IDV460

INTERACTIVE DATA VIZ SPRING 16

UNDERSTANDING THE DOM

WHAT IS THE DOM?

When the browser loads a web page, it creates a “model” of that page in memory. The **Document Object Model** (DOM) specifies how the browser should structure this model using a DOM tree.

The **D** in DOM stands for **Document** — the HTML page itself.

The **O** in DOM stands for **Object** — because the model of your page is made of objects.

The **M** in DOM stands for **Model** — and that is how we shall think about your HTML page for the purpose of using scripts on it.

WHAT DOES THE DOM DO?

The DOM defines methods and properties to access and update each object in the model — which in turn, changes what the user sees on the page.

The DOM states what your script can ask the browser about the HTML page, and tell the browser how to update the page.

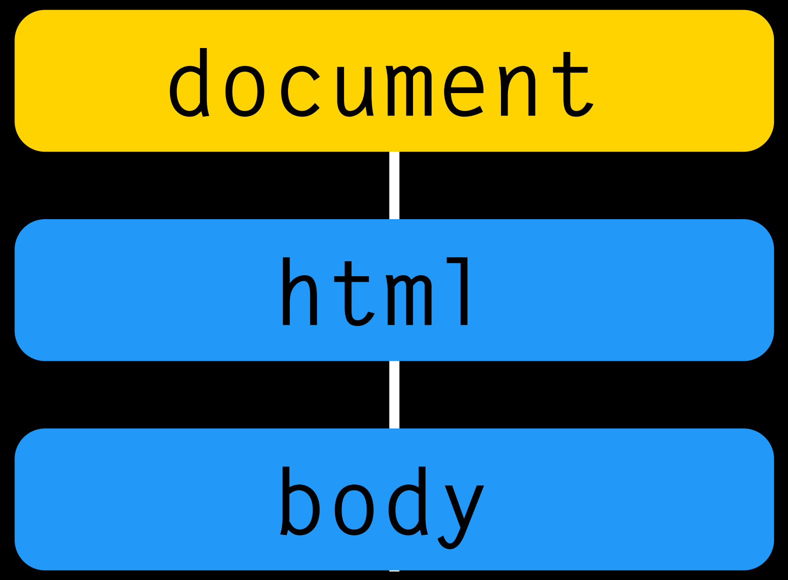
DOM TREE

Let's look at a simple HTML page, then translate it into a DOM tree.

```html
<html>
<body>
   <header>
      <h1>IDV<span>460</span></h1>
      <p>Interactive Data Viz Spring 16</p>
   </header>
   <div id="page-wrap">
      <nav>
         <ul>
            <li id="home" class="one">Home</li>
            <li id="inclass" class="one">Class</li>
            <li id="projects" class="two">Projects</li>
         </ul>
      </nav>
      <main>
         <h2>About me.</h2>
         <p>Some <em>type about me</em> goes here.</p>
      </main>
   </div> <!-- closes page-wrap -->
</body>
</html>
```

DOM TREE

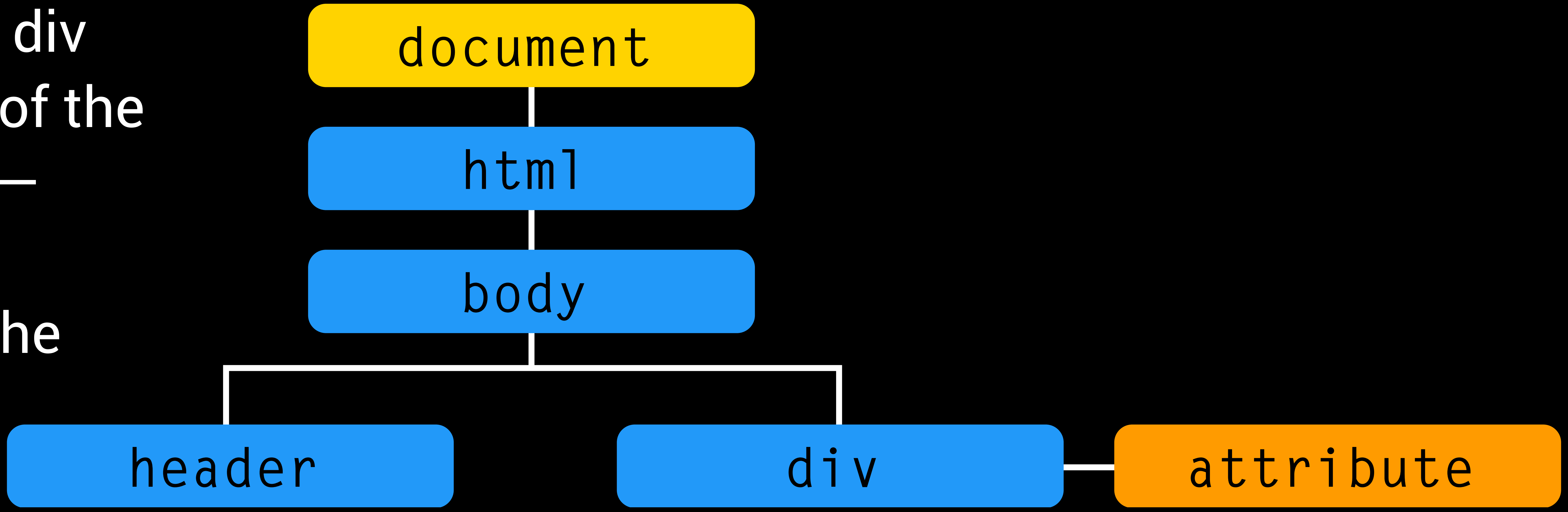
Every element, attribute and piece of text in the HTML is represented in the DOM tree by its own DOM **node**. At the top, a document node is added, which represents the entire page, then the nested elements (objects) within — the html node and body node.



```
<html>
<body>
    <header>
        <h1>IDV<span>460</span></h1>
        <p>Interactive Data Viz Spring 16</p>
    </header>
    <div id="page-wrap">
        <nav>
            <ul>
                <li id="home" class="one">Home</li>
                <li id="inclass" class="one">Class</li>
                <li id="projects" class="two">Projects</li>
            </ul>
        </nav>
        <main>
            <h2>About me.</h2>
            <p>Some <em>type about me</em> goes here.</p>
        </main>
    </div> <!-- closes page-wrap -->
</body>
</html>
```

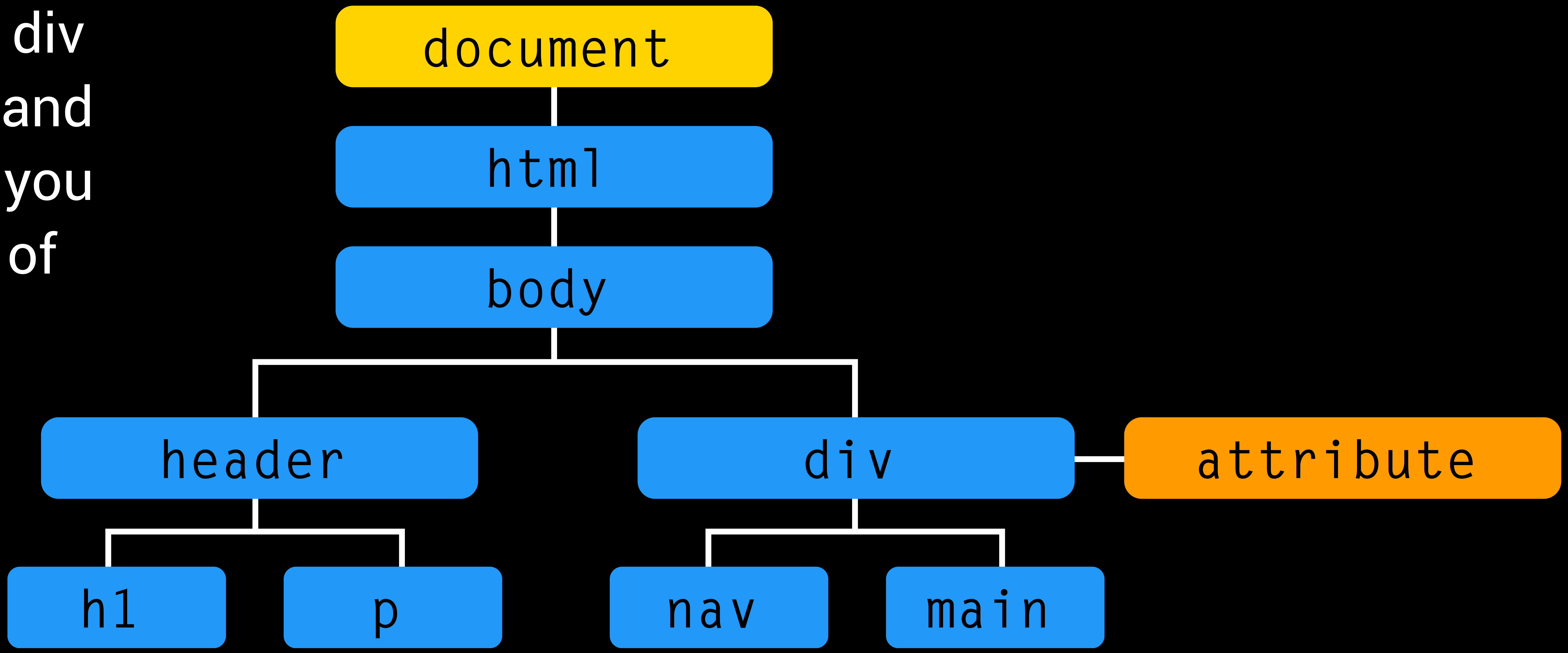
DOM TREE

From there, the header and div are *children* of the body object. The div has an attribute — for the sake of the DOM tree, usually a class or ID — while the header does not. An **attribute node** is not a *child* of the element that carries it, but a *part* of that element.



