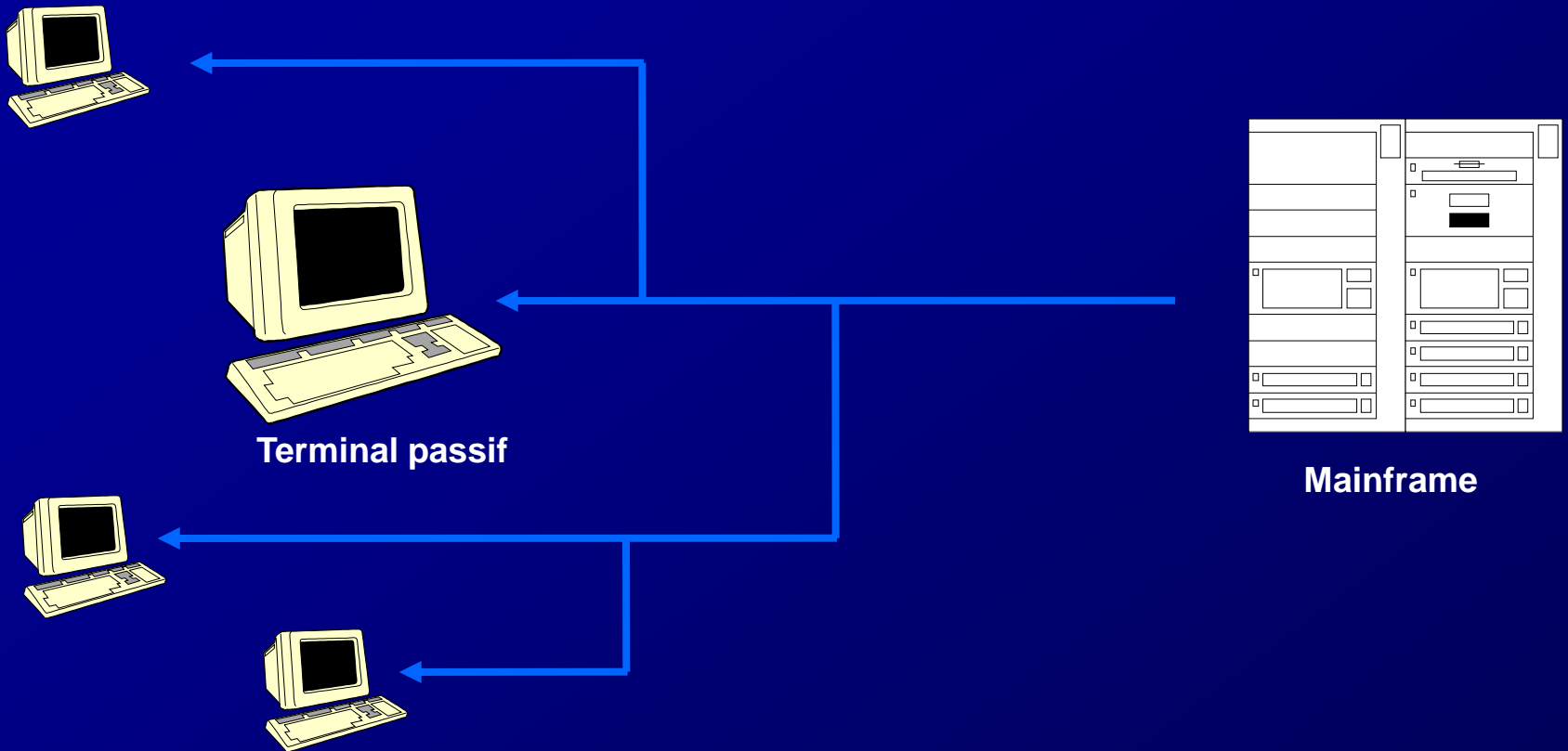


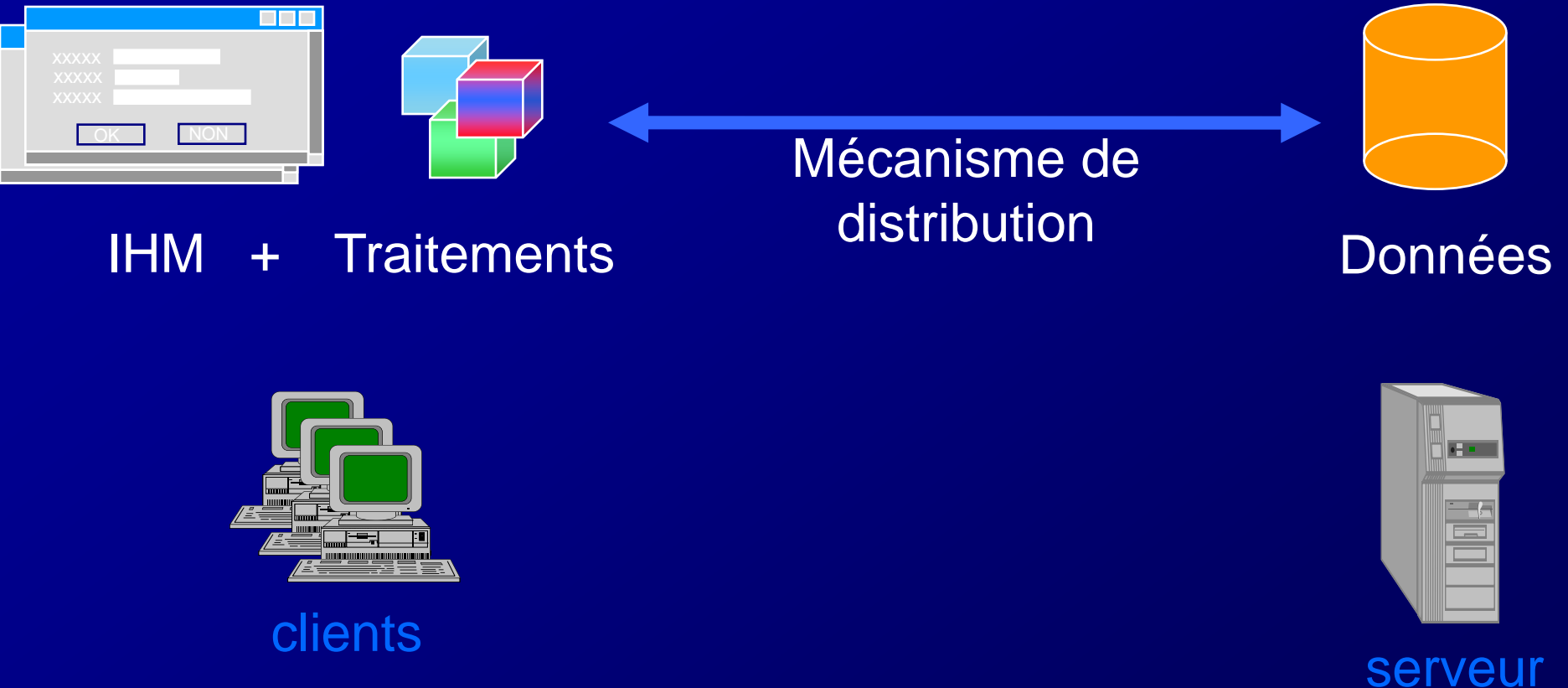
Présentation de l'architecture J2EE

Pierre Lefebvre

Informatique centralisée



Client-serveur à deux niveaux



Architecture à trois niveaux (modèle RPC)

- Clients légers: logique métier et données séparés de la présentation
- Le serveur intermédiaire est requis pour gérer les services



Architecture à trois niveaux (modèle objet)

- La logique métier et le modèle des données sont définis en objets
 - Corba, rmi, dcom



Architecture à trois niveaux (serveur web)

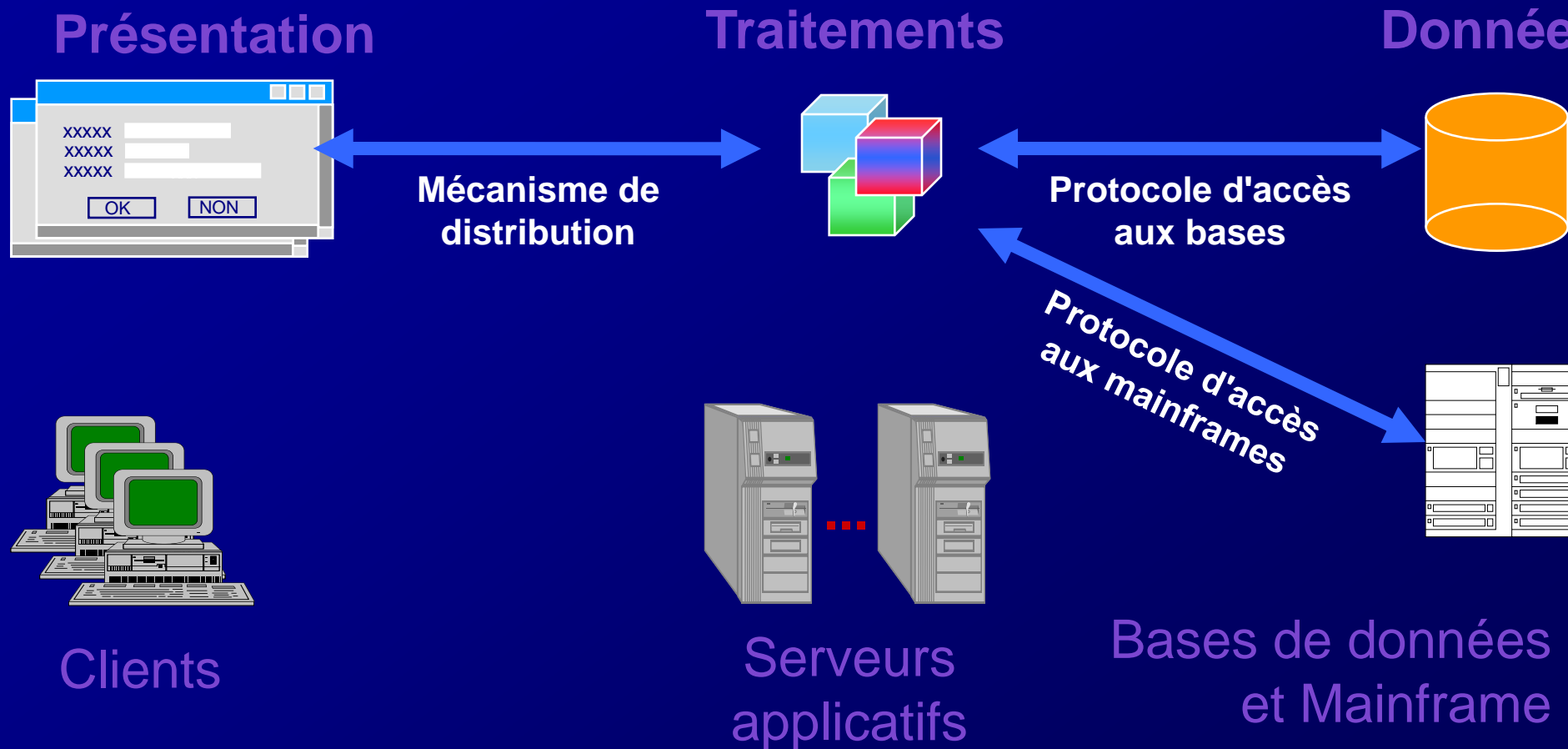
- Le navigateur gère la logique de présentation et communique avec le serveur web via http
- La logique métier et le modèle de données sont gérés par de la génération de contenu dynamique (CGI, Servlet/JSP, ASP)



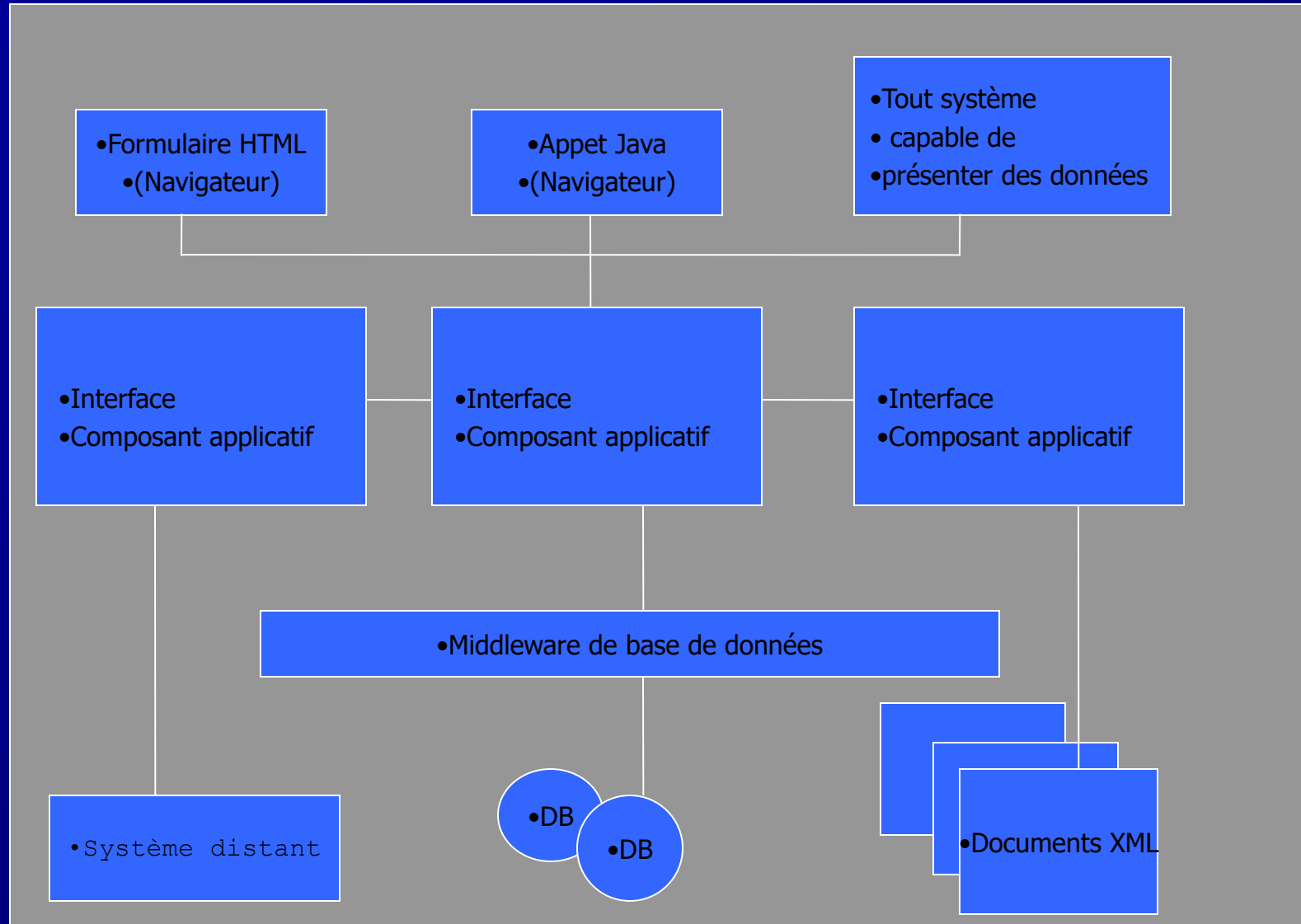
Problèmes

- Comment gérer la complexité du tiers intermédiaire ?
- Comment fournir les services systèmes nécessaires à toute application ?
- Solutions
 - Conteneurs partagés fournissant ces services
 - Solutions propriétaires vs solutions libres

