



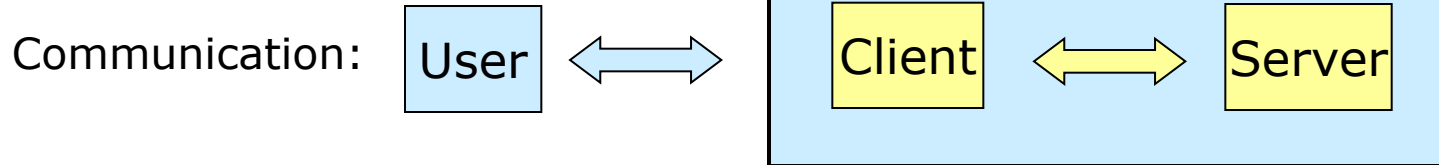
TECHNISCHE  
UNIVERSITÄT  
DRESDEN

Department of Computer Science Institute for System Architecture, Chair for Computer Networks

# Interaction models in the World Wide Web

# Outline

What is interaction?



2 steps of interaction: communication between user and application  
communication between client and server

## 1. Synchronous communication model:

- Classic web application model
  - “Click, wait, and refresh user interaction paradigm”

## 2. Non blocking communication model:

- A new approach to web applications
  - AJAX (Asynchronous JavaScript + XML) web application model

## 3. Asynchronous communication model:

- Comet via long-polling approach
- WebSockets ← Part of HTML5

# Communication models

Communication models	Criteria	
	Method of sending data with respect to user actions	Degree of dependence between HTTP Request and HTTP Response
Synchronous model -> Click, wait, and refresh user interaction paradigm	<b>synchronous</b> (user activities are blocked)	<b>high</b>
Non blocking model -> AJAX	<b>asynchronous</b> (user activities are not blocked)	<b>high</b>
Asynchronous model -> Comet via long-polling, WebSockets	<b>asynchronous</b>	<b>minimum</b>

# Synchronous communication model

## Criteria:

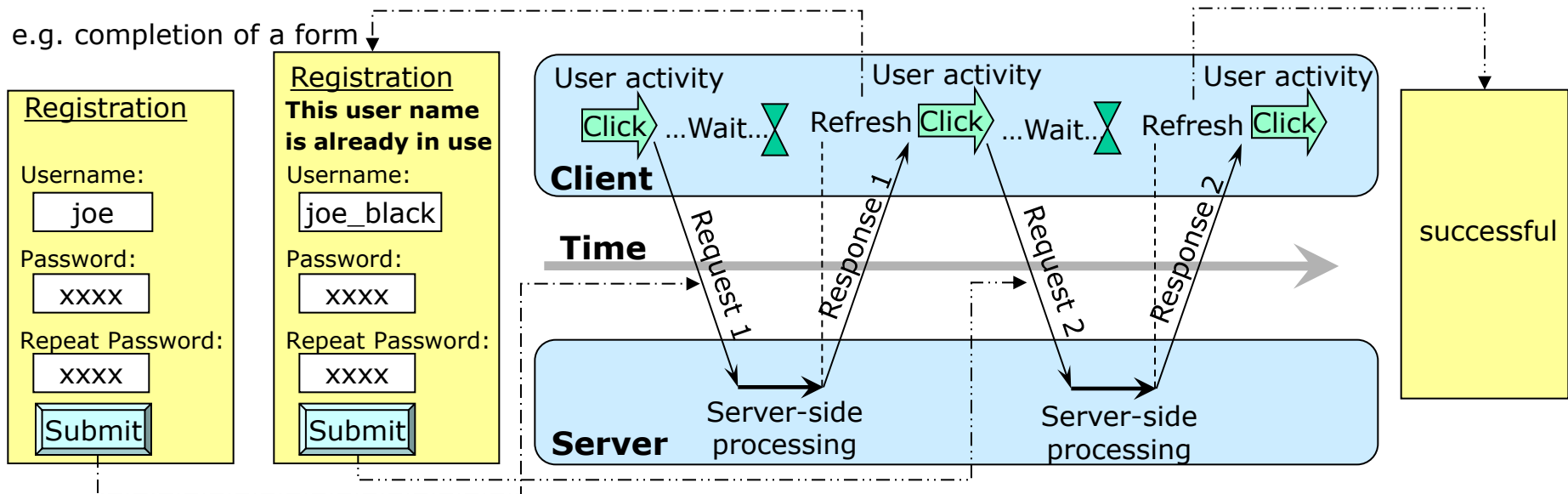
- Synchronous with respect to user actions means that in order to update the displayed data an HTTP Request is sent and the user actions are *blocked* until an HTTP Response from the server was sent
- High degree of dependence between HTTP Request and HTTP Response means that for every data transfer from server to client a request has to be sent directly before this transfer

⇒ Coupling of user activity and server-side processing

# Synchronous communication model

## “Click, wait, and refresh user interaction paradigm”

- A browser reacts to a user action by sending an HTTP Request to the web server, then the server processes the request and sends a response with a complete (X)HTML page to the browser
- The browser refreshes the screen and displays the new (X)HTML page



# Synchronous communication model

## Shortcoming of the classic web application model

- Slow performance due to “click, wait, and refresh”
- Loss of operation context during page refresh
- Excessive server load and bandwidth consumption due to redundant page refreshes

Result: slow, unreliable, low productivity, low interactivity and inefficient web applications

# Non blocking communication model

## History of AJAX

- In 1999 Microsoft published the first application "Outlook Web Access" which followed the approach of AJAX:
  - A smarter, more responsive, and more interactive web that does not use "click, wait, and refresh" approach (instead of the complete refreshing, only a part of the (X)HTML page can be updated with the help of the XMLHttpRequest object)
  - In this time the approach was named XMLHttpRequest
- Only IE supported the XMLHttpRequest object
- Three years later Mozilla and Netscape implemented the native XMLHttpRequest object in their browser version Mozilla 1.0 and Netscape 7
- But the success was missing
- First, in 2005
  - With an article written by Jesse James Garret who gave this approach a new name: "AJAX" (Asynchronous JavaScript + XML)
  - And with the help of Google web applications such as Google Suggest or Google Maps AJAX has gained popularity

