Persistent

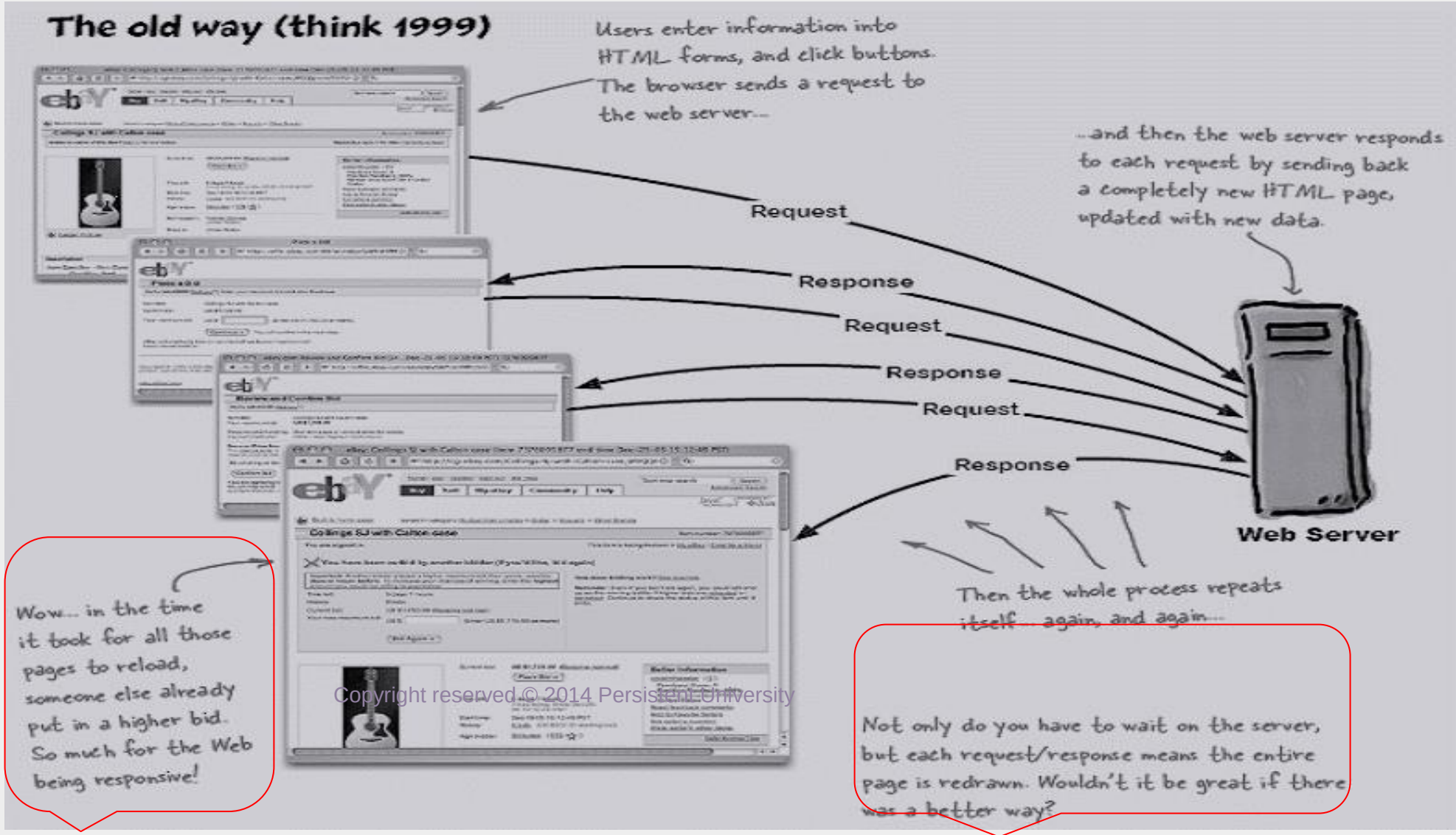
JavaScript: AJAX – Asynchronous JavaScript with XML

Persistent Interactive | Persistent University

Key learning points :

- Conventional web applications – characteristics & issues
- Ajax – need & essential features
- Basic Ajax process - states, methods and properties of XMLHttpRequest object
- Making a ‘Get/Post/Head’ Ajax call along with Get vs. Post request issues
- Ajax request & response using JSON data
- Ajax call using Fetch API

