Open Source Software Project Development

Dr. T.Y. Wong

Weeks 9 - 10

JavaScript Programming (2)

- Highlighting the features that are used most.

Client-side JavaScript

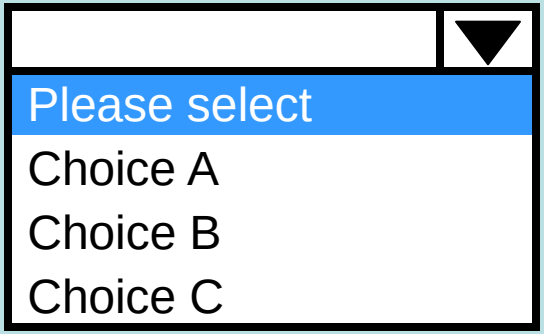
- Event Handling

See http://demo4140-tywong.rhcloud.com/10_js_event/

What is an event?

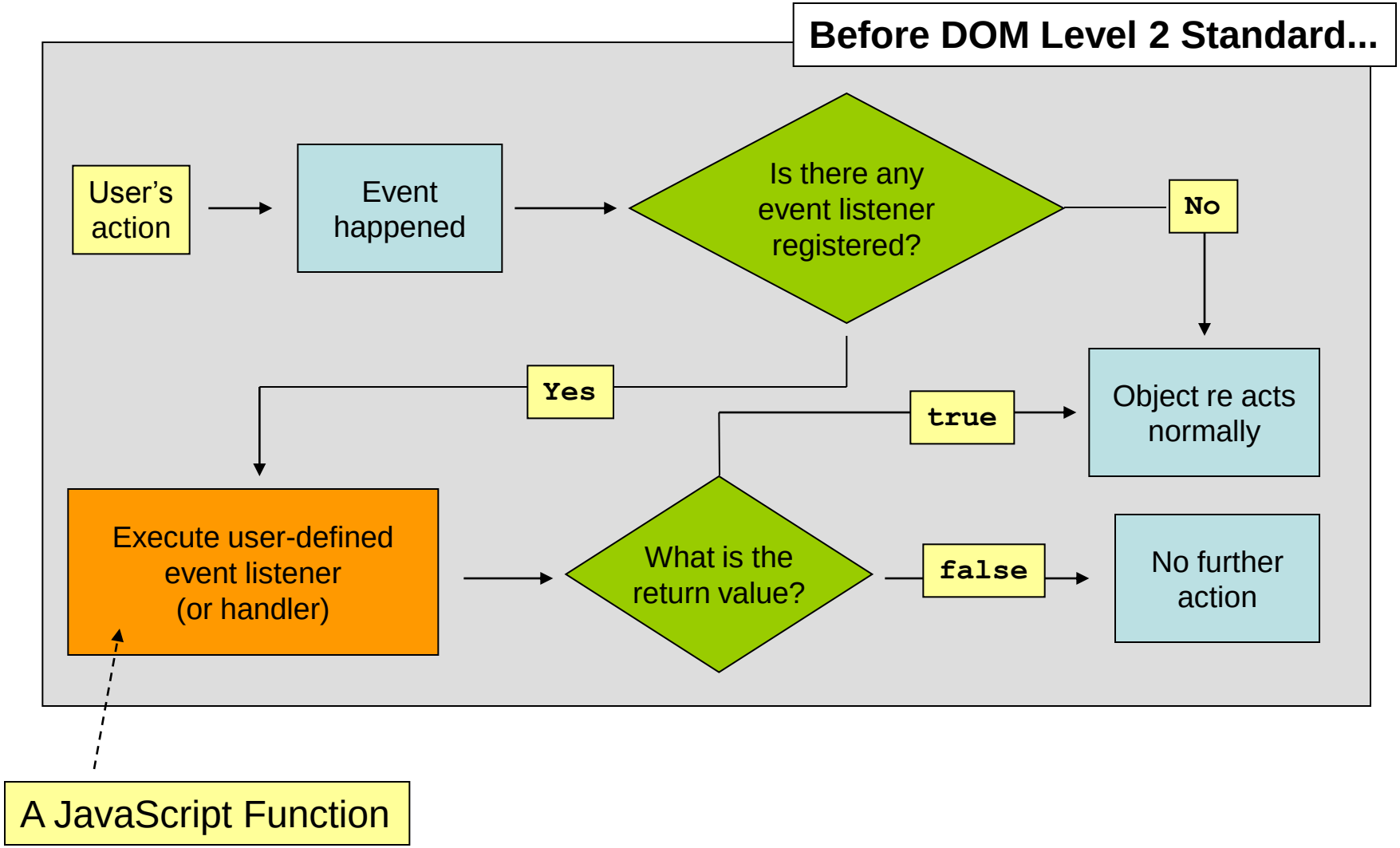
- An event is a control inside the browser.
  - In simple words, it is an action and will be trigger when a pre-defined condition happens.

Event	Triggered When…
`blur`	Element loses input focus
`change`	
`click`	Mouse press and release.
`dblclick`	Double click.
`submit`	Form submission.
`.....`	`.....`

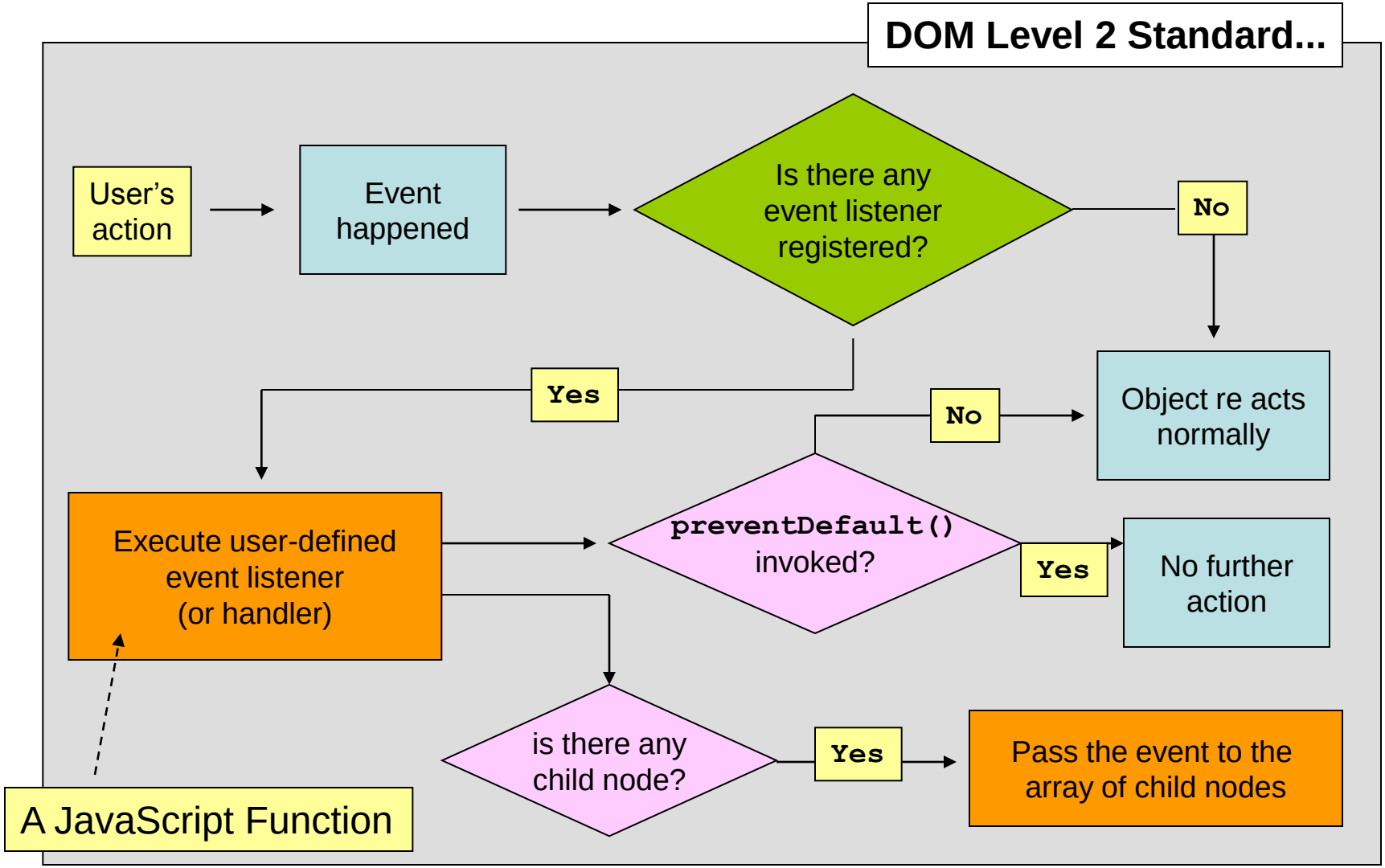


When you change the selection menu a “change” event will be triggered.

What is event handling?

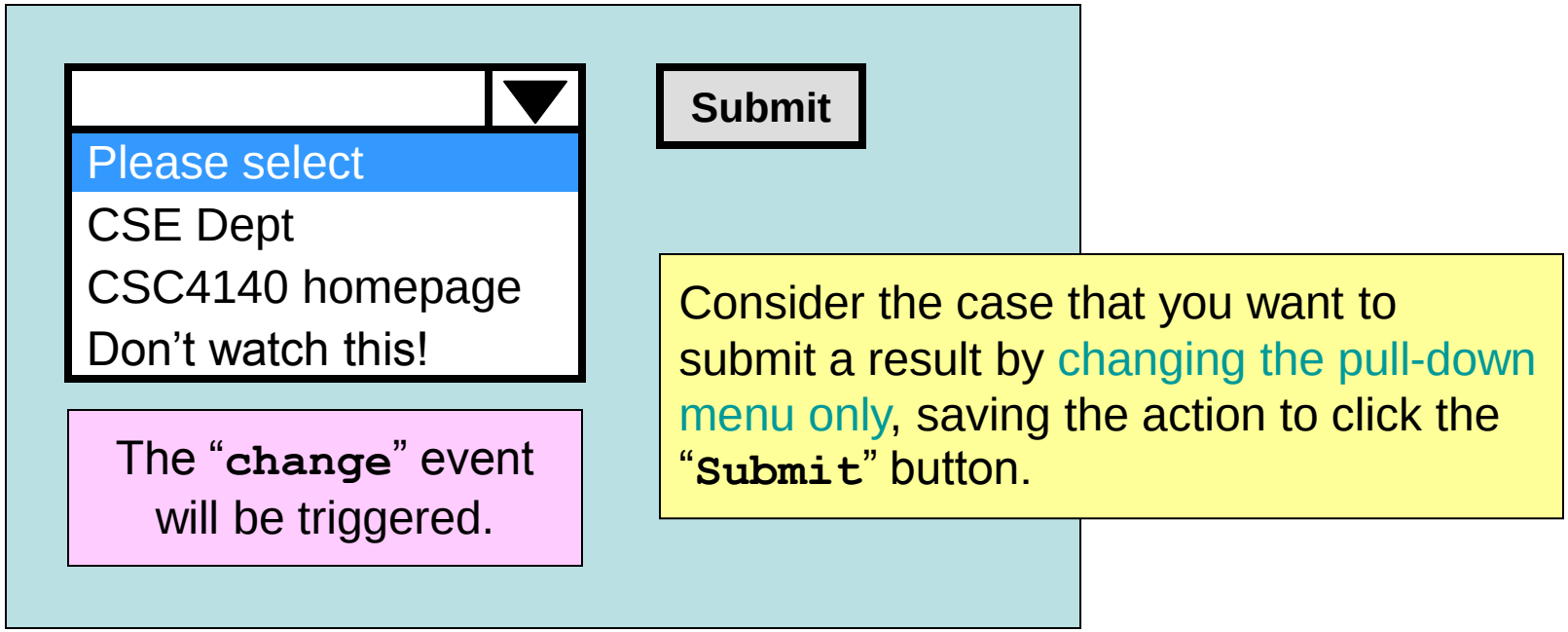


What is event handling?



