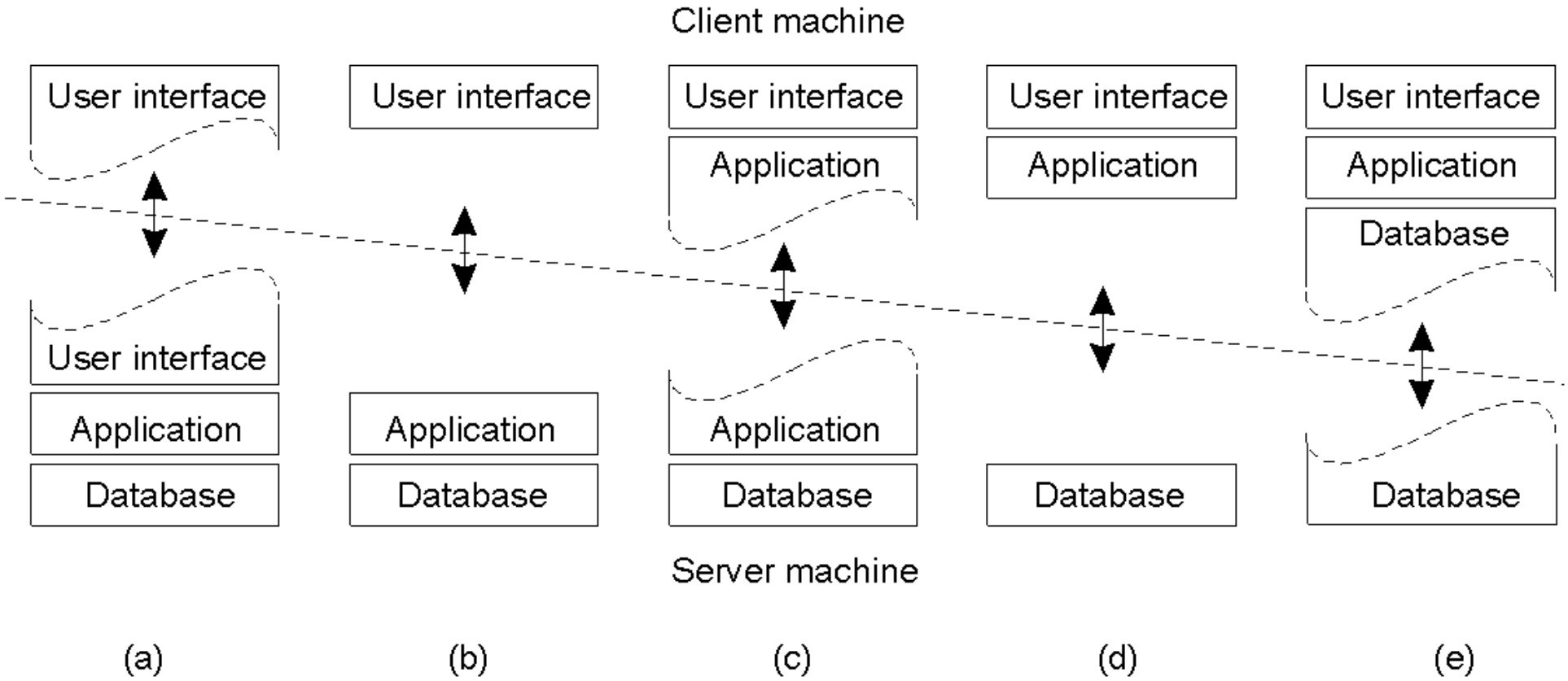
AJAX

Multi-tiered architectures

- One tier addressing one concern
  - typically Model-View-Controller (MVC) compliant



HTTP: the Hypertext Transfer Protocol

- HTTP is the protocol that supports communication between web browsers and web servers.
  - “HTTP is an application-level protocol with the lightness and speed necessary for distributed, hypermedia information systems.” (IETF)
  - Generally on top of TCP (but transport independent – RFC)
- A Standard:
  - RFC 1945 (HTTP 1.0)
  - RFC 2068 and 2616 (HTTP 1.1)
- Simple structure:
  - client sends a request
  - server returns a reply.
  - multiple request-reply exchanges possible over a single TCP connection.

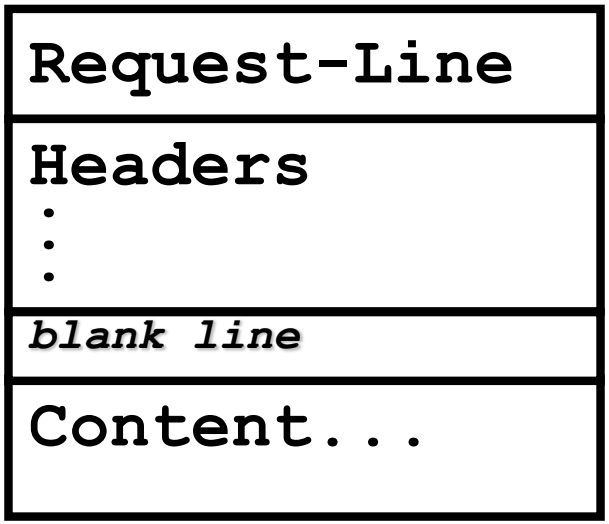
HTTP Request

- Request line:

```
Method URI HTTP-Version\r\n
```

scheme://hotstname[:port]/path or /path

  - GET: retrieve data identified by the URI.
  - HEAD: retrieve meta-information about the URI.
  - POST: send data to a URI and retrieve result.
  - *PUT: Store data in location named by URI.*



- Typically:
  - GET used to retrieve an HTML document.
  - HEAD used to find out if a document has changed.
  - POST used to submit a form: includes some **content** (raw bytes)

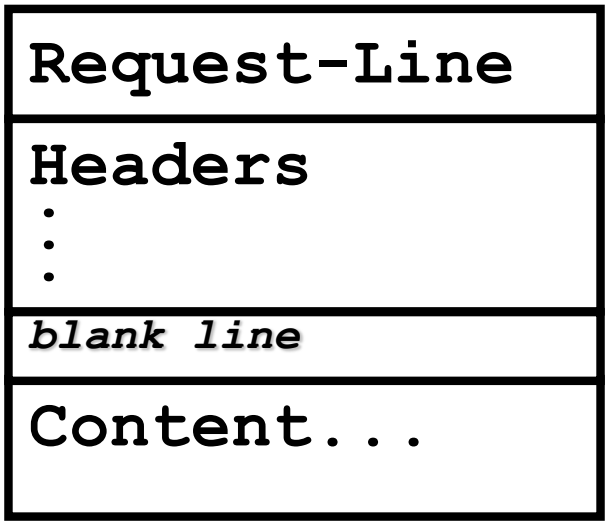
HTTP Request

- Request line:

```
Method URI HTTP-Version\r\n
```

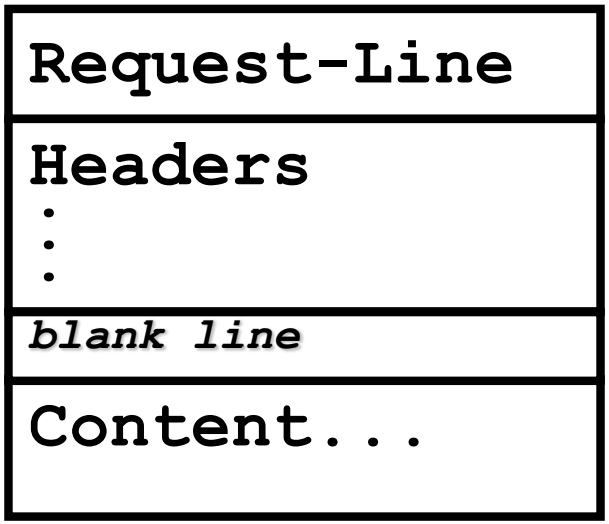
scheme://hotstname[:port]/path or /path

  - GET: retrieve data identified by the URI.
  - HEAD: retrieve meta-information about the URI.
  - POST: send data to a URI and retrieve result.
  - *PUT: Store data in location named by URI.*
  - *DELETE: remove entity identified by URI.*
  - *TRACE: trace HTTP forwarding through proxies, tunnels, etc.*
  - *OPTIONS: used to determine the capabilities of the server, or characteristics of named resource.*
  - *CONNECT: converts request connection to a transparent tunnel (typically HTTPS through unencrypted HTTP proxy)*



