



Cascading Style Sheets

Theory into Practice

Communication Design for the World Wide Web

Refresher...

Cascading Style Sheets

- *Cascading Style Sheets is a means to separate the presentation from the structural markup (xhtml) of a web site. By applying a CSS style you have the ability to keep the structure of your document lean and fast, while controlling the appearance of content.*

Content

- ***Content is the collective term for all the text, images, videos, sounds, animations, and files (such as PDF documents) that you want to deliver to your audience.***

Structure

- *XHTML enables you to define what each element of your content is (heading, paragraph, list of items, hyperlink, image, etc.)*
- *This is done with tags (enclosed in angle brackets <>) that identify each element of your content.*
- *Defines a document's structure.*

Process

- *Start with a blank page of content. Include your headers, navigation, a sample of the content and your footer.*
- *Next start adding your markup.*
- *Then start adding your CSS.*
- *Use HTML tables semantically – for tabular data, not layout.*

The Style Sheet - What

- *A style sheet is a set of stylistic CSS rules that tell a browser how the different parts of a XHTML document are presented.*
- *A style sheet is simply a text file with the file name extension **.css***

The Style Sheet - How

- Linked / External

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

- Embedded / Internal

```
<head>
```

```
  <style type="text/css">
```

```
    /* styles go here */
```

```
  </style>
```

```
</head>
```

- Inline

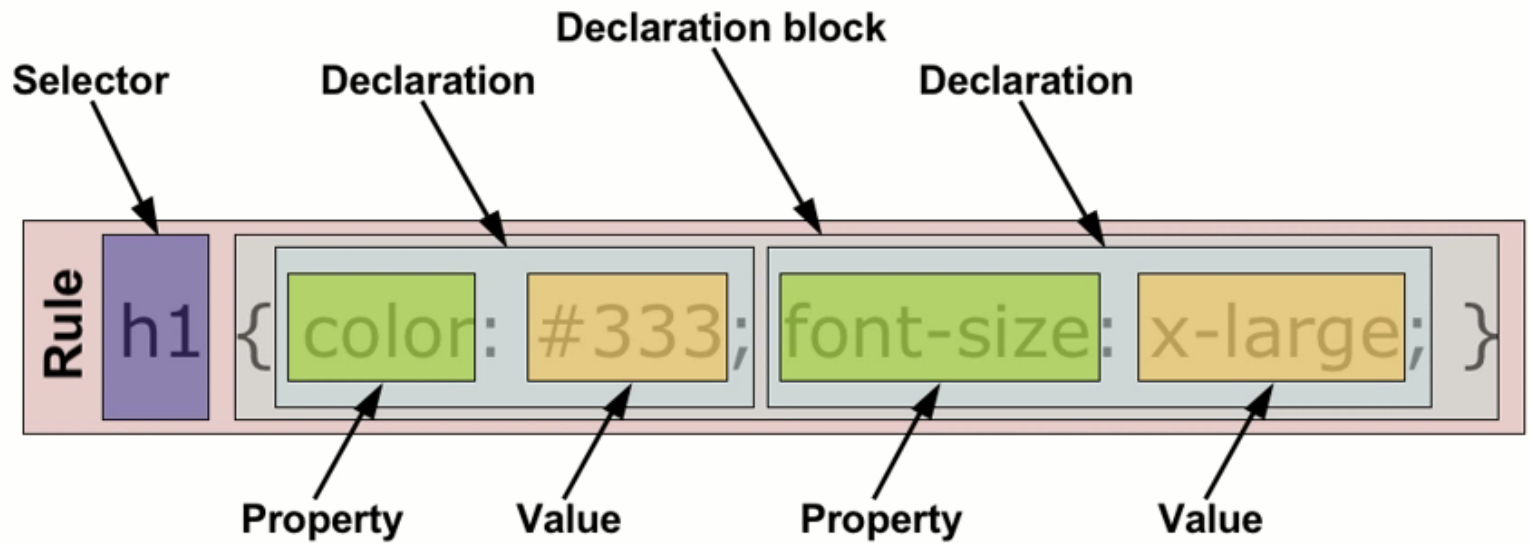
```
<p style="/* styles go here */">Text</p>
```

Anatomy...

