CSS Grid Layout

Robust Layout Using Rows & Columns

R. Scott Granneman & Jans Carton

Notes & URLs for this presentation can be found…

- underneath the link to this slide show on granneman.com
- at files.granneman.com/presentations/webdev/CSS-Layout.txt

W3C Candidate Recommendation

CSS Grid Layout Module Level 1

W3C Candidate Recommendation, 14 December 2017

This version:

https://www.w3.org/TR/2017/CR-css-grid-1-20171214/

Latest published version:

https://www.w3.org/TR/css-grid-1/

Editor's Draft:

https://drafts.csswg.org/css-grid/

Previous Versions:

https://www.w3.org/TR/2017/CR-css-grid-1-20170509/
https://www.w3.org/TR/2016/WD-css-grid-1-20160519/
https://www.w3.org/TR/2015/WD-css-grid-1-20150917/
https://www.w3.org/TR/2015/WD-css-grid-1-20150806/
https://www.w3.org/TR/2015/WD-css-grid-1-20150317/
https://www.w3.org/TR/2014/WD-css-grid-1-20140513/
https://www.w3.org/TR/2014/WD-css-grid-1-20140123/
https://www.w3.org/TR/2013/WD-css3-grid-layout-20130402/
https://www.w3.org/TR/2012/WD-css3-grid-layout-20121106/

Test Suite:

http://test.csswg.org/suites/css-grid-1_dev/nightly-unstable/

Issue Tracking:

Disposition of Comments
Inline In Spec
GitHub Issues

CSS Grid Layout - CR

Method of using a grid concept to lay out content, providing a mechanism for authors to divide available space for layout into columns and rows using a set of predictable sizing behaviors. Includes support for all `grid-*` properties and the `fr` unit.

Usage % of all users

Global 84.95% + 2.91% = 87.85%

unprefixed: 84.95%

As of October 2018

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Blackberry Browser	Opera Mobile *	Chrome for Android	Firefox for Android	IE M
		2-39	4-28										
		3 40-51	1 29-56		10-27								
6-9	2 12-15	4 52-53	4 57	3.1-10	1 28-43	3.2-10.2							
2 10	16	54-61	58-69	10.1-11.1	44-55	10.3-11.2		2.1-4.4.4	7	12-12.1			2
2 11	17	62	70	12	56	11.4	all	67	10	46	69	62	2
	18	63-64	71-73	TP		12							

Notes | Known issues (2) | Resources (13) | Feedback

1 Enabled in Chrome through the "experimental Web Platform features" flag in chrome://flags

2 Partial support in IE refers to supporting an older version of the specification.

CSS Grid Layout (level 1) - CR

Method of using a grid concept to lay out content, providing a mechanism for authors to divide available space for layout into columns and rows using a set of predictable sizing behaviors. Includes support for all `grid-*` properties and the `fr` unit.

Usage | % of all users | ?

Global 93.52% + 1.48% = 95%

unprefixed: 93.52%

As of March 2020

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	San Int
		2-39	4-28										
		3 40-51	1 29-56		10-27								
6-9	2 12-15	4 52-53	4 57	3.1-10	1 28-43	3.2-10.2							4-
2 10	16-79	54-72	58-79	10.1-12.1	44-65	10.3-13.2		2.1-4.4.4	12-12.1				6.2-
2 11	80	73	80	13	66	13.3	all	80	46	80	68	12.12	1
		74-75	81-83	TP									

Notes | Known issues (3) | Resources (10) | Feedback

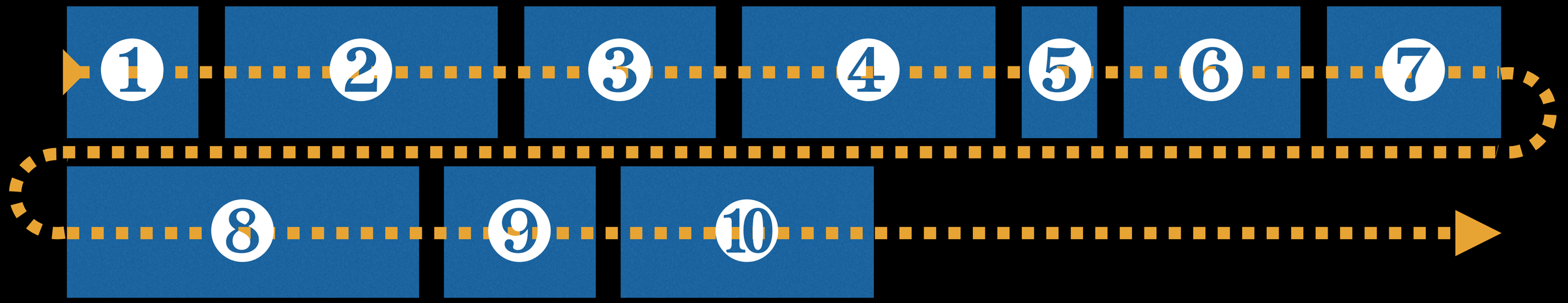
See also support for subgrids

1 Enabled in Chrome through the "experimental Web Platform features" flag in chrome://flags

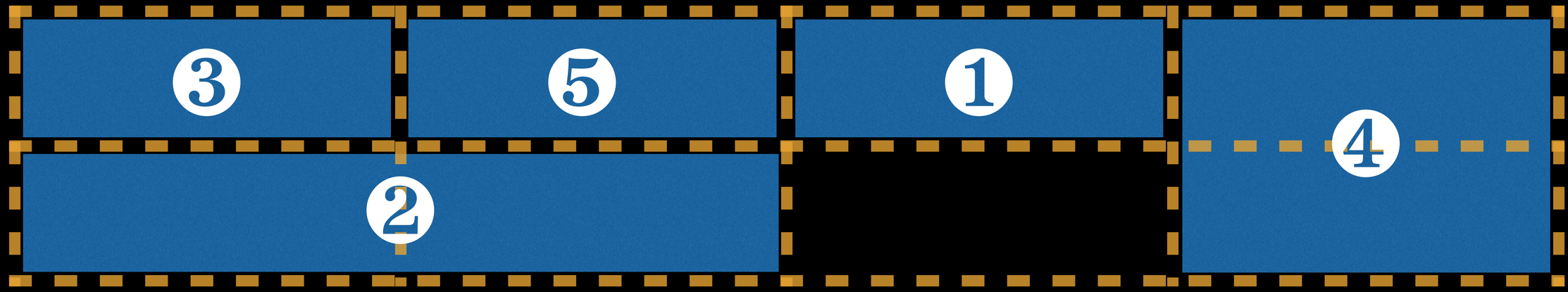
Flexbox vs Grid

Flexbox is for laying out elements in a particular direction along a (sometimes wrapped) line

Grid assigns objects within a matrix of columns & rows



Flexbox



Grid

With every other layout method except grid, you can visualize the layout itself via the HTML

Grid, however, defines all layout in the CSS — the HTML doesn’t necessarily tell you anything about the actual rendered layout other than certain grid items exist inside a grid container

Grid is ultimately about using CSS to define layout scaffolding & then placing rendered boxes onto that scaffolding

Dev tools for grid allow you to inspect something besides the DOM items you've seen with other layouts; instead, with grid you see the layout structures

Concepts & Terms

Grid *container* defines the grid structure

Grid is composed of *lines*, *cells*, *areas*, & *tracks*

Grid *items* are placed into areas

Grid *gutters* are thick lines between tracks

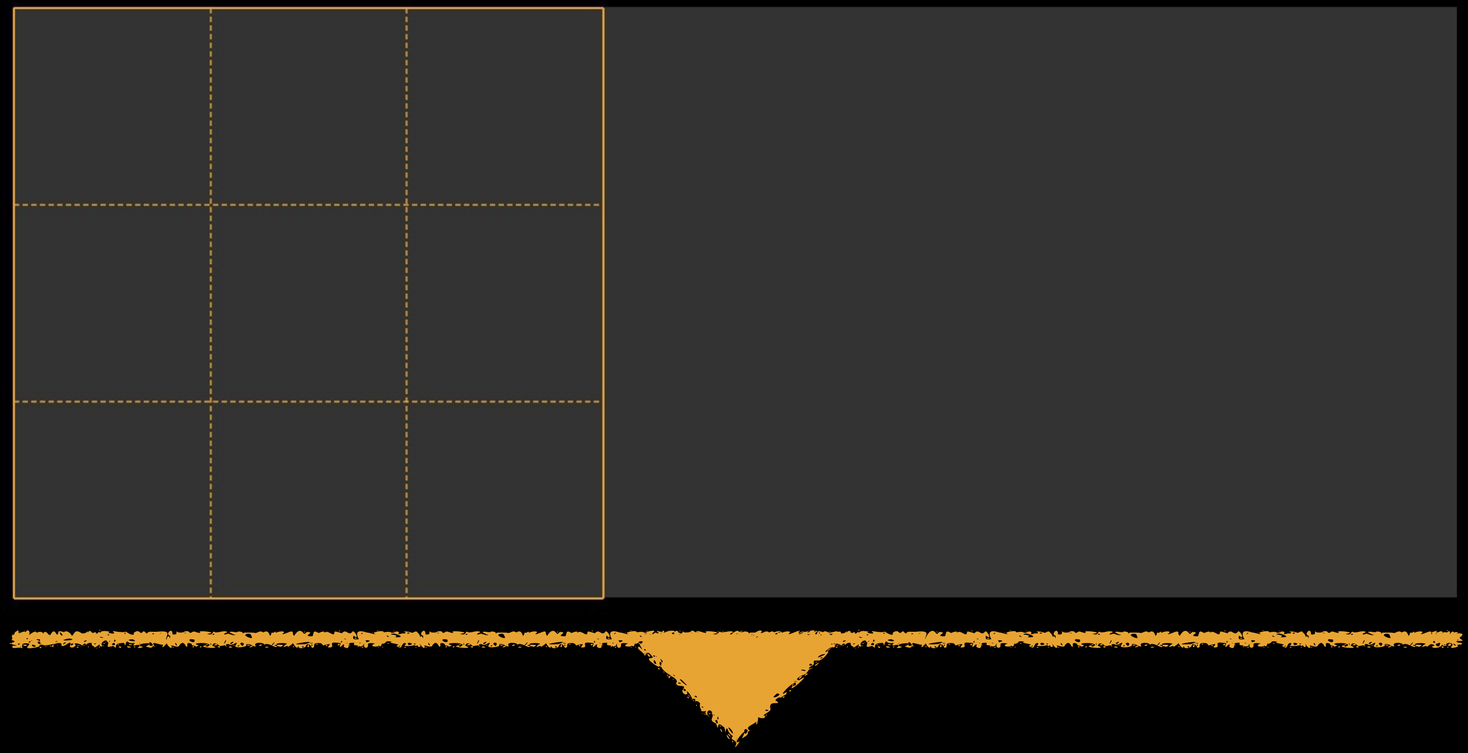
HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



Grid container

» creates a *grid layout context*

» can be bigger (or smaller) than the grid itself

HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS

Grid lines divide the grid, & they are key to understanding grid layout

This grid has 8 lines

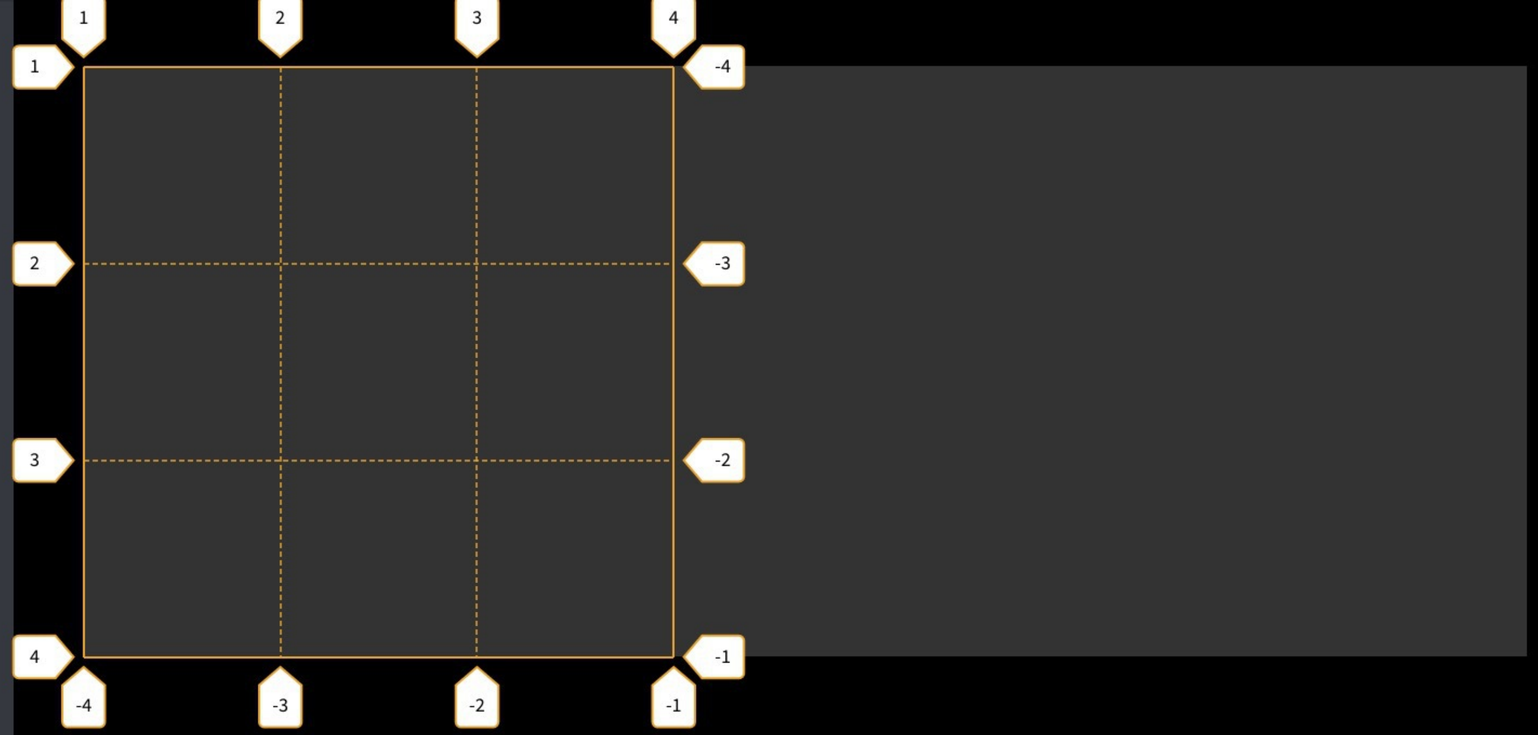
HTML

```
<div class="grid-container">

</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



Grid lines are numbered

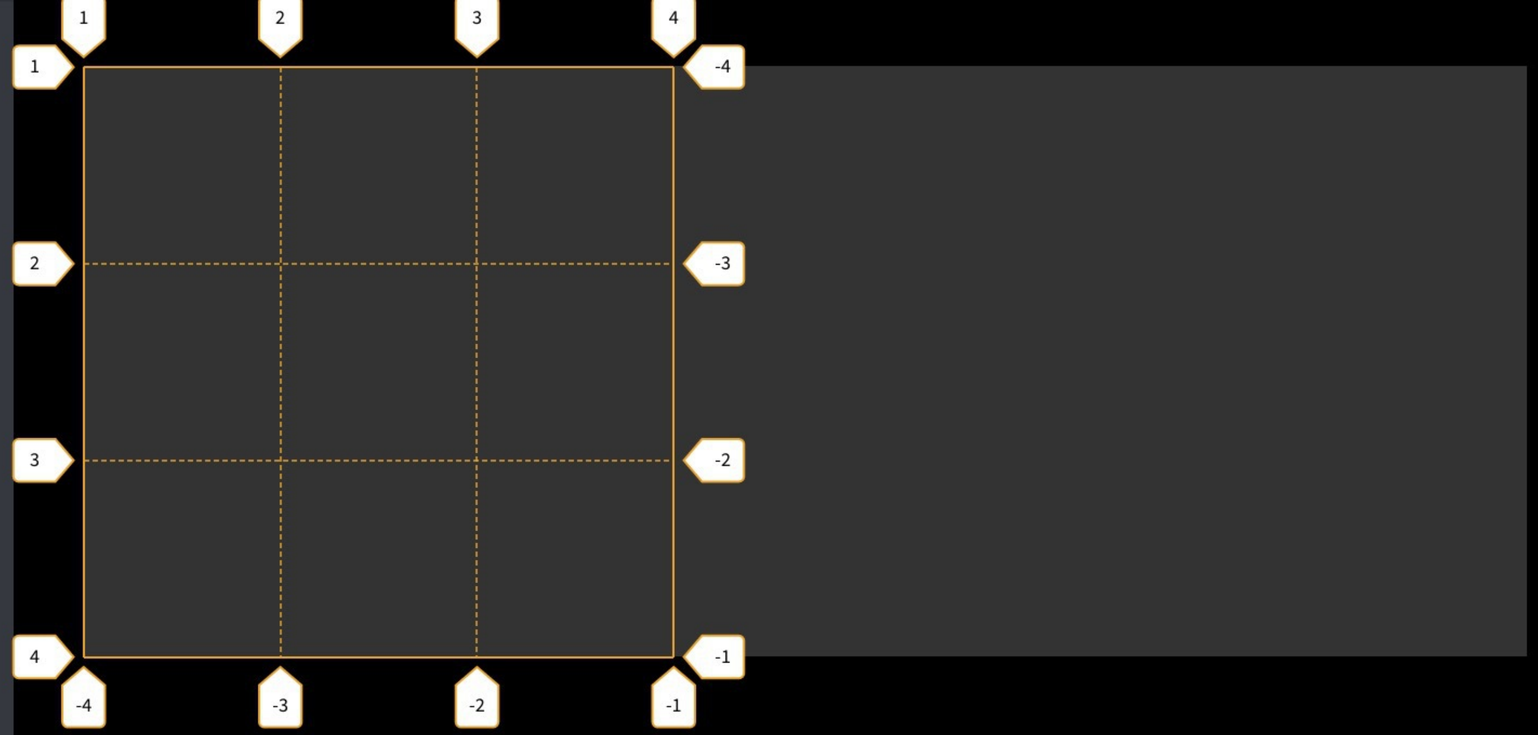
HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



Space between 4 adjacent grid lines defines a *grid cell*

This grid has 9 cells

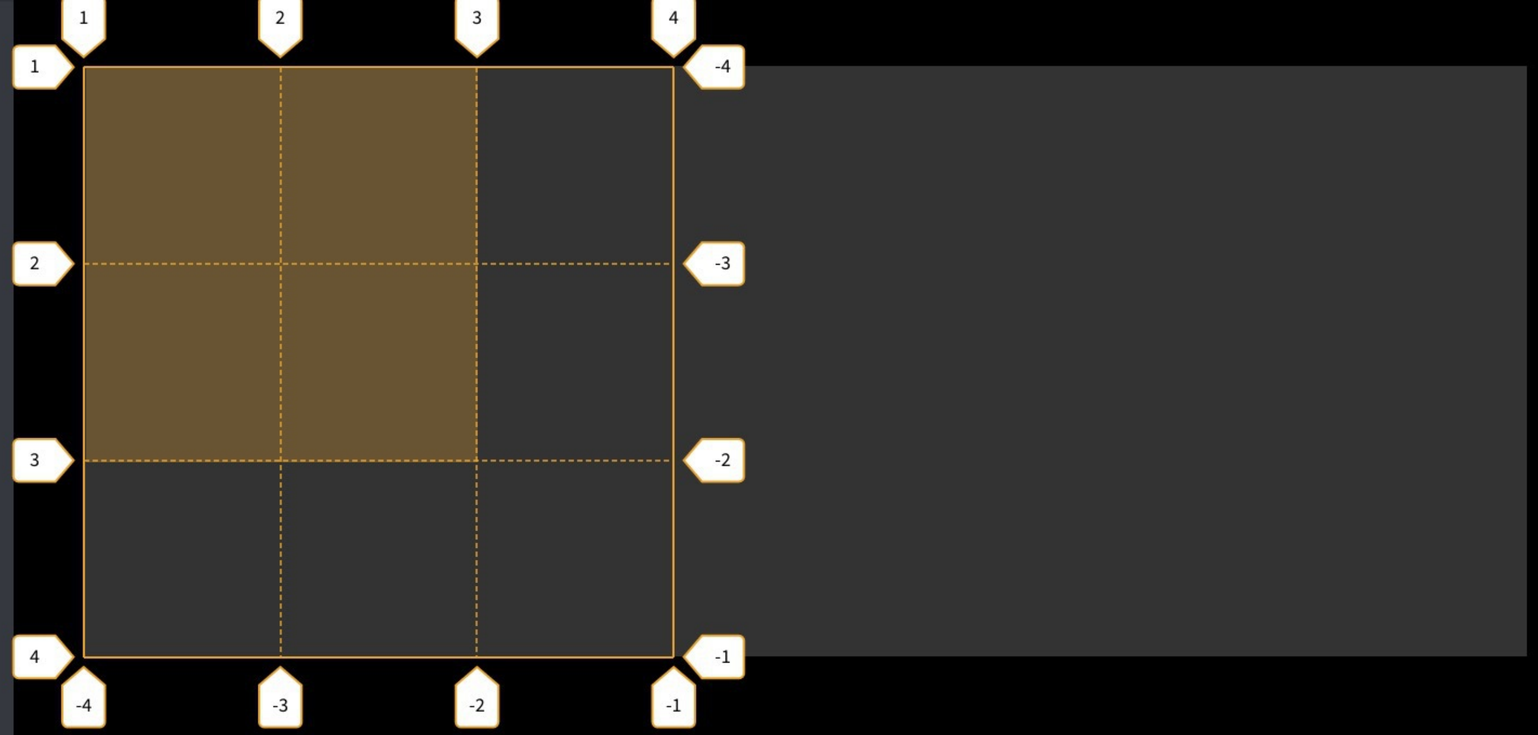
HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



Grid area is defined by 4 (not necessarily adjacent) grid lines

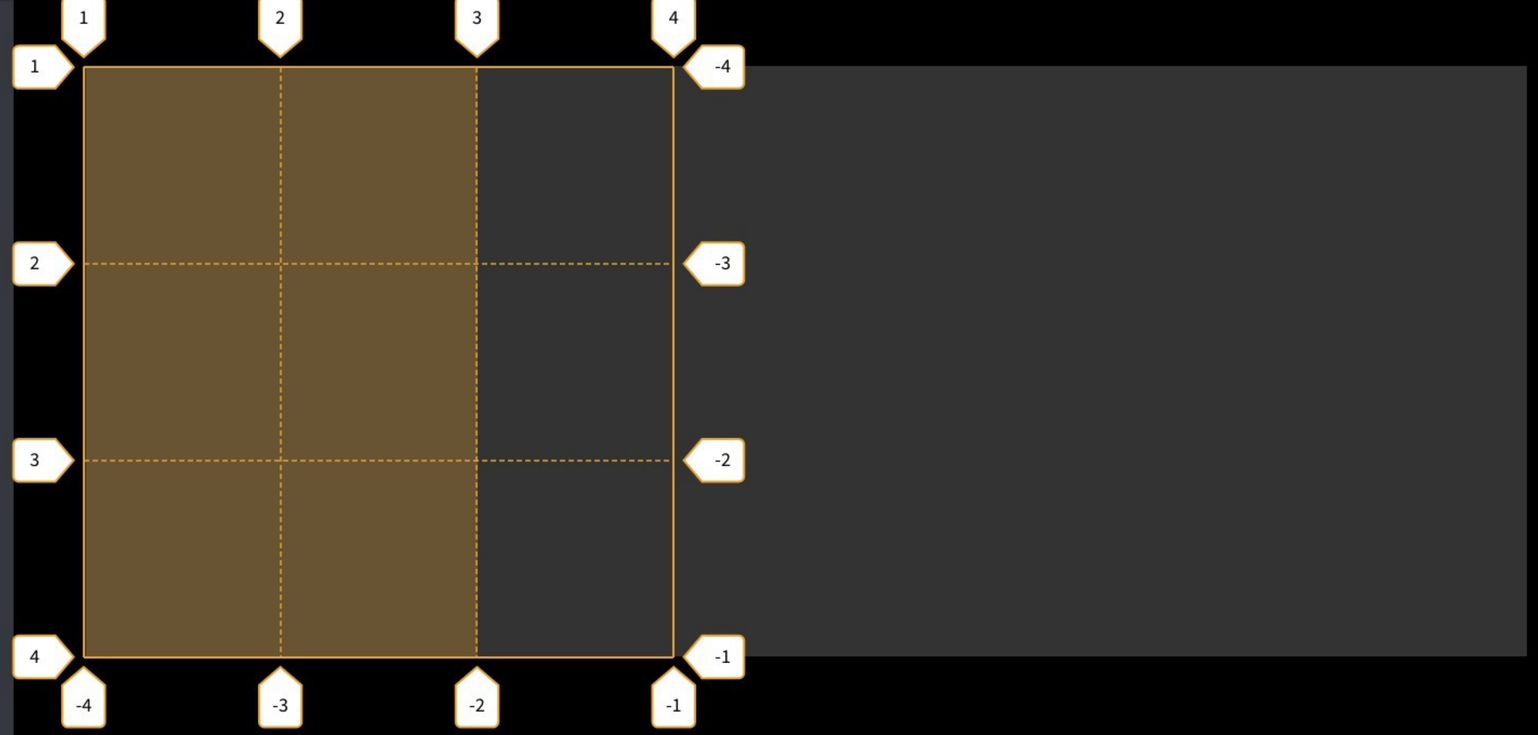
HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



Grid area is surrounded by 4 grid lines around any number of cells

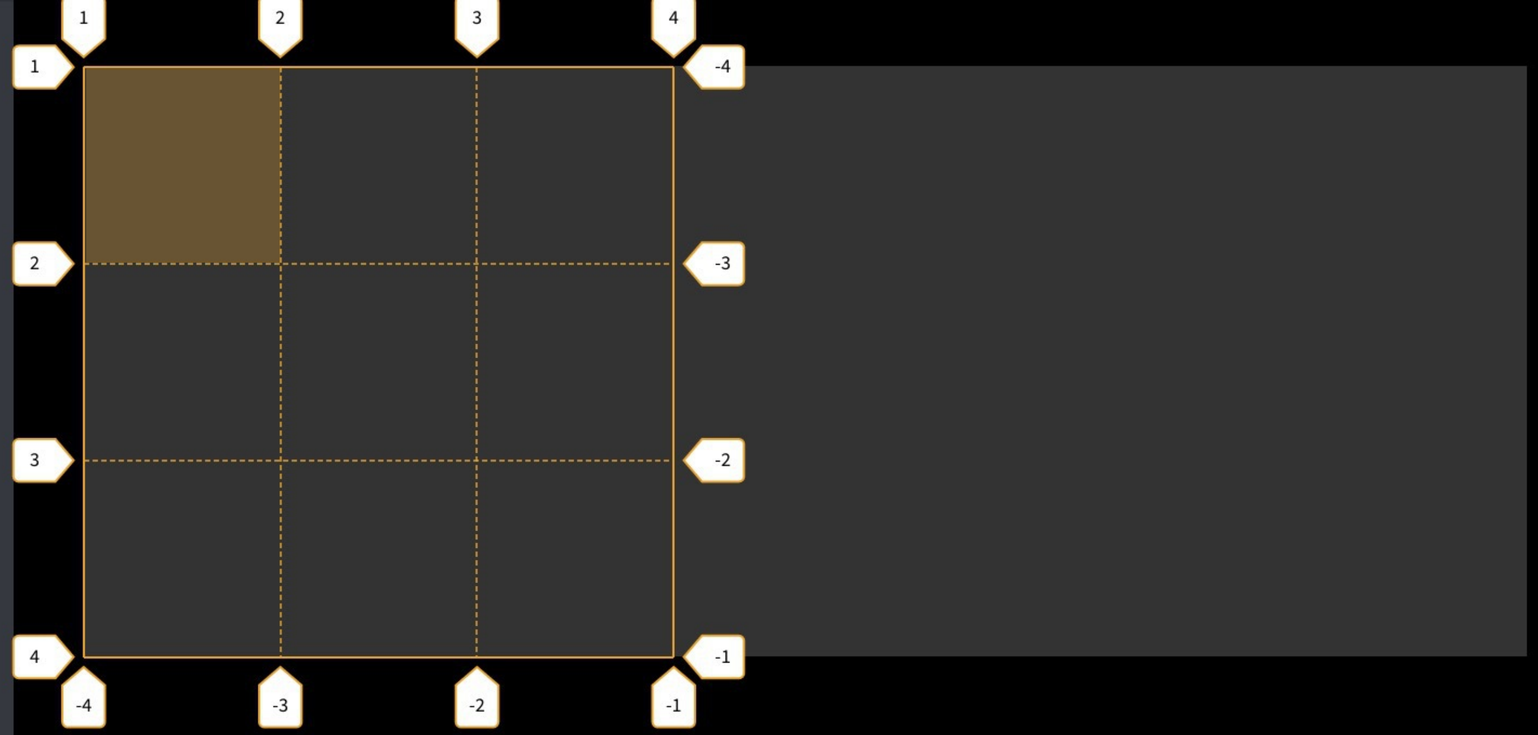
HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



A cell is an area, but not all areas are cells!

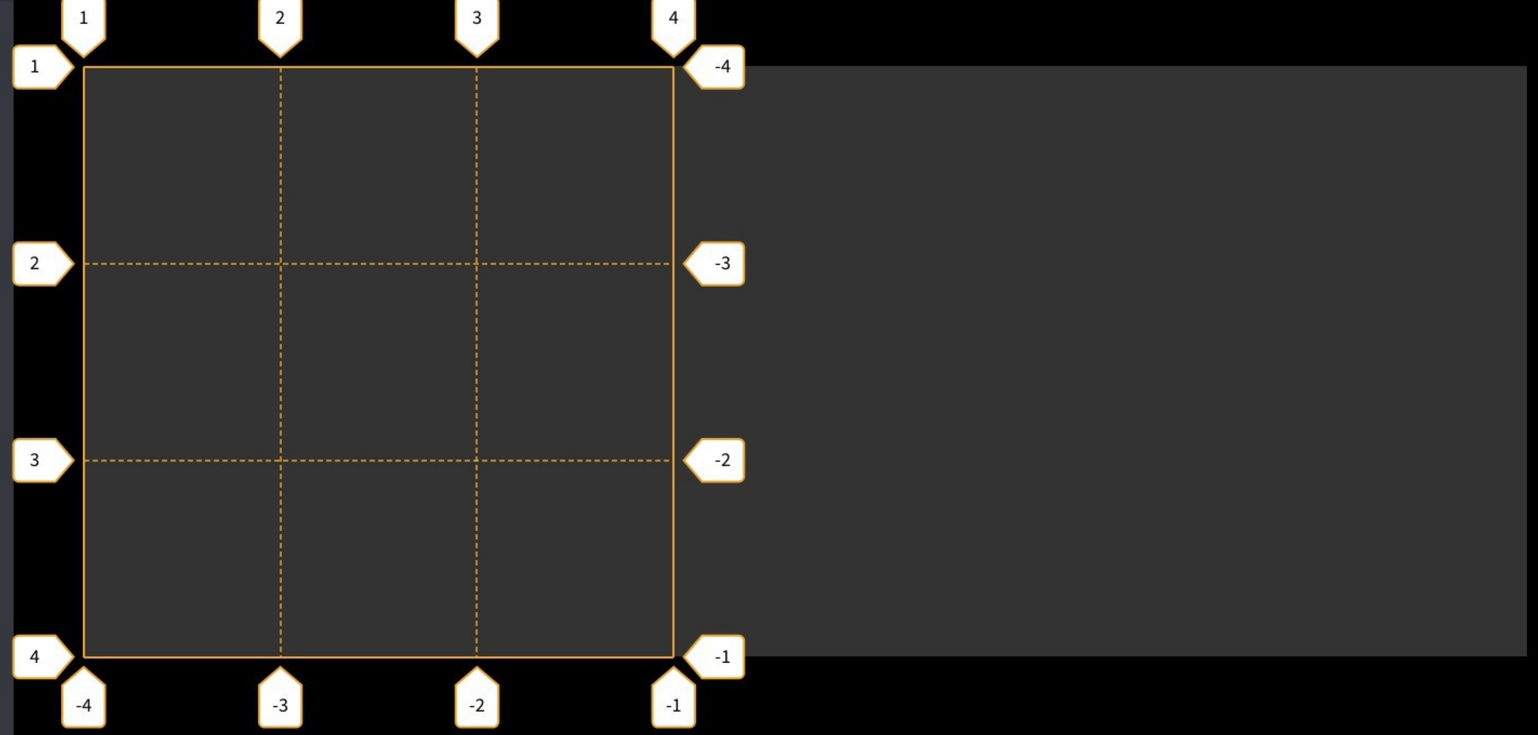
HTML

```
<div class="grid-container">

</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



How many areas are in this grid?

HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
```

CSS (SCSS)

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  > * {
    opacity: 0;
    animation: {
      name: flash;
      duration: 1s;
    }
  }
```

JS

HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
```

CSS (SCSS)

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  > * {
    opacity: 0;
    animation: {
      name: flash;
      duration: 1s;
    }
  }
```

JS

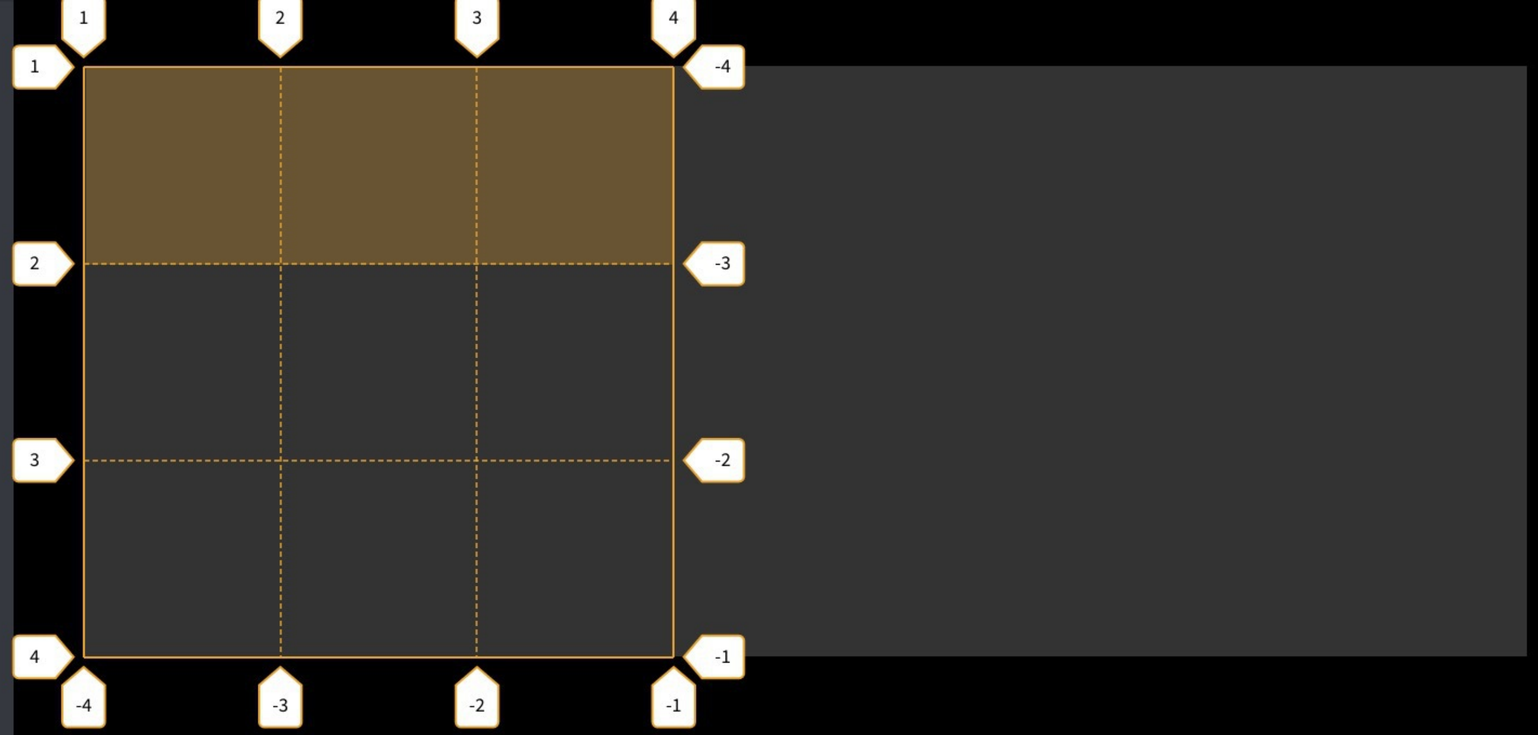
HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



Space between 2 adjacent grid lines defines *grid tracks* of columns or rows

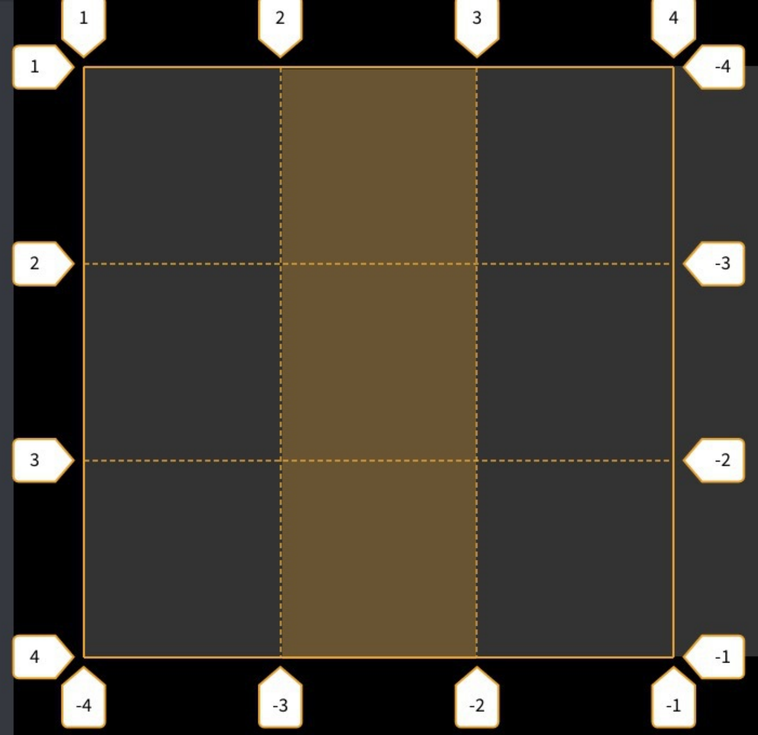
HTML

```
<div class="grid-container">

</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



A column track

How many total tracks are in this grid?

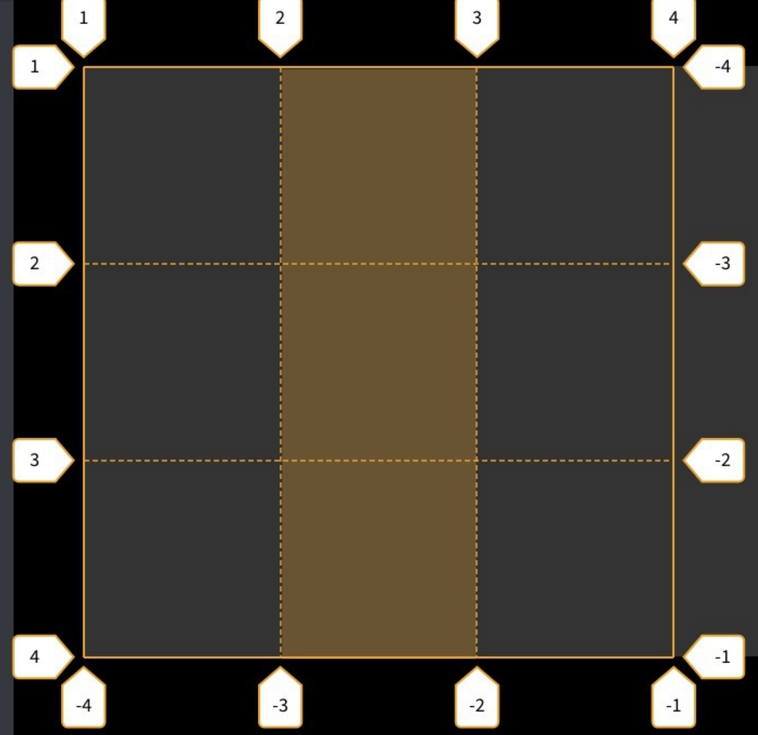
HTML

```
<div class="grid-container">

</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



A column track

How many total tracks are in this grid?

6!

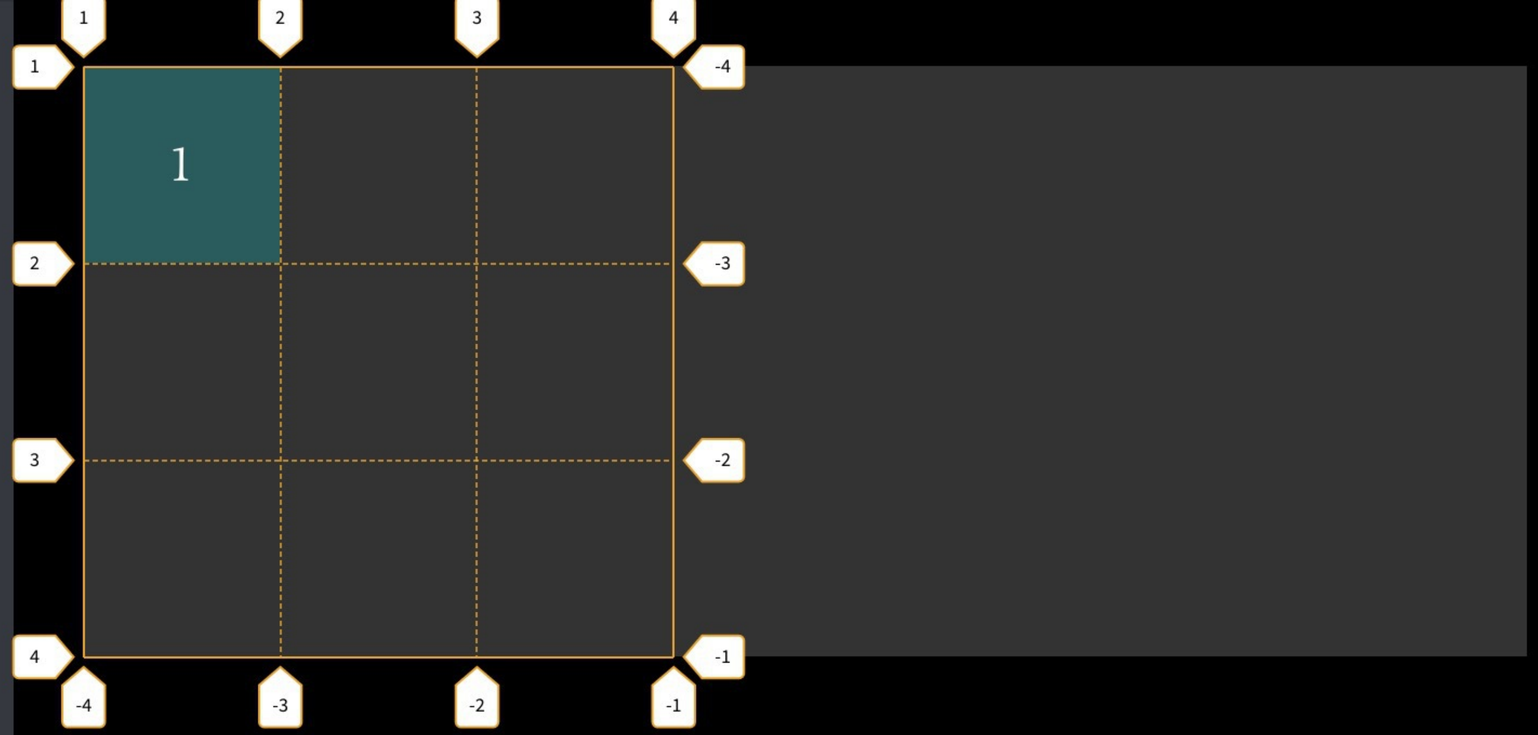
HTML

```html
<div class="grid-container">
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



Grid items are placed into grid areas based on grid lines — in this case, an area equal to 1 cell

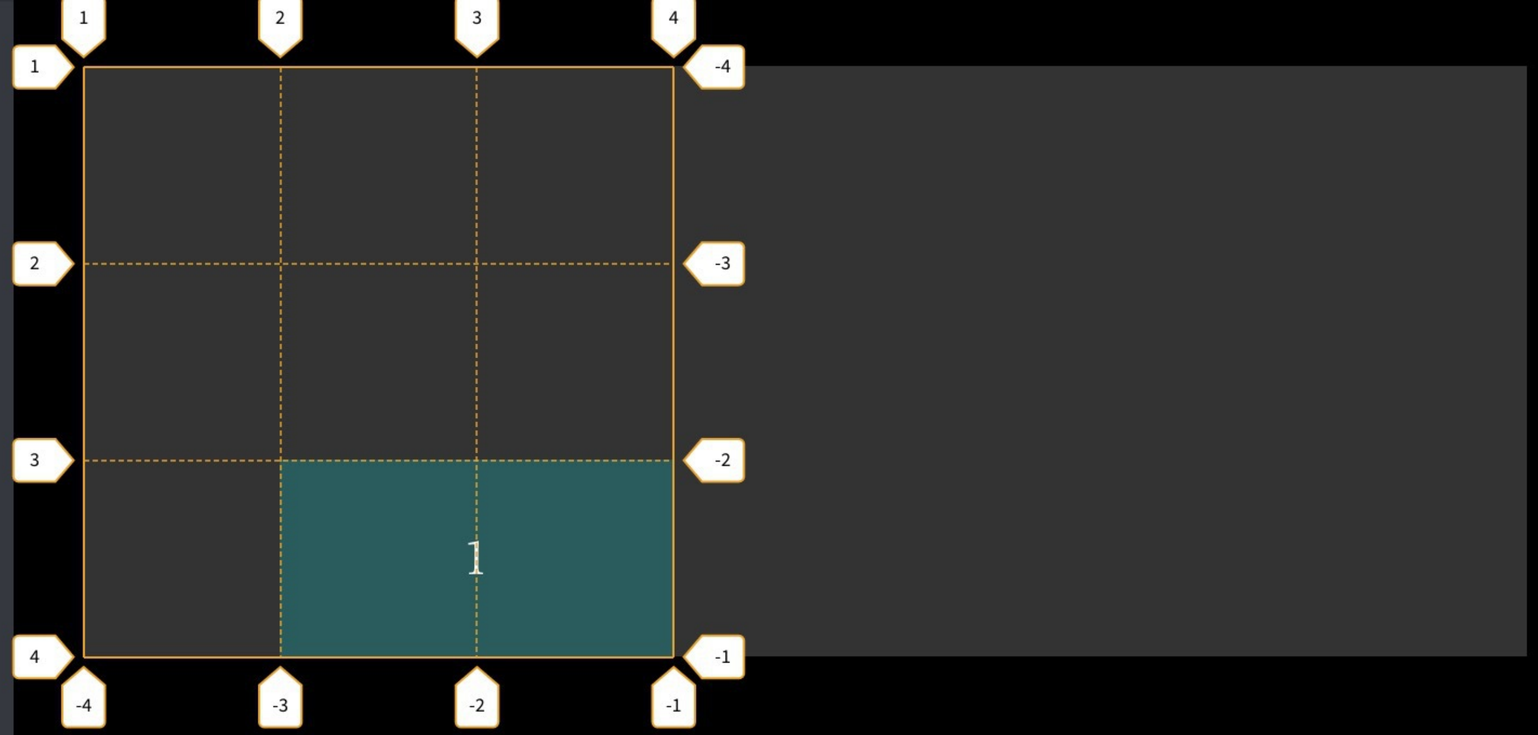
HTML

```html
<div class="grid-container">
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

div > div {
  grid-row: 3/4;
  grid-column: 2/4;
}
```

JS



Grid items are placed into areas that can span more than 1 cell

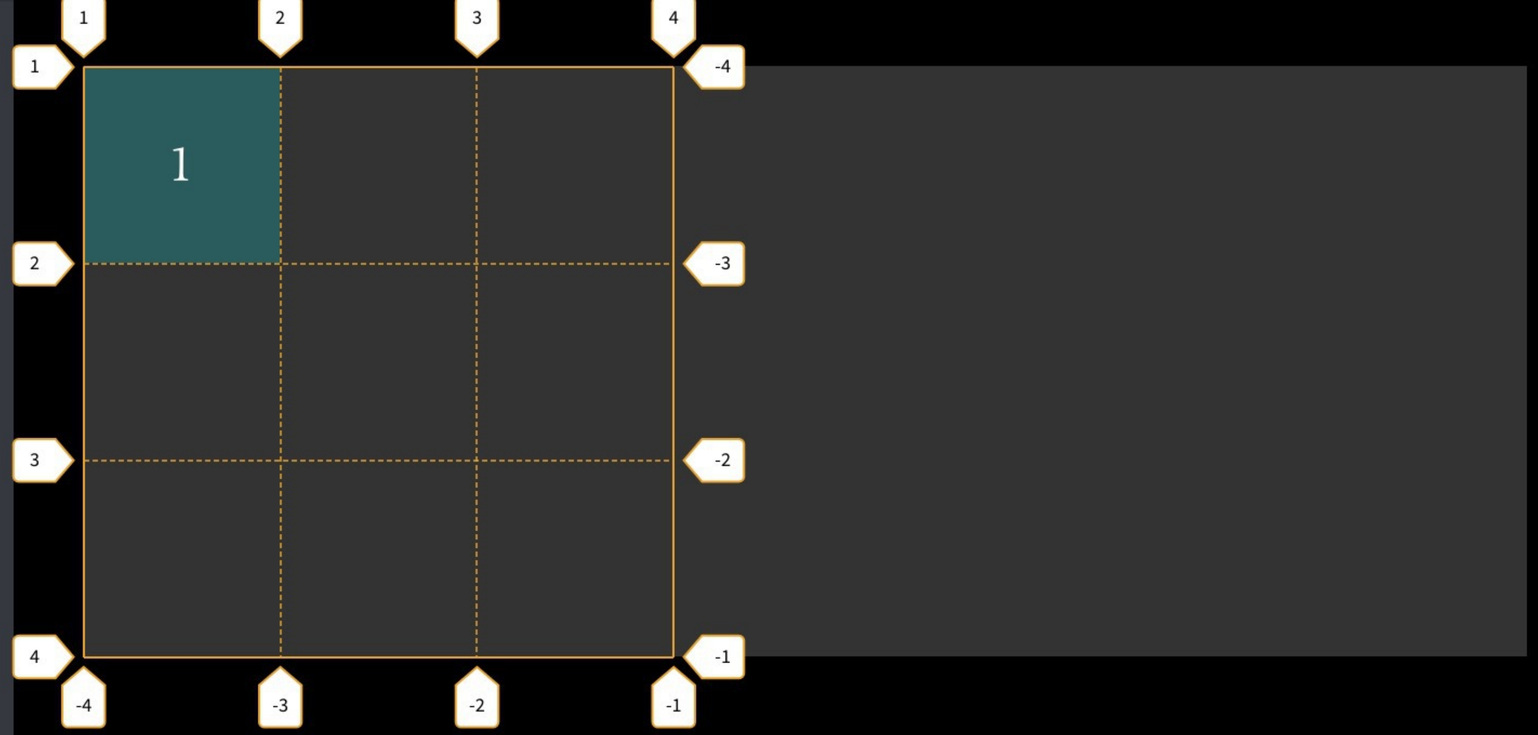
HTML

```html
<div class="grid-container">
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



All direct children of grid container are grid items*

* With a few exceptions

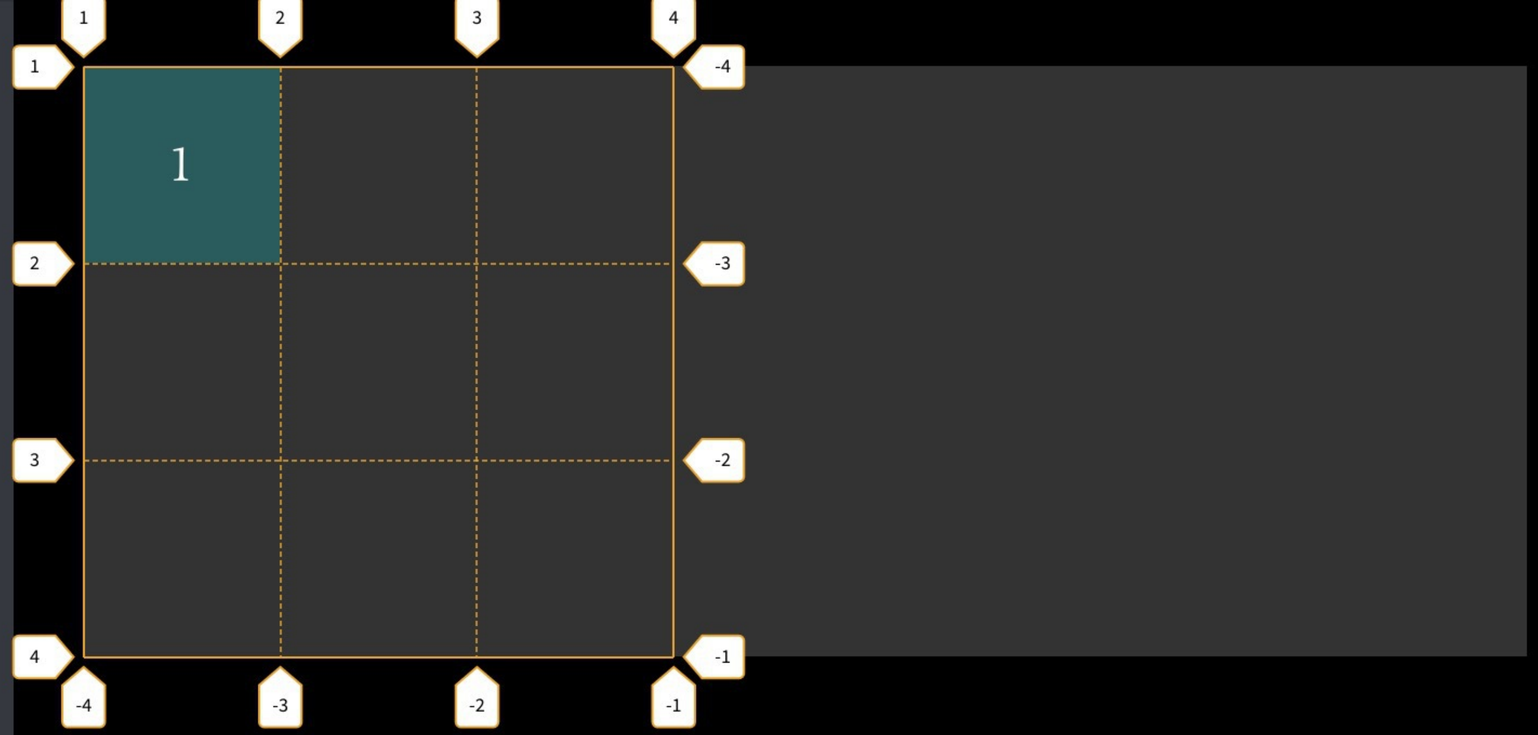
HTML

```
<div class="grid-container">
  <div></div>
</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



This grid has 9 cells but only 1 item

Cells are *not* part of the DOM so you *cannot* select them with CSS

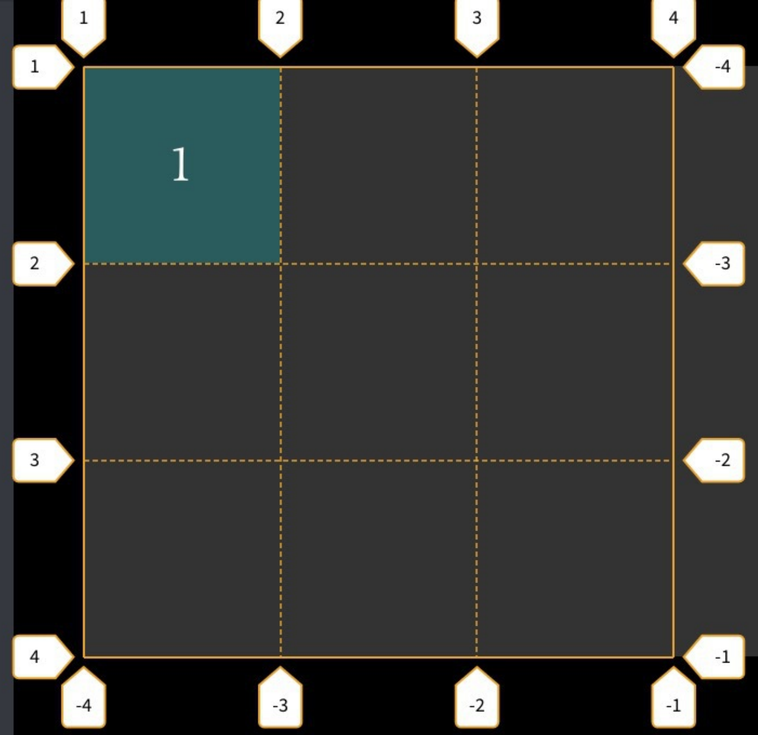
HTML

```
<div class="grid-container">
  <div></div>
</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



Grid item between:

» row lines 1 & 2, & column lines 1 & 2

» row lines -3 & -4, & column lines -3 & -4

» any others?

