USER LOGIN AND AJAX

Login and Ajax are different things

Announcements

- There are no course rules for what you may or may not use in your projects.
  - However: check the rules of use of things you get off the internet. Make sure that it is legal for you to use those capabilities.
  - E.g., Check out JQuery, it will help you with tedious stuff.
    - JQuery also supports a lot of scripts you can add to your project. (e.g., JQuery UI)

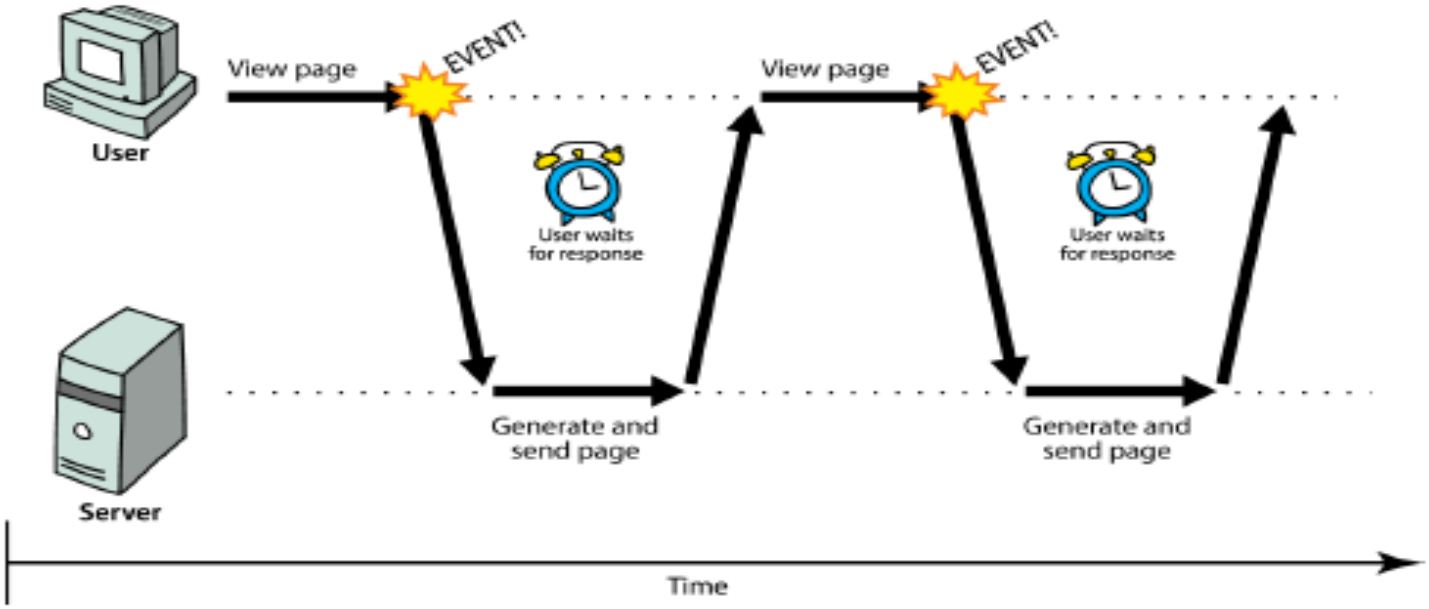
New Scrum Masters

Backslash	MICHAEL	HOLUPKA
C.O.D.E.	MINGJIAN	ZHANG
Cellar	EVAN	BASTA
ContraWeb	RUBY	REYNOSO
Hacklemore	SYDNEY	MARKS
Lanister	CAROLINE	SALIS
Llama	CHRISTOPHER	BELL
Scrum City	MERRILL	PRESKA-STEINBERG
Sk3m Team	MATTHEW	NING
SqlThePrql	JEREMY	WARNER
Tautology	TAIT	MADSEN

Topics

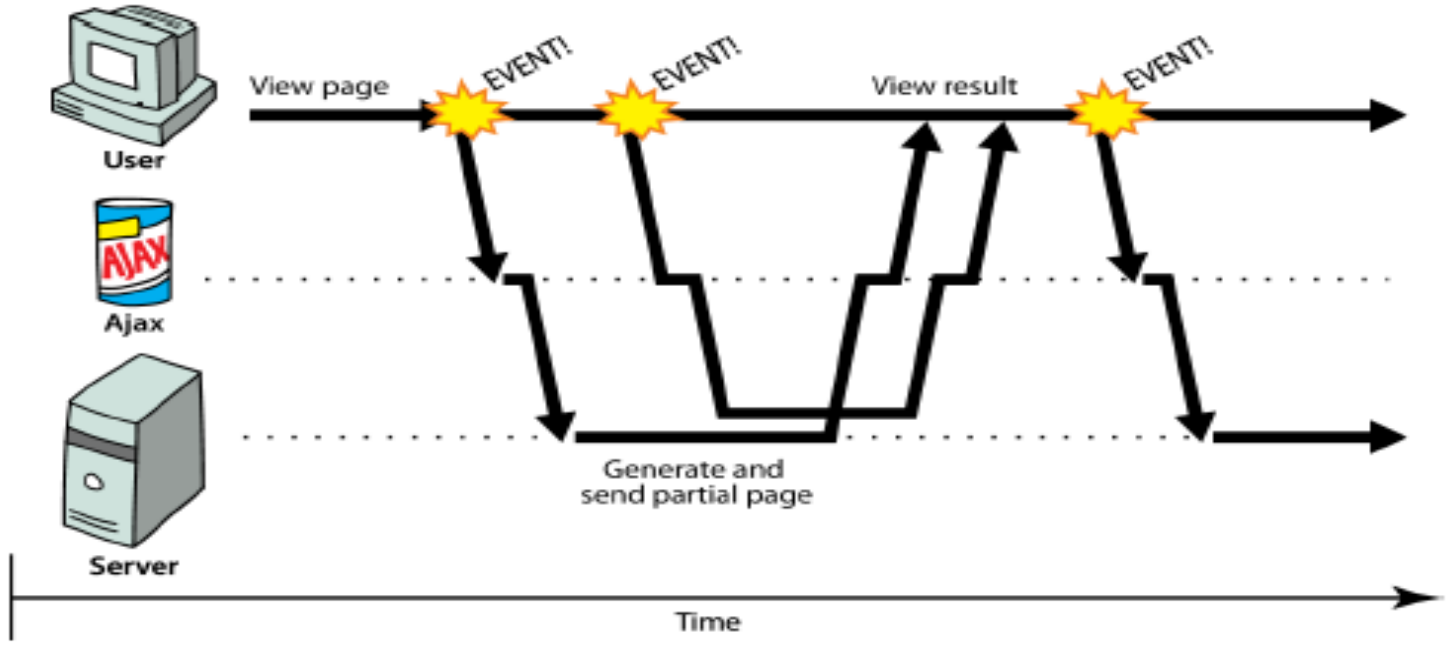
- Cookies: data sent back and forth between browser and server
- Sessions: Managing information on user based on cookies
- Ajax: Asynchronous HTTP

