Les Entreprise JavaBeans (EJB)

Jean-Marc Farinone
farinone@cnam.fr

Samia Bouzefrane
samia.bouzefrane@cnam.fr

Maîtres de Conférences
Conservatoire National des Arts et Métiers
CNAM Paris (France)

Architecture des Entreprise JavaBeans (EJB)

L'idée essentielle

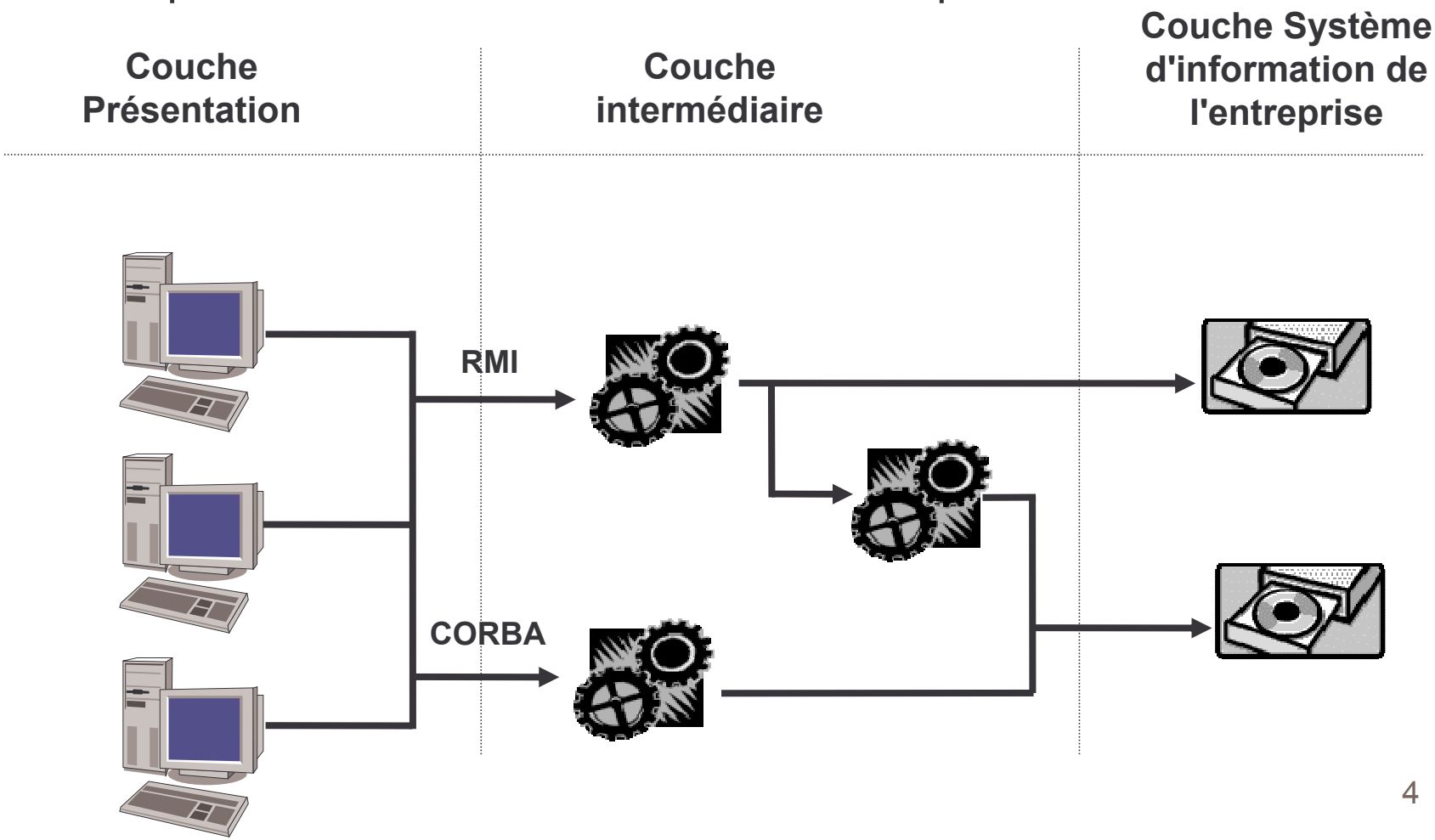
- Se concentrer sur la logique métier à développer (= modéliser, coder les notions du domaine, …) et sous-traiter les problèmes connus de :
  - Persistance
  - Transactions
  - Sécurité
  - Réserve (pool) d'objets, monter en charge, etc.

à un conteneur.

- Donc fabriquer des composants qui s'intégreront bien entre eux et avec le conteneur
- C'est le mariage entre le monde transactionnel et le monde des composants orienté objet
- Version 3.0 depuis le 27 juin 2005
- Compatibilité et interopérabilité avec les EJB 2.1

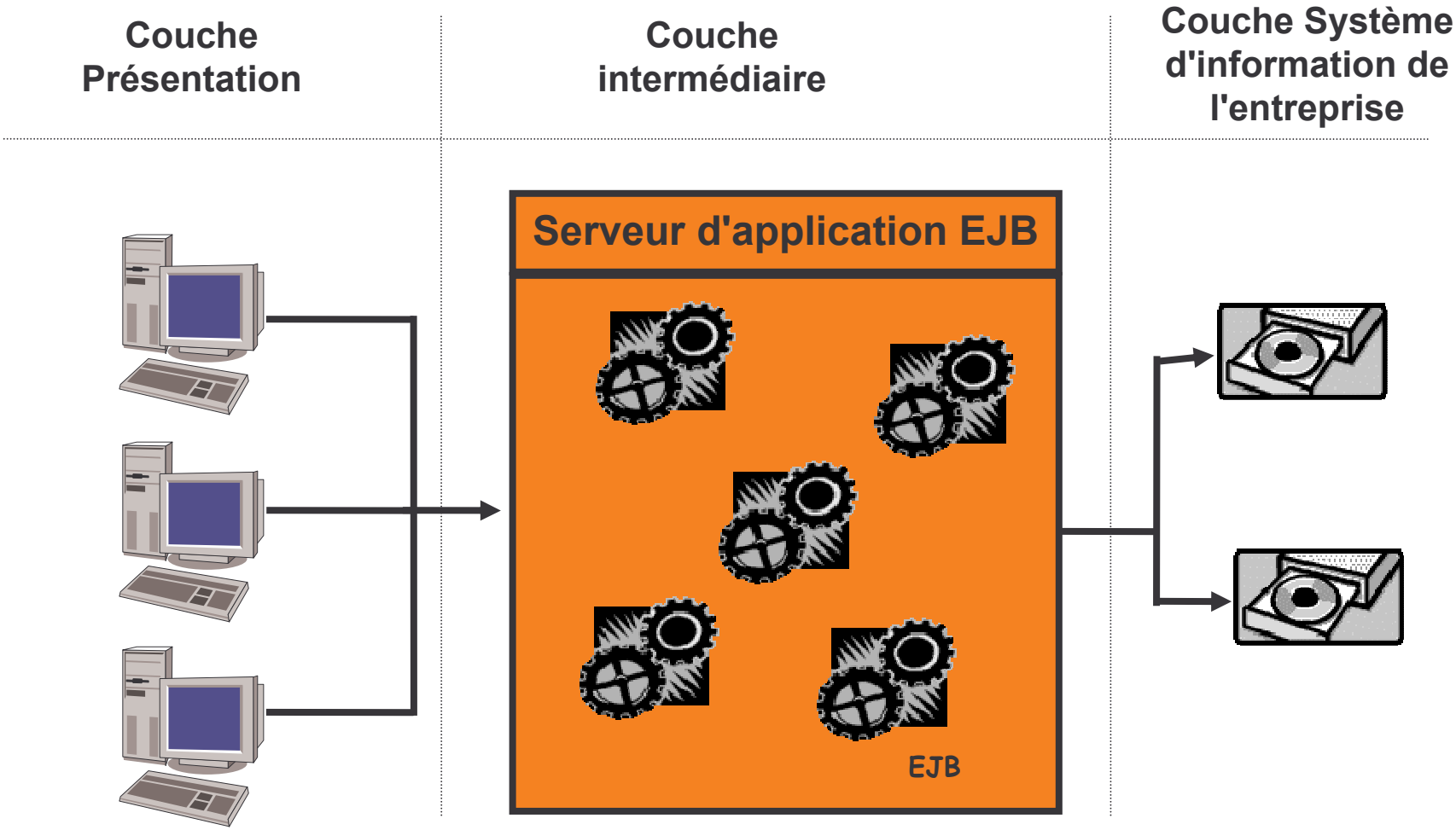
Architecture des EJB (1/2)

- On passe d'une architecture N-tiers classique :



