

<http://www-adele.imag.fr/users/Didier.Donsez/cours>

Message Oriented Middleware (MOM) Java Message Service (JMS)

Didier DONSEZ

Université Joseph Fourier (Grenoble 1)

PolyTech'Grenoble - LIG/ADELE

Didier.Donsez@imag.fr

Didier.Donsez@ieee.org

<http://www-adele.imag.fr/users/Didier.Donsez/cours>

Message Oriented Middleware (MOM)

Didier DONSEZ

Université Joseph Fourier (Grenoble 1)

PolyTech'Grenoble LIG/ADELE

Didier.Donsez@imag.fr

Didier.Donsez@ieee.org

Motivations

- **Modèle Client-Serveur**
 - requêtage synchrone
 - RPC DCE et DCOM, CORBA, RMI
 - inconvénient : connexion permanente des 2 parties
 - Problème des pannes/connexions transitoires
- **Une alternative : la messagerie inter-application**
 - les messages (qui peuvent être des requêtes et leurs réponses) sont envoyés quand la connexion est ouverte.
 - Principe du Store-and-Forward

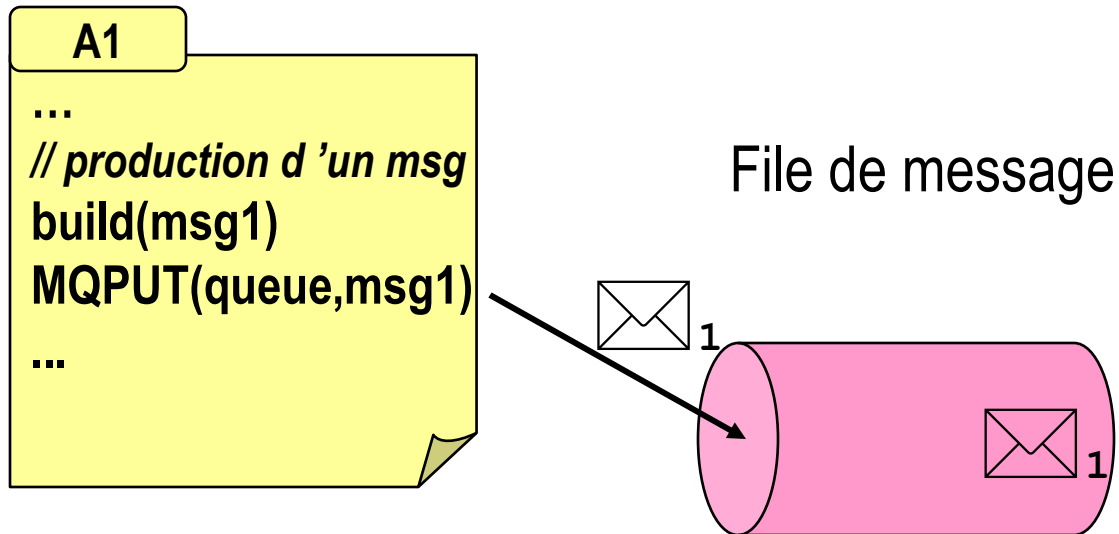
Motivations

- Applications (passage à très grande échelle)
 - Diffusion d 'information (push)
 - news, stock quote, ...
 - Messagerie inter-bancaire, workflow, ERP, ...
 - Synchronisation de BD nomades et réplicat asynchrone (hot standby)
 - EAI (Enterprise Application Integration), B2B
 - ESB (Enterprise Service Bus)
 - Data Warehouse (ETL : Extract Transform Load)
 - Collecte des données (journaux Firewall, mesures reseaux de capteurs, ...)
 - Déploiement grande échelle de logiciels (antivirus, ...)
 - ...

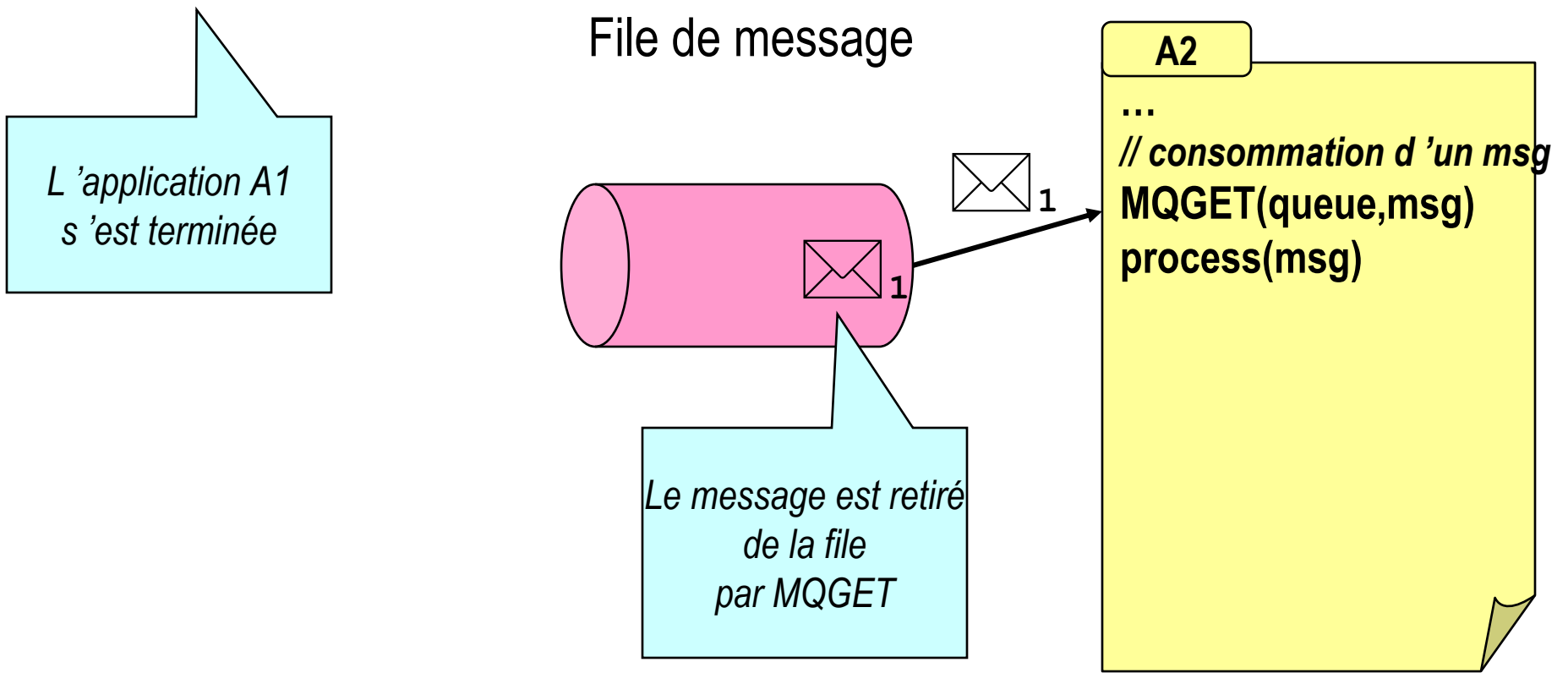
Principe

- Messagerie inter-application
 - Asynchrone
 - Non temps réel
 - s'oppose aux ORBs synchrones (Corba et COM+)
- Files de Messages (Message Queueing)
 - les messages sont mis dans une file d'attente persistante (i.e. sur disque)
avant d'être relayer vers l'application
 - Partage d'une file par plusieurs applications
 - Priorité des messages
 - Filtrage des messages à la réception
- Avantages
 - Insensible aux partitions de réseaux (sans fil, satellite, WLAN, ...)
 - Insensible aux applications non disponibles (temporairement)

Principe des Files de Messages (i)

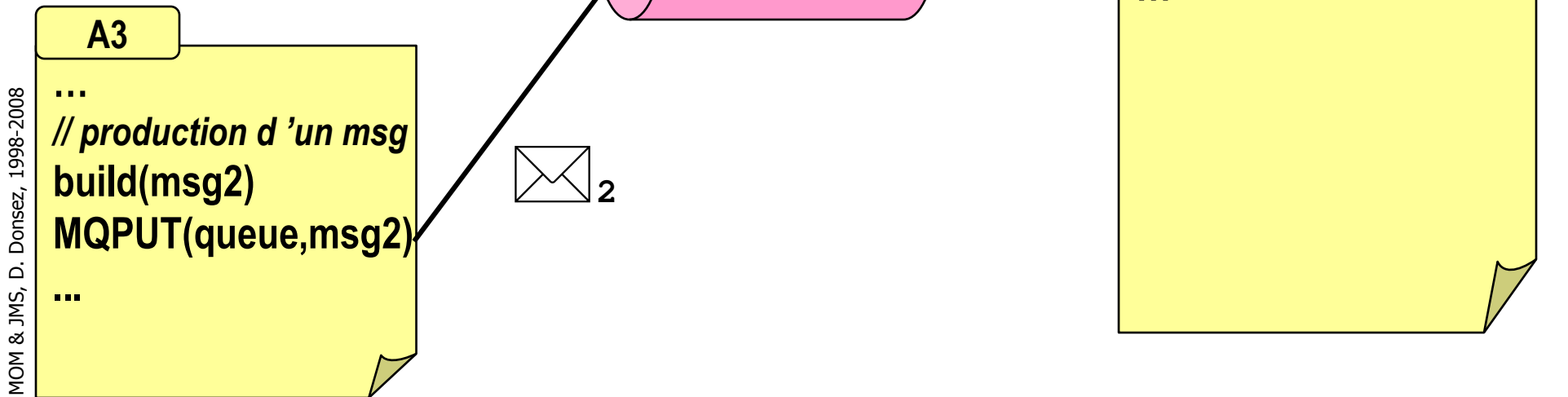


Principe des Files de Messages (ii)



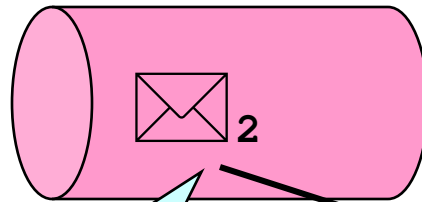
Principe des Files de Messages (iii)

File de message

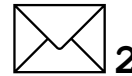


Principe des Files de Messages (iv)

File de message



Le message est retiré de la file par MQGET



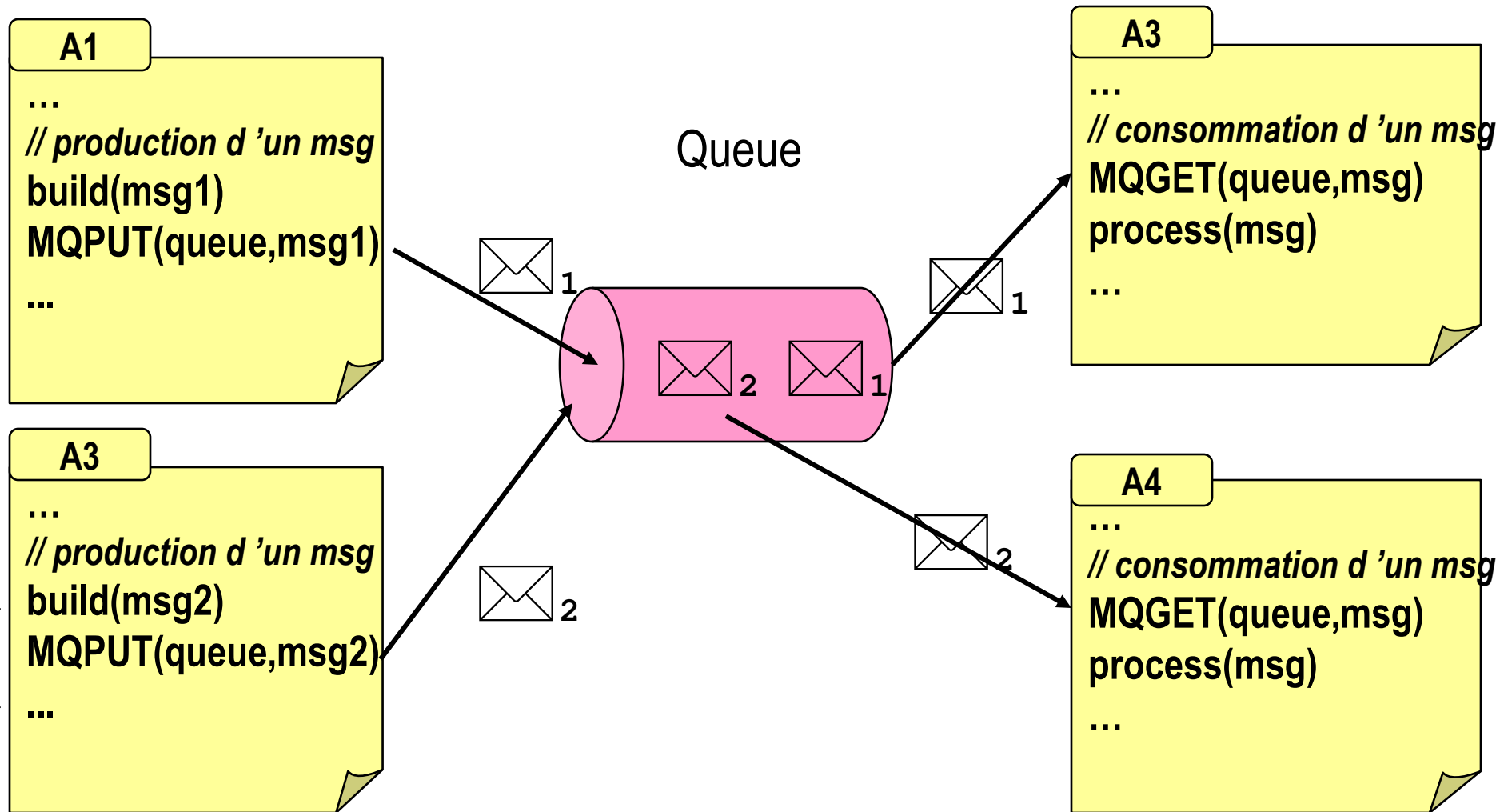
```
A2
...
// consommation d'un msg
MQGET(queue,msg)
process(msg)
...
// consommation d'un msg
MQGET(queue,msg)
process(msg)
...
```

L'application A3 s'est terminée

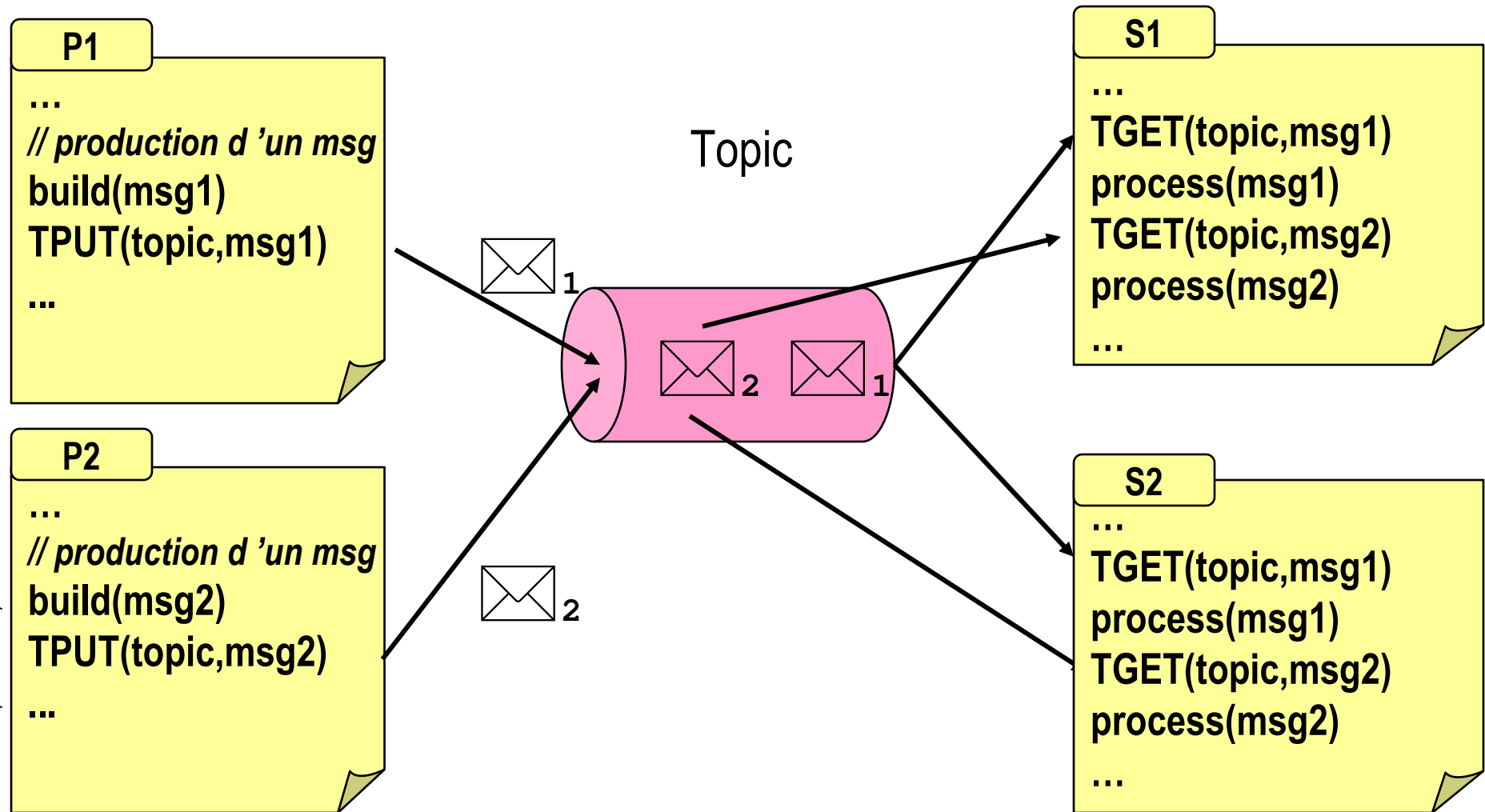
Modèles de messageries

- Routage de Message
 - par l'identité de l'application
 - par le contenu du message
 - chaque application consommateur définit un critère sur les messages à consommer
 - le critère peut être une expression booléenne sur les valeurs de champs du message
- Modèles
 - Message Queue
 - un message envoyé (produit) est consommé par un seul client
 - Publication-Souscription
 - un message publié est diffusé à tous les souscripteurs
 - Publication-Souscription par le contenu (content based publish-subscribe)
 - un message publié est diffusé à tous les souscripteurs par rapport au contenu du message (IBM' Gryphon, U. Colorado' Siena, ...)
 - Requête-Réponse
 - Client-Serveur asynchrone

