ScalAgent Distributed Technologies

JORAM

'un intergiciel de communication asynchrone'

André Freyssinet
ScalAgent Distributed Technologies

Andre.Freyssinet@scalagent.com

www.scalagent.com

Objectifs

- **Développer des applications « JMS »**
  - Utilisation d'un MOM (middleware à message)
  - Modèle de programmation JMS
  - Administration d'une application JMS
- **Mettre en œuvre des applications JMS avec JORAM**
  - L'architecture de JORAM
  - Guide d'installation et d'administration
  - JORAM par l'exemple
    - *Déploiement et administration de scénarios d'utilisation représentatifs*

Plan

- **Introduction aux MOM**
- **Présentation de l'API JMS**
  - Les fondamentaux et les limites de la spécification JMS
  - Le modèle de programmation JMS
- **L'architecture de JORAM**
  - Le MOM ScalAgent et la technologie à agents
  - L'architecture de JORAM : configuration et propriétés
- **Architecture JORAM centralisée**
  - Création de la configuration, lancement, administration
  - Administration programmée
- **Architecture JORAM distribuée**
  - Options d'architecture, création d'une configuration distribuée
  - objets JMS 'distribués'

les atouts du mode “asynchrone”

- **Application/système distribué**
  - Ensemble de composants logiciels coopérants
  - Coopération = communication + synchronisation
- **Systèmes faiblement couplés**
  - Couplage spatial : systèmes à grande échelle
    - *Communication « anonyme » : évolution dynamique des correspondants*
    - *Fonctionnement en mode partitionné : pannes temporaires de réseau*
  - Couplage temporel : systèmes autonomes communicants
    - *Communication « spontanée » en mode « push »*
    - *Fonctionnement en mode déconnecté : site absent ou utilisateur mobile*

Révolution ? pas vraiment !!

- **Internet et applications asynchrones**
  - Le courrier électronique (communication point-à-point)
    - *le producteur envoie un message à un destinataire qu'il connaît*
    - *le message est stocké sur un serveur, le consommateur reçoit ultérieurement le message lorsqu'il se connecte*
  - Les listes de diffusion (communication multi-points)
    - *Le message est diffusé à tous les éléments de la liste*
  - Les news (Anonymat, Publish/Subscribe)
    - *le consommateur s'abonne à une liste de diffusion*
    - *le producteur publie une information dans un forum*
    - *le consommateur lit le contenu du forum quand il le souhaite*
- **Applications d'intégration : MQ/Series date de la fin des années 70**

Les usages des systèmes asynchrones

- **Supervision**
  - Parc d'équipements distribués
  - Applications et processus métiers (BAM)
- **Echange de données (EDI)**
- **Intégration de données (ETL)**
  - Alimentation d'un datawarehouse/datamart depuis des sources de données hétérogènes autonomes
- **Intégration d'application**
  - Intra-entreprise : EAI (communication, routage, workflow)
  - Inter-entreprises : B2B et Web Services (communication, orchestration)
- **Informatique mobile**
  - Communication entre équipements mobiles et serveurs d'appplication

Principes directeurs

- **Couplage faible de l'émetteur et du destinataire**
  - ➔ Communication asynchrone
    - *« Store And Forward »*
  - ➔ Communication indirecte
    - *Désignation Anonyme : Indirecte, de groupe, associative..*
- **Persistance et fiabilité**
- **Messages typés**
  - ➔ Gestion de l'hétérogénéité
    - *Des données, des systèmes et des systèmes de communication.*

Modes de désignation

- **Désignation indirecte**
  - Les entités communiquent via un objet intermédiaire : BAL
- **Désignation de groupe**
  - groupe = ensemble de récipiendaires identifiés par un nom unique
    - *gestion dynamique du groupe :*
      - Protocole de découverte des membres,
      - Gestion des arrivée/départ de membres
    - *différentes politiques de service dans le groupe :*
      - Répartition de charge (1/N), T
      - Tolérance aux pannes (N/N)
- **Désignation associative**
  - les destinataires d'un message sont identifiés par des attributs du message

Modes de consommation

- « *Pull* » – consommation explicite
  - Les consommateurs programment explicitement l'accès aux messages
  - En cas d'absence de message : attente ou exception
- « *Push* » – consommation implicite
  - Une méthode prédéfinie (réaction) est attachée à la production d'un message (événement)
  - L'occurence d'un événement entraîne l'exécution de la réaction associée.

