Indirect Communication

Definition

- Communication between entities in a distributed system through an intermediary with no direct coupling between the sender and the receiver(s)

	Time-coupled	*Time-uncoupled*
Space coupling	*Properties*: Communication directed towards a given receiver or receivers; receiver(s) must exist at that moment in time *Examples*: Message passing, remote invocation (see Chapters 4 and 5)	*Properties*: Communication directed towards a given receiver or receivers; sender(s) and receiver(s) can have independent lifetimes *Examples*: See Exercise 15.3
Space uncoupling	*Properties*: Sender does not need to know the identity of the receiver(s); receiver(s) must exist at that moment in time *Examples*: IP multicast (see Chapter 4)	*Properties*: Sender does not need to know the identity of the receiver(s); sender(s) and receiver(s) can have independent lifetimes *Examples*: Most indirect communication paradigms covered in this chapter

Group Communication

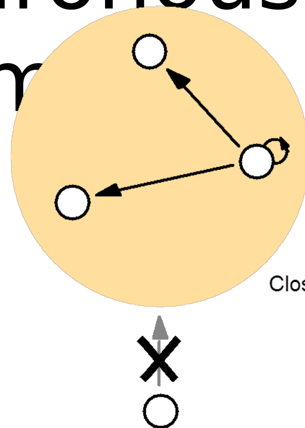
- A message is sent to a group and then delivered to all members of the group
- Applications
  - Reliable dissemination of informations
  - Support for collaborative applications
  - Support for fault tolerant strategies
  - Support for system monitoring and management

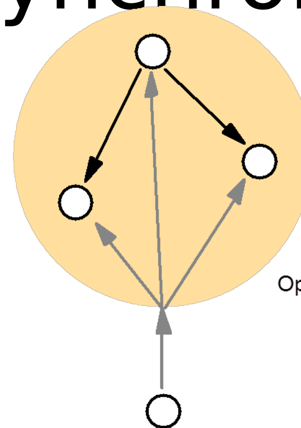
Programming model

- Concepts
  - Group
  - Group membership
  - Join/leave operations
- Multicast communication
  - One operation to send many messages
  - Allows different kind of optimisation
  - Allows to provide guarantees (delivery, ordering)

Kind of group services

- Process vs Object groups
- Open vs Closed Groups
- Overlapping vs non-overlapping groups
- Synchronous vs asynchronous system



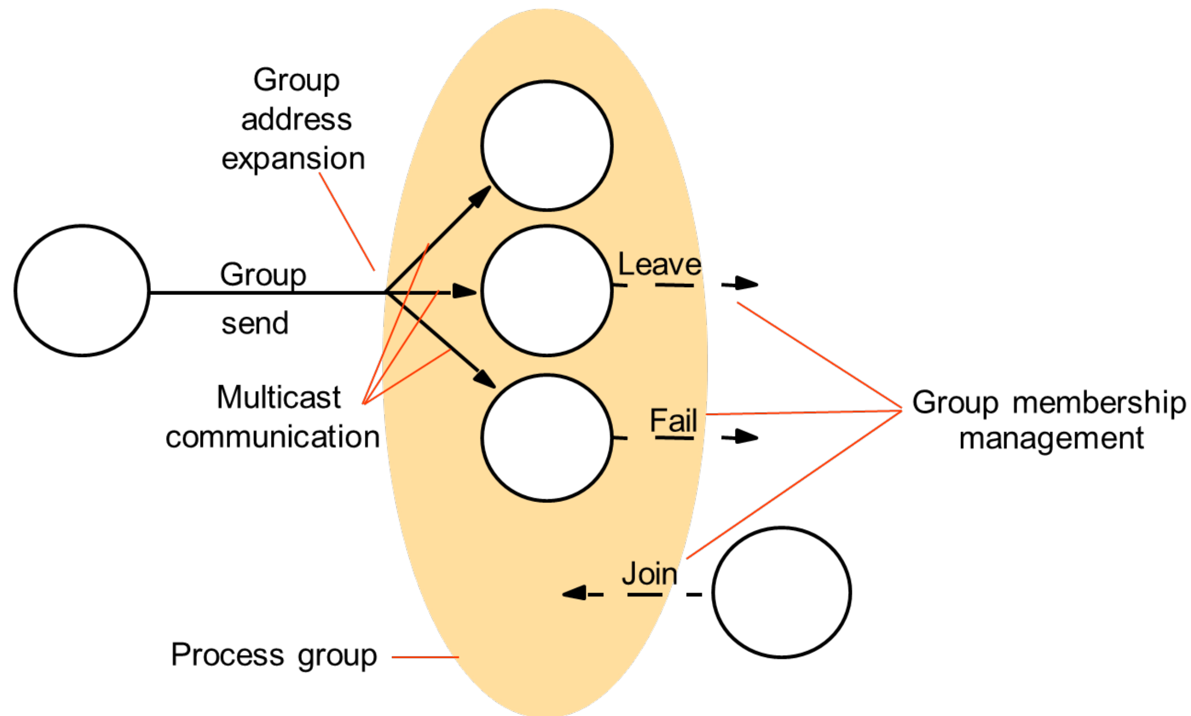


Implementation issues

- Reliability
  - Integrity (same message sent and received)
  - Validity (It is eventually delivered)
  - Agreement (If it is delivered to one process it is delivered to all)
- Ordered Multicast
  - FIFO ordering
  - Causal ordering
  - Total ordering

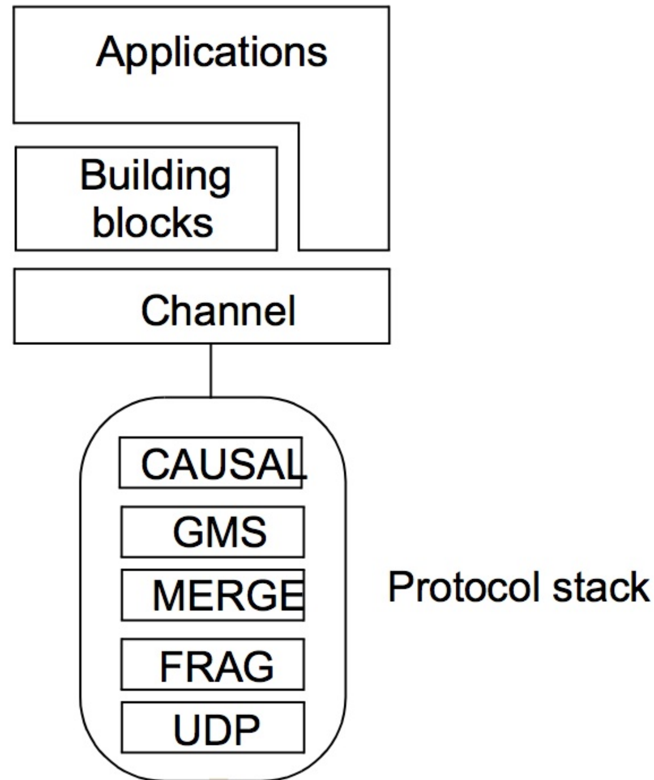
Implementation issues

- Group membership management



Example : Jgroup toolkit

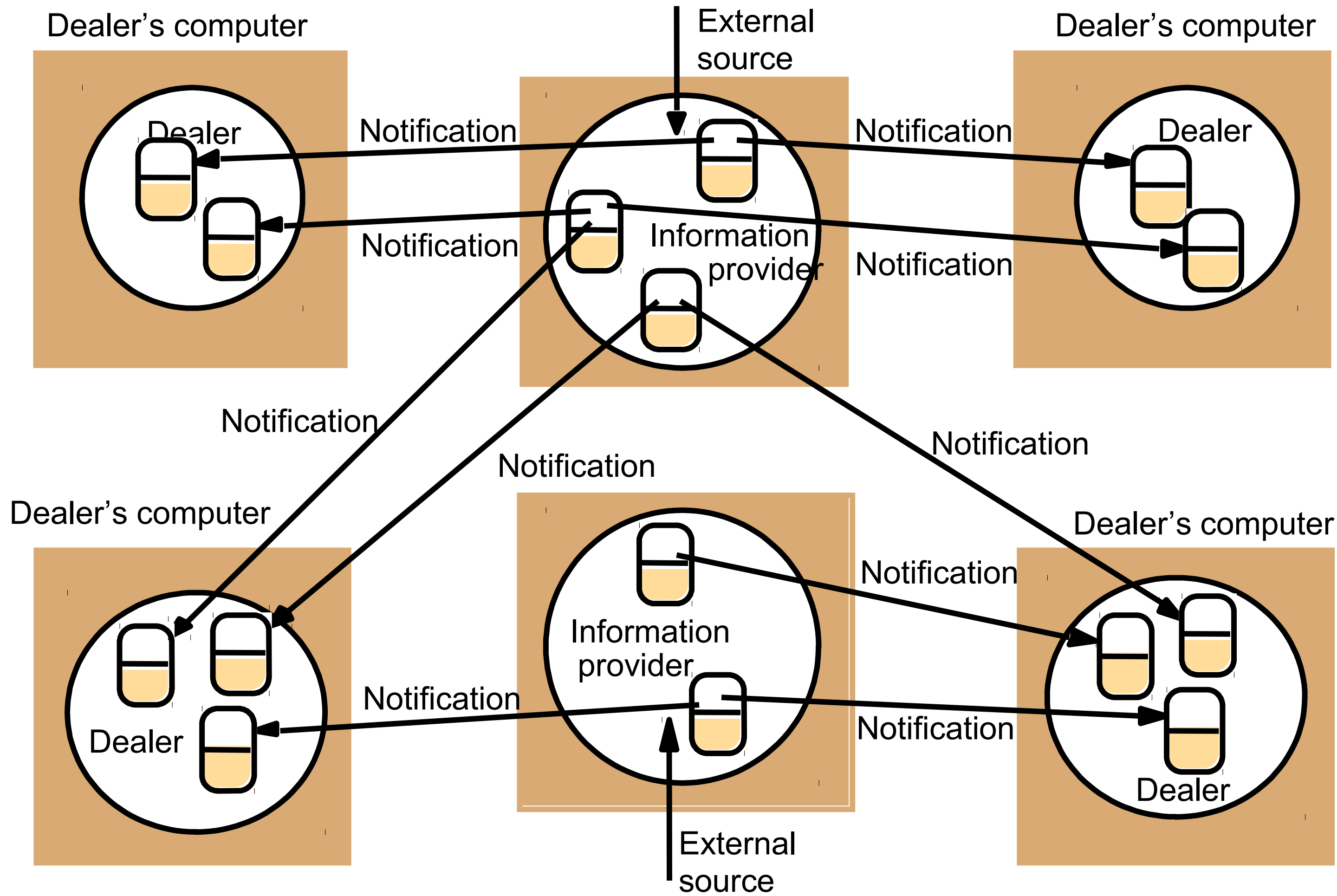
- Toolkit for reliable communication in Java



