Architecture des applications Web sous J2EE

DUT Informatique

Semestre 4

Mourad Ouziri

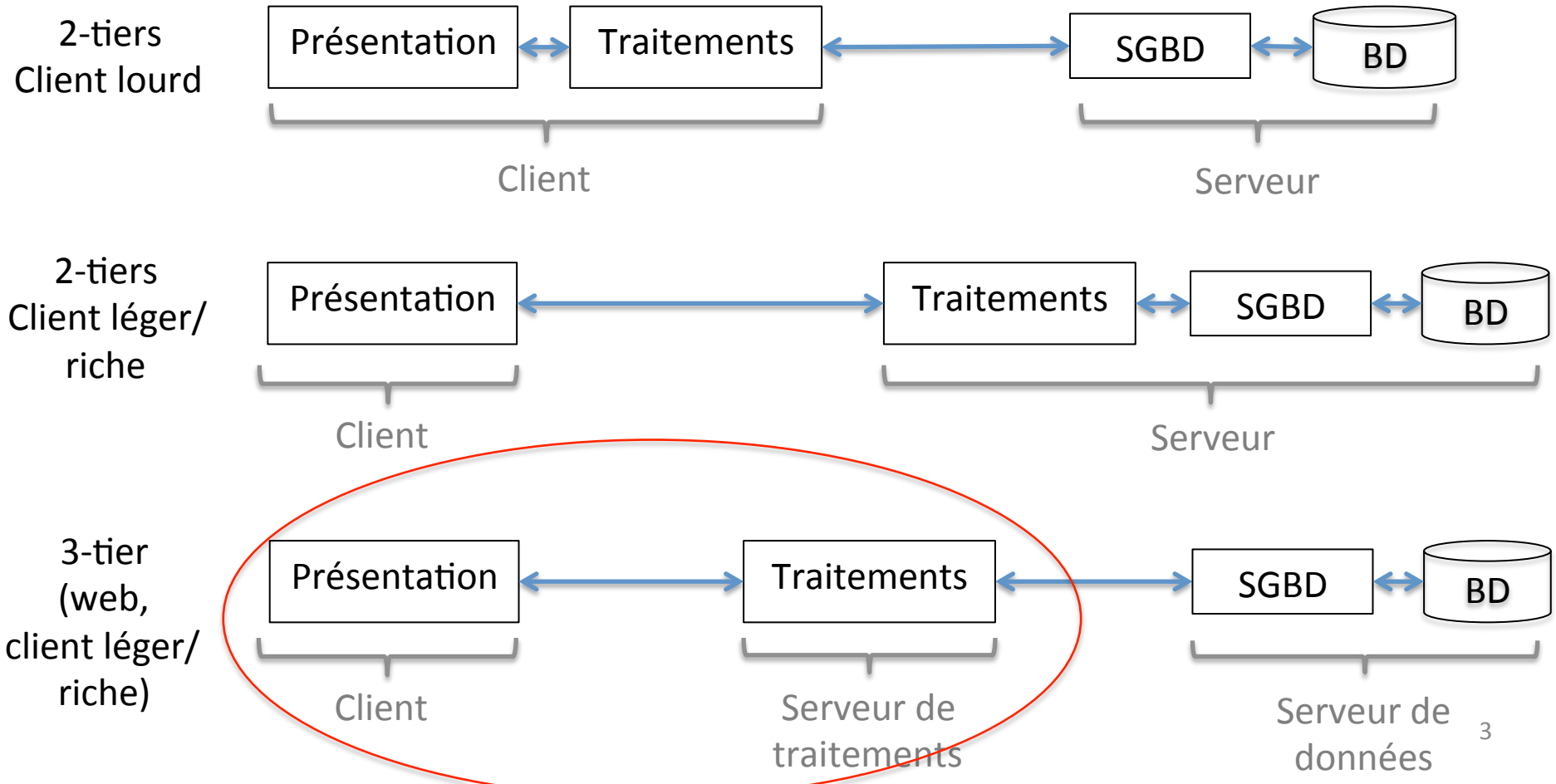
Mourad.Ouziri@parisdescartes.fr

Partie 2

Développement d'applications Web sous J2EE

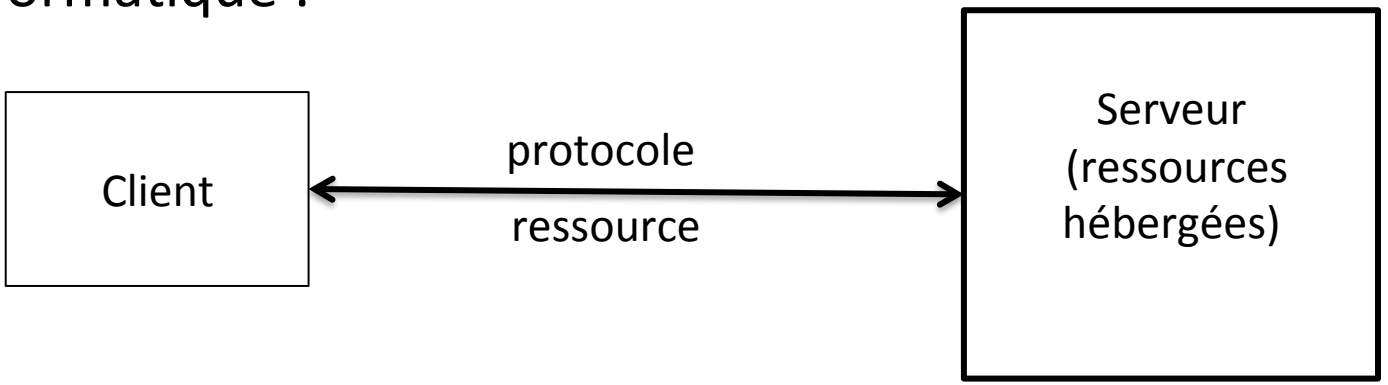
Introduction
rappel

- Architecture : composants, rôles, interactions



Introduction

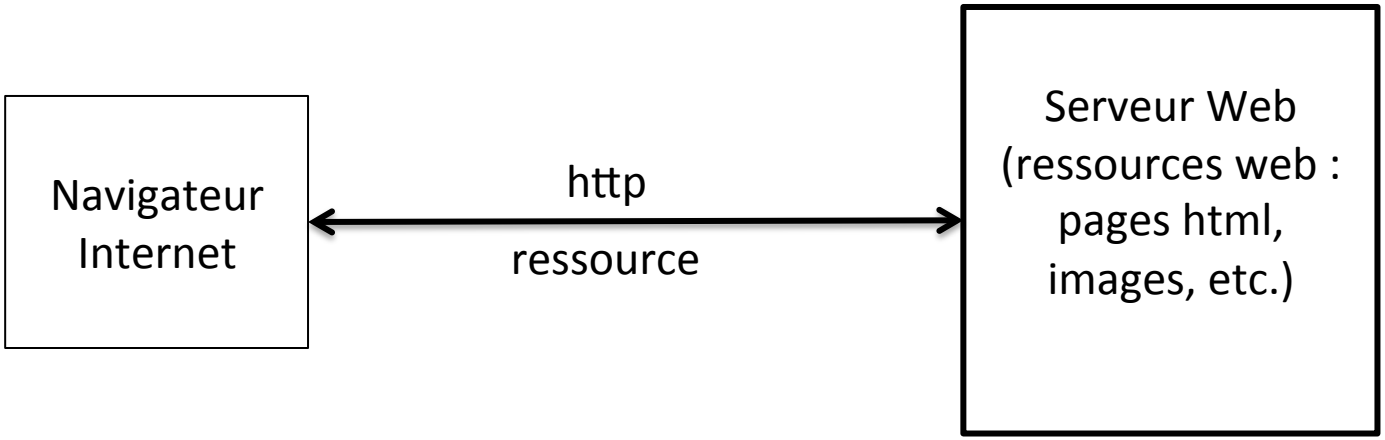
- Serveur informatique :



- Différents types de serveurs :
  - Serveurs de bases de données : protocole JDBC, données
