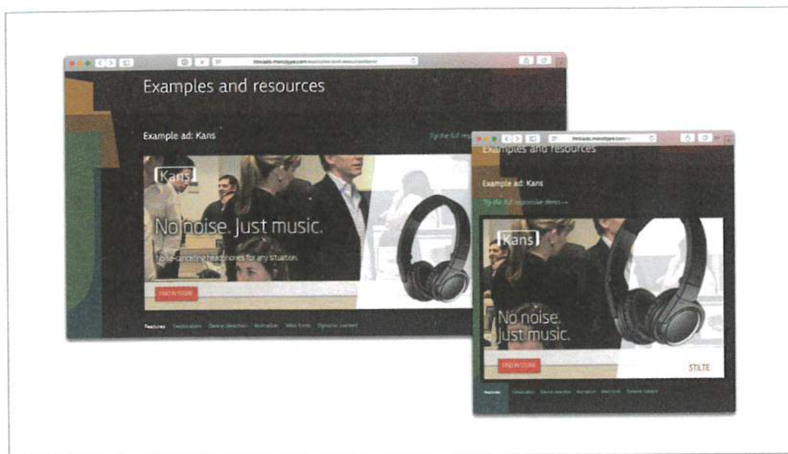


FIG 4.19: Monotype's various responsive ad formats are wonderful proofs-of-concept (http://bkaprt.com/rdpp/04-24/).

which should help make responsive advertising more broadly accessible (http://bkaprt.com/rdpp/04-25/).

Responsive advertising is still very much in its early days. As of this writing, the advertising industry hasn't made much progress on truly cross-device advertising, focusing instead on contractual language to improve ad visibility, and investing heavily on designing distinct ad formats for "mobile" and "desktop" (http://bkaprt.com/rdpp/04-26/). And publishers are painfully aware of this gap, as Peter Bale, CNN International Digital's vice president, noted recently (http://bkaprt.com/rdpp/04-27/):

> *The ad industry has not fully come down the pipe yet in terms of responsively designed ads that will particularly work to the same level of monetisation on any device—there is a lag there. We have to move ahead of that and that's very difficult.*

Until that "lag" disappears, it seems it's up to us to address user-facing issues like performance or layout—and to come up with our own ways of making advertising lightweight, flexible, and responsive.

5 DESIGNING THE INFINITE GRID

> *What works is better than what looks good. The* looks good *can change, but what works, works."*
>
> —RAY EAMES

THE PAST FEW CHAPTERS have probably felt a bit like we've been sitting at a microscope. We've been taking close looks at the more challenging *components* of a responsive design, and discussing common principles for dealing with navigation, images, and advertising. But no element of your design exists in isolation. There comes a time when these small layout systems need to be stitched together into something larger—something flexible, responsive, and—hopefully—beautiful.

This book began with a quote by Trent Walton, one I thought it'd be helpful to return to (http://bkaprt.com/rdpp/05-01/):

> *I traded the control I had in Photoshop for a new kind of control—using flexible grids, flexible images, and media queries to build not a page, but a network of content that can be rearranged at any screen size to best convey a message.*

"A network of content"—such a lovely image, that. But while we've been focusing on the discrete components of our responsive layouts—the content—it's important that we not lose sight of the other half of that phrase: the larger *network* that contains them. As responsive designers, we need to focus not just on the individual bits of a design, but also the relationship between those elements within a larger layout system. Joe Stewart described the responsive redesign of Virgin America in similar terms—not focused exclusively on individual breakpoints, but with a wider, more holistic view (http://bkaprt.com/rdpp/05-02/):

> *For whatever reason, people think it's okay to have focused user experiences being on mobile, but when you get to desktop it's about throwing in the kitchen sink. One of the great things about responsive is that it forces you to make those mobile decisions on a desktop...In terms of the overall way to think about design and design process or responsive, some people like to say there's a mobile-first way of looking at things but with responsive it's everything first.*

Tito Bottita, partner at design agency Upstatement, said that while "mobile first" is a critically important guiding principle, they planned the layout of the *Boston Globe* in a slightly different manner (http://bkaprt.com/rdpp/05-03/):

> *Our designs began at 960px, arguably the most complicated breakpoint, with several columns of content. Maybe this just came naturally after years of designing for that width. But I think it's more than that. It's easier to design with more screen real-estate—you see more at one time, you have a more nuanced hierarchy...So starting at 960, we designed downward. Every decision informed the one before it and the one after; we flipped back and forth between breakpoints a lot. As the Mobile First mantra suggests, designing for mobile was most instructive because it forced us to decide what was most important. And since we refused to hide content between breakpoints, the mobile view could send us flying back up to the top level to remove complexity. The process felt a bit like sculpting.*

As Upstatement found, beginning from the widest, most complex layout didn't preclude them from refining the smallest view of the design—in many instances, reviewing the work at one breakpoint informed the shape of the other. You could use InDesign, as Upstatement did, to quickly create breakpoint-friendly designs, and balance them against each other. Or you could dive into HTML and CSS and create a responsive prototype. The tools are secondary, as there's a larger question at hand: how do we assemble all these distinct components into a larger, useful responsive design?

On that note, we should probably talk about the F-word.

FRAMEWORKS

A number of responsive-specific CSS frameworks have appeared in recent years. Bootstrap (http://bkaprt.com/rdpp/05-04/) and Foundation (http://bkaprt.com/rdpp/05-05/) are two of the most popular, allowing you to quickly create responsive layouts using their established markup (**FIG 5.1**). For example, here's how to create a three-column row of elements with Foundation:

```
<div class="row">
  <div class="small-4 columns">...</div>
  <div class="small-4 columns">...</div>
  <div class="small-4 columns">...</div>
</div>
```

By default, Foundation's layouts are built on a twelve-column grid. By using `small-4`, a class that describes the number of columns each element should span, Foundation's CSS will arrange our three elements in each row (**FIG 5.2–5.3**). And if we wanted to change the priority at a wider breakpoint, we simply need to describe that change in the markup:

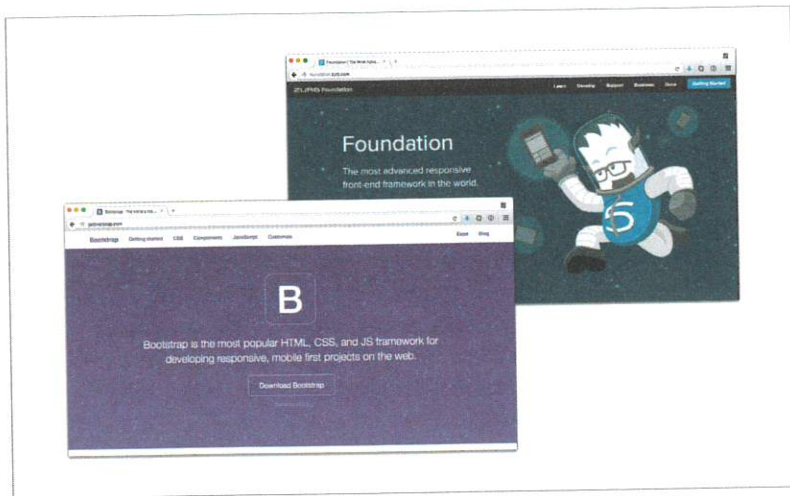


FIG 5.1: Third-party CSS frameworks, like Bootstrap and Foundation, can help you build a responsive layout more quickly.

```
<div class="row">
  <div class="small-12 medium-3 columns">...</div>
  <div class="small-12 medium-6 columns">...</div>
  <div class="small-12 medium-3 columns">...</div>
</div>
```

Above a specific breakpoint—in Foundation, that defaults to a viewport width above `650px`—our `medium-` classes allow the middle `div` to span six columns, while the outer `div`s are reduced to three columns each. And that's all made possible by simply changing a few classes in our HTML.