Architecture multiniveau



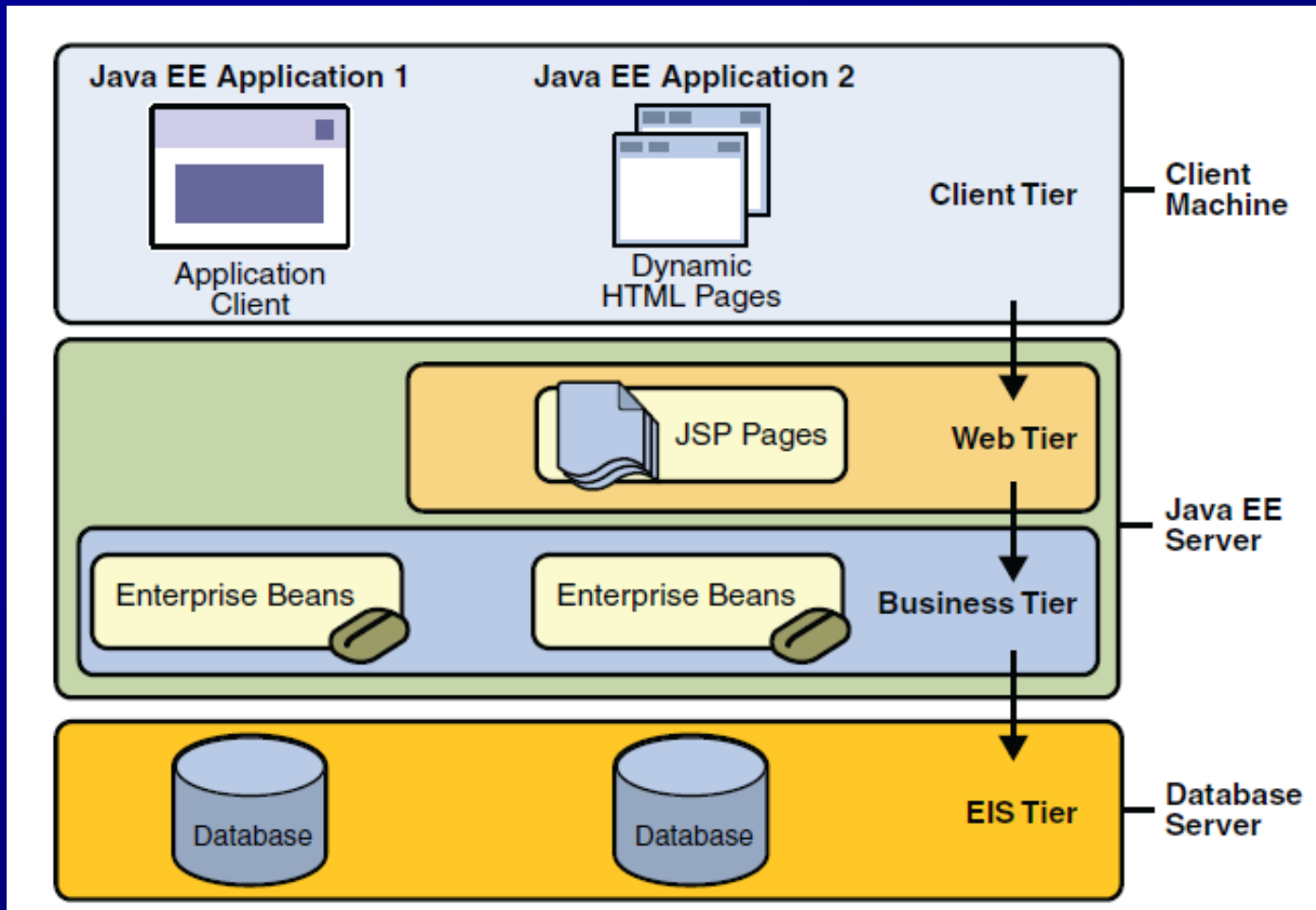
Architecture d'entreprise



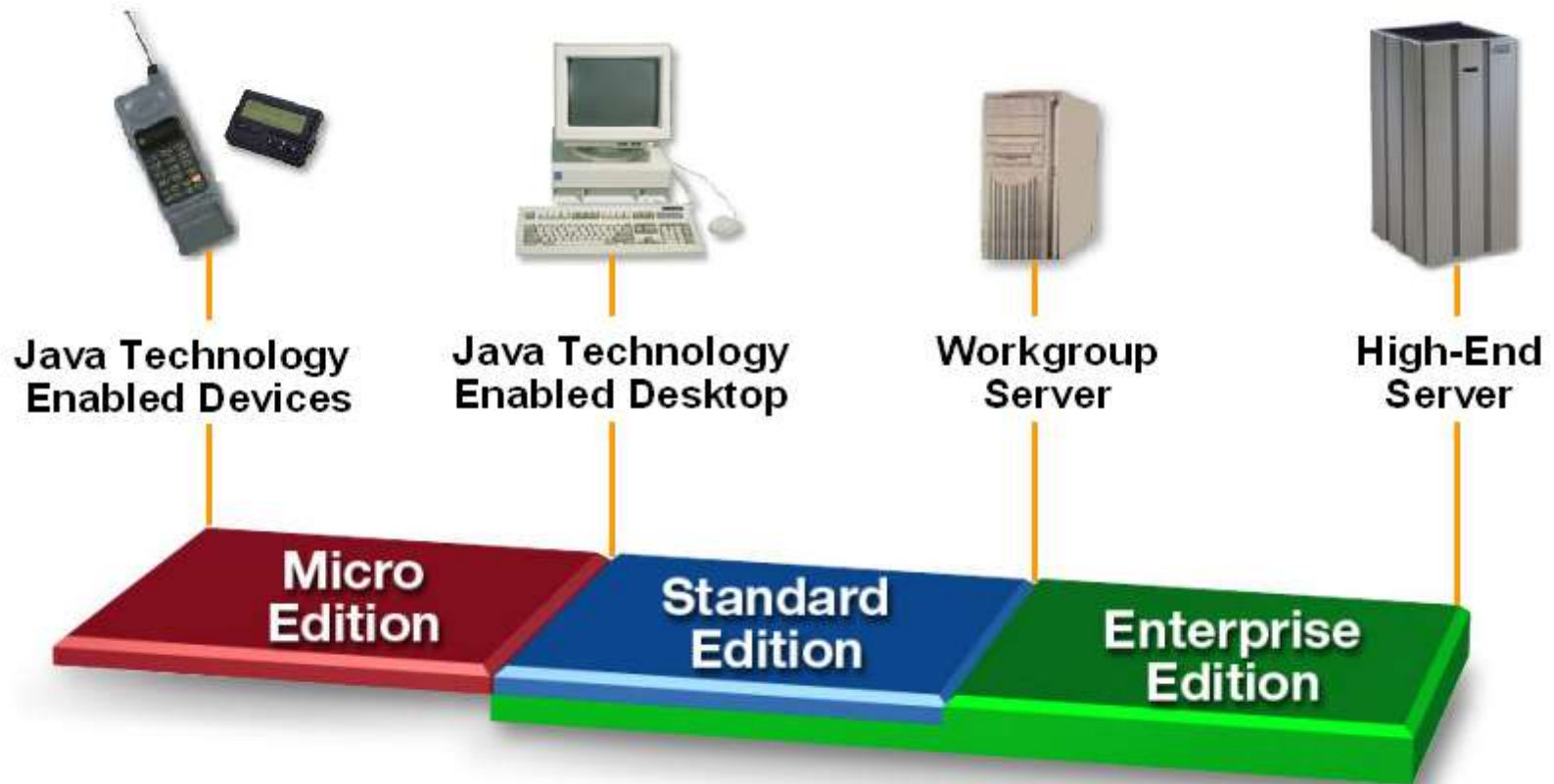
Qu 'est-ce que J2EE ?

- J2EE est une plate-forme ouverte et standard pour développer, déployer et gérer des applications d'entreprise
- C 'est un environnement Java fournissant les outils suivants:
 - une infrastructure d 'exécution pour héberger des applications
 - Un ensemble d '**API** d 'extension Java pour concevoir des applications
- J2EE est avant tout une norme

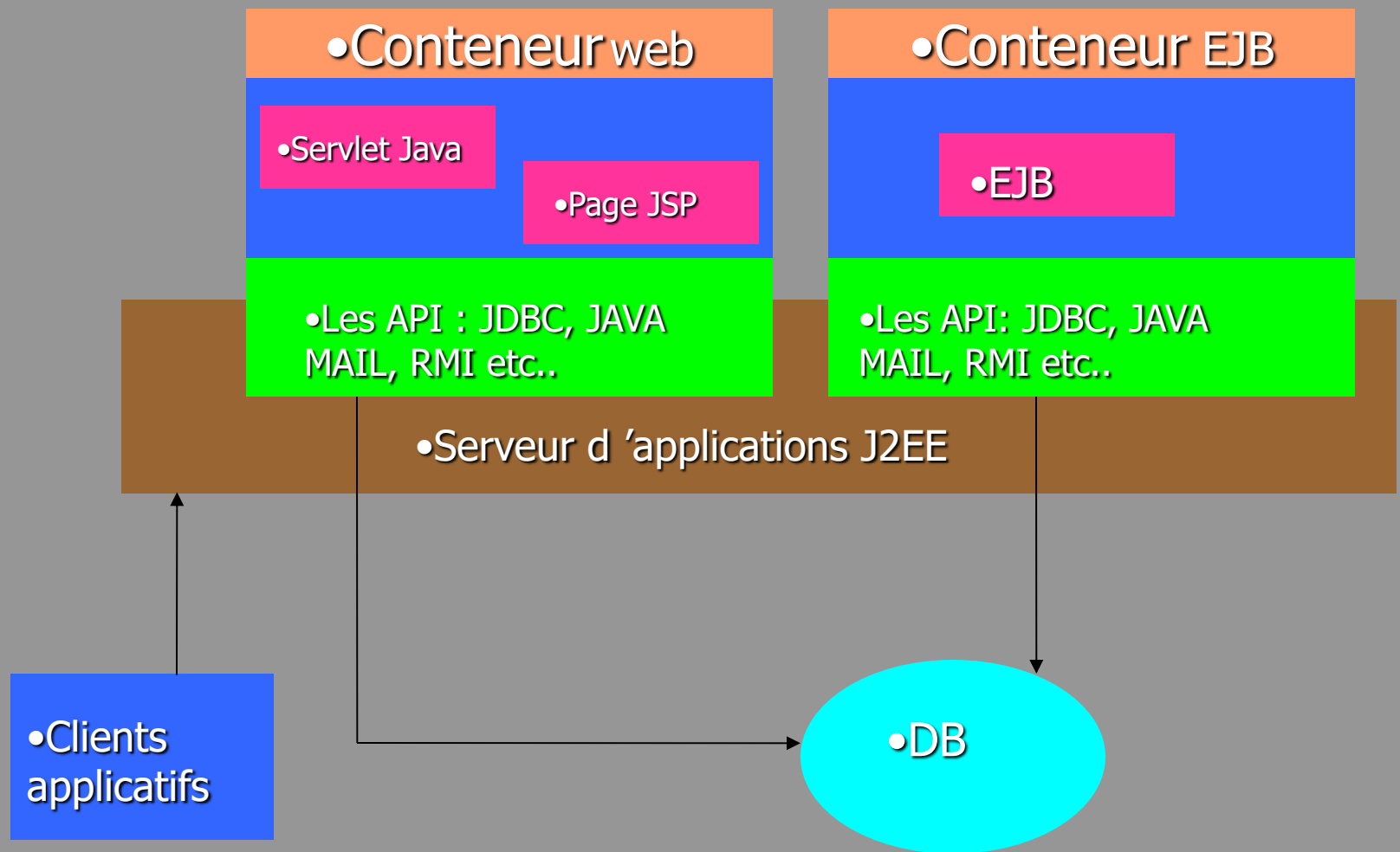
Architecture multi-tiers



The Java™ Platform



Architecture de J2EE



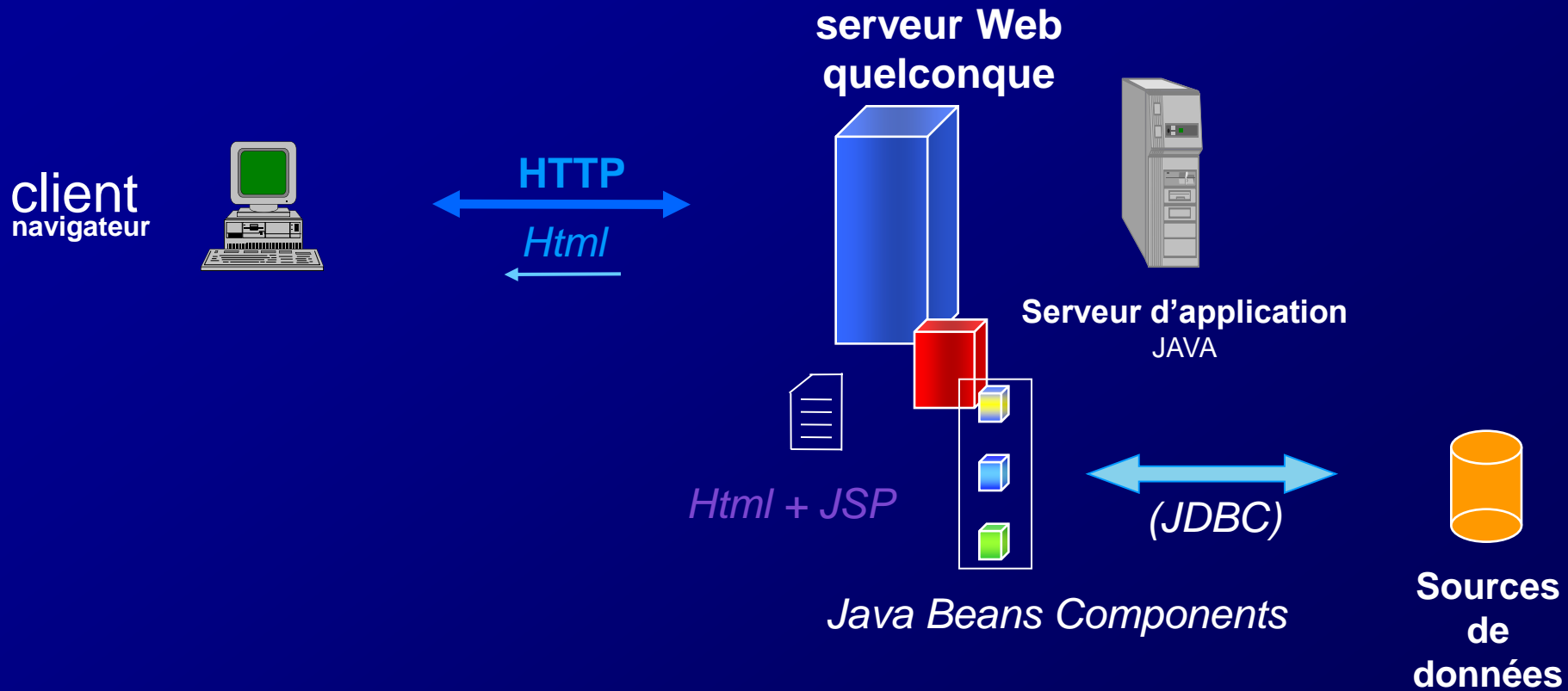
Valeurs ajoutés autour de JEE

- Utilisation possible de différentes implémentations de J2EE
- De nombreuses ressources disponibles
- Développement J2EE à base de composants
- Compétition de la valeur des implémentations
- Pas de maintenance d'API propriétaires mais de meilleurs implémentations de la spécification J2EE