➔ *Modèle Evénement / Réaction*

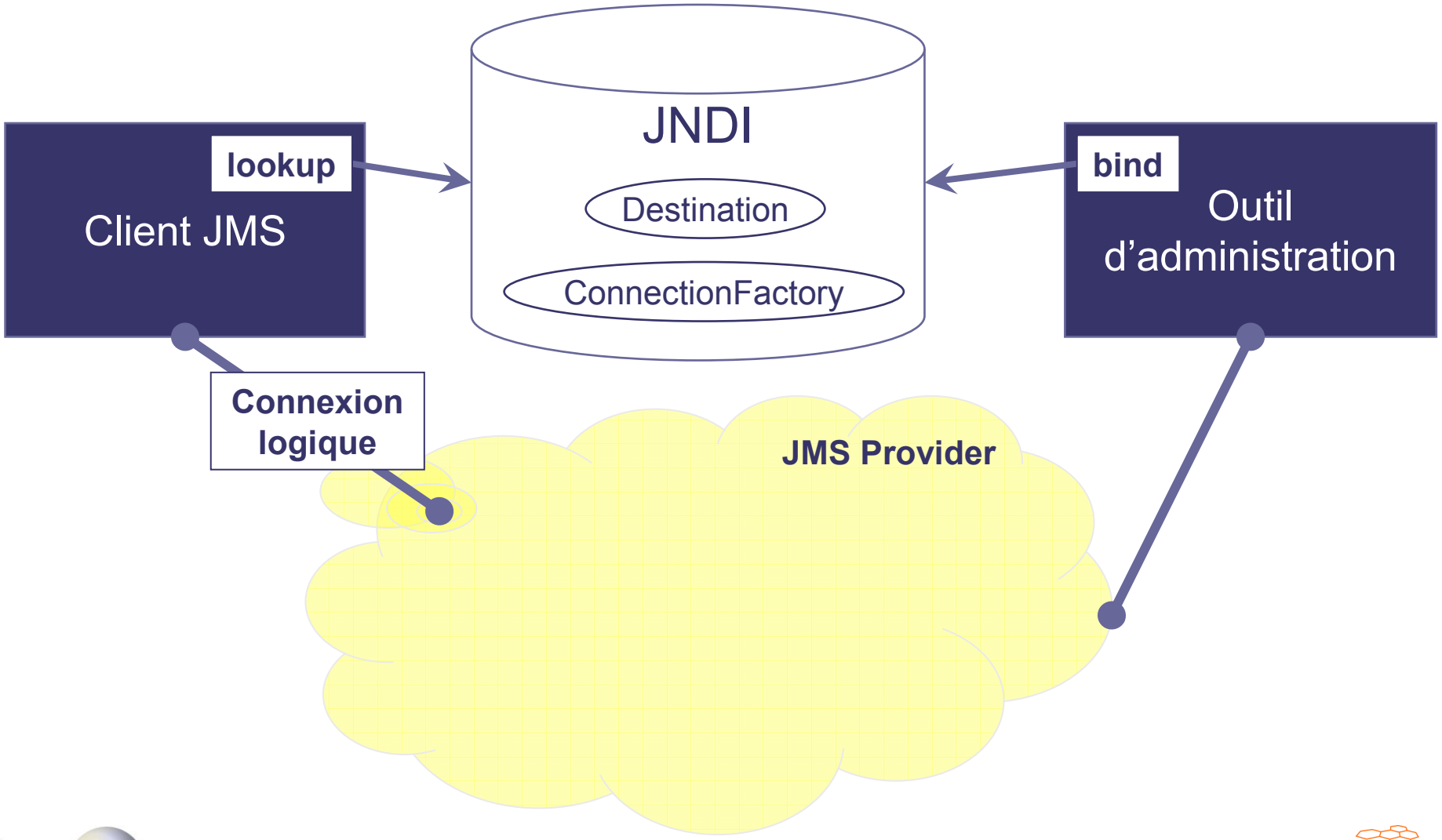
JMS - Java Message Service

- **Mapping Java entre une application cliente et un MOM**
- **Le support de JMS est requis dans J2EE 1.3**
  - Un composant essentiel de l'architecture J2EE
- **JMS ne spécifie pas le fonctionnement du MOM …**
  - … mais est défini pour couvrir la diversité de ceux-ci :
    - *Modèles de communication : "Point-to-Point", "Publish/Subscribe".*
    - *Réception : implicite, explicite.*
    - *Nombreux types de messages : textes, binaires, objets, etc.*
    - *Qualité de service: persistance, fiabilité, transactions, etc.*

Application JMS

- JMS Provider
- Clients Java JMS
- Objets administrés
  - ConnectionFactory, Destination.
- Messages

Architecture



« Messaging Domains »

- **Point-to-Point**
- **Publish/Subscribe**

- **JMS 1.1 : unification des domaines**
  - Réduit et simplifie l'API (à terme)
  - Permet l'utilisation de Queues et Topics dans une même connection (coût) et dans une même session (transaction)

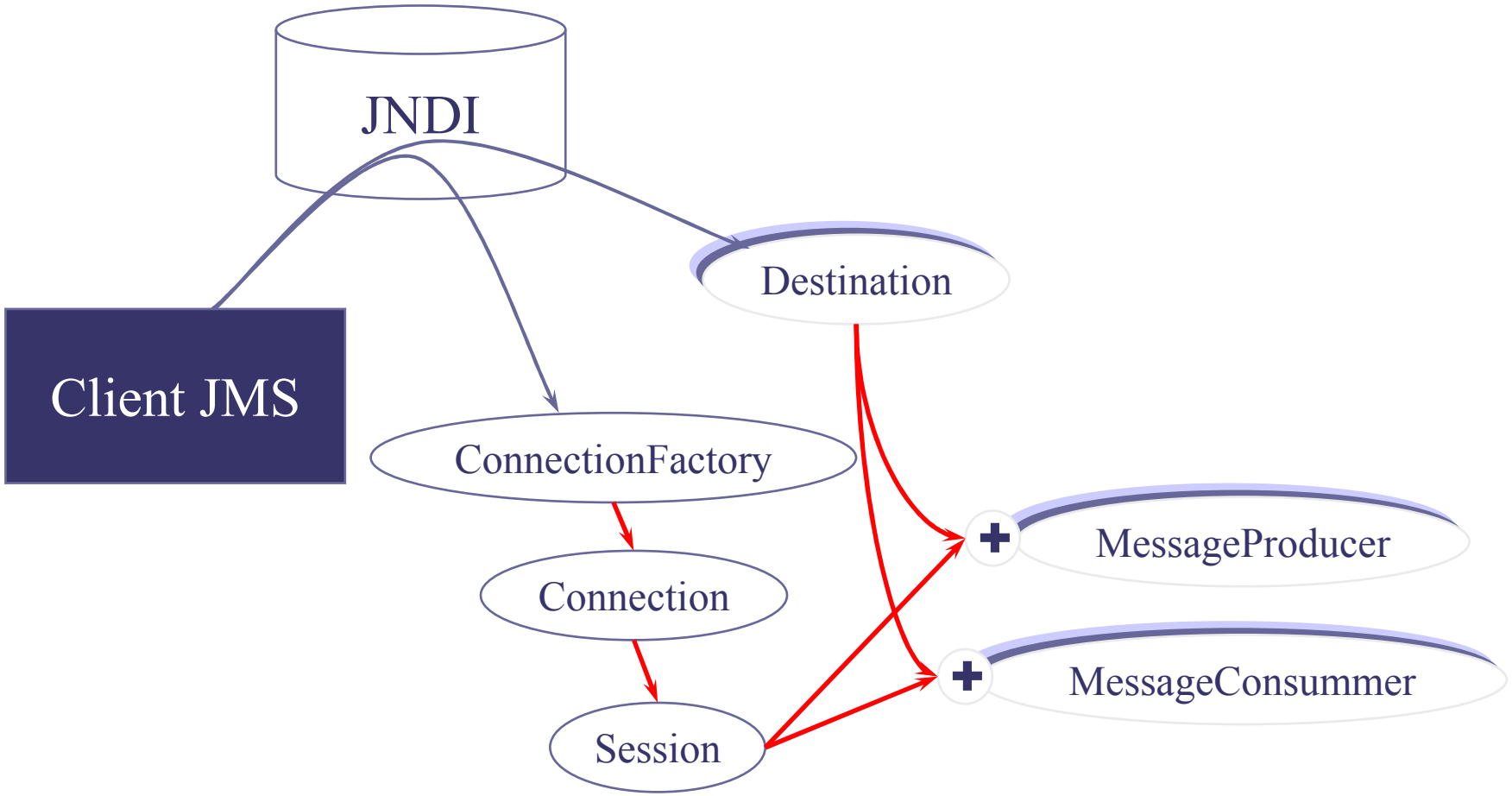
Les objets JMS

- **Objets administrés**
  - ConnectionFactory : point d'accès à un serveur MOM
  - Destination : Queue ou Topic
- **Connection**
  - Authentifie le client et encapsule la liaison avec le provider
  - Gère les sessions et l'ExceptionListener
- **Session**
  - Fournit un contexte mono-threadé de production/consommation de messages
  - Gère les destinations temporaires, sérialise l'exécution des MessageListener, les acquittements de messages et les transactions

Les objets JMS

- **MessageProducer**
  - Fabriqué par la session → QueueSender, TopicPublisher
  - Permet l'émission de message → send, publish
- **MessageConsumer**
  - Fabriqué par la session → QueueReceiver, TopicSubscriber
  - Permet la réception de message
    - *Synchrone → receive*
    - *Asynchrone → MessageListener*
  - Permet le filtrage des messages

