Javascript

Introduction

What Is JavaScript?

- Executes in Host Environment (mostly Browser).
- Interpreted language.
- Major use cases are:
  a. making web pages dynamic (animations, RICH UI)
  b. form validation
  c. background communication with server (AJAX) etc.
  d. ...

History

- Initially LiveScript from Netscape.
- JScript from Microsoft.
- ECMA introduced ECMAScript for standardized Scripting language.
- Current Version is 5.1 of ECMA-262.

The Core (ECMAScript)

- ECMA-262 describes it like this:

  "ECMAScript can provide core scripting capabilities for a variety of host environments, and therefore the core scripting language is specified...apart from any particular host environment."

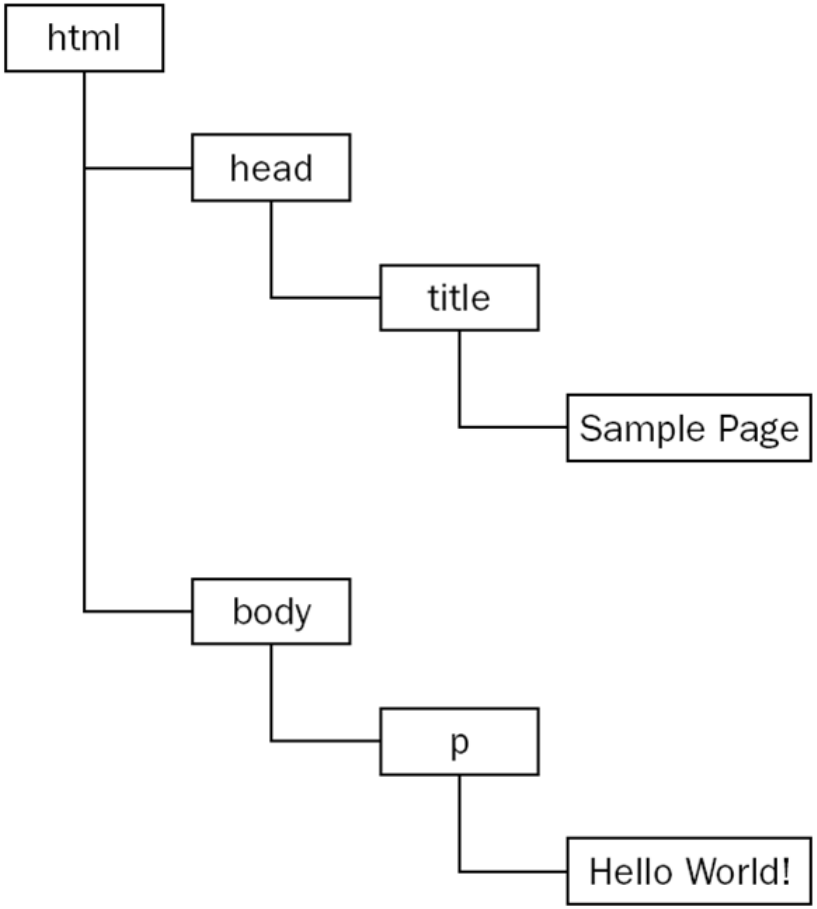
```
                    ECMAScript
                          /
          ----------------------------------------
-----
          /              /                  /
     JavaScript     Actionscript      ScriptEase
```

The Document Object Model (DOM)

- The *Document Object Model* (DOM) is an application programming interface (API) for HTML as well as XML.

