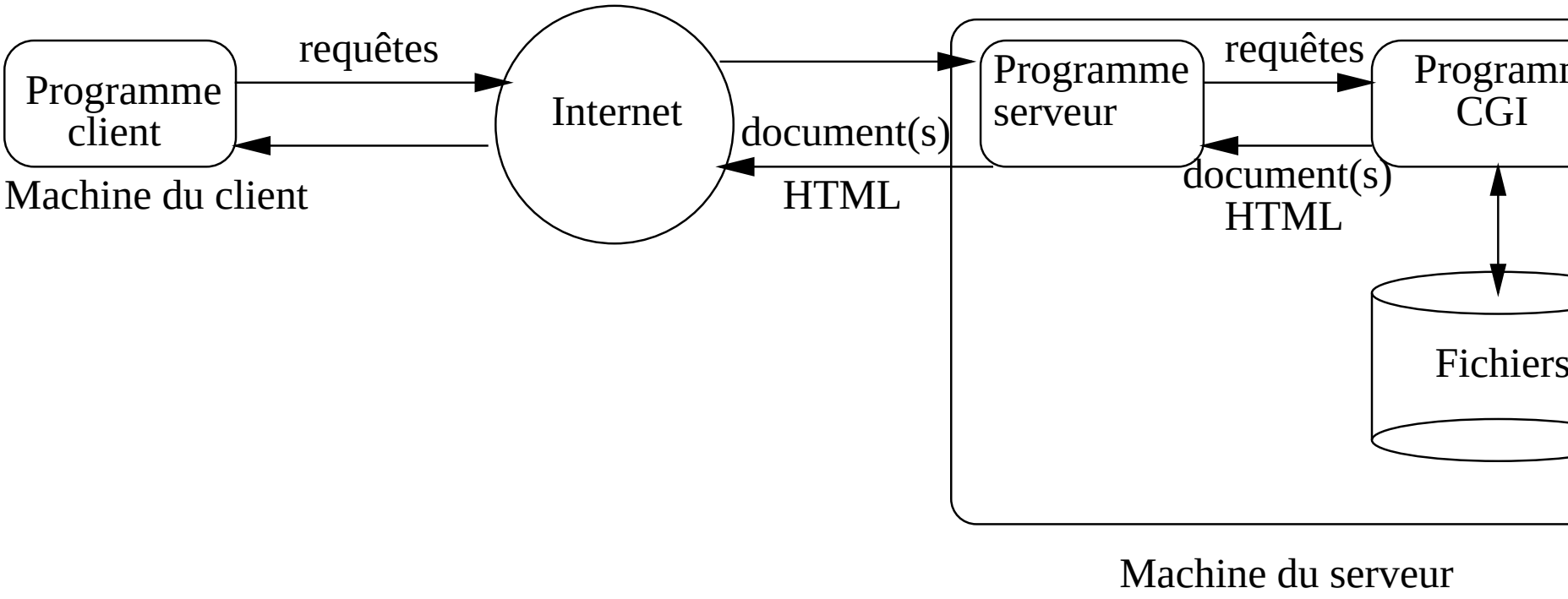
# *Échange et mise à jour de documents XML via le Web*

# *CGI : Common Gateway Interface*

Principe : on déclenche une action (un programme) sur la machine serveur

- On peut transmettre des informations (typiquement saisies dans un formulaire)
- Le programme CGI renvoie un document
- le moyen le plus ancien de créer **dynamiquement** des pages HTML

# Architecture CGI



# *Le programme CGI*

Lecture des paramètres

- transmis par le programme serveur

Traitement de la demande

- tout est permis ! Accès BD, génération d'images, ...

Production du document

- il est renvoyé au programme serveur, qui le transmet au navigateur

# *Transmission client => serveur*

En principe à partir d'un formulaire HTML (mais pas obligatoire !)

