

Building Intelligent Layouts with CSS Grid

“Intrinsic Web Design”

Jen Simmons





css-irl.info



[About](#)



[Twitter](#)

28 Posts

[Becoming a Tech Speaker](#)

23 February, 2019

In this post I'm taking a brief diversion from my usual CSS topics to discuss how becoming a tech speaker has helped me develop my career...

[My CSS Grid Wishlist](#)

03 February, 2019

If you follow this blog you'll know I'm a big fan of CSS Grid, and without a doubt it's given us

[To Grid or to Flex?](#)

10 February, 2019

A recent Twitter thread started by Chris Coyier got me thinking about how people in general interpret the use cases for CSS Grid Layout...

[A Year of Utility Classes](#)

28 January, 2019

Last year at Mud we adopted a utility-first approach to CSS (also known as atomic CSS)

Why do we need Grid?

2D layout

“Flexbox is great for taking a bunch of oddly-sized things and returning the most reasonable layout for those things”

Rachel Andrew

Why do we need Grid?

Simpler HTML and CSS

Why do we need Grid?

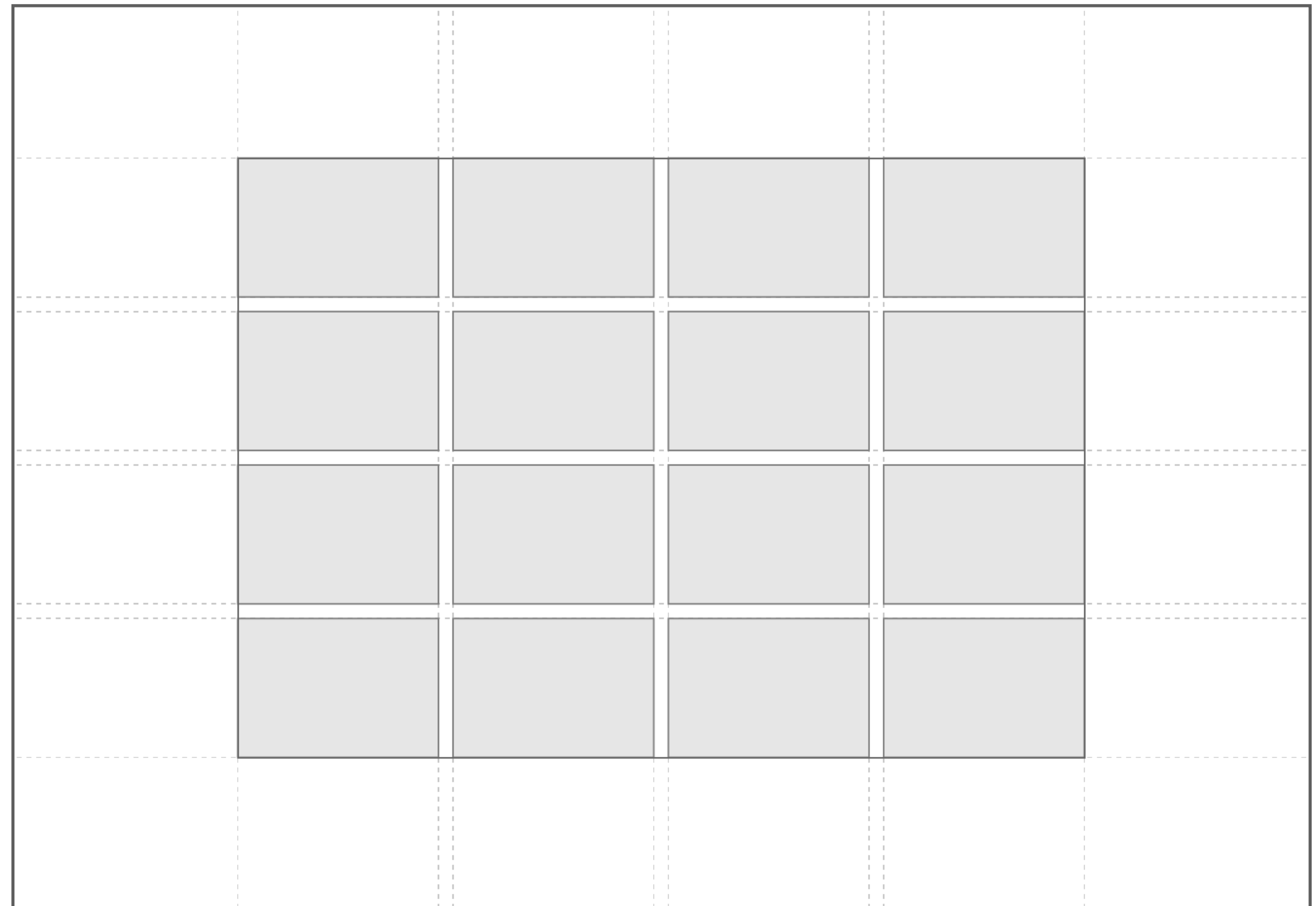
Complete control

Grid terminology

Grid terminology

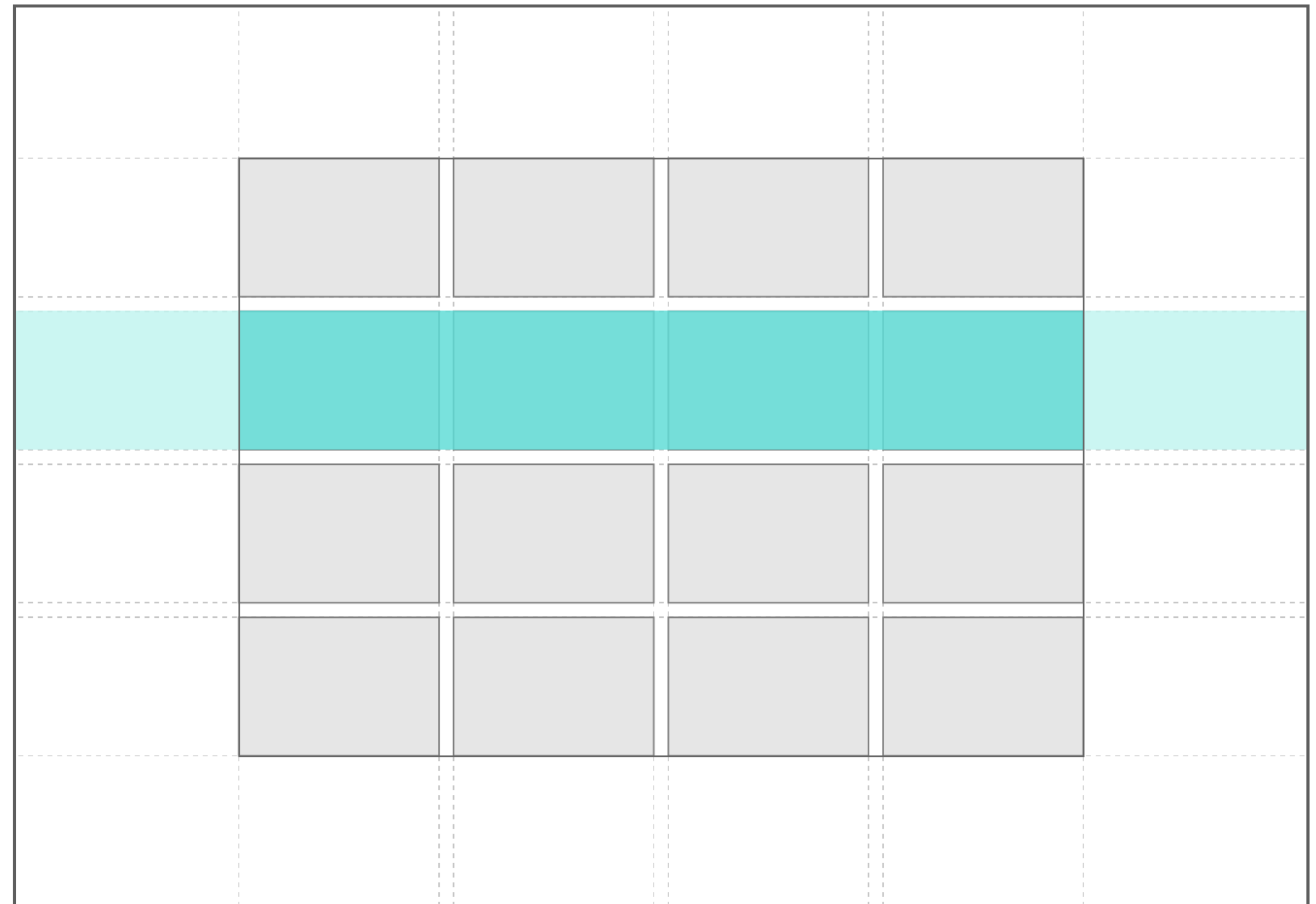
Grid

```
display: grid;
```



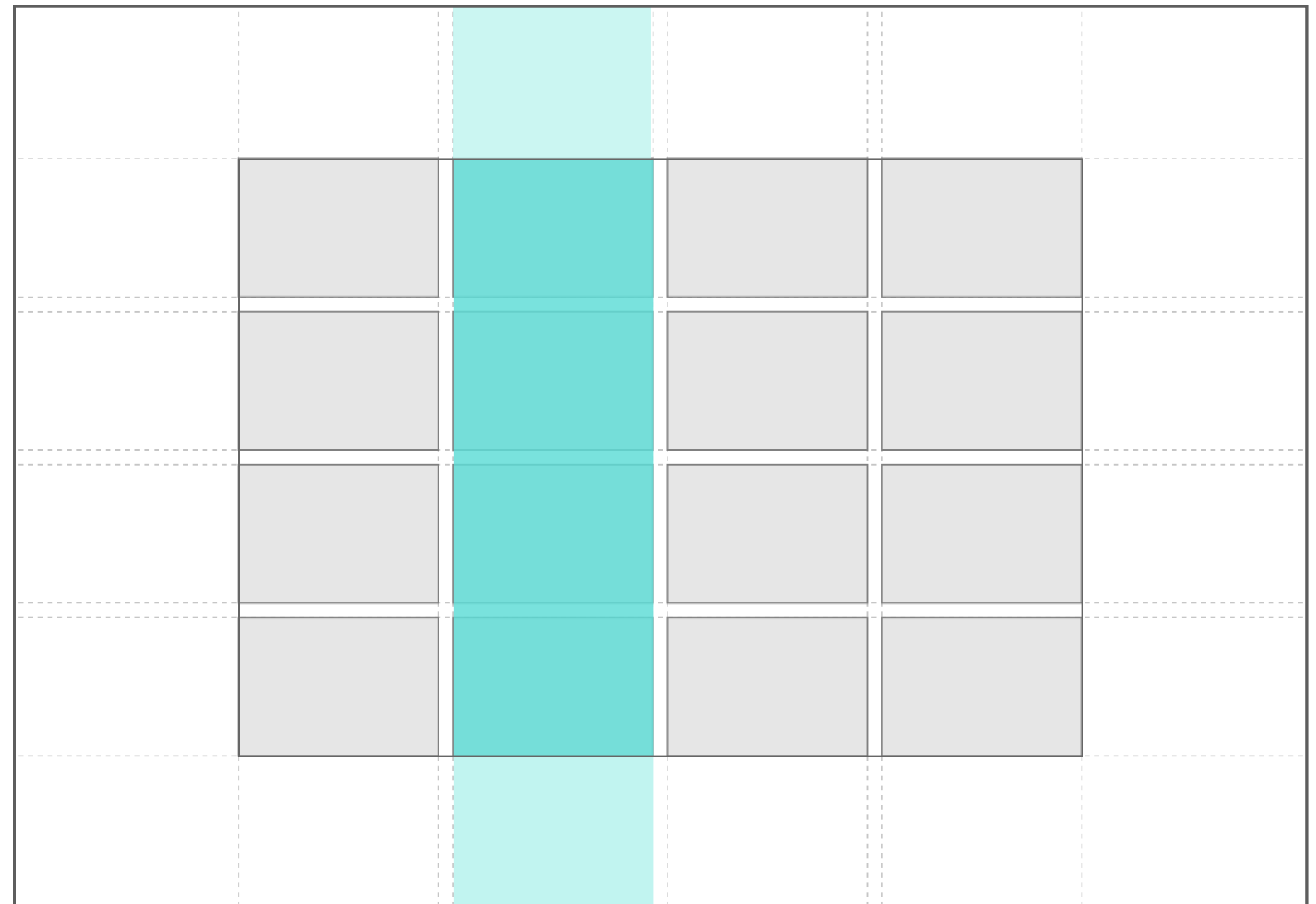
Tracks

`grid-template-rows`



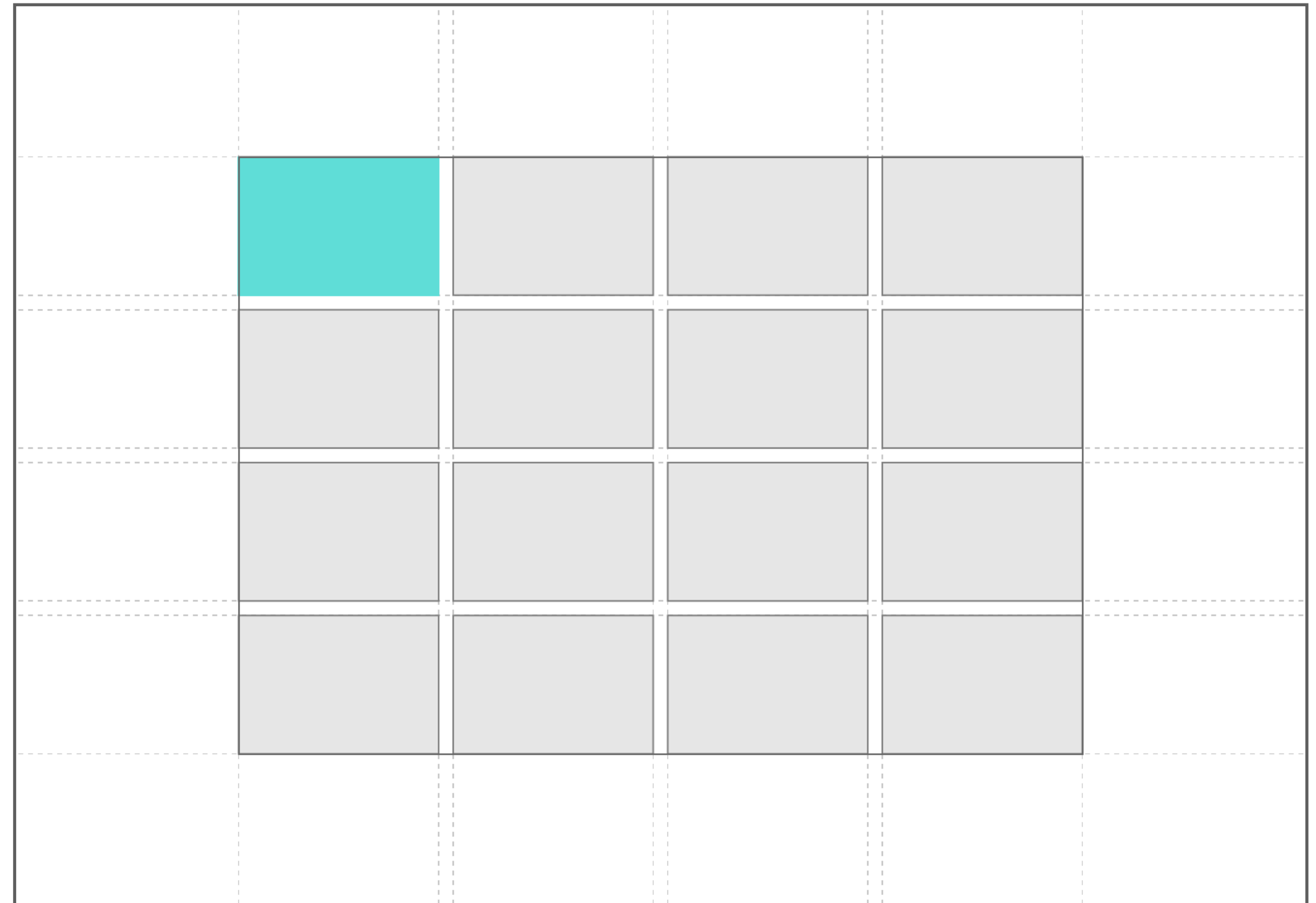
Tracks

`grid-template-columns`

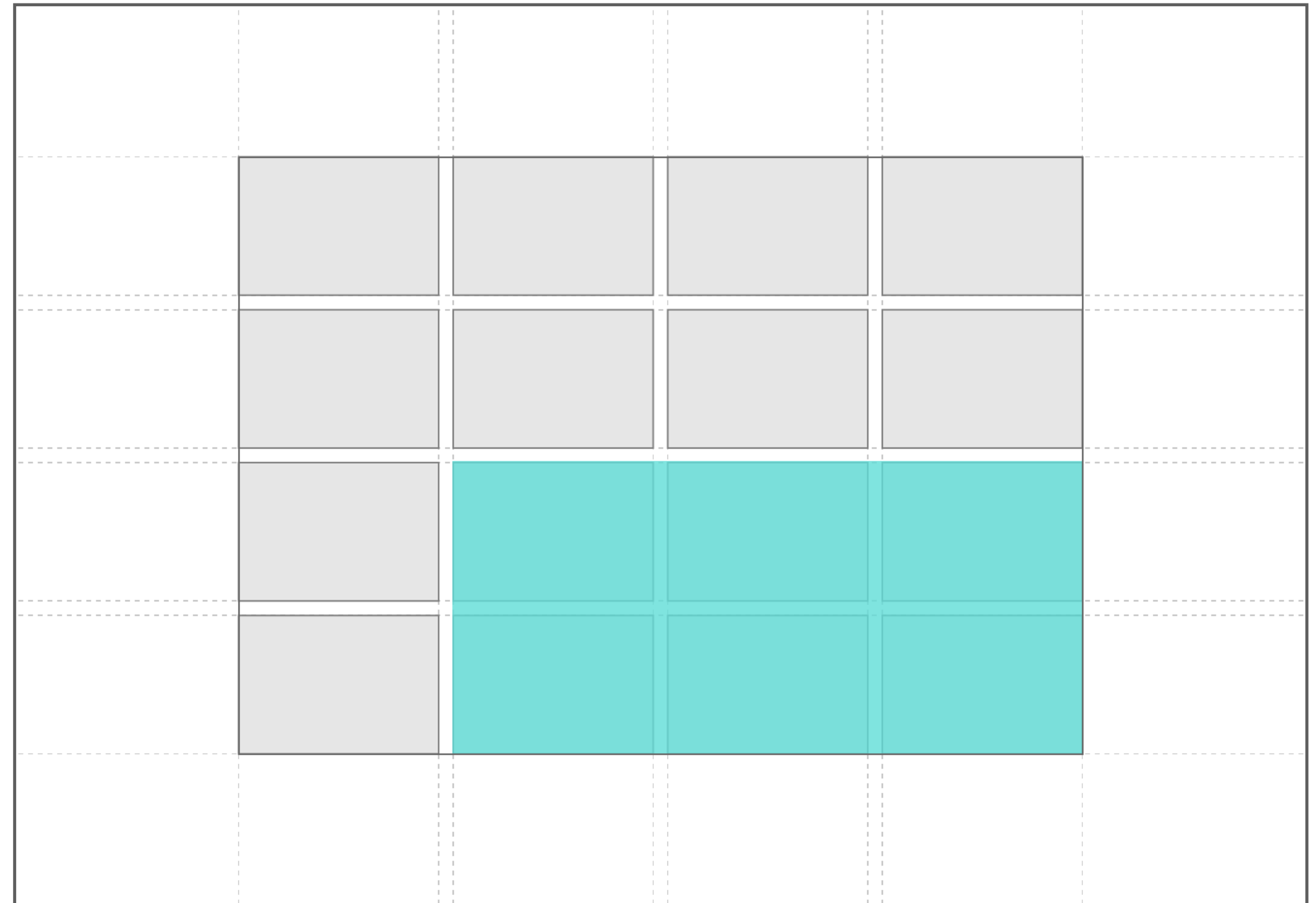


Grid terminology

Cells

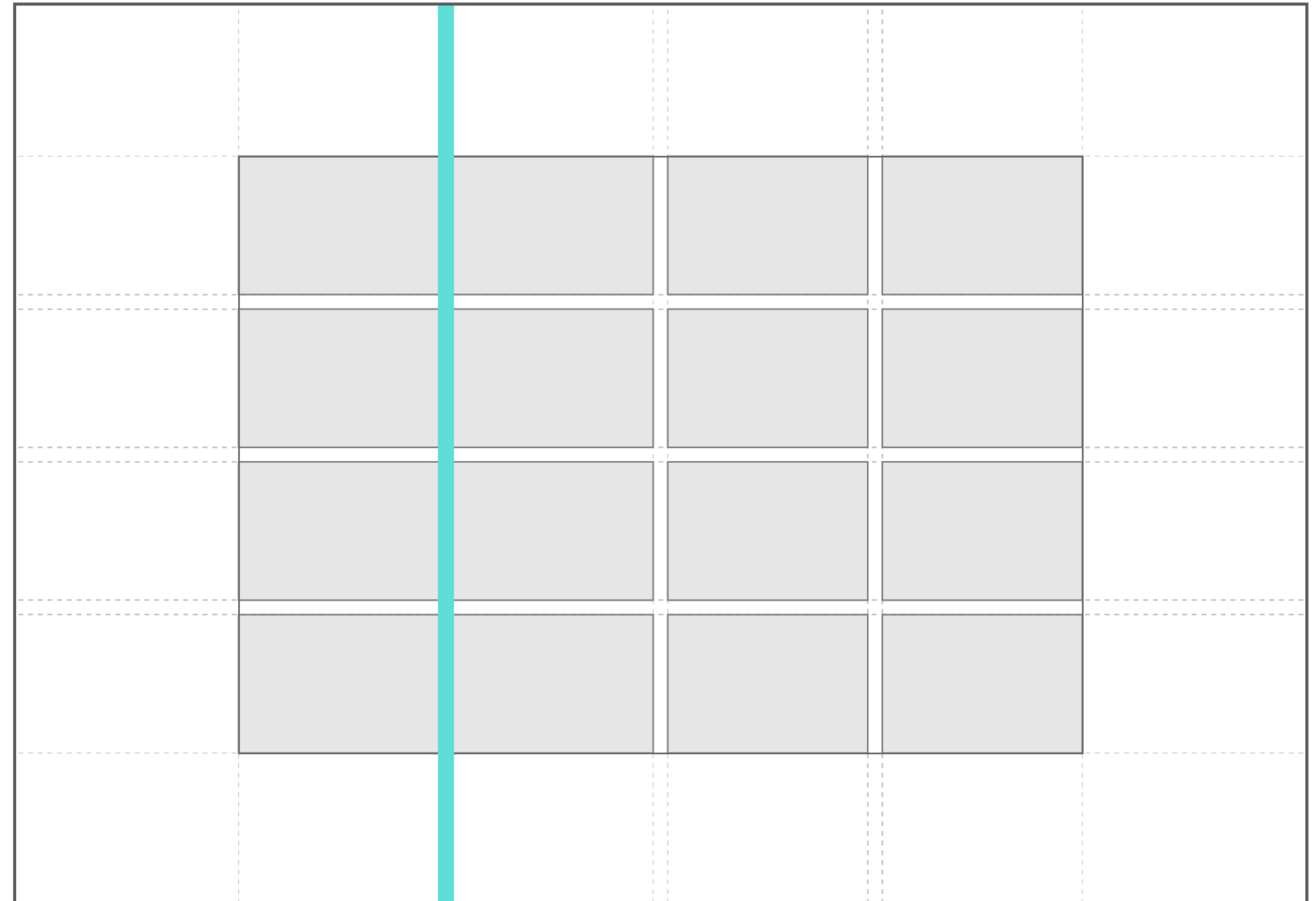


Grid areas



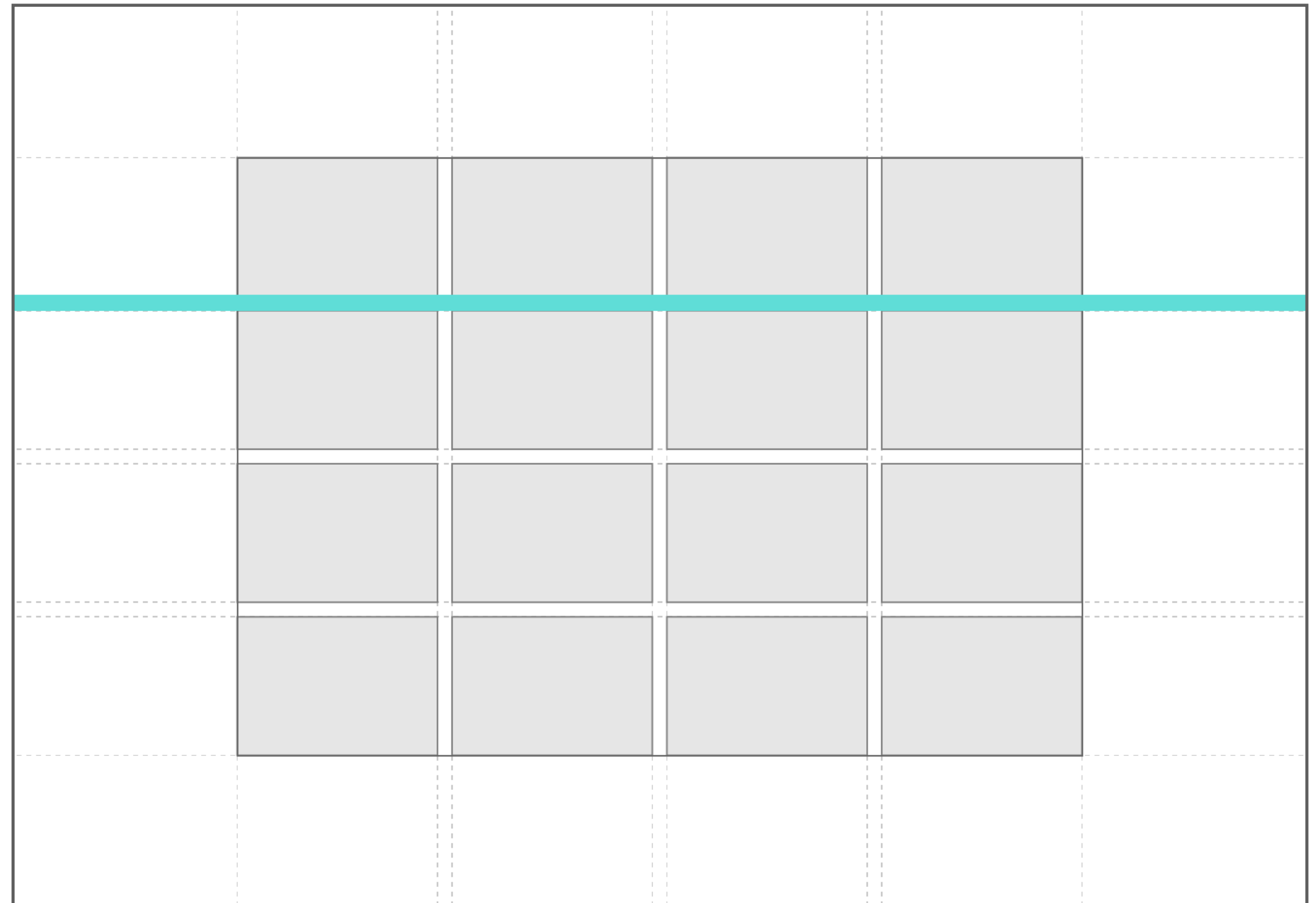
Gutters

`column-gap`
(Formerly `grid-column-gap`)



Gutters

`row-gap`
(Formerly `grid-row-gap`)



Defining a grid

CSS

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

codepen.io/michellebarker/pen/eLZwVg

CSS

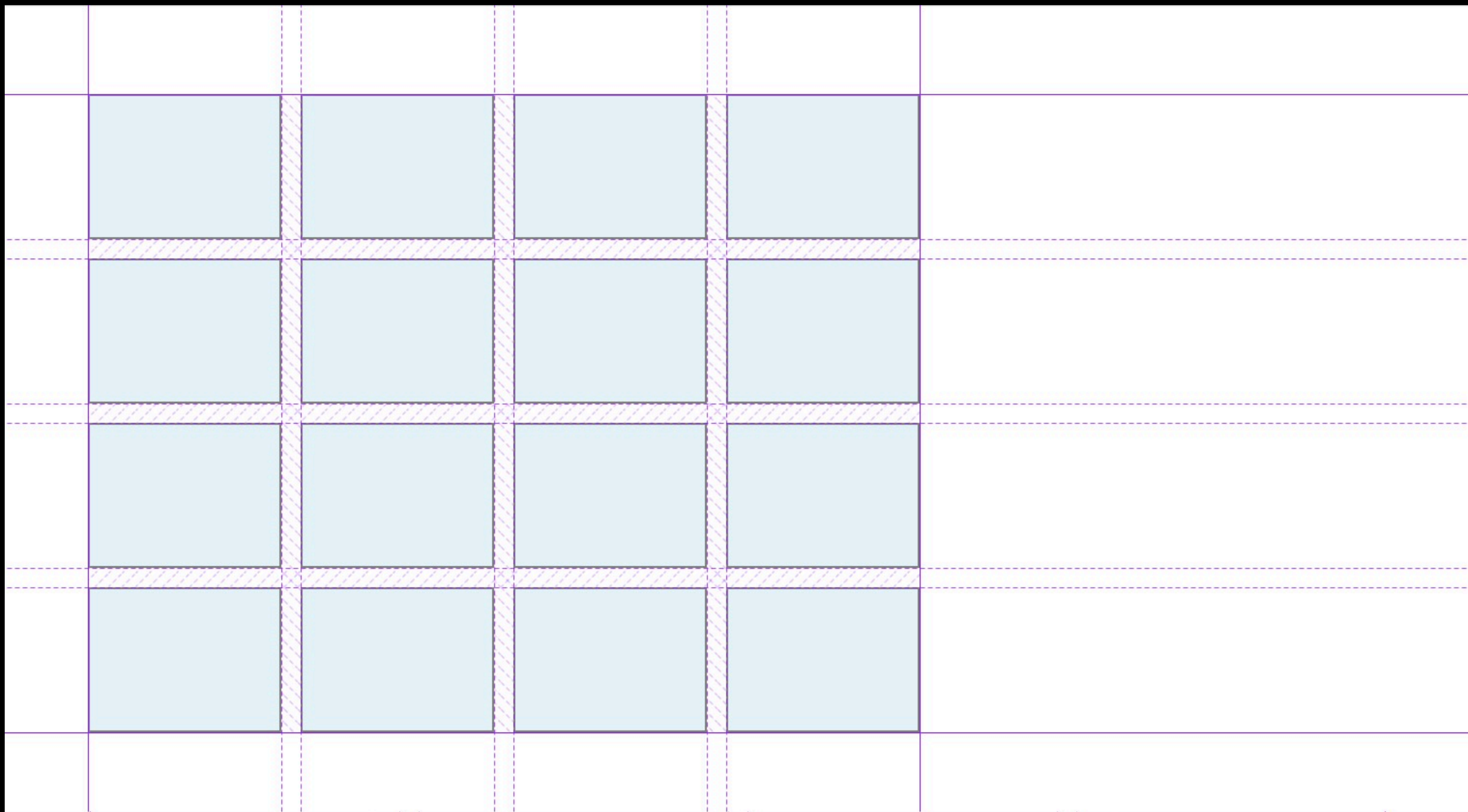
```
.grid {  
  display: grid;  
  grid-template-columns: 200px 200px 200px 200px;  
  grid-template-rows: 150px 150px 150px 150px;  
}
```

codepen.io/michellebarker/pen/eLZwVg

CSS

```
.grid {  
  display: grid;  
  grid-template-columns: 200px 200px 200px 200px;  
  grid-template-rows: 150px 150px 150px 150px;  
  column-gap: 20px;  
  row-gap: 20px;  
}
```

codepen.io/michellebarker/pen/eLZwVg



codepen.io/michellebarker/pen/eLZwVg

CSS

repeat()

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(4, 200px);  
  grid-template-rows: repeat(4, 150px);  
  grid-gap: 20px;  
}
```

codepen.io/michellebarker/pen/eLZwVg

CSS

Shorthand

```
.grid {  
  display: grid;  
  grid-template: repeat(4, 150px) / repeat(4, 200px);  
}
```

codepen.io/michellebarker/pen/eLZwVg

CSS

Shorthand

```
.grid {  
  display: grid;  
  grid-template: repeat(4, 150px) / repeat(4, 200px);  
  gap: 20px; // row-gap / column-gap  
}
```

codepen.io/michellebarker/pen/eLZwVg

CSS

Flexible units (fr)

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(4, 1fr);  
  grid-template-rows: repeat(4, 150px);  
  gap: 20px;  
}
```

codepen.io/michellebarker/pen/eLZwVg

Fr

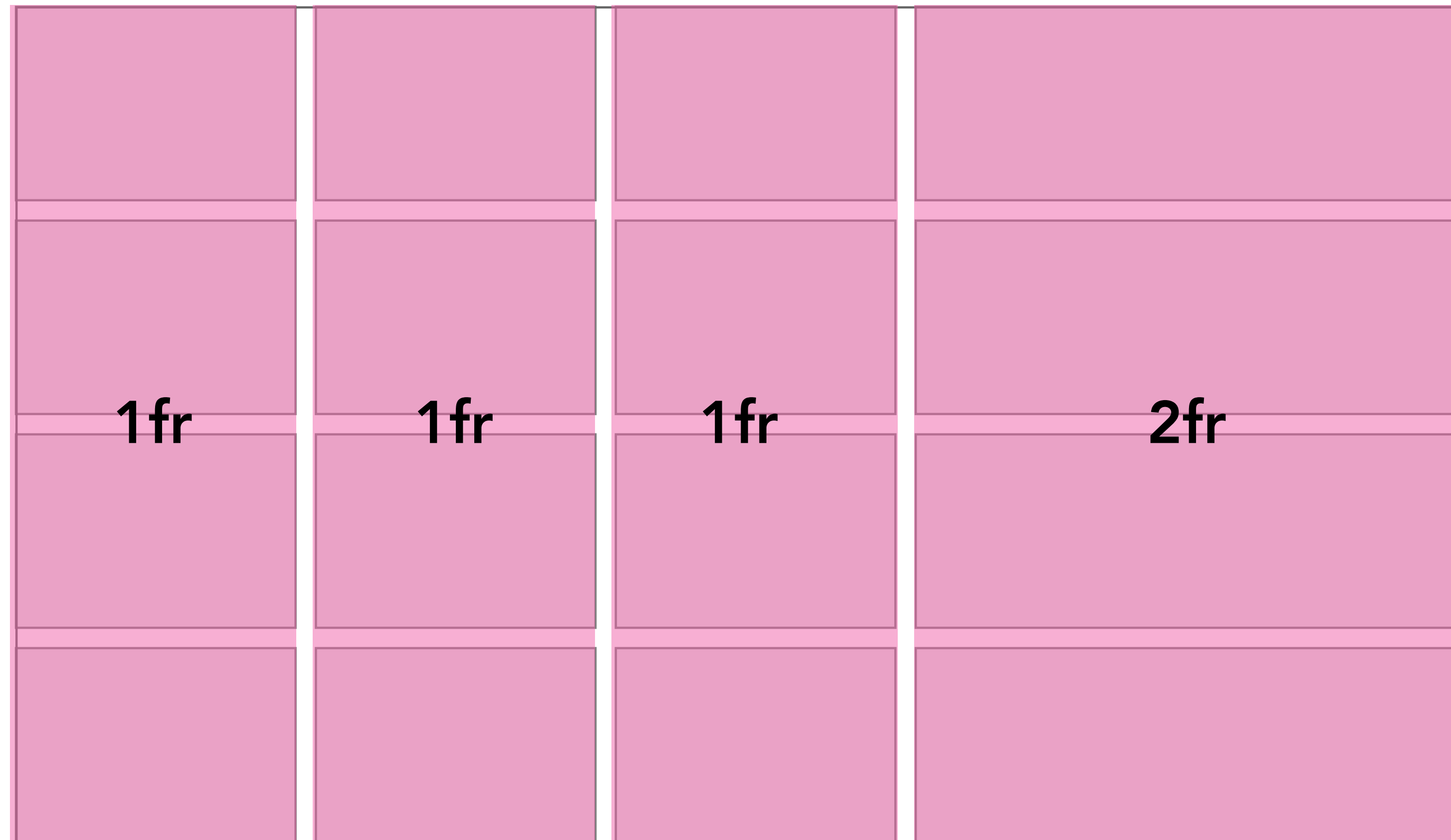
a fraction of the leftover space in the
grid container

CSS

Flexible units (fr)

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(4, 1fr);  
  grid-template-rows: repeat(4, 150px);  
  gap: 20px;  
}
```

codepen.io/michellebarker/pen/eLZwVg

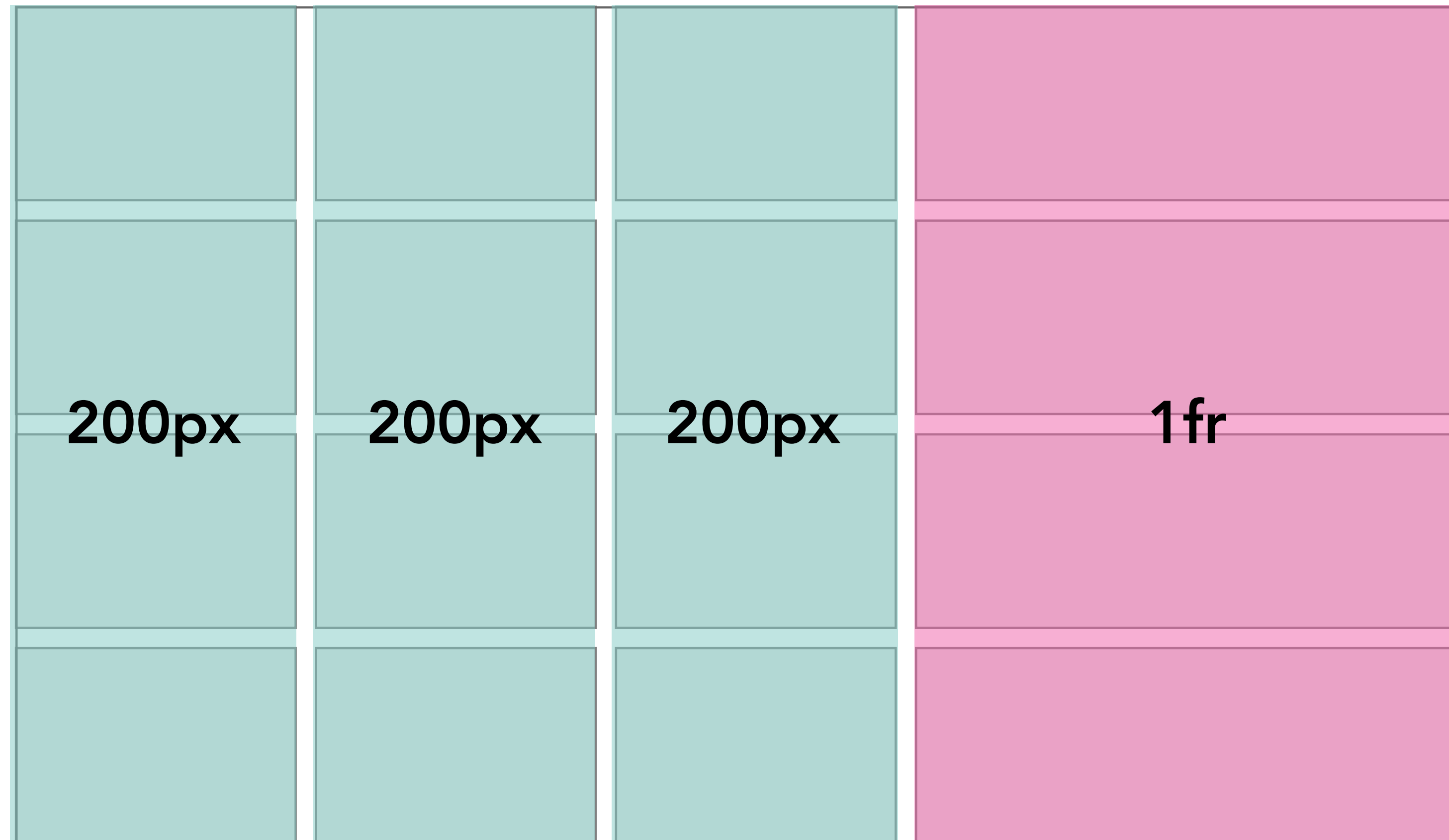


CSS

Flexible units (fr)

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(3, 200px) 1fr;  
  grid-template-rows: repeat(4, 150px);  
  gap: 20px;  
}
```

codepen.io/michellebarker/pen/eLZwVg



codepen.io/michellebarker/pen/wEdjqX

Implicit vs Explicit tracks

CSS

Implicit tracks

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(4, 1fr);  
  grid-auto-rows: 150px;  
  gap: 20px;  
}
```

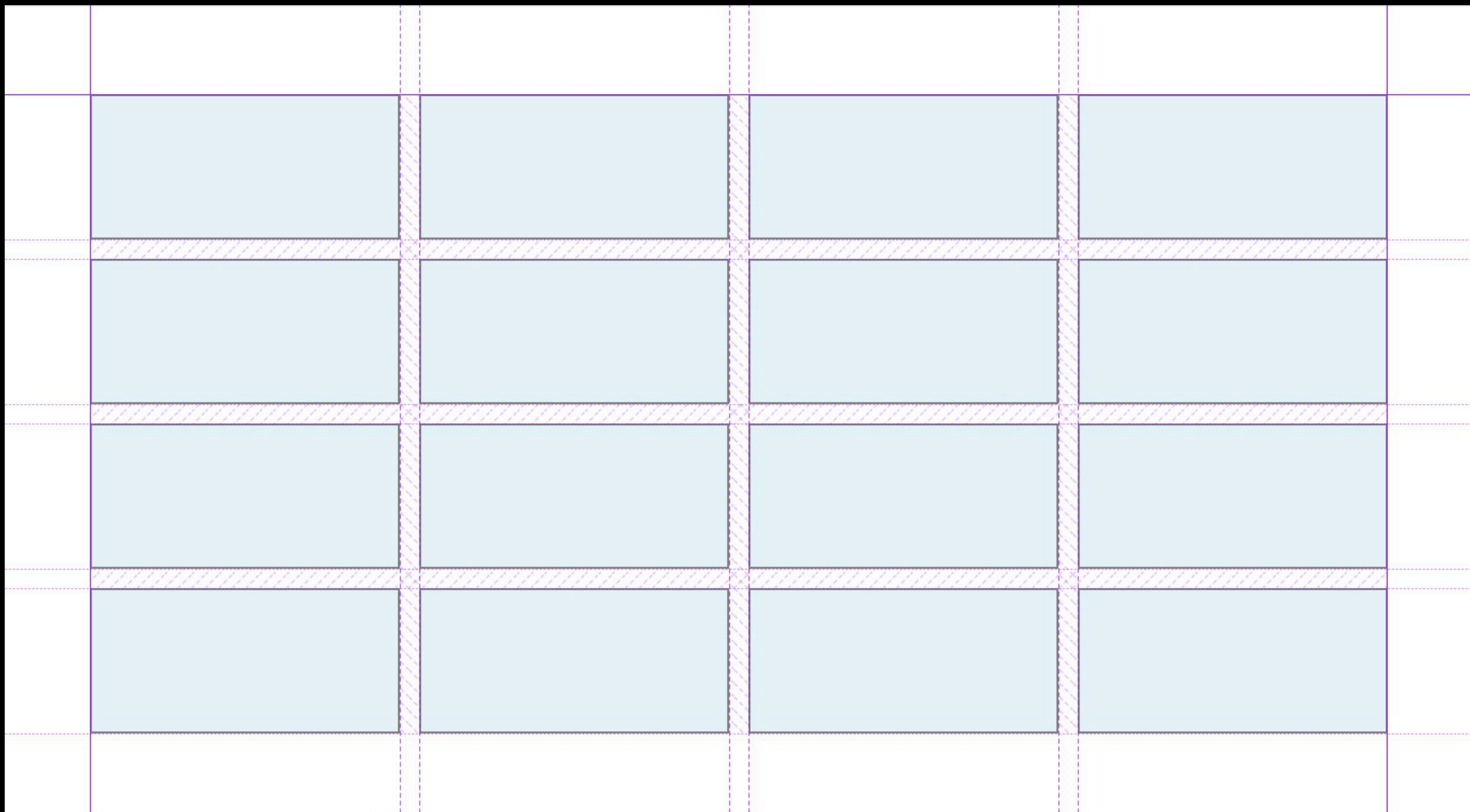
codepen.io/michellebarker/pen/eLZwVg

CSS

Implicit tracks

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(4, 1fr);  
  grid-template-rows: repeat(4, 150px);  
  grid-auto-rows: 150px;  
  gap: 20px;  
}
```