DOM

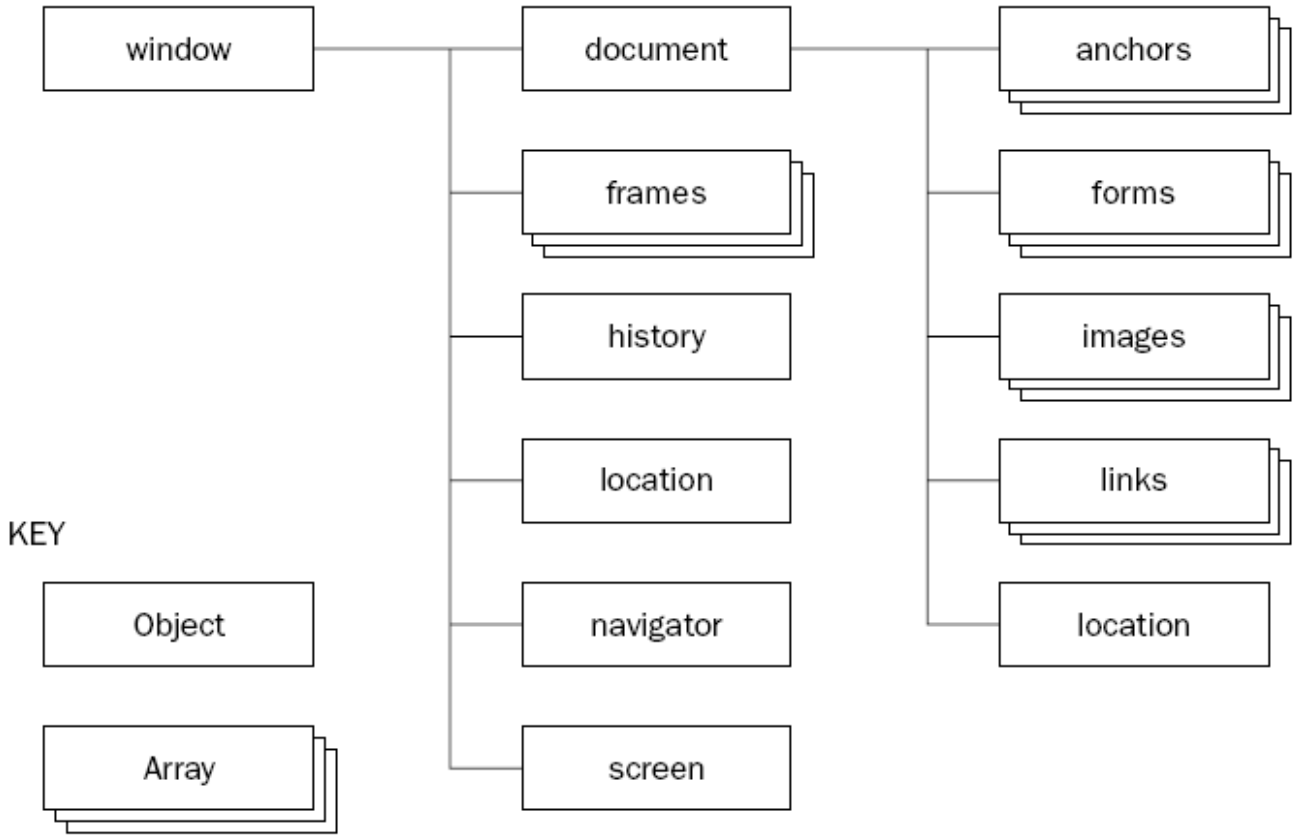
```
<html>
  <head>
      <title>Sample
  Page</title>
  </head>
  <body>
      <p>Hello
  World!</p>
  </body>
</html>
```



The Browser Object Model (BOM)

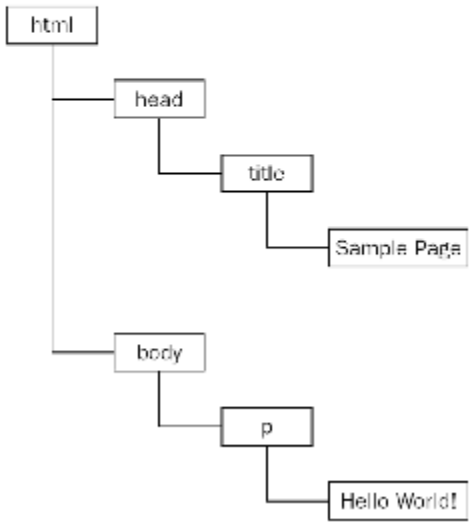
- BOM deals with the browser window and frames.
- All function and properties starting inside window object.
- Internet Explorer extends the BOM to include the ActiveXObject class, which can be used to instantiate ActiveX objects through JavaScript.

Browser Object Model (BOM)



DOM and JS World

Dom World



JS World

Object

key	value

How to include JS in a web page

Inside <script> tags

```
<script>
    var a = 10;
    alert(a);
</script>
```

Within external file

```
<script src="page.js"
type="text/javascript" ></script>
```

Javascript Basics

Syntax

- Mostly like C and java.
- Everything is case-sensitive.
- Variables are loosely typed.
- End-of-line semicolons are optional.
- Comments are the same as in Java, C, and Perl.

