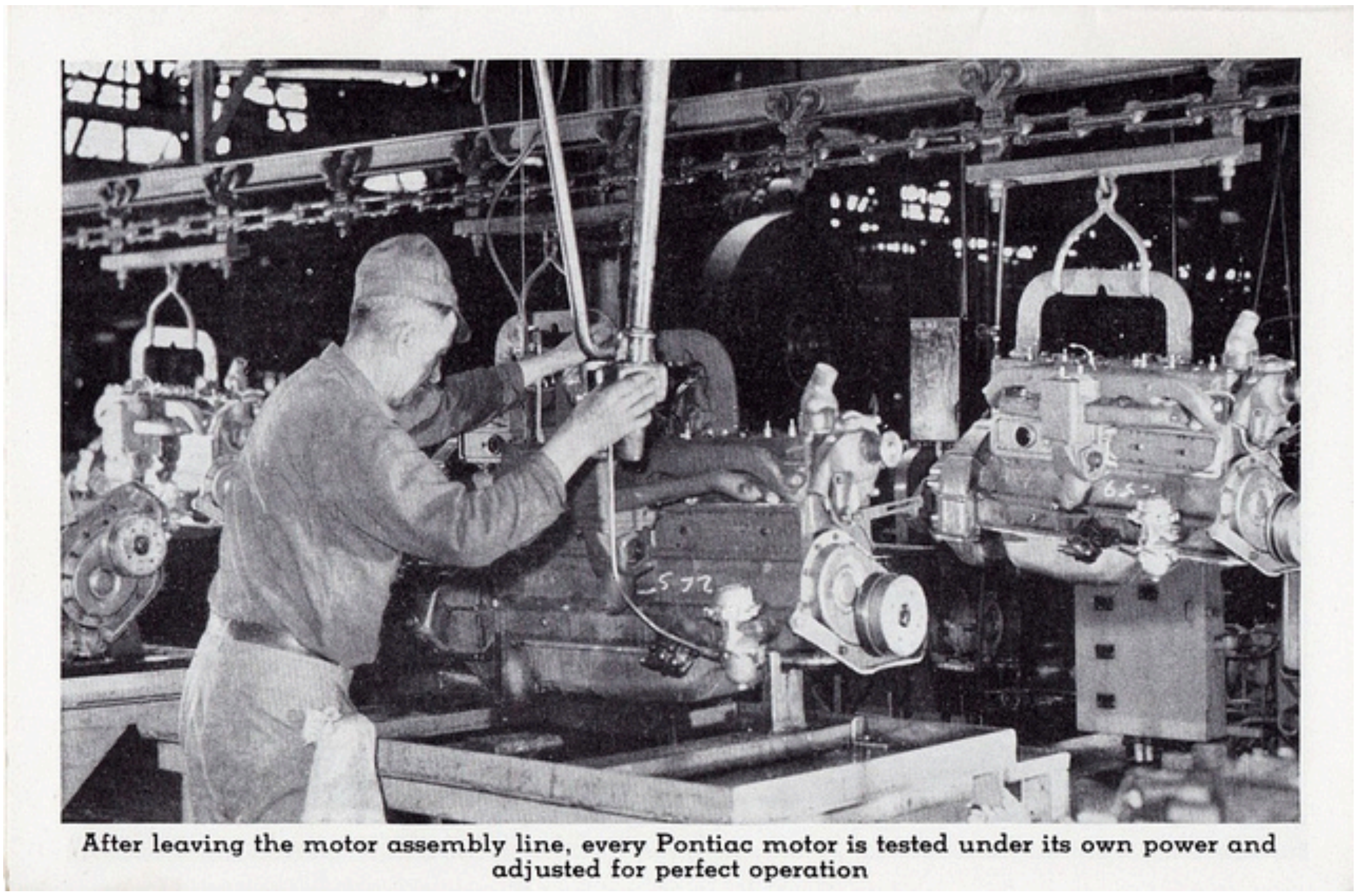
Modular CSS, JavaScript, and WordPress

Josh Williams
http://tucsonlabs.com
11 / 7 / 2012

Hi I'm Josh. I run tucsonlabs.com and we make awesome, WordPress powered websites.


After leaving the motor assembly line, every Pontiac motor is tested under its own power and adjusted for perfect operation

modular development

Separate an application into smaller parts that can be developed and deployed independently.

Image Credit: http://www.flickr.com/photos/autohistorian/7599294098/

The concept of modular development stems from traditional programing and the basic idea is to separate an application into smaller parts. As our websites become more complex, it can be advantageous to adopt a similar approach to managing our CSS and Javascript.

Advantages

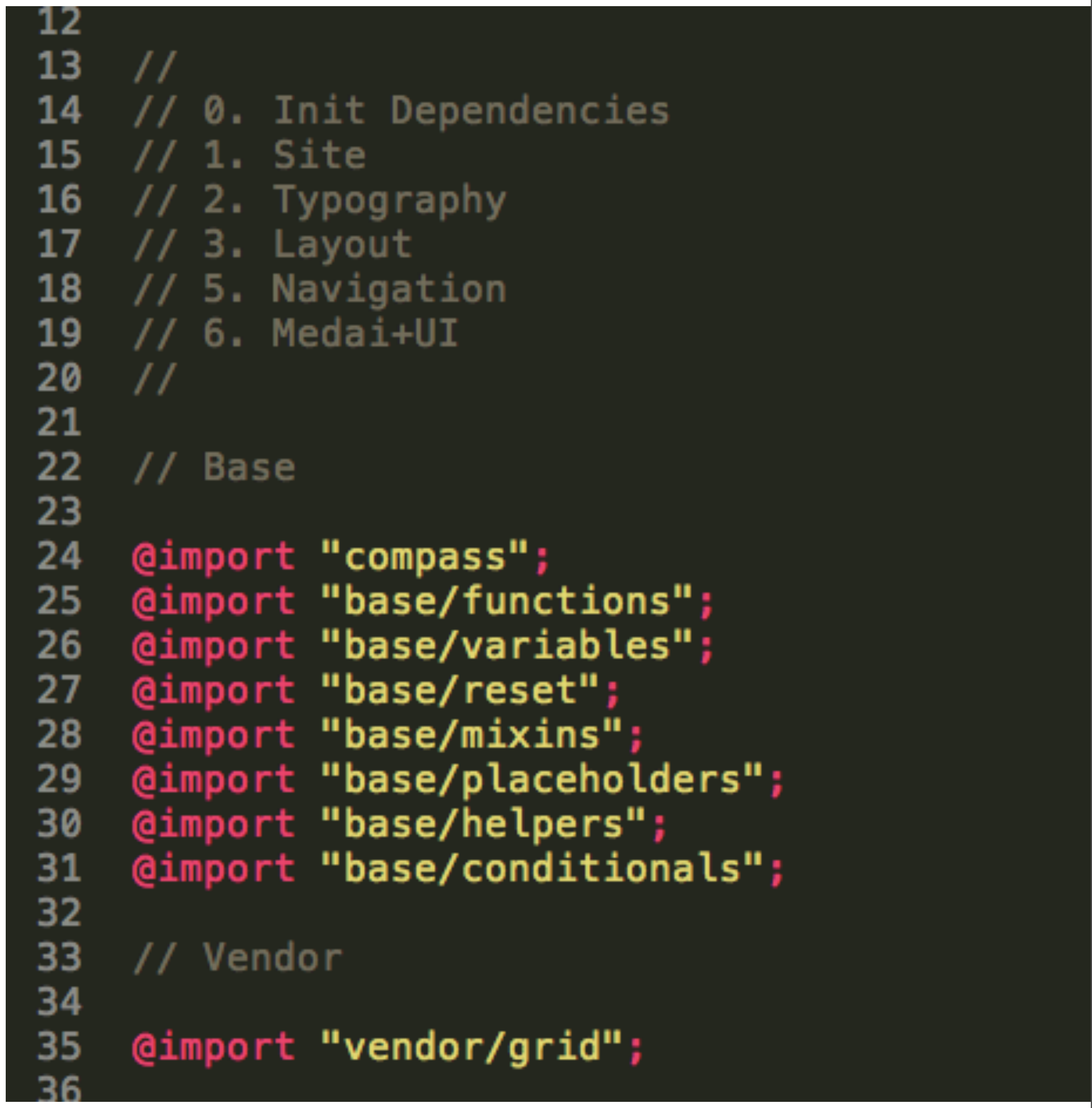
- smaller files
- easier to debug, maintain, and scale
- developers can work separately on different parts of an application

Credit: http://www.flickr.com/photos/vegaseddie/5700609302/in/photostream/

Taking a modular approach to front-end development forces you into separating your CSS and JavaScript into distinct parts and this is beneficial for many different reasons. No one likes editing someone else's spaghetti code and there's a better place for your styles than at the end of a huge style sheet.