Adding an event listener…

- There are two ways!
  - Let's start with easiest example…



See "select.html", "select_script.html"

Adding an event listener…

Before DOM Level 2

Must refer to: http://www.w3schools.com/jsref/dom_obj_select.asp

```
function change_link()   {
     ......
}
```

"`change_link()`" will be the JavaScript function that handles the event.

"`onchange`" means to listen to the "**change**" event.

```
<select id=select onchange="change_link();">
    <option value="none" selected>Please select</option>
    <option value=[URL]>CSE dept</option>
    <option value=[URL]>4140 homepage</option>
    <option value=[URL]>Don't watch this!</option>
</select>
```

Adding an event listener…

Before DOM Level 2

Must refer to: http://www.w3schools.com/jsref/dom_obj_select.asp

```
function change_link()  {
    var list = document.getElementById("select");
    var n = list.selectedIndex;

    if(list.options[n].value == "none")
        return false;

    windows.location = list.options[n].value;
    return true;
}
```

Under this case, an **object** with the ID "select" will be returned.

```
<select id=select onchange="change_link();">
    <option value="none" selected>Please select</option>
    <option value=[URL]>CSE dept</option>
    <option value=[URL]>4140 homepage</option>
    <option value=[URL]>Don't watch this!</option>
</select>
```

Adding an event listener…

DOM Level 2

- In DOM Level 2, the "**`on*`**" way to register event listener is gone.

A minor change on HTML part

"onchange" is removed because we will use JavaScript to add the listener.

```
<select id=select>
    <option value="none" selected>Please select</option>
    <option value=[URL]>CSE dept</option>
    <option value=[URL]>4140 homepage</option>
    <option value=[URL]>Don't watch this!</option>
</select>
```

Adding an event listener…

```
<script>

    window.addEventListener("load", init, false);
</script>
```

Add a listener to the "**`window`**" object.

Event: load ← Means: document load complete.

Handler: `init()`

See "select_dom2.html", "select_script_dom2.html"

Adding an event listener…

```
<script>

function init() {
     document.getElementById("select").
              addEventListener("change", change_link, false);
}

window.addEventListener("load", init, false);
</script>
```

Add a listener to the "**`select`**" object.

Event: change

Handler: `change_link()`

See "select_dom2.html", "select_script_dom2.html"

Adding an event listener…

```
<script>
```

You may wonder: why init() is needed?

```
document.getElementById("select").
        addEventListener("change", change_link, false);
```

They work in the same way! Why adding a listener for the window object?!

```
function init() {
    document.getElementById("select").
            addEventListener("change", change_link, false);
}

window.addEventListener("load", init, false);
</script>
```

See "select_dom2.html", "select_script_dom2.html"

Adding an event listener…

```
<script>
```

The reason is that:

The browser executes the script first because
it is located higher than the HTML code!

So, at the time this statement is called,
the "select" object is not created yet!

```
document.getElementById("select").
     addEventListener("change", change_link, false);
</script>
```

See "select_dom2.html", "select_script_dom2.html"

Adding an event listener…

```
<script>
    function change_link( e ) {
        var list = document.getElementById("select");
        var n = list.selectedIndex;

        if(list.options[n].value == "none")
            return false;

        parent.frames["display_frame"].location = list.options[n].value;
        return true;
    }

    function init() {
        document.getElementById("select").
                addEventListener("change", change_link, false);
    }

    window.addEventListener("load", init, false);
</script>
```

This is the **event object**. So, you will know, at least, what the event is about.

This function is kept unchanged.

See "select_dom2.html", "select_script_dom2.html"

The Event Object

Most-Frequently-Used Types of Event.	
`change`	Apply to: select, textarea, input Triggered when: Value changes.
`click, dblclick, mouseup, mousedown, mousemove, mouseout, mouseover`	Apply to: most objects Triggered when: see example – div_event.html
`keypress, keydown, keyup`	Apply to: form elements and body Triggered when: see example – textarea_event.html
`resize`	Apply to: body, frameset, window Triggered when: resize.
`submit`	Apply to: form Triggered when: submit button is pressed.
`load`	Apply to: body, frameset, window, img Triggered when: load complete.
`beforeunload`	Apply to: body, frameset Triggered when: document is about to be unloaded.

See “div_event.html”, “textarea_event.html”

The Event Object

Event Object	
`Object target;`	The target object;
`Object currentTarget;`	The target object or the object that the event propagates to [see next page].
`function preventDefault();` `readonly boolean cancelable;`	Turn off the default action of the browser if the action is cancelable.
`function stopPropagation();`	[see next page]
`integer offsetX, offsetY;`	The coordinates at which the event occurred, with respect to the source element.
`integer clientX, clientY;`	The coordinates at which the event occurred, with respect to the document.
`String type;`	The event type.
`integer keyCode;`	The Unicode of the input key.
`integer button;`	The mouse click; 0 – left; 1 – middle; 2 – right;
`boolean altKey, CtrlKey, shiftKey;`	Are those keys on?

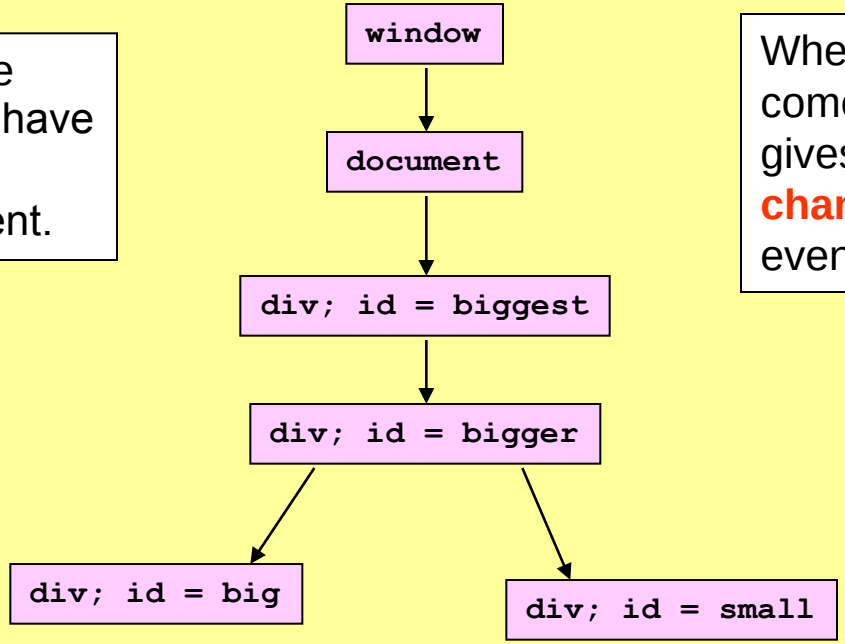
See “div_event.html”, “textarea_event.html”

Event Propagation…

true/false

```
[div].addEventListener("click", foobar, ?????);
```

For example, all the "**div**" components have registered the "**mouseclick**" event.



When a mouse click comes, the browser gives each component **2 chances** to handle the event.

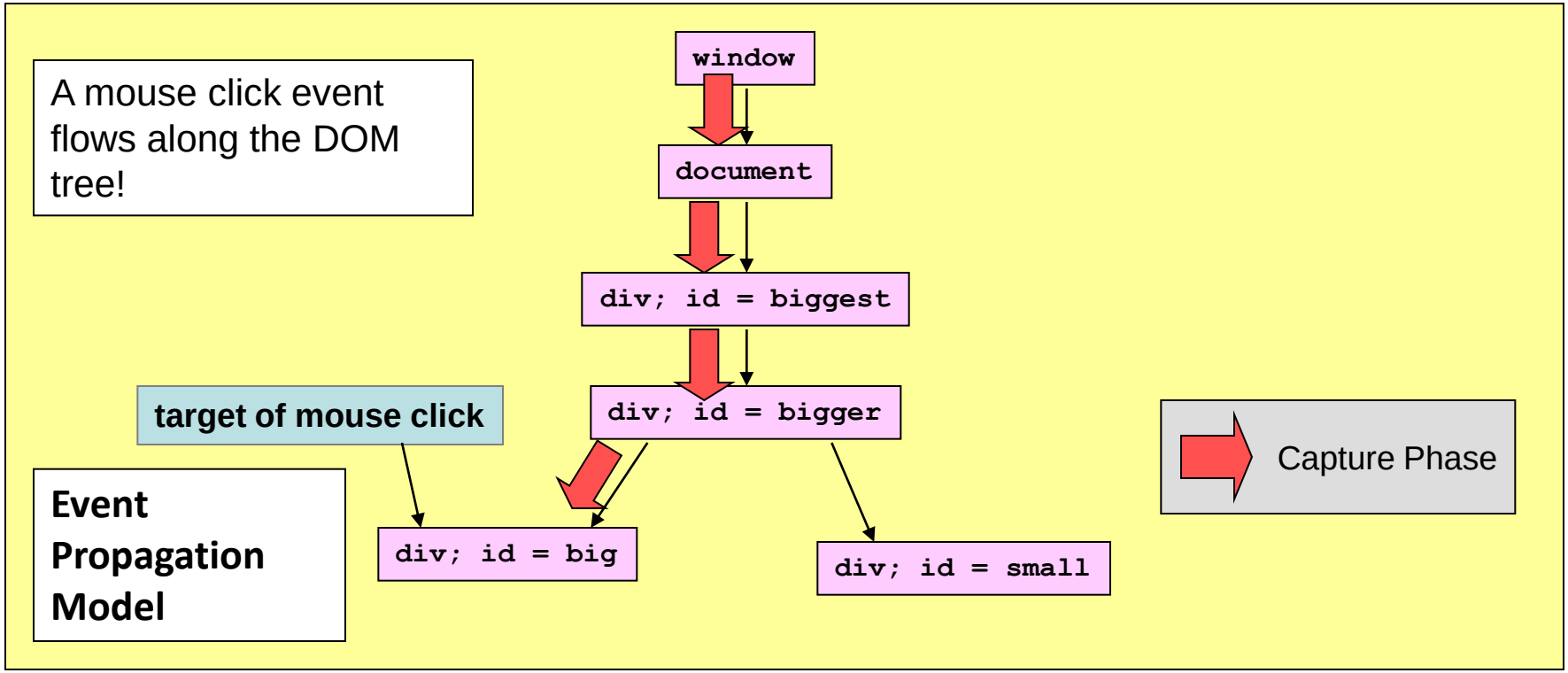
Event Propagation Model

See "event_propagation.html"

Event Propagation…capture phase

We say that: an object will **handle the event during the capture phase**.

```
[div].addEventListener("load", init, true );
```