Variables

- var test = "hi“, test2, $number;
- Variable declaration is optional.

Keywords

break	else	new	var
case	finally	return	void
catch	for	switch	while
continue	function	this	with
default	if	throw	
delete	in	try	
do	instanceof	typeof	

Reserved Words

abstract enum int short
boolean export interface static
byte extends long super
char final native
synchronized
class float package throws
const goto private transient
debugger implements protected volatile
double import public

Statements

- **The if statement**
  - `if (`*condition*`)` *statement1* `else` *statement2*
- **do-while**
  - `do { statements } while (condition )`
- **While**
  - `while (condition) { statements}`
- **for**
  - `for (initialization; condition; post-loop-expression) { statements}`
- **for-in**

  `for (property in expression) {statements}`

Data types

- Primitive values
- Reference values

Primitive values

- undefined
- null
- boolean
- number
- string

Use `typeof` keyword to check type of variable.

Reference Types

- Classes or Object
- The Object class – similar to java.lang.Object

```
var car = new Object();
```

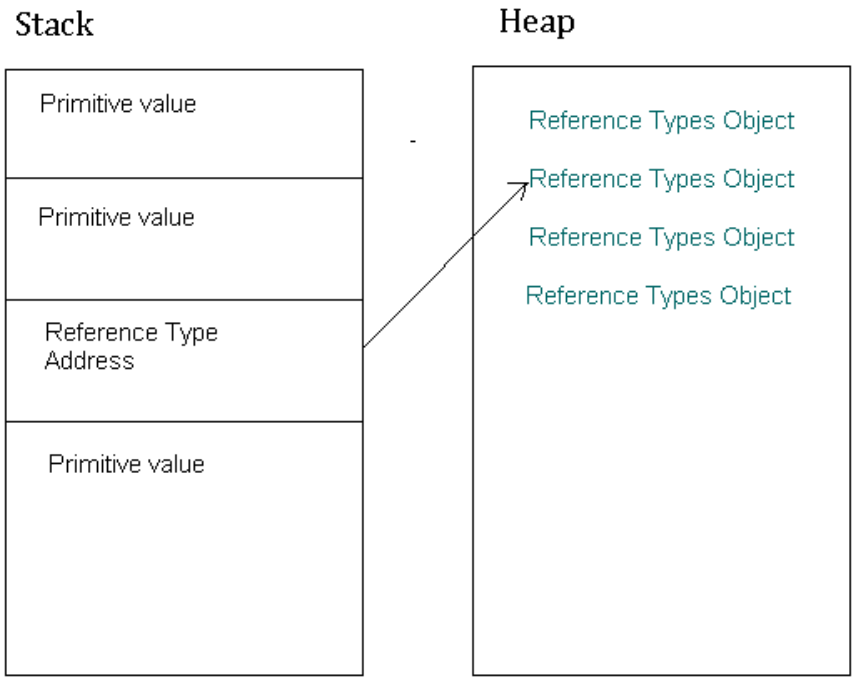
Or

```
var myClass = function(){};
var myObject = new myClass();
```

Or JSON way

```
var myObject = {};
```

Memory Allocation



Classes and objects in javascript

- No class keyword available, instead constructor function works as class definition.
- Classes and objects are dynamic, can be altered at runtime.

Builtin Objects

Object	Function	Array	String
Boolean	Number	Date	RegExp
Error	EvalError	RangeError	ReferenceError
SyntaxError	TypeError	URIError	

Function defination

```
function showInfo(a) {
     alert(a);
}

var showInfo = function(a) {
alert(a);}

var showInfo = new Function(“a”,
“alert(a)”);
```

Functions (contd.)

- Functions are also object, in fact everything is an object in JavaScript
- Functions can return any data type, ‘undefined’ if none specified.