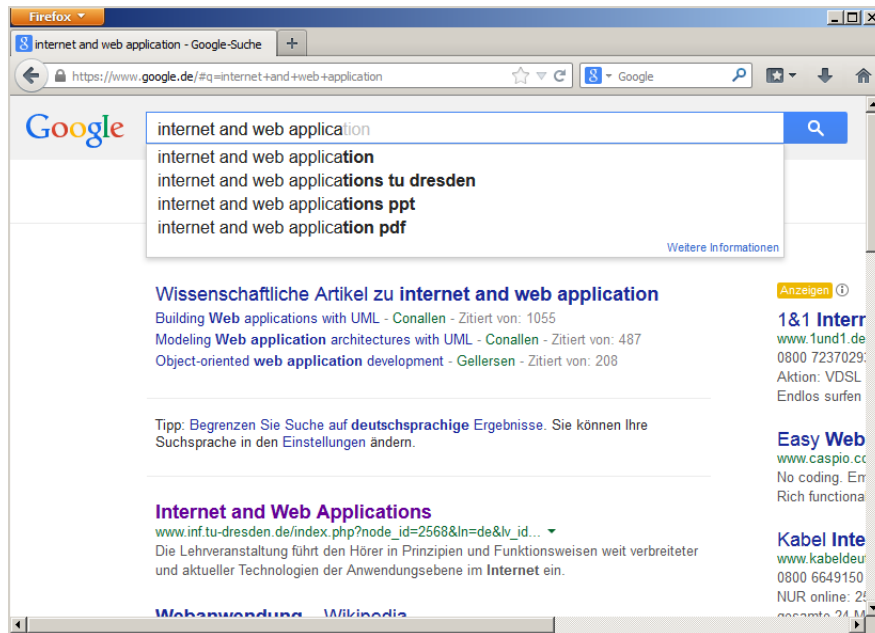
# Non blocking communication model

- Intentions of AJAX:
  - Create web applications that look and feel like desktop applications, e.g. AJAXWrite
  - Extend the classic web application with additional features, e.g. auto completion (Google Suggest)
- Definition of AJAX (J.J. Garret)
  - AJAX isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways. AJAX incorporates:
    - Standard-based presentation using XHTML and CSS
    - Dynamic display and interaction using the Document Object Model (DOM)
    - Data interchange and manipulation using XML and XSLT
    - Asynchronous data retrieval using XMLHttpRequest object
    - And JavaScript binding everything together

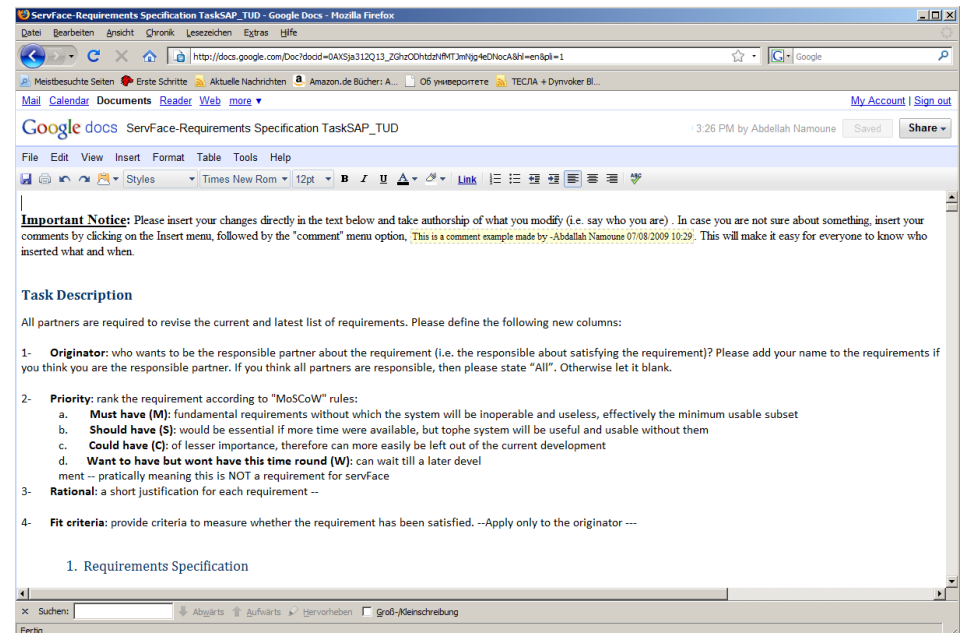


# Non blocking communication model

- Widespread examples for applications enabled by AJAX are:  
Google Suggestions  
Google docs



- As you type, Google Suggest requests suggestions from the server, showing you a drop-down list of search terms that you may be interested in and the particular number of results

