

# Introduction to the Document Object Model

---

# You're About to learn

- What is the DOM?
- Why should we care?
- DOM Manipulation
  - Searching the DOM
  - How to traverse the DOM
  - How to change the DOM

**The Document Object Model is what  
allows web pages to render, respond to  
user events and change**

# HTML vs DOM

```
<body>

  <h1>Hello</h1>

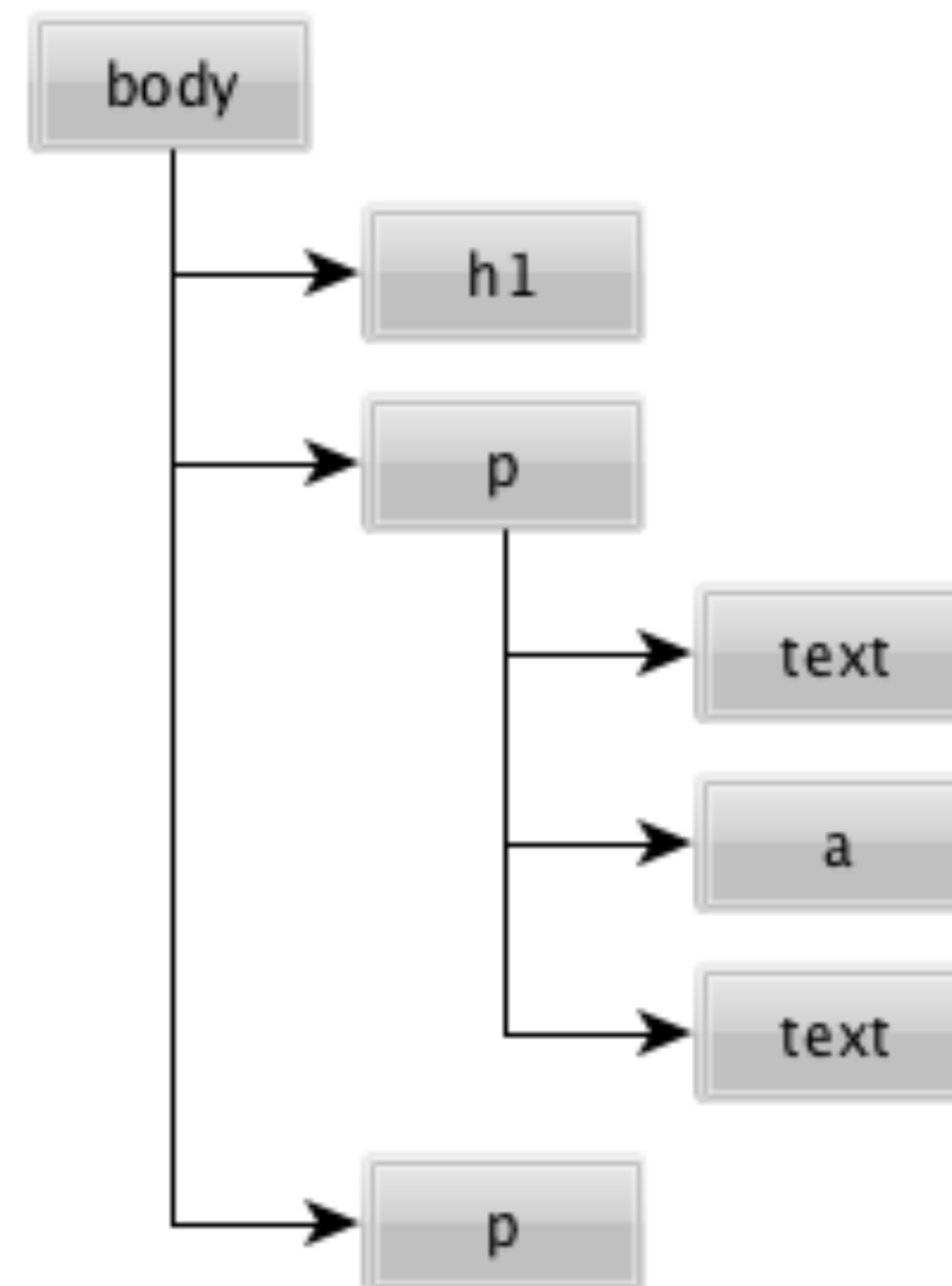
  <p>

    Check out my

    <a href="/page">Page!</a>

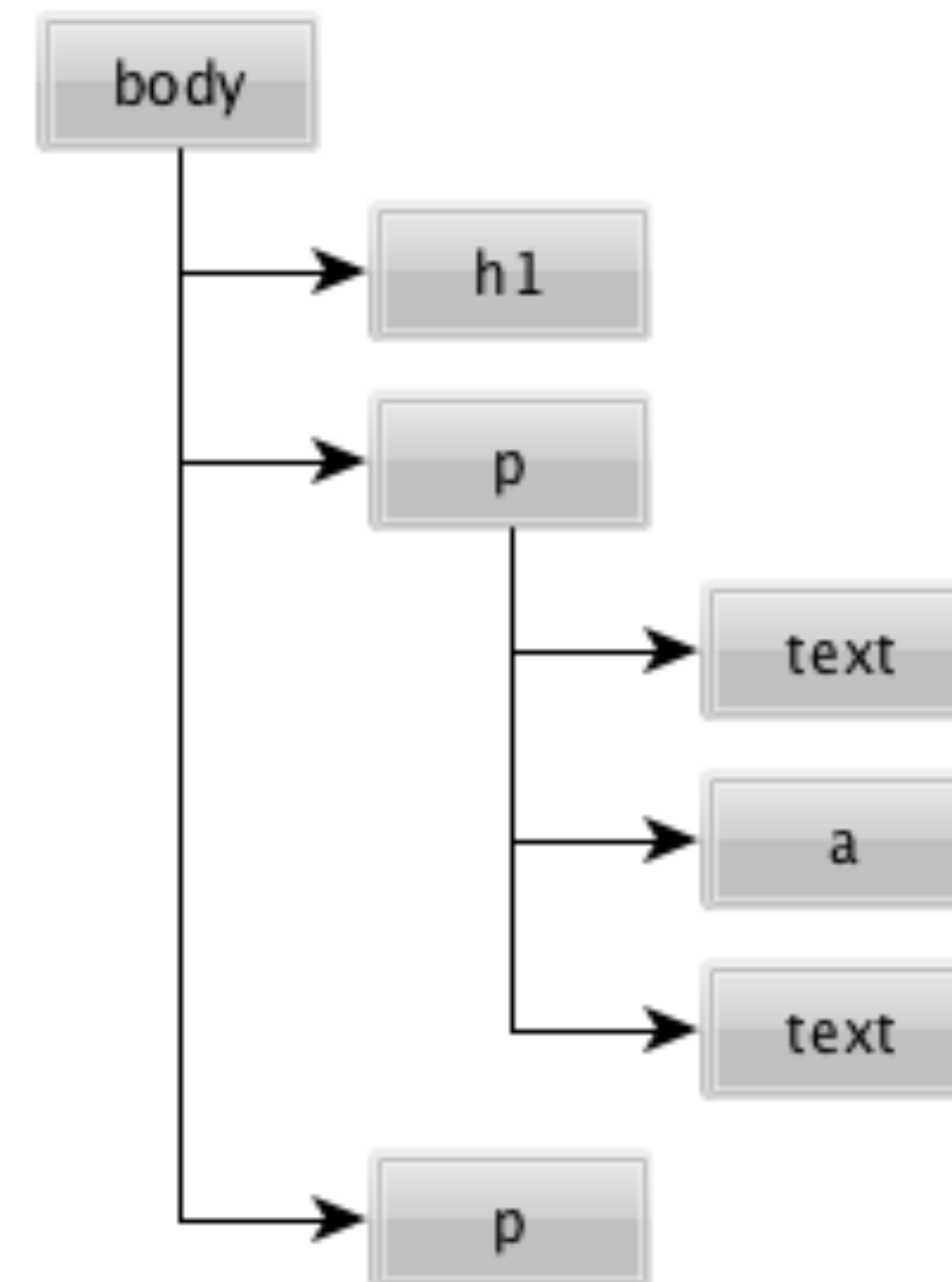
    It's the best page out there
  </p>

  <p>Come back soon!</p>
</body>
```

