Chapter 9
Traditional JavaScript Object Models

Adapted from

JavaScript: The Complete Reference 2nd and 3rd Editions

by

Thomas Powell & Fritz Schneider

Two Object Models?

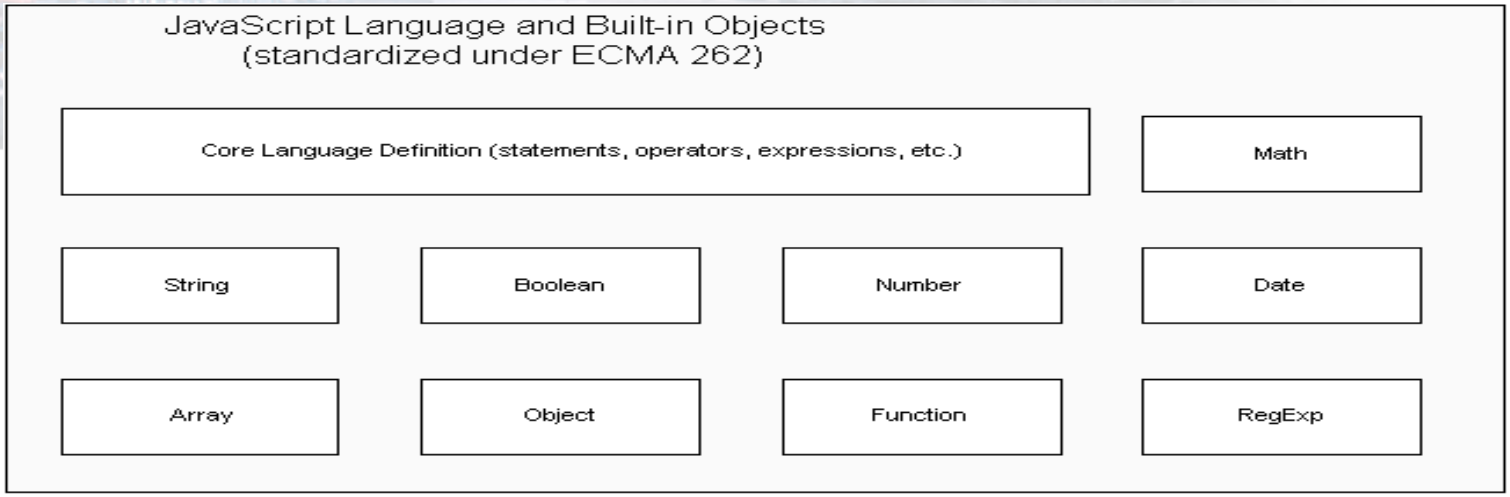
- An object model defines the interface to the various aspects of the browser and document that can be manipulated by JavaScript.
- In JavaScript, two primary object models are employed
  1. a browser object model (BOM)
     - The BOM provides access to the various characteristics of a browser such as the browser window itself, the screen characteristics, the browser history and so on.
  2. document object model (DOM).
     - The DOM on the other hand provides access to the contents of the browser window, namely the document including the various HTML elements ranging from anchors to images as well as any text that may be enclosed by such elements.

