OSGi ™

Programmation Java

- Chargement de classes
  - Recherche de classes
  - Chargement dynamique
- Programmation orientée services
  - Framework
  - Contrat

Chargement de classes

- Qui ?
- Où ?

Les unités de compilation

- Le code source d'une classe est appelé *unité de compilation.*
- Une classe a un nom qui est composé du nom du package et du nom de la classe
  - java.util.Vector

Les packages définition

- Unité d'organisation des classes
  - Organisation logique : `time.clock.Watch`
  - Organisation physique : `time/clock/Watch.class`
  - Il existe une correspondance directe entre le nom d'une classe et son emplacement dans le système de fichiers
- Espace de **nommage** hiérarchique
  - Description de la hiérarchie : `package time.clock;`
  - Notion de nom complet : `time.clock.Watch`
- Les bibliothèques java sont organisées en packages
  - java.util, java.net, org.objectweb…
  - Deux classes ayant le même nom complet ne peuvent pas s'exécuter en même temps … en théorie

Où ?

Chargement de classe

```
Hello t;            /* La classe n'est pas encore chargée*/
t=new Hello();      /* Le classloader charge la classe en mémoire */
```

- Les outils du système cherchent toutes les classes
  - Compilation et Exécution : typage fort C & E
  - Les classes sont recherchées sur le système de fichiers
  - Les classes standards sont automatiquement trouvées par les outils
  - Pour indiquer l'endroit sur le système de fichiers à partir duquel il faut chercher les classes, on utilise le **classpath**

Le classpath

- Il indique à partir de quel endroit rechercher une classe

```
java -classpath /usr/local/ titi.Toto
```

- La classe titi.Toto et **toutes** les classes lancées à partir de maintenant sont recherchées à partir du répertoire

```
/usr/local /*Résolution physique*/
```

- Il faut donc que la classe soit définie dans le fichier :

```
/usr/local/titi/Toto.class /*Résolution java*/
```

Le classpath : le jar

- Un jar est une archive java
  - Elle permet le regroupement de fichiers dans un seul fichier
- Extension du système de fichiers

```
jar tvf toto.jar
```

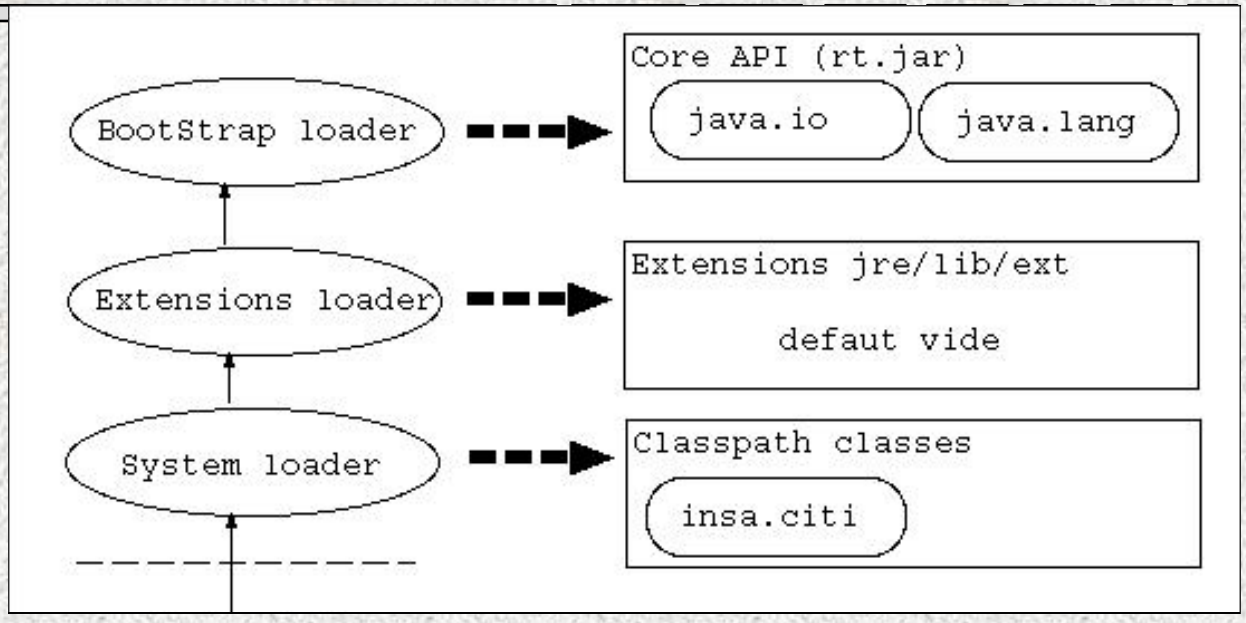
```
tutu/
tutu/ours/
tutu/ours/Grumly.class
```

```
javac ? ???
```

```
vi Test.java
```

```
package test;
import tutu.ours.Grumly;
public class Test {
  Grumly toto=new Grumly();
}
```

Chargement de classes



- Les classes sont recherchées par la machine virtuelle à trois emplacements
- Chaque emplacement est géré par un classloader
- Ces trois classloader sont définis au lancement de la machine virtuelle et sont immuables
- Ils sont organisés hiérarchiquement

Chargement Implicite et Explicite

- Une classe est chargée implicitement par la machine virtuelle à lors de sa première instanciation

```
Hello t;            /* La classe n'est pas encore chargée*/
t=new Hello();      /* Le classloader charge la classe en mémoire */
```

Chargement Explicite

- Le chargement d'une classe passe par les classloaders
- Il est possible de demander au classloader de charger une classe en se fondant sur la chaîne de caractères représentant son type
- Cette caractéristique permet de faire de la programmation semi-réflexive (programmation par les structures du langage)
- Il est possible de manipuler un type sans le posséder à la compilation, il n'est fournit qu'à l'exécution.

```
public static Class forName(String clazz);  /* Fabrique un type à partir de sa chaîne
                                                de description */
public Object newInstance(); /* Fabrique un objet conforme à la classe *
```

Exemple manipulation explicite de type

```
package reflexion;

import java.lang.reflect.Method;
import java.util.Hashtable;

public class Test2 {
  public static void main(String [] arg) throws Exception{
    Class c=Class.forName("java.util.Vector");
    Object o=c.newInstance();
    java.util.Vector v=(java.util.Vector)o;
    v.add("coucou");
    System.out.println("vector"+v+": "+v.size()+" : "+v.get(0));
```

...

Exemple manipulation explicite de type

```
...
   Object o2=c.newInstance();
   Method m1=c.getMethod("add", new Class[]{Object.class});
   m1.invoke(o2, new Object[]{new String("coucou")});
   Method m2=c.getMethod("size", null);
   System.out.println("Je suis un vector de "+m2.invoke(o2, null));
  }
}
```

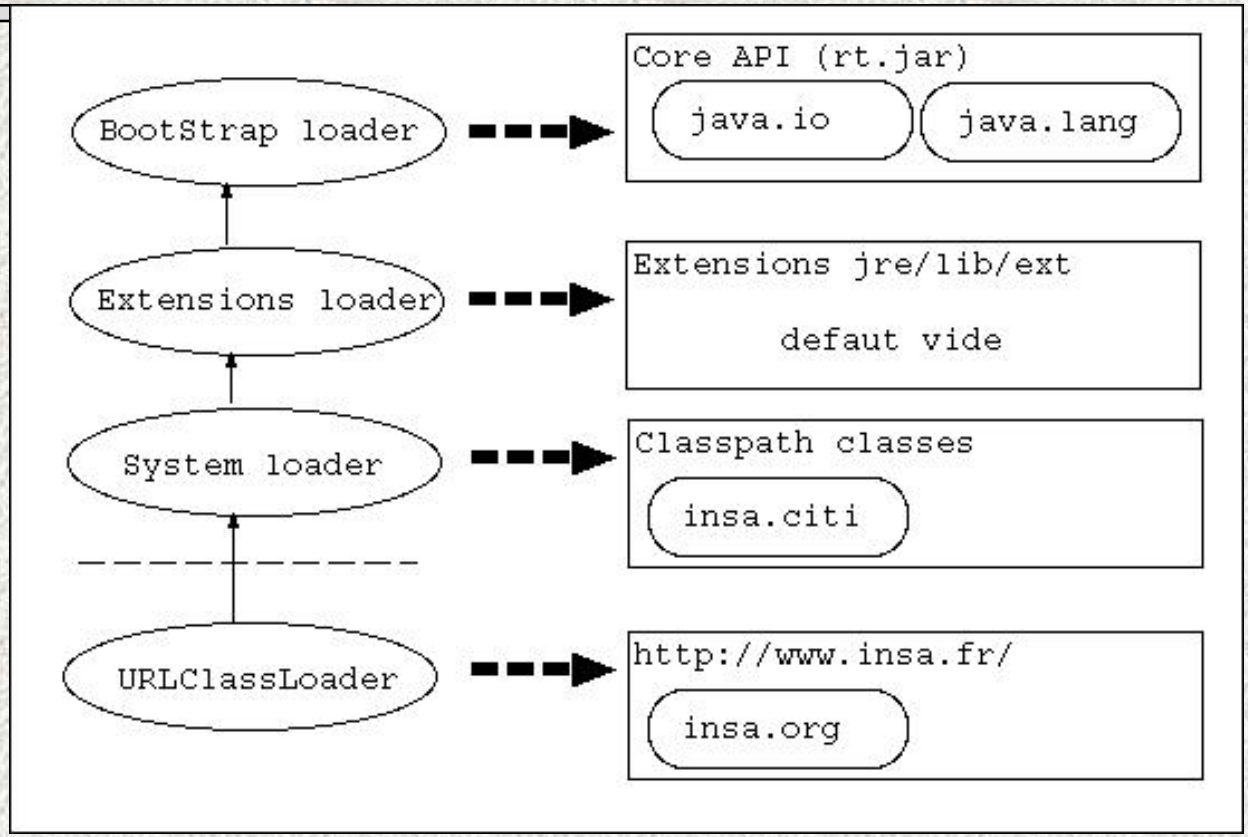
Le chargement explicite permet

- de piloter finement le chargement d une classe java
- de piloter finement le chargement des ressources
- de gérer des versions d'une même classe

- *de Rendre un code client (utilisateur) totalement indépendant du code d'implantation de service*

- Interface ifc=ClassLoader.load("implementation");

Extension des ClassLoader standards



==> Securité !

java -Djava.ext.dirs=myext MaClasse

grant codeBase "file:${java.home}/lib/ext/*"{permission java.security.AllPermission;};

Une classe est donc définie par

- Un nom complet
  - test.toto.Titi
- Et un classloader qui la défini
  - URLClassloader

- Class c={nom, classloader}

- Si une classe n'est pas trouvée dans son classloader, on remonte dans la hiérarchie des classloaders

Evolution de la programmation

- Nous sommes passé de la programmation
  - Client/serveur /* Programmation oo */
  - Client/Interface/Serveur /* Programmation Composant */
  - Client/Usine/Interface/Serveur /* Programmation service */
- Pour rendre un service de plus en plus adaptable
  - Mais de moins en moins simple à programmer

Exemple : La classe Point
Approche classique

```
public class Point {
  private int x;
  private int y;

  public void move (int dx, int dy){
    this.x+=dx;
  }
}
```

```
public class Client {
  public static void main( String [] arg){
    Point p=new Point(4,5);
    p.move(2,3);
  }
}
```

- Programmation OO
  - L'interface est liée à l'implantation
  - Il n'est pas possible de diffuser l'interface sans son implantation
  - Le cycle de vie du client est lié au cycle de vie du serveur

Exemple : La classe Point
Approche composant

```
public interface Point {
  public void move (int dx, int dy);
}
```

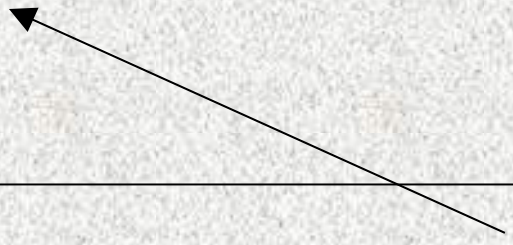
```
public class PointImpl
             implements Point {
  public void move
              (int dx, int dy){
    this.x+=dx;
  }
}
```

```
public class Client {
  public static void
              main(String [] arg){
    Point p=new PointImpl(4,5);
    p.move(2,3);
  }
}
```

- Ok pour la programmation Objet, mais…
  - Le client et le serveur sont liés par compilation
  - La classe d'implantaion du service est référencée dans le client...

Dépendance client/serveur

```
public class Client {
  public static void main(String [] arg){
    Point p=new PointImpl(4,5);
    p.move(2,3);
  }
}
```



- La correction du serveur impose l'arrêt du client
- Client et serveur sont dépendants
- Pour les rendre indépendants il faut un conteneur/intercepteur/fabrique...
- Le modèle est Client/Conteneur/Serveur

Programmation orientée service

- Un service est un composant
  - Interface de définition
  - Classe d'implantation
- Mais dont l'association Interface/Implantation est dynamique
  - Elle est fournie à l'exécution par un médiateur
  - Elle peut être remise en cause à tout moment
  - Elle isole entièrement le client du serveur **à l'interface de définition près**

Un serveur de point

```
public class ServeurDeClassePoint{
  public static Point createClassPoint( ){
    URL[] urls=new URL[]{new URL("file:resource/")};
    ClassLoader cl=new URLClassLoader(urls);
    Point pt=(Point)cl.loadClass("tp1.PointImpl").newInstance();
    return pt;
  }
}
```

- Ne repose pas sur la classe, mais
- sur le nom de la classe (String)
- Il repose sur un « ClassLoader » qui charge la classe de définition
- ! A mettre en œuvre

2 améliorations classiques

```
package tp1;
public class ServeurDeClassPoint{
  static ClassLoader c1;
  static Class ptClass;
  static String url;
  static String className;

  public static Point createClassPoint(Point tmp)
    throws Exception{
    if (ptClass==null){reloadImpl();}
    Point newPt=(Point)ptClass.newInstance();
    if (tmp!=null){
      newPt.initialize(tmp);
    }
    return newPt;
  }
```

```
public static void reloadImpl()
    throws Exception{
    setProperties();
    URL[] urls=new URL[]{new URL(url)};
    c1=new URLClassLoader(urls);
    ptClass=c1.loadClass(className);
  }
  private static void setProperties()
    throws Exception {
    URL u=new URL("file:params");
    BufferedReader is=new BufferedReader
      (new
        InputStreamReader(u.openStream()));
    url=is.readLine();
    className=is.readLine();
    System.out.println
      ("Les paramètres sont : "+url+className);
  }
}
```

Le client

```
package tp1;
public class Client{
  public static void main(String [] arg) throws Exception{
    while (true){
      Point p=ServeurDeClassPoint.createClassPoint(null);
      System.out.println(p);
      p.move(4,4);
      System.out.println(p);
      System.in.read();
      ServeurDeClassPoint.reloadImpl();
    }
  }
}
```

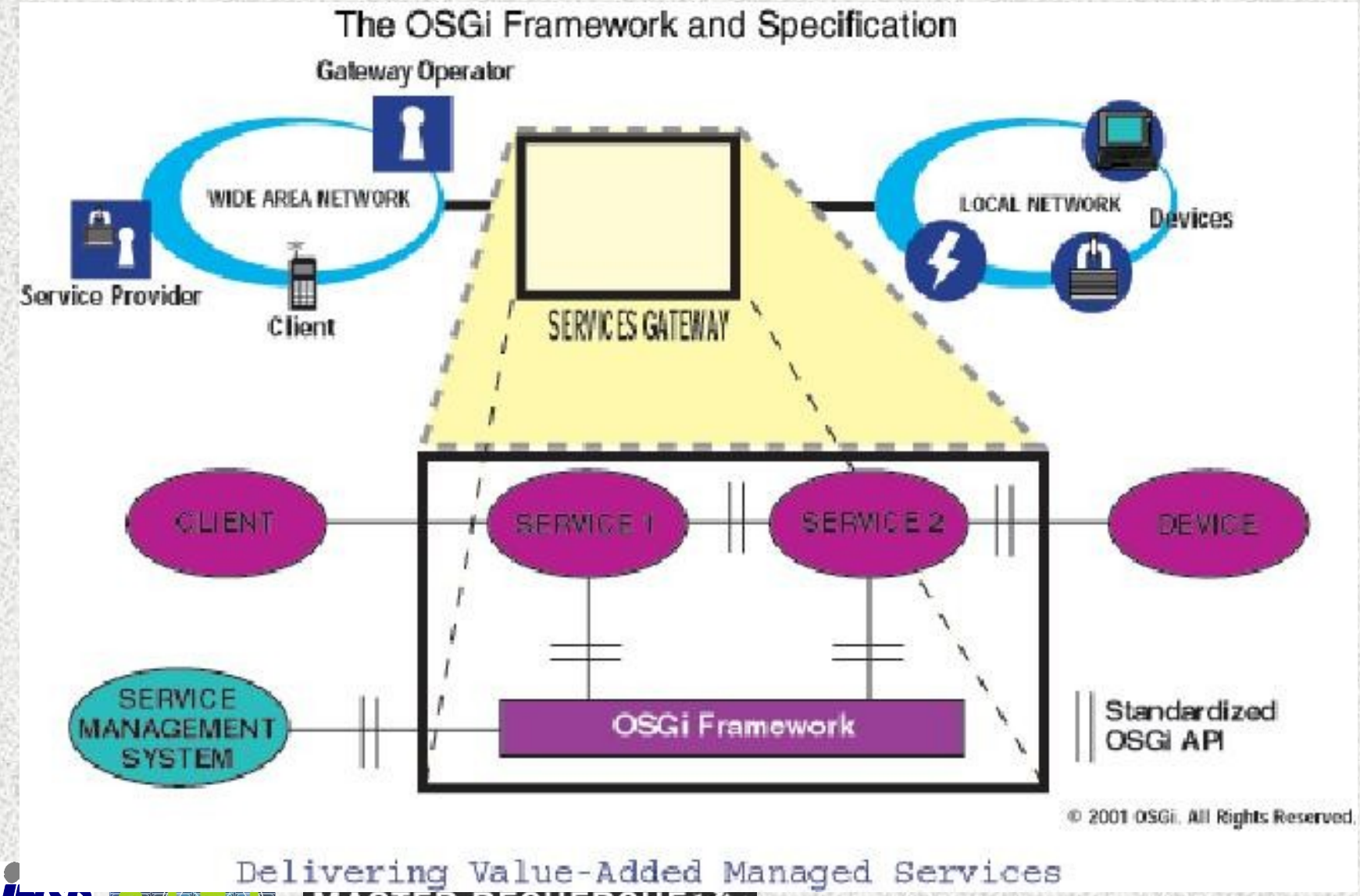
Le client

- 1. Ne doit pas référencer l'implantation
- 2. Les versions d'implantation et d'interface sont les mêmes
- 3. Il faut pouvoir transmettre l'ancien état
- 4. Le client doit fournir la nouvelle et détruire l'ancienne référence