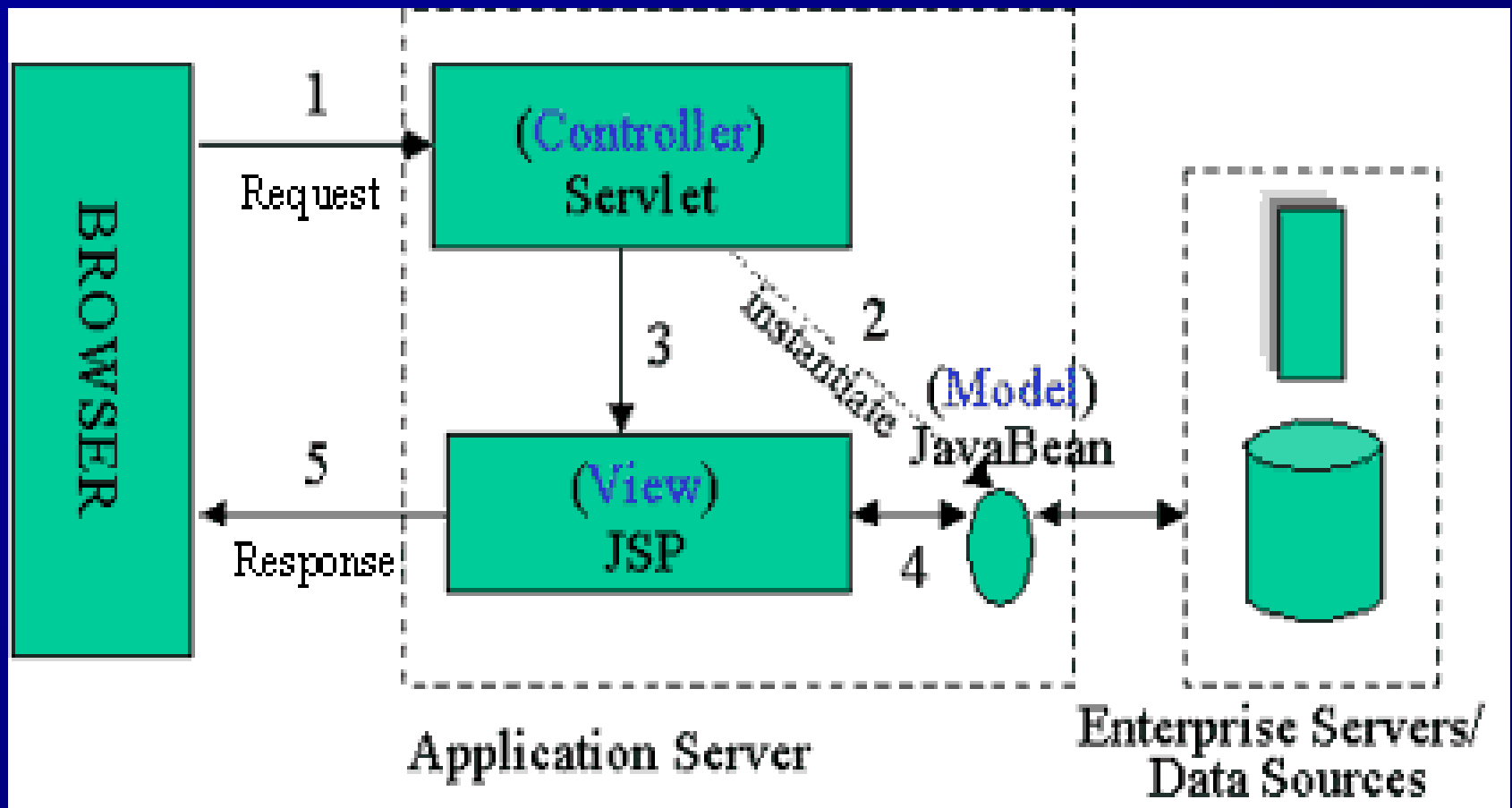
Qu 'est-ce un conteneur J2EE ?

- C 'est un environnement d'exécution chargé de gérer des composants applicatifs et de donner accès aux API J2EE
- En d 'autres termes, des instances des composants applicatifs sont créés et invoquées à l'intérieur de la JVM(Java Virtuel Machine) du conteneur

Exemple 1



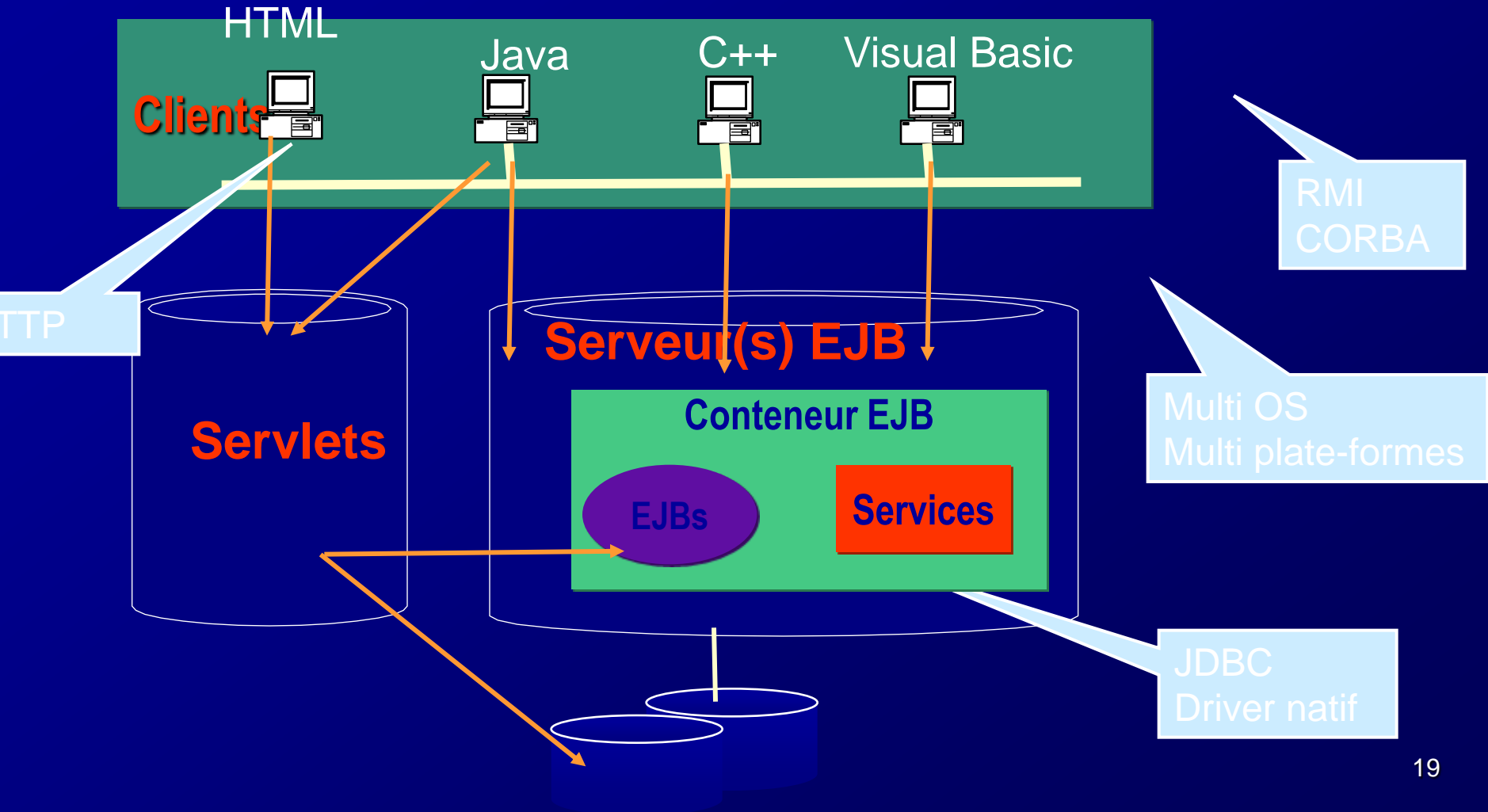
Structure MVC pour J2EE



Principes de fonctionnement de l'architecture MVC

- 1. Le client envoie une requête HTTP à destination d'une servlet
- 2. La servlet récupère les données transmises dans la requête HTTP et délègue les traitements avec ces données à des composants EJB et/ou JavaBean
- 3. Selon les traitements à effectuer, les composants EJB et/ou JavaBean peuvent accéder à des sources de données
- 4. Une fois les traitements terminés, les composants rendent la main à la servlet en lui retournant un résultat. La servlet stocke ce résultat dans un contexte (session, requête...)
- 5. La servlet transmet la suite du traitement de la requête vers une JSP
- 6. La JSP récupère les données stockées par la servlet dans un contexte et génère la réponse HTTP
- 7. La réponse HTTP est renvoyée au client

