Flexbox: The Space Deal

A Detailed Study Guide to `flex-grow`, `flex-shrink`, and `flex-basis`.

Terms of the Space Deal

Once a container knows its direction and axes, flex items start negotiating for space. Three properties run the table, deciding who stretches, who holds the line, and who quietly adapts. Everything happens along the **main axis**.

`` `flex-grow` ``

How strongly an item asks for *more* space.

`` `flex-shrink` ``

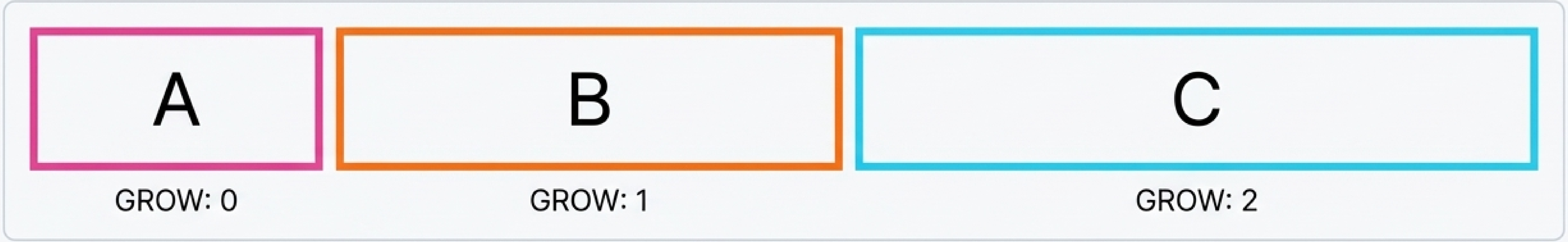
How willing an item is to be *squished*.

`` `flex-basis` ``

The size an item wants to *start* at.

`flex-grow`: Who Claims the Leftover Space

When there is extra space on the main axis, `flex-grow` decides who claims it. All `grow` values are compared as weights. A higher value gets a bigger slice of the extra space.



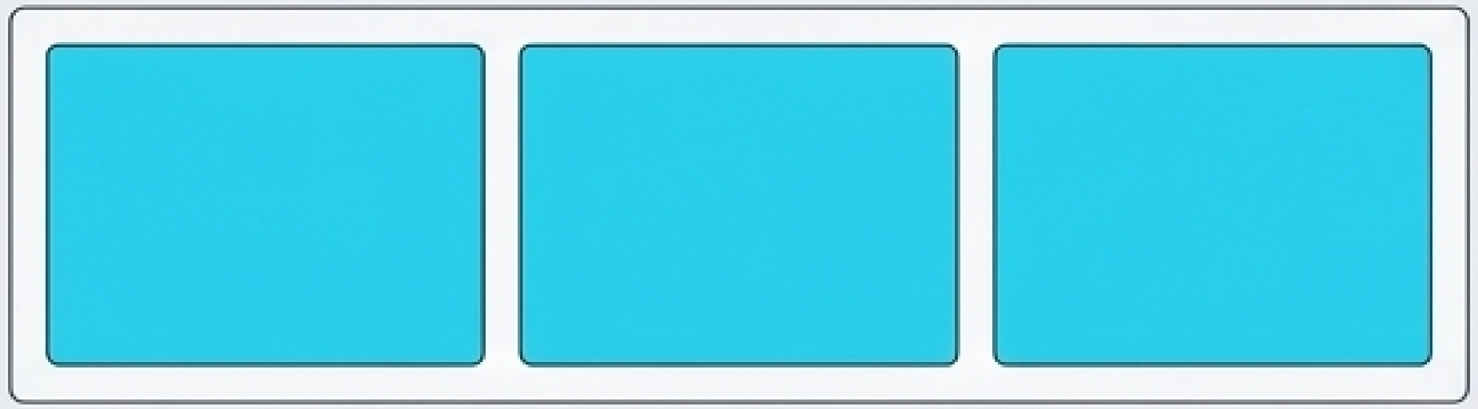
```
/* All slots start with the same basis and shrink values */
.item { flex-basis: 7rem; flex-shrink: 1; }

.item-a { flex-grow: 0; } /* Takes no extra space */
.item-b { flex-grow: 1; } /* Takes one share */
.item-c { flex-grow: 2; } /* Takes two shares */
```

`flex-grow` in Action: Common Patterns

`flex-grow` only matters when there is slack to distribute.
The distribution is always proportional to the grow values.