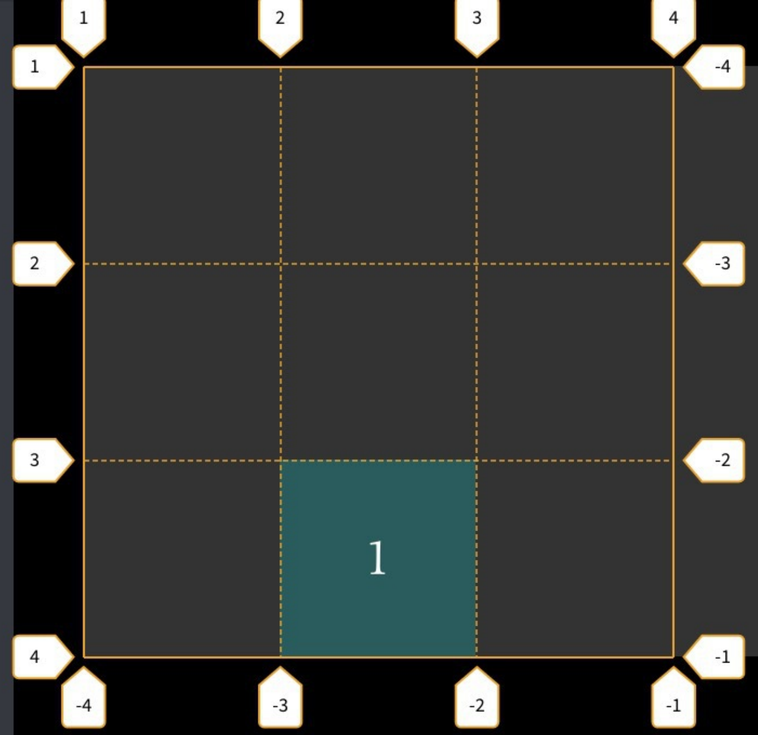
HTML

```
<div class="grid-container">
  <div></div>
</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

div > div {
  grid-row: 3/4;
  grid-column: 2/3;
}
```

JS



Grid item between:

» row lines 3 & 4, & column lines 2 & 3

» row lines -1 & -2, & column lines -2 & -3

» any others?

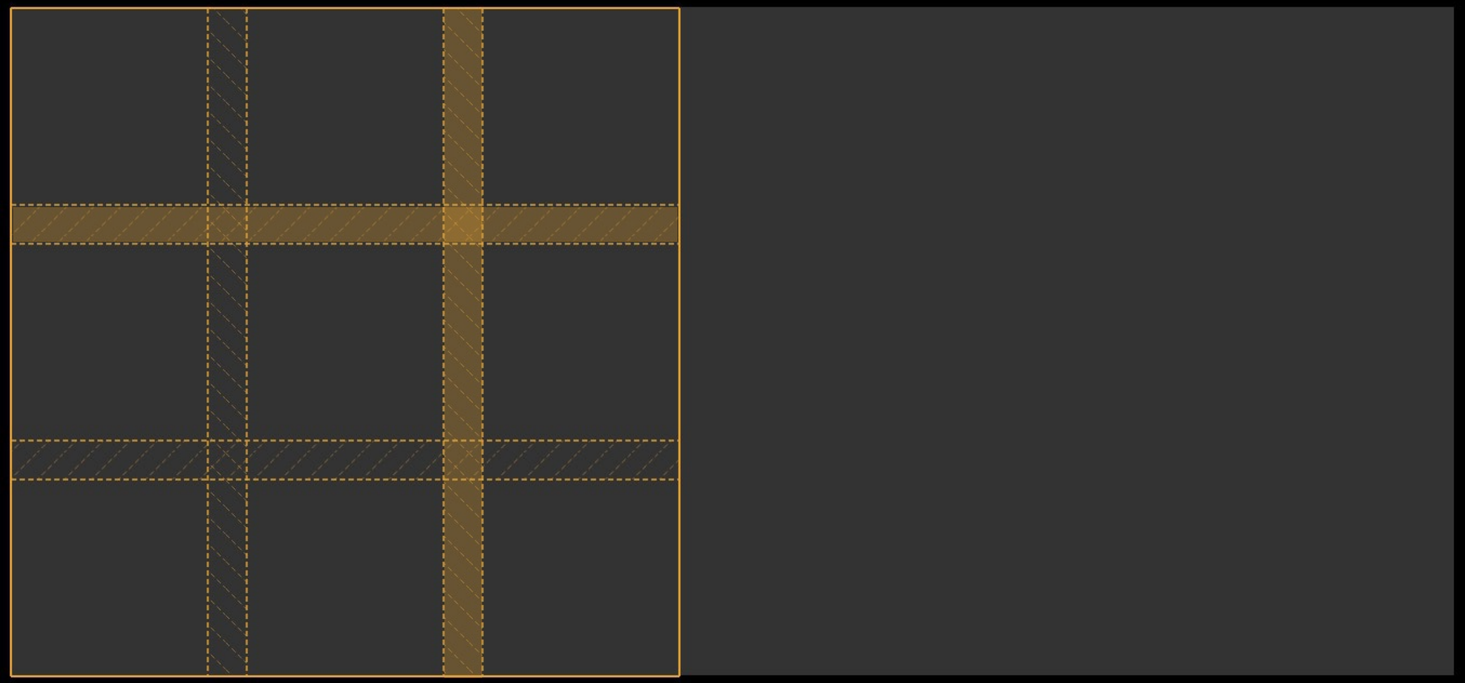
HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}

```

JS



Grid gutters are basically thick lines creating space between tracks

Your First Grid

```
display: grid

grid-template-columns
grid-template-rows

grid-gap/gap

grid-row-start
grid-row-end
grid-column-start
grid-column-end
```

HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
}
```

JS

Create a grid layout context with `display: grid`

Nothing to see because we triggered the grid layout but haven't yet built a grid

HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS

Build the grid with tracks using `grid-template-columns` & `grid-template-rows`

Now we see the grid container because there's a grid inside it

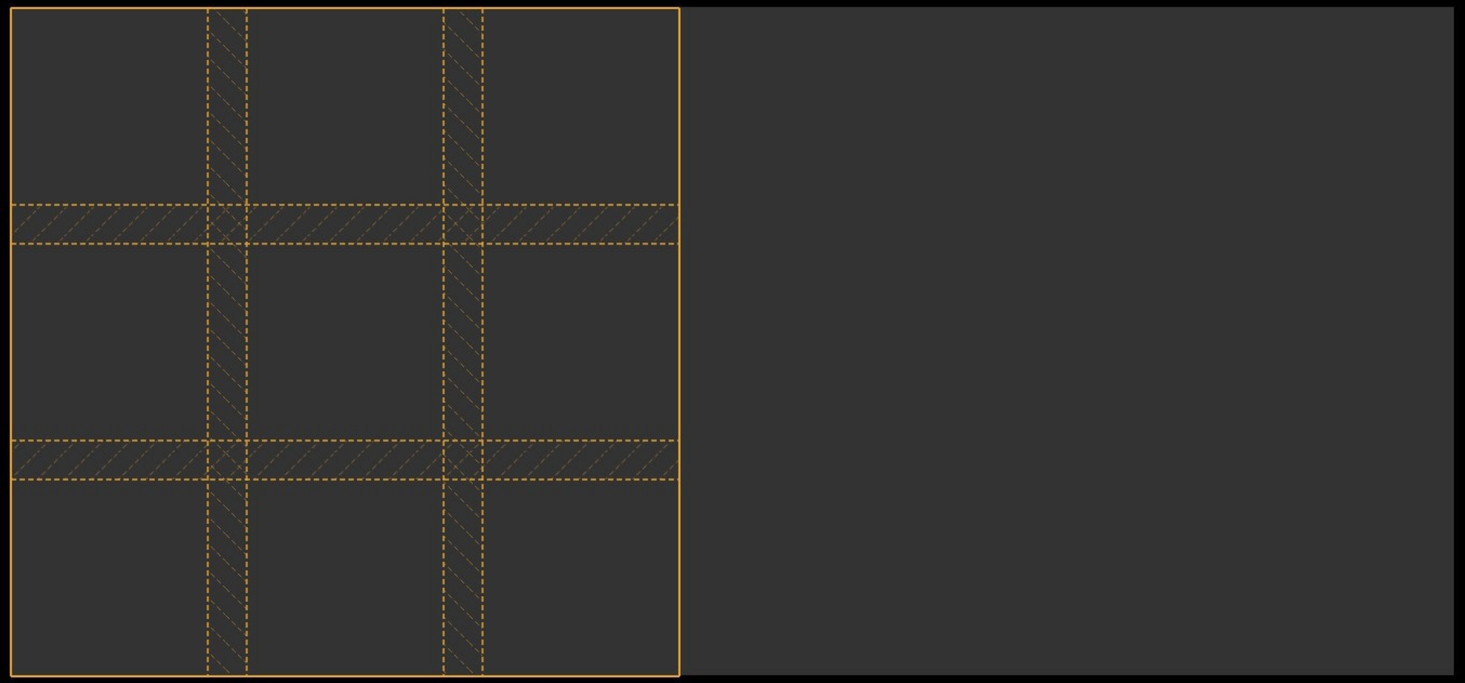
HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}
```

JS



Add space between tracks using `grid-gap`

Note the grid is now 340×340

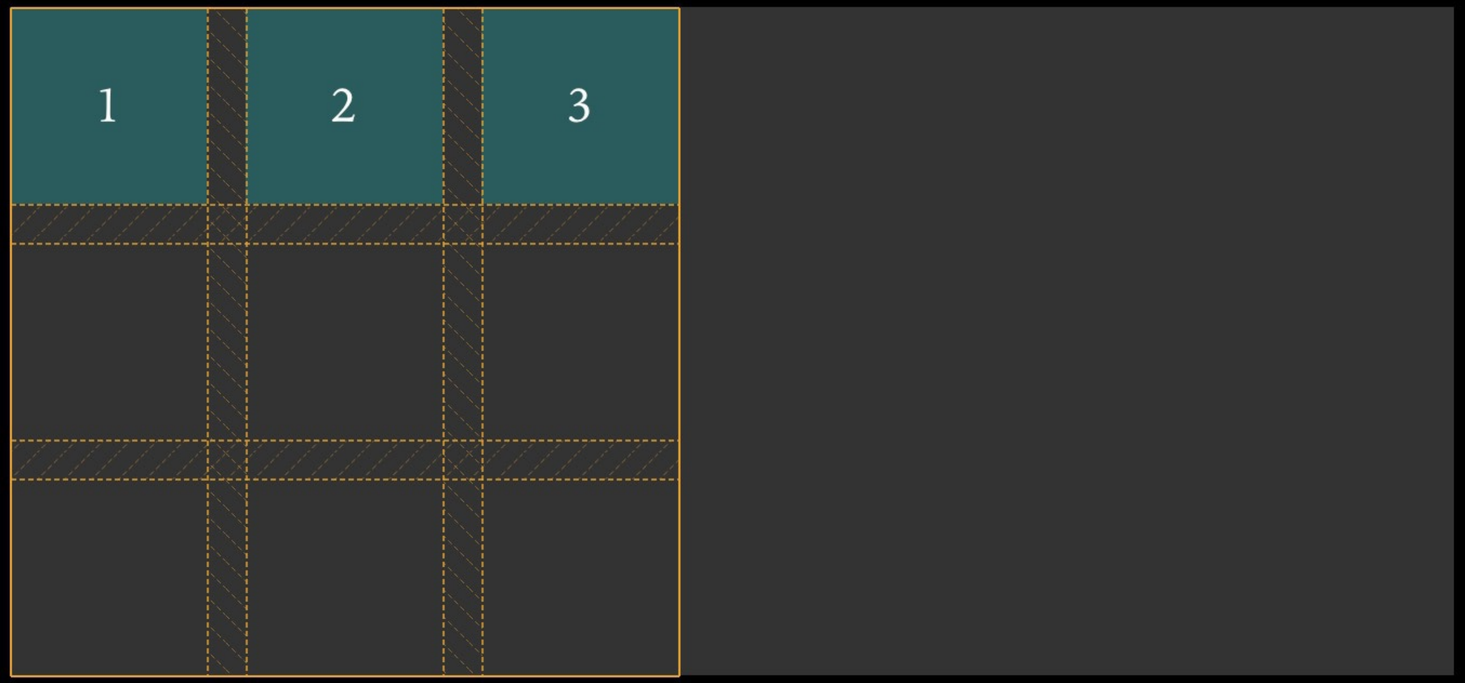
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}
```

JS



Add grid items, which are placed automatically by default

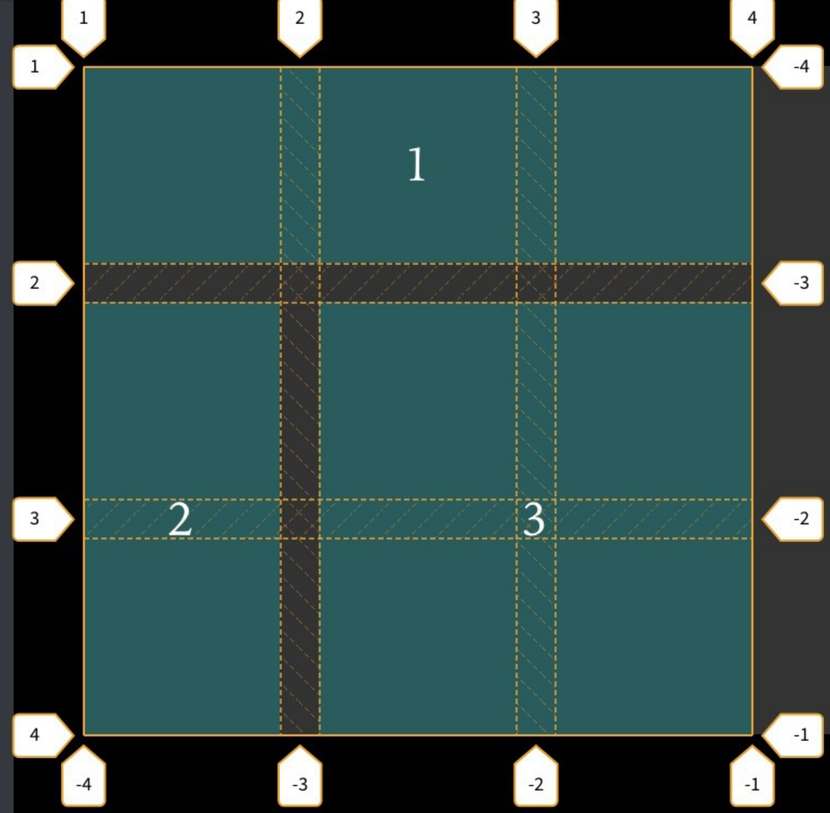
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}
.grid-container > :nth-child(1) {
  grid-column-start: 1;
  grid-column-end: 4;
}
.grid-container > :nth-child(2) {
  grid-row-start: 2;
  grid-row-end: 4;
}
.grid-container > :nth-child(3) {
  grid-column-start: 2;
  grid-column-end: 4;
  grid-row-start: 2;
  grid-row-end: 4;
}
```

JS

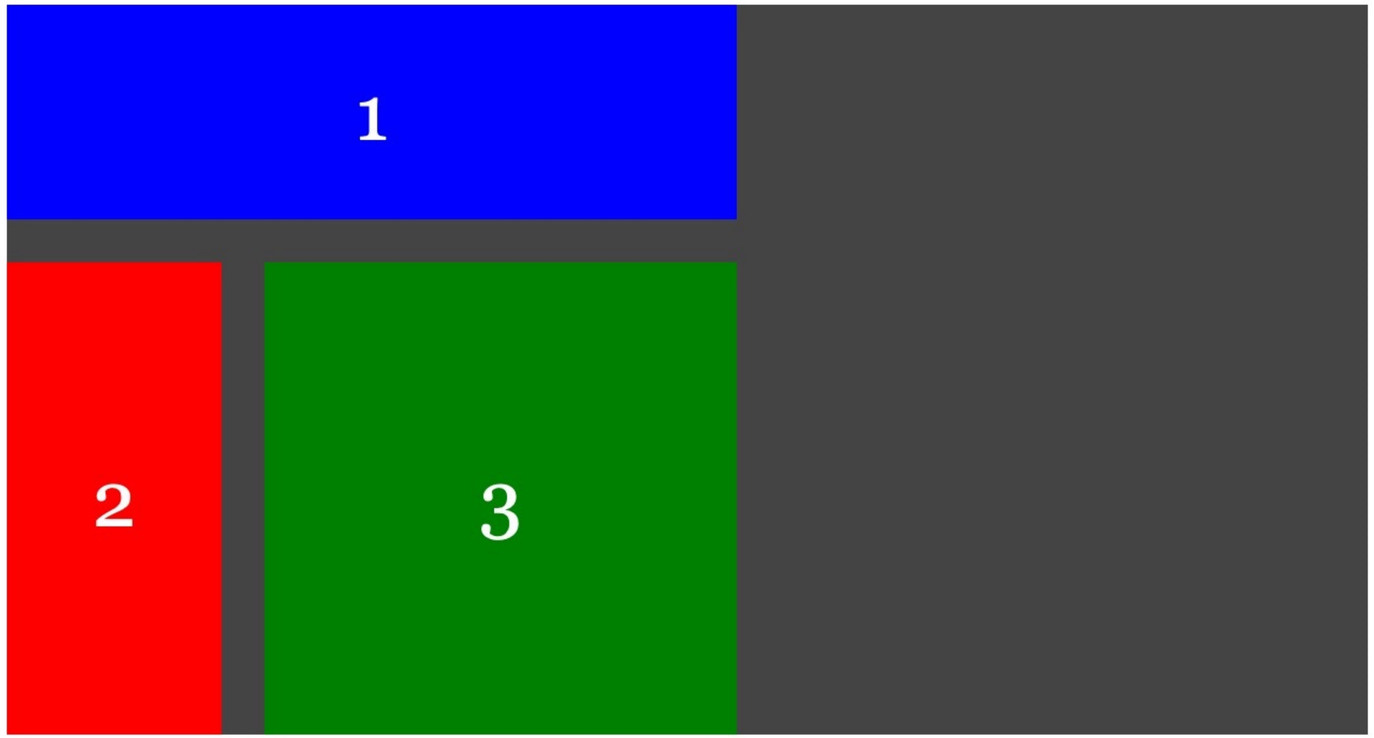


Place grid items using line numbers

Inspecting Grids

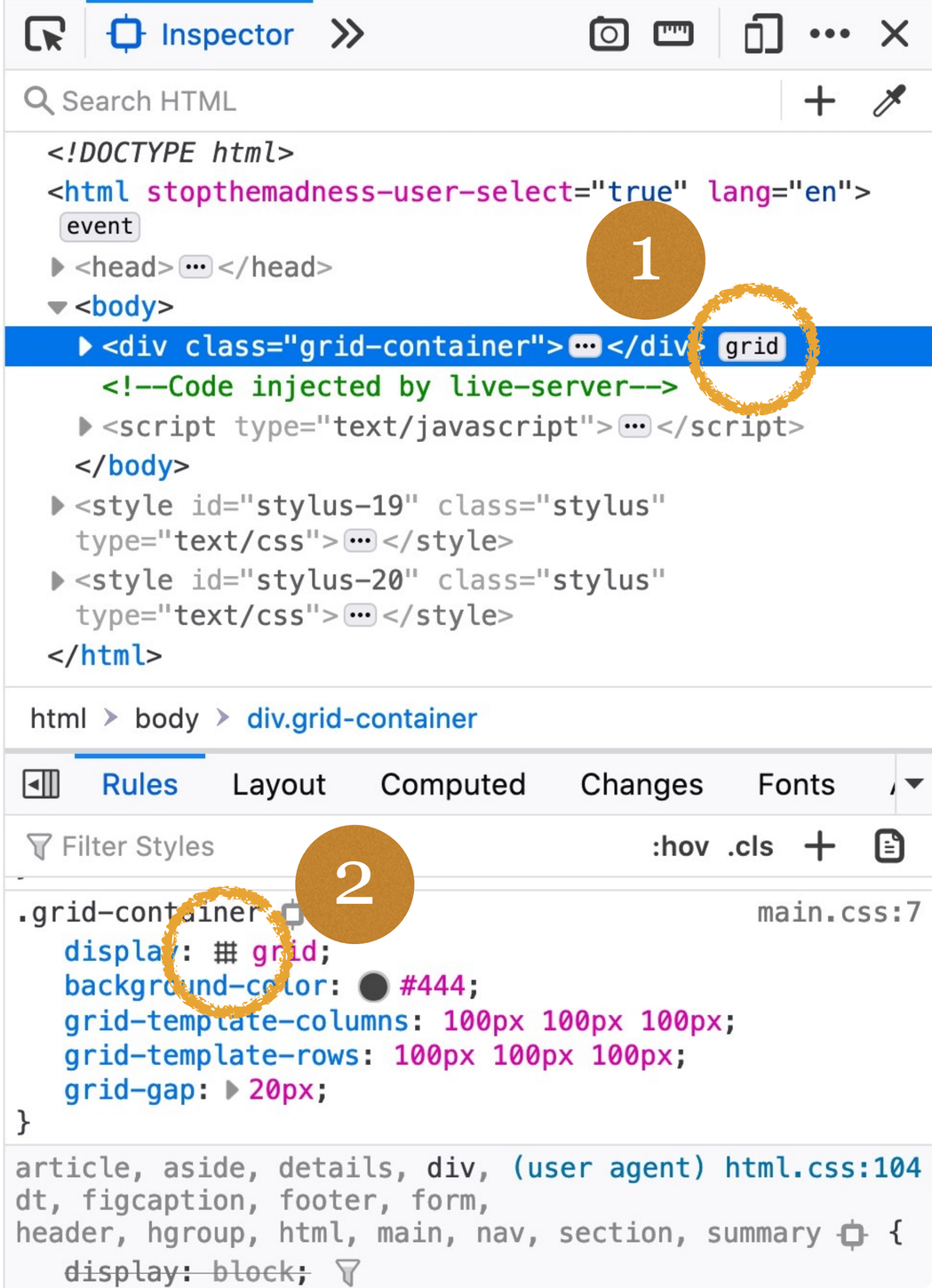
Firefox easily has the best grid inspector tools

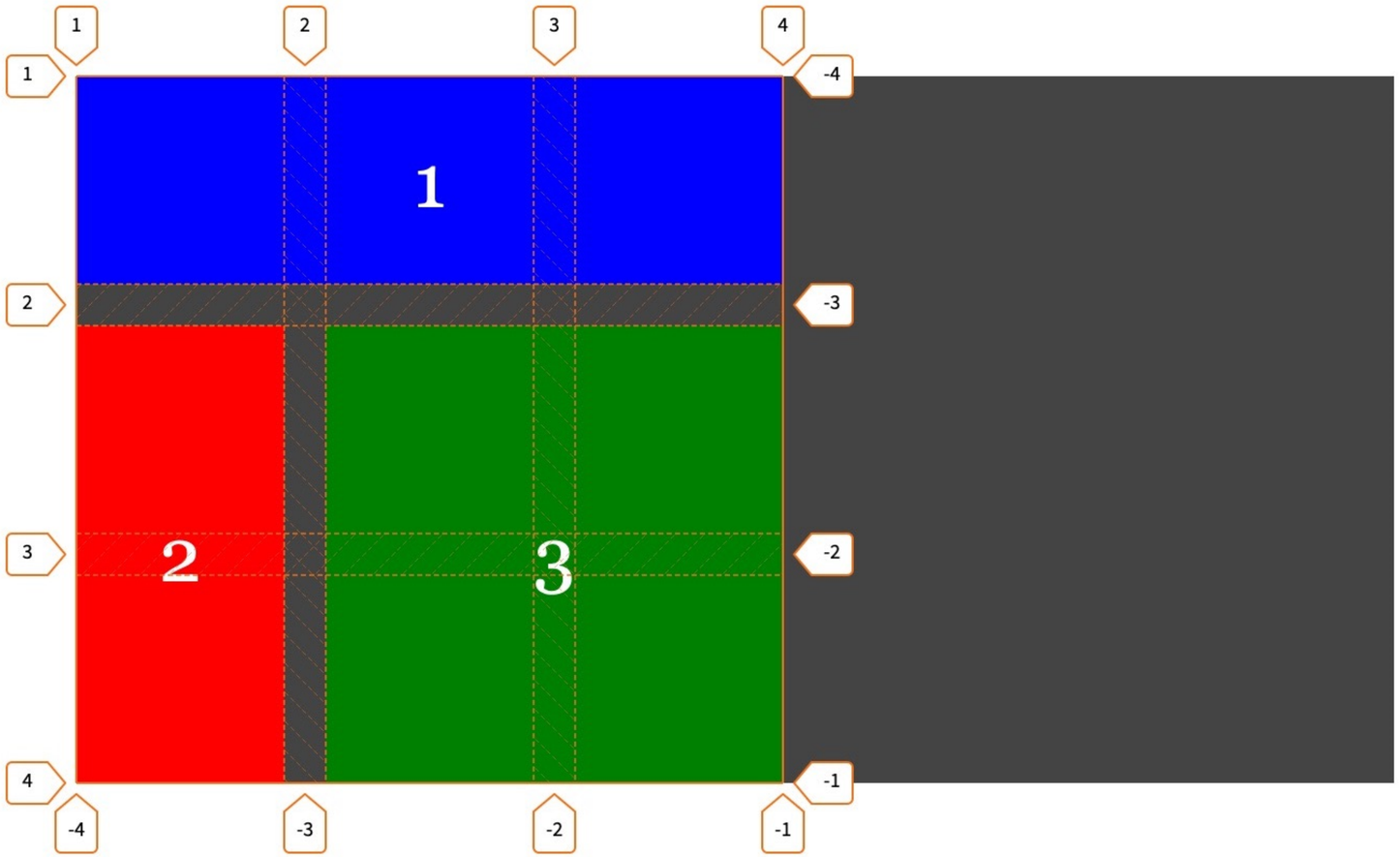
Right-click on a grid & select Inspect Element



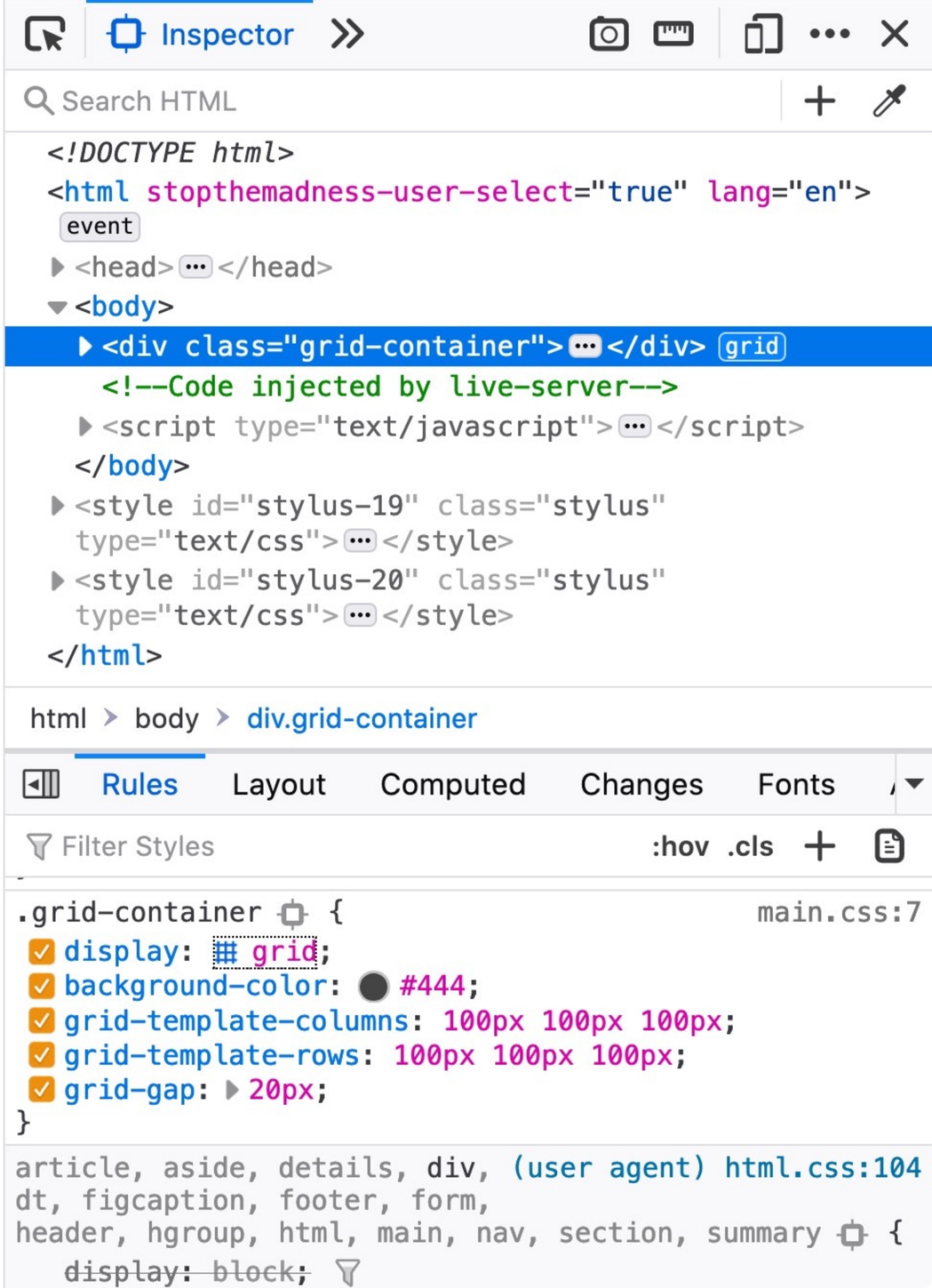
1. Indicates a grid container

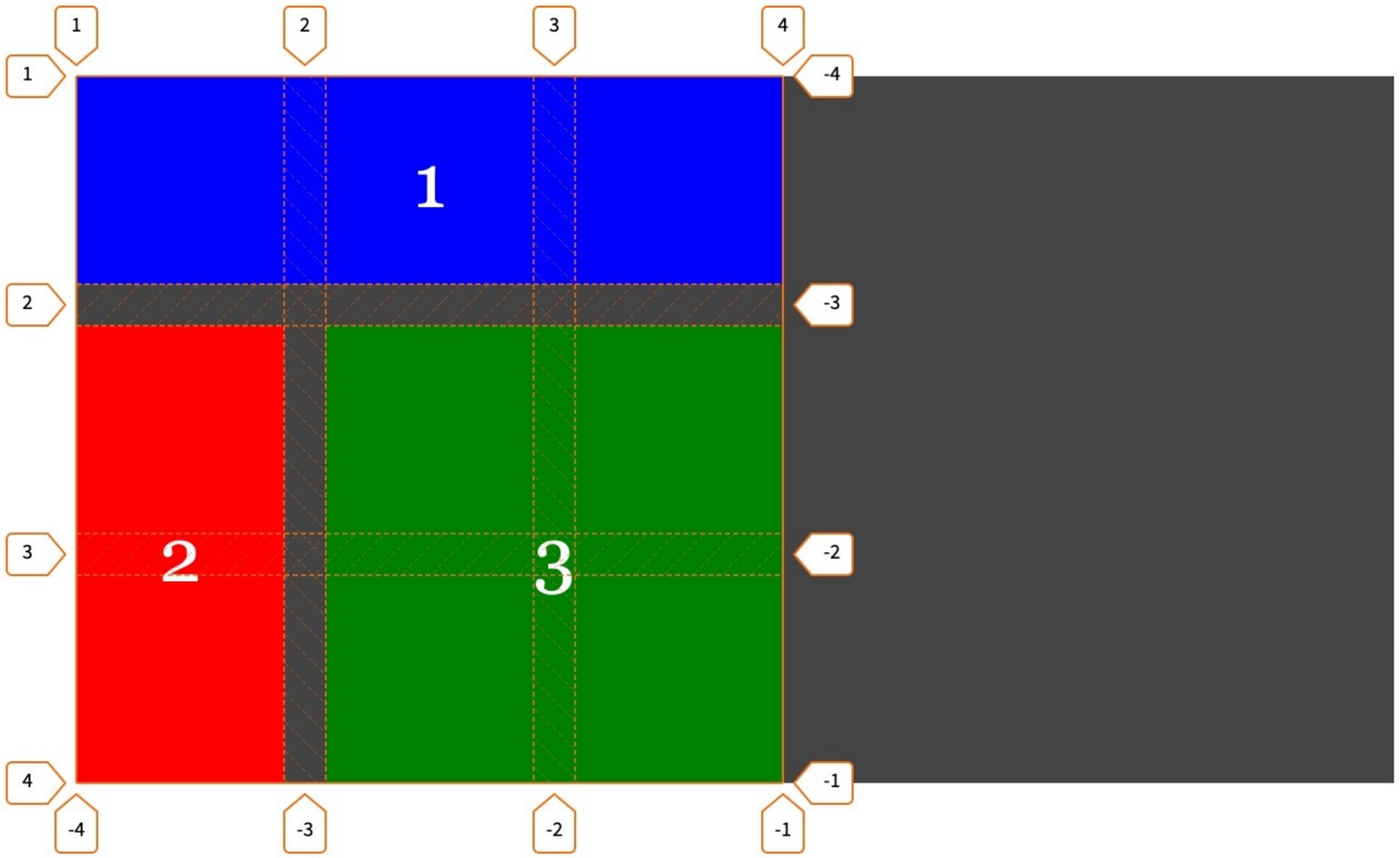
2. Click to reveal grid



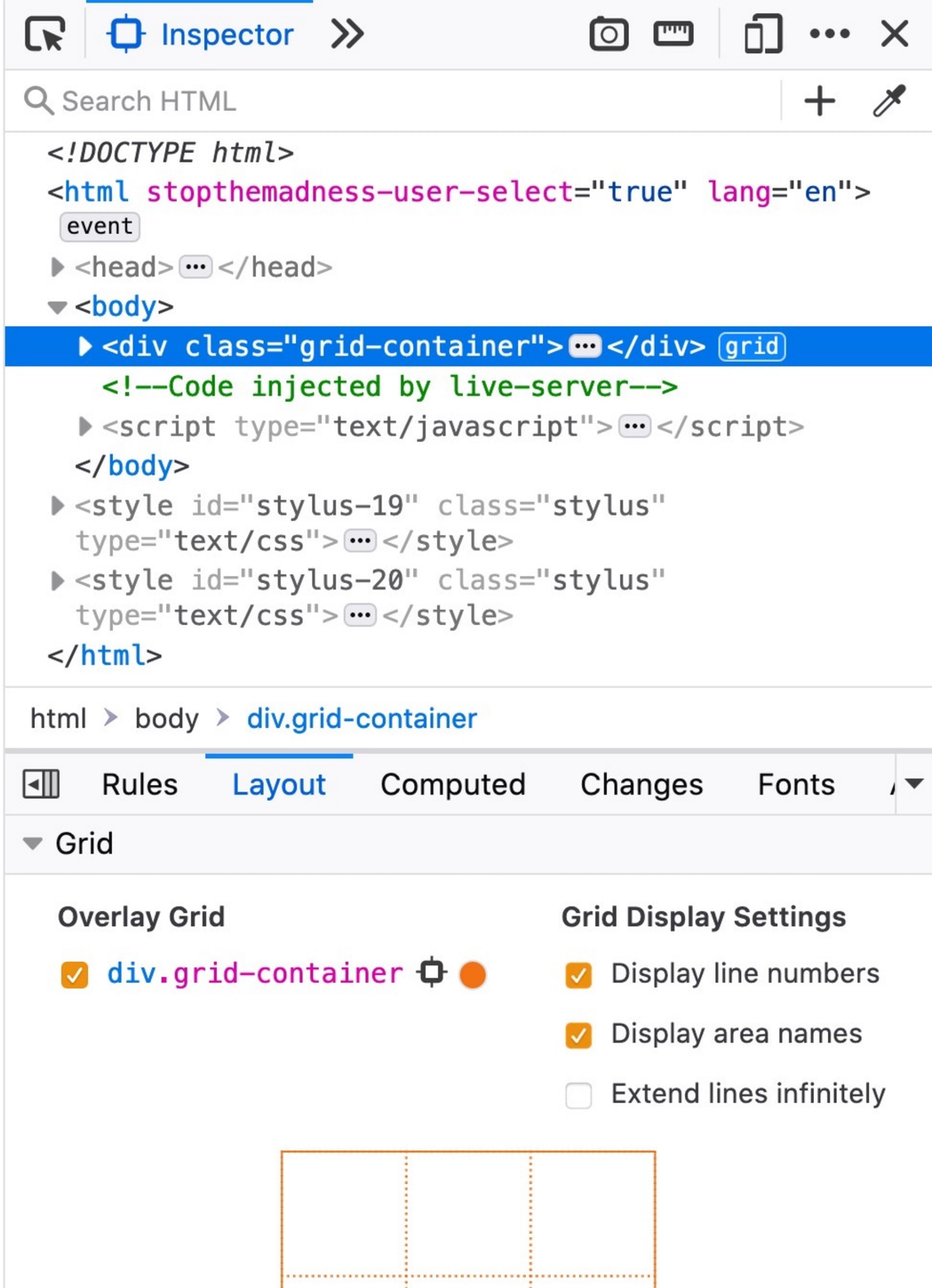


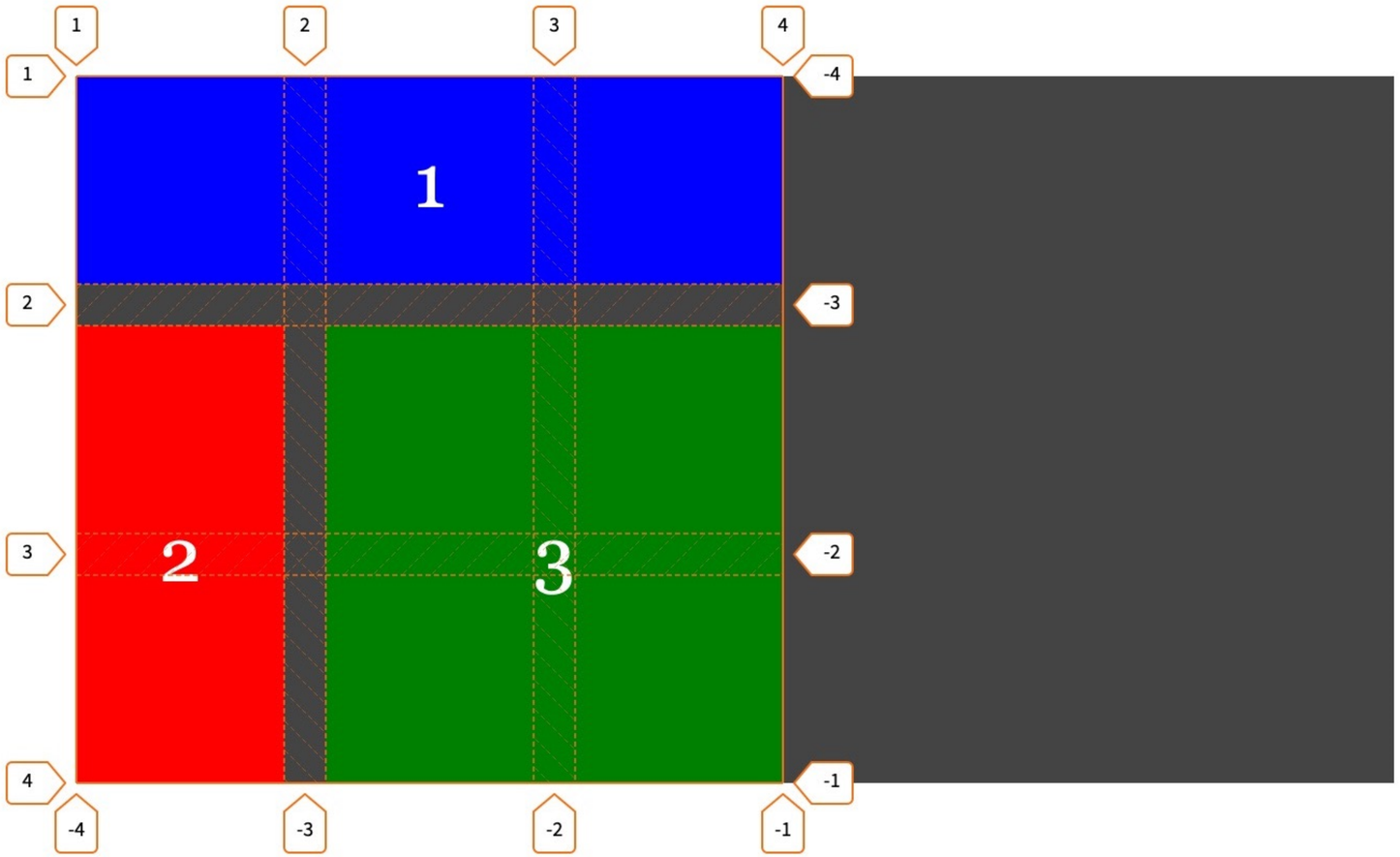
Line numbers appear — positive & negative — as do gutters





Under Layout > Grid, you can change what appears, including the colors of the lines





Inspector

Search HTML

```
<!DOCTYPE html>
<html stopthemadness-user-select="true" lang="en">
 event
 <head> ··· </head>
 <body>
  <div class="grid-container"> ··· </div> grid
   <!--Code injected by live-server-->
  <script type="text/javascript"> ··· </script>
 </body>
 <style id="stylus-1
  type="text/css"> ···
 <style id="stylus-2
  type="text/css"> ···
</html>
```

html › body › div.grid-co

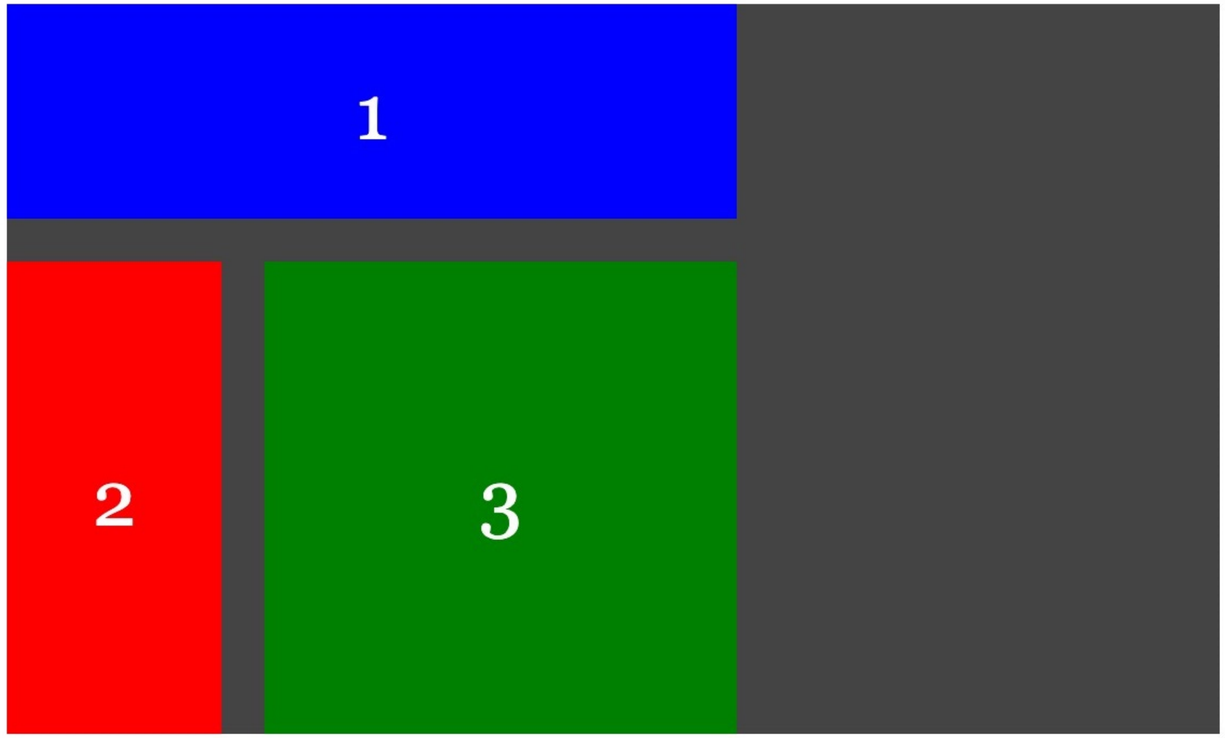
Rules Layout

Grid

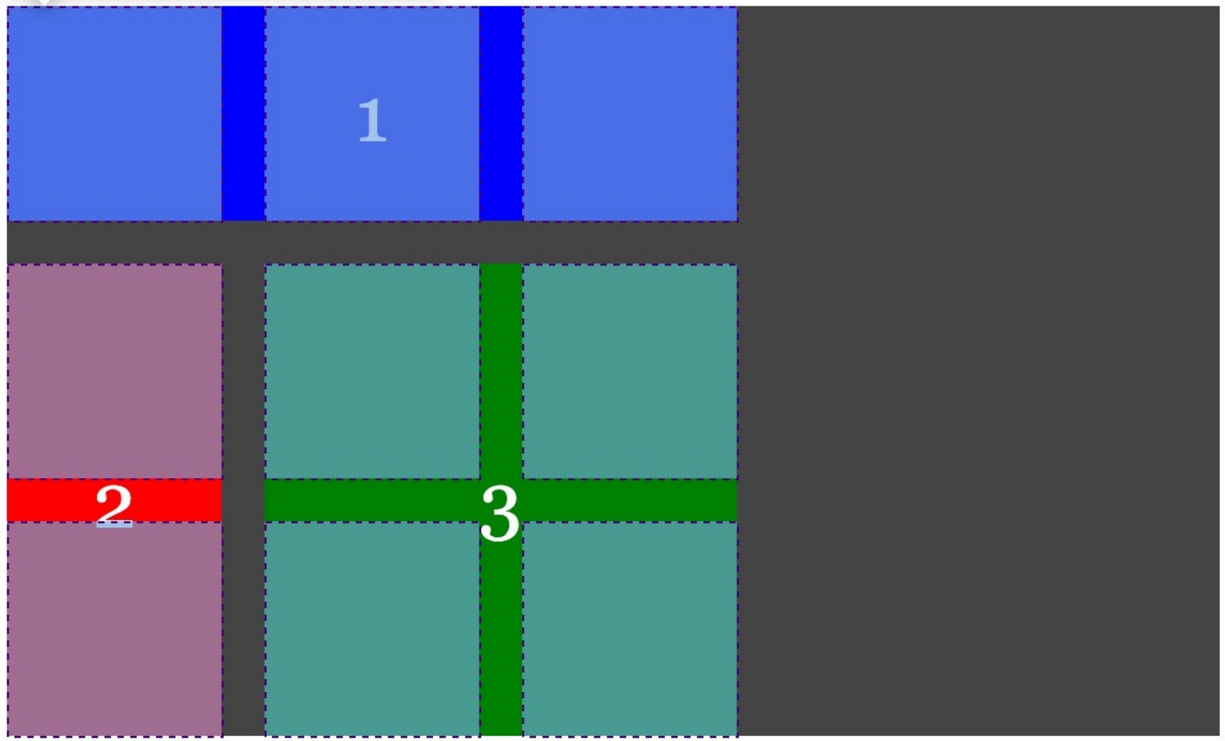
Overlay Grid

- [x] div.grid-container
- [x] Display line numbers
- [x] Display area names
- [ ] Extend lines infinitely

Chromium-based browsers are surprisingly not nearly as useful



Nothing much here...



Hover over the grid container

```
<!doctype html>
<html lang="en" stopthemadness-user-select="true">
▸ <head>…</head>
▾ <body>
  ▾ <div class="grid-container">
      <div>
            1
          </div>
      <div>
            2
          </div>
      <div>
            3
          </div>
    </div>
```

html style#stylus-19.stylus

Styles Event Listeners DOM Breakpoints Properties Accessibility

Filter :hov .cls +

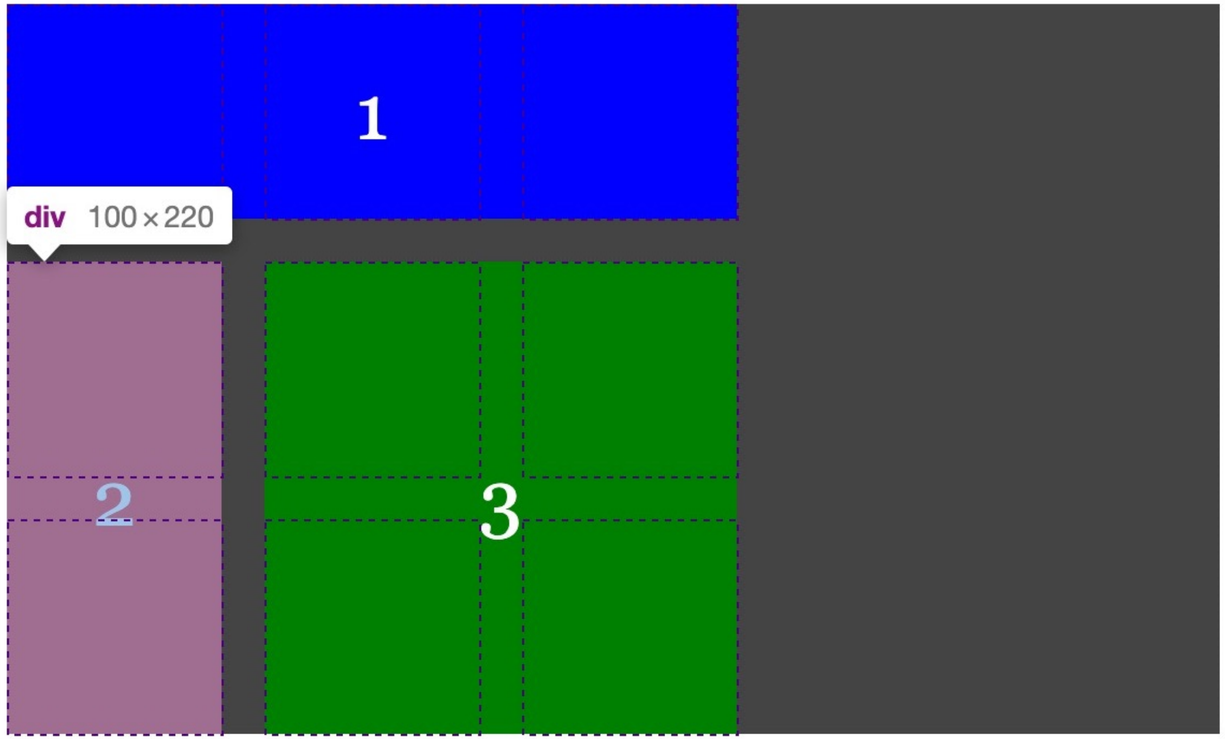
```
element.style {
}
html[Attributes Style] {
  -webkit-locale: "en";
}
html {                                  user agent stylesheet
  display: block;
}
```

margin -
border -
padding -
645 × 420

Filter Show all

display
block

height
420px



Elements Console Sources Network »

```
<!doctype html>
<html lang="en" stopthemadness-user-select="true">
▸ <head>…</head>
▾ <body>
  ▾ <div class="grid-container">
      <div>
          1
        </div>
      <div>
          2
        </div>
      <div>
          3
        </div>
    </div>
```

Hover over the grid item

html style#stylus-19.stylus

Styles Event Listeners DOM Breakpoints Properties Accessibility

Filter :hov .cls +

```
element.style {
}
html[Attributes Style] {
  -webkit-locale: "en";
}
html {                                  user agent stylesheet
  display: block;
}
```

margin -
border -
padding -
645 × 420

Filter Show all

display
block

height
420px

Safari’s Inspector does nothing special for grid, so it’s pretty useless

Triggering Grid Layout

`display: grid` (`<inside>`)
`display: block grid` (`<outside>` `<inside>`)

Creates a grid layout context inside the box:

» Grid box aligns vertically (because `block`)
» Creates the grid container
» Container can be bigger (or smaller) than the grid itself
» *Immediate* children become *grid items*

HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS

Note that there are no grid items yet!

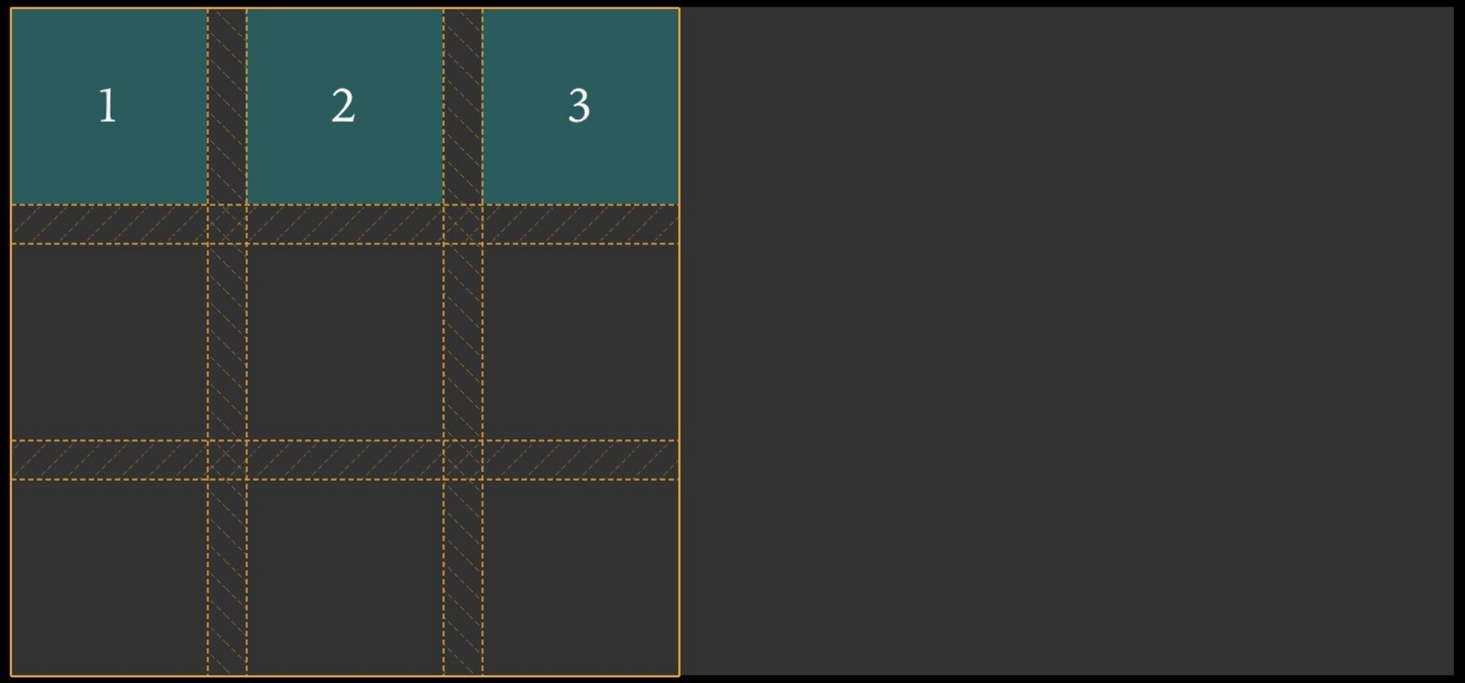
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}
```

