LECTURE 10

Responsive Design and Frameworks

Building one site that adapts to any screen — and the two major CSS frameworks that make it faster.

CSC336 Web Technologies

OVERVIEW

In This Lecture

1. Mobile-first design, the viewport meta tag, and fluid units
2. Media queries and breakpoints — every combination, plus responsive images
3. Component-based (Bootstrap) vs. utility-first (Tailwind) frameworks
4. Bootstrap's grid system in depth: auto-layout, nesting, offsets, ordering
5. Tailwind's utility classes, config file, and dark mode
6. Extending either framework with your own custom classes
7. CSS preprocessors: Sass, SCSS, and LESS — and customizing Bootstrap with Sass

Mobile-First Design

1. Write base CSS for small screens FIRST, then add styling for larger screens with media queries
2. The opposite of "desktop-first," where you design big and cram it onto a phone afterward
3. Mobile traffic is the majority of web visits — adding complexity as space grows is easier than stripping it away

The Viewport Meta Tag

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

width=device-width uses the real device width; initial-scale=1.0 starts at 100% zoom.

Never Forget This Tag

WARNING

Without the viewport meta tag, none of your media queries will work correctly on a real phone — the browser will still pretend it has a much wider screen.

Units That Scale, Not Fixed Pixels

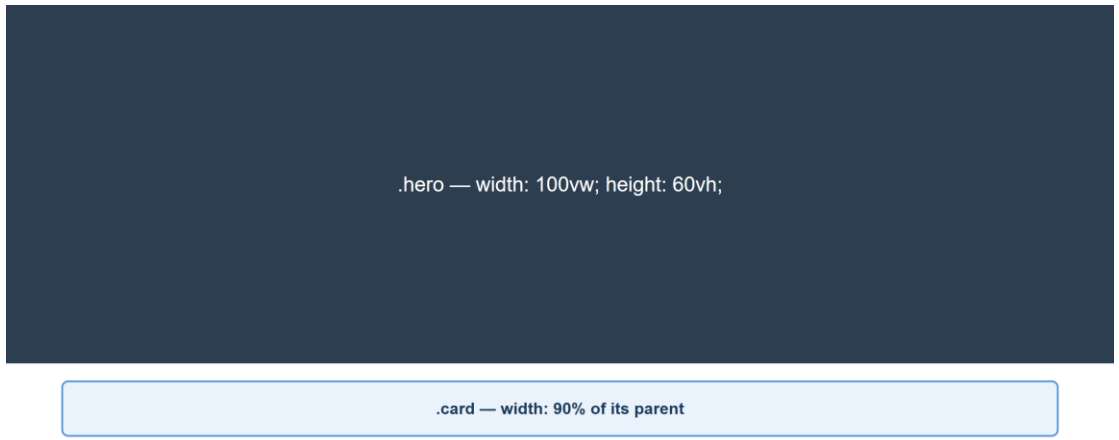
Unit	Relative To
%	The size of the parent element
rem	The root (<html>) element's font size
em	The current element's own font size
vw / vh	1% of the viewport's width / height

