Chapter 3: Web Paradigms and Interactivity

3.1 AJAX: Asynchronous Interactivity in the Web

3.2 Paradigms for Web-Based Communication

3.3 Reverse AJAX and COMET

3.4 Web Sockets and Web Messaging

3.5 Web Workers

Literature:

B. Brinzarea-Iamandi et al.: AJAX and PHP - Building Modern Web Applications, 2nd ed., Packt Publishing 2009

D. Cameron: A Software Engineer Learns HTML5, JavaScript and jQuery: A guide to standards-based web applications, 2nd ed., Cisdal Publishing 2013

Outline

*** = Nicht für Nebenfach !**

1. Introduction and Motivation

2. Interactive Web Applications
3. Web Paradigms and Interactivity *
4. Technology Evolution for Web Applications *
5. Communities, the Web, and Multimedia

Part I:
Web Technologies
for Interactive MM

6. Digital Rights - Definition and Management
7. Cryptographic Techniques
8. Multimedia Content Description

Part II:
Content-Oriented
Base Technologies

9. Electronic Books and Magazines
10. Multimedia Content Production and Distribution
11. Web Radio, Web TV and IPTV

Part IV:
Multimedia
Distribution Services

12. Multimedia Conferencing
13. Signaling Protocols for
    Multimedia Communication *

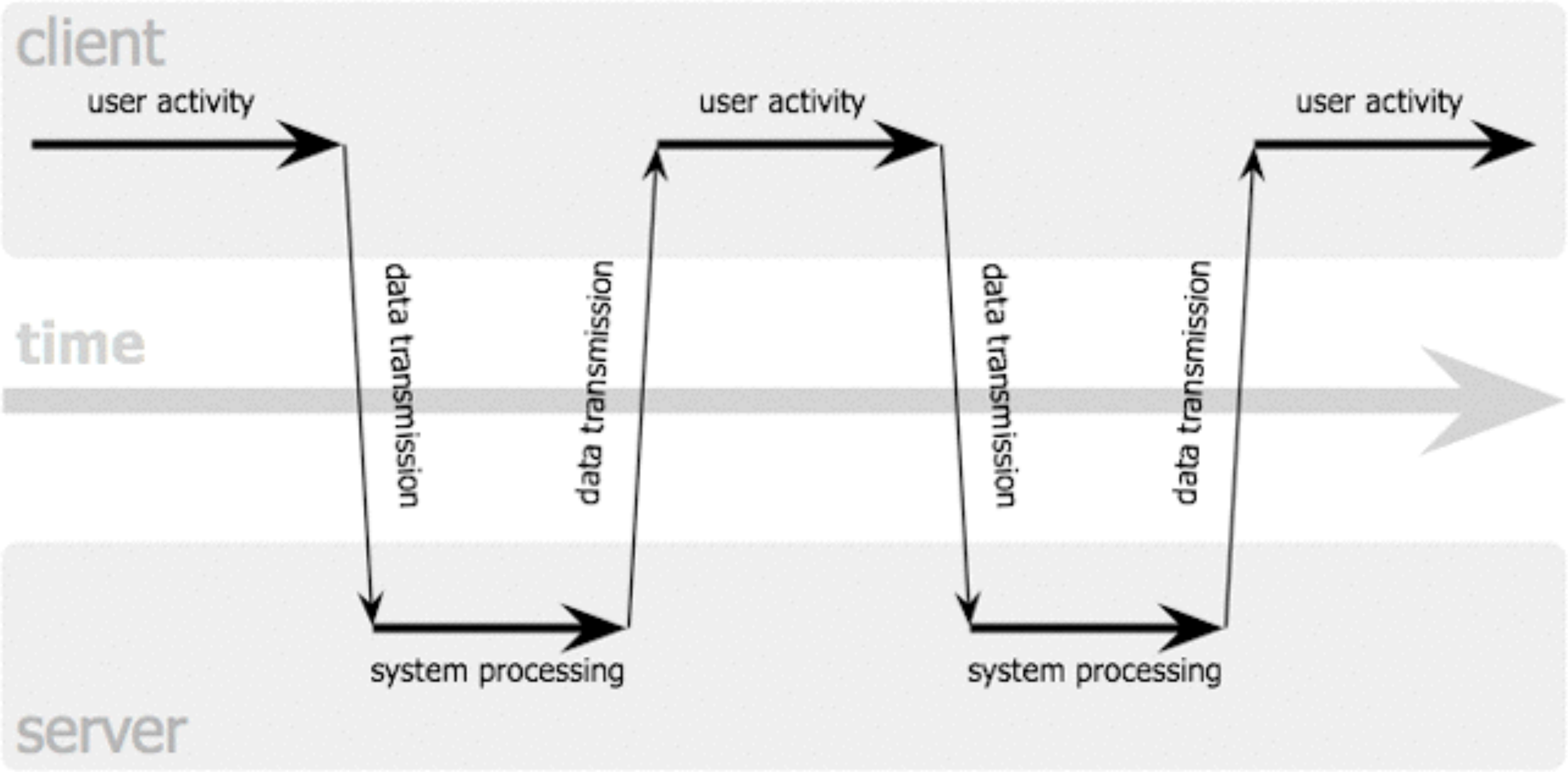
Part IV:
Conversational
Multimedia Services

14. Visions and Outlook

Asynchronous JavaScript + XML (AJAX)

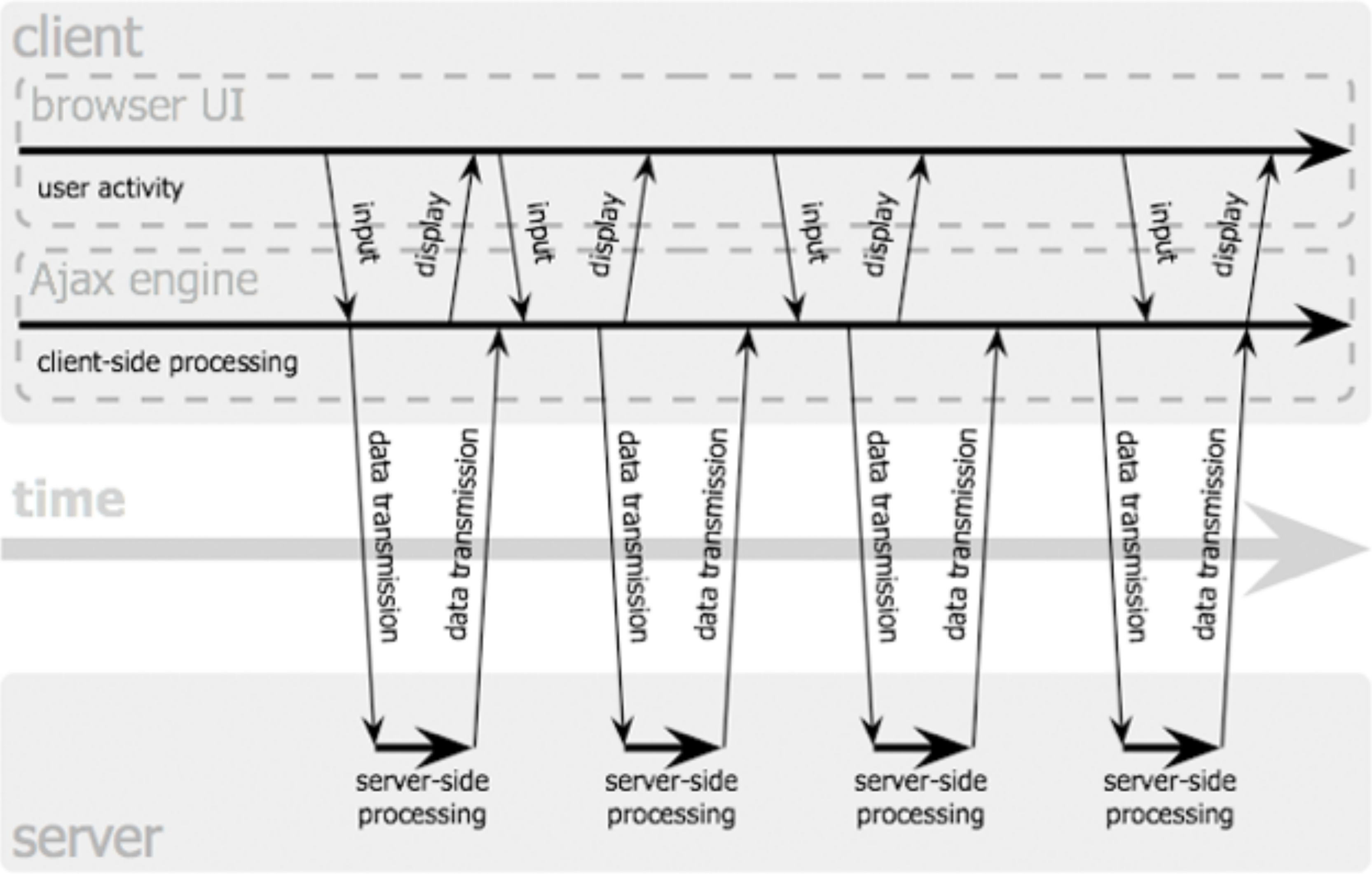
- James Garrett 2005: “Ajax: A New Approach to Web Applications”
  http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/
  - New name for an idea in use already at the time
- Decouple server communication from page reload
  - Fluid interaction
  - Presented display always stays up-to-date
- AJAX is ***not a technology!***
  - Combination of known technologies:
    XHTML, CSS, DOM, XML, XSLT, JavaScript, XMLHttpRequest
  - Idea is neither bound to JavaScript nor to XML!
  - E.g. using JSON encoding instead of XML