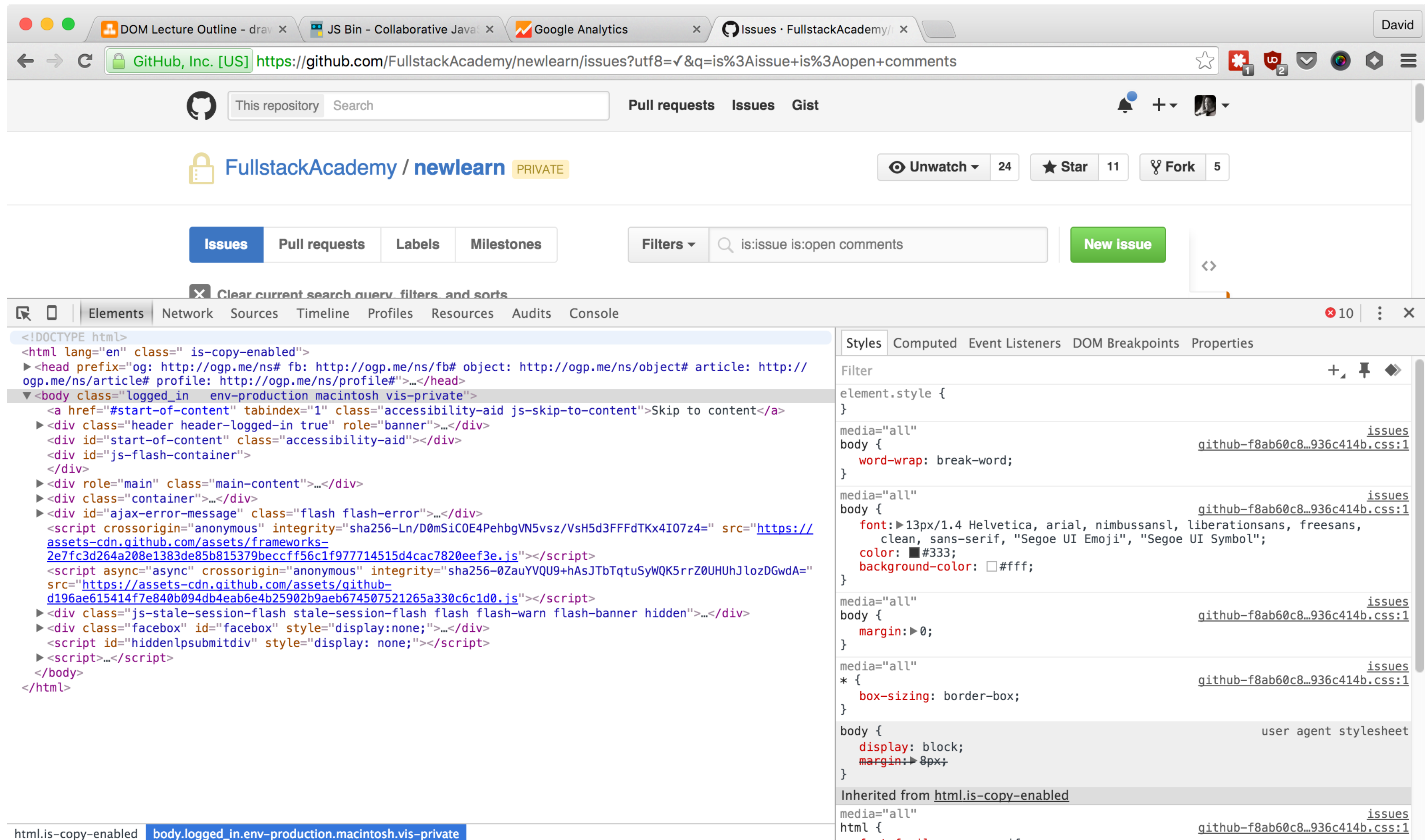


# The DOM is a Tree

- Trees are a data structure from computer science
- The main idea here: There is a Node that branches into other Nodes (its children Nodes)
  - Each Node can have 0 to many children Nodes
