# What is Web 2.0

"the network as platform"

- Services, not packaged software, with cost-effective scalability
- Control over unique, hard-to-recreate data sources that get richer as more people use them
- Trusting users as co-developers
- Harnessing collective intelligence
- Leveraging the long tail through customer self-service
- Software above the level of a single device
- Lightweight user interfaces, development models, AND business models

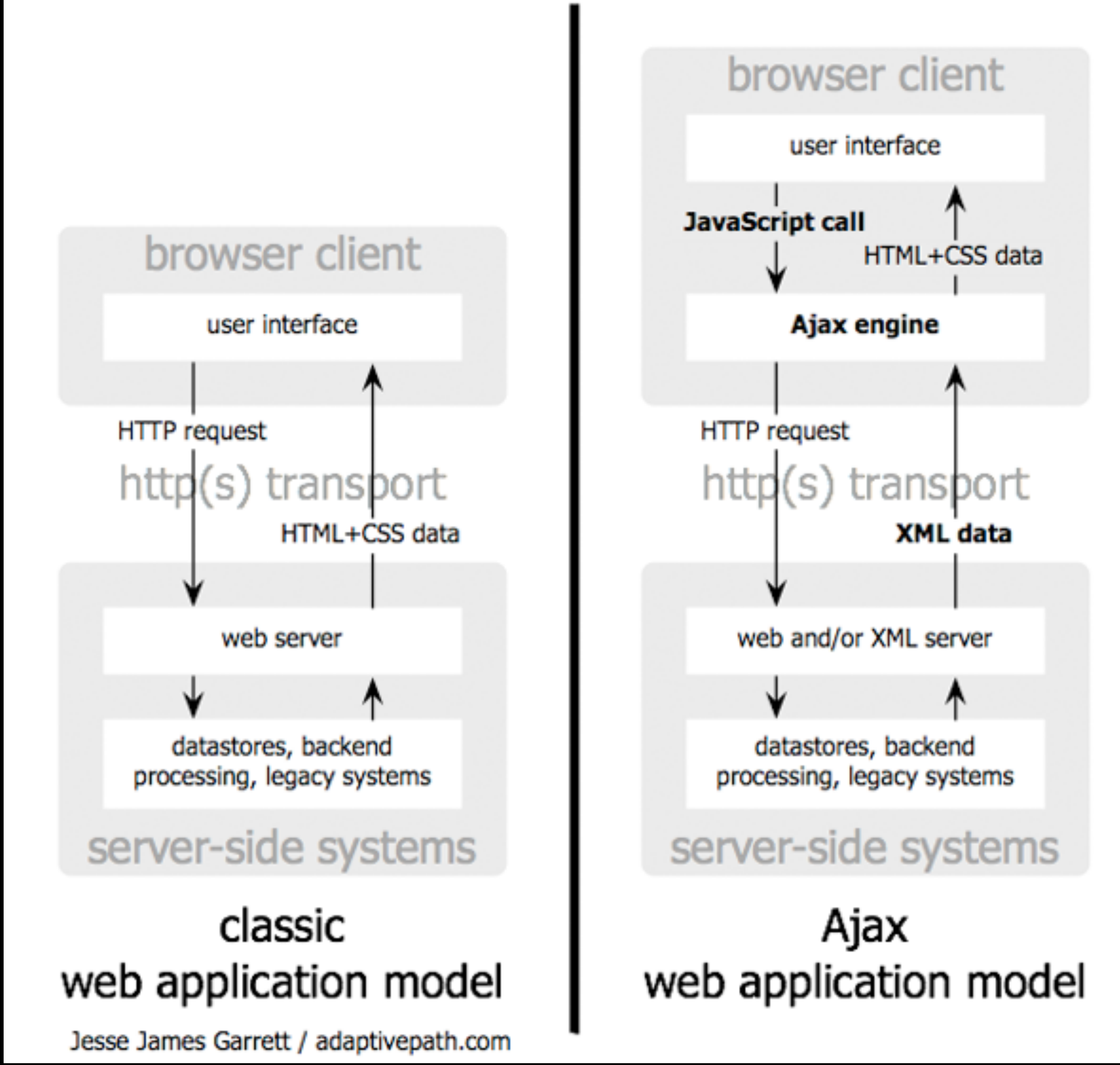
— **Tim O’Reilly**, 2005

The collection of technologies used by Google was christened **AJAX**, in a seminal essay by Jesse James Garrett of web design firm Adaptive Path.

"**Ajax** isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways.

Ajax incorporates:

- standards-based presentation using **XHTML** and **CSS**;
- dynamic display and interaction using the Document Object Model (DOM);
- data interchange and manipulation using XML and XSLT;
- asynchronous data retrieval using XMLHttpRequest;
- + **JavaScript** binding everything together."



Jesse James Garrett, 2005

(To view this link you'll need the Wayback Machine)

# Web Squared

“search as vote”

Modern search engines now use complex algorithms and hundreds of different ranking criteria to produce their results. Among the data sources is the feedback loop generated by the frequency of search terms, the number of user clicks on search results, and our own personal search and browsing history. For example, if a majority of users start clicking on the fifth item on a particular search results page more often than the first, Google's algorithms take this as a signal that the fifth result may well be better than the first, and eventually adjust the results accordingly.

“Our devices extend us, and we extend them.”

# rwd
# Responsive Web Design

Most of these notes are verbatim txt  from: Learning Web Design - Jennifer Niederst Robbins

## Layout

Rearranging content into different layouts may be the first thing you think of when you picture responsive design, and with good reason. The layout helps form our first impression of a site's content and usability.

Responsive design is based on fluid layouts that expand and contract to fill the available space in the viewport. One **fluid layout** is usually not enough, however, to serve all screen sizes. More often, two or three layouts are produced to meet requirements across devices, with small adjustments between layout shifts.

Designers typically start with a one-column layout that fits well on small handheld devices and rearrange elements into columns as more space is available. They may also have the design for the widest screens worked out early on so there is an end-point in mind. The design process may involve a certain amount of switching between views and making decisions about what happens along the way.

