PUCD 2035
CORE 1 INTERACTION

SECTION B
T 9–11h40

SESSION SEVEN
OCTOBER 10, 2023

EVEN MORE CSS

ENDLESS APPRECIATION
TO ERIC LI +
MICHAEL FEHRENBACH
FOR THE MATERIAL

DAVIS SCHERER
DSCHERER@NEWSCHOOL.EDU

AGENDA

1. MIDTERM!
2. CSS GRID
3. IMAGES
4. EVEN MORE CSS
5. A MUCH DESERVED BREAK
6. PRACTICE

MIDTERMS

8/29 The Internet	9/05 HTML	9/12 CSS	9/19 Layouts
9/26 Web Hosting + Responsive Design	10/3 Flexbox + CD LECTURE @10h30	10/10 Even More CSS	10/17 Midterm Critiques

(AFTER MIDTERMS)

10/24 Intro to Javascript	**10/31** Javascript + the DOM	**11/07** Javascript Libraries	**11/14** TBD
11/21 Metadata + Access	**11/28** Review + Wrap Up	**12/5** FINAL Critiques	

MIDTERMS

1.

A **FULLY** **FUNCTIONAL** set of five harmonic collection entries, linked by a hub page, submitted either as a zip file or a GitHub link.

MIDTERMS

2.

A **SHORT, 1-2 PAGE REFLECTION***
on your Harmonic Collection so far—
how has your theme evolved? How are you finding coding? How can you expand your entries, and what are some challenges you'd like to tackle next?

*** DOUBLE SPACED, 12PT… ~250-500 WDS**

MIDTERMS

3.

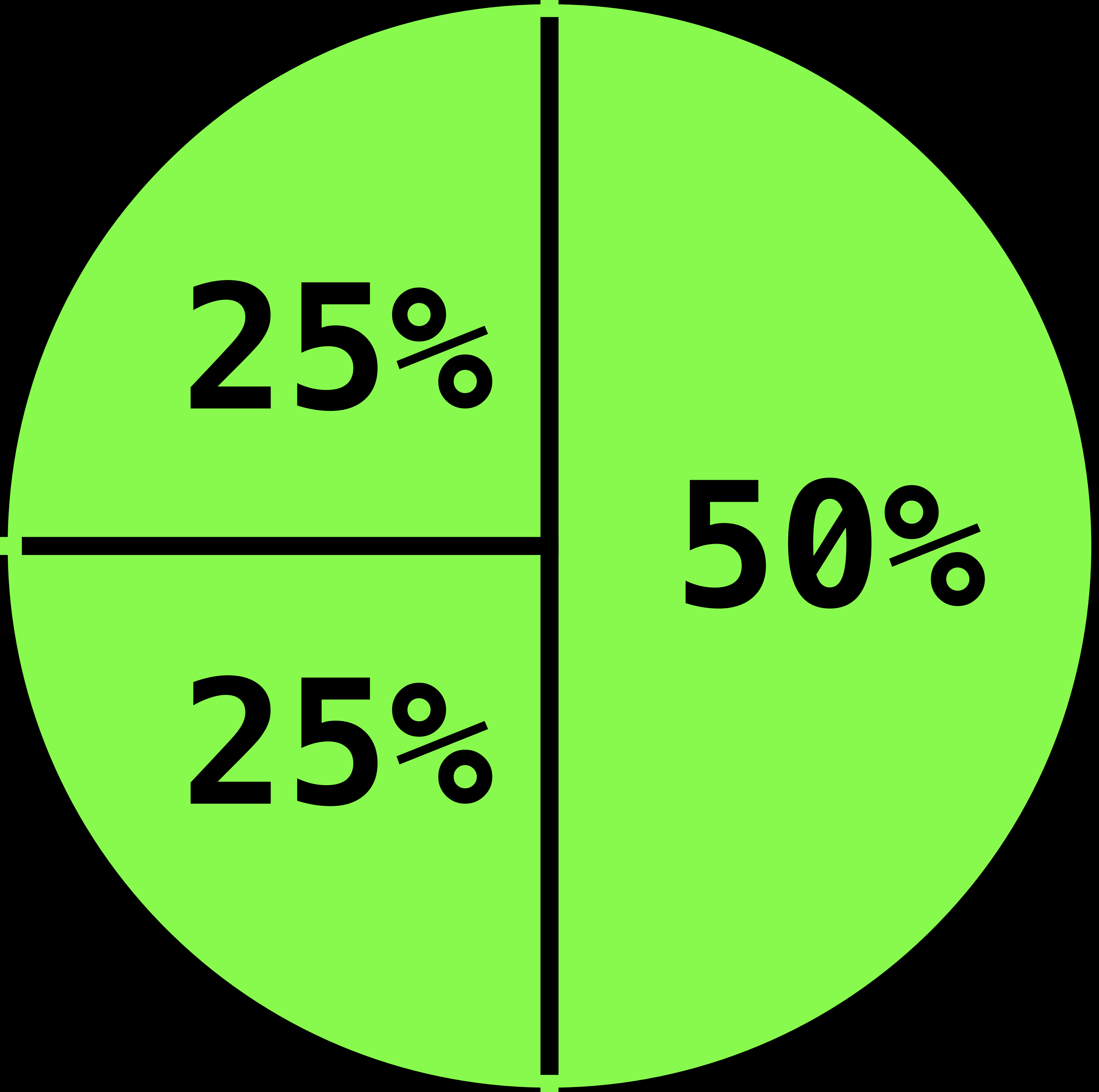
A **BRIEF** presentation of your work to the class—for 2-5 minutes, walk the class through your progress live in your code or via a presentation; the remaining time (4-7 minutes) will be a short critique.

PARTICIPATION IS FOR A GRADE

MIDTERMS

You will present off of your computers, to ensure functionality. If you need an accommodation, reach out to me ASAP to ensure I have your files in working order.

MIDTERMS



WRITTEN RESPONSE

IN-CLASS CRITIQUE

HARMONIC COLLECTION

MIDTERMS

A	All required elements are **completely functional, exceptionally rendered. conceptually robust,** and **well-designed.**
B	All required elements are **present, conceptually robust, considered, and graphically sound.**
C	Work is mostly functional, and of average quality and consideration.
D	Work is nonfunctional, unlinked, or otherwise flawed in a major way, but can be accessed.
F	Work is not submitted, nonfunctional, unlinked, or otherwise flawed in a major way that prevents access and review.

MIDTERMS

ABSOLUTELY NO LATE WORK WILL BE ACCEPTED.

MIDTERMS

SIGN UP FOR YOUR MIDTERM TIME SLOT AT YOUR EARLIEST CONVENIENCE

PRACTICE CRITIQUE

Let’s contextualize critique in this class by discussing what we might talk about next week.

PRACTICE CRITIQUE

The theme is “Voyage”. What can you say about its layout, connection to the concept, typography? How is this like and unlike critiquing other media?

Going,

Going, (BOEING)↗

gone

Last year, my family went on Vacation.
It was a really interesting voyage.
There was turbulence...

MIDTERMS

QUESTIONS?

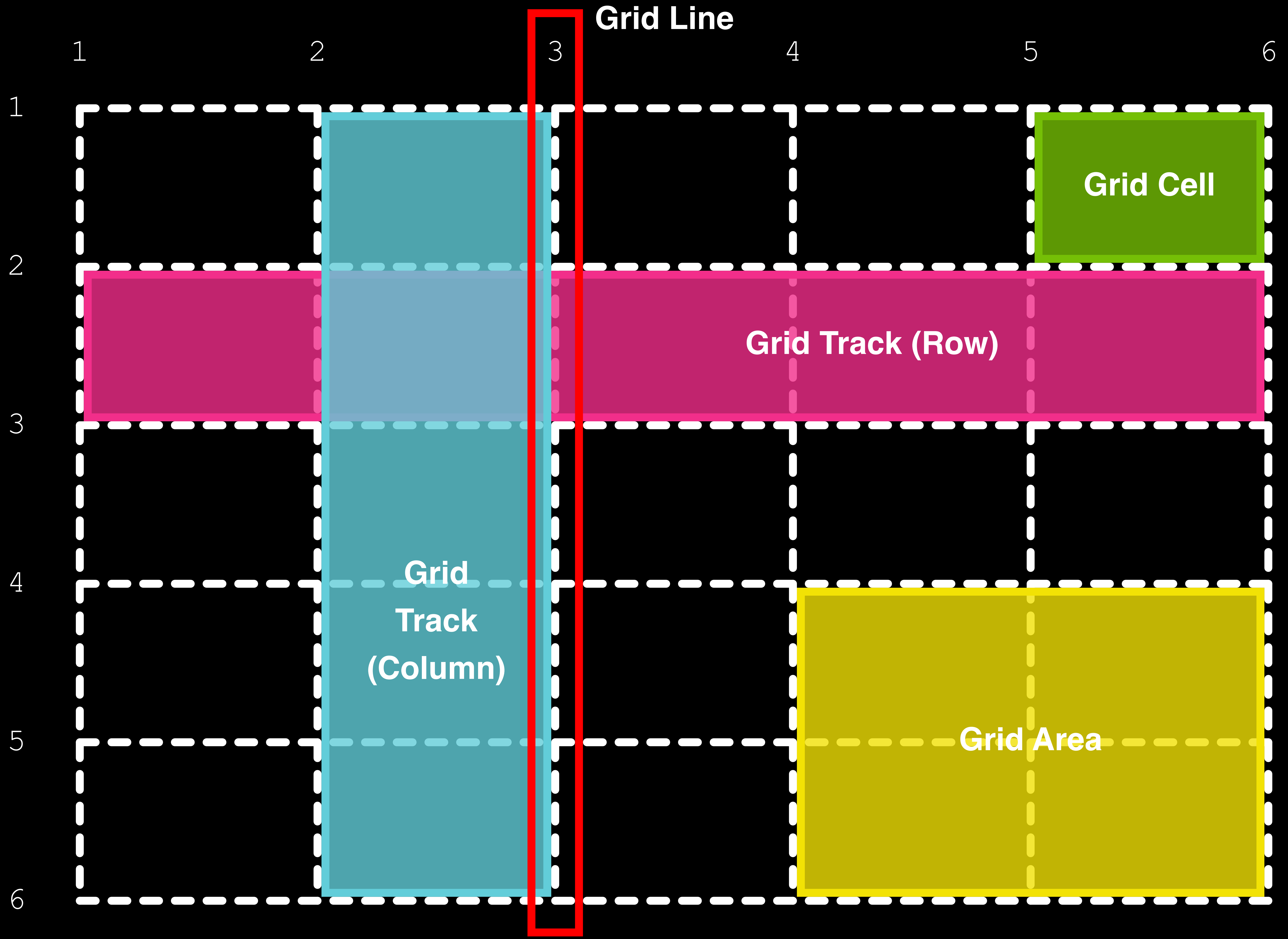
CSS GRID

I know, another grid system…
Unlike flexbox, CSS grid works in 2 dimensions; like flexbox, it is applied to a parent to affect its children.

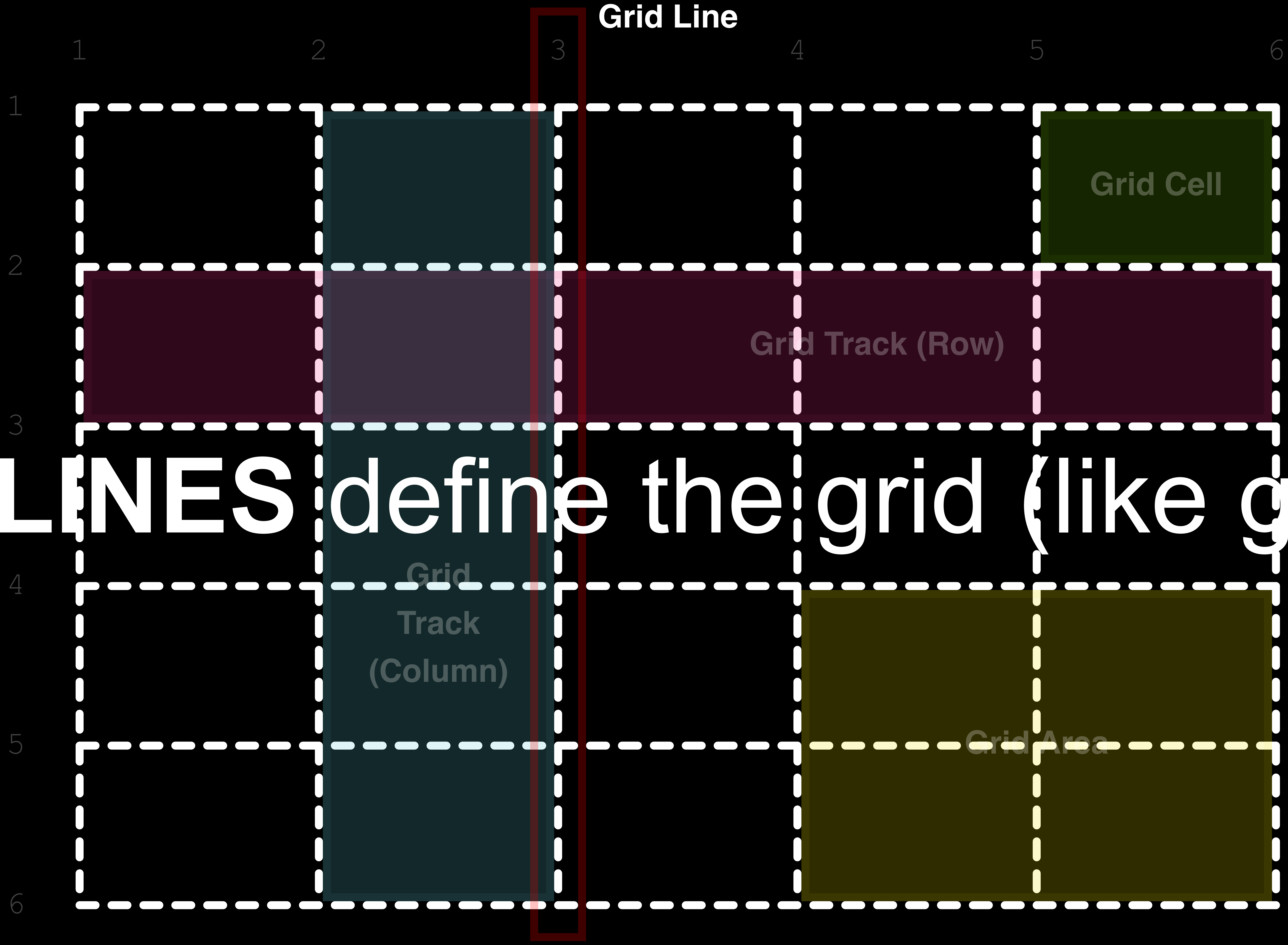
CSS GRID

We apply it with `display: grid;`
(or `inline-grid`)
and can use it in all sorts of powerful ways that more closely mirror the way you may think about print design.

CSS GRID

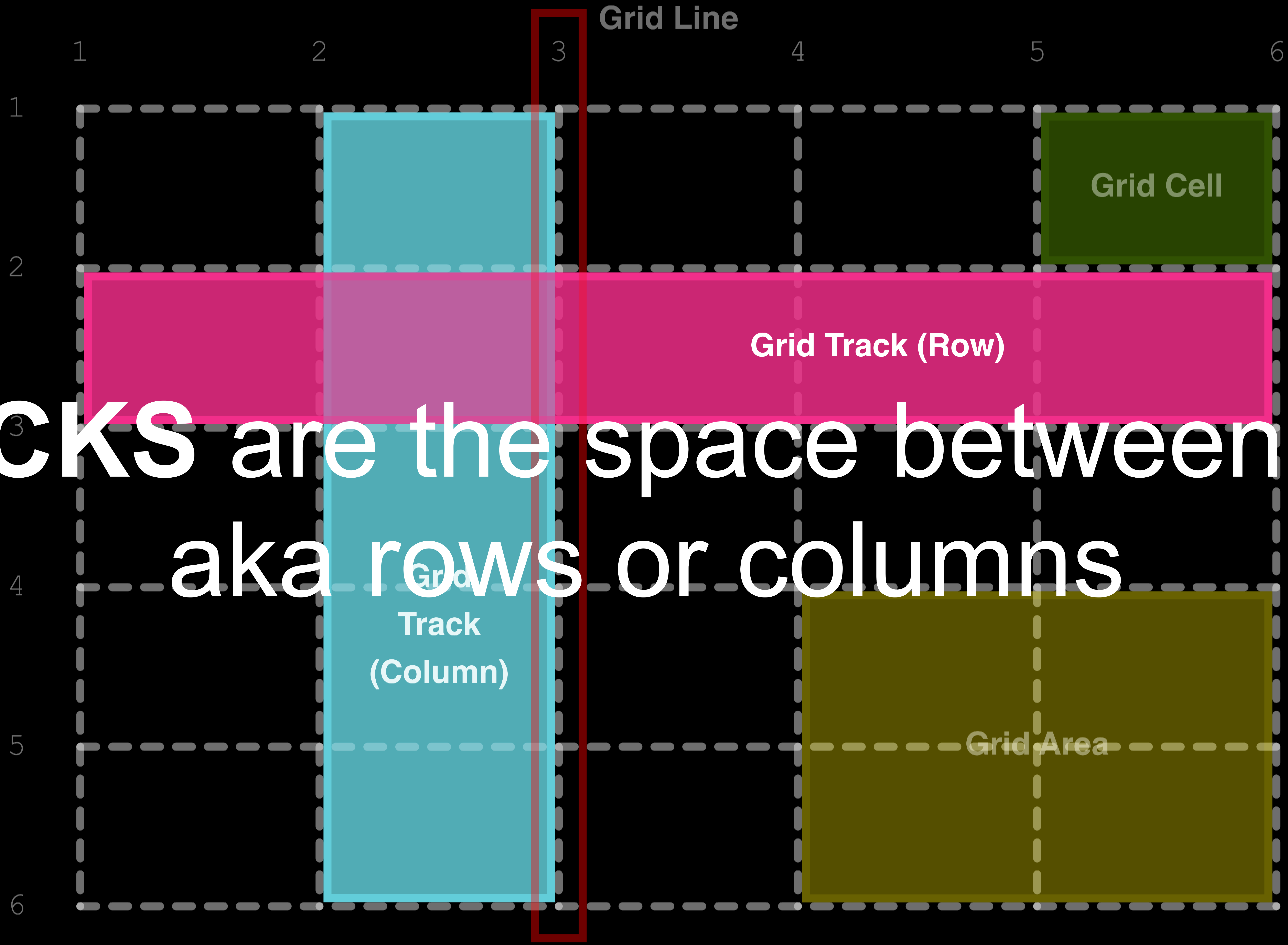

Grid Line
1
2
3
4
5
6
1
2
3
4
5
6
Grid Cell
Grid Track (Row)
Grid Track (Column)
Grid Area

CSS GRID



GRID LINES define the grid (like gutters)

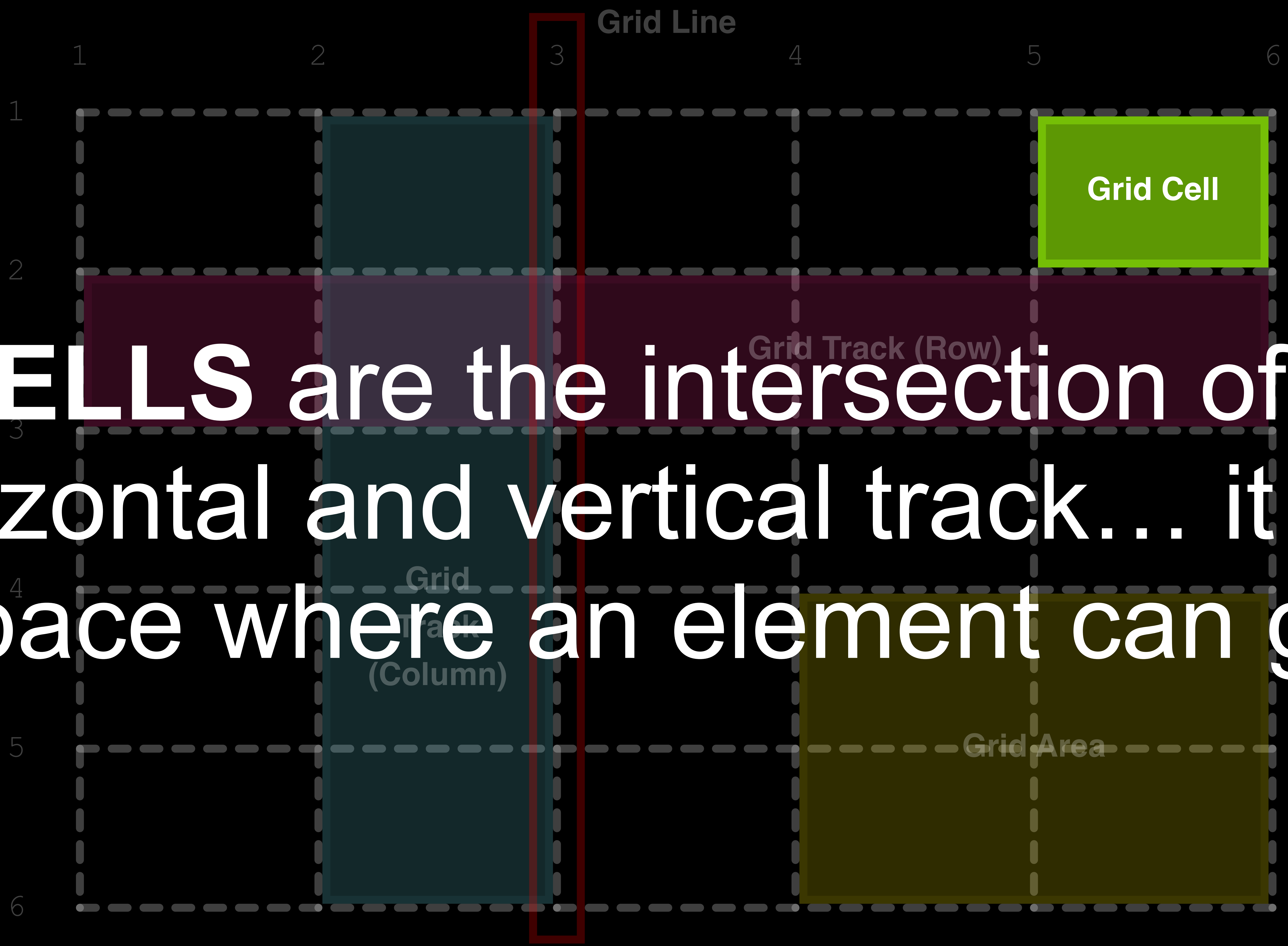
CSS GRID



TRACKS are the space between lines

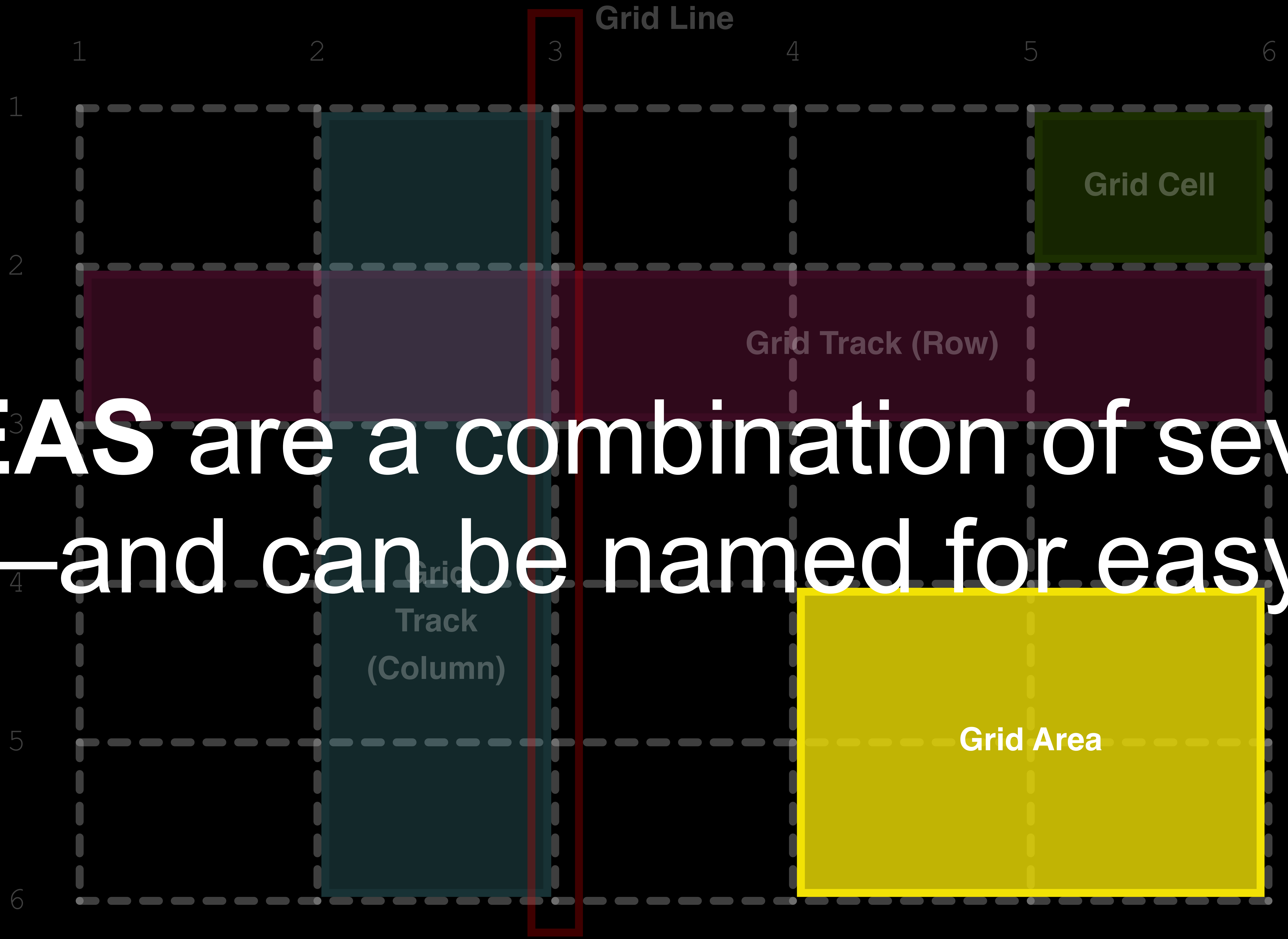
aka rows or columns

CSS GRID



CELLS are the intersection of a horizontal and vertical track… it is a space where an element can go

CSS GRID



AREAS are a combination of several cells—and can be named for easy use.