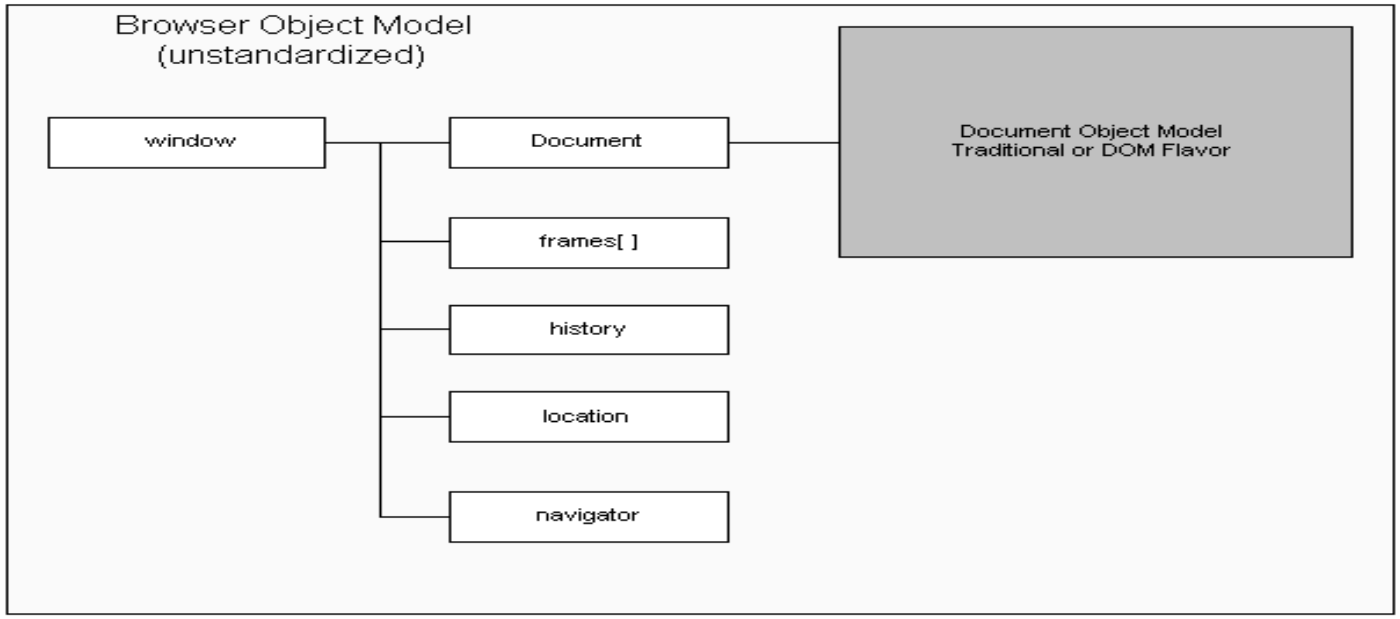
The Ugly Truth

- Unfortunately, the division between the DOM and the BOM at times is somewhat fuzzy and the exact document manipulation capabilities of a particular browser's implementation of JavaScript vary significantly.

The Big Picture

- Looking at the "big picture" of all various aspects of JavaScript including its object models. We see four primary pieces:
  1. The core JavaScript language (e.g. data types, operators, statements, etc.)
  2. The core objects primarily related to data types (e.g. Date, String, Math, etc.)
  3. The browser objects (e.g. Window, Navigator, Location, etc.)
  4. The document objects (e.g. Document, Form, Image, etc.)