## Responsive layout patterns

The manner in which a site transitions from a small-screen layout to a wide-screen layout must make sense for that particular site, but there are a few patterns (common and repeated approaches) that have emerged over the years. We can thank Luke Wroblewski (known for his “Mobile First” approach to web design, which has become the standard) for doing a survey of how responsive sites handle layout. Following are the top patterns Luke named in his article:

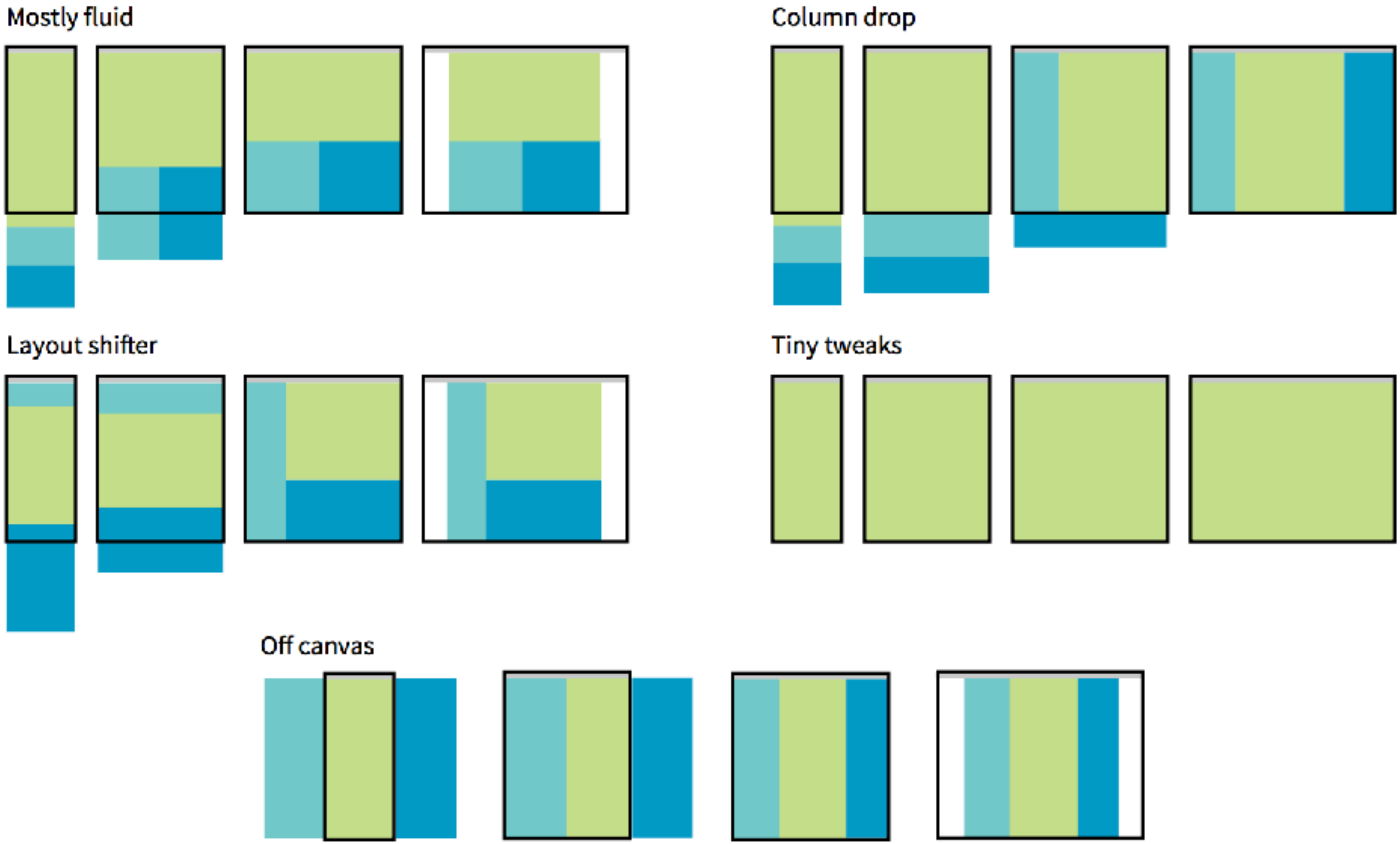


**FIGURE 17-9.** Examples of the responsive layout patterns identified by Luke Wroblewski.

## Mostly fluid

This pattern uses a single-column layout for small screens, and another fluid layout that covers medium and large screens, with a maximum width set to prevent it from becoming too wide. It generally requires less work than other solutions.

## Column drop

This solution shifts between one-, two-, and three-column layouts based on available space. When there isn’t room for extra columns, the sidebar columns drop below the other columns until everything is stacked verti- cally in the one-column view.

## Layout shifter

If you want to get really fancy, you can completely reinvent the layout for a variety of screen sizes. Although expressive and potentially cool, it is not necessary. In general, you can solve the problem of fitting your content to multiple environments without going overboard.

## Tiny tweaks

Some sites use a single-column layout and make tweaks to type, spacing, and images to make it work across a range of device sizes.

## Off canvas

As an alternative to stacking content vertically on small screens, you may choose to use an “off-canvas” solution. In this pattern, a page component is located just out of sight on the left or right of the screen and flies into view when requested. A bit of the main content screen remains visible on the edge to orient users as to the relationship of moving parts. This was made popular by Facebook, wherein Favorites and Settings were placed on a panel that slid in from the left when users clicked a menu icon

# Navigation

Navigation feels a little like the Holy Grail of Responsive Web Design. It is critical to get it right. Because navigation at desktop widths has pretty much been conquered, the real challenges come in re-creating our navigation options on small screens. A number of successful patterns have emerged for small screens, which I will briefly summarize here

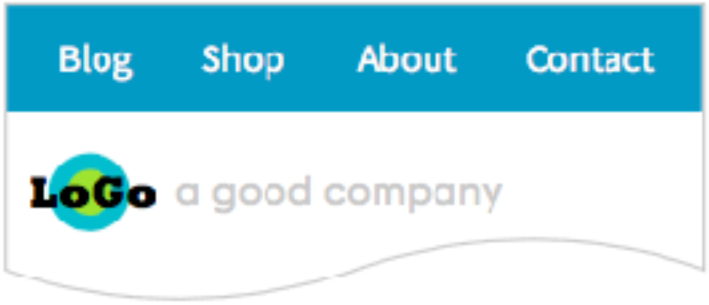
## Top navigation

If your site has just a few navigation links, they may fit just fine in one or two rows at the top of the screen.