NEW UNITS + FUNCTIONS

`fr`	refers to a fraction of space available in a grid container.
`min-content`	the *intrinsic minimum width* of a piece of content—e.g. the longest word in a paragraph.
`max-content`	the *intrinsic maximum width* of a piece of content—e.g. the whole length of a line of text.
`fit-content`	a combo of the min- and max-content units, that will automatically use available space.

NEW UNITS + FUNCTIONS

fr

```
.two-thirds-one-third{
   display: grid;
   grid-template-columns: 2fr 1fr;
}
```

min-content

```
.narrow-sidebar{
   display: grid;
   grid-template-columns: 1fr min-content;
}
```

max-content

```
.wide-sidebar{
   display: grid;
   grid-template-columns: 1fr max-content;
}
```

fit-content

```
.fit-sidebar{
   display: grid;
   grid-template-columns: 1fr fit-content;
}
```

NEW UNITS + FUNCTIONS

`minmax()`	A function that defines a range for a track, setting a minimum and maximum length together.
`repeat()`	This function repeats a track list, so you don’t have to write it over and over.

NEW UNITS + FUNCTIONS

minmax()

```
.flexible-sidebar {
   display: grid;
   grid-template-columns: 1fr minmax(200px, 400px);
}
```

repeat()

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

.also-twelve-columns {
   display: grid;
   grid-template-columns: repeat(12, 1fr);
}
```

PARENT PROPERTIES

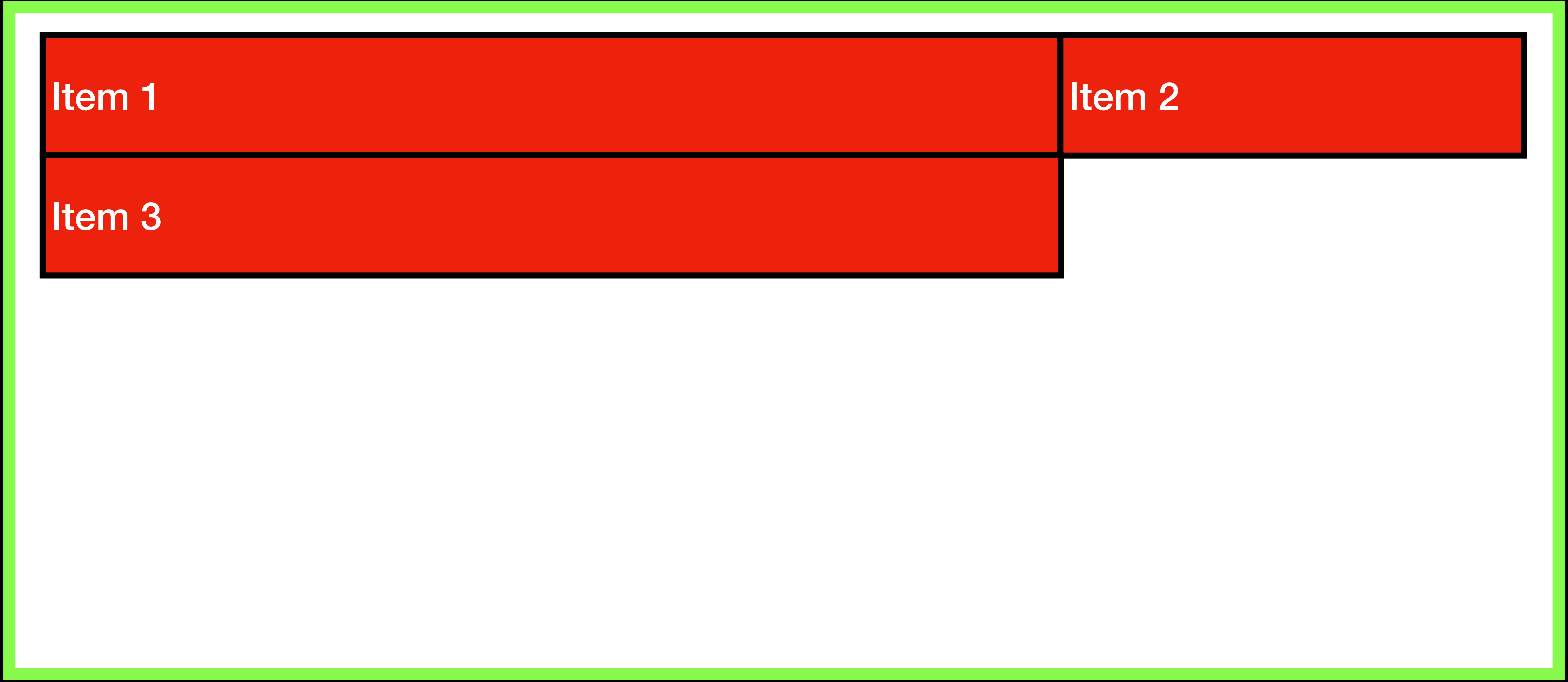
You’ve set `display: grid;` now what? You have to then declare a set of rows, columns, or both using `grid-template-columns` or `grid-template-rows`

PARENT PROPERTIES

These properties are then followed by a track list, defining the size of each track.

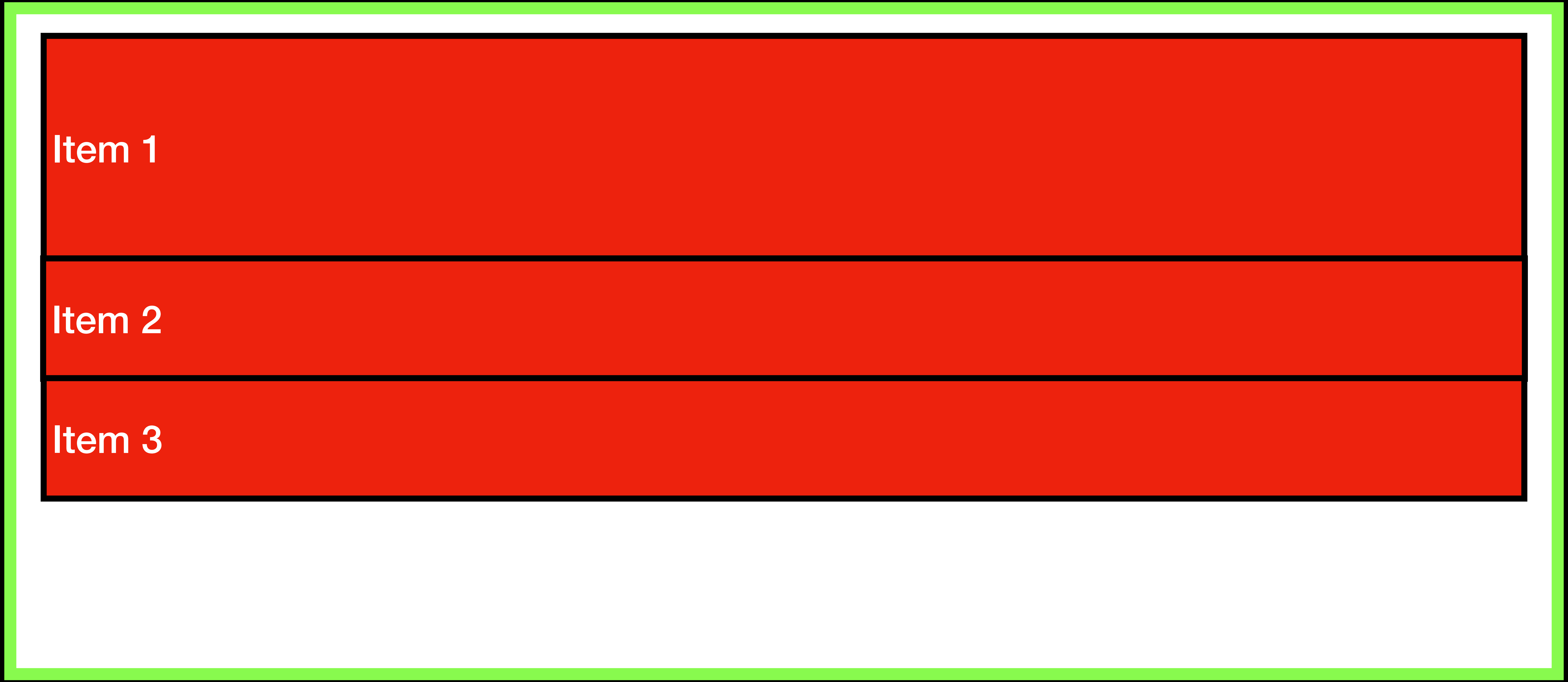
PARENT PROPERTIES

```
display: grid;
grid-template-columns: 2fr 1fr;
```



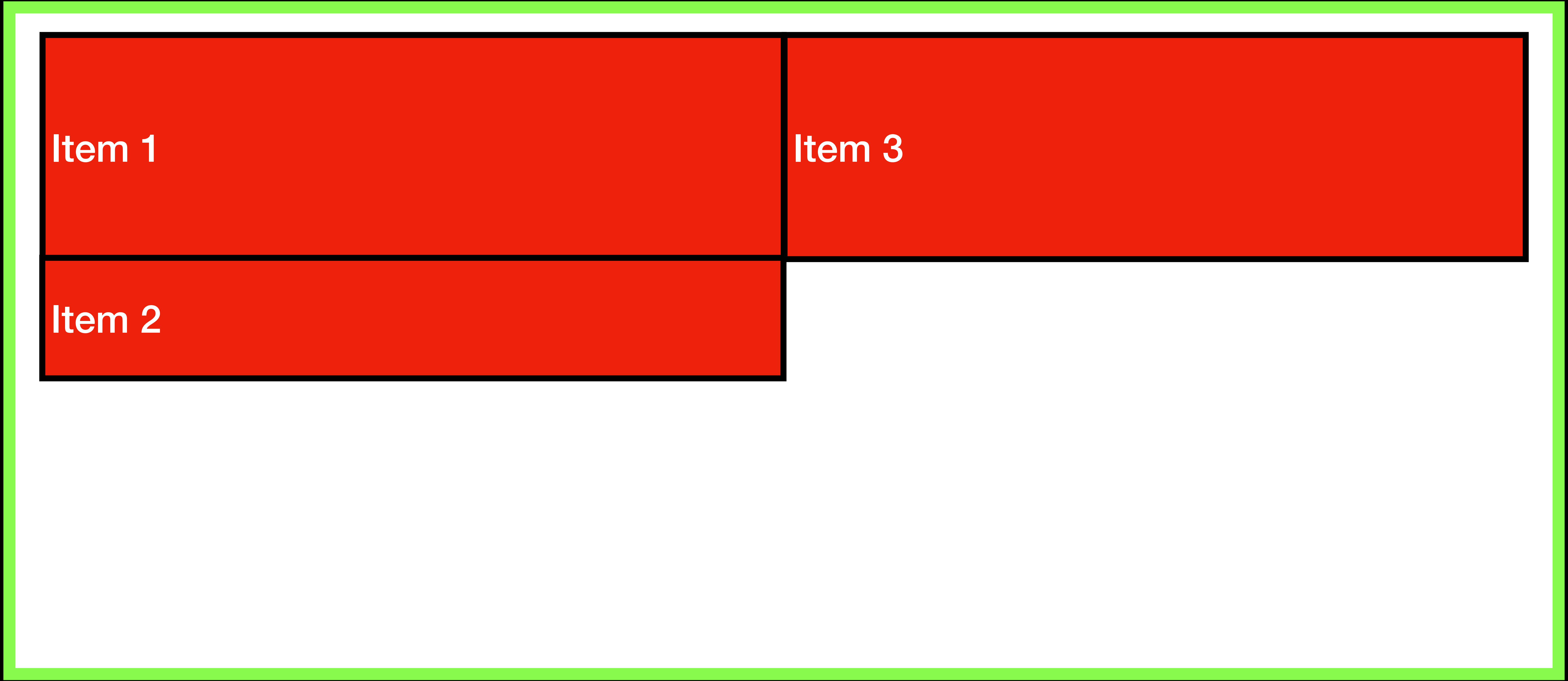
PARENT PROPERTIES

```
display: grid;
grid-template-rows: 2fr 1fr;
```



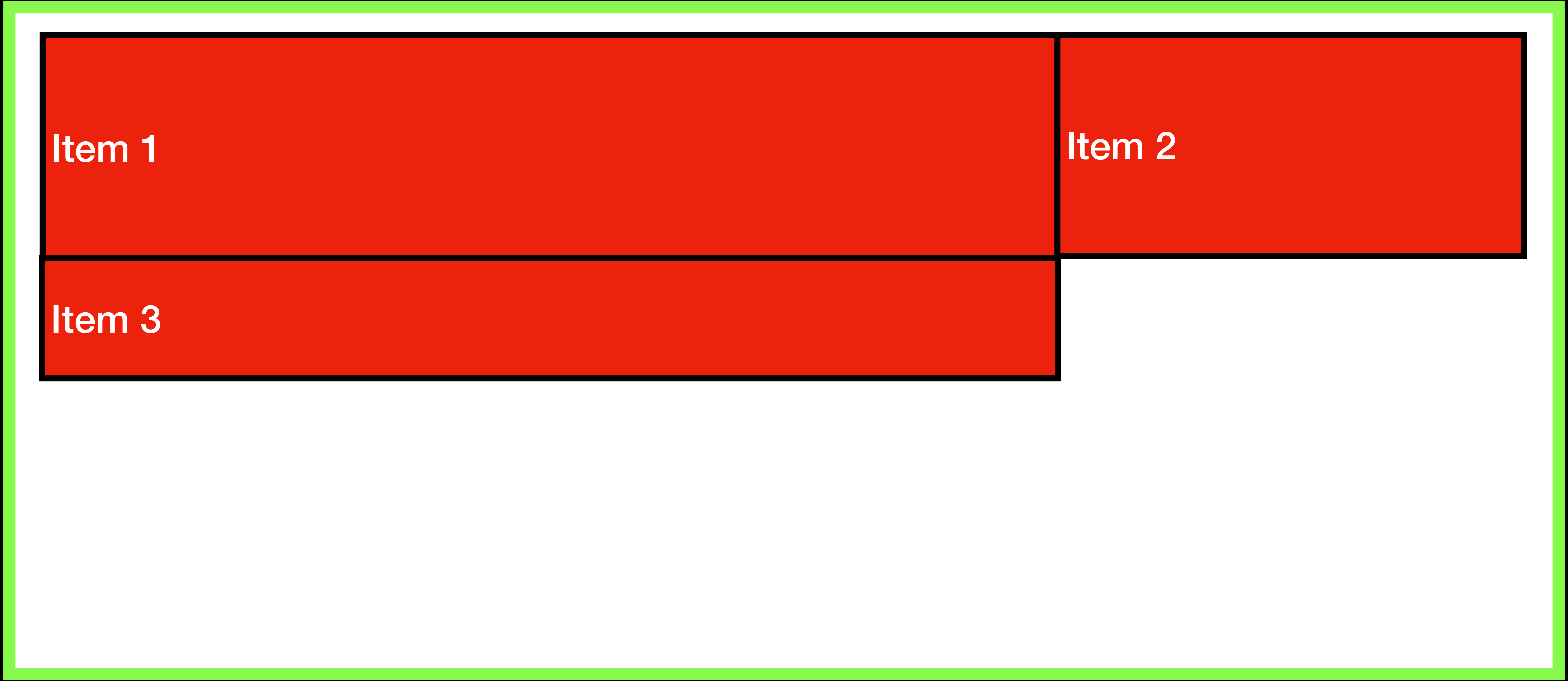
PARENT PROPERTIES

```
display: grid;
grid-template-rows: 2fr 1fr;
grid-auto-flow: column;
```



PARENT PROPERTIES

```
display: grid;
grid-template-rows: 2fr 1fr;
grid-template-columns: 2fr 1fr;
```

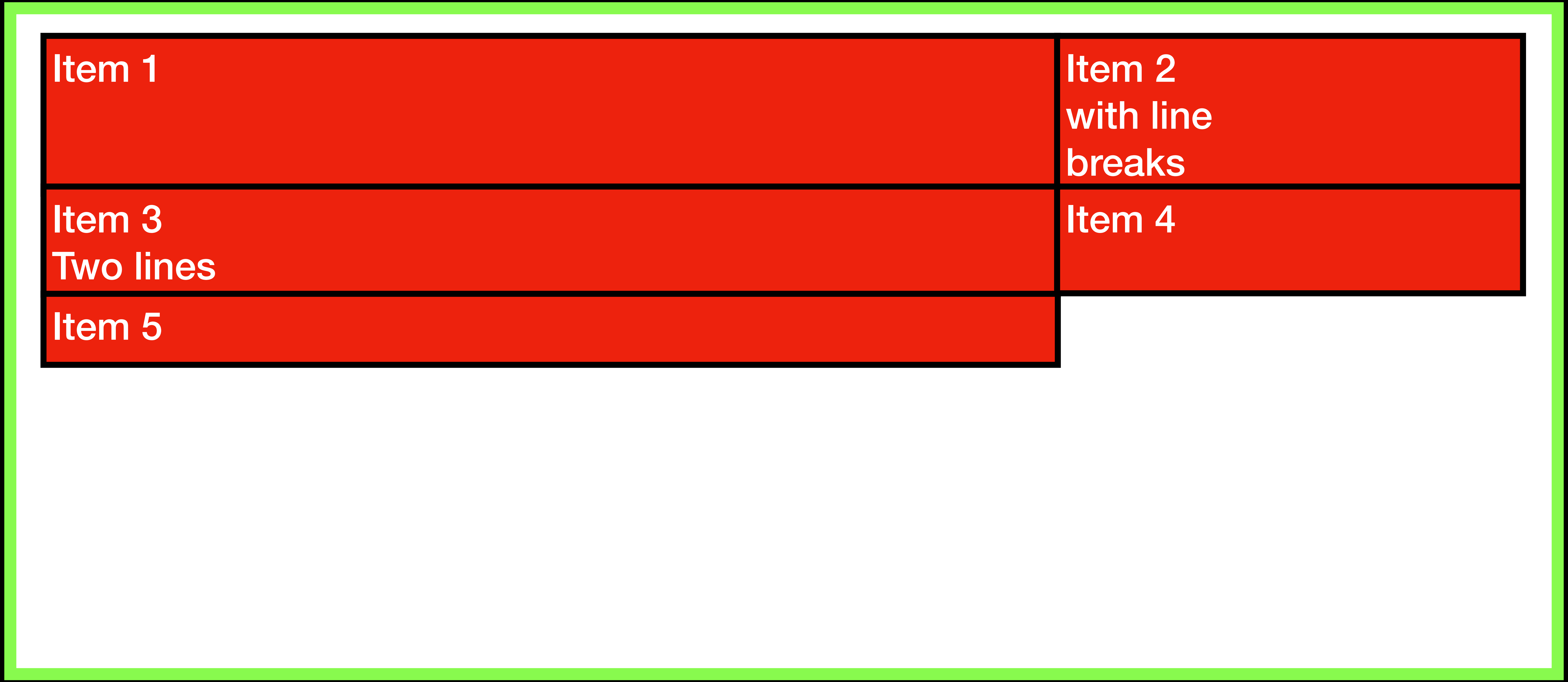


PARENT PROPERTIES

In many cases, you only need to set your columns, and your rows will flow naturally. This is the *implicit grid*.

PARENT PROPERTIES

```
display: grid;
grid-template-columns: 2fr 1fr;
```

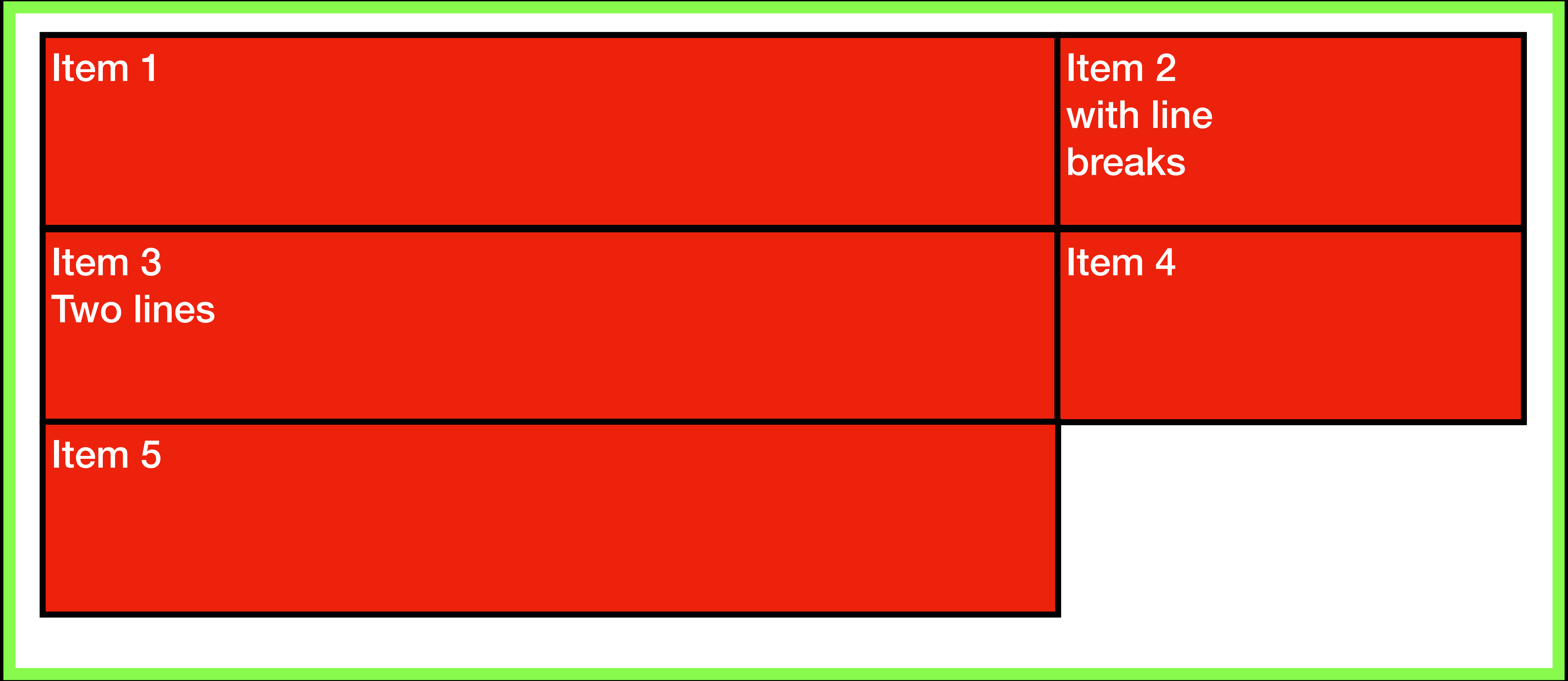


PARENT PROPERTIES

By default, these implicit grid tracks are sized to their content (`auto`), but you can specify a size.

PARENT PROPERTIES