HTTP Request: Headers

- Header lines:
  - Zero or more lines
  - Each header line contains an attribute name followed by a “:” followed by a space and the attribute value.
    - Client type (**`User-Agent: Mozilla/4.0`**)
    - Content accepted (**`Accept: text/html`**)
    - Who is the requestor (**`Host: www.eurecom.fr`**)
    - Origin (**`Referer: http://google.com/blah`**)
    - Size of POST data (**`Content-length: 365`**)
  - HTTP 1.1 requires a **`Host:`** header
- Each header ends with a CRLF (**`\r\n`**)
- The end of the header section is marked with a blank line (CRLF)



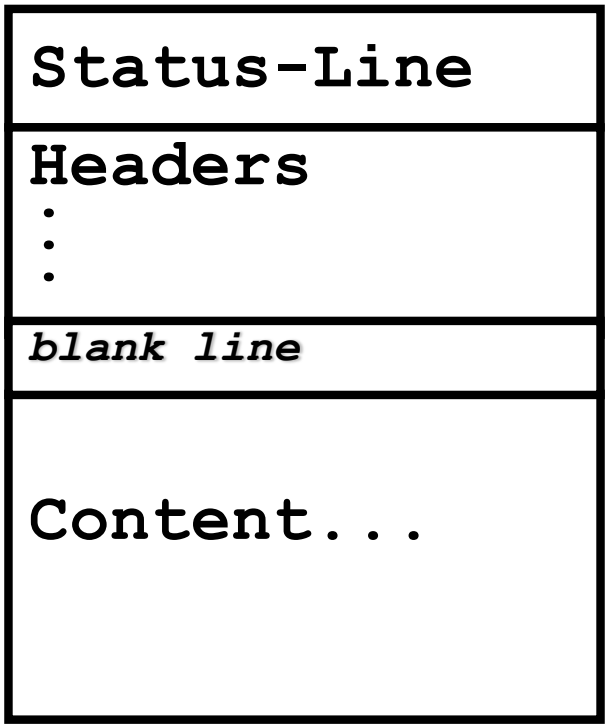
HTTP Response

- Status Line:

  HTTP-Version Status-Code Message

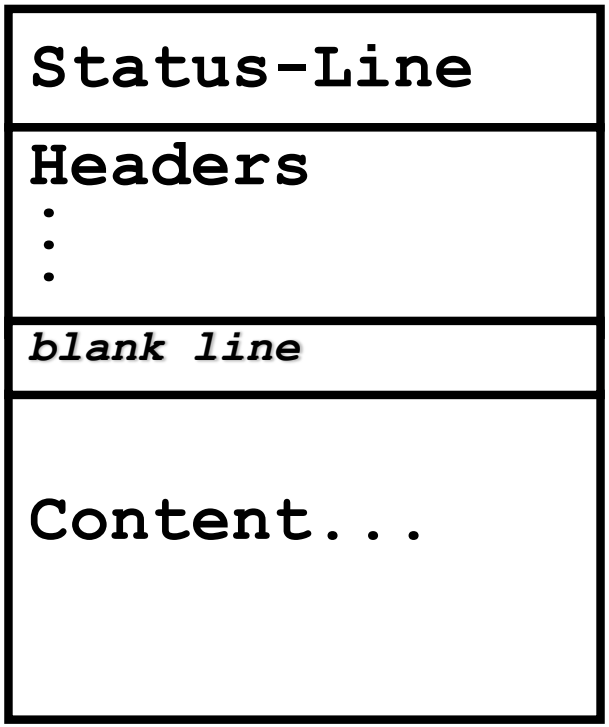
  – Status code is a 3 digit number
    - 1xx Informational
    - 2xx Success
    - 3xx Redirection
    - 4xx Client Error
    - 5xx Server Error

  – Message is text (for humans):
    - HTTP/1.0 200 OK
    - HTTP/1.0 301 Moved Permanently
    - HTTP/1.0 400 Bad Request
    - HTTP/1.0 500 Internal Server Error



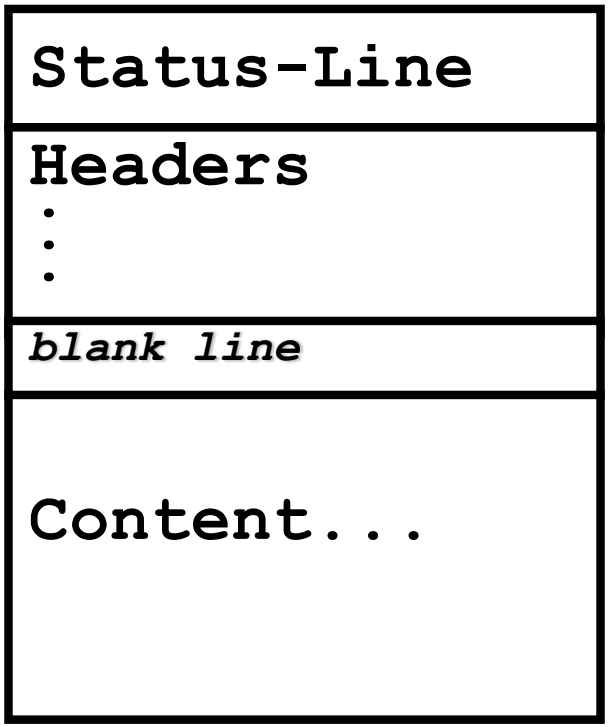
HTTP Response: Headers

- Information about returned document
  - Type (**`Content-Type: text/html`**)
  - Size (**`Content-Length: 1756)`**
  - Encoding (**`Content-Encoding: gzip)`**
  - Last modification (**`Date: Tue, 01 Dec 2009 09:18:17 CET`**)
  - Originator (**`Server: Apache/1.17`**)



HTTP Response: Content

- Content can be anything (not just text)
  - typically an HTML document or some kind of image.



Interactions with HTTP

- Single Request / Reply:
  - The client opens a connection and sends a request.
  - The server sends back the corresponding reply on the connection.
  - The server closes its socket.
  - The client requesting another document opens a new connection.
  - Default behavior for HTTP/1.0

