ScalAgent Distributed Technologies

MOM: Concepts & Applications

André Freyssinet
ScalAgent Distributed Technologies

Andre.Freyssinet@scalagent.com

www.scalagent.com

Plan

- **Introduction**
- **Exemple d'application (supervision)**
  - Synchrone vs Asynchrone
- **Message Oriented Middleware**
  - Modèle
  - Architecture
  - Exemples:MQSeries, SonicMQ, etc.
- **Java Message Service : une API Java pour le MOM**
  - Description
  - Utilisation

Plan (2)

- **Joram: Une implantation Open-Source de l'API JMS**
  - Présentation
  - Le MOM ScalAgent
  - Implémentation
- **Exemple et démonstration**
  - MGE-UPS : Une application de supervision dans le domaine de la distribution électrique

Introduction

- **Distribution, Intégration**
  - ➔ Application = ens. Distribué de composants
- **Du synchrone (RPC, RMI, etc.) …**
- **… à l'échange de messages asynchrone.**
  - Indépendance
  - Evolution
- **Architecture « faiblement couplée »**

Exemple - Supervision

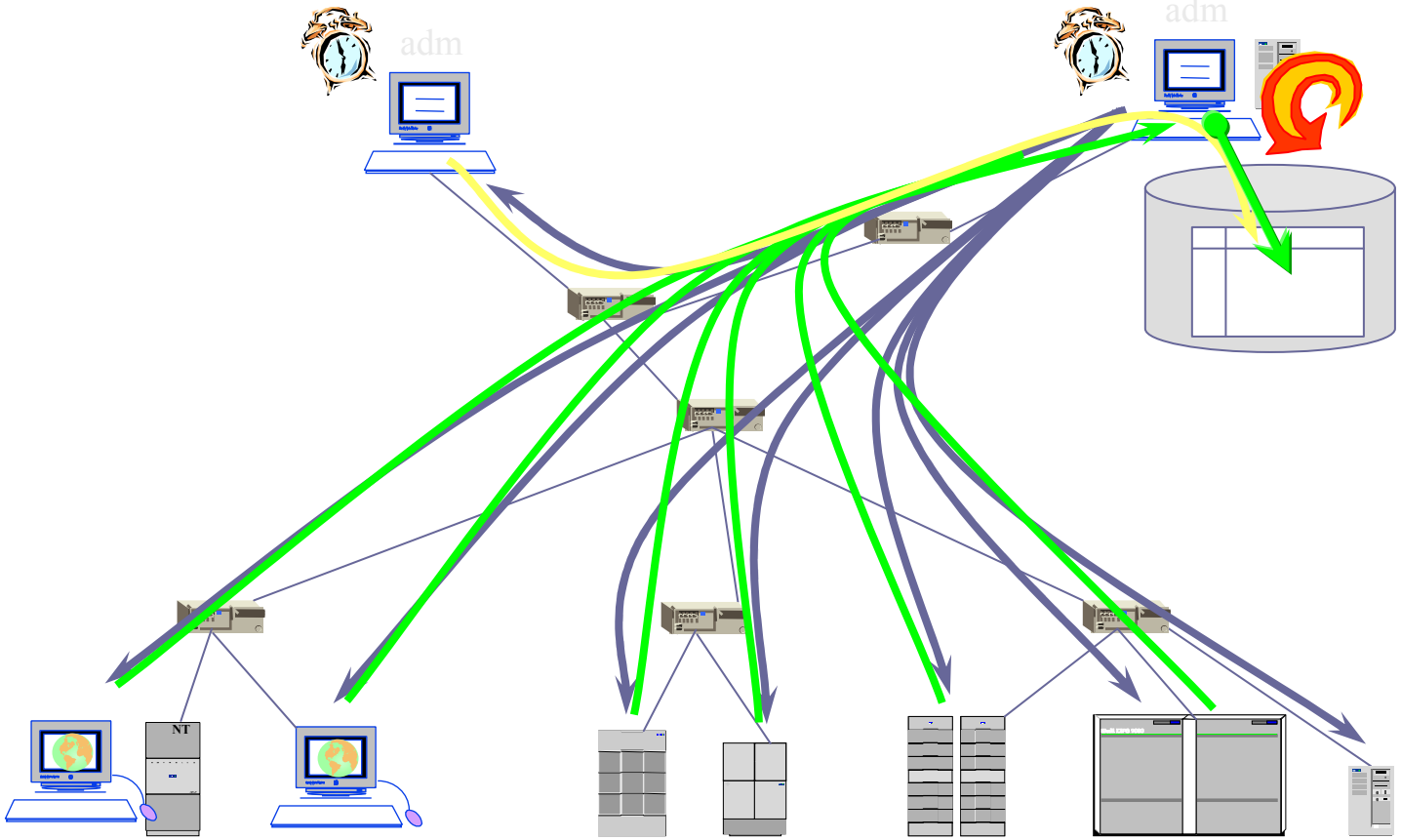
- Surveillance de l'état de machines, de systèmes d'exploitation et d'applications dans un environnement distribué.
- Flot continuel de données en provenance de sources diverses sur le réseau.
- Les éléments du système peuvent apparaître, disparaître, migrer, etc.
- Les administrateurs doivent pouvoir accéder à l'information quel que soit leur localisation

Solution traditionnelle client / serveur

- **Interrogation régulière des éléments à surveiller par l'application d'administration et mise à jour d'une base de données centralisée.**
  - Utilisation d'une configuration complexe afin de connaître l'ensemble des éléments à surveiller.
  - Maintien de cette configuration lorsque des machines ou des applications rejoignent, quittent ou se déplacent dans le système.

- **Interrogation par les administrateurs de la base centrale.**

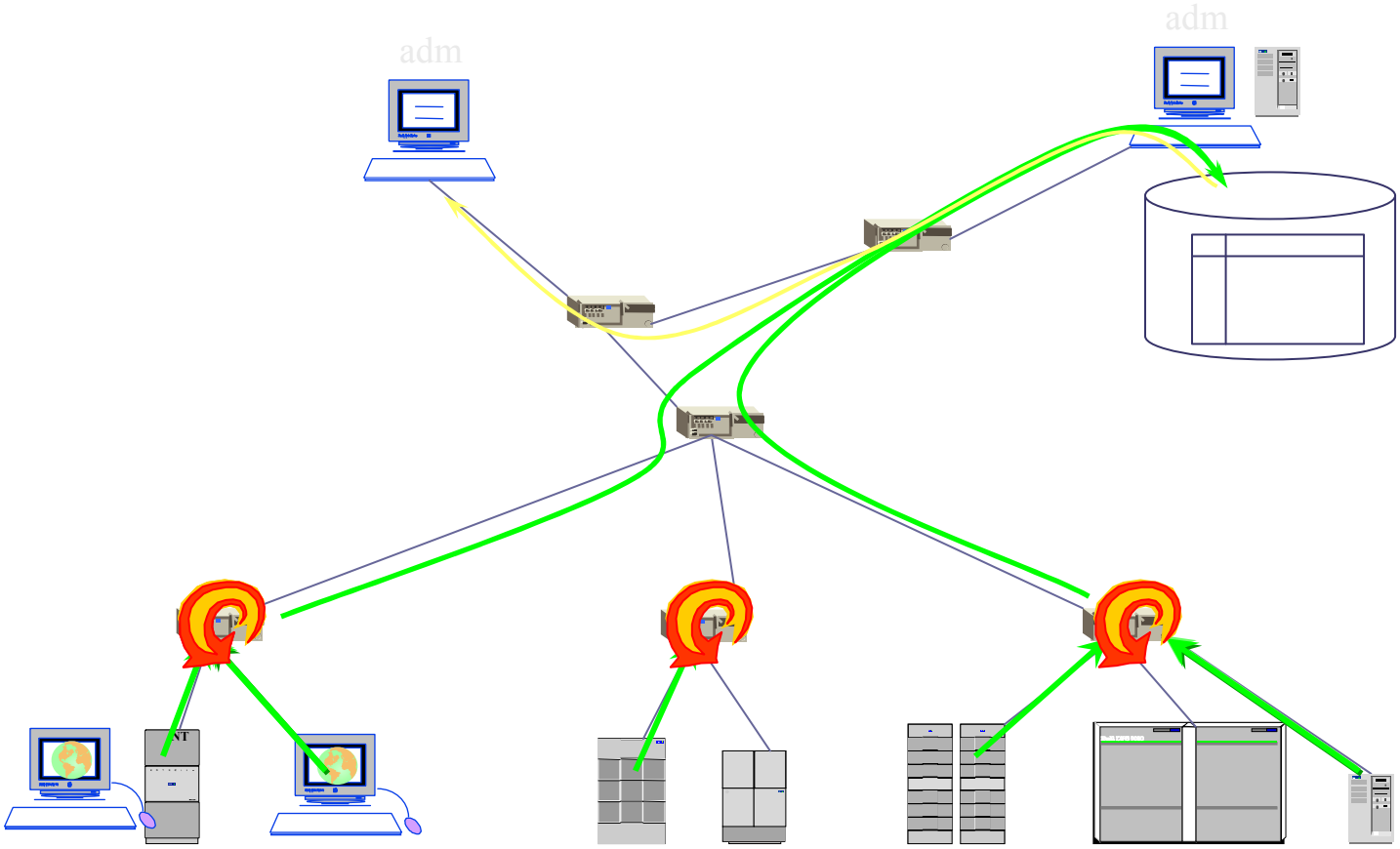
Solution traditionnelle client / serveur



Solution « Messaging »

- **Les différents éléments administrés émettent des messages :**
  - changements d'état et de configuration
  - alertes, statistiques
- **Un ou plusieurs démons reçoivent ses notifications et maintiennent l'état courant du système**
  - suivi des changements de configuration dynamiques
  - émission de messages signalant les changements d'états significatifs ou les mises à jour
- **Inversion des rôles des producteurs et des consommateurs de données**

Solution « Messaging »



Révolution ? pas vraiment !!

- **Années 70/80 : IBM MQ Series, etc.**
- **Internet et Asynchronisme**
  - Le courrier électronique (communication PTP)
    - *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 (Anonymat, Publish/Subscribe, Push)
    - *le consommateur s'abonne à une liste de diffusion, le producteur publie un message sur la liste (communication anonyme)*
    - *Le message est diffusé à tous les abonnés*
  - Les news (Anonymat, Pull)
    - *le producteur publie une information dans un forum*
    - *le consommateur va lire le contenu du forum quand il le souhaite*

Principes directeurs

- **Couplage faible de l'émetteur et du destinataire**
  - Communication asynchrone
    - *« Store And Forward »*
  - Communication indirecte
    - *Mode de désignation*
- **Persistance et fiabilité**
- **Messages typés**
- **Gestion de l'hétérogénéité**
  - Des données, des systèmes et des systèmes de communication.

