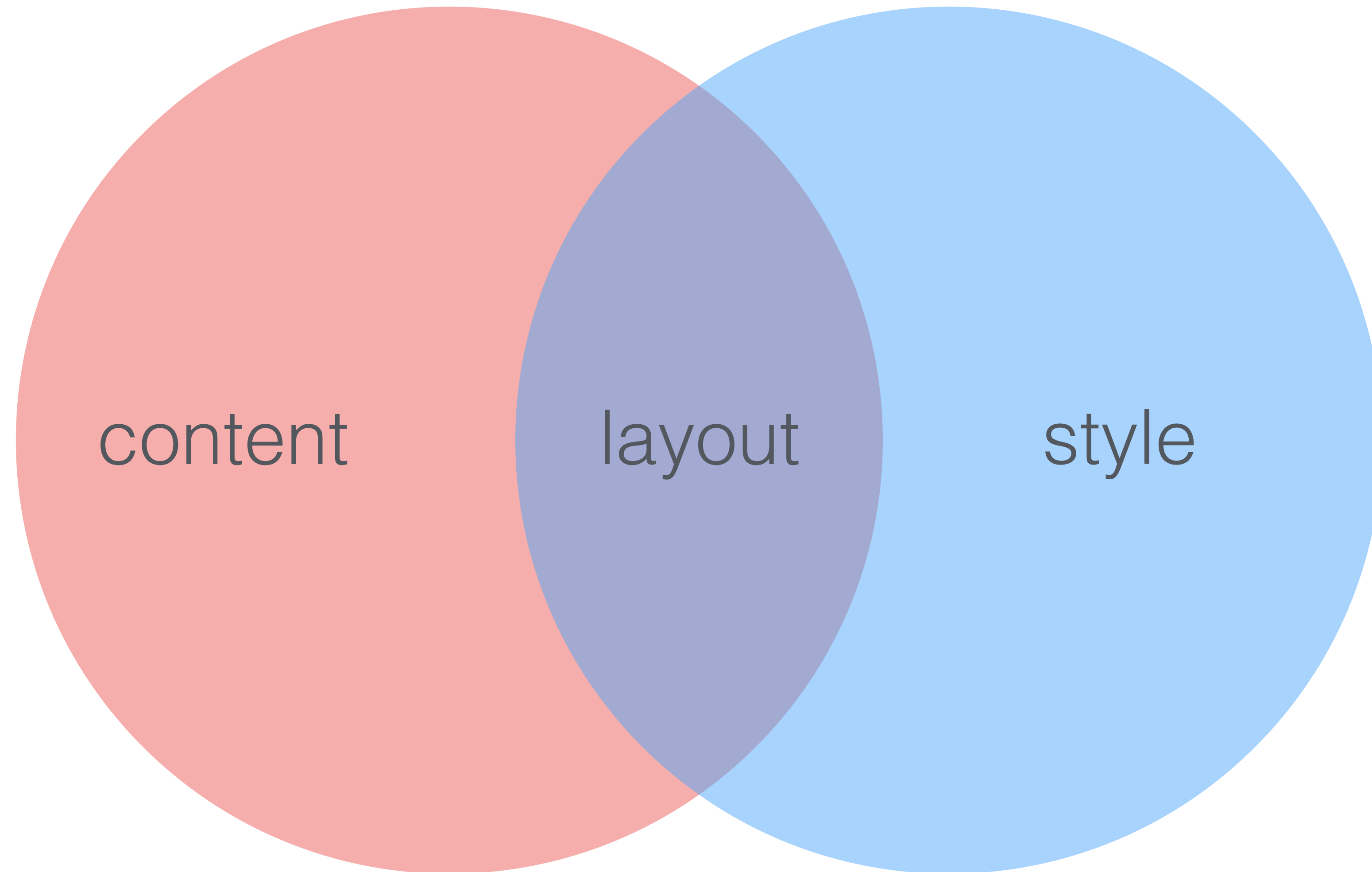


HTML & CSS

Layout laid out

HTML

CSS



WITH CSS

Workshop

Shoestring

Overview

Edit

Comments

Tracking

Pairs

Topics

Materials

Videos

Overview & Objectives

Many companies use a CSS framework for development speed & convenience. Popular frameworks are carefully designed, compatible across many browsers, and rich in features. However, there is a contingent of developers who believe that frameworks like Bootstrap are too aggressive or opinionated in what they provide, and that it's better to either build your own framework or write ad-hoc styles for each project.

