Introduction au développement par composants (distribués) Java EE

Intervenant : Chouki TIBERMACINE
Bureau : LIRMM (E.311)
Tél. : 04.67.14.97.24
Mél. : Chouki.Tibermacine@umontpellier.fr
Web : https://github.com/ctiber/composants/

Plan du cours

- Introduction aux architectures multi-niveaux (*N-Tiers*)
- Plate-forme Java EE (*Enterprise Edition*)
- Développement par composants JEE

Plan du cours

- Introduction aux architectures multi-niveaux (*N-Tiers*)
- Plate-forme Java EE (*Enterprise Edition*)
- Développement par composants JEE

Applications distribuées

- Application distribuée : ensemble de programmes s'exécutant sur des machines (physiques ou virtuelles) hôtes différentes

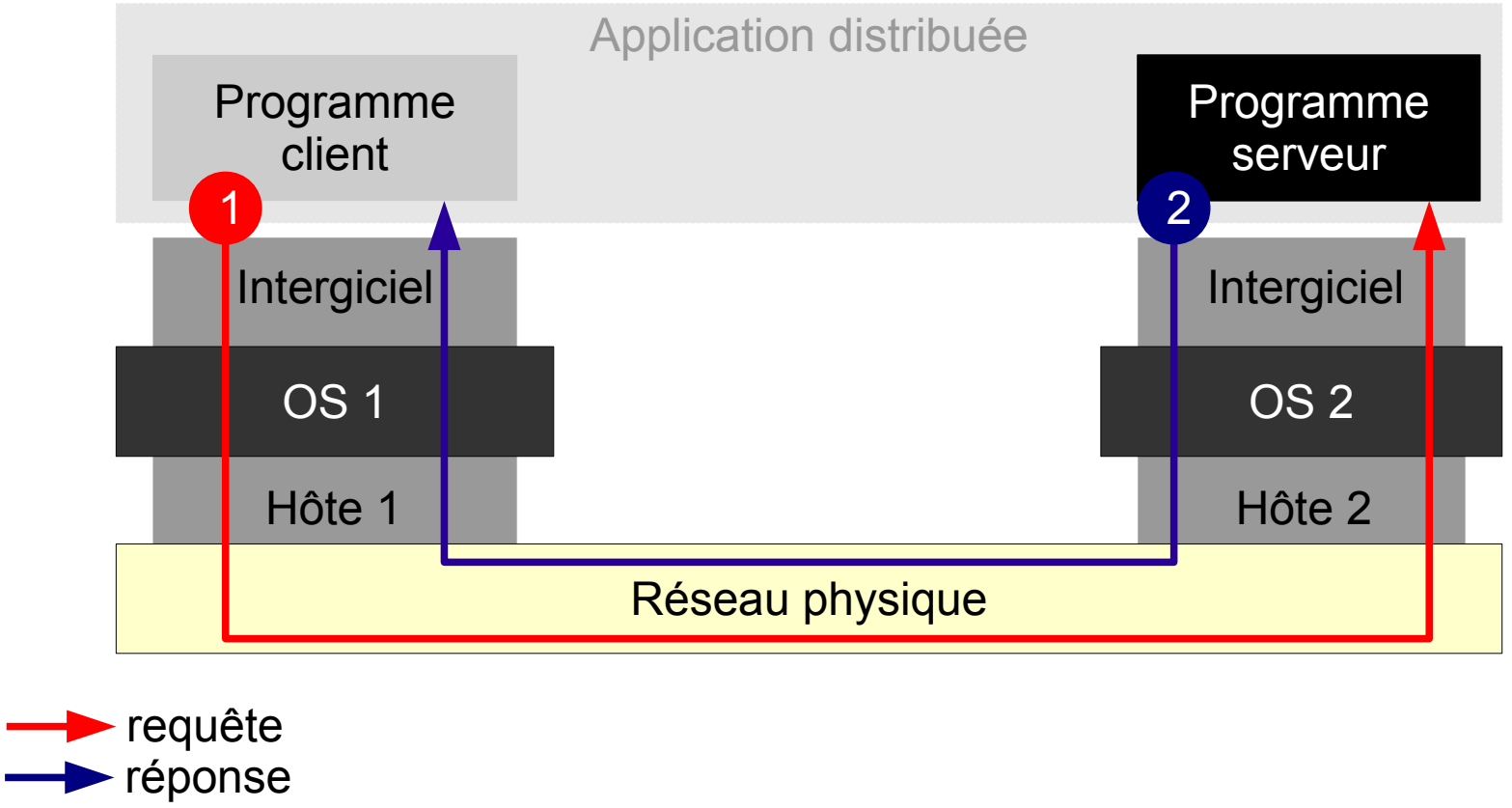
- Avantages multiples :
  - Répartition de charge
  - Optimisation de l'utilisation des ressources
  - Prise en compte de l'hétérogénéité des plate-formes
  - …

- Middleware (intergiciel) : couche logicielle s'interfaçant entre l'OS et les applications et garantissant une transparence vis-à-vis de la distribution des machines hôtes

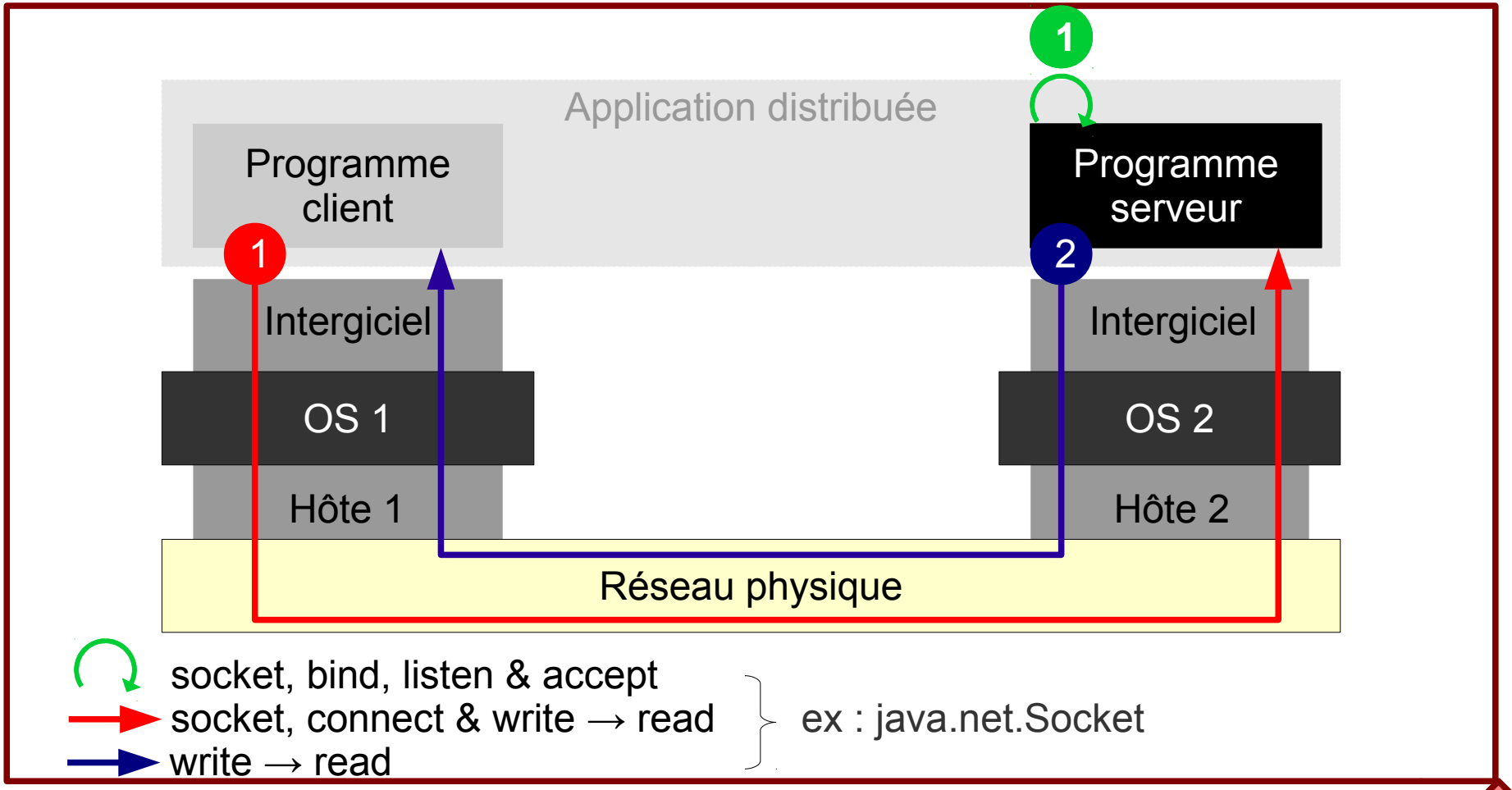
Services (techniques) offerts par les intergiciels

- Interopérabilité : capacité des applications à communiquer même si elles sont composées de programmes hétérogènes (langages différents : cas de CORBA, OS différents : cas de JEE, ...)
- Gestion des transactions : garantie des propriétés ACID (Atomicité, Cohérence, Isolation et Durabilité)
- Sécurité : garantie de l'intégrité et validité des données/utilisateurs
- Gestion du nommage : capacité d'utiliser des noms logiques indépendants de la localisation
- ...