Functions scope

By default in `window` scope

`this` points to current object's scope, defaults to `window`

Can be altered using `call` and `apply` method

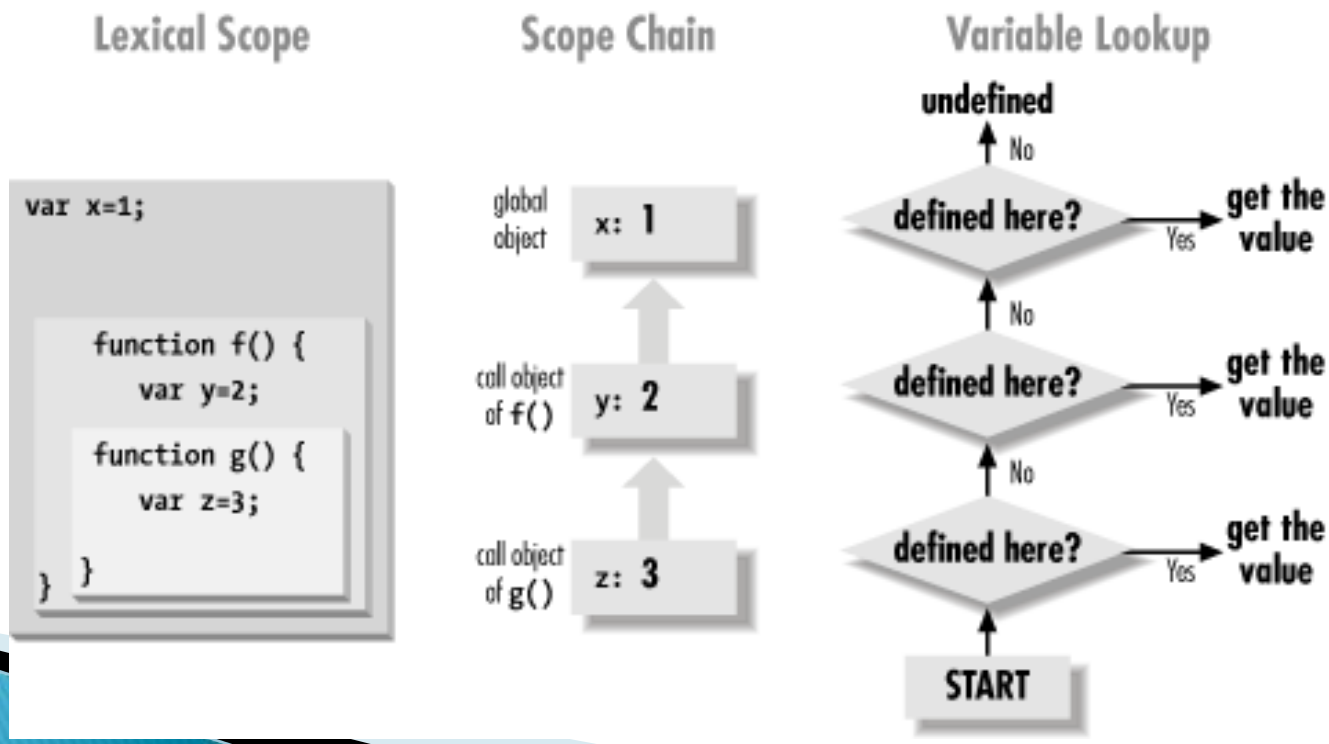
```
func.call(any_scope, [arg1, arg2, ..]);
func.apply(any_scope, arg1, arg2,..);
```

Using function as Class

```
var Policy = function(name) {
                this.policyName = name;
            }
var pol1 = new Policy("AccessPolicy");
var pol2 = new Policy("AuthenticationPolicy");

console.log(pol1.policyName); // AccessPolicy
console.log(pol2.policyName); //
AuthenticationPolicy
```

Variables scope

1. Start from local scope and ends till global scope.



function closures

- A **closure** is a function having access to the parent scope, even after the parent function has closed.
- JavaScript variables can belong to the **local** or **global** scope.
- Global variables can be made local (private) with **closures**.