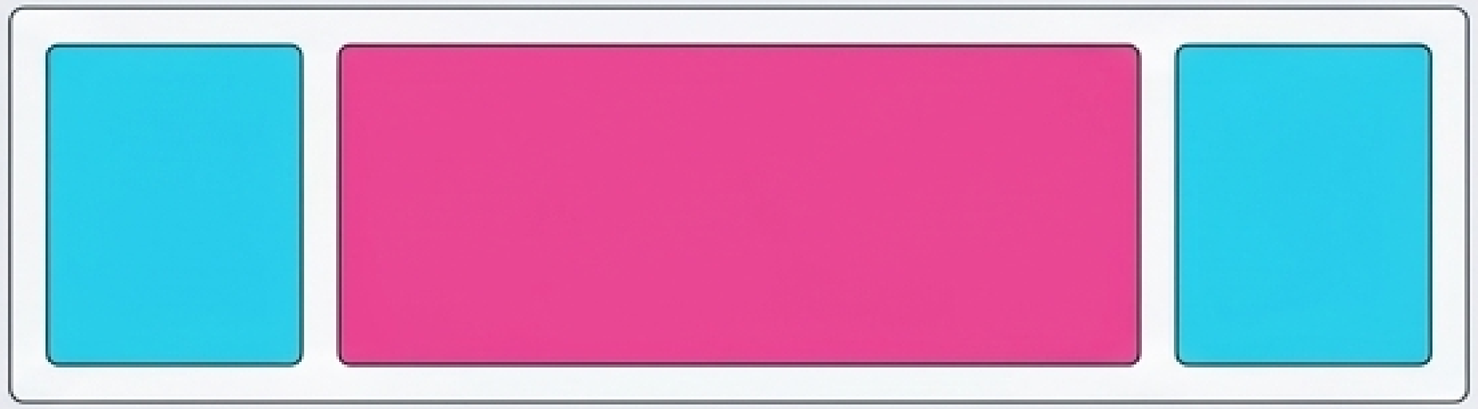
Pattern 1: Equal Partners



```
.item { flex-grow: 1; } /* All items grow equally */
```

With `flex-grow: 1` on all items, they share the extra space evenly.

Pattern 2: The Hero Item

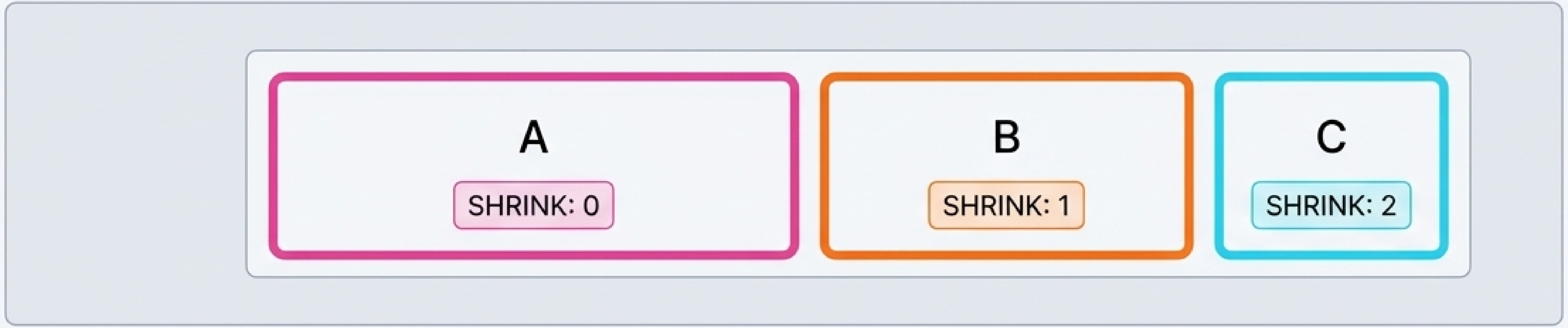


```
.item--hero { flex-grow: 3; } /* Middle item grows 3x faster */
.item       { flex-grow: 1; }
```

The middle item takes three parts of the extra space for every one part the others get.

