# Introduction to AJAX

# HTTP request message

- two types of HTTP messages: *request*, *response*
- HTTP request message:
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

```
GET /somedir/page.html HTTP/1.1
Host: www.mywebsite.com
User-agent: Mozilla/4.0
Connection: close
Accept-language:En
```

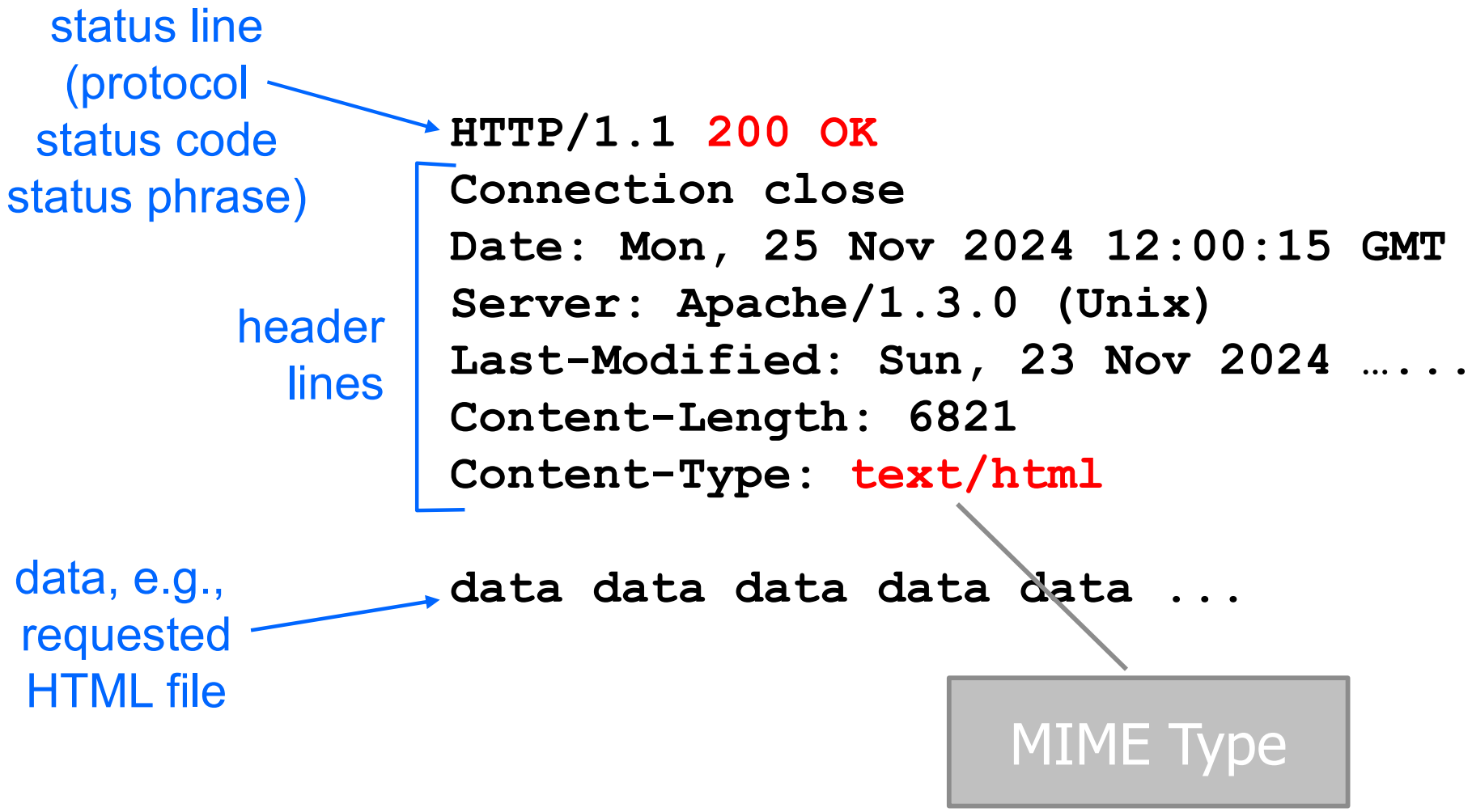
header lines

non-persistent

Carriage return, line feed indicates end of message

(extra carriage return, line feed)