  - Nodes can have 0 or 1 parent
  - Nodes can have 0 to many Sibling Nodes



# Developer Tools



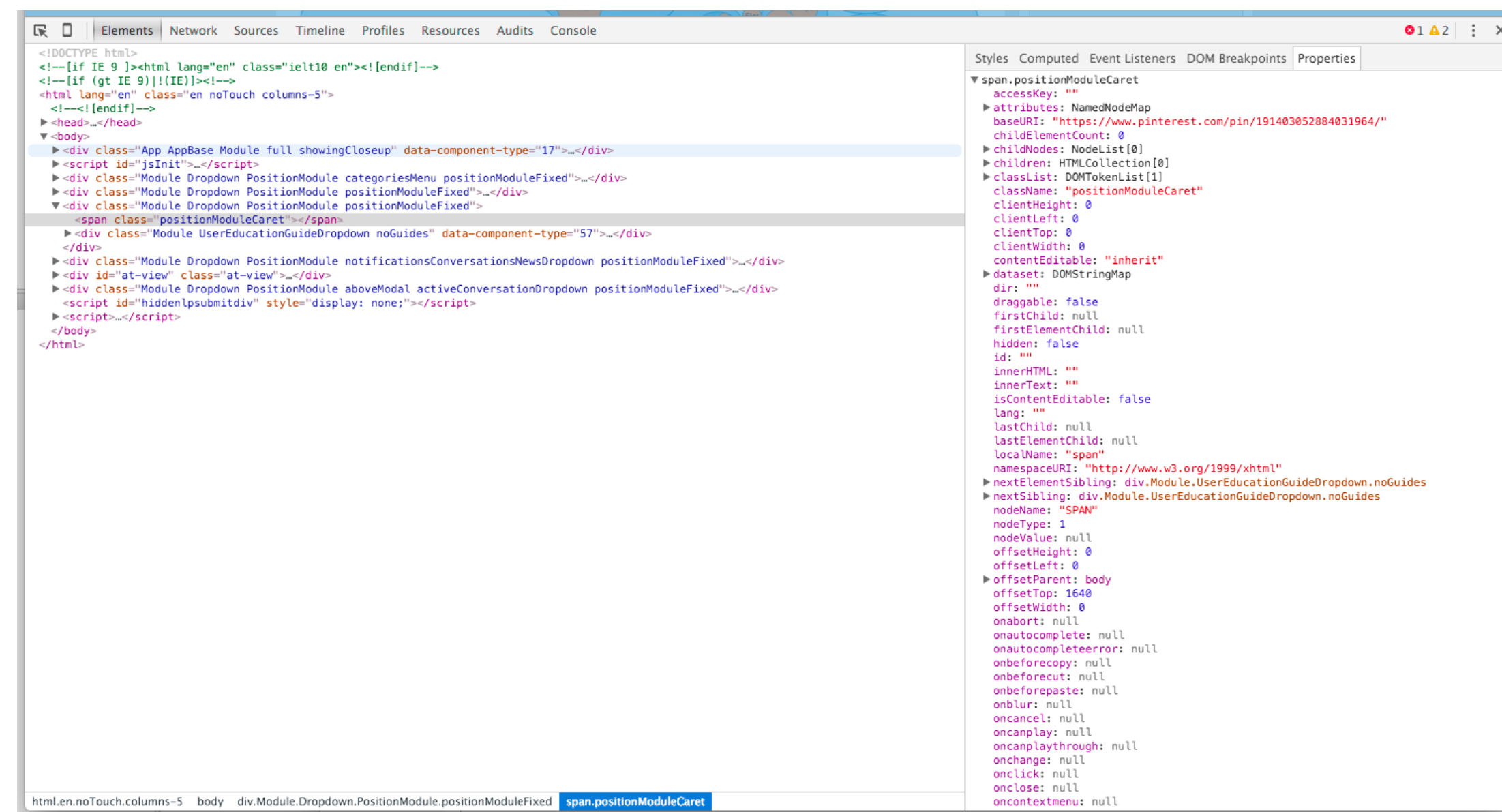
**The DOM makes it possible to use  
JavaScript to manipulate the document  
content and structure**



