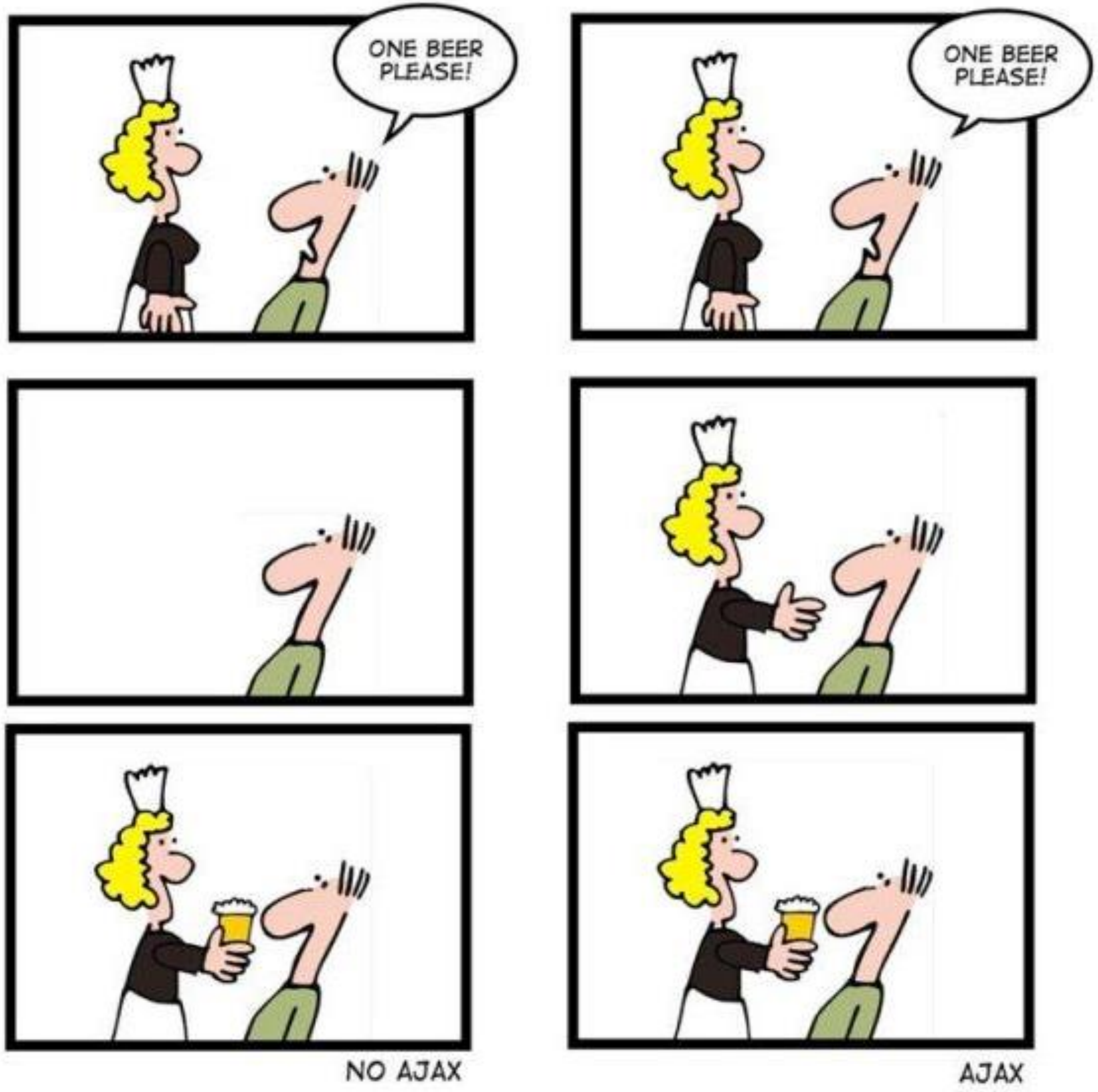
