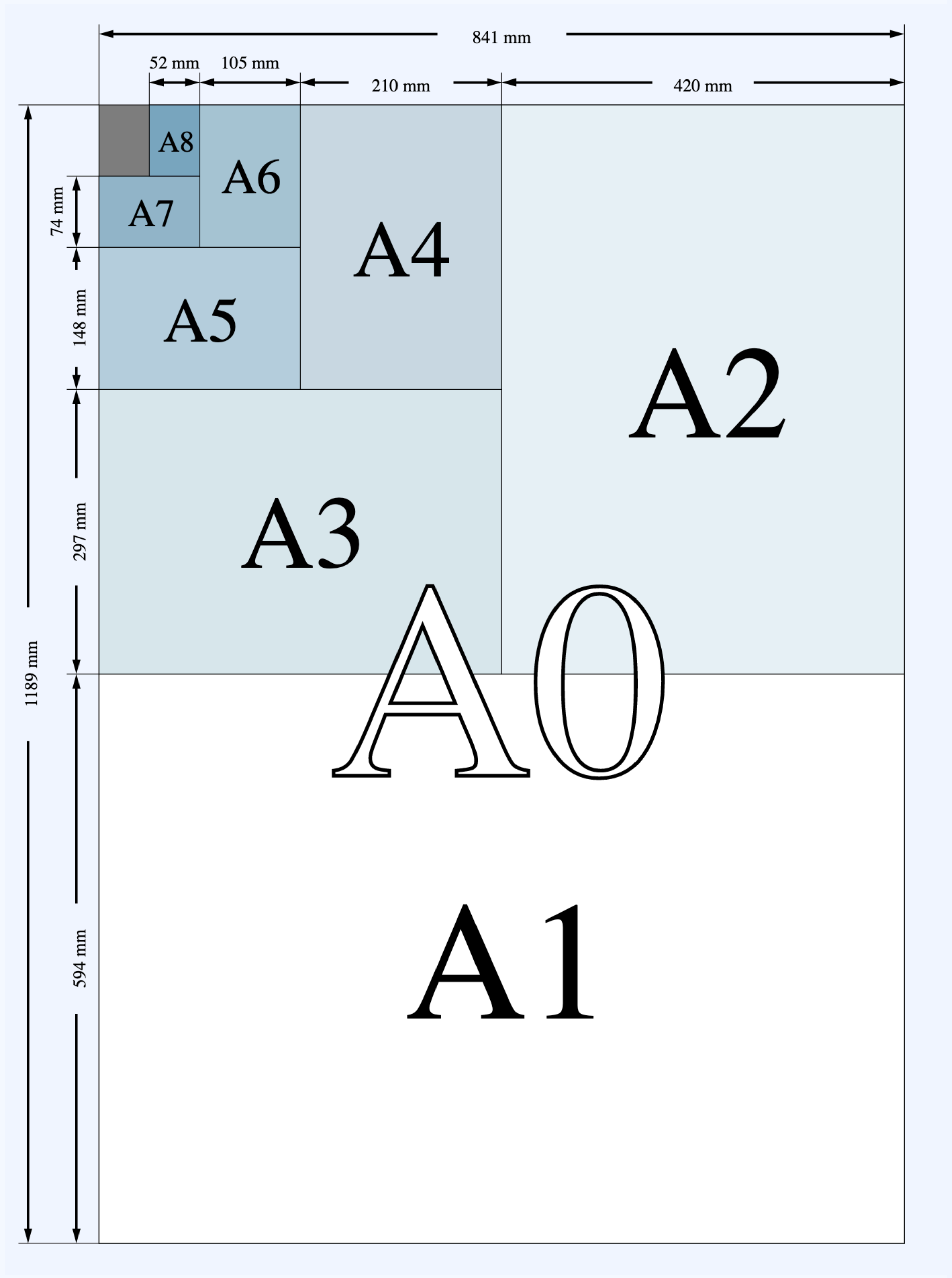


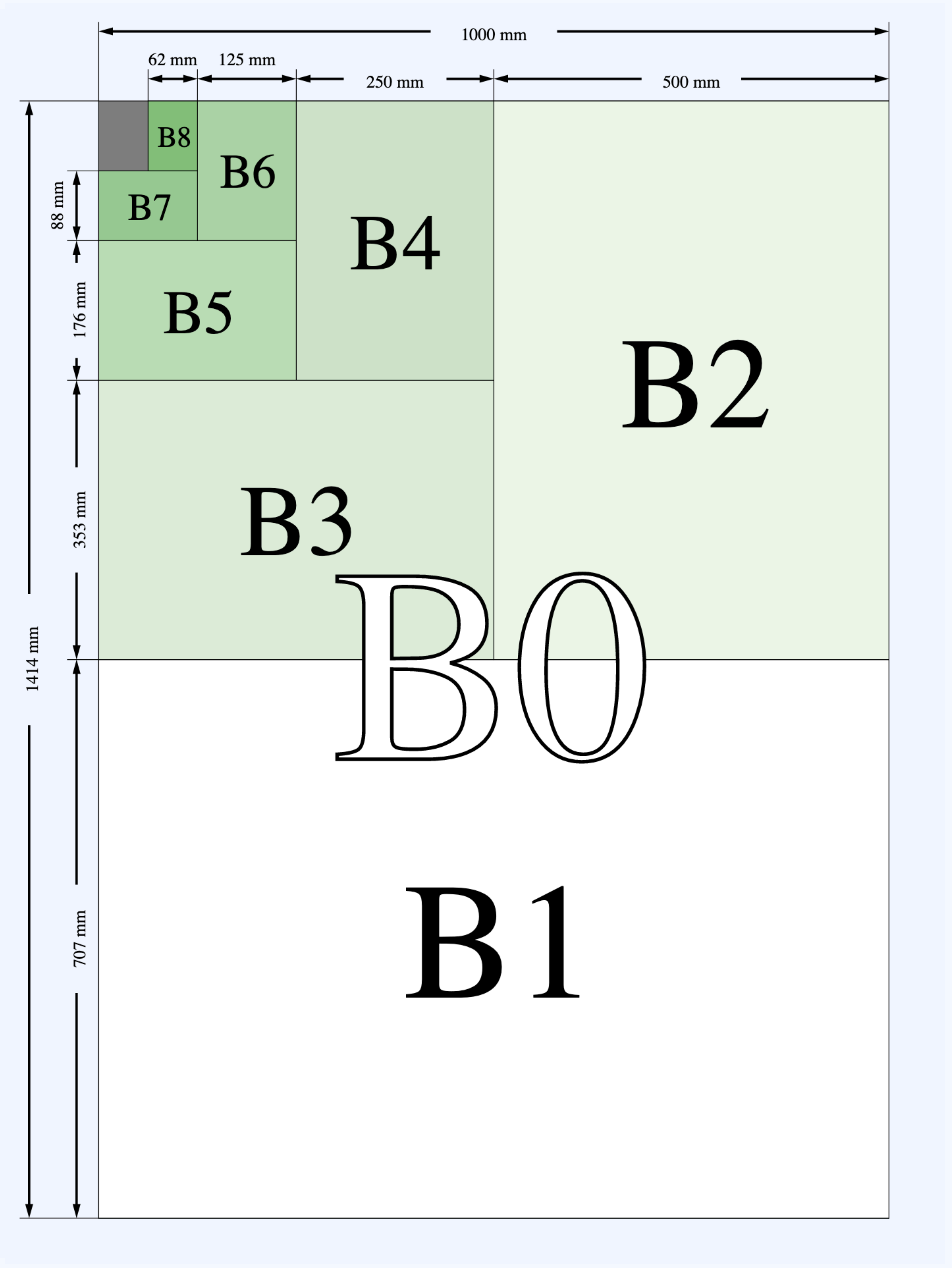




Josef Müller-Brockmann
Grid systems
in graphic design
A visual communication manual for graphic designers, typographers and three dimensional designers
Raster systeme
für die visuelle Gestaltung
Ein Handbuch für Grafiker, Typografen und Ausstellungsgestalter
Niggli

Photo JMB: MoMA





From: Wikipedia

Digital posters

From: Emerce

Digital posters

"More relevant, more local outreach to our customers"

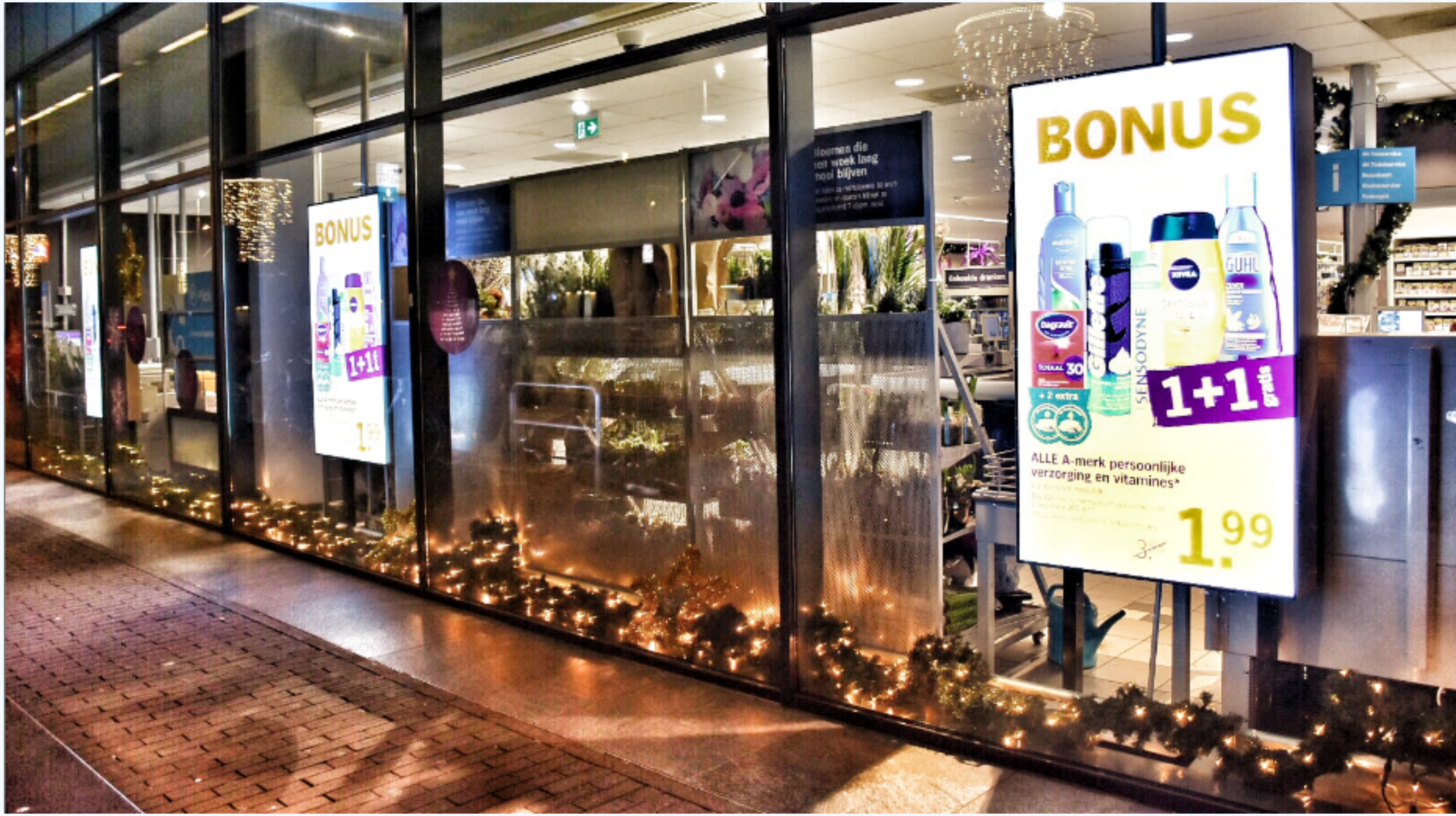
From: Emerce

Digital posters

COULD! USE! CSS!

From: Emerce

Digital posters



From: Emerce

The web

From: CERN (home.cern)

The web is everywhere,
there are infinite canvases.

The web is everywhere,
there are infinite canvases.

And lots of languages.

The web is everywhere,
there are infinite canvases.

And lots of languages. And
a lot of writing systems.

The web is everywhere,
there are infinite canvases.

And lots of languages. And
a lot of writing systems.

CSS is here to help!

Fixed sizes

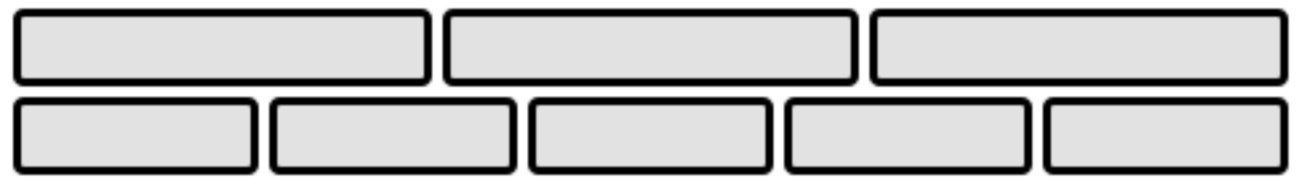
"When we define auto in CSS, we want it give give reasonable results, avoid dataloss/overflow and be a good default to build on"

– **Fantasai**, in her talk "Defining auto"

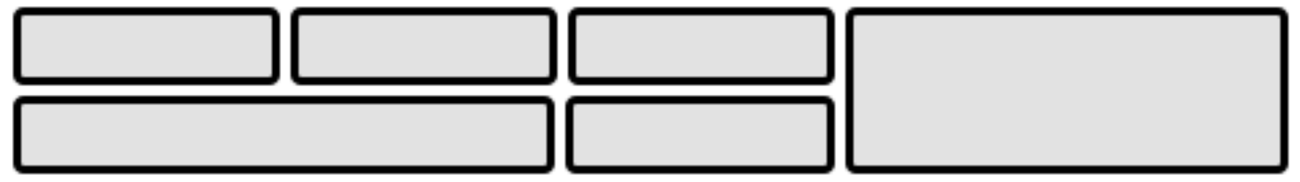
TERMINOLOGY

Block vs inline

Grid Layout is a layout model for CSS that has powerful abilities to control the sizing and positioning of boxes and their contents. Unlike Flexible Box Layout, which is single-axis–oriented, Grid Layout is optimized for 2-dimensional layouts: those in which alignment of content is desired in both dimensions.



Representative Flex Layout Example



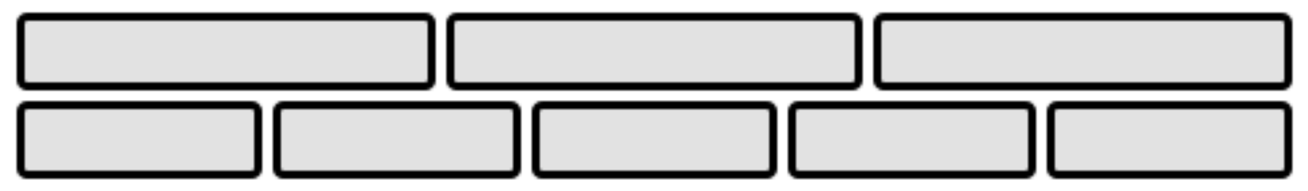
Representative Grid Layout Example

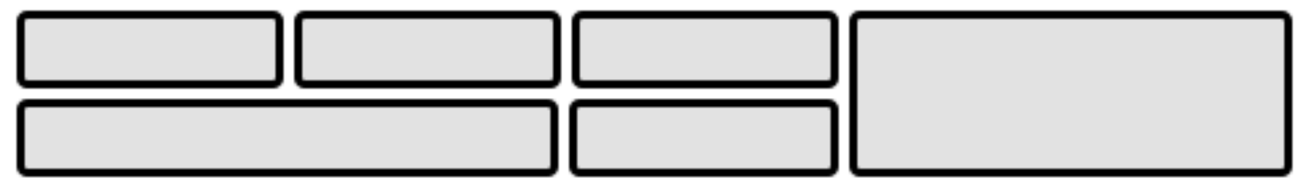
In addition, due to its ability to explicitly position items in the grid, Grid Layout allows dramatic transformations in visual layout structure without requiring corresponding markup changes. By combining media queries with the CSS properties that control layout of the grid container and its children, authors can adapt their layout to changes in device form factors, orientation, and available space, while preserving a more ideal semantic structuring of their content across presentations.

Although many layouts can be expressed with either Grid or Flexbox, they each have their specialties. Grid enforces 2-dimensional alignment, uses a top-down approach to layout, allows explicit overlapping of items, and has more powerful spanning capabilities. Flexbox focuses on space distribution within an axis, uses a simpler bottom-up approach to layout, can use a content-size–based line-wrapping system to control its secondary axis, and relies on the underlying markup

Block vs inline

Grid Layout is a layout model for CSS that has powerful abilities to control the sizing and positioning of boxes and their contents. Unlike Flexible Box Layout, which is single-axis–oriented, Grid Layout is optimized for 2-dimensional layouts: those in which alignment of content is desired in both dimensions.


Representative Flex Layout Example

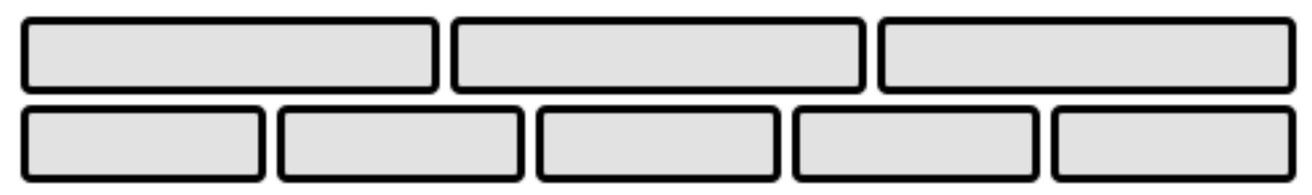

Representative Grid Layout Example

In addition, due to its ability to explicitly position items in the grid, Grid Layout allows dramatic transformations in visual layout structure without requiring corresponding markup changes. By combining media queries with the CSS properties that control layout of the grid container and its children, authors can adapt their layout to changes in device form factors, orientation, and available space, while preserving a more ideal semantic structuring of their content across presentations.

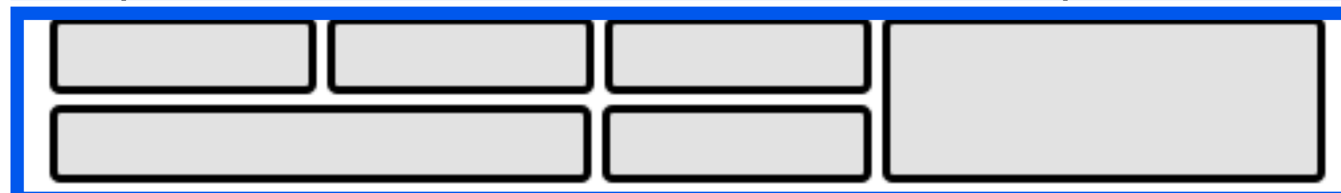
Although many layouts can be expressed with either Grid or Flexbox, they each have their specialties. Grid enforces 2-dimensional alignment, uses a top-down approach to layout, allows explicit overlapping of items, and has more powerful spanning capabilities. Flexbox focuses on space distribution within an axis, uses a simpler bottom-up approach to layout, can use a content-size–based line-wrapping system to control its secondary axis, and relies on the underlying markup

Block vs inline

Grid Layout is a layout model for CSS that has powerful abilities to control the sizing and positioning of boxes and their contents. Unlike Flexible Box Layout, which is single-axis–oriented, Grid Layout is optimized for 2-dimensional layouts: those in which alignment of content is desired in both dimensions.



Representative Flex Layout Example

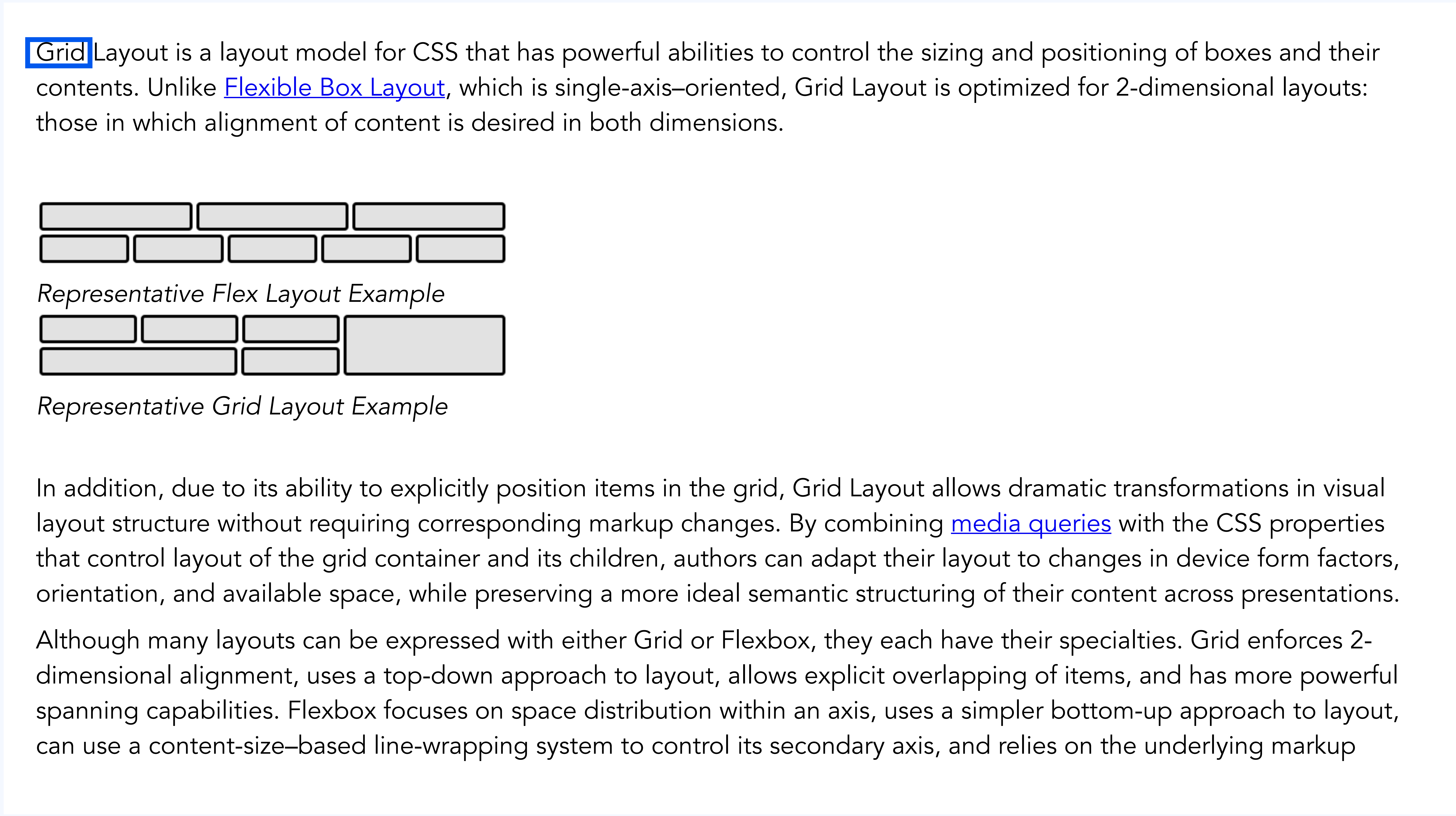


Representative Grid Layout Example

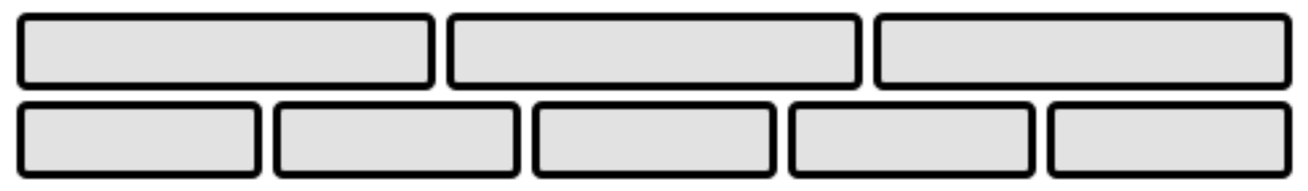
In addition, due to its ability to explicitly position items in the grid, Grid Layout allows dramatic transformations in visual layout structure without requiring corresponding markup changes. By combining media queries with the CSS properties that control layout of the grid container and its children, authors can adapt their layout to changes in device form factors, orientation, and available space, while preserving a more ideal semantic structuring of their content across presentations.

