Extroverted web applications

Lecture 6

With browser technologies we can

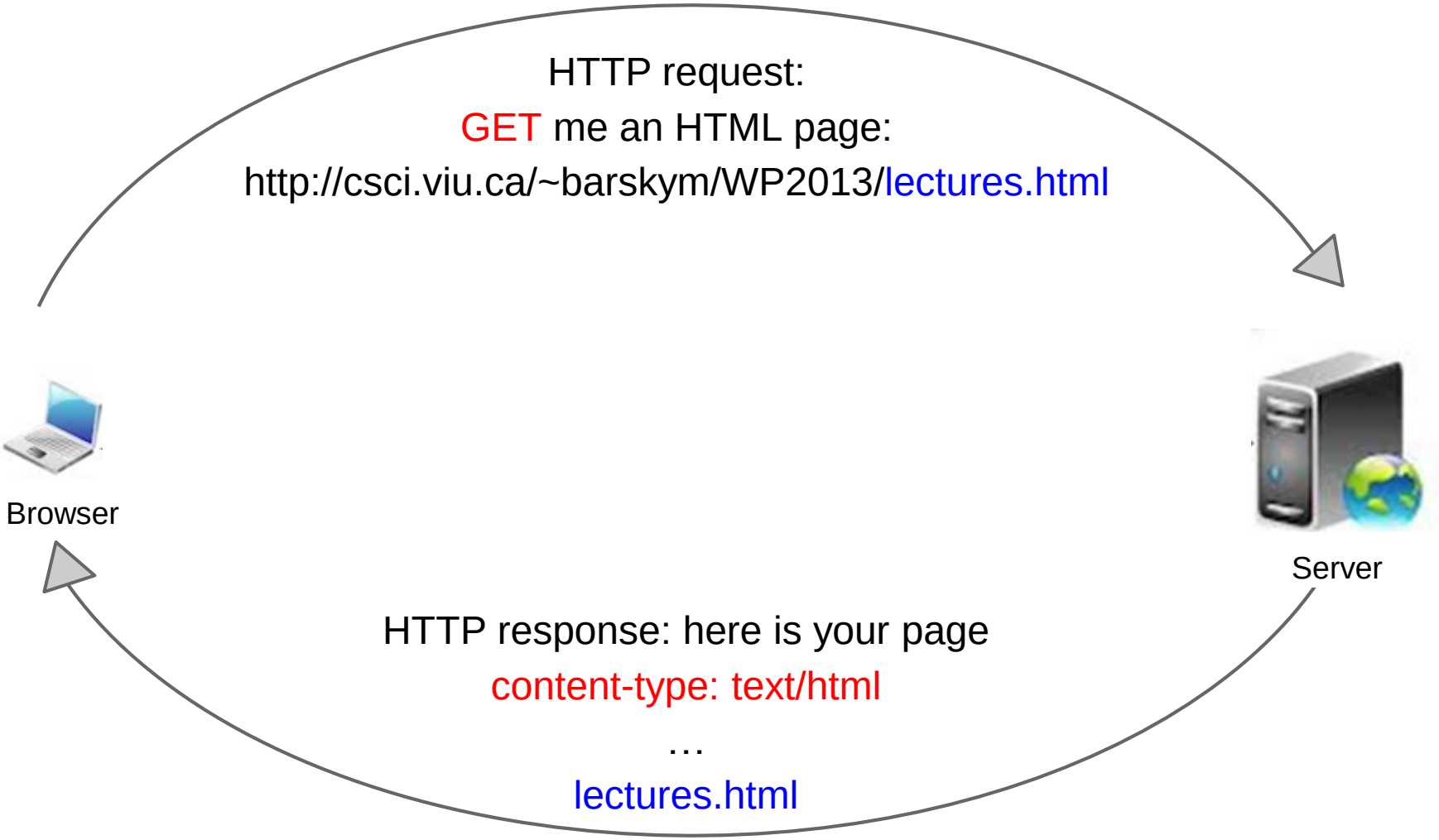
- Create GUI with *HTML5* and *CSS3*
- Dynamically manipulate page elements with *JavaScript*
- Remember data entered by user using *local storage*
- Visualize data using *canvas*
- Deliver applications to millions of potential users by uploading HTML/CSS/JavaScript files to a server