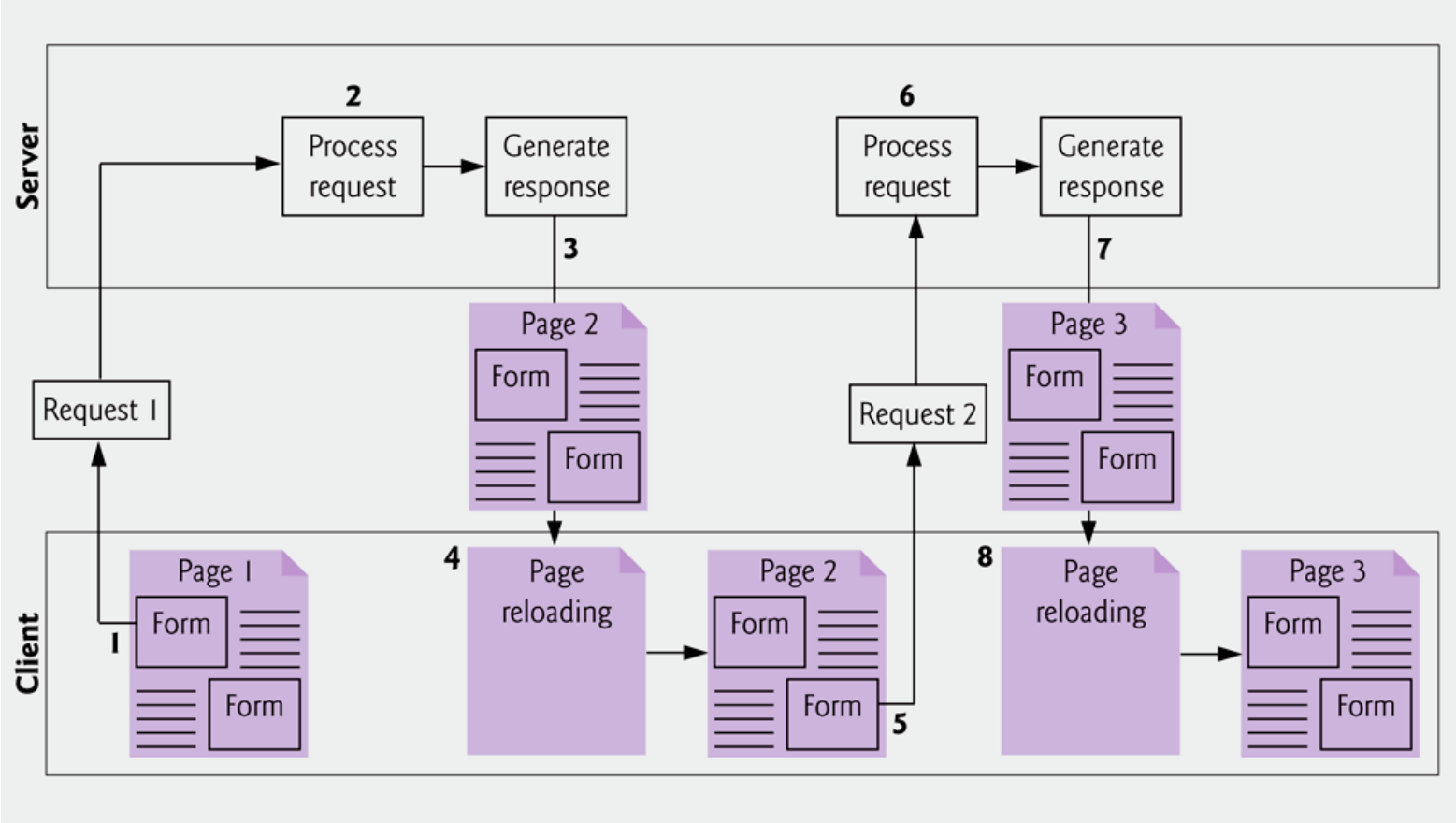
- Persistent Connections:
  - Multiple requests can be handled over a single TCP connection.
  - Behavior triggered by a **`Keep-alive:`** header for HTTP/1.0 Clients
  - Default behavior for HTTP/1.1
  - Information passed in the **`Connection:`** header (HTTP/1.1)

Traditional Web Applications

- “Click, wait, and refresh” user interaction
  - Page refreshes from the server needed for all events, data submissions, and navigation
- Synchronous “request/response” communication model
  - The user has to wait for the response
- Page-driven: Workflow is based on pages
  - Page-navigation logic is determined by the server

- Web pages were first static
  - Based solely on HTML
  - Extended with CGI scripts to generate dynamic content on the server side
  - XML dialects also introduced on the client side (e.g., Xforms from W3C, currently not a priority wrt. HTML5)

Traditional Web Applications



Traditional Web Applications: Pros

- Portability thanks to standardization
  - Ongoing process centralized at W3C
- Simple programming model
  - Well adapted to workflows
  - E.g., merchant acquisition workflows
- State management supported in different manners by the HTTP protocol.

Traditional Web Applications: Cons

- Interruption of user operation
  - Users cannot perform any operation while waiting for a response
- Loss of operational context during refresh
  - Loss of information on the screen
  - Loss of scrolled position
- No instant feedback on user activities
  - A user has to wait for the next page
- Constrained by HTML
  - Lack of useful widgets

- This explains the advent of “Rich Internet Applications” (RIA).

Rich Internet Application (RIA) Technologies

- DHTML (also with Hidden Iframe)
- Java Applets and WebStart
- Macromedia Flash
- Microsoft Silverlight
- .NET – No Touch Deployment
- AJAX, HTML5, CSS, jQuery

- Issues with technologies concurrent to AJAX
  - Require a plug-in
  - Not well supported by mobile browsers (e.g., performance and electrical consumption issues in iOS)
- One general concern
  - Security issues from either design (Flash, Javascript) or browser implementations (Java)

Real-Life Examples of AJAX Apps

- Google maps
  - http://maps.google.com/
- Google Suggest
  - http://www.google.com/webhp?complete=1&hl=en
- NetFlix
  - http://www.netflix.com/BrowseSelection?lnkctr=nmhbs
- Gmail
  - http://gmail.com/
- Yahoo Maps
  - http://maps.yahoo.com/

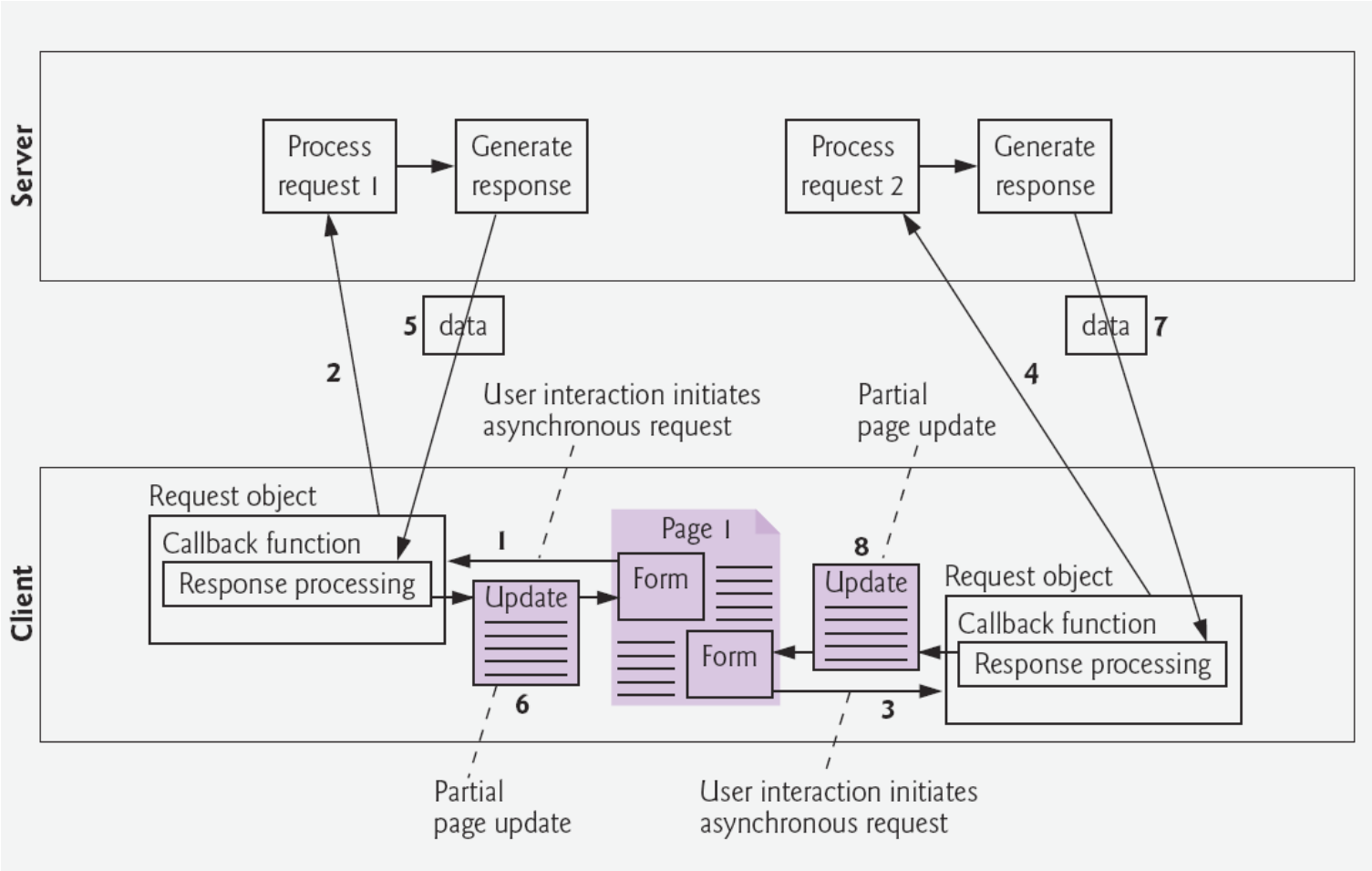
Already quite pervasive, many more are popping everywhere …