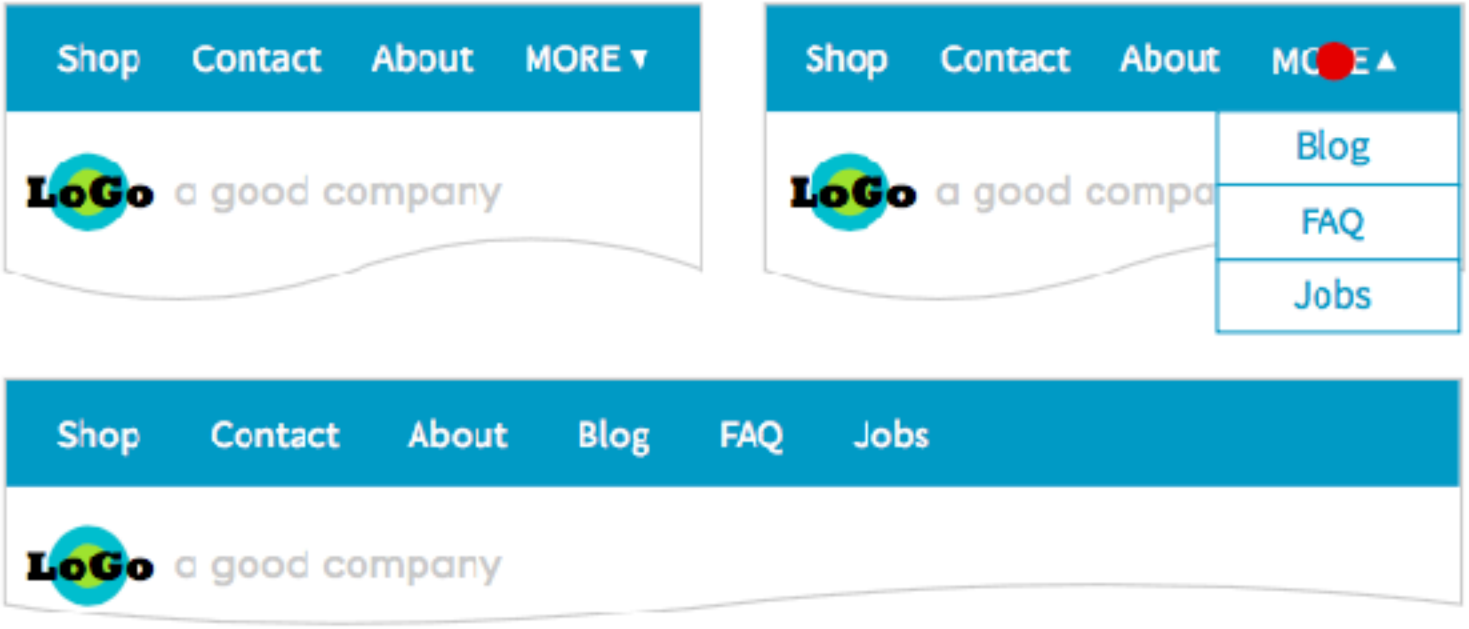
## Priority +

In this pattern, the most important navigation links appear in a line across the top of the screen alongside a More link that exposes additional options. The pros are that the primary links are in plain view, and the number of links shown can increase as the device width increases. The cons include the difficulty of determining which links are worthy of the prime small-screen real estate.