example

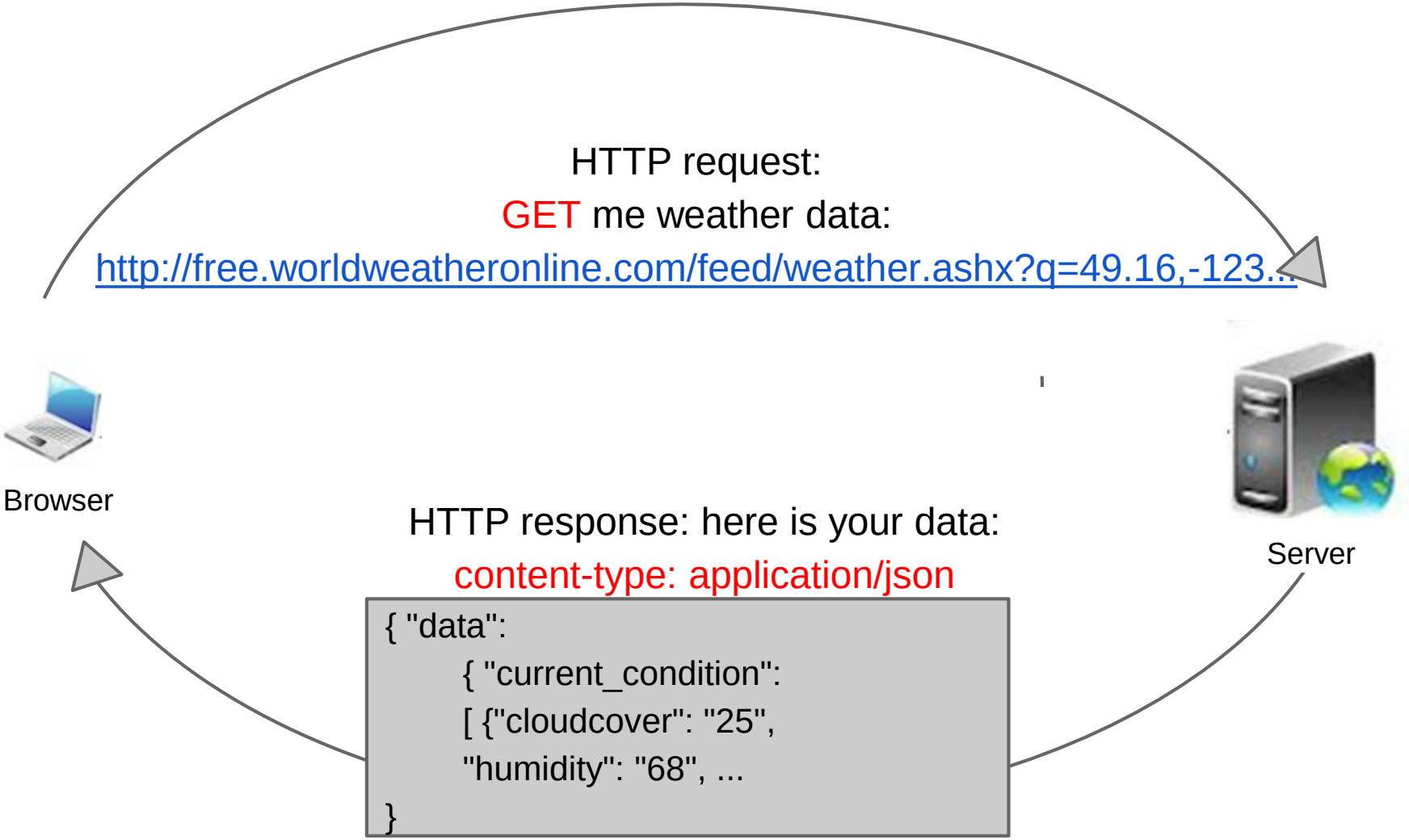
Today: new technology - AJAX

- Dynamically update a page using *external* data fetched from a web server
- Get external data directly into JavaScript code
- Incorporate new data into web page

Browsers can request *pages*



Browsers can also request *data*



Plan

1. Syntax of Ajax requests
2. Asynchronous program execution
3. Data formats

Issuing data request in JavaScript

Issuing request:

```
var ajax= new XMLHttpRequest();
ajax.open ("GET", "books.xml", false);
ajax.send (null);
```

When data arrives:

```
var outputElem = document.getElementById("output");
outputElem.value = ajax.responseText;
```

example

Meet Ajax

AJAX (Asynchronous JavaScript and XML)

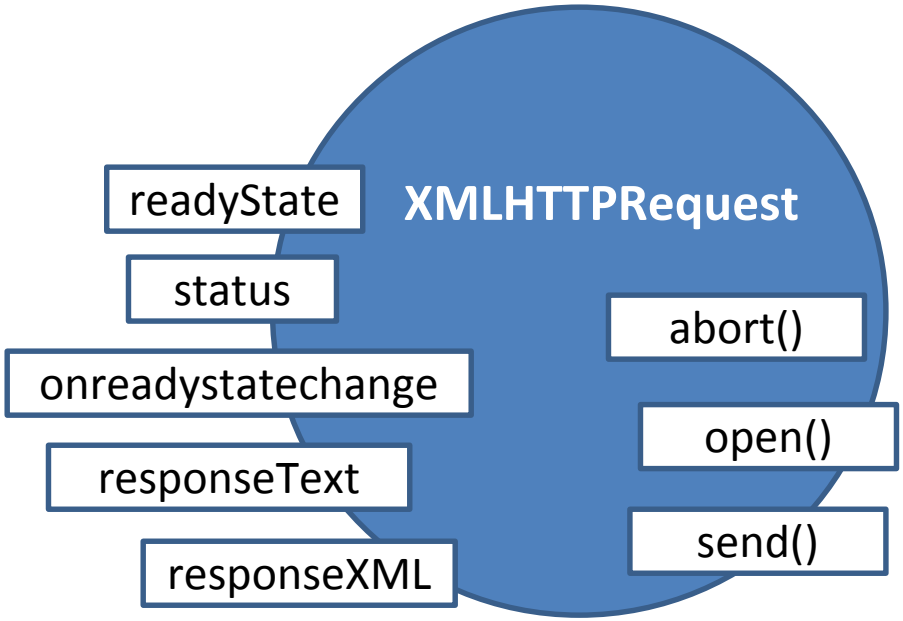
- not a programming language: a particular way of using client-side JavaScript to get data from a server
- downloads data from a server in the background
- allows dynamically updating a page without making the user wait: avoids the "click-wait-refresh" pattern

Widely used: Outlook web access, GMail, Google maps ...

