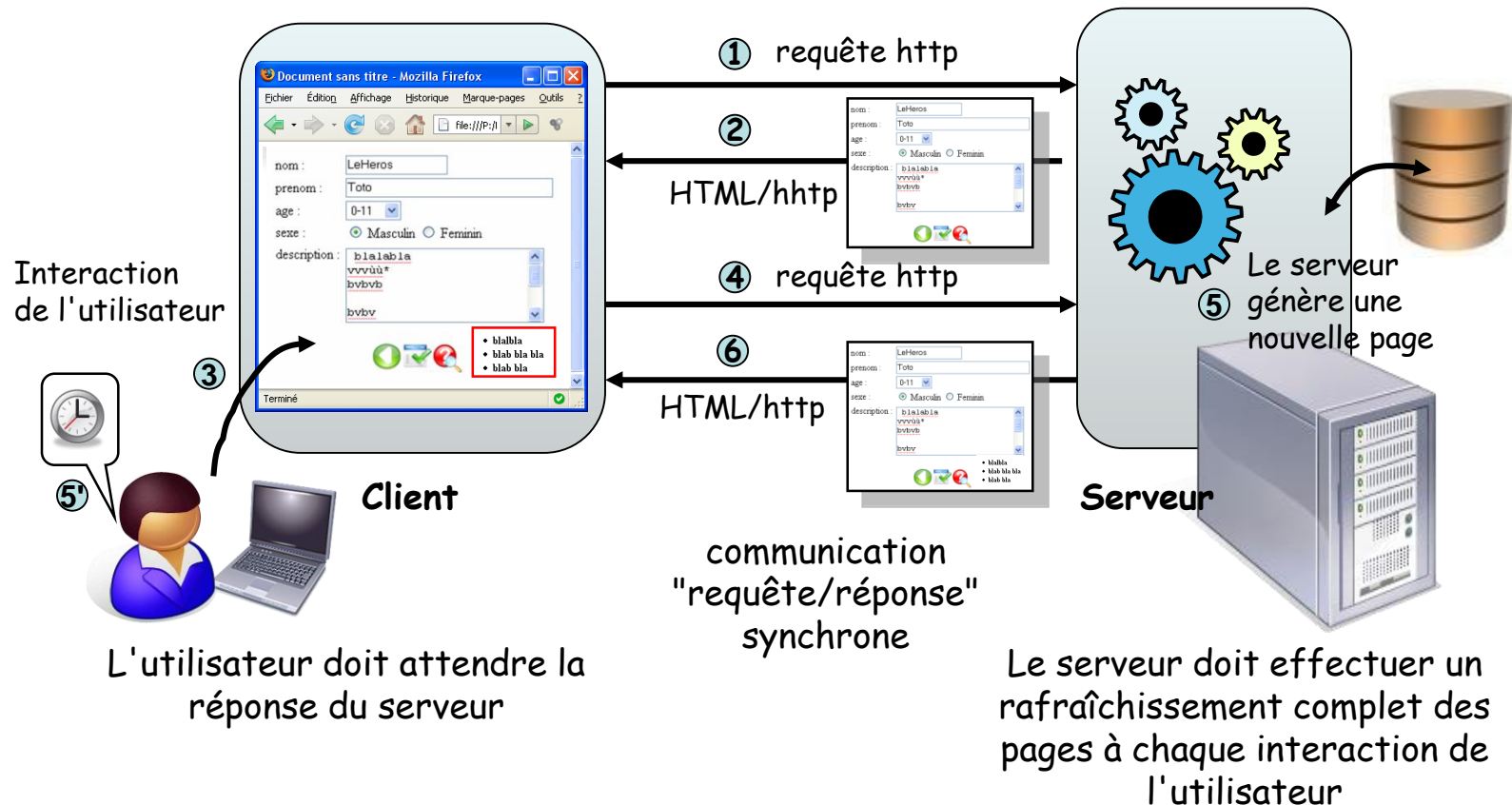


# **Web 2.0**

## **Introduction à Ajax**

# Caractéristiques des applications Web "Classiques"

- Navigateur outil générique d'affichage : il n'a aucune intelligence de l'application
- Logique de navigation sous forme d'enchaînement de pages est déterminée par le serveur.



# Limites des applications Web "Classiques"

---

- Ergonomie en retrait
  - Contrainte par HTML
    - Ensemble limité de widgets
  - Pas de retour immédiat aux activités de l'utilisateur
    - L'utilisateur doit attendre la page suivante générée par le serveur
  - Interruption des activités de l'utilisateur
    - L'utilisateur ne peut effectuer d'autres opérations pendant qu'il attend une réponse
  - Perte du contexte opérationnel suite au rafraîchissement
    - Perte de la position de scrolling dans la page
    - Le cerveau doit réanalyser entièrement toute nouvelle page

# Remédier aux limites du Web "Classique"

- Animation des écrans prise en charge du côté client
  - Animation d'un écran assurée par du code exécuté sur le navigateur
    - limite les échanges navigateur/serveur web
    - possibilité d'augmenter l'interactivité et de réaliser des comportements ergonomiques plus évolués
- Optimisation des échanges navigateur/serveur
  - **Communication asynchrone**
    - Lorsqu'une requête est émise par le client, celui-ci reprend la main immédiatement
  - Echange des données plutôt que de la présentation (**une fois la page initiale chargée**)

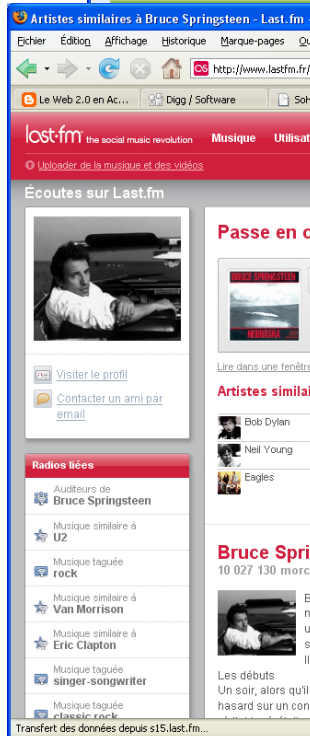
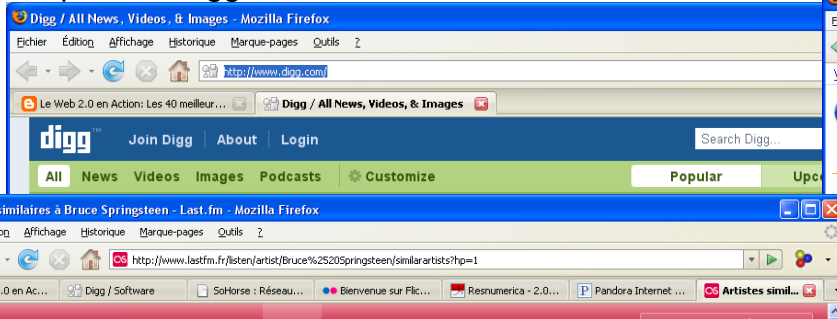
→ **Technologies RIA (Rich Internet Application)**