La programmation orientée service

- Les services sont spécifiés par une interface
- Ils sont packagés dans le concept de bundle (composant = unité de déploiement = unité de packaging)
- Les clients n'instancient jamais l'implantation…Elle est fournie au run-time par un framework
- Le framework
  - Impose le modèle à composant utilisé
  - Gère le cycle de vie des composants
  - Gère les interactions entre composants
    - Appels directs/Notification

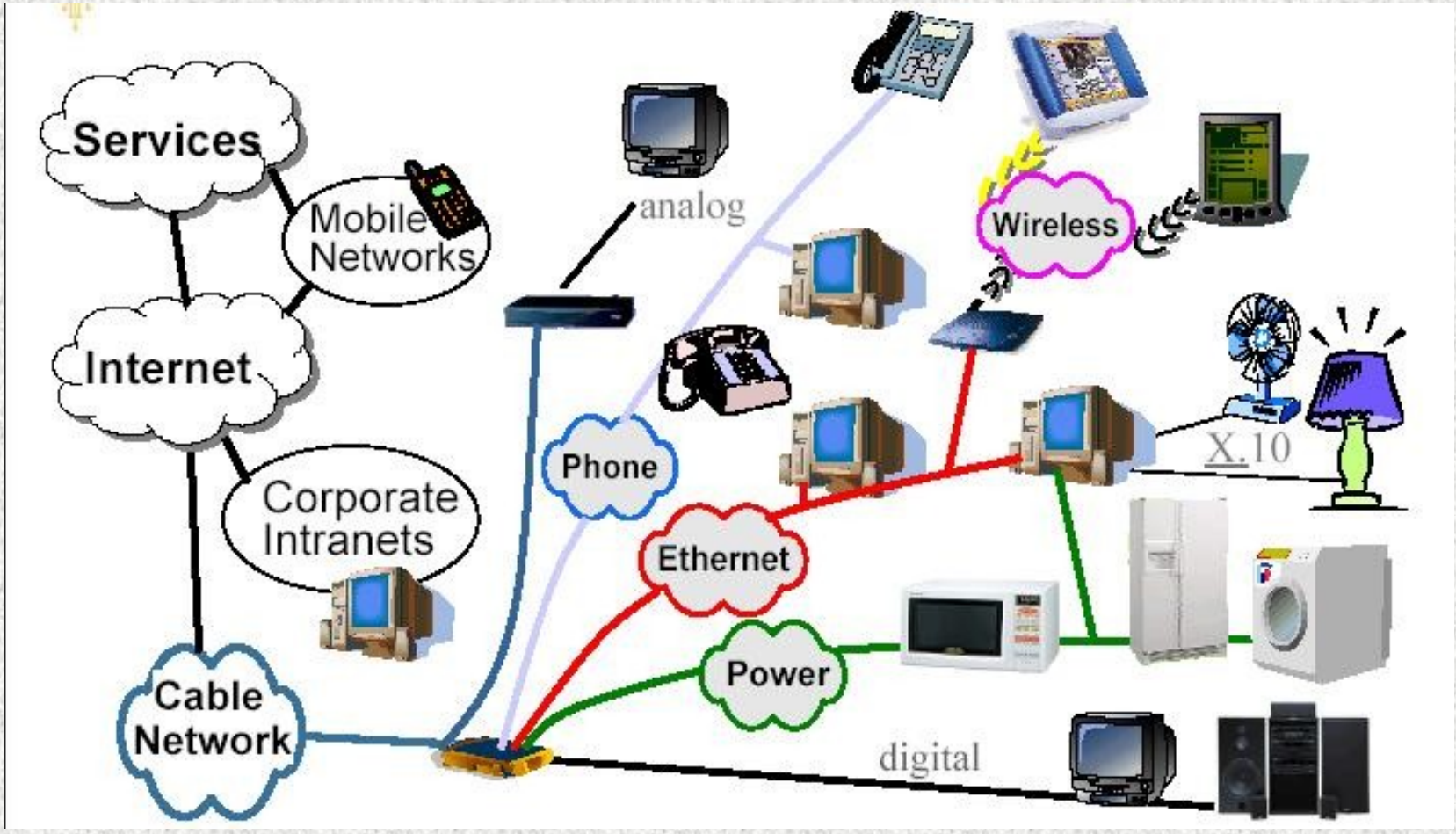
OSGi platform



OSGi

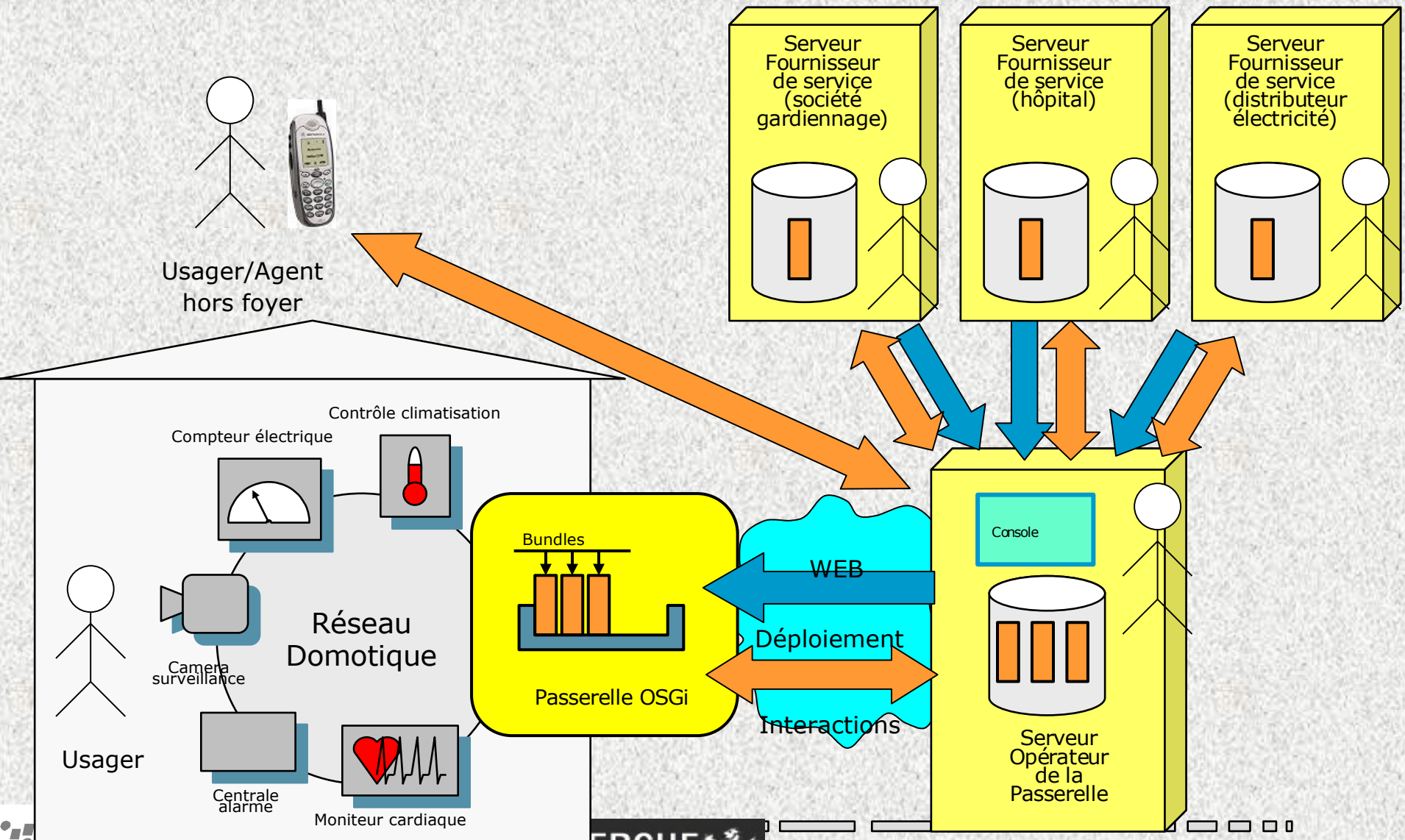
- Présentation générale
  - Gestion des packages
  - Gestion de services
- Projets du citi
- Approche pair-à-pair pour le déploiement de services

Gateway OSGi



Cable Labs CableHome Project - Juillet 2001

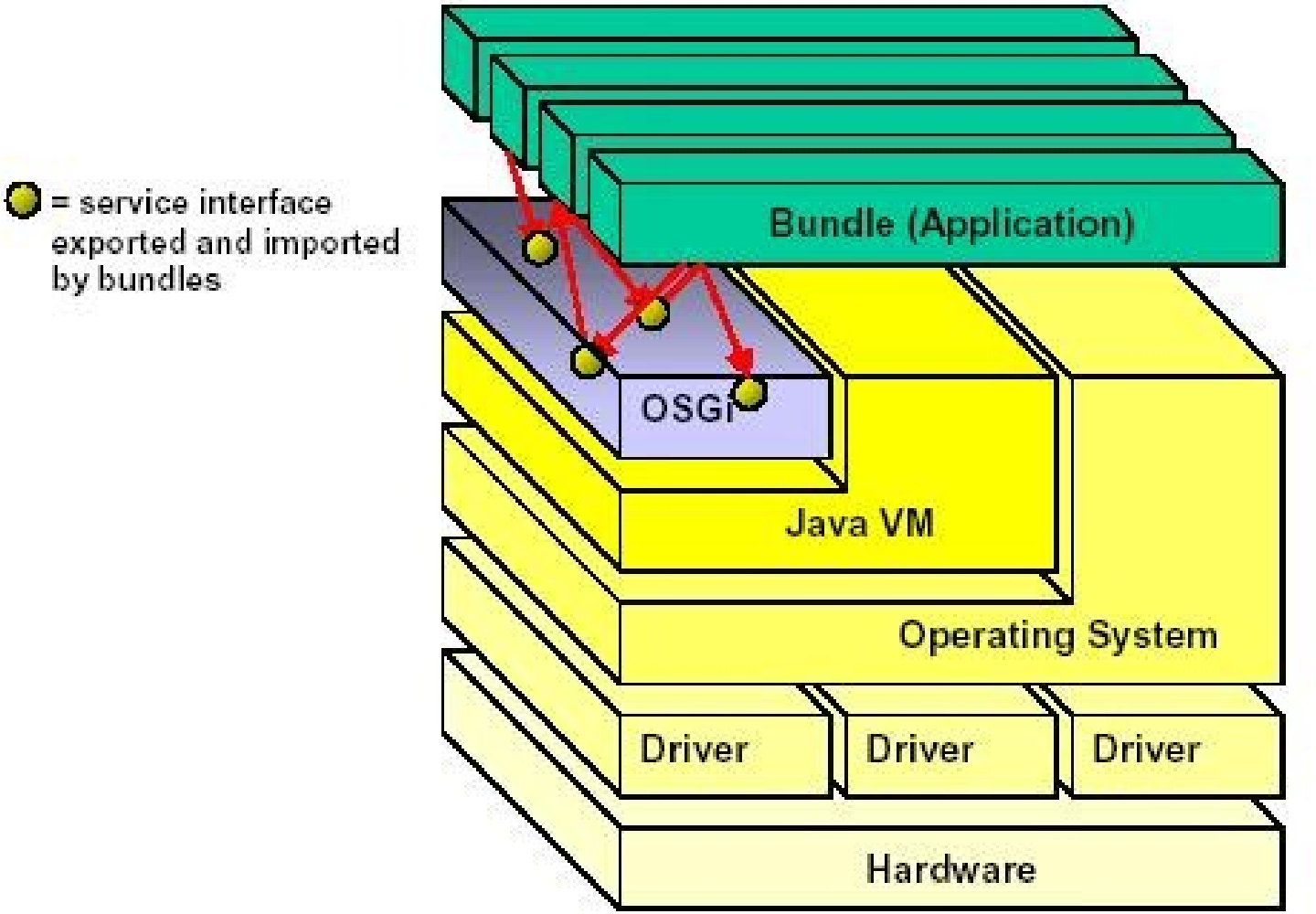
Architecture générale



Motivations

- Chargement/Déchargement de code dynamique
  - Langage Java
- Déploiement dynamique d'applications
  sans interruption de la passerelle
  - Installation, Lancement, Mise à jour, Arrêt, Retrait
- Résolution des dépendances versionnées de code
- Programmation orientée services
- Cible également des systèmes à mémoire restreinte

OSGi framework

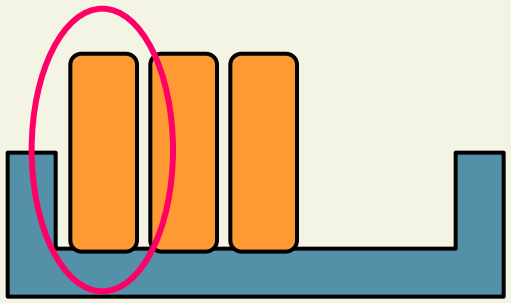


Bundles (fagot, liasse)

Bundle

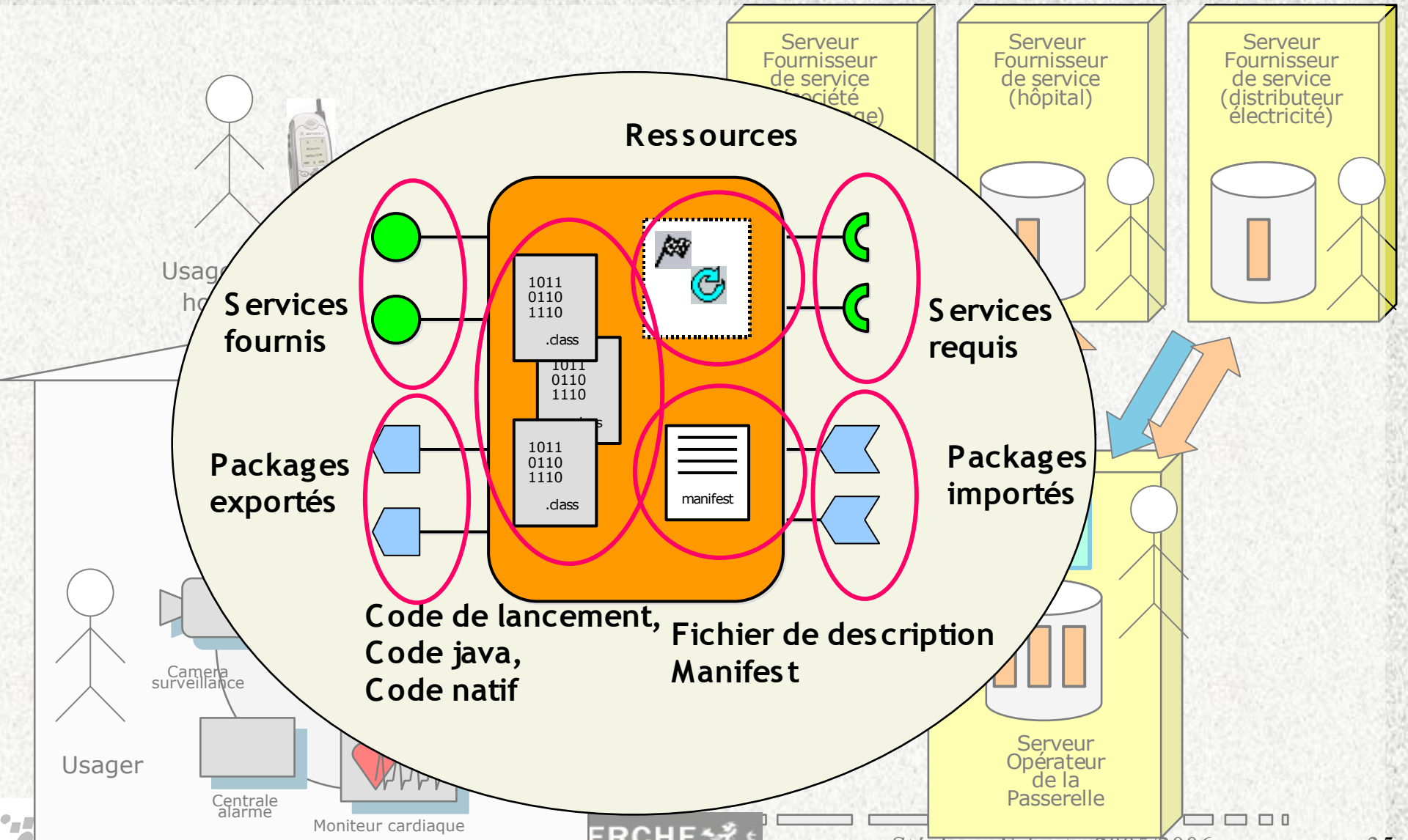
Unité fonctionnelle (offre des services)

