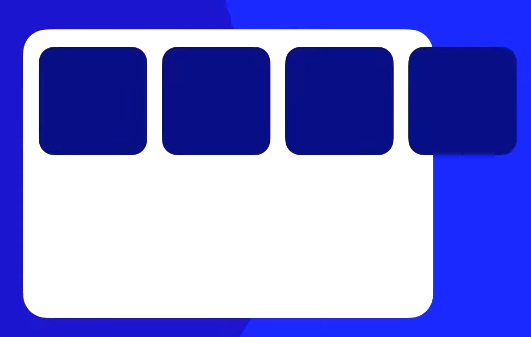
**MÓDULO 5 – FLEXBOX E GRID LAYOUT:**

**Flexible Box (2017): Qual o formato da água?** Ela ocupa o espaço que for disponível (adaptável) – depende do recipiente 🡪 **vai depender do container.**

Quando se cria um container display: block e itens filhos display: inline-block quando se altera o tamanho da tela os conteúdos (itens) podem acabar transbordando e saindo do container, pois não houve ajuste flexível.

**O Felxbox** veio justamente para corrigir esse transbordamento e deixar containers e conteúdos flexíveis e responsivos (para isso passar propriedade display: flex no container e nos intens flex: auto).

**Quem é o pai?** É o container, pai que vai levar a configuração **display: flex** o os itens são os flex-itens.

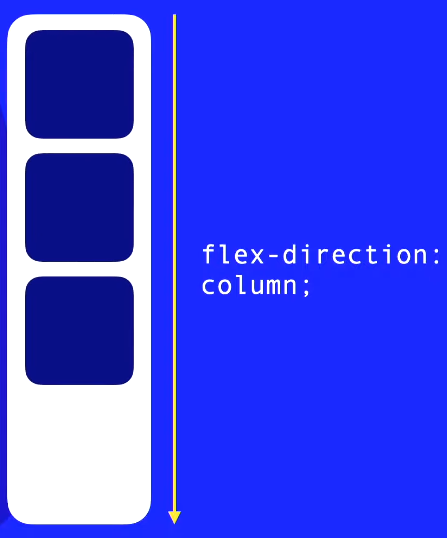
* Quando passamos container {display: flex} e item {flex: auto} ele vai preencher totalmente o container com os itens, ou seja, quantos itens existem? Esses itens que existem ocuparão tudo.

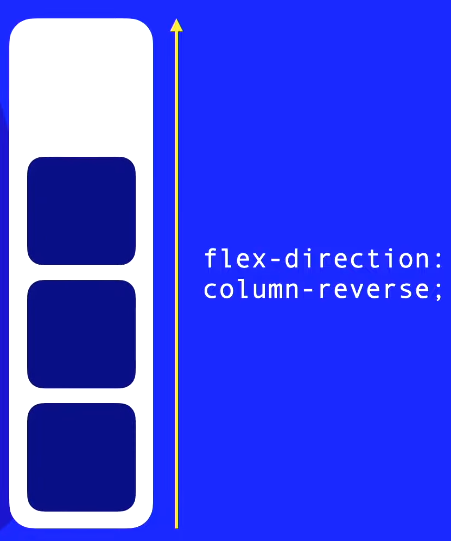
**DIREÇÕES E EIXOS:**

**Existem 4 flex-direction de configurações de container:**

1. **FLEX-DIRECTION: ROW:** Direção em linha, da esquerda para a direita e row-reverse começa da direita para esquerda