Synchronous web communication



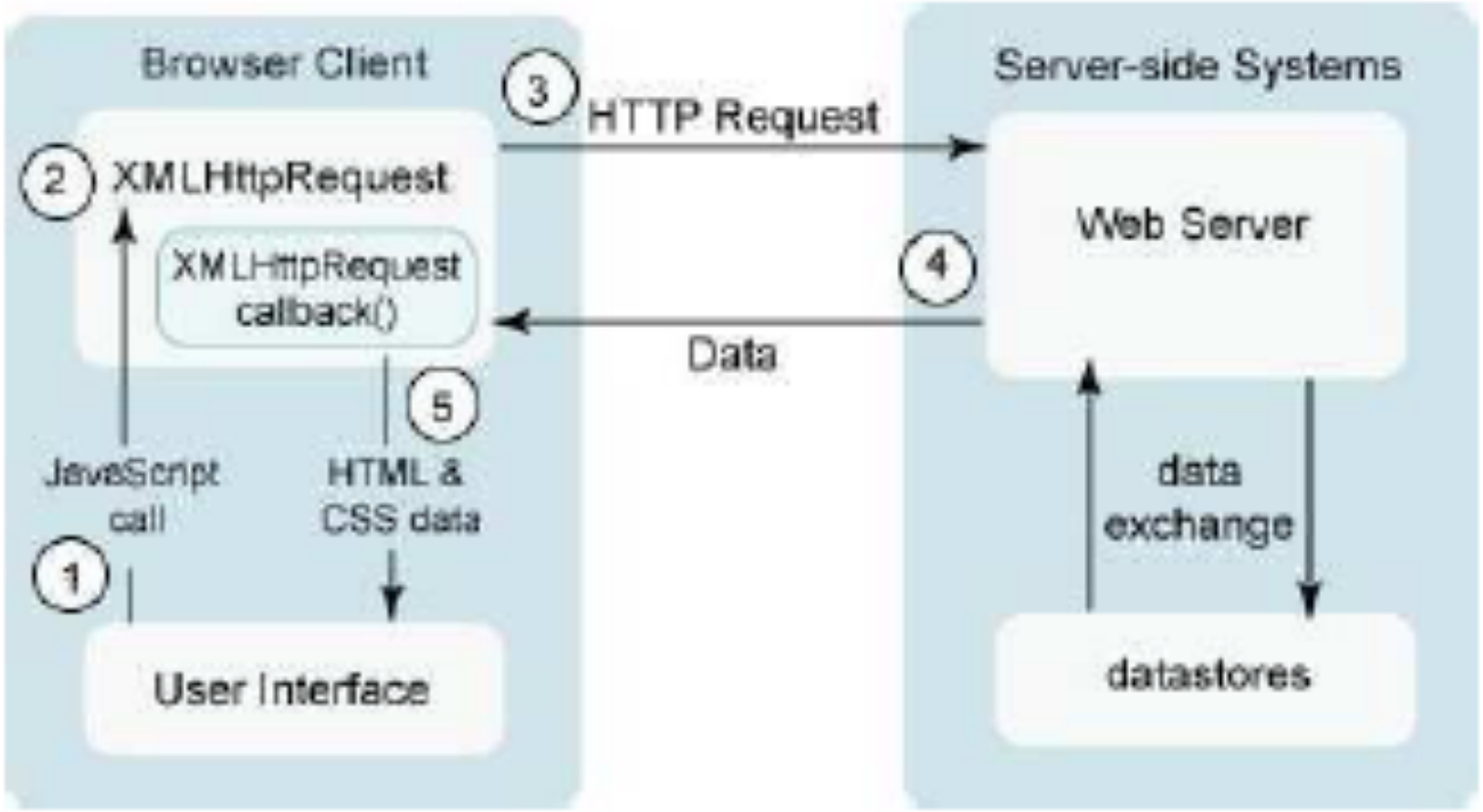
- synchronous: user must wait while new pages load
  - the typical communication pattern used in web pages (click, wait, refresh)

Asynchronous web communication



- **asynchronous:** user can keep interacting with page while data loads
  - communication pattern made possible by Ajax

A typical Ajax request



Standup

Discuss questions with your Scrum Team

Quiz

Quiz

1. How long does a session cookie last?
   a. Until the time out
   b. Until the server requests it
   c. until the client logs out
   d. Until the browser terminates
2. Which global abject are cookies associated with?
   a. window
   b. document
   c. navigation
   d. history
3. What is the first step in establishing a session
   a. The user requests a page
   b. The server sends a cookie
   c. The user sends a cookie
4. Ajax lets HTTP be
   a. asynchronous
   b. synchronous
   c. stateless
   d. anhydrous