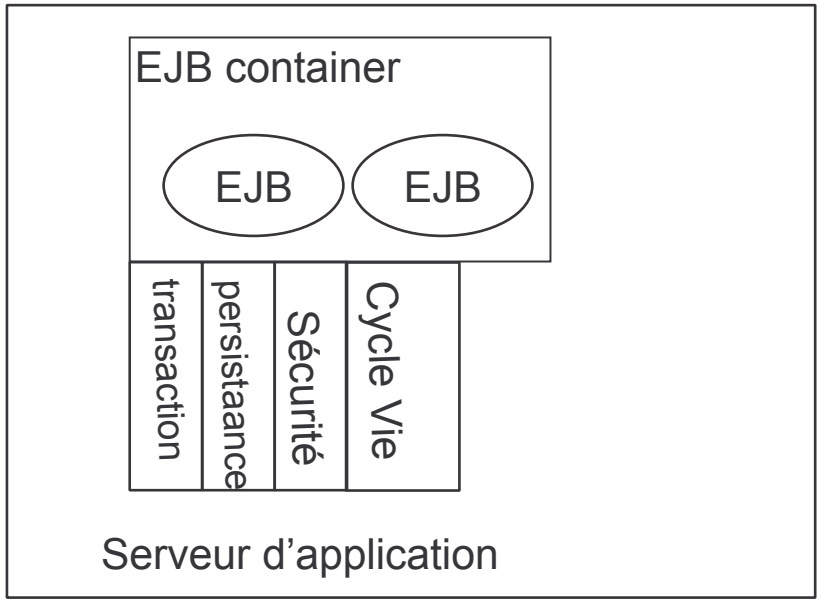
Architecture des EJB (2/2)

⌘ ... à une architecture qui encapsule les composants

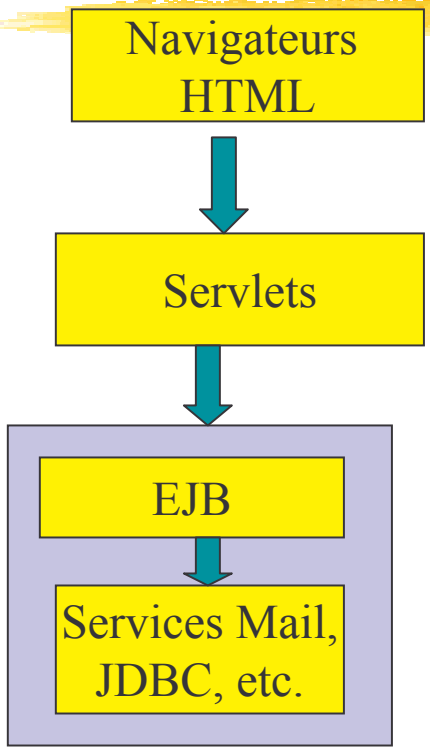


Ce qu'amène le conteneur

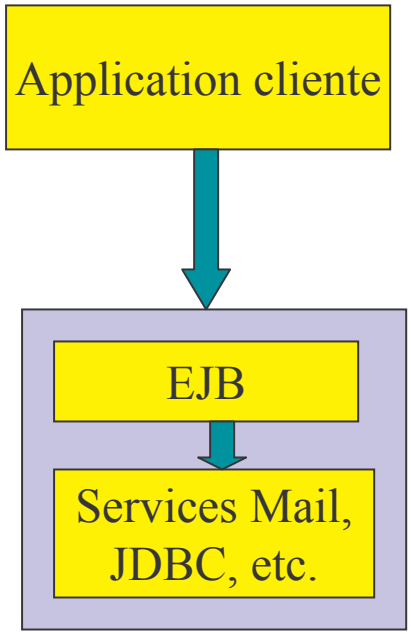
- amène des services de nommage, de gestion du cycle de vie, persistance, sécurité, transaction répartie
- ces services peuvent demander de lancer des méthodes fournies par le bean
- encapsulation d'un composant : les clients devront passer par le conteneur pour accéder à un EJB



Configurations possibles pour une application J2EE



Client léger : navigateur Web



Client lourd

Les acteurs d'une application utilisant EJB

Différents niveaux de responsabilité (rôle) :

- Le fournisseur des EJBs (développeur d'EJB)
: des composants métier réutilisables (par achat ou développement interne)

- L'assembleur d'applications : l'acteur qui construit une application à partir d'EJBs existants

- Le déployeur : l'acteur qui récupère l'application et qui la déploie sur un serveur d'applications

- L'administrateur : l'acteur qui contrôle et supervise le fonctionnement du serveur d'application

- Le fournisseur de serveurs : l'éditeur qui commercialise un serveur d'application

Fournisseurs de serveurs d'applications J2EE

Offre commerciale

- IBM / WebSphere (n° 1)
- BEA / WebLogic
- Sun One
- Oracle 9i Application Server
- Et aussi Borland Entreprise Server, Macromedia / Jrun, SAP Web
- Application Server, Iona / Orbix E2A

Offre « open source »

- JBoss (n° 1)
- JOnAS
- EJB : OpenEJB, EJBean

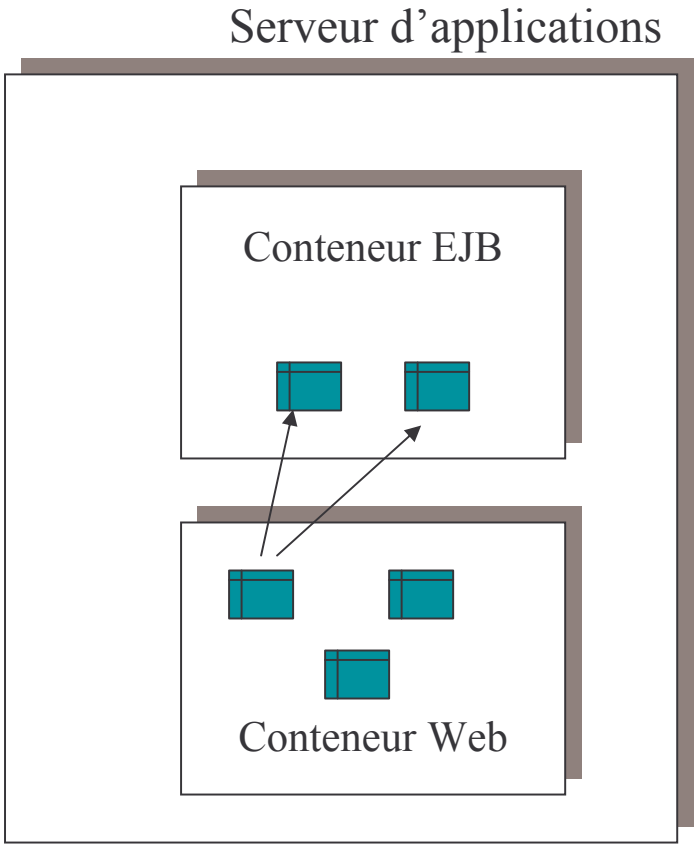
- Voir la liste des serveurs sur : http://java.sun.com/j2ee/licensees.html

Plus d'informations …

- http://java.sun.com/
- http://www.theserverside.com/
- http://developer.java.sun.com/developer/technicalArticles/J2EE/
- http://developer.java.sun.com/developer/onlineTraining/J2EE/
- http://www.triveratech.com/
- http://jonas.objectweb.org/

Serveur d'applications

Application J2EE =
- zéro, un ou plusieurs composants EJB
- zéro, un ou plusieurs composants Web
- reliés par un schéma d'assemblage



Composants Web/1

- Web Bean est un ensemble de :
  - JSP et/ou
  - Servlets et/ou
  - pages HTML

- Packagés dans un fichier archive .war

- Ce sont des composants qui :
  - implantent une logique de présentation simple pour des clients Web
  - servent de passerelle d'accès pour des composants EJB
  - peuvent implanter une logique de petits traitements

Composants Web/2

➢ Exemple :

Fichier.html

```
<html>
<head>
<title> Exemple </title>
</head>
<body>
<form method="POST"  action="/servlet/CalcServlet"  >
Addition de deux nombres : <br>
<input  type=text  name=Val1>
<br>
<input  type=text  name=Val2>
<br>
<input type=submit value="Additionner"  >
</body>
</html>
```

Fonctionnalités d'un container EJB/1

- la connectivité entre les clients et les EJB : le connecteur gère les communications entre les clients et les EJB. Après le déploiement d'un EJB dans un serveur d'applications, le client peut invoquer les méthodes de cet EJB comme si elles étaient situées dans la même machine virtuelle, les communications sont gérées par le middleware sous-jacent.

- la gestion de la persistance : les composants peuvent choisir de déléguer leur persistance au conteneur.

- la gestion des transactions : les composants transactionnels peuvent déléguer la gestion de leurs transactions au conteneur.

Fonctionnalités d'un container EJB/2

⏩ la gestion de la sécurité : le conteneur assure l'application des politiques de sécurité déclarées mais non codées par le développeur.

⏩la gestion de la concurrence : les composants peuvent être invoqués par un seul client ou bien par plusieurs clients simultanément.

⏩la gestion du cycle de vie des composants : le conteneur assure la création et la destruction des instances des composants.

