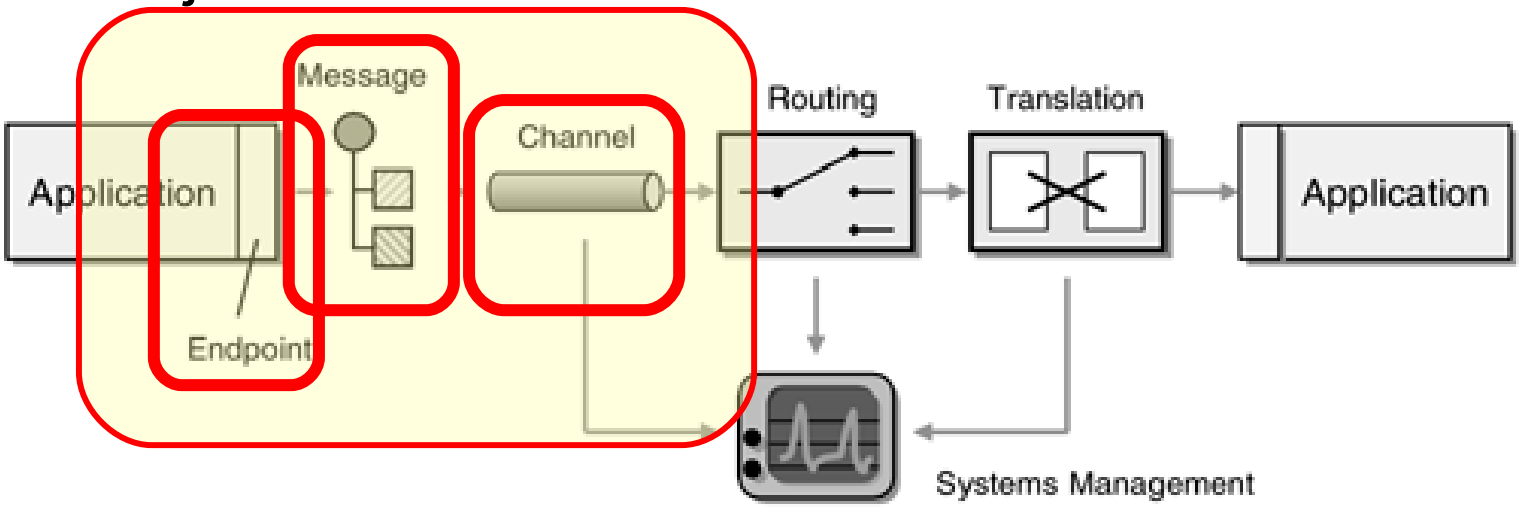
Introducción a JMS

Sistemas de Mensajería & JMS

Caracterización de JMS

- JMS es un API para trabajar con Sistemas de Mensajería



ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

JMS no especifica

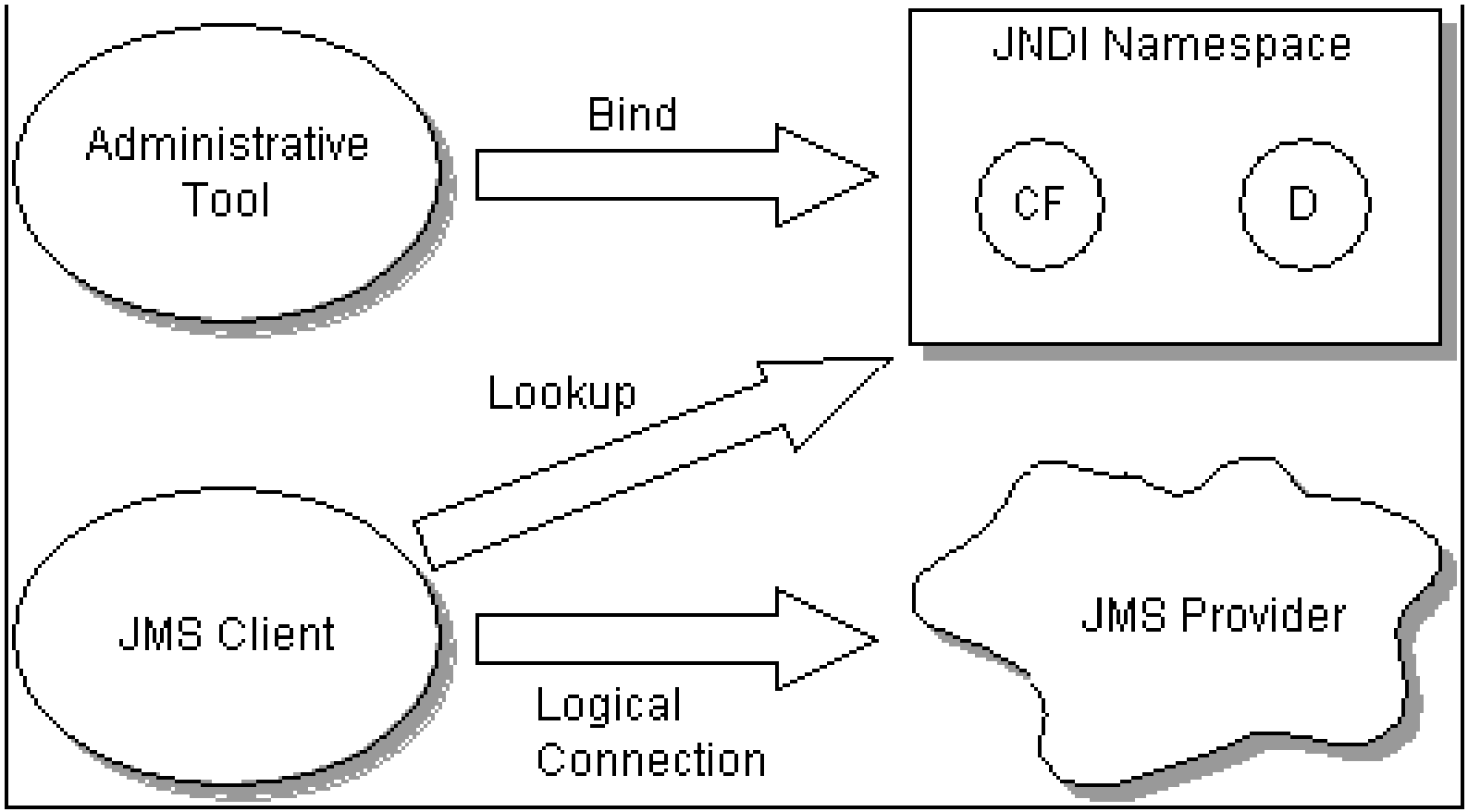
- HA Capabilities
  - Load Balancing / Fault Tolerance
- Error / Advisory Notification
- Administration
- Security
  - Se considera “JMS Provider feature”
- Wire Protocol
- Message Type Repository
  - No “message metadata”