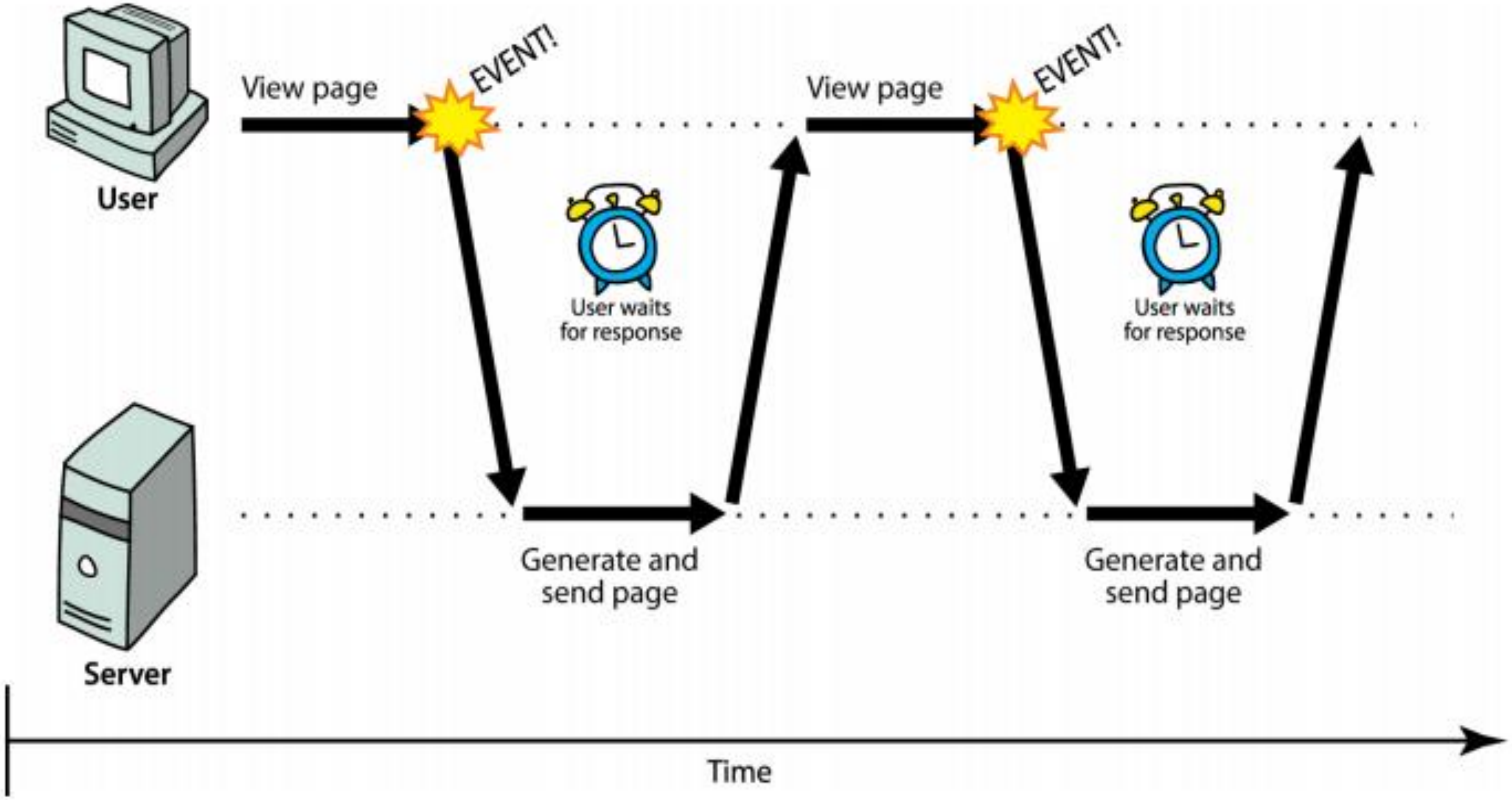
CSC 337