Anatomy of a CSS Rule

- A CSS rule is made up of the *selector*, which states which tag the rule selects (or targets), and the *declaration*, which states what happens when the rule is applied.
- The *declaration* itself is made up of a *property*, which states what is to be affected, and a *value*, which states what the property is set to.

Anatomy of a CSS Rule



h1 { color: #333; font-size: x-large; }

Selectors

- 3 types of selectors
 - Tag selectors
 - ID selectors
 - Class selectors
- Selectors should never start with a number, nor should they have spaces in them

Tag Selectors

- Can be any type of HTML element
 - body
 - p
 - div

```
p {  
    background-color: red;  
}
```

ID Selectors

- Give an ID to any element to style it differently than other similar elements
- You can not have two elements on one page with the same ID

```
p#special {  
    background-color: blue;  
}
```

Class Selectors

- Can give multiple elements a class to style them differently

```
p.semispecial {  
    background-color: green;  
}
```

Descendant Selectors

- You can chain selectors together!
Just put a space between them

```
body #container p.warning a {  
    color: red;  
}
```

What will be affected by this style?

Grouping Selectors

- You can group selectors together as well with commas

```
p.warning, p.special {  
    color: red;  
}
```

Now both classes, warning and special, will have red text.

Writing CSS Rules

- This basic structure of the selector and the declaration can be extended in three ways:
 - Multiple declarations within a rule.
p { color:red; font-size:12px; line-height:15px;}
 - Note that each declaration ends with a semicolon.

Writing CSS Rules

- Multiple selectors can be grouped.
h1 {color:blue; font-weight:bold;}
h2 {color:blue; font-weight:bold;}
h3 {color:blue; font-weight:bold;}
- Better to use shorthand:
h1, h2, h3 {color:blue; font-weight:bold;}
- Just be sure to put a comma after each selector except the last.

Writing CSS Rules

- Multiple rules can be applied to the same selector. If you decide that you also want just the h3 tag to be italicized, you can write a second rule for h3, like this:

```
h1, h2, h3 {color:blue; font-weight:bold;}
```

```
h3 {font-style: italic;}
```

Reset the Styling

- Before you do anything else when coding a website, you should start by overriding all the browser styles.
- Because of browser differences, it's a good idea to “zero out” the formatting for commonly used tags. Set up some basic styles at the beginning of your style sheet that remove the offensive formatting.

Reset the Styling

```
/* Normalizes margin, padding */  
body, div, h1, h2, h3, h4, h5, h6, p, ol, ul, dt, dl,  
  dd, form, blockquote, fieldset, input {  
  padding: 0;  
  margin: 0; }
```

```
/* Normalizes font-size for headers */  
h1, h2, h3, h4, h5, h6 {  
  font-size: 1em; }
```

```
/* Removes list-style from lists */  
ol, ul {  
  list-style: none; }
```

```
/* Removes text-decoration from links */  
a {  
  text-decoration: none; }
```

```
/* Removes border from img */  
a img {  
  border: none; }
```

Flexible Text Using Ems?

- Sizing text using em units—allows users to resize text. While ems are relative units, they also offer a bit more precision and control.
- The em is a sliding measure. One em is a distance equal to the type size. Thus a one em space is proportionately the same in any size.

Flexible Text Using Ems?

- On the Web, if the current **font-size** is the default medium setting (16px in most browsers), 1em would equal 16px. The advantage to using ems for font size, line height, and spacing between elements is that as text size is adjusted, those measurements will adjust *proportionately*.

Flexible Text Using Ems?

- Richard Rutter explains a crafty method for normalizing a base font size using ems

(<http://clagnut.com/blog/348/>), where the units match (more or less) to pixel equivalents. The technique assumes a default browser text size set at medium, which is most often 16px.

Flexible Text Using Ems?