Conventional Web Apps

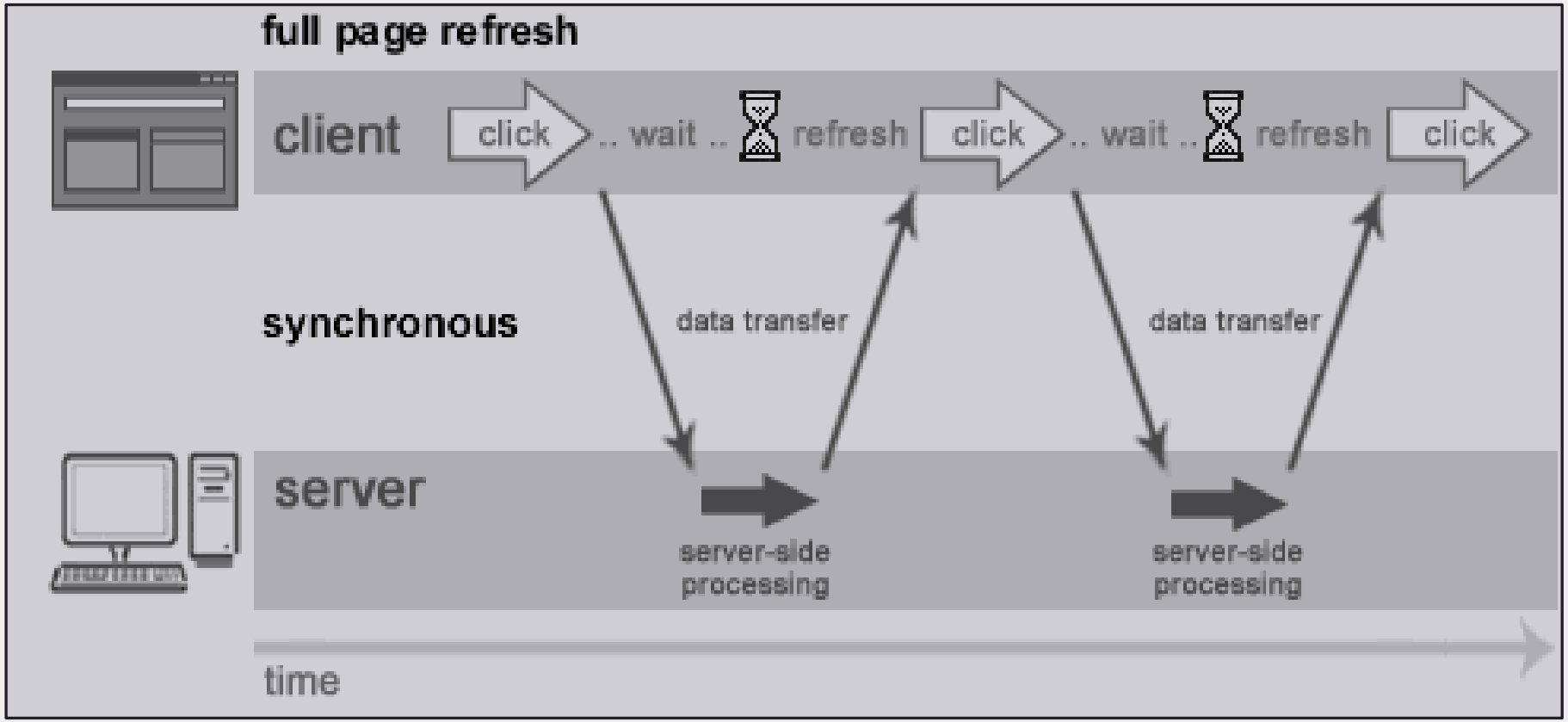


Persistent

Characteristics of these applications

- Response contains entire HTML markup
- Hence delay in loading page every time
- Page refreshes from the server needed for all events, data submissions, and navigation
- “Click, wait, and refresh” model often blocks the user
- No instant feedback's to user activities
- Loss of operational context during refresh