applying to your CSS workflow

- Organize and separate your CSS files and call them using @import
- Use a build tool (or a preprocessor) to concatenate your @import declarations



```
//
// 0. Init Dependencies
// 1. Site
// 2. Typography
// 3. Layout
// 5. Navigation
// 6. Medai+UI
//

// Base

@import "compass";
@import "base/functions";
@import "base/variables";
@import "base/reset";
@import "base/mixins";
@import "base/placeholders";
@import "base/helpers";
@import "base/conditionals";

// Vendor

@import "vendor/grid";
```

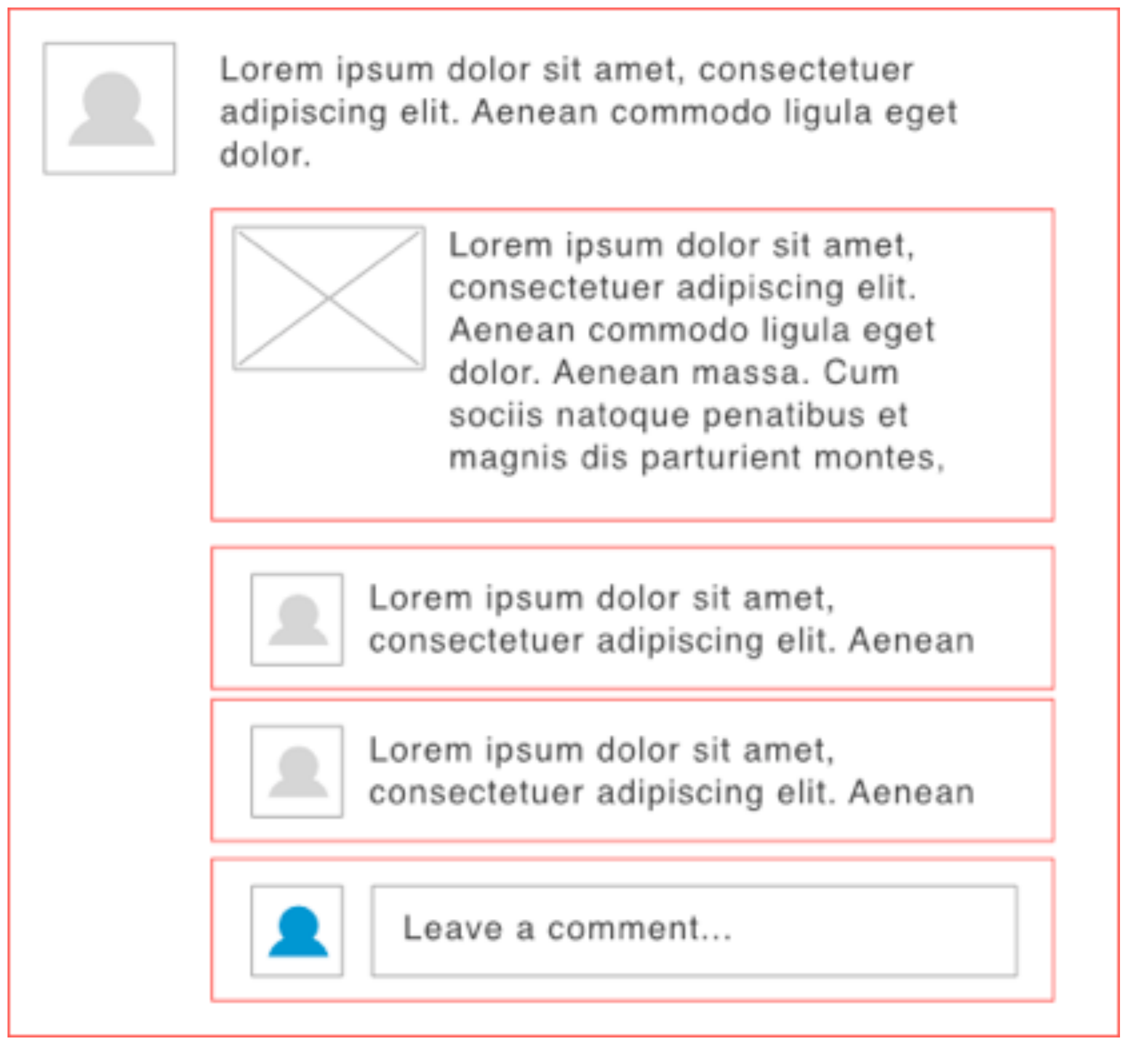
Thursday, November 8, 12

The first step to applying a modular approach to CSS is to separate your styles into their own files and use @import to include them in your main stylesheet. You can use a build tool to concatenate your files to improve performance, or use a CSS preprocess which will do this for you.

set fall back variables

```
$bodyBackgound: #fff !default;

html { background: $bodyBackground }
```

I want some of my modules to be portable.. like stick into someone else's project portable and not break. If you want to take this approach (and you're using SASS or LESS), you can make your modules more portable by using fallbacks for variables so they still work if a variable hasn't already been defined.

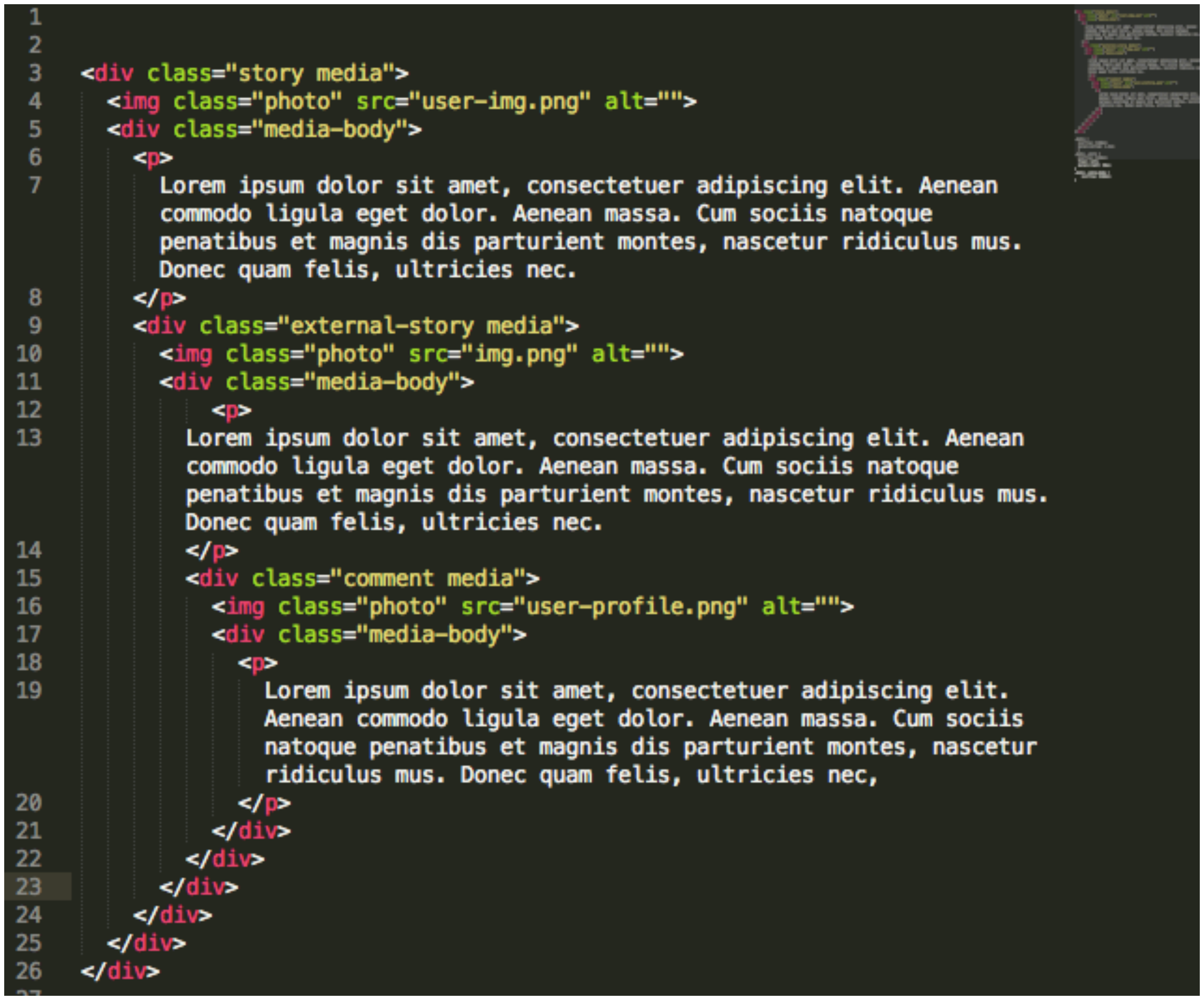

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor.
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes,
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
Leave a comment...