And the answer is …

Quiz

1. How long does a session cookie last?

d. Until the browser terminates

2. Which global abject are cookies associated with?

b. document

3. What is the first step in establishing a session

a. The user requests a page

4. Ajax lets HTTP be

a. asynchronous

Cookies

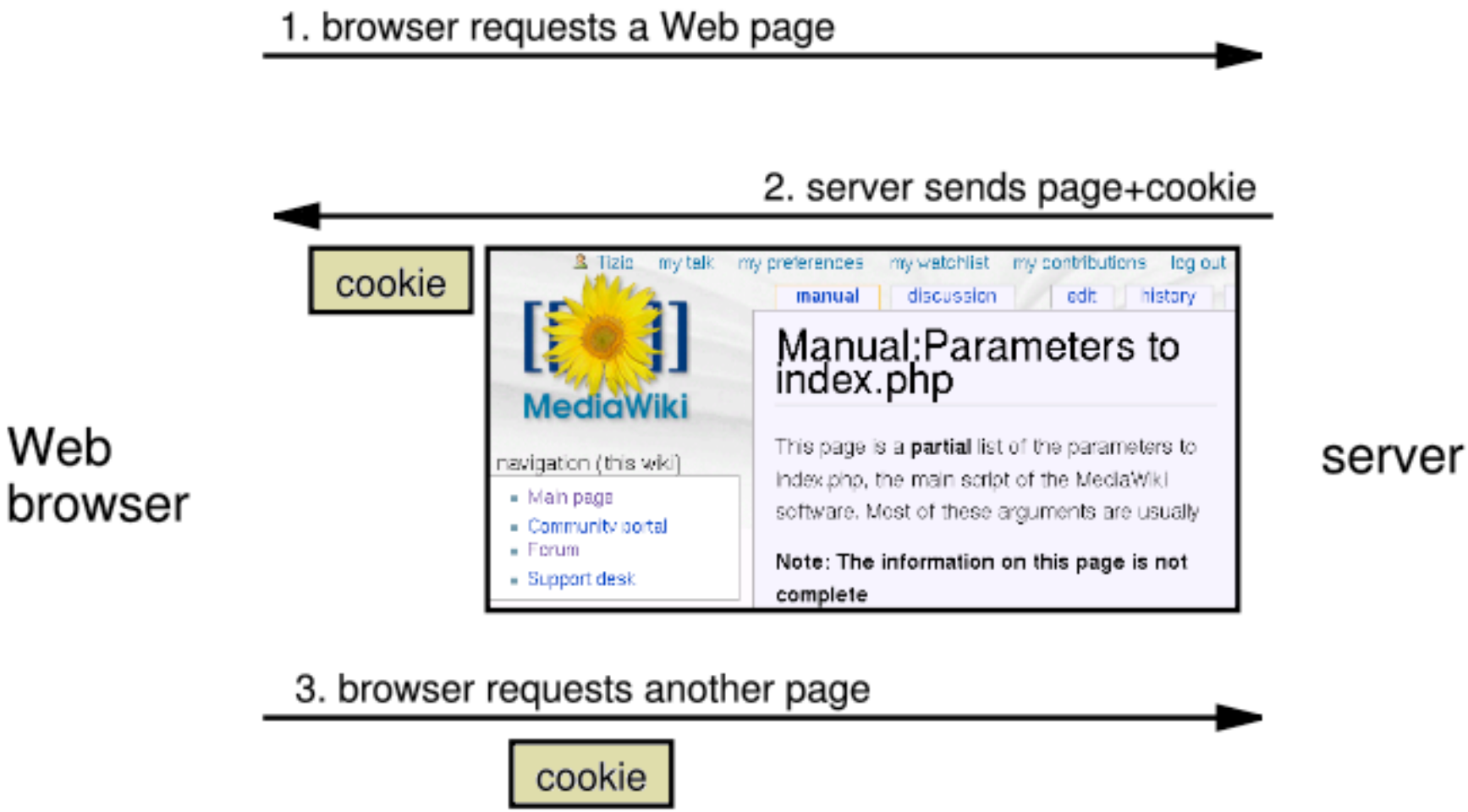
What is a cookie?

- cookie: a small amount of information sent by a server to a browser, and then sent back by the browser on future page requests
- cookies have many uses:
  - authentication
  - user tracking
  - maintaining user preferences, shopping carts, etc.
- a cookie's data consists of a single name/value pair, sent in the header of the client's HTTP GET or POST request

How cookies are sent

- when the browser requests a page, the server may send back a cookie(s) with it
- if your server has previously sent any cookies to the browser, the browser will send them back on subsequent requests
- alternate model: client-side JavaScript code can set/get cookies

How cookies are sent



How long does a cookie exist?

- session cookie (default): stored only in the browser's memory
  - when the browser closes, temporary cookies are erased
  - cannot be used for tracking long-term information
  - safer, because only the browser can access them
- persistent cookie: stored in a file on the browser's computer
  - can track long-term information
  - less safe, because users can see/change the cookie values, etc.

Cookies in JavaScript


```
document.cookie = "username=smith";   // setting two cookies
document.cookie = "password=12345";
// deleting a cookie
document.cookie = "age=29; expires=Thu, 01-Jan-1970 00:00:01 GMT";
...
```

- JS has a global document.cookie field (a string)
- you can manually set/get cookie data from this field (sep. by ;), and it will be saved in the browser
- to delete a cookie, set it to 'expire' in the past

Cookies in JavaScript


```
// (later)
var allCookies = document.cookie.split(";");    // ["username=smith", "password=12345"]
for (var i = 0; i < allCookies.length; i++) {
  var eachCookie = allCookies[i].split("=");    // ["username", "smith"]
  var cookieName = eachCookie[0];                // "username"
  var cookieValue = eachCookie[1];               // "smith"
  ...
}
```

- you can manually set/get cookie data from this field (sep. by ;), and it will be saved in the browser

Provided Cookie library


```
<!-- using the instructor-provided Cookie.js class -->
<script src="http://www.webstepbook.com/Cookie.js" type="text/javascript"></script>

Cookie.set("username", "smith");
// (later)
alert(Cookie.get("username"));   // smith
```