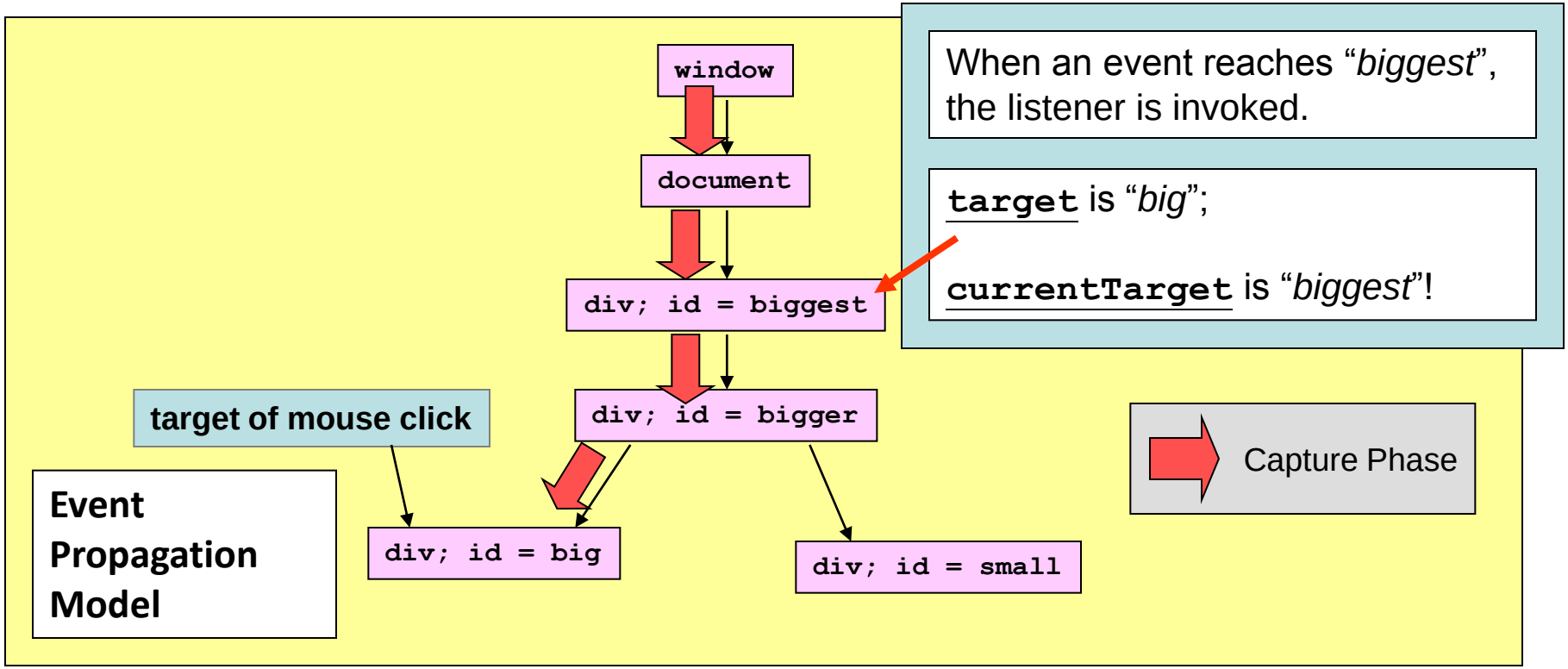


See "event_propagation.html"

Event Propagation…capture phase

We say that: an object will **handle the event during the capture phase**.

```
[div].addEventListener("load", init, true);
```



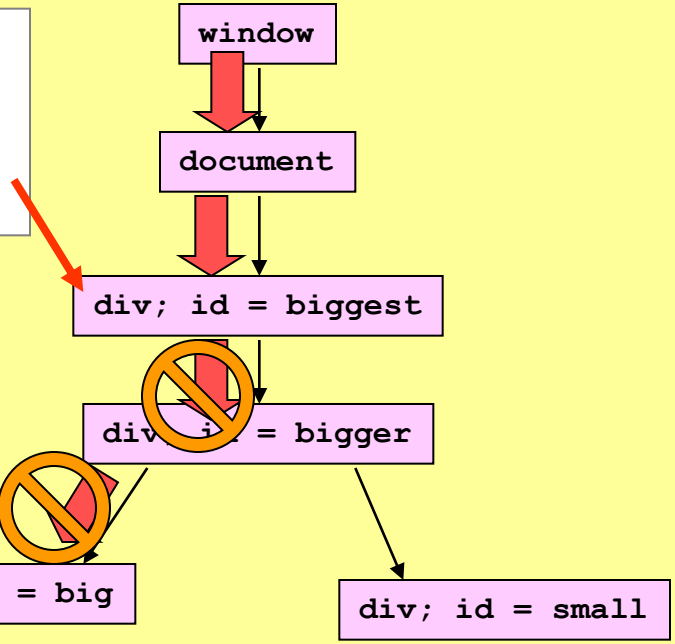
See "event_propagation.html"

Event Propagation…capture phase

We say that: an object will **handle the event during the capture phase**.

```
[div].addEventListener("load", init, true );
```

When the "`stopPropagation()`" method is invoked inside the event listener...



target of mouse click

Event Propagation Model

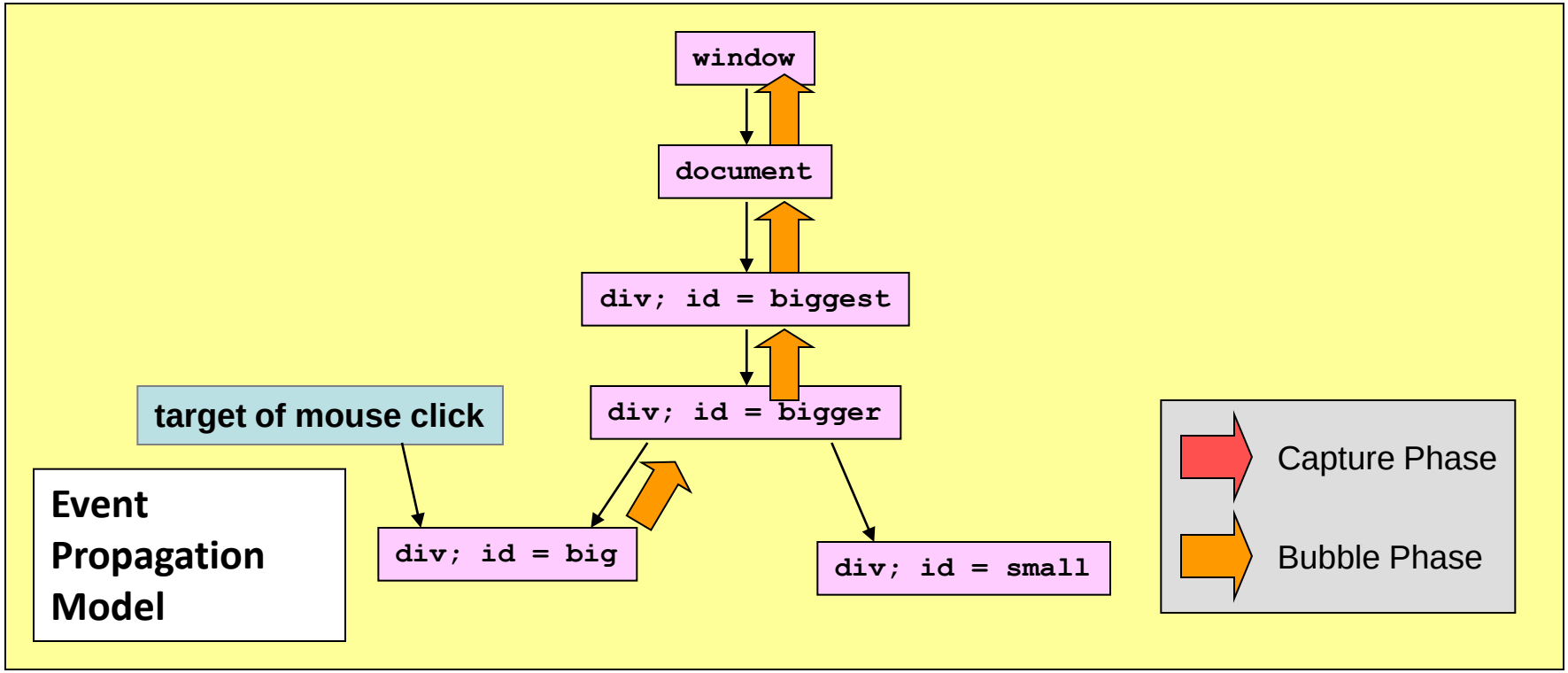
Capture Phase

See "event_propagation.html"

Event Propagation…bubble phase

We say that: an object will **handle the event during the bubble phase**.

```
[div].addEventListener("load", init, false);
```



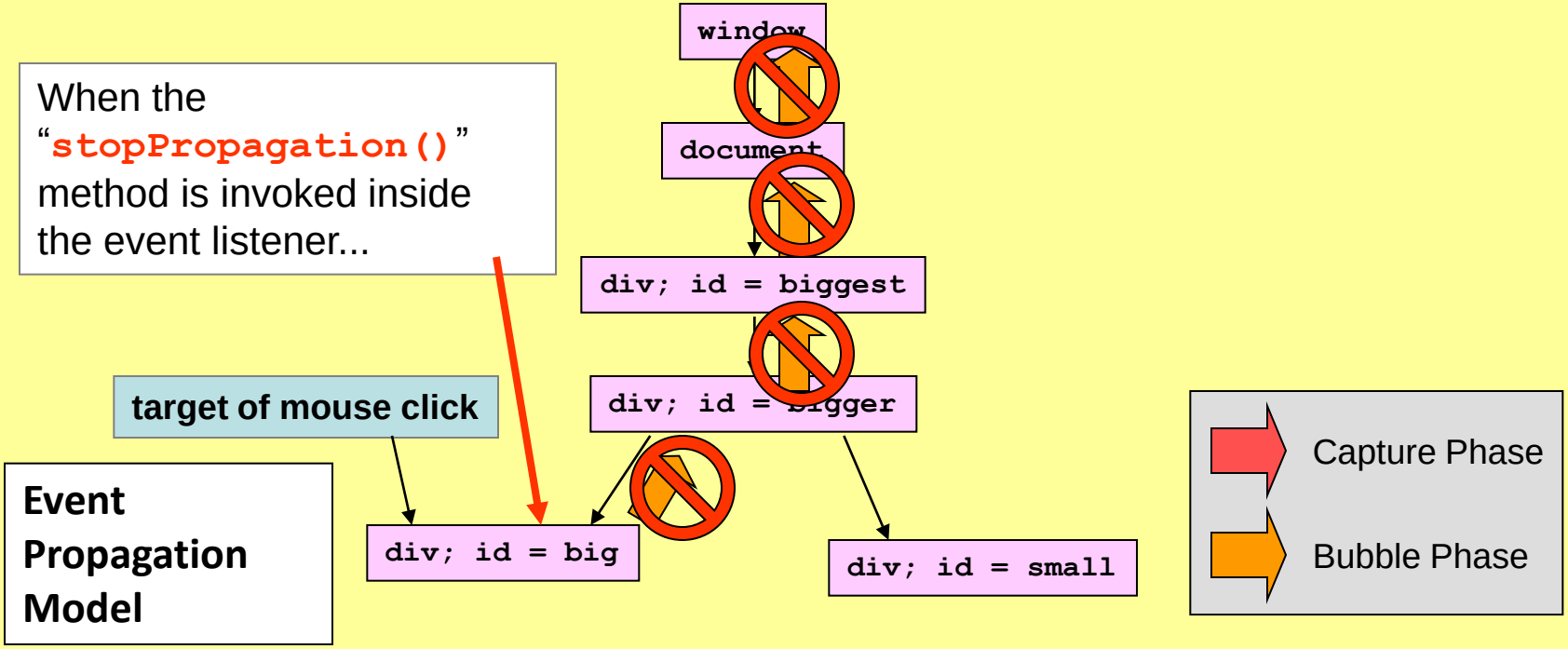
See "event_propagation.html"

Event Propagation…bubble phase

We say that: an object will **handle the event during the bubble phase**.

```
[div].addEventListener("load", init, false);
```