OOCSS

Combine these techniques with a modular approach.

I prefer to keep html as semantic as possible, and with the latest version of SASS 3.2, this is actually pretty simple. Looking at an object oriented CSS example, we can see how to make this object without adding non-semantic classes to our markup.

If you're not familiar with OOCSS concepts, check them out: http://oocss.org/

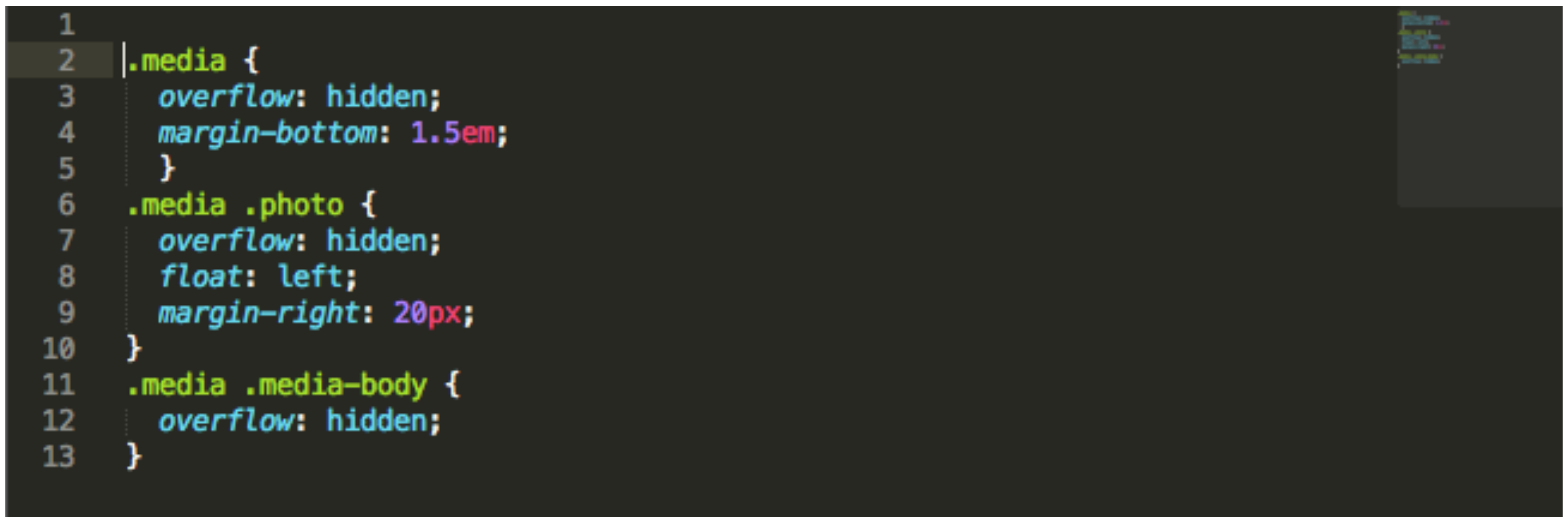


```

<div class="story media">
  <img class="photo" src="user-img.png" alt="">
  <div class="media-body">
    <p>
      Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
      commodo ligula eget dolor. Aenean massa. Cum sociis natoque
      penatibus et magnis dis parturient montes, nascetur ridiculus mus.
      Donec quam felis, ultricies nec.
    </p>
    <div class="external-story media">
      <img class="photo" src="img.png" alt="">
      <div class="media-body">
          <p>
        Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean
        commodo ligula eget dolor. Aenean massa. Cum sociis natoque
        penatibus et magnis dis parturient montes, nascetur ridiculus mus.
        Donec quam felis, ultricies nec.
        </p>
        <div class="comment media">
          <img class="photo" src="user-profile.png" alt="">
          <div class="media-body">
            <p>
              Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
              Aenean commodo ligula eget dolor. Aenean massa. Cum sociis
              natoque penatibus et magnis dis parturient montes, nascetur
              ridiculus mus. Donec quam felis, ultricies nec,
            </p>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

the markup

Thursday, November 8, 12

Looking at the mark up each media object, needs a containing div with a media class applied to it. If we're using standard CSS, we also need a class applied to the image (or video) and the body text. The CSS would look like this...



```

.media {
  overflow: hidden;
  margin-bottom: 1.5em;
  }
.media .photo {
  overflow: hidden;
  float: left;
  margin-right: 20px;
}
.media .media-body {
  overflow: hidden;
}
```

the CSS

Pretty simple. Now we can apply those classes to our markup anywhere we need to display this type of content. But there are draw backs...

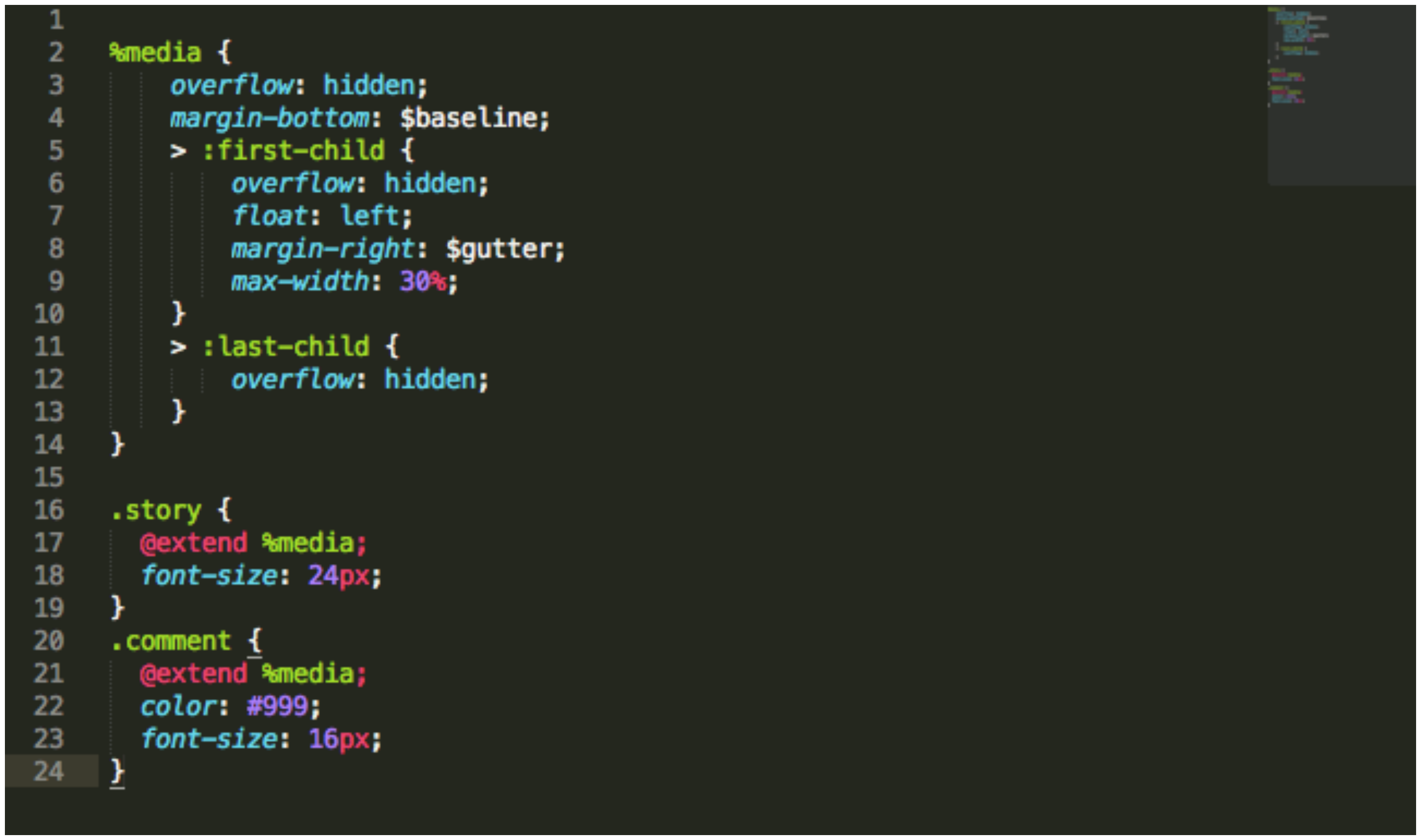
non-semantic markup

just because you can doesn't mean you should.

http://www.flickr.com/photos/izatrini/4495686498/sizes/z/in/photostream/

Thursday, November 8, 12

Taking this approach leads to a boatload of markup that doesn't mean anything. At some point as web developer, you need to make trade offs, get the job done, and add a non-semantic class to the markup so that you'll have more maintainable style sheets. However, using tools like SASS or LESS, we have other options.



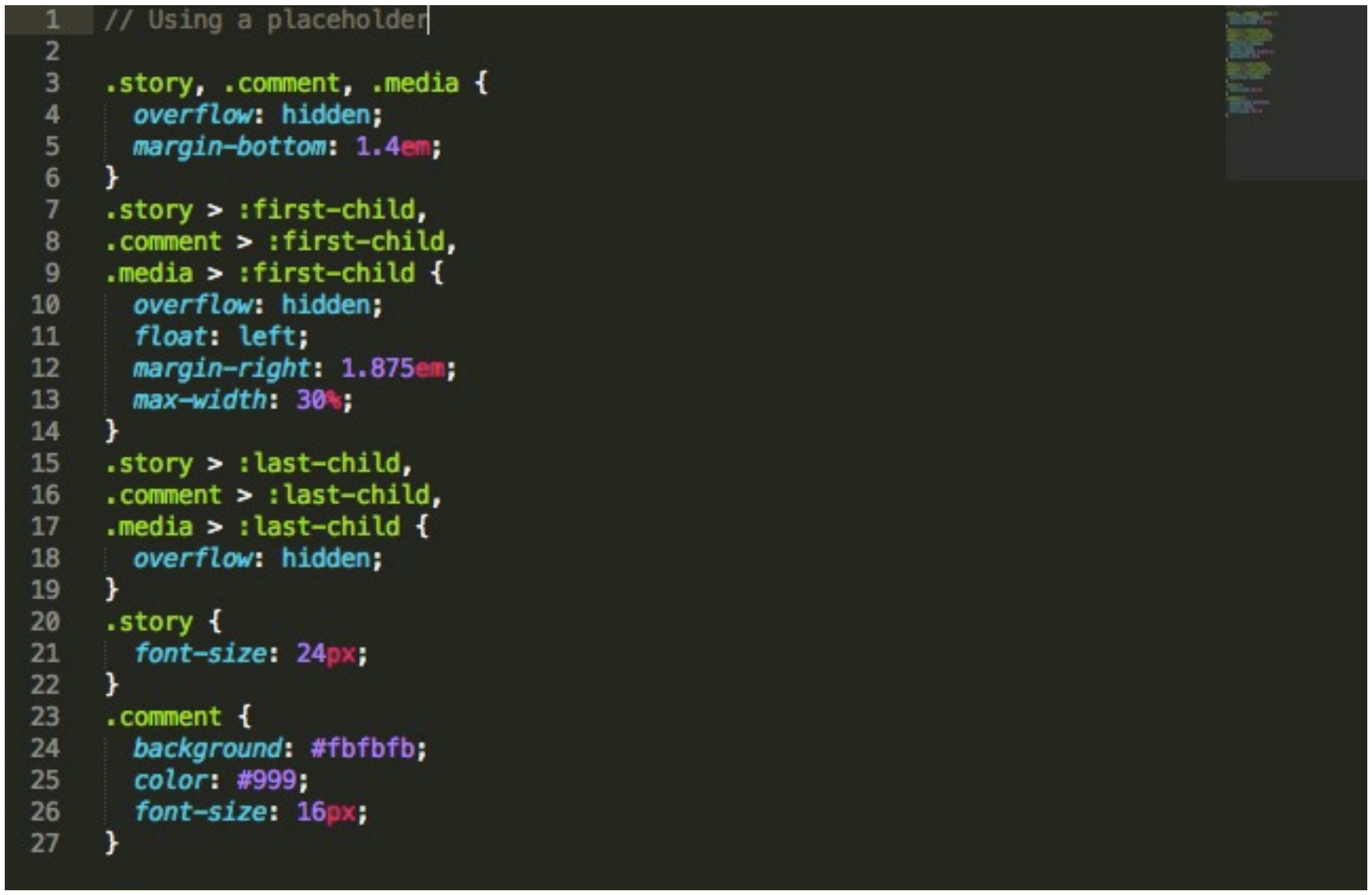
```scss

