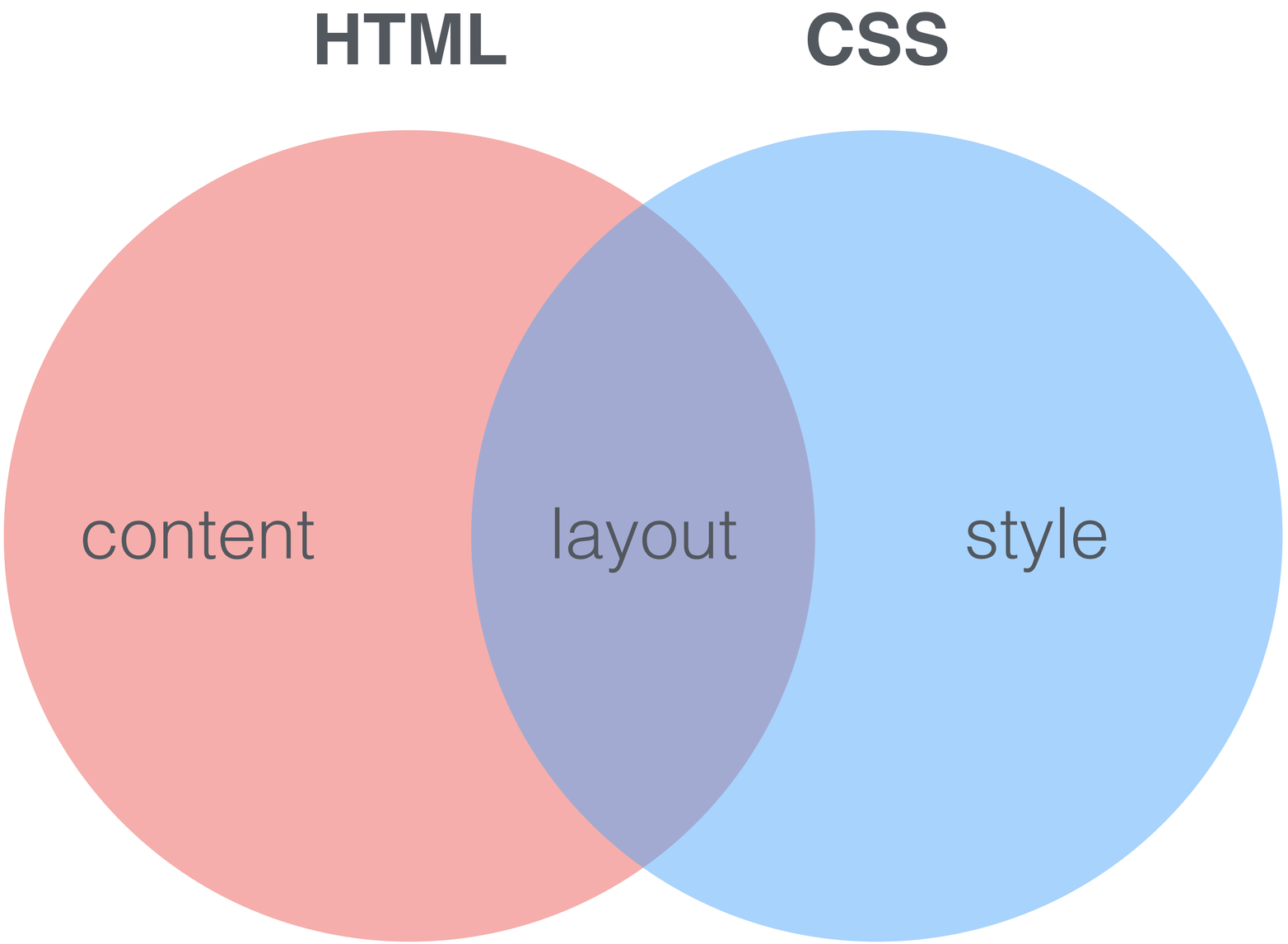
