Highly Maintainable, Efficient & Optimized CSS

August 31, 2010

Think Vitamin HTML & CSS Online Conference

Zoe Mickley Gillenwater

@zomigi

What I do

Books

now Flexible Web Design:
Creating Liquid and Elastic Layouts with CSS

www.flexiblewebbook.com

soon Stunning CSS3:
A Project-based Guide to the Latest in CSS

www.stunningcss3.com

Web

Freelance graphic designer and web developer

CSS consultant

Member of Adobe Task Force for WaSP

Why maintainability matters

These people need to be able to read, understand, and easily change your CSS:

- ✔ you
- ✔ your teammates
- ✔ your successor
- ✔ your clients

Why maintainability matters

If they can't:

- ✔ costs everyone time
- ✔ bloats CSS

Why efficiency matters

- ✔ users like fast-loading pages
- ✔ Google likes fast-loading pages
- ✔ clients like happy users and happy Google

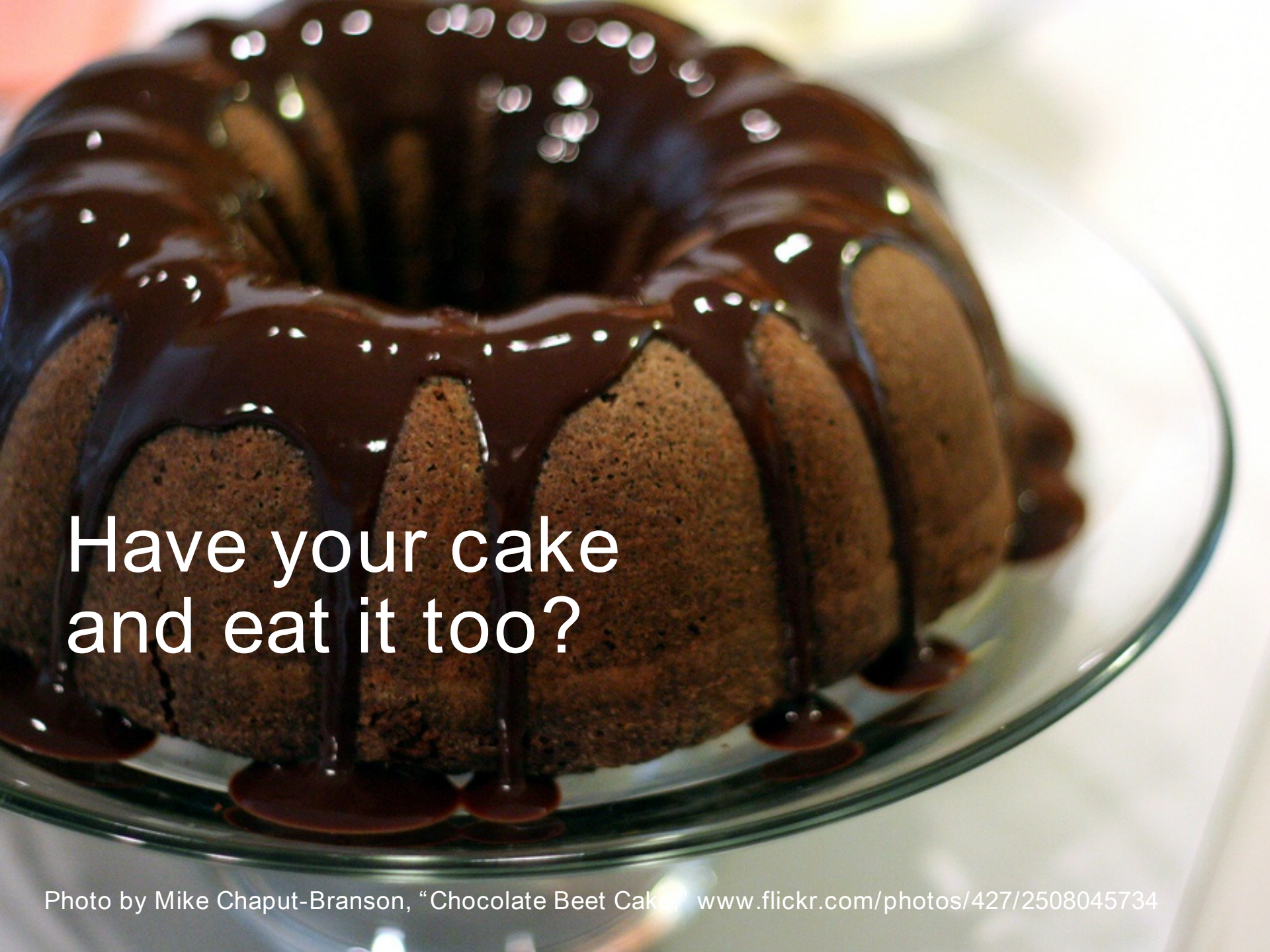

Photo by Mike Chaput-Branson, “Chocolate Beet Cake” www.flickr.com/photos/427/2508045734