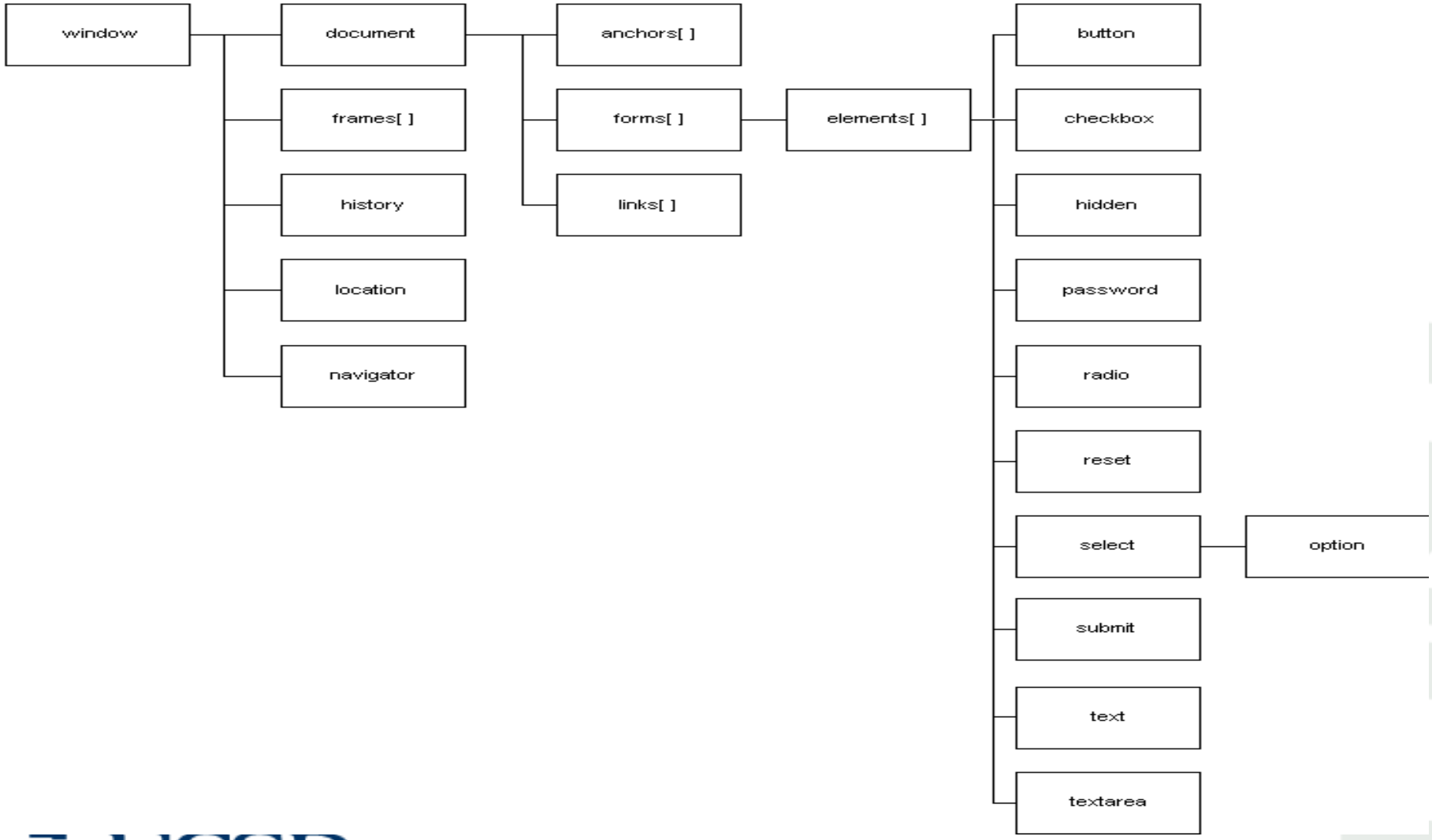

JavaScript Language and Built-in Objects
(standardized under ECMA 262)
Core Language Definition (statements, operators, expressions, etc.)
Math
String
Boolean
Number
Date
Array
Object
Function
RegExp


Browser Object Model
(unstandardized)
window
Document
frames[ ]
history
location
navigator
Document Object Model
Traditional or DOM Flavor

Four Models

- By studying the history of JavaScript we can bring some order to the chaos of competing object models. There have been four distinct object models used in JavaScript including:
  1. Traditional JavaScript Object Model (NS 2 & IE 3)
  2. Extended Traditional JavaScript Object Model (NS 3)
  3. Dynamic HTML Flavored Object Models
     1. a. IE 4
     2. b. NS 4
  4. Traditional Browser Object Model + Standard DOM (NS6 & Explorer 5)
  5. HTML5 model!

Traditional Object Model



Overview of Core Objects

Object	Description
Window	The object that relates to the current browser window.
Document	An object that contains the various HTML elements and text fragments that make up a document. In the traditional JavaScript object model, the **Document** object relates roughly the HTML **<body>** tag.
Frames[ ]	An array of the frames in the Window contains any. Each frame in turn references another **Window** object that may also contain more frames.
History	An object that contains the current window's history list, namely the collection of the various URLs visited by the user recently.
Location	Contains the current location of the document being viewed in the form of a URL and its constituent pieces.
Navigator	An object that describes the basic characteristics of the browser, notably its type and version.