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

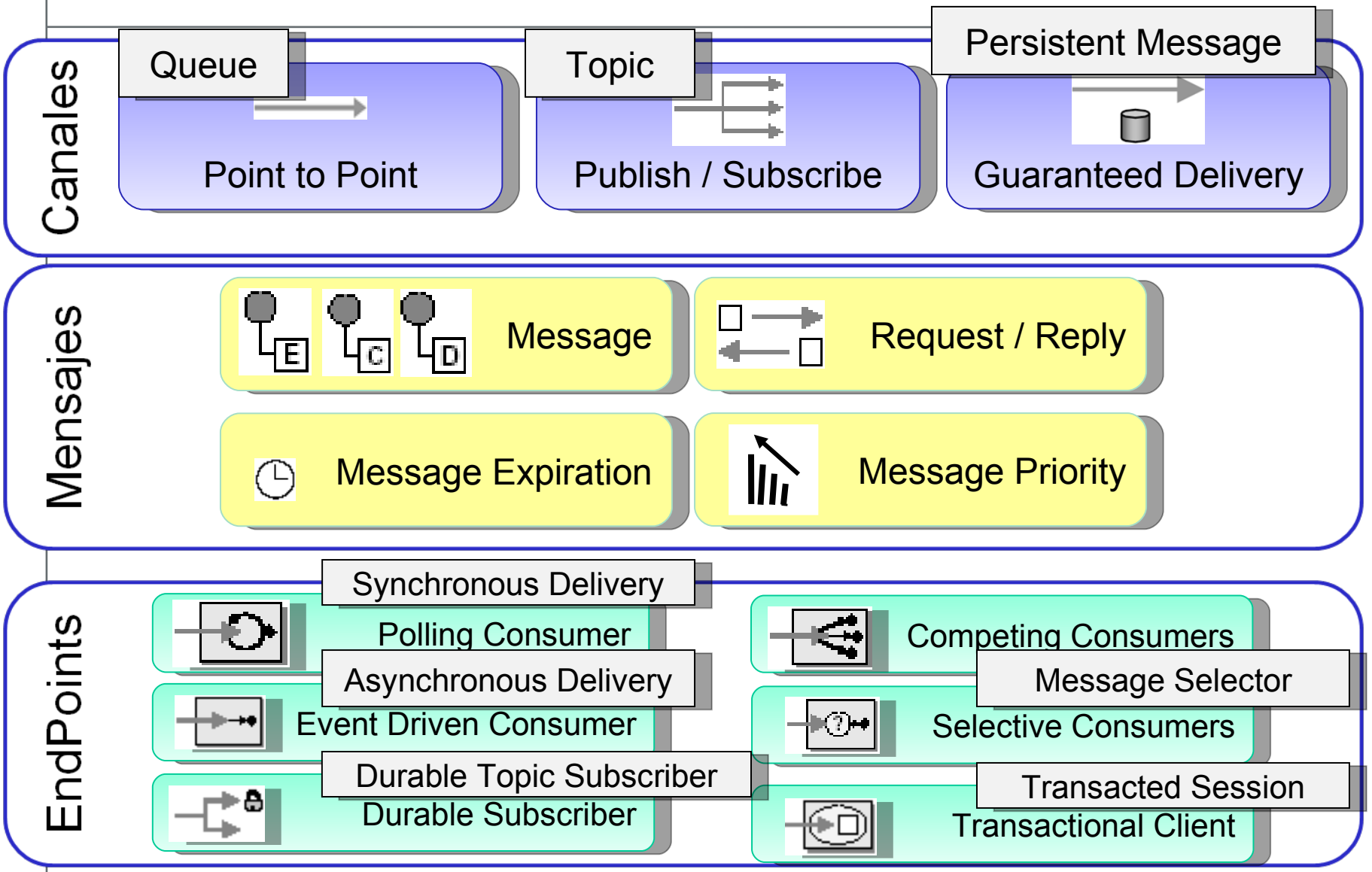
En qué consiste una Aplicación JMS?

- JMS Clients
  - Clientes Java
- Non-JMS Clients
  - Clientes usando interfaces nativas
- Messages
  - Cada aplicación utiliza un conjunto de mensajes
- JMS Provider
  - Messaging System implementing JMS
- Administered Objects
  - Destinations
  - ConnectionFactory

