Università della Svizzera italiana

Faculty of Informatics

AJAX

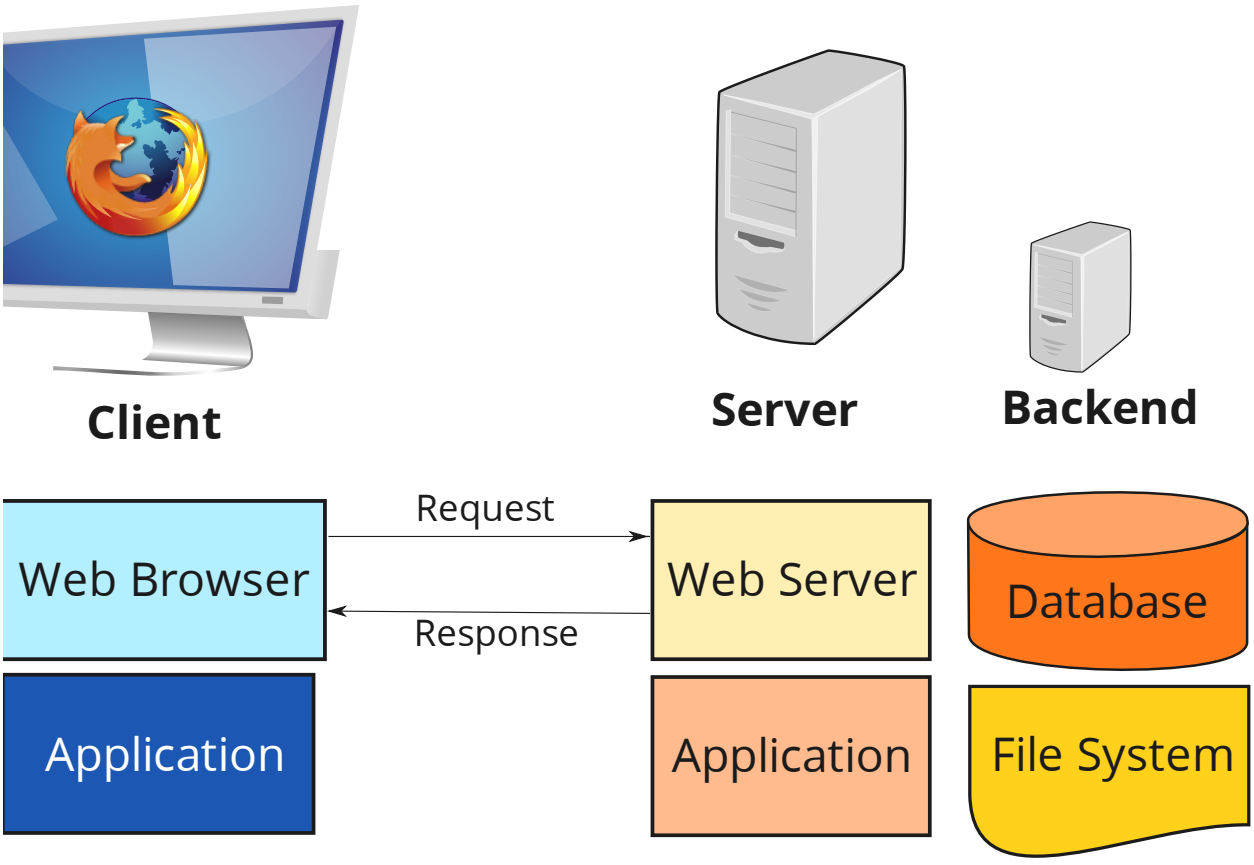
Prof. Cesare Pautasso

http://www.pautasso.info

cesare.pautasso@usi.ch

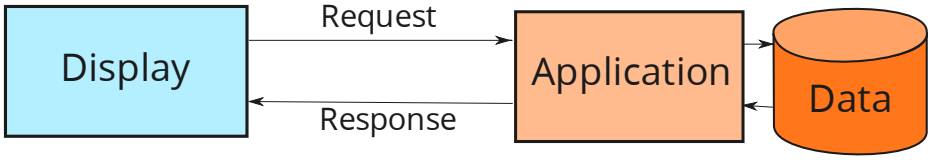
@pautasso

Web application Architecture

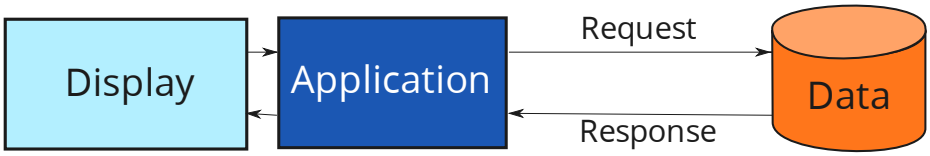


Very Thin Client

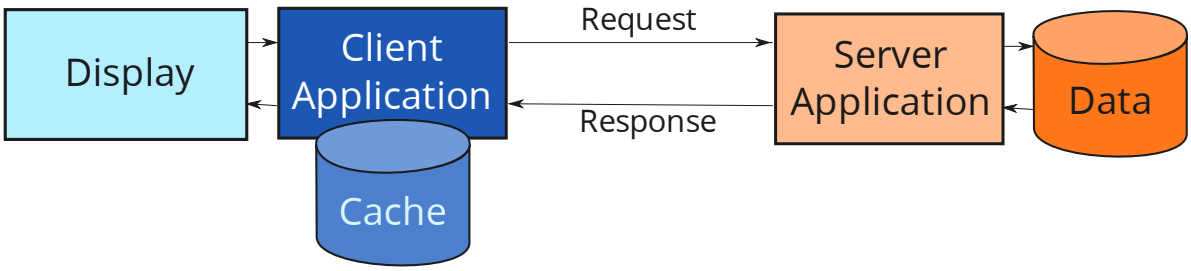
Client/Server



Rich Client



General Architecture



Rich vs. Thin Client

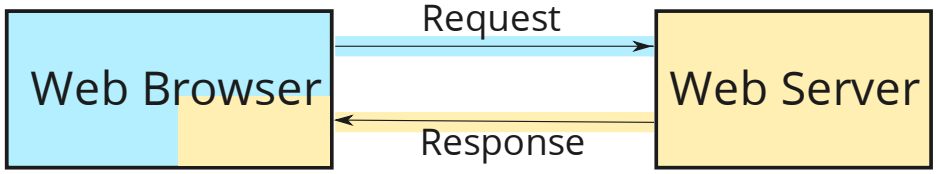
Rich Client

- Applications runs on the client (may use the server for storage)
- Platform Examples:
  - Windows, MacOS/X
  - Eclipse RCP/Java
- Software needs to be deployed on the client
- Zero latency
- Complete control, native access to the platform

Thin Client

- Application runs on the server, client only perform UI tasks
- Examples:
  - Dumb Terminals
  - Web 1.0 Applications
- Zero deployment/upgrade costs
- Cannot (yet) work offline: sensitive to network failures
- Limited control of the platform

What about security?



Web servers should not ever run any code sent by a Web browser

- Web browsers use a sandbox (secure virtual machine) to run code downloaded from a Web server