Example

```
function sayHello2(name)
{
var text = 'Hello ' + name; // Local variable
var say = function()
{ console.log(text);
 } return say;
}
var say2 = sayHello2('Raj');
say2();
```

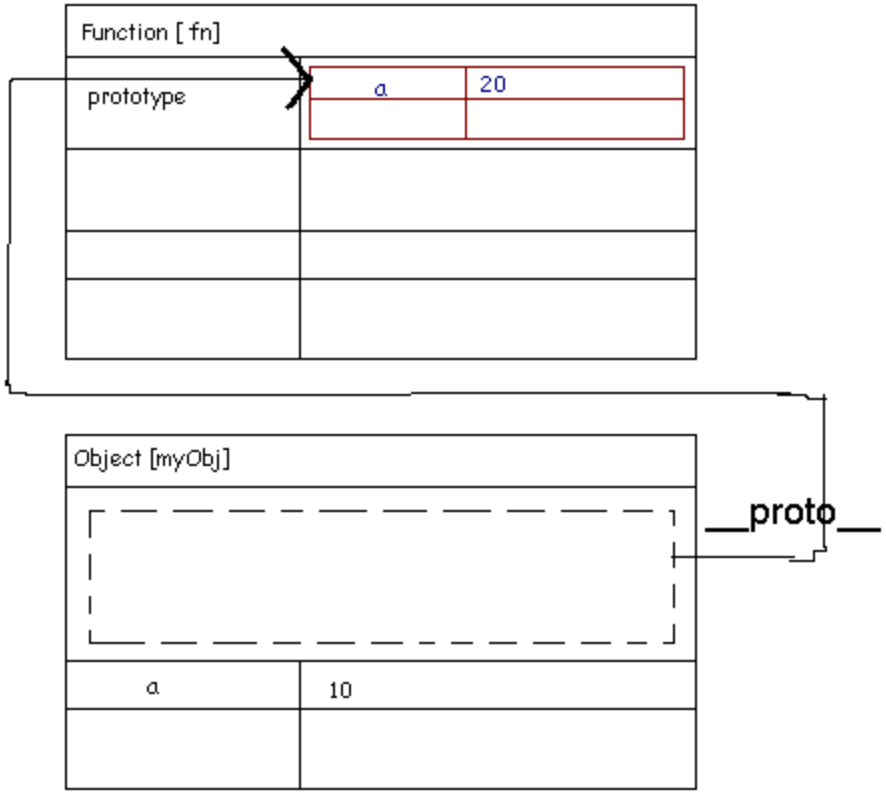
Prototype property

Every function has a prototype property

Every object's scope chain fall backs to constructor function's prototype.

```
var func = function() {this.a=10; }
func.prototype.a = 20;
var obj = new func();
console.log(obj.a); // 10
delete obj.a;
console.log(obj.a); // 20
```

Prototype



Prototype facts

- Prototype object is common across all instances of the function(class)
- Prototype is dynamic, any changes made in prototype gets reflected in all object instances instantly.

__proto__ and constructor

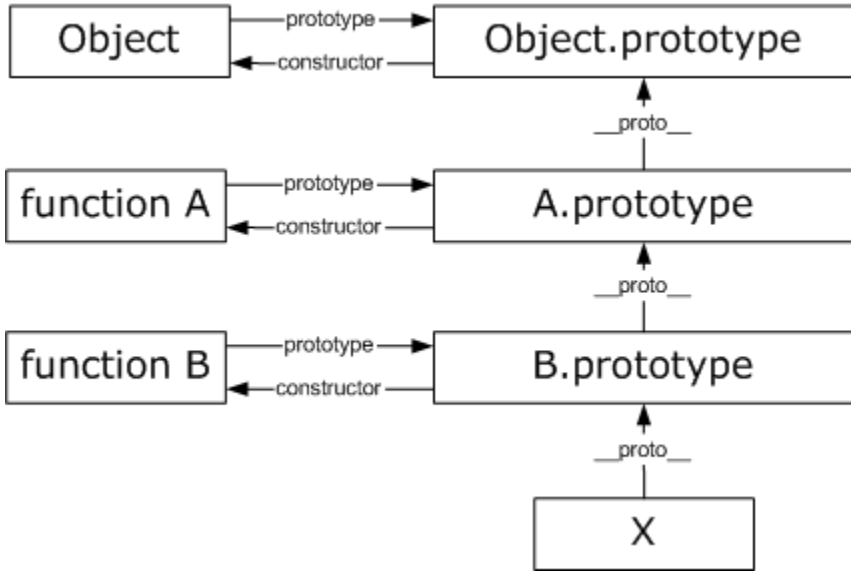
```
var func = function() { }
func.prototype.a = 10;
var obj = new func();
obj.a; // 10
obj.__proto__.a ; // 10
obj.constructor.prototype.a; // a
```

