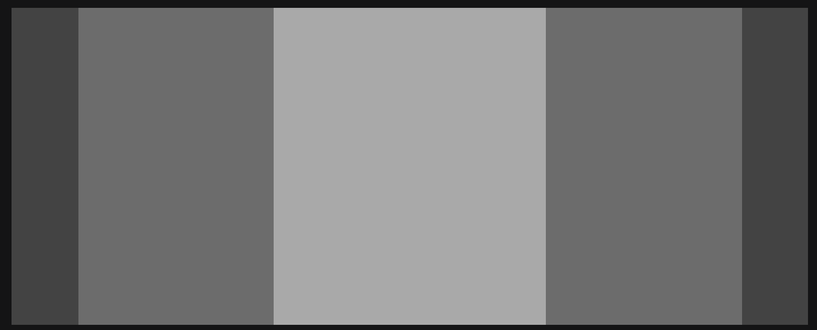
Responsive



Não é só sobre ecrãs,
é um estilo de vida e de senso comum.

Título

I’m curious ,when using the proximity principle in complex interfaces with overlapping elements (like dashboards or data-heavy UIs), how do you strike the right balance between grouping related items and avoiding visual clutter? Any tips or examples?

Título

I’m curious ,when using the proximity principle in complex interfaces with overlapping elements (like dashboards or data-heavy UIs), how do you strike the right balance between grouping related items and avoiding visual clutter? Any tips or examples?

Título

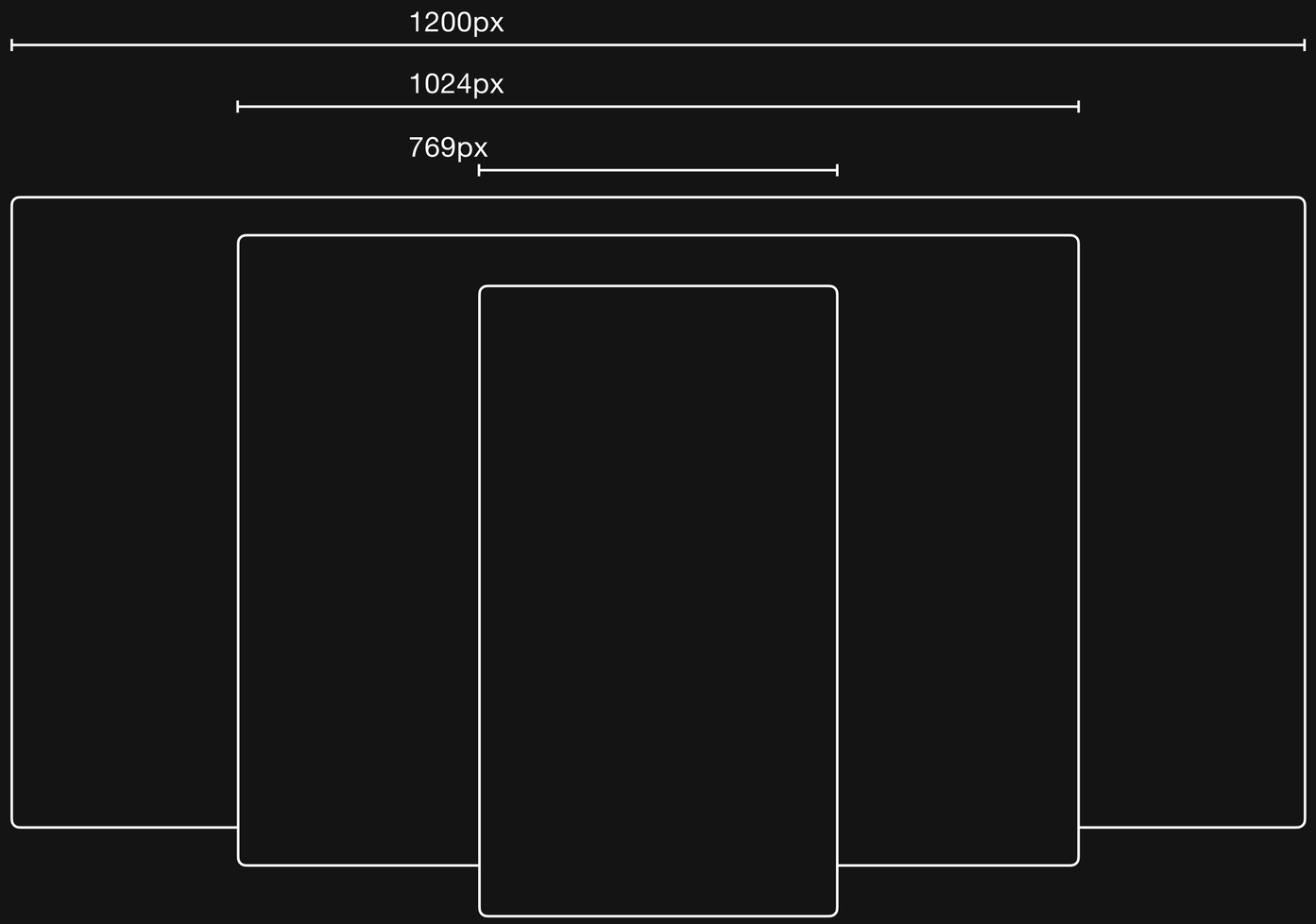
I’m curious ,when using the proximity principle in complex interfaces with overlapping elements (like dashboards or data-heavy UIs), how do you strike the right balance between grouping related items and avoiding visual clutter? Any tips or examples?