Publish-Subscribe Systems

- Distributed event based system
  - A publisher publish events to an event service
  - Subscribers express interest in events through subscription
  - The system match subscriptions to events and ensure the delivery
- Application
  - Financial information systems
  - Cooperative work
  - Ubiquitous computing
  - Monitoring

Example : Dealing Room System

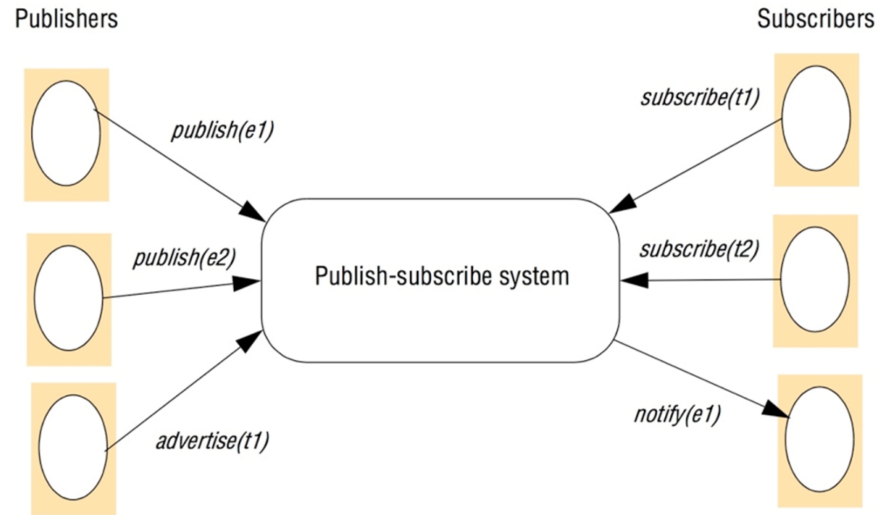


Characteristics

- Heterogeneity
- Asynchronicity
- Different delivery guarantees
  - Depends on the application requirements

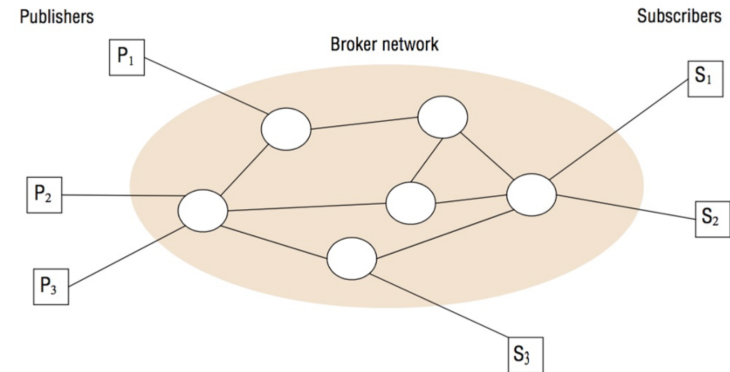
Programming Model

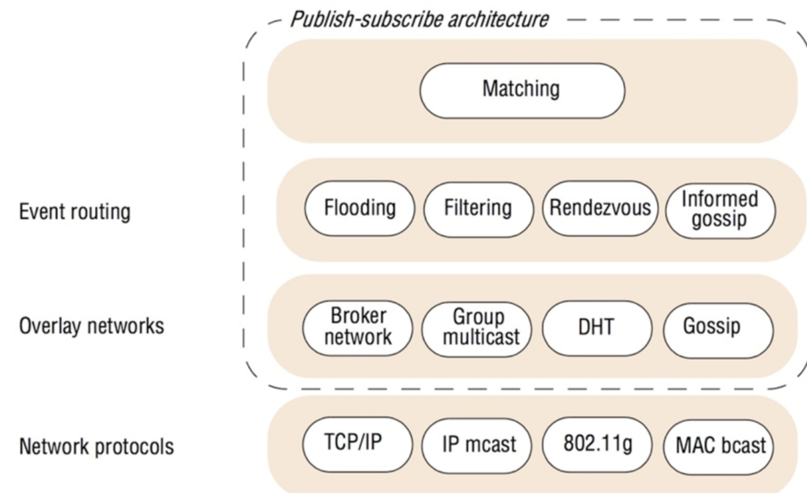
- Operations
  - Publish(e)
  - Subscribe(filter)
  - Unsubscribe(filter)
  - Notify(e)
  - *Advertise*(f)
- Schemes
  - Channel Based
  - Topic Based
  - Content Based
  - Type Based



Implementation issues

- Centralized vs Distributed implementations
  - Centralized = single point of failure + performance
  - Distributed = survive node failure
- System Architecture





Publish Subscribe Systems

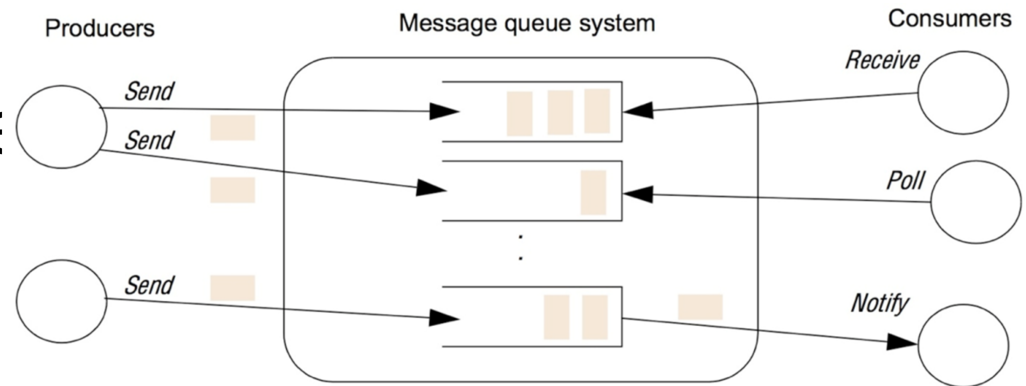
System (and further reading)	*Subscription model*	*Distribution model*	*Event routing*
CORBA Event Service (Chapter 8)	Channel-based	Centralized	-
TIB Rendezvouz [Oki *et al.* 1993]	Topic-based	Distributed	Ffiltering
Scribe [Castro *et al.* 2002b]	Topic-based	Peer-to-peer (DHT)	Rendezvous
TERA [Baldoni *et al.* 2007]	Topic-based	Peer-to-peer	Informed gossip
Siena [Carzaniga *et al.* 2001]	Content-based	Distributed	Filtering
Gryphon [www.research.ibm.com]	Content-based	Distributed	Filtering
Hermes [Pietzuch and Bacon 2002]	Topic- and content-based	Distributed	Rendezvous and filtering
MEDYM [Cao and Singh 2005]	Content-based	Distributed	Flooding
Meghdoot [Gupta *et al.* 2004]	Content-based	Peer-to-peer	Rendezvous
Structure-less CBR [Baldoni *et al.* 2005]	Content-based	Peer-to-peer	Informed gossip

Message queues

- Message Oriented Middleware
- Used for Enterprise Application Integration (EAI)

Programming model

- A producer sends a message to a queue
- Consumer receives messages from queue(s)
- Style of receive
  - Blocking receive
  - Non Blocking receive
  - Notify