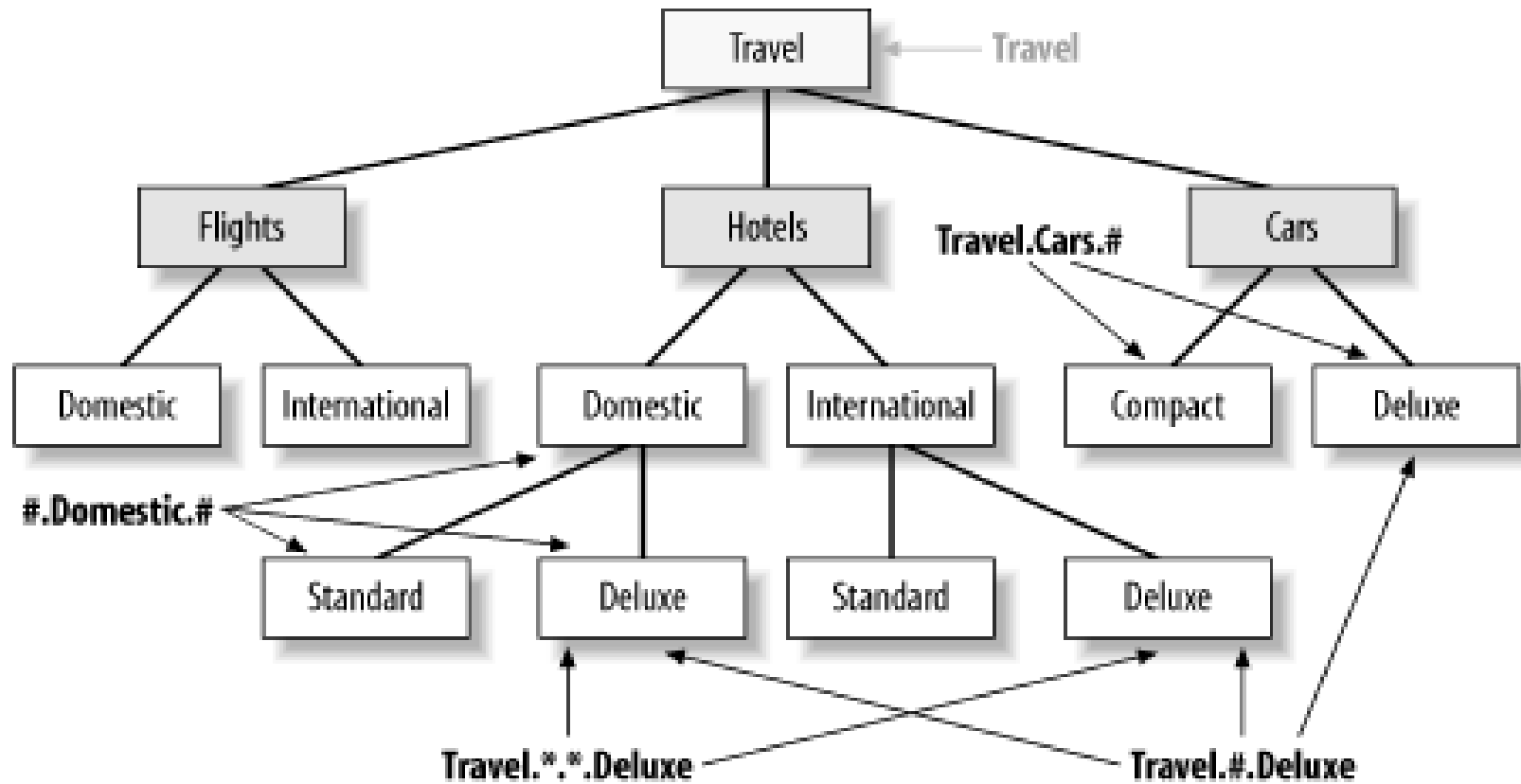
Modèle des Message Queues



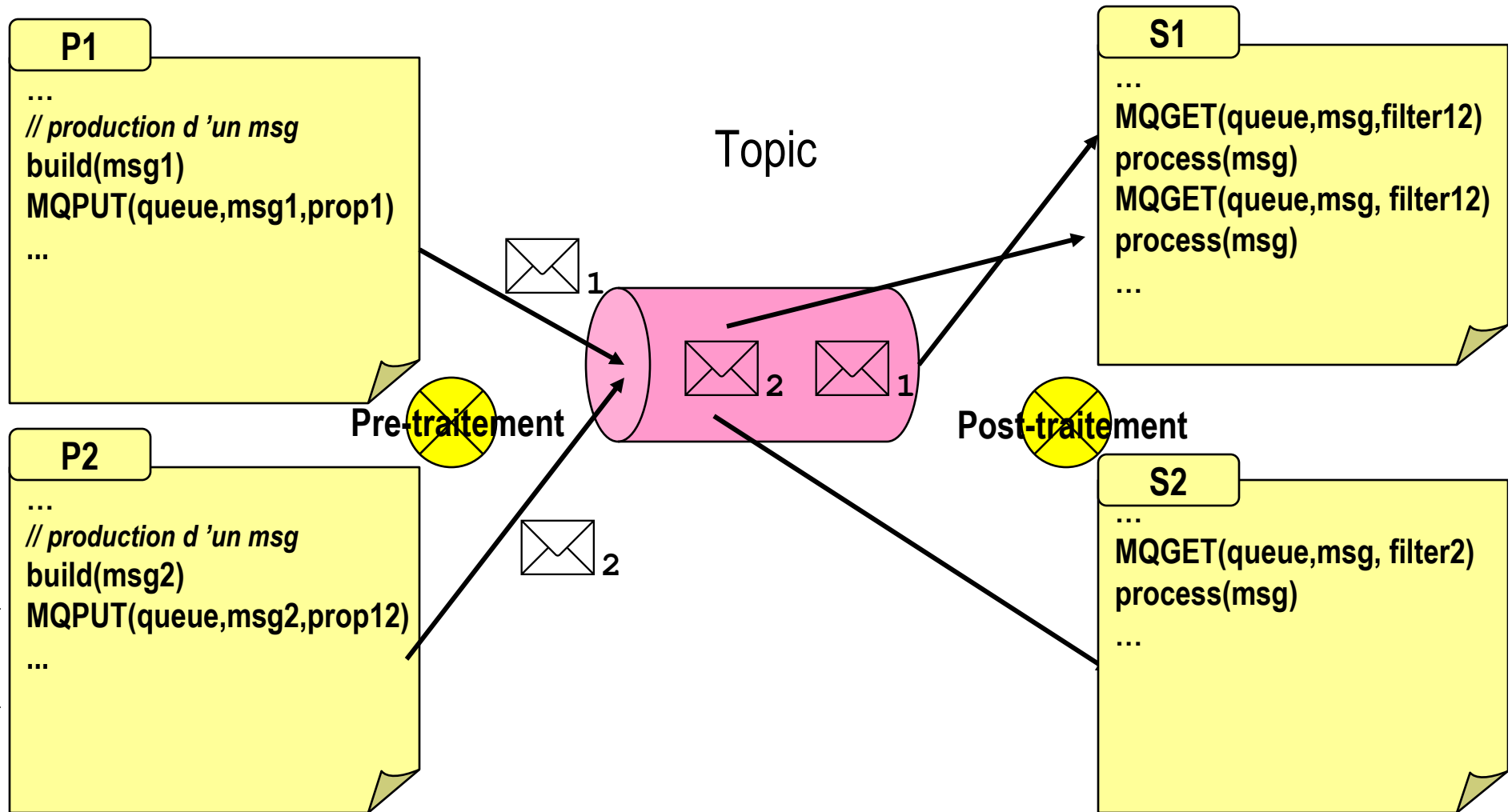
Modèle Publication-Souscription



Publication-Souscription sur des *topics* hiérarchiques

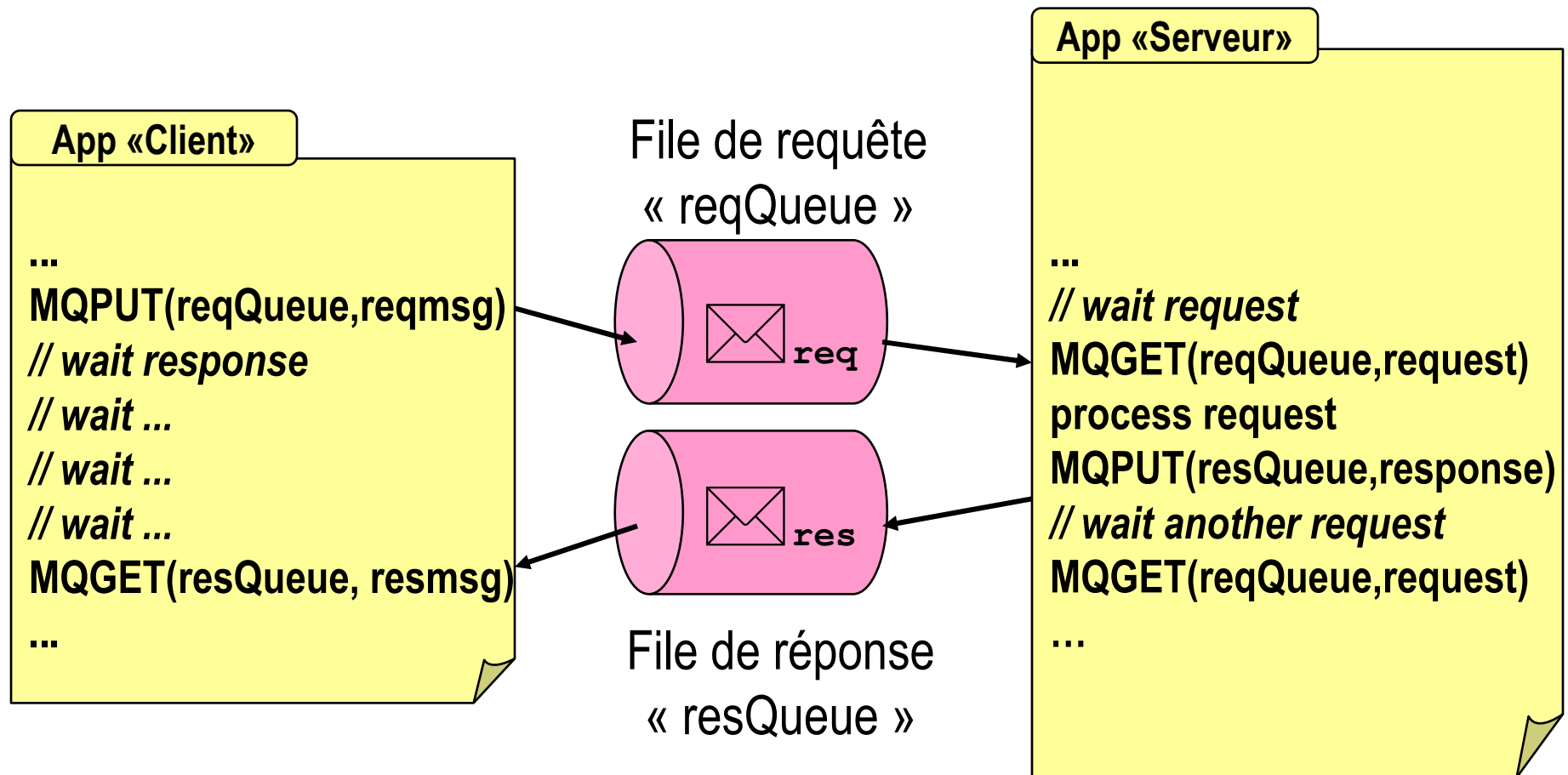


Publication-Souscription par le contenu



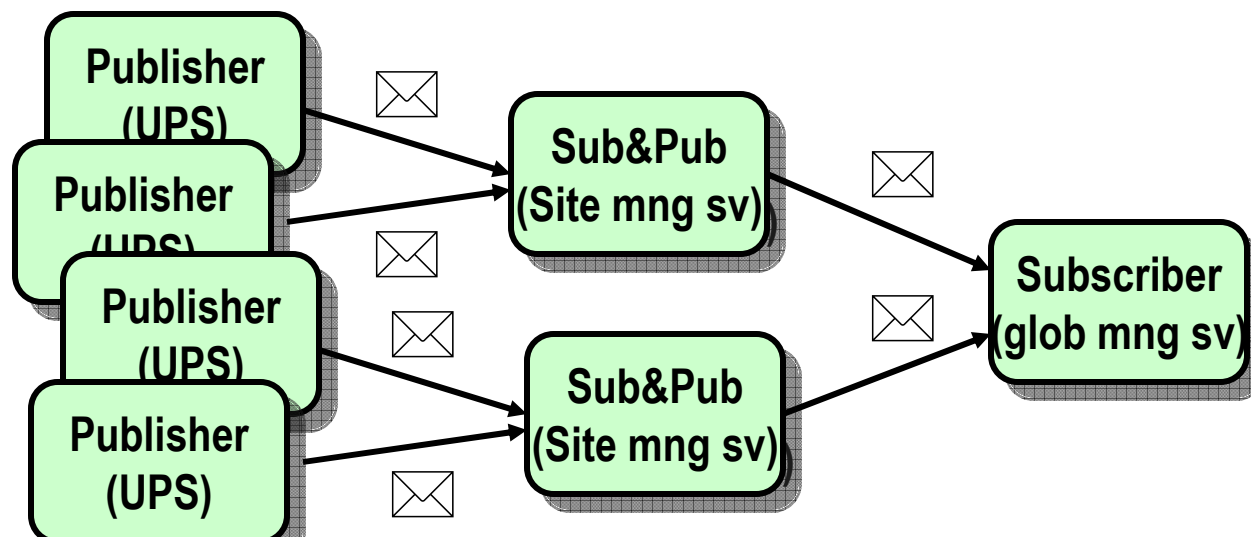
Modèle Requête-Réponse

- Implémente le modèle Client-Serveur



Modèles de messageries

- Routage hiérarchiques de Messages
 - Motivation : Passage à l'échelle de la remontée des évènements
 - Hiérarchisation des publicateurs et des souscripteurs
 - Fonctions des routeurs
 - Filtrage, fusion, store and forward
 - Exemple
 - Parc d'onduleur --- event (charge, conso, ...) ---> maintenance server
 - Réseaux d'opérateurs



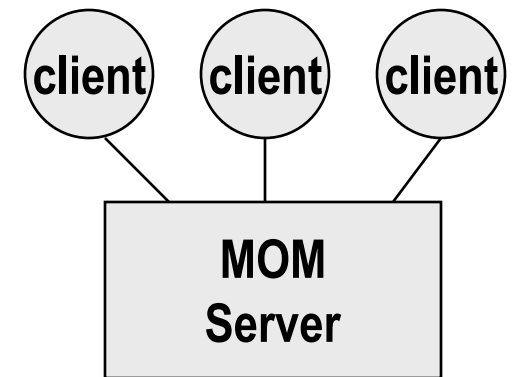
Architecture d'un MOM

- **Client MOM**
 - relié de manière permanente à un serveur MOM
 - envoie et reçoit des messages
- **Serveurs MOM**
 - reliés entre eux de manière épisodique
 - réseau mobile, réseau WAN sur lignes dédiés, ...
 - maintiennent des copies des messages
 - réplication (serveurs primaires, serveurs secondaires)
- **Administrateur/Contrôleur du MOM**
 - crée et surveille les files
 - définit la topologie des interconnexions entre serveur
 - définit la politique de connexion (période, ...)