- Les arguments sont fournis dans une chaîne au format `nom1=val1&nom2=val2&...`
- Deux méthodes principales:

  `GET` : la chaîne est transmise dans l'URL

  `POST` : la chaîne est transmise séparément

# *Transmission serveur => programme*

La requête est transmise sous forme de variables d'environnement

- `SERVER_NAME` : nom du serveur
- `HTTP_USER_AGENT` : nom du client
- `REQUEST_METHOD` : `GET` ou `POST`
- `QUERY_STRING` : la chaîne des paramètres
- `CONTENT_LENGTH` : longueur de la chaîne

# *Action du programme*

Trois phases:

- Décryptage des paramètres (pénible !)
- Traitement (accès BD, calculs, etc)
- Écriture du résultat sur la sortie standard (*stdout*)

Peut être n'importe quel script (shell, Perl) ou programme exécutable (C, C++)

# *Un programme CGI très simple !*

```
#!/bin/sh
echo Content-type: text/plain
echo
echo Logiciel serveur = $SERVER_SOFTWARE
echo Machine serveur = $SERVER_NAME
echo Type de requête = $REQUEST_METHOD
echo Navigateur = $HTTP_USER_AGENT
echo La requête = $QUERY_STRING
echo Longueur requête = $CONTENT_LENGTH
```

Démo

# *Formulaires HTML:* <FORM>

```
<FORM ACTION='SimSple.cgi'
METHOD='GET' NAME='monForm'>
```

- ACTION est l'URL du script ou du programme à déclencher (en principe, sur le serveur)
- METHOD est GET ou POST (meilleur)
- NAME est le nom du formulaire (utile pour les contrôles JavaScript)
- ENCTYPE est le mode d'encodage des informations.

# *Exemple : un formulaire HTML (*Démo*)*

```
<HTML><HEAD>
<TITLE>Simulation financière</TITLE>
</HEAD><BODY bgcolor="WHITE">

Entrez le montant, le taux et la durée :

<FORM ACTION="SimSple.cgi" METHOD="GET">

Mt : <INPUT TYPE=TEXT SIZE=20 NAME="mtfin"><
Taux : <INPUT TYPE=TEXT SIZE=20 NAME="taux">

Durée : <SELECT NAME='duree'>
            <OPTION VALUE="6">6 mois</OPTION>
            <OPTION VALUE="12">1 an</OPTION>
            <OPTION VALUE="24">2 ans</OPTION>
          </SELECT></TD>

<INPUT TYPE=SUBMIT VALUE="Exécuter">
</FORM></BODY></HTML>
```

# *Champs de formulaire :* <INPUT>

Très général: saisie de texte, ou choix dans des listes.
L'attribut TYPE vaut :

- TEXT pour les chaînes de caractères
- HIDDEN pour les champs cachés
- CHECKBOX pour un choix multiple
- RADIOBOX pour un choix exclusif
- SUBMIT pour déclencher l'action
- FILE pour transmettre un fichier

# *Champs HIDDEN et SUBMIT (*Démo*)*

```
<HTML><HEAD>
  <TITLE>Balise INPUT-SUBMIT</TITLE>
</HEAD><BODY bgcolor='white'>

<FORM ACTION='ExHTTP.cgi' METHOD='GET'>
  <P>
  <INPUT TYPE='HIDDEN'
          VALUE='valeur' NAME='champ'>
  <P>
  <INPUT TYPE='SUBMIT'
          VALUE='Pas le choix !'>
</FORM>

</BODY></HTML>
```

# *Champs de saisie de texte (*Démo*)*

```
<HTML><HEAD>
  <TITLE>Exemple de la balise INPUT</TITLE>
</HEAD><BODY bgcolor='white'>

<FORM ACTION='ExHTTP.cgi'
      METHOD='GET'>
  <P>
  Texte :  <INPUT TYPE='TEXT'
                  SIZE=20 NAME='texte'> <P>
  Nombre : <INPUT TYPE='TEXT'
                  SIZE=4 NAME='nombre' VALUE
  <P>
  <INPUT TYPE=SUBMIT VALUE='Allons-y'>
</FORM>

</BODY></HTML>
```

# Boutons radio

```
<HTML><HEAD>
  <TITLE>Exemple de la balise INPUT-RADIO</T
</HEAD><BODY bgcolor='white'>

<FORM ACTION='ExHTTP.cgi' METHOD='GET'>
  <P>
    Le beurre : <INPUT TYPE='RADIO'
                       VALUE='1' NAME='choix'>
    L'argent du beurre : <INPUT TYPE='RADIO'
                       VALUE='2' NAME='choix'>
  <P>
  <INPUT TYPE=SUBMIT VALUE='Allons-y'>
</FORM>

</BODY></HTML>
```