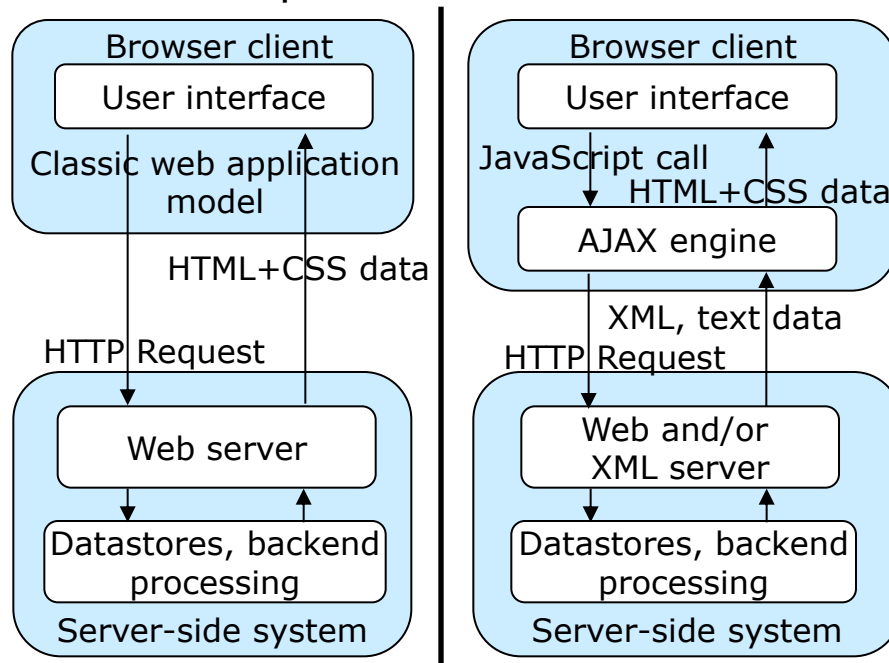


- A free alternative to Microsoft Word
- It allows you to edit word files in your browser
- The document can be saved in many formats
  - MS Word, Open Office, RTF, Text, PDF

# Non blocking communication model

- An AJAX application eliminates the “click, wait, and refresh user interaction paradigm” on the web by introducing an intermediary — the so called “AJAX engine”
- This engine is responsible for rendering the user interface and communicating with the server on the user’s behalf
- The Ajax engine allows the user’s interaction with the application in an asynchronous way

➡ User actions are decoupled from communication with the server



# Non blocking communication model

## Functionality of the AJAX engine

- User actions generate a JavaScript call to the AJAX engine instead of an HTTP Request
  - Not every response to user actions requires a request to the server; the engine can operate on its own, e.g. simple data validation, editing data in memory, and even some navigation.
  - If the engine needs something from the server in order to response to a user action, then it sends an HTTP Request to the server to get the needed data for the processing of this response
- > The engine makes those requests asynchronously

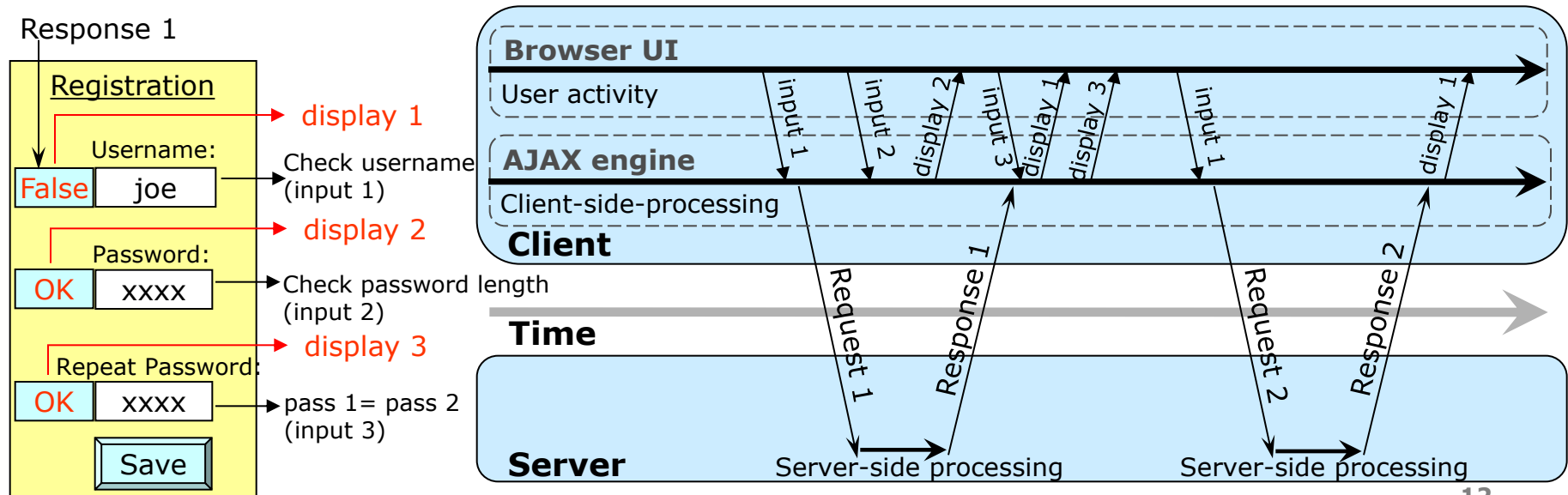
# Non blocking communication model

## Criteria:

- Asynchronous with respect to the user actions means that to update the displayed data the user actions are not blocked by waiting for the HTTP Response from the server
- High degree of dependence between HTTP Request and HTTP Response means in this context that for each server-side processed update a full request/response round trip between AJAX engine and server is necessary

⇒ User activity and server-side processing are decoupled

## AJAX → non blocking model



e.g. completion of a form

## Non blocking communication model

- AJAX does not specify a server-side programming language

### XMLHttpRequest – the heart of AJAX

- The object type XMLHttpRequest provides a client-side programming API (Application Programming Interface) for transfer of data via HTTP behind the scenes — without causing a complete page refresh
- The request to the server is initialised and the corresponding response is handled via the XMLHttpRequest object

### Creation of a XMLHttpRequest object:

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

- Formerly, the Internet Explorer (IE) (up to and including IE6) used an ActiveX component XMLHTTP instead of XMLHttpRequest (*has almost same functionality as XMLHttpRequest has*)

- For IE5 and IE6:

```
var xmlhttpRequest = new ActiveXObject("Msxml2.XMLHTTP");
```

- For older versions of IE:

```
var xmlhttpRequest = new ActiveXObject("Microsoft.XMLHTTP");
```