codepen.io/michellebarker/pen/eLZwVg



codepen.io/michellebarker/pen/eLZwVg

@mbarker_84

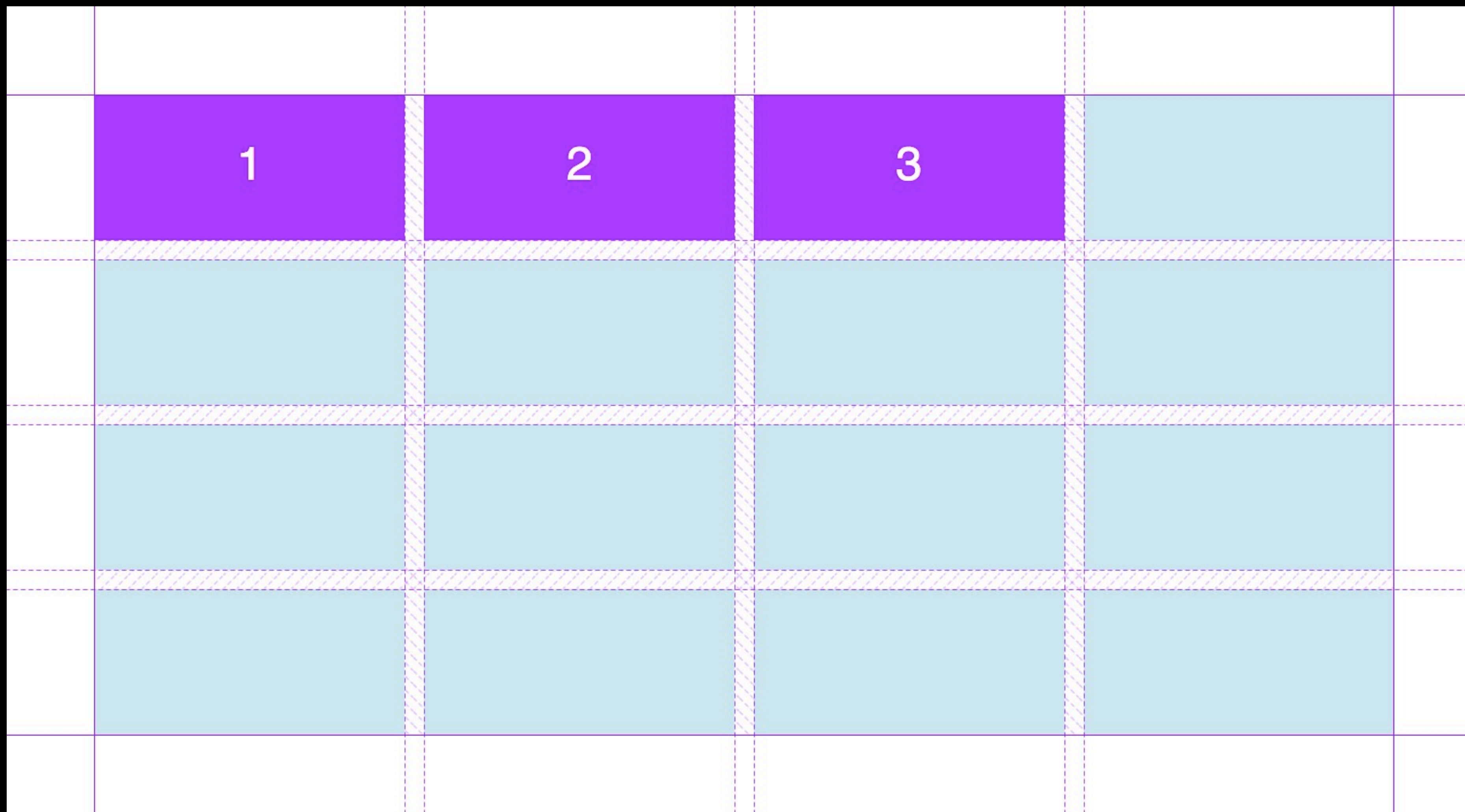
Placing items

HTML

Auto-placement

```
<div class="grid">  
  <div class="grid__item">1</div>  
  <div class="grid__item">2</div>  
  <div class="grid__item">3</div>  
</div>
```

codepen.io/michellebarker/pen/gdrVXP



codepen.io/michellebarker/pen/gdrVXP

<header>

<main>

<aside>

HTML

Placing items by grid line

```
<div class="grid">  
  <header></header>  
  <main></main>  
  <aside></aside>  
</div>
```


	1	2	3	4	5
1					
2					
3					
4					
5					

codepen.io/michellebarker/pen/YOqmjP

CSS

Placing items by grid line

```
header {  
  grid-column-start: 1;  
  grid-column-end: 5;  
}
```

codepen.io/michellebarker/pen/YOqmjP

	1	2	3	4	5
1					
2					
3					
4					
5					

codepen.io/michellebarker/pen/YOqmjP

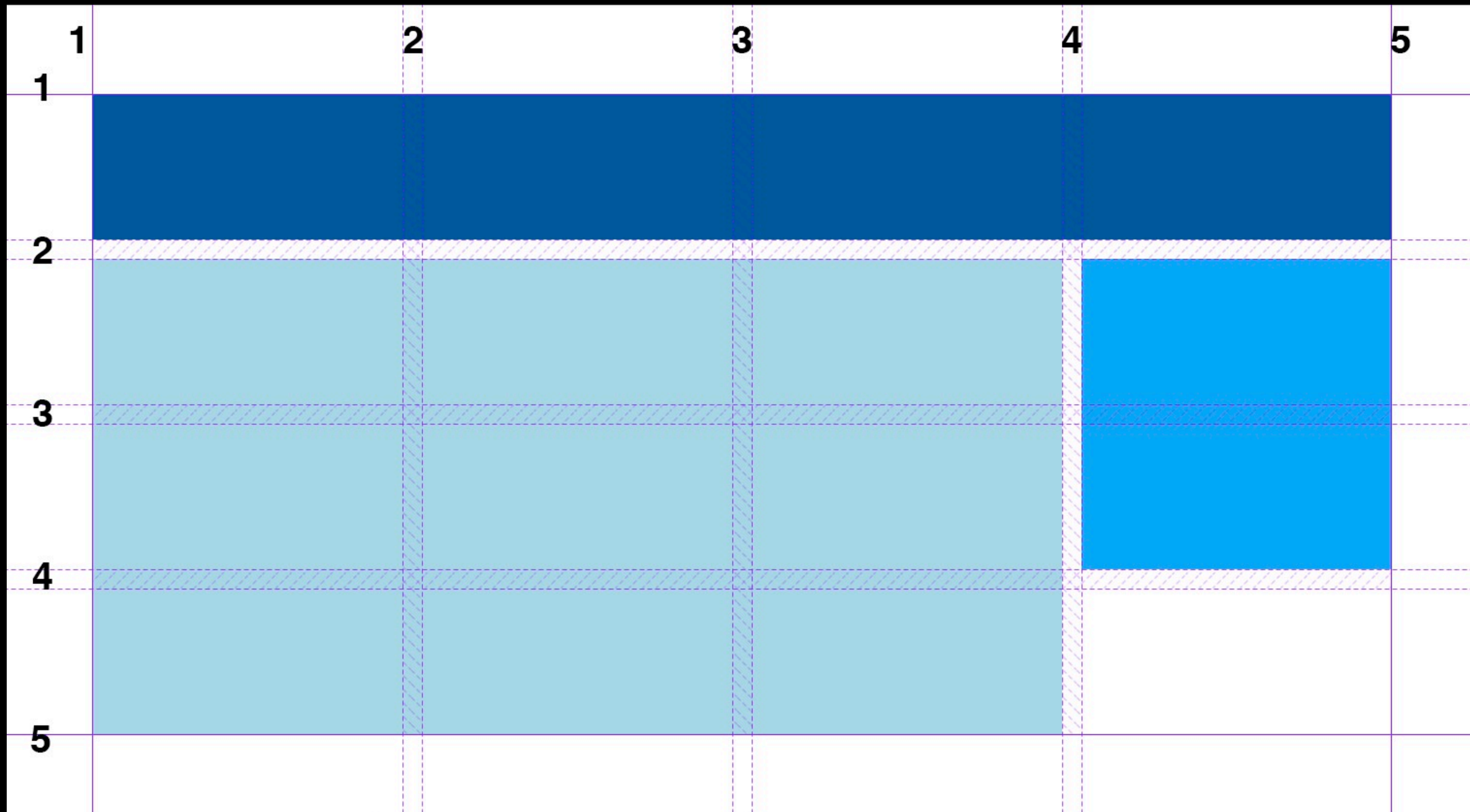
CSS

Placing items by grid line

```
main {  
  grid-column-start: 1;  
  grid-column-end: 4;  
  grid-row-start: 2;  
  grid-row-end: 5;  
}
```

```
aside {  
  grid-column-start: 4;  
  grid-column-end: 5;  
  grid-row-start: 2;  
  grid-row-end: 4;  
}
```

codepen.io/michellebarker/pen/YOqmjP



codepen.io/michellebarker/pen/YOqmjP

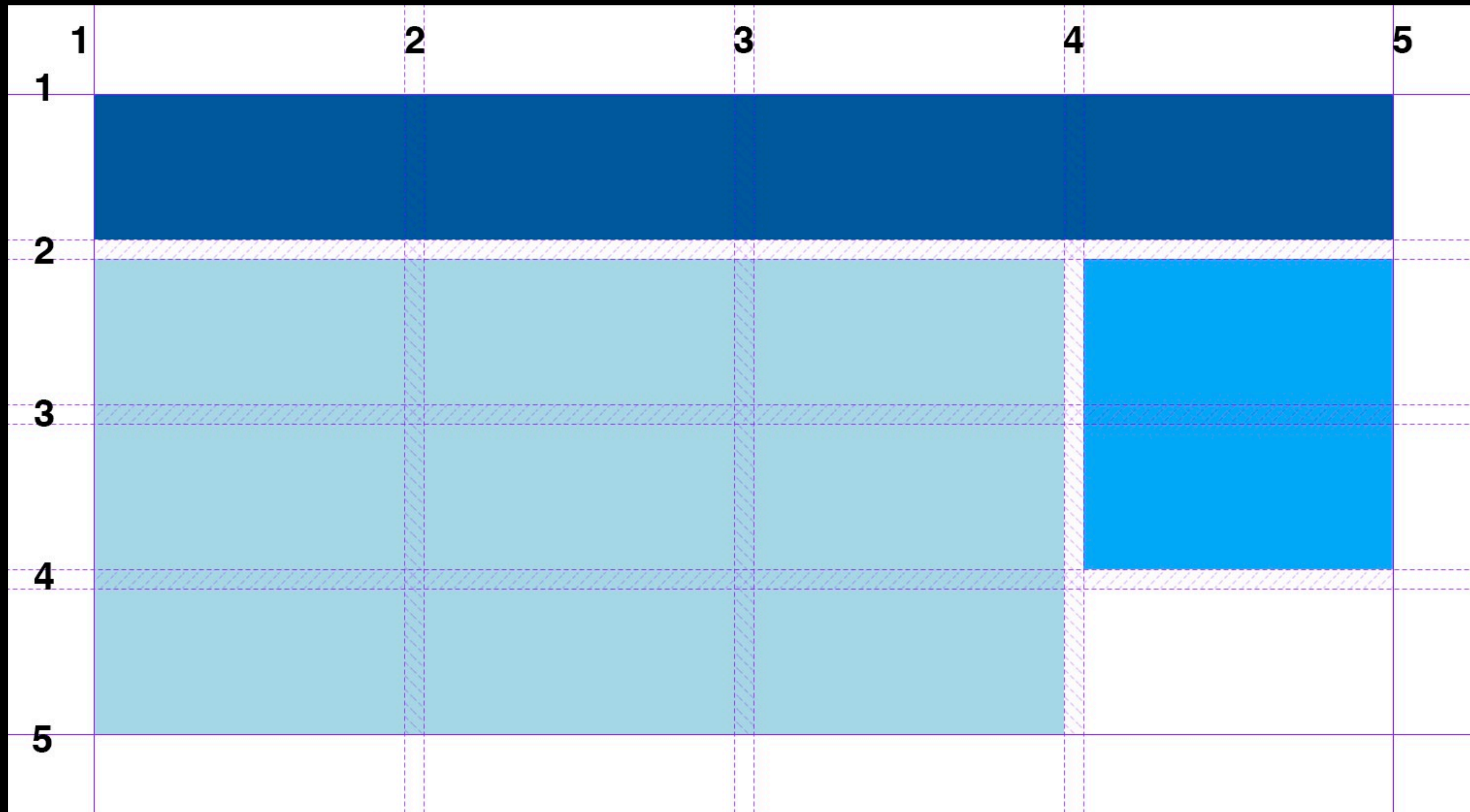
CSS

```
header {  
  grid-column: 1 / 5;  
}
```

```
main {  
  grid-column: 1 / 4;  
  grid-row: 2 / 5;  
}
```

```
aside {  
  // grid-column: 4 / 5; – unnecessary as will be  
  auto-placed  
  grid-row: span 2;  
}
```

codepen.io/michellebarker/pen/YOqmjP



codepen.io/michellebarker/pen/YOqmjP

CSS

Start line / end line

```
main {  
  grid-column: 1 / 4;  
  grid-row: 2 / 4;  
}
```

codepen.io/michellebarker/pen/YOqmjP

CSS

Span (with auto-placement)

```
main {  
  grid-column: span 3;  
  grid-row: 2 / 4;  
}
```

codepen.io/michellebarker/pen/YOqmjP

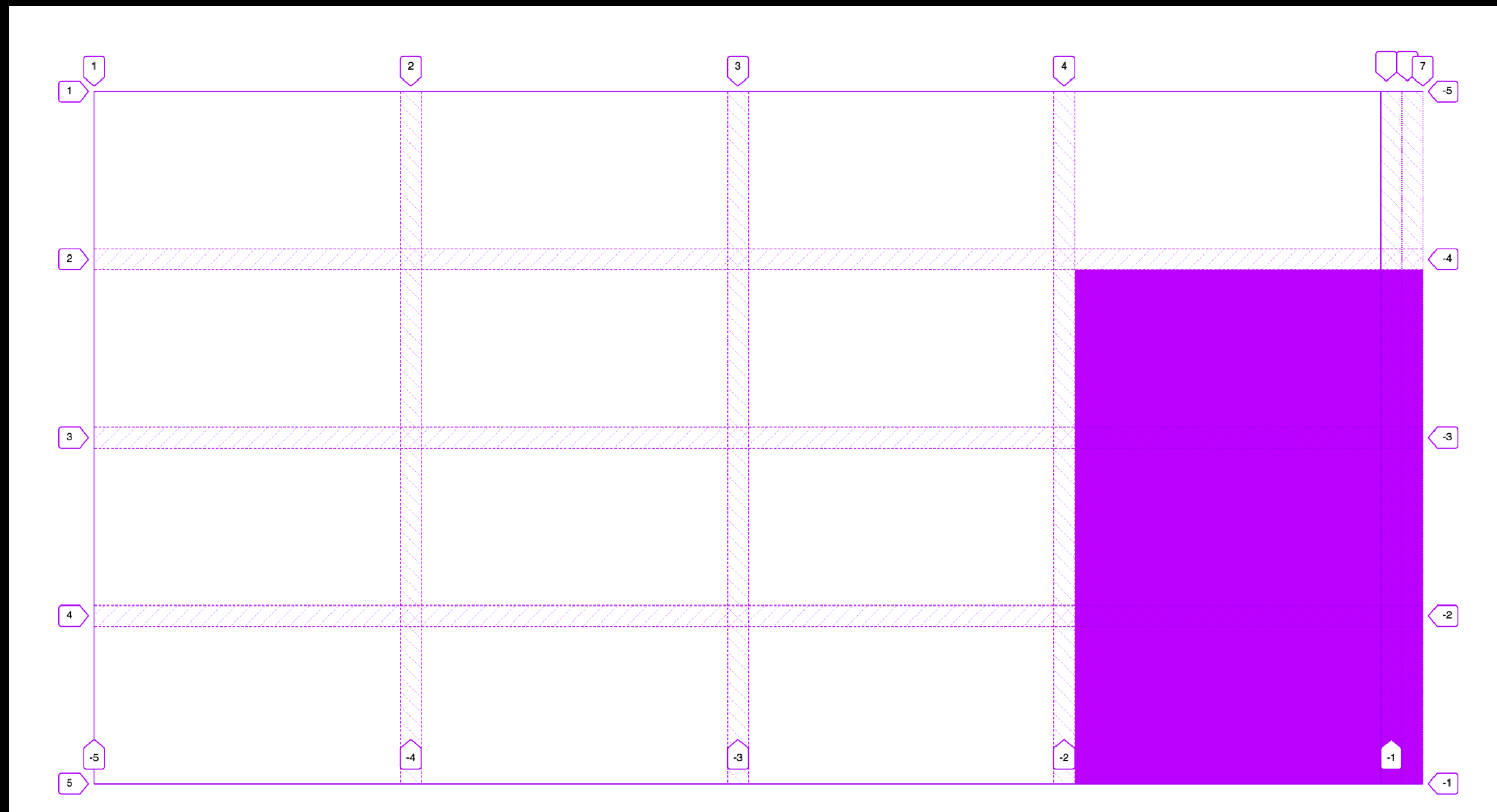
CSS

Start line / span

```
main {  
  grid-column: 1 / span 3;  
  grid-row: 2 / 4;  
}
```

codepen.io/michellebarker/pen/YOQLLZ

```
main {  
  grid-column: 4 / span 3;  
}
```



codepen.io/michellebarker/pen/rRoLRe

CSS

Span / end line

```
main {  
  grid-column: span 3 / 4;  
  grid-row: 2 / 4;  
}
```

codepen.io/michellebarker/pen/YOqmjP

CSS

Naming grid lines

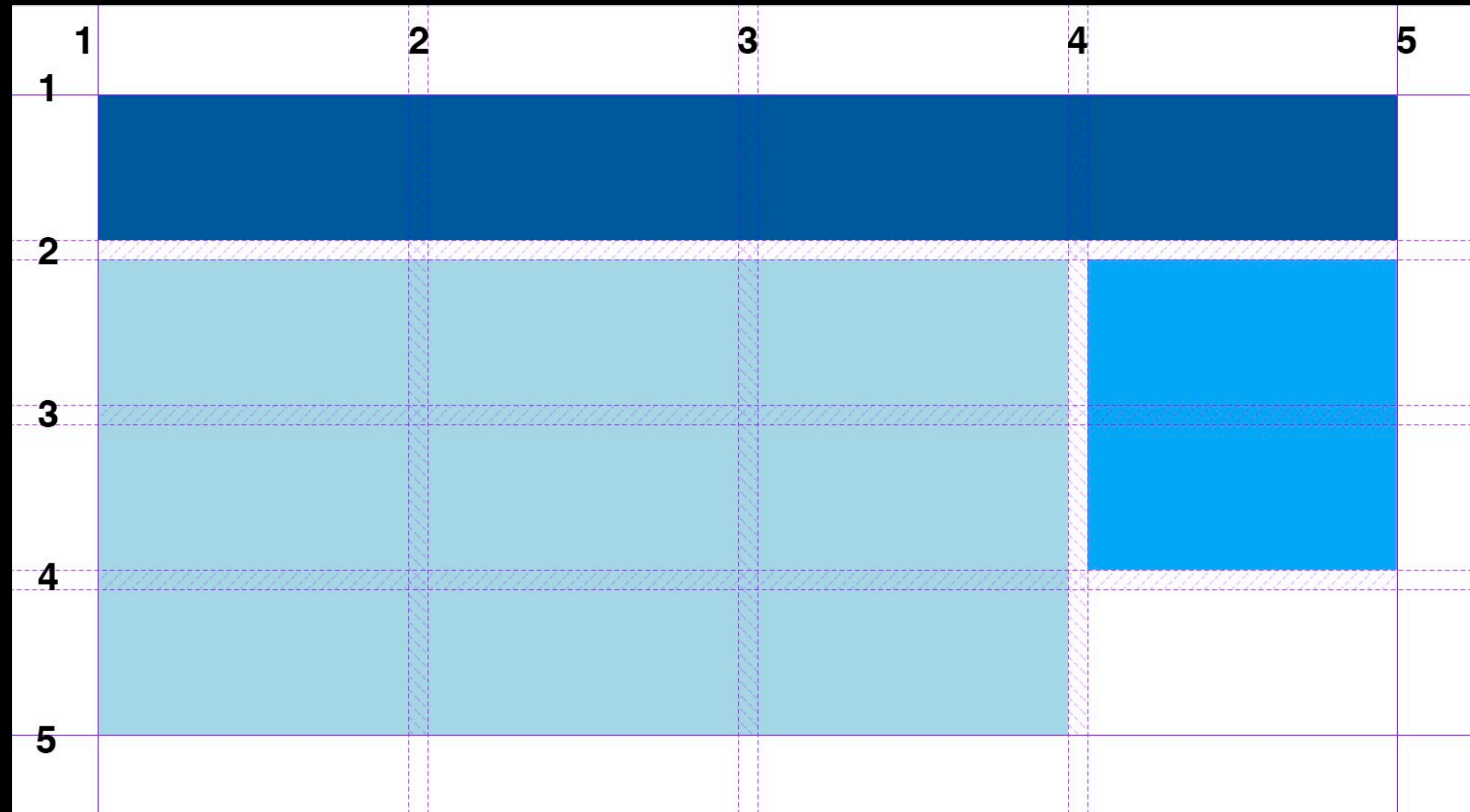
```
.grid {  
  display: grid;  
  grid-template-columns:  
    [header-start main-start]  
    repeat(3, 1fr)  
    [main-end] 1fr [header-end];  
  grid-auto-rows: 150px;  
}
```

codepen.io/michellebarker/pen/OobJaa

header-start
main-start

main-end

header-end



codepen.io/michellebarker/pen/OobJaa

CSS

```
header {  
  grid-column: header;  
}
```

```
main {  
  grid-column: main;  
  grid-row: span 3;  
}
```

codepen.io/michellebarker/pen/rqZevv

CSS

grid-template-areas

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(4, 1fr);  
  grid-auto-rows: 150px;  
  
  grid-template-areas: "h h h h"  
                      "m m m a"  
                      "m m m a"  
                      "m m m .";  
  
  gap: 20px;  
}
```

CSS

grid-template-areas

```
header {  
  grid-area: h;  
}
```

```
main {  
  grid-area: m;  
}
```

```
aside {  
  grid-area: a;  
}
```

CSS Grid for Complex Components

A Case Study



SERVICES

We combine the most advanced technology with uniquely experienced talent, providing a bespoke service and a truly exceptional post-production experience every time.



FACILITIES

Built to satisfy our clients' every need, our wealth of facilities includes ADR recording, multiple re-recording stages, 4K screening room, and 50+ cutting rooms and production offices.





SERVICES

We combine the most advanced technology with uniquely experienced talent, providing a bespoke service and a truly exceptional post-production experience every time.



FACILITIES

Built to satisfy our clients' every need, our wealth of facilities includes ADR recording, multiple re-recording stages, 4K screening room, and 50+ cutting rooms and production offices.





We combine the most advanced technology with uniquely experienced talent, providing a bespoke service and a truly exceptional post-production experience every time.



SERVICES



HTML