Top navigation



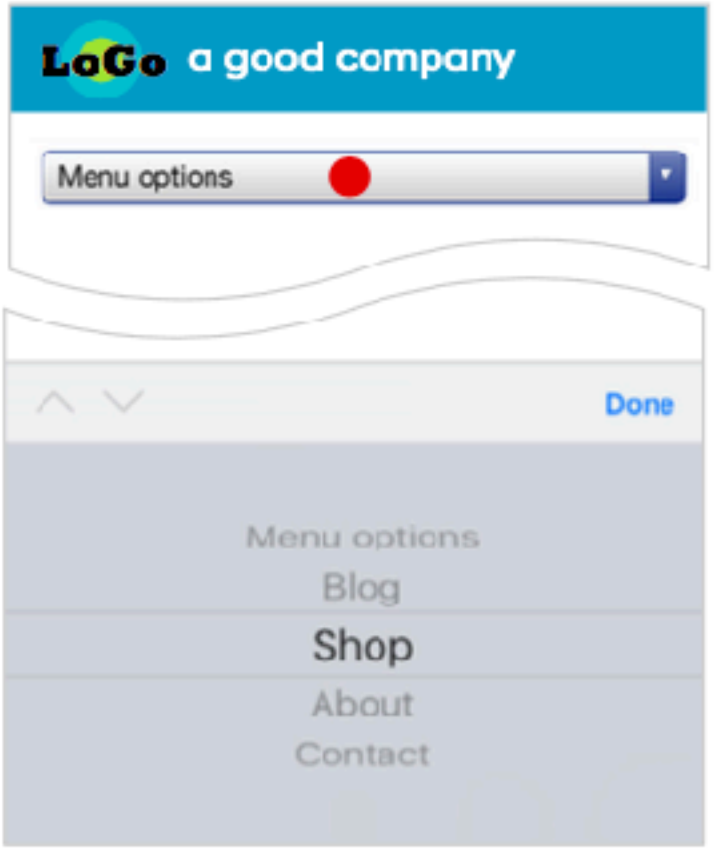
Priority +



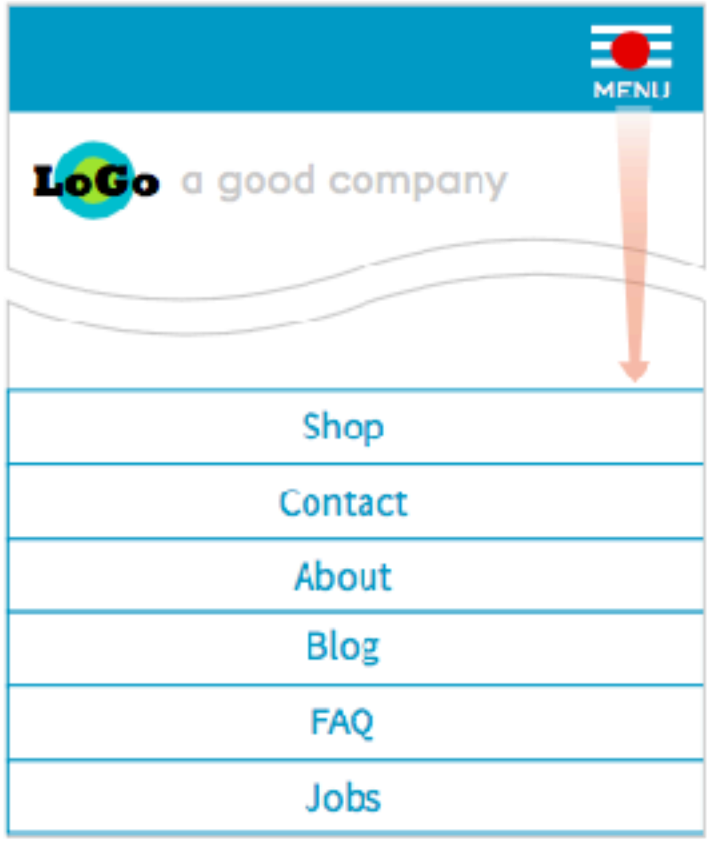
KEY

● = tap

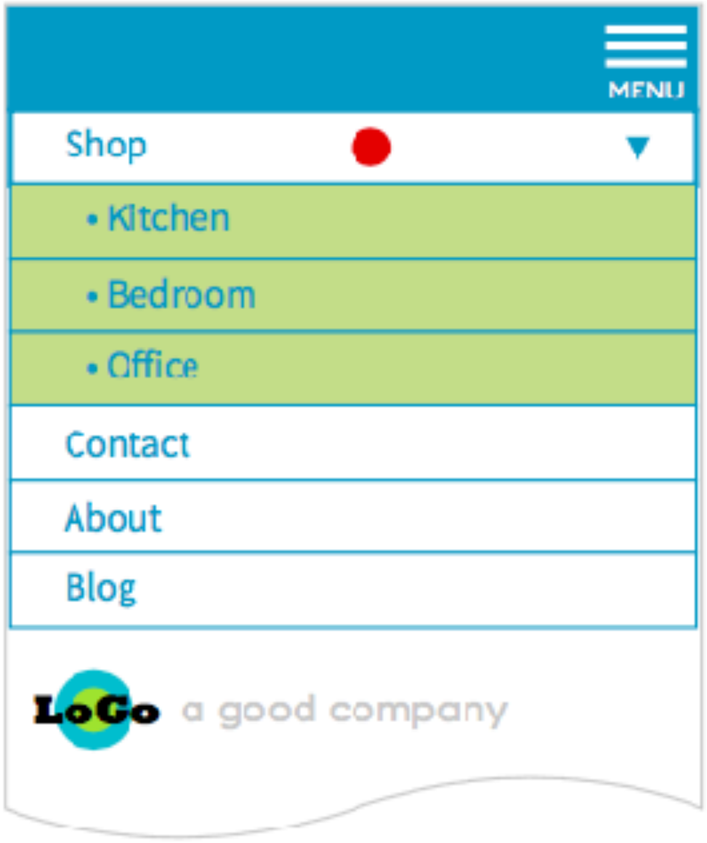
Select menu



Link to footer menu



Accordion sub-navigation



## Select menu

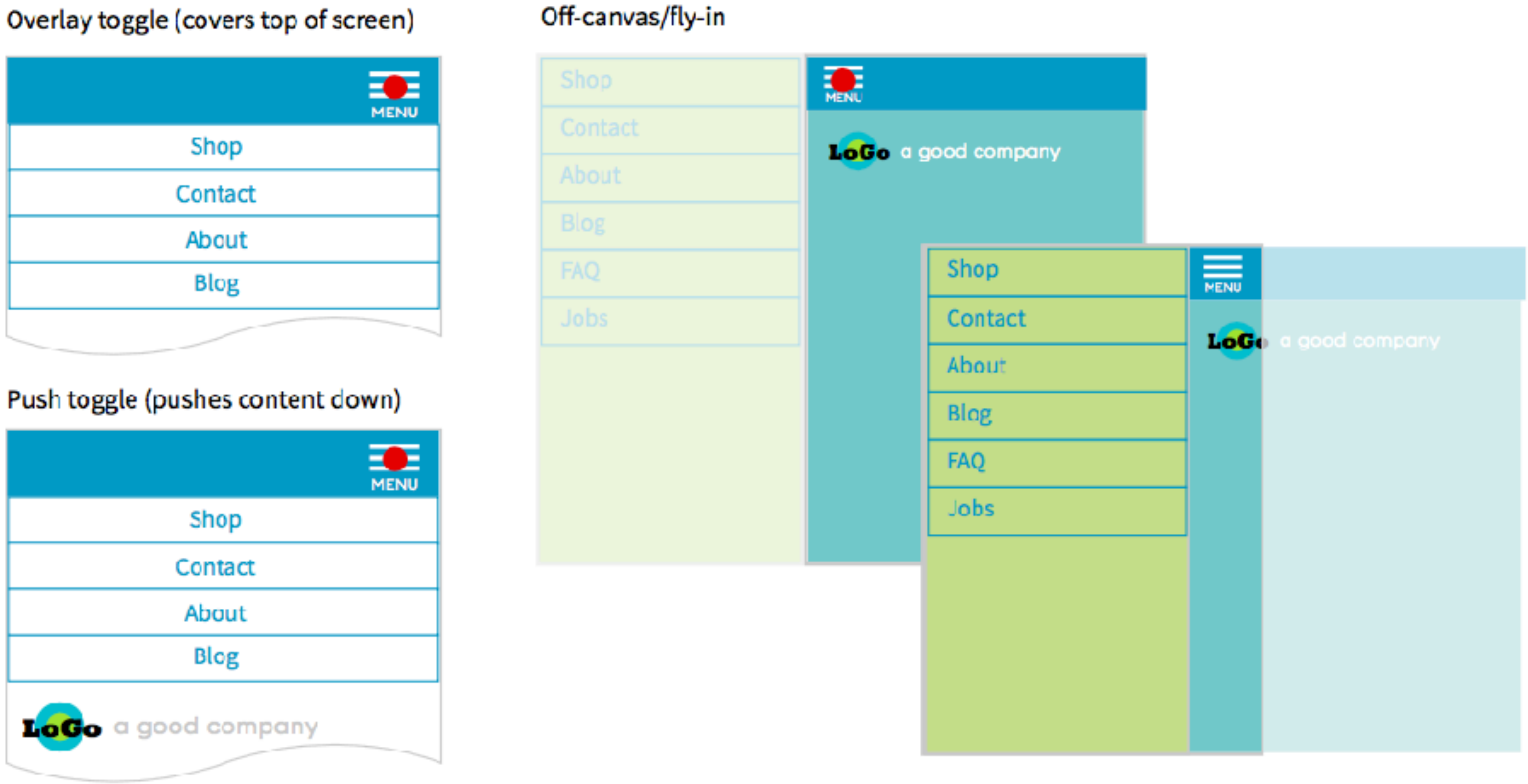
For a medium list of links, some sites use a select input form element. Tapping the menu opens the list of options using the select menu UI of the operating system, such as a scrolling list of links at the bottom of the screen or on an overlay. The advantage is that it is compact, but on the downside, forms aren’t typically used for navigation, and the menu may be overlooked.