JS



Each grid item creates a new layout context, so each grid item can itself be a flow, flexbox, or grid container

Track Basics

When you create tracks, you define

» whether they are columns or rows
» how many tracks there are
» the size of the tracks
» optional names for the lines adjacent to the tracks

```
grid-template-columns

grid-template-rows
```

`grid-template-columns` & `grid-template-rows` are foundational for creating the grid tracks, which in turn define the grid

`grid-template-columns`

Explicitly defines *size & number of grid columns, & line names* via a `<track list>`

Can mix & match any units: `px`, `em`, `%`, `fr` …

`<track list>` can be…

» a single size value for 1 column: `200px`
» multiple size values for >1 columns: `200px 1fr 300px`
» a mix of values & line names: `200px [hpl] 300px`

`grid-template-rows`

Explicitly defines *size & number of grid rows* via a `<track list>`

Same rules as `grid-template-columns`

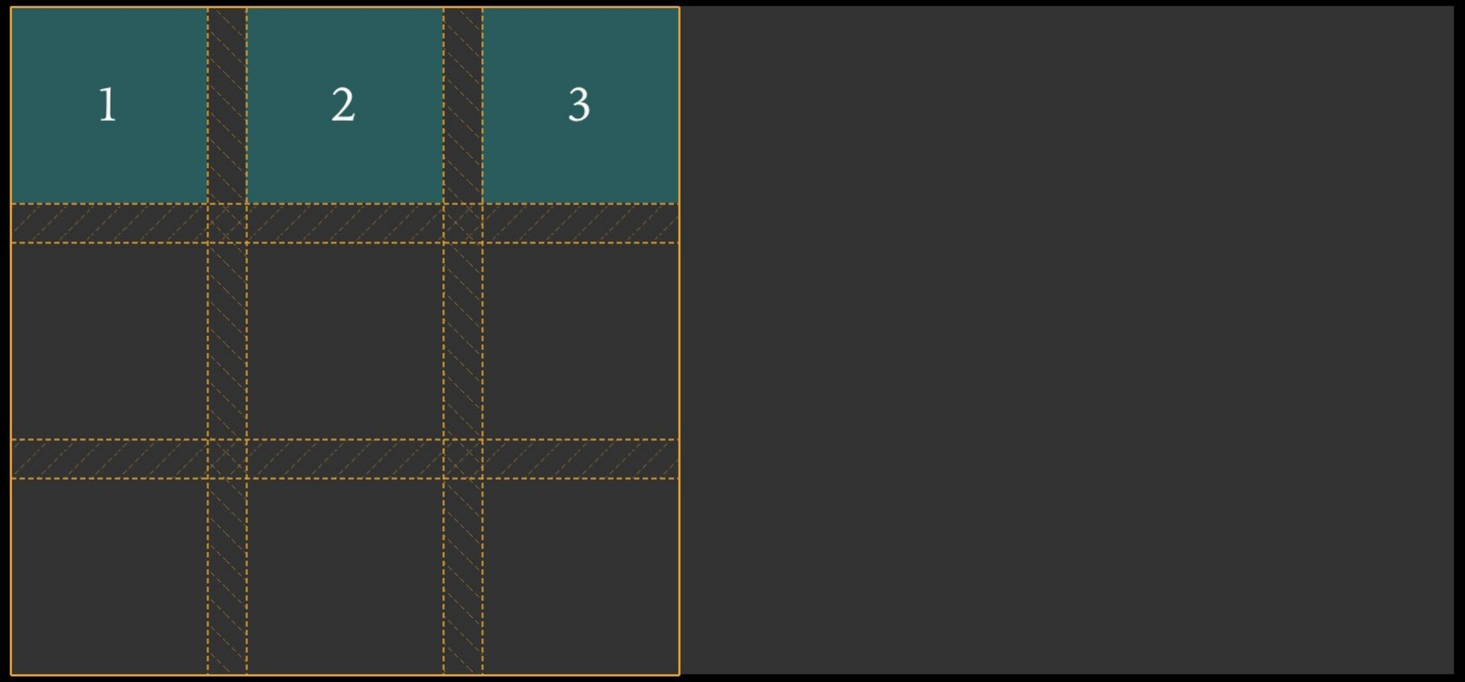
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}
```

JS



We will mention `grid-template-columns` & `grid-template-rows` constantly throughout the rest of this presentation

					iOS		
`grid-template-columns`	–*	16	52	10.1	10.3	57	57
`grid-template-rows`	–*	16	52	10.1	10.3	57	57

* IE uses the older `grid-columns` & `grid-rows`, which autoprefixer should take care of for you

Placing Items

5 ways to place grid items

- Automatic
- Numbered lines
- Named lines
- Named areas
- Spans

By default grid items are placed automatically in the same order as your code, as you've seen*

You can position grid items manually, however

* Details & exceptions having to do with `grid-auto-flow` will be covered later

```
grid-row-start
grid-row-end

grid-row


grid-column-start
grid-column-end

grid-column
```

```
grid-area
```

Values:

- `<integer>`: positive or negative
- `<custom-ident>`: name you choose
- `span <integer> && <custom-ident>`: tracks to stretch across

Numbered Lines

`grid-row-start`
`grid-row-end`
`grid-column-start`
`grid-column-end`

Properties that determine where 1 edge of a grid item is placed

Values specify *start or end line*; e.g.,

```
grid-row-start: 3
```

`grid-row`
`grid-column`

Shorthand properties that combine `-start` & `-end` for a *given track direction (row or column)*

Values specify start *and* end lines; e.g.,

```
grid-row: grid-row-start / grid-row-end
```

```
grid-row: 3/5
```

`grid-area`

Shorthand property that combines `-start` & `-end` for *both track directions*

Value specifies 4 start & end lines; e.g.,

```
grid-area: grid-row-start / grid-column-
start / grid-row-end / grid-column-end
```

```
grid-area: 2/2/4/5
```

					iOS		
grid-row-start	–	16	52	10.1	10.3	57	57
grid-row-end	–	16	52	10.1	10.3	57	57
grid-column-start	–	16	52	10.1	10.3	57	57
grid-column-end	–	16	52	10.1	10.3	57	57
grid-row	–	16	52	10.1	10.3	57	57
grid-column	–	16	52	10.1	10.3	57	57

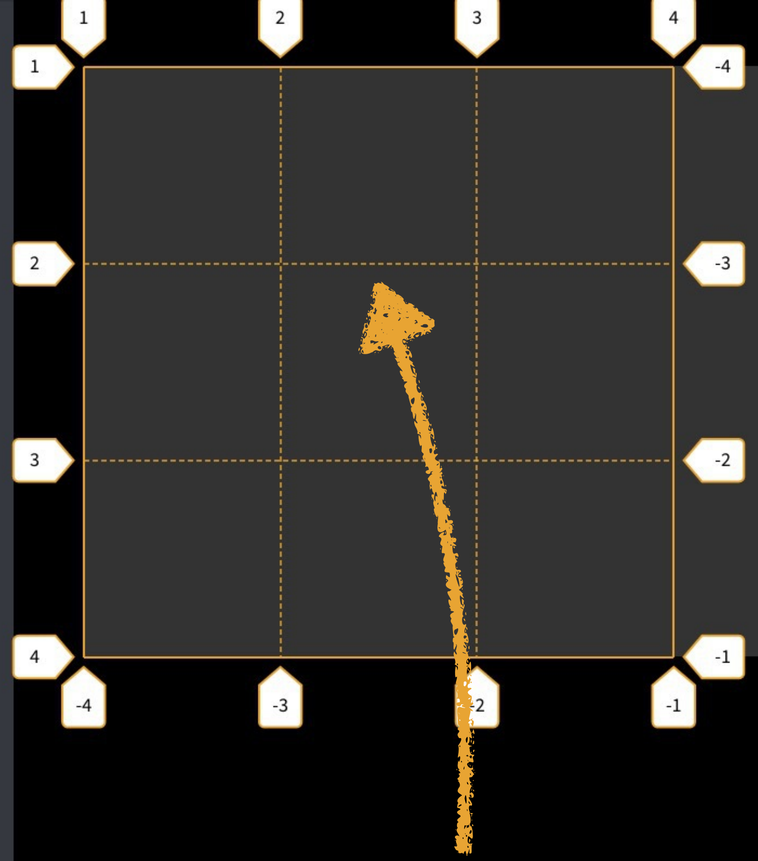
HTML

```html
<div class="grid-container">

</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



What number is this line?

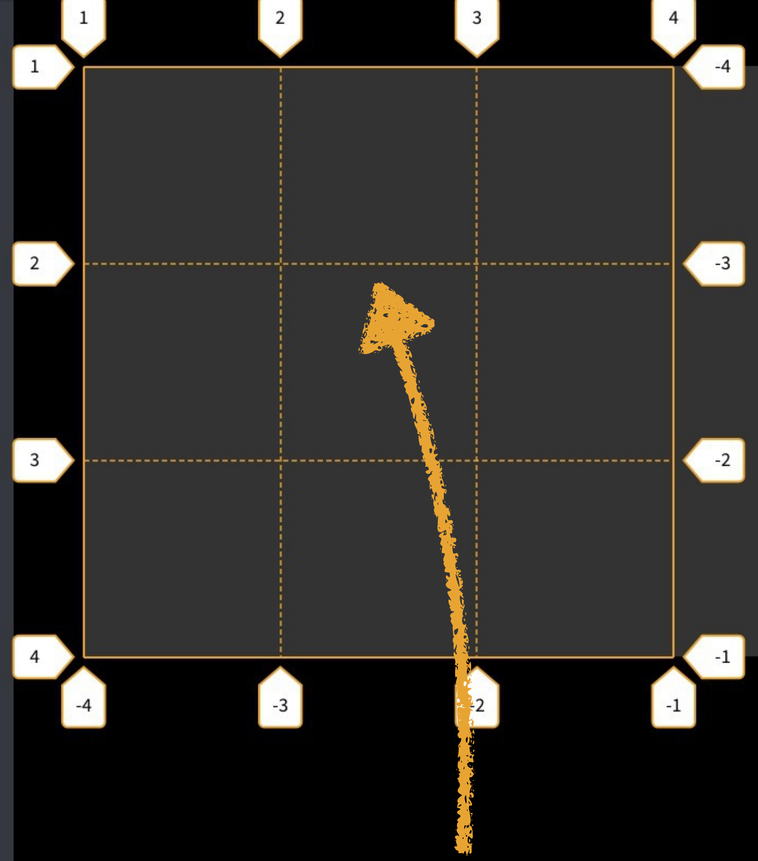
HTML

```
<div class="grid-container">

</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



What number is this line?

2 *and* −3

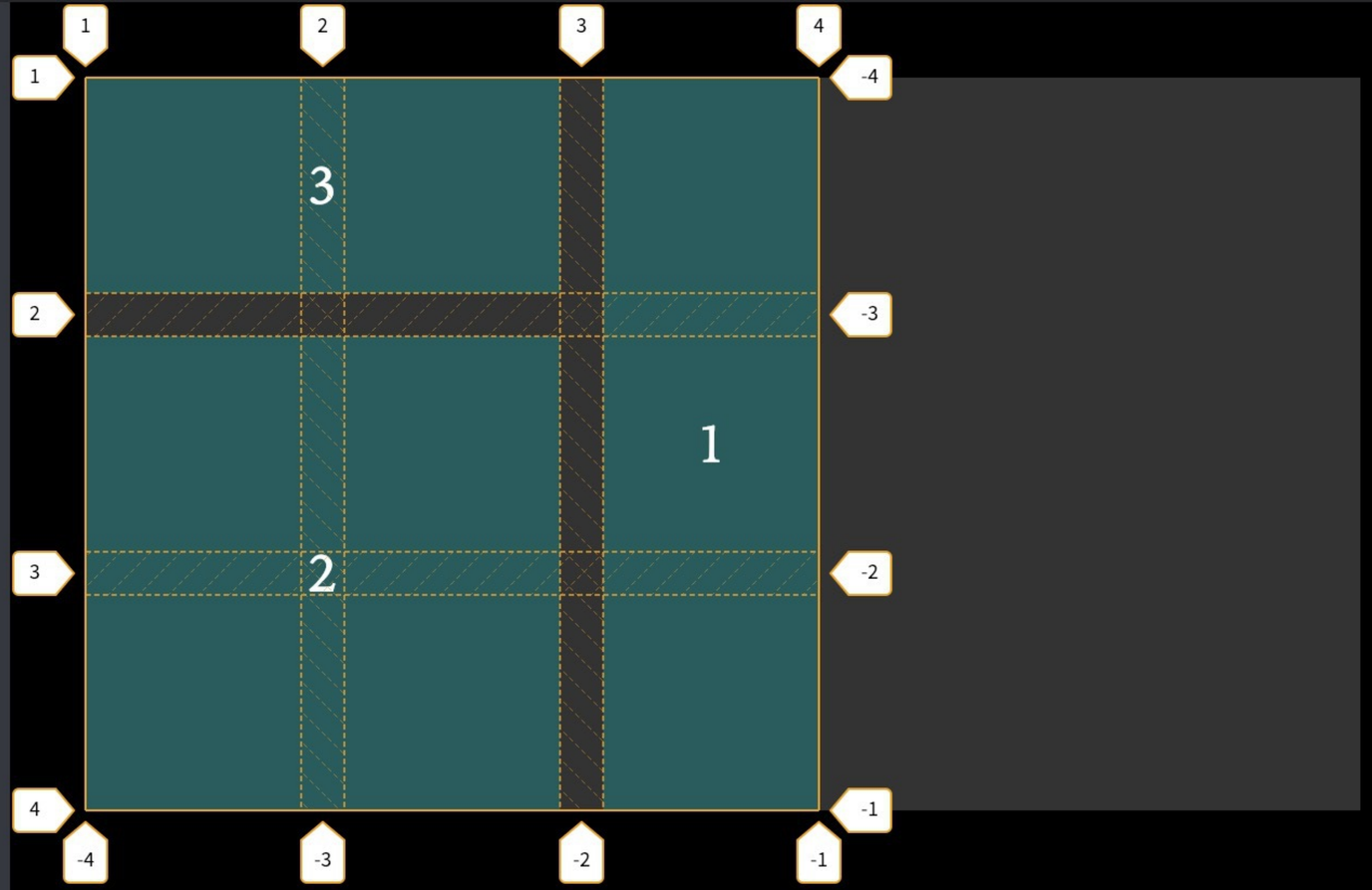
HTML

```
    <div></div>
    <div></div>
    <div></div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}
.grid-container > :nth-child(1) {
  grid-row-start: 1;
  grid-row-end: 4;
  grid-column-start: 3;
  grid-column-end: 4;
}
.grid-container > :nth-child(2) {
  grid-row: 2 / 4;
  grid-column: 1 / 3;
}
.grid-container > :nth-child(3) {
  grid-area: 1 / 1 / 2 / 3;
}
```

JS



Negative Line Numbers

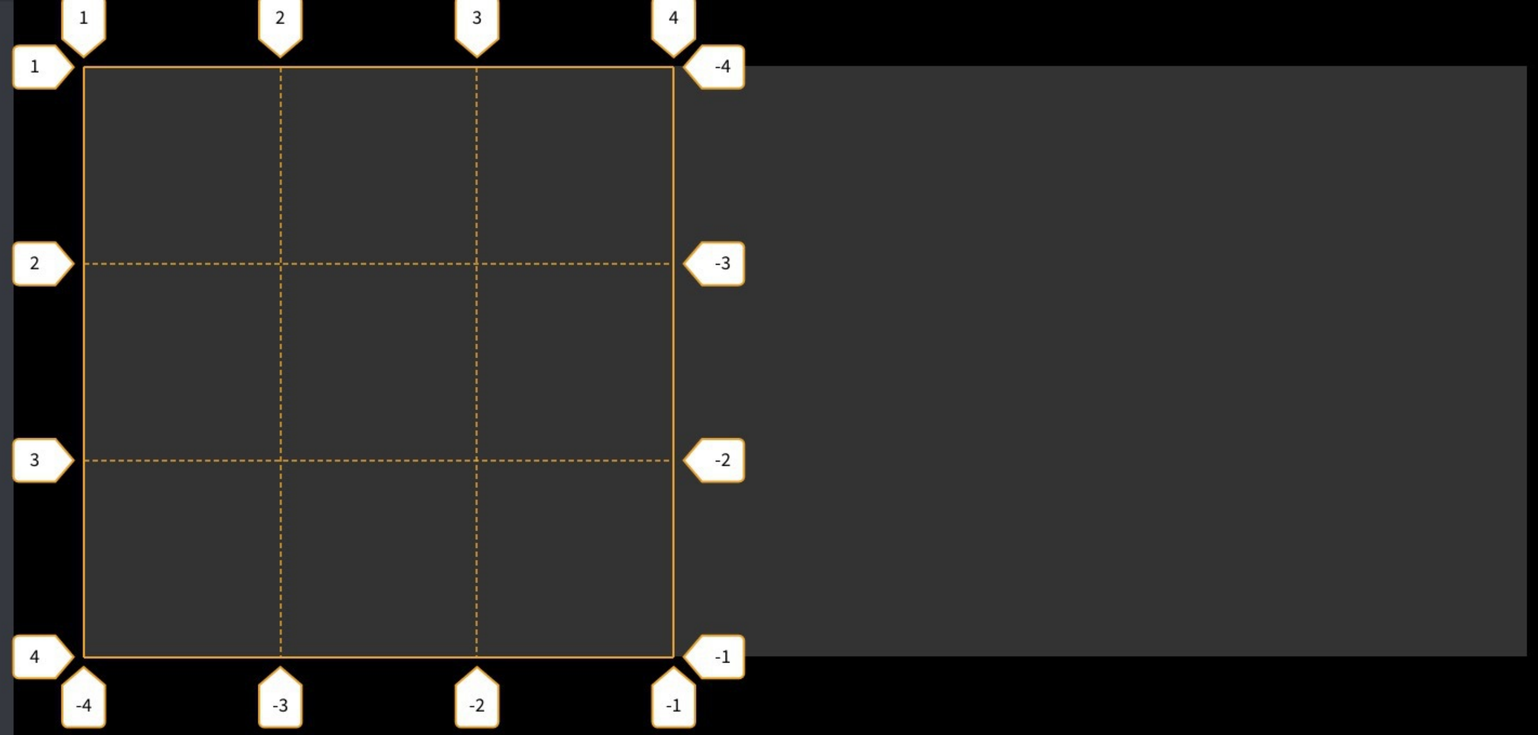
HTML

```
<div class="grid-container">

</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}

```

JS



Use negative numbers to count backwards from the ends

Why negative line numbers?

The number of tracks might change, & you don’t know the line number for the end

`1` always represents the start
`-1` *always* represents the end

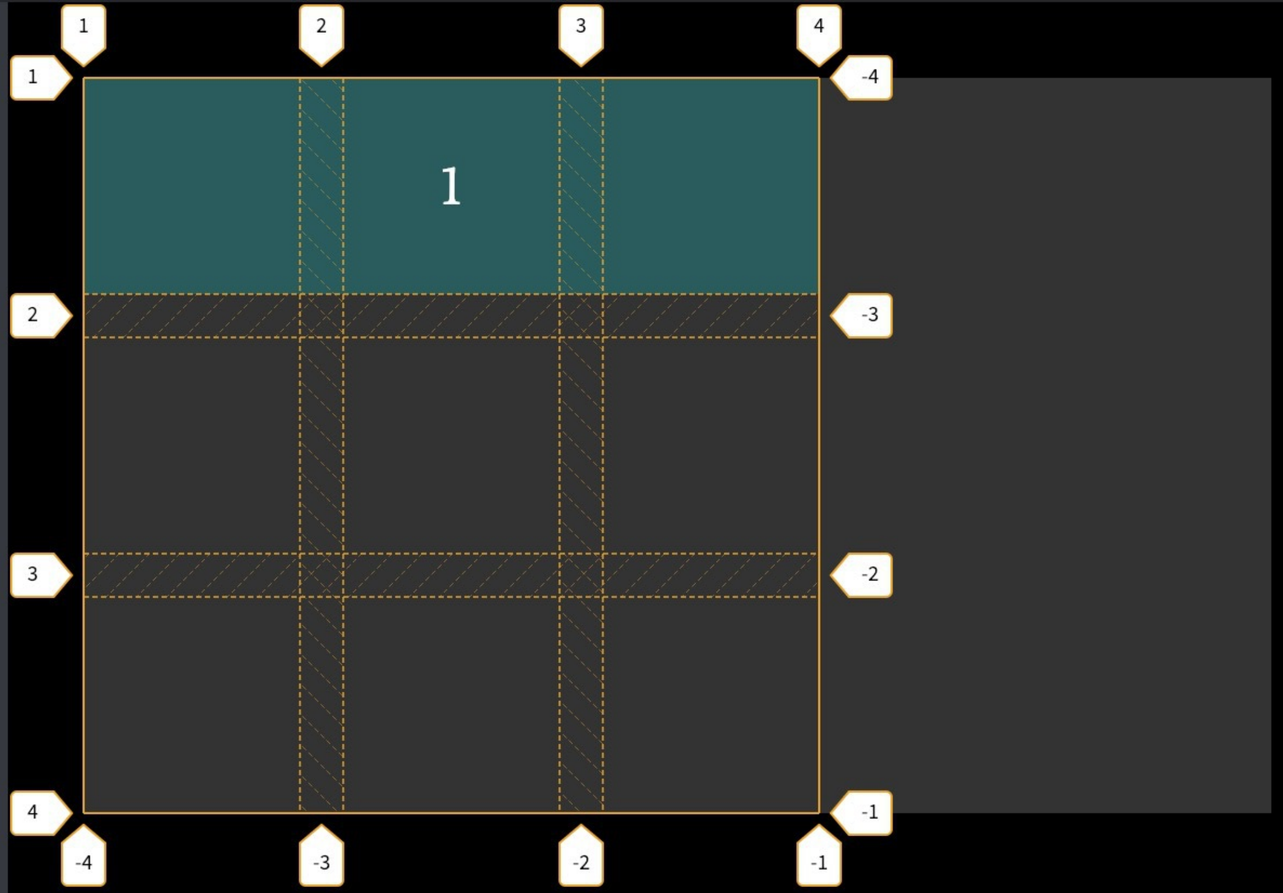
HTML

```html
<div class="grid-container">
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}
.grid-container > :nth-child(1) {
  grid-row-start: 1;
  grid-row-end: 2;
  grid-column-start: 1;
  grid-column-end: 4;
}
```

4 is fine because we know the number of tracks



JS

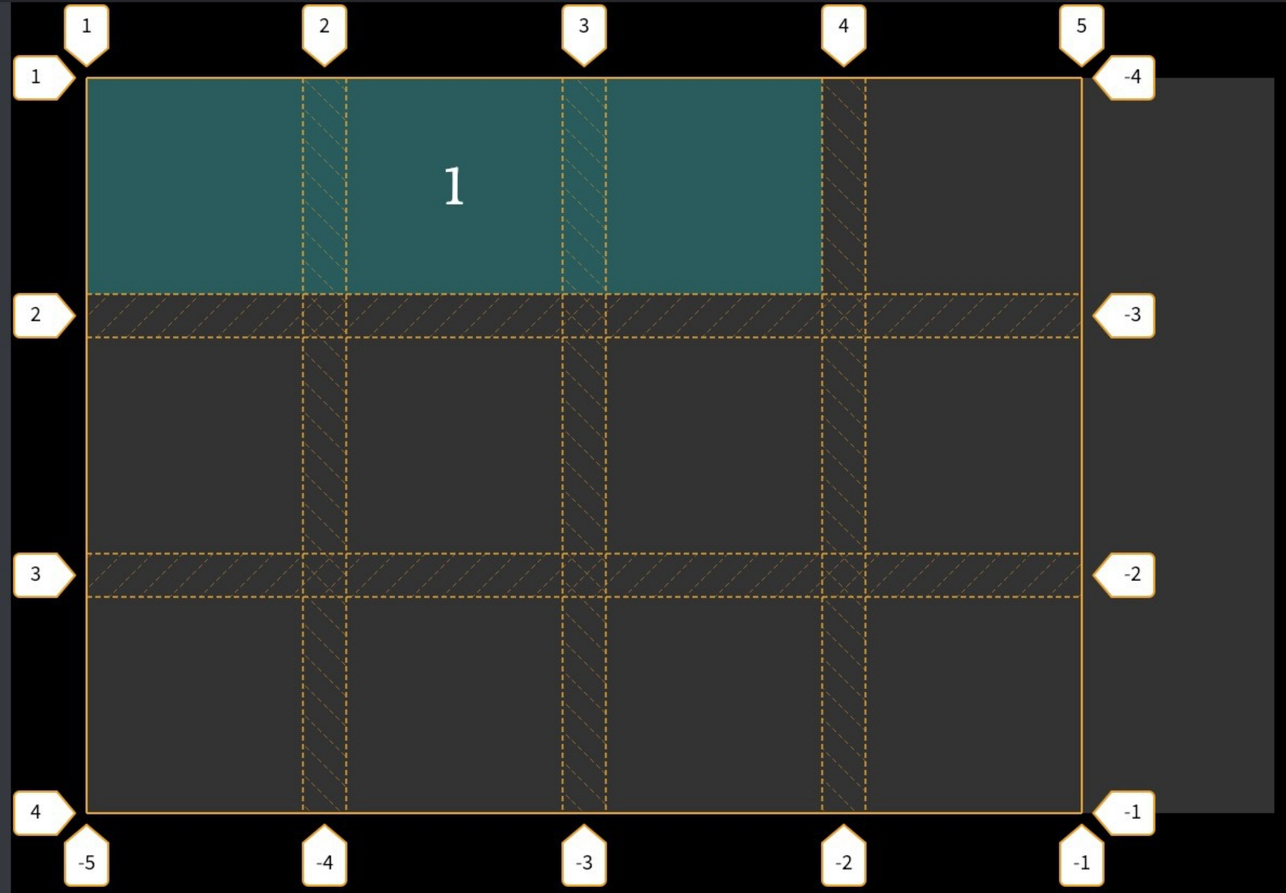
HTML

```html
<div class="grid-container">
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}
.grid-container > :nth-child(1) {
  grid-row-start: 1;
  grid-row-end: 2;
  grid-column-start: 1;
  grid-column-end: 4;
}
```

4 no longer works — the number of tracks changed



JS

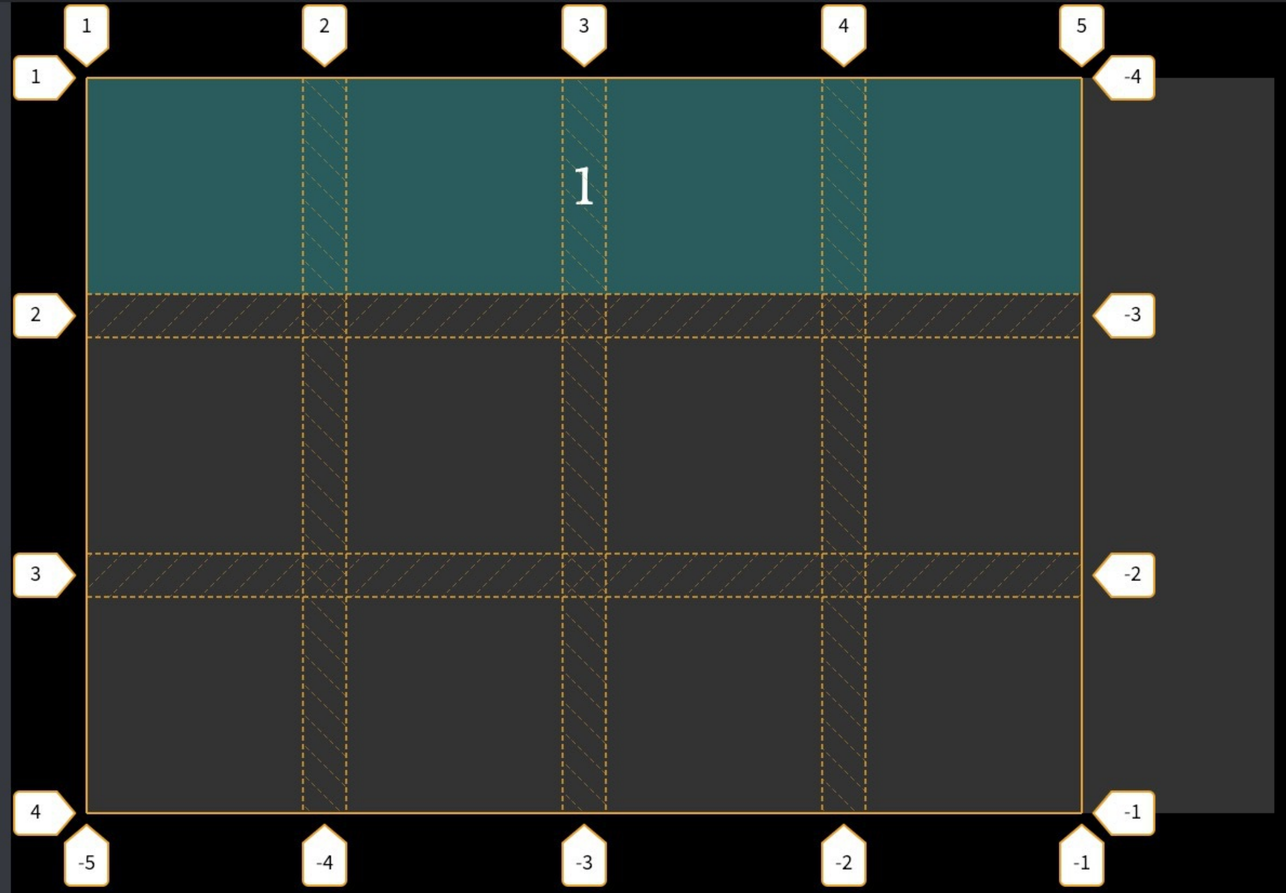
HTML

```
<div class="grid-container">
  <div></div>
</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}
.grid-container > :nth-child(1) {
  grid-row-start: 1;
  grid-row-end: 2;
  grid-column-start: 1;
  grid-column-end: -1;
}
```

-1 fixes it no matter how many tracks



JS

Named Lines

You can assign names to some or all grid lines, & those names are mixed with your track sizes

The names, which you create, must go inside `[ ]`

```
grid-template-rows: [start] 100px [line-2]
100px [line-3] 100px [end]
```

You do not have to name every single line, & can instead name just the key lines in your layout

```
grid-template-rows: [start] 100px 100px
[line-3] 100px [end]
```

You can give a line more than 1 name if it serves more than one purpose

```
grid-template-rows: [start] 100px 100px
[line-3 foo bar] 100px [end]
```

You can use the same name on multiple lines

```
grid-template-rows: [start] 100px 100px
[line-3 foo bar] 100px [end foo]
```

Once lines have names, you can use the name to place an item rather than a line number

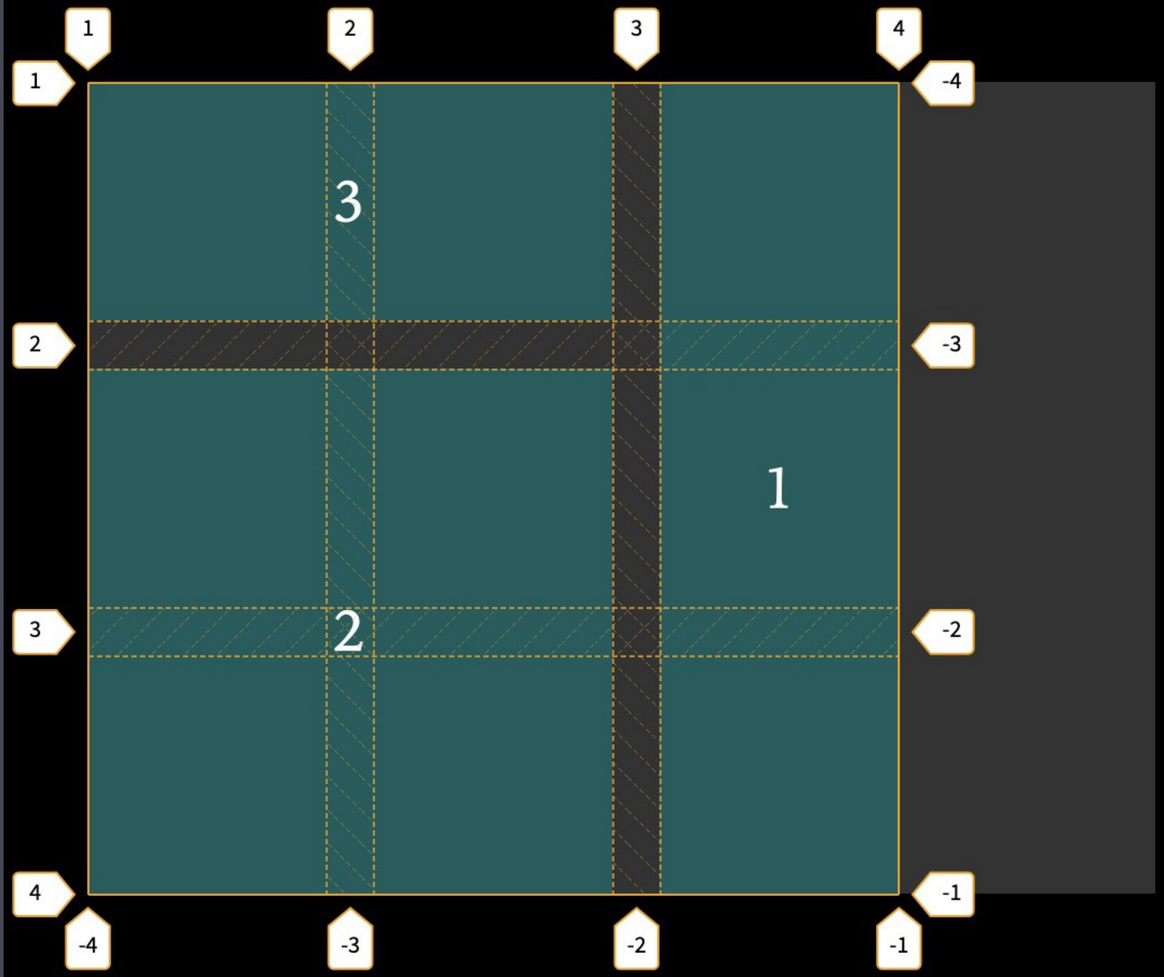
You can mix named lines & line numbers, but that will lead to INSTANT INSANITY

HTML

```
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

JS



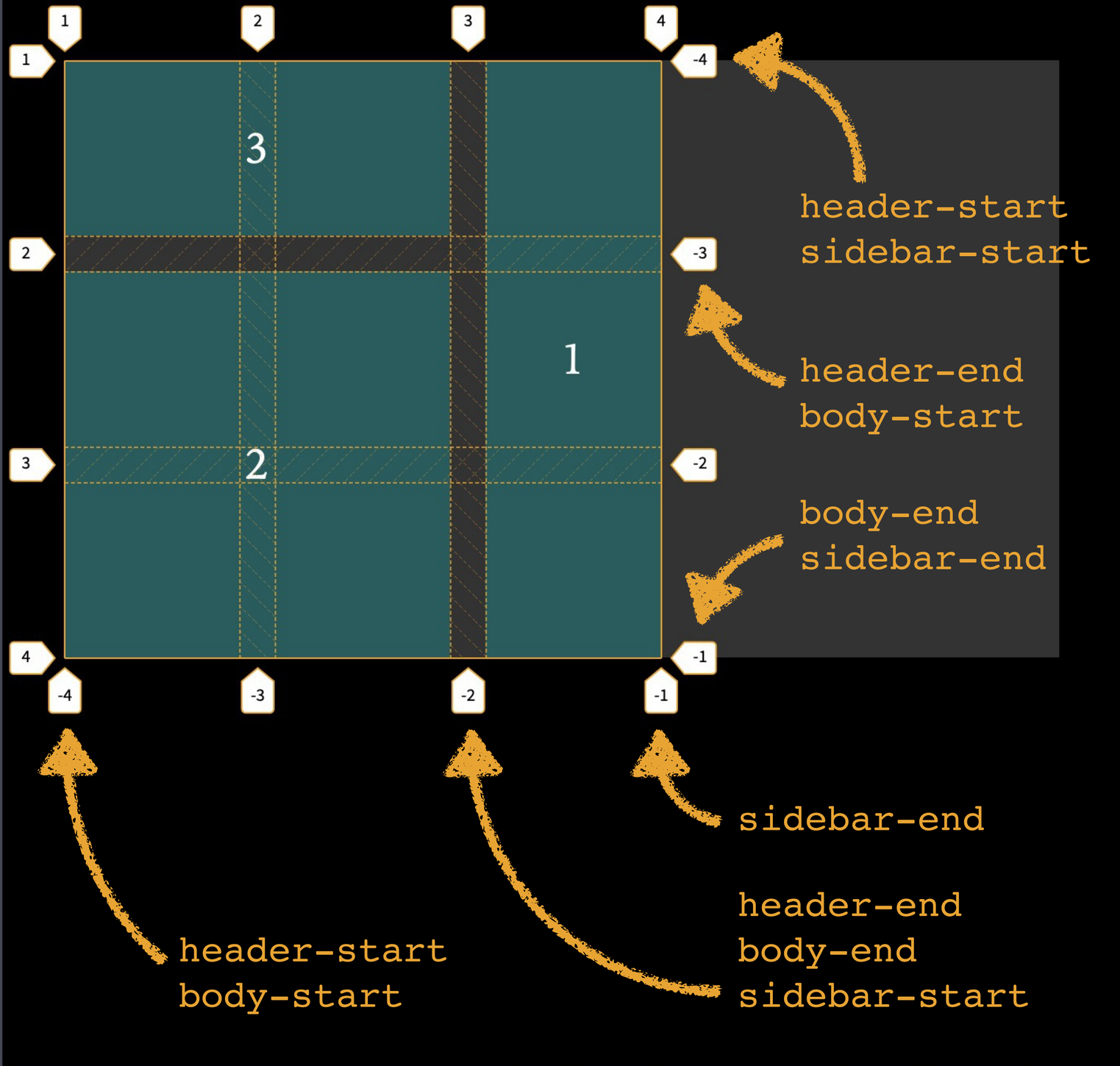
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS)

```scss
.grid-container {
  display: grid;
  grid-template-rows:
    [header-start sidebar-start]
    100px
    [header-end body-start]
    100px
    100px
    [body-end sidebar-end];
  grid-template-columns:
    [header-start body-start]
    100px
    100px
    [header-end body-end sidebar-start]
    100px
    [sidebar-end];
```

JS



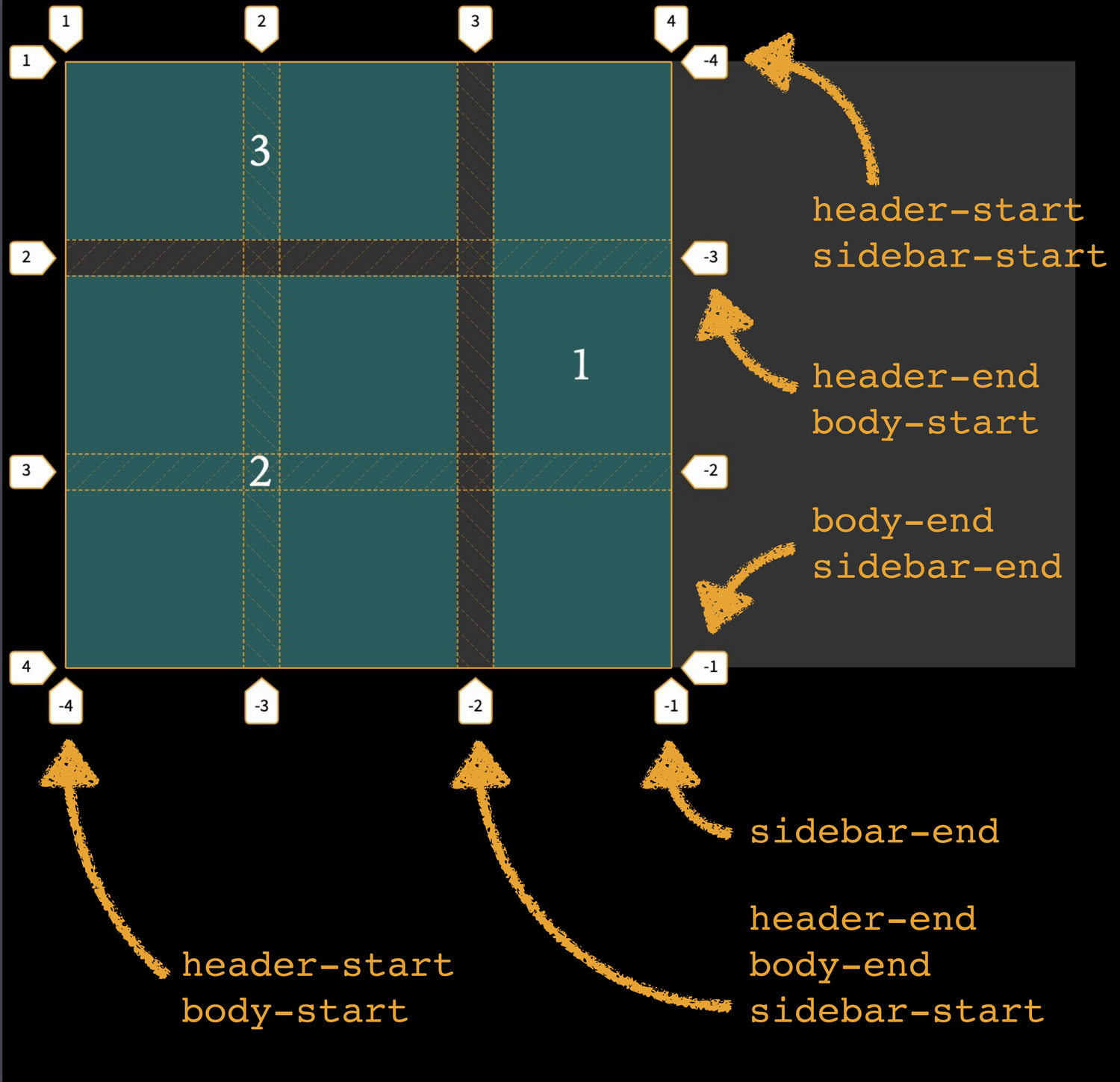
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container > :nth-child(1) {
  grid-row-start: sidebar-start;
  grid-row-end: sidebar-end;
  grid-column-start: sidebar-start;
  grid-column-end: sidebar-end;
}
.grid-container > :nth-child(2) {
  grid-row: body-start / body-end;
  grid-column: body-start / body-end;
}
.grid-container > :nth-child(3) {
  grid-area: header-start / header-start / header-end / header-end;
}
```

JS



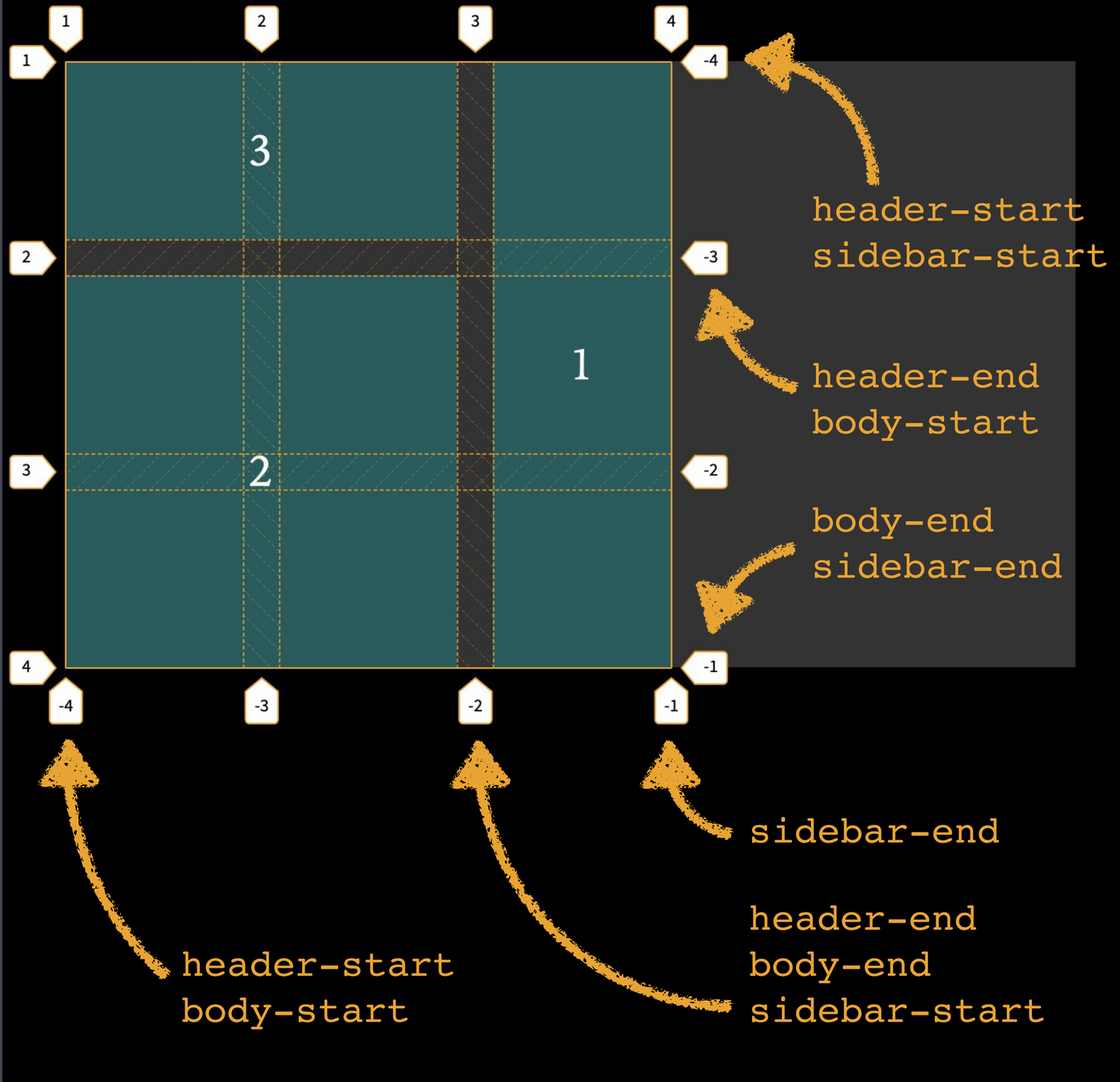
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container > :nth-child(1) {
  grid-row-start: sidebar-start;
  grid-row-end: sidebar-end;
  grid-column-start: sidebar-start;
  grid-column-end: sidebar-end;
}
.grid-container > :nth-child(2) {
  grid-row: body-start / body-end;
  grid-column: body-start / body-end;
}
.grid-container > :nth-child(3) {
  grid-area: header-start / header-start / header-end / header-end;
}
```

JS



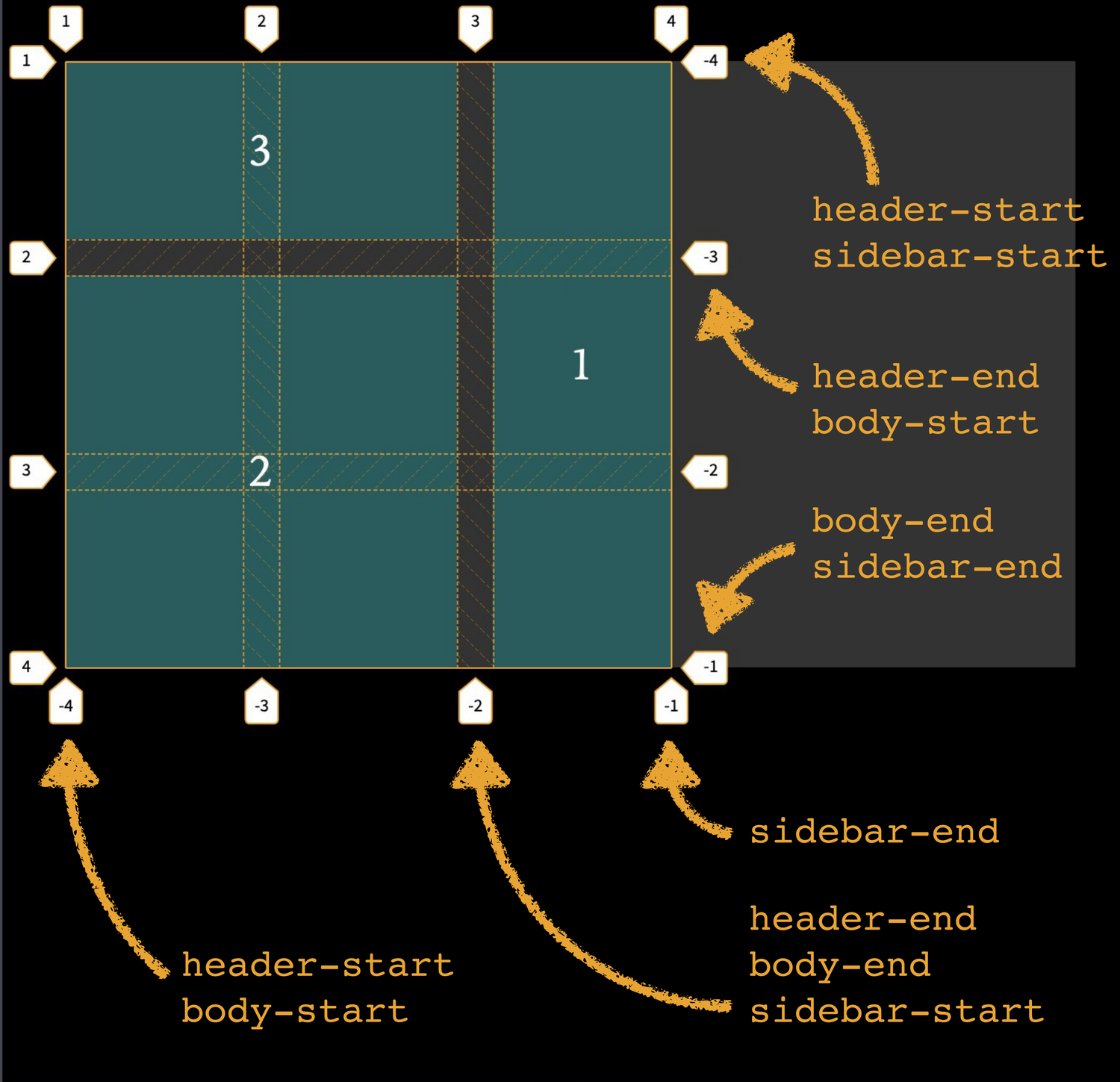
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container > :nth-child(1) {
  grid-row-start: sidebar-start;
  grid-row-end: sidebar-end;
  grid-column-start: sidebar-start;
  grid-column-end: sidebar-end;
}
.grid-container > :nth-child(2) {
  grid-row: body-start / body-end;
  grid-column: body-start / body-end;
}
.grid-container > :nth-child(3) {
  grid-area: header-start / header-start / header-end / header-end;
}
```

JS



Named Areas

grid-template-areas

grid-area

We know how to layout a grid by positioning grid items via grid lines

Another method: use grid *template areas*

`grid-template-areas`

Defines *named areas in the grid*

All named areas must be rectangular

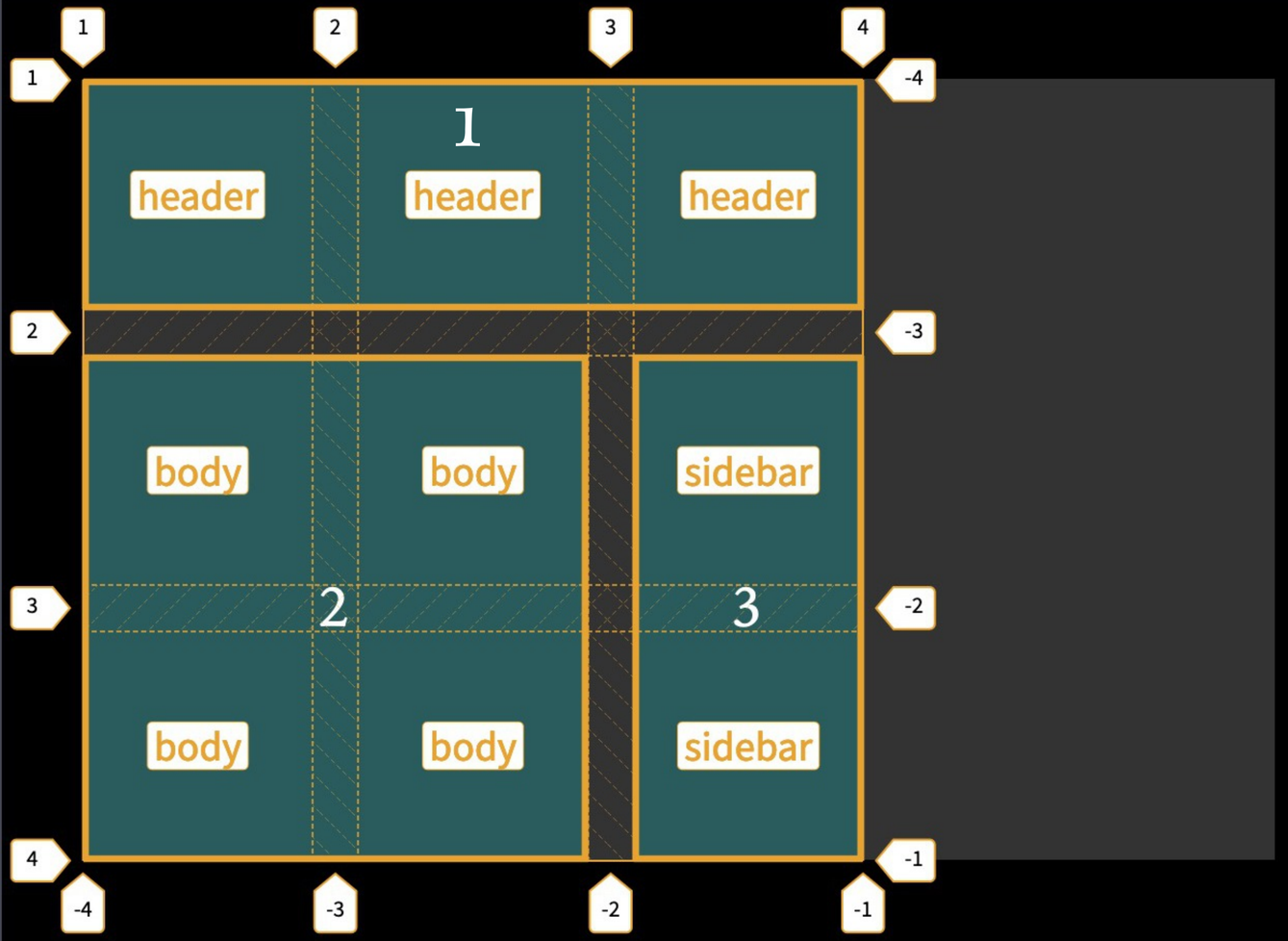
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS)