- The book authors have written a Cookie.js helper class with methods set, get, exists, remove, and remember

Setting a cookie in PHP


```
setcookie("name", "value");

setcookie("username", "martay");
setcookie("favoritecolor", "blue");
```

- setcookie causes your script to send a cookie to the user's browser
- setcookie must be called before any output statements (HTML blocks, print, or echo)
- you can set multiple cookies (20-50) per user, each up to 3-4K bytes

Setting a cookie in PHP (alt)


```
header("Set-Cookie: username=martay; path=/; secure");
```

- technically, a cookie is just part of an HTTP header, and it could be set using PHP's header function (but this is less convenient, so you would not want to do this)

Retrieving info from a cookie


```
$variable = $_COOKIE["name"];   # retrieve value of the cookie

if (isset($_COOKIE["username"])) {
  $username = $_COOKIE["username"];
  print("Welcome back, $username.\n");
} else {
  print("Never heard of you.\n");
}
print("All cookies received:\n");
print_r($_COOKIE);
```

- cookies sent back to $_COOKIE associative array
- use isset function shows if a given cookie name exists
- unset function deletes a cookie

Setting a persistent cookie in PHP


```
setcookie("name", "value", timeout);

$expireTime = time() + 60*60*24*7;   # 1 week from now
setcookie("CouponNumber", "389752", $expireTime);
setcookie("CouponValue", "100.00", $expireTime);
```

- to set a persistent cookie, pass a third parameter for its timeout in seconds
- time function returns the current time in seconds
  - date function can convert a time in seconds to a readable date

Removing a persistent cookie


```
setcookie("name", "", time() - 1);

setcookie("CouponNumber", "", time() - 1);
```

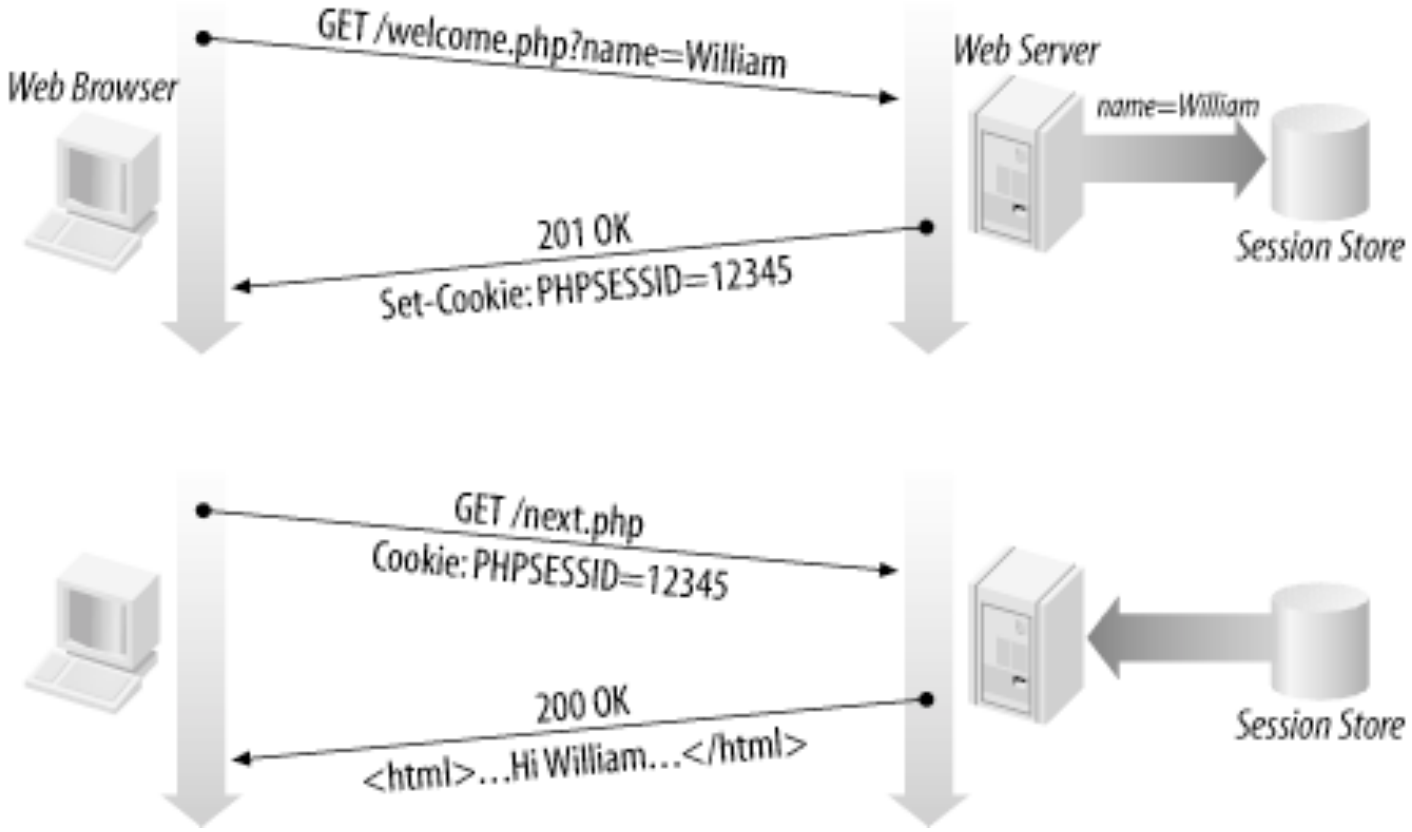
- if the server wants to remove a persistent cookie, it should set it again, passing a timeout that is prior to the present time

Sessions

What is a session?