Unité de livraison et de déploiement sous forme d'un jar



Une passerelle OSGi

Structure d'un bundle



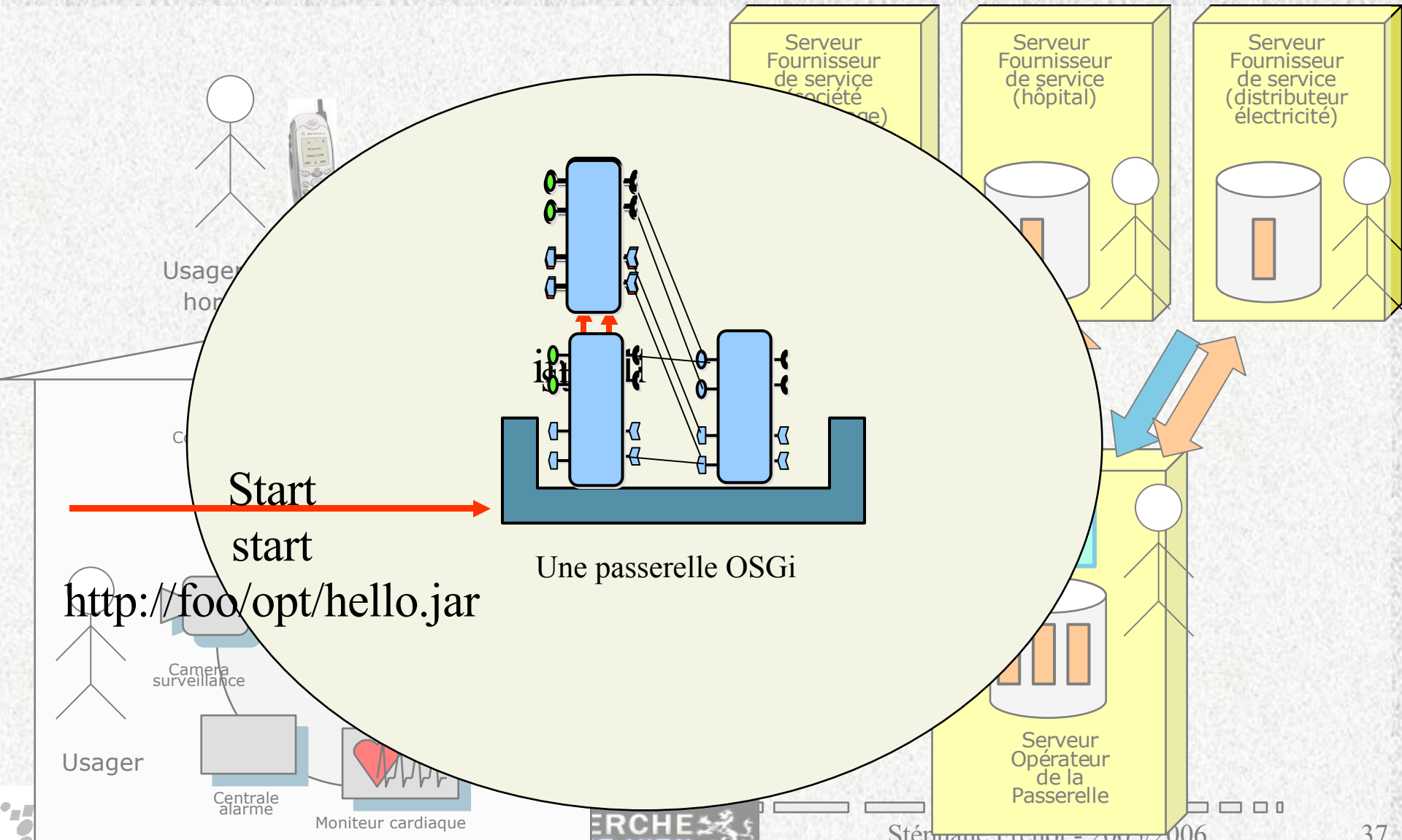
Le bundle hello

```
package hello;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class HelloWorld implements BundleActivator {
  public void start(BundleContext bc){
    System.out.println("Bonjour");
  }

  public void stop(BundleContext bc){
    System.out.println("Au revoir");
  }
}
```

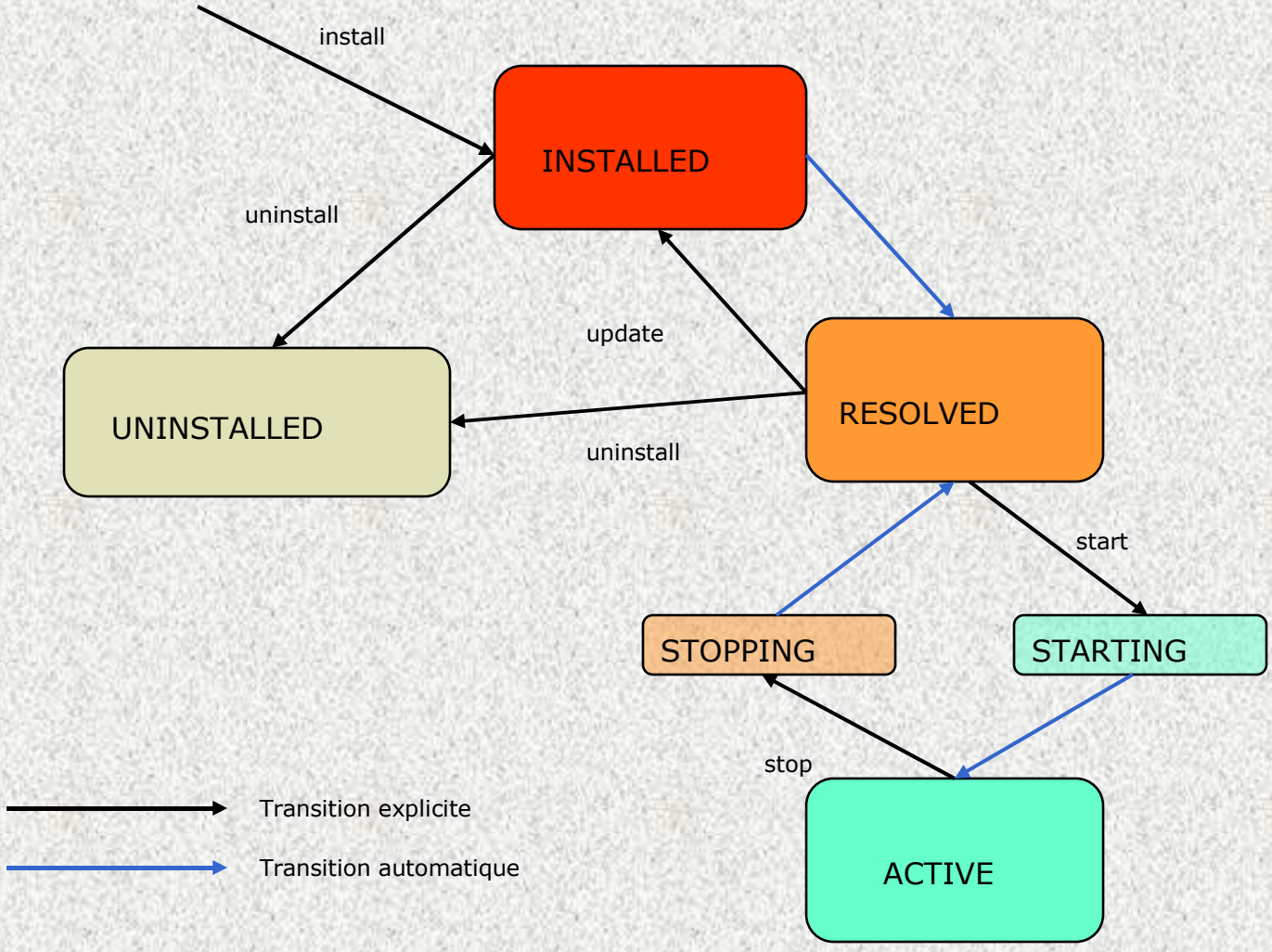
```
Bundle-Name: Hello World
Bundle-Description: The simple bundle
Bundle-Activator: hello.HelloWorld
```

```
200 Tue Jun 15 16:20:00 CEST 2004 META-INF/MANIFEST.MF
723 Tue Jun 15 16:20:00 CEST 2004 hello/HelloWorld.class
```