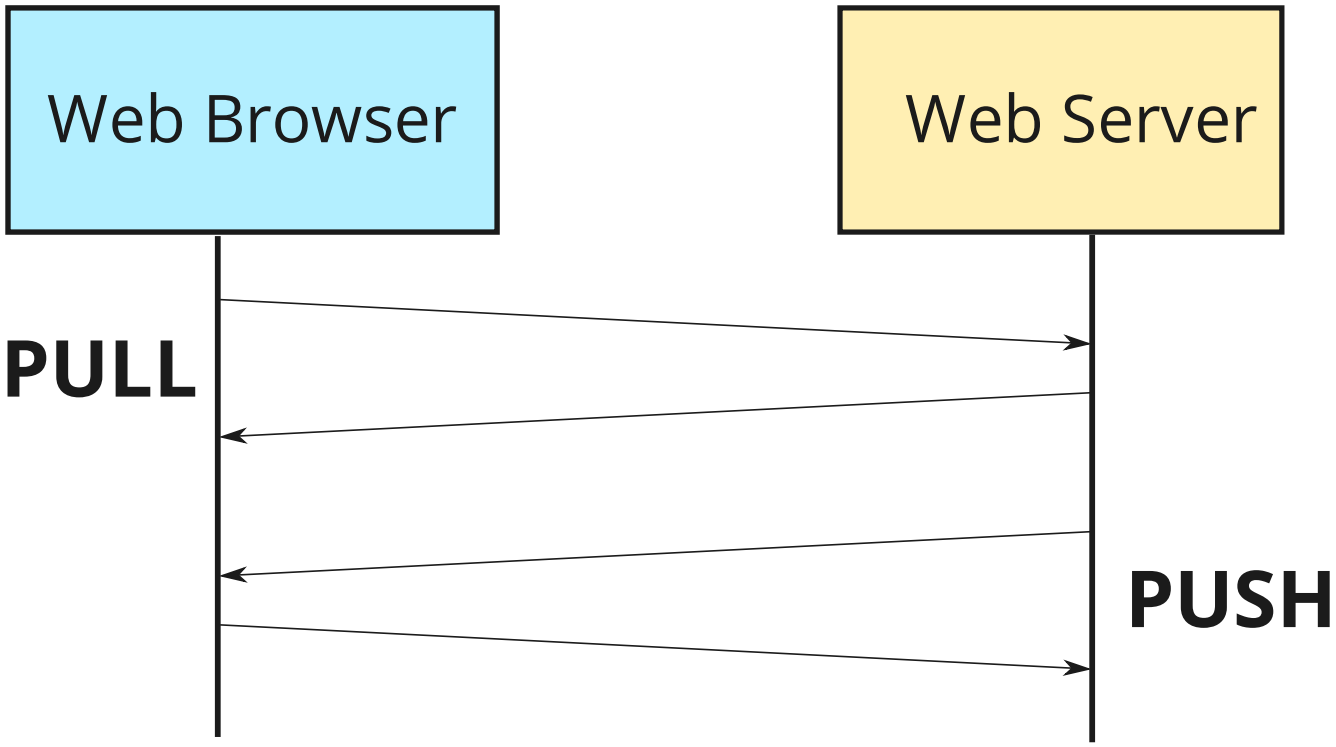
Interconnect

How to connect the client with the server?

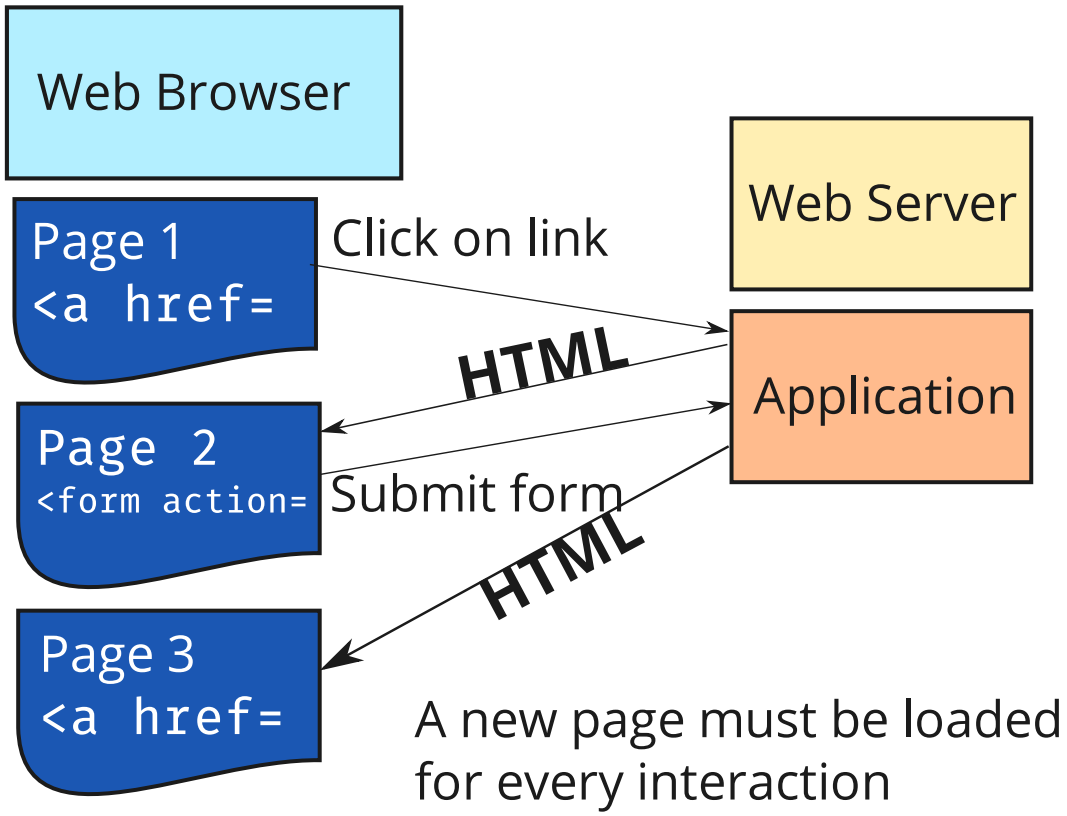
- Send user commands and input data as HTTP requests from the client to the server

How to connect the server with the client?