Démo

# *Transfert de fichiers*

```
<HTML><HEAD>
  <TITLE>Exemple de la balise INPUT-FILE</TI
</HEAD><BODY bgcolor='white'>

<FORM ACTION='ExHTTP.cgi'
   METHOD='GET' ENCTYPE='multipart/form-data
 <P>
Fichier:<INPUT TYPE='FILE' SIZE=20 NAME='fil
  <P>
  <INPUT TYPE=SUBMIT VALUE='Allons-y'>
</FORM>

</BODY></HTML>
```

Démo

# Menus déroulant

```
<HTML><HEAD>
  <TITLE>Balise INPUT-SELECT</TITLE>
</HEAD><BODY bgcolor='white'>

<FORM ACTION='ExHTTP.cgi' METHOD='GET'>
  Choix :
  <SELECT NAME='choix'>
   <OPTION VALUE='1'>Saucisson
   <OPTION VALUE='2'>Rillettes
   <OPTION VALUE='3'>Paté
  </SELECT>
  <P>
  <INPUT TYPE=SUBMIT VALUE='Allons-y'>
</FORM>

</BODY></HTML>
```

Démo

# ***Champ*** <TEXTAREA>

```
<HTML><HEAD>
  <TITLE>Balise INPUT-SUBMIT</TITLE>
</HEAD><BODY bgcolor='white'>

<FORM ACTION='ExHTTP.cgi' METHOD='GET'>
  <P>
  <TEXTAREA NAME='champ' COLS='30' ROWS='4'>
      Entrez votre texte
  </TEXTAREA>
  <BR>
  <INPUT TYPE=SUBMIT VALUE="C'est parti">
</FORM>

</BODY></HTML>
```

Démo

# *Quelques limites du CGI*

- Passage des paramètres : pénible
- Pas d'intégration avec HTML
- Lancement d'un programme à chaque requête !
- Mode totalement déconnecté : pas de mémoire des requêtes précédentes

Améliorations avec des langages récents (PHP, ASP, JSP)

# *Langages de scripts*

PHP, ASP, JSP : de nombreux points communs

- Code inclus dans du texte HTML
- Interpréteur inclus dans le serveur
- Instructions exécutées par le serveur
- Décodage automatique des variables provenant du client