→ **Web 2.0 versus Web 1.0**

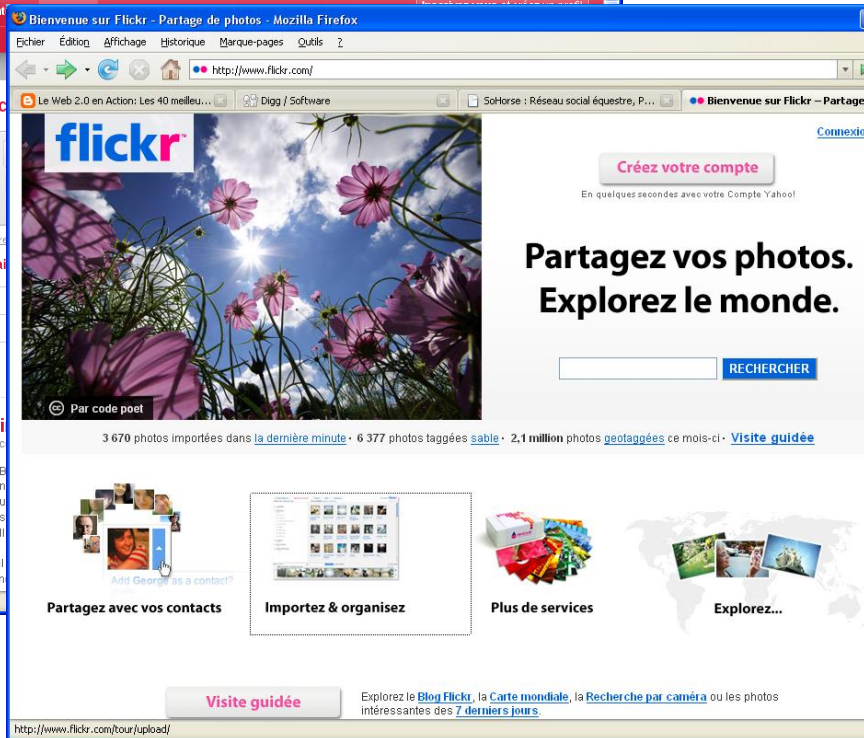
# Exemples de sites web 2.0

Google maps

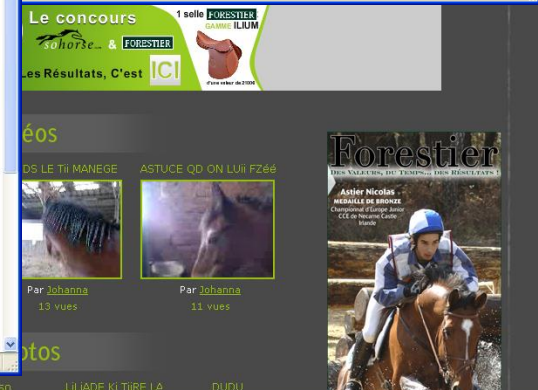
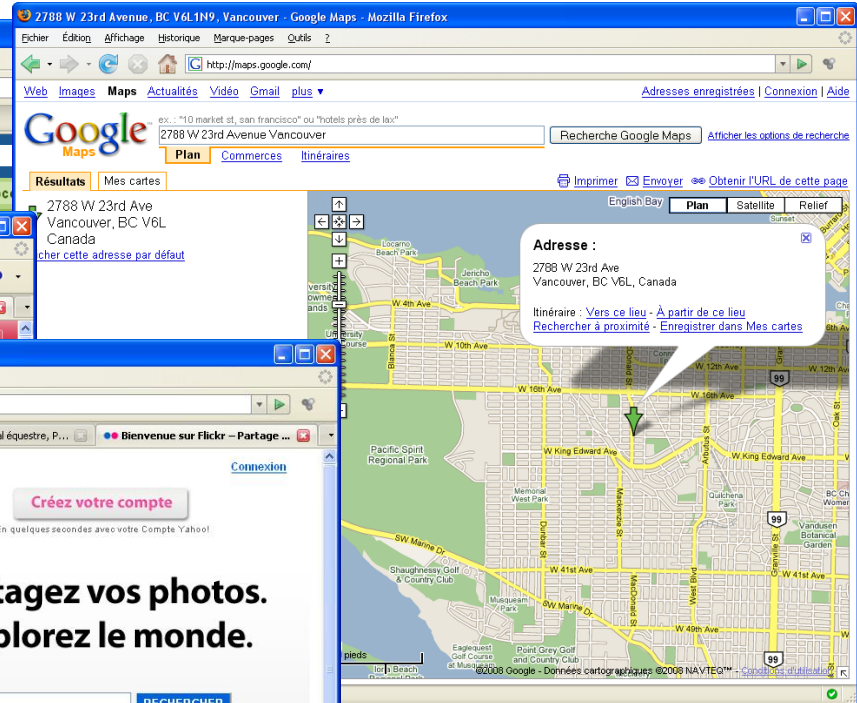
<http://www.digg.com/>



<http://www.lastfm.fr>



<http://www.flickr.com/>



<http://www.soshore.com/mysoshore?sessionId=8BE62E7B110567A5BC4315269F92E30>

# Technologies RIA (Rich Internet Applications)

- De nombreuses technologies (pas toutes récentes)
  - Applets (1995)
    - Code java
  - DHTML (Dynamic HTML)
    - Javascript (1995) + DOM + CSS
  - Flash/FLEX (1996, 2004) Macromedia-Adobe
    - MXML + ActionScript
  - Silverlight (2006) Microsoft
    - XAML + ECMAScript
  - JavaFX (2008) Sun ...
  - AJAX : **A**synchronous **J**avascript **A**nd **X**ML
    - AJAX = Javascript + XmlHttpRequest
      - Code client écrit en Javascript
      - Communications avec le serveur réalisées à l'aide de l'objet Javascript **XmlHttpRequest**

Technologie anciennes :  
Javascript 1995  
XmlHttpRequest : 1999 dans IE5

Ce qui est nouveau est leur  
utilisation conjointe

# Technologies utilisées en AJAX

---

**AJAX = Javascript + XmlHttpRequest**

plus précisément

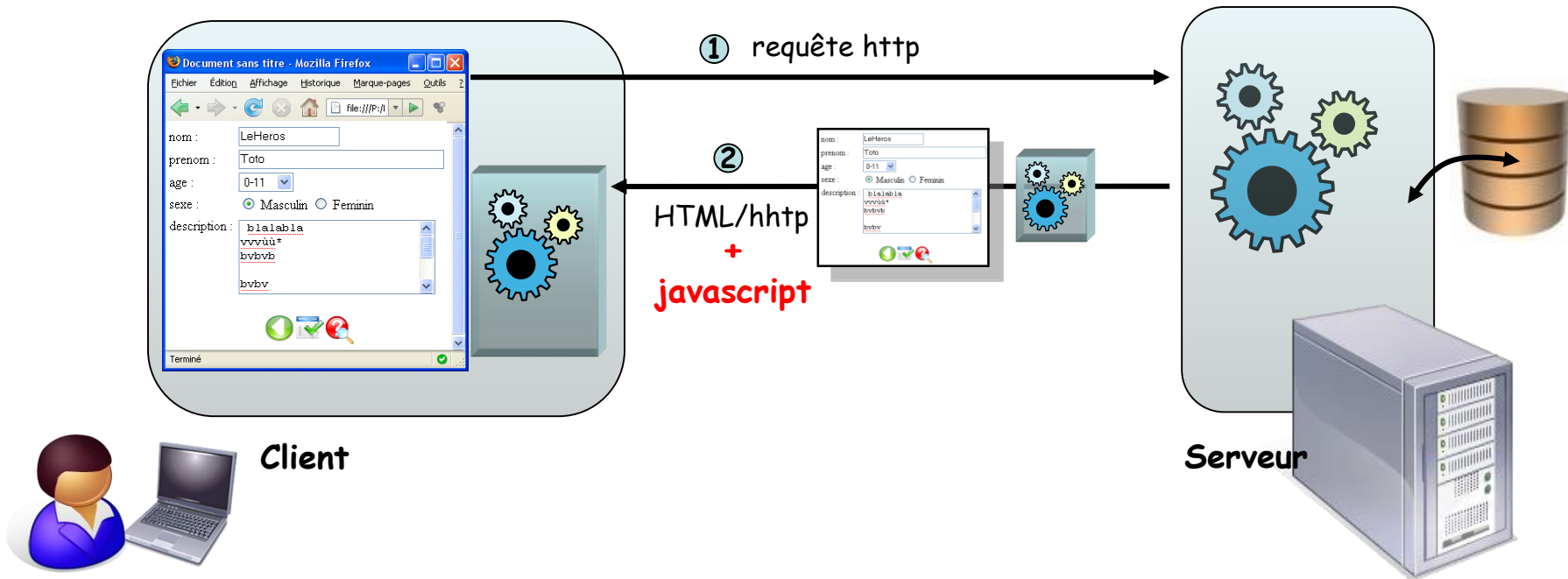
**AJAX = DHTML (DOM + CSS + Javascript) + XmlHttpRequest**

