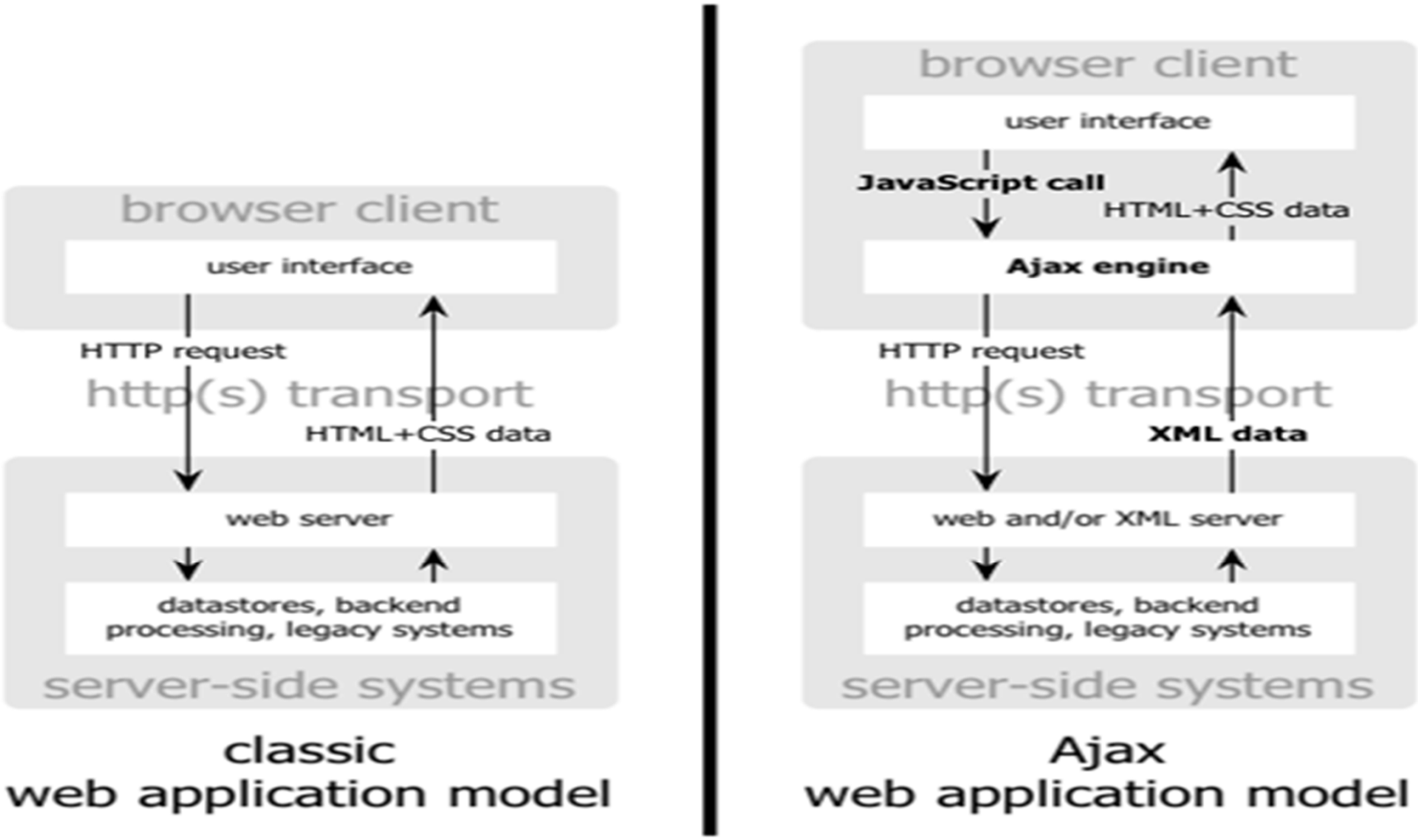