  - Serveurs de fichiers : ftp, fichiers
  - Serveurs d'emails : smtp, emails
  - **Serveur Web : http, ressources web**

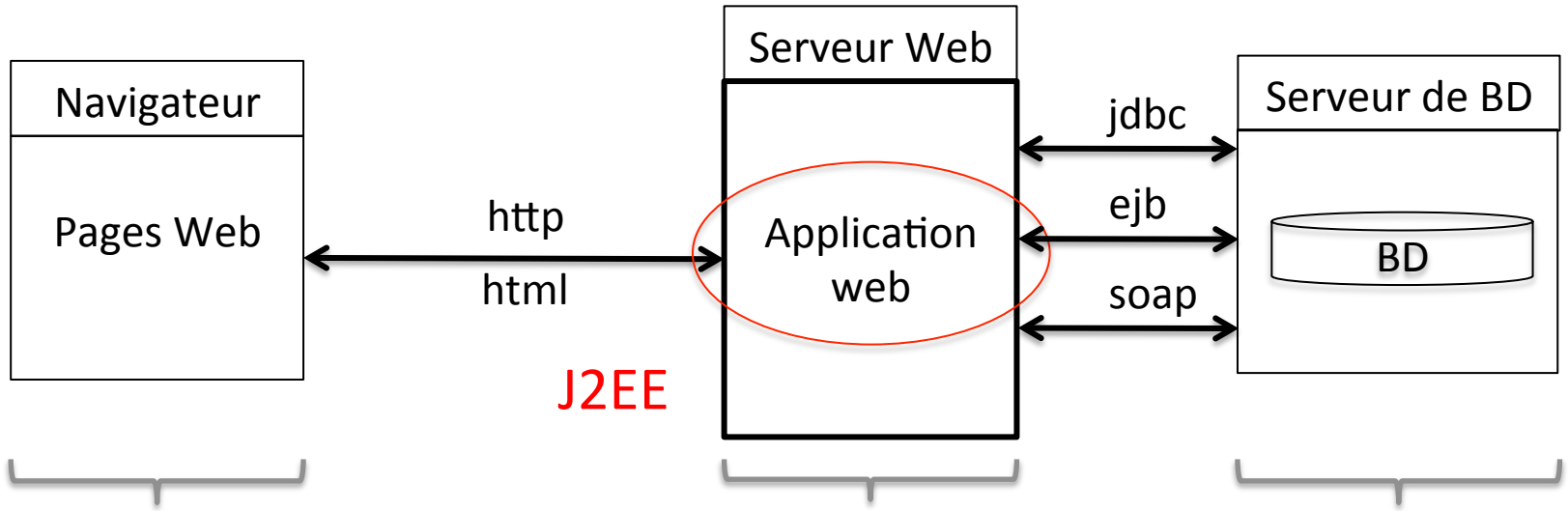
Application Web

- Serveur Web :
  - Répond aux requêtes HTTP
  - Héberge des ressources : pages html, images, scripts, programmes, etc.



Introduction

- Application Web : générer des réponse dynamiquement
  - Application qui sait recevoir, traiter et répondre à des requêtes HTTP
  - Ensemble de services applicatifs accessibles en c/s sur le web