Neat, right? I think CSS frameworks, responsive or otherwise, are fantastic. If you're working in a team environment, a CSS framework—whether off-the-shelf or developed internally—can take a lot of subjectivity out of creating layouts, eliminate arbitrary class names and HTML structures, and ensure all collaborators are using the same conventions. And for prototyping, there's nothing better: when I'm discussing

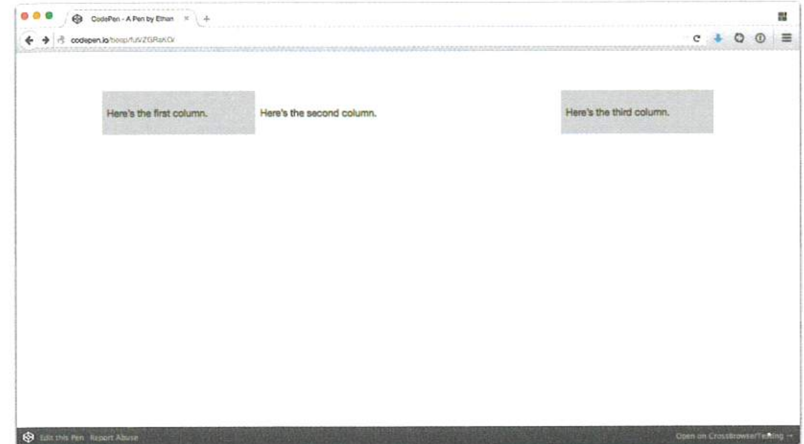


FIG 5.2: A row of three elements, built from simple markup on the Foundation framework.

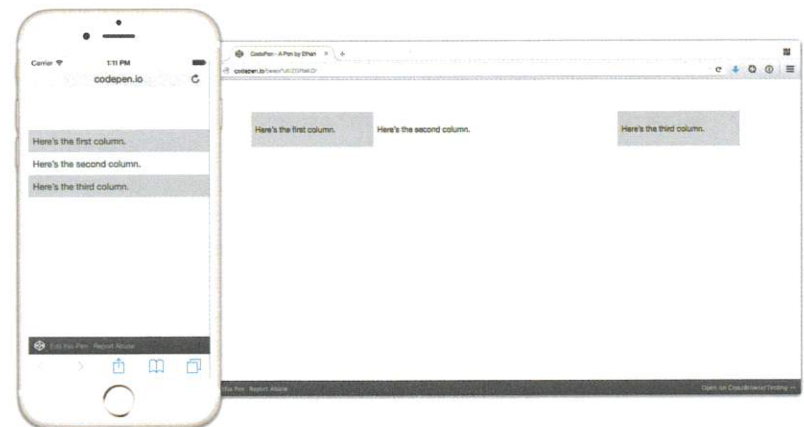


FIG 5.3: With a few of Foundation's markup patterns, we've got a flexible, responsive grid layout.

responsive designs with a client, I'll often grab a third-party CSS framework to quickly mock up a page, using disposable code to get a layout into browsers as quickly as possible. But most important, these third-party CSS frameworks are wonderful

	BOOTSTRAP	FOUNDATION
Small screens	Below 768px	Below 640px
Medium screens	Above 768px	Above 641px
Large screens	Above 992px	Above 1024px
Extra large screens	Above 1200px	Above 1440px

FIG 5.4: Bootstrap and Foundation, two popular responsive frameworks, define their breakpoints in pixels, using values that are closely aligned with common device sizes.

learning resources, allowing designers to better understand the fundamentals of page layout.

But they're heavy.

When I say "heavy," I'm not referring to the weight of the code used in the frameworks, though that could be a valid concern; their additional classes and markup can be, well, bulky. And as we'll see later in the chapter, many CSS-based methods allow us to create robust layout systems without describing columns and rows in our markup.

But putting the bytes aside, there's a larger concern: CSS frameworks are *conceptually* heavy. The layouts they provide are bound to an ideal grid, usually with twelve or sixteen columns of uniform widths. From there, the classes in the markup describe how that grid should adapt at specific breakpoints. And when it comes to Foundation and Bootstrap, the breakpoints they use are very device-specific (**FIG 5.4**).

It's worth noting that these are default values, and they're easy to change. But the out-of-the-box breakpoints are closely associated with specific common devices: `768px` is a common width for 10" tablets, like the iPad, held in portrait mode; `640px` lines up with many smartphones, like the Samsung Galaxy or HTC One, in landscape mode.

As extensible and well-engineered as these frameworks are, their breakpoints are a snapshot of the web as we currently understand it. With the increasing proliferation of browsers, screen sizes, and device classes, we need lighter frameworks—

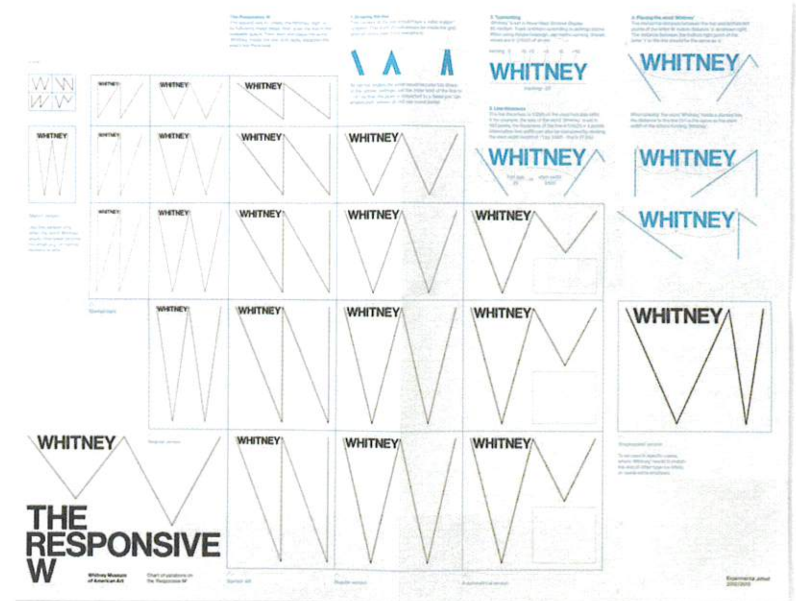


FIG 5.5: The Whitney's new logo in action, which they (coincidentally!) call their "responsive W" (http://bkaprt.com/rdpp/05-07/).

frameworks that can adapt as nimbly as our designs themselves, ensuring they survive beyond just "mobile, tablet, and desktop."

IN 2013, THE WHITNEY MUSEUM rebranded, launching an elegant site (http://bkaprt.com/rdpp/05-06/) alongside its new identity (**FIG 5.5**). Both are cool, contemporary affairs, emphasizing expansive margins and bold, angular lines. While the logo may seem spare and minimal, it has a dizzying array of applications. It can incorporate artwork from the museum, and even be encountered on differently shaped media throughout museumgoers' days (**FIG 5.6–7**). On the Whitney's semi-responsive website, the logo changes dramatically at different breakpoints. Yet among all these variations across countless kinds of media, as much as it reshapes itself, the mark is still recognizable as the Whitney's distinctive "W" (**FIG 5.8**).

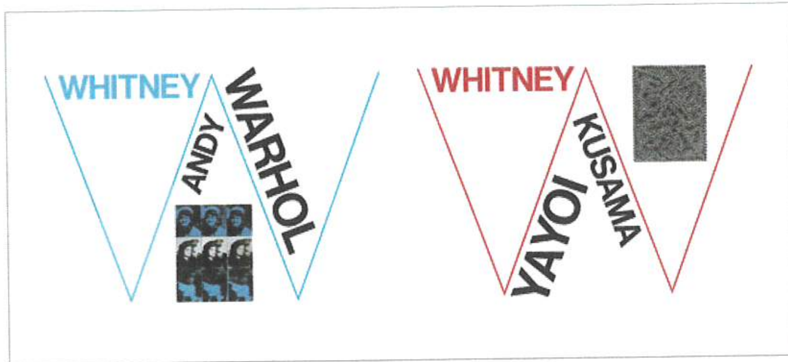


FIG 5.6: As flexible as it is minimal, the Whitney's logo can incorporate artwork from the museum (http://bkaprt.com/rdpp/05-07/).

FIG 5.7: The logo also lives in the physical world, sketched here as it might appear on a bus shelter (http://bkaprt.com/rdpp/05-07/).

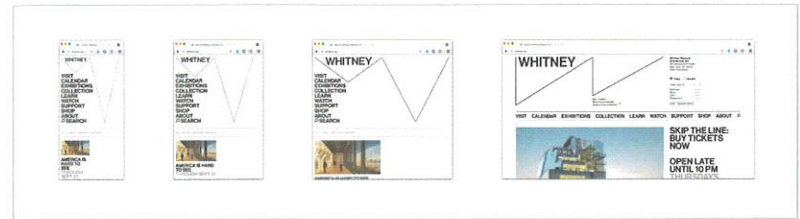


FIG 5.8: The Whitney's logo is just as adaptive as their website, wonderfully enough.