How it works

1. The browser sends a request for data to a server and waits for the response
2. The server receives the request and creates the response by packaging the data in XML
3. The entire XML file is returned to the browser in the Ajax response
4. The browser unpacks the response data and carefully incorporates it into the page using JavaScript and DOM

XMLHTTPRequest object: properties and methods

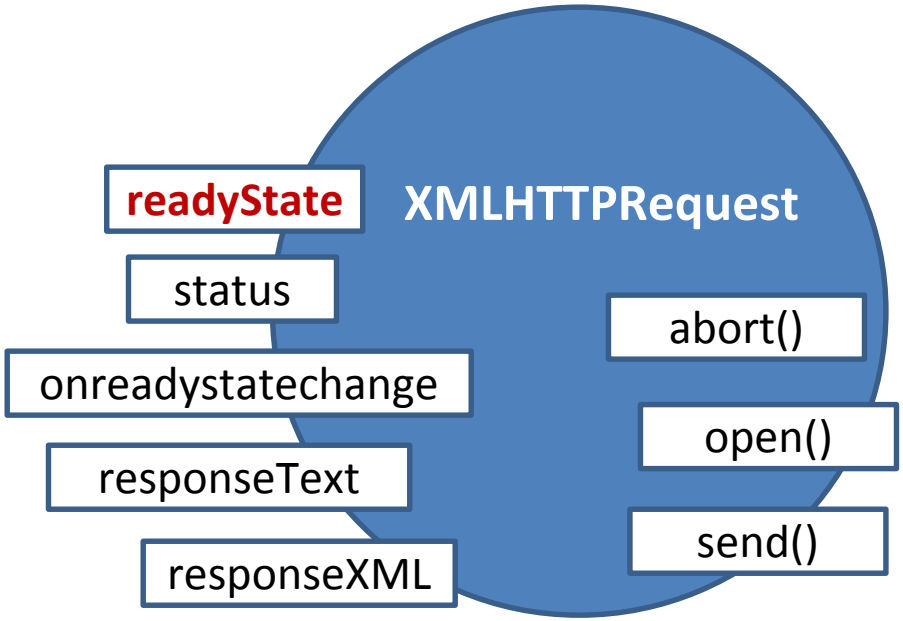


XMLHTTPRequest: properties

readyState

The numeric state of the request:

- 0 (uninitialized)
- 1 (open)
- 2 (sent)
- 3 (receiving)
- 4 (loaded)

