Introducción al middleware

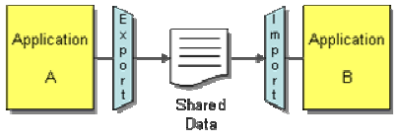
Mensajería

Agenda

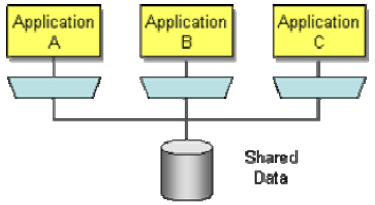
- Conceptos de Mensajería
- Sistemas de Mensajería
- Patrones de diseño
- Caso de Estudio
- Ejemplo - Java Message Service (JMS)

Estilos de Integración

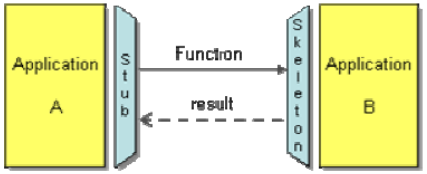
- File Transfer
  - Escritura, transferencia, lectura de un archivo.



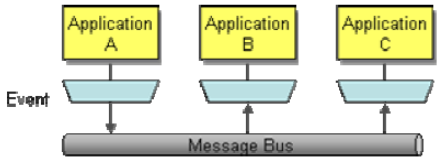
- Shared Database
  - Acceso compartido a una misma base de datos.



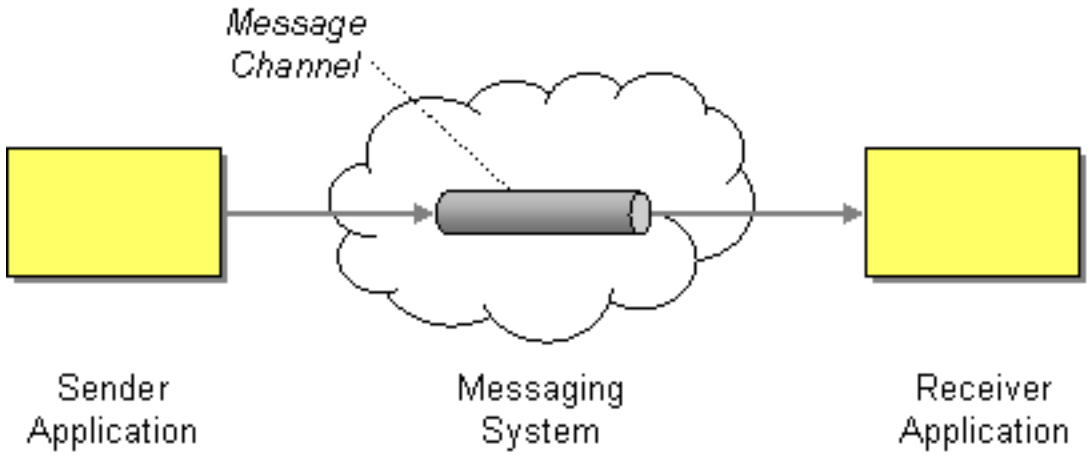
- Remote Procedure Call
  - Las aplicaciones clientes ejecutan servicios
  - remotos en forma sincrónica.



- Messaging
  - Envío de mensajes a través de canales en forma asincrónica.



¿Qué es mensajería?



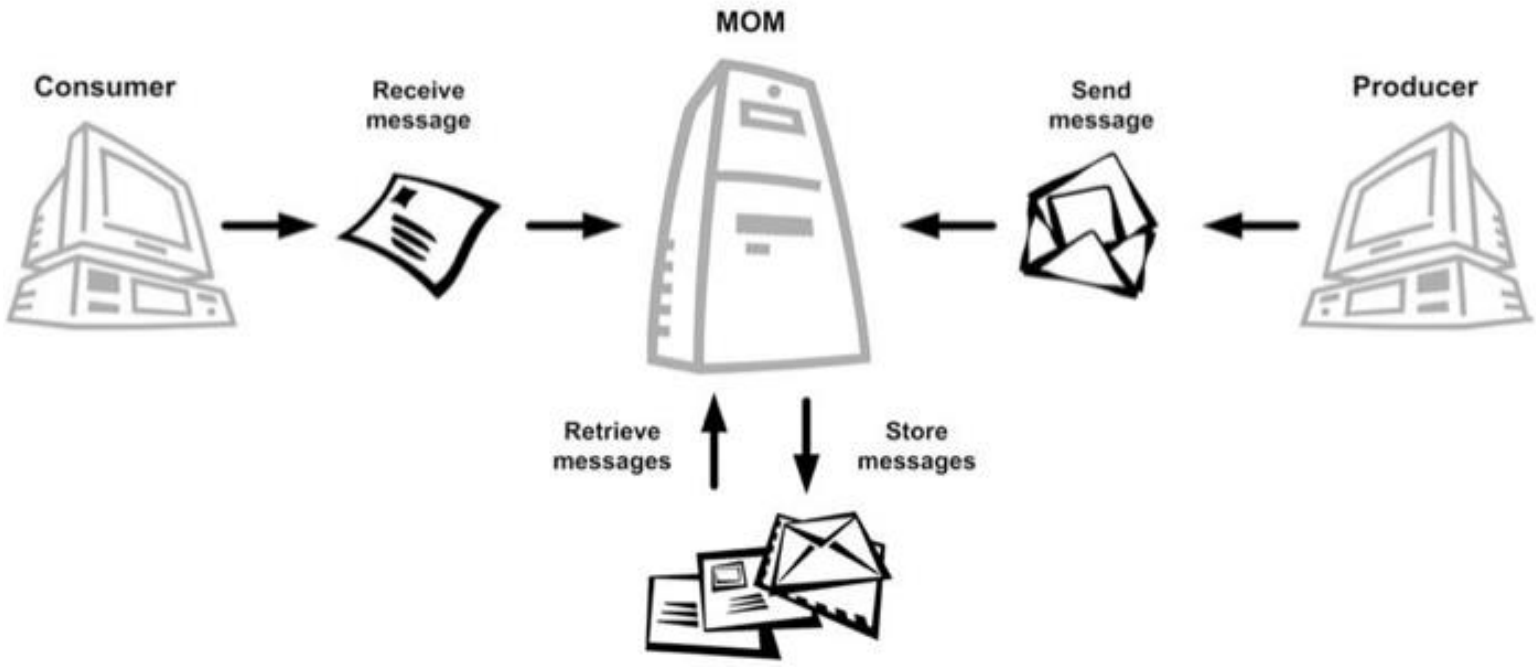
¿Qué es mensajería?

- Es una tecnología de middleware que permite establecer una comunicación de tipo programa-programa con las siguientes características:
  - Alta velocidad
  - Asincrónica
  - Garantía de entrega

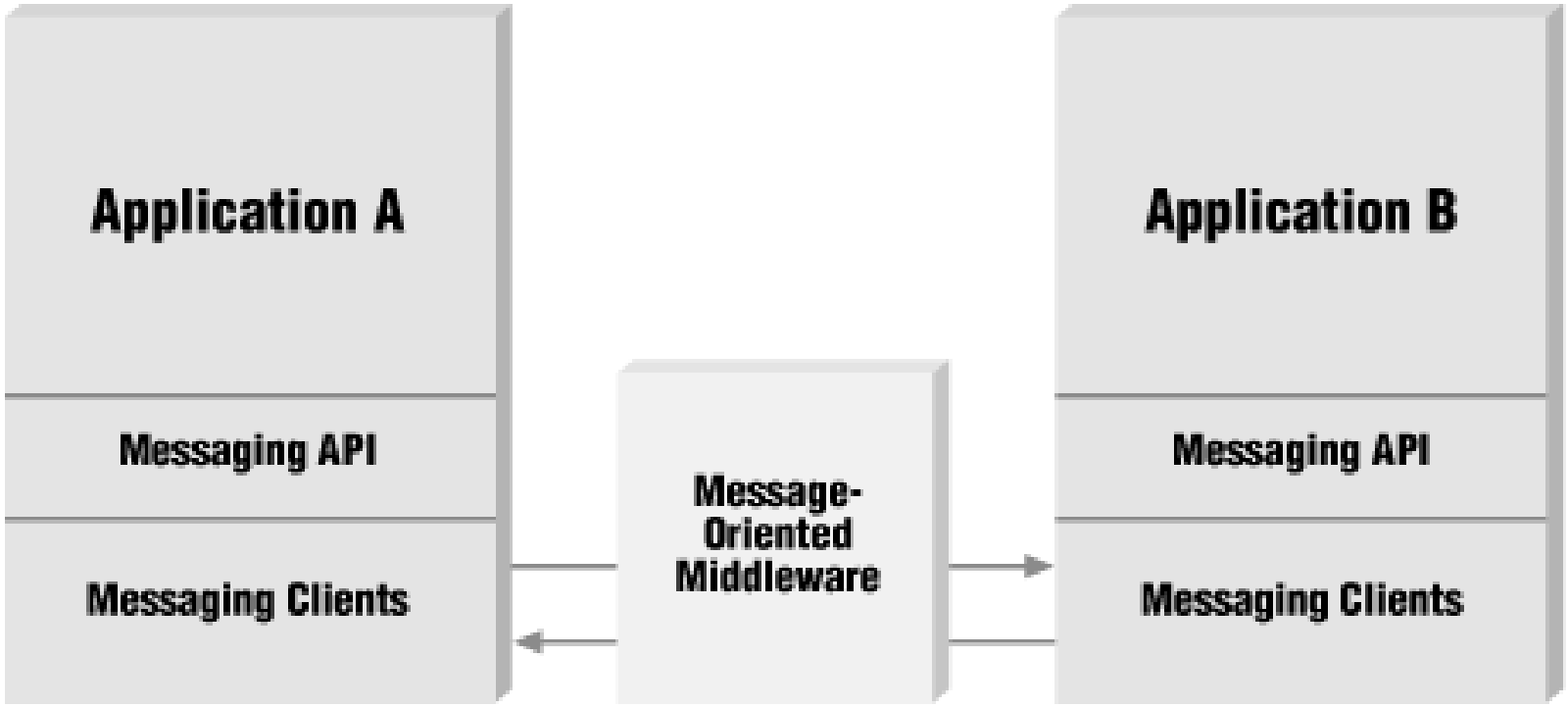
¿Qué es mensajería?

- Comunicación entre aplicaciones mediante el envío de paquetes de datos, llamados *mensajes*.
- Para el envío de estos mensajes se dispone de *canales*, que funcionan como colecciones de mensajes compartidas entre todas las aplicaciones, donde se dejan y desde donde se toman los mensajes.
- Un emisor/productor es una aplicación que envía los mensajes escribiendo el mensaje en el canal
- Un receptor/consumidor es una aplicación que recibe un mensaje leyéndolo (y borrándolo) del canal

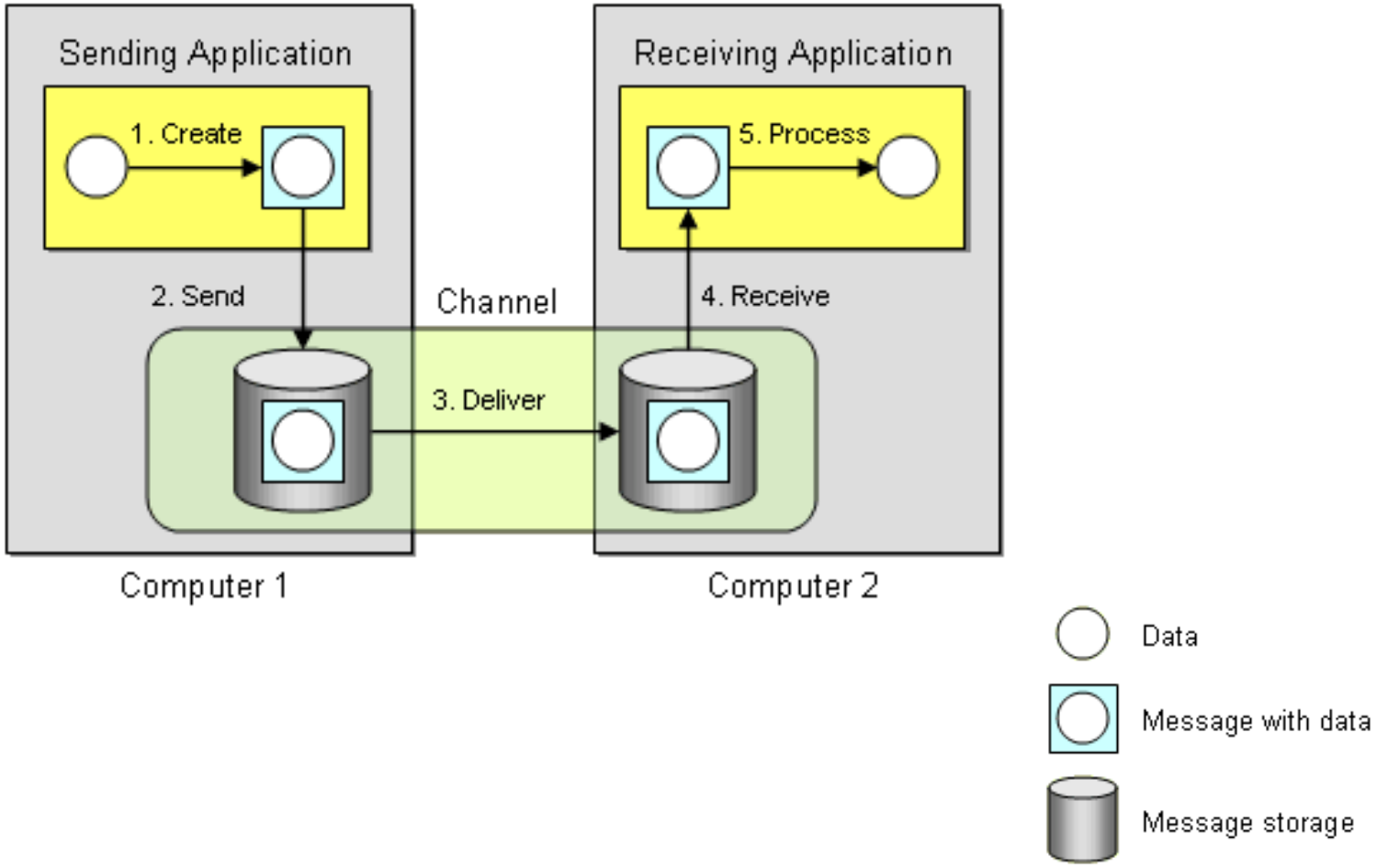
Sistemas de Mensajería o Message Oriented Middleware



Message Oriented Middleware



Sistemas de Mensajería o Message Oriented Middleware



Entrega de mensajes

- **1. Crear** - El emisor crea un mensaje y coloca en el los datos que desea transmitir.
- **2. Enviar** - El emisor coloca el mensaje en el canal.
- **3. Entregar** - El sistema de mensajería transporta el mensaje logrando que esté disponible para el receptor.
- **4. Recibir** - El receptor lee el mensaje desde el canal.
- **5. Procesar** - El receptor extrae los datos del mensaje.

Sistemas de mensajería

- La razón de usar un sistema de mensajería es que las computadores y las redes que las conectan no son confiables.
- Que una aplicación esté lista para enviar, no significa que la otra esté lista para recibir.
- En casos donde ambas los estén, puede ser que la red no esté funcionando correctamente.
- Un sistema de mensajería intenta repetidamente enviar el mensaje hasta que este es recibido.

Conceptos importantes

- **Send and forget (paso 2)**
  - Una vez que la aplicación entrega el mensaje en el canal, puede seguir ejecutando dado que esta delegó la entrega del mensaje al MOM.
  - La aplicación confiará en que el receptor recibirá el mensaje, y no esperará hasta que esto ocurra.

- **Store and forward (paso 3)**
  - El MOM almacena el mensaje previo a reenviarlo al destinatario

Productos tipo MOM

- Los MOM no son un concepto nuevo
- Productos de MOM existen hace mucho tiempo
  - MSMQ
  - IBM WebSphere MQ
  - ActiveMQ
  - RabbitMQ
  - Oracle Advance Queuing

¿Qué brinda un MOM ?

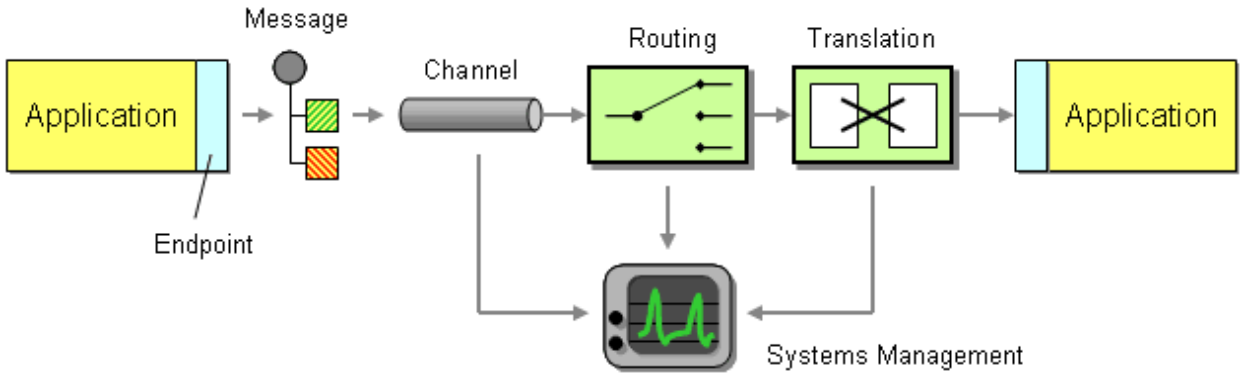
- En general
  - Garantía de entrega de mensajes
  - Comunicación asíncrona
  - Mensajería transaccional
  - Servicio básico de ruteo de mensajes
  - Servicios de notificación de entrega de mensajes

¿Por qué usar Mensajería ?