Format des messages

- **Entête :**
  - Information permettant l'identification et l'acheminement du message
  - Id. unique, Destination, Priorité, durée de vie, etc.
- **Attributs :**
  - Couples (nom, valeur) utilisables par le système ou l'application pour sélectionner les messages
- **Données Définies par l'application**

Modes de désignation

- **Communication de groupe**
  - groupe = ensemble de récipiendaires identifiés par un nom unique
    - *gestion dynamique du groupe : arrivée/départ de membres*
    - *différentes politiques de service dans le groupe : 1/N, N/N*
- **Communication anonyme**
  - désignation associative : les destinataires d'un message sont identifiés par leurs propriétés et non par leur nom
- **applications :**
  - tolérance aux fautes (gestion de la réplication), travail coopératif

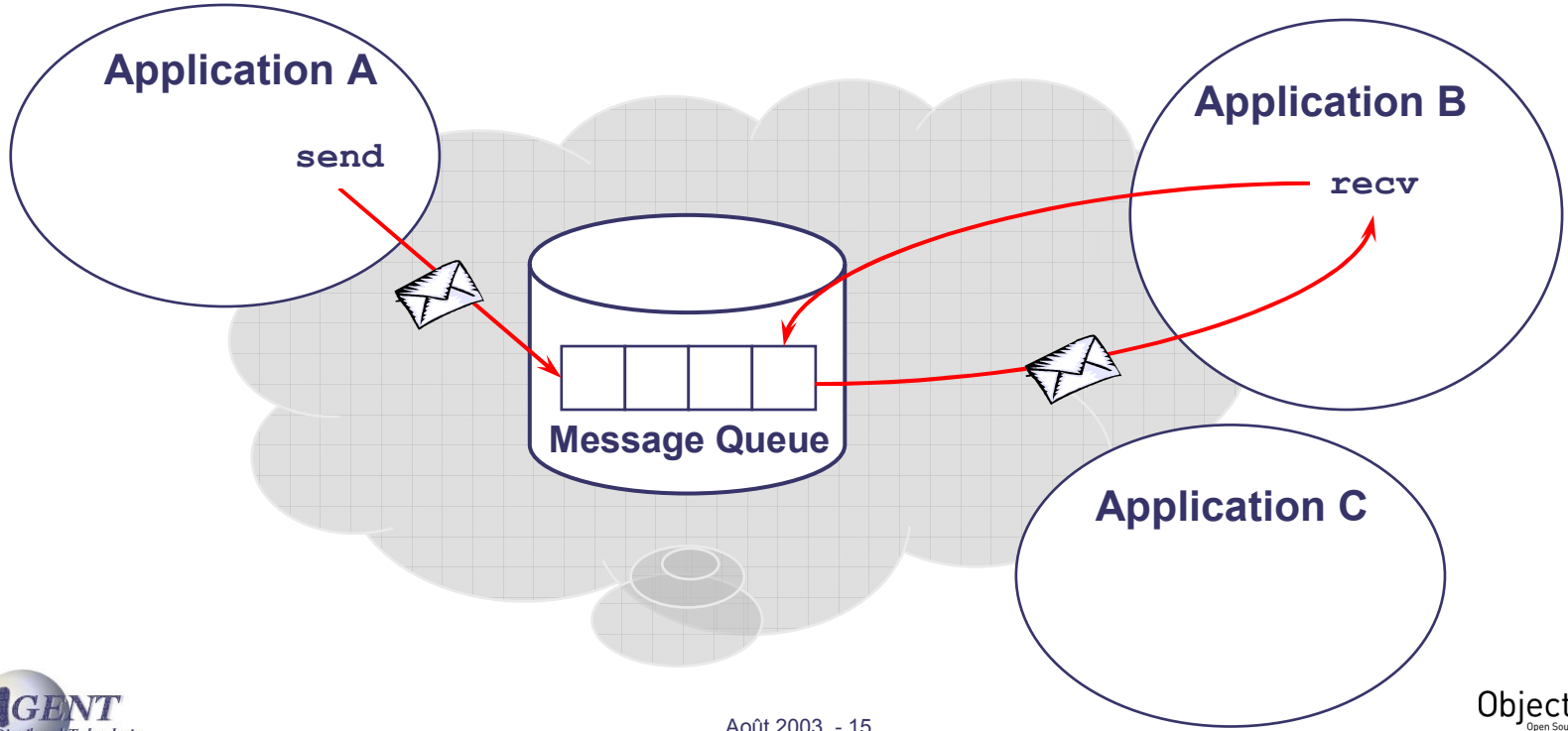
Modes de délivrance

- **4 Relations entre producteur et consommateur**
  - 1 producteur → 1 consommateur
  - 1 producteur → N consommateurs
  - N producteur → 1 consommateur
  - N producteurs → N consommateurs

- **2 Modéles**
  - Point-To-Point
  - Publish/Subscribe

Modèle « Point-to-Point »

- Un message émis sur une queue de messages donnée est consommé par une unique application
  - asynchronisme et fiabilité

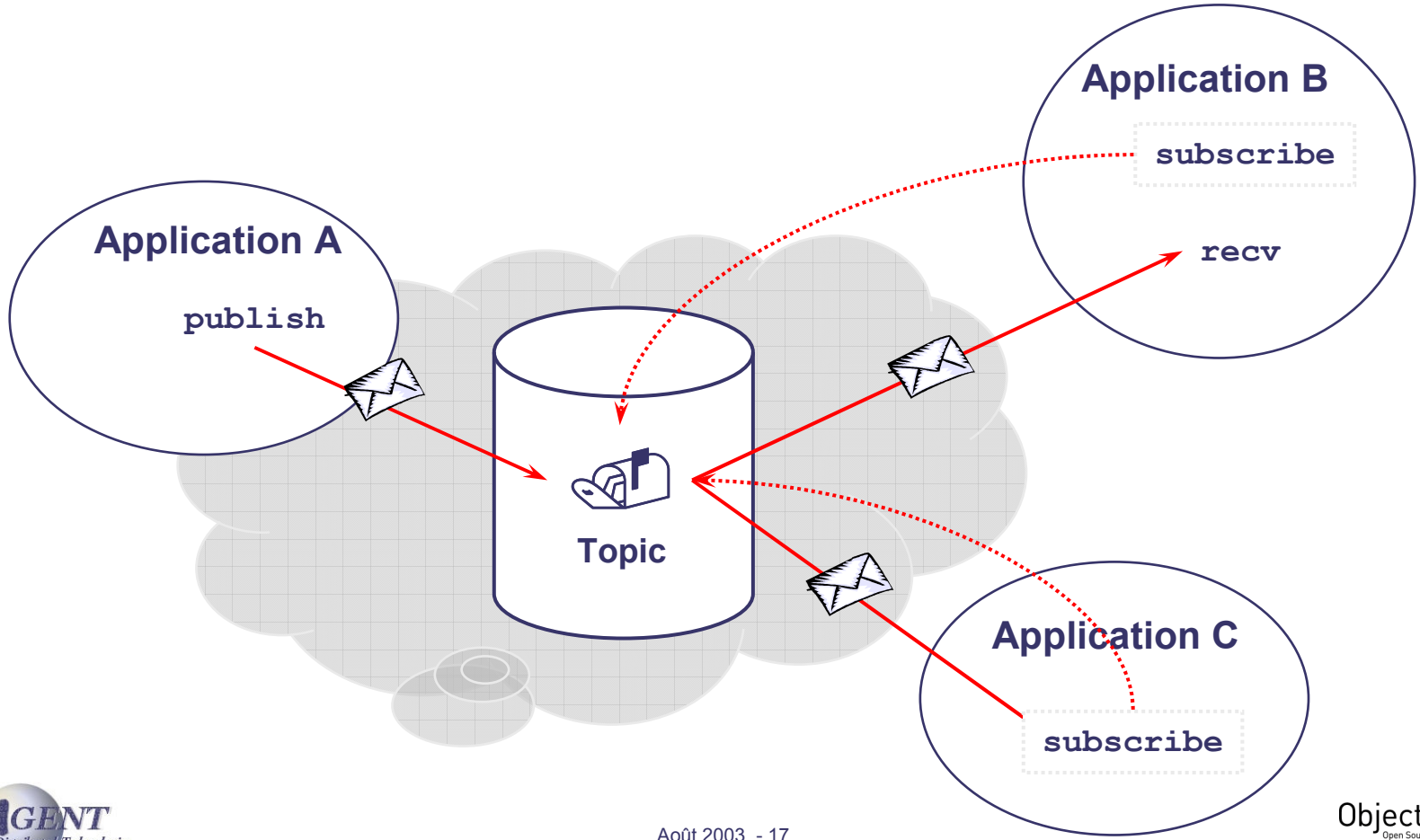


Modèle « Point-to-Point »

- 1 ! Destinataire
- Indépendance de l'émetteur et du destinataire
  - Anonymat → Evolution
  - Indépendance temporelle
- Acquittement du traitement par le destinataire

Modèle « Publish/Subscribe »

- Un message émis vers un *Topic* donné est délivré à l'ensemble des applications ayant souscrit à ce topic.



Modèle « Publish/Subscribe »

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

Consommation des messages

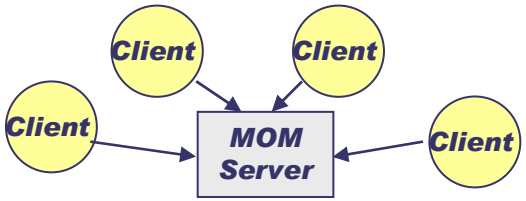
- **« *Pull* » – réception explicite**
  - Les clients viennent prendre périodiquement leurs messages sur le serveur.

- **« *Push* » – délivrance implicite**
  - Une méthode prédéfinie (réaction) est attachée à chaque type de message (événement)
  - la réception d'un événement entraîne l'exécution de la réaction associée.

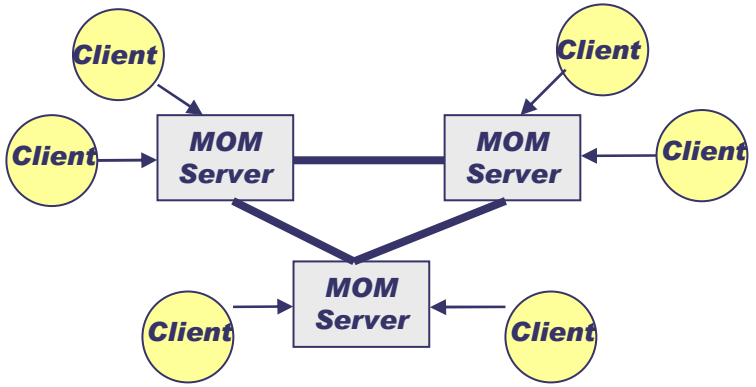
  ➔ Evénement / Réaction

Architecture

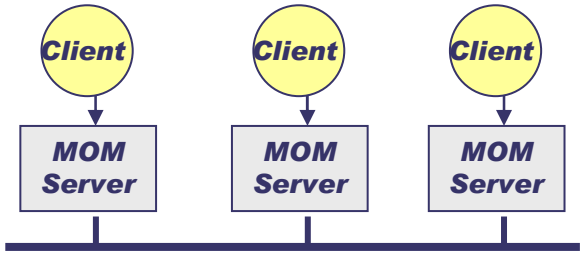
- Centralisée : *Hub and Spoke*



- Distribuée : *Snowflake*



- Distribuée : *Bus*



Qualité de service

- **Fiabilité**
  - Persistance des messages + Garantie de délivrance.
- **Scalabilité**
  - Nombre de sites, Taille des messages
  - Wide Area Networks (hétérogénéïté, etc.)
- **Transaction**
- **Sécurité**
- **Répartition de charge**
- **Ordonnancement**

Fonctions additionnelles

- **Routage des messages par le contenu**

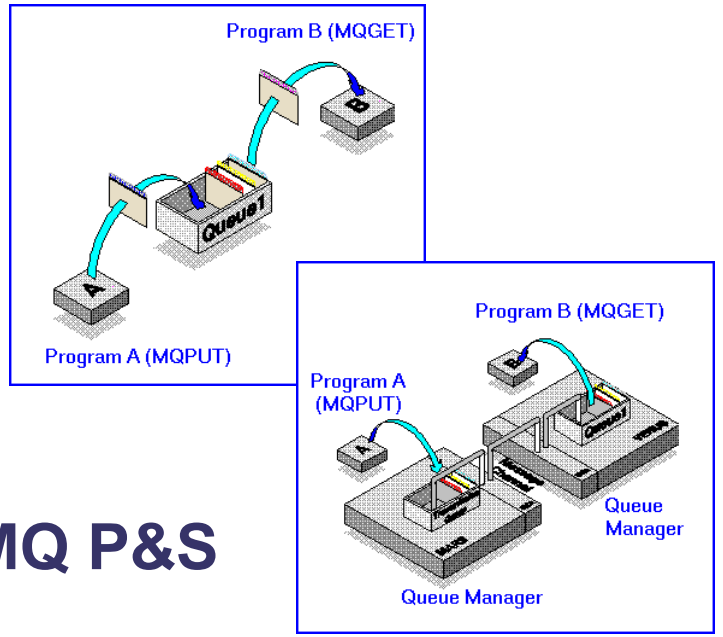
- **Filtrage des messages**
  - émetteur, type de message, priorité
  - Attributs, contenu

Exemples

- **Message Queuing → PTP**
  - MQ/S
- **Bus logiciels → P/S**
  - Koala, SoftBench, etc.
- **Unification PTP, P/S … Broker de messages**
  - JMS → PTP & P/S
    - *Sonic, Sun™ ONE, Joram, etc.*

IBM MQ-Series (WebSphere MQ)

- **Interfaces de programmation**
  - Message Queue Interface (MQI)
  - JMS1.1
- **Architecture « centralisée » C/S**
  - Interconnexion de Queue-Manager
- **Publish/Subscribe → Broker → MQ P&S**
- **MQ Everyplace → Mobile**
- **35 plateformes, nbx languages (C, C++, Cobol, etc) et protocoles**
- **Modules Workflow, EDI, EAI, etc.**