```
display: grid;
grid-auto-rows: 80px; /*row height*/
grid-template-columns: 2fr 1fr;
```

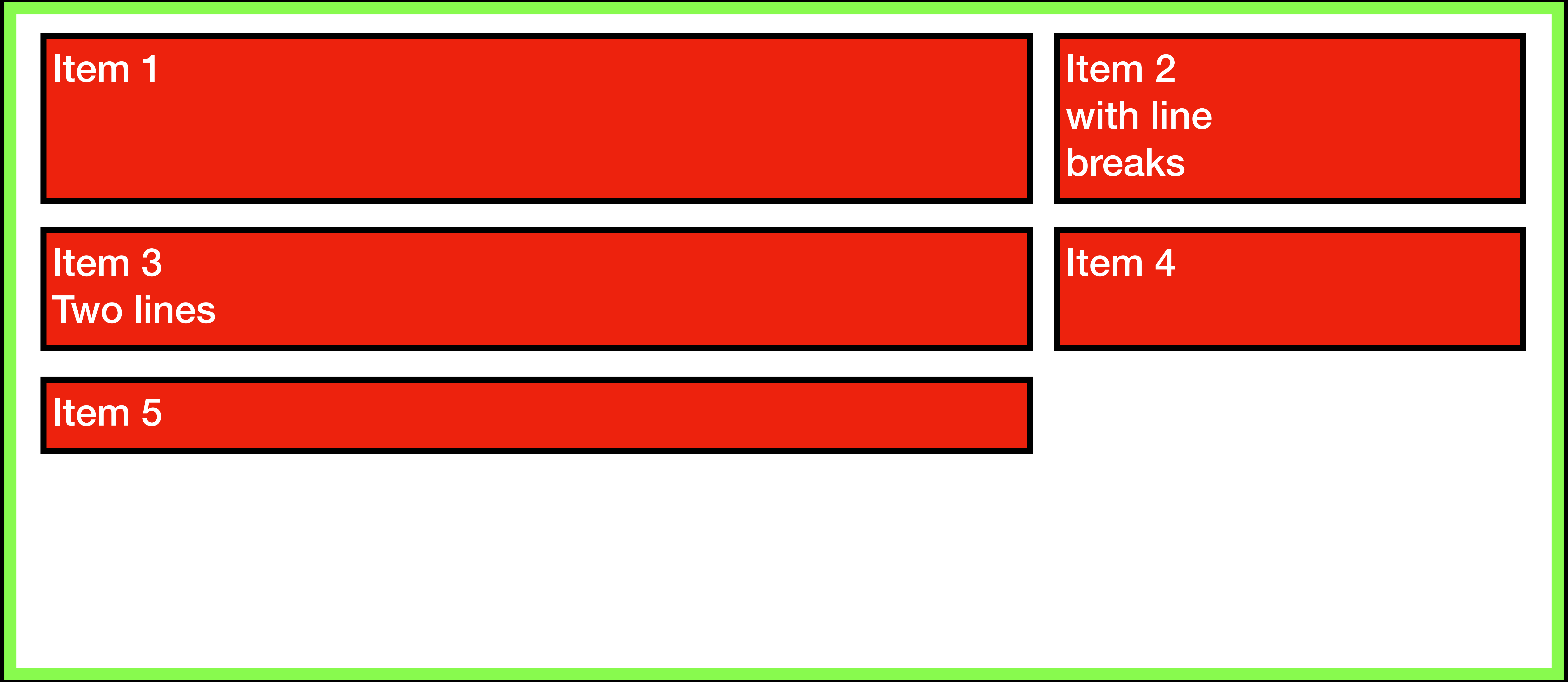


PARENT PROPERTIES

You can control the gap between tracks using the same syntax as we used in flexbox.

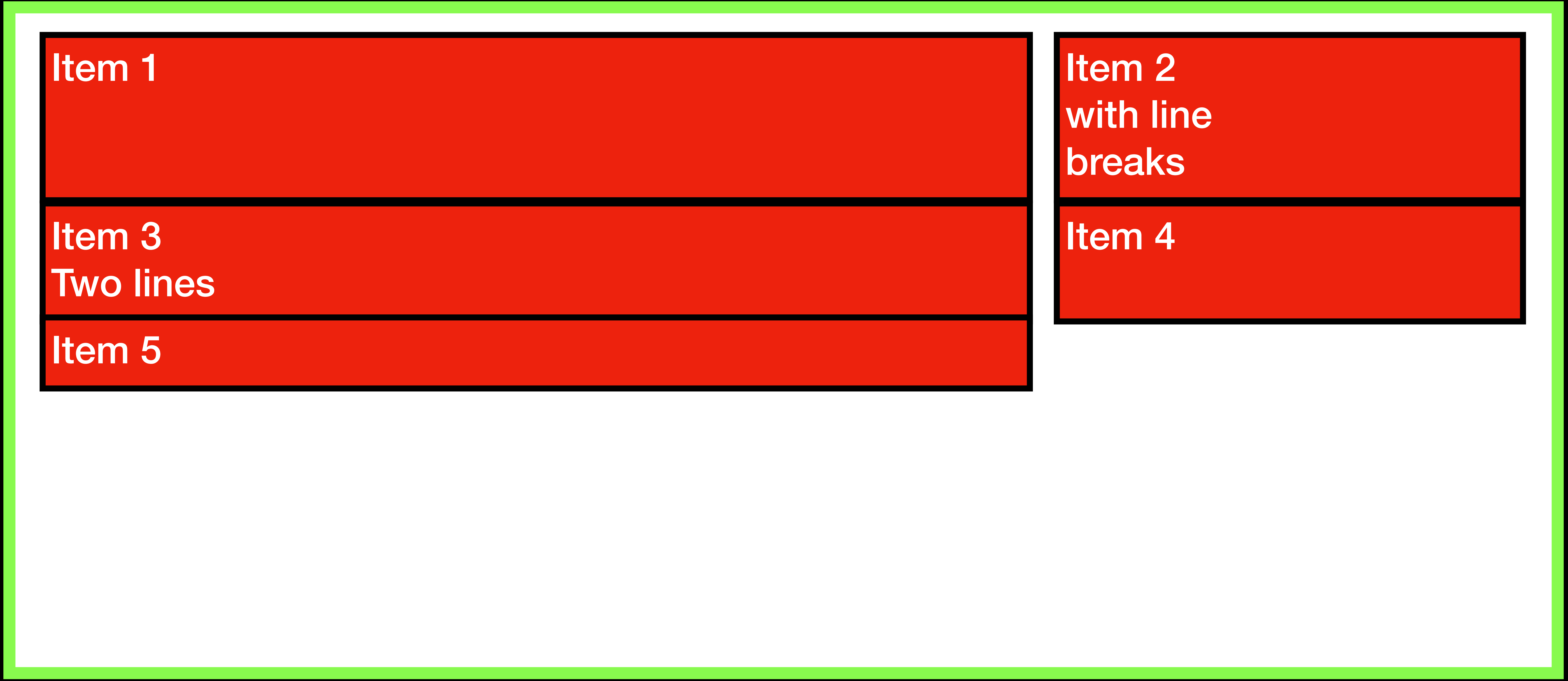
PARENT PROPERTIES

```
display: grid;
grid-template-columns: 2fr 1fr;
gap: 10px;
```



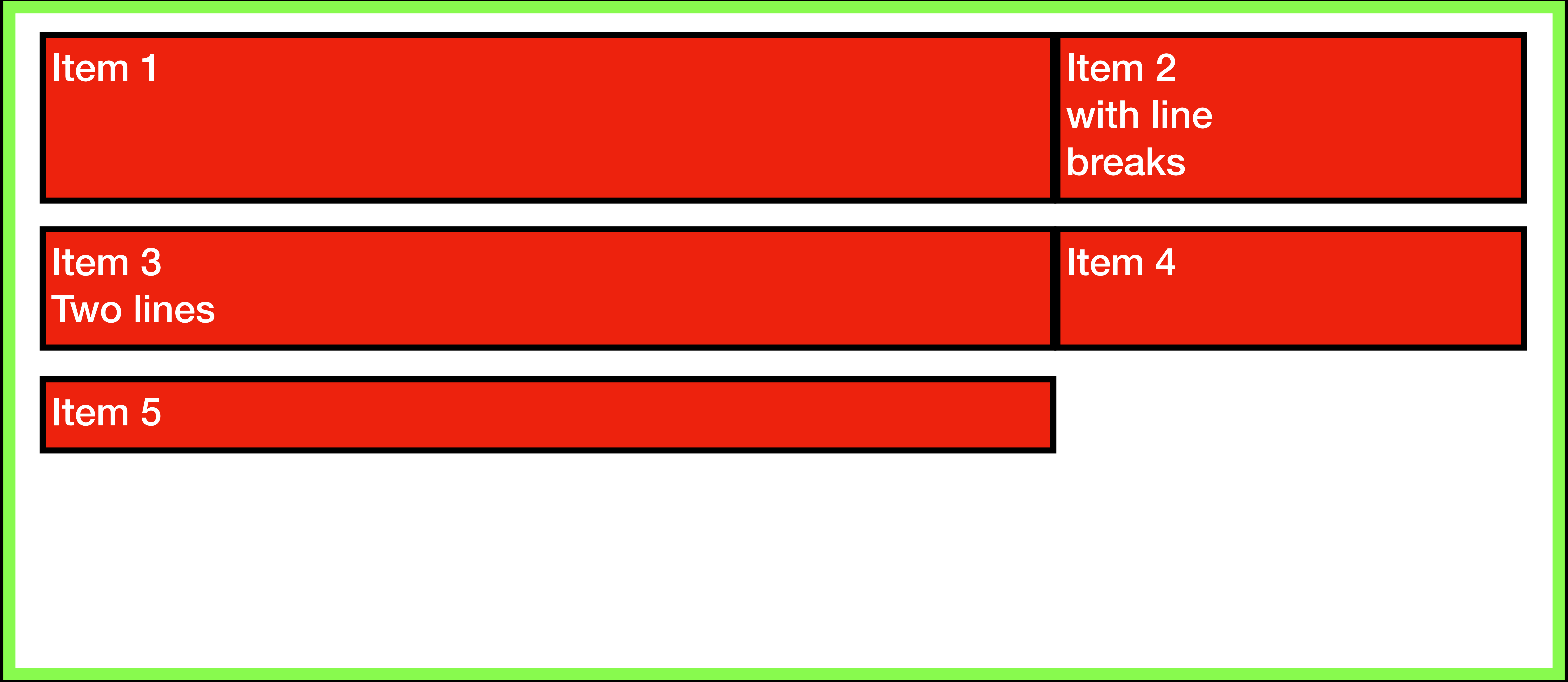
PARENT PROPERTIES

```
display: grid;
grid-template-columns: 2fr 1fr;
column-gap: 10px;
```



PARENT PROPERTIES

```
display: grid;
grid-template-columns: 2fr 1fr;
row-gap: 10px;
```

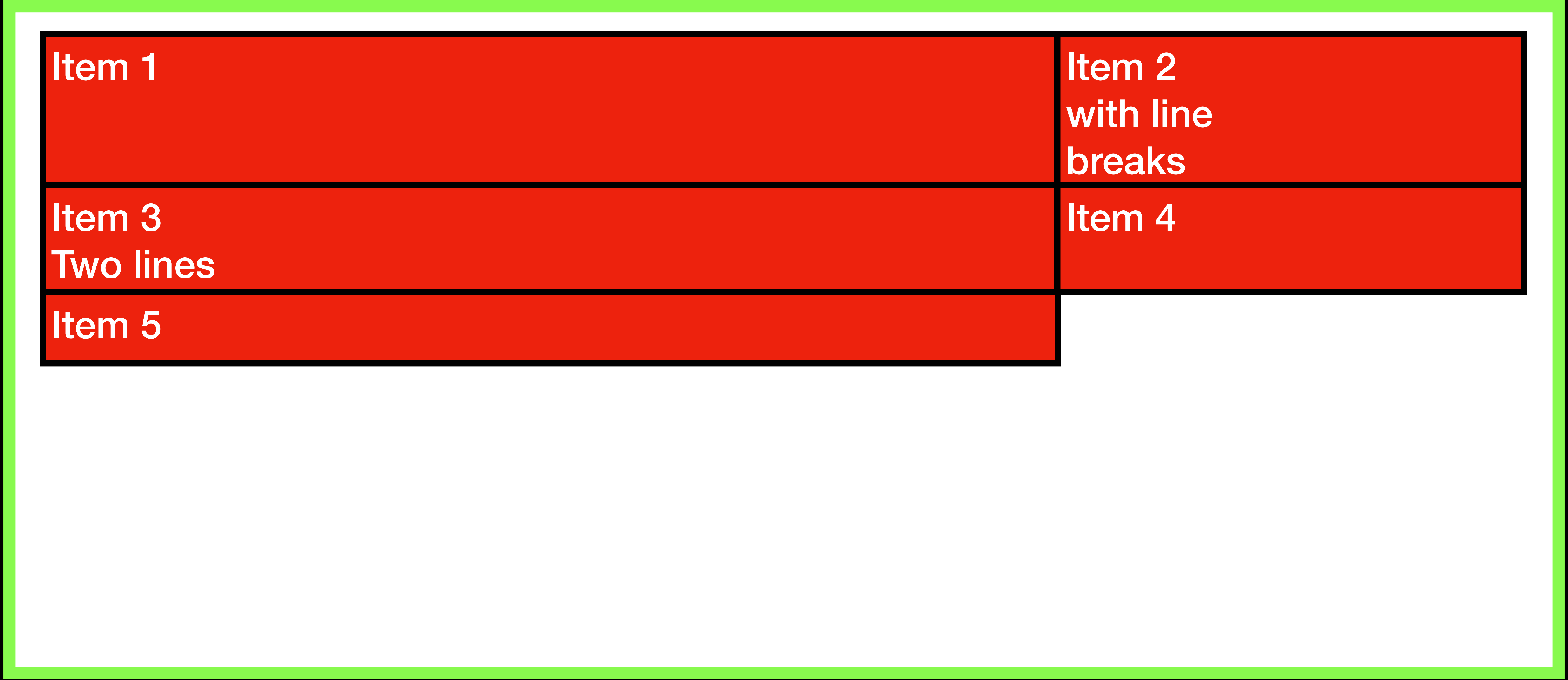


PARENT PROPERTIES

We can also justify and align in grid—unlike flexbox, justify is always horizontal, and align is always vertical.

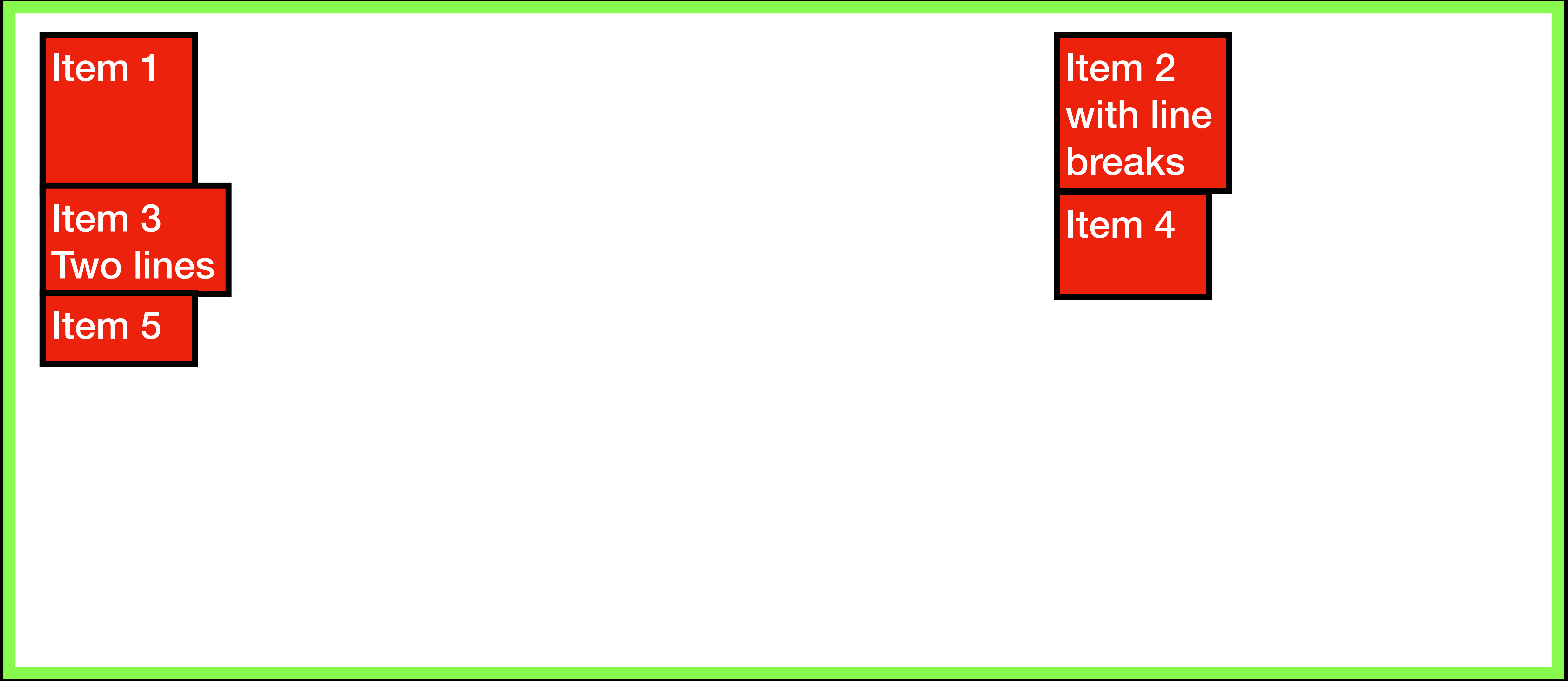
PARENT PROPERTIES: JUSTIFY

```
display: grid;
grid-template-columns: 2fr 1fr;
justify-items: stretch; /*default*/
```



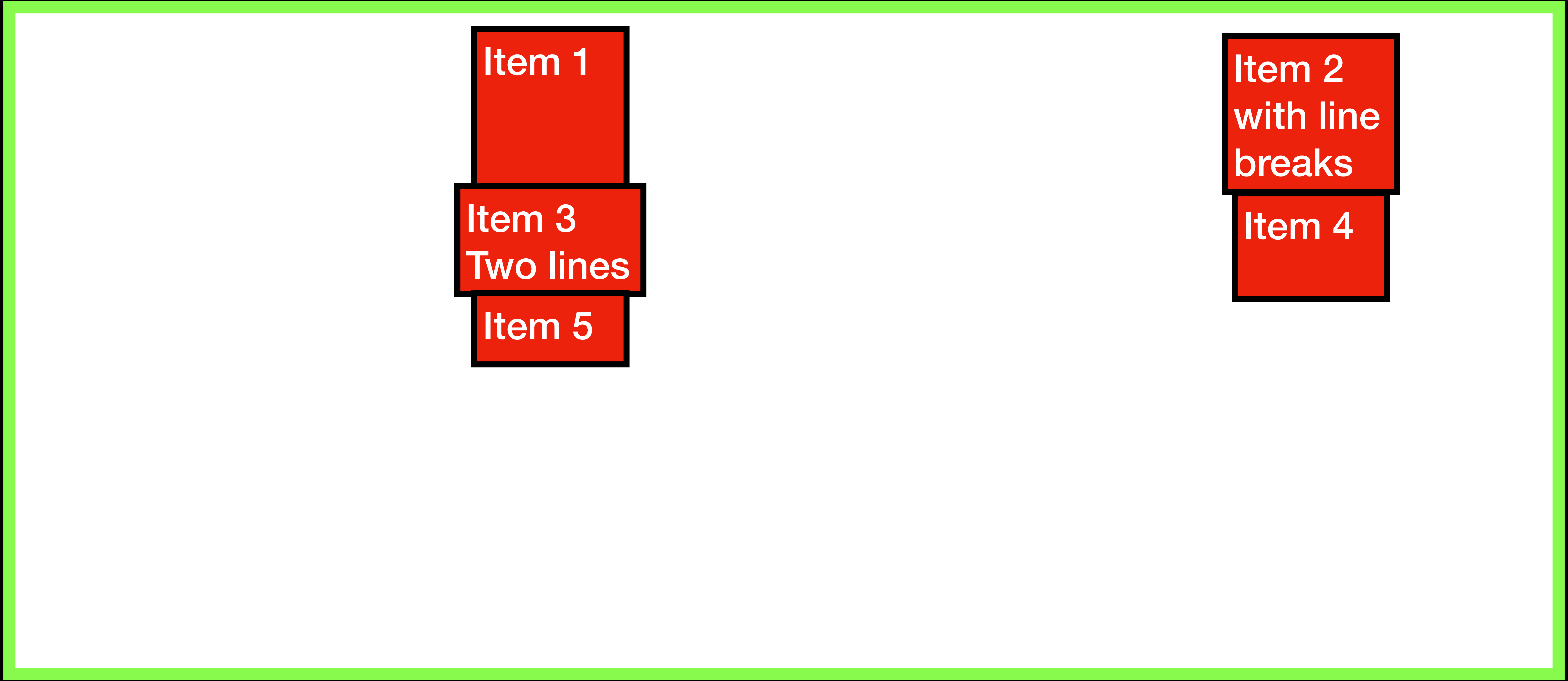
PARENT PROPERTIES: JUSTIFY

```
display: grid;
grid-template-columns: 2fr 1fr;
justify-items: start;
```



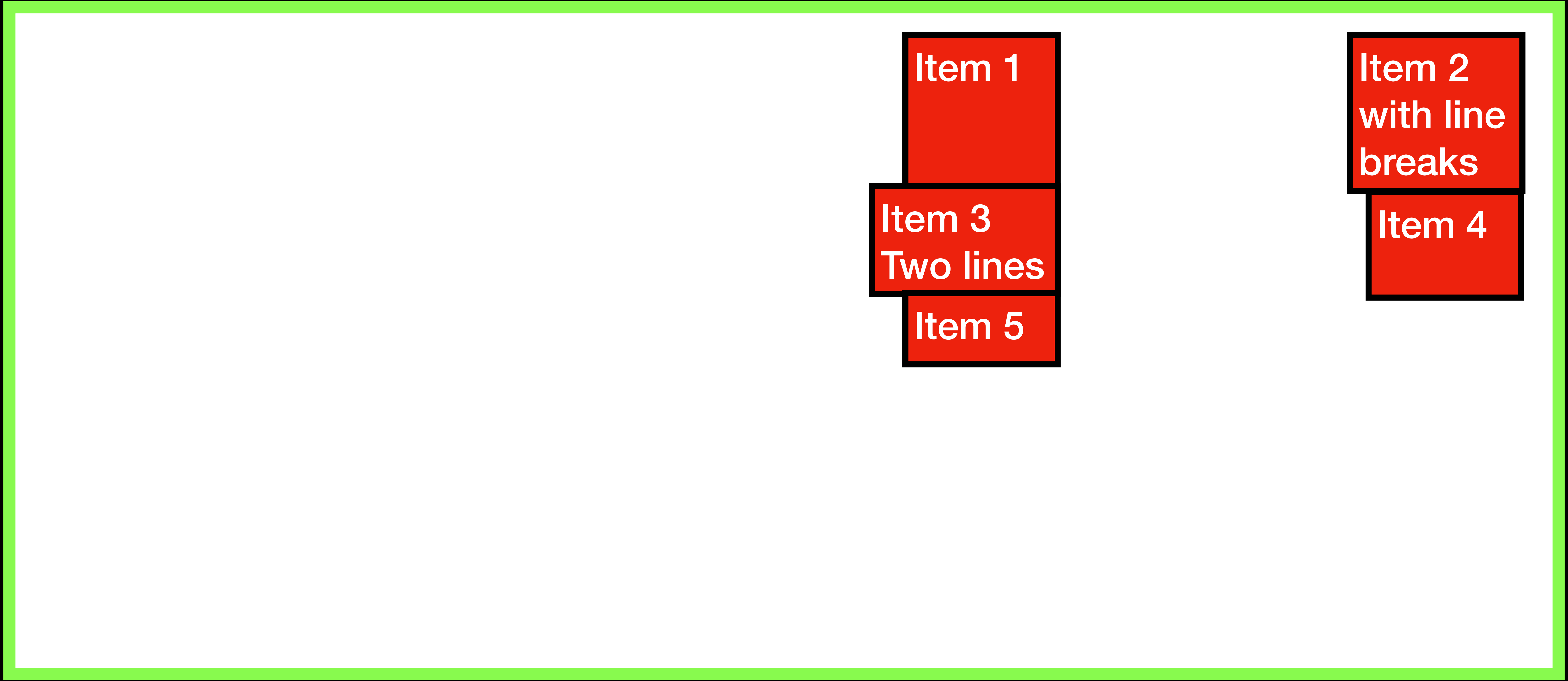
PARENT PROPERTIES: JUSTIFY

```
display: grid;
grid-template-columns: 2fr 1fr;
justify-items: center;
```



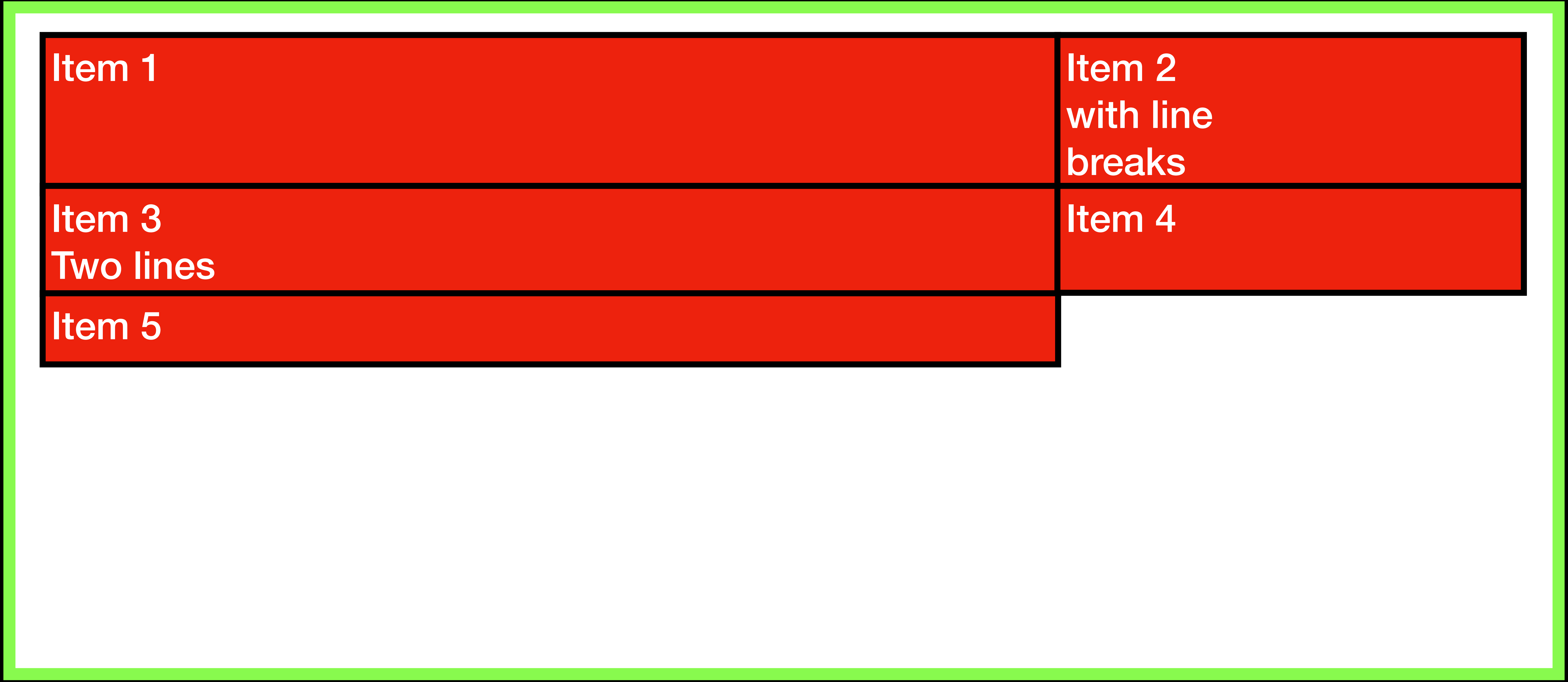
PARENT PROPERTIES: JUSTIFY