- Since IE7 the native XMLHttpRequest object is supported

# Non blocking communication model

## Methods of the XMLHttpRequest object


- `open("method", "URL", ["async", "userName", "password"])`
  - Realises pre-set-up of a connection to the server
  - The request methods are typically GET or POST
  - URL – A string indicating the URL to send the request to
  - `async`, `userName` and `password` are optional
  - The boolean value “`async`” indicates whether the data transfer should be made asynchronously, at default: `async="false"`
- `send("data")`
  - Activates the attributes of the open-method
  - If the request method is GET then the data is “null”
  - If the request method is POST then the data consists of the key value pairs which should be sent to the server
    - E.g. `page=1&section=4`
- `abort()`
  - Cancels the current request

# Non blocking communication model

## Properties of the XMLHttpRequest object

- During the asynchronous transfer of data the XMLHttpRequest object creates status events
- These events can be fetched and processed further by a callback function (function that is passed as an argument to another function)
- Afterwards the result of the data transfer (e.g. XML or text data) can be interpreted and used for updating the page
- *onreadystatechange*
  - An event handler which points to the callback function
  - The callback function is necessary for the asynchronous data transfer
  - If the readyState property changes, a readystatechange event is fired and the onreadystatechange event handler is called

# Non blocking communication model

- *readyState*
    - Returns the current status of the processing of an HTTP Request
    - There are five possible values for *readyState*:
      - 0 (Uninitialized)*: The object has been created but the *open()* method hasn't been called
      - 1 (Loading)*: The *open()* method has been called but the request hasn't been sent
      - 2 (Loaded)*: The request has been sent
      - 3 (Interactive)*: A partial HTTP Response has been received
      - 4 (Complete)*: All data has been received and the connection has been closed
  - *responseText*
    - Returns the value (result of the server-side processing) of the HTTP Response from the server as a string
  - *responseXML*
    - Returns the value of the HTTP Response in XML format
  - *status (HTTP Status Codes)*
    - Returns numeric status code regarding an HTTP Request
  - *statusText (HTTP Status Codes)*
    - Returns the corresponding text to the status
- 
- Extracted from HTTP Response



# Non blocking communication model

## AJAX hello world example

- The example consists of three files:
  - helloworld.html – HTML page for presentation
  - helloworld.js – JavaScript for client-side processing
  - helloworld.xml – XML file that includes content
- The data of the XML file is used for updating the HTML page without complete refresh

# Non blocking communication model

## helloworld.html

```
<html>
<head><title>AJAX hello world example </title>
<script type="text/javascript" src="helloworld.js"></script></head>
<body>
<h1>AJAX hello world example</h1>
<!-- Initialise the XMLHttpRequest via a click on this button -->
<p><form>
<input type="button" onclick="javascript:loadData();"
  value="Get XML data"/></form></p>
<!-- <div id="present">... A reserved block that is filled with dynamic
  information (extracted from XML content of the HTTP Response -->
<div id="present"></div>
</body>
</html>
```

# Non blocking communication model

## helloworld.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<helloworld>
<message> <lang>Chinese</lang> <text>你好世界!</text>
</message>
<message> <lang>English</lang> <text>Hello world!</text>
</message>
<message> <lang>French</lang> <text>Salut le monde!</text>
</message>
<message> <lang>German</lang> <text>Hallo Welt!</text>
</message>
<message> <lang>Russian</lang> <text>здравствуйте мир!</text>
</message>
<message> <lang>Vietnamese</lang> <text>Chào thế giới!</text>
</message>
</helloworld>
```

# Non blocking communication model

## helloworld.js (1/2)

```
function loadData() {  
    // Get an instance of the XMLHttpRequest Object or ActiveX Object  
    var req = getXMLHttpRequest();  
    // Pre set-up a connection to the server  
    req.open("GET", "helloworld.xml", true);  
    // Event handler function  
    req.onreadystatechange = function() {  
        // If all data has been received and the request was successful  
        if(req.readyState == 4 && req.status == 200) {  
            // Create a table in the reserved block <div> with ID "present"  
            var tab_content = "<table><tr><td id=\"content\"></td></tr></table>";  
            // The property "document" represents the current HTML document  
            document.getElementById("present").innerHTML = tab_content;  
            // Get the table col with ID "content" to update with XML data  
            var output = document.getElementById("content");  
            var xml = req.responseXML;
```

# Non blocking communication model

## helloworld.js (2/2)

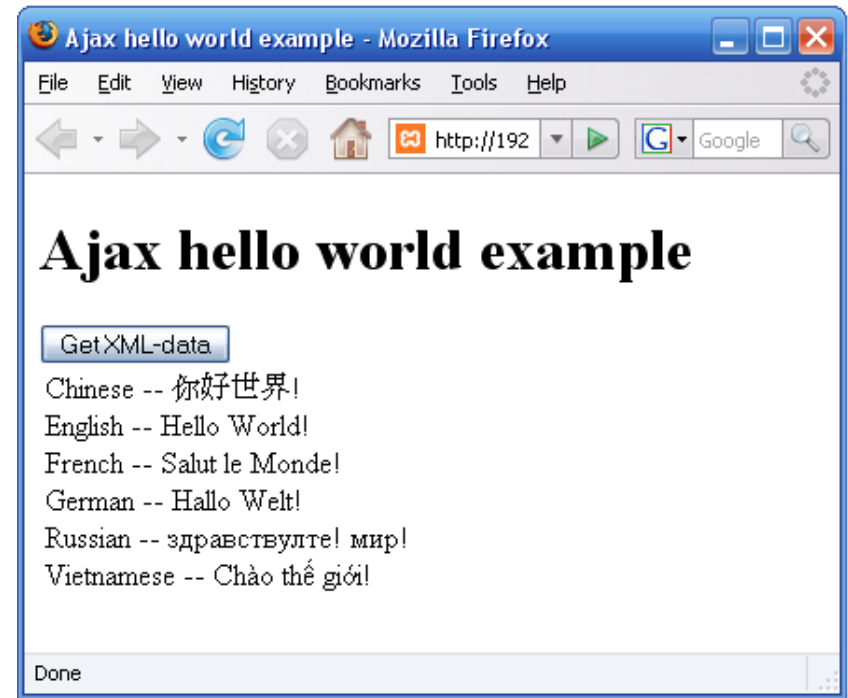
```
// Access all childNodes of the root element via documentElement
for(i=0;i<xml.documentElement.childNodes.length;i++) {
// Firefox interprets a space or line break between tags as childNode ("TEXT_NODE")
// continue - causes the iteration to be skipped
if(xml.documentElement.childNodes[i].nodeType==3) continue;
// Append the value of <lang> and <text> to table column with ID "content"
var langTag =xml.documentElement.childNodes[i].getElementsByTagName("lang")[0].
firstChild.nodeValue;
output.appendChild(document.createTextNode(langTag));
output.appendChild(document.createTextNode(" -- "));
var textTag = xml.documentElement.childNodes[i].getElementsByTagName("text")[0].
firstChild.nodeValue;
output.appendChild(document.createTextNode(textTag));
output.appendChild(document.createElement("br"));
} } }
req.send(null);
}
// The XMLHttpRequest object creation function of slide 11
function getXMLHttpRequest() {....}
```

## Full source code