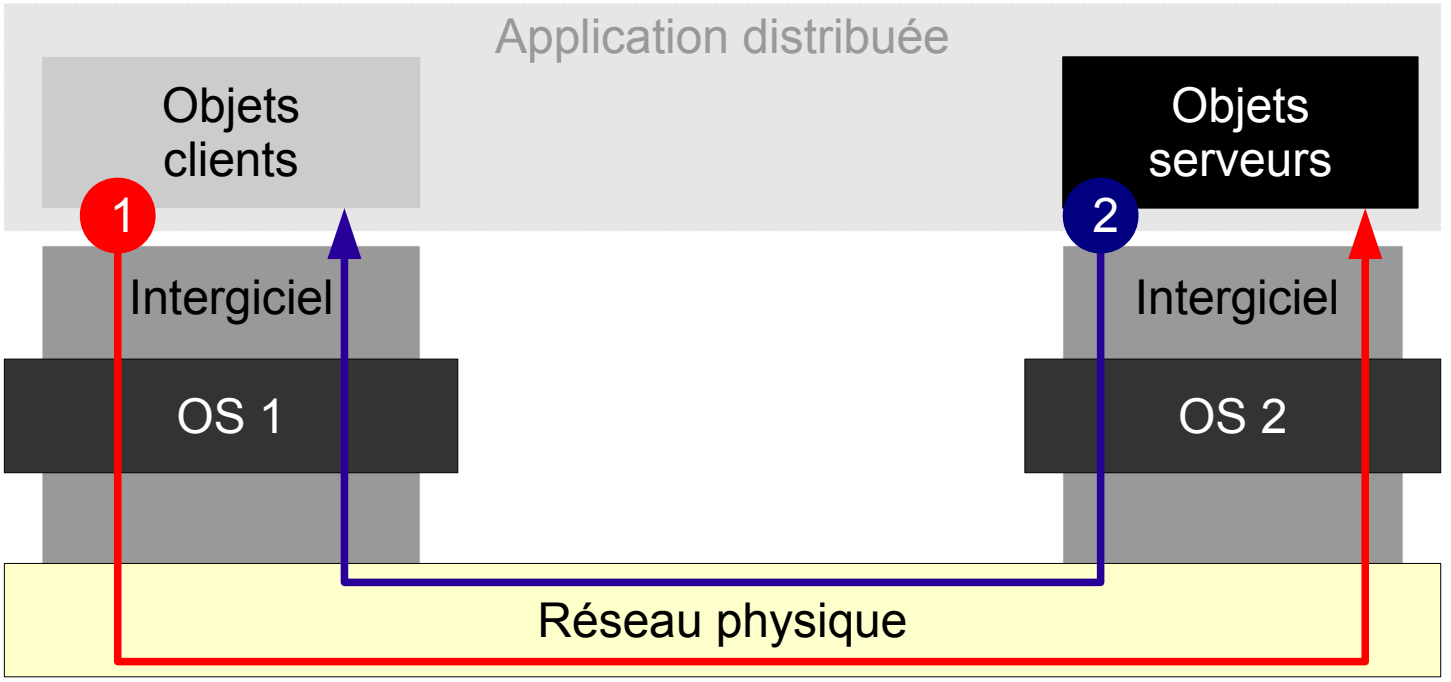
Application distribuée avec une architecture client-serveur



Une architecture client-serveur avec des sockets



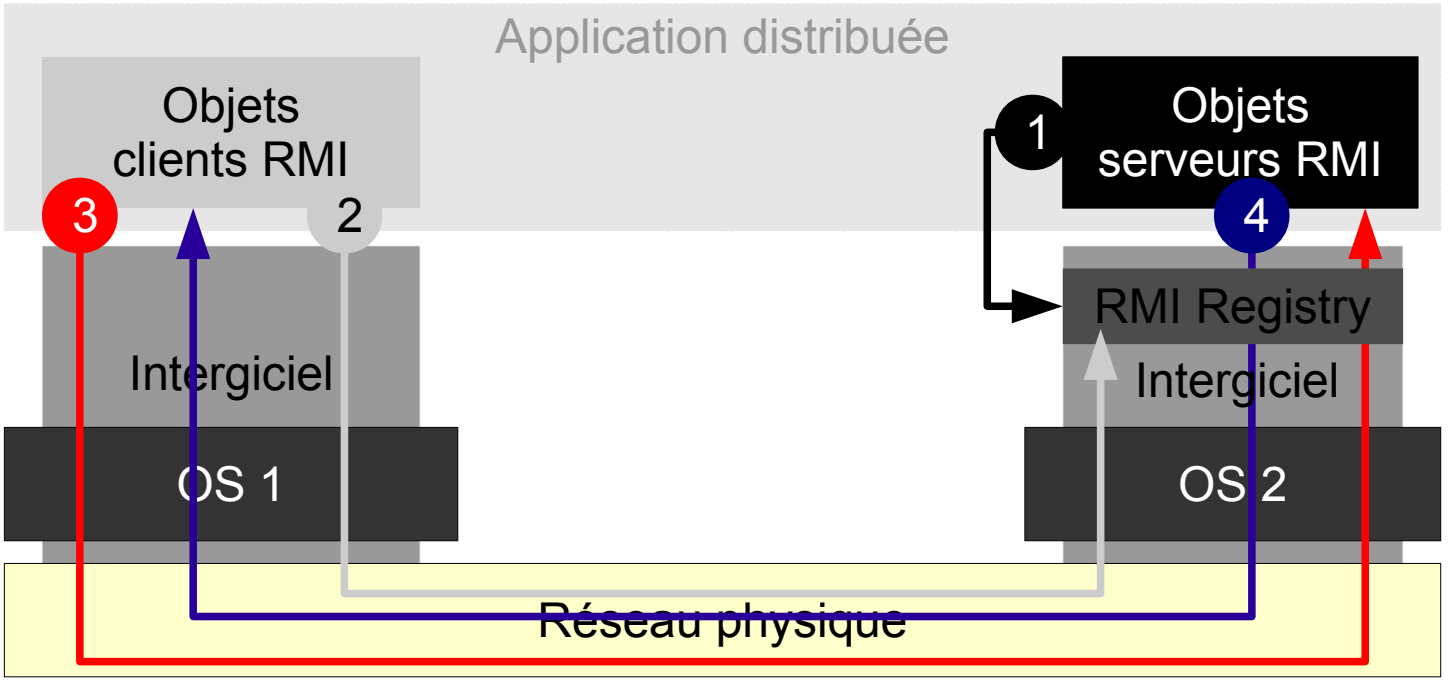
Application distribuée à objets



invocation de méthode distante (masque les opérations de l'API Socket)