```
<div class="grid">  
  <h2 class="grid__heading"></h2>  
  <figure class="grid__image"></figure>  
  <div class="grid__body"></div>  
</div>
```

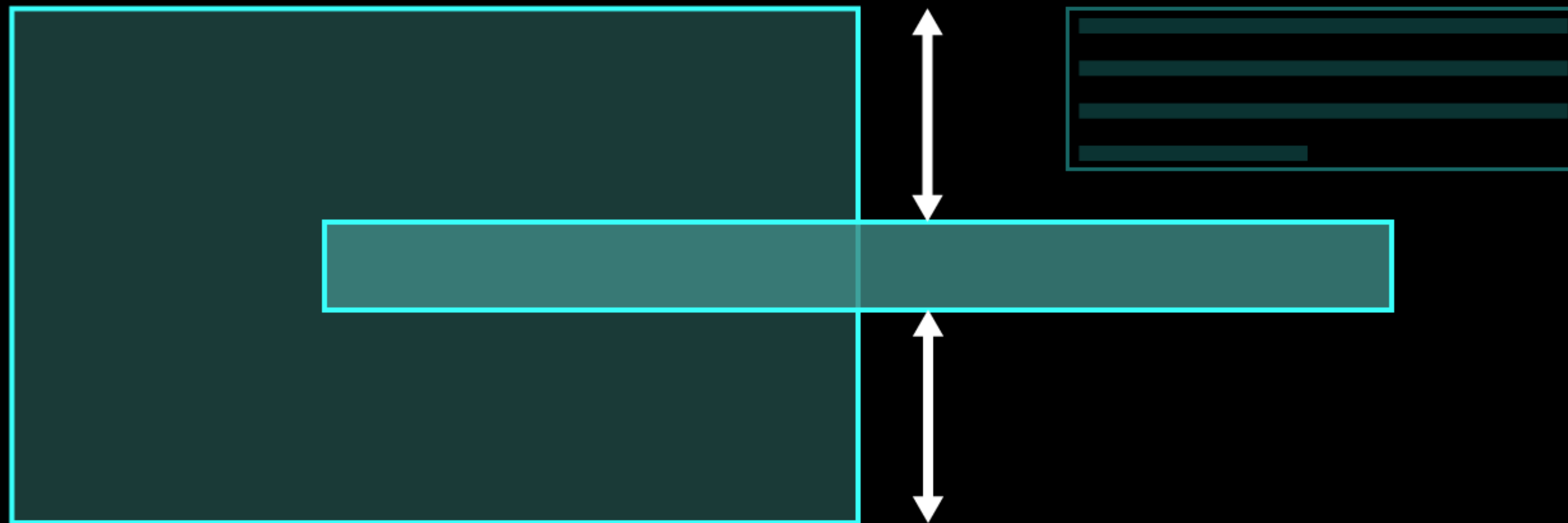
Some rules



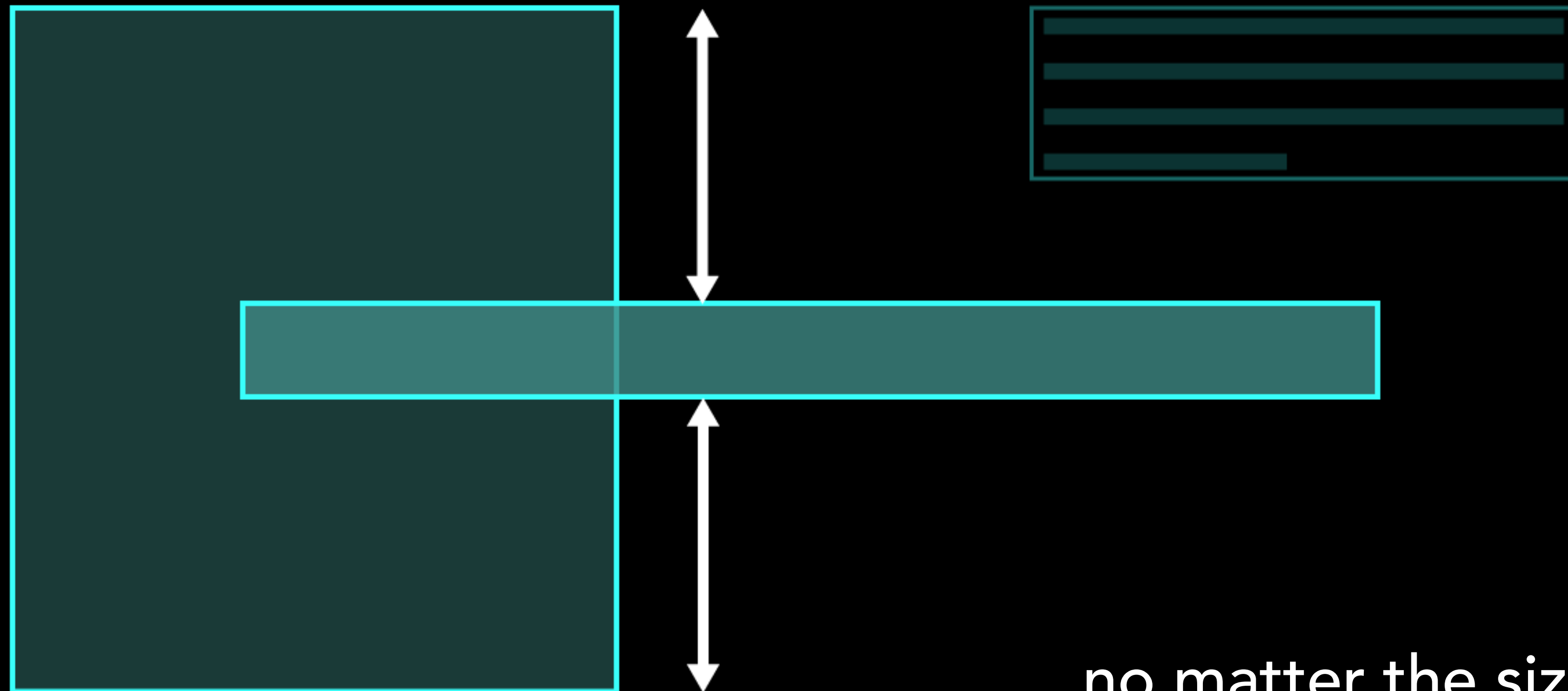
1. Dynamic content



2. The heading must always be vertically centred over the image

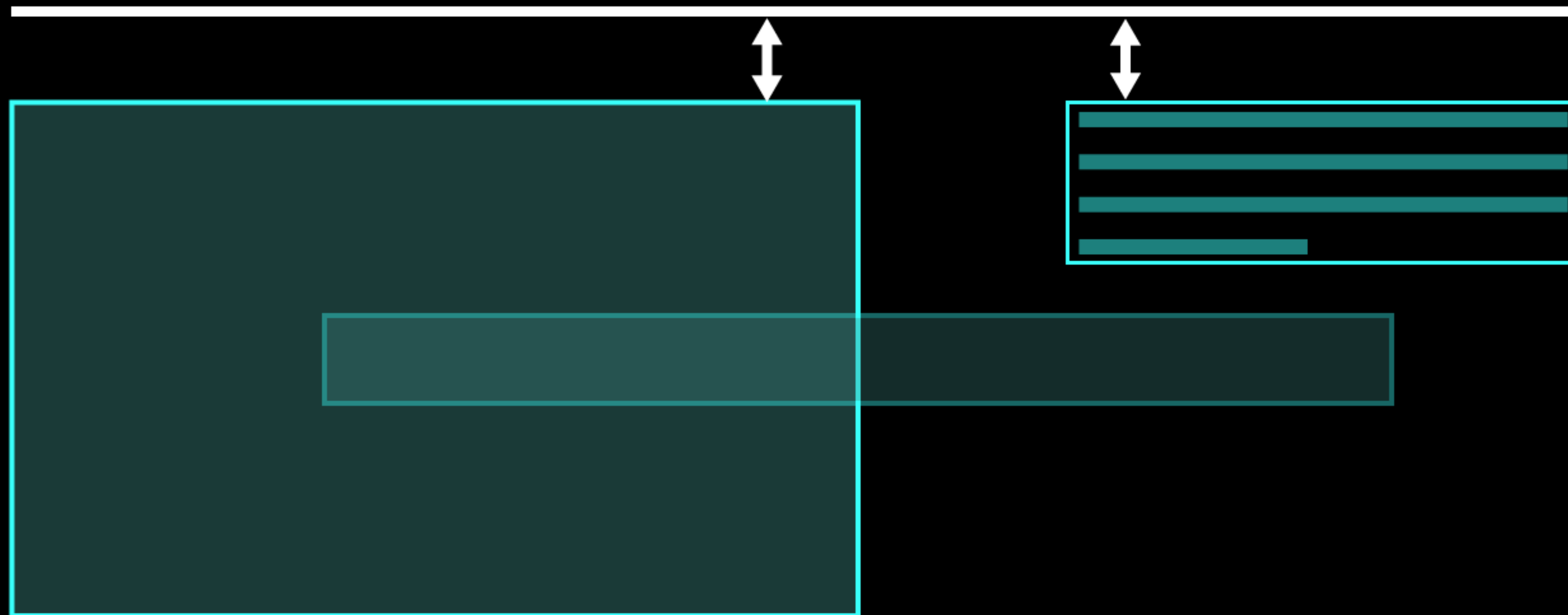


2. The heading must always be vertically centred over the image

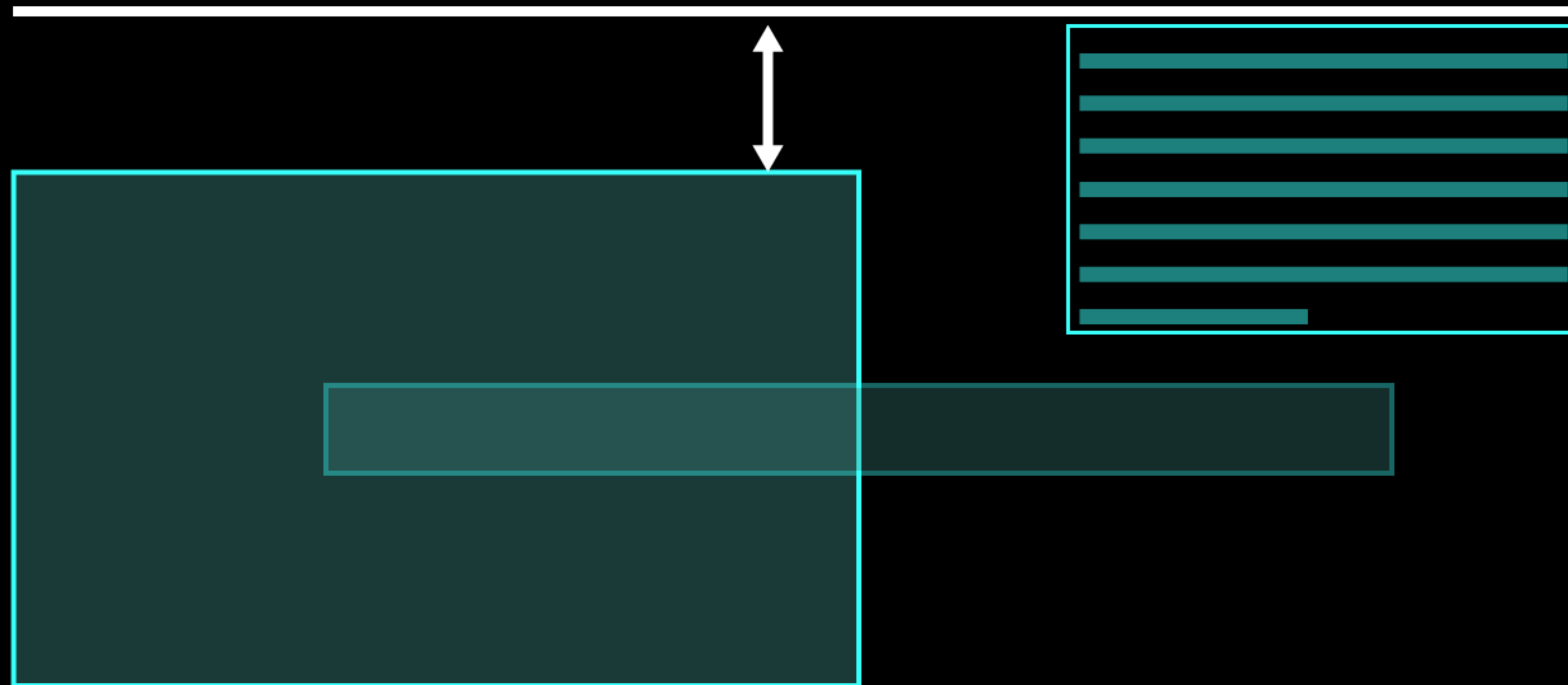


...no matter the size of the
image or the text

2. Text block must align to the top of the image

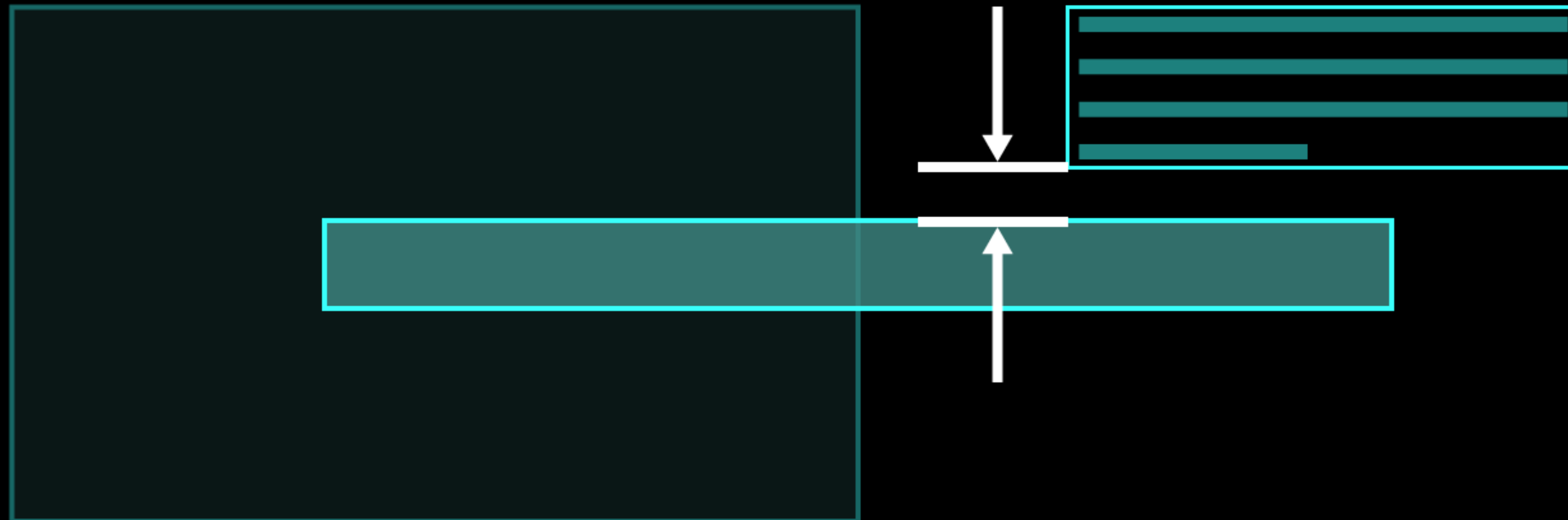