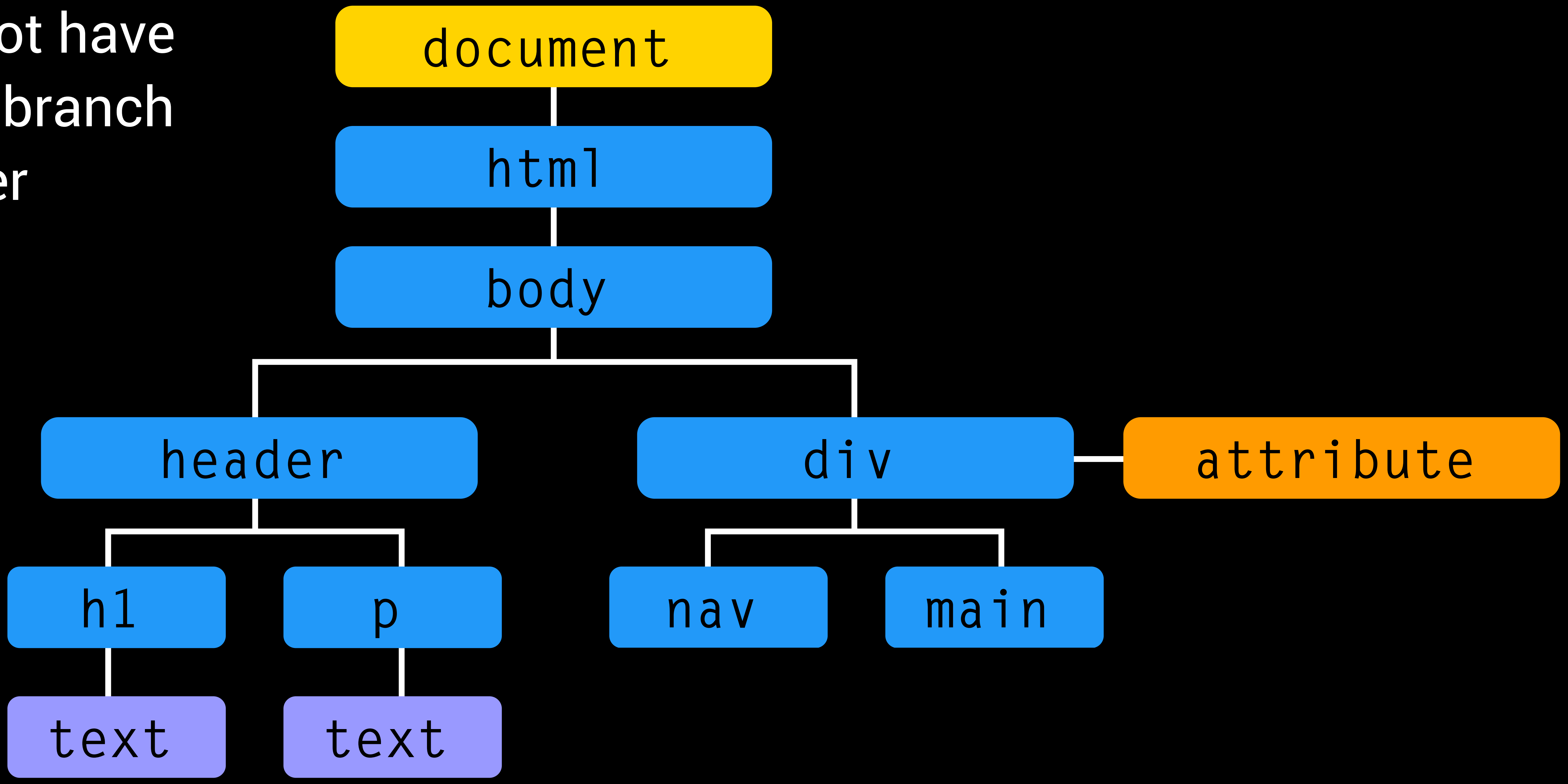
DOM TREE

The header has two nested objects — an h1 and a paragraph. The div has two children, too: the nav and main objects. The HTML tags you see here in blue are examples of **element nodes**.



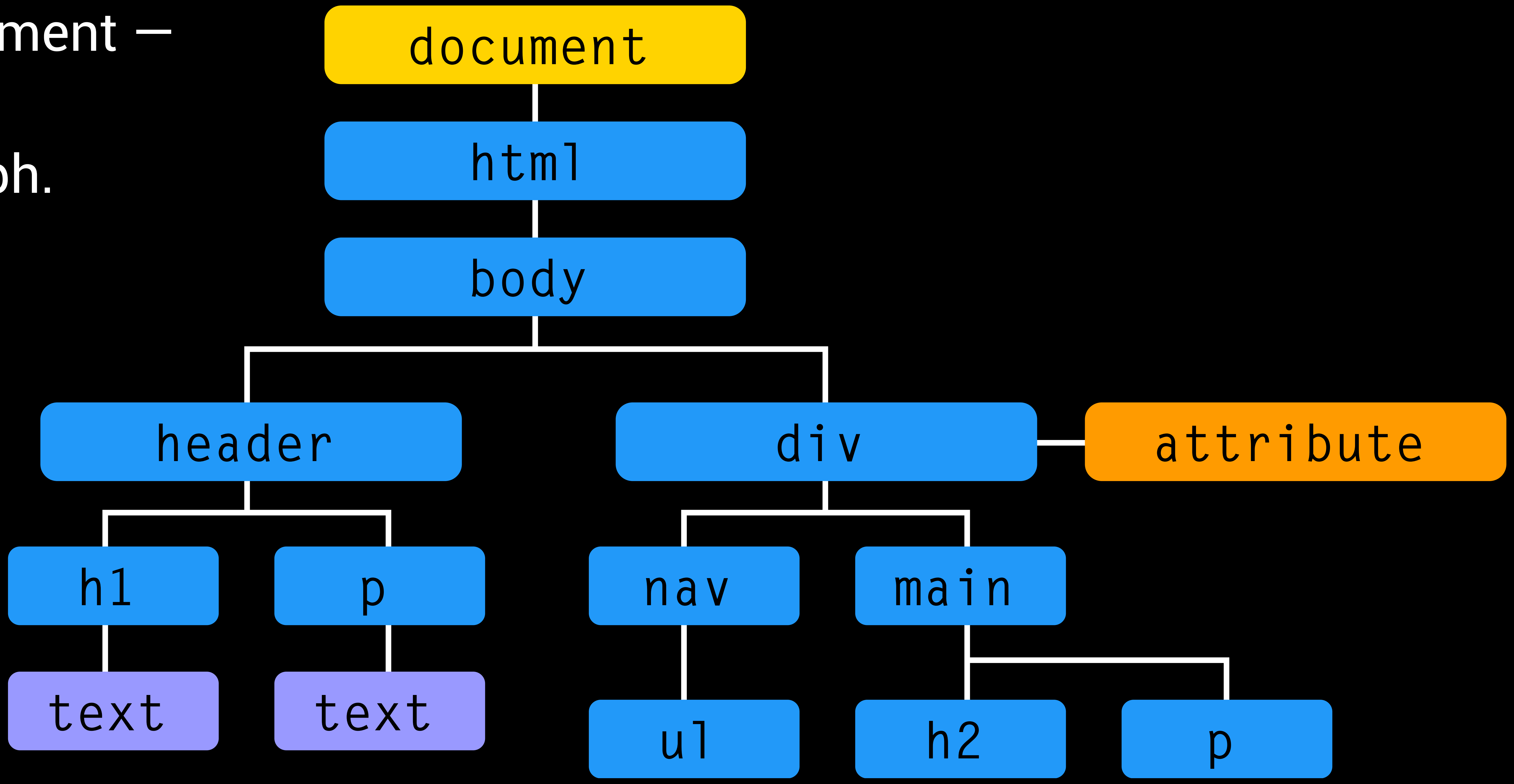
DOM TREE

The two tags in the main object both contain text. Text nodes cannot have children. A **text node** is a new branch on the DOM tree, but no further branches can come from it.