```
display: grid;
grid-template-columns: 2fr 1fr;
justify-items: end;
```



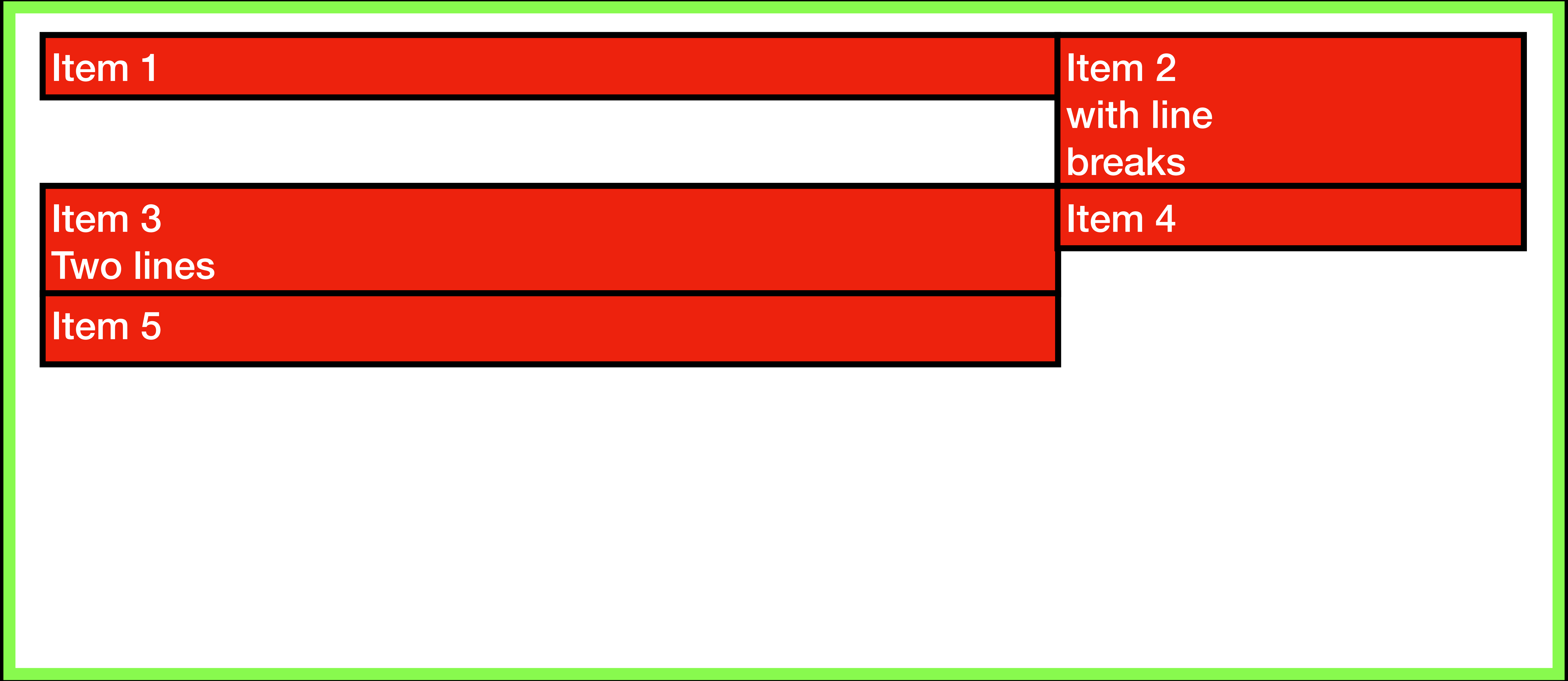
PARENT PROPERTIES: ALIGN

```
display: grid;
grid-template-columns: 2fr 1fr;
align-items: stretch; /*default*/
```



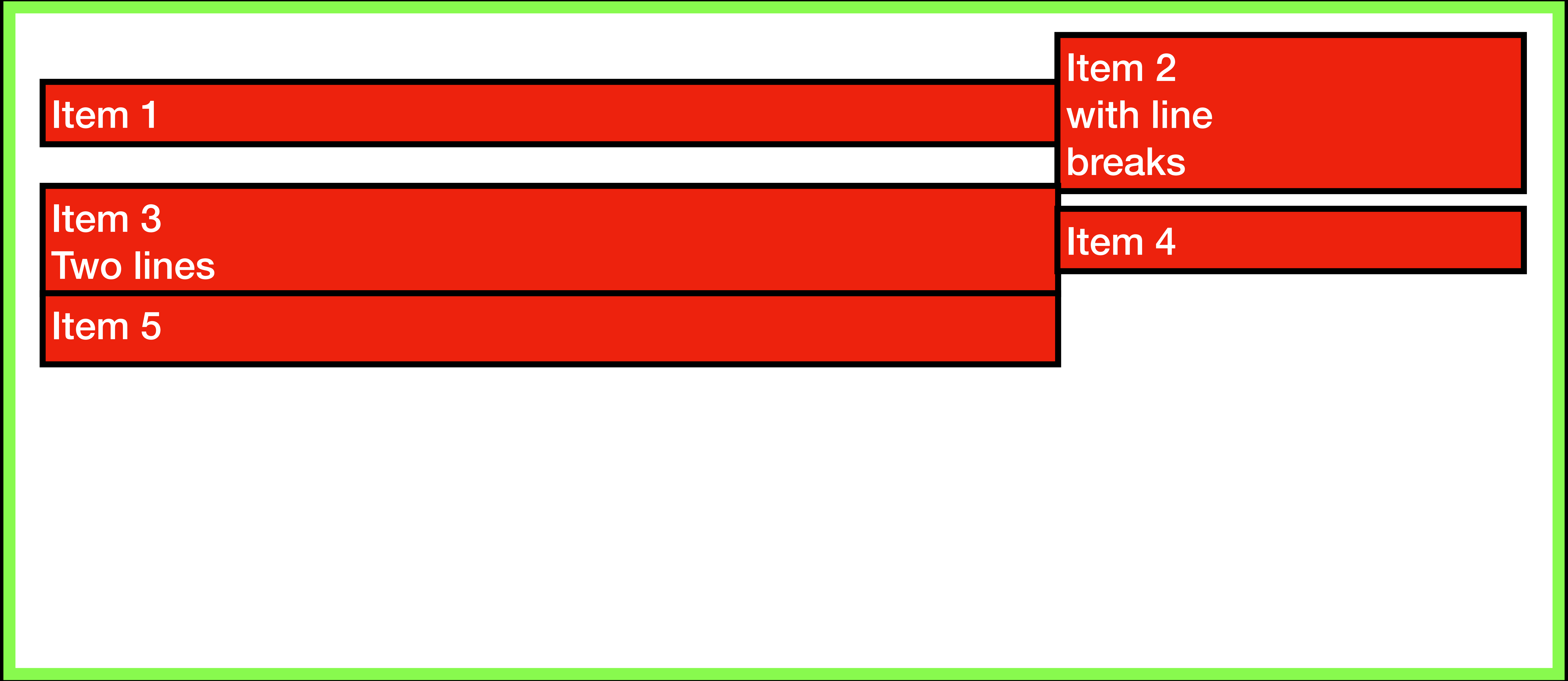
PARENT PROPERTIES: ALIGN

```
display: grid;
grid-template-columns: 2fr 1fr;
align-items: start;
```



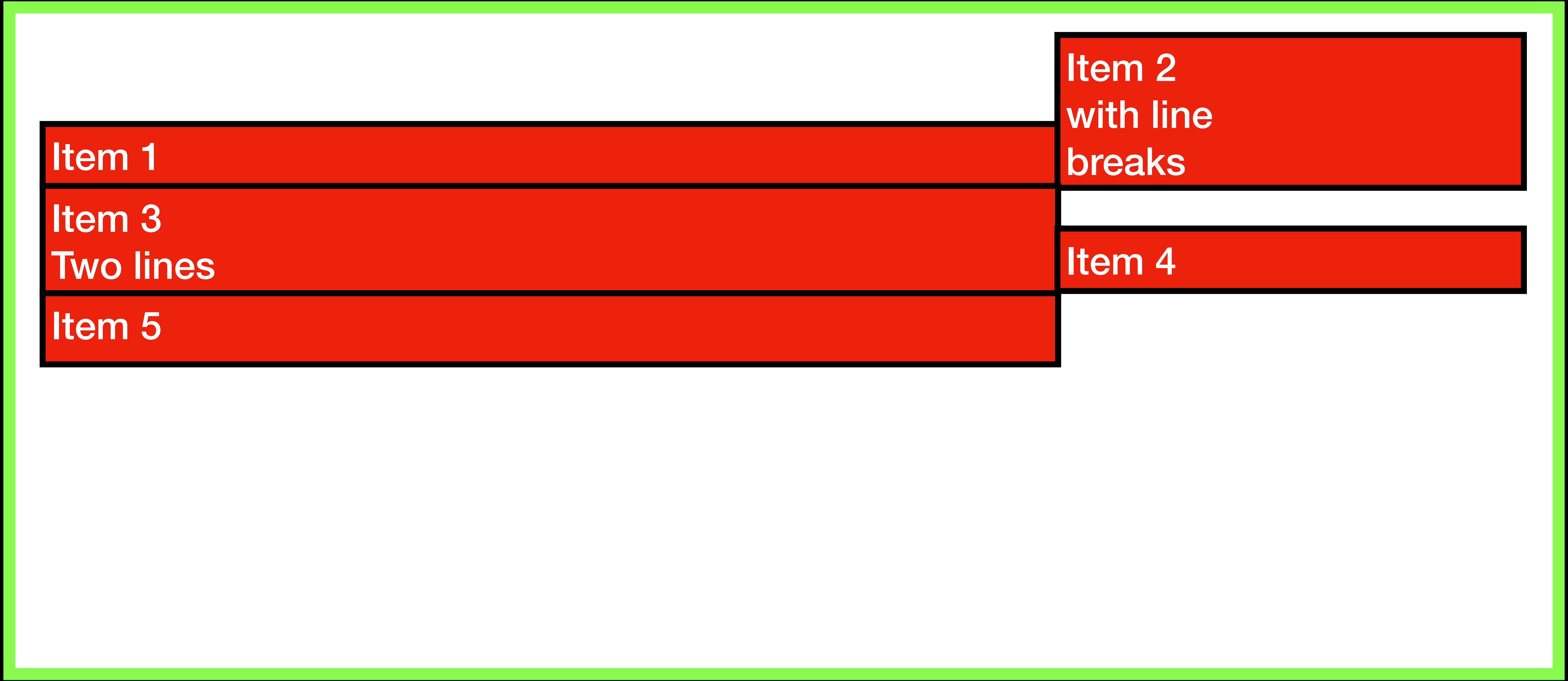
PARENT PROPERTIES: ALIGN

```
display: grid;
grid-template-columns: 2fr 1fr;
align-items: center;
```



PARENT PROPERTIES: ALIGN

```
display: grid;
grid-template-columns: 2fr 1fr;
align-items: end;
```

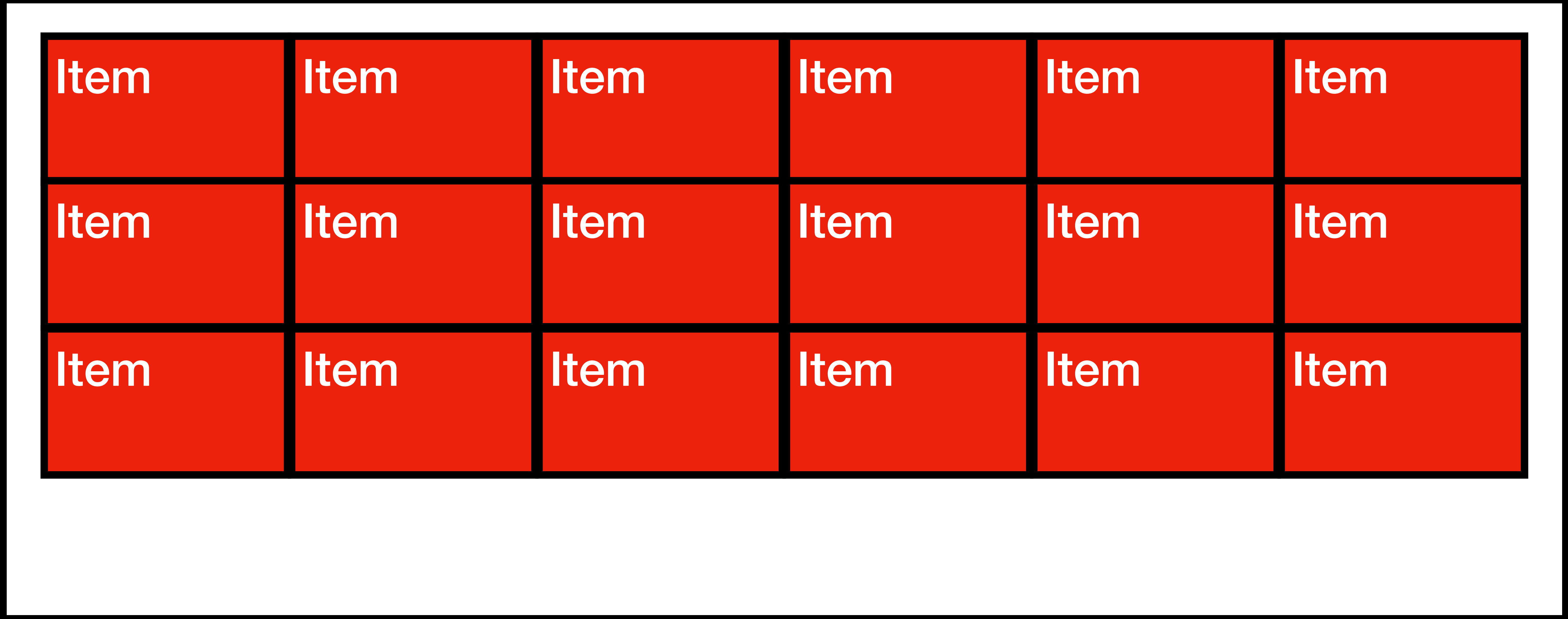


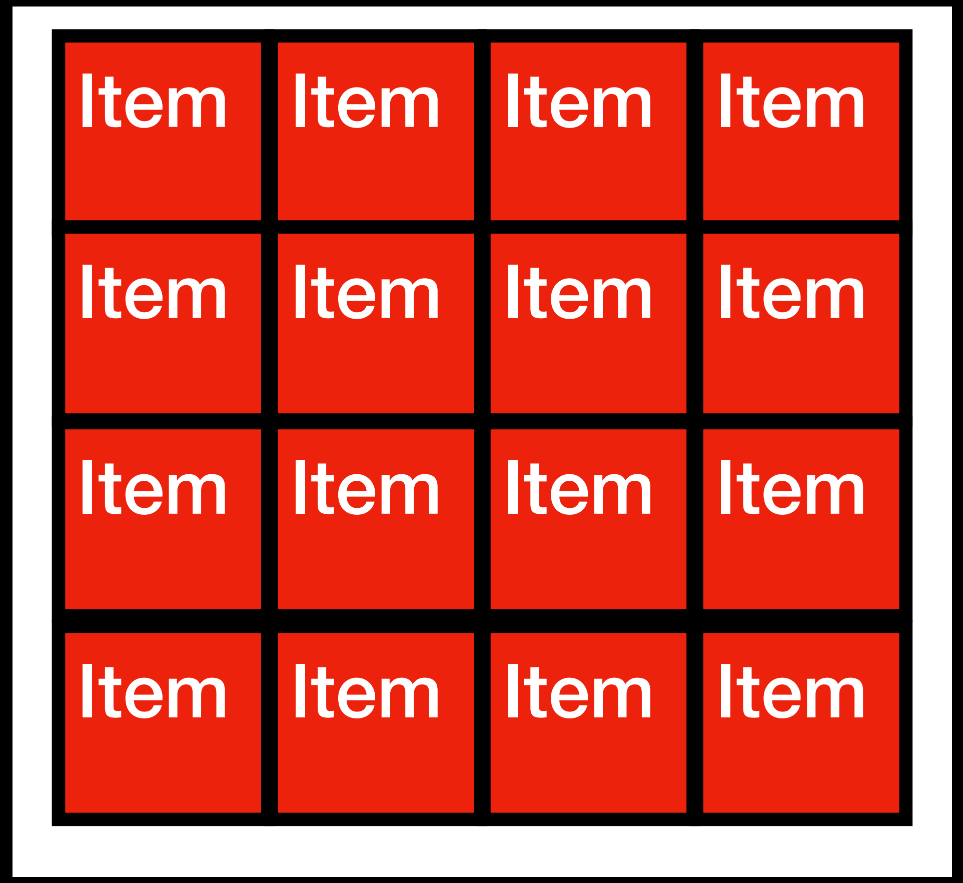
PARENT PROPERTIES: REPEAT()

You can use the repeat function to make even column grids easily, and adjust with media queries.

PARENT PROPERTIES: REPEAT

```
…
section{
  --columns: 4;

  display: grid;
  grid-template-columns:
  (var(--columns), 1fr);
}

@media(min-width:400px){
  --columns: 6;
}
…
```


Item Item Item Item Item Item
Item Item Item Item Item Item
Item Item Item Item Item Item



PARENT PROPERTIES: AUTOFILL/FIT

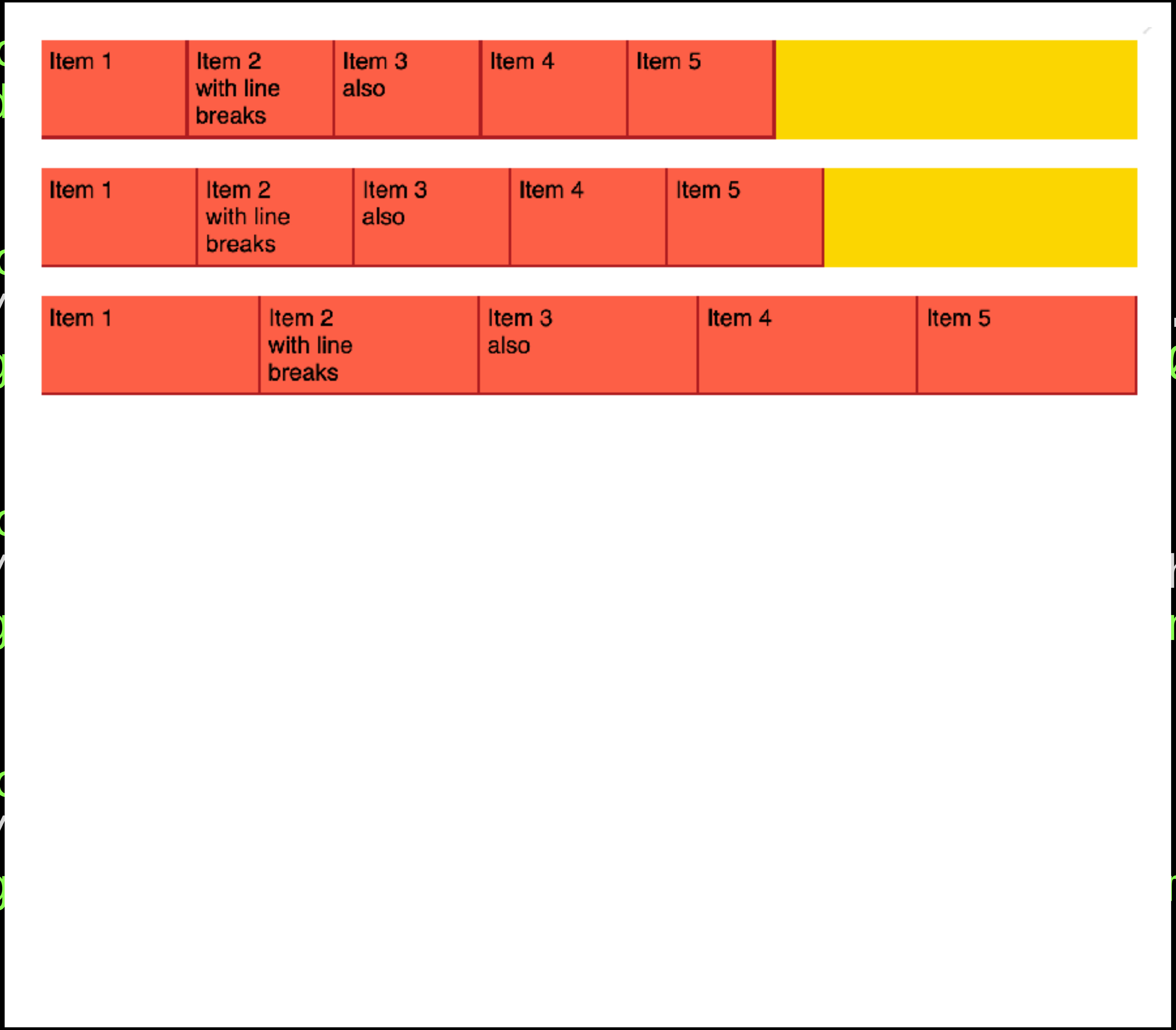
Using the repeat function, you can also autofit or autofill your columns/rows—essentially building in inherent and automatic responsiveness.

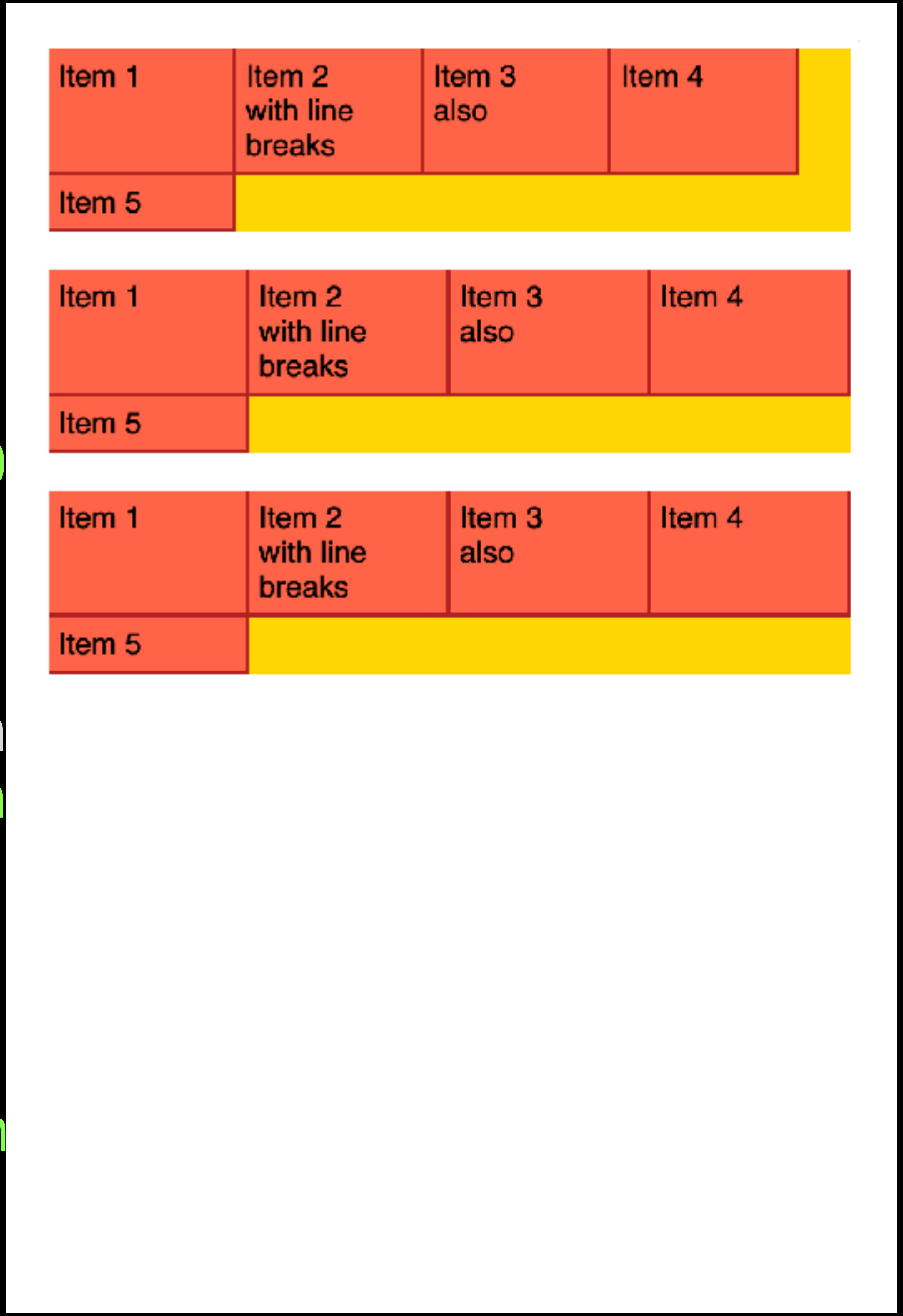
PARENT PROPERTIES: AUTOFILL/FIT

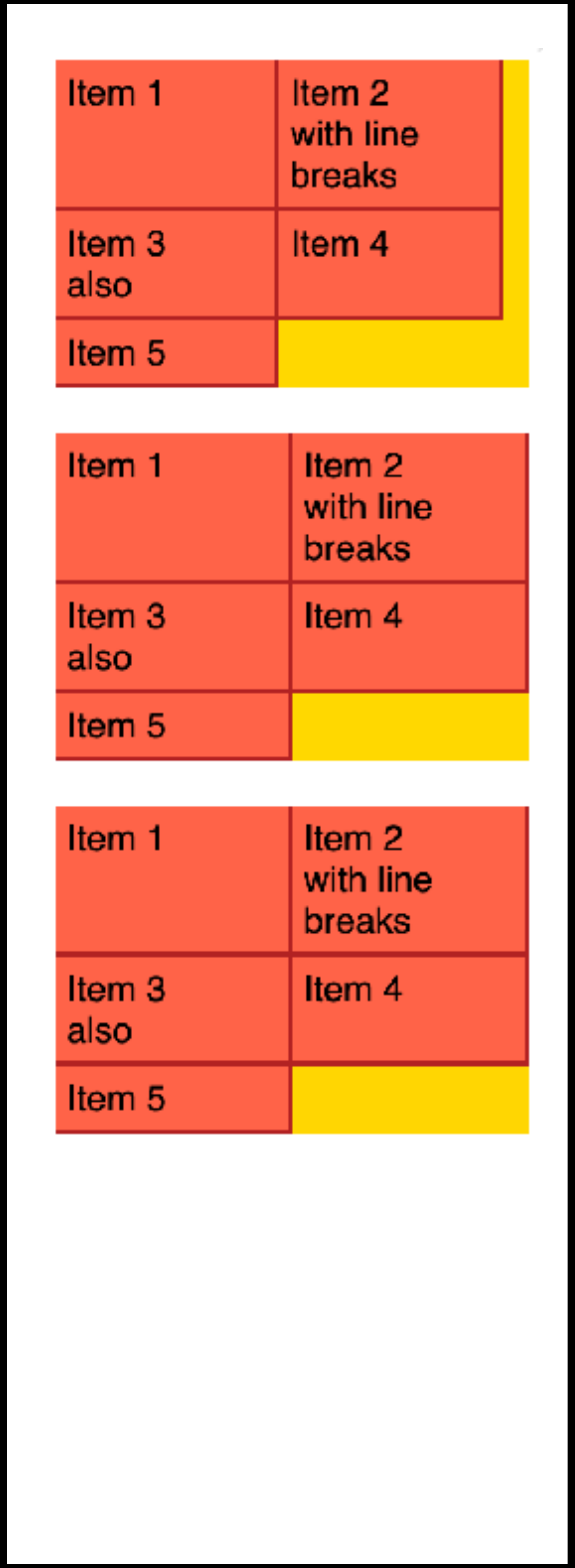
```css
section {
  display: grid;
}

section:first-child {
  /* Fill as many `100px` columns as possible. */
  grid-template-columns: repeat(auto-fill, 100px);
}

section:nth-child(2) {
  /* Add flexible `1fr` columns when larger than `100px`. */
  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
}

section:last-child {
  /* Same, but always fit them to the row! */
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
```

PARENT PROPERTIES: AUTOFILL/FIT