# Nodes have lots of Attributes

- Nodes are JavaScript Objects
- Nodes have Attributes that are JavaScript properties
- Attributes define how the Node looks and responds to User activity

ONE NODE



Hundreds of properties!



# The *document* Object

- Global reference to the DOM entry point
- Provides methods for:
  - Navigating the DOM
  - Manipulating the DOM
- The *document* object is the important connection between the DOM and JavaScript code

# Searching the DOM

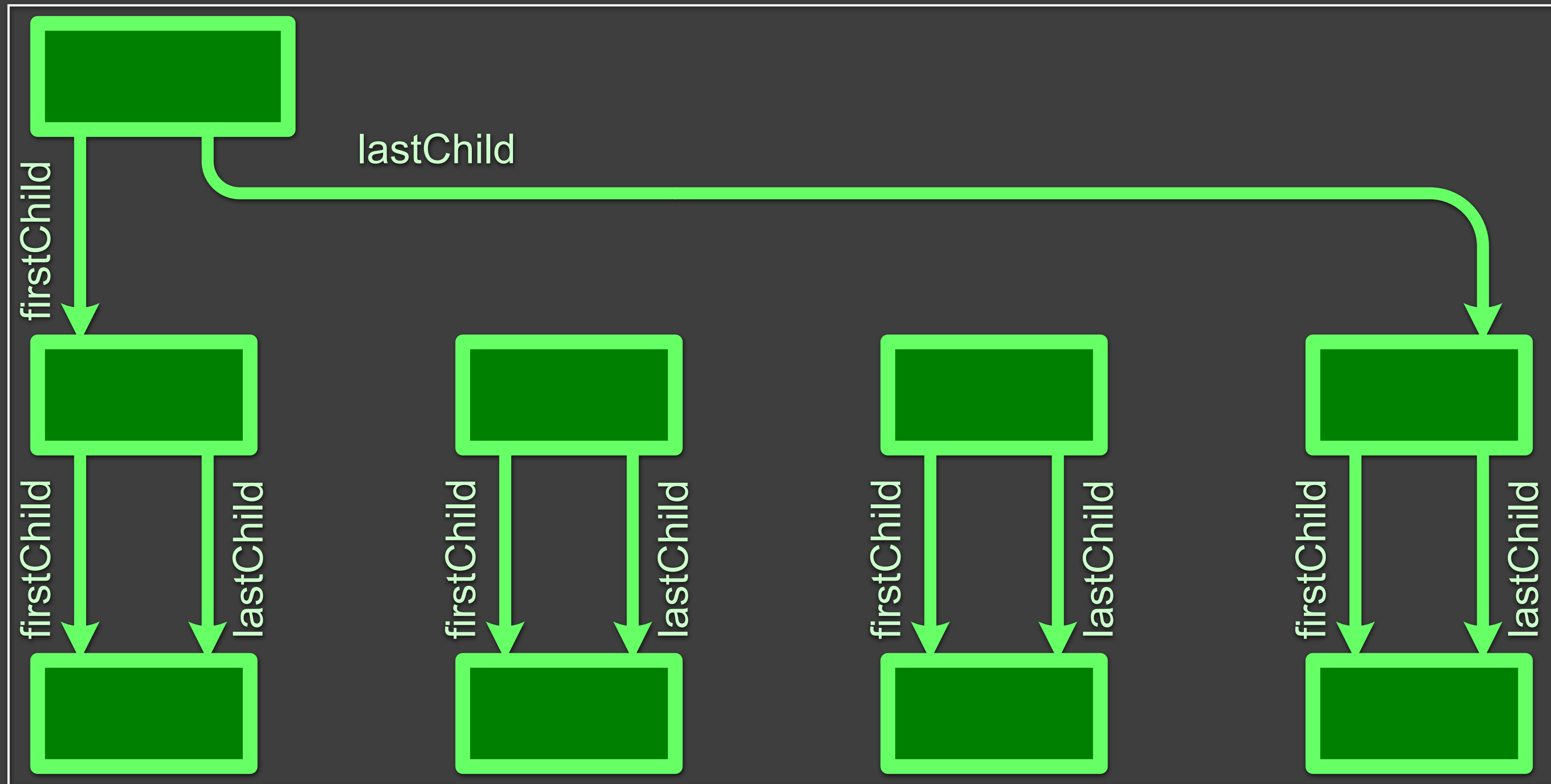
- ◉ **getElementById** (find nodes with a certain ID attribute)
  - `document.getElementById("will");`
- ◉ **getElementsByClassName** (find nodes with a certain CLASS ATTRIBUTE)
  - `document.getElementsByClassName("will");`
- ◉ **getElementsByTagName** (find nodes with a certain HTML tag)
  - `document.getElementsByTagName("div");`
- ◉ **querySelector, querySelectorAll** (search using CSS selectors)
  - `document.querySelector("#will .will:first-child");`