Classical Synchronous Web Application Model



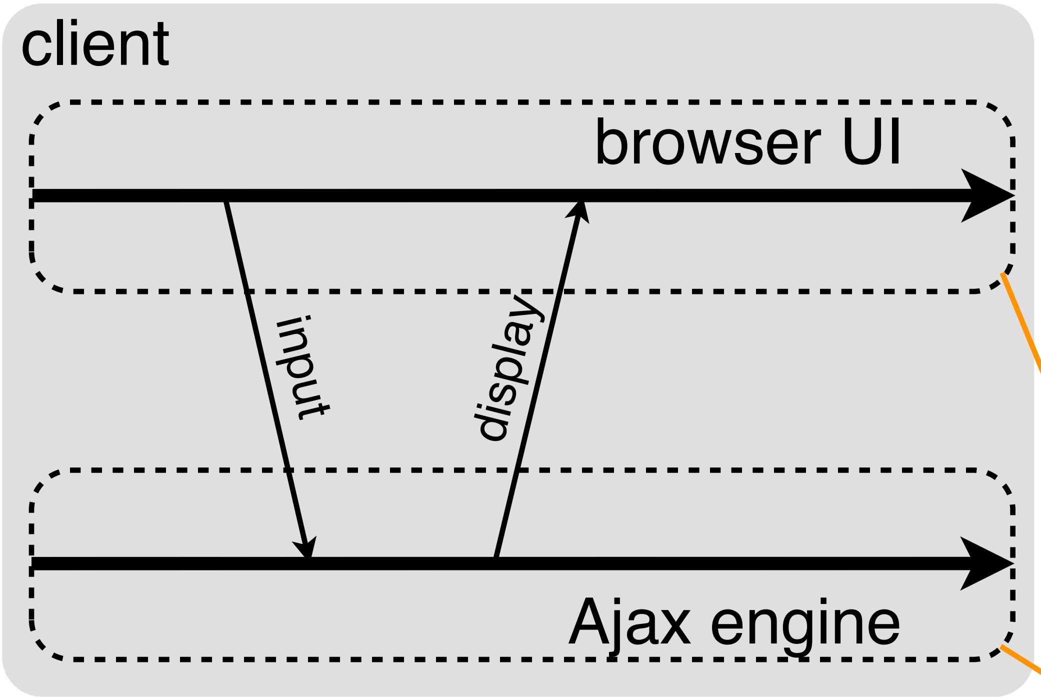
Jesse James Garrett / adaptivepath.com

Asynchronous Web Application Model



Jesse James Garrett / adaptivepath.com

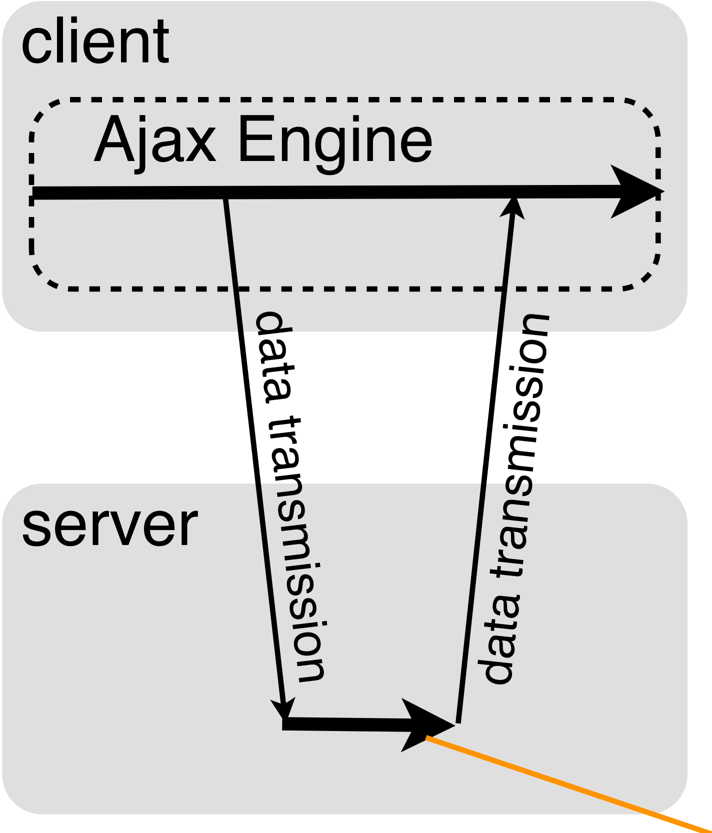
AJAX and Client-Side Scripting



- UI JavaScript
  - Access to loaded/ displayed HTML via DOM
  - Flexible input elements (HTML5)
  - Graphics (HTML5 canvas)
- Engine JavaScript
  - Event handling
- jQuery is a good fit for AJAX

Written in JavaScript

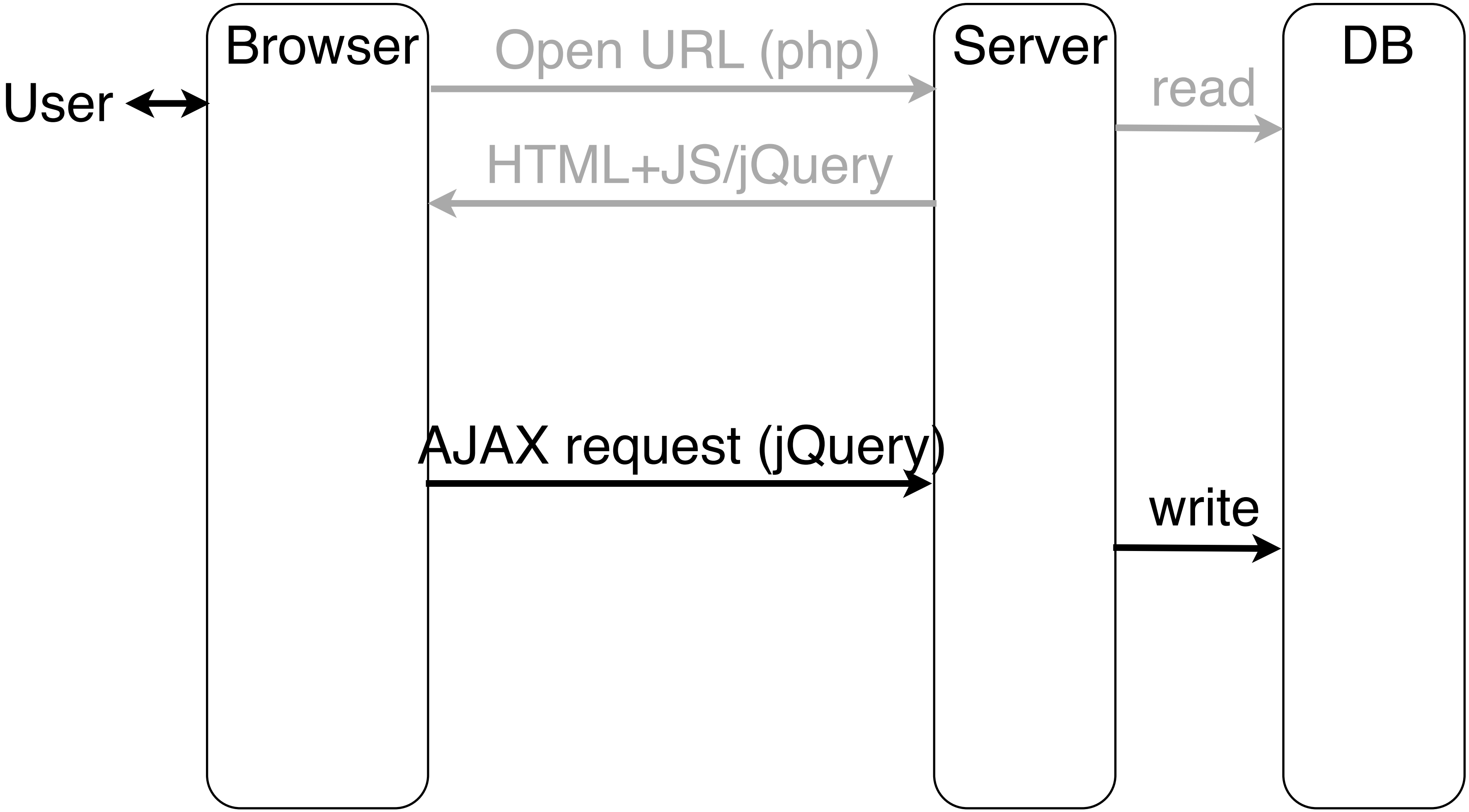
AJAX and Server-Side Scripting



- Typical examples for asynchronous server interaction:
  - Assistance in form filling (search suggestions, post or bank code decoding)
  - Real-time data (news ticker, stock prices)
  - Event notification (incoming mail, update of presence status)
  - Live chat

Any language for
server-side processing
(e.g. PHP, also JavaScript)

Example 1 (Last Lecture), Using jQuery



Recap!

Sending Request Using jQuery

```
$('#btn').click(function(){
  var selIdsTextArray = $('#mysongs input:checked').
    map(…).toArray();
  var selIdsJson = JSON.stringify(selIdsTextArray);

  $.ajax({
    type: 'POST',
    url:  'serverDummy.php',
    data: {selection: selIdsJson}
  });
});
```

dbajax_jquery.php

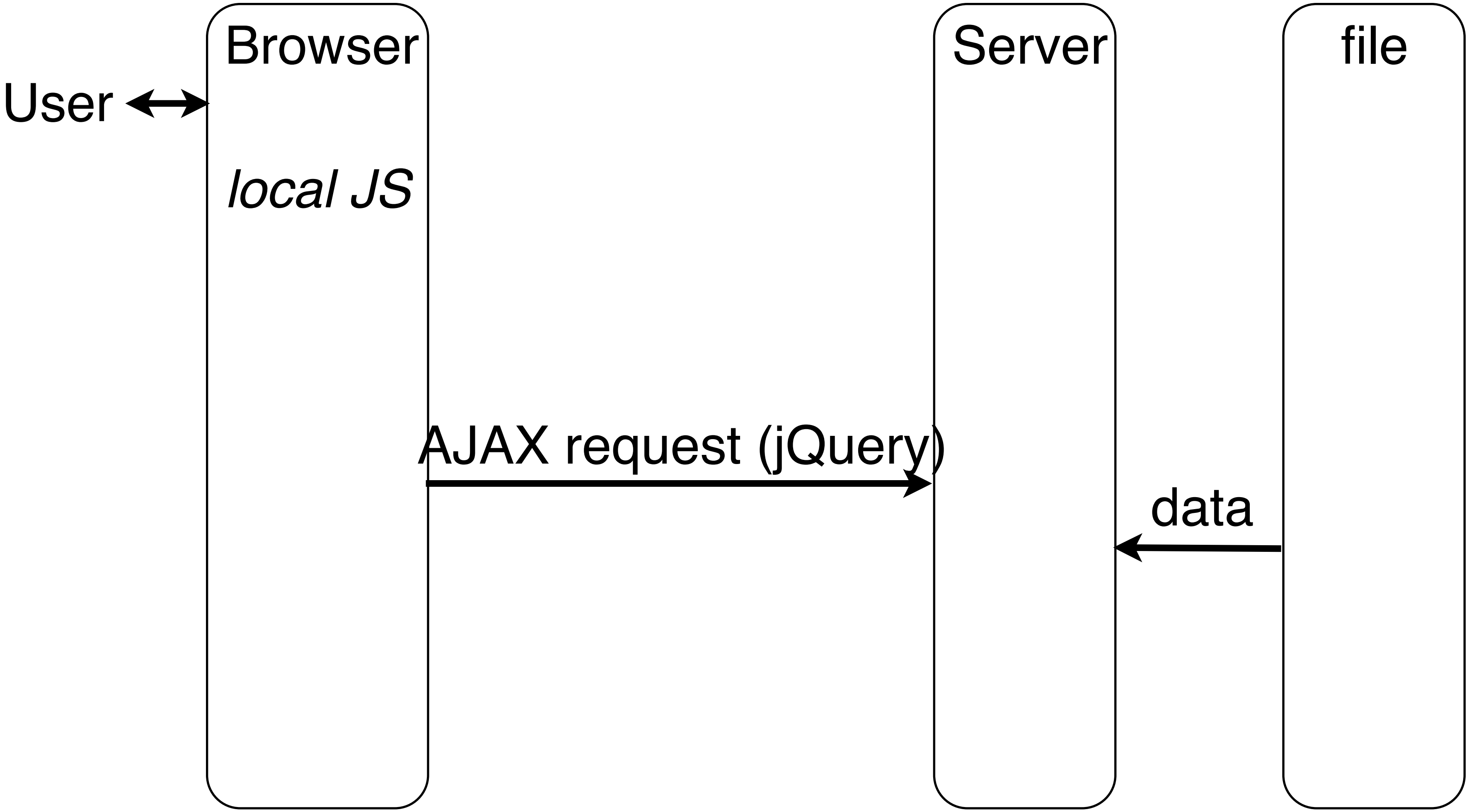
Recap!