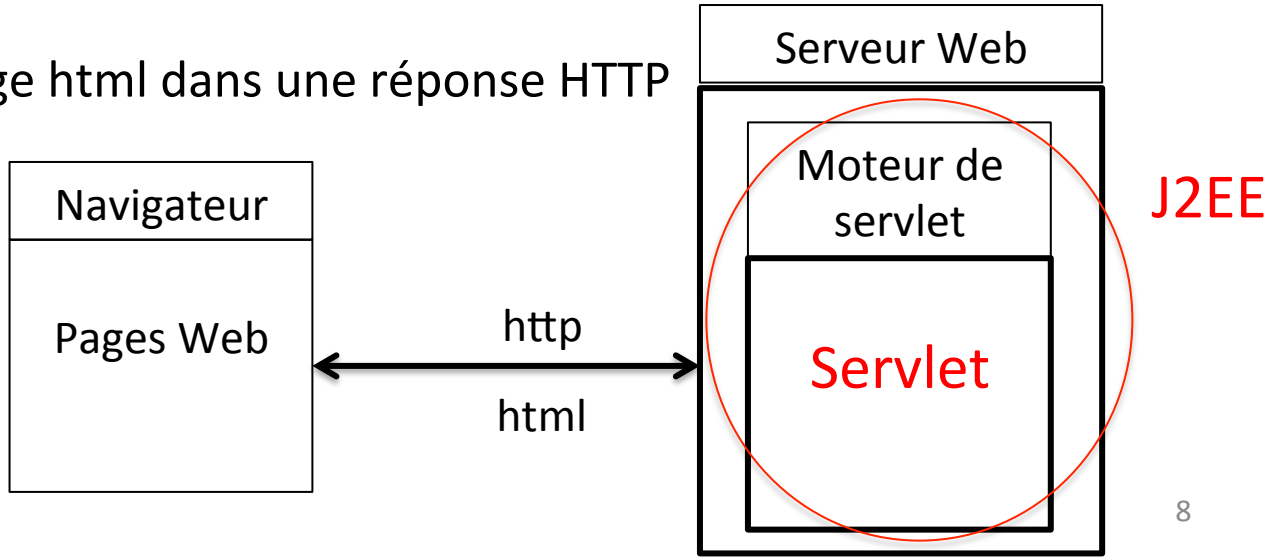


Introduction
J2EE

- JEE : Java Entreprise Edition
  - Plateforme de développement d'applications côté serveur
  - Deux parties : services d'infrastructure de base, API de développement
- Avantages :
  - Simplifier, structurer et assister le développement d'applications d'entreprises
  - Séparer les applications de leur environnement/infrastructure
  - Décharger les développeurs des services de base d'infrastructure

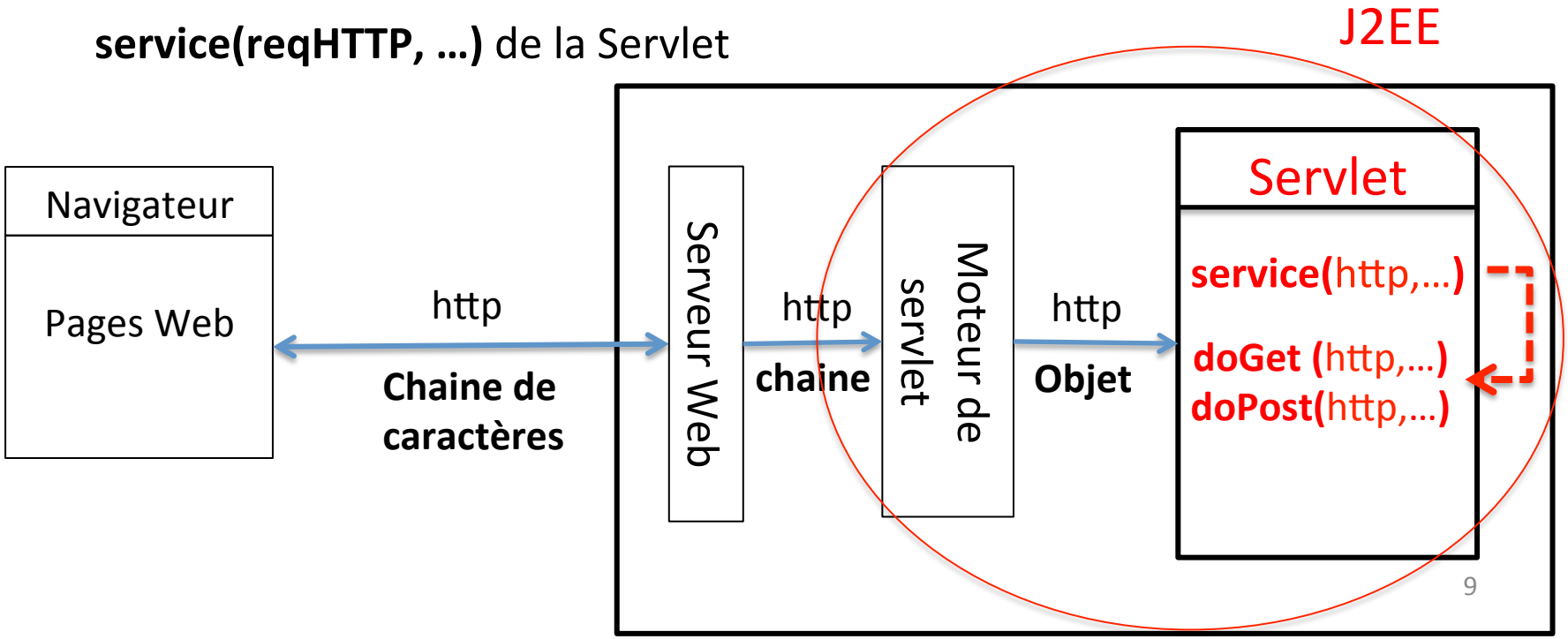
Servlet

- Servlet : API Java dédiée aux protocoles (incluant le HTTP)
  - API (classes abstraites et interfaces) + Environnement d'exécution (serveur web, moteur de servlet)
  - Permet de recevoir des requêtes HTTP et récupérer ses paramètres
  - Générer un résultat au format html (mais aussi texte, xml, json, etc.)
  - Retourner la page html dans une réponse HTTP



Servlet

- Moteur de Servlet : environnement d'exécution de Servlet
  - Offre les services génériques de traitements de requêtes HTTP : réception de requêtes, redirection, sérialisation/désérialisation
  - Transmet les requêtes HTTP à la Servlet : appelle la méthode **service(reqHTTP, …)** de la Servlet