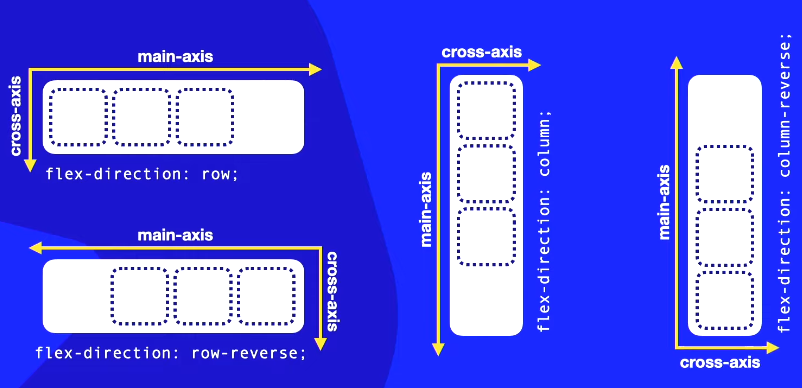
****

1. **FLEX-DIRECTION: COLUMN:** Direção em coluna, de cima para baixo



**EIXOS: Eixo principal** – é o eixo gerado igual ao eixo do flex-direction. Por exemplo se a propriedade for flex-direction: row, o eixo principal será o eixo “x” (linha). Também é gerado o **cross-axis** que é um eixo perpendicular ao eixo principal. Isso é o “padão” para configuração flex-direction: row com o nosso idioma (que é lido da esquerda para direita).