Exemple 2



Les types d'EJB

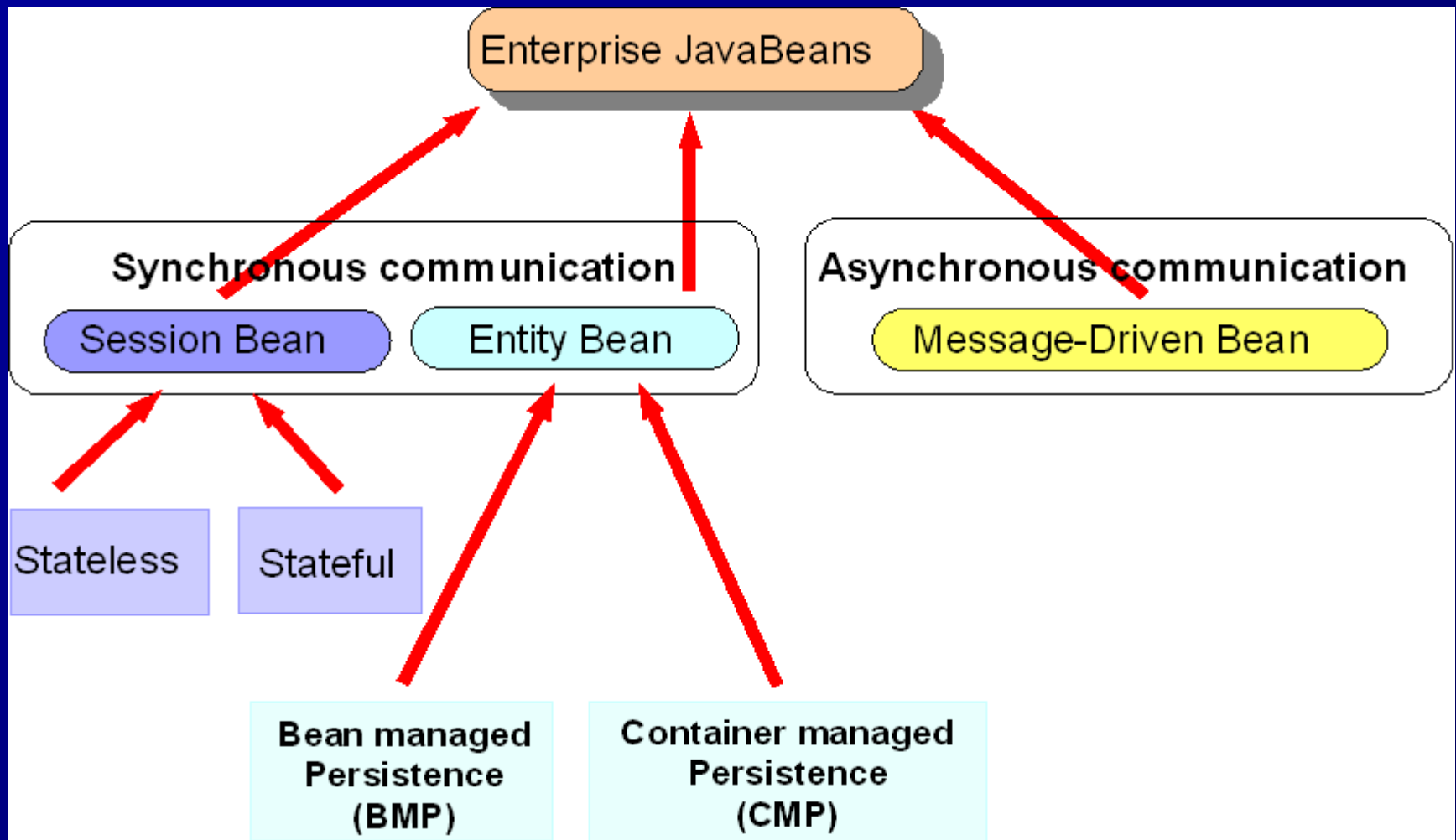
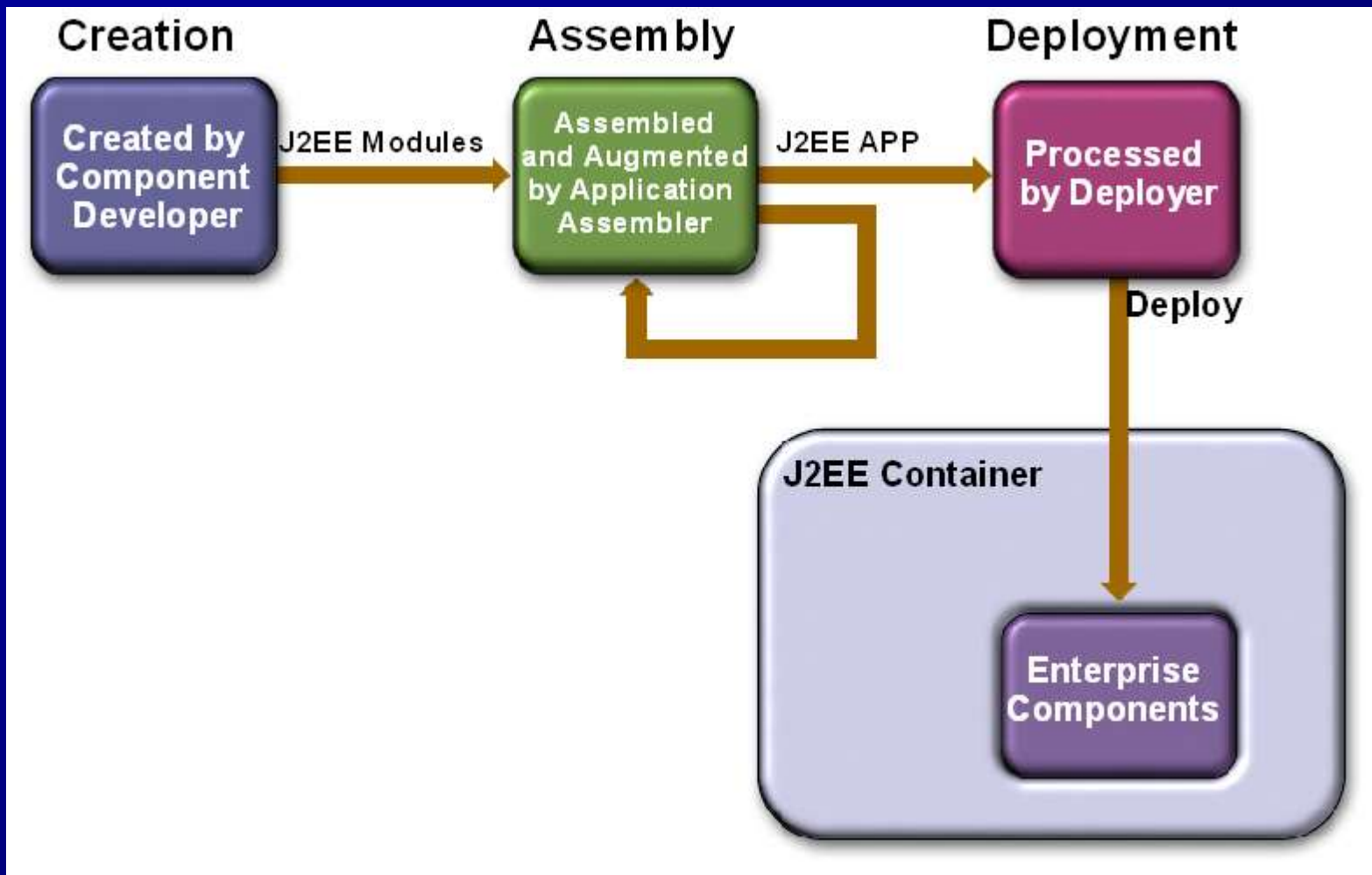
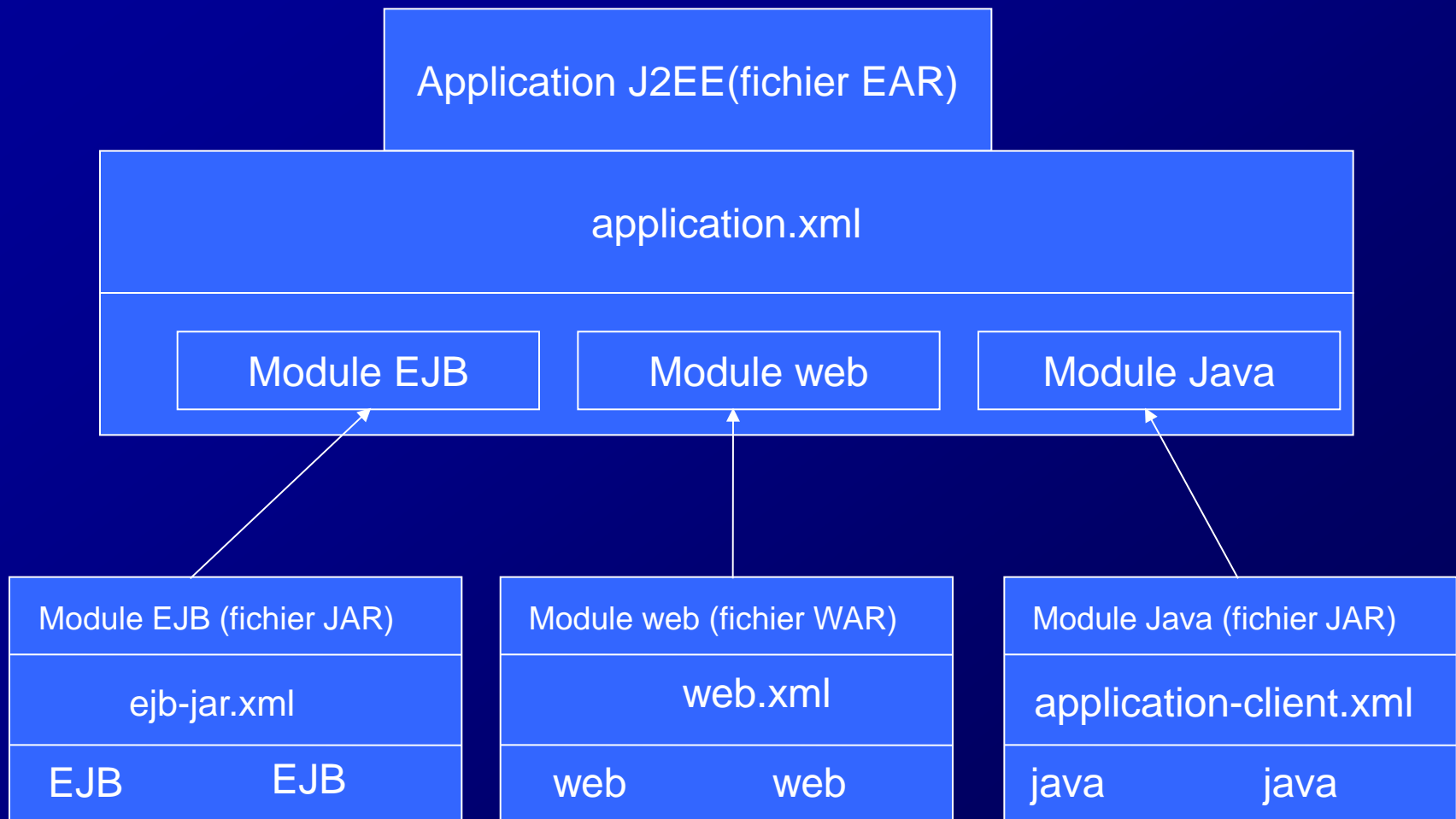


Illustration du cycle de vie d'un développement



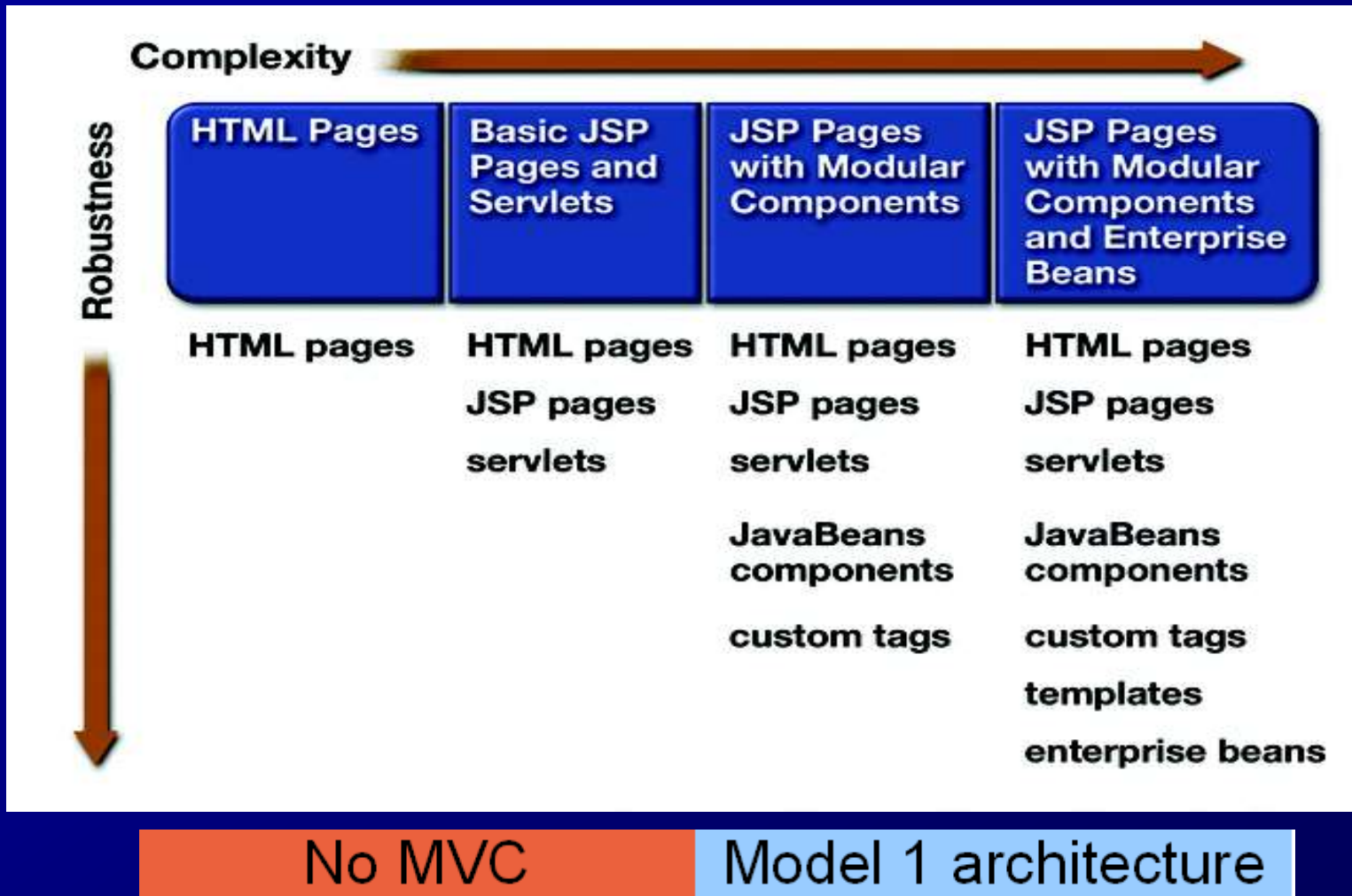
Constitution de modules en application j2EE



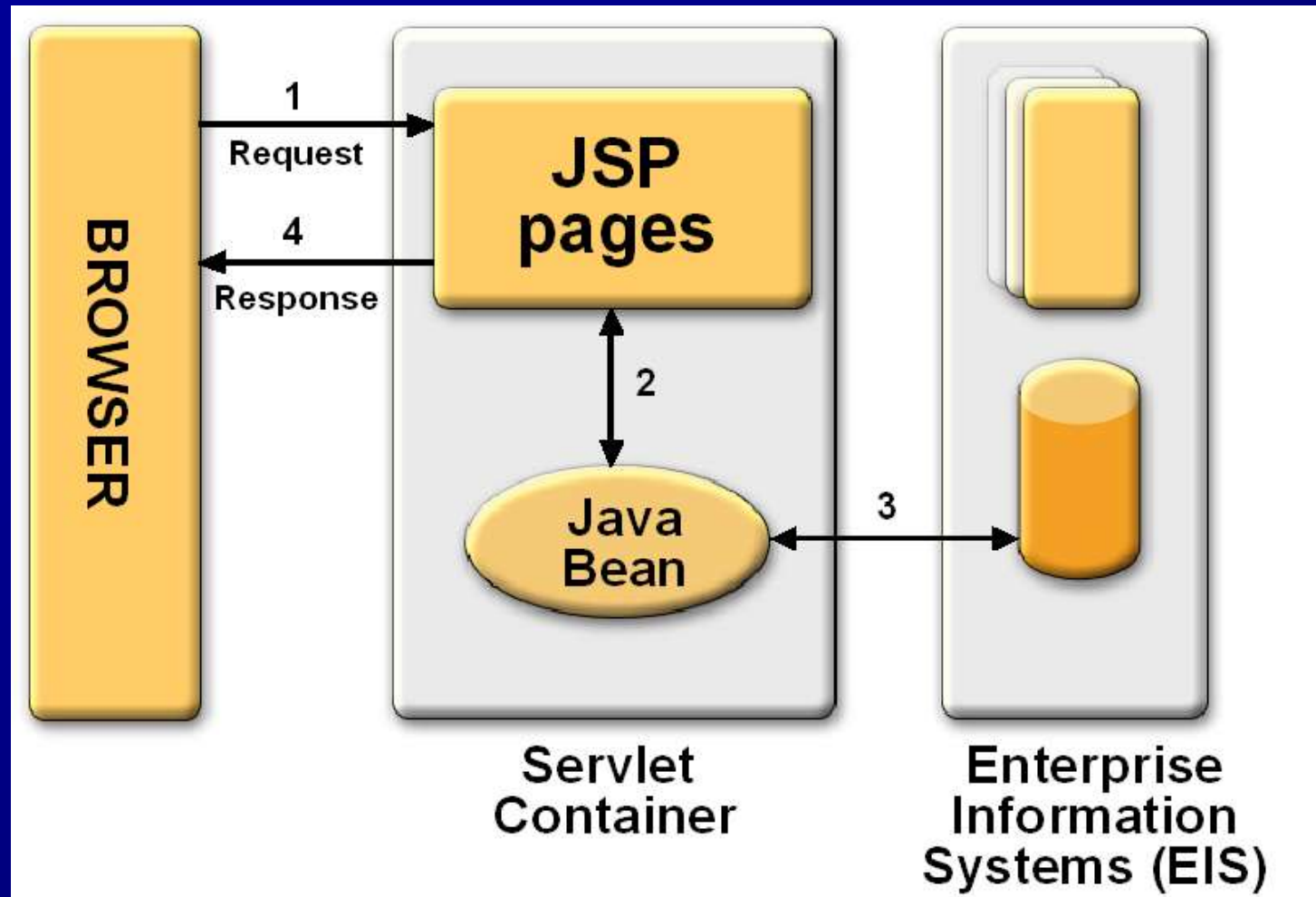
Evolution de l'architecture MVC

- 1. Pas de MVC
- 2. MVC modèle 1 (centré page)
- 3. MVC modèle 2 (centré servlet)
- 4. framework d'applications web
 - Struts
- 5. framework standard d'applications web
 - Javaserwer faces