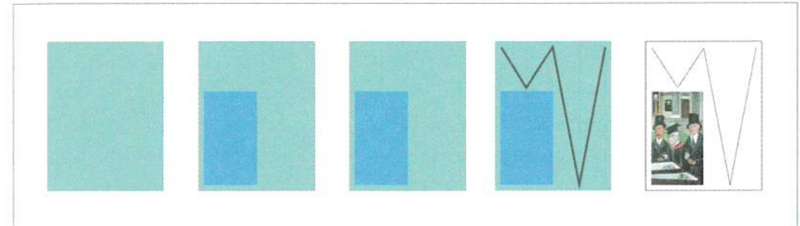


FIG 5.9: A simple framework can yield hundreds, if not thousands, of logo variations (http://bkaprt.com/rdpp/05-07/).

Now, you'd be forgiven for thinking that across all these hundreds, if not thousands, of variations that there was some massive computer churning away in a closet, with pillars of smoke pouring out as it generated algorithmically perfect variations of the logo. (Okay, the secret's out: I've never actually seen a computer.) But no! As it happens, the designers of the logo came up with a straightforward technique for generating a near-infinite number of adaptations (**FIG 5.9**):

1. Within a specific area—a piece of paper or a section of a web page—there will be some elements that need to be incorporated.
2. Divide the remaining available area into four equal columns.
3. From there, it's a simple matter of connecting the dots: from the top of the first column to the bottom of the next; the bottom of the second column to the top of the third; and so on until the "W" is complete (**FIG. 5.9**).

This is, in every sense of the word, a *framework,* but it's so much lighter than the ones we use on the web. It's focused less on *execution*—on laying out specific elements or arranging the columns and rows—and more on defining the characteristics of a desirable outcome: of shaping the conditions that would give rise to a successful mark. And with all the challenges we're facing—and those we're about to face—this is a wonderful model for the kinds of frameworks we *really* need on the web.

THERE'S A STORY that might be relevant here. It's a story about two artists, a boy who dreamed, and drawings that moved.

In the early part of the twentieth century, cartoonist Winsor McCay was one of the most widely circulated artists in the United States. His masterpiece, *Little Nemo in Slumberland,* was massively popular, and with good reason: his broadsheet-sized comics were awash with color and detail, featuring cinematic layouts that have been all but lost on modern newspapers' ever-shrinking comics pages (**FIG 5.10**).

Reportedly inspired by his son's flip books, McCay decided to try his hand at "making moving pictures." His first attempt required over four thousand frames, each meticulously hand-drawn, and featured the characters of *Little Nemo* dancing, fighting, and smoking the odd cigar. (**FIG 5.11**) Over the next decade, McKay created ten films, showcasing his characteristic style and draftsmanship, and translating his lovingly hatched line art into motion. And they are, each and every one of them, stunning (**FIG 5.12**).

I don't think it diminishes any of McCay's achievements to suggest that his animation also feels, well, a tad rudimentary. Viewed through modern eyes, it's easy to see where footage is reused or reversed, allowing McCay to conserve a little effort, and his characters' movements are often a bit mechanical. That's not a criticism—working alongside early animators like Émile Cohl, James Stuart Blackton, and Max Fleischer, McCay and his peers were at the forefront of defining what animation could be.

But it wasn't until Walt Disney formed his studio a few decades later that people began to take animation seriously as an art form. That might sound a bit strong, but it's true: the

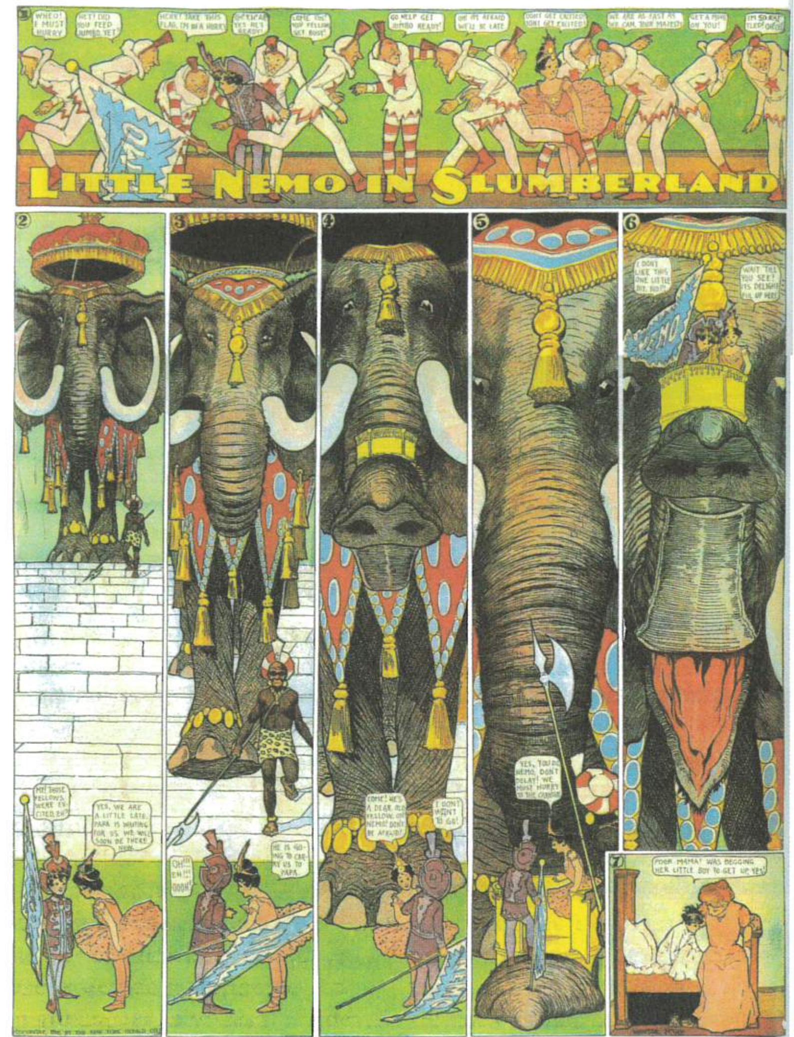


FIG 5.10: Winsor McKay's *Little Nemo in Slumberland* is stunning, even over a century after its publication. And thankfully, it's freely available on the Internet Archive (http://bkaprt.com/rdpp/05-08/).

FIG 5.11: For his first animated short film, Winsor McKay adapted *Little Nemo* for the screen. Video image from YouTube (http://bkaprt.com/rdpp/05-09/).

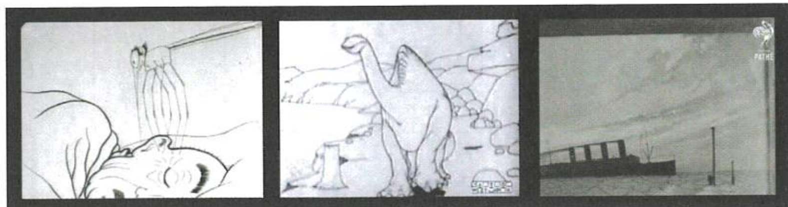


FIG 5.12: McKay's handcrafted films were part of the earliest days of line-drawn animation, covering topics ranging from the slightly macabre (http://bkaprt.com/rdpp/05-10/), to the patently adorable (http://bkaprt.com/rdpp/05-11/), to the sweepingly dramatic *The Sinking of the Lusitania* (http://bkaprt.com/rdpp/05-12/).

characters in Disney's features possessed an elegance of motion that hadn't been seen in animated movies before. Marc Davis, one of the studio's original animators, said that when Walt Disney started his studio, "animation had been done before, but stories were never told." Now, that is some forceful trash-

talking, as Davis is calling out everyone who'd ever tried their hand at animation before Disney—but I think there's some truth to it. To be honest, it's pretty hard to exaggerate the effect the studio's work had on filmgoing audiences. Once those simple, spare line drawings were colored and animated, they felt real. They felt *human*. And that's something that hadn't really been done before.

The studio's success was due, in part, to Disney himself, but it wasn't because he was a more talented animator than his employees or competitors. Rather, it was because he was an incredibly exacting director; he demanded that the animation his studio produced possess a "caricature of realism" or "illusion of life" with which his audiences could connect and empathize. Some decades later, two of Disney's original animators, Frank Thomas and Ollie Johnston, borrowed that phrase as the title of their wonderful book, *The Illusion of Life*. In it, Thomas and Johnston define what they called "The Twelve Basic Principles of Animation," which allowed Disney's animators to meet Walt's rather exacting demands. These principles—covering animation concepts like staging, timing, arcs, and more—were the bedrock for the studio's work, and have become the foundation for what we consider to be quality animation in modern times (**FIG 5.13**).