- session: a sequence of requests and responses between a browser and server
  - HTTP doesn't support sessions, but PHP does
- sessions vs. cookies:
  - a cookie is data stored on the client
  - a session's data is stored on the server (1 per client)
- sessions are often built on top of cookies:
  - client only stores is a cookie holding a unique session ID
    - client sends its session ID from cookie
    - the server retrieves the client's session data with the ID

How sessions are established



How sessions are established

- client's browser makes an initial request to the server
- server notes client's IP address/browser, stores some local session data, and sends a session ID back to client
- client sends that same session ID back to server on future requests
- server uses session ID to retrieve the data for the client's session later

Sessions in PHP: session_start

session_start();

- session_start signifies your script wants a session with the user must be called at the top of your script, before any HTML output is produced
- when you call session_start: if the server hasn't seen this user before, a new session is created
- otherwise, existing session data is loaded into $_SESSION associative array
- you can store data in $_SESSION and retrieve it on future pages

Session Functions

- Complete list of session functions from us.php.net
  - http://us.php.net/manual/en/ref.session.php

Where is session data stored?

- on the client, the session ID is stored as a cookie with the name PHPSESSID
- on the server, session data are stored as temporary files such as /tmp/sess_fcc17f071...
- you can find (or change) the folder where session data is saved using the session_save_path function
- for very large applications, session data can be stored into a SQL database (or other destination) using the session_set_save_handler function

Session timeout

- the server doesn't know when a session is over
  - ideally, user explicitly logs out, but many don't
  - client deletes session cookies when browser closes
- server cleans up old sessions after a while
  - old sessions consume resources; present a security risk
  - adjustable in PHP server settings or with session_cache_expire function
  - explicitly delete a session by calling session_destroy

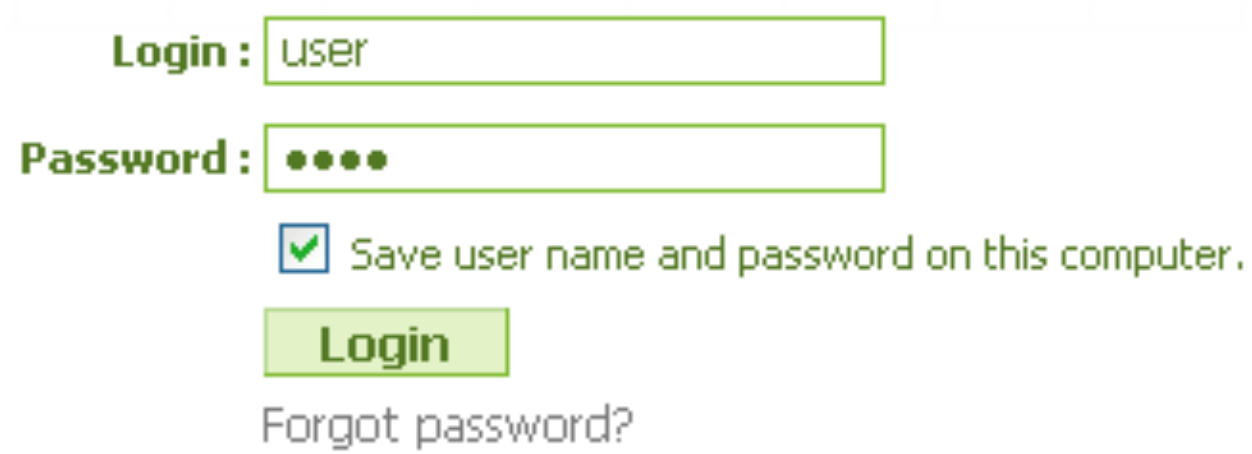
Ending a session


```
session_destroy();
session_regenerate_id(TRUE);   # flushes out session ID number
session_start();
```

- session_destroy ends your current session
- if you call session_start again later, it sometimes reuses the same session ID/data you used before
- if you want to start a completely new empty session later, it is best to flush out the old one

Implementing User Logins

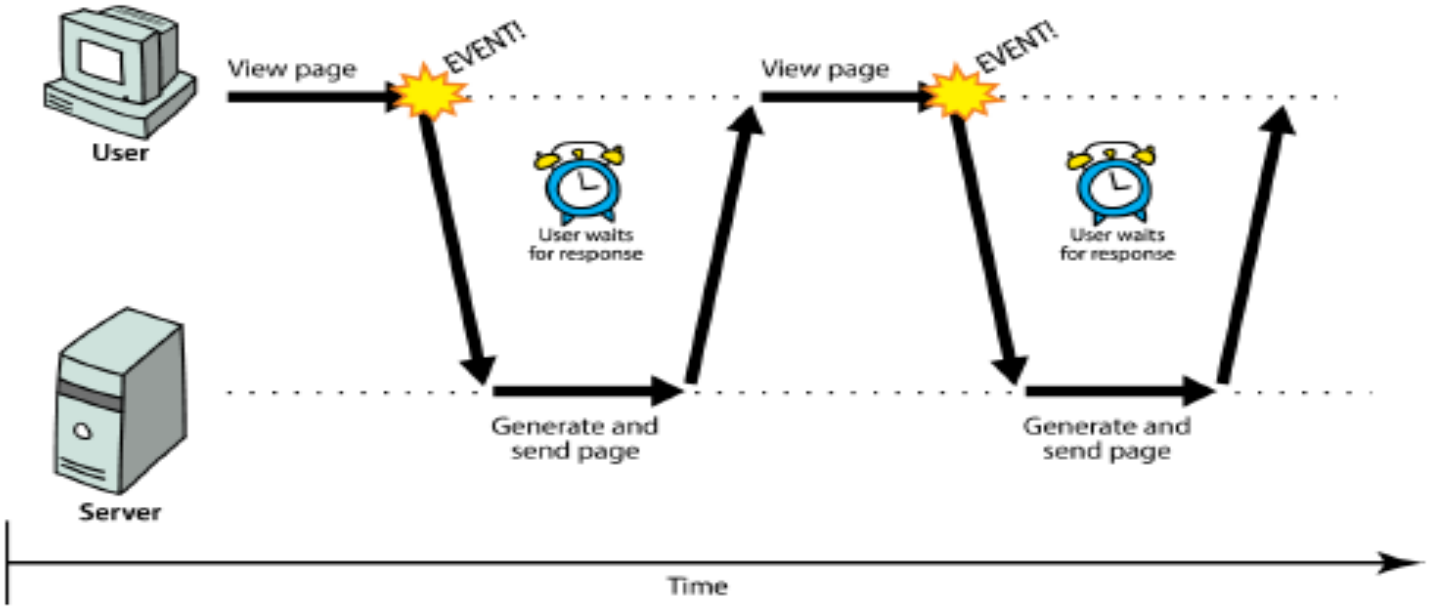
Implementing user logins



- many sites create accounts and log in users
- most apps have a database of user accounts
- when you try to log in, your name/pw are compared to those in the database