- Set the base **font-size** of the page to 62.5% on the `<body>` element:
`body { font-size: 62.5%; }`
- This magic percentage essentially takes the default medium text down from 16px to 10px. Having a base of 10px means you'll have a nice round number to deal with when designing and you can *think* in pixels while actually setting type in ems.

Flexible Text Using Ems?

- For example, after you apply 62.5% to the `<body>`, 1em would appear as 10px, 1.2em as 12px, .9em as 9px, 1.8em as 18px, and so on. If we were specifying different values for various elements on the page, we might do something like this:

Flexible Text Using Ems?

```
body { font-size: 62.5%; } /* 10px */  
h1 { font-size: 2em; } /* 20px */  
h2 { font-size: 1.8em; } /* 18px */  
p { font-size: 1.2em; } /* 12px */  
#nav { font-size: 1em; } /* 10px */
```

Anchor Link Pseudo-Classes

- Style for Links
 - Four psuedo-classes let you format links in four different states based on how a visitor has interacted with that link. They identify when a link's in one of the following four states:

Anchor Link Pseudo-Classes

- **a:link** denotes any link that your guest hasn't visited yet while the mouse isn't hovering over or clicking it. Your regular, unused Web link.
- **a:visited** is a link that your visitor has clicked before, according to the web browser's history. You can style this type of link differently than a regular link to tell your visitor, "Hey, you've been there already!"

Anchor Link Pseudo-Classes

- **a:hover** lets you change the look of a link as your visitor passes the mouse over it. The rollover effects you can create aren't just for fun—they can provide useful visual feedback for buttons on a navigation bar.
- **a:active** lets you determine how a link looks as your visitor clicks. It covers the brief moment when the mouse button is pressed.

Anchor Link Pseudo-Classes

- In most cases, you'll include at least *:link*, *:visited*, and *:hover* styles in your style sheets for maximum design control. But in order for that to work, you must specify the links in a particular order: *link*, *visited*, *hover*, and *active*.

Anchor Link Pseudo-Classes

- Use this easy mnemonic:

LOVE/HATE. So here's the proper way to add all four link styles:

```
a:link { color: #f60; }  
a:visited { color: #900; }  
a:hover { color: #f33; }  
a:active { color: #b2f511; }
```

Targeting Particular Links

- The styles in the previous section are basic *a* tag styles. They target certain link states, but they style *all* links on a page. What if you want to style some links one way and some links another way? A simple solution is to apply a class to particular link tags.

Targeting Particular Links

- `Site Link`
- To style this link in it's own way, you'd create styles like this:

```
a.footer:link { color: #990000; }
```

```
a.footer:visited { color: #000066; }
```

```
a.footer:hover { color: #3F5876; }
```

```
a.footer:active { color:#990000; }
```

Building Navigation Bars

- Every site needs good navigation features to guide visitors to the information they're after—and help them find their way back. CSS makes it easy to create a great looking navigation bar, rollover effects and all.

Building Navigation Bars

- At heart, a navigation bar is nothing more than a bunch of links. More specifically, it's actually a *list* of the different sections of a site. Lists provide us with a way of grouping related elements and, by doing so, we give them meaning and structure.

Building Navigation Bars

- A navigation menu is based on a simple *list* inside a div, like this:

```
<ul id="NavBar">  
  <li><a href="#">Home</a></li>  
  <li><a href="#">About</a></li>  
  <li><a href="#">Events</a></li>  
  <li><a href="#">Forum</a></li>  
</ul>
```

Divs <div>

- Divs help add structure to a page. Stands for *division* and marks a logical group of elements on a page.
- Divs divide the page into rectangular, box-like areas. Invisible unless you turn on borders or color background.
- Include <div> tags for all major regions of your page, such as header, main content, sidebar, etc.

Divs <div>

- Once your <div> tags are in place, add either a class or ID for styling each <div> separately.
- For parts of the page that appear only once and form the basic building blocks of the page, web designers usually use an ID.
- ID selectors are identified using a hash character; (#) class selectors are identified with a period(.).