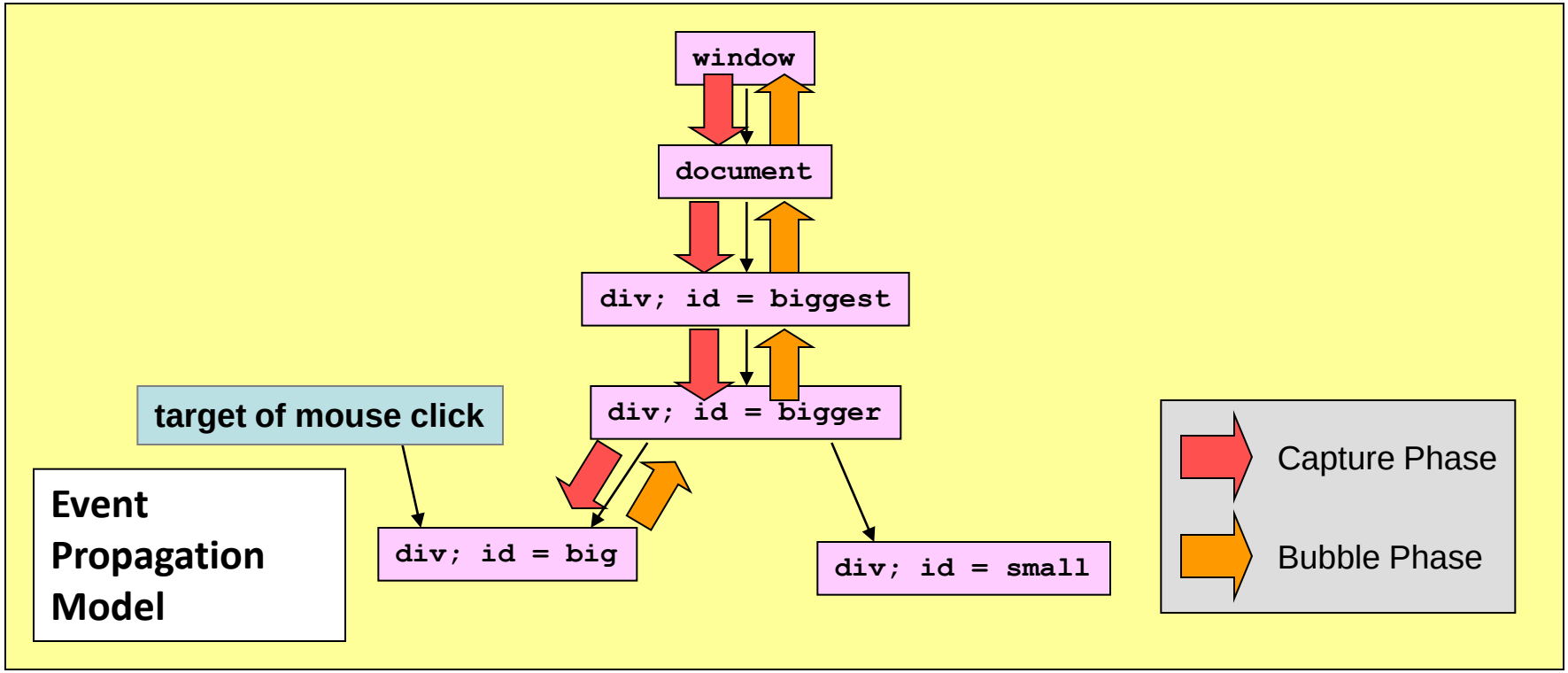
When the "`stopPropagation()`" method is invoked inside the event listener...



See "event_propagation.html"

Event Propagation…

- Remember, **both event propagation flows co-exist**!
  - The **`addEventListener()`** call only defines which direction you're interested in!



See "event_propagation.html"

Event Propagation…

- Some notes:
  - An object can register more than one event listener.
  - An object can register more than one event listener for a particular type of events.
  - An object can remove a particular event listener as follows:

A signature for the event registration.

```
window.addEventListener("click", foobar, true);
......
window.removeEventListener("click", foobar, true);
```

Client-side JavaScript

- XMLHTTPRequest (or XHR)

See http://demo4140-tywong.rhcloud.com/11_js_xhr

What is **XMLHttpRequest**?

- It is an object!
  - It allows you to script HTTP requests!

- Because of conformance…let's be conservative…

```
<script language=javascript>
    try {
        if(new XMLHttpRequest())
            alert("This is a good browser!");
    }
    catch (e) {
        alert("Uncle Bill...you win!");
    }
</script>
```

Exception is caught when the constructor does not exist…

See "test_browser.html"

`XMLHttpRequest` not found?

- The notorious Internet Explorer…

IE 6.0 and before	`new ActiveXObject("Msxml2.XMLHTTP");`
	`new ActiveXObject("Microsoft.XMLHTTP");`
IE 7.0 and after	`new XMLHttpRequest();`

You couldn't find IE 6.0 failing to execute the Gmail AJAX interface because IE 6.0 has another way to create the HTTP scripting object…

So, how to cope with different browsers?

XMLHttpRequest not found?

- To deal with the notorious Internet Explorer…

```
function new_request()  {

    var _factories = [
        function () { return new XMLHttpRequest(); },
        function () { return new ActiveXObject("Msxml2.XMLHTTP"); },
        function () { return new ActiveXObject("Microsoft.XMLHTTP"); }
    ];

    for(var i = 0; i < _factories.length; i++) {
        try {
            var factory = _factories[i];
            var r = factory();
            if(r != null)
                return r;
        }
        catch(e) {
            continue;
        }
    }
    return null;  // a hopeless browser...
}
```

From now on, we'll use this function in our discussion…

A try-them-out loop.

Try-and-catch until you can instantiate an object.

A simple GET HTTP request…

Assume that you're using the `new_request()` function defined previously.

```
function read_url(url) {
    var request = new_request();
    request.open("GET", url, false);
    request.send(null);

    if(request.readyState == 4) {
        if(request.status != 200)
            alert("Error code = " + rec
        else
            return request.responseText
    }
    return null;
}
```

`request.open()`

There are many different combinations for you to open a connection.

- **Argument #1**: the method "GET" or "POST";

- **Argument #2**: the URL. Be aware of the **single-origin policy**.

- **Argument #3**: Asynchronous or not.

A simple GET HTTP request…

In the case of asynchronous HTTP, this value is kept changing. Each change generates a “readystatechange” event.

```
function read_url(url) {
    var request = new_request();
    request.open("GET", url, fal
    request.send(null);

    if(request.readyState == 4)
        if(request.status != 200
            alert("Error code =
        else
            return request.respo
    }
    return null;
}
```

`request.readyState`

0	open() has not been called.
1	open() has been called, but send() has not been called.
2	send() has been called, but the server has not responded yet.
3	Data is being received from the server
4	The server’s response is complete.

A simple GET HTTP request…

```
function read_url(url) {
    var request = new_request();
    request.open("GET", url, false);
    request.send(null);

    if(request.readyState == 4) {
        if(request.status != 200)
            alert("Error code = " + request.status);
        else
            return request.responseText;
    }
    return null;
}
```

Check the HTTP response code, expecting "**200 OK**".

The content of the HTTP response.

See "get_sync.html" & "hello.txt"

How about an asynchronous call?

Asynchronous request

```
function read_url(url) {
    var request = new_request();
    request.open("GET", url, true);
    request.send(null);

    request.onreadystatechange = function () {

    };
}
```

Event Handler

The event is called the "**`readystatechange`**" event.

After the XHR object is instantiated, its "**`readystate`**" field will be changing all the time until it reaches the value 4. Whenever its value changes, a "**`readystatechange`**" event is fired.

How about an asynchronous call?

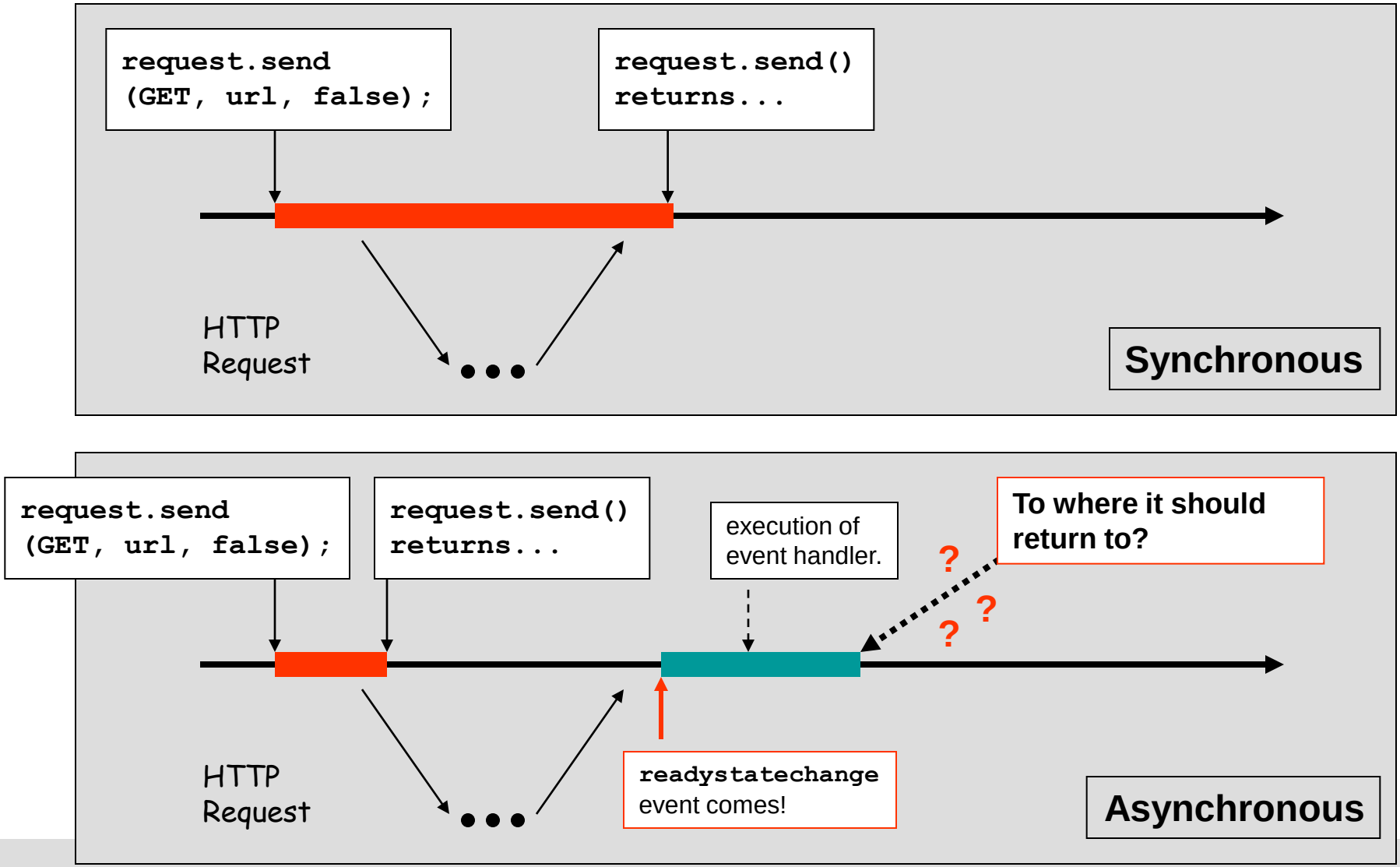
```
function read_url(url) {
    var request = new_request();
    request.open("GET", url, true);
    request.send(null);

    request.onreadystatechange = function () {
        if(request.readyState == 4) {
            if(request.status != 200)
                alert("Error code = " + new String(request.status));
            else
                return request.responseText;
        }
    };
}
```

This function will be called when the "`readyState`" becomes 4.

Why can't we write in this way?

See "get_async_wrong.html"

How about an asynchronous call?



How about an asynchronous call?