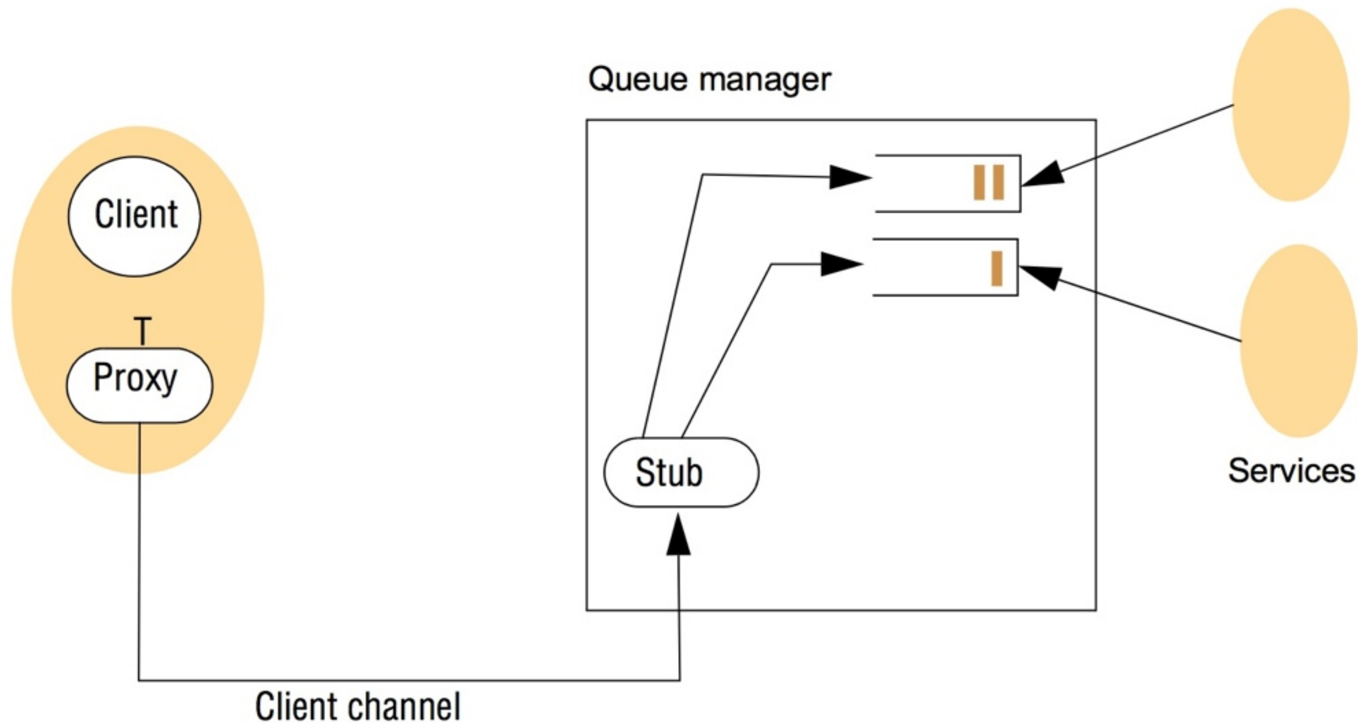


Programming model

- Message are persistent
- Reliable delivery : a message sent will eventually be delivered
  - Validity
  - Integrity
- Can be transactional
- Can be secure

Implementation issue

- Centralized vs Distributed



Message Driven Bean

- Based on JMS, the Java Messaging Service
- How to use JMS
- Message Driven Beans

Alternative to Direct communication

- The client is blocked
- Time and location coupling
- Fault management
- One client talk to one server

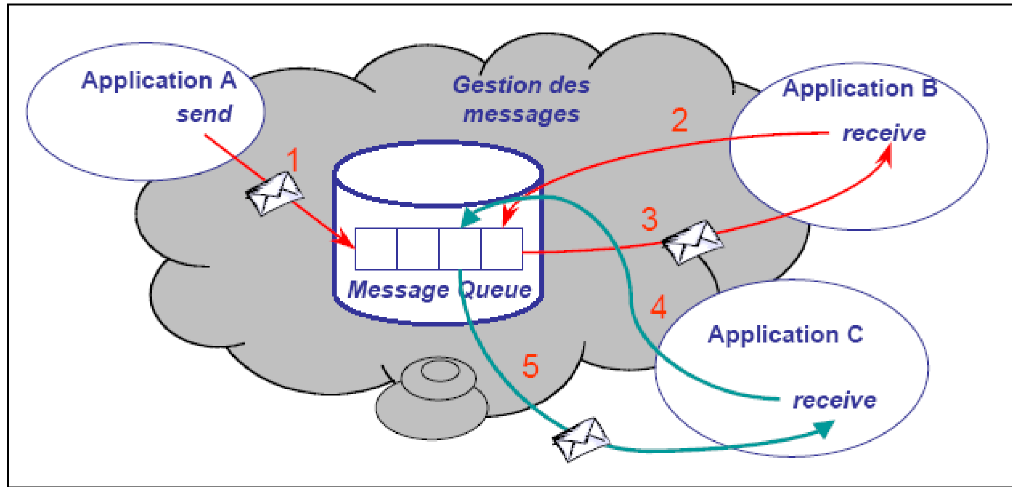
Message oriented approach

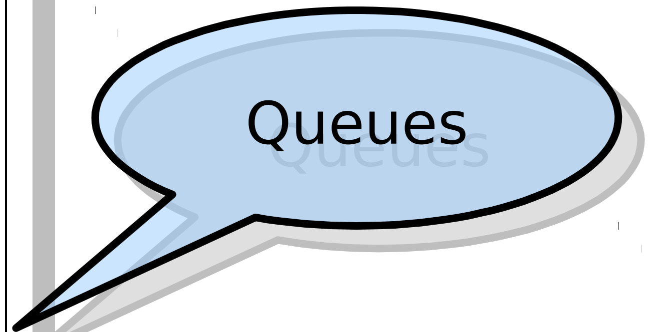
- Non blocking request
- Low Coupling
- Safety
- Many sender/receiver
- Lower performance (indirection)

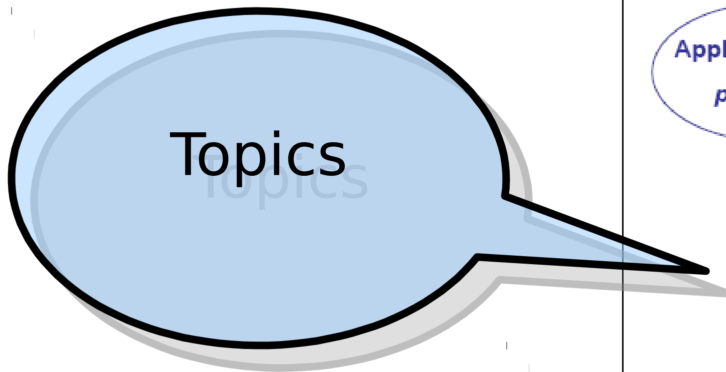
JMS

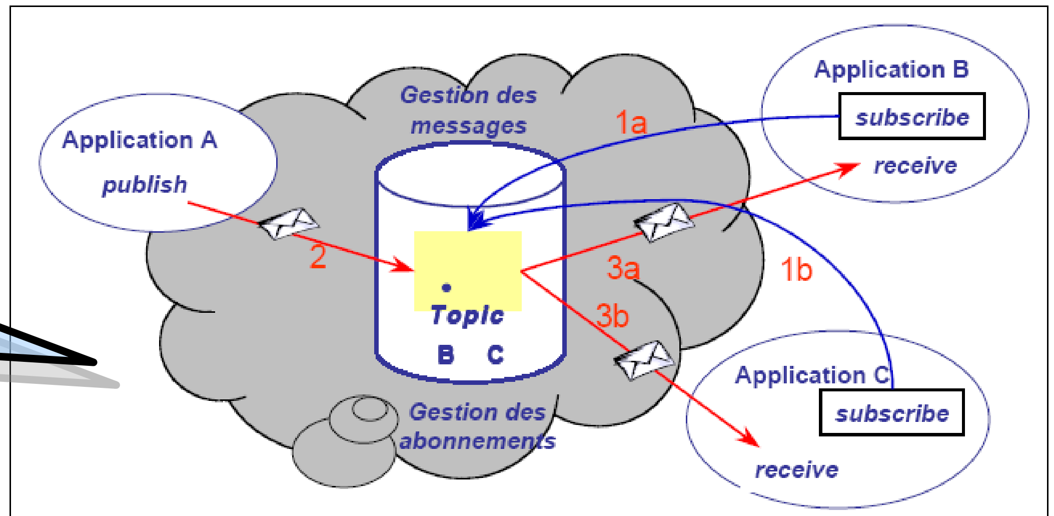
- MOM specification in java
  - JMS Provider : implementation
  - JMS Client : produces and consumes messages.
  - JMS Message : object that contain the message payload