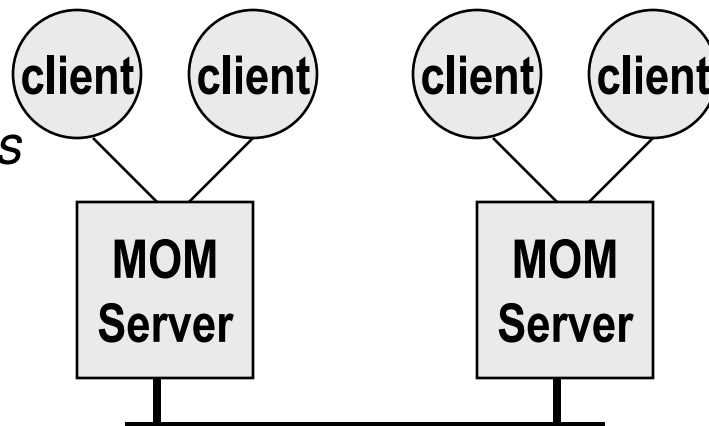
Implémentation

■ Architecture

- Centralisée : *Spoke and Hub*



- Distribuée : *Bus*

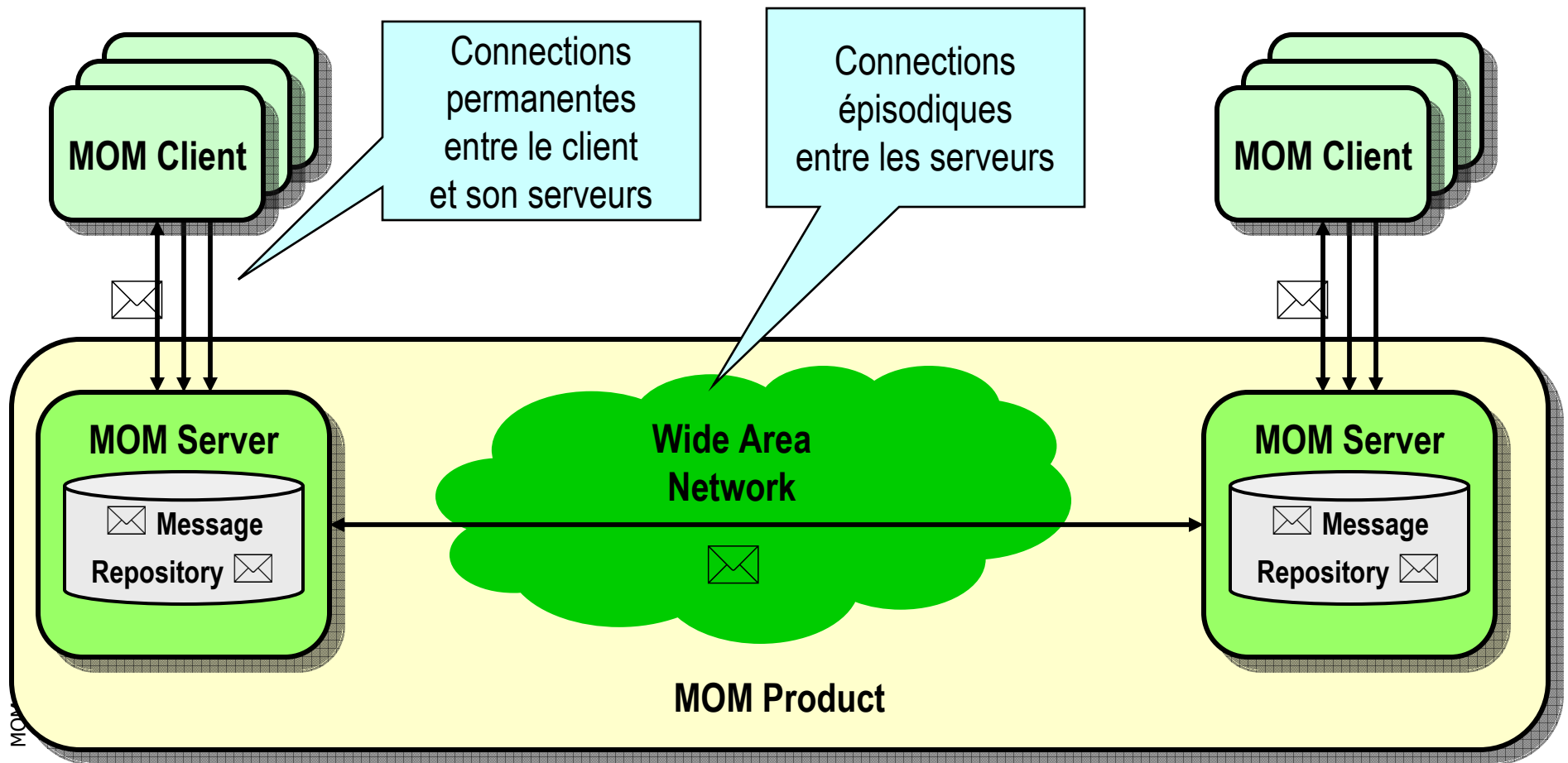


- Pair à Pair : *Snowflake*

■ QoS

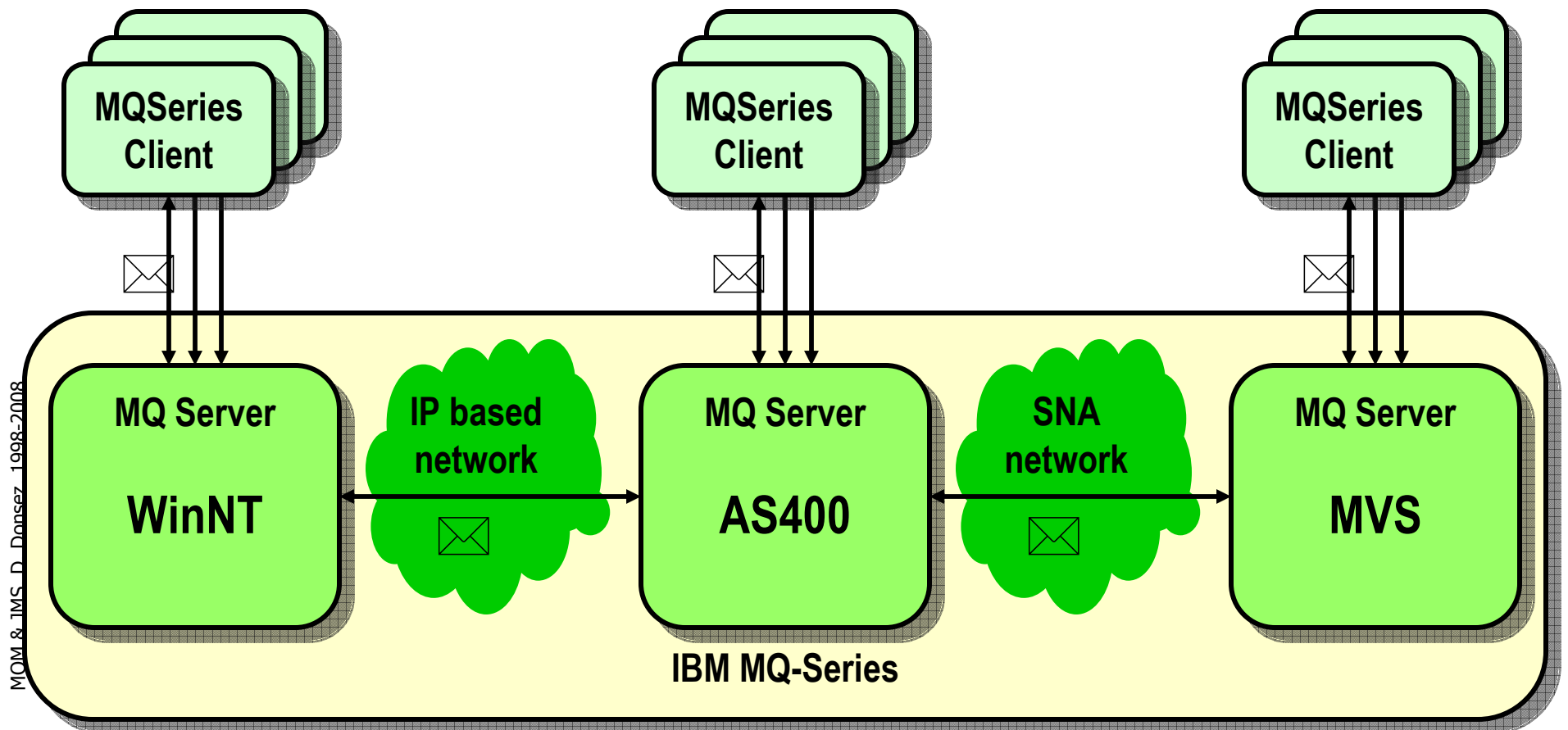
- Disponibilité, Fiabilité, Passage à l'échelle, Sécurité, ...

Architecture d'un MOM



Exemple multiplateforme d'un MOM (IBM MQ-Series)

- Hétérogénéité de Systèmes et de Réseaux



Avantages

- Réutilisation
 - Technique d 'encapsulation
 - semblable au BOA
- Fiable
- Simple d 'Utilisation
- Répandu
- Supporté par de grands acteurs

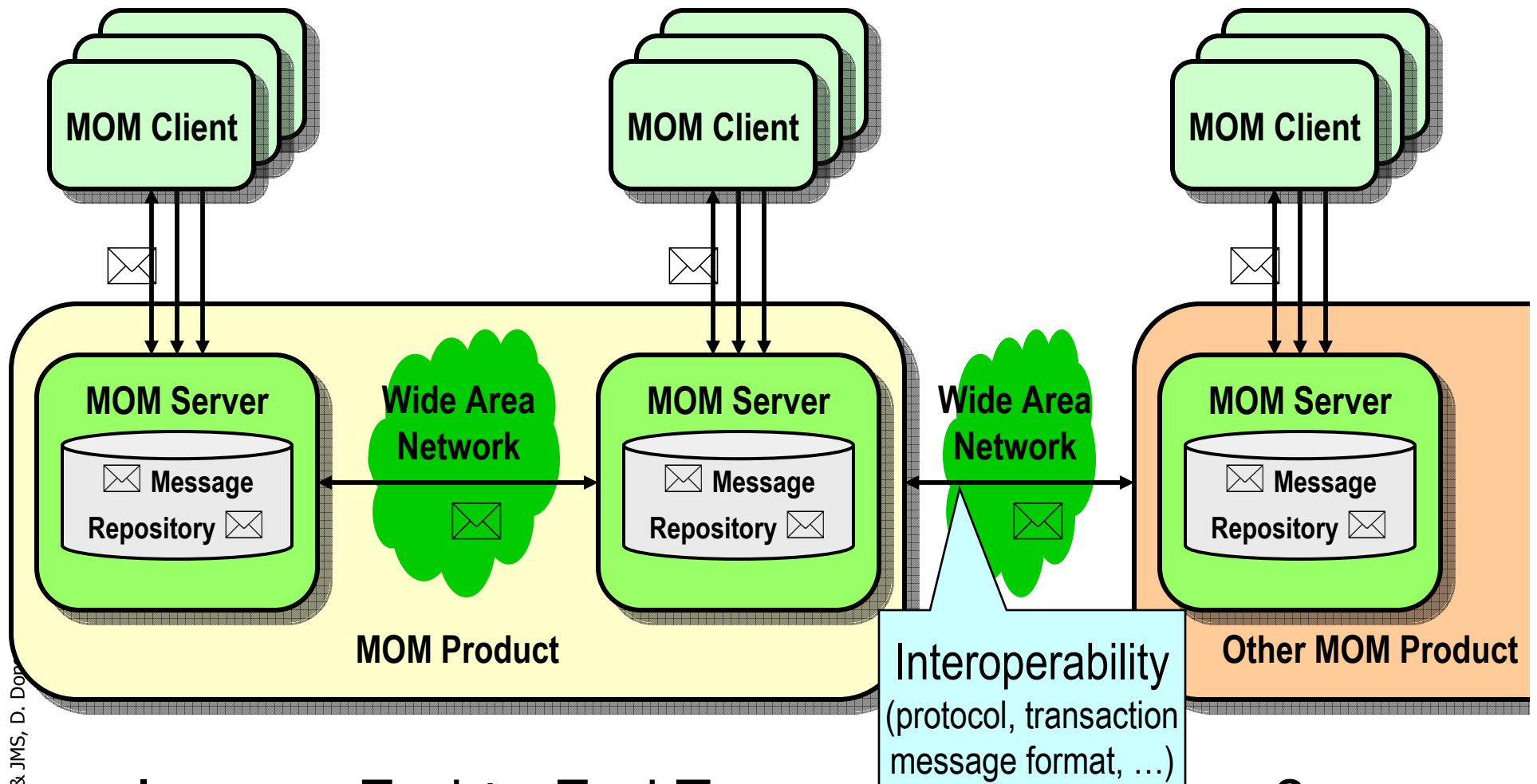
29/04/2008

Comparaison RPC et MOM

Interopérabilité entre MOM

- Difficulté de faire interopérer des MOM
- Pas de standardisation entre les MOM
 - Spécification BMQ : Business Messaging Quality
 - initiative de Candle (projet ROMA)
encouragée par IBM, MicroSoft, HP, AT&T, ... Voir <http://www.bqm.org>
- Des pistes pour l'interopérabilité
 - Une autorité : MOMA
 - Message Oriented Middleware Association
 - CORBA 3.0
 - introduction de la notion de messages asynchrones
 - J2EE
 - JMS javax.jms
 - API Java permettant à des clients d'envoyer/recevoir des messages
avec des serveurs implémentant des JMS SPI
 - EJB : Message Driven Bean

Interopérabilité entre MOM



MOM & JMS, D. Dora

- Issues : End-to-End Transactional delivery ?

AMQP

Advanced Message Queuing Protocol

- <http://amqp.org/>
 - Red Hat, Cisco Systems, IONA, iMatix, ...
- Standardiser l'échange de messages entre serveurs de message
 - Support des transactions XA

- Implémentations
 - Red Hat Enterprise MRG, IONA, ...
 - OpenAMQ, Apache QPid, ...

Le Transactionnel

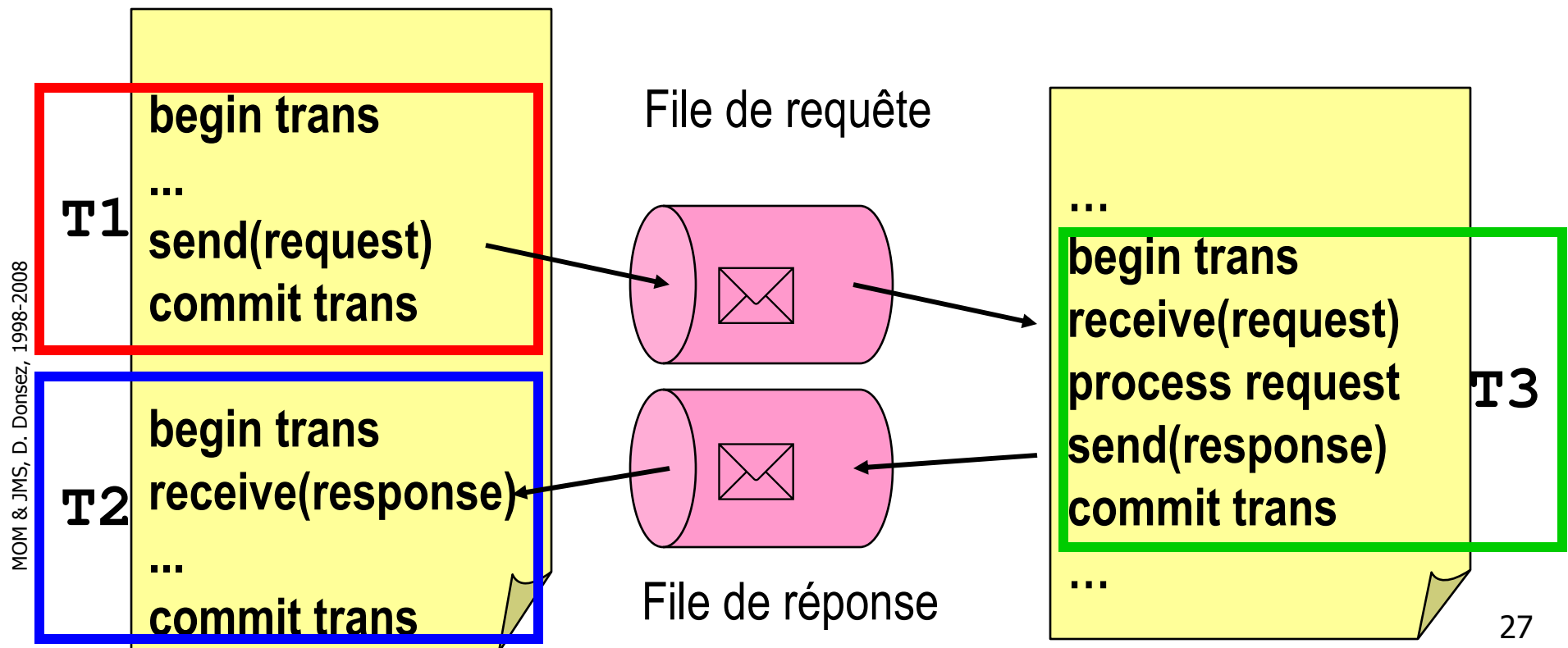
- La consommation et la production de messages peuvent être des actions recouvrables
 - la file des messages est considérée comme une ressource recouvrable

- Elles ne sont effectives qu'à la validation d'une transaction
 - tous les messages produits sont envoyés à la validation
 - en cas d'abandon de la transaction, les messages produits sont abandonnés et les messages consommés ne sont retirés de la file

- La transaction peut être distribuée
 - Moniteur transactionnel (XA, MTS, ...)

Conséquences du Transactionnel

- Conception de Requête-Réponse transactionnel
 - l'envoi de la requête et la réception de la réponse sont forcement dans 2 transactions successives



Conséquences du Transactionnel

- L'ordre de consommation des messages peut être différent de l'ordre de production

```
begin T1
T1 produit M1
T1 produit M2
commit T1
begin T2
T2 consomme M1
begin T3
T3 consomme M2
abort T2
commit T3
begin T4
T4 consomme M1
commit T4
```

Acteurs et Produits

- BEA Systems
- IBM - MQ Series
 - 25 plateformes
- MicroSoft - MSMQ (Message Queue Server)
 - essentiellement NT
- Level 8 Systems - Falcom MQ
 - passerelle vers MSMQ et MQ Series
- Sybase - DBQ
 - Adaptive Serveur
- Tibco - TIB/RendezVous
 - accord avec Oracle pour Oracle 8
- Sun - Java Messaging Service
 - API pour les MQ

IBM MQ/Series

- Leader du marché (66% du marché)
- Plates-formes
 - >20 plates-formes
 - 5 protocoles réseaux
 - langages (C++, C, Cobol, Java, PL/1, ...)
- Nombreux modules
 - Publish/Subscribe, Workflow, ...