serverDummy.php

```
<?php
   $value = $_REQUEST['selection'];
   $file = fopen("dummyData.txt", "w");
   if ($file) {
        fputs($file, "selection: " . $value . "\n");
        fclose($file);
   }
?>
```

- Of course, in a realistic setting, data received by the server is processed by operating background systems
  - Here, may want to create a table in MySQL referring to *mysongs* table

Example 2 (Very Simple Request), Using jQuery



Example 2 (Very Simple Request), Using jQuery

```
<p>The following text is replaced with data retrieved from
server (data.txt):</p>
<hr/>
<p id='text'>Text to be inserted here</p>
<hr/>
<script type='text/javascript'> …
     $(document).ready( function() {
          $.ajax({
               type: 'GET',
               url: "http://localhost/~hussmann/data.txt",
               success: function(data, status) {
                    alert("Status: "+status);
                    $('#text').html(data);
               }
          });
     });
</script>
```

Callback Function

jquery/ajaxreq_simple_txt.html

Recap!

jQuery AJAX Requests with Result

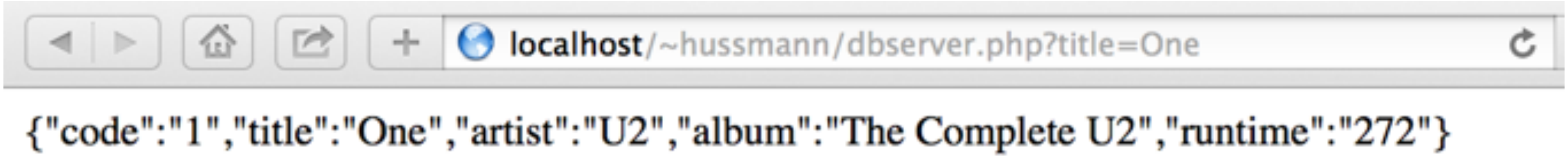
- jQuery **`ajax`** method
  - (and shorthands **`get`** and **`post`**)
  - creates a request to server
- Standard arguments, like:
  - **`url`**: URL address to send request to
  - settings: Key-value pairs (may contain JSON data)
- Example settings:
  - **`dataType`**: Kind of data expected for answer (e.g. xml, json, html)
  - **`success(data,status)`**:
    JavaScript function to be called in case of successful server response
  - **`error (requestObj, message, errorObject):`**
    JavaScript function to be called in case of server response indicating errors

Example 3 (Answered Request), DB Server

```
<?php
  $db = new mysqli('localhost','root','demopw','music');
  if ($db->connect_error) {
    die('Failed to connect: '.$db->connect_error);
  };
  $title = $_REQUEST['title'];
  $query = "SELECT * FROM mysongs WHERE title='$title'";
  $result = $db->query($query)
    or die ('Query failed'.$db->error);
  $row = $result->fetch_assoc();
  echo json_encode($row);

  $result->free();
  $db->close();
?>
```

dbserver.php



localhost/~hussmann/dbserver.php?title=One

```
{"code":"1","title":"One","artist":"U2","album":"The Complete U2","runtime":"272"}
```

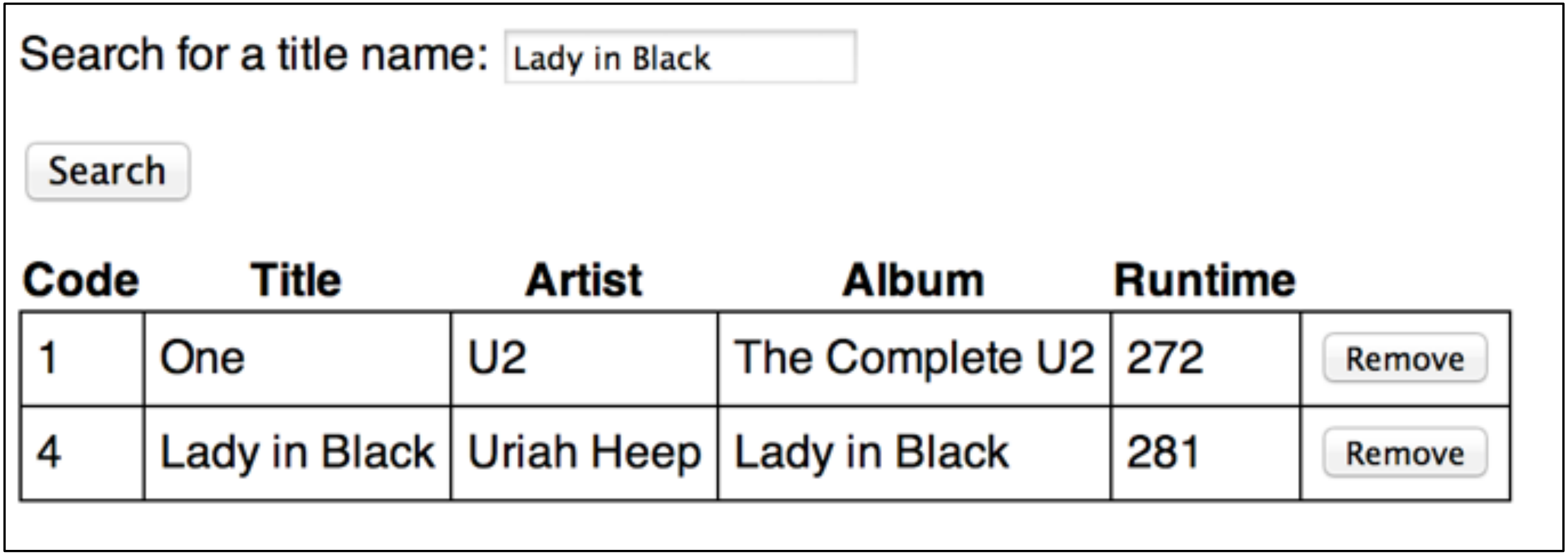
Example 3 (Answered Request), Request

```
<input id='inp_title' type='text' size='20'></input><br/>
<input id='btn' type='button' value='Search'></input>
<table id='results' class='result_displ'>
  <thead>...</thead>
  <tbody></tbody>
</table>

<script type='text/javascript'> ...
  $('#btn').click( function() {
     $.ajax({
      type: 'GET',
      url: 'http://localhost/~hussmann/dbserver.php',
      data: {title: $('#inp_title').val()},
      dataType: 'json',
      success: function(data) {
         $('#results tbody').append(
             '<tr><td>'+data.code+'</td>'+ ...</tr>'
         );
     });
   });
```

jquery/ajaxreq_result_jsn.html

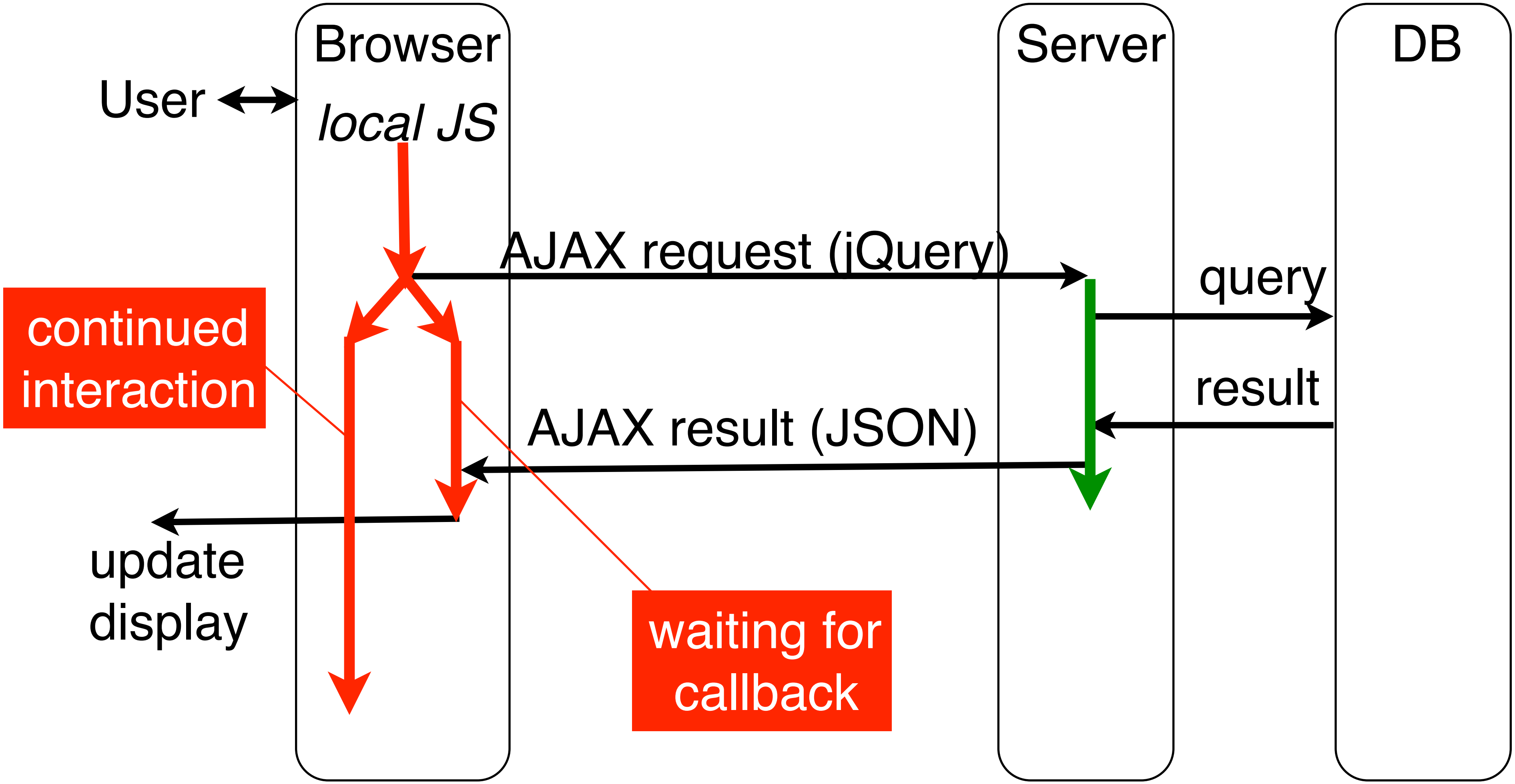
Building a List of Search Results



Search for a title name: Lady in Black

Code	Title	Artist	Album	Runtime	
1	One	U2	The Complete U2	272	Remove
4	Lady in Black	Uriah Heep	Lady in Black	281	Remove

Example 3 (Answered Request), Asynchronous!