What is AJAX?

- *"Asynchronous Javascript And XML(HttpRequest)"*
  - allows the updating of a web page without doing a page reload
    - creates much nicer user experience
- AJAX is not really a technology by itself, but a combination of technologies
    - XHTML, CSS, DOM, XSLT
    - XML, JSON
    - Javascript and some server scripting language to manipulate data (server-side scripting could be done in PHP, .NET, Java Servlet or Java Server Pages)
- AJAX combines the ability to **interact asynchronously** with a server-side component and to **dynamically update/rewrite the source of an HTML page** in the browser based on the resulting XML/JSON/text response
    - Initially part of the W3C standards, yet supported (in different fashions) by Firefox, Internet Explorer, Safari, Opera, and other popular browsers.
    - The term "AJAX" was coined in 2005, but the core XMLHttpRequest object was first supported by Internet Explorer several years before this.

AJAX Approach

- AJAX web applications add a layer between the client and the server to manage communication between the two.



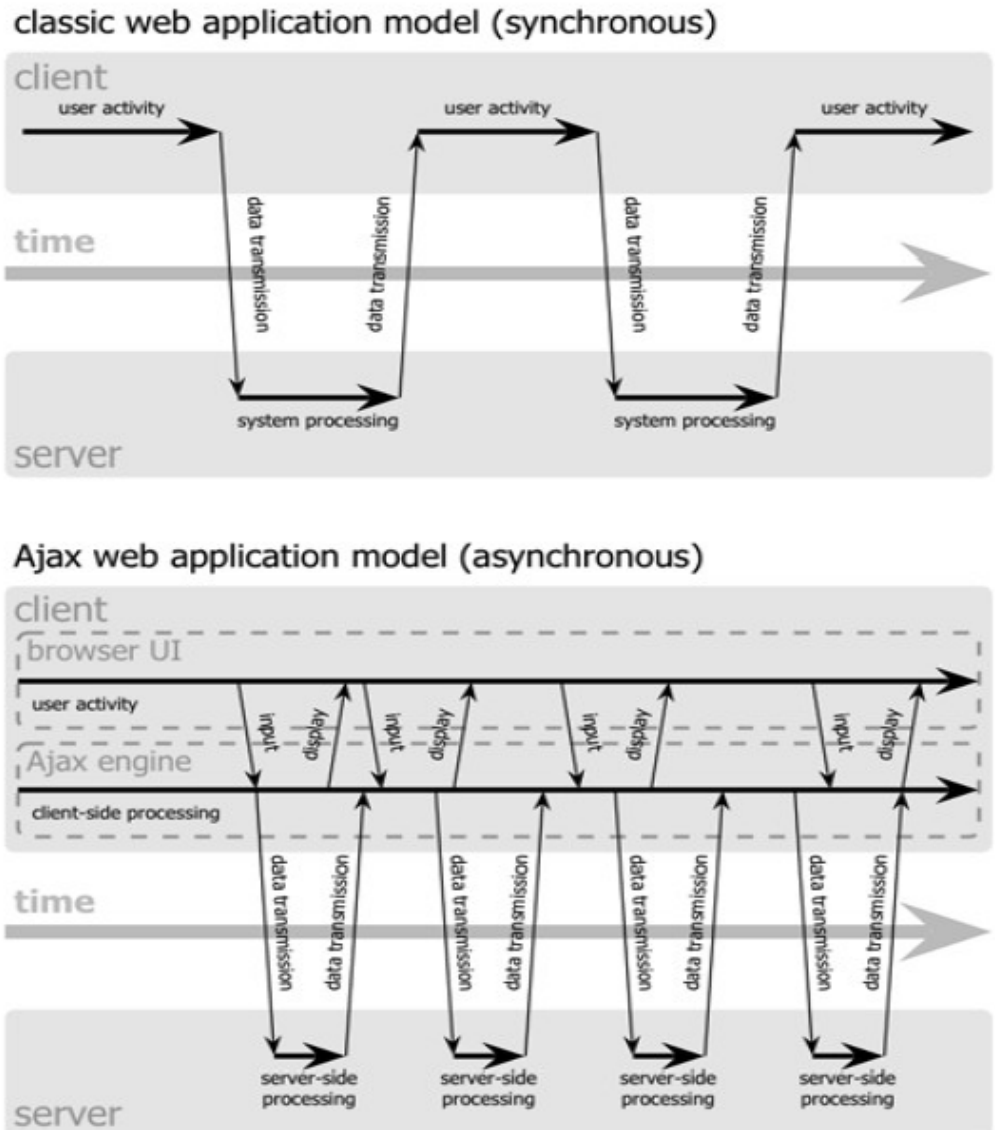
AJAX: client-server interactions

- When the user interacts with the page, the client requests information from the server (Step 1).
- The request is intercepted by the AJAX controls and sent to the server as an *asynchronous request* (Step 2)
- The user can continue interacting with the application in the client browser while the server processes the request.
- Other user interactions could result in additional requests to the server (Steps 3 and 4).
- Once the server responds to the original request (Step 5), the AJAX control calls a client-side function to process the data returned by the server and possibly display it to the user.
- This function—known as a callback function—uses partial page updates (Step 6) to display the data without reloading the entire page.
- At the same time, the server may be responding to the second request (Step 7) and the client browser may be starting another partial-page update (Step 8).

AJAX vs. HTTP request/response

- Standard request/response
  - Synchronous: operations are blocking and even disable the display of the page
  - Each click on a hyperlink or button presents (and transfers) and whole new page
- AJAX:
  - Asynchronous: each action sends data and receives results in the background
  - The callback function updates only a designated part of the page.
  - Such partial-page updates help make web applications more responsive, making them feel more like desktop applications.
  - The web application does not load a new page while the user interacts with it.