2. Text block must align to the top of the image

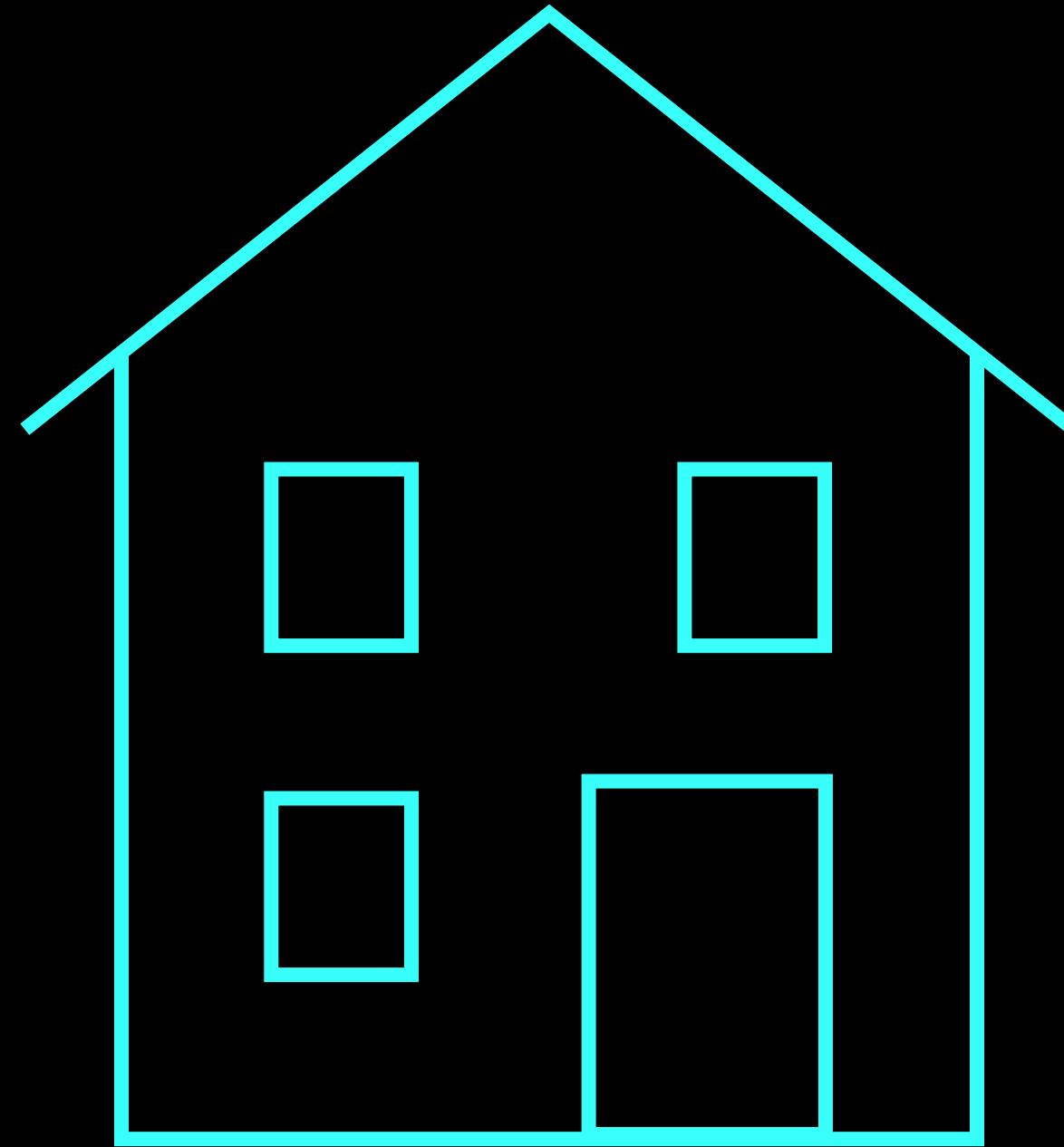


...unless the text is too long

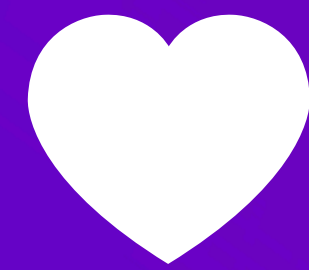
3. Space must always be maintained between the text block and heading



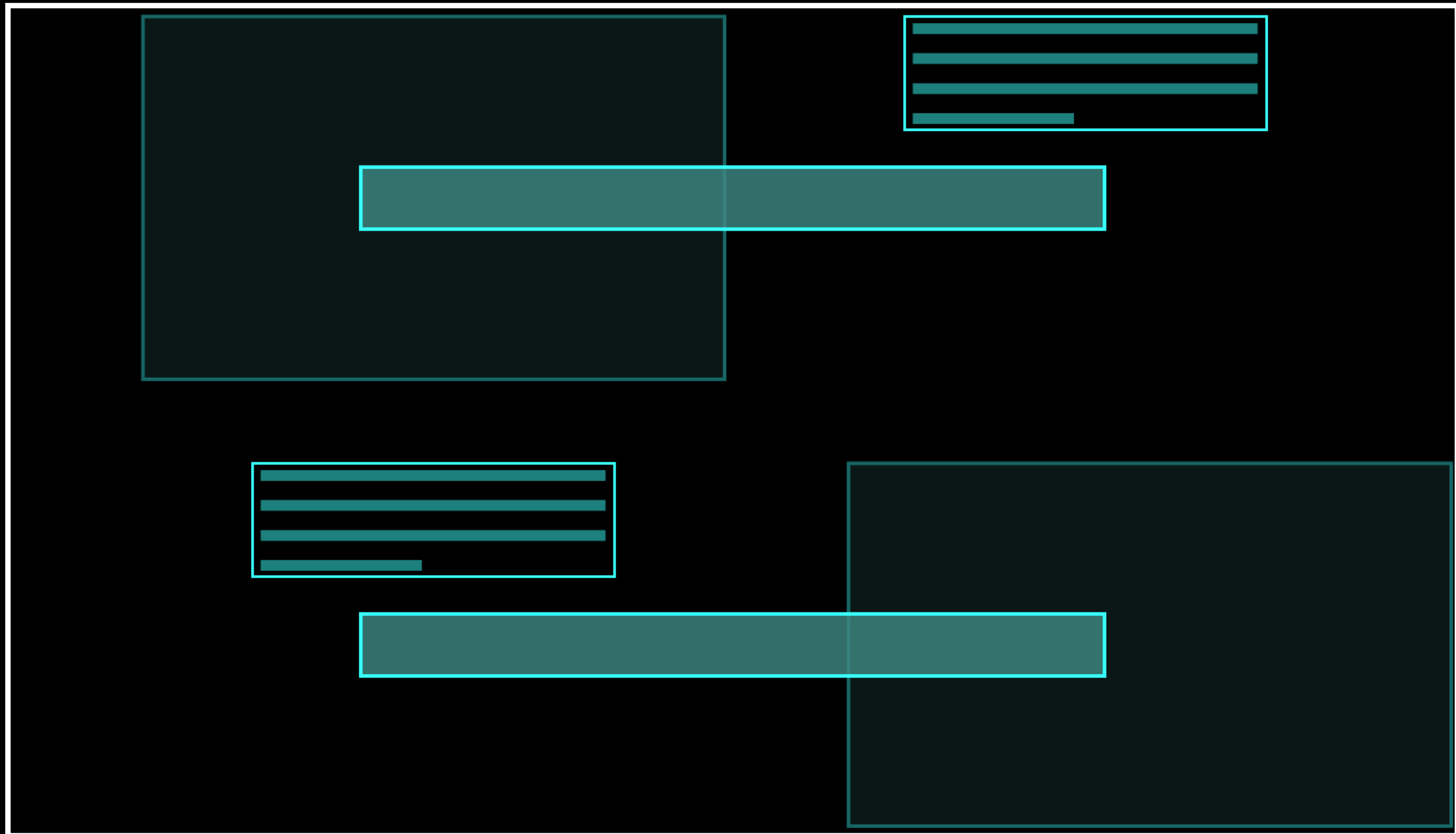
Context-aware

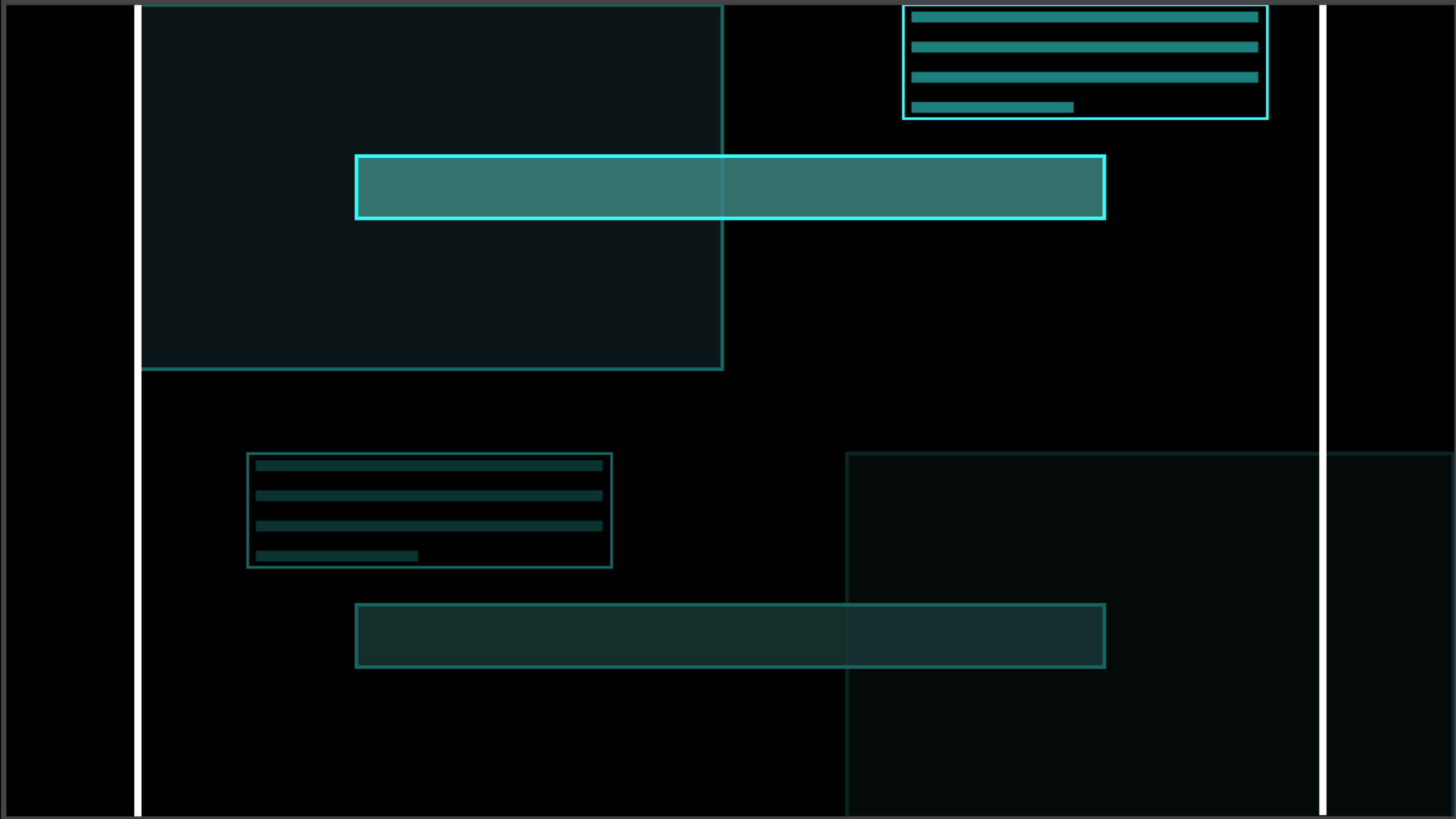


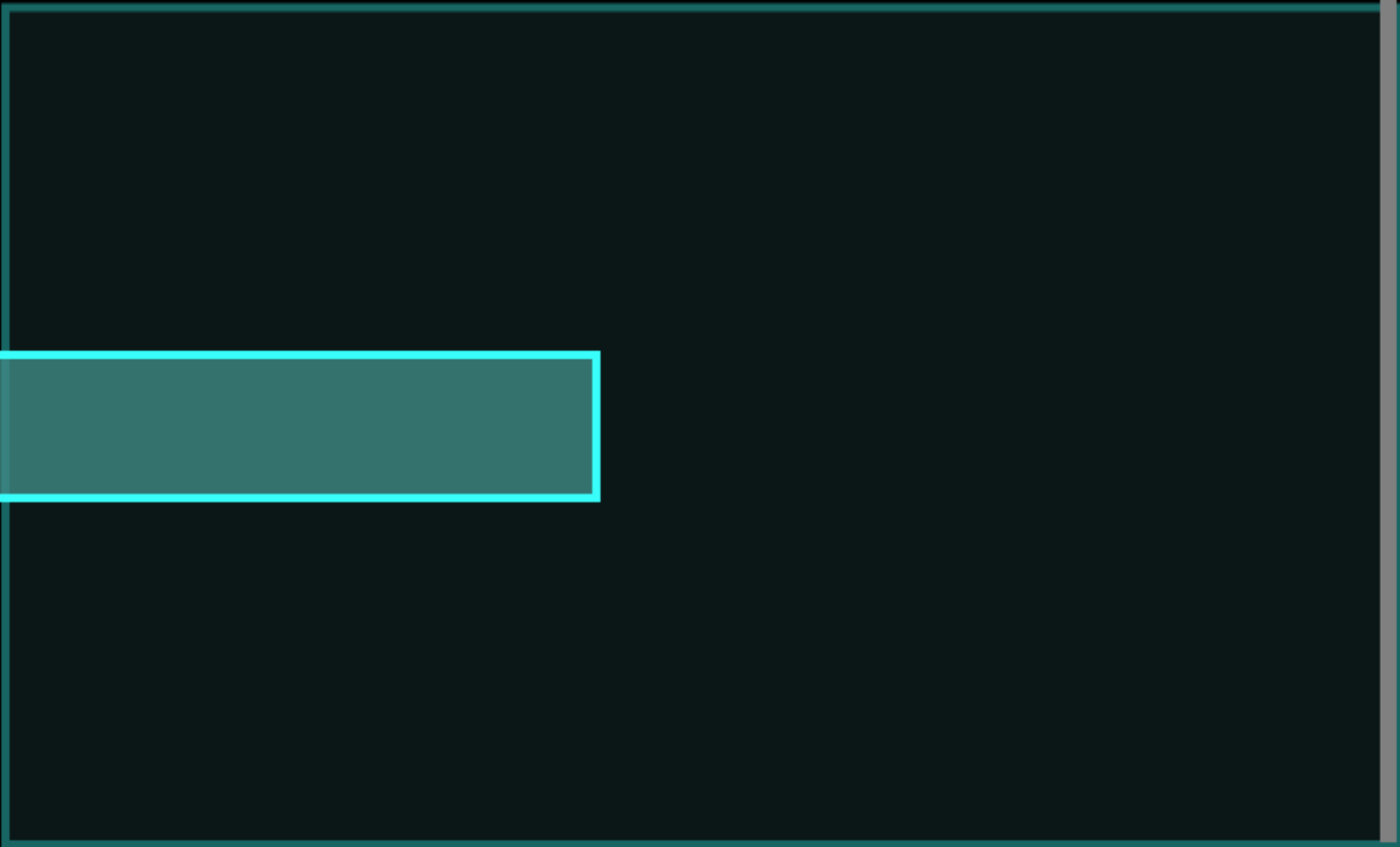
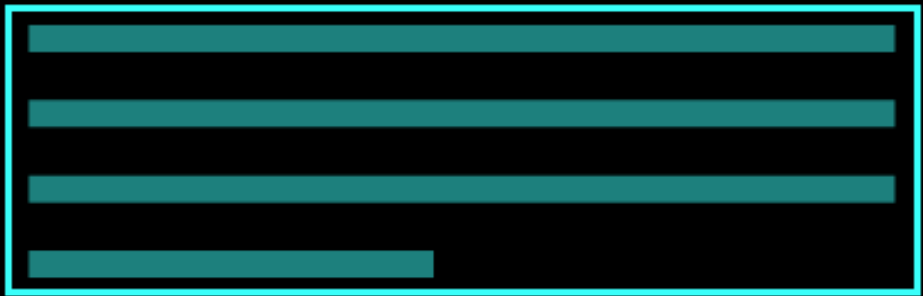
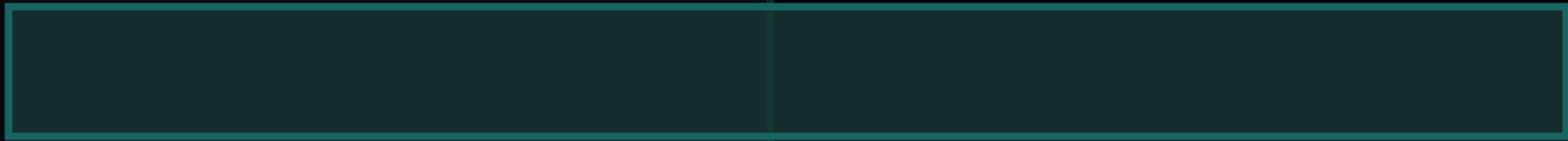
CSS Grid

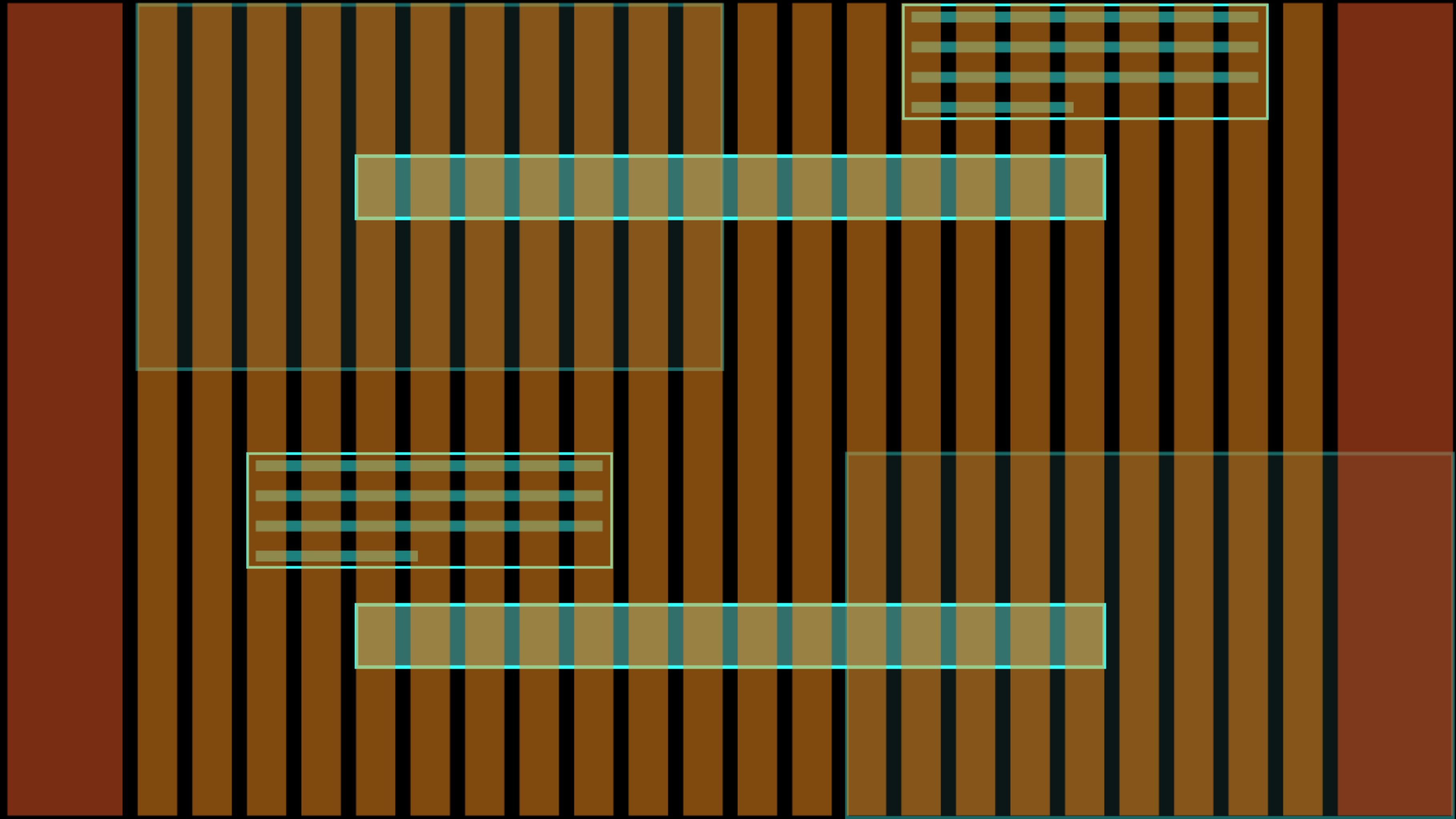


2D Layout

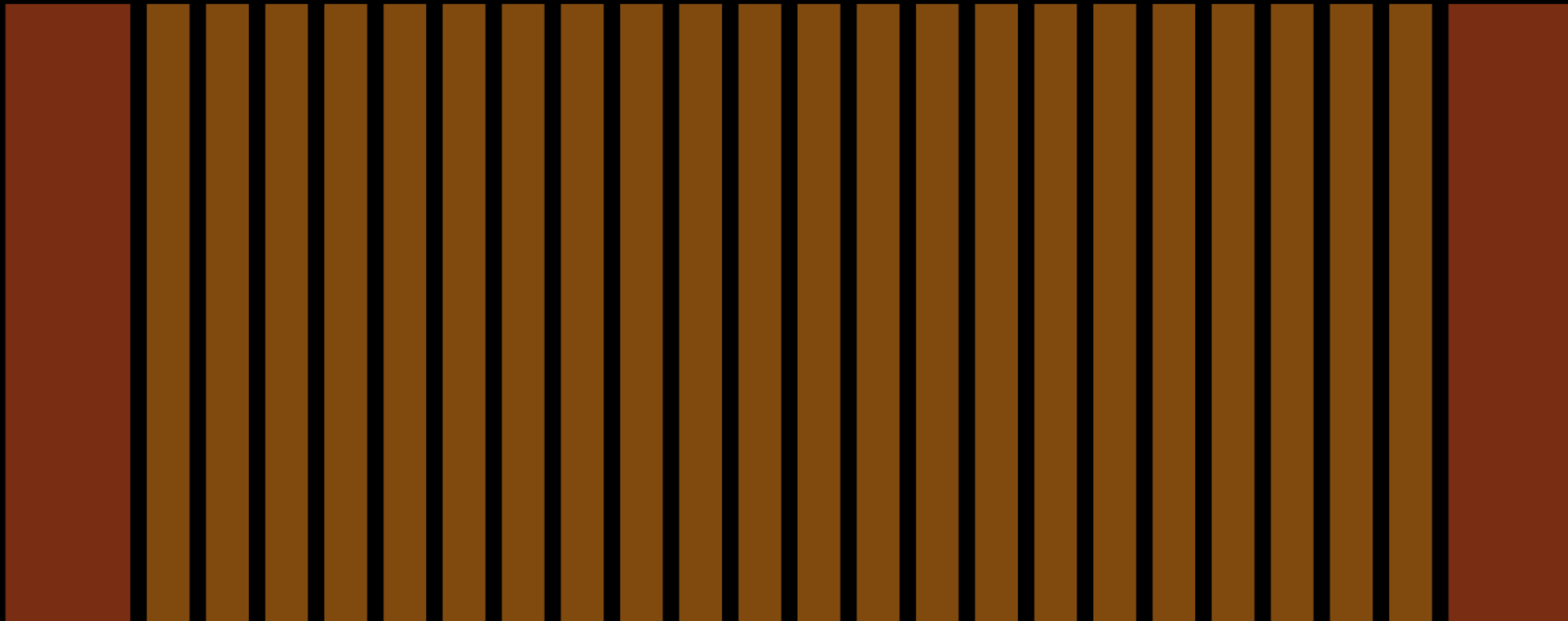




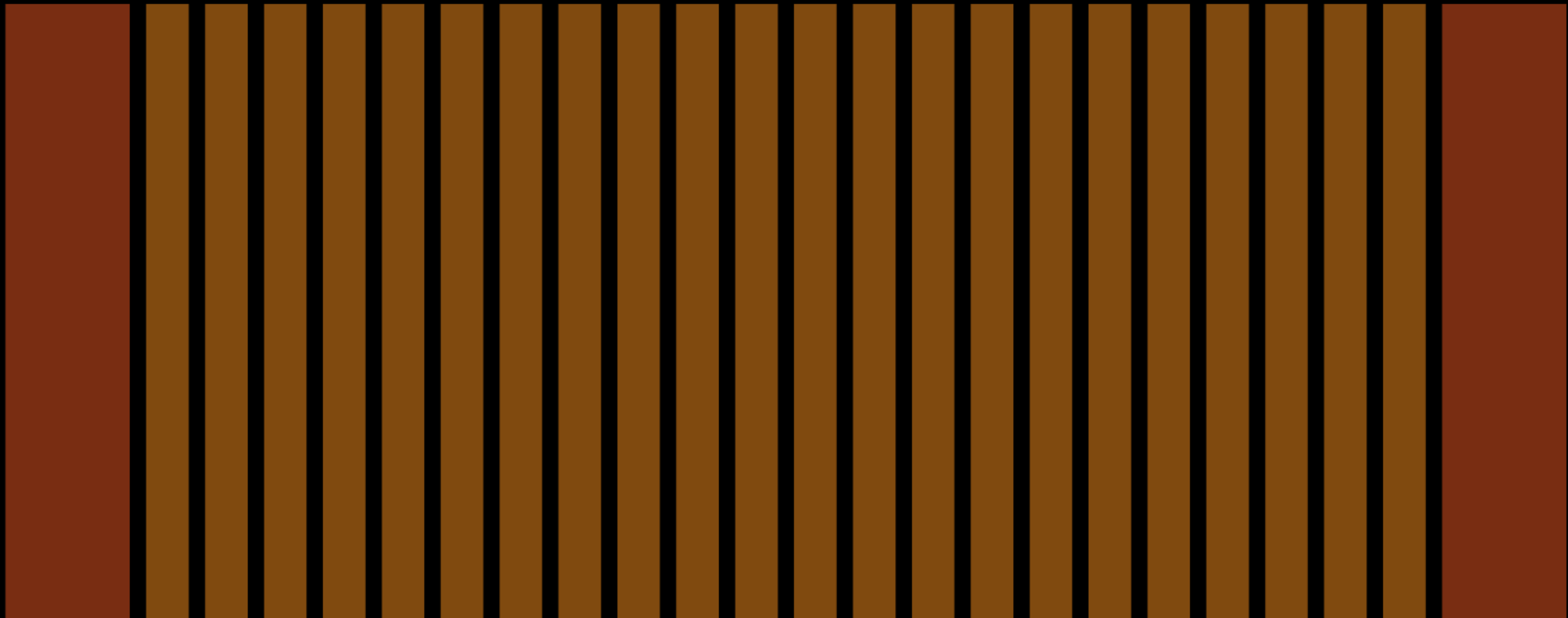


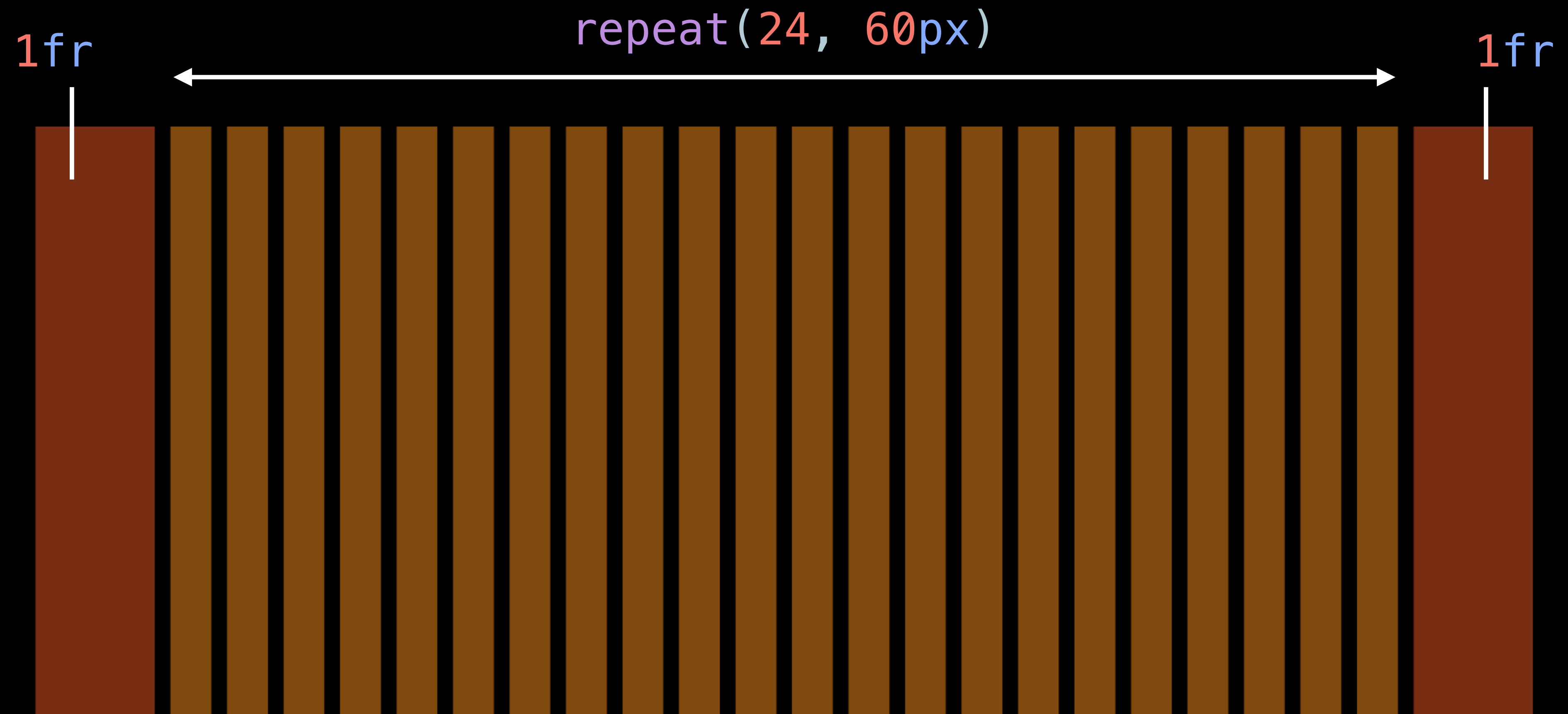