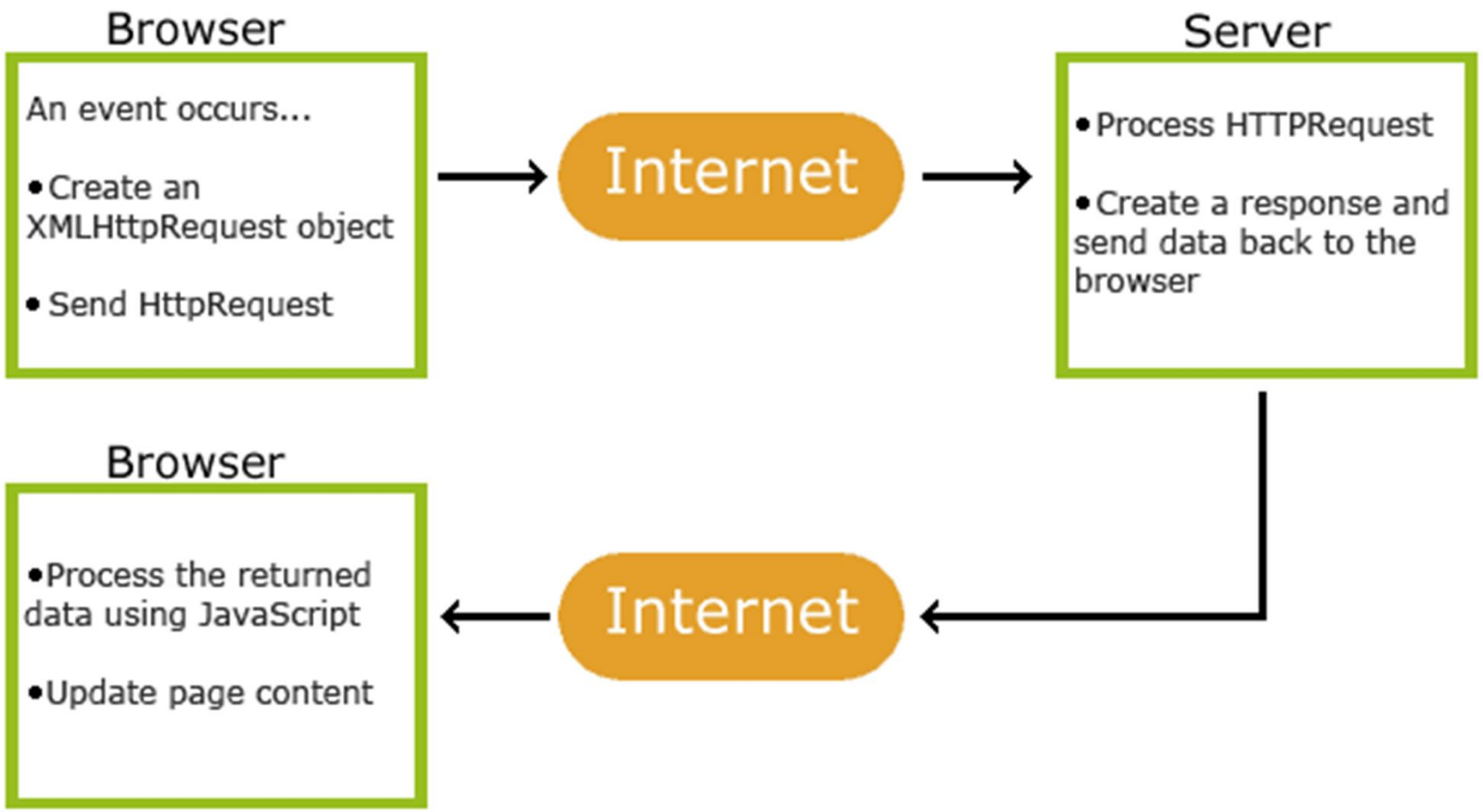
Introduction to AJAX