# HTTP response message




# Developers' need

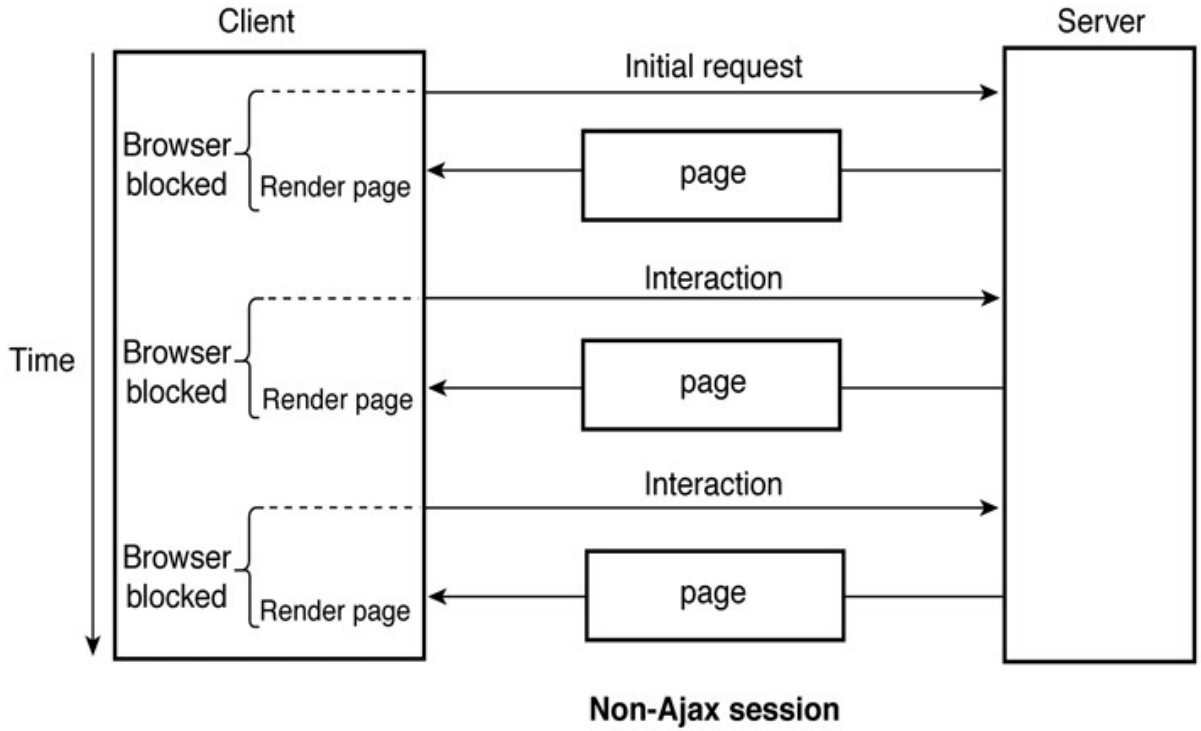
- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Send data to a server - without submitting a form
- **In the background, transparent to the user**