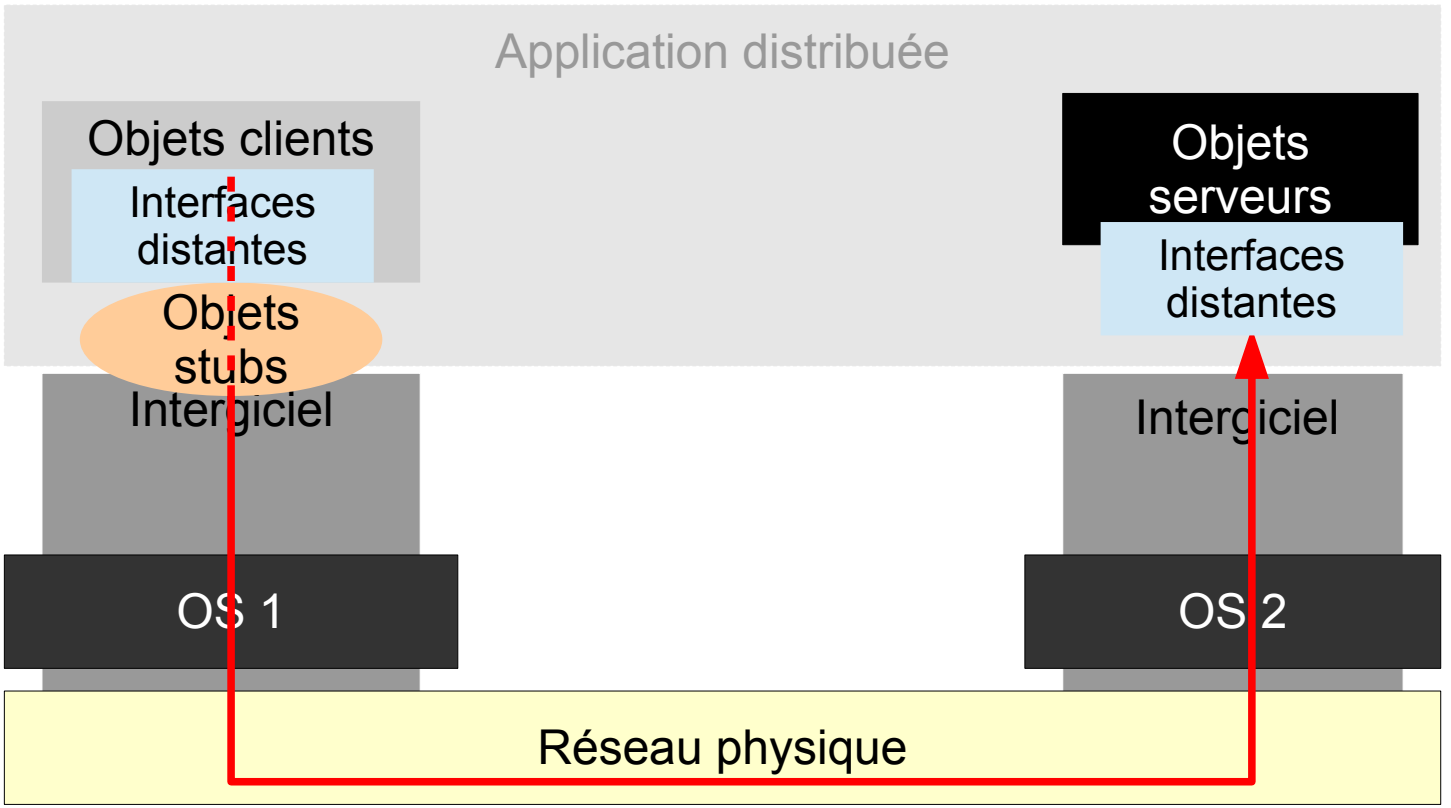
réponse à l'invocation

Application distribuée à objets Java RMI

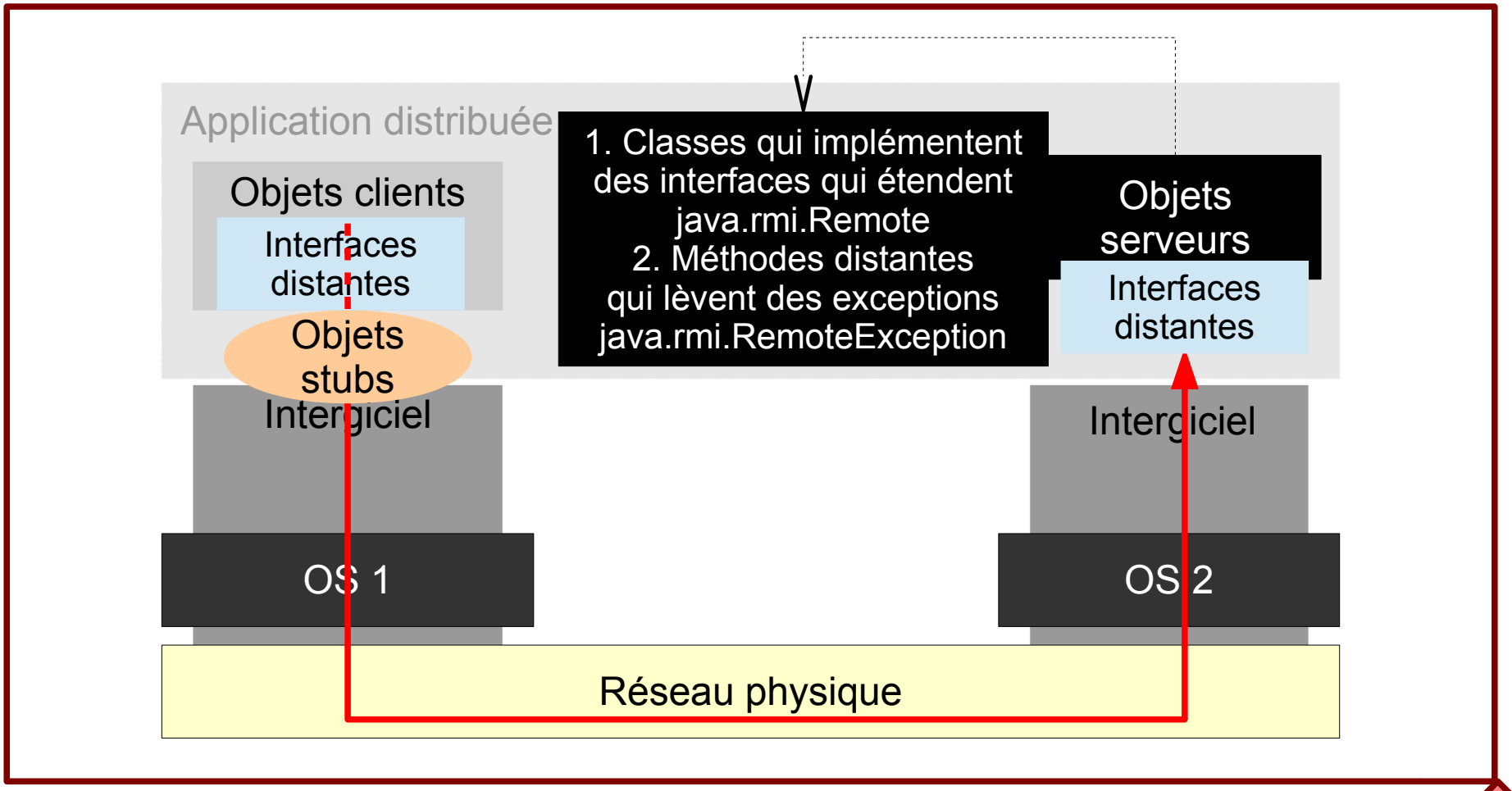


publication (bind) des noms/références
invocation de méthodes distantes
recherche (lookup) de références
réponse à l'invocation

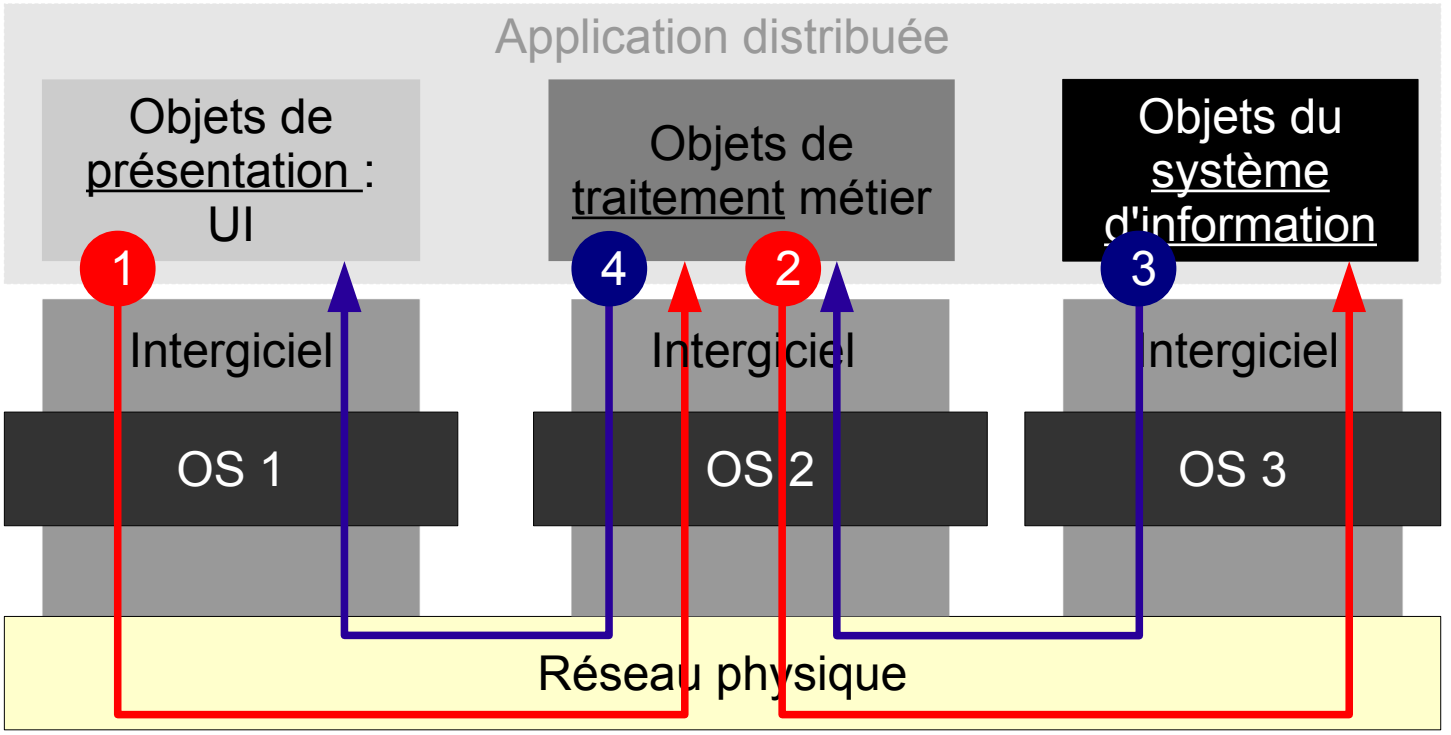
Application distribuée à objets Java RMI