Document Object

- The **Document** object provides access to page elements such as anchors, form fields, and links as well as page properties such as background and text color.
- Consider
  - document.alinkColor, document.bgColor, document.fgColor, document.URL
  - document.forms[ ], document.links[ ], document.anchors[ ]
- We have also used the methods of the **Document** object quite a bit
  - document.write( ) , document.writeln( ), document.open( ), document.close( )

Object Access by Document Position

- HTML elements exposed via JavaScript are often placed in arrays or collections. The order of insertion into the array is based upon the position in the document.
- For example, the first **<form>** tag would be in document.forms[0], the second in document.forms[1] and so on.
- Within the form we find a collection of elements[ ] with the first **<input>, <select>** or other form field in document.forms[0].elements[0] and so on.
- As arrays we can use the length property to see how many items are in the page.
- The downside of access by position is that if the tag moves the script may break

Object Access by Name

- When a tag is named via the **name** attribute (HTML 4.0 - **<a>, <img>,** embedded objects, form elements, and frames) or by **id** attribute (pretty much every tag) it should be scriptable.
- Given

```
<form id="myform" name="myform">
  <input type="text" name="username"
id="username">
</form>
```

we can access the form at window.document.myform and the first field as window.document.myform.username

Object Access by Associative Array

- The collection of HTML objects are stored associatively in the arrays.
- Given the form named “myform” we might access it using

  window.document.forms[“myform”]
- In Internet Explorer we can use the **item( )** method like so

  window.document.forms.item(“myform”)

Event Models

- JavaScript reacts to user actions through event handlers (code associated with a particular event or object and event in the page)
- Common events include Click, MouseOver, MouseOut, etc.
- Events can be registered through HTML event handlers like onclick or via JavaScript directly
  - <input type="button" value="press" onclick="alert('hi')">
  - document.onload = new Function("alert('hi')");
- We'll see events primarily with links, form items and mouse movement

All Together

- Once document objects are accessed either by user event or script event we can then modify the various properties of the elements.
- The following examples on the next slides show reading and writing of form fields as a demonstration of this.