Type of Web Page Layouts

- Nearly every page design you see falls into one of three types of layouts:
 - *fixed width*
 - *liquid*
 - *elastic*

Web Page Layouts: Fixed

- Fixed width offers consistency. In some cases, the design clings to the left edge of the browser window, or more commonly, it's centered.
- Many fixed width designs are about 760px wide—a good size for 800 x 600 screens (leaves room for scrollbars).
- However, more and more sites are about 950 pixels wide, on the assumption that visitors have at least 1024 x 768 monitors.

Web Page Layouts: Liquid

- A liquid design adjusts to fit the browser's width. Your page gets wider or narrower as your visitor resizes the window. Makes the best use of the available browser window real estate, but it's more work to make sure your design looks good at different window sizes.
- On very large monitors, these types of designs can look really wide.

Web Page Layouts: Elastic

- An elastic design is a fixed-width design with a twist—type size flexibility. You define the page's width using em values. An em changes when the browser's font size changes, so the design's width is based on the browser's base font size.
- Elastic designs keep everything on your page in the same relative proportions.

Two-Column Fixed Layout

- Very common layout; it contains a narrow left column for navigation and a right column that houses the rest of the page's content. In this example, the navigation column is a fixed width, but the content area is fluid—that is, it changes width depending on the width of the browser window.

Two-Column Fixed Layout

```
<html>
<head>
<title>A Simple Two Column Layout Without CSS
  Applied</title>
</head>

<body>
<div id="nav">
  <ul>
    <li><a href="#">Link 1</a></li>
    <li><a href="#">Link 2</a></li>
    <li><a href="#">Link 3</a></li>
  </ul>
</div>

<div id="content">
  <h1>A Simple Two column Layout</h1>
  <p><strong>Step X - bold text here... </strong>More text
  here...</p>
  <p>More text here</p>
  <p>More text here</p>
  <p>More text here</p>
</div>
</body>
</html>
```

Two-Column Fixed Layout

- CSS Applied:

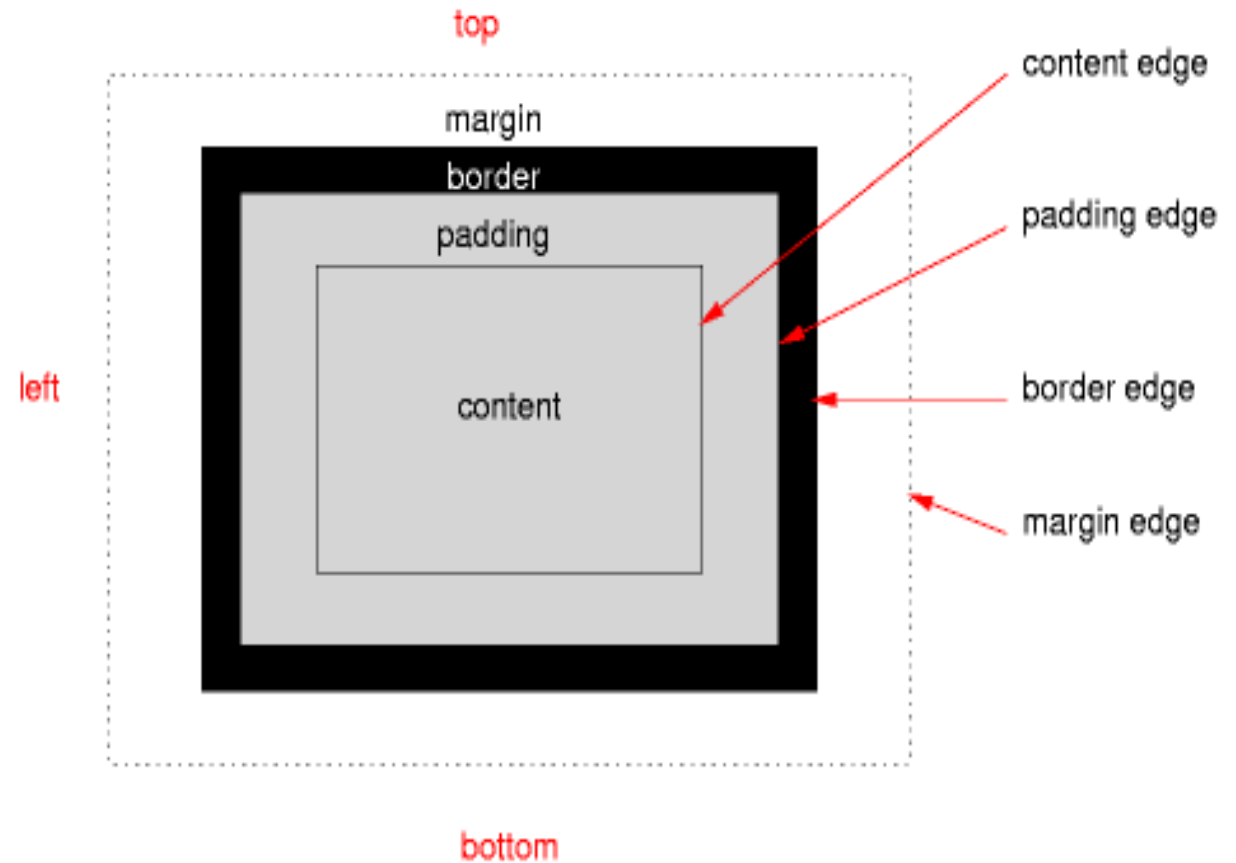
```
body {  
    margin: 0px;  
    padding: 0px;  
}  
div#nav {  
    position: absolute;  
    width: 150px;  
    left: 0px;  
    top: 0px;  
    border-right: 2px  
    solid red;  
}
```