Using prototype to create class

```
var funcA = function() {this.a = 10;}
funcA.prototype.b = 20;
funcA.prototype.doSomething = function() { }

var objA = new funcA();
funcA.prototype.c = 30;
console.log(objA);
```

Prototype chaining

```
var A = function(){};

var B = function(){};
B.prototype = new A();
B.prototype.constructor
= B;

var X = new B();
```



JSON

JavaScript Object Notation

```
Array []
Object {“key”:”value, ….}

var obj = {};
obj.a = 10;
obj.doSomething = function(){};
```

Using JSON to create object and class

```
var myClass = function (arg1) {
            var _property1  = 20;
            var  _method = function() {};
            return {
                  public_property:arg1,
                  public_method:function() {
                        _method();
                        }
                  };
}
var myObject = new myClass(10);
```

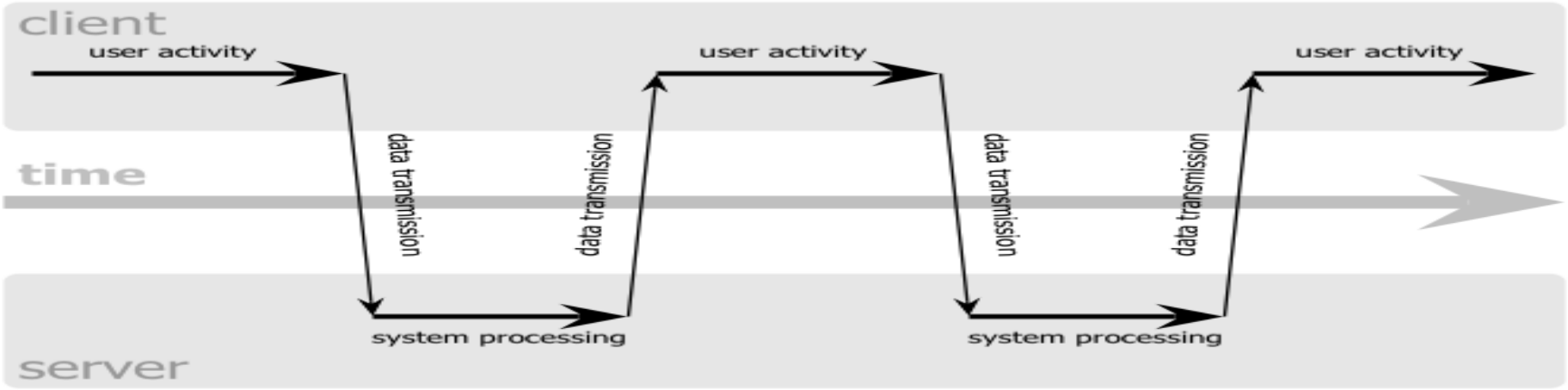
AJAX

Asynchronous JavaScript And (Advanced) XML

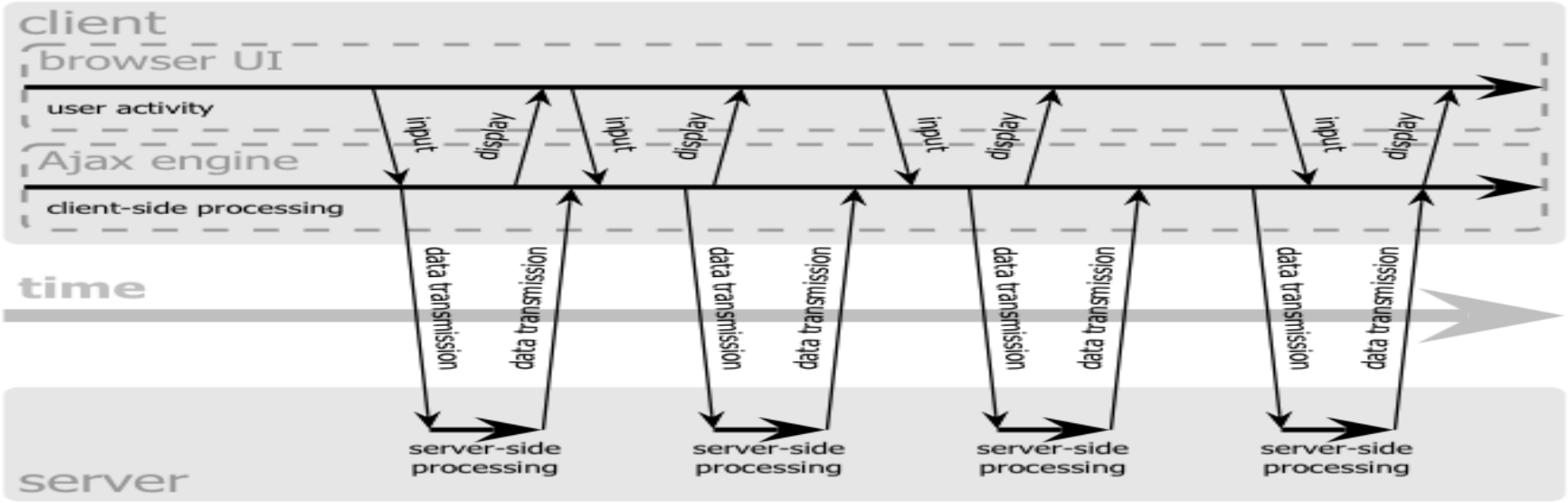
XMLHttpRequest Object

AJAX Request

classic web application model (synchronous)



Ajax web application model (asynchronous)



XMLHttpRequest – cross browser Support

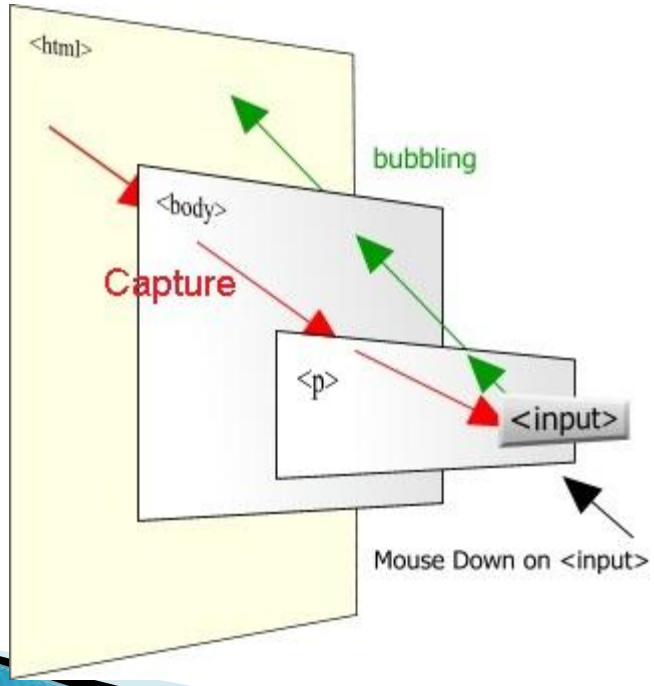
```
if (typeof XMLHttpRequest == "undefined") {
 XMLHttpRequest = function () {
  try { return new