Fluid Units in Practice

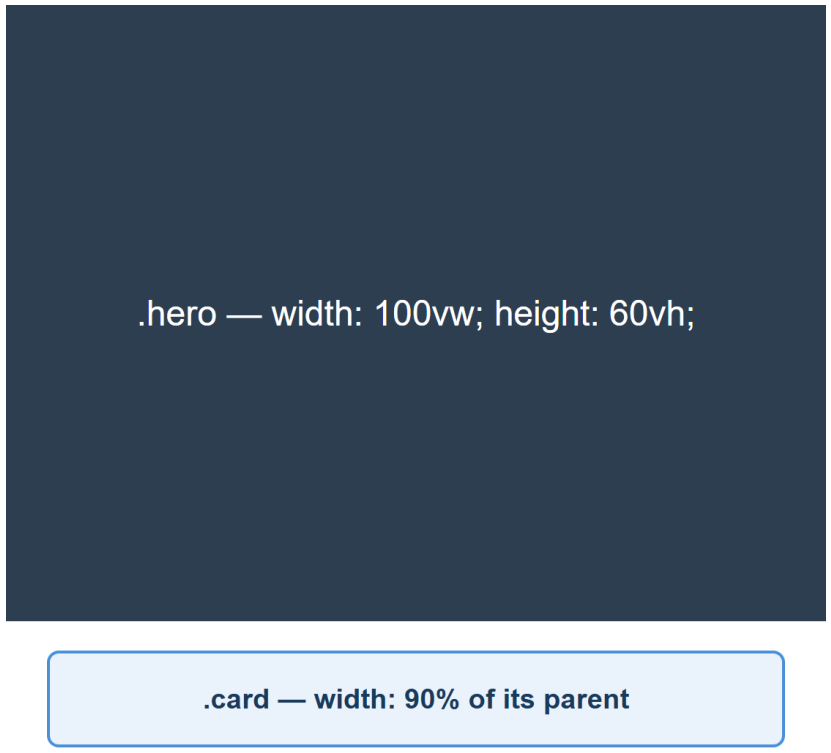
```
.hero {
  width: 100vw;   /* always fills the full screen width */
  height: 60vh;   /* always 60% of the visible screen height */
}
.card { width: 90%; } /* always 90% of its parent, whatever that is */
```

The Same Page, Two Widths

WIDE



NARROW



A Mobile-First Breakpoint

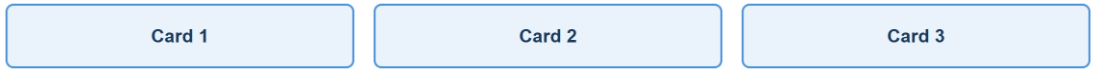
```
.container { display: flex; flex-direction: column; }

@media (min-width: 768px) {
  .container { flex-direction: row; }
}
```

Mobile-First Breakpoint Ladder

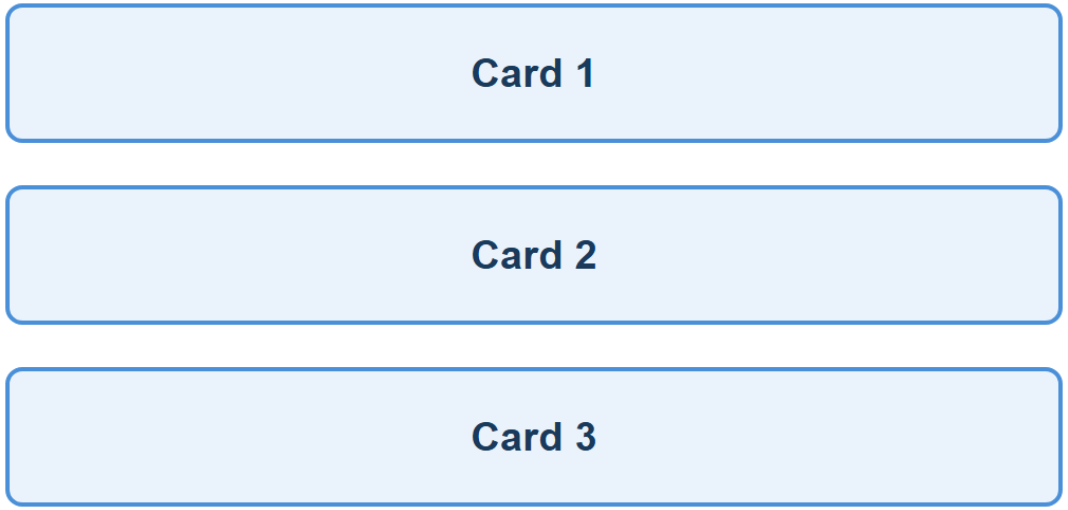
Mobile-first means the base (unqueried) styles target the smallest screen — each breakpoint only ADDS rules for wider viewports, never overrides them back down.

Three Cards, Two Widths

WIDE (ROW)



NARROW (COLUMN)



and, Comma, and not

Operator	Meaning
and	Combines conditions — ALL must be true
, (comma)	Works like OR — ANY one condition must be true
not	Inverts the entire media query (needs a media type)
orientation / prefers-color-scheme / print	Portrait vs. landscape / dark mode / printed page — same @media syntax

The One Rule That Matters

```
img {
  max-width: 100%;
  height: auto;
}
```

max-width: 100% keeps an image from ever growing wider than its container; height: auto preserves its aspect ratio.