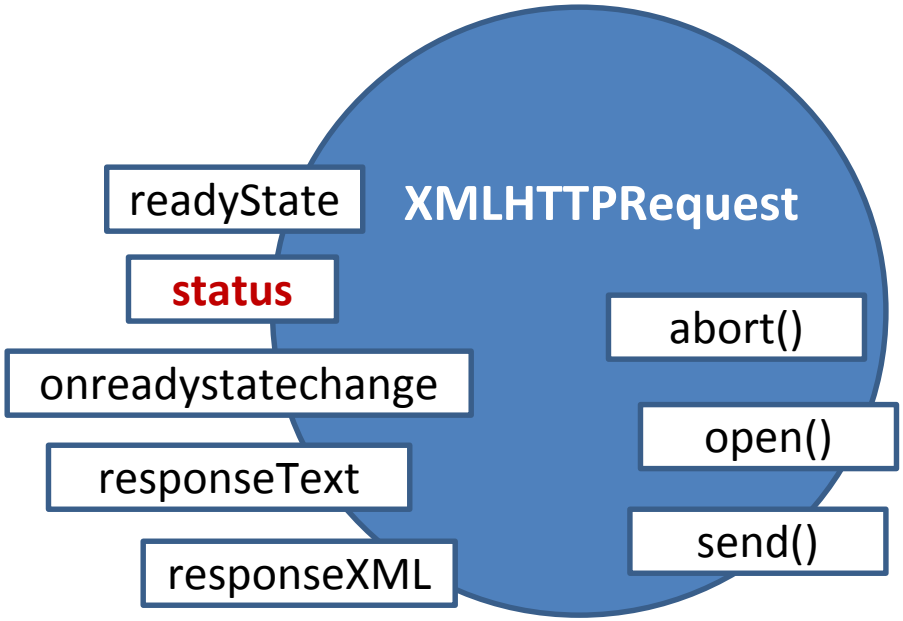


XMLHTTPRequest: properties

status

The HTTP status code of the request, such as

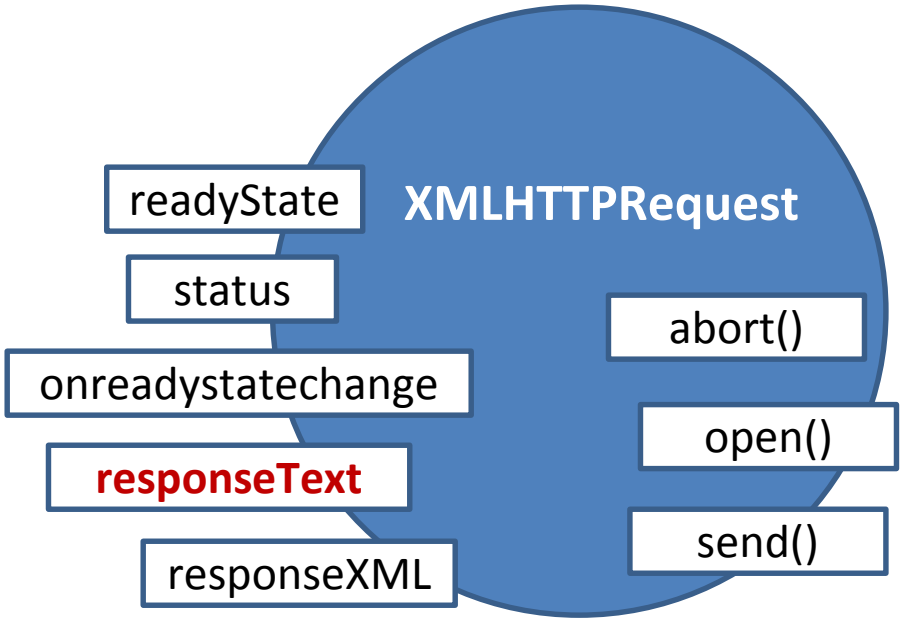
- 404 (not found)
- 200 (OK)



XMLHTTPRequest: properties

responseText

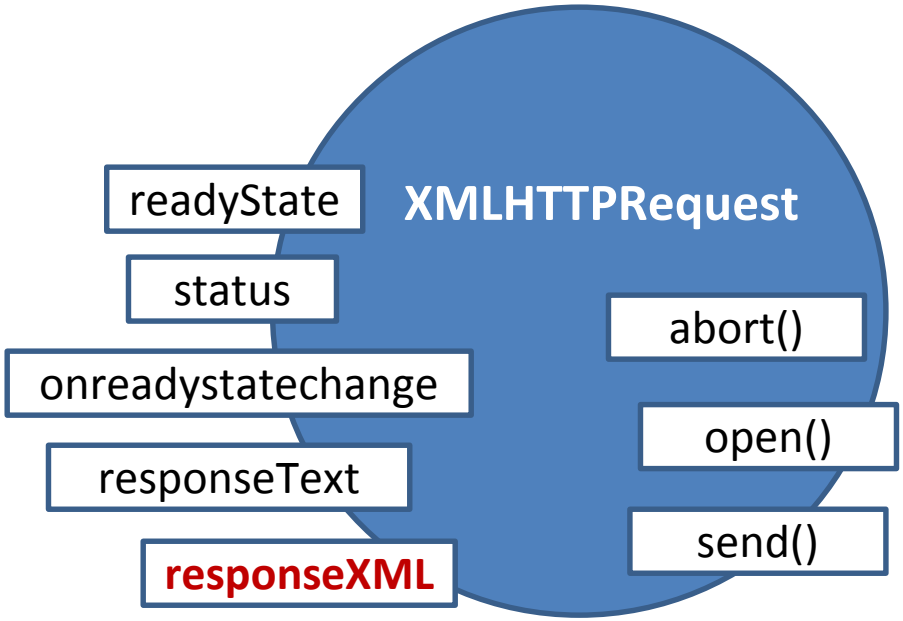
The response data returned from the server, as a string of plain text.



XMLHTTPRequest: properties

responseXML

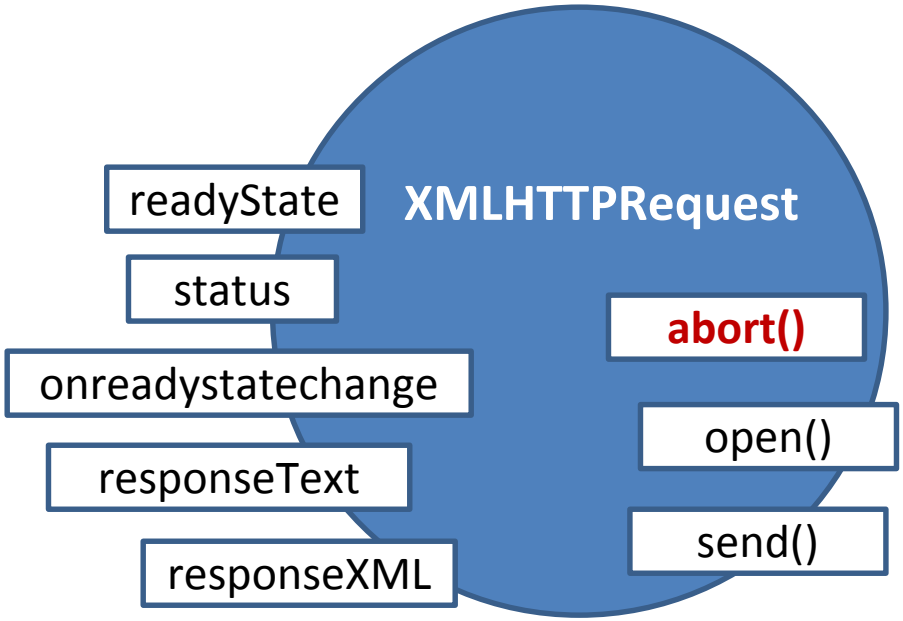
The response data returned from the server, as an object consisting of a tree of XML nodes