- Comunicación remota
- Integración entre plataformas/lenguajes
- Comunicación asincrónica
- Control sobre el ritmo de procesamiento
- Comunicación confiable
- Operación sin conexión (disconnected)
- Mediación por parte del sistema de mensajería
- …

Componentes de un MOM

- Mensajes
- Canales
- Endpoints
- Routers
- Translators
- System management



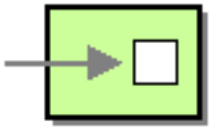
Mensajes

- Unidades discretas de datos
- Estructura
  - Header
    - Información sobre el mensaje
    - ID, Origen, Destino
    - Properties
      - Timestamp, correlationID, sequenceNumber…
  - Body
    - Datos transmitidos
    - Generalmente ignorados por el sistema de Mensajería

Canales

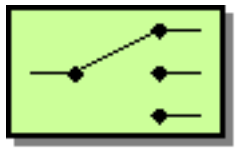
- Direcciones lógicas en el sistema de Mensajería
- Se diseñan considerando la intencionalidad del mensaje
- Generalmente son estáticos
  - Necesidad de acuerdo entre distintas aplicaciones

Endpoints

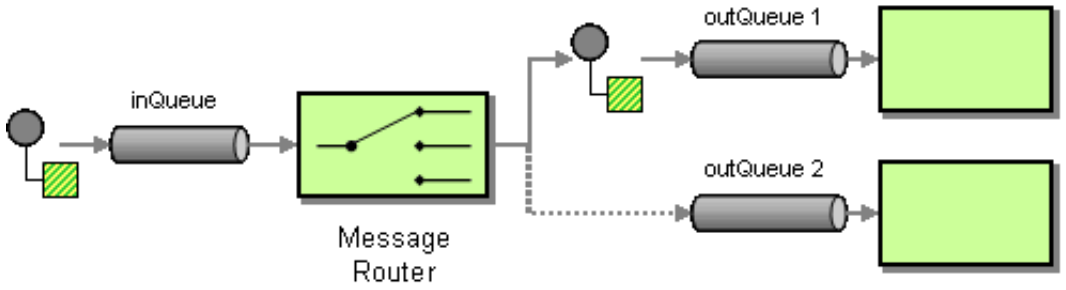


- Es el middleware que permite la conexión de la aplicación con el sistema de mensajería
- Para mandar mensajes, se debe establecer una conexión, similar a una Base de Datos
- Ejemplos
  - Plataforma Java: JMS
  - Plataforma .Net: MSMQ

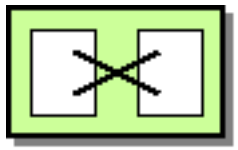
Routers



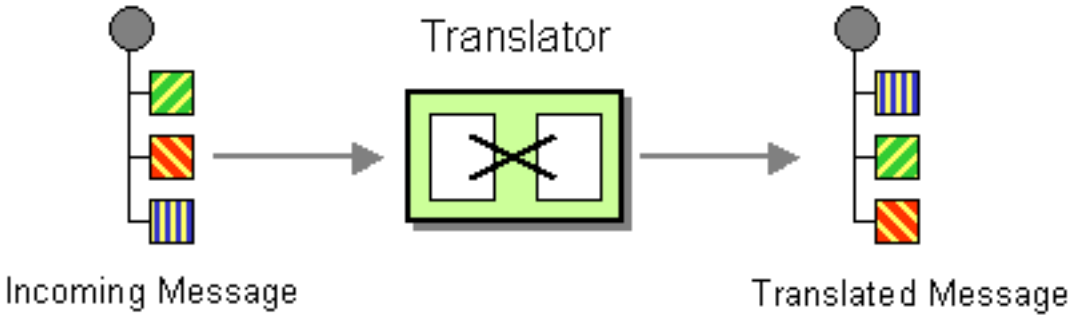
- Direccionan mensajes de un canal entrante a uno de salida
- Esto desacopla aún más al productor de conocer el destino final del mensaje
- Se puede tener una jerarquía de estos para obtener enrutamientos complejos



Translators



- ¿Cómo pueden los sistemas usando diferentes formatos de datos comunicarse unos con otros usando mensajería ?
- Se necesita tener un tipo especial de componente encargado de traducir de un formato a otro



System Management

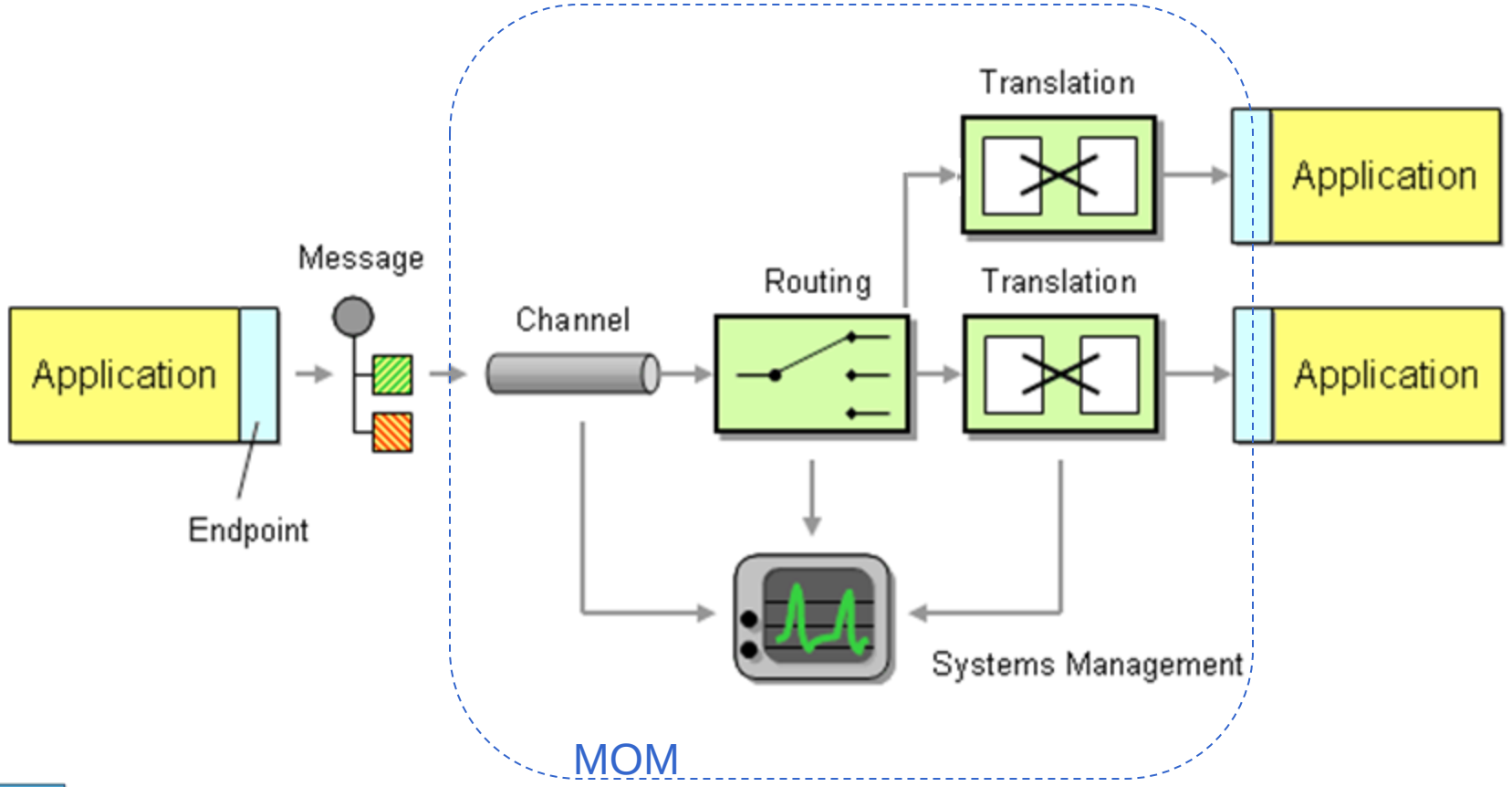
- ¿Cómo puedo monitorear, administrar, testear y analizar los mensajes en tránsito de un sistema de mensajería?

- El System management es un componente especial encargado de proveer las herramientas para llevar a cabo estas tareas

Escenario de uso

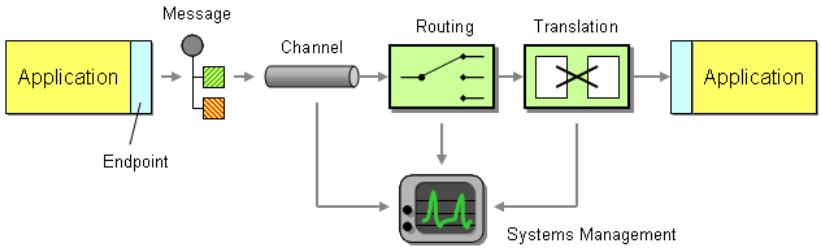
- Una aplicación debe comunicar información financiera a otra y se decide utilizar un MOM.
- La aplicación emisora envía la información empaquetada en un mensaje a un canal definido dentro del MOM
- Se requiere que todo mensaje con información sobre transacciones en ATMs se envíe al componente A y las transacciones Web se envíen al componente B de la aplicación.
- Previo al envío del mensaje a cada componente, es necesario transformar determinados datos del mensaje para que estos los comprendan.

Ejemplo básico



Patrones de Mensajería

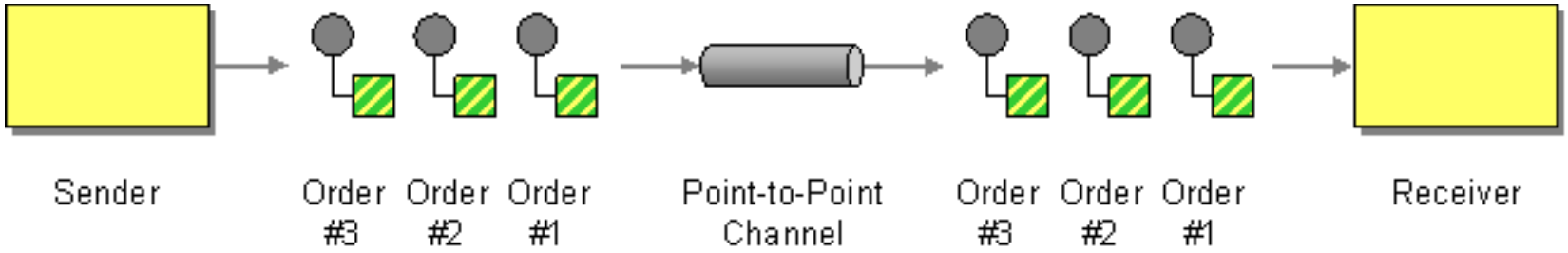
- Son “soluciones tipo” a problemas comunes en el ámbito de la EAI basada en mensajería
- Basados en los componentes de un MOM
- Categorías:
  - Message Channel Patterns
  - Message Construction Patterns
  - Message Routing Patterns
  - Message Transformation Patterns
  - Message Endpoint Patterns
  - System Management Patterns



Message Channel Patterns

- Aspectos a tener en cuenta a la hora de diseñar los canales
  - ¿Cuántas aplicaciones deben recibir el mensaje?
  - ¿El canal me garantiza la entrega del mensaje?
  - ¿Qué hacer con mensajes que no se pueden procesar correctamente?
  - ¿Qué hacer con mensajes que no se pueden entregar correctamente?

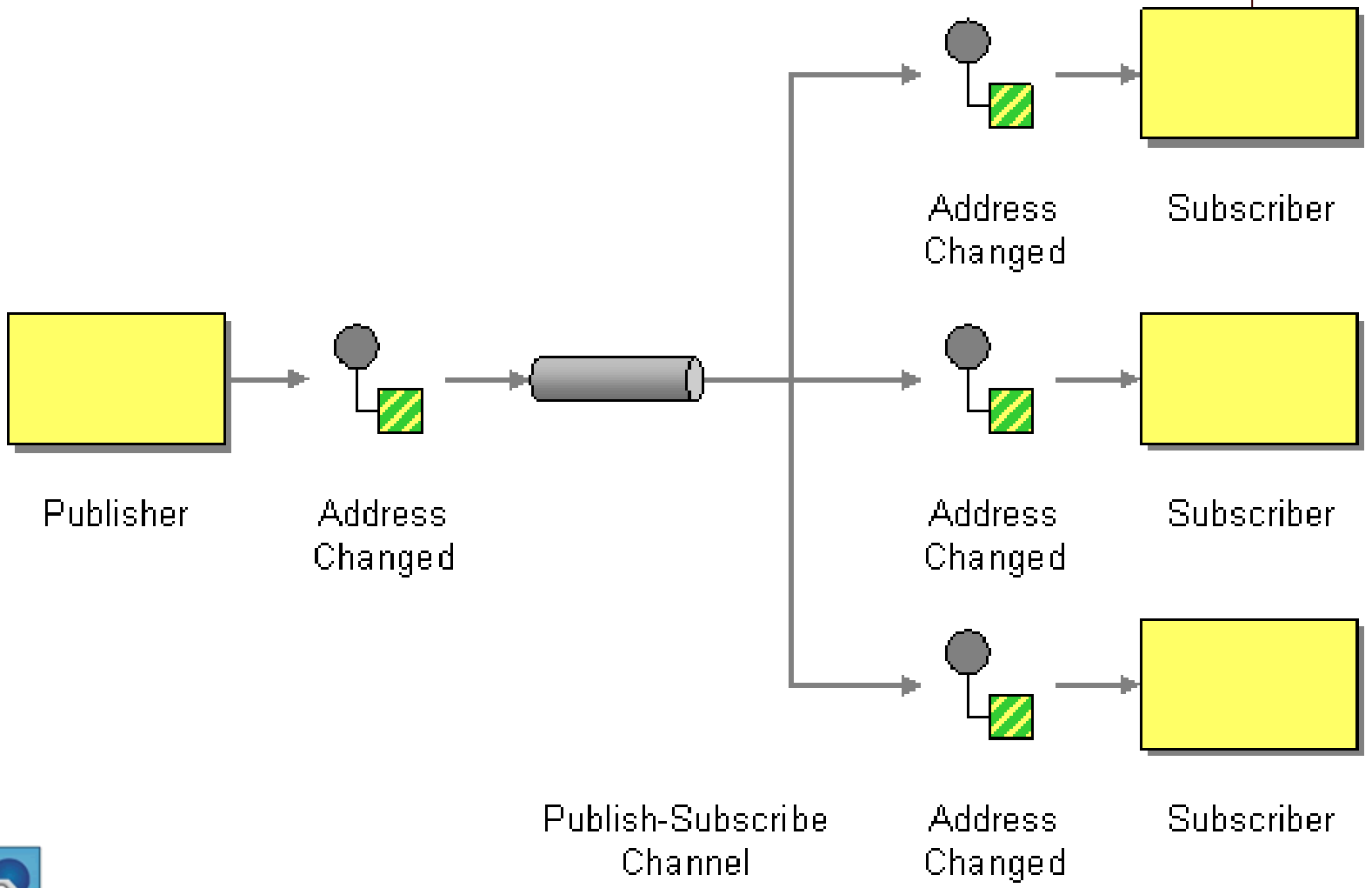
Point to Point Channel



Point to Point Channel

- Comunicación punto a punto
- Solamente un consumidor tomará el mensaje
- Se mantiene el orden de entrega

Publish-Suscribe Channel

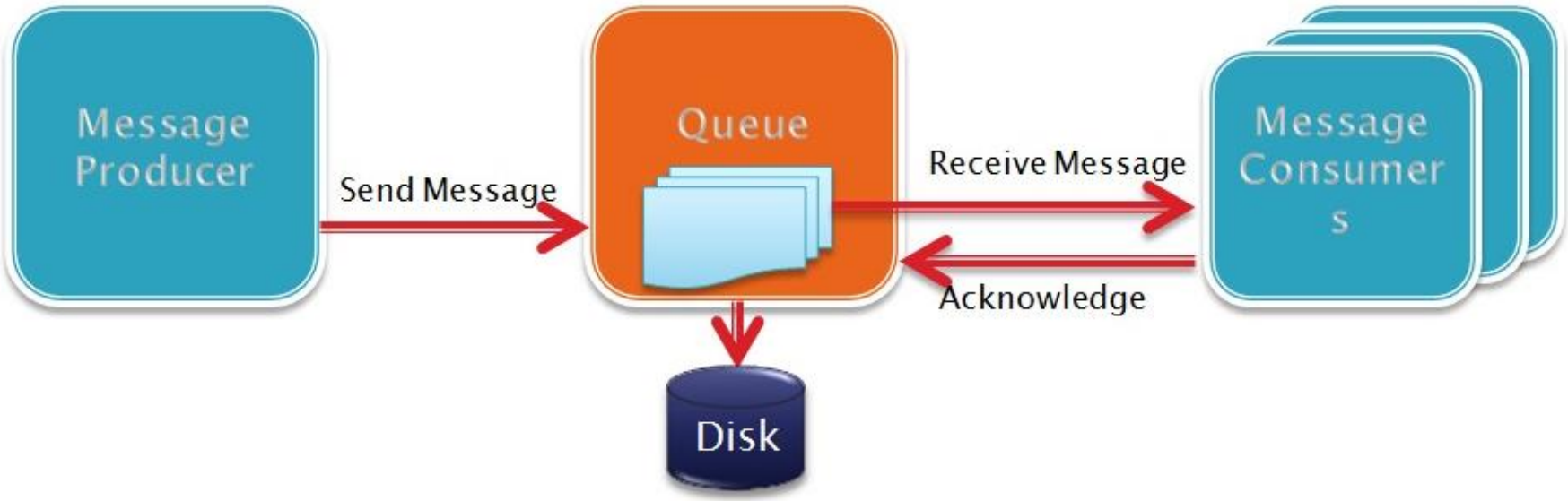


Publish-Suscribe Channel

- Existen uno o varios productores de mensajes
- El canal envía una copia del mensaje a todos los consumidores registrados (suscriptores)
  - Similar a Observer Pattern (OO)
- Se mantiene el orden de los mensajes

Guaranteed delivery

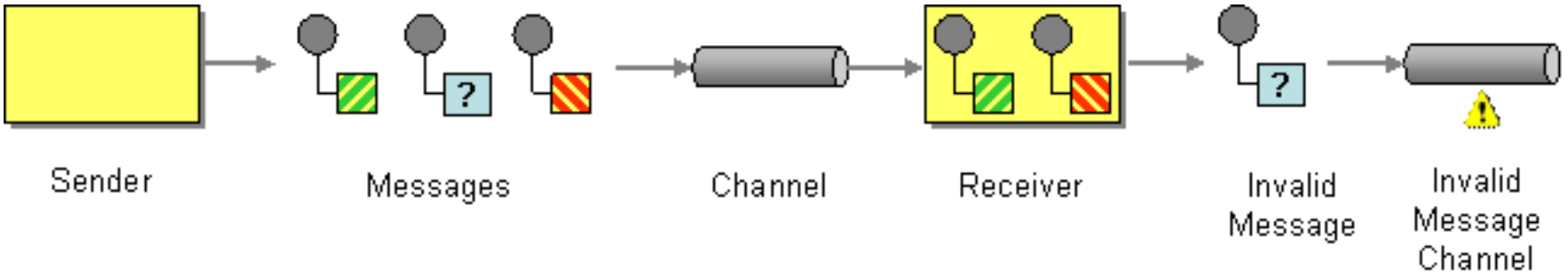
- ¿Cómo puede el emisor asegurarse el mensaje se entregará, incluso si el MOM falla?