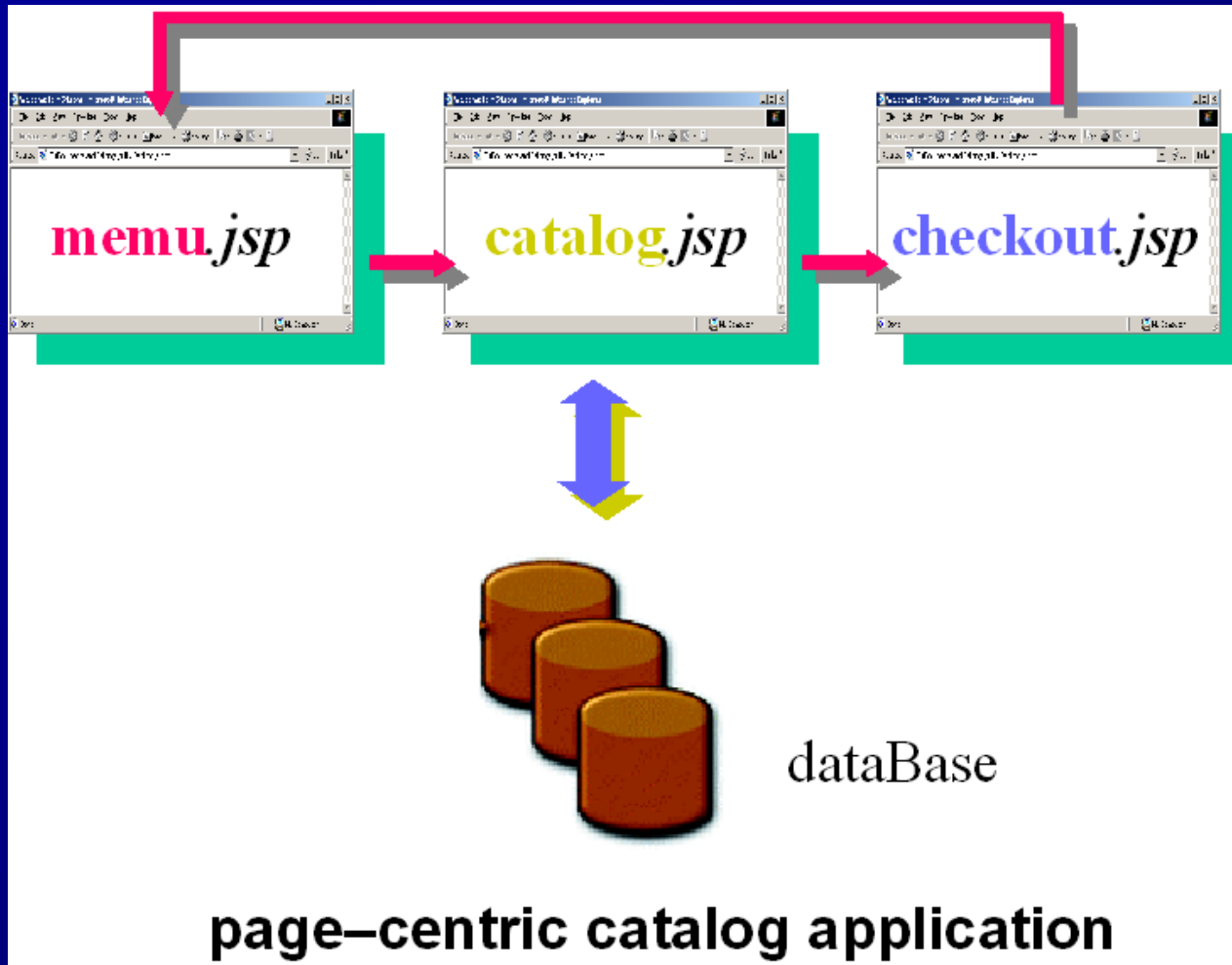
Evolution de la conception des architectures web



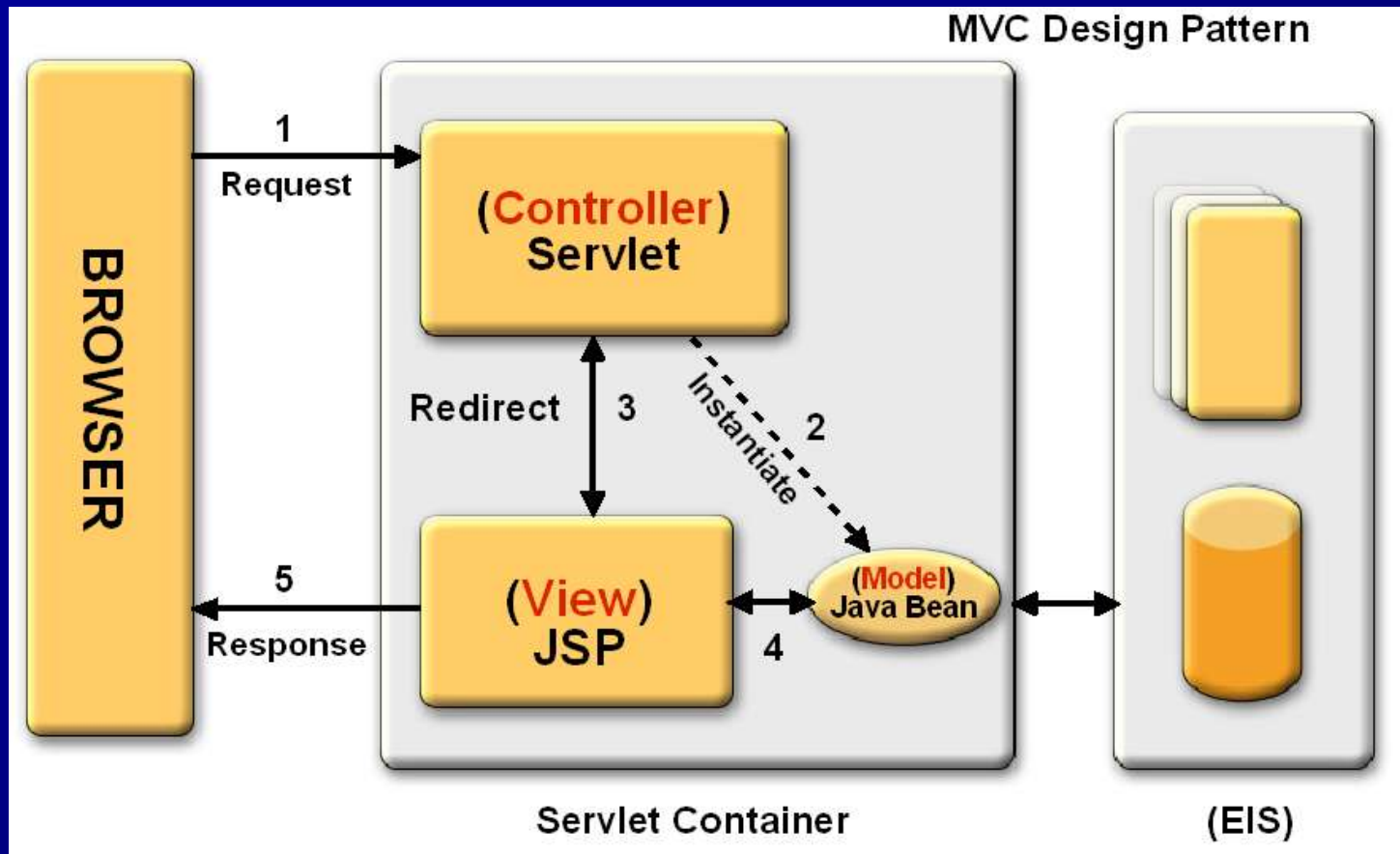
Architecture à base de jsp (modèle 1)



Example



Architecture de modèle 2



Résumé

- J2EE est une plate-forme de choix pour le développement et le déploiement d'applications d'entreprise web multiniveaux basées sur des composants
- J2EE est une architecture standard
- J2EE évolue en accord avec les besoins des entreprises

Présentation de la technologie servlet avec J2EE

Pierre Lefebvre

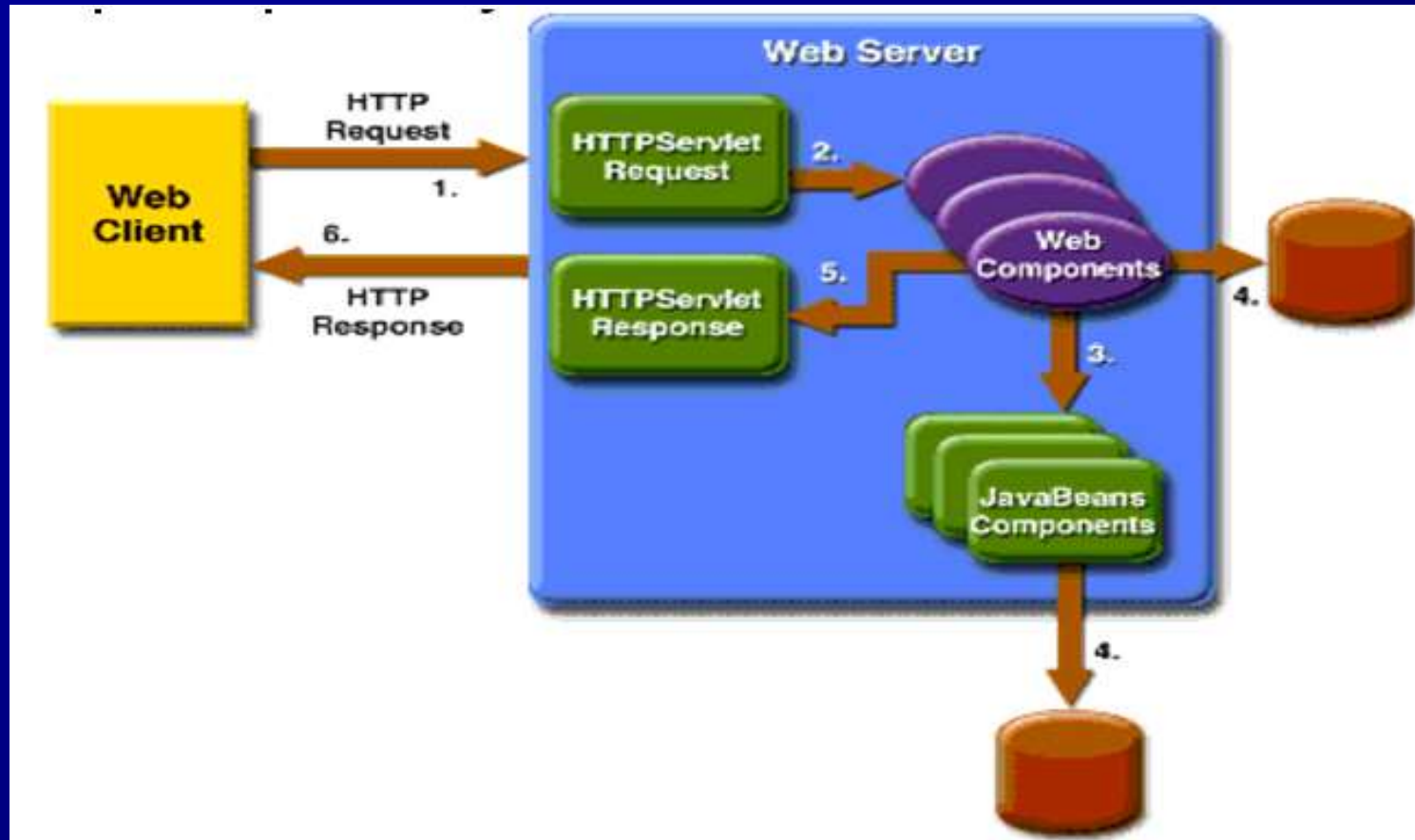
Qu'est-ce qu'une Servlet ?

- Des objets Java qui sont basés sur le framework et les API servlet et qui étendent les fonctionnalités d'un serveur HTTP.
- Mappée sur des URLs et managée par le conteneur
- Disponible sur les serveurs d'applications (indépendance)

Qu'est-ce que fait une servlet ?