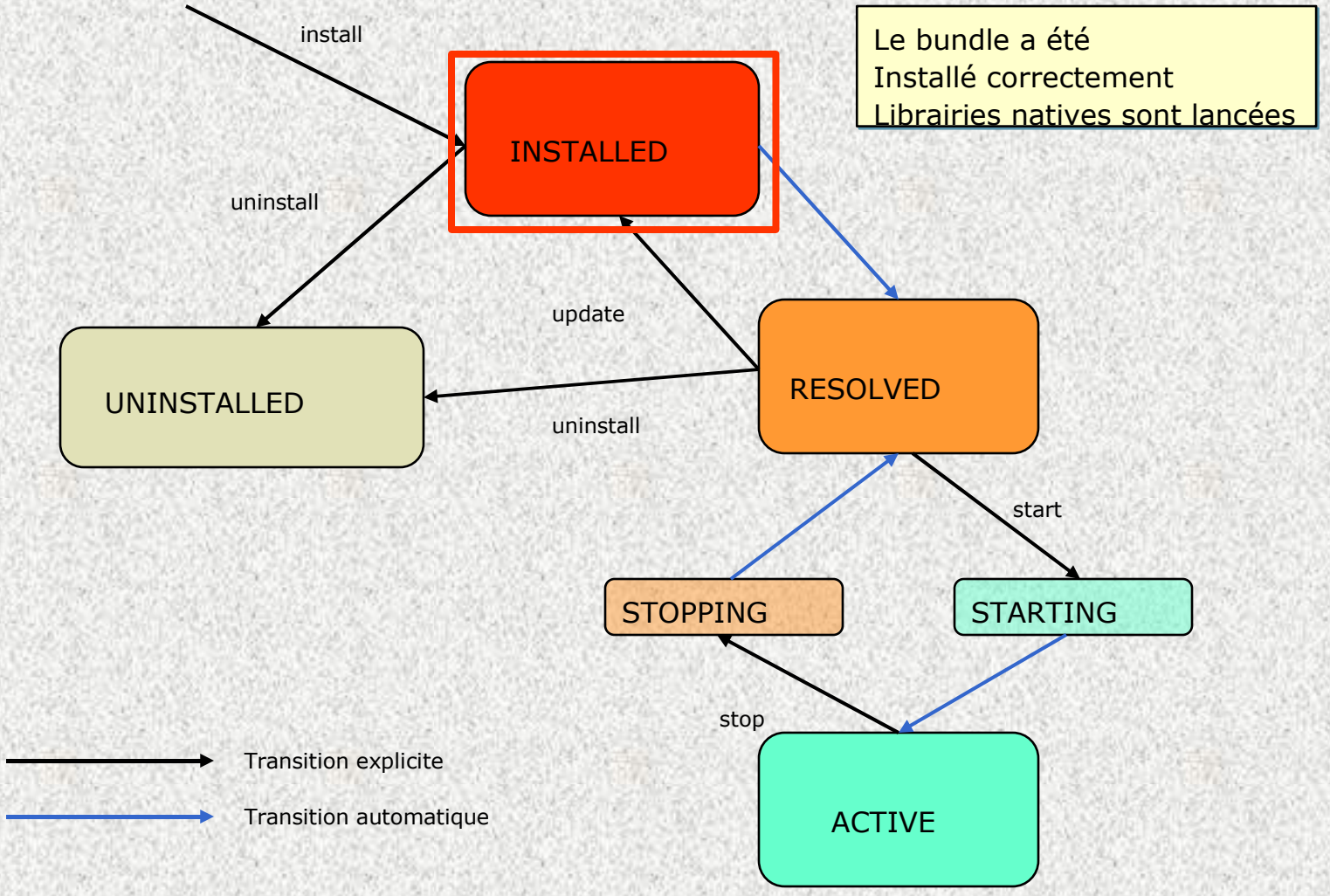
Bundle et Service



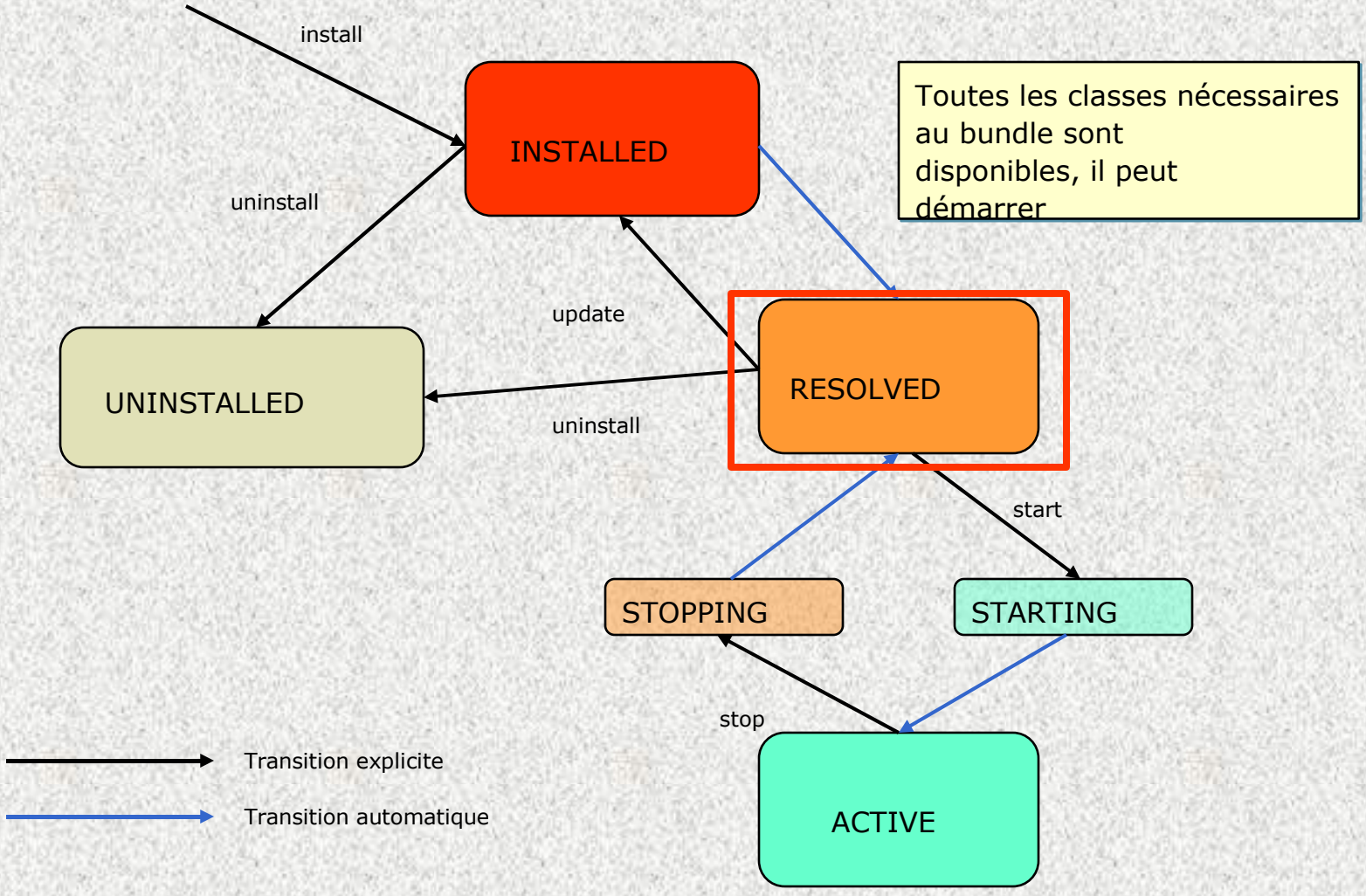
Cycle de vie et état d'un bundle



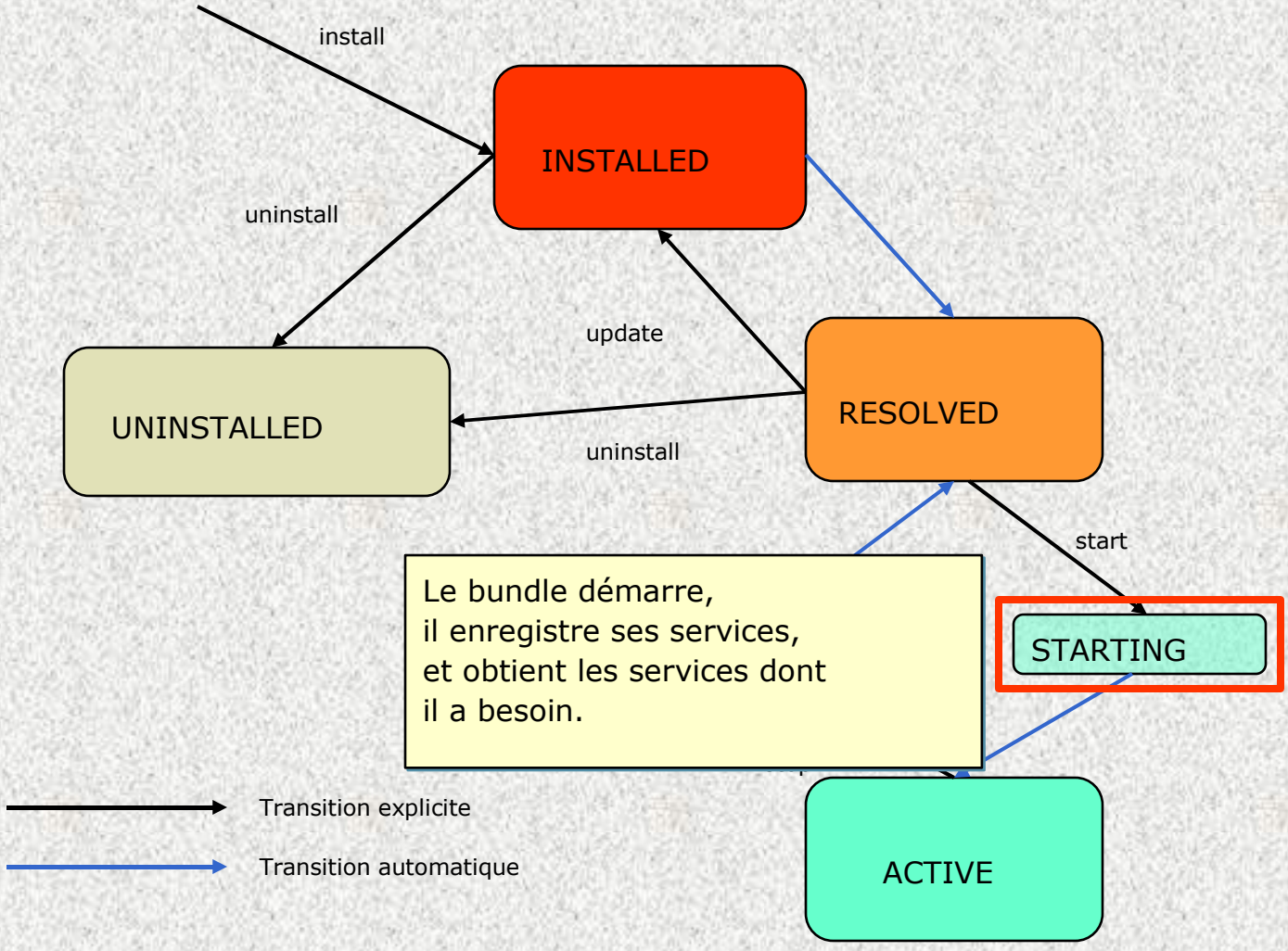
Cycle de vie et état d'un bundle



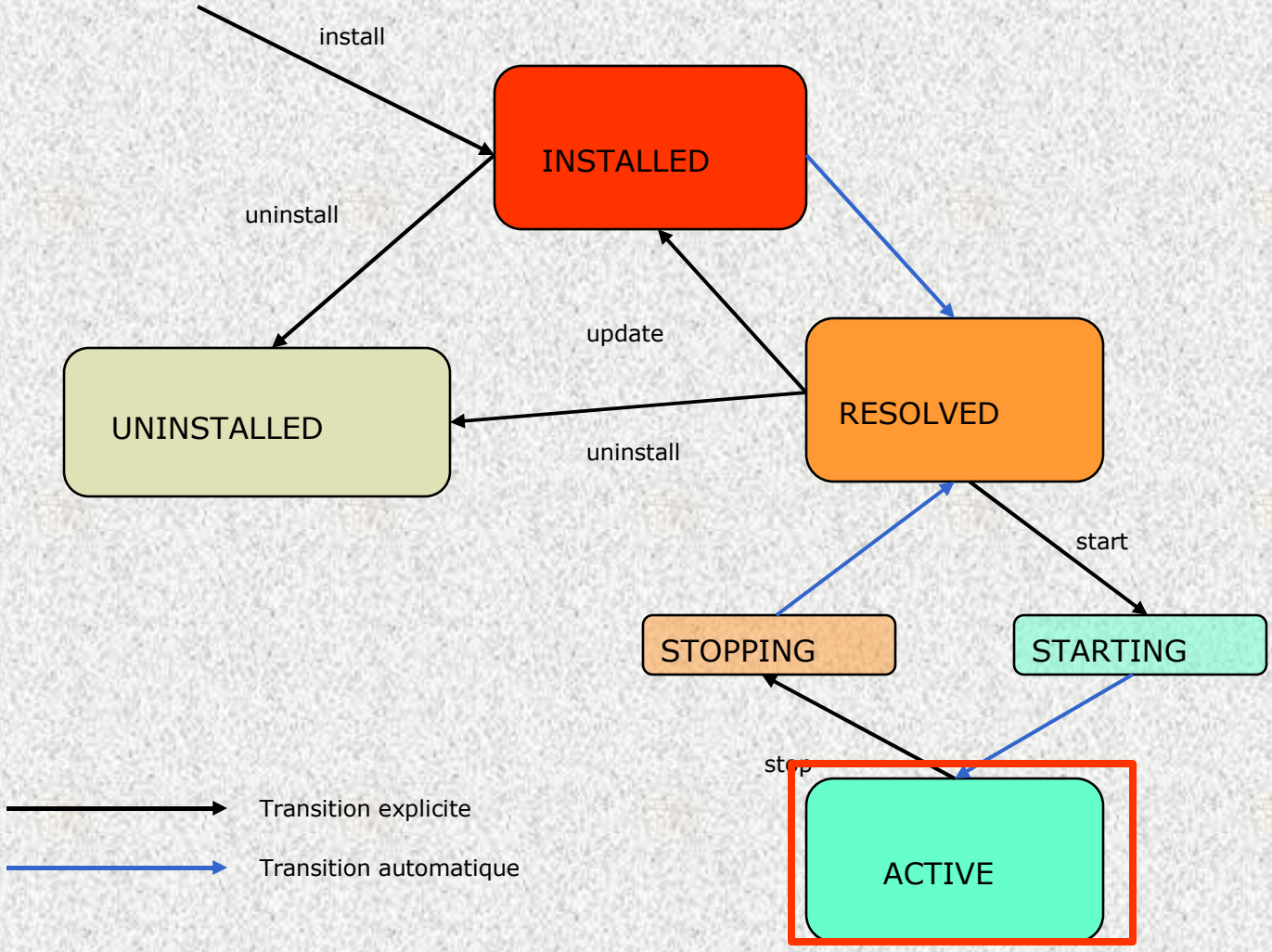
Cycle de vie et état d'un bundle



Cycle de vie et état d'un bundle

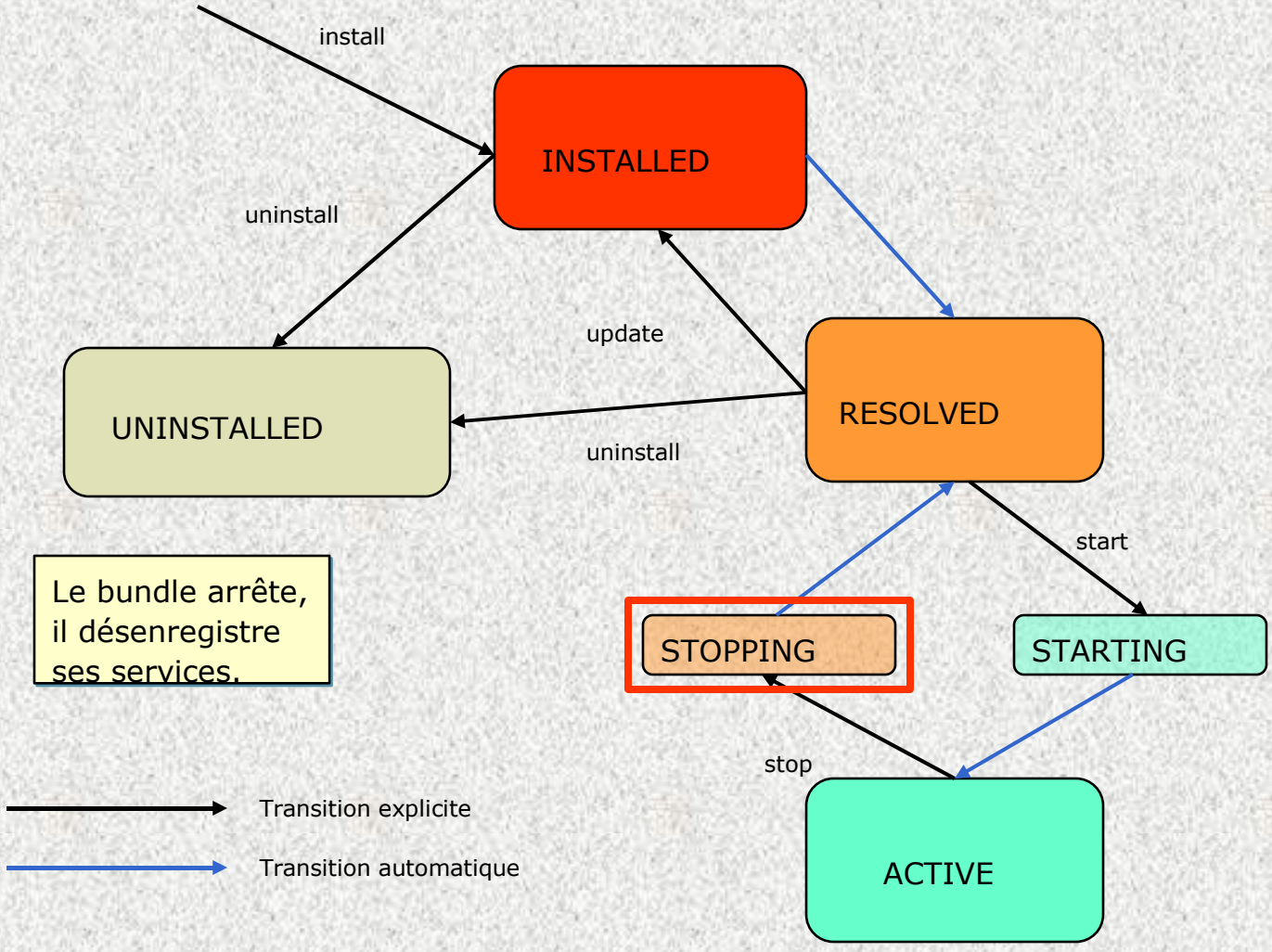


Cycle de vie et état d'un bundle

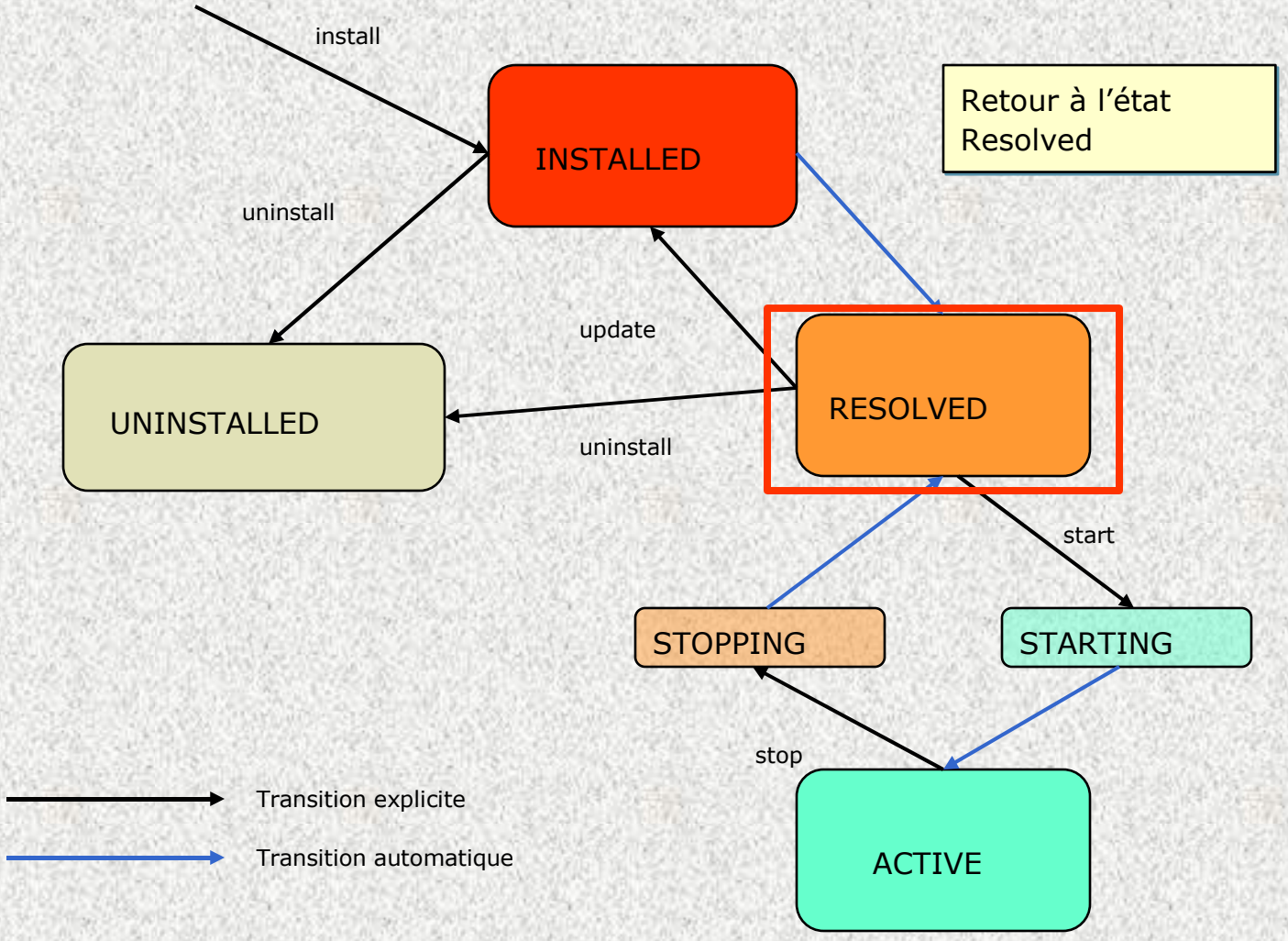


Le bundle est Actif

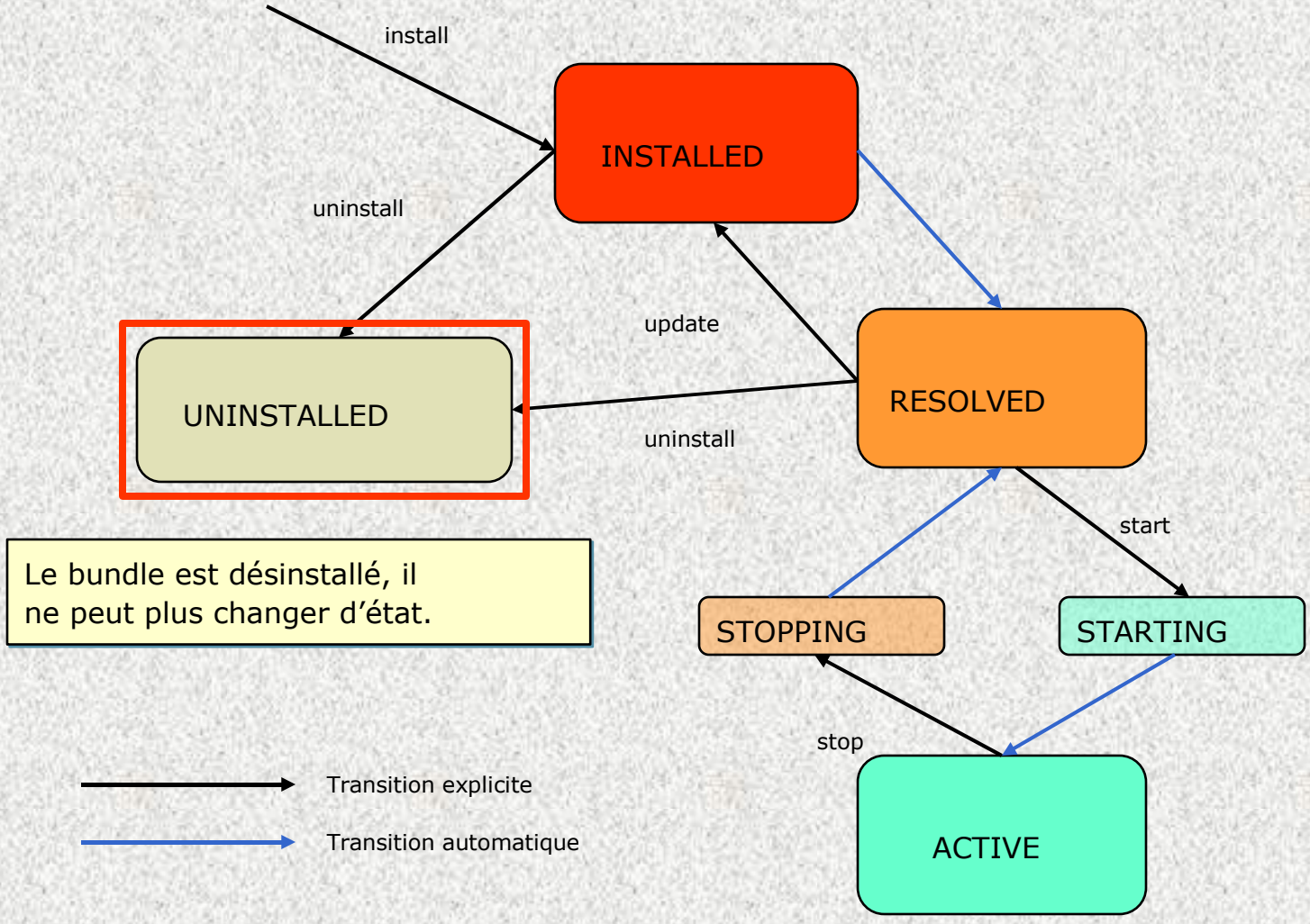
Cycle de vie et état d'un bundle



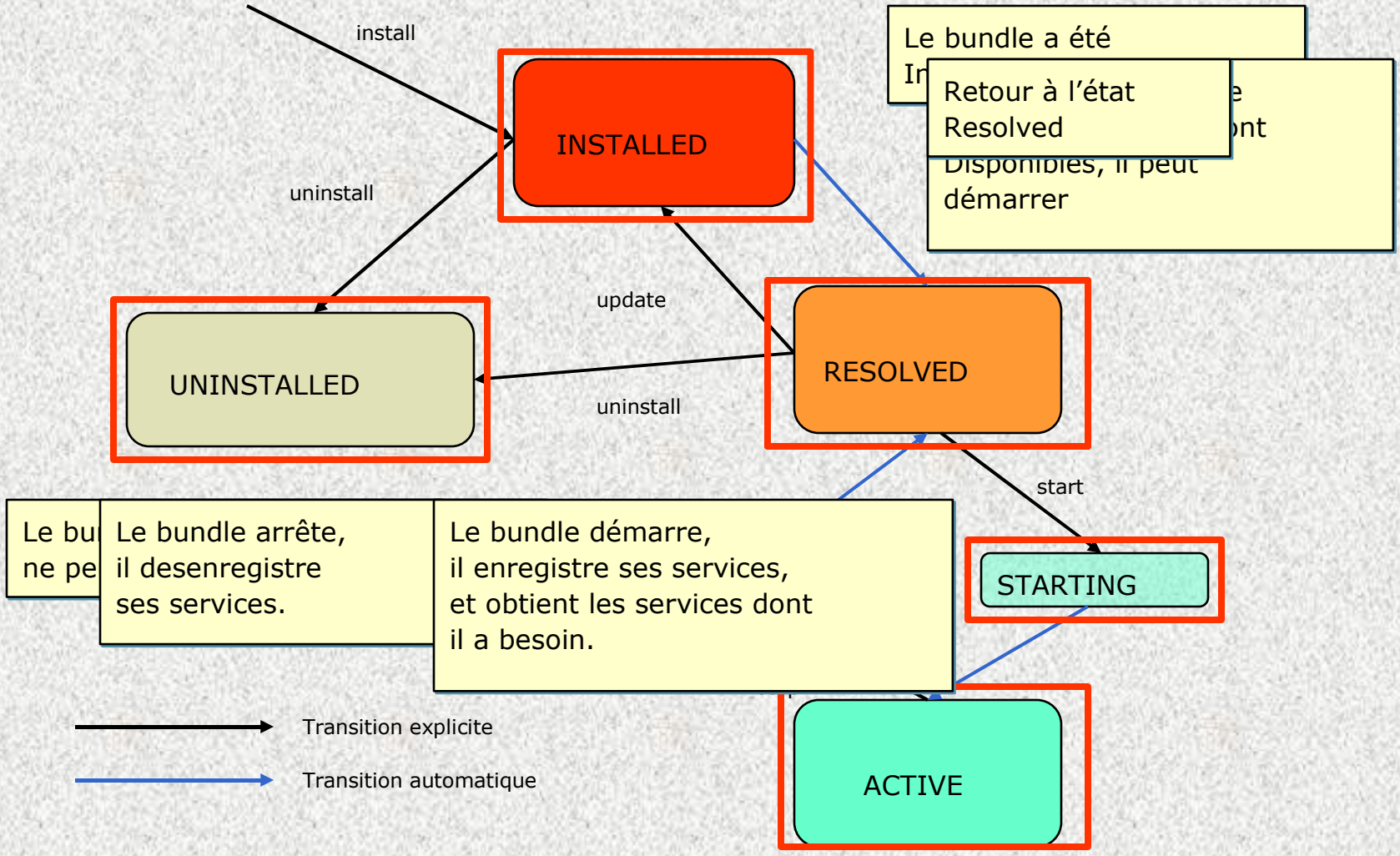
Cycle de vie et état d'un bundle



Cycle de vie et état d'un bundle



Cycle de vie et état d'un bundle



Services

- Le service est la brique de la programmation orientée Service (SOP)
- Il se décompose en deux parties :
  - Une interface de description.
    - Dans le cas d'OSGi c'est une interface java
  - Une ou plusieurs instances d'implantations

  * La description d'un service n'est pas liée à un bundle particulier
- Pour **utiliser** un service
  - Il faut demander au framework s'il existe un service qui met en œuvre une interface particulière (interface==service)
  - Dans une seconde étape demander à obtenir une implantation de ce service
- Pour **mettre** à disposition un service
  - Il faut fabriquer une instance d'implantation du service
  - Mettre à disposition du framework cette implantation sous l'interface voulue
- L'association entre deux services n'est pas gérée par le framework
  - Soit contrôle systématique, soit événement

Événements

- ServiceEvent
  - Notifie l'enregistrement ou le retrait de services
  - interface `ServiceListener` méthode `serviceChanged`
    - Traitement séquentiel et synchrone des listeners
- FrameworkEvent
  - Notifie le démarrage et les erreurs du Framework
  - interface `FrameworkListener` méthode `frameworkEvent`
    - Traitement séquentiel et asynchrone des listeners (par event dispatcher)
- BundleEvent
  - Notifie les changements dans le cycle de vie des bundles
  - interface `BundleListener` méthode `bundleChanged`
    - Traitement séquentiel et asynchrone des listeners (par event dispatcher)
  - interface `SynchronousBundleListener` méthode `bundleChanged`
    - Traitement séquentiel et synchrone des listeners (avant le traitement du changement d'état)

Recherche (Courtage) de services

- Filtrage par des expressions de condition LDAP (RFC1960) sur les propriétés enregistrées par les services
- Expressions de filtrage
  - Expressions simples (attribut opérateur valeur)
  - Valeurs de type `String, Numerique, Character, Boolean, Vector, Array`
  - Attribut insensible aux majuscules/minuscules
  - L'attribut `objectClass` représente le nom du service