Guaranteed delivery

- Persistencia de mensajes
  - Los mensajes se persisten a disco para prevenir la pérdida de mensajes frente a caídas del MOM
  - La operación de envío/recepción de mensaje no finaliza hasta que el mismo está correctamente almacenado.
    - Al enviarse un mensaje, primero se persiste en el MOM y luego se le confirma al emisor este fue recibido correctamente
    - Al retirarse un mensaje, primero se recibe la confirmación del receptor que lo recibió correctamente y luego se elimina el mensaje.
  - Menor procesamiento de mensajes por segundo
    - Menor performance!

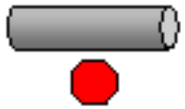
Invalid Message Channel

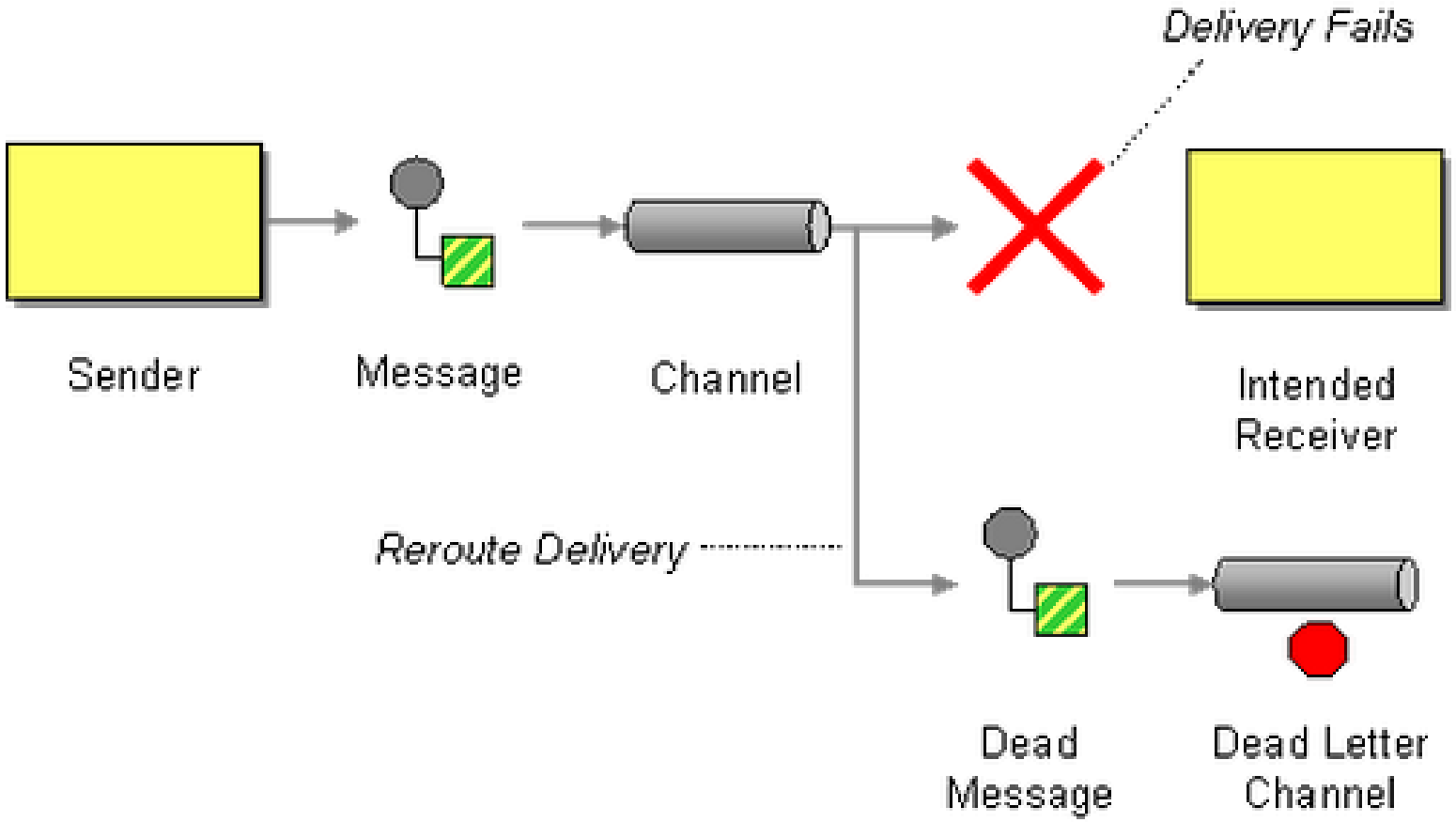


Invalid Message Channel

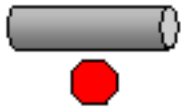
- Distintas situaciones pueden impedir el procesamiento de un mensaje
  - Datos faltantes, errores en la estructura, etc
- Enviando el mensaje a un canal especial, permite un posterior logging, debugging, monitoreo, alertas, etc.
  - Recordar asincronismo!

Dead Letter Channel





Dead Letter Channel



- Algunos mensajes no se pueden entregar
  - Receptor no disponible o expiración del mensaje
- Para no mantenerlos en colas en producción se mueven a un canal especial
  - También conocido como “dead letter queue”

Escenario de uso

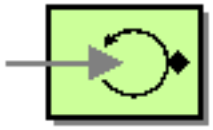
- Una aplicación desea notificar a un conjunto de aplicaciones la creación de un cliente
- El middleware debe filtrar aquellas notificaciones con información no válida.
- Las notificaciones que no pudieron ser enviadas luego de un tiempo X, se deben almacenar para posterior análisis
- El sistema debe ser tolerante a caídas o fallas. No se deben perder mensajes

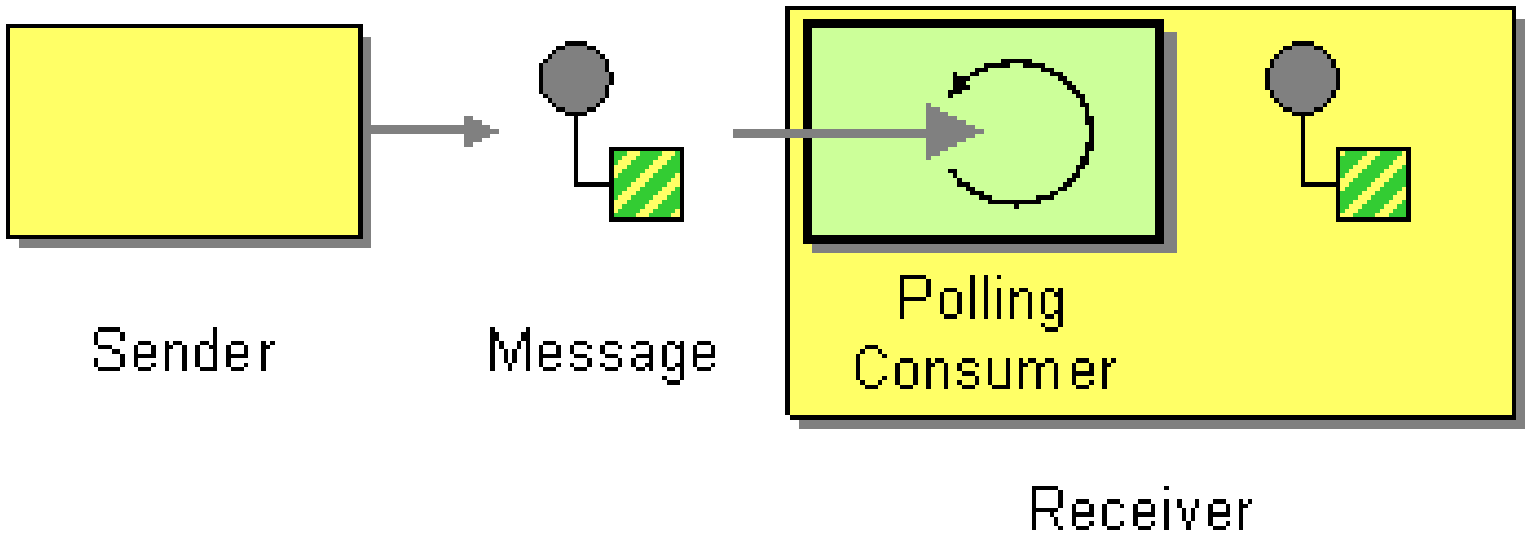
Message Endpoints Patterns

Message Endpoints Patterns

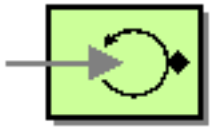
- Las aplicaciones intercambian información mediante el envío de mensajes a través de canales
- Posibilidad de diferentes patrones de retiro de mensajes:
  - Suscripción sincrónica o “Polling consumers”
  - Suscripción asincrónica o “Event-driven consumers”
  - Durable suscriber
  - Competing consumers
  - Transactional client

Polling Consumers





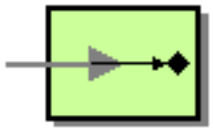
Polling Consumers

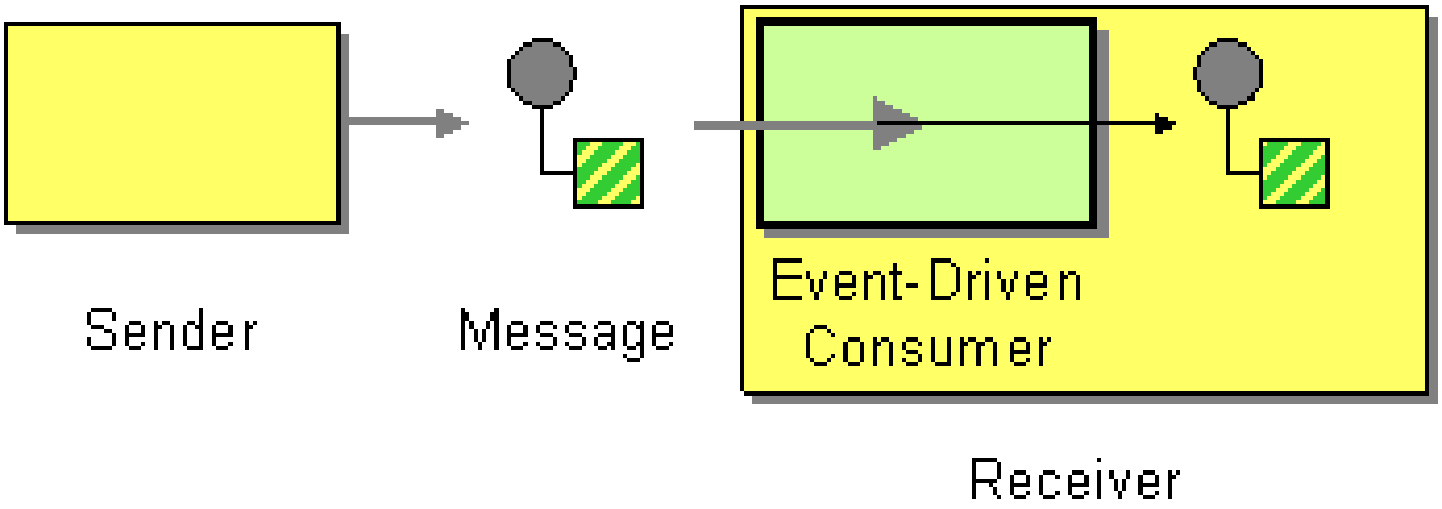


- Se desea tener un control estricto de los mensajes que se van consumiendo
  - Ej: capacidad de procesamiento limitada

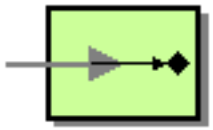
- Se hace la solicitud de lectura de mensajes y se espera hasta que haya uno disponible
  - El receptor bloquea su thread
  - Es posible definir un tiempo máximo de espera

Event Driven Consumer





Event Driven Consumer

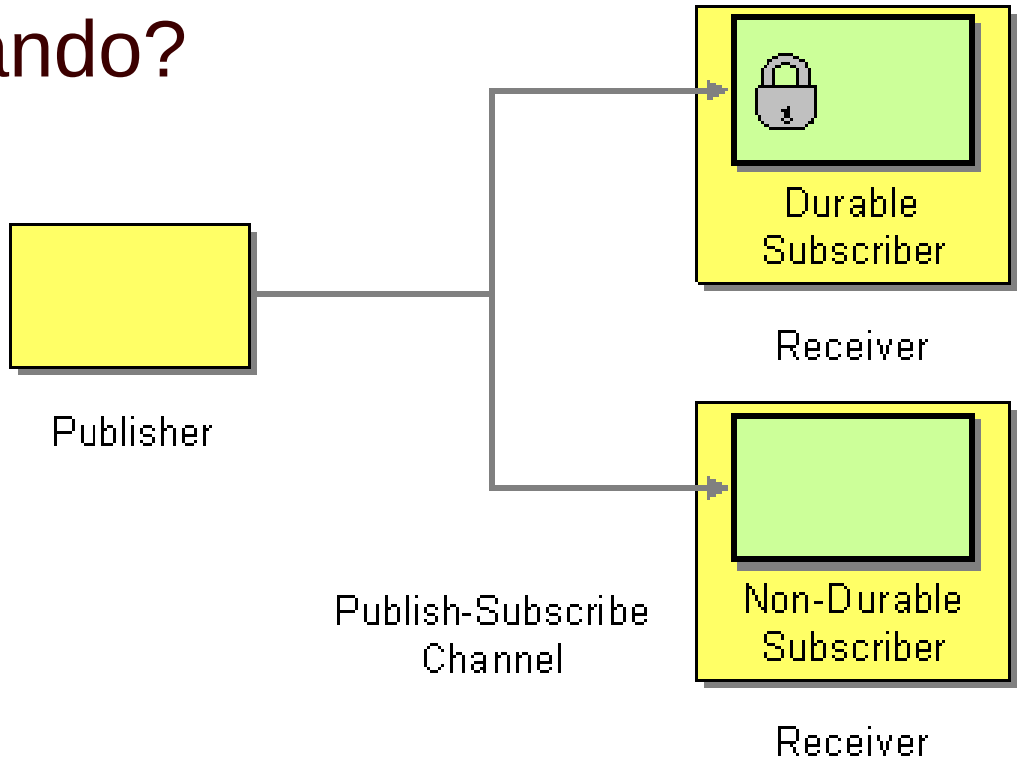


- El receptor se suscribe al canal y le indica una dirección a dónde se deben enviar los mensajes

- Cada vez que llega un mensaje, el canal lo envía a la dirección definida por el suscriptor.
  - No es necesario mantener recursos mientras no se están procesando mensajes

Durable suscriber

- ¿Cómo puede hacer un event-driven consumer para evitar perder mensajes cuando no está escuchando?