```
grid-template-columns: repeat(26, 1fr);
```

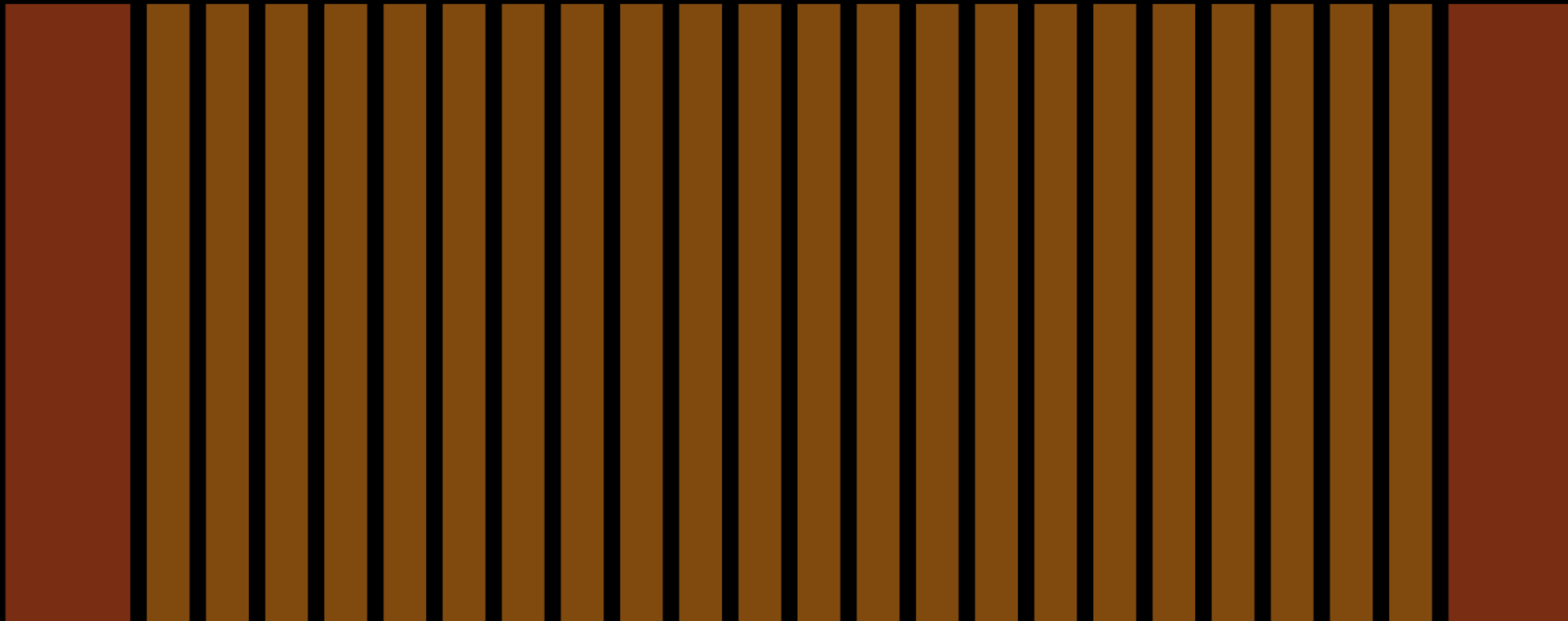