MSMQ (MicroSoft Message Queue)

- Plates-formes NT/2000 (v2) et XP (v3)
 - Réseaux IP et IPX
 - IP Multicast (avec PGM pour la tolerance aux pertes) (v3)
 - Transport sur HTTP/HTTPS et message à enveloppe SOAP (v3)
- Modèles (v3)
 - One-To-One, One-To-Many
 - Distribution Lists
 - Real-Time Messaging Multicast
 - Message Queuing Triggers
 - (activation d'une méthode d'un objet COM sur reception)
 - SDK MSMQ pour C, C++, ActiveX, MSMQ Explorer

MSMQ (MicroSoft Message Queue)

- Serveur (v2)
 - 4 types de serveur
 - PEC pour Primary Enterprise Controller
 - informations sur la topologie (sites, liaisons entre sites et RC)
 - PSC pour Primary Site Controller
 - informations sur les sites (serveurs, clients et files d'attente)
 - BSC pour Backup Site Controller
 - secours et équilibrage de charge de PSC
 - RS pour Routing Server
 - MSMQ Information Store (MQIS)
 - référentiel (utilise SQL Server ou Active Directory)
 - Dépôt transactionnel de message (MTS)
 - 2 Go par file (v2), 1 To par queue (v3)
- Client
 - Windows CE, Win9x, ...
 - SDK MSMQ pour C, C++, ActiveX, MSMQ Explorer

Exemple d 'ASP utilisant MSMQ et MTS

```

<%@ TRANSACTION=REQUIRED LANGUAGE=JScript %>
<HTML><HEAD><TITLE>Envoi transactionnel par MSMQ</TITLE></HEAD><BODY>
<h1>Envoi transactionnel par MSMQ</h1><hr>
<%
    QueueInfo = Server.CreateObject("MSMQ.MSMQQueueInfo")
    QueueInfo.pathname = ".\IIS_SDK_TRANSACTED";
    Queue = QueueInfo.Open(2, 0);
    Msg = Server.CreateObject("MSMQ.MSMQMessage");
    Msg.body = "Corps du Message";    Msg.Label = "Label du Message";
    Msg.Delivery = 1;        // recouvrable : résiste au crash et au shutdown
    Msg.PrivLevel = 1;      // chiffré
    Msg.Send(Queue);
    Queue.Close();
%>
</BODY></HTML>
<%
    function OnTransactionCommit() {
        Response.Write("<p>La transaction est validée et le message MSMQ est envoyé.");}

    function OnTransactionAbort() {
        Response.Write("<p>La transaction est abandonnée");
        Response.Write("et le message MSMQ n 'a pas été envoyé."); } %>

```

JORAM (ObjectWeb & Scalagent)

■ MOM JMS

- Destination : PtoP (Queue) et PubSub (Topic)
- Architecture Multi-Serveurs
- Open Source
- Intégré à JONAS
- Disponibilité sur OSGi pour déployer des bundles OSGi
- Version kJORAM pour KVM
- Administration par des MBeans (Console JMX)

■ Utilisation

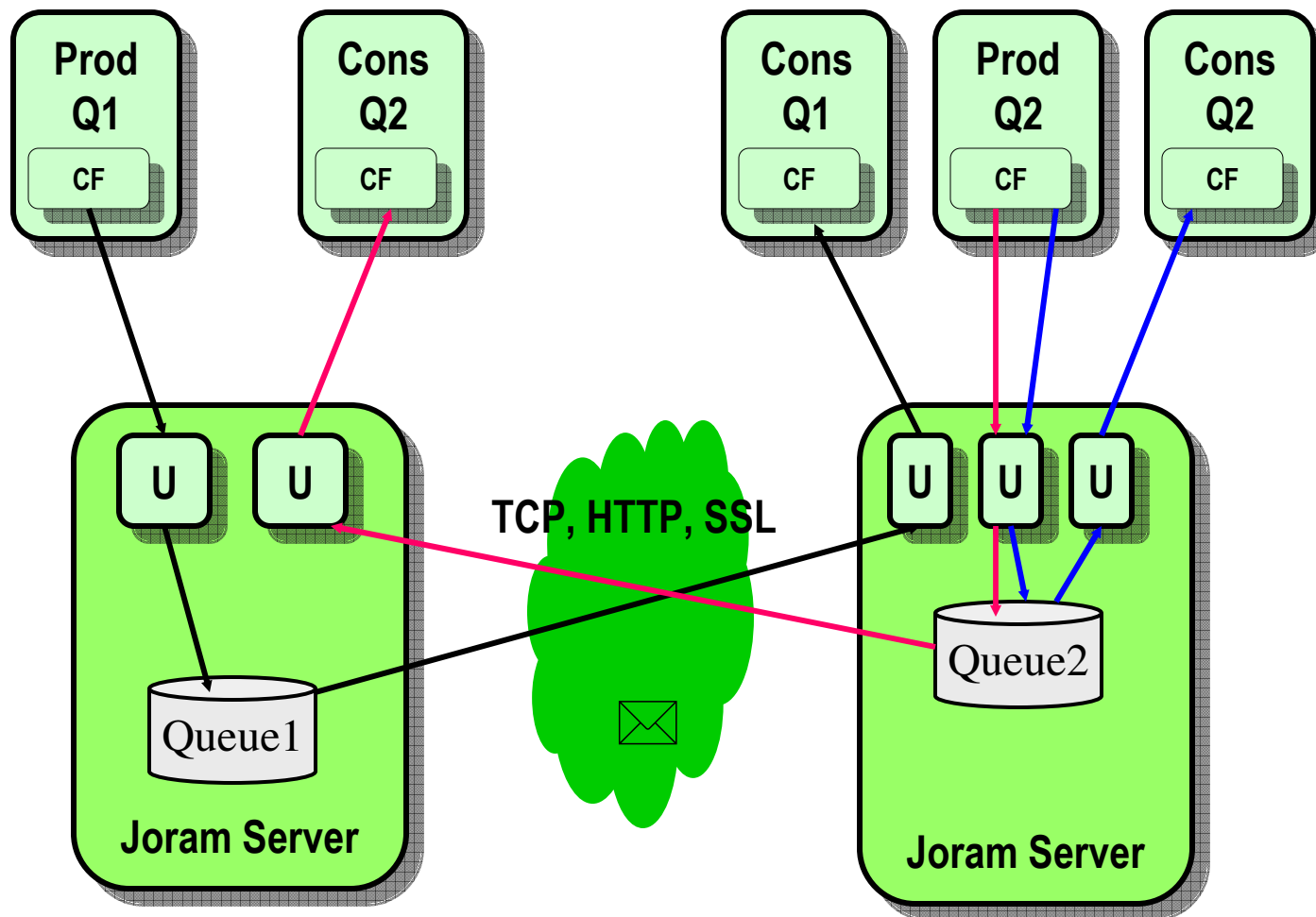
- Kelkoo (remontée de log)
- Schneider Electric (remontée de mesures de capteurs)
- ...

JORAM (ObjectWeb & Scalagent)

- Architecture Multi-Serveurs
 - Une destination par serveur
 - La ConnectionFactory est connecté au serveur
 - Equilibrage de charge (*Load Balancing*)
 - La Destination est répliquée sur R serveurs (pair à pair)
 - Connections: TCP, HTTP, SSL, ...
 - Privilégie la consommation locale des messages
 - Pas d'ordre globale des messages
 - Ordre local
 - Haute disponibilité (*High Availability*)
 - Serveur maître répliquant (JGroup) ses queues/topics sur S serveurs esclaves ($S > 0$)
 - La ConnectionFactory du client JMS peut basculer du serveur maître vers un des serveurs esclaves

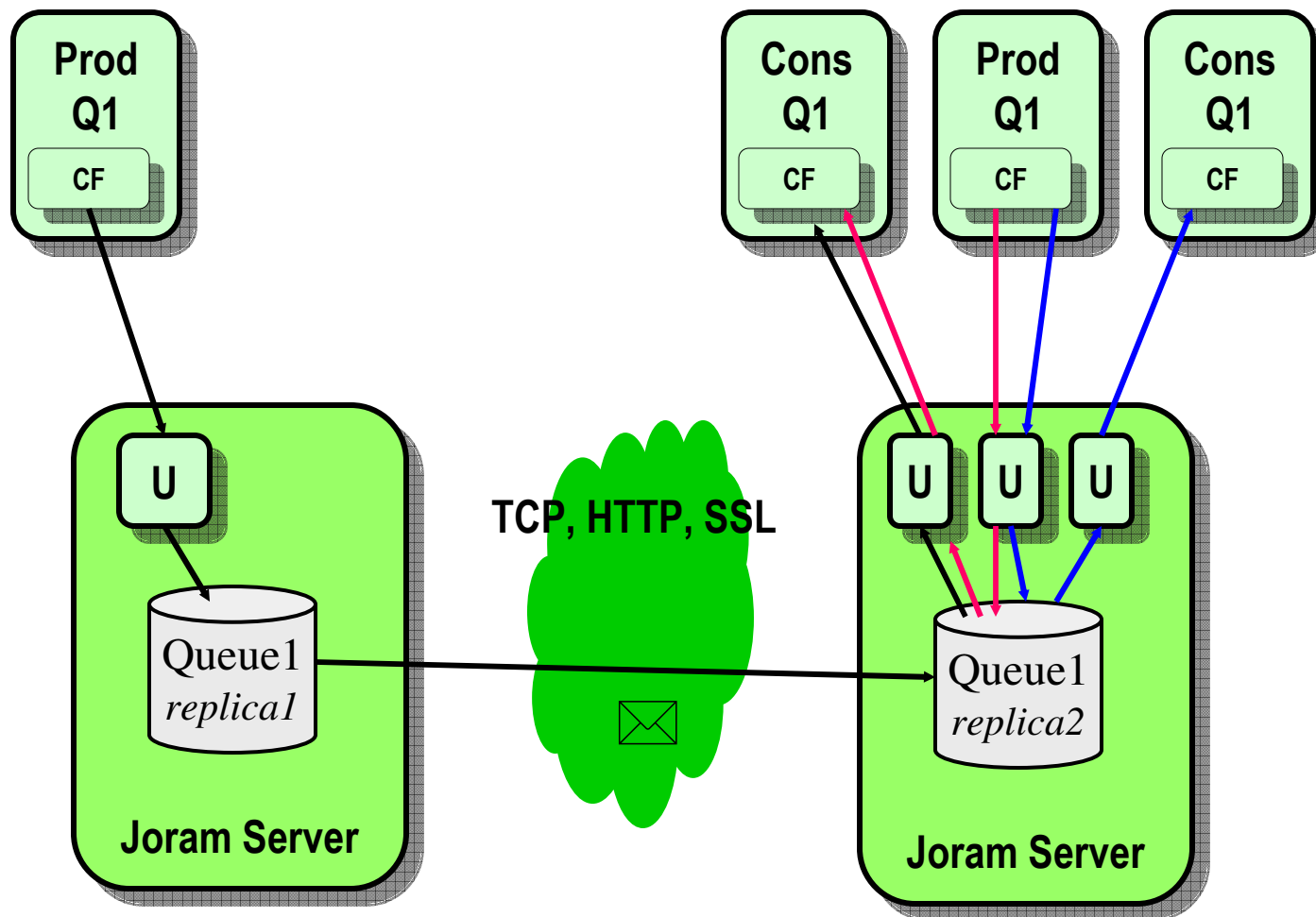
JORAM (ObjectWeb & Scalagent) Architecture Multi-Serveurs

- Une destination par serveur



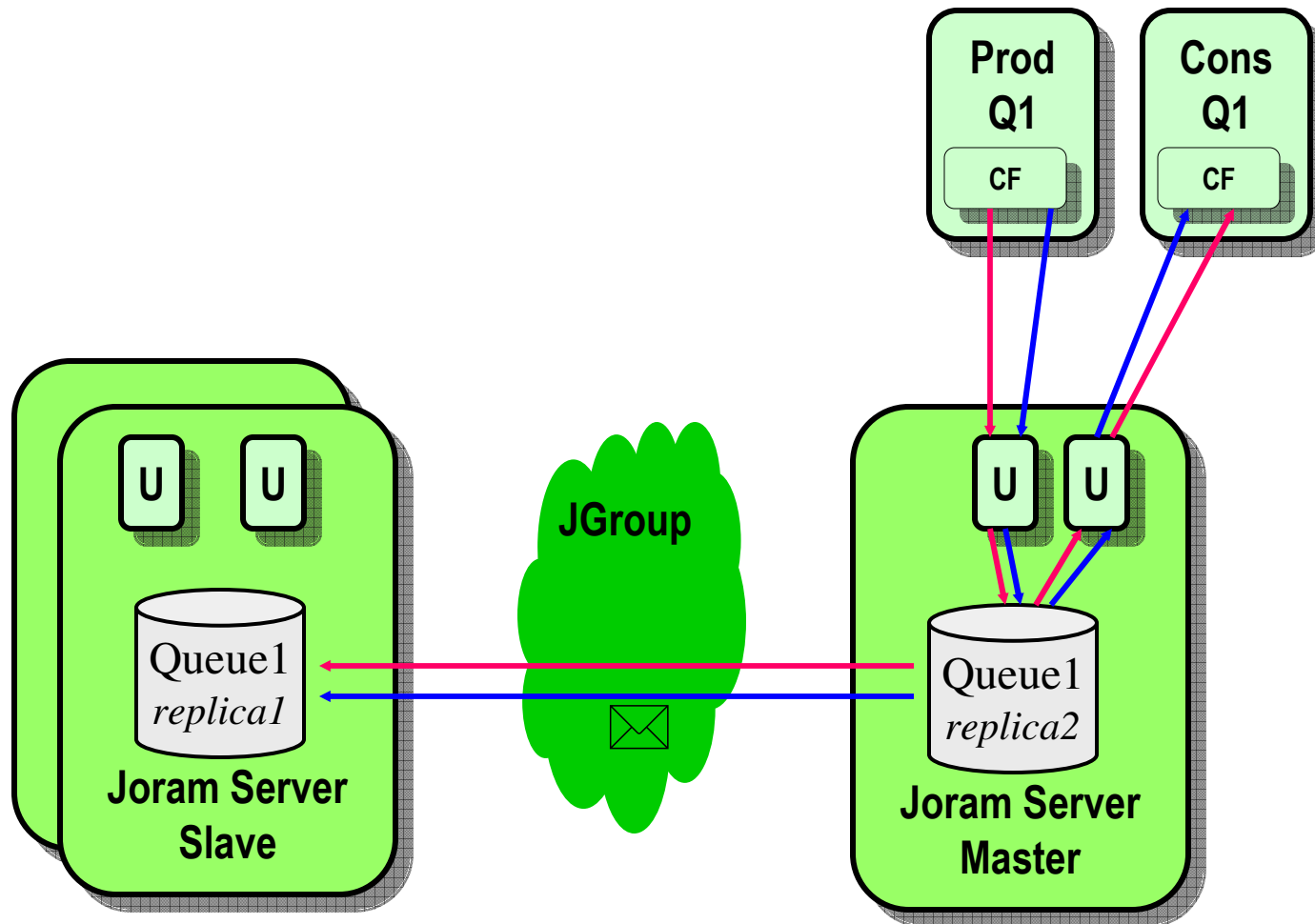
JORAM (ObjectWeb & Scalagent) Architecture Multi-Serveurs