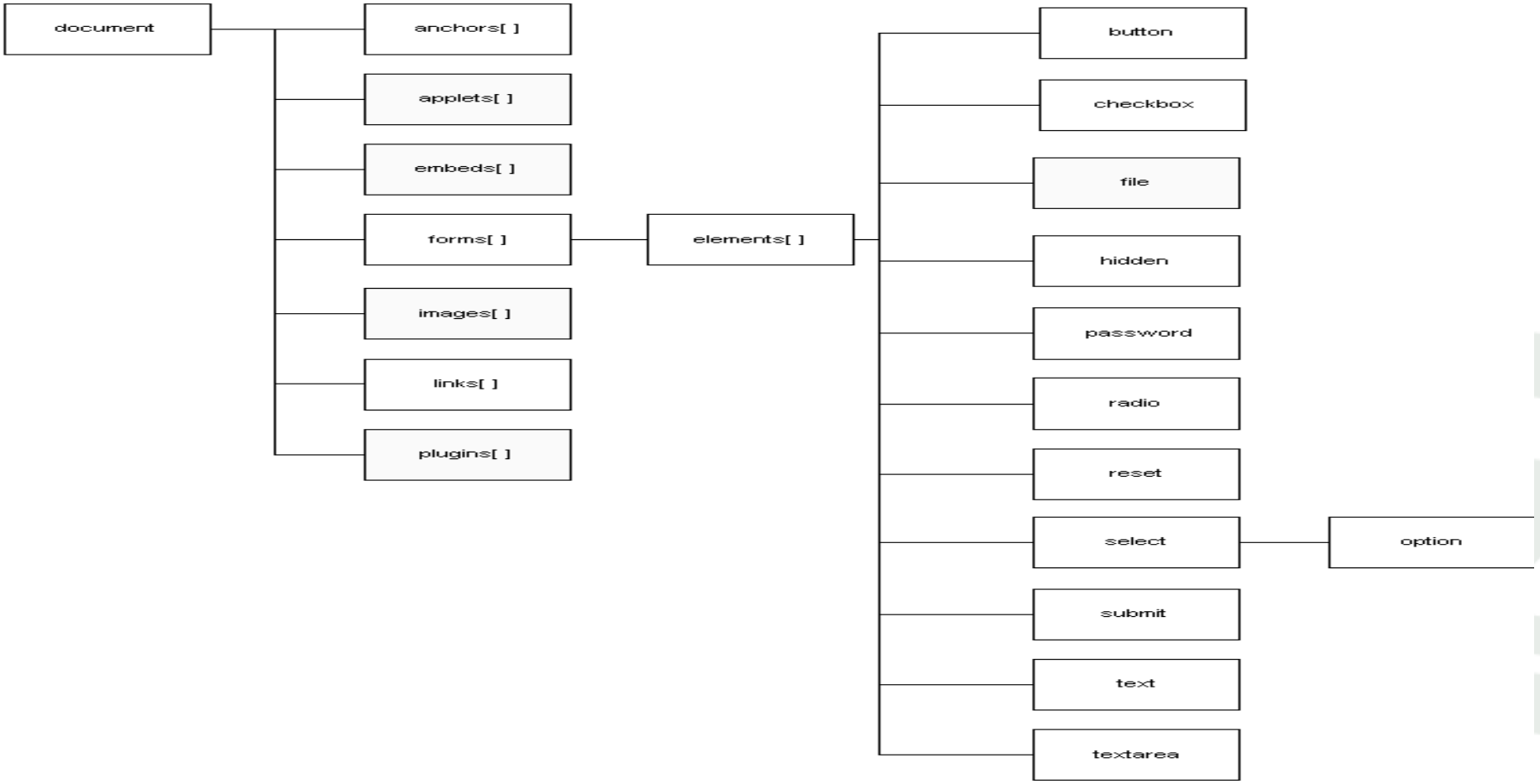
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Meet and Greet</title>
<script language="JavaScript" type="text/javascript">
<!--
function sayHello()
{
 var theirname=document.myform.username.value;
 if (theirname !="")
  alert("Hello "+theirname+"!");
 else
  alert("Don't be shy.");
}
// -->
</script></head><body>
<form name="myform" id="myform">
<b>What's your name?</b>
<input type="text" name="username" id="username"  size="20"><br><br>
<input type="button" value="Greet" onclick="sayHello()">
</form>
</body>
</html>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>Meet and Greet 2</title>
<script language="JavaScript" type="text/javascript">
<!--
function sayHello()
{