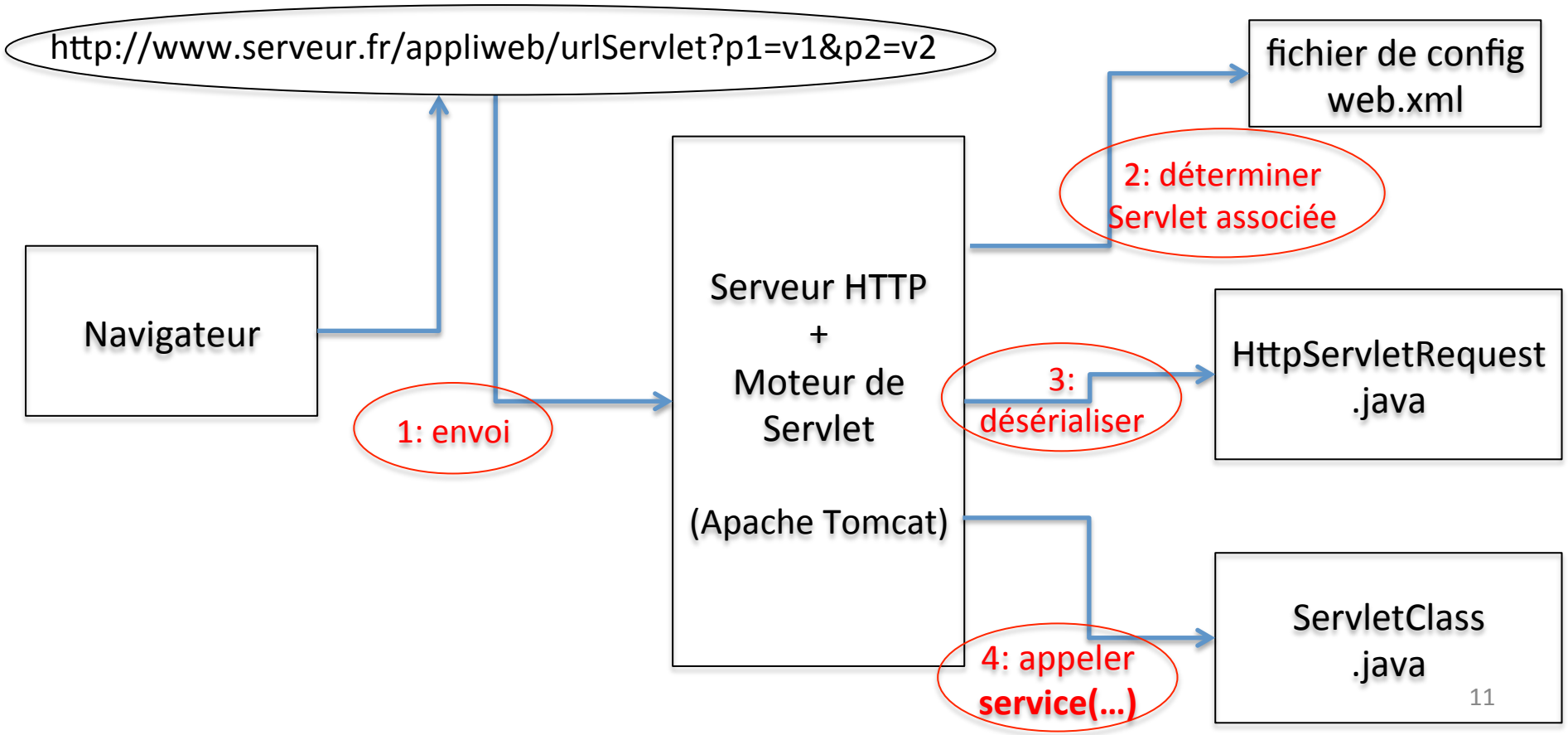


Servlet

- Avantages :
  - Programmation objet de HTTP : puissance de l'Objet, Kit de dév. Java/J2EE
  - Services génériques fournis : moteur de servlet
  - Maintenabilité :
    - Servlet : interfaces et classes abstraites
    - Redéploiement facile : changement de moteur de Servlet
  - Efficacité : un seul chargement (au démarrage!), mono-instance (spécification J2EE, cycle de vie par défaut)

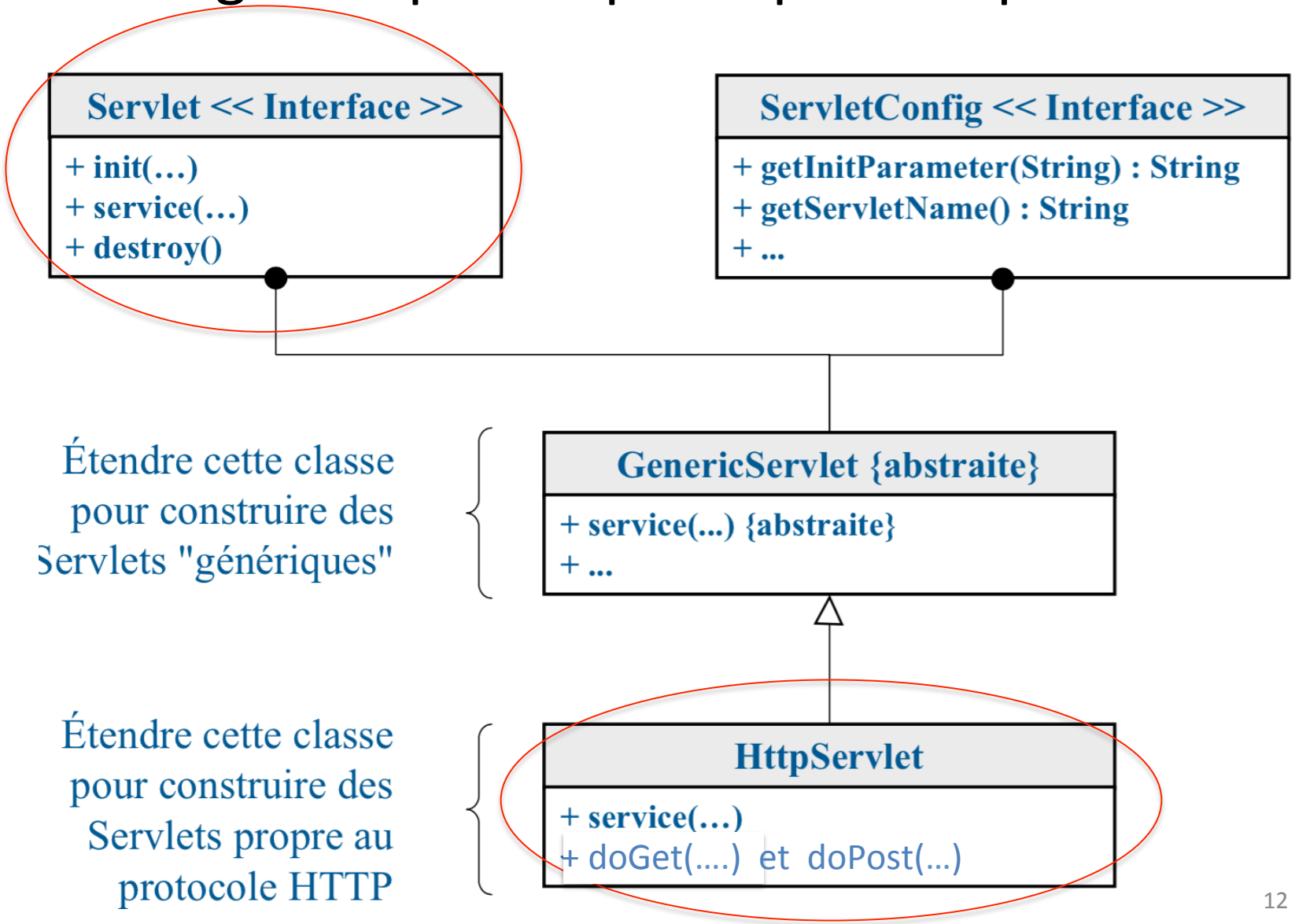
Traitement de requête http
Rôle du moteur de Servlet