In this workshop, we're going to try to recreate the look of a certain [Bootstrap Template](#) without actually using Bootstrap. To accomplish this, we'll have to create our own CSS framework — a subset of Bootstrap which we'll affectionately call "Shoestring". Shoestring will have three key components:

- Typography
- Grids
- Forms
- You are also encouraged to implement another major Bootstrap component, the navbar.

Along the way we'll learn about building modern semantic CSS using tools like Sass (a high-quality CSS extension language).

< Next >

1. Introduction

Pre-reading

Overview & Objectives

2. Setup

Get the Repo

Start the App

3. Sass and Grids

Intro to Sass

Using Sass in Our App

Grid Systems

Build It Already

4. Responsive Setup

What Is Responsive Layout?

Feature Branches

Update the View

5. Responsive Exercise

Write the Responsive Branch

Pull Requests

WITHOUT CSS

Workshop

Shoestring

Overview

Edit

Comments

Tracking

Pairs

Topics

Materials

Videos

Overview & Objectives

Many companies use a CSS framework for development speed & convenience. Popular frameworks are carefully designed, compatible across many browsers, and rich in features. However, there is a contingent of developers who believe that frameworks like Bootstrap are too aggressive or opinionated in what they provide, and that it's better to either build your own framework or write ad-hoc styles for each project.

In this workshop, we're going to try to recreate the look of a certain [Bootstrap Template](#) without actually using Bootstrap. To accomplish this, we'll have to create our own CSS framework — a subset of Bootstrap which we'll affectionately call "Shoestring". Shoestring will have three key components:

- Typography
- Grids
- Forms
- You are also encouraged to implement another major Bootstrap component, the navbar.

Along the way we'll learn about building modern semantic CSS using tools like Sass (a high-quality CSS extension language).

[Edit](#)