XMLHTTPRequest: methods

abort()

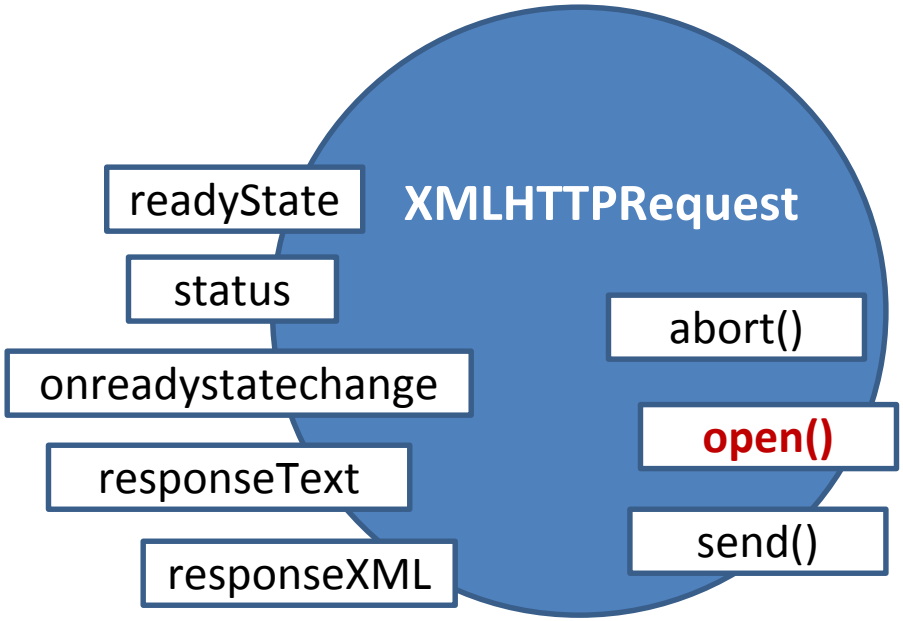
Cancel the request



XMLHTTPRequest: methods

open()

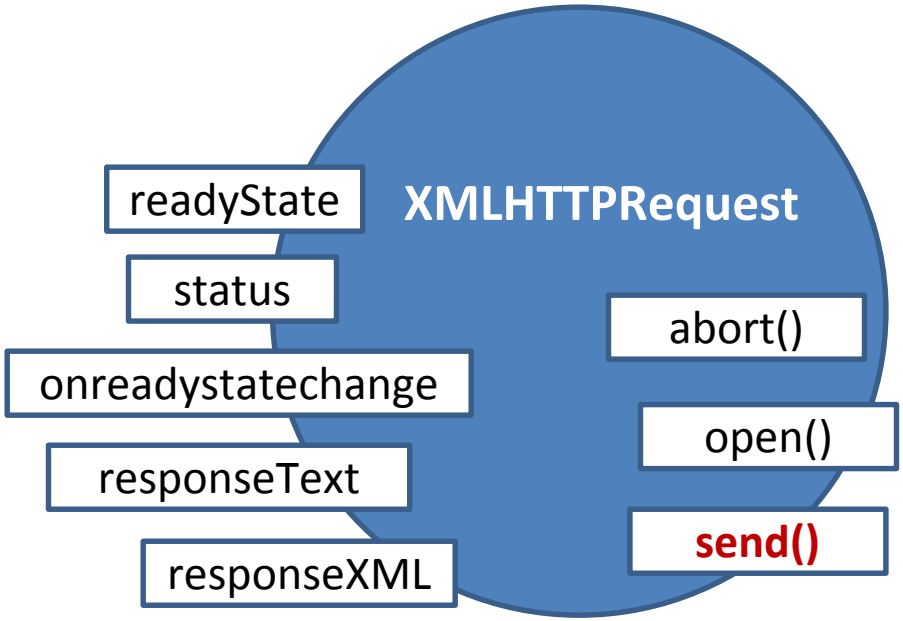
Prepare a request by specifying its type and URL, among other things.



XMLHTTPRequest: methods

send()

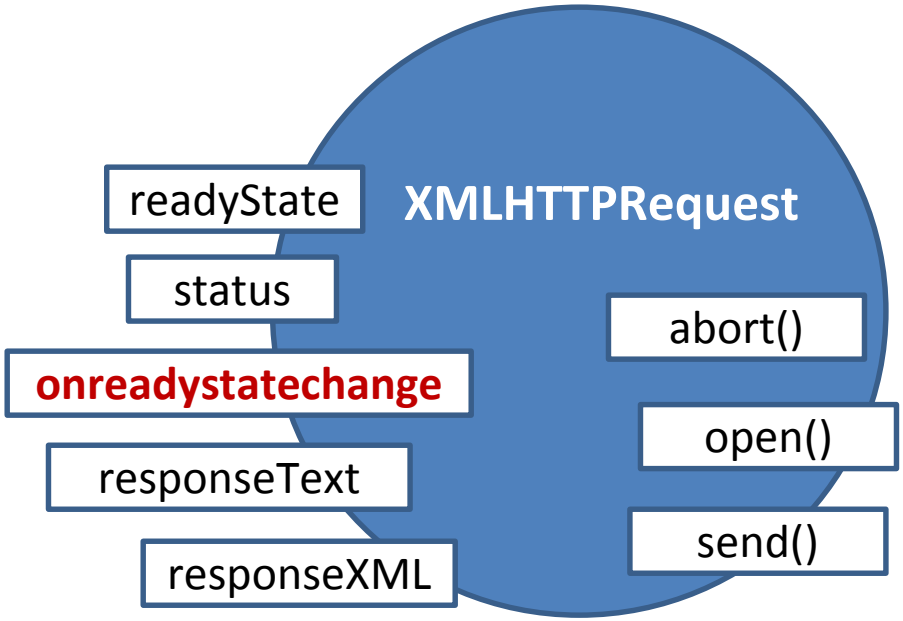
Send the request to the server for processing.