- Javascript
  - Langage de script orienté objet et faiblement typé
  - Fonctions javascript invoquées lorsque intervient un événement sur la page
  - "Glue" pour tout le fonctionnement d'AJAX
- DOM (Document Object Model)
  - API pour accéder à des documents structurés
  - Représente la structure de documents XML et HTML
- CSS (Cascading Style Sheets)
  - Permet une séparation claire du contenu et de la forme de la présentation
  - Peut être modifié par le code Javascript
- XmlHttpRequest
  - Objet Javascript qui assure une interaction **asynchrone** avec le serveur

# Application Web AJAX

- Une partie de l'intelligence fonctionnelle de l'application est déportée vers le navigateur

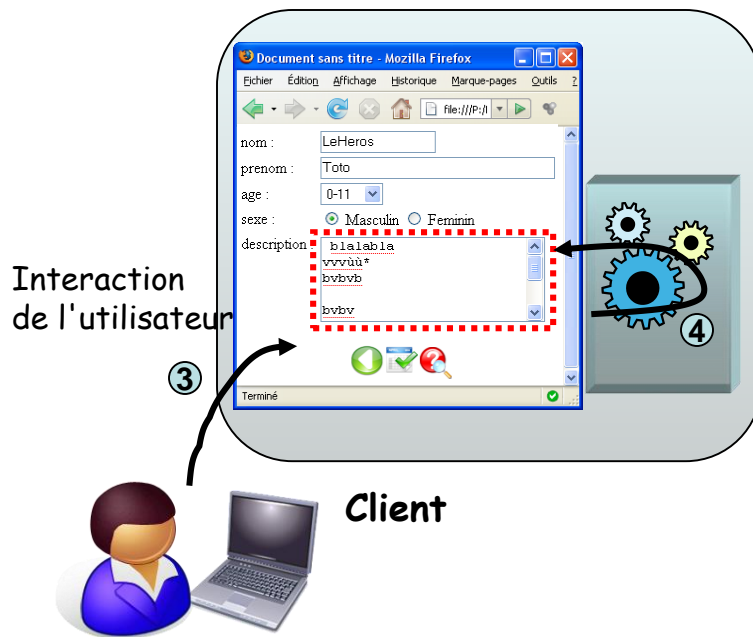
1<sup>er</sup> échange similaire au web "classique" : le serveur envoie une page au client mais en y embarquant de l'intelligence (code javascript)



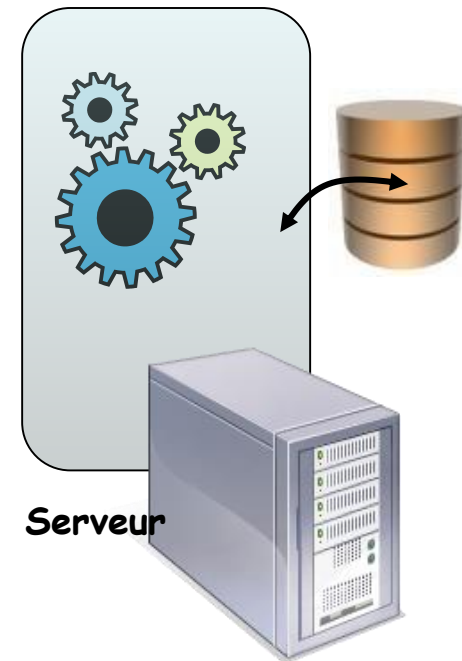


# Application Web AJAX

- Une partie de l'intelligence fonctionnelle de l'application est déportée vers le navigateur



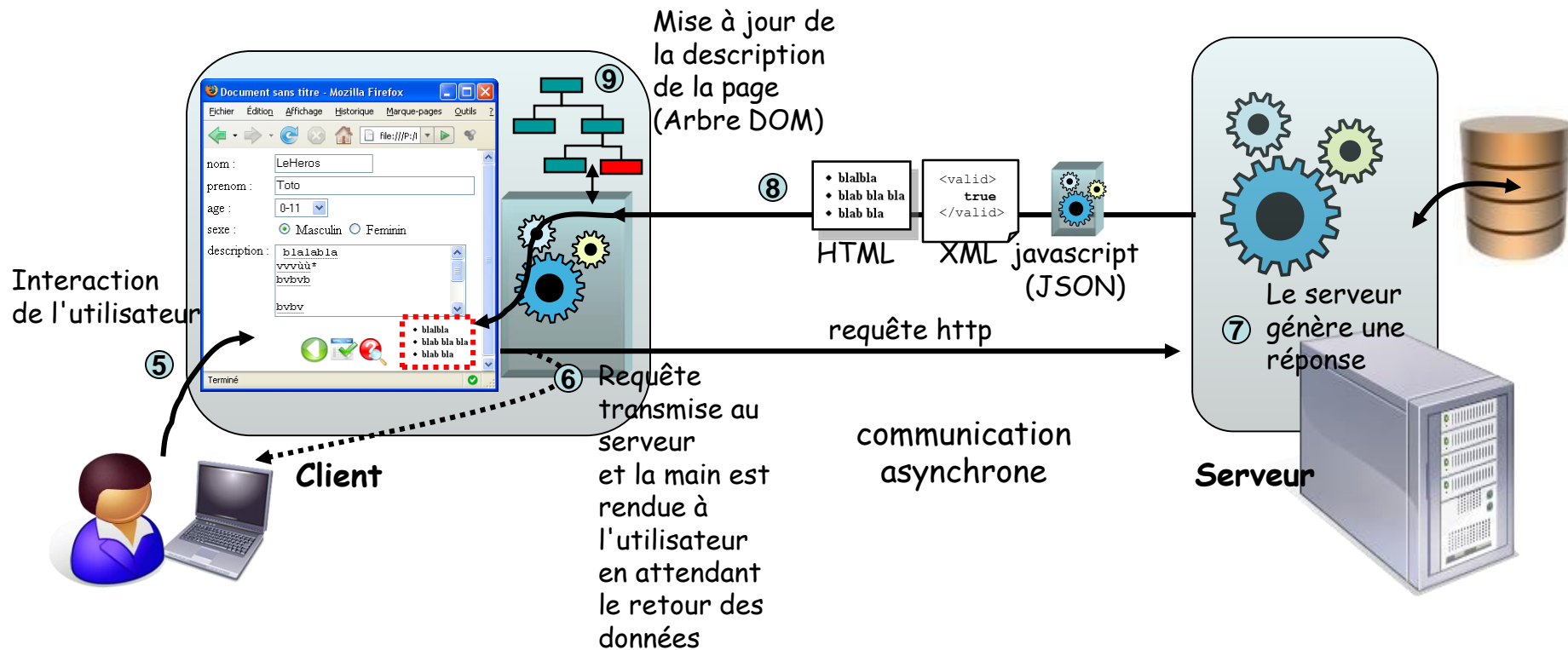
Certaines requêtes de l'utilisateur sont traitées localement par le navigateur grâce à la couche d'intelligence qui accompagne la présentation