Architecture



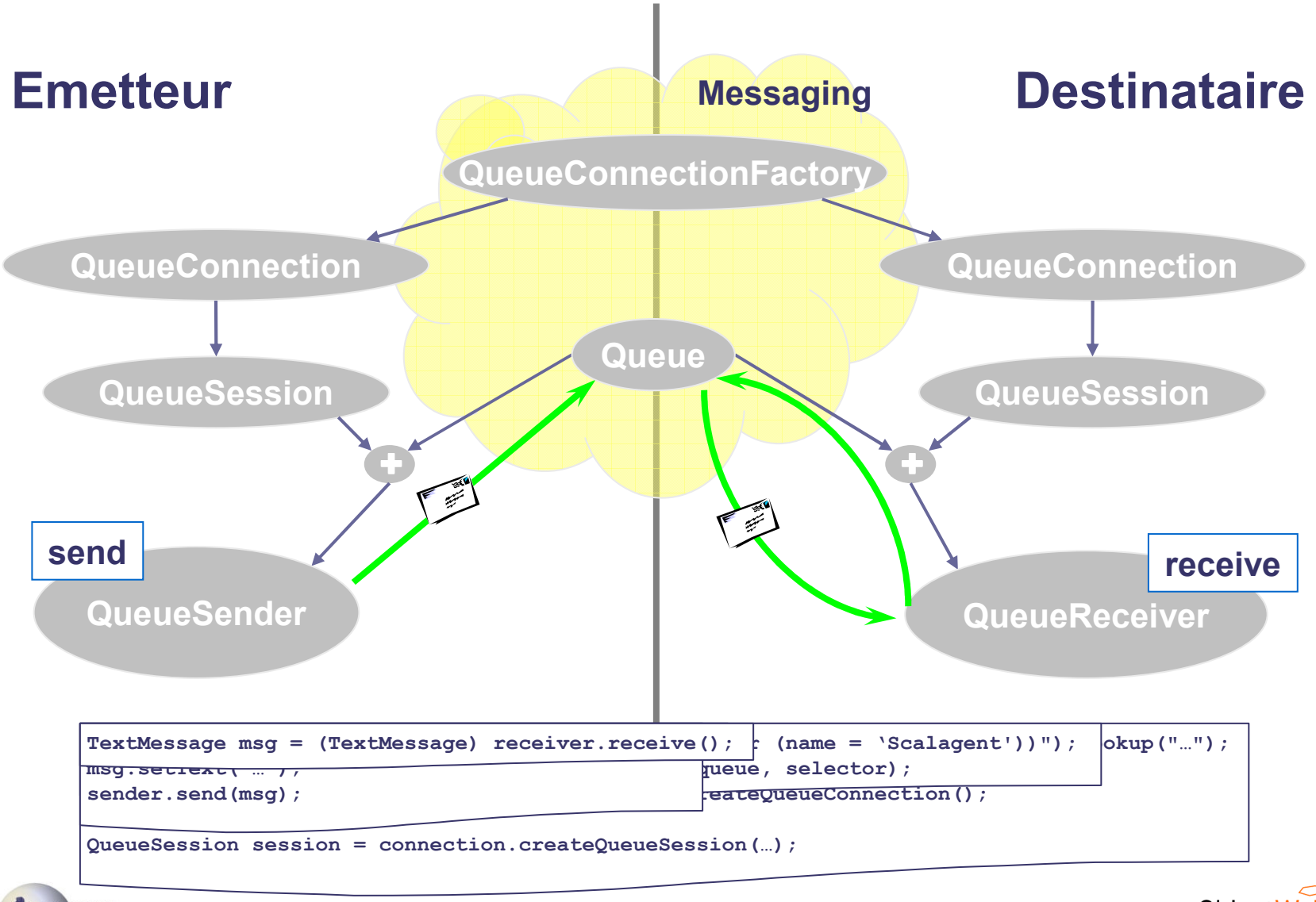
Le message JMS

- **Entête**
  - JMSMessageId, JMSDestination, JMSDeliveryMode, JMSExpiration, JMSPriority, etc.
- **Propriétés**
  - Couple <nom, valeur>
- **Corps**
  - TextMessage, MapMessage
  - StreamMessage, ObjectMessage
  - BytesMessage

JMS Domain « Point-to-Point »

- **Un message émis sur une queue de messages donnée est consommé par une unique application**
  - asynchronisme et fiabilité
- **1 ! Destinataire**
- **Indépendance de l'émetteur et du destinataire**
  - Anonymat → Evolution
  - Indépendance temporelle
- **Acquittement du traitement par le destinataire**

JMS - "Point-to-Point"



JMS - "Point-to-Point"

```
QueueConnectionFactory connectionFactory = (QueueConnectionFactory) messaging.lookup("…");
Queue queue = (Queue) messaging.lookup("…");
QueueConnection connection = connectionFactory.createQueueConnection();
connection.start();
QueueSession session = connection.createQueueSession(…);
```

```
QueueSender sender = session.createSender(queue);
```

```
String selector = new String("(name = 'ObjectWeb') or (name = 'Scalagent'))");
QueueReceiver receiver = session.createReceiver(queue, selector);
```

```
TextMessage msg = session.createTextMessage();
msg.setText("…");
sender.send(msg);
```

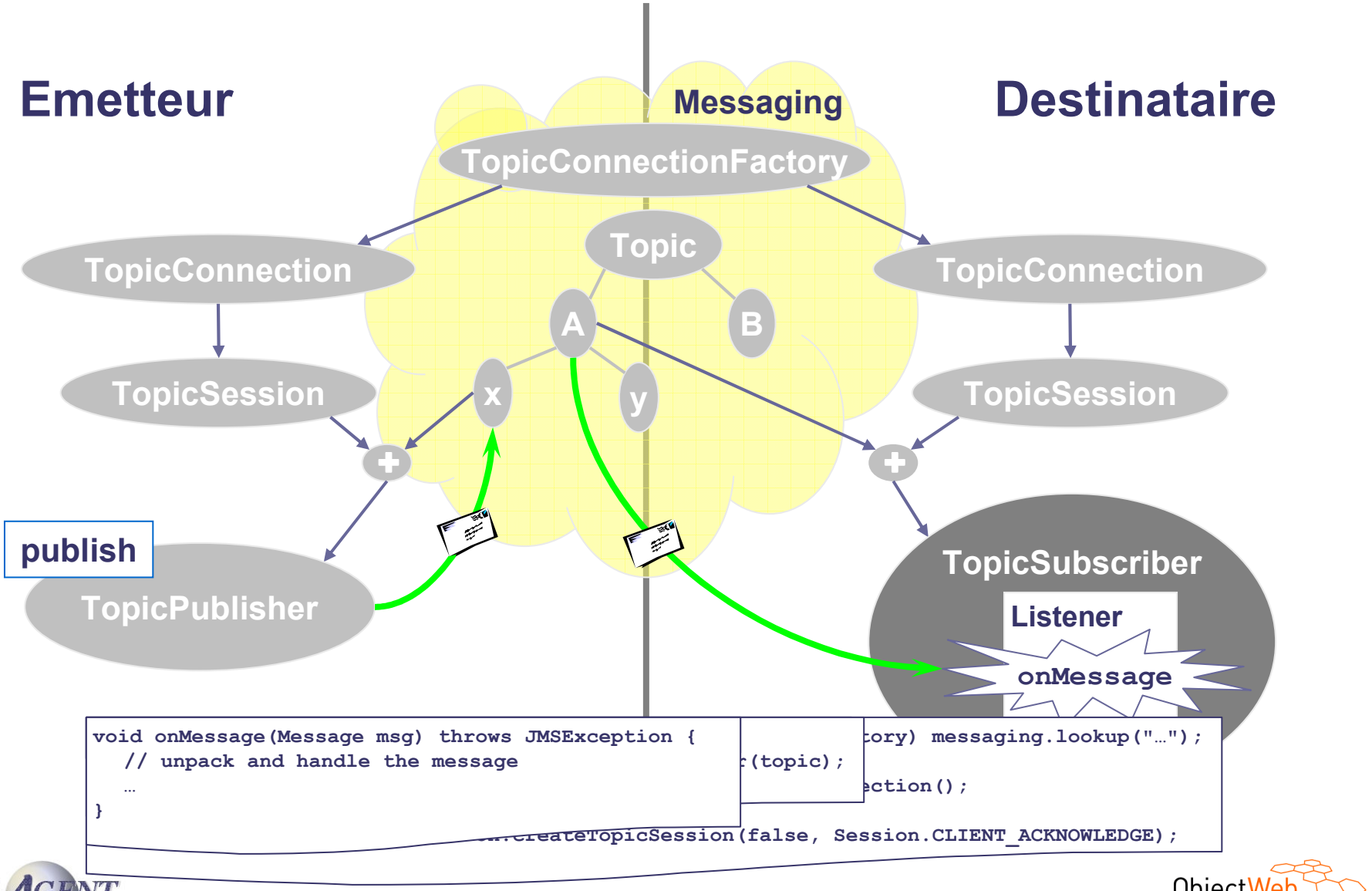
```
TextMessage msg = (TextMessage) receiver.receive();
```