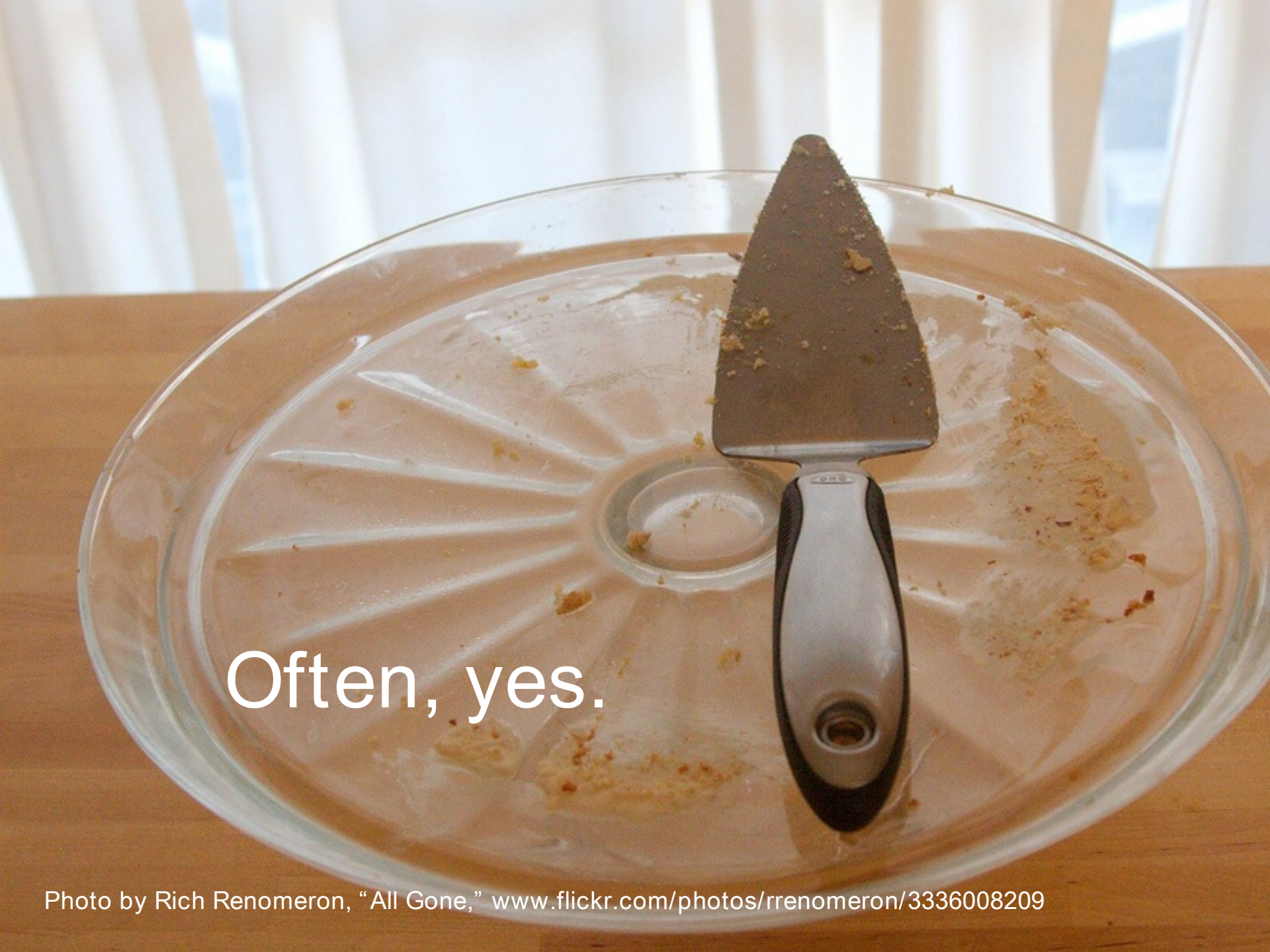

Photo by Rich Renomeron, “All Gone,” www.flickr.com/photos/rrenomeron/3336008209

