Synchronous Communication

- User and server take turns waiting
  - User requests pages while browsing
    - Waits for server to respond
    - Waits for the page to load in the browser
  - Server locates/creates pages and transmits them
    - Waits for requests from users

- Dynamic content is strictly separated
  - JavaScript on the client-side
  - PHP on the server-side
Asynchronous Approach

- User generates many requests while browsing
  - Some are for complete pages
  - Some are for small amounts of information
    - Done via AJAX
- Server answers the requests
  - Transmits pages and/or individual pieces of data
- User need not always wait for a complete page to load
  - Portions are updated dynamically with server data
    - AJAX makes the requests
    - JavaScript updates the page once the answer is received
  - Several requests can be pending simultaneously
    - Each one is handled individually as answers are received

Definitions and History

- AJAX: Asynchronous JavaScript and XML
- Created at Microsoft
  - Began as XMLHttpRequest in IE5 as an ActiveX control
- W3C standardized the XMLHttpRequest object
  - Adopted fairly quickly by all non-IE browsers
  - Adopted by IE7 and beyond
    - IE6 uses XMLHTTP via ActiveX
- First referred to as AJAX around 2005
Why is AJAX Useful?

- Form data management
  - Validate it against the server's data
  - Complete values being entered by the user
- Content
  - Load a page incrementally
  - Frequently update information
  - Use outside material
- And more ...

Using AJAX

```javascript
var ajax = new XMLHttpRequest();
```

- XMLHttpRequest is the core object
  - Request data with its open and send functions

- `open(method, URL, isAsync)`
  - method mimics HTML's method attribute
    - Values can be "GET" or "POST"
  - URL is the URL of the resource to fetch (as a string)
  - isAsync specifies the type of request
    - true for asynchronous, false for synchronous
Using AJAX

```javascript
var ajax = new XMLHttpRequest();
```

- **send(data)**
  - Sends the request to the server
  - Always called after `open(…)`
  - The value of `data` depends on the method used
    - `null` for "GET"
    - A possibly empty string for "POST"

- **abort()**
  - Cancels a request made to the server

---

Synchronous Text Request

```javascript
window.onload = loadText;

function loadText() {
  var ajax = new XMLHttpRequest();
  ajax.open("POST", "...", false);
  ajax.send(null);
  document.getElementById("...").innerHTML = ajax.responseText;
}
```

- Waits for the request to be fulfilled
  - Commands are executed sequentially and synchronously

This property contains the result of the request
### XMLHttpRequest Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>readyState</td>
<td>The status of the request: 0 (uninitialized), 1 (unsent), 2 (sent), 3 (in progress) or 4 (finished)</td>
</tr>
<tr>
<td>responseText</td>
<td>A string containing the returned text</td>
</tr>
<tr>
<td>responseXML</td>
<td>A document containing the returned XML data</td>
</tr>
<tr>
<td>status</td>
<td>The HTTP status code returned by the web server: 404 (not found) or 200 (OK)</td>
</tr>
<tr>
<td>statusText</td>
<td>The HTTP status message returned by the web server</td>
</tr>
</tbody>
</table>

---

### Asynchronous Requests

- **Use the onreadystatechange event**
  - Called when an XMLHttpRequest's ready state is changed
  - Specify a function/handler to call when the state changes
    - Known as a *callback*

- **Issue: the object's state will change several times**
  - Ensure that `readyState == 4` for a completed request

- **Issue: need the XMLHttpRequest object during callback**
  - Assign an *anonymous nested function* as the handler
  - This function can access variables in its containing function
    - Including the XMLHttpRequest object
XML

- Extensible Markup Language
- Used to organize data in a hierarchical manner
  - Nest elements within other elements
- No pre-defined tags or attributes
  - Use any name for any tag or attribute at any time
- Why use XML?
  - It's a recognized standard
  - There are many great parsing tools available

```javascript
window.onload = loadText;
function loadText()
{
    var ajax = new XMLHttpRequest();
    ajax.onreadystatechange = function() {
        if (ajax.readyState == 4) {
            document.getElementById("...").innerHTML =
                ajax.responseText;
        }
    }
    ajax.open("POST", "...", true);
    ajax.send(null);
}
```
### Proper AJAX

- **Use the** `responseXML` **property of** `XMLHttpRequest`  
  - Contains the fetched data in XML form
- **Use the** DOM methods to access the information

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parentNode</td>
<td>The parent node of the current node</td>
</tr>
<tr>
<td>childNodes[]</td>
<td>An array of the current node's children</td>
</tr>
<tr>
<td>nextSibling, previousSibling</td>
<td>The next and previous nodes with the same parent node</td>
</tr>
<tr>
<td>firstChild, lastChild</td>
<td>The first and last child nodes of the current node</td>
</tr>
<tr>
<td>getElementsByTagName(tagName)</td>
<td>All elements with the given <code>tagName</code></td>
</tr>
</tbody>
</table>