University of Glasgow

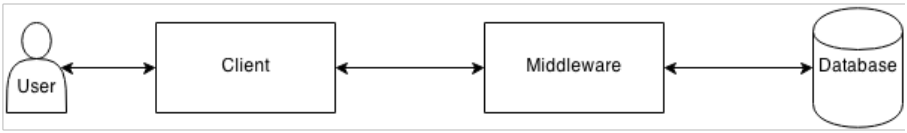
Client Side Environment

Web Application Development 2

Client Side Environment

- Client Tier
- Document Object Model (DOM)
- Event Handling

3-Tier Context

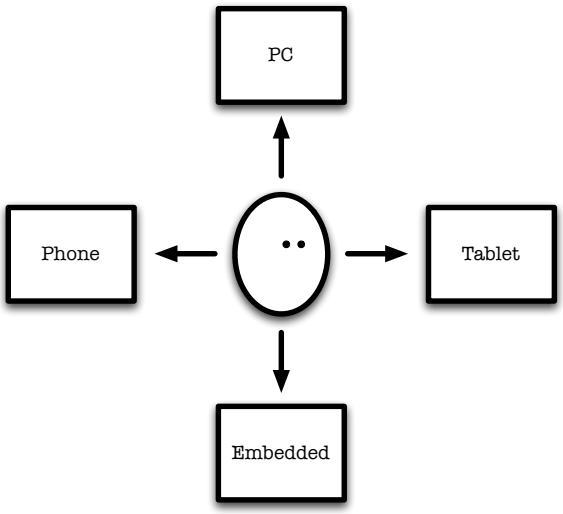


Chrome / Edge / Firefox / Safari

Web server: Apache
Application Server: Django

Database Server: MySQL

Clients



User: Plant

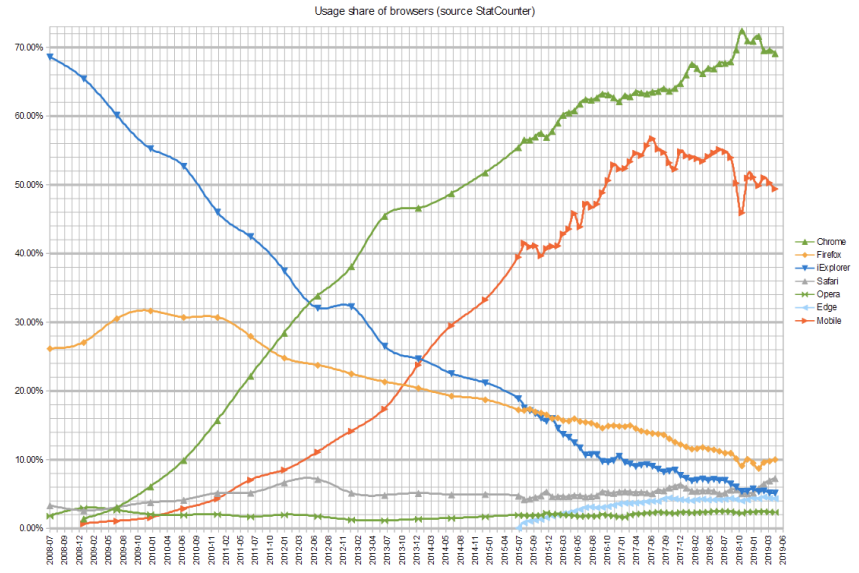
You didn't water me enough. 146 days ago
URGENT! Water me! 148 days ago

