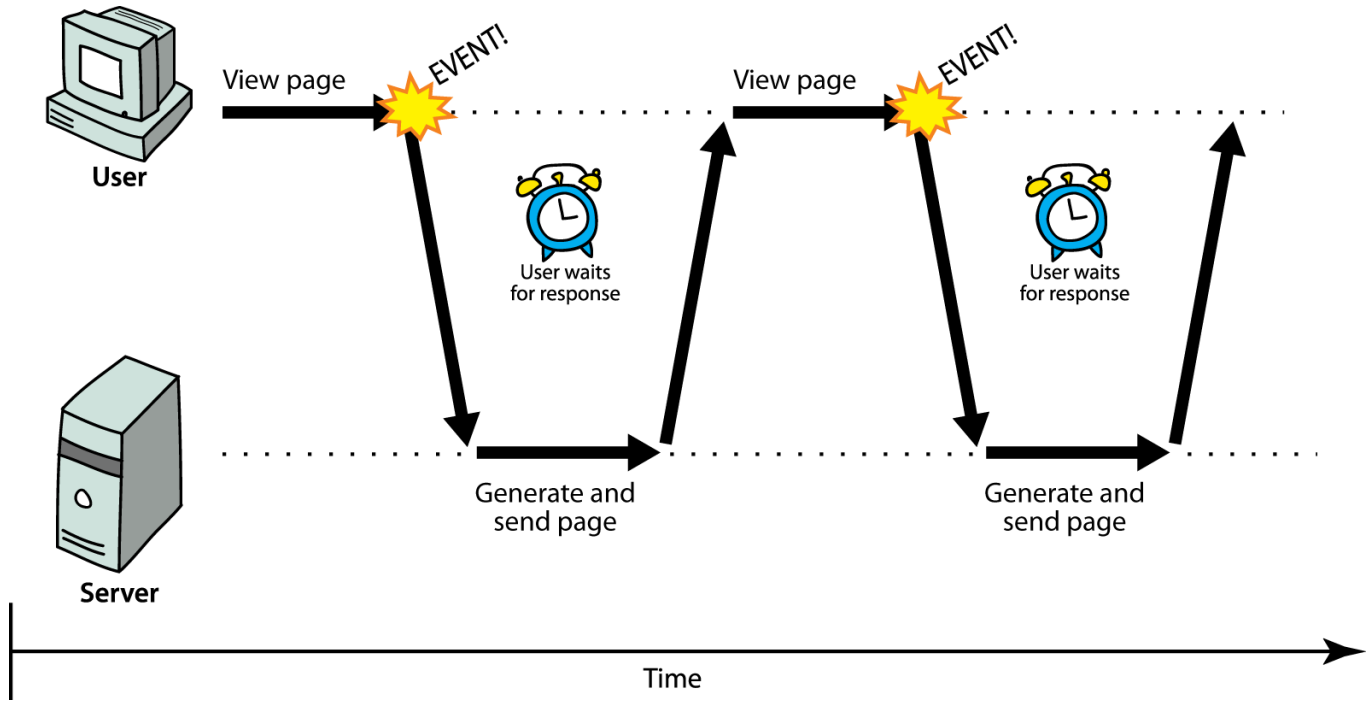
CSE 154

LECTURE 22: AJAX

Synchronous web communication

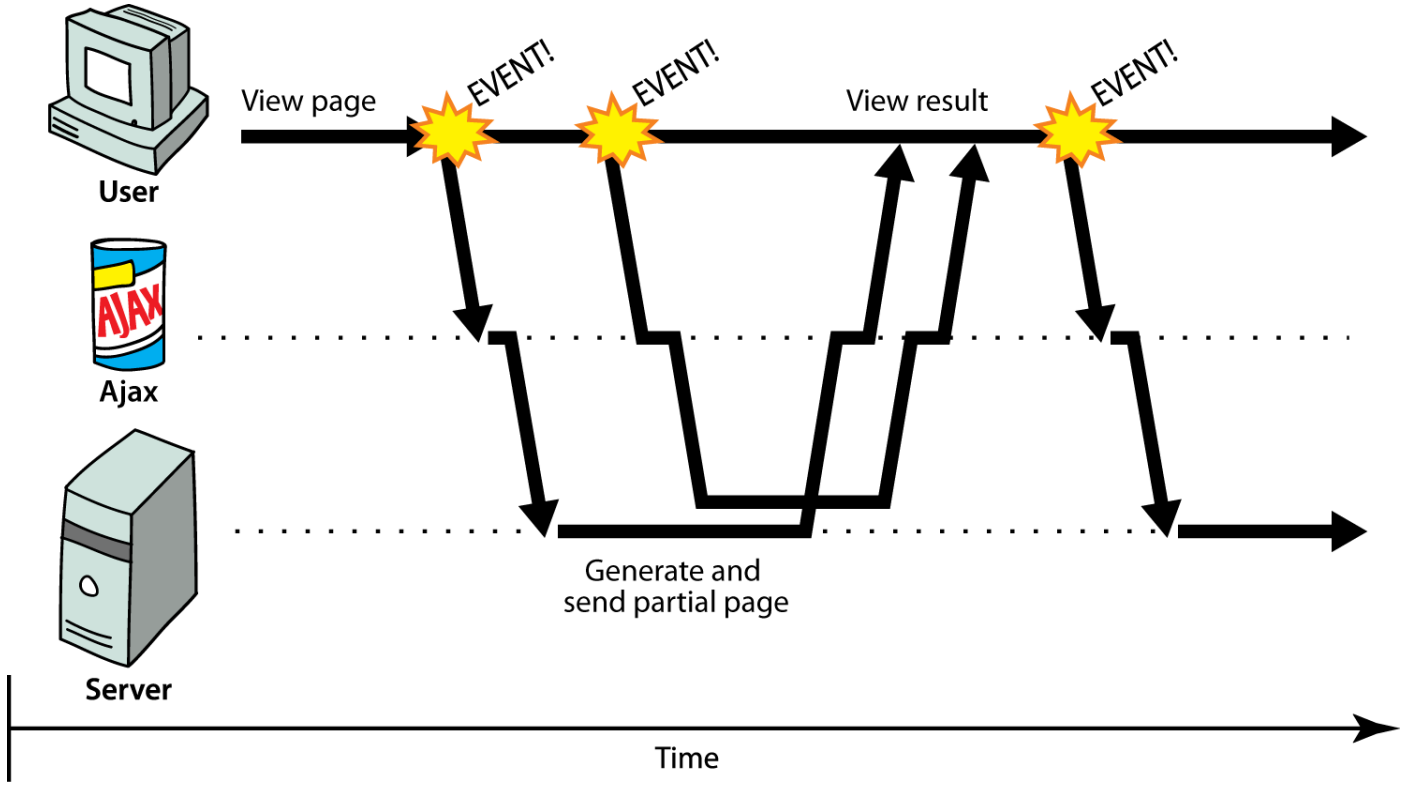


- **synchronous**: user must wait while new pages load
  - the typical communication pattern used in web pages (click, wait, refresh)

