Styling React Components

How to Escape from Selector Hell

Charlie Marsh
Khan Academy
July 9, 2014

“My only beef with CSS is the entire idea of ‘selectors’. If we could get rid of those, [CSS would be] much more predictable [and] easy.”

- Pete Hunt, React Guru

Selectors

“Patterns used to select the element(s) you want to style.”

Selector:

```
div { ... }
```

Applies to:

```
<div>
    ...
</div>
```

Selector:

```
div p { ... }
```

Applies to:

```
<div>
    <p>
        ...
    </p>
</div>
```

“Wow, selectors seem really cool!”

Selectors: Extended

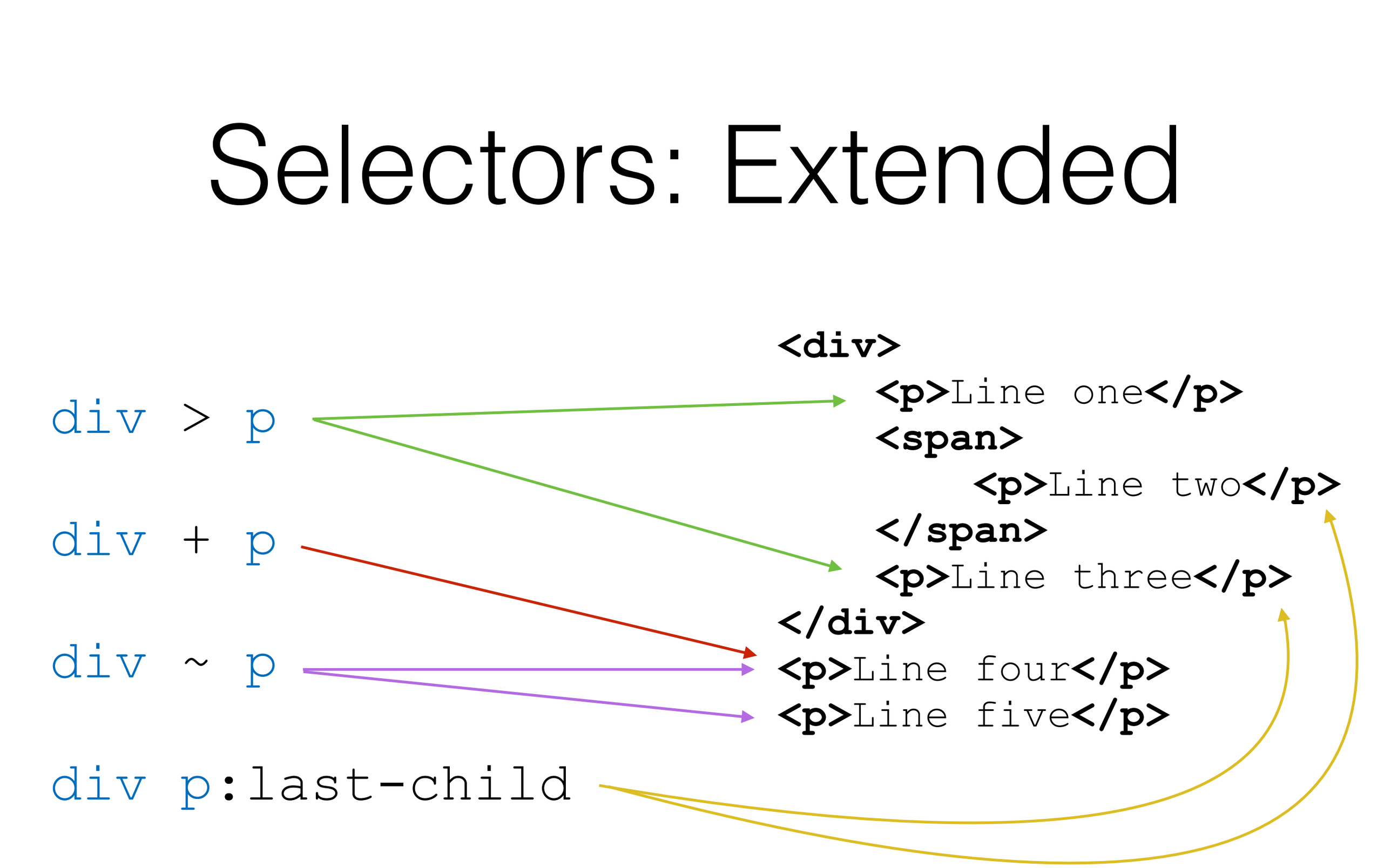
```
div > p

div + p

div ~ p

div p:last-child
```

```
<div>
    <p>Line one</p>
    <span>
        <p>Line two</p>
    </span>
    <p>Line three</p>
</div>
<p>Line four</p>
<p>Line five</p>
```



```
div.foo ~ div.bar:after > span:last-of-type + a:visited
```

“But Charlie, no one *really* does that!”

“But Charlie, no one I *work with* really does that!”

Selectors @ KA

```
.framework-thing .paragraph > ul:not(.thing-widget-radio)
```

```
input[type="radio"]:checked + span:before
```

```
.thing-content > .container .thing-content-view-root > .content-pane-inner .main-header > .topic-info > .topic-icon
```

1. *You don't know what will happen when…*
   you actually load your webpage.

2. *You don't know what will happen when…*
   you change **anything** in the DOM.

3. *You don't know what will happen when…*
   you change **anything** in your stylesheet.

Result:

Selectors make it very, very difficult to refactor.

Preprocessors

- Preprocessors *do* make CSS easier
  - Solve the “unpredictable cascading” problem
- What do they do wrong?

```
#main {
   width: 97%;

   p, div {
      font-size: 2em;
      a { font-weight: bold; }
   }

   pre { font-size: 3em; }
}
```

```
body {
    div.container {
        div.content {
            div.articles {
                & > div.post {
                    div.title {
                        h1  {
                            a  {
                            }
                        }
                    }
                }
```

“Wow, this is a mess! What can we do?”

- Audience member

Back to React

- Aiming for:
  - Modularity
  - Composability
  - Maintainability
- Stylesheets and selectors can poison your components

Solution #1: Stylesheets as Dependencies

- **Goal**: Make React component <—> stylesheet relationship explicit

```
// Applying a stylesheet
var css = require("style!css!./file.css");

// Chaining loaders
var css = require("style!css!less!./file.less");
```

- Implemented in Webpack (Browserify alternative)

Solution #2: Inlining

- **Goal**: Put *all* styling in your JavaScript file

```
var styles = $styleSheet.create({
  base: {
    width: 38,
    height: 38
  }
});
```

```
<span style={styles.base} />
```

Solution #3: RCSS

- **Goal**: Convert JS style objects into CSS classes

```
var button = {
  padding: '6px 12px',
};

// Add class to HTML page
RCSS.createClass(button);
```

```
<button className={button.className}>
```

Progress

- Joel moved react-components over to RCSS
- Charlie moved three Perseus widgets (+ some miscellaneous components) over to RCSS
- Scattered development taking place on our own fork (Khan/RCSS)

LESSons

1. Don’t be afraid to style from *within* your JavaScript file

2. Avoid adding deeply nested selectors (follow the *Inception Rule*)

3. Avoid mimicking the DOM

Result:

React components as they’re meant to be.