- **Pull**: Fetch and refresh output data
- **Push**: Notify client about state changes



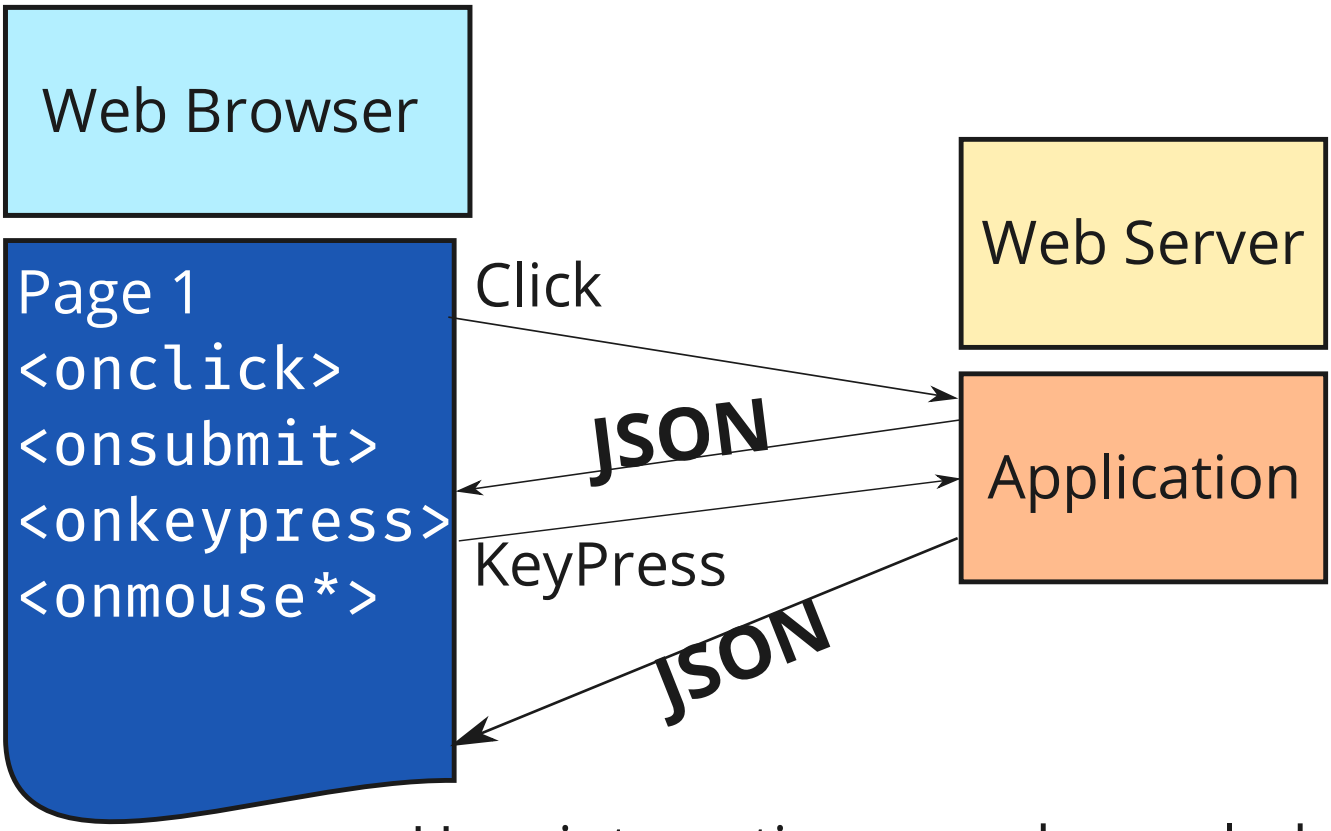
Web 1.0 Architecture



Web 1.0 Architecture - Problems

- UI not Responsive
  - The entire UI must be refreshed for every interaction, even if only parts of it need to be updated
  - The browser is blocked until the new page is downloaded from the server
- Server unnecessarily busy rendering Web pages in HTML when it could be just sending JSON and offload the rendering to the browser