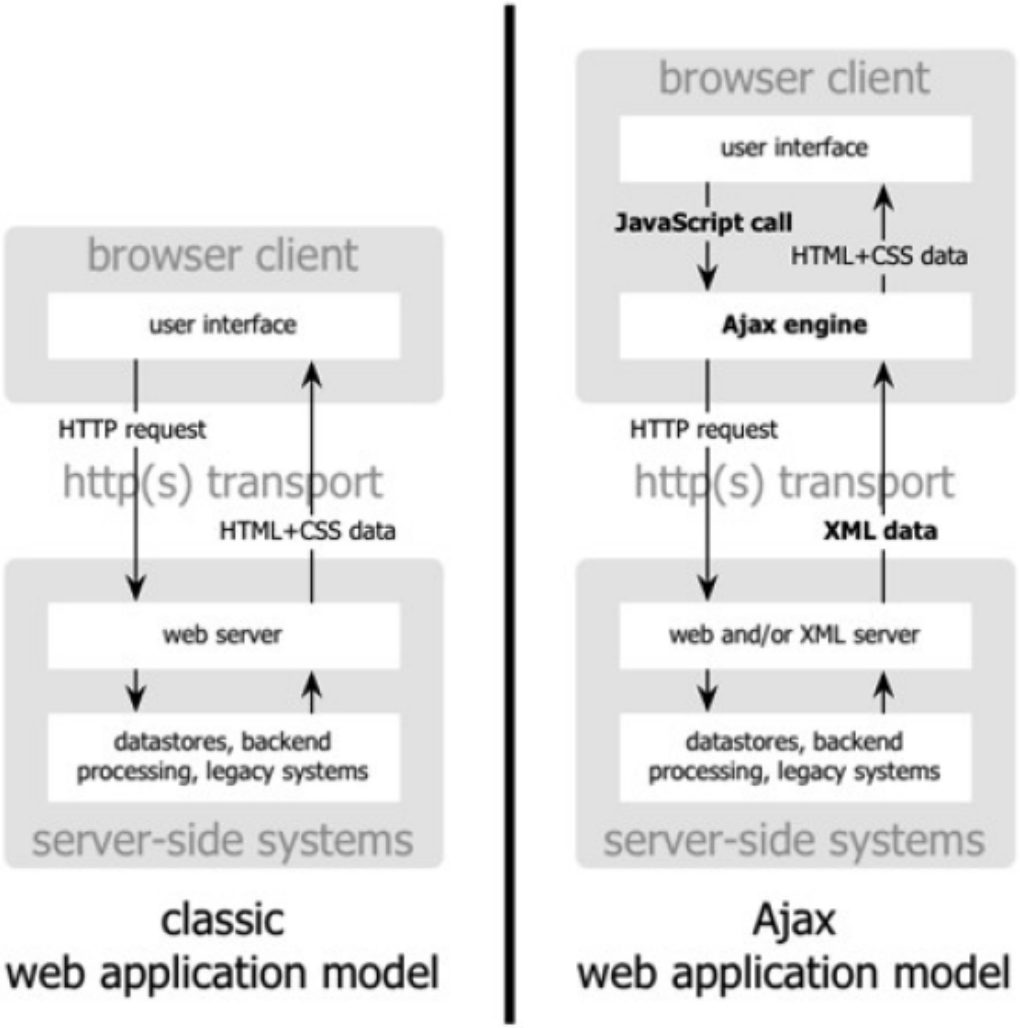
AJAX: client-server interactions



Interrupted user operation while the data is being fetched

Uninterrupted user operation while data is being fetched

AJAX Stacks


browser client
user interface
HTTP request
http(s) transport
HTML+CSS data
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
XML data
web and/or XML server
datastores, backend
processing, legacy systems
server-side systems
Ajax
web application model

Why AJAX?

- Intuitive and natural user interaction
  - No special clicking required
  - Mouse movement is a sufficient event trigger
- "Partial screen update" replaces the "click, wait, and refresh" user interaction model
  - Only user interface elements that contain new information are updated (fast response)
  - The rest of the user interface remains displayed without interruption (no loss of operational context)
- Data-driven (as opposed to page-driven)
  - UI is handled in the client while the server provides data
- Asynchronous communication replaces "synchronous request/response model."
  - A user can continue to use the application while the client program requests information from the server in the background
  - Separation of displaying from data fetching
- Made possible by the emergence of broadband communications
  - AJAX-based JavaScript can take considerable bandwidth to download

Implementing AJAX

- To implement AJAX we need to answer three questions:
  - What triggers the AJAX request?
    - Usually a Javascript event (onblur, onclick, etc.)
  - What is the server process that handles the AJAX request and issues the response?
    - Some kind of URL (use a Service Locator)
  - What processes the response from the server(what is the callback method)?
    - A Javascript function that gets the response and manipulates the DOM, based on the text returned.

Technologies Used In AJAX

- Javascript
  - Loosely typed scripting language
  - **Javascript function called when event occurs in a page**
  - Glue for the whole AJAX operation
- DOM
  - API for accessing and manipulating structured documents
  - Represents the structure of XML **and HTML documents**
- CSS
  - Style sheets attached to HTML documents **may be changed programmatically** by Javascript
- XMLHttpRequest
  - JavaScript object that performs **asynchronous interaction** with the server

JavaScript Language

- Created by Brendan Eich in Netscape Navigator 2.0 in 1995
  - Introduced under the name LiveScript, “JavaScript” used for marketing purposes
  - Standardized as ECMAScript
- Prototype based scripting language
  - Dynamic typing (types associated with values, not variables which can successively have different types)
  - Object-based: JavaScript object = associative array + prototypes (no inheritance)
  - Functions are first class (themselves objects) and thus have properties and methods (such as .call() and .bind())
  - Interpreted language: for instance, eval() can be called on a string
  - Some performance issues, optimizations exist (depending on browser)
  - Single-threaded

JavaScript Usage

- Runs on user's browser
  - Placed between <script> ... </script> in HTML code
  - Runs on page load
- Also used on server side
  - Initially by Netscape (1995)
  - More notably Node.js framework (2009) for implementing event-driven non-blocking I/O servers
- Usages
  - AJAX
  - Animation of page elements, fading them in and out, resizing them, moving them, etc.
  - Interactive content, for example games, and playing audio and video
  - Validating input values of a form to make sure that they are acceptable before being submitted
  - Collecting user reading habits and browsing activities (web analytics, ad tracking, personalization)

JavaScript Usage

- Example script:

```
<!DOCTYPE html>
 <meta charset="utf-8">
<title>Minimal Example</title>
 <h1 id="header">This is JavaScript</h1>

<script>

    document.body.appendChild(document.createTextNode('Hello World!'));

    var h1 = document.getElementById('header'); // holds a reference to the <h1> tag
    h1 = document.getElementsByTagName('h1')[0]; // accessing the same <h1> element

</script>

<noscript>Your browser either does not support JavaScript, or has it turned off.</noscript>
```

Including JavaScript in a Page

- Inlined:

```
<html> <head></head>
 <body>
  <h1>Here is my Document</h1>
  <script type="text/javascript”>
   // <![CDATA[
    alert("Hello from JavaScript");
   // ]]>
  </script>
  <h1>Here is my second Header</h1>
 </body>
</html
```

For XHTML and HTML (strict) validation

- Library:

```
<html> <head>
   <script type=“text/javascript” src=“lib/library.js”></script>
 </head>
 <body>
   <script>  <!-- Browser will execute this code -->
     var foo=3;
     var bar= functionInLibrary(foo);
  </script>
 </body>
</html>
```

JavaScript: Event Handling

- Some HTML tags have attributes which contain JavaScript that runs when DOM events related to the tag occur
  - Example: onchange, onclick, onmousedown, onmouseup …
  - Described at http://en.wikipedia.org/wiki/DOM_events
  - In the HTML page:

```
<a href="http://www.eurecom.fr" onclick="alert('Hi!')">Hello!</a>
<a href="http://www.noshow.com" onclick="alert('Stop!');return false;">Plan B</a>
```