JMS Administration



Modelo JMS

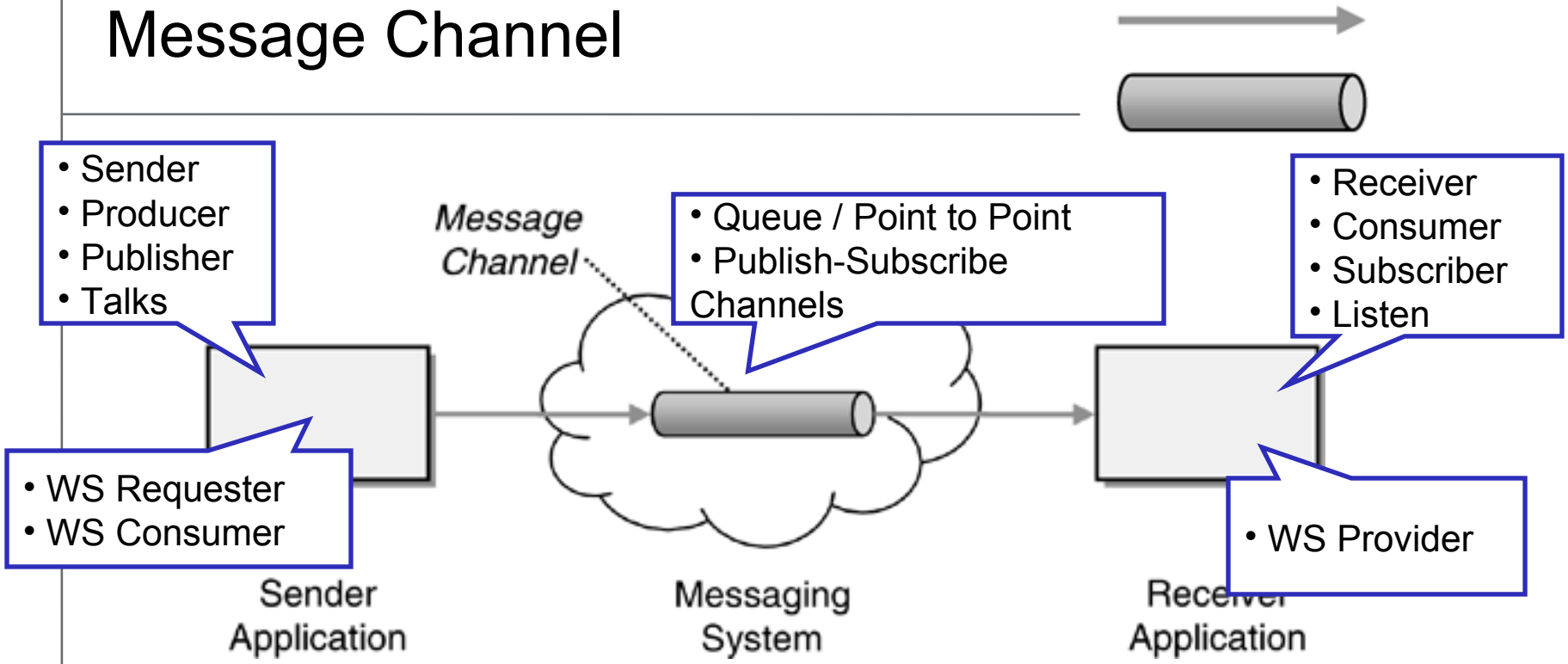

Canales
Queue
Point to Point
Topic
Publish / Subscribe
Persistent Message
Guaranteed Delivery
Mensajes
E
C
D
Message
Request / Reply
Message Expiration
Message Priority
EndPoints
Synchronous Delivery
Polling Consumer
Asynchronous Delivery
Event Driven Consumer
Durable Topic Subscriber
Durable Subscriber
Competing Consumers
Message Selector
Selective Consumers
Transacted Session
Transactional Client

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

JMS 1.1

- Versión anterior 1.0.2b
  - Dos “dominios”
    - Point to Point
    - Publish / Subscribe
  - Diferentes APIs
- Introduce un API común
  - Se mantienen los APIs anteriores
  - Se recomienda el uso del nuevo API “domain-independent”
    - Common Interfaces
- Ventajas
  - Modelo más simple
  - Queues & Topics in the same transaction (same session)

Message Channel



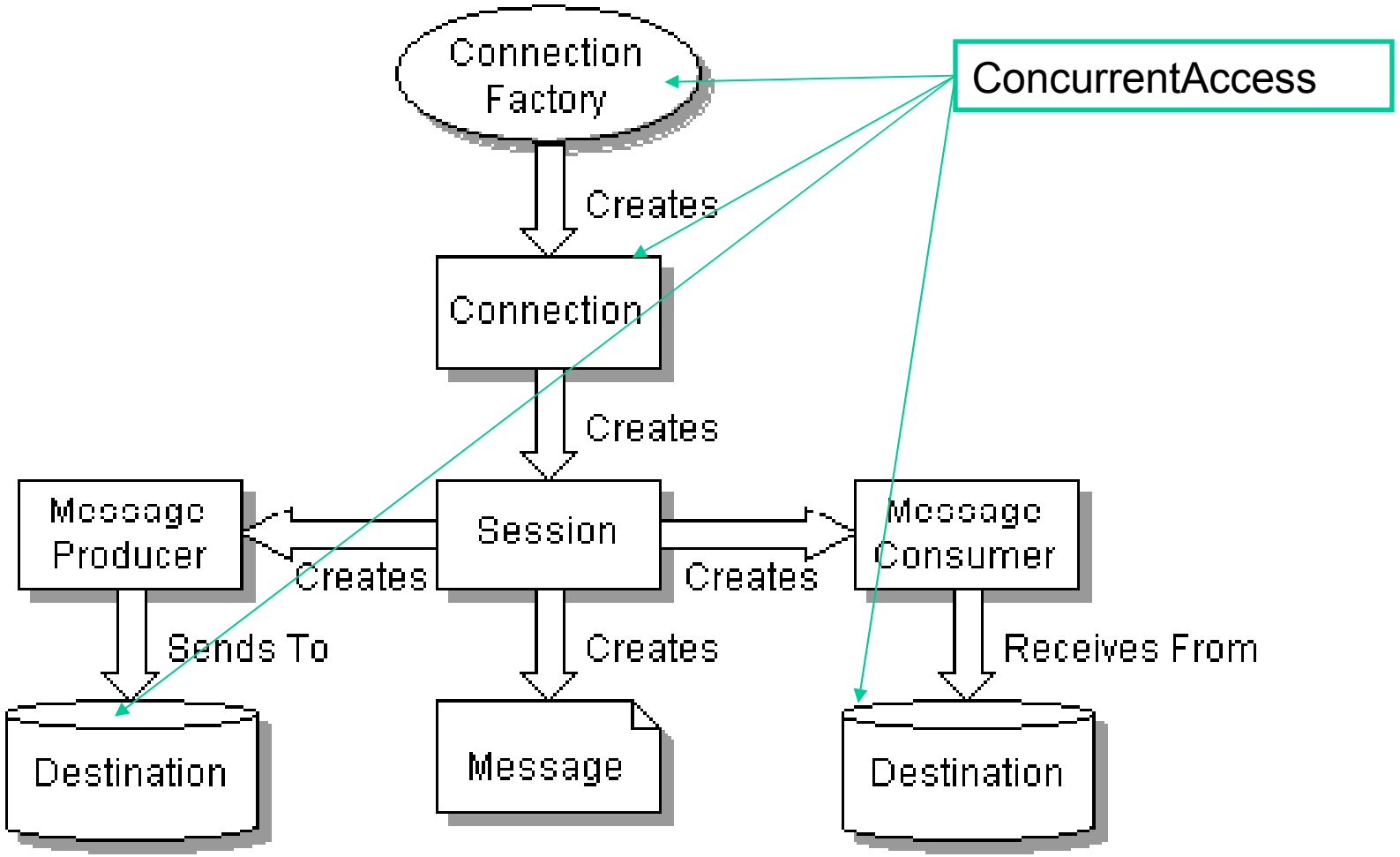
- Direcciones lógicas en el sistema de Mensajería
- Se diseñan considerando la intencionalidad del Mensaje
- Generalmente son estáticos
  - Necesidad de acuerdo entre distintas aplicaciones

Ejemplo: J2EE 1.4 RI

- Creación
  - j2eeadmin -addJmsDestination jms/mytopic topic
  - j2eeadmin -addJmsDestination jms/myqueue queue
- Acceso desde un cliente JMS
  - Context jndiContext = new InitialContext();
  - Queue myQueue = (Queue) jndiContext.lookup("jms/myqueue");
  - Topic myTopic = (Topic) jndiContext.lookup("jms/mytopic");