# AJAX

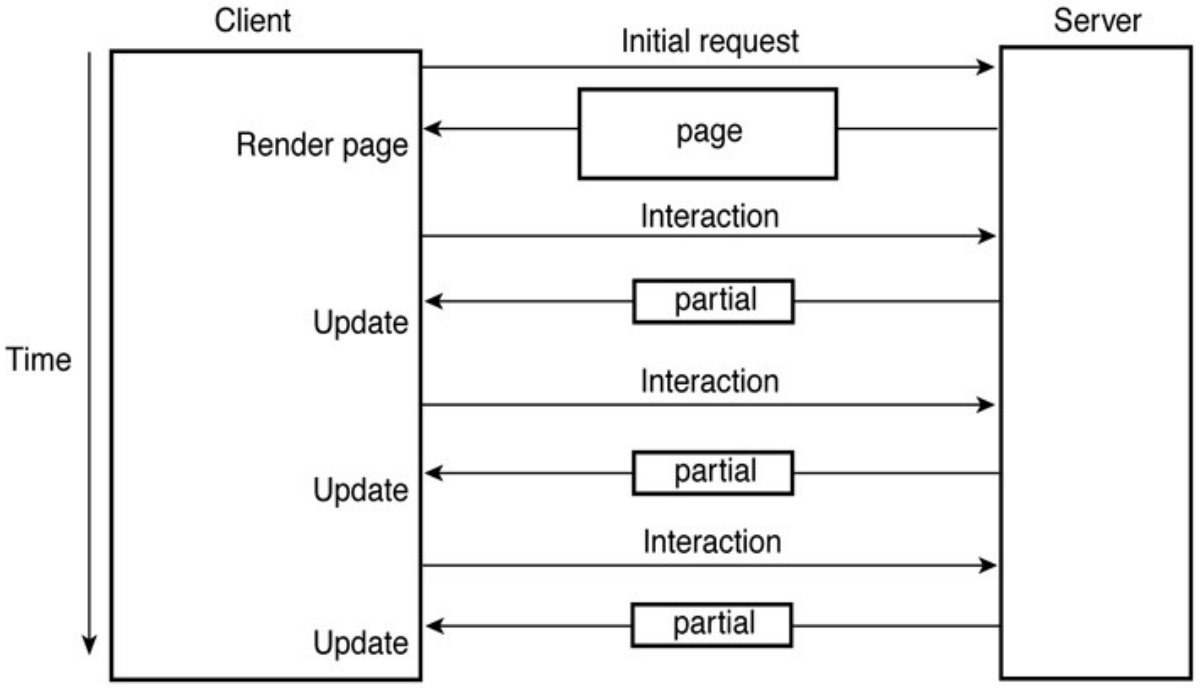
- AJAX: Asynchronous JavaScript and XML
- **AJAX is not a programming language**
- Goal of Ajax: provide Web-based applications with responsiveness approaching that of desk-top applications
- Microsoft's `XmlDocument` and `XMLHTML` ActiveX objects in IE5 – for asynchronous requests
- Now, most modern browsers support `XMLHttpRequest`
- Google Maps and Google Mail
- AJAX applications might use **XML** to transport data, can also use **plain text** or **JSON**

# Overview of Ajax

- Goals:
  - Client requests are handled asynchronously
  - Only small parts of the current document are updated
- Specific kind of Web applications that benefit from Ajax are those that have frequent interactions between the client and the server
- Ajax does not need any new language(s):
  - Client side: JavaScript, XHTML, DOM, CSS, XML
  - Server side: any that can handle HTTP requests
- the `XMLHttpRequest` object is used




**Non-Ajax session**




**Ajax session**

# A typical scenario

- A form gathers client information:
  - asks for the zip code before the names of the city and province
- As soon as the zip code is entered, the application sends a request to the server, which looks up the province and city for the given zip code and returns them to the form
- Uses Express on the server to look up the city and province
- Uses JavaScript to put the city and province names in the form

# The basics of Ajax

- Two functions are required by the application:
  - An event to trigger a function (blur for example)
  - A function to handle the response

# AJAX - The XMLHttpRequest Object

- The keystone of AJAX is the XMLHttpRequest object.
- The XMLHttpRequest object can be used to exchange data with a server behind the scenes.
  - Asynchroneously
- This way, parts of a web page can be updated, without reloading the whole page.
- All modern browsers support the XMLHttpRequest object.

# The onreadystatechange Property

- The **readyState** property holds the status of the XMLHttpRequest.
- The **onreadystatechange** property defines a function to be executed when the readyState changes.
- The status property and the **statusText** property hold the status of the **XMLHttpRequest** object.

# readyState values

- 0: request not initialized
- 1: server connection established
- 2: request received
- 3: processing request
- **4: request finished, and response is ready**

# status values

HTTP status messages

- **200: "OK”**
- 403: "Forbidden”
- 404: "Page not found”
- …

# Check if response is ready

- The onreadystatechange function is called **every time the readyState changes**.
- When readyState is **4** and status is **200**, the response is ready to be used
- We usually test for these before updating the document based on the response

# The Request Phase

- Create an asynchronous request object through the XMLHttpRequest object

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

- Register the call back function

```
xhr.onreadystatechange = receiveResponse
```