```scss
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-template-areas:
    "header header header"
    "body   body   sidebar"
    "body   body   sidebar";

  > :nth-child(1) {
    grid-area: header;
  }
  > :nth-child(2) {
    grid-area: body;
  }
  > :nth-child(3) {
    grid-area: sidebar;
  }
}
```

body is a rectangle



Everything lays out nicely!

JS

HTML

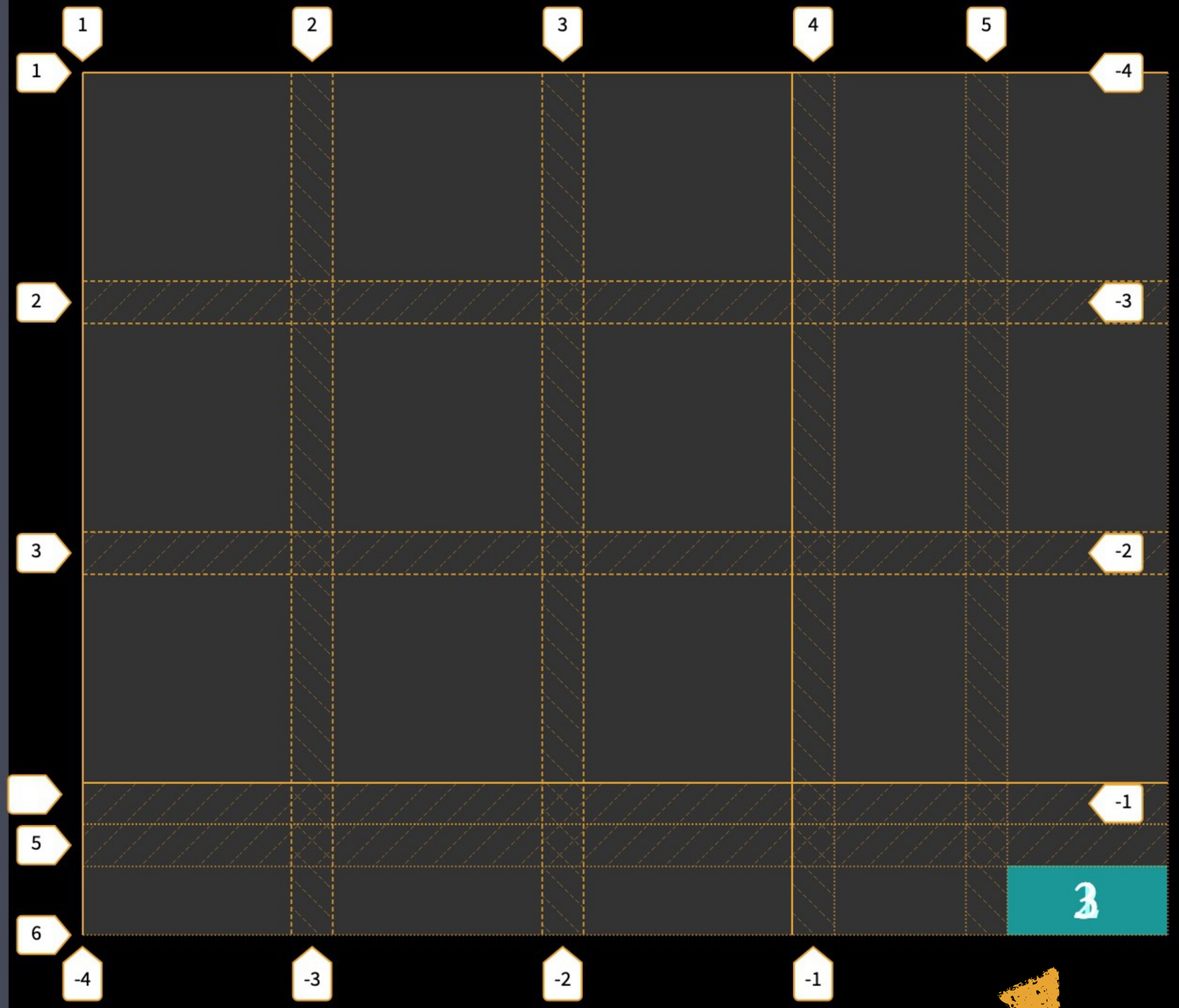
```
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS)

```
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-template-areas:
    "header body header"
    "body   body   sidebar"
    "body   body   sidebar";

  > :nth-child(1) {
    grid-area: header;
  }
  > :nth-child(2) {
    grid-area: body;
  }
  > :nth-child(3) {
    grid-area: sidebar;
  }
}
```

JS

body is no longer a rectangle



Ruh-roh

`grid-area`

Places grid item in a named area

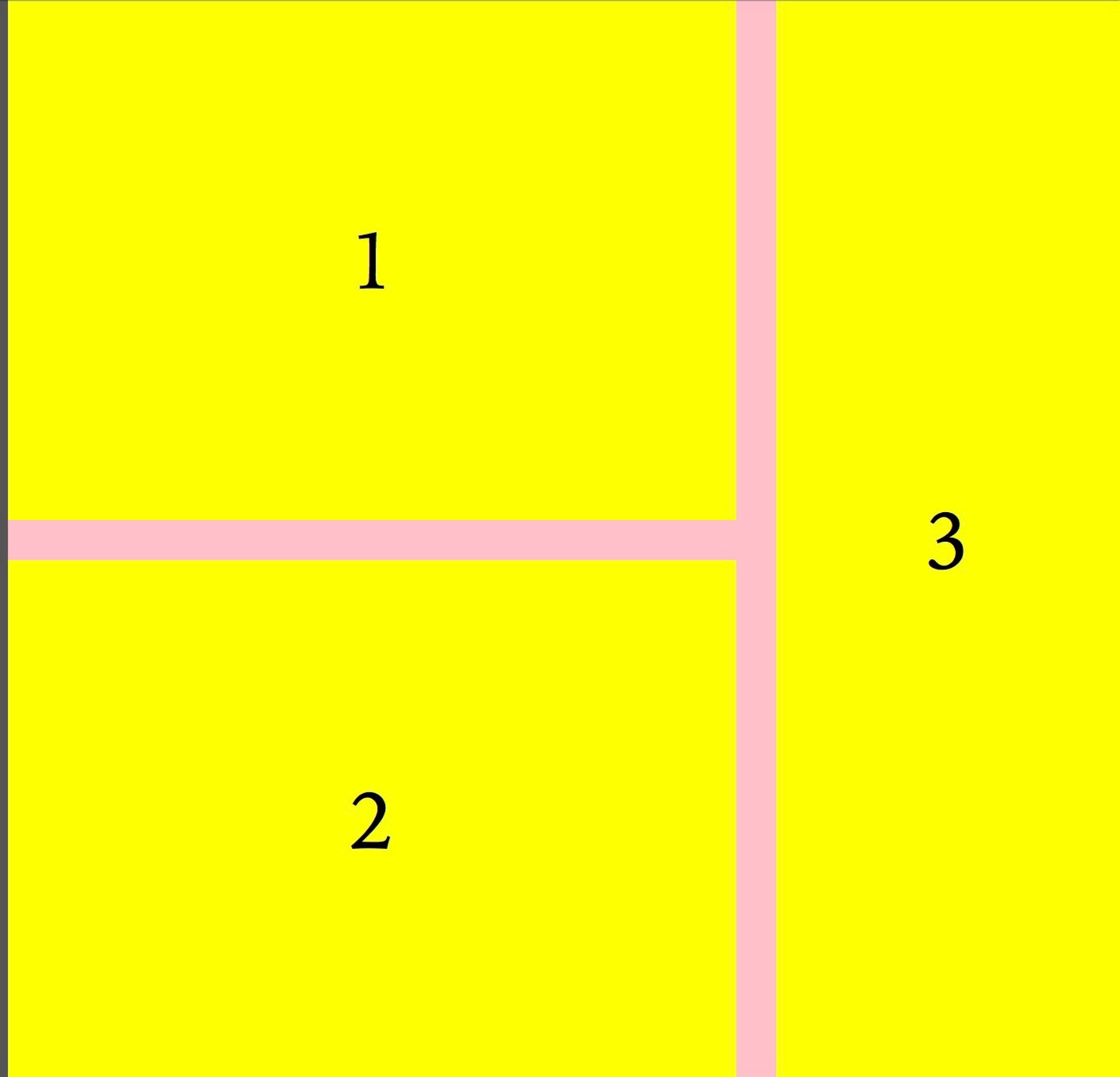
HTML

```html
<div class="grid-container">
  <div class="grid-item top">1</div>
  <div class="grid-item bottom">2</div>
  <div class="grid-item sidebar">3</div>
</div>
```

CSS

```css
.grid-container {
  width: 100vw;
  min-height: 100vh;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(2, 1fr);
  grid-gap: .5em;
  grid-template-areas:
    "tp tp side"
    "bt bt side";
}

.top {
  grid-area: tp;
}

.bottom {
```

JS



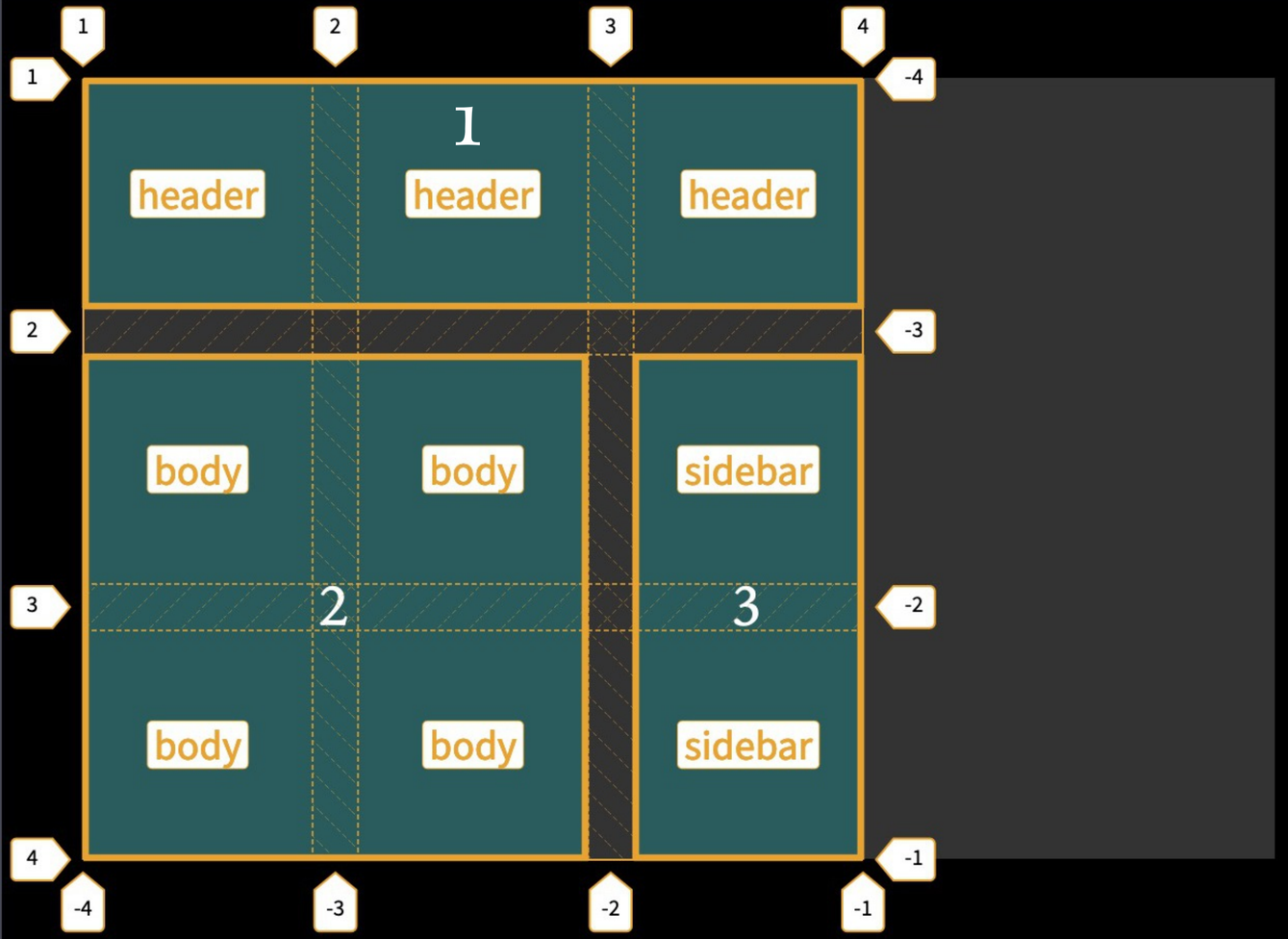
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS)

```scss
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-template-areas:
    "header header header"
    "body   body   sidebar"
    "body   body   sidebar";

  > :nth-child(1) {
    grid-area: header;
  }
  > :nth-child(2) {
    grid-area: body;
  }
  > :nth-child(3) {
    grid-area: sidebar;
  }
}
```

JS



When you define a named area, the lines around those areas are automatically assigned implicit line names

You can use these implicit line names when placing items via named lines

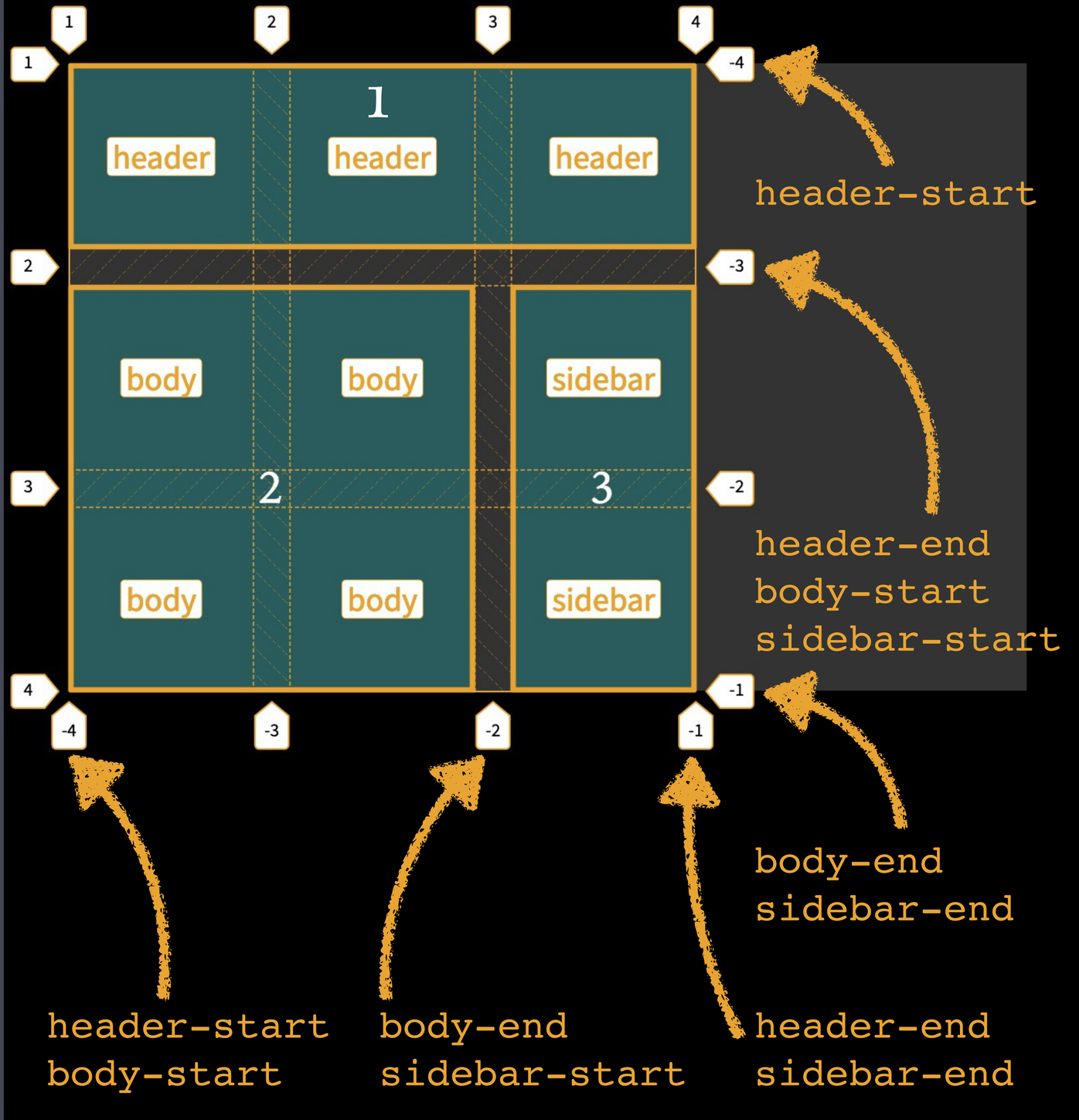
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS)

```scss
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-template-areas:
    "header header header"
    "body   body   sidebar"
    "body   body   sidebar";

  > :nth-child(1) {
    grid-area: header;
  }
  > :nth-child(2) {
    grid-area: body;
  }
  > :nth-child(3) {
    grid-area: sidebar;
  }
}
```

JS



Represents a *null (empty) cell token*

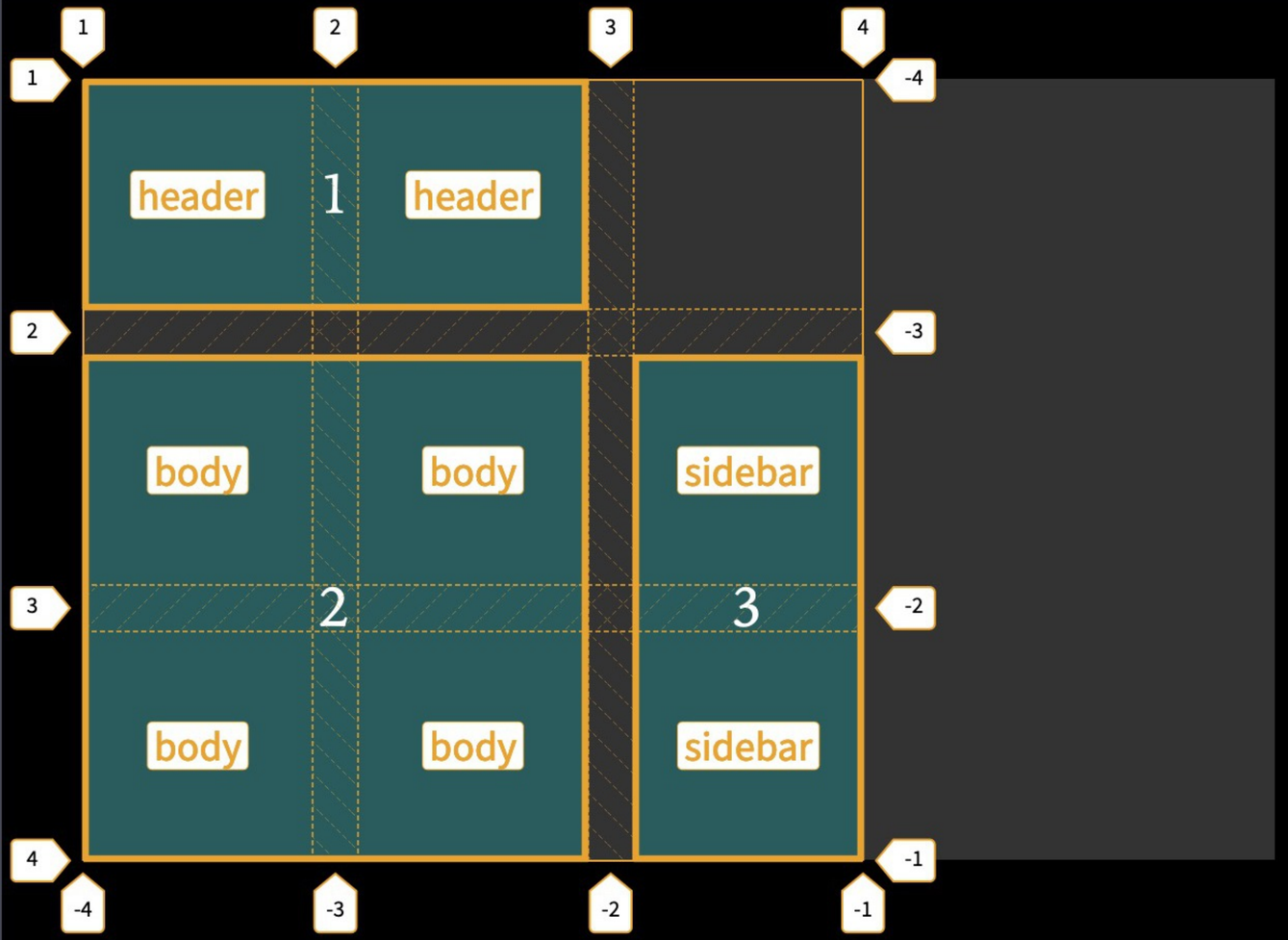
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS)

```scss
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-template-areas:
    "header header ."
    "body   body   sidebar"
    "body   body   sidebar";

  > :nth-child(1) {
    grid-area: header;
  }
  > :nth-child(2) {
    grid-area: body;
  }
  > :nth-child(3) {
    grid-area: sidebar;
  }
}
```

JS



					iOS		
grid-template-areas	–	16	52	10.1	10.3	57	57
grid-area	–	16	52	10.1	10.3	57	57

Spans

`span <integer>`

Define *how many lines the grid area should extend*

If you don't supply a `<integer>`, `span` defaults to `1` (no `-<integers>` or `0`

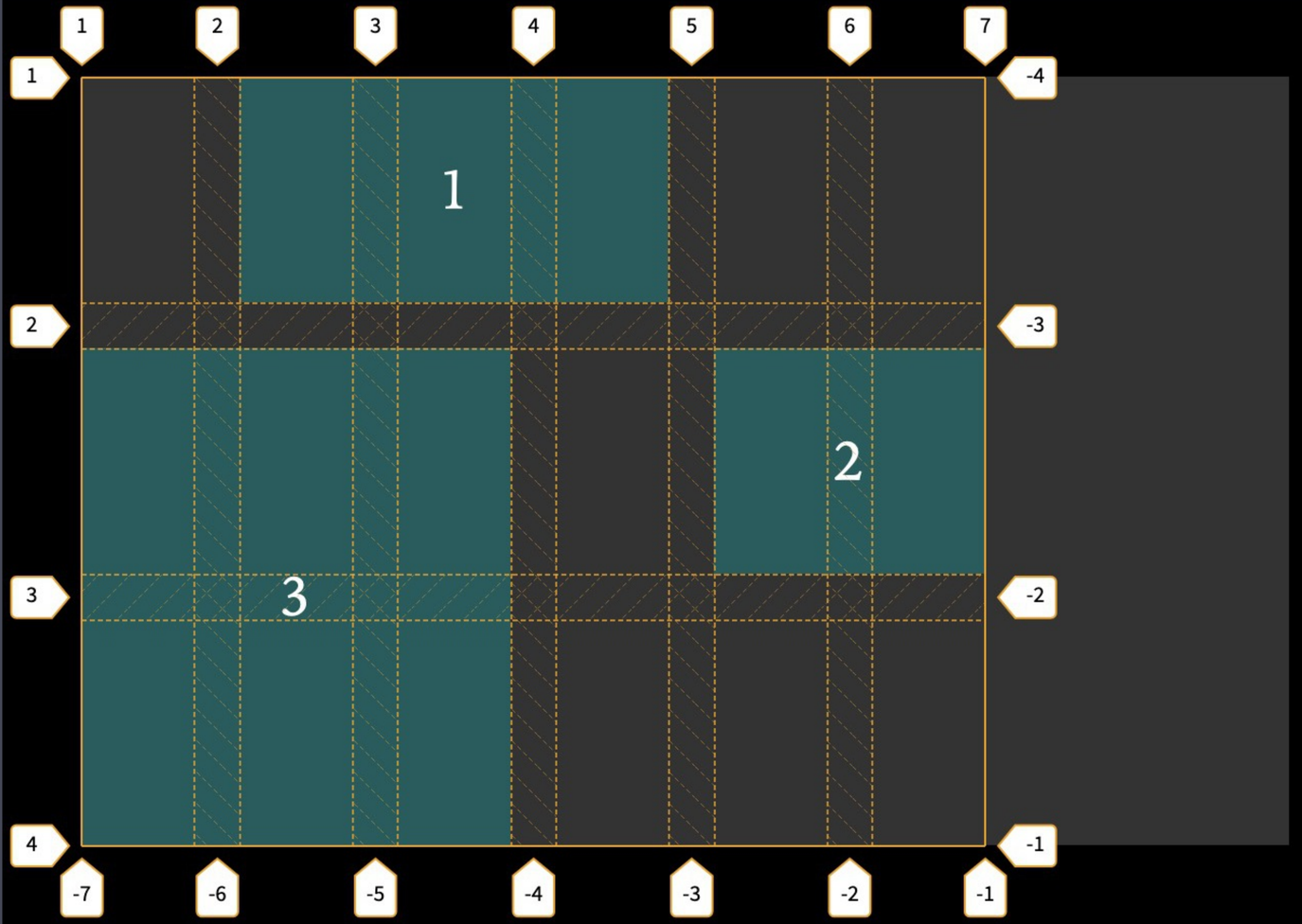
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(6, 50px);
  grid-template-rows: 100px 100px 100px;
}
```

JS

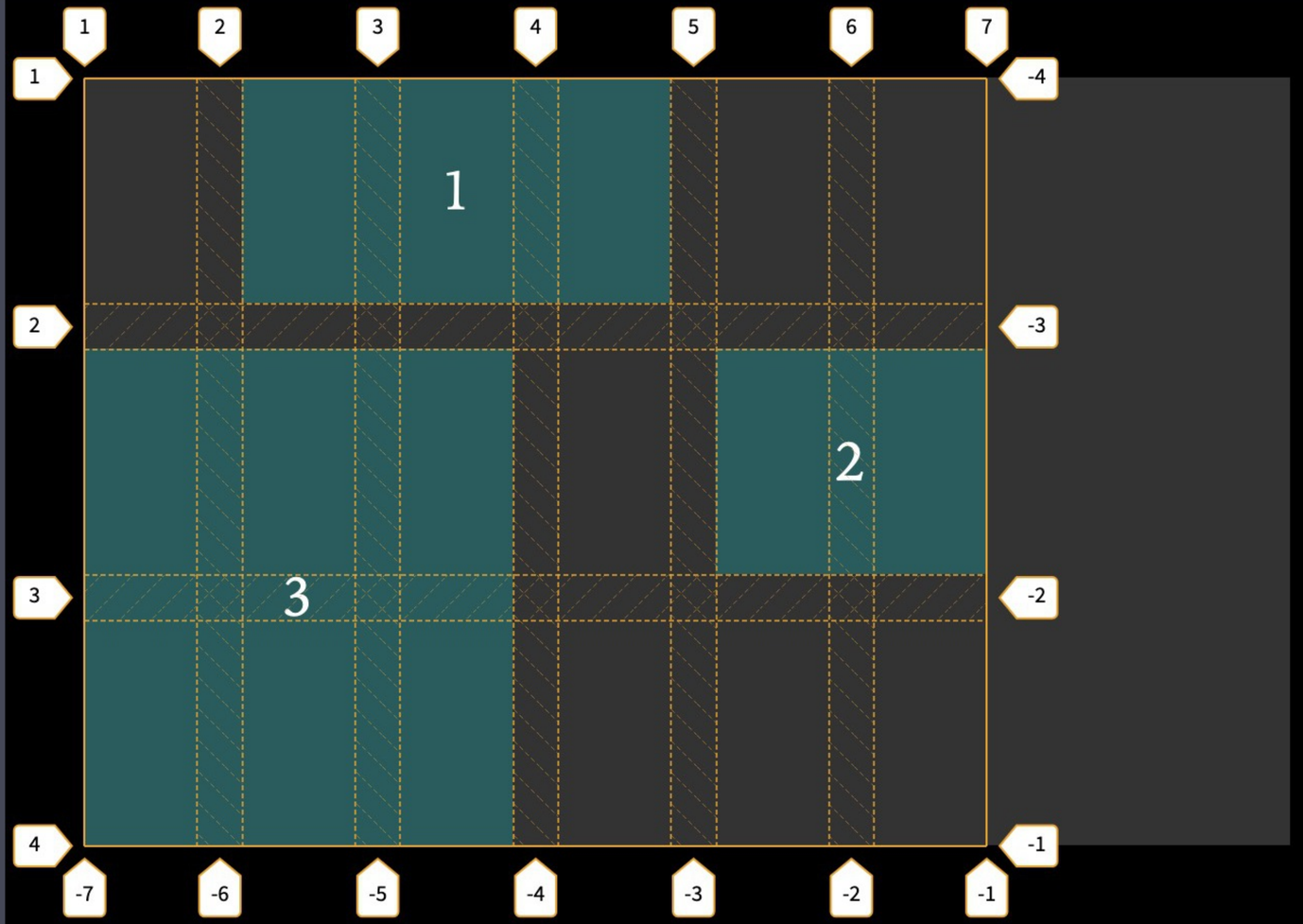


HTML

CSS (SCSS) Compiled

```scss
.grid-container > :nth-child(1) {
  grid-row-start: 1;
  grid-column-start: 2;
  grid-row-end: 2;
  grid-column-end: span 3;
}
.grid-container > :nth-child(2) {
  grid-row-start: 2;
  grid-column-start: span 2;
  grid-row-end: 3;
  grid-column-end: -1;
}
.grid-container > :nth-child(3) {
  grid-row-start: 2;
  grid-column-start: 1;
  grid-row-end: span 2;
  grid-column-end: span 3;
}

```

JS



Sizing Tracks

Various ways to size row & column tracks

- `<length>`
- `<flex>` `fr` unit
- `max-content`
- `min-content`
- `fit-content()`
- `minmax()`
- `auto` (default)
- `<percentage>`
- `subgrid`

`<length>` data type; e.g.:

- `10px`
- `10em`
- `10rem`
- `10vh`

More in *CSS Typography* & *CSS Data Types*

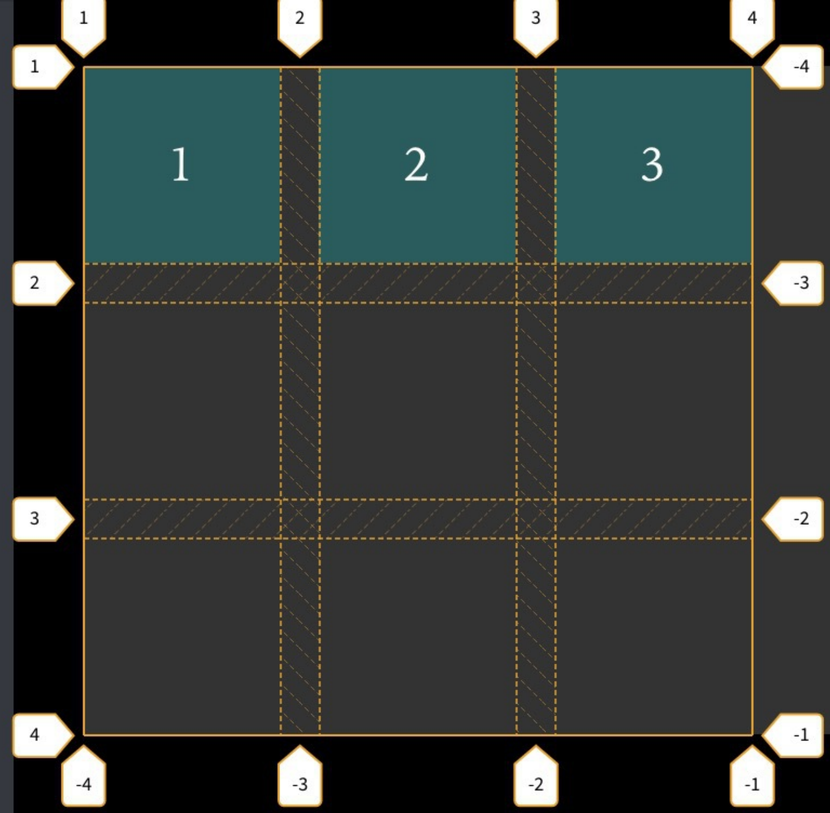
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 20px;
}
```

JS



`fr`

Grid introduces a new unit: `fr`, short for *fraction of the free space* in the grid container

`fr` is calculated after any non-flexible items

```
grid-template-columns: 200px 1fr 200px;
grid-template-rows: 1fr 2fr 1fr;
```

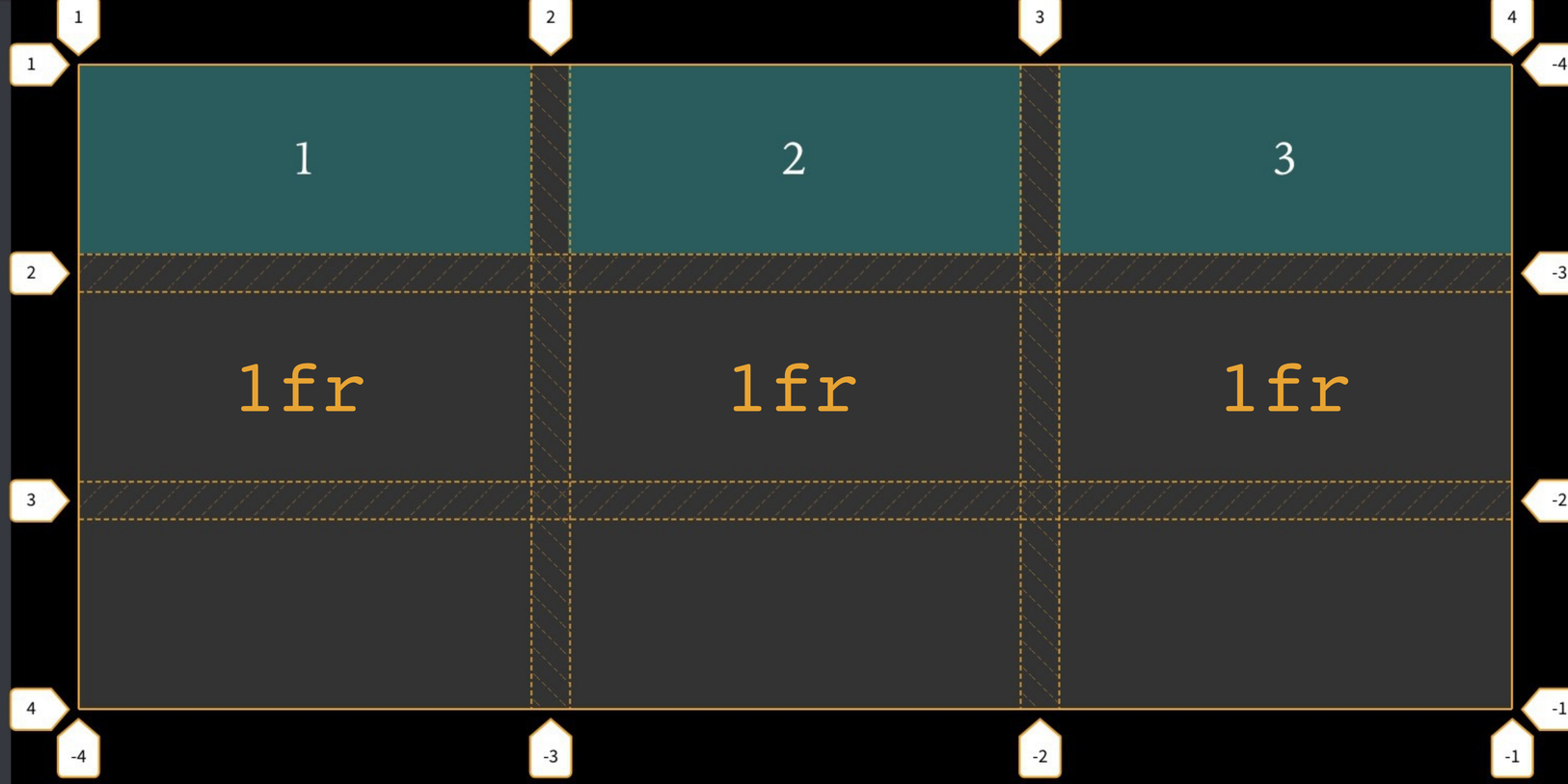
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 100px 100px 100px;
}
```

JS



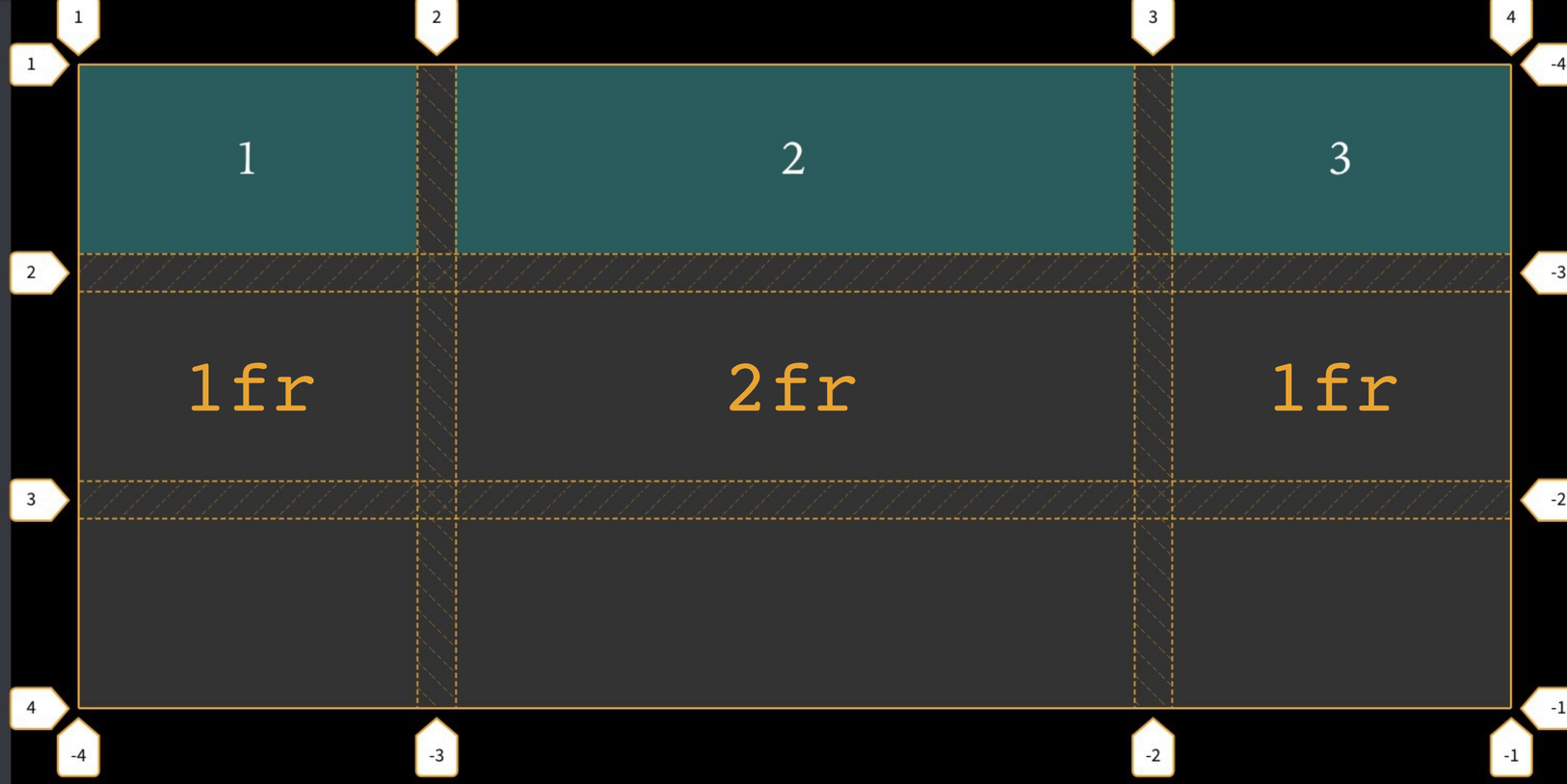
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 1fr 2fr 1fr;
  grid-template-rows: 100px 100px 100px;
}
```

JS



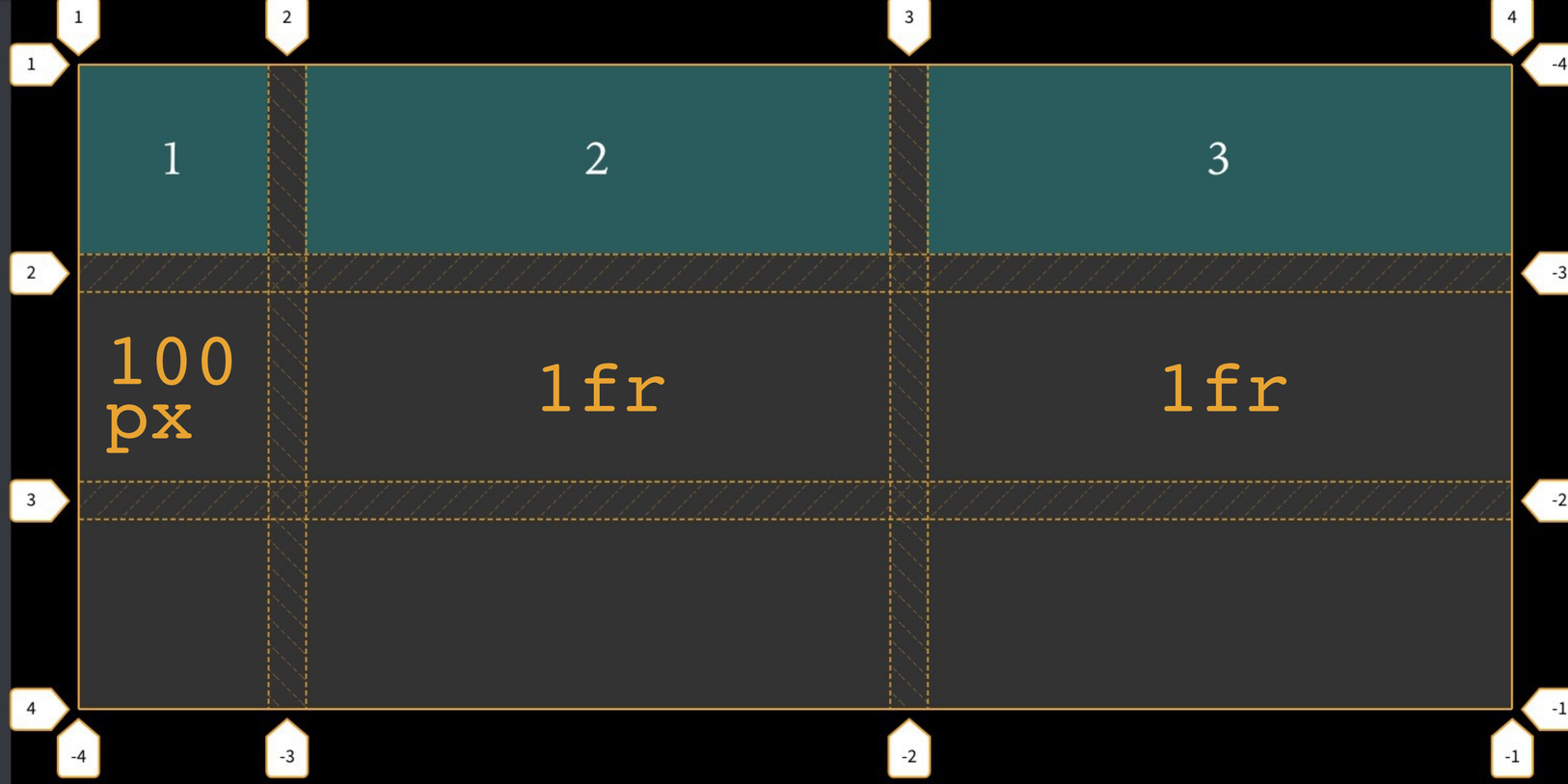
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 1fr 1fr;
  grid-template-rows: 100px 100px 100px;
}
```

JS



`max-content`

Largest content in a grid item determines the size of the track

Similar to `white-space: nowrap`

No good use cases (prove us wrong!) — it's the W3C being completionist again

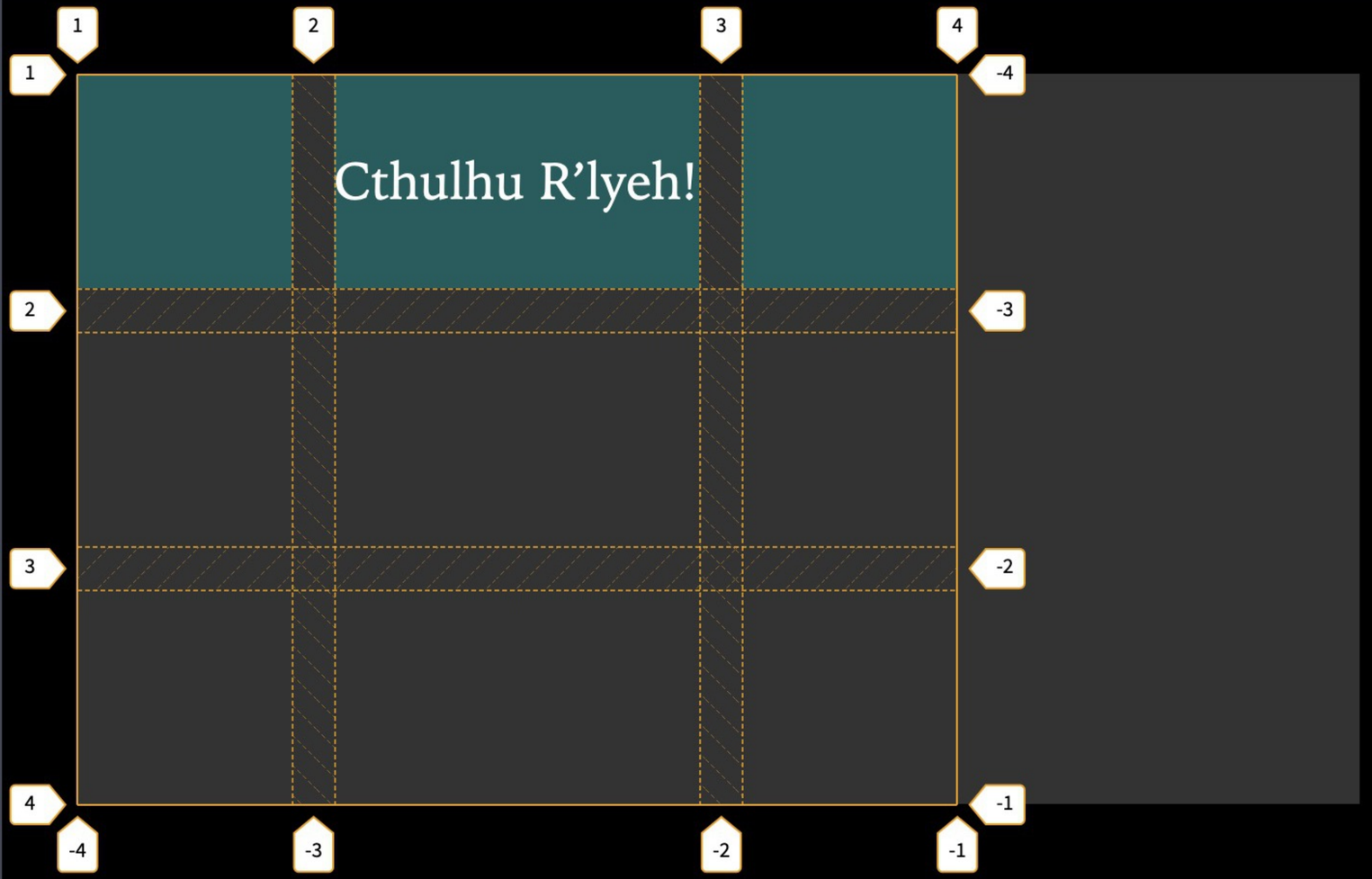
HTML

```html
<div class="grid-container">
  <div></div>
  <div>Cthulhu R’lyeh!</div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px max-content 100px;
  grid-template-rows: 100px 100px 100px;
}

/* uninteresting stuff below here */
.grid-container > ::before {
  content: none;
}
```

JS



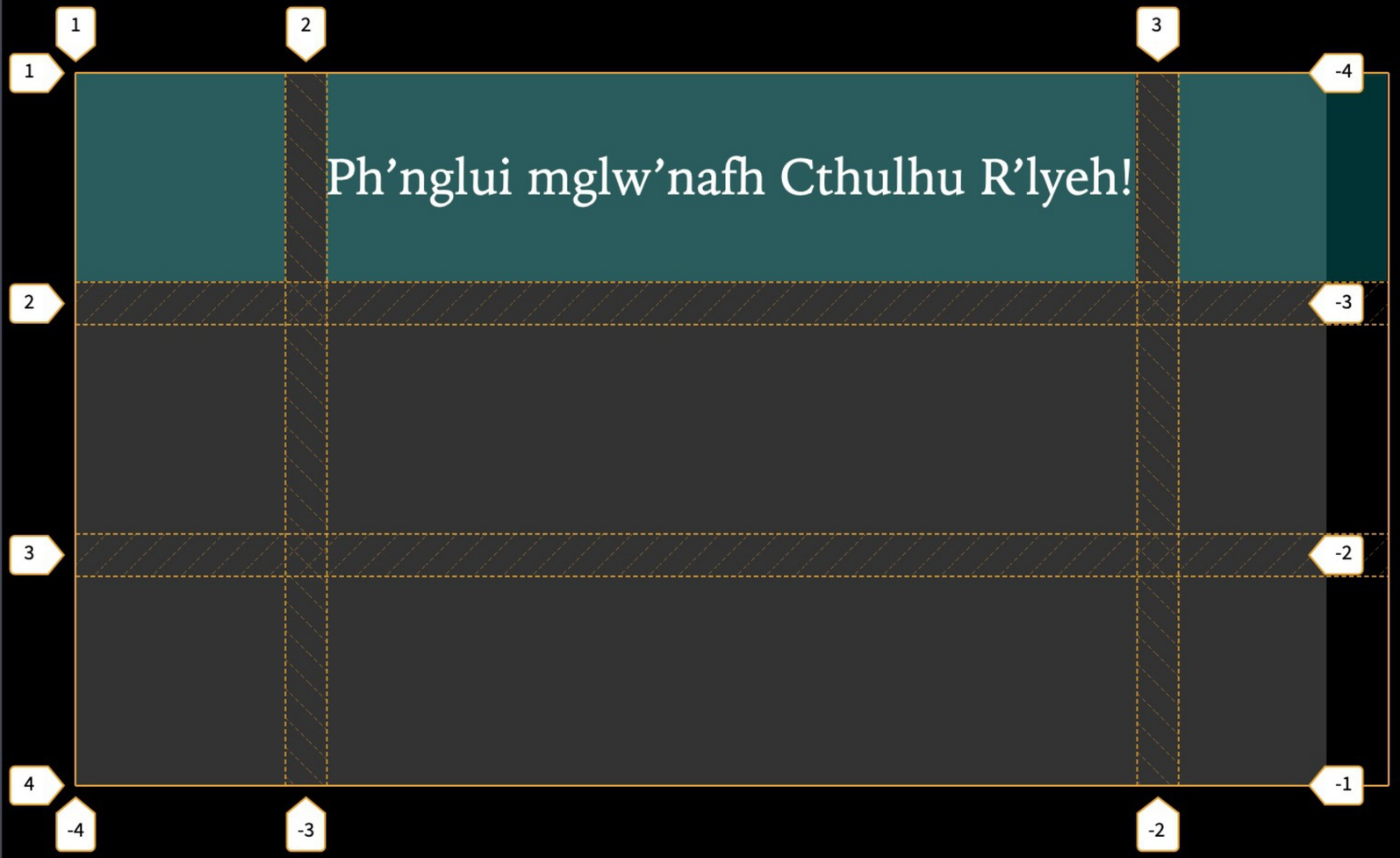
HTML

```html
<div class="grid-container">
  <div></div>
  <div>Ph’nglui mglw’nafh Cthulhu R’lyeh!</div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px max-content 100px;
  grid-template-rows: 100px 100px 100px;
}

/* uninteresting stuff below here */
.grid-container > ::before {
  content: none;
}
```

JS



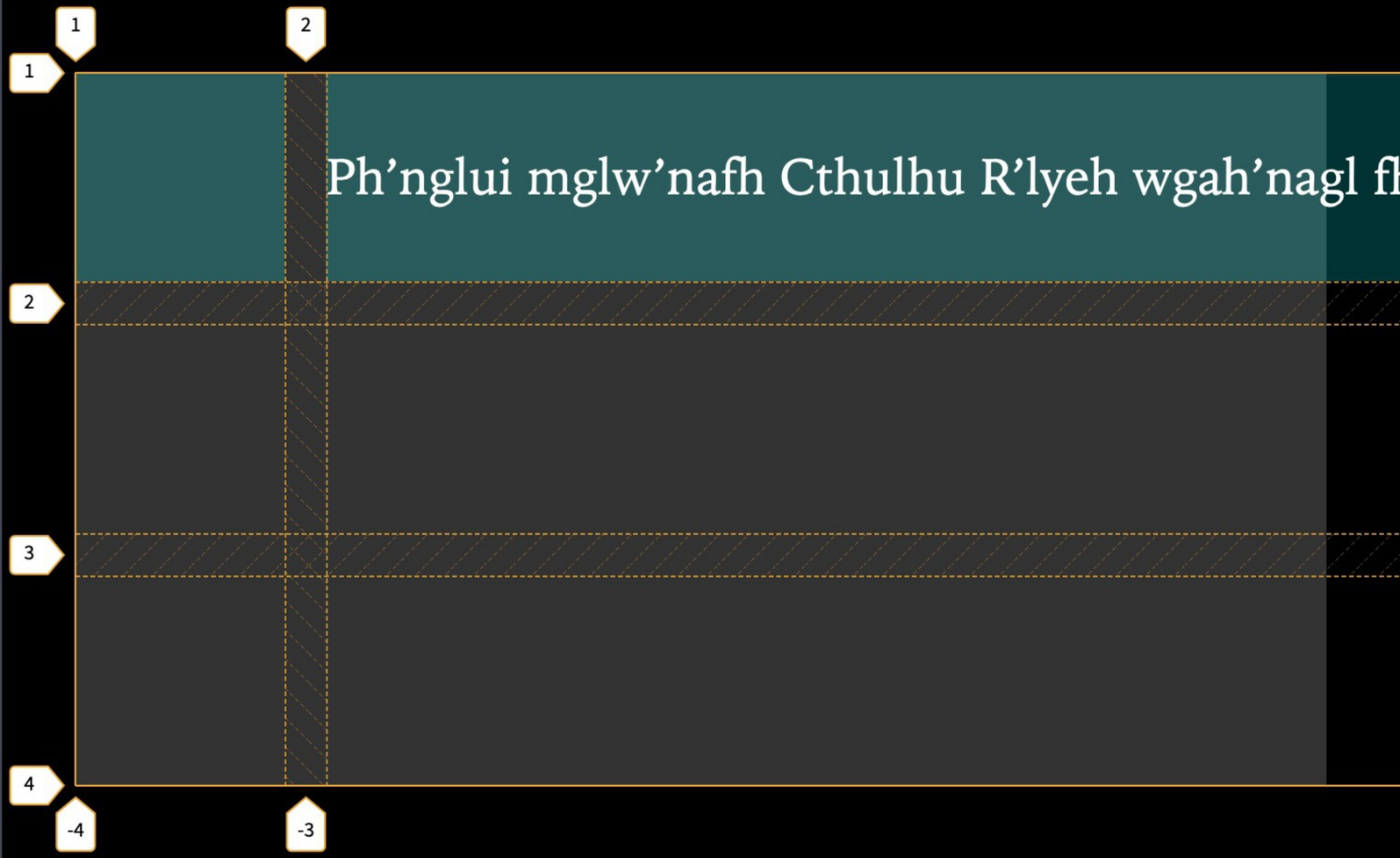
HTML