Dernier problème : les échanges client/serveur en mode dé-connecté

# *Applications Java/Web*

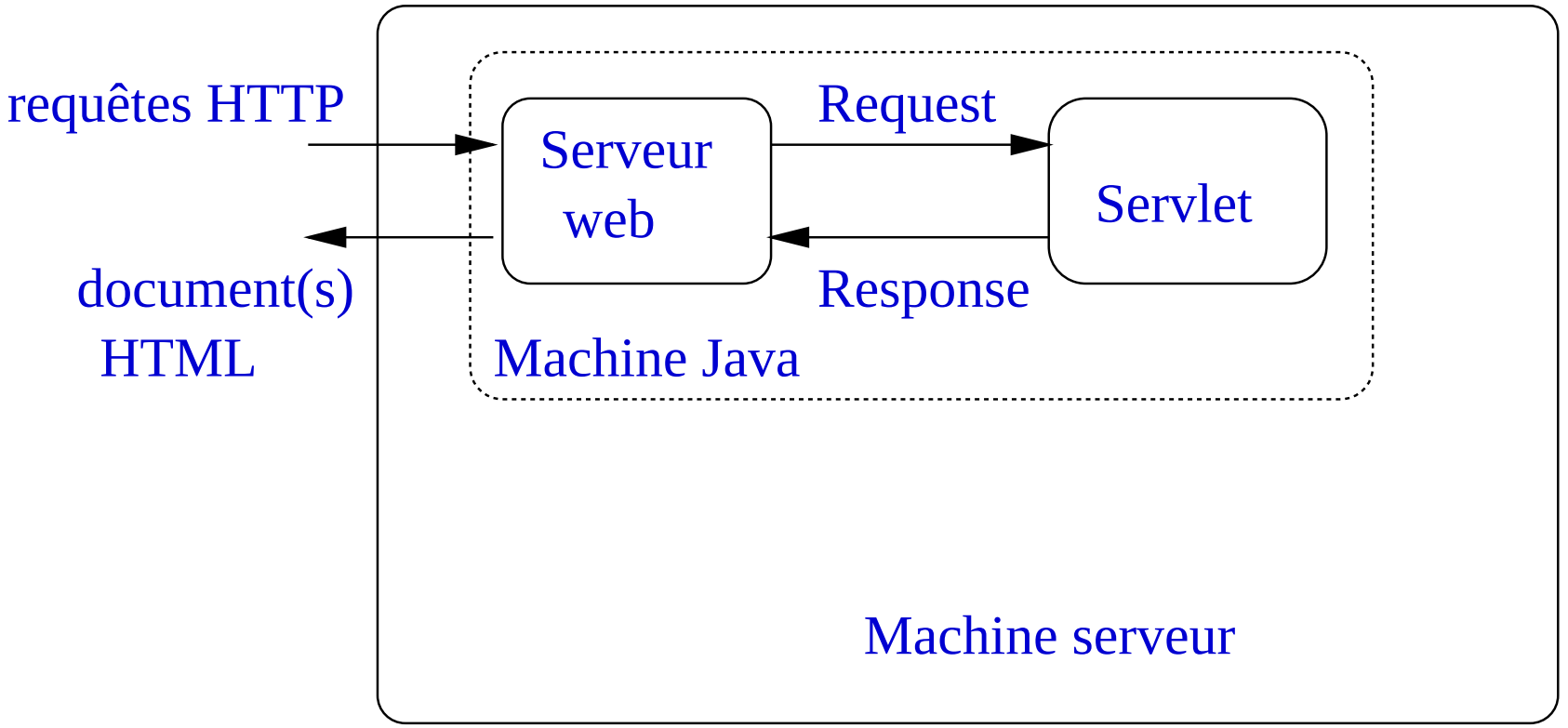
# *Les servlets*

Une servlet, c'est un objet intégré au serveur !

- qui hérite de `HTTPServlet`
- qui reçoit une requête (sous forme d'objet)
- qui effectue tous les calculs et traitements
- qui envoie une réponse (grâce à un objet)

On peut faire appel à tous les outils Java.

# Architecture Servlet



# *Un exemple de servlet (*Démo*)*

```
import java.io.*; import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ExServ1 extends HttpServlet {
 public void doGet(HttpServletRequest reques
                   HttpServletResponse response)
     throws IOException, ServletException
{
 PrintWriter out = response.getWriter();

 out.println("<html><head><title>Babar</titl
 out.println("</head><body bgcolor='white'><

 out.println("<b>Date</b>" + new Date());
 out.println("</body></html>");
 }
}
```

# *Programmation CGI et Servlet*

Une servlet, ça sert d'abord à rendre plus propre et plus facile la programmation CGI. Exemples :

- l'objet `request` fournit un service `getParameter`
- l'objet `response` fournit un service `setContentType`

On dispose d'une interface Java pour toute la programmation CGI

# *Formulaire + Servlet*

```
<HTML><HEAD>
<TITLE>Dis bonjour au monsieur</TITLE>
</HEAD><BODY bgcolor="WHITE">

<FORM
ACTION="http://localhost:8080/rigaux/servlet
METHOD="GET">

Votre nom :
  <INPUT TYPE=TEXT SIZE=20 NAME="nom"><BR>
  <INPUT TYPE=SUBMIT VALUE="Exécuter">

</FORM></BODY></HTML>
```

Démo

# *Le code de la servlet*

```
import java.io.*; import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ExServ2 extends HttpServlet {
 public void doGet(HttpServletRequest reques
                   HttpServletResponse response
    throws IOException, ServletException
{
 response.setContentType("text/plain");
 PrintWriter out = response.getWriter();

 out.println("Bonjour "
             + request.getParameter("nom")
             + " !!");
 }
}
```

# *Servlet et JSP*

Problèmes des servlets : la production de HTML est laborieuse (appels à `out.println()`)
Les *Java Server Pages* :

- inclusion de code Java dans du HTML
- la page HTML + JSP est compilée dans une servlet temporaire
- intégration avec les composants JavaBeans

=> on retrouve une approche semblable à PHP, avec toute la puissance des API Java

# *Exemple de JSP (*Démo*)*

Les parties Java sont marquées par <% et %>

```
<HTML>
<HEAD><TITLE>ExJSP1</TITLE></HEAD>
<BODY bgcolor="white">
 Voici le résultat
<%@ page import="java.io.*,java.util.*" %>
Date : <%= new Date()%>
</BODY>
```

# *Script JSP pour dire bonjour (*Démo*)*

```
<HTML>
<BODY bgcolor="white">
<%@ page import="java.io.*,java.util.*" %>
<%
  if (request.getParameter("nom") == null)
  {
%>
 <%@ include file="FormJSP2.html" %>
<%
  } else
  {
%>
 Bonjour <%=  request.getParameter("nom") %>
<% } %>
</BODY>
```