`flex-shrink`: Who Gives Up Space First

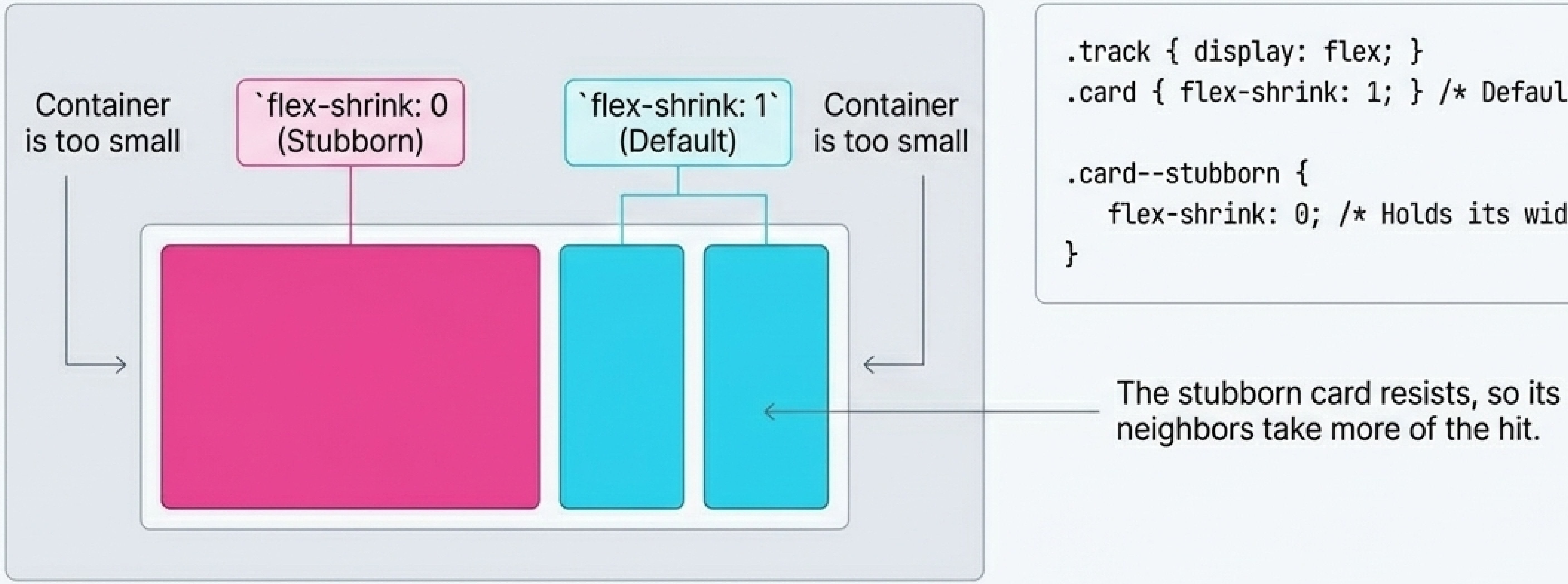
When the track is narrower than the sum of all bases, `flex-shrink` controls who slims down. A higher value means an item is more willing to be squished.



```css
/* All slots start with the same basis and grow values */
.item { flex-basis: 8rem; flex-grow: 0; }
.item-a { flex-shrink: 0; } /* Resists shrinking */
.item-b { flex-shrink: 1; } /* Shrinks at a normal rate */
.item-c { flex-shrink: 2; } /* Shrinks twice as fast */
```

`flex-shrink: 0` — The Stubborn Item

The default `flex-shrink` is `1`, meaning items can be squished. Setting `flex-shrink: 0` tells an item to hold its `flex-basis` (or `width) as long as possible. A shrink value of zero means "do not shrink me."