  - Opérateurs `>=, <=, =, ~=` (approximativement égal), `=*` (présent)
  - Connecteurs logiques `&, |, !`

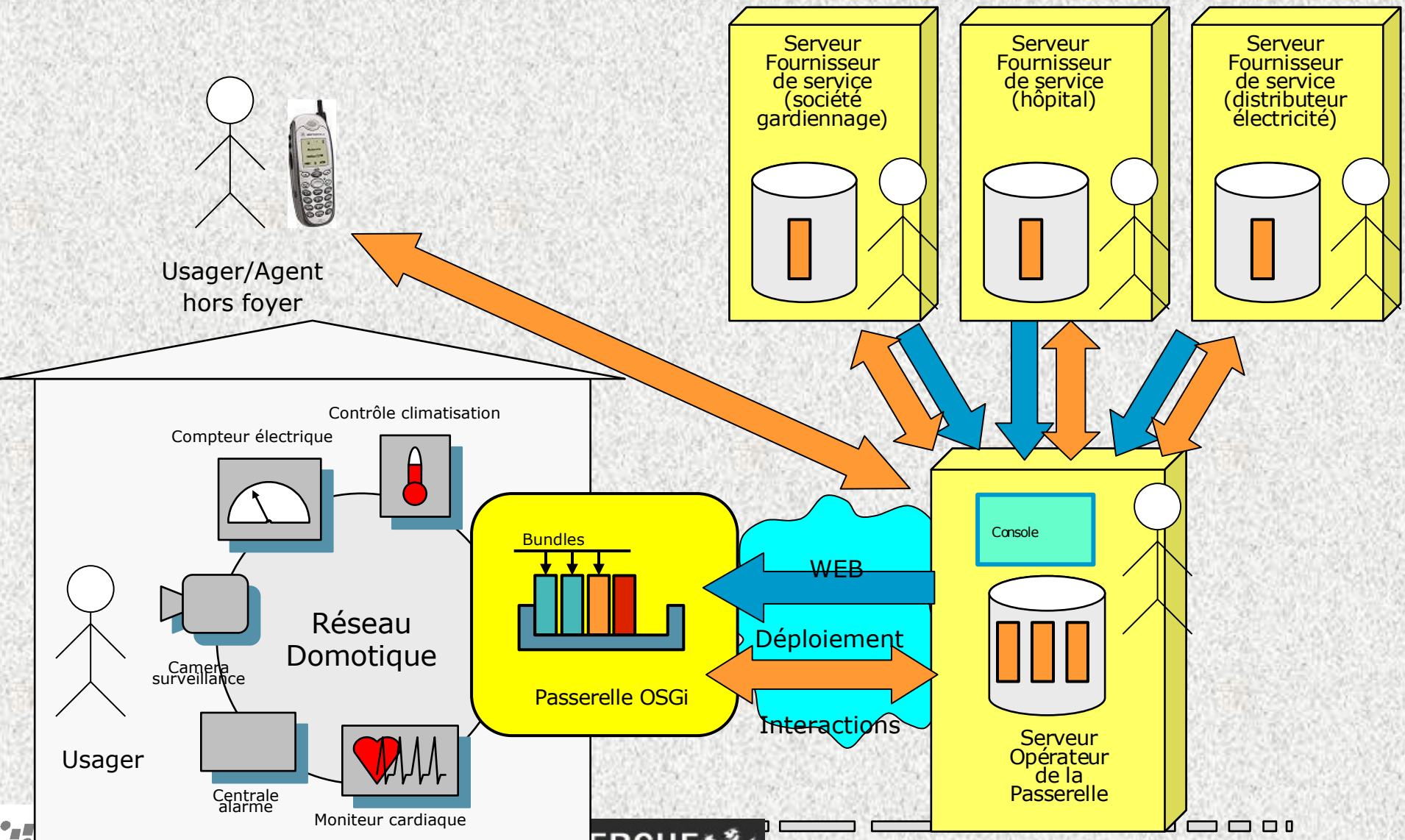
Enregistrement de services

```
package com.lexmark.printer.laser.impl;
public class Activator implements BundleActivator {
  private ServiceRegistration reg=null;
  private PrintService theService=null;
  public void start(BundleContext ctxt) throws BundleException {
    theService=new PrintServiceImpl();
    Properties props=new Properties();
    props.put("type", "laser");
    props.put("dpi", "72,150,300,600,1200");
    props.put("location", "1st floor");
    reg=ctxt.registerService("org.device.print.PrintService", theService, props);
  }

  public void stop(BundleContext ctxt) throws BundleException {
    if(reg != null) reg.unregister();
  }
}
```

Recherche de services

```
package org.eclipse.texteditor.impl
import org.device.print.PrintService;
class Activator implements BundleActivator {
   public void start(BundleContext ctxt) throws BundleException {
              private PrintService ser;
              // On va voir si quelqu'un offre un PrintService ...
              ServiceReference[] tempRefs
                        =ctxt.getServiceReferences
                        ("org.device.print.PrintService","(location=1st floor)");
              if(tempRefs!=null) {
                 System.out.println("Found a PrintService! I will use it!!!");
                 // On prend le premier offert!
                ser=(PrintService) ctxt.getService(tempRefs[0]);
              }
              ...
   }
   ...
}
```

Architecture générale



OSGi en résumé

- Sur-couche à java (initialement imaginé en python !)
  - Approche modules au dessus de java
    - Librairies natives et jar
    - Garantie d'exécution (pas de NoClassDefFound)
    - Simplicité de mise en oeuvre
  - Déploiement de composants
  - Gestion fine du cycle de vie (Conteneur)
  - Programmation Orienté Services
- Middleware de services standards
  - Service communication (http, wireAdmin, device discovery)
  - Services de management (obr, shell, Jmx, useradmin, properties)
- Autres Applications
  - Middleware de services, Grille, Agents actifs ==> Systèmes à plugins
- Système à exécution centralisée et à déploiement distant

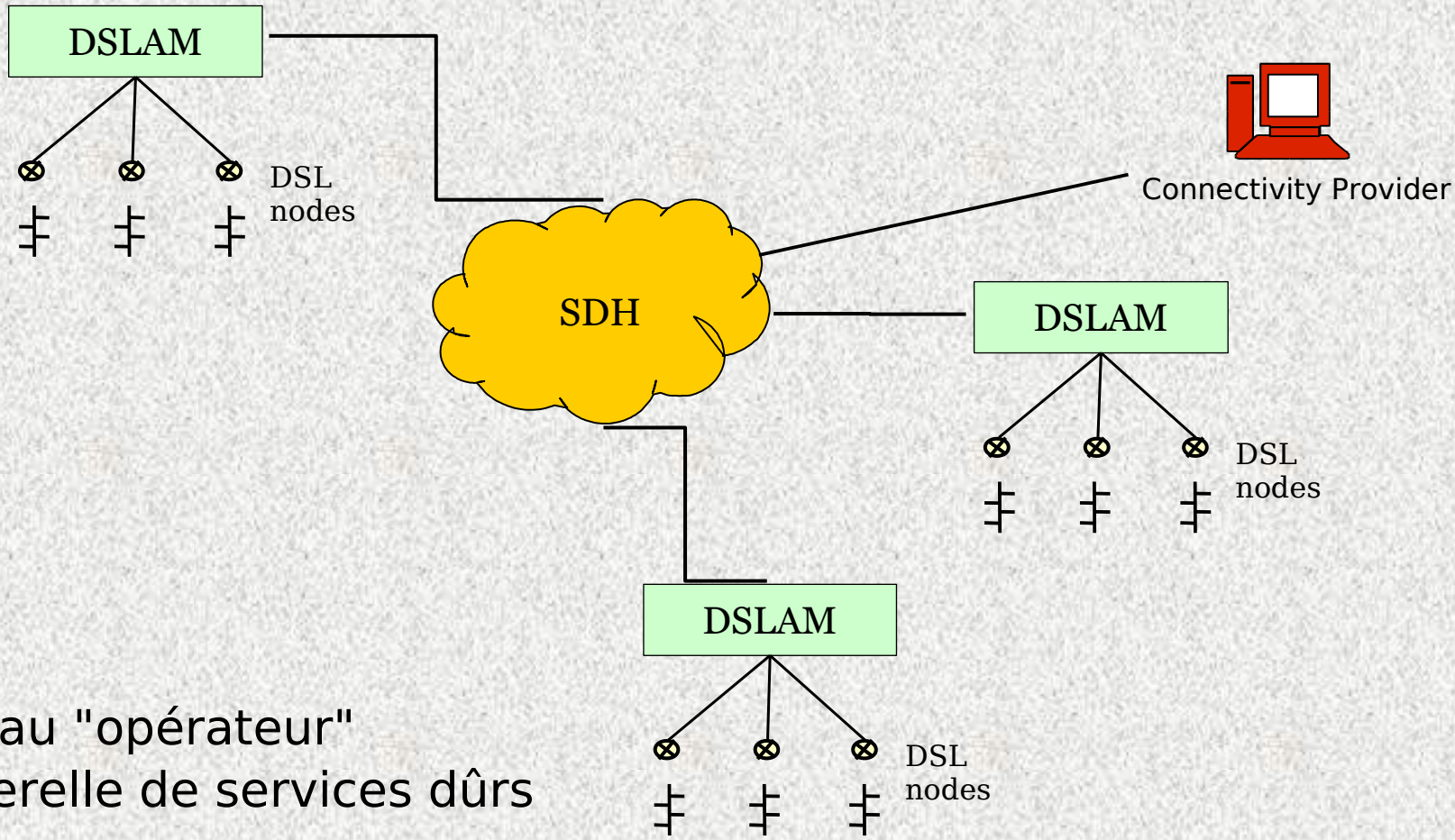
Ressources

- Où
  - Environnements embarqués
  - Téléphones mobiles
  - Eclipse (equinox)
- Qui
  - Motorola, BMW, Ericson, Electrolux, Deutsche telecom, Acunia
- Quoi
  - Oscar : http://oscar.objectweb.org
  - Knopferlish : http://www.knopferflish.org/
  - IBM, Prosyst, Sun
- Les autres approches
  - Microsoft .NET embedded
  - Avalon, Metro, PicoContainer