Although many layouts can be expressed with either Grid or Flexbox, they each have their specialties. Grid enforces 2-dimensional alignment, uses a top-down approach to layout, allows explicit overlapping of items, and has more powerful spanning capabilities. Flexbox focuses on space distribution within an axis, uses a simpler bottom-up approach to layout, can use a content-size–based line-wrapping system to control its secondary axis, and relies on the underlying markup

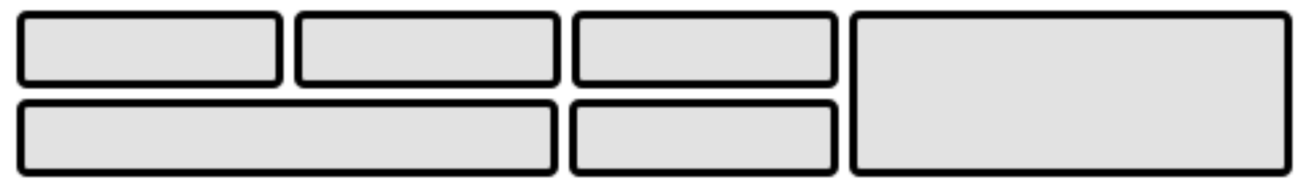
Block vs inline



Grid Layout is a layout model for CSS that has powerful abilities to control the sizing and positioning of boxes and their contents. Unlike Flexible Box Layout, which is single-axis–oriented, Grid Layout is optimized for 2-dimensional layouts: those in which alignment of content is desired in both dimensions.



Representative Flex Layout Example

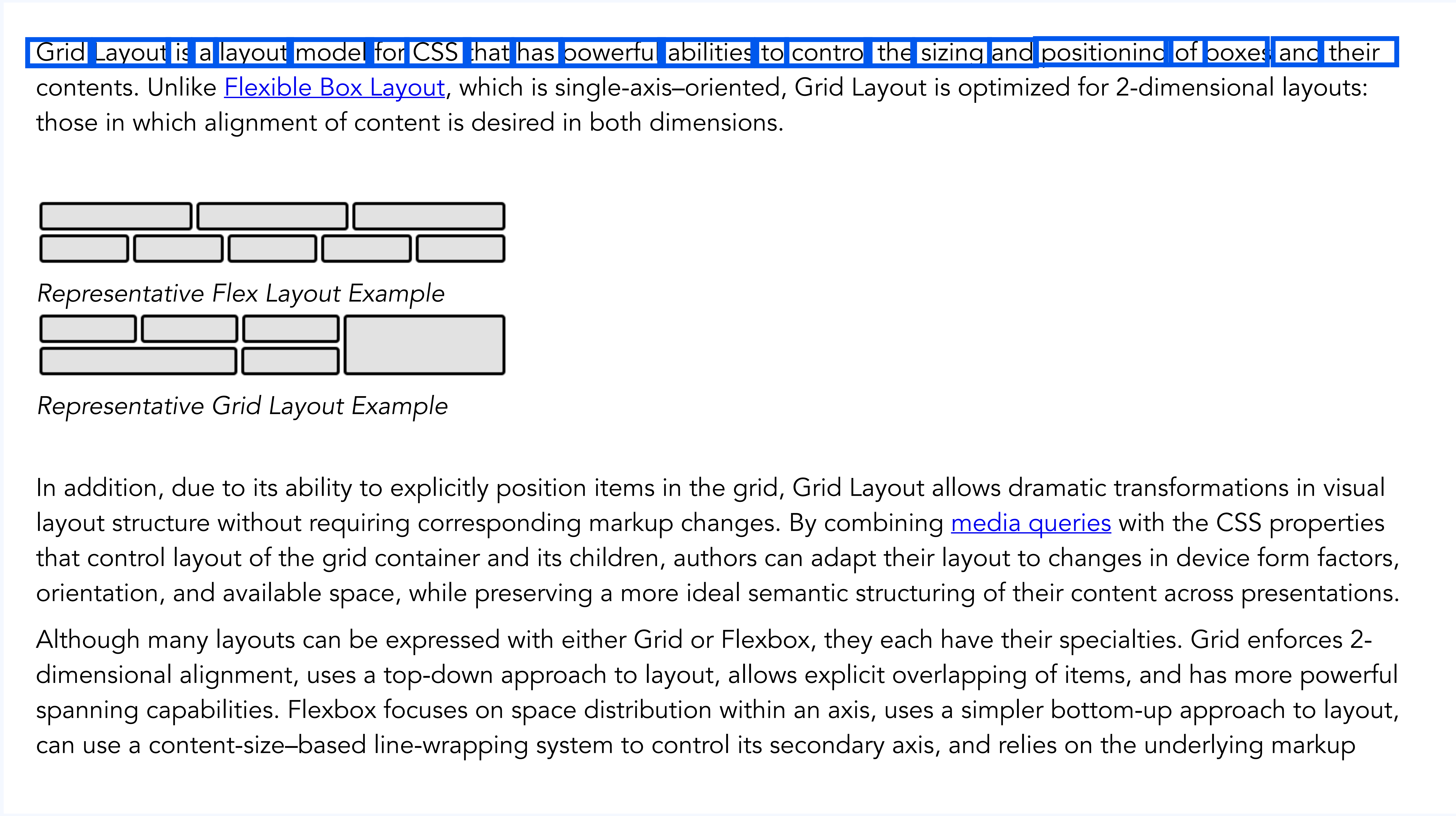


Representative Grid Layout Example

In addition, due to its ability to explicitly position items in the grid, Grid Layout allows dramatic transformations in visual layout structure without requiring corresponding markup changes. By combining media queries with the CSS properties that control layout of the grid container and its children, authors can adapt their layout to changes in device form factors, orientation, and available space, while preserving a more ideal semantic structuring of their content across presentations.

Although many layouts can be expressed with either Grid or Flexbox, they each have their specialties. Grid enforces 2-dimensional alignment, uses a top-down approach to layout, allows explicit overlapping of items, and has more powerful spanning capabilities. Flexbox focuses on space distribution within an axis, uses a simpler bottom-up approach to layout, can use a content-size–based line-wrapping system to control its secondary axis, and relies on the underlying markup

Block vs inline



Grid Layout is a layout model for CSS that has powerful abilities to control the sizing and positioning of boxes and their contents. Unlike Flexible Box Layout, which is single-axis–oriented, Grid Layout is optimized for 2-dimensional layouts: those in which alignment of content is desired in both dimensions.

Representative Flex Layout Example

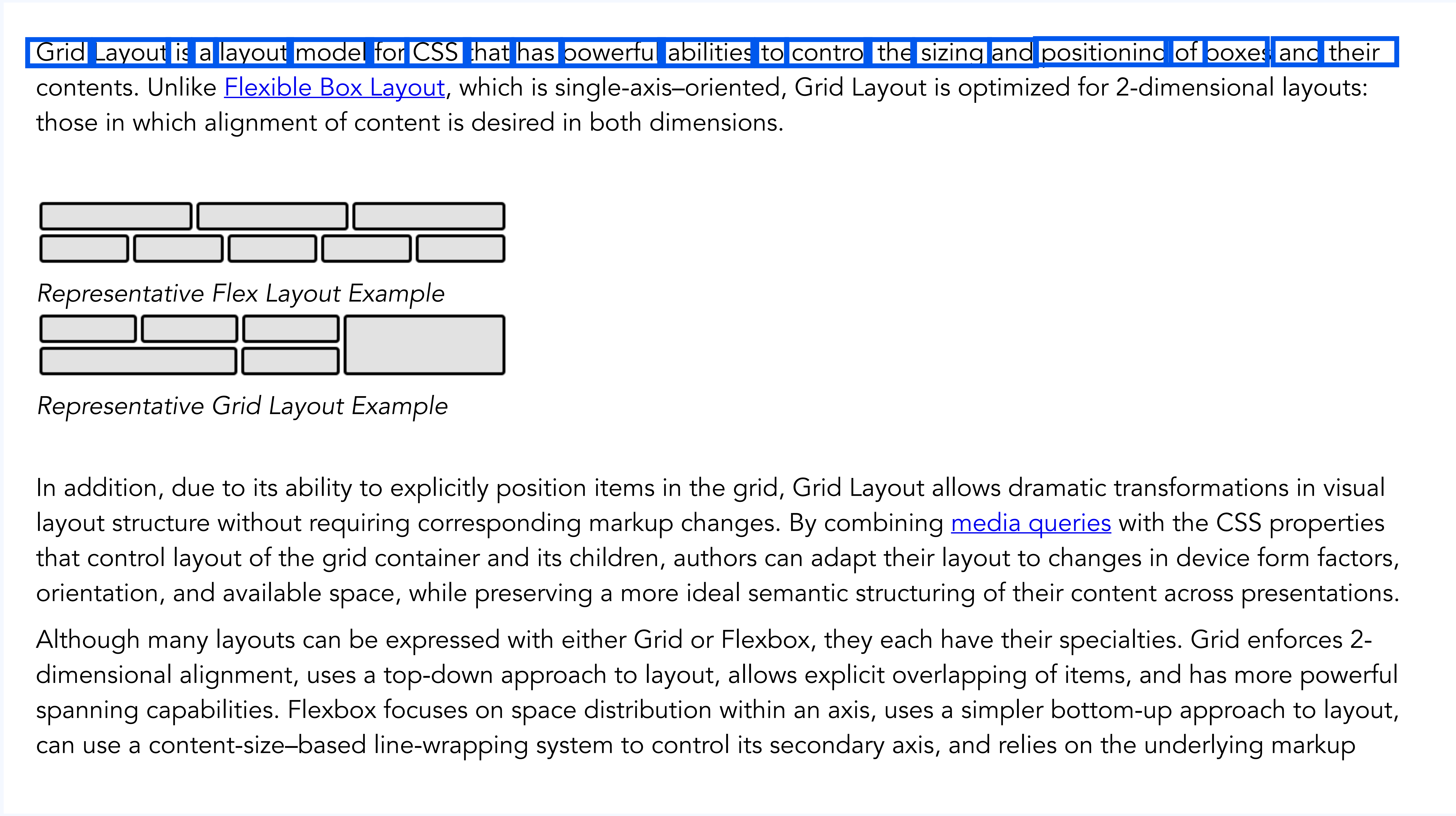
Representative Grid Layout Example

In addition, due to its ability to explicitly position items in the grid, Grid Layout allows dramatic transformations in visual layout structure without requiring corresponding markup changes. By combining media queries with the CSS properties that control layout of the grid container and its children, authors can adapt their layout to changes in device form factors, orientation, and available space, while preserving a more ideal semantic structuring of their content across presentations.

Although many layouts can be expressed with either Grid or Flexbox, they each have their specialties. Grid enforces 2-dimensional alignment, uses a top-down approach to layout, allows explicit overlapping of items, and has more powerful spanning capabilities. Flexbox focuses on space distribution within an axis, uses a simpler bottom-up approach to layout, can use a content-size–based line-wrapping system to control its secondary axis, and relies on the underlying markup

Block* vs *inline

(In left to right,
top to bottom layout)



Grid Layout is a layout model for CSS that has powerful abilities to control the sizing and positioning of boxes and their contents. Unlike Flexible Box Layout, which is single-axis–oriented, Grid Layout is optimized for 2-dimensional layouts: those in which alignment of content is desired in both dimensions.

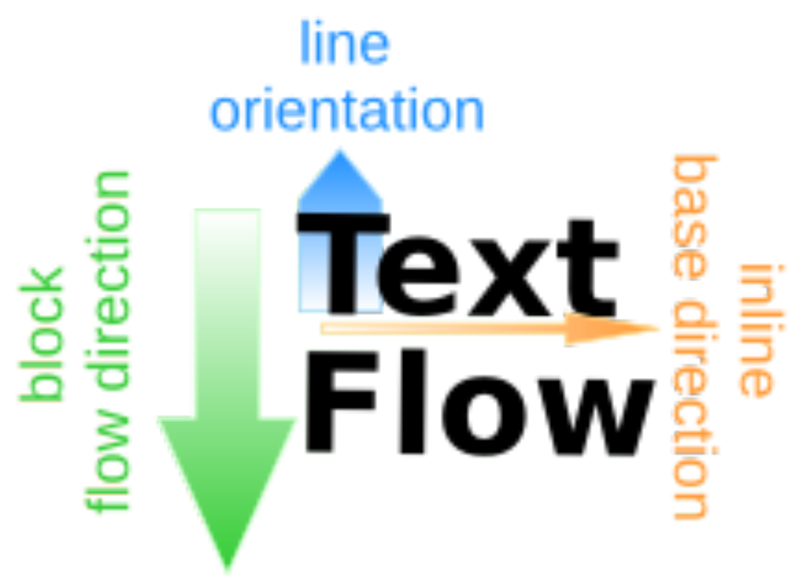
Representative Flex Layout Example

Representative Grid Layout Example

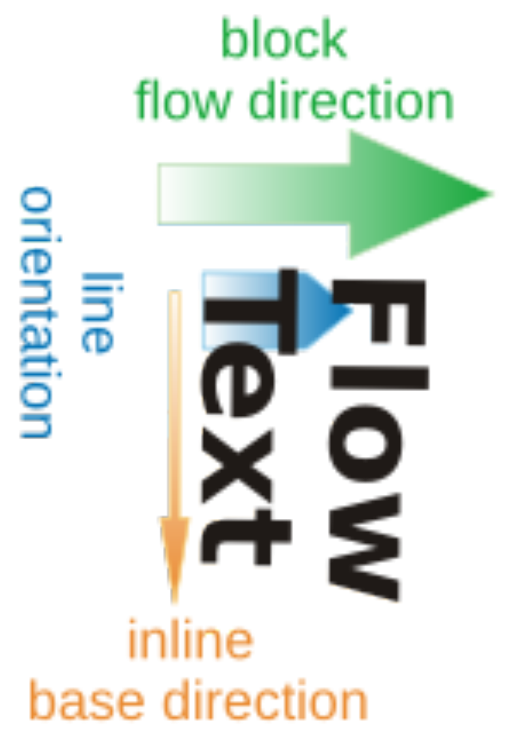
In addition, due to its ability to explicitly position items in the grid, Grid Layout allows dramatic transformations in visual layout structure without requiring corresponding markup changes. By combining media queries with the CSS properties that control layout of the grid container and its children, authors can adapt their layout to changes in device form factors, orientation, and available space, while preserving a more ideal semantic structuring of their content across presentations.

Although many layouts can be expressed with either Grid or Flexbox, they each have their specialties. Grid enforces 2-dimensional alignment, uses a top-down approach to layout, allows explicit overlapping of items, and has more powerful spanning capabilities. Flexbox focuses on space distribution within an axis, uses a simpler bottom-up approach to layout, can use a content-size–based line-wrapping system to control its secondary axis, and relies on the underlying markup

