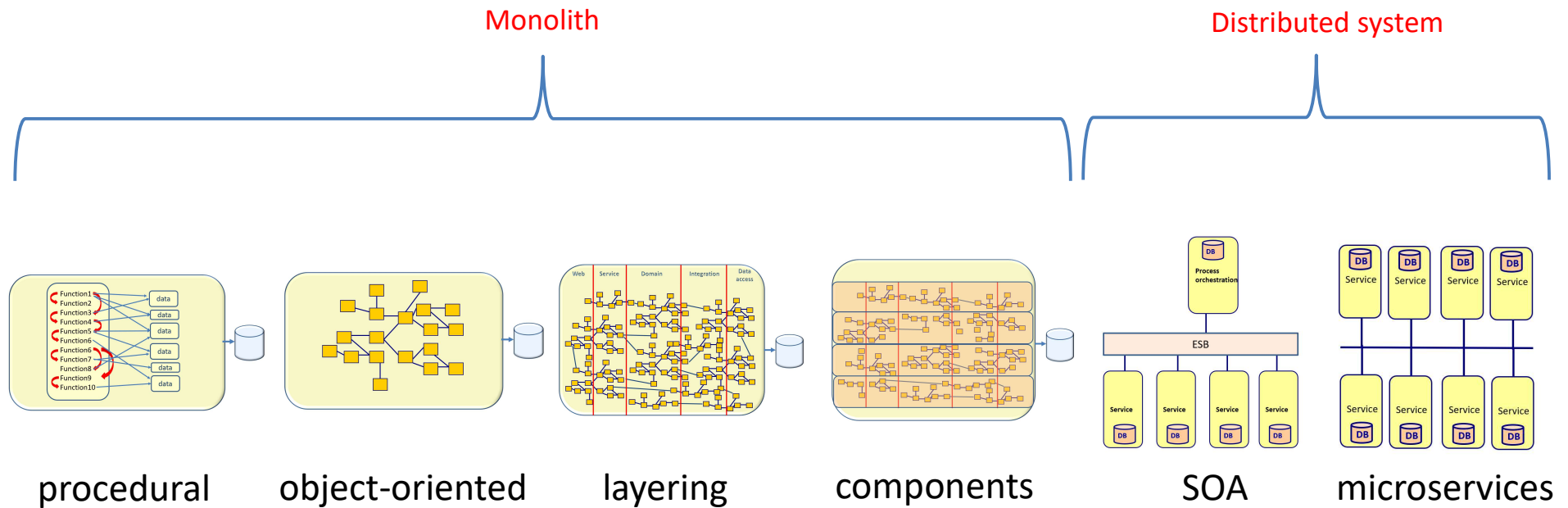


Lesson 6

SERVICE ORIENTED ARCHITECTURE INTEGRATION PATTERNS



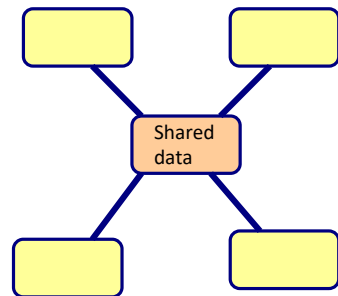
Architecture evolution



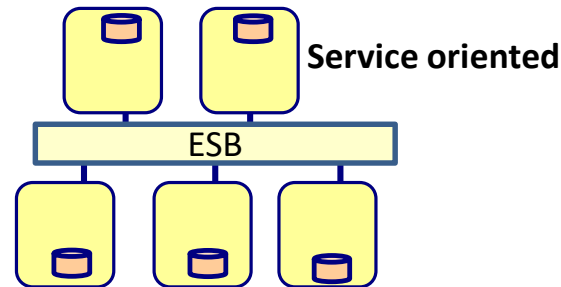
- Smaller and simpler parts
- More separation of concern
- More abstraction
- Less dependencies



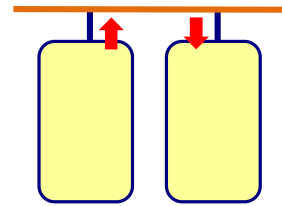
Architecture styles to connect applications



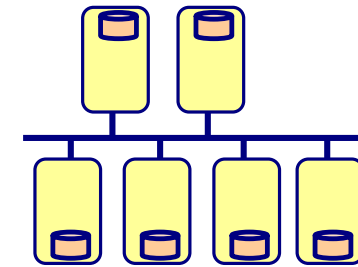
Blackboard



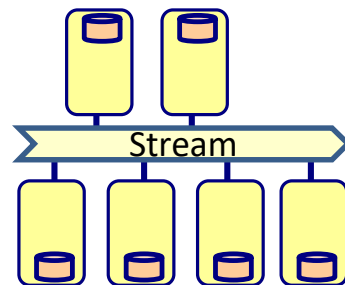
Service oriented



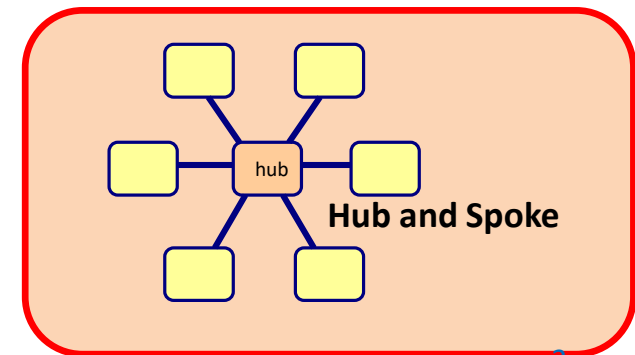
Event Driven



Microservices



Stream based

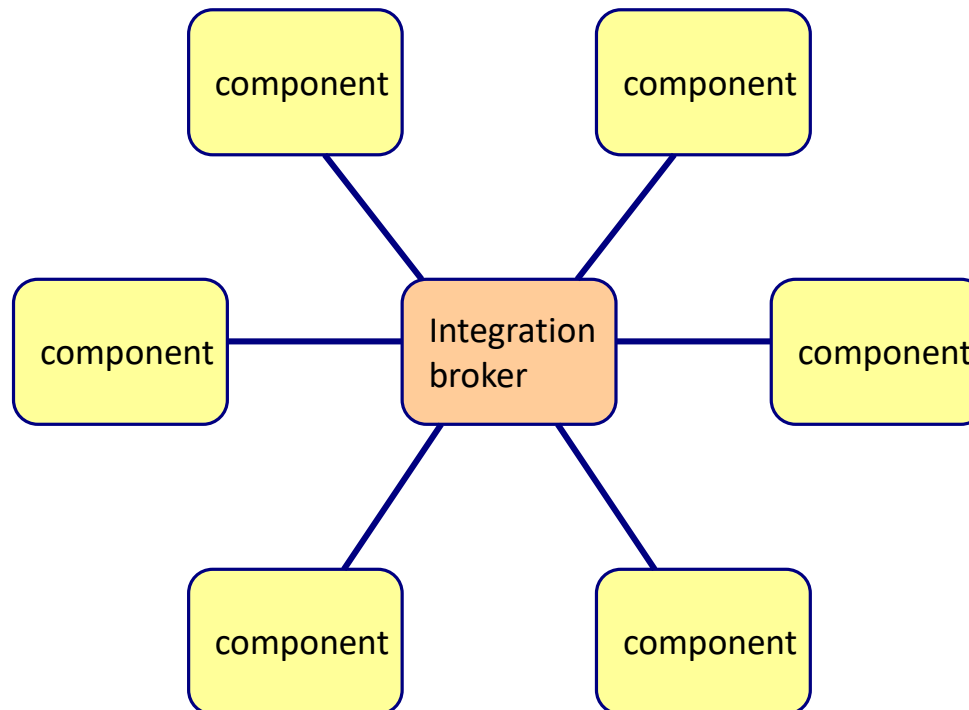


Hub and Spoke



Hub and Spoke

- Integration broker



Hub and Spoke

- Functionality:

- Transport

- Transformation

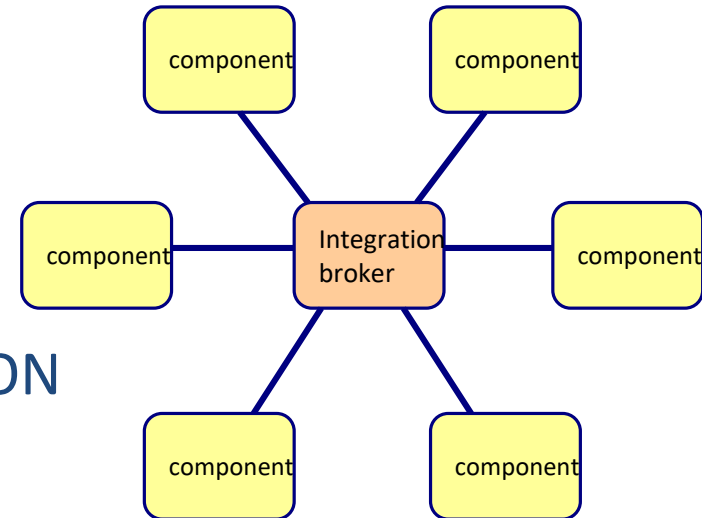
- For example from XML to JSON

- Routing

- Send the message to a component based on certain criteria (content based routing, load balancing, etc.)

- Orchestration

- The business process runs within the integration broker



Hub and spoke

■ Benefits

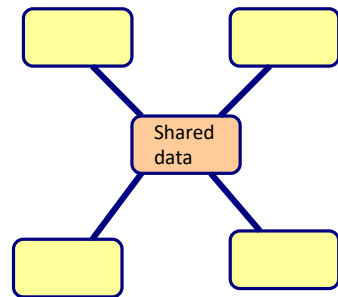
- Separation of integration logic and application logic
- Easy to add new components
- Use adapters to plugin the integration broker

■ Drawbacks

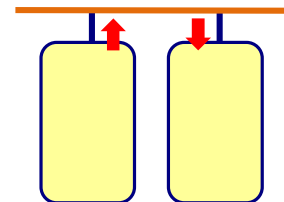
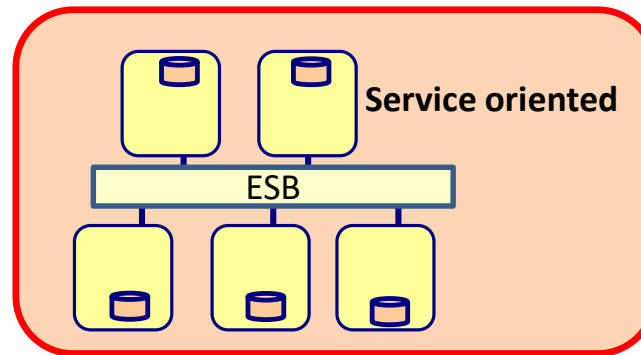
- Single point of failure
- Integration brokers are complex products
- Integration broker becomes legacy itself



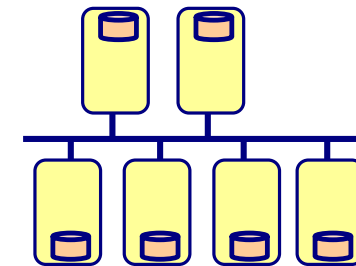
Architecture styles to connect applications