```html
<div class="grid-container">
  <div></div>
  <div>Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!</div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px max-content 100px;
  grid-template-rows: 100px 100px 100px;
}

/* uninteresting stuff below here */
.grid-container > ::before {
  content: none;
}
```

JS



`min-content`

Track becomes as *small as it can be to accommodate the width of the longest word*, image, video, fixed-size `<div>`, & so on

Smallest content in a grid item determines the size of the track, forcing all text to wrap

Useful for limiting text to the width of an image or video, for instance

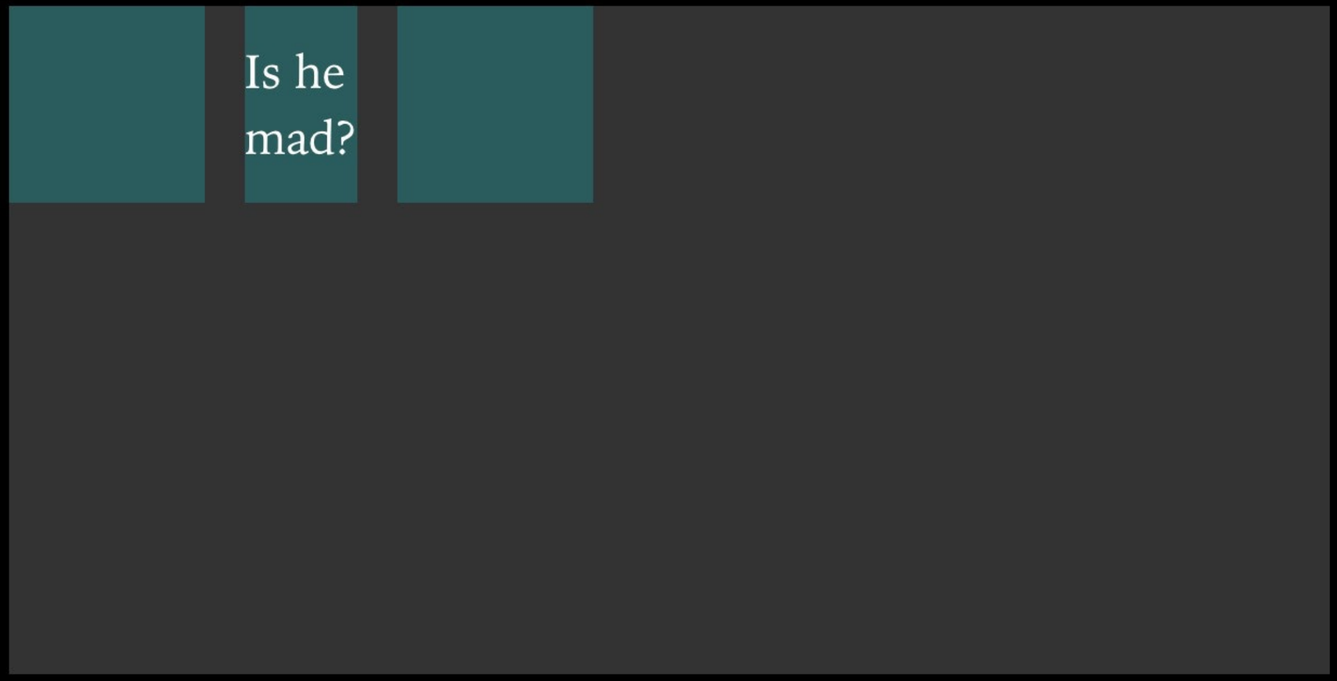
HTML

```html
<div class="grid-container">
  <div></div>
  <div>Is he mad?</div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px min-content 100px;
  grid-template-rows: 100px 100px 100px;
}

/* uninteresting stuff below here */
.grid-container > ::before {
  content: none;
}

```

JS



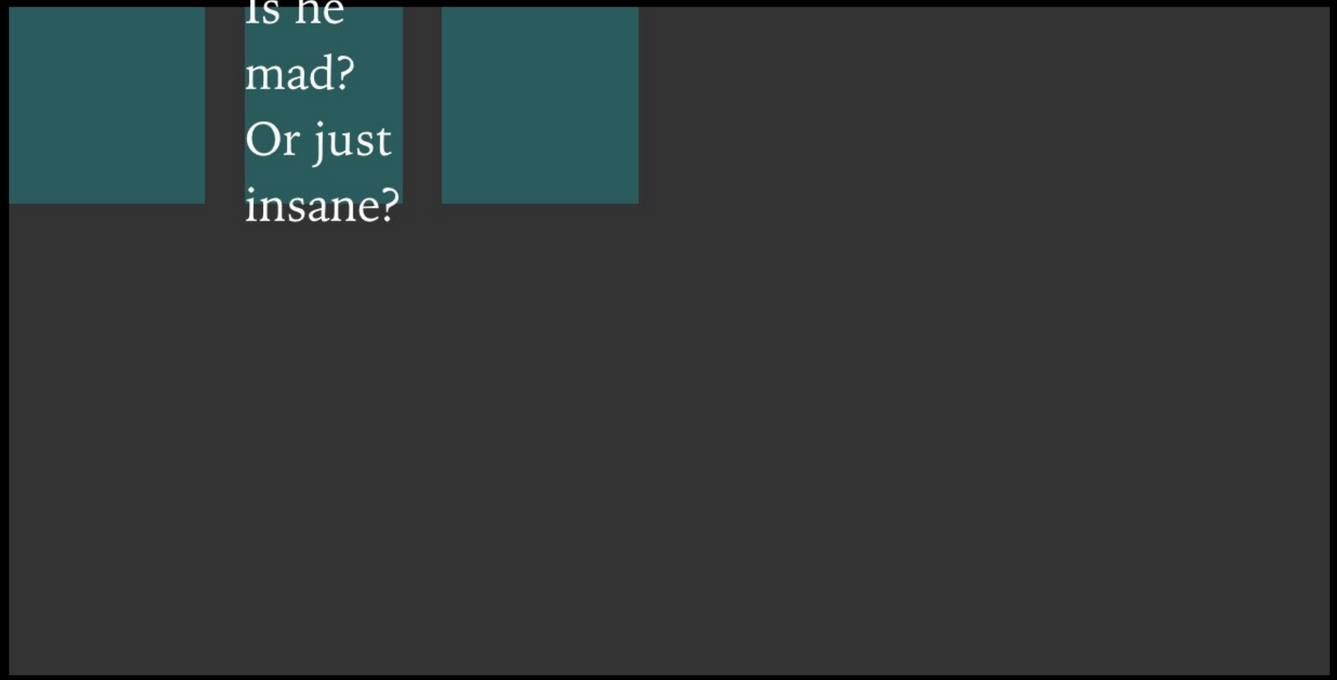
HTML

```html
<div class="grid-container">
  <div></div>
  <div>Is he mad? Or just insane?</div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px min-content 100px;
  grid-template-rows: 100px 100px 100px;
}

/* uninteresting stuff below here */
.grid-container > ::before {
  content: none;
}

```

JS



`fit-content(<length>|<percentage>)`

Largest content in a grid item determines the size of the track, but not bigger than `(<length>|<percentage>)`

Useful!

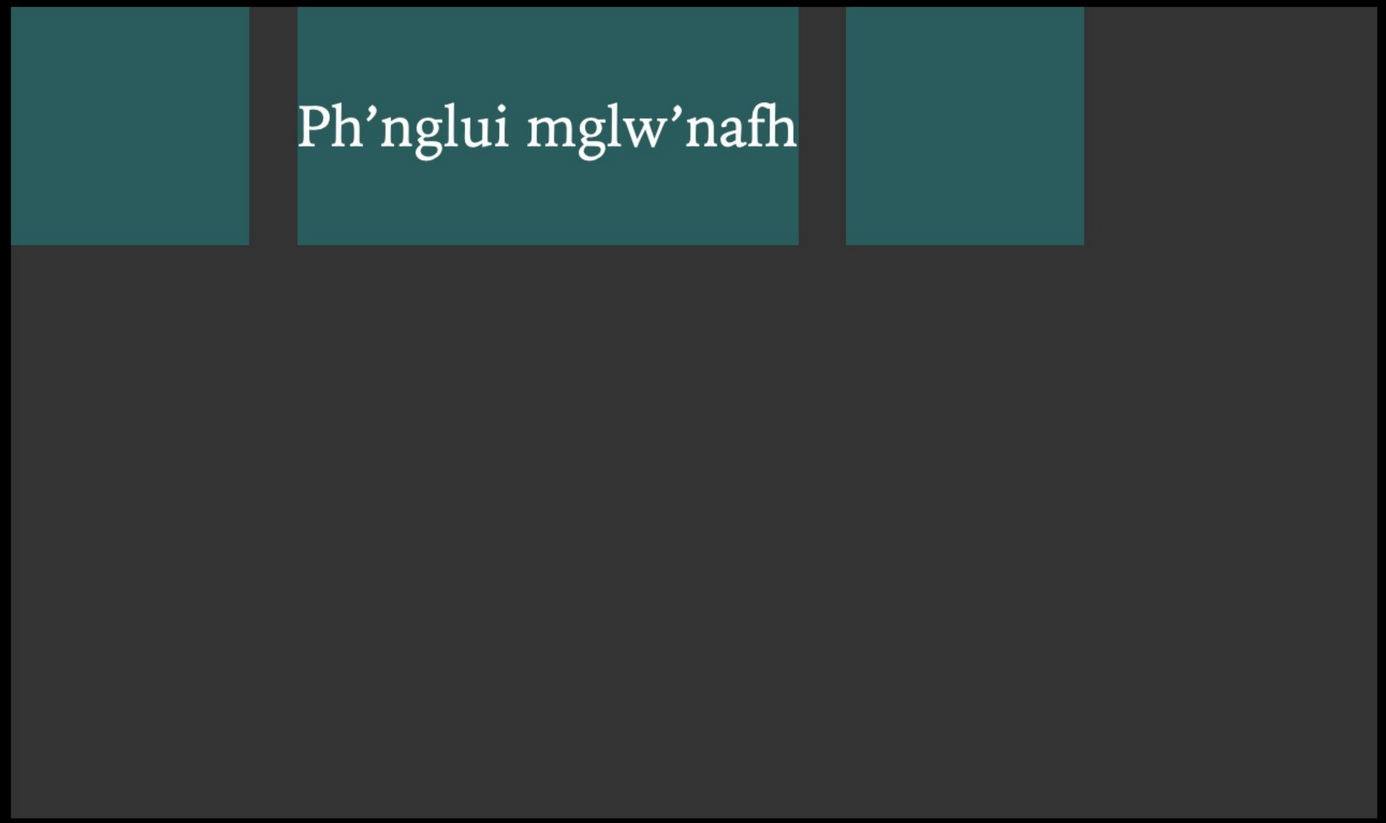
HTML

```
<div class="grid-container">
  <div></div>
  <div>Ph’nglui mglw’nafh</div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px fit-content(250px) 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



In this case, the text is less than 250px, so the grid item is as long as the text

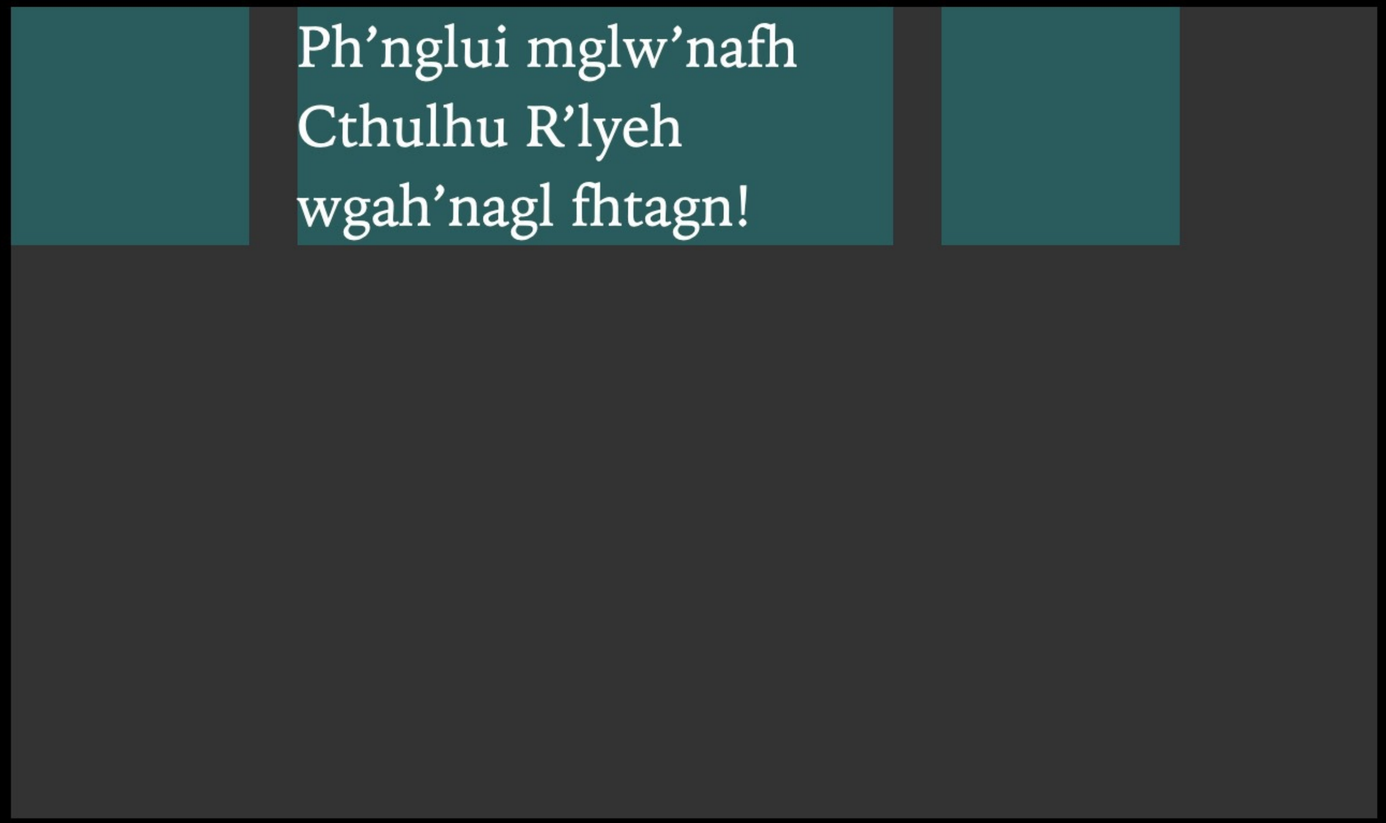
HTML

```
<div class="grid-container">
  <div></div>
  <div>Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!</div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px fit-content(250px) 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



This text is bigger than 250px, so it’s wrapped

`minmax(min, max)`

Defines the size of a track as a *minimum to maximum range*

HTML

```html
<div class="grid-container">
  <div></div>
  <div>Cthulhu R’lyeh!</div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px minmax(100px, 250px) 100px;
  grid-template-rows: 100px 100px 100px;
}

/* uninteresting stuff below here */
.grid-container > ::before {
  content: none;
}
```

JS

The middle grid item is 250px, which is the maximum size we set

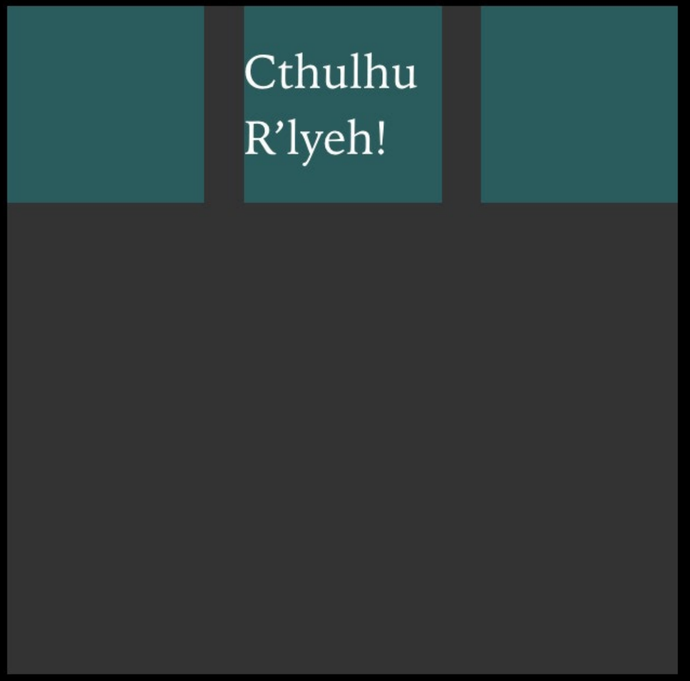
HTML

```html
<div class="grid-container">
  <div></div>
  <div>Cthulhu R’lyeh!</div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px minmax(100px, 250px) 100px;
  grid-template-rows: 100px 100px 100px;
}

/* uninteresting stuff below here */
.grid-container > ::before {
  content: none;
}
```

JS



The middle grid item will not get smaller than 100px

`auto`

Width automatically calculated based on content, similar to table layout algorithm

You abdicate control when you use `auto`, so you may get unexpected results

Note that `auto` track sizes (and only `auto` track sizes) can be stretched by `align-content: stretch` & `justify-content: stretch` (more later)!

HTML

```
<div class="grid-container">
  <div>Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!</div>
  <div>Nyarlathotep</div>
  <div>Great Old One</div>
</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: auto auto auto;
}
```

JS

Ph’nglui mglw’nafh Cthulhu R’lyeh wgah’nagl fhtagn!	Nyarlathotep	Great Old One

The rendering engine determined those widths, which may not be what you wanted

`<percentage>`

Using `%` with `grid-template-columns` & `grid-template-rows` vastly complicates things because you have to take `grid-gap` into account

Do *not* use `<percentage>` — use `fr` instead

`subgrid`

Tells a *child grid to re-use the parent grid's lines* for rows &/or columns

We'll discuss this later

HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS)

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  // grid-template-columns: 1fr 1fr 1fr;
  // grid-template-columns: 1fr 2fr 1fr;
  // grid-template-columns: 100px 1fr 1fr;
  // grid-template-columns: 100px minmax(100px, 250px) 100px;
  // grid-template-columns: 100px max-content 100px;
  // grid-template-columns: 100px min-content 100px;
  // grid-template-columns: 100px fit-content(250px) 100px;
  // grid-template-columns: 100px minmax(100px, 250px) 100px;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
```

JS

Track sizing playground

codepen.io/
websanity/pen/oQLoBL

					iOS		
`fr`	10 `-ms-`	16	52	10.1	10.3	57	57
`max-content`	10 `-ms-`	16	52	10.1	10.3	57	57
`min-content`	10 `-ms-`	16	52	10.1	10.3	57	57
`fit-content()`	10 `-ms-`	16	51	10.1	10.3	29	57
`minmax()`	—	12	52	10.1	10.3	57	80

`repeat()` Function

`repeat(x, y)`

CSS function for defining *repeated tracks in a grid*

Value for `grid-template-columns` & `grid-template-rows` only

`repeat(x, y)`

`x` is how many times to repeat `y`:

» `<positive-integer>`
» `auto-fill`
» `auto-fit`

`y` is a `<track-list>`; for example:

» `1fr`
» `min-content 1fr`
» `100px 1fr 200px`

```
grid-template-columns: repeat(3, 100px);
```

`3`: Number of times to repeat
`100px`: `<track list>` to repeat

“Repeat `100px` 3 times”

Equivalent to `grid-template-columns: 100px 100px 100px;`

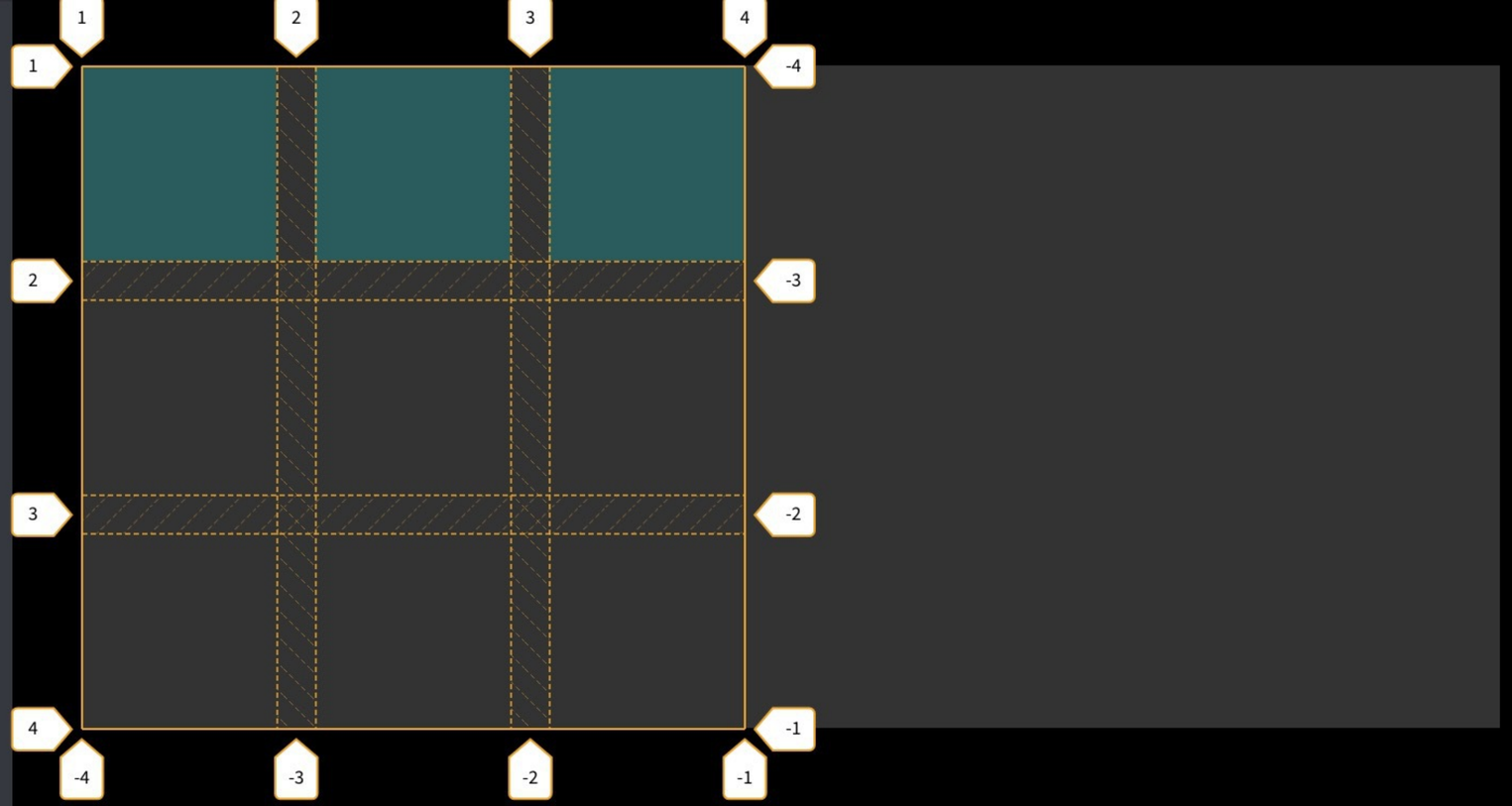
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: 100px 100px 100px;
}
```

JS



```
grid-template-columns: repeat(2, 50px 1fr);
```

`2`: Number of times to repeat
`50px 1fr`: `<track list>` to repeat

“Repeat `50px 1fr` 2 times”

Equivalent to `grid-template-columns: 50px 1fr 50px 1fr;`

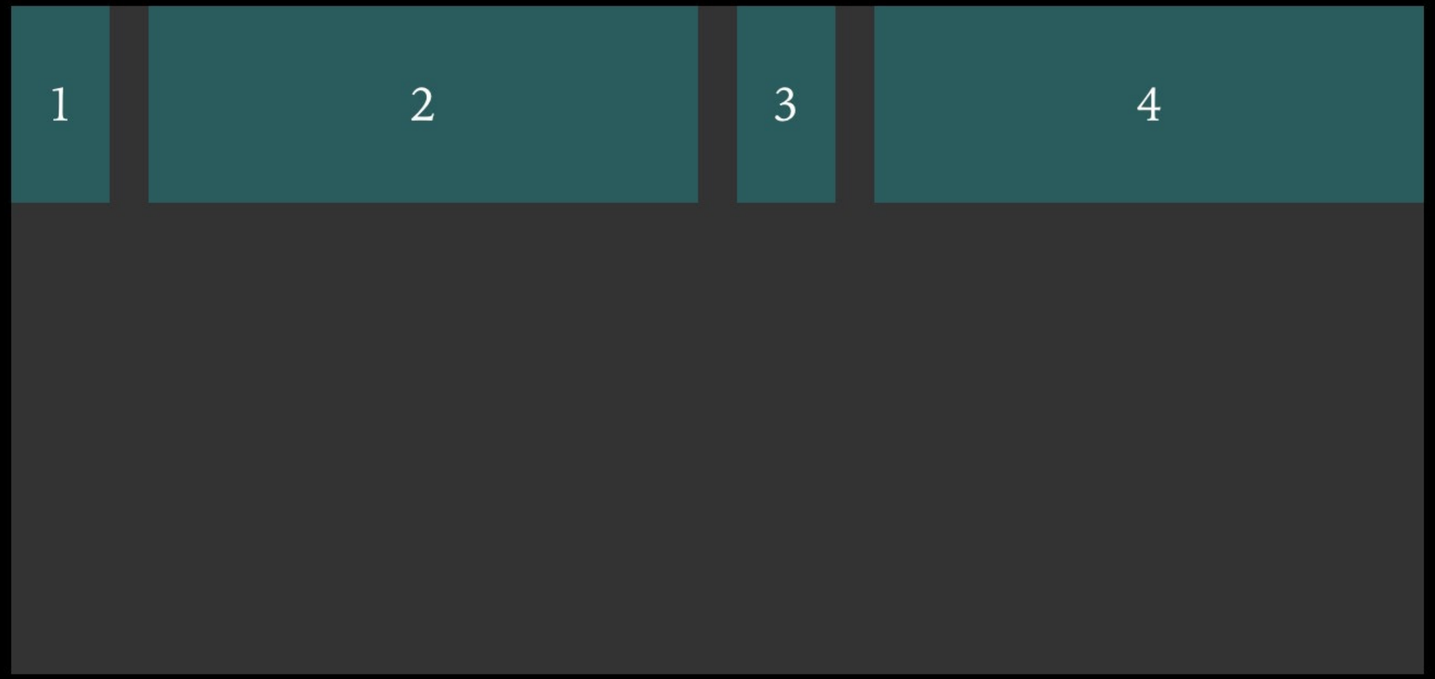
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(2, 50px 1fr);
  grid-template-rows: 100px 100px 100px;
}
```

JS



```
grid-template-columns: repeat(2, 50px 1fr)
100px;
```

`2`: Number of times to repeat
`50px 1fr`: `<track list>` to repeat

“Repeat `50px 1fr` 2 times, then insert `100px`”

Equivalent to `grid-template-columns: 50px 1fr 50px 1fr 100px;`

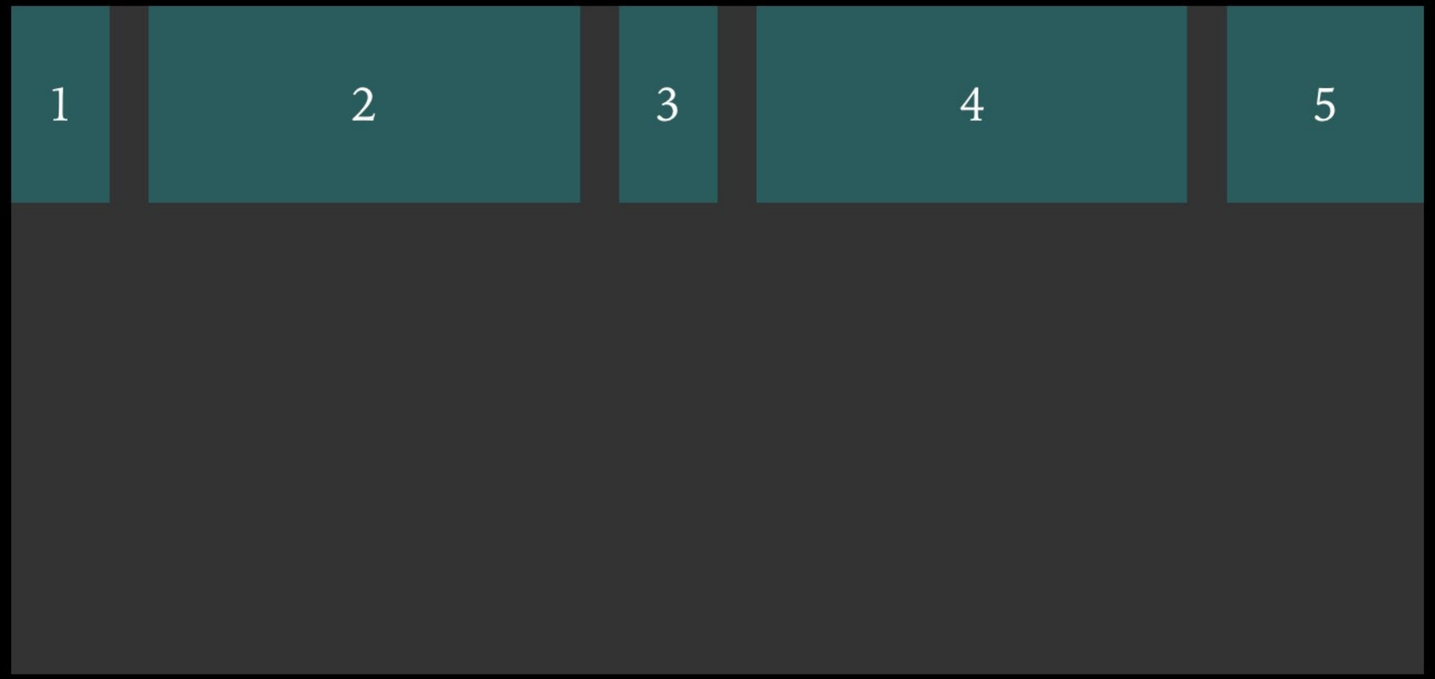
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(2, 50px 1fr) 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



```
grid-template-columns: 100px repeat(2, 1fr)
100px;
```

`2`: Number of times to repeat
`1fr`: `<track list>` to repeat

“Insert `100px`, then repeat `1fr` 2 times, then insert `100px`”

Equivalent to `grid-template-columns: 100px 1fr 1fr 100px;`

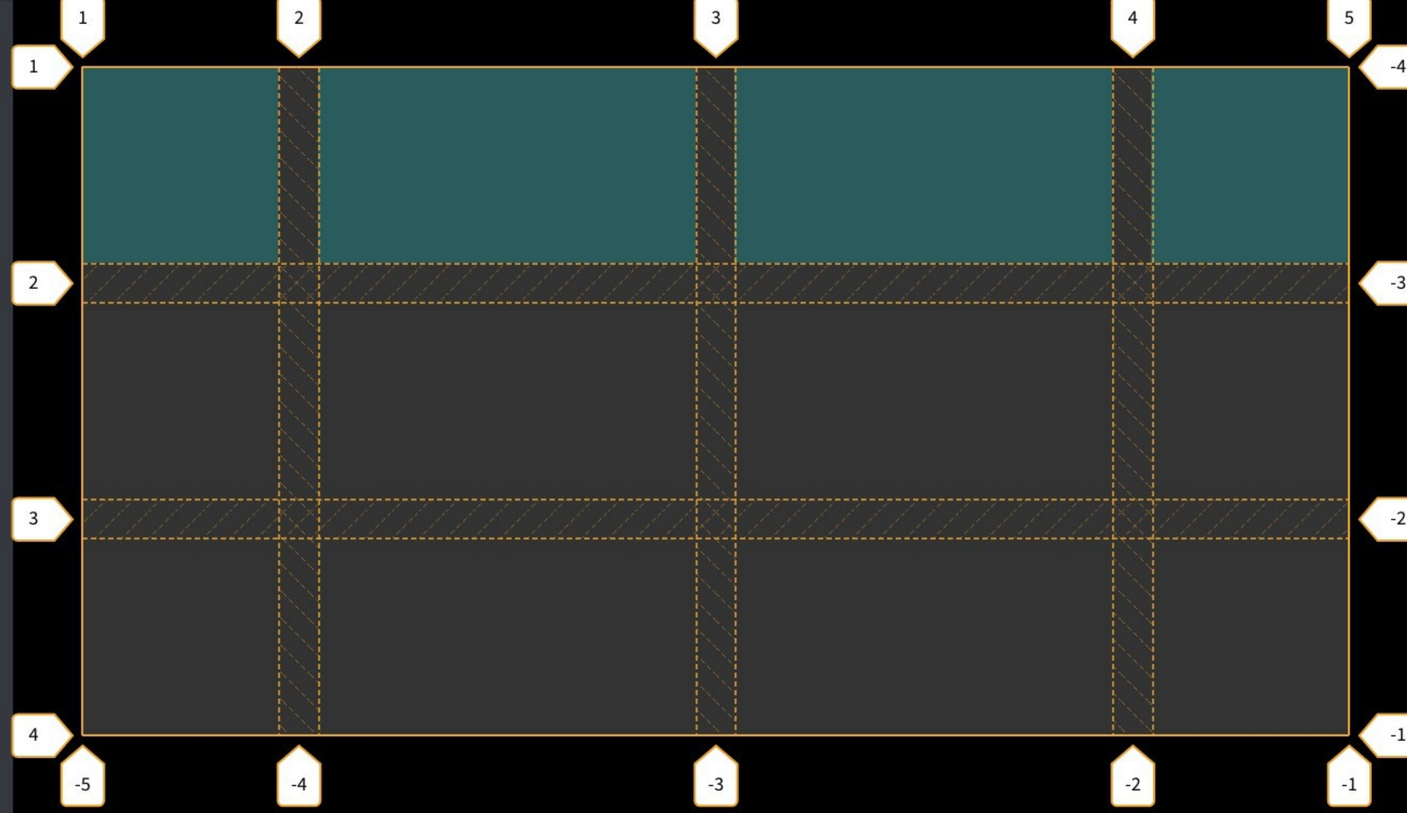
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px repeat(2, 1fr) 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



```
grid-template-columns: repeat(3, minmax(100px, 1fr));
```

`3`: Number of times to repeat

`minmax(100px, 1fr)`: `<track list>` to repeat

"Insert a minimum size of `100px` & a max of `1fr`, then repeat 3 times"

Equivalent to `grid-template-columns:`

```
minmax(100px, 1fr) minmax(100px, 1fr) minmax(100px, 1fr);
```

HTML

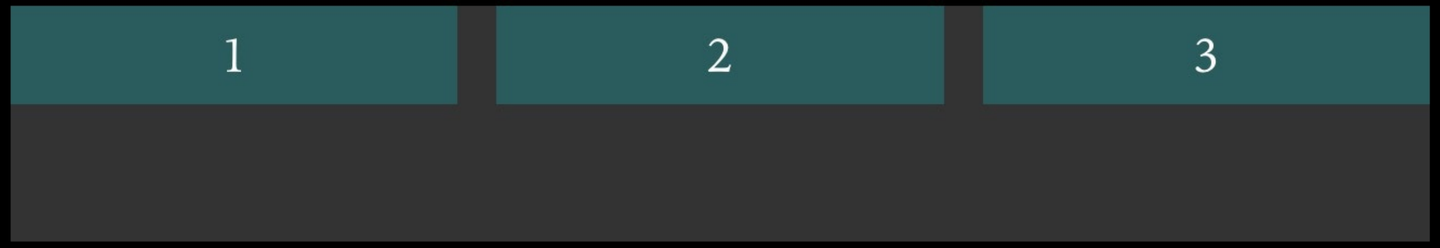
```
<div class="grid-container numeric">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.numeric {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

JS

numeric



`1fr` wide

HTML

```
<div class="grid-container numeric">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.numeric {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

JS

numeric

100px wide

HTML

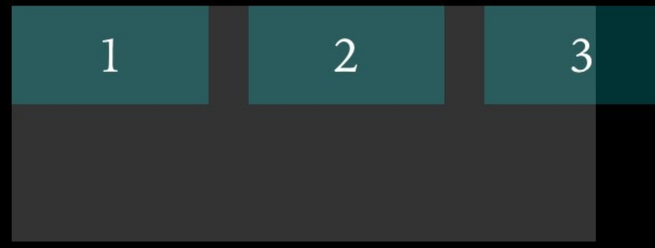
```
<div class="grid-container numeric">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.numeric {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

JS

numeric



100px wide 😖

`auto-fill`

Create new tracks to *fill the container when there is enough room*

`auto-fit`

Creates new tracks when there is enough room, but then *resizes tracks that have items so they fit the container*

HTML

```html
<h3>auto-fill</h3>
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>

<h3>auto-fit</h3>
<div class="grid-container fit">
  <div></div>
```

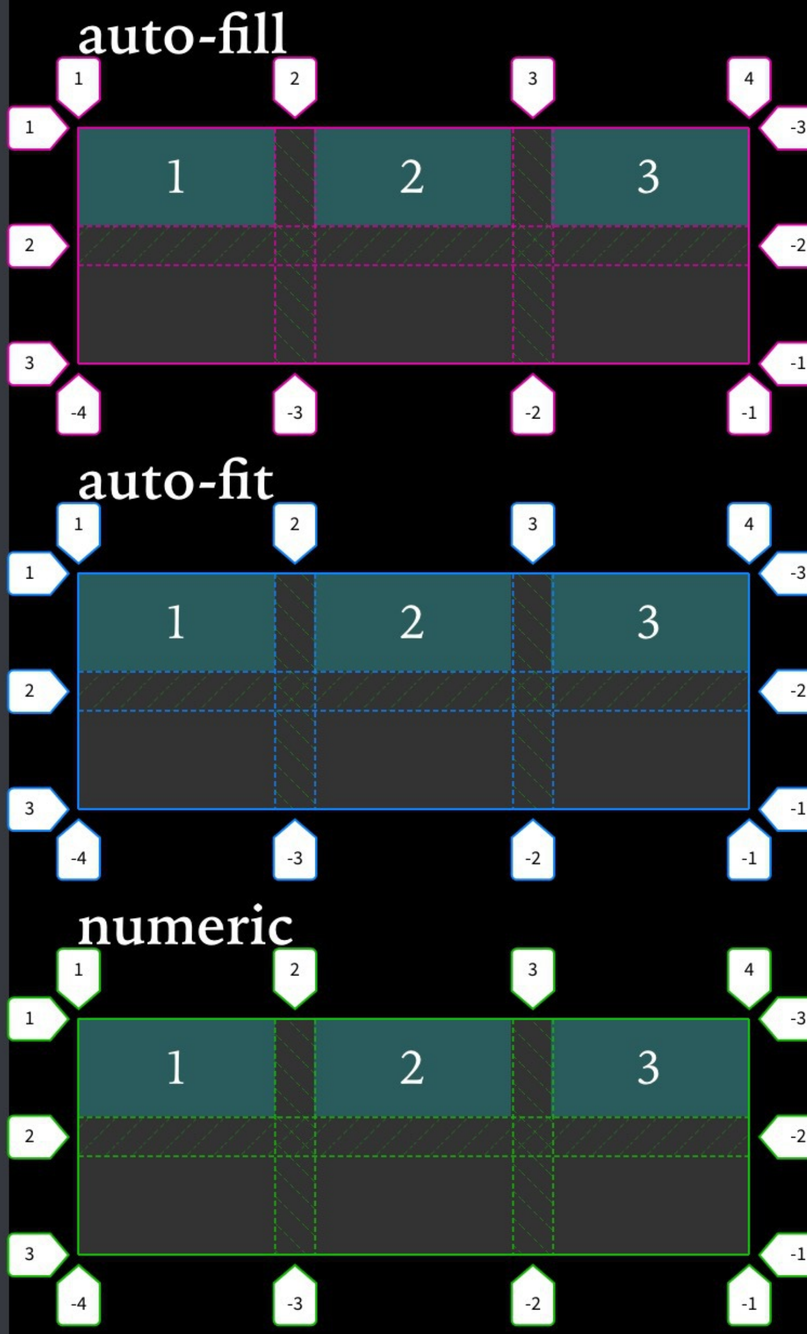
All columns are `100px`

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
}
.grid-container.fit {
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
.grid-container.numeric {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

Why are there 3 columns?

JS



HTML

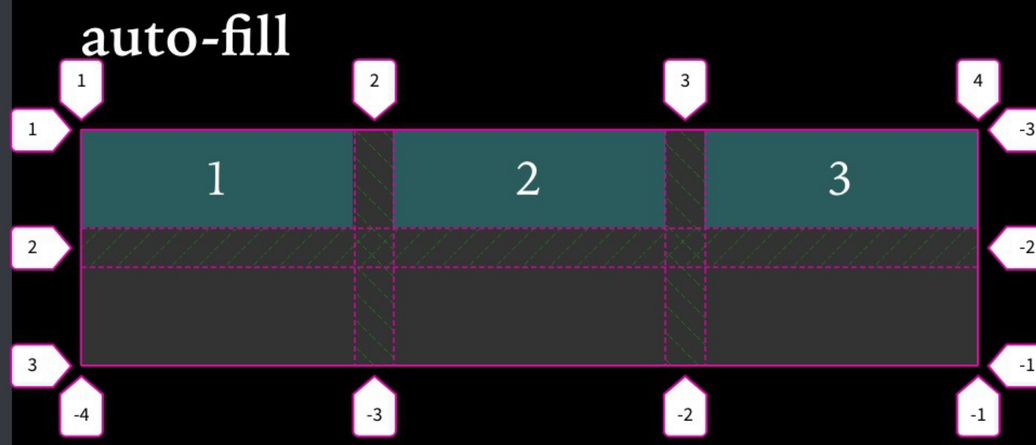
```html
<h3>auto-fill</h3>
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>

<h3>auto-fit</h3>
<div class="grid-container fit">
  <div></div>
```

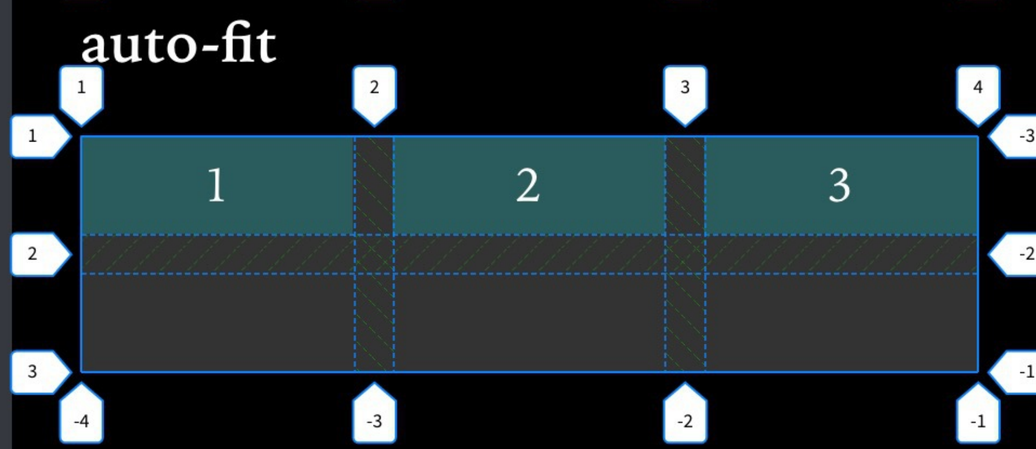
CSS Compiled

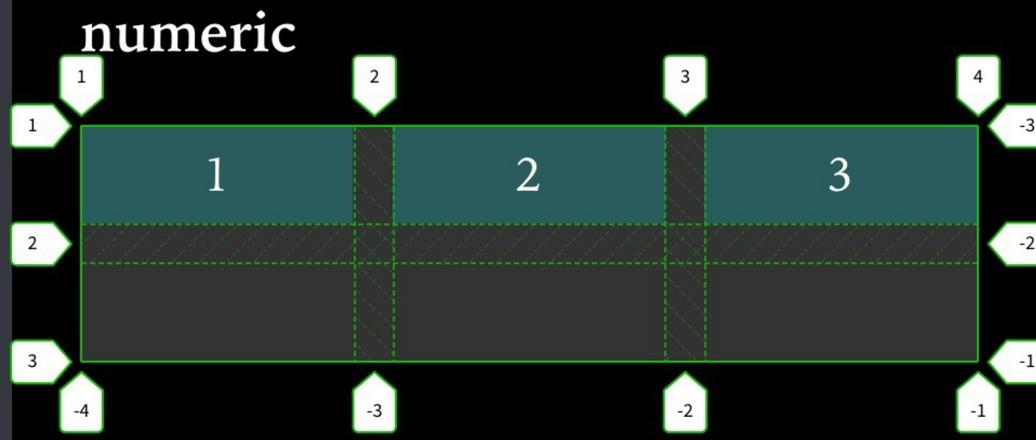
```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
}
.grid-container.fit {
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
.grid-container.numeric {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

JS

All columns are `1fr`







HTML

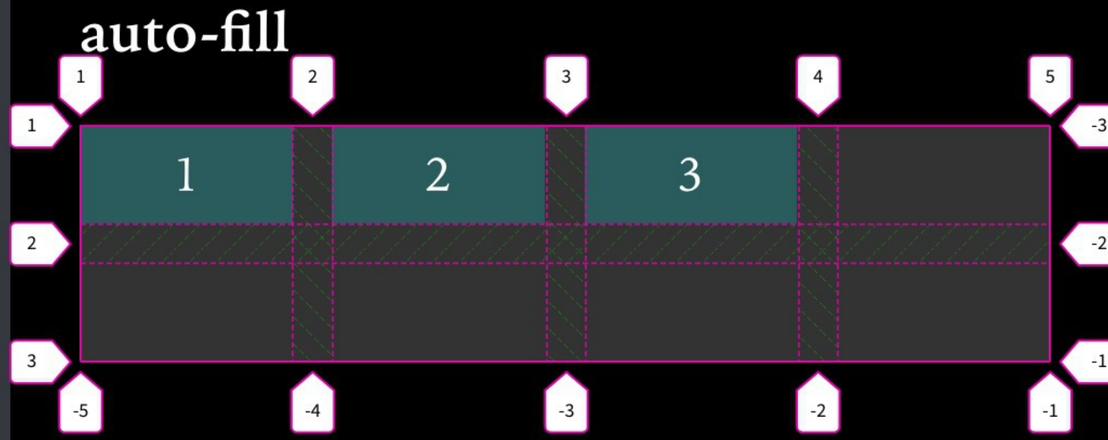
```html
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>

<h3>auto-fit</h3>
<div class="grid-container fit">
  <div></div>
  <div></div>
```

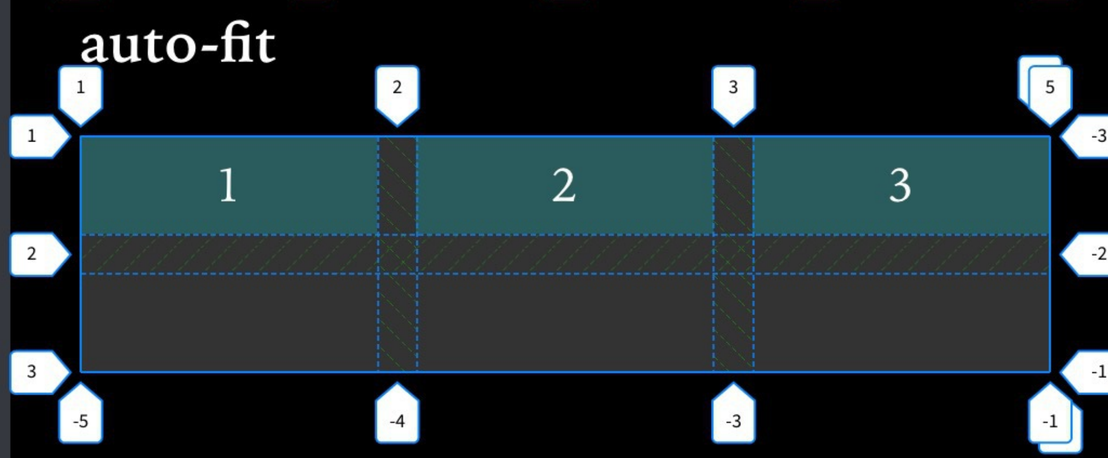
Container has `100px` of space, so `auto-fill` creates a new track

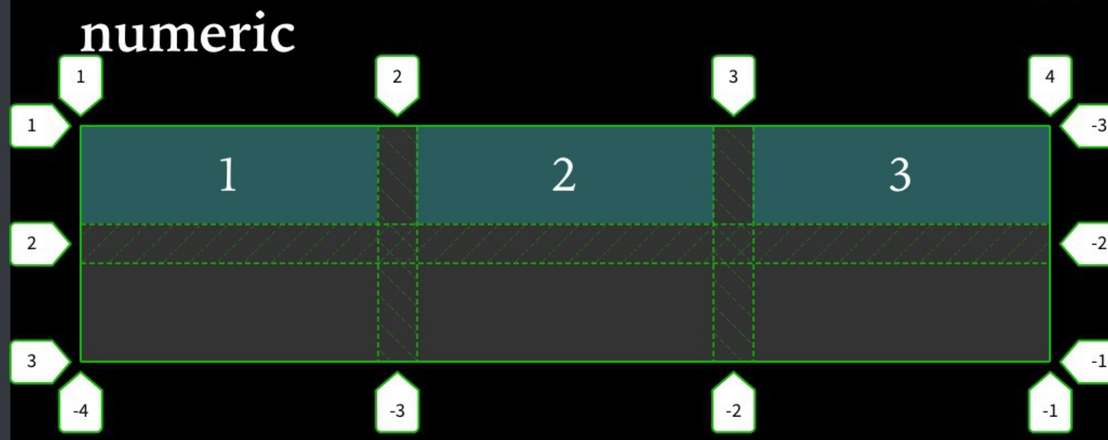
CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
}
.grid-container.fit {
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
.grid-container.numeric {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

JS







HTML

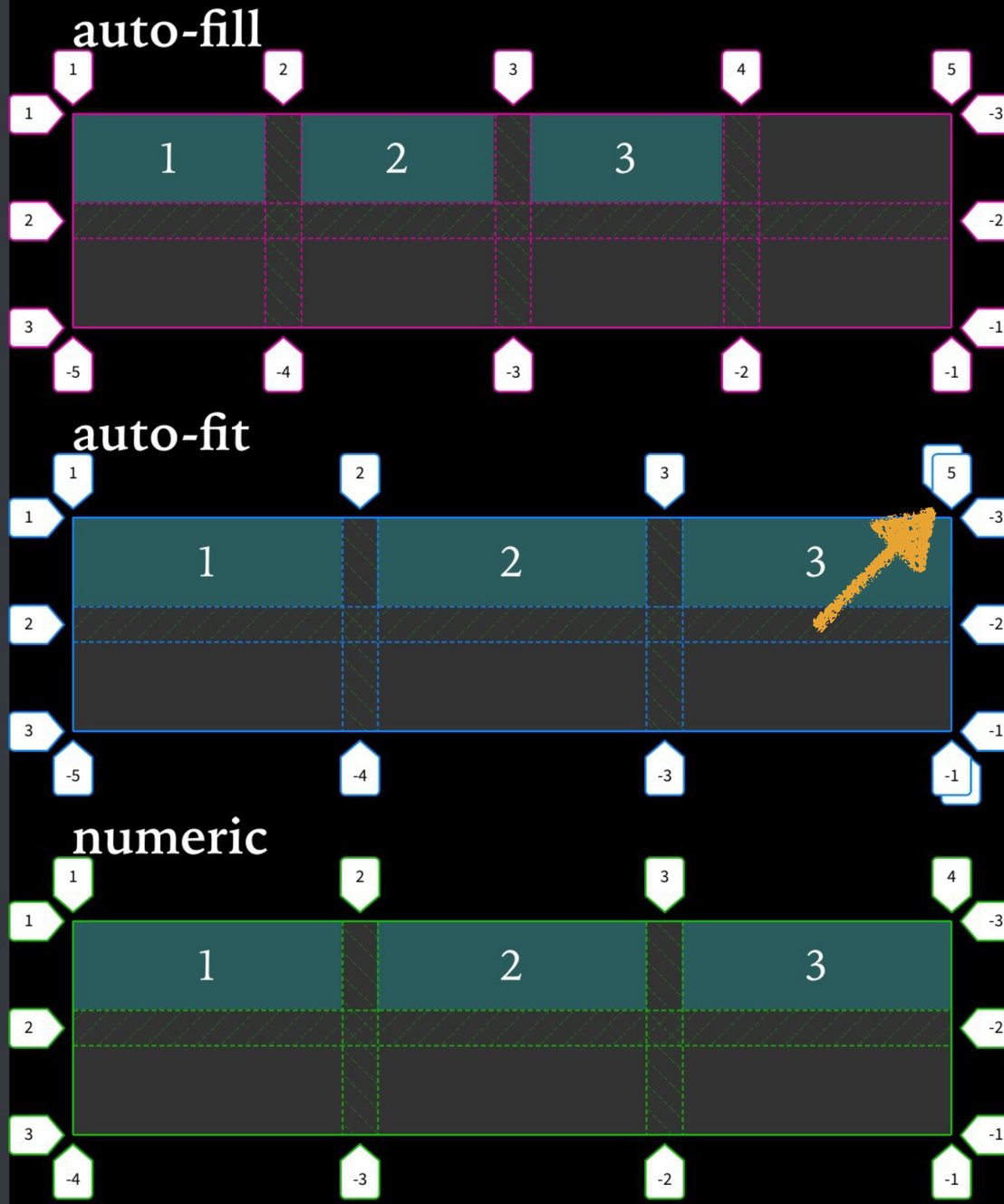
```
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>

<h3>auto-fit</h3>
<div class="grid-container fit">
  <div></div>
  <div></div>
```

CSS Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
}
.grid-container.fit {
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
.grid-container.numeric {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

JS

`auto-fit` created a new track also (see the 5?) but only the 3 tracks with items are resized