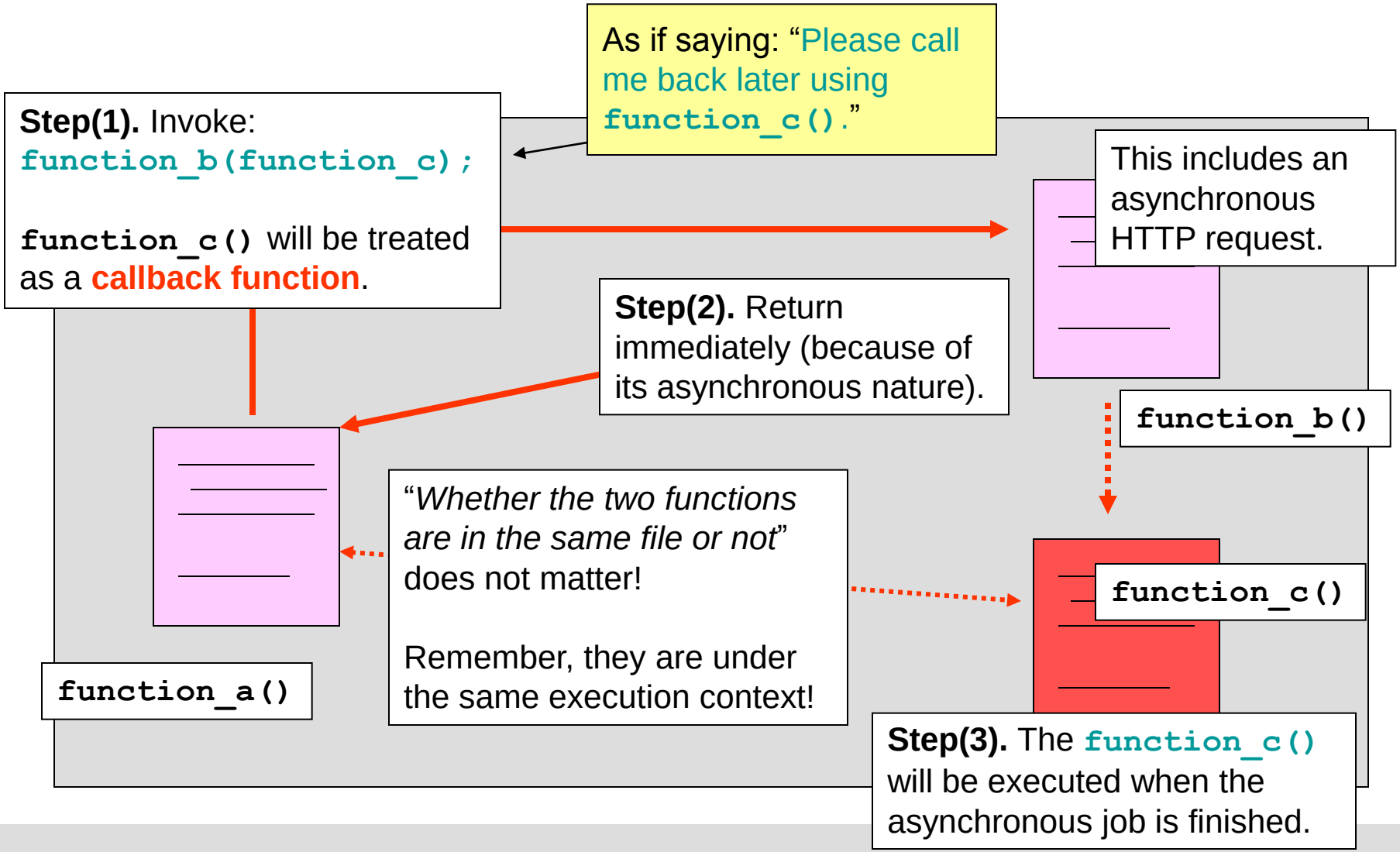
```
function read_url(url) {
    var request = new_request();
    request.open("GET", url, true);
    request.send(null);

    request.onreadystatechange = function () {
        if(request.readyState == 4) {
            if(request.status != 200)
                alert("Error code = " + new String(request.status));
            else {
                var h1 = document.createElement("h1");
                h1.appendChild(
                   document.createTextNode(request.responseText) );
                document.body.appendChild(h1);
            }
        }
    };
}
```

That's why: DOM scripting is our good friend.

As a matter of fact, a more generic approach is to enable **the use of the callback function**.

See "get_async_correct.html"

What is a callback function?



How to use callback function?

```
function read_url(url, callback) {
    var request = new_request();

    request.onreadystatechange = function () {
        if(request.readyState == 4) {
            var result = null;
            if(request.status == 200) {
                result = request.responseText;
                if(callback)
                    callback(result);
            }
        }
    };

    request.open("GET", url, true);
    request.send(null);
}
```

A callback function is expected.

Callback function is invoked here!

See "use_callback.html"

How to use callback function?

```
function read_url(url, callback) {
    var request = new_request();

    request.onreadystatechange = function () {
        if(request.readyState == 4) {
            var result = null;
            if(request.status == 200) {
                result = request.responseText;
                if(callback)
                    callback(result);
            }
}
```

The callback function

```
function report_result(input) {
    document.getElementsByTagName("body")[0].
                innerHTML = "<h1>" + input + "</h1>";
}
```

How about a POST request?

```
function post_login(url) {
    var request = new_request();
    request.open("POST", url, false);
    request.setRequestHeader("Content-type",
                "application/x-www-form-urlencoded");

    var input = "login=tywong&passwd=sosad";
    request.send(input);

    if(request.readyState == 4) {
        if(request.status != 200) {
            alert("Error code = " + new String(request.status));
            return null;
        }
        else
            return request.responseText;
    }
}
```

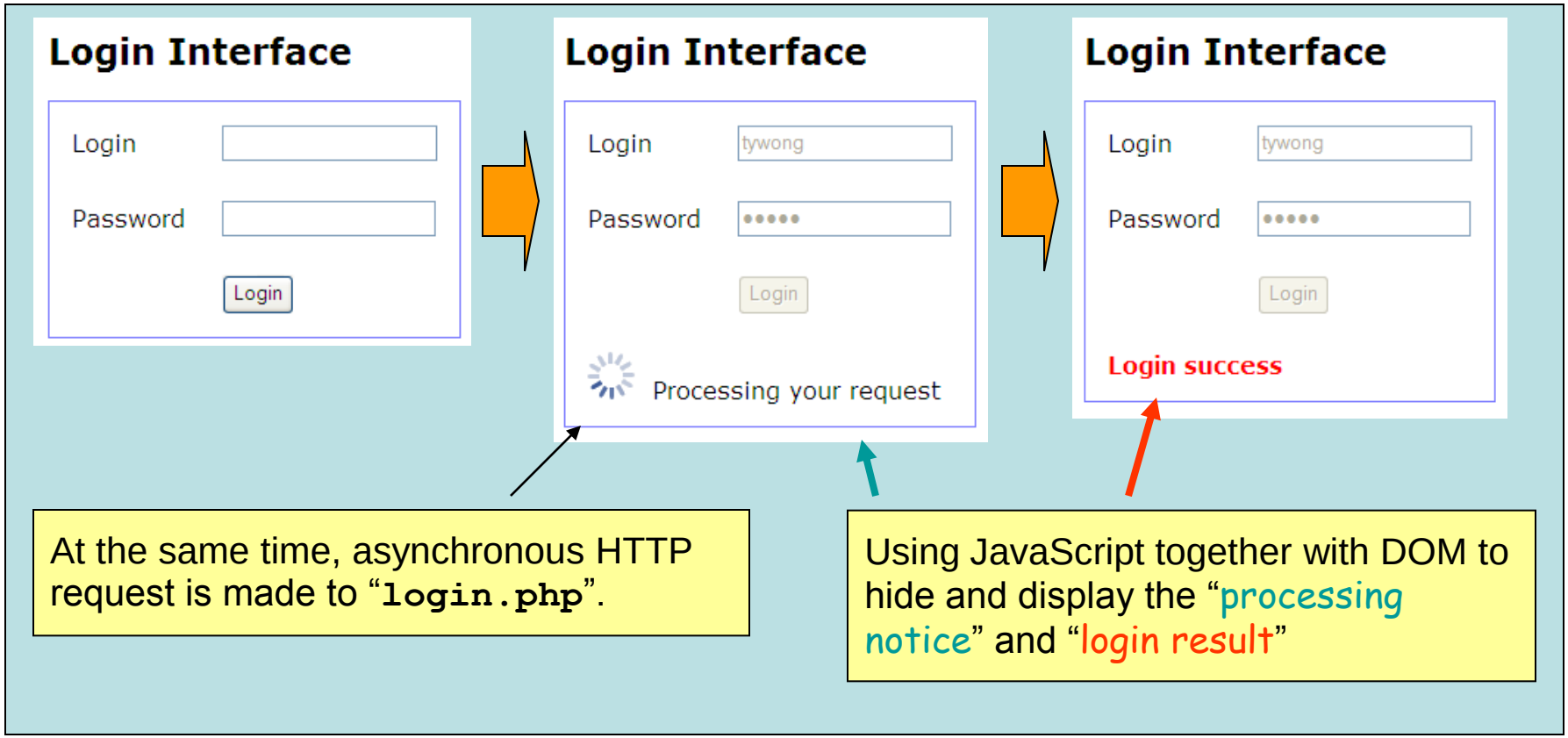
This is required in order to **pretend to be a form submission**.

Data is constructed in JavaScript.

Note that if the data requires to be encoded, use the method `encodeURIComponent()`.

See "post_sync.html" & "login.php"

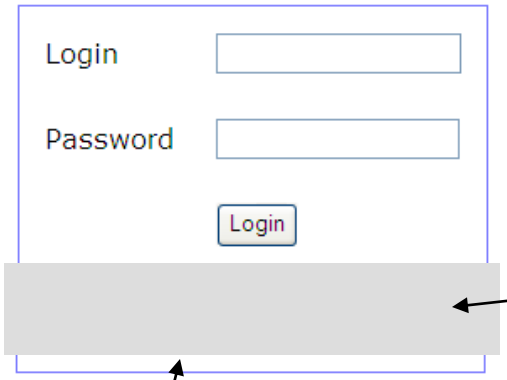
Recalling our previous example



Our previous example…

This table row is hidden by default, using CSS definition.

Login Interface

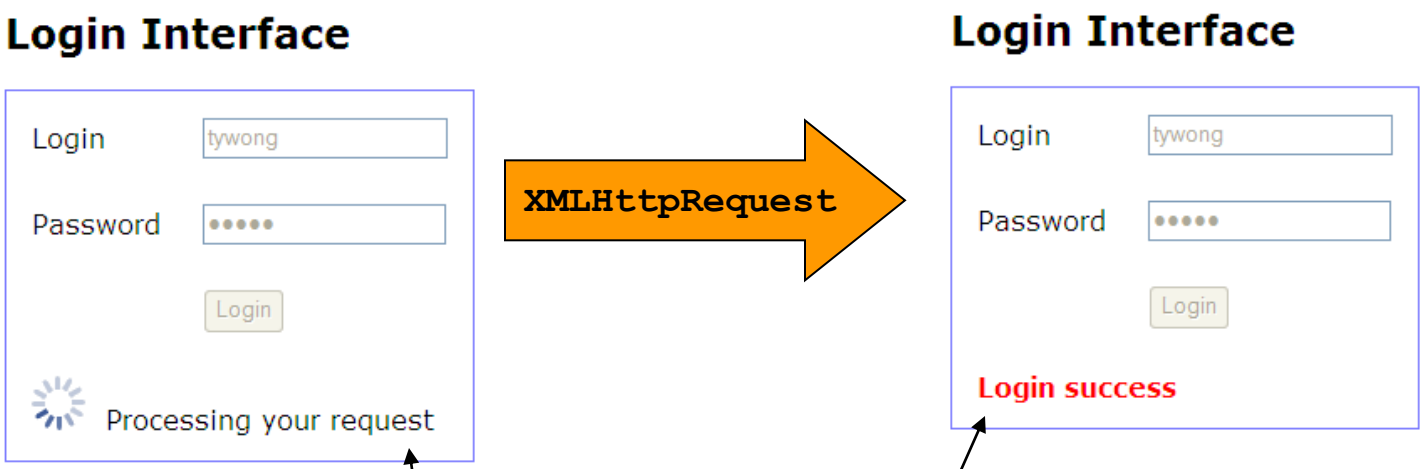



```
<tr id=result_tr
    style="display:none;">
    <td id=result_td colspan=2>
        <img src=loading.gif>
        Processing your request
    </td>
</tr>
```

```
.......
document.getElementById("result_tr").style.display = "";
......
```

Displaying the hidden component using JavaScript.