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

JMS Object Relationships



ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

JMS Domains & APIs

JMS Common API	PTP-specific API	Pub/Sub-specific API
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver, QueueBrowser	TopicSubscriber

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

JMS Messages

- Header
  - Mismos campos para todos los mensajes
- Properties
  - Standard-properties
  - Application-specific
    - Permiten la aplicación de criterios de selección usando criterios específicos de la aplicación.
  - Provider-specific
- Body
  - JMS define varios tipos de mensajes

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Message Header Fields

- JMSDestination
- JMSDeliveryMode
  - NON_PERSISTENT, PERSISTENT
- JMSMessageID
- JMSTimestamp
- JMSCorrelationID
- JMSReplyTo
- JMSRedelivered
- JMSType
- JMSExpiration
- JMSPriority

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Setting Header Values

Header Fields	Set By	Adm. Overriding	Used in Msg. Selector
JMSDestination	Send Method		
JMSDeliveryMode	Send Method	x	x
JMSExpiration	Send Method	x	
JMSPriority	Send Method	x	x
JMSMessageID	Send Method		x
JMSTimestamp	Send Method		x
JMSCorrelationID	Client		x
JMSReplyTo	Client		
JMSType	Client		x
JMSRedelivered	Provider		

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Message Properties

- Se establecen antes del envío
  - Son read-only para el receptor
- Su manejo es más costoso que el Body del mensaje
  - Su uso se orienta a la extensión del Header para permitir la selección de mensajes
  - Pueden ser redundantes con parte del contenido del mensaje
- Valores
  - boolean, byte, short, int, long, float, double, String

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Message Selection

- JMS soporta selección de mensajes a nivel del Provider
  - Message selector expression
  - Sintaxis basada en SQL 92 conditional expression
    - Operadores lógicos
    - BETWEEN, LIKE, IN (...), IS NULL
- Por ejemplo:
  - TipoDeCliente = ‘Mayorista’
  - Edad > 18
  - Monto > 1000 AND Rubro IN (‘A’, ‘B’)

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Tipos de Mensajes JMS (Contenido)

- TextMessage
  - textMessage.getText() retorna String
- ByteMessage
  - byteMessage.readBytes(byteArray)
- ObjectMessage
  - objectMessage.getObject()
- StreamMessage
  - readBoolean, readChar(), ...
- MapMessage
  - Java.util.Map, String keys
  - getBoolean(“isEnabled”), getInt(“numberOfItems”)

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Transactions

- Una Session puede declararse Transacted
  - commit(), rollback()
  - JTS / JTA
- Distributed Transactions
  - Son opcionales
  - Soportadas mediante JTA XAResource API
  - Se propone su uso “pensando” en integración de JMS en Application Servers

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Orden de Recepción de Mensajes

- Mensajes enviados por una sesión a un destino deben ser entregados respetando el orden en que fueron enviados.
  - Algunas condiciones pueden incluso alterar este orden
- Esto no garantiza orden para
  - Mensajes enviados a distintos destinos
  - Mensajes enviados a un mismo destino desde distintas sesiones

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Mensajes desde una sesión a un destino

- Condiciones que pueden afectar el orden:
  - Message Priority
  - Mensajes NON_PERSISTENT pueden perderse
  - El orden se garantiza dentro de la misma modalidad de mensajes
    - PERSISTENT, NON_PERSISTENT

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Message Acknowledgment

- Sesiones transaccionales manejan acknowledge automático
- En forma manual
  - DUPS_OK_ACKNOWLEDGE
    - Lazily acknowledges handled by session
    - Posibilidad duplicación de mensajes
  - AUTO_ACKNOWLEDGE
    - ACK automático al finalizar el método de recepción / callback
  - CLIENT_ACKNOWLEDGE
    - ACK manejado por el cliente
    - Alcanza a todos los mensajes recibidos por esa sesión
    - Configuración de recursos suficientes
- Recover()
  - Permite recepción de los mensajes sin ACK.

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Request / Reply

- JMS soporta Request/Reply tanto en PTP cómo Pub/Sub
- Enfoque básico
  - Especificación de Destino para Respuesta
  - JMSCorrelationID para identificar el mensaje original
  - Creación de Queues/Topics temporales
- Cada proveedor puede extender el soporte básico

ideasoft®
TECHNOLOGY & BUSINESS PERFORMANCE

Mensajería y Confiabilidad

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Mensajería y Confiabilidad

- Conceptos
  - Garantía de entrega ?
  - Cuántas veces se entrega ?
  - Atomicidad ?
  - Se mantiene el orden ?
- Consideraciones
  - Qué mecanismos deben activar Senders & Receivers?
  - Qué funcionalidades de JMS modifican el funcionamiento básico ?
  - Bajo qué contextos se garantiza el funcionamiento estándar ?
  - Qué efectos se producen bajo condiciones excepcionales ?

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

JMS Reliability

- Es el resultado de la combinación de:
  - Transacted Session
  - Message Acknowledge
  - Persistent Message
  - Durable Receiver
  - Message Expiration
  - Message Priority
  - Temporal Destination

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Reliable Queue bajo J2EE

- Usar
  - Persistent Message
  - Transacted Session
    - Lo impone EJB
  - Message Acknowledge
    - Lo impone EJB
- Según el caso (con cuidado)
  - Message Expiration
  - Priority
- No usar
  - Temporal Destinations

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Reliable Topic bajo J2EE

- Usar
  - Persistent Message
  - Durable Subscription
  - Transacted Session
    - Lo impone EJB
  - Message Acknowledge
    - Lo impone EJB
- Según el caso (con cuidado)
  - Message Expiration
  - Priority

Queues

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

QueueBrowser

- Un cliente puede utilizar un objecto *QueueBrowser* para obtener los mensajes en una cola sin consumirlos

```
session.createBrowser(queue);
session.createBrowser(queue, selector);
```

- El método *getEnumeration* retorna una instancia de *java.util.Enumeration* que sirve para recorrer los mensajes.
  - Puede contener todos los mensajes de la cola o un subconjunto si se utilizó un “message selector”
- Mientras se recorre la enumeración pueden estar llegando o consumiéndose mensajes.
  - El comportamiento en este caso depende del proveedor

ideasoft®
TECHNOLOGY & BUSINESS PERFORMANCE

Topics

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Durable Subscriptions

- Se debe utilizar el método ***createDurableSubscriber(Topic topic, String subscriptionName)*** de Session:

```
TopicSubscriber sc =
  session.createDurableSubscriber(topic, "Mis
  Alertas");
```

- Además la conexión debe tener asociado un ClientID:

```
connection = cf.createConnection();
connection.setClientID(myClientId);
```