AJAX

Marty Stepp
Rick Mercer
Allison Obourne

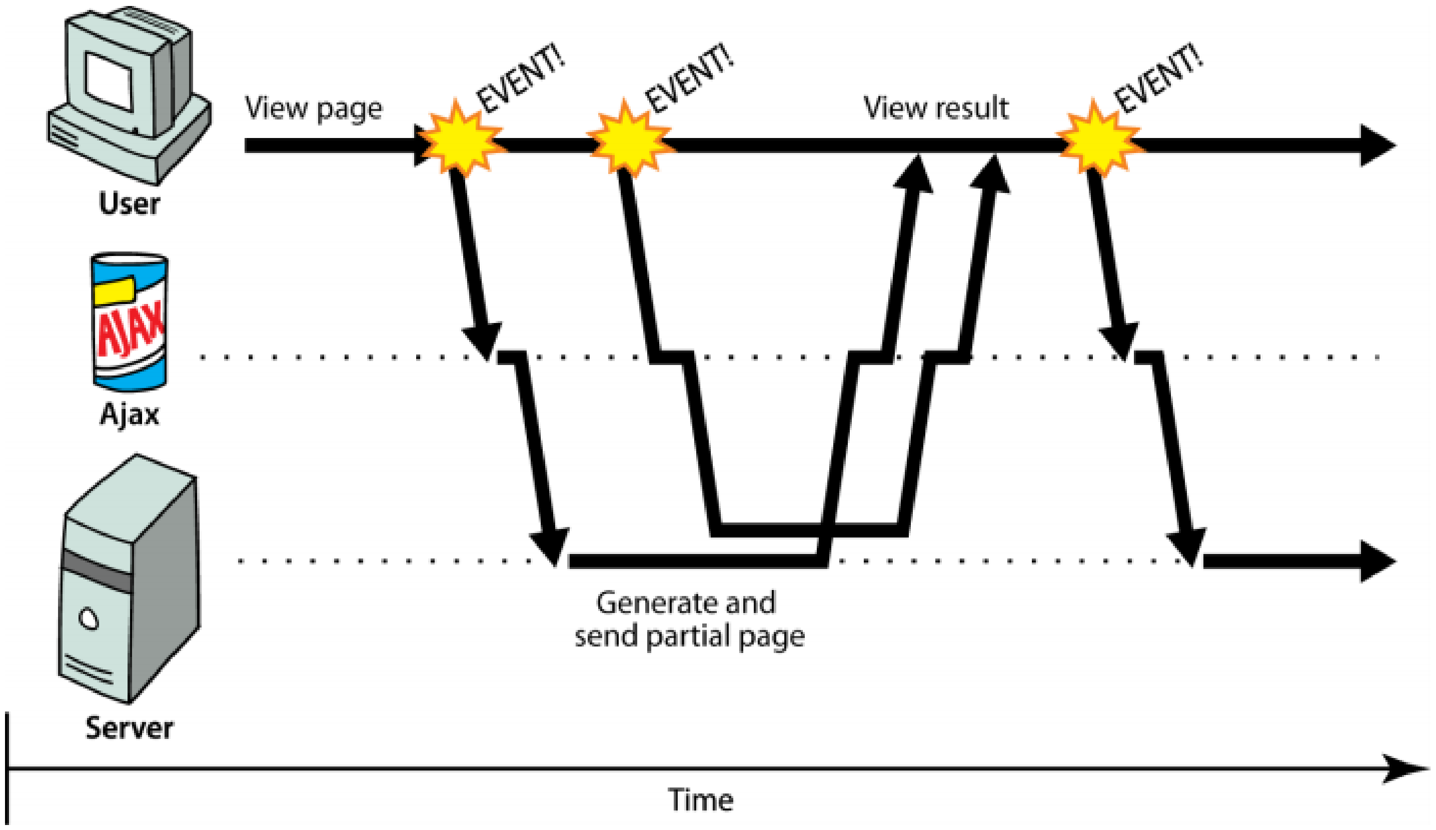


Synchronous web communication



- **Synchronous:** user waits until new page loads
  - the typical communication pattern used in web pages (click, wait, refresh)

Asynchronous web communication



- **Asynchronous:** user can keep interacting with the page while data loads

To Refresh or not to Refresh a Page

- Example to show the difference

http://www.tutorialspoint.com/ajax/ajax_examples.htm

Web applications and AJAX

- Lets the website mimics the feel of a desktop application (aka Rich Internet Application or RIA)
- Presents a continuous user experience instead
- Examples everywhere
  - Gmail, Google Maps, Zillow, Rally, . . .