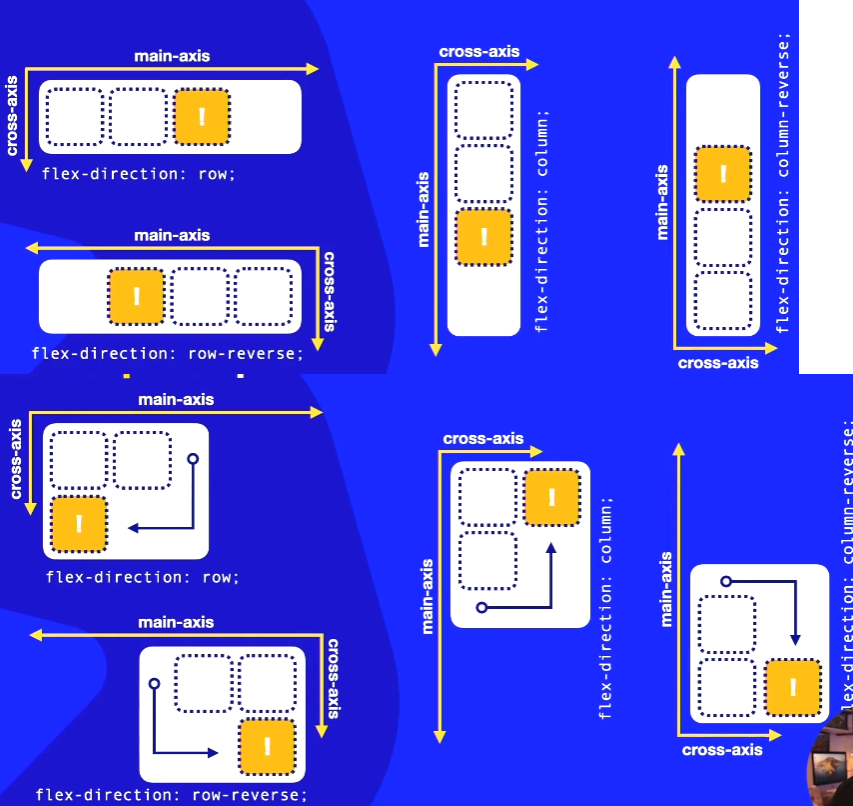




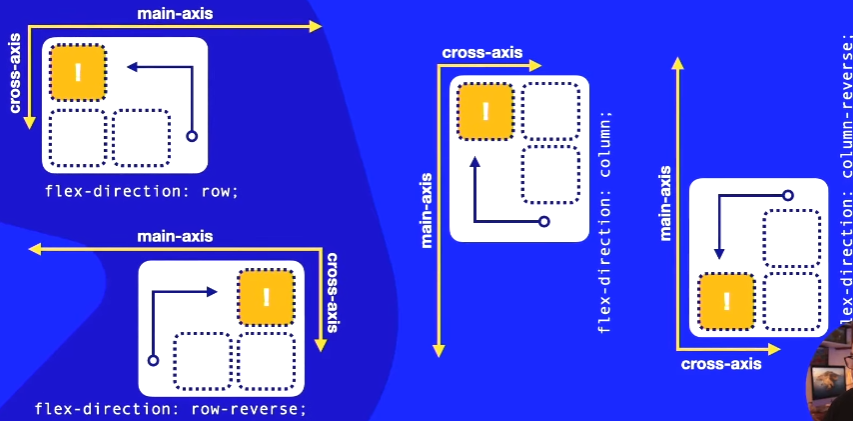
**FLEX-WRAP:** Empacotamento (encapsulamento) de itens dentro de um container. Essa propriedade irá determinar como a cápsula (container) irá se comportar. Por padrão o empacotamento vem como flex-wrap: nowrap. O empacotamento irá depender do contaúdo, por exemplo, se o conteúdo for um texto ele irá espremer até o limite da palavra mais longa. Os valores nowrap e wrap se sobra espaço no container ou o container está no limite do tamanho do conteúdo, **nada acontece,** ele começa a mudar quando o contêiner precisa ficar “menor” eu o conteúdo.

**Flex-wrap: wrap 🡪** Sempre fará a quebra no eixo cross-axis.

flex-wrap: nowrap;

**EFEITOS DE WRAP EM CADA CASO:**

**EFEITOS DE WRAP EM CASO DE WRAP-REVERSE:**

****

**\*SHORT-HAND (FLEX-FLOW = FLEX-DIRECTION + FLEX-WRAP):** Simplifica as duas declarações em apenas uma.

Flex-direction: row;

Flex-wrap: nowrap;

**Flex-flow: row nowrap;**

**FLEX-BOX: ALINHAMENTO NOS EIXOS: justyfi-content // align-items.** Da mesma maneira que o display: flex é passado no contêiner, as propriedades de alinhamento com flex também são passadas no container.

* **Justify-content para Direction ROW:** É o alinhamento **principal** (justify-content 🡪 main-axis) se for **row (referência é linha) se for column (referência é coluna)**

**1.justify-content: flex-start** = row main-axis é linha e flex-start é grudado na esquerda. (sobra espaço livre na direita)