- La pareja ClientId – SubscriptionName identifican únicamente a la suscripción

ideasoft®
TECHNOLOGY & BUSINESS PERFORMANCE

MDB - Message Driven Beans

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

MDB - Message Driven Beans

- Beans que no ofrecen un API hacia los clientes
- La comunicación con ellos se realiza a través de mensajes
- Un MDB puede ser el receptor de un *Topic* o de una *Queue*
- En la práctica un MDB es una clase Java que implementa las interfaces:
  - *javax.ejb.MessageDrivenBean*
  - *javax.jms.MessageListener*

MDB – Ejemplo

```
public class MDBSample implements MessageDrivenBean,
                MessageListener {
    public void setMessageDrivenContext(MessageDrivenContext ctx)
       {
        this.ctx = ctx;
    }
    public void ejbCreate() {
    }
    public void ejbRemove() {
    }
    public void onMessage(Message msg) {
        //procesamiento del mensaje
    }

    private MessageDrivenContext ctx;
}
```

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

MDB - Deployment Descriptor

- En el caso de los MDBs, en este archivo se especifica:
  - El nombre del bean
  - La clase que lo implementa
  - El modelo de mensajería (Queue o Topic).
  - Un MessageSelector que permite filtrar los mensajes en la cola o tópico especificado
- Por ejemplo:

```
<message-driven>
        <ejb-name>name</ejb-name>
        <ejb-class>com.x.y.mdb</ejb-class>
        <message-selector>Property='Value'</message-selector>
        <message-driven-destination>
           <destination-type>javax.jms.Queue</destination-type>
        </message-driven-destination>
</message-driven>
```

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

MDB – Annotations

- En JEE5 se puede reemplazar el deployment descriptor por *annotations*
- Por ejemplo:

```
@MessageDriven(activationConfig =
{
  @ActivationConfigProperty(propertyName="destinationType",
       propertyValue="javax.jms.Queue"),
  @ActivationConfigProperty(propertyName="destination",
       propertyValue="queue/testQueue"),
})
public class MDBSample implements MessageListener {
   ...
```

Clientes

- Los clientes para enviar o recibir los mensajes del MDB son clientes JMS
- Se deben conectar a la Queue o Topic:
  - Obtener una ConnectionFactory desde JNDI
  - Utilizando la factory, crear una conexión (Connection).
  - Crear una sesión
  - Obtener el objeto Queue o Topic desde JNDI
  - Para enviar mensajes:
    - Crear un Producer sobre el destino (Queue o Topic).
  - Para recibir mensajes:
    - Crear un Consumer sobre el destino.
    - Asignarle un MessageListener para recibir los mensajes.

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Un productor de mensajes

```
InitialContext jndi = getInitialContext();
ConnectionFactory factory = (ConnectionFactory)
   jndi.lookup("ConnectionFactory");

Queue pedidosQueue = jndi.lookup("PedidosQueue");
Connection connection = factory.createConnection();

Session session = connection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);
MessageProducer producer = session.createProducer(pedidosQueue);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

MapMessage msg = session.createMapMessage();
//carga de datos en el mensaje msg
producer.send(msg);
```

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

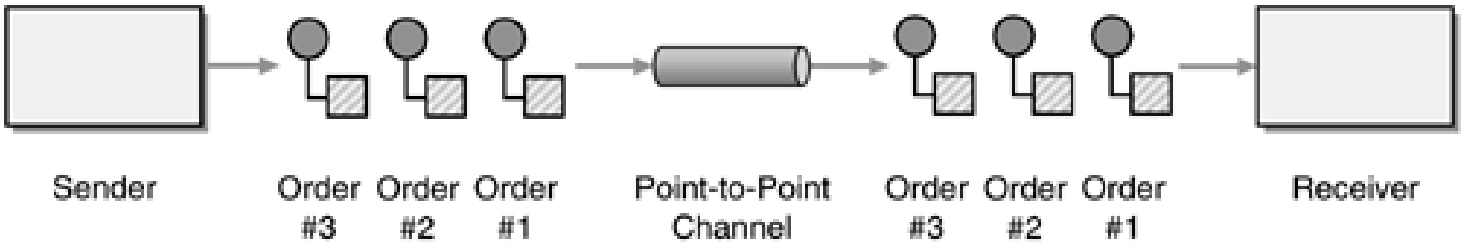
Un receptor de mensajes

```
public class Receptor implements MessageListener {
public Receptor() throws Exception {
    InitialContext jndi = getInitialContext();
    ConnectionFactory factory = (ConnectionFactory)
       jndi.lookup("ConnectionFactory");
    Queue respuestasQueue = jndi.lookup("RespuestasQueue");
    Connection connection = factory.createConnection();
    Session session = connection.createSession(false,
        Session.AUTO_ACKNOWLEDGE);
    MessageConsumer consumer =
       session.createConsumer(respuestasQueue);
    consumer.setMessageListener(this);
    connection.start();
}

public void onMessage(Message msg) {
   //procesamiento del mensaje
}
}
```

Patterns en JMS

Point to Point Channel

- PTP
  - Como máximo una entrega del mensaje.
  - Cuando hay más de un consumidor registrado
    - Pattern Competing Consumers
    - Facilita la escalabilidad



ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

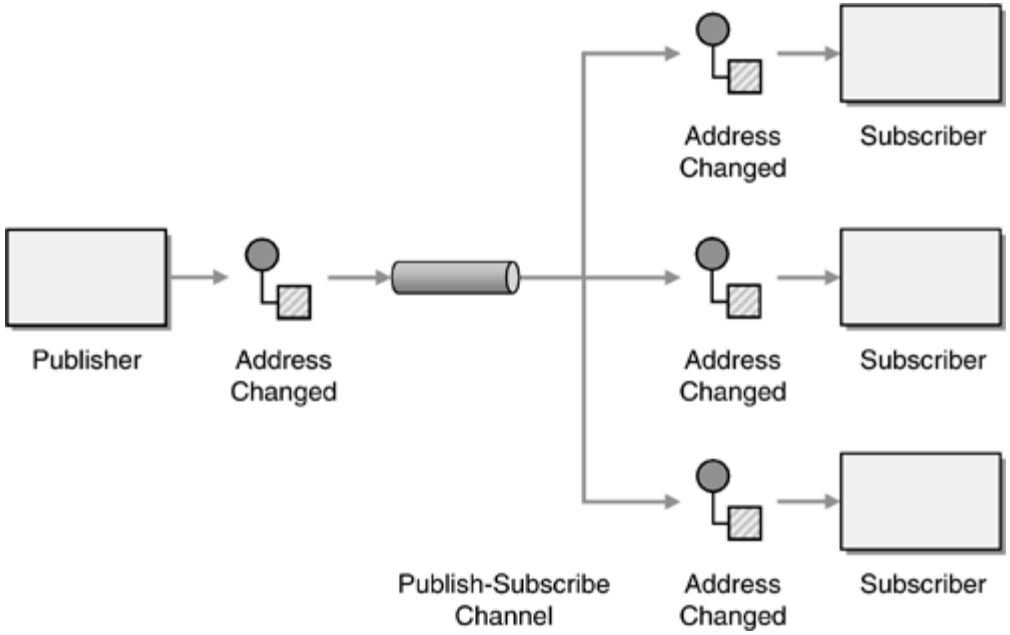
Point to Point Channel

```
InitialContext jndi = getInitialContext();
ConnectionFactory factory = (ConnectionFactory)
   jndi.lookup("ConnectionFactory");

Destination destination = jndi.lookup("PedidosQueue");
Connection connection = factory.createConnection();

Session session = connection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);

MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
MapMessage msg = session.createMapMessage();
//carga de datos en el mensaje msg

producer.send(msg);
```