Two-Column Fixed Layout

Now that we have our two-column layout, we need to think about making it look more presentable.

Let's get to work...

Margin, Padding, Border



- Box Model

What's Important for Coders?

- Clean Separation of Code/Content
- Clean Menu Systems
- Provide 'hooks' for the designers

What's Important for Coders?

- Provide 'hooks' for the designers
 - Class and ID on each body element
 - class should be the site section
 - ID should be the page
 - Designers can then use descendant selectors
page-by-page

What's Important for Coders?

- Provide 'hooks' for the designers
 - ID each major 'section' of the page
 - Header
 - Footer
 - Leftside
 - Rightside

**Down the line...
go modular?**

A system for
managing your
CSS

Step 1

Build your HTML



HTML file

Step 2

Build your CSS



HTML file



Master
CSS file

Master CSS file

```
/* container styles */  
#container { }
```

```
/* header styles */  
#header { }  
#header h1 { }
```

```
/* content styles */  
#content { }  
#content h2 { }
```

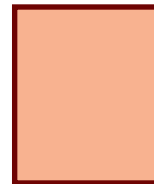
```
/* footer styles */  
#footer { }  
#footer p { }
```

Step 3

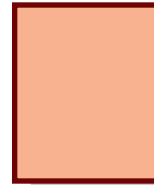
Separate your CSS



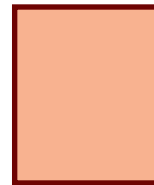
container.css



header.css



content.css



New CSS files

```
#container { }
```

container styles

```
#header { }  
#header h1 { }
```

header styles

```
#content { }  
#content h2 { }
```

content styles

```
#footer { }  
#footer p { }
```

footer styles

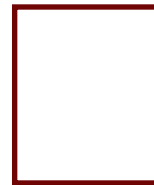
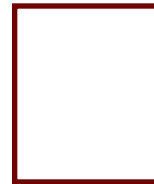
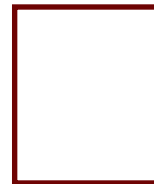
Question

Why separate CSS files?

