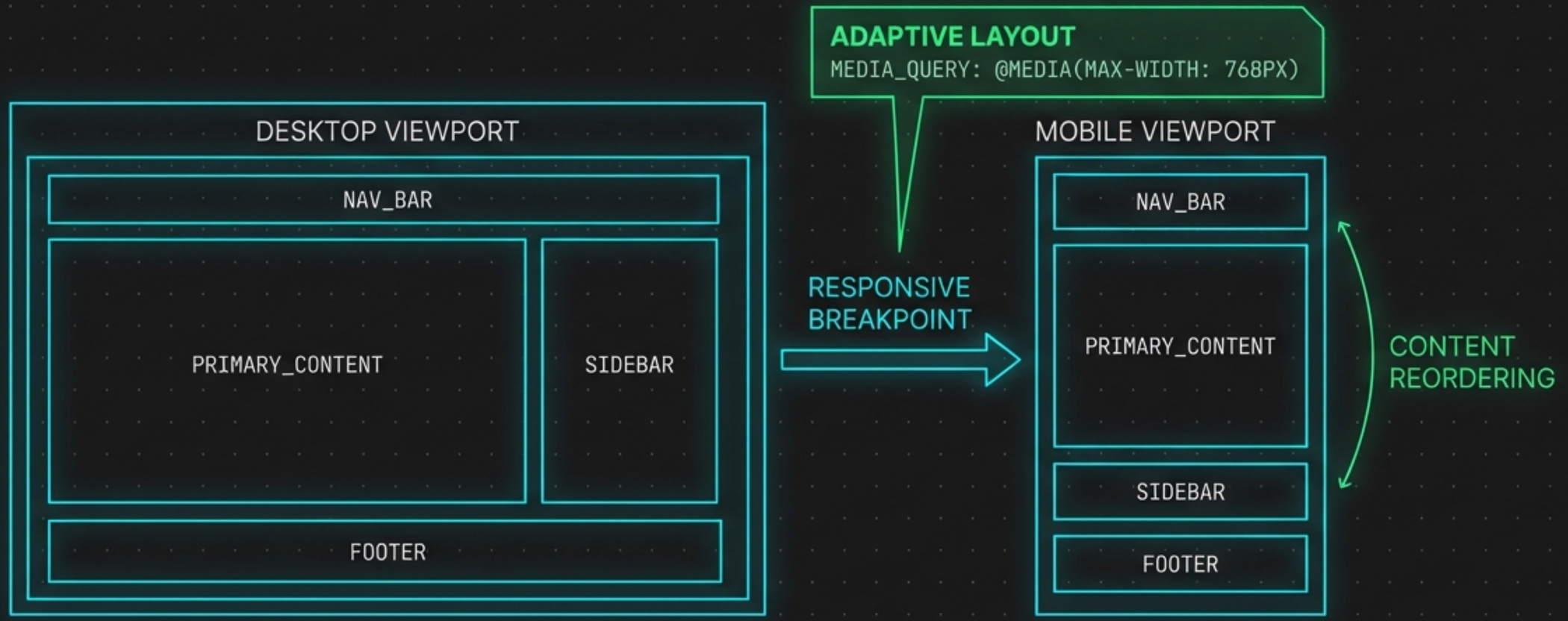