XMLHTTPRequest: properties

onreadystatechange

This property holds a reference to the custom event handler function that is called when the readyState of the Ajax request changes




```
var ajax= new XMLHttpRequest();

//attach event handler function to be executed when state changes
ajax.onreadystatechange = function (){
        if (ajax.readyState === 4)
                displayResults (ajax.responseXML);
```

Coding Ajax applications: example

```
function load()      {
            var url = "books.txt";

            try      {
                     downloadData (url, displayData);
            }
            catch (e)      {
                     alert (e.title + "\n" +e.message);
            }
      }
```

callback function

In *downloadData*:

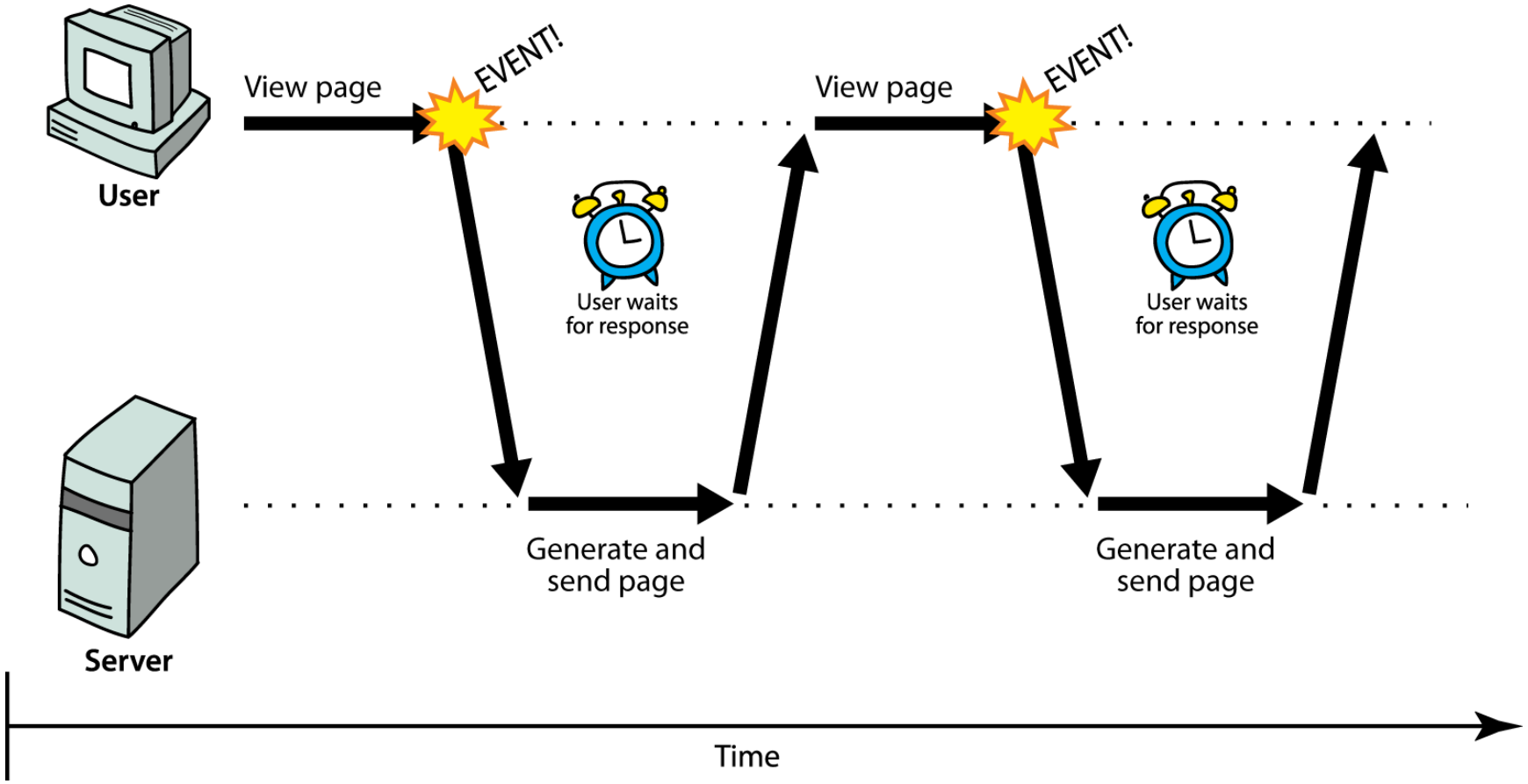
```
function downloadData (url, callbackFunction)      {
        var ajax= new XMLHttpRequest();

        //attach event handler to be executed
        //when event fires
        ajax.onreadystatechange = function ()      {
                if(ajax.readyState === 4)
                        callbackFunction (ajax);
                                                   };

        ajax.open("GET", url, true);
        ajax.send (null);
}
```