Our previous example…




```
.......
document.getElementById("result_td").innerHTML = "Login success";
......
```

Changing the table row using DOM scripting.

Important...

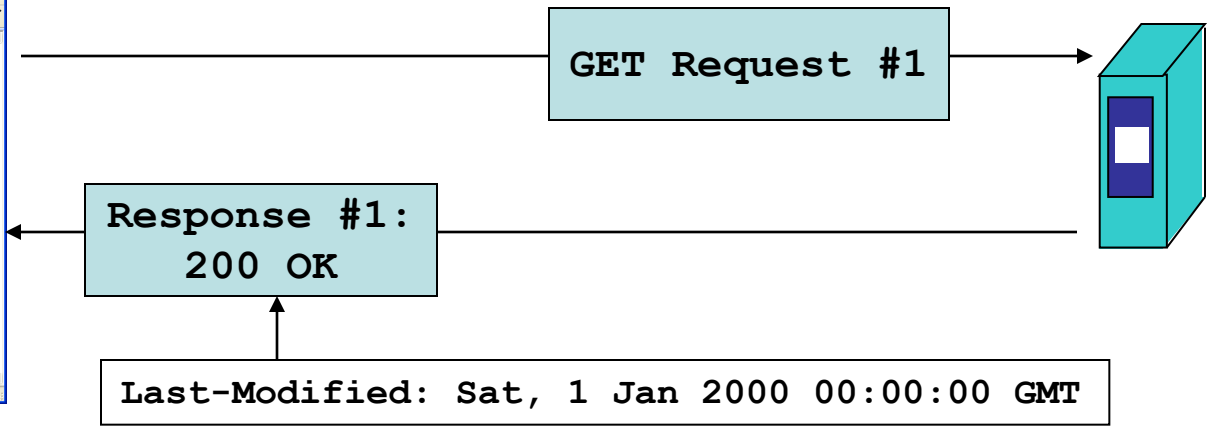
- Loading the HTML file is usually “200 OK”...
  - Yet...XHR gives you “304 Not Modified”!

No.	Time	Source	Destination	Protocol	Info
12	0.255802	137.189.90.240	192.168.1.162	TCP	http > 51368 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=
13	0.255890	192.168.1.162	137.189.90.240	TCP	51368 > http [ACK] Seq=1 Ack=1 Win=65700 Len=0
14	0.258881	192.168.1.162	137.189.90.240	HTTP	GET /~csci4140/cgi-bin/js_xhr/script/get_async_correct.l
15	0.557256	192.168.1.162	137.189.90.240	HTTP	[TCP Retransmission] GET /~csci4140/cgi-bin/js_xhr/scri
16	0.565521	137.189.90.240	192.168.1.162	TCP	http > 51368 [ACK] Seq=1 Ack=478 Win=6912 Len=0
17	0.569133	137.189.90.240	192.168.1.162	HTTP	HTTP/1.1 200 OK (text/html)
18	0.591852	192.168.1.162	137.189.90.240	HTTP	GET /~csci4140/cgi-bin/js_xhr/script/hello.txt HTTP/1.1
19	0.601713	137.189.90.240	192.168.1.162	HTTP	HTTP/1.1 304 Not Modified
20	0.799244	192.168.1.162	137.189.90.240	TCP	51368 > http [ACK] Seq=1069 Ack=953 Win=64748 Len=0
21	2.416219	192.168.1.162	137.189.90.240	HTTP	ript/get_async_correct.l
22	2.427600	137.189.90.240	192.168.1.162	HTTP	
23	2.453831	192.168.1.162	137.189.90.240	HTTP	GET /~csci4140/cgi-bin/js_xhr/script/hello.txt HTTP/1.1
24	2.465280	137.189.90.240	192.168.1.162	HTTP	HTTP/1.1 200 OK (text/plain)
25	2.667342	192.168.1.162	137.189.90.240	TCP	51368 > http [ACK] Seq=2079 Ack=2078 Win=65332 Len=0
48	17.467163	137.189.90.240	192.168.1.162	TCP	http > 51368 [FIN, ACK] Seq=2078 Ack=2079 Win=10496 Len

Why not 200 OK?!

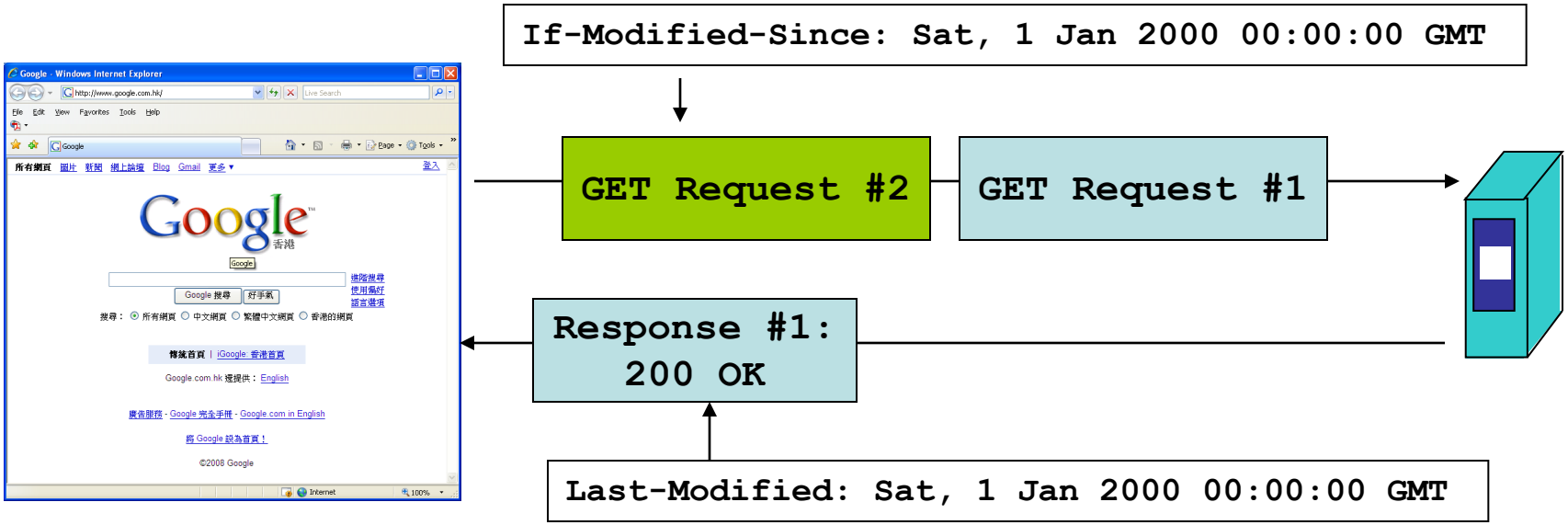
See “get_sync.html” again with firebug / WireShark / Developer Tool

HTTP Caching



If a response carries the "**`Last-Modified`**" header, then it implies that the last modified time of that object is ...

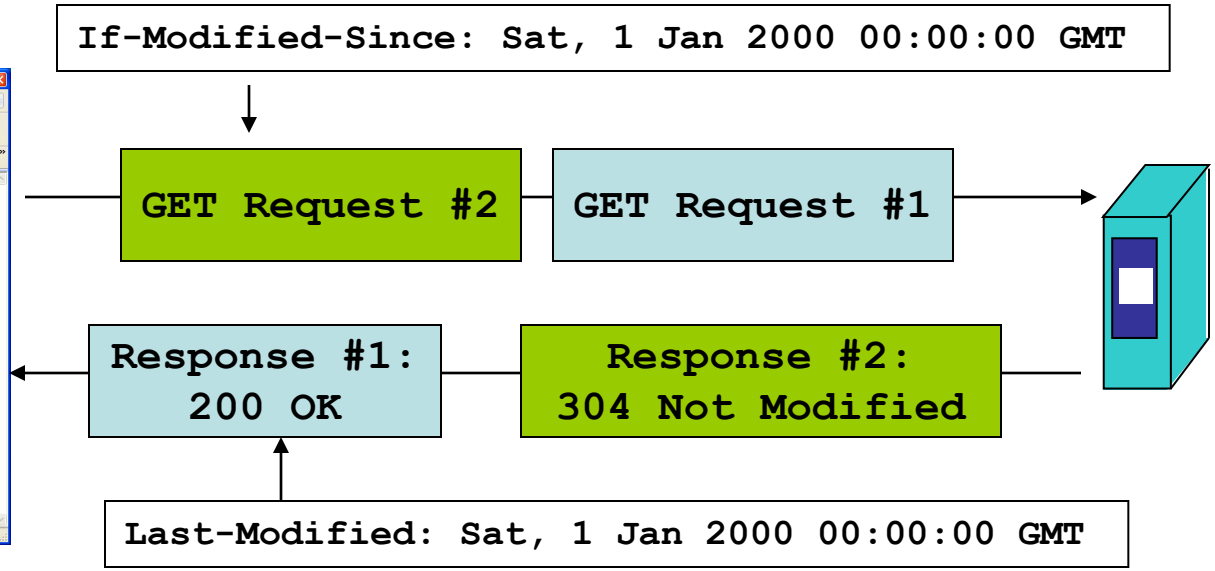
HTTP Caching



For further requests on the same object, "**`If-Modified-Since`**" header will be added to the requests.

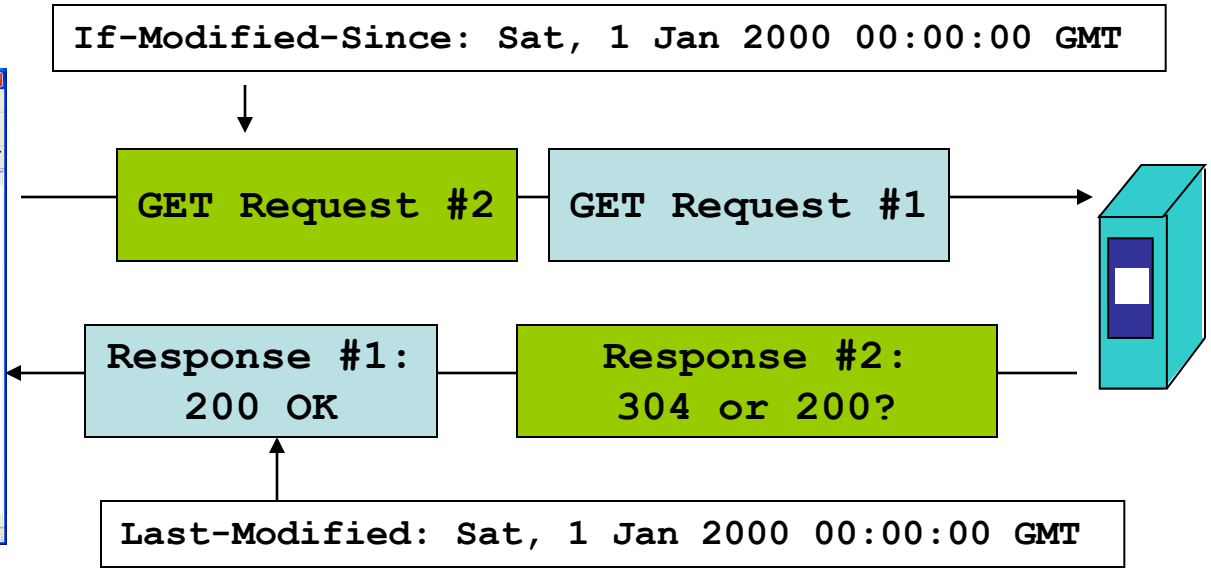
Then, the server will compare the value of the "**`If-Modified-Since`**" header and the modified time of the actual object.