# Application Web AJAX

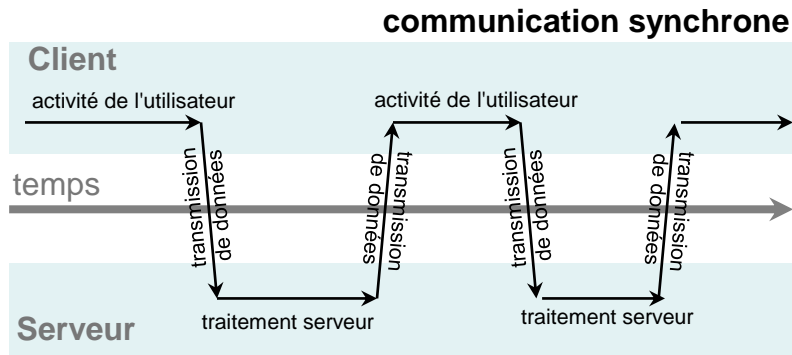
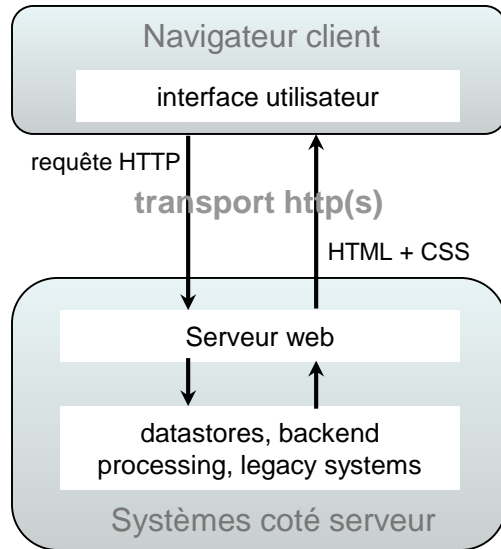
- Une partie de l'intelligence fonctionnelle de l'application est déportée vers le navigateur

D'autres requêtes nécessitent l'interrogation du serveur

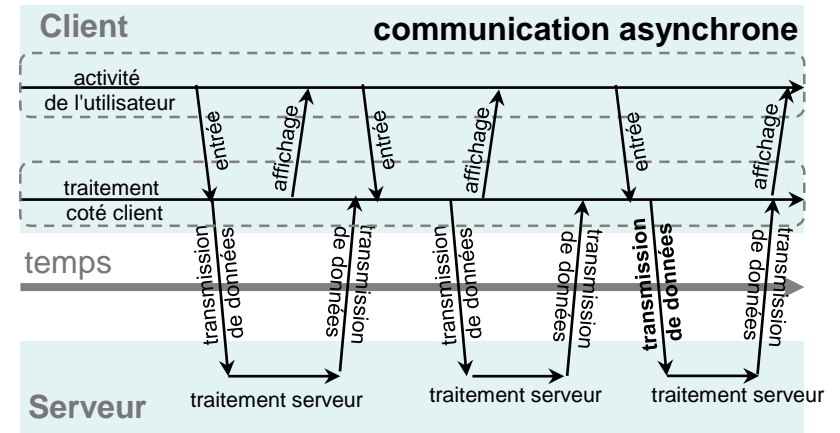
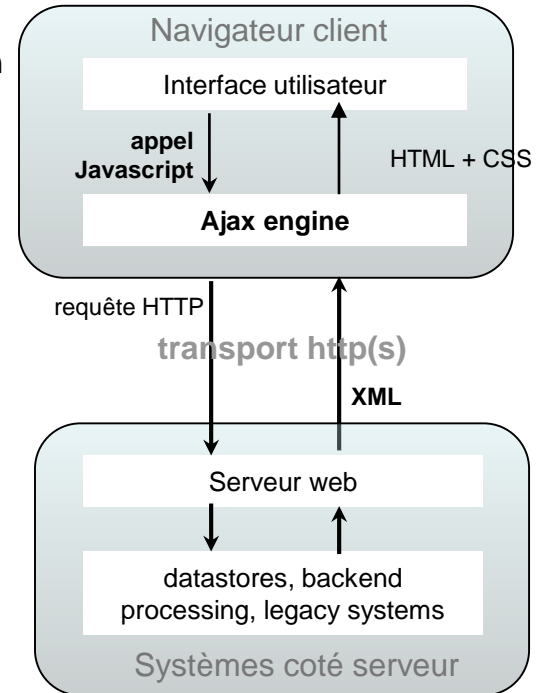


# Modèles des applications WEB

## Application Web "classique"



## Application Web AJAX



# Modèles des applications WEB

## Application Web "classique"

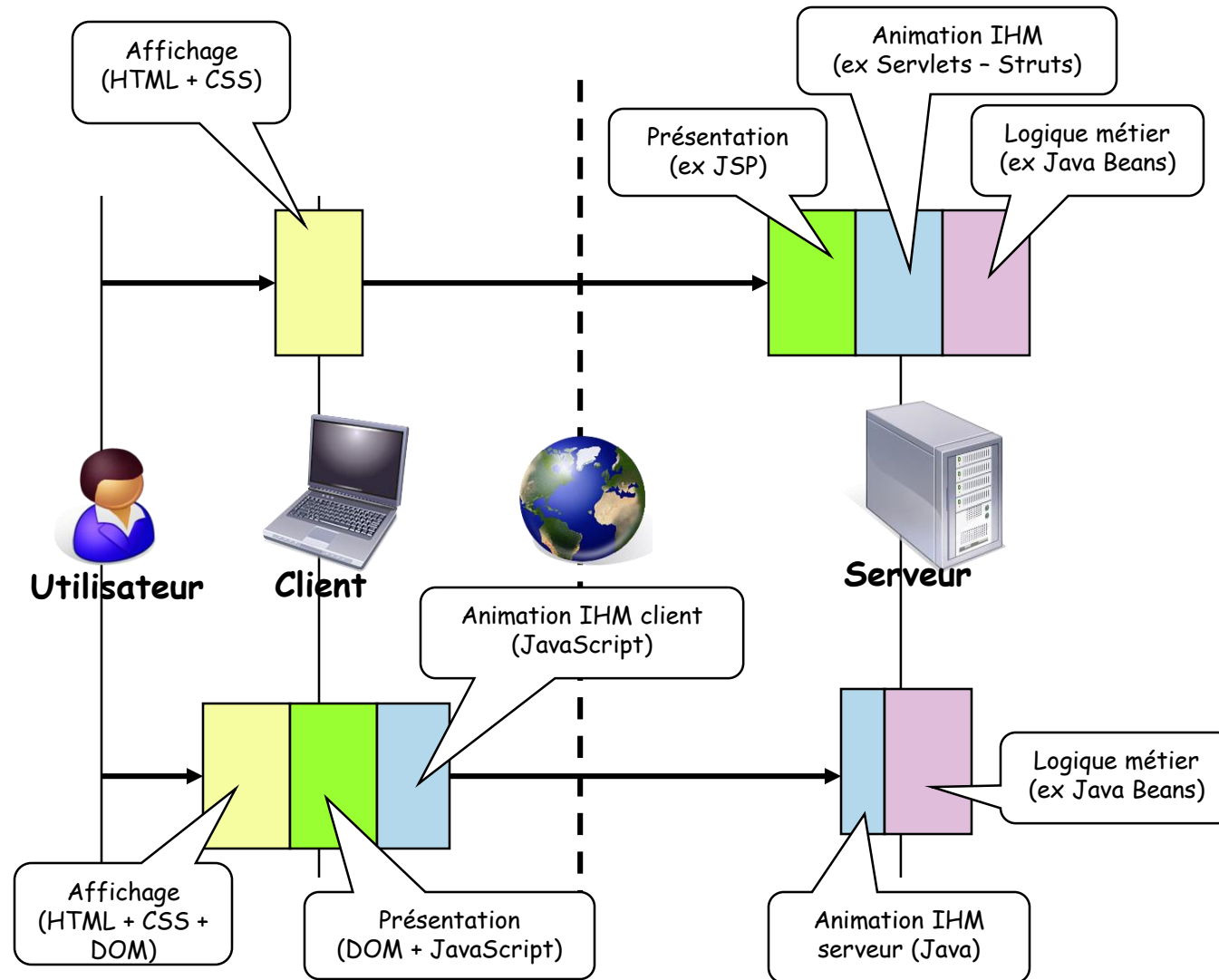
- Peu de javascript
- Charge serveur très importante
- Ergonomie faible

## Application Web AJAX Léger

- Cohabitation entre technologie "classique" (ASP, PHP, JSP) et AJAX
- Charge serveur importante
- Ergonomie améliorée

## Application Web AJAX complet

- Client en javascript
- Charge serveur modérée
- Ergonomie supérieure



# Anatomie d'une interaction AJAX

- Exemple d'après : *AJAX Basics and Development Tools* de Sang Shin (sang.shin@sun.com, Sun Microsystems) [www.javapassion.com/ajaxcodecamp](http://www.javapassion.com/ajaxcodecamp)

**Validation d'un formulaire en utilisant AJAX**

Cet exemple montre comment utiliser AJAX pour effectuer une validation de données côté serveur sans rechargement de la page.  
D'après le cours de Sang Shin, **18-week "Free" AJAX and Web 2.0 Programming (with Passion!)**  
**Online Course** [www.javapassion.com/ajaxcodecamp/](http://www.javapassion.com/ajaxcodecamp/)

In the form below enter a user id. By default the user ids "greg" and "duke" are taken. If you attempt to enter a user id that has been taken an error message will be displayed next to the form field and the "Create Account" button will be disabled. After entering a valid user id and selecting the "Create Account" button that user id will be added to the list of user ids that are taken.