```
grid-template-columns: repeat(26, 1fr);
```





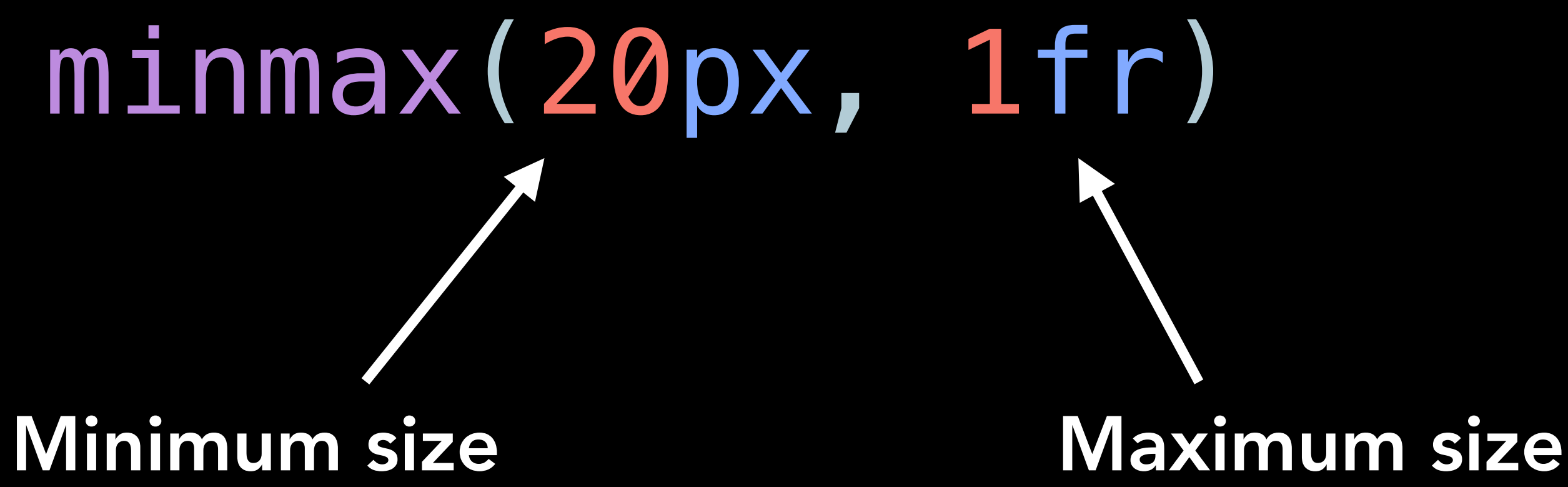

```
grid-template-columns: 1fr repeat(24, 60px) 1fr;
```



minmax()

minmax(20px, 1fr)

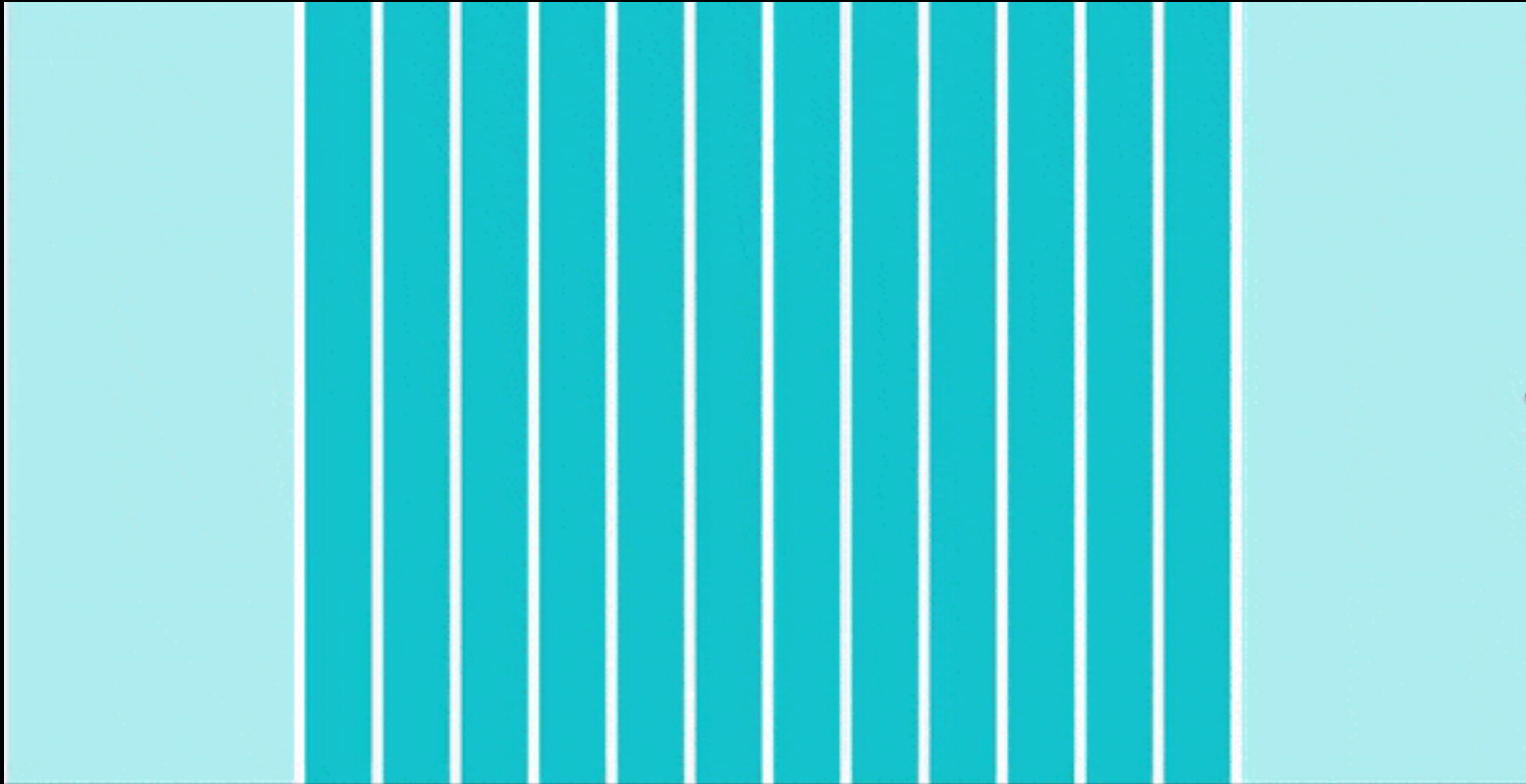
Minimum size



The diagram illustrates the parameters of the minmax() function. It shows the function call minmax(20px, 1fr) with the first parameter '20px' in red and the second parameter '1fr' in blue. Below the first parameter, the text 'Minimum size' is written, with a white arrow pointing from it to the '20px' value. Similarly, below the second parameter, the text 'Maximum size' is written, with a white arrow pointing from it to the '1fr' value.

Maximum size

`minmax(20px, 1fr)`

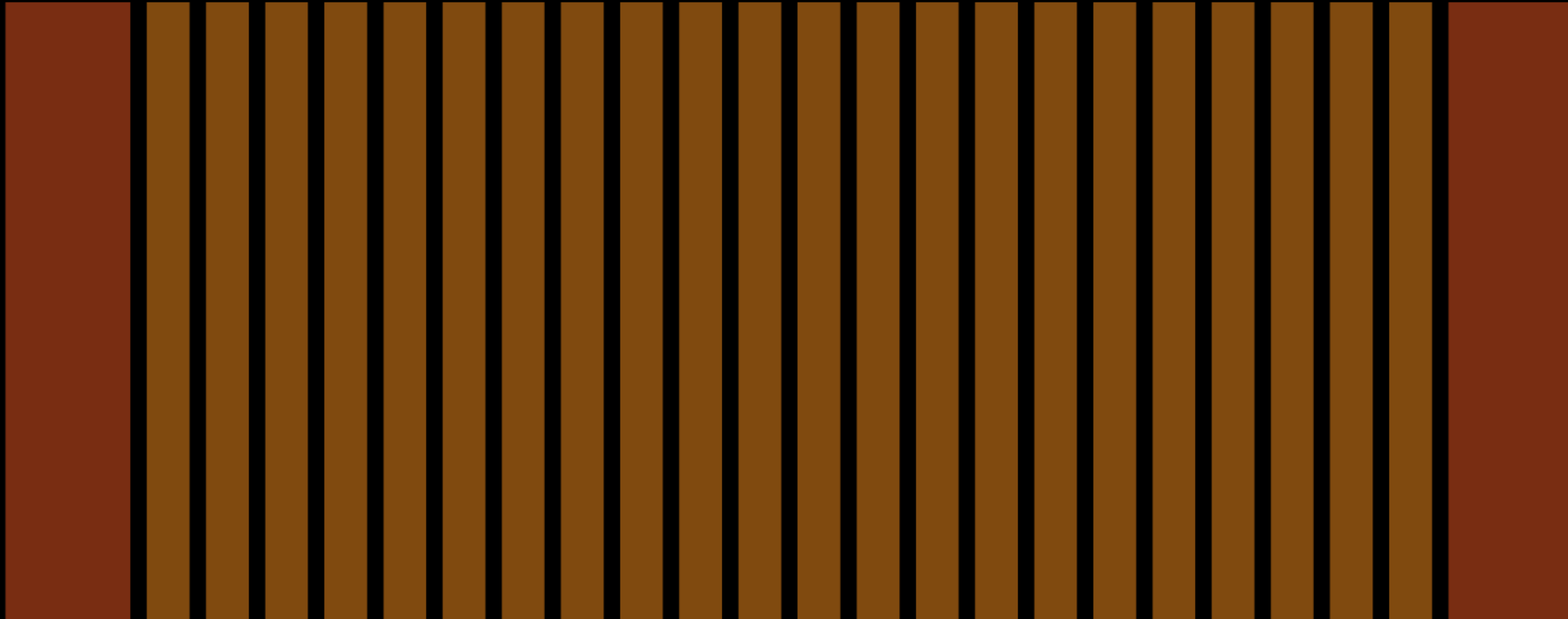


`minmax(0, 60px)`

codepen.io/michellebarker/pen/wYGMPQ

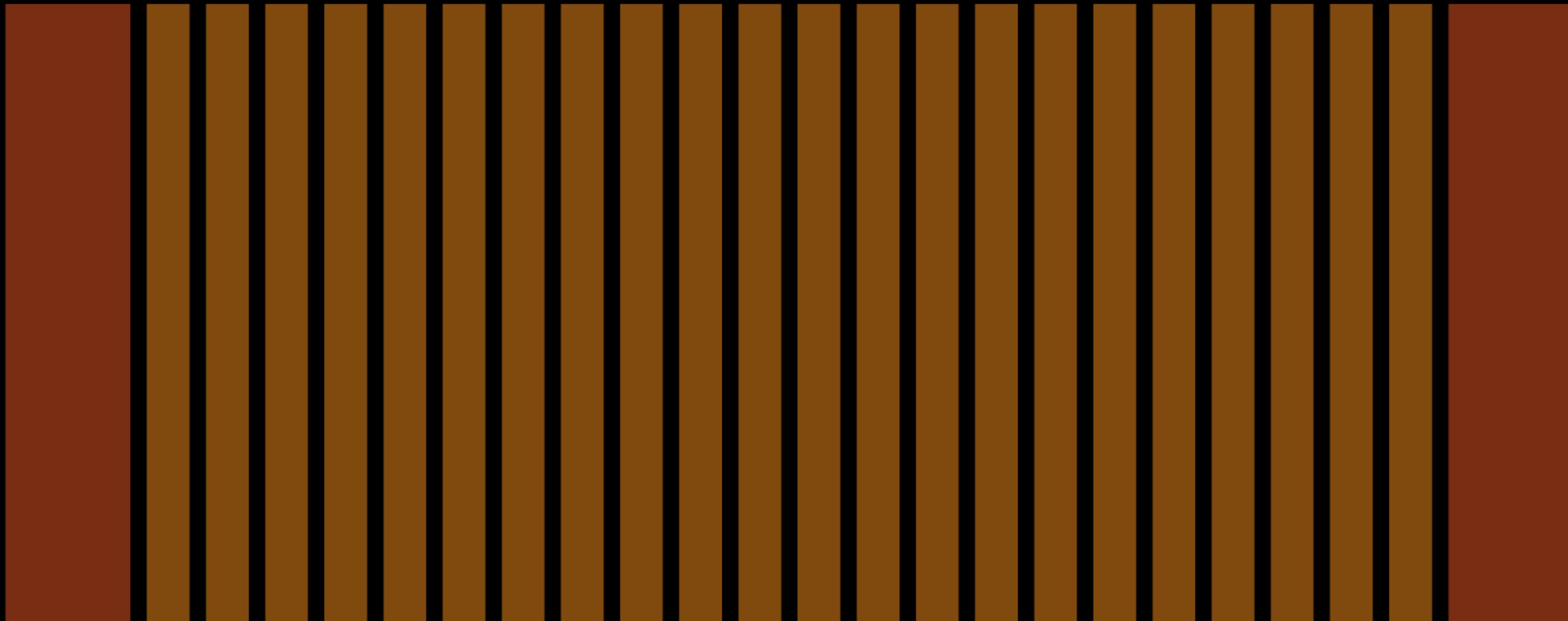
```
grid-template-columns:
```

```
minmax(20px, 1fr) repeat(24, minmax(0, 60px)) minmax(20px, 1fr);
```

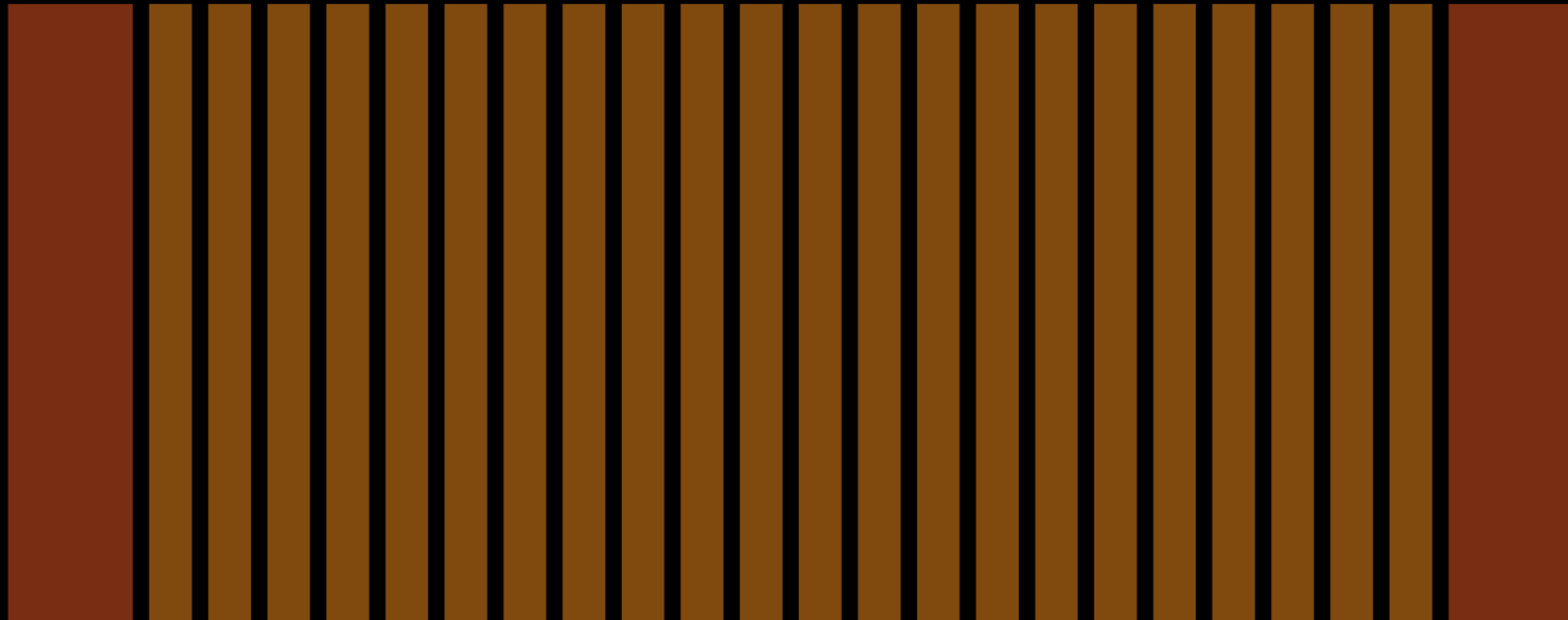


grid-template-columns:

minmax(20px, 1fr) repeat(24, minmax(0, 60px)) minmax(20px, 1fr);