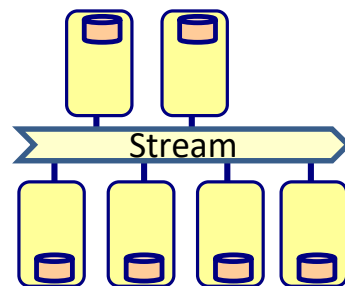
Blackboard



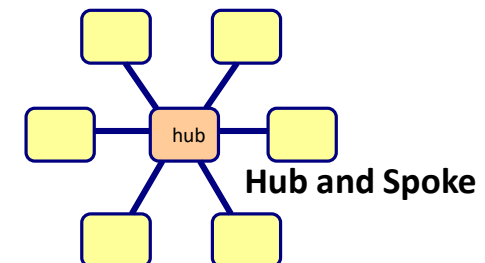
Event Driven



Microservices



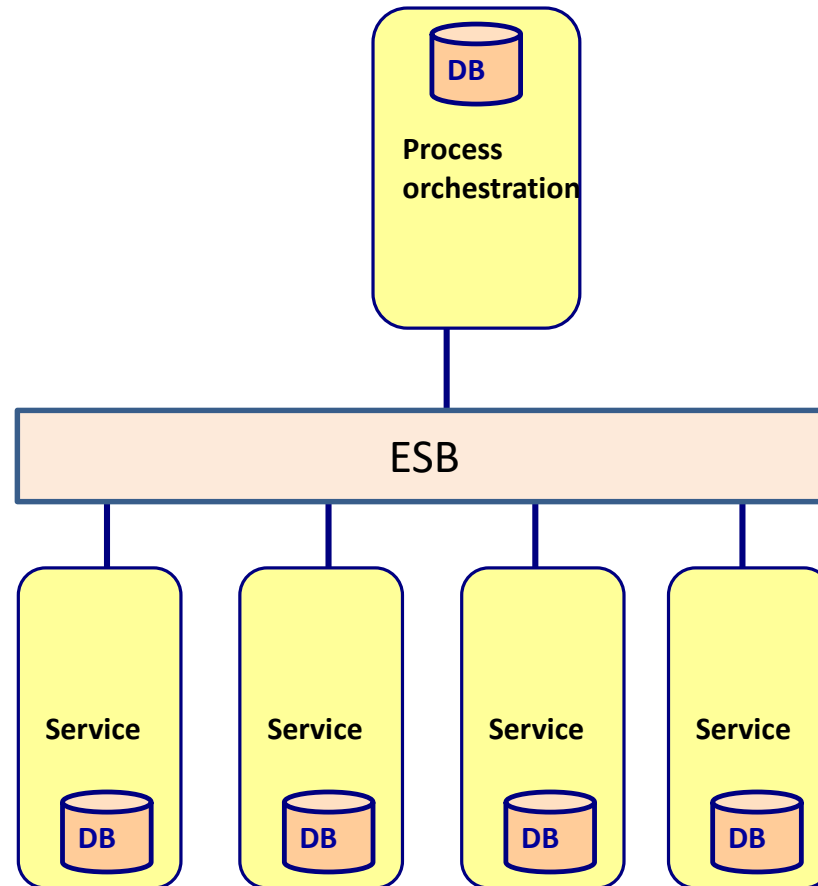
Stream based



Hub and Spoke



Service Oriented Architecture



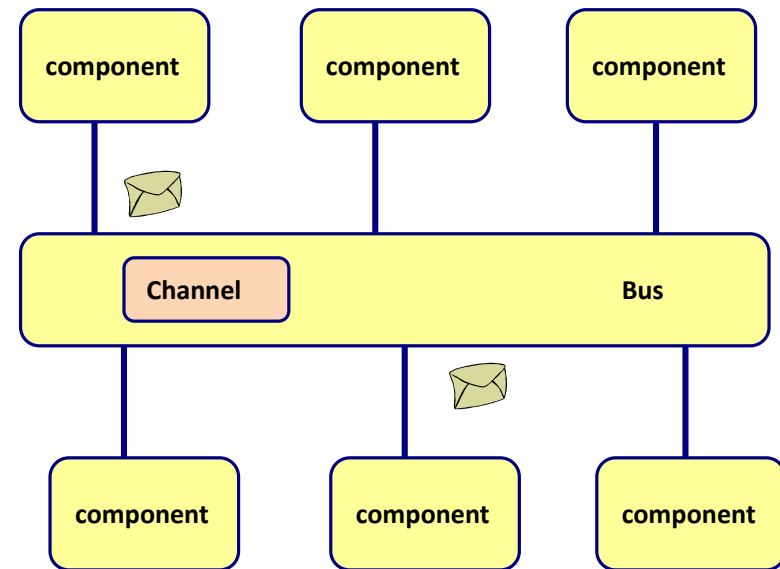
3 different aspects of a SOA

1. Communication through ESB
 - Standard protocols
2. Decompose the business domain in services
 - Often logical services
3. Make the business processes a 1st class citizen
 - Separate the business process from the application logic

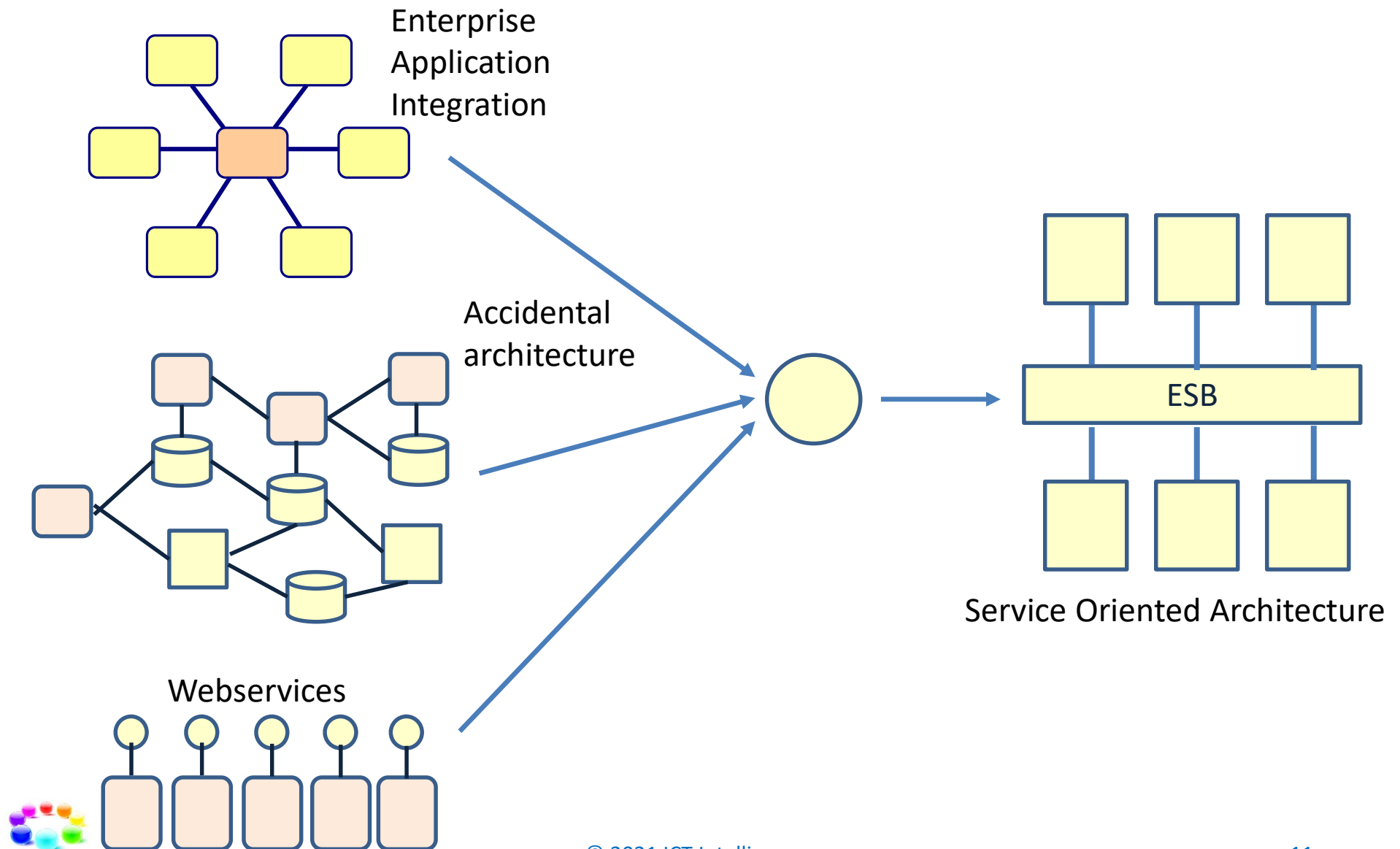


Responsibility of the bus

- Routing
 - Static
 - Content based
 - Rule based
 - Policy based
- Message transformation
- Message enhancing/filtering
- Protocol transformation
 - Input transformation
 - Output transformation
- Service mapping
 - Service name, protocol, binding variables, timeout, etc.
- Message processing
 - Guaranteed delivery
- Process choreography
 - Business process
 - BPEL
- Transaction management
- Security

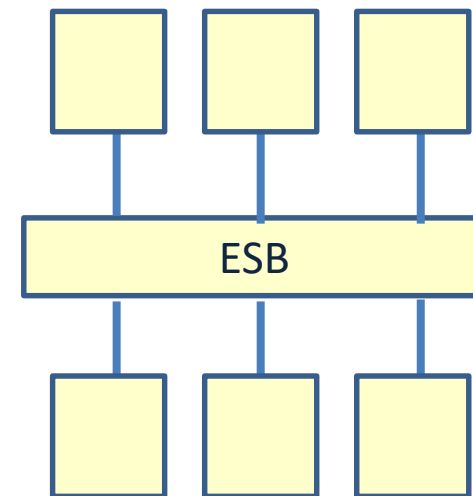


How did we get to SOA?