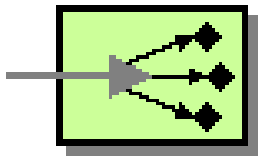
Durable suscriber

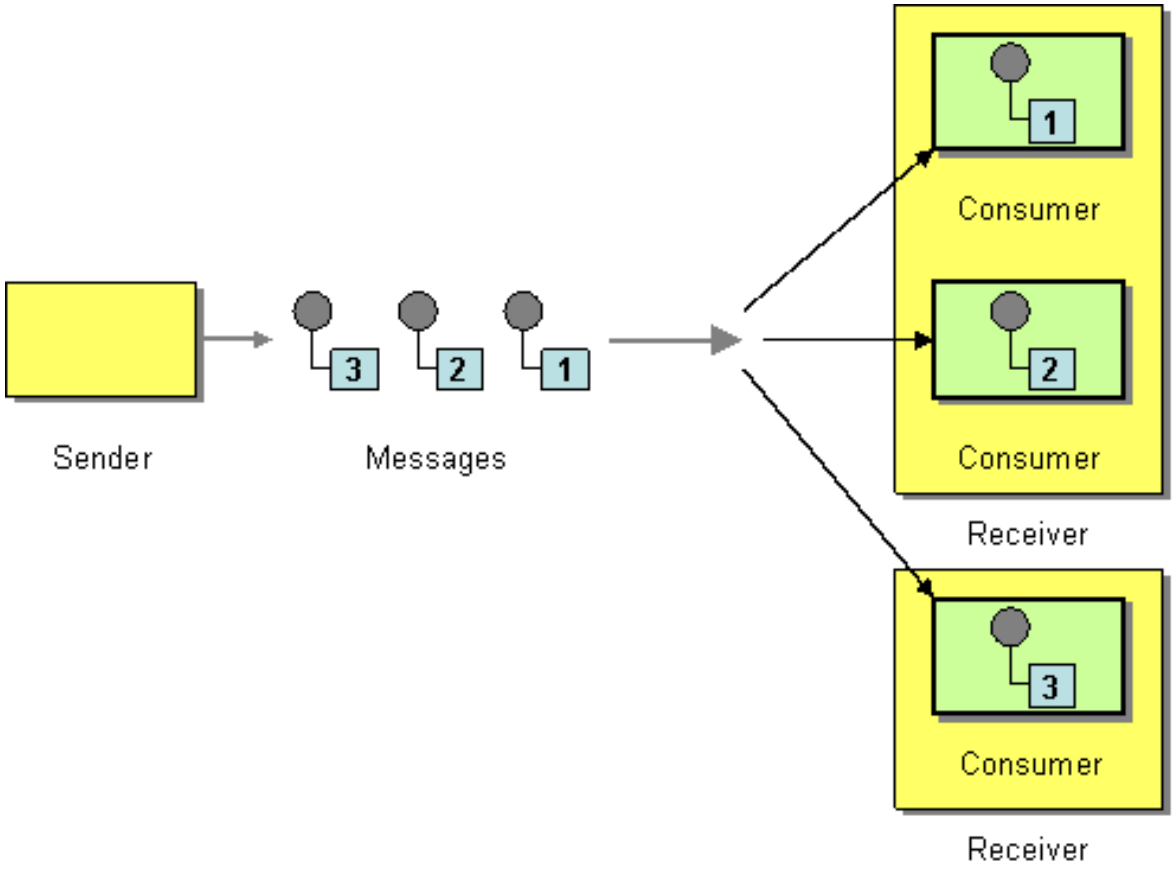
- Solo aplica a canales tipo Publish/Subscribe
- El consumidor debe suscribirse al canal como un suscriptor durable.
- Las suscripciones durables guardan los mensajes en caso de que el suscriptor no esté disponible.
- Los mensajes son enviados cuando el suscriptor está nuevamente en línea.
- No tiene efecto si el suscriptor está en línea
- Determina el comportamiento del MOM

Durable suscriber

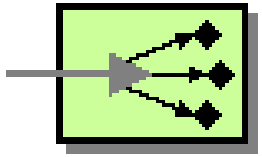
- ¿Por qué no hay suscripción durable en los canales punto a punto?

Competing Consumers



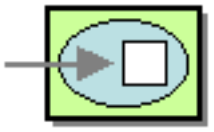


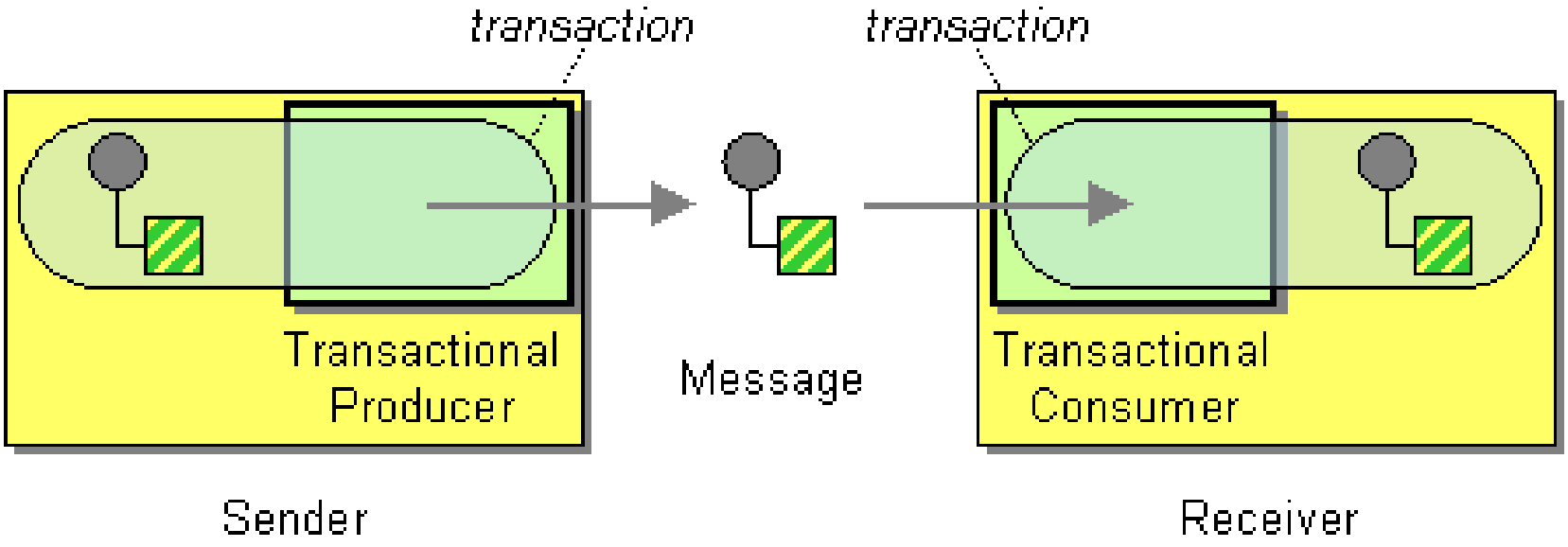
Competing Consumers



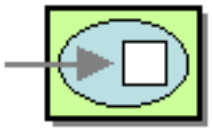
- Se definen varios consumidores por canal
  - Cada consumidor toma un mensaje
  - Aplica solo a canales Point-to-Point
- Procesamiento paralelo de mensajes
  - No importa orden de procesamiento
- Mayor escalabilidad!

Transactional Client



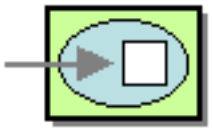


Transactional Client



- Atomicidad en publicación/lectura de mensajes
- Definición de transacciones de mensajes
  - No confundir con transacciones de base de datos!
  - El mensaje no se marca como publicado/leído hasta que el publicador/receptor no confirma publicación/recepción.
  - Ante un error se hace el “rollback” de la transacción y se retorna el mensaje al canal
- Algunos sistemas “conectan” transacciones ACID con transacciones de mensajes
  - Ej: Transacciones XA: JMS y JDBC

Transactional Client



- Ejemplo de uso:
  - Se retira un mensaje y se envía a un Web Service
  - La transacción implica:
    1. Retirar el mensaje
    2. Invocar al Web Service
    3. Confirmar o Rollback
  - Si la invocación al Web Service es exitosa, el mensaje se marca como leído y se retira del canal
  - En caso de error, el mensaje no se marca como leído ni se retira del canal
    - Útil para reintentos ante caídas de Web Service

Message Construction Patterns

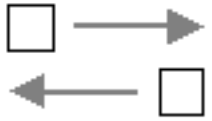
Message Construction Patterns

- Son patrones que guían cómo crear los mensajes y qué propiedades deben tener
- Se debe tener en cuenta:
  - Intención del mensaje
  - Tipo de interacción
  - Posible recepción tardía del mensaje

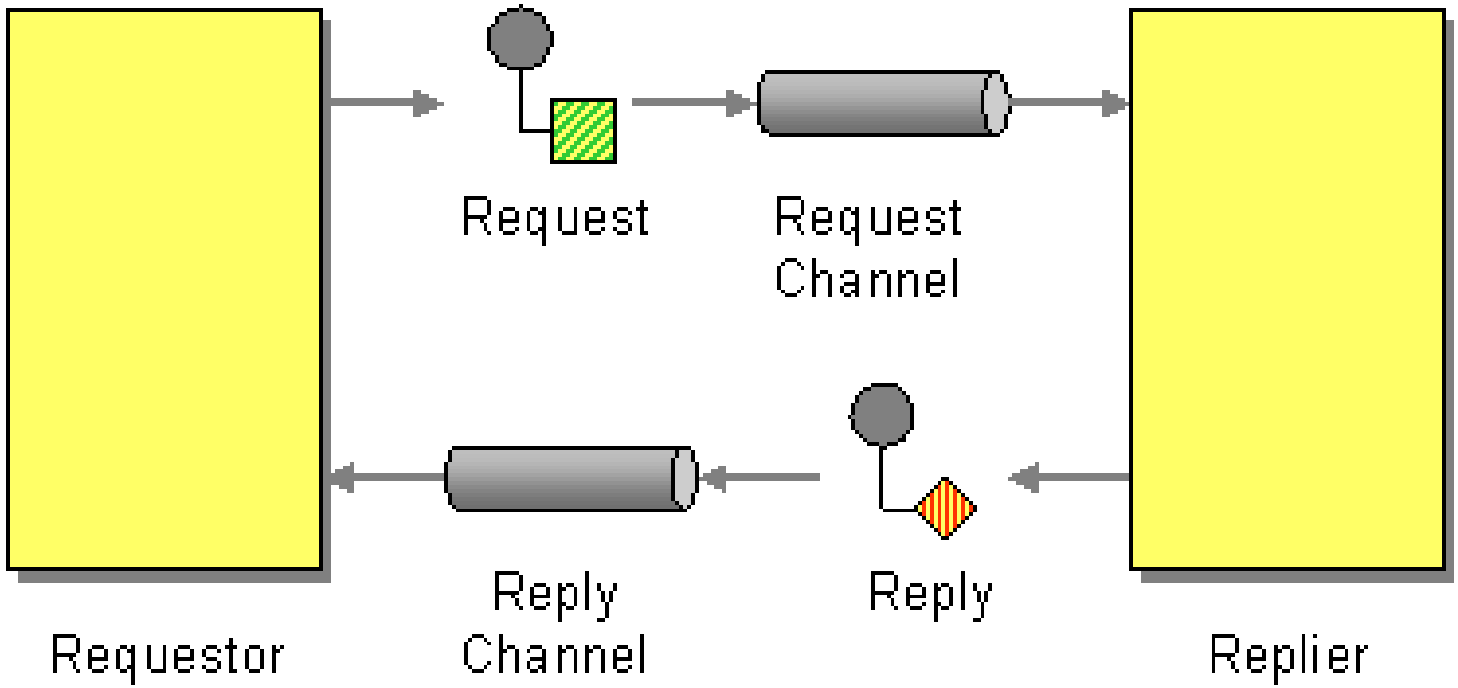
Intención del mensaje

- El emisor puede tener diferentes intenciones de qué debe hacer el receptor con el mensaje.
  - Ejecución de un comando (Command message)
    - Similar a RPC pero con mensajería
    - Ej: Obtener la información de un cliente/Comenzar el procesamiento batch
  - Envío de información (Document message)
    - Ej: información de un cliente
  - Notificación de un evento (Event message)
    - Ej: se creó un nuevo cliente

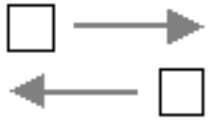
Interacción: Request/Reply



- ¿Cuando una aplicación manda un mensaje cómo hace para esperar la respuesta?



Interacción: Request/Reply

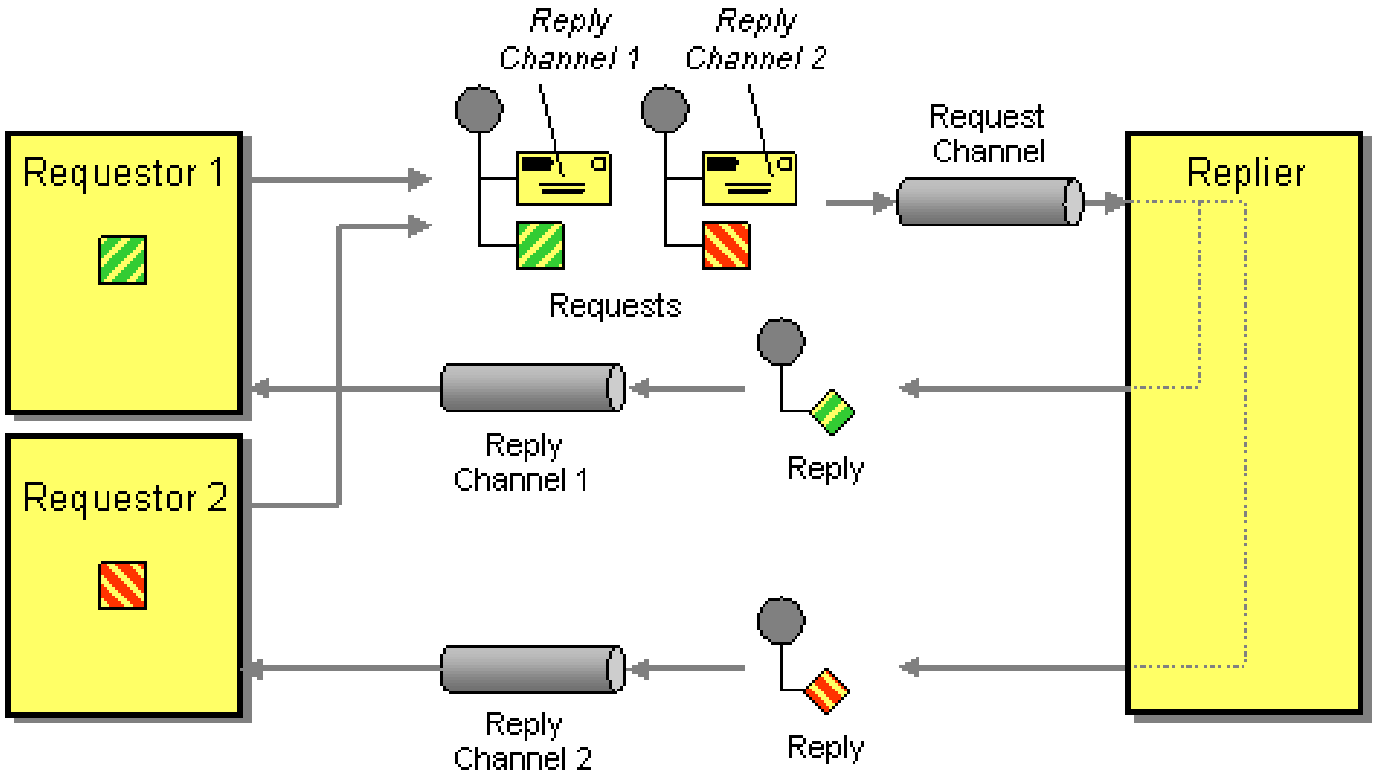


- Se utilizan dos canales
  - Un canal request para enviar el mensaje de solicitud
  - Un canal response para recibir la respuesta

- Los canales pueden ser de cualquiera de los tipos mencionados anteriormente:
  - Point-to-Point, Pub/Sub, Datatype, etc.

Return Address

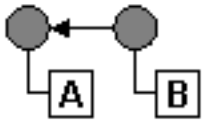
- ¿Cómo sabe el receptor de un mensaje a qué canal enviar el mensaje de respuesta?