The Same Image, Two Widths

WIDE

Container is 60% of the page width. The image's real file is 800×400px.

NARROW

Container is 60% of the page width. The image's real file is 800×400px.

Component-Based vs. Utility-First

BOOTSTRAP (COMPONENT-BASED)

- Ships pre-styled, ready-to-use components (.btn, .card, .navbar)
- Fast to get a decent-looking page up quickly
- Overriding built-in styles can fight the framework

TAILWIND (UTILITY-FIRST)

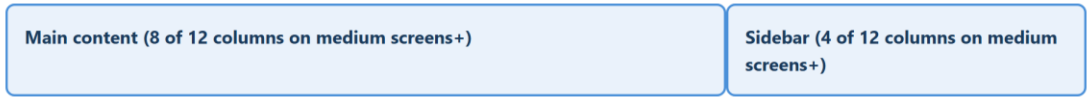
- Ships hundreds of tiny, single-purpose utility classes (flex, p-4)
- Nothing looks "designed" until you style it yourself
- Takes longer at first — many class names to learn

The 12-Column Grid

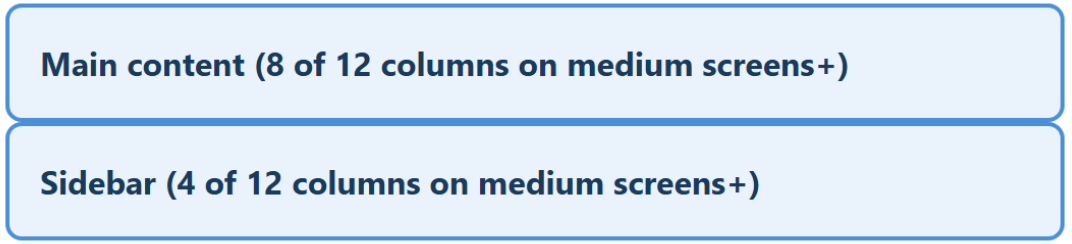
```
<div class="container">
  <div class="row">
    <div class="col-md-8">Main content</div>
    <div class="col-md-4">Sidebar</div>
  </div>
</div>
```

The Grid, Two Widths

WIDE (SIDE BY SIDE)



NARROW (STACKED)



Grid Modifier Classes

Class pattern	Effect
.col (no number)	Auto-layout: shares available width equally
.col-{n}	Exactly n of 12 columns, at all screen sizes
.col-{bp}-{n}	n columns from that breakpoint up (mobile-first)
.offset-{bp}-{n}	Pushes the column right by n empty columns
.order-{n}	Visual display order (0-5), independent of HTML order
.row-cols-{n}	Shorthand: every child column becomes 12/n wide
.g-{n} / .gx-{n} / .gy-{n}	Gutter size between columns — all / x / y

Every one of these is just a wrapper around the Flexbox properties from Lecture 9 — .row is display: flex; .col is flex: 1 0 0%.

Auto-Layout Columns and Nesting

```
<div class="row">
  <div class="col">col</div>
  <div class="col">col</div>
  <div class="col">col</div>
</div>

<div class="row">
  <div class="col-8">
    col-8
    <div class="row">
      <div class="col-6">nested col-6</div>
      <div class="col-6">nested col-6</div>
    </div>
  </div>
  <div class="col-4">col-4</div>
</div>
```

Offsetting and Reordering Columns

offset-md-4 centers a 4-wide column; order- rearranges visual order independent of HTML order.*

```
<div class="col-4 offset-md-4">col-4 offset-md-4</div>

<div class="order-3">First in HTML — order-3</div>
<div class="order-1">Second in HTML — order-1</div>
<div class="order-2">Third in HTML — order-2</div>
```

.col-4.offset-md-4 (centers a 4-wide column)

col-4 offset-md-4

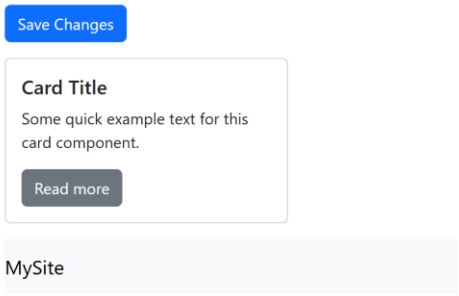
.order-* (visual order independent of HTML order)