Microsoft MSMQ

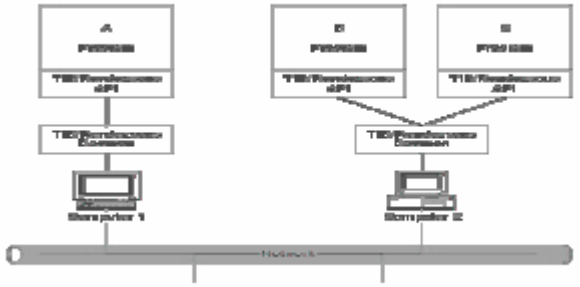
- **Environnement de prog. Microsoft**
- **« Intégré » à l'OS.**
- **Propriétés**
  - Garantie de délivrance, « one-to-many » (P/S) → 3.0 (XP)
  - Routage « cost-based »
  - Transactions,
  - Sécurité, priorités
- **Interopérabilité → MQSeries**

BEA MessageQ

- **Architecture : bus de messages**
  - MQ groups + routage
  - MQ Server
    - *Persistance des messages, fiabilité et routage*
  - MQ client
- **Propriétés**
  - Garantie de délivrance, P/S
  - « Self-describing Message », Sélection de messages, CorrelationId
  - Portabilité, Interopérabilité

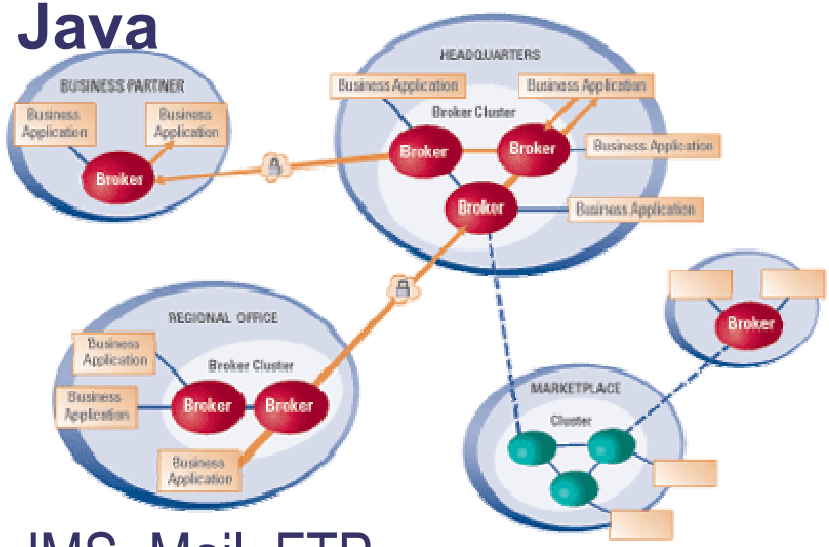
Tibco RendezVous

- **Implémentation JMS1.1**
- **Architecture : bus de messages**
  - Un démon (RVD) par site
  - Protocole fiable de diffusion sur UDP
    - *chaque démon filtre les messages qu'il doit retransmettre à ces clients*
  - WAN: routeurs "intelligents" de messages
- **routage des messages en fonction de leurs sujets**
- **Interopérabilité : WebLogic, WebSphere**



Progress SonicMQ

- **Implémentation JMS1.1, 100% Java**
- **Architecture distribuée C/S**



- **SonicMQ Bridge**
  - IBM MQSeries, Tibco RendezVous, JMS, Mail, FTP
- **SonicMQ Clients**
  - Accès à l'ensemble des fonctions JMS depuis C, C++, etc.
  - Windows, Solaris.

FioranoMQ

- Implémentation JMS1.1
- XML « content based routing »
- Scalabilité, sécurité (SSL)
- Interopérabilité :
  - IBM MQSeries, MSMQ.
  - COM (ActiveX), C, C++.

Softwired iBus

- **Implémentation JMS, 100% Java**
- **iBus//MessageServer**
  - Architecture centralisée « hub & Spoke »
- **iBus//MessageBus**
  - Architecture bus de messages
    - *Protocole IP-Multicast (intranet)*
    - *Passerelles TCP (internet)*
- **Transactions XA, contrôle d'accès, sécurité (SSL), persistance**

Sun™ ONE Message Queue

- **Implémentation JMS1.1, 100% Java**
- **Architecture C/S**
  - Distribuée → Enterprise Edition
- **Sécurité, Scalabilité**
- **Interopérabilité, portabilité**
  - SOAP, C (3.5β)
  - Windows, Linux, Solaris
  - TCP, HTTP, SSL
- **Integration Server → EAI, B2B**

Interopérabilité

- Pas de standardisation entre les MOM
  - **Spécification BMQ : Business Messaging Quality**
  - **Une autorité MOMA (Message Oriented Middleware Association)**
- CORBA 3.0
  - **introduction de la notion de messages asynchrones**
- J2EE
  - **API JMS**

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.*

JMS - Java Message Service

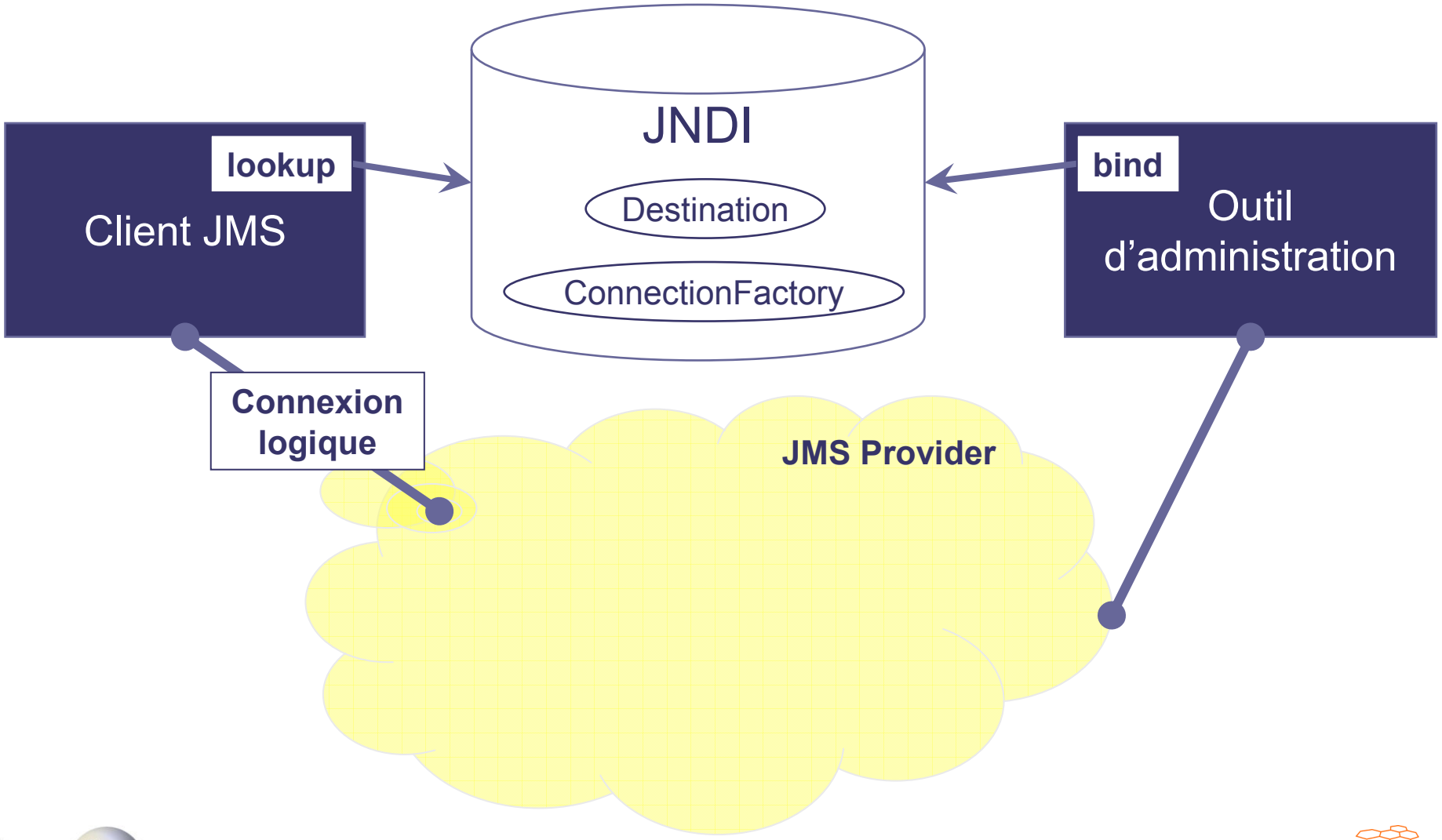
- **La spécification JMS est limitée :**
  - déploiement, administration (JSR 77), etc.
  - « *hierarchical topics* », « *Dead Message Queues* », etc.
- **Les applications utilisant l'API JMS sont (presque) indépendante du MOM utilisé (portabilité)**
  - … mais actuellement l'interopérabilité entre deux MOM's nécessite une passerelle.

Application JMS

- « JMS Provider »
- Clients JMS
- Objets administrés
  - ConnectionFactory, Destination.
- Messages

- Clients « natifs »

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 session (transaction)