JMS Domain « Publish/Subscribe »

- **Un message émis sur un sujet (*Topic)* donné est délivré à l'ensemble des applications abonnées à ce *Topic*.**

- **Multiples destinataires**
  - Anonymat
  - Dépendance temporelle
- **Critères d'abonnement**
  - "subject based" versus "content based"
  - Organisation hiérarchique
- **Abonnements persistants**

JMS - "Publish/Subscribe"



JMS - "Publish/Subscribe"

```
TopicConnectionFactory connectionFactory = (TopicConnectionFactory) messaging.lookup("…");
Topic topic = (Topic) messaging.lookup("/A/x");
TopicConnection connection = connectionFactory.createTopicConnection();
connection.start();
TopicSession session = connection.createTopicSession(false, Session.CLIENT_ACKNOWLEDGE);
```

```
TopicPublisher publisher = session.createPublisher(topic);
```

```
Topic topic = (Topic) messaging.lookup("/A");
TopicSubscriber subscriber = session.createSubscriber(topic);
Subscriber.setMessageListener(listener);
```

```
publisher.publish(msg);
```

```
void onMessage(Message msg) throws JMSException {
   // unpack and handle the message
   …
}
```

Le composant JORAM

- **Implantation open source de l'API cliente JMS**
  - Disponible sur ObjectWeb : http://joram.objectweb.org
- **Usage double**
  - Service de messagerie autonome pour applications Java
  - Composant de messagerie asynchrone intégré dans un serveur d'application J2EE (JonAS, JBoss, etc.)
- **Basé sur le MOM ScalAgent**
  - Technologie à base d'agents
    - *Comportement Transactionnel*
    - *Architecture distribuée*

Le composant JORAM

- **Joram implémente la dernière spécification JMS 1.1**
  - Topics hiérarchique, queues et topics clusterisés
  - DeadMessageQueue
  - Support de SOAP / XML
  - Client léger (J2ME) pour périphérique portable, client C++
  - Persistance fichier et BD
  - Outils d'administration et support JMX
- **Joram est la solution JMS intégrée dans JOnAS**
  - Officiellement certifié J2EE 1.4 (JMS1.1)

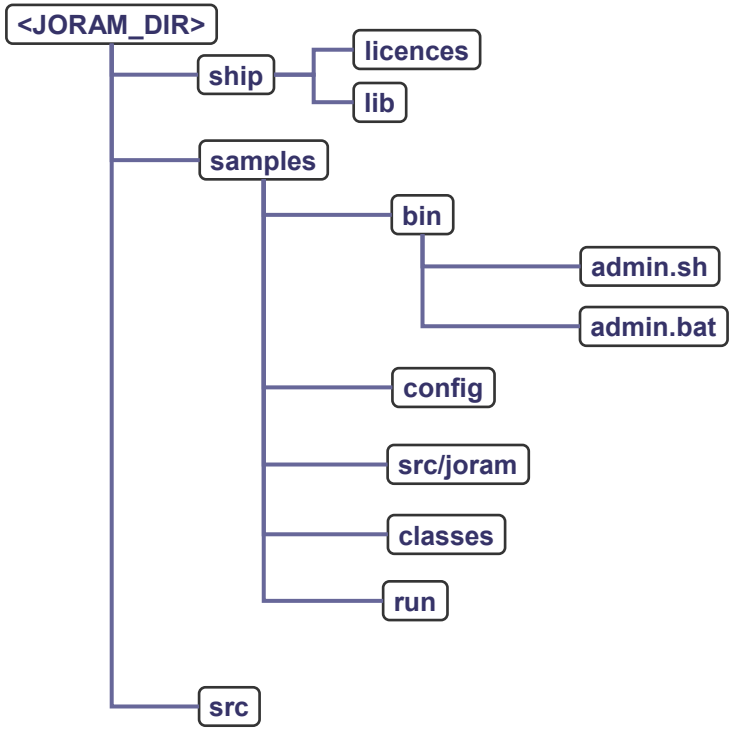
Installation

- **Choix du répertoire d'installation de Joram: DIR**
  - JORAM_DIR=DIR/joram-4.3.x
- **Décompression de la livraison**
  - Windows → WinZip …

    joram-4.3.x.tgz → **Création du répertoire `joram-4.3.x`**
  - Unix :

```
cd <DIR>
gunzip -c <LIVR_DIR>/joram-4.3.x.tgz | tar xvf -
```

Installation



Configuration

Adaptation des scripts

- répertoire *<JORAM_DIR>/samples/bin*
- Variable d'environnement JORAM_HOME
  - *Windows :* `set JORAM_HOME=C:\Joram4.3.x`
  - *Unix :* `export JORAM_HOME=/home/joram4.3.x`
- Variable d'environnement JAVA_HOME
  - *Windows :*
    - `set JAVA_HOME=C:\j2sdk1.4.2`
    - `%JAVA_HOME%\bin\java -fullversion`
  - *Unix :*
    - `export JAVA_HOME=/usr/local/j2sdk1.4.2`
    - `$JAVA_HOME/bin/java -fullversion`