User Id:  Invalid User Id

Create Account

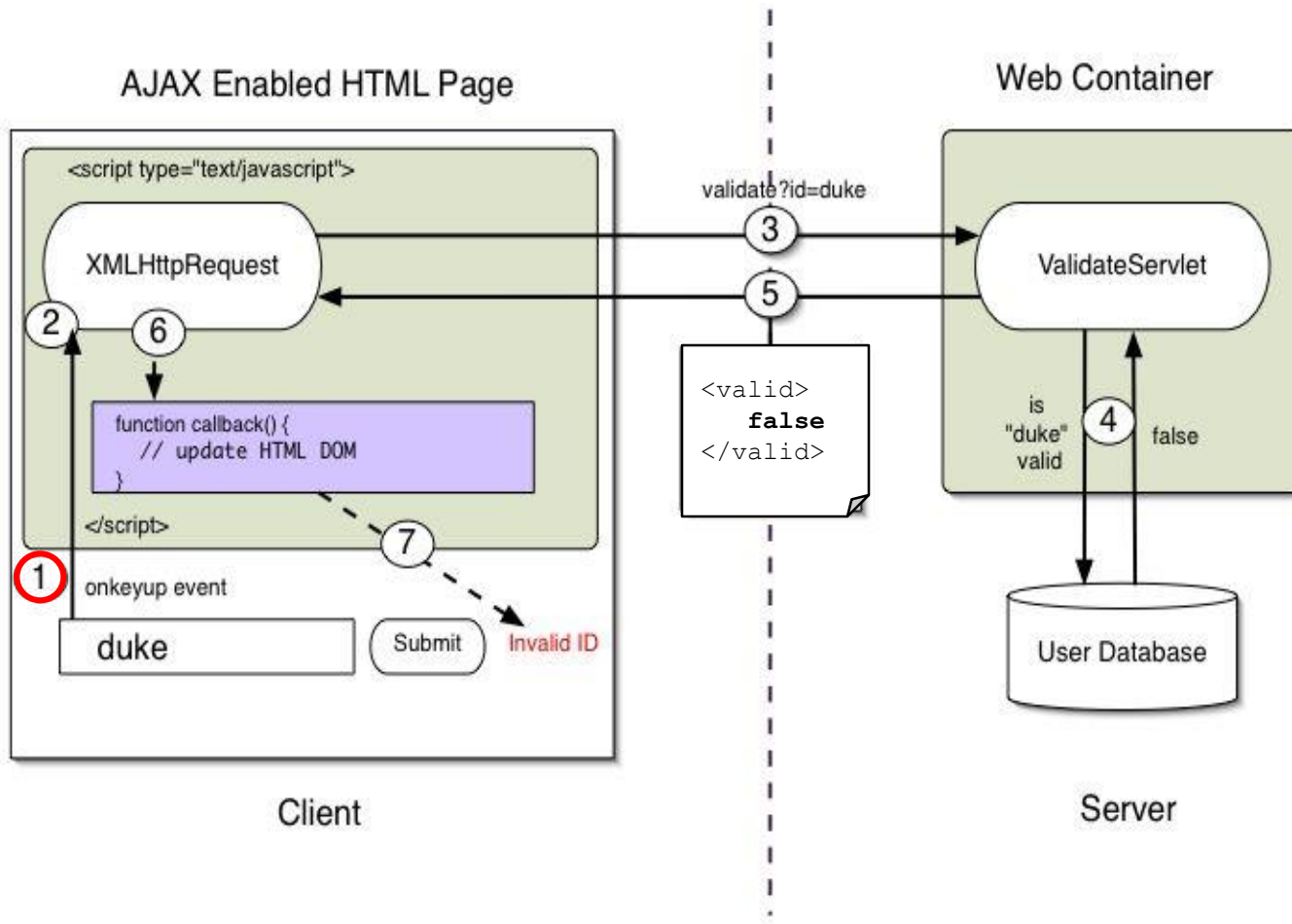
Terminé

L'utilisateur saisit un identifiant

Au fur et à mesure de la frappe un message indiquant la validité ou non de l'identifiant est affiché

Le bouton de création n'est activé que si l'identifiant est valide (n'est pas déjà utilisé)

# Anatomie d'une interaction AJAX



- ① Événement sur le client  
→ appel d'une fonction javascript
- ② Création et configuration d'un objet XMLHttpRequest
- ③ L'objet XMLHttpRequest fait une requête asynchrone
- ④ Le servlet valide l'identifiant soumis
- ⑤ Le servlet retourne un document XML contenant le résultat de la validation
- ⑥ L'objet XMLHttpRequest appelle la fonction `callback()` et traite ce résultat
- ⑦ Mise à jour de la page HTML (DOM)