DOM TREE

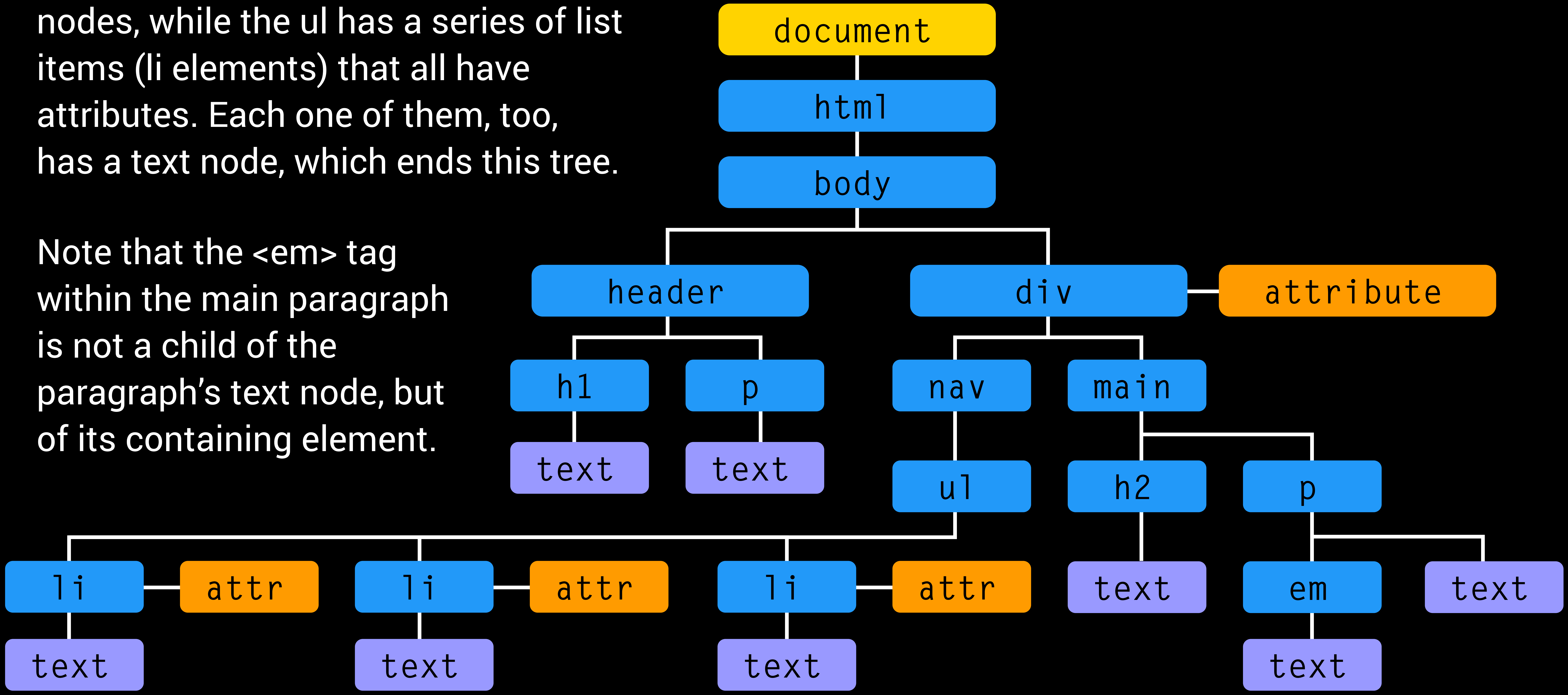
Within the div branch of our DOM tree, the nav has one child element — the ul — while the main object includes an h2 and a paragraph.



DOM TREE

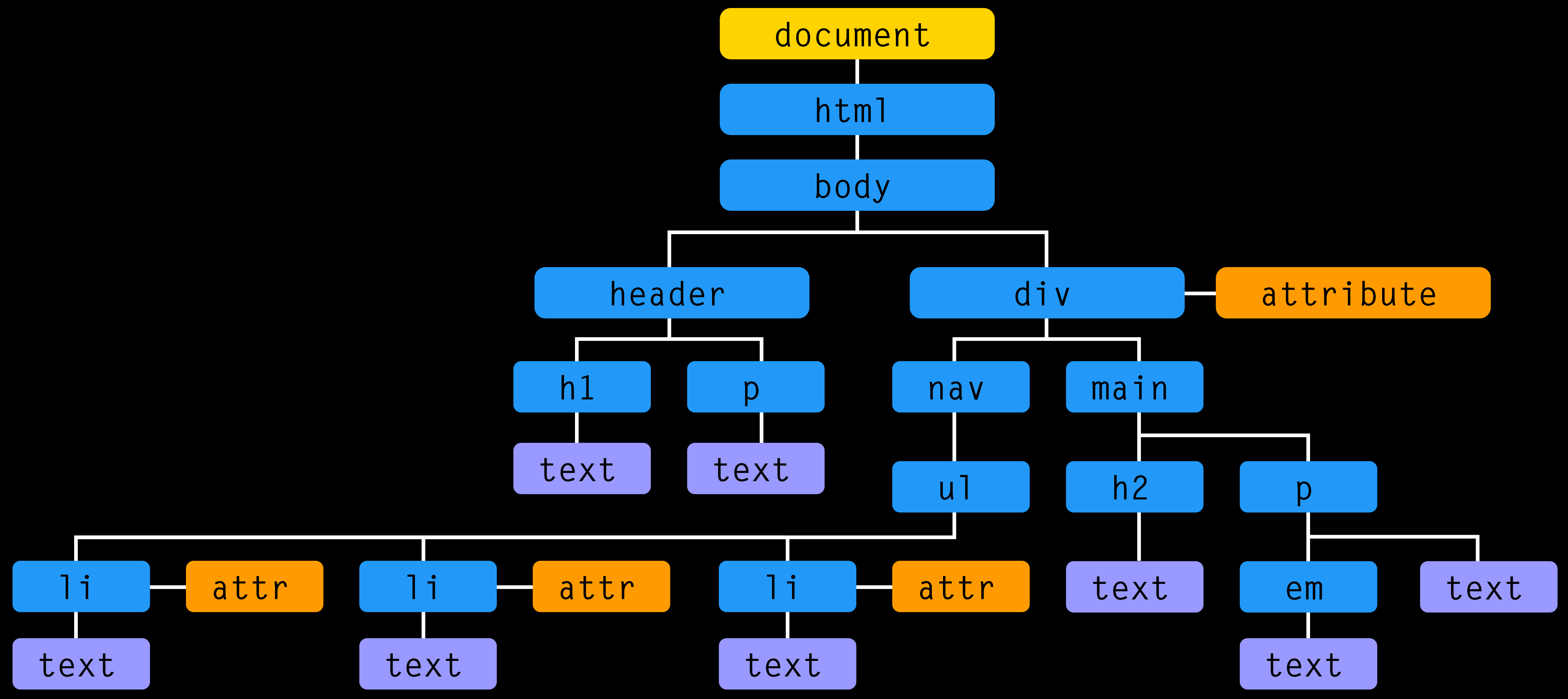
The h2 and paragraph tags have text nodes, while the ul has a series of list items (li elements) that all have attributes. Each one of them, too, has a text node, which ends this tree.

Note that the <em> tag within the main paragraph is not a child of the paragraph's text node, but of its containing element.



DOM TREE

Here is how the DOM tree compares with the HTML page.

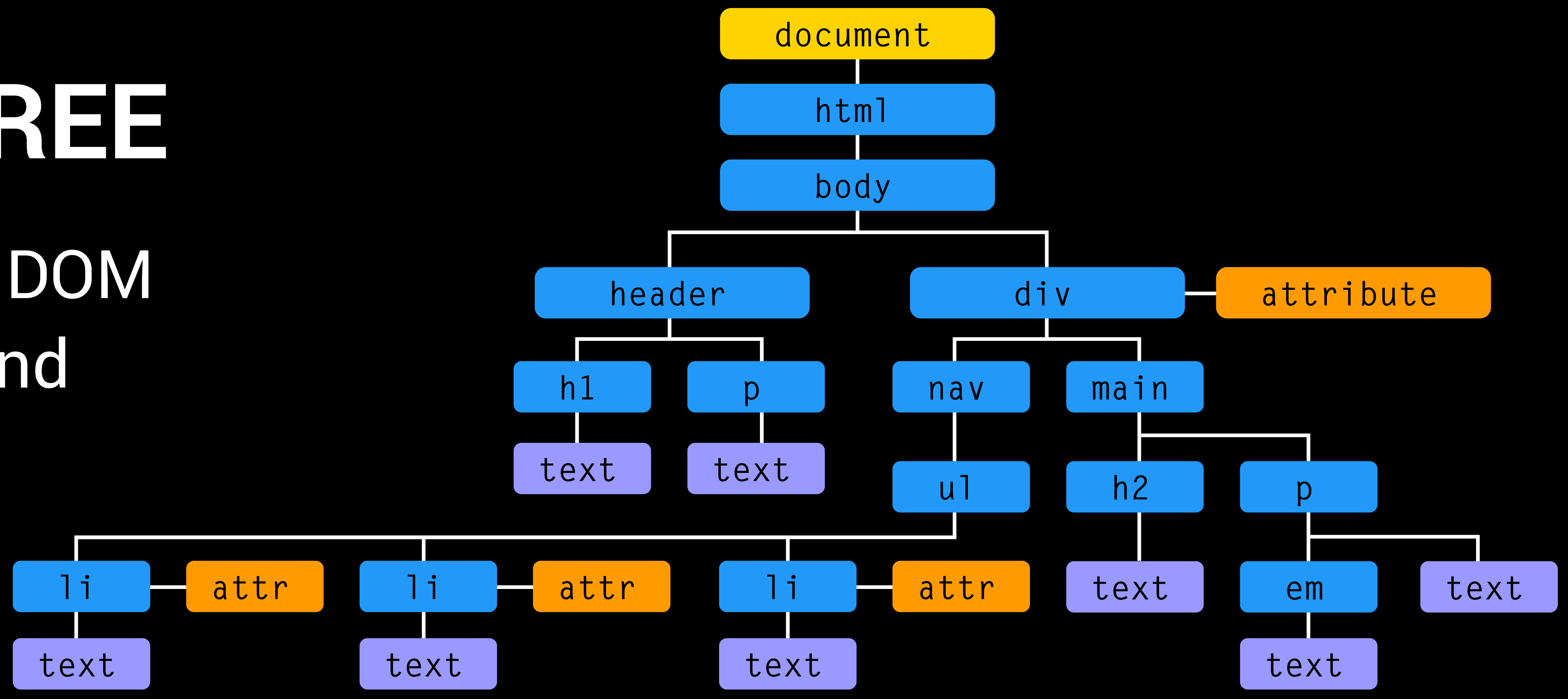


```
<html>
<body>
    <header>
        <h1>IDV<span>460</span></h1>
        <p>Interactive Data Viz Spring 16</p>
    </header>
    <div id="page-wrap">
        <nav>
            <ul>
                <li id="home" class="one">Home</li>
                <li id="inclass" class="one">Class</li>
                <li id="projects" class="two">Projects</li>
            </ul>
        </nav>
        <main>
            <h2>About me.</h2>
            <p>Some <em>type about me</em> goes here.</p>
        </main>
    </div> <!-- closes page-wrap -->
</body>
</html>
```