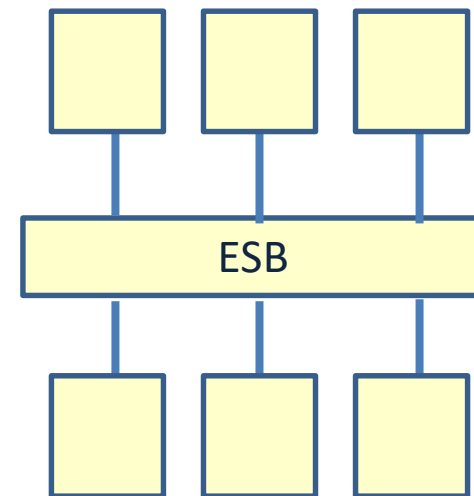
Characteristics of a SOA

- Business processes run on the ESB
- Course grained services
 - To manage performance
 - To manage transactions



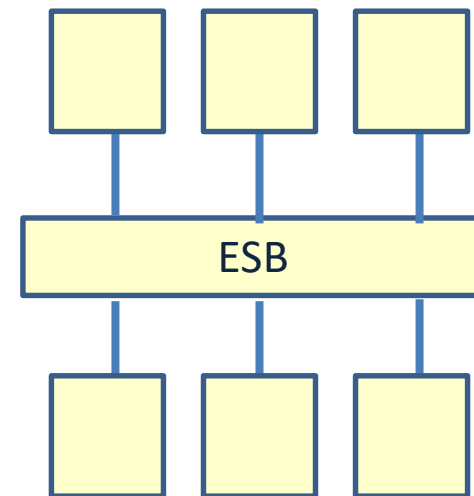
Service Oriented Architecture

- Advantages
 - Independent services
 - Easy to add new services
 - Separation of business processes and service logic
 - Architecture is optimized for the business
 - Reuse of services
 - Architecture flexibility



Service Oriented Architecture

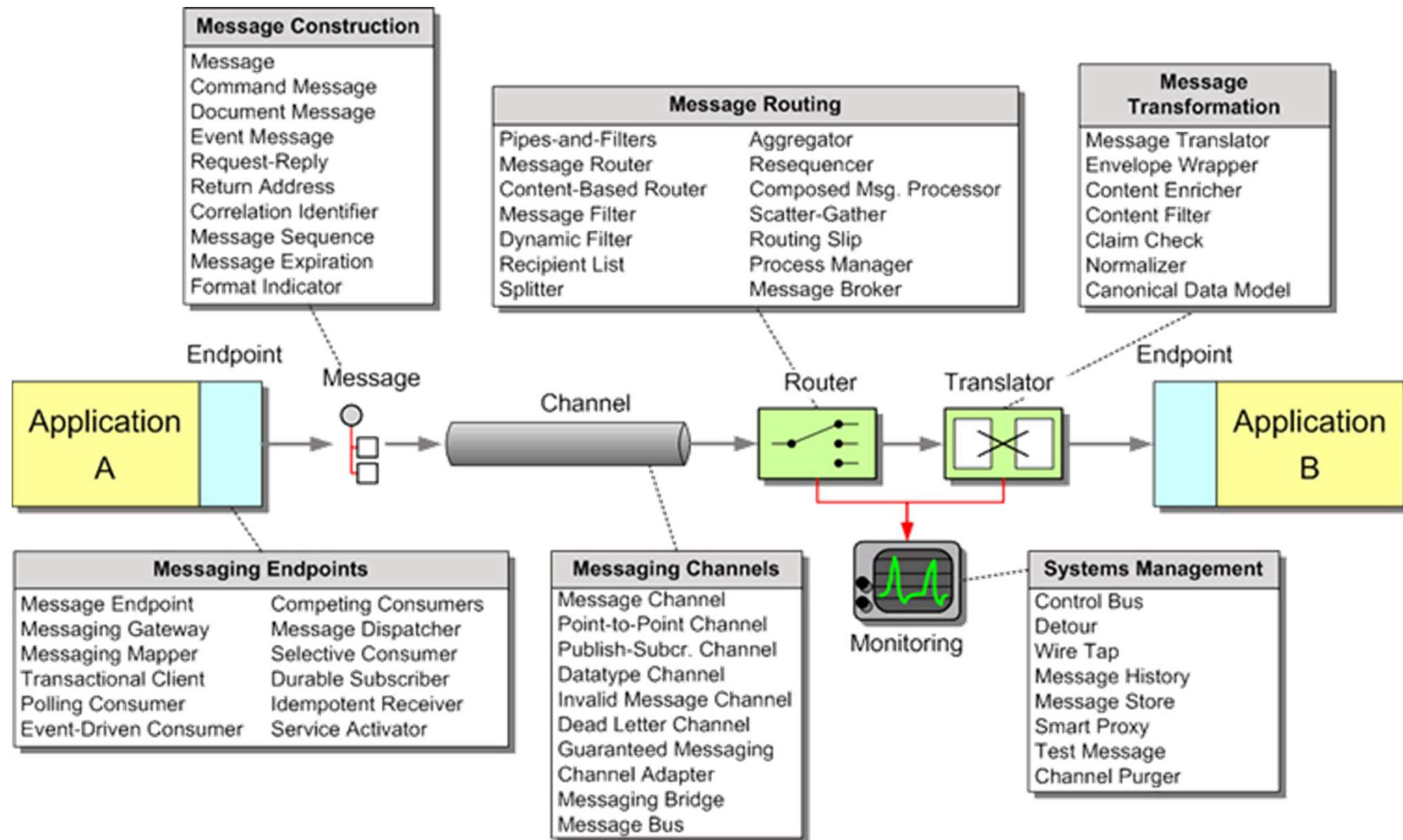
- Disadvantages
 - Complex ESB
 - Changing the business process while still business processes are running is very difficult
 - Most SOA's are build on top of monoliths