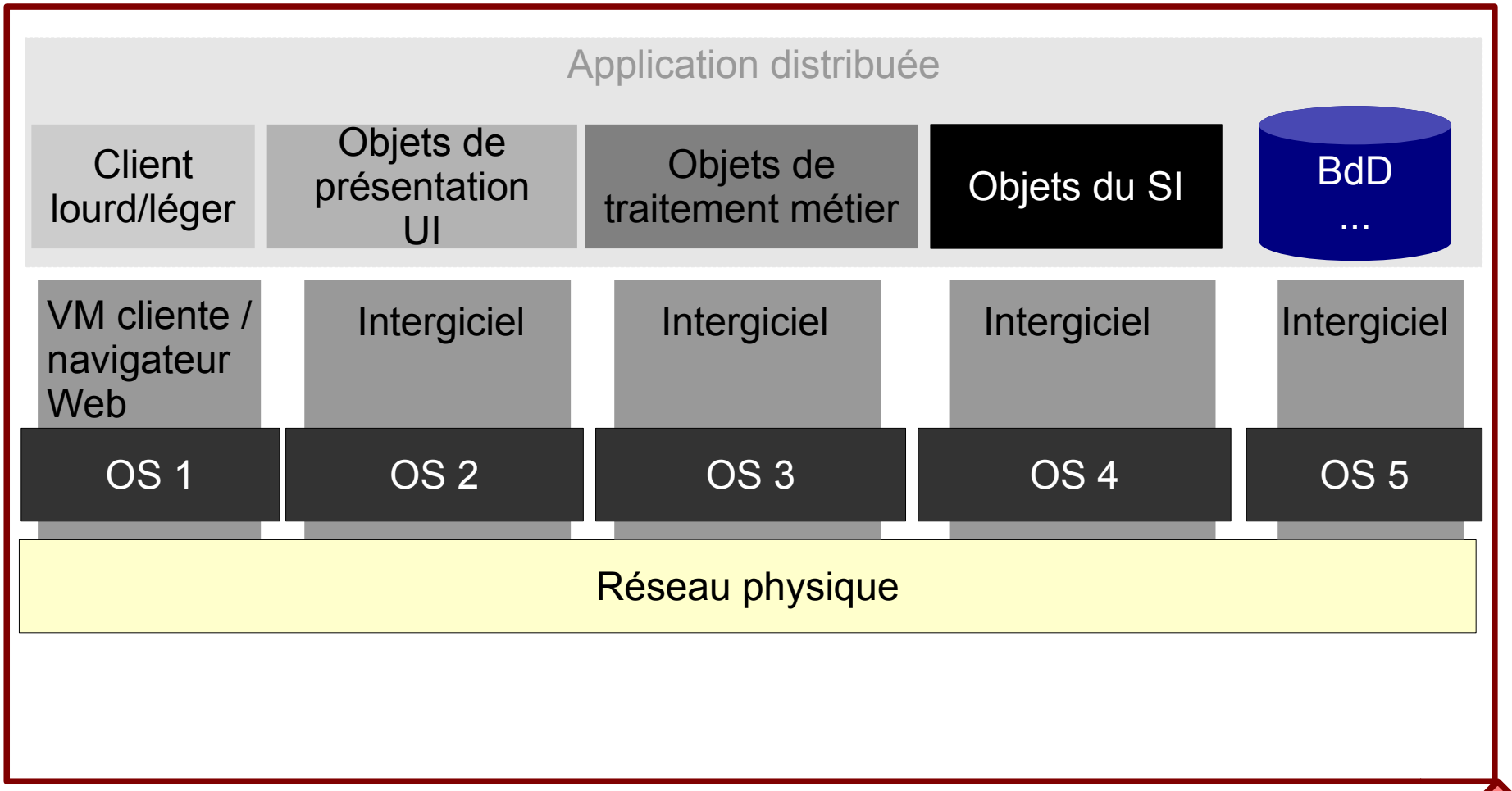
Application distribuée à objets Java RMI



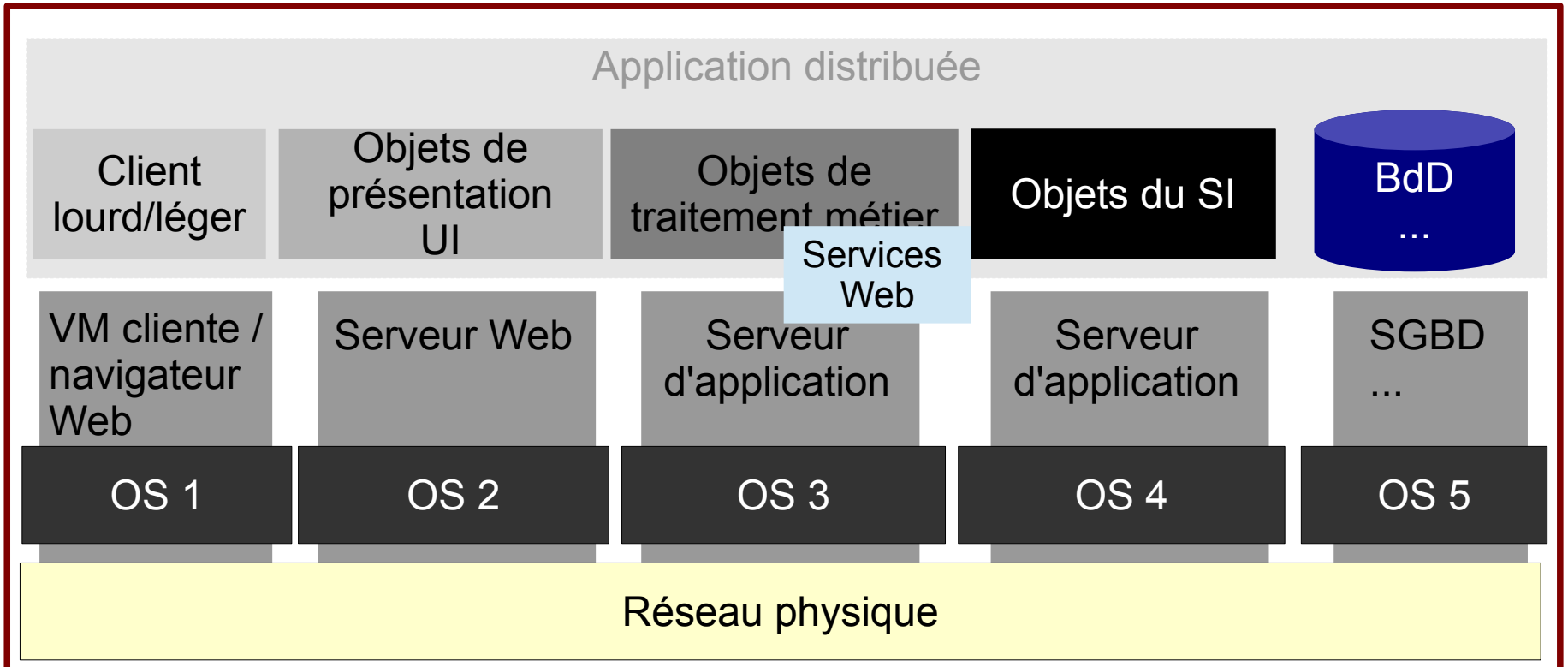
Application distribuée avec une architecture 3/N-Tiers