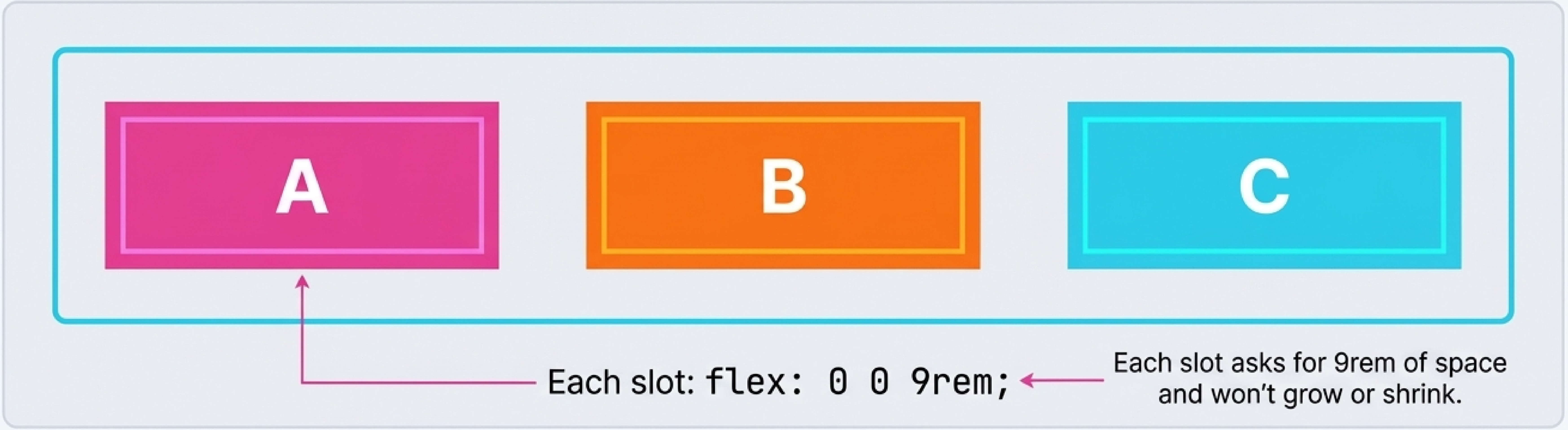

Container is too small
`flex-shrink: 0 (Stubborn)
`flex-shrink: 1` (Default)
Container is too small
The stubborn card resists, so its neighbors take more of the hit.

```
.track { display: flex; }
.card { flex-shrink: 1; } /* Default: can shrink */

.card--stubborn {
   flex-shrink: 0; /* Holds its width */
}
```

`flex-basis`: The Starting Point

`flex-basis` sets an item's starting size along the main axis, *before* grow or shrink negotiations adjust it. It is the preferred size an item asks for.



Each slot: `flex: 0 0 9rem;`

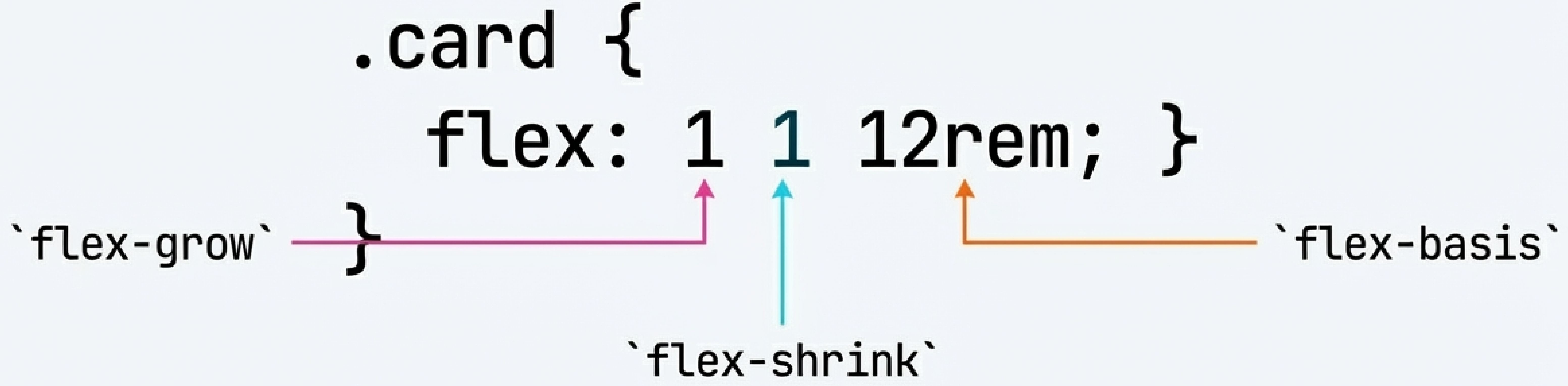
Each slot asks for 9rem of space and won't grow or shrink.

Basis vs. Width/Height

- If `flex-basis` is set, it usually overrides `width` or `height` along the main axis.
- In a `flex-direction: row`,`flex-basis` acts like a starting `width`.
- In a `flex-direction: column`, `flex-basis` acts like a starting `height`.