Ajax

Synchronous web communication



- synchronous: user must wait while new pages load
  - the typical communication pattern used in web pages (click, wait, refresh)

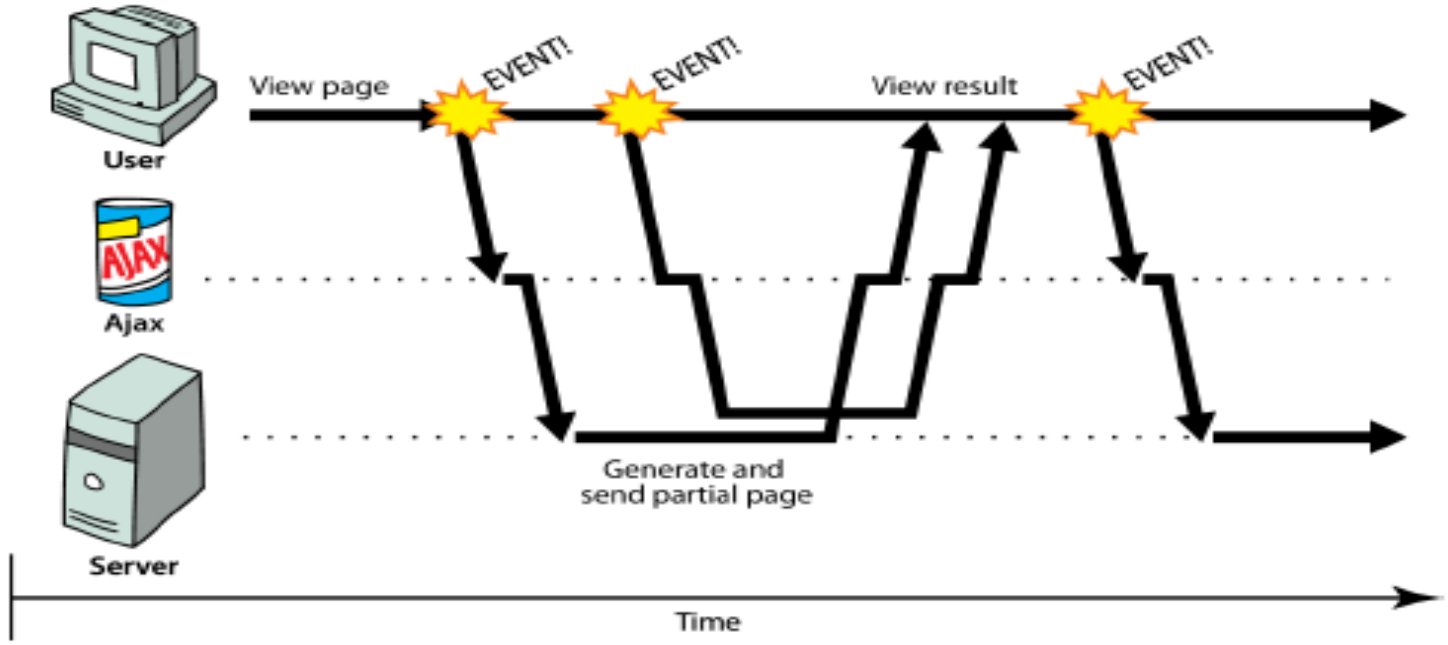
Web applications and Ajax

- **web application:** a dynamic web site that mimics the feel of a desktop app
  - presents a continuous user experience rather than disjoint pages
  - examples: Gmail, Google Maps, Google Docs and Spreadsheets, Flickr, A9

Web applications and Ajax

- **Ajax**: Asynchronous JavaScript and XML
  - not a programming language; a particular way of using JavaScript
  - downloads data from a server in the background
  - allows dynamically updating a page without making the user wait
  - avoids the "click-wait-refresh" pattern
  - Example: Google Suggest

Asynchronous web communication



- **asynchronous:** user can keep interacting with page while data loads
  - communication pattern made possible by Ajax

`XMLHttpRequest` (and why we won't use it)

- JavaScript includes an XMLHttpRequest object that can fetch files from a web server
  - supported in IE5+, Safari, Firefox, Opera, Chrome, etc. (with minor compatibilities)
- it can do this asynchronously (in the background, transparent to user)
- the contents of the fetched file can be put into current web page using the DOM