HTTP Caching



If the modified time of the object is not the later one, then "**304 Not Modified**" will be sent! Note that **the actual object will not be sent** in this case.

HTTP Caching



The object will be sent only when:
- the object is updated;
- Shift + Reload;

So, the question is: how to force "**200 OK**" instead of "**304 Not Modified**"?

Defeating HTTP Caching

```
function read_url(url) {
    var request = new_request();
    request.open("GET", url, false);
    request.setRequestHeader
     ("If-Modified-Since", (new Date(0)).toGMTString());
    request.send(null);

    if(request.readyState == 4) {
        if(request.status != 200)
            alert("Error code = " + request.status);
        else
            return request.responseText;
    }
    return null;
}
```

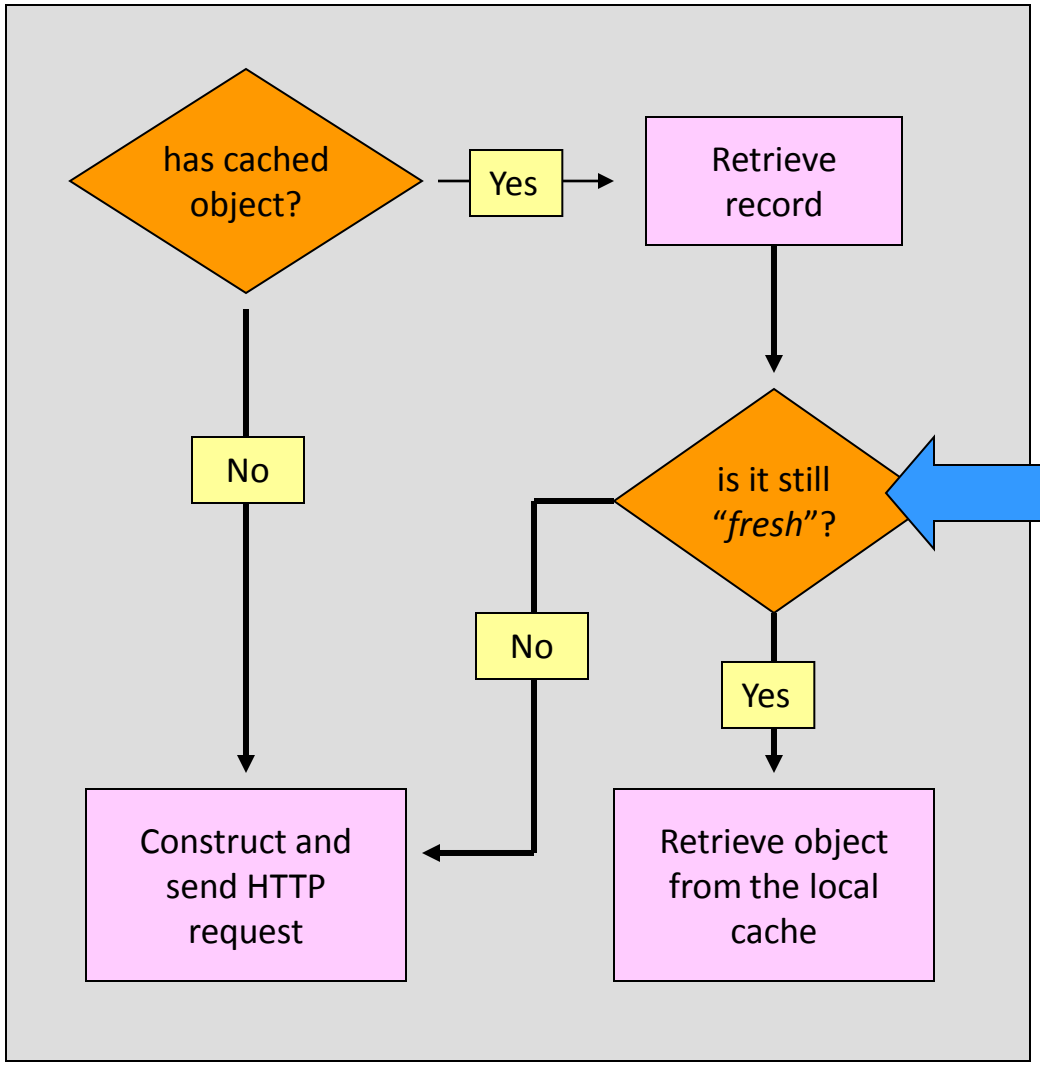
What is it?

This is: Thu, 1 Jan 1970, 00:00:00 GMT

Reference: http://blog.httpwatch.com/2009/08/07/ajax-caching-two-important-facts/

See "no_cache.html"

HTTP Caching – client decision




```
if ( "Expires" is set &&
 not expired ) {
    return "Yes";
}
else {
  if (Cache-Control is set) {
      if( max-age is reached )
          return "No";
      else
          return "Yes";
  }
  else
    return "No";
}
```

HTTP Caching – store or not

- Some important "**`Cache-Control`**" keys…

Key	Description
`max-age=3600`	- This is a very common key. This key implies that *the object should be stored in the cache*. - The key value states the time that the cached object is considered to be fresh. - The key value is expressed in terms of seconds.
`must-revalidate`	- It is usually used together with the "**`max-age`**" key. - It forbids the caching system to bypass the cache control mechanism.
`no-cache`	Don't cache the object (but it is not working in Firefox).
`no-store`	Don't store the cached object (but, it may be cached in memory, e.g., Firefox).

Server-side: HTTP caching control

- It is not hard to enable and to disable caching...

```
/* in PHP, to cache the thing "forever" */

header("Expires: Thu, 1 Jan 2036 00:00:00 GMT");
```

```
/* in PHP, never cache it */

header("Cache-Control: no-cache, no-store");
```

- In a nutshell, controlling caching is necessary together after logging in.

Side-track: how about jQuery?

- I know jQuery, how about the "cache" property?

```
/* in jQuery */

$.ajax({
    type:    "GET",
    url:     "http://www.cse.cuhk.edu.hk",
    cache:   false
 });
```

What does "cache" mean then?

- It means: appending "`?_=[random string]`" to URL so that:
  - The request is **pointing to another place**!
  - No cached object would then be found!

See "jquery.html"

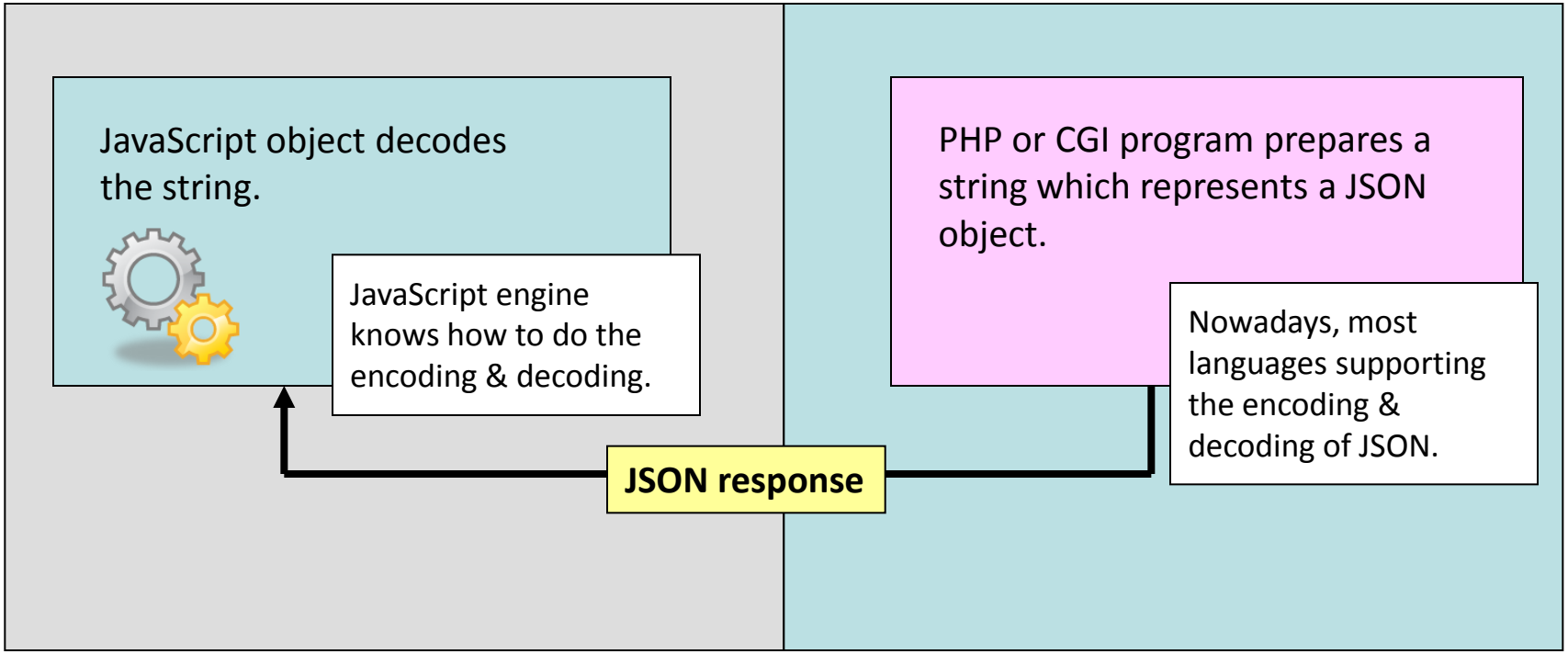
Make-Your-Life-Easier JavaScript

- JSON and simple web service.

See http://demo4140-tywong.rhcloud.com/12_json

Who is JSON?

- JSON – JavaScript Object Notation.



Who is JSON?

http://www.json.org/

```
var array = ["hello", "world"];
```

Defining an array...

```
var object = {"login":"tywong", "passwd":"sosad"};
```

Defining an object...

JSON is to enable:

- converting the objects into strings, and then allow you to write the strings to files (backup) or to send the strings to remote servers (upload).

- converting a string, in JSON format, into an array or an object so as to support **dynamic array and object declarations**!

JSON in JavaScript

- As a matter of fact, JavaScript supports a **code injection mechanism**.
  - Using the method `eval()`.

```
eval( "alert(\"hello world\");" );
```

The code inside the string will be compiled and be running immediately.

```
try {
    eval( "alert(hello world);" );
}
catch(e){
    alert(e);
}
```

When the supplied code cannot be compiled, an exception will be thrown.