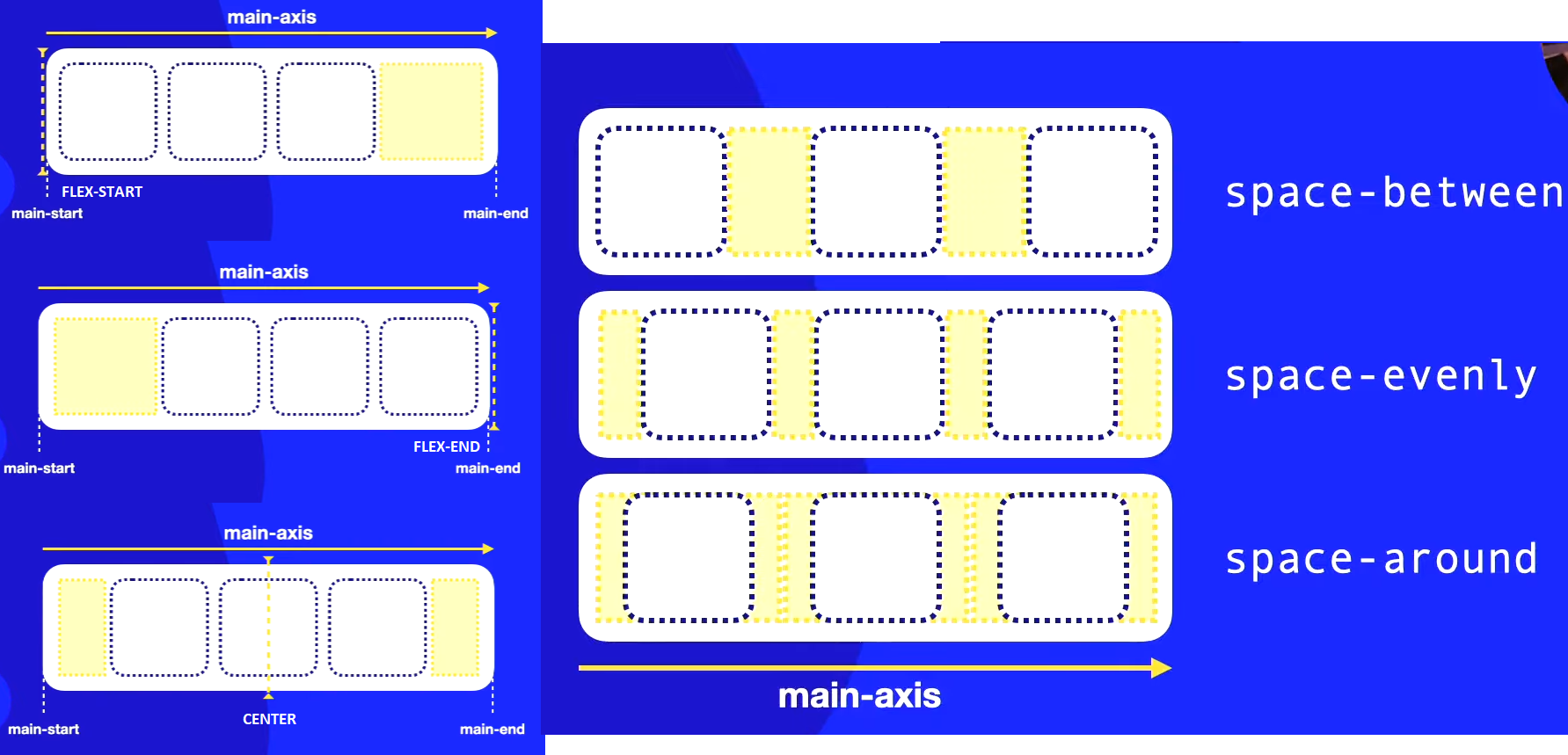
**2. justify-content: flex-end** = row main-axis é linha e flex-end é grudado na direita. (sobra espaço livre na esquerda).

**3.** **justify-content: center =** row main-axis é linha, então flex-center é grudado no meio. (sobra para os dois lados).

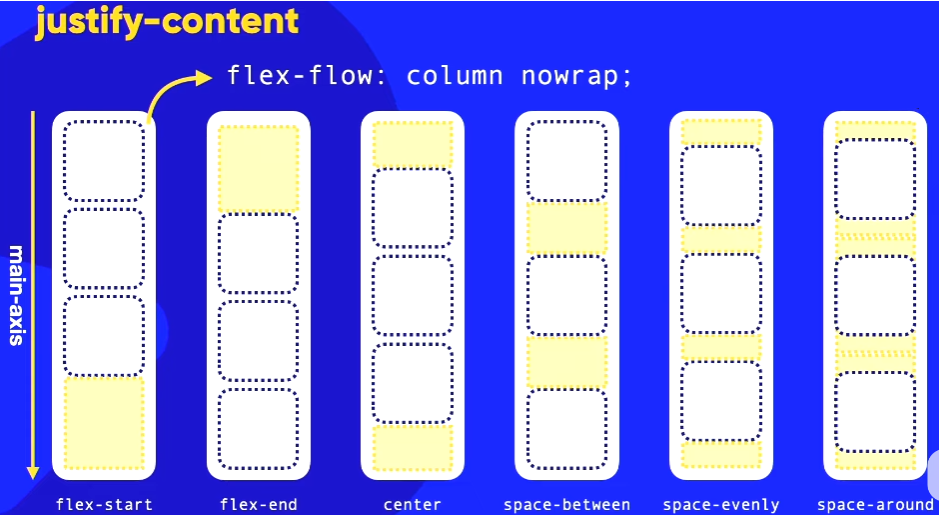
**4. justify-content: space-btween =** row o main-axis é linha, então gruda nos dois lados e deixa um espaço padrão entre os itens. (1° item no flex-start e último item no flex-end e o restante vai ser colocado no meio com espaçamento igual entre eles.)