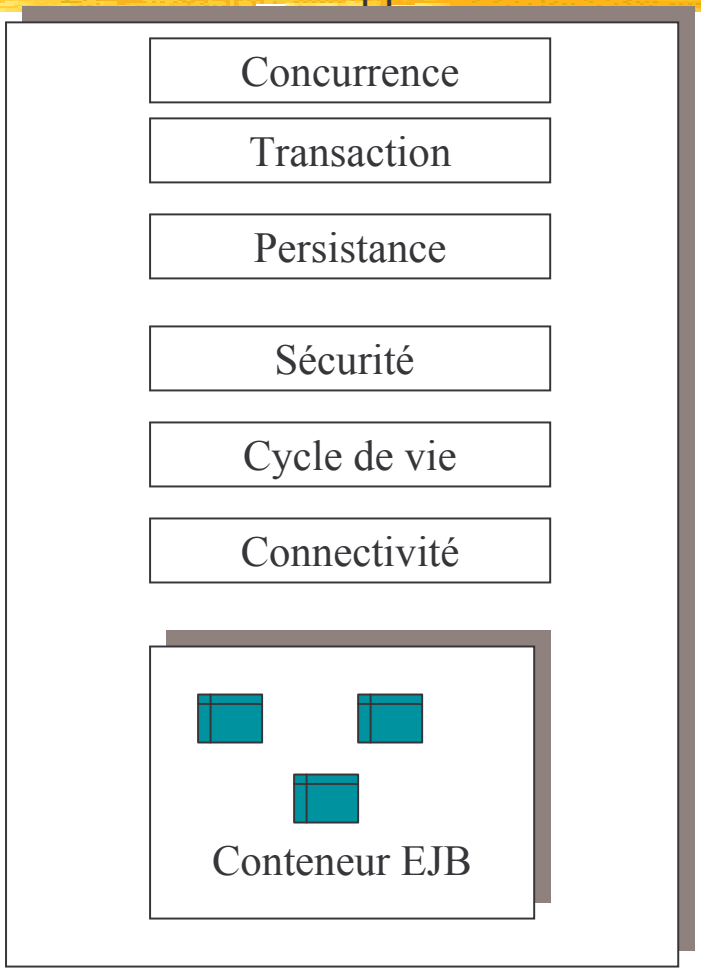
⏩ la création de réserves de connexions : l'obtention d'une connexion sur une base de données est coûteuse en termes de ressources, le nombre de connexions étant limité par le nombre de licences, le conteneur peut gérer une réserve de connexions.

Fonctionnalités d'un container EJB/3

6 services fournis par le serveur d'applications au conteneur EJB :

- transaction
- persistance
- sécurité
- cycle de vie
- concurrence
- connectivité

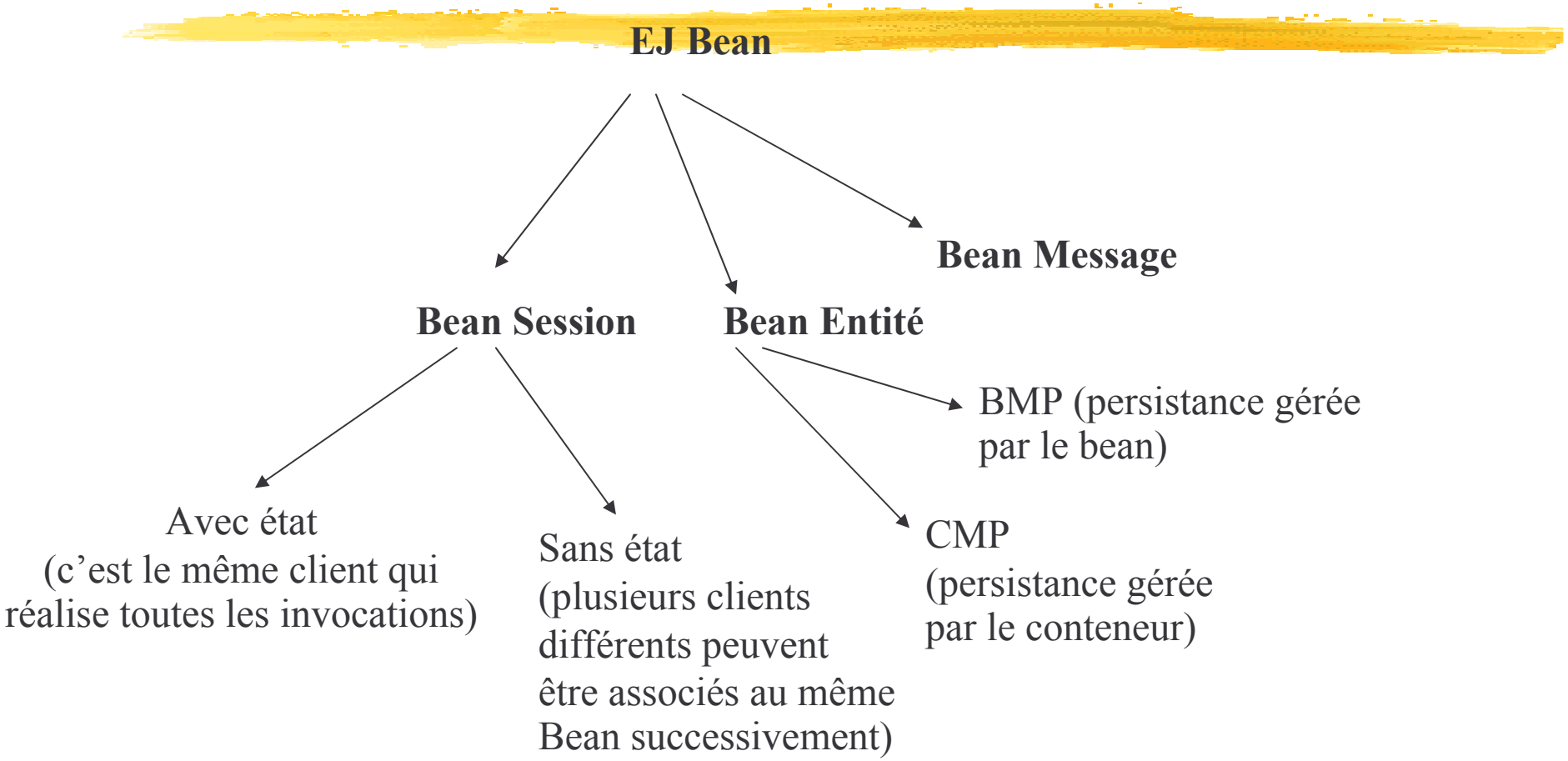
Ces services sont intégrés dés le départ à la plate-forme.



Composants EJB

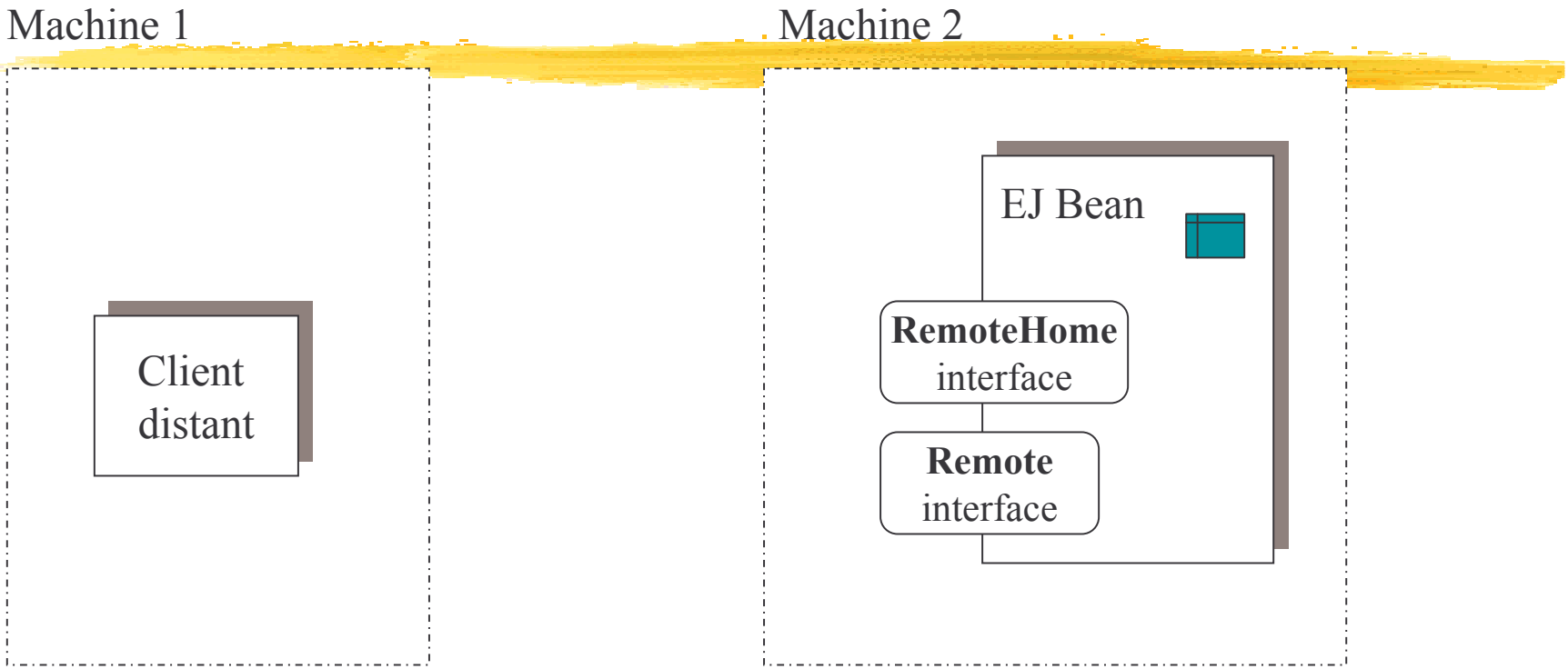
- Composants applicatifs de J2EE (code métier)
- Potentiellement répartis et transactionnels
- se focalisent sur la logique applicative
- sont portables d'un serveur d'application à un autre
- Trois profils
  - **Session** : instances dédiées à un contexte d'interaction d'un client particulier
  - **Entité** : instances partagées représentant les données de l'entreprise
  - **Orienté message** : instances neutres réagissant à l'arrivée de messages asynchrones
- Gérés par le « container » EJB