```
sec
  d
}

sec
  /                                                  .
  g                                                  0
}

sec
  /                                                  h
  g                                                  n
}

sec
  /
  g                                                  m
}
```





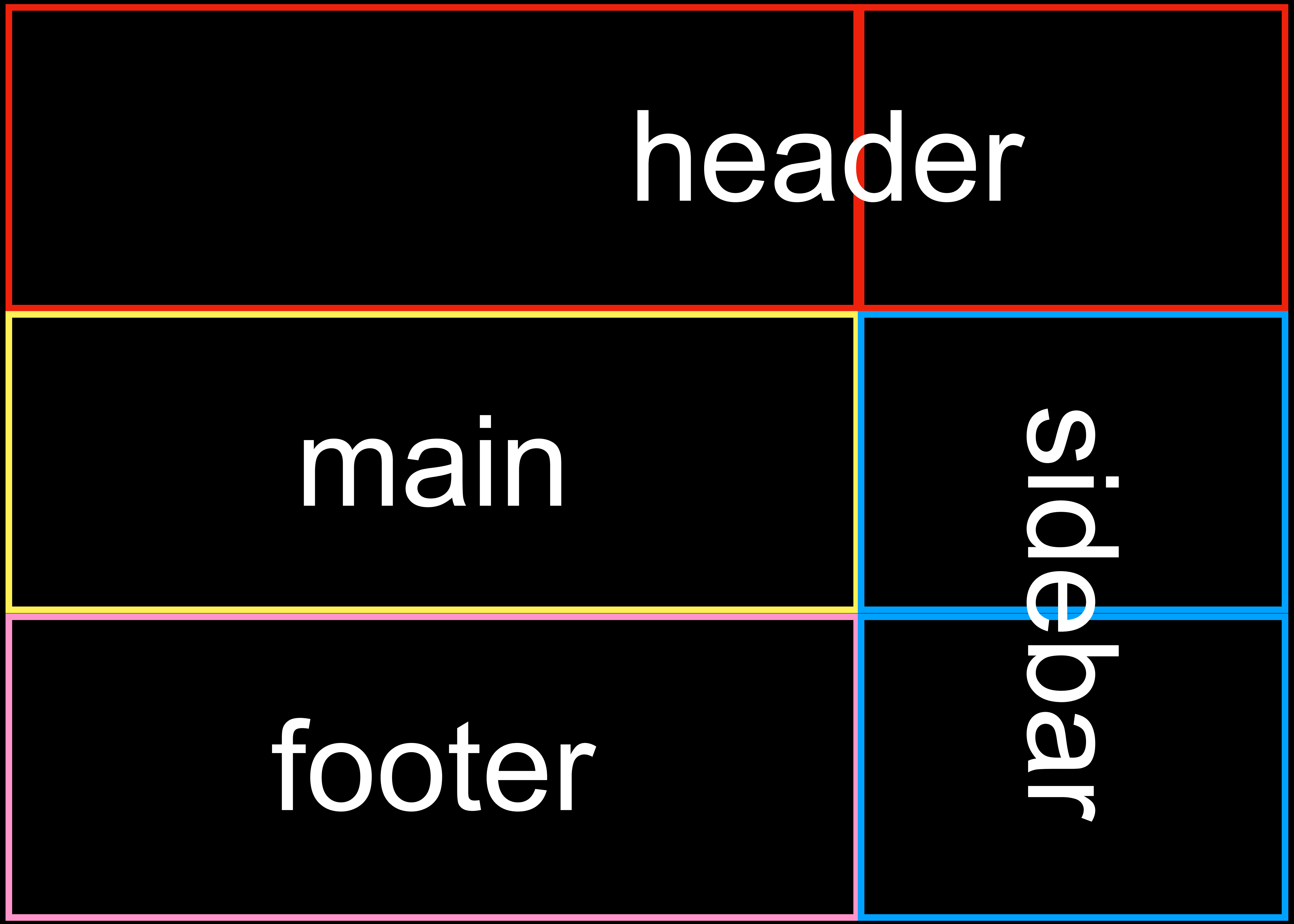


PARENT PROPERTIES: AREAS

Using a style a bit like ASCII, you can name parts of your grid so that its children can reference them.

PARENT PROPERTIES: AREAS

```
section {
  display: grid;
  grid-template-columns: 2fr 1fr;
  grid-template-areas:
    "header header "
    "main   sidebar"
    "footer sidebar";
}
```



CHILD PROPERTIES

Whereas parent properties start to build consistency, you can start to break the grid in interesting ways using child properties.

CHILD PROPERTIES: GRID-AREA

Say you’ve named grid areas, as we just saw. You can assign individual children to these areas.

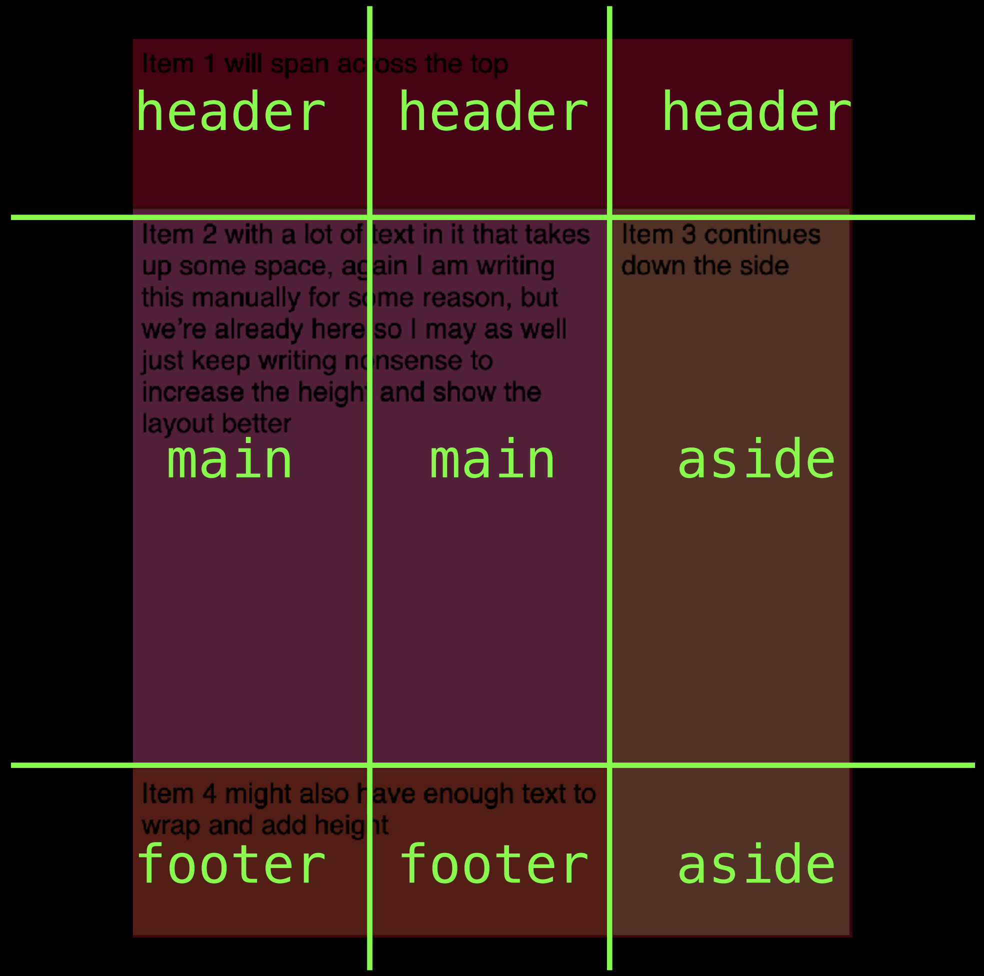
CHILD PROPERTIES: GRID-AREA

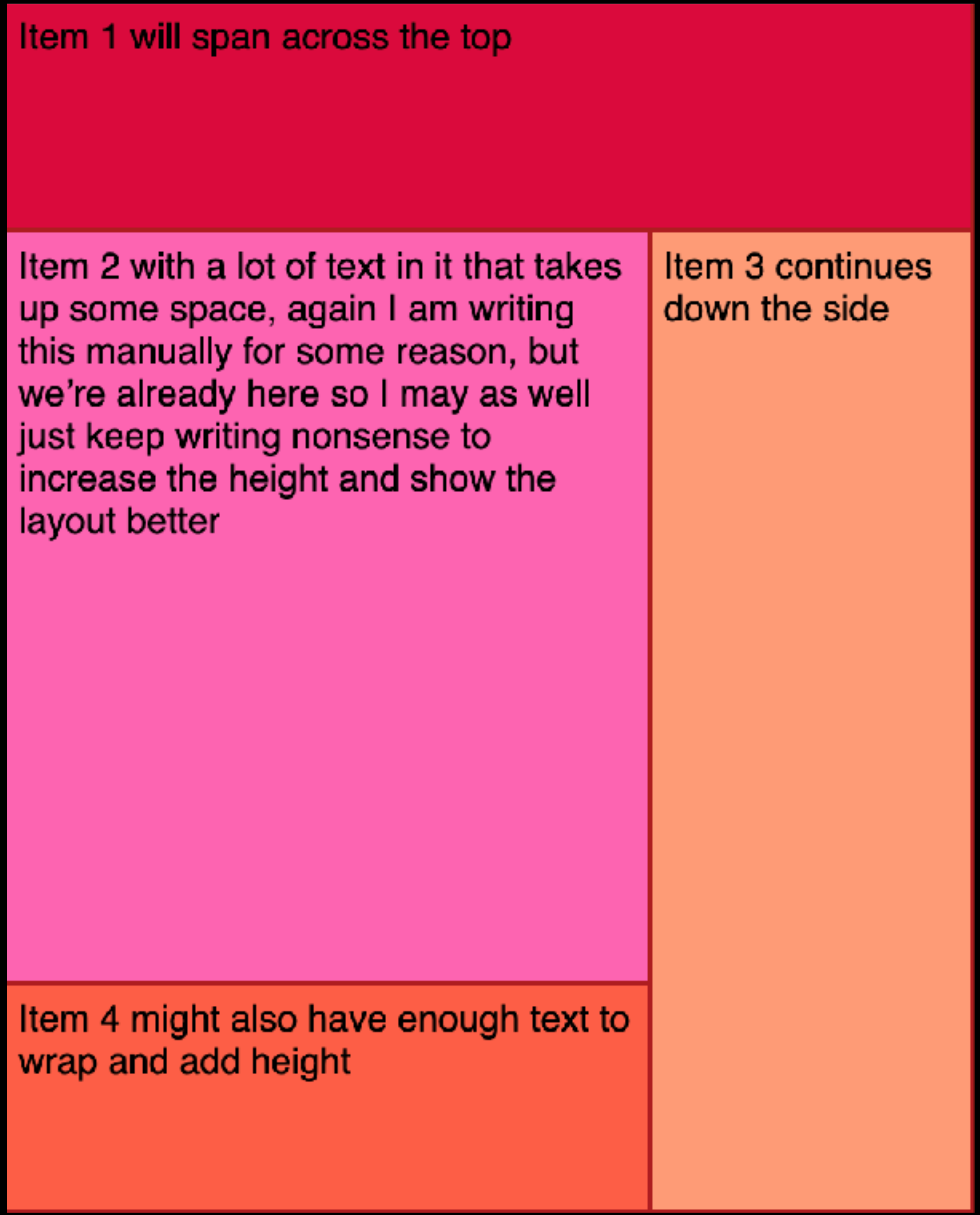
```css
section {
   display: grid;
   grid-template-columns: repeat(3, 1fr);
   grid-template-rows: 100px 50vh 100px;
   grid-template-areas:
      "header header header"
      "main   main   aside"
      "footer footer aside";
}

header {
   background-color: crimson;
   grid-area: header;
}

main {
   background-color: hotpink;
   grid-area: main;
}

aside {
   background-color: lightsalmon;
   grid-area: aside;
}

footer {
   grid-area: footer;
}
```

```html
<!DOCTYPE html>
<html>
   <head>
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <link href="/assets/styles/reset.css" rel="stylesheet">
      <link href="setup.css" rel="stylesheet">
      <link href="style.css" rel="stylesheet">
   </head>
   <body>
      <section>
         <header>
            <p>Item 1 will span across the top</p>
         </header>
         <main>
            <p>Item 2 with a lot of text in it that takes up some space, again
            I am writing this manually for some reason, but we’re already here
            so I may as well just keep writing nonsense to increase the height
            and show the layout better</p>
         </main>
         <aside>
            <p>Item 3 continues down the side</p>
         </aside>
         <footer>
            <p>Item 4 might also have enough text to wrap and add height</p>
         </footer>
      </section>
   </body>
</html>
```

CHILD PROPERTIES: GRID-AREA

```
section {
   display: grid;
   grid-template-columns: repeat(3, 1fr);
   grid-template-rows: 100px 50vh 100px;
   grid-template-areas:
      "header header header"
      "main   main   aside"
      "footer footer aside";
}

header {
   background-color: crimson;
   grid-area: header;
}

main {
   background-color: hotpink;
   grid-area: main;
}

aside {
   background-color: lightsalmon;
   grid-area: aside;
}

footer {
   grid-area: footer;
}
```





Item 1 will span across the top

Item 2 with a lot of text in it that takes up some space, again I am writing this manually for some reason, but we’re already here so I may as well just keep writing nonsense to increase the height and show the layout better

Item 3 continues down the side

Item 4 might also have enough text to wrap and add height

CHILD PROPERTIES: GRID-COLUMN/ GRID-ROW

You can also control item placement based on defined start and end grid lines

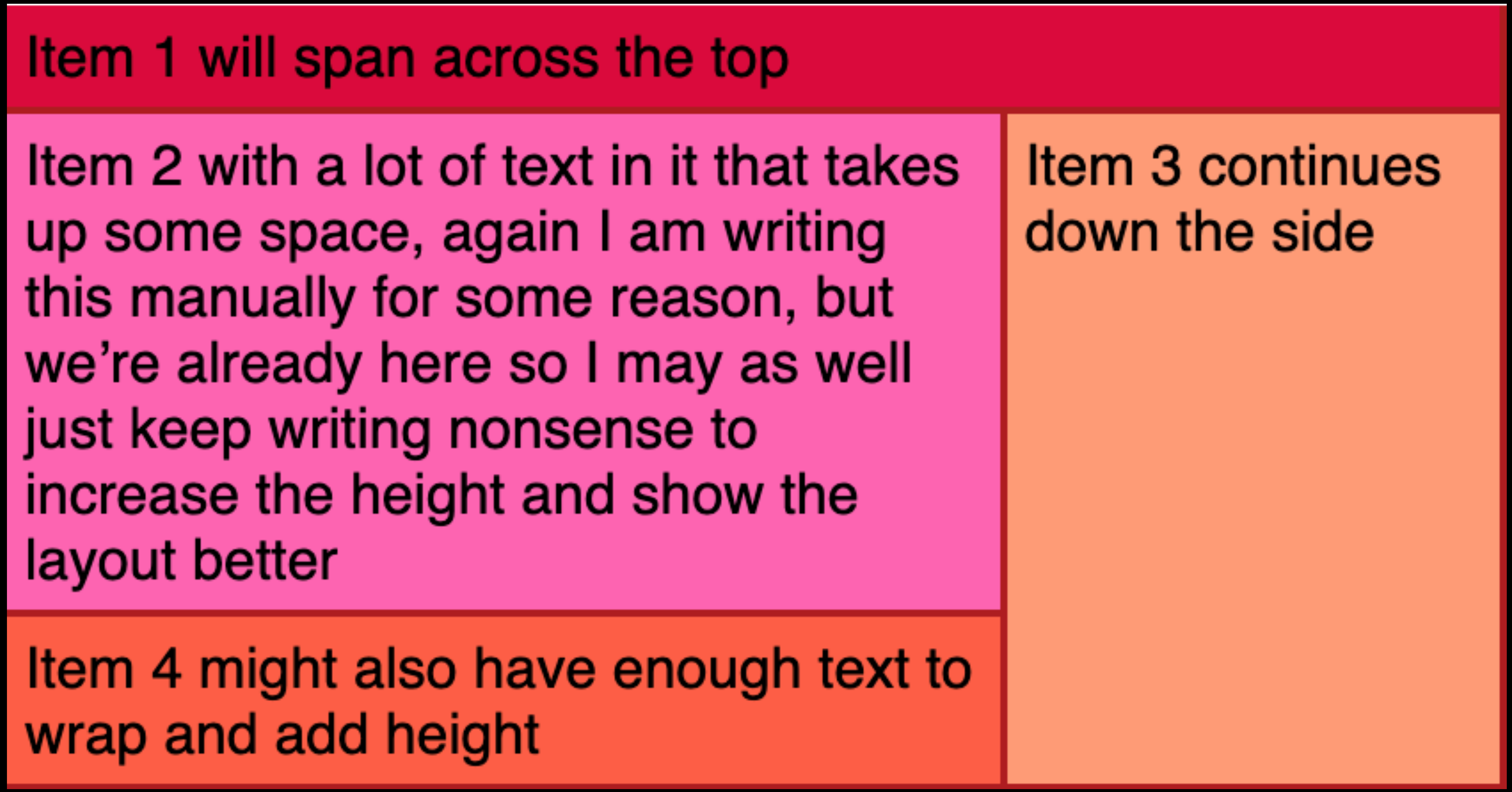
CHILD PROPERTIES: GRID-AREA

```css
section {
    display: grid;
    grid-template-columns: 2fr 1fr;
}

div:first-child {
    background-color: crimson;
    grid-column: 1 / span 2;
}

div:nth-child(2) {
    background-color: hotpink;
}

div:nth-child(3) {
    background-color: lightsalmon;
    grid-column: 2;
    grid-row: 2 / 4; /* Instead of span. */
}
```

```html
<!DOCTYPE html>
<html>
   <head>
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <link href="/assets/styles/reset.css" rel="stylesheet">
      <link href="setup.css" rel="stylesheet">
      <link href="style.css" rel="stylesheet">
   </head>
   <body>
      <section>
         <div>
            <p>Item 1 will span across the top</p>
         </div>
         <div>
            <p>Item 2 with a lot of text in it that takes up some space,
            again I am writing this manually for some reason, but we’re
            already here so I may as well just keep writing nonsense to
            increase the height and show the layout better</p>
         </div>
         <div>
            <p>Item 3 continues down the side</p>
         </div>
         <div>
            <p>Item 4 might also have enough text to wrap and add height</p>
         </div>
      </section>
   </body>
</html>
```

CHILD PROPERTIES: GRID-AREA

```
section {
    display: grid;
    grid-template-columns: 2fr 1fr;
}

div:first-child {
    background-color: crimson;
    grid-column: 1 / span 2;
}

div:nth-child(2) {
    background-color: hotpink;
}

div:nth-child(3) {
    background-color: lightsalmon;
    grid-column: 2;
    grid-row: 2 / 4; /* Instead of span. */
}
```



Item 1 will span across the top

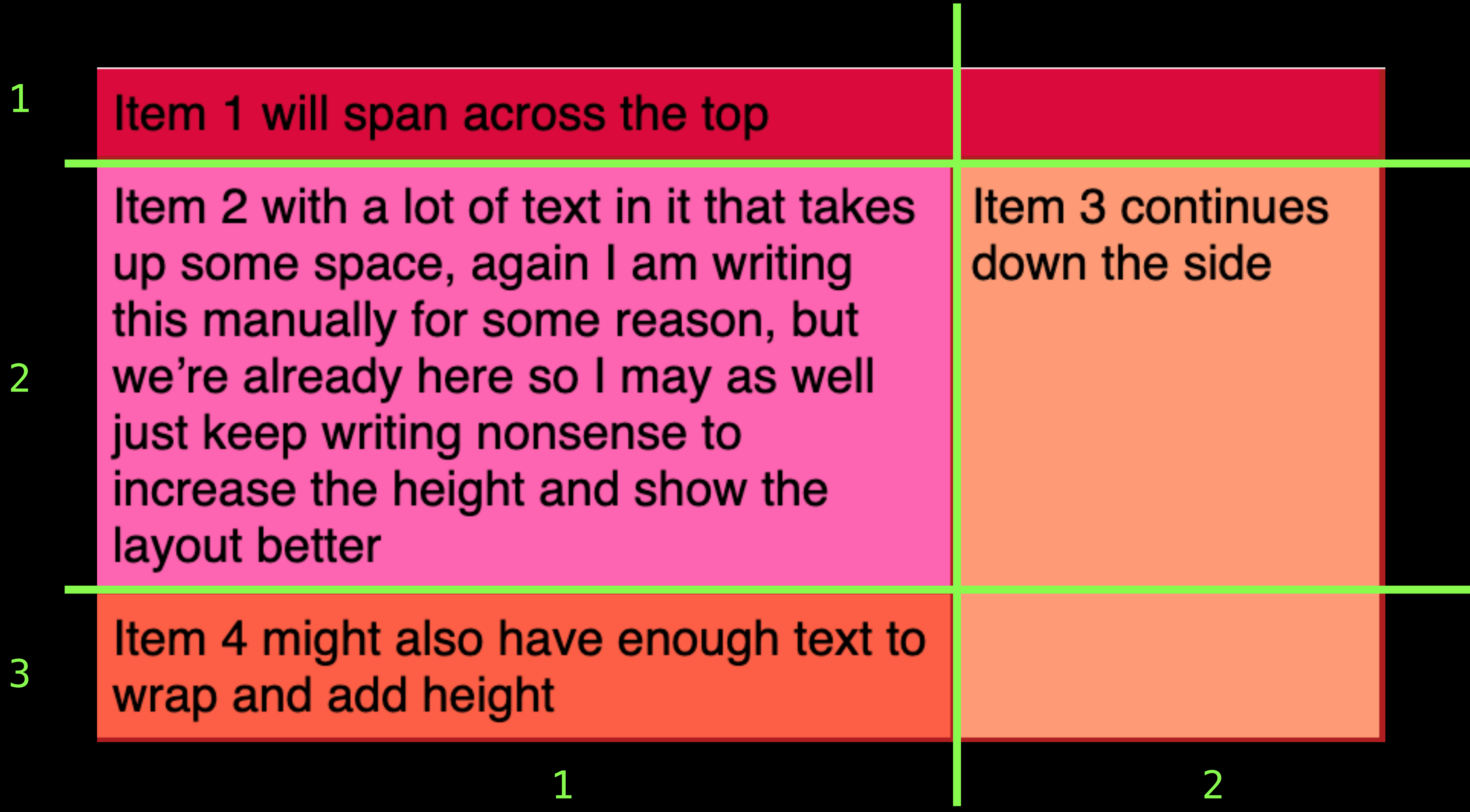
Item 2 with a lot of text in it that takes up some space, again I am writing this manually for some reason, but we’re already here so I may as well just keep writing nonsense to increase the height and show the layout better

Item 3 continues down the side

Item 4 might also have enough text to wrap and add height

CHILD PROPERTIES: GRID-AREA

```
section {
    display: grid;
    grid-template-columns: 2fr 1fr;
}

div:first-child {
    background-color: crimson;
    grid-column: 1 / span 2;
}

div:nth-child(2) {
    background-color: hotpink;
}

div:nth-child(3) {
    background-color: lightsalmon;
    grid-column: 2;
    grid-row: 2 / 4; /* Instead of span. */
}
```

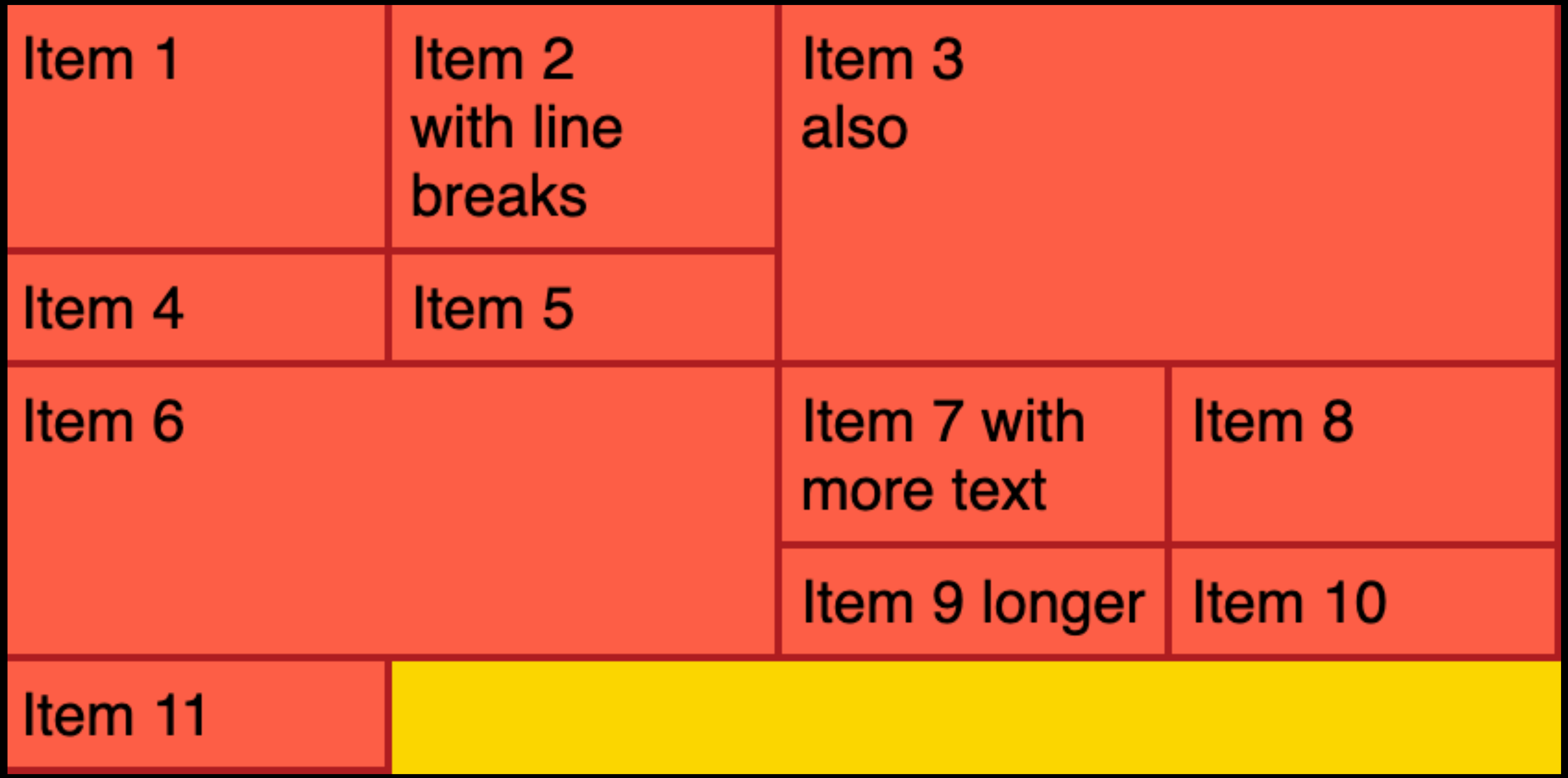

1
2
3
Item 1 will span across the top
Item 2 with a lot of text in it that takes up some space, again I am writing this manually for some reason, but we're already here so I may as well just keep writing nonsense to increase the height and show the layout better
Item 3 continues down the side
Item 4 might also have enough text to wrap and add height
1
2

CHILD PROPERTIES: GRID-COLUMN/ GRID-ROW

You can leave off the start line and just specify a span as well, if you just want to adjust the size of something wherever it flows in the grid.

CHILD PROPERTIES: GRID-COLUMN/ GRID-ROW