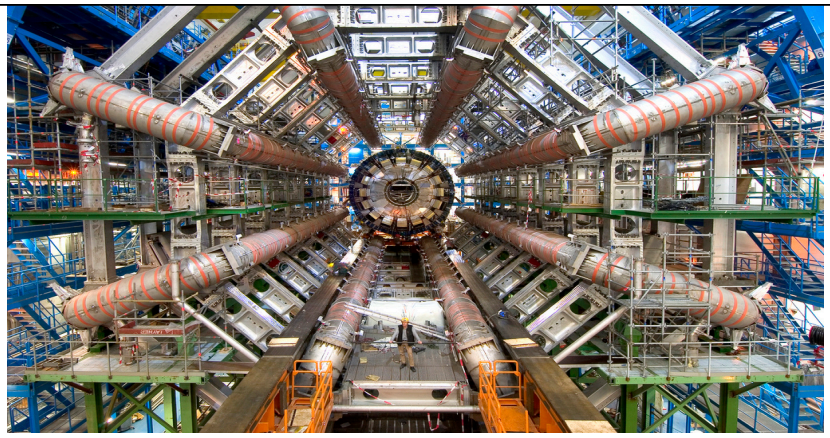
Water me please. 150 days ago

You didn't water me enough. 150 days ago
You over watered me. 153 days ago

http://www.botanicalls.com/about/

Browser Wars

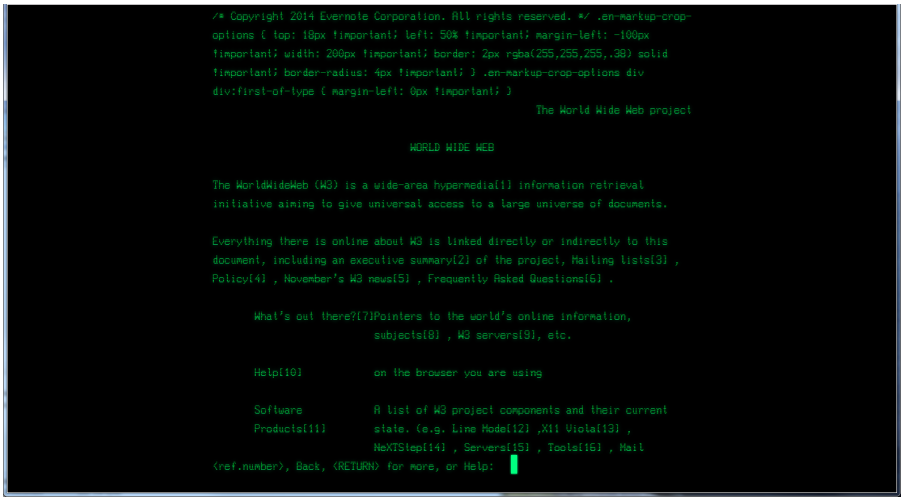




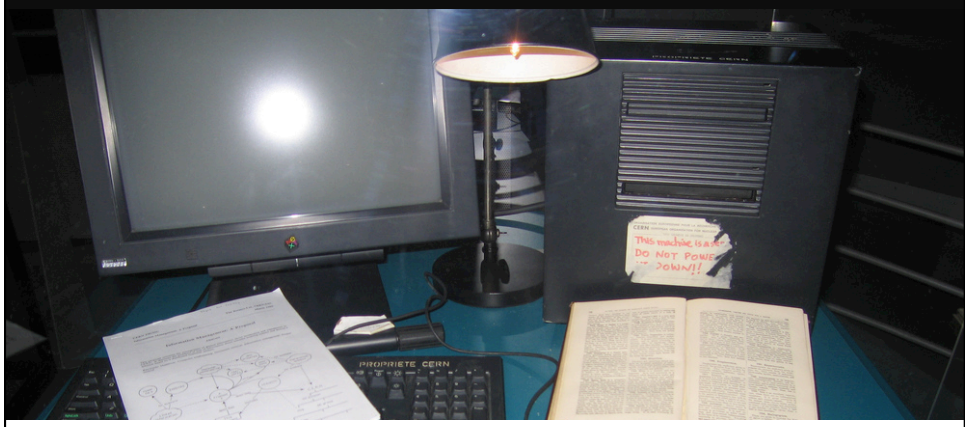
IN THE BEGINNING

The Creator
(of the web)