 var theirname = document.myform.username.value;
 if (theirname != "")
  document.myform.response.value="Hello "+theirname+"!";
 else
  document.myform.response.value="Don't be shy.";
}
// -->
</script></head><body>
<form name="myform" id="myform">
<b>What's your name?</b>
<input type="text" name="username" id="username"  size="20">
<br><br>
<b>Greeting:</b>
<input type="text" name="response" id="response" size="40">
<br><br>
<input type="button" value="Greet" onclick="sayHello()">
</form></body></html>
```

The Object Models

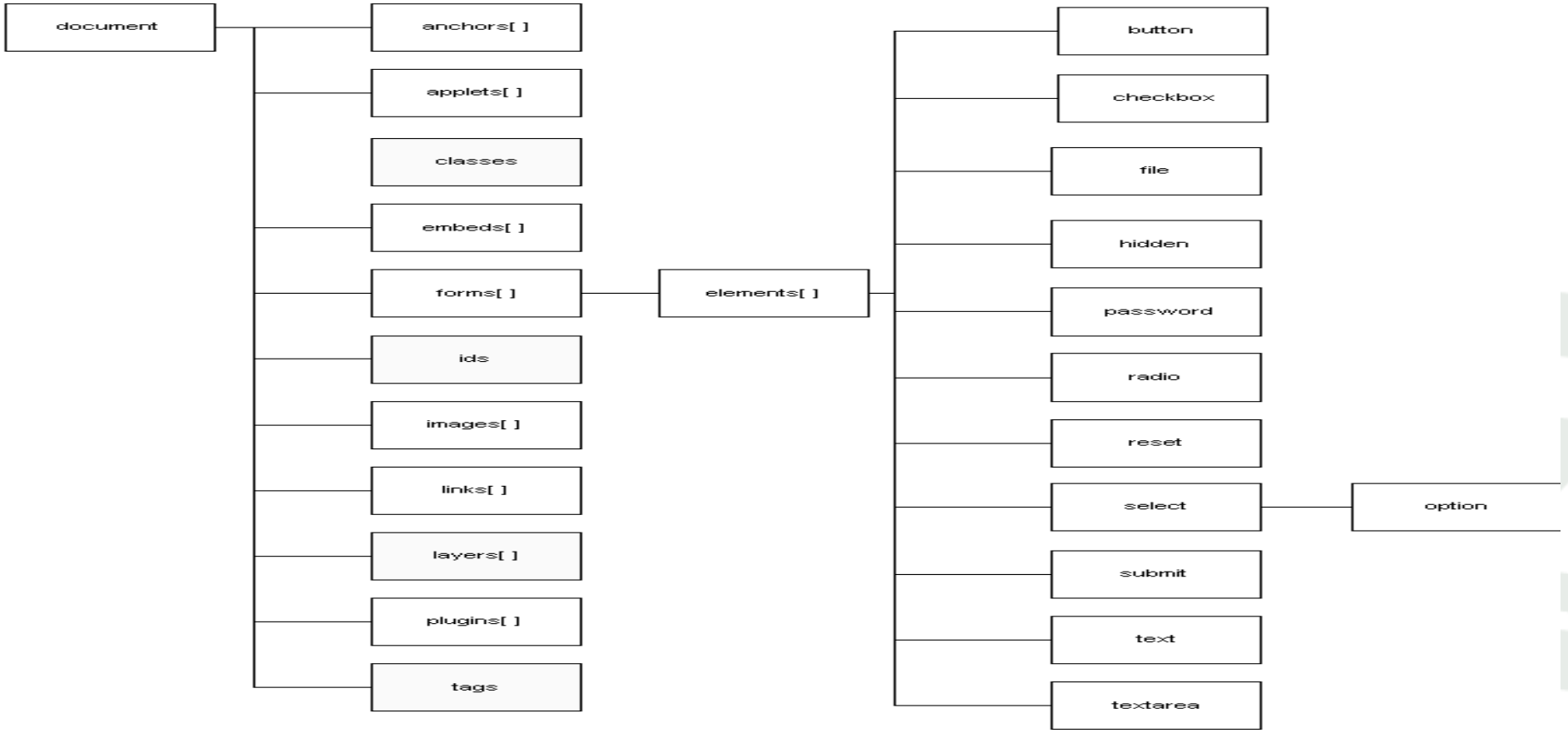
- The next few slides present the various object models supported pre-standard DOM. In JavaScript 1 we focus primarily on the Netscape 3 DOM with some introduction to the non-standard DHTML object models.