AJAX

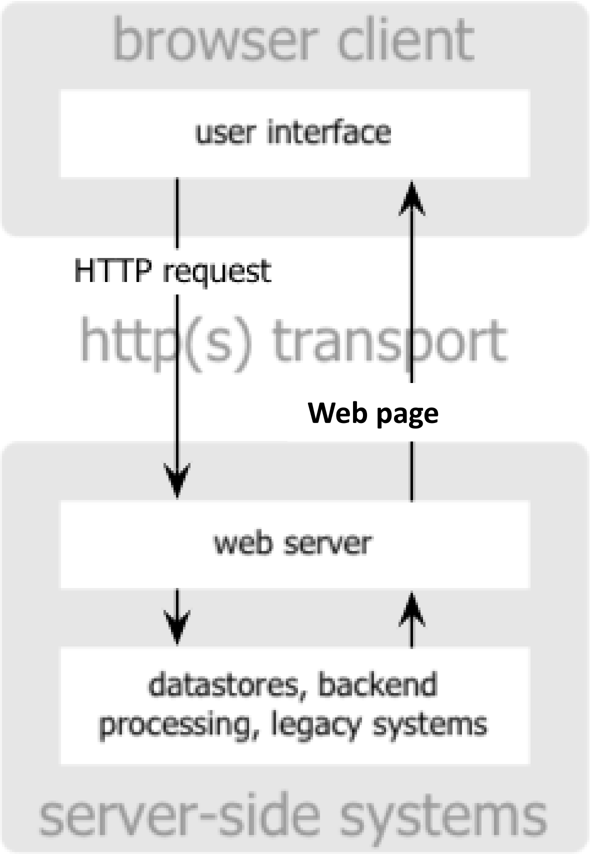
- **AJAX**: **A**synchronous **JA**vascript and **X**ML is a technique for web applications for sending to and receiving data from a server asynchronously, without reloading the page
    - not a programming language
        - it is a particular way of using JavaScript
    - downloads data from a server in the background
    - dynamically updates a part of the page without making the user wait
    - avoids the “click, wait, refresh” pattern
- XML being replaced with JSON, should it be renamed AJAJ?

AJAX from Wikipedia

Ajax is a set of Web development techniques using many Web technologies on the client side to create asynchronous Web applications. With Ajax, Web applications can send data to and retrieve from a server asynchronously (in the background) without interfering with the display and behavior of the existing page. By decoupling the data interchange layer from the presentation layer, Ajax allows for Web pages, and by extension Web applications, to change content dynamically without the need to reload the entire page.

A typical AJAX request

- User generates an event to call an event handler (a JS function)
  - onchange onclick oninput
- Handler creates an XMLHttpRequest object
- XMLHttpRequest object requests data from server
  - Use GET or POST to pass values to server
- Server processes something, like accessing a data base
  - echos back output as plain text, XML, or JSON (discussed later)
- XMLHttpRequest fires an event when data arrives
  - executes a function, known as a callback function
  - you can attach a handler function to this event
- Callback event handler processes data and changes the DOM