Synchronous Call in Web Application

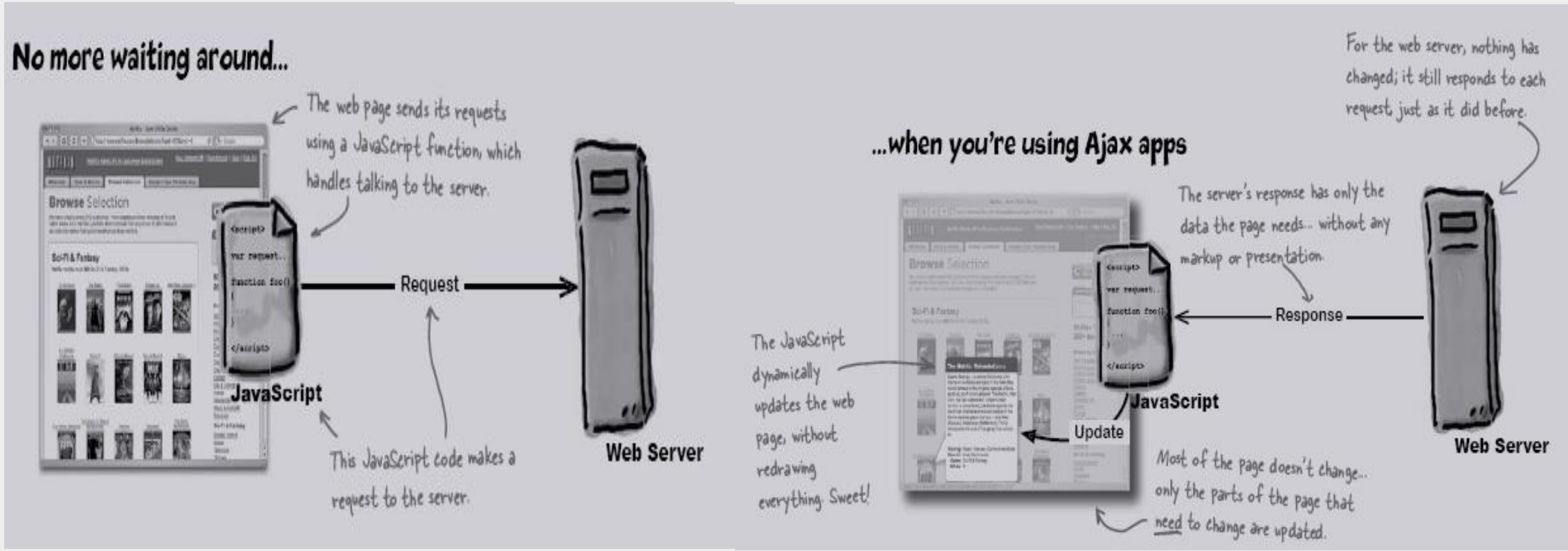


Persistent

Limitations of synchronous communication model

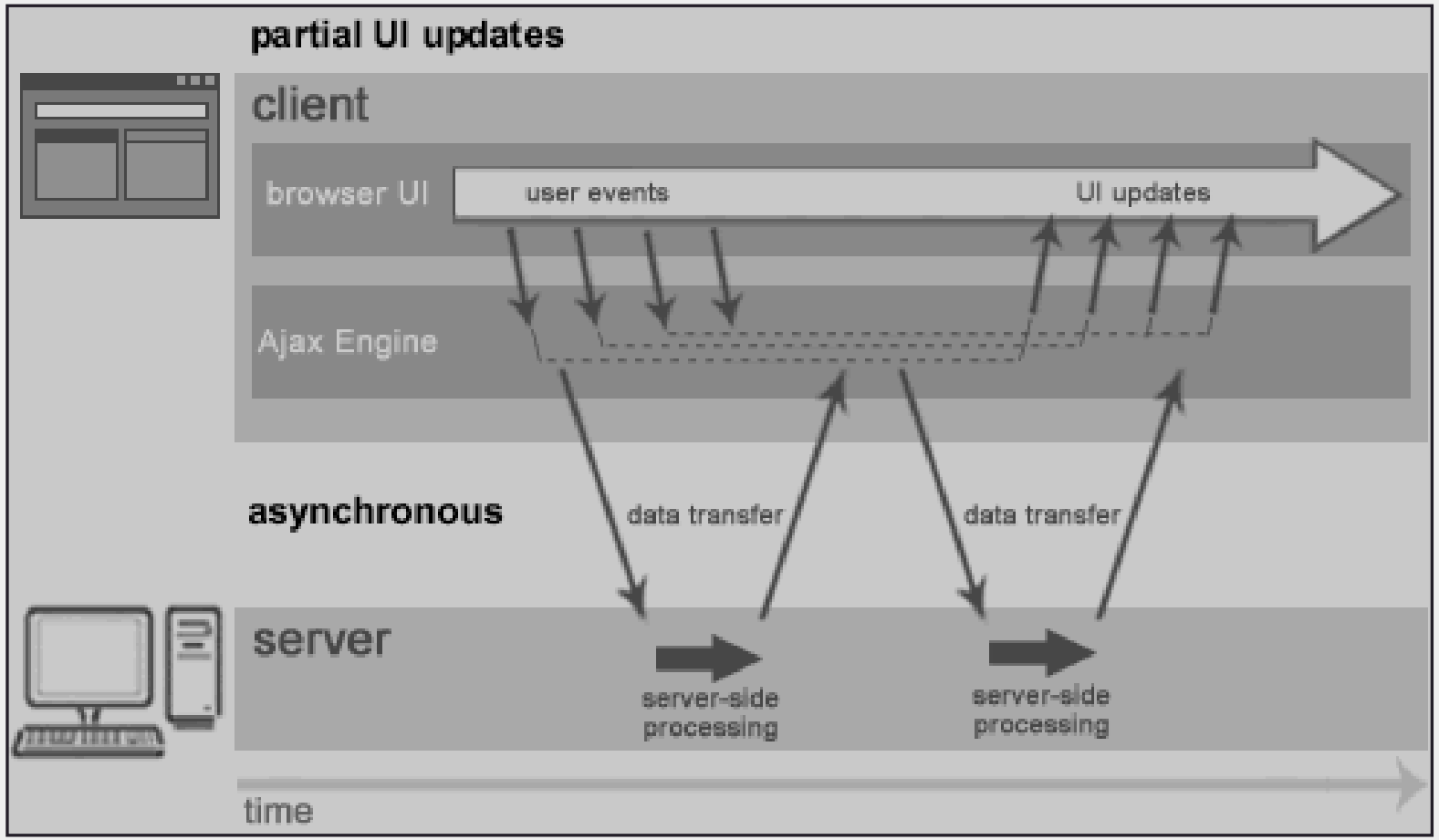
- Slow response hence blank screen
- Page-driven: Workflow is based on pages
- Page-navigation logic is determined by the server
- Hence, Rich Internet Application (RIA) technologies were born.