media queries
@containers
min and max values
clamps
etc

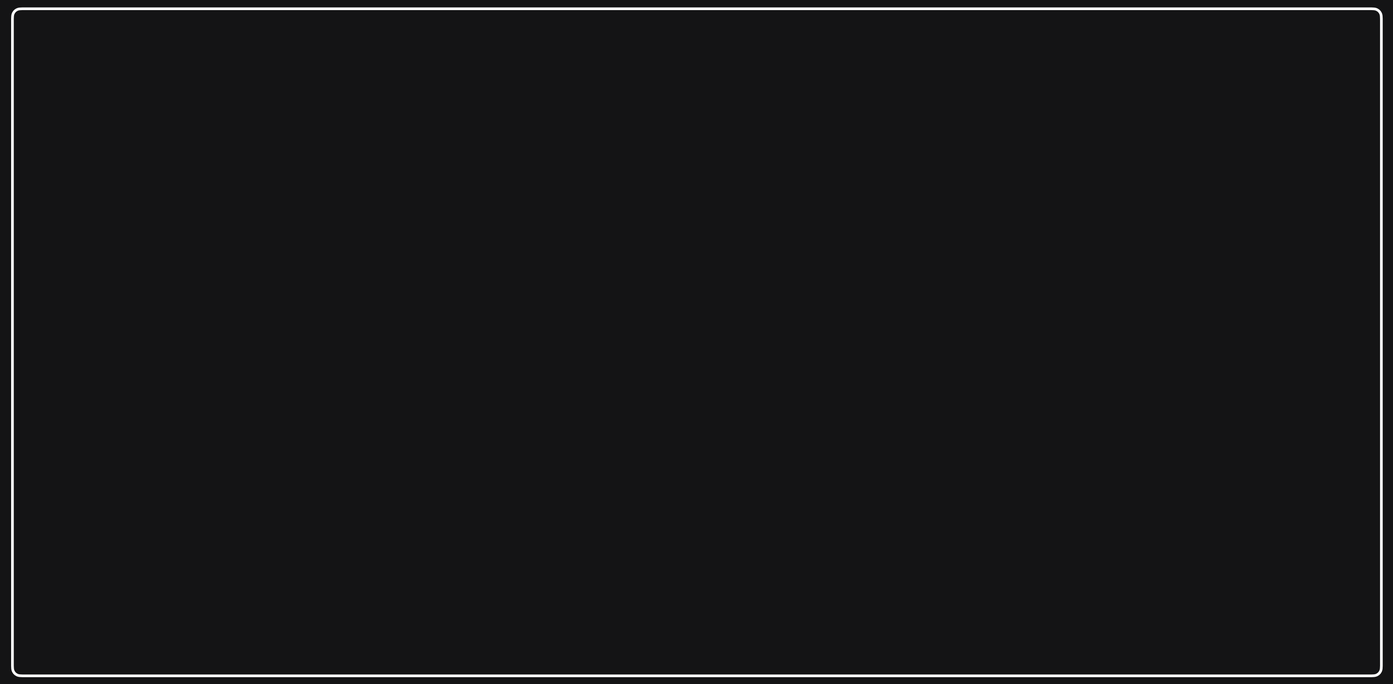
São formas de lidar com fluidez

media queries

Os dispositivos têm naturalmente width e height. São propriedades que o css reconhece juntamente com o browser e