**5. justify-content: space-evenly =** row o main-axis é linha, ele coloca espaçamento igual **antes** e **depois** de todos os itens. (espaçamento antes e depois é tudo igual).

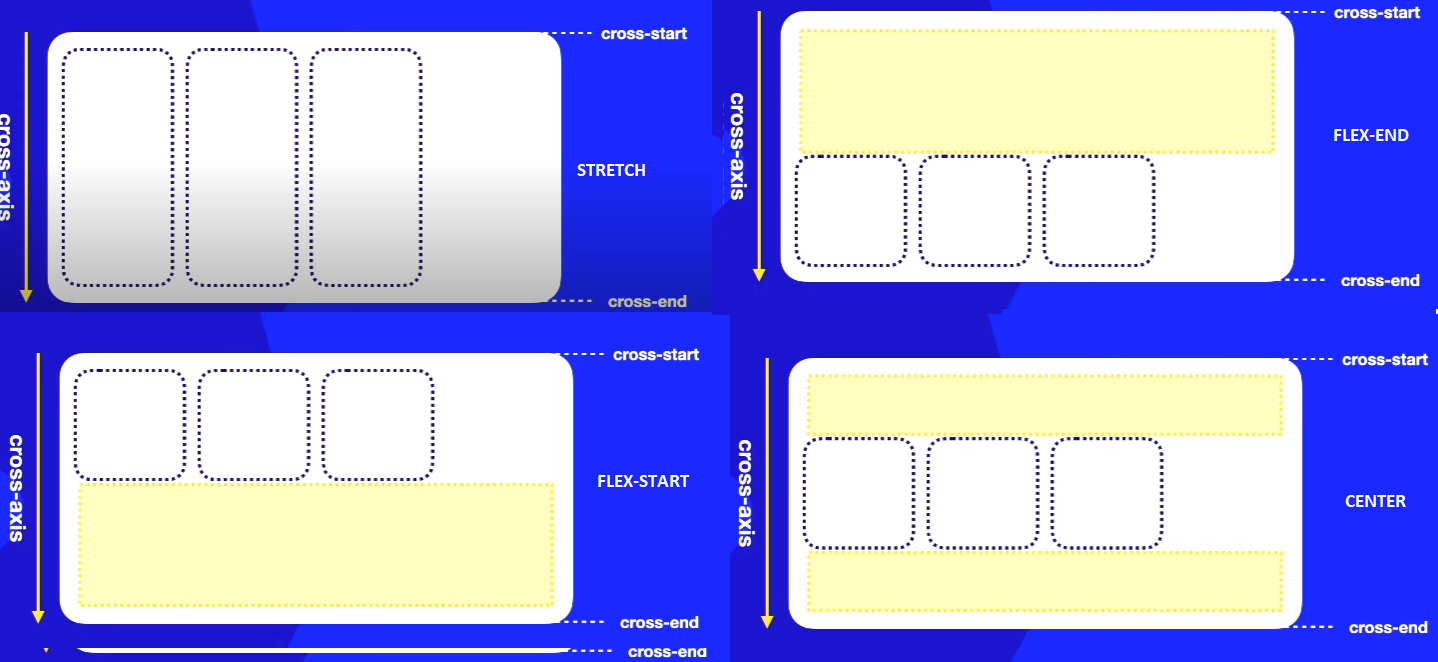
**6. justify-content: space-arround=** row o main-axis é linha, então ele divide o container na quantidade de itens e coloca o idem dentro dessa sobra. (sobra do “space-around” espaço ao redor é somada).