Les EJ Beans/1



Syntaxe et notions EJB 2.x

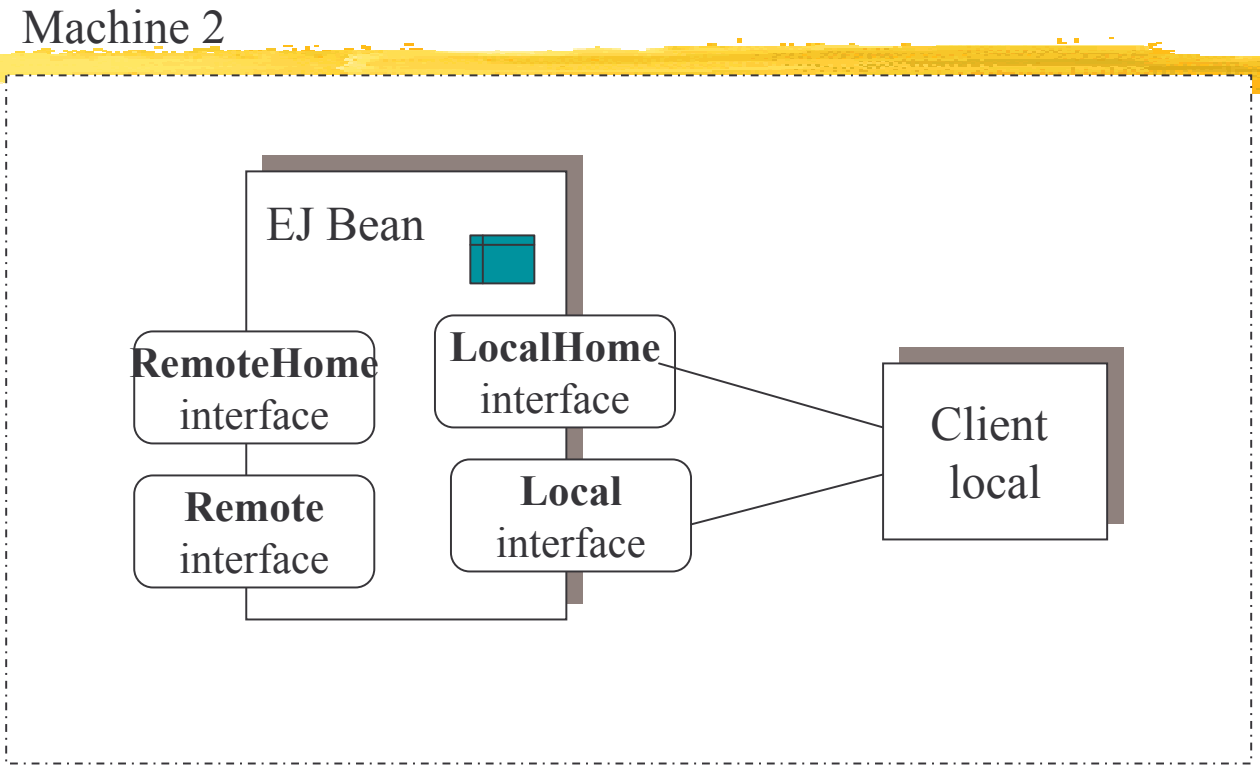
Les EJ Beans/2



Chaque EJ Bean session ou entité fournit deux interfaces d'accès **distant**

- **`Remote`** : les services « métiers » (méthodes) fournis par le bean
- **`RemoteHome`** : interface de gestion du composant (création, recherche, destruction d'instances)

Les EJ Beans/3



+ éventuellement deux interfaces d'accès **local** (meilleure performance pour les clients hébergés dans le même conteneur).

- **`Local`** : les services « métiers » (méthodes) fournis par le bean
- **`LocalHome`** : interface de gestion du composant (création, recherche, destruction d'instances)

Les Beans Session/1

- Session dont la durée de vie est liée à celle de son client, c'est une prolongation du processus client dans un serveur d'application
- Mis en disponiblité lorsque le client n'en a plus besoin (d'où l'idée de session)
- Bean à durée d'utilisation plutôt courte
- Un Bean Session est appelé par son client, utilisé ensuite par son client

1. Bean Session sans état (Stateless session bean)
   - Ne préserve pas d'état d'un appel à un autre
   - Deux instances quelconques d'un tel bean sont équivalentes
   - Exemple: services de calcul, services de recherche d'informations dans une BDD
2. Bean Session avec état (stateful session bean)
   - Effectue des opérations pour le compte du client
   - Gère un état en mémoire (suite d'interaction) pour maintenir l'état du client
   - Exemple: un panier de commandes sur un site de commerce électronique avec les attributs nom du client et les articles sélectionnés

Les Beans Session/2

Quand utiliser un Bean Session ?

- Pas de besoin spécifique de partage de données entre les clients

1. Bean Session sans état (Stateless session bean)
   - Pour des tâches génériques
   - Pour consulter en **lecture seule** des données persistantes
   - Efficaces et faciles à implémenter
   - Les données sont passées comme paramètres de la méthode

1. Bean Session avec état (stateful session bean)
   - L'état du Bean représente l'état de l'interaction entre le client et le Bean
   - Le Bean doit conserver de l'information entre deux invocations du client
   - Dédié à un client pendant toute sa durée d'utilisation
   - Le même Bean est utilisé pour servir tous les appels du même client

Les Interfaces métier Beans Session

Définissent les interfaces que le Bean peut rendre au client. Elles sont locales ou distantes.

- Interface métier distante (**`Remote`**)
  - Accessibles par des composants locaux ou distants au Bean
  - Hérite de l'interface `javax.ejb.EJBObject`
  - Les méthodes de l'interface lèvent l'exception `RemoteException`

- Interface métier locale (**`Local`**)
  - Accessible uniquement par les composants situés sur la même machine que le Bean
  - Hérite de l'interface `javax.ejb.EJBLocalObject`
  - Pas d'exception `RemoteException`

Interface Remote d'un Bean Session

Exemple.

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Calc extends javax.ejb.EJBObject {
    public double add (double val1, double val2) throws RemoteException;
}
```

Interface Local d'un Bean Session

Exemple.

```
import javax.ejb.EJBLocalObject;

public interface CalcLocal extends javax.ejb.EJBLocalObject {
    public double add(double val1, double val2) ;
}
```

Les Interfaces de gestion du cycle de vie

Locales ou distantes, gèrent la création, la recherche et la suppression des EJ Beans.

- Interface Home distante (**`RemoteHome`**)
  - Gère le cycle de vie du Bean
  - Hérite de l'interface `javax.ejb.EJBHome`
  - Les méthodes de l'interface lèvent les exceptions `RemoteException` et `CreateException`

- Interface Home locale (**`LocalHome`**)
  - Gère le cycle de vie du Bean
  - Hérite de l'interface javax.ejb.`EJBLocalHome`
  - Pas d'exception `RemoteException`

- Méthodes possibles : `create` (création d'instances de bean, retourne l'interface `Remote` ou `Local` selon que l'interface est locale ou distante)

  Plusieurs méthodes `create` peuvent être définies avec plusieurs signatures

Interface Home d'un Bean Session

Exemple.

```
import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import ja.rmi.RemoteException;


public interface CalcHome extends javax.ejb.EJBHome {
    public Calc create() throws CreateException, RemoteException;
}
```

Interface HomeLocal d'un Bean Session

Exemple.

```
import javax.ejb.EJBLocalHome;
import javax.ejb.CreateException;

public interface CalcLocalHome extends javax.ejb.EJBLocalHome {
    public CalcLocal create() throws CreateException;
}
```

Implémentation du Bean Session

Classe Java :

- qui définit les méthodes de spécification indiquées dans les interfaces `Remote` et `RemoteHome`

- qui implémente l'interface `javax.ejb.SessionBean`

- la classe d'implémentation de `javax.ejb.SessionBean` ne déclare pas l'implémentation des interfaces

Method Summary	
`void ejbActivate()`	Est appelée quand l'instance passe de l'état passif à l'état actif (swap)
`void ejbPassivate()`	Est appelée quand l'instance passe de l'état actif à l'état passif (swap)
`void ejbRemove()`	La conteneur d'EJB appelle cette méthode avant de détruire l'EJB session
`void setSessionContext()`	Appelé à l'instanciaton de l'EJB

Développement

Fournir des méthodes pour les interfaces `Remote` et `RemoteHome`

- même méthodes que dans l'interface `Remote`
- une méthode `ejbCreate` pour chaque `create` de l'interface `RemoteHome`
- même profil que `create`
- retourne `void`

Exemple de classe d'implémentation

```
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.CreateException;
public class CalcBean implements SessionBean {
    SessionContext sessionContext;
public CalcBean() {}
public void ejbCreate() throws CreateException { }
public void ejbRemove() { }
//public void ejbActivate() { } non utilisee par un Bean sans etat
//public void ejbPassivate() { } non utilisee par un Bean sans etat
public void setSessionContext (SessionContext sessionContext)
{this.sessionContext  = sessionContext;}
public double add(double val1, double val2)
    {   return val1+val2; }
} // fin de CalcBean
```

Client EJB

Développement côté client :

1. Rechercher l'interface `Home` du Bean par son nom via JNDI
JNDI : API accès services nommage
(LDAP, CORBA COSNaming, DNS, RMI registry, etc.)

2. Accéder au Bean
L'interface `Home` permet d'accéder aux instances existantes du Bean ou d'en créer de nouvelles
=> on récupère une référence sur une interface `Remote`

3. Invocation du Bean

Classe Java du Client EJB