browser client

user interface

HTTP request

http(s) transport

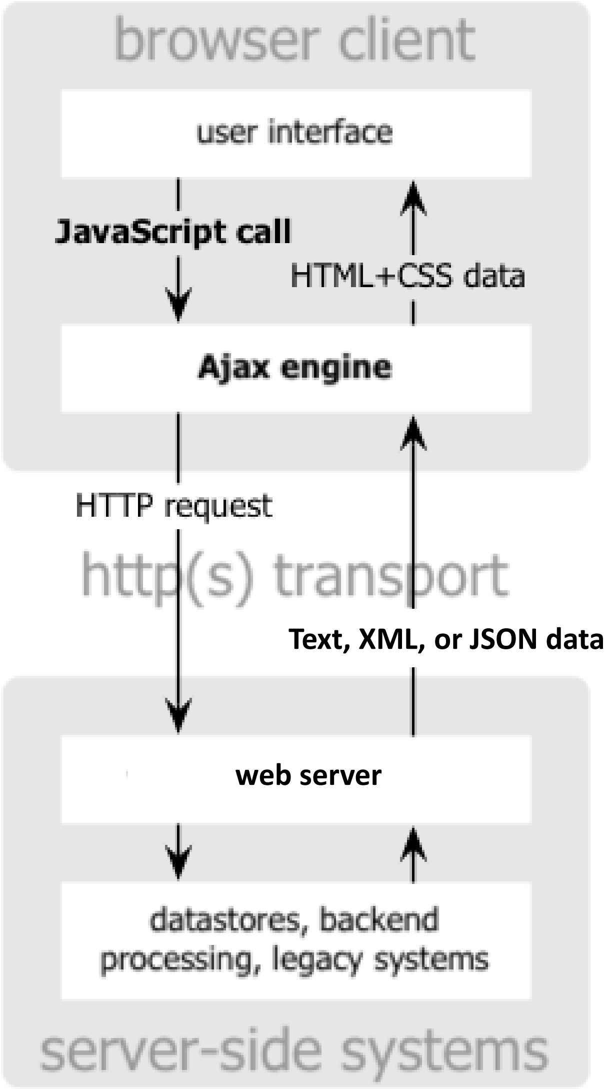
Web page

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

Text, XML, or JSON data

web server

datastores, backend processing, legacy systems

server-side systems

Ajax
web application model

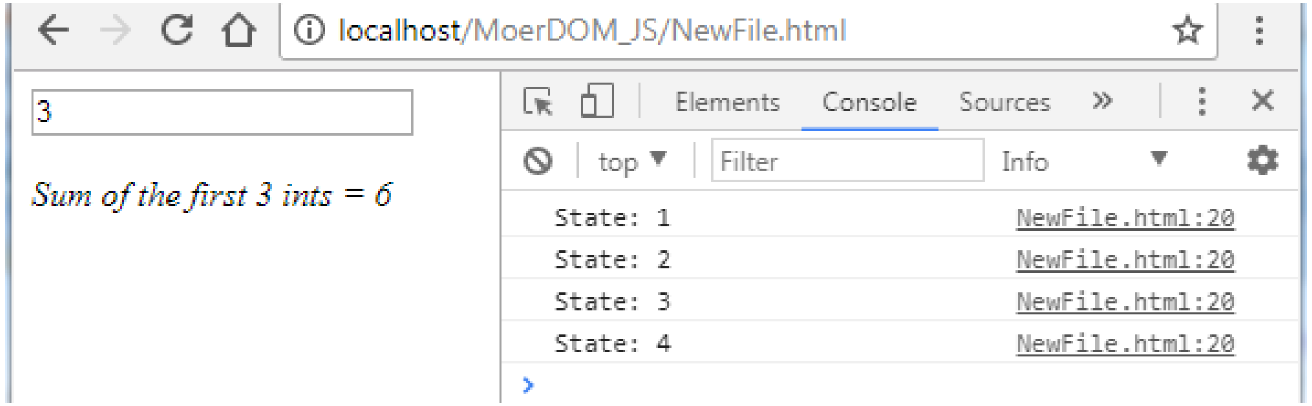
Asynchronous call

- When we make an asynchronous call like this

```
ajax.onreadystatechange = function() {
   console.log("State: " + ajax.readyState);
   if (ajax.readyState == 4 && ajax.status == 200) {
      document.getElementById("divToChange").innerHTML = '<br><i>'
                                    + ajax.responseText + '</i>';
   }
} // End anonymous function
```

we can observe the ready states

- 0 (uninitialized)
- 1 (loading)
- 2 (loaded)
- 3 (interactive)
- 4 (complete)