```
function loadData()
{
    var req = getXMLHttpRequest();
    req.open("GET", "helloworld.xml", true);
    req.onreadystatechange = function()
    {
        if(req.readyState == 4 && req.status == 200)
        {
            var tab_content = "<table><tr><td id=\"content\"></td></tr></table>";
            document.getElementById("present").innerHTML = tab_content;
            var output = document.getElementById("content");
            var xml = req.responseXML;
            for(i=0;i<xml.documentElement.childNodes.length;i++) {
                if(xml.documentElement.childNodes[i].nodeType==3) continue;
                var langTag=xml.documentElement.childNodes[i].getElementsByTagName("lang")[0].firstChild.nodeValue;
                output.appendChild(document.createTextNode(langTag));
                output.appendChild(document.createTextNode(" -- "));
                var textTag=xml.documentElement.childNodes[i].getElementsByTagName("text")[0].firstChild.nodeValue;
                output.appendChild(document.createTextNode(textTag));
                output.appendChild(document.createElement("br"));
            }
        }
        req.send(null);
    }
}

function getXMLHttpRequest(){
    if (window.XMLHttpRequest) {return new XMLHttpRequest();}
    else if (window.ActiveXObject)
    {
        try {return new ActiveXObject("Msxml2.XMLHTTP");}
        catch(e) {
            try {return new ActiveXObject("Microsoft.XMLHTTP");}
            catch(e) {return null;}
        }
    }
    return false;
}
```

# Non blocking communication model



## Alternative data representation

- X in AJAX stands for XML → XML creates unnecessary overhead
- An often applied alternative is JSON:
  - A lightweight data interchange text format
    - Compact encoding and less overhead compared to XML
  - Language independent but uses conventions that are familiar to the C-family of programming languages (C, C++, C#, Java, ...)
  - The parsing of native data structures of the used programming language is faster than parsing XML
- Simple example:

```
{ "helloworld":  [
    { "lang": "English", "text": "Hello World!" },
    { "lang": "French", "text": "Salut le Monde!" },
    { "lang": "German", "text": "Hallo Welt!" },
  ]
}
```



# Alternative data representation

## JSON (JavaScript Object Notation)

- JSON structures are classified in objects, arrays and their values
  - A JSON object is an unordered set of name/value pairs
    - Objects are enclosed in braces `{...}`
    - Each object name is followed by a colon `:`
    - Name/value pairs are separated by a comma `,`
  - A JSON array is an ordered collection of values
    - Values are enclosed in brackets `[...]`
    - Values are separated by a comma `,`
  - JSON value can be:
    - Strings (in quotation marks "...")
    - A Number
    - Boolean (true or false)
    - An object or
    - An array
- These structures can be nested

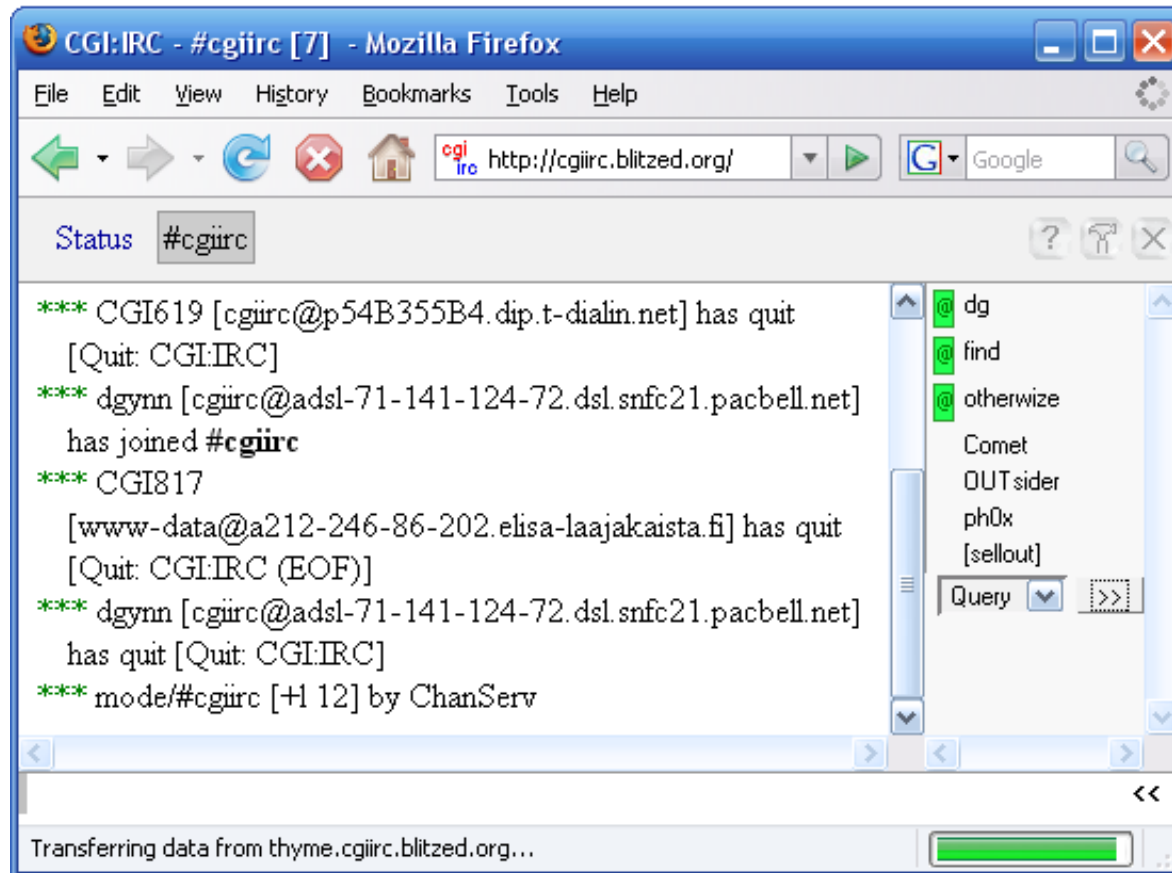
# Asynchronous communication model

## Comet

- AJAX has a high degree of dependence between HTTP Request and HTTP Response
- Comet enables web servers to send data to the client with only an initial HTTP Request and after this without having any need for the client to request the data, so it is completely asynchronous -> low-latency data transfer
- With a weblog article of march 2006 Alex Russell, a project leader of Dojo Toolkit (a Ajax framework) coined the phrase "Comet"
- Comet stands for nothing, like AJAX it is a quite often used buzzword and it also uses already existing technologies
- Comet methodology was used in many web applications before the phrase Comet was born
- Scope of application:
  - For applications in which available data should be transmitted in real-time
  - E.g. Chat, stock exchange applications or the next generation of ebay (always inform about latest information without additional clicks)