On a personal note, I can't recommend Thomas and Johnston's book highly enough—it's a lovely read, especially for such a technical book. But if you're pressed for time, a paraphrased version of their guidelines are available on their website (http://bkaprt.com/rdpp/05-15/). As you read through them, you'll notice these principles aren't especially technical. Rather than using obtuse jargon, they explain *how* to judge whether or not a drawing possesses that illusion of life that defined Disney's work. Did a character's arm properly "anticipate" that it was about to throw a ball? Did a character's gait have enough "squash and stretch" as it walked from one end of the frame to the other?

These principles were a kind of a shared vocabulary, one that allowed the studio's animators to discuss how their work measured up to Disney's famously high standards. Rather than dictating the use of certain animation techniques or emphasiz-

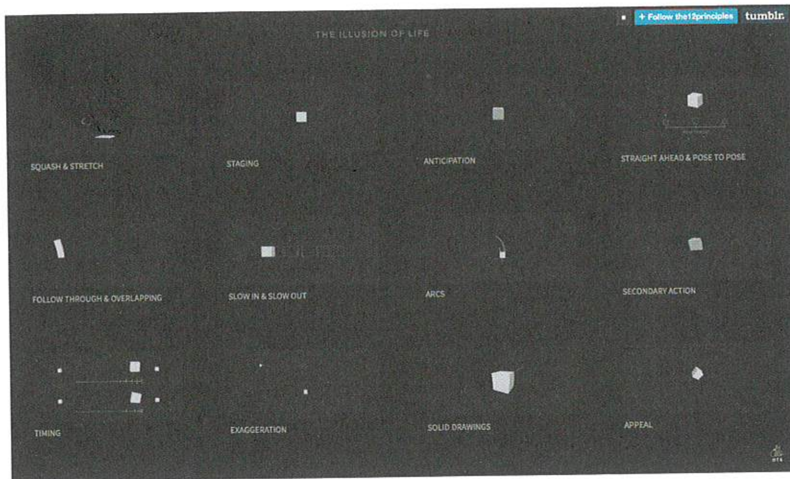


FIG 5.13: Disney's principles of animation were recently adapted by Cento Lodigiani into—you guessed it—an animated primer that demonstrates them handily (http://bkaprt.com/rdpp/05-13/). Video image from Tumblr (http://bkaprt.com/rdpp/05-14/).

ing steps in their production workflow, the principles allowed the studio to discuss and evaluate the *quality* of its work.

This is the conversation we need to have.

Over the past few years, we've been learning how to adapt our layouts to the infinite canvas of the web. Our sites can be viewed on any size screen, at any time, and responsive design is one approach that lets us accommodate the web's variable shape. But with all of the challenges we're facing and those yet to come, we need to begin building not just patterns, but *principles* for responsive design—principles that will allow us to focus not just on layout, but on the quality of our work.

If each part of your responsive interface is more or less self-contained—with its own layout rules, content needs, and breakpoints—then the code behind each element's design is far less important than thinking carefully about *how* and *why* an element should adapt. In other words, how do we move beyond thinking in terms of columns and rows, and start talking about the quality of our responsive designs? And what would frameworks to support that look like?

FINDING THE WORDS

Honestly, there's no perfect answer to that question. But recently, a number of designers and organizations have started sharing the vocabulary they use to decide how and when their responsive designs should adapt. Vox Media, for example, thinks of their content as existing within a river—and in keeping with the metaphor, the flow of that content can be interrupted at certain points. Here's how they describe the front pages of Vox.com (http://bkaprt.com/rdpp/05-16/):

> *Content flows around "rocks" and "breakers", which are modules such as a "Most Commented" list or a row of "Popular Videos." Many of these behaviors remain in the new layout system, but the key difference is an added contextual layer. Elements in the river are laid out to better highlight the diversity of content on Vox — articles, features, videos, editorial apps, card stacks, to name a few. Each one displays differently depending on its type and neighboring entries.*

Note that the language they use to talk about the quality of their layouts doesn't revolve around columns or rows. There's nary a mention of grids. For Vox, the design process begins with content priority and evolves into a layout. By understanding the weight and importance of each piece of content that flows through the river, the Vox team can algorithmically generate a responsive layout that best reflects the importance of the information within it.

Starting with an abstract system of columns and rows would be wrong for them—and, I'd argue, for every web designer. After all, according to Mark Boulton, there are three fundamental benefits of a grid system (http://bkaprt.com/rdpp/05-17/):

- *Grid systems* ***create connectedness.*** *A well-made grid can visually connect related pieces of content or, just as importantly, separate unrelated elements from each other. In other words, they help us create narratives from our layout.*
- *By establishing predefined alignment points, grid systems help designers* ***solve layout problems.***

- *A well-designed grid system will* ***provide visual pathways for the reader's eye to follow,*** *and allow them to better understand a visual hierarchy.*

As Boulton notes, we historically created grid systems by adopting a "canvas in" method. Working from the edges of a printed page, designers would subdivide a page into a system of columns and rows, and place images and text upon that grid in pleasing, rational arrangements. But the web doesn't have any such boundary—after all, it's the first truly fluid design medium. As a result, Boulton argues we should instead adopt a "content out" approach to designing our grids: to build more complex layout systems out from a foundation of small, modular pieces of content. And to do so, Boulton proposes three guiding principles:

- ***Define relationships from your content.*** *A grid for the web should be defined by the content, not the edge of an imaginary page.*
- ***Use ratios or relational measurements above fixed measurements.***
- ***Bind the content to the device.*** *Use CSS media queries, and techniques such as responsive web design, to create layouts that respond to the viewport.*

By understanding the shape of our content, we can create flexible layouts that support connectedness—not just between related pieces of information, but between our layouts and the device. We can make responsive grid systems that don't just fit on an ever-increasing number of screens—they'll feel at home, wherever they're viewed.

FINDING THE SEAMS

Principles are wonderful, of course, but we still have to find a means of implementing them: of translating those ideals into practical responsive patterns and layouts. For me, that "content out" process begins by looking at the smallest version of a piece

of content, then expanding that element until its seams begin to show and it starts to lose its shape. Once that happens, that's an opportunity to make a change—to introduce a breakpoint that reshapes the element and preserves its integrity.

But first, we need a method of finding an element's seams, and understanding how it loses its shape. For me, that process begins by looking at four characteristics: **width**, **hierarchy**, **interaction**, and **density**.

Width

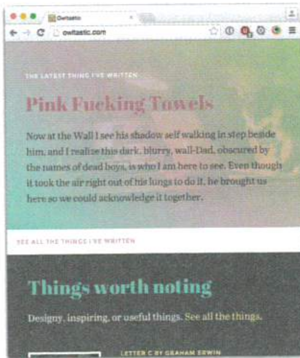
Width might be a little self-evident. As the width of a viewport changes, so does the width of a responsive design. But as the design gets wider or narrower, so will the elements within it, and as those modules expand or contract, there may be opportunities to add a breakpoint (**FIG 5.14**).

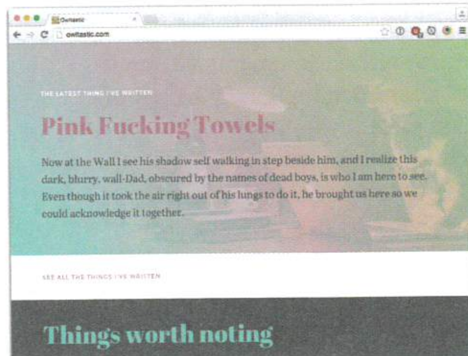
Hierarchy

Width is, I'm sure you'll agree, the most common characteristic of a responsive design—but it's not the only one. As the shape of an element changes, the *hierarchy* of elements may need to change as well.

Let's take a quick look at a product page on Tattly's responsive ecommerce site (http://bkaprt.com/rdpp/05-19/). When viewed on wider screens, the primary content area has two key pieces of information: a photo carousel of the product on the left, and a call to action to purchase the product on the right (**FIG 5.15**). But that's just one view of this particular part of the design, because as screens get narrower, we lose the ability to place multiple columns side by side. That's where a question of hierarchy arises: in a single-column layout, which piece of content should appear first? Tattly opted, quite rightly, to lead with photos of the product—but you may answer hierarchy questions differently on your site (**FIG 5.16**).