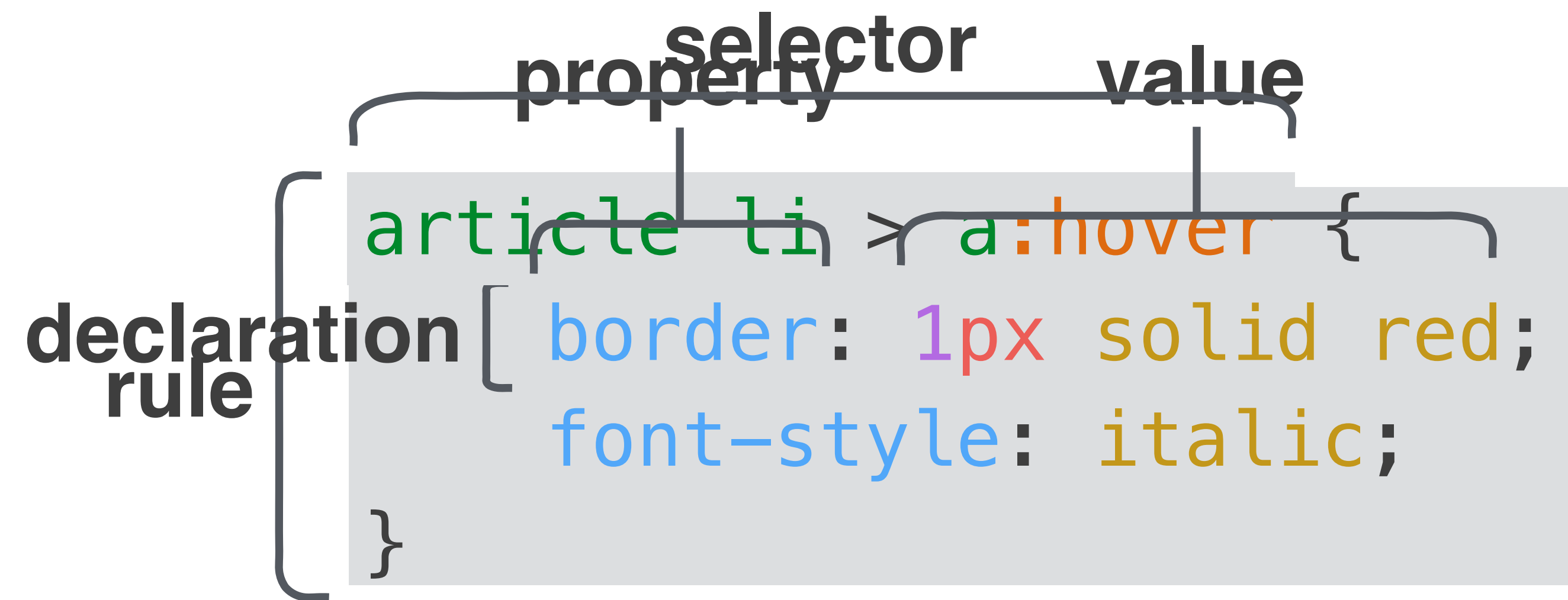
Select Content

- 1510FE
- 1511
- 1511JS
- 1511JS-MTD
- 1601FE
- 1601
- 1601F
- 1601GH

☐ Next

- 1. Introduction
 - [Pre-reading](#)
 - [Overview & Objectives](#)

TERMS



RULE EXAMPLE

apply **these** styles → 

```
article li > a:hover {  
  border: 1px solid red;  
  font-style: italic;  
}
```

to any elements matching **this** selector

even for any future changes ***declarative!***

SELECTORS

tag	<code>input</code>
class	<code>.btn</code>
id	<code>#upload</code>
attribute	<code>[type="file"]</code>
pseudo-element	<code>::after</code>
pseudo-class	<code>:hover</code>
*	*

BEWARE!

- `tag.class` element with BOTH `tag` AND `.class`
- `tag .class` element with `.class` whose ANCESTOR matches `tag`
- `tag, .class` element with EITHER `tag` OR `.class`