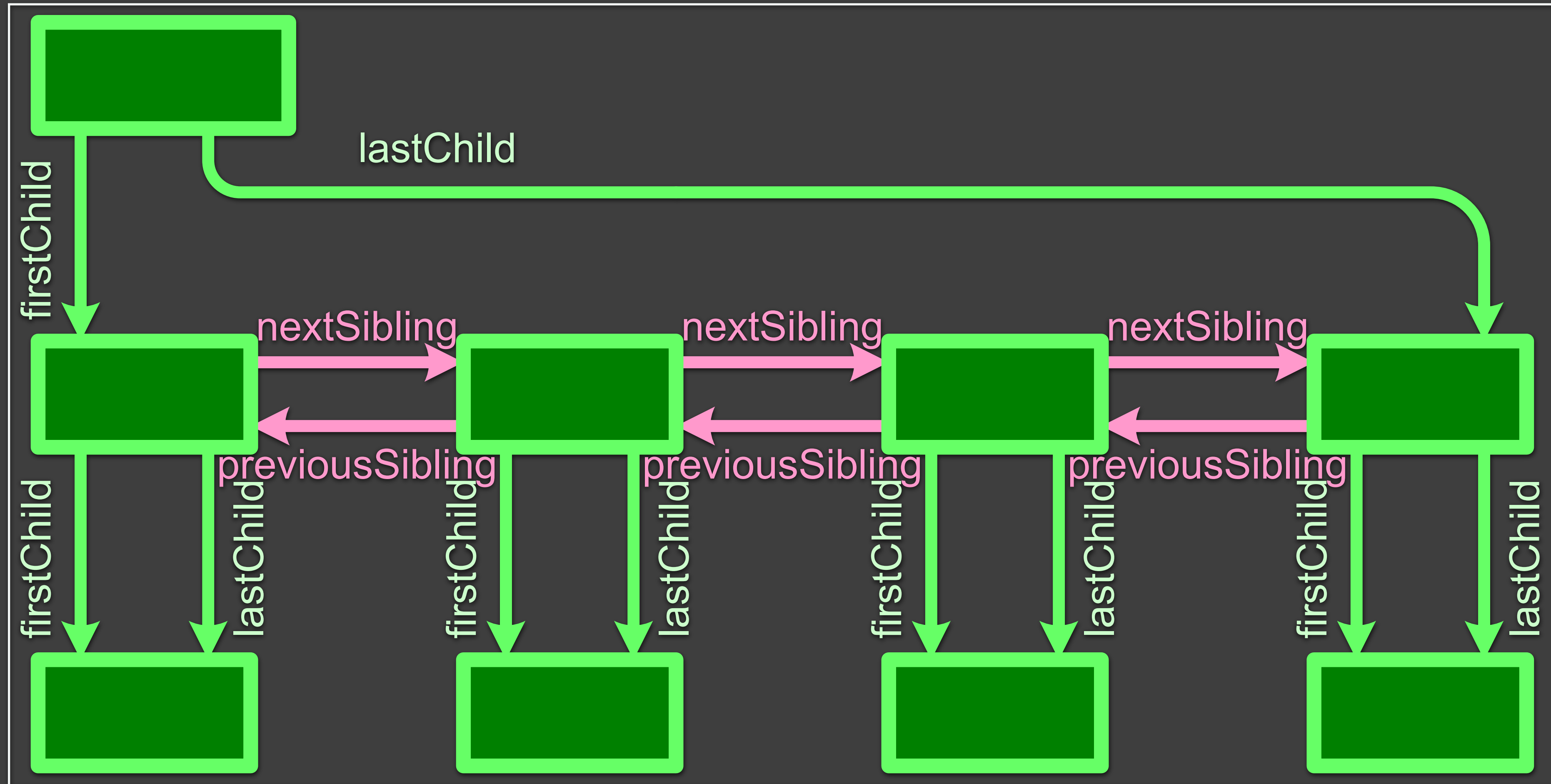
# Array-Like Objects? Bleh!