   ActiveXObject("Msxml2.XMLHTTP.6.0"); } catch (e)
{}
  try { return new
   ActiveXObject("Msxml2.XMLHTTP.3.0"); } catch (e)
{}
  try { return new
   ActiveXObject("Microsoft.XMLHTTP"); } catch (e)
{}
  throw new Error("This browser does not support
XMLHttpRequest."); };
```

Ajax example

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange =
 function() {
  if(xhr.readyState == 4){
     alert(xhr.responseText);
  }
 };
xhr.open("GET","page.xml",true);
xhr.send(null);
```

Event Handling

Event propagation and event Bubbling



IE only support event bubbling

Adding event Handlers

Mozilla compatible browsers

```
[elementObject].addEventListener("event_handler", handlerFunction, boolCapture);
[elementObject].removeEventListener("event_handler", handlerFunction, boolCapture);
```

IE

```
[elementObject].attachEvent("event_handler", handlerFunction);
[elementObject].detachEvent("event_handler", handlerFunction);
```

Timing functions

setTimeout- calls only once

```
var timeoutVar = setTimeout(
function_to_call, milliseconds);
clearTimeout(timeoutVar);
```

setInterval - calls repeatedly

```
var intervalVar = setInterval(
function_to_call, milliseconds);
clearInterval(intervalVar);
```

DOM API

```
document.getElementById
document.getElementsByTagName

document.createElement
document.appendChild

elementObject.innerHTML
```


Thank
You!