JORAM – Interface JMS du MOM ScalAgent

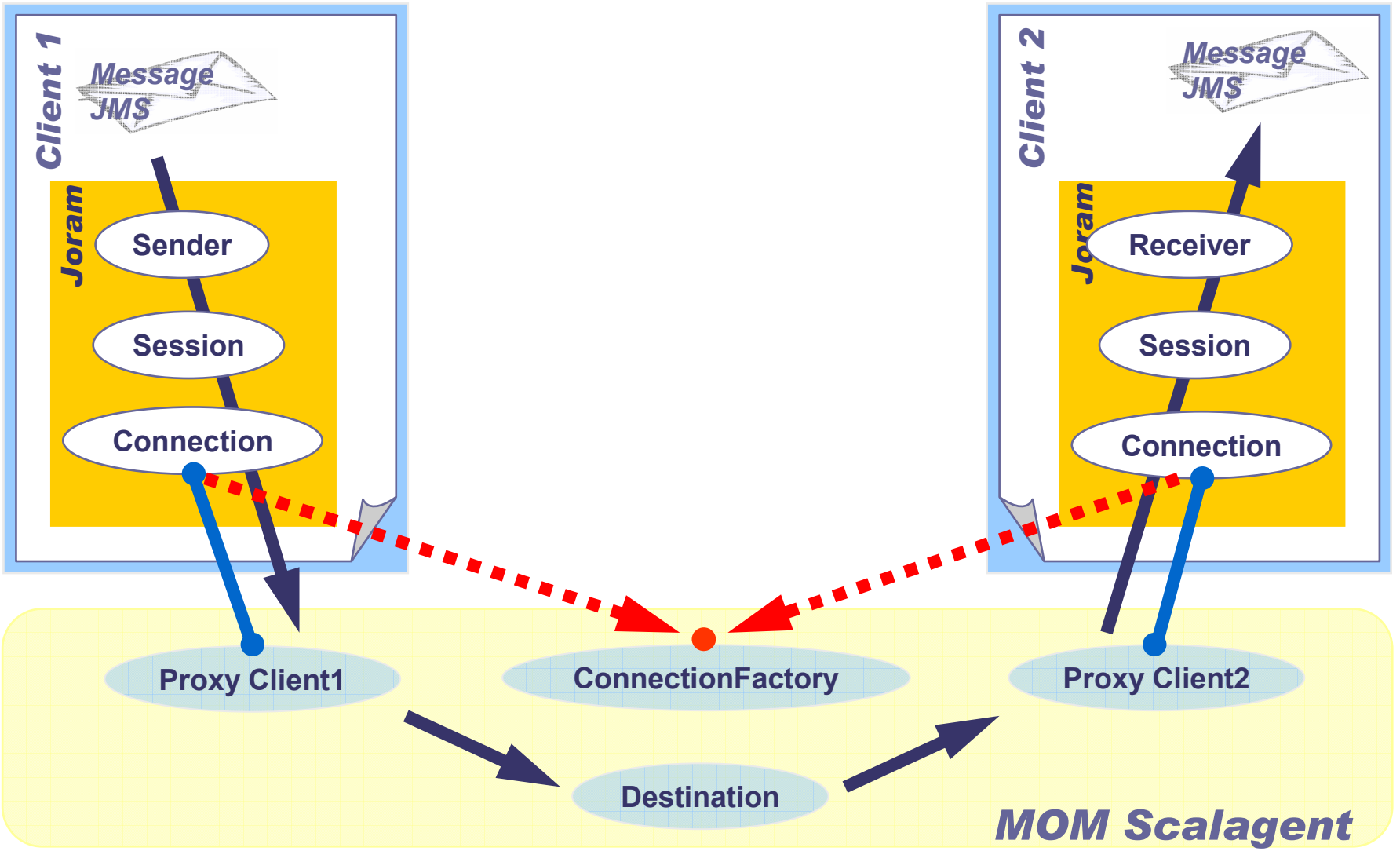
- **Les queues et topics sont des agents**
- **Les messages sont encapsulés dans des notifications**
  - Les messages échangés par les clients JMS transitent via le MOM
- **Un agent « ConnectionManager » sur chaque nœud**
  - Gestion des utilisateurs
  - Mise en place des connections

Déployé en tant que service → Configuration

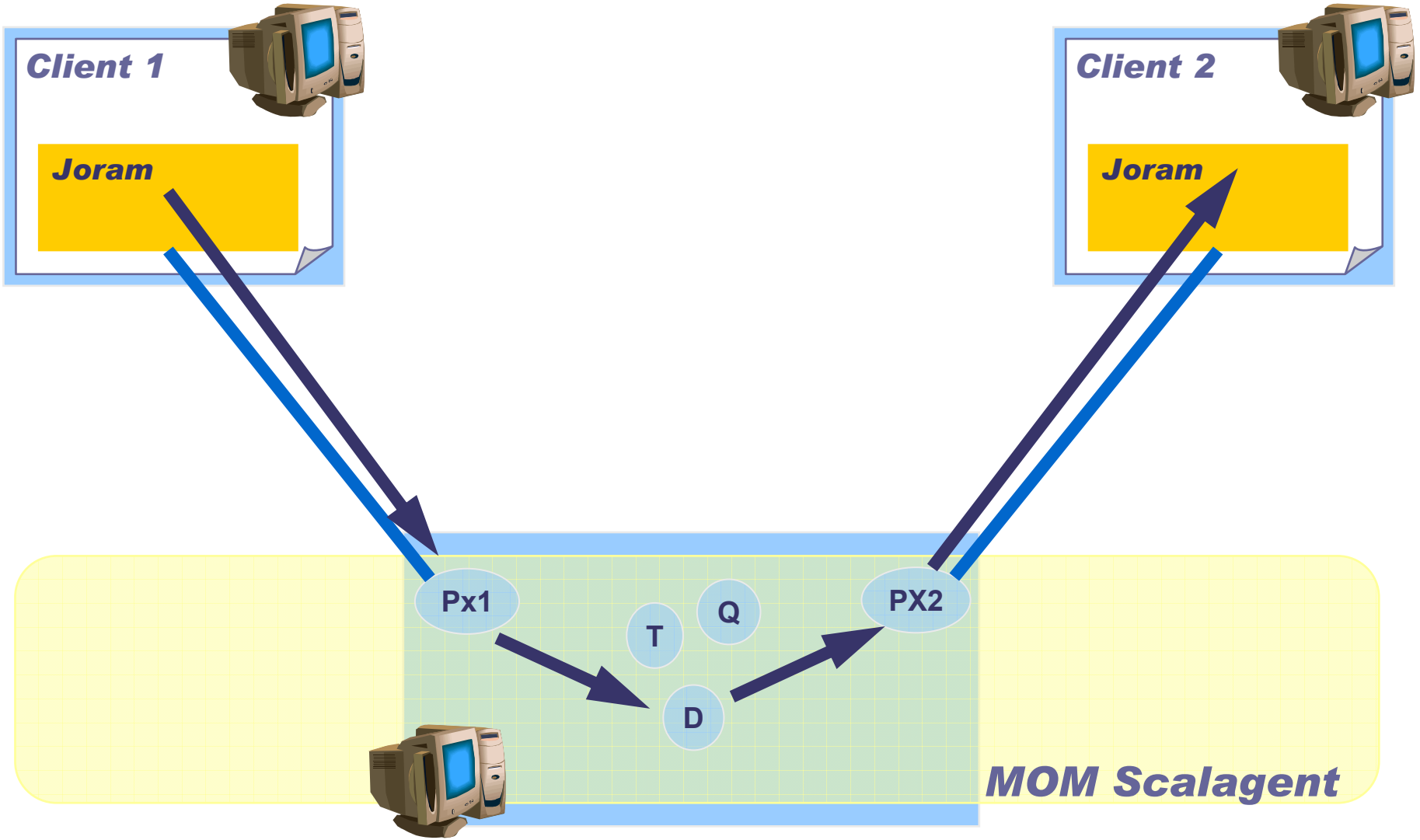
- **Chaque « client JMS » est représenté par un agent**
  - Gestion de la connection, dialogue avec les destinations