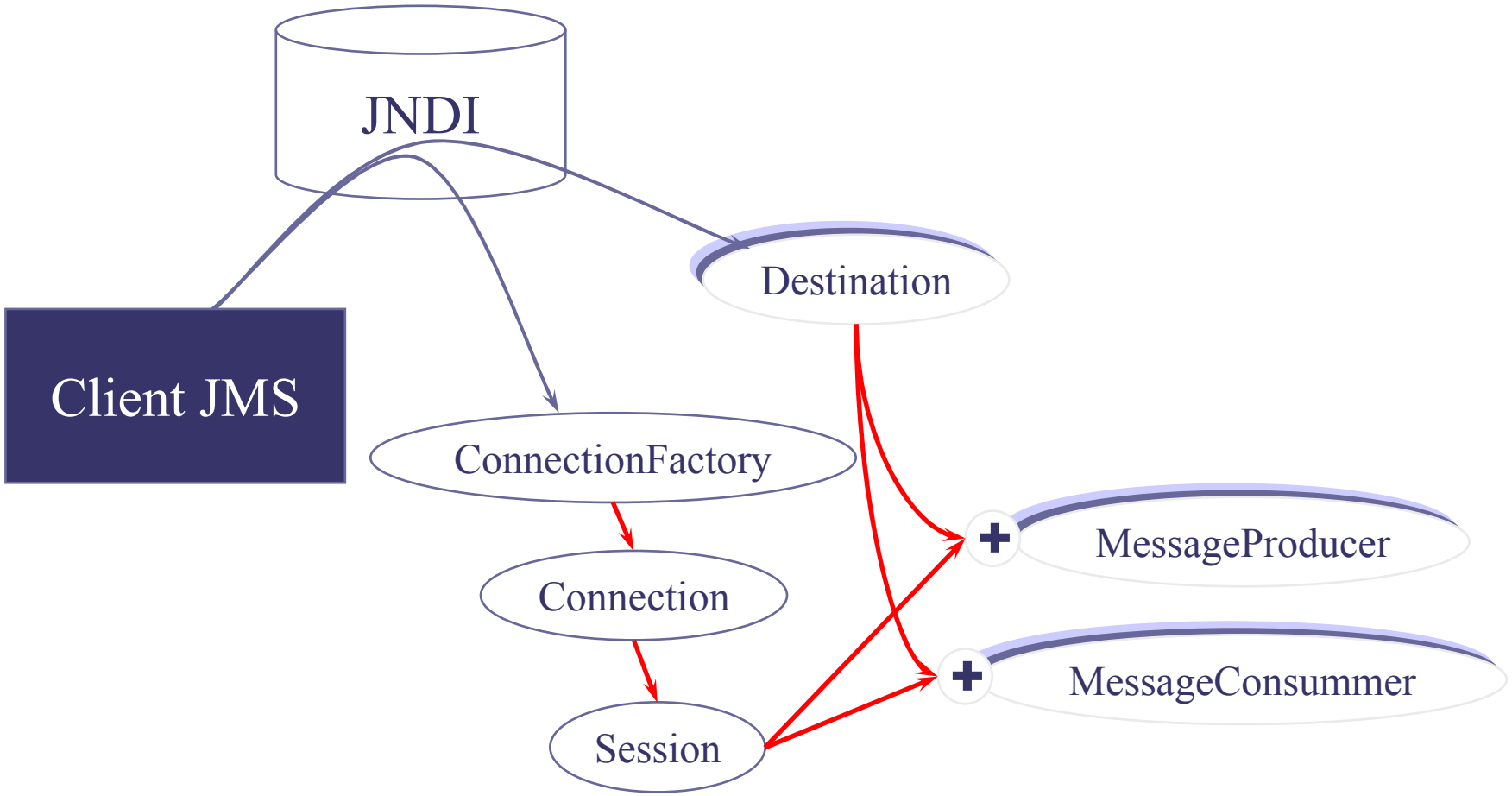
Les objets JMS

- **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

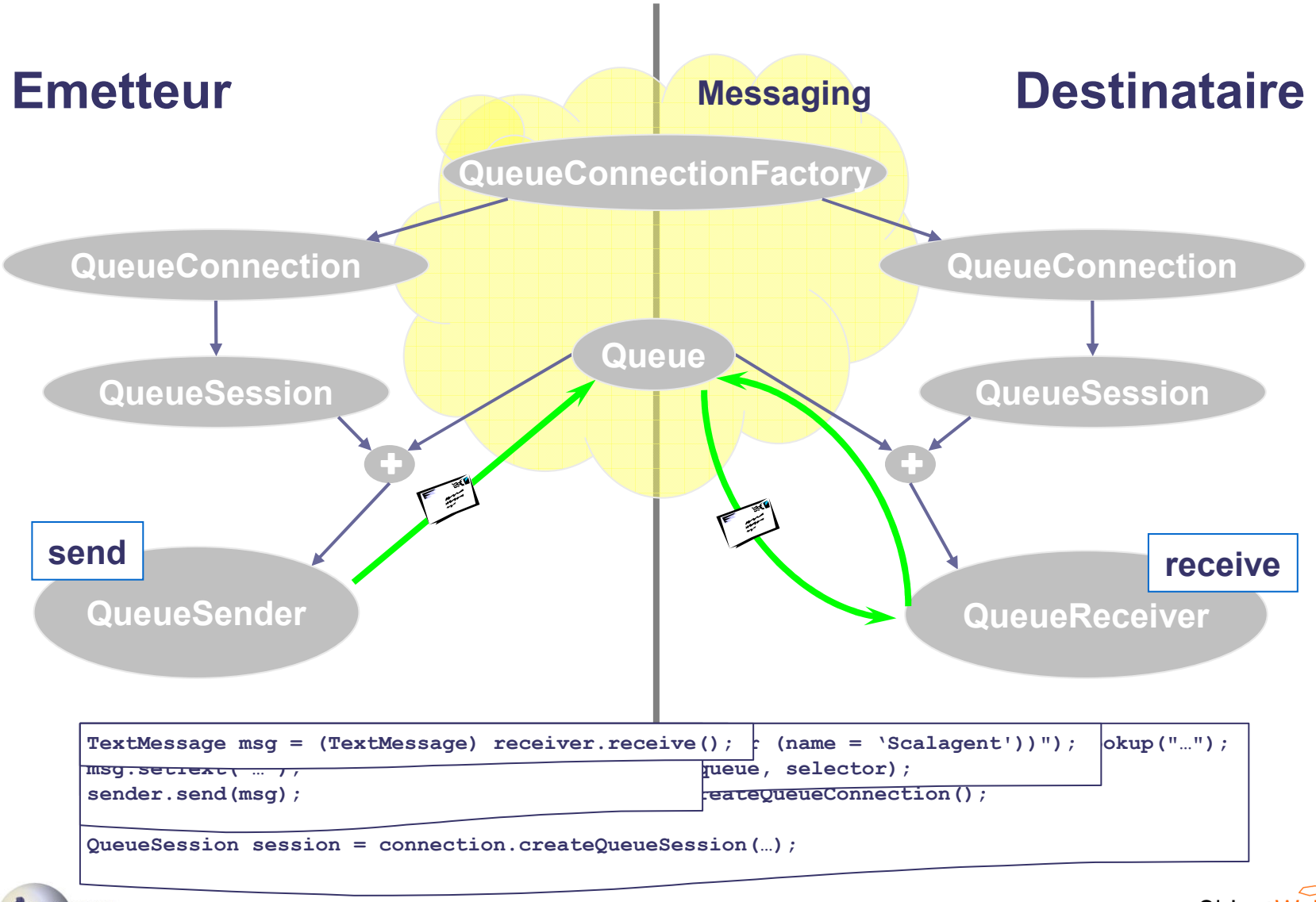
Architecture



Le message JMS

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

JMS - "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(…);
```

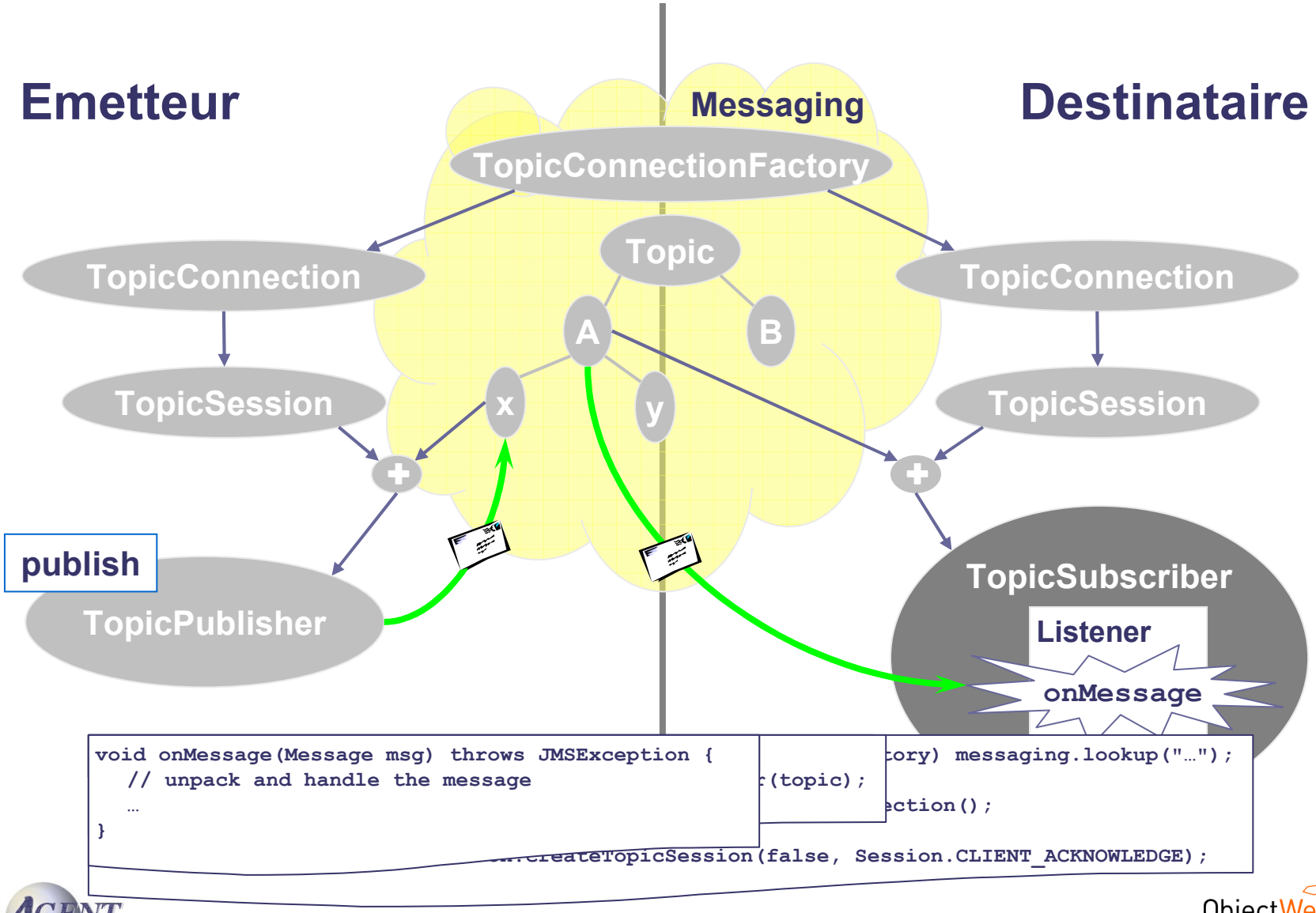
```
QueueSender sender = session.createSender(queue);
```