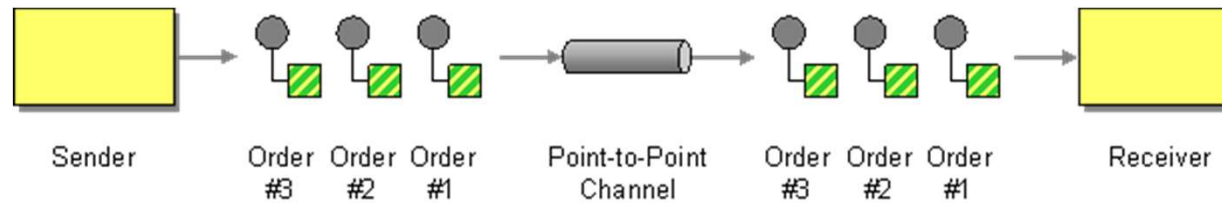
ENTERPRISE INTEGRATION PATTERNS



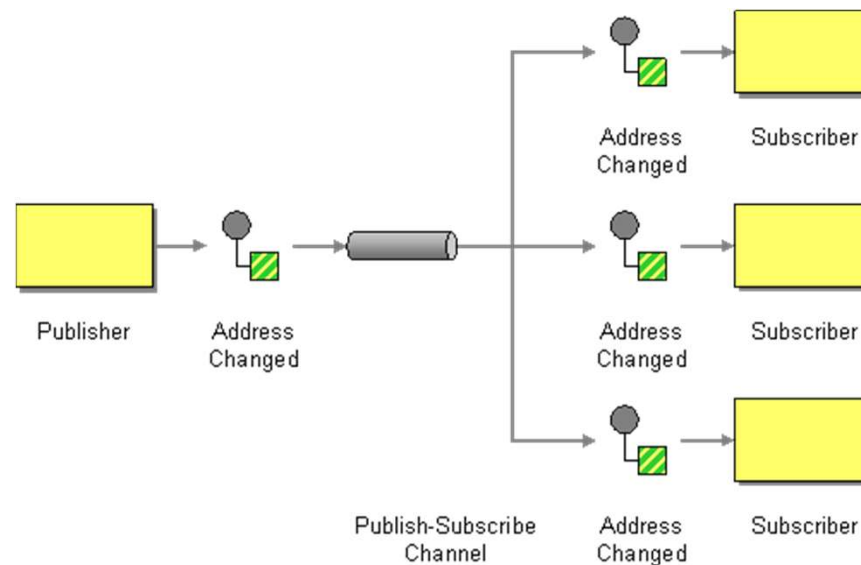
Enterprise Integration Patterns



Messaging channel patterns



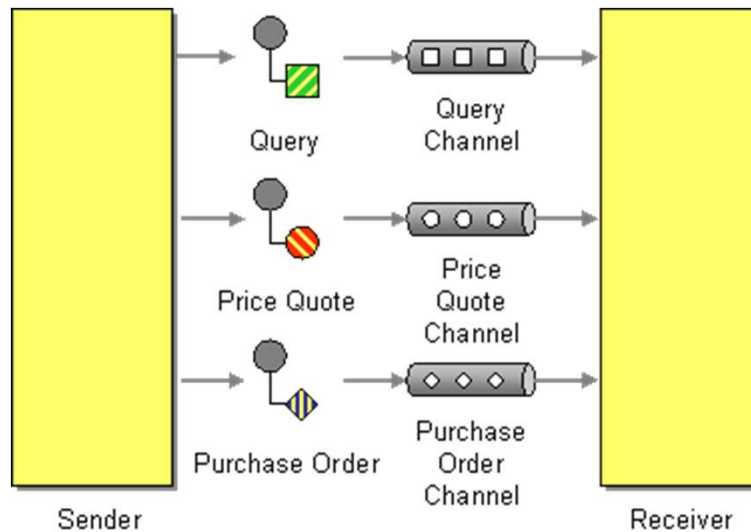
Point-to-point: only one receiver will receive the message



Publish-Subscribe : every subscriber will receive the message



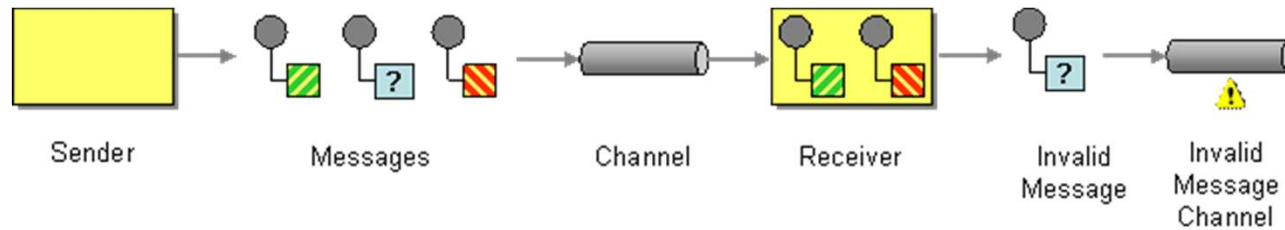
Messaging channel patterns



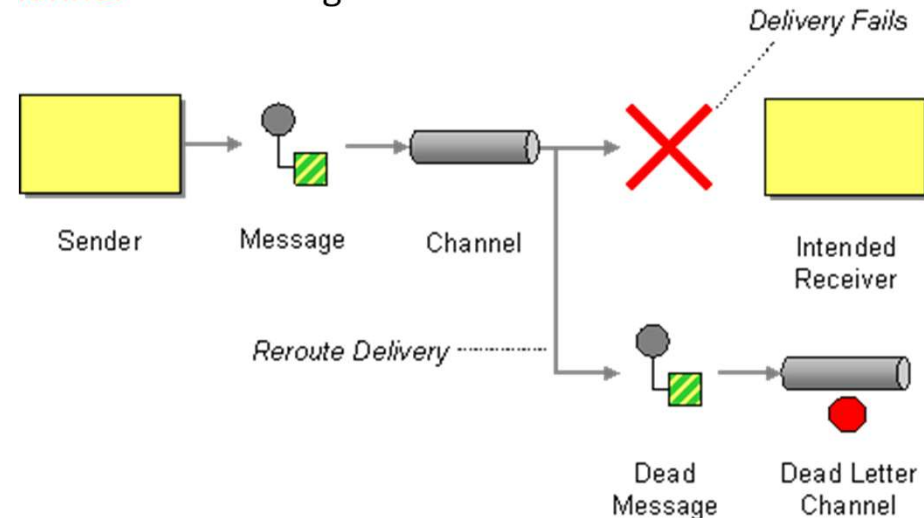
Datatype Channel : use a channel for each data type, so that the receiver know how to process it



Messaging channel patterns



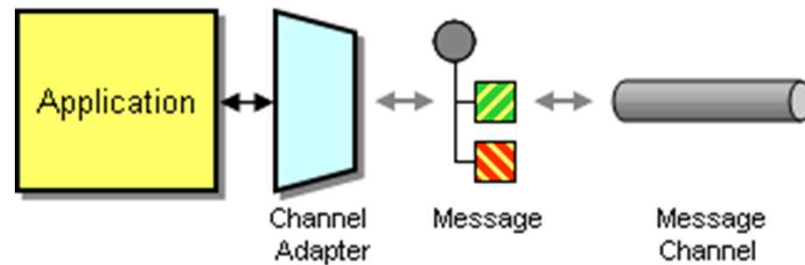
Invalid Message Channel : for messages that don't make sense for the receiver



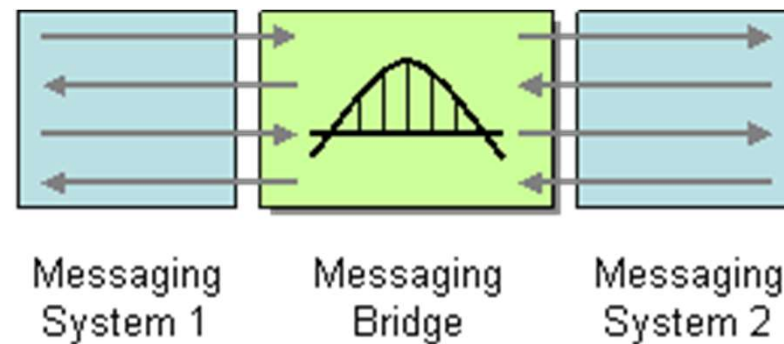
Dead Letter Channel : for messages that can't be delivered



Messaging channel patterns



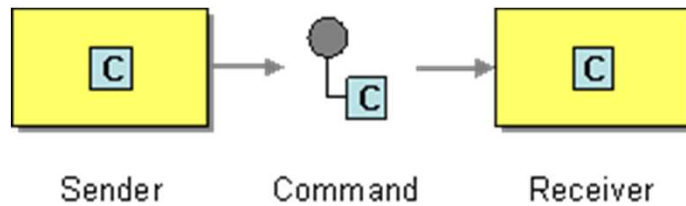
Channel adapter : connect the application to the messaging system