Hierarchy is generally a reminder to be more *vertically aware* in our designs. After all, we have `min-width` and `max-width` media queries, but can also avail ourselves of `min-height` and `max-height` queries more often. I think the navigation menu for





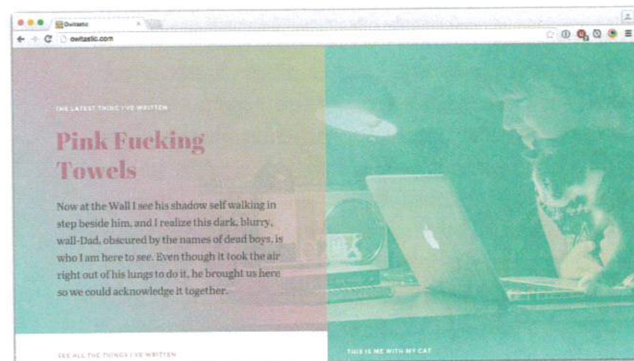


FIG 5.14: On her stunning responsive portfolio, Meagan Fisher adjusts the typography of certain elements—not just their layout—as their width expands and contracts (http://bkaprt.com/rdpp/05-18/).

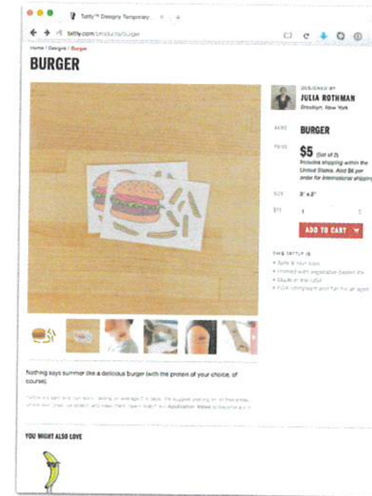


FIG 5.15: On Tattly's responsive ecommerce site, the product content is laid out in a pleasing two-column grid on wider screens.

FIG 5.16: On narrower viewports, the hierarchy of product information shifts from two columns to one.

the Field Museum (http://bkaprt.com/rdpp/05-20/) beautifully balances vertical and horizontal layouts (FIG 5.17). On wider screens, the navigation is anchored to the left edge of the design, and spans the full height of the viewport. You may notice that they're using the flexible box model, or *flexbox,* an advanced CSS layout method we'll look at later in this chapter (http://bkaprt.com/rdpp/05-21/). But since flexbox allows elements to automatically fill the space available to them, as the menu gets taller or shorter, the navigation elements resize vertically—but below a certain width *or* height, the menu is placed at the top of the page.

By minding the navigation's vertical edges, the Field Museum was able to introduce alternate layouts to ensure the content inside their navigation menu was never concealed, obscured, or clipped. In other words, the breakpoints we introduce to our responsive designs aren't tied to the shape of a device's screen.

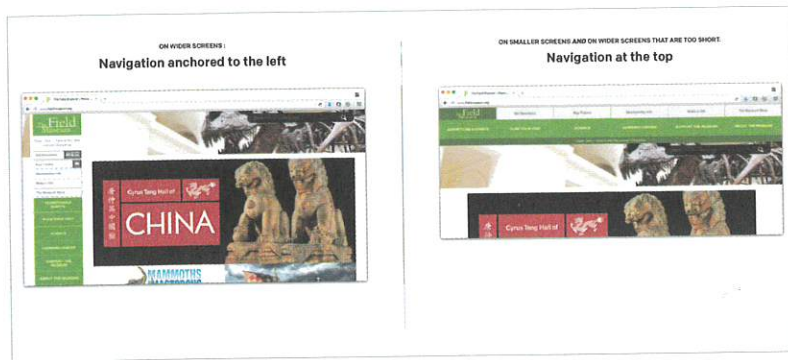


FIG 5.17: The responsive navigation for the Field Museum, which occupies the height of the design. Below a certain width, it moves to the top of the screen to avoid cropping.

Instead, our media queries defend the integrity of the content we're designing.

Interaction

The way we *interact* with an element may change along with the design. Responsive navigation systems are probably the most obvious example of this. As we saw in Chapter 2, menus are often displayed in full at wider breakpoints but concealed at smaller ones, perhaps hidden behind expandable icons or links when space is at a premium (**FIG 5.18**).

But navigation isn't the only kind of content that might require interaction changes. For example, take the responsive sports brackets designed by SB Nation (http://bkaprt.com/rdpp/05-23/). While they appear as complex-looking charts at wider breakpoints, a simpler, more linear view of the brackets is shown on smaller screens (**FIG 5.19**). Along with the simplified layout, the brackets are presented as carousels in the smaller view, where real estate is more dear. Each of the four regions for the bracket are a single slide in the carousel, allowing the user to cycle through for details. The information in both visual states is the same, but the interaction model changes.

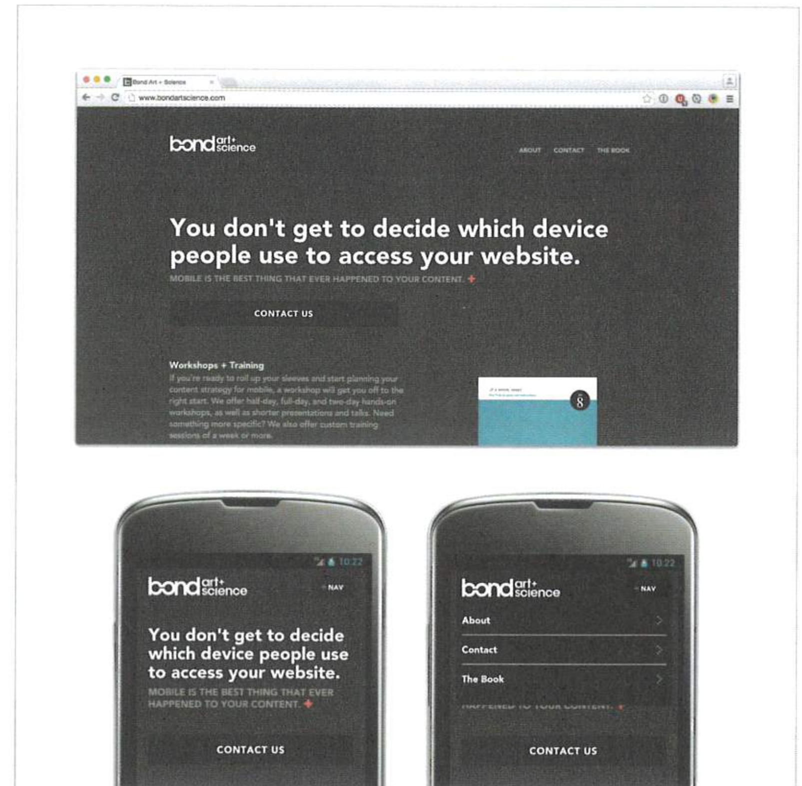


FIG 5.18: Karen McGrane's company site has a traditional-looking navigation at wider breakpoints, but on smaller viewports the user toggles the visibility of the menu. Same links, but a new interaction model (http://bkaprt.com/rdpp/05-22/).

Density

Finally, the amount of information you're showing in an element might need to vary over time—in other words, the *density* of information can change. The *Guardian's* feature on the 2015 Oscars is full of examples of this, with responsively designed timelines displaying a significant amount of movie data. Above a

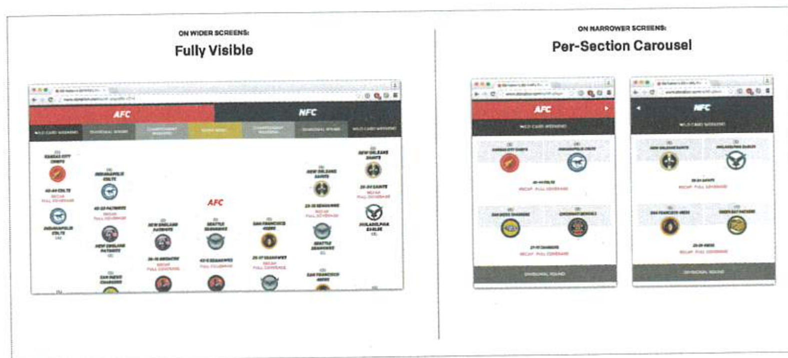


FIG 5.19: I don't know from sports, but I know I like SB Nation's responsive brackets: complex charts on wide screens, but a carousel of match-ups on smaller viewports. Same information, different interaction.