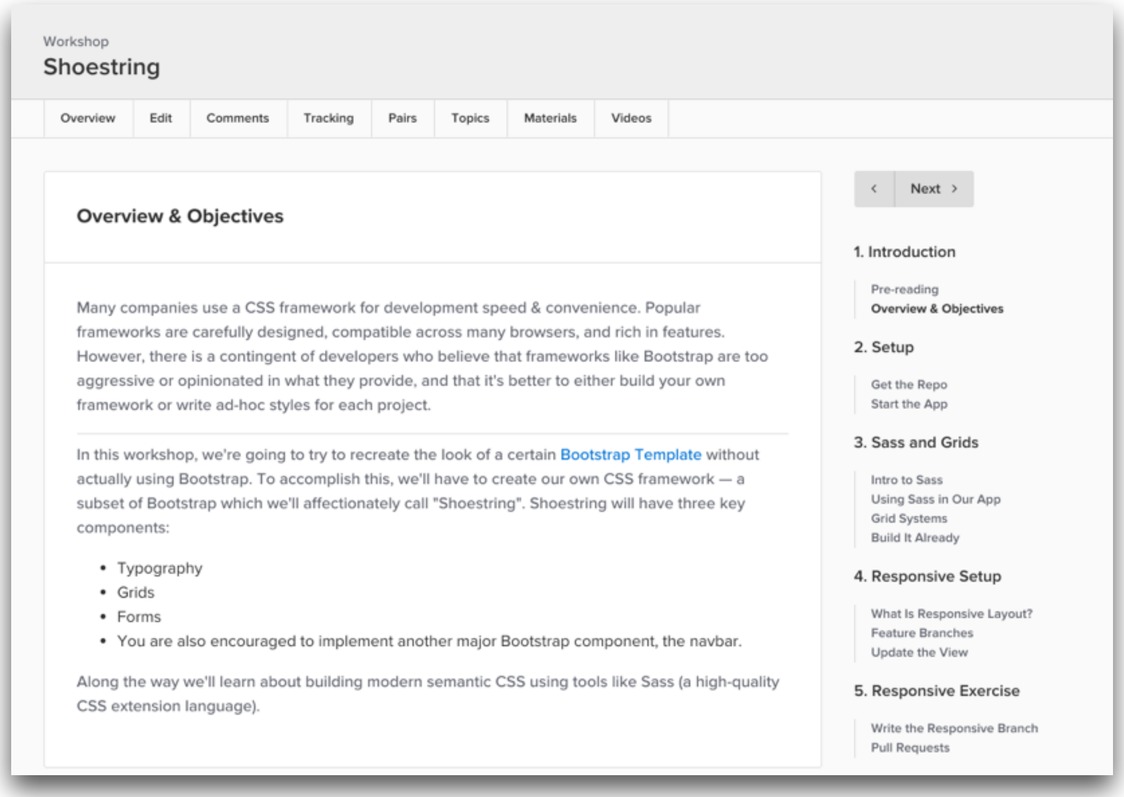
HTML & CSS