# Anatomie d'une interaction AJAX

1

## Gestion des événements dans le formulaire HTML

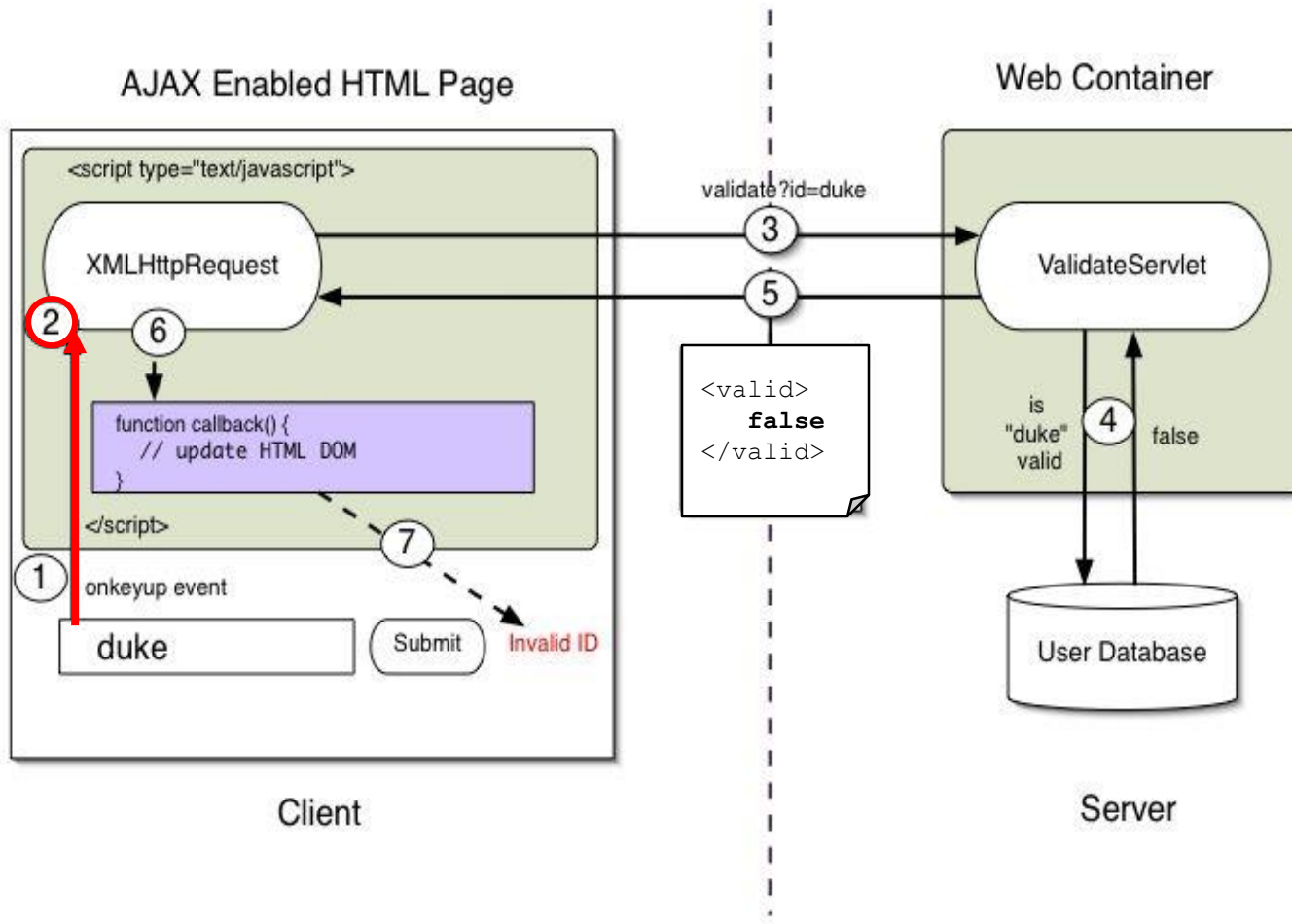
La fonction Javascript `validateUserId` est associée au champ de saisie de texte "`userid`" pour la gestion des événements de type `onkeyup` : `validateUserId` est appelée chaque fois que l'utilisateur tape une lettre dans le champ de saisie.

```
<form name="updateAccount" action="validate" method="post">
  <input type="hidden" name="action" value="create"/>
  <table border="0" cellpadding="5" cellspacing="0">
    <tr>
      <td><b>User Id:</b></td>
      <td>
        <input type="text" size="20" id="userid" name="id" onkeyup="validateUserId()" />
        <div id="userIdMessage"></div>
      </td>
    </tr>
    <tr>
      <td align="right" colspan="2">
        <div align="center">
          <input id="submit_btn" type="Submit" value="Create Account">
        </div>
      </td>
    </tr>
  </table>
</form>
```



L'élément `div` d'id `userIdMessage` spécifie la position où sera affiché le message de validation de l'entrée

# Anatomie d'une interaction AJAX



- ① Événement sur le client  
→ appel d'une fonction javascript
- ② Création et configuration  
d'un objet XMLHttpRequest
- ③ L' objet XMLHttpRequest fait  
une requête asynchrone
- ④ Le servlet valide l'identifiant  
soumis
- ⑤ Le servlet retourne un  
document XML contenant le  
résultat de la validation
- ⑥ L'objet XMLHttpRequest  
appelle la fonction callback()  
et traite ce résultat
- ⑦ Mise à jour de la page  
HTML (DOM)



# Anatomie d'une interaction AJAX

## Coté client :

la fonction JavaScript invoqué à chaque événement  
"keyup" sur le champ de saisie

```
var req;

function validateUserId() {

    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }

    req.onreadystatechange = processRequest;

    if (!target)
        target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);

    req.open("GET", url, true);

}
```

2

## Création et configuration d'un objet XMLHttpRequest

Selon le navigateur l'objet  
**XMLHttpRequest** est crée différemment

fonction callback : fonction  
Javascript (voir plus loin) qui sera  
invoquée lorsque le serveur aura fini de  
traiter la requête :

*En Javascript les fonctions sont des  
objets et peuvent être manipulées en  
tant que tels*

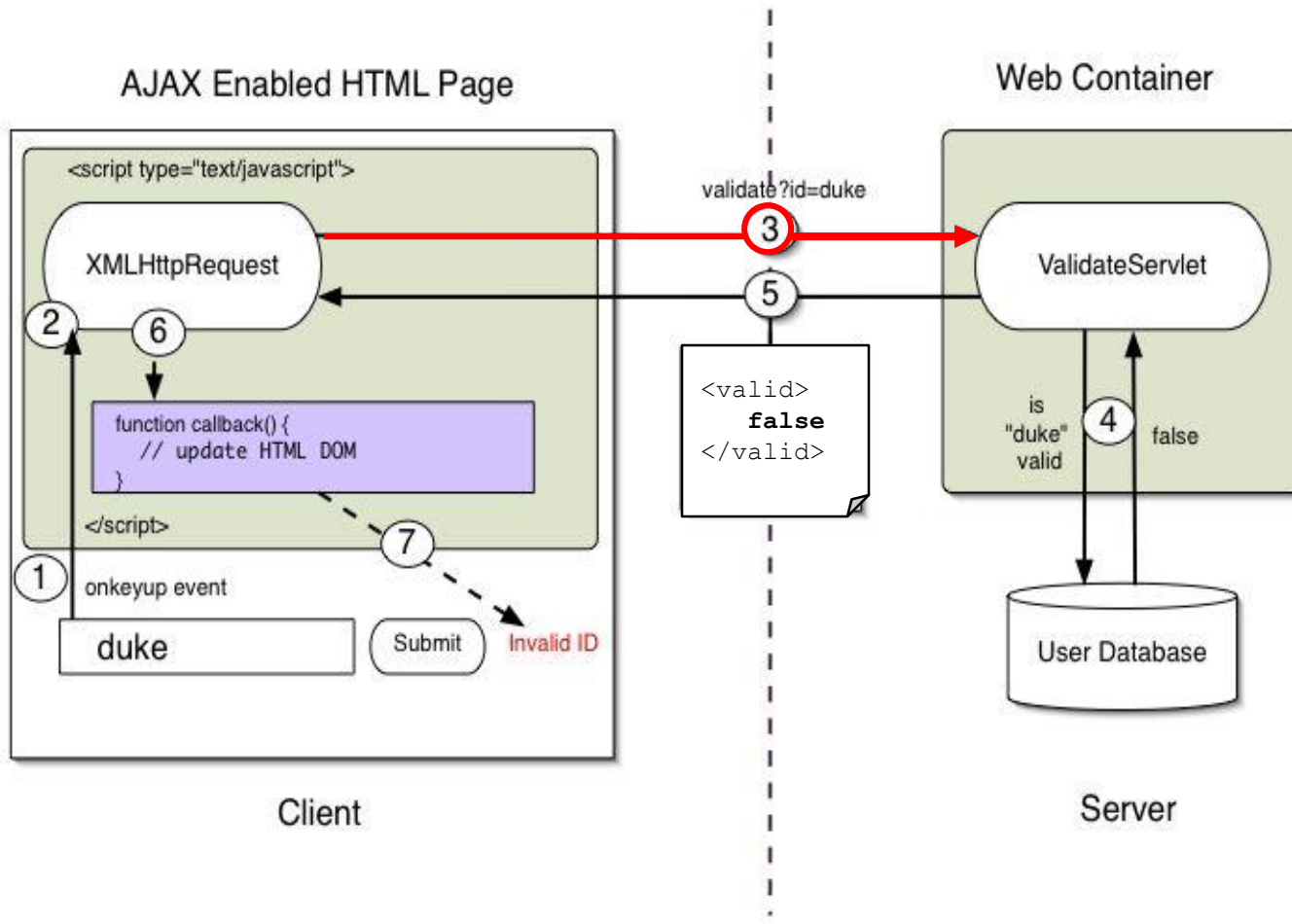
Récupération de la valeur **userid** tapée par  
l'utilisateur (via API DOM)

```
<input type="text" size="20" id="userid"  
name="id" onkeyup="validateUserId()">
```

et construction de l'url du composant  
serveur qui sera invoqué

L'appel sera asynchrone

# Anatomie d'une interaction AJAX



- ① Événement sur le client  
→ appel d'une fonction javascript
- ② Création et configuration d'un objet XMLHttpRequest
- ③ L' objet XMLHttpRequest fait une requête asynchrone
- ④ Le servlet valide l'identifiant soumis
- ⑤ Le servlet retourne un document XML contenant le résultat de la validation
- ⑥ L'objet XMLHttpRequest appelle la fonction callback() et traite ce résultat
- ⑦ Mise à jour de la page HTML (DOM)