- Equilibrage de charge



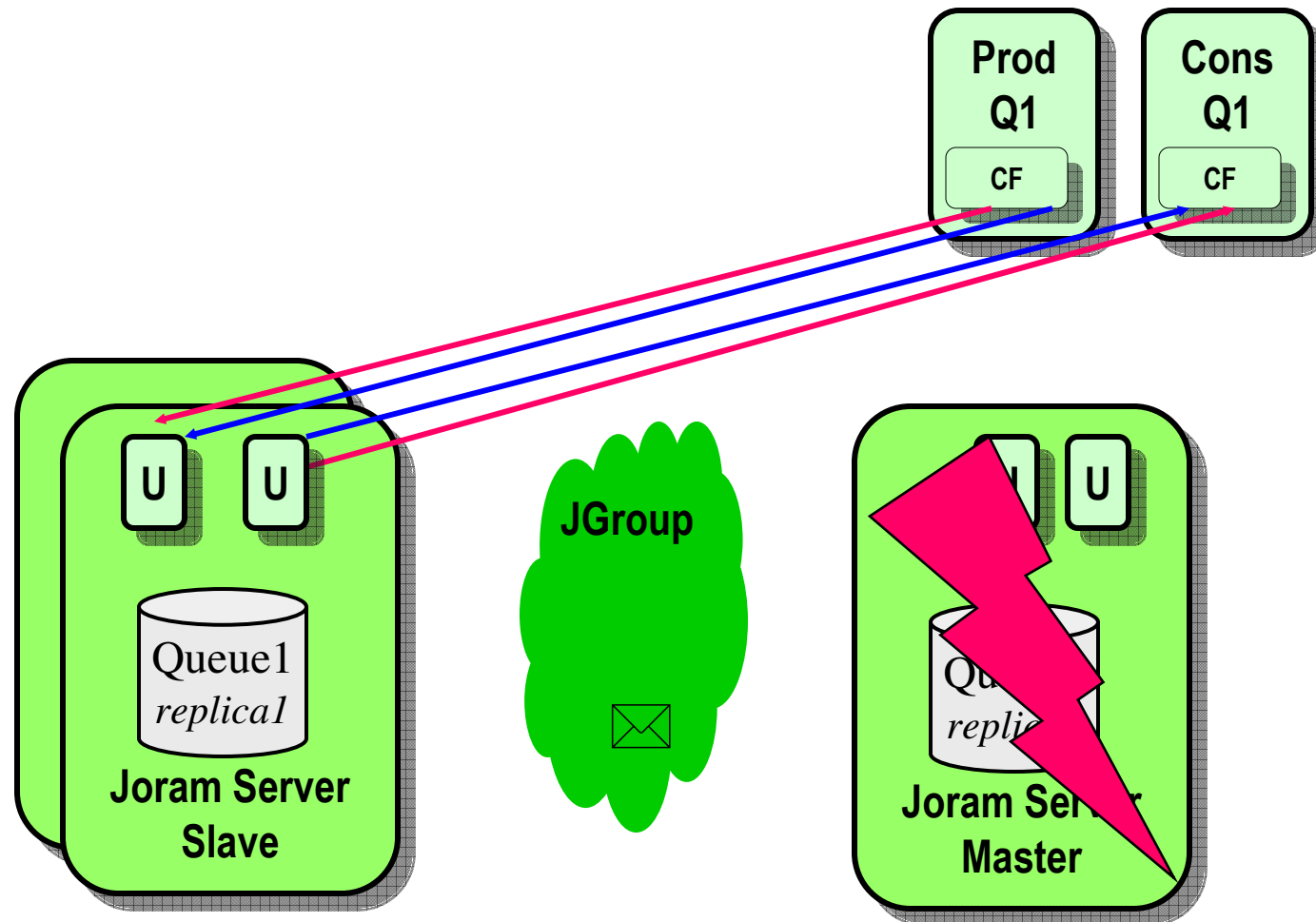
JORAM (ObjectWeb & Scalagent) Architecture Multi-Serveurs

- Haute disponibilité (1)



JORAM (ObjectWeb & Scalagent) Architecture Multi-Serveurs

- Haute disponibilité (2)



MOM et BDs Nomades (Mobiles)

- Motivation
 - BD Nomades
 - Laptops, PDA, ...
 - Mise à jour asynchrone des réplicats nomades
 - car les BDs nomades ne sont toujours connectés au site central

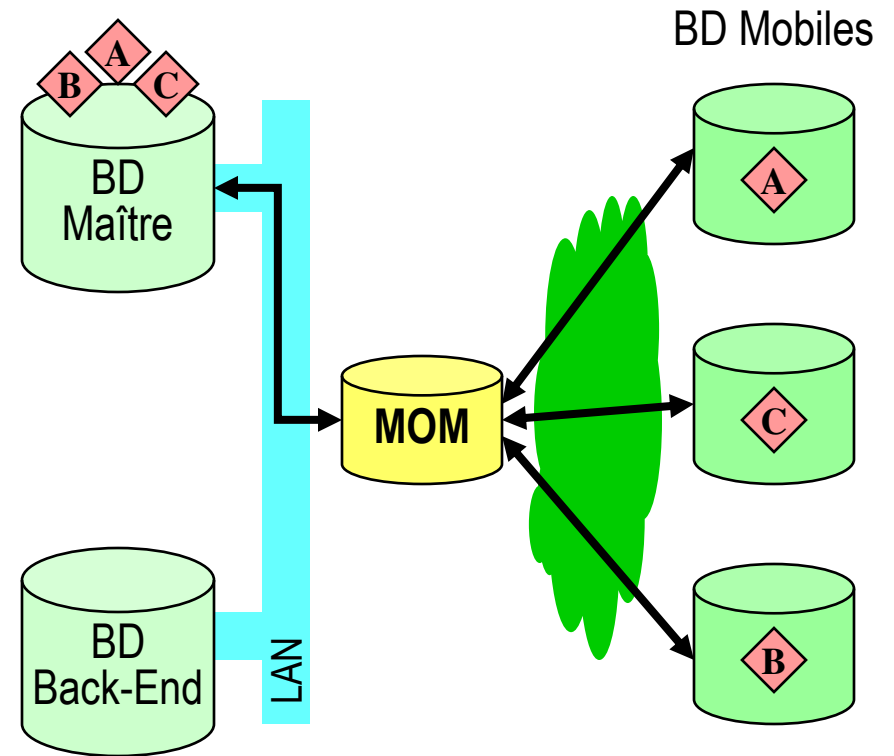
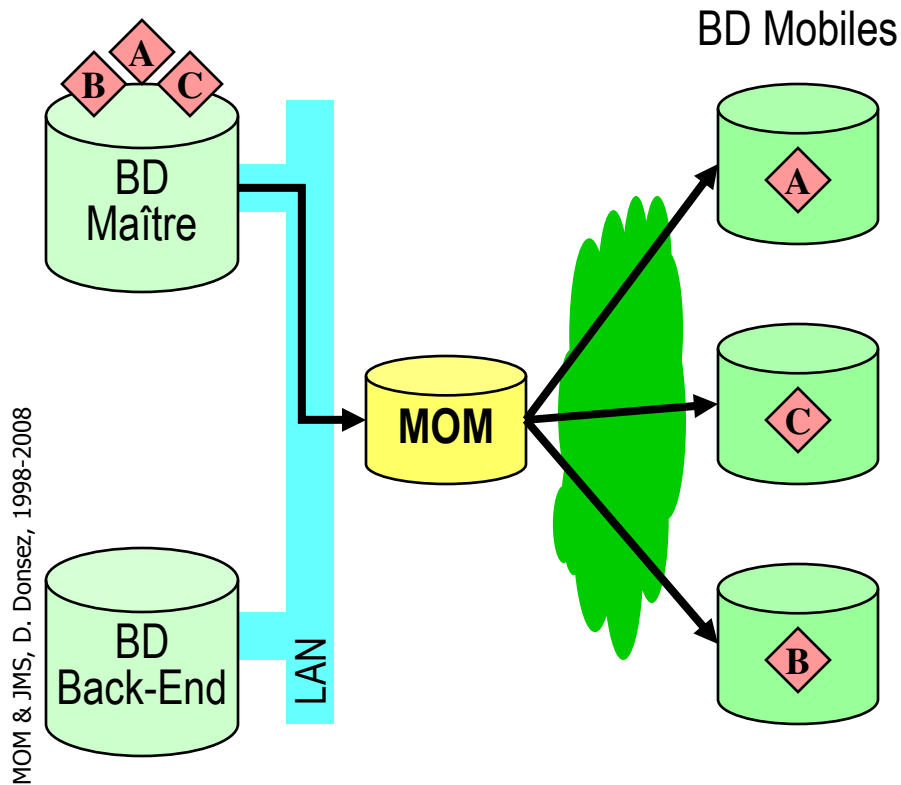
- Réplication des BDs
 - voir le cours sur les « BDs Distribuées et la Réplication »

- Produits
 - RemoteWare de Xcellenet, MediaTransfer de Telelogo,

MOM et BDs Nomades (Mobiles)

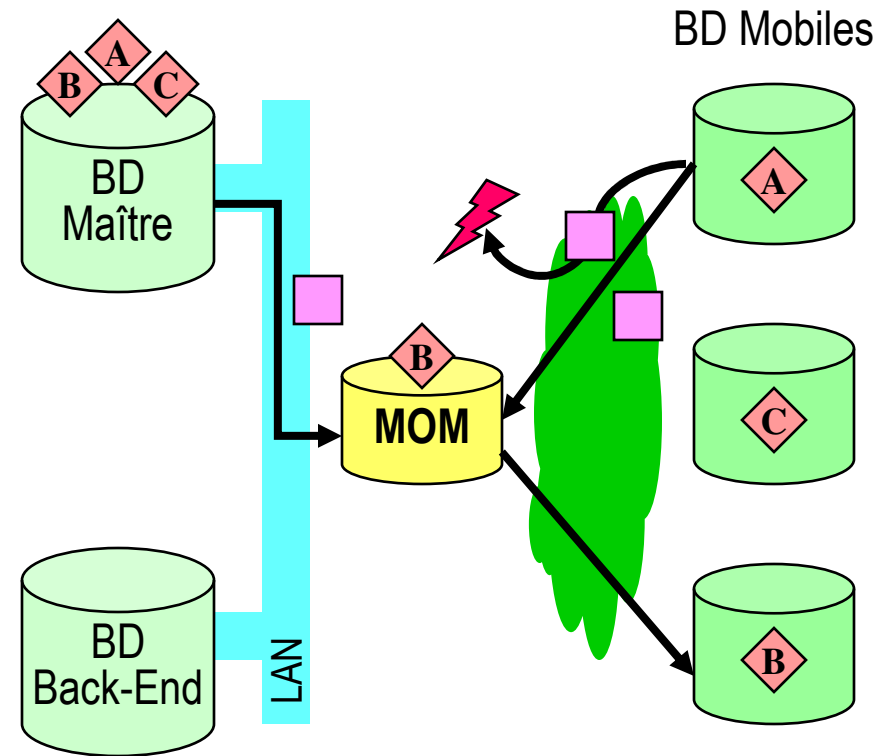
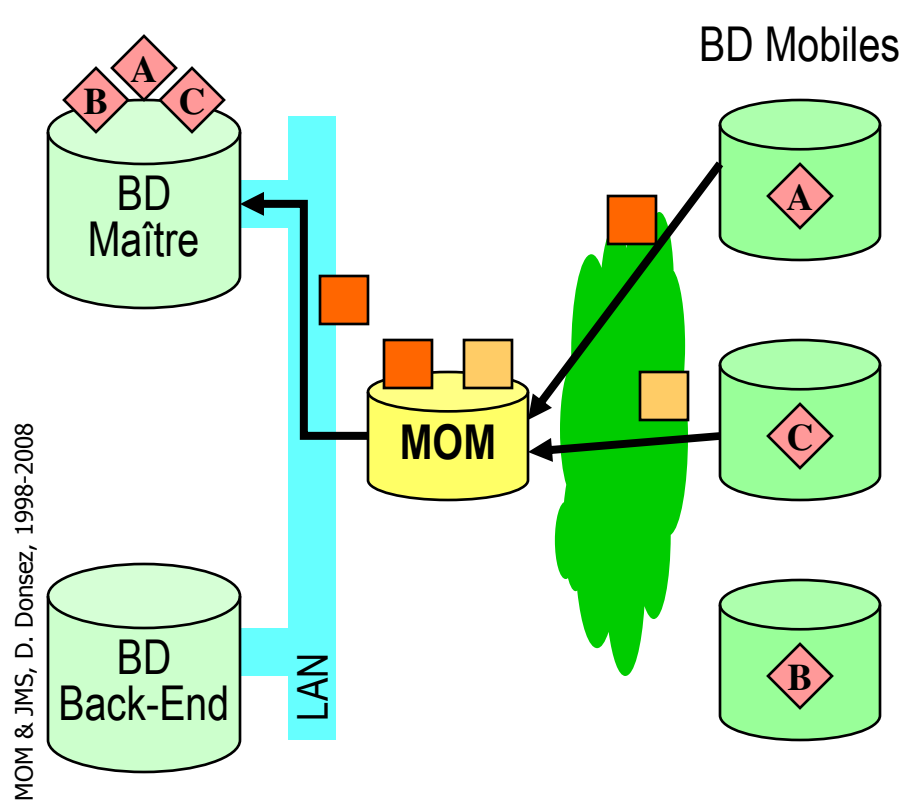
- 1- Réplication

- 2- Synchronisation



MOM et BDs Nomades (Mobiles)

- 3- Détection de Collision
- 4- Reprise sur Erreur



MOM & JMS, D. Donsez, 1998-2008

MOM et Composants

- Motivations
 - Fournir la couche de communication pour le paradigme Événement dans des modèles à composants (qui le supportent)
- Modèles
 - CORBA CCM
 - .NET Asynchronous [OneWay] calls
 - J2EE/EJB Message Driven Beans (pas de typage des msg)

Bibliographie

- Gregor Hohpe, Enterprise Integration Patterns, <http://www.enterpriseintegrationpatterns.com>
 - Très bon livre traitant de l'utilisation des MOMs

<http://www-adele.imag.fr/users/Didier.Donsez/cours>

Java Message Service (JMS)

Didier DONSEZ

Université Joseph Fourier (Grenoble 1)

PolyTech'Grenoble LIG/ADELE

`Didier.Donsez@imag.fr`

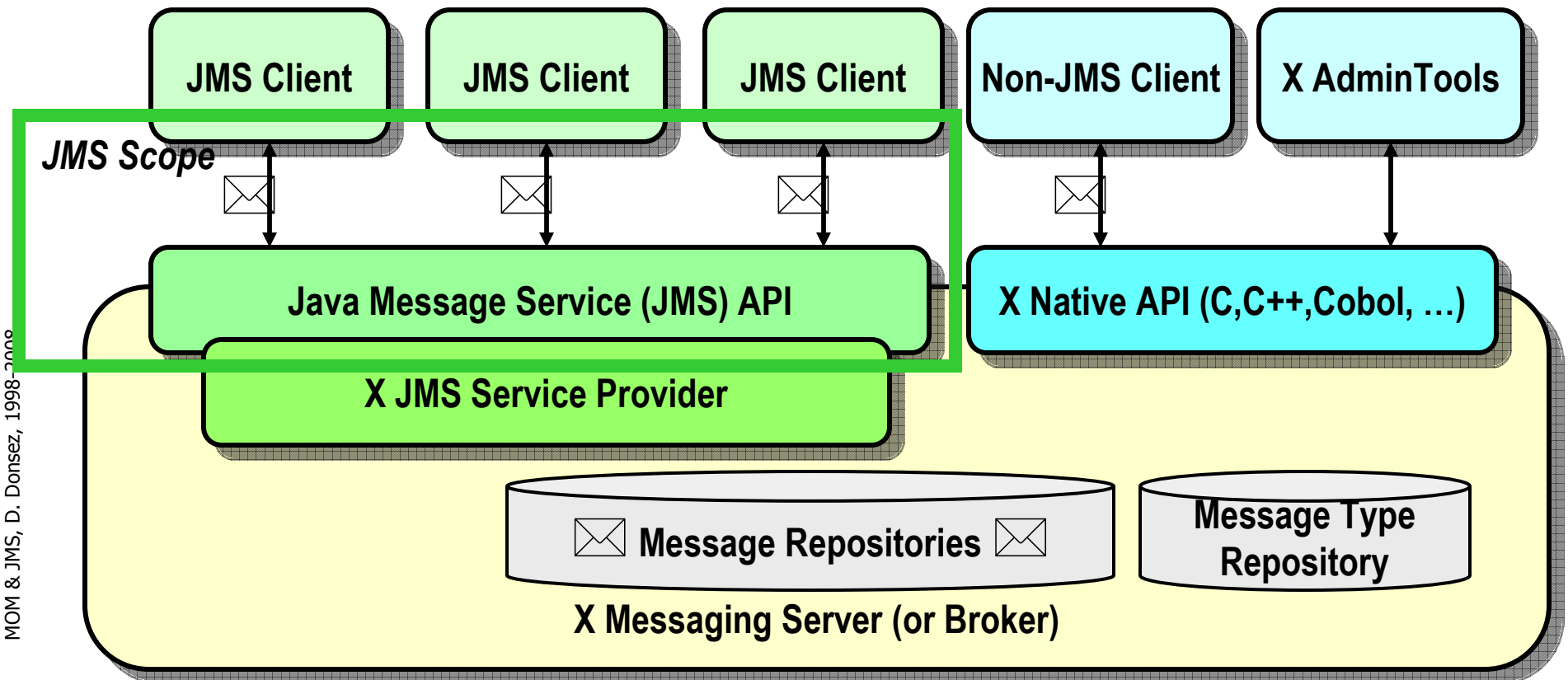
`Didier.Donsez@ieee.org`

Motivation

- Messaging Oriented Middleware
 - messagerie inter-applicative
 - l'envoi et la réception des messages entre applications est asynchrone
 - les messages sont structurés et correspondent à des événements, des requêtes, des rapports, ...
 - ne nécessite pas de connexion permanente comme pour le C/S
 - *ne pas confondre avec le courrier électronique (API JavaMail)*
- `javax.jmx` API Java d'un client à un serveur MOM
 - Modèles de messagerie (*messaging*)
 - Point à Point (*Point-to-Point*)
 - concept de **Queue**, un file d'attente de messages
 - Publication-Souscription (*Publish-Subscribe*)
 - concept de **Topic**, un sujet auquel s'abonne un ou plusieurs *Subscribers*
 - support pour les transactions distribuées