%media {
    overflow: hidden;
    margin-bottom: $baseline;
    > :first-child {
        overflow: hidden;
        float: left;
        margin-right: $gutter;
        max-width: 30%;
    }
    > :last-child {
        overflow: hidden;
    }
}

.story {
  @extend %media;
  font-size: 24px;
}
.comment {
  @extend %media;
  color: #999;
  font-size: 16px;
}
```

SASS placeholders

keeps our markup clean, and still gives us maintainable CSS

Thursday, November 8, 12

Using placeholders, we can reduce our dependence on non-semantic markup. This example would need a polyfil for browsers that don't support first and last-child pseudo elements, but I think the trade off is worth it.



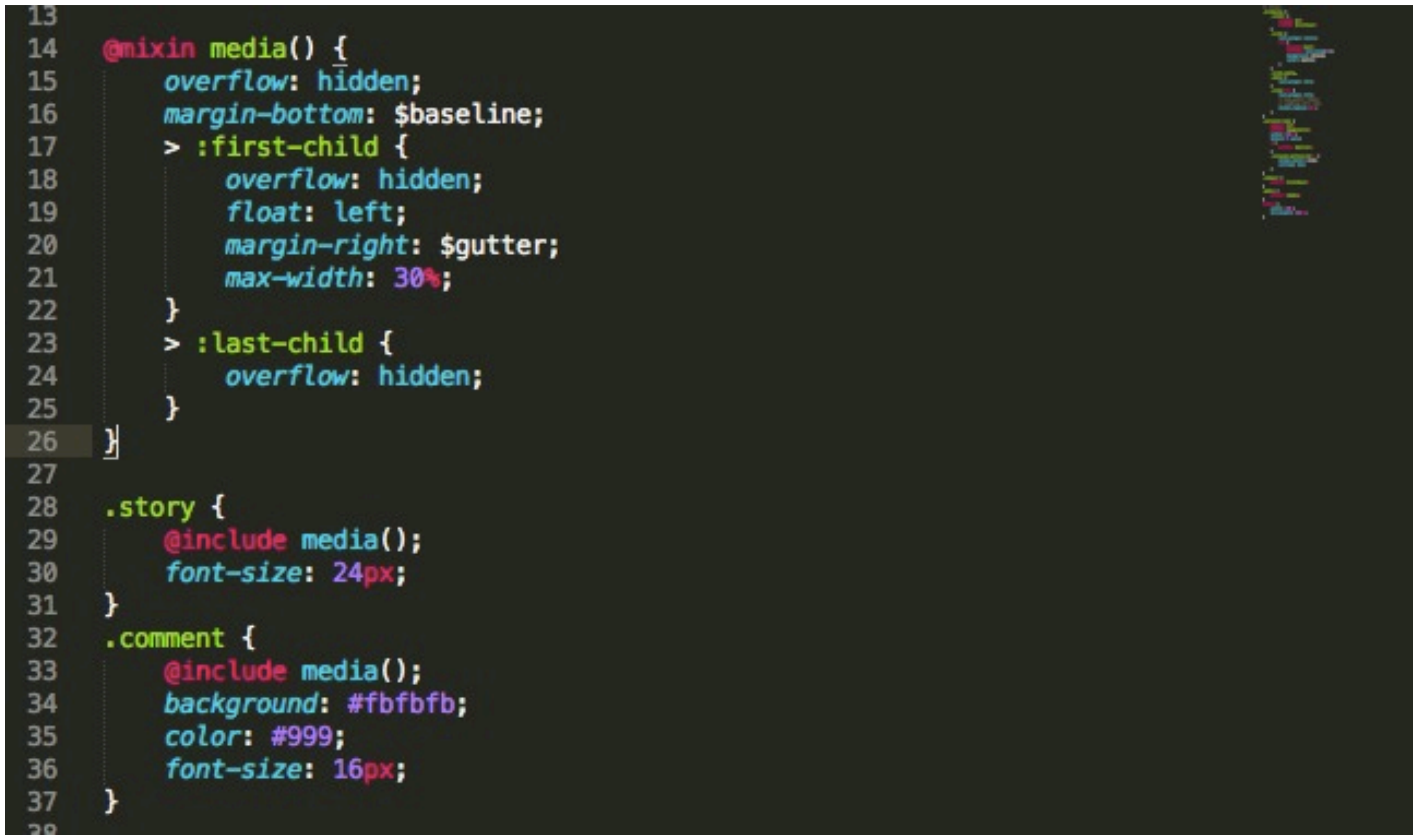
```css
// Using a placeholder