- Mise à disposition de requêtes http : moteur de Servlet



Servlet

générique et spécifique à http



Traitement de requêtes HTTP
implémentation

- Une seule classe abstraite à implémenter : ***HttpServlet***
- Méthodes de *HttpServlet* à implémenter:
  - Trois méthodes (et d'autres) de traitement de requêtes http: *service(), doGet(), doPost()*
  - Une méthode d'initialisation exécutée une seule fois au chargement : *init ()*
  - Une méthode de finalisation exécutée une seule fois à la destruction de la Servlet : *destroy()*
- Elle reçoit les requêtes dans ***HttpServletRequest*** :
  - Méthodes d'accès aux attributs de la requête: *getParameter(), getAttribute(), ...*
  - Méthodes de modification des paramètres de la requête: *setParameter(), setAttribute(), …*
  - Méthodes d'accès aux info sur la requête, le client distant et l'environnement du serveur
- Elle génère des résultats (dynamiques) HTML avec ***HttpServletResponse*** :
  - Méthodes d'accès aux attributs de la requête: *getParameter(), getAttribute(), ...*

Traitement de requêtes HTTP
implémentation

- Une seule classe abstraite à implémenter : *HttpServlet*
- Redéfinir ses méthodes *doGet()* et/ou *doPost()* : selon les types de requêtes http pris en compte

```
public class HelloWorld extends HttpServlet {

        public void doGet(HttpServletRequest request, HttpServletResponse response) {
                ...
        }

        public void doPost(HttpServletRequest req, HttpServletResponse res) {
                        …
        }
}
```

Traitement de requêtes HTTP
Descripteur de déploiement

- Définir le path/URL relatif de la Servlet : **web.xml**

```
<html> <body>
  <form action="./url-pattern">
  </form>
</body> </html>
```

http://www.serveur.fr/appliweb/url-pattern?p1=...

```
package servlet;
public class ServletBiblio extends HttpServlet {
        public void doGet (req, resp) {}
```

☞ Fichier de déploiement de l'application Web : *web.xml*

```
<web-app>
    <servlet>
      <servlet-name> myServlet </servlet-name>
      <servlet-class> servlet.ServletBiblio </servlet-class>
    </servlet>
    <servlet-mapping>
      <servlet-name> myServlet </servlet-name>
      <url-pattern>/url-pattern </url-pattern>
    </servlet-mapping>
</web-app>
```

Traitement de requêtes HTTP
Configuration du path Servlet par annotation


```
@WebServlet ("/url-pattern")
public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        ...
    }

    public void doPost(HttpServletRequest req, HttpServletResponse res) {
        ...
    }
}
```

Traitement de requêtes HTTP
implémentation

http://www.serveur.fr/biblio/hello?**nom=ouziri**

```
@WebServlet ("/hello")
public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        String nom = request.getParameter("nom");
        response.setContentType("text/html");          // format du contenu de la réponse
        PrintWriter out = response.getWriter();
        out.println("<html>");                         // générer le résultat dynamique
            out.println("<body>");
                out.println( "Bonjour  " + nom);
            out.println("</body>");
        out.println("</html>");
    }
}
```

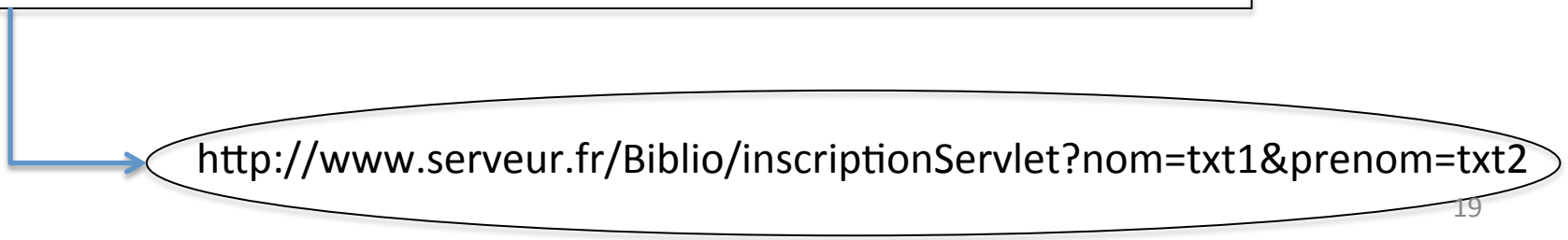
Traitement de requêtes HTTP
implémentation

- Générer le contenu des réponses HTTP : ***HttpServletResponse***
  - *void setContentType(String)* : définit le type de contenu MIME
    - text/plain : contenu texte simple, sans formatage
    - text/html: contenu au format html, balises définies dans le langage
    - application/xml: contenu au format xml, balises et structure définie par l'application
    - application/json: contenu au format JSON, structure définie par l'application
    - Image/jpeg, image/png, video/mpeg, audio_x-wav, etc.
  - *ServletOutputStream getOutputStream()* : flot pour envoyer des données binaires au client
  - *void sendRedirect(String)* : redirige la requête vers une autre URL
  - *void setStatus(int)* : définit le code de retour de la réponse