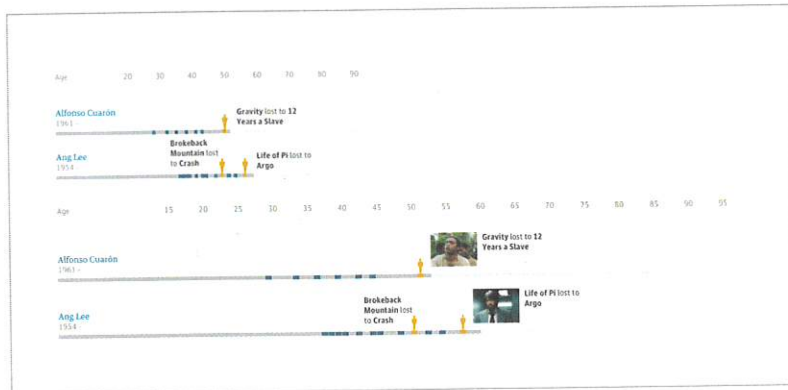


FIG 5.20: The *Guardian's* responsive cinematic timelines gradually increase in density, displaying an extra image above a certain width (http://bkaprt.com/rdpp/05-24/).

certain width, thumbnail images are loaded in, slightly increasing the visual (and informational) density of the timeline (**FIG 5.20**).

Density is, as you might have guessed, an area where you should tread very carefully. As we've discussed before, remov-

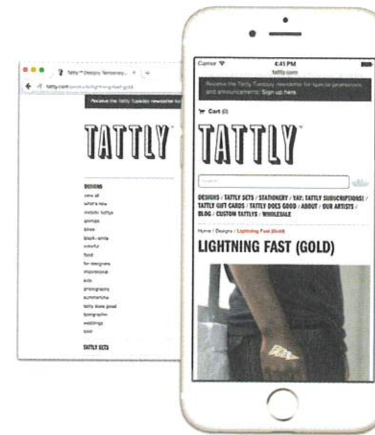


FIG 5.21: Tattly hides its submenu entirely, reducing its navigation to a list of primary sections on smaller screens.

ing or hiding information because it doesn't fit can be problematic (**FIG 5.21**). Personally, I think the *Guardian's* timelines work so well because the images shown at wider breakpoints are enhancements: they're not critical to understanding the information around them. Could they have designed alternate versions of the timelines to show images at all breakpoints? Possibly. But I think this is a wonderful example of density used to lighten the visual impact of a design, removing extraneous information without impeding access to the content.

SHAPING OUR SEAMS: MOVING BEYOND CLASSES

Width, hierarchy, interaction, and density work incredibly well for identifying the outer limits of our tiny layout systems, but you may want to supplement them with other concepts. Recently, designer Nathan Ford suggested a number of useful patterns for identifying when *relationships* between elements begin to break down, including layout anti-patterns like sevens,

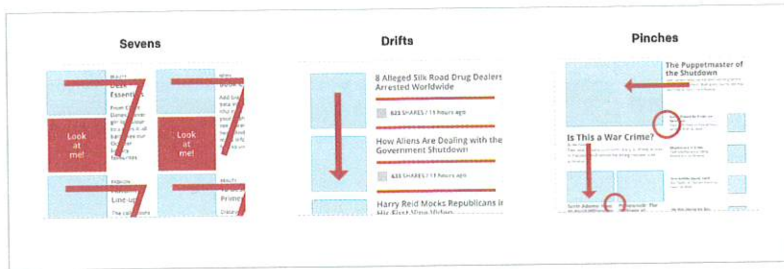


FIG 5.22: Sevens, drifts, and pinches—oh my! Nathan Ford suggests a number of useful areas where your design might degrade (http://bkaprt.com/rdpp/05-25/).

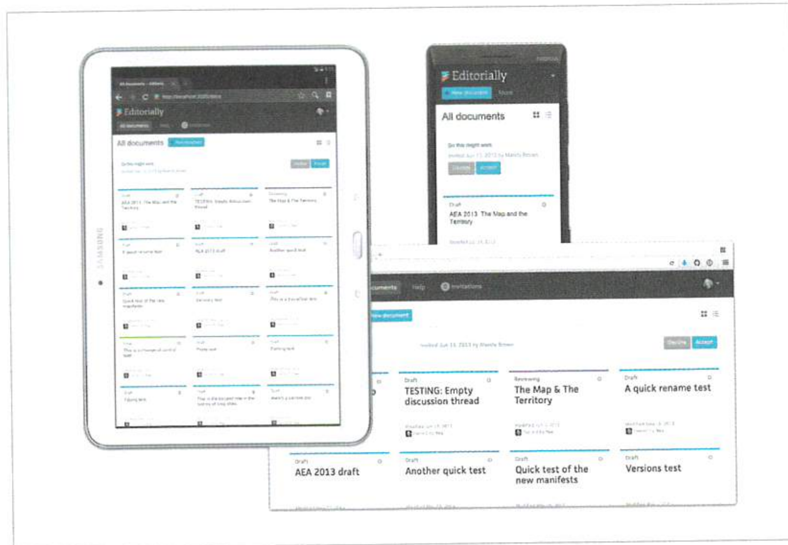


FIG 5.23: Editorially's responsive dashboard used a custom, lightweight framework to create content-driven breakpoints.

drifts, and pinches (FIG 5.22). But regardless of how you choose to find the seams in your layout, moving beyond simple markup classes will give you considerably more flexibility.

Editorially, a (sadly defunct) web service for writers and editors I cofounded, used a custom layout framework for the

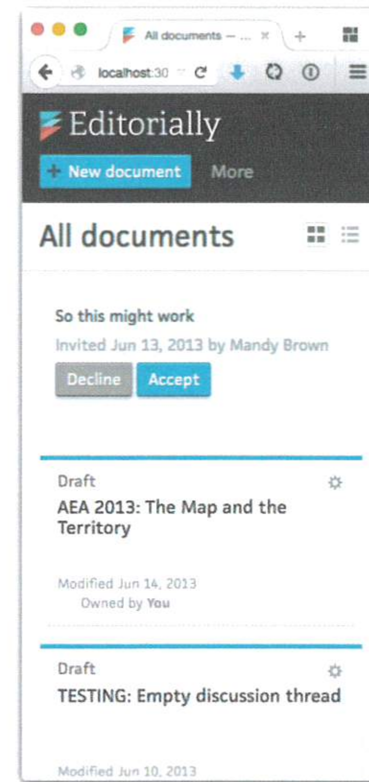


FIG 5.24: The foundation for Editorially's layout? A single-column grid.

most complex parts of its interface. The most visible example of this was on Editorially's dashboard, which displayed a list of documents owned by and shared with the user (FIG 5.23). As part of the team that built the dashboard, I started with a small-screen-friendly layout: a series of tasks and content, laid out vertically in a single-column, hierarchical grid (FIG 5.24).

Using a "mobile-friendly" layout as your foundation is a sign of a responsibly made responsive layout—but we don't have to stop at one column. For example, as Editorially's dashboard reached a width of `31em`, it shifted to a two-column layout, using some concise CSS (FIG 5.25):

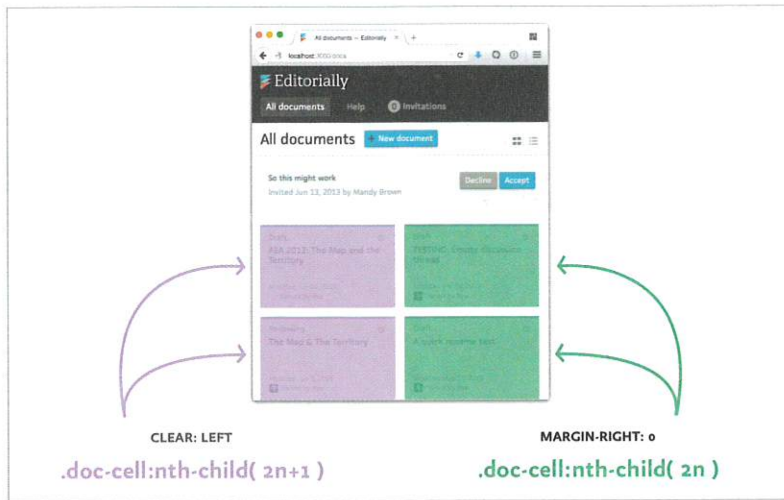


FIG 5.25: With a little `:nth-child()` magic, we can quickly create a two-column layout.

```
@media screen and (min-width: 31em) {
  /* Set widths */
  .doc-cell {
    float: left;
    width: 47.602739726027397260%;
      /* 278px / 584px */
  }
  /* Build the new layout */
  .doc-cell:nth-child(2n) {
    margin-right: 0;
  }
  .doc-cell:nth-child(2n+1) {
    clear: left;
  }
}
```