```
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 - "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
   …
}
```

JORAM

- **Un MOM Open-Source et 100% Java**
  - Modèle de communication : message queuing et publish/subsribe
  - Fournit l'interface JMS
  - Basée sur la plateforme ScalAgent

- **Le MOM ScalAgent s'appuie sur une technologie à base d'agents**
  - Comportement Transactionnel
  - Architecture distribuée

JORAM

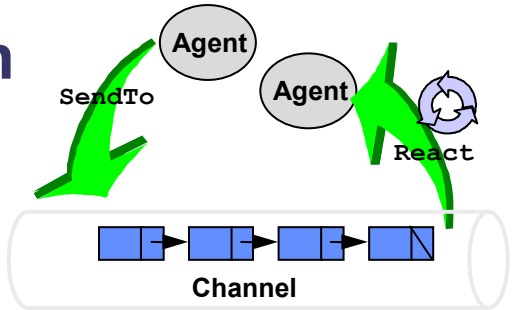
- **Joram implémente la dernière spécification JMS 1.1**
  - Topics hiérarchique, topics clusterisés
  - DeadMessageQueue
  - Support de SOAP / XML
  - kJoram: un client léger (J2ME) pour périphérique portable
- **Extensions en cours**
  - Gain de Performances, persistance BD
  - Outils d'administration et support JMX
- **Joram est la solution JMS intégrée dans JOnAS (MDB)**

Le MOM ScalAgent

- **Bus logiciel à base d'agents communicants**

- **Agents = objets réactifs**
  - Persistants
  - Légers: infrastructure d'exécution partagée au sein d'un serveur d'agents

- **Modèle asynchrone événement / réaction**
  - Événement = message / notification
  - Réaction = fonction de la classe Agent



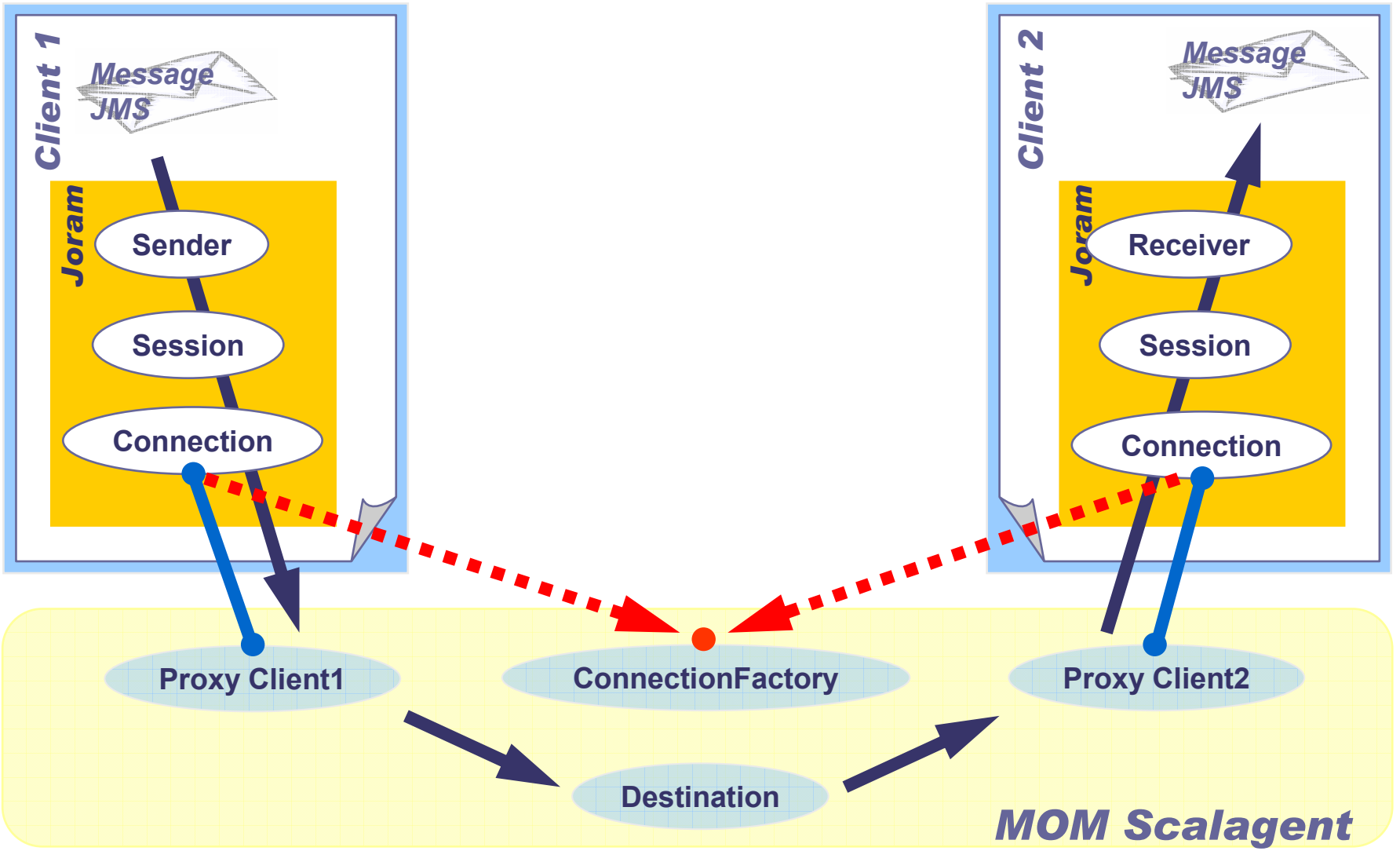
Le MOM ScalAgent

- **Persistance des agents et des messages**
- **Atomicité des réactions**
  - Cohérence garantie par un moniteur transactionnel