ajudam-nos a ajustar o conteúdo da forma mais adequada, para tal podemos usar as media queries para definir intervalos de desktop, tablet, mobile, etc e atuar sobre eles.



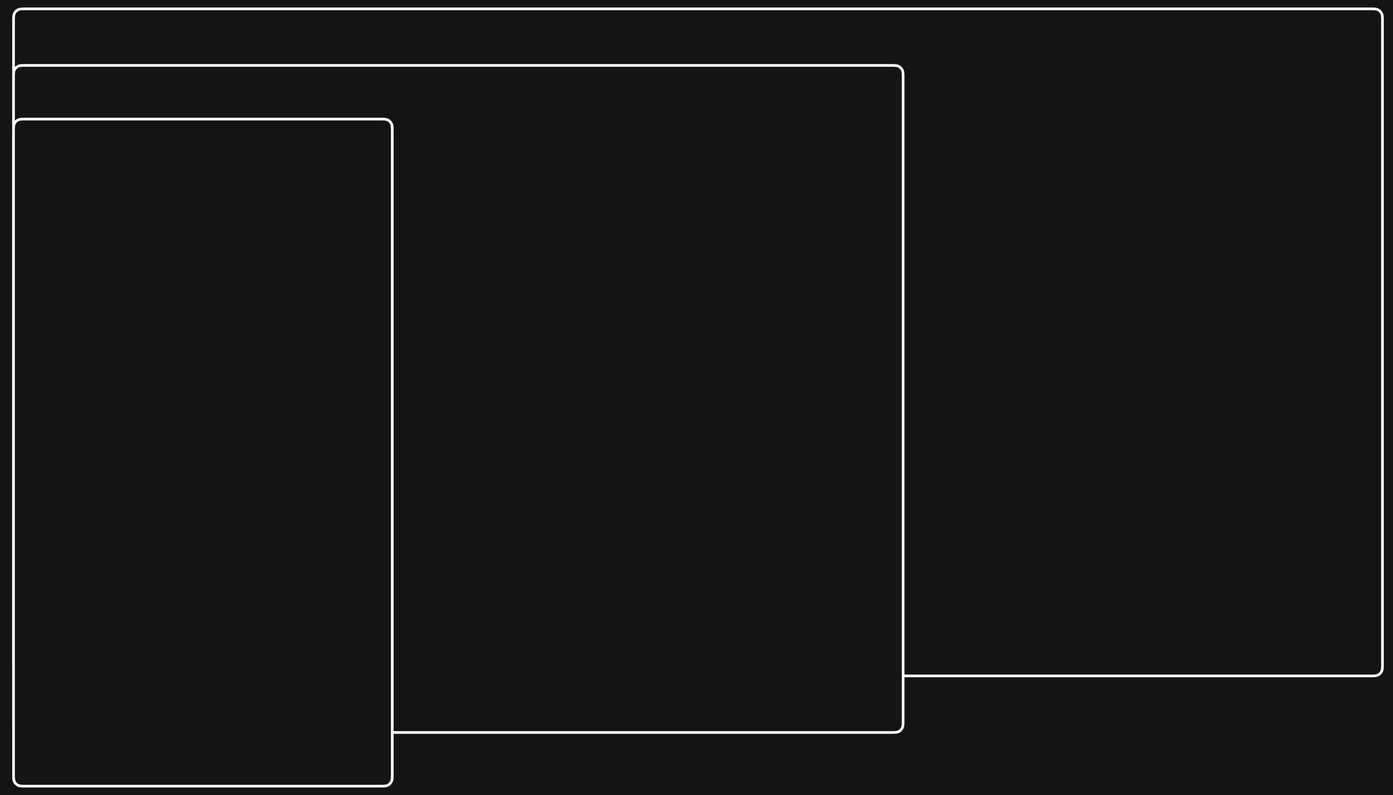
(min-width: 1200px) // acima de 1200 é desktop