Wait—what just happened? Three selectors, and we have a new grid layout? Yep, and it's all thanks to the `:nth-child()` pseudo-class. Instead of relying on a CSS framework that requires classes in our HTML to describe our layout,

`:nth-child()` provides us with an incredibly powerful way to address specific elements of our design based on where they fall within the structure of our document. For example, if I wrote `.doc-cell:nth-child(4)`, my CSS would select the `.doc-cell` element that was the fourth child of its parent:

```
<div class="doc-grid">
  <div class="doc-cell">…</div>
  <div class="doc-cell">…</div>
  <div class="doc-cell">…</div>
  <div class="doc-cell">…</div>
  <div class="doc-cell">…</div>
</div>
```

Similarly, if I wrote `.doc-cell:nth-child(2)`, my rule would select the `.doc-cell` element that's the second child of our grid:

```
<div class="doc-grid">
  <div class="doc-cell">…</div>
  <div class="doc-cell">…</div>
  <div class="doc-cell">…</div>
  <div class="doc-cell">…</div>
  <div class="doc-cell">…</div>
</div>
```

Seems fairly straightforward, right? Well, things get *really* interesting when `n` appears inside `:nth-child()`. When that happens, `n` acts as a counter: its value begins at zero, then increments by one each time. So `:nth-child(2n)` becomes simple multiplication: just multiply the value of `n` by the number next to it, then add one to `n`, and then repeat the process.

```
2 × 0 = 0
2 × 1 = 2
2 × 2 = 4
2 × 3 = 6
…
```

In our CSS above, `.doc-cell:nth-child(2n)` is a way to instantly select every even-numbered cell in our grid, regardless of whether there are twenty cells in the grid, or twenty thousand. Once we've made that selection, applying a `margin-right: 0;` ensures that every second cell of our two-column grid is aligned flush against the right edge of the design.

`.doc-cell:nth-child(2n+1)`, on the other hand, selects every second cell—but the `+1` in the selector says we should start from *one,* not zero. Here's how the math would look:

```
(2 × 0) + 1 = 1
(2 × 1) + 1 = 3
(2 × 2) + 1 = 5
(2 × 3) + 1 = 7
…
```

Therefore, `.doc-cell:nth-child(2n+1)` applies a `clear: left` to every odd-numbered cell, ensuring that each row is a nice, discrete grouping of two documents (**FIG 5.25**).

Since we're using `:nth-child()`, rather than specific classes that exist in our markup, we can quickly—and dramatically—revise the grid's layout at each breakpoint. For example, as the dashboard viewport gets a little wider, we can introduce a three-column layout at a breakpoint of `44em` (**FIG 5.26**):

```
/* 3-column */
@media screen and (min-width: 44em) {
  /* Set new widths */
  .doc-cell,
  .doc-cell:nth-child(n) {
    margin-right: 3.0439684329199549O4%; /* 27 /
  887 (per comp) */
    width: 31.003382187147688838%;
      /* 278px / 887px (per comp) */
  }
  /* Reset clears from previous breakpoint */
  .doc-cell:nth-child(n) {
    clear: none;
  }
```

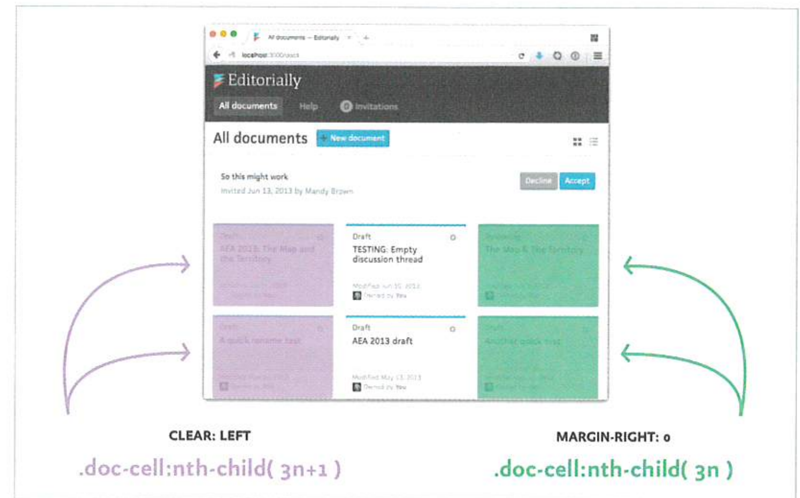



FIG 5.26: Another breakpoint, another grid layout: this time, three columns with `:nth-child(3n)`.

```
  /* Build the new layout */
  .doc-cell:nth-child(3n) {
    margin-right: 0;
  }
  .doc-cell:nth-child(3n+1) {
    clear: left;
  }
}
```

While this media query looks a little complex, it's simply repeating the process from our two-column layout:

1. New, flexible widths and margins are assigned to the `doc-cell` elements in our grid.
2. We can use `:nth-child(n)` to quickly reset styles inherited from the previous breakpoint. (In this case, we're removing the `clear: left;` applied to the start of each row.)
3. Then, we use `:nth-child(3n)` to remove the right margin of every third cell of our dashboard.

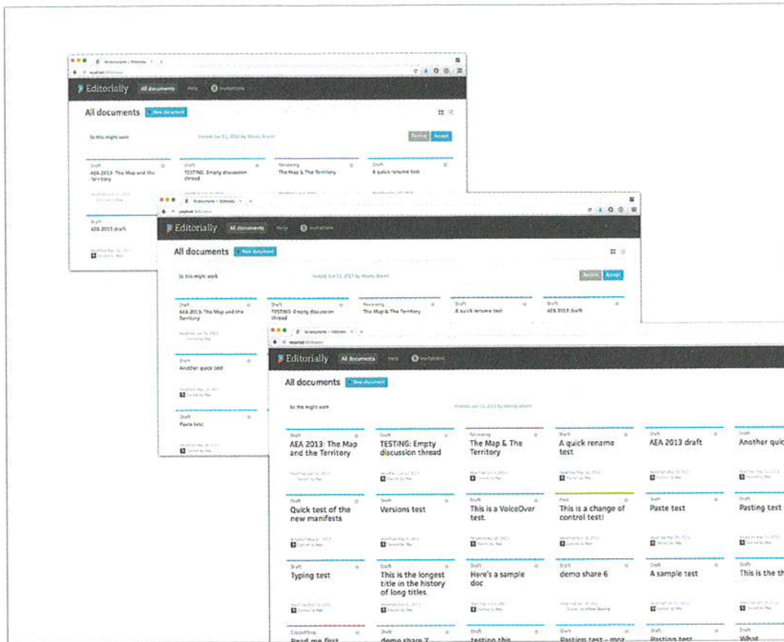


FIG 5.27: Sky's the limit: lightweight markup and powerful CSS can bring our responsive grids from four columns, to five, and finally up to a six-column layout. (And theoretically, beyond.)

4. Finally, `:nth-child(3n+1)` instructs the first cell in every three-column row to clear the cells before it.

I think you can see where this is going. Over time, Editorially's dashboard evolved from a single-column grid into two- and three-column variants, continuing all the way up to a final six-column layout (**FIG 5.27**). Realistically, we were only constrained by the time and resources we poured into the design. By moving the layout logic out of markup and into CSS, we gained infinitely more flexibility (**FIG 5.28**).

Thankfully, it's not just `:nth-child()`. There are a *truckload* of equally nimble layout tools coming out of the CSS specification. Flexbox, which places elements in horizontal or vertical

FIG 5.28: Index pages on our site for the Responsive Design Podcast use a similar `:nth-child()`-based layout framework (http://bkaprt.com/rdpp/05-26/).

stacks, is one of the most popular. The masthead on Frank Chimero's blog is a wonderful example of this. By setting `display: flex` on the header, the two elements within it—the navigation and his logo—are immediately laid out horizontally, each becoming a column occupying the full width that row (**FIG 5.29**).