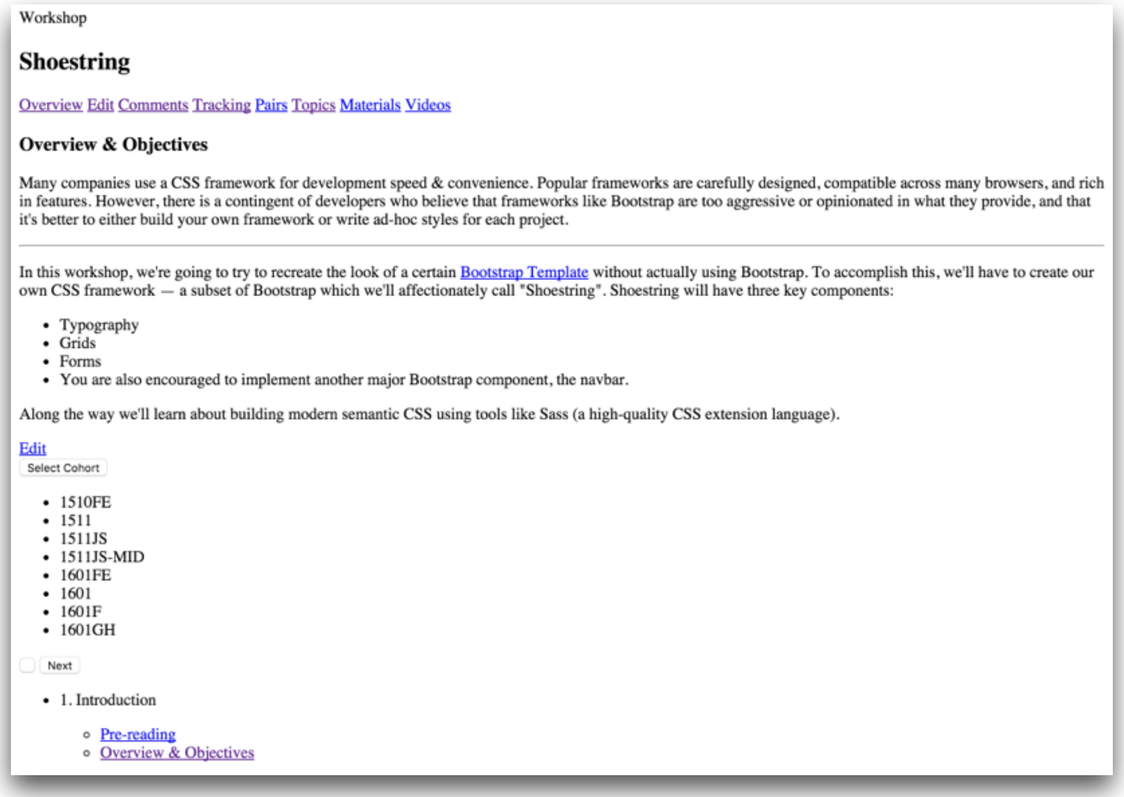
Layout laid out


HTML
CSS
content
layout
style

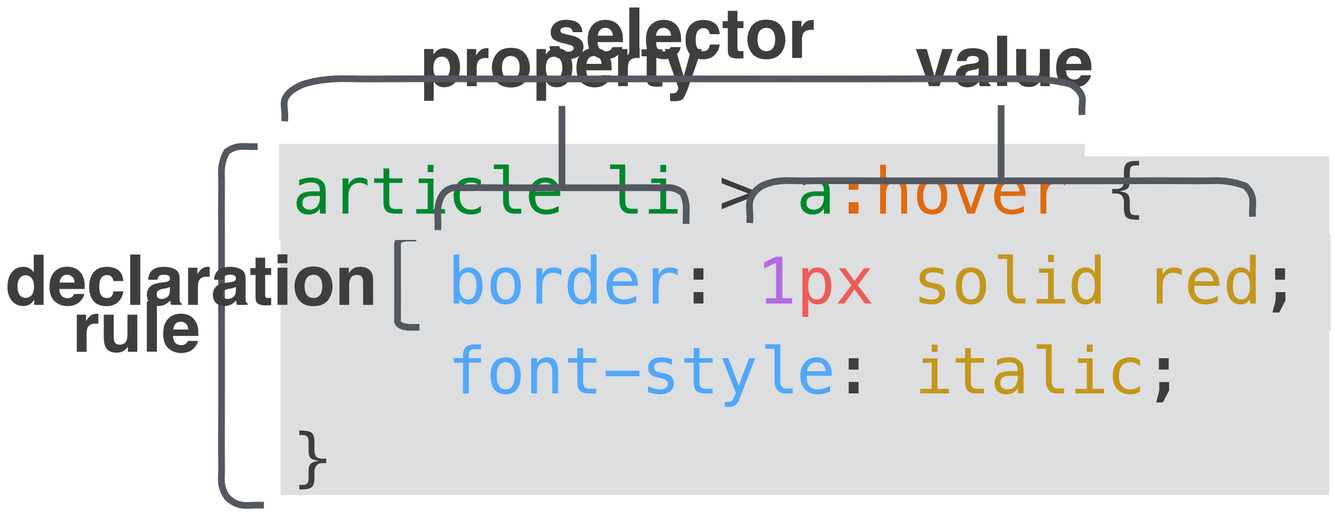
WITH CSS



WITHOUT CSS



TERMS


selector
property
value
declaration
rule
article li > a:hover {
border: 1px solid red;
font-style: italic;
}

