styled-components

+

CSS Grid

に感じる無限の可能性

2018/09/20 meguro.css #3

自己紹介

- S. Suzuki
- @terrierscript
- フリーランスエンジニア

今回話すこと

- styled-componentsの説明
- CSS Gridの説明
- 本題：ふたつを組み合わせる話

styled-components

styled components

Documentation • Ecosystem • Releases

Visual primitives for the component age.

Use the best bits of ES6 and CSS to style your apps without stress 💅

styled-components

- React用のライブラリ
- いわゆるCSS-in-JSの１つ
- CSSをtemplate literalで記述する

例

```
const Item = styled.div`
  color: #182438;
  padding: 1em;
  background: ${(props) => `${props.bgColor}`};
`;
```

```
export const Usage = () => {
  return <Item bgColor="#497acc">Hello styled</Item>;
};
```

div要素

```
const Item = styled.div`
  color: #182438;
  padding: 1em;
  background: ${(props) => `${props.bgColor}`};
`;
```

```
export const Usage = () => {
  return <Item bgColor="#497acc">Hello styled</Item>;
};
```

template literal
でCSSを指定

```
const Item = styled.div`
  color: #182438;
  padding: 1em;
  background: ${(props) => `${props.bgColor}`};
`;
```

```
export const Usage = () => {
  return <Item bgColor="#497acc">Hello styled</Item>;
};
```

```
const Item = styled.div`
  color: #182438;
  padding: 1em;
  background: ${(props) => `${props.bgColor}`};
`;
```

Itemという
Componentになる

```
export const Usage = () => {
  return <Item bgColor="#497acc">Hello styled</Item>;
};
```

```
const Item = styled.div`
  color: #182438;
  padding: 1em;
  background: ${(props) => `${props.bgColor}`};
`;
```

propsも使える

```
export const Usage = () => {
  return <Item bgColor="#497acc">Hello styled</Item>;
};
```

```
const Item = styled.div`
  color: #182438;
  padding: 1em;
  background: ${(props) => `${props.bgColor}`};
`;
```

props を渡す

```
export const Usage = () => {
  return <Item bgColor="#497acc">Hello styled</Item>;
};
```

```
const Item = styled.div`
  color: #182438;
  padding: 1em;
  background: ${(props) => `${props.bgColor}`};
`;
```

```
export const Usage = () => {
  return <Item bgColor="#497acc">Hello styled</Item>;
};
```

出力される実態はこんなイメージのもの

```
<style>
  .btNdYJ {
    color: #182438;
    padding: 1em;
    background: #497acc;
  }
</style>


<div class="btNdYJ">
  Hello styled
</div>
```

出力される実態はこんなイメージのもの

```
<style>
  .btNdYJ {
    color: #182430;
    padding: 1em;
    background: #497acc;
  }
</style>


<div class="btNdYJ">
  Hello styled
</div>
```

css-modulesとか
と理屈は一緒

拡張も出来ます

定義済みの
Componentを再利用

```
const ItemWithBorder = styled(Item)`
  color: #182438;
  background: ${(props) => `${props.bgColor}`};
  border: ${(props) => `10px solid ${props.borderColor}`};
`;
```

拡張も出来ます

```
const ItemWithBorder = styled(Item)`
  color: #182438;
  background: ${(props) => `${props.bgColor}`};
  border: ${(props) => `10px solid ${props.borderColor}`};
`;
```

borderが追加される

今回
styled-componentsは
このぐらいまで知ってもらえば十分

CSS Grid

MDNを見てみると・・・・

CSS グリッドレイアウトは、ページを大きな領域に分割することや、 HTML のプリミティブから構成されたコントロールの部品間の、寸法、位置、レイヤーに関する関係を定義することに優れています。

表と同様に、グリッドレイアウトによって要素を列と行に整列させることができます。しかし、 CSS グリッドを使用すると、表で実現するよりもより複雑なレイアウトが可能で、あるいは簡単に実現できます。例えば、グリッドのコンテナー内にある子要素は、 CSS の位置指定された要素と同様に自分自身の位置を決めることができるので、実際に重ね合わせてレイヤーになるように配置することができます。

引用元: https://developer.mozilla.org/ja/docs/Web/CSS/CSS_Grid_Layout

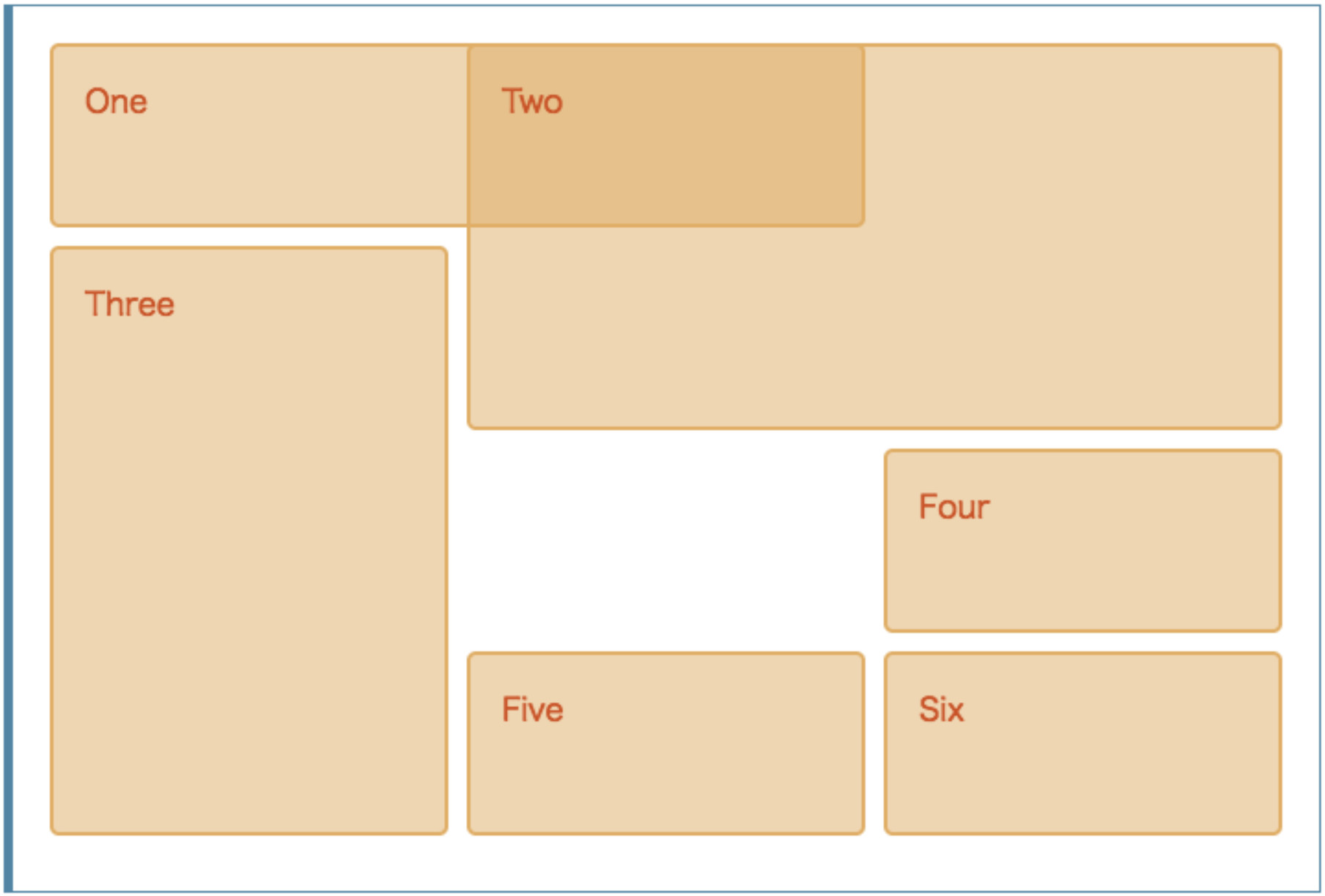
HTML

```
<div class="wrapper">
  <div class="one">One</div>
  <div class="two">Two</div>
  <div class="three">Three</div>
  <div class="four">Four</div>
  <div class="five">Five</div>
  <div class="six">Six</div>
</div>
```

引用元: https://developer.mozilla.org/ja/docs/Web/CSS/CSS_Grid_Layout

CSS

```css
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 10px;
  grid-auto-rows: minmax(100px, auto);
}
.one {
  grid-column: 1 / 3;
  grid-row: 1;
}
.two {
  grid-column: 2 / 4;
  grid-row: 1 / 3;
}
.three {
  grid-column: 1;
  grid-row: 2 / 5;
}
.four {
  grid-column: 3;
```


One
Two
Three
Four
Five
Six

難しい

一個ずつ見てみる

CSS

```css
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 10px;
  grid-auto-rows: minmax(100px, auto);
}
.one {
  grid-column: 1 / 3;
  grid-row: 1;
}
.two {
  grid-column: 2 / 4;
  grid-row: 1 / 3;
}
.three {
  grid-column: 1;
  grid-row: 2 / 5;
}
.four {
  grid-column: 3;
```

一個ずつ見てみる

CSS

```css
.wrapper {
  display: grid;
  grid-template-columns:
  grid-gap: 10px;
  grid-auto-rows: minmax(100px, auto);
}
.one {
  grid-column: 1 / 3;
  grid-row: 1;
}
.two {
  grid-column: 2 / 4;
  grid-row: 1 / 3;
}
.three {
  grid-column: 1;
  grid-row: 2 / 5;
}
.four {
  grid-column: 3;
```

display: grid

一個ずつ見てみる

CSS

```css
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 10px;
  grid-auto-rows: minma
}
.one {
  grid-column: 1
  grid-row: 1;
}
.two {
  grid-column: 2 / 4;
  grid-row: 1 / 3;
}
.three {
  grid-column: 1;
  grid-row: 2 / 5;
}
.four {
  grid-column: 3;
```

grid-template-columns
= 横方向へのテンプレート指定

一個ずつ見てみる

CSS

```css
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 10px;
  grid-auto-rows: minmax(100px, auto);
}
.one {
  grid-column: 1 / 3;
  grid-row: 1;
}
.two {
  grid-column: 2 / 4;
  grid-row: 1 / 3;
}
.three {
  grid-column: 1;
  grid-row: 2 / 5;
}
.four {
  grid-column: 3;
```

repeat(3, 1fr)
= 1frの単位を3回繰り返す

一個ずつ見てみる

CSS

```css
.wrapper {
  display: grid;
  grid-template-columns: repeat(3
  grid-gap: 10px;
  grid-auto-rows: minmax(
}
.one {
  grid-column: 1 / 3;
  grid-row: 1;
}
.two {
  grid-column: 2 / 4;
  grid-row: 1 / 3;
}
.three {
  grid-column: 1;
  grid-row: 2 / 5;
}
.four {
  grid-column: 3;
```

grid-gap
= paddingみたいなもの

一個ずつ見てみる

CSS

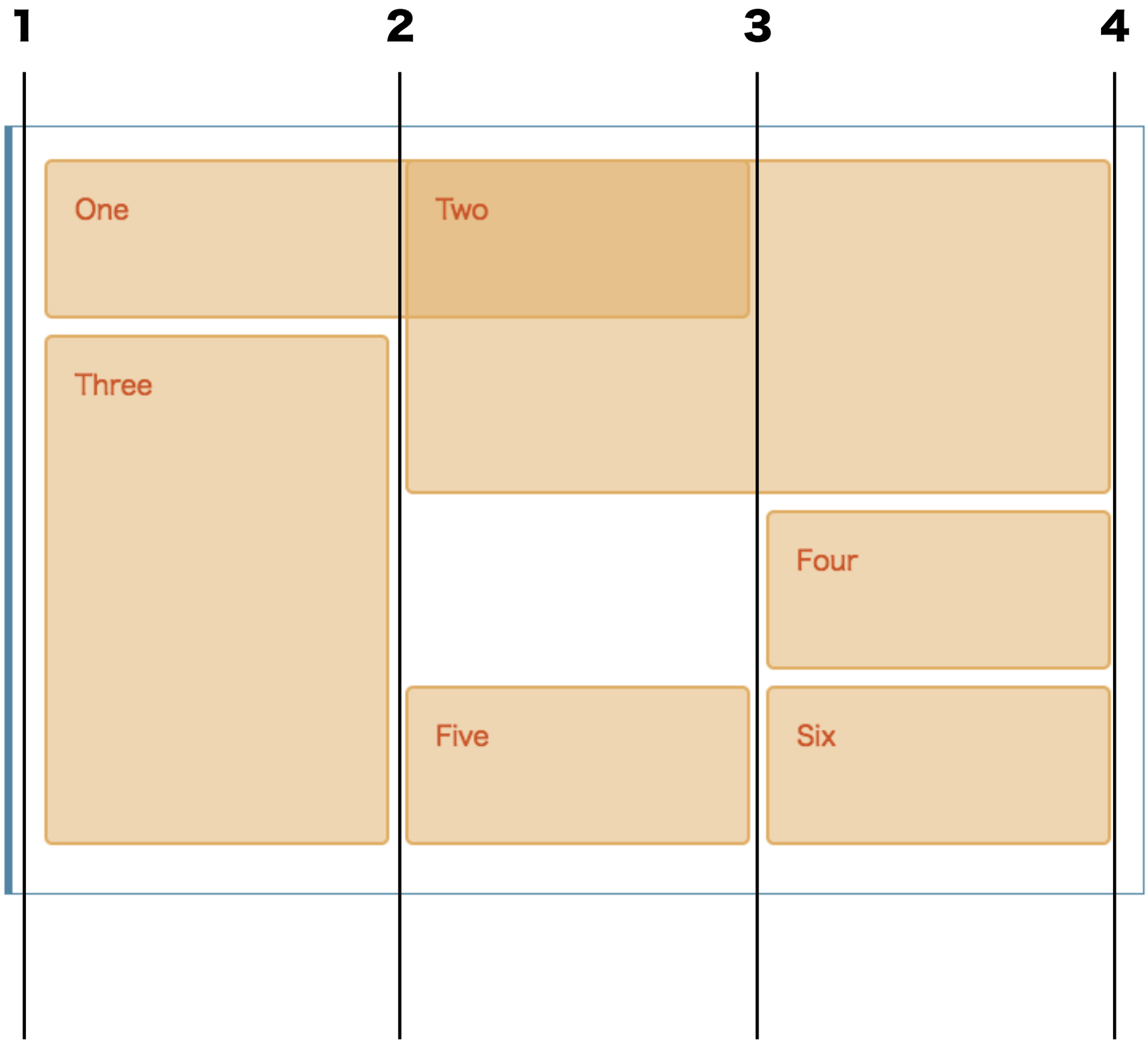
```css
.wrapper {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-gap: 10px;
  grid-auto-rows: minmax(100px, auto);
}
.one {
  grid-column: 1 / 3;
  grid-row: 1;
}
.two {
  grid-column: 2 / 4;
  grid-row: 1 / 3;
}
.three {
  grid-column: 1;
  grid-row: 2 / 5;
}
.four {
  grid-column: 3;
```