# Asynchronous communication model

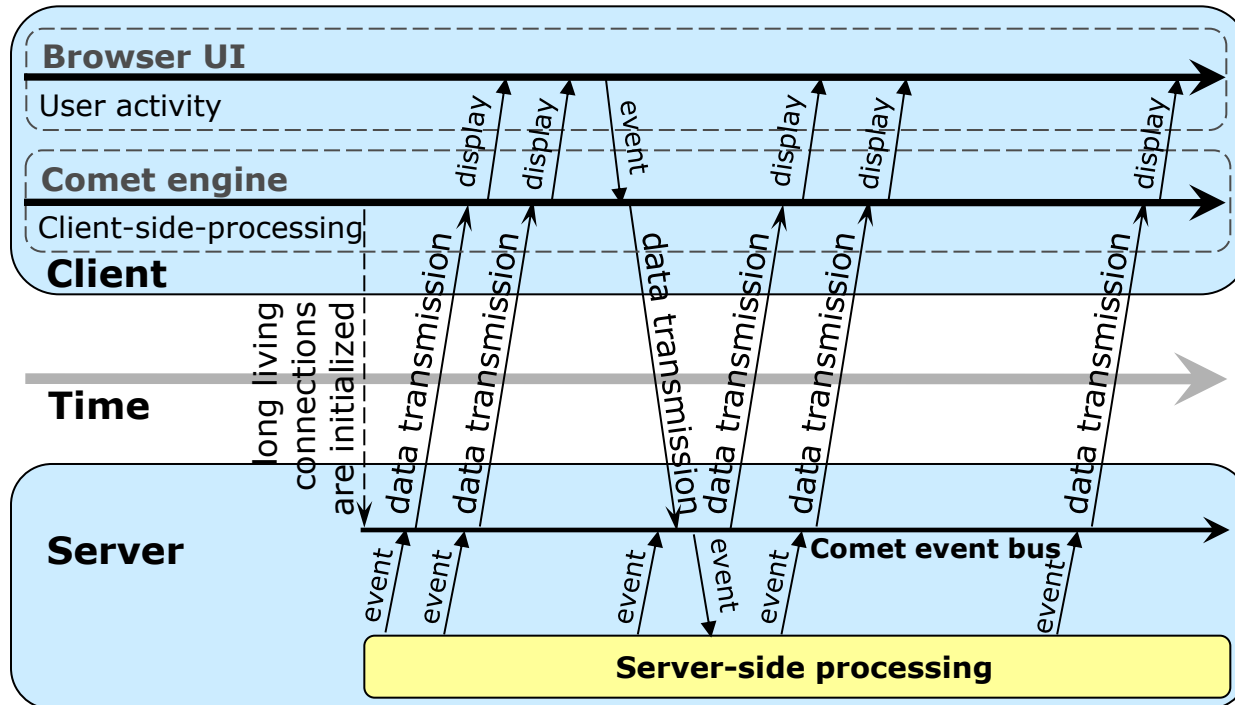
## An example for Comet implementation



Web based IRC client

# Asynchronous communication model

## Comet – asynchronous model

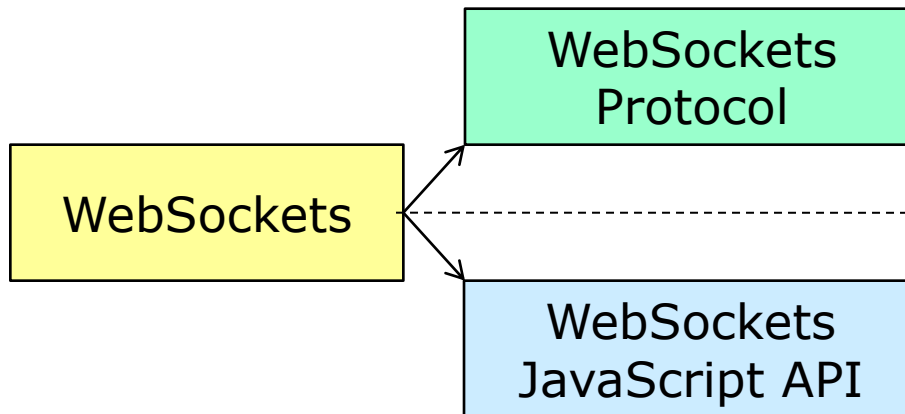


<http://alex.dojotoolkit.org/?p=545>

- Initially established long living "HTTP Connection" only used for data push
  - Data push is triggered by the event output of the server
- For uploading data to the server a second connection to the HTTP server is necessary
- The described method of transmitting data is also named "server push" or "HTTP streaming"

# WebSockets

- HTTP-based streaming has two central shortcomings:
  - Overhead due to HTTP header and unnecessary HTTP requests in long-polling approaches
  - Communication channel is unidirectional: Separate HTTP requests are necessary for realizing bidirectional data exchange
- WebSockets are a standard for TCP-based bidirectional communication channels avoiding these drawbacks
- Standard is associated to HTML5 and already widely supported in state-of-the-art browsers and Web servers

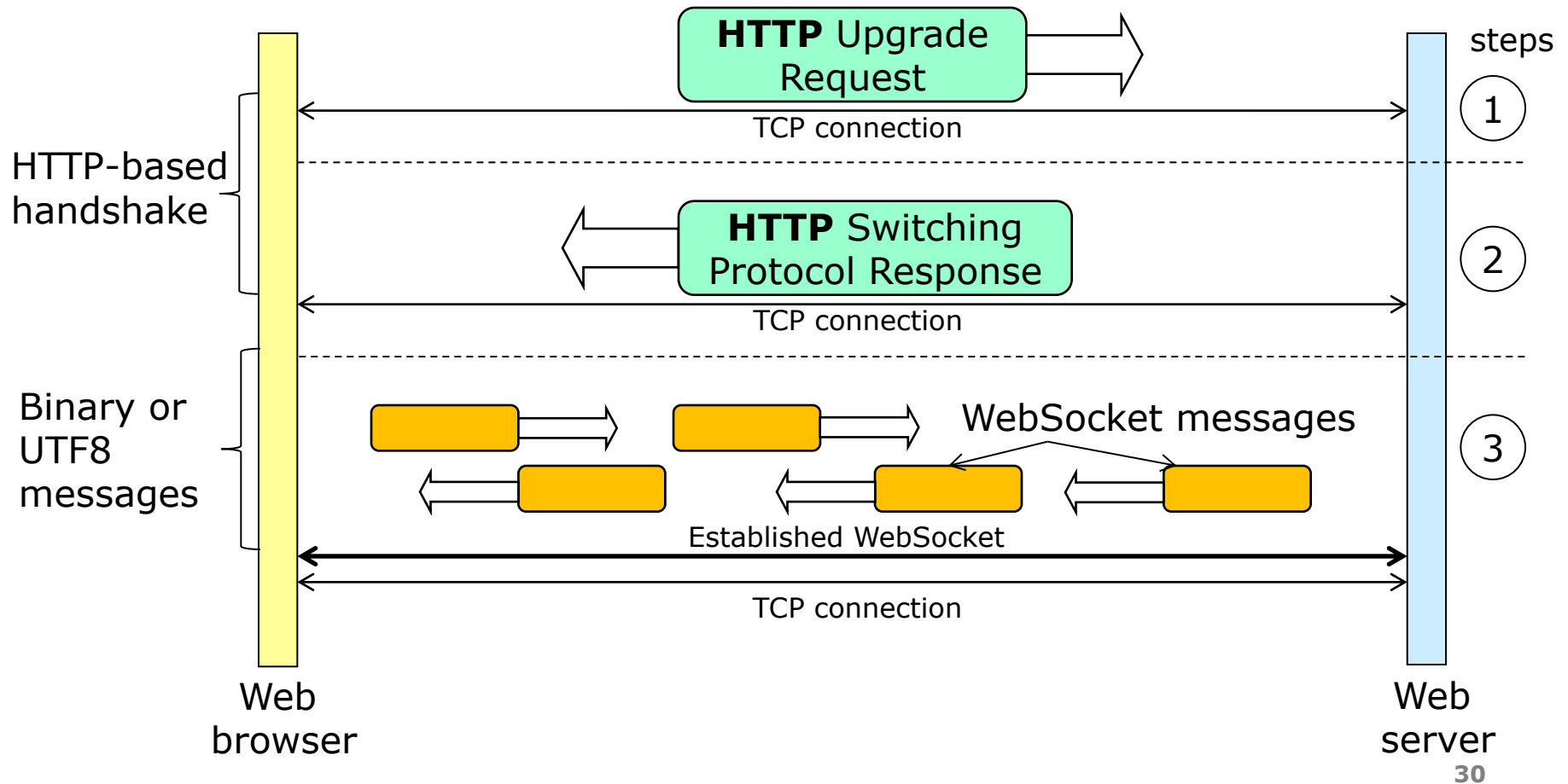


- Defined by Internet Engineering Task Force in RFC 6455
- Covers specification for establishing connection ("handshake") and for data exchange

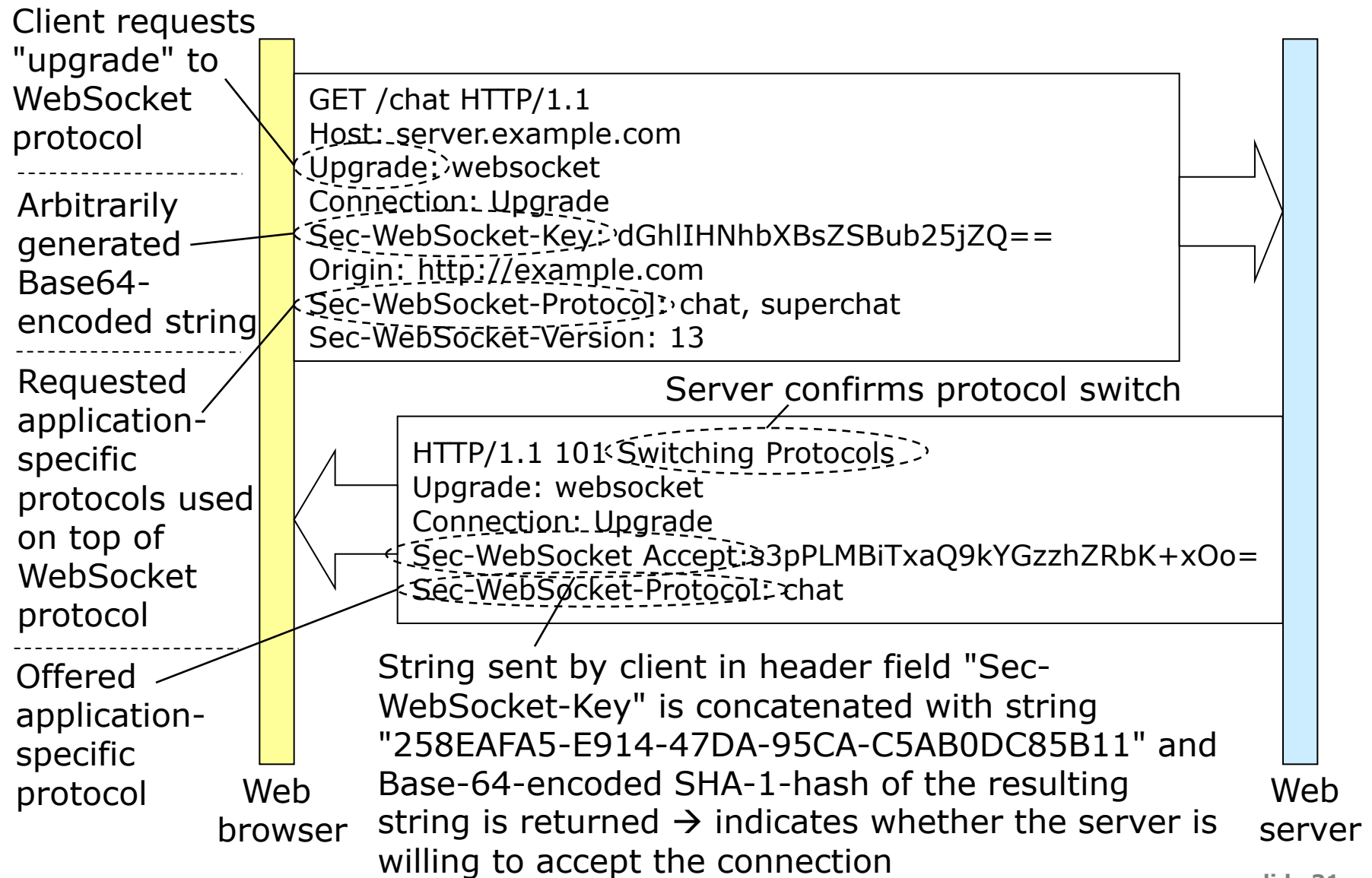
- Defined by World Wide Web Consortium

# WebSockets

- Applying the HTTP protocol, a WebSocket connection is established using the existing TCP connection
- Data encapsulated into frames can be exchanged in full-duplex mode



# WebSockets Protocol – handshake details



# WebSockets - API

- W3C has defined a simple API for creating and using WebSockets
- WebSocket object is created via constructor "WebSocket"