- **Persistance + Atomicité = Fiabilité**
  - Chaque notification est délivrée une et une seule fois
- **Architecture distribuée hybride**
  - Configuration de domaines de communication (bus)
  - Routage entre les domaines

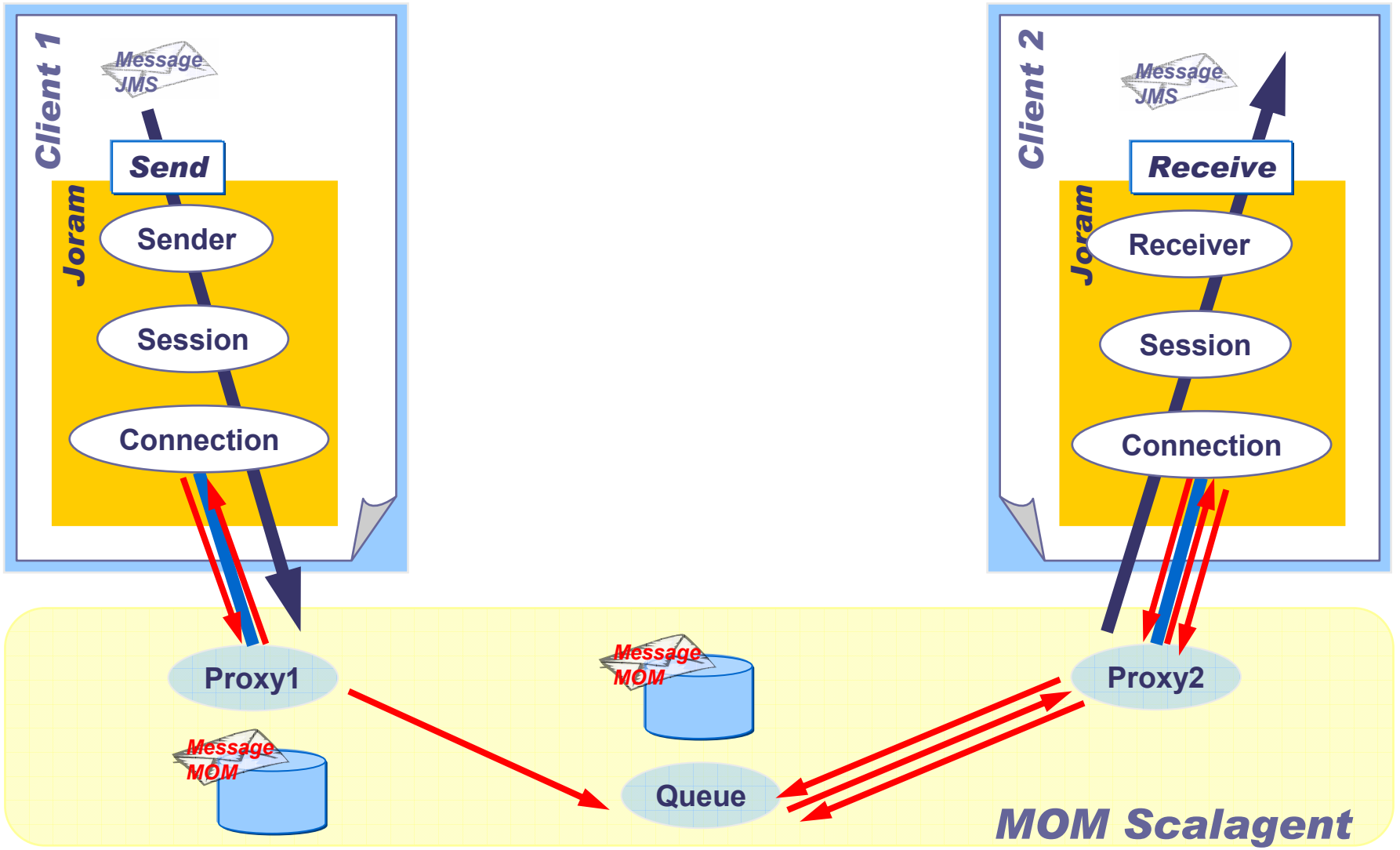
JORAM – Interface JMS du MOM ScalAgent

- **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 « ConnectionFactory » sur chaque nœud**
  - Mise en place des connections
- **Chaque « client JMS » est représenté par un agent « proxy »**
  - Gestion de la connection, dialogue avec les destinations
- **L'architecture est naturellement distribuée**

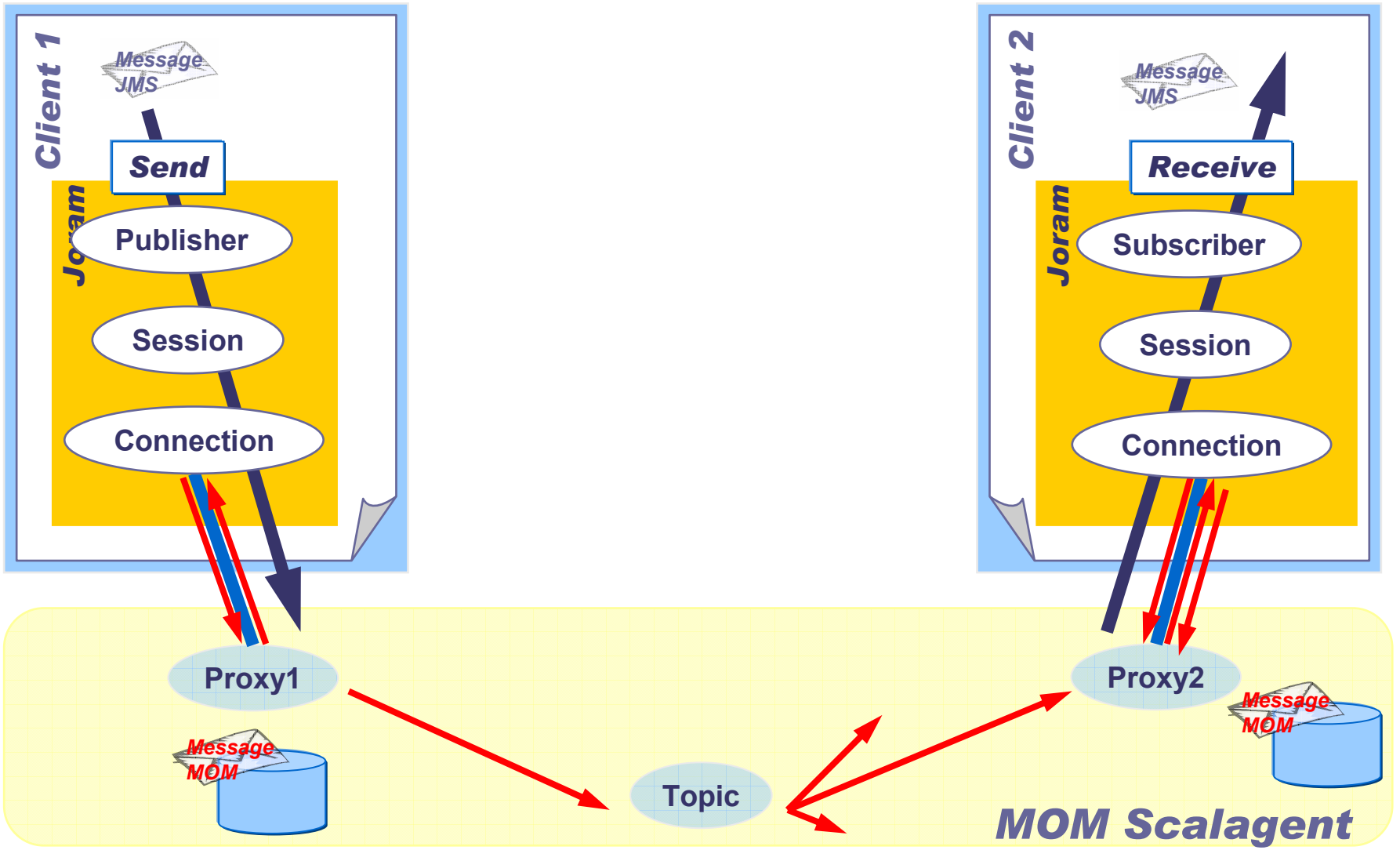
Joram – Architecture logique



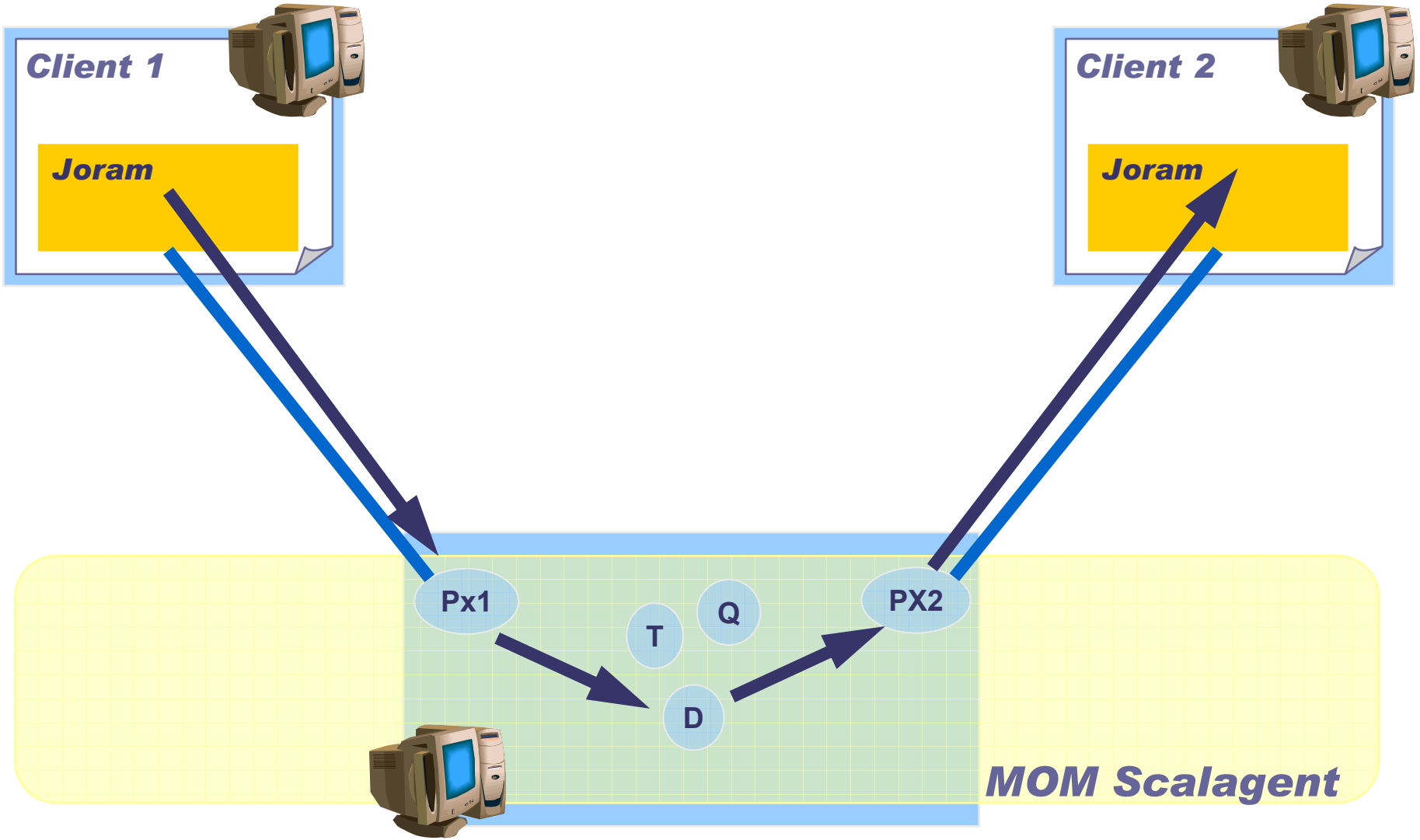
Joram – Point To Point



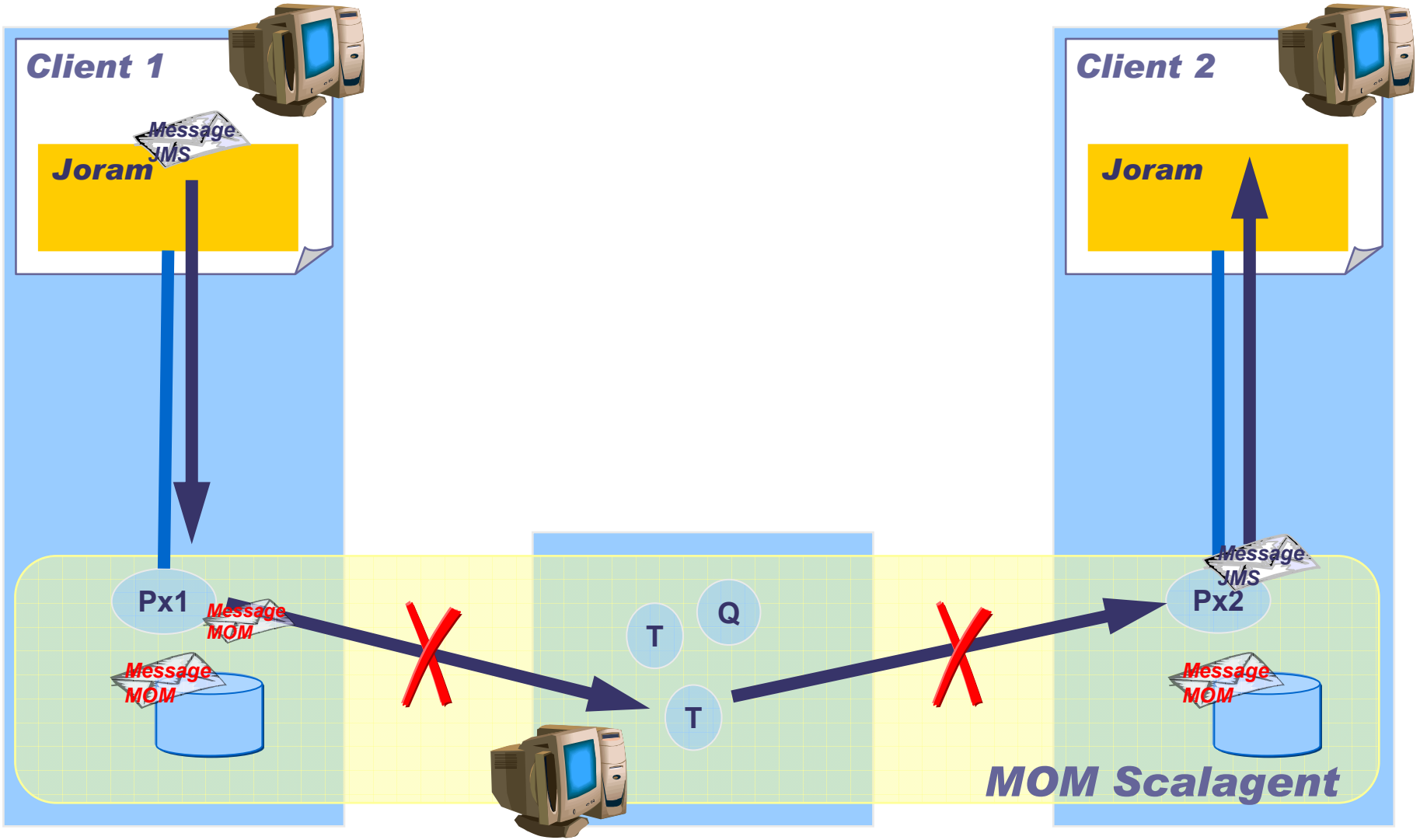
Joram – Publish/Subscribe



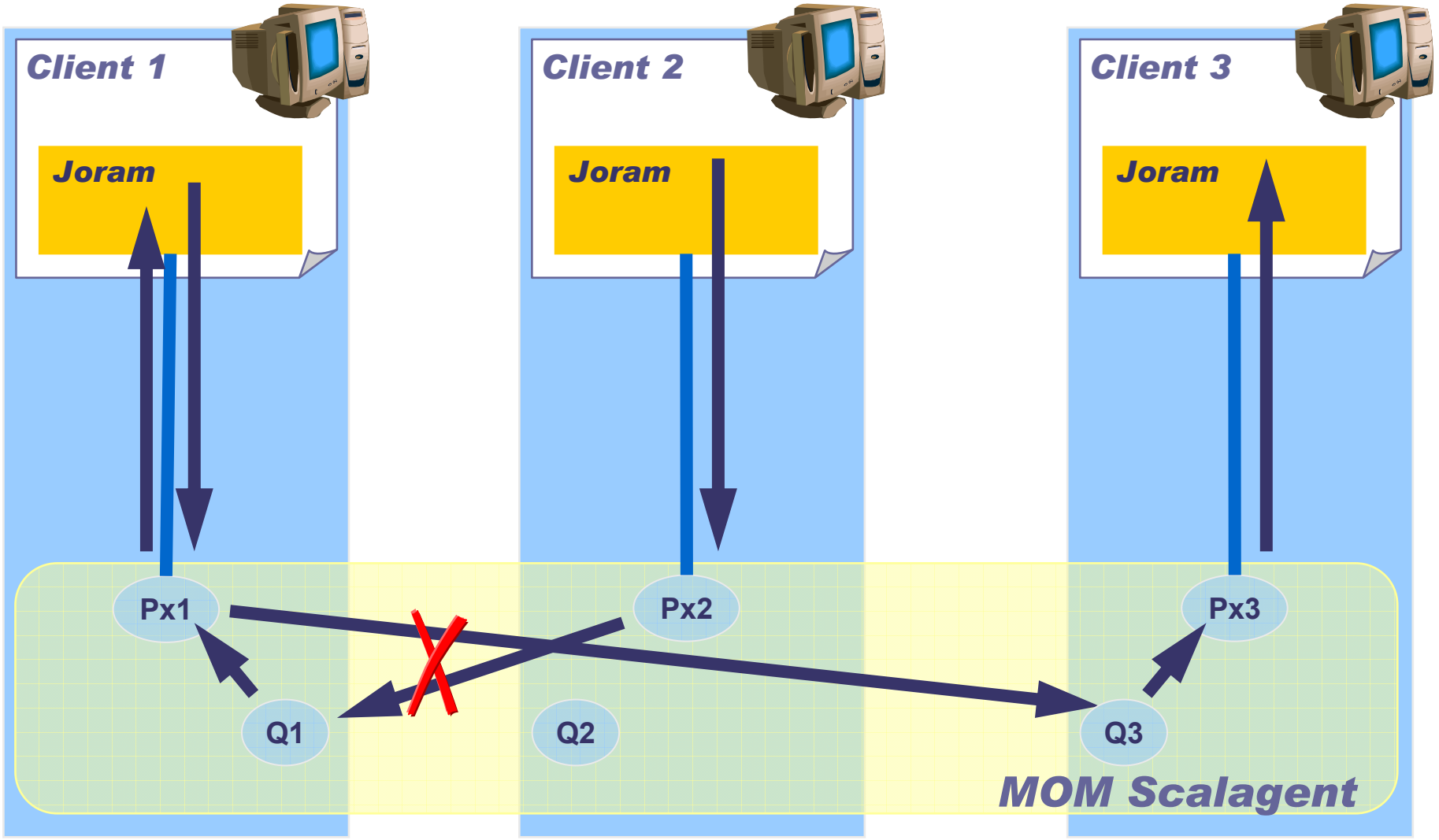
Joram – Architecture centralisée



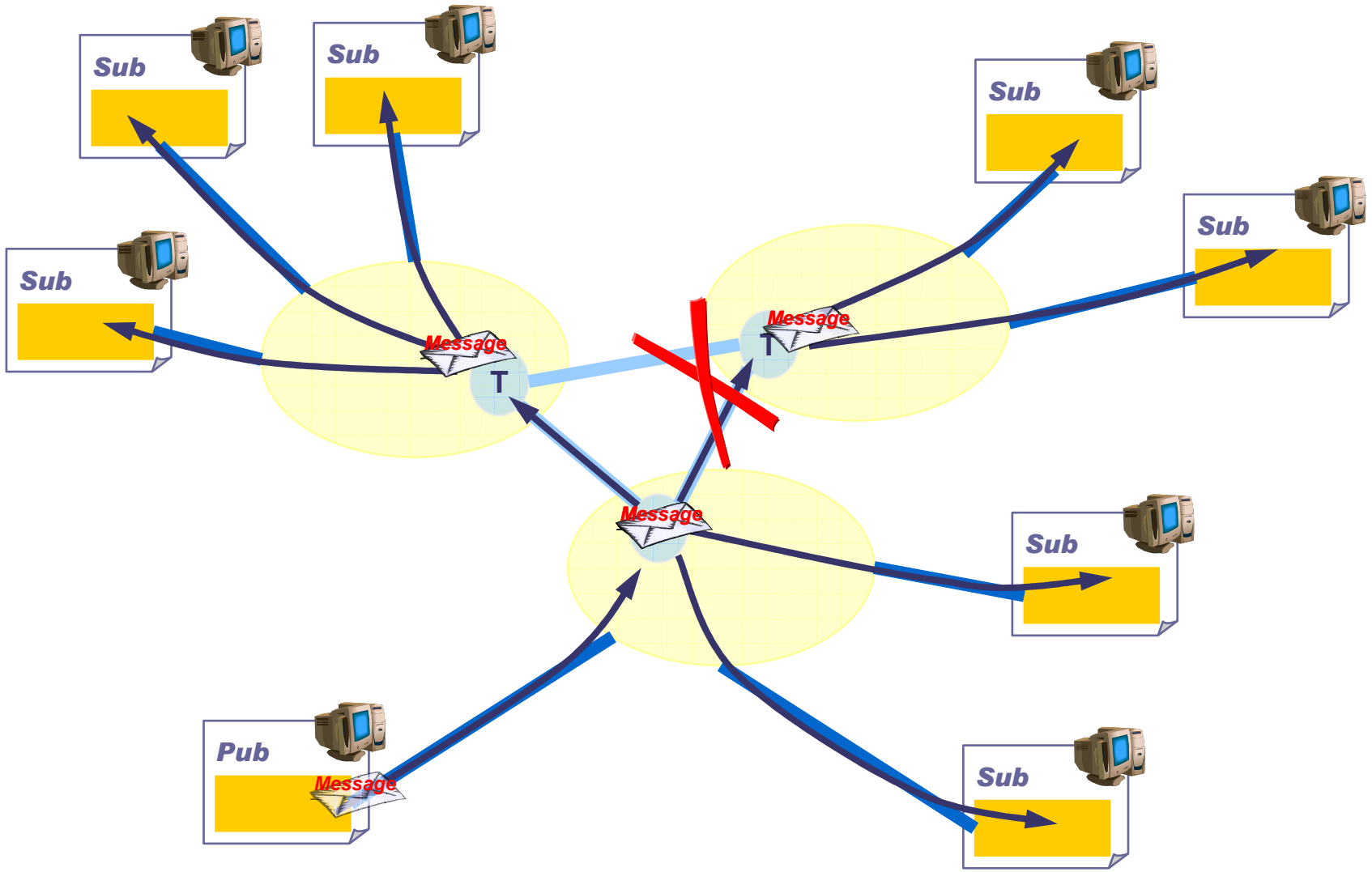
Joram – Architecture distribuée



Joram – Architecture distribuée



Joram – Topic « clusterisé »



Joram - Administration

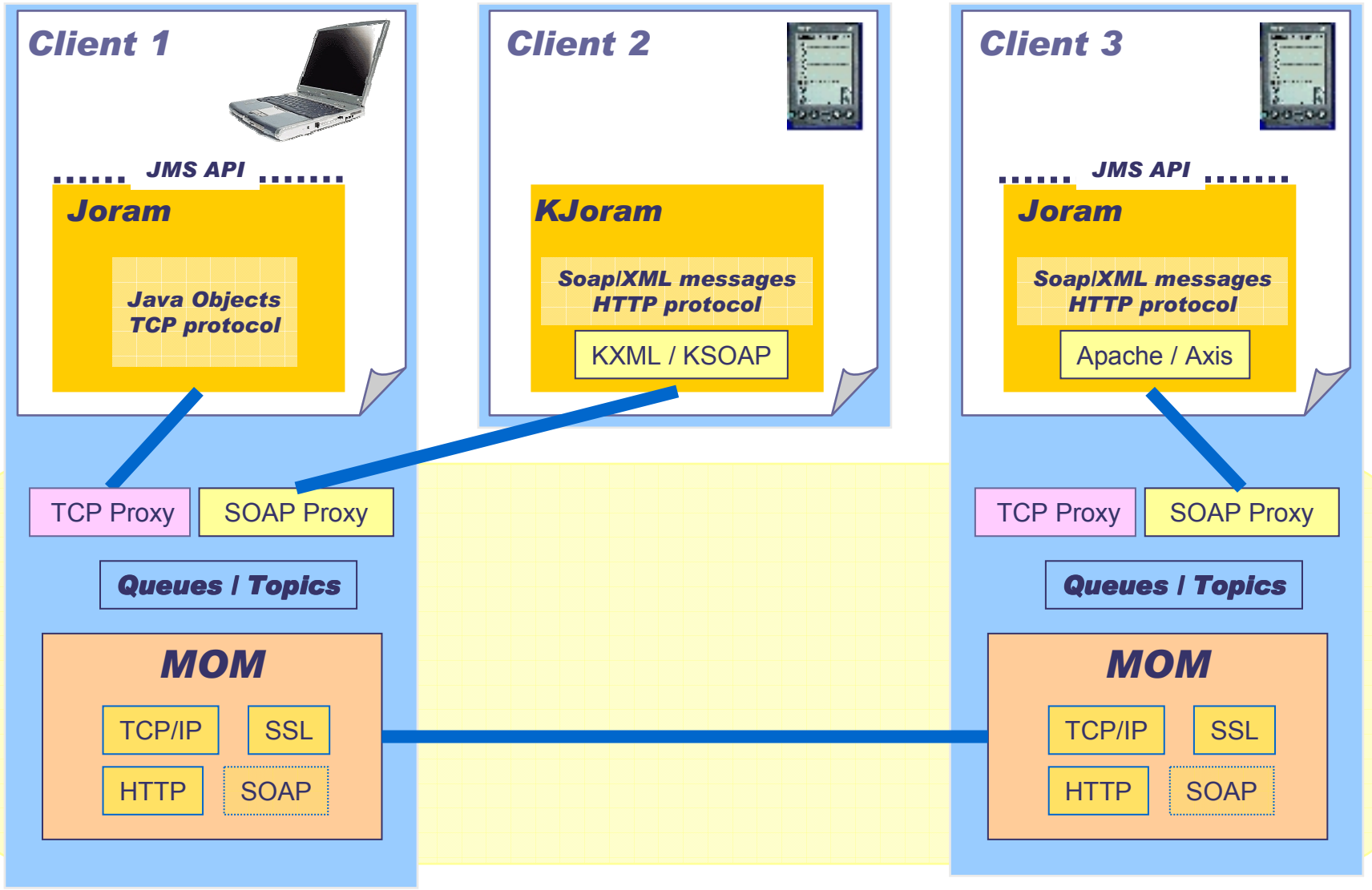
Au travers de JMS

- 1 Topic d'administration sur chaque serveur → Cluster
- API client d'administration
  - *Dialogue au travers de Message JMS*

Au travers de JMX

- Topic d'administration
- Proxy, Queue et Topic

Joram - Interconnexion



Qualité de service

- **Actuellement deux modes :**