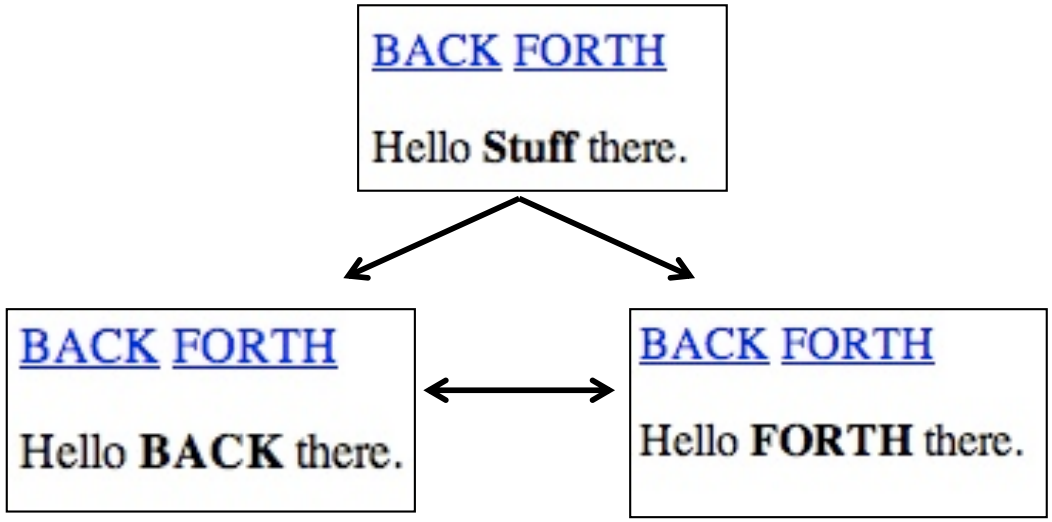
    - Returning false inhibits the link
    - More powerful events:

```
<body onload="message()">
<input type="button" onclick="window.location='/';return false;"
```

JavaScript and DOM

- JavaScript can manipulate the content of HTML documents
  - Syntax of the page described with the Document Object Model
  - Elements of the page referred to by their ID
  - Cf. http://en.wikipedia.org/wiki/Document_Object_Model

```
<p>
<a href="#" onclick="document.getElementById('stuff').innerHTML = 'BACK';">BACK</a>
<a href="#" onclick=" document.getElementById('stuff').innerHTML = 'FORTH';">FORTH</a>
</p><p>
Hello <b> <span id="stuff">Stuff</span> </b> there.
</p>
```

CSS: Cascading Style Sheets

- Data format used to create style sheets attached to web pages
  - Includes inheritance from other style sheets (“cascades”)
  - Formatting structure reflecting the logical tree of the document
- Objectives:
  - Allows for a clear separation of the presentation style from the content
  - Reduce page latency transfer (style sheets used for all the HTML pages of a website)
- CSS usage must be adapted together with AJAX, for instance:
  - Loading in jQuery (HTML file):

```
<script type="text/javascript">
   if (!($('#ajaxCss').length)) { // check whether already loaded
      $("head").append("<link>");
      $("head").children(":last").attr({
        id:   "ajaxCss",
        rel:  "stylesheet",
        type: "text/css",
        href: "./your-file.css »   });
   }
</script>
```

  - Waiting for CSS file to load (HTML file): `<div id="ajaxMainFrame" style="visibility: hidden;"></div>`
  - Finally make it usable (CSS file): `<div id="ajaxMainFrame" style="visibility: hidden;"></div>`

JSON: JavaScript Object Notation

- Alternative to XML
  - Simpler to parse, especially in JavaScript (cf. RESTful Web Services)
- Client-side: data can be retrieved with the JavaScript eval() method
  - Example: .json file:

```
{ ”menu”: ”File”, ”commands” : [
{ ”title”: ”New”, ”action”:”CreateDoc” },
{ ”title”: ”Open”, ”action”: ”OpenDoc” },

{ ”title”: ”Close”, ”action”: ”CloseDoc” }
```

  - Corresponding JavaScript code:

```
req.open(”GET”, ”fichier.json”, true); // request
var doc = eval(’(’ + req.responseText + ’)’); // retrieval
var nameMenu = document.getElementById(’jsmenu’); // lookup
nameMenu.value = doc .menu. value ; // assignment
doc.commands[0]. title // the value ”title” is read in the array

doc.commands[0].action // the corresponding value ”action” is read in the array
```

- Server-side: using a library specific to the language used:
  - Java: org.json.*
  - Perl: JSON
  - PHP: json
  - …

XMLHttpRequest

- The heart of AJAX
  - Initially introduced by Microsoft in its browser in 1999
  - Objective was to replace frames, full-screen updates, provide drag and drop, automatic field completion, etc.
- Adopted by modern browsers
  - Mozilla™, Firefox, Safari, and Opera
  - W3C API for fetching resources – defined at: http://www.w3.org/TR/XMLHttpRequest/
  - Communicates with a server via standard HTTP GET/POST
- Javascript object, works in the background (no interruption for user)
  - asynchronous communication with the backend server
- The XMLHttpRequest name is misleading:
  - This can send any data, not just XML (as initially envisioned/implemented)
  - Notably supports any text based format, including XML and JSON
  - Can be used to make requests over both HTTP and HTTPS
  - Supports "requests" in a broad sense of the term as it pertains to HTTP; namely all activity involved with HTTP requests or responses for the defined HTTP methods.

XMLHttpRequest Object

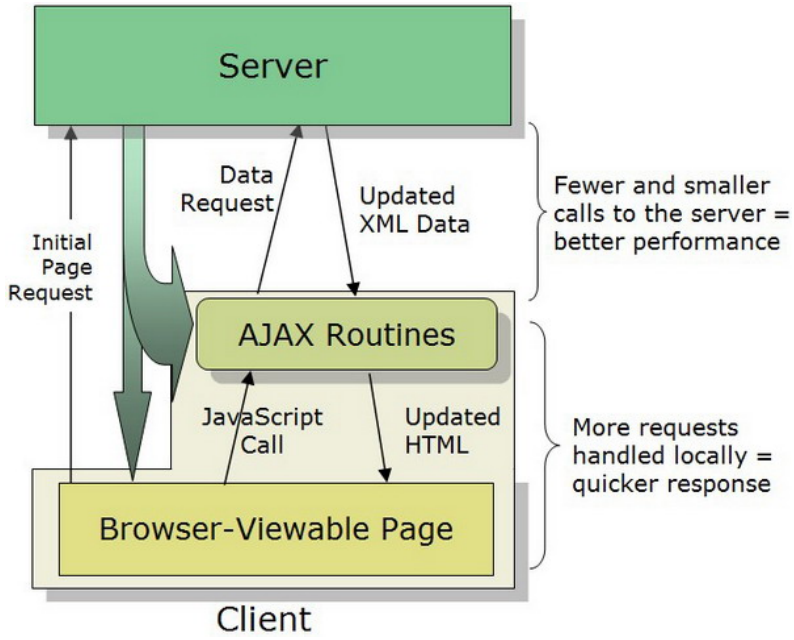
- A few methods:
  - abort() - stop the current request
  - getAllResponseHeaders - Returns complete set of headers (labels and values) as a string
  - getResponseHeader(:headerLabel”) – returns the string value of the requested header field
  - open(“method”,”URL”) sets a pending request
  - send(content) – transmits the request
  - setRequestHeader(“label”,”value”) – sets label/value in the header
  - …

XMLHttpRequest Properties

- onreadystatechange
  - Set with a JavaScript event handler that fires at each state change
- readyState – current status of request
  - 0 = uninitialized
  - 1 = loading
  - 2 = loaded
  - 3 = interactive (some data has been returned)
  - 4 = complete
- status
  - HTTP Status returned from server: 200 = OK
- responseText
  - String version of data returned from the server
- responseXML
  - XML document of data returned from the server
- statusText
  - Status text returned from server

Client-Side AJAX processing

- Request sending
  - Request object (XMLHttpRequest) created
  - Request elements (URL, method, HTTP headers, parameters) specified
  - Event handler definition
  - Object sending
- Response reception
  - For every change to the request state: test if in ready state
  - Processing of the data received (page update, XSL transformations if XML is used as the data format, etc.)