```
grid-template-columns:
minmax(20px, 1fr) repeat(24, minmax(0, 60px)) minmax(20px, 1fr);
```

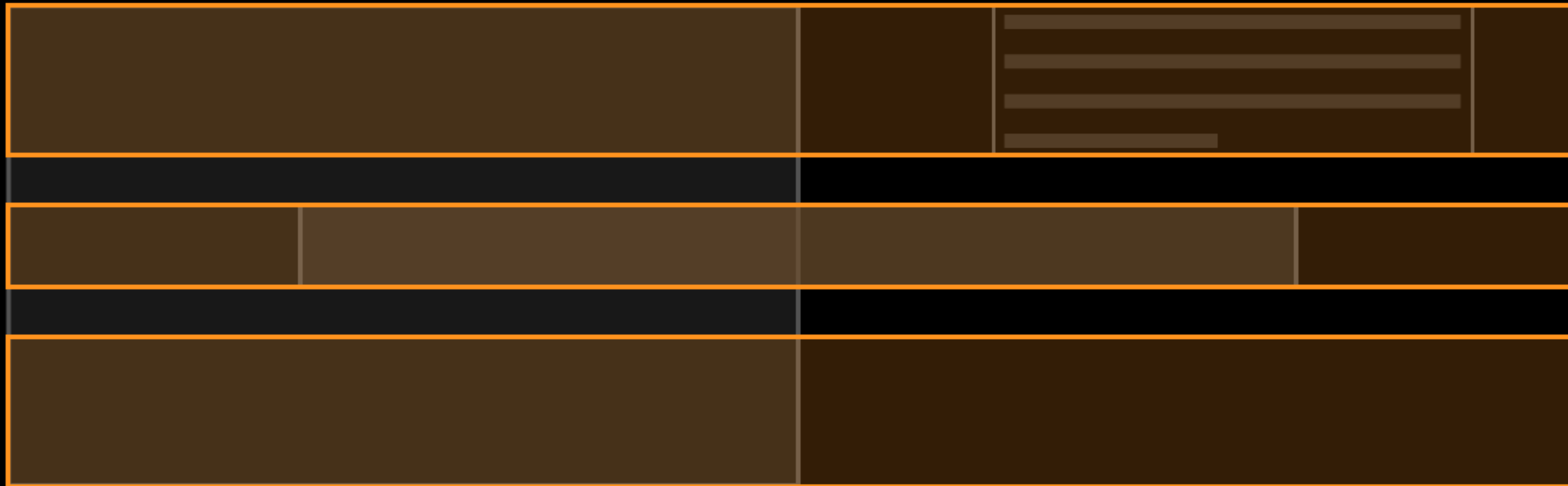


CSS

```
.grid {  
  display: grid;  
  grid-template-columns:  
    minmax(20px, 1fr) // flexible outer column  
    repeat(24, minmax(0, 60px)) // 24 central columns  
    minmax(20px, 1fr); // flexible outer column  
  column-gap: 20px;  
}
```



```
grid-template-rows: 1fr auto 1fr;
```



CSS

```
.grid {  
  display: grid;  
  grid-template-columns:  
    minmax(20px, 1fr)  
    repeat(24, minmax(0, 60px))  
    minmax(20px, 1fr);  
  grid-template-rows: 1fr auto 1fr;  
  column-gap: 20px;  
}
```


CSS

```
.grid {  
  display: grid;  
  grid-template-columns:  
    minmax(20px, 1fr)  
    repeat(24, minmax(0, 60px))  
    minmax(20px, 1fr);  
  grid-template-rows: 1fr auto 1fr;  
  gap: 40px 20px;  
}
```

CSS

```
.grid {  
  display: grid;  
  grid-template-columns:  
    minmax(20px, 1fr)  
    repeat(24, minmax(0, 60px))  
    minmax(20px, 1fr);  
  grid-template-rows: 1fr auto 1fr;  
  gap: 40px 20px;  
  align-items: center;  
}
```

CSS

Placing items

```
.grid {  
  grid-template-columns:  
    [outer-start]  
    minmax(20px, 1fr)  
    [wrapper-start]  
    repeat(24, minmax(0, 60px))  
    [wrapper-end]  
    minmax(20px, 1fr)  
    [outer-end];  
  grid-template-rows:  
    1fr [heading-start] auto [heading-end] 1fr;  
  ...  
}
```


CSS

Placing items

```
.grid {  
  grid-template-columns:  
    [outer-start]  
    minmax(20px, 1fr)  
    [wrapper-start]  
    repeat(24, minmax(0, 60px))  
    [wrapper-end]  
    minmax(20px, 1fr)  
    [outer-end];  
  grid-template-rows:  
    1fr [heading-start] auto [heading-end] 1fr;  
  ...  
}
```

CSS

Placing items

```
.grid {  
  grid-template-columns:  
    [outer-start]  
    minmax(20px, 1fr)  
    [wrapper-start]  
    repeat(24, minmax(0, 60px))  
    [wrapper-end]  
    minmax(20px, 1fr)  
    [outer-end];  
  grid-template-rows:  
    1fr [heading-start] auto [heading-end] 1fr;  
  ...  
}
```

CSS

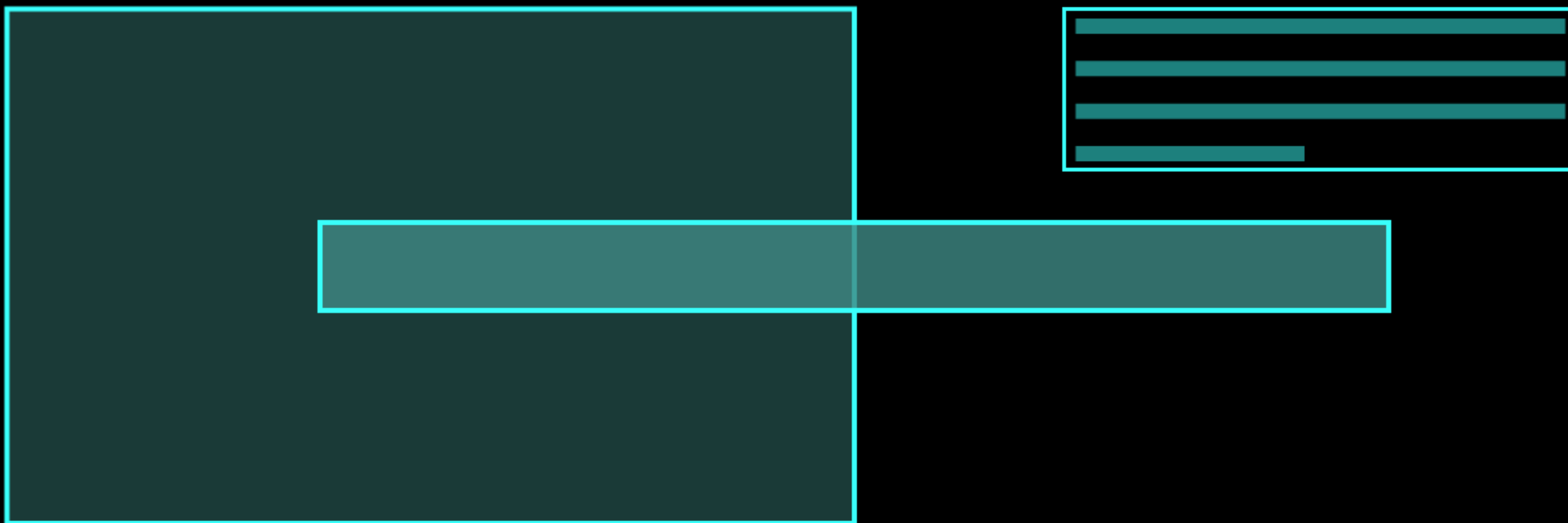
```
.grid__image {  
  grid-column: wrapper-start / 14;  
  grid-row: 1 / -1;  
}  
  
.grid__body {  
  grid-column: span 5 / wrapper-end;  
  grid-row: 1;  
  align-self: flex-start;  
}  
  
.grid__heading {  
  grid-column: 5 / -5;  
  grid-row: heading;  
}
```

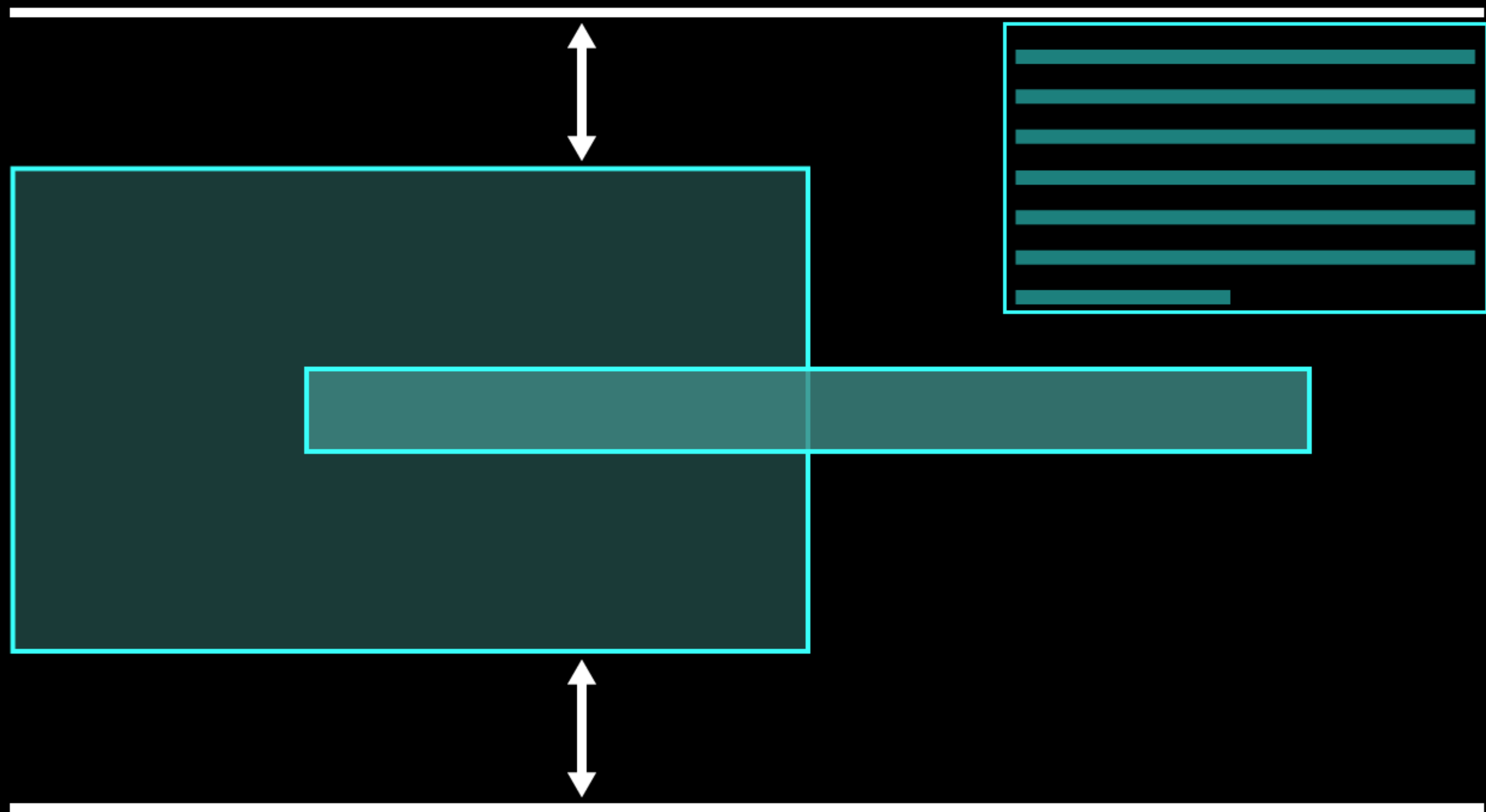

CSS

```
.grid__image {  
  grid-column: wrapper-start / 14;  
  grid-row: 1 / -1;  
}  
  
.grid__body {  
  grid-column: span 5 / wrapper-end;  
  grid-row: 1;  
  align-self: flex-start;  
}  
  
.grid__heading {  
  grid-column: 5 / -5;  
  grid-row: heading;  
}
```

CSS

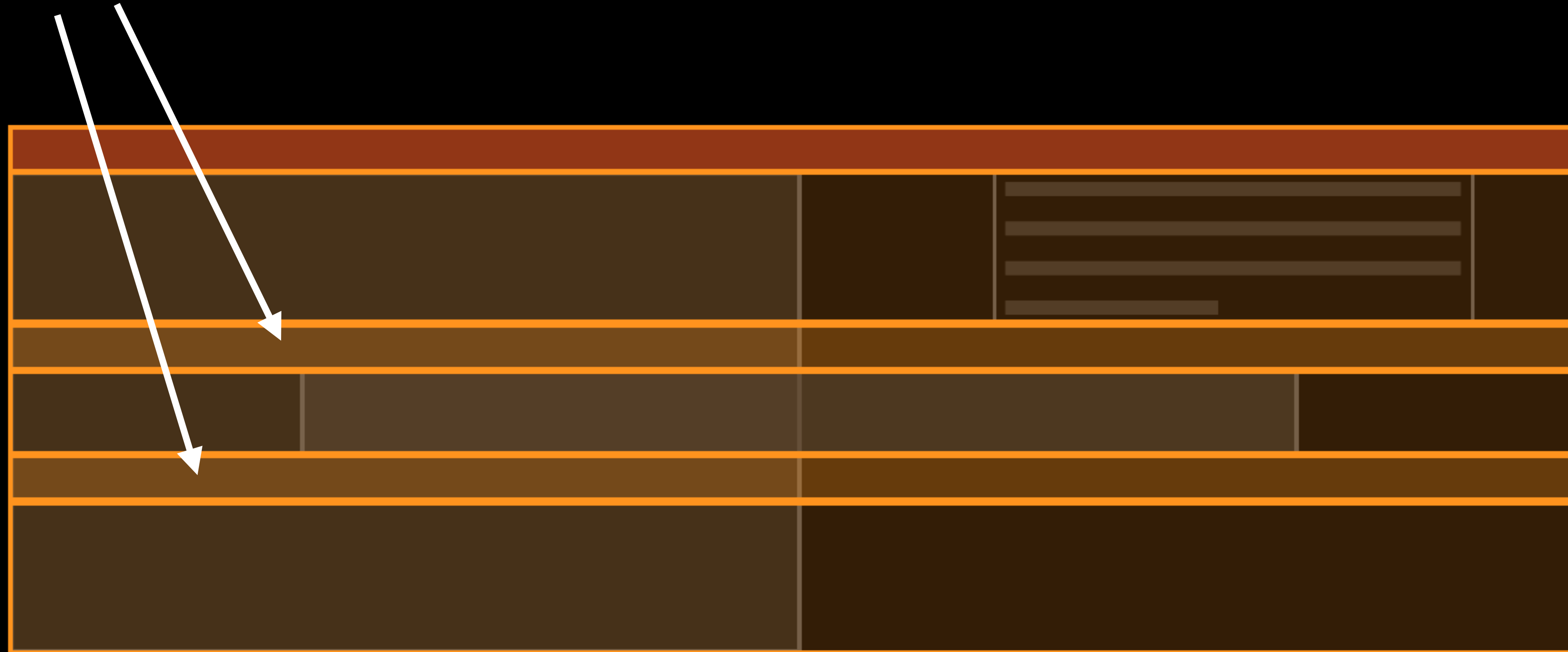
```
.grid__image {  
  grid-column: wrapper-start / 14;  
  grid-row: 1 / -1;  
}  
  
.grid__body {  
  grid-column: span 5 / wrapper-end;  
  grid-row: 1;  
  align-self: flex-start;  
}  
  
.grid__heading {  
  grid-column: 5 / -5;  
  grid-row: heading;  
}
```



I didn't solve this problem

Gutter rows

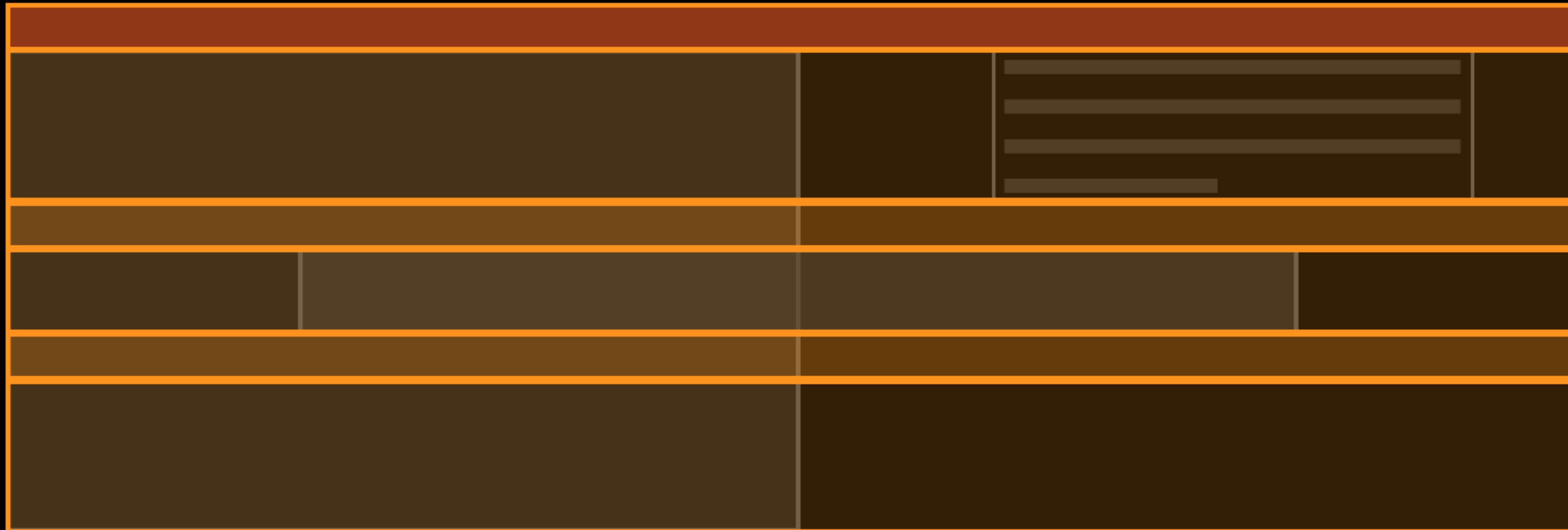


The diagram illustrates a table structure with a header row and several data rows. The header row is a solid dark orange. The data rows are divided into two main sections by a vertical line. The left section contains two rows with a light orange background, which are the 'gutter rows' indicated by white arrows. The right section contains two rows with a dark brown background. The first data row (row 2) is divided into three columns: a large left column, a medium right column, and a small right column containing four horizontal bars. The second data row (row 3) is divided into two columns: a large left column and a medium right column. The third data row (row 4) is divided into two columns: a large left column and a medium right column. The fourth data row (row 5) is divided into two columns: a large left column and a medium right column. The fifth data row (row 6) is divided into two columns: a large left column and a medium right column.

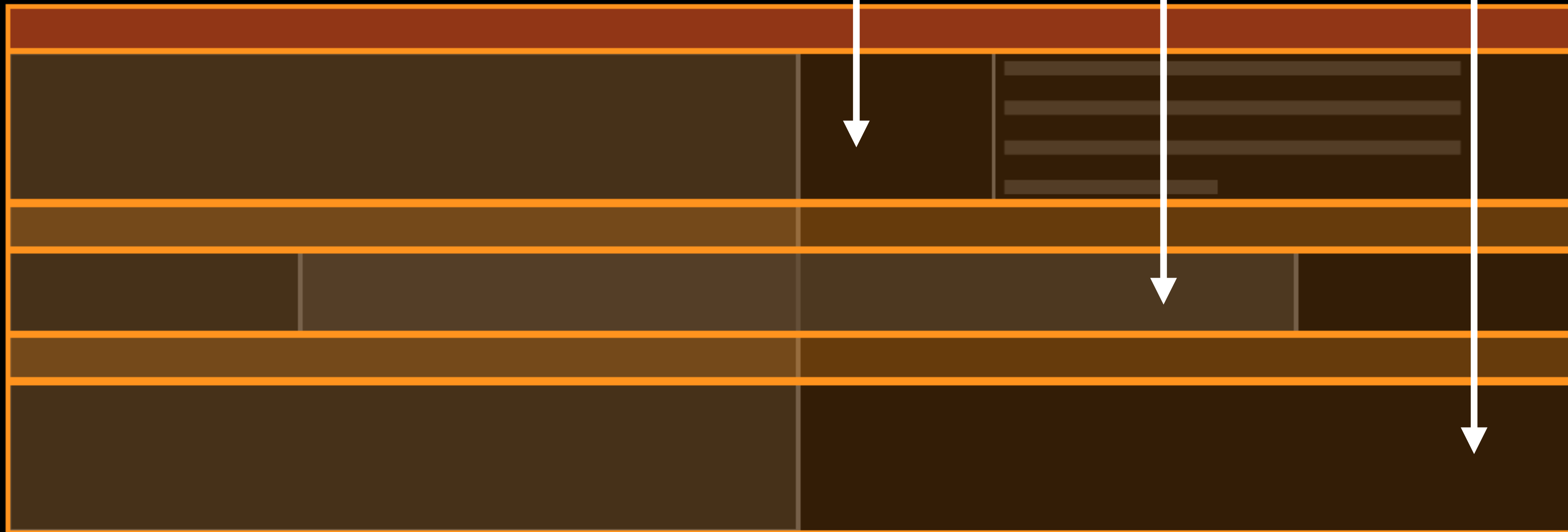


Project Overview			
Project Details	Project Name	Project ID	Project Manager
	Project Description	Project Start Date	Project End Date
	Project Budget	Project Status	Project Risk
	Project Location	Project Team	Project Sponsor
	Project Contact	Project Email	Project Phone
Project Summary		Project Notes	
Project Objectives		Project Deliverables	
Project Milestones		Project Risks	
Project Resources		Project Budget	
Project Schedule		Project Status	
Project Performance		Project Feedback	

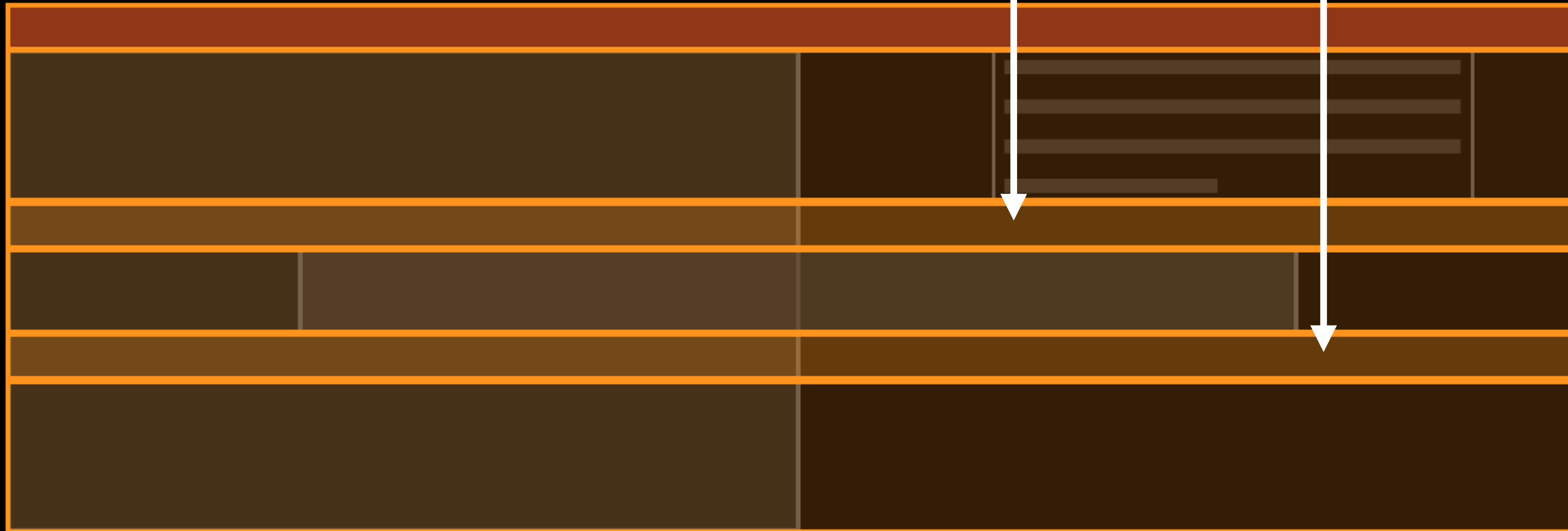
```
grid-template-rows: 1fr auto 40px auto 40px auto;
```