The first web page



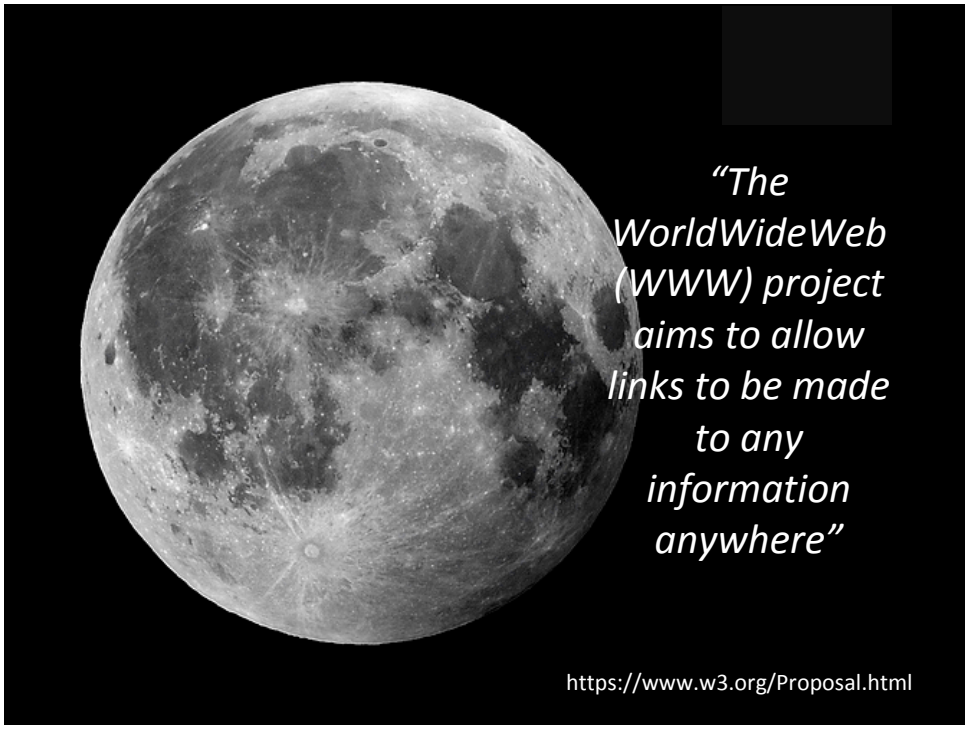
The Whole Stack

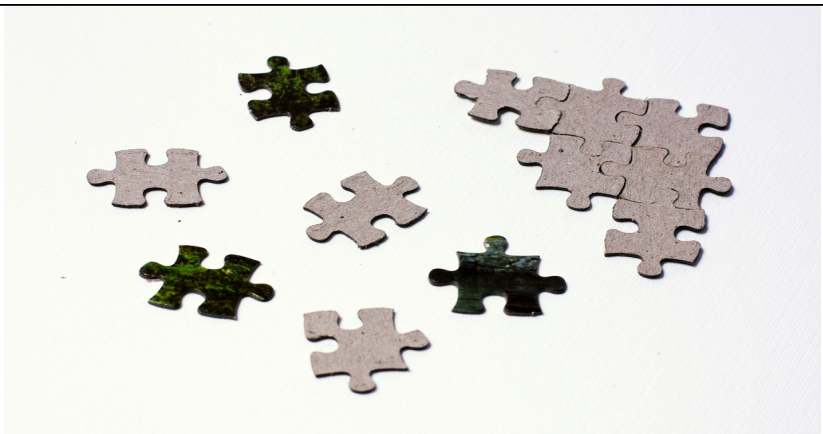


HTTP
HTTP server
HTML
Web browser / editor / Usenet / FTP
Only ran on NeXT

Mostly developed and in place by

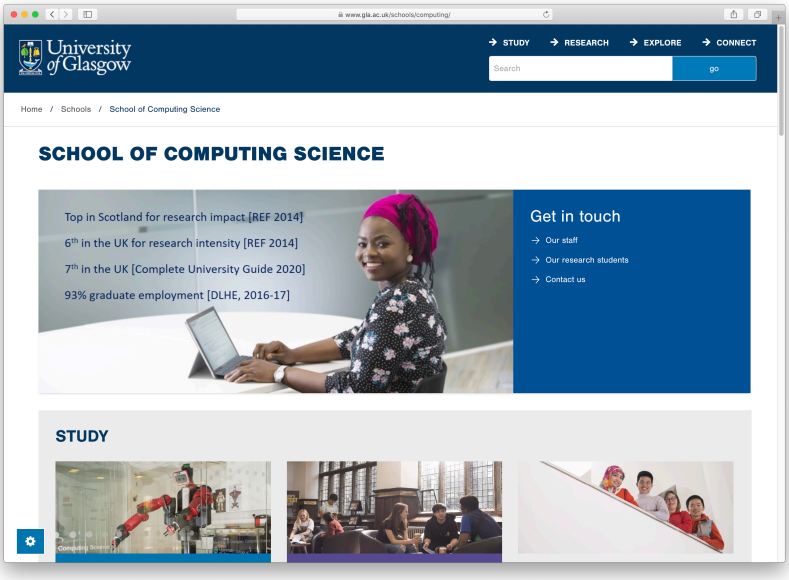
1990





BROWSER COMPONENTS

What is in a page?



HTML Source Code

```
<!doctype html>
<html class="no-js" lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>University of Glasgow - Schools - School of Computing Science</title>
    <meta http-equiv="X-UA-Compatible" content="IE=EDGE" />
    <link rel="canonical" href="https://www.gla.ac.uk/schools/computing/" />
    <!--standard meta tags-->
    <meta name="keywords" content="Computer Science, Russell Group, Glasgow, Scotland, Top UK education and research, C
    <meta name="description" content="Computer Science, Computing Science, University of Glasgow" />
    <!--twitter meta tags-->
    <meta name="twitter:card" content="summary_large_image" />
    <meta name="twitter:site" content="@UofGlasgow" />
    <meta name="twitter:creator" content="@UofGlasgow" />
    <!--opengraph meta tags-->



    <!--pagetype-->
        <meta name="pagetype" content="landing" />

    <!--dev flag-->
      <script>var development = true;</script>

<!-- navigation object : 3kHead -->
<!--icons-->
<meta name="mobile-web-app-capable" content="yes">
<meta name="theme-color" content="#003865">
<link rel="apple-touch-icon" sizes="128x128" href="/3t4/img/hd_hi.png">
<link rel="apple-touch-icon" sizes="64x64" href="/3t4/img/hd_small.png">
<script
  src="/3t4/js/libs/jquery/jquery-3.4.1.min.js"
></script>
<script src="/3t4/js/libs/jquery/jquery-migrate-1.4.1.min.js"></script>
<script src="/3t4/js/libs/jqueryui/jquery-ui.min.js"></script>
<script>
  document.domain = 'gla.ac.uk';
            var $function = {};
</script>
<script>
  var loaded = function(){document.body.style.visibility='visible';}
</script>

<noscript>
  <style>
    #leftNavlist nav ul li a,
    #leftNavlist nav ul li span {
      display: block;
    }

    #leftNav {
      width: 350px;
      box-shadow: 6px 0px 37px -9px rgba(0, 0, 0, 0.1);
      top: 128px;
    }

    #content-container {
      margin-top: 64px;
```