See "hello_world.html"

JSON in JavaScript

- Some people love to use **`eval()`** together with **`unescape()`** to *discourage* others to look into their codes…

```
var str =
    unescape(
        "%61%6c%65%72%74%28%22hello%20world%22%29" );
    );
eval(str);
```

See "try.html"

JSON in JavaScript

- What is about JSON is the return value of **`eval()`** and the input string...

```
try {
    var array = eval( "[1,2,3,4]" );
    alert(array);

    var object = eval
        ("{\"login\":\"tywong\", \"passwd\":\"sosad\"}");
    alert(object);
}
catch(e){
    alert(e);
}
```

See “eval.html”

JSON in JavaScript

- But, using `eval()` makes your program becoming vulnerable to code injection attack, so...

```
try {
    var array = JSON.parse( "[1,2,3,4]" );
    alert(array);
}
catch(e){
    alert(e);
}
```

See "jsonParse.html"

JSON in JavaScript

- Also, `JSON.parse()` provides a great deal of flexibility....

```
try {
    var array = JSON.parse(
        "{\"login\":\"tywong\", \"passwd\":\"sosad\"}",
         filter );
    alert(array);
}
catch(e){
    alert(e);
}

function filter(key, value) {
    if(key == "login" && value == "tywong")
        return "";   // tywong is rejected!
    else
        return value;
}
```

See "jsonFilter.html"

JSON in JavaScript

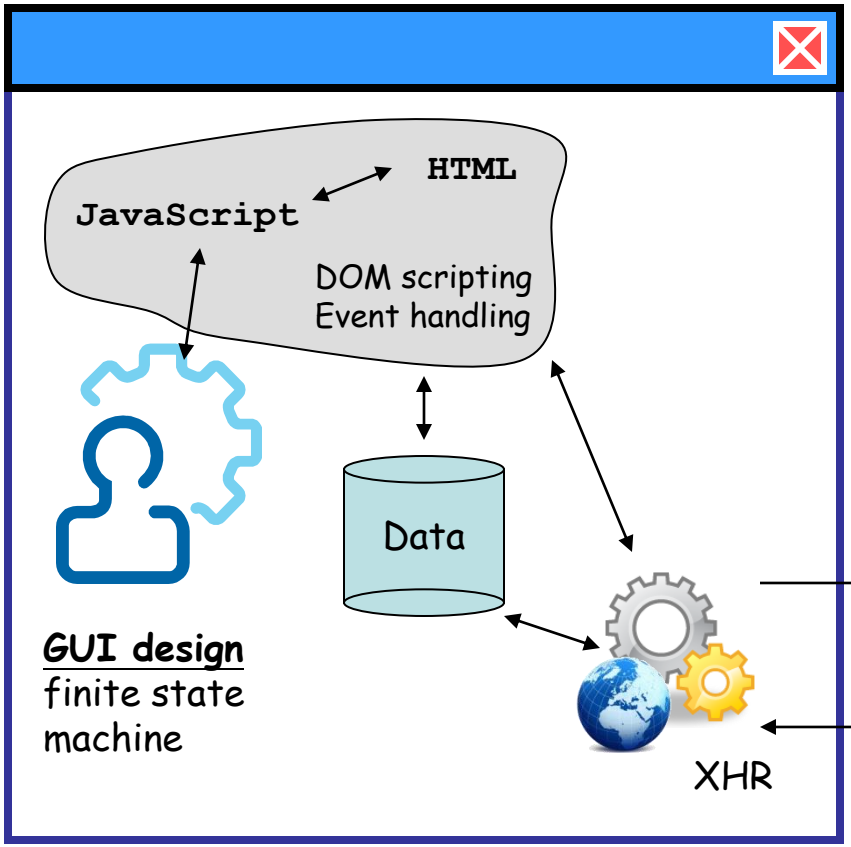
- **`JSON.parse()`** is reversing **`JSON.stringify()`**
  - means to convert an object or an array into a string!

```
var object = new Object;
object.login = "tywong";
object.passwd = "sosad";

var string = JSON.stringify(object);

alert(string);
```

If you're going to set this string out as a part of the CGI, **`encodeURIComponent()`** can help you a lot!

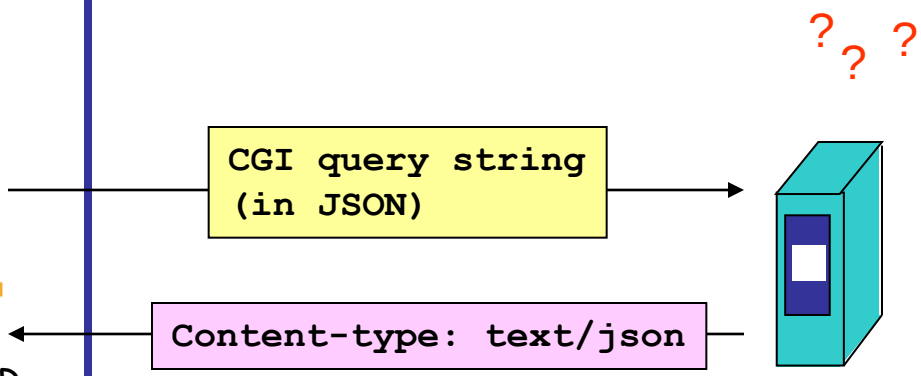
See "jsonStringify.html"

Back to the big picture...



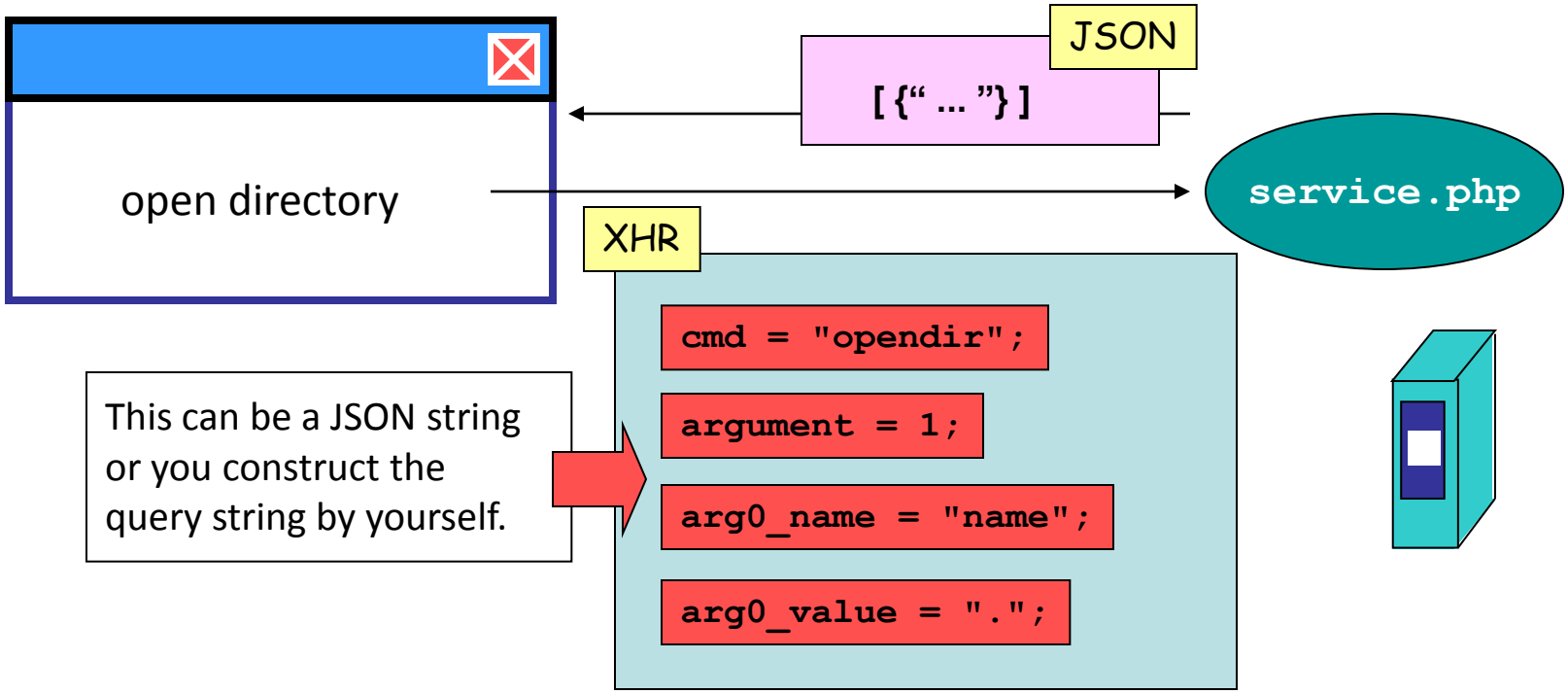
We now know that the client can handle JSON well enough.

But, how about the server?



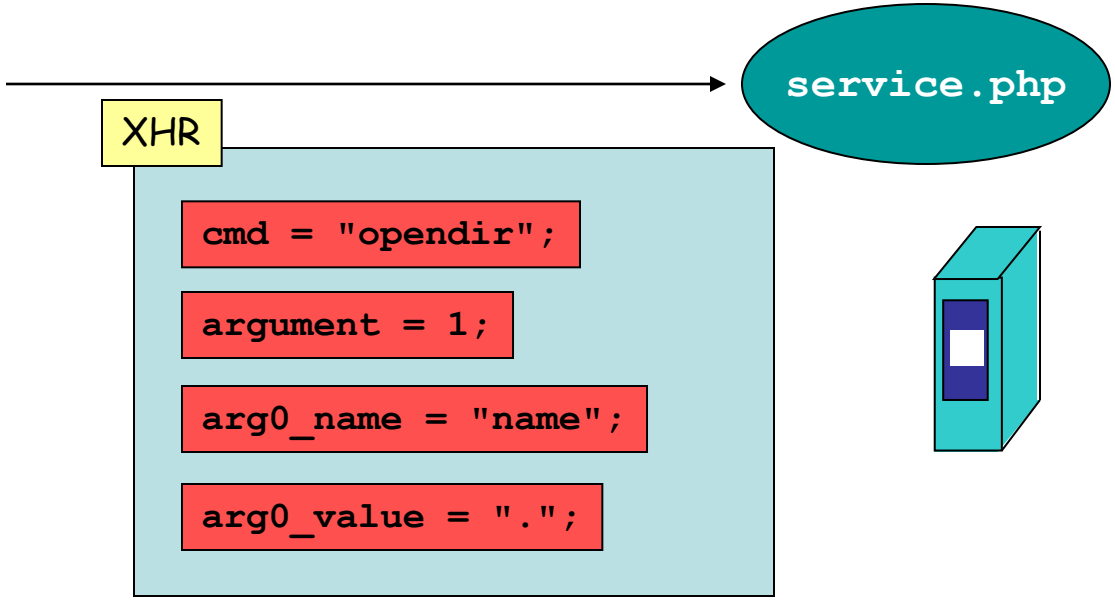
A web service design...

- A web service is something similar to the remote procedure call (RPC)...



A web service design...

- A web service is something similar to the remote procedure call (RPC)...
  - This design is better than to have different programs and each serves one purpose.

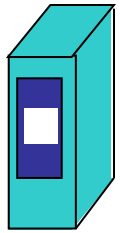


A web service design...using JSON

- In PHP, two functions support JSON:
  - `json_encode()`: to convert an associative array to string;
  - `json_decode()`: to convert a string to an object.

```
<?
    header("Content-type: text/plain");
    $str =
      "{\"login\":\"tywong\",\"passwd\":\"sosad\"}";
    $obj = json_decode($str);
    print_r($obj);

    $str = json_encode($obj)
    echo $str;
?>
```

service.php



See “json_encode.php”, “json_decode.php”, “ls.php”, and “ls.html”

Summary

- Using JSON is great because...
  - You don’t have to worry about how to convert data from one end to another
  - since both ends can use objects as the common medium!
- JSON is everywhere...
  - You find it in Firefox, Chrome, Facebook, Gmail, etc...
- Remember, JSON and web service is just a design practice, not a standard.

Classroom activity

- Facebook XHR & bookmark export of Firefox!