- AJAX = Asynchronous JavaScript and XML.
- AJAX is a technique for creating fast and dynamic web pages.
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
- Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.
- Examples of applications using AJAX: Google Maps, Google Suggest, Gmail, Youtube, Flickr, and Facebook tabs.
- **AJAX meant to increase the web page's interactivity, speed, and usability.**

Comparison


browser client
user interface
HTTP request
http(s) transport
HTML+CSS data
web server
datastores, backend
processing, legacy systems
server-side systems
classic
web application model
browser client
user interface
JavaScript call
HTML+CSS data
Ajax engine
HTTP request
http(s) transport
XML data
web and/or XML server
datastores, backend
processing, legacy systems
server-side systems
Ajax
web application model

How AJAX works?



AJAX is based on Internet Standards

1. XMLHttpRequest object (to exchange data asynchronously with a server)
2. JavaScript/DOM (to display/interact with the information)
3. CSS (to style the data)
4. XML (often used as the format for transferring data)

Note: AJAX applications are browser- and platform-independent!

AJAX can be used for interactive communication with a database and an XML file .

Sample Code

```
<!DOCTYPE html>
<html>  <body>
<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button type="button" onclick="loadXMLDoc()">Change
Content</button>
</body></html>
```

```
<head>
<script>
function loadXMLDoc()
{
.... AJAX script goes here ...
}
</script>
</head>
```

C:/wamp/www/first
AJAX.html

```
<html> <head> <script>
function loadXMLDoc(){
var xmlhttp;
if (window.XMLHttpRequest)
xmlhttp=new XMLHttpRequest();
xmlhttp.onreadystatechange=function()
{
if (xmlhttp.readyState==4 && xmlhttp.status==200)
document.getElementById("myDiv").innerHTML=xmlhttp.response
Text;   }
xmlhttp.open("GET","AJAX_TextFile.txt",true);
xmlhttp.send(); } </script> </head> <body>
```

```
<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button type="button" onclick="loadXMLDoc()">Change
Content</button>
</body> </html>
```

C:/wamp/www/AJAXTextFile.txt

AJAX is not a new programming language.

AJAX is a technique for creating fast and dynamic web pages.

The XMLHttpRequest Object

The XMLHttpRequest object is used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

```
var xmlhttp;
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
```

Send a Request To a Server

xmlhttp.open("GET","ajax_info.txt",true); xmlhttp.send();

xmlhttp.open("GET","demo.asp?t=" + Math.random(),true);

xmlhttp.open("GET","demo.asp?fname=Henry&lname=Ford",true);

The file can be any kind of file, like .txt and .xml, or server scripting files like .asp and .php (which can perform actions on the server before sending the response back).

Method	Description
open(*method,url,async*)	Specifies the type of request, the URL, and if the request should be handled asynchronously or not. ***method***: the type of request: GET or POST ***url***: the location of the file on the server ***async***: true (asynchronous) or false (synchronous)
send(*string*)	Sends the request off to the server. ***string***: Only used for POST requests

Asynchronous - True or False?

- Sending asynchronous requests is a huge improvement for web developers. Many of the tasks performed on the server are very time consuming. Before AJAX, this operation could cause the application to hang or stop.
- With AJAX, the JavaScript does not have to wait for the server response, but can instead:
  - execute other scripts while waiting for server response
  - deal with the response when the response ready

```
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("myDiv").innerHTML=xmlhttp.res
ponseText;
    }
  }
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
```

Async=false

- Using async=false is not recommended, but for a few small requests this can be ok.
- Remember that the JavaScript will NOT continue to execute, until the server response is ready. If the server is busy or slow, the application will hang or stop.

- **Note:** When you use async=false, do NOT write an onreadystatechange function - just put the code after the send() statement:
- xmlhttp.open("GET","ajax_info.txt",false);
  xmlhttp.send();
  document.getElementById("myDiv").innerHTML=xmlhttp.respo
  nseText;

AJAX - Server Response

Property	Description
responseText	get the response data as a string
responseXML	get the response data as XML data

If the response from the server is not XML, use the responseText property.

```
document.getElementById("myDiv").innerHTML
=xmlhttp.responseText;
```

- If the response from the server is XML, and you want to parse it as an XML object, use the responseXML property:
- Request the file cd_catalog.xml and parse the response:

```
xmlDoc=xmlhttp.responseXML;
txt=“ “;
x=xmlDoc.getElementsByTagName("ARTIST");
for (i=0;i<x.length;i++)
  {
  txt=txt + x[i].childNodes[0].nodeValue + "<br>";
  }
document.getElementById("myDiv").innerHTML=txt;
```

The onreadystatechange event

- When a request to a server is sent, we want to perform some actions based on the response.
- The onreadystatechange event is triggered every time the readyState changes.
- The readyState property holds the status of the XMLHttpRequest.
- Three important properties of the XMLHttpRequest object:

```
xmlhttp.onreadystatechange=function()  {
 if (xmlhttp.readyState==4 && xmlhttp.status==200)    {
  document.getElementById("myDiv").innerHTML=
                xmlhttp.responseText;    }  }
```

Note: The onreadystatechange event is triggered five times (0-4), one time for each change in readyState.