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Publish-Subscribe Channel

- Pub/Sub Channels
  - Multiplican los mensajes entre todos los subscriptores activos
  - Cada mensaje es entregado a cada Receptor registrado
  - Esto está en la línea del Observer Pattern (pe. Java Listeners).



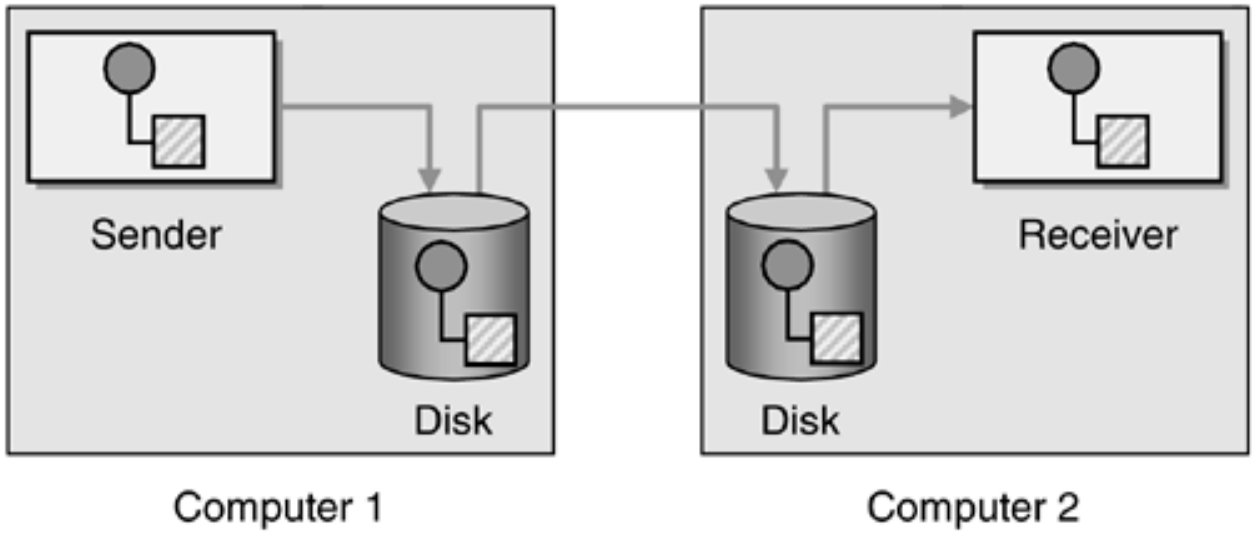
ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Publish-Subscribe Channel

```
InitialContext jndi = getInitialContext();
ConnectionFactory factory = (ConnectionFactory)
   jndi.lookup("ConnectionFactory");

Destination destination = jndi.lookup("AvisosTopic");
Connection connection = factory.createConnection();

Session session = connection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);

MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
MapMessage msg = session.createMapMessage();
//carga de datos en el mensaje msg

producer.send(msg);
```

Guaranteed Delivery

- Mediante el uso de almacenamiento estable, el sistema de mensajería puede garantizar la entrega de los mensajes
  - Diferencia entre Pub/Sub y PTP

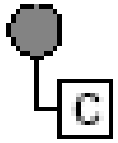


ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

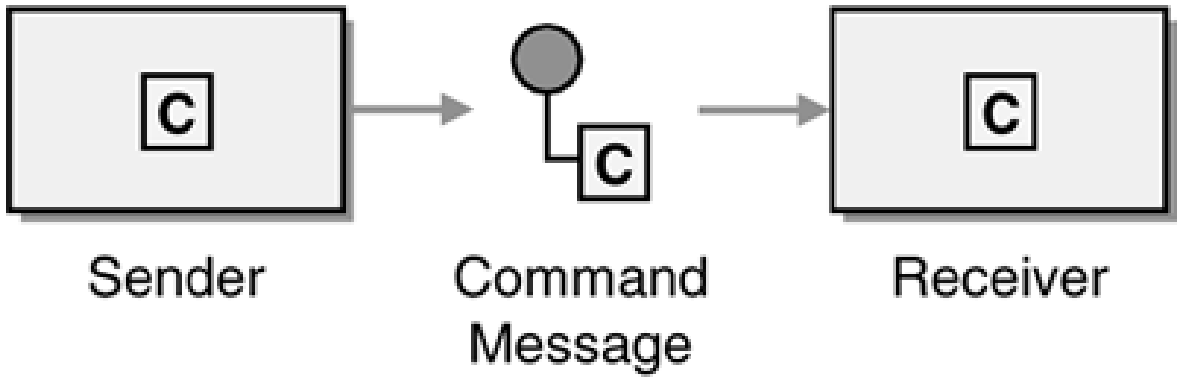
Guaranteed Delivery

```
InitialContext jndi = getInitialContext();
ConnectionFactory factory = (ConnectionFactory)
   jndi.lookup("ConnectionFactory");

Destination destination = jndi.lookup("PedidosQueue");
Connection connection = factory.createConnection();

Session session = connection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);

MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.PERSISTENT);
MapMessage msg = session.createMapMessage();
//carga de datos en el mensaje msg

producer.send(msg);
```

Command Message



- Invocación asincrónica de un procedimiento a través de un mensaje
  - Diferencias con respecto a RPC
  - Relativo acoplamiento entre Sender y Receiver



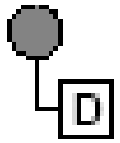
C = getLastTradePrice('DIS');