Second in HTML — order-1	Third in HTML — order-2	First in HTML — order-3

Components, Rendered

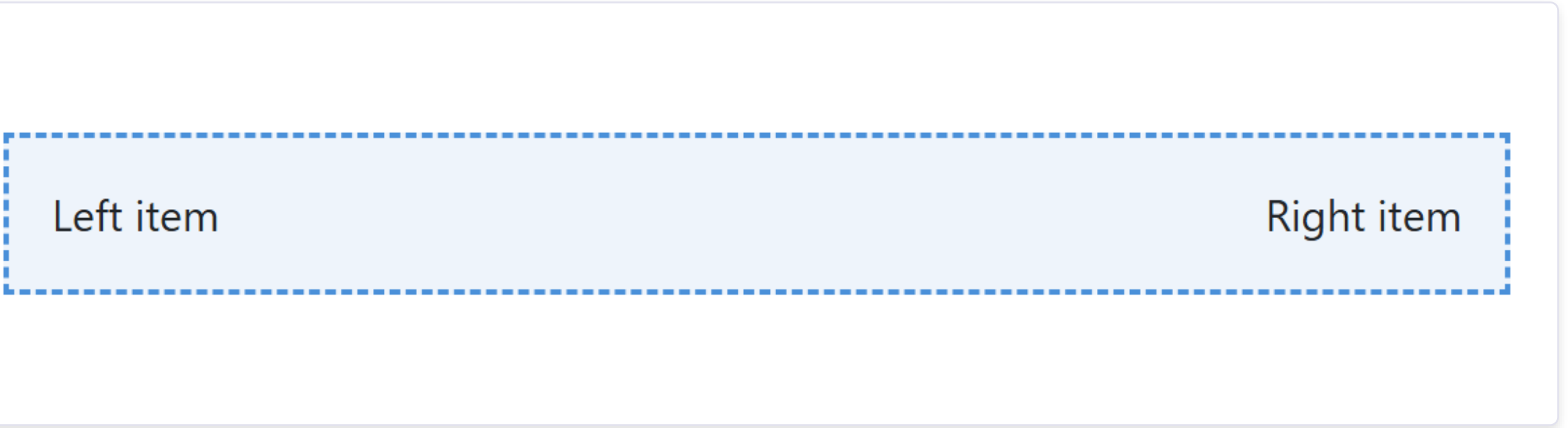
Three lines of HTML, zero custom CSS.

```
<button class="btn btn-primary">Save Changes</button>

<div class="card"><div class="card-body">
  <h5 class="card-title">Card Title</h5>
</div></div>

<nav class="navbar navbar-expand-lg navbar-light bg-light">...</nav>
```



Utilities, Rendered

```
<div class="d-flex justify-content-between p-3">
  <!-- d-flex: display:flex; justify-content-between: space-between; p-3: padding -->
</div>
```



A Fully Styled Button

```
<button class="bg-blue-600 hover:bg-blue-700 text-white font-semibold
               px-4 py-2 rounded-lg shadow">
  Save Changes
</button>
```

The Button, Rendered

Responsive by Default

```
<div class="flex flex-col md:flex-row">
  <!-- stacked by default, a row from md breakpoint up -->
</div>
```

flex-col md:flex-row, Two Widths

WIDE (ROW)

NARROW (COLUMN)

Responsive Width

```
<div class="w-full lg:w-1/3">
  <!-- full width by default, one-third width from the "lg" breakpoint up -->
</div>
```

w-full lg:w-1/3, Two Widths

WIDE (1/3)

NARROW (FULL)

More Utility Categories

Category	Example classes
Flexbox	flex, justify-center, items-center, gap-4
Grid	grid, grid-cols-3, col-span-2
Sizing	w-1/2, h-screen, max-w-md
Borders	border, border-2, rounded-full
Typography	text-lg, font-bold, truncate
State variants	hover:, focus:, active:, disabled: (prefix any utility)

The spacing/color scale is numeric and consistent — p-4 = 1rem, blue-100 (near-white) through blue-900 (near-black).