Demonstrating Asynchronicity of Handling

- Make the database server respond slowly:

  `sleep(5);` before sending the answer

- Make the currently displayed results interactive:
  - “Remove” button in each row
  - Can be operated while waiting for server!

```
$('#results tbody').append(
  '<tr><td>'+data.code+'</td>'+ … +
  '<td><input type="button" value="Remove"></input></td></tr>'
).last().find('input').click( function() {
    $(this).parents('tr').remove();
  });
```

jquery/ajaxreq_result_jsn_slow.html

Promise Programming Style for jQuery AJAX

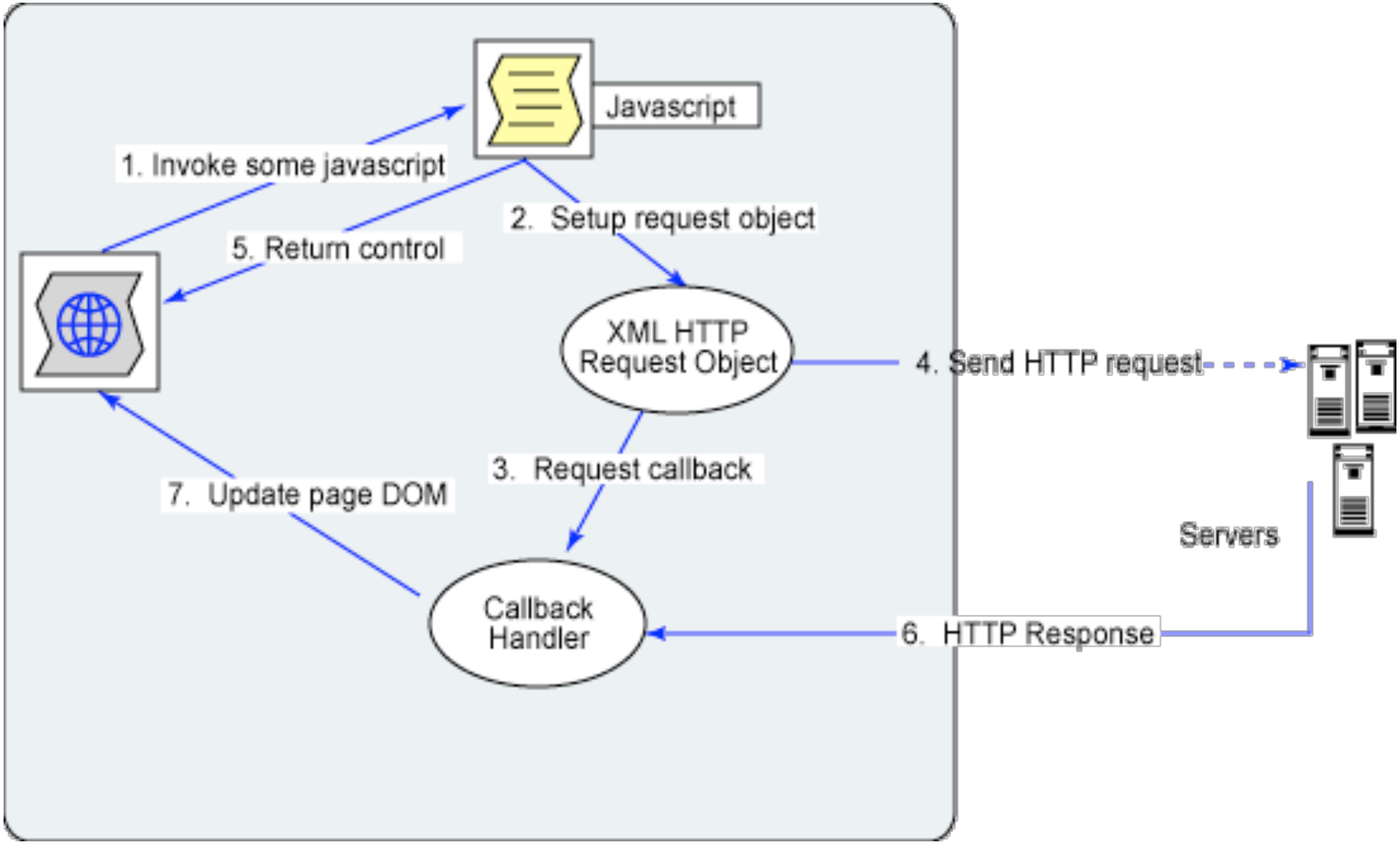
- *Promise* object:
  - Representation for the result of an asynchronous operation
  - Enables flexible management of callback functions
  - Chaining of callbacks, synchronous callbacks, conditional callbacks
- jQuery:
  - ***`Promise`*** interface
  - Result of **`$.ajax()`** implements **`Promise`** interface
- Example (D. Cameron):

```
promise = $.ajax({
  type: "GET",
  url: "/server/xyz.php",
});
promise.done(function (data) {
  console.log(data);
});
promise.fail(function () {
  console.log('A failure occurred');
});
```

AJAX Functionality (Without Using jQuery)

- Main functionalities required:
  - Construction of a request
  - Sending a request to the server
  - Waiting (asynchronously) until server responds
  - Calling functions to analyze server response
- All in one single object:
  - XMLHttpRequest

Basic Control Flow



http://www.ibm.com/developerworks, Dojo framework

XMLHttpRequest (XHR)

- Outlook Web Access for Internet Explorer 5 (end 90s):
  - XMLHTTP object invented at Microsoft
  - Realized as ActiveX object
- Nowadays in all modern browsers
  - XMLHttpRequest object
  - Plain JavaScript, including Internet Explorer >7
- Under W3C standardization until 2012, then moved to WHATWG
  - Currently "living standard" at https://xhr.spec.whatwg.org
  - Last "snapshot" of living standard at W3C: 26 May 2014

```
var XMLHTTP = new XMLHttpRequest();
```

- Historic browser incompatibilities have to be handled
  - Built into frameworks like *Prototype* or *jQuery*

Construction of an HTTP Request

- **`open()`** method of XMLHttpRequest object
  - Note: No interaction with the server yet!
- Required parameters:
  - HTTP method: GET, POST or HEAD
  - URL to send the request to
- Optional parameters:
  - Boolean indication whether to use asynchronous or synchronous treatment (default asynchronous = true)
  - Username and password for authentication
- Examples:

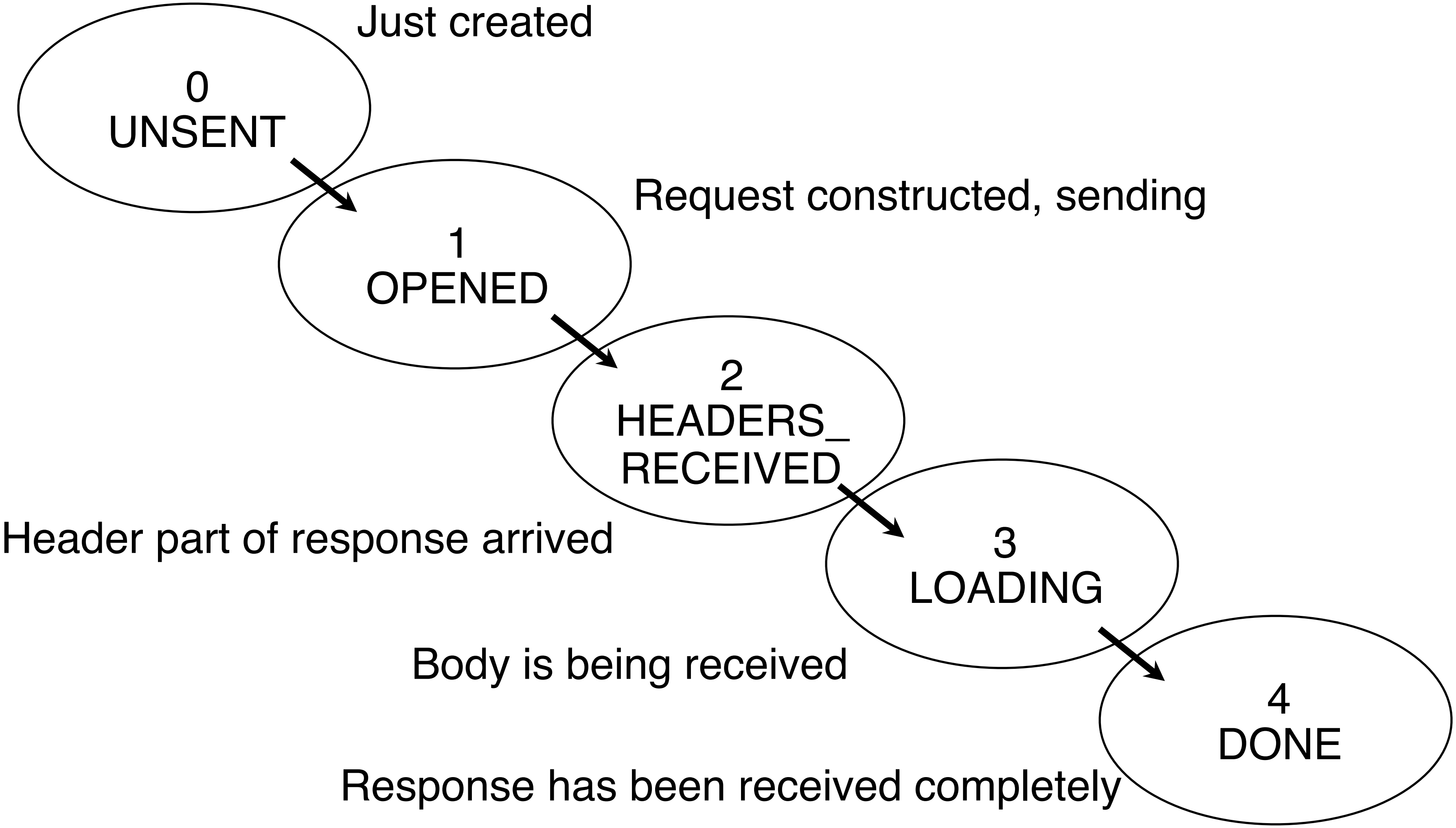
```
XMLHTTP.open("GET", "fibonacci.php?fib=12")
XMLHTTP.open("POST", "/start.html", false, un, pwd);
```