3

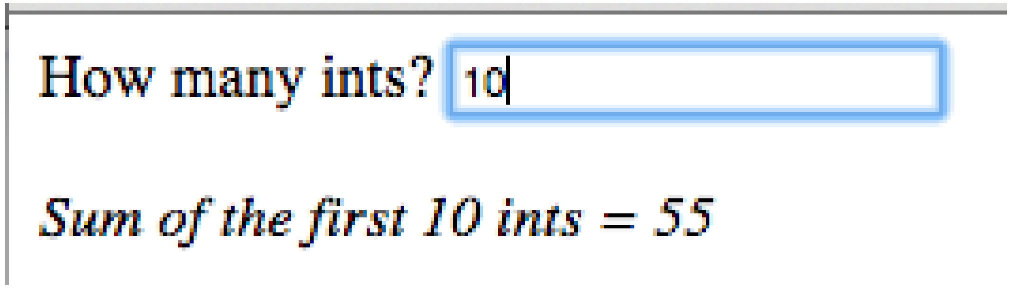
Sum of the first 3 ints = 6

```
State: 1          NewFile.html:20
State: 2          NewFile.html:20
State: 3          NewFile.html:20
State: 4          NewFile.html:20
```

```
<input type="text" id="howMany" onchange="sumOfInts()">
<div id="divToChange">?</div>

<script>
function sumOfInts() {
 var n = document.getElementById("howMany").value;
 var ajax = new XMLHttpRequest();
 ajax.open("GET", "firstN.php?n=" + n, true); // Arguments Method, url, async
 ajax.send();

 // This anonymous callback will execute when the server responds
 ajax.onreadystatechange = function() {
  console.log("State: " + ajax.readyState);
  if (ajax.readyState == 4 && ajax.status == 200) {
   // Do something with xhttp.repsonseText after firstN.php runs
   document.getElementById("divToChange").innerHTML = '<br><i>'
                    + ajax.responseText + '</i>';
  }
 } // End anonymous function
} // End function sumOfInts
</script>
```



How many ints? 10

Sum of the first 10 ints = 55

PHP script used during the asynchronous call

```
<?php
// Filename: firstN.php on the server
$n = $_GET ['n'];

$result = 0;
for($anInt = 1; $anInt <= $n; $anInt++)
   $result += $anInt;

echo 'Sum of the first ' . $n . ' ints = ' . $result;
?>
```

New Things

- Previous example shows many new things
  - Construct an XMLHttpRequest object

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

  - four messages

    xhttp.onreadystatechange **xhttp.**open **xhttp.**send **xhttp.**responseText

  - An anonymous function that executes when the server is done processing the PHP script

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

XMLHttpRequest methods

- The core JavaScript object that makes Ajax possible

Method	**Description**
open(*method*, *url*, *async*)	specifies the URL and HTTP request method

send()
send(*postData*)
	sends the HTTP request to the server, with optional POST parameters

abort()	stops the request
getAllResponseHeaders(), getResponseHeader(*name*), setRequestHeader(*name*,*value*)	for getting/setting raw HTTP headers

XMLHttpRequest attributes (properties)

Property	**Description**
responseText	the entire text of the fetched page, as a string
responseXML	the entire contents of the fetched page, as an XML document tree
status	the request's HTTP status code (200 = OK, etc.)
statusText	HTTP status code text (e.g. "Bad Request" for 400)
timeout	how many MS to wait before giving up and aborting the request (default 0 = wait forever)
readyState	request's current state *(0 = not initialized, 1 = set up, 2 = sent, 3 = in progress, 4 = complete)*

1. Synchronized requests (bad)

```
// this code is in some control's event handler
var ajax = new XMLHttpRequest();
ajax.open("GET", url, false);
ajax.send();
do something with ajax.responseText;
```