- **L'architecture est naturellement distribuée**

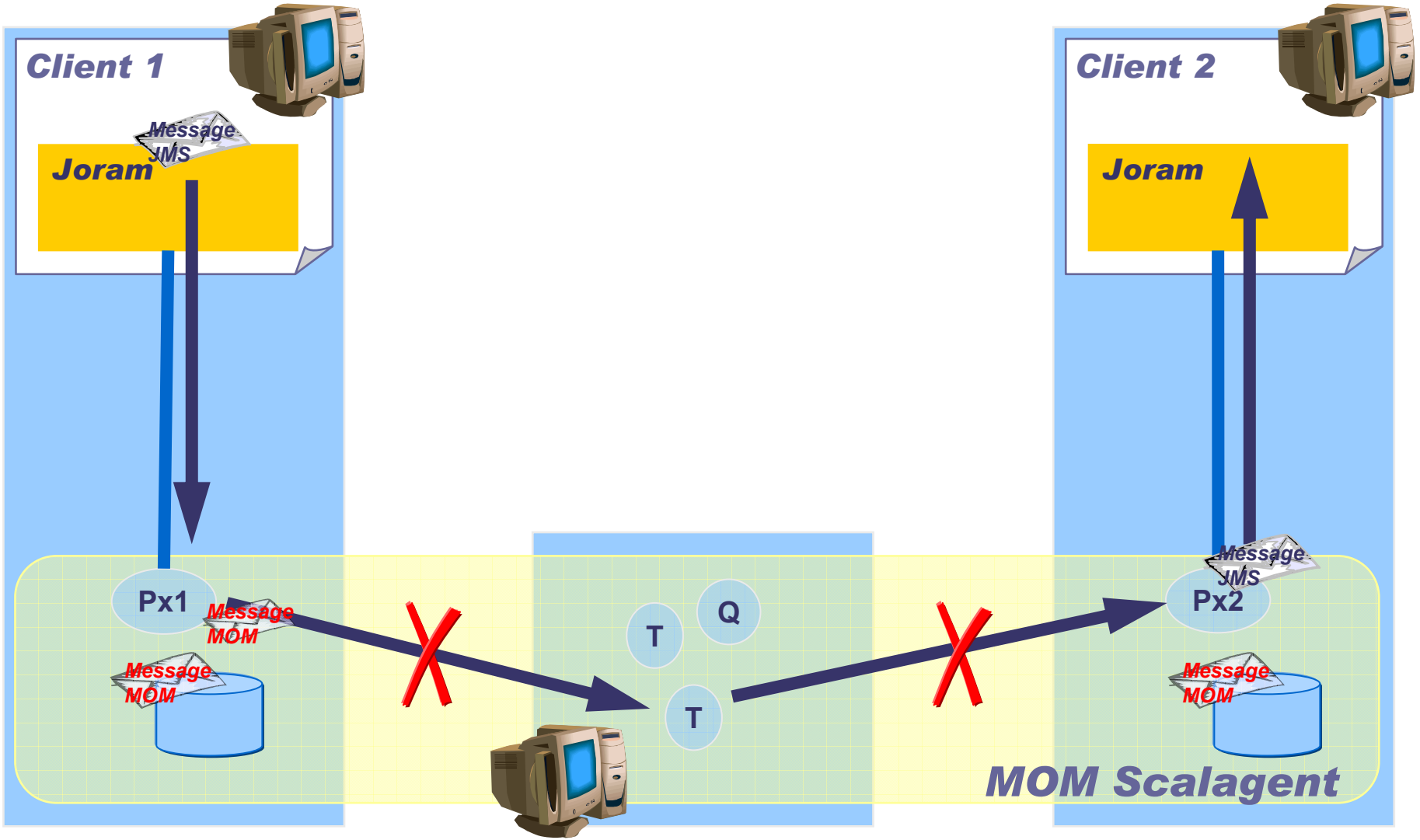
Joram – Architecture logique



Joram – Architecture centralisée



Joram – Architecture distribuée



Joram - Administration

Au travers de JMS

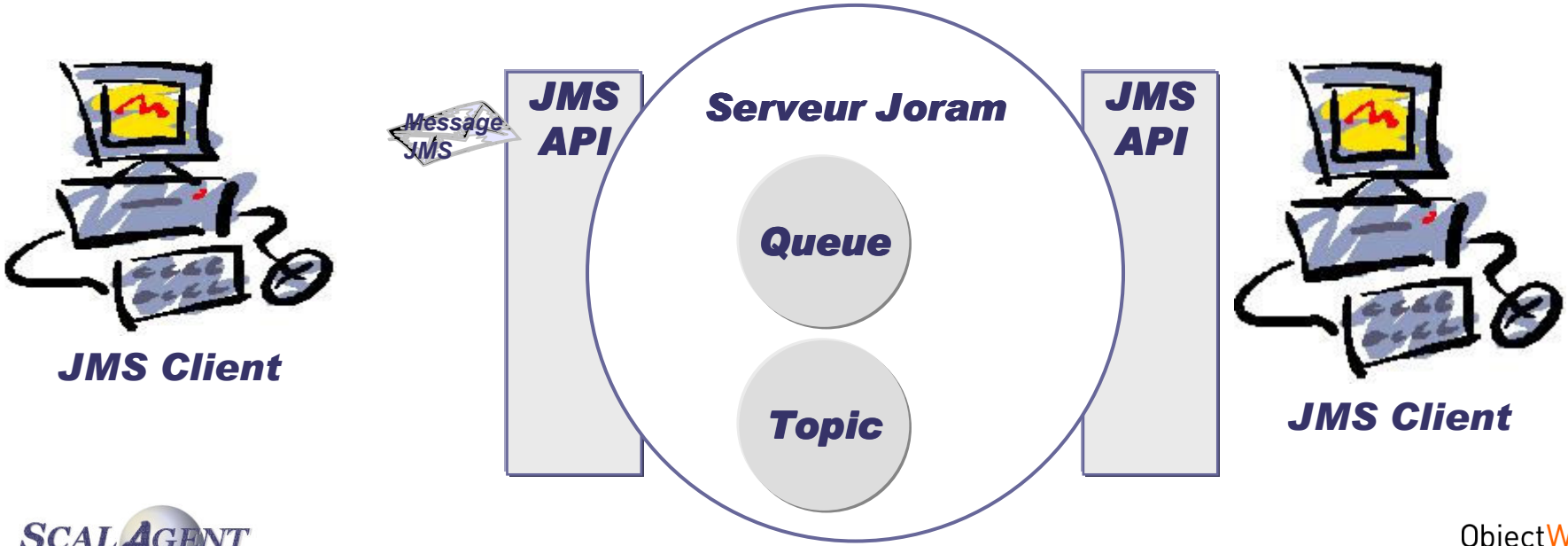
- API client d'administration : AdminModule
  - *Dialogue au travers de Message JMS*
- 1 Topic d'administration sur chaque serveur
- Utilisé par l'outils graphique JAMT et l'interpréteur de script XML

Au travers de JMX :

- Objets « AgentServer »
- Objets « Joram »
  - *Proxy, Queue et Topic*

Un exemple simple

- Echange de messages entre un producteur et un consommateur
  - Point-to-Point : Queue
  - Publish/Subscribe : Topic



Un exemple simple

- **Compilation des exemples**

```
cd samples/src/joram
ant clean compile
```

- **Configuration du serveur**

```
samples/config/centralized_a3servers.xml
```

- **Lancement du serveur Joram**

```
ant reset single_server
```

```
../../bin/single_server.sh
..\..\bin\single_server.bat
```

Un exemple simple - Configuration

centralized_a3servers.xml

```
<?xml version="1.0"?>
<config>
  <server id="0" name="S0" hostname="localhost">
    <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
             args="root root"/>
    <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"
             args="16010"/>
    <service class="fr.dyade.aaa.jndi2.server.JndiServer" args="16400"/>
  </server>
</config>
```

Un exemple simple

- **Configuration de l'application**

  Programme d'administration utilisant des méthodes de l'API Joram (hors JMS)

  ```
  ant classic_admin
  ```

  ➔ Création de la queue, du topic

  ➔ Enregistrement des *objets administrés* dans JNDI

- **Lancement du GUI d'administration**

  ```
  ../../bin/admin.sh
  ..\..\bin\admin.bat
  ```

  ```
  ant admin_gui
  ```

Un exemple simple - Administration

```
import org.objectweb.joram.client.jms.admin.*;
import org.objectweb.joram.client.jms.*;

public class ClassicAdmin {
  public static void main(String[] args) throws Exception {
    AdminModule.connect("root", "root", 60);
    jndiCtx = new javax.naming.InitialContext();

    cf = TcpConnectionFactory.create("localhost", 16010);
    qcf = QueueTcpConnectionFactory.create("localhost", 16010);
    tcf = TopicTcpConnectionFactory.create("localhost", 16010);

    jndiCtx.bind("cf", cf);
    jndiCtx.bind("qcf", qcf);
    jndiCtx.bind("tcf", tcf);

    …
```