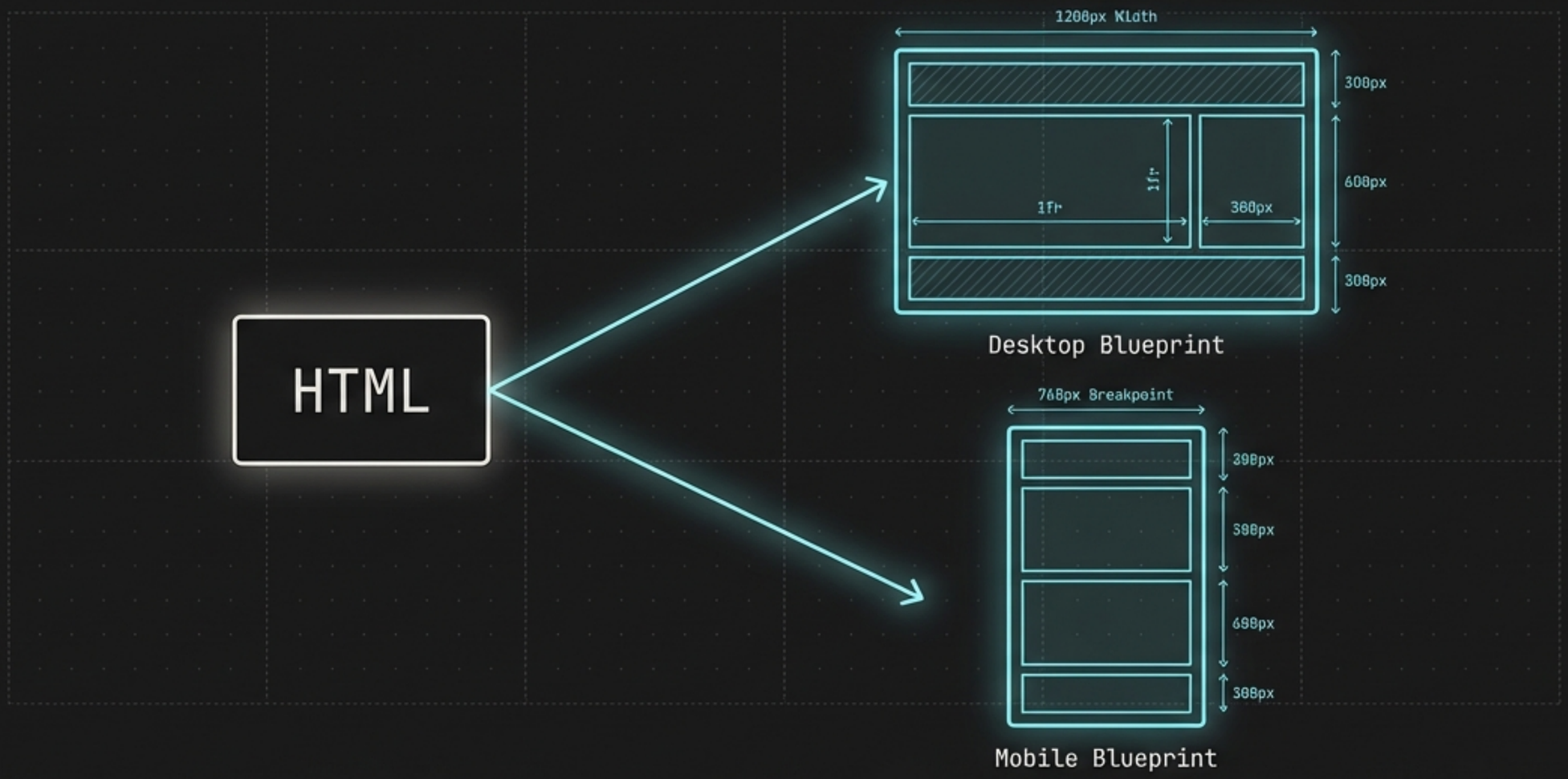
Responsive Areas