Message bridge : connect 2 messaging systems

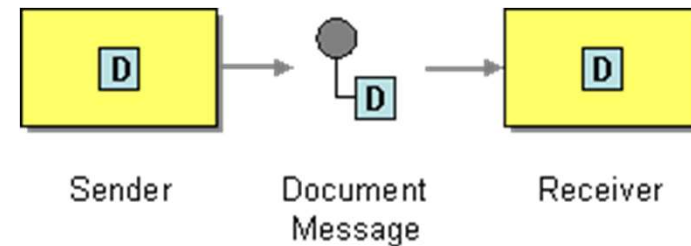


Message construction patterns

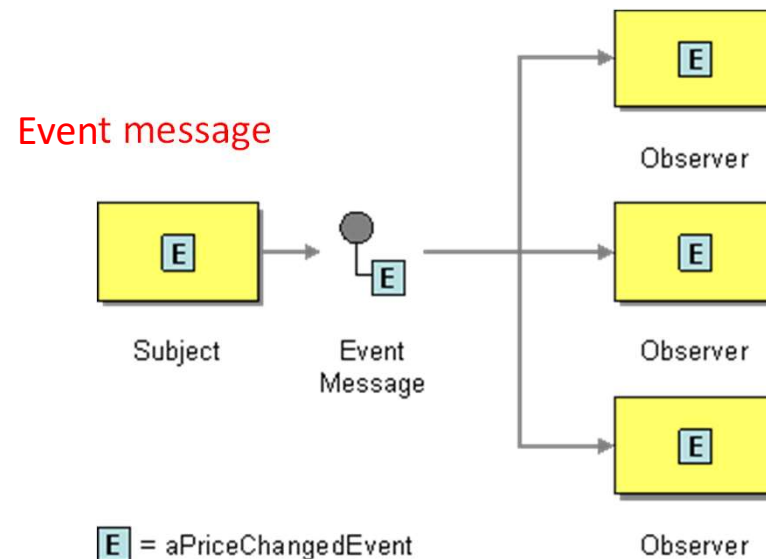


C = getLastTradePrice("DIS");