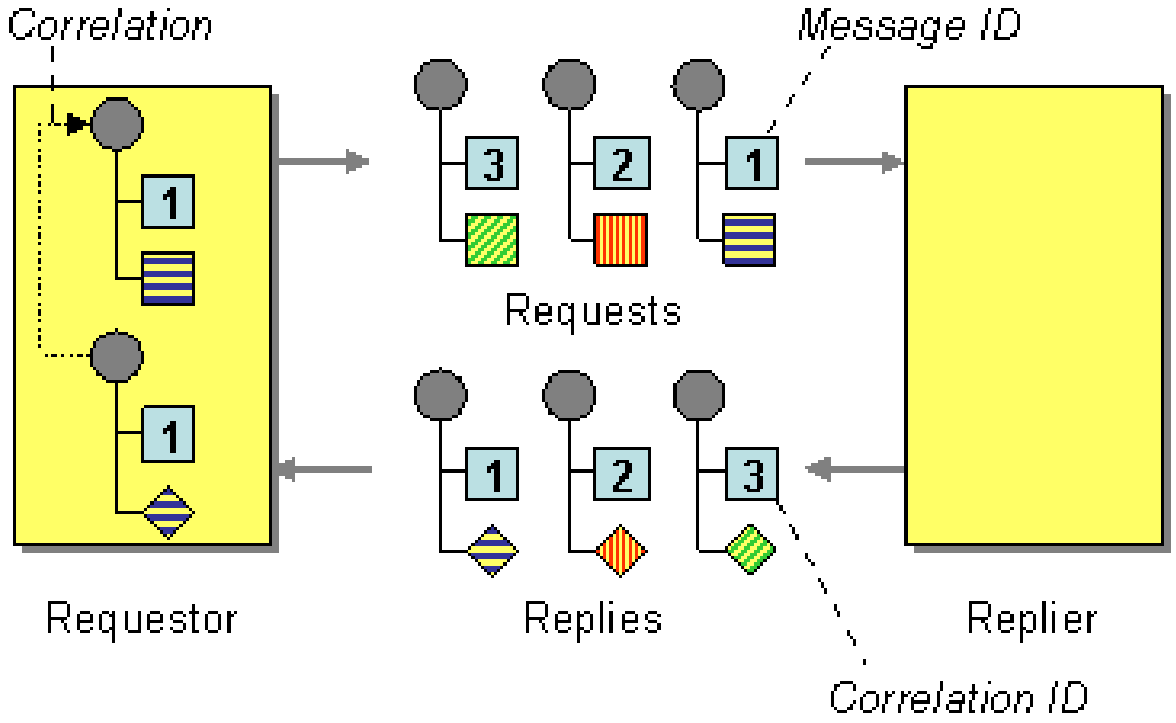
Return Address

- El mensaje debe tener una “Return Address” que le indique al receptor a qué canal enviar los mensajes con la respuesta
- No tiene porqué ser un canal con la dirección del emisor
- El canal de respuesta se determina dinámicamente en función de los datos del emisor
- La información de “Return Address” se coloca en los cabezales de los mensajes.

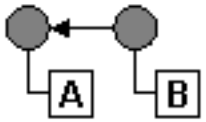
Correlation Identifier



- ¿Al obtener un mensaje de respuesta cómo sabemos a que mensajes de solicitud se corresponde?



Correlation Identifier

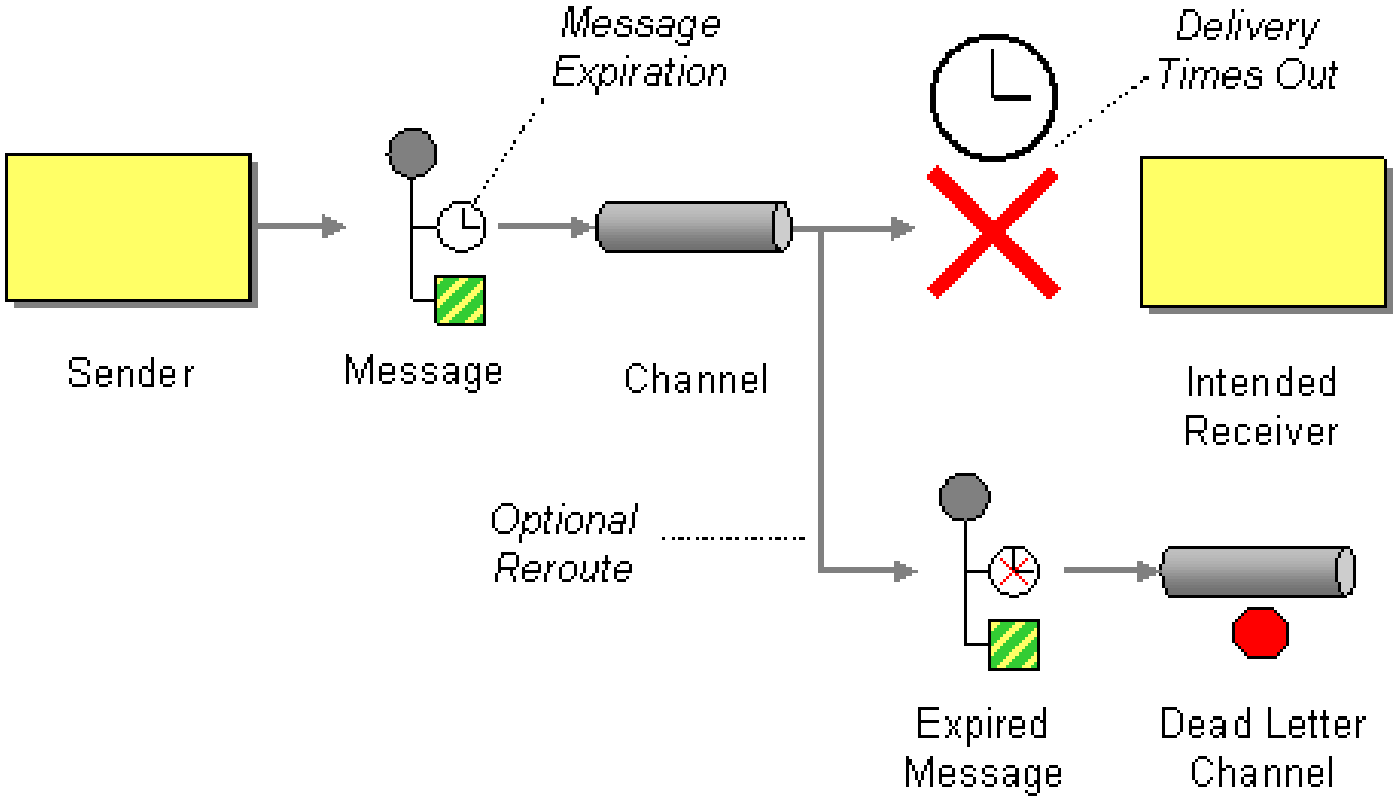


- Cada mensaje de solicitud debe tener un identificador que lo identifica de forma unívoca

- Cada mensaje de respuesta debe tener un:
  - Identificador del mensaje
  - Un correlation id indicando a qué solicitud corresponde

Message Expiration

- ¿Como hace el emisor para indicar el tiempo de vida en un mensaje?



Message Expiration

- Se define en el mensaje un tiempo de límite por el cual se está dispuesto a esperar para que sea procesado.
  - Si se usa persistencia de mensajes, permite liberar espacio en disco por mensajes obsoletos.
- Los mensajes pueden ser enviados a una dead letter channel

Escenario

- El sistema A provee diferentes informes a otros sistemas de la organización, los cuales requieren de tiempo prolongado de CPU para generarse. Este sistema ofrece un canal para recibir solicitudes de informe.

- Los sistemas B y C solicitan diariamente varios informes al sistema A y cada uno provee un canal para recibirlos

Escenario

- Identificar tipos de mensajes enviados entre los sistemas
- Tipo de interacción
- Canales involucrados en la solución

Message Routing Patterns

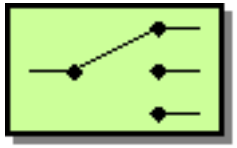
Message Routing Patterns

- Tres categorías:
  - Simple Routers
    - Rutean mensajes desde un canal entrante a uno o más canales de salida
  - Composed Routers
    - Combinan múltiples Simple Routers para crear flujos de mensajes más complejos
  - Architectural Patterns
    - Describen estilos arquitectónicos basados en Messages Routers
    - No lo veremos en este curso

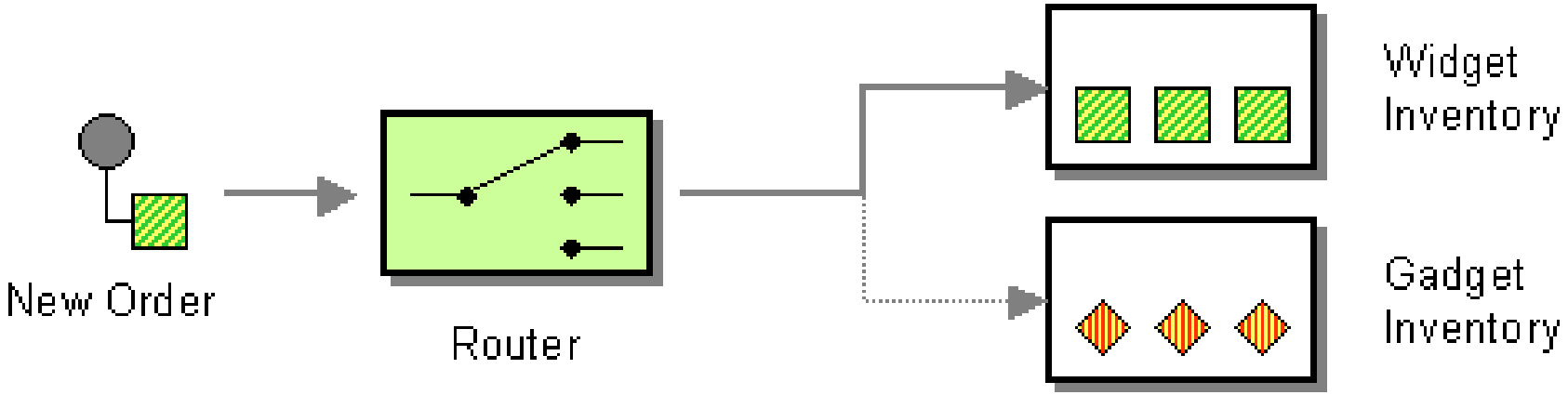
Simple Routers

- Content-based router
- Message filter
- Recipient list
- Splitter
- Aggregator

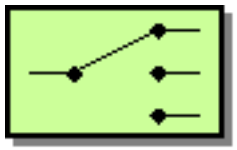
Content-Based Router



- ¿Cómo hacer para manejar situaciones donde la lógica de negocio está distribuida en varios sistemas?
  - Ej: la consulta de inventario se consulta en varios sistemas producto de la fusión de dos empresas



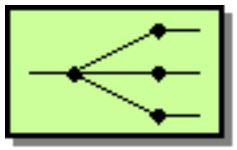
Content-Based Router



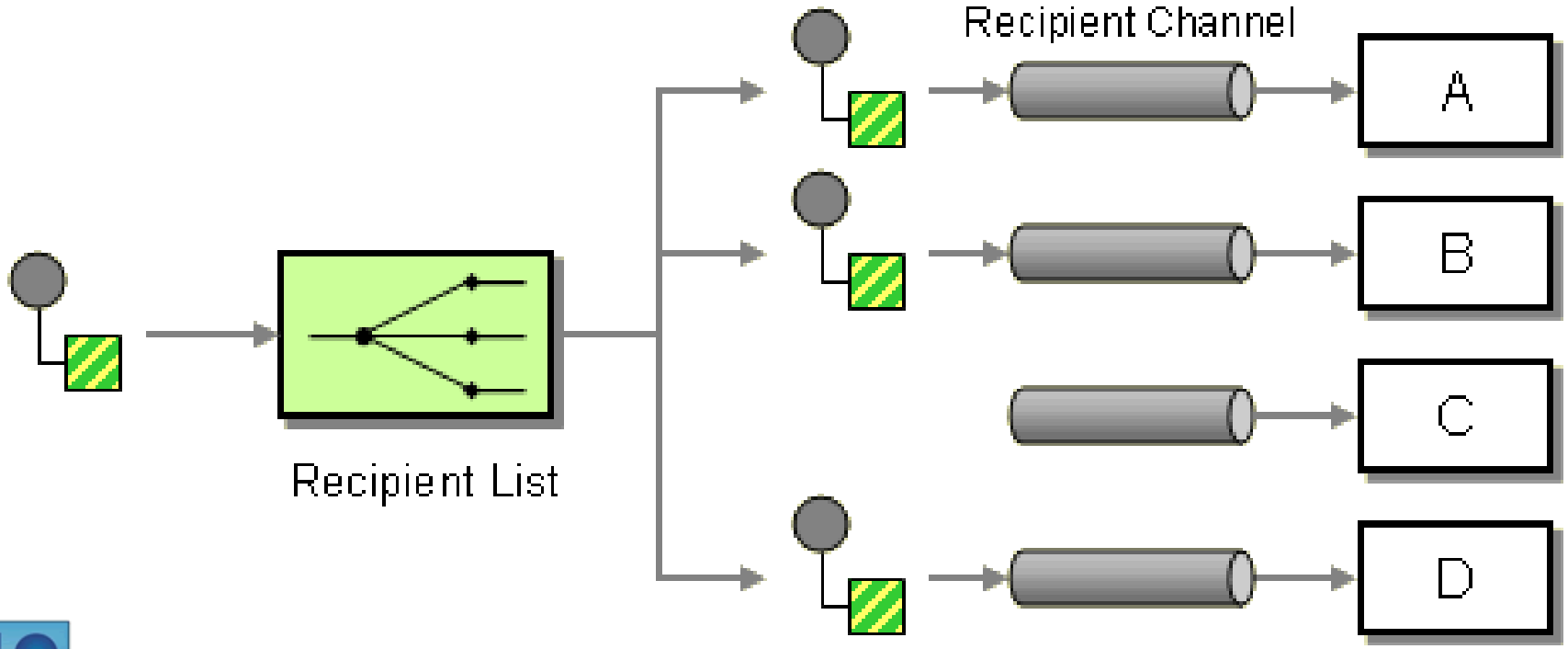
- Permite rutear mensajes a un canal destino
- La decisión se toma a partir del contenido del mensaje
  - Se examinan el cuerpo o los cabezales del mensaje
- No se modifica el contenido del mensaje

- En escenarios más complejos se puede hacer pasar el mensaje por un motor de reglas

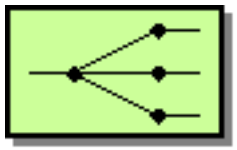
Recipient List



- ¿Cómo determinar dinámicamente el envío de mensajes a una lista de destinatarios?



Recipient List



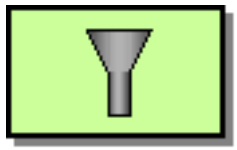
- Este Router tiene varios canales de salida
- Cada canal tiene asociada una condición
- Si el mensaje cumple con la condición asociada al canal, se le envía una copia del mensaje
  - No se modifica el mensaje

- ¿Qué diferencia hay con el canal Pub/Sub?

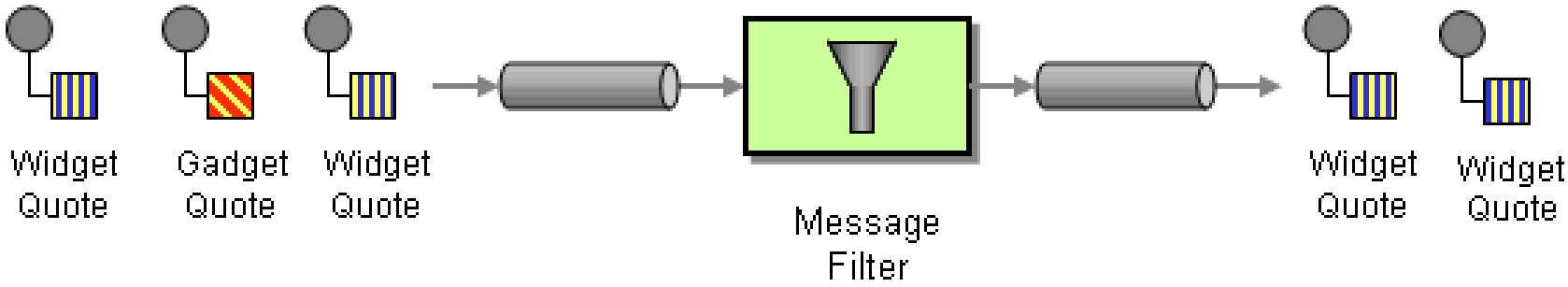
Recipient List vs Publish/Subscribe

Recipient List	Publish/Subscribe con filtros
Control y mantenimiento centralizado. Ruteo predictivo	Control y mantenimiento distribuido Filtrado reactivo
Router necesita conocer los destinatarios Se necesita actualizar el router para agregar o quitar destinatarios	No se necesita conocer los destinatarios. Agregar o quitar un destinatario es sencillo.
Utilizado en transacciones de negocios (solicitud de información)	Utilizado para notificar eventos o información (document o event messages)

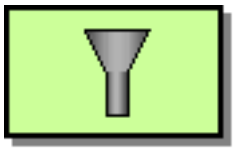
Message Filter



- ¿Cómo un receptor puede descartar mensajes que no le interesa recibir?
  - Ej: un cliente no está interesado en recibir novedades sobre determinados productos

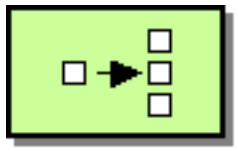


Message Filter

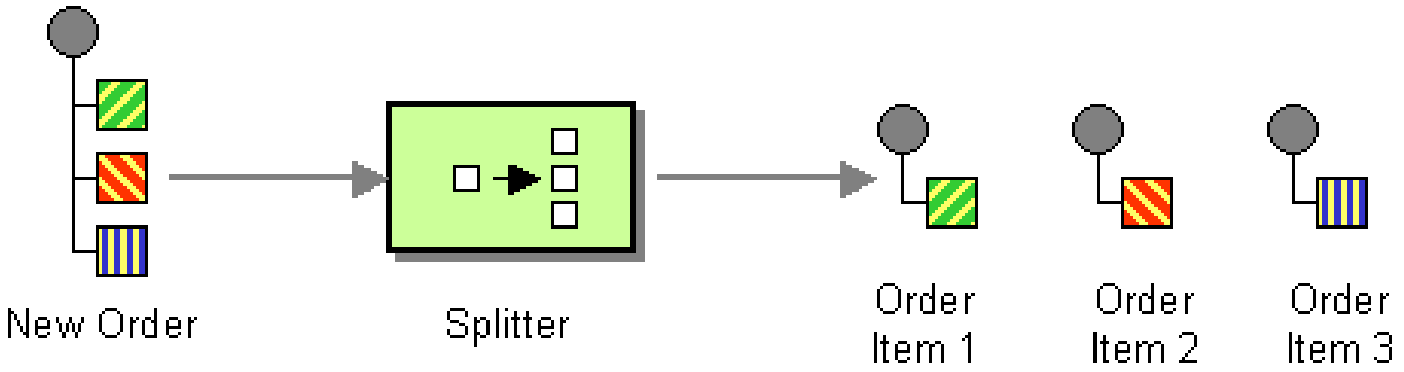


- El Message Filter tiene un solo canal de salida
- Los mensajes que cumplen con un determinado criterio son enviados por dicho canal
- Los mensajes que no cumplen con la condición son descartados
  - Pueden ser enviados a un canal especial de mensajes descartados para su posterior procesamiento o análisis.