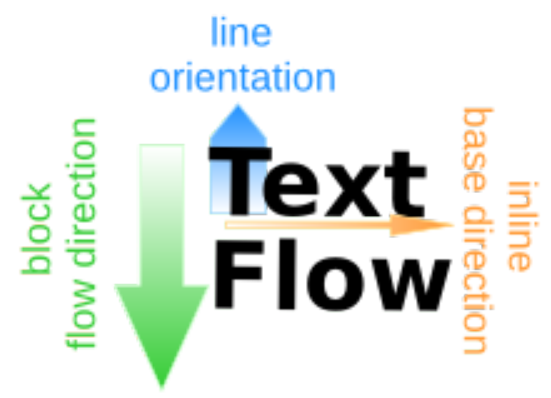
Writing modes



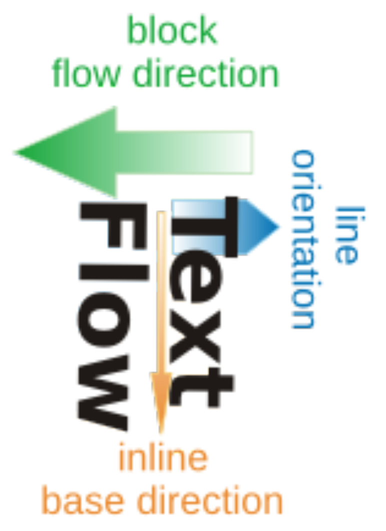
Latin-based



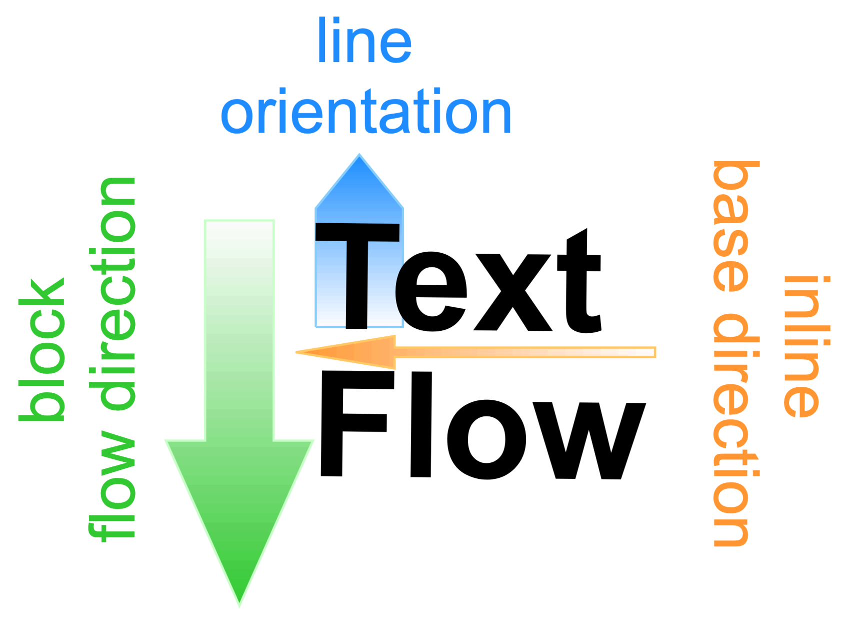
Mongolian-based



or

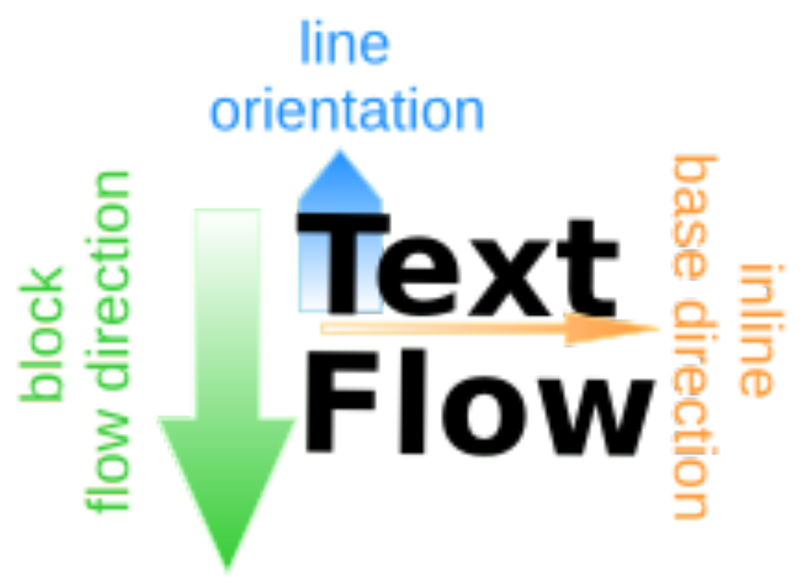


Han-based

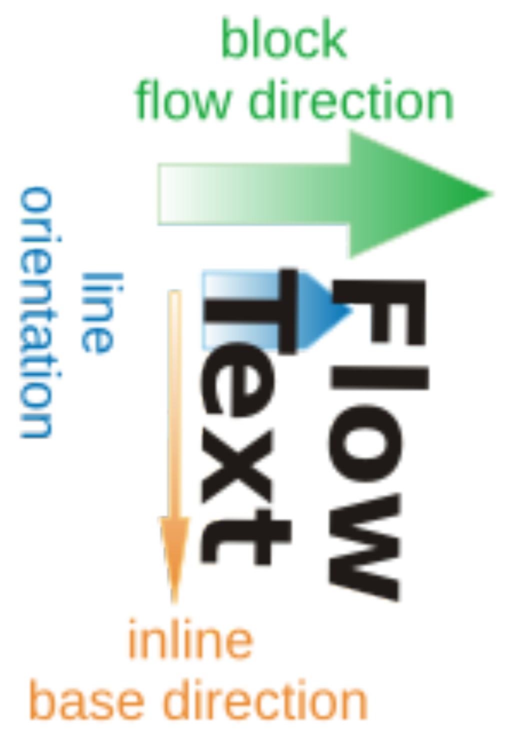


Arabic-based

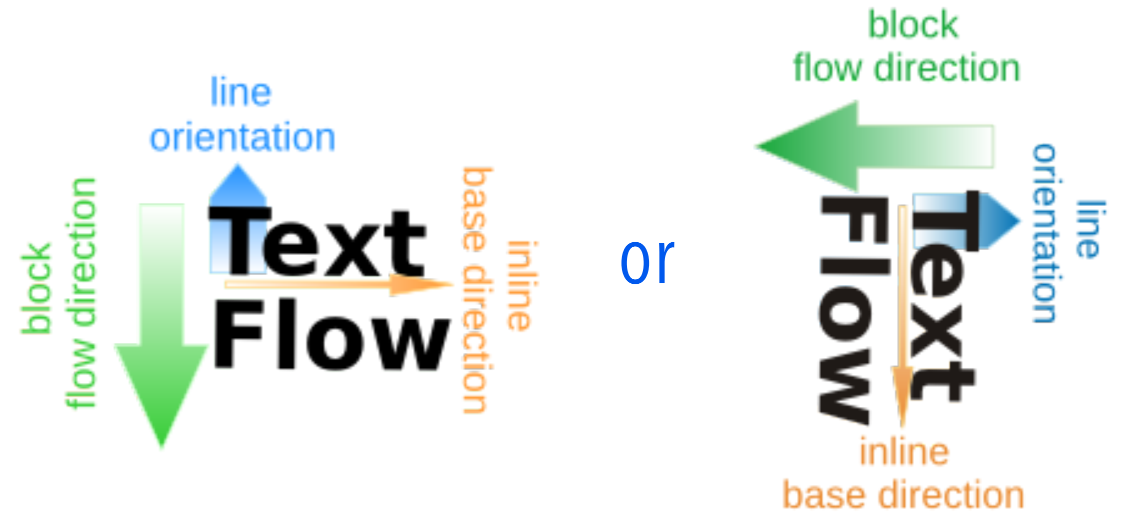
Writing modes



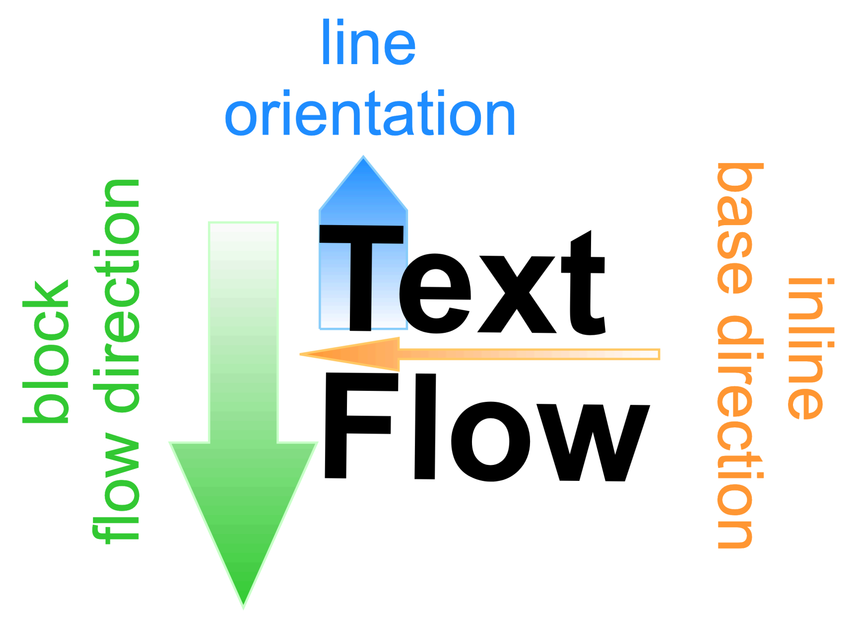
Latin-based



Mongolian-based

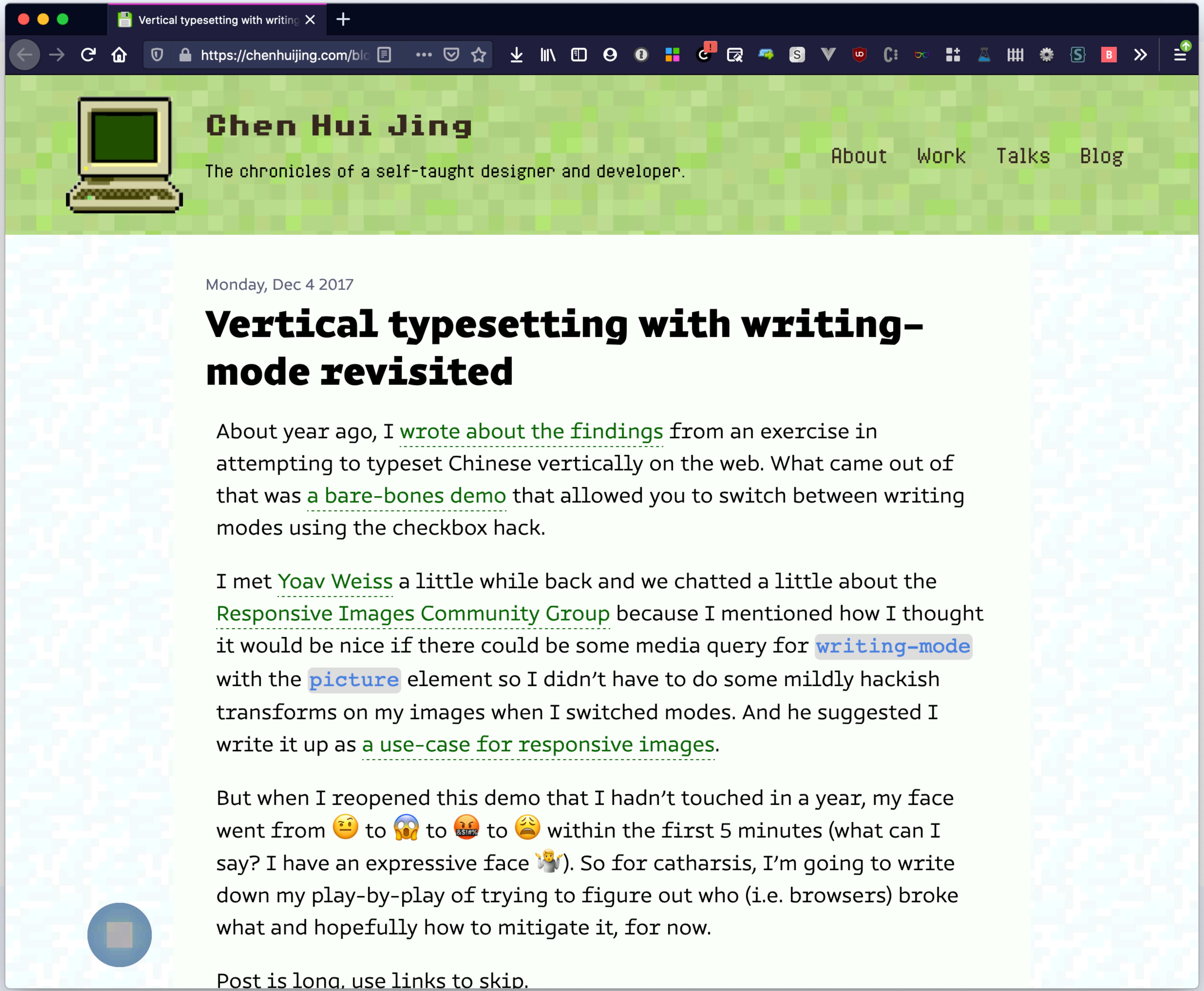


Han-based



Arabic-based

Vertical type on the web still tricky


Vertical typesetting with writing

https://chenhuijing.com/blo

Chen Hui Jing

The chronicles of a self-taught designer and developer.

About Work Talks Blog

Monday, Dec 4 2017

Vertical typesetting with writing-mode revisited

About year ago, I wrote about the findings from an exercise in attempting to typeset Chinese vertically on the web. What came out of that was a bare-bones demo that allowed you to switch between writing modes using the checkbox hack.

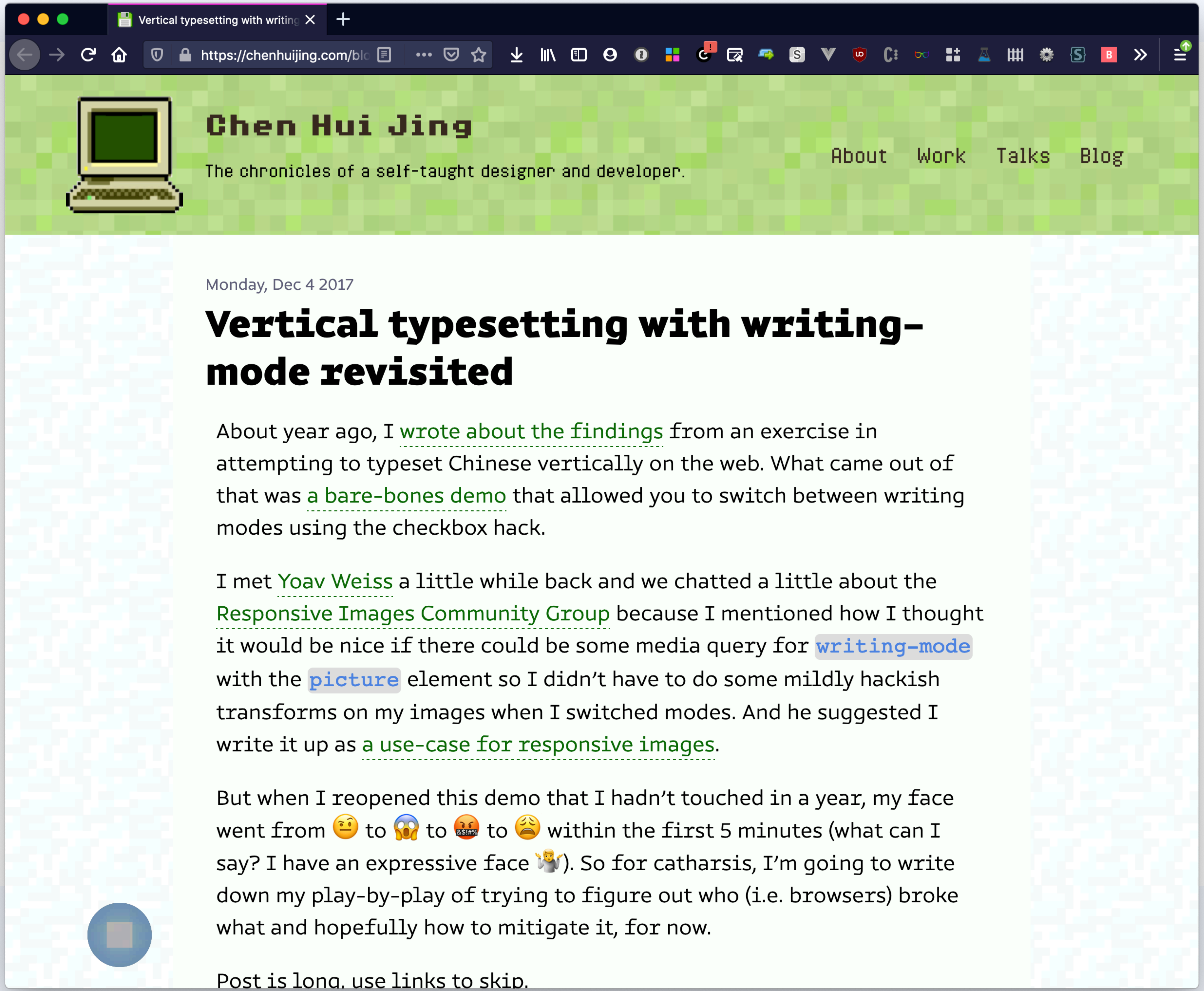
I met Yoav Weiss a little while back and we chatted a little about the Responsive Images Community Group because I mentioned how I thought it would be nice if there could be some media query for `writing-mode` with the `picture` element so I didn't have to do some mildly hackish transforms on my images when I switched modes. And he suggested I write it up as a use-case for responsive images.

But when I reopened this demo that I hadn't touched in a year, my face went from 🤨 to 😱 to 🤬 to 😩 within the first 5 minutes (what can I say? I have an expressive face 🤷). So for catharsis, I'm going to write down my play-by-play of trying to figure out who (i.e. browsers) broke what and hopefully how to mitigate it, for now.

Post is long, use links to skip.

Vertical type on the web still tricky

"Unfortunately, 10 minutes into the attempt, I broke my brain."


Vertical typesetting with writing
https://chenhuijing.com/blo
Chen Hui Jing
The chronicles of a self-taught designer and developer.
About
Work
Talks
Blog

Monday, Dec 4 2017

Vertical typesetting with writing-mode revisited

About year ago, I wrote about the findings from an exercise in attempting to typeset Chinese vertically on the web. What came out of that was a bare-bones demo that allowed you to switch between writing modes using the checkbox hack.

I met Yoav Weiss a little while back and we chatted a little about the Responsive Images Community Group because I mentioned how I thought it would be nice if there could be some media query for `writing-mode` with the `picture` element so I didn't have to do some mildly hackish transforms on my images when I switched modes. And he suggested I write it up as a use-case for responsive images.

But when I reopened this demo that I hadn't touched in a year, my face went from 🤨 to 😱 to 🤬 to 😩 within the first 5 minutes (what can I say? I have an expressive face 🤷). So for catharsis, I'm going to write down my play-by-play of trying to figure out who (i.e. browsers) broke what and hopefully how to mitigate it, for now.

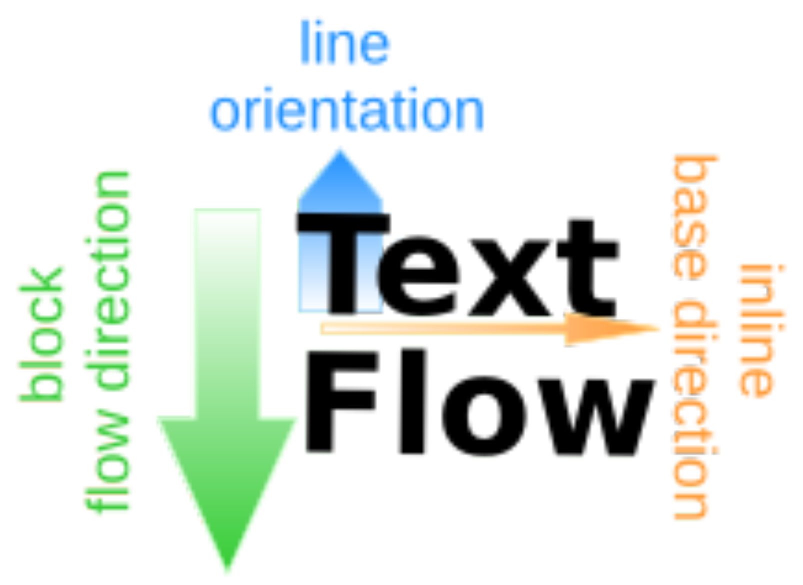
Post is long, use links to skip.

Writing modes

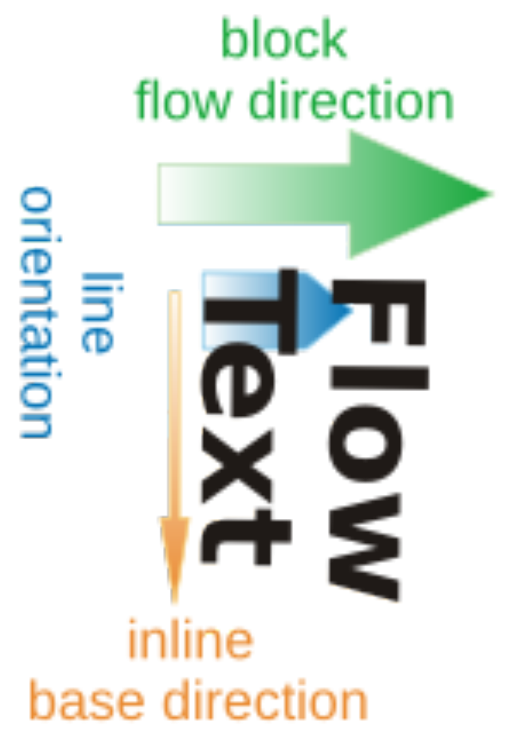
"I've found understanding Writing Modes incredibly helpful when understanding Flexbox and CSS Grid"

– **Jen Simmons**, in "CSS Writing Modes" on 24 Ways

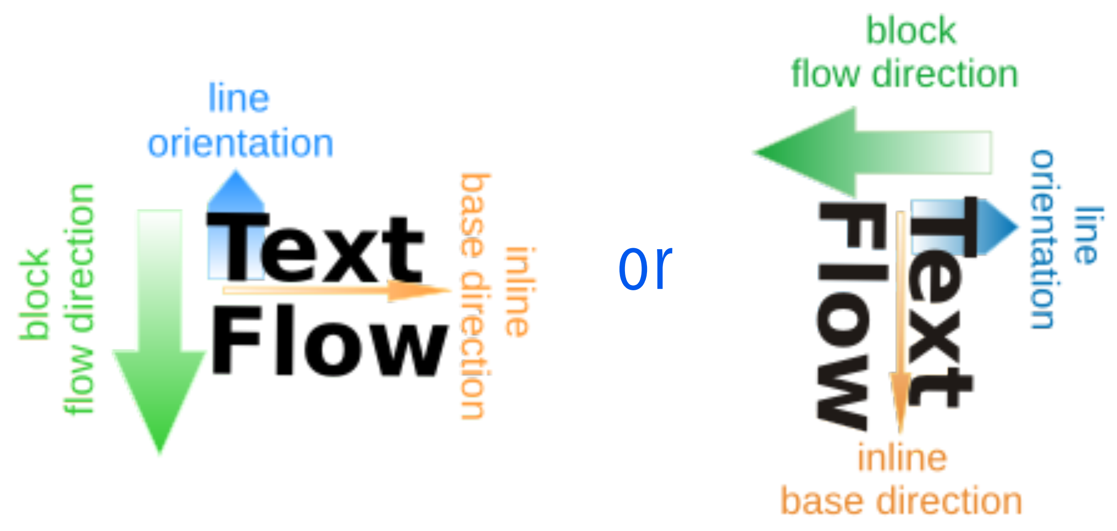
Writing modes



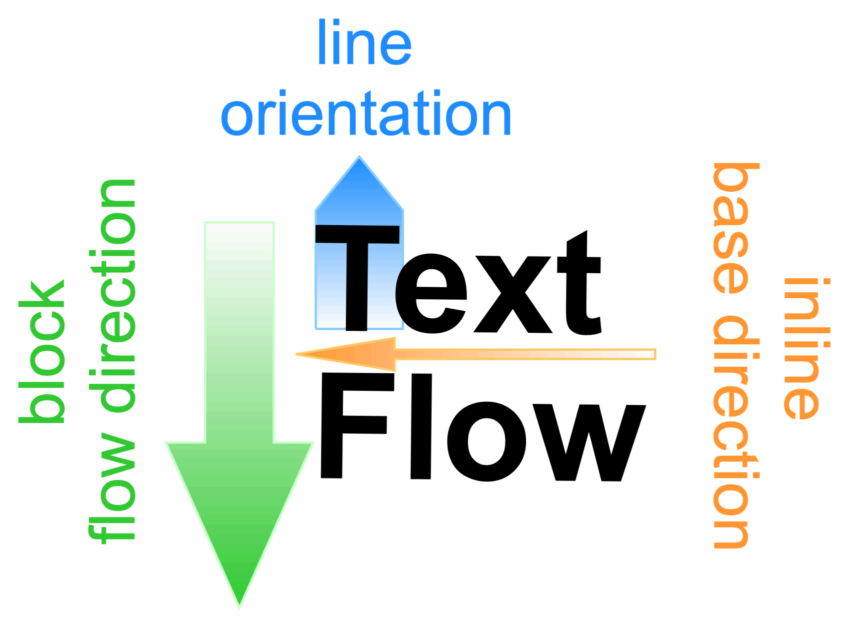
Latin-based



Mongolian-based



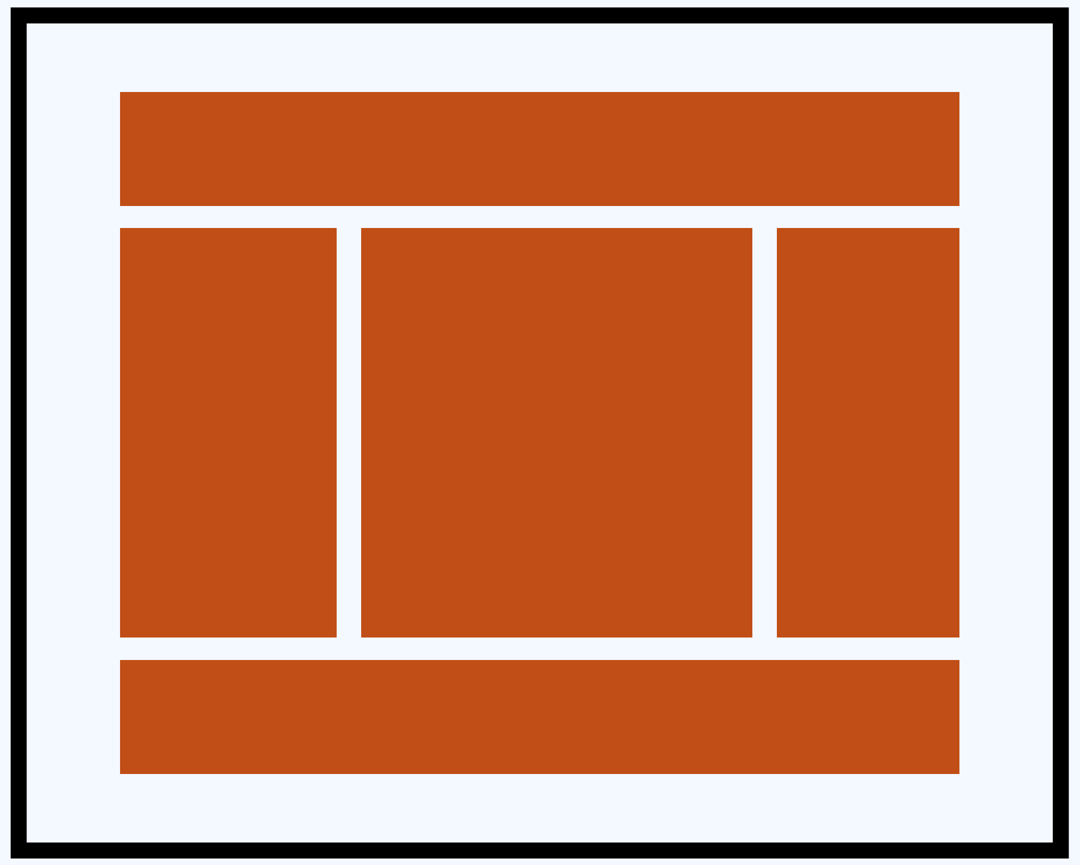
Han-based



Arabic-based

Writing modes

```
.grid {
  writing-mode: horizontal-tb;
  writing-mode: vertical-rl;
  writing-mode: vertical-lr;
  writing-mode: sideways-rl;
  writing-mode: sideways-lr;
}
```

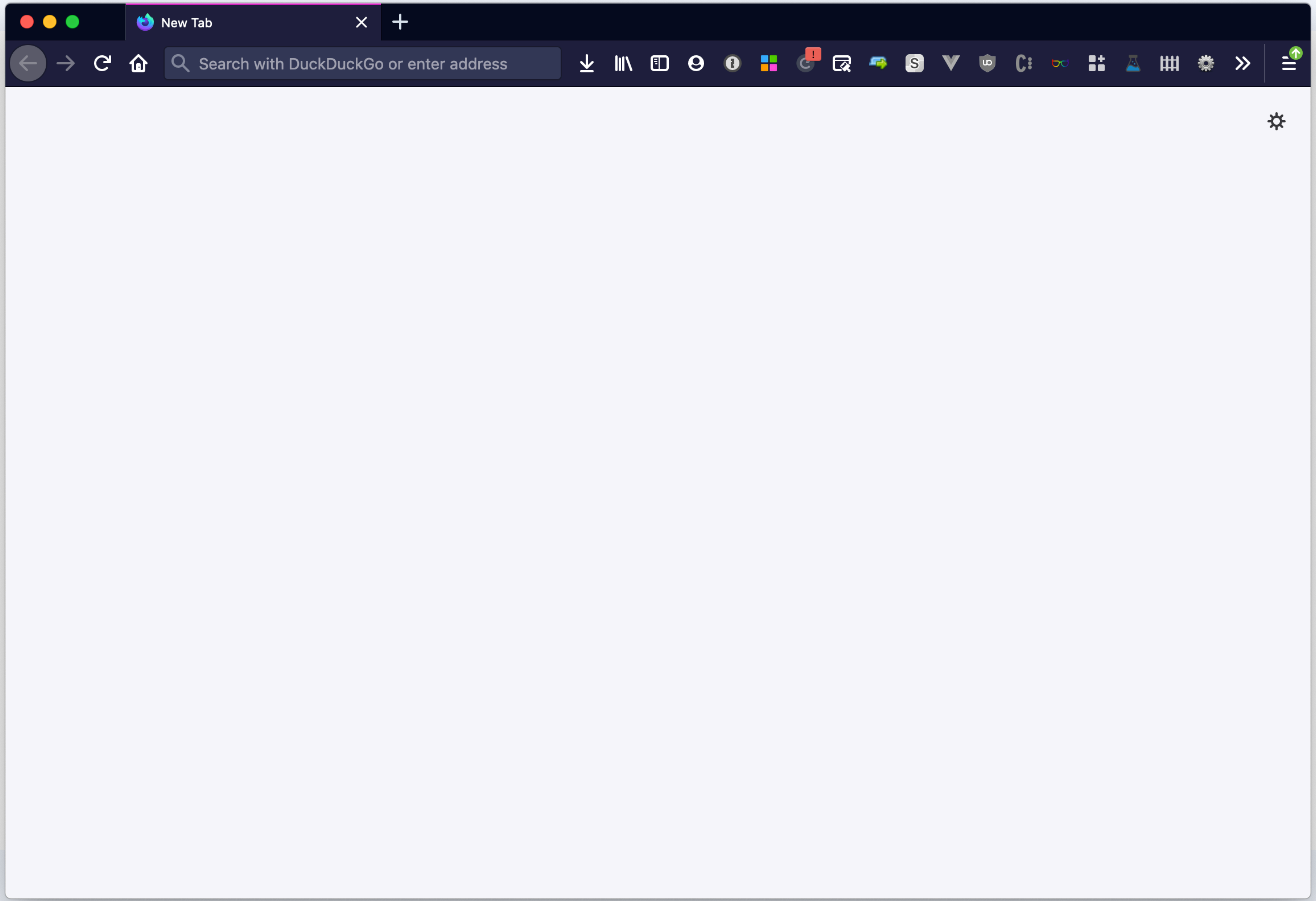


THE GRID CONTAINER

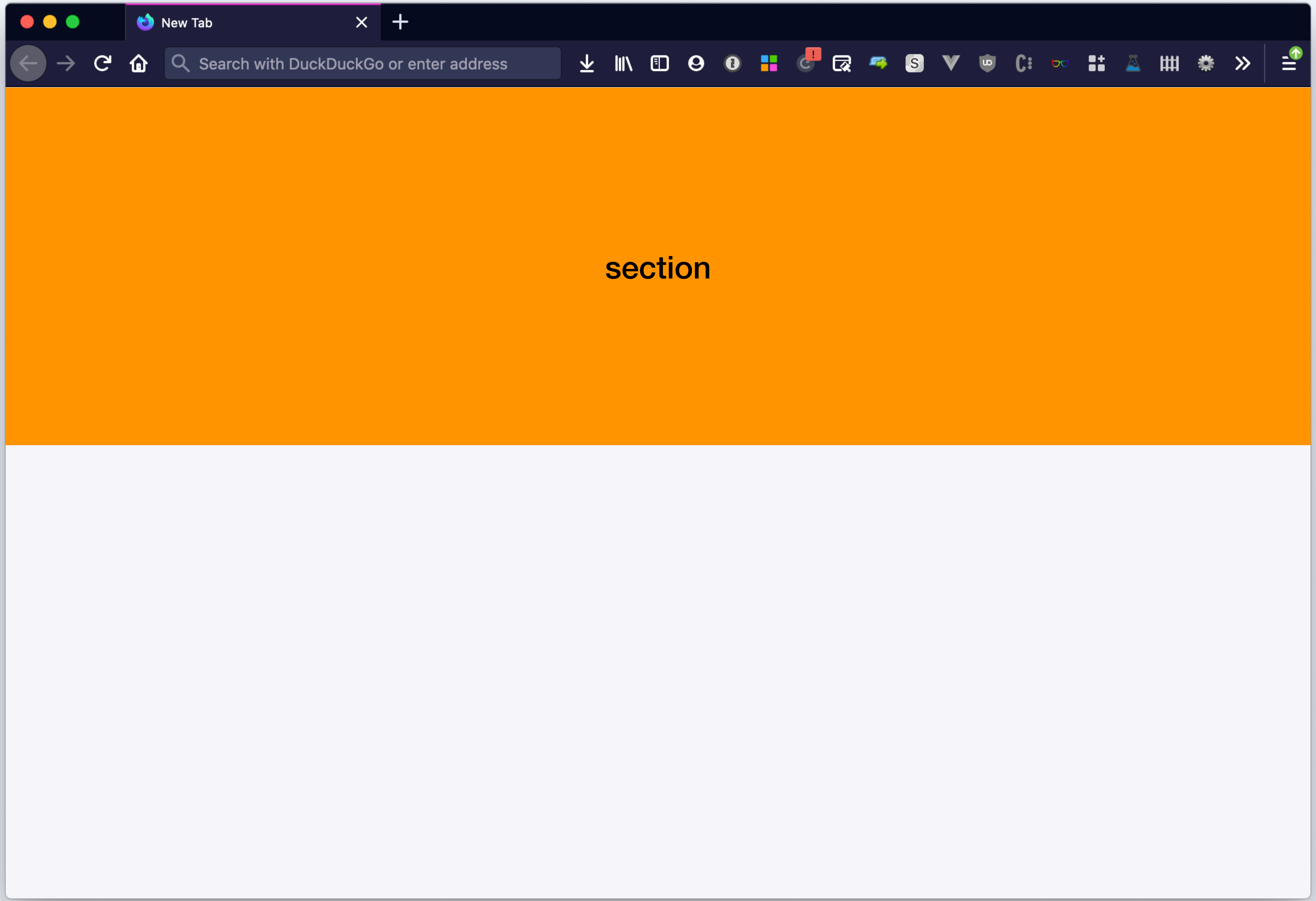
Creating a Grid

```
.grid {
  display: grid;
}
```