AJAX - Asynchronous + JavaScript + XML



Persistent

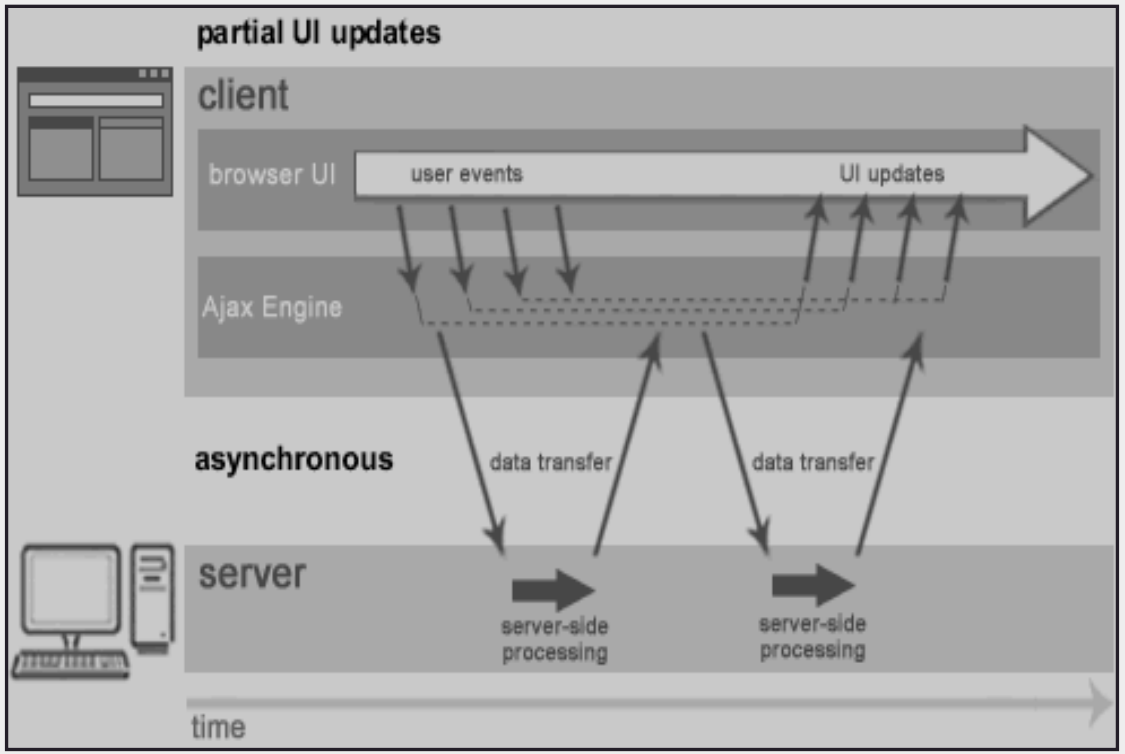
Asynchronous Call in Web Application



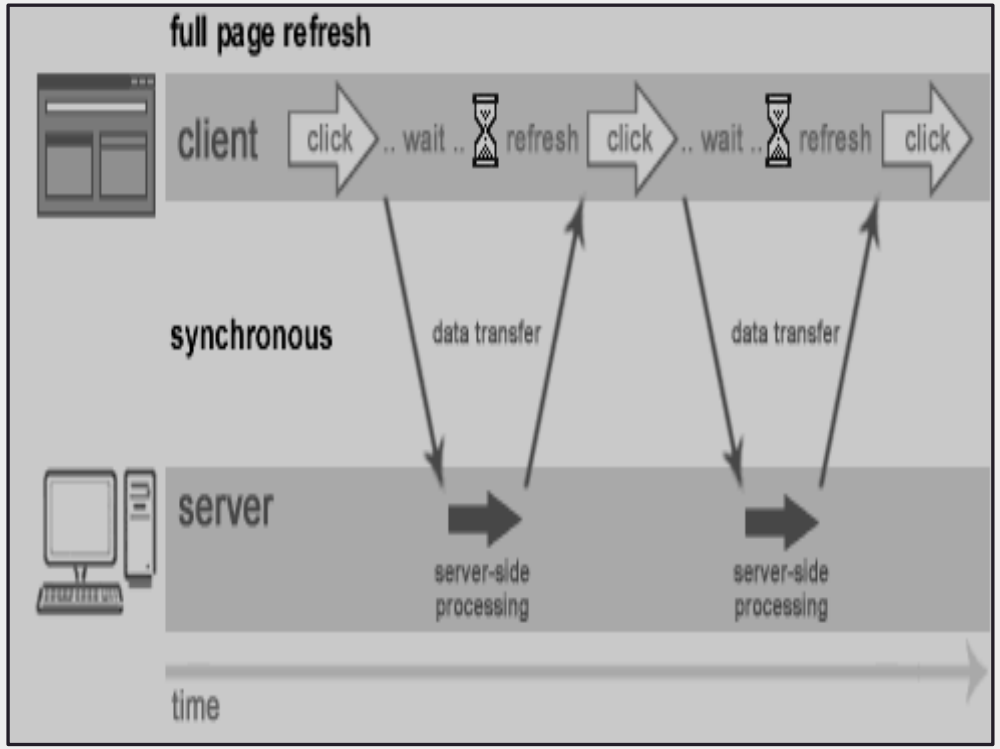
Persistent

Asynchronous Vs Synchronous

- Asynchronous



- Synchronous



Persistent

Technologies Used In Ajax

- JavaScript
  - Glue for the whole AJAX operation
- DOM
  - API for accessing and manipulating structured documents
- CSS
  - Beautifies & adds styles to the html page
- XMLHttpRequest Object

XMLHttpRequest Object

- JavaScript object that works in the background
  - for performing asynchronous communication with the backend server

- Adopted by modern browsers
  - Chrome, Internet Explorer, Mozilla Firefox, Safari, and Opera

- Communicates with a server via standard HTTP GET/POST

Basic Steps

- Create an XMLHttpRequest Object
- Build the URL to connect to
- Open a connection to the server
- Set up a call back handler to handle response
- Send the request
- Process the response in the call back handler

Initiate Request

```
function callServer() {
        // 1. Create new XMLHttpRequest
        var xmlHttp = new XMLHttpRequest();

        // 2. Get the city and state from the web form
        var city = document.getElementById("city").value;

        // 3. Build the URL to connect to
        var url = “StateServer.jsp?city=" + city;

        // 4. Open a connection to the server
        xmlHttp.open("GET", url, true);

        // 5. Setup a function for the server to run when it's done
        xmlHttp.onreadystatechange = updatePage;

        // 6. Send the request
        xmlHttp.send(null);
}
```