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
- **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
  - examples: UW's CSE 14x Diff Tool, Practice-It; Google Suggest

Asynchronous web communication



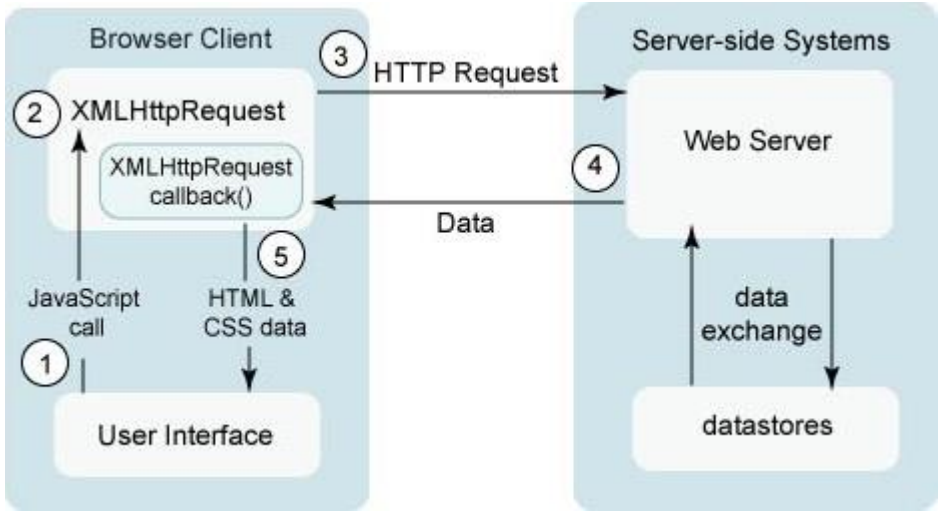
- **asynchronous**: user can keep interacting with page while data loads