Un exemple simple - Administration

```
    …

    User user = User.create("anonymous", "anonymous");

    Queue queue = (Queue) Queue.create("queue");
    queue.setFreeReading();
    queue.setFreeWriting();
    jndiCtx.bind("queue", queue);

    Topic topic = (Topic) Topic.create("topic");
    topic.setFreeReading();
    topic.setFreeWriting();
    jndiCtx.bind("topic", topic);

    jndiCtx.close();
    AdminModule.disconnect();
  }
}
```

Un exemple simple

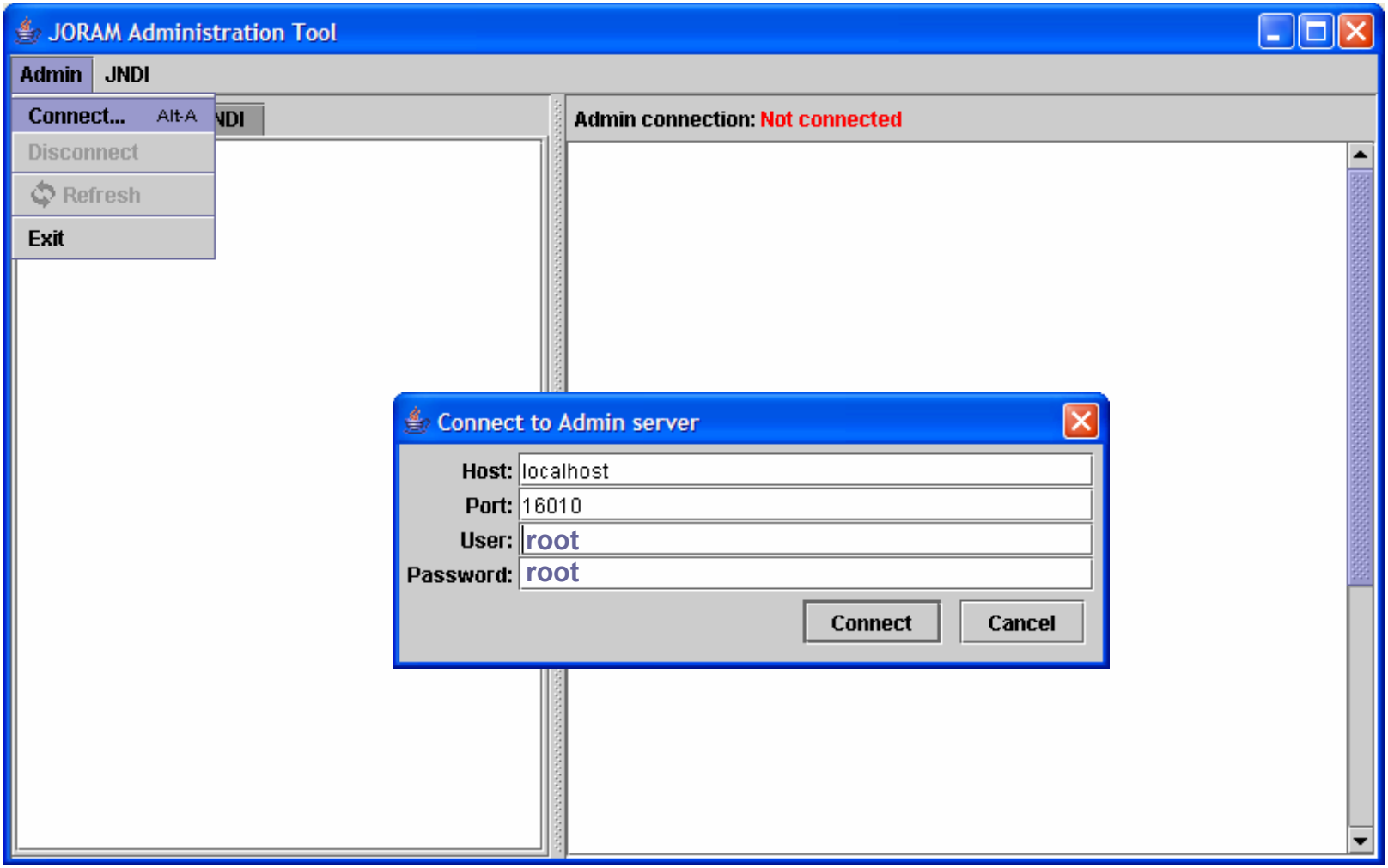
- **Point-to-Point**

```
ant sender
ant browser
ant receiver
```

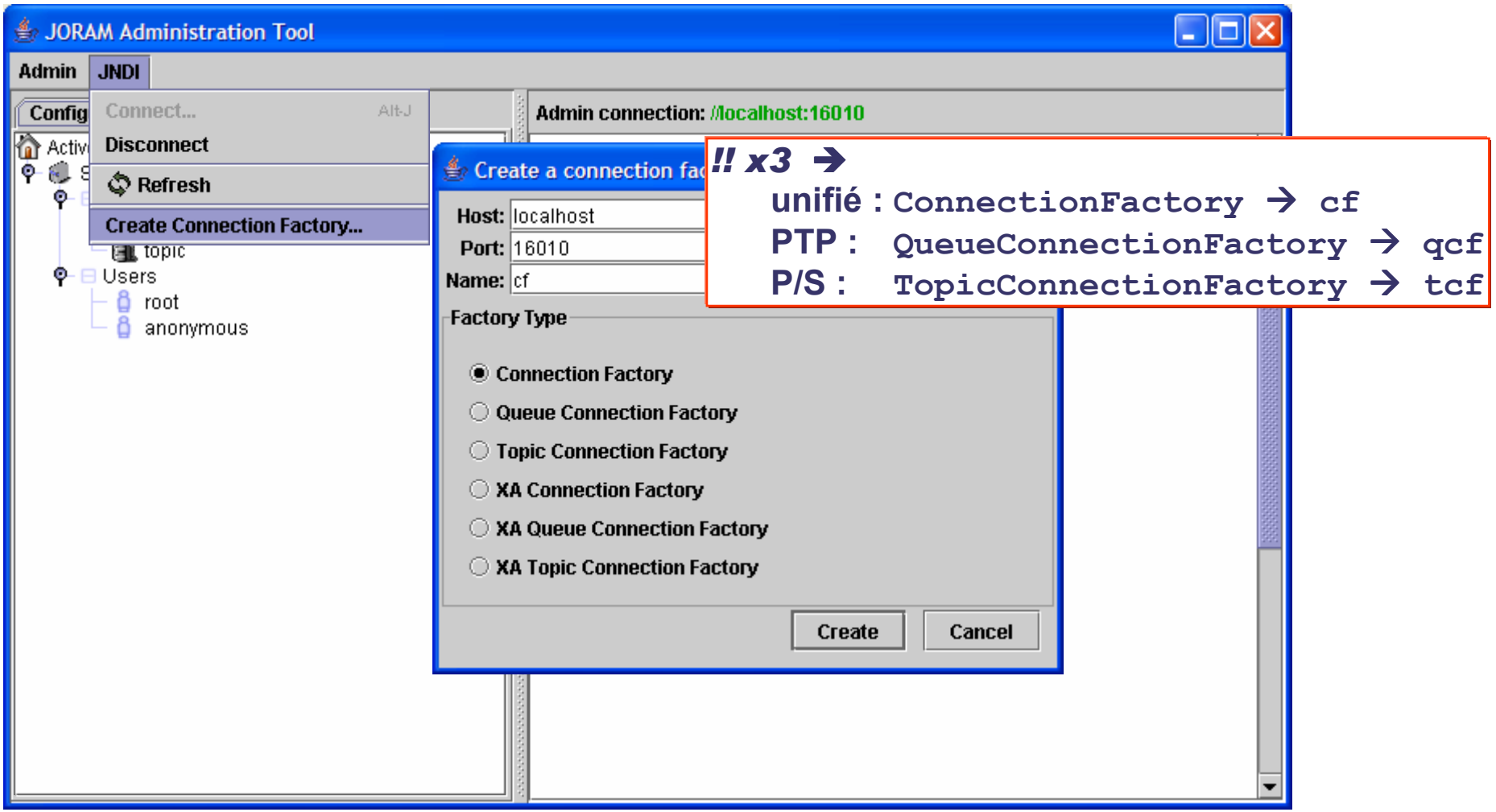
- **Publish/Subscribe** ! Abonnement non durable ➔ dépendance temporelle

```
ant subscriber
ant publisher
```