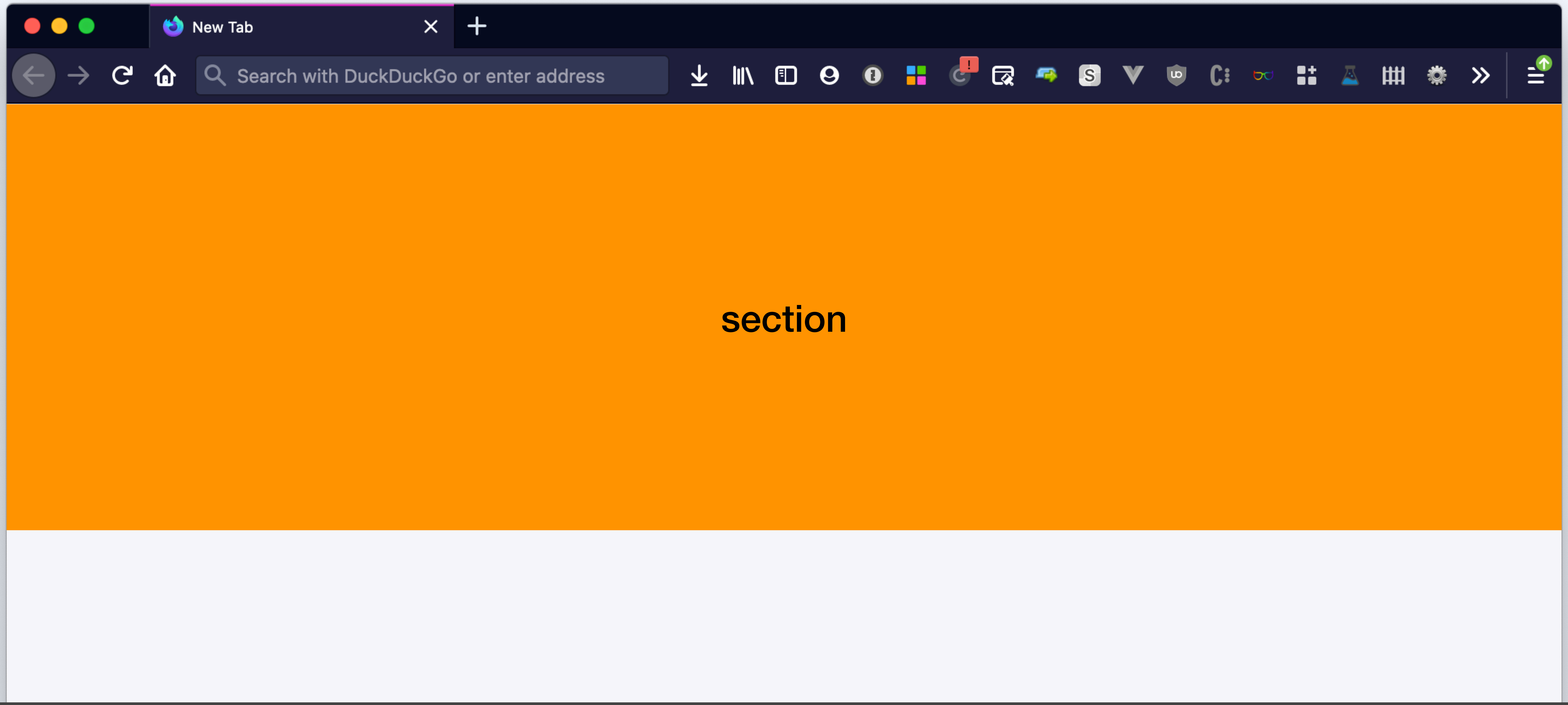
Inline size



Inline size

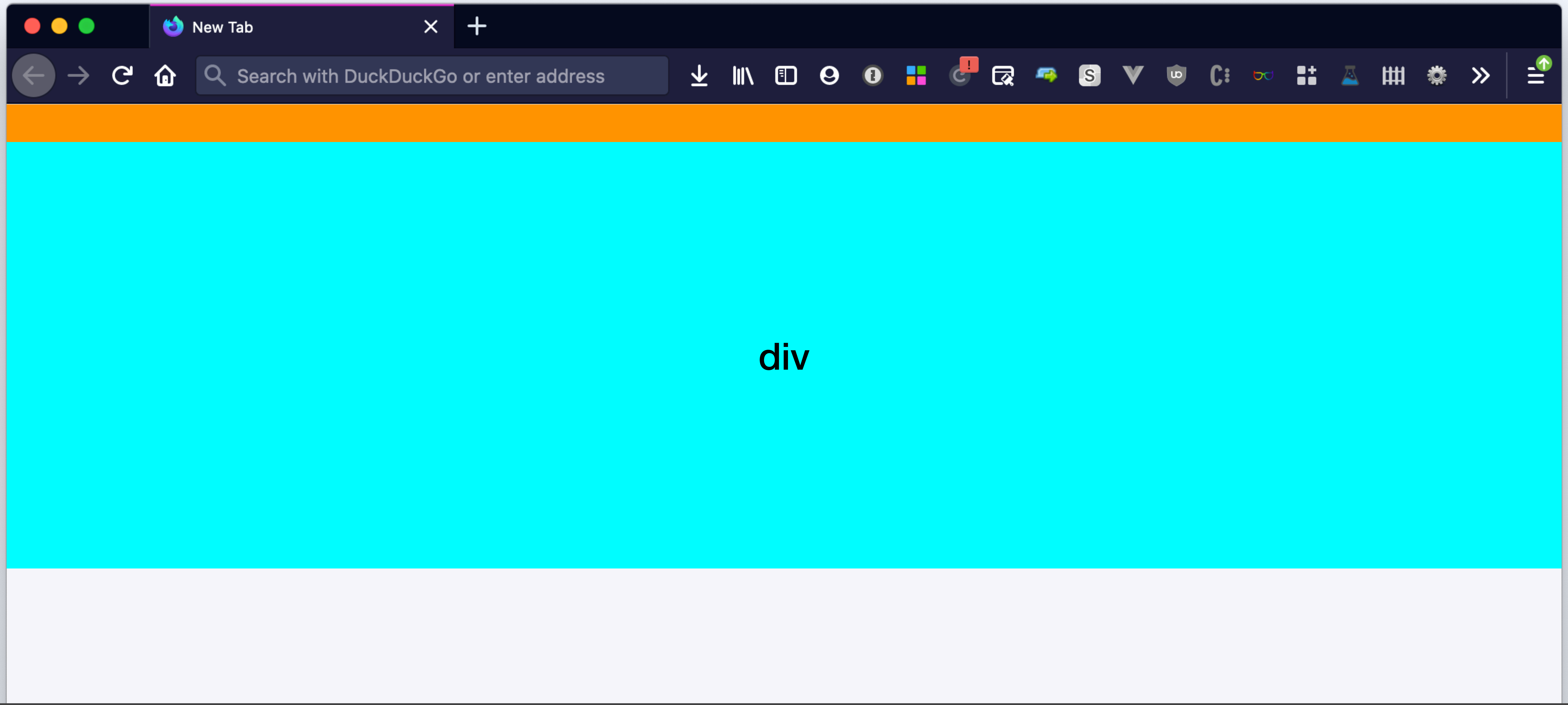


Inline size



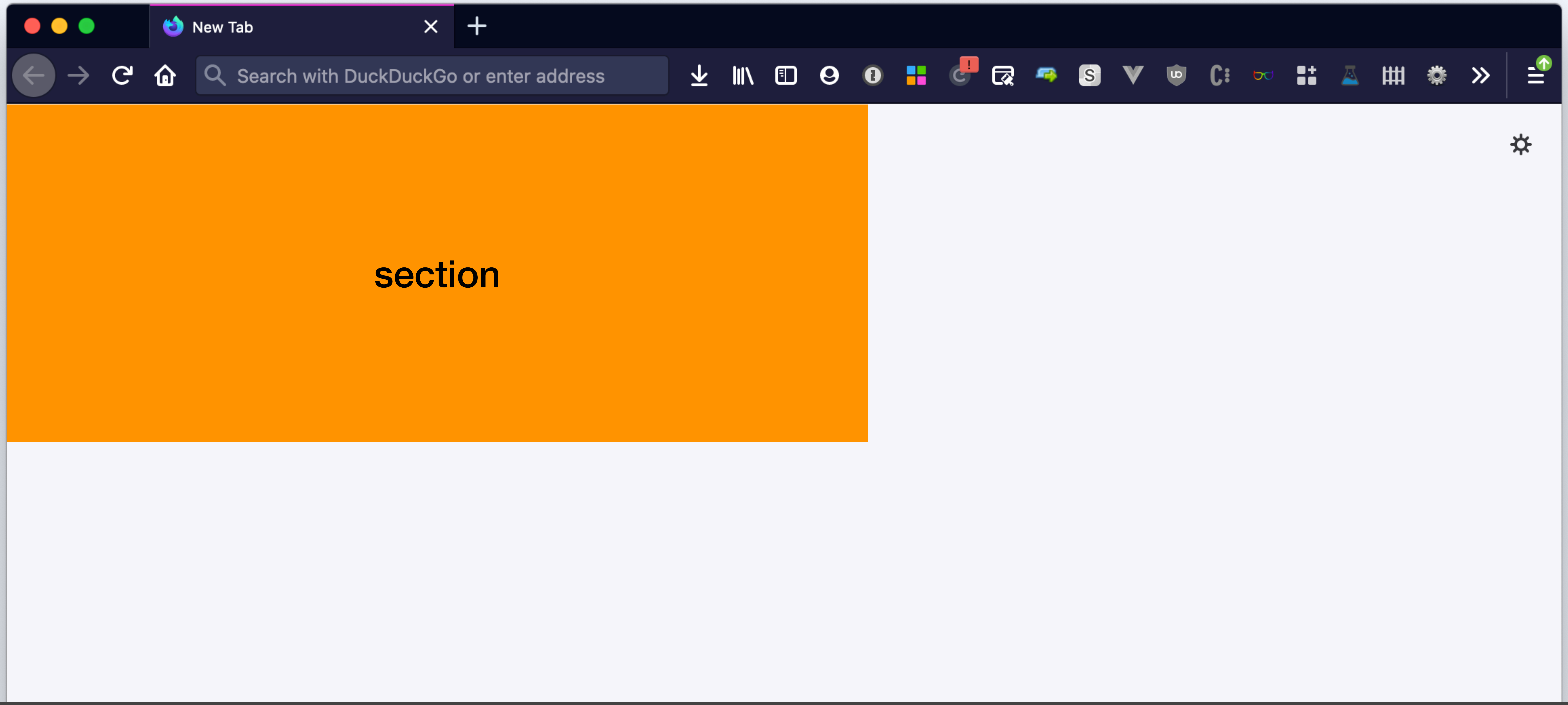
```
<!-- width: 100% of window -->
<section></section>
```

Inline size



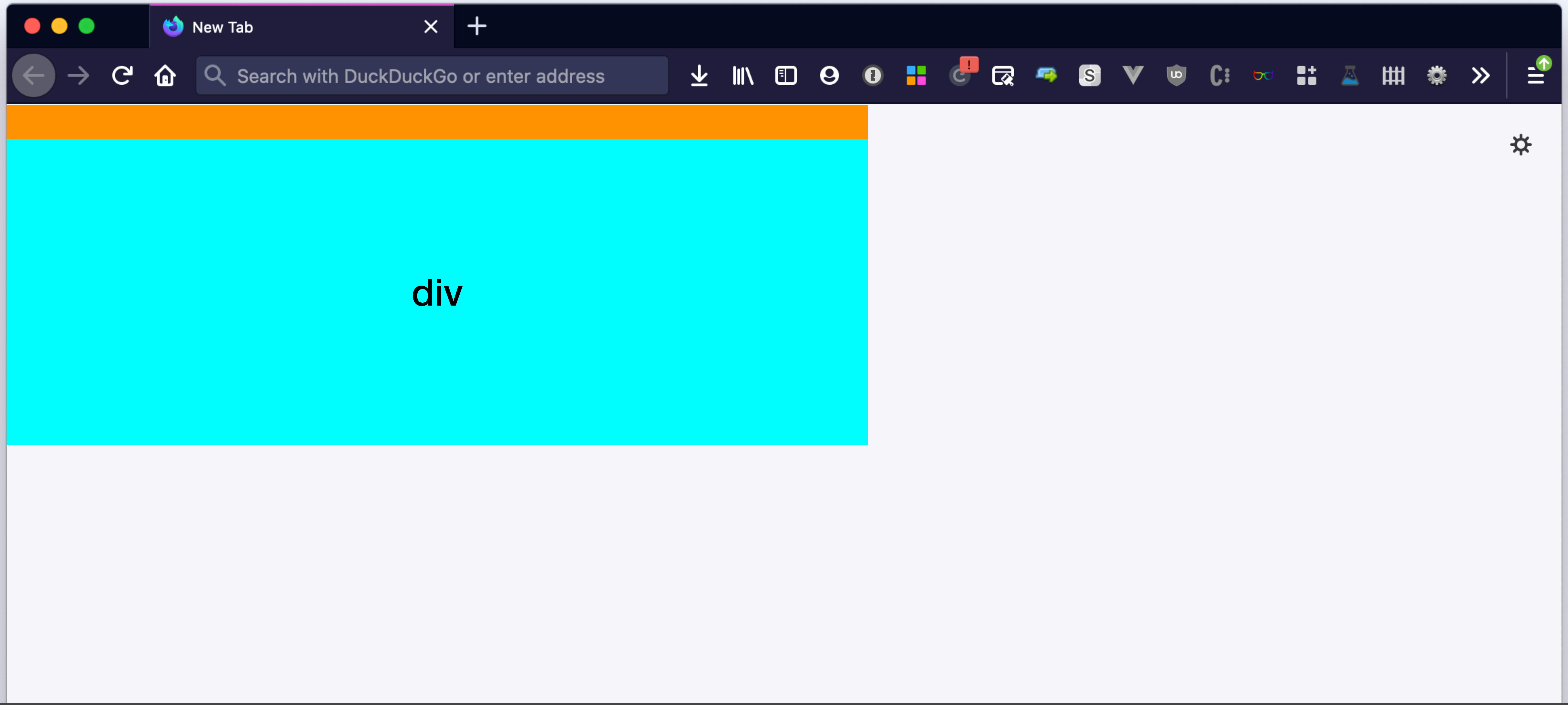
```
<!-- width: 100% of window -->
<section>
  <!-- width: also 100% of window -->
  <div style="display: grid;">
  </div>
```

Inline size



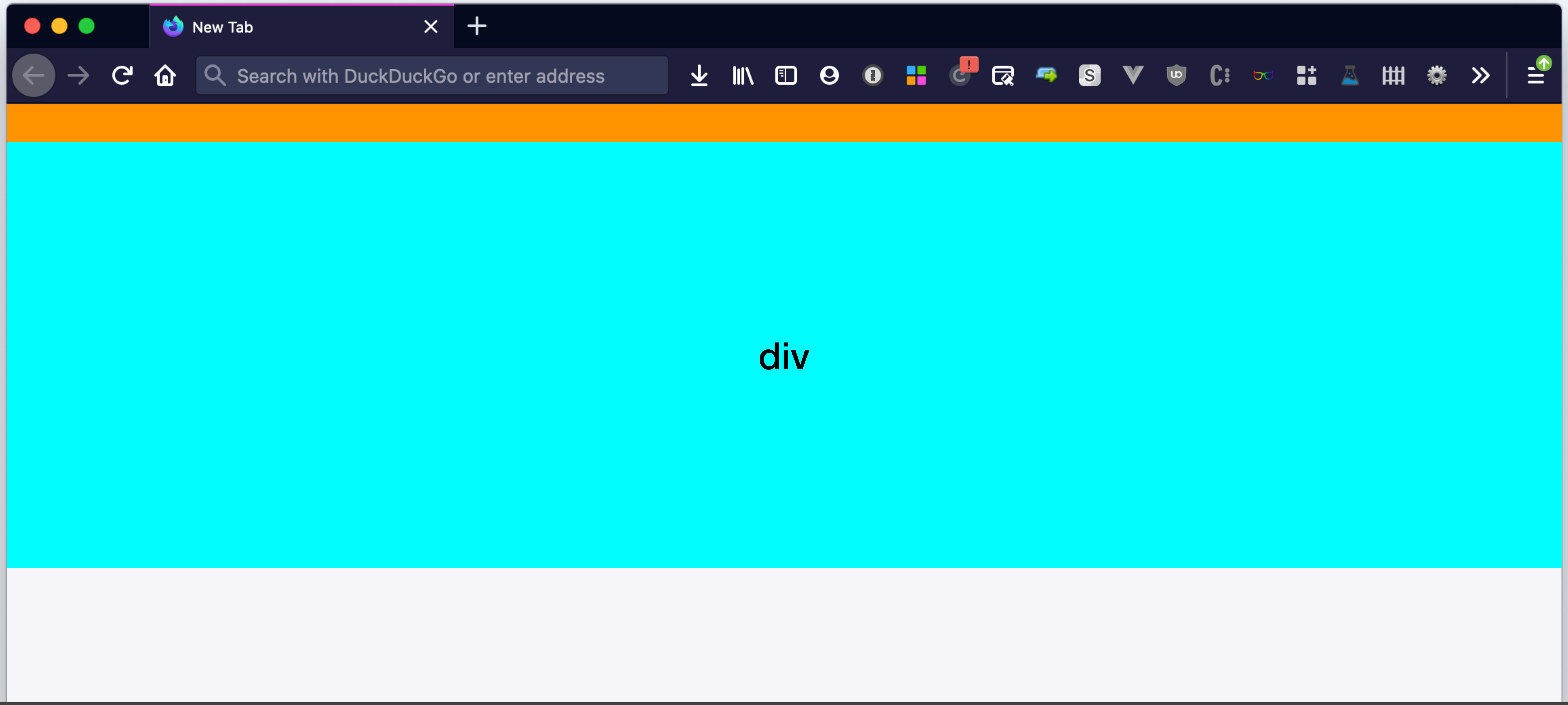
```
<!-- width: 500px -->
<section style="width: 500px;">
</section>
```

Inline size



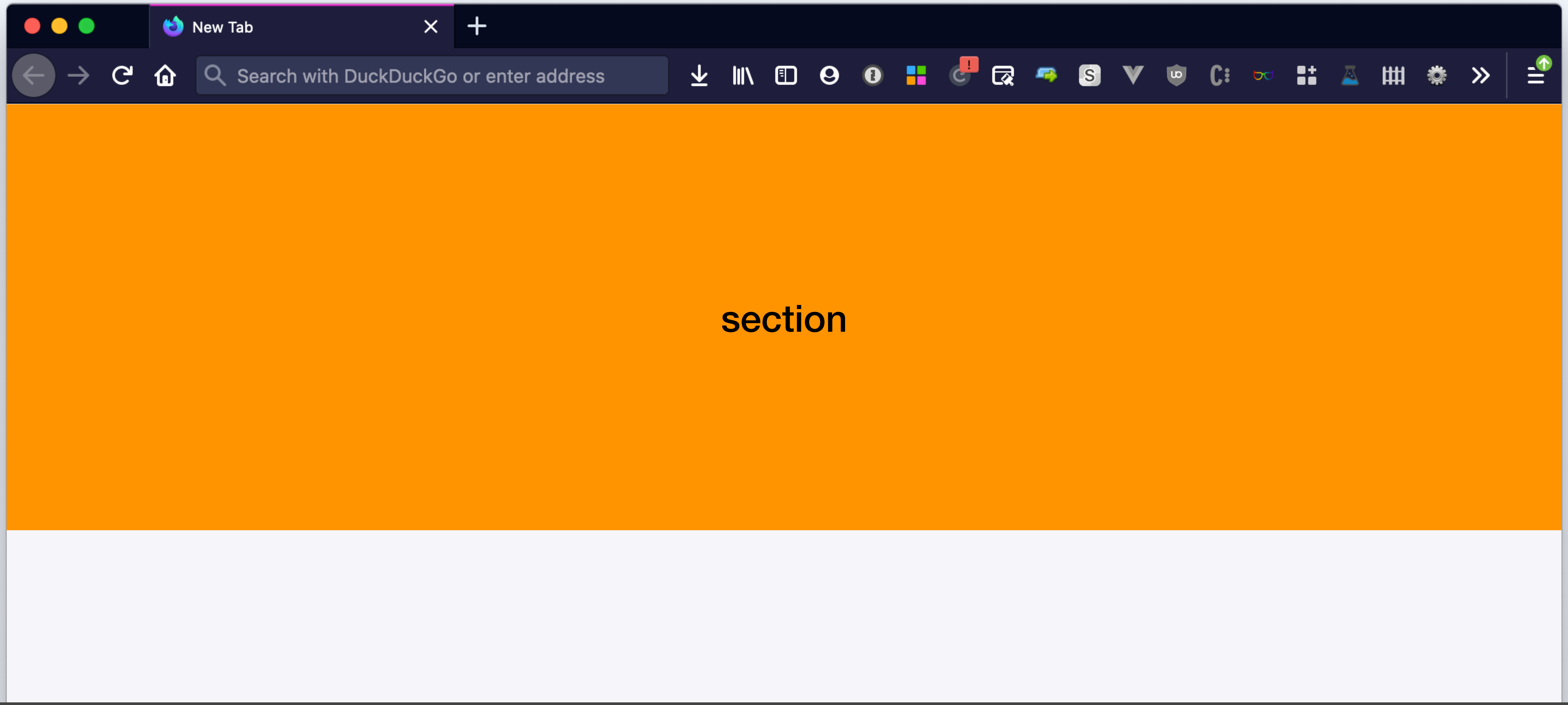
```
<!-- width: 500px -->
<section style="width: 500px;">
  <!-- width: also 500px -->
  <div style="display: grid;">
  </div>
```

Inline size (float)