Déploiement P2P de services OSGi

Stéphane Frénot
Laboratoire CITI
Equipe INRIA Arès
INSA Lyon
14 juin 2005

Réseau d'accès



- Réseau "opérateur"
- Passerelle de services dûrs

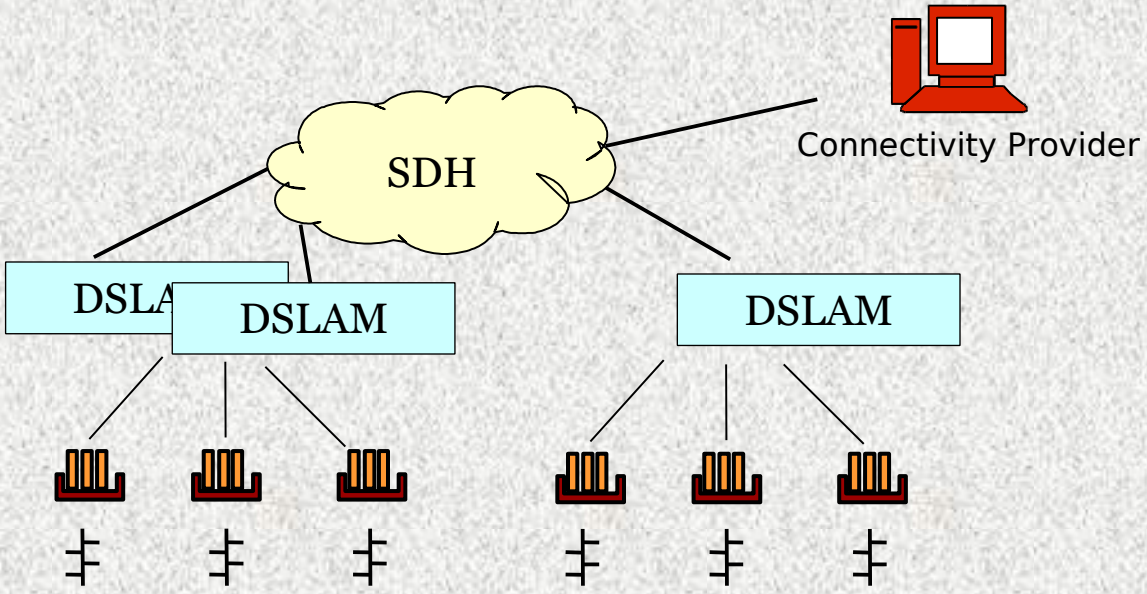
Nouveaux services

- Cas d'utilisation : le e-frigo
- Domotique
  - Pannes : température, lumière
  - Intervention technique
- Grande distribution
  - Compartiments vides
  - Préférences utilisateur
  - Commandes automatiques

Nouveaux besoins

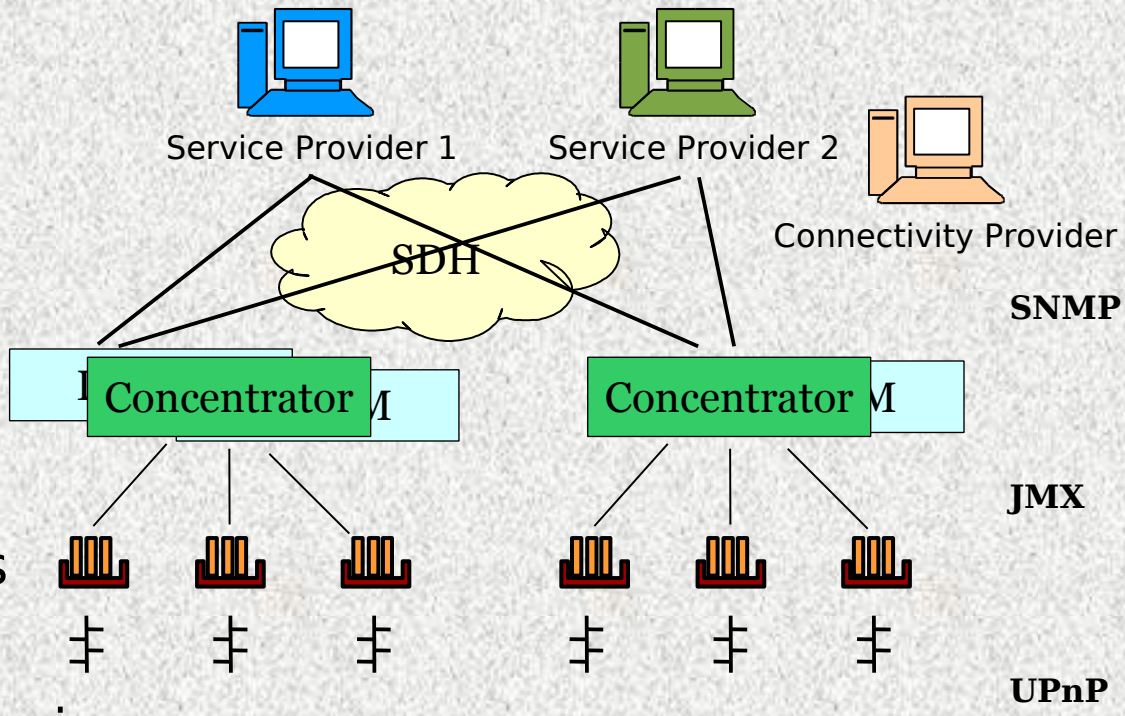
- Environnements multi-fournisseurs, multi-services
- Chacun supervise ses services (connexion, fournisseurs, utilisateur)
- Déploiement et mise à jour de services
- Explosion du nombre d'objets à gérer
- Dynamique et mobilité des objets
- Sécurisation globale
- Diversité des protocoles mis en oeuvre

Réseau overlay d'opérateurs de services



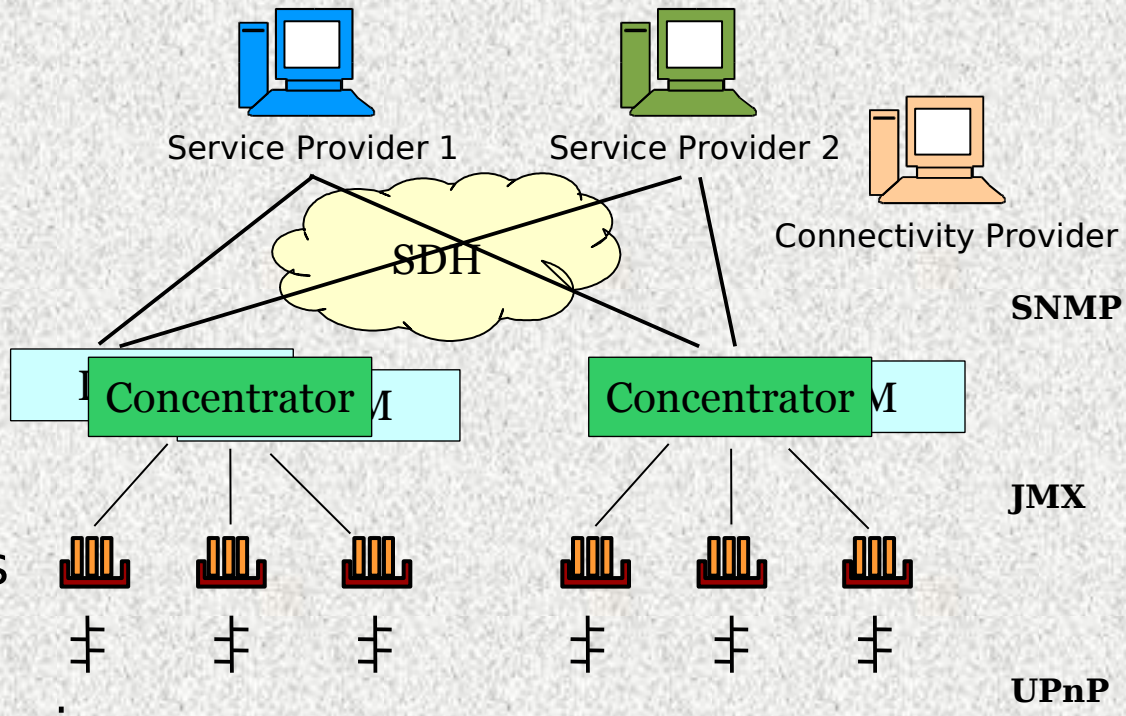
Réseau overlay d'opérateurs de services

- **Fautes / Performance**
  - Passage à l'échelle
  - Cohérence globale
- **Securité**
  - Confiance
  - Detection d'anomalies
- **Facturation**
  - Instrumentation de services
- **Configuration (Déploiement)**
  - Installation CDN
  - P2P



Réseau overlay d'opérateurs de services

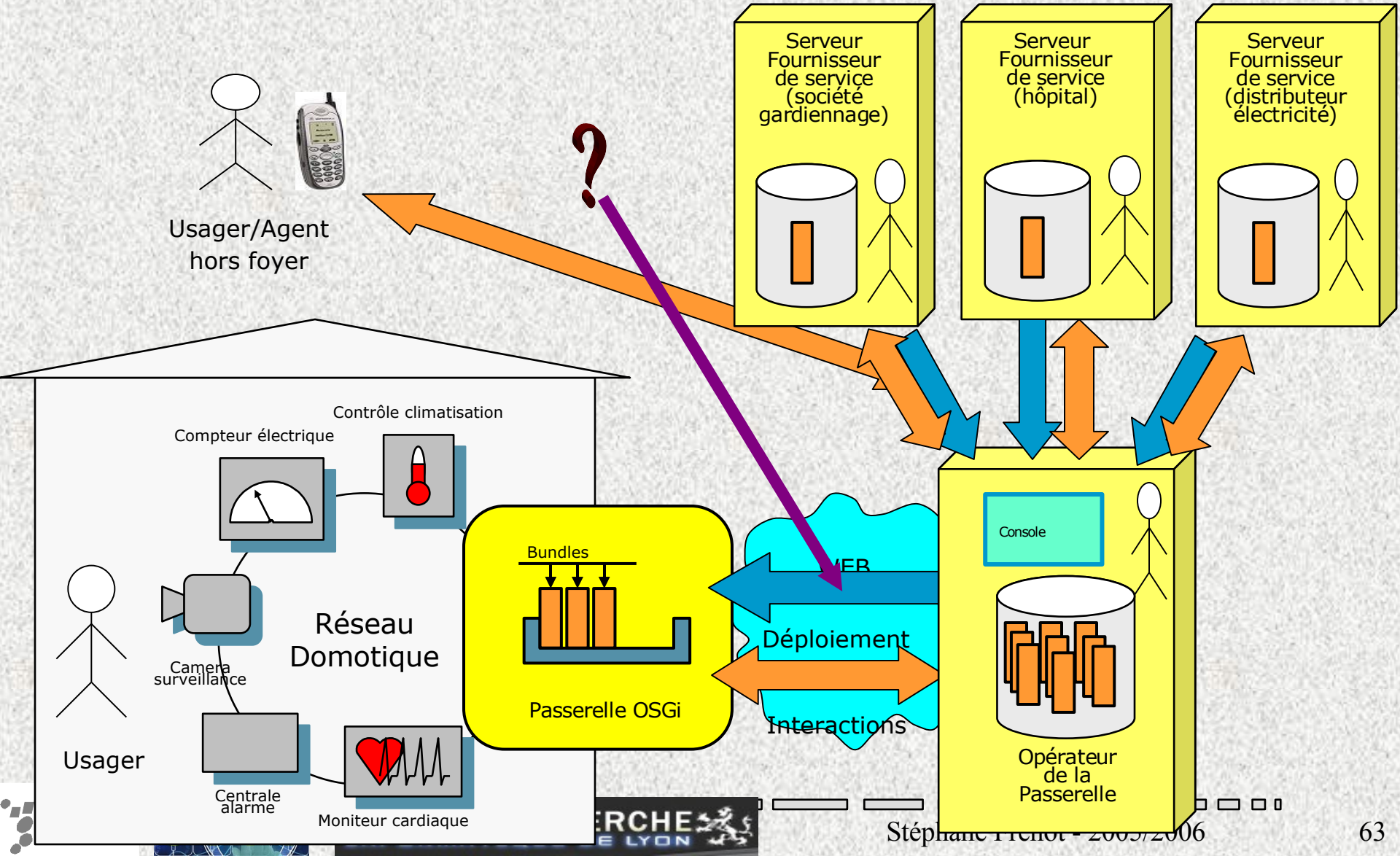
- **Fautes / Performance**
  - Passage à l'échelle
  - Cohérence globale
- **Securité**
  - Confiance
  - Detection d'anomalies
- **Facturation**
  - Instrumentation de services
- **Configuration (Déploiement)**
  - Installation CDN
  - P2P



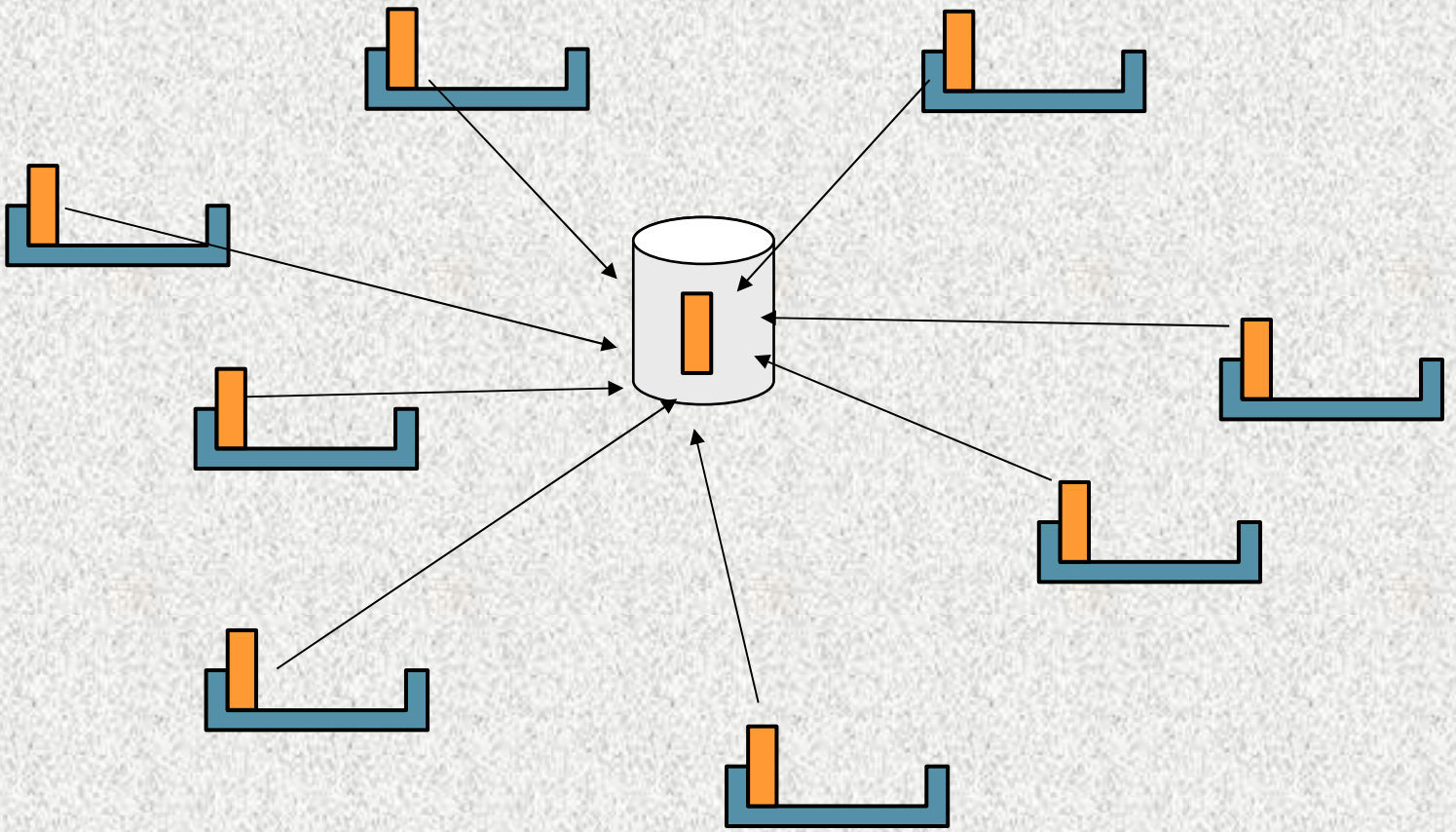
OSGi au CITI

- Supervision de passerelles
  - Intégration d'un agent de supervision JMX
  - Instance de supervision d'une gateway
  - Console de supervision
- Virtualisation de passerelles
  - Core et Virtual Gateways
  - Chaque fournisseur ne voit que son instance
- Auto-Configuration, Auto-Lancement
  - Système linux minimaliste (noyau, modules, java, OSGi)
- Déploiement P2P de bundles