```
div:nth-child(3),
div:nth-child(6) {
  grid-column: span 2;
  grid-row: span 2;
}
```

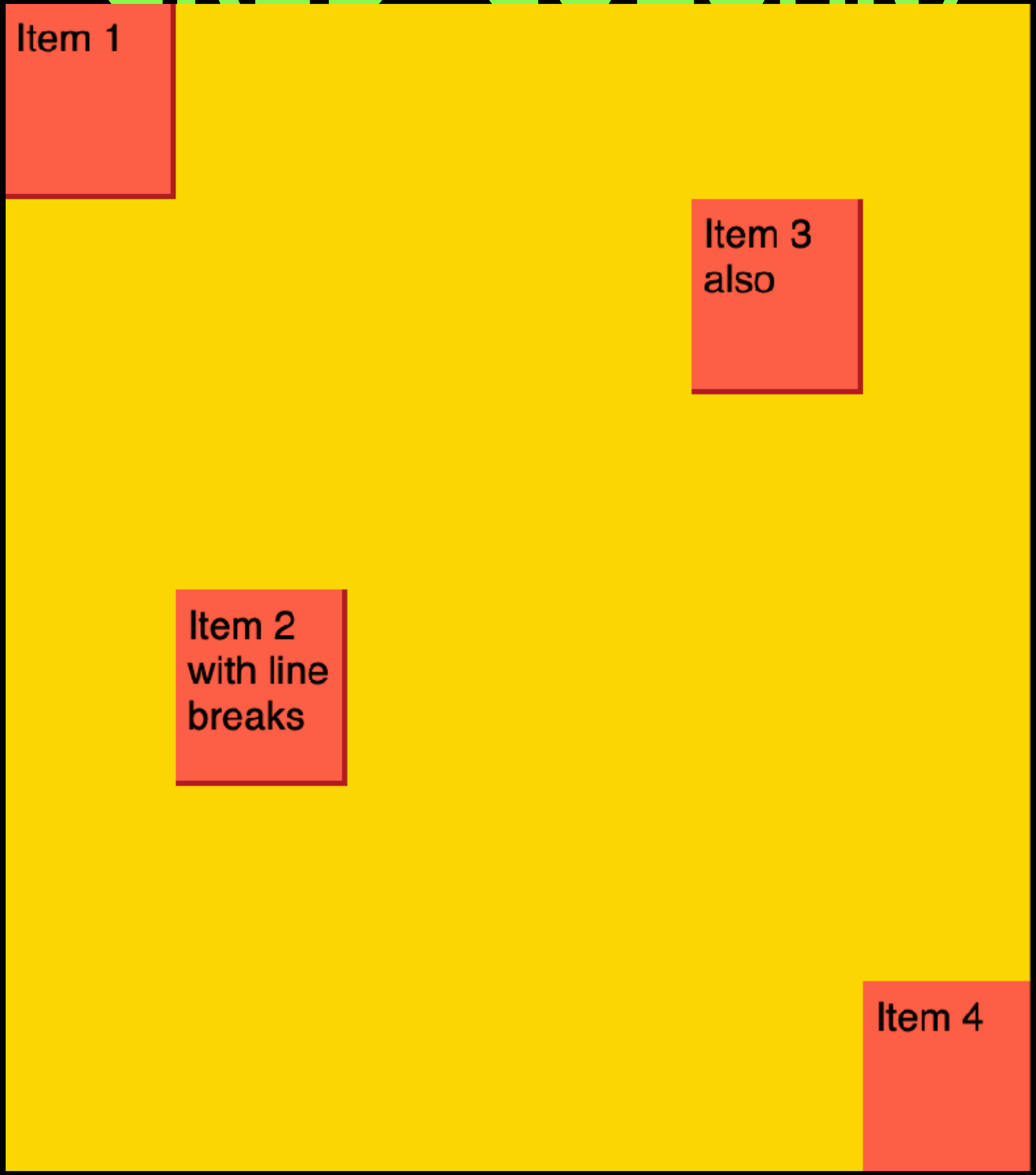


CHILD PROPERTIES: GRID-COLUMN/ GRID-ROW

In this way, you can also implicitly insert empty cells between your elements.

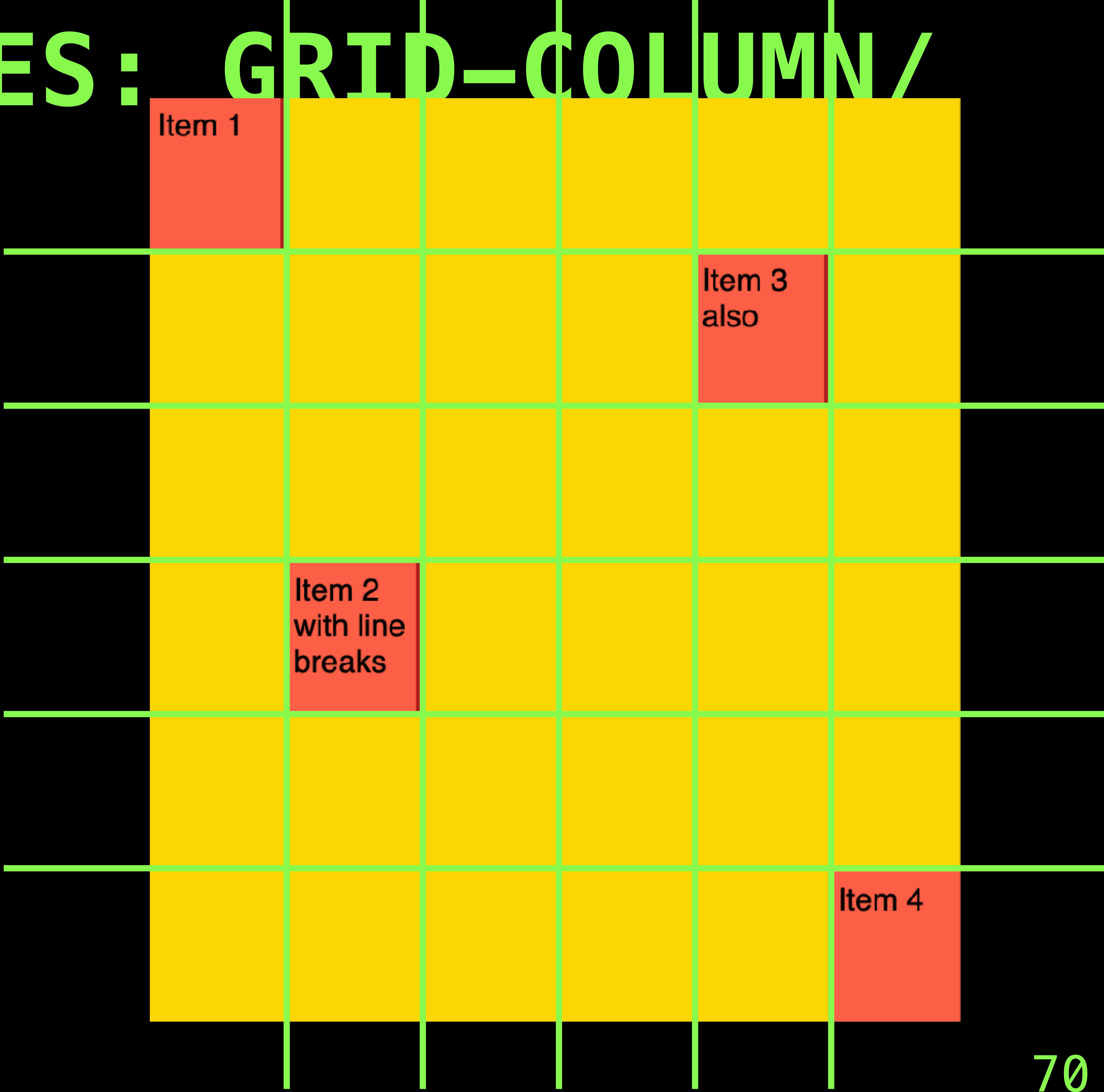
CHILD PROPERTIES: GRID-COLUMN/

```
section {
  display: grid;
  /* Give the implicit tracks a size. */
  grid-auto-rows: 80px;
  grid-auto-columns: 1fr;
}

div:nth-child(2) {
  grid-column: 2;
  grid-row: 4;
}

div:nth-child(3) {
  grid-column: 5;
  grid-row: 2;
}

div:last-child {
  grid-column: 6;
  grid-row: 6;
}
```



CHILD PROPERTIES: GRID-COLUMN/

```
section {
  display: grid;
  /* Give the implicit tracks a size. */
  grid-auto-rows: 80px;
  grid-auto-columns: 1fr;
}

div:nth-child(2) {
  grid-column: 2;
  grid-row: 4;
}

div:nth-child(3) {
  grid-column: 5;
  grid-row: 2;
}

div:last-child {
  grid-column: 6;
  grid-row: 6;
}
```

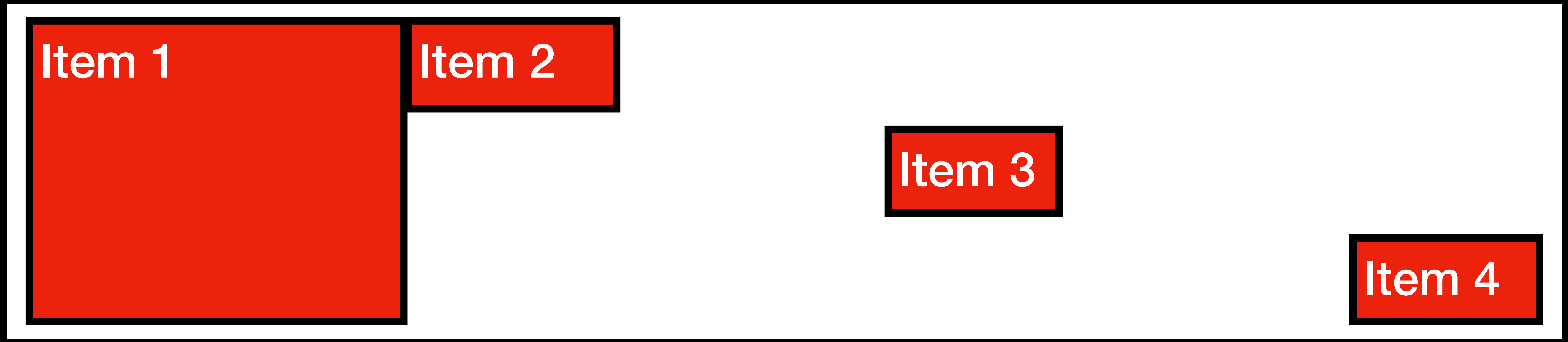


CHILD PROPERTIES: JUSTIFY/ ALIGN SELF

Lastly, you can justify or align children, just like in flexbox.

CHILD PROPERTIES: GRID-COLUMN/ GRID-ROW

```
section {
  display: grid;
  grid-auto-rows: 100px;
  grid-template-columns: repeat(4, 1fr);
}

div:first-child {
  /* Defaults. */
  justify-self: stretch;
  align-self: stretch;
}

div:nth-child(2) {
  justify-self: start;
  align-self: start;
}

div:nth-child(3) {
  justify-self: center;
  align-self: center;
}

div:nth-child(4) {
  justify-self: end;
  align-self: end;
}
```



BREAK (5 MIN)

IMAGES

As you know… this class has been largely about layouts and typography. Let’s chat images for a bit.

IMAGE FORMATS

.gif GRAPHICS INTERCHANGE FORMAT
GIFs don't have to be animated… but that's the only real reason to use them. They're compressed and have reduced color palettes.

.jpg JOINT PHOTOGRAPHIC *EXPERTS* GROUP
JPGs are ubiquitous and perfectly fine—just remember that they're lossy image formats. Use jpgs for photos.

.png PORTABLE NETWORK GRAPHICS
PNGs are often more precise in terms of colors, and can be lossless. Illustrations or graphics can be in pngs, and have transparency.

.svg SCALABLE VECTOR GRAPHICS
SVGs are vector images that are great for illustrations or logos. What's more, they can be adjusted directly in your code.

SIZING

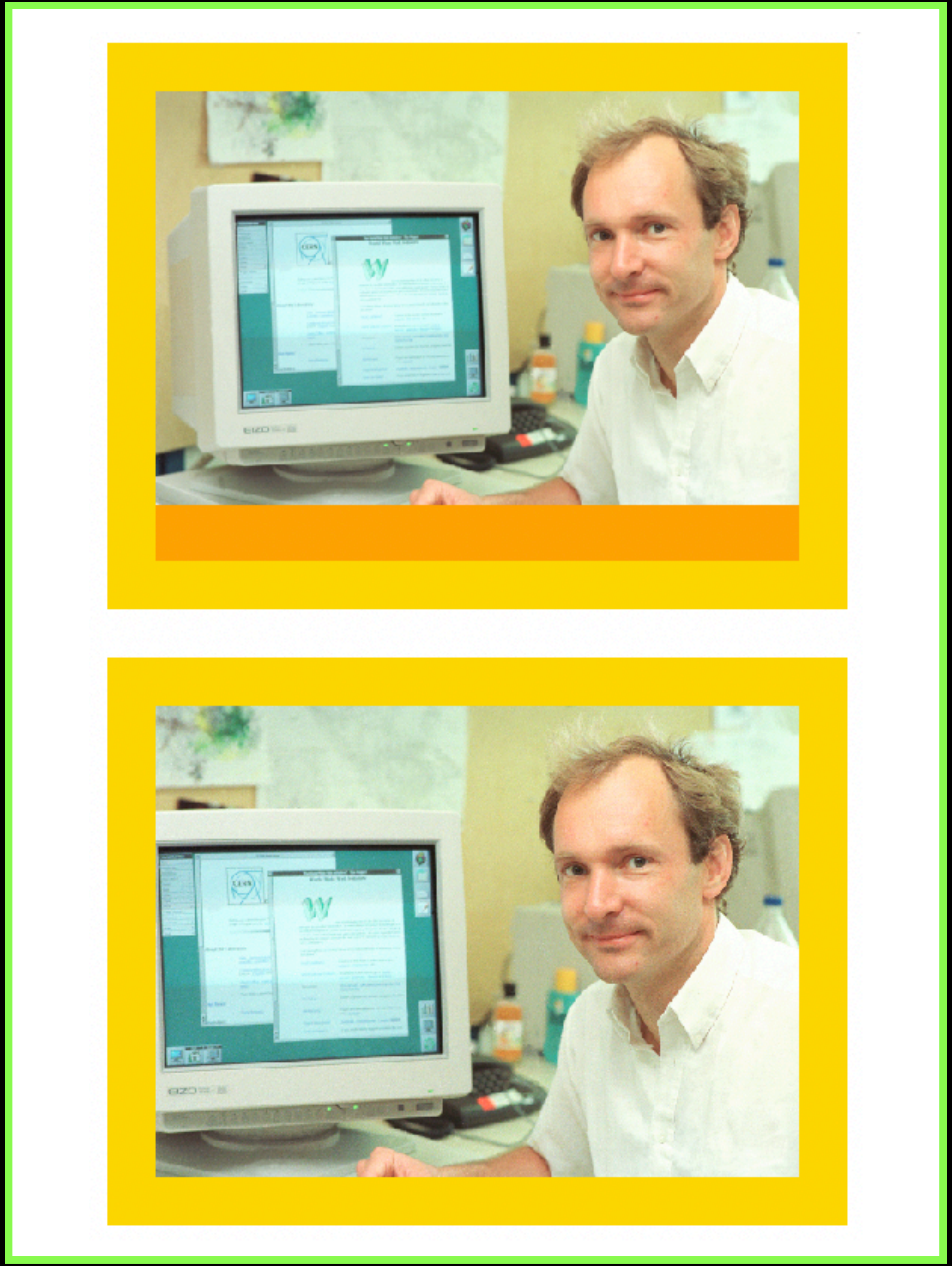
By default, an image will scale at 100% of its pixel dimension, and will be inline.

Consider setting it to `display: block;` before starting to position or size it.

OBJECT-FIT

CSS has added object-fit and object-position to automatically size images within containers.

OBJECT-FIT

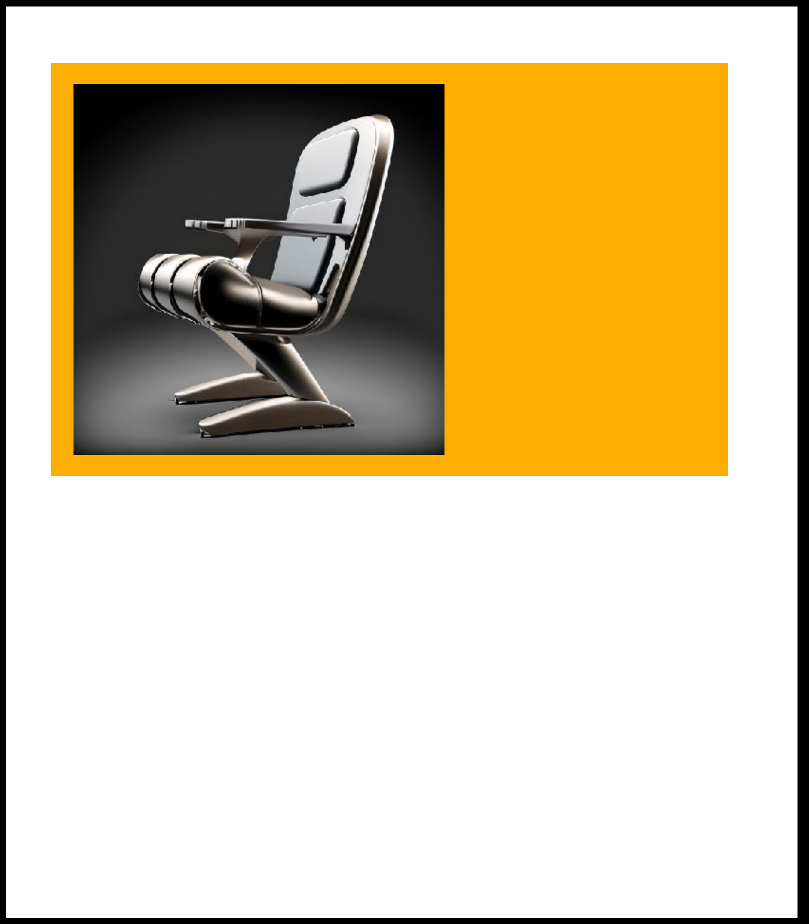


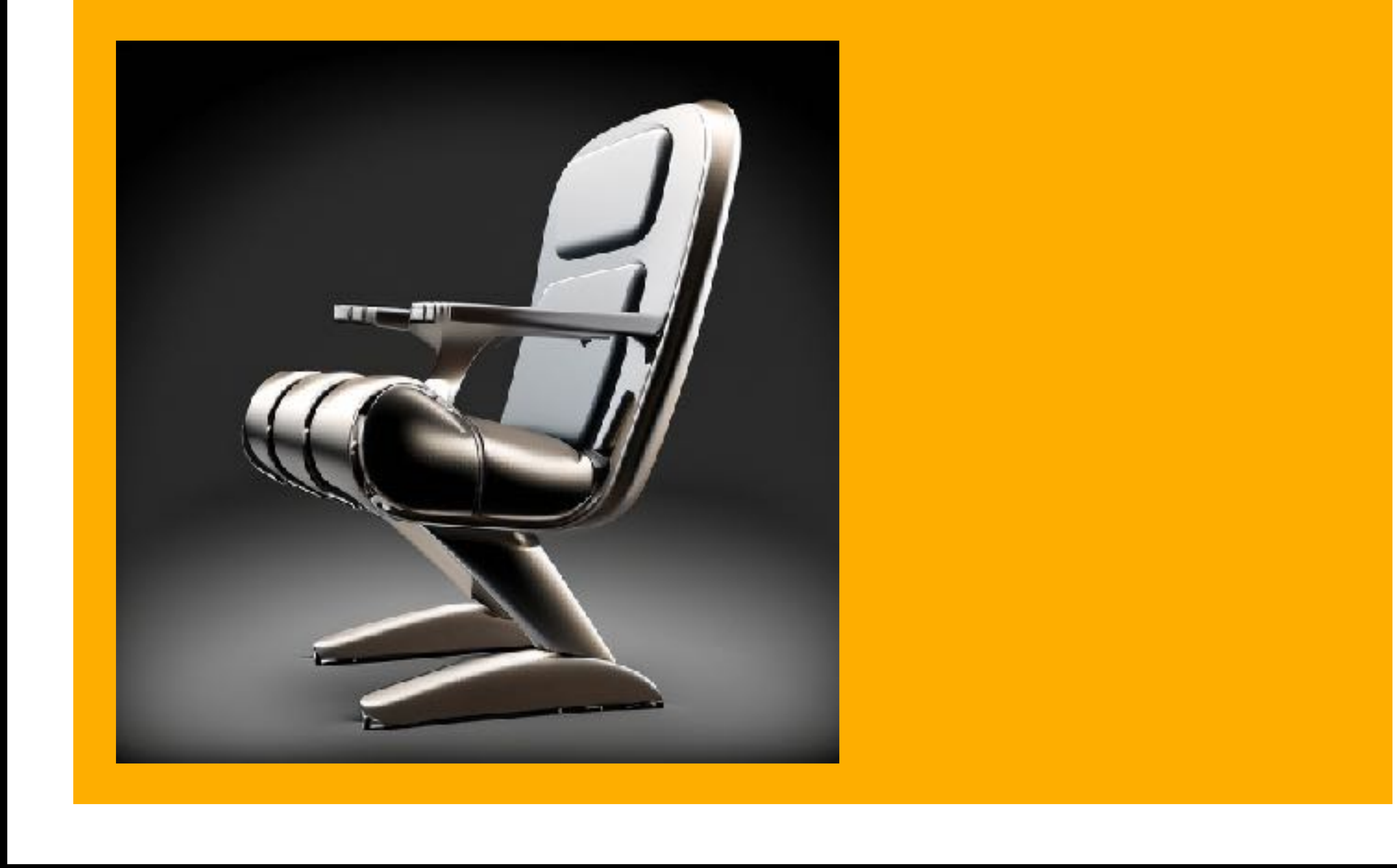
```
body, section { padding: 20px; }

section {
  background-color: gold;
  height: 40vh;
}

section:not(:first-child) { margin-top: 20px;}

img {
  background-color: orange;
  height: 100%; /* Fill the parent. */
  width: 100%;
}

section:first-child img {
  object-fit: contain; /* Fit the image. */
  object-position: left top; /* Corner. */
}

section:last-child img {
  object-fit: cover; /* Cover the parent. */
  object-position: right center; /* Side. */
}
```

ASPECT-RATIO

CSS also supports aspect ratios as a property (for anything! Not just images) to maintain dimensional ratios while an element scales.

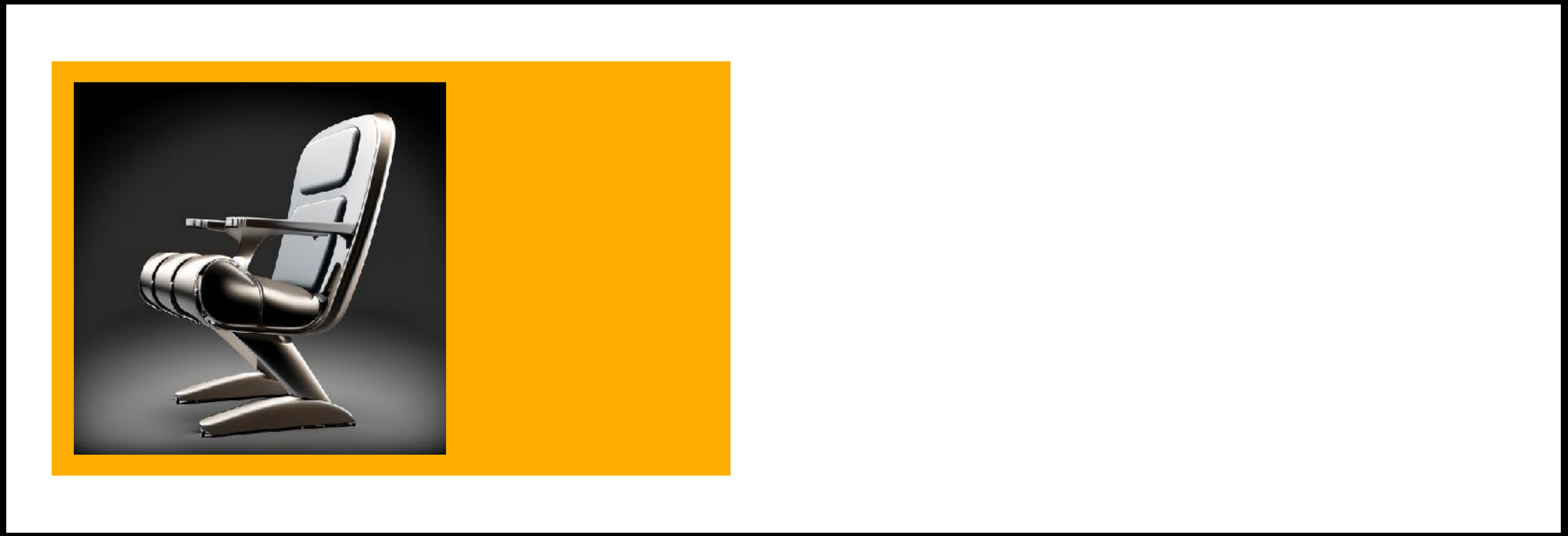
ASPECT-RATIO