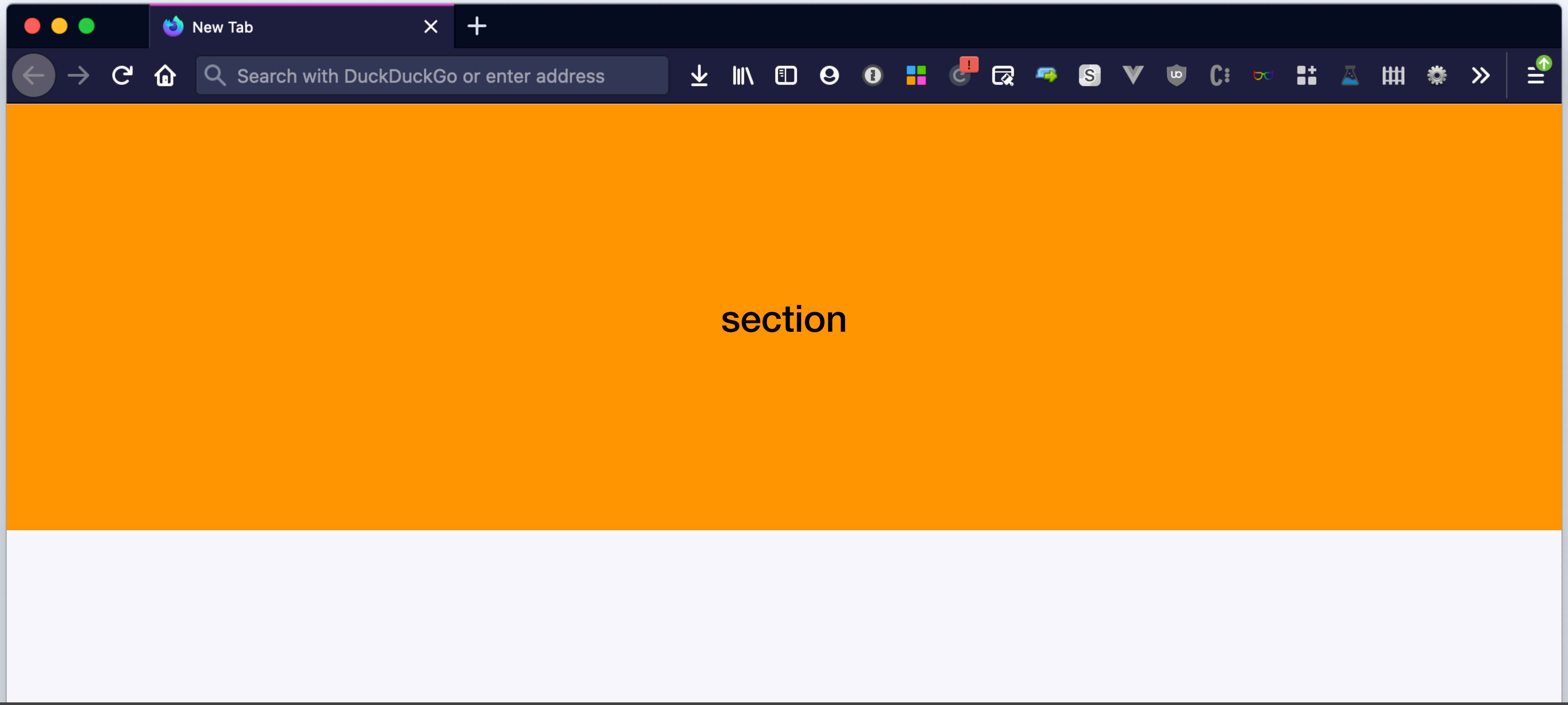
```
<!-- width: 100% of window -->
<section>
  <!-- width: also 100% of window -->
   <div style="display: grid;”>
   </div>
```

Inline size (float)



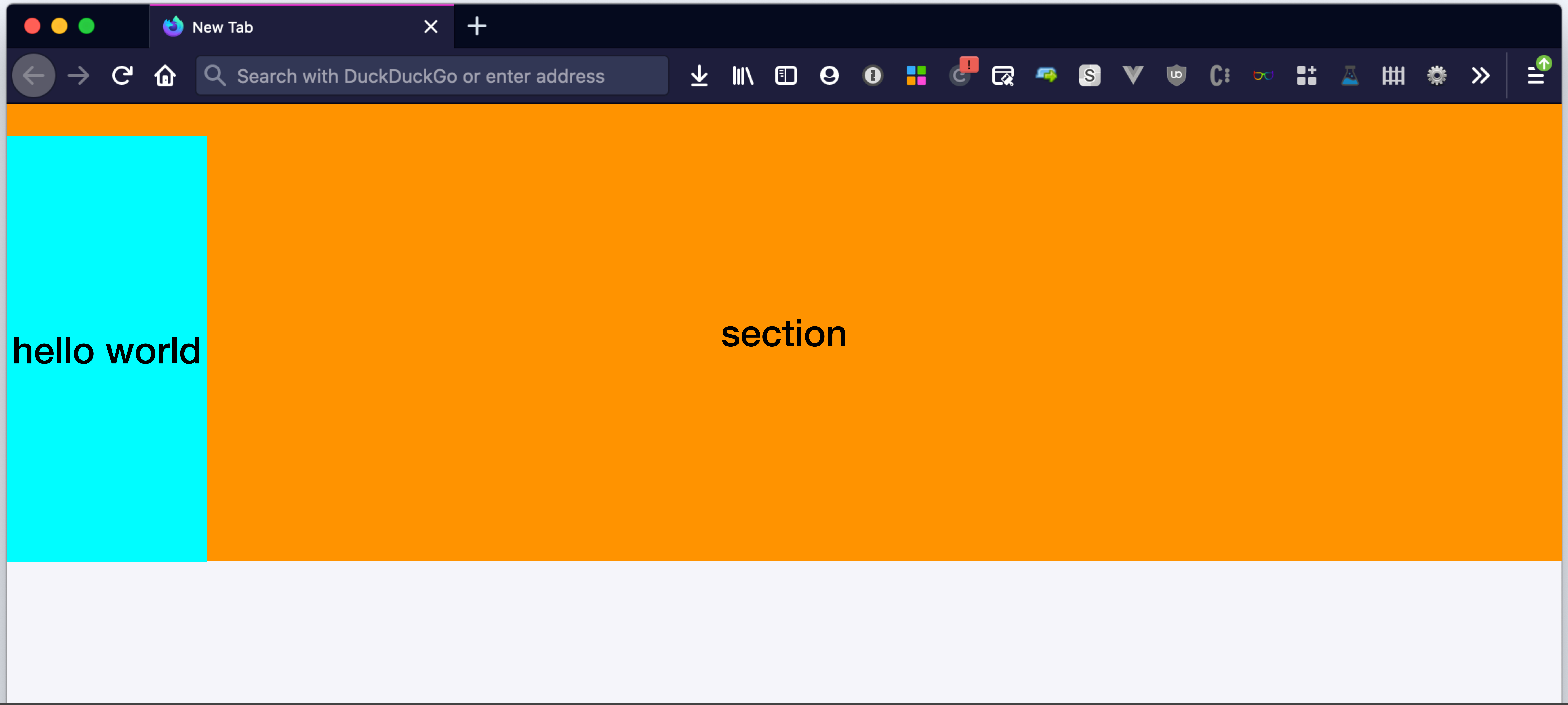
```
<!-- width: 100% of window -->
<section>
  <!-- width: [?] -->
  <div style="display: grid; float: left;”>
  </div>
```

Inline size (posabs)



```
<!-- width: 100% of window -->
<section>
  <!-- width: [?] -->
  <div style="display: grid; position: absolute;”>
  </div>
```

Inline size (posabs)

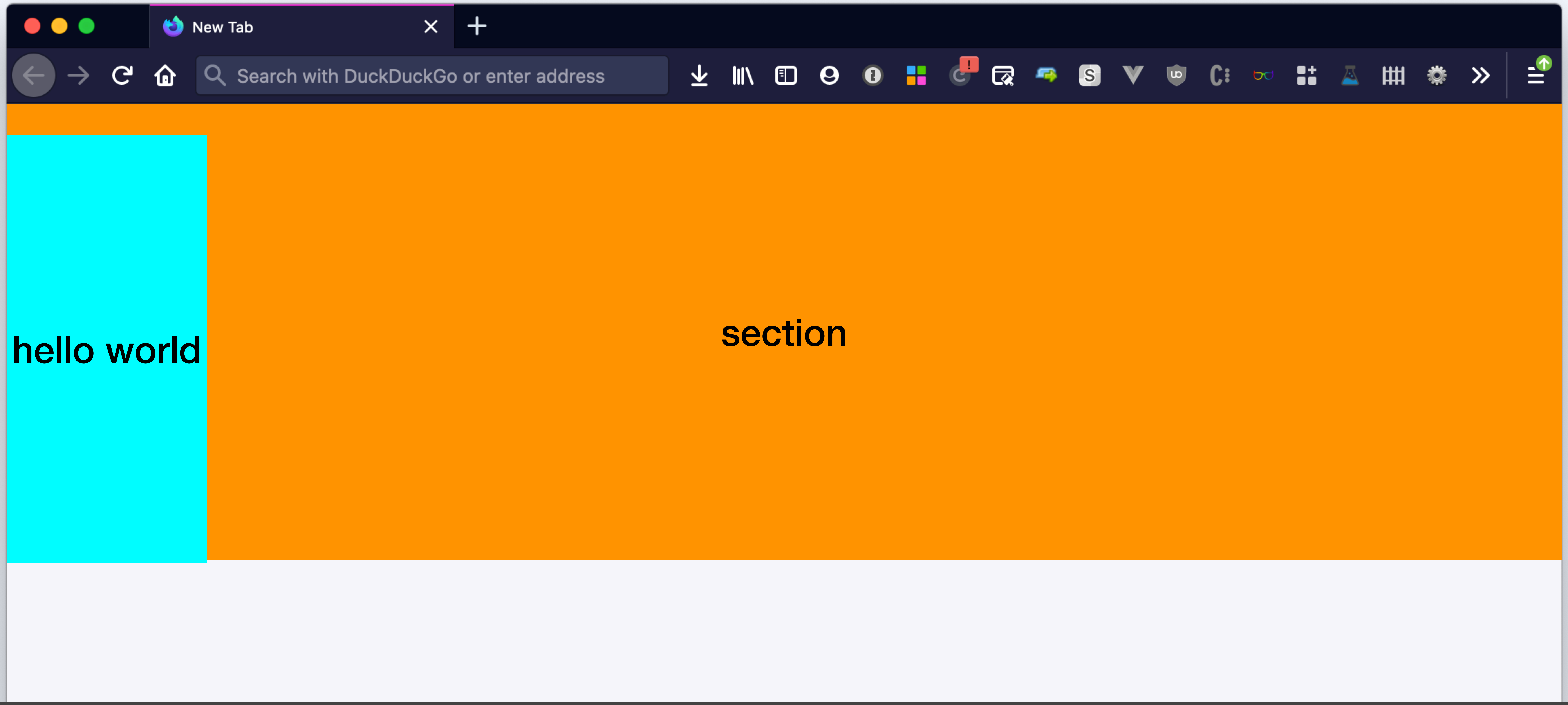

Search with DuckDuckGo or enter address

hello world

section

```
<!-- width: 100% of window -->
<section>
  <!-- width: as much as content requires -->
  <div style="display: grid; position: absolute;”>
    hello world
```

Inline grid



hello world

section

```
<!-- width: 100% of window -->
<section>
  <!-- width: as much content requires -->
  <div style="display: inline-grid;">
    hello world
```

GRID CONTAINER

INLINE

GRID CONTAINER

INLINE

Size of containing element

WHEN

– grid has no explicit width

GRID CONTAINER

INLINE

Size of containing element

WHEN

– grid has no explicit width

Size that the content needs

WHEN

– `float` or `position:absolute`

– inline-grid has no explicit width

GRID CONTAINER

INLINE

Size of containing element

WHEN

– grid has no explicit width

Size that the content needs

WHEN

– `float` or `position:absolute`

– inline-grid has no explicit width

Size that you specified

WHEN

– you specified a size

GRID CONTAINER

BLOCK SIZE

GRID CONTAINER

BLOCK SIZE

Size of containing element

WHEN

– `position:absolute` and it has height 100%

GRID CONTAINER

BLOCK SIZE

Size of containing element

WHEN

– `position:absolute` and it has height 100%

Size that the content needs

WHEN

– no exception applies

GRID CONTAINER

BLOCK SIZE

Size of containing element

WHEN

– `position:absolute` and it has height 100%

Size that the content needs

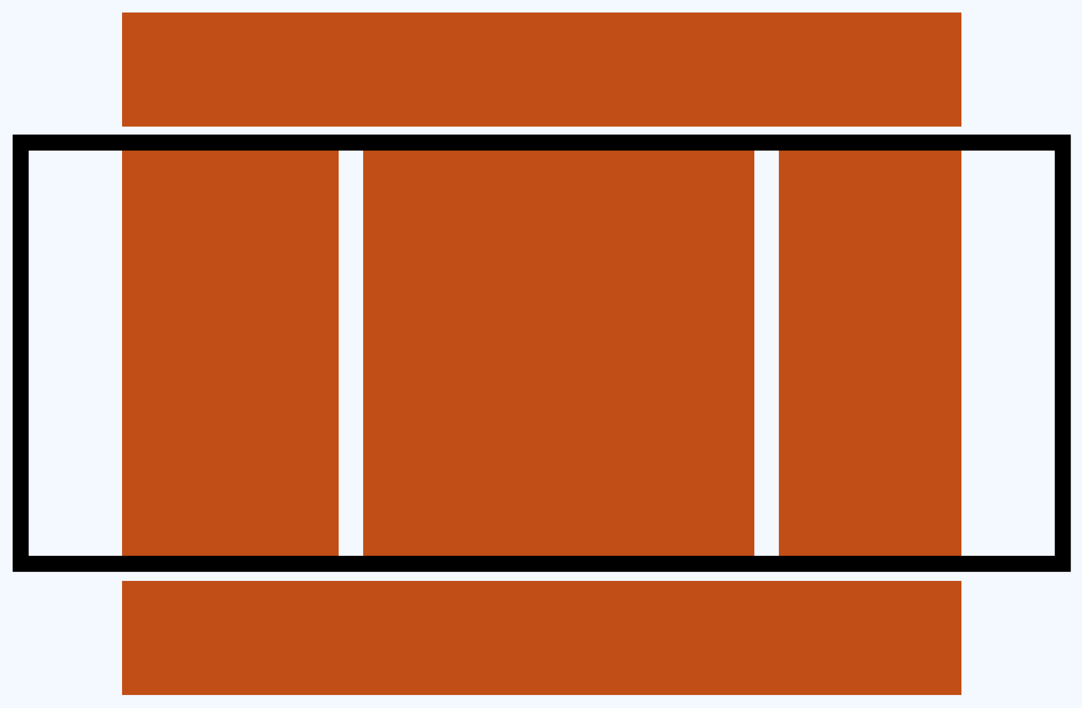
WHEN

– no exception applies

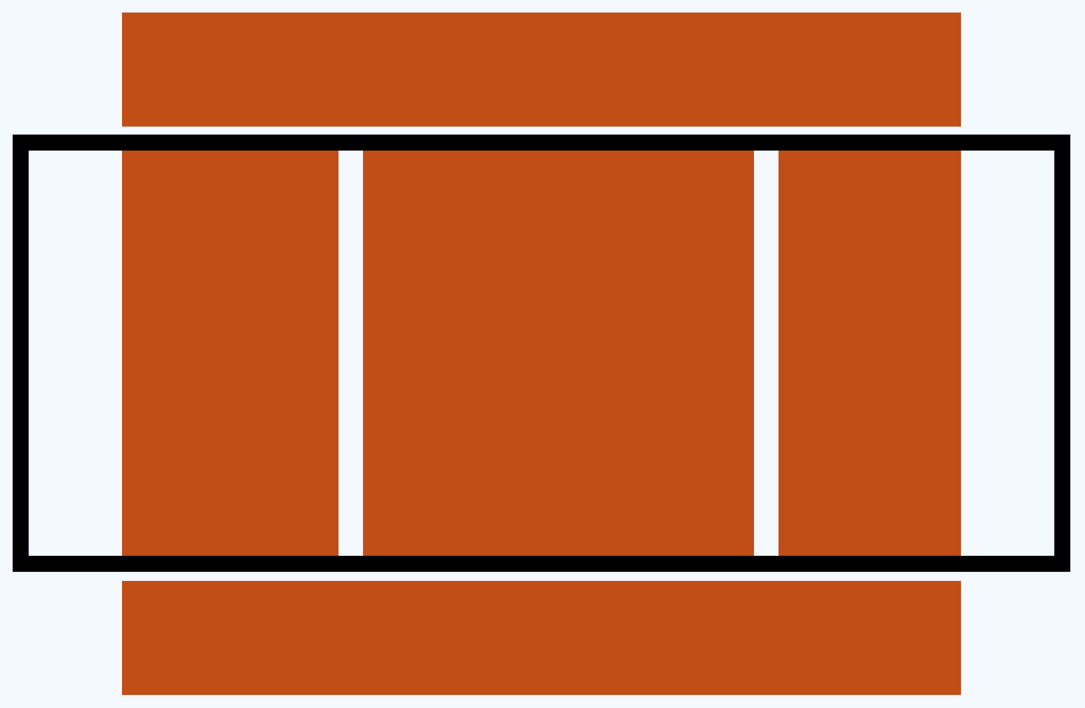
Size that you specified

WHEN

– you specified a size



GRID TRACKS



GRID TRACKS COLUMNS / ROWS

Creating columns

```
.grid {
  display: grid;
  grid-template-columns: 100px 400px 200px;
}
```

Creating rows

```
.grid {
  display: grid;
  grid-template-rows: 100px 400px 200px;
}
```

GRID TRACKS: FIXED

Fixed sizes

```css
.grid {
  display: grid;
  grid-template-columns: 5cm 10cm 5cm;
}
```

Fixed sizes

```
.grid {
  display: grid;
  grid-template-columns: 200px 500px 100px;
}
```

Fixed sizes

```css
.grid {
  display: grid;
  grid-template-columns: 50em 120ch 50rem;
}
```

Fixed sizes

```
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE

```
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE

ABSOLUTE

```
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE

- em
- ex
- ch
- rem

font

ABSOLUTE

```
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE

- em
- ex
- ch
- rem

font

- vw
- vh
- vmin
- vmax

viewport

ABSOLUTE

```
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE		ABSOLUTE
– em	font	– cm
– ex	font	– mm
– ch	font	– Q
– rem	font	– in
– vw	viewport	– pc
– vh	viewport	– pt
– vmin	viewport	– px
– vmax	viewport	

```
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE

- em
- ex
- ch
- rem

font

- vw
- vh
- vmin
- vmax

viewport

ABSOLUTE

- cm `1cm = 96px/2.54`
- mm
- Q
- in
- pc
- pt
- px

```
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE		ABSOLUTE	
– em	font	– cm	`1cm = 96px/2.54`
– ex	font	– mm	`1mm = 1/10th of 1cm`
– ch	font	– Q	
– rem	font	– in	
– vw	viewport	– pc	
– vh	viewport	– pt	
– vmin	viewport	– px	
– vmax	viewport		

```css
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE

– em	*font*
– ex	
– ch	
– rem	
– vw	*viewport*
– vh	
– vmin	
– vmax	

ABSOLUTE

– cm	`1cm = 96px/2.54`
– mm	`1mm = 1/10th of 1cm`
– Q	`1Q = 1/40th of 1cm`
– in	
– pc	
– pt	
– px	

```css
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE

- em
- ex
- ch
- rem

font

- vw
- vh
- vmin
- vmax

viewport

ABSOLUTE

– cm	1cm = 96px/2.54
– mm	1mm = 1/10th of 1cm
– Q	1Q = 1/40th of 1cm
– in	1in = 2.54cm = 96px
– pc	
– pt	
– px	

```css
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE

- em
- ex
- ch
- rem

font

- vw
- vh
- vmin
- vmax

viewport

ABSOLUTE

– cm	`1cm = 96px/2.54`
– mm	`1mm = 1/10th of 1cm`
– Q	`1Q = 1/40th of 1cm`
– in	`1in = 2.54cm = 96px`
– pc	`1pc = 1/6th of 1in`
– pt	
– px	

```css
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

CSS Values and Units Module Level 3 · https://www.w3.org/TR/css-values-3/

Fixed sizes

RELATIVE

– em	*font*
– ex	
– ch	
– rem	
– vw	*viewport*
– vh	
– vmin	
– vmax	

ABSOLUTE

– cm	1cm = 96px/2.54
– mm	1mm = 1/10th of 1cm
– Q	1Q = 1/40th of 1cm
– in	1in = 2.54cm = 96px
– pc	1pc = 1/6th of 1in
– pt	1pt = 1/72th of 1in
– px	

```css
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

RELATIVE

– em	font
– ex	
– ch	
– rem	
– vw	viewport
– vh	
– vmin	
– vmax	

ABSOLUTE

– cm	1cm = 96px/2.54
– mm	1mm = 1/10th of 1cm
– Q	1Q = 1/40th of 1cm
– in	1in = 2.54cm = 96px
– pc	1pc = 1/6th of 1in
– pt	1pt = 1/72th of 1in
– px	1px = 1/96th of 1in

```css
.grid {
  display: grid;
  grid-template-columns: /* use any size here */;
}
```

Fixed sizes

"In CSS sizing primitives, a fixed size means a size that is independent of layout or content"

– **Fantasai**, in her talk "Defining auto"

Fixed track sizes

HTML
CSS
JS
1
2
3
4
-1
-4
-3
-2
000000000

```html
<section>
  <div>000</div>
  <div>000</div>
  <div>000</div>
</section>
```

```css
section {
  display: grid;
  grid-template-columns: 3ch 3ch 3ch;
}
```

Fixed track sizes

HTML

```html
<section>
  <div>000</div>
  <div>Singapore</div>
  <div>000</div>
</section>
```

CSS

```css
section {
  display: grid;
  grid-template-columns: 3ch 3ch 3ch;
}
```

JS
1
2
3
4
-1
-4
-3
-2
000000000

Fixed track sizes

HTML
CSS
JS
1
2
3
4
-1
-2
-3
-4
1
2
-1

```html
<section>
  <div>000</div>
  <div>Singapore</div>
  <div>000</div>
</section>
```

```css
section {
  display: grid;
  grid-template-columns: 3ch 3ch 3ch;
}
```

000Singapore000

Fixed track sizes

HTML

```html
<section>
  <div>000</div>
  <div>Singapore</div>
  <div>000</div>
</section>
```

CSS

```css
div:nth-child(2) {
  color: green;
  overflow: hidden;
}
```

JS
1
2
3
4
-1
-4
-3
-2
000Sing000

GRID TRACKS: FRACTIONS

Fractions

This is not auto

```
.grid {
  display: grid;
  grid-template-columns: 50em 1fr 10em;
}
```

Sizing with fractions

```html
<section>
  <div></div>
  <div></div>
  <div></div>
</section>
```

```css
section {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}

div:nth-child(2) {
```

HTML
CSS
JS
1
2
3
-1
1
2
-4
-3
-2

Sizing with fractions

HTML
CSS
1
2
3
-1
2
-4
-3
-2

```html
<section>
  <div></div>
  <div>blijkbaar</div>
  <div></div>
</section>
```

```css
section {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}

div:nth-child(2) {
```

blijkbaar

This default is a feature, usually you don’t want text on top of other text.

Sizing with fractions

The distribution of leftover space occurs after all non-flexible track sizing functions have reached their maximum.

in "CSS Grid Layout Module Level 1"

Sizing with fractions

The distribution of leftover space occurs after all non-flexible track sizing functions have reached their maximum. The total size of such rows or columns is subtracted from the available space, yielding the leftover space,

in “CSS Grid Layout Module Level 1”

Sizing with fractions

The distribution of leftover space occurs after all non-flexible track sizing functions have reached their maximum. The total size of such rows or columns is subtracted from the available space, yielding the leftover space, which is then divided among the flex-sized rows and columns in proportion to their flex factor.

in “CSS Grid Layout Module Level 1”

Sizing with fractions

HTML

```html
<section>
  <div></div>
  <div>blijkbaar</div>
  <div></div>
</section>
```

CSS

```css
section {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}

div:nth-child(2) {
```

1
2
3
1
-1
2
-4
blijkbaar

Sizing with fractions

HTML

```
<section>
  <div></div>
  <div>blijkbaar</div>
  <div></div>
</section>
```

CSS

```
section {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}

div:nth-child(2) {
```

1
2
3
1
-1
blijkbaar
2
-4
-2

One cell in a track can affect the whole track’s size.