Application distribuée avec une architecture N-Tiers



Outils intergiciels pour une architecture N-Tiers



Java EE : Plate-forme pour applications N-Tiers à base de composants

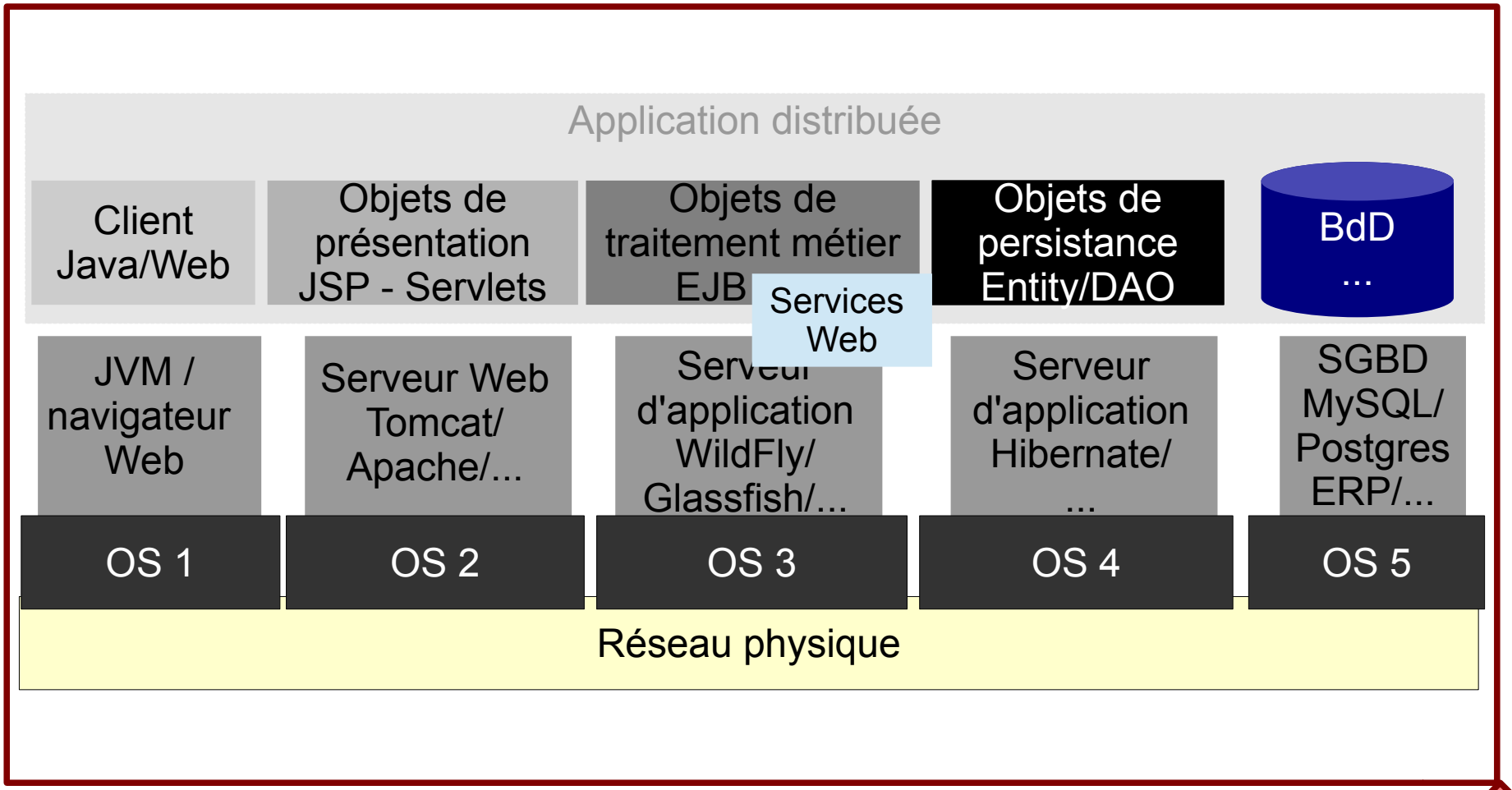
Plan du cours

- Introduction aux architectures multi-niveaux (*N-Tiers*)
- Plate-forme Java EE (*Enterprise Edition*)
- Développement par composants JEE

Plate-forme Java EE

- Anciennement J2EE devenu JEE (suite au passage de Java 2 à 5)
- Une solution globale à base de composants pour les applications N-Tiers en Java
- Spécification proposée par Oracle (2018 : fondation Eclipse Jakarta EE)
- Implémentations sous la forme de frameworks de développement et environnements de déploiement (serveurs d'applications)
- Implémentations actuelles :
  - Implémentation de référence : Glassfish d'Oracle (utilisé en TP)
  - Libres : JBoss (WildFly) de Red Hat/IBM, JOnAS de OW2, ...
  - Propriétaires : WebSphere d'IBM, ...

Java EE et les applications N-Tiers



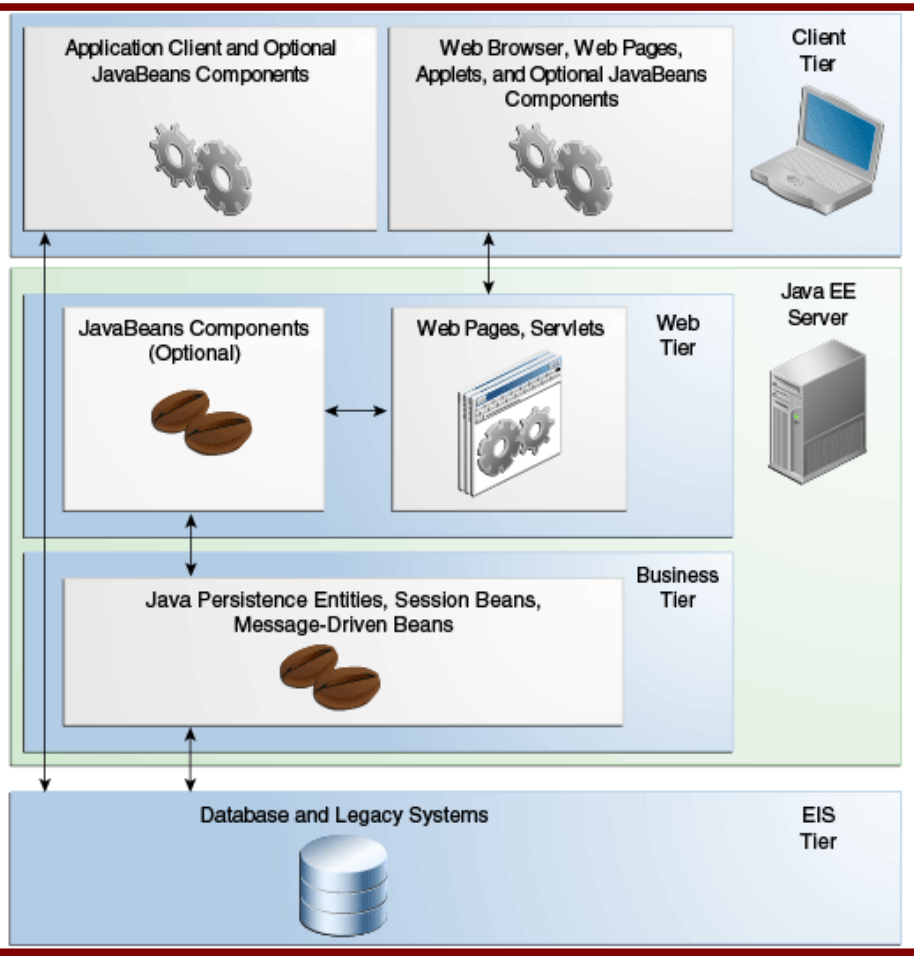
Caractéristiques de la Plate-forme JEE

- Supporte le développement de composants correspondants à chaque niveau (*Tier*) de l'architecture N-Tier
  - Applications clientes : Java (POJO, Applets, JavaBeans, ...), ...
  - Composants Web : HTML, CSS, JS, JSP, Servlets, XML, …
  - Composants métier : Java, EJB, services Web, …