WORKING WITH THE DOM TREE

You need to first access the elements of the DOM tree before you can do something to them, and update the page.

There are several ways to select elements:



```
<html>
<body>
    <header>
        <h1>IDV<span>460</span></h1>
        <p>Interactive Data Viz Spring 16</p>
    </header>
    <div id="page-wrap">
        <nav>
            <ul>
                <li id="home" class="one">Home</li>
                <li id="inclass" class="one">Class</li>
                <li id="projects" class="two">Projects</li>
            </ul>
        </nav>
        <main>
            <h2>About me.</h2>
            <p>Some <em>type about me</em> goes here.</p>
        </main>
    </div> <!-- closes page-wrap -->
</body>
</html>
```

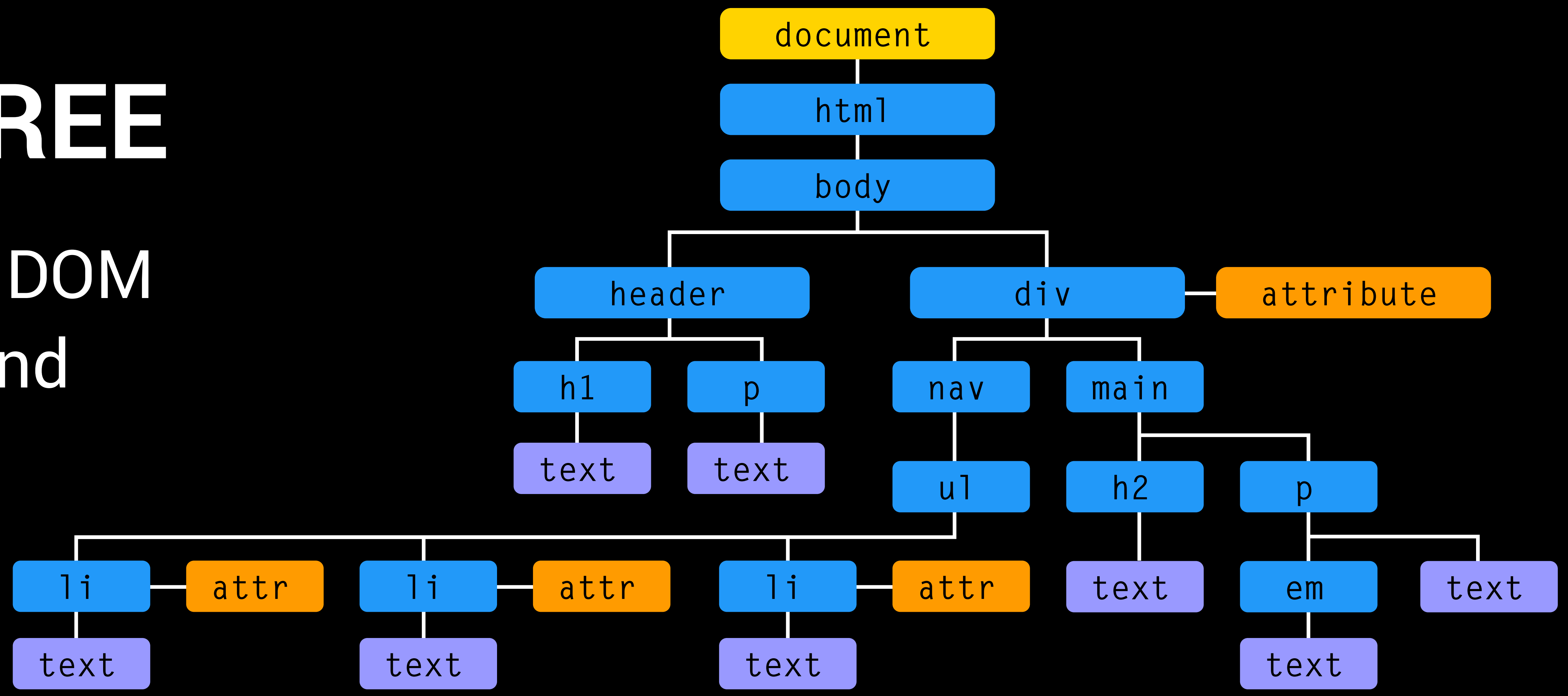
WORKING WITH THE DOM TREE

You need to first access the elements of the DOM tree before you can do something to them, and update the page.

There are several ways to select elements:

```
document.getElementById('home');
```

This uses the value of the element's ID attribute, which should be unique within the page.



```
<html>
<body>
     <header>
          <h1>IDV<span>460</span></h1>
          <p>Interactive Data Viz Spring 16</p>
     </header>
     <div id="page-wrap">
          <nav>
               <ul>
                    <li id="home" class="one">Home</li>
                    <li id="inclass" class="one">Class</li>
                    <li id="projects" class="two">Projects</li>
               </ul>
          </nav>
          <main>
               <h2>About me.</h2>
               <p>Some <em>type about me</em> goes here.</p>
          </main>
     </div> <!-- closes page-wrap -->
</body>
</html>
```

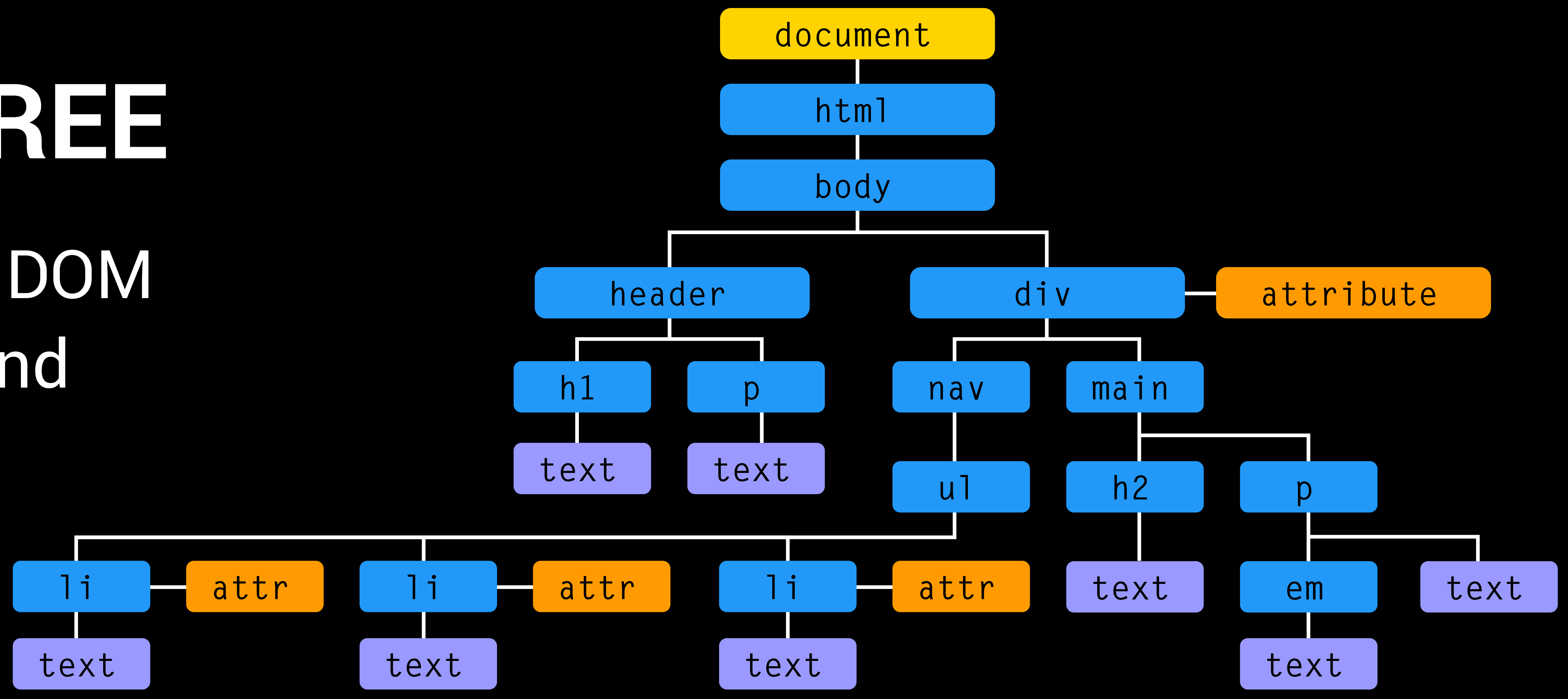
WORKING WITH THE DOM TREE

You need to first access the elements of the DOM tree before you can do something to them, and update the page.