RULE EXAMPLE

SELECTORS

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

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

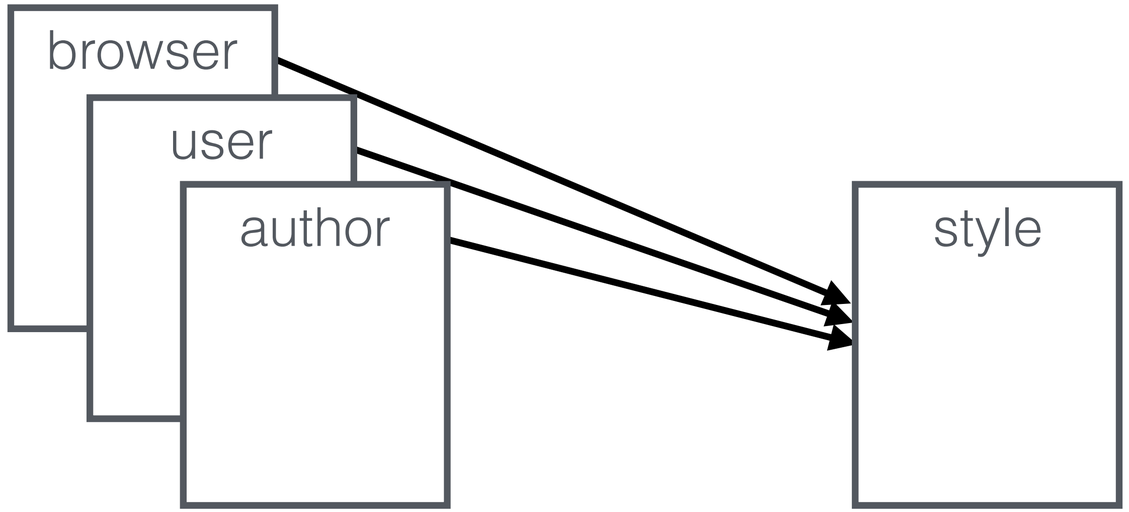
CASCADING

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

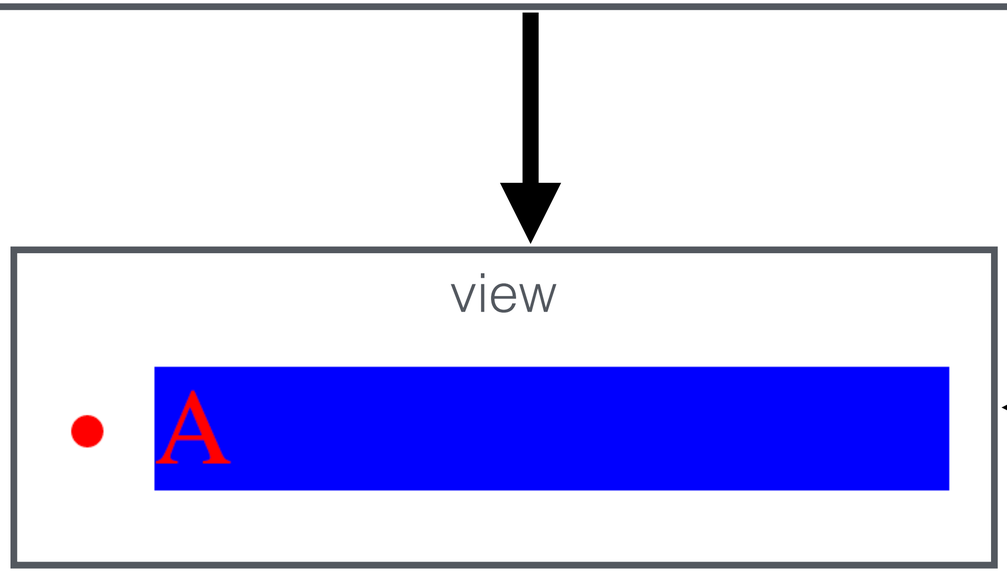
```
<head>
  <link rel="stylesheet" href="styles-B.css" />
  <link rel="stylesheet" href="styles-A.css" />
</head>
<body>
  <ul>
    <li style="background-color:blue;">A</li>
  </ul>
</body>
```

styles-A.css

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

styles-B.css

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



style

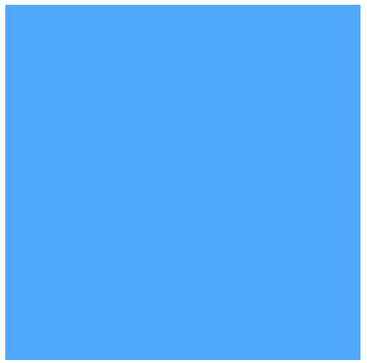
```
element.style {
  background-color: blue;
}

li {                                    styles-A.css:1
  color: red;
}

li {                                    styles-B.css:1
  font-size: 40px;
}

li {                                    user agent stylesheet
  display: list-item;
  text-align: -webkit-match-parent;
}
```

What happens when declarations conflict?