CASCADING STYLE SHEETS

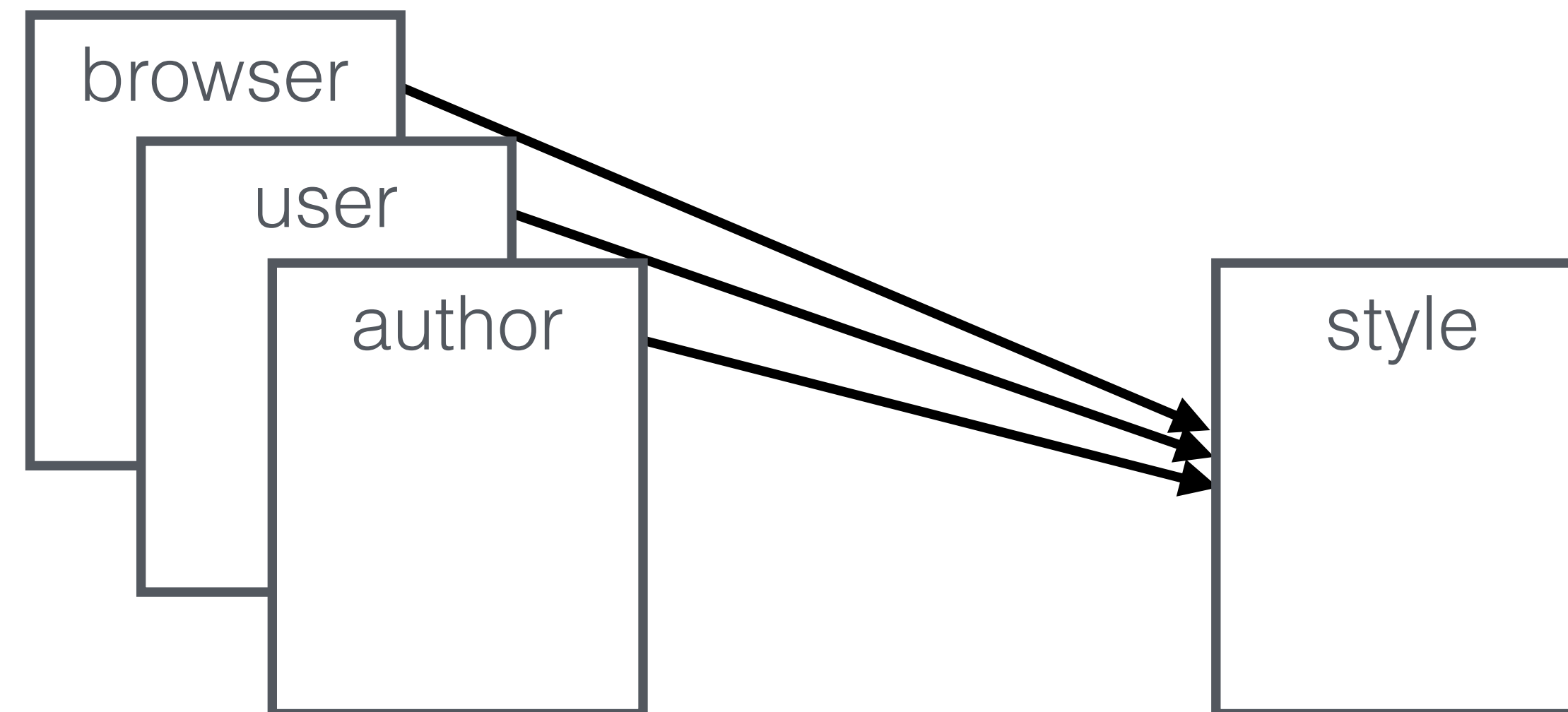
CASCADING

In ~1994... *CSS had one feature that distinguished it from all the [competing style languages]: it took into account that on the Web the style of a document couldn't be designed by either the author or the reader on their own, but that their wishes had to be combined, or "cascaded," in some way.*

CASCADING STYLE SHEETS, DESIGNING FOR THE WEB, BY HÅKON WIUM LIE AND BERT BOS (1999) - CHAPTER 20

CASCADING

An element's style is a merge of every rule whose selector matches



index.html

```
<head>
  <link rel="stylesheet" href="styles-B.css" />
  <link rel="stylesheet" href="styles-A.css" />
</head>
<body>
  <ul>
    <li style="background-color:blue;">A</li>
  </ul>
</body>
```

styles-A.css

```
li {
  color: red;
}
```

styles-B.css

```
li {
  font-size: 40px;
}
```

style

```
element.style {
  background-color: ■ blue;
}
li {
  color: ■ red;
} styles-A.css:1
li {
  font-size: 40px;
} styles-B.css:1
li {
  display: list-item;
  text-align: -webkit-match-parent;
} user agent stylesheet
```

view



What happens when declarations conflict?




```
<div id="thing"></div>
```

```
div {  
  background: red;  
}
```



```
#thing {  
  background: blue;  
}
```



```
<div class="foo"></div>
```

```
div {  
  background: red;  
}
```