- Easier to find rules
- More than one developer at a time
- Files can be turned on/off as needed

Step 4

Add a bridging CSS file



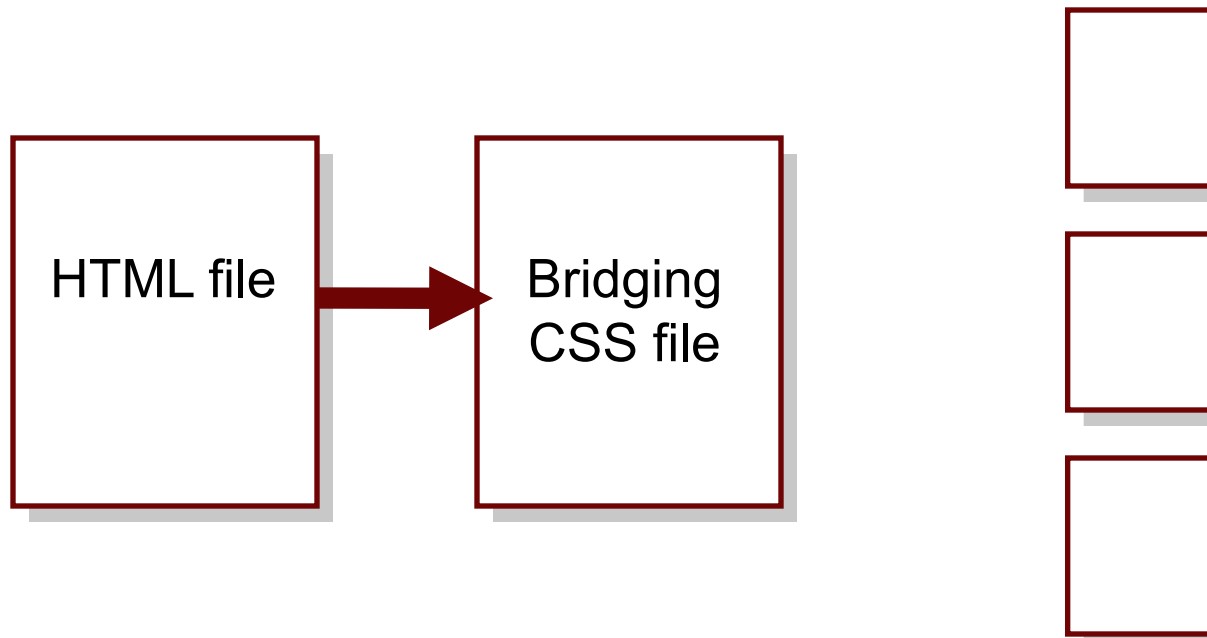
Question

Why add a bridging file?

- One link to all CSS files in HTML
- Change CSS without changing HTML
- Add or remove files as needed