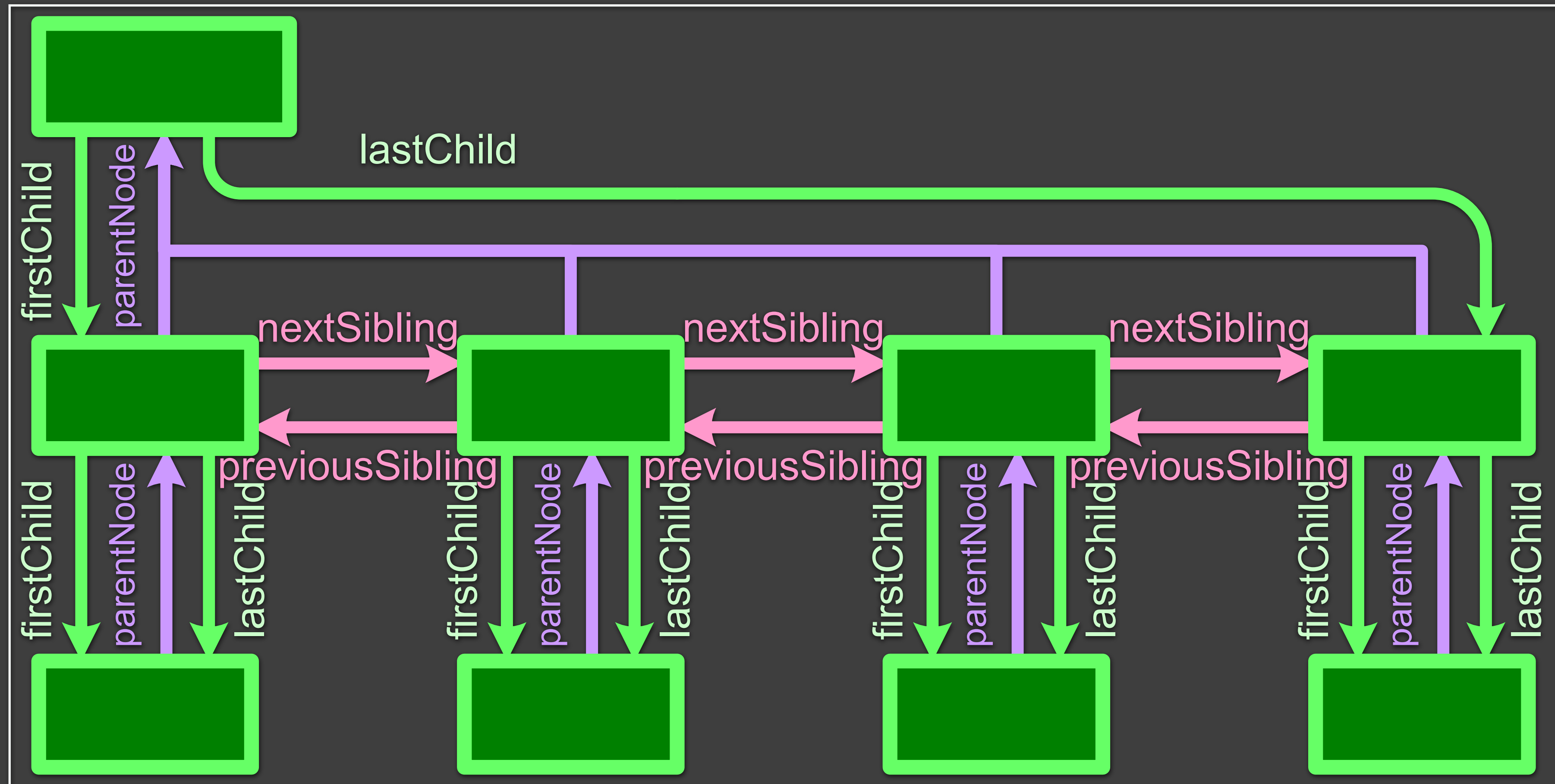
- ◉ `const realArr = [].prototype.slice.call(arrayLike)`
- ◉ `const realArr = Array.from(arrayLike)`
- ◉ `const realArr = [...arrayLike]`