DOCUMENT OBJECT MODEL

Document Object Model

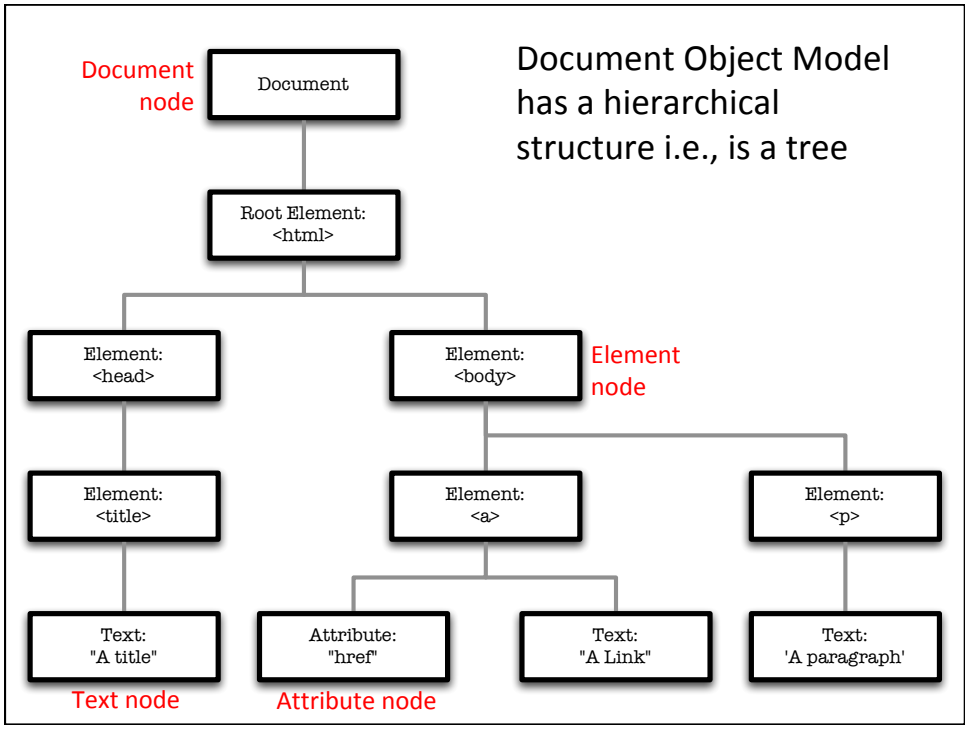
*"The Document Object Model is a platform- and language-neutral **interface** that will allow **programs** and scripts to dynamically **access** and **update** the **content**, **structure** and **style** of **documents**.*

The document can be further processed and the results of that processing can be incorporated back into the presented page."