* **Justify-content para Direction COLUMN:** É o alinhamento principal (justify-contet 🡪 main-axis) se for **row (referência é linha) se for column (referência é coluna).**

1. **Justify-content: flex-start =** column é coluna então main-axis é vertical, então flex-start é topo. (sobra espaço livre em baixo).
2. **Justify-content: flex-end =** column é coluna então main-axis é vertical, então flex-end é em baixo. (sobra espaço livre em cima).
3. **Justify-content: center: =** column é coluna e main-axis é vertical, então ele centraliza o conteúdo. (sobra espaço igual em cima e em baixo).
4. **Justify-content: space-btween**  = column é coluna e main-axis é coluna, então coloca o 1° e último item grudados, respectivamente no topo e em baixo e preenche com o restante no meio.
5. **Justify-content: space-evenly** = column é coluna e o main-axis é vertical, então ele coloca espaços iguas entre os itens.
6. **Justify-content: space-arround =** column é coluna e o main-axis é vertical, então ele soma os espaços anteriores e posteriores, o 1° e últimos item não possuem espaços anteriores e posteriores, respectivamente. (sobra é a soma do espaço ao redor, ou seja, “space-around”).

* **Align-items:** é o posicionamento não principal e é relativo ao cross-axis (eixo perpendicular). Se vier no tipo row (linha) o cross-axis é o eixo vertical. Ele vem por padrão com o valor **stretch** (estica o conteúdo junto com o esticamento do cross-exis).

1. **Align-items: flex-start =** sendo fllex-direction: row o main-axis é linha então o cross-axis é vertical, portante o flex-start é no topo do cross-axis. (sobra espaço em baixo).
2. **Align-itens: flex-end =** sendo o flex-direction do tipo row o main-exis é linha, portanto o cross-axis é vertical, portanto o flex-start vai ser em baixo. (sobra espaço no topo).
3. **Align-itens: center =** se o flex-direction for row, o main-axis é linha e o cross-axis é vertical, portanto o alinhamento é no centro da coluna. (Sobra espaço igual no topo e na base).

**Modulo flex-box:** Para centralizar elementos e independente do tamanho da tela e do formato do container, o elemento sempre ficará no meio (centralizado). Para centralizar um item dentro de um container pai, independente do flex-direction (row ou column), passando as prorpriedades justify-conter: para o main-axie e a pripriedade align-itens: center o item ficará centralizado no container independente do tamanho e formato do container.

