



# **COMP 4021**

# **Internet Computing**

## **DOM**

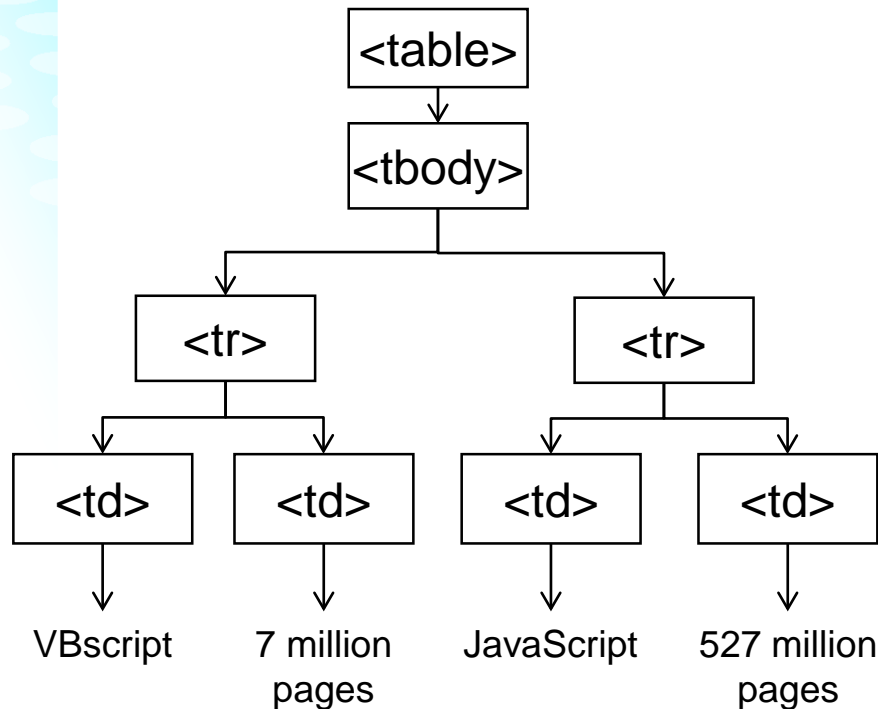
Dr. Kenneth LEUNG

Slides created by Dr. David ROSSITER

# This Presentation

- This presentation considers the following:
  - Simple DOM example
  - DOM representation
  - Using relations to traverse the tree – examples
  - Referring to nodes - three methods

# Simple DOM Example



```
<table>
  <tbody>
    <tr>
      <td>VBscript</td>
      <td>7 million pages</td>
    </tr>
    <tr>
      <td>JavaScript</td>
      <td>527 million
pages</td>
    </tr>
  </tbody>
</table>
```

# The DOM Standard

- Scripting languages (not only JavaScript) can access any part of the DOM including relationships (parent/sibling, etc.)
- You can actively alter, create and destroy *any* part of the DOM structure, at *any* time
- The same code will work for all browsers, e.g., IE, Firefox and Opera without any changes
- The same techniques can also be used in lots of other languages i.e. Java, C++, PHP, etc.