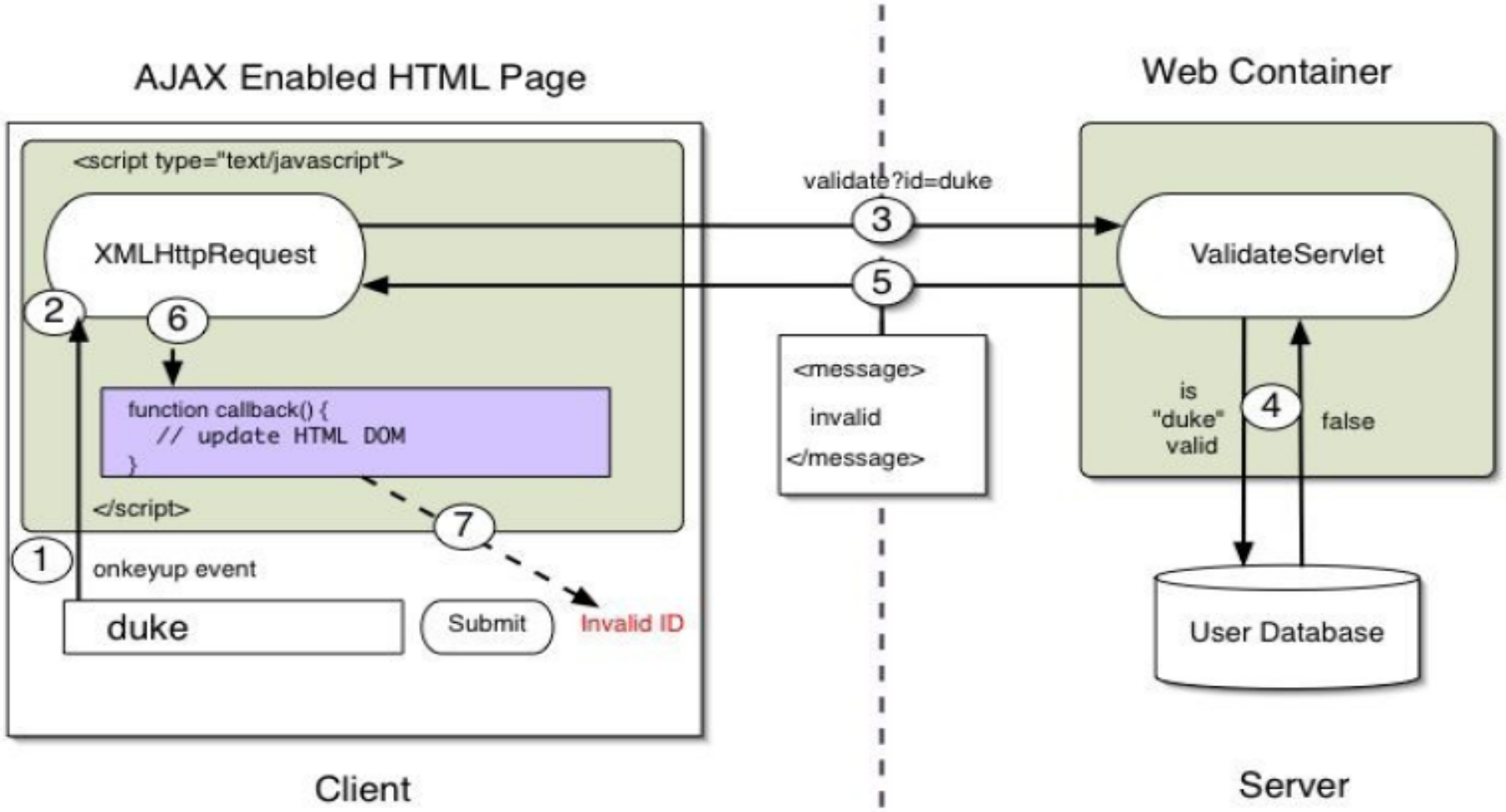


http: //www.codeproject.com/KB/showcase/FarPointAJAX.aspx

Server-Side AJAX Request Processing

- Server programming model remains the same
  - It receives standard HTTP GETs/POSTs
  - Can use Servlet, JSP, JSF, ...
- With minor constraints
  - More frequent and finer-grained requests from client
  - Response content type can be
    - text/xml
    - text/plain
    - text/json
    - text/javascript

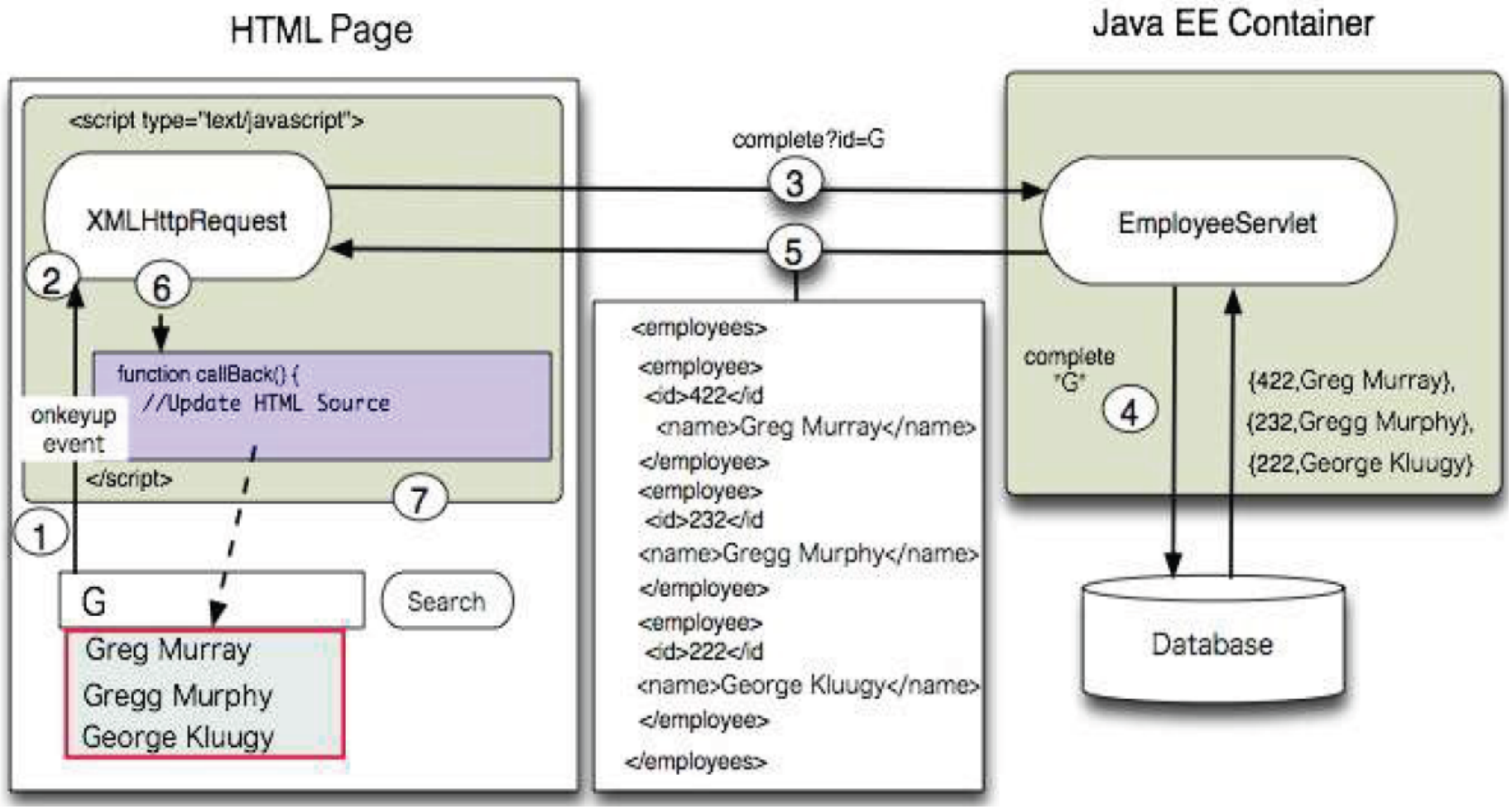
AJAX: Sample App



Steps of AJAX Operation

1. A client event occurs
2. An XMLHttpRequest object is created
3. The XMLHttpRequest object is configured
4. The XMLHttpRequest object makes an asynchronous request
5. The ValidateServlet returns an XML document containing the result
6. The XMLHttpRequest object calls the callback() function and processes the result
7. The HTML DOM is updated