Traitement de requêtes HTTP
Exemple

Inscription.html

```
<html>
   <body>
      <h1> Formulaire d'inscription  </h1>
      <form method="GET" action="/Biblio/inscriptionServlet">
            Nom : <input type="text" name="nom">
            Prénom : <input type="text" name="prenom"> </p>
            <input type="submit" value="S'inscrire">
      </form>
   </body>
</html>
```

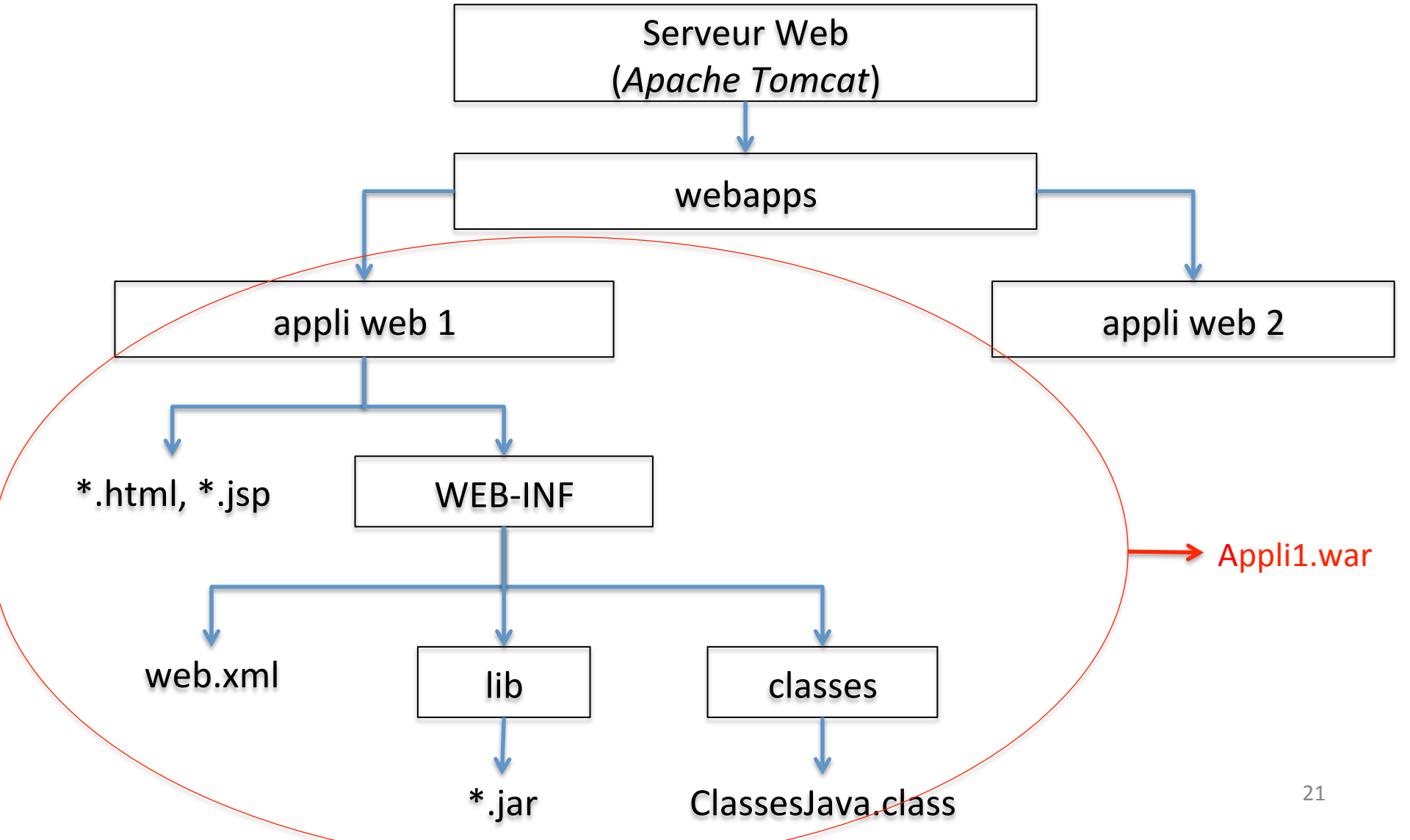


Traitement de requêtes HTTP
Exemple

```
public class BiblioServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        String nom = request.getParameter(“nom”);
        String prenom = request.getParameter(“prenom”);
        boolean inscription = inscrireDansBD (nom, prenom);        // accès JDBC
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
            out.println("<body>");
            if (inscription) {
                out.println( nom + "est inscrit avec succès");
            }
            else {
                out.println("Erreur : inscription échouée! ");
            }
            out.println(”</body>");
        out.println("</html>");
}}
```