Architecture générale



Le déploiement standard OSGi

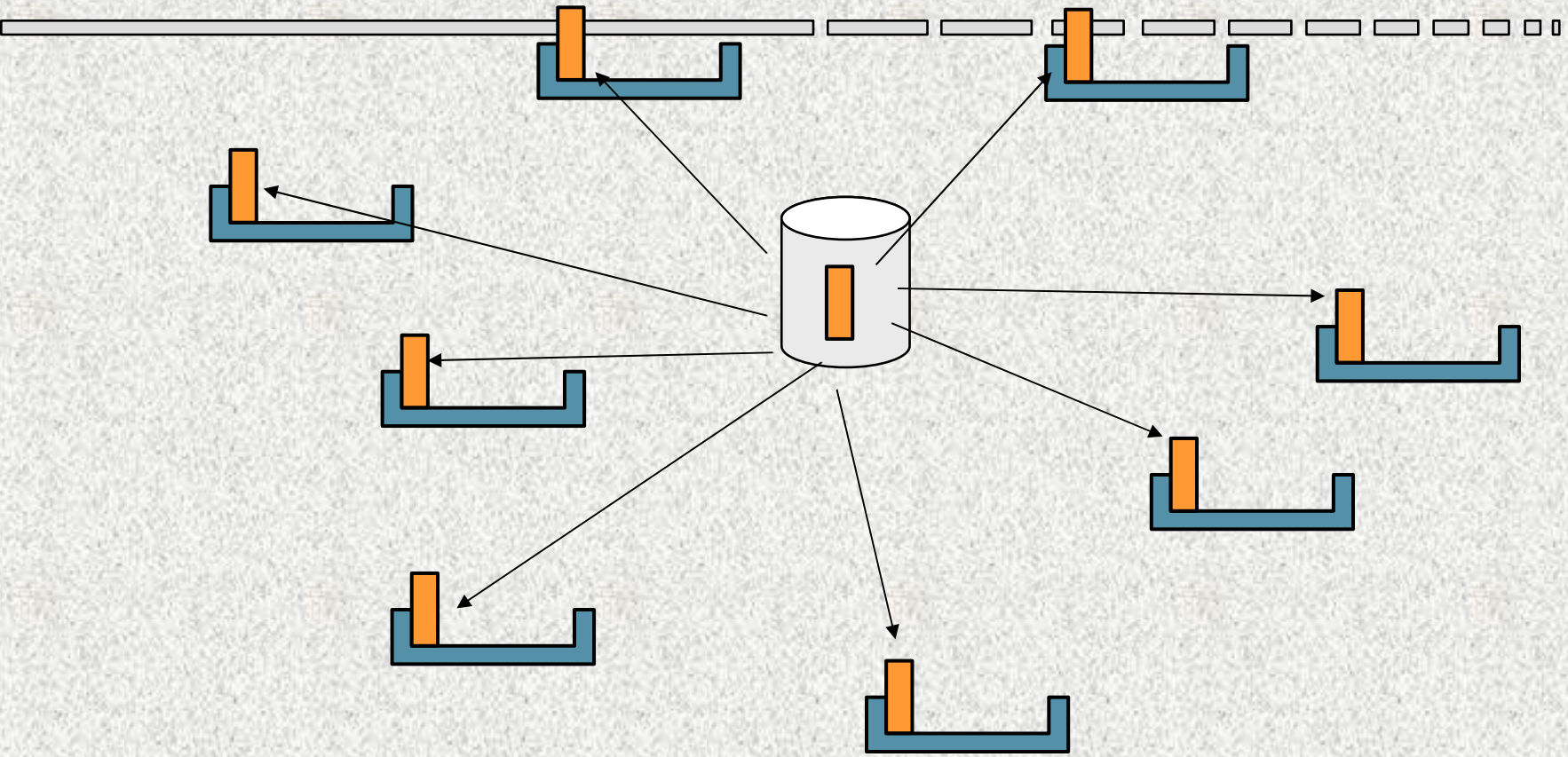


Chaque passerelle doit faire un pull de service

> start http://unsite.quelquepart/service.jar

==> Comment savoir quand déclencher la mise à jour ?

Push de services

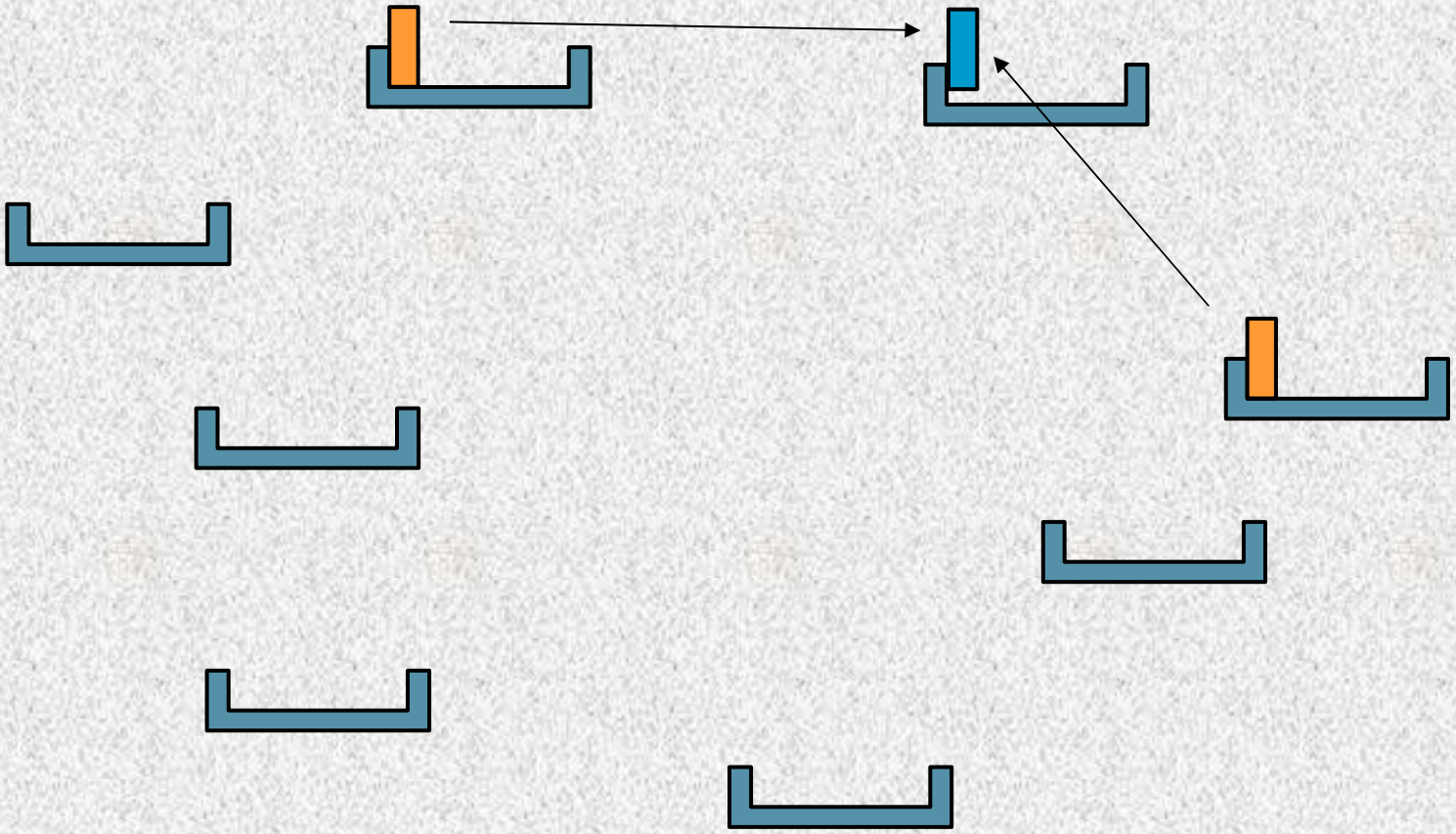


Un nœud central peut faire une diffusion de service

> push service.jar

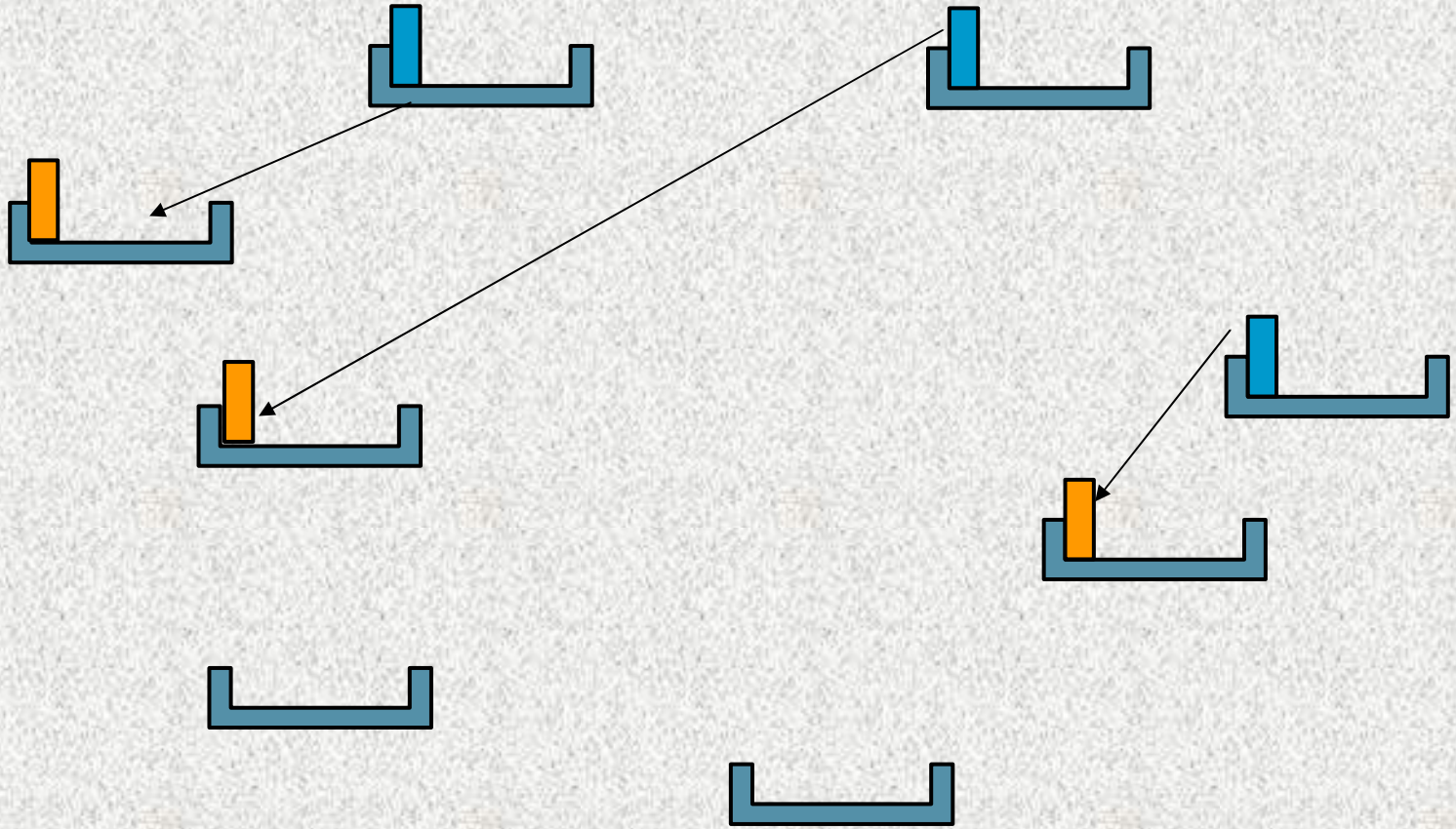
Comment faire avec le côté dynamique des passerelles ?

Déploiement P2P de services



Un nœud quelconque reçoit la nouvelle version du service
> deploy service.jar

Déploiement
P2P de services



Les autres passerelles sont inondées

Proposition

- Infrastructure de déploiement p2p des bundles
  - Chaque gateway est un nœud du réseau P2P
  - Chaque bundle est initialement déposé sur une ou plusieurs gateways
  - Les bundles sont propagés en fonction des besoins

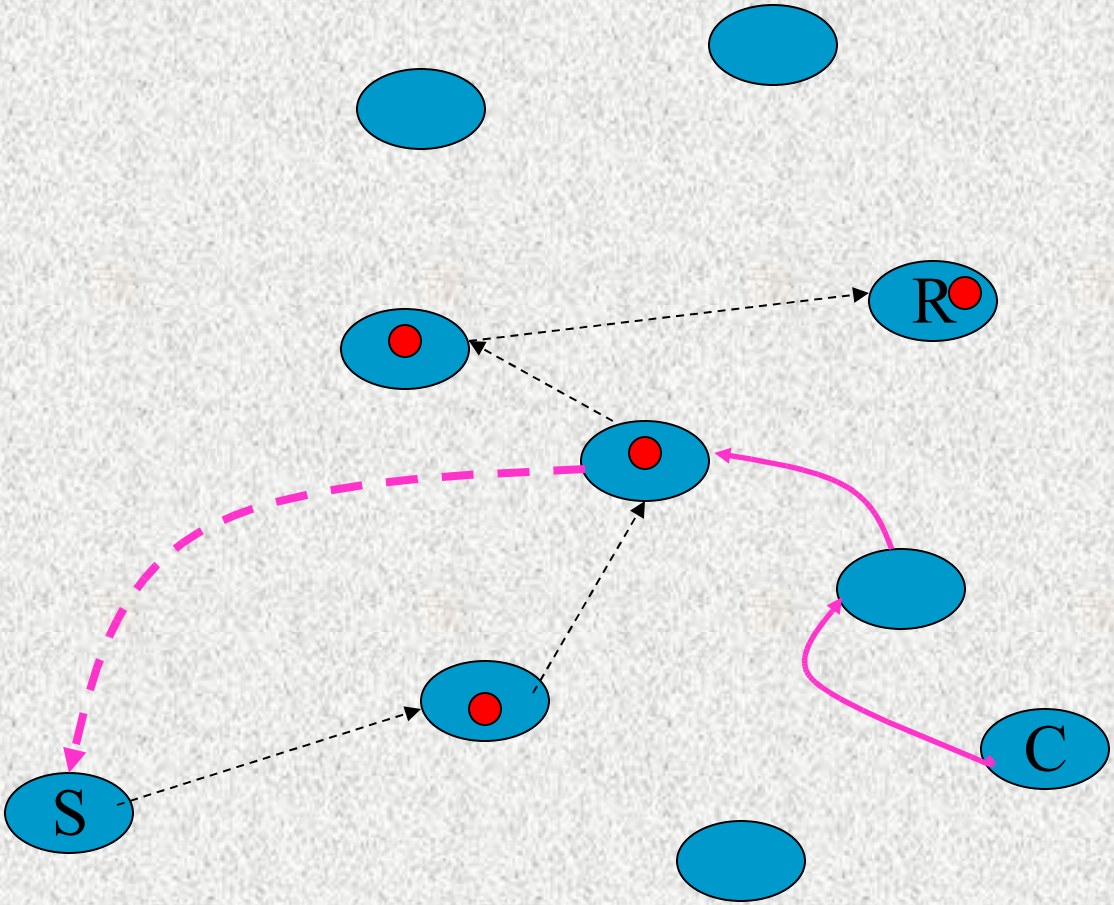
Présentation

- Infrastructures P2P
  - Types et propriétés requises
  - Choix
- Implantation dans OSGi
  - Description des services
- Conclusions