Communication domain



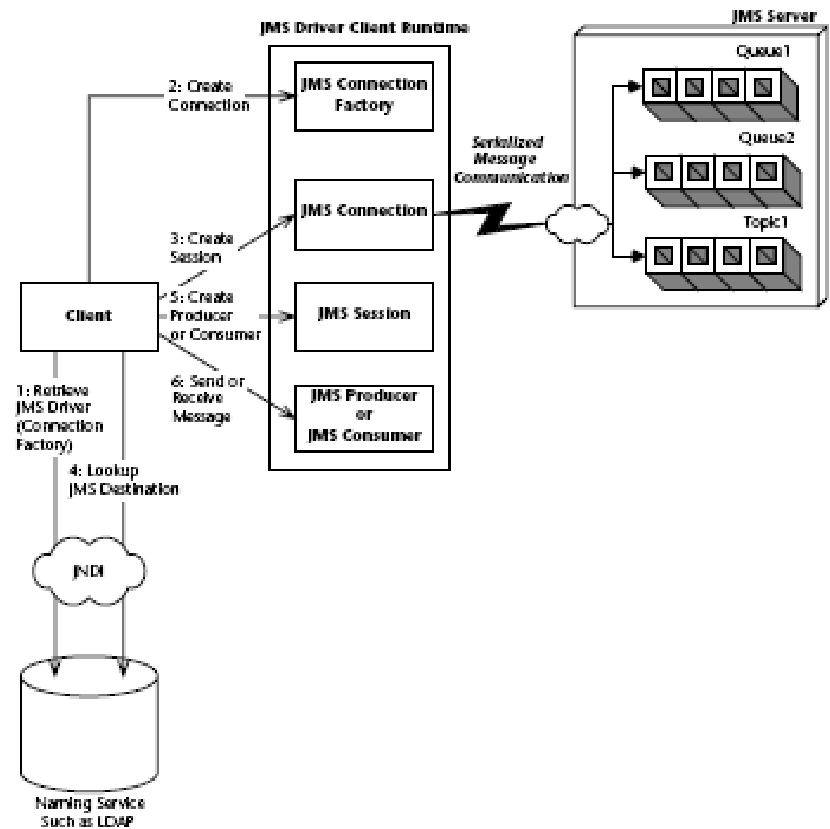






JMS Protocol

- Find the Connection Factory
- Create a Connection
- Create a Session
- Find the destination
- Create a sender or receiver
- Send or Receive Message



Delivery guarantee

- A message sent will eventually be delivered
- A message is sent
- It is consumed
- If the MOM does not receive a ack it is put back in the queue
- Alternative
  - Certified message delivery
  - Store and Forward

Simple example

```
public static void main(String[] args) throws NamingException, JMSException {
    System.setProperty("java.naming.factory.initial",
                "com.sun.enterprise.naming.SerialInitContextFactory");
    System.setProperty("java.naming.factory.url.pkgs",
                "com.sun.enterprise.naming");
    System.setProperty("java.naming.factory.state",
                "com.sun.corba.ee.impl.presentation.rmi.JNDIStateFactoryImpl");
    System.setProperty("java.naming.provider.url",
                "localhost");
    InitialContext ctx=new InitialContext();
```

Several steps

- Lookup to get a ref on the QueueConnectionFactory
- Connection creation
- Session creation
- Get a reference on the queue (Remote)

```
QueueConnectionFactory qcf=(QueueConnectionFactory) ctx.lookup("jms/queueConnFactory");
QueueConnection qc=(QueueConnection)qcf.createConnection();
QueueSession session=qc.createQueueSession(false,Session.AUTO_ACKNOWLEDGE );
Queue q=(Queue)ctx.lookup("jms/myQueue");
```

Receiving messages

```
QueueReceiver receiver=session.createReceiver(q);
qc.start();
while (true) {
    TextMessage text=(TextMessage)receiver.receive(0);
    System.out.println(text.getText());
    if (text.getText().equals("end")) break;
}
receiver.close();
session.close();
qc.stop();
qc.close();
```

Easy

Sending messages

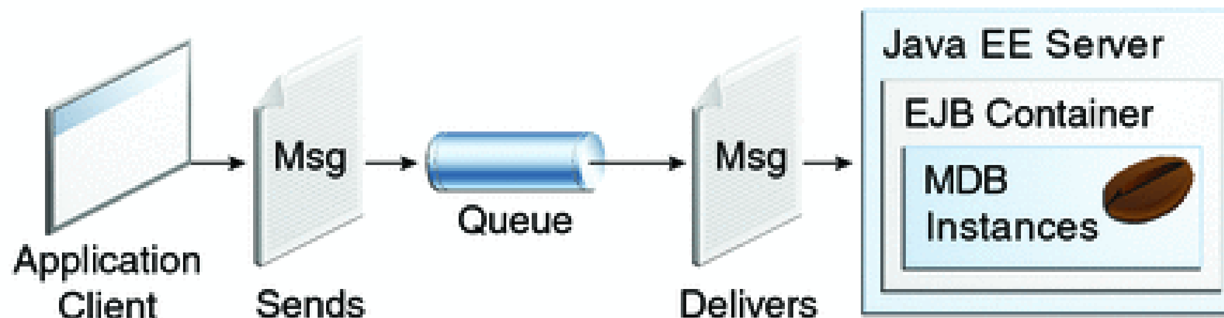
```
QueueSender sender=session.createSender(q);
for (int i=0;i<4;i++) {
    TextMessage txt=session.createTextMessage("message "+i);
    sender.send(txt);
}
TextMessage fin=session.createTextMessage("end");
sender.close();
```

Message Driven Beans

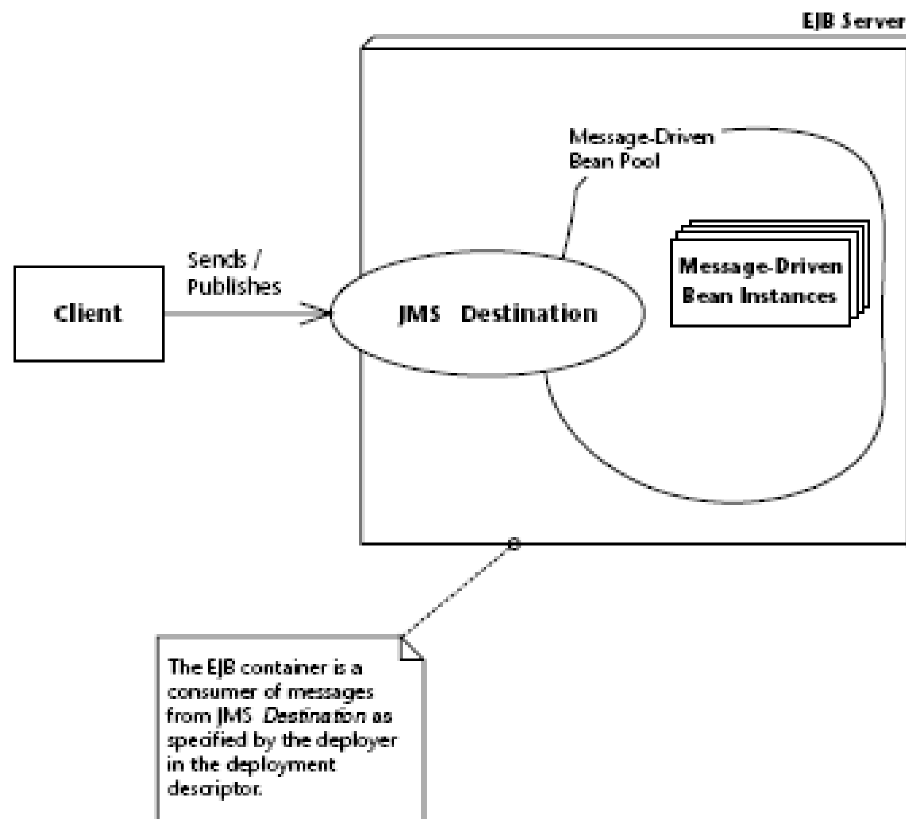
- Without MDB
  - Lot of boiler plate code
  - Life cycle management
  - Context management
- With a MDB
  - The container do the job
  - threading

MDB

- EJB without an interface
- One method : onMessage (Listener)
- No return message
- No exception
- Stateless
- Single threaded



Client = JMS Client



Example

```java
public class ReceiveBean implements MessageListener {

    /** Creates a new instance of ReceiveBean */
    public ReceiveBean() {
    }

    public void onMessage(Message message) {
        TextMessage txt =(TextMessage)message;
        try {
            System.out.println(txt.getText());
        } catch (JMSException ex) {
            ex.printStackTrace();
        }
    }
```

Advanced configuration

```
@MessageDriven(mappedName = "jms/myQueue", activationConfig =  {
        @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "messageSelector", propertyValue = "JMSType='log'"),
        @ActivationConfigProperty(propertyName = "subscriptionDurability", propertyValue = "NonDurable")
    })
public class ReceiveBean implements MessageListener {
```