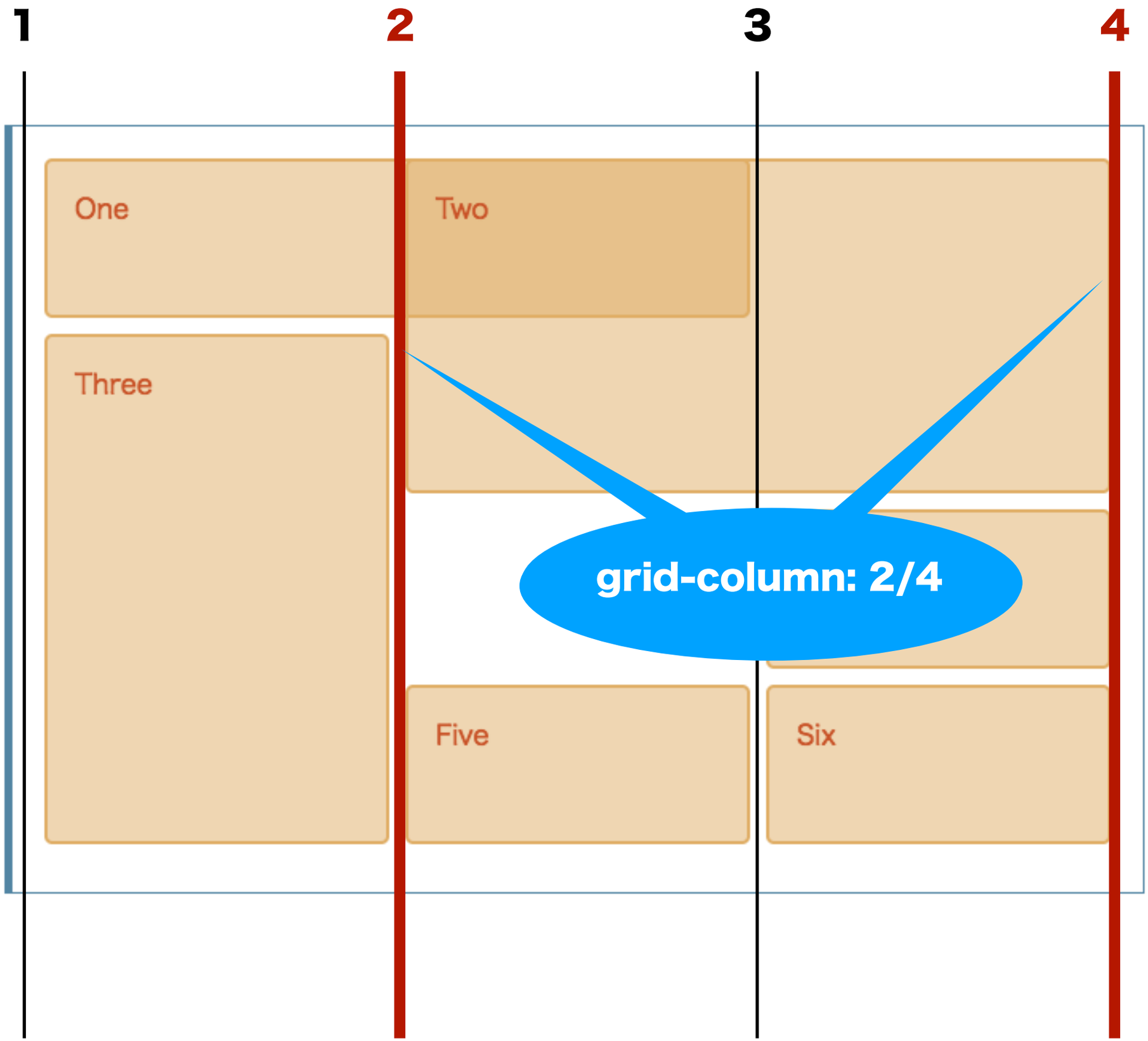
row(縦）は
100pxを最小、autoを最大
(mimax)

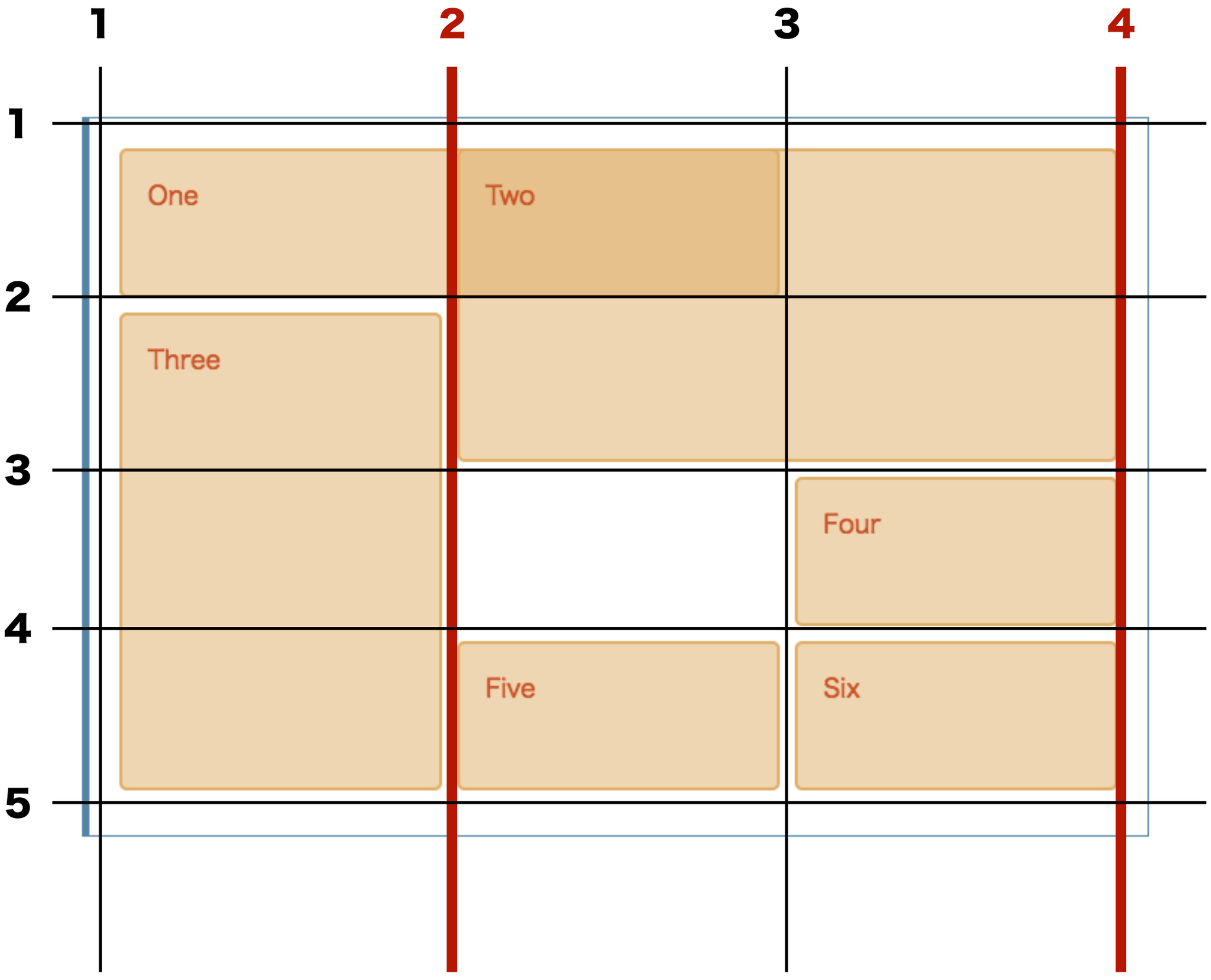
例えば.twoの要素を見てみる

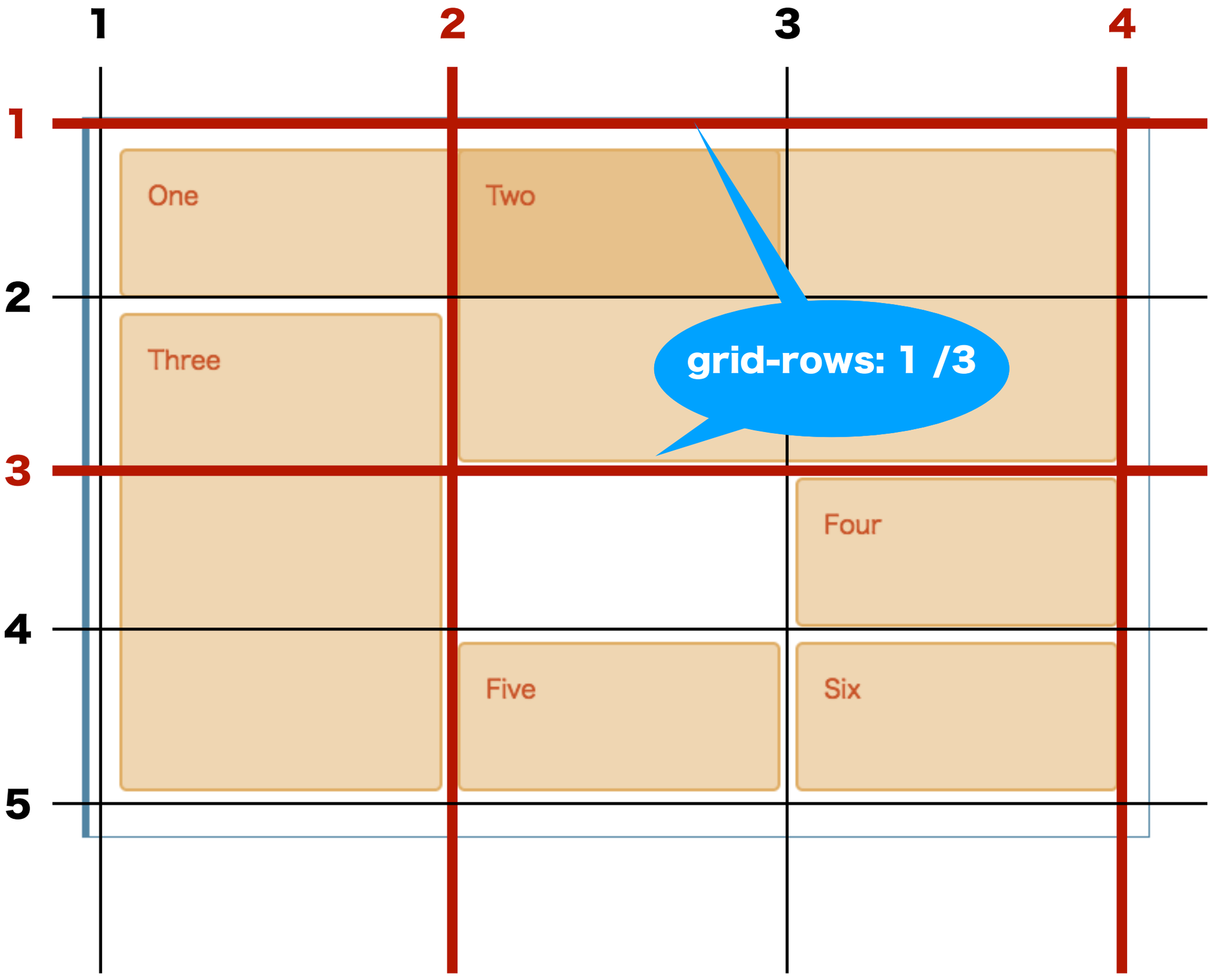
```
}
.two {
  grid-column: 2 / 4;
  grid-row: 1 / 3;
}
```