```
grid-template-rows: 1fr auto 40px auto 40px auto;
```

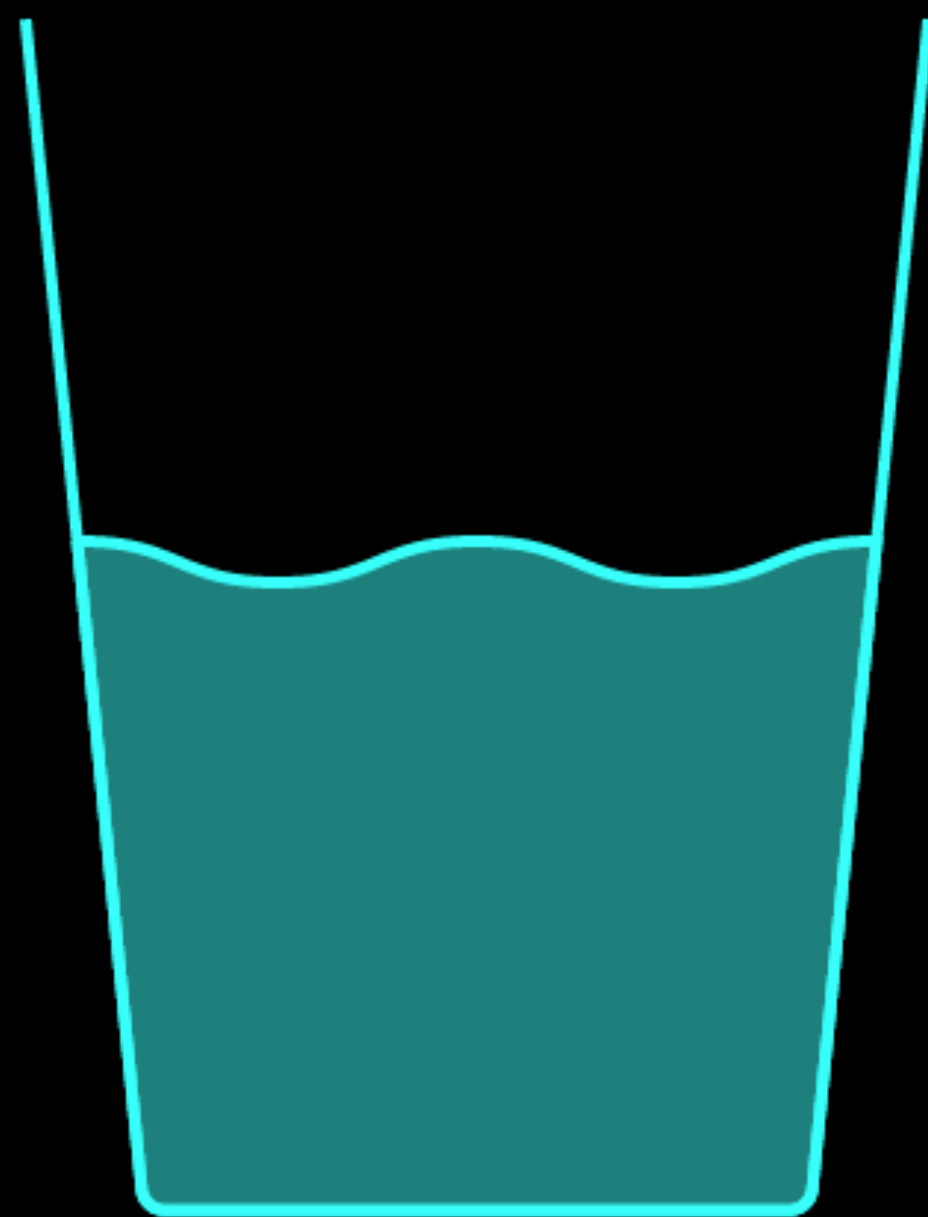


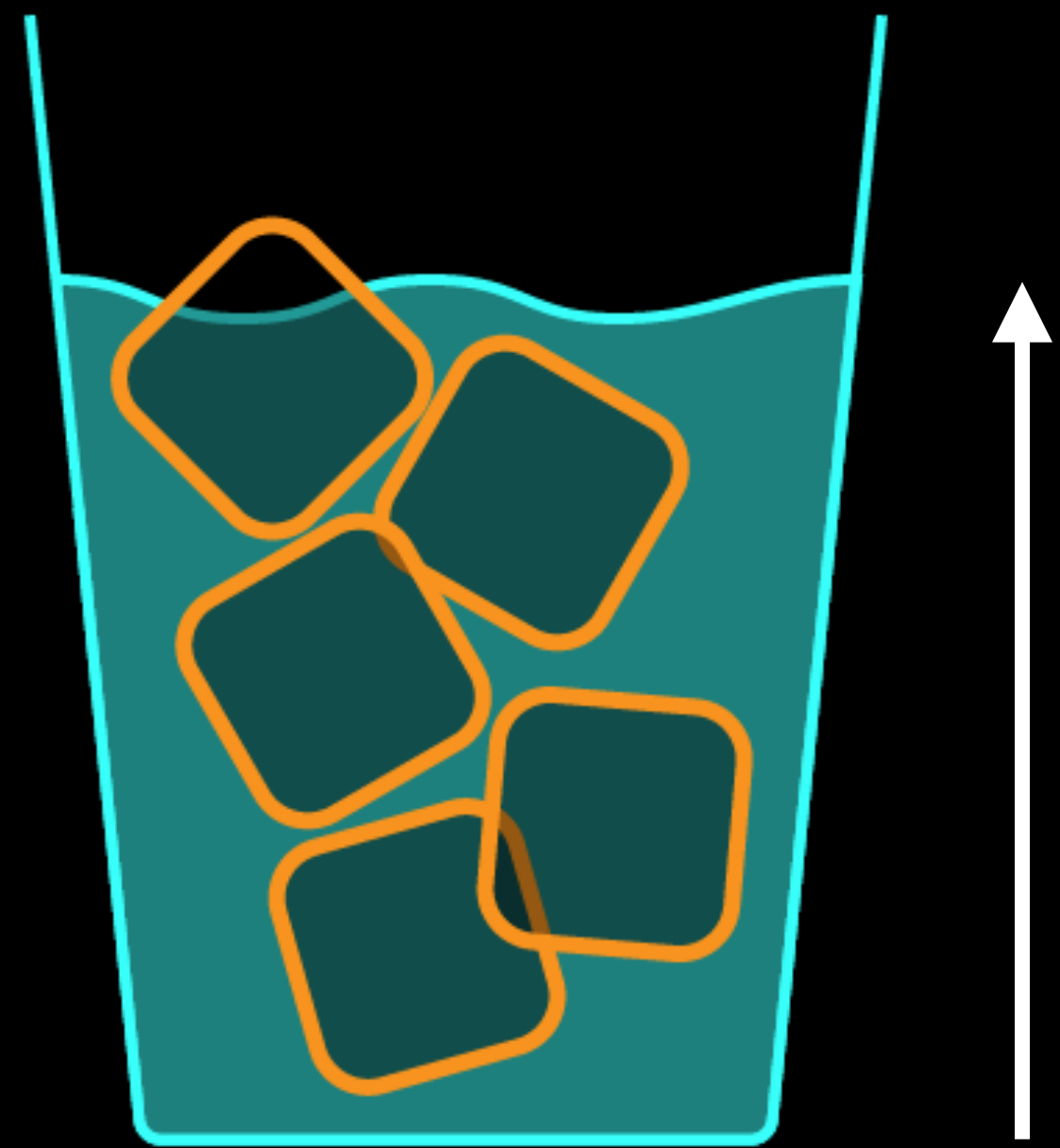
```
grid-template-rows: 1fr auto 40px auto 40px auto;
```




```
grid-template-rows: 1fr auto 40px auto 40px auto;
```



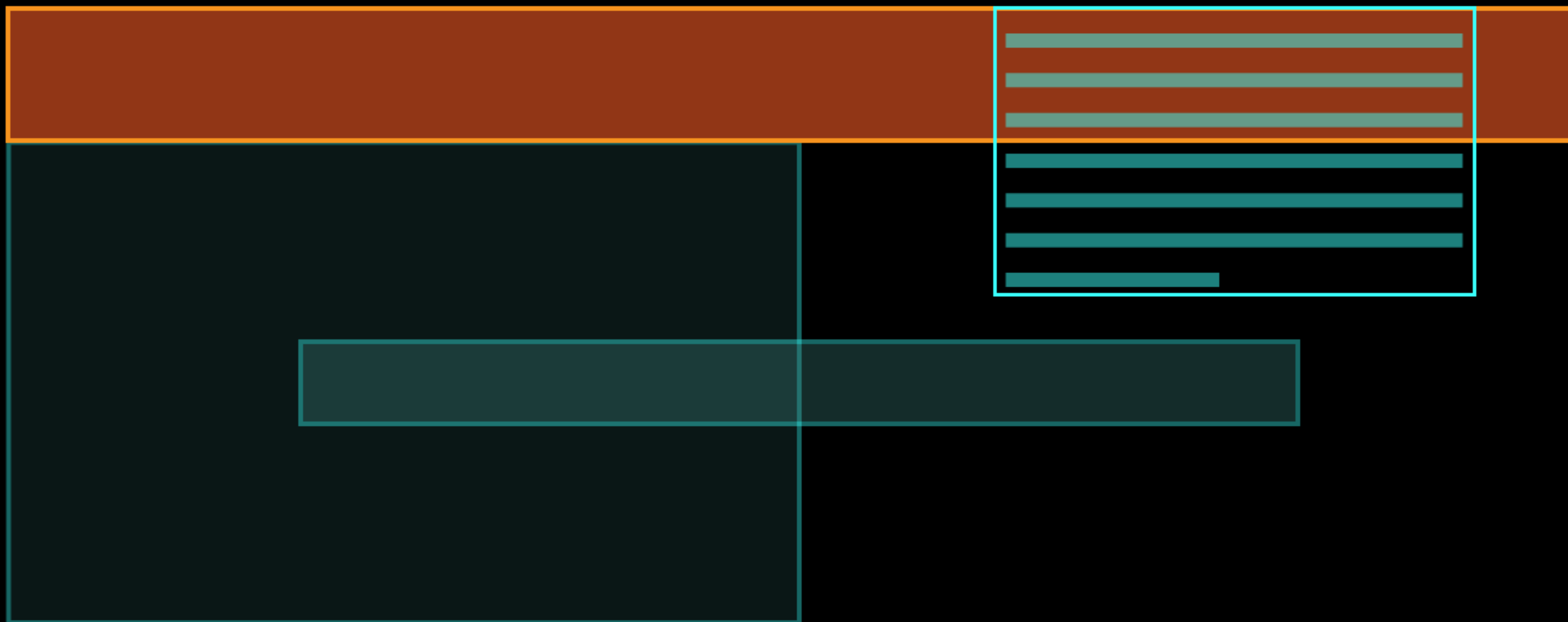


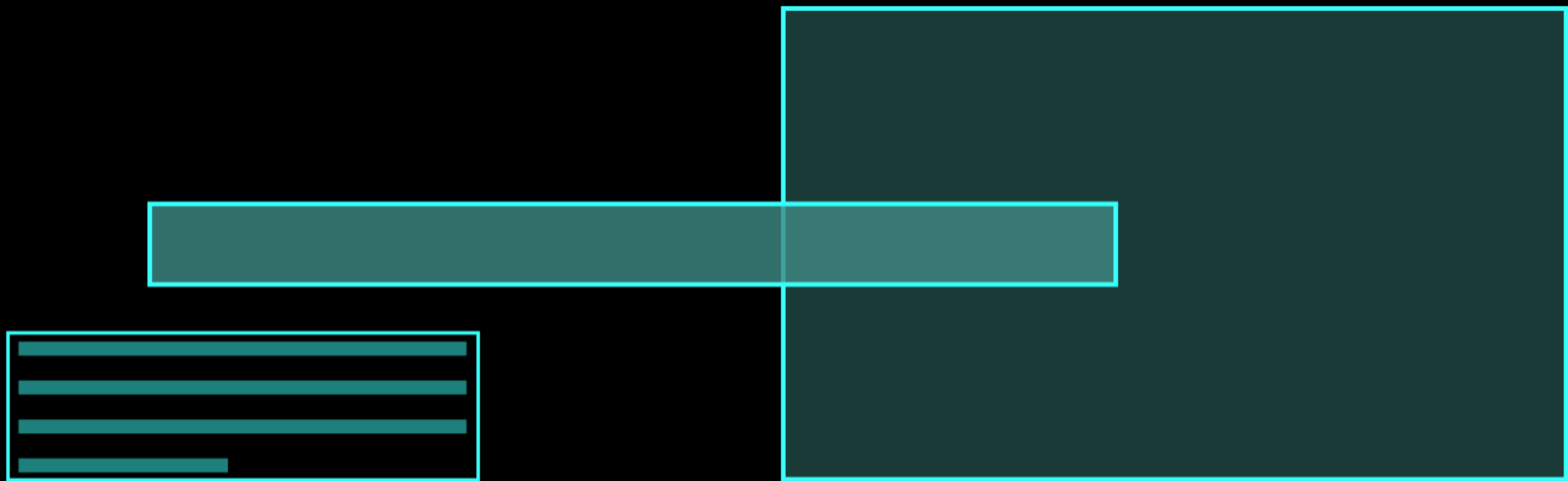


CSS

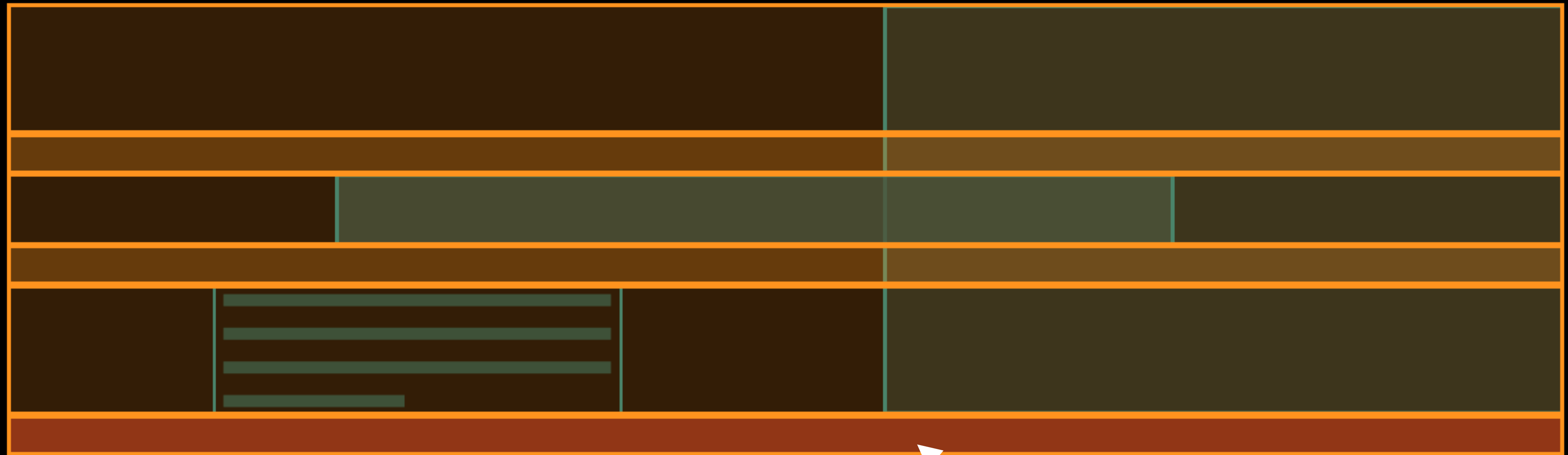
```
.grid {  
  ...  
  grid-template-rows:  
    [text-start] 1fr auto [text-end]  
    40px // gap  
    [heading-start] auto [heading-start]  
    40px // gap  
    auto;  
  ...  
}
```

```
.grid__body {  
  grid-row: text;  
}
```






```
grid-template-rows: auto 40px auto 40px auto 1fr;
```



Secret hidden row

CSS

```
.grid--text-bottom {  
  grid-template-rows:  
    auto 40px  
    [heading-start] auto [heading-end]  
    40px [text-start] auto  
    1fr [text-end]; // hidden row  
}
```


CSS

```
.grid--text-bottom .grid__body {  
  align-self: flex-end;  
}
```

Component variants

SCSS

```
/* Second variant of our component */  
.grid--2 {  
  .grid__image {  
    grid-column: 13 / outer-end;  
  }  
  
  .grid__body {  
    grid-column: wrapper-start / span 5;  
  }  
}
```


CSS Variables

(a.k.a. Custom Properties)

CSS
Variables

==

Dynamic
variables

CSS

Defining variables

```
:root {  
  --bgColor: red;  
}
```


CSS

Using variables

```
.my-component {  
  background-color: var(--bgColor);  
}
```

CSS

```
.grid__image {  
  grid-column:  
    var(--mediaColStart) / var(--mediaColEnd);  
  grid-row: media;  
}  
  
.grid__body {  
  grid-column: var(--textColStart) / span 5;  
  grid-row: text;  
}
```


CSS

```
.grid__image {
```

```
  grid-column:  
    var(--mediaColStart) / var(--mediaColEnd);
```

```
  grid-row: media;  
}
```

```
.grid__body {  
  grid-column: var(--textColStart) / span 5;  
  grid-row: text;  
}
```

CSS

```
.grid__image {  
  grid-column:  
    var(--mediaColStart) / var(--mediaColEnd);  
  grid-row: media;  
}
```

```
.grid__body {  
  grid-column: var(--textColStart) / span 5;  
  grid-row: text;  
}
```

CSS

```
.grid {  
  --mediaColStart: wrapper-start;  
  --mediaColEnd: 14;  
  --textColStart: -8;  
}  
  
.grid--2 {  
  --mediaColStart: 13;  
  --mediaColEnd: outer-end;  
  --textColStart: 3;  
}
```


codepen.io/michellebarker/pen/yGzWme

Resources

Grid by Example (Rachel Andrew)

gridbyexample.com

Layout Land (Jen Simmons)

layout.land

CSS Grid Playground (MDN)

mozilladevelopers.github.io/playground/css-grid

CSS { In Real Life }

css-irl.info

Thank you for listening

@mbarker_84 / @CSSInRealLife