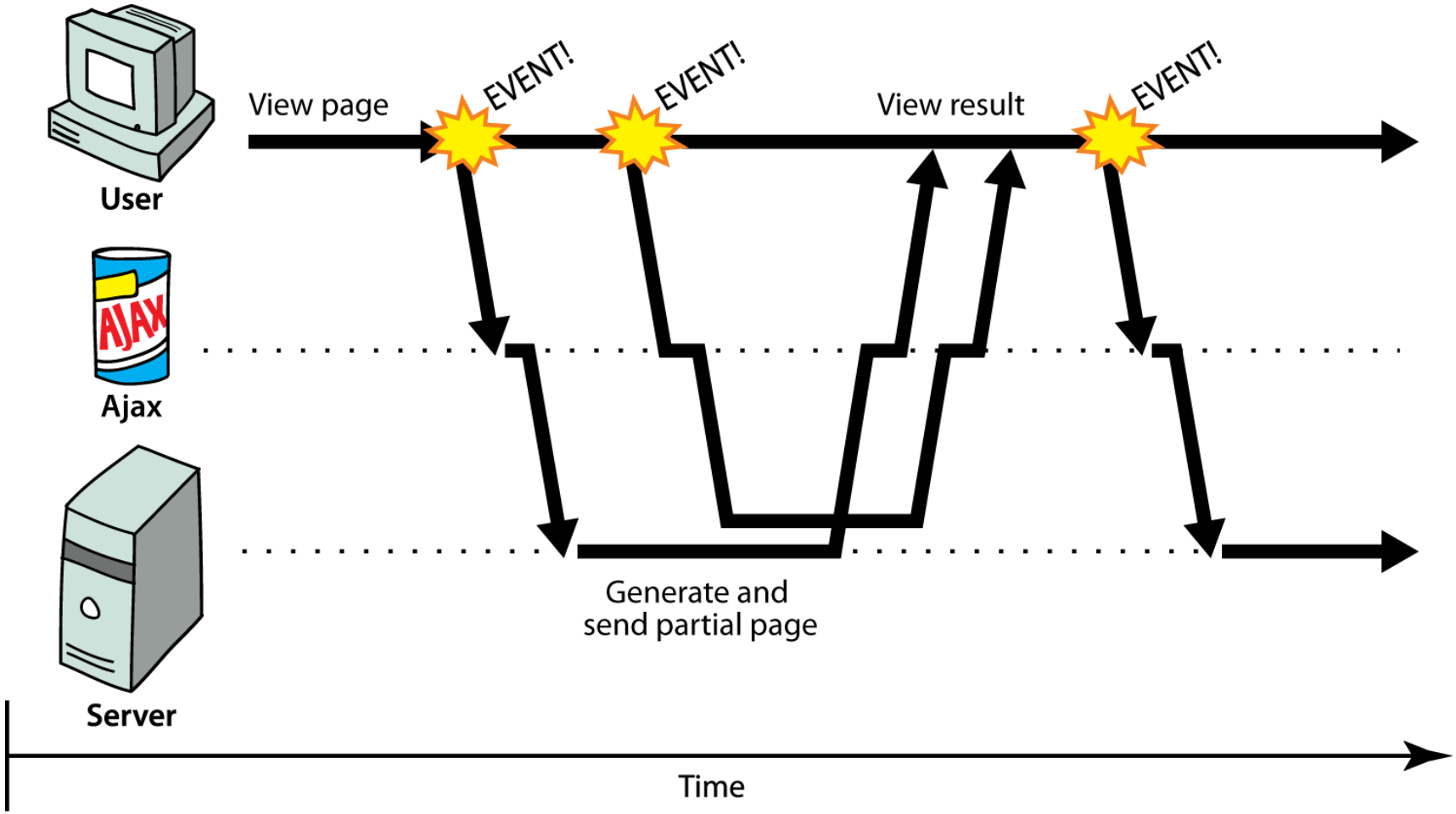
Synchronous requests: normal

ajax.**open**("GET", url, **false**);



Asynchronous requests:

ajax.**open**("GET", url, **true**);



Data formats

- Plain text
- XML
- JSON

Meet XML

- HTML: The “ML” stands for markup language to create the “HT” - hypertext.
- XML is another markup language that is used to create anything you want. That’s what the “X” means—anything! (Extensible)
- Any type of data can be encoded as tags and attributes.

<element attribute="value">content</element>

XML: custom tag sets

- HTML has a fixed set of tags and attributes
- XML does not enforce specific tags or attributes—it sets the rules for how they should be created and used.