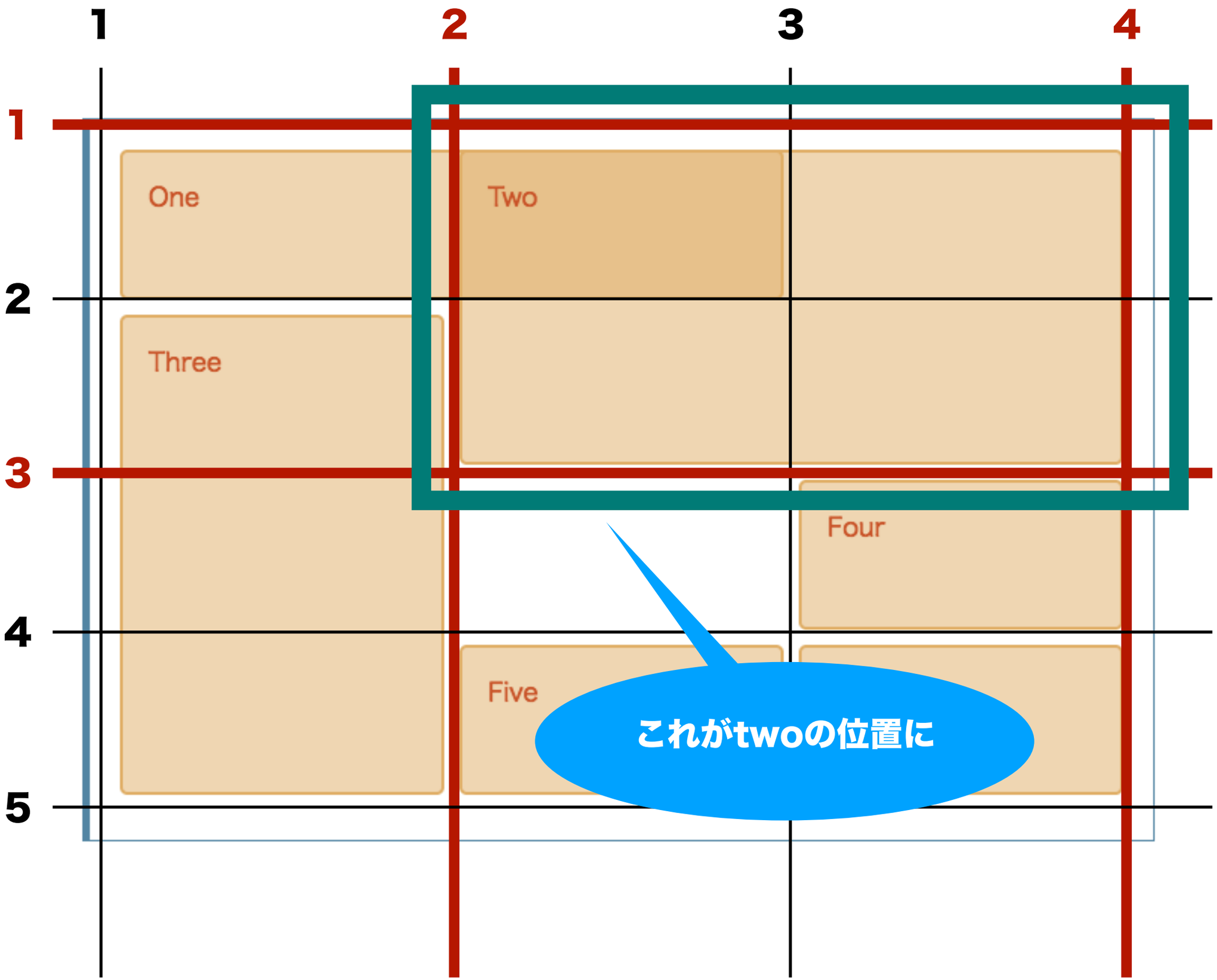
始点の線 / 終点の線を
縦横で指定


1
2
3
4
One
Two
Three
Four
Five
Six


1
2
3
4
One
Two
Three
Five
Six
grid-column: 2/4


1
2
3
4
1
2
3
4
5
One
Two
Three
Four
Five
Six


1
2
3
4
1
2
3
4
5
One
Two
Three
Four
Five
Six
grid-rows: 1 /3


1
2
3
4
1
2
3
4
5
One
Two
Three
Four
Five
これがtwoの位置に

もう一つのGrid指定

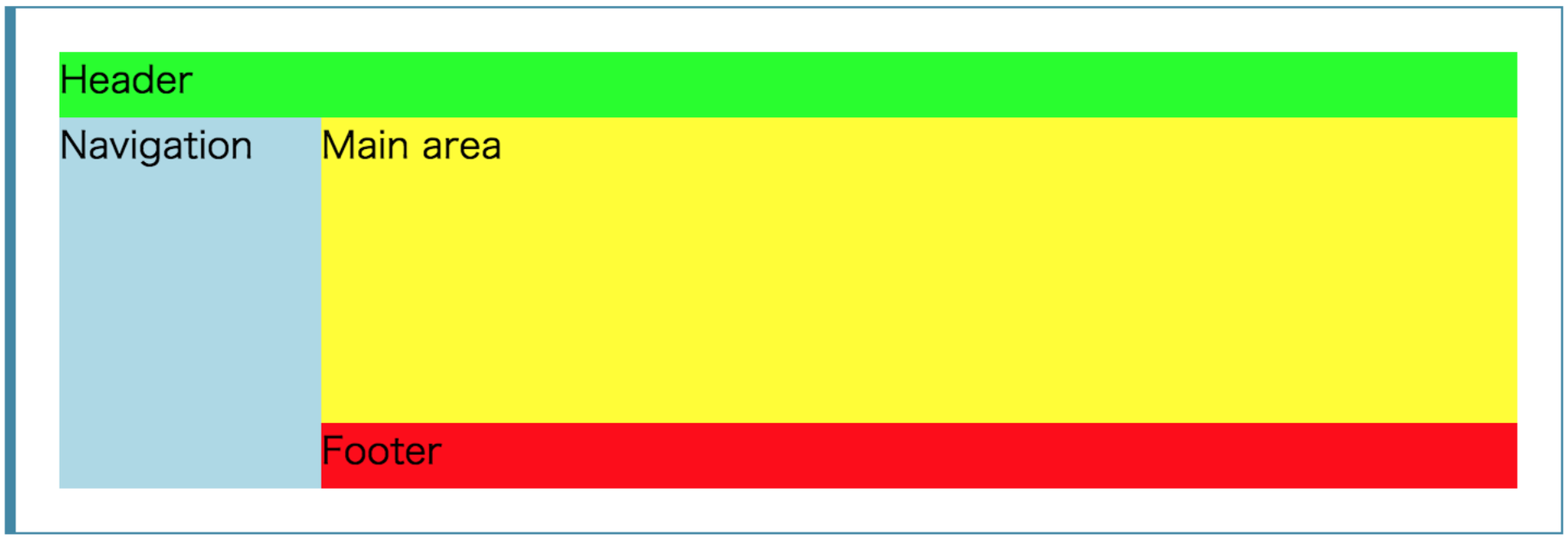
grid-template(-area)

複雑なレイアウトを

HTML

```
<section id="page">
  <header>Header</header>
  <nav>Navigation</nav>
  <main>Main area</main>
  <footer>Footer</footer>
</section>
```

Result



https://developer.mozilla.org/ja/docs/Web/CSS/grid-template

文字列での指定

```
#page {
  display: grid;
  width: 100%;
  height: 200px;
  grid-template: [header-left] "head head" 30px [header-right]
                 [main-left]   "nav  main" 1fr  [main-right]
                 [footer-left] "nav  foot" 30px [footer-right]
                 / 120px 1fr;
}
```

https://developer.mozilla.org/ja/docs/Web/CSS/grid-template

文字列での指定

areaを文字列で指定

```
#page {
  display: grid;
  width: 100%;
  height: 200px;
  grid-template: [header-left]  "head head" 30px [header-right]
                 [main-left]    "nav  main" 1fr  [main-right]
                 [footer-left]  "nav  foot" 30px [footer-right]
                 / 120px 1fr;
}
```

https://developer.mozilla.org/ja/docs/Web/CSS/grid-template

文字列での指定

```
#page {
  display: grid;
  width: 100%;
  height: 200px;
  grid-template: [header-left]  "head head" 30px [header-right]
                 [main-left]    "nav  main" 1fr  [main-right]
                 [footer-left]  "nav  foot" 30px [footer-right]
                 / 120px 1fr;
}
```

areaを文字列で指定

grid-template-areasの場合は
ここだけの記述でOK

https://developer.mozilla.org/ja/docs/Web/CSS/grid-template

文字列での指定

```
#page {
  display: grid;
  width: 100%;
  height: 200px;
  grid-template: [header-left] "head head" 30px [header-right]
                 [main-left]   "nav  main" 1fr  [main-right]
                 [footer-left] "nav  foot" 30px [footer-right]
                 / 120px 1fr;
}
```

「線」に名前付けも出来る

https://developer.mozilla.org/ja/docs/Web/CSS/grid-template

文字列での指定

```
#page {
  display: grid;
  width: 100%;
  height: 200px;
  grid-template: [header-left] "head head" 30px [header-right]
                 [main-left]   "nav  main" 1fr  [main-right]
                 [footer-left] "nav  foot" 30px [footer-right]
                 / 120px 1fr;
}
```

それぞれの高さを指定

https://developer.mozilla.org/ja/docs/Web/CSS/grid-template

文字列での指定

```
#page {
  display: grid;
  width: 100%;
  height: 200px;
  grid-template: [header-left] "head head" 30px [header-right]
                 [main-left]   "nav  main" 1fr  [main-right]
                 [footer-left] "nav  foot" 30px [footer-right]
                 / 120px 1fr;
}
```

横幅を指定

https://developer.mozilla.org/ja/docs/Web/CSS/grid-template

２つの指定方法があることを意識すると少し混乱しなくなる

- 行や列で指定する方法 => grid-template-rows / grid-template-columns

- 全体をゴリッと指定する方法 => grid-template / grid-template-area

CSS Gridについて
もっと知りたい方は下記などおすすめです

- https://learncssgrid.com/
- https://css-tricks.com/snippets/css/complete-guide-grid/
- https://www.layoutit.com/grid

本題

styled-componentsと
CSS Gridが
手を組むとどうなるのか？

こんな事が起きる

- JavaScriptでareaを処理出来る
-
- gridのレイアウトやareaをコンポーネント化として記述出来る。
-

こんな事が起きる

- JavaScriptでareaを処理出来る
  - ⇒複雑なtemplateもprogrammableに記述可能
- gridのレイアウトやareaをコンポーネント化として記述出来る。
  - ⇒ 繰り返し処理になるコンポーネントも楽に書ける

どんなコードになるか？

```jsx
import React from "react";
import styled from "styled-components";

const template = ["i", "ro", "ha", "ni"].map((area) => `[${area}]`).join(" 1fr ");
// template = [i] 1fr [ro] 1fr [ha] 1fr [ni]

const GridLayout = styled.div`
  display: grid;
  grid-template-columns: ${template};
`;

const Area = styled.div`
  grid-column: ${(props) => props.area};
`;

export const MyComponent = () => {
  return (
    <GridLayout>
      <Area area="i">い</Area>
      <Area area="ro">ろ</Area>
      <Area area="ha">は</Area>
      <Area area="ni">に</Area>
    </GridLayout>
  );
};
```

```jsx
import React from "react";
import styled from "styled-components";

const template = ["i", "ro", "ha", "ni"].map((area) => `[${area}]`).join(" 1fr ");
// template = [i] 1fr [ro] 1fr [ha] 1fr [ni]

const GridLayout = styled.div`
  display: grid;
  grid-template-columns: ${template};
`;

const Area = styled.div`
  grid-column: ${(props) => props.area};
`;

export const MyComponent = () => {
  return (
    <GridLayout>
      <Area area="i">い</Area>
      <Area area="ro">ろ</Area>
      <Area area="ha">は</Area>
      <Area area="ni">に</Area>
    </GridLayout>
  );
};
```

templateをJavaScript側で生成

```jsx
import React from "react";
import styled from "styled-components";

const template = ["i", "ro", "ha", "ni"].map((area) => `[${area}]`).join(" 1fr ");
// template = [i] 1fr [ro] 1fr [ha] 1fr [ni]

const GridLayout = styled.div`
  display: grid;
  grid-template-columns: ${template
`;

const Area = styled.div`
  grid-column: ${(props) => props.a
`;

export const MyComponent = () => {
  return (
    <GridLayout>
      <Area area="i">い</Area>
      <Area area="ro">ろ</Area>
      <Area area="ha">は</Area>
      <Area area="ni">に</Area>
    </GridLayout>
  );
};
```

この場合は
[i] 1fr [ro] 1fr [ha] 1fr [ni]
みたいなtemplateが作られる

```jsx
import React from "react";
import styled from "styled-components";

const template = ["i", "ro", "ha", "ni"].map((area) => `[${area}]`).join(" 1fr ");
// template = [i] 1fr [ro] 1fr [ha] 1fr [ni]

const GridLayout = styled.div`
  display: grid;
  grid-template-columns: ${template};
`;

const Area = styled.div`
  grid-column: ${(props) => props.area};
`;

export const MyComponent = () => {
  return (
    <GridLayout>
      <Area area="i">い</Area>
      <Area area="ro">ろ</Area>
      <Area area="ha">は</Area>
      <Area area="ni">に</Area>
    </GridLayout>
  );
};
```