- Fournit un mécanisme d'injection de dépendances, qui satisfait tous les requis des composants, en termes de ressources (autres composants connus du serveur, des connexions aux BdD, …) :
  - Dans le passé (mais toujours possible) : configuration en XML
  - Actuellement : simples annotations dans le code

- Séparation entre aspects fonctionnels et non-fonctionnels : développeurs se focalisent sur la logique métier→composants réutilisables

Communication entre les niveaux

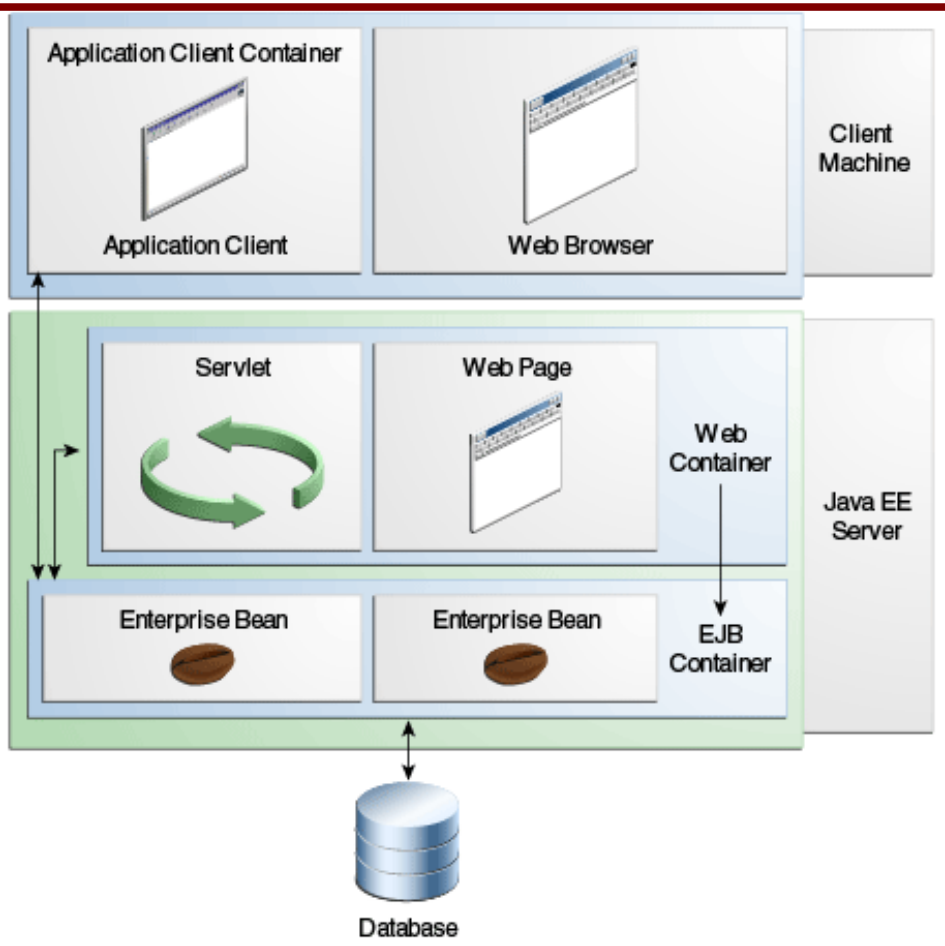


Site Web d'Oracle
https://docs.oracle.com/javaee/7/tutorial

Containers Java EE

- Entités logiques qui font partie du serveur d'application
- Ils fournissent les services non-fonctionnels dont les composants ont besoin : interface avec le serveur
- Services non-fonctionnels : sécurité, gestion des transactions, nommage et répertoires de noms, et gestion de la distribution
- Les composants sont déployés dans ces containers

Containers Java EE - suite

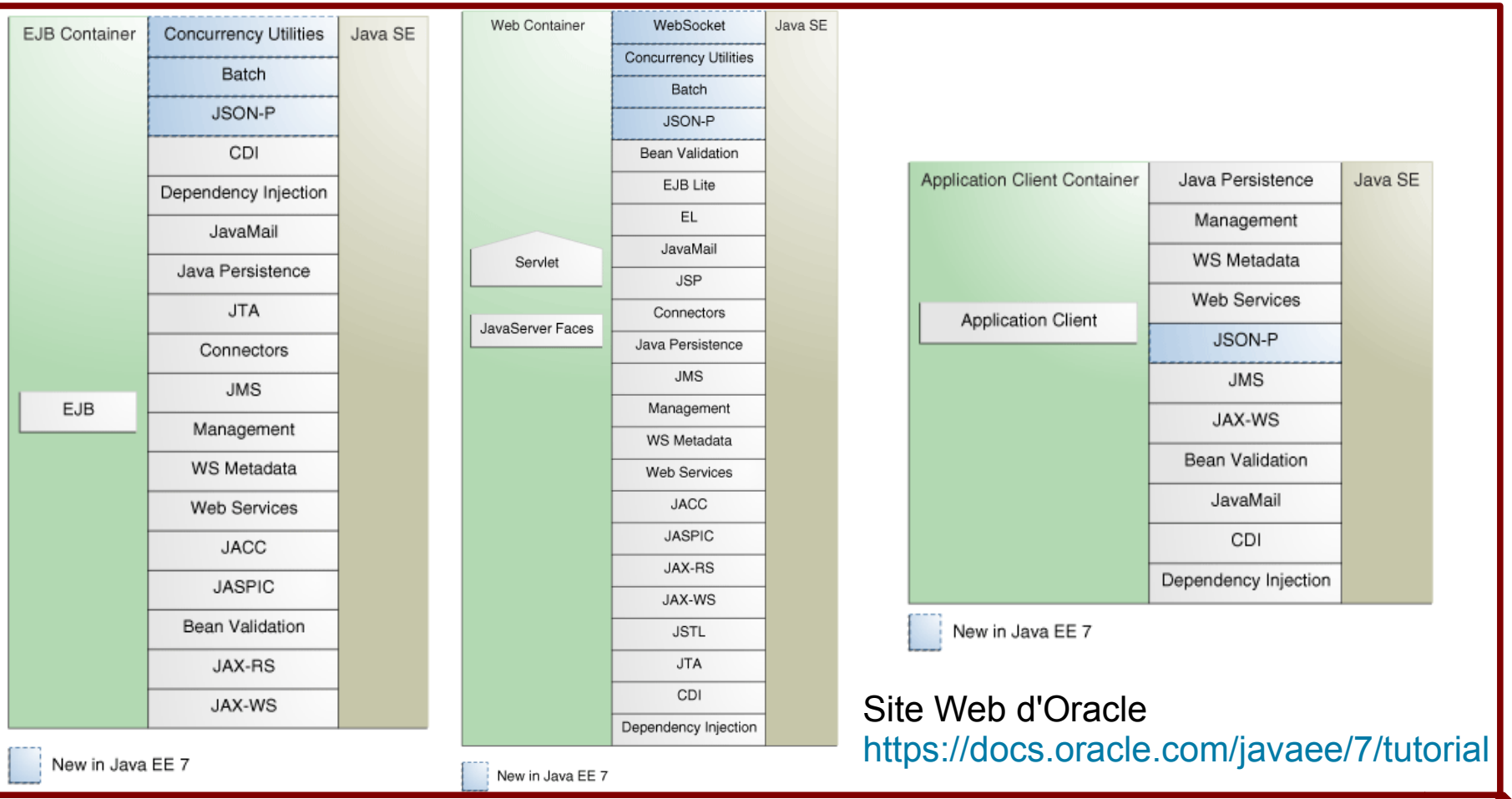


Site Web d'Oracle
https://docs.oracle.com/javaee/7/tutorial

Support des services Web

- La plate-forme Java EE fournit les outils nécessaires pour développer, déployer des services Web et leurs clients
- Service Web : programme serveur qui utilise des standards ouverts pour la description des interfaces et pour les protocoles de communication, comme WSDL, SOAP, XML, HTTP, …
- JEE masque les détails de ces technologies de bas niveau : favoriser l'interopérabilité (conversion en XML transparente, ...)