Web 2.0 Architecture



User interactions are decoupled from client/server interactions

Advantages

- When the user interacts with the application we send a JSON/XML request to the server and receive a JSON/XML response back.
- The HTML rendering is done on the client
- JSON is faster, smaller, cheaper to encode, send and decode compared with XML/HTML
- Clients do not have to download the entire data but can fetch the data they need when they need it

Problems

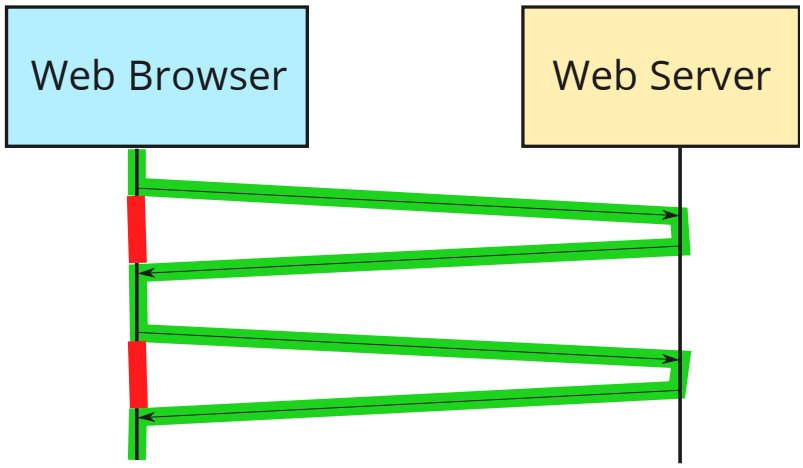
- Since the whole application runs in the same page:
  - Back button broken
  - Cannot bookmark current "state" of the application (could use URI #fragments)
- HTTP connections are expensive:
  - Do not poll the server too often
  - Browsers limit the number of parallel connections to the same server

AJAX

AJAX combines different technologies:

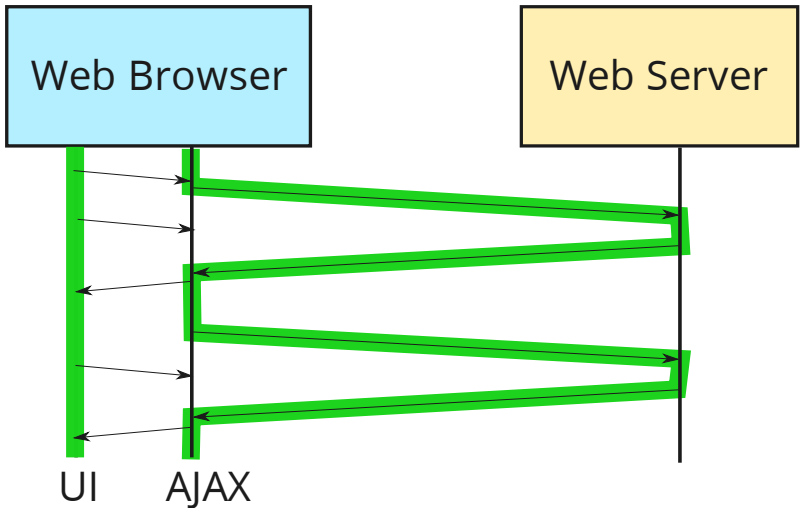
- HTML5 and CSS in the display
- Dynamic display and interaction with DOM
- Data interchange and manipulation using XML/XSLT
- Asynchronous data retrieval with XMLHttpRequest
- Javascript binding everything together

Synchronous Interaction



The user waits for the server to process each request

Asynchronous Interaction



The UI thread is never blocked since server interactions run in the background

XMLHttpRequest (GET, Synch)

```
function GET(url) {
  var xhr = new XMLHttpRequest();
  xhr.open("GET", url, false);
  xhr.send(null);
  //this will continue after the response has arrived
  if (xhr.status == 200)
    return xhr.responseText;
  else
    //handle error
}
```

`false` = synchronous

`responseText` contains the JSON string to be parsed

XMLHttpRequest (GET, Asynch)

```
function GET(url, callback) {
  var xhr = new XMLHttpRequest();
  xhr.open("GET", url, true);
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
      if (xhr.status == 200)
        callback(xhr.responseText);
      else
        //handle error
    }
  }
  xhr.send(null);
  //this will continue immediately
}
```