There are several ways to select elements:

```
document.getElementById('home');
document.getElementsByClassName('one');
```

This selects all elements that have the specified value for their class attribute.



```
<html>
<body>
    <header>
        <h1>IDV<span>460</span></h1>
        <p>Interactive Data Viz Spring 16</p>
    </header>
    <div id="page-wrap">
        <nav>
            <ul>
                <li id="home" class="one">Home</li>
                <li id="inclass" class="one">Class</l
                <li id="projects" class="two">Project
            </ul>
        </nav>
        <main>
            <h2>About me.</h2>
            <p>Some <em>type about me</em> goes here.<
        </main>
    </div> <!-- closes page-wrap -->
</body>
</html>
```

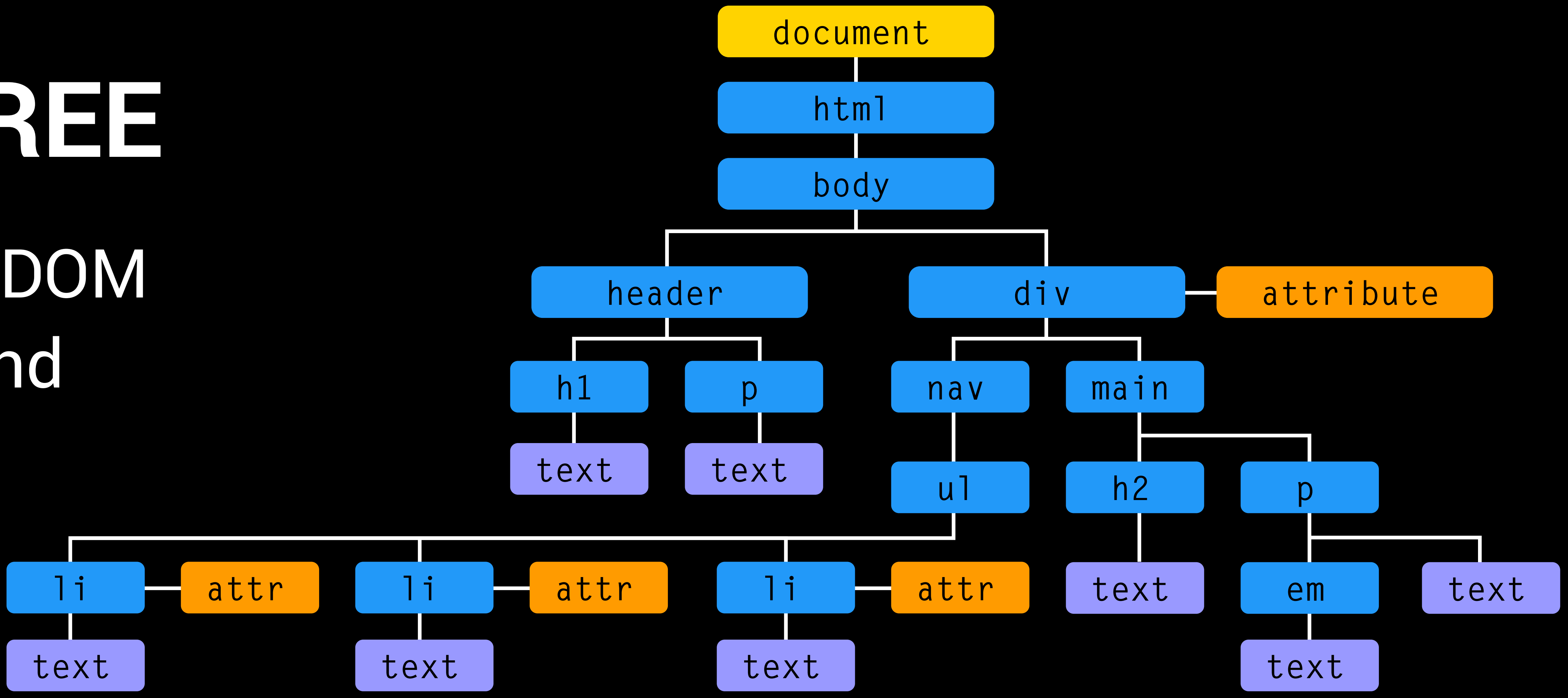
WORKING WITH THE DOM TREE

You need to first access the elements of the DOM tree before you can do something to them, and update the page.

There are several ways to select elements:

```
document.getElementById('home');
document.getElementsByClassName('one');
document.getElementsByTagName('li');
```

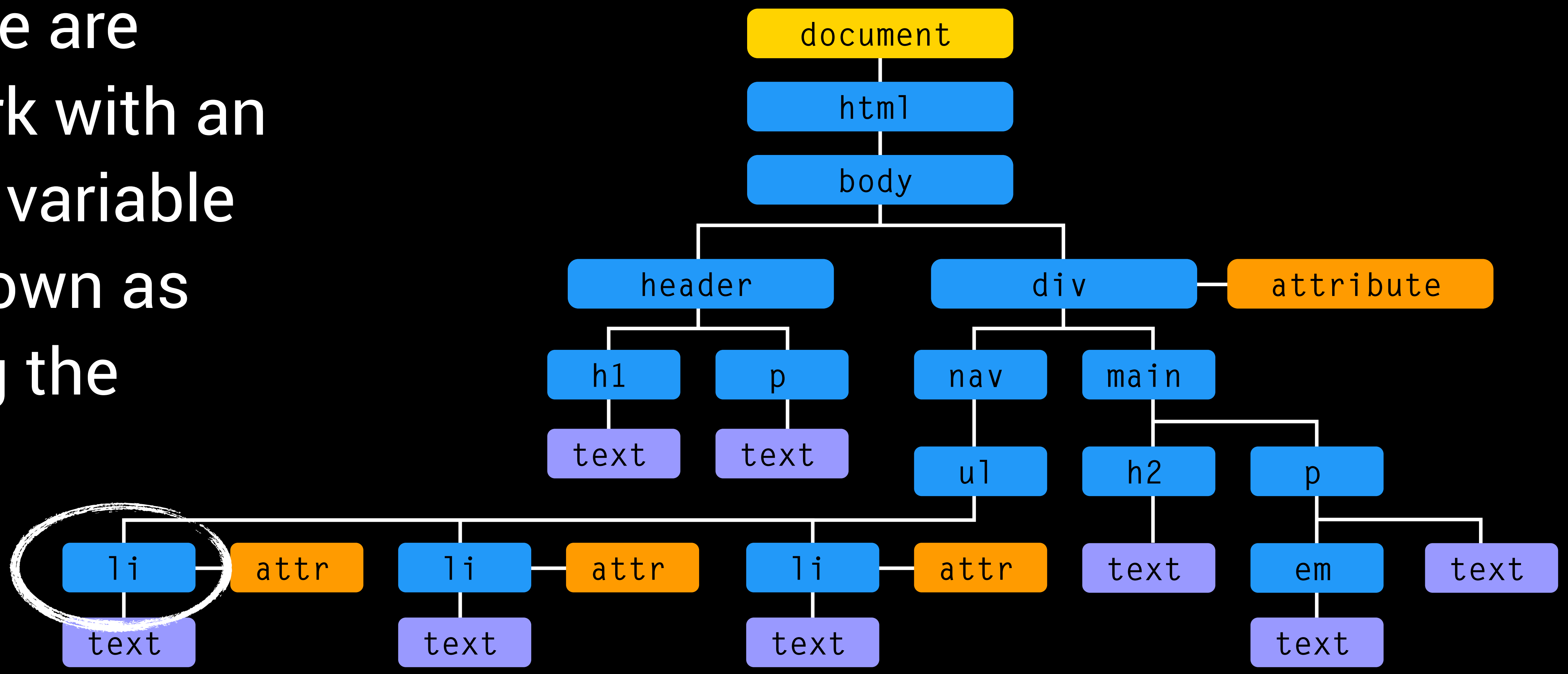
This selects all elements that have the specified tag name.



```
<html>
<body>
     <header>
          <h1>IDV<span>460</span></h1>
          <p>Interactive Data Viz Spring 16</p>
     </header>
     <div id="page-wrap">
          <nav>
               <ul>
                    <li id="home" class="one">Home</li>
                    <li id="inclass" class="one">Class</l
                    <li id="projects" class="two">Project
               </ul>
          </nav>
          <main>
               <h2>About me.</h2>
               <p>Some <em>type about me</em> goes here.
          </main>
     </div> <!-- closes page-wrap -->
</body>
</html>
```