## Link to footer menu

One straightforward approach places a Menu link at the top of the page that links to the full navigation located at the bottom of the page. The risk with this pattern is that it may be disorienting to users who suddenly find themselves at the bottom of the scroll.

## Accordion sub-navigation

When there are a lot of navigation choices with sub-navigation menus, the small-screen solution becomes more challenging, particularly when you can’t hover to get more options as you can with a mouse. Accordions that expand when you tap a small arrow icon are commonly used to reveal and hide sub-navigation. They may even be nested several levels deep. To avoid nesting navigation in accordion submenus, some sites simply link to separate landing pages that contain a list of the sub-navigation for that section.



Overlay toggle (covers top of screen)

Push toggle (pushes content down)

Off-canvas/fly-in

**FIGURE 17-11.** Responsive navigation patterns.

## Display Property

The display property specifies if/how an element is displayed. Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is block or inline. A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can). An inline element does not start on a new line and only takes up as much width as necessary.

```
	display: none;
```

commonly used with JavaScript to hide/show elements without deleting and recreating them.

## Overriding Default Display

Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow the web standards.

```
li {
    display: inline;
}

span {
    display: block;
}
```

Note: Setting the display property of an element only changes how the element is displayed, NOT what kind of element it is. So, an inline element with display: block; is not allowed to have other block elements inside it.

## Overflow Property

The CSS overflow property controls what happens to content that is too big to fit into an area. The overflow property specifies whether to clip content or to add scrollbars when the content of an element is too big to fit in a specified area. The overflow property only works for block elements with a specified height. The overflow property has the following values:

- **visible** - Default. The overflow is not clipped. It renders outside the element's box
- **hidden** - The overflow is clipped, and the rest of the content will be invisible
- **scroll** - The overflow is clipped, but a scrollbar is added to see the rest of the content
- **auto** - If overflow is clipped, a scrollbar should be added to see the rest of the content

# Properties for left and right

**overflow-x**

specifies what to do with the left/right edges of the content.

**overflow-y**

specifies what to do with the top/bottom edges of the content.

```
div.theExample1 {
    background-color: lightblue;
    height: 40px;
    width: 200px;
    overflow-y: scroll;
}

div.theExample2 {
    background-color: lightblue;
    height: 40px;
    width: 200px;
    overflow-y: hidden;
}
```

**code from W3**

# Media Queries

the @media rule tells the browser to include a block of CSS properties only if a certain condition is true.

So this:

```
@media only screen and (max-width: 500px)  {
    body {
      background-color: light blue;
    }
}
```

Translates to:

```
if (the maximum width of the web page is 500 pixels) {
                then do this stuff
}
```

# Breakpoint

```
/* For mobile phones: */
[class*="col-"] {
    width: 100%;
}
@media only screen and (min-width: 768px) {
    /* For desktop: */
    .col-1 {width: 8.33%;}
    .col-2 {width: 16.66%;}
    .col-3 {width: 25%;}
    .col-4 {width: 33.33%;}
    .col-5 {width: 41.66%;}
    .col-6 {width: 50%;}
    .col-7 {width: 58.33%;}
    .col-8 {width: 66.66%;}
    .col-9 {width: 75%;}
    .col-10 {width: 83.33%;}
    .col-11 {width: 91.66%;}
    .col-12 {width: 100%;
}
```

add a **breakpoint** where certain parts of the design will behave differently on each side of the breakpoint

many examples: https://www.w3schools.com/Css/css_rwd_mediaqueries.asp

## Flexbox & CSS Grid

By Flexbox and CSS Grid are designed to be responsive and eliminate the need for media queries. However, they both offer many options as to the design while remaining responsive.

- Example of Responsive Image Gallery
- CSS Flexbox
- CSS Grid + Responsive Layout

# Flexbox & CSS Grid

"The basic difference between CSS Grid Layout and CSS Flexbox Layout is that flexbox was designed for layout in one dimension - either a row or a column. Grid was designed for two-dimensional layout - rows, and columns at the same time. The two specifications share some common features, however, and if you have already learned how to use flexbox, the similarities should help you get to grips with Grid."

**MDN**

The flex-direction property defines in which direction the container wants to stack the flex items - either flex-direction: row or flex-direction: column. However, by using flex-wrap property. Read all about **CSS Flexbox @ W3**.