Same HTML. New blueprint.

You can rebuild the entire page layout at different breakpoints without touching your HTML. This is where CSS Grid becomes unfair.



The Two-Step Pattern

1. Define the desktop blueprint.

   Establish your primary layout with `grid-template-areas`.

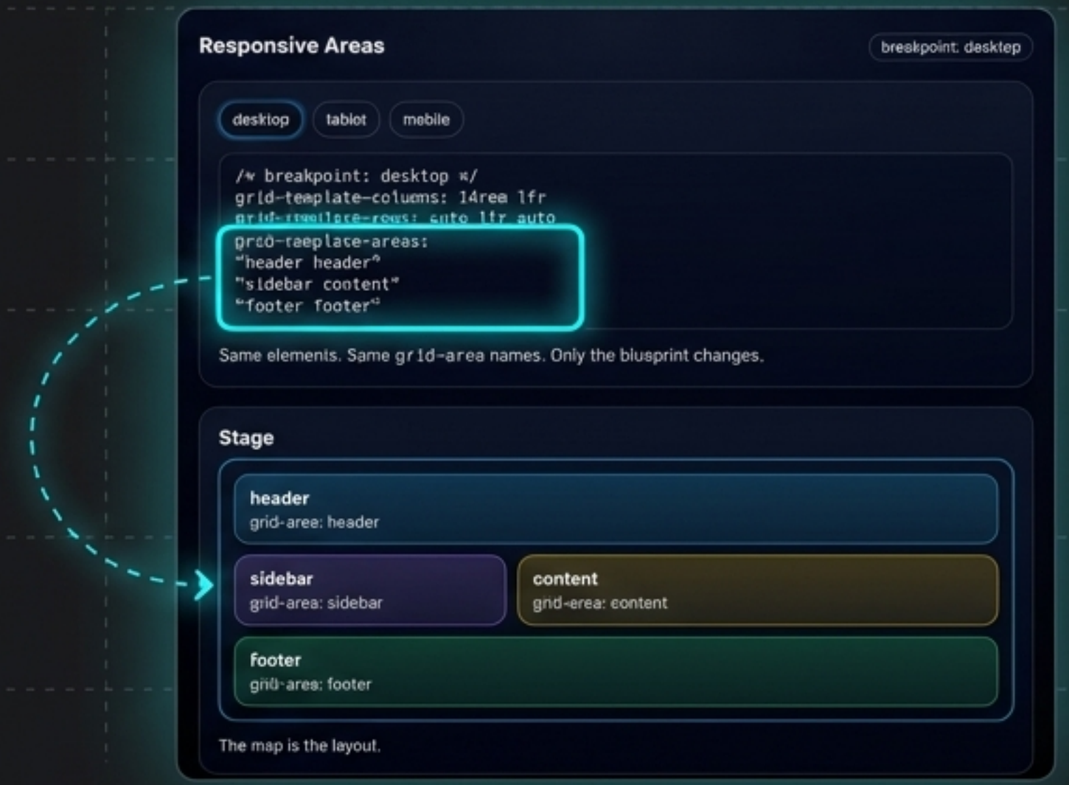
2. Override the blueprint in a media query.

   Inside a breakpoint, redefine the `grid-template-areas` “map”.

The secret sauce: same elements, same `grid-area` names, different map.

Example: The Desktop Blueprint

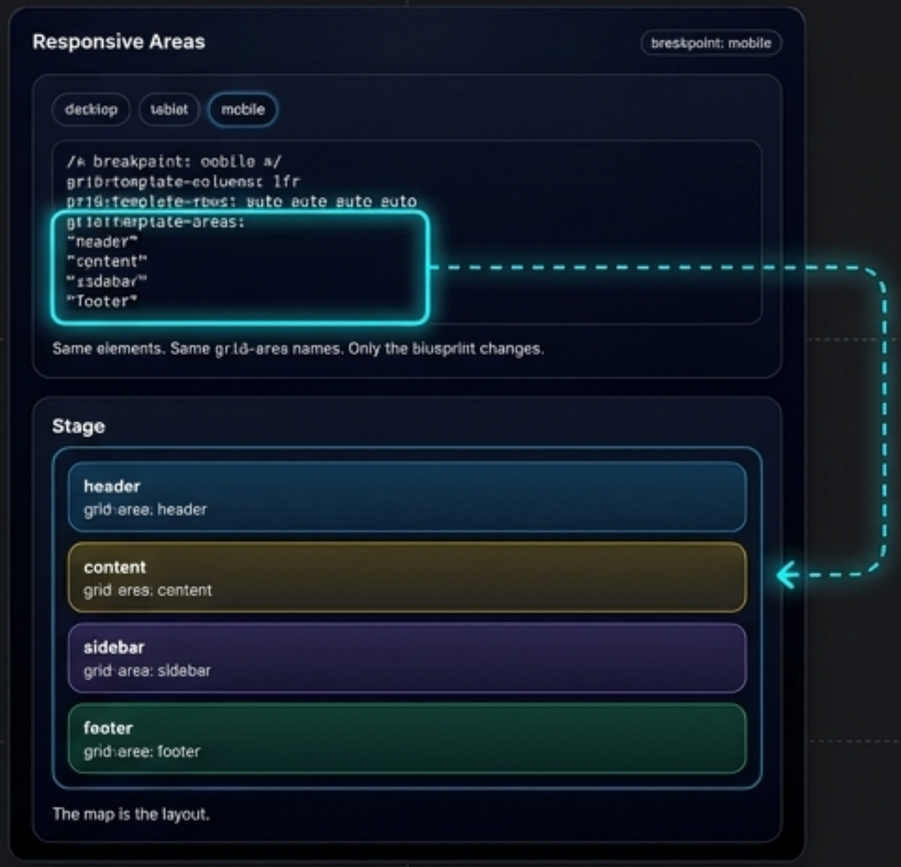
The `grid-template-areas` property defines the visual map.



The desktop layout uses two columns, placing the `sidebar` next to the `content`.

Example: The Mobile Blueprint

At a smaller viewport, a media query provides a new, single-column map.