Sending a Request

- Before sending: **`XMLHTTP.setRequestHeader()`**
  - Setting headers for the request
  - Needed for POST method: **`Content-Type`** (MIME type)

- **`XMLHTTP.send()`**
  - Sends request to server
- Parameter:
  - In the simplest case (in particular for GET method): **`null`**
  - For POST method: "Request entity body" given as parameter

States of an XMLHttpRequest Object



Just created

0
UNSENT

1
OPENED

Request constructed, sending

2
HEADERS_
RECEIVED

Header part of response arrived

3
LOADING

Body is being received

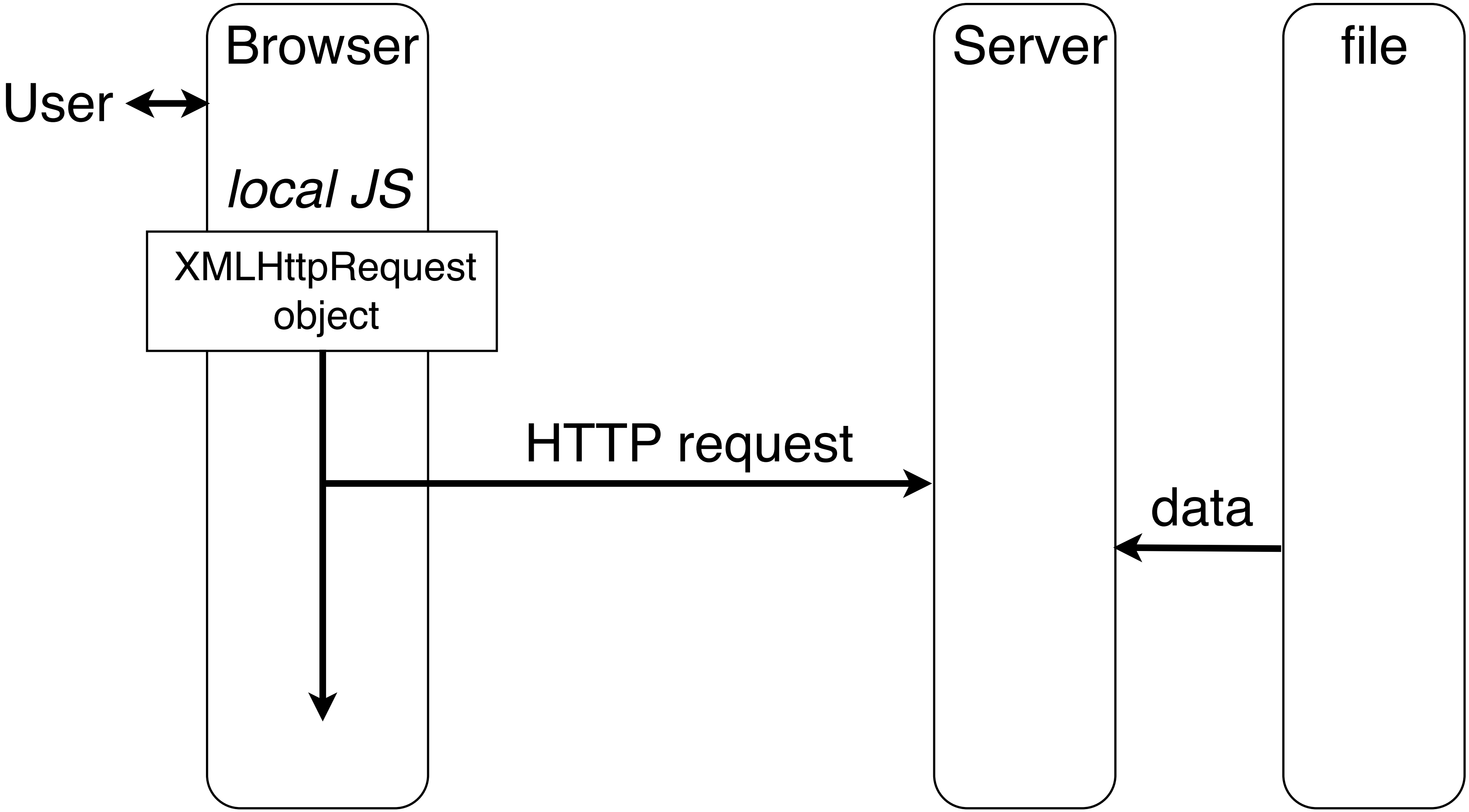
4
DONE

Response has been received completely

Asynchronous Reaction by Event Handler

- Registering an event handler for **`XMLHttpRequest`** state changes
  - Event **`readystatechange`** of **`XMLHttpRequest`** object
  - Callback function, called at *any* state change:
    - » `XMLHTTP.addEventListener("readystatechange", function);`
- Testing for the relevant state change:
  - **`readystate`** attribute of **`XMLHttpRequest`** object gives current state (as number)
- Other attributes:
  - **`status`** gives return code
  - **`statusText`** gives associated text
  - **`responseText`** and **`responseXml`** give response content

Example 2 (Very Simple Request), Without jQuery

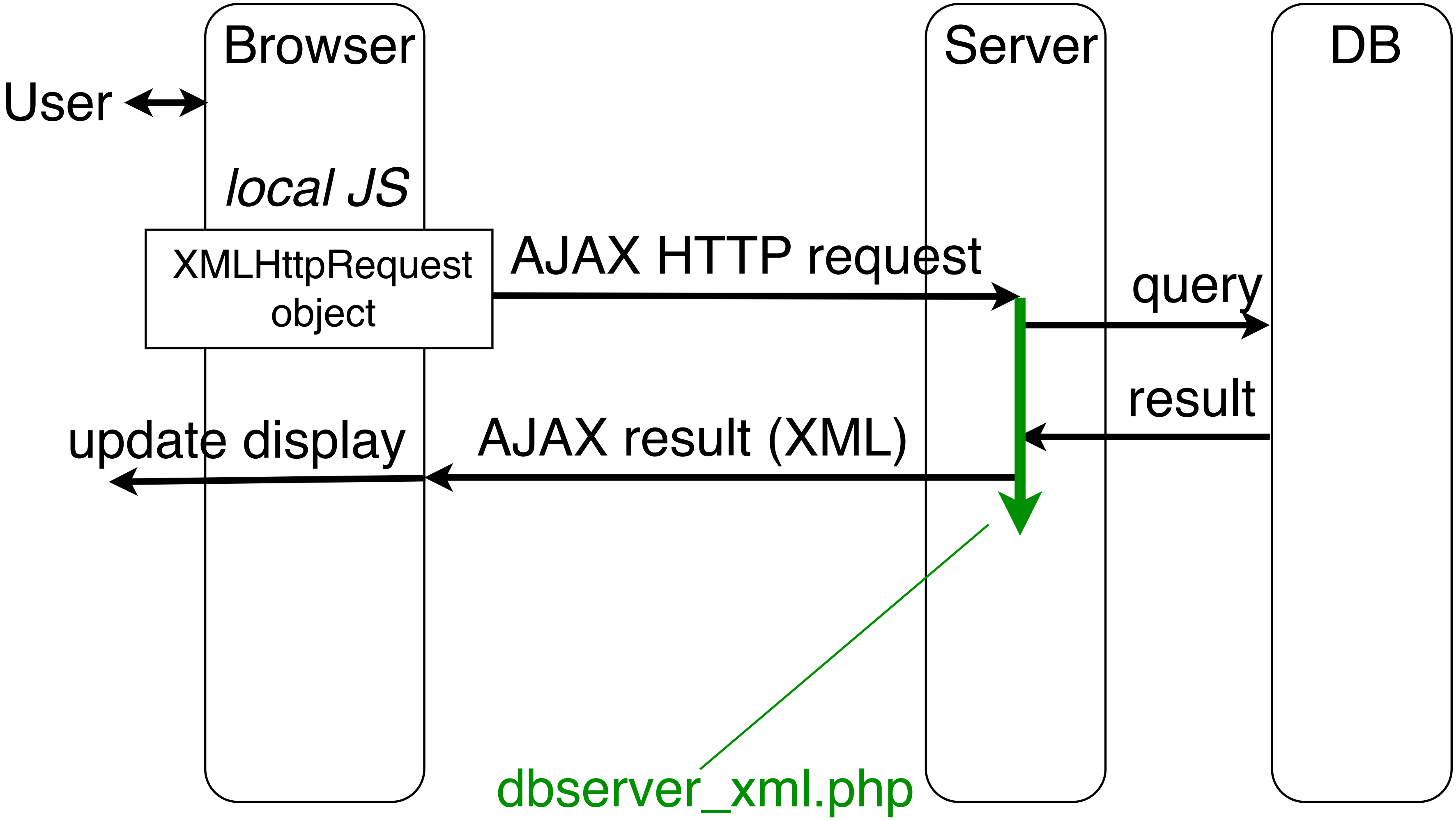


Example 2 (Very Simple Request)

```
<body>
  <p>The following text is replaced with data retrieved from
  server (data.txt):</p>
  <hr/>
  <p id='text'>Text to be inserted here</p>
  <hr/>
…
  <script type = "text/javascript">
    var XMLHTTP = new XMLHttpRequest();
    document.addEventListener("DOMContentLoaded",function() {
      XMLHTTP.open("GET",
         "http://localhost/~hussmann/data.txt", true);
      XMLHTTP.addEventListener("readystatechange",function() {
        if (XMLHTTP.readyState == 4) {
          alert("Status: "+XMLHTTP.statusText);
          var d = document.getElementById("text");
          d.innerHTML = XMLHTTP.responseText;
        }
      }, false);
      XMLHTTP.send(null);
     }, false);
  </script>
</body>
```

ajax/simplerequest.html

Example 3 (Answered Request), Without jQuery



Example 3 (Simplified) with Pure AJAX

```php
<?php
        header("Content-type: text/xml");

        $db = new mysqli('localhost','root','demopw','music');
        $title = $_REQUEST['title'];
        $query = "SELECT * FROM mysongs WHERE title='$title'";

        $xml = "<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>\n";
        $xml .= "<songs>\n";
        while ($row = $result->fetch_assoc()) {
                $xml .= "\t\t<song>\n";
                foreach ($row as $tag => $value) {
                        $xml .= "\t\t\t<" . $tag . ">\n";
                        $xml .= "\t\t\t\t" . $value . "\n";
                        $xml .= "\t\t\t</" . $tag . ">\n";
                };
                $xml .= "\t\t</song>\n";
        };
        $xml .= "</songs>\n";

        echo $xml;

        $result->free();
        $db->close();

?>
```