The `flex` Shorthand: The Master Contract

In production, we rarely write the three properties separately. The `flex` shorthand combines them into a single, concise declaration.



```
.card {
  flex: 1 1 12rem; }
}
```

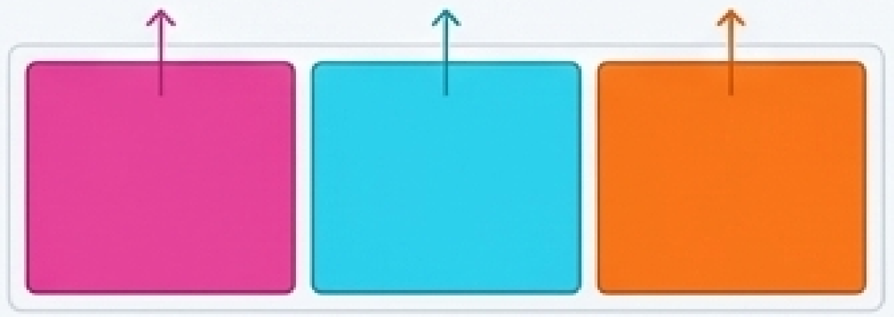
Mastering this order (`grow`, `shrink`, `basis`) is essential for efficient and readable Flexbox code.

The Flexbox Codex: Common Recipes

Equal Partners

```
.item {
  flex: 1;
}
```

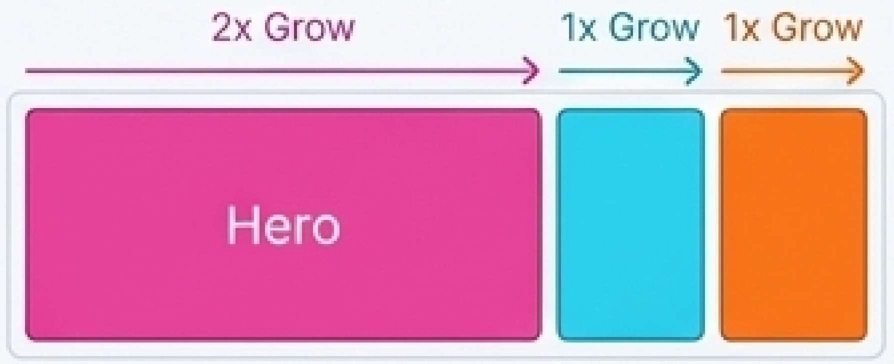
Shorthand for `flex: 1 1 0`. Items start at size 0 and grow to fill the container equally. The classic "equal columns" pattern.



Hero + Supporting Items

```
.hero {
  flex: 2 1 12rem;
}
```

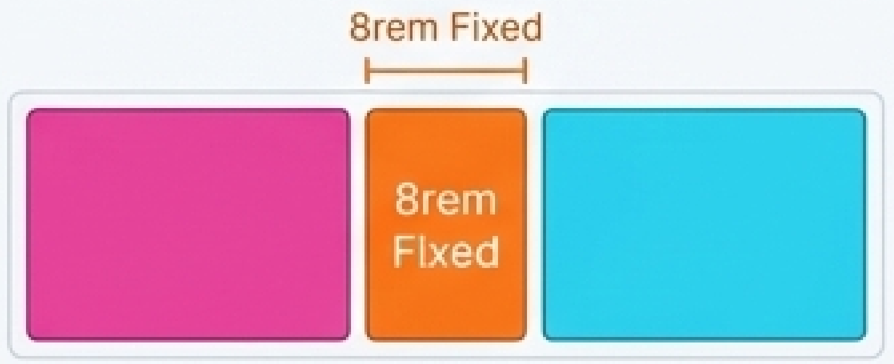
Starts wider (`12rem`) and grows twice as fast as neighbors. A dominant, flexible item.



Fixed-Size Item

```
.fixed {
  flex: 0 0 8rem;
}
```

No grow, no shrink, fixed basis of `8rem`. Creates a rigid, inflexible item that holds its size.



The Rules of Thumb

The entire negotiation can be simplified to these three rules.

`` `grow` = extra space ``

"When the container is bigger than the items, `flex-grow` distributes the surplus."

`` `shrink` = not enough space ``

"When the items are bigger than the container, `flex-shrink` decides who concedes."

`` `basis` = starting point ``

"This is the initial size request, before any growing or shrinking occurs."

Common Pitfalls & Traps

Forgetting Basis Defaults

`flex: 1` is shorthand for `flex: 1 1 0`. The `0` basis is often what you want for equal distribution, but forgetting it can lead to confusion if items have intrinsic sizes.

Confusing Grow with Percentages

`flex-grow` distributes *available* space, not total container space. An item with `flex-grow: 2` is not “twice the width” of an item with `flex-grow: 1`—it just gets twice the share of the *leftover* room.

Expecting Grow/Shrink to Work in a Void

`flex-grow` has no effect if there is no extra space. `flex-shrink` has no effect if there’s plenty of space. The properties only activate when the conditions are right.