(min-width: 1200px) // acima de 1200 é desktop

(min-width: 768px) // acima de 768 até 1220 é tablet

até 768px é mobile



Sintaxe

Media queries são condições que escrevemos normalmente no css. A sua função é definir se o argumento é verdadeiro, e se sim, executar um bloco de código em específico.

```
1  .item--list {
2       background: red;
3   }
```

Sintaxe

O bloco início <ln.1> é verdadeiro e executado até a condição da media query <ln.5> seja verdadeira.

Quando o viewport tiver 769px ou superior, o bloco interior será executado e posteriormente, o valor da .item—list será modificado deixando o background azul.

```
1  .item--list {
2       background: red;
3   }
4
5   @media all and (min-width: 769px) {
6     .item-list {
7       background: blue;
8     }
9   }
```

Sintaxe

O bloco início <ln.1> é verdadeiro e executado até a condição da media query <ln.5> seja verdadeira.

Quando o viewport tiver 769px ou superior, o bloco interior será executado e posteriormente, o valor da .item—list será modificado deixando o background azul.

Ao chegar aos 1200px, os dois blocos anterior serão redeclarados e o valor computador final será o da <ln.11> deixando o .item—list laranja.

```
1  .item--list {
2      background: red;
3  }
4
5  @media all and (min-width: 769px) {
6    .item-list {
7      background: blue;
8    }
9  }
10
11 @media all and (min-width: 1200px) {
12   .item-list {
13     background: orange;
14   }
15 }
```

Sintaxe

A ordem das media queries é importante, segue o princípio de cascata e de resolução de conflitos do CSS.

A última declaração em linha tem prioridade e define o argumento.

Neste caso, num viewport de 1300px,a media quey expectável é a da linha 5. No entanto, a media query da linha 11 aparece posteriormente, torna assim o elemento .item—list azul porque a condição é também verdadeira.

```
1  .item--list {
2      background: red;
3  }
4
5   @media all and (min-width: 1200px) {
6    .item-list {
7      background: orange;
8    }
9  }
10
11  @media all and (min-width: 769px) {
12    .item-list {
13      background: blue;
14    }
15  }
```

Sintaxe

Podemos também definir media queries por intervalos, ou seja, por exemplo: Entre 1000px e 1500px.

Para tal, usamos uma combinação de min-width e max-width para obter o intervalo.

Pode não ser a solução mais eficaz num contexto global de css complexo.

O principio geral adotado é de sempre mobile-first. Ou seja, um exemplo:

CSS Geral
- mobile

min-width: 768px // table

min-width: 1024px // desktop

min-width: 1200px // large screens.

```
@media all and (min-width: 1000px) and (max-width: 1500px) {
  .item-list {
     background: orange;
  }
}
```

@container queries

Começamos por definir, no parent, duas propriedades:
- container-type
- container-name

Esta propriedades vão marcar esta relação do parent para com outros elementos e atribuem um identificador.

```
.cards {
    container-type: inline-size;
    container-name: cards;
}

.card {
    ...
    width: 25%;
}
```