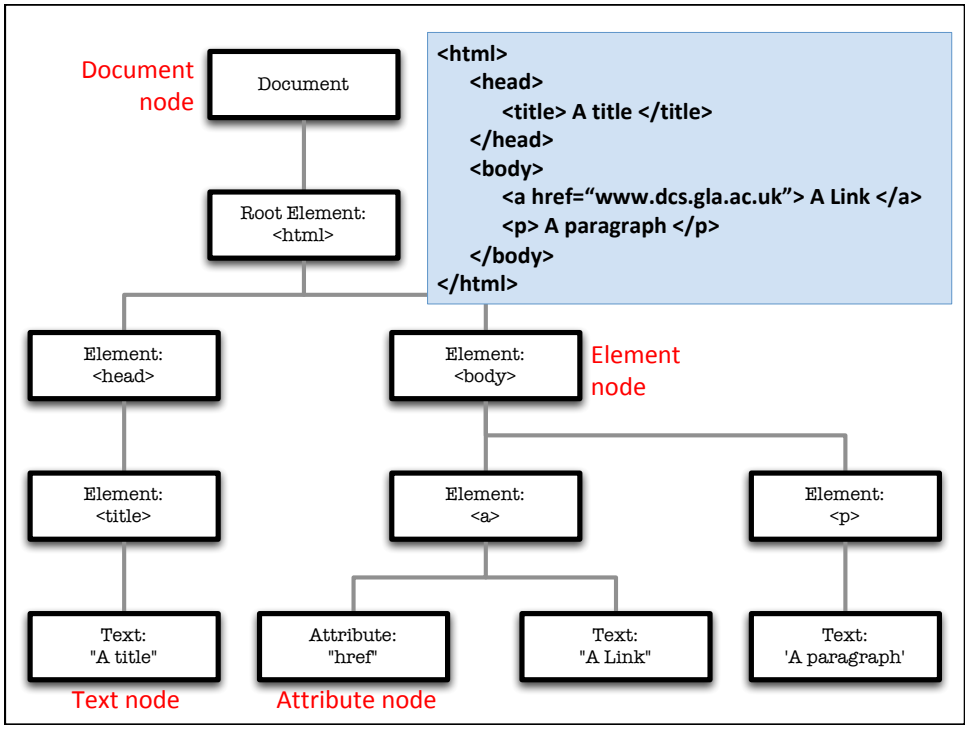
http://www.w3.org/DOM/#what

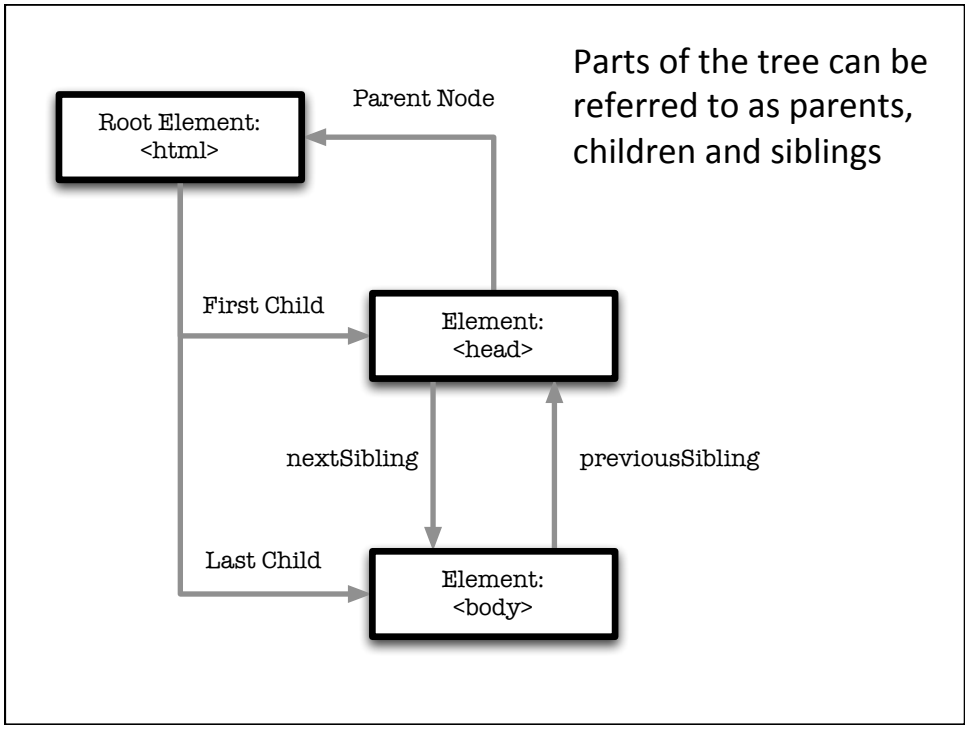
Nodes in the DOM

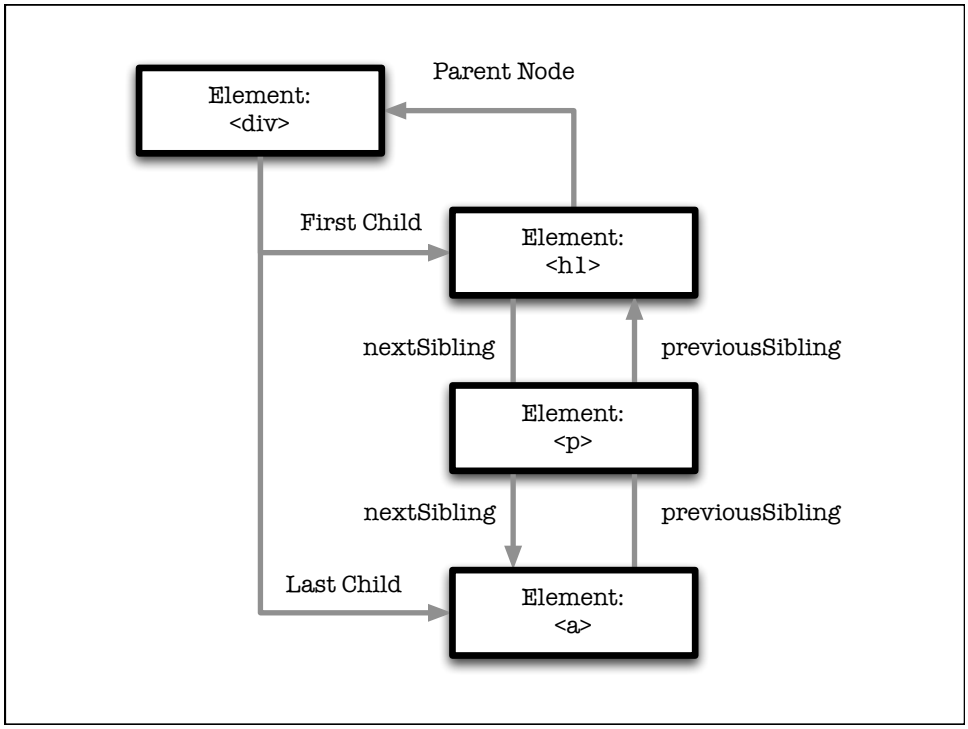
- In the HTML DOM (Document Object Model), everything is a **node**:
  - The document itself is a **document node**
  - All HTML elements are **element nodes**
  - All HTML attributes are **attribute nodes**
  - Text inside HTML elements are **text nodes**
  - Comments are **comment nodes**
- Element nodes can have **child nodes** of type element nodes, attribute nodes, text nodes, or comment nodes
- A **NodeList object** represents a list of nodes, like an HTML element's collection of child nodes