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

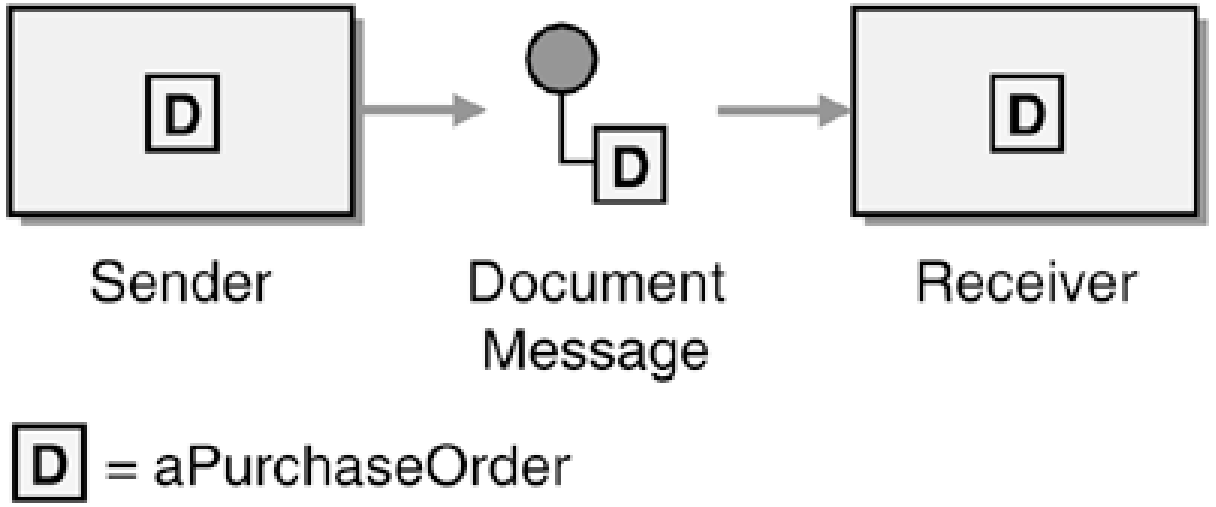
Command Message

```
InitialContext jndi = getInitialContext();
ConnectionFactory factory = (ConnectionFactory)
   jndi.lookup("ConnectionFactory");

Destination destination = jndi.lookup("PedidosQueue");
Connection connection = factory.createConnection();

Session session = connection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);

MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
MapMessage msg = session.createMapMessage();
msg.setString("command", "GetLastTradePrice");
msg.setString("symbol", "DIS");

producer.send(msg);
```

Document Message



- Transferencia de información entre aplicaciones

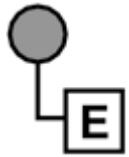


ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

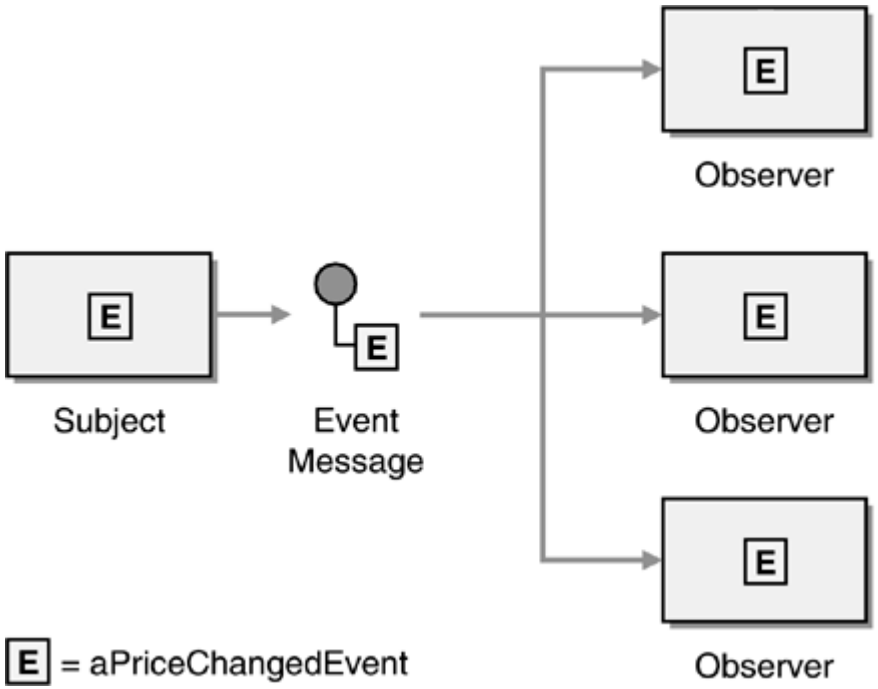
Document Message

```
InitialContext jndi = getInitialContext();
ConnectionFactory factory = (ConnectionFactory)
   jndi.lookup("ConnectionFactory");

Destination destination = jndi.lookup("PedidosQueue");
Connection connection = factory.createConnection();

Session session = connection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);

MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
PurchaseOrder order = new PurchaseOrder();
//cargar datos en la orden
ObjectMessage msg = session.createObjectMessage(order);
producer.send(msg);
```

Event Message



- Notificación confiable, asíncrona de eventos entre aplicaciones
  - Cuánta información debe viajar en el evento?



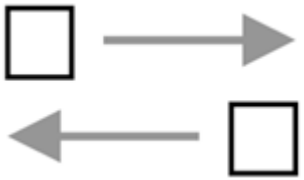
ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Event Message