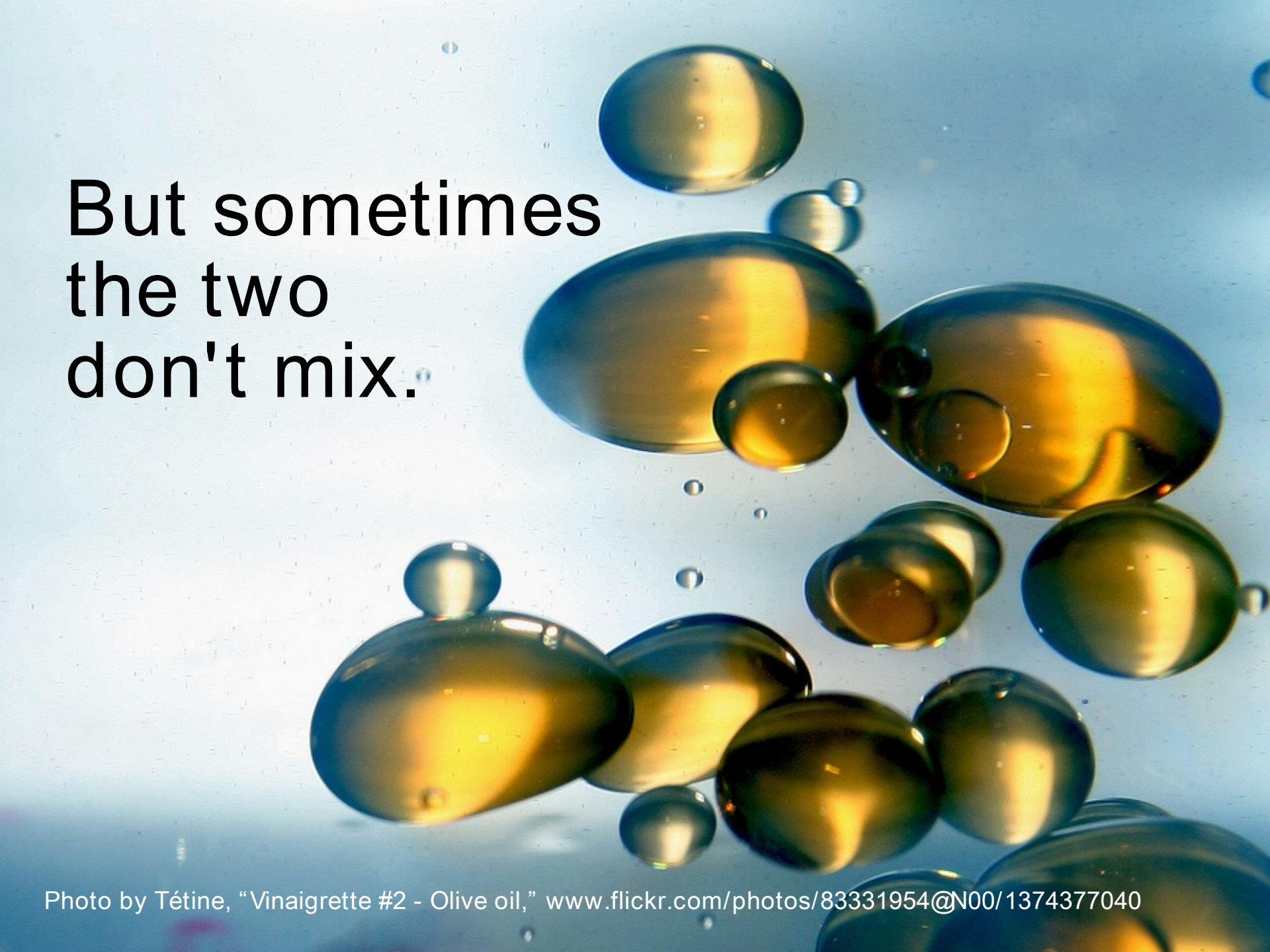

Photo by Tétine, “Vinaigrette #2 - Olive oil,” www.flickr.com/photos/83331954@N00/1374377040