# A Simple (Incomplete) DOM Example

```
<body id="bodyNode">
```

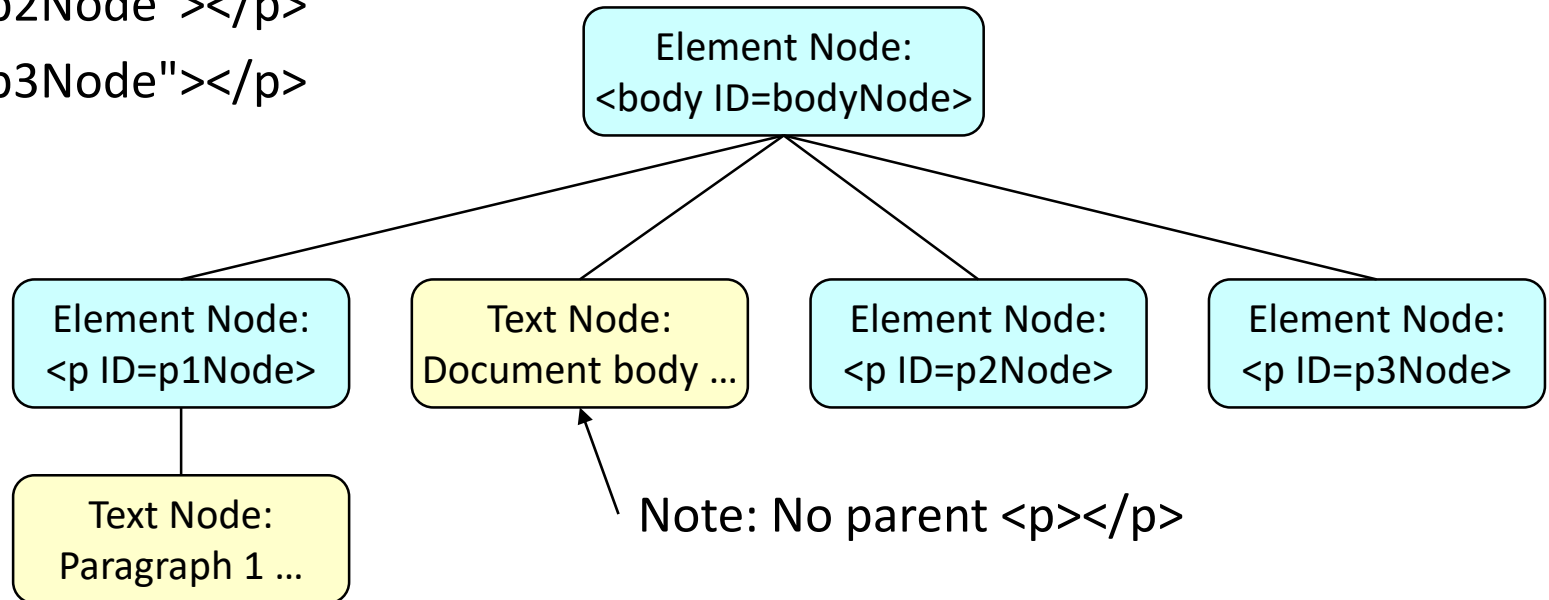
```
<p id = "p1Node">Paragraph 1 ...</p>
```

```
Document body ...
```

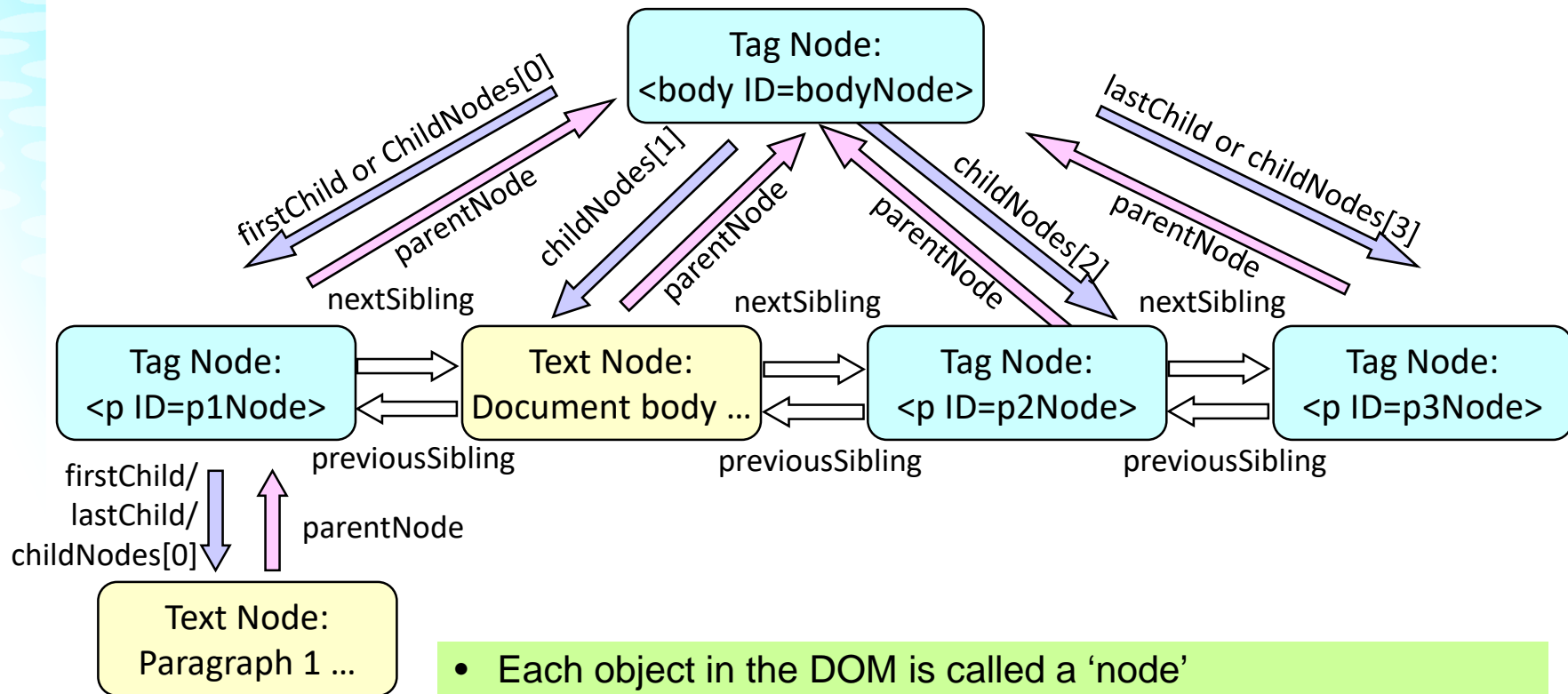
```
<p id = "p2Node"></p>
```

```
<p id = "p3Node"></p>
```

```
</body>
```