## CSS Grid

A grid is an intersecting set of horizontal and vertical lines - one set defining columns and the other rows. Elements can be placed onto the grid, respecting these column and row lines.

## How Grid Layout Works

The process for using the CSS Grid Layout Module is fundamentally simple:

+ Use the display property to turn an element into a grid container. The element's children automatically become grid items.
+ Set up the columns and rows for the grid. You can set them up explicitly and/or provide directions for how rows and columns should get created on the fly (the css grid is very flexible).
+ Assign each grid item to an area on the grid. If you don't assign them explicitly, they flow into the cells sequentially.

The element that has the display: **grid property** applied to it becomes the grid container and defines the context for grid formatting. All of its direct child elements automatically become grid items that end up positioned in the grid. You can define an explicit grid with grid layout but the specification also deals with the content added outside of a declared grid, which adds additional rows and columns when needed. Features such as adding "as many columns that will fit into a container" are included.

### Grid line

The horizontal and vertical dividing lines of the grid are called grid lines.

### Grid cell

The smallest unit of a grid is a grid cell, which is bordered by four adjacent grid lines with no grid lines running through it.

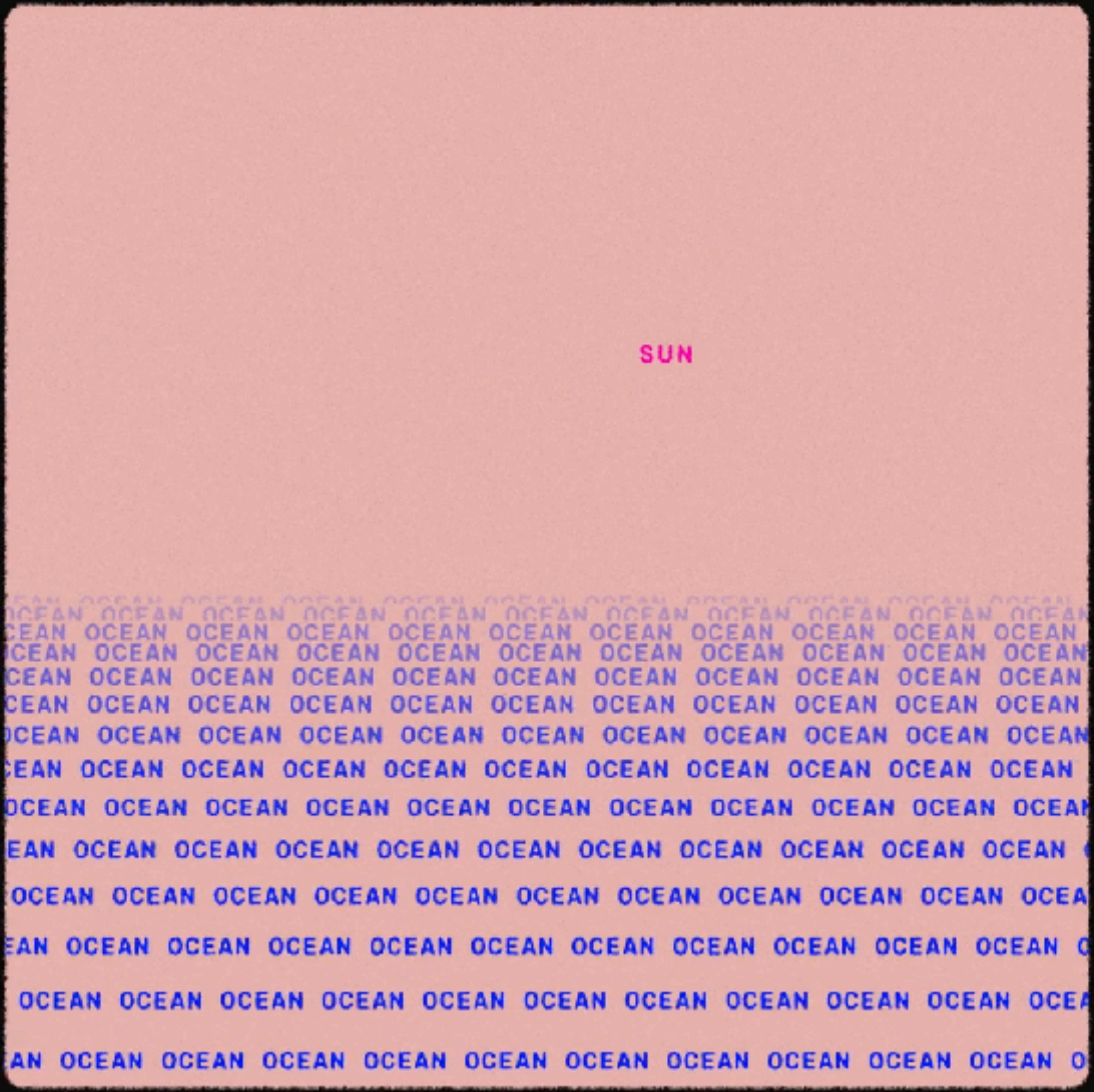
### Grid area

A grid area is a rectangular area made up of one or more adjacent grid cells.

### Grid track

The space between two adjacent grid lines is a grid track, which is a generic name for a grid column or a grid row. Grid columns are said to go along the block axis, which is vertical (as block elements are stacked) for languages written horizontally. Grid rows follow the inline (horizontal) axis.

The structure established for the grid is independent from the number of grid items in the container. You could place 4 grid items in a grid with 12 cells, leaving 8 of the cells as 'whitespace.' That's the flexibility of grids. You can also set up a grid with fewer cells than grid items, and the browser adds cells to the grid to accommodate them.



**Words**

Enle Li + Liz Xiong

# CSS TRANSITIONS

CSS Transitions smooth out otherwise abrupt changes to property values between two states over time by filling in the frames in between. Animators call that tweening. When used with reserve, CSS Transitions can add sophistication and polish to your interfaces and even improve usability.