XML example: books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year><price>30.00</price>
  </book>

  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year><price>29.99</price>
  </book>
  ...
</bookstore>
```

XML prologue

XML document tag (analogous to HTML)

Custom tag

Custom attribute

Parsing XML in JavaScript

- JavaScript unpacks XML data sent in the Ajax response and incorporates it into the page
- Use DOM model to parse XML data as a tree of nodes

Method name	Description
getElementsByTagName(tagName)	Returns an array of elements inside the current element that match the given tag name
getAttribute (attribName)	Returns the value of the node's attribute with the given attribute name

XML methods are the same as HTML DOM methods: both are trees

- properties:
  - **firstChild, lastChild, childNodes, nextSibling, previousSibling, parentNode**
  - **nodeName**, **nodeType**, **nodeValue**, **attributes**
- methods:
  - **appendChild, insertBefore, removeChild, replaceChild**
  - **getElementsByTagName**, **getAttribute**, **hasAttributes**, **hasChildNodes**

- **caution**: cannot use HTML-specific properties like *innerHTML* in the XML DOM!

Parsing *books.xml*

```
var xmlData = ajax.responseXML;
1 var doc=xmlData.
    getElementsByTagName("bookstore")[0];

  var books =
    doc.getElementsByTagName("book");
```

2 Now we have an array of book XML nodes.

3 We can access the category attribute of each book:

```
var category = books [i].getAttribute ("category");
```

4 And write it dynamically into the page:

```
var div = document.createElement ("div");
div.innerHTML = category;
document.body.appendChild (div);
```

Source file: link

```
<bookstore>
 <book category="cooking">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year><price>30.00</price>
 </book>

 <book category="children">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year><price>29.99</price>
 </book>

<book category="computers">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year><price>39.95</price>
 </book>
</bookstore>
```

XML: real power

- XML contains both data and metadata (description of this data).

  <element attribute="metadata">data</element>

  rule of thumb: data = element tag, metadata = attribute

- The real power of XML is that we can cook up an entirely custom markup language for any purpose.

  Examples: MathML, MusicXML, Scalable Vector Graphics, Microformats …

XML is both human-readable and computer-parsable

"self-describing data“

```
<movie id=“1”>
  <title>Gleaming the Cube</title>
  <releaseDate>01/13/1989</releaseDate>
  <director>Graeme Clifford</director>
  <summary>A skateboarder investigates the death of his
                                        adopted brother.
  </summary>
</movie>
```

Gleaming the Cube, 01/13/1989,Graeme Clifford,

A skateboarder investigates the death of his adopted brother

Get over XML, meet JSON

- JavaScript doesn't lock you into using XML as a data format for Ajax requests and responses

- **JavaScript Object Notation (JSON):** format that represents data as a set of JavaScript objects
  - invented by JS guru Douglas Crockford of Yahoo!
  - natively supported by all modern browsers
  - quickly rising in popularity

Recall: JavaScript object literals

```
//book object
var book = {
    "category": "cooking",
    "year": 2005,
    "price": 30.00,
     "title": "Everyday Italian",
    "author": "Giada De Laurentiis"
};

//array of books
var bookStore = [ ] ;
```

Equivalent representation of books.xml using JSON

```
{
  "bookstore":
  [
    {"category": "cooking", "year": 2005, "price": 30.00,
    "title": "Everyday Italian", "author": "Giada De Laurentiis"},
    {"category": "computers", "year": 2003, "price": 49.99,
    "title": "XQuery Kick Start", "author": "James McGovern"},
    {"category": "children", "year": 2005, "price": 29.99,
    "title": "Harry Potter", "author": "J K. Rowling"},
    {"category": "computers", "year": 2003, "price": 39.95,
    "title": "Learning XML", "author": "Erik T. Ray"}
  ]
}
```

Parsing JSON in JavaScript

- Use Ajax to fetch data in JSON format
- Call *JSON.parse* on it to convert into an object
- Interact with that object as you would with any other JavaScript object

Method name	Description
JSON.parse(*string*)	converts the given string of JSON data into an equivalent JavaScript object and returns it
JSON.stringify(*object*)	converts the given object into a string of JSON data (the opposite of JSON.parse)

Exercise: JSON format

```
var data =
    JSON.parse(ajax.responseText);
```

Given the parsed JSON data on the right, what expressions would produce:

1. The window's title?
2. The image's third coordinate?
3. The number of messages?
4. The y-offset of the last message?

```
{
"window":
   {"title": "Sample Widget",
     "width": 500, "height": 500 },
"image":
   { "src": "images/logo.png",
     "coords": [250, 150, 350, 400],
     "alignment": "center" },
"messages":          [
   {"text": "Save", "offset": [10, 30]} ,
   {"text": "Help", "offset": [ 0, 50]},
   {"text": "Quit", "offset": [30, 10]},
                          ],
"debug": "true"
}
```

Book application with JSON

```
var url = "books.json";
var jsonObj = JSON.parse(ajax.responseText);
var books= jsonObj["bookstore"];
for (var i=0; i< books.length; i++)
{
        var book = books [i];
        var bookDescription = book.category + " " +
                        book.author + " "+book.title;       …
```

link

Dynamic sales example

Ajax security restrictions

- cannot be run from a web page stored on your hard drive
- can only be run from a web page stored on a web server
- can only fetch data from the same site that the page is on

  http://**www.foo.com**/a/b/c.html can only connect to **www.foo.com**

Next lecture

- Overcoming security restrictions with JSON-P
- Creating web services
- Getting data from any server on the web
- Data visualization