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

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


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



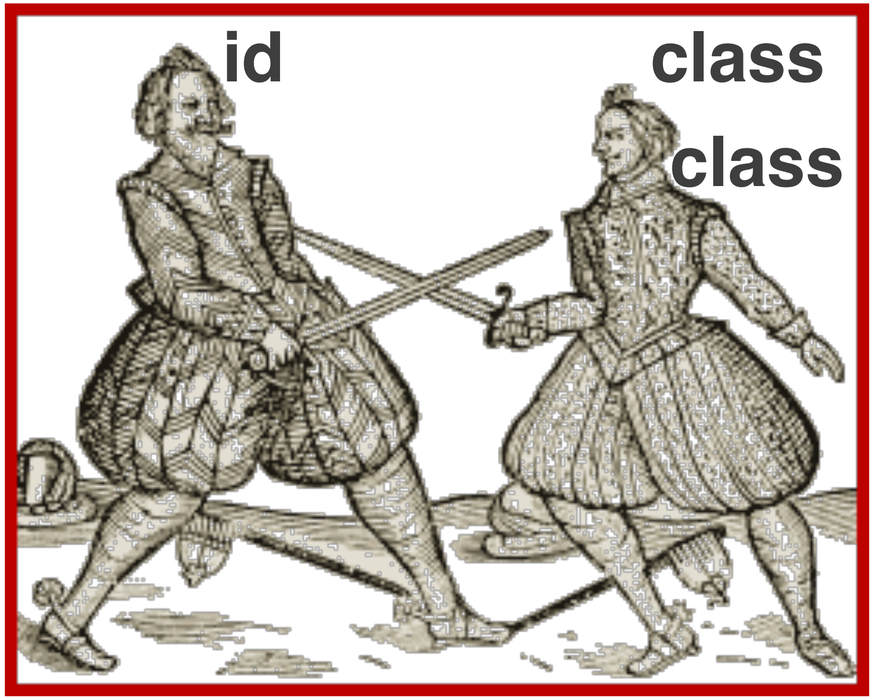
```
<div class="foo"></div>
```

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


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

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

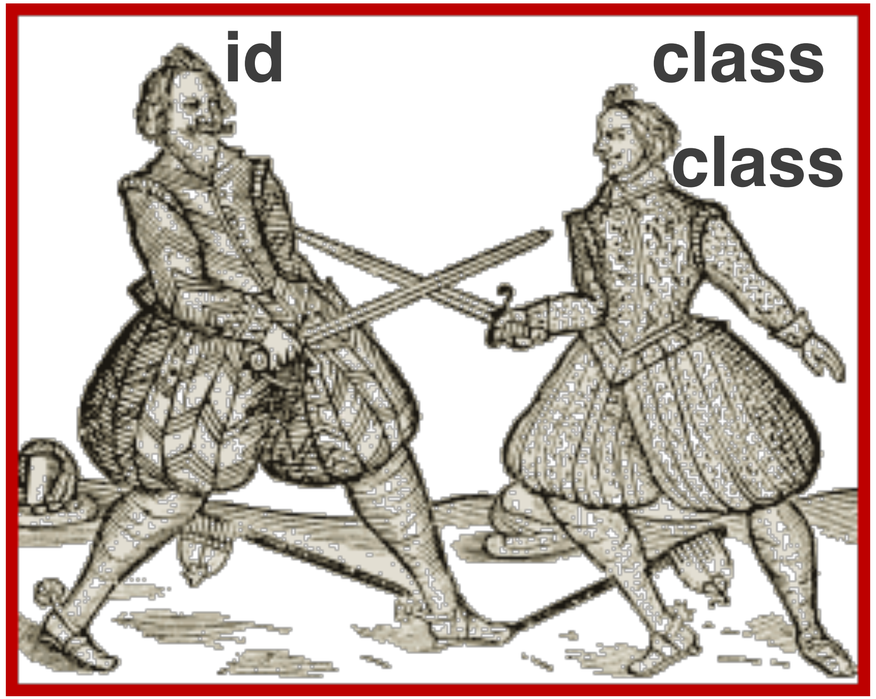
```
#thing {
  background: blue;
}
```




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