```
import javax.rmi.*;
import javax.naming.*;
public class ClientCalc {
   CalcHome calcHome; // reference sur l'objet Home distant
   Calc myCalc; //ref sur l'objet EJB distant
   try {// obtention du contexte initial
     Context ctx = new InitialContext();
     //recherche de l'interface Home distante
     Object ref= ctx.lookup("Calc");
    calcHome = (CalcHome)PortableRemoteObject.narrow(ref,CalcHome.class);
    // creation d'un EJB Calc
    myCalc = calcHome.create();
    //invocation des methodes metier add()
    double res=myCalc.add(2.34, 4.56);
  } catch(Exception e) {e.printStackTrace(); }
  }
```

Couche Présentation/1

➢ Exemple :

Fichier.html

```
<html>
<head>
<title> Exemple </title>
</head>
<body>
<form method="POST"  action="/servlet/CalcServlet"  >
Addition de deux nombres : <br>
<input  type=text  name=Val1>
<br>
<input  type=text  name=Val2>
<br>
<input type=submit value="Additionner"  >
</body>
</html>
```

Couche Présentation/2

CalcServlet.java

```
public class CalcServlet extends HttpServlet {
  private static final String CONTENT_TYPE="text/html";
  private calcHome CalcHome;
```

// a l'initialisation de la servlet, on recupere la reference de l'interface Home de Calc

```
  public void init() throws ServletException {
       try {
    Context ctx=new InitialContext();
    Object ref=ctx.lookup("Calc");
    calcHome = (CalcHome)PortableRemoteObject.narrow(ref,
    CalcHome.class);

     } catch (Exception e) {e.printStackTrace();}

   } // fin de init()
```

Couche Présentation/3

CalcServlet.java (suite)

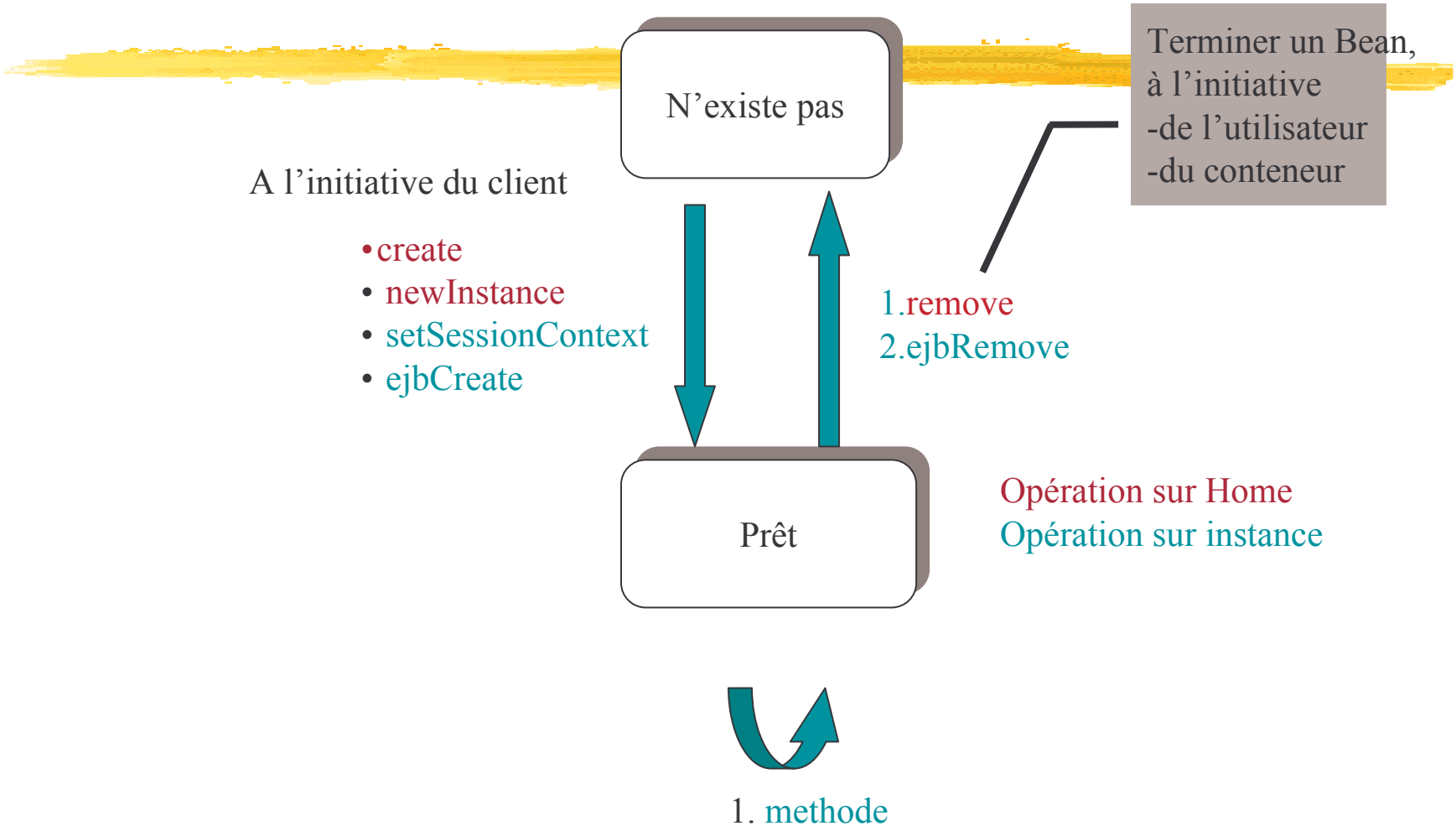
```
  public void doPost(HttpServletRequest requete, HttpServletResponse
reponse) throws ServletException, IOException {
    reponse.setContentType(CONTENT_TYPE);
    PrintWriter sortie=reponse.getWriter();
    Double d1=new Double(requete.getParameter("Val1"));
    Double d2=new Double(requete.getParameter("Val2"));
    double val1= d1.doubleValue();
    double val2=d2.doubleValue();
    // creation d'un EJB Calc
        myCalc = calcHome.create();
        //invocation des methodes metier add()
        double res=myCalc.add(val1, val2);
    sortie.println("<html><head> <title> Resultat </title> </head>
<body>");
```

Couche Présentation/4

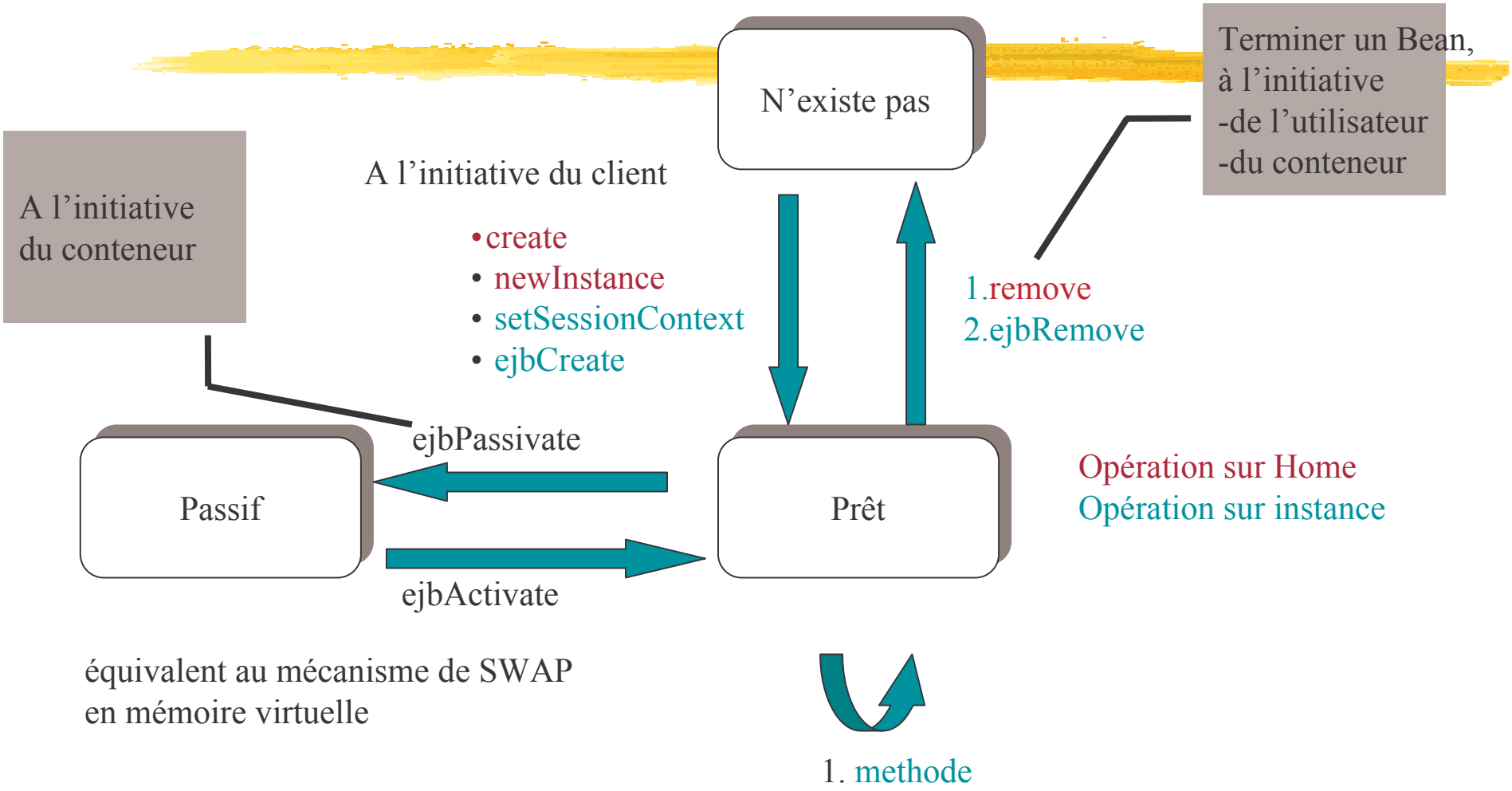
CalcServlet.java (suite)