```
aspect-ratio:  1 / 1;
```

W / H



BACKGROUND-IMAGE

Using `background-image`, you can put an image in the background of a container (or entire webpage). You can't effectively control its crop or scale, so don't use this for content.

BACKGROUND-IMAGE

```
body {
 background-image: url(“metalpizza.png”)
 background-size:cover
 background-origin: border-box
}
```

FIGURES

You can use the `<figure>` element in your code to associate an image with a description.

FIGURES

```
<figure>
  <img src=“planewindow.jpg”>
  <figcaption>Woman peering out of
  an airplane window</figcaption>
</figure>
```

```
…
figure { background-color: red; }

figcaption {
  font-family: sans-serif;
  margin-top: 10px;
}
```

Woman peering out of an airplane window

`<PICTURE>`

There’s also a `<picture>` element, which allows you to select image sources by media query.

<PICTURE>

```
<picture>
  <source media="(max-width: 428px)"
  srcset="tim.jpg">
  <source media="(max-width: 640px)"
  srcset="tim--md.jpg">
  <img alt="Tim Berners-Lee at a computer."
  src="tim--lg.jpg">
</picture>
```

SOME FILTERS

The CSS `filter` property allows you to apply select visual effects live in the browser… but, you may prefer to simply edit photos offline and load those in.

(LIST OF PROPERTIES)

EVEN MORE CSS

Typically, interactivity should be introduce using specialized languages that do it best, but CSS has a limited capacity it.

OVERFLOW

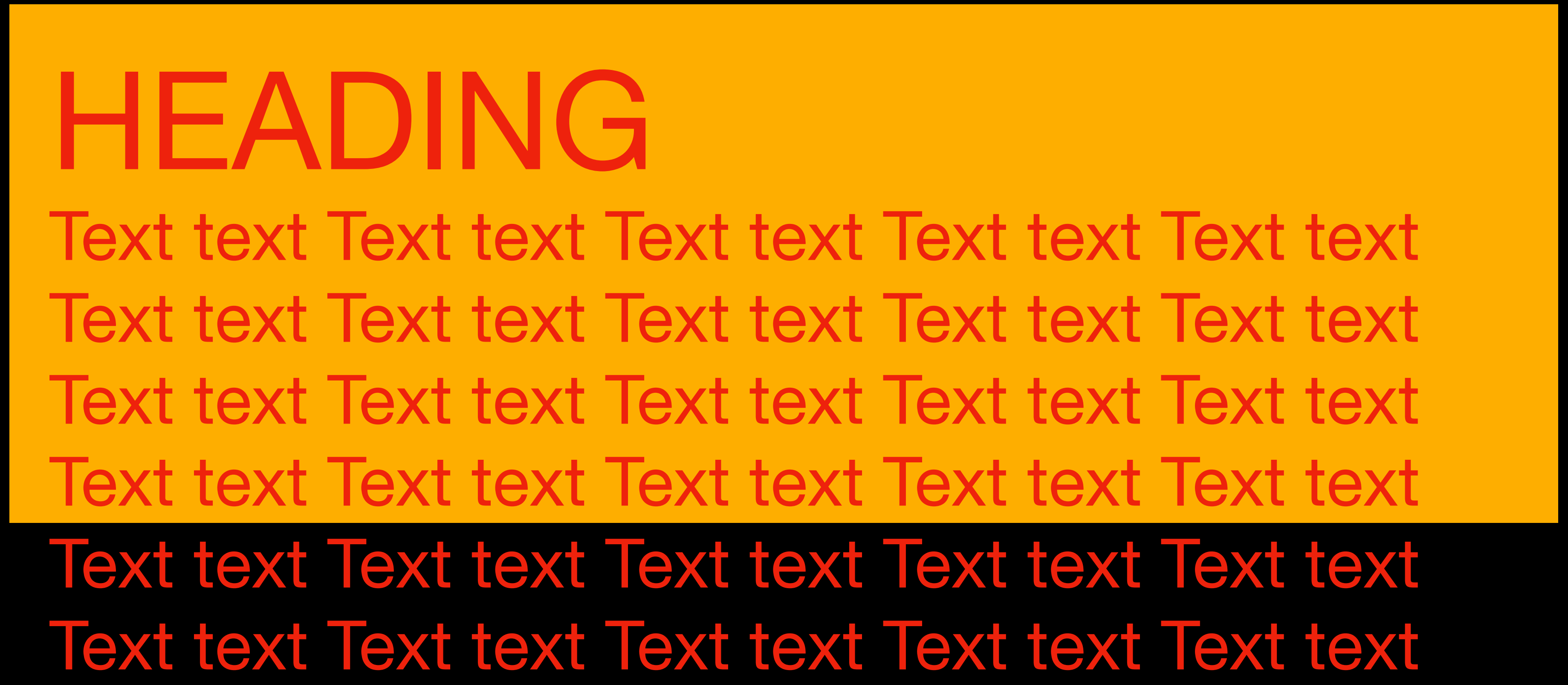
An **overflow** happens when there is too much content to fit into a container—often due to imposed constraints on height or width. You can use this to crop content or create scrolling areas

OVERFLOW

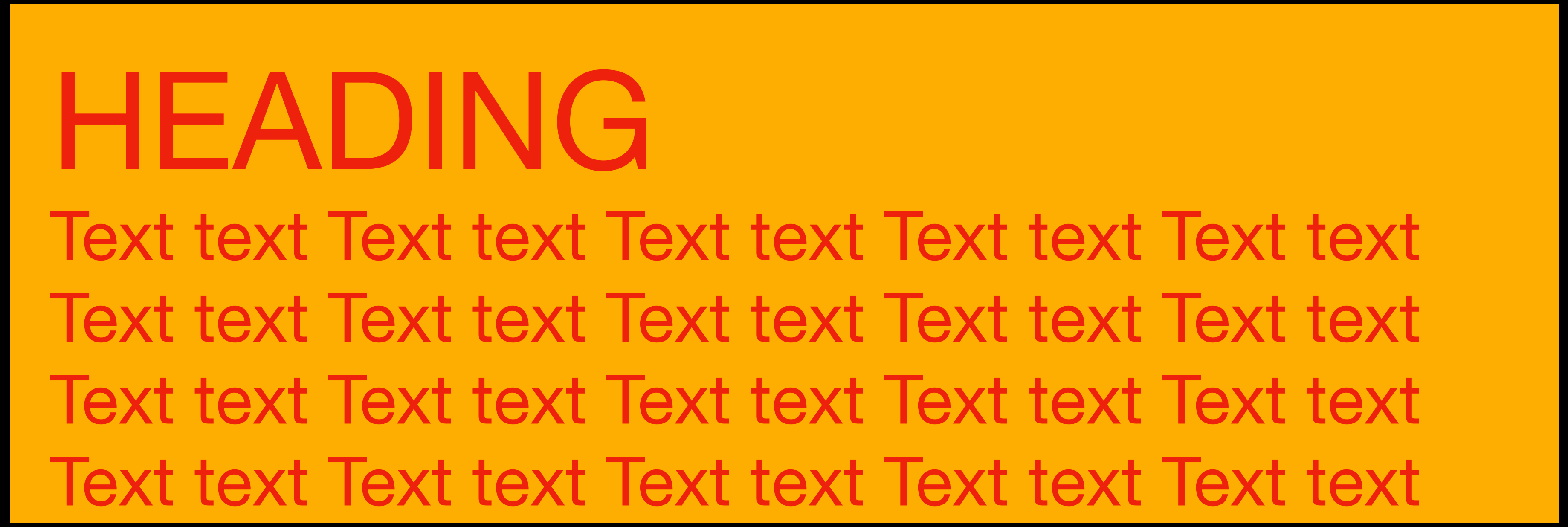
An **overflow** happens when there is too much content to fit into a container—often due to imposed constraints on height or width. You can use this to crop content or create scrolling areas

OVERFLOW

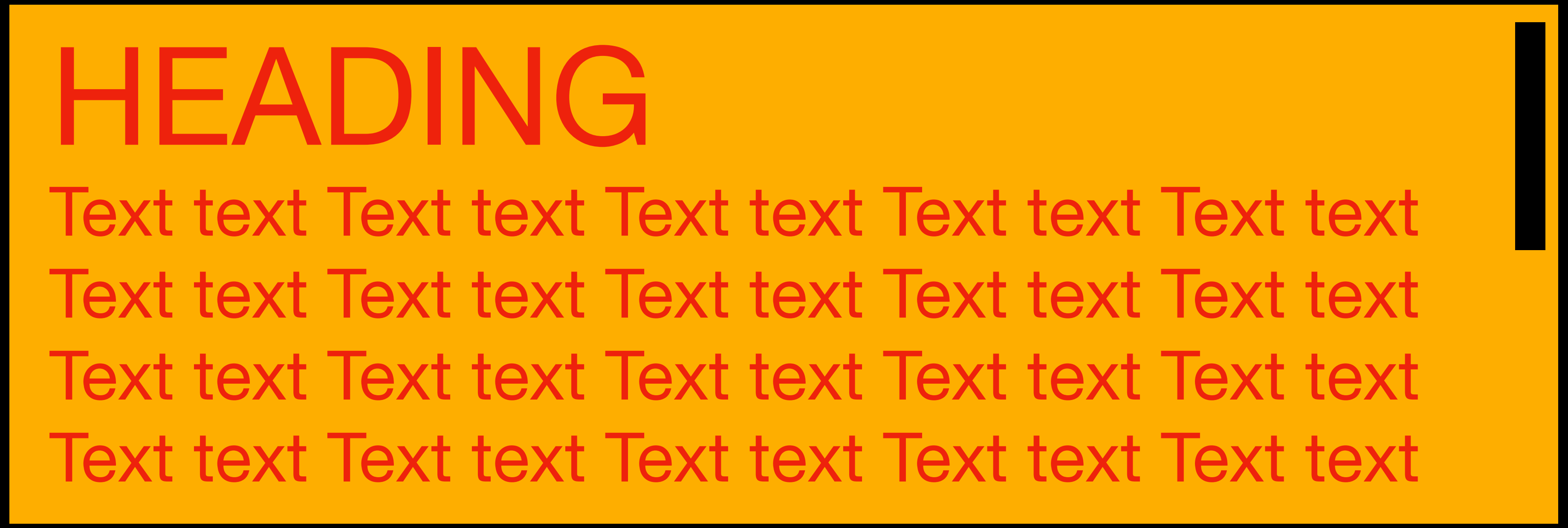
overflow: visible


HEADING
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text

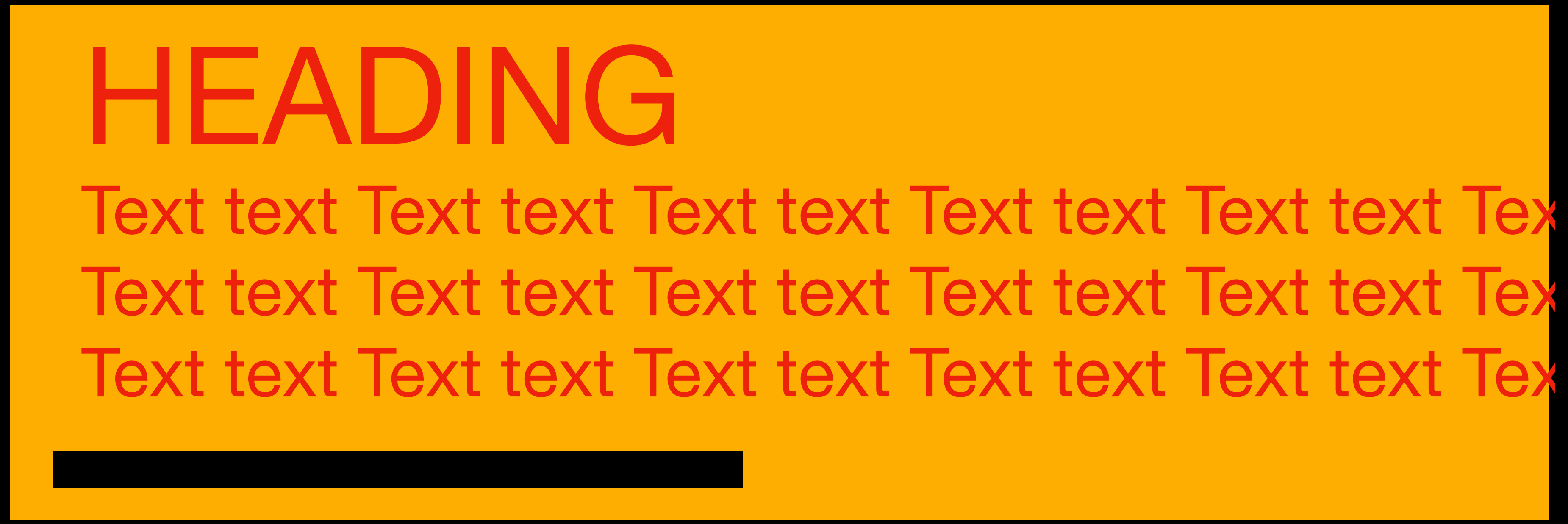
overflow: hidden


HEADING
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text

overflow-x: auto


HEADING
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text
Text text Text text Text text Text text Text text

overflow-y: auto

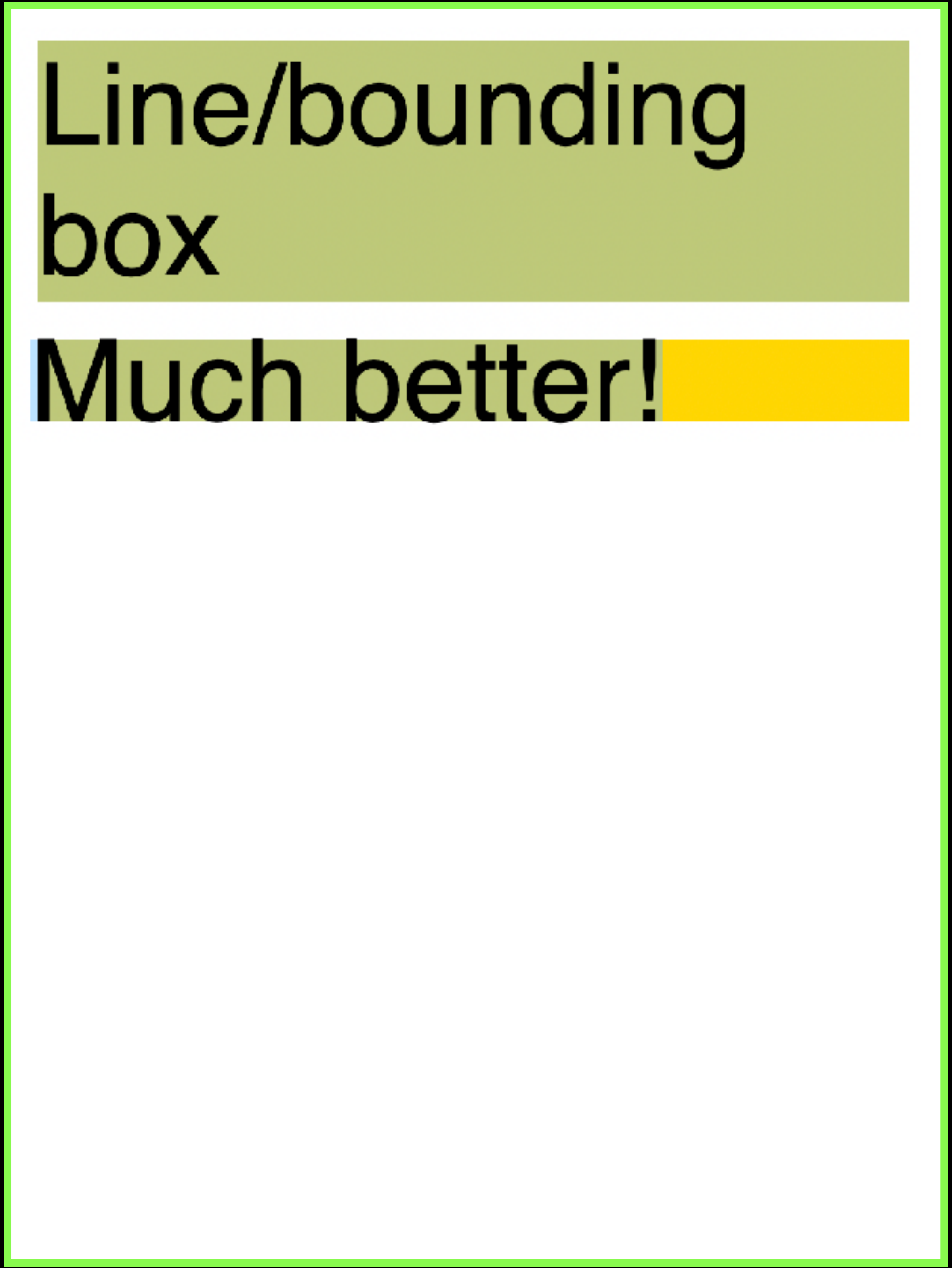

HEADING
Text text Text text Text text Text text Text text Tex
Text text Text text Text text Text text Text text Tex
Text text Text text Text text Text text Text text Tex

TYPOGRAPHY

HTML automatically renders space around text elements—called the *line box* or *bounding box*. This makes precision typography difficult. You can use pseudelement selectors* to finesse your type with negative margins to negate this.

*see week 3 lecture for a refresh

TYPOGRAPHY

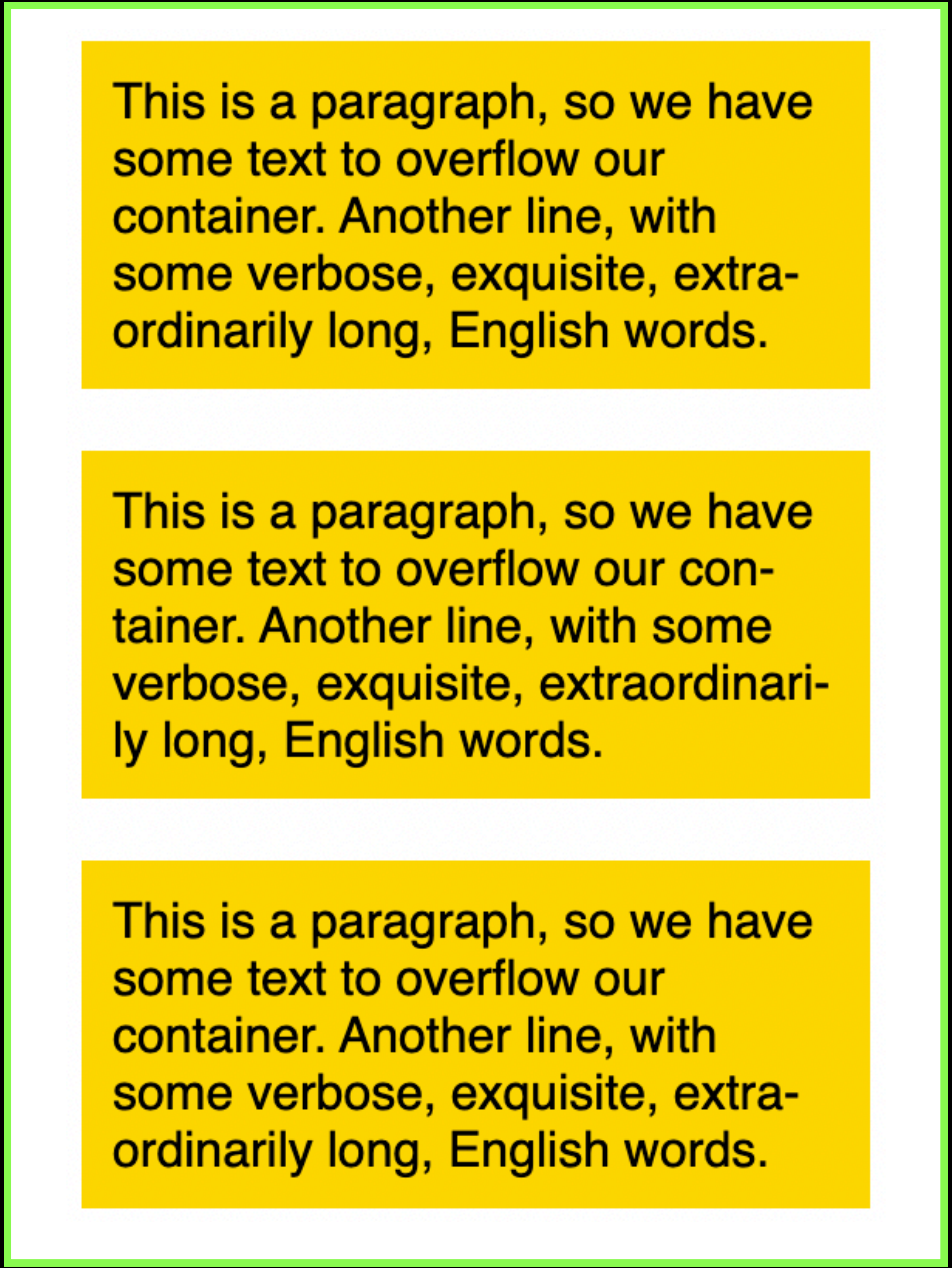


```
body {
   --base: 20px;

   display: grid;
   font-family: 'Helvetica', sans-serif;
   gap: var(--base);
   padding: var(--base);
}

section { background-color: gold; }

h2 { font-size: calc(var(--base) * 3)
   background-color: hsla(200, 100%, 50%, 33%);
}

.precise:before,
.precise:after {
   content: ''; /* Empty. */
   display: block;
   visibility: hidden;
}

.precise:before { margin-top: var(--inset--top); }
.precise:after { margin-bottom: var(--inset--bottom); }

.precise {
   --inset--top:    -0.2em; /* Font-dependent! */
   --inset--bottom: -0.23em;
   --inset--left:   -0.07em;
   --inset--right:  -0.06em;

   display: flow-root; /* Cinch the height. */
   margin-left: var(--inset--left);
   margin-right: var(--inset--right);
   width: fit-content; /* Cinch the width. */
}
```

TYPOGRAPHY: HYPHENS

HTML also automatically breaks up your text based on its parent/container, leaving potentially crunchy rags. Here are some (fallible) solves.

TYPOGRAPHY: HYPHENS



This is a paragraph, so we have some text to overflow our container. Another line, with some verbose, exquisite, extra-ordinarily long, English words.

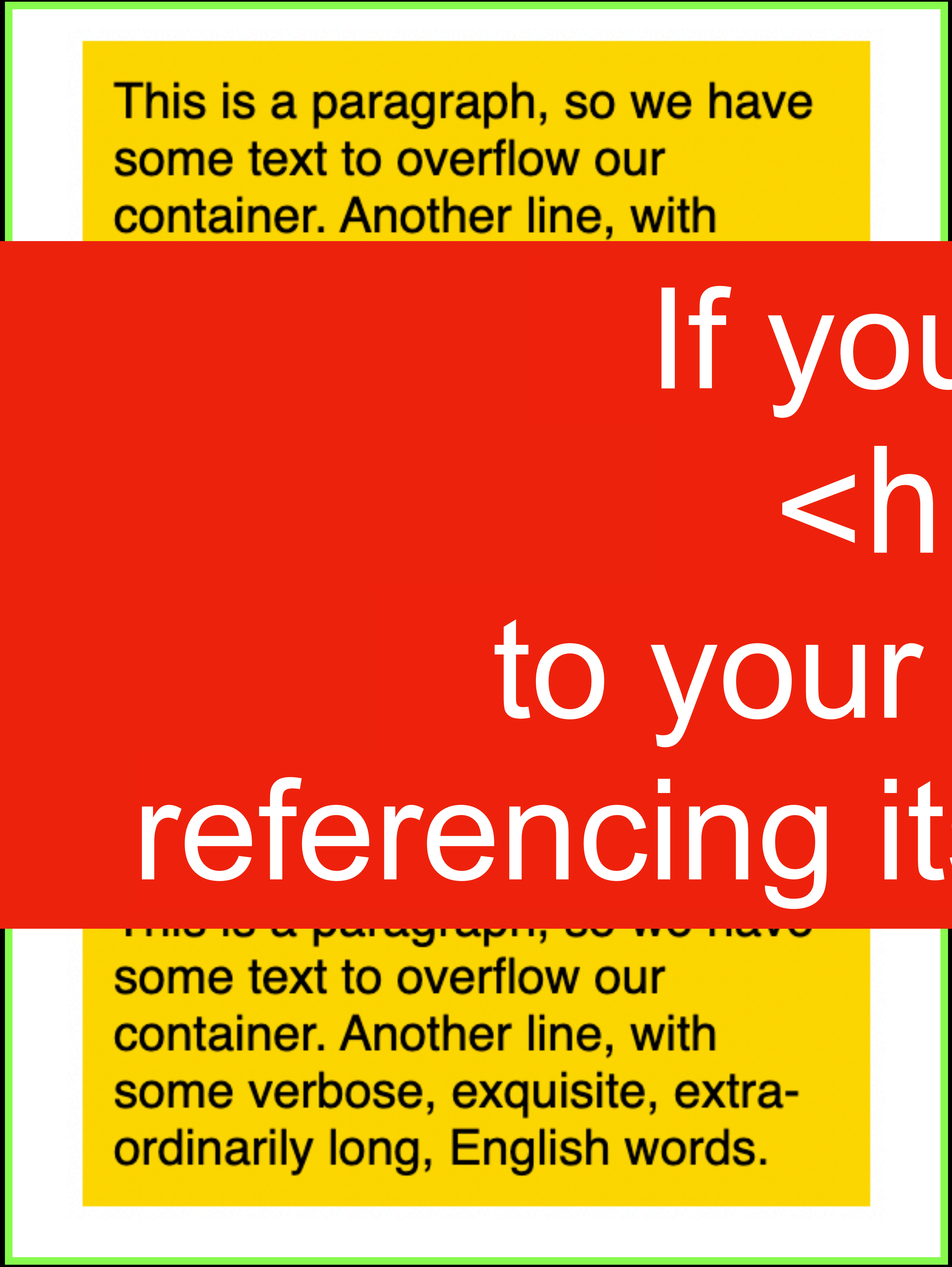
This is a paragraph, so we have some text to overflow our con-tainer. Another line, with some verbose, exquisite, extraordinari-ly long, English words.

This is a paragraph, so we have some text to overflow our container. Another line, with some verbose, exquisite, extra-ordinarily long, English words.