`true` = asynchronous

readyState	
0	uninitialized
1	opened
2	sent
3	receiving
4	complete

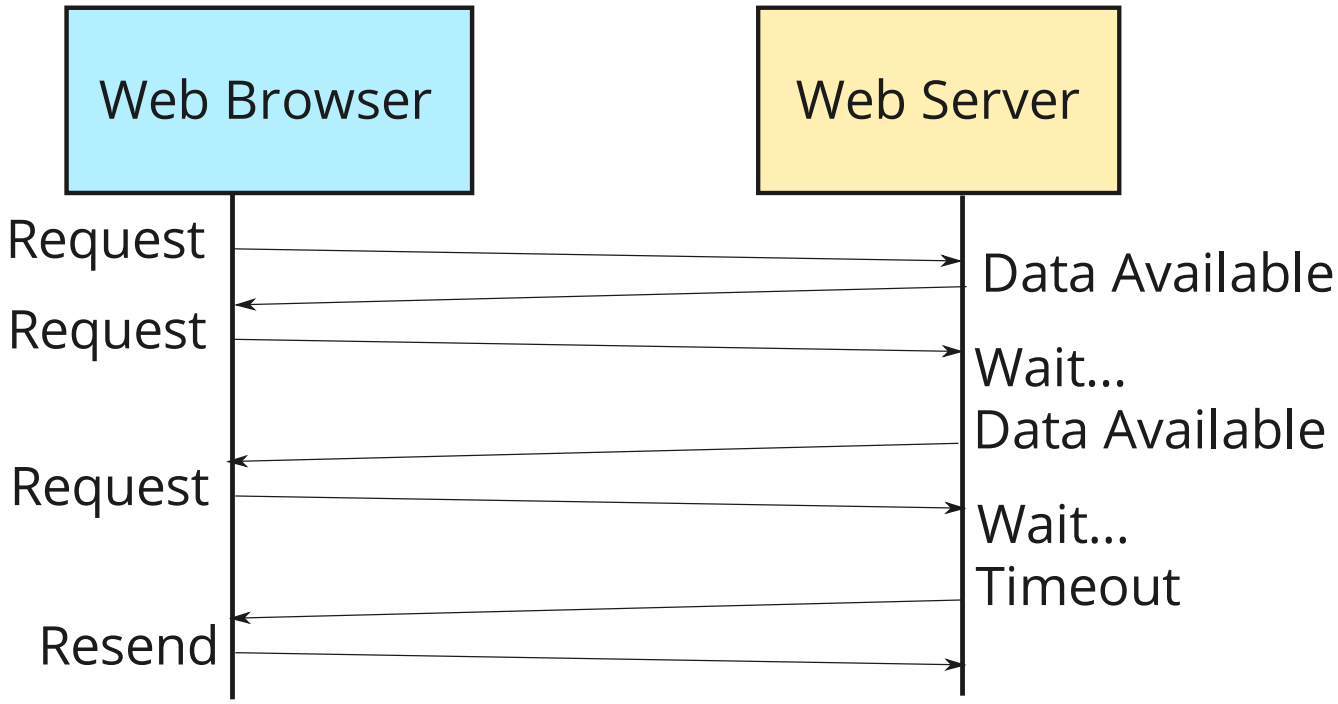
XMLHttpRequest (POST, Asynch)

```
function POST(url, params, callback) {
  var xhr = new XMLHttpRequest();
  xhr.open("POST", url, true);
  xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
      if (xhr.status == 200)
        callback(xhr.responseText);
    }
  }
  xhr.setRequestHeader("Content-Type",
                       "application/x-www-form-urlencoded");
  xhr.send(params);
  //this will continue immediately
}
```