APIs Java EE



Site Web d'Oracle
https://docs.oracle.com/javaee/7/tutorial

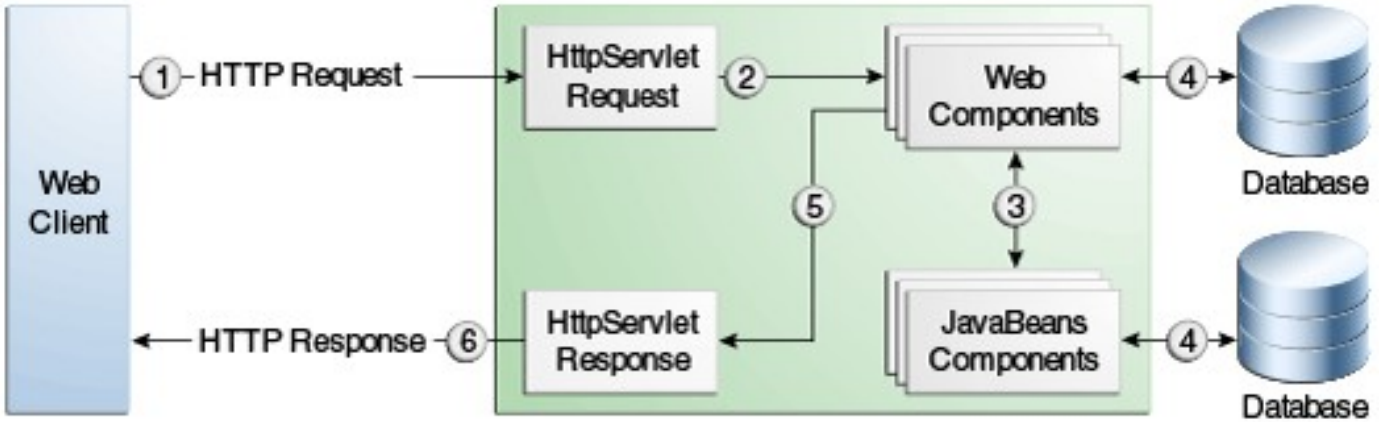
Plan du cours

- Introduction aux architectures multi-niveaux (*N-Tiers*)
- Plate-forme Java EE (*Enterprise Edition*)
- Développement par composants JEE

Composants Web

- Archive constituée de programmes et ressources Web :
  - Des documents HTML, CSS et JavaScript
  - Des images, vidéos, …
  - Des servlets et programmes JSP
  - Des classes Java, JavaBeans, …
- Servlets et programmes JSP : scripts côté serveur en Java
  - Ils sont exécutés suite à la réception d'une requête HTTP
  - Servlets : classes Java implémentant des méthodes doGet, doPost, ...
  - Programmes JSP (équivalents aux scripts Php) : scripts Java transformés en servlets lors du déploiement
- Exécution gérée par les containers Web (serveur Web : Tomcat)

Composants Web - suite



Site Web d'Oracle
https://docs.oracle.com/javaee/7/tutorial

Contenu des scripts JSP

- Scriptlet : balises <% et %> :
  - instructions Java exécutées pour réaliser des traitements lors de la réception d'une requête HTTP et pendant la fabrication de la réponse HTTP
- Expressions : balises <%= et %>
  - expression dont la valeur, convertie en chaîne, est incluse dans le flot HTML produit dans la réponse HTTP
- Déclarations : balises <%! et %>
  - déclaration de classe, de méthode, d'attribut, etc, utilisables dans les scriptlet et expressions précédentes
- Directives d'inclusion : balises <%@ et %>
  - directive d'inclusion de bibliothèques ou de fichiers
- Commentaire : balises <%-- et --%>

Variables pré-définies et pré-initialisées dans les scripts

- request (HttpServletRequest) : objet correspondant à la requête HTTP (durée de vie = celle d'une seule interaction avec un client)
- response (HttpServletResponse) : objet réponse HTTP
- out (PrintWriter) : objet utilisé pour écrire dans le flot HTML de la réponse HTTP → out.print(…) ;
- session (HttpSession) : objet correspondant à la session (durée de vie = celle de la session avec un client)
- application (ServletContext) : objet correspondant à l'application (durée de vie = celle de l'application)
- ...

Exemple de script JSP

```
<html> <head> <title>Converter</title> </head>
<body>
  <h1><center>Converter</center></h1> <hr>
  <p>Enter an amount to convert:</p>
  <form method="get">
    <input type="text" name="amount" size="25"> <br>
    <input type="submit" value="Submit"><input type="reset" value="Reset">
  </form>
  <% String amount = request.getParameter("amount");
    if ( amount != null && amount.length() > 0 ) {
    Double d = new Double (amount); %> <p>
  <%= amount %> dollars =
  <%= converter.dollarToYen(d.doubleValue()) %> Yen.</p><p>
  <%= amount %> Yen = <%= converter.yenToEuro(d.doubleValue()) %> Euro.
  </p><% } %>
</body>
</html>
```

Composants EJB

- Composants implémentant la logique métier de l'application
- Composants écrits en Java avec le framework Enterprise JavaBeans (EJB)
- Quand utiliser des composants EJB ?
  - Application doit passer à l'échelle (nombre grandissant d'utilisateurs)
  - Intégrité des données avec les transactions
  - Application peut avoir des utilisateurs variés

Enterprise JavaBeans

- La spécification EJB a été développée par IBM en 1997 puis adoptée par Sun Microsystems (racheté depuis par Oracle) en 1999
- Versions majeures :
  EJB 1.0 (1998), EJB 2.0 (2001), puis EJB 3.0 (2006)
- La version en vigueur est la 3.2 (mai 2013)

Sortes de composants EJB

- Il existe deux sortes de composants EJB
  - Les *session beans*
  - Les *message-driven beans (MDB)*
- A l'origine, il y avait une troisième sorte de composants : entity beans (devenus de simples classes pour POJO)

Session Beans

- Un session bean encapsule une logique métier pouvant être invoquée par un programme client (local ou distant)

- Il existe trois sortes de session beans :
  - Stateful : conserve un état conversationnel avec un client (dans les variables d'instances de la classe qui l'implémente)
  - Stateless : ne conserve pas d'état
    - Solution efficace : le serveur gère un pool de beans
    - Bean implémente un service Web
  - Singleton : bean instancié une fois pour toute l'application
    - Gère un état partagé par plusieurs clients
    - S'occupe de l'initialisation ou du « nettoyage » (libération des ressources utilisées par) l'application

Développement d'un session bean

- Écrire l'interface du composant : une interface Java standard

```
public interface Converter {
    public double toCurrency(double amount, String currency);
    public Map<Monnaie,Double> toOtherCurrencies(double amount);
}
```

- Pour une interface locale, annoter l'interface par @Local (annotation par défaut) javax.ejb.Local
- Pour une interface distante, annoter l'interface par @Remote
- Dans ce cas, le composant qui l'implémente est un bean dont les méthodes peuvent être invoquées à distance (dans d'autres JVM : autres serveurs, simples JVM ou machines physiques)

Développement d'un session bean - suite

- Écrire la classe du composant : une classe Java ordinaire
  - Mettre une annotation @Session : pour bean sans interface
  - Ajouter une annotation @Stateless pour un bean sans état

```
@Stateless
public class ConverterBean implements Converter {
      ...
}
```

- Pour un bean stateful, remplacer l'annotation @Stateless par @Stateful

- Pour un composant distribué, ajouter l'annotation : @Remote(Converter.class)

- Pour un composant local, on met l'annotation @Local

Accès à un bean

- L'obtention d'une référence vers un bean peut se faire de deux façons :
  - En profitant de l'injection de dépendances fournie par le container EJB :
    - façon la plus simple
    - mais ça ne marche que pour les composants déployés dans un serveur Java EE
  - En utilisant le service de répertoire de noms (JNDI : Java Naming and Directory Interface) fourni par le container :
    - solution qui marche pour n'importe quel client (Java SE, ...)