Configuring Tailwind: theme.extend

```
module.exports = {
  theme: {
    extend: {
      colors: { brand: '#ff6b35' },  // usable as bg-brand, text-brand...
    },
  },
};
```

A Custom Color, Rendered

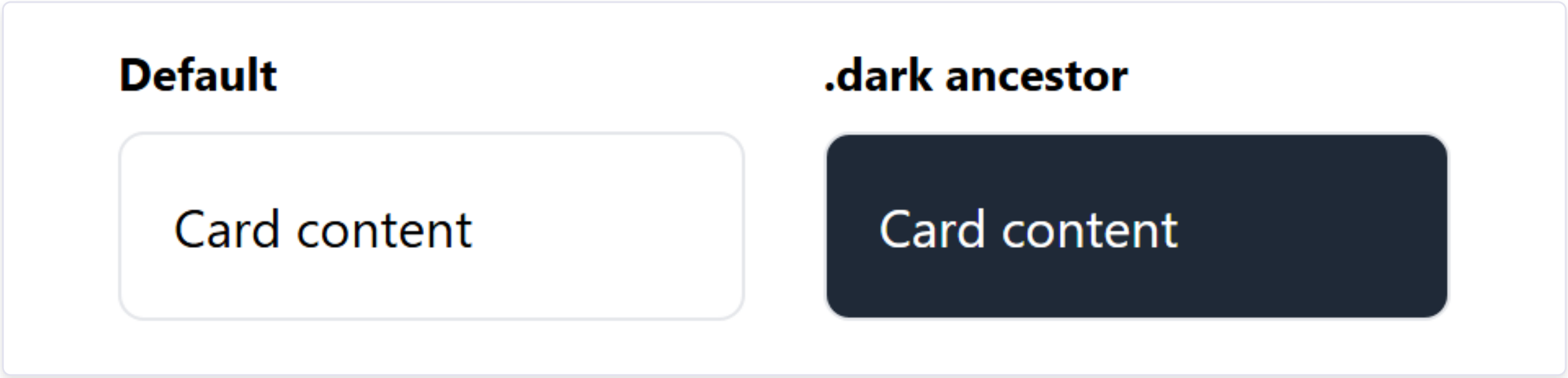
theme.extend adds to Tailwind's defaults, so bg-brand works with every state-variant prefix, just like a built-in color.

```
<button class="bg-brand hover:bg-orange-700 text-white font-semibold
               px-4 py-2 rounded-lg">
  Save Changes
</button>
```

Dark Mode

darkMode: 'class' in the config activates dark: utilities for any element inside a .dark ancestor.

```
<div class="bg-white dark:bg-gray-800 text-black dark:text-white">
  Card content
</div>
```



The @apply Directive

```
.btn-primary {
  @apply bg-blue-600 hover:bg-blue-700 text-white font-semibold
         px-4 py-2 rounded-lg shadow;
}

<button class="btn-primary">Save Changes</button>
```

Produces the exact same button as writing all six utility classes by hand — @apply just names a repeated combination.

Bootstrap + Your Own Class

```
<button class="btn btn-primary my-cta-button">Get Started</button>

.my-cta-button { text-transform: uppercase; letter-spacing: 1px; }
```

Bootstrap + Custom, Rendered

Tailwind + Your Own Class

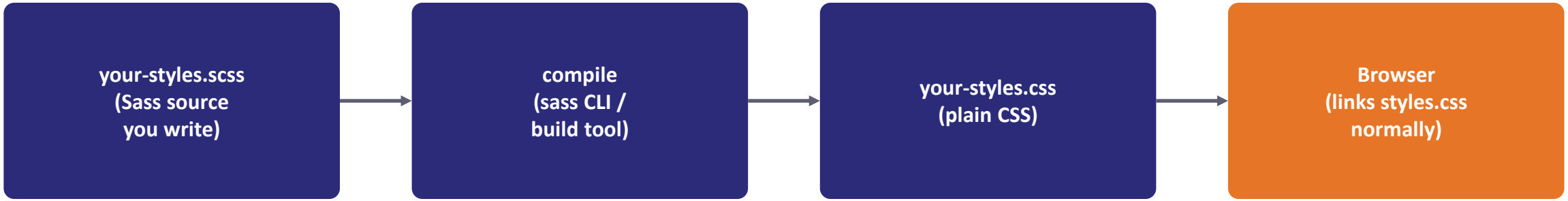
```
<span class="text-blue-600 font-bold brand-underline">New!</span>

.brand-underline { text-decoration: underline wavy; text-underline-offset: 4px; }
```

Tailwind + Custom, Rendered

What Is a CSS Preprocessor?