Gridのコンポーネントを作る

```
import React from "react";
import styled from "styled-components";

const template = ["i", "ro", "ha", "ni"].map((area) => `[${area}]`).join(" 1fr ");
// template = [i] 1fr [ro] 1fr [ha] 1fr [ni]

const GridLayout = styled.div`
  display: grid;
  grid-template-columns: ${template};
`;

const Area = styled.div`
  grid-column: ${(props) => props.area};
`;

export const MyCompo
  return (
    <GridLayout>
      <Area area="i">い
      <Area area="ro">ろ</Area>
      <Area area="ha">は</Area>
      <Area area="ni">に</Area>
    </GridLayout>
  );
};
```

Areaは引数を取る形で
コンポーネント化出来る

```jsx
import React from "react";
import styled from "styled-components";

const template = ["i", "ro", "ha", "ni"].map((area) => `[${area}]`).join(" 1fr ");
// template = [i] 1fr [ro] 1fr [ha] 1fr [ni]

const GridLayout = styled.div`
  display: grid;
  grid-template-columns: ${template};
`;

const Area = styled.div`
  grid-column: ${(props) => props.area};
`;

export const MyComponent = () => {
  return (
    <GridLayout>
      <Area area="i">い</Area>
      <Area area="ro">ろ</Area>
      <Area area="ha">は</Area>
      <Area area="ni">に</Area>
    </GridLayout>
  );
};
```

AreaとGridを組み合わせる！

Areaの要素はこんな風にも処理可能

```
const Items = () => {
  const items = [
    { area: "i", content: "い" },
    { area: "ro", content: "ろ" },
    { area: "ha", content: "は" },
    { area: "ni", content: "に" }
  ];
  return (
    <>
      {items.map(({ area, content }) => {
        return <Area area={area}>{content}</Area>;
      })}
    </>
  );
};

export const MyComponent = () => {
  return (
    <GridLayout>
      <Items />
    </GridLayout>
  );
};
```

他には例えばこんな
twitterっぽいレイアウトなら

terrierscript

@terrierscript

あのイーハトーヴォの すきとおった風、 夏でも底に冷たさをもつ青いそら、 うつくしい森で飾られたモーリオ市、 郊外のぎらぎらひかる草の波。

こんなareaを
設定してあげて

```
const template = `
  "avater name  name  .     .  menu"
  ".      body  body  body  body  body"
  ".      com   like  rt    .     ."
`;

const Grid = styled.div`
  display: grid;
  grid-template: ${template};
  background: white;
  border-radius: 8px;
  border: 1px solid #ccc;
  padding: 0.5em;
  grid-gap: 0.2em;
`;
```

要素を並べると

```
export const Twitter = () => {
  return (
    <Grid>
      <Area area={"avater"}><Avater src={icon} /></Area>
      <Area area={"name"}>
        <Name>terrierscript</Name>
        <UserName>@terrierscript</UserName>
      </Area>
      <Area area={"menu"}><Fa icon={faAngleDown} color="#aaa" /></Area>
      <Area area={"body"}>
        <Body>
          あのイーハトーヴォの すきとおった風、 夏でも底に冷たさをもつ青いそら、
          うつくしい森で飾られたモーリオ市、 郊外のぎらぎらひかる草の波。
        </Body>
      </Area>
      <Area area={"com"}><Fa icon={faComment} color="#aaa" /></Area>
      <Area area={"like"}><Fa icon={faHeart} color="#aaa" /></Area>
      <Area area={"rt"}><Fa icon={faRetweet} color="#aaa" /></Area>
    </Grid>
  );
};
```

要素を並べると

```
export const Twitter = () => {
  return (
    <Grid>
      <Area area={"avater"}><Avater src={icon} /></Area>
      <Area area={"name"}>
        <Name>terrierscript</Name>
        <UserName>@terrierscript</UserName>
      </Area>
      <Area area={"menu"}><Fa icon={faAngleDown} color="#aaa" /></Area>
      <Area area={"body"}>
        <Body>
          あのイーハトーヴォの すきとおった風、 夏でも底に冷たさをもつ青いそら、
          うつくしい森で飾られたモーリ ン市、 郊外のぎらぎらひかる草の波。
        </Body>
      </Area>
      <Area area={"com"}><Fa icon={faComment} color="#aaa" /></Area>
      <Area area={"like"}><Fa icon={faHeart} color="#aaa" /></Area>
      <Area area={"rt"}><Fa icon={faRetweet} color="#aaa" /></Area>
    </Grid>
  );
};
```

avater

name

menu

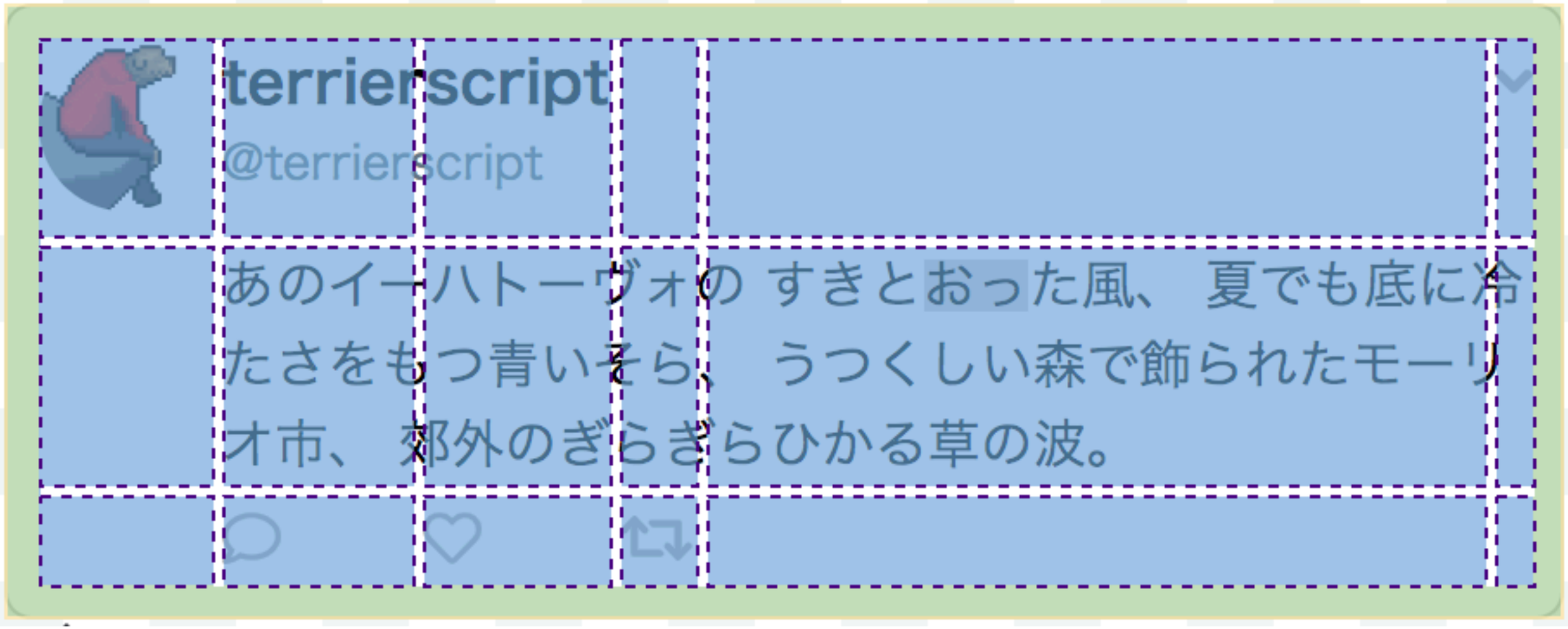
body

comment

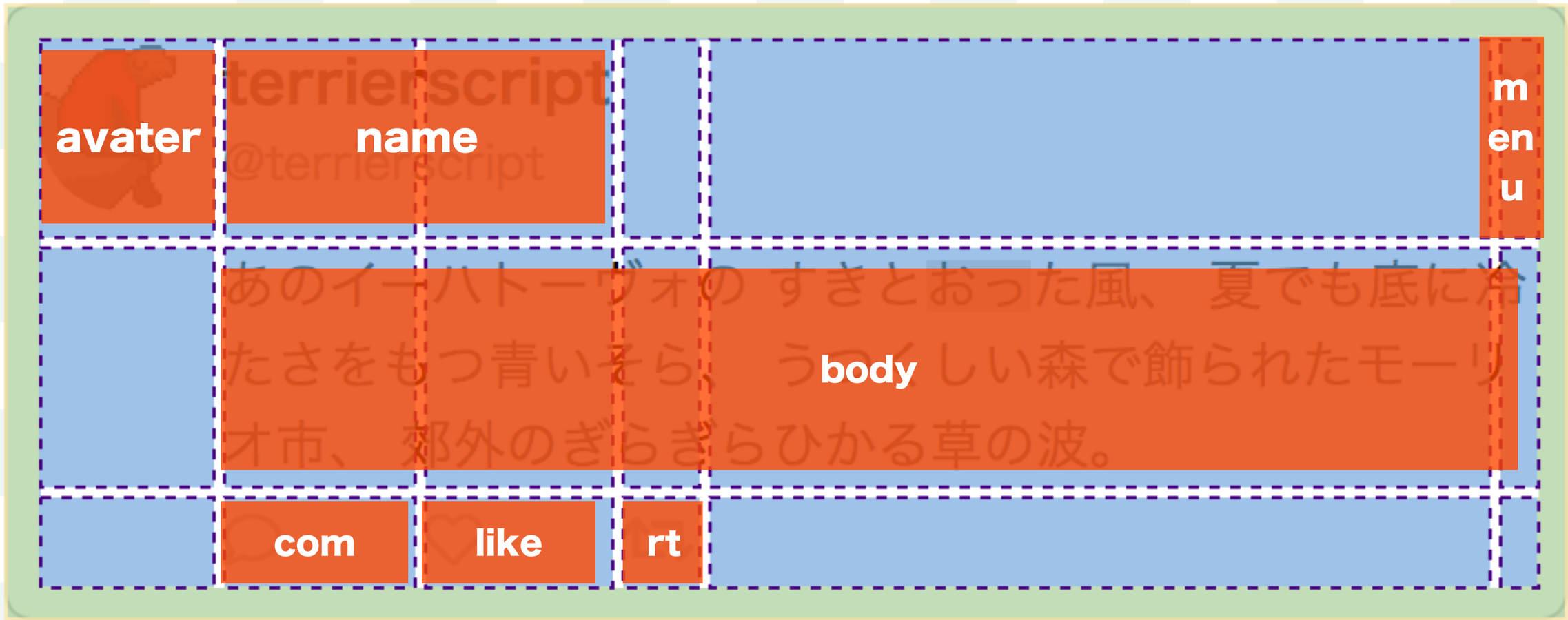
like

rt

こんなふうにGridで表示されるといった具合



こんなふうに割り当てられて
表示されるといった具合



さて、そろそろ感じませんか

css-grid +
styled-components
から

無限の可能性を…！

ということでここから
可能性を感じる使い方を
列挙していきます

また、今回は無限の可能性を追求した都合上
パフォーマンスやブラウザ依存等については
基本的にあまり考慮していません
ご了承下さい

columns/rows編

カレンダー

まずはGridを指定

```
export const CalendarGrid = styled.div`
  display: grid;
  width: fit-content;
  grid-template-columns: repeat(7, 1fr);
  grid-auto-columns: max-content;
  grid-auto-rows: max-content;
  grid-column-gap: 1.5em;
  grid-row-gap: 1em;
  padding: 1.5em;
`;
```

まずはGridを指定

7日
なのでcolumnsで
repeat(7)

```
export const CalendarGrid = styled.div`
  display: grid;
  width: fit-content;
  grid-template-columns: repeat(7, 1fr);
  grid-auto-columns: max-content;
  grid-auto-rows: max-content;
  grid-column-gap: 1.5em;
  grid-row-gap: 1em;
  padding: 1.5em;
`;
```