- Reçoit la requête d'un client (sous la forme d'une requête HTTP)
- Extrait l'information de la requête
- Prépare la génération du contenu ou exécute la logique métier (accès à une base de données, invocation d'EJBs, etc)
- Crée et envoie la réponse au client (souvent sous la forme d'une réponse HTTP) ou redirige la requête à une autre servlet ou page JSP

Modèle de requête web



Première servlet

```
Public class HelloServlet extends HttpServlet {  
  
    public void doGet(HttpServletRequest request,  
HttpServletResponse response) {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<title>Hello World!</title>");  
    }  
    ...  
}
```

Récupérer les paramètres envoyés par le client

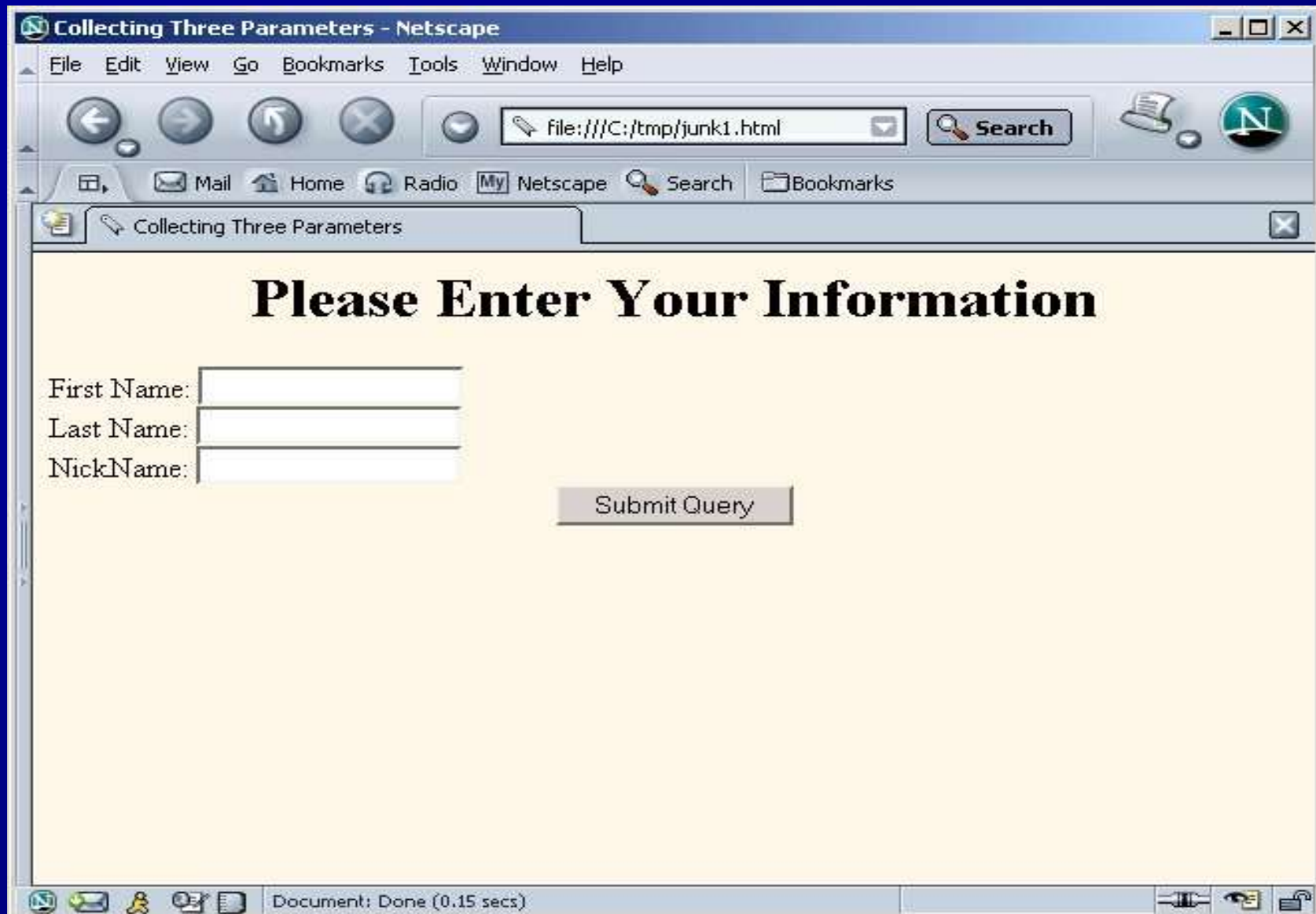
- Une requête peut arriver avec un certain nombre de paramètres
- Les paramètres sont envoyés à partir des formulaires HTML :
 - GET : comme une chaîne de requête, ajouté à l'URL
 - POST : comme des données encodées dans la section du POST, n'apparaissant pas dans l'URL
- `getParameter("paraName")`
 - Renvoie la valeur de `paraName`
 - Renvoie `null` si ce paramètre n'est pas présent

Un formulaire simple utilisant GET

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Collecting Three Parameters</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Please Enter Your Information</H1>

<FORM ACTION="/sample/servlet/ThreeParams">
  First Name:  <INPUT TYPE="TEXT" NAME="param1"><BR>
  Last Name:   <INPUT TYPE="TEXT" NAME="param2"><BR>
  Class Name:  <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT">
  </CENTER>
</FORM>

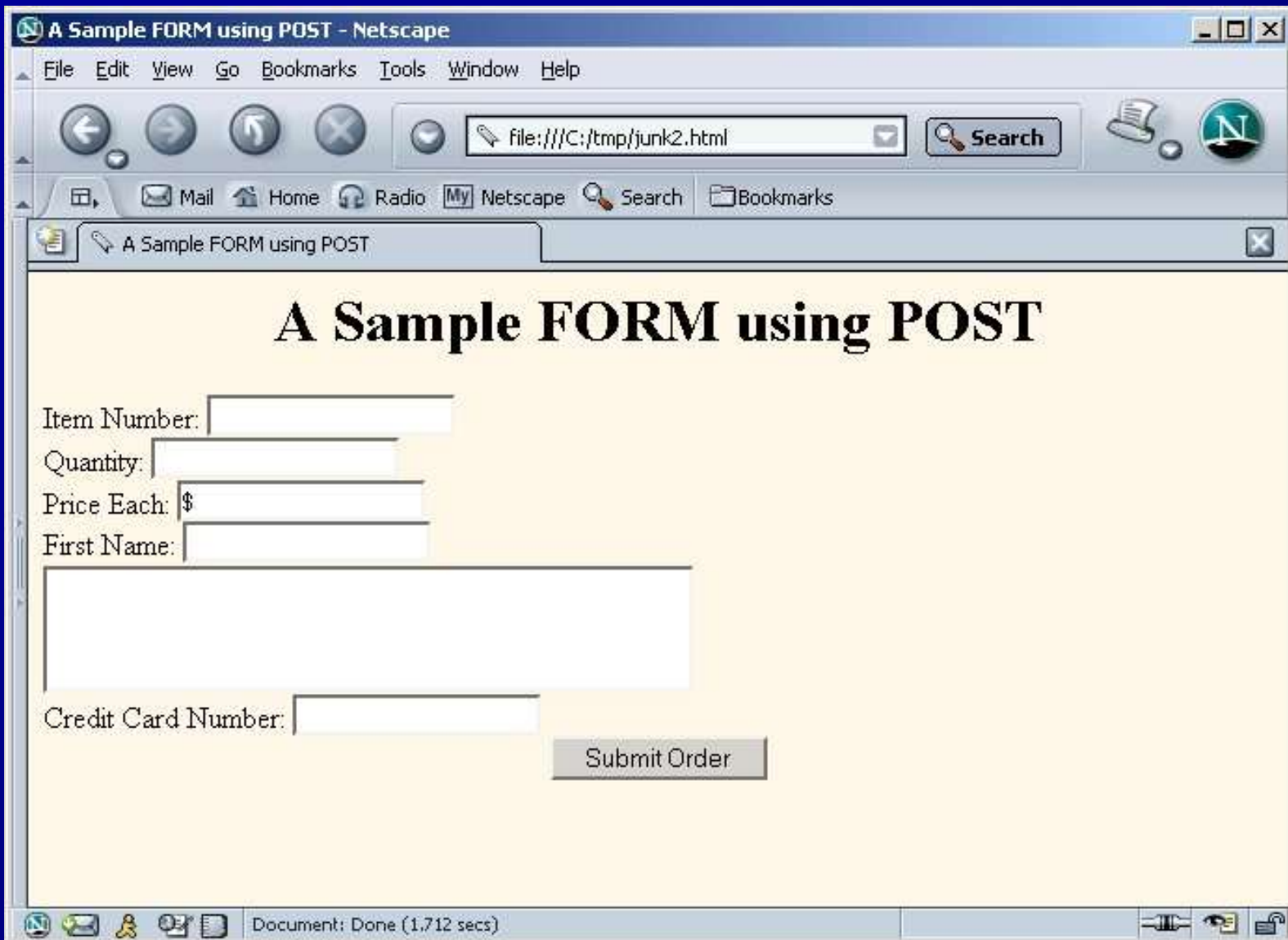
</BODY>
</HTML>
```



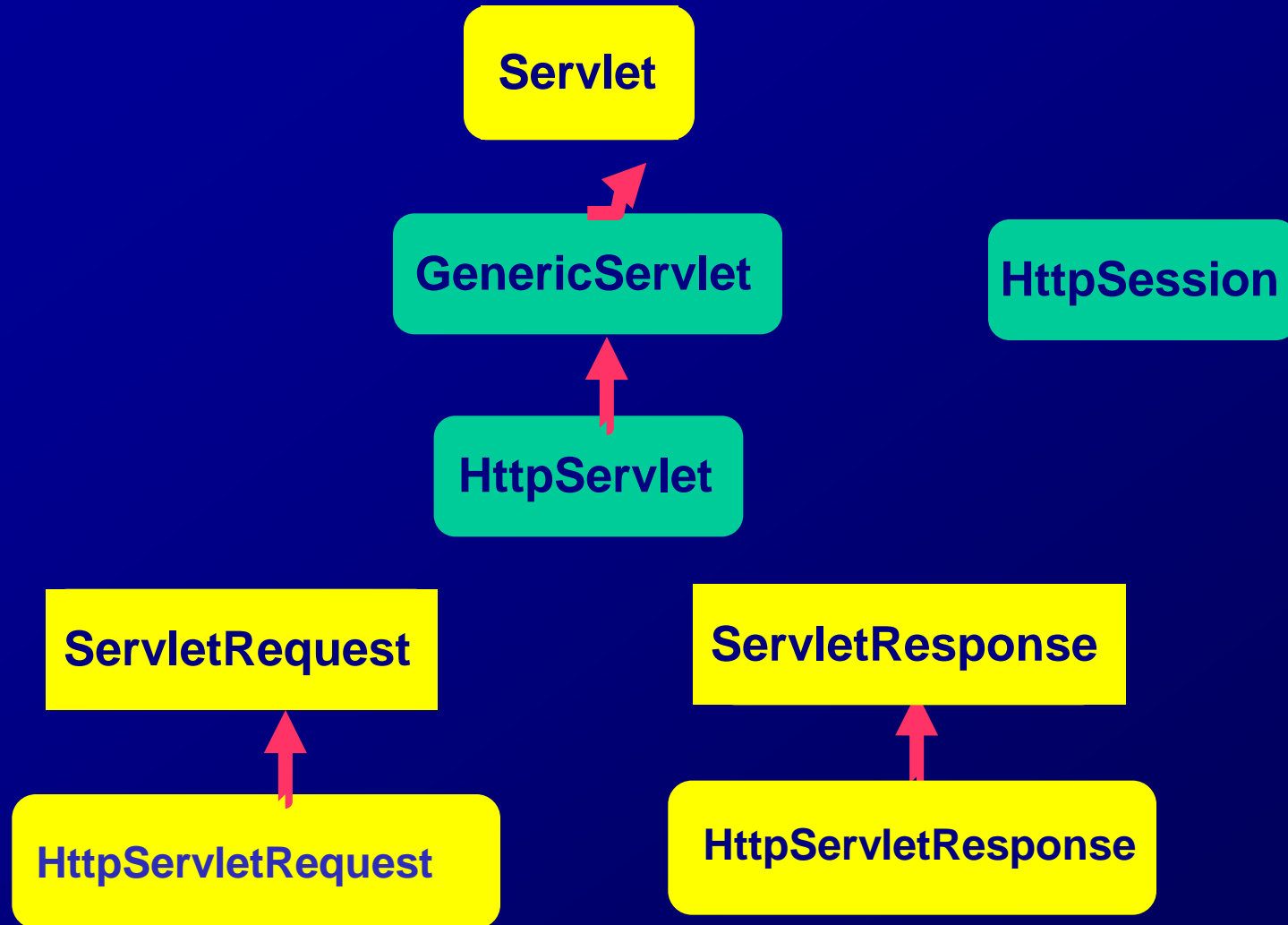
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
/** Simple servlet that reads three parameters from the html form */
public class ThreeParams extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Your Information";
        out.println("<HTML>" +
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
            "<UL>\n" +
            "  <LI><B>First Name in Response</B>: "
            + request.getParameter("param1") + "\n" +
            "  <LI><B>Last Name in Response</B>: "
            + request.getParameter("param2") + "\n" +
            "  <LI><B>NickName in Response</B>: "
            + request.getParameter("param3") + "\n" +
            "</UL>\n" +
            "</BODY></HTML>");
    }
}
```