Document Object Model has a hierarchical structure i.e., is a tree



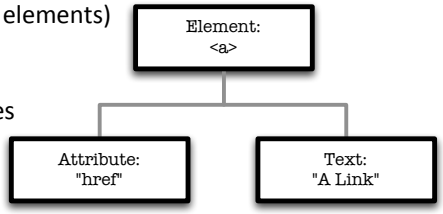

Parts of the tree can be referred to as parents, children and siblings
Root Element:
<html>
Parent Node
First Child
Element:
<head>
nextSibling
previousSibling
Last Child
Element:
<body>


Element:

Parent Node
First Child
Element:
<h1>
nextSibling
previousSibling
Element:
<p>
nextSibling
previousSibling
Last Child
Element:
<a>

Element Properties & Methods

Properties (values you can get or set, like changing the content of an HTML element):
someElement.**innerHTML** - the text value
someElement.**nodeName** - the name
someElement.**nodeValue** - the value (null for elements)
someElement.**parentNode** - the parent node
someElement.**childNodes** - the child nodes
someElement.**attributes** - the attributes nodes



Methods (actions you can do, like add or deleting an HTML element):
someElement.**getElementById(id)** - get the element with a specified id
someElement.**getElementsByTagName(name)** - get all elements with a specified tag name
someElement.**appendChild(node)** - insert a child node
someElement.**removeChild(node)** - remove a child node (… so update the page)

Note: these are JavaScript methods

```
<html>
  <body>
    <p id="intro">Web Application Development 2</p>
    <p>Introduction to DOM</p>
    <p>Example usage of some DOM functions</p>

    <a id="link" href="http://www.google.com">Google</a>

    <p id="move">This paragraph will be moved to the div block</p>

    <p>This comes just before the div block</p>

    <div id="divblock"><!--children 1 and 3 are empty text nodes-->
      <p>This paragraph will be removed from the div block</p>
    </div>

    <script type="text/javascript">
      num = document.getElementsByTagName("p").length;
      document.write("<p>There are "+num+" paragraph elements</p>");

      txt=document.getElementById("intro").innerHTML;
      document.write("<p>The text from the intro paragraph: "+txt+"</p>");

      num = document.getElementsByTagName("p").length;
      document.write("<p>There are now "+num+" paragraph elements</p>");

      document.getElementsByTagName("P")[2].innerHTML = "Hello World!";

      element = document.getElementById("intro");
      document.write("<p>Intro element: "+element+"</p>");
      document.write("<p>Intro element has name: "+element.nodeName+"</p>");
      document.write("<p>Intro element's value: "+element.nodeValue+"</p>”);
```

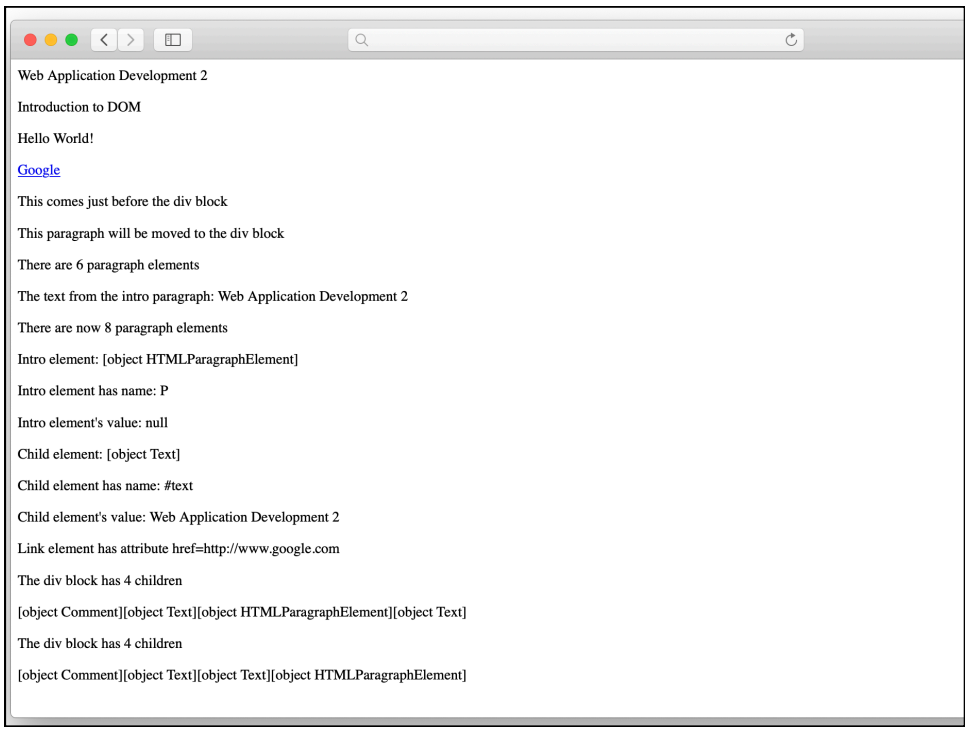
```
        element = element.firstChild;
        document.write("<p>Child element: "+element+"</p>");
        document.write("<p>Child element has name: "+element.nodeName+"</p>");
        document.write("<p>Child element's value: "+element.nodeValue+"</p>");

        element = document.getElementById("link");
        attr = element.attributes;
        val = attr.getNamedItem("href").value;
        document.write("<p>Link element has attribute href="+val+"</p>");

        element = document.getElementById("divblock");
        document.write("<p>The div block has "+element.childNodes.length+" children</p>");
        document.write(element.childNodes[0]);
        document.write(element.childNodes[1]);
        document.write(element.childNodes[2]);
        document.write(element.childNodes[3]);

        elementChild = element.childNodes[2];
        element.removeChild(elementChild);
        element2 = document.getElementById("move");
        element.appendChild(element2);

        document.write("<p>The div block has "+element.childNodes.length+" children</p>");
        document.write(element.childNodes[0]);
        document.write(element.childNodes[1]);
        document.write(element.childNodes[2]);
        document.write(element.childNodes[3]);
    </script>
</body> </html>
```