HTML

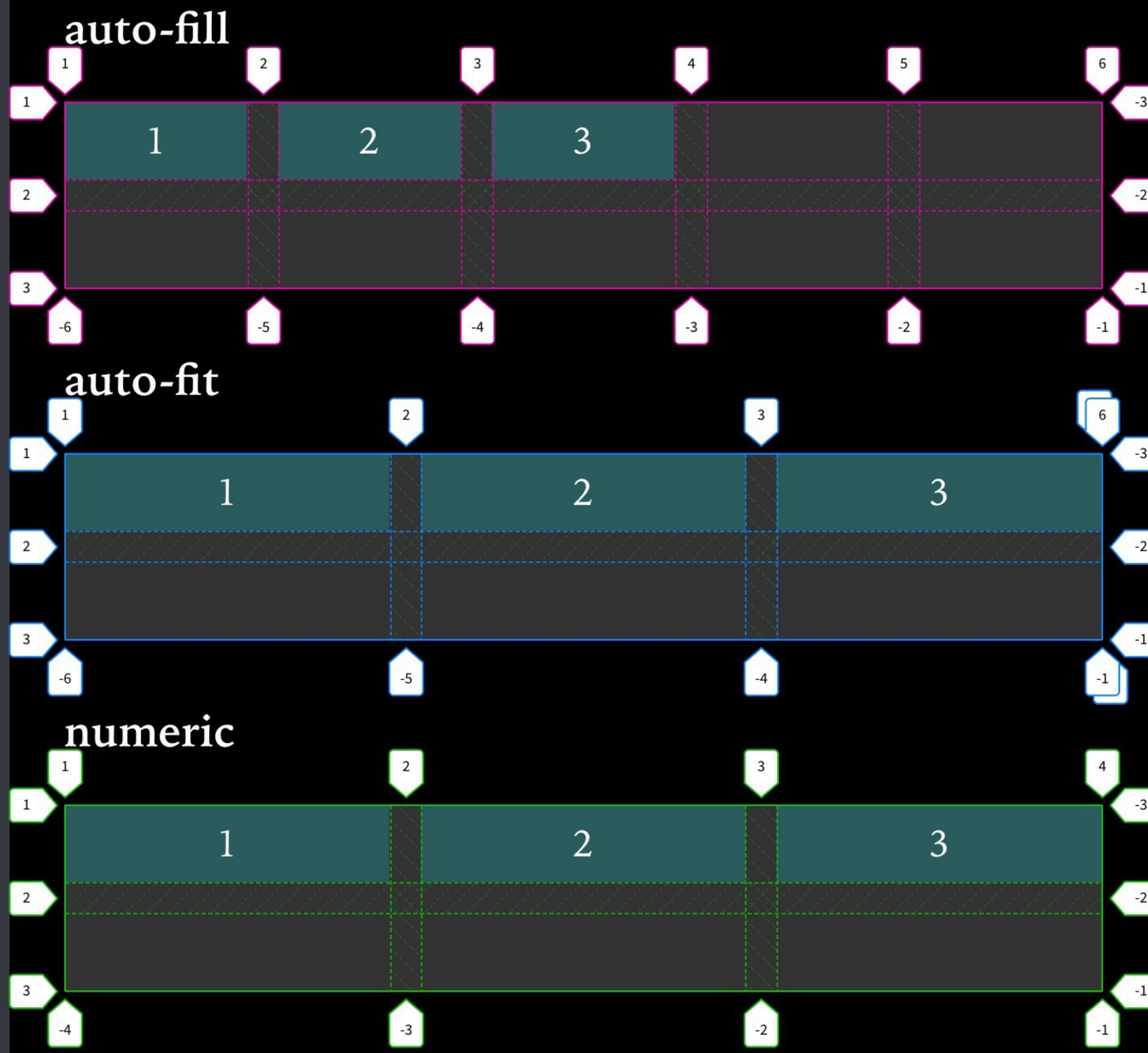
```
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>

<h3>auto-fit</h3>
<div class="grid-container fit">
  <div></div>
  <div></div>
```

auto-fill creates 2 extra tracks

CSS Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns: repeat(auto-fill,
minmax(100px, 1fr));
}
.grid-container.fit {
  grid-template-columns: repeat(auto-fit,
minmax(100px, 1fr));
}
.grid-container.numeric {
```

JS



HTML

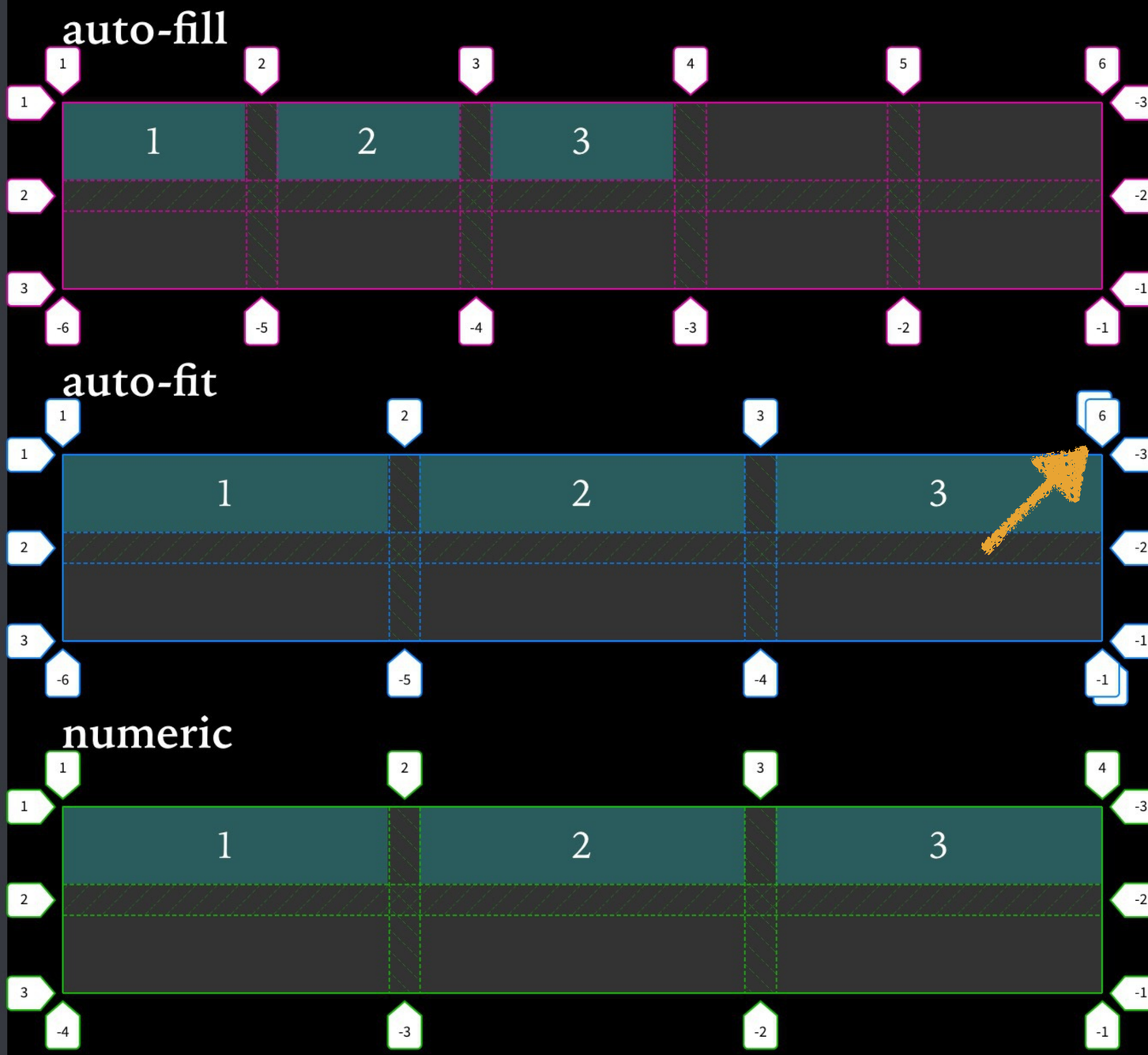
```
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>
<h3>auto-fit</h3>
<div class="grid-container fit">
  <div></div>
  <div></div>
  <div></div>
</div>
<h3>numeric</h3>
<div class="grid-container numeric">
  <div></div>
  <div></div>
  <div></div>
</div>
```

`auto-fit` created a new track also (see the 6?) but only the 3 tracks with items are resized

CSS Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
}
.grid-container.fit {
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
.grid-container.numeric {
```

JS



auto-fill creates 4 extra tracks

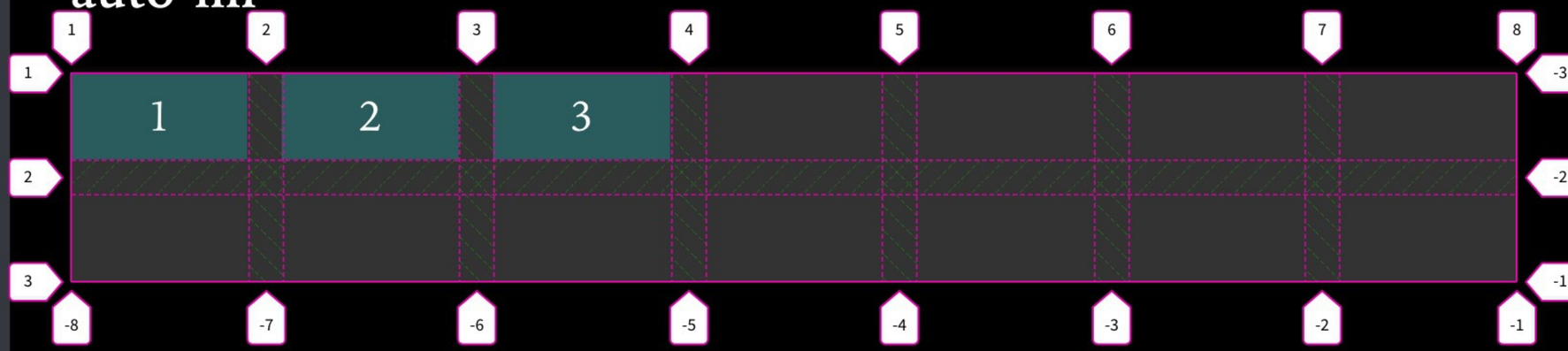
HTML

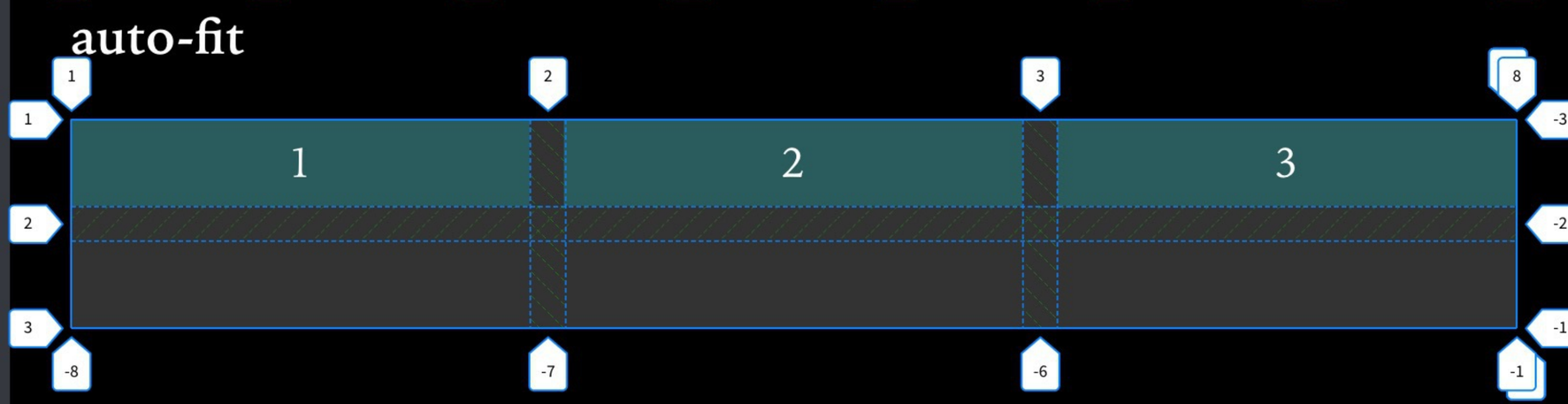
```
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>

<h3>auto-fit</h3>
<div class="grid-container fit">
  <div></div>
  <div></div>
```

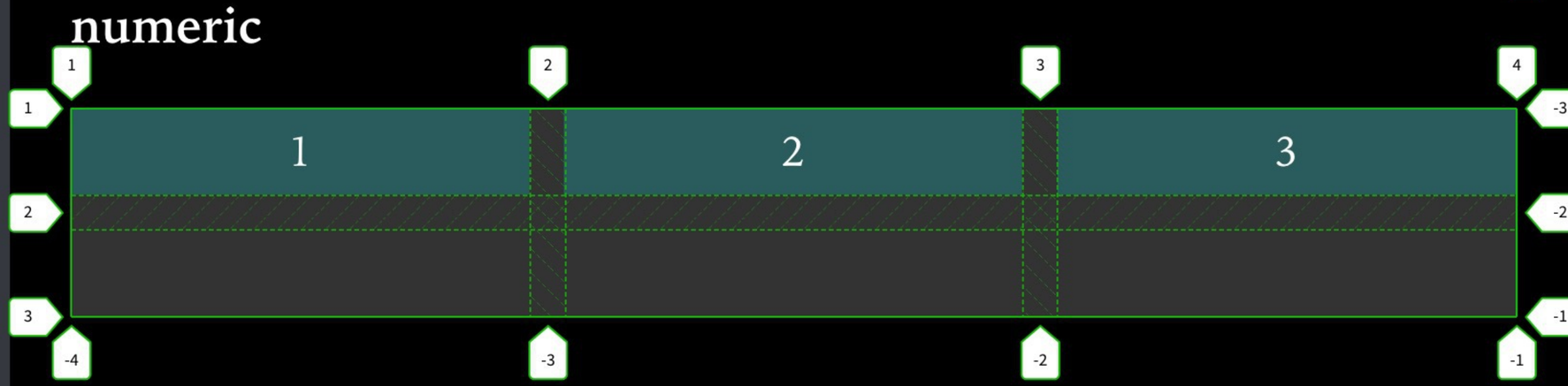
CSS Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns:
repeat(auto-fill, minmax(100px,
1fr));
}
.grid-container.fit {
  grid-template-columns:
repeat(auto-fit, minmax(100px,
1fr));
```

JS







HTML

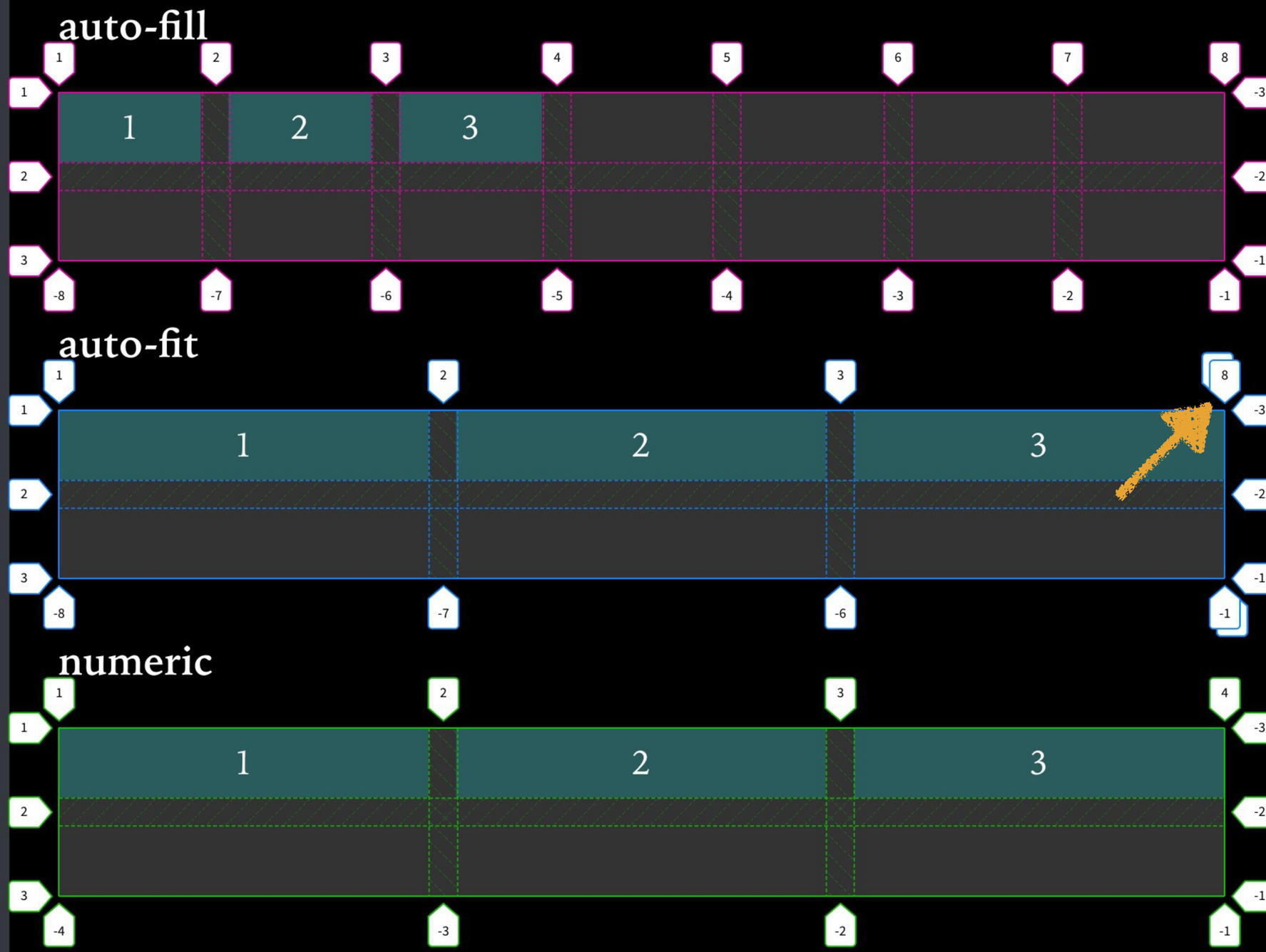
auto-fit created a new track also (see the 8?) but only the 3 tracks with items are resized

```
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>
<div class="grid-container fit">
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns:
repeat(auto-fill, minmax(100px,
1fr));
}
.grid-container.fit {
  grid-template-columns:
repeat(auto-fit, minmax(100px,
1fr));
}
```

JS



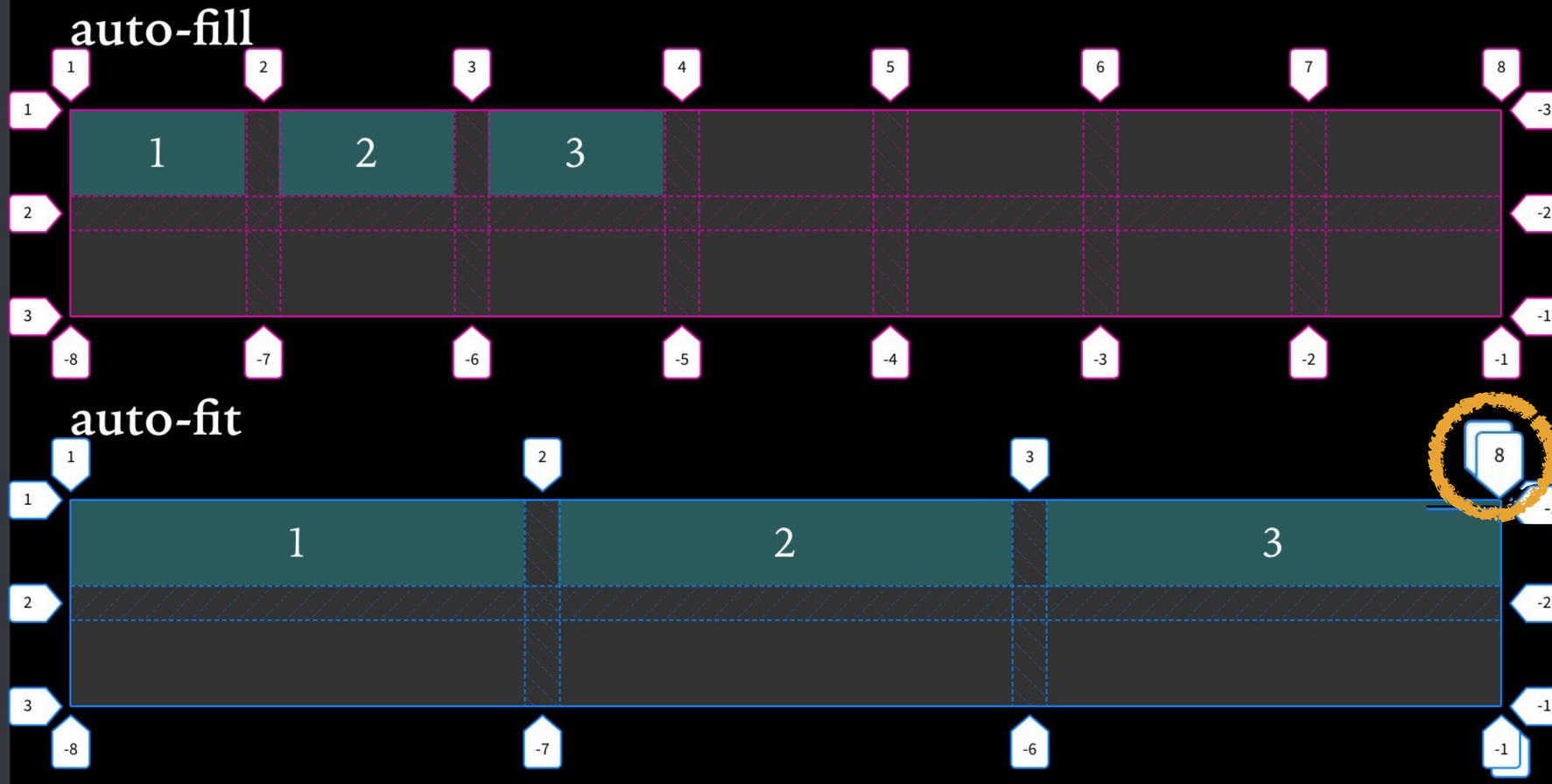
HTML

```html
<h3>auto-fill</h3>
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>

<h3>auto-fit</h3>
<div class="grid-container fit">
  <div></div>
```

CSS Compiled

```css
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns:
repeat(auto-fill, minmax(100px,
1fr));
}
.grid-container.fit {
  grid-template-columns:
repeat(auto-fit, minmax(100px,
1fr));
```

JS



Why 8 lines for `auto-fit`? New tracks are created, but they don't have items in them, so they take up no space & do not affect the layout

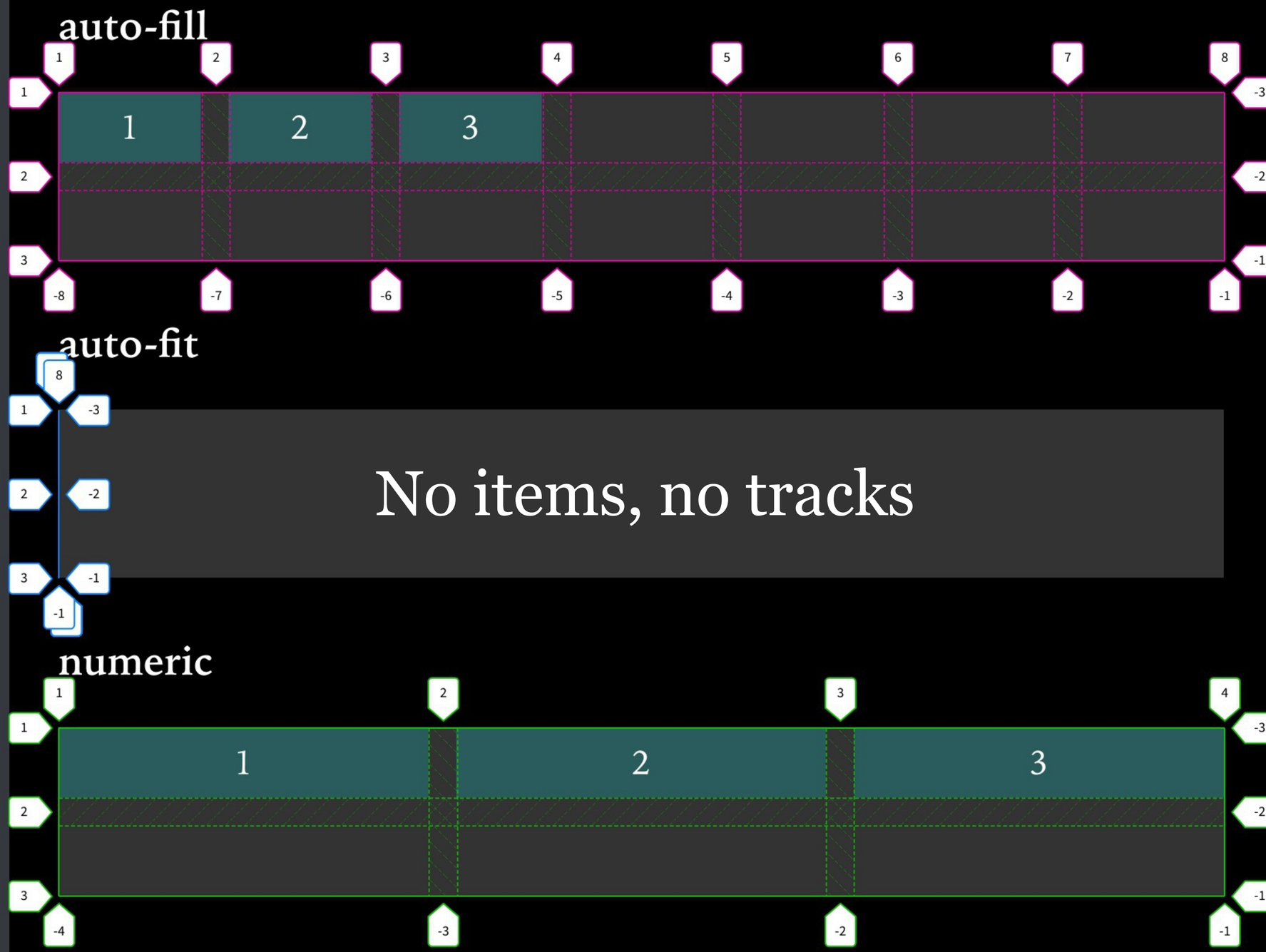
HTML

```
<h3>auto-fit</h3>
<div class="grid-container fit">

</div>

<h3>numeric</h3>
<div class="grid-container
numeric">
  <div></div>
  <div></div>
```

CSS Compiled

```
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns:
repeat(auto-fill, minmax(100px,
1fr));
}
.grid-container.fit {
  grid-template-columns:
repeat(auto-fit, minmax(100px,
1fr));
}
```

JS



HTML

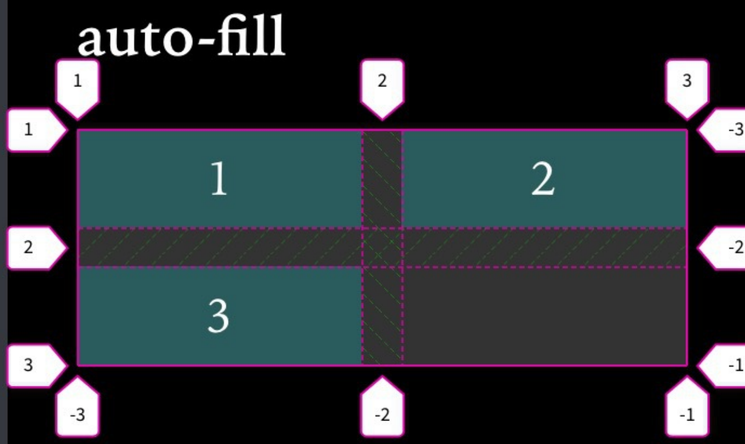
```
<h3>auto-fill</h3>
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>

<h3>auto-fit</h3>
<div class="grid-container fit">
  <div></div>
```

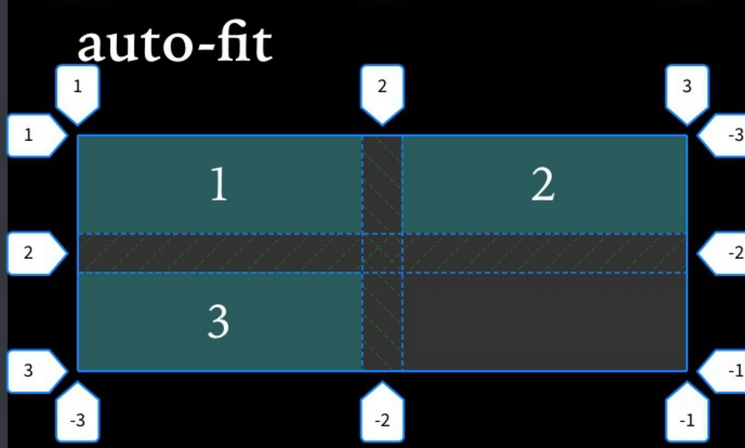
CSS Compiled

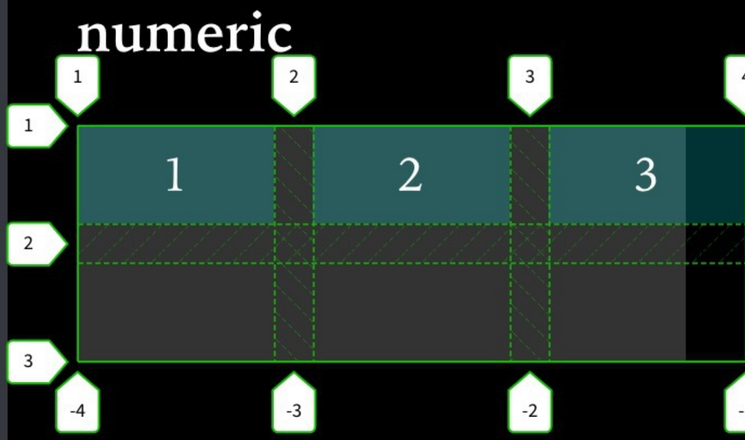
```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
}
.grid-container.fit {
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
.grid-container.numeric {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

JS

`auto-fill` & `auto-fit` only create 2 column tracks, but there's not enough room with numeric for its 3 column tracks, so it blows out of the container







HTML

```
<h3>auto-fill</h3>
<div class="grid-container fill">
  <div></div>
  <div></div>
  <div></div>
</div>

<h3>auto-fit</h3>
<div class="grid-container fit">
  <div></div>
```

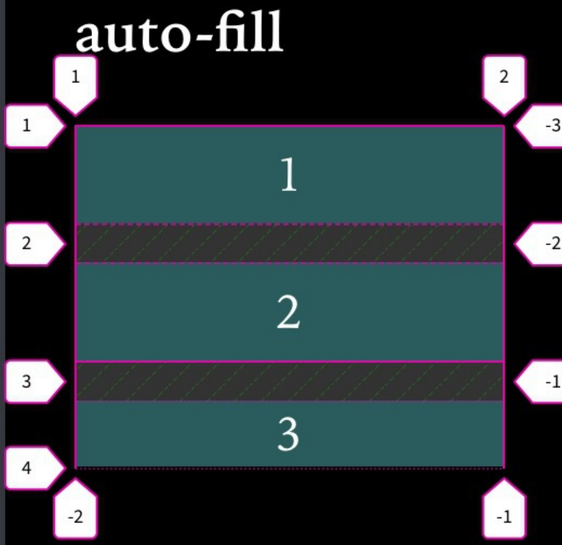
Not yet large enough to create 2 ≥ `100px` tracks… & numeric continues to blow out of the container

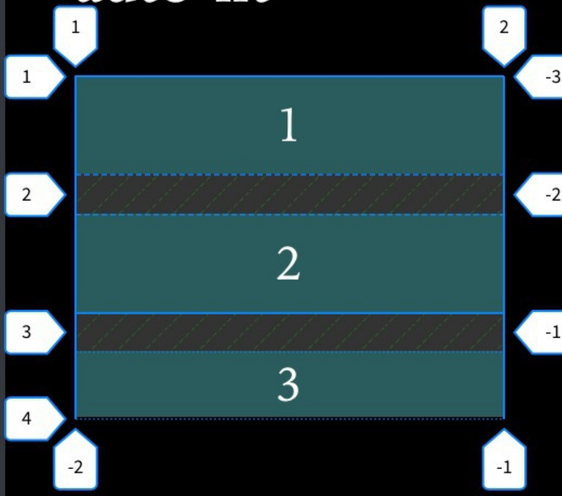
CSS Compiled

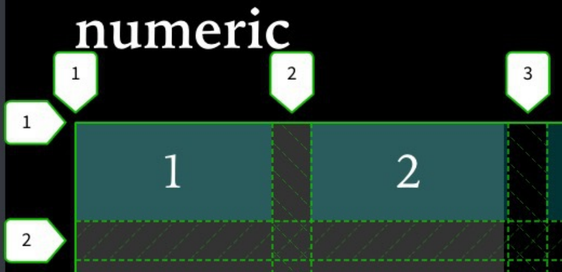
```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-rows: 50px 50px;
}
.grid-container.fill {
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
}
.grid-container.fit {
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
.grid-container.numeric {
  grid-template-columns: repeat(3, minmax(100px, 1fr));
}
```

Since the `grid-gap` is `20px`, we need `220`–`339px` to create 2 tracks

JS







					iOS		
`repeat()`	–	16	59*	10.1	10.3	57	80

* `repeat(auto-fill, ...)` & `repeat(auto-fit, ...)` still only support 1 repeated column as of version 72 (March 2020)

Aligning Within the Grid

Gutters

```
row-gap
column-gap

gap
```

December 14, 2017 W3C Candidate Recommendation for CSS Grid Layout Module Level 1 announced this change:

“Removed `grid-row-gap`, `grid-column-gap`, and `grid-gap` properties, replacing with `row-gap`, `column-gap`, and `gap` which are now defined in CSS Box Alignment. (Issue 1696)”

All `gap` properties have to do with setting the minimum amount of space between rows & columns

`row-gap`

Defines *minimum size of grid gutter between rows*

`column-gap`

Defines *minimum size of grid gutter between columns*

`gap`

Defines *minimum size of grid gutter between rows & columns*

Shorthand for setting `row-gap` & `column-gap`

Values for `row-gap`, `column-gap`, & `gap`

- `<length>`
- `<percentage>`: never use this, or you shall sink into a Stygian madness

All `gap` properties can accept 1 or 2 values, e.g.:

`gap: 1em`
Sets value for *both* `row-gap` & `column-gap`

`gap: .5em 1em`
Sets value for `row-gap` & *then* `column-gap`

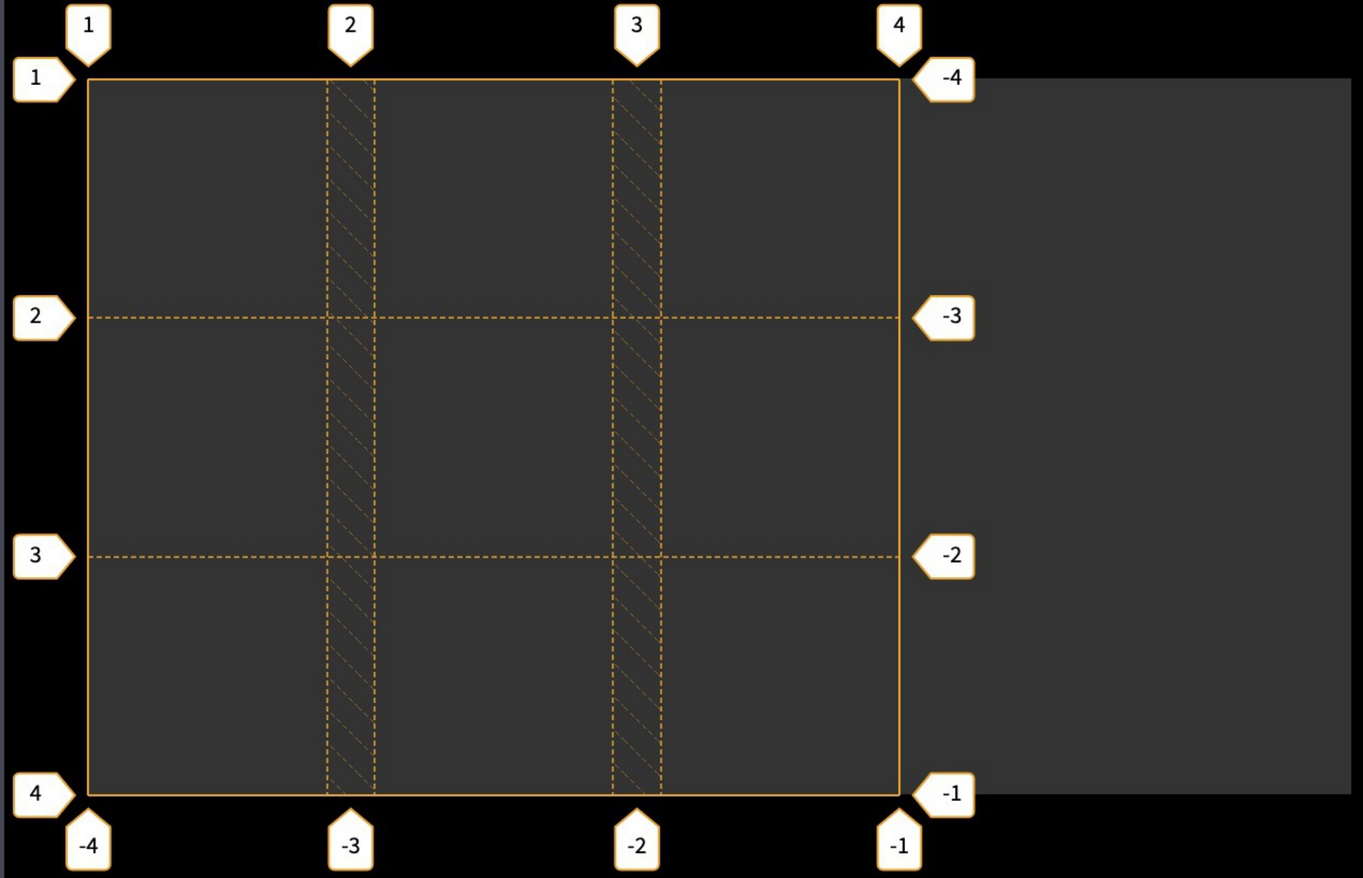
HTML

```html
<div class="grid-container">

</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  column-gap: 20px;
}
```

JS



No value set for rows, so they default to 0

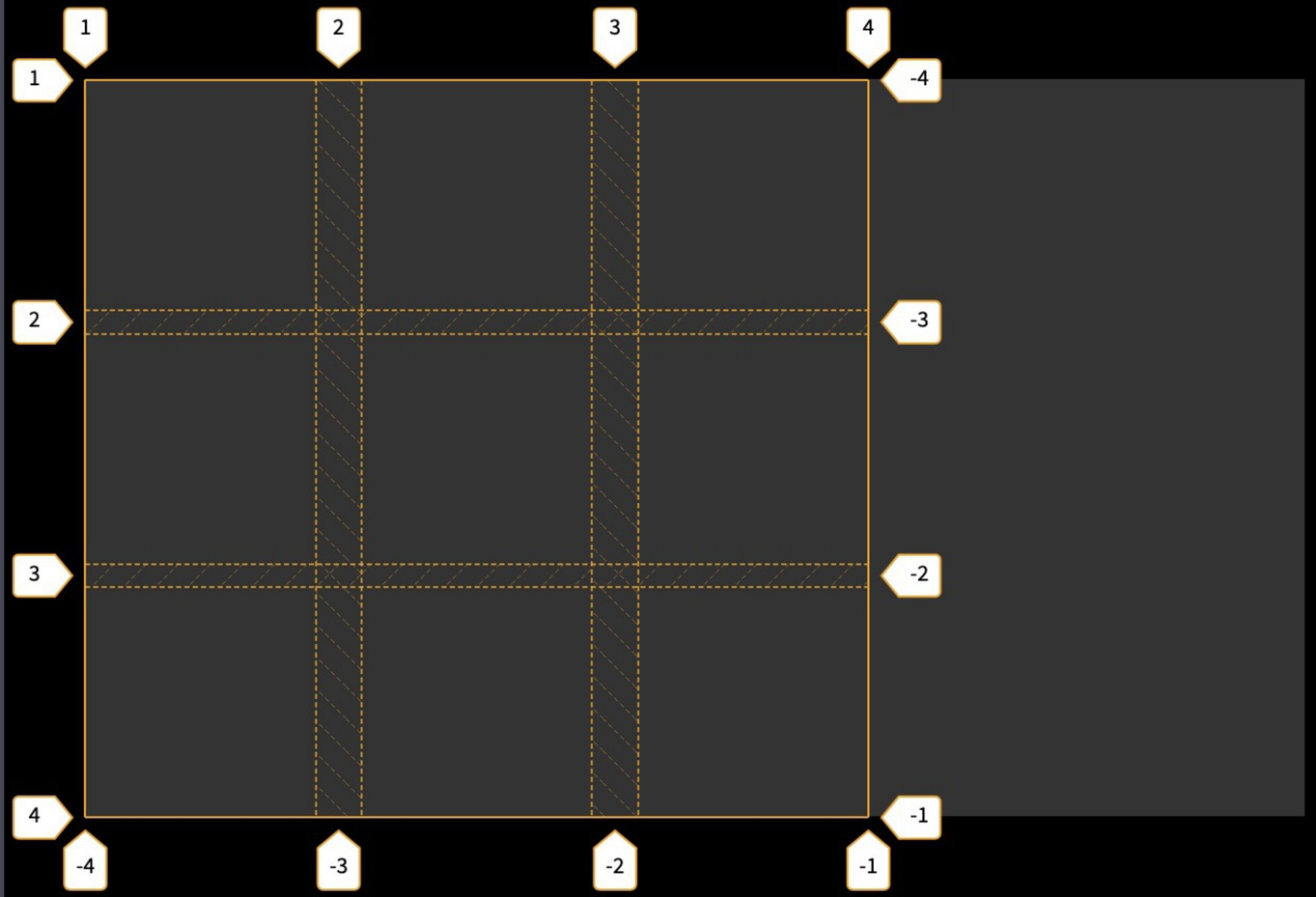
HTML

```html
<div class="grid-container">

</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  row-gap: 10px;
  column-gap: 20px;
}
```

JS



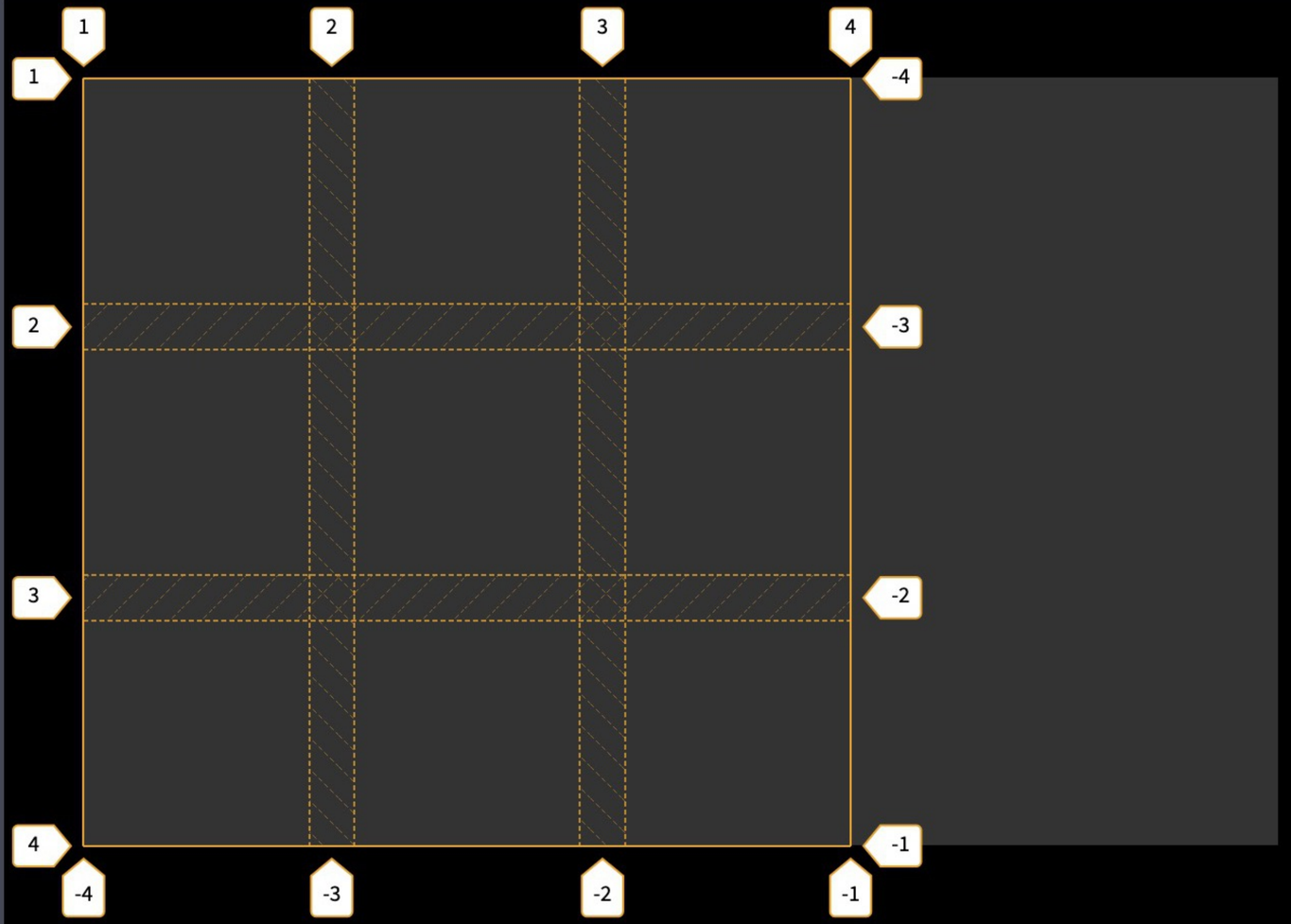
HTML

```html
<div class="grid-container">

</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  gap: 20px;
}
```

JS



`gap: 20px` sets value for *both* `row-gap` & `column-gap`

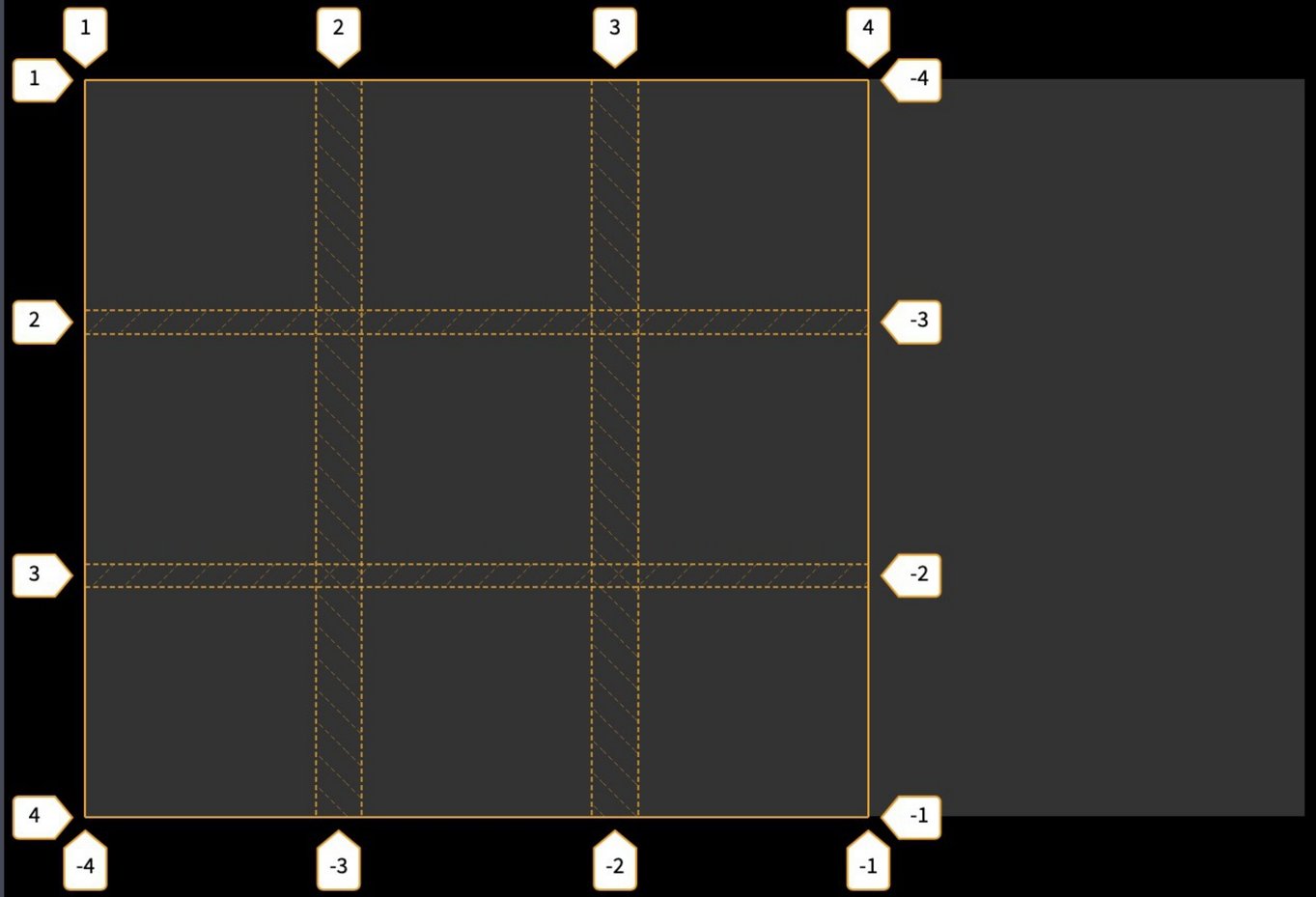
HTML

```html
<div class="grid-container">

</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  gap: 10px 20px;
}
```

JS



`gap: 10px 20px` sets value for `row-gap` & then `column-gap`

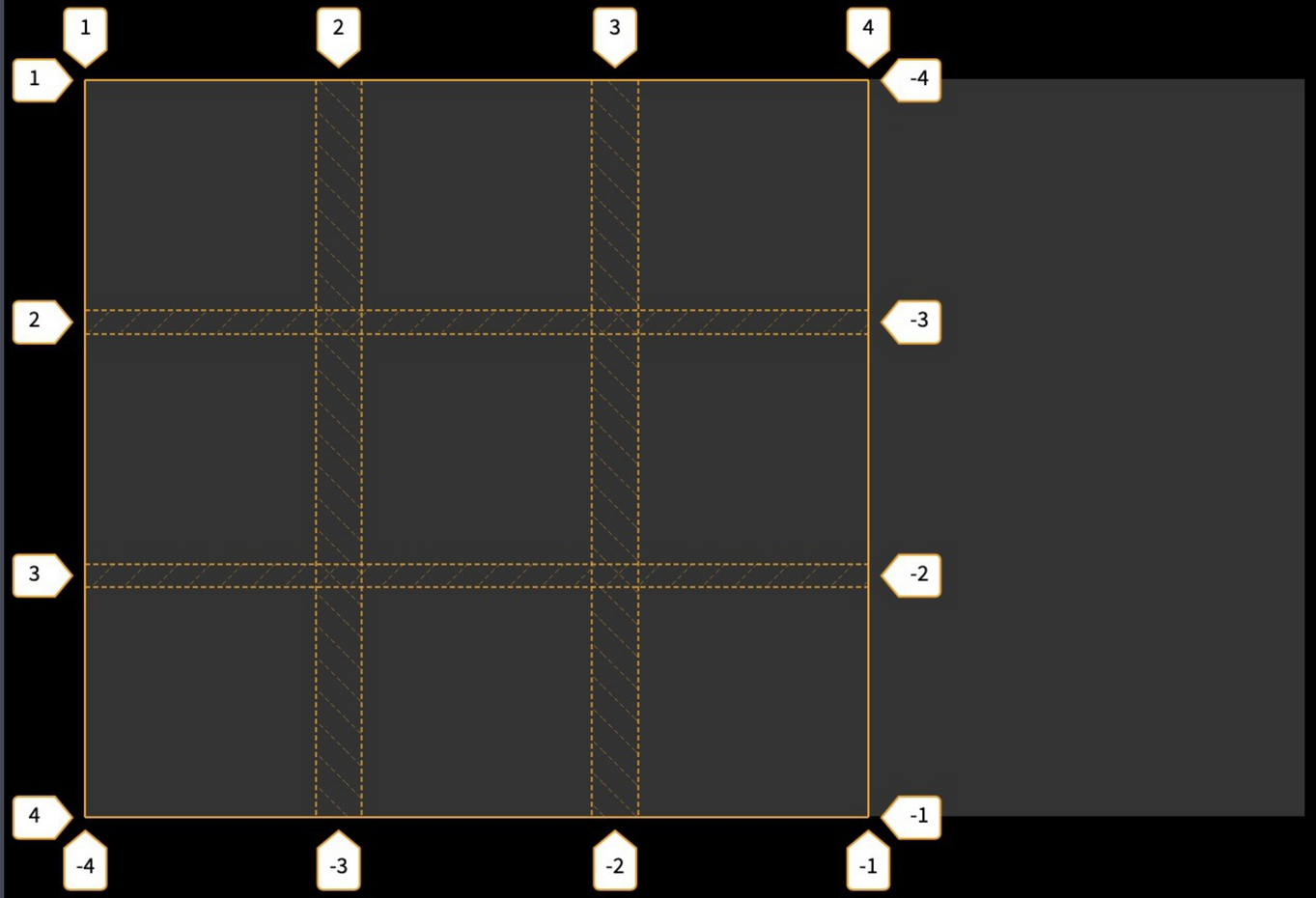
HTML

```html
<div class="grid-container">

</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  gap: 10px 20px;
}
```

JS



All `gap` properties set the size of the grid's gutter between tracks only, *not* between the container & the items