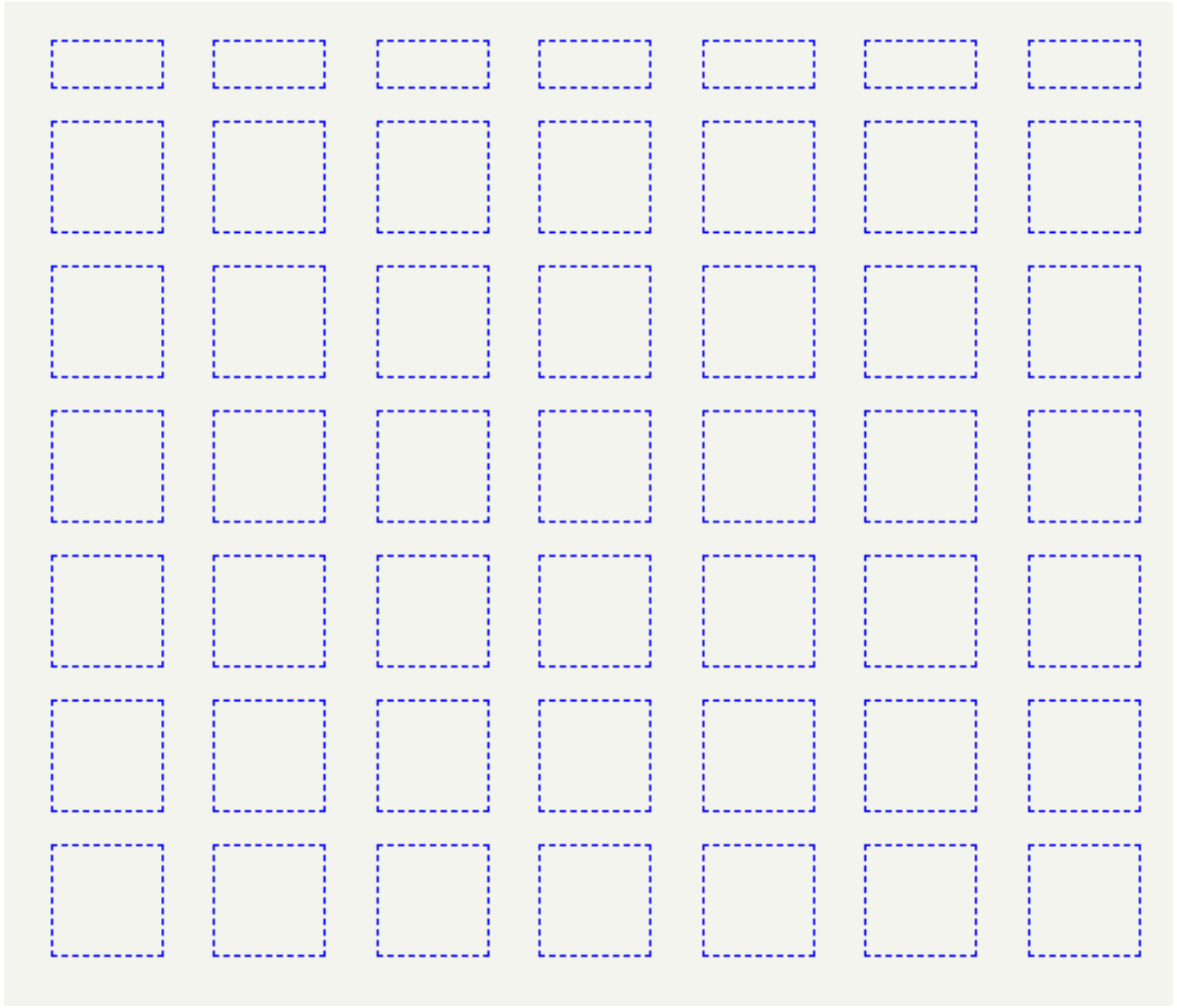
まずはGridを指定

7日
なのでcolumnsで
repeat(7)

```
export const CalendarGrid = styled.div`
  display: grid;
  width: fit-content;
  grid-template-columns: repeat(7, 1fr);
  grid-auto-columns: max-content;
  grid-auto-rows: max-content;
  grid-column-gap: 1.5em;
  grid-row-gap: 1em;
  padding: 1.5em;
`;
```

gap
をrowとcolumnで
変えてあげる

するとこんな感じになります



曜日をつけたら

```
const WeekdaysHeader = () => {
  return (
    <>
      {["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"].map((weekday) => {
        return <WeekdayItem>{weekday}</WeekdayItem>;
      })}
    </>
  );
};
```

こうなって

Sun	Mon	Tue	Wed	Thu	Fri	Sat

日付を入れれば・・・

```
// generateFilledCalendarは↓こんな感じで配列返すみたいなもの
// [
//   null, null, null, null, null, "1", "2",
//   "3", "4", "5", "6", "7", "8", "9",
//   "10", "11", "12", "13", "14", "15", "16",
//   "17", "18", "19", "20", "21", "22", "23",
//   "24", "25", "26", "27", "28", "29", "30",
//   "31", null, null, null, null, null, null
// ]
const Days = ({ year, month }) => {
  return (
    <>
      {generateCalendar(year, month).map((date, i) => {
        return (
          <div key={i}>
            <Day date={date} />
          </div>
        );
      })}
    </>
  );
};
```

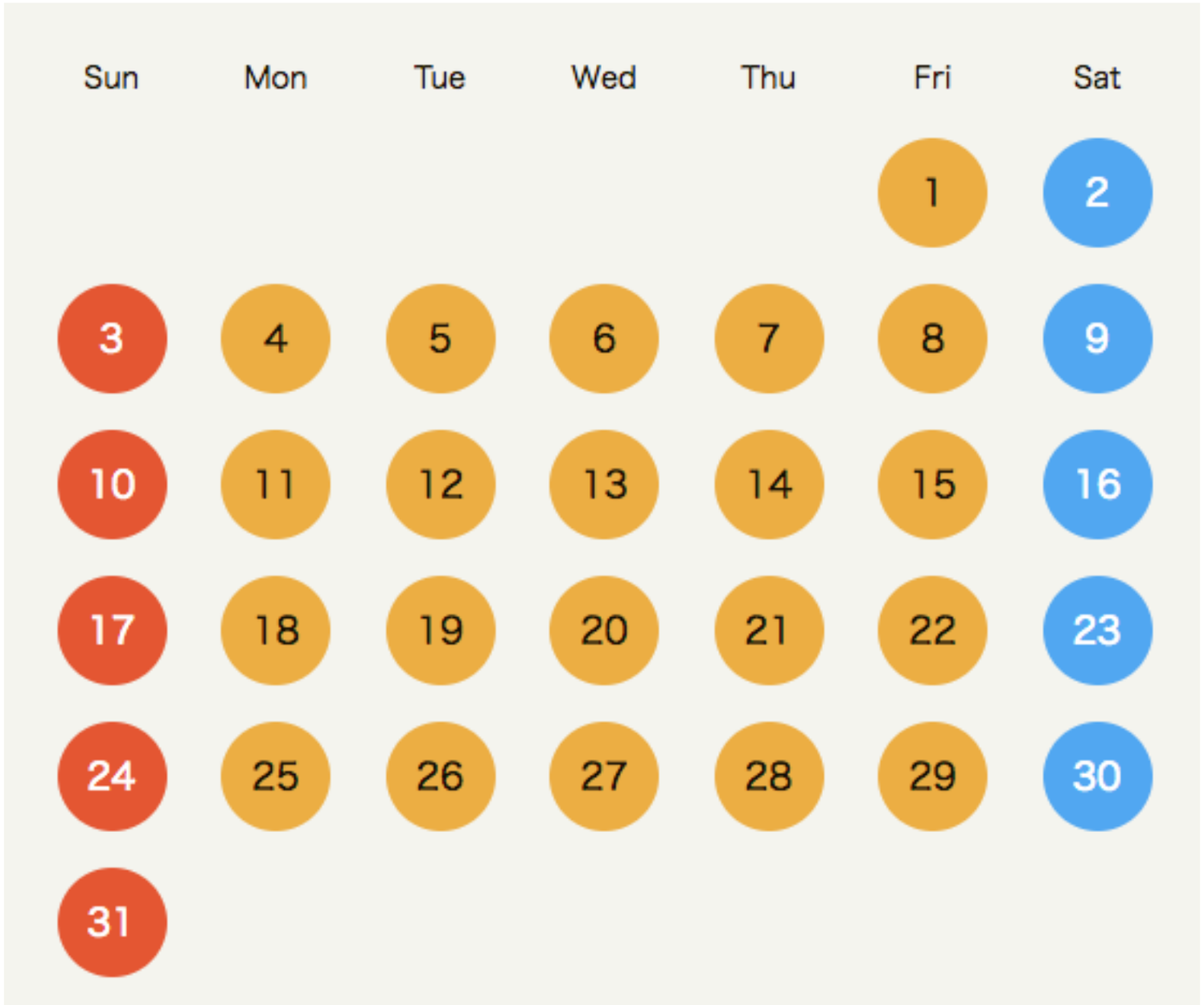
カレンダー！

Sun	Mon	Tue	Wed	Thu	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

あとはちょっと装飾して

```
const colorCss = css<any>`
  color: ${({ fontColor }) => fontColor};
  background-color: ${({ bgColor }) => bgColor};
`;

const size = "3em";
const DayItem: any = styled.div`
  text-align: center;
  vertical-align: middle;
  height: ${size};
  line-height: ${size};
  font-size: 1em;
  width: ${size};
  border-radius: 50%;
  ${colorCss};
`;
```

はいできた！


Sun
Mon
Tue
Wed
Thu
Fri
Sat
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

タイムスケジュール表

今度は少し複雑

```
const times = rangeTimes(); // 時間を15分単位で排出
const rowTemplate = ["[t-header]"]
  .concat(times.map((time) => `[t-${time.hour}${time.min}]`))
  .join(" 1fr ");
// rowTemplateは↓こんな感じの文字列に
// [t-1000] 1fr [t-1015] 1fr [t-1030] 1fr [t-1045] 1fr [t-1100] 1fr [t-1115]
1fr [t-1130] ..

const Grid = styled.div`
  display: grid;
  background: white;
  box-sizing: border-box;
  grid-template-columns: [time] 0.5fr [taro] 1fr [jiro] 1fr [hanako] 1fr;
  grid-template-rows: ${rowTemplate};
`;
```

今度は少し複雑

```
const times = rangeTimes(); // 時間を15分単位で排出
const rowTemplate = ["[t-header]"]
  .concat(times.map((time) => `[t-${time.hour}${time.min}]`))
  .join(" 1fr ");
// rowTemplateは↓こんな感じの文字列に
// [t-1000] 1fr [t-1015] 1fr [t-1030] 1fr [t-1045] 1fr [t-1100] 1fr [t-1115]
1fr [t-1130] ..

const Grid = styled.div`
  display: grid;
  background: white;
  box-sizing: border-box;
  grid-template-columns: [time] 0.5fr [taro] 1fr [jiro] 1fr [hanako] 1fr;
  grid-template-rows: ${rowTemplate};
`;
```

線に時間で名前付けし
てあげる

今度は少し複雑

```
const times = rangeTimes(); // 時間を15分単位で排出
const rowTemplate = ["[t-header]"]
  .concat(times.map((time) => `[t-${time.hour}${time.min}]`))
  .join(" 1fr ");
// rowTemplateは↓こんな感じの文字列に
// [t-1000] 1fr [t-1015] 1fr [t-1030] 1fr [t-1045] 1fr [t-1100] 1fr [t-1115]
1fr [t-1130] ..

const Grid = styled.div`
  display: grid;
  background: white;
  box-sizing: border-box;
  grid-template-columns: [time] 0.5fr [taro] 1fr [jiro] 1fr [hanako] 1fr;
  grid-template-rows: ${rowTemplate};
`;
```

縦線は人で名前付け

Areaとして縦方向は
startとendを指定出来るものを用意

```
const Area = styled.div`
  grid-column: ${({ column }) => column};
  grid-row: ${({ rowStart, rowEnd }) => {
    if (rowEnd) {
      return `t-${rowStart} / t-${rowEnd}`;
    }
    return `t-${rowStart}`;
  }};
`;
```

線を定義して

```
const Border = styled(Area)`
  border-left: solid 1px #ccc;
  min-height: 3em;
  border-top: ${({ color = "#ccc" }) => `solid 1px ${color}`};
`;
```

書きます

```
const Borders = () => {
  const elms = rows.map((column) => {
    return times.map((time, i) => (
      <Border
        color={time.min == "00" ? "#333" : "#ccc"}
        column={column}
        rowStart={`${time.hour}${time.min}`}
        key={`${column}-${i}`}
      />
    ));
  });
  return flatten(elms);
};
```

線が書けました！

時間とヘッダを載せます

```
export const Timetable = () => {
  return (
    <Grid>
      <Borders />
      <Headers />
      <Times />
    </Grid>
  );
};
```

こうなります

	taro	jiro	hanako
10:00			
10:15			
10:30			
10:45			
11:00			
11:15			
11:30			
11:45			
12:00			
12:15			
12:30			
12:45			

スケジュールのっければ・・

```
export const Timetable = () => {
  return (
    <Grid>
      <Borders />
      <Headers />
      <Times />

      <Schedule name="taro" start={"1030"} end={"1145"}>
        外出
      </Schedule>
      <Schedule name="taro" start={"1345"} end={"1600"}>
        外出
      </Schedule>
      <Schedule name="hanako" start={"1130"} end={"1200"}>
        お昼
      </Schedule>
      <Schedule name="jiro" start={"1230"} end={"1400"}>
        会議
      </Schedule>
      <Schedule name="hanako" start={"1230"} end={"1400"}>
        会議
      </Schedule>
    </Grid>
  );
};
```

出来上がり！

	taro	jiro	hanako
10:00			
10:15			
10:30	**10:30** **[taro]: 外出**		
10:45			
11:00			
11:15			
11:30			**11:30** **[hanako]: お昼**
11:45			
12:00			
12:15			
12:30		**12:30** **[jiro]: 会議**	**12:30** **[hanako]: 会議**
12:45			

ところで

```
export const Timetable = () => {
  return (
    <Grid>
      <Borders />
      <Headers />
      <Times />

      <Schedule name="taro" start={"1030"} end={"1145"}>
        外出
      </Schedule>
      <Schedule name="taro" start={"1345"} end={"1600"}>
        外出
      </Schedule>
      <Schedule name="hanako" start={"1130"} end={"1200"}>
        お昼
      </Schedule>
      <Schedule name="jiro" start={"1230"} end={"1400"}>
        会議
      </Schedule>
      <Schedule name="hanako" start={"1230"} end={"1400"}>
        会議
      </Schedule>
    </Grid>
  );
};
```

ところで

```tsx
export const Timetable = () => {
  return (
    <Grid>
      <Borders />
      <Headers />
      <Times />

      <Schedule name="taro" start={"1030"} end={"1145"}>
        外出
      </Schedule>
      <Schedule name="taro" start={"1345"} end={"1600"}>
        外出
      </Schedule>
      <Schedule name="hanako" start={"1130"} end={"1200"}>
        お昼
      </Schedule>
      <Schedule name="jiro" start={"1230"} end={"1400"}>
        会議
      </Schedule>
      <Schedule name="hanako" start={"1230"} end={"1400"}>
        会議
      </Schedule>
    </Grid>
  );
};
```