Step 5

Link to bridging file

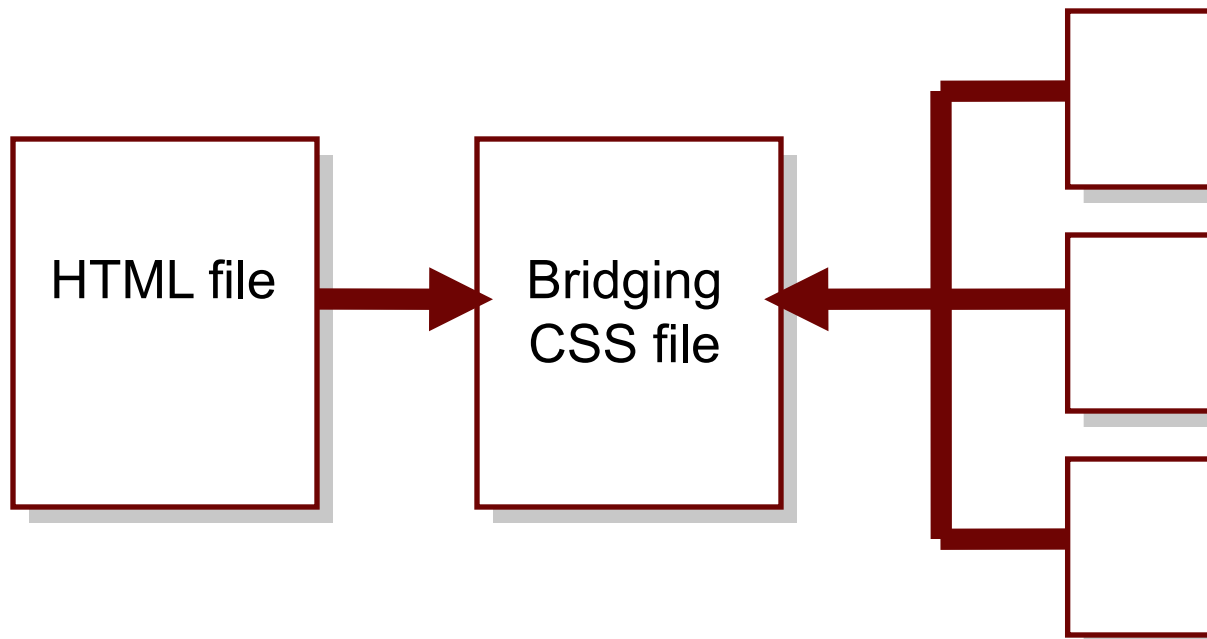


HTML file

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Modular CSS</title>
  <link rel="stylesheet" href="bridging.css"
    type="text/css" media="screen, projection">
</head>
<body>
...
```

Step 6

Import CSS



Bridging CSS file

```
@import "container.css";  
@import "header.css";  
@import "content.css";  
@import "footer.css";
```

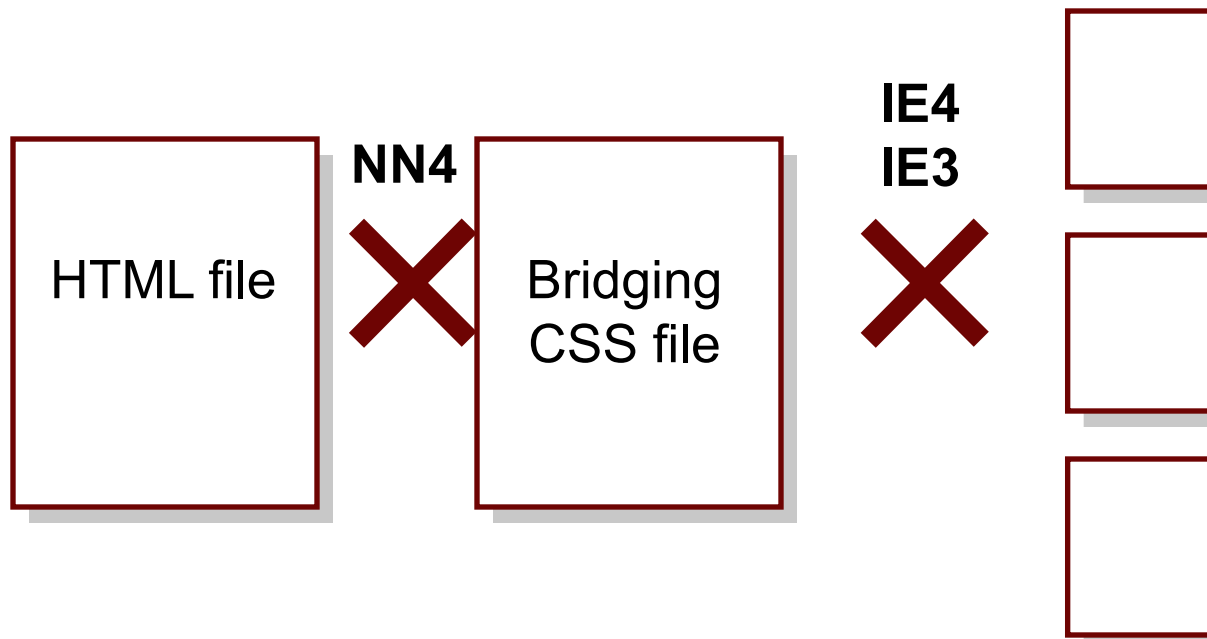
Question

How do @imports work?

- Imports all rules from one file into other
- Exactly as if written in other file
- Order and placement are important
- Cannot be read by older browsers

Old browsers

No modular CSS for you!



The End

Let's go play now!

Communication Design for the World Wide Web