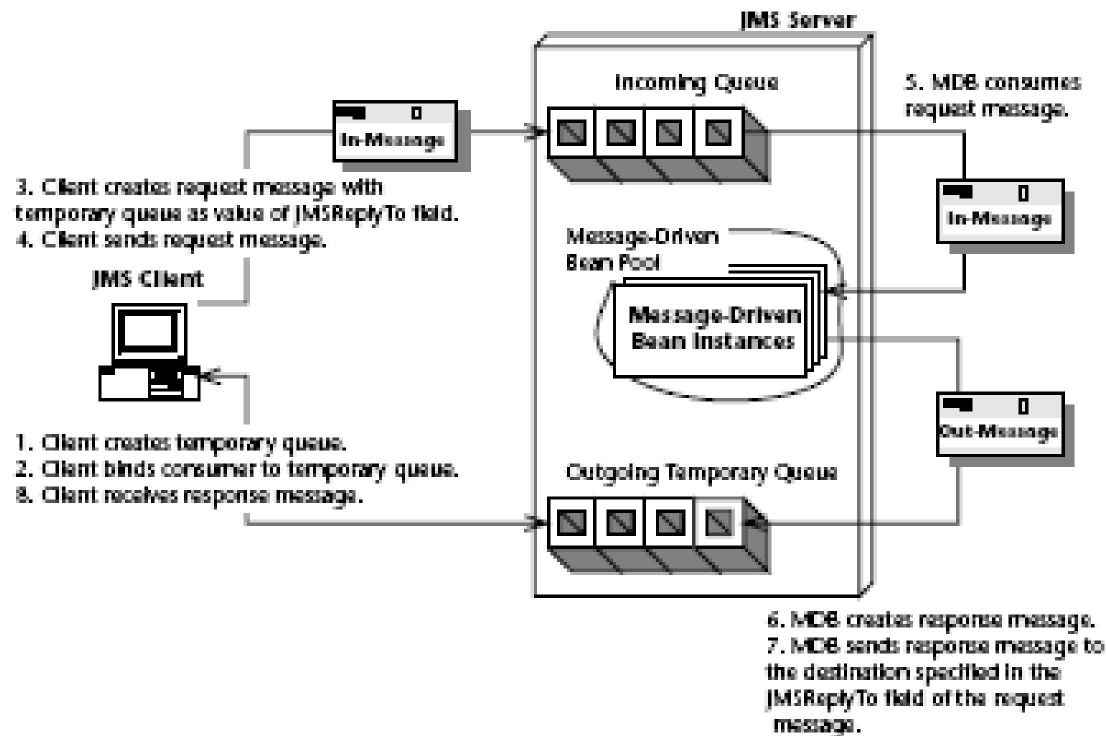
Advanced concept

- Sender and receiver transaction context are different
- Security context is not transmitted
- Load Balancing is easier

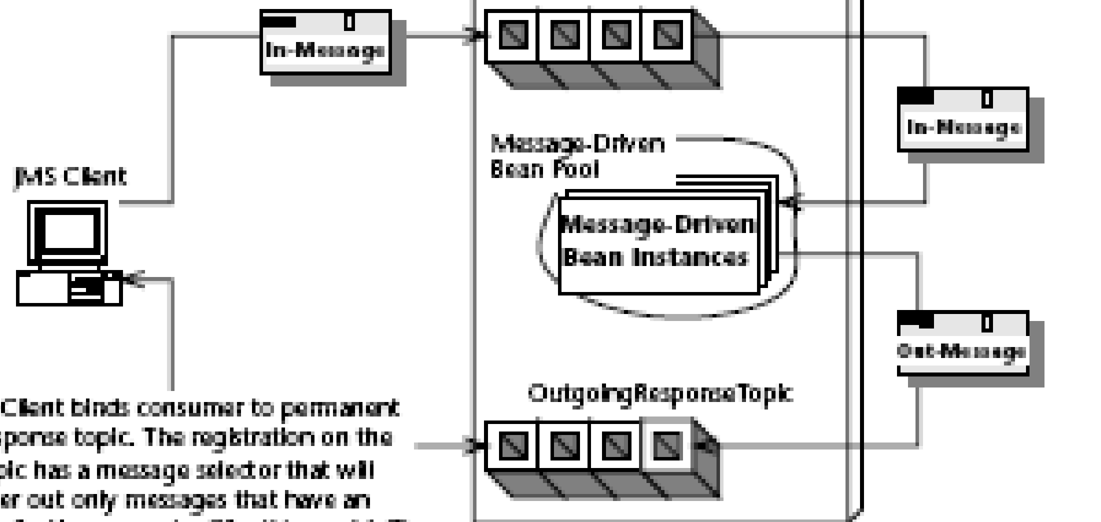
Issues

- No guarantee on message order delivery
- Poison messages
  - Manual management by the admin

Temporary queues



Generic Topic

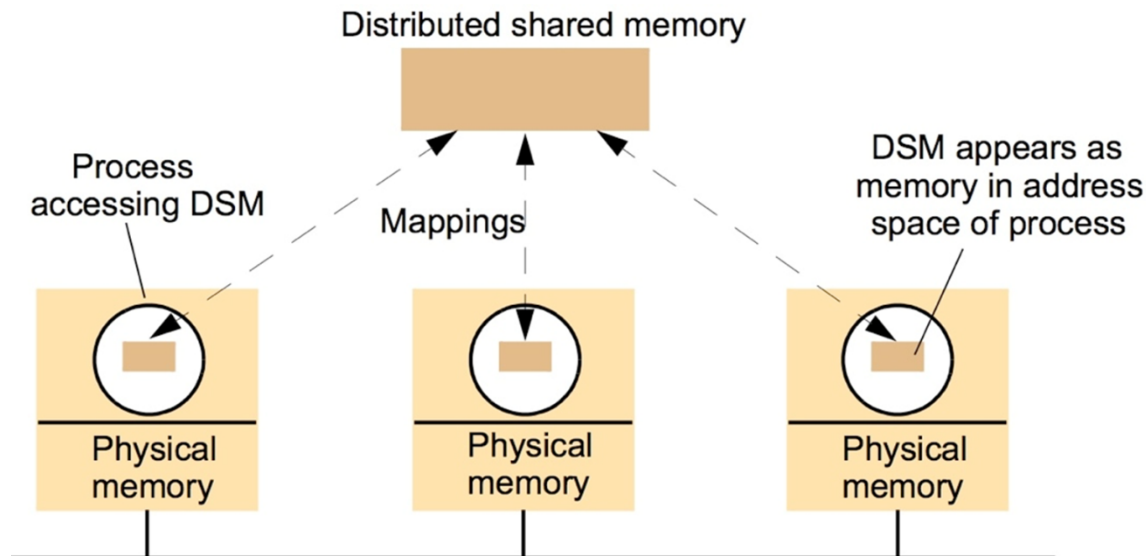


Sur les MDB

- Still primitive solution
- Very good for
  - Fault tolerance
  - Load balancing

Shared Memory

- Distributed shared memory
  - Developed for parallel computing
- Sharing data between computer that do not share physical memory



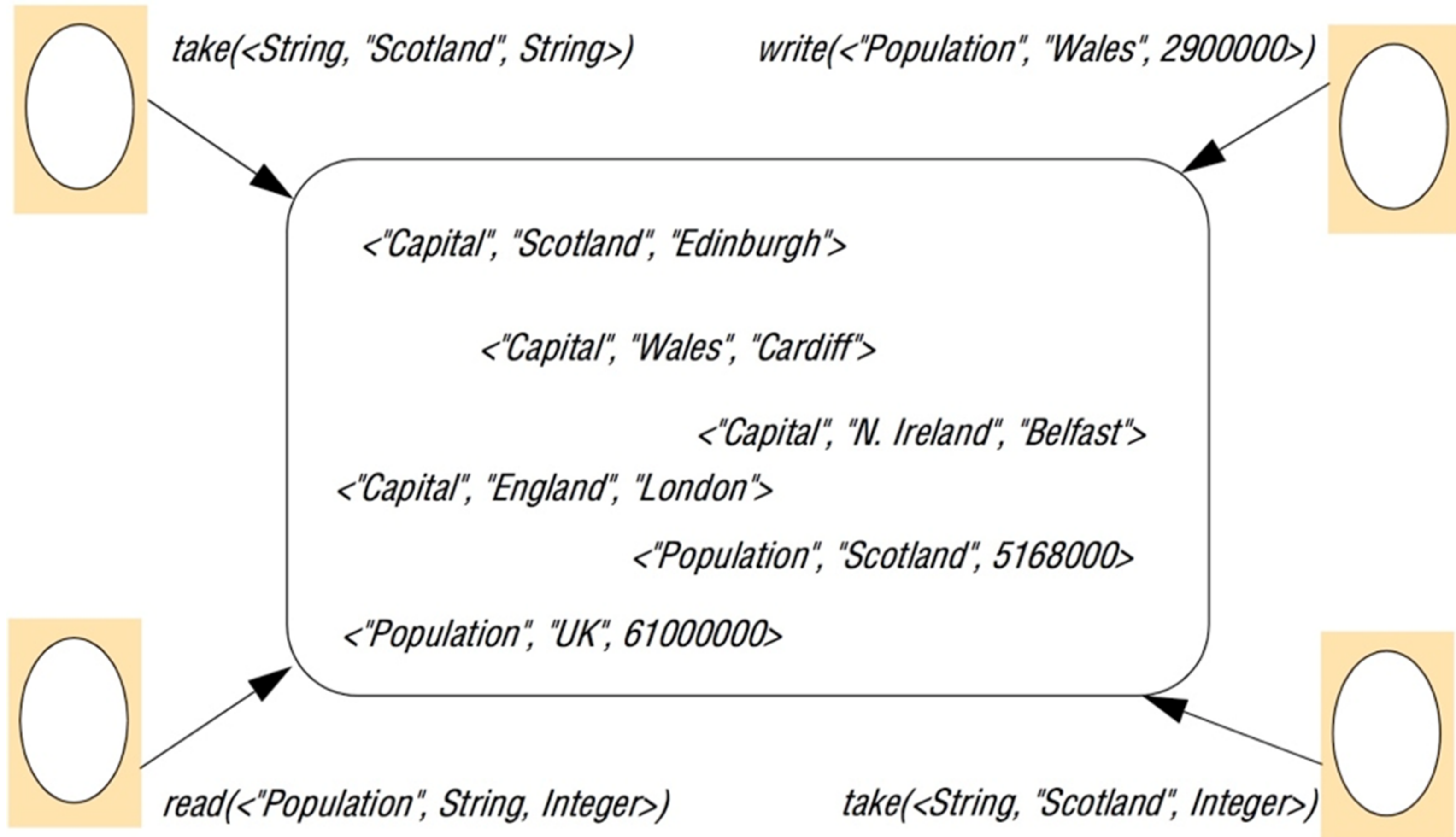
Tuple Space Communication

- Process communicate by placing tuple in a tuple space
  - Linda, JavaSpaces, TSpaces