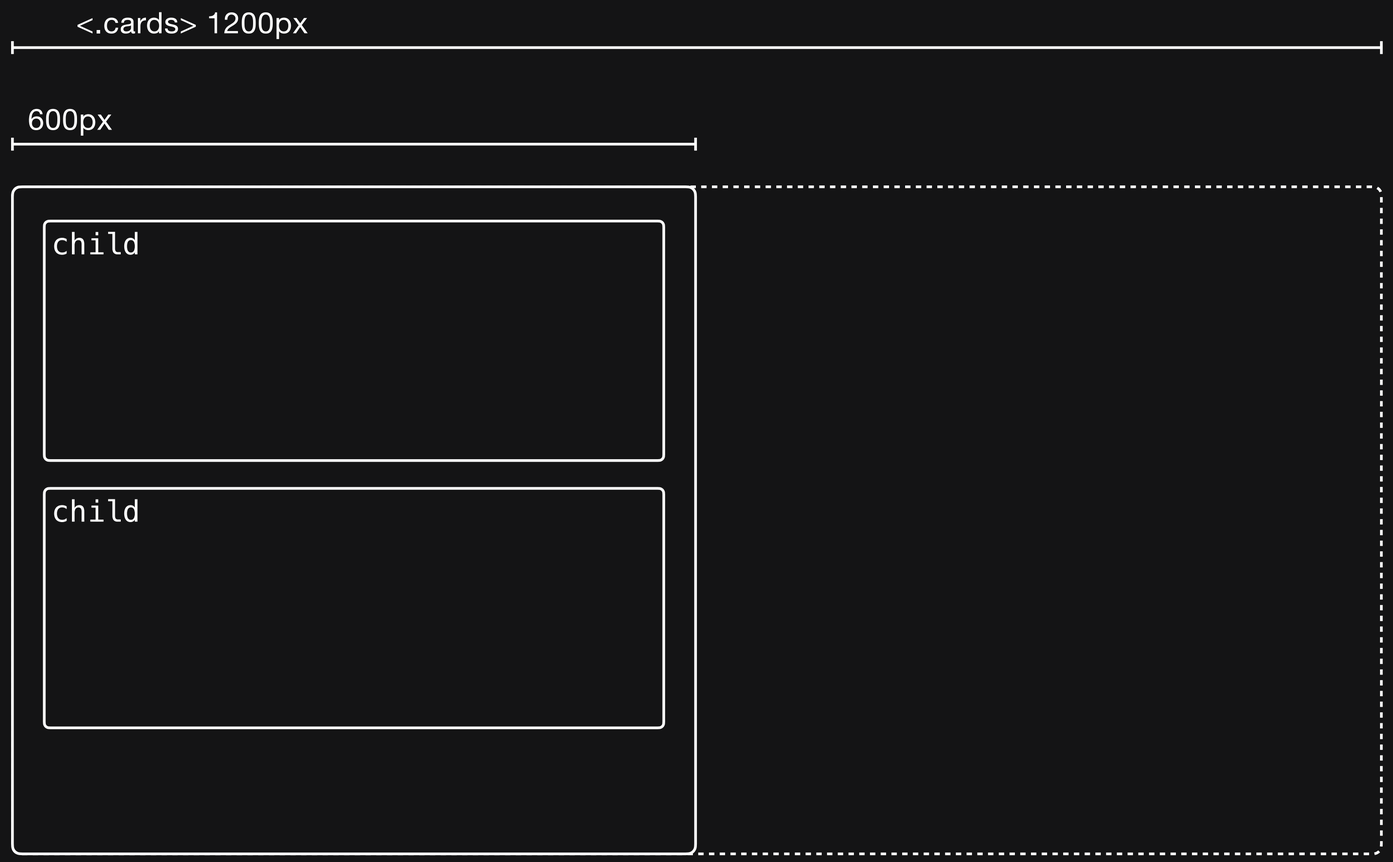
@container queries

Como o parent <.cards> tem o seu identificador e está marcado como container, podemos de seguida nos childs <.card> definir propriedades com base na largura do parent.

Se o parent tiver até 600px, os <.card> terão 100% de largura

```
.cards {
    container-type: inline-size;
    container-name: cards;
}

.card {
    ...
    width: 25%;
    @container (max-width: 600px) {
      width: 100%;
    }
}
```