.story, .comment, .media {
  overflow: hidden;
  margin-bottom: 1.4em;
}
.story > :first-child,
.comment > :first-child,
.media > :first-child {
  overflow: hidden;
  float: left;
  margin-right: 1.875em;
  max-width: 30%;
}
.story > :last-child,
.comment > :last-child,
.media > :last-child {
  overflow: hidden;
}
.story {
  font-size: 24px;
}
.comment {
  background: #fbfbfb;
  color: #999;
  font-size: 16px;
}
```

Compiled CSS

Placeholders use CSS inheritance rather than duplicating properties

Thursday, November 8, 12

Looking at the CSS, we can see that SASS smartly uses CSS inheritance rather than just duplicating CSS properties.

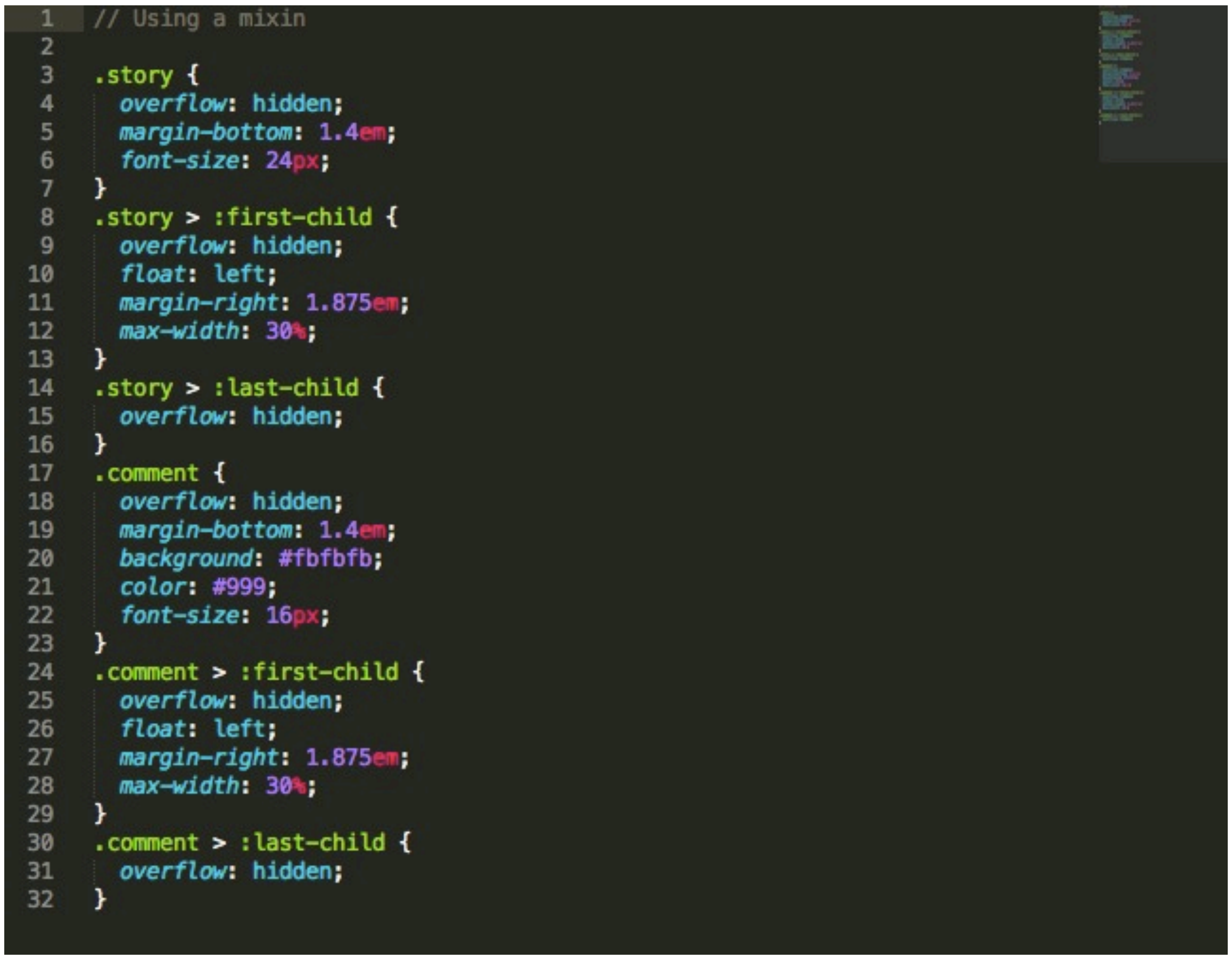


```
@mixin media() {
    overflow: hidden;
    margin-bottom: $baseline;
    > :first-child {
        overflow: hidden;
        float: left;
        margin-right: $gutter;
        max-width: 30%;
    }
    > :last-child {
        overflow: hidden;
    }
}

.story {
    @include media();
    font-size: 24px;
}
.comment {
    @include media();
    background: #fbfbfb;
    color: #999;
    font-size: 16px;
}
```

Mixins

uses just about the exact same amount of source code

We can do the same thing using mixins also, but the CSS it produces is slightly different.


```
// Using a mixin

.story {
  overflow: hidden;
  margin-bottom: 1.4em;
  font-size: 24px;
}
.story > :first-child {
  overflow: hidden;
  float: left;
  margin-right: 1.875em;
  max-width: 30%;
}
.story > :last-child {
  overflow: hidden;
}
.comment {
  overflow: hidden;
  margin-bottom: 1.4em;
  background: #fbfbfb;
  color: #999;
  font-size: 16px;
}
.comment > :first-child {
  overflow: hidden;
  float: left;
  margin-right: 1.875em;
  max-width: 30%;
}
.comment > :last-child {
  overflow: hidden;
}
```

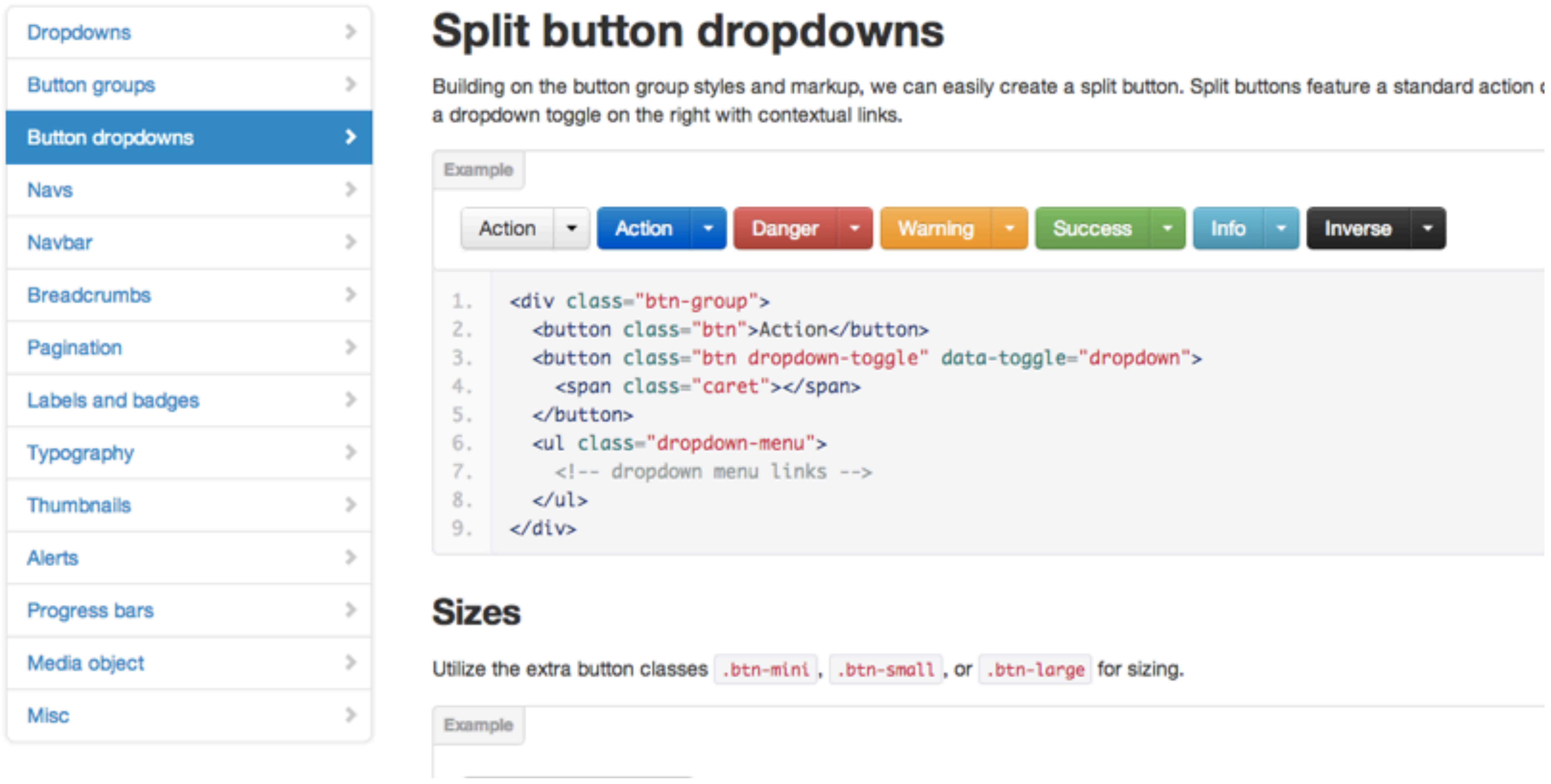
Mixin Compiled CSS

Properties are duplicated for each selector

The trade off is the amount of generated CSS. Using a mixin, you're going to get a lot more code bloat as it applies each property in a mixin to the separate classes it's applied too.

That's not to say mixins don't have a purpose, but I recommend using them when you require a variable to be passed in to the properties. Think of them more as a tool functionality tool.

Let's look at some frameworks that take a modular approach.



Split button dropdowns

Building on the button group styles and markup, we can easily create a split button. Split buttons feature a standard action o
a dropdown toggle on the right with contextual links.