Maintainable but inefficient

```
#widget { /* for Widget modules in main content area */
  overflow: auto; /* for float containment */
  width: 200px;
  background: #ccc;
  border: 1px solid #999;
  }
  #widget h2 {
    color: #39C;
    font-family: Georgia, "Times New Roman", Times, serif;
    }
  #widget li {
    background: url(bullet1.png) no-repeat;
    }
    #widget li li {
      background: url(bullet2.png) no-repeat;
      }
```

Efficient but hard to maintain

```
#widget{overflow:auto;width:200px;background:#ccc;border:1
px solid #999}

#widget h2{color:#39C;font-family: Georgia,"Times New
Roman",Times,serif}

#widget li{background:url(bullet1.png) no-repeat}

#widget li li{background:url(bullet2.png) no-repeat}
```

Balance maintainability and efficiency to get optimized CSS

- Which one gets more weight depends on the project
- Tools can help

What we'll cover

- Some things basic, some things more complicated
- Don't have to use them all
- Some based on personal preference

Maintainable CSS is:

- ✔ organized
- ✔ readable
- ✔ streamlined

1 Organized

How you should divide up (or not) your style sheets and the rules within them to make it easier to find and work with what you need.

No embedded or inline CSS

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Widgets - Acme Inc.</title>
<link rel="stylesheet" type="text/css" href="css/main.css">
<style>
body { background: #C0BCDF }
</style>
</head>

<body>
<div id="wrapper">
  <div id="header">
    <img src="images/logo.png" width="200" height="80">
    <h1 style="color: #211587">Widgets</h1>
```

Separating into multiple sheets

- Media types
- Rule types (layout.css, text.css, etc)
- Site sections (home.css, store.css, etc)
- IE hacks via conditional comments
- CSS3 browser-prefixed properties

Separating media types

Separate screen and print styles:

```
<link href="main.css" media="screen" rel="stylesheet"
type="text/css">

<link href="print.css" media="print" rel="stylesheet"
type="text/css">
```

Overlapped screen and print styles:

```
<link href="main.css" rel="stylesheet" type="text/css">

<link href="print.css" media="print" rel="stylesheet"
type="text/css">
```

Keeping media types together

```
body {
  color:#333;
  background-color:#ccc;
}
@media screen, projection {
  body { background-image:url(background.png); }
}
@media print {
  body { background-color: #fff; color:#000; }
}
```

- One less HTTP request
- Easier to remember non-screen styles

Separating by rule type or site sections

- Pros:
  - + Easy to know where to look for a style
  - + Pick and choose which page gets which CSS files
- Cons:
  - – Redundancy of styles between sheets
  - – Hard to know where to look for a style
  - – Extra HTTP requests
- Bottom line: depends on project

Separating IE hacks with conditional comments

One IE sheet:

```
<!--[if lte IE 8]>
  <link rel="stylesheet" href="ie_all.css" type="text/css">
<![endif]-->
```

Different IE sheets for different IE versions:

```
<!--[if IE 6]>
  <link rel="stylesheet" href="ie_6.css" type="text/css">
<![endif]-->

<!--[if IE 7]>
  <link rel="stylesheet" href="ie_7.css" type="text/css">
<![endif]-->

<!--[if IE 8]>
  <link rel="stylesheet" href="ie_8.css" type="text/css">
<![endif]-->
```

Separating IE hacks with conditional comments

- Pros:
  - + No invalid/confusing hacks in main sheet
  - + Non-IE don't download kb they don't need
  - + Easy to get rid of later when not needed (ha)
- Cons:
  - – Rules for single widget kept in two or more places
  - – Extra HTTP requests
  - – Triggers IE 8 to wait for main sheet to load completely before loading anything else

Conditional comments to add IE classes to html tag

The HTML:

```
<!--[if lt IE 7]> <html class="ie6"> <![endif]-->
<!--[if IE 7]>    <html class="ie7"> <![endif]-->
<!--[if IE 8]>    <html class="ie8"> <![endif]-->
<!--[if IE 9]>    <html class="ie9"> <![endif]-->
<!--[if gt IE 9]> <html> <![endif]-->
<!--[if !IE]>-->  <html> <!--<![endif]-->
```

The CSS:

```
#widget { min-height: 100px; }
.ie6 #widget { height: 100px; }
```

Separating CSS3 browser-prefixed properties

Put sheet with prefixes first in HTML:

```
<link href="prefixes.css" rel="stylesheet" type="text/css">
<link href="main.css" rel="stylesheet" type="text/css">
```

In main style sheet:

```
#widget { transform: rotate(90deg); }
```

In prefixes.css:

```
#widget {
  -moz-transform: rotate(90deg);
  -o-transform: rotate(90deg);
  -webkit-transform: rotate(90deg);
}
```

Separating CSS3 browser-prefixed properties

- Pros:
  - + No invalid properties in main sheet
  - + Easy to get rid of later when not needed
  - + Clean code makes you feel warm and fuzzy
- Cons:
  - – Extra HTTP request
  - – Rules for single widget kept in two places
  - – Easy to forget prefixed properties are subject to change

Zoe's rule of thumb regarding organizing style sheets

- Unless working on a very large site, keep everything together as much as possible
- Why:
  - Avoids errors
  - Less debugging and maintenance time
  - Avoids redundancies
  - More efficient

Organizing rules within a sheet

Use comments to group related rules

```
/* LAYOUT ------------------- */
#wrapper { ... }
#header { ... }

/* TEXT ---------------------- */
h1, h2, h3, h4, h5, h6 { ... }

/* IMAGES -------------------- */
.chart { ... }

/* FORMS --------------------- */
label { ... }

/* ARTICLES ------------------ */
.pullquote { ... }
```

Finding rules within a sheet

Add table of contents to top of sheet referring to commented sections

```
/* This sheet contains rules for: */
/* LAYOUT, TEXT, IMAGES, FORMS, ARTICLES */
```

Add special character as searching aid

```
/* =ARTICLES ---------------- */
.pullquote { ... }
```

2 Readable

Understandable by humans just looking at your CSS out of context. Formatting rules to make it easy to tell what does what.

Indent related rules

```
#widget {
  overflow: auto;
  width: 200px;
  background: #ccc;
  border: 1px solid #999;
  }
  #widget h2 {
    color: #39C;
    font-family: Georgia, "Times New Roman", Times, serif;
    }
  #widget li {
    background: url(bullet1.png) no-repeat;
    }
    #widget li li {
      background: url(bullet2.png) no-repeat;
      }
```

Order of declarations in a rule

- Choose a system, document it, and stick with it
- Options:
  - Alphabetical
  - Group related properties (eg: `position`, `top`, `left`)
  - Layout related properties first (eg: display, then positioning, then box model, then text)

Formatting declarations

Each declaration on new line

```
#widget {
  overflow: auto;
  width: 200px;
  background: #ccc;
  }
```

Each declaration on same line

```
#widget { overflow:auto; width:200px; background:#ccc; }
```

Meaningful class/ID names

Bad:

```
.big-red-box {
  background: #f00;
  color: #fff;
  font-size: 150%;
  }
```

Good:

```
.warning {
  background: #f00;
  color: #fff;
  font-size: 150%
  }
```

Informational comments

- Meta info: author, creation date, etc.
- Key of color codes
- Explanations for confusing things
  - Hacks
  - CSS3 browser prefixes
  - Non-intuitive properties/values
  - Items in progress

CSS practices that aren't so readable

- Sprites
  - Bloated, non-intuitive CSS
  - But efficient: avoids HTTP requests
  - CSS3 can reduce need
- Frameworks
  - Class names often don't make sense out of context
  - Whole team needs to be and stay familiar with framework's conventions
  - Make your own

3 Streamlined

Achieve a particular look with as little CSS as possible.

Consolidate properties

- Shorthand properties to group related properties
- Resets to remove browser defaults in one place instead of several
- Inheritance to set default values in one place, then override only when needed

Inheritance

Bad:

```
#sidebar h3 { color: #000; }
.sale h3    { color: #fff; }
.article h3 { color: #000; }
.news h3    { color: #000; }
```

Better:

```
#sidebar h3, .article h3, .news h3 { color: #000; }
.sale h3 { color: #fff; }
```

Best:

```
h3 { color: #000; }
.sale h3 { color: #fff; }
```

Consolidate rules

- Grouped selectors
- Classes instead of IDs
- Classes instead of styling generic-elements based on their location
- Classes with reusably broad names

Classes instead of IDs

Bad:

```
#sidebar-news li { border-bottom: 1px dashed #ccc; }
#footer-news li { border-bottom: 1px dashed #ccc; }
#press-releases li { border-bottom: 1px dashed #ccc; }
```

Good:

```
.news li { border-bottom: 1px dashed #ccc; }
```

Classes instead of location-based element styles

Bad:

```
#sidebar li { border-bottom: 1px dashed #ccc; }
```

Could end up looking like this:

```
#sidebar li { border-bottom: 1px dashed #ccc; }
#sidebar .twitter li { border-bottom: none; color: #3C9; }
#footer-news li { border-bottom: 1px dashed #ccc; }
```

Good:

```
.news li { border-bottom: 1px dashed #ccc; }
.twitter { color: #3C9; }
```

Give classes broad names

Bad:

```
.contact-form-error { color: #f00; }
.login-form-error   { color: #f00; }
.search-form-error  { color: #f00; }
```

Good:

```
.error { color: #f00; }
```

Remove unused rules using Dust-Me Selectors

- www.sitepoint.com/dustmeselectors
- Firefox extension that analyzes your page or entire site and creates report of selectors that don't apply to anything

Efficient CSS is quick to:

- ✔ download
- ✔ render/redraw

Reduce load times

- Reduce file size:
  - Remove unnecessary characters
  - Use Gzip compression
- Reduce HTTP requests:
  - Combine multiple CSS files
  - Use sprites
  - Use CSS3
- Improve parallel downloading:
  - Don't combine @import and link
- Improve CSS parsing speed:
  - Use efficient selectors

Minify your CSS

- Remove unnecessary characters (white space, comments) to reduce file size
- Only on live file, not working files
  - Make changes to non-minified file
  - Copy sheet, minify, upload

How to minify your CSS

- Manual as part of formatting practices
- Semi-automated web tools:
  - www.codebeautifier.com
  - www.csscompressor.com
  - www.cssoptimiser.com
  - www.cssdrive.com/index.php/main/csscompressor
- Semi-automated command-line tools:
  - http://yuilibrary.com/projects/yuicompressor
  - http://csstidy.sourceforge.net

Minify your CSS (and more) with the Minify script

- http://code.google.com/p/minify
- What it does:
  - Removes unnecessary characters
  - Combines multiple CSS or JavaScript files
  - Serves CSS/JS with gzip compression and optimal cache headers
- Automatic when you upload changed file

Gzip compression

- HTTP compression utility
- Can compress many file types on server
- Browsers decode compressed files automatically on client-side
- Needs to be activated on your server and your site

Efficient CSS3 techniques

- Reduce the need for images/JS/Flash:
  - border-radius
  - box-shadow
  - text-shadow
  - gradients
  - transforms
- Reduce the need for separate mobile style sheets, sites, or width/device detection scripts using media queries

Don't use @import and link

- Details at www.stevesouders.com/blog/2009/04/09/dont-use-import
- Causes style sheets to be loaded sequentially instead of at same time
  - @import and link together in head blocks IE
  - link with @import in it blocks all browsers

Efficient selectors

- Browsers read selectors right to left
- IDs fastest, then classes, tags, universal
  - Fast: `div ul #widget { ... }`
  - Slow: `#widget li a { ... }`
- Don't add tags to start of selectors
  - Bad: `div#header { ... }`
- Avoid descendent selectors when possible
  - Bad: `#footer p { font-size: 80% }`
  - Good: `#footer { font-size: 80% }`

Selectors aren't *that* important

- Don't sacrifice code quality and maintainability
- Speed tests: www.stevesouders.com/blog/2009/03/10/performance-impact-of-css-selectors

Learn more

Download slides, get links:

www.zomigi.com/blog/maintainable-efficient-css/

Book:

www.stunningcss3.com

Zoe Mickley Gillenwater

@zomigi

design@zomigi.com

www.zomigi.com

Questions?

Zoe Mickley Gillenwater
@zomigi
design@zomigi.com
www.zomigi.com