min | max and clamp

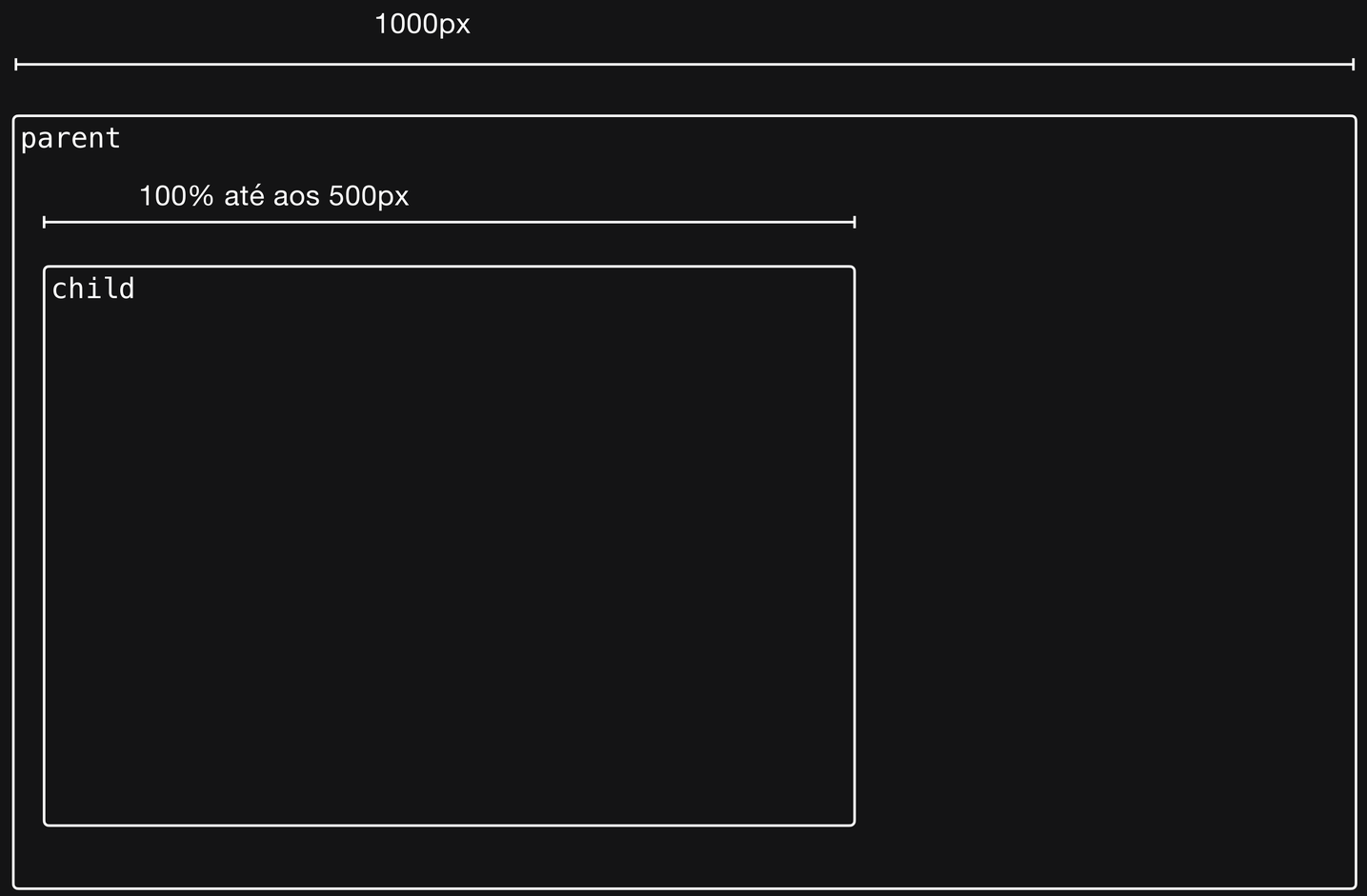
São propriedades e funções que nos ajudam definir parâmetros de fluidez.

Podemos definir que os nossos elementos aumentam ou diminuem até um determinado valor.

max-width

A propriedade max-width define que o element <child> irá ter uma width máxima de 500px.
Isto complementado com o width: 100% deixa o elemento fluído, sendo sempre 100% até ao máximo de 500px.

```
.child {
    ...
    width: 100%;
    max-width: 500px
}
```