Response Handler

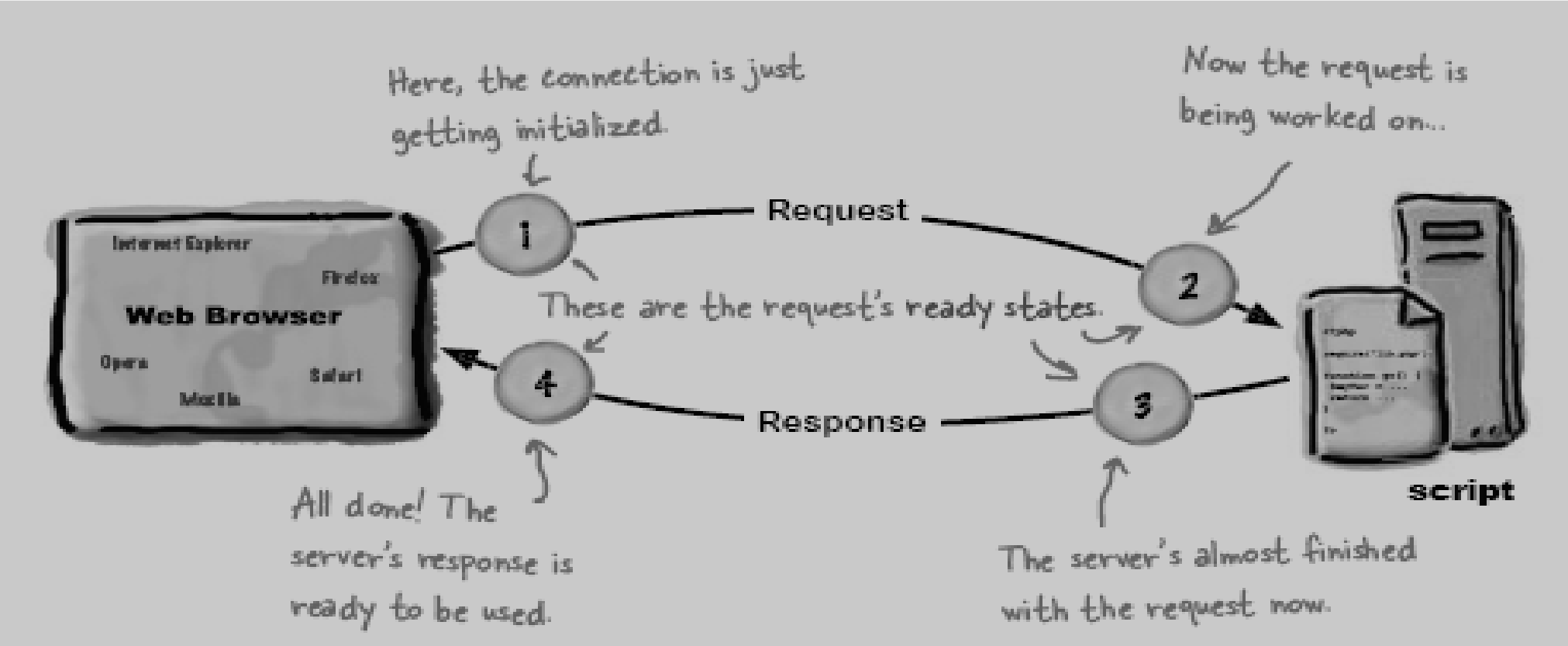
- Register a handler for response

```
xmlHttpReq.onreadystatechange = updatePage;

function updatePage() {

        var response = xmlHttp.responseText;
        document.getElementById("zipCode").value = response;
}
```

Make Sure the Server is finished



Persistent

HTTP ready states

- 0: The request is uninitialized
- 1: The request is set up, but hasn't been sent
- 2: The request was sent and is being processed
- 3: The request is being processed; often some partial data is available from the response, but the server hasn't finished with its response
- 4: The response is complete; you can get the server's response and use it

```
function updatePage() {

    if (request.readyState == 4 &&
            request.status==200)

        alert("Server is done!");
}
```

Caching Problem

- Solution 1 :-
  - add current time to the url

```
function getBoardsSold() {
    request = new XMLHttpRequest();
    var url = “getUpdatedBoardSales-ajax.jsp”;
    url = url + “?dummy=“ + new Date().getTime();
    request.open(“GET”, url, true);
    request.onreadystatechange = updatePage;
    request.send(null);
}
```

Caching Problem continued..