Event Notification

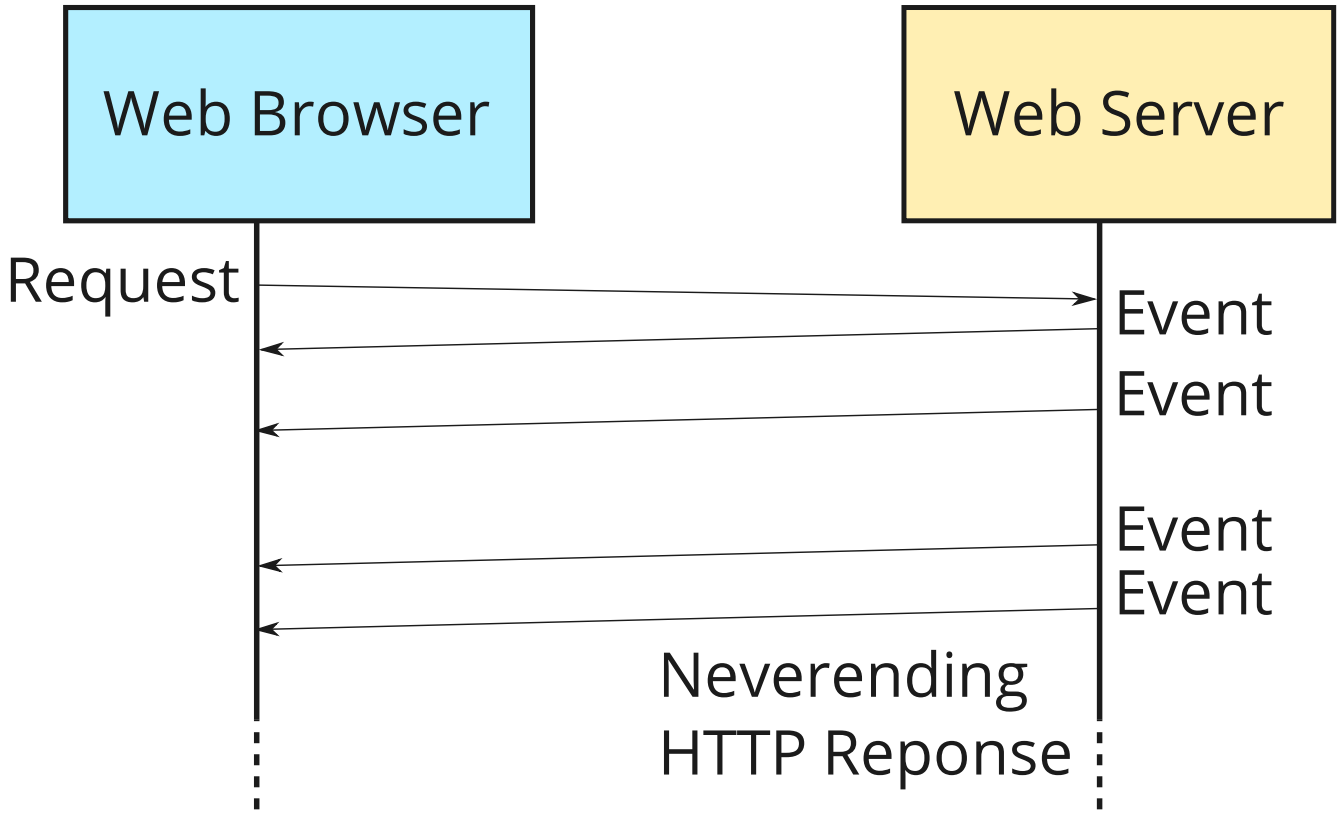
- XMLHttpRequest helps to read data from the server when the browser needs to refresh parts of a Web page
- How can the server decide when to make the browser update its Web page?
  - Long Polling
  - Server push
  - WebSockets

Long Polling



The client polls the server for updates which are sent only when they become available