Un formulaire simple basé sur POST

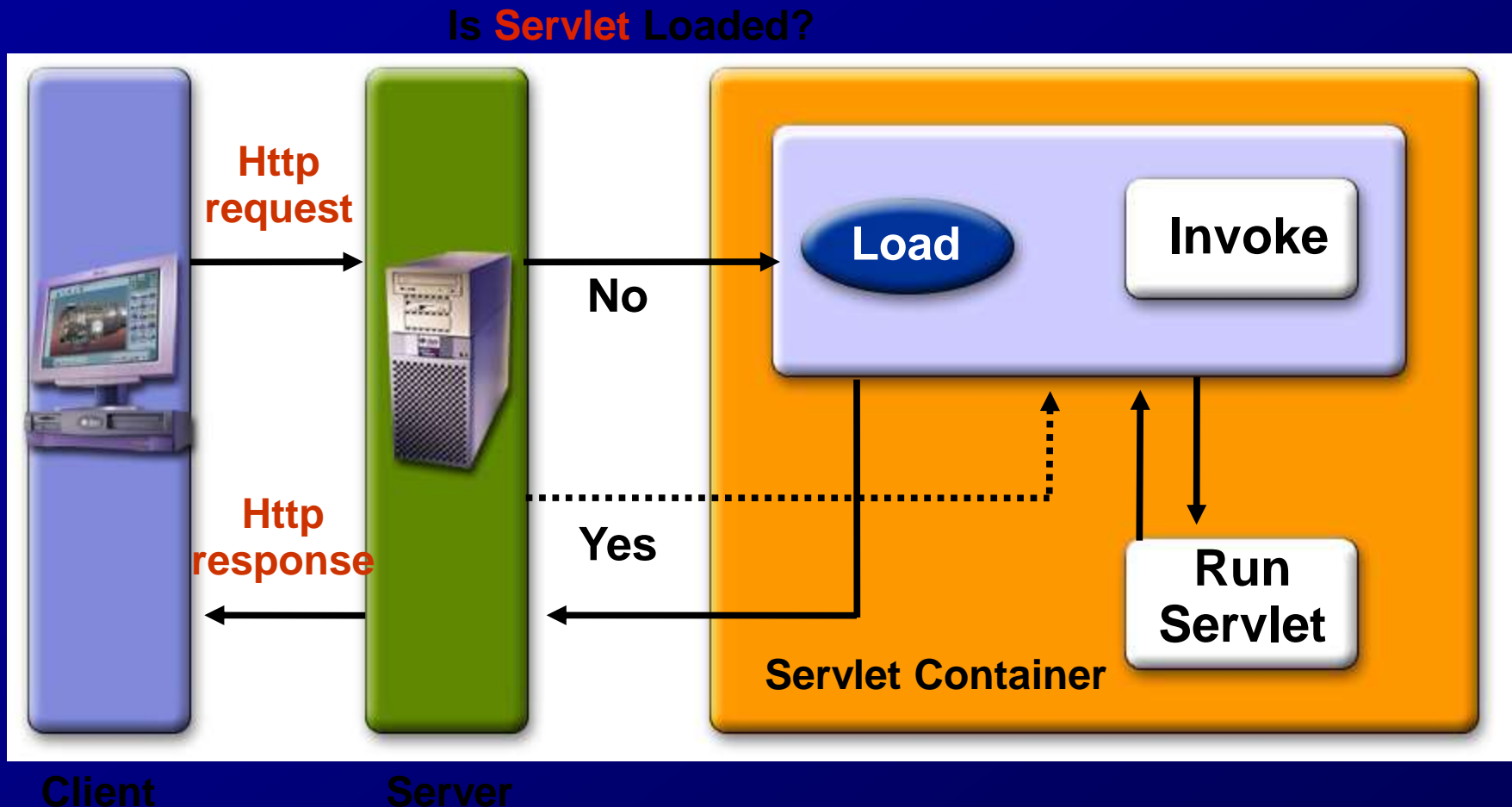
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>A Sample FORM using POST</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">A Sample FORM using POST</H1>
<FORM ACTION="/sample/servlet/ShowParameters" METHOD="POST">
  Item Number: <INPUT TYPE="TEXT" NAME="itemNum"><BR>
  Quantity: <INPUT TYPE="TEXT" NAME="quantity"><BR>
  Price Each: <INPUT TYPE="TEXT" NAME="price" VALUE="$"><BR>
  First Name: <INPUT TYPE="TEXT" NAME="firstName"><BR>
  <TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
  Credit Card Number:
  <INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT" VALUE="Submit Order">
  </CENTER>
</FORM>
</BODY>
</HTML>
```



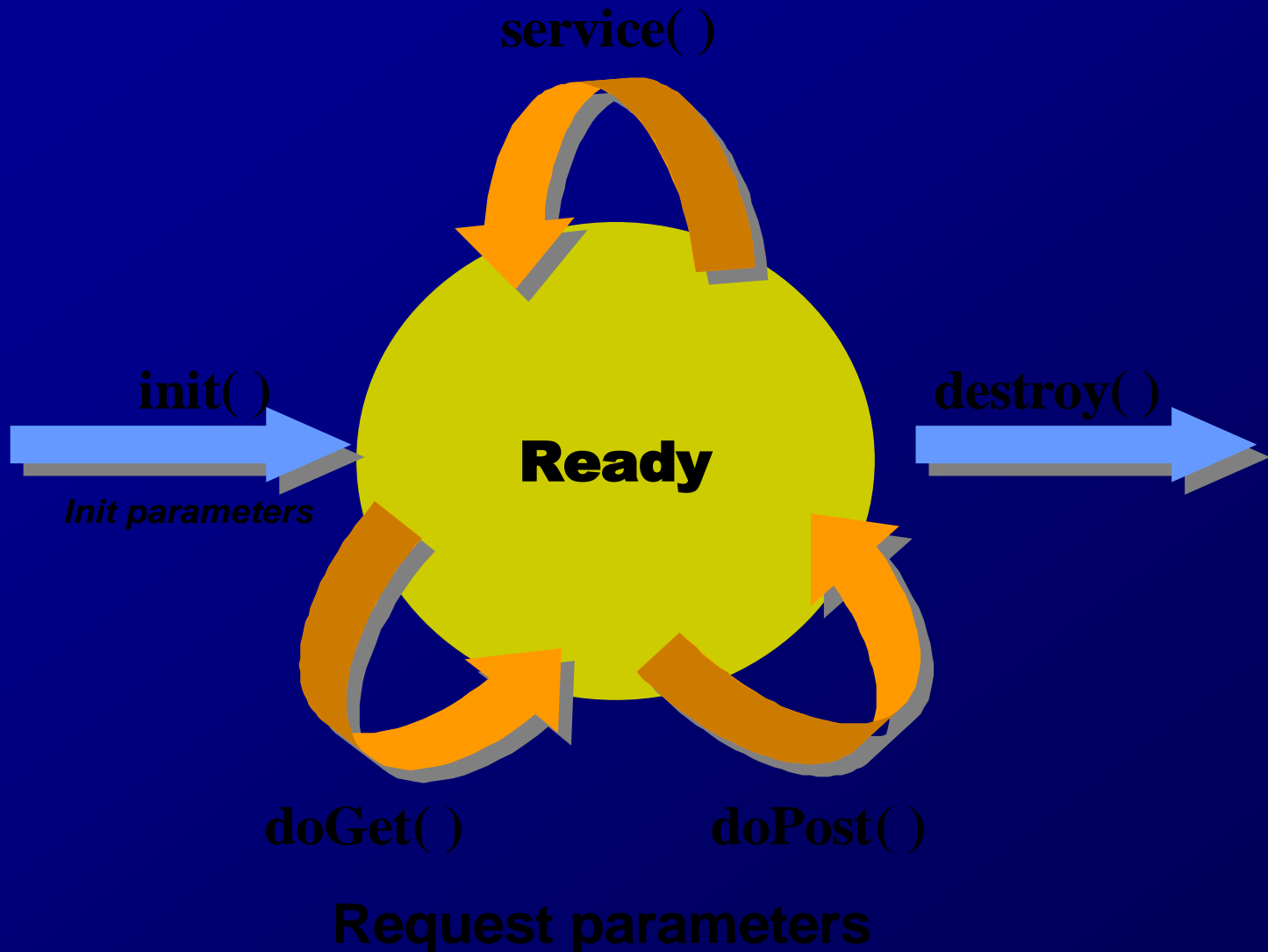
Servlet Interfaces & Classes



Le cycle de vie d'une servlet



Les méthodes du cycle de vie d'une servlet



La méthode *init*

```
public class CatalogServlet extends HttpServlet {  
    private BookDB bookDB;  
  
    // Perform any one-time operation for the servlet,  
    // like getting database connection object.  
  
    // Note: In this example, database connection object is assumed  
    // to be created via other means (via life cycle event mechanism)  
    // and saved in ServletContext object. This is to share a same  
    // database connection object among multiple servlets.  
    public void init() throws ServletException {  
        bookDB = (BookDB)getServletContext().  
            getAttribute("bookDB");  
        if (bookDB == null) throw new  
            UnavailableException("Couldn't get database.");  
    }  
    ...  
}
```

Lire les paramètres de configuration

```
public void init(ServletConfig config) throws
    ServletException {
    super.init(config);
    String driver = getInitParameter("driver");
    String fURL = getInitParameter("url");
    try {
        openDBConnection(driver, fURL);
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

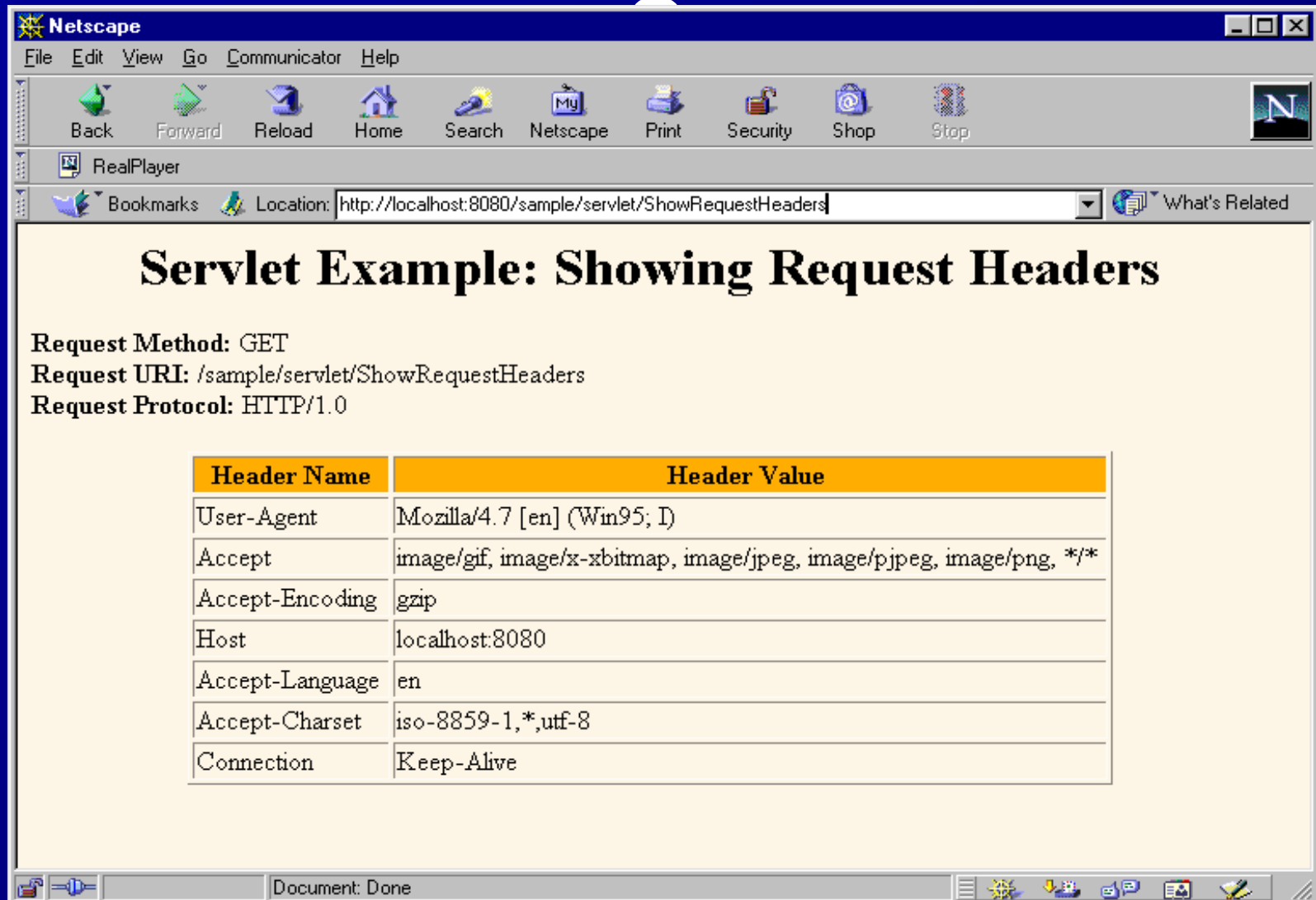
Les paramètres dans web.xml

```
<web-app>
  <servlet>
    <servlet-name>chart</servlet-name>
    <servlet-class>ChartServlet</servlet-class>
    <init-param>
      <param-name>driver</param-name>
      <param-value>
        COM.cloudscape.core.RmiJdbcDriver
      </param-value>
    </init-param>

    <init-param>
      <param-name>url</param-name>
      <param-value>
        jdbc:cloudscape:rmi:CloudscapeDB
      </param-value>
    </init-param>
  </servlet>
</web-app>
```

Les entêtes d'une requête HTTP

```
//Shows all the request headers sent on this particular request.
public class ShowRequestHeaders extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Servlet Example: Showing Request Headers";
        out.println("<HTML>" + ...
            "<B>Request Method: </B>" +
            request.getMethod() + "<BR>\n" +
            "<B>Request URI: </B>" +
            request.getRequestURI() + "<BR>\n" +
            "<B>Request Protocol: </B>" +
            request.getProtocol() + "<BR><BR>\n" +
            ...
            "<TH>Header Name<TH>Header Value");
        Enumeration headerNames = request.getHeaderNames();
        while(headerNames.hasMoreElements()) {
            String headerName = (String)headerNames.nextElement();
            out.println("<TR><TD>" + headerName);
            out.println("    <TD>" + request.getHeader(headerName));
        }
        ...
    }
}
```



La portée des objets

- Permettre le partage d'information parmi des composants web collaborant via des attributs maintenus dans une portée
 - Les attributs sont de type paire nom/objet
- Les attributs maintenus dans une portée sont accédés à l'aide de :
 - `getAttribute()` & `setAttribute()`
- 4 portées sont définies
 - context, session, requête, page

Récupérer la valeur d'un attribut à partir de ServletContext

```
public class CatalogServlet extends HttpServlet {  
    private BookDB bookDB;  
    public void init() throws ServletException {  
        // Get context-wide attribute value from  
        // ServletContext object  
        bookDB = (BookDB) getServletContext().  
            getAttribute("bookDB");  
        if (bookDB == null) throw new  
            UnavailableException("Couldn't get database.");  
    }  
}
```

Pourquoi HttpSession ?

- Besoin d'un mécanisme pour **maintenir l'état du client** à travers un ensemble de requêtes du même utilisateur (ou provenant du même navigateur) pendant une certaine durée de temps
 - Exemple: panier de commerce électronique
- HttpSession maintient l'état du client
 - Utilisé par des Servlets pour positionner ou récupérer des valeurs dans la portée session

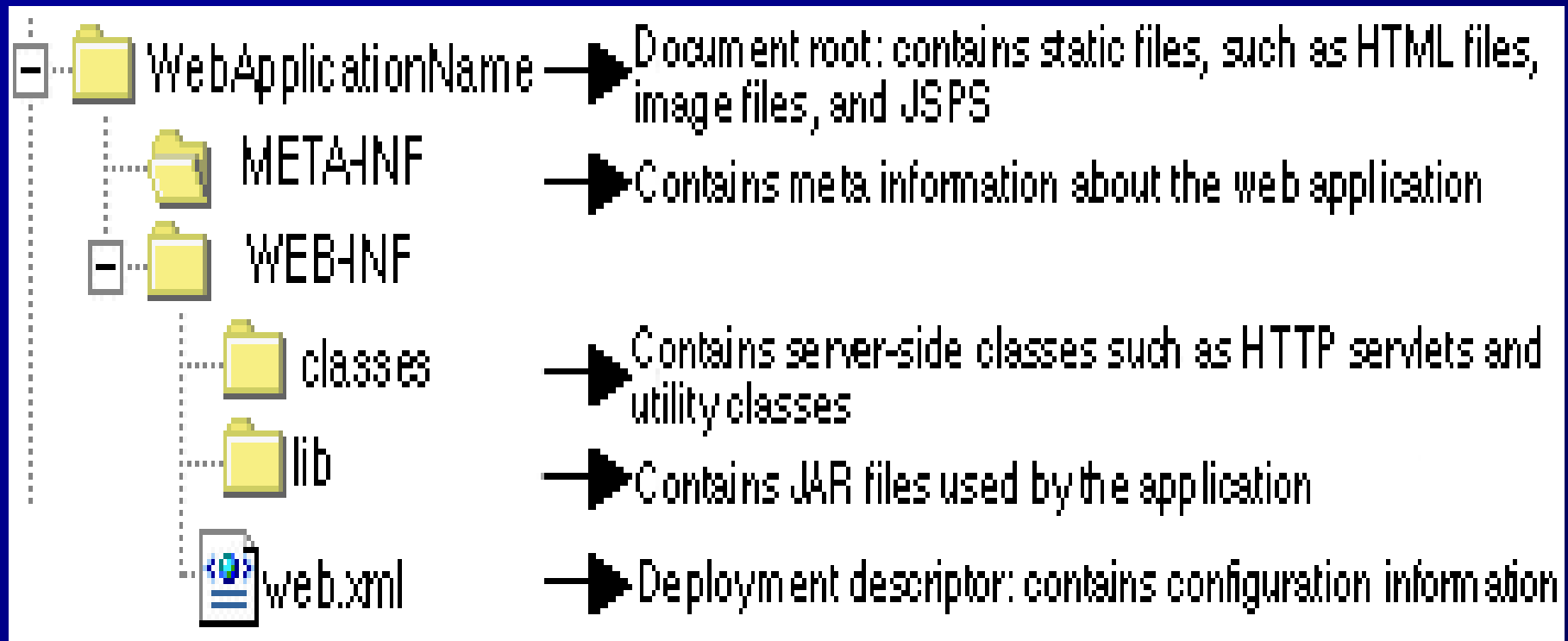
Exemple

```
public class CashierServlet extends HttpServlet {  
    public void doGet (HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
  
        // Get the user's session and shopping cart  
        HttpSession session = request.getSession();  
        ShoppingCart cart =  
            (ShoppingCart) session.getAttribute("cart");  
  
        ...  
        // Determine the total price of the user's books  
        double total = cart.getTotal();  
    }  
}
```