Architecture JMS

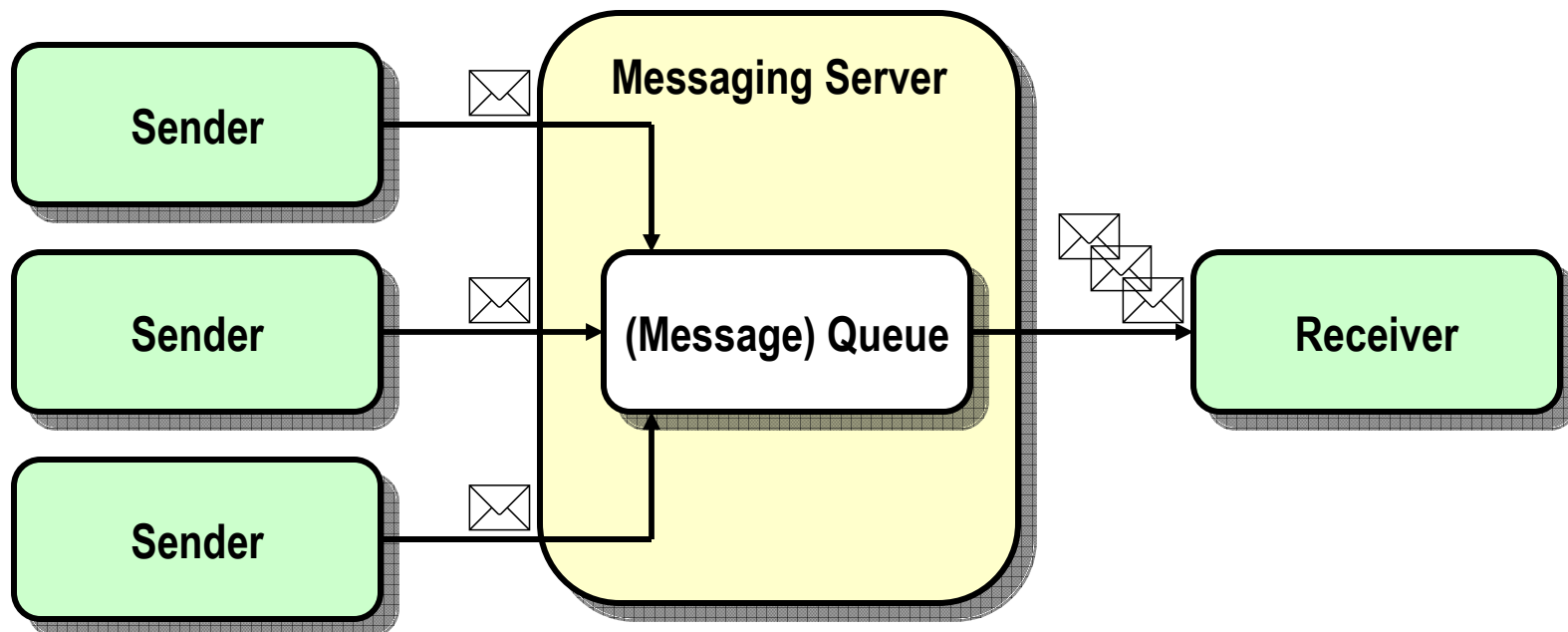
- le client utilise les classes du package `javax.jmx` pour l'envoi et la réception de messages
- le serveur implante un JMS Service Provider qu'interface à son noyau



Modèle Point à Point

Point-to-Point

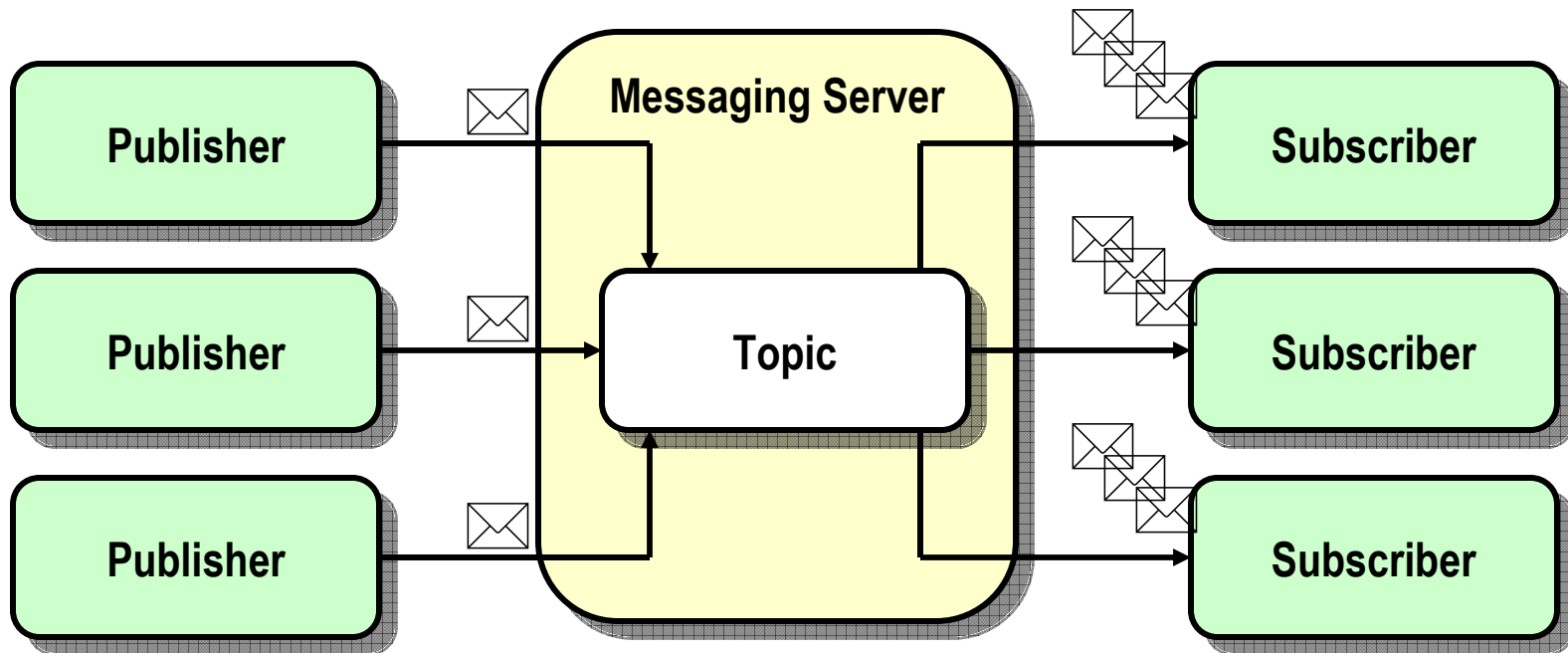
- Concept de Queue
 - Un (ou plusieurs) *Sender* envoie (*produit*) un message à une *Queue* (i.e. file d'attente de messages)
 - Un et un seul *receiver* reçoit (*consomme*) les messages de la *Queue*



Modèle Publication-Souscription

Publish-Subscribe

- Concept de Topic (centre d'intérêt)
 - Un (ou plusieurs) *publishers* envoient un message à un *Topic*
 - Tous les *subscribers* de ce *Topic* reçoivent le message



API JMS

■ Superclasses communes a PTP et PubSub

JMS Parent

Modèle PTP

Modèle Pub/Sub

ConnectionFactory

QueueConnectionFactory

TopicConnectionFactory

- *un objet administré par le provider pour créer une Connection*

Connection

QueueConnection

TopicConnection

- *une connexion active vers un JMS provider*

Destination

Queue

Topic

- *encapsule l'identité d'une destination*

Session

QueueSession

TopicSession

- *contexte (mono-thread) pour l'émission et la réception de messages*

MessageProducer

QueueSender

TopicPublisher

- *objet pour la production (l'envoi) de messages vers une Destination (créé par la Session)*

MessageConsumer

QueueReceiver, QueueBrowser

TopicSubscriber

- *objet pour la consommation (la réception) de messages d'une destination (créé par la Session)*

Fonctionnement d'un client typique JMS

- Phase d'initialisation (setup)
 - trouver l'objet ConnectionFactory par JNDI
 - trouver un (ou plusieurs) objet Destination par JNDI
 - créer une Connection JMS
 - créer une (ou plusieurs) Session avec la Connection JMS
 - créer le(s) MessageProducer ou/et MessageConsumer avec la Session et la (les) Destination
 - demander à la Connection de démarrer la livraison des messages
- Phase de consommation/production de messages
 - créer des messages et les envoyer
 - recevoir des messages et les traiter

Les objets administrés par le Provider

- Les objets administrés par le Provider
 - ConnectionFactory
 - point d'accès à un serveur MOM
 - accessible par JNDI et enregistré par l'administrateur du serveur MOM
 - Destination
 - Queue ou Topic administré par le serveur MOM
 - accessible par JNDI et enregistré par l'administrateur du serveur MOM
- Les objets
 - Connection
 - encapsule la liaison TCP/IP avec le Provider JMS
 - authentifie le client et spécifié un identifiant unique de client
 - crée les Sessions et fournit le ConnectionMetaData
 - supporte (optionnel) l'ExceptionListener
 - Session
 - encapsule la liaison TCP/IP avec le Provider JMS

javax.jms.Connection

- Rôle
 - encapsule la liaison TCP/IP avec le Provider JMS
 - authentifie le client et spécifié un identifiant unique de client
- Cycle de vie
 - initialisation/setup
 - crée les Sessions, MessageProducers et les MessageConsumers.
 - fournit le ConnectionMetaData
 - supporte (optionnel) l' ExceptionListener
 - démarrage start()
 - procède à la livraison des Messages
 - pause stop()
 - interrompt la livraison des Messages entrants, n' affecte pas l' envoi
 - reprise start()
 - fermeture close()
 - libération des ressources

javax.jms.Session

■ Rôle

- contexte mono-threadé
- fabrique les MessageProducers et les MessageConsumers
- fabrique les destinations temporaires TemporaryDestination
- définit les numéros de série des messages (consommés et produits)
 - un ordre par Session et par Destination. Pas d'ordre entre Destinations
- sérialise l'exécution des MessageListeners enregistrés
 - méthode onMessage()
- définit une chaîne de transactions atomiques
 - la portée est définie par commit() ou rollback()
- Acquittement des messages

javax.jms.MessageConsumer

- représente l'objet consommateur des messages
 - fabriqué par la Session
 - `QueueReceiver QueueSession.createReceiver(Queue queue, String messageSelector)`
 - `TopicSubscriber TopicSession.createSubscriber(Topic topic, String messageSelector, boolean)`
 - Remarque : il peut y avoir plusieurs `MessageConsumers` sur une `Session` pour une même `Destination`
 - Sous-interfaces : `QueueReceiver`, `TopicSubscriber`
- Réception Synchrones des messages
 - le client se met en attente du prochain message
 - `Message receive()`, `Message receive(long timeout)`, `Message receiveNoWait()`
- Réception Asynchrone des messages
 - le client crée un objet qui implémente l'interface `MessageListener` et positionne cet objet à l'écoute des messages
 - `void setMessageListener(MessageListener) / MessageListener getMessageListener()`
 - quand un message arrive, la méthode `void onMessage(Message)` de l'objet `MessageListener` est invoqué
 - La session n'utilise qu'une seule thread pour exécuter séquentiellement tous les `MessageListeners`

javax.jms. MessageProducer

- représente l'objet producteur des messages
 - fabriqué par la Session
 - `QueueSender QueueSession.createSender(Queue queue)`
 - `TopicPublisher TopicSession.createPublisher(Topic topic)`
 - Remarque : il peut y avoir plusieurs MessageProducers sur une Session pour une même Destination
 - Sous-interfaces : QueueSender, TopicPublisher
- Production des messages
 - `void QueueSender.send(Message)`
 - `void TopicPublisher.publish(Message)`

Le modèle Point à Point

Point-To-Point PTP

■ Classes et interface

JMS Parent

ConnectionFactory

Connection

Destination

Session

MessageProducer

MessageConsumer

--

Modèle PTP

QueueConnectionFactory

QueueConnection

Queue, TemporaryQueue

QueueSession

QueueSender

QueueReceiver

QueueBrowser

■ Remarques

- plusieurs sessions peuvent se partager une même queue : JMS ne spécifie pas comment le provider réparti les messages entre les QueueReceiver de ces sessions.
- Les messages non sélectionnés restent dans la queue : l'ordre de réception n'est plus alors l'ordre de production
- QueueBrowser permet de consulter les messages de la Queue sans les retirer en les énumérant (méthode Enumeration getEnumeration())

Le modèle Publication-Souscription

Publish-Subscribe PubSub

■ Classes et interface

JMS Parent

ConnectionFactory

Connection

Destination

Session

MessageProducer

MessageConsumer

Modèle Pub/Sub

TopicConnectionFactory

TopicConnection

Topic

TopicSession

TopicPublisher

TopicSubscriber

■ Remarque

- le terme « publier » correspond à « produire »
- le terme « souscrire » correspond à « consommer »

Le modèle de Messages JMS

L 'interface javax.jms.Message

- Motivation
 - Modèle commun à tous les produits
 - Supporte l'hétérogénéité des clients (non JMS, langage, plate-forme, ...)
 - Supporte les objets Java et les documents XML
- L 'interface *javax.jms.Message*
 - sous-interfaces: *BytesMessage*, *MapMessage*, *ObjectMessage*, *StreamMessage*, *TextMessage*
- Structure d 'un Message
 - Entête (*header*)
 - champs communs aux Providers (Produit) et obligatoires
 - destinés au routage et l 'identification du message
 - Propriété (*property*)
 - champs supplémentaires
 - propriétés standards : équivalent aux champs d 'entête optionnels
 - propriétés spécifiques au (produit) provider
 - propriétés applicatives : peuvent répliquées le contenu d 'une valeur du corps
 - Corps (*body*)

Le modèle de Messages JMS

Les champs d'entête

■ Nom des champs d'entête

JMSDestination : destination du message

JMSDeliveryMode : sûreté de distribution

NON_PERSISTENT : faible

PERSISTENT : garantie supplémentaire qu'un message ne soit pas perdu

JMSExpiration : calculé à partir du TTL (Time To Live) depuis la prise en charge

JMSMessageID : identifiant du Message fourni par le provider

JMSTimestamp : date de prise en charge du message par le provider

JMSCorrelationID : identifiant d'un lien avec un autre Message (Request/Reply)

JMSReplyTo : identifiant de la Destination pour les réponses

JMSType : identifiant du type de message dans le Message Type Repository

JMSRedelivered : le récepteur n'acquiesce pas immédiatement la réception (Transaction)

JMSPriority : priorité (de 0 à 9) du message

■ Consultation/Modification

- méthodes setXXX() / getXXX() où XXX est le nom du champs d'entête

Le modèle de Messages JMS

Les champs de propriétés

- Nom des propriétés
 - propriétés standards : nom préfixé par JMSX
 - correspondent à des champs d'entête optionnels
 - JMSXUserID, JMSXAppID** : identité de l'utilisateur et de l'application
 - JMSXGroupID, JMSXGroupSeq** : groupe de messages et son numéro de séq
 - JMSXProducerTXID, JMSXConsumerTXID** : identifiants de transaction
 - JMSXRcvTimestamp** : date de réception
 - JMSXState** : 1(waiting), 2(ready), 3(expired), 4(retained)
 - JMSXDeliveryCount** : The number of message delivery attempts
 - propriétés spécifiques : nom préfixé par JMS_VendorName
 - propriétés applicatives : autre
 - servent au filtrage
 - servent comme données complémentaires au corps
 - exemple : la date d'expiration d'un document XML véhiculé par un StringMessage

Le modèle de Messages JMS

Les champs de propriétés

■ Enumération des propriétés

Enumeration getPropertyNames() / Enumeration ConnectionMetaData.getJMSXPropertyNames()

énumère les propriétés (JMSX) du message

boolean propertyExists(String)

teste l'existence d'une propriété

■ Valeur des propriétés

- Type : boolean, byte, short, int, long, float, double, String
 - méthodes void setIntProperty(String,int), void setBooleanProperty(String,boolean), ...
 - méthodes int getIntProperty(String), boolean getBooleanProperty(String), ...
 - L'appel à getXXXProperty() retourne null si la propriété n'existe pas
 - Les propriétés d'un message reçus sont en lecture seule
sinon l'exception MessageNotWriteableException est levée
- Type Objet correspondant : Boolean,Byte,Short,Int,Long,Float,Double,String
 - méthode void setObjectProperty(String,Object)
 - méthode Object getObjectProperty(String)
- Réinitialisation des propriétés : void clearProperties()

Le modèle de Messages JMS

Le corps (body) du message

- Sous interfaces de `javax.jms.Message`

`javax.jms.TextMessage`

- contient un `String` qui peut être un document XML, un message SOAP...
 - `String getText()/ setText(String)`

`javax.jms.MapMessage`

- contient un ensemble de paires name-value
 - `int getInt(String name) / setInt(String name, int value), ...`
 - `Object getObject(String)/ setObject(String, Object), Enumeration getMapNames()`

`javax.jms.ObjectMessage`

- contient un objet Java sérialisé : `Object getObject()/ setObject(Object)`

`javax.jms.BytesMessage`

- contient un tableau de bytes (non interprétés)
 - `int readInt()/writeInt(int), ..., int readBytes(Byte[])/writeBytes(Byte[]),`
 - `Object readObject()/ writeObject(Object), reset()`

`javax.jms.StreamMessage`

- contient un flot de données (*Java primitives*) qui s'écrivent et se lisent séquentiellement

`int readInt()/writeInt(int) String readString()/writeString(String)`

Remarque: Message SOAP sur JMS

- *In a recent "Strategic Planning" research note, Gartner issued a prediction that "by 2004, more than 25 percent of all standard Web services traffic will be asynchronous...." and "by 2006, more than 40 percent of the standard Web services traffic will be asynchronous."*

Le modèle de Messages JMS

Le corps (body) du message

- Méthodes
 - la méthode `clearBody()` réinitialise le corps

- Remarque
 - le corps d'un message reçu est en lecture seule

Le modèle de Messages JMS

Le corps (body) du message

```
String textstr = "<?xml version='1.0'?><!DOCTYPE person SYSTEM 'person.dtd'"
    + "<person><firstname>Joe</firstname><lastname>Smith</lastname>"
    + "<salary value='10000.0'><age value='38'></person>";
```