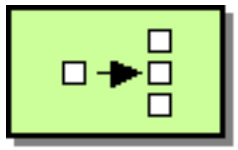
Splitter



- ¿Cómo podemos procesar un mensaje si éste contiene múltiples elementos que pueden ser procesados de diferente forma?

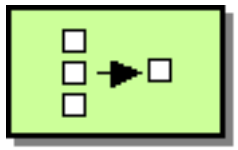


Splitter

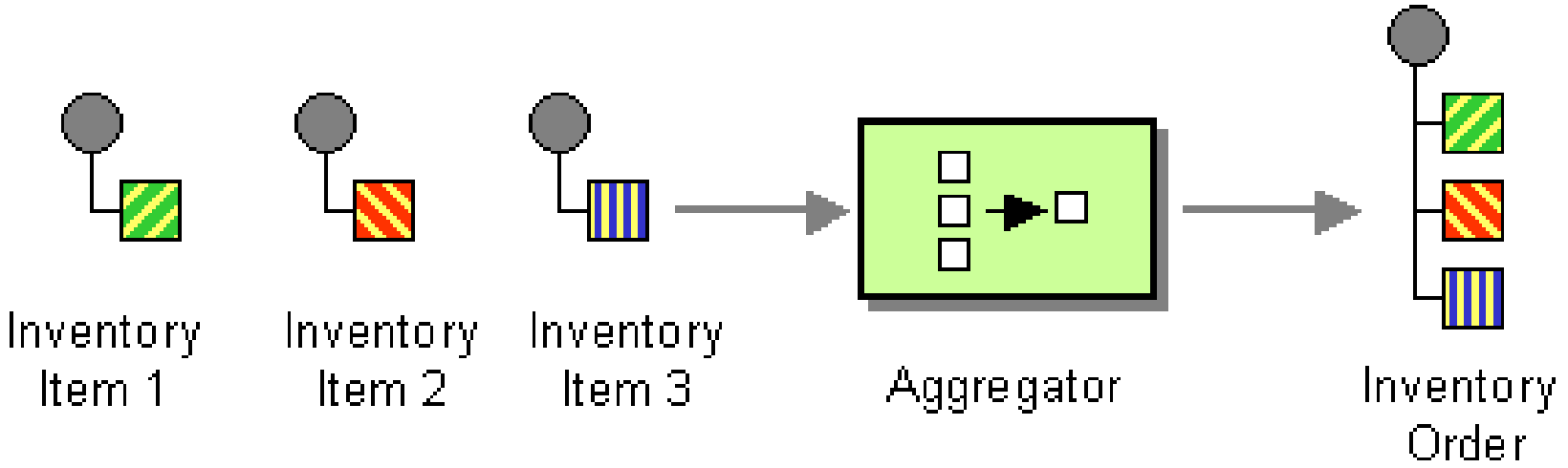


- Un mensaje puede estar compuesto por elementos que deben ser procesados por diferentes sistemas.
  - Ej: una solicitud de compra de varios artículos a proveer por diferentes sistemas.
- Se necesita procesar todo el mensaje, pero tratando a cada parte de forma individual
- El componente Splitter permite descomponer el mensaje en varios mensajes
  - Cada mensaje con una porción del mensaje original
  - Cada mensaje se procesa luego de forma individual

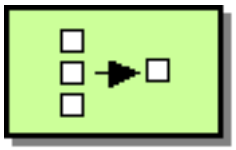
Aggregator



- ¿Cómo combinar varios mensajes individuales que tienen sentido en su conjunto?

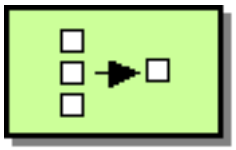


Aggregator



- En determinados escenarios es necesario agrupar un conjunto de mensajes
  - Ej: recolectar todas las ofertas para un determinado producto.
- Es un componente con estado que agrupa y almacena mensajes individuales hasta completar un cierto conjunto
  - CorrelationId para agrupar mensajes
  - Condición para liberar el grupo
- Genera un nuevo mensaje compuesto por el grupo de mensajes correlacionados y lo publica en un canal dado

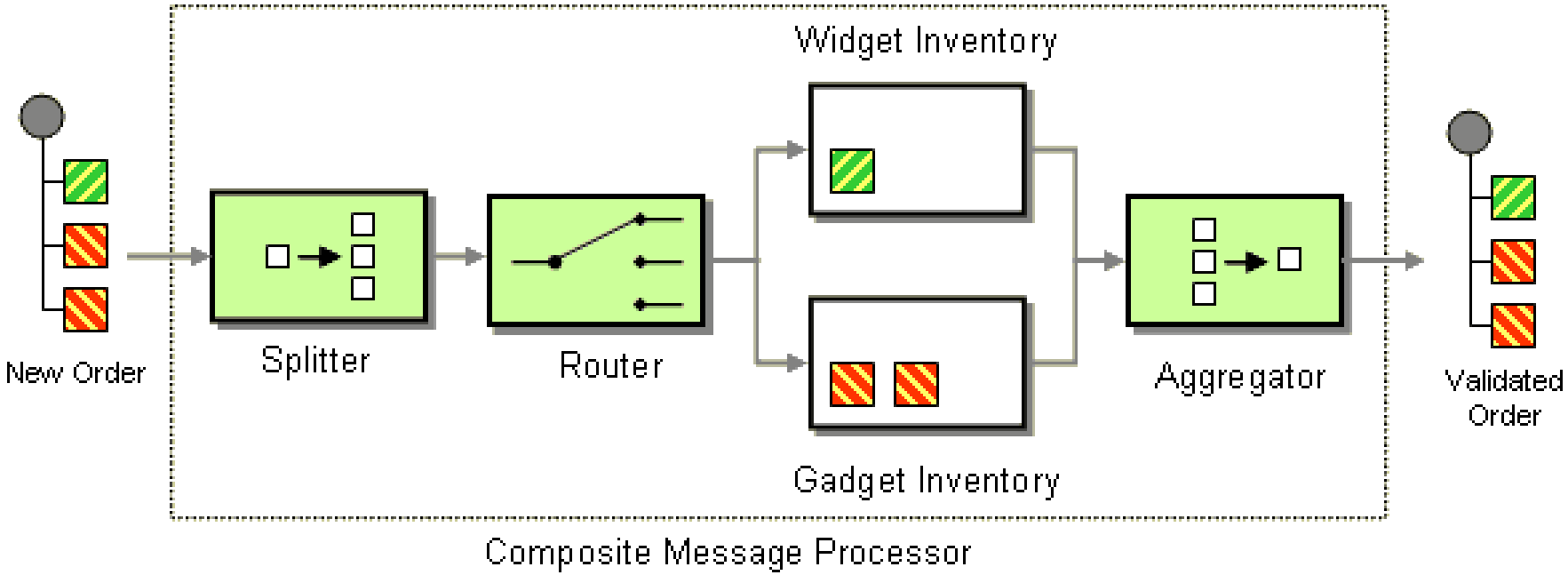
Aggregator



- ¿Qué patrón visto previamente es candidato a componerse con el patrón aggregator?

- La combinación del patrón Message Sequencer y Aggregator permitiría fácilmente desarrollo el intercambio de un gran volumen de información.
- La combinación con el patrón Splitter permitiría distribuir los mensajes y luego agruparlos obteniendo el resultado final.

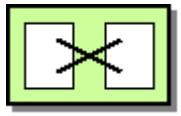
Composed routers



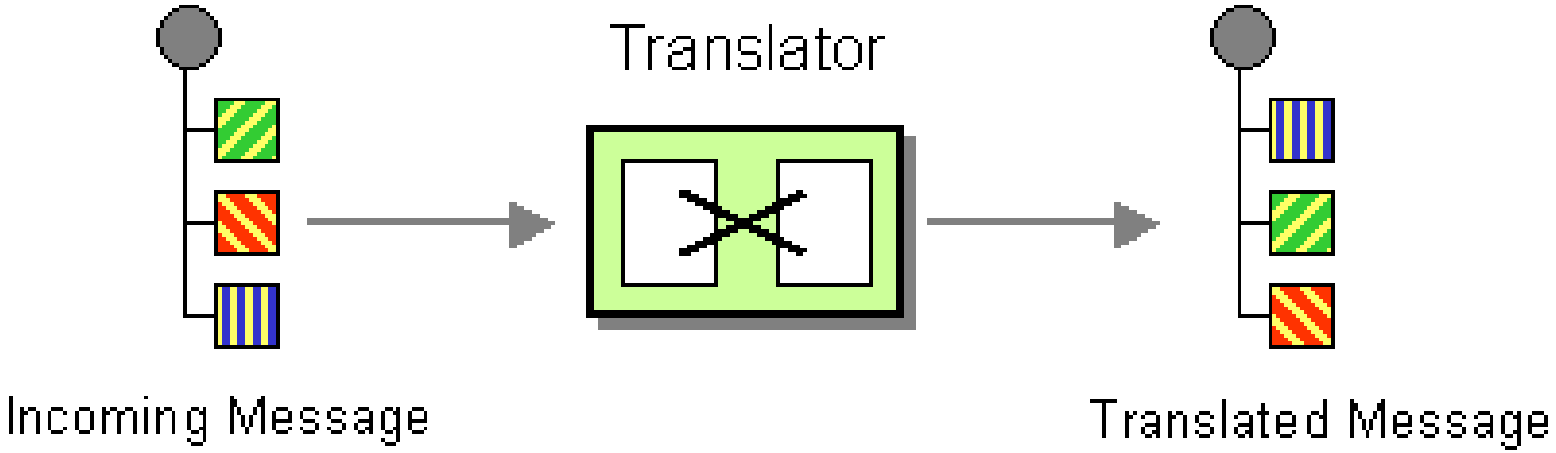
Transformation Patterns

- Aplicaciones con diferentes formatos de datos
  - Datos en XML, en archivos planos, JSON, etc
- Modificar las aplicaciones existentes no siempre es posible
  - El middleware debe conciliar las diferencias!
- Patterns:
  - Message Translators
  - Content Enricher
  - Content Filter

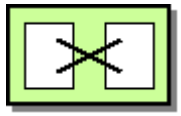
Message Translator



- ¿Cómo dos aplicaciones pueden comunicarse si utilizan diferente formato de mensaje?

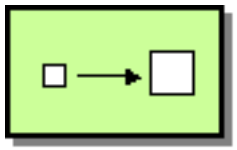


Message Translator

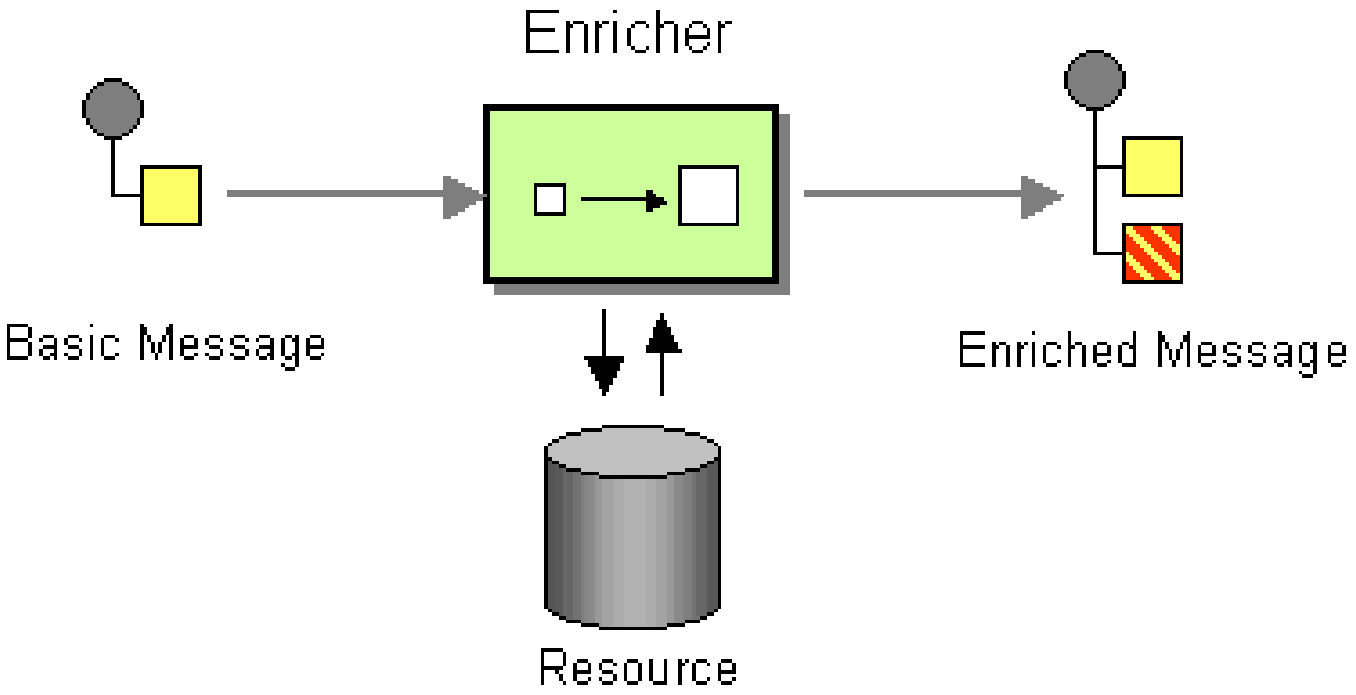


- Transformación de mensajes XML
  - XPath Translators
  - XSLT
- Transformación de objetos a XML y viceversa
- Transformación de objetos a JSON y viceversa
- Transformación de objetos a objetos
- ...

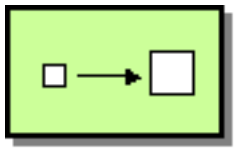
Content Enricher



- ¿Cómo nos comunicamos con otro sistema si el mensaje original no contiene todos los datos requeridos?

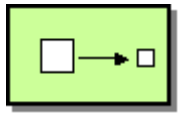


Content Enricher

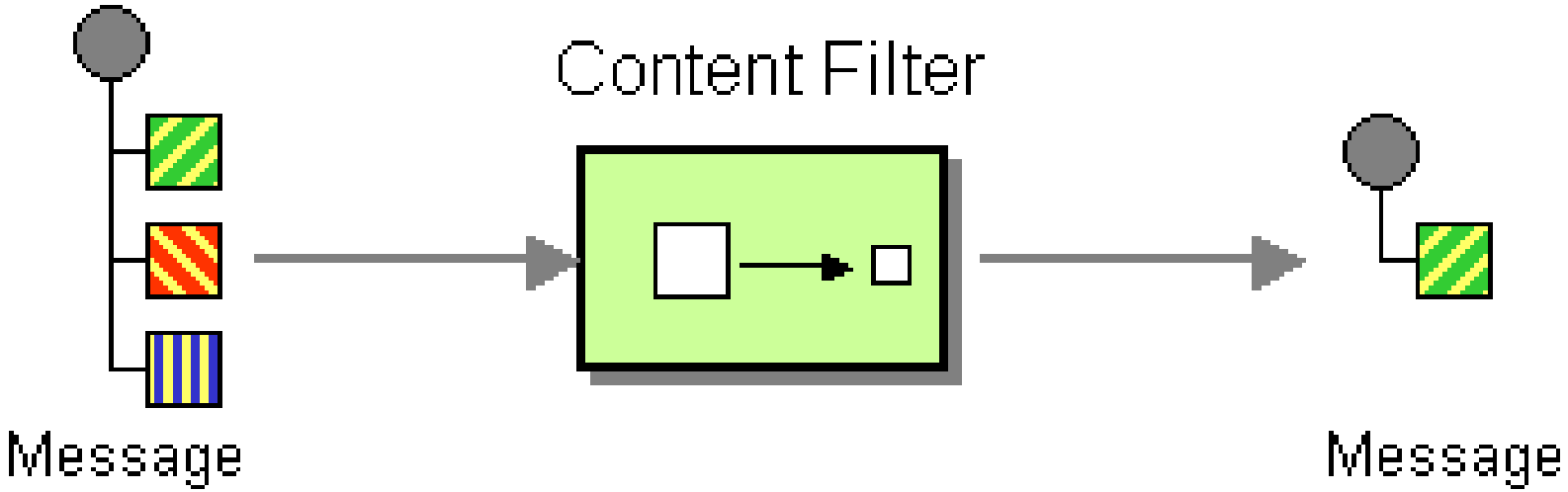


- Escenario:
  - Agregar un token de autenticación
- Es un transformador especial que accede a una fuente externa y agrega información faltante.
- Se utilizan datos del mensaje como clave de consulta a la fuente externa
- El contenido original puede o no permanecer en el mensaje

Content Filter



- ¿Cómo simplificar el procesamiento de grandes mensajes si se está interesado solamente en una porción del mismo?