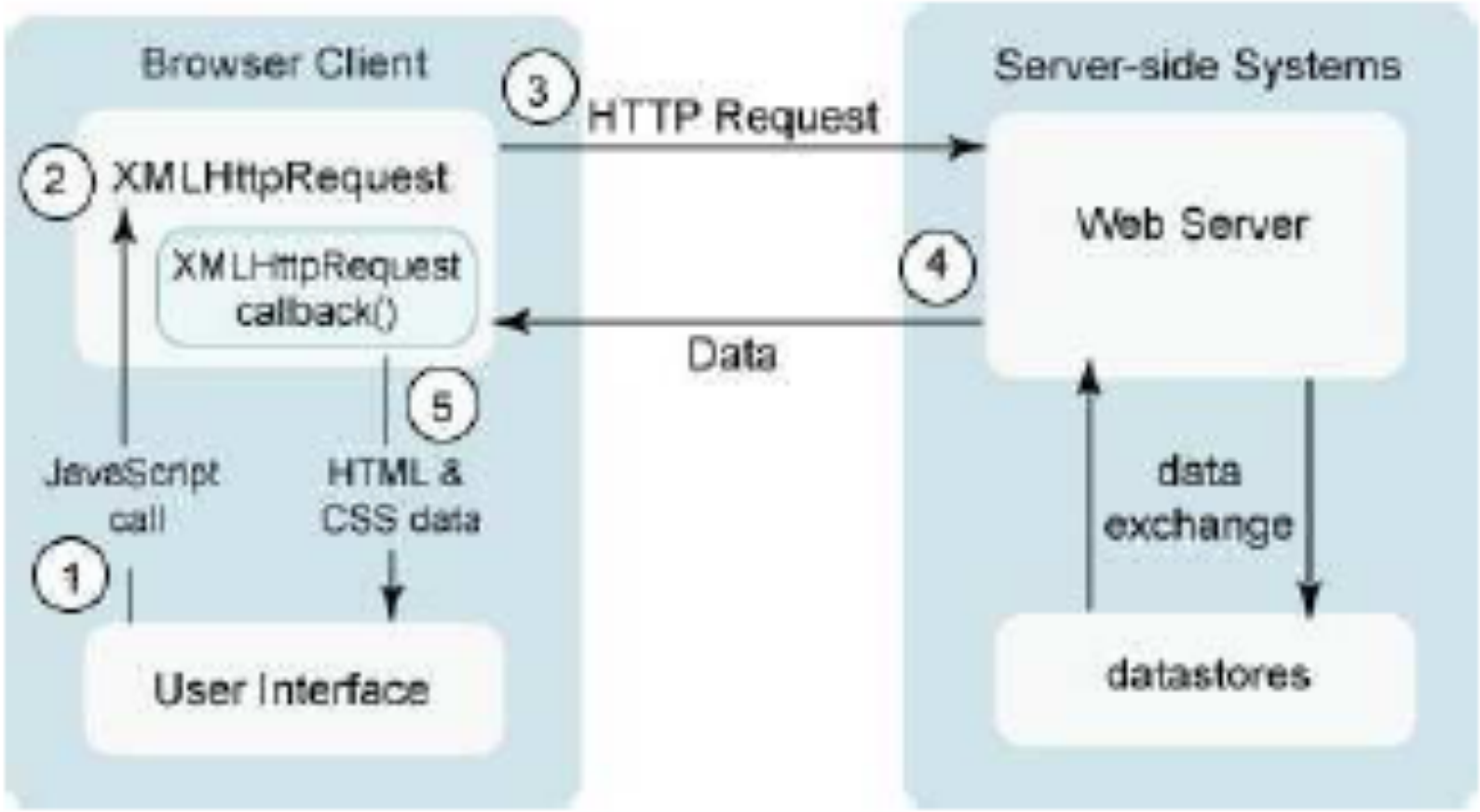
`XMLHttpRequest` (and why we won't use it)

- sounds great!...
- ... but it is clunky to use, and has various browser incompatibilities
- Prototype provides a better wrapper for Ajax, so we will use that instead

A typical Ajax request

1. user clicks, invoking an event handler
2. handler's code creates an `XMLHttpRequest` object
3. `XMLHttpRequest` object requests page from server
4. server retrieves appropriate data, sends it back
5. `XMLHttpRequest` fires an event when data arrives
   - this is often called a callback
   - you can attach a handler function to this event
6. your callback event handler processes the data and displays it

A typical Ajax request



Prototype's Ajax model


```
new Ajax.Request("url",
{
        option : value,
        option : value,
        ...
        option : value
}
);
```
JS

- construct a Prototype Ajax.Request object to request a page from a server using Ajax
- constructor accepts 2 parameters:
  1. the URL to 1. fetch, as a String,
  2. a set of options, as an array of key : value pairs in {} braces (an anonymous JS object)

Prototype Ajax methods and properties

option	description
method	how to fetch the request from the server (default "post")
parameters	query parameters to pass to the server, if any
asynchronous (default true), contentType, encoding, requestHeaders	

options that can be passed to the Ajax.Request constructor

Prototype Ajax methods and properties

event	description
onSuccess	request completed successfully
onFailure	request was unsuccessful
onException	request has a syntax error, security error, etc.

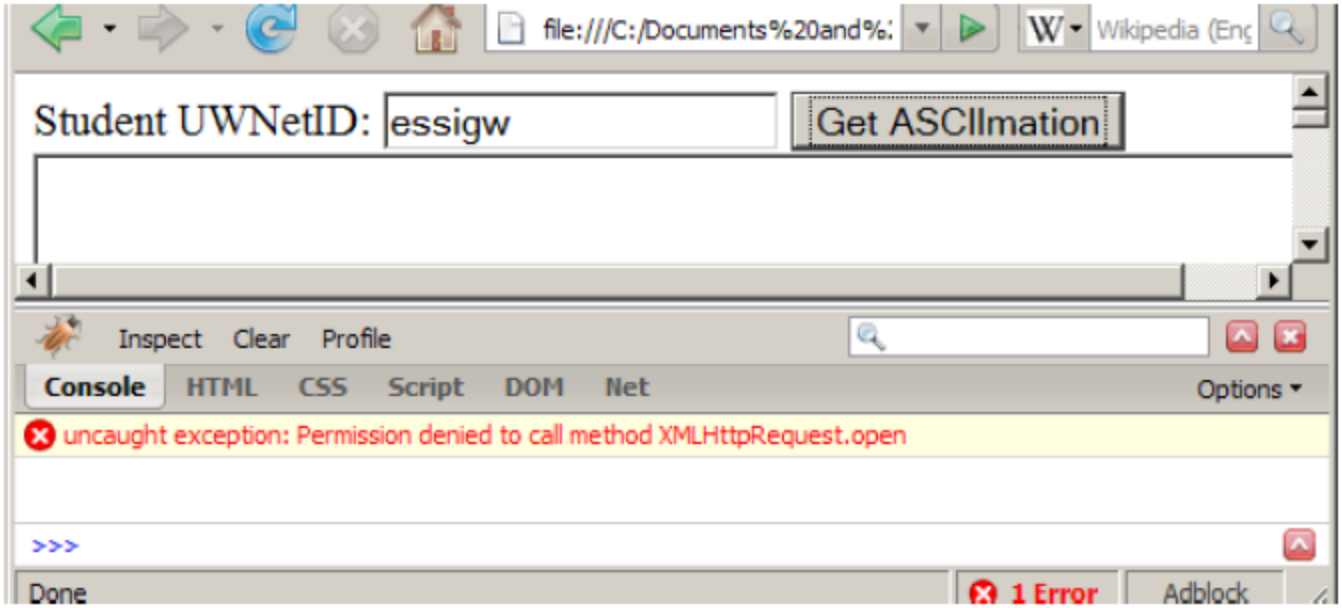
events in the Ajax.Request object that you can handle