Les réseaux P2P

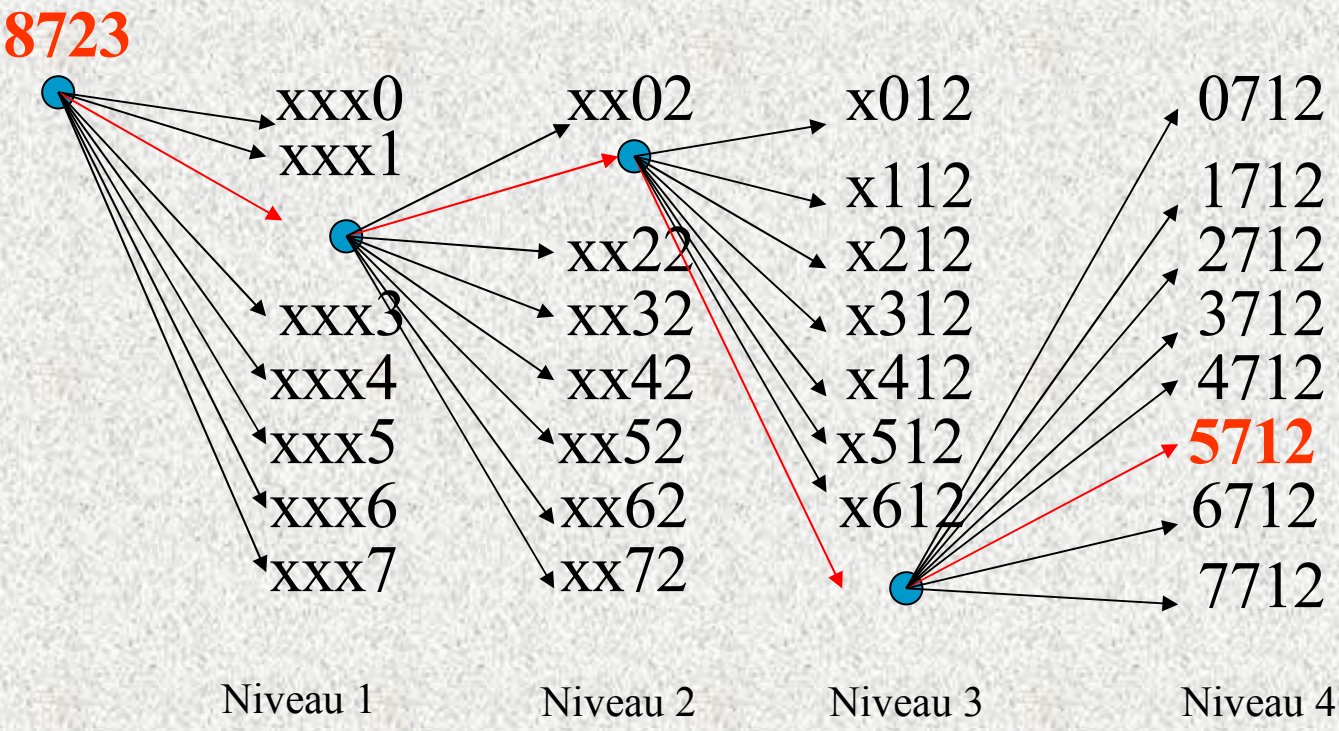
- Les réseaux à index
  - Napster, Kazaa…
  - Partage de ressources
- Les réseaux à inondation
  - Gnutella
  - Disparition des index centralisés
- Les réseaux à traînées
  - Freenet, FreePastry
  - Anonymisation, Réplication
- Les Hashtables distribuées (DHT)
  - Chord, CAN
  - Surcouche de localisation

Routage dans DHT



Pastry

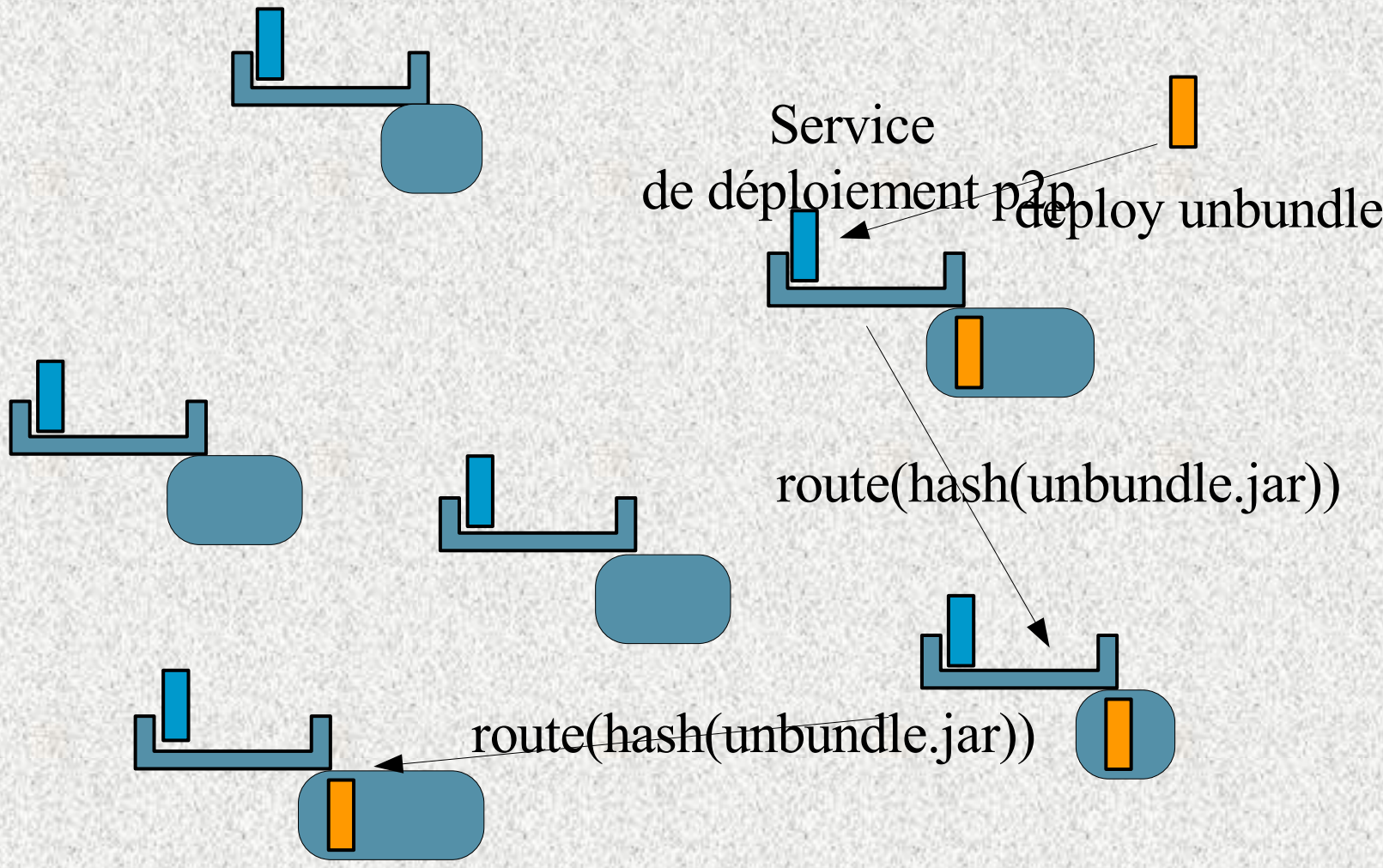
- Table pour de routage pour 5712 à partir de 8723



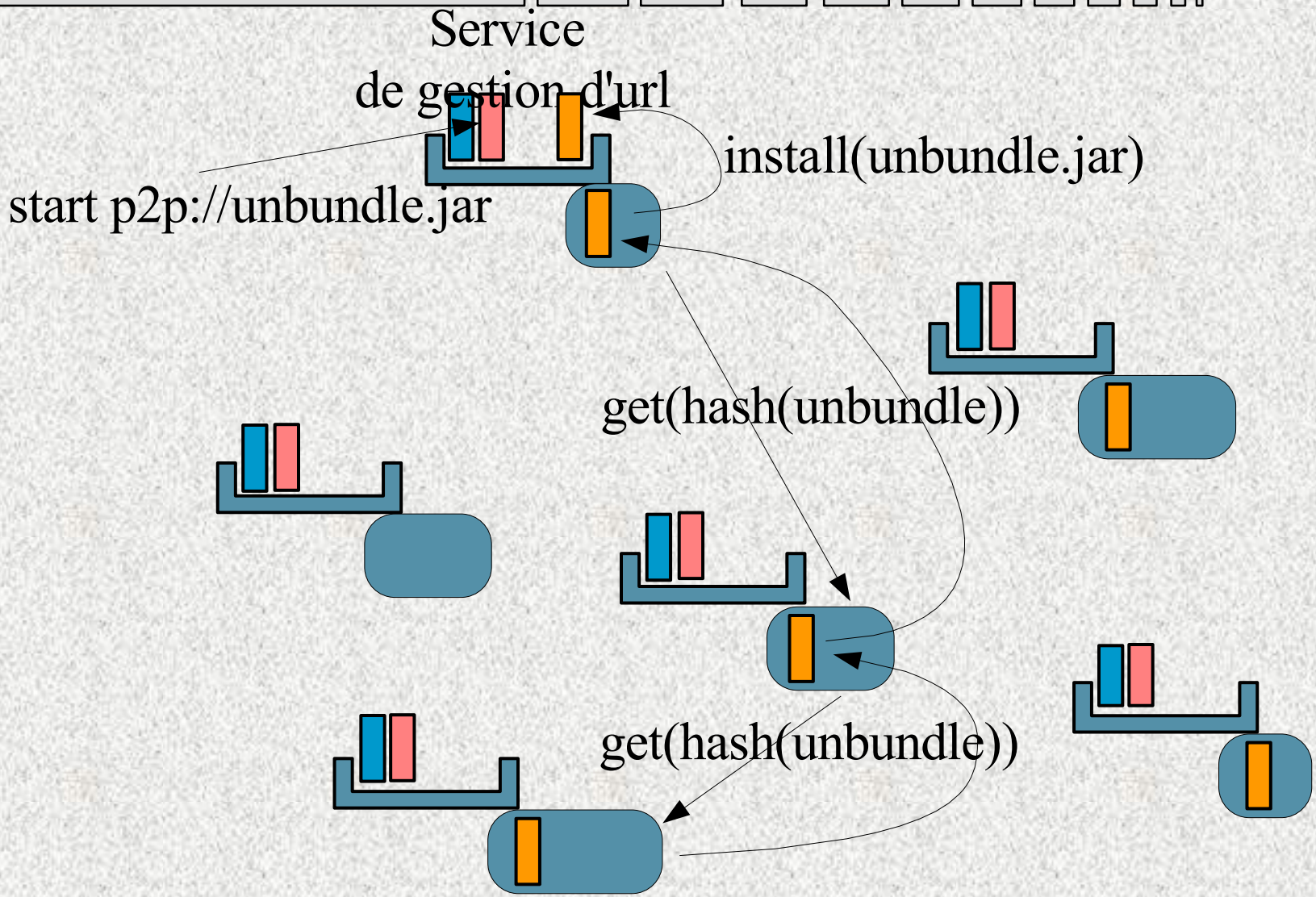
Services P2P / OSGi

- Localisation
  - Clé de hash sur les bundles
- Réplication
  - Le bundle déployé est installé sur les nœuds intermédiaires
- Inondation
  - Les ressources les plus utilisées sont les plus répliquées
- Identification / authentification
  - MD5 d'un bundle

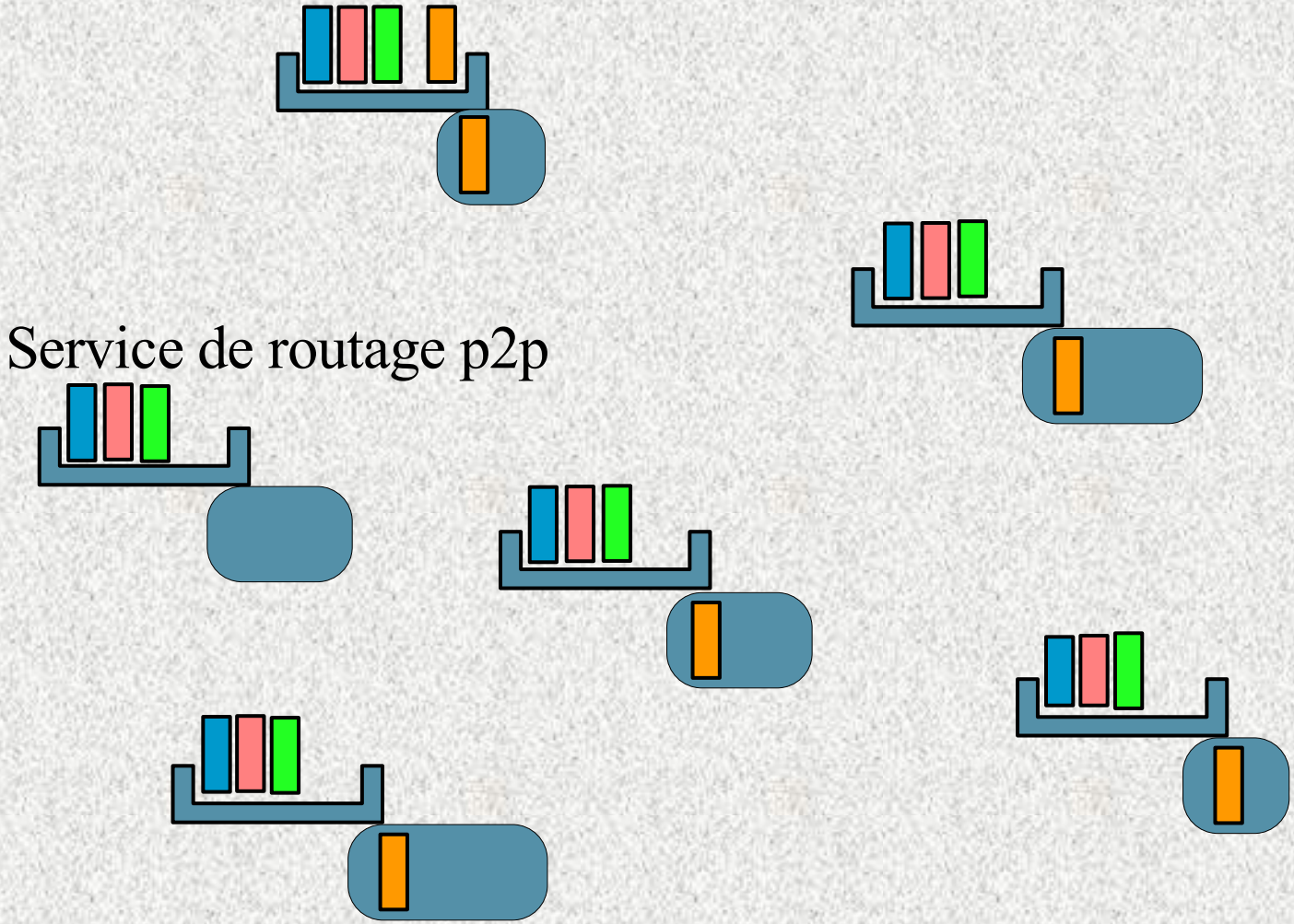
Diffusion d'un bundle



Installation d'un bundle



Architecture générale



Evolutions

- Gestion des versions
  - La diffusion se fait en deux phases
    - publish(unbundle, unbundle_0.0.1.jar)
    - publish(unbundle_0.0.1, byteCode)
- Index Central
  - Obr - Object Bundle Repository
    - Service standard Oscar
    - Gère les dépendances
    - Contient la description générique du bundle
- Exploitation du cache local
  - Pas de base locale spécifique
  - On met en diffusion les bundles INSTALLES

Mise en œuvre sous freePastr

- Composant d'interface à FreePastry (FP)
  - Emballe FP
  - Offre l'interface OSGi permettant d'interagir avec FP
- Extension du bundle Repository
  - Extension du composant offert par oscar (obr)
  - Il permet d'intégrer des ressources du type p2p://service
- Service d'administration de obr-e
  - Permet d'ajouter un nouveau bundle (Calcul automatique du nœud d'hébergement)

Commentaires

- Infrastructure de déploiement de bundle OSGi
  - Repose sur Freepastry
  - Intégrée de manière transparente à oscar (compatible obr)
  - 3 services
- Approche OSGi
  - Chaque nœud installe les services (pull)
  - Unicité des noms de bundles (nom+version)
- P2P
  - Possibilité de substituer par un autre réseau p2p (API standard)

Extensions

- Installation multi-Sources
  - Taille des bundles < 100ko
  - Si ressources passage en bittorrent
- Sécurité
  - Signature, anonymisation
- Découverte de services
  - Ne pas diffuser les bundles, mais les interfaces de services et les packages. (grain fin de déploiement
  - Application directe aux Machines virtuelles (gnuClasspath)