```
<div class="btn-group">
  <button class="btn">Action</button>
  <button class="btn dropdown-toggle" data-toggle="dropdown">
    <span class="caret"></span>
  </button>
  <ul class="dropdown-menu">
    <!-- dropdown menu links -->
  </ul>
</div>
```

Sizes

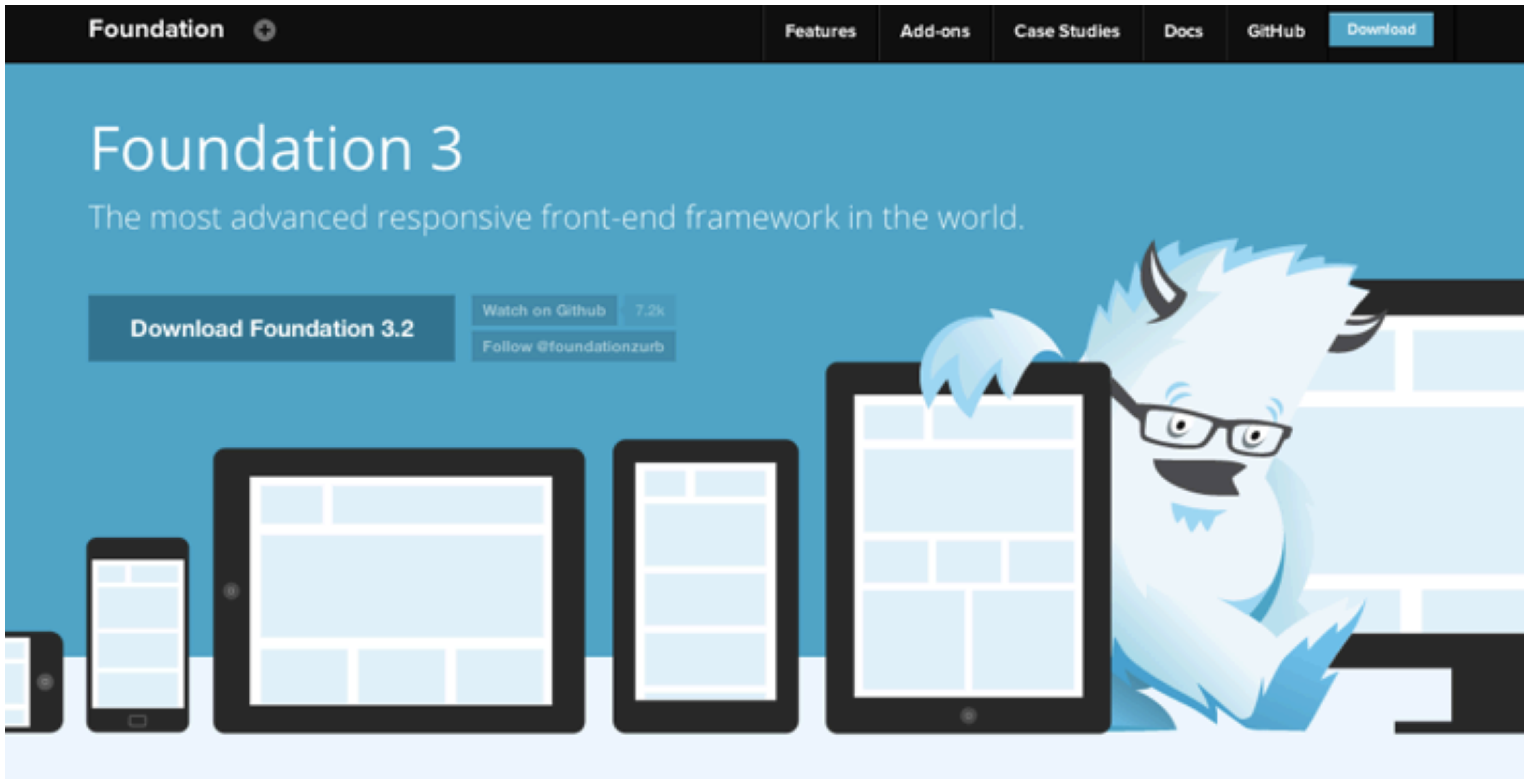
Utilize the extra button classes `.btn-mini`, `.btn-small`, or `.btn-large` for sizing.

twitter bootstrap

The kitchen sink approach

Bootstrap, Foundation, 320 and Up. Look to these for examples, but bootstrap and foundation in particular are fairly large frameworks, so if size is an issue, you might want to build your own or just use bits and pieces.

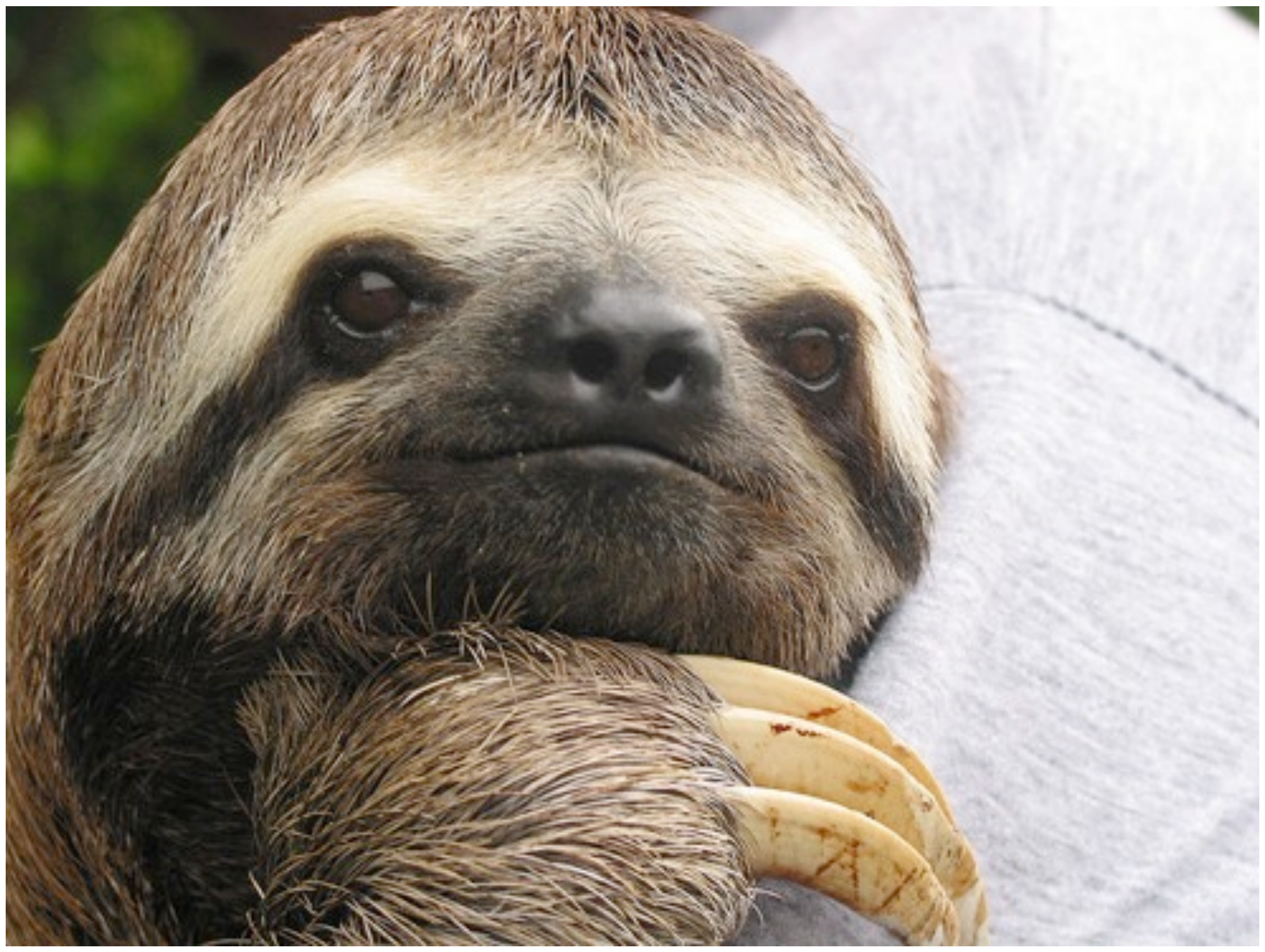
*Twitter bootstrap is great, but the license isn't quite right for WordPress themes



zurb's foundation

MIT license, which is more WP friendly

Like bootstrap, zurb is trying to do everything for everyone. You'll probably only use 10% of the code. It's difficult to take one small piece of foundation and use it for you project. It's well structured though and worth looking at as a reference.

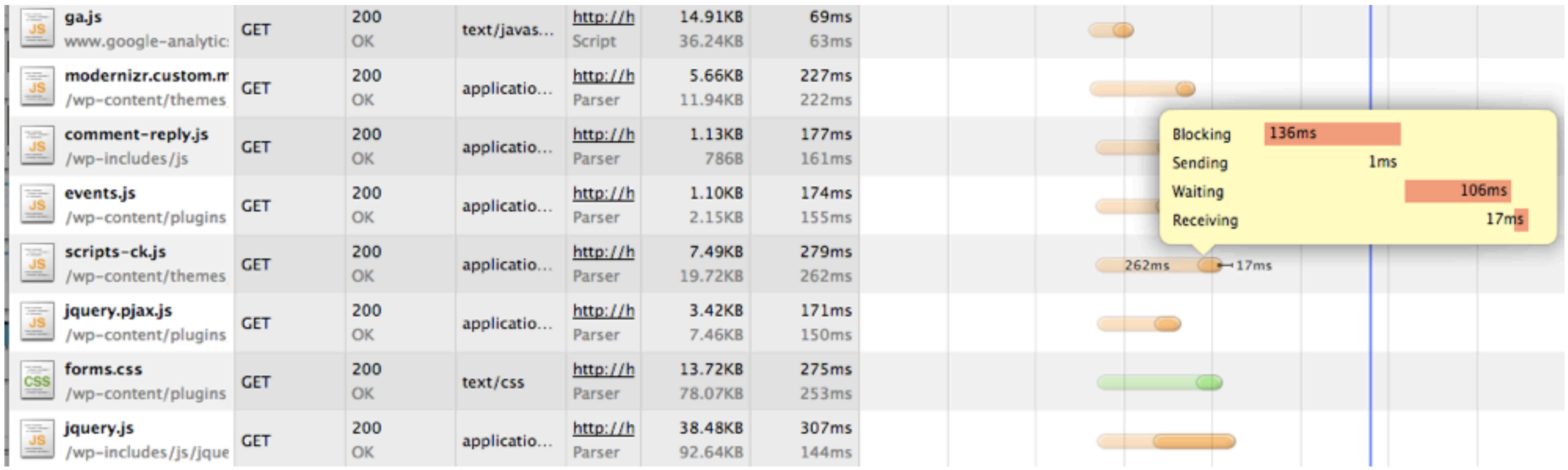


JavaScript

has always had a bad wrap, but everyone uses it. we can speed it up

http://www.flickr.com/photos/praziquantel/30950009/sizes/z/in/photostream/

JavaScript had a rocky start, but it’s proven itself as a very capable even though it has its quirks.



ga.js www.google-analytic:	GET	200 OK	text/javas...	http://h Script	14.91KB 36.24KB	69ms 63ms	
modernizr.custom.m /wp-content/themes	GET	200 OK	applicatio...	http://h Parser	5.66KB 11.94KB	227ms 222ms	