DOM Advantages

- **XML / Tree structure makes the DOM easy to traverse**
  - Elements can be accessed one or more times
- **XML / Structure of the Tree is modifiable**
  - Values/Elements/Structure can be added, changed and modified
- **It a standard of the W3C**
  - i.e., the unofficial/official law of the jungle

DOM disadvantages

- Resource intensive, consuming lots of memory
  - it needs to be fully loaded in main memory
- **Can be slow**
  - depends on the size and complexity of the tree
- **May not be the best choice for all devices/apps**
  - i.e., a graphics intensive application or game may well not be suited to this model
- A **better alternative** might be to use the **Canvas** directly
  - i.e., **OpenGL**

Working with the DOM

XHTML
CSS
JavaScript
jQuery

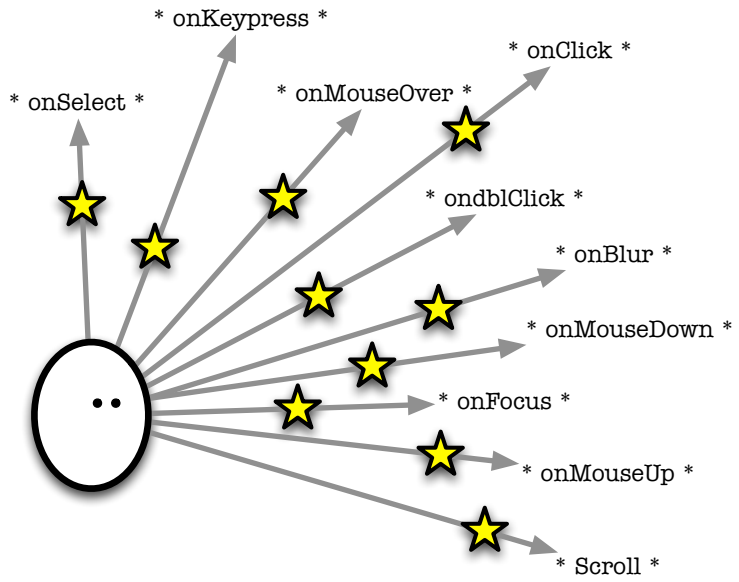
AJAX
XML
DOM / SAX Parsing

MORE INFO…

W3Schools DOM Tutorial:
http://www.w3schools.com/js/js_htmldom.asp
http://www.w3schools.com/jsref/dom_obj_document.asp

MONITORING USER INTERACTIONS

Handle this…



Event Object

- Each event has an associated object
- Part of the DOM
- The Event Object provides information about:
  - target element in which the event occurred
  - state of the keyboard keys
  - location of the mouse cursor
  - state of the mouse buttons

Event Handling

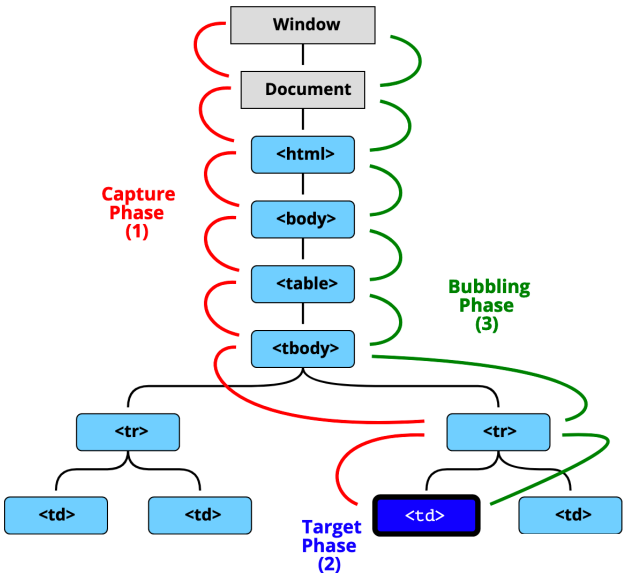
- Like any interactive application, events can be caught and used to execute functions
- For example, user input from forms can be validated on the client-side using JavaScript

```
<form name="login_form" action="login" onsubmit="return validateForm()" method="post">
  Username: <input type="text" name="username">
  Password: <input type="password" name="password">
  <input type="submit" value="Submit">
</form>
```