Command message



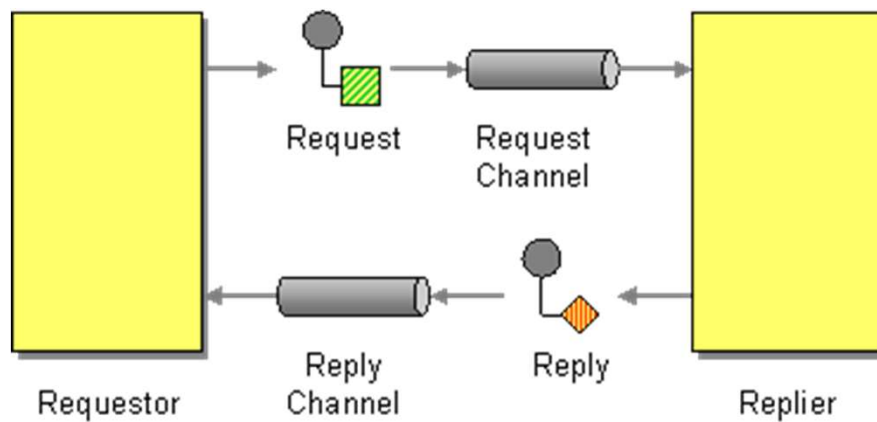
D = aPurchaseOrder Document message



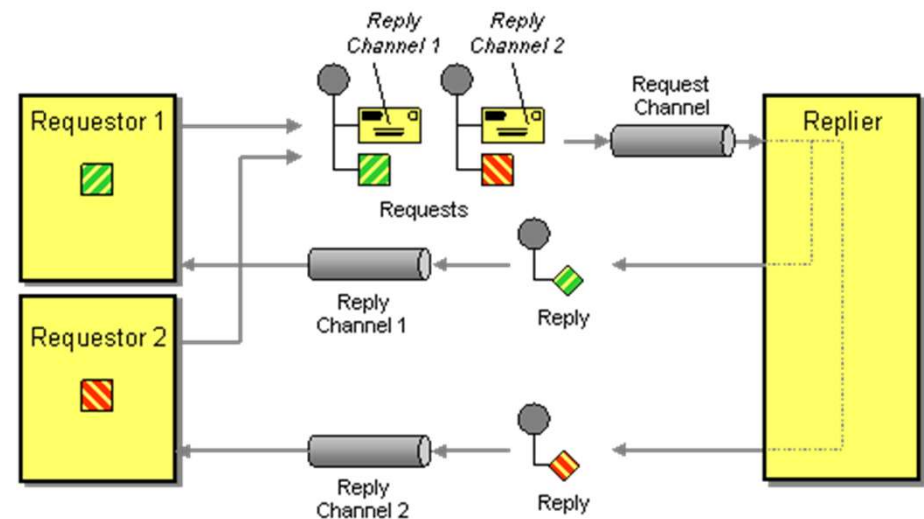
E = aPriceChangedEvent



Message construction patterns



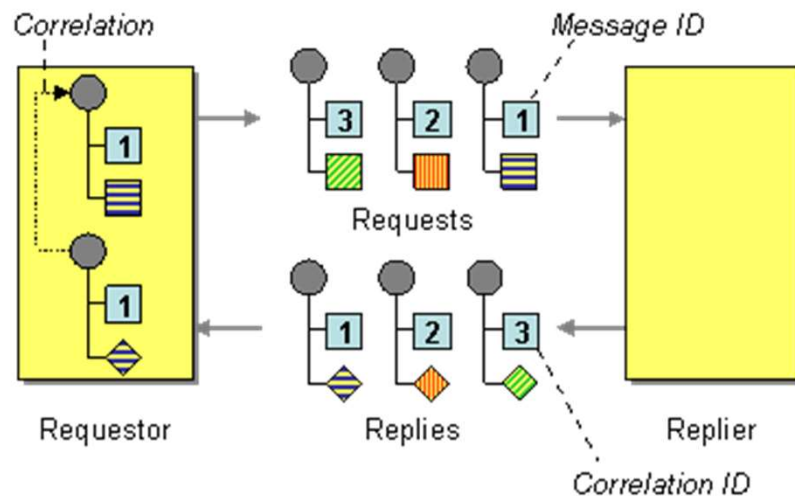
Request-Reply



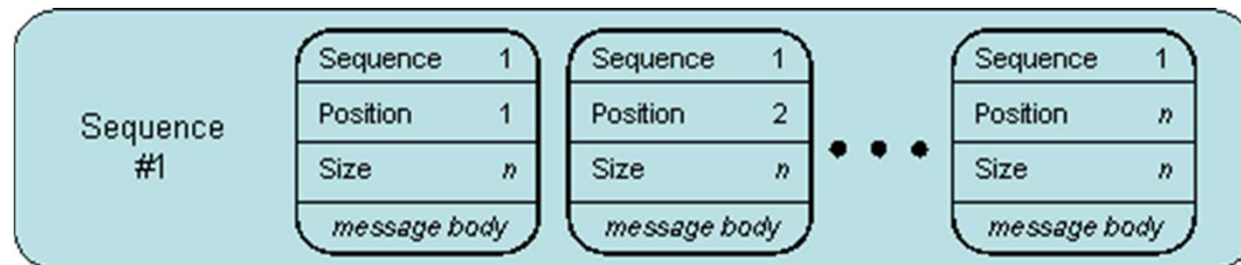
Return address



Message construction patterns



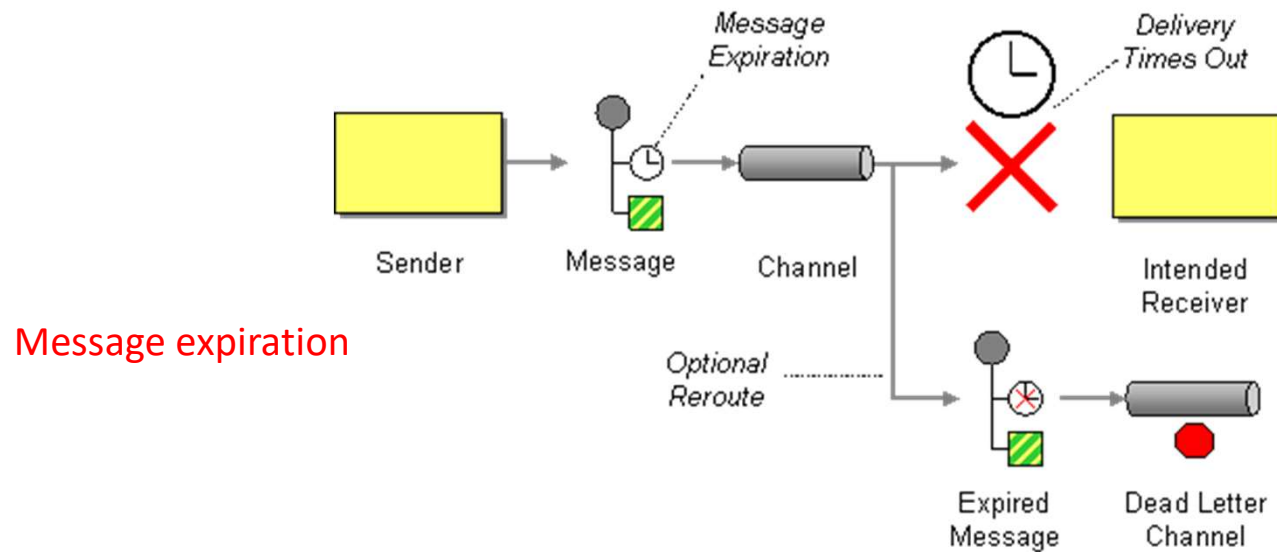
Each reply message should contain a **Correlation Identifier**, a unique identifier that indicates which request message this reply is for



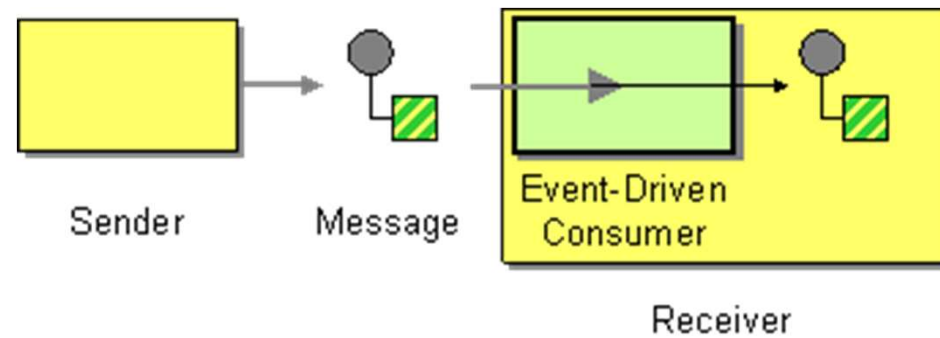
Whenever a large set of data may need to be broken into message-size chunks, send the data as a **Message Sequence** and mark each message with sequence identification fields.