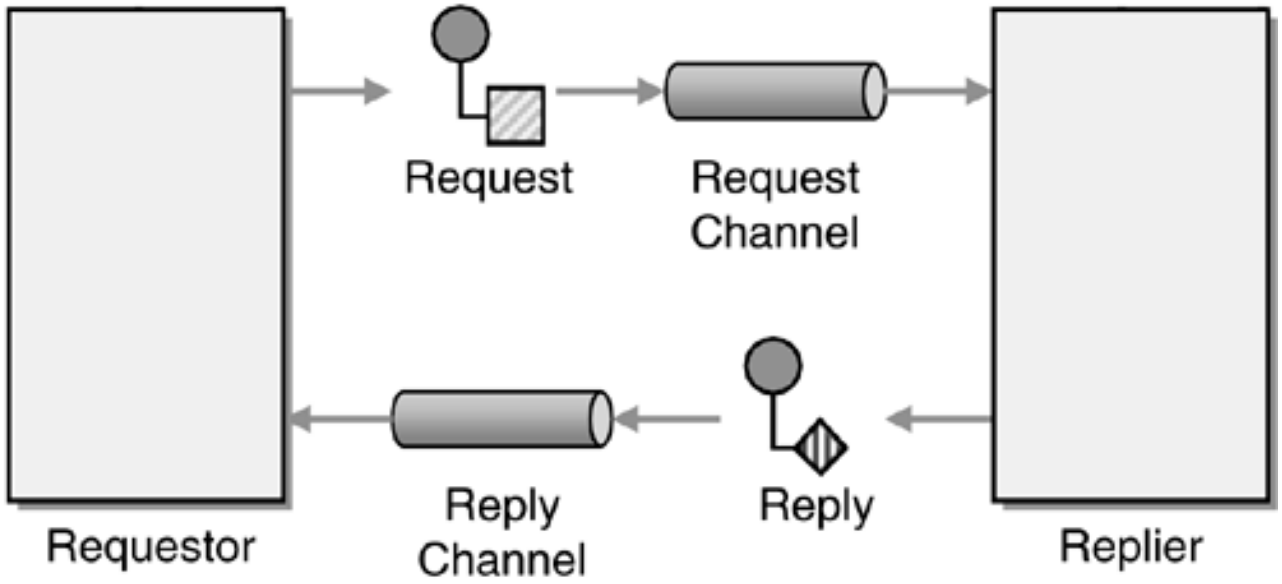
```
InitialContext jndi = getInitialContext();
ConnectionFactory factory = (ConnectionFactory)
   jndi.lookup("ConnectionFactory");

Destination destination = jndi.lookup("PedidosQueue");
Connection connection = factory.createConnection();

Session session = connection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);

MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
MapMessage msg = session.createMapMessage();
msg.setString("event", "PriceChangedEvent");
msg.setString("product", "12345");

producer.send(msg);
```

ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Request-Reply



- Se utilizan dos canales
- Dos alternativas
  - Bloqueante
  - Asincrónica



ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Request-Reply

```
InitialContext jndi = getInitialContext();
ConnectionFactory factory = (ConnectionFactory)
   jndi.lookup("ConnectionFactory");

Destination destination = jndi.lookup("PedidosQueue");
Connection connection = factory.createConnection();

Session session = connection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);
Destination tmpQueue = session.createTemporaryQueue();

MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
MapMessage msg = session.createMapMessage();
msg.setJMSReplyTo(tmpQueue);
//carga de datos en el mensaje msg
producer.send(msg);
```

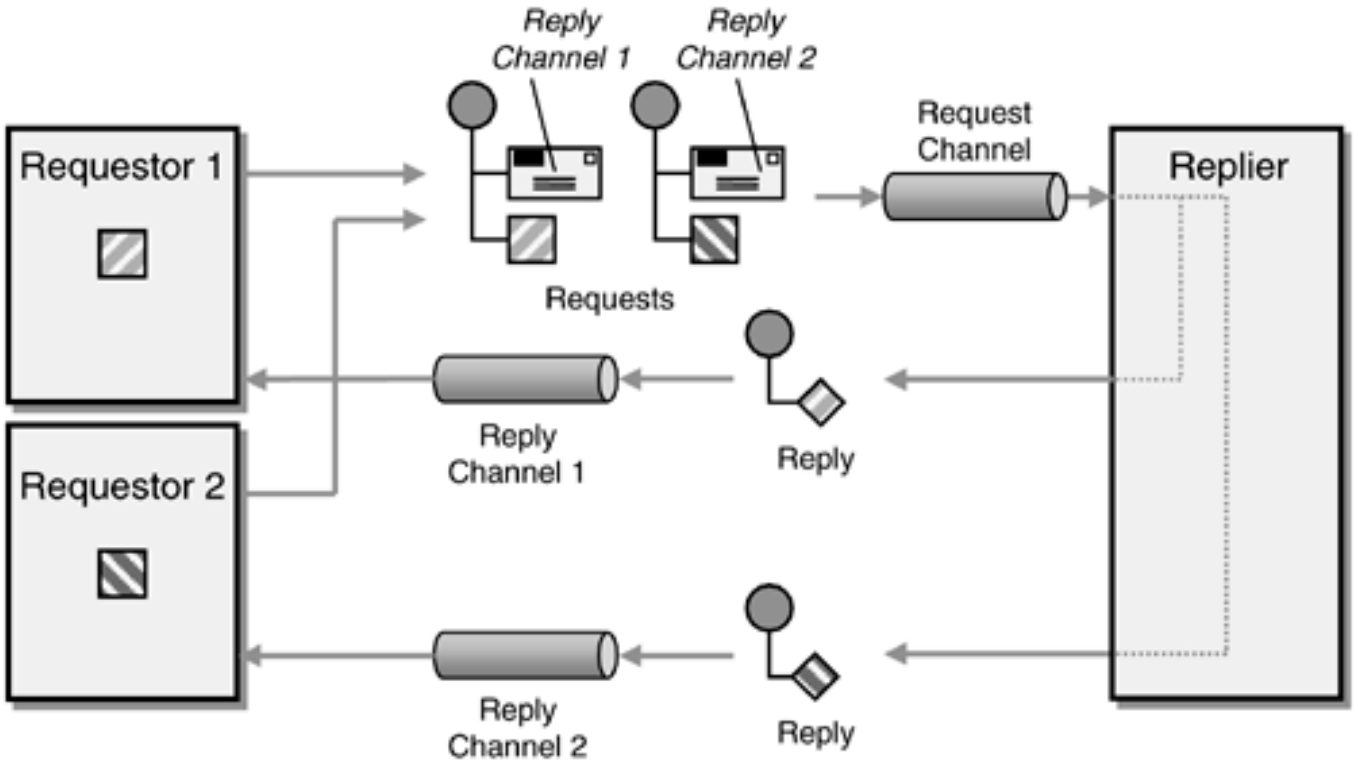
ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Request-Reply

```
MessageConsumer consumer = session.createConsumer(tmpQueue);
Message response = consumer.receive();
//procesar la respuesta

consumer.close();
tmpQueue.delete();
```

Return Address

- Dirección de ruteo de la respuesta esperada
  - Puede o no ser la dirección del Requestor
  - Si es donde se espera procesar la respuesta



ideasoft
TECHNOLOGY & BUSINESS PERFORMANCE

Return Address

```
InitialContext jndi = getInitialContext();
ConnectionFactory factory = (ConnectionFactory)
   jndi.lookup("ConnectionFactory");

Destination destination = jndi.lookup("PedidosQueue");
Destination responseDestination = jndi.lookup("ResponseQueue");

Connection connection = factory.createConnection();

Session session = connection.createSession(false,
   Session.AUTO_ACKNOWLEDGE);

MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
MapMessage msg = session.createMapMessage();
msg.setJMSReplyTo(responseDestination);
//carga de datos en el mensaje msg
producer.send(msg);
```

ideasoft®
TECHNOLOGY & BUSINESS PERFORMANCE

Gracias