System Management

System Management

- Soluciones en producción pueden producir, rutear y transformar millones de mensajes por día
- Se debe tratar diariamente con excepciones, problemas de performance, cuellos de botella y cambios en los sistemas participantes.
- Los beneficios de arquitecturas de bajo acoplamiento hacen muy complejo el testing
  - "*architect's dream, developer's nightmare* "

System Management

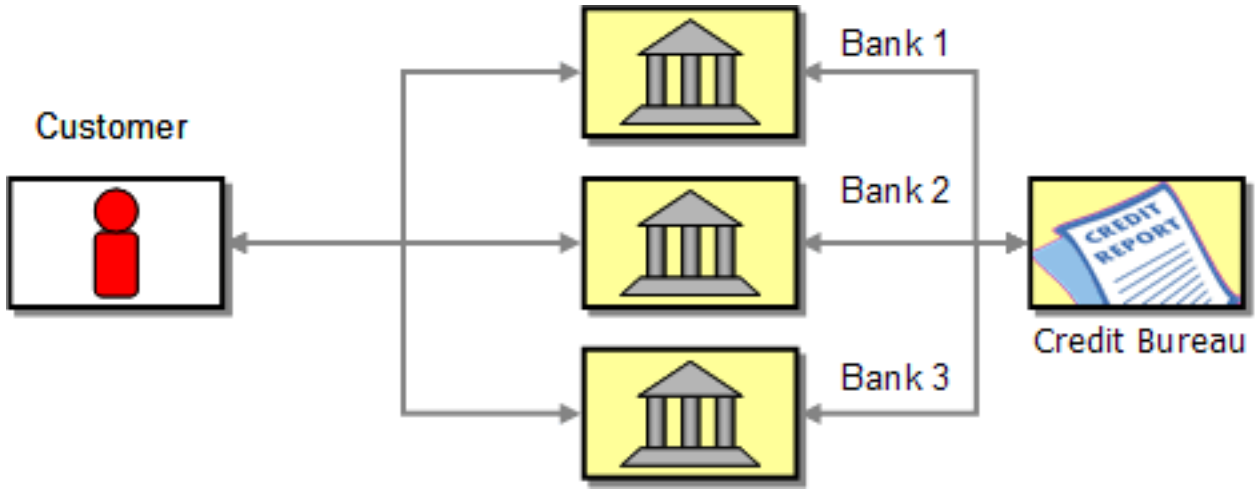
- Asincronismo y aspectos temporales son difíciles de controlar
- La infraestructura de mensajería típicamente asegura la entrega pero no en que tiempo
- Testear un sistema donde quien produce mensajes no sabe quien los va a consumir hace el panorama aún más complejo
- Se requiere de herramientas que permitan:
  - Monitorear y controlar el sistema
  - Observar y analizar el tráfico
  - Testing y debbuging

System Management

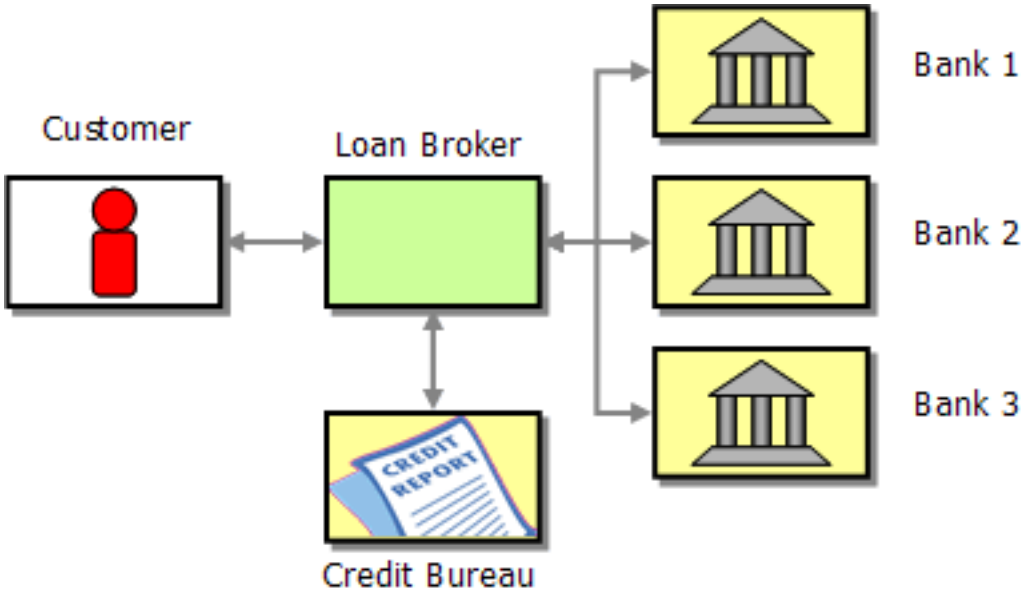
- Monitoring and Controlling
  - Control Bus, Detour
- Observing and Analyzing Message Traffic
  - Wire Tap, Message History, Message Store, Smart Proxy
- Testing and Debugging
  - Test Message, Channel Purger

Caso de Estudio - Loan Broker

- Cotización de un prestamo

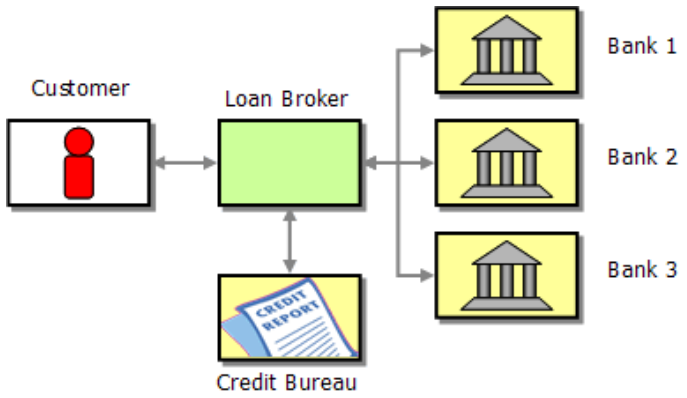


Loan Broker - Análisis



Loan Broker - Análisis

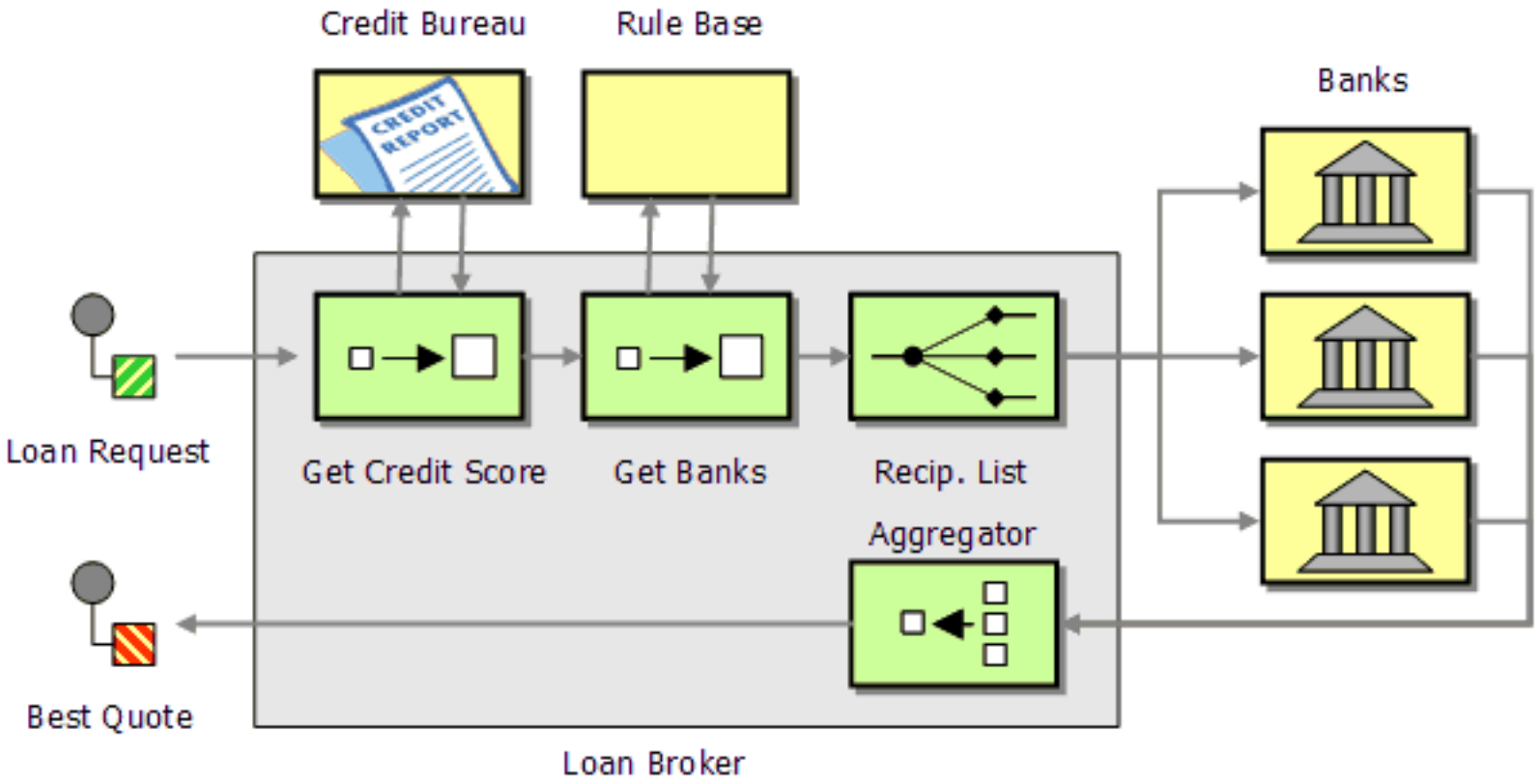
- Tareas a realizar por el Loan Broker:
  - Recibir el requerimiento del cliente
  - Solicitar a la agencia de crédito el historial del cliente
  - Determinar los bancos más apropiados a contactar
  - Mandar una solicitud de crédito a cada banco
  - Determinar la mejor cotización
  - Devolver la cotización al cliente



Loan Broker - Diseño

- Cómo diseñamos la solución ?
- Qué Patrones utilizarían ?

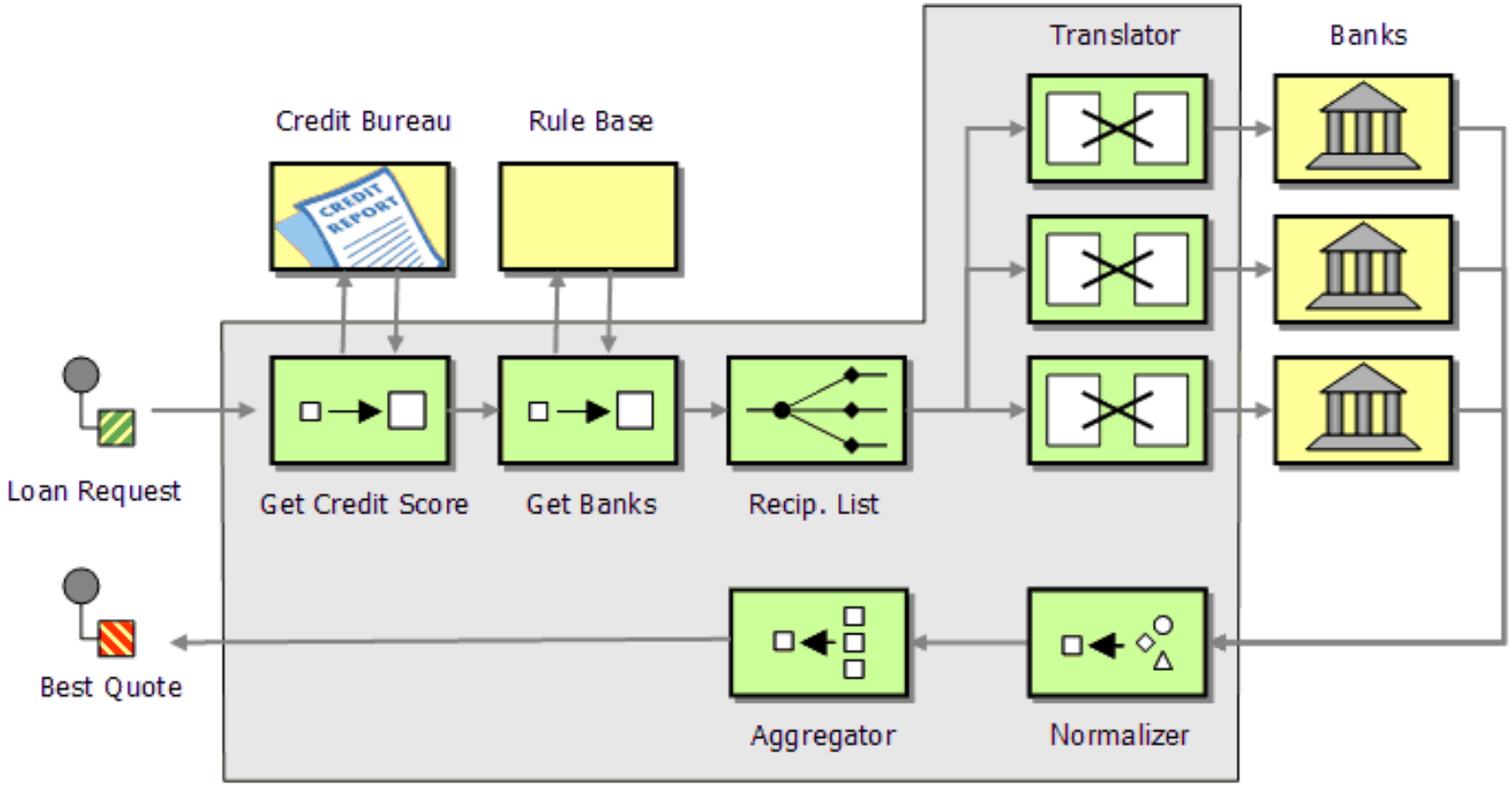
Loan Broker - Diseño



Loan Broker - Diseño

- Es suficiente ?
- Algún otro patrón ?

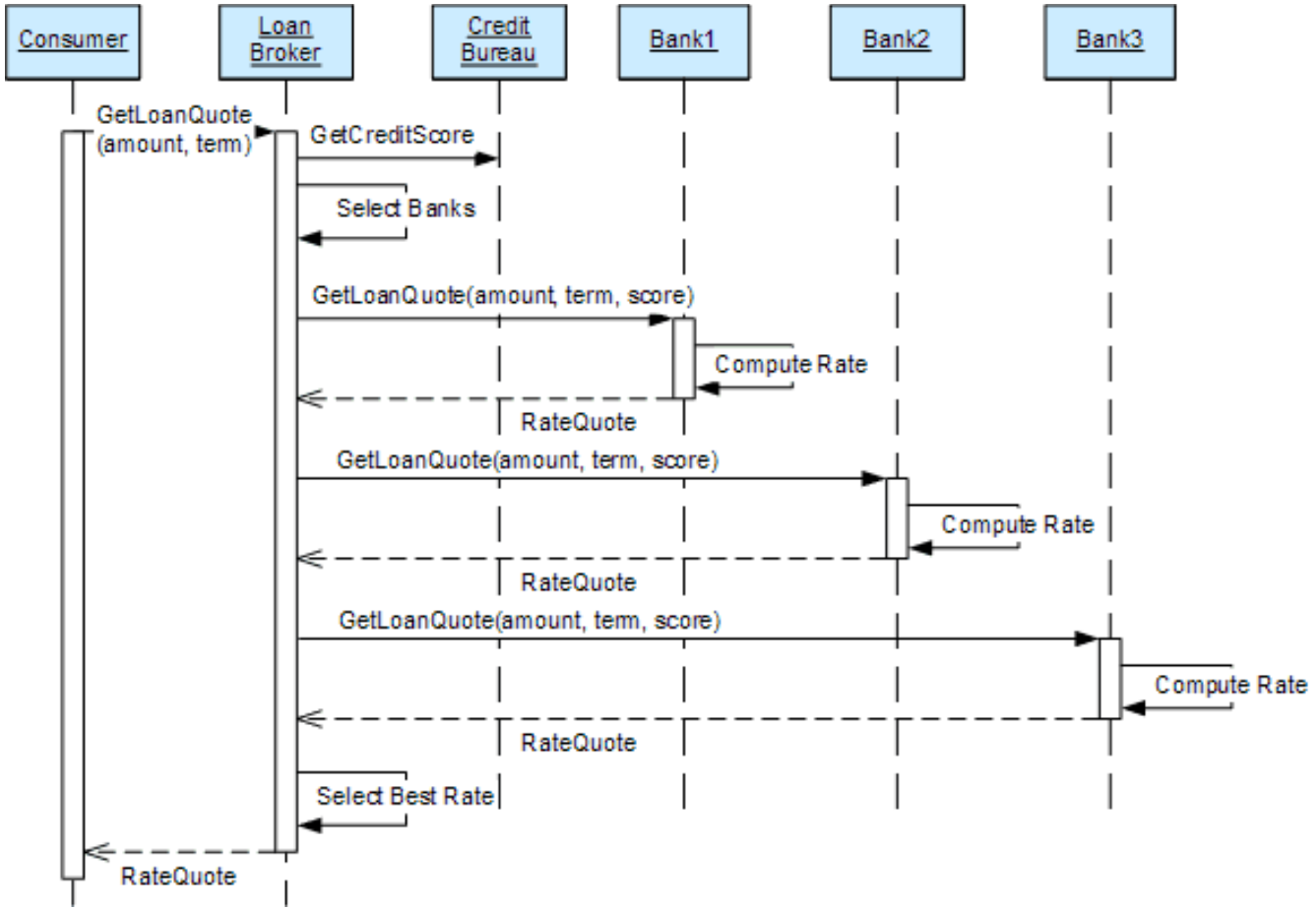
Loan Broker - Diseño



Loan Broker – Diseño Secuenciamiento de mensajes

- Dos Opciones
  - **Sincrónica** (secuencial): El broker le pregunta a cada banco la cotización y espera por la respuesta antes de preguntarle al siguiente banco.
  - **Asincrónica** (paralela): El banco manda el requerimiento a todos los bancos en paralelo.