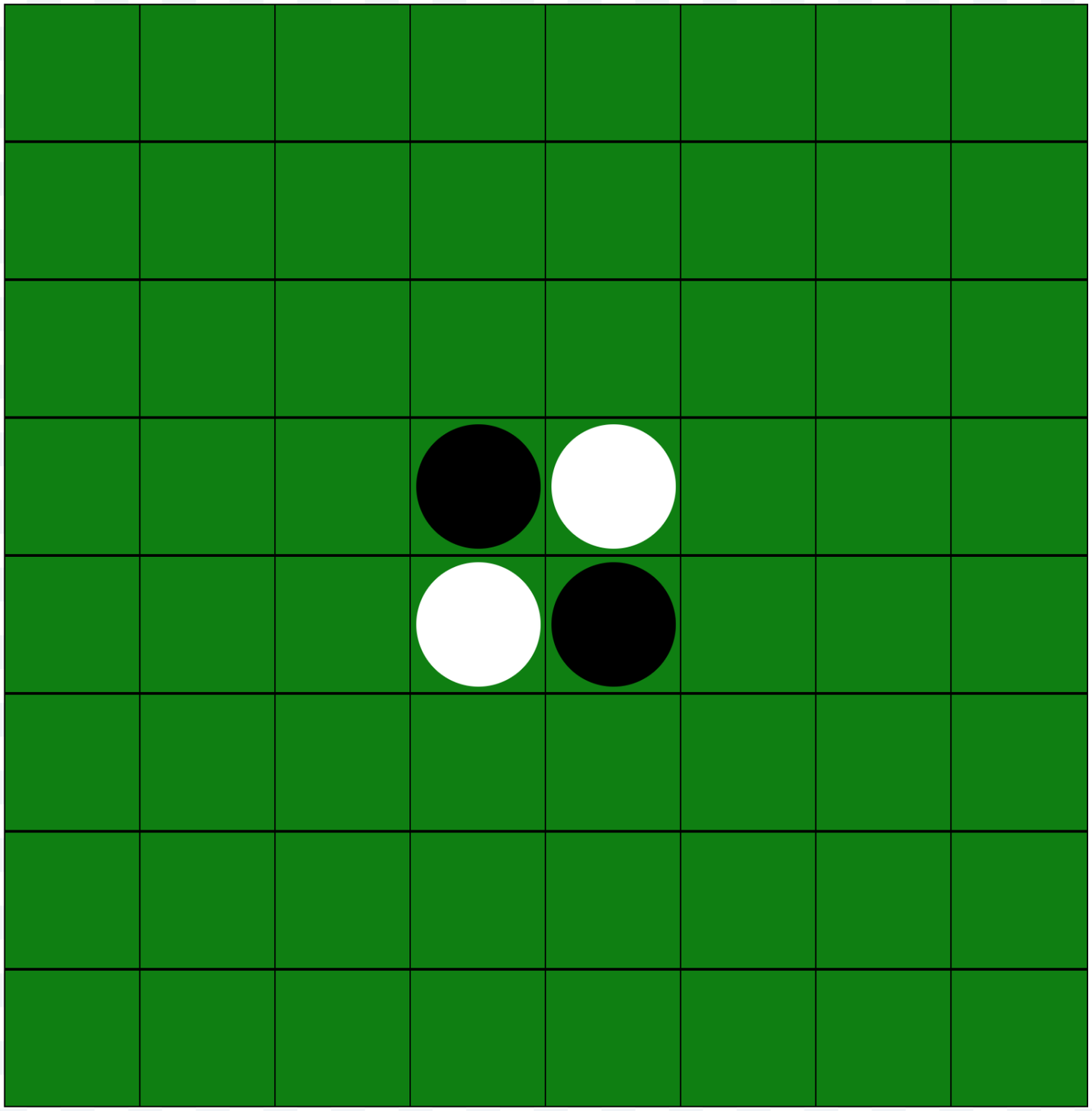
Scheduleを表から分離して書けるの
めちゃくちゃ良くないですか

同じ要領で

こんな風にすれば

```tsx
// @ts-ignore
const Board: SFC<{}> = () => {
  return Array.from({ length: 8 }).map((_, i) => {
    return Array.from({ length: 8 }).map((_, j) => {
      const row = i + 1;
      const column = j + 1;
      return <BoardCell key={`${row}_${column}`} row={row} column={column} />;
    });
  });
};

export const Othero = () => {
  return (
    <Grid>
      <Board />
      <Piece color="black" row="4" column="4" />
      <Piece color="white" row="4" column="5" />
      <Piece color="white" row="5" column="4" />
      <Piece color="black" row="5" column="5" />
    </Grid>
  );
};
```

オセロ！



さっきと同じなので

```tsx
// @ts-ignore
const Board: SFC<{}> = () => {
  return Array.from({ length: 8 }).map((_, i) => {
    return Array.from({ length: 8 }).map((_, j) => {
      const row = i + 1;
      const column = j + 1;
      return <BoardCell key={`${row}_${column}`} row={row} column={column} />;
    });
  });
};

export const Othero = () => {
  return (
    <Grid>
      <Board />
      <Piece color="black" row="4" column="4" />
      <Piece color="white" row="4" column="5" />
      <Piece color="white" row="5" column="4" />
      <Piece color="black" row="5" column="5" />
    </Grid>
  );
};
```

これも盤とコマを分離して
記述出来る！

areas編

電卓

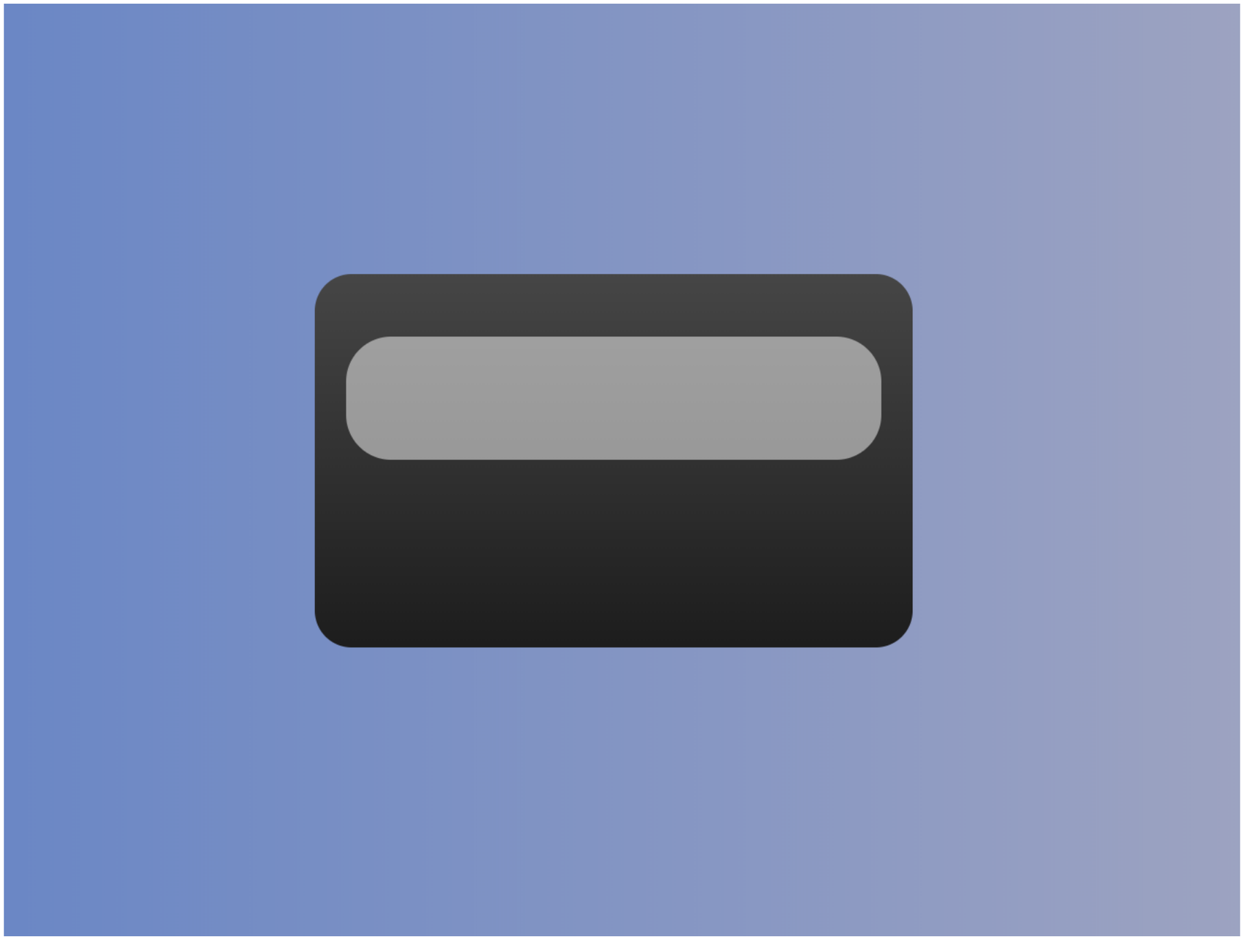
areaはこんな風に指定

```
const template = `
  ". . . so so"
  "ii ii ii ii"
  "ac pm pc di"
  "k7 k8 k9 pw"
  "k4 k5 k6 mi"
  "k1 k2 k3 pl"
  "k0 k0 do eq" / 60px 60px 60px 60px
`;

const Grid = styled.div`
  display: grid;
  grid-template: ${template};
  grid-gap: 1em;
`;
```

inputを作って

```
export const Calcurator = () => {
  return (
    <Container>
      <BlackBackground>
        <Grid>
          <Area area="ii">
            <Input />
          </Area>
        </Grid>
      </BlackBackground>
    </Container>
  );
};
```

inputを作って



ボタンを並べたら…

```
const Operator = () => (
  <>
    <Area area="di">
      <OrangeButton>÷</OrangeButton>
    </Area>
    <Area area="pw">
      <OrangeButton>×</OrangeButton>
    </Area>
    <Area area="mi">
      <OrangeButton>−</OrangeButton>
    </Area>
    <Area area="pl">
      <OrangeButton>+</OrangeButton>
    </Area>
    <Area area="do">
      <NumberButton>.</NumberButton>
    </Area>
    <Area area="eq">
      <OrangeButton>=</OrangeButton>
```

こう！

数字ボタンも並べて

```
const Numbers = () => {
  return (
    <>
      {Array.from({ length: 10 }).map((_, i) => {
        return (
          <Area area={`k${i}`} key={i}>
            <NumberButton>{i}</NumberButton>
          </Area>
        );
      })}
    </>
  );
};
```

出来上がり！

リーンキャンパス

こうなって・・・

```
const template = `
  "s1 s1 s4 s4 s3 s3 s9 s9 s2 s2"
  "s1 s1 s4 s4 s3 s3 s9 s9 s2 s2"
  "s1 s1 s8 s8 s3 s3 s5 s5 s2 s2"
  "s1 s1 s8 s8 s3 s3 s5 s5 s2 s2"
  "s7 s7 s7 s7 s7 s6 s6 s6 s6 s6"
  "s7 s7 s7 s7 s7 s6 s6 s6 s6 s6"
`;
```

こうなって・・・

```
const template = `
  "s1 s1 s4 s4 s3 s3 s9 s9 s2 s2"
  "s1 s1 s4 s4 s3 s3 s9 s9 s2 s2"
  "s1 s1 s8 s8 s3 s3 s5 s5 s2 s2"
  "s1 s1 s8 s8 s3 s3 s5 s5 s2 s2"
  "s7 s7 s7 s7 s7 s6 s6 s6 s6 s6"
  "s7 s7 s7 s7 s7 s6 s6 s6 s6 s6"
`;
```