  - Persistant / Transactionnel
  - « Transient »

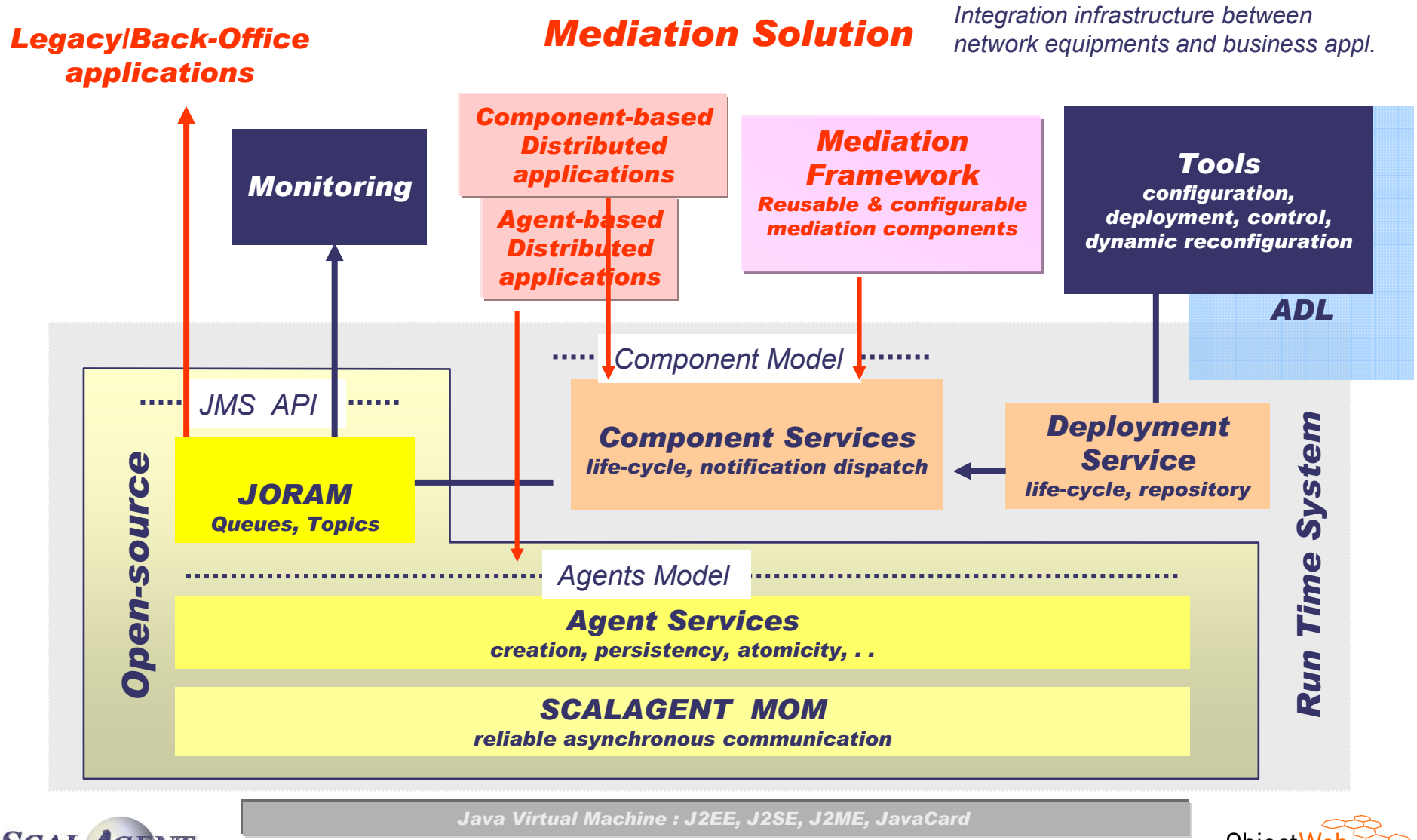
  ➔ 2 Mondes ☹

- **Futur**
  - Destination : Persistent / Transient
  - Messages :
    - *Priorité*
    - *Persitent / Transient*

Scalagent Distributed Technologies

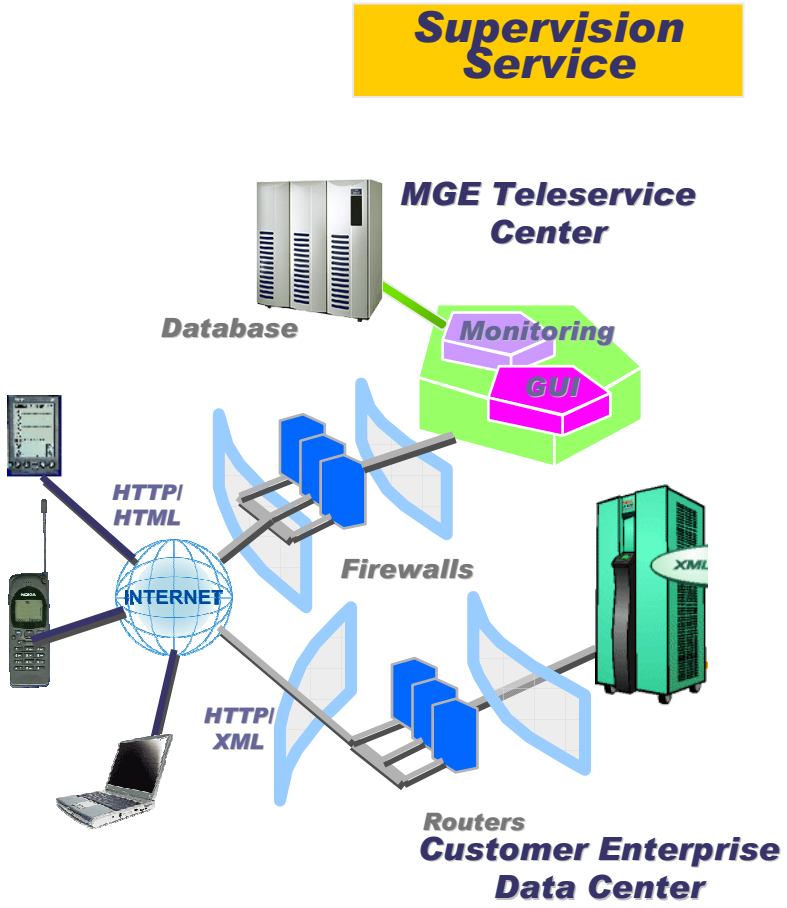
- **Joram est un élément d'une technologie plus vaste**
  - Mise en évidence des propriétés de la plateforme
    - *Distribution, fiabilité, etc.*
- **Le package Open-Source offre des fonctions additionnelles**
  - Modèle de programmation Agent : Evénement / Réaction
    - *Workflow, EAI, ETL, etc.*
- **Ensemble de briques à valeur ajoutée**
  - Modèle à composant et outils associés

ScalAgent Distributed Technologies



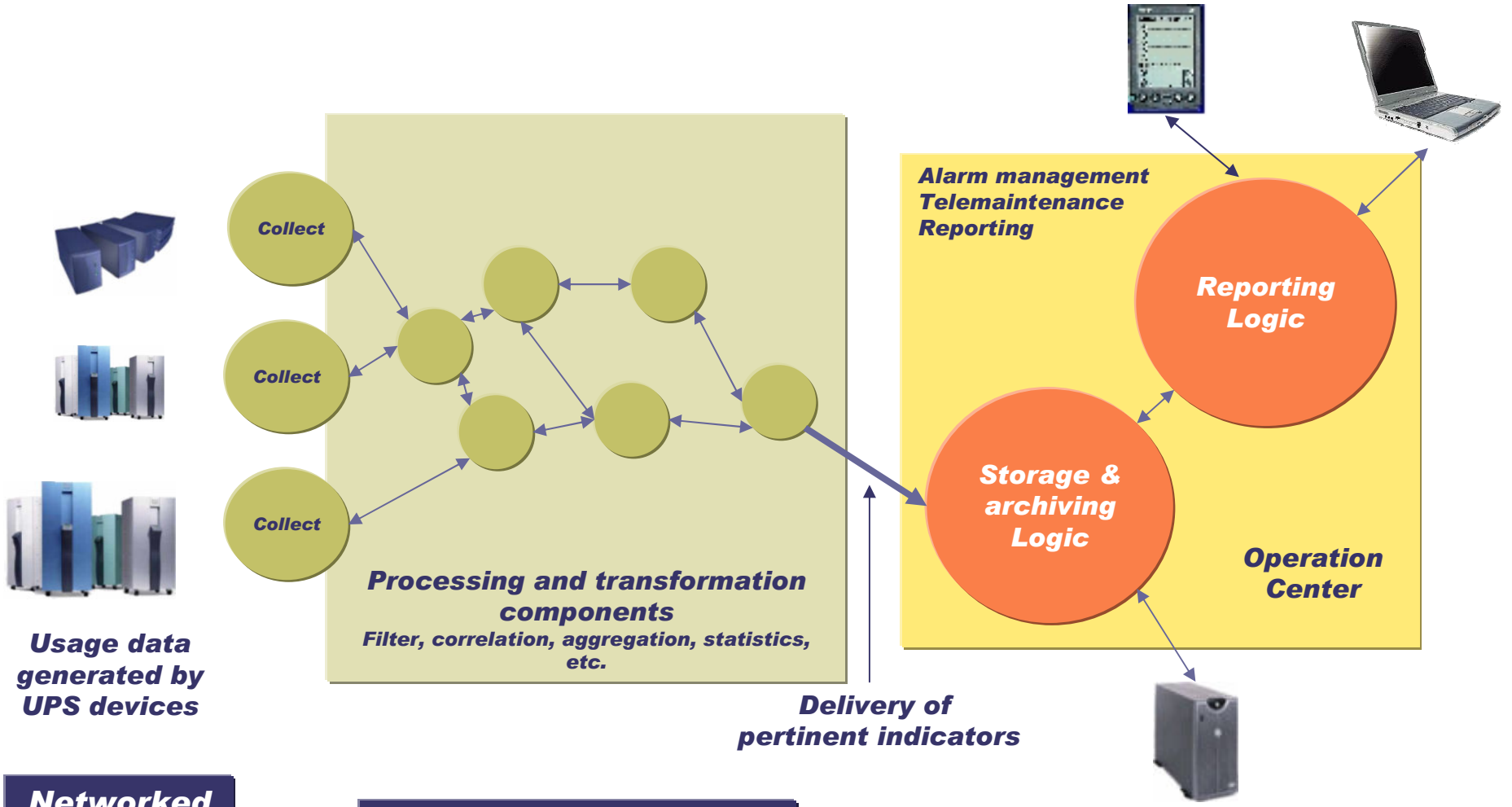
Example: Supervision of UPS Devices

- **Objectives**
  - Supervision of distributed Uninterruptable Power Supply Devices (UPS)
    - *Collecting usage data from UPSs in real-time*
    - *Computing indicators from a set of UPSs and reporting to a central control point*
- **System components**
  - JVM embedded in the UPS device communication board
  - ScalAgent mediation solution (from the UPS board to the management center)
  - Joram and kJoram as connectors to the business applications
  - JOnAS application server for supporting business applications (archiving and reporting)



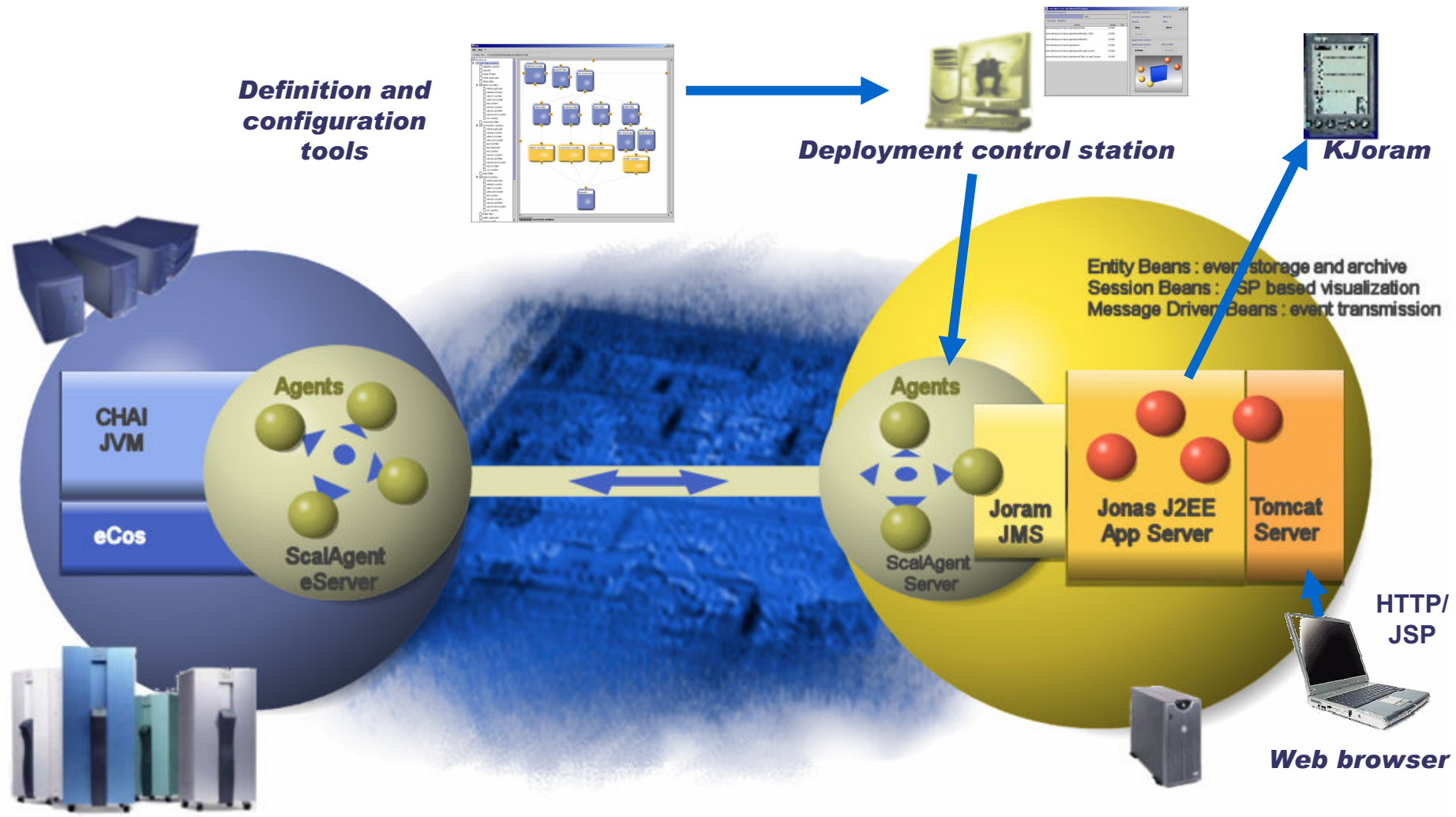