Structure d'une application Web J2EE

Organisation sur le disque



Traitement de requêtes HTTP
Cycle de vie de Servlet

- Le cycle de vie d'une Servlet est entièrement géré par le moteur de Servlet

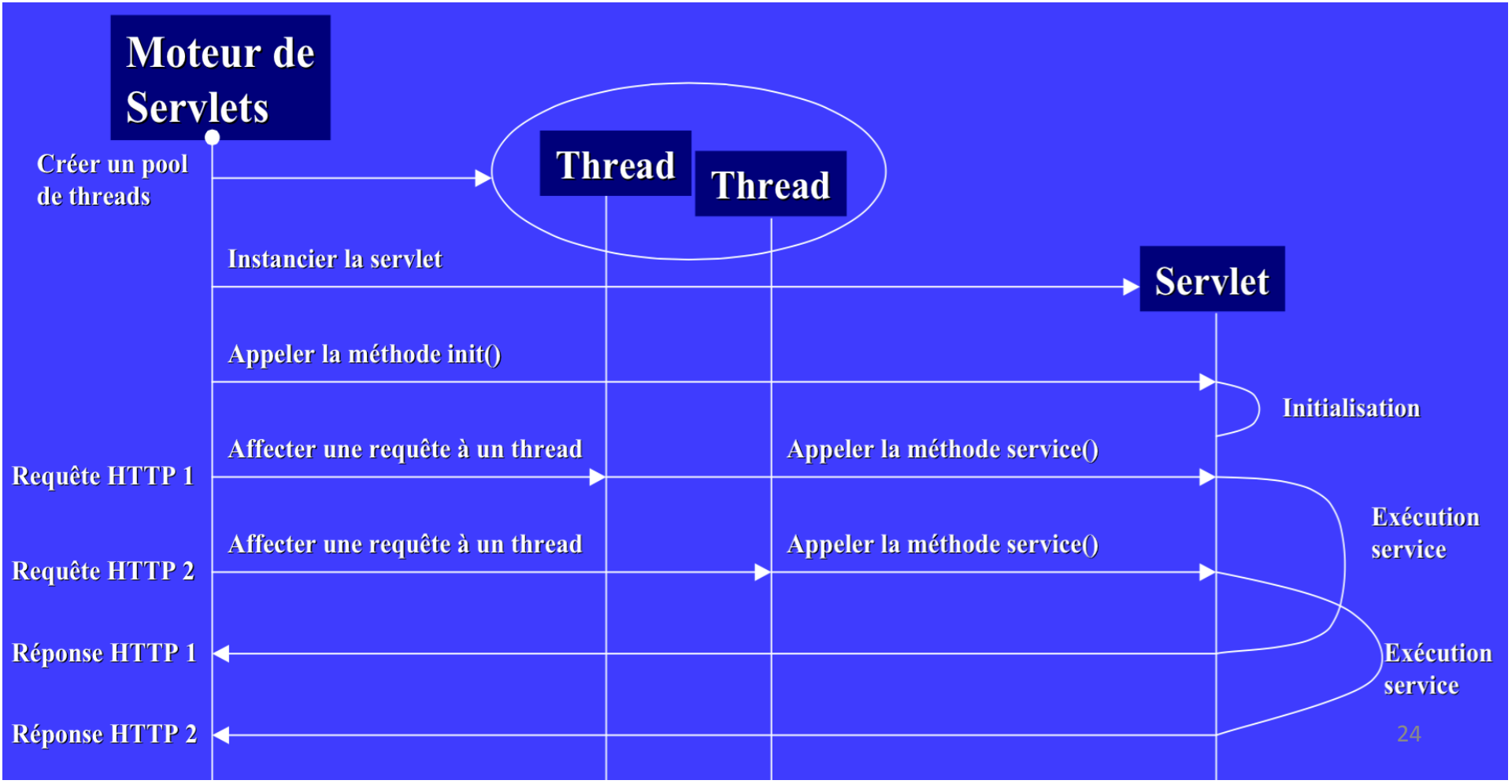
```
public class UneServlet extends HttpServlet {

    public void init(ServletConfig config) {
        appelée une seule fois au chargement de la Servlet
    }

    public void doGet (HttpServletRequest req, …) ou doPost (…) ou service(…) {
        appelée à chaque réception d'une requête http req
    }

    public void destroy () {
        appelée une seule fois à la destruction de l'instance de Servlet
    }
}
```

Servlet : une classe mono-instance

- Servlet : une classe mono-instance
  - Les requêtes sont exécutées par la même instance (objet unique)
  - L'instance permet de garder un état pour plusieurs requêtes HTTP (pb de http)
  - Coût de chargement en mémoire réduit
- Ressource critique à verrouiller
  - Attributs de la classe : partagés par les requêtes !
- Exécuter les requêtes par plusieurs objets : Servlet multi-instances
  - Implémenter l'interface : *SingleThreadModel*

Servlet : une classe mono-instance

- Traitement des requêtes : processus légers parallèles (Thread)



Traitement de requêtes HTTP
Ressources partagées

- Accès concurrents à l'état de la Servlet : ressource *compteur*

```
public class UneServlet extends HttpServlet {

        private int compteur = 0;

        public void doGet (HttpServletRequest req, …) ou doPost (…) ou service(…) {
                PrintWriter out = res.getWriter();
                synchronized (this) {
                        compteur ++;
                        out.println ("Ceci est l'appel numéro: " + compteur );
                }
        }
}
```

Traitement de requêtes HTTP
Ressources partagées

```
public class RecherchePersonneServlet extends HttpServlet {
    private Connexion connexion ;
    PreparedStatement ptsm;
    public void init(ServletConfig config) {
        super.init(config) ;
        this.connexion = GestionBD.creerConnexion ();
        pstm = this.connexion.prepareStatement ("SELECT * FROM Pers WHERE nom=?");
    }
    public void doGet (HttpServletRequest req, …) ou doPost (…) ou service(…) {
        synchronized (this.pstm) {
            pstm.setInt (req.getParameter());
            resultat = pstm.executeQuer();
            out.println ("resultat : " + resultat.getString ("adr") );
        }
    }
    public void destroy () {
        this.pstm.close();          this.connexion.close();
    }
```

Servlet : suivi de sessions

- HTTP : protocole non connecté
  - Pas d'état global !
  - Indépendance des requêtes HTTP
- Servlet : état global à toutes les requêtes HTTP
- Problématique : état propre à chaque utilisateur
  - Comment transmettre les données entre requêtes d'un même utilisateur ?
  - Exemple : identification, panier électronique
- Solutions possibles :
  - Incorporer les données dans les pages de réponses (en paramètres cachés) : lourde à gérer par les développeurs !
  - Cookies : peuvent être désactivés par les navigateurs !
  - Session utilisateur : solution présentée !