```
<p>This is a paragraph, so we have some text to
overflow our container. Another line, with some
verbose, exquisite, extra&shy;ordinarily long,
English words.</p>
```

```
section:nth-child(1) {
  -webkit-hyphens: none; /* Safari uses prefix. */
  hyphens: none; /* Default. */
}

section:nth-child(2) {
  -webkit-hyphens: auto; /* Safari uses prefix. */
  hyphens: auto; /* Browser decides. */
}

section:nth-child(3) {
  -webkit-hyphens: manual; /* Safari uses prefix. */
  hyphens: manual; /* Specify with `&shy;` */
}
```

TYPOGRAPHY: HYPHENS



This is a paragraph, so we have
some text to overflow our
container. Another line, with

some text to overflow our
container. Another line, with
some verbose, exquisite, extra-
ordinarily long, English words.

```
<p>This is a paragraph, so we have some text to
overflow our container. Another line, with some
verbose, exquisite, extra&shy;ordinarily long,
```

```
section:nth-child(3) {
  -webkit-hyphens: manual; /* Safari uses prefix. */
  hyphens: manual; /* Specify with `&shy;` */
}
```

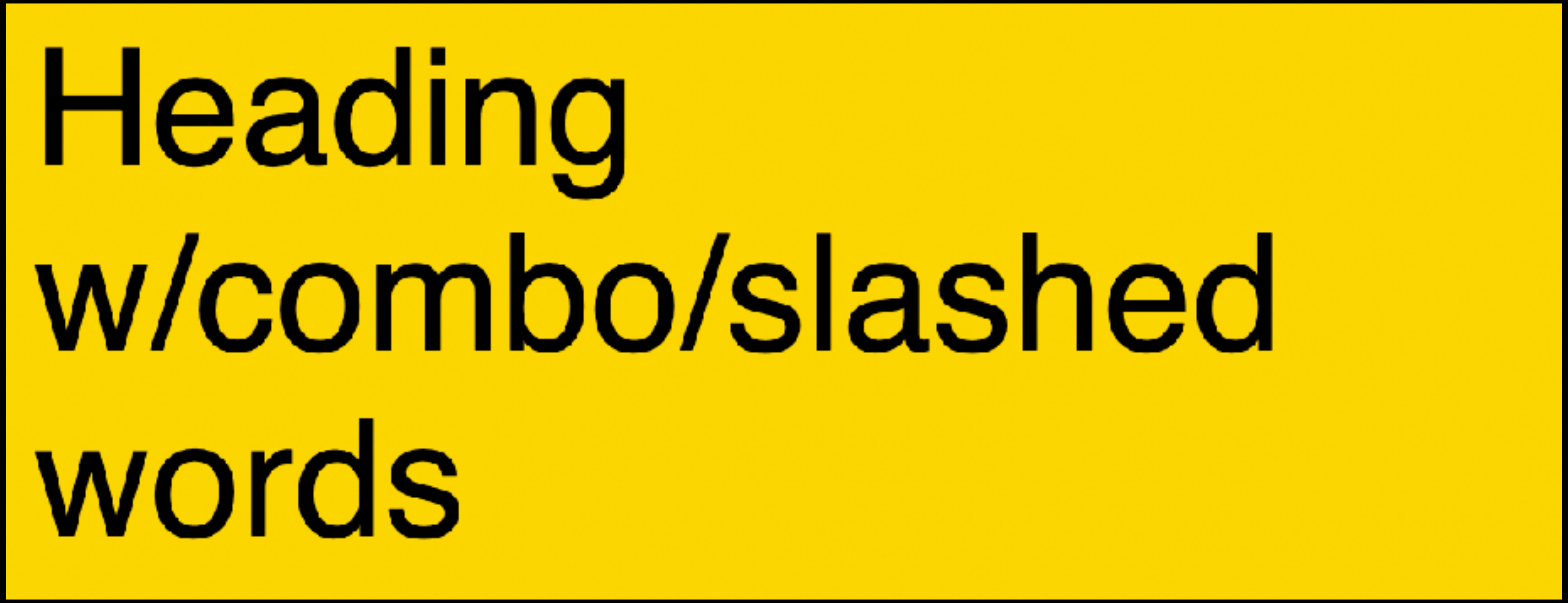
If you set this to auto, add
`<html lang="en">`
to your head, so the browser is
referencing its internal English dictionary

TYPOGRAPHY: HYPHENS

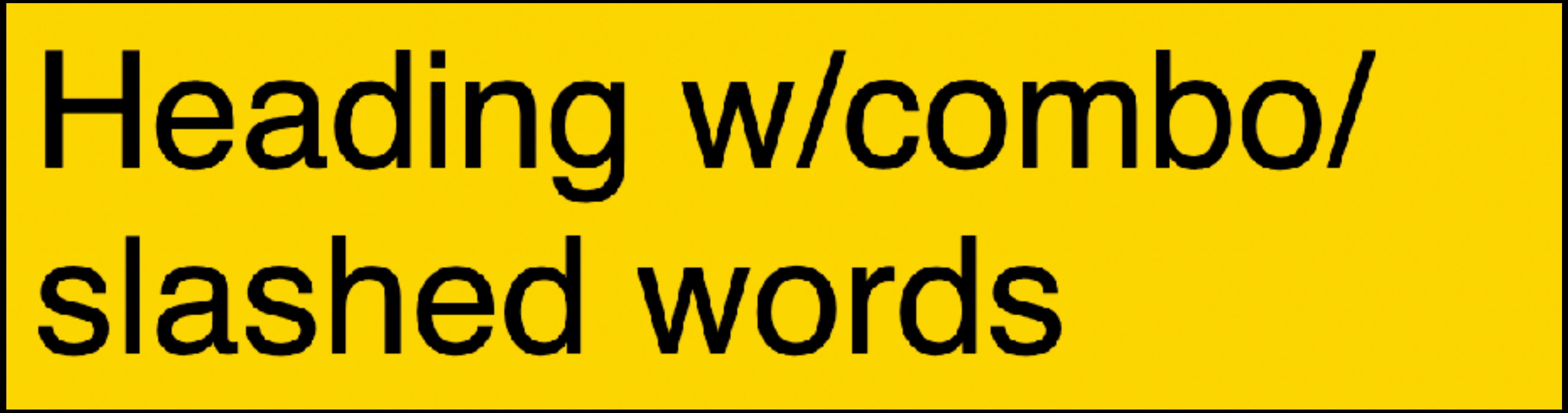
You could also use <wbr> as an optional line break for where a single long word (without a hyphen) could wrap.

TYPOGRAPHY: HYPHENS

```
<p>Heading w/combo/
slashed words</p>
```


Heading
w/combo/slashed
words

```
<p>Heading w/combo/<wbr>
slashed words</p>
```


Heading w/combo/
slashed words

TYPOGRAPHY: <NOBR> AND &NBSP;

You can wrap multiple words in a `<nobr>` tag—which is an inline element, like `<strong>`—to tell your code not to break them up on two lines. You might use this to keep a name or date together, or to avoid orphans at the end of a paragraph.

TYPOGRAPHY: <NOBR> AND &NBSP;

You can also insert the character `&nbsp;` in between words manually to do the same thing.

TRANSFORM

CSS also supports (limited) visual manipulation of elements using transforms. These are applied by the browser after the rest of your CSS is processed.

TRANSFORM

Apply transforms in css with the property,

`transform: value;`

`scale() / scaleX() / scaleY() / scaleZ() / scale3d()`	Changes the displayed size of an element
`skew() / skewX() / skewY()`	Distorts an element, like turning a rectangle into a parallelogram
`rotate() / rotate3d()`	Rotates an element

TRANSFORM

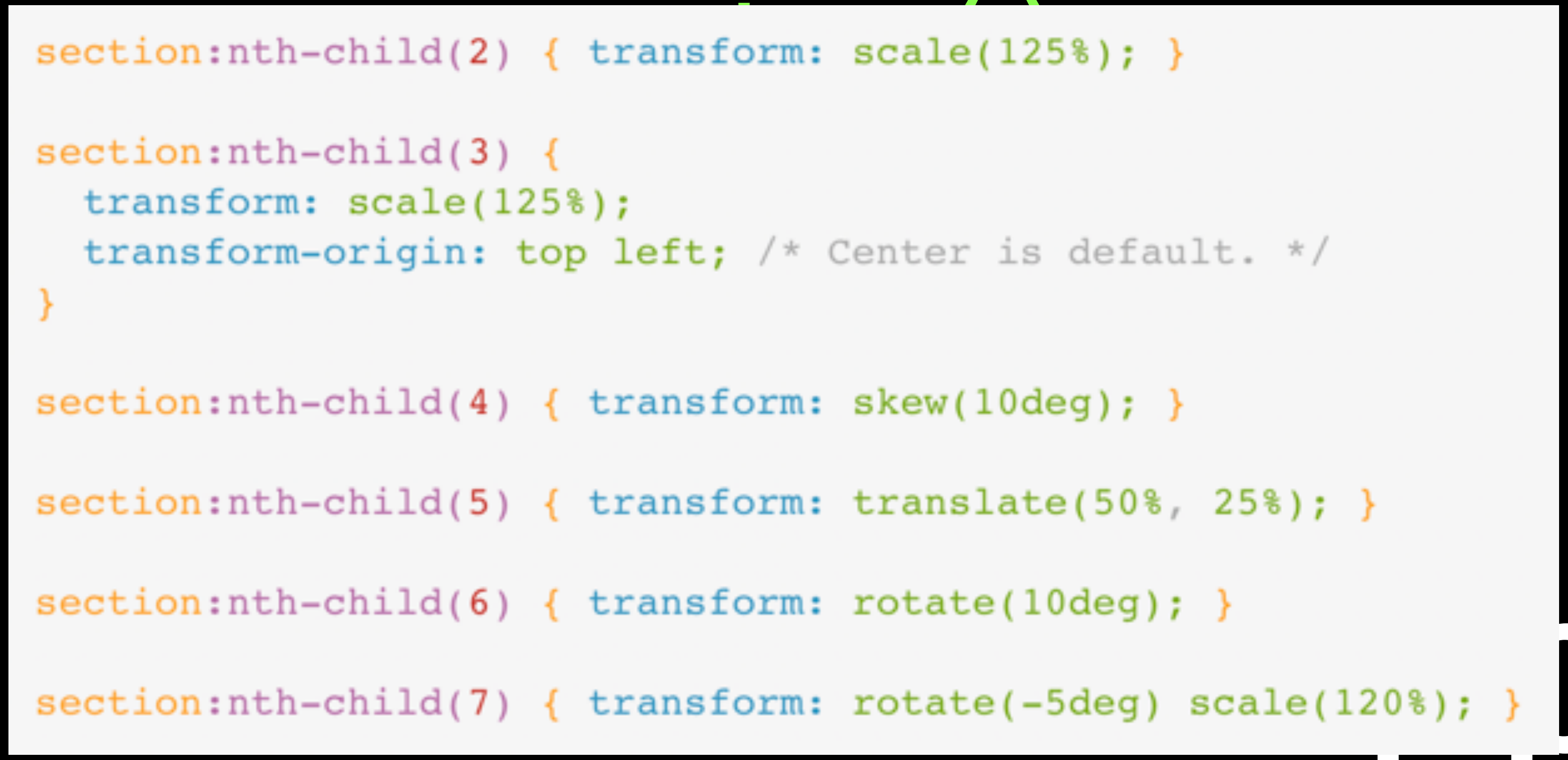
`translate() / translateX() / translateY() / translate3d()`	Moves an element left / right / up / down / in 3d space
`perspective()`	Sets the distance between the user and the element on the z-axis

You can apply multiple transforms at once with shorthand by separating them with a space.

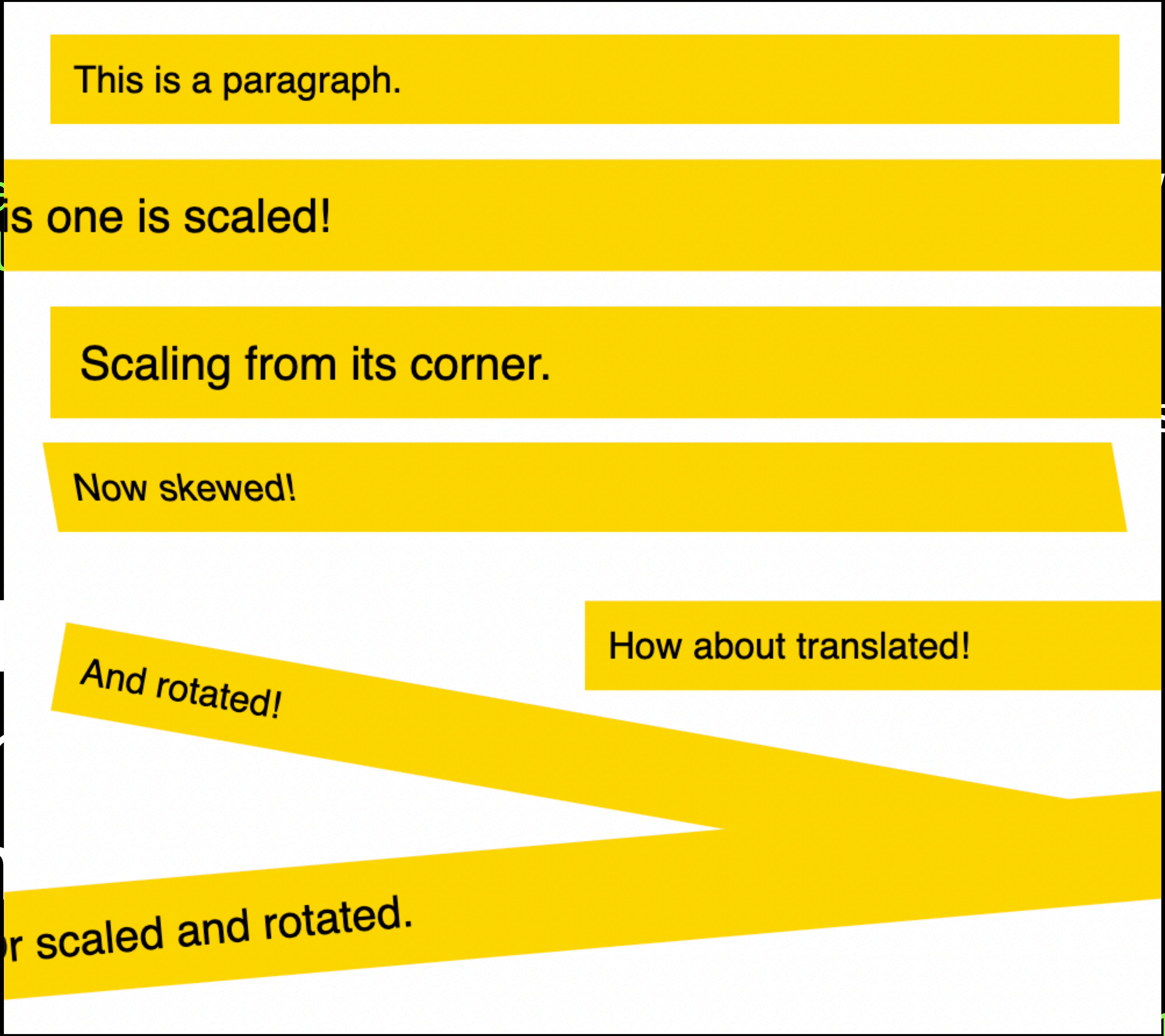
TRANSFORM

translate() / transla
translateY() / transl

once with sh
then



```
section:nth-child(2) { transform: scale(125%); }

section:nth-child(3) {
  transform: scale(125%);
  transform-origin: top left; /* Center is default. */
}

section:nth-child(4) { transform: skew(10deg); }

section:nth-child(5) { transform: translate(50%, 25%); }

section:nth-child(6) { transform: rotate(10deg); }

section:nth-child(7) { transform: rotate(-5deg) scale(120%); }
```



This is a paragraph.

is one is scaled!

Scaling from its corner.

Now skewed!

How about translated!

And rotated!

r scaled and rotated.

TRANSITIONS

Transitions allow for you to move between two CSS property values. We can give this transition a duration, acceleration, or delay.

TRANSITIONS

Transitions allow for you to move between two CSS property values. We can give this transition a duration, acceleration, or delay.

TRANSITION SHORTHAND

```
.example-transition {
 transition: all 2s 1s linear;
}
```

=

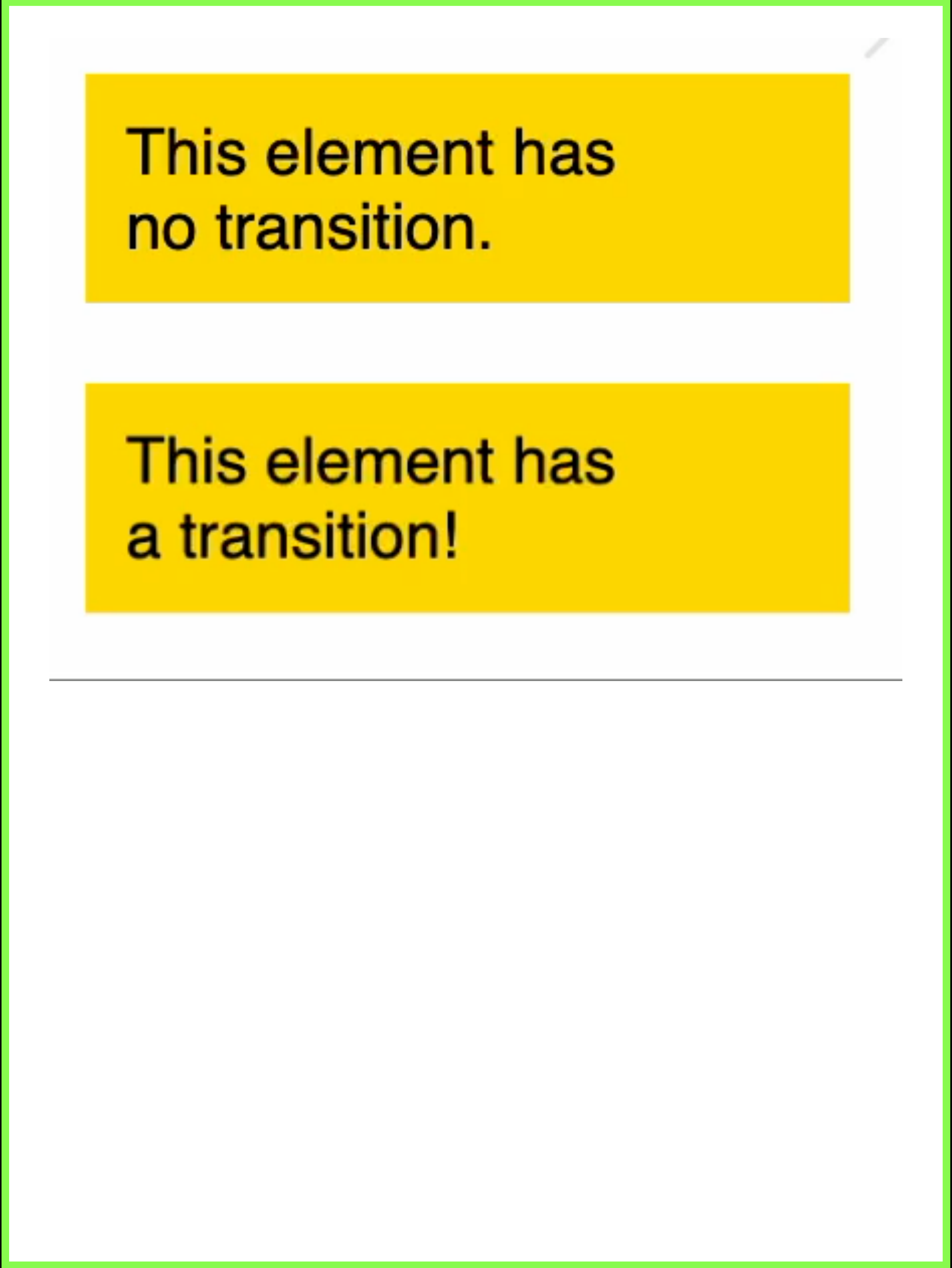
```
.example-transition {
 transition-delay: 1s;
 transition-duration: 2s;
 transition-property: all;
 transition-timing-function: linear;
}
```

```
.example-transition-combo {
 transition: background-color 2s
 linear, transform 1s ease-in-out;
}
```

=

```
.example-transition-combo {
 transition-duration: 2s, 1s;
 transition-property: background-
 color, transform;
 transition-timing-function: linear,
 ease-in-out;
}
```

TRANSITION


This element has no transition.

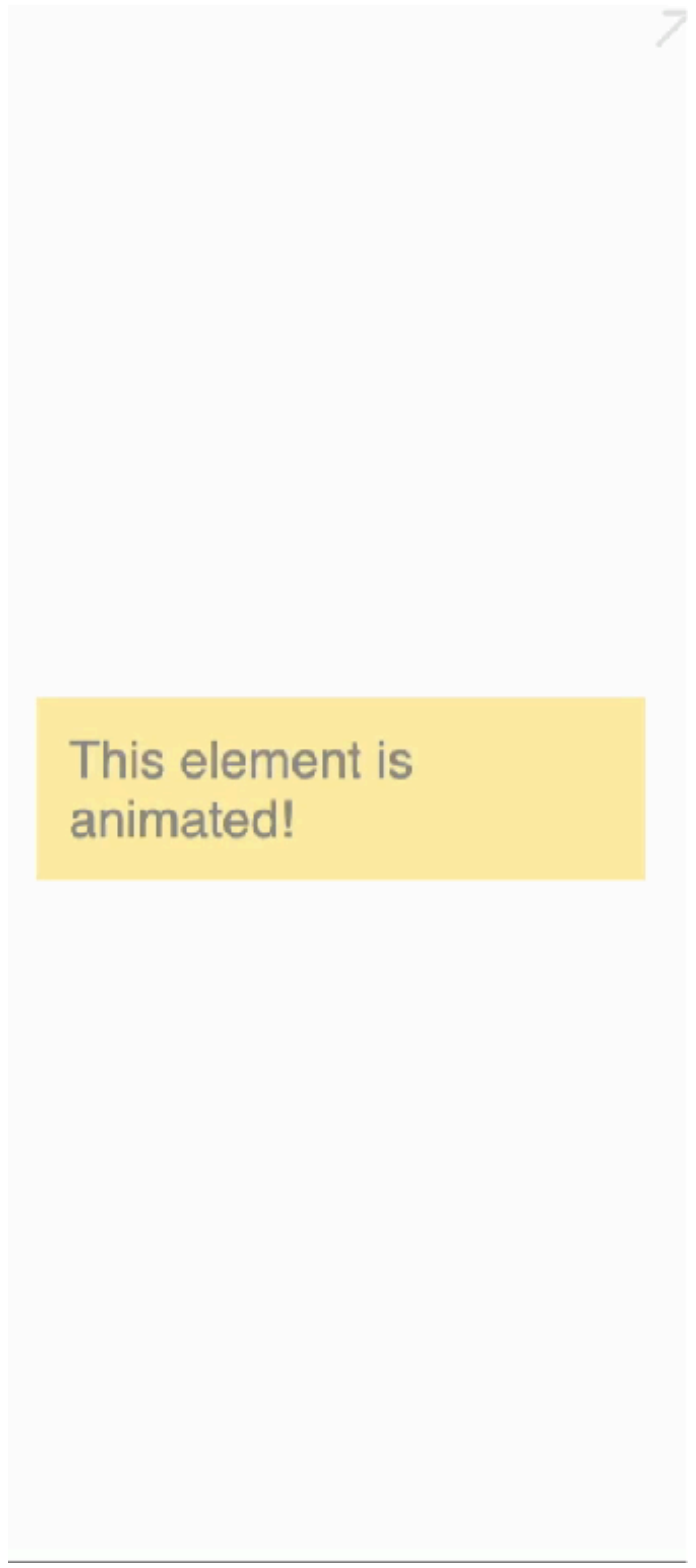
This element has a transition!

```
section:hover {
   background-color: aquamarine;
   transform: scale(105%);
}

section:nth-child(2) {
   transition-duration: 2s, 1s;
   transition-property: background-color,
   transform;
   transition-timing-function: linear, ease-
   in-out;
}
```

ANIMATE

LAST UP! Some shifts are just not realistic with transitions. In these cases, try @keyframe animations. You define an element’s initial state, and then essentially define a function to change it.

TRANSITION



```
@keyframes blinking {
  0% {
    opacity: 1;
    transform: translateY(0vh);
  }

  50% {
    opacity: 0;
    transform: translateY(75vh);
  }

  100% {
    opacity: 1;
    transform: translateY(0vh);
  }
}

section {
  animation: blinking 3s infinite ease-in-out;
}
```

TRANSITION SHORTHAND

```
section { animation: blinking
3s infinite ease-in-out; }
```

=

```
section {
  animation-duration: 3s;
  animation-iteration-count: infinite;
  animation-name: blinking;
  animation-timing-function: ease-in-out;
}
```

BREAK (15 MIN)

SESSION SIX
OCTOBER 3, 2023

THANK YOU

DAVIS SCHERER
DSCHERER@NEWSCHOOL.EDU