comment-reply.js /wp-includes/js	GET	200 OK	applicatio...	http://h Parser	1.13KB 786B	177ms 161ms	
events.js /wp-content/plugins	GET	200 OK	applicatio...	http://h Parser	1.10KB 2.15KB	174ms 155ms	
scripts-ck.js /wp-content/themes	GET	200 OK	applicatio...	http://h Parser	7.49KB 19.72KB	279ms 262ms	262ms 17ms
jquery.pjax.js /wp-content/plugins	GET	200 OK	applicatio...	http://h Parser	3.42KB 7.46KB	171ms 150ms	
forms.css /wp-content/plugins	GET	200 OK	text/css	http://h Parser	13.72KB 78.07KB	275ms 253ms	
jquery.js /wp-includes/js/jque	GET	200 OK	applicatio...	http://h Parser	38.48KB 92.64KB	307ms 144ms	

Blocking	136ms
Sending	1ms
Waiting	106ms
Receiving	17ms

scripts block other assets from downloading

Thursday, November 8, 12

One of the biggest downsides to JS is actually loading it in your website or application. It blocks everything else on the page from downloading and needless to say, this isn't good. In an ideal world that would everything would download in parallel, but their are reasons why this is difficult...

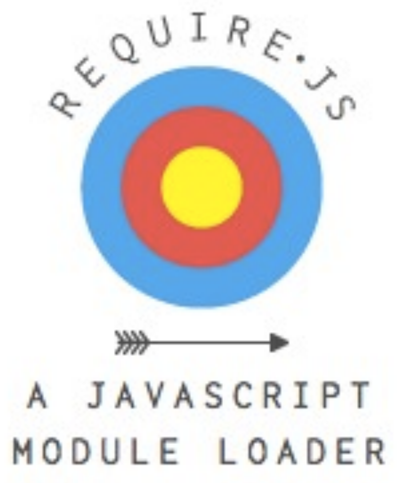
script loaders to the rescue

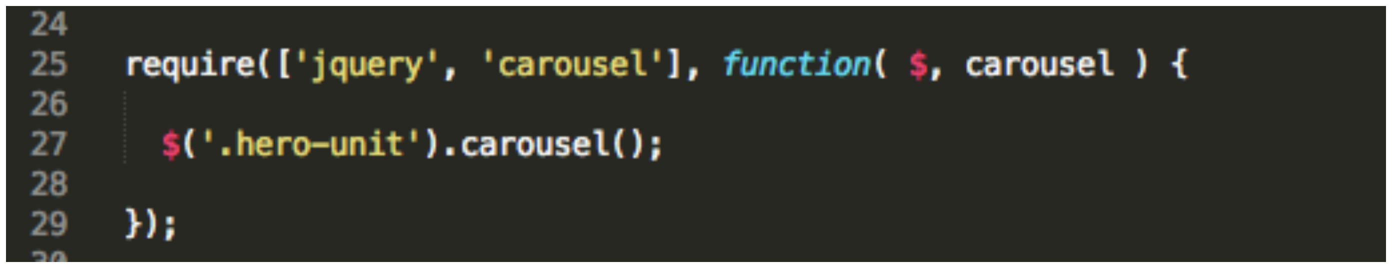
http://www.flickr.com/photos/nerdsoncall/5759398793/sizes/z/in/photostream/

Thursday, November 8, 12

Contrary to popular belief, downloading several scripts in parallel is generally faster than loading one gigantic concatenated script files. Not only that, many large files can't be cached. iPhone has a 25kb limit on cached resources. Script loaders can help load are js asynchronously and allow other files to download.

Asynchronous Module Definition

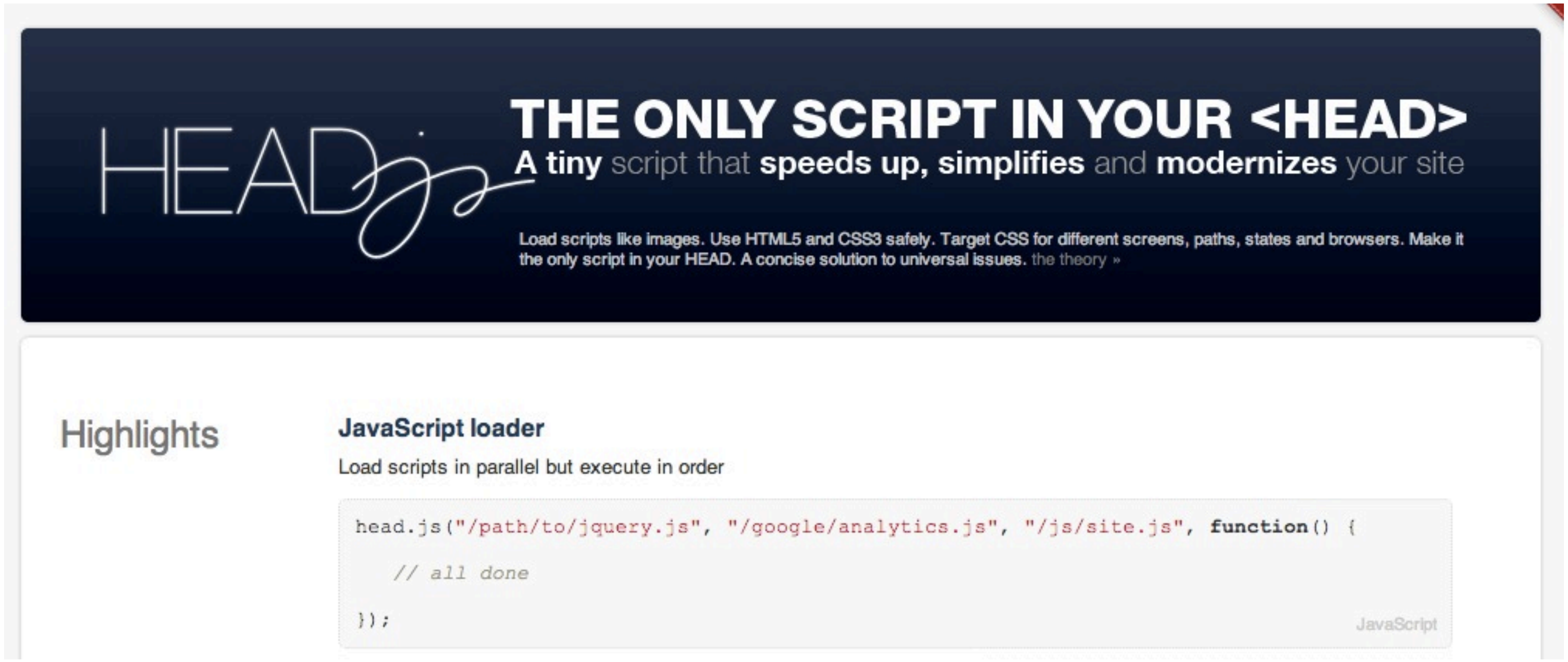