Sizing with fractions

"If you [don't want your track to grow to prevent overflow], you can do so by making 0 the first value in minmax(). "

– **Rachel Andrew**, in "CSS Writing Modes" on 24 Ways

Rachel Andrew, "How big is that box?" • https://www.smashingmagazine.com/2018/01/understanding-sizing-css-layout/

Fractions

```
.grid {
  display: grid;
  grid-template-columns:
    50em 1fr 10em;
}
```

Fractions

```
.grid {
  display: grid;
  grid-template-columns:
    50em minmax(0, 1fr) 10em;
}
```

GRID TRACKS: KEYWORDS

min-content

HTML
CSS
JS
Result
1
2
-1
-4
-3
1
2

```css
.grid {
  display: grid;
  grid-template-columns:
    min-content min-content min-content;
}
```

in dit vak zal je niet een lang woord zien

het zijn wel gekke tijden dit jaar

het is een goed idee om niet te lang

min-content

HTML
CSS
JS
Result

```
.grid {
  display: grid;
  grid-template-columns:
    min-content min-content min-content;
}
```

1
2
-1
-4
-3
1
2
in dit vak zal je niet een lang woord zien
het zijn wel gekke minimumtemperaturen dit jaar
het is een goed idee om niet te lang

max-content

HTML
CSS
JS
Result
1
2
-4
-3
-1

```
.grid {
  display: grid;
  grid-template-columns:
    max-content max-content max-content;
}
```

in dit vak het zijn wel gekke het is goed

fit-content(value)

HTML
CSS
JS
Result
1
2
-1
1
2
-4
-3
in dit vak
het zijn wel gekke
het is goed

```
.grid {
  display: grid;
  grid-template-columns:
    max-content fit-content(50px) max-content;
}
```

GRID TRACKS: AUTO

Auto track size

```
.grid {
  display: grid;
  grid-template-columns: 50em auto 10em;
}
```

Auto is what you'll get if you don't size your track.

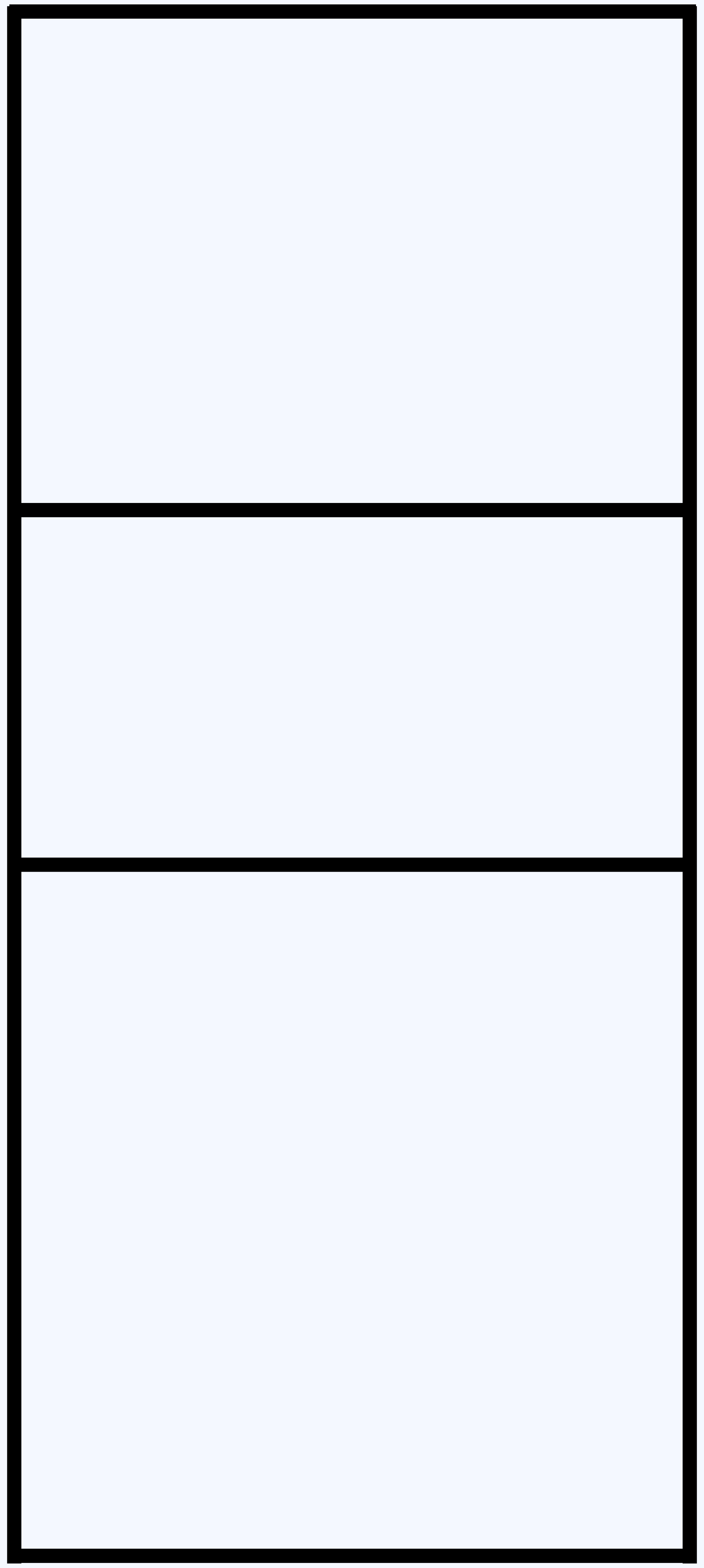
(Overridable with `grid-auto-rows` / `grid-auto-columns`)

Auto track size

```
.grid {
  display: grid;
}
```

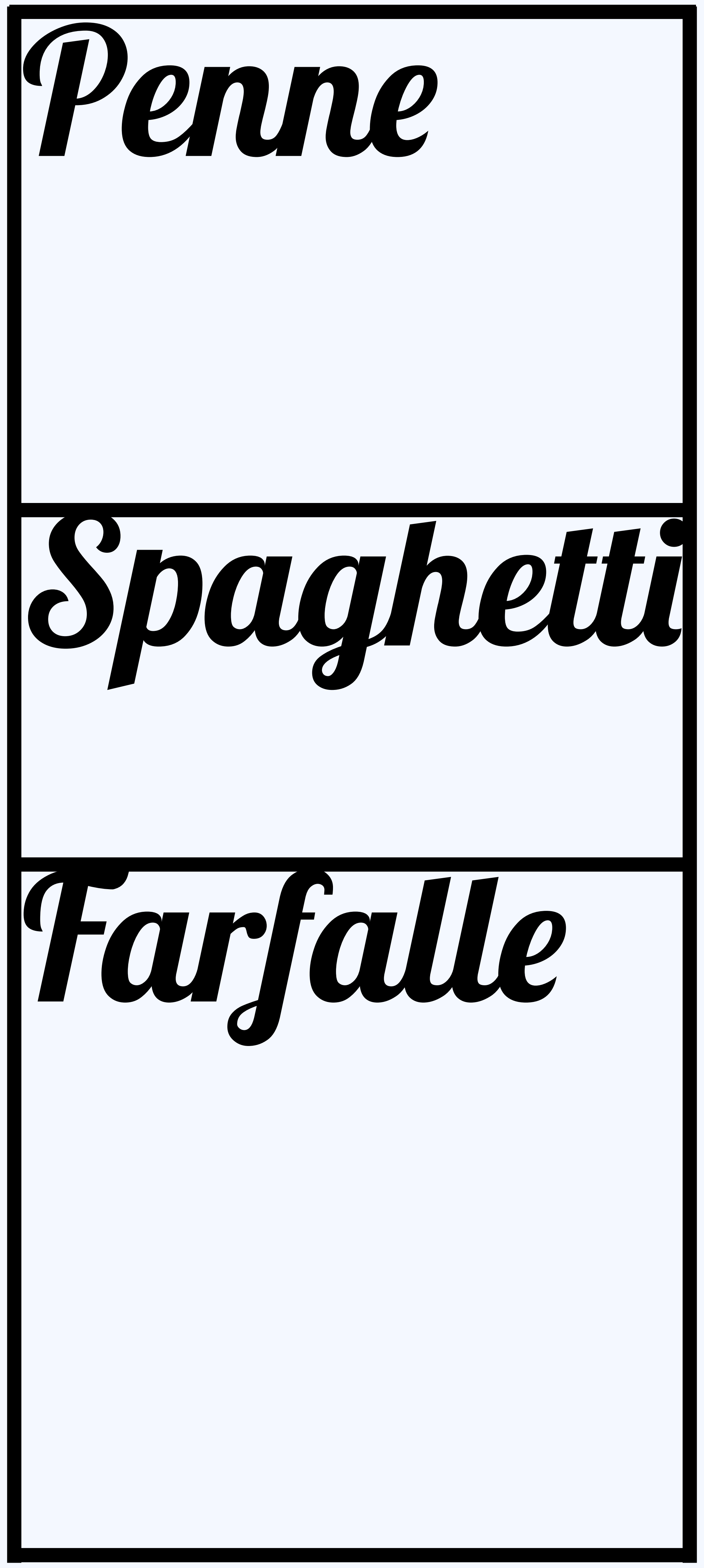
Auto track size

Maximum track size: of all grid items, pick the one with the largest `max-content`. This is your track size.



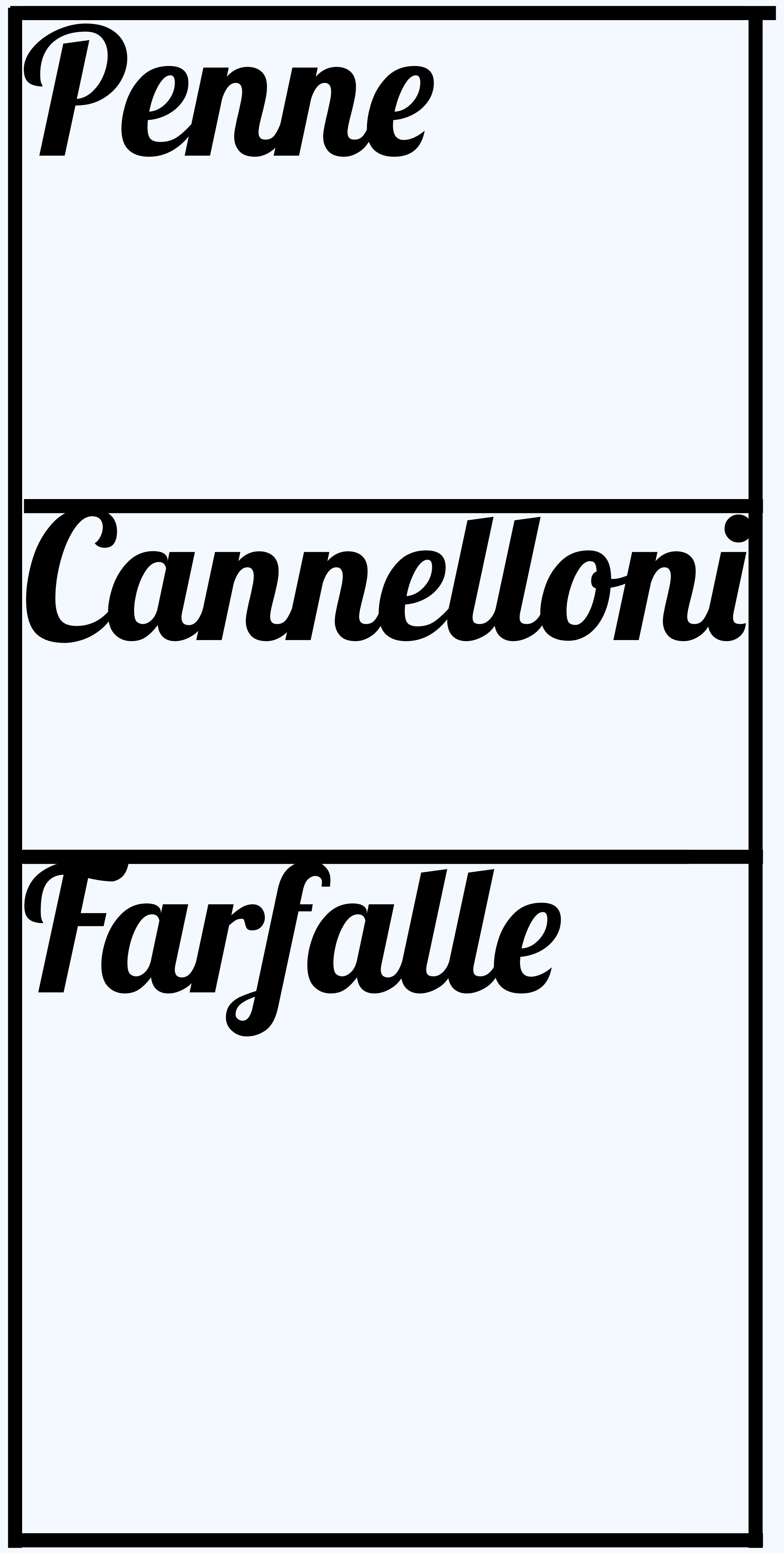
Auto track size

Maximum track size: of all grid items, pick the one with the largest `max-content`. This is your track size.



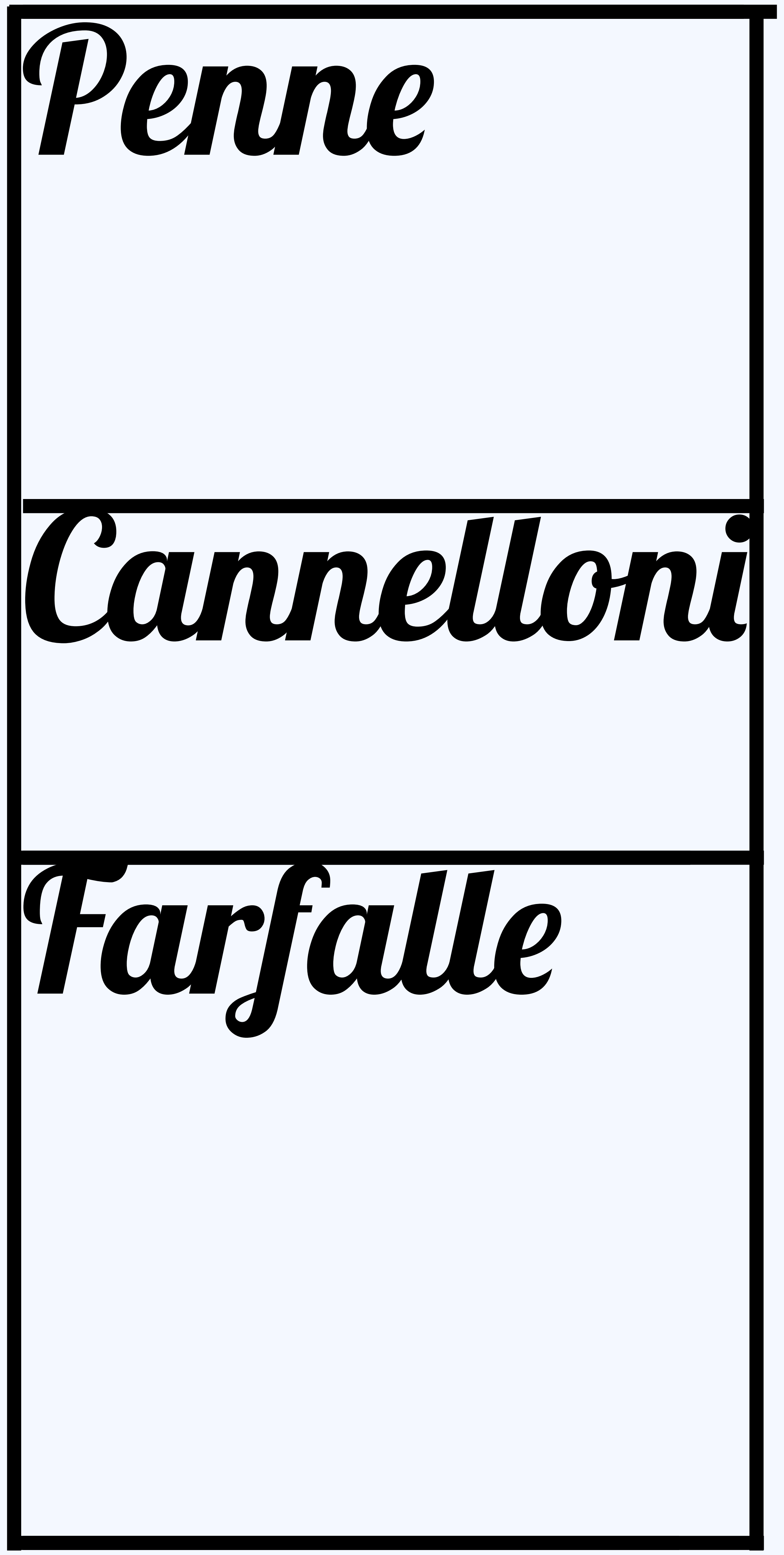
Auto track size

Maximum track size: of all grid items, pick the one with the largest `max-content`. This is your track size.



Auto track size

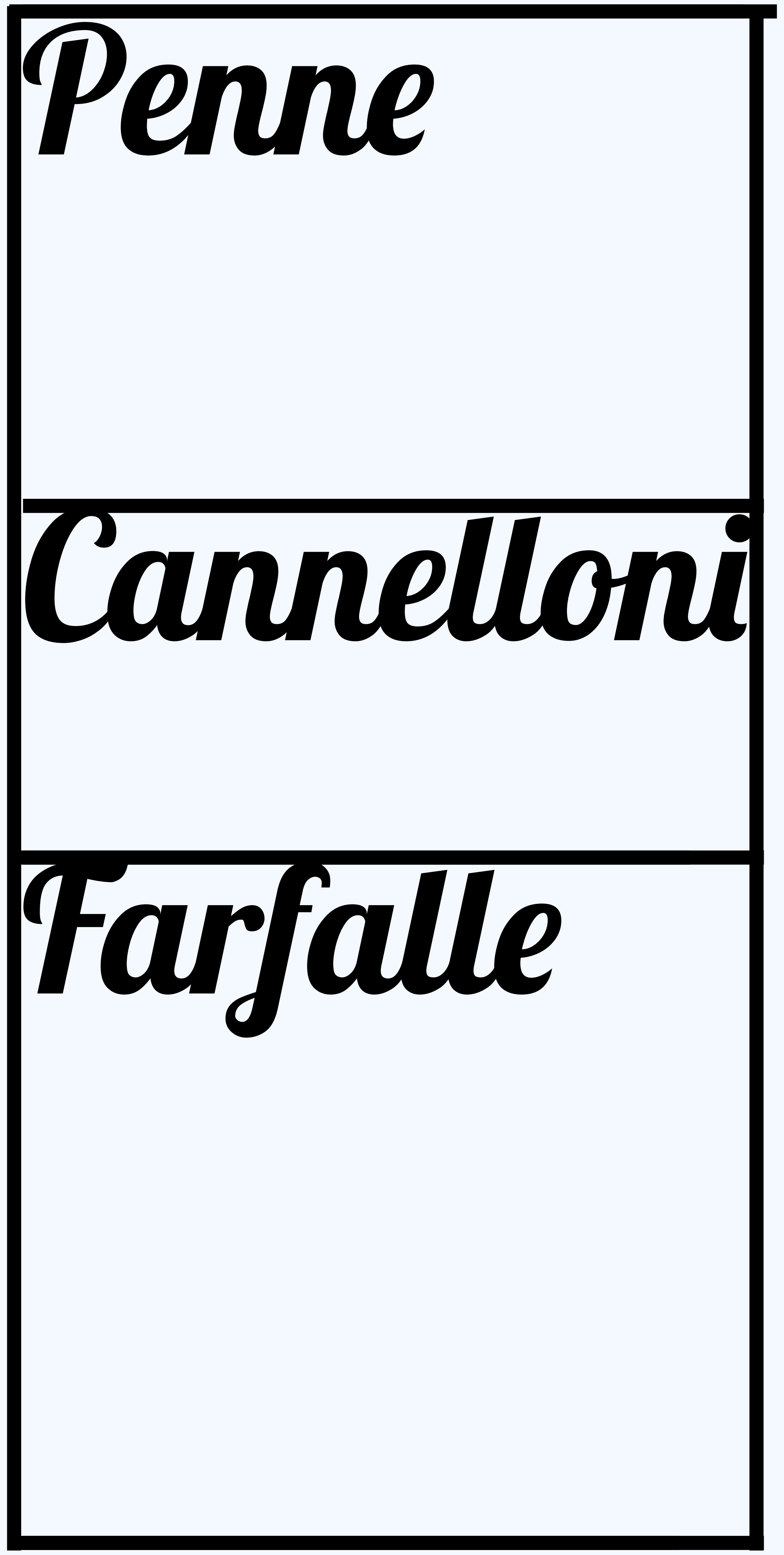
Minimum track size: pick the grid item with the largest minimum* size. That's your track's minimum size.



Auto track size

Minimum track size: pick the grid item with the largest minimum* size. That’s your track’s minimum size.

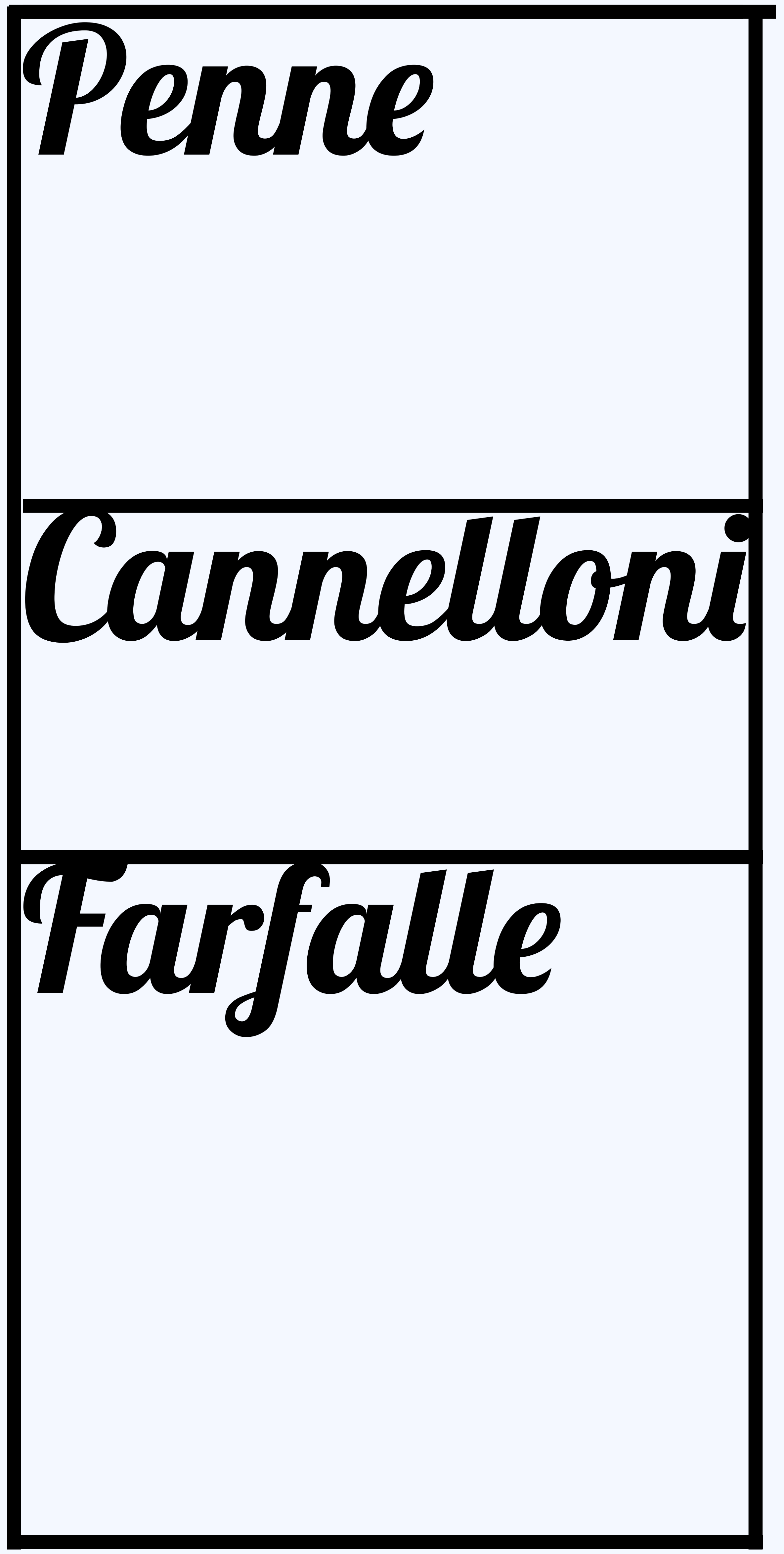
* min-width/min-height value



Auto track size

Minimum track size: pick the grid item with the largest minimum* size. That's your track's minimum size.