PHP server
(accessing database),
returning XML Text

php/dbserver_xml.php

Example Server Output (XML) for Example 3

Request

```
GET /~hussmann/dbserver_xml.php
      ?title=One HTTP/1.1
Host: localhost:80
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 29 Oct 2014 19:29:41 GMT
Server: Apache/2.2.26 (Unix) DAV/2 PHP/5.4.30
X-Powered-By: PHP/5.4.30
Content-Length: 248
Content-Type: text/xml

<?xml version="1.0" encoding="iso-8859-1"?>
<songs>
      <song>
         <code>
            1
         </code>
         <title>
            One
         </title>
         <artist>
            U2
         </artist>
         <album>
            The Complete U2
         </album>
         <runtime>
            272
         </runtime>
      </song>
</songs>
```

Example 3 with Pure AJAX – HTML

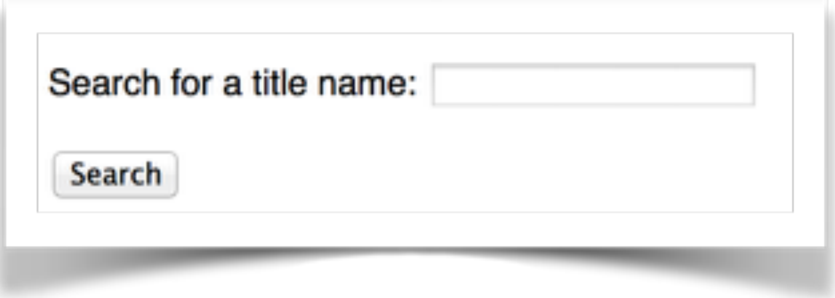
```
<html>
   <head>
      <title>Pure Ajax Request with XML encoded result</title>
      <style>…</style>
   </head>

   <body id="bodytext">
      <p>
         Search for a title name:
         <input id="inp_title" type="text" size="20"></input><br/>
      </p>
      <p>
         <input id="btn" type="button" value="Search"></input>
      </p>

   </body>

   <script type = "text/javascript">
      JavaScript code
   </script>

</html>
```



Search for a title name:

ajax/req_result_XML.html

Example 3 with Pure AJAX – HTTP Request

```
var XMLHTTP = new XMLHttpRequest();
var btn = document.getElementById("btn"); //not needed
var inp_title = document.getElementById("inp_title"); //not needed
var bodytext = document.getElementById("bodytext"); // not needed

btn.addEventListener("click", function() {
   XMLHTTP.open("GET",
      "http://localhost/~hussmann/dbserver_xml.php?title="
      +inp_title.value);
   XMLHTTP.send(null);
}, false);

XMLHTTP.addEventListener("readystatechange", function() {
   if (XMLHTTP.readyState == 4) {
      DOM JavaScript code
      }
   }
}, false);
```

ajax/req_result_XML.html

Example 3 with Pure AJAX – DOM JavaScript

```
var xml = XMLHTTP.responseXML;
var songs = xml.getElementsByTagName("song");
if (songs.length > 0) {
   var artist = songs[0].
      getElementsByTagName("artist")[0].firstChild.nodeValue;
   var album = songs[0].
      getElementsByTagName("album")[0].firstChild.nodeValue;
   var line = document.createElement("p");
   var text = document.createTextNode(
     "Artist: "+artist+"; "+"Album: "+album);
   line.appendChild(text);
   bodytext.appendChild(line);
}
```

Read XML (tree)

Modify/write HTML (tree)

ajax/req_result_XML.html

AJAX: Potential and Problems

- Potential:
  - Reaction to ***any*** user action (e.g. mouse move, typing)
  - Enables classic GUIs for "Web Apps"
- Problems:
  - Back button
  - Bookmarks
  - Search engines

Chapter 3: Web Paradigms and Interactivity

3.1 AJAX: Asynchronous Interactivity in the Web

3.2 Paradigms for Web-Based Communication

3.3 Reverse AJAX and COMET

3.4 Web Sockets and Web Messaging

3.5 Web Workers

Basic Web Paradigms: Documents

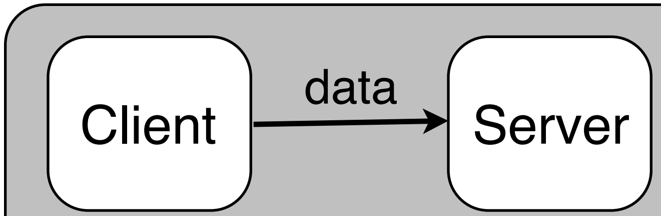
- HTML:
  - Originally intended for scientific papers: Limited structure
  - Purely static
  - Not object-oriented
- HTML5:
  - More flexible structure, graphics output, input elements, media playback
- DOM:
  - Dynamic changes of documents
- CSS:
  - Separation content/presentation, presentation classes
- JavaScript:
  - Dynamic changes, object-orientation

Basic Web Paradigms: Communication

- HTTP:
  - Request-response architecture:
    - » Requests have to be initiated by client
  - Restricted parameter syntax (keyword-value pairs)
  - Synchronicity: Client has to wait for response
- AJAX:
  - Enables asynchronous handling of requests in client
- Basic restriction to request ➔ response remains!
  - “Client-driven” architecture

Types of Client-Server Interaction

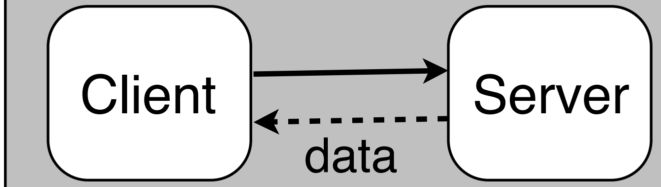
Client-driven



Send data to server

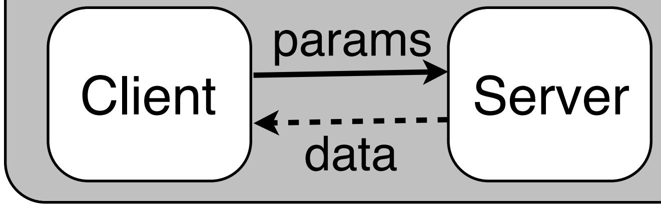
Example 1: Sending shopping cart contents

Other examples: Location update, logging



Pull data from server

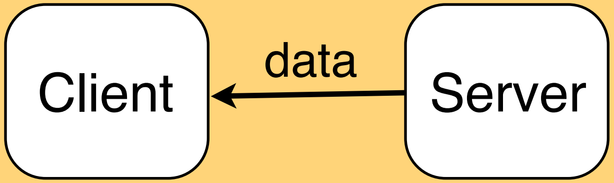
Example 2: Very simple request



Pull selected data from server

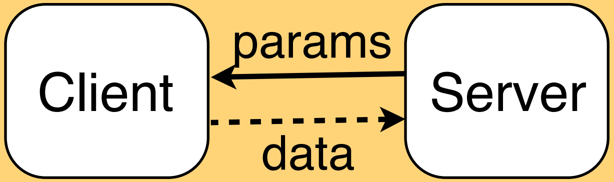
Example 3: Database query

Manifold other examples



Push data from server to client

Examples: New mail, breaking news, chat



Request data from client

Examples: Status inquiry, security check

Server-driven

Server-Driven Applications in the Web

- Frequent and easy solution: ***Polling***
  - Client sends requests to server in regular intervals
- Disadvantages:
  - Redundant load to client, server, network
  - Changes traffic characteristics
  - Limited time resolution for real-time events
- Alternatives:
  - (a) “Reverse AJAX”/”COMET” – Tricking the Web architecture
  - (b) Going beyond traditional HTTP

Chapter 3: Web Paradigms and Interactivity

3.1 AJAX: Asynchronous Interactivity in the Web

3.2 Paradigms for Web-Based Communication

3.3 Reverse AJAX and COMET

3.4 Web Sockets and Web Messaging

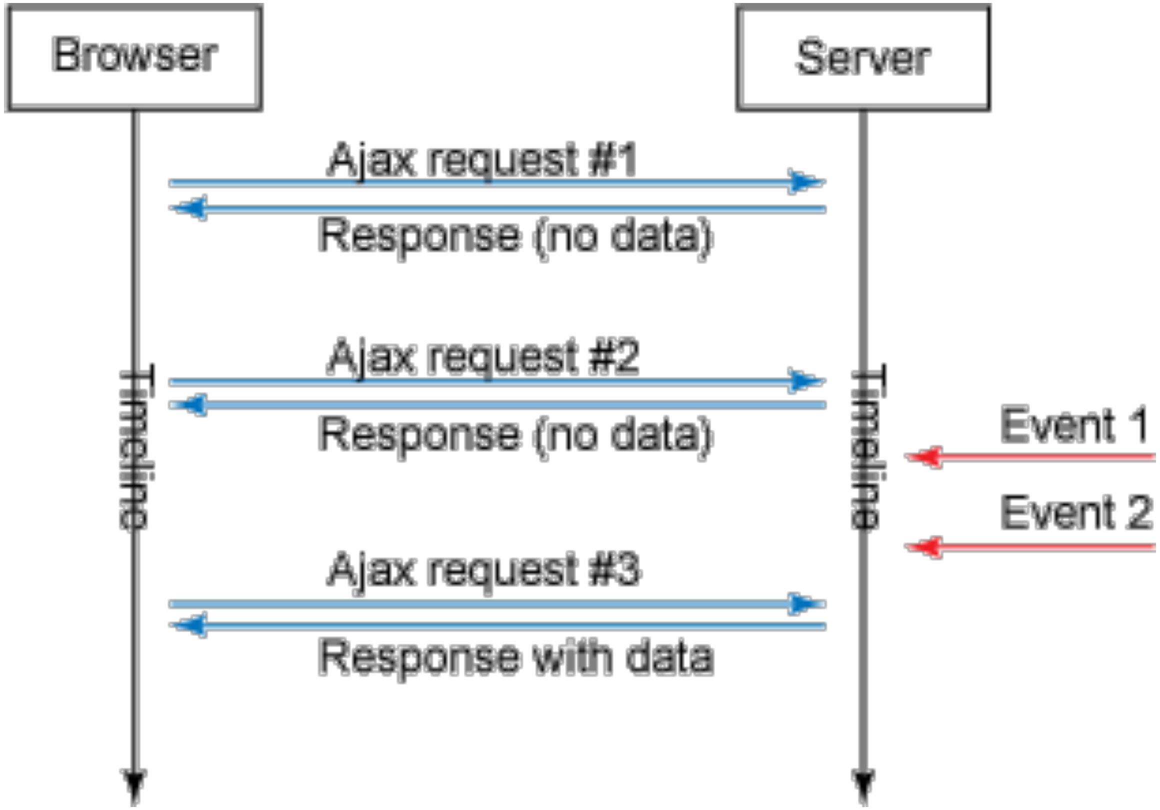
3.5 Web Workers

Literature:

Mathieu Carbou: Reverse Ajax, Part 1: Introduction to Comet, http://www.ibm.com/developerworks/web/library/wa-reverseajax1/

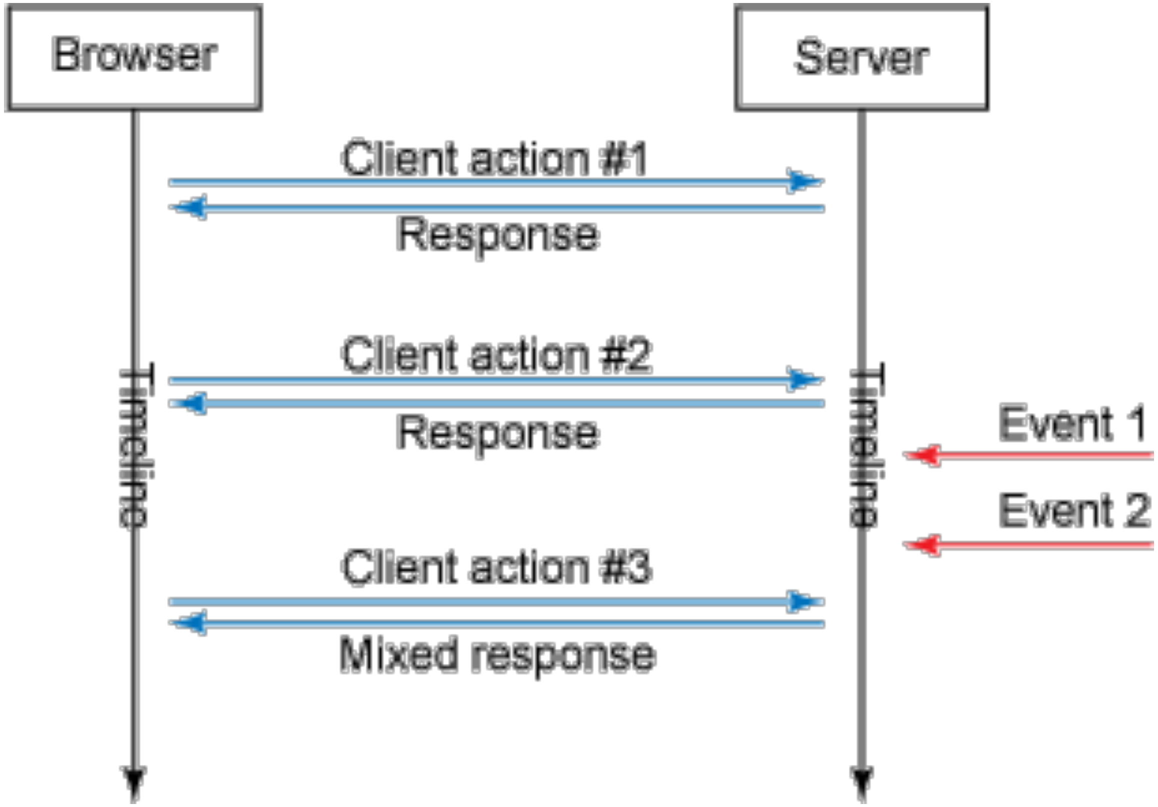
D. Cameron: A Software Engineer Learns HTML5, JavaScript and jQuery: A guide to standards-based web applications, 2nd ed., Cisdal Publishing 2013

Reverse Ajax with HTTP Polling



- Server event information pulled by client through regular polling
- Easily realizable in JavaScript using “setInterval()”
- High network load, imprecise timing

Reverse Ajax with Piggyback Polling

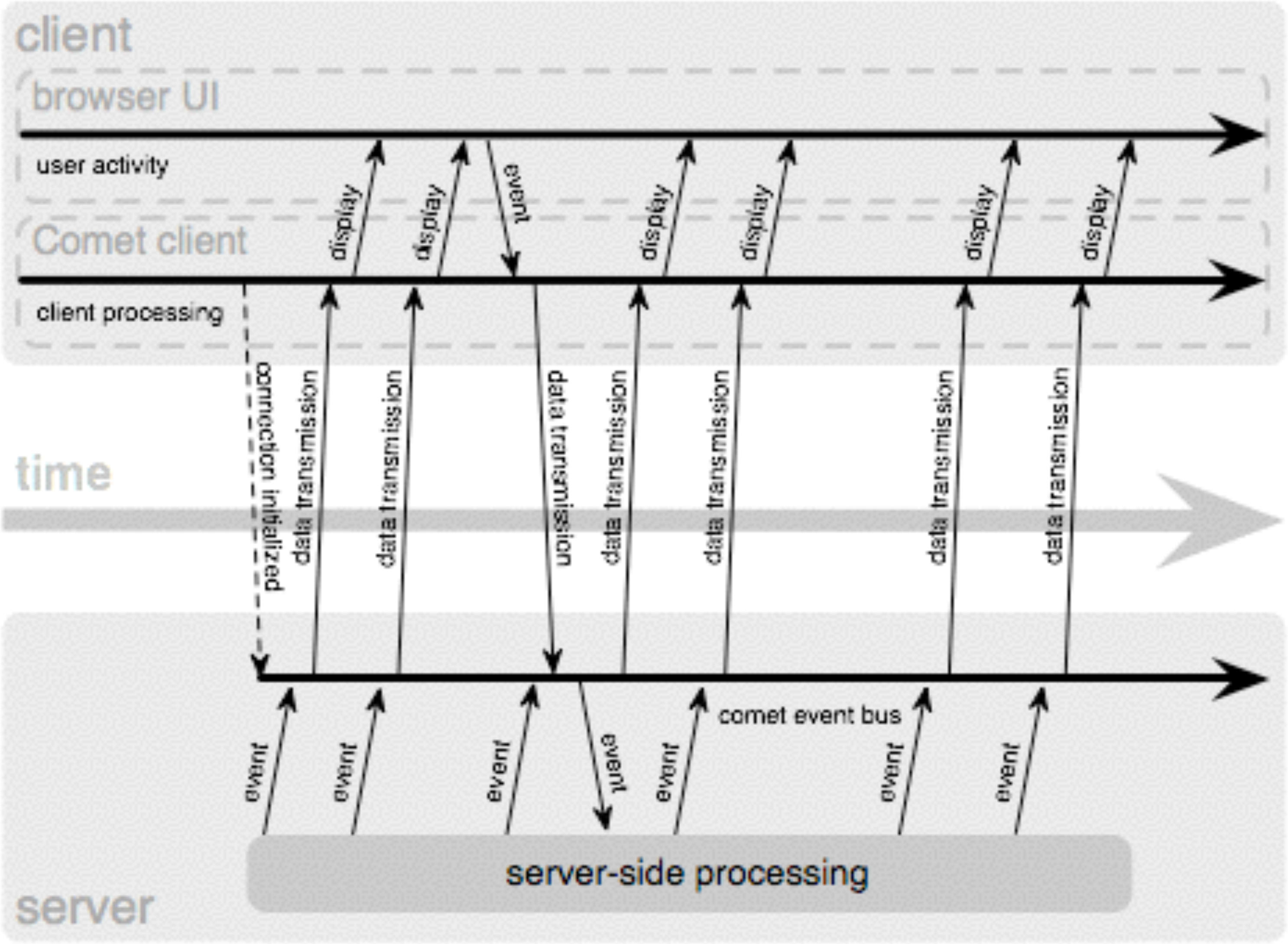


- Assuming different needs for information exchange between client and server
- Whenever a client-triggered request is processed, additional information about latest server-side events is added to the response

Reverse Ajax with the Comet Model

- Proper support for asynchronous server-side events:
  - Availability of a channel for the server to push information to the
  - Server-client connections maintained over a long period of time
- Alex Russell 2006 (Blog)
  http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/
  - Web Applications exist which use server-side events and long-l server connections (Gmail GTalk, Meebo)
  - "Lacking a better term, I've taken to calling this style of event-dr push data streaming "Comet". It doesn't stand for anything, and that it should."
  - Other terms for the same idea: Ajax Push, HTTP Streaming, HT push
    - » Sometimes also Reverse Ajax...