# *Améliorations possibles*

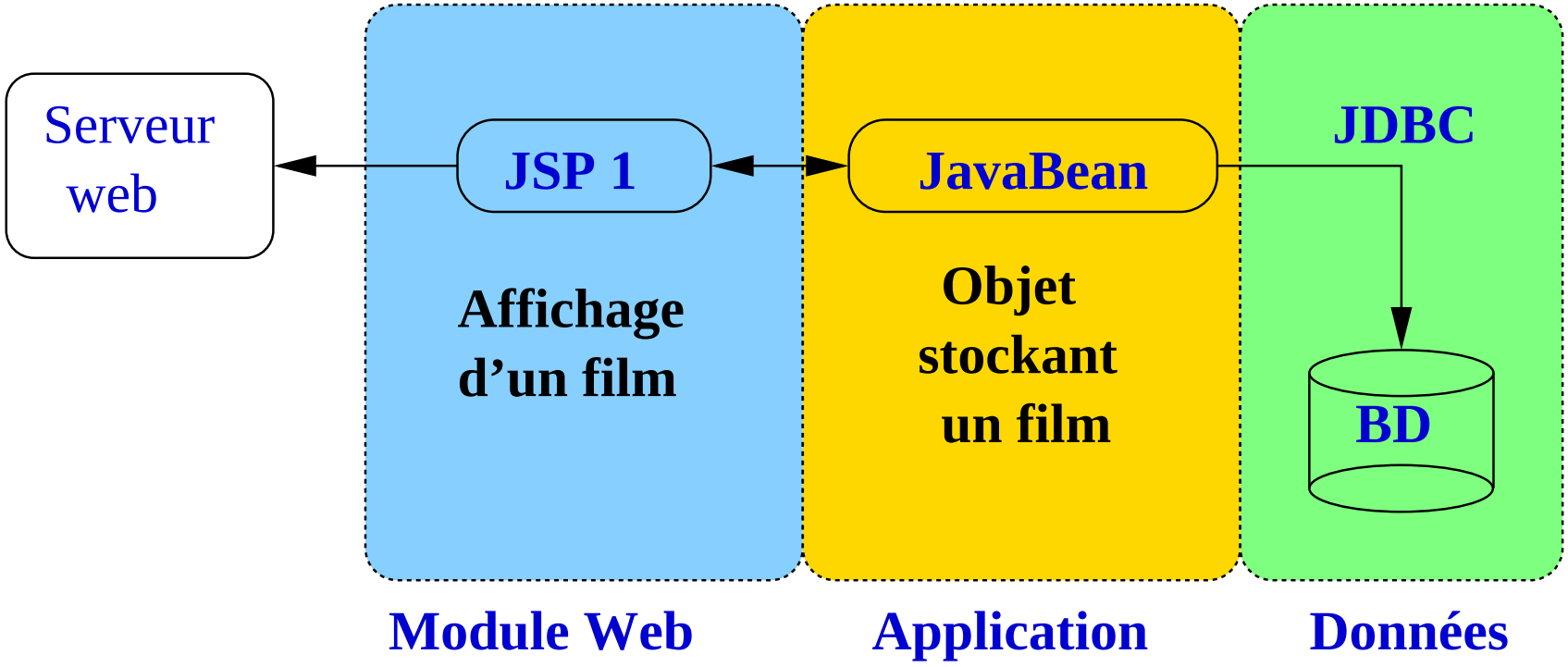
On peut mettre tout le code dans une JSP. Mais ...

- C’est du code dédié au module web
  - on aimerait faire appel à des composants réutilisables dans d’autres contextes
- beaucoup de langages à connaître !
  - on aimerait **séparer les points de vue** (présentation, application, BD...)

# *Programmation Bases de données avec les JSP*

# *Une architecture plus évoluée*

JSP = outil de **présentation**, interagissant avec des composants, et une base de données.