- Solution 2 :-
  - set below headers in the response object of jsp/servlet

```
response.setHeader("Cache-Control", "no-cache");

response.setHeader("Pragma", "no-cache");
```

Get Vs Post

- Get
  - Data appended to URL, which is visible
  - Size of data that can be sent is limited
  - Cannot send binary data
  - Requests can be cached

- Post
  - Data sent in body
  - No size limitations on data
  - Can send binary data
  - No caching issues

No Caching Problems in POST

- Browsers don’t cache POST requests
- Set the content type

```
request.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
```

Sample example to initiate request

```
function sendRequestWithData(address, data,
responseHandler) {
        request = getRequestObject();
        request.onreadystatechange = responseHandler;
        request.open("POST", address, true);
        request.setRequestHeader("Content-
Type","application/x-www-form-urlencoded");
        request.send(data);
}

function showTimeInCity() {
        var address = "../show-time-in-city";
        var city = document.getElementById("city").value;
        var data = "city=" + escape(city);
        sendRequestWithData(address, data,
showResponseAlert);
}
```

Handle Response

```
function showResponseAlert() {
        if ((request.readyState == 4) &&(request.status ==
200)) {

                alert(request.responseText);
        }
}
```

Common mistakes in understanding Ajax

- XMLHttpRequest: Poor names and HTTP
- The requests are HTTP, not XML
- Two ways of working with XML data
  - To send a request from a Web page to a server in XML format
  - To receive a request from a server in your Web page in XML format

Receiving XML from Server

- Client Side :

- Don’t forget at server side:

```
var xmlDoc = req.responseXML;

var boardsSoldElements=
xmlDoc.getElementsByTagName(“boards-sold”);

response.setContentType("text/xml");
```

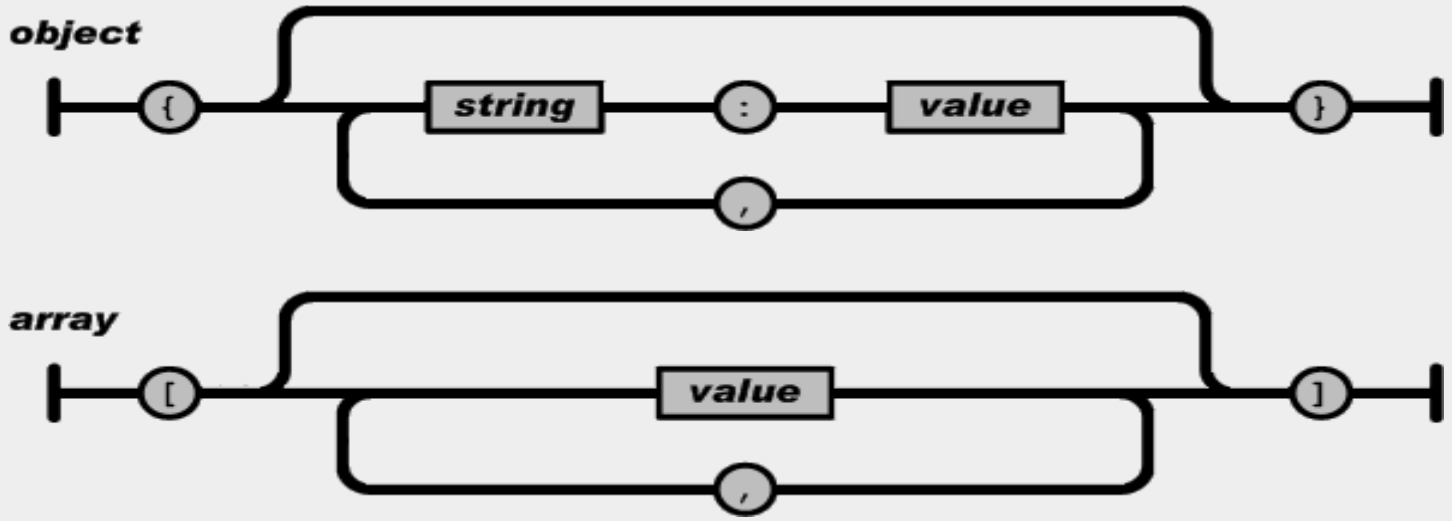
Sending XML: Good or bad ?

- XML is not simple to construct
- XML doesn't add anything to your requests
- Server script should accept XML
  - Generally sent an XML request to servers that only accepts XML

JavaScript Object Notation - Better than XML !