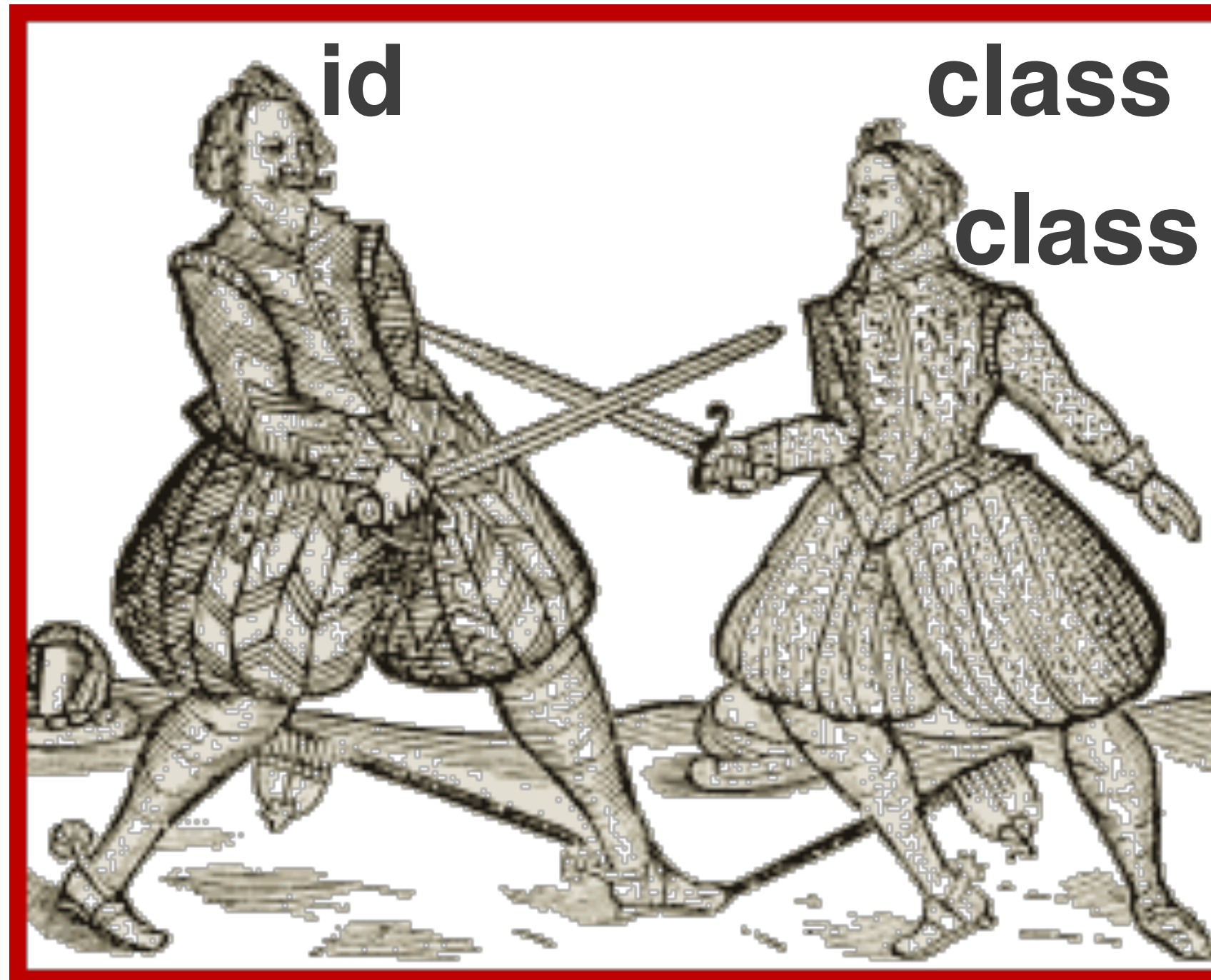


```
.foo {  
  background: green;  
}
```




```
<div id="thing" class="foo bar"></div>
```

```
#thing {  
  background: blue;  
}
```

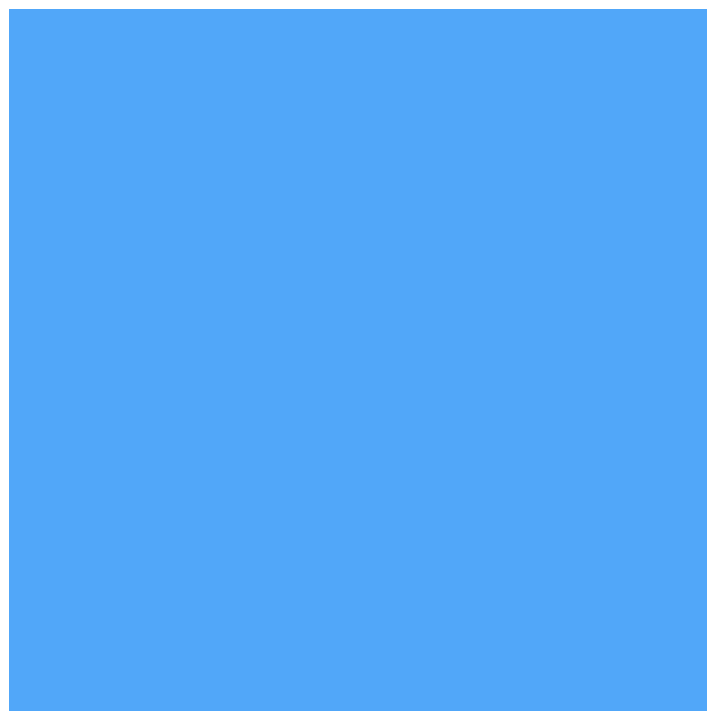


```
.foo.bar {  
  background: green;  
}
```



```
<div class="outer">  
  <div id="thing" class="foo" style="background:orange;"></div>  
</div>
```

```
#thing {  
  background: blue;  
}
```



```
.outer .foo {  
  background: green;  
}
```