Programming model

- A tuple is a sequence of typed data field
- A tuple space contains tuples
- Operations
  - Write (add a tuple)
  - Read
  - Take (remove the tuple)
- Access to tuple is associative
- Tuples are immutable

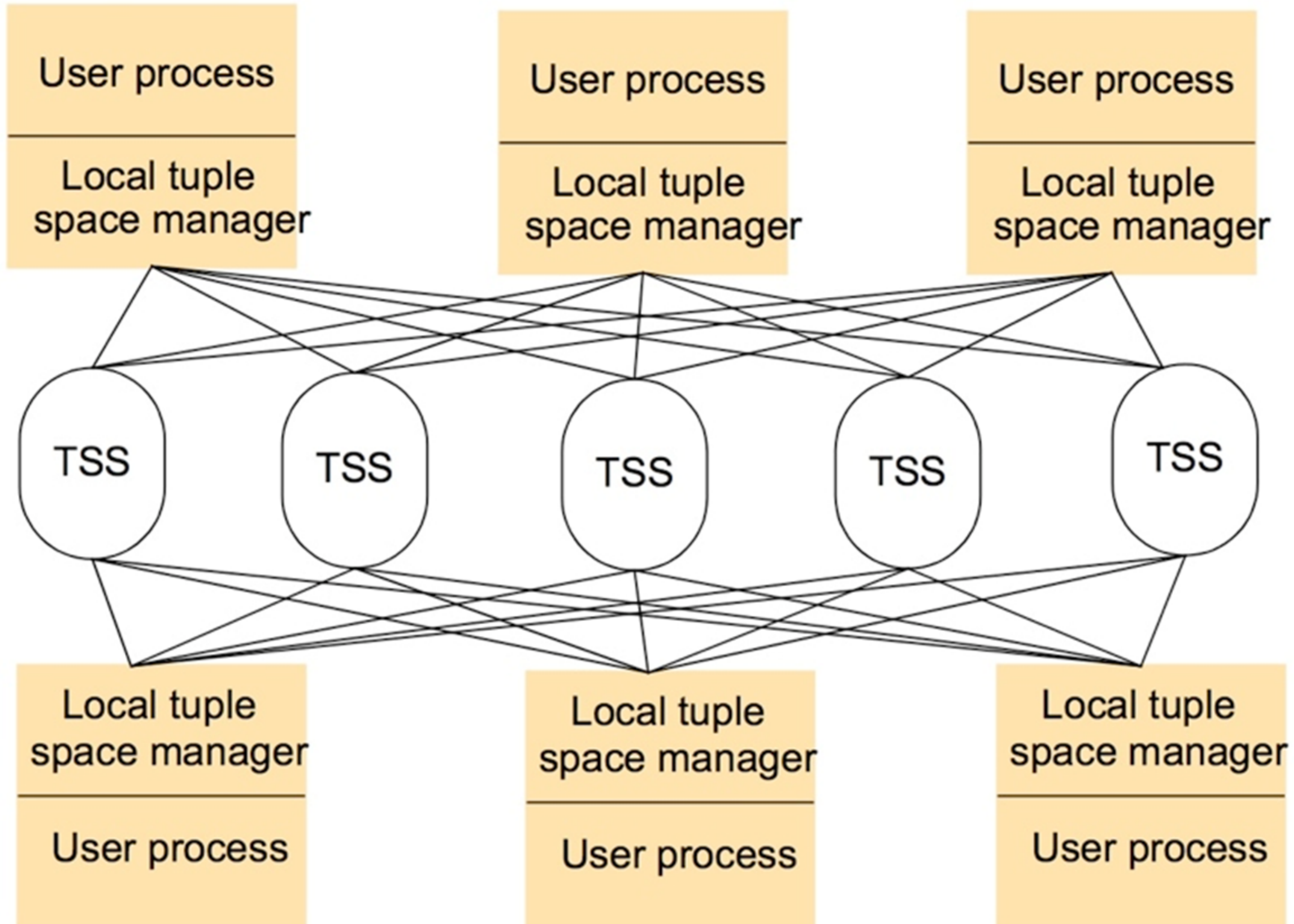
Example



Implementation issues

- Centralized : Single point of failure/Do not scale
- Distributed : replication using
  - A totally ordered multicast algorithm
  - Or partitioning of spaces (Xu and Liskov)

York Linda Kernel



Summary

- Indirect communication lead to interesting properties
- Space and Time uncoupling

	Groups	*Publish-subscribe systems*	*Message queues*	*DSM*	*Tuple spaces*
Space-uncoupled	Yes	Yes	Yes	Yes	Yes
Time-uncoupled	Possible	Possible	Yes	Yes	Yes
Style of service	Communication-based	Communication-based	Communication-based	State-based	State-based
Communication pattern	1-to-many	1-to-many	1-to-1	1-to-many	1-1 or 1-to-many
Main intent	Reliable distributed computing	Information dissemination or EAI; mobile and ubiquitous systems	Information dissemination or EAI; commercial transaction processing	Parallel and distributed computation	Parallel and distributed computation; mobile and ubiquitous systems
Scalability	Limited	Possible	Possible	Limited	Limited
Associative	No	Content-based publish-subscribe only	No	No	Yes

Message Driven Bean

- Using JMS
- Message Driven Beans in practice

Problems with RMI

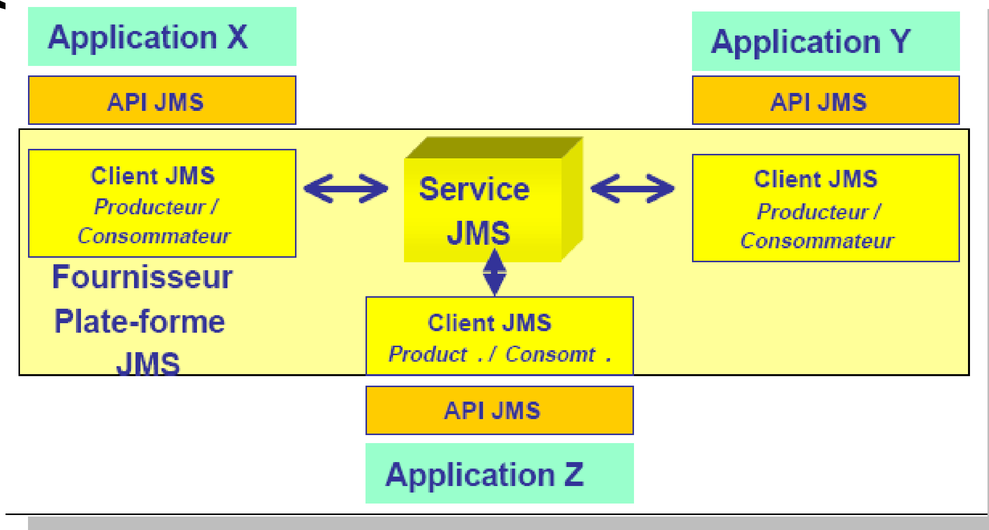
- The client must wait the answer
- The client is coupled to the server
- Faults
  - Server
  - Network
- On client talk to one server

MOM

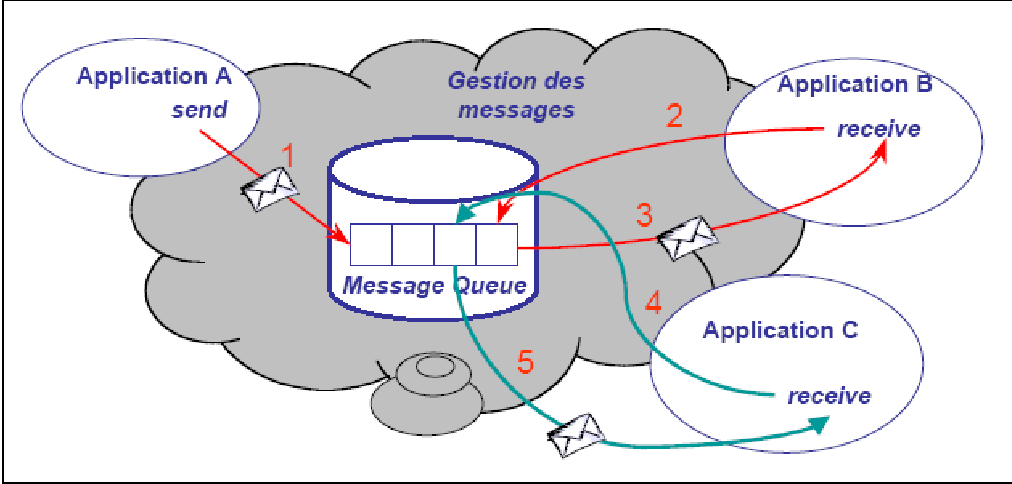
- Non blocking
- Not coupled
- Safety
- Several senders, several receivers
- Lower performances