The XMLHttpRequest object

- JavaScript includes an `XMLHttpRequest` object that can fetch files from a web server
  - supported in IE7+, Safari, Firefox, Opera, Chrome, etc. (all major browsers)
  - IE5/6 don't have it, but we will ignore this
  - (technically, MS/IE invented `XMLHttpRequest` and Ajax for use in an online version of MS Outlook (credit where it's due!)
- 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

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



Levels of using XMLHttpRequest

1. synchronized, text/HTML-only (SJAT?)
2. asynchronous, text/HTML-only (AJAT?)
3. asynchronous w/ XML data (Ajax ... seen next lecture)

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 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 (seen later)
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;
```
JS

- 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`

XMLHttpRequest events

Event	Description
load	occurs when the request is completed
error	occurs when the request fails
timeout	occurs when the request times out
abort	occurs when the request is aborted by calling abort()
loadstart, loadend, progress, readystatechange	progress events to track a request in progress

2. Asynchronous requests, basic idea

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("GET", url, true);
ajax.send();
...
function functionName() {
  do something with this.responseText;
}
```
JS

- attach an event handler to the `load` event
- handler will be called when request state changes, e.g. finishes
- *function* contains code to run when request is complete
  - inside your handler function, `this` will refer to the `ajax` object
  - you can access its `responseText` and other properties

What if the request fails?

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

JS

- web servers return status codes for requests (200 means Success)
- you may wish to display a message or take action on a failed request

Handling the error event

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.onerror = errorFunctionName;
ajax.open("GET", "url", true);
ajax.send();
...
function functionName(e) {
  do something with e, this.status, this.statusText, ...
}
```
JS

- the graceful way to handle errors is to listen for the `error` event
- the handler is passed the error/exception as a parameter
- you can examine the error, as well as the request status, to determine what went wrong

Example Ajax error handler

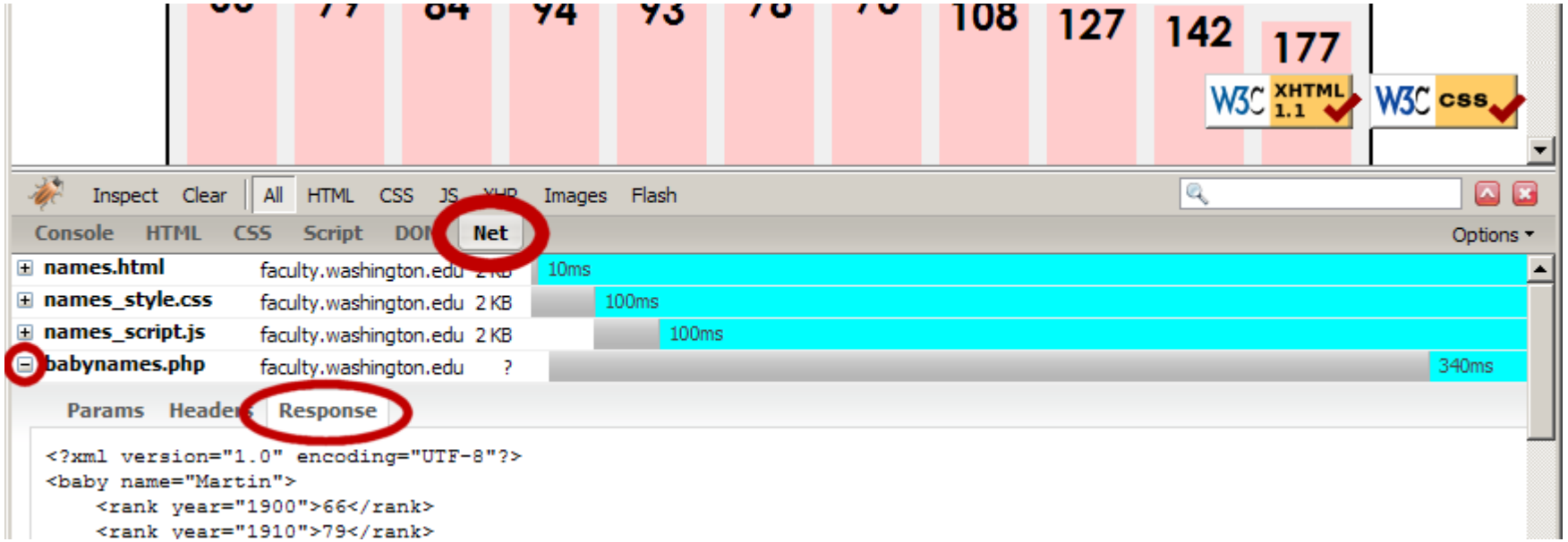
```
var ajax = new XMLHttpRequest();
...
ajax.onerror = ajaxFailure;
...