# Traversing the DOM

- ◎ **Tree Structures are easy to navigate:**
  - At any point in the DOM you are at a Node
  - No matter where you go, you're still at a Node
    - Child
    - Parent
    - Sibling
  - All Nodes share similar DOM navigation methods









# Traversing the DOM

- **Access children**

- `element.children`, `element.lastChild`, `element.firstChild`

- **Access siblings**

- `element.nextElementSibling`, `element.previousElementSibling`

- **Access parent**

- `element.parentElement`

# Changing style attributes

```
element.style.backgroundColor = "blue";
```

## ● CSS

- background-color →
- border-radius →
- font-size →
- list-style-type →
- word-spacing →
- z-index →

## ● JavaScript

- backgroundColor
- borderRadius
- fontSize
- listStyleType
- wordSpacing
- zIndex

# Changing CSS Classes

- *className* attribute is a string of all of a Node's classes
- *classList* is HTML5 way to modify which classes are on a Node

```
document.getElementById("MyElement").classList.add('class');
```

```
document.getElementById("MyElement").classList.remove('class');
```

```
if ( document.getElementById("MyElement").classList.contains('class') )
```

```
document.getElementById("MyElement").classList.toggle('class');
```

# Creating Elements

- **Create an element**
  - `document.createElement(tagName)`
- **Duplicate an existing node**
  - `node.cloneNode()`
- **Nodes are just free floating, not connected to the document itself, until you *link* them to the DOM.**

# Adding elements to the DOM

- **Insert newNode at end of current node**
  - `node.appendChild(newNode);`
- **Insert newNode at end of current node**
  - `node.prependChild(newNode);`
- **Insert newNode before a certain childNode**
  - `node.insertBefore(newNode, sibling);`

# Removing Elements

- ◉ Removes the `oldNode` child.
  - ◉ `node.removeChild(oldNode);`
- ◉ Quick hack:
  - `oldNode.parentNode.removeChild(oldNode);`