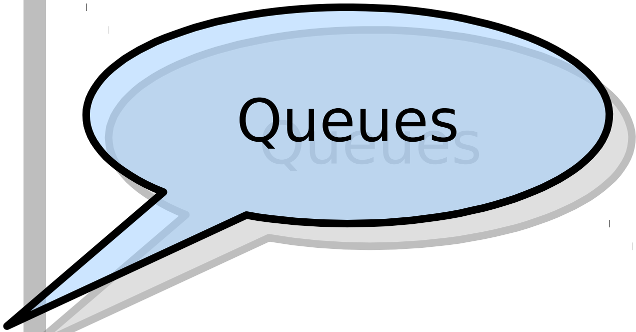
JMS

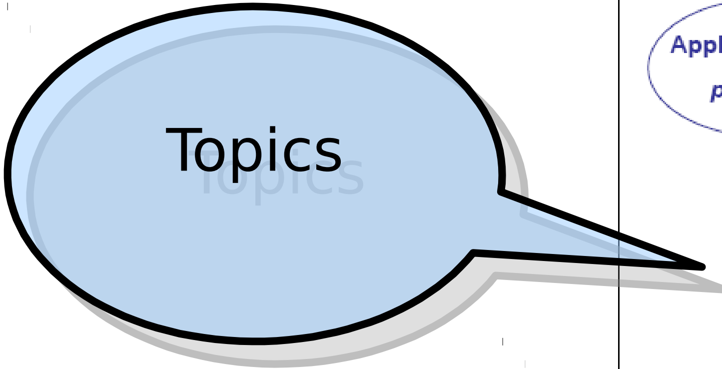
- Specification of a messaging service in Java
  - JMS Provider
  - JMS Client
  - JMS Message

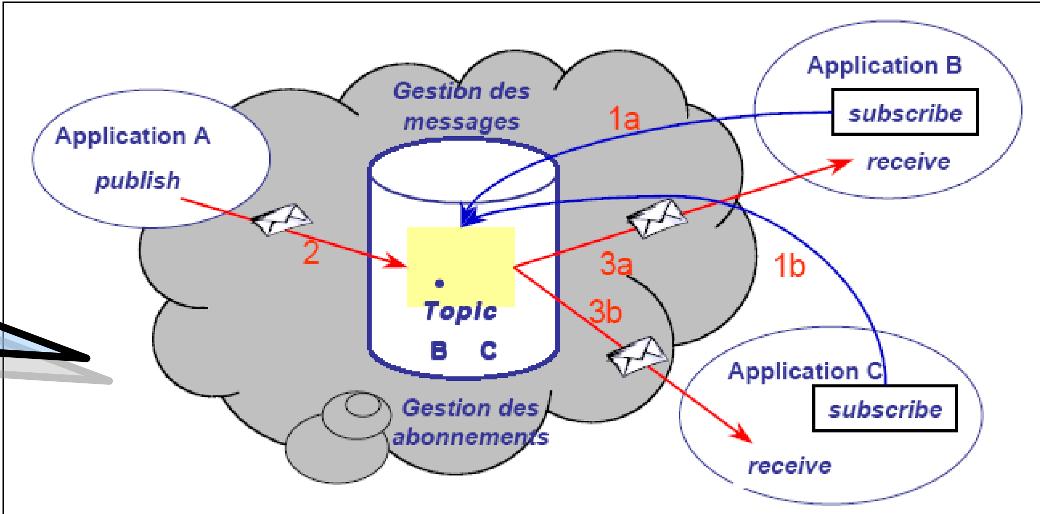


Communication domains



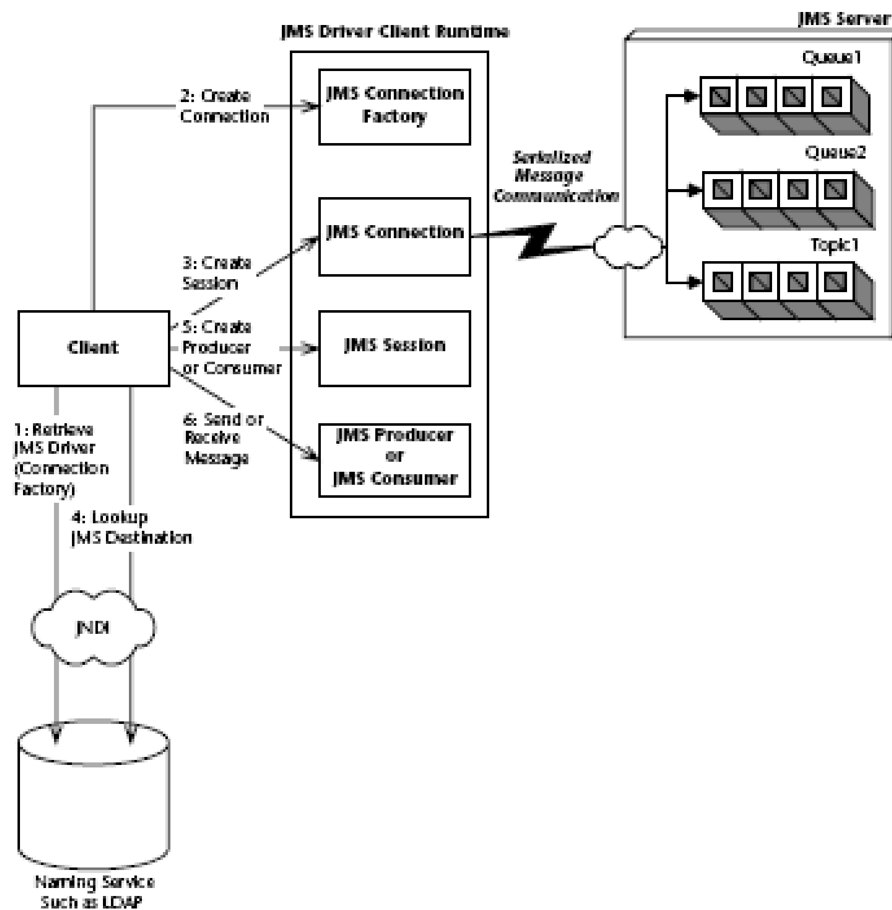






Le protocole JMS

- Trouver le



- Envoyer ou

Garantie d'acheminement

- Dans un MOM, l'acheminement des messages est garanti.
- Un message est émis
- Il est consommé
- Si le MOM ne reçoit pas de « ack », il remet le message dans la queue
- Variantes
  - Certified message delivery
  - Store and Forward

Un exemple simple

```
public static void main(String[] args) throws NamingException, JMSException {
    System.setProperty("java.naming.factory.initial",
            "com.sun.enterprise.naming.SerialInitContextFactory");
    System.setProperty("java.naming.factory.url.pkgs",
            "com.sun.enterprise.naming");
    System.setProperty("java.naming.factory.state",
            "com.sun.corba.ee.impl.presentation.rmi.JNDIStateFactoryImpl");
    System.setProperty("java.naming.provider.url",
            "localhost");
    InitialContext ctx=new InitialContext();
```

Comme d'habitude, on se connecte au serveur de nommage

Les étapes

- Récupération de la QueueConnectionFactory
- Création de la connection
- Création de la session
- Récupération de la queue

```
QueueConnectionFactory qcf=(QueueConnectionFactory) ctx.lookup("jms/queueConnFactory");
QueueConnection qc=(QueueConnection)qcf.createConnection();
QueueSession session=qc.createQueueSession(false,Session.AUTO_ACKNOWLEDGE );
Queue q=(Queue)ctx.lookup("jms/myQueue");
```

La réception des messages

```
QueueReceiver receiver=session.createReceiver(q);
qc.start();
while (true) {
    TextMessage text=(TextMessage)receiver.receive(0);
    System.out.println(text.getText());
    if (text.getText().equals("end")) break;
}
receiver.close();
session.close();
qc.stop();
qc.close();
```

Facile à comprendre

L'envoi de message

```
QueueSender sender=session.createSender(q);
for (int i=0;i<4;i++) {
    TextMessage txt=session.createTextMessage("message "+i);
    sender.send(txt);
}
TextMessage fin=session.createTextMessage("end");
sender.close();
```

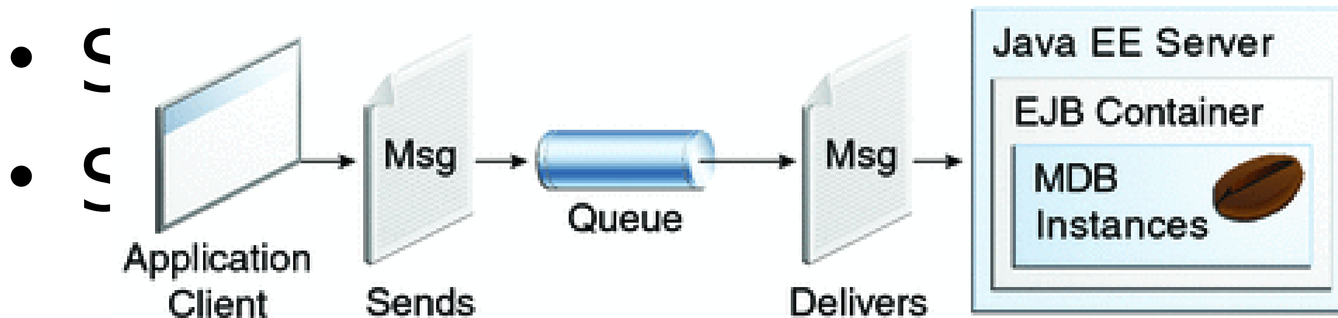
Le début est pareil.
Ensuite, on crée les messages et on les envoie