AJAX: Another App



AJAX: Coding Involved

- index.jsp Page Auto-Complete Form

```
<form name="autofillform" action="autocomplete" method="get">
    <input type="text" size="20" autocomplete="off"
      id="completeField" name="id"
      onkeyup="doCompletion();">
    <input id="submit_btn" type="Submit" value="Lookup Employee">
</form>
```

AJAX: Coding Involved

- Client-side: AutoComplete XMLHttpRequest
  - Worth noting: the object construction depends on the browser!

```
function initRequest(url) {
    if (window.XMLHttpRequest) {
        return new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        return new ActiveXObject("Microsoft.XMLHTTP");
    }
}
```

AJAX: Coding Involved

- Client-side: AutoComplete Event Handler

```
function doCompletion() {
      if (completeField.value == "") {
            clearTable();
      } else {
            var url = "autocomplete?action=complete&id=" +
                                                escape(completeField.value);
            var req = initRequest(url);
            req.onreadystatechange = function() {
              if (req.readyState == 4) {
                  if (req.status == 200) {
                        parseMessages(req.responseXML);
                  } else if (req.status == 204){
                        clearTable();
                     }
                  }
              };
            req.open("GET", url, true);
            req.send(null);
      }
}
```

AJAX: Coding Involved

- Server-side: AutoComplete Servlet doGet()

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException
      {
...
String targetId = request.getParameter("id");
Iterator it = employees.keySet().iterator();
while (it.hasNext()) {
      EmployeeBean e = (EmployeeBean)employees.get((String)it.next());
      if ((e.getFirstName().toLowerCase().startsWith(targetId) ||
                  e.getLastName().toLowerCase().startsWith(targetId)) && !targetId.equals("")) {
                  sb.append("<employee>");
                  sb.append("<id>" + e.getId() + "</id>");
                  sb.append("<firstName>" + e.getFirstName() + "</firstName>");
                  sb.append("<lastName>" + e.getLastName() + "</lastName>");
                  sb.append("</employee>");
                  namesAdded = true; } // if
} // while
if (namesAdded) {
      response.setContentType("text/xml");
      response.setHeader("Cache-Control", "no-cache");
      response.getWriter().write("<employees>" + sb.toString() + "</employees>");
} else {
      response.setStatus(HttpServletResponse.SC_NO_CONTENT);
}
} // doGet
```

AJAX: Coding Involved

- Client-side: Processing the response

```
function parseMessages(responseXML) {
clearTable();
var employees = responseXML.getElementsByTagName("employees")[0];
if (employees.childNodes.length > 0) {
     completeTable.setAttribute("bordercolor", "black");
     completeTable.setAttribute("border", "1");
} else {
     clearTable();
}
for (loop = 0; loop < employees.childNodes.length; loop++) {
     var employee = employees.childNodes[loop];
     var firstName = employee.getElementsByTagName("firstName")[0];
     var lastName = employee.getElementsByTagName("lastName")[0];
     var employeeId = employee.getElementsByTagName("id")[0];
     appendEmployee(firstName.childNodes[0].nodeValue,lastName.childNodes[0].nodeValue
     , employeeId.childNodes[0].nodeValue);
}
}
```

Frameworks: jQuery

- Cross-platform JavaScript library supporting AJAX
  - Most popular library in use today
  - see others at: http://en.wikipedia.org/wiki/List_of_Ajax_frameworks
- To use jQuery, it must be included in the HTML
  - Typically done in the <head> area of a page:

```
<head>
<title>App Engine - HTML</title>
<link href="/static/glike.css" rel="stylesheet" type="text/css" />
<script type="text/javascript" src="/static/js/jquery-1.2.6.min.js"></script>

</head>
```

  - Aimed at simplifying client-side scripting of HTML
  - DOM element selections, traversal and manipulation—enabled by its *selector engine* ("Sizzle"), JSON parsing
  - Extensible (plug-ins), notably: jQuery UI, plug-in for abstracting advanced effects, animations, themable widgets, etc.
  - Programming style fuses algorithms and DOM-data-structures
    - Functions can be chained as they all return jQuery objects
- Two usage styles
  - the $ function is a factory method for the jQuery object
  - $.-prefixed functions, do not act directly on jQuery object

jQuery: $ function

- The $ funtion is also called a command
  - Alias for jQuery object
  - Once executed, the DOM of a document is available:

```
jQuery(document).ready(function() {
   // DOM is entirely defined here …
}
```

or:

```
$(document).ready(function() {
   // DOM entirely defined here …
});
```

- Typically used to access and manipulate multiple DOM nodes
  - command may contain a CSS selector string: $(selector)
    - Selector may refer to tag name or #ID or .CLASS (class attribute of a tag) or * for all tags
    - Selector may also refer to tags with an attribute: [attr] or based on its value: [value="val"]
  - Results in the jQuery object matching elements in the HTML page
  - Methods can then be called on the jQuery object or on nodes themselves

- Example: find HTML SELECT element with ID="carmakes" and add an OPTION element with value "VAG" and text "Volkswagen":

```
$('select#carmakes').append($('<option />').attr({value:"VAG"}).append("Volkswagen"));
```

jQuery: $.-prefixed functions

- $.- or $(…)-prefixed functions also called utility functions
  - Example: .html('text') to extend the HTML, .css() to update style, etc.
- Typically used to implement browser independent AJAX queries and to manipulate remote data:
  - $.ajax function and its associated methods
  - Also $.get(), $.post(), $.getScript(), $.getJSON()

Example: posting data to server and providing feedback to user:

```
$.ajax({
  type: "POST",
  url: "example.php",
  data: "name=John&location=Boston"
}).success( function(msg) {
  alert( "Data Saved: " + msg );
}).fail( function( xmlHttpRequest, statusText, errorThrown ) {
  alert(
    "Your form submission failed.\n\n"
      + "XML Http Request: " + JSON.stringify( xmlHttpRequest )
      + ",\nStatus Text: " + statusText
      + ",\nError Thrown: " + errorThrown );
});
```

Example: updating messages sent over a chat application:

```
<div id="chatcontent">    Loading... </div>

<script> /* <![CDATA[ */
function updateMsg() {
  $.ajax({
     url: "/messages",
     cache:false,
     success:function(frag){$("#chatcontent").html(frag);}});
  setTimeout('updateMsg(),4000);
}
updateMsg();
/* ]]> */ </script>
```

AJAX: Conclusions

- URLs: minor issues
  - History in the browser
  - bookmarking pages
    - Not a problem if AJAX implements a full-fledged application (e.g. a spreadsheet)
  - Indexing by search engines (SEO)
- Solutions:
  - The URL anchor(#) can possibly be modified as a solution
  - Unique URL pattern
  - Also solved by mature APIs: HTML5 History API, jQuery BBQ: Back Button & Query Library, PathJS lib …

AJAX: Conclusions

- Security = major issue:
  - Additional browser firewalling is required (e.g. Google Chrome)
  - cross-site vulnerabilities: XSS, CSRF
  - Malicious client
  - Browser and sandbox implementation errors
- Now mature technology
  - Responsive, offers a nice interactivity
  - used to implement applications on web platforms
  - Plenty of available frameworks to support its deployment
    - E.g. the GWT SDK for the development of browser-based web application
- Major advantage: it can be deployed everywhere