// l'API javax.xml est préférable pour construire le document XML

```
TextMessage textmsg = session.createTextMessage();
```

```
textmsg.setText(textstr);
```

```
MapMessage mapmsg = session.createMapMessage();
```

```
mapmsg.setString("Firstname", "Joe");    mapmsg.setString("Lastname", "Smith");
```

```
mapmsg.setDouble("Salary", 10000.0);    mapmsg.setLong("Age", 38);
```

```
Person object = new Person("Joe", "Smith", 37); object.setAge(38); object.setSalary(10000.0);
```

```
ObjectMessage objectmsg = session.createObjectMessage();
```

```
objectmsg.setObject(object); // Person doit implémenter java.io.Serializable
```

```
Byte[] bytearray = { 'J','o','e',' ','S','m','i','t','h'};
```

```
BytesMessage bytesmsg = session.createBytesMessage();
```

```
bytesmsg.writeBytes(bytearray); bytesmsg.writeInt(38); bytesmsg.writeDouble(10000.0);
```

```
StreamMessage streammsg = session.createStreamMessage();
```

```
streammsg.writeString("Joe");
```

```
streammsg.writeString("Smith");
```

```
streammsg.writeLong(38);
```

```
streammsg.writeFloat(10000.0);
```

Le modèle de Messages JMS

Le corps (body) du message

```
TextMessage textmsg = (TextMessage)receivedmsg;  
String textstr=textmsg.getText(textstr); System.out.println(textstr);  
// le document XML peut être parsé
```

```
MapMessage mapmsg = (MapMessage)receivedmsg;  
String firstname= mapmsg.getString("Firstname");  
String lastname= mapmsg.getString("Lastname");  
long age= mapmsg.getLong("Age"); double salary= mapmsg.getDouble("Salary");  
System.out.println(firstname + " " + lastname + " " + age + " " + salary);
```

```
ObjectMessage objectmsg = (ObjectMessage)receivedmsg;  
Person object = (Person)objectmsg.getObject();  
System.out.println(object.toString());
```

```
BytesMessage bytesmsg = (BytesMessage)receivedmsg;  
Byte[] bytearray ;  
int length=bytesmsg.readBytes(bytearray);  
long age= bytesmsg.readLong(); double salary= bytesmsg.readDouble();
```

```
StreamMessage streammsg = (StreamMessage)receivedmsg;  
String firstname= streammsg.readString(); String lastname= streammsg.readString();  
long age= streammsg.readLong(); double salary= streammsg.readDouble();  
System.out.println(firstname + " " + lastname + " " + age + " " + salary);
```

Le modèle de Messages JMS

La sélection (filtrage) des messages

■ Motivation

- le MessageProducer catégorise les messages à envoyer avec les propriétés
- le MessageConsumer peut filtrer les messages reçus
 - l'expression de sélection est spécifiée à la fabrication du MessageConsumer
 - les messages reçus sont ceux qui vérifient l'expression de sélection

■ Expressions de sélection

- sous-ensemble de SQL-92
 - identifiants, littéraux (Chaînes, Numériques, TRUE/FALSE)
 - connecteurs OR, AND, NOT, ()
 - >, <, >=, <=, <>, =, BETWEEN, LIKE, IN, IS [NOT] NULL

■ Exemple

```
String selector="JMSType='car' AND ( color='blue' OR color='red' ) AND price IS NOT NULL";
```

```
QueueReceiver receiver= session.createReceiver(queue, selector);
```

Exemple : Point à Point

La partie commune

```
class JMSQueueTest {
    static Context messaging;          static QueueConnectionFactory queueConnectionFactory; static Queue queue;
    static QueueConnection queueConnection;      static QueueSession queueSession;
    static QueueSender queueSender;             static QueueReceiver queueReceiver;
    static void setup(String queueConnectionFactoryName, String queueName){
        messaging = new InitialContext();
        queueConnectionFactory = (QueueConnectionFactory)messaging.lookup(queueConnectionFactoryName);
        queue = (Queue) messaging.lookup(queueName);
        queueConnection = queueConnectionFactory.createQueueConnection();
        queueSession = queueConnection.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);
    }
    static void setupSender(String queueConnectionFactoryName, String queueName){
        setup(queueConnectionFactoryName, queueName);
        queueSender = queueSession.createSender(queue);
        queueConnection.start();
    }
    static void setupReceiver(String queueConnectionFactoryName, String queueName){
        setup(queueConnectionFactoryName, queueName);
        queueReceiver = queueSession.createReceiver(queue);
        queueConnection.start();
    }
}}
```

Exemple : Point à Point *Sender*

```
// java JMSQueueSenderTest testServer testQueue Hello 10.0 10
public class JMSQueueSenderTest extend JMSQueueTest {

static void send(String stringValue,double doubleValue, long doubleValue, boolean booleanProperty) {
    MapMessage message = session.createMapMessage();
    message.setObject("stringValue", stringValue);
    message.setDouble("doubleValue", doubleValue);
    message.setLong("longValue", longValue);
    message.setBooleanProperty("exit", booleanProperty);
    queueSender.send(message);
}

public static void main(String args[]) {
    setupSender(argv[0],argv[1]);
    long cpt=Long.parseLong(argv[4]);
    while(--cpt>0) {
        send(argv[2]+cpt,Double.parseDouble(argv[3]).valueDouble()+cpt,cpt, false);
    }
    send(argv[2]+cpt,argv[3]+cpt,cpt, true);
    close();
}}
```

Exemple : Point à Point

Receiver synchrone

```
// java JMSQueueSyncReceiverTest testServer testQueue
public class JMSQueueSyncReceiverTest extend JMSQueueTest {
    static boolean exit=false;
    static void handleMessage(MapMessage message) {
        String stringValue=message.getString("stringValue");           System.out.print(" stringValue="+stringValue);
        double doubleValue=message.getDouble("doubleValue");           System.out.print(" doubleValue="+doubleValue);
        long longValue=((Long)message.getObject("longValue")).longValue(); System.out.println(" longValue="+longValue);
        exit = message.getBooleanProperty("exit");
    }
    static void syncReceive() {
        MapMessage message;
        while(!exit) {
            message= (MapMessage)queueReceiver.receive();
            handleMessage(message);
        }
    }
    public static void main(String args[]) {
        setupReceiver(argv[0],argv[1]);
        syncReceiver();
        close();
    }
}
```

Exemple : Point à Point

Receiver asynchrone

```

// java JMSQueueAsyncReceiverTest testServer testQueue public class MyListener implements
    javax.jms.MessageListener {
    void onMessage(Message message) { JMSTest.handleMessage((MapMessage)message); }
}
public class JMSQueueAsyncReceiverTest extend JMSQueueTest {
    static boolean exit=false;
    static void handleMessage(MapMessage message) {
        String stringValue=message.getString("stringValue");          System.out.print(" stringValue="+stringValue);
        double doubleValue=message.getDouble("doubleValue");          System.out.print(" doubleValue="+doubleValue);
        long longValue=((Long)message.getObject("longValue")).longValue(); System.out.println(" longValue="+longValue);
        exit = message.getBooleanProperty("exit");
    }
    static void asyncReceive() {
        queueReceiver.setMessageListener(new MyListener());
        while(!exit) { /* doSomething */ }
    }
    public static void main(String args[]) {
        setupReceiver(argv[0],argv[1]);
        asyncReceiver();
        close();
    }
}

```

Exemple : Publication Souscription

La partie commune

```
class JMSTopicTest {
    static Context messaging;          static TopicConnectionFactory topicConnectionFactory; static Topic topic;
    static TopicConnection topicConnection;          static TopicSession topicSession;
    static TopicPublisher topicPublisher ;          static TopicSubscriber topicSubscriber ;
    static void setup(String topicConnectionFactoryName, String topicName){
        messaging = new InitialContext();
        topicConnectionFactory = (TopicConnectionFactory)messaging.lookup(topicConnectionFactoryName);
        topic = (Topic) messaging.lookup(topicName);
        topicConnection = topicConnectionFactory.createTopicConnection();
        topicSession = topicConnection.createTopicSession(false,Session.AUTO_ACKNOWLEDGE);
    }
    static void setupPublisher(String topicConnectionFactoryName, String topicName){
        setup(topicConnectionFactoryName, topicName);
        topicPublisher = topicSession.createPublisher(topic);
        topicConnection.start();
    }
    static void setupSubscriber(String topicConnectionFactoryName, String topicName){
        setup(topicConnectionFactoryName, topicName);
        topicSubscriber = topicSession.createSubscriber(topic);
        topicConnection.start();
    }
}}
```

Exemple : Publication Souscription

Publisher

```
// java JMSTopicPublisherTest testServer testTopic Hello 10.0 10
public class JMSTopicPublisherTest extend JMSTopicTest {

static void publish(String stringValue,double doubleValue, long doubleValue, boolean booleanProperty) {
    MapMessage message = session.createMapMessage();
    message.setObject("stringValue", stringValue);
    message.setDouble("doubleValue", doubleValue);
    message.setLong("longValue", longValue);
    message.setBooleanProperty("exit", booleanProperty);
    topicPublisher.publish(message);
}

public static void main(String args[]) {
    setupPublisher(argv[0],argv[1]);
    long cpt=Long.parseLong(argv[4]);
    while(--cpt>0) {
        publish(argv[2]+cpt,Double.parseDouble(argv[3]).valueDouble()+cpt,cpt, false);
    }
    publish(argv[2]+cpt,argv[3]+cpt,cpt, true);
    close();
}}
```

Exemple : Publication Soudscription

Subscriber synchrone

```

// java JMSTopicSyncSubscriberTest testServer testTopic
public class JMSTopicSyncSubscriberTest extend JMSTopicTest {
    static boolean exit=false;
    static void handleMessage(MapMessage message) {
        String stringValue=message.getString("stringValue");           System.out.print(" stringValue="+stringValue);
        double doubleValue=message.getDouble("doubleValue");           System.out.print(" doubleValue="+doubleValue);
        long longValue=((Long)message.getObject("longValue")).longValue(); System.out.println(" longValue="+longValue);
        exit = message.getBooleanProperty("exit");
    }
    static void syncSubscribe() {
        MapMessage message;
        while(!exit) {
            message= (MapMessage)topicSubscriber.subscribe();
            handleMessage(message);
        }
    }
    public static void main(String args[]) {
        setupSubscriber(argv[0],argv[1]);
        syncSubscriber();
        close();
    }
}

```

Exemple : Publication Souscription

Subscriber asynchrone

```

// java JMSTopicAsyncSubscriberTest testServer testTopic
public class MyListener implements javax.jms.MessageListener {
    void onMessage(Message message) { JMSTest.handleMessage((MapMessage)message); }
}
public class JMSTopicAsyncSubscriberTest extend JMSTopicTest {
    static boolean exit=false;
    static void handleMessage(MapMessage message) {
        String stringValue=message.getString("stringValue");           System.out.print(" stringValue="+stringValue);
        double doubleValue=message.getDouble("doubleValue");           System.out.print(" doubleValue="+doubleValue);
        long longValue=((Long)message.getObject("longValue")).longValue(); System.out.println(" longValue="+longValue);
        exit = message.getBooleanProperty("exit");
    }
    static void asyncSubscribe() {
        topicSubscriber.setMessageListener(new MyListener());
        while(!exit) { /* doSomething */ }
    }
    public static void main(String args[]) {
        setupSubscriber(argv[0],argv[1]);
        asyncSubscriber();
        close();
    }
}

```

Le Transactionnel dans JMS

■ Motivation

- inclure la production et consommation de messages d'un client dans la portée d'une transaction distribuée (JTA)
 - une Queue ou un Topic sont considérés comme des ressources XA
 - en cas d'abandon, les messages ne sont pas postés ou retirés
 - le JMS Provider doit garantir la livraison « une et une seule fois »
 - Remarque : un message ne peut pas être posté et retiré dans la même transaction

■ API

JMS Root

ServerSessionPool
 ServerSession
 ConnectionConsumer
 XAConnectionFactory
 XAConnection

PTP Interface

XAQueueConnectionFactory
 XAQueueConnection

Pub/Sub Interface

XATopicConnectionFactory
 XATopicConnection

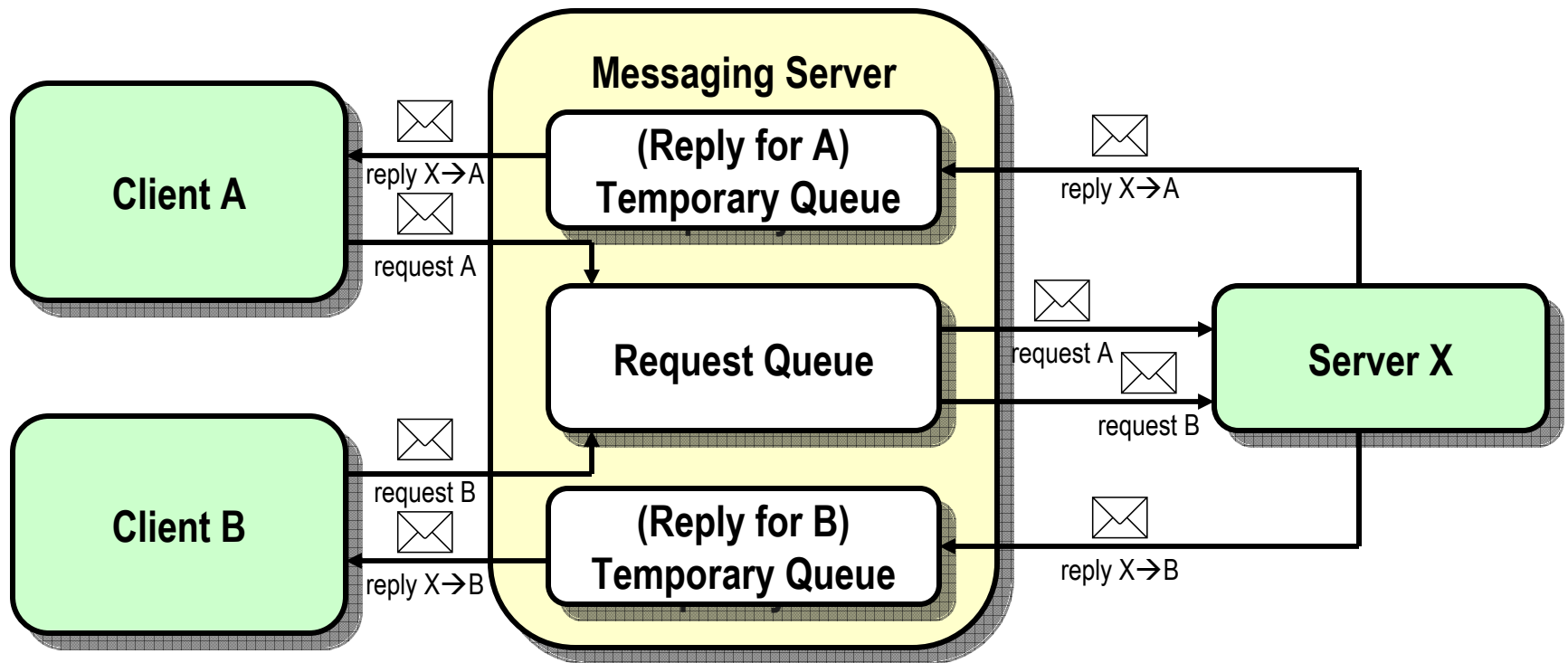
Modèle Requête-Réponse (i)

Request-Reply

- Principe
 - Client-Serveur en mode Asynchrone
 - mais nombreux styles de Requête-Réponse
 - 1- un message de requête donne un message de réponse
 - 2- un message de requête donne un flot de messages de réponse de la part de plusieurs serveurs (*respondents*)
 - ...
- Non explicitement supporter par JMS
 - mais JMS fournit des facilités pour le construire
 - la création de TemporaryQueue et de TemporaryTopic temporaires utilisées pour les réponses (à Destination unique)
 - le champ d 'entête de message JMSReplyTo
 - qui spécifie la Destination pour la réponse
 - le champ d 'entête de message JMSCorrelationID
 - qui peut être utilisé comme référence de la requête originelle

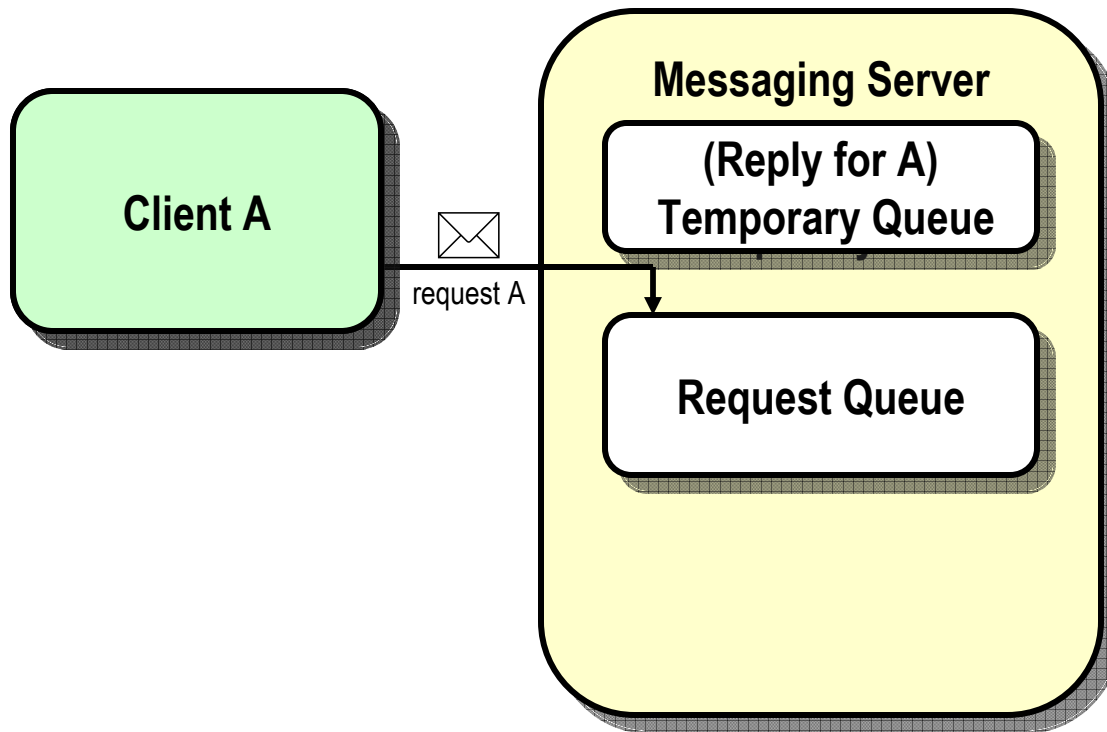
Modèle Requête-Réponse (ii)