# Anatomie d'une interaction AJAX

## Coté client :

la fonction JavaScript invoqué à chaque événement  
"keyup" sur le champ de saisie

```
var req;

function validateUserId() {

    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        isIE = true;
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }

    req.onreadystatechange = processRequest;

    if (!target)
        target = document.getElementById("userid");
    var url = "validate?id=" + escape(target.value);

    req.open("GET", url, true);

    req.send(null);
}
```

2

## Création et configuration d'un objet XMLHttpRequest

Selon le navigateur l'objet XMLHttpRequest est crée différemment

fonction callback : fonction Javascript (voir plus loin) qui sera invoquée lorsque le serveur aura fini de traiter la requête :

*En Javascript les fonctions sont des objets et peuvent être manipulées en tant que tels*

Récupération de la valeur userid tapée par l'utilisateur (via API DOM)

```
<input type="text" size="20" id="userid"
      name="id" onkeyup="validateUserId()">
```

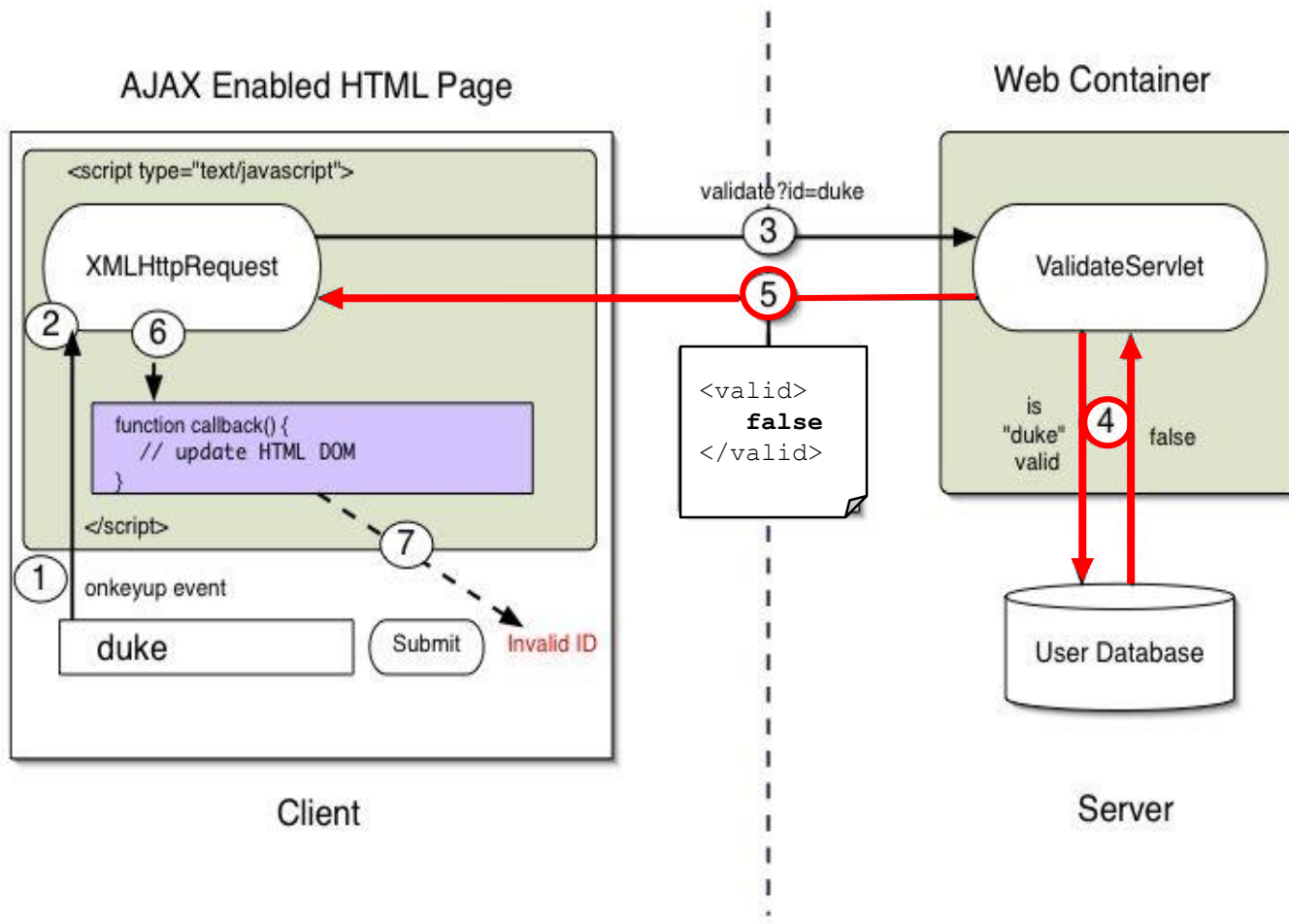
et construction de l'url du composant serveur qui sera invoqué

L'appel sera asynchrone

3

L'objet XMLHttpRequest effectue une requête asynchrone

# Anatomie d'une interaction AJAX



- ① Événement sur le client  
→ appel d'une fonction javascript
- ② Création et configuration d'un objet XMLHttpRequest
- ③ L' objet XMLHttpRequest fait une requête asynchrone
- ④ Le servlet valide l'identifiant soumis
- ⑤ Le servlet retourne un document XML contenant le résultat de la validation
- ⑥ L'objet XMLHttpRequest appelle la fonction `callback()` et traite ce résultat
- ⑦ Mise à jour de la page HTML (DOM)

# Anatomie d'une interaction AJAX

## Coté Serveur :

la servlet traitant la requête GET émise par la fonction  
JavaScript `validateUserId`

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws IOException, ServletException {

    String targetId = request.getParameter("id");

    if ((targetId != null) &&
        LoginManager.validateUserId(targetId.trim())) {

        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().write("<valid>true</valid>");
    } else {