```
 sortie.println("<p> Résultat de l'Opération " + val1 + "+" + val2 +"="+
"<b>"+res +"</b>"+ "<p>");

 sortie.println("</body></html>");

} // fin de doPost

} // fin de la servlet
```

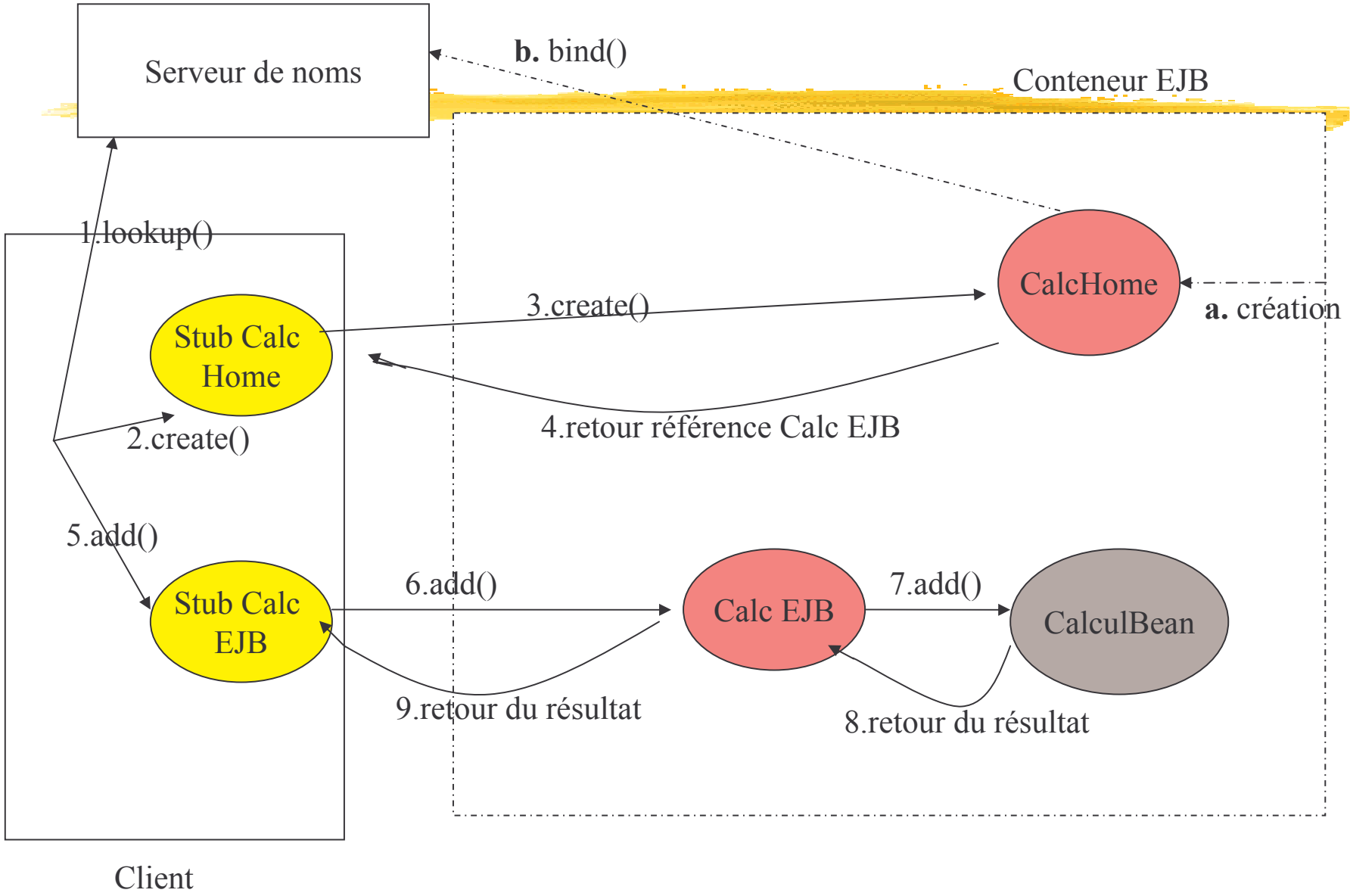
Cycle de vie d'un Bean Session sans état



Cycle de vie d'un Bean Session avec état



Intéraction entre un client et le Bean



Spécifications des propriétés du Bean

Un Bean est défini grâce à ses propriétés :
-Pour un Bean Session, indiquer s'il est avec ou sans état
-Pour un Bean Entité, indiquer si la persistance est gérée par le conteneur ou par le programmeur
-Pour les deux types de Beans, il faut indiquer si la gestion des transactions sera gérée par le conteneur ou par le programmeur

Les propriétés du Bean sont fournies dans un fichier XML appelé descripteur de déploiement

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" "<http://java.sun.com/dtd/ejb-jar_2_0.dtd>">
<ejb-jar>
   <enterprise-beans>
     <session>
      <display-name>Calc</display-name>
      <ejb-name>Calc</ejb-name>
      <home>rep.Calc</home>
      <remote>rep.Calc</remote>
      <ejb-class>rep.CalcBean </ejb-class>
    <session-type>Stateless </session-type>
    <transaction-type>Container</transaction-type>
      </session>
```

Spécifications des propriétés du Bean

```
</enterprise-beans>
    <assembly-descriptor>
    <container-transaction>
        <method>
            <ejb-name>Calc</ejb-name>
            <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
     </assembly-descriptor>
</ejb-jar>
```

Déploiement du Bean

Une fois que le Bean est défini, il faut le déployer dans un serveur d'application. Il faut alors assembler tous les éléments du Bean dans un fichier JAR qui va contenir :

- Les interfaces du Bean
- La classe du Bean et les classes auxiliaires
- Le descripteur de déploiement.

Les serveurs d'application offrent des interfaces graphiques, appelées consoles d'administration qui permettent de rééditer le descripteur de déploiement. Une fois l'objet déployé, ce dernier enregistre la référence à l'objet EJBHome dans un serveur de noms accessibles via JNDI.

Calc.jar = { Calc.class, CalcLocal.class, CalcHome.class,CalcHomeLocal.class, CalcBean.class, ejb-jar.xml }

A partir du fichier JAR, le serveur est capable d'extraire son contenu et de rendre le Bean utilisable par les clients, càd, générer des composants nécessaires à la communication (Stub et Skeleton) entre le Bean et ses clients (on parle de code déployé), on peut trouver :

-l'objet EJB qui implémente l'interface métier
-L'objet EJBHome qui implémente l'interface Home.

Restrictions des EJBs et Gestion des ressources

Restrictions des EJBs

-les EJBs ne doivent pas manipuler des threads
-les EJBs ne doivent pas effectuer des opérations d'entrée/sortie
-les EJBs ne doivent pas manipuler les sockets serveur
-les EJBs ne doivent pas charger des librairies écrites en code natif (C, C++)

Gestion des ressources

Les serveurs EJB gèrent une charge importante tout en gardant un bon niveau de performance.
Les EJBs mettent en œuvre deux techniques pour gérer un grand nombre de Beans (donc de clients):les pools d'instance et le partage de ressources entre les Beans
Un pool d'instances des EJBs est une réserve d'instances créées à l'avance: technique utilisée par les Beans Session sans état, les Beans Message et Entité.