Specific Object Models: Netscape 3

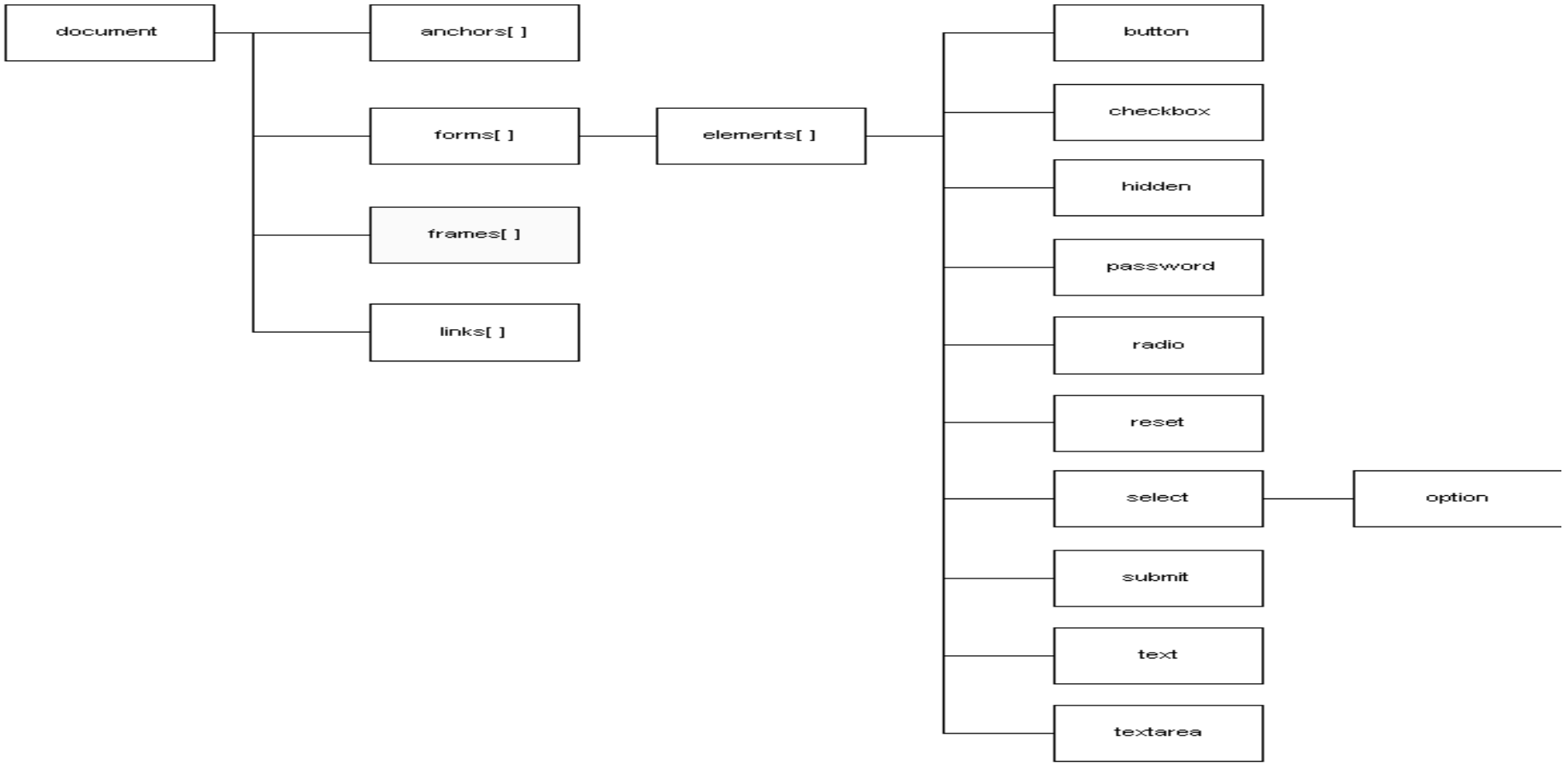


New in Navigator 3

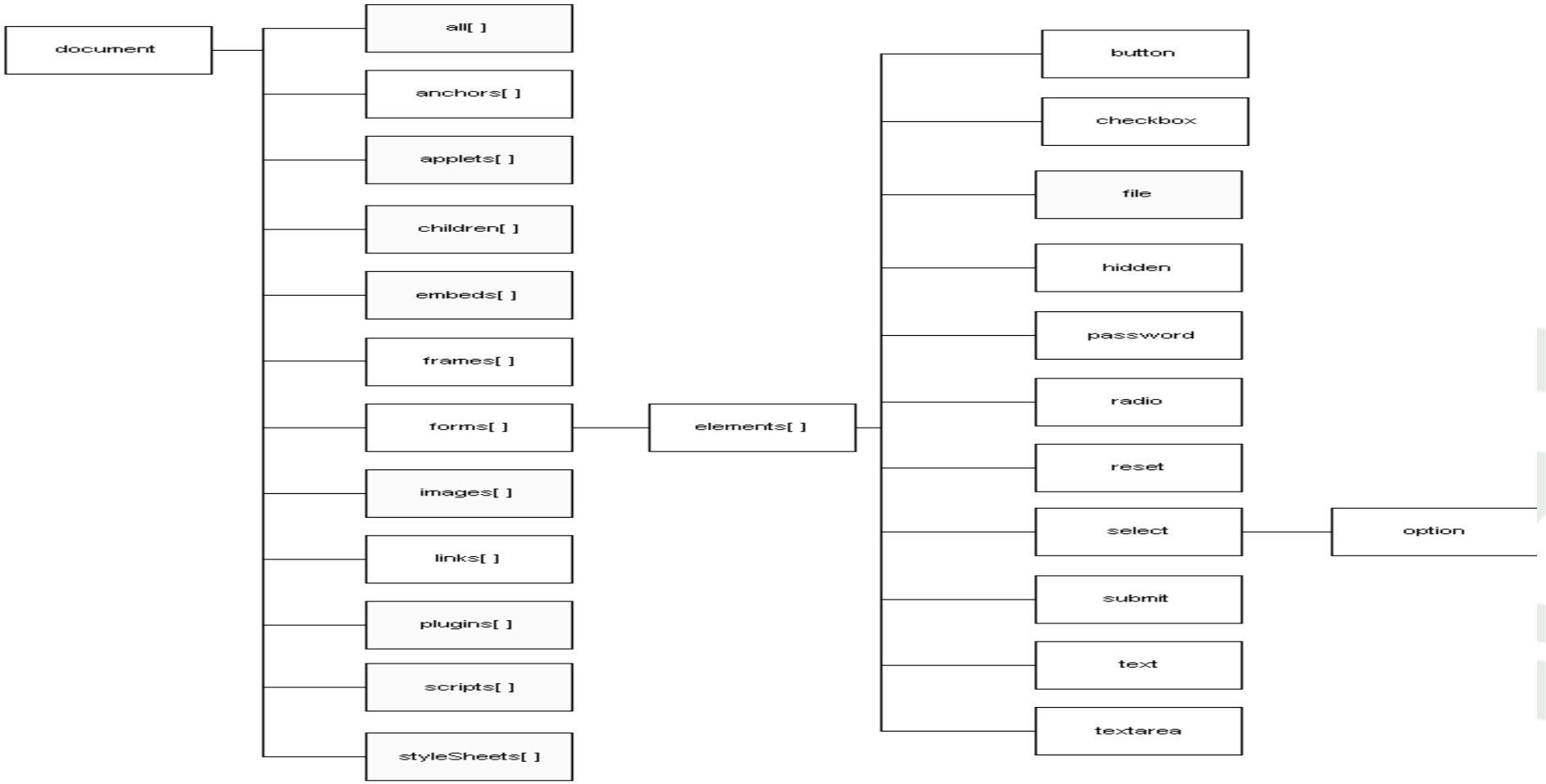
Specific Object Models: Netscape 4



Specific Object Models: Internet Explorer 3



Specific Object Models: Internet Explorer 4



The Cross Browser Nightmare

- The problem we face with JavaScript is that each object model is different
- Somehow we either have to find a common ground (traditional model), use object detection, use browser detection, pick a particular object model like IE and stick with it or just hope the standards work out
- We'll see with the rise of the Document Object Model (DOM) that someday maybe only certain BOM features will be non-standard and all browsers will have the same ability to manipulate page content.

Modern Access Solutions

- document.getElementById()
  - id is not a name replacement completely, think form fields (name-value pairs)
- document.getElementsByClassName()
- document.querySelectorAll()
- $() this is a wrapper function folks!
- Modern doesn't always equal better as we'll see…things are still a mess at times

Summary

- This chapter introduces how to access browser and document objects in the traditional manner
  - By name and by document location
- All the various object models were presented from NS 2 to modern day browsers
- The problem of cross browser compatibility starts to become apparent when you compare the models