Basic Prototype Ajax template

property	description
status	the request's HTTP error code (200 = OK, etc.)
statusText	HTTP error code text
responseText	the entire text of the fetched page, as a String
responseXML	the entire contents of the fetched page, as an XML DOM tree (seen later)

```
function handleRequest(ajax) {
        alert(ajax.responseText);
}
```

XMLHttpRequest security restrictions



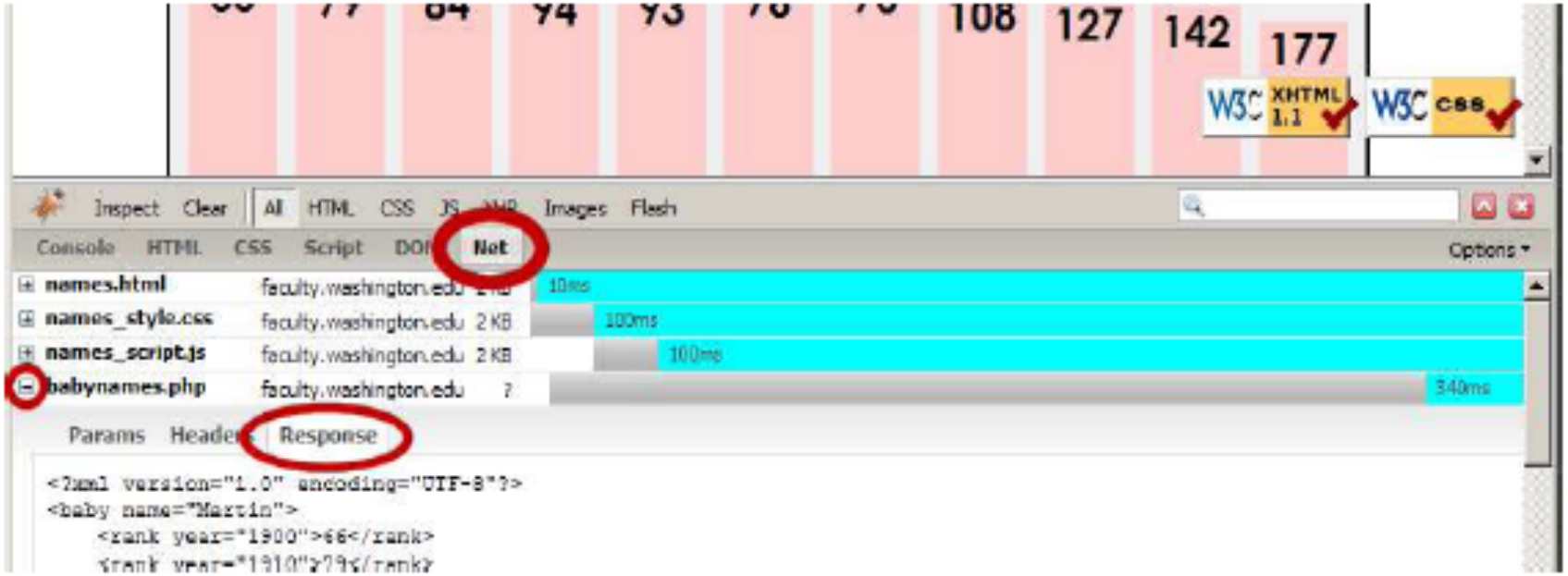
- cannot be run from a web page stored on your hard drive
- can only be run on a web page stored on a web server
- can only fetch files from the same site that the page is on www.foo.com/a/b/c.html can only fetch from www.foo.com

Handling Ajax errors


```
new Ajax.Request("url",
{
	method: "get",
	onSuccess: functionName,
	onFailure: ajaxFailure,
	onException: ajaxFailure
}
);
...
function ajaxFailure(ajax, exception) {
	alert("Error making Ajax request:" + "\n\nServer
status:\n" + ajax.status + " " + ajax.statusText +
"\n\nServer response text:\n" + ajax.responseText);
	if (exception) {
		throw exception;
	}
}
```

JS

Debugging Ajax code



- Net tab shows each request, its parameters, response, any errors
- expand a request with + and look at Response tab to see Ajax result

Creating a POST request


```
new Ajax.Request("url",
{
        method: "post", // optional
        parameters: { name: value, name: value, ..., name:
value },
        onSuccess: functionName,
        onFailure: functionName,
        onException: functionName
}
);                                                          JS
```

Creating a POST request

- Ajax.Request can also be used to post data to a web server
- method should be changed to "post" (or omitted; post is default)
- any query parameters should be passed as a parameters parameter
  - written between {} braces as a set of name : value pairs (another anonymous object)
  - get request parameters can also be passed this way, if you like

Prototype's Ajax Updater


```
new Ajax.Updater(
        "id",
        "url",
        {
                method: "get"
        }
);
```
JS

- Ajax.Updater fetches a file and injects its content into an element as innerHTML
- additional (1st) parameter specifies the id of element to inject into