L'accès à un bean via JNDI

- Lors du déploiement d'un composant EJB, les beans qui y sont définis s'enregistrent auprès d'un service de répertoire JNDI

- Le client d'un bean recherche (fait un lookup) d'un bean en utilisant trois espaces de noms JNDI possibles :
  - java:global[/application name]/module name /enterprise bean name[/interface name]
    pour rechercher un bean distant
  - java:module/enterprise bean name/[interface name]
    pour rechercher un bean qui se trouve dans le même composant
  - java:app[/module name]/enterprise bean name [/interface name]
    pour rechercher un bean dans la même application EAR (Enterprise ARchive)

- JAR : archive d'un composant EJB, WAR : archive d'un composant Web et EAR : archive d'une application (ensemble de JAR et WAR)

L'accès à un bean via JNDI - suite

- Invoquer la méthode lookup avec le nom JNDI du bean
- Exemple :

```
InitialContext context = new InitialContext();
Converter converter = (Converter) context.lookup(
        "java:global/Converter/Converter-ejb/ConverterBean");
```

- Ou :

```
InitialContext.doLookup("java:...");
```

L'accès à un bean via l'injection de dépendances

- Il suffit d'annoter un attribut de la classe du bean par l'annotation : @EJB (javax.ejb.EJB)

- Exemple :

```
@EJB
Converter converter;
```

Le container recherchera un bean de type Converter et affectera sa référence à l'attribut converter

- A partir d'un script JSP, utiliser la balise
  <jsp:useBean id="converter" scope="session" class="conv.Converter" />

Message-Driven Beans (MDB)

- Composants permettant aux applications JEE de s'exécuter en partie (en traitant des messages) de façon asynchrone
- Ils sont utilisés pour ne pas obliger le serveur de traiter des réceptions de messages bloquantes
- Ils agissent le plus souvent comme écouteurs de messages de type JMS (Java Messaging Service)
- Ces messages peuvent être envoyées par n'importe quel autre composant JEE ou programme, en général
- Les MDB ne maintiennent pas d'état conversationnel avec un client (ils sont gérés sous la forme de pool de beans)

Caractéristiques des Message-Driven Beans

- Les MDB n'ont pas d'interfaces utilisables directement par des programmes clients
- Ils sont exécutés de façon automatique à la réception d'un message : leur méthode callback onMessage(…) est invoquée
- Le message est reçu en paramètre de la méthode onMessage
- Ils écoutent une destination de messages : une ressource gérée par le container EJB
- Les clients envoient des messages vers cette ressource

Implémentation de Message-Driven Beans

- Ils sont implémentés par une classe unique (classe du bean)

```
import javax.ejb.* ; import javax.jms.*;
@MessageDriven(mappedName = "jms/MailContentQueue", activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationType",
    propertyValue = "javax.jms.Queue")})
public class MailerMDB implements MessageListener {
    @EJB
    Converter converter;
    public MailerMDB() {
    }
    @Override
    public void onMessage(javax.jms.Message message) {
        try {
            if (message instanceof TextMessage) { … }
        }
        catch (JMSException ex) {ex.printStackTrace();}
    … }
}
```

Envoi de message à un MDB

- Utiliser JNDI pour obtenir une référence vers la destination du message (une file de message : *message queue*)
- Ensuite déposer le message en utilisant l'API JMS
- Exemple :

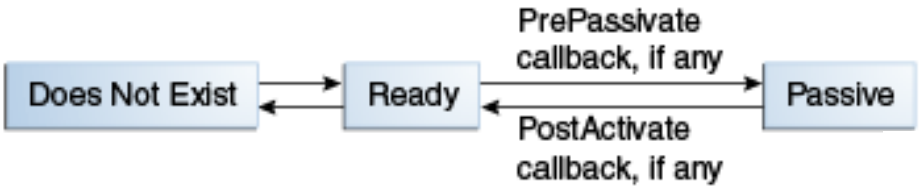
```
Context jndiContext = new InitialContext();
javax.jms.ConnectionFactory connectionFactory =
(QueueConnectionFactory)jndiContext.lookup(
"jms/MailContentQueueFactory");
Connection connection = connectionFactory.createConnection();
Session sessionQ = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
TextMessage message = sessionQ.createTextMessage();
String text = "Hello World !!!"; message.setText(text);
javax.jms.Queue queue = (javax.jms.Queue)
jndiContext.lookup("jms/MailContentQueue");
MessageProducer messageProducer=sessionQ.createProducer(queue);
messageProducer.send(message);
```

Cycle de vie des composants EJB

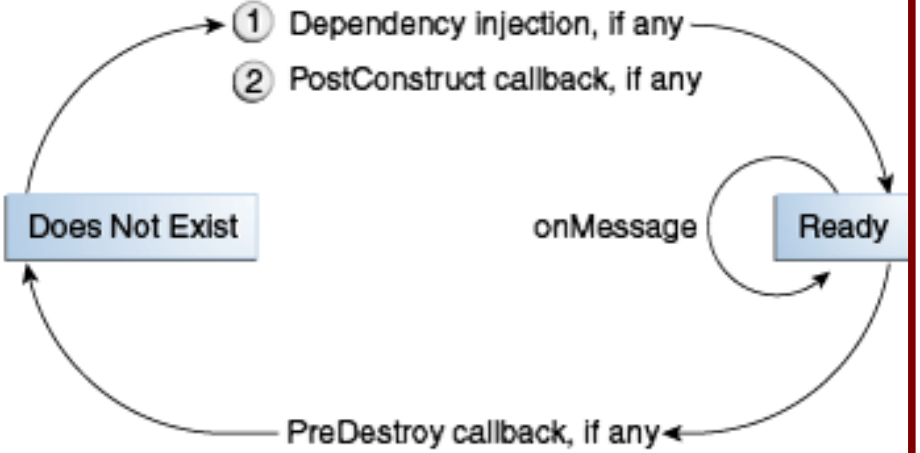
Site Web d'Oracle
https://docs.oracle.com/javaee/7/tutorial

Session Bean Stateful

1. Create
2. Dependency injection, if any
3. PostConstruct callback, if any
4. Init method, or ejbCreate<METHOD>, if any

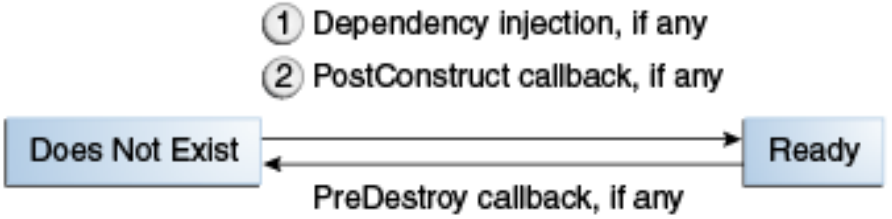


1. Remove
2. PreDestroy callback, if any



Message-Drive Bean

Session Bean Stateless



EJB et services Web

- Possibilités : 1) d'invoquer les opérations de services Web (SW) depuis un bean et 2) de publier un bean comme un SW
- WSDL (Web Services Description Language) : langage de description des interfaces de services Web
  - Il est basé sur XML
  - Interface de service Web :
    - Nom et localisation du service sur le Web
    - Ensemble d'opérations avec leurs signatures (types, ...)
    - Moyens d'utiliser le service Web : quel protocole, ...
  - Implémentation du service Web : dans n'importe quel langage
- SOAP (Simple Object Access Protocol) : protocole de communication (requête/réponse) entre clients et services Web
  - Il est basé sur XML
  - Il est souvent utilisé au dessus de HTTP

Interagir avec des services Web

- Générer la classe du proxy du SW et les classes JavaBeans (des types de paramètres …) en utilisant une commande du JDK

  wsimport http://webservices.gama-system.com/exchangerates.asmx?WSDL -d .

  Solution qui marche avec n'importe quel client Java (EE ou SE)

- Créer le proxy du service Web à partir des classes générés
- Invoquer les méthodes métiers

Produire des services Web

- Un bean session sans état peut être publié comme service Web
- Son interface sera considérée comme l'interface du service Web et sa classe, l'implémentation du SW
- Il suffit d' :
  - Annoter la classe du bean avec @WebService (javax.jws)
  - Annoter les méthodes publiées avec @WebMethod
- Il existe une méthode plus automatisée : New > Web Service > Create Web Service from Existing Session Bean
- Lors du déploiement, le serveur d'application génère le document WSDL qui décrit le service Web (et fournit une interface Web de test des opérations du service)

Services Web REST

- Il existe une forme de services Web plus légère : services Web REST (REpresentational State Transfer)
- Ce type de services s'appuient principalement sur HTTP (méthodes GET, POST, …) et rarement sur SOAP
- Communication entre client et service Web plus légère (pour les messages, pas d'enveloppe SOAP dans les req/rep HTTP)
- Invocation des opérations du service=accès à des ressources Web
- Exemple :

http://currencies.apps.grandtrunk.net/getlatest/<fromcode>/<tocode>
http://currencies.apps.grandtrunk.net/getlatest/EUR/USD

Quelques références

- **Tutoriel Java EE. Site Web d'Oracle :**
  https://docs.oracle.com/javaee/7/tutorial/
- **Introduction to Java Platform, Enterprise Edition 7.**
  Oracle White Paper. Juin 2013.
  http://www.oracle.com/technetwork/java/javaee/javaee7-whitepaper-1956203.pdf

Questions