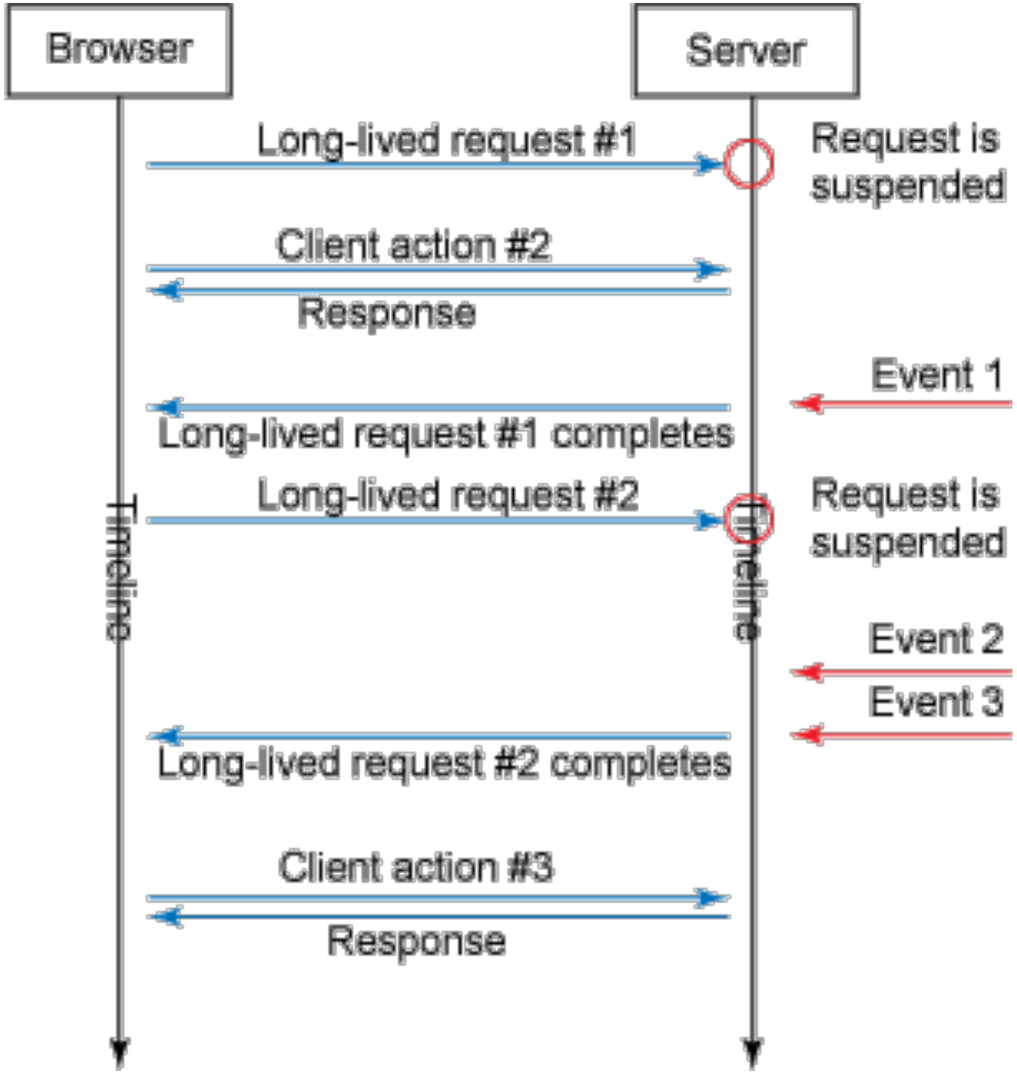
Comet Web Application Model



Connection Management in Comet

- Comet based on *HTTP Streaming*:
  - Single TCP/IP connection kept open between client and server
  - For instance using the “multipart response” supported by many browsers
    - Origin: “server push” feature by Netscape in 1995,
      e.g. to send new versions of an image by the server
    - Response is “stretched over time”
- Comet based on *Long Polling*:
  - Standard XMLHttpRequest sent by client
  - Server suspends response until event happens
    - Specific programming techniques on server required
    - Storing the request context
  - As soon as client receives response (and processes it), client sends new request (which is suspended again)
  - Relatively easy to realize with current browsers and XMLHttpRequest

Reverse Ajax with Comet



- Client request is suspended at server
- Server responds to the request each time a new server-side event happens

Server-Sent Events API (SSE)

- API for sending events and data from server to client
  - Standardized by WHATWG (Communication APIs for HTML)
  - Supported by many browsers, currently not by IE
  - Uses standard HTTP protocol
- Server side:
  - Response with content type **`text/event-stream`**
  - Body lines start with **`data:`**, **`event:`**, or **`id:`**
  - Server easily constructible with e.g. PHP or Node.js
- Client side (example):

```
var source = new EventSource('updates.cgi');
source.onmessage = function (event) {
  alert(event.data);
};
```

- See e.g. http://www.html5rocks.com/en/tutorials/eventsource/basics/

Chapter 3: Web Paradigms and Interactivity

3.1 AJAX: Asynchronous Interactivity in the Web

3.2 Paradigms for Web-Based Communication

3.3 Reverse AJAX and COMET

3.4 Web Sockets and Web Messaging

3.5 Web Workers

Literature:

Mathieu Carbou: Reverse Ajax, Part 2: Web Sockets, http://www.ibm.com/developerworks/web/library/wa-reverseajax2/

http://websocket.org

General Idea and General Problem

- Idea:
  - Web client (browser) communicates
    at the *same time* and
    in the *same data space* with
    several different hosts
- Security problem: “Cross-site scripting”
  - Web application A gets access to data from Web application B
  - In the worst case including authentication data
- Current principle in browsers:
  - Only one Web application at a time communicates with a browser instance
  - Being relaxed in new approaches (under security precautions)

WebSockets

- Originated in HTML5 (WHAT Working Group)
  - HTML5 Web Sockets specification
  - Full-duplex communication channel between client and server
  - Establishment ("handshake") client-initiated, over HTTP
  - One connection for bi-directional communication, very small latency
    - "sub 500 millisecond" latency
    - Near real-time!
  - Able to traverse firewalls and proxies (port 80)
  - Secure connection can be used (HTTP/S)
- WebSockets have been separated out of HTML5
  - API developed by W3C, protocol ("ws:") standardized as IETF RFC 6455
  - Browser support:
    - Earlier unsecure version disabled
    - RFC 6455 supported in all modern major browsers

WebSocket Client API (JavaScript)

- Connect to an endpoint (WebSocket handshake):

```
var myWebSocket =
  new WebSocket("ws://www.websockets.org");
```

- Associate event handlers to established connection:

```
myWebSocket.addEventListener("open", function);
myWebSocket.addEventListener("message", function);
myWebSocket.addEventListener("close", function);
```

- Send message to server over established connection:

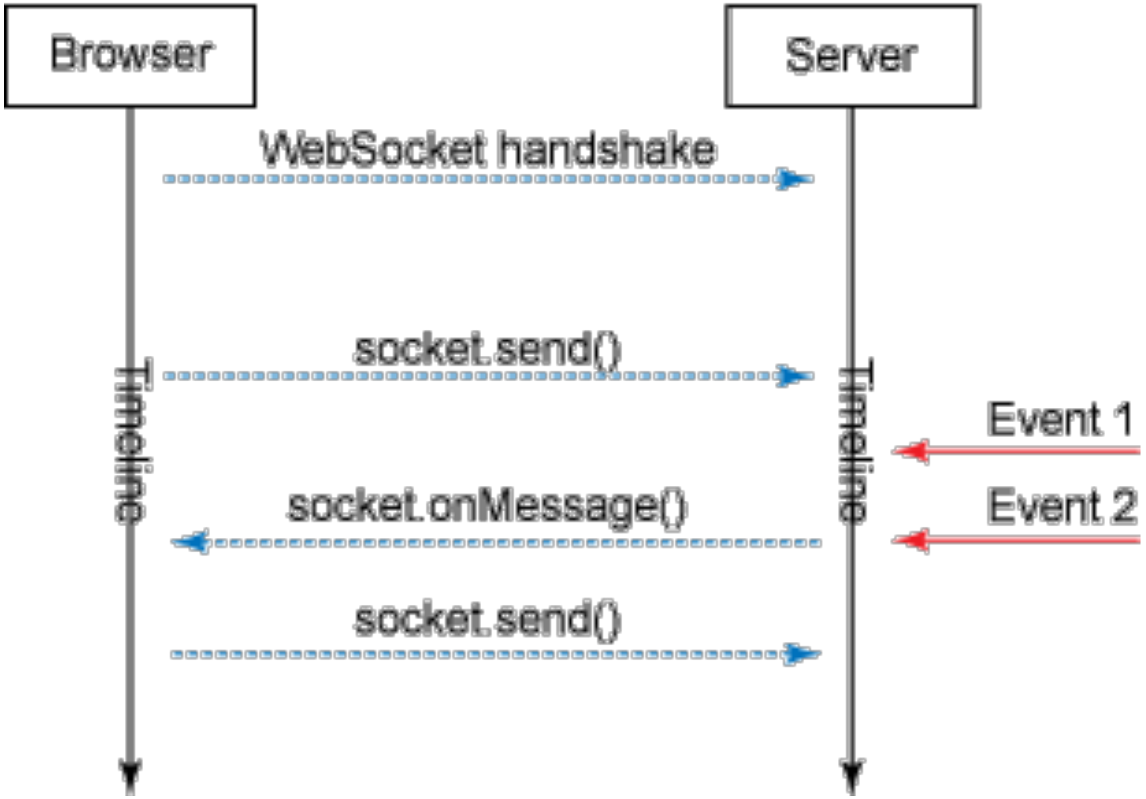
```
myWebSocket.send("hello");
```

- Disconnect from endpoint:

```
myWebSocket.close();
```

- Demos: https://www.websocket.org/echo.html

Reverse Ajax with WebSockets



- Simple, low-latency solution
- New standard, not yet widely used – probably the way to go in future
- *Abstraction APIs* help to keep programs independent of transport
  - See e.g. socket.IO

Web Messaging

- HTML5 Web Messaging
  - Standardized by W3C, driven by Google
  - Candidate recommendation May 01, 2012
- Document A, if knowing about another document B, can send a (text) message to document B (on a different domain)
- Specific *iframe* in document A calls **`postMessage()`** referring to domain and window of document B.
- Document B can handle the event in event handler
  - Gets information about origin, ***which needs to be checked***
  - Document B checks format of message and takes additional precautions
- Simple to use, high security risks

Chapter 3: Web Paradigms and Interactivity

3.1 AJAX: Asynchronous Interactivity in the Web

3.2 Paradigms for Web-Based Communication

3.3 Reverse AJAX and COMET

3.4 Web Sockets and Web Messaging

3.5 Web Workers

Literature:

B. Lawson, R. Sharp: Introducing HTML5, New Riders 2011

Threading in Web Browsers

- Thread = Sequence of instructions to be executed
- Traditionally, Web browsing is *single-threaded*
- Complex Web applications (and multimedia) require *multi-threading*
  - Example: Asynchronous interaction in Ajax and Reverse Ajax
  - Example: Playing back a movie/sound, being still able to control it
  - Example: Synchronizing a movie with subtitles or animations
  - Example: Long loading time for multimedia document – user has decided to do something else
  - Example: Independent animations on a single page (content and advertisement)
- Web Worker:
  - Specification for light-weight JavaScript threads in browsers
  - Originated by WHATWG, now separated from HTML5
  - Supported e.g. in Safari, Chrome, Opera and Firefox

Principles for Using Web Workers

- Creating a new worker:
  - **`var worker = new Worker("my_worker.js");`**
- Sending a message to the worker:
  - **`worker.postMessage("hello worker");`**
- Receiving a message from the worker:
  - **`worker.addEventListener("Message",`** ***`function`*****`, false);`**
  - ***`function`*** **`(event) { … event.data … }`**
- What a worker can do:
  - Communicate, including Web Messaging and Web Sockets
  - Send and process Ajax requests
  - Establish timers
  - Basic JavaScript (but ***no*** DOM access)
  - Web SQL databases
  - Web Workers (!)
- Shared Worker: Working with multiple documents