```
var ws = new WebSocket("ws://localhost:9998/chat");
```

- In order to react to different states of the WebSocket and events associated to it, event listeners can be defined:
  - **onopen**: Called as soon as the connection is established
  - **onclose**: Called if the connection is closed
  - **onerror**: Listener is executed if a connection or communication error occurs
  - **onmessage**: Called if a message arrives from server
- Messages are sent to server via method send(string)

```
ws.send("MESSAGE") ;
```

- Connection is closed via method close()

```
ws.close();
```



# Web Sockets - API

```
<!DOCTYPE HTML> ...<head>
<script type="text/javascript">
    function webSocketExample() {
        var ws = new WebSocket("ws://localhost:9998/chat");
        ws.onopen = function() {
            ws.send("Example message...");
            alert("Message is sent..."); ...
        };
        ws.onmessage = function (evt) {
            var received_msg = evt.data;
            alert("Message is received...");
            ...
        };
        ws.onclose = function() {
            alert("Connection is closed..."); ...
        };
    }
</script>
</head>
<body>
<a href="javascript:webSocketExample()">Run example</a>
</body>
</html>
```

Create new WebSocket and establish connection to specified server

Function sending a message to server is executed after the connection is established

Executed as soon as a message is sent by the server

As soon as link is clicked JavaScript function named "webSocketExample" is executed

# Conclusion

## 1. Synchronous communication model:

- “Click, wait, and refresh user interaction paradigm”
  - Slow, unreliable, low productivity, low interactivity and inefficient web applications

## 2. Non blocking communication model:

- AJAX web application model
  - A smarter, more responsive and more interactive web that does not use the “click, wait, and refresh” approach (instead of the complete refreshing, it is possible to update only a part of the (X)HTML page with the help of XMLHttpRequest object)
  - But not asynchronous because of high degree of dependence between HTTP Request and HTTP Response

## 3. Asynchronous communication model:

- Comet web application model
  - Complete asynchronous data transfer
  - Shortcomings of realization via long-polling: Overhead and inefficiency
- WebSockets
  - TCP-based bi-directional communication channels established using the HTTP protocol avoiding the drawbacks of long-polling

## References

- Jesse Jamex Garret, **Ajax: A New Approach to Web Applications**, <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications>
- Alex Russel: **Comet: Low Latency Data for the Browser**, <http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/>
- **JSON Specification**, <http://www.json.org/>
- **WebSockets** protocol specification in IETF RFC 6455, <http://tools.ietf.org/html/rfc6455>
- **WebSockets** API specification provided by W3C, <http://dev.w3.org/html5/websockets>