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

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




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


CSS
Global
HD

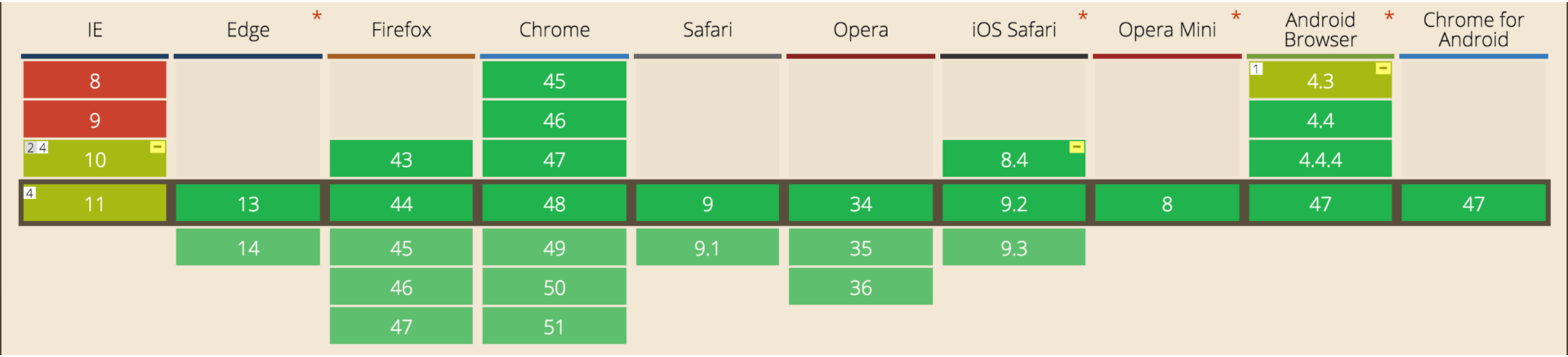
FLEXBOX

What is CSS Flexbox?

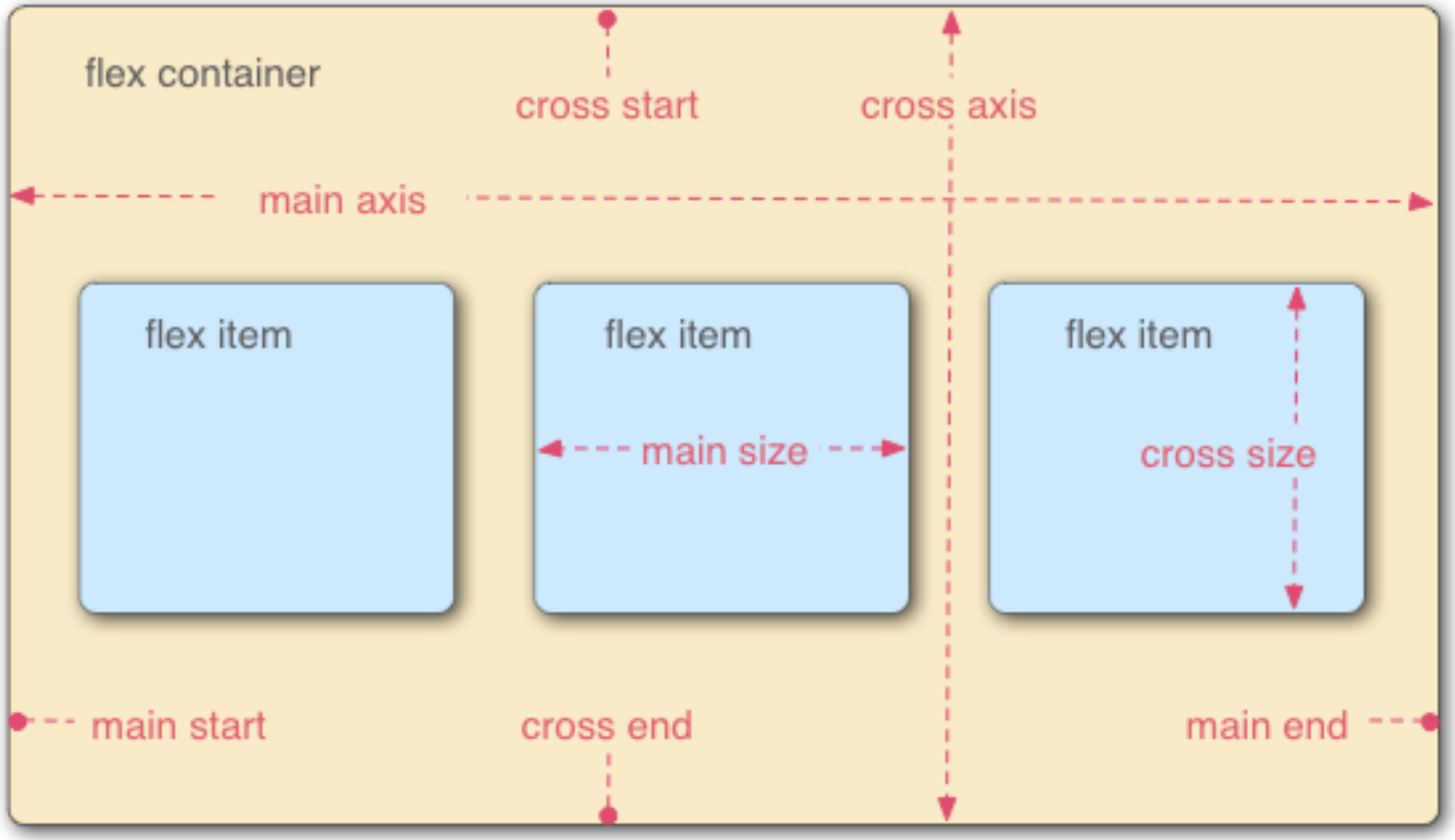
- **The Flexbox Layout module - more efficient way to**
  - lay out, align and distribute space among items in a container
  - even when the items' sizes are unknown and/or dynamic (thus the word "flex").