* min-width/min-height value
** usually like `min-content`



GRID TRACKS

GRID TRACKS

Size you specified

WHEN

– you use fixed sizing units (relative or absolute)

GRID TRACKS

Size you specified

WHEN

– you use fixed sizing units (relative or absolute)

More than you specified

WHEN

– there's a long word

GRID TRACKS

Size you specified

WHEN

– you use fixed sizing units (relative or absolute)

More than you specified

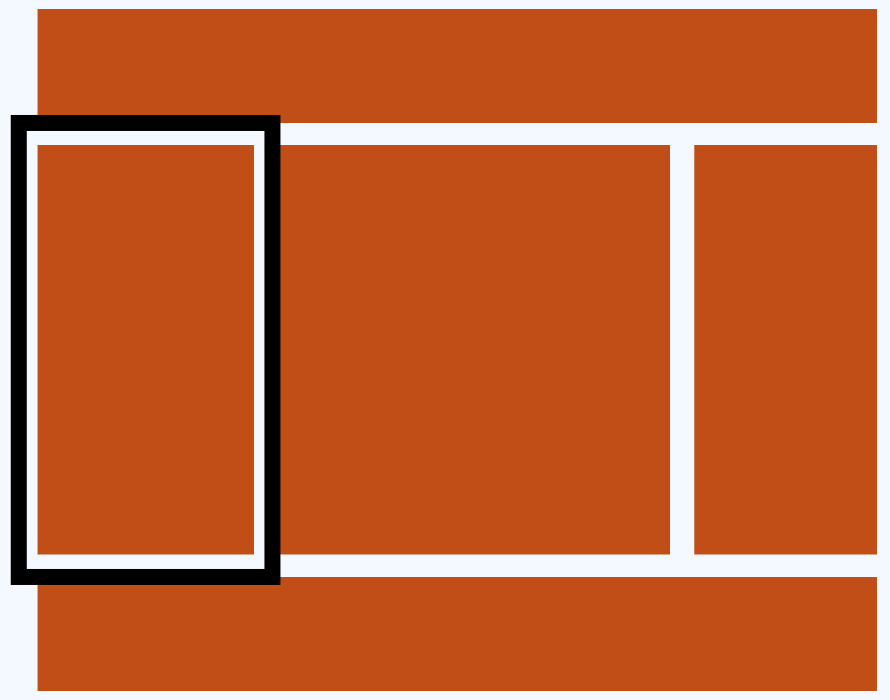
WHEN

– there's a long word

Something perfect

WHEN

– you've given the browser flexibility: keywords, fractions, auto.



GRID ITEMS

Grid item size depends on alignment.

Alignment is what happens when you have more space than content.

Alignment is what happens when you have more space than content.

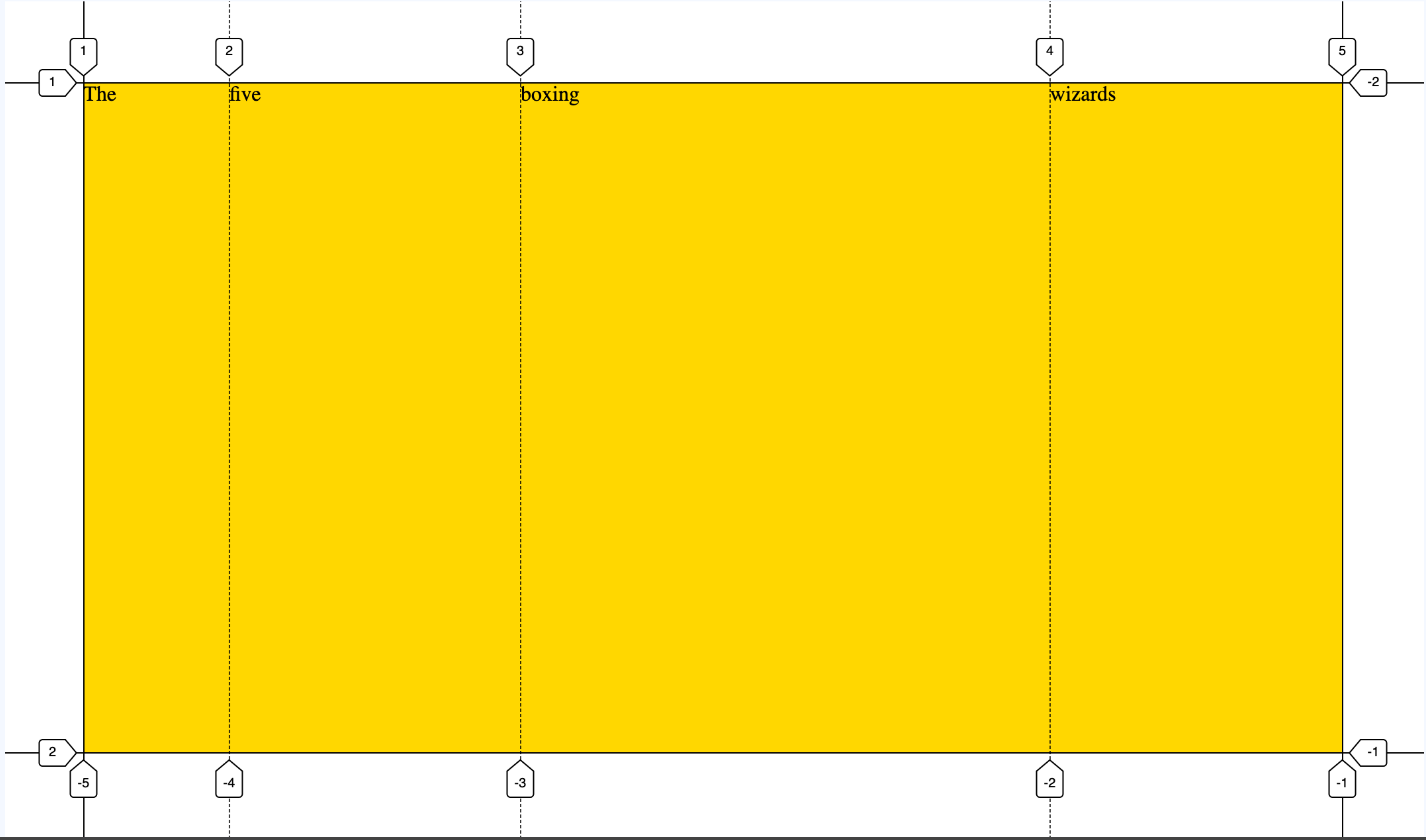
Alignment is what happens when you have more space than content.

Alignment is what happens when you have more space than content. You can only do it if you have more space, because if there is text everywhere, it cannot align.

Alignment is what happens when you have more space than content. You can only do it if you have more space, because if there is text everywhere, it cannot align.

Alignment is what happens when you have more space than content. You can only do it if you have more space, because if there is text everywhere, it cannot align.

A grid with five items

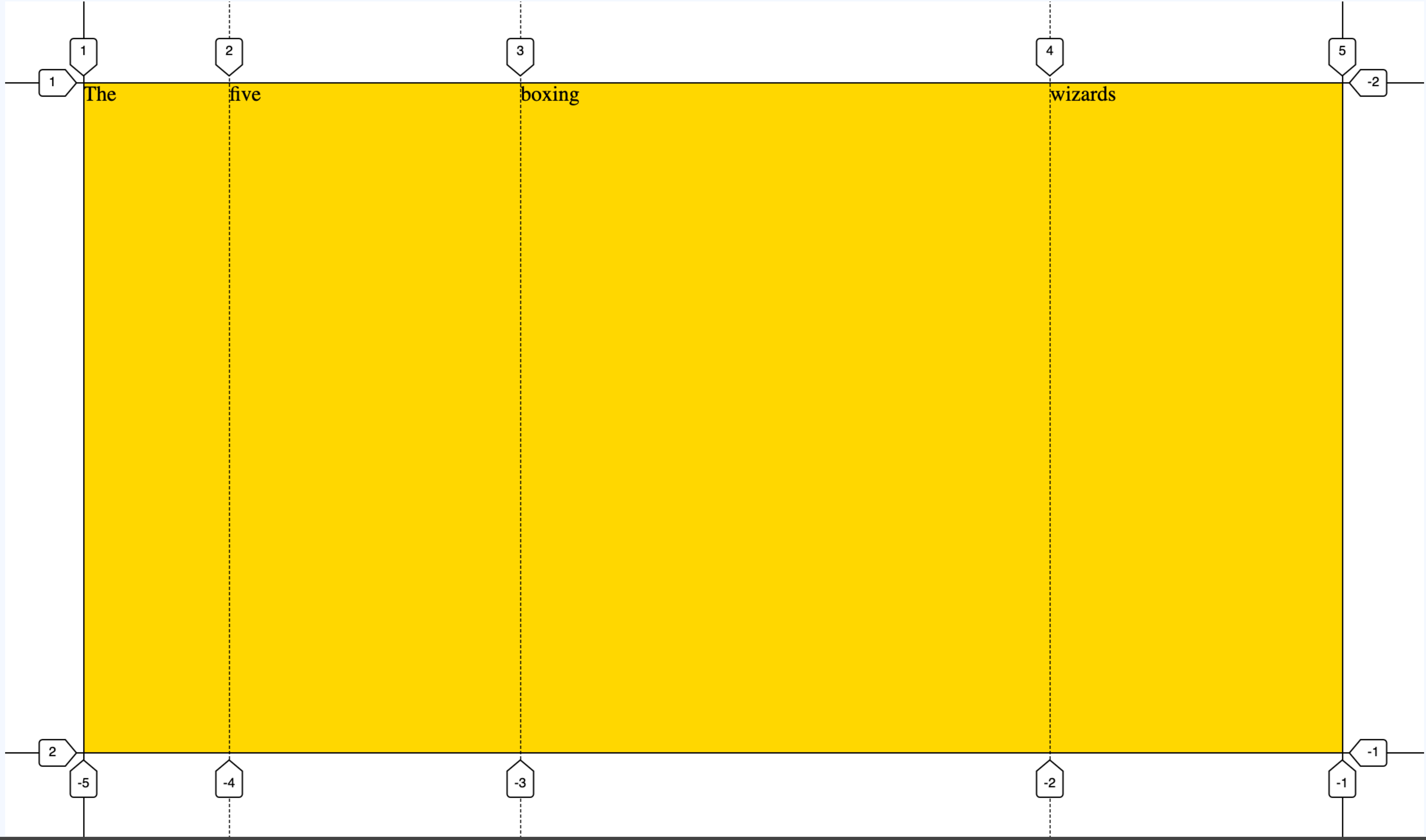


```
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```
div {
  display: grid;
  grid-template-columns: 1fr 2fr 50ch 2fr;
  grid-template-rows: 1fr;
  background-color: gold;
  min-height: 80vh;
}
```

A grid with five items

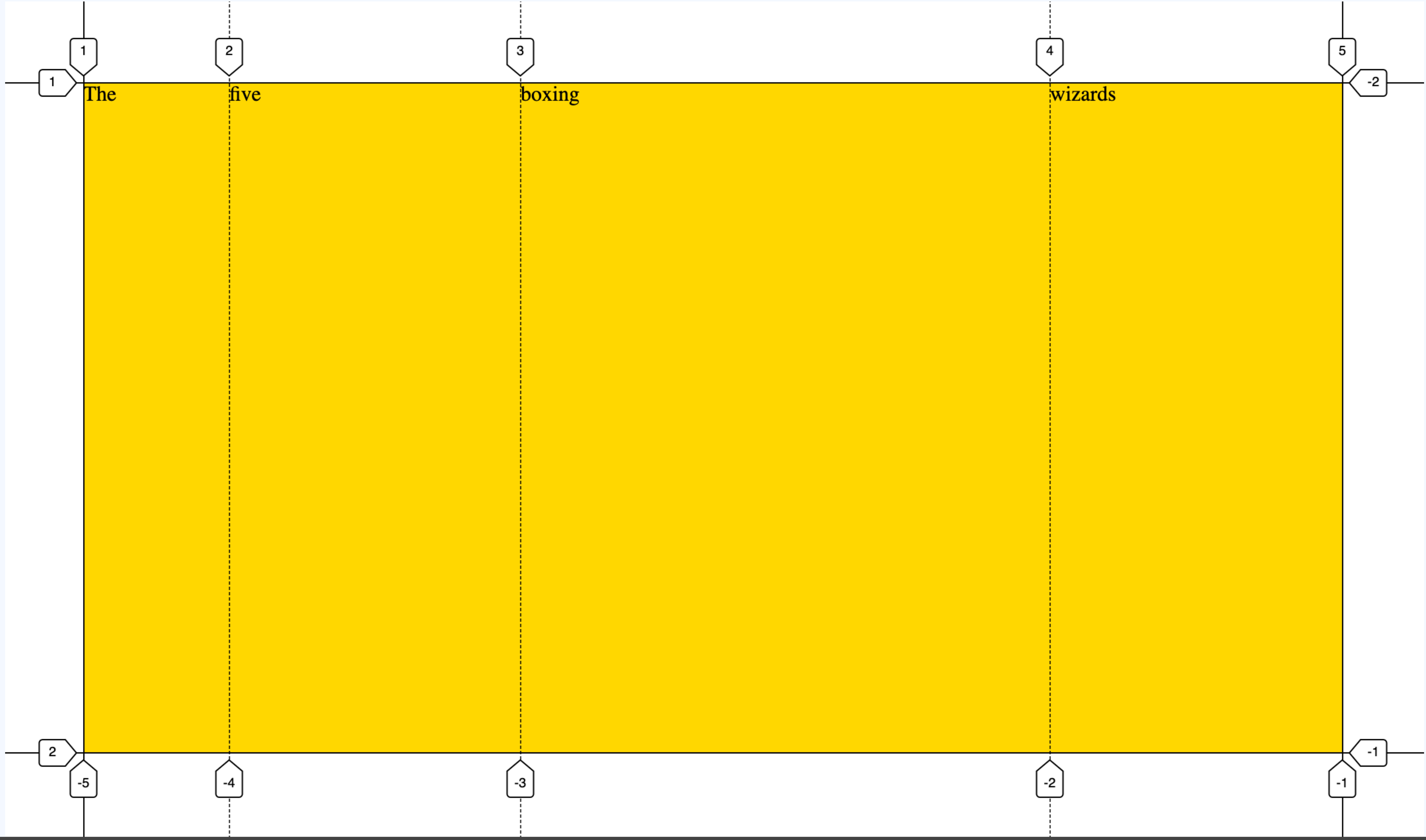
The default alignment is `stretch`, so all whitespace is used.



```html
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```css
div {
  display: grid;
  grid-template-columns: 1fr 2fr 50ch 2fr;
  grid-template-rows: 1fr;
  background-color: gold;
  min-height: 80vh;
}
```

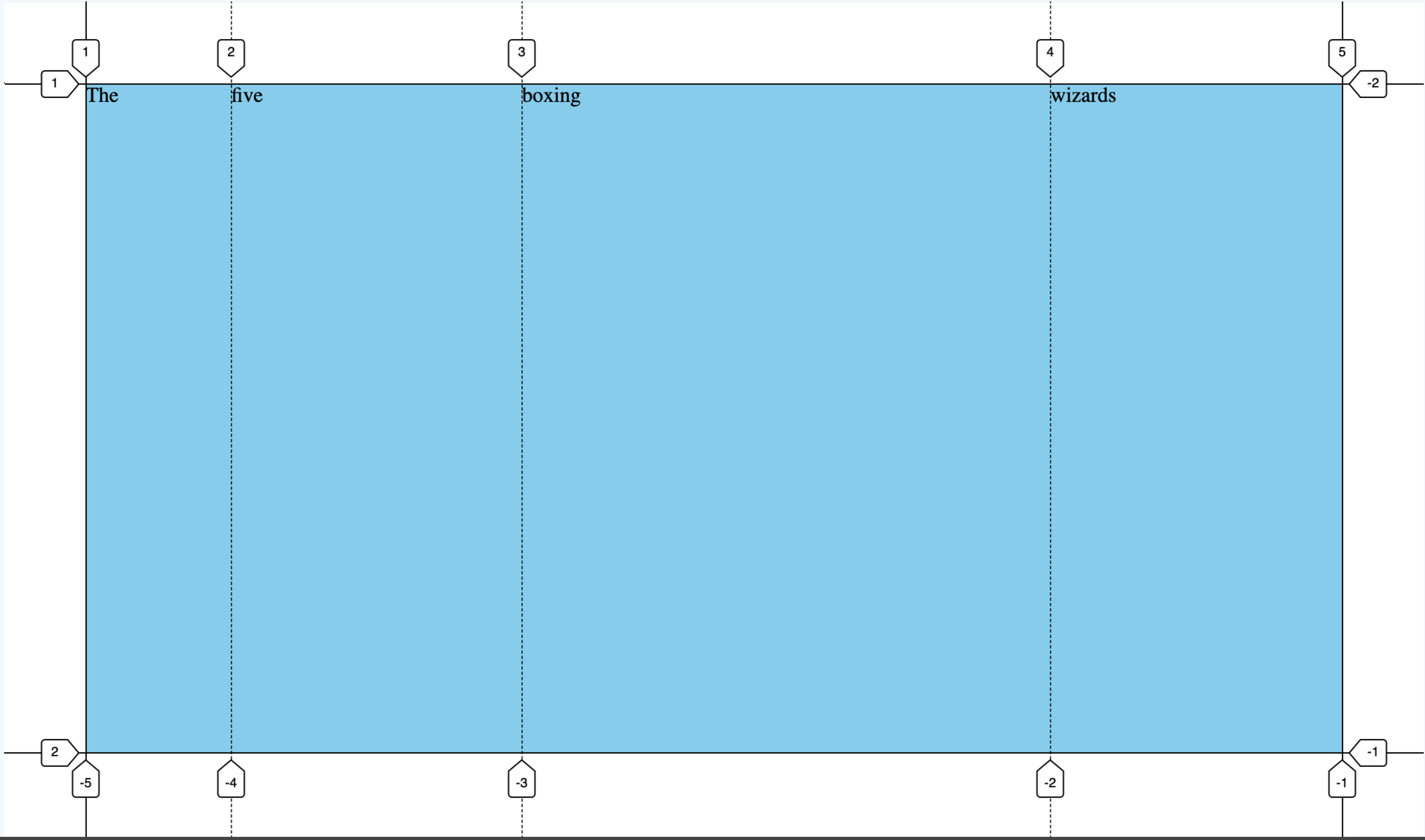
A grid with five items



```
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```
p {
  background-color: skyblue;
}
```

A grid with five items

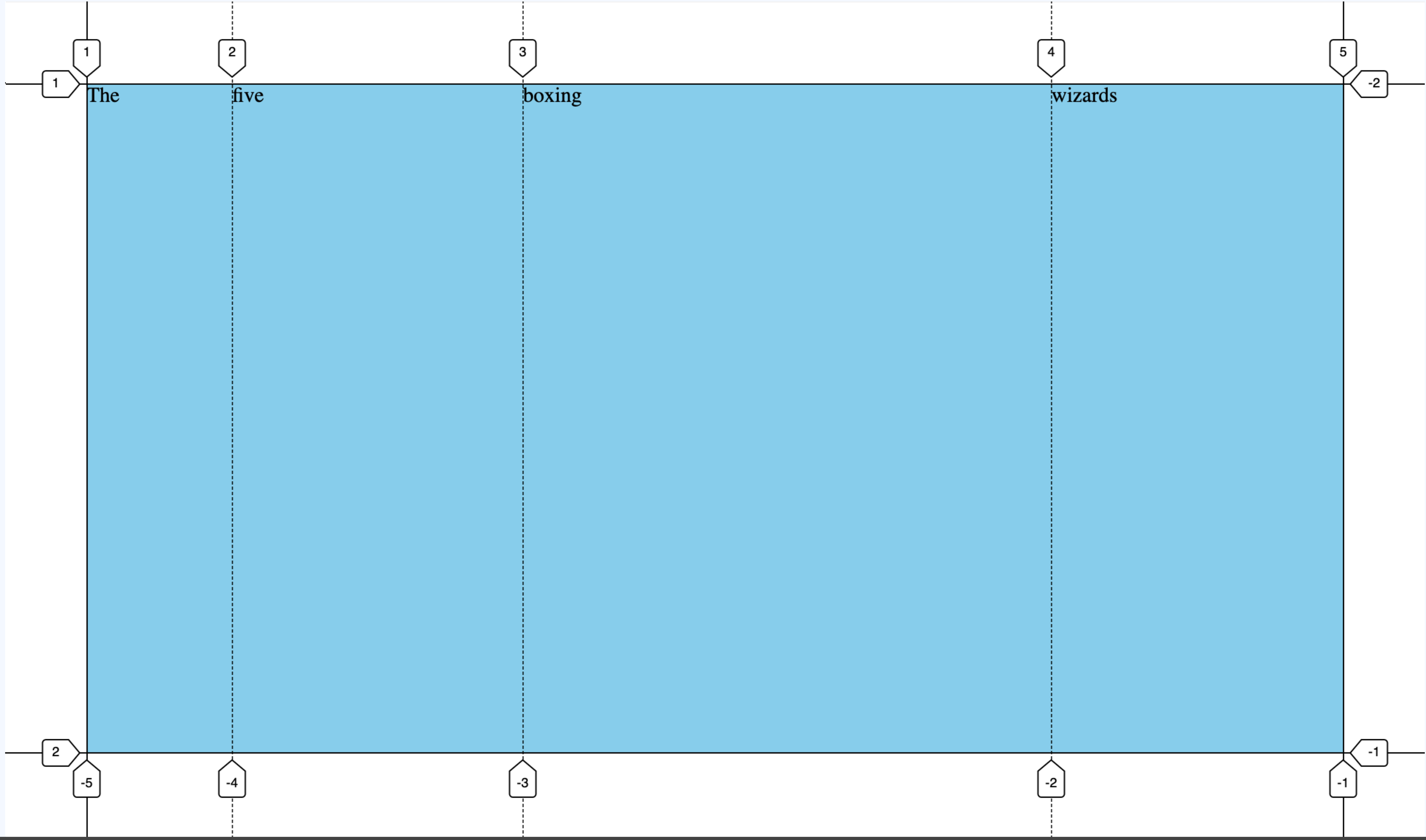


```
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```
p {
  background-color: skyblue;
}
```