# *Exemple d'un JavaBean*

Indépendant du web.

```
public class ExBean {
   private String titre;
   public ExBean() {
      titre = null;
   }
   public void setTitre(String t) {
     titre = t;
   }
   public String getTitre() {
     return titre;
   }
}
```

# *JSP et Beans (*Démo*)*

Plus de code Java !

```
<HTML><BODY bgcolor="white">
<%@ page import="java.io.*,java.util.*,
     ExBean" %>
<jsp:useBean id="monBean" class="ExBean"/>
<!-- Je stocke le titre dans le bean -->
<jsp:setProperty
      name="monBean" property="titre"/>
<!-- Je vérifie qu'il est bien là -->
 Mon titre est <B>
    <jsp:getProperty
       name="monBean" property="titre"/>
</B>
</BODY></HTML>
```

# *setTitre revue avec JDBC*

```
public void setTitre(String t) {
  titre = t;

  // Connexion à la base
  Connection conn = DriverManager.getConnect
       ("Films", "login", "password");

  // Exécution de la requête
  Statement stmt = conn.createStatement ();
  ResultSet rset = stmt.executeQuery
        ("select * from Film "
        + "where titre='" + titre + "'");

  // On conserve le résultat dans le bean
  annee =  rset.getInt (2);
}
```

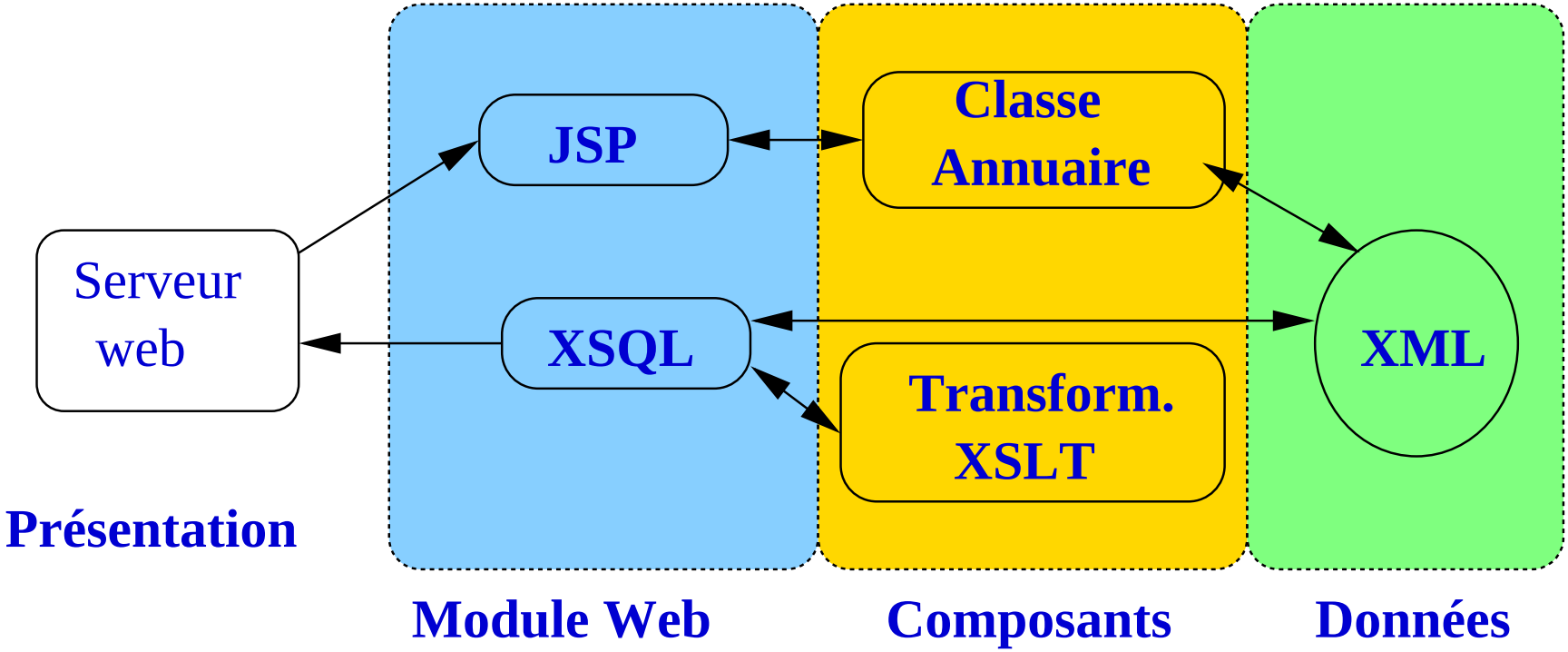
# *Interrogation de la base (*Démo*)*

```
<HTML><BODY bgcolor="white">
<%@ page import="java.io.*,ExBeanJDBC" %>
<jsp:useBean id="monBean" class="ExBeanJDBC"
<!-- Initialisation du film -->
<jsp:setProperty
        name="monBean" property="titre"/>
<!-- Affichage du résultat  -->
Film <jsp:getProperty
          name="monBean" property="titre"/>,
paru en  <jsp:getProperty
          name="monBean" property="annee"/>
</BODY></HTML>
```