Un Entity Bean /1

Un Bean Entité

- n'est pas lié à la durée de vie des sessions avec les clients
- peut être **partagé** par plusieurs clients
- ses **données** sont gérées de manière **persistante**
- est identifié de manière unique par une **clé primaire**
- peut être **relié** à d'autres entity beans (≡ relations dans un SGBDR)

2 sortes d'EJB entités

Bean Managed Persistence (BMP)

- - la gestion de la persistance est à la charge du Bean

Container Managed Persistence (CMP)

- la persistence des données est gérée automatiquement par le conteneur
- configuration via un descripteur de déploiement

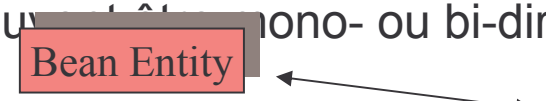
Un Entity Bean/2

Un Bean Entity est équivalent à un tuple dans une table relationnelle

Il existe 3 catégories de variables d'instance dans un entity bean :
- **persistante** ≡ attribut de la table
- **relationnelle** ≡ clés étrangère qui réf. un attribut d'une autre table/bean
-**temporaire** ≡ donnée "de travail" non sauvegardée

Cardinalité des relations entre entity beans
- 1-1
- 1-n (ex. :une commande contient n lignes)
- n-1 (ex. : plusieurs lignes de commandes peuvent concerner le même produit)
-n-n (ex. : un cours comporte plusieurs étudiants qui suivent plusieurs cours)

Relations peu[illegible]ono- ou bi-dir



Nom	Solde
Anne	1200
Eric	300

Développement

Un Bean Entity :

Deux interfaces + une classe

Interface `Remote` .
- definit les services "métiers" fournis par le *bean*
- étend l'interface `javax.ejb.EJBObject`

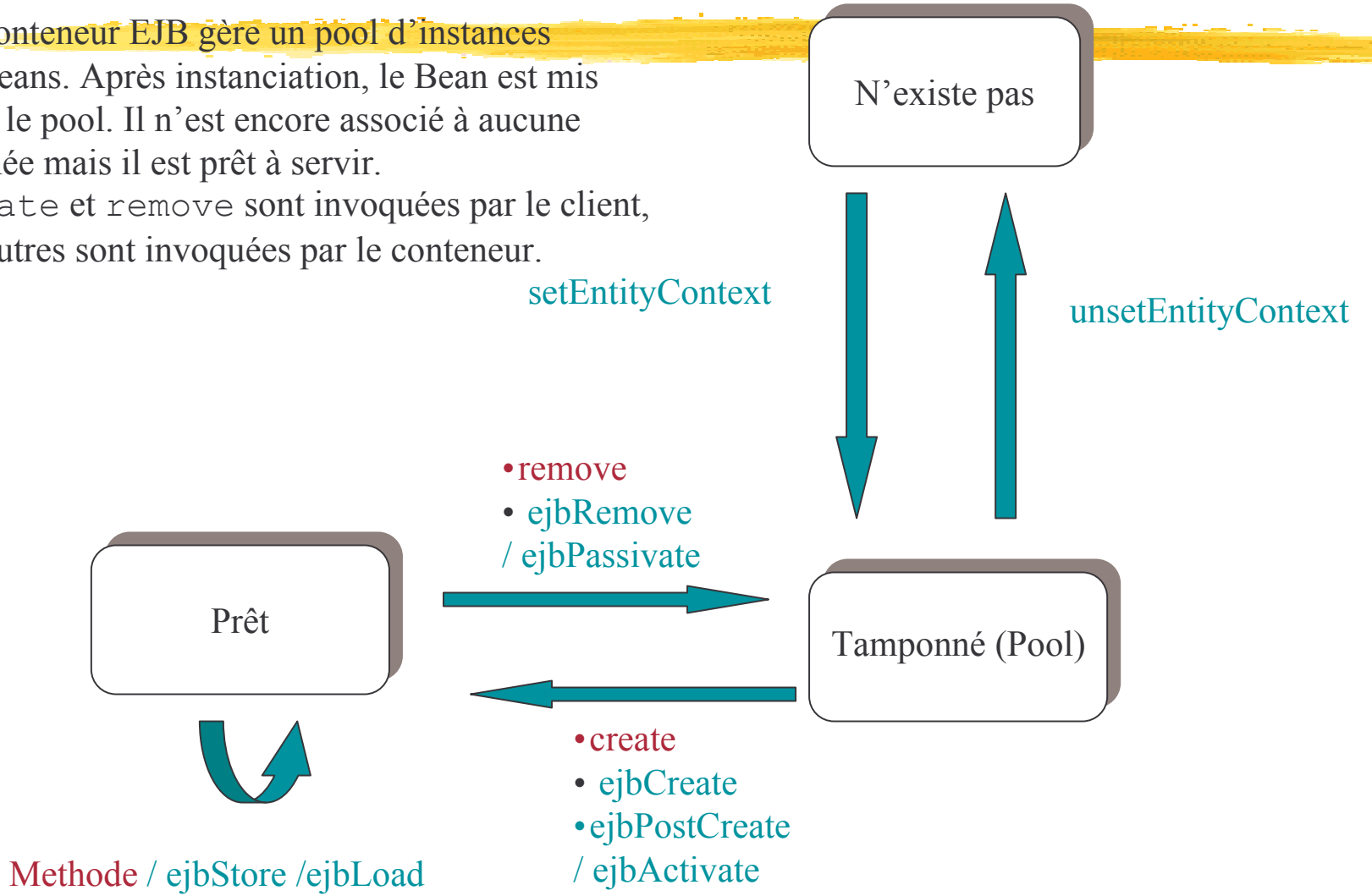
Interface `RemoteHome`
- definit l'interface de gestion du *bean*
- étend l'interface `javax.ejb.EJBHome`

Classe Java
- fournissant des méthodes pour les interfaces `Remote` et `RemoteHome`
-implantant l'interface `javax.ejb.EntityBean`

+ éventuellement interfaces `Local` et `LocalHome`
+ éventuellement classe clé primaire

Cycle de vie d'un Bean Entity

Le conteneur EJB gère un pool d'instances de Beans. Après instanciation, le Bean est mis dans le pool. Il n'est encore associé à aucune donnée mais il est prêt à servir.
`create` et `remove` sont invoquées par le client, les autres sont invoquées par le conteneur.



Bibliographie (EJB)

Composants Logiciels, Cours de Gérard Florin, CEDRIC (CNAM), 2004.

EJB 2.0 : Mise en œuvre, Christophe Calandreau, Alain Fauré
Nader Soukouti, *Ed. Dunod*, 2002, ISBN : 2 10 004 729 9.

L'environnement J2EE : principes, fonctions, utilisation
P. Déchamboux, *École d'été sur les Intergiciels et sur la Construction d'Applications Réparties*, ICAR'2003, http://sardes.inrialpes.fr/ecole/2003/

Enterprise java Bean
Lionel Seinturier, Université Pierre & Marie Curie, octobre 2003
http://www-src.lip6.fr/homepages/Lionel.Seinturier/middleware/ejb.pdf

Programmation Java côté Serveur : **Servlets, JSP et EJB**
Andrew Patzer, *Ed. Eyrolles*, 2000, ISBN : 2 212 09109 5.

Tutorial J2EE de Sun : http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

Les EJB Message-Driven

Message-Driven EJB =

- Ce sont des EJB qui sont abonnés à des "systèmes d'envoi" de messages
- Des clients publient sur ces systèmes.
- L'EJB s'abonne à ces systèmes.
- ~ traitement asynchrone d'événements
- => il n'y a pas de connexion directe entre un client et un message-driven bean
- => le client ne sait pas quel message-driven EJB va traiter son message
- => le message-driven EJB ne sait pas quel client a posté le message

- On utilise les concepts et la technologie JMS (Java Message Service)

JMS =

- Java Message Service
- Traitement asynchrone ou/et événementiel. Les récepteurs peuvent recevoir des messages même s'ils étaient inactifs au moment de l'émission
- JMS est un MOM (Middleware Oriented Message) : les applications receptives ne sont pas obligées de fonctionner en permanence. C'est le MOM qui reste actif.
- Spécification de Sun. Trouver des implémentations

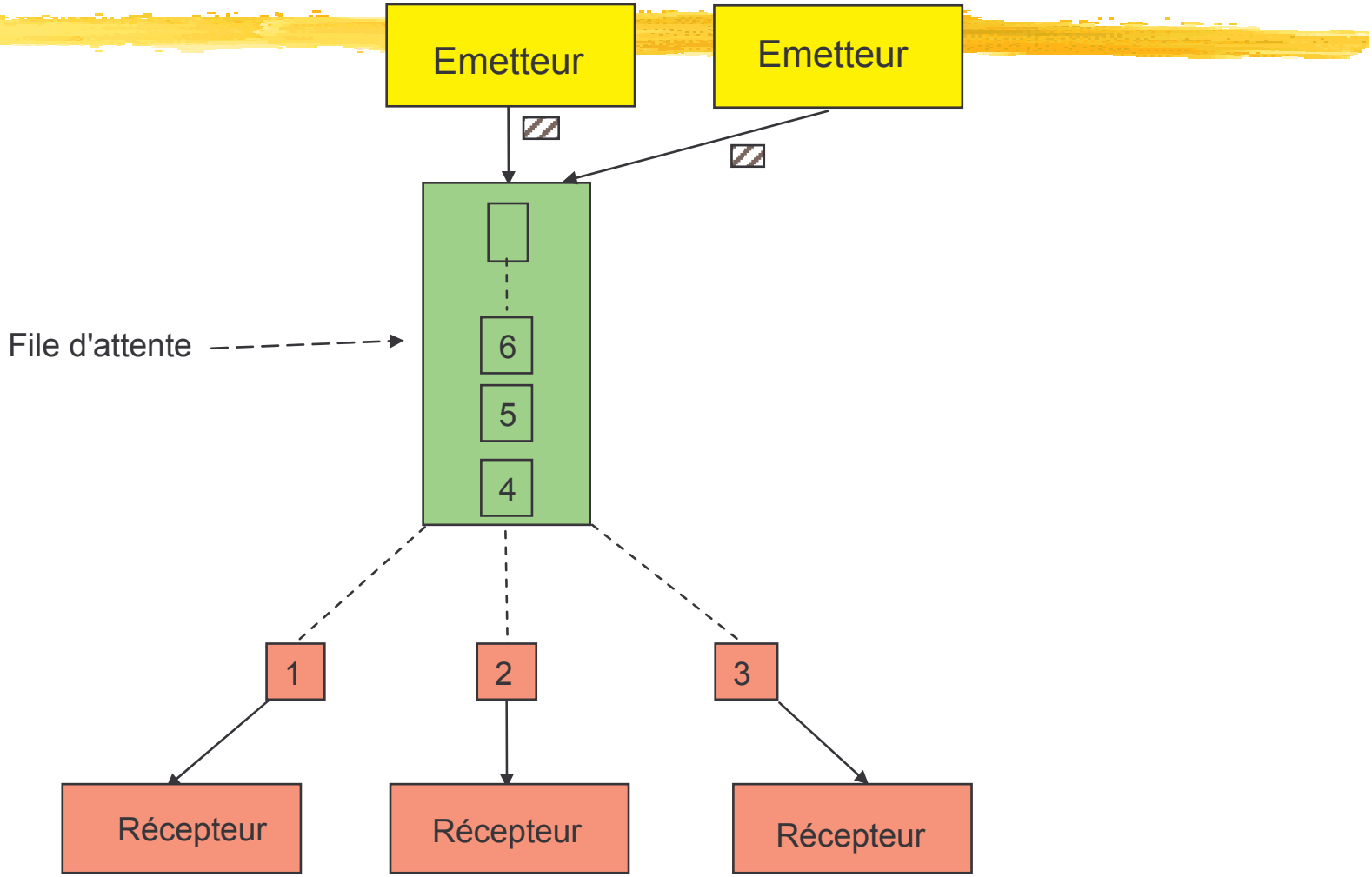
- Notion de file d'attente (queue) et sujet (topic)
- Exemple d'utilisation :
  - Un appel d'offres est lancé sur une liste de diffusion (= un sujet de discussion)
  - Les candidats répondent sur une file de réponses