When applying a transition, you have a few decisions to make, each of which is set with a CSS property:

- Which CSS property to change (**transition-property**) (Required)
- How long it should take (**transition-duration**) (Required)
- The manner in which the transition accelerates (**transition-timing- function**)
- Whether there should be a pause before it starts (**transition-delay**)

Transitions require a **beginning state** and an **end state**. The element as it appears when it first loads is the beginning state. The end state needs to be triggered by a state change such as :hover, :focus, or :active…

## CSS Animation Selectors

: hover

: focus

: active

## transition-property

identifies the CSS property that is changing and that you want to transition smoothly. In our example, it’s the background-color. You can also change the foreground color, borders, dimensions, font- and text-related attributes, and many more. TABLE 18-1 lists the animatab CSS properties as of this writing. The general rule is that if its value is a color, length, or number, that property can be a transition property.

## Backgrounds

background-color
background-position

## Borders and outlines

border-bottom-color
border-bottom-width
border-left-color
border-left-width
border-right-color
border-right-width
border-top-color
border-top-width
border-spacing
outline-color
outline-width

## Color and opacity

color
opacity
visibility

## Font and text

font-size
font-weight
letter-spacing
line-height
text-indent
text-shadow
word-spacing
vertical-align

## Position

top
right
bottom
left
z-index
clip-path

## Transforms

transform
transform-origin

## Element box measurements

height
width
max-height
max-width
min-height
min-width
margin-bottom
margin-left
margin-top
padding-bottom
padding-left
padding-right
padding-top

## Animateable CSS Properties

## Timing Functions

```
. thisAwesomeClass {
        transition-timing-function :

                ease

                linear

                ease-in

                ease-out

                ease-in-out
                step-start

                step-end

                steps

                cubic-bezier(#,#,#,#)


    }
```

the css property

possible values you can set

The **property** and the **duration** are required and form the foundation of a transition, but you can refine it further. There are a number of ways a **transition** can roll out over time.

For example, it could start out fast and then slow down, start out slow and speed up, or stay the same speed all the way through, just to name a few possibilities. I think of it as the transition “style,” but in the spec, it is known as the timing function or easing function.

The timing function you choose can have a big impact on the feel and believability of the animation, so if you plan on using transitions and CSS animations, it is a good idea to get familiar with the options.

## ease

Starts slowly, accelerates quickly, and then slows down at the end. This is the default value and works just fine for most short transitions.

## linear

Stays consistent from the transition's beginning to end. Because it is so consistent, some say it has a mechanical feeling.

## ease-in

Starts slowly, then speeds up.

## ease-out

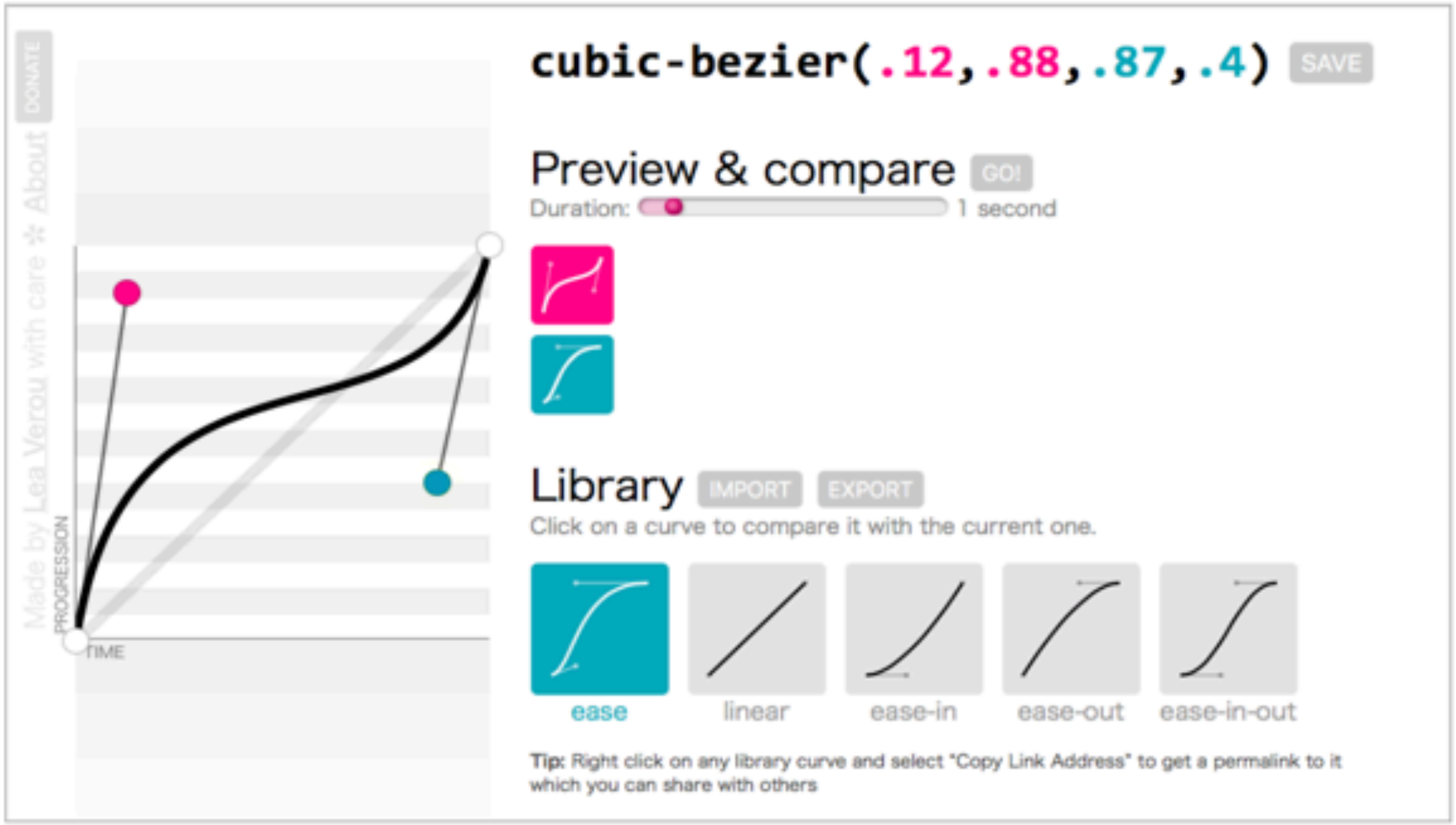
Starts out fast, then slows down.