Servlet : suivi de sessions

- Une session utilisateur
  - Ensemble de requêtes provenant d'un même utilisateur dans une période de temps limitée
  - Identifier l'utilisateur des requêtes http
  - Espace mémoire (objet) dédié à chaque utilisateur qui sera partagé par ses requêtes http
- Classe **HttpSession**
  - Permet de stocker les données d'une session : données (valeurs numériques, textes, références d'objets) pendant la durée de vie de la session !
  - Un seul objet est créé par session et partagé par les requêtes de la session
- API :
  - Méthodes : setAttribute (String, Object), getAttribute (String, Object)

Servlet : suivi de sessions

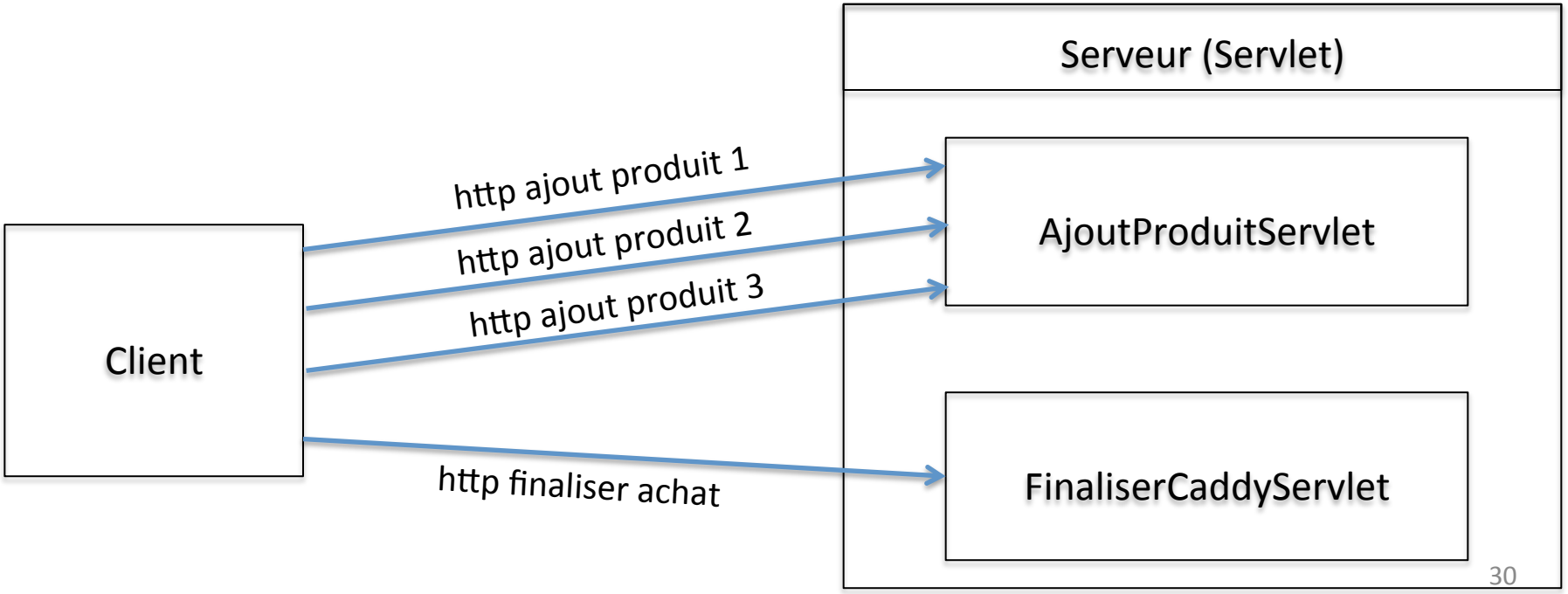
- Créer/obtenir (l'objet) la session de la requête
  - Méthode de HttpServletRequest :

    HttpSession **getSession** (boolean)

  boolean : true pour créer la session si elle n'existait pas encore
- Partager une donnée dans la session
  - Méthode : session.setAttribute ("idDonnée", donnée);
  - Exemple : session.setAttribute ("login", "ouziri");
- Récupérer une donnée partagée dans la session
  - Méthode : session.getAttribute ("idDonnée");
  - Exemple : (String) session.getAttribute ("login");

Servlet : suivi de sessions

- Exemple : caddie électronique
  - Objet propre à chaque utilisateur, utilisé par plusieurs requêtes
  - Plusieurs requêtes HTTP distinctes pour l'ajout de produits : AjoutProduitServlet
  - Requête de finalisation du Caddy : FinaliserCaddyServlet



Servlet : suivi de sessions

- Exemple : Servlet d'ajout de produits dans un Caddy électronique

```
public class AjouterProduitServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse resp) {
        HttpSession session = request.getSession(true);
        Caddy leCaddy = session.getAttribute("caddy");
        if (caddy==null) {
            leCaddy = new Caddy();
            session.setAttribute ("caddy", leCaddy);
        }
        String refProduit = request.getParameter("refProd")) ;
        int quantite = Integer.parseInt(request.getParameter("quantite")) ;
        leCaddy.addProduit(new LigneCommande (refProduit, quantite));
        …
        }
    }
```

Servlet : suivi de sessions

- Exemple : Servlet de finalisation du Caddy

```
public class FinaliserCaddyServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse resp) {
        HttpSession session = request.getSession(true);
        Caddy leCaddy = session.getAttribute("caddy");
        if (caddy!=null) {
            GestionBD.stockerCaddy (leCaddy);
            session.removeAttribute ("caddy");
            …
        }
    }
}
```

Serveur et moteur de Servlet *Tomcat*

- Produit d'Apache:
  - https://tomcat.apache.org/download-90.cgi
- Implémentation de la spécification Servet (et *JSP*) de J2EE
- Administration du serveur :
  - Démarrage/arrêt du serveur : ./bin/startup.bat, shutdown.bat *(.sh)*
  - Utilisateurs : triplet (nom, password, rôle) défini dans : ./conf/*tomcat-users.xml*

    <user name="nom" password="secret" roles="standard,manager" />
- Administration des applications Web :
  - Interface Web : http://serveur:8080/manager/html
  - Accès avec le rôle : manager-gui
  - Déploiement, démarrage, arrêt et rechargement d' applications Web hébergées

Conclusion

- Servlet : traitement de requêtes http côté serveur
- Maintenabilité, performances et sécurité
- Puissance : on peut tout faire avec les Servlet !
  - Ne pas lui attribuer trop de responsabilités : maintenabilité !
  - Limiter son rôle au dialogue http avec les clients/navigateurs
  - *L'utiliser dans une architecture de qualité (MVC)*