Can I use it?

caniuse.com



The Container



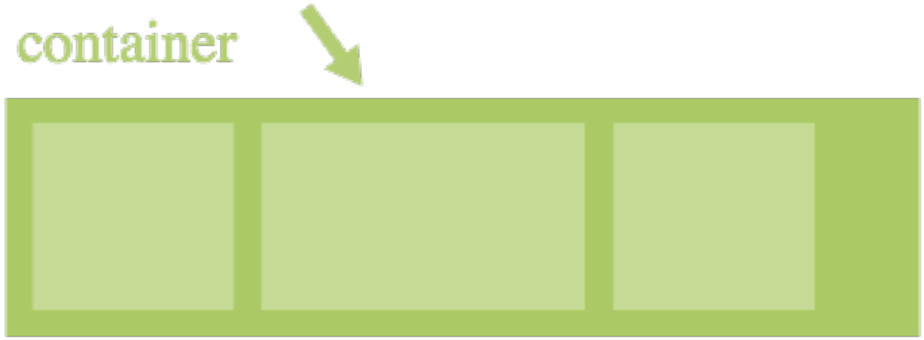
Flexbox Container: display

- This defines a flex container; It enables a flex context for all its direct children.

```
.container {
  display: flex; /* or inline-flex */
}
```

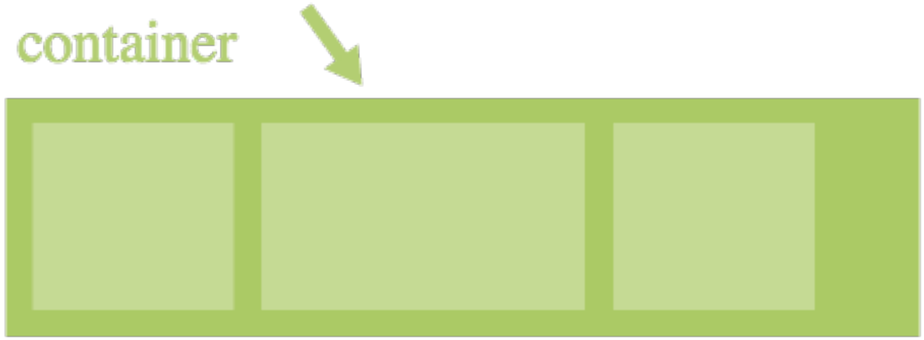
Flexbox Container

- Flexbox gives the container the ability to alter its items dimensions (and order) to best fill the available space.



Flexbox Container

- A flex container expands flexible items to fill free space, or shrinks them to prevent overflow.



Flexbox Container: flex-direction

- Flexbox is (aside from optional wrapping) a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.
- “main axis” and “cross axis”

Flexbox Container: Direction

- Think of flex items as primarily laying out either in horizontal rows or vertical columns.