Les MDB

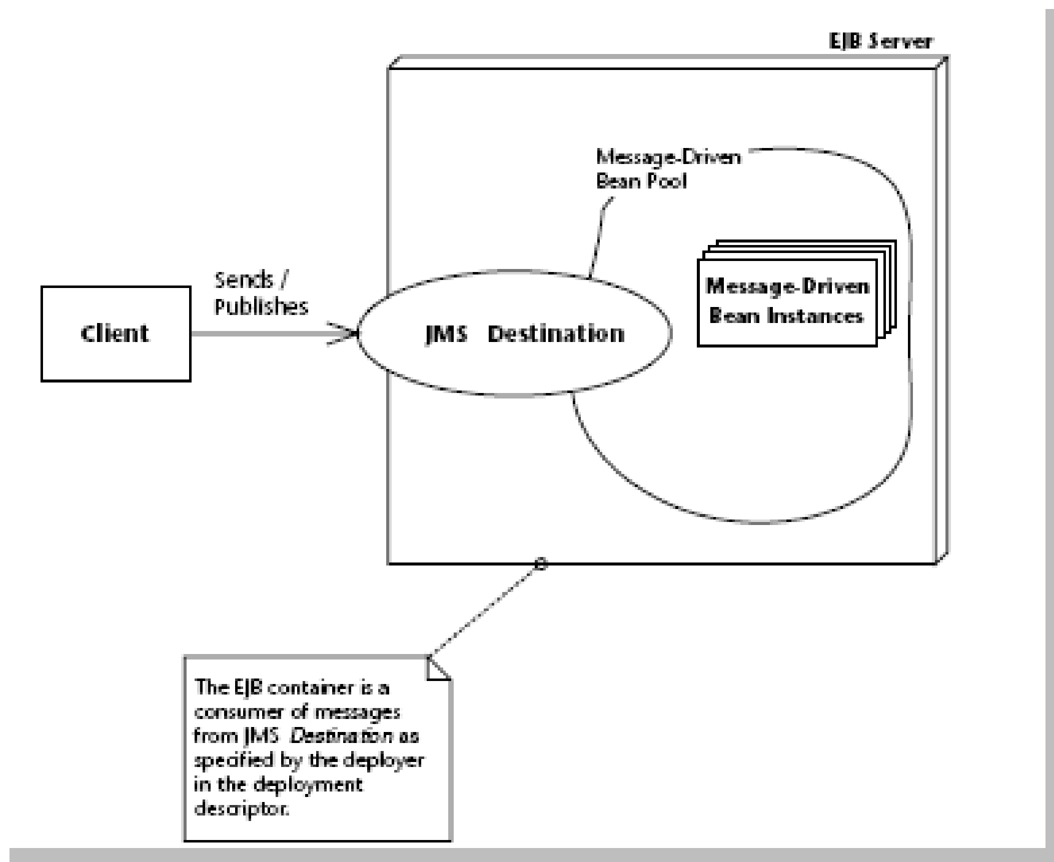
- Approche sans MDB
  - Configuration et connection à faire à la main
  - Gestion du cycle de vie
  - Gestion du contexte d'exécution
- Approche avec Session Bean
  - Aussi gestion du cycle de vie
  - threading

Un MDB

- EJB sans remote/local interface
- Une seule méthode : onMessage (Listener)
- Pas de message de retour
- Pas d'exception



Le client est un client JMS



Un exemple

```
    ..
public class ReceiveBean implements MessageListener {

    /** Creates a new instance of ReceiveBean */
    public ReceiveBean() {
    }

    public void onMessage(Message message) {
        TextMessage txt =(TextMessage)message;
        try {
            System.out.println(txt.getText());
        } catch (JMSException ex) {
            ex.printStackTrace();
        }
    }
```

Configuration avancée

```
@MessageDriven(mappedName = "jms/myQueue", activationConfig =  {
        @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "messageSelector", propertyValue = "JMSType='log'"),
        @ActivationConfigProperty(propertyName = "subscriptionDurability", propertyValue = "NonDurable")
    })
public class ReceiveBean implements MessageListener {
```

Pour les détails, voir la spécification

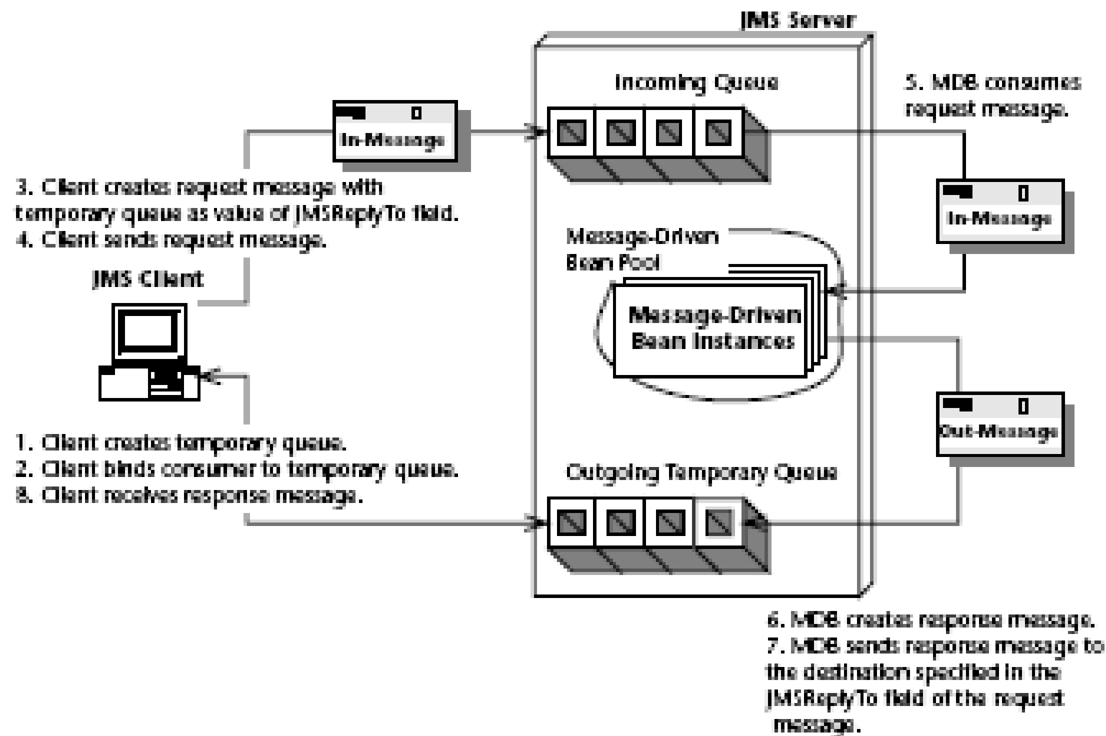
Concepts avancés

- Transaction différente de celui de l'émétteur du message
- Le contexte de sécurité n'est pas transmis explicitement
- Load Balancing plus simple mais attention au comportement des topics dans les clusters

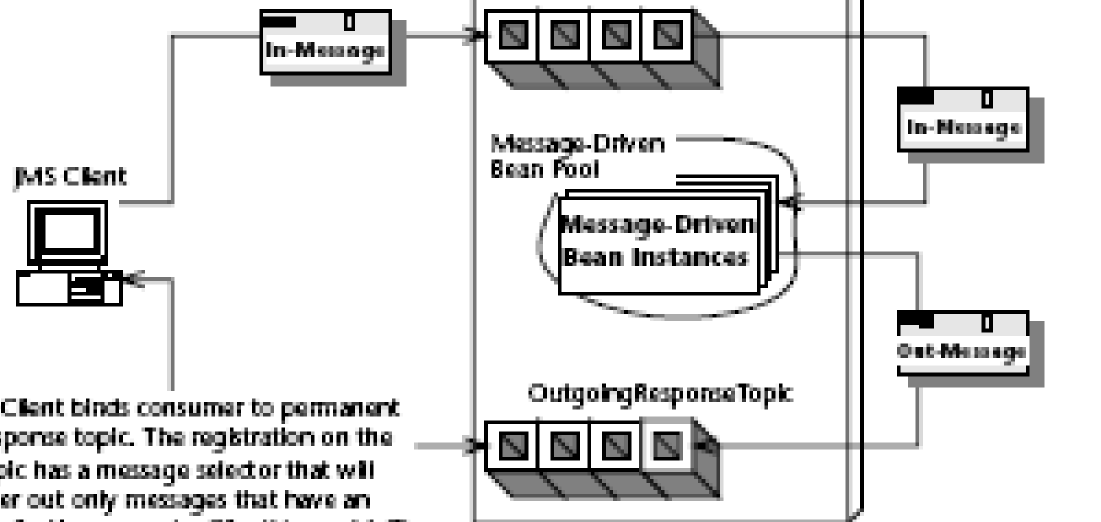
Les problèmes

- L'ordre de traitement des messages n'est pas garanti
- Les messages poisons (messages faisant toujours échouer le traitement)
  - Traitement manuel
  - Gestion par l'administrateur

Les réponses : queue temporaire



Les réponses : topic commun



Sur les MDB

- Encore primitif
  - À quand les vrais appels asynchrones
- Efficace pour la répartition de la charge
- Meilleure tolérance aux pannes
- Moins performants