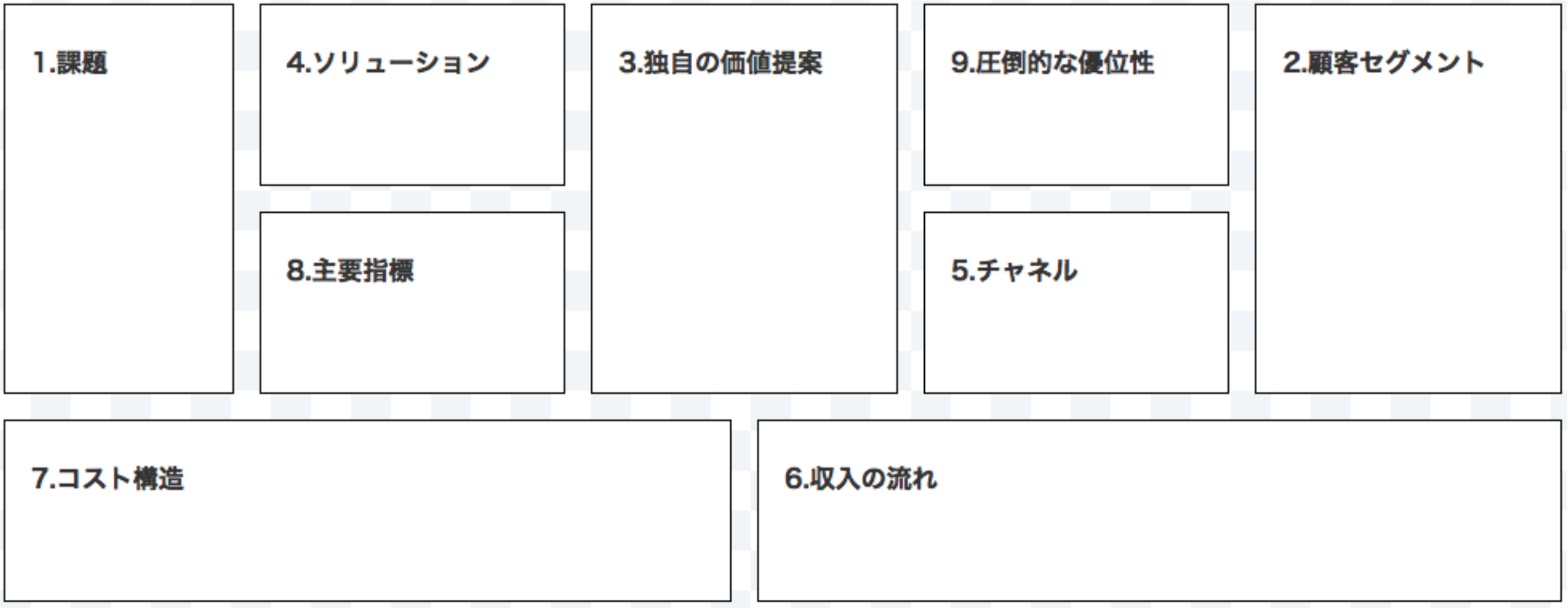
要素作って・・・

```
const Areas = ({ canvasItem }) => {
  const areas = [
    { area: "s1", name: "1.課題" },
    { area: "s2", name: "2.顧客セグメント" },
    { area: "s3", name: "3.独自の価値提案" },
    { area: "s4", name: "4.ソリューション" },
    { area: "s5", name: "5.チャネル" },
    { area: "s6", name: "6.収入の流れ" },
    { area: "s7", name: "7.コスト構造" },
    { area: "s8", name: "8.主要指標" },
    { area: "s9", name: "9.圧倒的な優位性" }
  ];
  return (
    <>
      {areas.map(({ area, name }, i) => {
        return (
          <Canvas style={{ gridArea: area }} key={i}>
            <Title>{name}</Title>
            <Edit contentEditable>{canvasItem[i]}</Edit>
          </Canvas>
        );
      })}
    </>
  );
};
```

リーンキャンパス！

1.課題	4.ソリューション	3.独自の価値提案	9.圧倒的な優位性	2.顧客セグメント
	8.主要指標		5.チャネル	
7.コスト構造			6.収入の流れ	

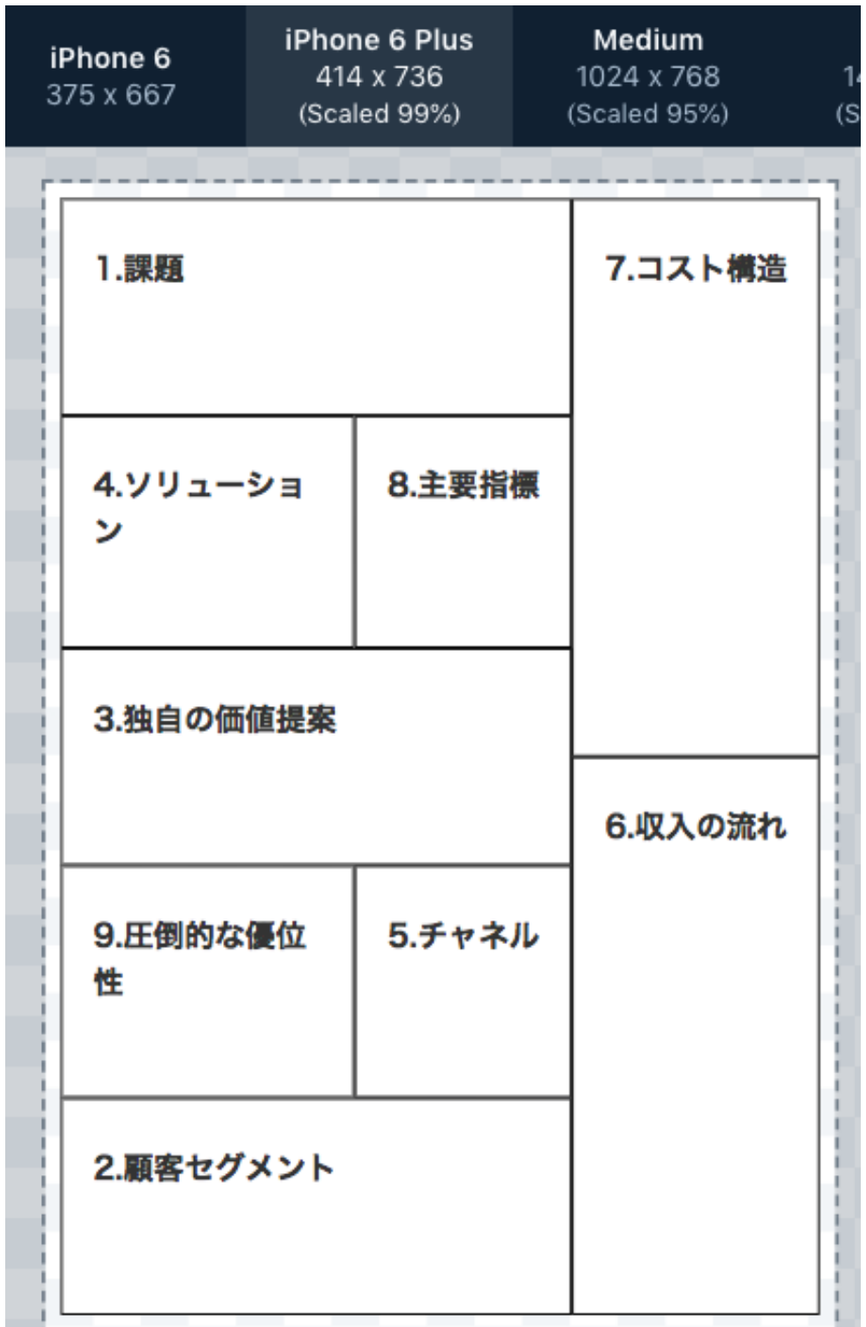
grid-gapつけたらこう



90度回したら・・・

```
const rotate = `
  "s1 s1 s1 s1 s7 s7"
  "s1 s1 s1 s1 s7 s7"
  "s4 s4 s8 s8 s7 s7"
  "s4 s4 s8 s8 s7 s7"
  "s3 s3 s3 s3 s7 s7"
  "s3 s3 s3 s3 s6 s6"
  "s9 s9 s5 s5 s6 s6"
  "s9 s9 s5 s5 s6 s6"
  "s2 s2 s2 s2 s6 s6"
  "s2 s2 s2 s2 s6 s6"
`;
```

レスポンシブ対応！


iPhone 6
375 x 667
iPhone 6 Plus
414 x 736
(Scaled 99%)
Medium
1024 x 768
(Scaled 95%)
1.課題
7.コスト構造
4.ソリューション
8.主要指標
3.独自の価値提案
6.収入の流れ
9.圧倒的な優位性
5.チャネル
2.顧客セグメント

キーボード

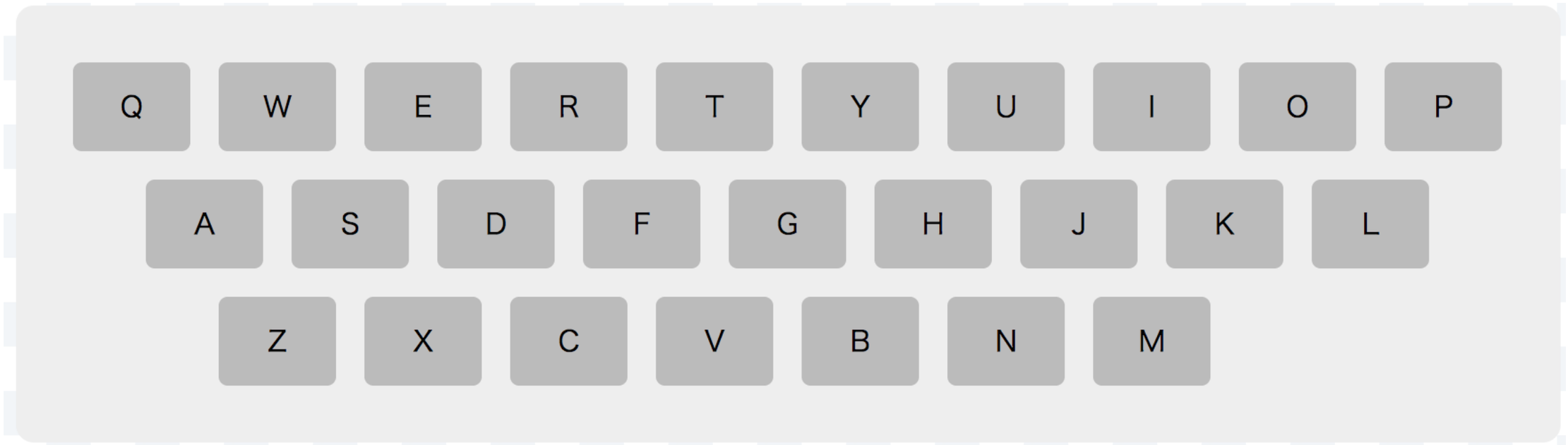
愚直に入力します・・・

```
const template = `
  "qq qq ww ww ee ee rr rr tt tt yy yy uu uu ii ii oo oo pp pp"
  ".. aa aa ss ss dd dd ff ff gg gg hh hh jj jj kk kk ll ll .."
  ".. .. zz zz xx xx cc cc vv vv bb bb nn nn mm mm .. .. .. .."
`;

const Grid = styled.div`
  display: grid;
  grid-template-areas: ${template};
  grid-gap: 1em;
  grid-auto-columns: 25px;
`;
```

キーを並べたら・・・

```
// @ts-ignore
const KeyMaps: SFC<{}> = () => {
  const keys = "abcdefghijklmnopqrstuvwxyz".split("");
  return keys.map((k) => {
    const area = `${k}${k}`;
    return (
      <Area area={area} key={area}>
        <Key>{k.toUpperCase()}</Key>
      </Area>
    );
  });
};

export const Keyboard = () => {
  return (
    <Grid>
      <KeyMaps />
    </Grid>
  );
};
```

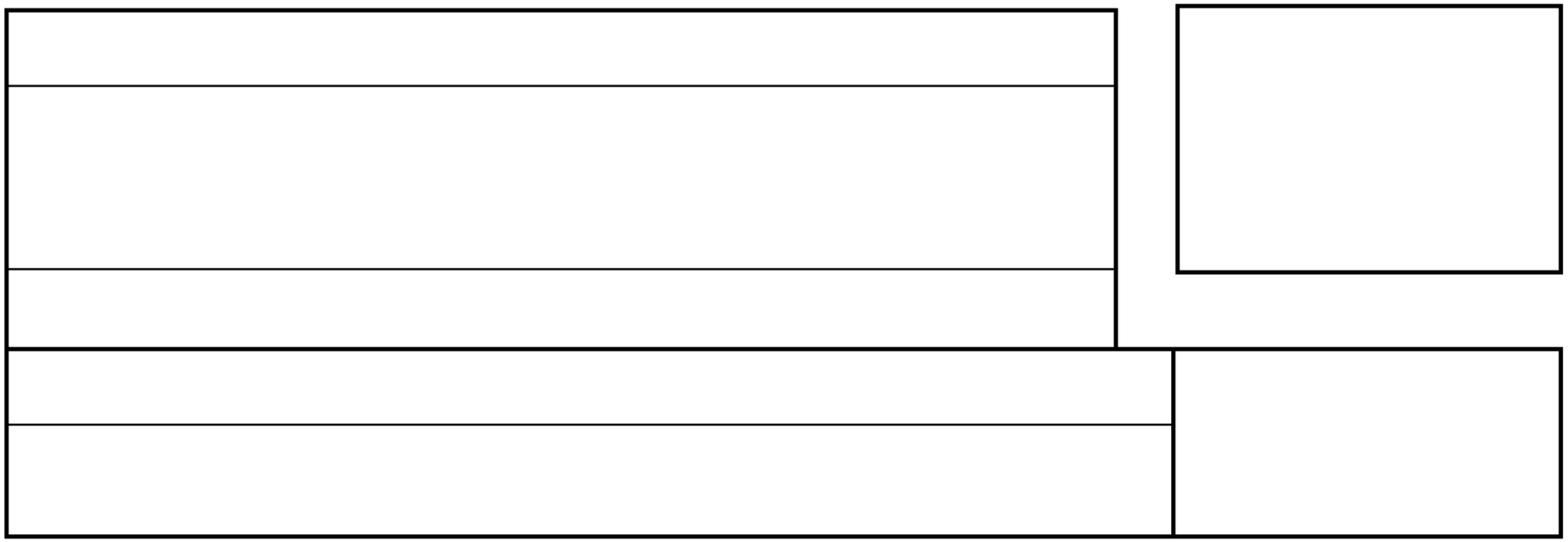
出来上がり！



タイピングゲームとかで使えるかも

履歴書

こういう線がほしい



どうするか

- borderを色々組み合わせて頑張る
  - -> borderはちょこちょこずれる
- Gridの中にGridを入れる
  - -> 一番キレイに書けそう
- 線すらもGridで書いてしまう
  - -> クレイジーで一番楽しそう