## ease-in-out

Starts slowly, speeds up, and then slows down again at the very end. It is similar to ease, but with less pronounced acceleration in the middle.

# cubic-bezier(x1,y1,x2,y2)

The acceleration of a transition can be plotted with a curve called a Bezier curve. The steep parts of the curve indicate a fast rate of change, and the flat parts indicate a slow rate of change.



```
cubic-bezier(.12,.88,.87,.4)
```

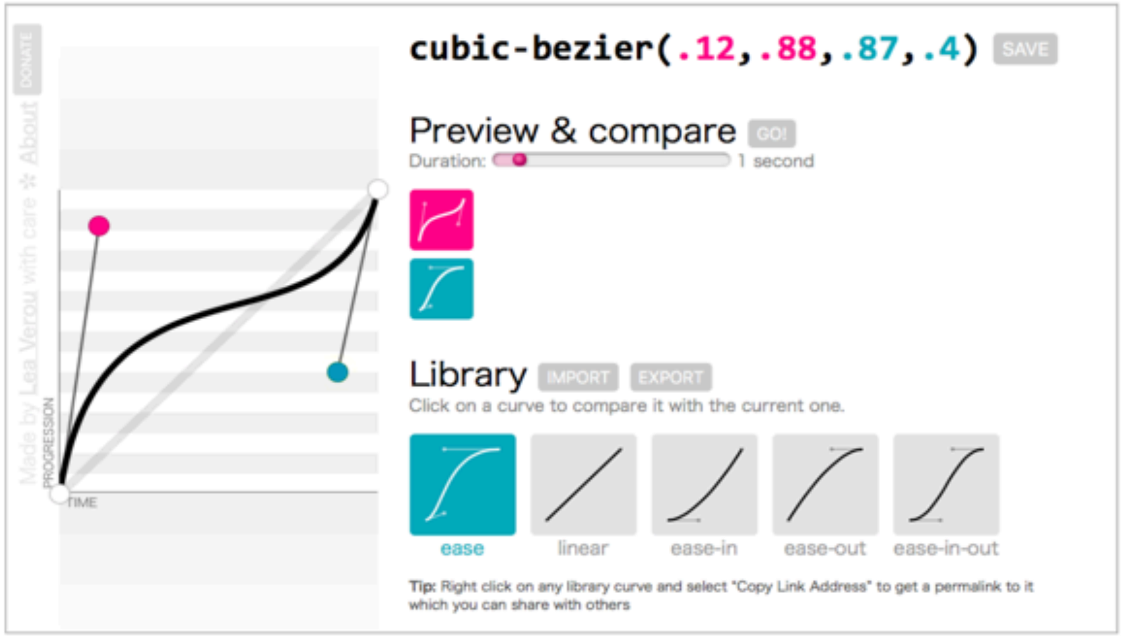
Click on a curve to compare it with the current one.

Tip: Right click on any library curve and select "Copy Link Address" to get a permalink to it which you can share with others

FIGURE 18-2. Examples of Bezier curves from Cubic-Bezier.com. On the left is my custom curve that starts fast, slows down, and ends fast.

You can see that the ease curve is a tiny bit flat in the beginning, gets very steep (fast), then ends flat (slow). The linear keyword, on the other hand, moves at a consistent rate for the whole transition.

You can get the feel of your animation just right by creating a custom curve. The site Cubic-Bezier.com is a great tool for playing around with transition timing and generating the resulting code. The four numbers in the value represent the x and y positions of the start and end Bezier curve handles (the pink and blue dots.



```
cubic-bezier(.12,.88,.87,.4)
```

Duration: 1 second

Click on a curve to compare it with the current one.

Tip: Right click on any library curve and select "Copy Link Address" to get a permalink to it which you can share with others

FIGURE 18-2. Examples of Bezier curves from Cubic-Bezier.com. On the left is my custom curve that starts fast, slows down, and ends fast.

### steps(#, start|end)

Divides the transitions into a number of steps as defined by a stepping function. The first is the number of steps, and the **start** and **end** keywords define whether the change in sta happens at the begin- ning (**start**) or end of each step. Step animation is especially useful keyframe animation with sprite images. For a better explanation and examples, I recomm the article “Using Multi-Step Animations and Transitions,” by Geoff Graham on CSS-Tricks (*css-tricks.com/using-multi- step-animations-transitions/*).

### step-start

Changes states in one step, at the beginning of the duration time (the same as **steps(1,st** The result is a sudden state change, the same as if no transition had been applied at all.

### step-end

Changes states in one step, at the end of the duration time (the same as **steps(1,end)**).

## transition-delay

The transition-delay property, as you might guess, delays the start of the animation by a specified amount of time.

## The Shorthand transition Property

The authors of the CSS3 spec had the good sense to give us the shorthand transition property to combine all of these properties into one declaration.You've seen this sort of thing with the shorthand border property. Here is the syntax:

```
transition: property duration timing-function delay;
```

```
.theClass {
    transition: background-color 0.3s ease-in-out 0.2s;
    }
```

The values for each of the **transition-*** properties are listed out, separated by character spaces. The order isn't important as long as the **duration** (which is required) appears before **delay** (which is optional). If you provide only one time value, it will be assumed to be the duration.

The sub-properties of the **animation** property are:

**animation-delay**
Configures the delay between the time the element is loaded and the beginning of the animation sequence.

**animation-direction**
Configures whether or not the animation should alternate direction on each run through the sequence or reset to the start point and repeat itself.

**animation-duration**
Configures the length of time that an animation should take to complete one cycle.

**animation-iteration-count**
Configures the number of times the animation should repeat; you can specify infinite to repeat the animation indefinitely.

**animation-name**
Specifies the name of the **@keyframes** at-rule describing the animation’s keyframes.

**animation-play-state**

Lets you pause and resume the animation sequence.

**animation-timing-function**

Configures the timing of the animation; that is, how the animation transitions through keyframes, by establishing acceleration curves.

**animation-fill-mode**

Configures what values are applied by the animation before and after it is executing.

@keyframes + animation property