# Detailed DOM Example



- Each object in the DOM is called a 'node'
- Both nodes and relationships between nodes are shown
- Any node can be given a name (the ID attribute) for reference by other nodes and scripts

# Using Node Relations

- Scripts can access all of these relations between nodes:
  - parentNode
  - childNodes[], firstChild, lastChild
  - previousSibling, nextSibling
  - and more...
- There is more than one way to write some things  
i.e. `childNodes[0]` is the same as `firstChild`
- `childNodes.length` returns the number of child nodes
  - So `childNodes[childNodes.length-1]` equals `lastChild`

# Using Relations to Traverse the Tree - 1

- The code below starts with any node 'node' in the DOM, and then traverses up the branches, each time adding the name of the parent to a string, until the root is reached
- The result is to create a string which contains the path from the root to the starting 'node',
  - e.g., **#document->HTML->BODY->UL->LI->A**

```
function click() {  
    var node=this; // this = current object  
  
    tree=node.nodeName;  
    while (node.parentNode) {  
        node = node.parentNode;  
        tree = node.nodeName + " -> " +  
tree;    }  
    alert(tree);    }
```



## Using Relations to Traverse the Tree - 2

- This example is more advanced, using recursion
- It shows how code can be written to access every single element in the DOM (i.e., everything in the web page)
- It goes to every node and instructs that when an *onmouseover* event occurs to that node, the function *do\_someth* will be executed
  - The exact purpose of the *do\_someth* is not important for this demo; it could be as simple as changing the colour of the node to red

# Using Relations to Traverse the Tree - 2

```
function processChildren(node) {  
    var currentNode = node.firstChild; // start with the first child  
    do {  
        currentNode.onmouseover = do_someth; // do something with node  
        if (currentNode.hasChildNodes) { // if node has children  
            processChildren(currentNode); // process them (recursive)  
        }  
  
        currentNode = currentNode.nextSibling; // move to the next sibling  
  
    } while (currentNode != node.lastChild // repeat until last child  
            && currentNode != null) // or until nothing more  
}
```

- Traversal of the entire DOM can be done in different ways
- Upon reaching a node, attach an event handler **do\_someth** (function not shown, e.g., change the background colour of the node)

# How to Locate One Particular Thing?

- Method 1: Use the exact **DOM path**
  - May be hard to work out the exact position
  - Easy to make mistakes
  - Load into another browser – DOM may be a bit different, not work!
- Method 2: Use **getElementsByTagName()**
  - Require you to know the exact tag name (I.e. is it h2 or h3?)
  - Also, there might be several nodes of that type, so you have to know exactly which one it is (I.e. first one? second one?)
- Method 3: Use **getElementById()**
  - If you give the nodes unique names then this method is the easiest to refer to them

# Methods 1, 2, 3 - Examples

```
<html>
<head>
<script language="JavaScript">
  function change_col_script1() {
    document.childNodes[0].childNodes[1].childNodes[0].style.color="red";    }
  function change_col_script2() {
    document.getElementsByTagName("h2")[0].style.color = "yellow";    }
  function change_col_script3() {
    document.getElementById("cute_text").style.color = "blue";    }
</head>
</script>
<body>
  <h2 id="cute_text">Click below to change the colour of this text</h2>
  <form>
    <input onclick="change_col_script1()" type="button" value="Change using method 1">
    <input onclick="change_col_script2()" type="button" value="Change using method 2">
    <input onclick="change_col_script3()" type="button" value="Change using method 3">
  </form>
</body>
</html>
```

# Why Absolute Addressing does not Work?

```
<html>
<head> <script language="JavaScript">
  function change_col_script1() {
    document.childNodes[0].childNodes[1].childNodes[0].style.color="red";  }
  ... ..
</script>
</head>
<body>
<h2 id="cute_text">
Click below to change the colour
</h2>
... ..
```