- A JSON object
  - begins with { (left brace) and ends with } (right brace)

- Each name is followed by : (colon) and the name/value pairs are separated by , (comma)



Persistent

JSON Examples

```
{
   ‘firstname’ : ‘Vishal’,

    ’lastname’: ’Gupta’,

    ‘isPermanent’ : true

}
{
    ‘brand’: ’Hyundai’,

     ‘model’ : ‘i20’,

      ’wheels’: 4
}
```

Why JSON over XML ?

- JSON objects are typed while XML data is type less
  - JSON types: string, number, object, Boolean etc.
  - XML data are all string

- Native data form for JavaScript code
  - Data is readily accessible as JSON objects
  - XML data needed to be parsed and assigned to variables through tedious DOM APIs
  - Retrieving values is as easy as reading from an object property

JSON Structures

- A collection of name/value pairs
  - in various languages, it is realized as an object, record, struct, dictionary, hash table, keyed list etc
- An ordered list of values
  - In most languages, this is realized as an array, vector, list, or sequence etc
- Universal data structures supported by most modern programming languages

Data Types supported in JSON

- String
  - {“name“ : “ABCD"}

- Number
  - {“age“ : 50}

- Boolean
  - {“avail“ : true, “status“ : false}

Data Types supported in JSON continued..

- Objects
  - {“users“ : {“user1“ : {“name“ : ”Ajay”}}}
  - Arrays
    - {“names“ : [“Ajay", “Vijay"]}
  - Null
    - {“age“ : null}
    -

How to access values?

- Creation
- Access values using
  - Dot operator

```
var jsonObj = {
                    “name“ : “Ajay“,
                    ”age” : 50
                }

jsonObj.name

jsonObj.age
```

Example: JSON Object

- Members can be retrieved
  - using dot or subscript operators

- JSON disallows JavaScript keywords as element names

```
var myJSONObject = { “users":  [
                                {“fname": “Vishal", “lname":
“Gupta", “phone": 76854},
                                {“fname": “Priti", “lname":
“Singh", “phone": 76854},
                                {“fname": “Jake", “lname":
“Hook", “phone": 76854}
                                 ]
                          };
myJSONObject.users[0].fname
```

Send JSON from client to server

- Create a JSON String directly
  - Make a usual Post Ajax call and send the JSON string
- Create a JavaScript Object and covert to JSON
  - Convert JavaScript Object into JSON representation
  - Make a usual Post Ajax call and send the JSON string

```
var user = {
                    “username”:”John”,

                    ”password”:”xyz”
};
```

JSON Text to JSON Object Conversion

- JSON parser
  - A JSON parser will only recognize JSON text and so is much safer
  - use it when security is a concern or source cannot be trusted

```
var jsonObject = JSON.parse(jsonText);
```

JSON Text to JSON Object Conversion continued..

- Using eval() function
  - eval() invokes the JavaScript compiler
  - JSON is a proper subset of JavaScript, the compiler will correctly parse the text and produce an object structure
  - can compile and execute any JavaScript program, so there can be security issues
  - use eval() when the source can be trusted

```
var myObject = eval('(' + myJSONtext + ')');
```

JSON Object to JSON Text Conversion

- Convert JSON object into JSON text using
  - JSON.stringify method

```
var jsonText = JSON.stringify(jsonObject);
```

AJAX call using fetch API

- The Fetch API provides a JavaScript interface for accessing and manipulating parts of HTTP pipeline, such as requests and responses
- It provides a global fetch() that provides an easy, logical way to fetch resources asynchronously across the network.
- Before this API, XMLHttpRequest was used to make API request.
- fetch is JavaScript’s own built-in way to make API request

AJAX call using fetch API

- fetch() includes Promises.
- The fetch() method returns a Promise, which get handled by then()

```
fetch(url)
        .then(function()  {
                //handle the response
})
```

Summary : Session

With this we have come to an end of our session, where we discussed :

- Characteristics of both Conventional and Ajax based applications
- Implementation of Ajax in JavaScript using XMLHttpRequest object.
- Ajax request & response using JSON
- Using fetch() API

At the end of this session, we expect you to :

- Understand features, concepts & implementation of Ajax
- using various Http methods & data transfer medium
- Apply these concepts as per requirement

Appendix

- References
- Key Contacts

Reference Material : Books

Professional: JavaScript® for Web Developers

- By: Nicholas C. Zakas
- Publisher: Wrox

- **JavaScript Step by Step**
  - By: Steve Suehring
  - Publisher: Microsoft Press

Persistent

Thank you!

Persistent Interactive | Persistent University