The `sidebar` is now re-positioned below the `content`. The HTML source order is unchanged.

Anatomy of the Refactor

WHAT CHANGES

- `grid-template-columns`: The number and size of the grid's columns.
- `grid-template-areas`: The visual map that arranges the named areas.

WHAT REMAINS THE SAME

- HTML Source Order: Content stays semantic and logical.
- `grid-area` names: The labels connecting HTML elements to the grid map.

Why This Beats DOM Reordering

This CSS-first approach is fundamentally more robust than legacy methods. Layout becomes a CSS-only concern.

Your HTML stays semantic.

The document's structure remains logical, independent of its visual presentation.

Accessibility stays sane.

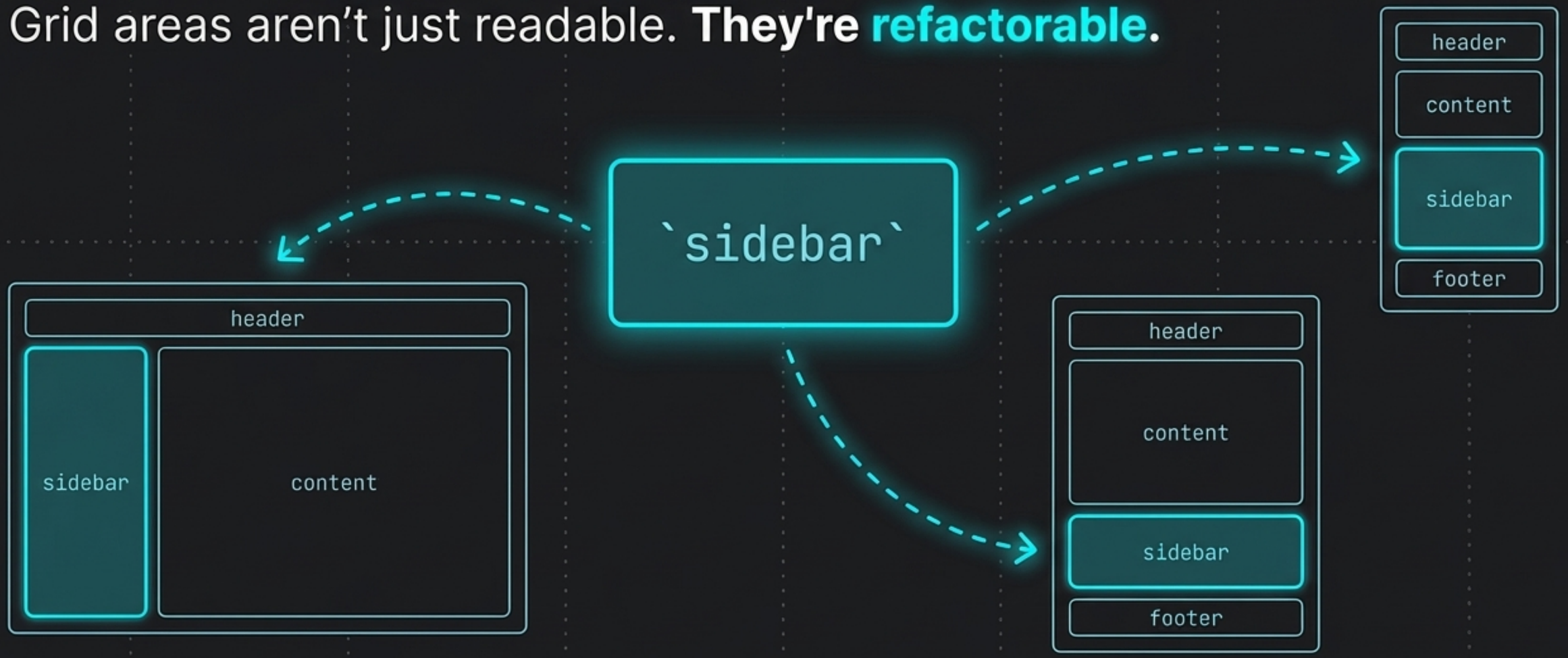
Screen readers follow the logical HTML order, providing a coherent user experience.

CSS owns layout.

True separation of concerns is respected. Layout is managed exclusively in the stylesheet.

From Layout to Layout System

Grid areas aren’t just readable. **They're refactorable.**



“CSS changes the blueprint. HTML stays semantic.”

p.s., keep learning!