WORKING WITH THE DOM TREE

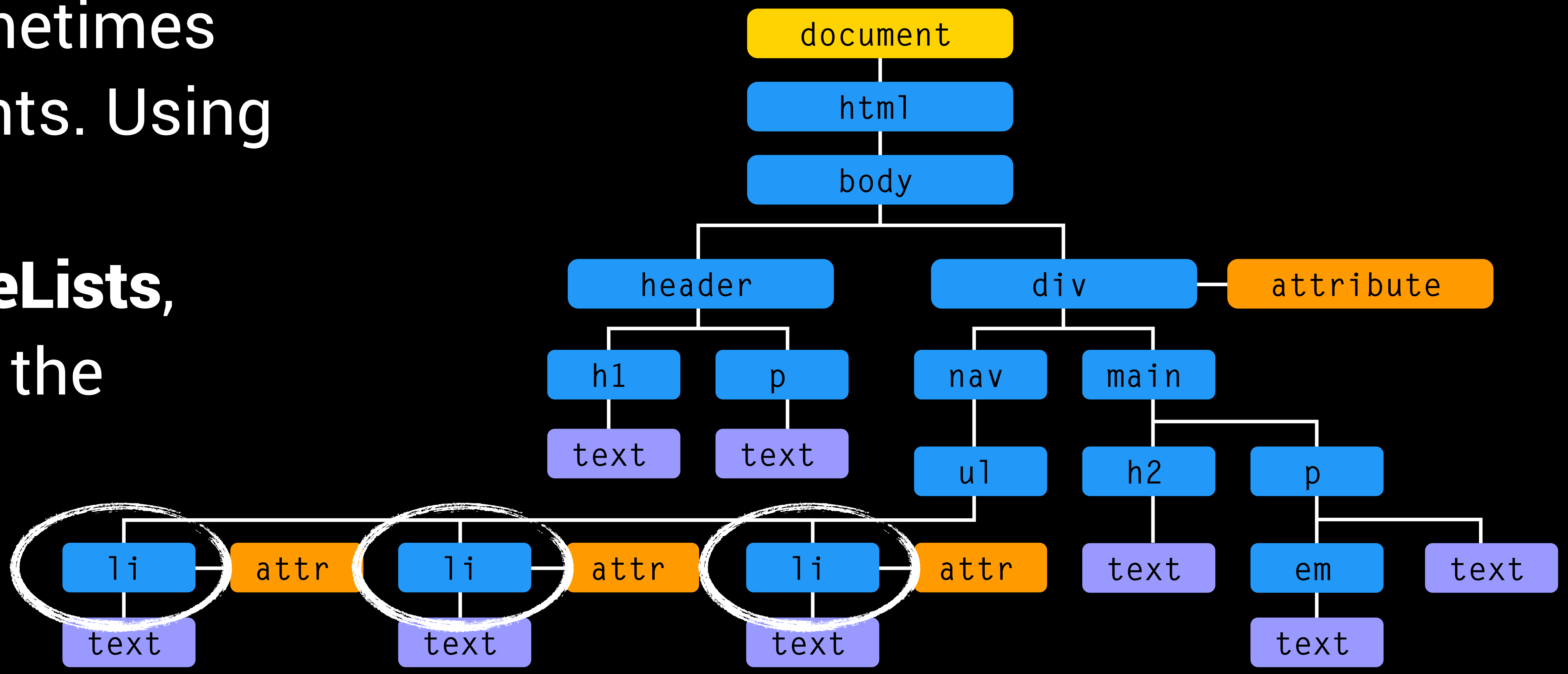
Methods that find elements in the DOM tree are called DOM queries. When you need to work with an element more than once, you should use a variable to store the result of your query. This is known as **“caching”** the selection. (It is in fact storing the location of the node in the DOM tree.)



```
var homeLink = document.getElementById('home');
```

WORKING WITH THE DOM TREE

This example selects one element, but sometimes you may want to change a group of elements. Using the methods getElementsByClassName or getElementsByTagName will produce **NodeLists**, even if there only one matching element in the HTML.



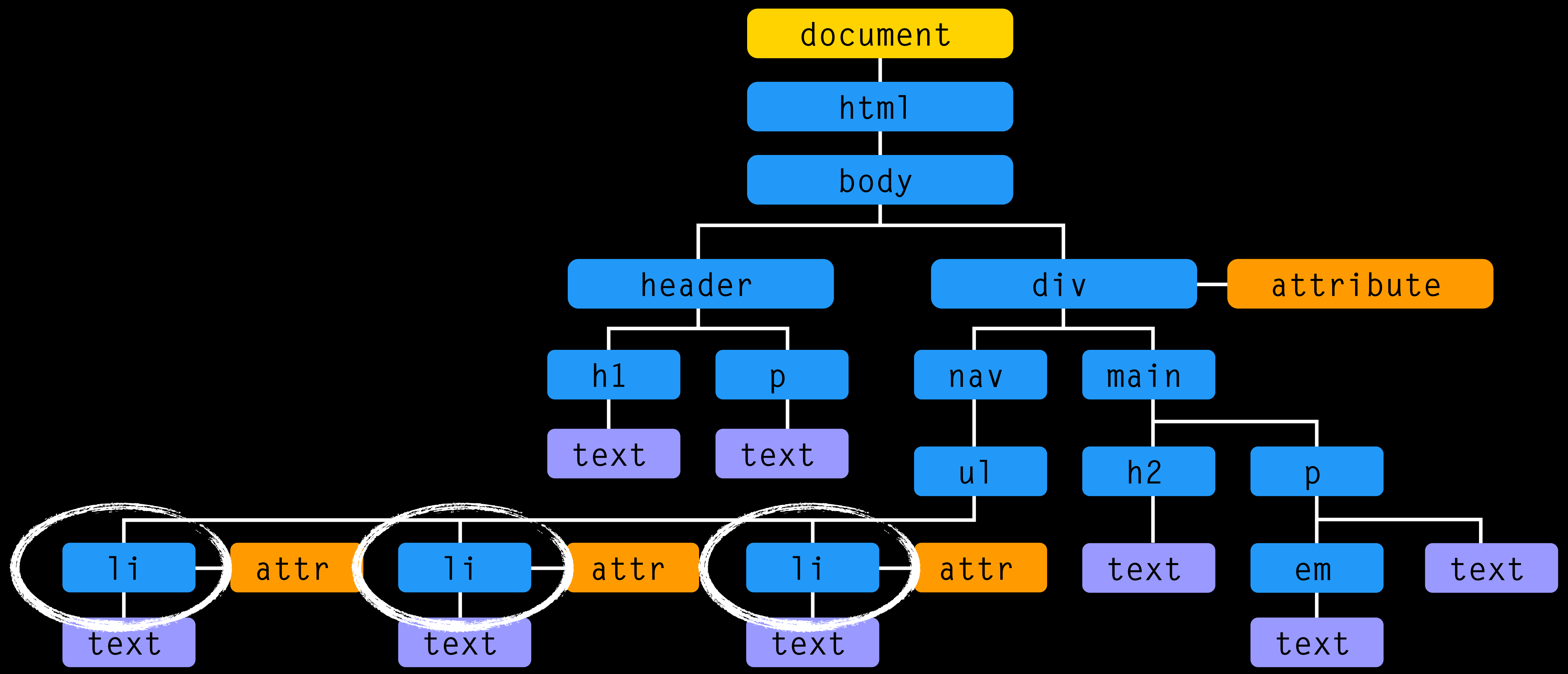
```
var menuItems = document.getElementsByTagName('li');
```

This method returns three elements on *our* page.

```
<li id="home" class="one">Home</li>
<li id="inclass" class="one">Class</li>
<li id="projects" class="two">Projects</li>
```

WORKING WITH THE DOM TREE

Each node is given an index number (that is, one that starts with zero rather than 1, as with an array).
The order is the same as the order they appear in the HTML.