💡 PRO TIP

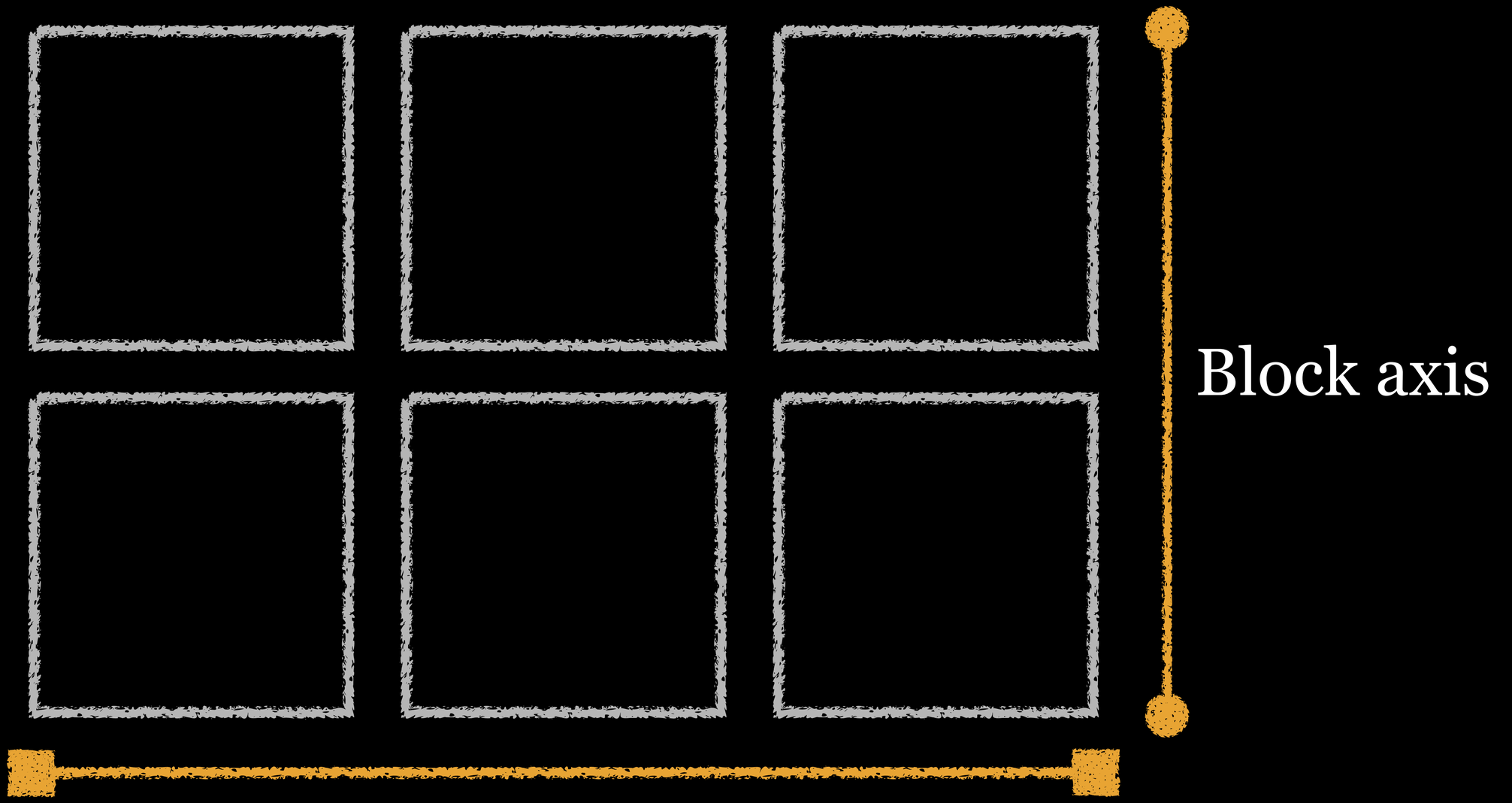
All `gap` properties set the minimum distance between tracks, however, `justify-content` & `align-content` (covered next) can increase the distance

					iOS		
grid-row-gap	–	16	52	10.1	10.3	57	57
row-gap	–	16	61	12	12	66	66
grid-column-gap	–	16	52	10.1	10.3	57	57
column-gap	–	16	61	12	12	66	66
grid-gap	–	16	57	10.1	10.3	57	57
gap	–	16	66	12	12	66	66

Aligning Grid Tracks

```
justify-content
align-content

place-content
```

Grid lines



All of the `*-content` properties in this section align tracks & gaps in relation to the grid container

Therefore, the container must be larger than the grid for these to take effect

`gap` properties set minimum amount of space between tracks

The `*-content` properties align tracks within any free space by specifying how the free space is used

This may result in increasing the gutter size set by `gap`

`justify-content`

Sets *alignment of grid tracks & gaps along the inline axis* relative to the grid container; i.e., putting space around columns

This will likely get used more than `align-content` (which is for the block axis, & which usually has a fixed height), because it's more likely you'll have viewports that are wider than the content, giving you free space

Values for `justify-content`

- `normal`/`stretch`: resizes grid items so grid *fills full width* of container (default)
- `start`: aligns grid flush with *start edge* of container
- `end`: aligns grid flush with *end edge* of container
- `center`: aligns grid in *center* of container

`normal`
`stretch`

Resizes tracks so *grid fills full width of container*

`normal` is the default, but it is equivalent to `stretch` in grid layout

`stretch` only affects tracks that are sized `auto`

If a track's size is not `auto`, then `normal` & `stretch` both behave like `start`

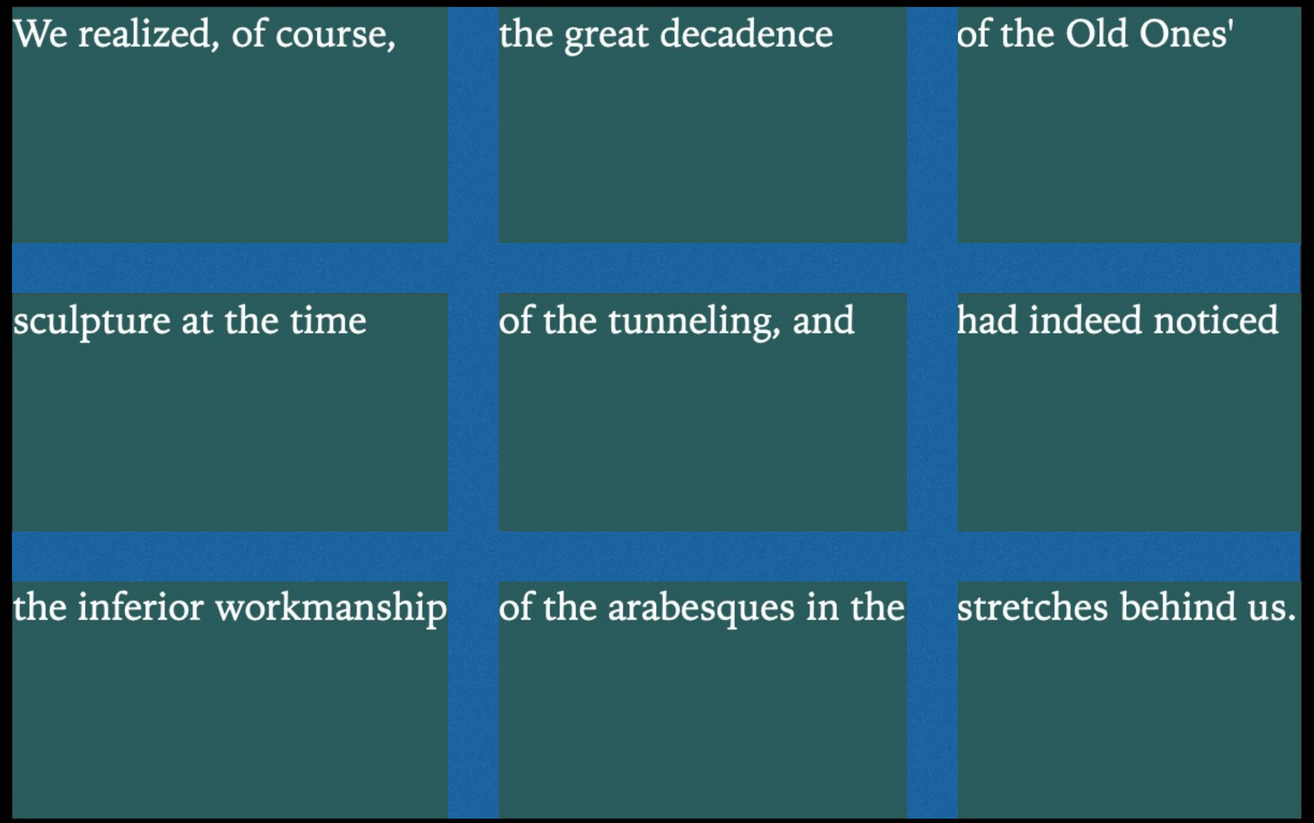
HTML

```html
<div class="grid-container">
  <div>We realized, of course,</div>
  <div>the great decadence</div>
  <div>of the Old Ones'</div>
  <div>sculpture at the time</div>
  <div>of the tunneling, and</div>
  <div>had indeed noticed</div>
  <div>the inferior workmanship</div>
  <div>of the arabesques in the</div>
  <div>stretches behind us.</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  gap: 20px;
  grid-template-columns: auto auto auto;
  grid-template-rows: repeat(3, 100px);
  /* stretch is default, but column size must be `auto` */
  justify-content: stretch;
}
```

JS



Gap

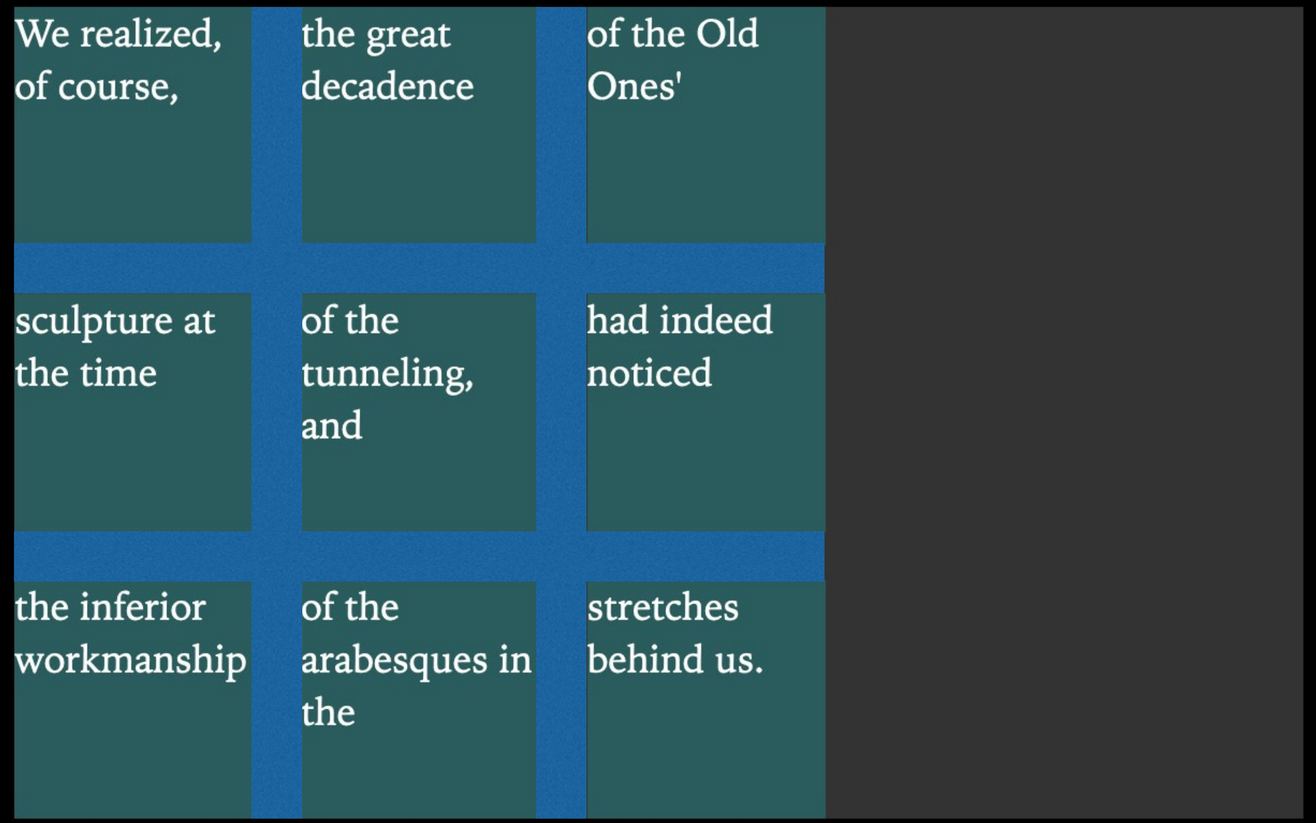
HTML

```html
<div class="grid-container">
  <div>We realized, of course,</div>
  <div>the great decadence</div>
  <div>of the Old Ones'</div>
  <div>sculpture at the time</div>
  <div>of the tunneling, and</div>
  <div>had indeed noticed</div>
  <div>the inferior workmanship</div>
  <div>of the arabesques in the</div>
  <div>stretches behind us.</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  justify-content: start;
}
```

JS



Gap

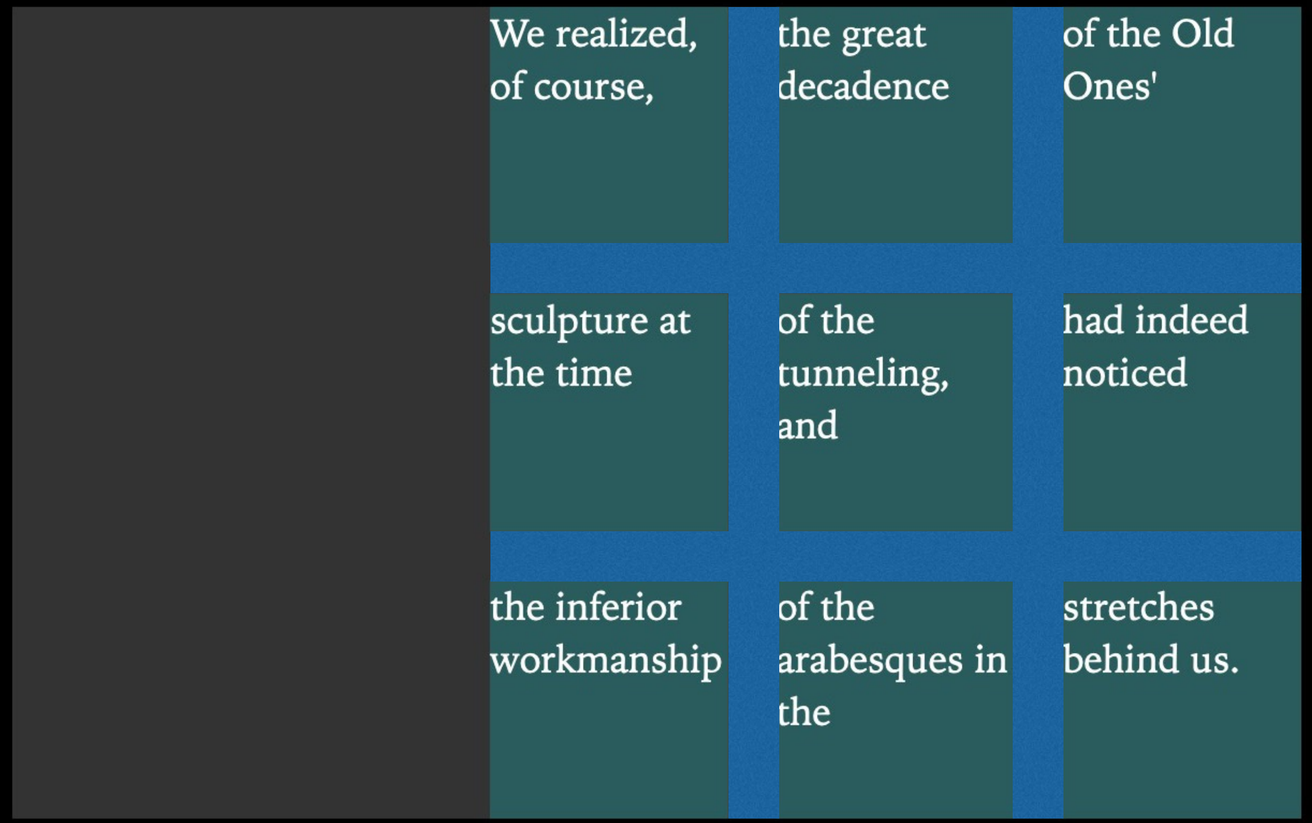
HTML

```html
<div class="grid-container">
  <div>We realized, of course,</div>
  <div>the great decadence</div>
  <div>of the Old Ones'</div>
  <div>sculpture at the time</div>
  <div>of the tunneling, and</div>
  <div>had indeed noticed</div>
  <div>the inferior workmanship</div>
  <div>of the arabesques in the</div>
  <div>stretches behind us.</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  justify-content: end;
}
```

JS



Gap

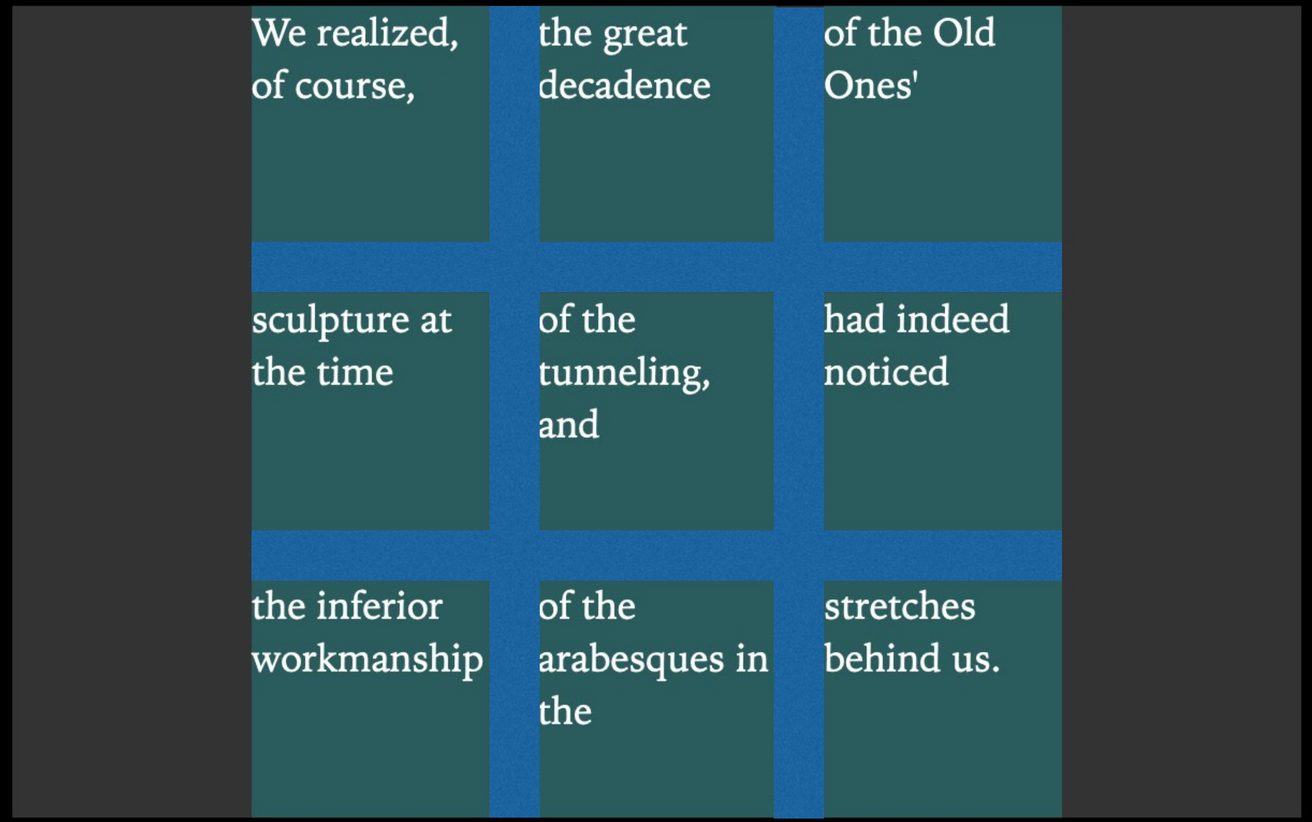
HTML

```html
<div class="grid-container">
  <div>We realized, of course,</div>
  <div>the great decadence</div>
  <div>of the Old Ones'</div>
  <div>sculpture at the time</div>
  <div>of the tunneling, and</div>
  <div>had indeed noticed</div>
  <div>the inferior workmanship</div>
  <div>of the arabesques in the</div>
  <div>stretches behind us.</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  justify-content: center;
}
```

JS



Gap

More values for `justify-content`

- `space-between`: distributes all free space *between grid columns*
- `space-around`: distributes all free space equally *around each column*
- `space-evenly`: distributes all free space equally *between each column, as well as start & end of container*

`space-between`

Distributes all free space *between grid columns*, so there is no free space at the start & end of the container

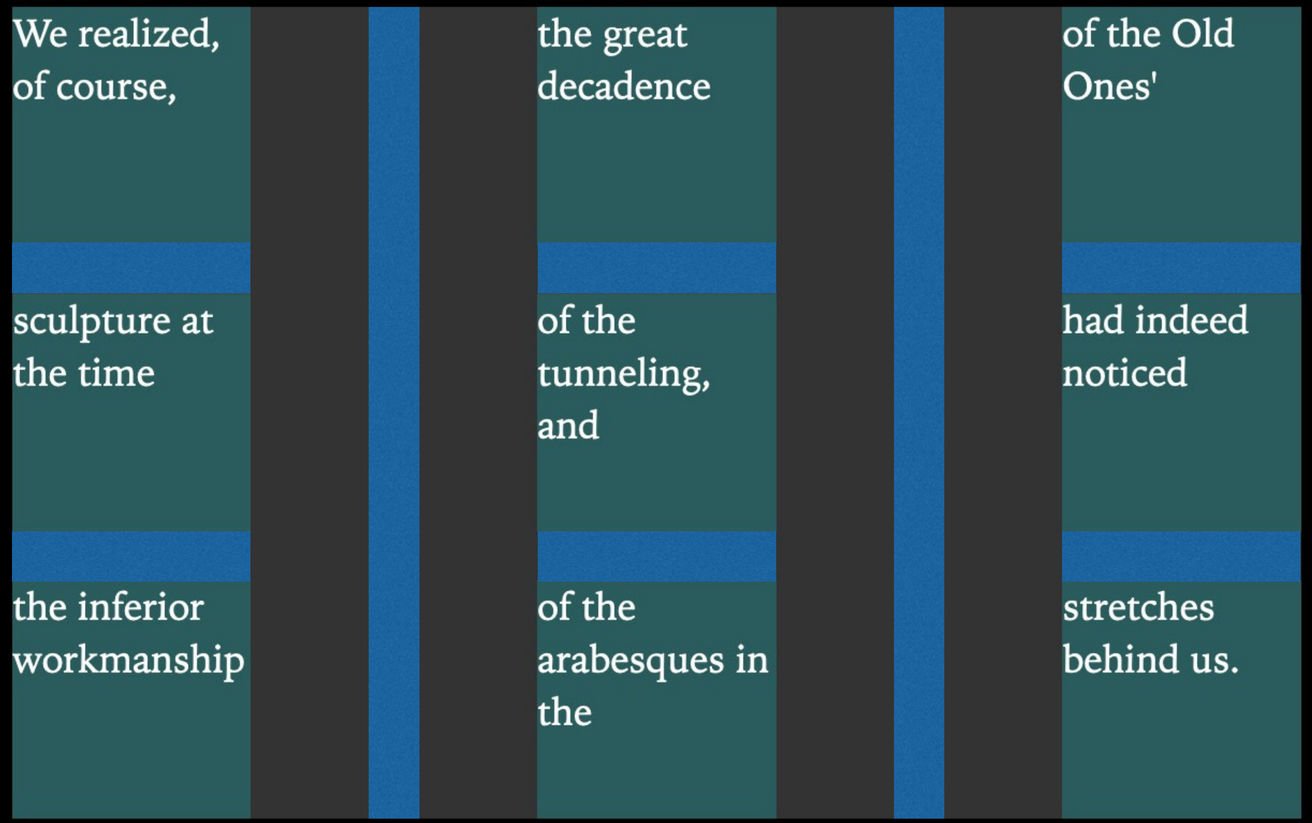
HTML

```html
<div class="grid-container">
  <div>We realized, of course,</div>
  <div>the great decadence</div>
  <div>of the Old Ones'</div>
  <div>sculpture at the time</div>
  <div>of the tunneling, and</div>
  <div>had indeed noticed</div>
  <div>the inferior workmanship</div>
  <div>of the arabesques in the</div>
  <div>stretches behind us.</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  justify-content: space-between;
}
```

JS



Gap

`space-around`

Distributes free space *equally around (on either side of) each column*

The amount of space added between each column is twice that which is added to the start & end of the container

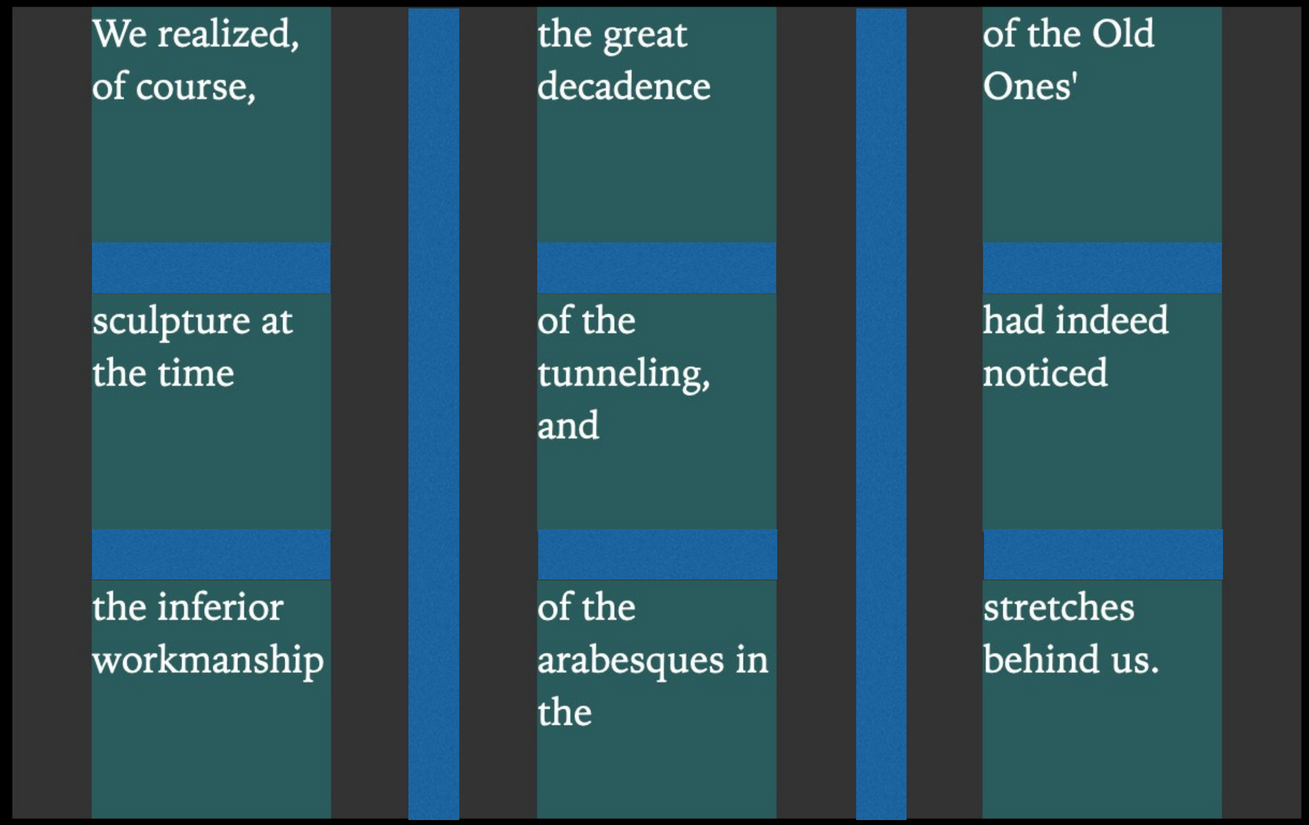
HTML

```html
<div class="grid-container">
  <div>We realized, of course,</div>
  <div>the great decadence</div>
  <div>of the Old Ones'</div>
  <div>sculpture at the time</div>
  <div>of the tunneling, and</div>
  <div>had indeed noticed</div>
  <div>the inferior workmanship</div>
  <div>of the arabesques in the</div>
  <div>stretches behind us.</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  justify-content: space-around;
}
```

JS



Gap

`space-evenly`

Distributes all free space *equally between each column, as well as start & end of container*, i.e.:

» start of container & 1st column
» between each column
» between last column & end of container

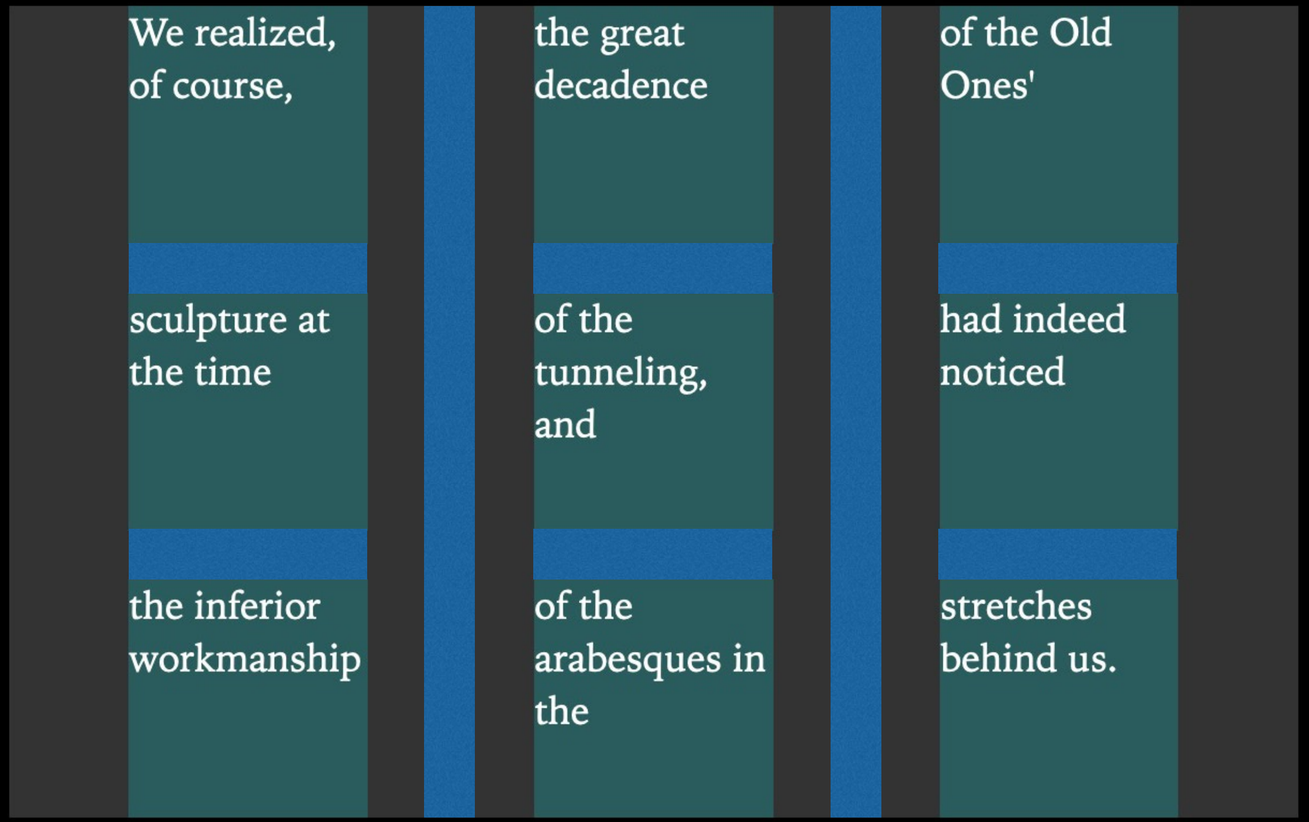
HTML

```html
<div class="grid-container">
  <div>We realized, of course,</div>
  <div>the great decadence</div>
  <div>of the Old Ones'</div>
  <div>sculpture at the time</div>
  <div>of the tunneling, and</div>
  <div>had indeed noticed</div>
  <div>the inferior workmanship</div>
  <div>of the arabesques in the</div>
  <div>stretches behind us.</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  justify-content: space-evenly;
}
```

JS



Gap

✏️ SIDE NOTE

Why didn’t you see the Firefox Grid Inspector in the last series of screenshots?

Because `*-content` alignment causes very buggy behavior in Firefox & it doesn’t highlight the grid properly

`align-content`

Sets *alignment of grid tracks & gaps along the block axis* relative to the grid container

Values & behavior are exactly like `justify-content`, but for putting space around *rows* instead of columns

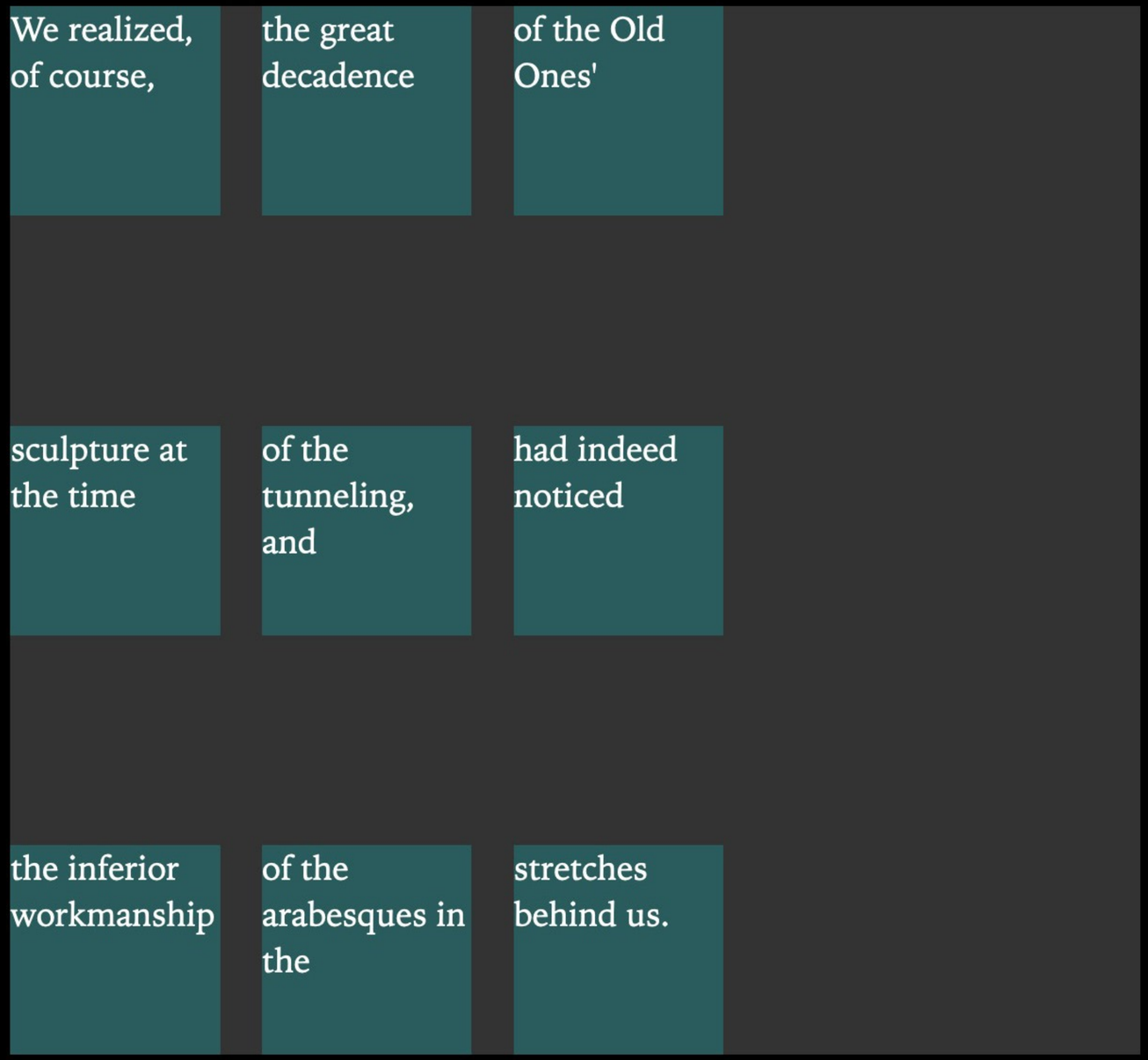
HTML

```html
<div class="grid-container">
  <div>We realized, of course,</div>
  <div>the great decadence</div>
  <div>of the Old Ones'</div>
  <div>sculpture at the time</div>
  <div>of the tunneling, and</div>
  <div>had indeed noticed</div>
  <div>the inferior workmanship</div>
  <div>of the arabesques in the</div>
  <div>stretches behind us.</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  height: 500px;
  align-content: space-between;
}
```

JS



`place-content`

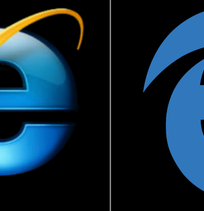
Sets *alignment of grid tracks & gaps along the block & inline axes* relative to the grid container

Shorthand for `align-content` & `justify-content`

`place-content` can accept 1 or 2 values, e.g.:

`place-content: start`
Sets value for *both* `align-content` & `justify-content`

`place-content: start space-between`
Sets value for `align-content` & *then* `justify-content`

					iOS		
align-content	–	16	52	10.1	10.3	57	57
justify-content	–	16	52	10.1	10.3	57	57
place-content	–	79	60	11	11	59	Y

Aligning Grid Items

```
align-items
justify-items

place-items


align-self
justify-self

place-self
```

All of the `*-items` & `*-self` properties in this section align grid items in relation to the areas in which they are placed

Therefore, the area must be larger than the grid item for these to take effect

Align All Grid Items

`justify-items`

Aligns all grid items along the inline axis of their areas

Values for `justify-items`

- `normal`: acts as either `stretch` or `start` (default)
- `stretch`: stretches grid items to fill their areas but only if items are sized `auto`
- `start`: align grid items flush with inline start edges of area
- `end`: align grid items flush with inline end edges of area
- `center`: align grid items in inline centers of areas

✏️ SIDE NOTE

There are some values — e.g., `self-start` & `self-end` — that are for edge cases that we aren't going to cover here

`normal`

Default value that acts like `stretch`, unless the flex item has intrinsic dimensions, in which case it acts like `start`

Isn’t CSS fun?

`stretch`

Stretches grid items to fill their areas but only if items are sized `auto` (which is the default), so `stretch` is what you're going to see most of the time when you first create a grid

If they are sized any other way, `stretch` behaves like `start`

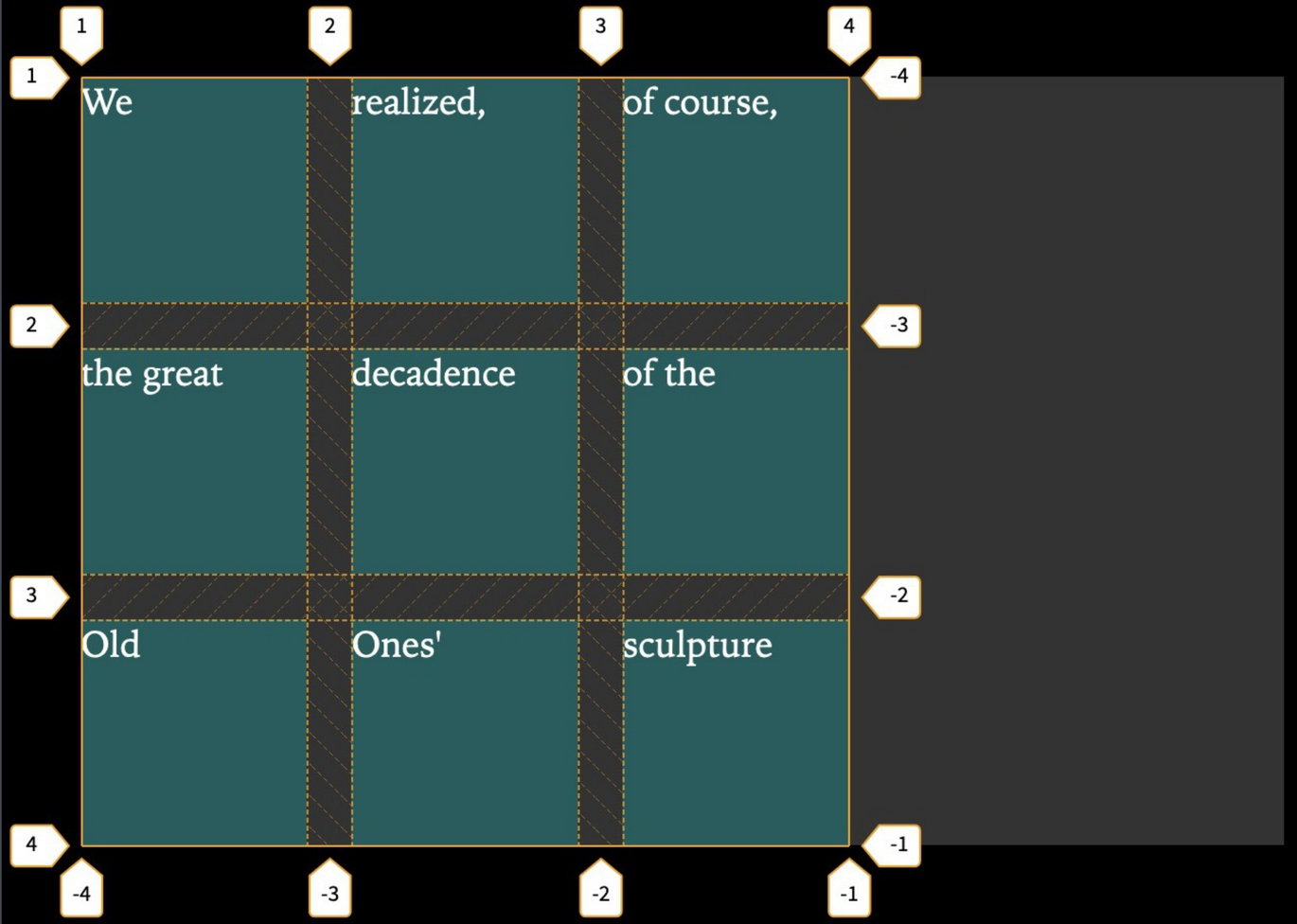
HTML

```html
<div class="grid-container">
  <div>We</div>
  <div>realized,</div>
  <div>of course,</div>
  <div>the great</div>
  <div>decadence</div>
  <div>of the</div>
  <div>Old</div>
  <div>Ones'</div>
  <div>sculpture</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  justify-items: stretch;
}
```

JS



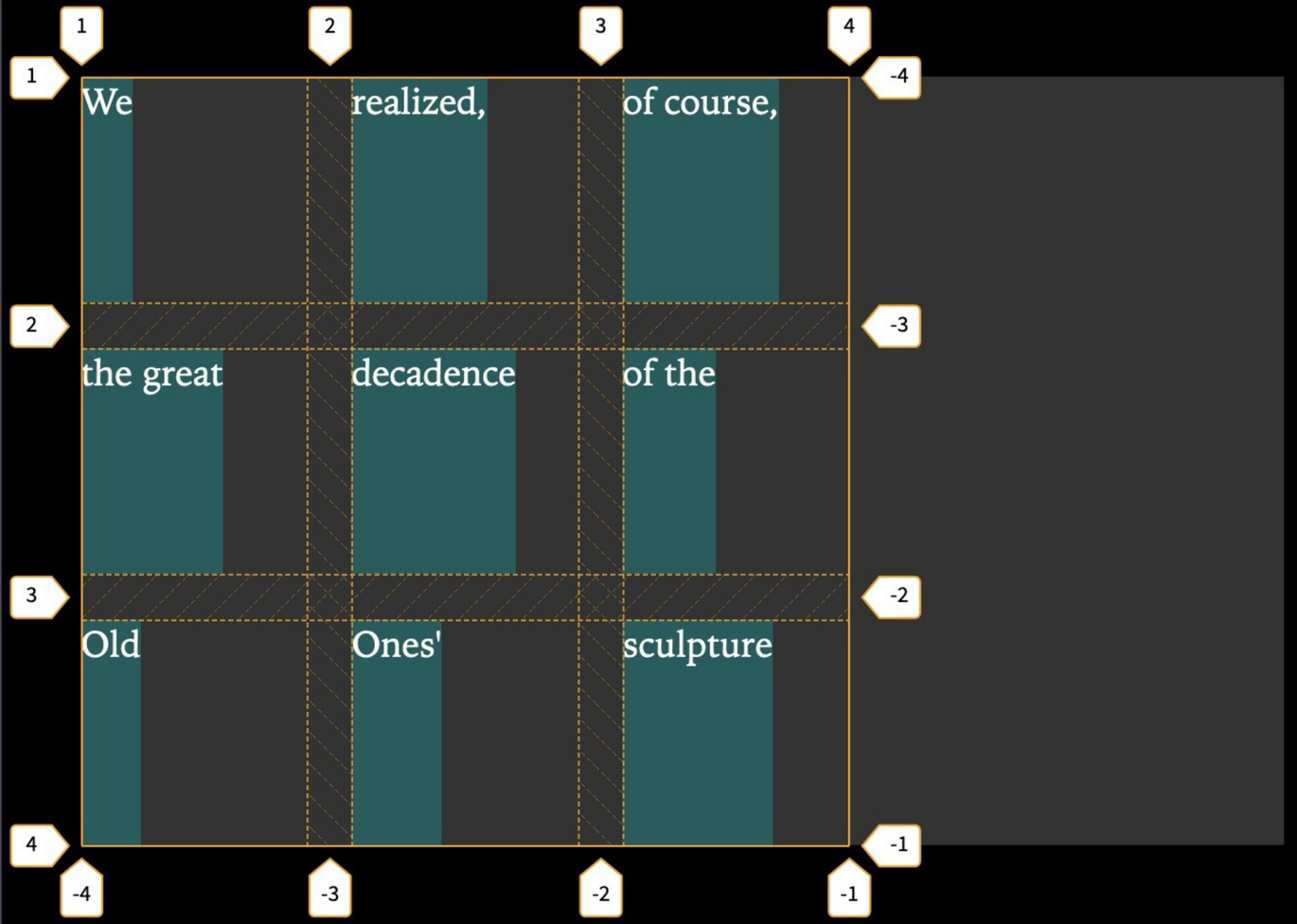
HTML

```html
<div class="grid-container">
  <div>We</div>
  <div>realized,</div>
  <div>of course,</div>
  <div>the great</div>
  <div>decadence</div>
  <div>of the</div>
  <div>Old</div>
  <div>Ones'</div>
  <div>sculpture</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  justify-items: start;
}
```

JS



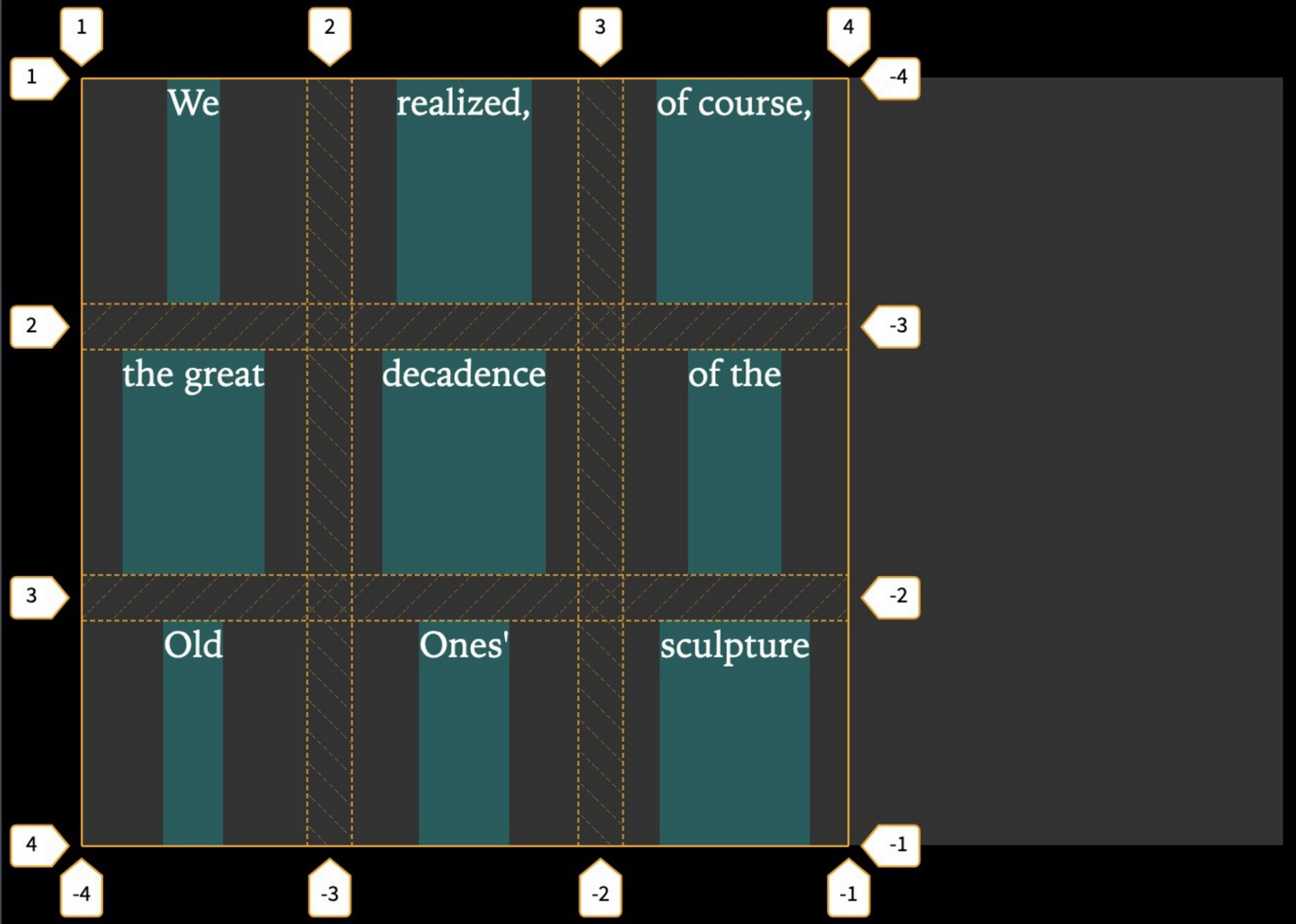
HTML

```html
<div class="grid-container">
  <div>We</div>
  <div>realized,</div>
  <div>of course,</div>
  <div>the great</div>
  <div>decadence</div>
  <div>of the</div>
  <div>Old</div>
  <div>Ones'</div>
  <div>sculpture</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  justify-items: center;
}
```

JS



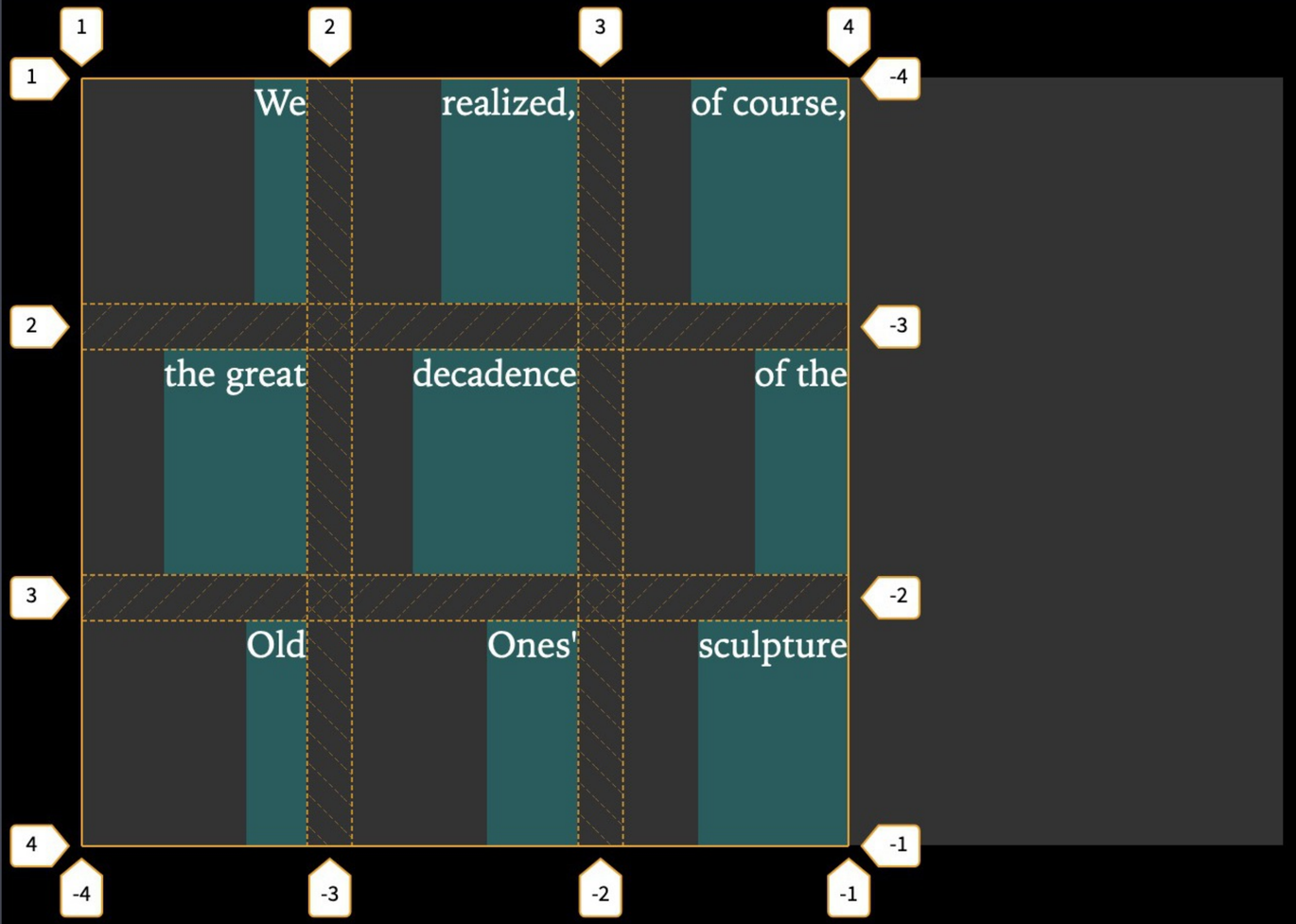
HTML

```html
<div class="grid-container">
  <div>We</div>
  <div>realized,</div>
  <div>of course,</div>
  <div>the great</div>
  <div>decadence</div>
  <div>of the</div>
  <div>Old</div>
  <div>Ones'</div>
  <div>sculpture</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(3, 100px);
  grid-template-rows: repeat(3, 100px);
  justify-items: end;
}
```

JS



`align-items`

Aligns all grid items along the block axis of their areas

Values & behavior are exactly like `justify-items`, but for affecting things along the block axis instead of inline

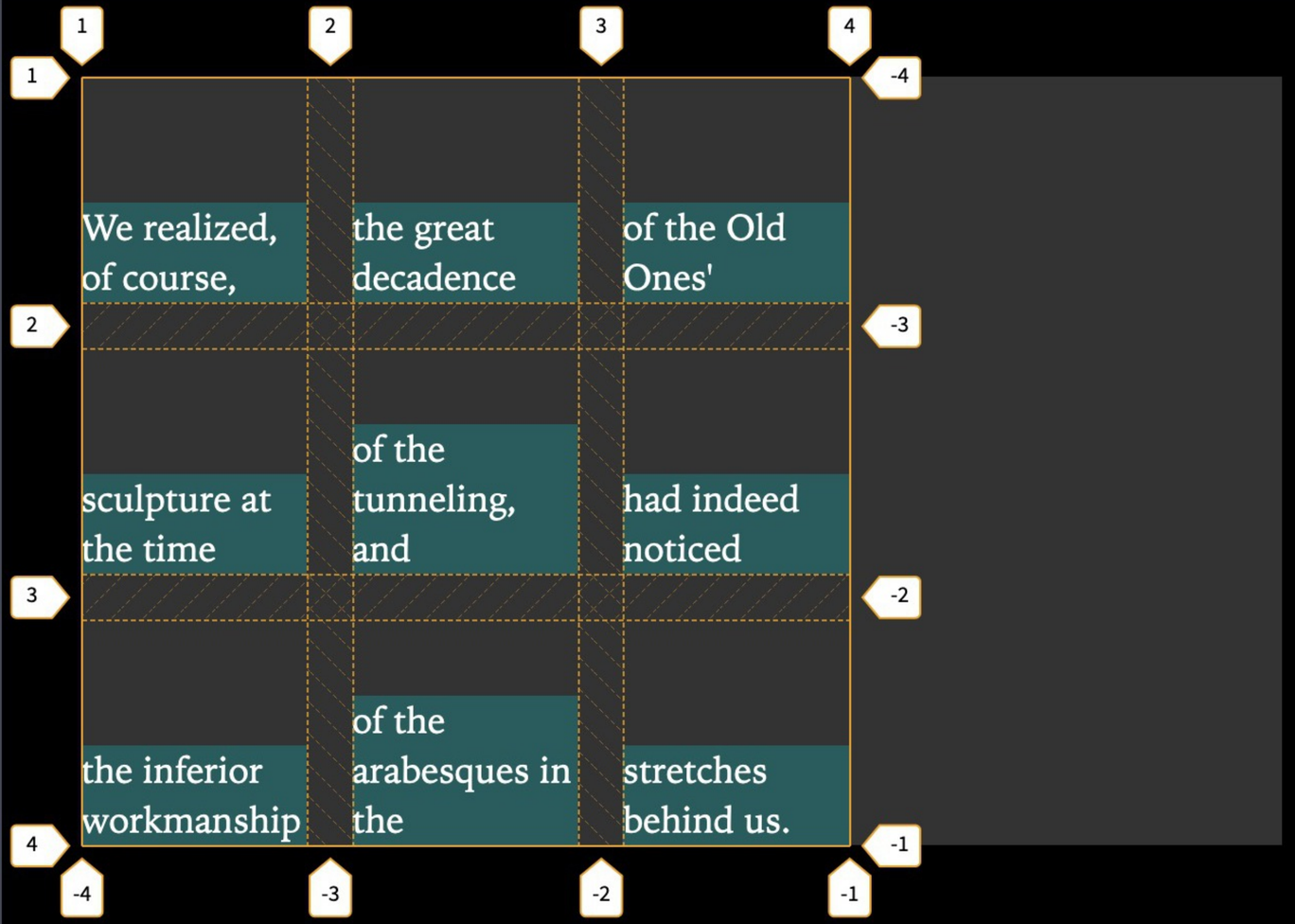
HTML

```html
<div class="grid-container">
  <div>We realized, of course,</div>
  <div>the great decadence</div>
  <div>of the Old Ones'</div>
  <div>sculpture at the time</div>
  <div>of the tunneling, and</div>
  <div>had indeed noticed</div>
  <div>the inferior workmanship</div>
  <div>of the arabesques in the</div>
  <div>stretches behind us.</div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  align-items: end;
}
```