Event Flow

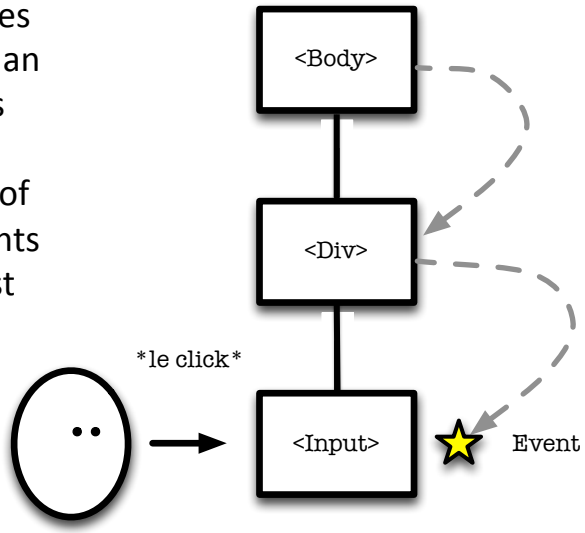
- Each event object has an 'Event Target', e.g., any node in the DOM tree from where an event originated
- There are two main types of event flow
  - event capture (*global handling*)
  - event bubbling (*local handling*)
- Eventflow follows a "RoundTrip" pattern/model

Event Flow



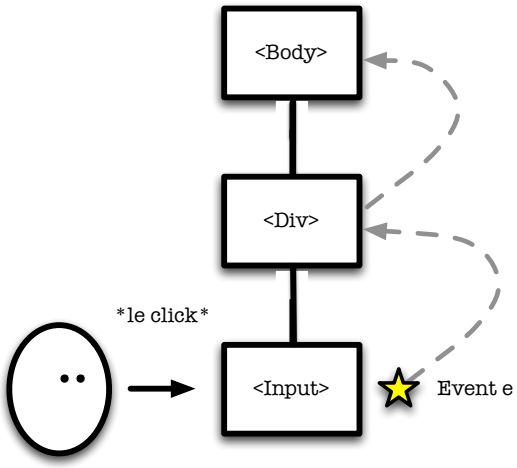
Event Capture

- The event propagates downwards though an element's ancestors
- Any event listeners of the ancestor elements will be executed first



Event Bubbling

- The event propagates upwards though an element's ancestors
- Any event listeners of the element will be executed first
- Ancestors can then potentially handle the event

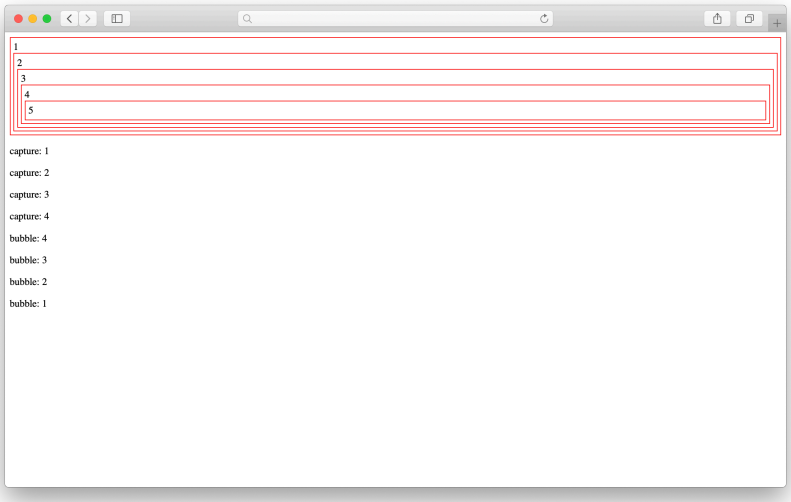


Example

```
<div onclick="alert('Handler 1')">
  <span onclick="alert('Handler 2')">
    <button onclick="alert('Handler 3')">Click here</button>
  </span>
</div>
```

- Event handling order under event capture:
  handler1() - > handler2() -> handler3()
- Event handling order under event bubbling:
  handler3() - > handler2() -> handler1()
- Both types of event generated, but bubbling caught by handlers by default & is more often used in practice

EventHandling.html



Example: Clicked on “4”

EventHandling.html

```
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="EventHandling.css"/>
    </head>
    <body>
        <div>1
           <div>2
              <div>3
                 <div>4
                    <div>5</div>
                 </div>
              </div>
           </div>
         </div>
         <p id="log"></p>
         <script type="text/javascript" src="EventHandling.js"></script>
     </body>
</html>
```

EventHandling.js

```
var node = document.getElementById("log");
var divs = document.getElementsByTagName("div");

function append(line) {
    var text = node.innerHTML;
    node.innerHTML = text + "<p>" + line + "</p>";
}

function capture() {
    append("capture: " + this.firstChild.nodeValue);
}

function bubble() {
    append("bubble: " + this.firstChild.nodeValue);
}

for (var i = 0; i < divs.length; i++) {
    // "click" is event type, capture/bubble are funcs from above,
    //  bool is useCapture mode (don't have to include - defaults to false)
    divs[i].addEventListener("click", capture, true);
    divs[i].addEventListener("click", bubble, false);
}
```