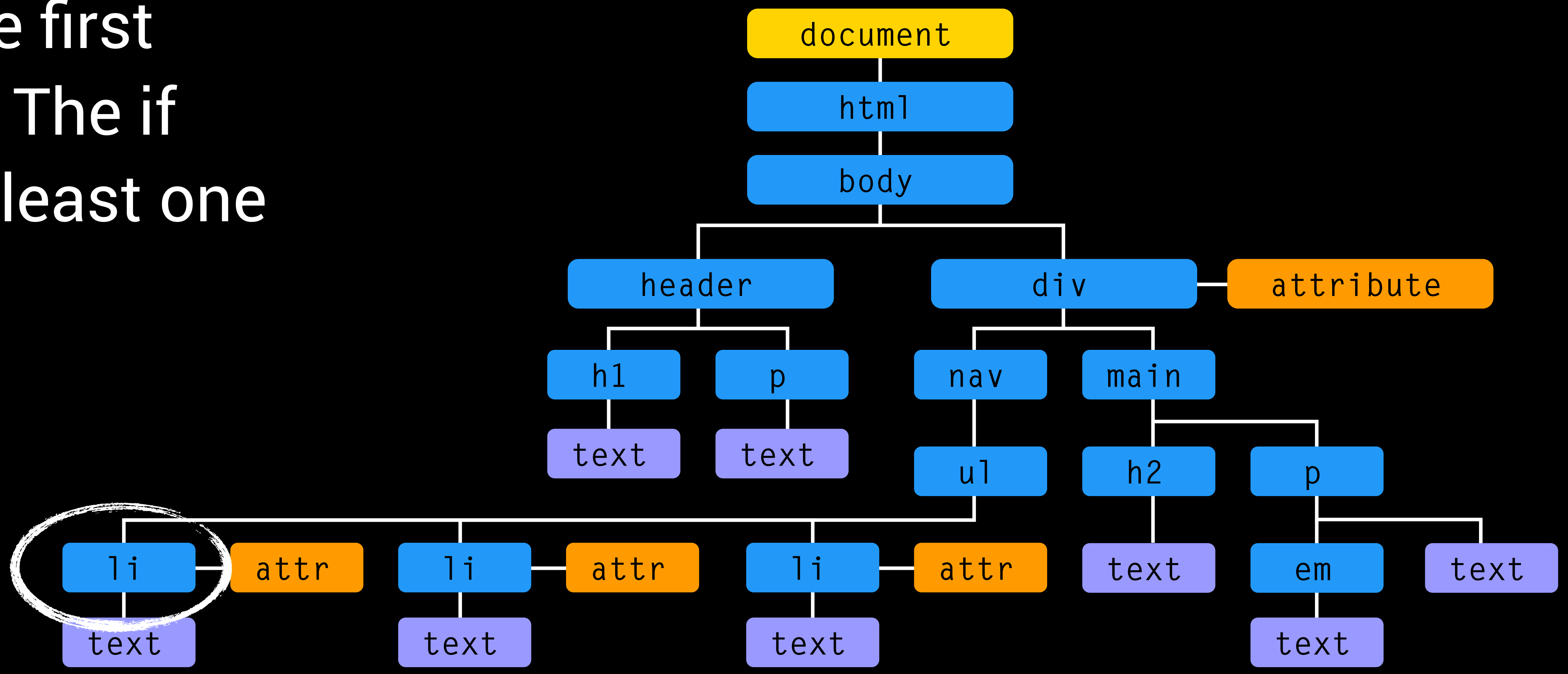
```
var menuItems = document.getElementsByTagName('li');
```

The index numbers can be used to specify just one of these elements.

```
0  <li id="home" class="one">Home</li>
1  <li id="inclass" class="one">Class</li>
2  <li id="projects" class="two">Projects</li>
```

WORKING WITH THE DOM TREE

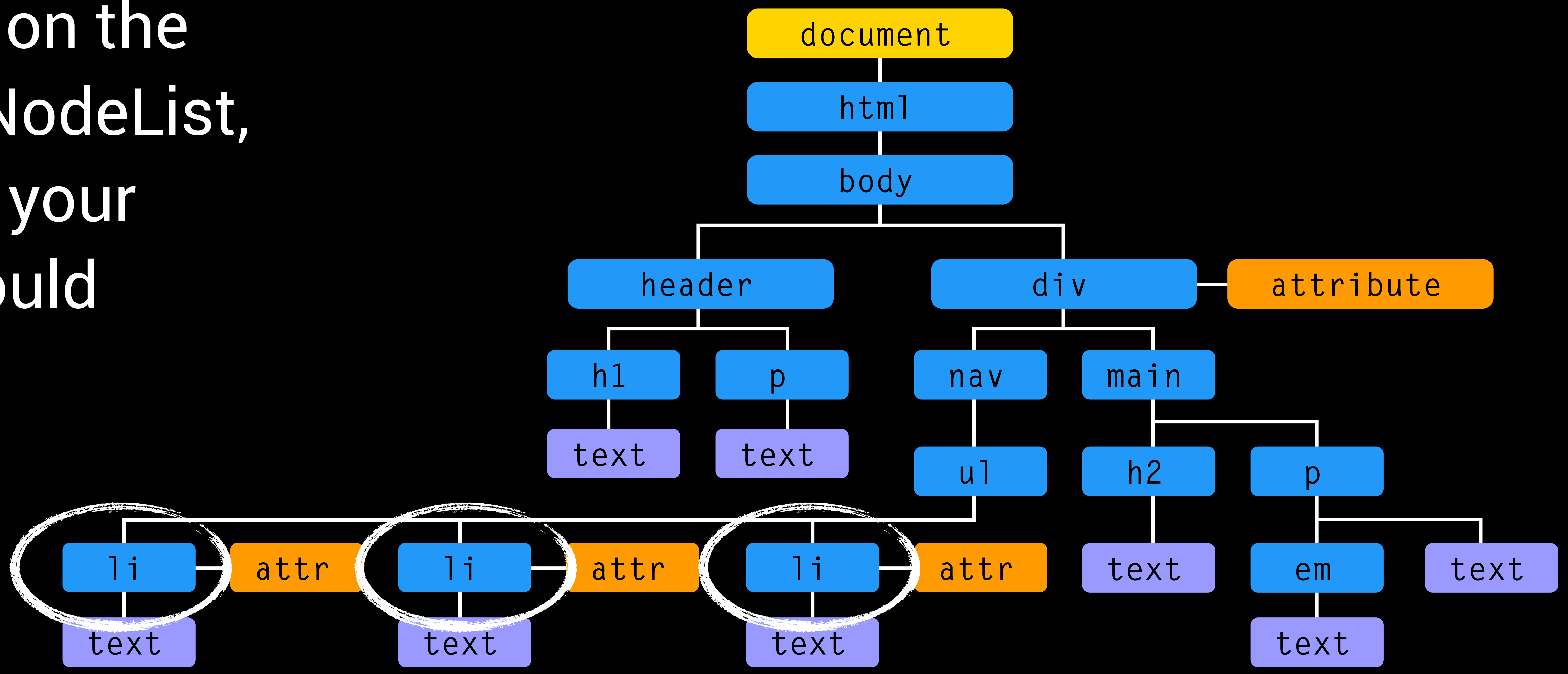
This piece of code would designate just the first item among the <li> elements on the page. The if statement is used to make sure there is at least one li in the NodeList.



```
var menuItems = document.getElementsByTagName('li');
if menuItems.length >= 1 {
 firstItem = menuItem [0];
}
```

WORKING WITH THE DOM TREE

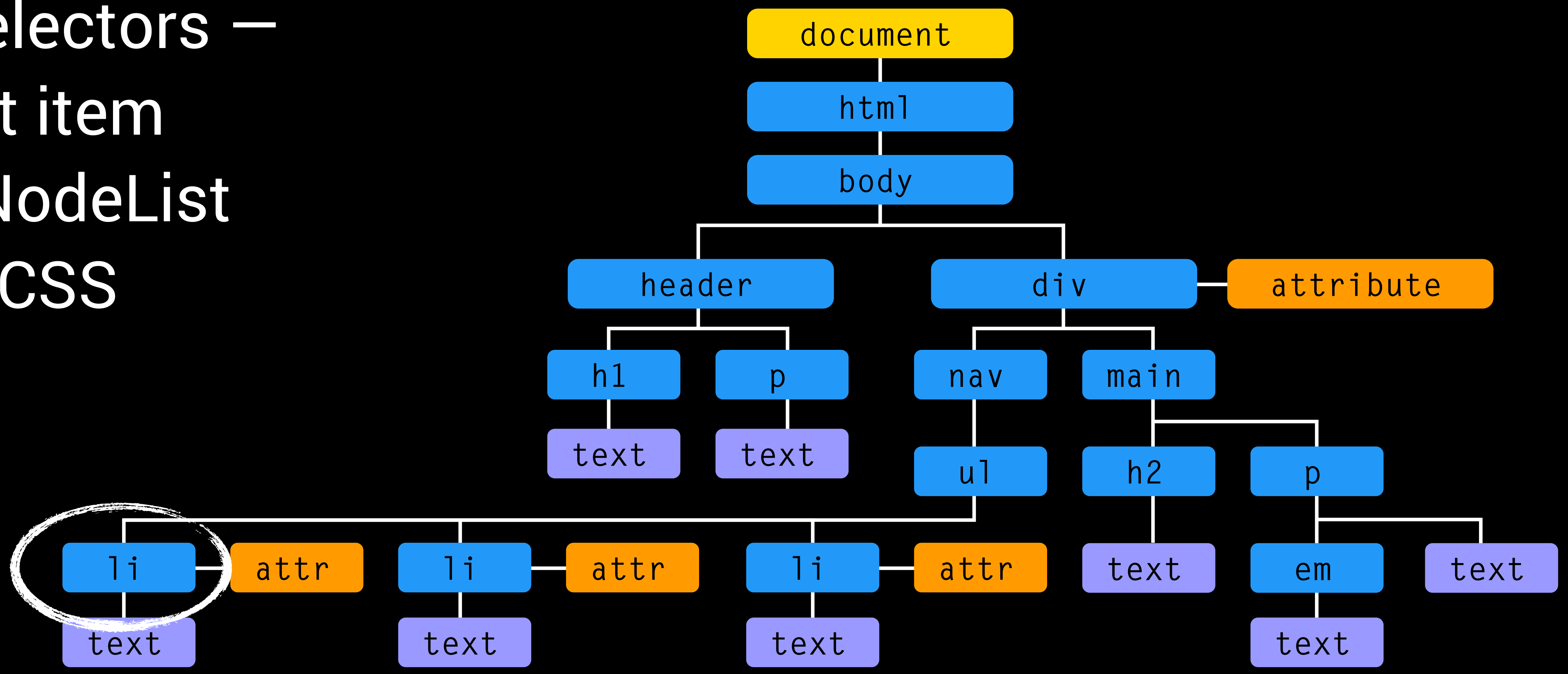
If you wanted to change all the li elements on the page, you would have to loop through the NodeList, no matter how many list items you have in your document. Here, the enclosing function would change the color of list item text.



```
var menuItems = document.getElementsByTagName('li');
for (var i=0; i<menuItems.length; i++) {
 menuItems[i].style.color = rgb(211,255,0);
}
```

WORKING WITH THE DOM TREE

You can also select elements using CSS selectors — either a single item (though always the first item that matches the CSS-style selector) or a NodeList of all the matches. Both methods take the CSS selector as their only parameter.