- Call the open methods of the xhr object

```
xhr.open("GET",
        "/updatestudent/param1/param2", true)
```

  - HTTP method, GET or POST, quoted
  - The URL of the response document on the server
  - A Boolean literal to indicate whether the request is to be asynchronous (true) or synchronous (false)

# The Request Phase

- The request is sent with the send method

```
xhr.send(null);
```

- The function

```
function receiveResponse(){

  // check if readyState is 4 and status is 200

  // get the response from xhr (e.g., responseText)

  // update the document
}
```

# The Response Document

- The server side can return the content of a static file, or a dynamic result generated by JavaScript, PHP, python…
- As long as it returns the **expected output** in the **expected format**

# The receiveResponse Phase

- Actions of receiver function
  - Put all actions in the then clause of a selector that checks to see if **readyState** is 4
  - Get the response value from the **responseText** property of the XHR object
  - Process the value
  - Update any UI widgets in the document

# The receiver phase

- A JavaScript function with no parameters
  - Fetch the server response (text), process it,, and set the corresponding UI elements with the received value(s)
- The receiver function must be able to access the XHR

```
xhr.onreadystatechange = function () {
    if (xhr.readyState == 4 && xhr.status == 200) {
      var result = xhr.responseText;
      // process result
      // update UI
      document.getElementById("someID").value = …

    }
```

# Cross-Browser Support

```
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp = new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
  }
```

- Not really needed anymore
- After all, who is still using IE(5, 6, or even later)

# Roadmap again

- An event occurs in a web page (the page is loaded, a button is clicked, a timer expires …)
- An XMLHttpRequest object is created by JavaScript
- The XMLHttpRequest object sends a request to a web server
- The server processes the request
- The server sends a response back to the web page
- The response is read by JavaScript
- Proper action (like page update) is performed by JavaScript

# Example

1. A web page that shows a greeting
2. At startup, the greeting is "Hello World!"
3. After 3 seconds, AJAX gets a new greeting from the server ("Bonjour tout le monde!")
   - The greeting on the server is in a text file: ajax.txt
4. A button on the page resets the greeting to "Hello World!" when pressed.
5. Go to point 3

# Express

```
const express = require("express");
const app = express();

// IMPORTANT: Tell Express app to stick to basic encoding
app.use(express.urlencoded({ extended: false }));

// Use the following folder for public/static content
app.use(express.static("public")); // scripts.js should be in this folder

app.get("/", (req, res) => {
  res.sendFile(__dirname + "/doc.html");
});

app.get("/updategreeting", (request, response) =>{
  response.send("Bonjour tout le monde");
})

const PORT = 5000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

# doc.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Greeting manipulation by AJAX</title>
</head>

  <body>
    <h1 id="greeting">Hello World!</h1>
    <button value="button" onclick="englishGreeting()">Put
back</button>

    <script type="text/javascript"
src="scripts.js"></script>
  </body>
</html>
```

# scripts.js

```
setInterval(frenchGreeting, 3000); // runs the function every 3 sec
                                   // just to simulate requests to the server
function frenchGreeting() {
  xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function () {
    if (xhr.readyState == 4 && (xhr.status = 200)) {
      result = xhr.responseText;
      document.getElementById("greeting").innerText = result;
    }
  };
  xhr.open("GET", "/updategreeting", true);
  xhr.send(null);
}
function englishGreeting() {
  var head = document.getElementById("greeting");
  head.innerText = "Hello World!";
}
```

# Ajax.txt

Bonjour tout le monde!

# JSON

- A textual way to represent objects
- Two structures
  - collections of name/value pairs
  - array of values
- A simpler alternative to XML

**JSON**

```
{ "xx" : "yyy" }
```

**XML**

```
< xx> yyy < /xx>
```

```
{ "weatherdetails" : "yyy" }
```

```
< weatherdetails>
    yyy
< /weatherdetails>
```

## JSON

```
{ "xx": {"yy":"nn"} }
```

## XML

```
< xx yy="nn"><
  /xx>
```

```
{ "weatherreport" :
  {
    "city" : " Montreal ",
    "weather" : ”-5"
   }
}
```

```
< weatherreport
  city="Montreal"
  weather=”-5" >
< /weatherreport>
```