- A separate language that adds variables, nesting, and mixins on top of CSS, then COMPILES to plain CSS the browser reads
- Sass has two syntaxes: the indented .sass, and SCSS (.scss) — CSS-compatible braces/semicolons, by far the more common choice, including in Bootstrap's own source
- LESS solves the same problems with a different syntax (@var instead of $var) — used by Bootstrap 3, before it switched to Sass

Source vs. What the Browser Runs



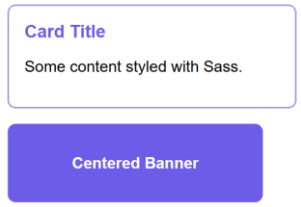
The browser never sees Sass or LESS directly — it only ever loads the compiled, plain .css output, the same "source vs. what ships" split as modern JavaScript that gets compiled/bundled before it runs.

Sass Variables vs. CSS Custom Properties

	Sass ($name)	CSS custom property (--name)
Resolved	At compile time — a fixed value	At runtime, in the browser
Changes after load?	No	Yes — JS or media queries can update it
Needs a build step?	Yes — must compile the .scss	No — works in a plain <style> tag

Variables, Nesting, and Mixins

```
$primary-color: #6c5ce7;

.card {
  border: 1px solid $primary-color;
  .title { color: $primary-color; }
  &:hover { box-shadow: 0 4px 10px rgba(0,0,0,0.2); }
}

@mixin flex-center { display: flex; justify-content: center; align-items: center; }
.banner { @include flex-center; background: $primary-color; }
```



Partials and @use

```
// _variables.scss
$primary-color: #6c5ce7;

// main.scss
@use 'variables' as v;
.card { border: 1px solid v.$primary-color; }
```

A filename starting with _ is a partial — Sass pulls it into whatever file imports it, instead of compiling it separately.

Sass (SCSS) vs. LESS

Feature	Sass (SCSS)	LESS
Variable syntax	$name	@name
Mixin definition	@mixin name { ... }	.name() { ... }
Mixin use	@include name;	.name();
Used by	Bootstrap 4 and 5, most new projects	Bootstrap 3, older codebases

Override Variables Before Importing

```
// custom.scss — override BEFORE importing Bootstrap's source
$primary: #ff6b35;
$border-radius: 1rem;
$grid-gutter-width: 3rem;

@import "bootstrap/scss/bootstrap";

// sass custom.scss custom.css
```

Every component reading these variables picks up the new values everywhere, automatically — no HTML changes at all.

Stock vs. Sass-Customized Bootstrap

STOCK BOOTSTRAP

Stock Bootstrap (default variables)

Save Changes

Card Title

Default radius and color.

Col	Col	Col

SASS-CUSTOMIZED

Sass-customized Bootstrap (custom variables)

Save Changes

Card Title

Custom radius and color.

Col	Col	Col

KEY TAKEAWAYS

Lecture 10 in Eight Points

1. Mobile-first: base styles for small screens, then add complexity with min-width media queries.
2. The viewport meta tag is required for media queries to work correctly on real phones.
3. Fluid units (%, rem, vw, vh) scale relative to something else, unlike fixed px.
4. Bootstrap's grid goes beyond col-md-*: auto-layout columns, nesting, offset-*, and order-* cover real layouts.
5. Bootstrap gives finished components; Tailwind gives composable utilities, configured via tailwind.config.js and dark:.
6. Sass/SCSS and LESS compile to plain CSS, adding variables (compile-time, unlike runtime CSS custom properties), nesting, and mixins.
7. Bootstrap's own source is Sass — overriding its variables before importing it recompiles the whole framework around your values.
8. Both frameworks expect your own custom CSS on top — they solve the common 80%, not the last 20% that makes a site unique.