- Proposition d'implantation de 1 (avec les Queues de JMS)

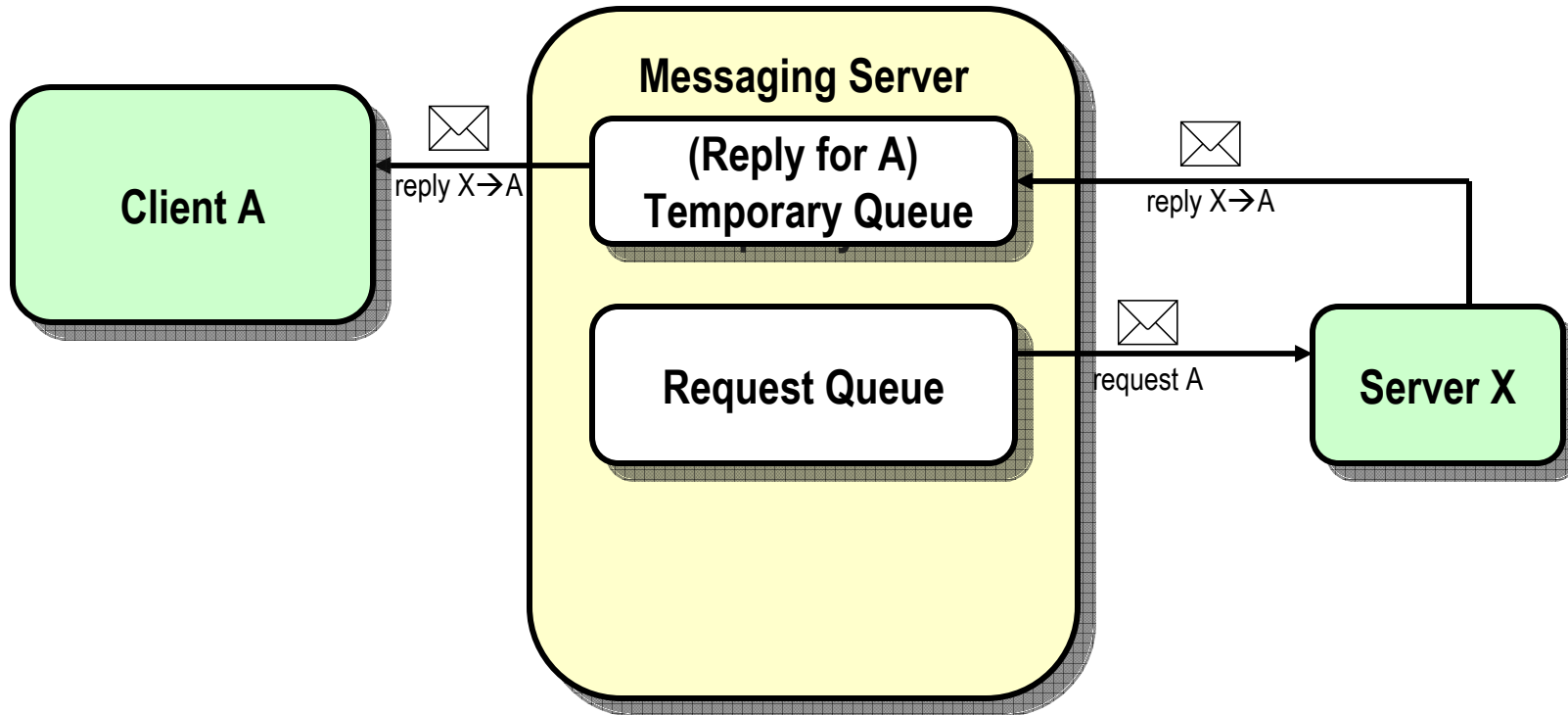


- Question :
 - A quoi peut servir le partage de la Queue par plusieurs serveurs ?

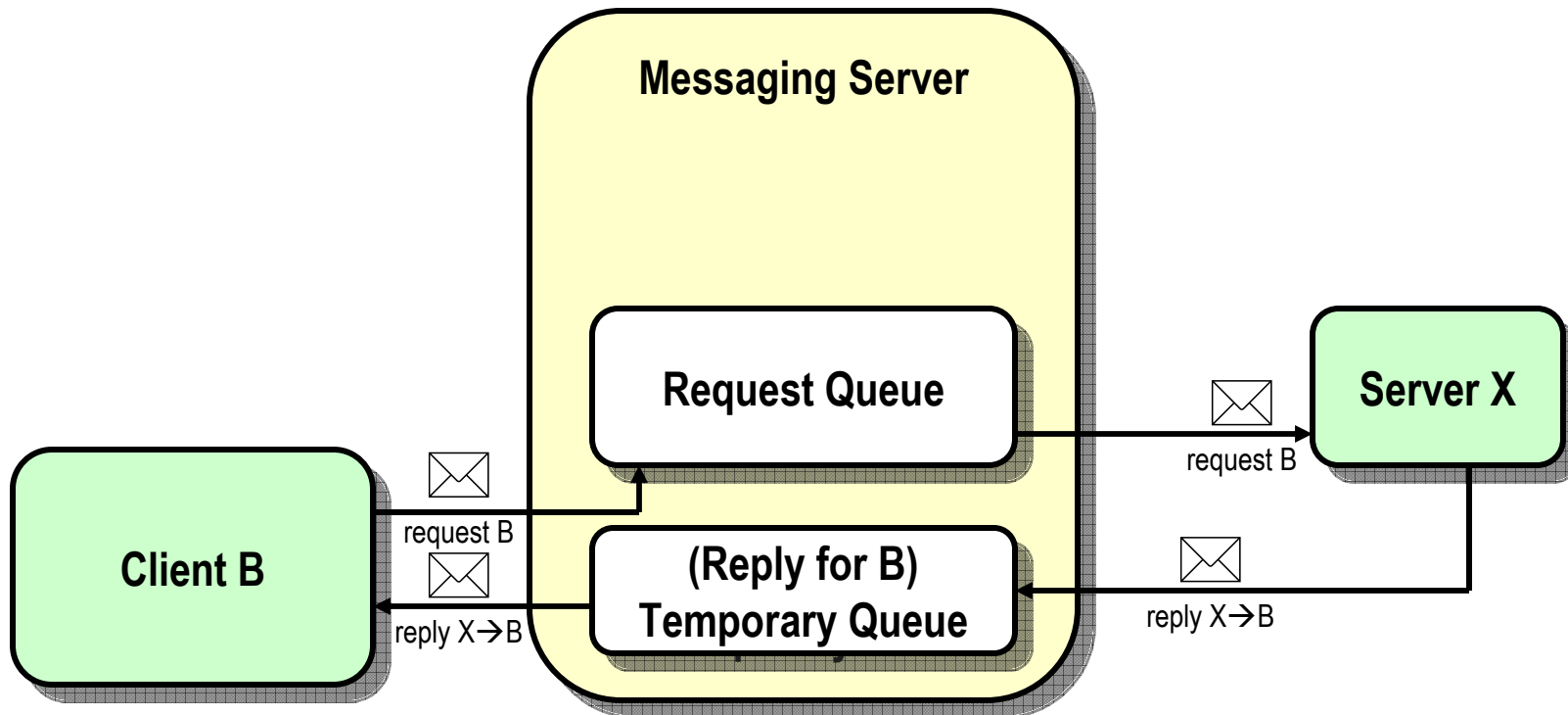
Modèle Requête-Réponse (ii)



Modèle Requête-Réponse (ii)

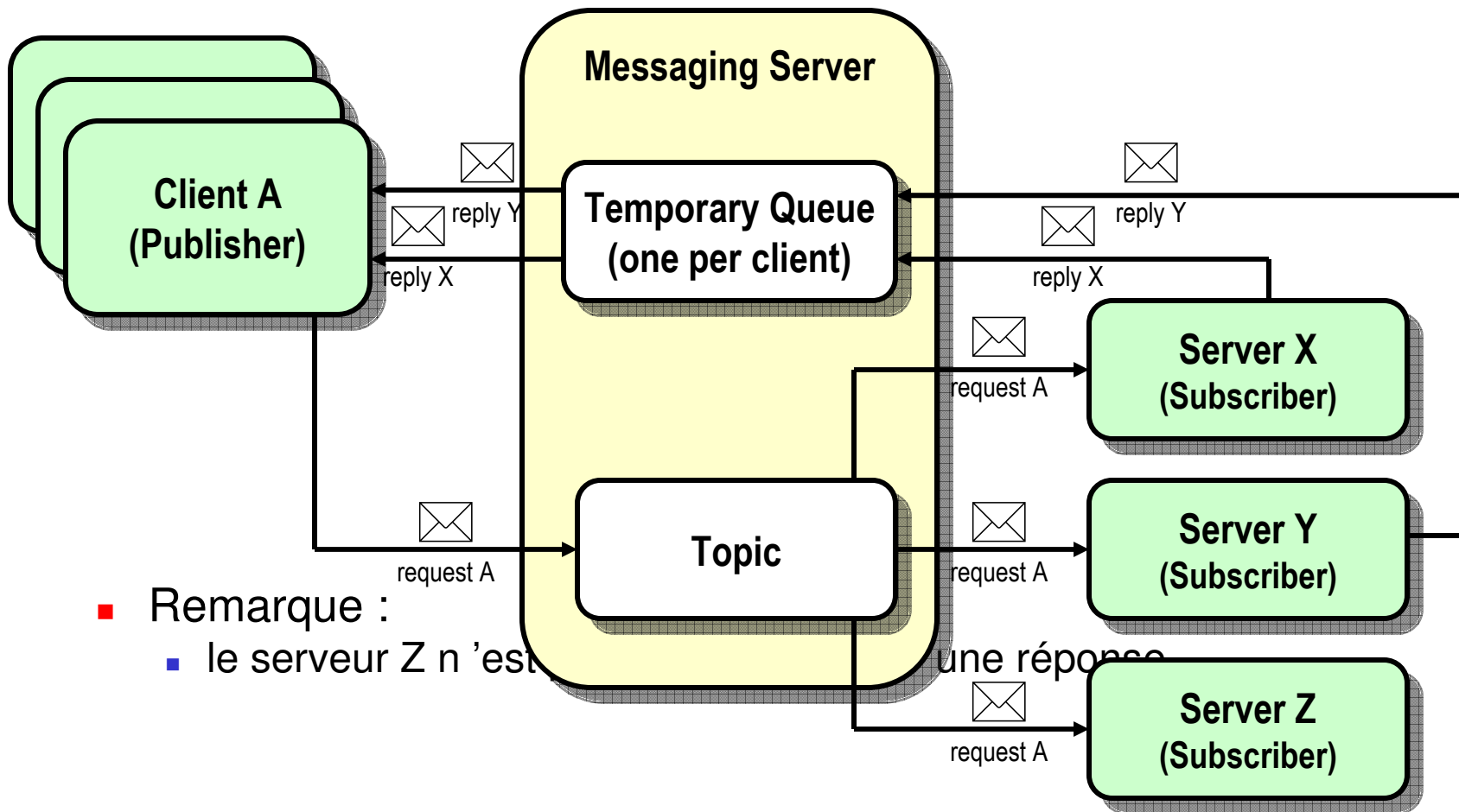


Modèle Requête-Réponse (ii)



Modèle Requête-Réponse (iii)

- Proposition d'implantation de 2 (avec les Topics de JMS)



- Remarque :
 - le serveur Z n'est pas une réponse

Modèle Requête-Réponse (iii)

- QueueRequestor/TopicRequestor
 - classes utilitaires
qui implément de la modèle synchrone de Requête-Réponse
 - la Session ne peut être transactionnelle
- Cycle de vie
 - création avec le constructeur
QueueRequestor(QueueSession session, Queue queue)
TopicRequestor(TopicSession session, Topic topic)
 - envoi d'une requête et attente de la réponse
Message request(Message msg)
 - fermeture pour libérer les ressources void close()
- Remarque
 - la Session ne peut être transactionnelle

Les Destinations Temporaires

- objet unique, qui ne peut être copié
 - fabriqué par la Session dont la durée de vie n 'excède pas la Connection
 - peut être détruit (`delete()`) après usage pour libérer des ressources
 - ne peut être consommée que par la Connection
 - typiquement utilisé pour le modèle Requête-Réponse (propriété *JMSReplyTo*)
- **TemporaryQueue**
 - sous interface de Queue
 - fabriquée par `QueueSession.createTemporaryQueue()`
 - n 'existe que durant une `QueueConnection`
 - **TemporaryQueue**
 - objet unique, sous interface de Queue
 - fabriquée par `QueueSession.createTemporaryQueue()`
 - n 'existe que durant une `QueueConnection`

Ce que JMS n 'adresse pas

- En tant qu 'API Client, JMS n 'adresse pas
 - Administration du produit MOM
 - Équilibrage de charge et Tolérance aux Pannes
 - Dépôt des types de message
 - JMS ne fournit pas un dépôt des formats (types) de message ni un langage de définition
 - Notification d 'erreur et d 'avertissement
 - chaque produit peut envoyer aux clients des messages systèmes
 - JMS ne normalise pas ces messages.
 - Sécurité
 - JMS ne contrôle pas la confidentialité et l 'intégrité des messages
 - Protocole de transport
 - Déploiement
- Extensions proposées par des éditeurs de MOM
 - Topics hiérarchiques, Dead queue, XMLMessage ...

Produits JMS

■ Open Source ou Free

- JORAM (Univ. Grenoble)
 - <http://www.objectweb.org/joram>
- OpenJMS
 - <http://openjms.exolb.org/>
- JDBMS (package JMS)
 - <http://www.jdbms.org>
- XmlBlaster
 - <http://www.xmlBlaster.org/>

■ Vendeurs

- Fiorano/EMS Lite
 - (<http://www.fiorano.com>)
- Allaire Corporation - JRun Server
- BEA Systems, Inc.
- GemStone
- IBM
- Nirvana
- Oracle Corporation
- Orion
- Progress Software
- SAGA Software, Inc.
- SoftWired Inc.
- SpiritSoft, Inc. (formerly Push Technologies Ltd.)
- Sun (Java Message Queue)
- Sunopsis
- SwiftMQ
- Venue Software Corp

JMS dans J2EE

- Partie intégrante de J2EE/EJB
 - JNDI : recherche des objets administrés
 - JTA : XASession
 - EJB : Message-Driven Bean

Bibliographie

Généralités

- P.A. Bernstein, E. Newcomer, «Principles of Transaction Processing for the Systems Professional», Ed. Morgan Kaufmann, 1997, ISBN 1-55860-415-4.
 - *la bible des Moniteurs Transactionnels, il décrit aussi le Messages Queueing (chapitres 4 et 5)*
- Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, Anne-Marie Kermarrec, The many faces of publish/subscribe, ACM Computing Surveys, Vol 35, No. 2, June 2003, pp. 114–131.
- Dave Chappell, Enterprise Service Bus, Pub O'Reilly, June 2004, ISBN 0-596-00675-6, 274 pages

Bibliographie

orienté Programmation

- Alan Dickman, « Designing Applications With Msmq : Message Queuing for Developers », (August 1998), Addison-Wesley Pub Co; ISBN: 0201325810
- Neil Crane, « MSMQ From Scratch », 368 pages (December 7, 1999), Que Education & Training; ISBN: 0789721279
- Rhys Lewis, « Advanced Messaging Applications with MSMQ and MQSeries », 1 edition (December 17, 1999), Que Education & Training; ISBN: 078972023X
- Nayan Ruparelia, « MQ Series Messaging », 400 pages (December 2000) Ed Manning Publications Company; ISBN: 1884777988
- Alex Homer, David Sussman, « Professional MTS and MSMQ Programming with VB and ASP », Ed Wrox Press Inc, 512 pages (June 1998), ISBN: 1861001460;
 - *la bible des Moniteurs Transactionnels, il décrit aussi le Messages Queueing*
- Scott Grant, Michael P. Kovacs, Meeraj Kunnumpurath, Silvano Maffeis, K. Scott Morrison, Gopalan Suresh Raj, Paul Giotta, « Professional JMS », March 2001, Wrox Press Inc; ISBN: 1861004931
- Richard Monson-Haefel & David Chappell , Java Message Service, Oreilly, December 2000, ISBN 0-596-00068-5

Bibliographie

Autres

- Les spécifications JMS
 - <http://java.sun.com/products/jms>
- Site du MOMA (MOM Association)
 - <http://www.moma-inc.org>
- Tutorial
 - voir l'exemple des spécifications
 - « Stock Trader » de Gopalan Suresh Raj
 - http://www.execpc.com/~gopalan/java/java_tutorial.html
- Benchmark
 - <http://www.sys-con.com/java/articleprint.cfm?id=639>