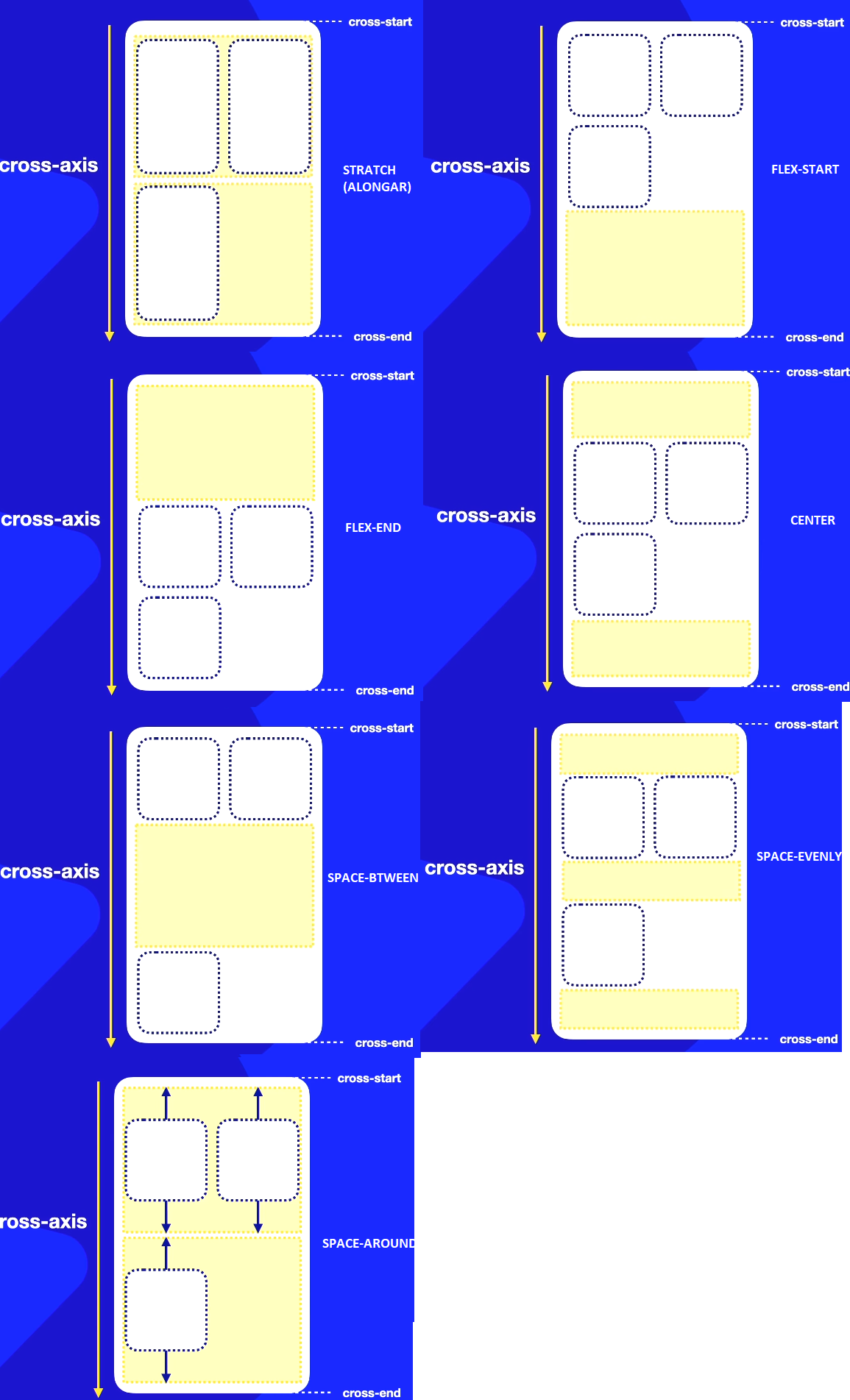
Para fazer a centralização de um item com flexbox: sempre se pergunte **QUEM É O PAI? QUEM É O FILHO (A)?**

**CONTEÚDOS EMPACOTADOS:** Alinhamento de conteúdos empacotados. Basicamente quando diminuímos o tamanho de um container também precisamos decidir **o que irá acontecer** com os itens que estão dentro desse container.

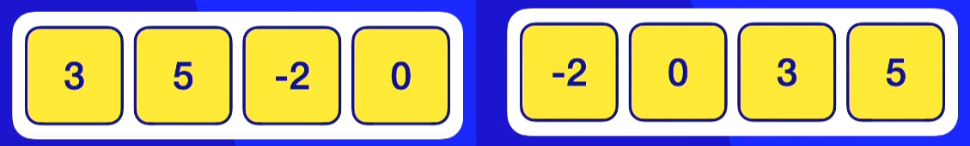
* **Justify-content:** Alinha o conteúdo dos itens de acordo com o eixo principal (main-axis).
* **Align-itens:** Alinha o conteúdo dos itens de acordo com o eixo transversal (cross-axis).
* **Align-content:** Alinha os elementos no eixo transversal quando eles estão empacotados.

**ALIGN-CONTENT: Propriedade que só funciona quando uso wrap.**Alinha os conteúdos de acordo com o eixo transversal (cross-axis) quando os elementos estão empacotados (wrap). O align-content por padrão vem com o valor stratch (alongar).

* **Align-content: stratch =** Divide o conteúdo em espaços iguais de acordo com a quantidade de linhas ou colunas e alonga o conteúdo para caber nesses espaços iguais.
* **Align-content: flex-start =** Coloca os conteúdos de acordo com o cross-axis com sobra de espaço para baixo.



**PROPRIEDADES DE ITENS (EM CONTAINERS):** Todo filho dentro de um container flex terá o **order: 0** (que é a ordem de exibição dos elementos). Esse tipo de propriedade vai ser comum quando precisarmos alterar a ordem dos itens, por exemplo, em menus.

O próprio CSS irá fazer esse ajuste para deixar na ordem crescente.