A grid with five items

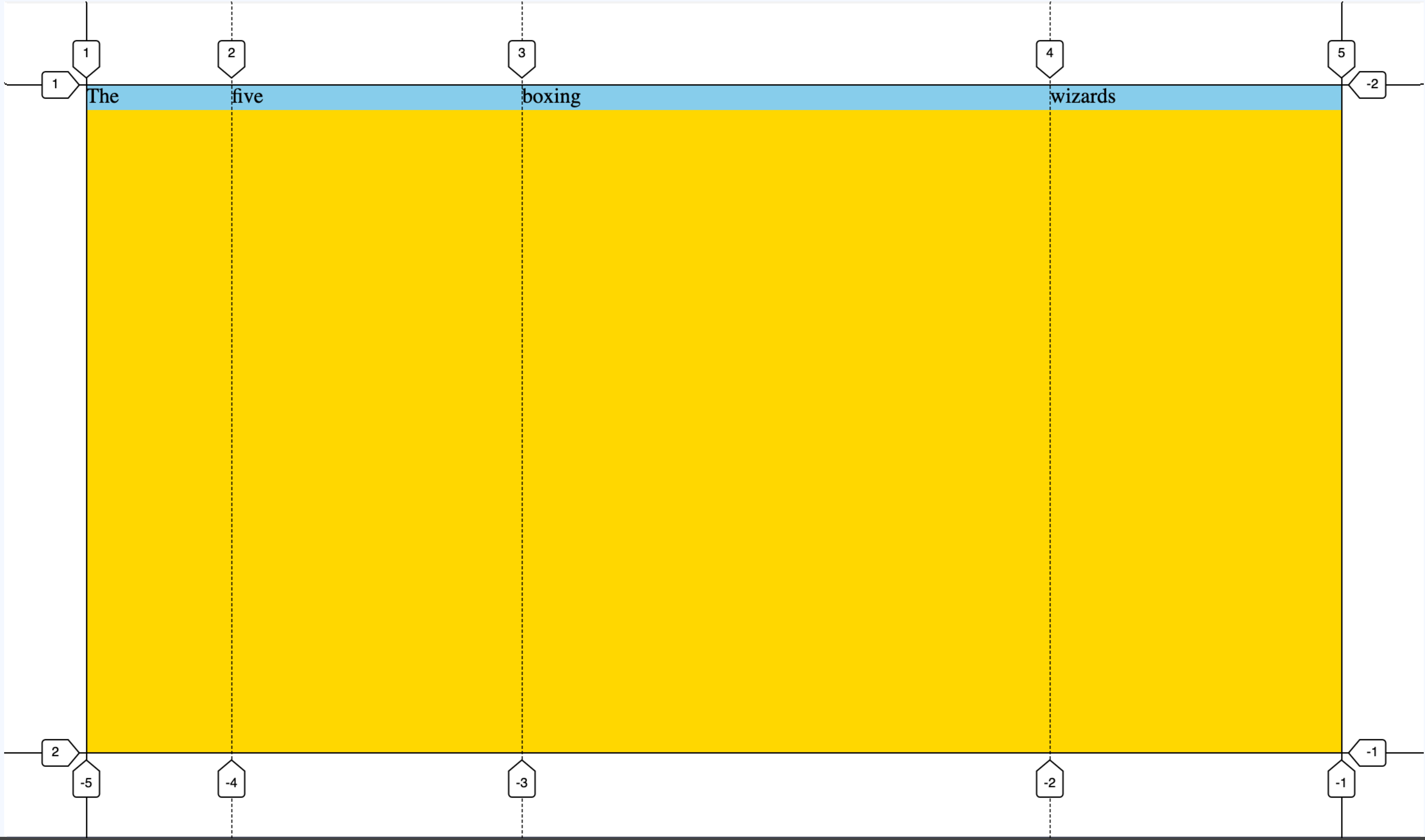
The default alignment is `stretch`, so all whitespace is used.



```
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```
p {
  background-color: skyblue;
}
```

!stretch (start)

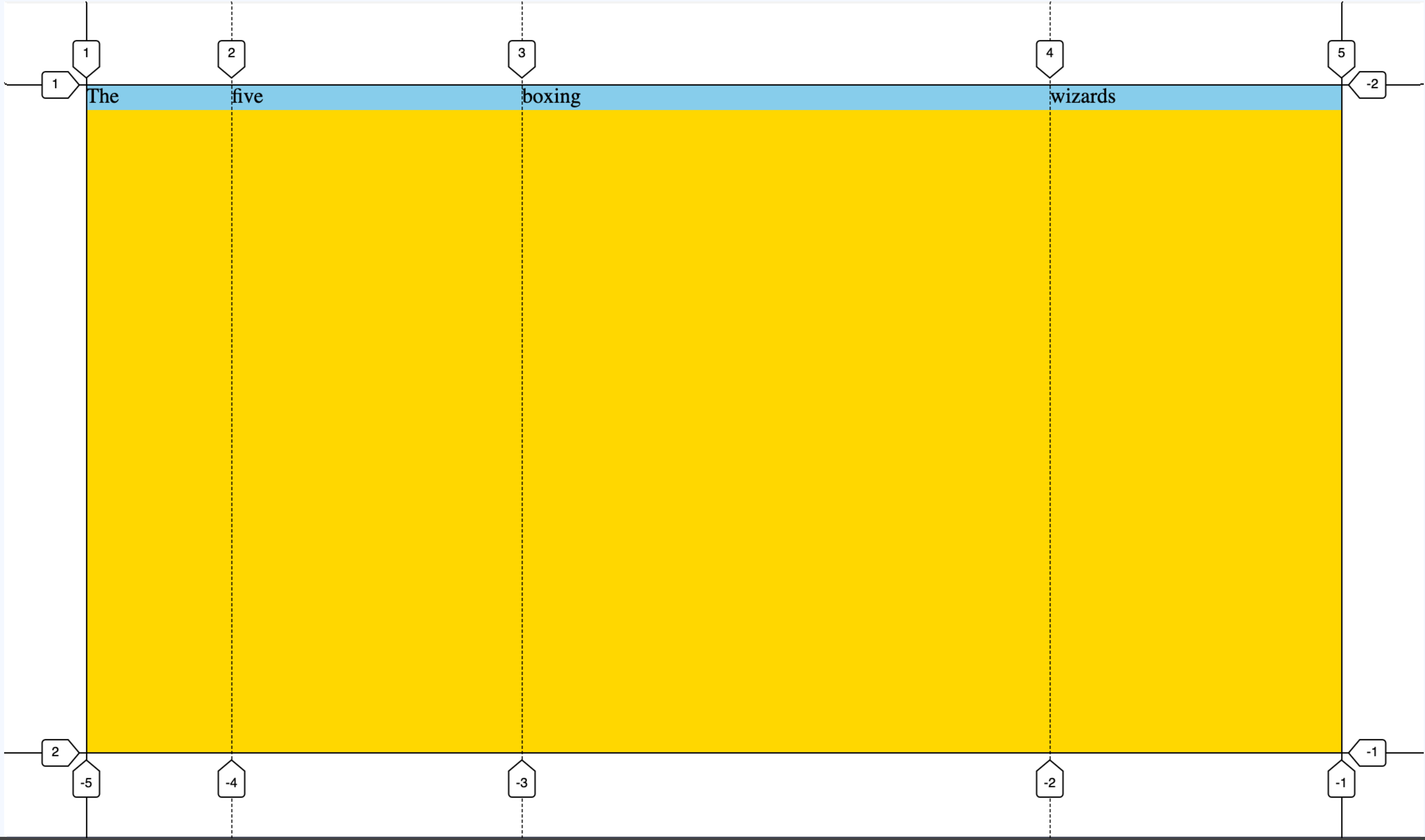


```
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```
p {
  background-color: skyblue;
  align-self: start;
}
```

!stretch (start)

If grid items are aligned other than `stretch`, their size is what is needed.

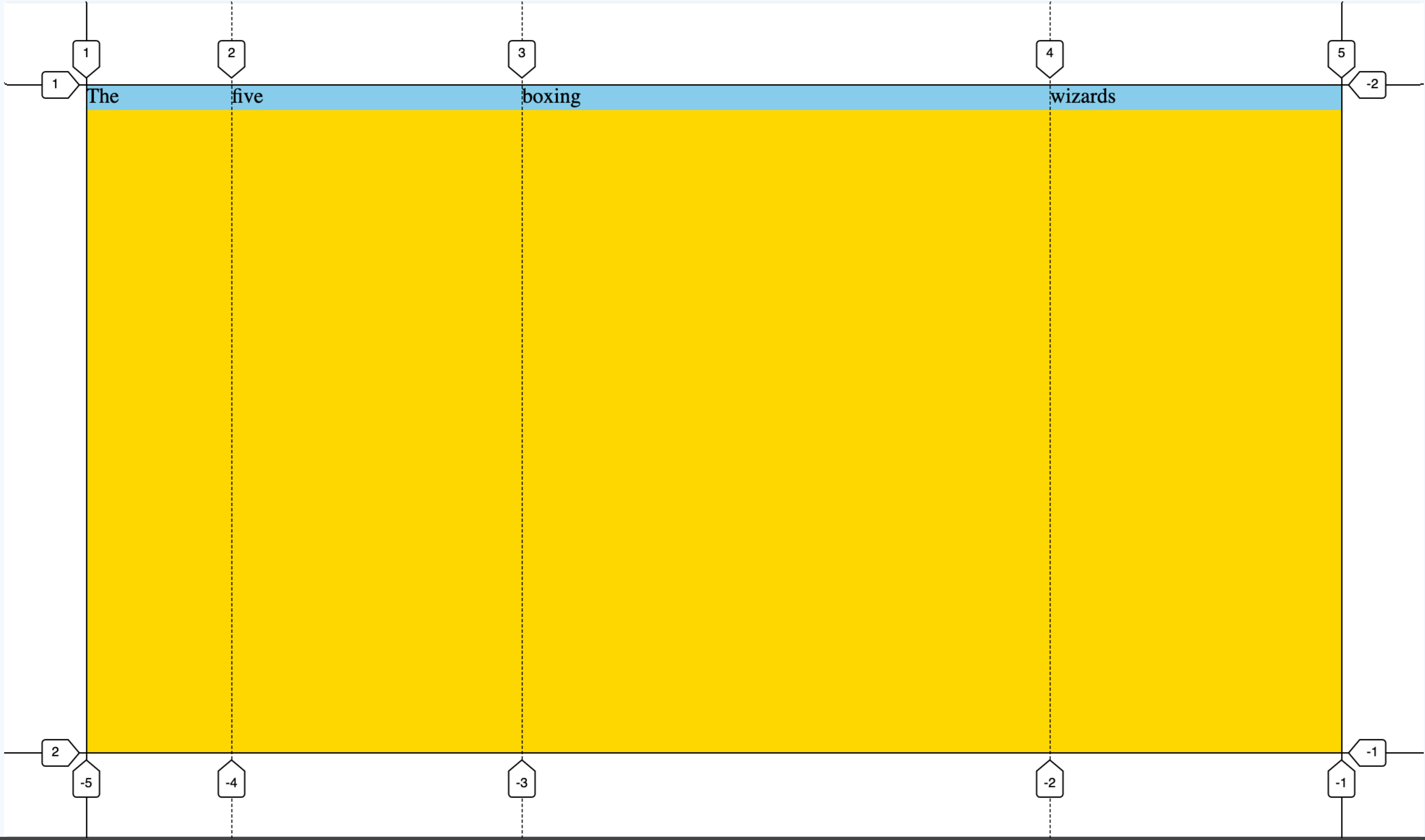


```
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```
p {
  background-color: skyblue;
  align-self: start;
}
```

!stretch (start)

If grid items are aligned other than `stretch`, their size is what is needed.

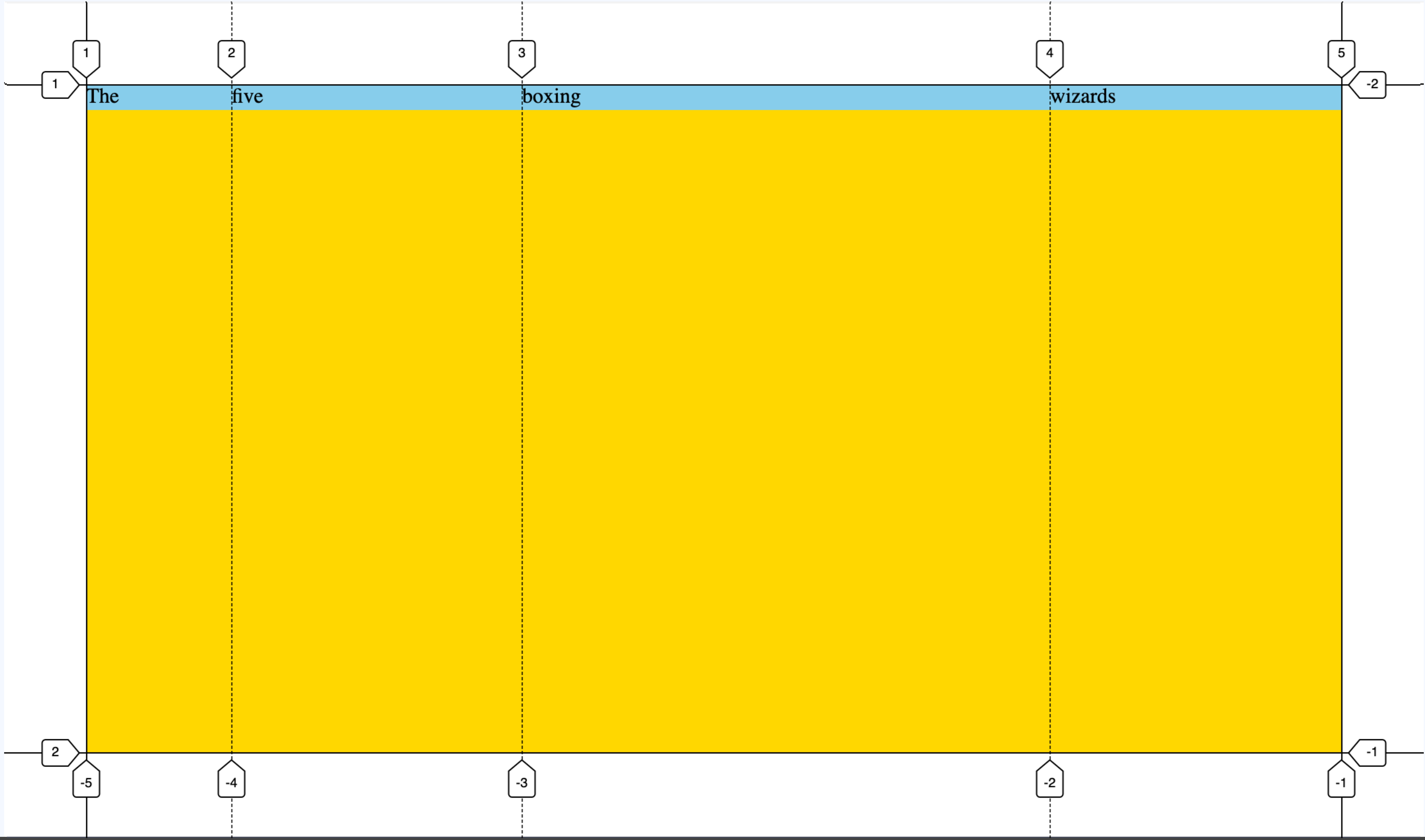


```
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```
p {
  background-color: skyblue;
  margin-bottom: auto;
}
```

!stretch (start)

If grid items are aligned other than `stretch`, their size is what is needed.

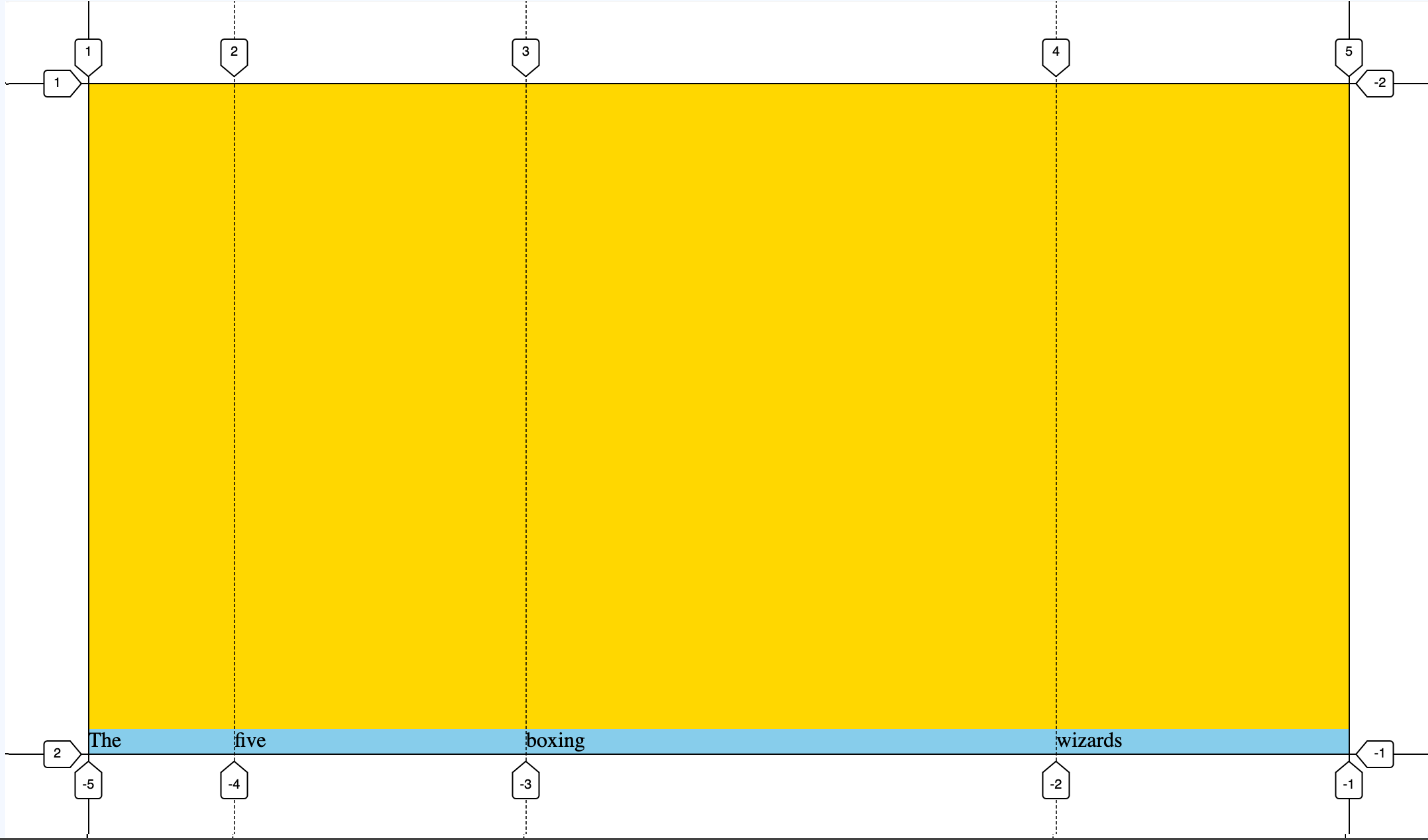


```html
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```css
p {
  background-color: skyblue;
  margin-block-end: auto;
}
```

!stretch (end)

If grid items are aligned other than `stretch`, their size is what is needed.



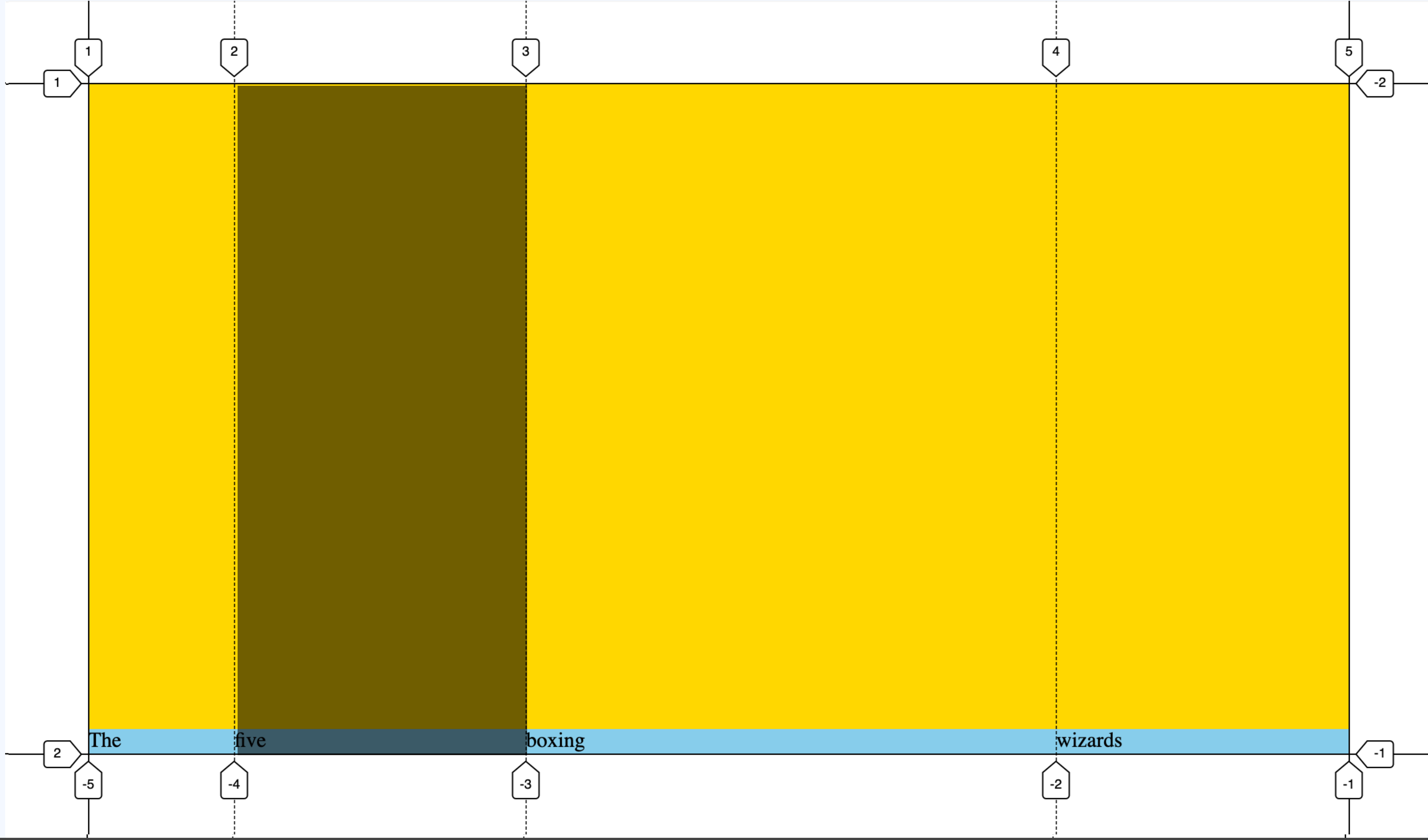
```html
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```css
p {
  background-color: skyblue;
  align-self: end;
}
```

!stretch (end)

If grid items are aligned other than `stretch`, their size is what is needed.

The cell they are in can still be larger.



```
<div>
  <p>The</p>
  <p>five</p>
  <p>boxing</p>
  <p>wizards</p>
</div>
```

```
p {
  background-color: skyblue;
  align-self: end;
}
```

GRID ITEMS

GRID ITEMS

Size of track

WHEN

– item is aligned "stretch"

GRID ITEMS

Size of track

WHEN

– item is aligned "stretch"

Size that the content needs

WHEN

– item is aligned in a way that leaves whitespace

GRID ITEMS

Size of track

WHEN

– item is aligned "stretch"

Size that the content needs

WHEN

– item is aligned in a way that leaves whitespace, or has an auto margin that returns whitespace to the track

CONCLUDING...

Grid Layout helps with more international CSS, by being less physical.

Grid Layout helps with more international CSS, by being less physical.

For websites, letting the browser decide flexibly, can make your life easier.

MORE READING/WATCHING

- Tab Atkins Jr., Elika J. Etemad / fantasai, Rossen Atanassov "CSS Grid Layout Module Level 2" (https://www.w3.org/TR/css-grid-2/)
- Rachel Andrew, "How Big Is That Box? Understanding Sizing In CSS Layout" (https://www.smashingmagazine.com/2018/01/understanding-sizing-css-layout/)
- Elika J. Etemad (fantasai), "Defining auto" (https://vimeo.com/134597090)
- Hui Jing Chen, "Vertical typesetting with writing-mode revisited" (https://chenhuijing.com/blog/vertical-typesetting-revisited/#%F0%9F%91%9F)
- Rachel Andrew, "Writing Modes and CSS Layout" (https://www.smashingmagazine.com/2019/08/writing-modes-layout/)
- Jen Simmons, "CSS Writing Modes" (https://24ways.org/2016/css-writing-modes/)

MORE READING/WATCHING

- Tab Atkins Jr., Elika J. Etemad / fantasai, Rossen Atanassov "CSS Grid Layout Module Level 2" (https://www.w3.org/TR/css-grid-2/)
- Rachel Andrew, "How Big Is That Box? Understanding Sizing In CSS Layout" (https://www.smashingmagazine.com/2018/01/understanding-sizing-css-layout/)
- Elika J. Etemad (fantasai), "Defining auto" (https://vimeo.com/134597090)
- Hui Jing Chen, "Vertical typesetting with writing-mode revisited" (https://chenhuijing.com/blog/vertical-typesetting-revisited/#%F0%9F%91%9F)
- Rachel Andrew, "Writing Modes and CSS Layout" (https://www.smashingmagazine.com/2019/08/writing-modes-layout/)
- Jen Simmons, "CSS Writing Modes" (https://24ways.org/2016/css-writing-modes/)

SLIDES: HTTPS://TALKS.HIDDEDEVRIES.NL QUESTIONS: @HDV

THANK YOU!