```
require(['jquery', 'carousel'], function( $, carousel ) {

  $('.hero-unit').carousel();

});
```

Thursday, November 8, 12

There are a few popular script loaders out there, but require.js is probably the most popular. Basically it requires each script to be in it's own separate file and loaded then is loaded through require.js. This is great for larger applications as you clearly see what script has file dependencies.
*Load scripts through js (not php, ruby, or html)
*Forces you to think modularly, which leads to more reusable code


THE ONLY SCRIPT IN YOUR <HEAD>

A tiny script that speeds up, simplifies and modernizes your site

Load scripts like images. Use HTML5 and CSS3 safely. Target CSS for different screens, paths, states and browsers. Make it the only script in your HEAD. A concise solution to universal issues. the theory »

Highlights

JavaScript loader

Load scripts in parallel but execute in order

```
head.js("/path/to/jquery.js", "/google/analytics.js", "/js/site.js", function() {

   // all done

});
```

head.js

smaller, easier to use, you don't need to write your scripts as modules

Thursday, November 8, 12

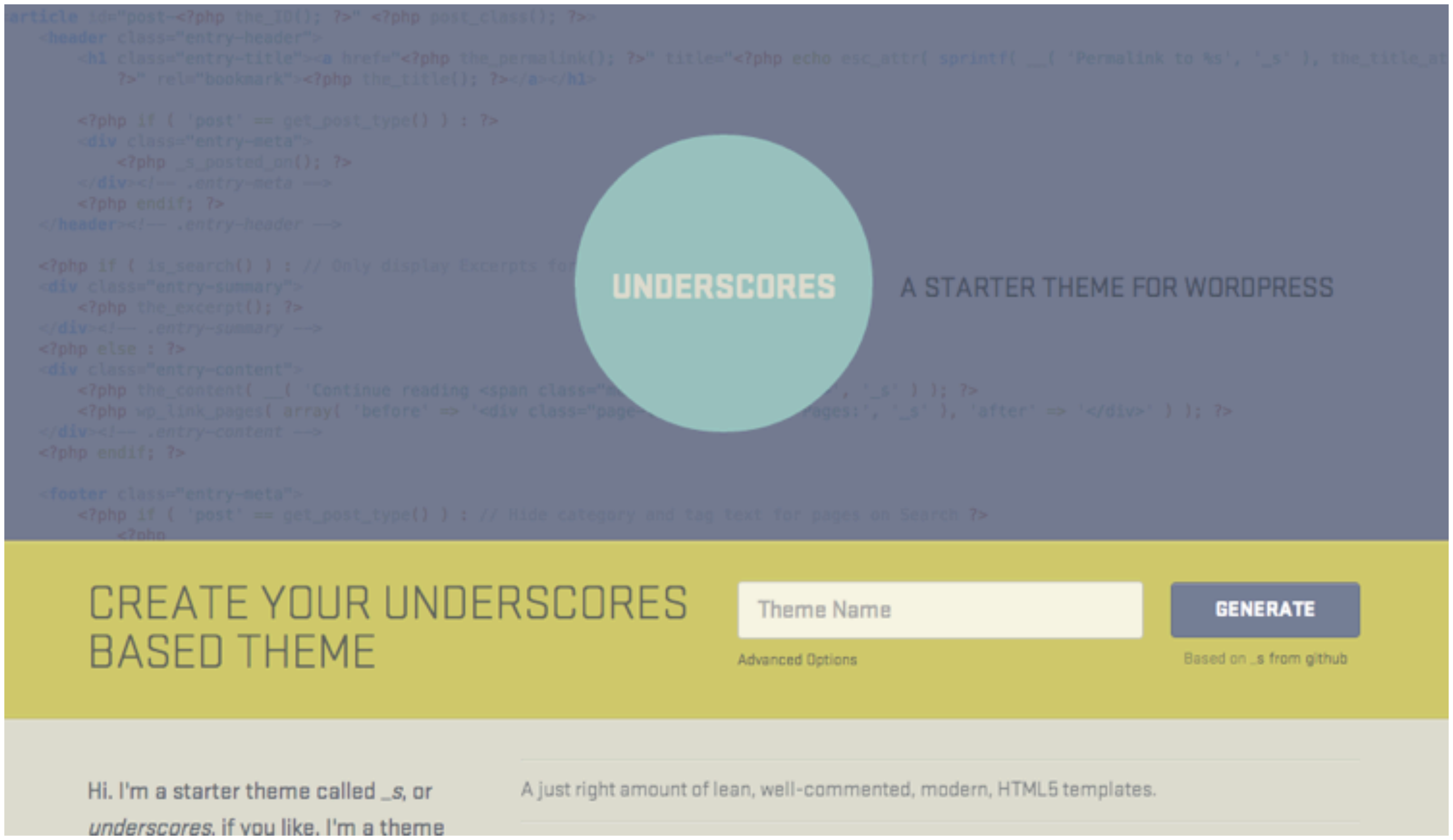
There's also a script loader called labs.js, but it's not under active development anymore because the community has really gotten on board with AMD.

WordPress and the 80/20 rule

Thursday, November 8, 12

So how does all this relate to WordPress? Well one crucial aspect of a better UX on the web is having a fast loading website. About 80% the response time happens on the front-end, so this is where we can gain the most performance benefits. WordPress has proven that it's platform can scale, just look at wp.com. As long as you're not doing some silly database queries from within your template files, focus on improving your front end architecture as that's where most of your performance gains will come from.

http://www.yuiblog.com/blog/2006/11/28/performance-research-part-1/



```
article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
    <header class="entry-header">
        <h1 class="entry-title"><a href="<?php the_permalink(); ?>" title="<?php echo esc_attr( sprintf( __( 'Permalink to %s', '_s' ), the_title_at
            ?>" rel="bookmark"><?php the_title(); ?></a></h1>

        <?php if ( 'post' == get_post_type() ) : ?>
        <div class="entry-meta">
            <?php _s_posted_on(); ?>
        </div><!-- .entry-meta -->
        <?php endif; ?>
    </header><!-- .entry-header -->

    <?php if ( is_search() ) : // Only display Excerpts for
    <div class="entry-summary">
        <?php the_excerpt(); ?>
    </div><!-- .entry-summary -->
    <?php else : ?>
    <div class="entry-content">
        <?php the_content( __( 'Continue reading <span class="m                 ', '_s' ) ); ?>
        <?php wp_link_pages( array( 'before' => '<div class="page-              ages:', '_s' ), 'after' => '</div>' ) ); ?>
    </div><!-- .entry-content -->
    <?php endif; ?>

    <footer class="entry-meta">
        <?php if ( 'post' == get_post_type() ) : // Hide category and tag text for pages on Search ?>
            <?php
```

UNDERSCORES

A STARTER THEME FOR WORDPRESS

CREATE YOUR UNDERSCORES BASED THEME

Hi. I'm a starter theme called *_s*, or *underscores*, if you like. I'm a theme

A just right amount of lean, well-commented, modern, HTML5 templates.

_s theme

integrates well with a modular approach

Thursday, November 8, 12

Using a scalable theme like _s can really help you build a modular front-end for your website. It’s great because it doesn’t make too many assumptions about what you want to build, or how you should structure your front-end assets.

Thanks!

josh@tucsonlabs.com
@tucsonlabs

Thursday, November 8, 12

Thanks to spoke6 for hosting the event!