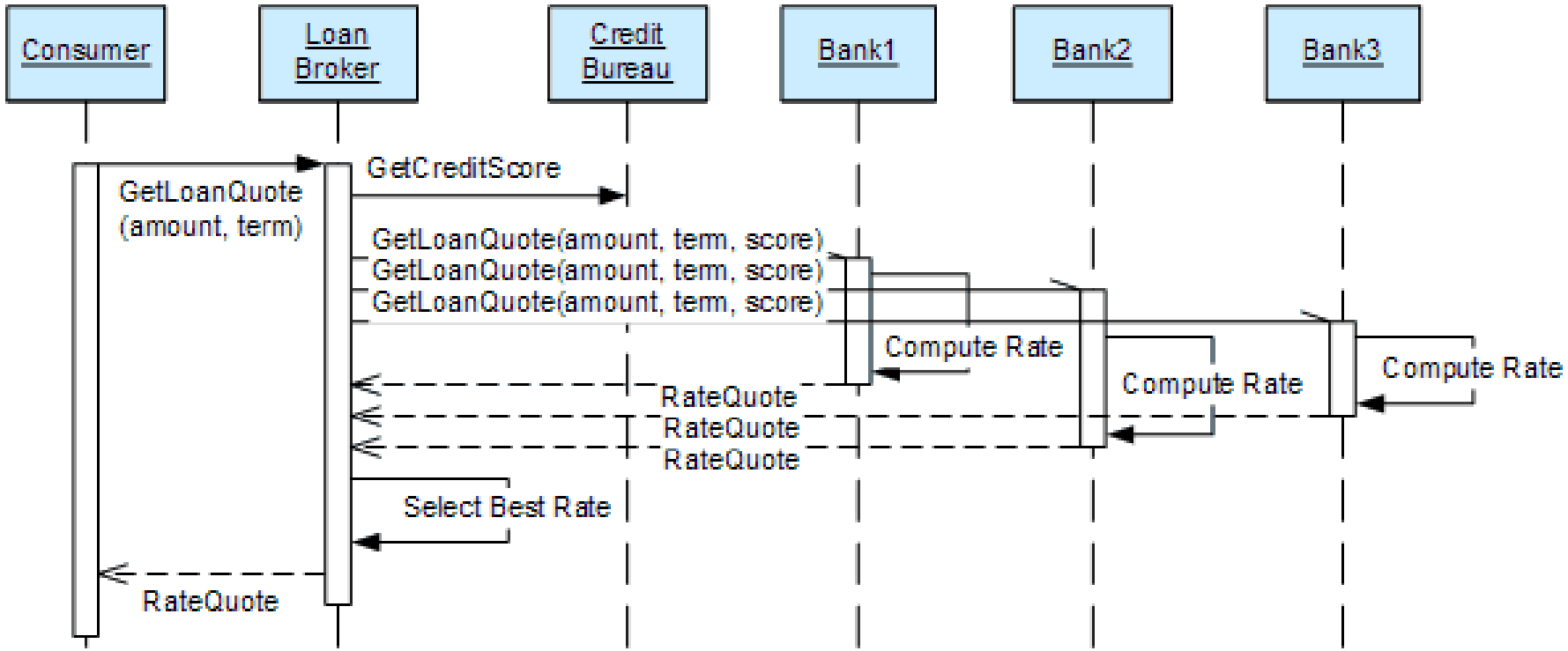
Loan Broker – Diseño Solución sincrónica



Loan Broker – Diseño Solución sincrónica

- Ventajas
  - Simple, no necesitamos manejar aspectos como concurrencia, threads, etc.
- Desventajas
  - Ineficiente, cada banco es independiente y por lo tanto podrían resolver el requerimiento en paralelo.
  - Se tiene que esperar un largo tiempo hasta obtener la respuesta.

Loan Broker – Diseño Solución asincrónica

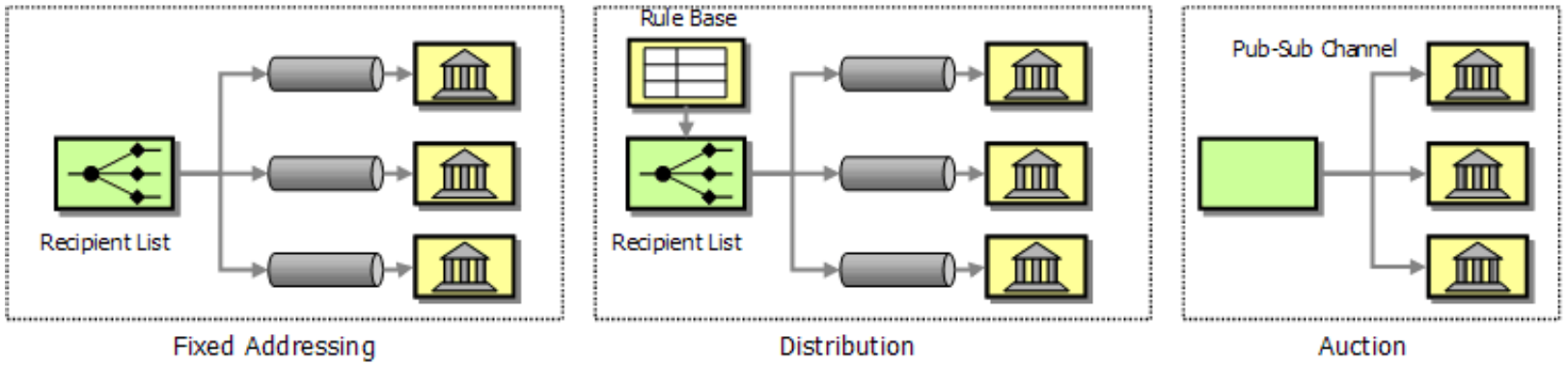


Loan Broker – Diseño Solución asincrónica

- Ventajas
  - Rapidez, los bancos pueden ejecutar en paralelo, por lo tanto la respuesta se obtendrá más rápida.
  - Escalabilidad, ante un cuello de botella por ejemplo en Credit Bureau, la solución se puede escalar teniendo más de una instancia del mismo.
- Desventajas
  - Secuenciamiento de mensajes más complejo
  - Y si alguno no contesta?
  - Timer?

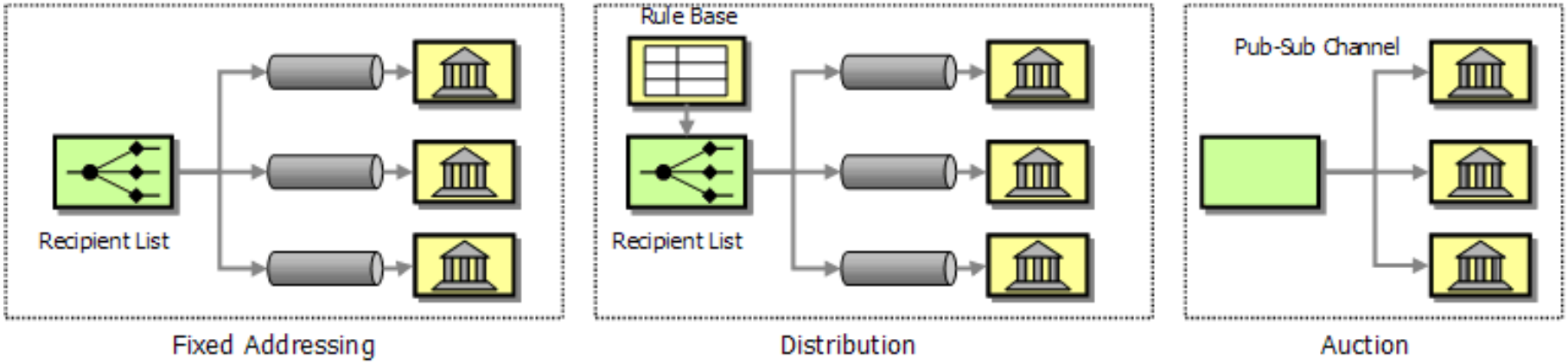
Loan Broker – Diseño Direccionamiento

- Fixed
  - La lista de bancos está hard-coded
- Distribution
  - El Loan Broker mantiene criterios para saber que bancos son buenos dependiendo de cada requerimiento
- Auction
  - El Loan Broker hace broadcast usando un Pub-Sub Channel



Loan Broker – Diseño Direccionamiento

- Cuál es la mejor opción para nuestro escenario ?



Loan Broker – Diseño Direccionamiento - Fixed

- Ventajas
  - Simple, se tiene un buen control sobre la lista de bancos
- Desventajas
  - Si la participación de los bancos es muy dinámica se hace pesada la administración
  - Algunos bancos pueden querer no estar recibiendo constantemente pedidos de cotización que no les intresan

Loan Broker – Diseño
Direccionamiento - Distribution

- Ventajas
  - Se tiene mejor control de a qué bancos involucrar
- Desventajas
  - Requiere lógica de negocio adicional a mantener en el Loan Broker

Loan Broker – Diseño Direccionamiento - Auction

- Ventajas
  - Hacer broadcast a todos los bancos le permite a cada banco decidir que solicitudes de cotización quiere recibir
- Desventajas
  - Requiere trabajo del lado de los bancos
  - Es más complejo esperar la respuesta
    - Considerar dificultad en saber cuantos bancos están suscritos
    - Número máximo de respuestas?, Timeout ?

Loan Broker - Diseño Agregación

- Al recibir las respuestas de los bancos tenemos dos opciones:
  - Recibir en un solo canal de respuesta
  - Recibir en múltiples canales, uno para cada banco
- Qué opción es mejor ?

Loan Broker - Diseño Agregación

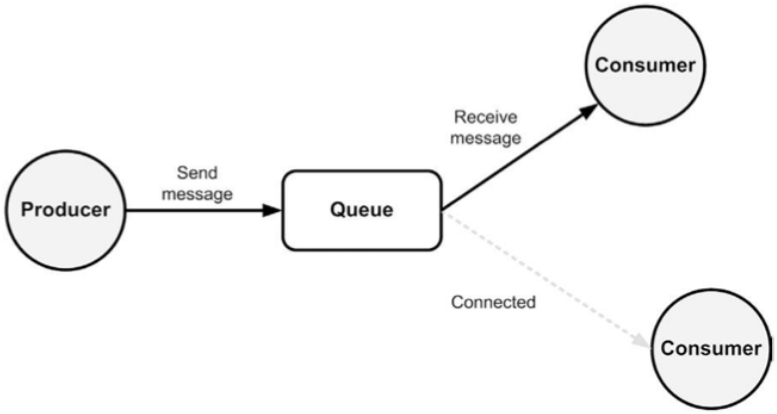
- Un solo canal de respuesta
  - No se tiene un mantenimiento pesado de muchos canales pero cada respuesta tiene que traer un identificador del banco que está respondiendo.
- Múltiples canales de respuesta
  - Cada banco sabe a donde responder, no se necesita id de banco pero se tienen que mantener más canales

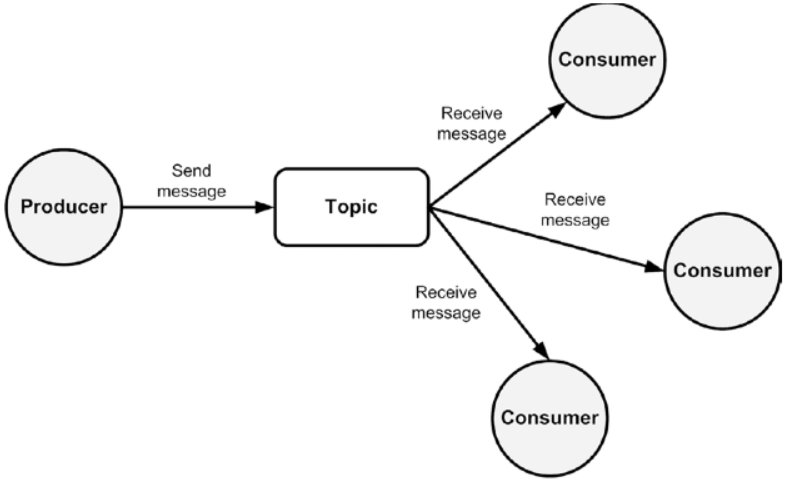
Java Message Service (JMS)

- Es el estándar para la mensajería en Java EE
- El API JMS es el equivalente en el mundo de los mensajes, al API JDBC en el mundo de las bases de datos
  - Actualmente en versión 2.0
- JMS provee una forma unificada de acceder a un MOM desde Java, evitando el uso de APIs propietarias

JMS Brevemente

- Se tienen dos modelos de comunicación
  - Point to Point (PTP)
  - Publish – Subscribe (Pub-Sub)





JMS – Produciendo Mensajes

- Lo primero que tengo que hacer es crear un JMSContext

```
ConnectionFactory factory = new
        ActiveMQJMSConnectionFactory("tcp://localhost:61616");
JMSContext context = factory.createContext();
```

- Un JMSContext representa
  - Una conexión física con el servidor de mensajería
  - Un contexto de tipo single-threaded para enviar y recibir mensajes.

JMS – Produciendo Mensajes

- También debo obtener el canal al cual quiero enviar los mensajes (Queue o Topic)

Destination queue = ActiveMQDestination.createQueue(queueName);

Destination topic = ActiveMQDestination.createTopic(topicName);

- En este caso son colas y tópicos en ActiveMQ

JMS – Produciendo Mensajes

- Por último, crear los mensajes y enviarlos

```
ObjectMessage message = session.createObjectMessage();
ShippingRequest shippingRequest = new ShippingRequest();
shippingRequest.setItem(item);
shippingRequest.setShippingAddress(address);
shippingRequest.setShippingMethod(method);
shippingRequest.setInsuranceAmount(amount);

context.createProducer().send(queue, shippingRequest.toString());
```

- Recomendación:
  - Enviar mensajes de tipo texto para reducir acoplamiento entre productor y consumidor

JMS – Produciendo Mensajes

- Un JMSContext representa
  - Una conexión física con el servidor de mensajería
  - Un contexto de tipo single-threaded para enviar y recibir mensajes.

- La conexión es costosa de crear
  - Usar un pool de conexiones si se requieren muchas conexiones

JMS – Produciendo Mensajes

- Es necesario siempre cerrar la conexión
- JMSContext implementa AutoCloseable
- Usando try-with-resources no es necesario cerrar explícitamente la conexión.
  - Java lo hace por nosotros
  - https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

JMS – Produciendo Mensajes

```
public void sendMessageJMS20(ConnectionFactory connectionFactory,
                                        Queue queue, String text) {

  try (JMSContext context = connectionFactory.createContext();){
    context.createProducer().send(queue, text);
  } catch (JMSRuntimeException ex) {
    // handle exception (details omitted)
  }

}
```

JMS – Recibiendo Mensajes

- Recordar existen dos tipos de suscriptores:
  - Event-driven consumer
  - Polling consumer

JMS – Polling consumer

- Debemos crear el JMSContext
- Crear un consumidor
- Recibir el mensaje
  - No olvidar cerrar la conexión!

JMSConsumer consumer = context.createConsumer(queueName);

String body = consumer.receiveBody(String.class);

JMS – Event-driven consumer

- Debemos crear el JMSContext
- Crear un consumidor
- Crear una clase MessageListener e implementar el método de procesamiento del mensaje
- Asociar el consumidor a nuestro MessageListener

JMS – Produciendo Mensajes

```
JMSConsumer consumer = context.createConsumer(queue);
consumer.setMessageListener(new MessageListener() {
        @Override
        public void onMessage(Message message) {
                // Procesar mensaje

        }
    });
```

Resumen

- Mensajería es una forma de comunicar aplicaciones a través del envío de paquetes de datos llamados mensajes
- Las aplicaciones disponen de canales que son utilizados para enviar/recibir mensajes
- Los MOM son el Middleware de base para implementar soluciones basada en Mensajería
- Los MOM brindan: Garantía de entrega de mensajes, Comunicación asíncrona, Soporte transaccional, etc

Resumen

- Soluciones basadas en Mensajería presentas nuevos desafíos al estar las aplicaciones desconectadas

- Los EIP brindan un marco conceptual para el diseño, implementación y comunicación de soluciones basadas en Mensajería.

Bibliografía

- **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**. Gregor Hohpe, Bobby Woolf. 2003.
- **Enterprise Integration Patterns**
  http://www.eaipatterns.com/
- **Java Message Service**. Mark Richards, David A Chappell. 2009