JS



Align Individual Grid Items

`align-self`

Aligns a grid item along the block axis of its area

`justify-self`

Aligns a grid item along the inline axis of its area

Values for `align-self` (block) & `justify-self` (inline)

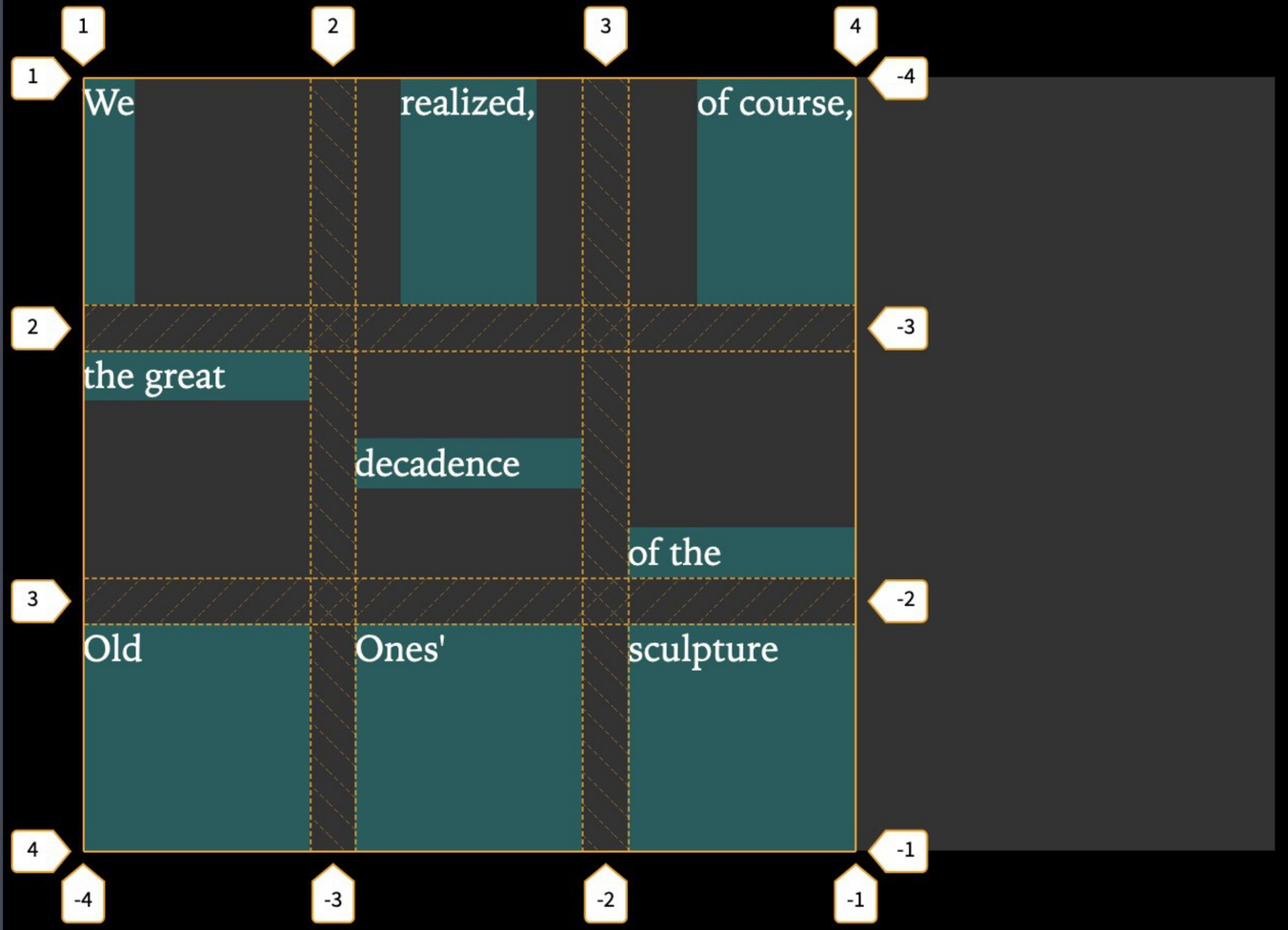
- `normal`: acts as either `stretch` or `start` (default)
- `stretch`: stretches a grid item to fill its area but only if item is sized `auto`
- `start`: align a grid item flush with start edge of area
- `end`: align a grid item flush with end edge of area
- `center`: align a grid item in center of area

HTML

CSS (SCSS) Compiled

```
    grid-template-columns: 100px 100px 100px;
    grid-template-rows: 100px 100px 100px;
  }
  .grid-container > *:nth-child(1) {
    justify-self: start;
  }
  .grid-container > *:nth-child(2) {
    justify-self: center;
  }
  .grid-container > *:nth-child(3) {
    justify-self: end;
  }
  .grid-container > *:nth-child(4) {
    align-self: start;
  }
  .grid-container > *:nth-child(5) {
    align-self: center;
  }
  .grid-container > *:nth-child(6) {
    align-self: end;
  }

```

JS



`place-self`

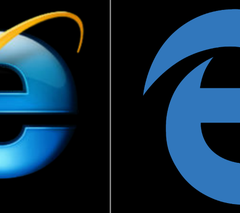
Sets *alignment of grid item along the block & inline axes* relative to its area

Shorthand for `align-self` & `justify-self`

`place-self` can accept 1 or 2 values, e.g.:

`place-self: center`
Sets value for *both* `align-self` & `justify-self`

`place-self: start center`
Sets value for `align-self` & *then* `justify-self`

					iOS		
align-items	–	16	52	10.1	10.3	57	57
justify-items	–	16	45	10.1	10.3	57	57
align-self	10*	16	52	10.1	10.3	57	57
justify-self	–	16	45	10.1	10.3	57	57
place-self	?	?	45	–	–	59	59

*Use `-ms-grid-column-align`

There are 2 ways tracks are created — either *by* you or *for* you

- *Explicit*: you explicitly define how many rows & columns there are in the grid (that's what we've learned until now)

- *Implicit*: items that don't fit within the grid's explicitly defined rows & columns will cause the needed rows & columns to be created for you

Implicit Grid

```
grid-auto-rows

grid-auto-columns

grid-auto-flow
```

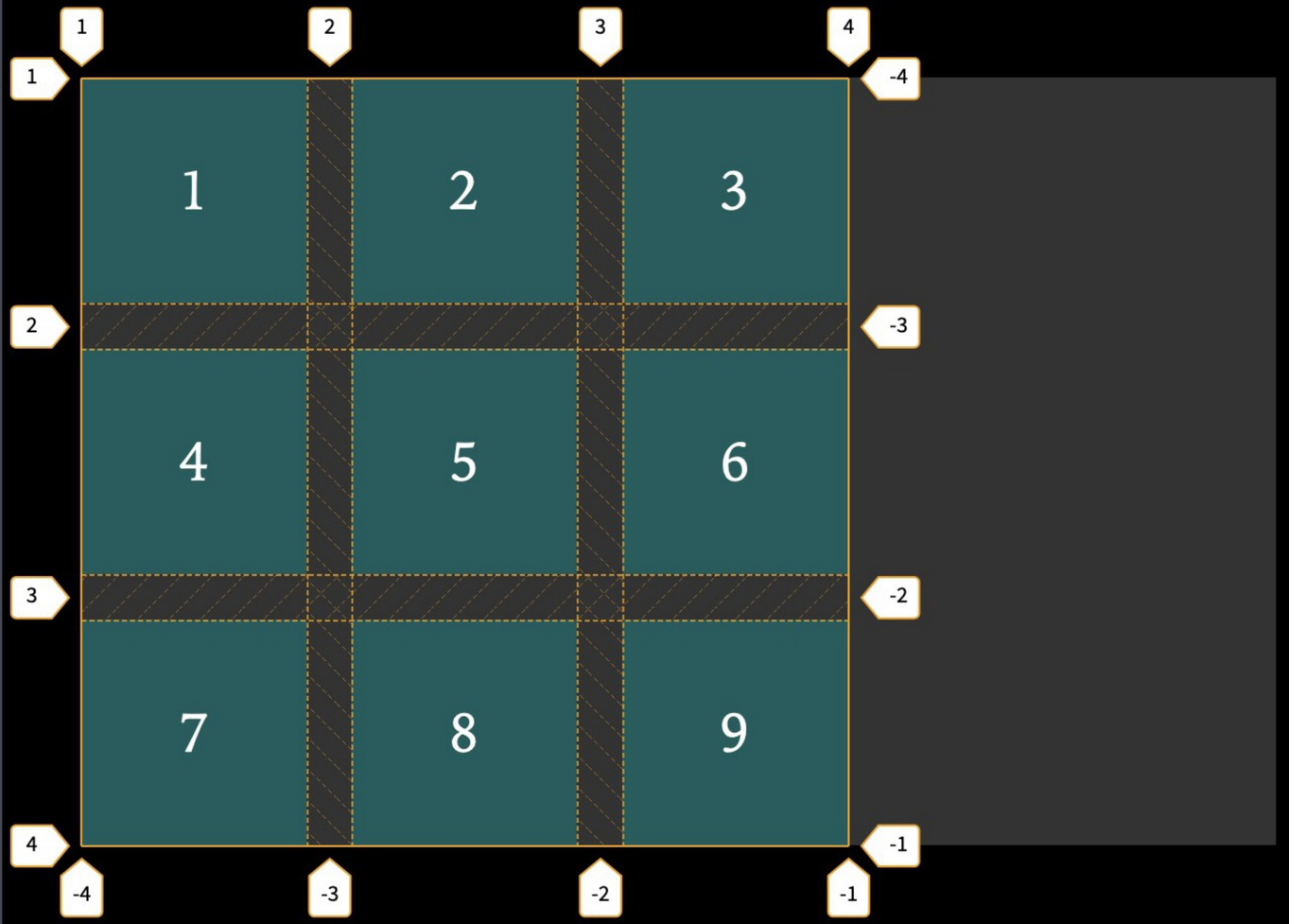
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



We explicitly created 6 tracks:
3 columns & 3 rows

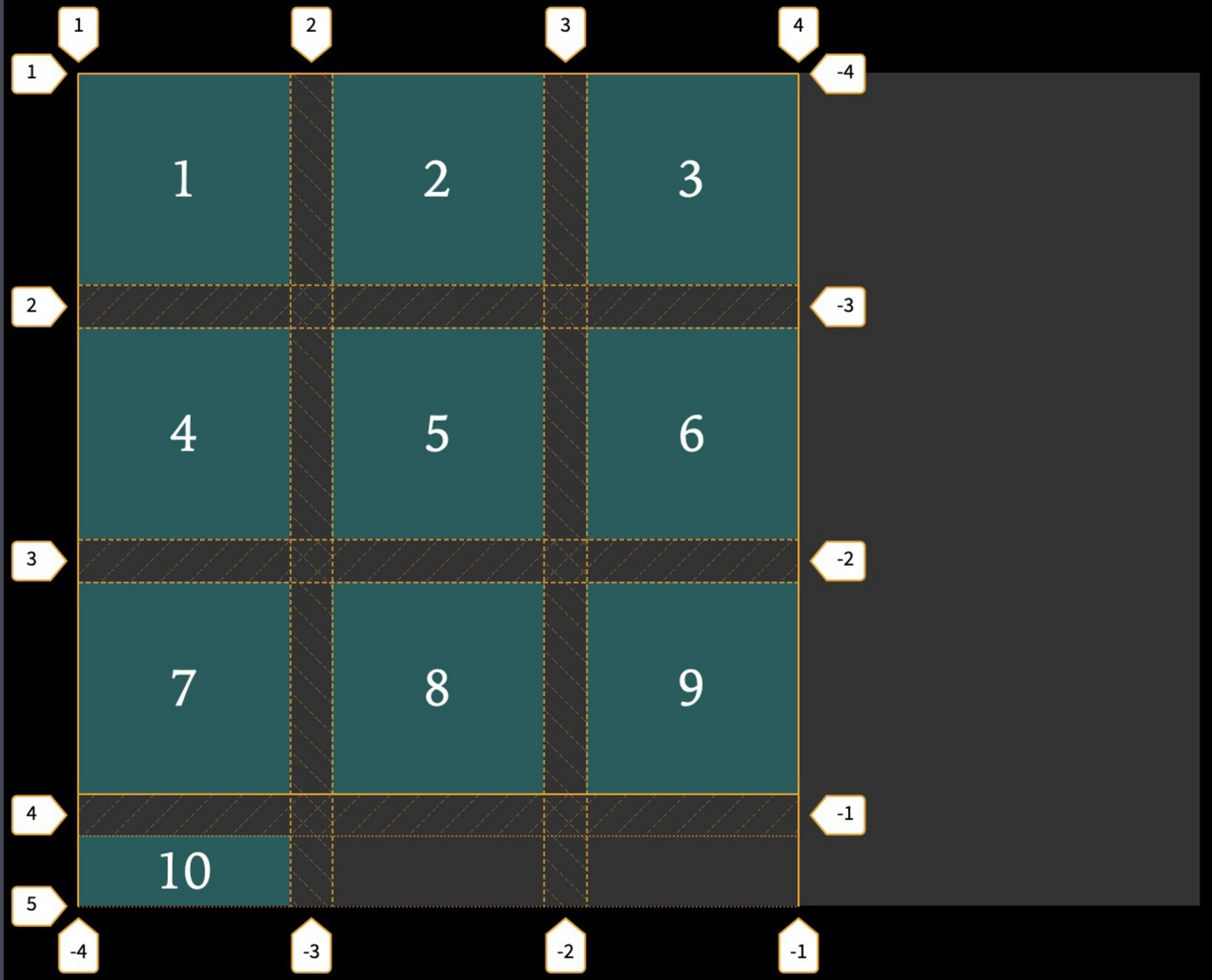
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
```

JS



We added a grid item that doesn’t fit, so a new implicit track is created

`grid-auto-flow`

Specifies *3 important behaviors of grid items*:

» the *direction in which automatically placed items fill the grid*: row or column?
» the *direction in which implicit tracks are created*: rows or columns?
» how automatically placed items are *packed*

Values for `grid-auto-flow`

- `row`: automatically placed items go in rows, & new implicit tracks are rows (default)
- `column`: automatically placed items go in columns, & new implicit tracks are columns
- `dense`: items attempt to fill in empty areas earlier in the grid where previous items wouldn’t fit, which may very well cause items to appear out of order!

You can combine either `row` or `column` with `dense`

- `row dense`: fill each row using the `dense` algorithm (identical to `dense`, so you'll never need to use it)
- `column dense`: fill each column using the `dense` algorithm

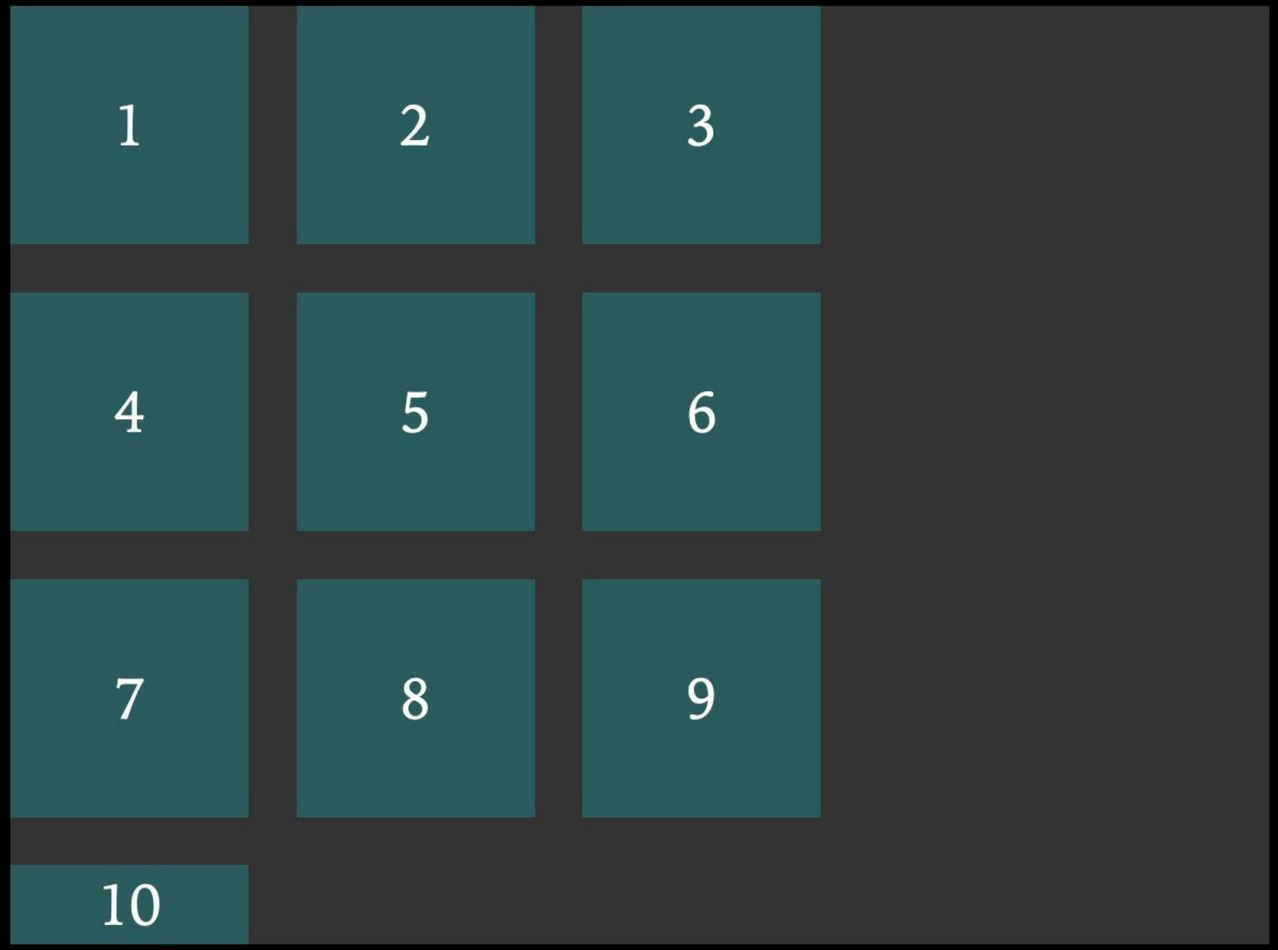
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-auto-flow: row;
}
```

JS



Why is 10 so short? Because the default for sizing is `auto`

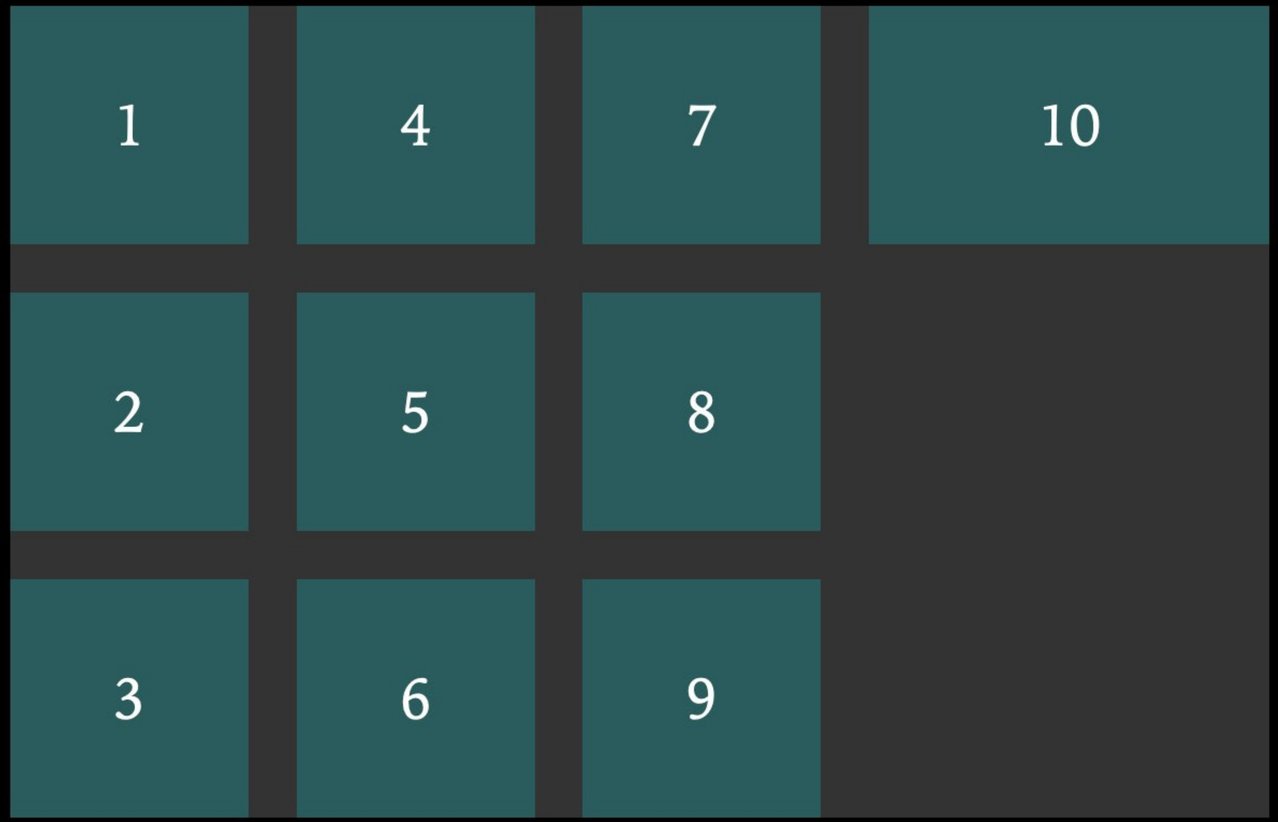
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
  grid-auto-flow: column;
}
```

JS



Why is 10 so wide? Because the default for sizing is `auto`

Sparse packing is the default: each item is placed in order, & if the item doesn’t fit in an area, it skips ahead to the next area it does fit, leaving behind empty areas

Dense packing: each item is placed into the next area in which it will fit, including previous empty areas, causing items to appear out of order

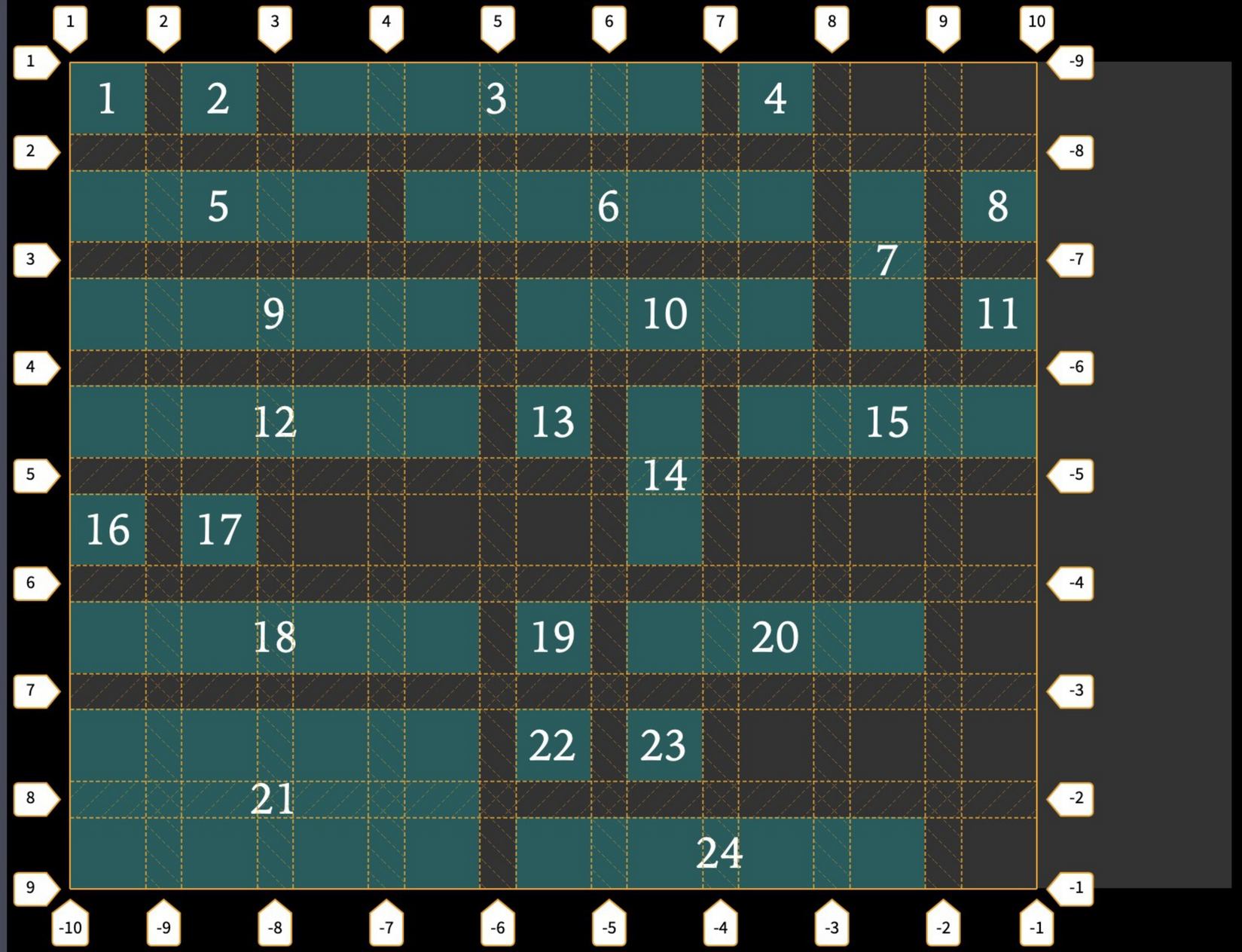
HTML

```
<div class="grid-container">
```

CSS (SCSS) Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns:
repeat(9, 42px);
  grid-template-rows:
repeat(8, 40px);
}
.grid-container > :nth-
child(3n) {
  grid-column-start: span 4;
}
.grid-container > :nth-
child(5n) {
  grid-column-start: span 3;
}
.grid-container > :nth-
child(7n) {
  grid-row-start: span 2;
}
```

JS



Sparse packing by row

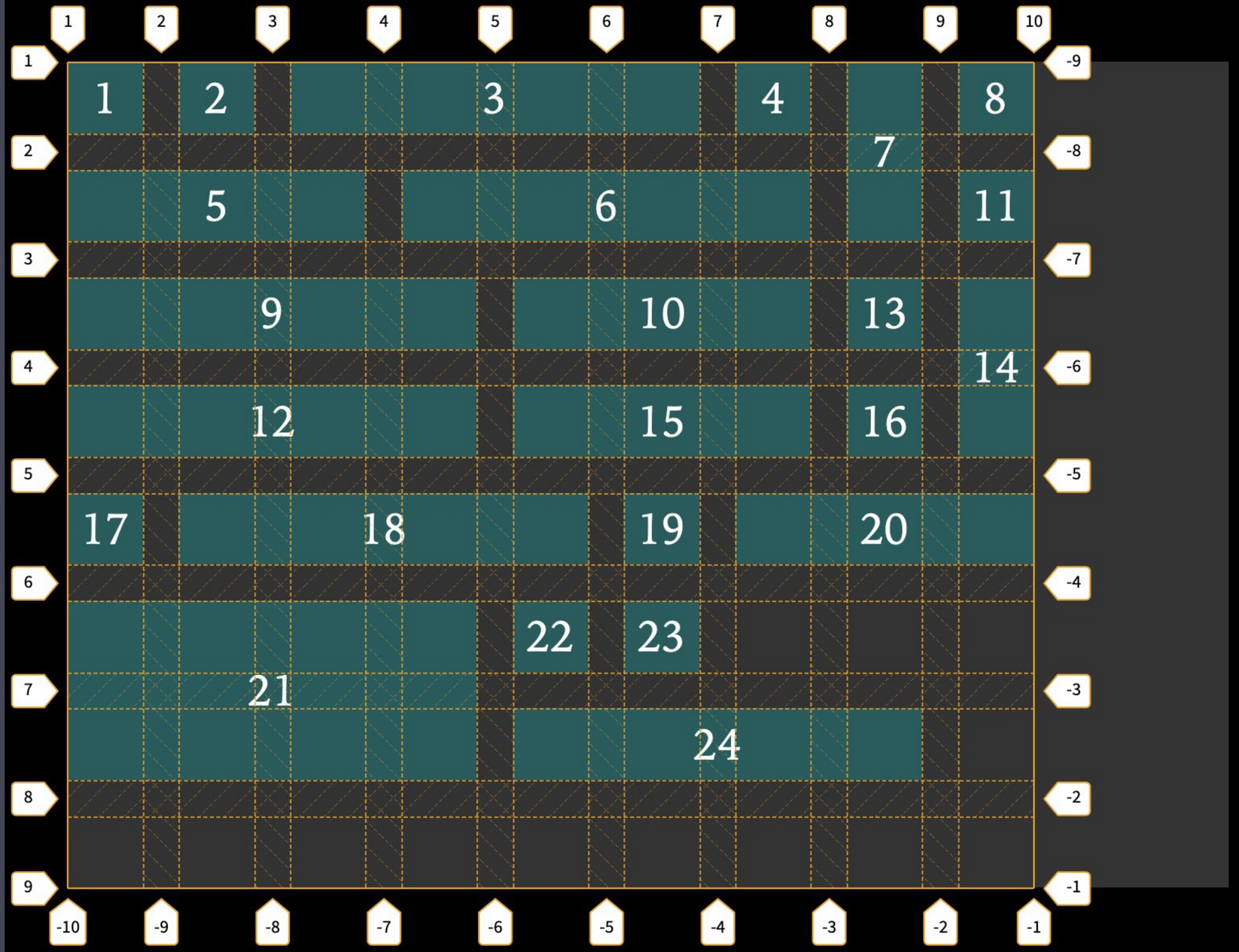
HTML

```
<div class="grid-container">
```

CSS (SCSS) Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(9, 42px);
  grid-template-rows: repeat(8, 40px);
  grid-auto-flow: dense;
}
.grid-container > :nth-child(3n) {
  grid-column-start: span 4;
}
.grid-container > :nth-child(5n) {
  grid-column-start: span 3;
}
.grid-container > :nth-child(7n) {
  grid-row-start: span 2;
```

JS



`dense` packing by row

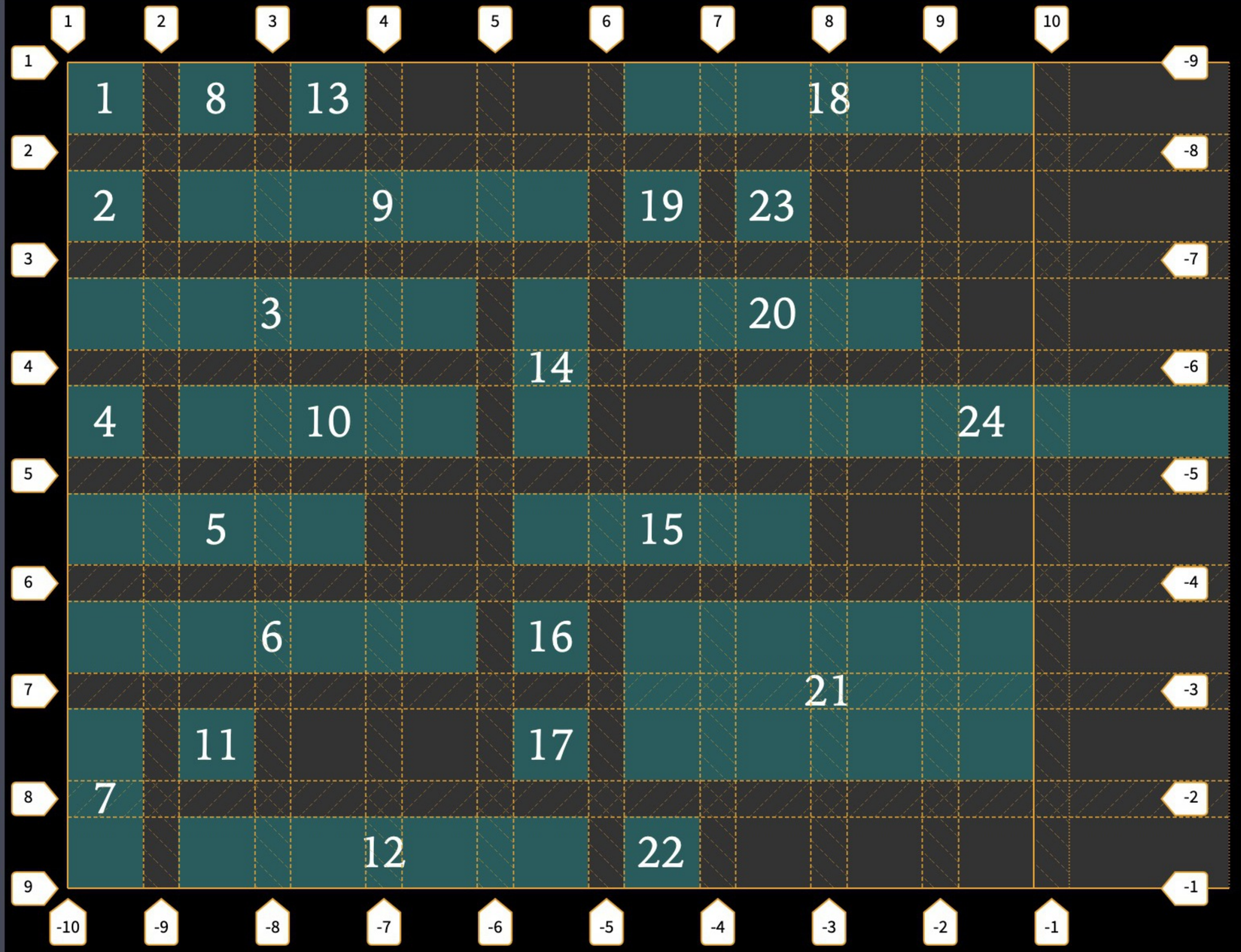
HTML

```
<div class="grid-container">
```

CSS (SCSS) Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: repeat(9, 42px);
  grid-template-rows: repeat(8, 40px);
  grid-auto-flow: column;
}
.grid-container > :nth-child(3n) {
  grid-column-start: span 4;
}
.grid-container > :nth-child(5n) {
  grid-column-start: span 3;
}
.grid-container > :nth-child(7n) {
  grid-row-start: span 2;
```

JS



Sparse packing by column

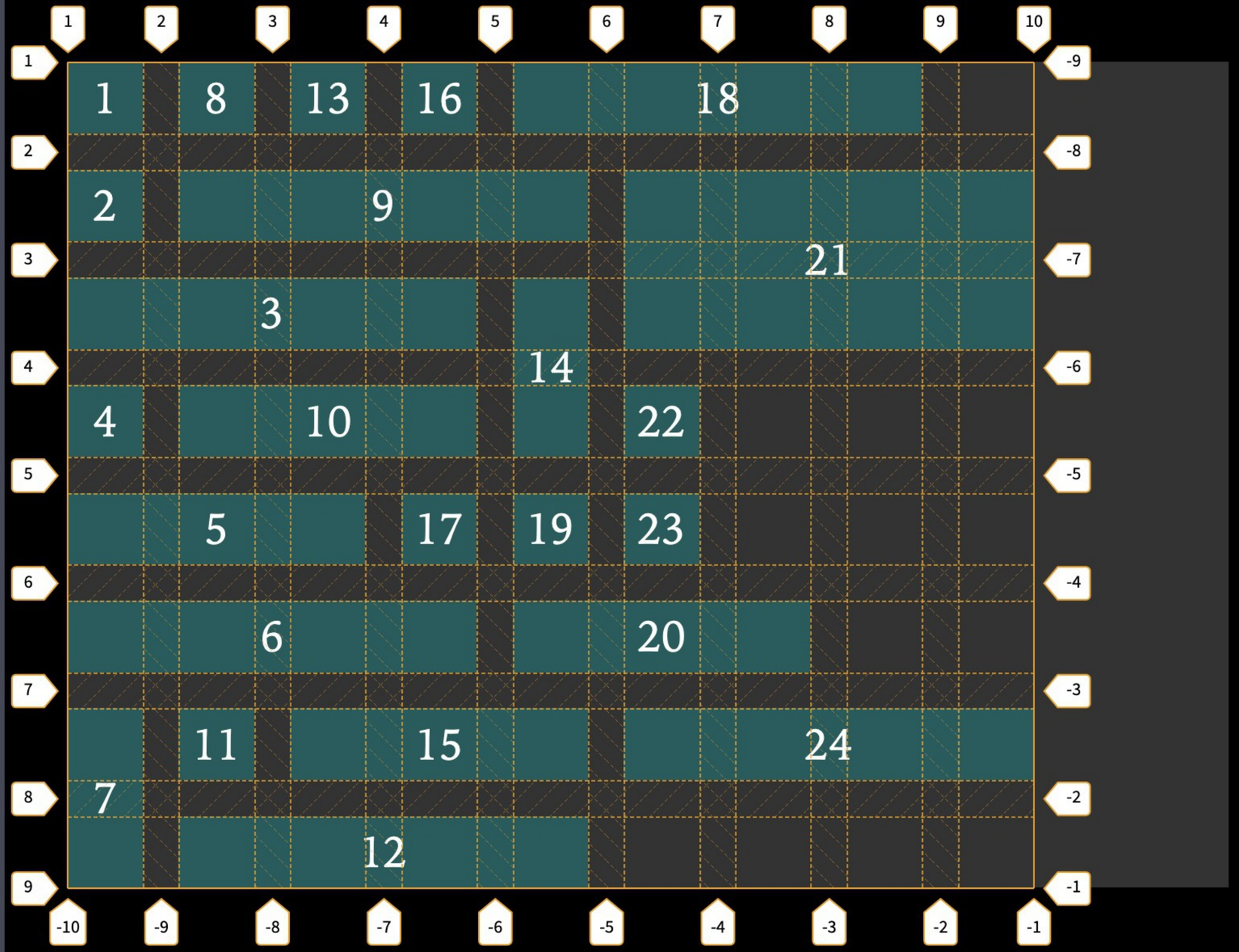
HTML

```
<div class="grid-container">
```

CSS (SCSS) Compiled

```
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns:
repeat(9, 42px);
  grid-template-rows:
repeat(8, 40px);
  grid-auto-flow: column
dense;
}
.grid-container > :nth-
child(3n) {
  grid-column-start: span 4;
}
.grid-container > :nth-
child(5n) {
  grid-column-start: span 3;
}
.grid-container > :nth-
child(7n) {
```

JS



dense packing by column

`grid-auto-rows`

Specifies the *size of an implicitly-created grid row track* or pattern of tracks when `grid-auto-flow: row`

`grid-auto-columns`

Specifies the *size of an implicitly-created grid column track* or pattern of tracks when `grid-auto-flow: column`

Values for `grid-auto-rows` & `grid-auto-columns`

- `<length>`
- `<percentage>`
- `<flex>` `fr` unit
- `max-content`
- `min-content`
- `minmax()`
- `auto`

`repeat()` is missing because values auto repeat, as you will see

Note that you cannot combine `grid-auto-rows` & `grid-auto-columns` because only 1 will have an effect due to `grid-auto-flow`

Either rows will be created (if `grid-auto-flow: row`, which is the default) or columns (if `grid-auto-flow: column`)

HTML

```html
<div class="grid-container">

</div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 100px 100px;
  grid-auto-rows: 100px 50px;
}
```

JS

Implicit tracks are created when items can’t fit in the grid

Since there are no items, no implicit rows are created, & therefore no grid is created

HTML 2 unsaved changes

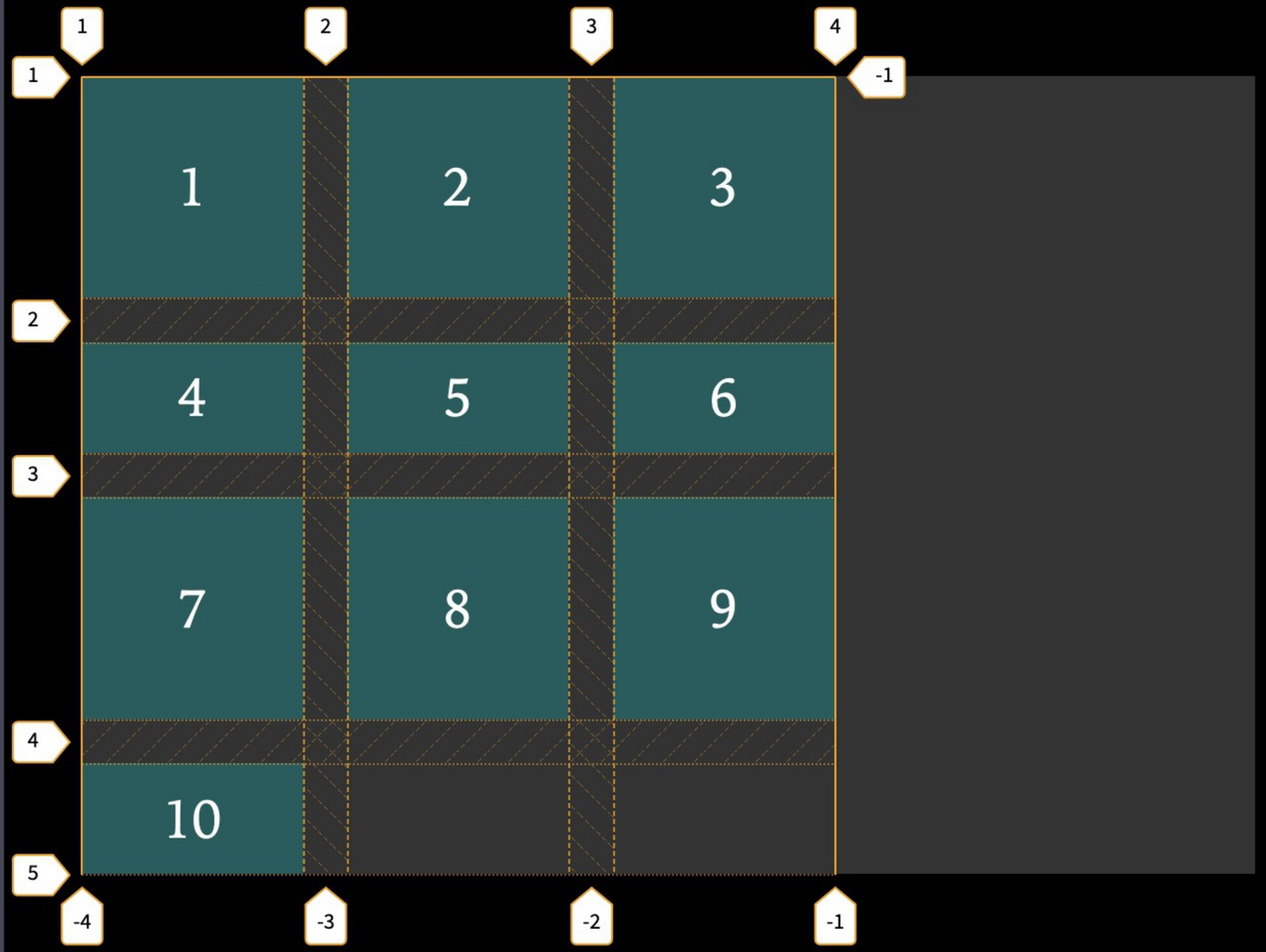
```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
</div>
```

Note that the rows repeat automatically, without needing `repeat()`

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 100px 100px;
  grid-auto-rows: 100px 50px;
}
```

JS



Implicit rows are created to accommodate grid items, using the sizes we set

					iOS		
`grid-auto-flow`	–	16	52	10.1	10.3	57	Y
`grid-auto-rows`	10*	16	70	10.1	10.3	57	Y
`grid-auto-columns`	10*	16	70	10.1	10.3	57	Y

* Uses `-ms-grid-rows` & `-ms-grid-columns`

Subgrid

```
grid-template-rows: subgrid
grid-template-columns: subgrid
```

Tells a child grid to *re-use the parent grid's lines* for rows &/or columns

HTML

```
<h1>With subgrid</h1>
<div class="cards grid">
  <div class="card subgrid">
    <header>↔</header>
    <main>↔</main>
    <footer>↔</footer>
  </div>
  <div class="card subgrid">
    <header>↔</header>
    <main>↔</main>
    <footer>↔</footer>
  </div>
```

CSS (SCSS) Compiled

```
.grid {
  display: grid;
  gap: 30px;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
}
.grid .subgrid {
  grid-row: span 3;
  display: grid;
  grid-template-rows: subgrid;
  gap: 0;
}
```

1 2 3

1 -1

Dunwich Horror	At the Mountains of Madness	Call of Cthulhu
"Oh, my Gawd, my Gawd," the voice choked out. "It's a-goin' agin, an' this time by day! It's aout—it's aout an' a-movin' this very minute, an' only the Lord knows when it'll be on us all!"	The toughness of the things was almost incredible. Even the terrific pressure of the deepest sea-bottoms appeared powerless to harm them.	The most merciful thing in the world, is the inability of the human mind to correlate all its contents. We live on a placid island of ignorance in the midst of black seas of infinity, and it was not

With subgrid

Dunwich Horror	At the Mountains of Madness	Call of Cthulhu
"Oh, my Gawd, my Gawd," the voice choked out. "It's a-goin' agin, an' this time by day! It's aout—it's aout an' a-movin' this very minute, an' only the Lord knows when it'll be on us all!" The speaker panted into silence, but another took up his message.	The toughness of the things was almost incredible. Even the terrific pressure of the deepest sea-bottoms appeared powerless to harm them.	The most merciful thing in the world, is the inability of the human mind to correlate all its contents. We live on a placid island of ignorance in the midst of black seas of infinity, and it was not meant that we should voyage far.
Innumerable Whateleys	Elder Things	R'lyeh

Without subgrid

Dunwich Horror

"Oh, my Gawd, my Gawd," the voice choked out. "It's a-goin' agin, an' this time by day! It's aout—it's aout an' a-movin' this very minute, an' only the Lord knows when it'll be on us all!"

The speaker panted into silence, but another took up his message.

Innumerable Whateleys

At the Mountains of Madness

The toughness of the things was almost incredible. Even the terrific pressure of the deepest sea-bottoms appeared powerless to harm them.

Elder Things

Call of Cthulhu

The most merciful thing in the world, is the inability of the human mind to correlate all its contents. We live on a placid island of ignorance in the midst of black seas of infinity, and it was not meant that we should voyage far.

R'lyeh

					iOS		
subgrid	–	–	71	–	–	–	–

Techniques

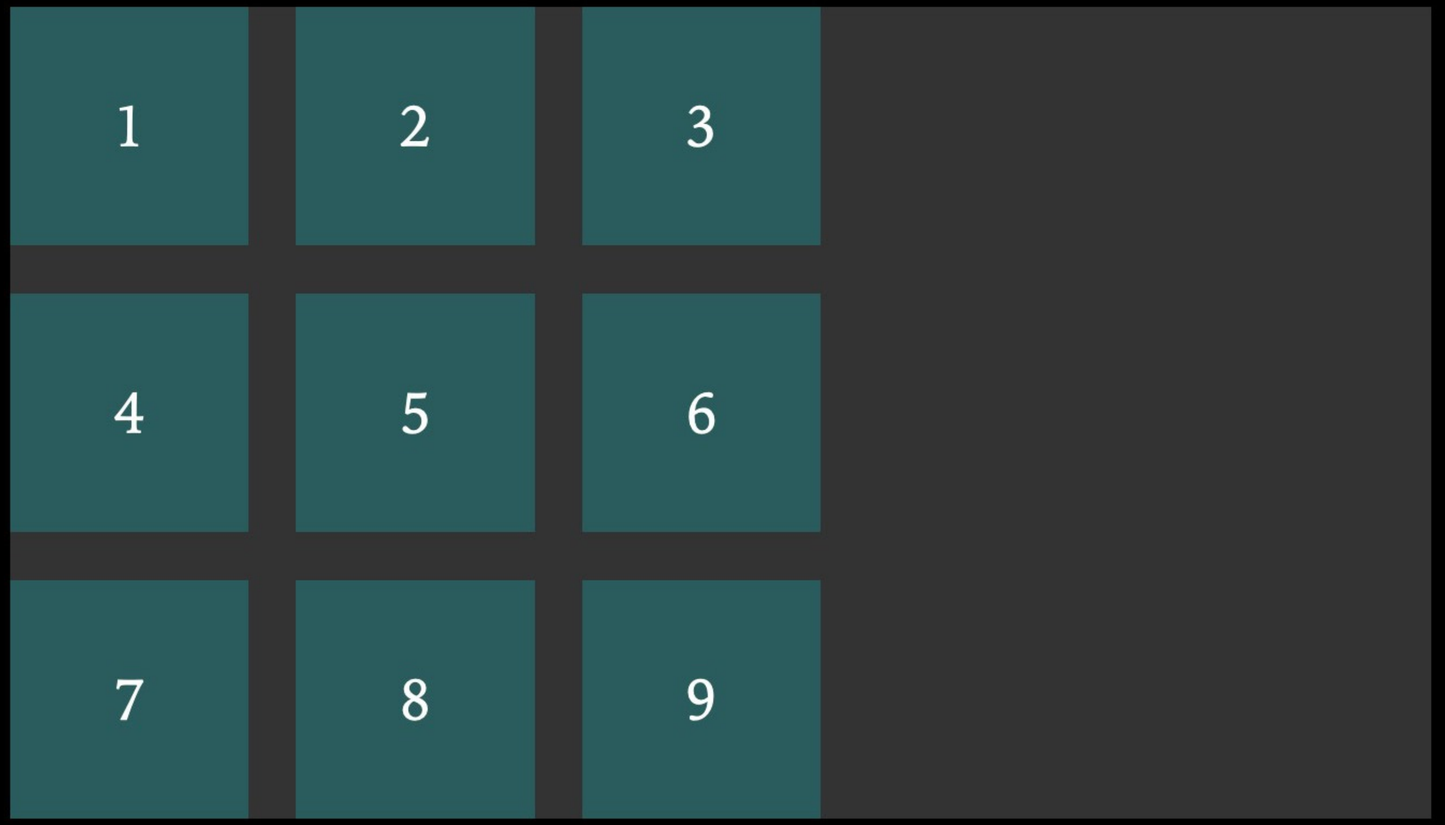
HTML

```html
<div class="grid-container">
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
  <div></div>
```

CSS (SCSS) Compiled

```scss
.grid-container {
  display: grid;
  grid-gap: 20px;
  grid-template-columns: 100px 100px 100px;
  grid-template-rows: 100px 100px 100px;
}
.grid-container > * {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

JS



Grid items can also be flex containers

Tools

Guide

A Complete Guide to Grid

Last Updated
Jul 7, 2020

Our comprehensive guide to CSS grid, focusing on all the settings both for the grid parent container and the grid child elements.

CSS Grid Layout is the most powerful layout system available in CSS. It is a 2-dimensional system, meaning it can handle both columns and rows, unlike flexbox which is largely a 1-dimensional system. You work with Grid Layout by applying CSS rules both to a parent element (which becomes the Grid Container) and to that element's children (which become Grid Items).

Grid Bugs

A curated list of Grid interop issues

css css-grid-layout

23 commits **1** branch **0** releases **2** contributors MIT

Branch: **master** New pull request Find file Clone or download

rachelandrew committed on Sep 21, 2017 update on issue 14 Latest commit `62c8f9b` on Sep 21, 2017

.gitignore	Initial files	9 months ago
LICENSE	Initial commit	9 months ago
README.md	update on issue 14	8 months ago

README.md

GridBugs

Inspired by Flexbugs this list aims to be a community curated list of CSS Grid Layout bugs, incomplete implementations and interop issues. Grid shipped into browsers in a highly interoperable state, however there are a few issues - let's document any we find here.

While I'd like to focus on issues relating to the Grid Specification here, it is likely that related specs such as Box Alignment will need to be referenced. If you think you have spotted an issue and it seems to relate to Grid, post it. We can work out the details together and make sure browser or spec issues get logged in the right place.

The bugs

1. `grid-auto-rows` and `grid-auto-columns` should accept a track listing
2. Repeat notation should accept a track listing
3. Fragmentation support
4. Sizing of items with an intrinsic size inside fixed size grid tracks
5. Items with an intrinsic aspect ratio should align start
6. The `grid-gap` property should accept percentage values
7. Grid gaps should behave as auto-sized tracks?
8. Setting max-height to a percentage should scale down a larger image inside a grid track
9. `min-content` and `max-content` in track listings
10. Some HTML elements can't be grid containers
11. A textarea that is a grid item collapses to zero width
12. Space distributed to empty tracks spanned by an item with content
13. Inconsistency with percentage padding and margins
14. fit-content is stretching

1. `grid-auto-rows` and `grid-auto-columns` should accept a track listing

Demos	Browsers affected	Tracking bugs	Tests
1.1 — *bug*	Firefox	Firefox #1339672	WPT

The properties `grid-auto-rows` and `grid-auto-columns` enable an author to set the size of tracks created in the implicit grid. The spec says:

Thank you!

scott@granneman.com
www.granneman.com
ChainsawOnATireSwing.com
@scottgranneman

jans@websanity.com
websanity.com

CSS Grid Layout

Robust Layout Using Rows & Columns

R. Scott Granneman & Jans Carton

Changelog

2020-07-17 3.3: Updated screenshots for `max-content`

2020-07-16 3.2: Added details re: `grid-template-columns` & `grid-template-rows`; added `span`; changed subtitle; added compatibility tables where missing; added source links where missing; updated lots more

Changelog

2020-05-02 3.1: Added new section on *Inspecting Grids*

2018-10-19 3.0: Moved grid slides out of *CSS Layout* & into its own slide deck; updated grid screenshot from Can I Use; completely re-organized the entire slide deck & added tons of new content

Licensing of this work