URL d'une requête HTTP


- `http://[host]:[port]/[request path]?[query string]`
- `[request path]` est fait de :
 - Context: `/<context de web app>`
 - Servlet name: `/<component alias>`
 - Path information: le reste
- Examples
 - `http://localhost:8080/hello1/greeting`
 - `http://localhost:8080/hello1/greeting.jsp`
 - `http://daydreamer/catalog/lawn/index.html`

Organisation d'un projet web



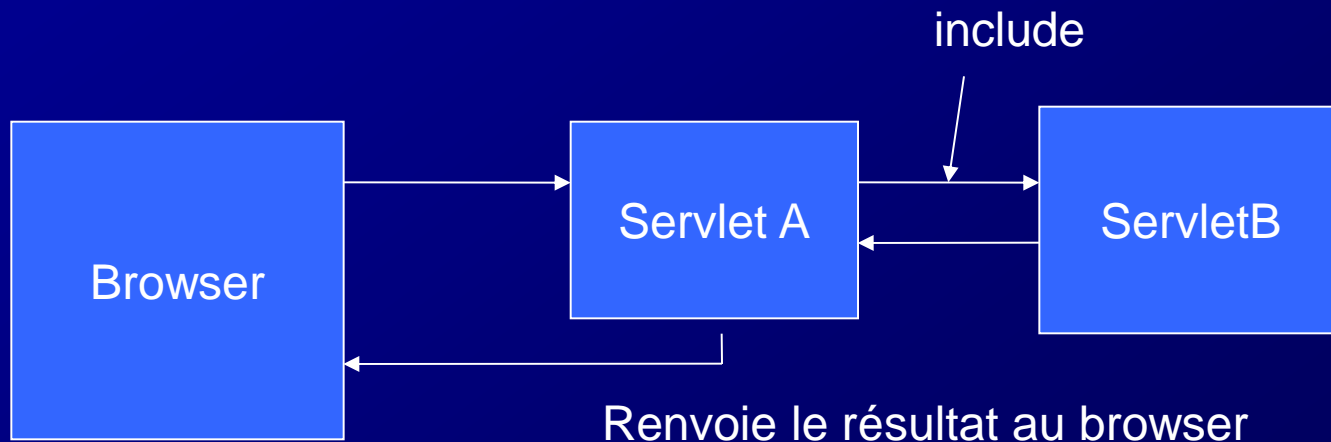
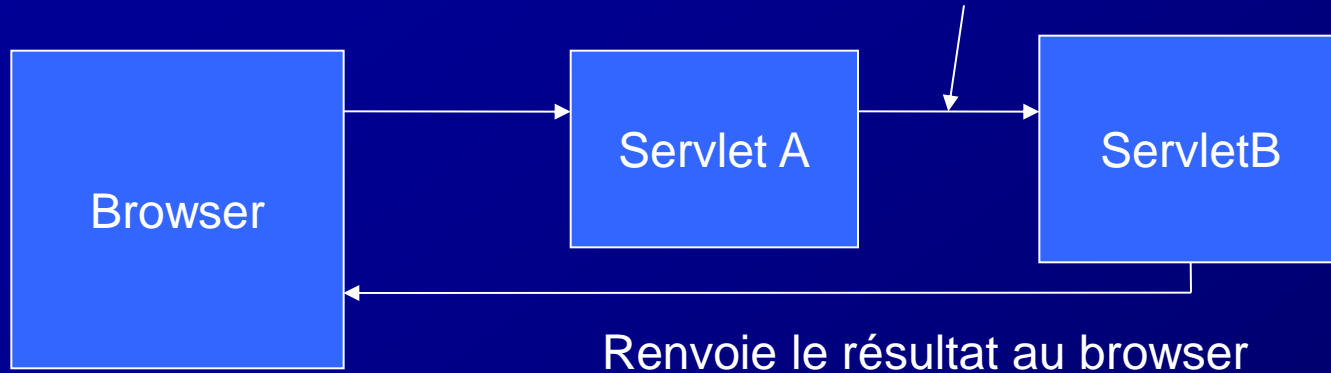
Alias Paths (de web.xml)

```
<servlet>
  <servlet-name>greeting</servlet-name>
  <display-name>greeting</display-name>
  <description>no description</description>
  <servlet-class>GreetingServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>response</servlet-name>
  <display-name>response</display-name>
  <description>no description</description>
  <servlet-class>ResponseServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>greeting</servlet-name>
  <url-pattern>/greeting</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>response</servlet-name>
  <url-pattern>/response</url-pattern>
</servlet-mapping>
```



The diagram consists of two curved arrows on the right side of the code block. The top arrow originates from the `<url-pattern>/greeting</url-pattern>` line in the `<servlet-mapping>` section and points to the `<servlet-class>GreetingServlet</servlet-class>` line in the first `<servlet>` section. The bottom arrow originates from the `<url-pattern>/response</url-pattern>` line in the second `<servlet-mapping>` section and points to the `<servlet-class>ResponseServlet</servlet-class>` line in the second `<servlet>` section.

Le RequestDispatcher



Utiliser un objet RequestDispatcher

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

    HttpSession session = request.getSession(true);
    ResourceBundle messages = (ResourceBundle)session.getAttribute("messages");

    // set headers and buffer size before accessing the Writer
    response.setContentType("text/html");
    response.setBufferSize(8192);
    PrintWriter out = response.getWriter();

    // then write the response
    out.println("<html>" +
               "<head><title>" + messages.getString("TitleBookDescription") +
               "</title></head>");

    // Get the dispatcher; it gets the banner to the user
    RequestDispatcher dispatcher =
        session.getServletContext().getRequestDispatcher("/banner");

    if (dispatcher != null)
        dispatcher.include(request, response);
    ...
}
```


Servlet et JDBC ?

■ JDBC définit des interfaces standards

- Importer le package `java.sql` dans une application java

- Interfaces implémentées par les drivers

Exemple d'interfaces
JDBC

```
interface Driver{...}
```

```
interface Connection{...}
```

```
interface Statement{...}
```

```
interface ResultSet{...}
```

Driver JDBC, comme Oracle

```
class AAA
```

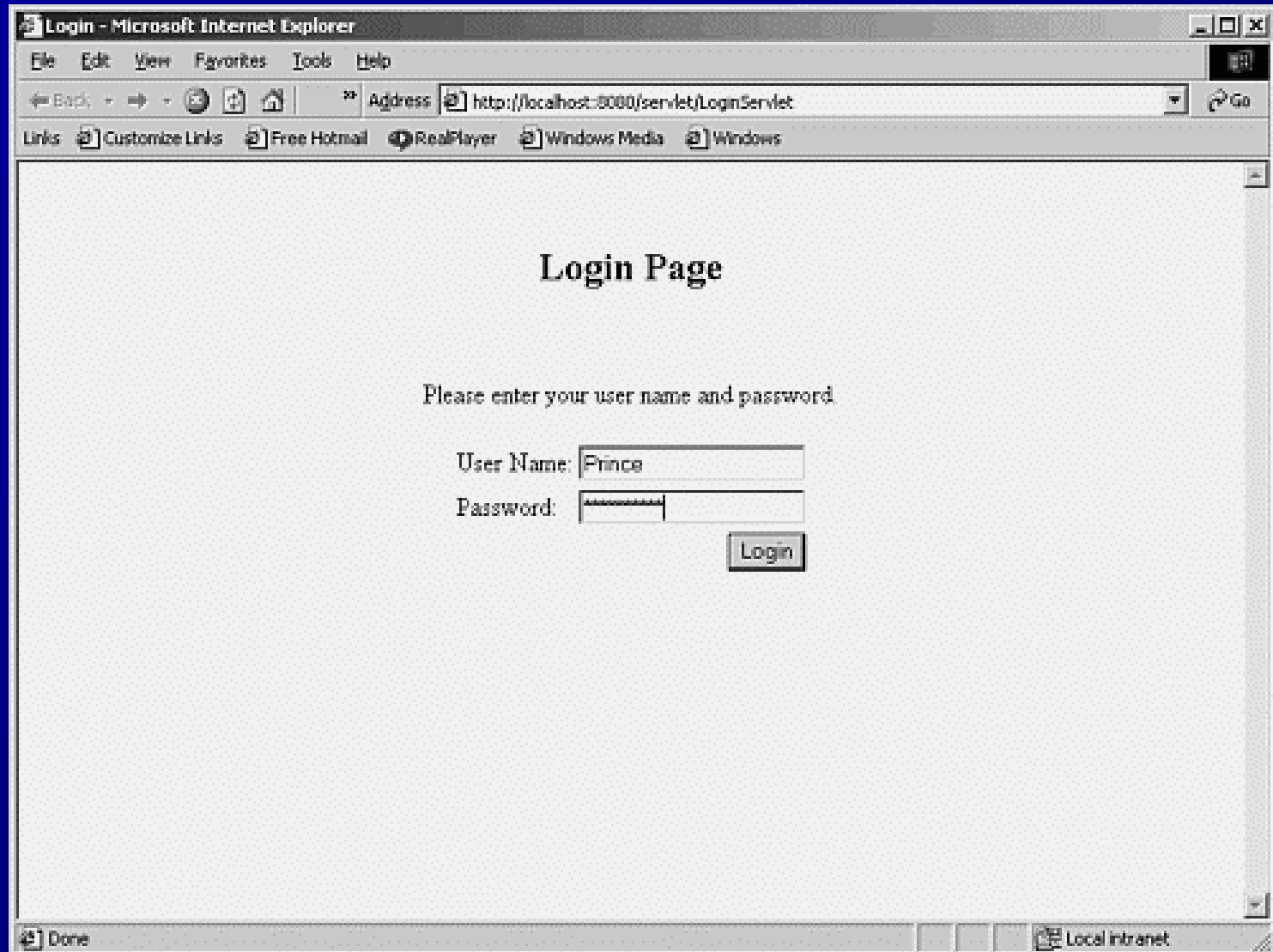
```
    implements Driver{...}
```

```
class BBB
```

```
    implements Connection{...}
```

```
etc...
```

Servlet et JDBC



Servlet login via une base de donnée

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

public class LoginServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        sendLoginForm(response, false);
    }

    private void sendLoginForm(HttpServletResponse response,
        boolean withErrorMessage)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Login</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<CENTER>");
```

```

if (withErrorMessage)
    out.println("Login failed. Please try again.<BR>");

    out.println("<BR>");
    out.println("<BR><H2>Login Page</H2>");
    out.println("<BR>");
    out.println("<BR>Please enter your user name and
password.");
    out.println("<BR>");
    out.println("<BR><FORM METHOD=POST>");
    out.println("<TABLE>");
    out.println("<TR>");
    out.println("<TD>User Name:</TD>");
    out.println("<TD><INPUT TYPE=TEXT NAME=userName></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD>Password:</TD>");
    out.println("<TD><INPUT TYPE=PASSWORD
NAME=password></TD>");
    out.println("</TR>");
    out.println("<TR>");
    out.println("<TD ALIGN=RIGHT COLSPAN=2>");
    out.println("<INPUT TYPE=SUBMIT VALUE=Login></TD>");
    out.println("</TR>");
    out.println("</TABLE>");
    out.println("</FORM>");
    out.println("</CENTER>");
    out.println("</BODY>");
    out.println("</HTML>");
}

```

```

public void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    String userName = request.getParameter("userName");
    String password = request.getParameter("password");
    if (login(userName, password)) {
        RequestDispatcher rd =
            request.getRequestDispatcher("AnotherServlet");
        rd.forward(request, response);
    }
    else {
        sendLoginForm(response, true);
    }
}

boolean login(String userName, String password) {
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con =
            DriverManager.getConnection("jdbc:odbc:JavaWeb");
        System.out.println("got connection");

        Statement s = con.createStatement();
        String sql = "SELECT UserName FROM Users" +
            " WHERE UserName='" + userName + "'" +
            " AND Password='" + password + "'";
        ResultSet rs = s.executeQuery(sql);
    }
}

```

```
        if (rs.next()) {
            rs.close();
            s.close();
            con.close();
            return true;
        }
        rs.close();
        s.close();
        con.close();
    }
    catch (ClassNotFoundException e) {
        System.out.println(e.toString());
    }
    catch (SQLException e) {
        System.out.println(e.toString());
    }
    catch (Exception e) {
        System.out.println(e.toString());
    }
    return false;
}
}
```

Requêtes précompilées

■ Si vous avez besoin d'exécuter une requête plusieurs fois, avec des paramètres variables

– Utiliser un objet `PreparedStatement`

```
try {  
    Connection conn = DriverManager.getConnection(...);  
  
    PreparedStatement pstmt =  
        conn.prepareStatement("update EMP set SAL = ?");  
    ...  
} catch (SQLException e) {...}
```

Lier des variables et executer une requête

PreparedStatement

```
try {  
    PreparedStatement pstmt =  
        conn.prepareStatement("update EMP set SAL = ?");  
    ...  
    pstmt.setBigDecimal(1, new BigDecimal(55000));  
    pstmt.executeUpdate();  
  
    pstmt.setBigDecimal(1, new BigDecimal(65000));  
    pstmt.executeUpdate();  
    ...  
} catch (SQLException e) {...}
```


Pour conclure