This application has been developed in the framework of the RNTL PARFUMS project (MGE-UPS, INRIA, ScalAgent, Silicomp)

Supervision of UPS: solution overview



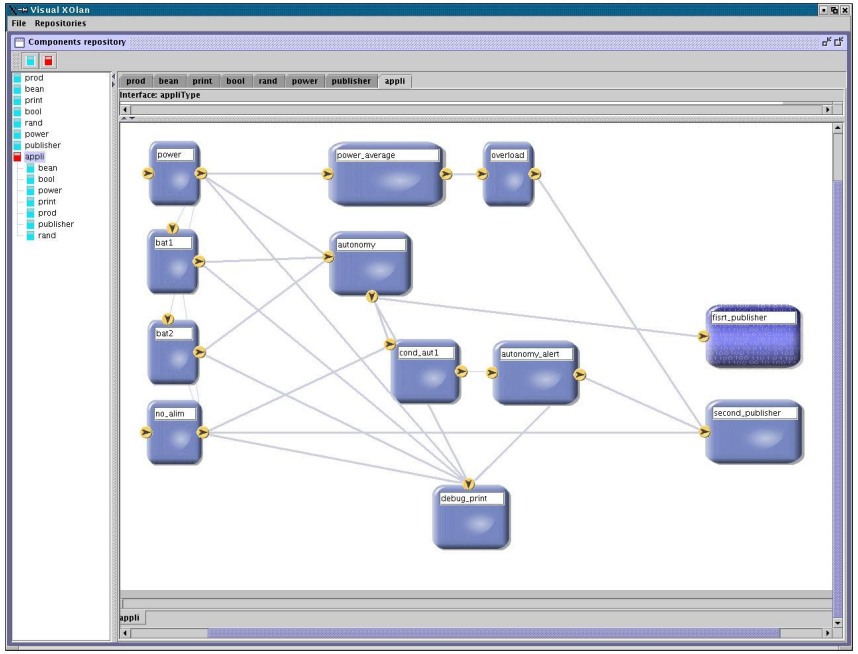
This application has been developed in the framework of the RNTL PARFUMS project (MGE-UPS, INRIA, ScalAgent, Silicomp)

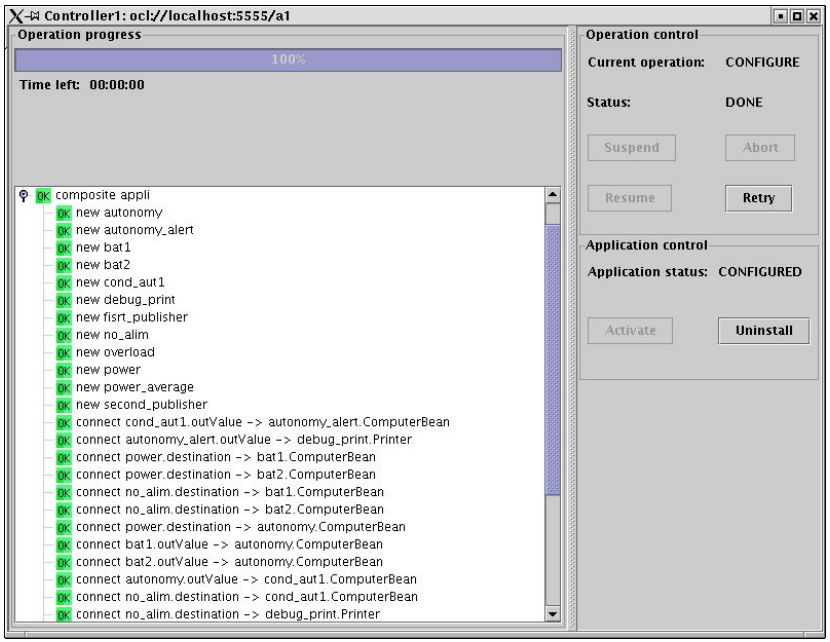
UPS supervision: technical components

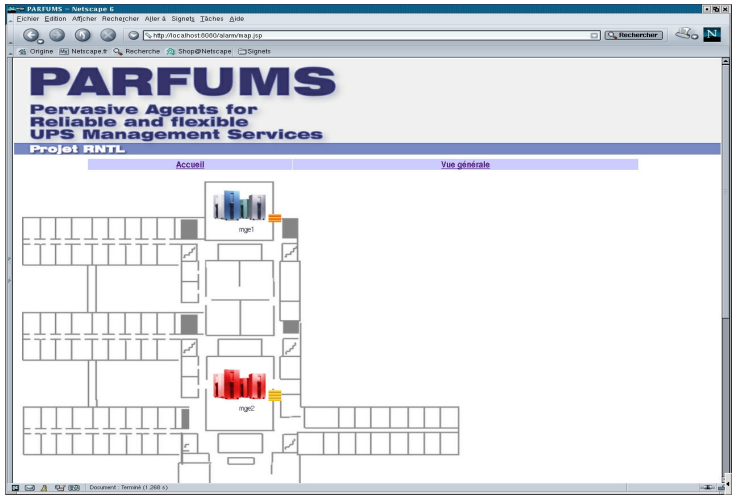


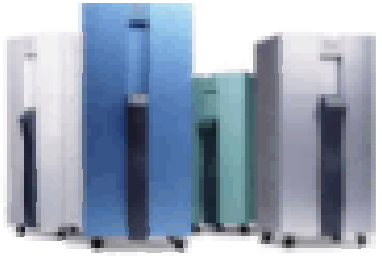
This application has been developed in the framework of the RNTL PARFUMS project (MGE-UPS, INRIA, ScalAgent, Silicomp)

UPS supervision in use









Bibiliographie

- BEA MessageQ
  - http://www.bea.com/content/products/more/messageq
- FioranoMQ 5
  - http://www.fiorano.com/products/fmq/overview.htm
- IBM WebSphere MQ
  - http://www.ibm.com/software/integration/mqfamily
- Microsoft Message Queue Server (MSMQ)
  - http://www.microsoft.com/windows2000/technologies/communications/msmq
- ObjectWeb JORAM
  - http://joram.objectweb.org
  - http://www.scalagent.com

Bibiliographie

- Progress Sonic MQ
  - http://www.sonicsoftware.com/products/enterprise_messaging
- Softwired iBus//MessageBus
  - http://www.softwired-inc.com/products/products.html
- Sun Java Message Service (JMS)
  - http://java.sun.com/products/jms
- TIBCO Rendezvous
  - http://www.tibco.com/solutions/products/active_enterprise/rv