Un exemple simple - Administration



Un exemple simple - Administration



Un exemple simple - JMX

- **Options de lancement de la JVM :**

  -Dcom.sun.management.jmxremote ➔Activation du serveur JMX dans la JVM

  -DMXServer=com.scalagent.jmx.JMXServer ➔Enregistrement des MBeans

- **Lancement de la console :**

  jconsole

Architecture distribuée

- **Motivations**
  - Distribution inhérente à l'application
  - Fiabilité → serveurs, liens réseaux
  - Répartition de charge

→ **Distribution du proxy utilisateur**

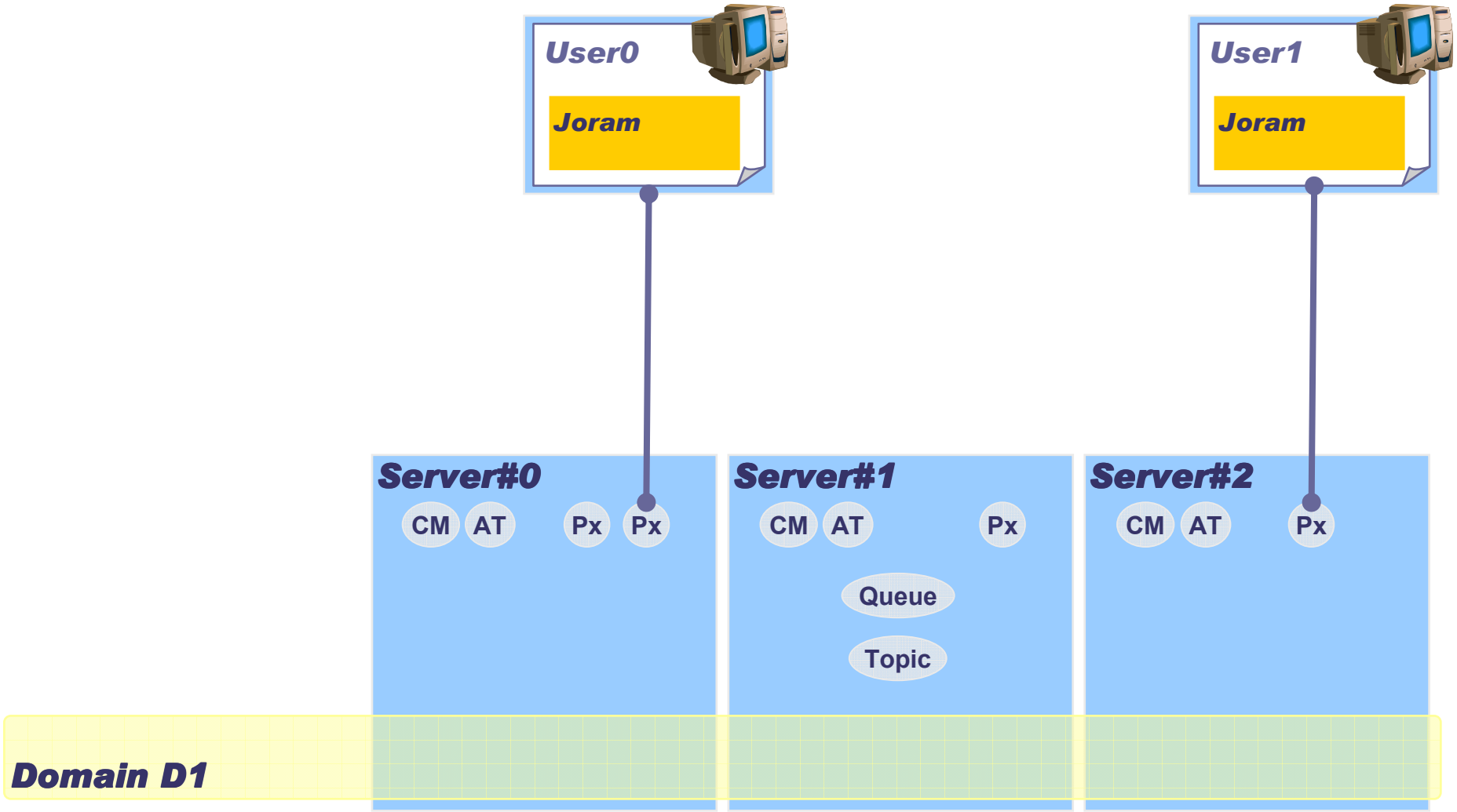
→ Store and Forward

→ **Placement des queues de messages**

→ **Utilisation de destinations distribués**

→ Queues et topics cluster, topics hiérarchiques

Architecture distribuée



Configuration distribuée

- **Configuration du serveur**

  `samples/config/distributed_a3servers.xml`

- **Lancement du serveur**

  `ant reset servers (e.g server[0,1,2])`

- **Lancement du code d'administration**

  `ant admin_gui`

Configuration distribuée

distributed_a3servers.xml

```
<?xml version="1.0"?>
<config>
  <domain name="D1"/>

  <server id="0" name="S0" hostname="localhost">
    <network domain="D1" port="16300"/>
    <service class="org.objectweb.joram.mom.proxies.ConnectionManager"
             args="root root"/>
    <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"
             args="16010"/>
    <service class="fr.dyade.aaa.jndi2.server.JndiServer" args="16400"/>
  </server>

  <server id="1" name="S1" hostname="localhost">
    <network domain="D1" port="16301"/>
    <service class="org.objectweb.joram.mom.proxies.ConnectionManager"/>
    <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"
             args="16011"/>
  </server>
  …
```

Configuration distribuée

distributed_a3servers.xml

```
  …
  <server id="2" name="Nasdaq" hostname="localhost">
    <network domain="D1" port="16302"/>
    <service class="org.objectweb.joram.mom.proxies.ConnectionManager"/>
    <service class="org.objectweb.joram.mom.proxies.tcp.TcpProxyService"
             args="16012"/>
  </server>

</config>
```

Configuration distribuée - Administration

```
…

cf0 = TcpConnectionFactory.create("localhost", 16010);
cf2 = TcpConnectionFactory.create("localhost", 16012);

jndiCtx.bind("cf0", cf0);
jndiCtx.bind("cf2", cf2);

…

…

User user0 = User.create("anonymous", "anonymous", 0);
User user2 = User.create("anonymous", "anonymous", 2);

Queue queue = (Queue) Queue.create("queue", 1);
Topic topic = (Topic) Topic.create("topic", 1);

…
```

Configuration distribuée

- **Administration automatique**

```
ant archi_admin
```

- **Mode PTP**

```
ant archi_sender
ant archi_receiver
```

- **Mode P/S**

```
ant archi_sub
ant archi_pub
```