Property	Description
onreadystatechange	Stores a function (or the name of a function) to be called automatically each time the readyState property changes
readyState	Holds the status of the XMLHttpRequest. Changes from 0 to 4: 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 404: Page not found

AJAX with JSON

```
<html><head><script type = "text/javascript">
      function loadJSON(){
      var data_file = "http://www.tutorialspoint.com/json/data.json";
         /* var data_file = "data.json";*/
        var http_request = new XMLHttpRequest();
        try{              // Opera 8.0+, Firefox, Chrome, Safari
          http_request = new XMLHttpRequest();
        }catch (e){              // Internet Explorer Browsers
 try{                 http_request = new ActiveXObject("Msxml2.XMLHTTP");
          }catch (e) {
            try{
              http_request = new ActiveXObject("Microsoft.XMLHTTP");
            }catch (e){                  // Something went wrong
              alert("Your browser broke!");                  return false;
}}}
```

```
http_request.onreadystatechange = function(){
                if (http_request.readyState == 4  ){
// Javascript function JSON.parse to parse JSON data
                var jsonObj = JSON.parse(http_request.responseText);
// jsonObj variable can be accessed
document.getElementById("Name").innerHTML =
jsonObj.name;
document.getElementById("Country").innerHTML =
jsonObj.country;
                }}
          http_request.open("GET", data_file, true);
          http_request.send(); }     </script>   </head>
```

```
<body>
<h1>Cricketer Details</h1>
<table class = "src">
 <tr><th>Name</th><th>Country</th></tr>
 <tr><td><div id = "Name">Sachin</div></td>
 <td><div id = "Country">India</div></td></tr></table>
<div class = "central">
        <button type = "button" onclick = "loadJSON()">Update
Details </button>      </div>   </body></html>
```


file:///F:/Academic/Web%20Technologies/ITE1002/Programs/AJAX/second.html
Cricketer Details
Name Country
Sachin India
Update Details


file:///F:/Academic/Web%20Technologies/ITE1002/Programs/AJAX/second.html
Cricketer Details
Name Country
brett Australia
Update Details

Exercise

- The following example will demonstrate how a web page can communicate with a web server while a user type characters in an input field:

Start typing a name in the input field below:

First name: a

Suggestions: Anna , Amanda

Start typing a name in the input field below:

First name: m

Suggestions: no suggestion

Exercise

Please enter the first letter of the email:

c Search

Email	Password	Birthday
css100@uregina.ca	temp1234	2018-10-12
cs100@uregina.ca	12345678	2018-02-27
cs110@uregina.ca	12345678	2018-02-28
cs215@uregina.ca	1qa2ws3e	1992-12-29
cs111@uregina.ca	1qaz2wsx	1212-12-12
cs1111@uregina.ca	1qaz2wsx	1212-12-12

- **Persistent (or Keep-alive) Connections**
- In HTTP/1.0, the server closes the TCP connection after delivering the response by default (Connection: Close). That is, each TCP connection services only one request. This is not efficiency as many HTML pages contain hyperlinks (via <a href="url"> tag) to other resources (such as images, scripts – either locally or from a remote server). If you download a page containing 5 inline images, the browser has to establish TCP connection 6 times to the same server.
- The client can negotiate with the server and ask the server not to close the connection after delivering the response, so that another request can be sent through the same connection. This is known as persistent connection (or keep-alive connection). Persistent connections greatly enhance the efficiency of the network. For HTTP/1.0, the default connection is non-persistent. To ask for persistent connection, the client must include a request header "Connection: Keep-alive" in the request message to negotiate with the server.
- For HTTP/1.1, the default connection is persistent. The client do not have to sent the "Connection: Keep-alive" header. Instead, the client may wish to send the header "Connection: Close" to ask the server to close the connection after delivering the response.
- Persistent connection is extremely useful for web pages with many small inline images and other associated data, as all these can be downloaded using the same connection. The benefits for persistent connection are:
- CPU time and resource saving in opening and closing TCP connection in client, proxy, gateways, and the origin server.
- Request can be "pipelined". That is, a client can make several requests without waiting for each response, so as to use the network more efficiently.
- Faster response as no time needed to perform TCP's connection opening handshaking.
- In Apache HTTP server, several configuration directives are related to the persistent connections:
- The KeepAlive directive decides whether to support persistent connections. This takes value of either On or Off.
- KeepAlive On|OffThe MaxKeepAliveRequests directive sets the maximum number of requests that can be sent through a persistent connection. You can set to 0 to allow unlimited number of requests. It is recommended to set to a high number for better performance and network efficiency.
- MaxKeepAliveRequests 200The KeepAliveTimeOut directive set the time out in seconds for a persistent connection to wait for the next request.
- KeepAliveTimeout 10