- Create the request object, open a connection, send the request
- when send returns, the fetched text will be stored in request's responseText property

Why synchronized requests suck

- Your code waits for the request to completely finish before proceeding
- Easier for you to program, but ...
  - the user's *entire browser LOCKS UP* until the download is completed
  - a terrible user experience (especially if the page is very large or slow to transfer)
- Better solution: use an *asynchronous request* that notifies you when it is complete
  - this is accomplished by learning about the event properties of XMLHttpRequest

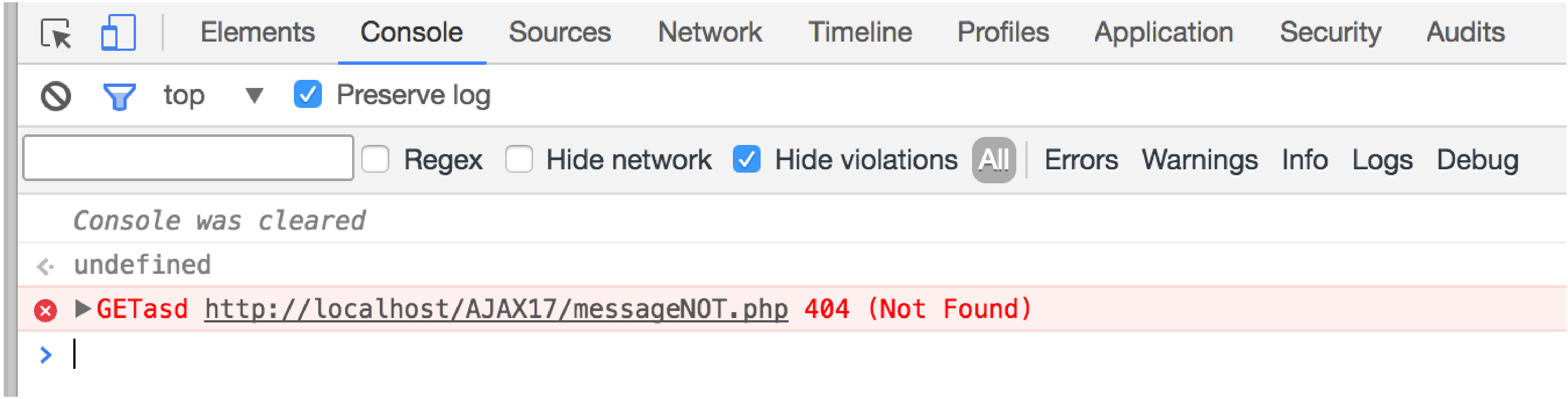
What if the request fails?

```
var ajax = new XMLHttpRequest();
ajax.open("GET", "url", true);
ajax.send();
ajax.onreadystatechange = function {
  if (this.status == 200) {   // 200 means request succeeded
    do something with this.responseText;
  } else {
    code to handle the error;
  }
}
```

- Web servers return status codes for requests (200 means Success)
- You may display a message or take action on a failed request

Debugging Ajax code

```
  }  // End anonymous function
  xhttp.open("GETasd", "messageNOT.php", true);
  xhttp.send();
}
```



```
Console was cleared
<· undefined
✕ ▶GETasd http://localhost/AJAX17/messageNOT.php 404 (Not Found)
> |
```

- Check Console tab (Dev Tools) for errors thrown by requests

Passing query parameters to a request

```
ajax.open("GET", "url?name1=value1&name2=value2&...", true);
ajax.send();
```

- To pass parameters, concatenate them to the URL yourself
  - you may need to URL-encode the parameters by calling the JS `encodeURIComponent(`*`string`*`)` function on them
  - won't work for POST requests *(see next slide)*

POST or GET? *from W3CSchools*

- GET is simpler and faster than POST
- However, always use POST requests when:
  - A cached file is not an option
    - update a file or database on the server
  - Sending a large amount of data to the server
    - POST has no size limitations
- Example code with POST:

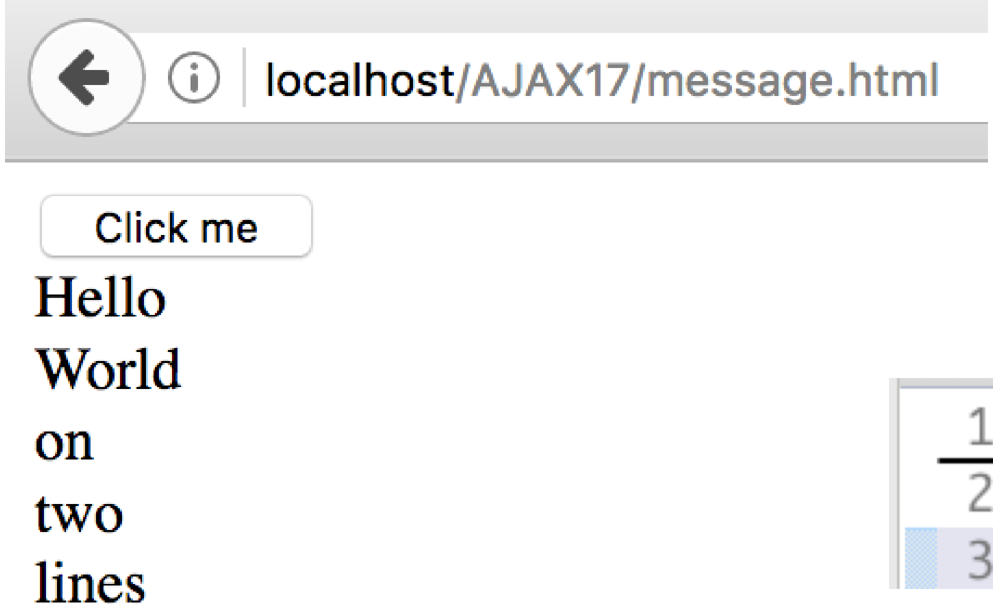
```
xhttp.open("POST", "message.php", true);
xhttp.setRequestHeader("Content-type",
                        "application/x-www-form-urlencoded");
xhttp.send("?n=" + n);
```

Review of AJAX flow from John Mueller

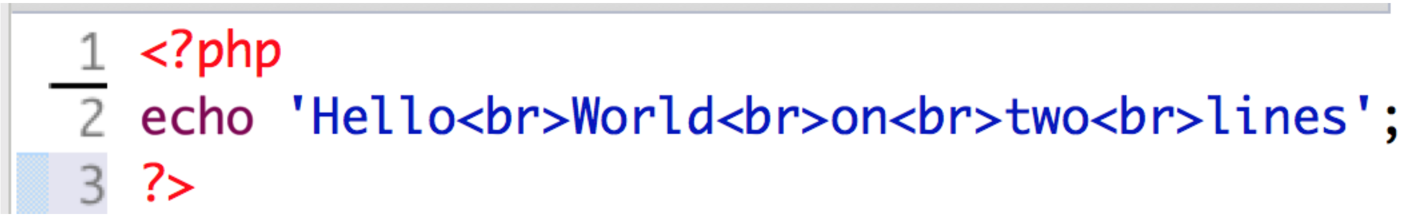
1. An event occurs at the browser.
2. JavaScript creates a new XMLHttpRequest object. The object will be configured to perform its work asynchronously using a callback function
3. JavaScript sends the request to the server for processing.
    - The page continues performing tasks while waiting for a response
4. The server receives the XMLHttpRequest object that JavaScript sent and processes it.
5. The server creates a response and sends it back to the browser.
6. The browser’s callback function provided with the original request receives the response from the server.
7. The callback function performs any required post-processing of the response and a part of the page updates (usually).

Active Collaborative Learning

- In teams of two (find an elbow partner), make an asynchronous call to a php page that sets the innnerHTML to five 'lines' of text generated by a PHP script.



```
Hello
World
on
two
lines
```



```php
<?php
echo 'Hello<br>World<br>on<br>two<br>lines';
?>
```