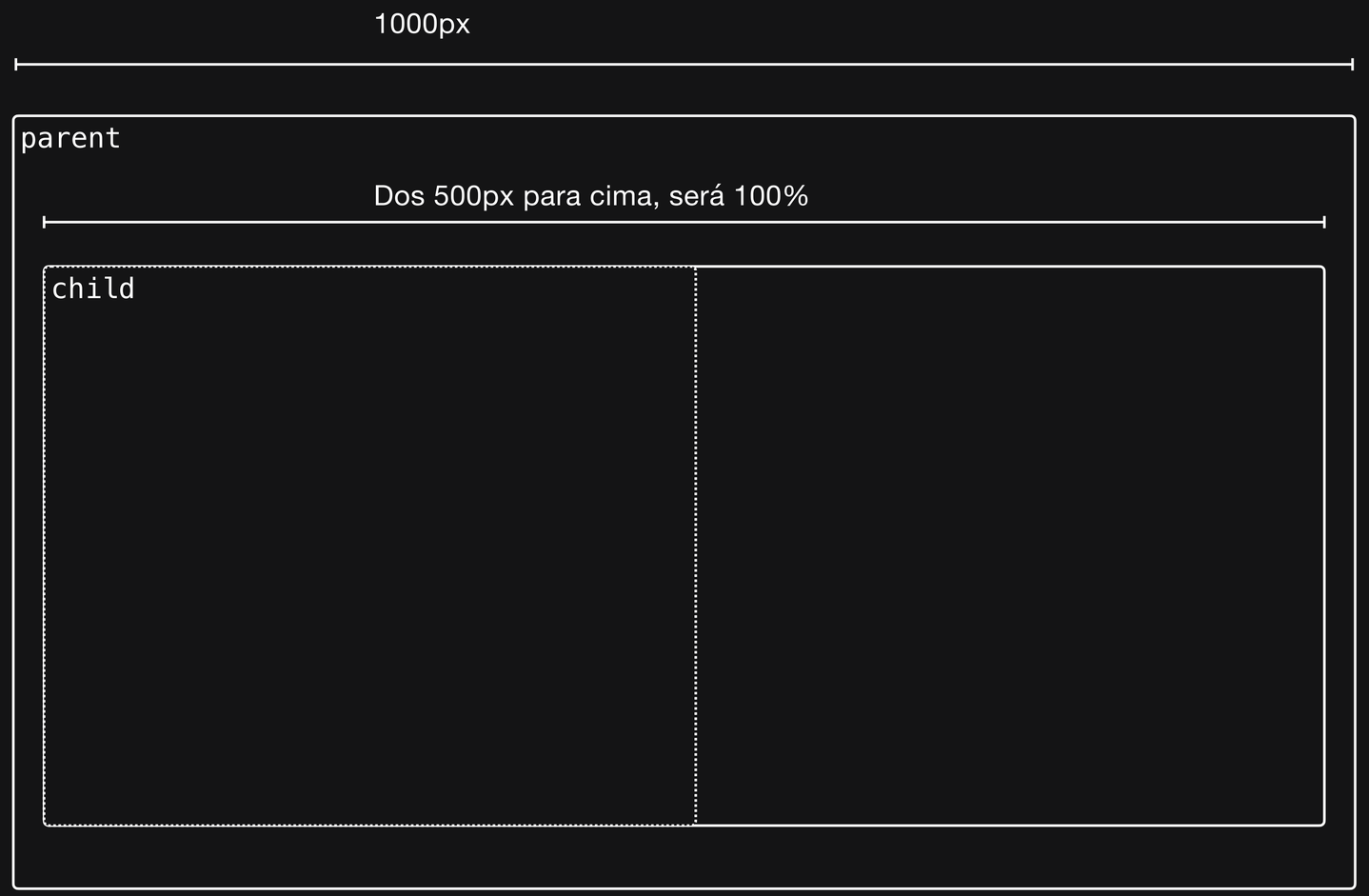


min-width

A propriedade min-width define que o element <child> irá ter no mínimo 500px;

Isto complementado com o width: 100% deixa o elemento com no mínimo 500px. Portanto entre 500px e n

```
.child {
    ...
    width: 100%;
    min-width: 500px
}
```



clamp()

O clamp() é uma função que complementa o min-width e o max-width.

Define que o child poderá ter entre (min, desejado, max).

Traduzido, o child terá entre 500px e 600px. Um valor entre estes dois parâmetros é 100%.

```
1  .child {
2      ...
3      width: clamp(500px, 100%, 600px)
4  }
```