JMS : les files (d'attente) et les sujets

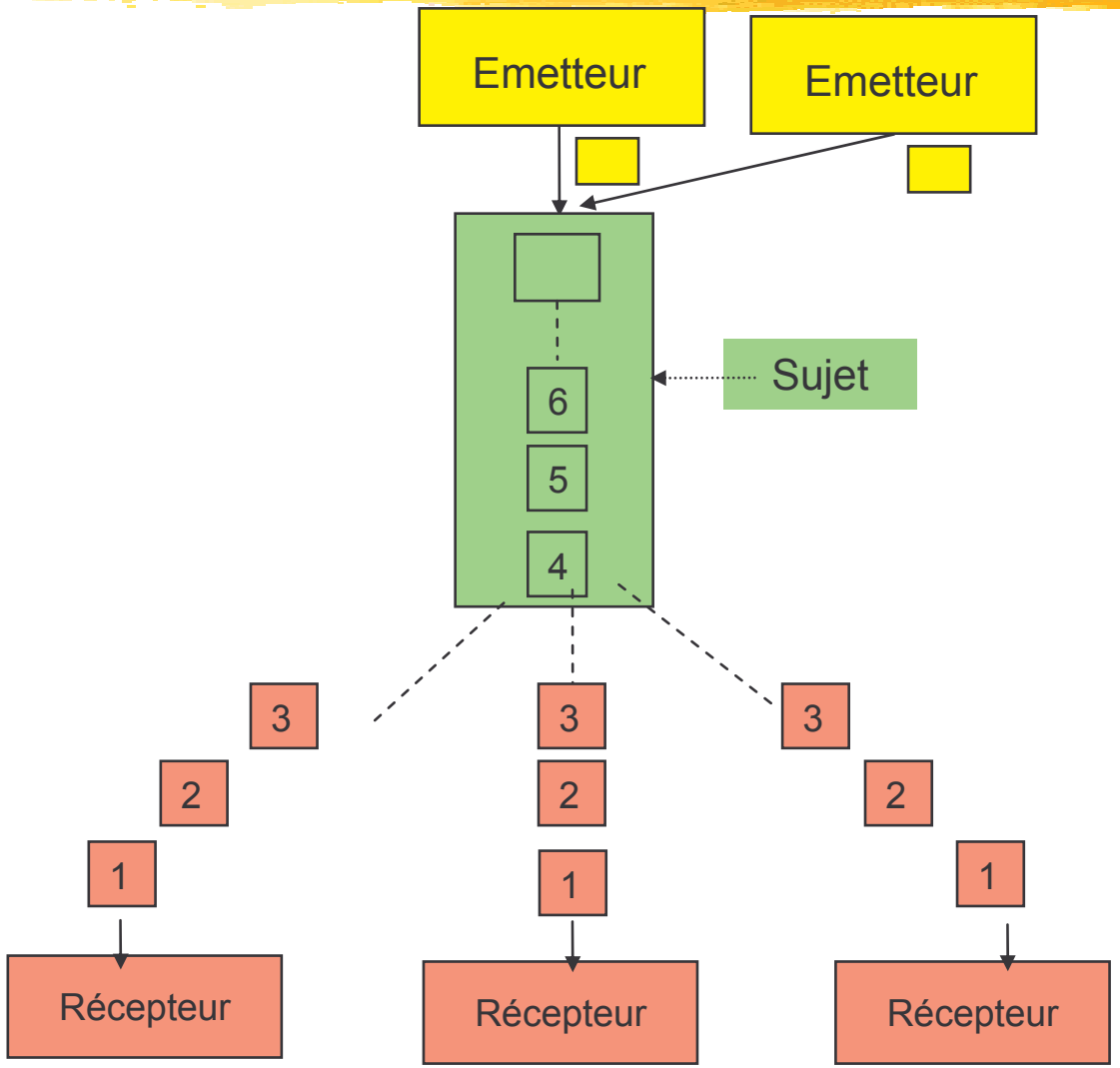
- File d'attente (queue) : comme à la poste. Une file commune. On passe à un seul guichet = on est traité par une seule personne

- Sujet (topic) : une liste de diffusion. On s'abonne à une liste de diffusion. On reçoit, comme tous les abonnés, les bulletins publiés sur cette liste.
- Sujet = publication/abonnement

File d'attente JMS



Sujet JMS


Emetteur
Emetteur
Sujet
6
5
4
3
2
1
3
2
1
3
2
1
Récepteur
Récepteur
Récepteur

EJB Message Driven : un peu de code
Le code du client

- Source Java EE 5 Tutorial chapitre 23 EJB MD
- `SimpleMessageClient` envoie des messages sur une file d'attente. L'EJB MD lira dans cette file d'attente.
- Le client envoie des messages sur la file d'attente.

```
message = session.createTextMessage();
for (int i = 0; i < NUM_MSGS; i++) {
  message.setText("This is message " + (i + 1));
  System.out.println("Sending message: " +
    message.getText());
  messageProducer.send(message);
}
```

EJB Message Driven : un peu de code
Le code de l'EJB MD (1/2)

- Les spécifications EJB 3.0 précisent qu'un EJB MD :
  - doit implémenter l'interface `javax.jms.MessageListener` (si on utilise JMS)
  - doit utiliser les annotations MessageDriven si il n'utilise pas un descripteur de déploiement
  - doit être une classe public et doit contenir un constructeur public sans argument
  - ne doit pas être une classe abstraite ou finale et ne doit pas définir la méthode `finalize()`.
  - N'a pas de local ou remote interface (!= EJB session ou entité)
  - Pas de méthodes métiers. La méthode qui sera invoquée lors de l'utilisation de l'EJB est `onMessage(...)`.

EJB Message Driven : un peu de code
Le code de l'EJB MD (2/2)

- Le code utilise les noms JNDI commun avec le client :

```
@MessageDriven(mappedName="jms/Queue")
public class SimpleMessageBean implements MessageListener {
   ...}
```

- Le code de la méthode `public void onMessage(Message msg)`

```
static final Logger logger = Logger.getLogger("SimpleMessageBean");
public void onMessage(Message inMessage) {
  TextMessage msg = null;

  try {
    if (inMessage instanceof TextMessage) {
      msg = (TextMessage) inMessage;
      logger.info("MESSAGE BEAN: Message received: " + msg.getText());
  } catch (JMSException e) {
    …
  }
```

Démonstration (1/2) : A) Créer les ressources JMS

- Dans `<INSTALL_TUTORIAL>/javaeetutorial5/examples/ejb/simplemessage`
- Il faut tout d'abord :
  - Une fabrique JMS de connexion
  - Une ressource JMS de destination
  - Une destination physique correspondant à ces ressources (noms logiques)
- obtenu par les commandes :

```
asant create-cf
asant create-queue
```

- Ces commandes
  - construisent une fabrique de connexion nommée jms/ConnectionFactory
  - créée physiquement une file d'attente nommée jms/Queue
- A ne faire qu'une fois dans l'architecture Application Server PE 9

Un descripteur de déploiement pour un EJB MD

- Alternative aux annotations
- Fichier XML `sun-ejb-jar.xml` à ajouter au déploiement

```
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>MessageBean</ejb-name>
      <mdb-connection-factory>
        <jndi-name>jms/JupiterConnectionFactory</jndi-name>
      </mdb-connection-factory>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

- L'élément XML `ejb-name` contient le nom du paquetage du bean
- L'élement `mdb-connection-factory` est un nom JNDI indiquant la fabrique de connexion pour le bean.

Fin