        response.setContentType("text/xml");
        response.setHeader("Cache-Control", "no-cache");
        response.getWriter().write("<valid>false</valid>");
    }
}
```

### 4 Le servlet valide l'identifiant soumis

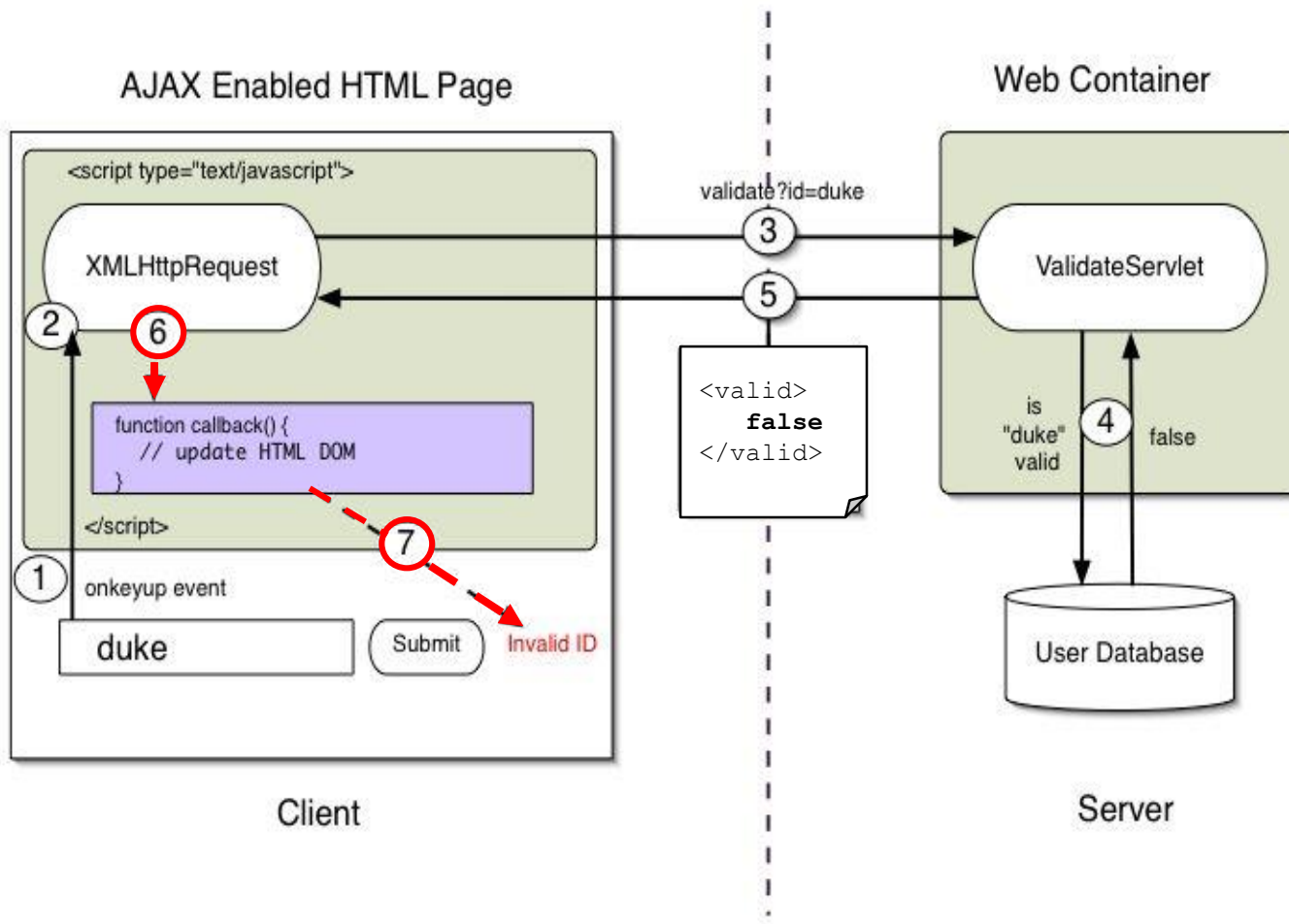
S'agit-il d'un identifiant déjà utilisé ?

### 5 Le servlet retourne un document XML contenant le résultat de la validation

<valid>  
true  
</valid>

<valid>  
false  
</valid>

# Anatomie d'une interaction AJAX



- ① Événement sur le client  
→ appel d'une fonction javascript
- ② Création et configuration d'un objet `XMLHttpRequest`
- ③ L'objet `XMLHttpRequest` fait une requête asynchrone
- ④ Le servlet valide l'identifiant soumis
- ⑤ Le servlet retourne un document XML contenant le résultat de la validation
- ⑥ L'objet `XMLHttpRequest` appelle la fonction `callback()` qui traite ce résultat
- ⑦ Mise à jour de la page HTML (DOM)

# Anatomie d'une interaction AJAX

## Coté client :

### 6 L'objet XMLHttpRequest appelle la fonction callback() et traite ce résultat

```
function processRequest() {
    if (req.readyState == 4) {
        if (req.status == 200) {
            var message = req.responseXML.
                getElementsByTagName("valid")[0].
                childNodes[0].nodeValue;

            setMessageUsingDOM(message);

            var submitBtn = document.getElementById("submit_btn");
            if (message == "false") {
                submitBtn.disabled = true;
            } else {
                submitBtn.disabled = false;
            }
        }
    }
}
```

Cette fonction est invoquée chaque fois que le champ readyState de l'objet XMLHttpRequest est modifié

readyState == 4 et status = 200 indiquent que la réponse a été correctement reçue par le client

Extraction de la valeur true ou false des données retournées par le serveur

**<valid>  
false  
</valid>**

### 7 Mise à jour de la page HTML (DOM)

Affiche dans la zone prévue à cet effet la validité ou non de l'identificateur fourni

Active ou désactive le bouton de soumission du formulaire selon validité de l'identificateur fourni

# Anatomie d'une interaction AJAX

## 7 Mise à jour de la page HTML (DOM)

```
function setMessageUsingDOM(message) {  
  
    var userMessageElement = document.getElementById("userIdMessage");  
  
    var messageText;  
    if (message == "false") {  
        userMessageElement.style.color = "red";  
        messageText = "Invalid User Id";  
    } else {  
        userMessageElement.style.color = "green";  
        messageText = "Valid User Id";  
    }  
  
    var messageBody = document.createTextNode(messageText);  
  
    if (userMessageElement.childNodes[0]) {  
        userMessageElement.replaceChild(  
            messageBody, userMessageElement.childNodes[0]);  
    } else {  
        userMessageElement.appendChild(messageBody);  
    }  
}
```

```
<td>  
    <div id="userIdMessage"></div>  
</td>
```

Invalid User Id

Récupération de l'objet DOM  
correspondant à la zone du message  
grâce à l'id inséré dans le code HTML

Préparation du message

Création du message  
Si il existe déjà un  
message le remplace  
par le nouveau, sinon  
le rajoute



# Comment faire de l'AJAX ?

- Une multitude de solutions pour faire de l'AJAX. Plus de 210 outils dénombrés en mai 2007

(source <http://ajaxian.com/archives/210-ajax-frameworks-and-counting>)

## Pure Javascript

Multipurpose	37
Remoting	19
Graphics and Effects	6
Flash	3
Logging	5
XML	6
Specialised	3
<i>Subtotal</i>	<i>79</i>

## Server-Side


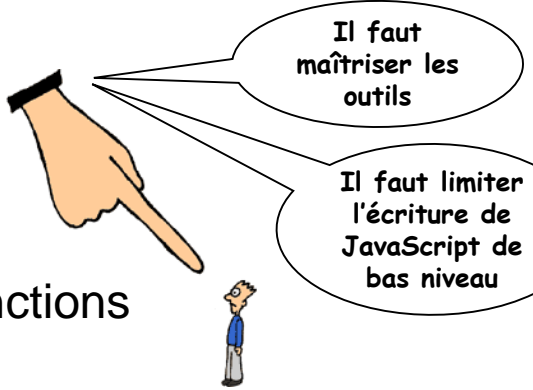
4D	1
C++	1
Coldfusion	4
Eiffel	0
DotNet (+ASP/C*)	19
Java	44
Lisp	1
Lotus Notes	2
Multi-Language	11
Perl	2
PHP	38
Python	5
Ruby	1
Smalltalk	1
Tcl	1
<i>Subtotal</i>	<i>131</i>

## Moralité :

*"If you're still rolling your own XMLHttpRequests or visual effects, now might be a good time to start investigating the alternatives".*

Michael Mahemoff , Ajaxian, mai 2007

# Développer en AJAX

- L'une des difficultés d'AJAX est le développement Javascript
    - Test et débogage pas toujours facile
      - Les choses se sont grandement améliorées grâce à
        - des plugins tels FireBug  **Firebug** web development evolved
        - et maintenant aux outils de développement intégrés aux navigateurs
    - Portabilité difficile, il faut différentes versions des fonctions selon les navigateurs
- 
- 1<sup>ère</sup> solution : utilisation des bibliothèques javascript
    - Prototype, Script.aculo.us, DOJO, Yahoo UI, **JQuery**
  - 2<sup>ème</sup> solution : Utilisation des bibliothèques de tags, composants Struts ou JSF "ajaxifiés"
    - Les composants génèrent du javascript (AjaxTags, AjaxAnywhereRichFace, ajax4JSF, IceFaces...)
  - 3<sup>ème</sup> solution : traduction d'un langage en Javascript
    - GWT (Google Web Toolkit) java → javascript