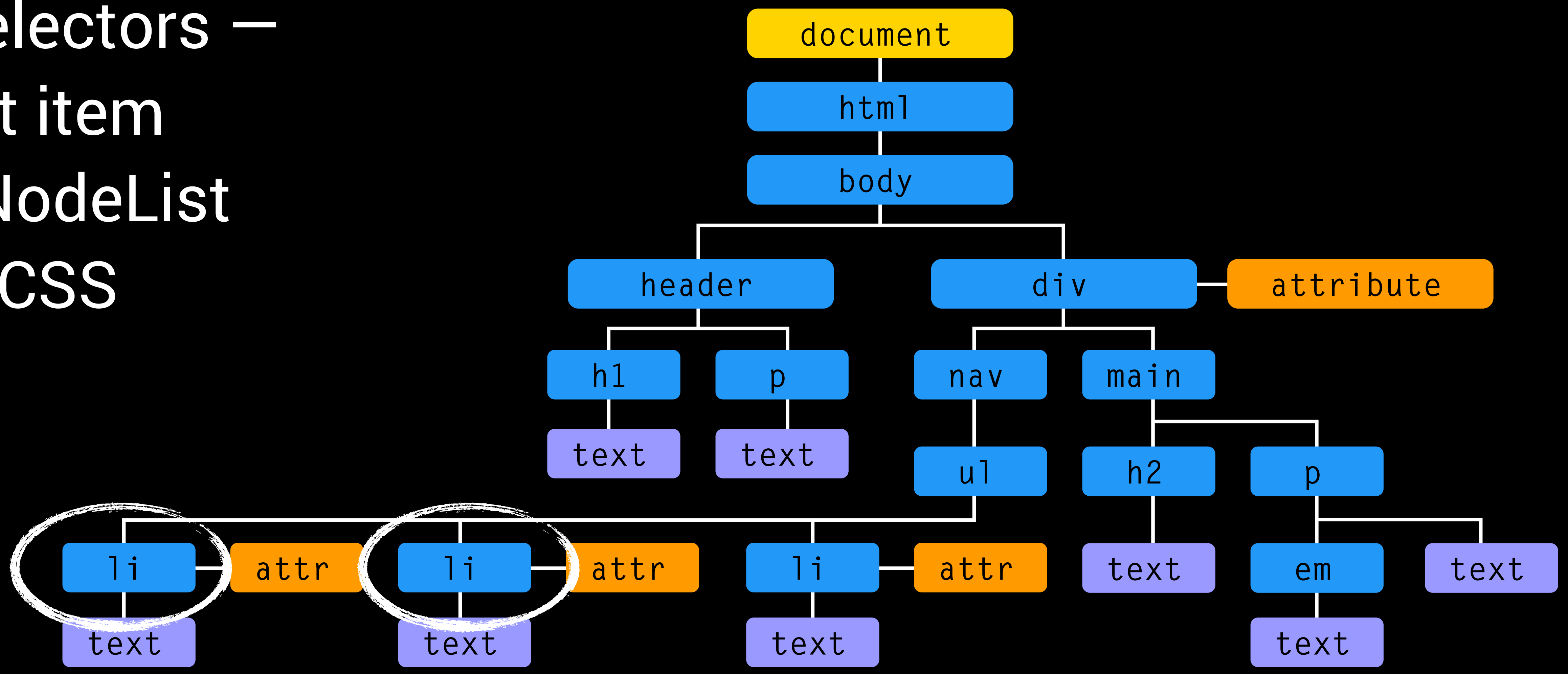


```
var menuItems = document.querySelector('li.one');
```

This would just return the first <li> element with a class of "one".

WORKING WITH THE DOM TREE

You can also select elements using CSS selectors — either a single item (though always the first item that matches the CSS-style selector) or a NodeList of all the matches. Both methods take the CSS selector as their only parameter.



```
var menuItems = document.querySelector('li.one');
```

This would just return the first <li> element with a class of "one".

```
var menuItems = document.querySelectorAll('li.one');
```

This would return both <li> elements with a class of "one" as a NodeList.

EXERCISE

Today, we will experiment with the DOM tree, using it to select and change some CSS style on a basic page. First, we will use our template to create some basic HTML code. We will add a button at the bottom to produce a change in color when it is clicked.

```
<button class="myButton" onclick="colorChange()">Change!</button>
```

Add a link to a new CSS style in the head, and add some style to make the button look nifty.

```
<link rel="stylesheet" href="css/color.css">
```

Finally, add a <script> tag at the bottom of the document, just above the closing body tag.

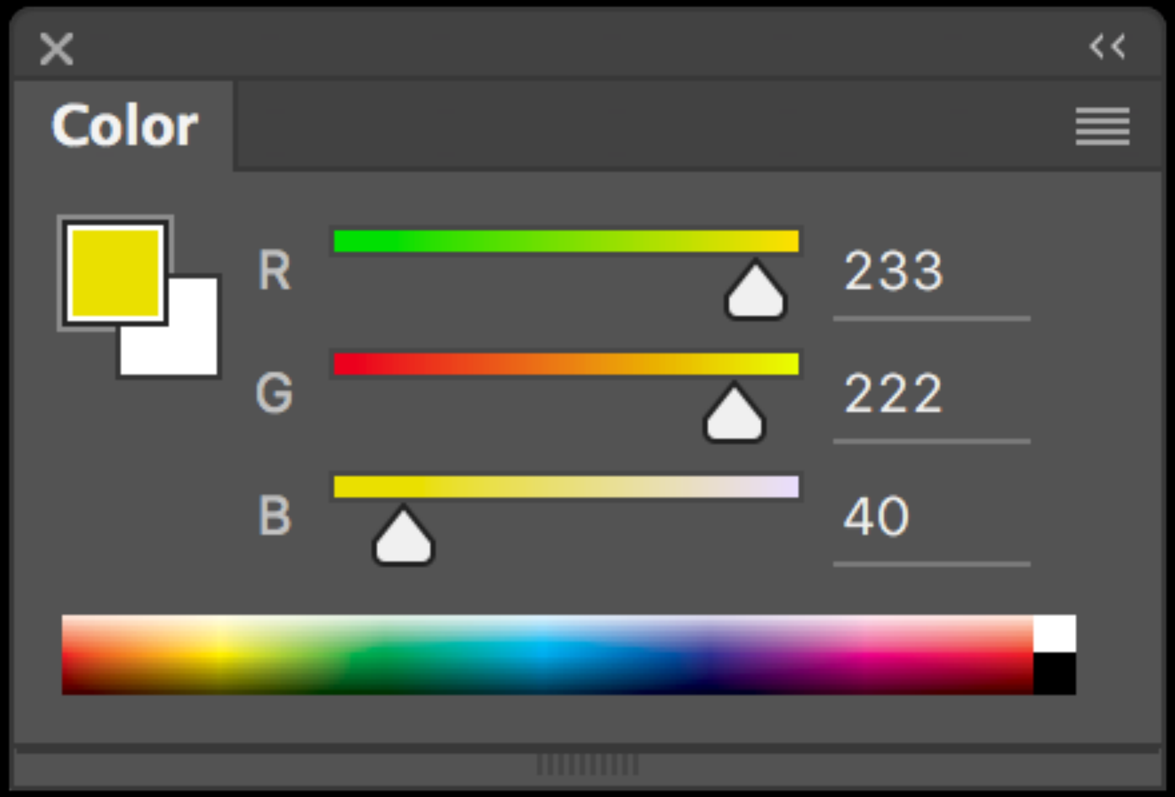
```
<script src="js/color.js"></script>
```

EXERCISE

We will go through each of the DOM queries we have talked about today, and add code that will change the color of the selected object to a random color.

First, soon after the function opens, you will add this code to generate a random RGB color. This will use the Math object, which has properties and methods for various mathematical functions.

```
(Math.floor(Math.random() * 256))
```



This small chunk of code will generate an integer between 0 and 255. The numbers 0 through 255 refer to RGB color positions.

EXERCISE

To generate a random RGB value, you need to stitch together several strings. Remember, RGB values are written like this:

```
rgb(233,222,40)
```

To create a random assortment of the three numbers between 0 and 255, you would add this code:

```
var hue = 'rgb(' + (Math.floor(Math.random() * 256))
+ ',' + (Math.floor(Math.random() * 256)) + ',' +
(Math.floor(Math.random() * 256)) + ')';
```

This random number will be stored in a variable called “hue.”

EXERCISE

Use various DOM queries to change the following:

- The color of the band with your name in it — the h2 in the header *(use getElementById)*.
- The color of the anchor tags *(use getElementsByTagName).*
- The color of the button itself, and the word inside *(use getElementsByClassName, and create a second random RGB color).*
- The color of the "460" in the header *(use a CSS selector).*