# *Programmation XML avec les JSP*

# *Un exemple de base*

XSQL (**présentation**) + JSP (**Programmation Web**) + DOM (**mise à jour du document**)



# *Le document de base*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ANNUAIRE>
  <PERSONNE id='1'>
    <NOM>Amann</NOM>
    <PRENOM>Bernd</PRENOM>
    <EMAIL>amann@cnam.fr</EMAIL>
  </PERSONNE>
  <PERSONNE id='2'>
    <NOM>Rigaux</NOM>
    <PRENOM>Philippe</PRENOM>
    <EMAIL>rigaux@lri.fr</EMAIL>
  </PERSONNE>
</ANNUAIRE>
```

Démo

# *Classe Annuaire.java : constructeur*

```
Document dom;
Node elementRacine ;

public Annuaire (File document)
{
  // Instanciation du parseur
  DocumentBuilderFactory factory =
          DocumentBuilderFactory.newInstance(
  DocumentBuilder builder =
          factory.newDocumentBuilder();
  // Analyse du document
  dom = builder.parse(document);
  elementRacine = dom.getDocumentElement();
}
```

# *Suppression d'un élt. PERSONNE*

```
// Méthode pour supprimer un élément Personn
      étant donné son id
public void supPersonne(int id)
{
  // On prend tous les noeuds PERSONNE
  NodeList noeudList =
       dom.getElementsByTagName("PERSONNE");
 // Parcours de la liste
 for (int i=0;i<noeudList.getLength();i++)
 {
  Element noeud = (Element) noeudList.item(i
  String s = noeud.getAttribute ("id");
  int  val = Integer.parseInt(s);
  if (val == id)
   noeud.getParentNode().removeChild(noeud);
 }
}
```

# *Annuaire.java : éléments simples*

Méthode privée pour ajouter un couple (élément + texte)

```
// Cette méthode crée un élément avec un fil
//      de type Texte
private Element ElTexte
   (Document dom, String nomEl, String texte
{
 Element noeudEl = dom.createElement (nomEl)
 Node noeudText = dom.createTextNode (texte)
 noeudEl.appendChild  (noeudText);
 return noeudEl;
}
```

# *Ajout d'un élément PERSONNE*

```
public void ajoutPersonne
    (String nom, String prenom, String email
{
 Integer id = this.chercheMaxID ();
 // Création du nouvel élément
 Element el = dom.createElement ("PERSONNE")
 elPersonne.setAttribute ("id", id.toString(

 // Ajout des trois fils
 el.appendChild (ElTexte(dom, "NOM", nom));
 el.appendChild
        (ElTexte(dom, "PRENOM", prenom));
 el.appendChild
        (ElTexte(dom, "EMAIL", email));

 // Ajout du nouvel élément
 elementRacine.appendChild (elPersonne);
}
```

# *Le document XSQL*

Construit une page avec les formulaires.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet
    href="Annuaire.xsl"
    type="text/xsl"?>
<page xmlns:xsql="urn:oracle-xsql">
  <xsql:include-xml href="Annuaire.xml"/>
</page>
```

# *Le traitement JSP (1)*

```
<%
 File fdom = new File ("Annuaire.xml");
 String Operation = request.getParameter("op

 // Instanciation du parseur
 Annuaire annuaire = new Annuaire (fdom);

 // Appel a la méthode d'ajout
 if (Operation.equals("add"))
 {
  nom = request.getParameter("nom");
  prenom = request.getParameter("prenom");
  email = request.getParameter("email");
  annuaire.ajoutPersonne (nom, prenom, email
 }
 ...
```

# *Le traitement JSP (2)*

```
...
if (Operation.equals("sup"))
{
  int id = request.getParameter("id");
  annuaire.supPersonne(id);
}

// Sérialisation du résultat
Serialiseur ser =
     new Serialiseur (annuaire.getDOM());
ser.sortie (nomFichier);
%>

<jsp:forward page="Annuaire.xsql"/>
```