どうするか

- borderを色々組み合わせて頑張る
  - -> borderはちょこちょこずれる
- Gridの中にGridを入れる
  - -> 一番キレイに書けそう
- 線すらもGridで書いてしまう
  - -> クレイジーで一番楽しそう

Spreadsheetで下書き

"	line7	line7	line7	line7	.	line1	line1	line1	"	2px
"	line8	kana	kana	line6	.	line2	photo	line3	"	1fr
"	line8	name	name	line6	.	line2	photo	line3	"	1fr
"	line8	name	name	line6	.	line4	line4	line4	"	2px
"	line8	name	name	line6	.	.	.	.	"	1fr
"	line8	birth	gender	line6	.	.	.	.	"	1fr
"	line8	birth	gender	line5	line5	line5	line5	line5	"	2px
"	line8	zip	zip	zip	zip	zip	zip	line10	"	1fr
"	line8	place	place	place	place	place	place	line10	"	1fr
"	line9	line9	line9	line9	line9	line9	line9	line9	"	2px
/	2px	1fr	1fr	2px	1fr	2px	1fr	2px		

templateに移して調整…

```
const template = `
"tl  tl  tl  ts  .  .  .  ln1  ln1  ln1" 2px
".  .  .  .  .  .  .  ln2  photo ln3" 1fr
"ln7 ln7  ln7  ln7  ln7  .  .  ln2  photo ln3" 2px
"ln8 .  kanaf kanal  ln6  .  .  ln2  photo ln3" 2em
"ln8 ln11  ln11  ln11  ln6  .  .  ln2  photo ln3" 1px
"ln8 .  namef namel  ln6  .  .  ln2  photo ln3 " 0.1fr
"ln8 .  namef namel  ln6  .  .  ln2  photo ln3" 5em
"ln8 ln12  ln12  ln12  ln6  .  .  ln2  photo ln3" 1px
"ln8 .  birth gender  ln6  .  .  ln4  ln4  ln4" 2px
"ln8 .  birth gender  ln6  .  .  .  .  .  " 2em
"ln8 ln13  ln13  ln13  ln5  ln5  ln5  ln5  ln5  ln5" 2px
"ln8 .  zip  zip  zip  zip  ln15  phone phone ln10  " 2em
"ln8 ln14  ln14  ln14  ln14  ln14  ln15  phone phone ln10  " 1px
"ln8 .  place  place  place  place  ln15  phone  phone ln10  " 3em
"ln9 ln9  ln9  ln9  ln9  ln9  ln9  ln9  ln9  ln9" 2px
/ 2px 10px  1fr  1fr  2px  0.1fr 2px  2px  0.7fr 2px
`;
```

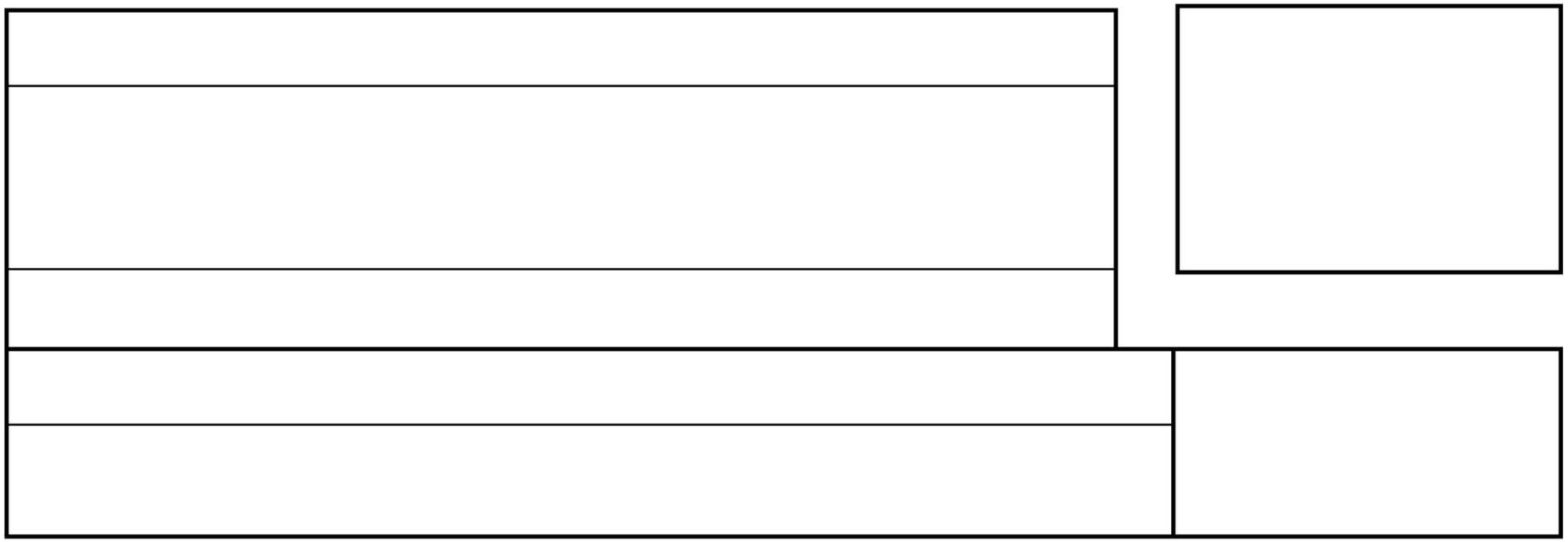
こんな感じで線の領域と太さを決めて

```
const template = `
"  tl  tl  tl  ts  .  .  .  ln1  ln1  ln1  " 2px
"  .  .  .  .  .  .  .  ln2  photo  ln3  " 1fr
"  ln7  ln7  ln7  ln7  ln7  .  .  ln2  photo  ln3  " 2px
"  ln8  .  kanaf  kanal  ln6  .  .  ln2  photo  ln3  " 2em
"  ln8  ln11  ln11  ln11  ln6  .  .  ln2  photo  ln3  " 1px
"  ln8  .  namef  namel  ln6  .  .  ln2  photo  ln3  " 0.1fr
"  ln8  .  namef  namel  ln6  .  .  ln2  photo  ln3  " 5em
"  ln8  ln12  ln12  ln12  ln6  .  .  ln2  photo  ln3  " 1px
"  ln8  .  birth  gender  ln6  .  .  ln4  ln4  ln4  " 2px
"  ln8  .  birth  gender  ln6  .  .  .  .  .  " 2em
"  ln8  ln13  ln13  ln13  ln5  ln5  ln5  ln5  ln5  ln5  " 2px
"  ln8  .  zip  zip  zip  zip  ln15  phone  phone  ln10  " 2em
"  ln8  ln14  ln14  ln14  ln14  ln14  ln15  phone  phone  ln10  " 1px
"  ln8  .  place  place  place  place  ln15  phone  phone  ln10  " 3em
"  ln9  ln9  ln9  ln9  ln9  ln9  ln9  ln9  ln9  ln9  " 2px
/  2px  10px  1fr  1fr  2px  0.1fr  2px  2px  0.7fr  2px
`;
```

線を引きます

```
const Line = styled(Area)`
  background: black;
  width: 100%;
  height: 100%;
`;

const Lines = () => {
  return (
    <>
      {Array.from({ length: 15 }).map((_, i) => {
        const num = i + 1;
        const area = `ln${num}`;
        return (
          <Line area={area} key={area}></Line>
        );
      })}
    </>
  );
};
```

線！

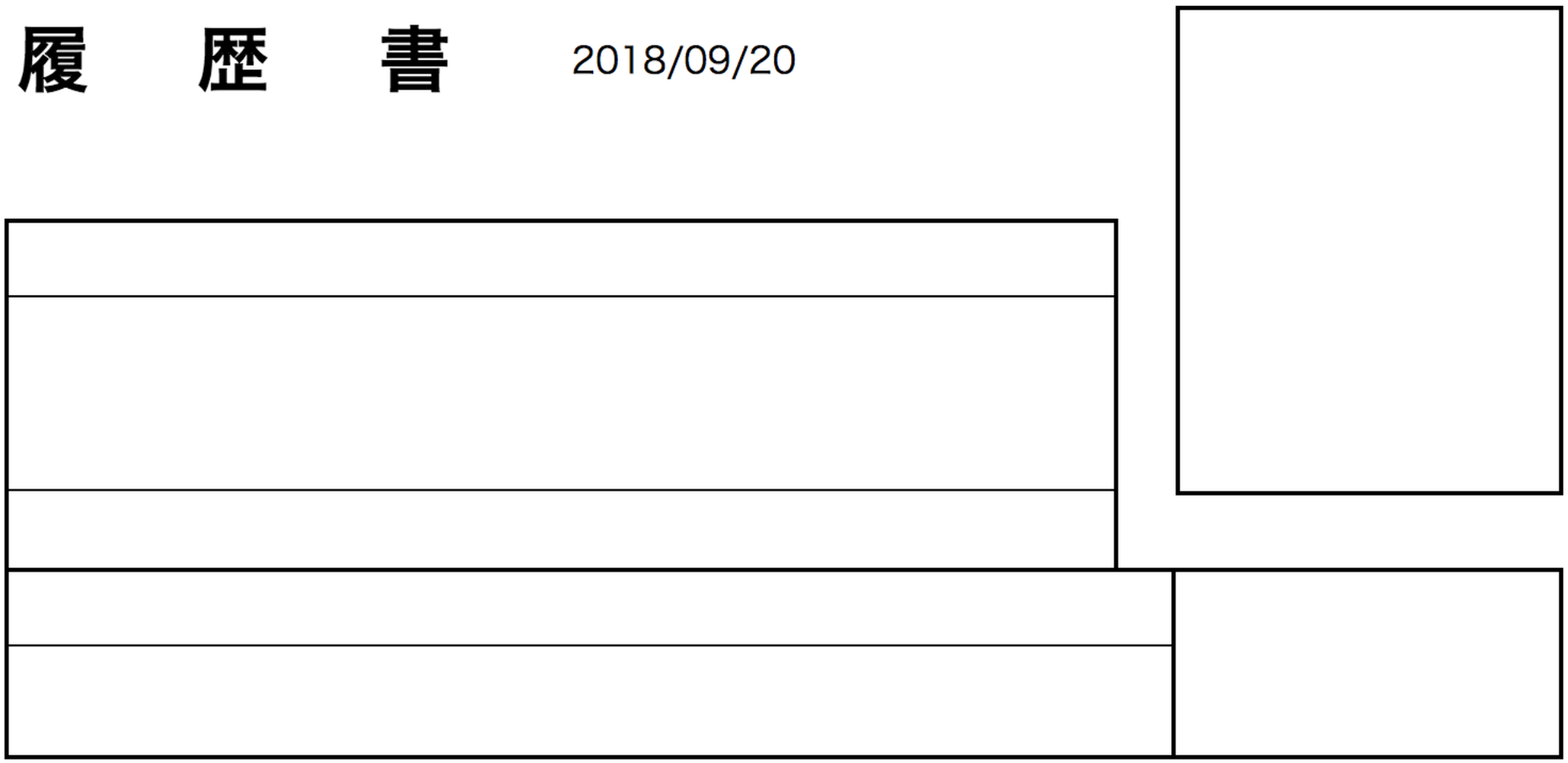


あとは色々のっけていって

```
const BoxArea = styled(Area)`
  padding: 0.2em;
`
export const Resume = () => {
  return (
    <Grid>
      <Lines />
      <Area area={"tl"}>
        <Title>履歴書</Title>
      </Area>
      <Area area={"ts"}>2018/09/20</Area>
      <BoxArea area={"kanaf"}><Kana>しいえす</Kana></BoxArea>
      <BoxArea area={"kanal"}><Kana>えすたろう</Kana></BoxArea>
      <BoxArea area={"namef"}><Name>椎江州</Name></BoxArea>
      <BoxArea area={"namel"}><Name>得太郎</Name></BoxArea>
      <BoxArea area={"gender"}>男</BoxArea>
      <BoxArea area={"birth"}>昭和 60 年 1 月 1 日</BoxArea>
      <BoxArea area={"zip"}>〒 153-0063</BoxArea>
      <BoxArea area={"place"}>東京都目黒区目黒３丁目９−１ 目黒 須田 ビル 5F</BoxArea>
      <BoxArea area={"phone"}>090-1234-5678</BoxArea>

      <Area area={"photo"}>
        <Photo />
      </Area>
    </Grid>
  );
};
```

見出し

履　歴　書　2018/09/20



写真

履　歴　書

2018/09/20

完成！

履　　歴　　書　　2018/09/20

しいえす	えすたろう
椎江州	得太郎
昭和 60 年 1 月 1 日	男

〒 153-0063	090-1234-5678
東京都目黒区目黒３丁目９−１ 目黒 須田 ビル 5F	

他にも思いつくアイディアは
たくさん・・・！

- ガントチャート
- 請求書
- ドット絵ツール・ピクロス
- ピアノ
- などなど・・・

まとめ

- 当たり前ですが普通のGridでもstyled-componentsは十分有益に使えます
- tableでcolspan/rowspan使ってたものは概ねいい感じに出来る可能性が高い
- flexとの使い分けはまだ悩み中
- Excel方眼紙に封印されていた者たちをGridに解き放てそう

__人人人人人人人人人人人__

> キミだけの最強の <

>CSS Gridを作り出せ！<

￣Y^Y^Y^Y^Y^Y^YY^Y^￣