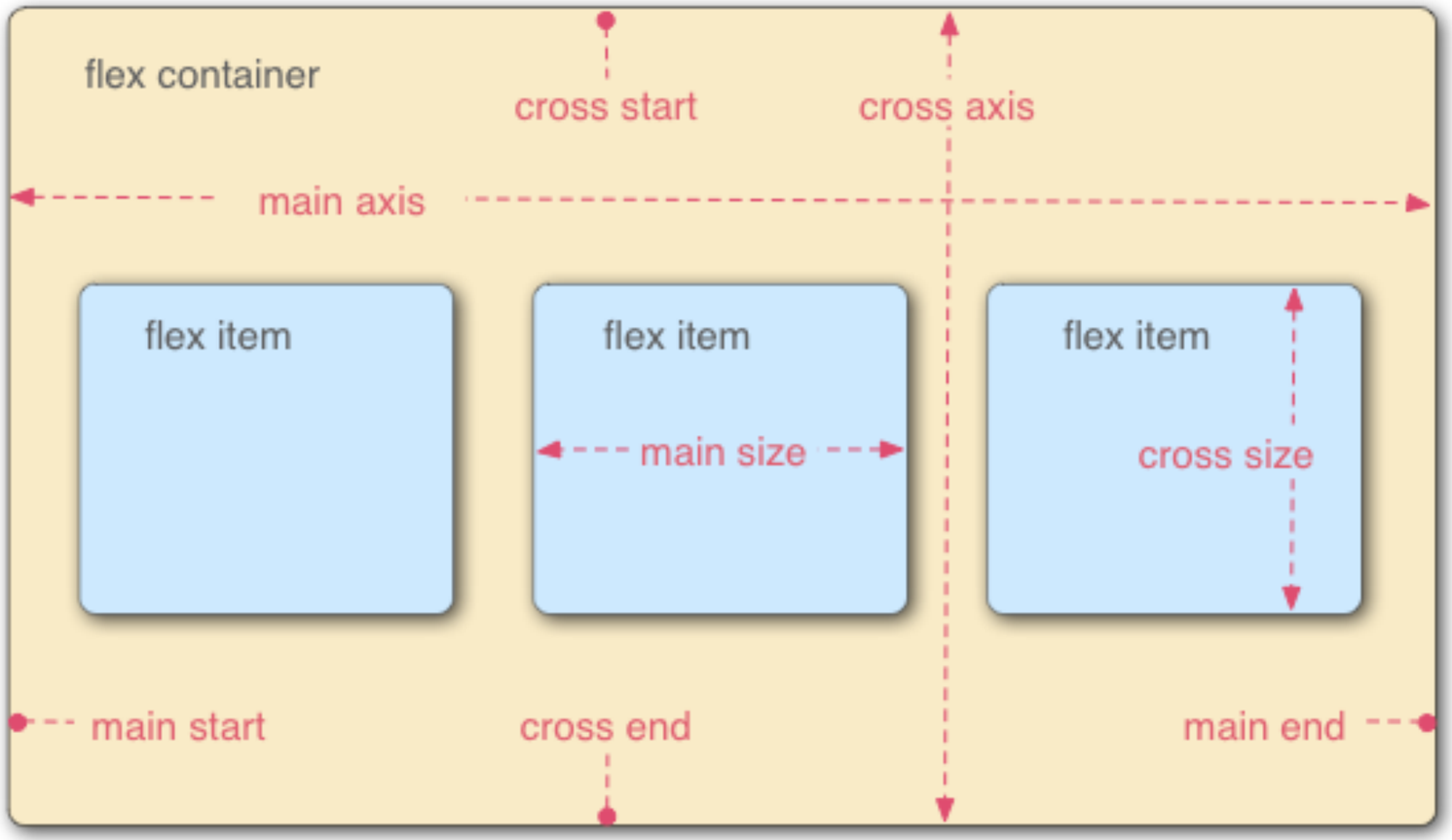

```
.container {
  flex-direction: row | row-reverse | column | column-reverse;
}
```

Flexbox Container: wrap

- Items will all try to fit onto one line. Items can wrap as needed with this property. Direction also plays a role here, determining the direction new lines are stacked in.

```
.container{
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```

The Items


flex container
cross start
cross axis
main axis
flex item
flex item
main size
flex item
cross size
main start
cross end
main end

Flexbox Items: flex-grow

- accepts a unitless value that serves as a proportion
- Dictates the proportion of space inside the flex container the item should take up


```
.item {
  flex-grow: <number>; /* default 0 */
}
```

Flexbox Items: flex-grow

- e.g. if all items have flex-grow set to 1, every child will set to an equal size inside the container.
- If one item is set to 2, that child would take up twice as much space as the others.


```
.item {
  flex-grow: <number>; /* default 0 */
}
```

Flexbox Items: flex-shrink

- This defines the ability for a flex item to shrink if necessary. Negative numbers are invalid.
- Not that necessary

```
.item {
  flex-shrink: <number>; /* default 1 */
}
```

Flexbox Items: flex-basis

- Like width property (or height, depending on flex-direction).
- If a relative value, indicates proportion of that item's width that should be applied.
- Default size of element before flex-grow or flex-shrink kick in

```
.item {
  flex-basis: <length> | auto; /* default auto */
}
```

Flexbox Items: flex

- This is the shorthand for flex-grow, flex-shrink and flex-basis combined. The second and third parameters (flex-shrink and flex-basis) are optional. Default is 0 1 auto.

```
.item {
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
}
```

<liveCode />

RESPONSIVE

RESPONSIVE


I heard you want to be a web developer
Here are a few devices to test your site

RESPONSIVE DESIGN

- Website is fully functional for all screen sizes, resolutions and orientations
- Born out of necessity (see previous slide)
- Developers and designers should cater to the user's environment, not the other way around

<liveCode />

Chrome Developer Tools