Hypothetically, if Chimero were so inclined, he could reverse the order of the two elements by adding `flex-direction: row-reverse` to the masthead—all without touching the markup (**FIG 5.30**). This acts a bit like a change in gravity: while they're still laid out in a row, the order of the two items instantly reverses.

These more lightweight layout models have become incredibly popular, and have been embraced by a number of responsive designs (**FIG 5.31**). But as flexible as they are, flexbox and `:nth-child()` aren't without their drawbacks. As Jake Archibald notes, flexbox is ideal for small-scale layouts, but can negatively affect page rendering if used for page-level grids (http://bkaprt.com/rdpp/05-30/). Additionally, some of these properties won't work in older browsers—`:nth-child()` isn't supported by Internet Explorer 8 or below, and `display: flex` won't work at all in IE9 or lower. (And confusingly, there are differing implementations of flexbox in IE10, Safari, and many versions of Android.)

But if we take these considerations to heart, and if we design appropriate fallbacks for the browsers that need them, these lightweight layout tools allow us to design grid systems that

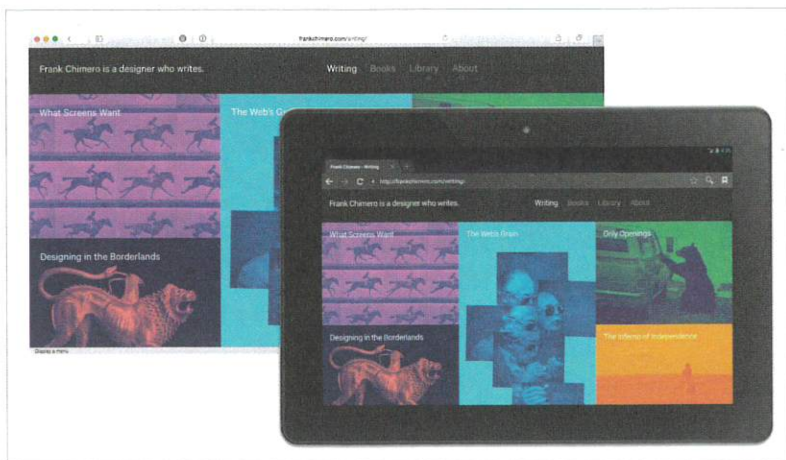


FIG 5.29: Frank Chimero's lovely responsive site features a little flexbox in its layout (http://bkaprt.com/rdpp/02-27/).

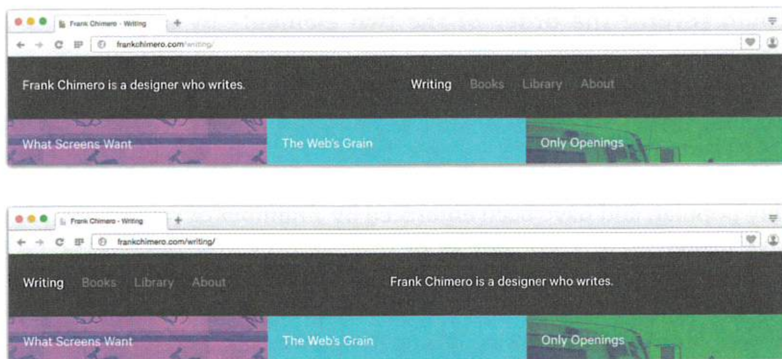


FIG 5.30: By changing the `flex-direction` on a flexible box, you can quickly reverse the, um, direction of the elements within it.

are effectively infinite in scale. As we move away from device-specific breakpoints, and adapt our layout systems according to the location of their seams, we'll be creating more robust, more future-friendly layouts. In other words, we'll be better prepared for the devices and browsers we haven't even imagined yet.

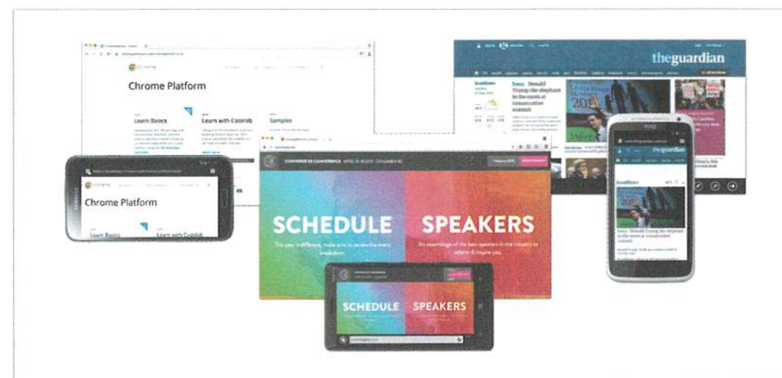


FIG 5.31: Google Chrome's platform reference (http://bkaprt.com/rdpp/05-27/), the ConvergeSE conference (http://bkaprt.com/rdpp/05-28/), and the *Guardian* (http://bkaprt.com/rdpp/05-29/): all fine examples of flexbox lovingly applied to multi-device layouts.

THE SEAMS WITHIN

And really, that's the sticking point, isn't it? We've moved far beyond the desktop, but we're still trying to find the right words to encompass the scope of what we're designing and where it'll appear. Despite that shift, the three words I hear most often on a responsive project—"mobile," "tablet," and "desktop"—are also the least helpful. They're not *bad* as such, but they're shorthand and often obscure the design challenges we face.

As a quick example, ask a colleague to describe what "mobile" means to them. Depending on who you ask, the term might suggest a small, touchscreen-enabled device, one that uses a slower cellular network to browse the web. But what if the user's device is connected to Wi-Fi? Alternately, "desktop" might suggest a widescreen device, perhaps with a mouse or trackpad. But what if their laptop's tethered to their phone's 3G connection? What if it also has a touch interface?

In other words, it's not just that we're designing for more device classes than ever before. Rather, the lines between "mobile," "tablet," and "desktop" are blurring: there are phones approaching the size of some smaller tablets (and vice versa); our sites can appear on web-enabled smartwatches, with view-

FIG 5.32: Responsive on your wrist: GOV.UK's lovely responsive site, as rendered on a Moto 360 smartwatch. Screenshots courtesy Anna Debenham.

FIG 5.33: Don't Digg and drive, kids: Tesla's Model S electric car has a touchscreen that, yes, has a WebKit-based browser. Photograph by Chris Martin (http://bkaprt.com/rdpp/05-31/).

INPUT METHOD	Touch	Keyboard/ Mouse	Hybrid	Speech	Joystick/ Analog
SCREEN SIZE	Small	Mid-Range	Wide		
NETWORK SPEED	Slow (EDGE/ GPRS)	Medium (3G)	Fast		
NETWORK CONDITION	Primarily Offline	Spotty, high latency	Reliable, stable		

FIG 5.34: Rather than discussing broad device classes, it's helpful to focus on specific features and conditions that might affect your responsive design.

ports roughly the same size as many smartphones (**FIG. 5.32**); heck, there are cars available today with browsers embedded in their dashboards (**FIG. 5.33**).

These are just a few reasons why I find it helpful to talk about *features,* not device classes. For example, I'll often talk with clients about how a responsive design performs across a few broad categories, usually focusing on **input method**, **screen size**, **network speed**, and **network condition** (**FIG. 5.34**).

It's not a comprehensive list, of course: an animation-heavy project might need a row for the quality of various devices' graphics processors, or perhaps you'll want finer-grained options in the network rows. And my row of input methods is often too brief, almost to a fault—perhaps you're designing for gestural interfaces, directional pad-driven devices like TV remotes or console controllers, or stylus-enabled screens.

But a table *like* this helps me decouple discussions of layout and screen size from, say, the quality of the user's network, or the input method she uses to interact with her device. Doing so helps avoid situations where it's assumed that every widescreen device is mouse-enabled, or that every small device is limited to a spotty 3G connection.

Much of this book has been about breaking down the page into its component parts: understanding how our small layout systems need to adapt, and then using those modules to gradually build more complex, responsive layout systems. As our layouts become more flexible and device-agnostic, the words we use to talk about our responsive designs should follow suit. Because when we're accounting for the conditions under which our responsive designs might be viewed—the myriad network conditions, input modes, and screen sizes—we need a design language that's as nimble and modular as our layout systems are becoming.

After all, these challenges aren't new. In a sense, we've been walking through this forest for some time, and we've got quite some distance to go yet. But as we walk, it's worth remembering there will always be trees around us, and we'll still manage to build beautiful, nimble, responsive designs.

Let's get started.