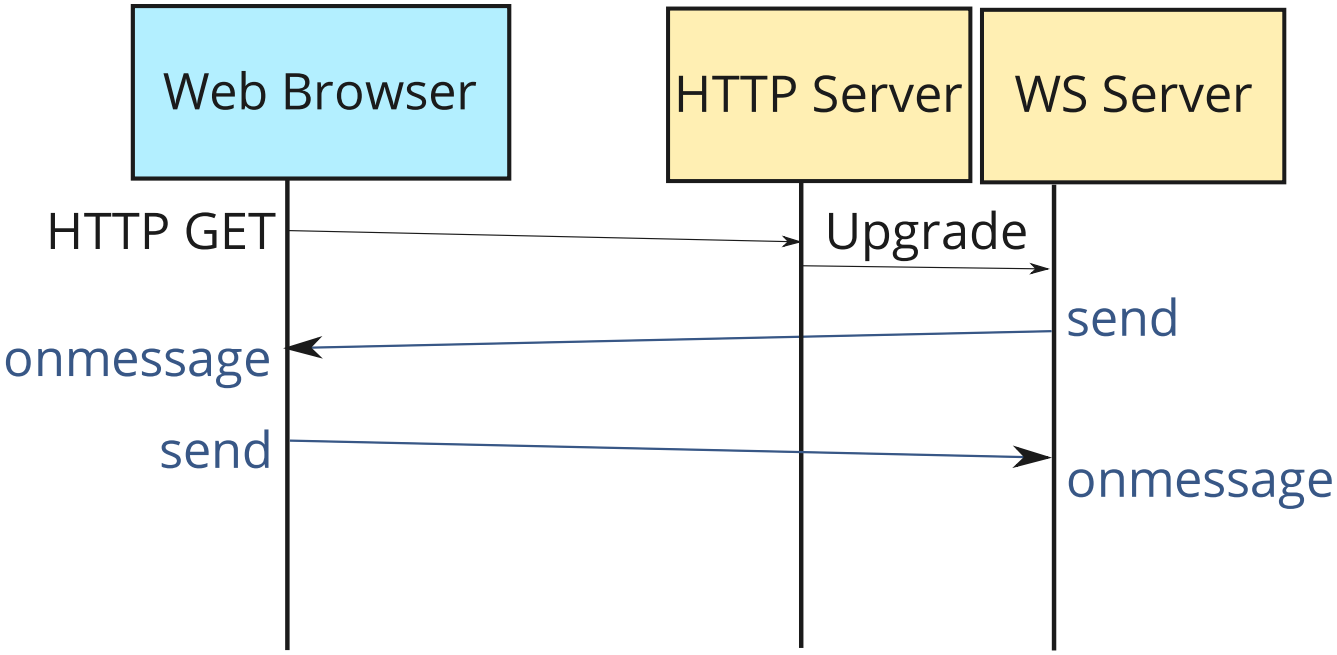
Server Push



The server misuses HTTP by keeping the response open forever

HTML5 WebSockets

- Bi-directional, full-duplex communication channel
- Web browsers connect to Web servers and exchange messages
- Optimized with low overhead for message payloads (2 bytes)
- Just a socket for the browser, nothing to do with the Web



The original HTTP connection is upgraded to use the WebSocket protocol

WebSocket (Client)

```
var location = "ws://www.nyse.com/GOOG";
//open a WebSocket connection with the server
var socket = new WebSocket(location);
socket.onopen = function(event) {
  //connection established
  //send a message to the server
  socket.send("Hello, WebSocket");
}
socket.onmessage = function(message) {
  //message received from the server
  console.log(message.data);
}
socket.onclose = function(event) {
  //connection closed by the server
  console.log("closed");
}
socket.onerror = function(event) {
  //communication error
  console.log("error!"+event);
}
```

WebSocket (Server)

```
var WebSocketServer = require('websocket').server;
var http = require('http');

var server = http.createServer(
function(request, response) {
  console.log('HTTP Request: ' + request.url);
  response.writeHead(404);
  response.end();
});
server.listen(8888);

// Create a WebSocket Server wrapping the HTTP Server
wsServer = new WebSocketServer({
    httpServer: server
});

//Check if the request origin is allowed to connect
function originIsAllowed(origin) { return true; }

wsServer.on('request', function(request) {
  if (!originIsAllowed(request.origin)) {
    request.reject(); return;
  }
  // Connection Accepted
  var connection = request.accept(null, request.origin);
  var echo = function(message) {
    if (message.type === 'utf8') {
      console.log('Received: ' + message.utf8Data);
      connection.sendUTF(message.utf8Data);
    }
    else if (message.type === 'binary') {
      console.log('Received binary data');
      connection.sendBytes(message.binaryData);
```

```
    }
  }
  connection.on('message', echo);
  connection.on('close',
    function(reasonCode, description) {...});
});
```


References

- Gottfried Vossen, Stephan Hagemann, Unleashing Web 2.0 – From Concepts to Creativity, Morgan Kaufmann, 2007
- Paul Graham, The Other Road Ahead (On the advantages of Web applications) (http://www.paulgraham.com/road.html) , September 2001.
- Adam Bosworth, Why AJAX Failed (Then Succeeded) (http://www.eweek.com/c/a/IT-Infrastructure/Googles-Bosworth-Why-AJAX-Failed-Then-Succeeded/) , Jan 2007
- Jesse James Garrett, Ajax: A New Approach to Web Applications (http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications) , Feb 2005
- Chris Anderson, The Long Tail: Why the Future of Business is Selling Less of More (http://www.thelongtail.com) , July 2006,
- Tim O'Reilly, What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software (http://oreilly.com/web2/archive/what-is-web-20.html) , Sept 2005