function ajaxFailure(exception) {
  alert("Error making Ajax request:" +
        "\n\nServer status:\n" + this.status + " " + this.statusText +
        "\n\nServer response text:\n" + this.responseText);
  if (exception) {
    throw exception;
  }
}
```

JS

- for user's (and developer's) benefit, show an error message if a request fails

Debugging Ajax code



- Firebug **Net** tab (or Chrome's Network tab) shows each request, parameters, response, errors
- expand a request with **+** and look at **Response** tab to see Ajax result
- check **Console** tab for any errors that are thrown by requests

Passing query parameters to a request

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("GET", "url?name1=value1&name2=value2&...", true);
ajax.send();
```
JS

- 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)*

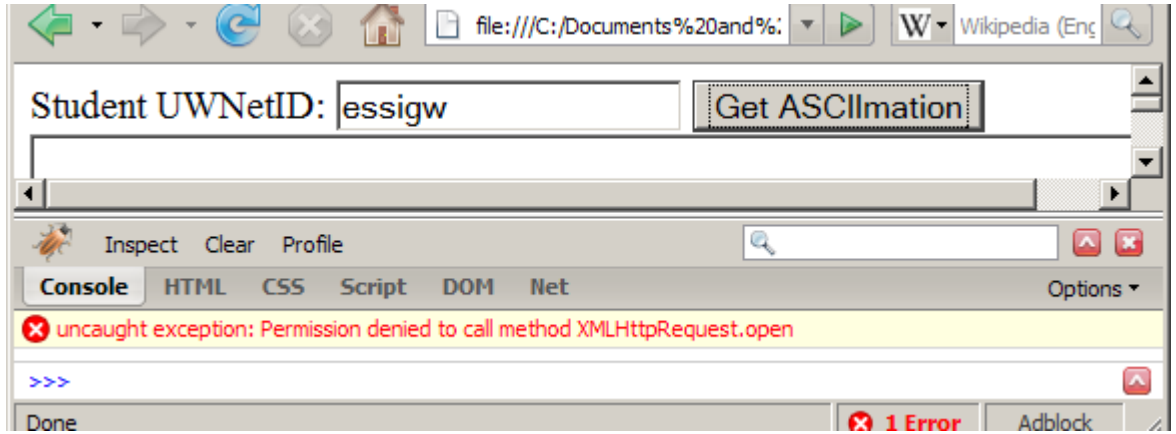
Creating a POST request

```
var params = new FormData();
params.append("name", value);
params.append("name", value);

var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("POST", "url", true);
ajax.send(params);
```
JS

- use a `FormData` object to gather your POST query parameters
- pass the `FormData` to the request's `send` method
- `method` passed to `open` should be changed to `"POST"`

XMLHttpRequest security restrictions



- Ajax must be run on a web page stored on a **web server**
  - *(cannot be run from a web page stored on your hard drive)*
- Ajax can only fetch files from the **same server** that the page is on
  - `http://www.foo.com/a/b/c.html` can only fetch from `http://www.foo.com`