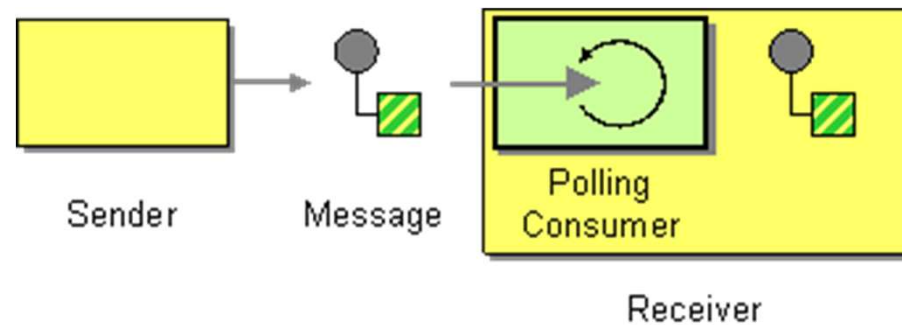
Message construction patterns



Message Endpoint



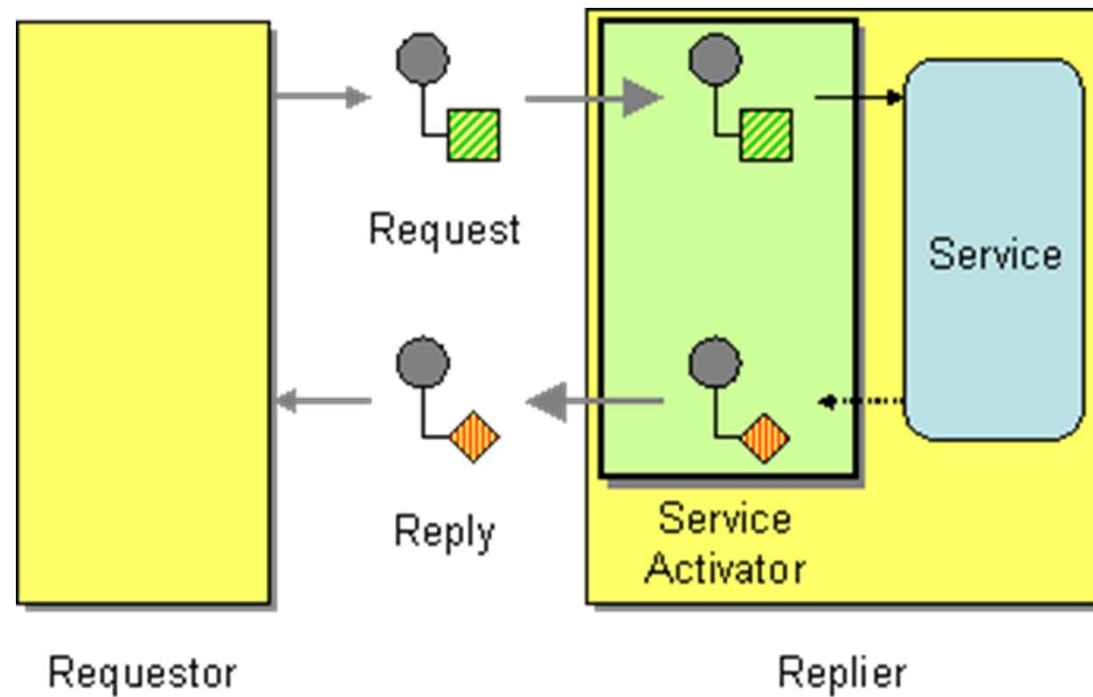
Event driven consumer



Polling consumer



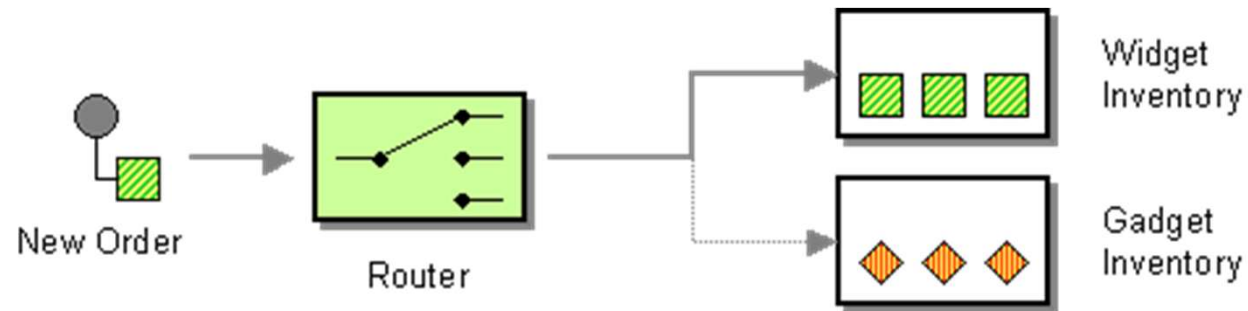
Message Endpoint



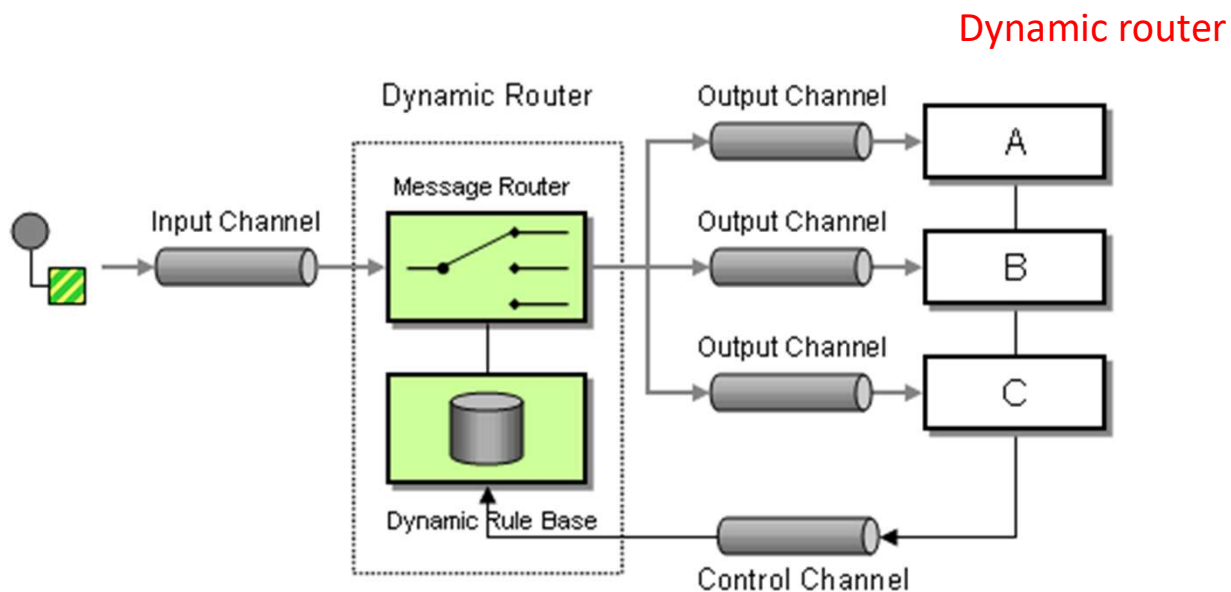
Service activator



Message Routing



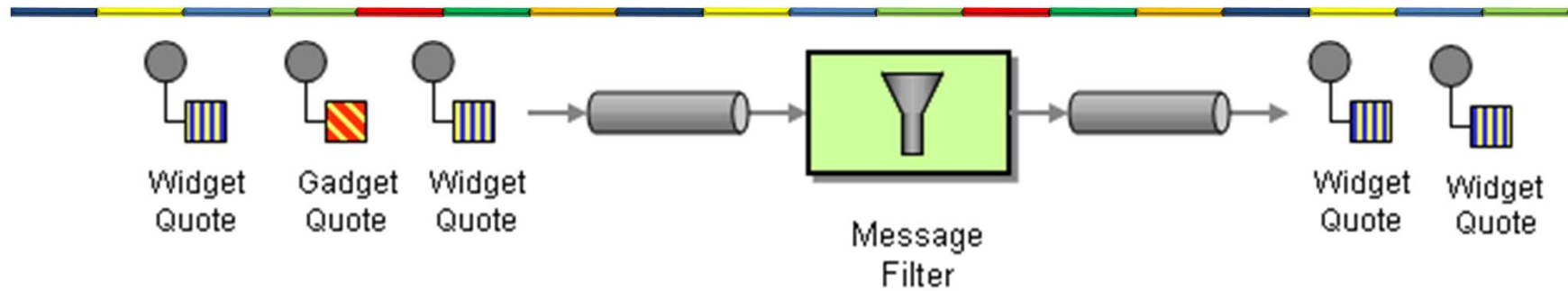
Content based router



Dynamic router

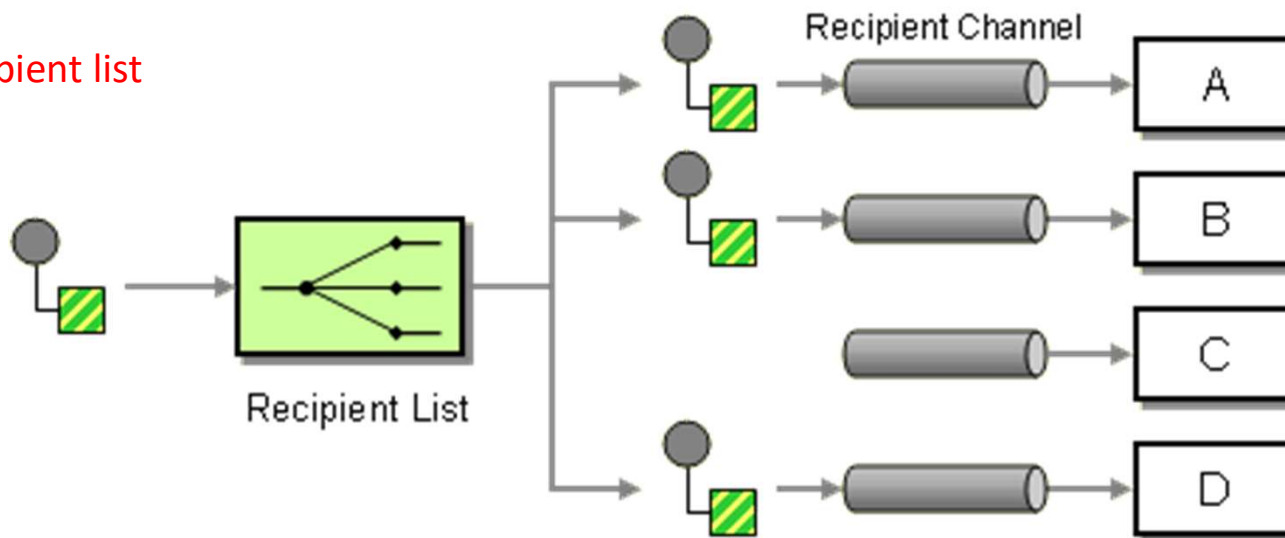


Message Routing

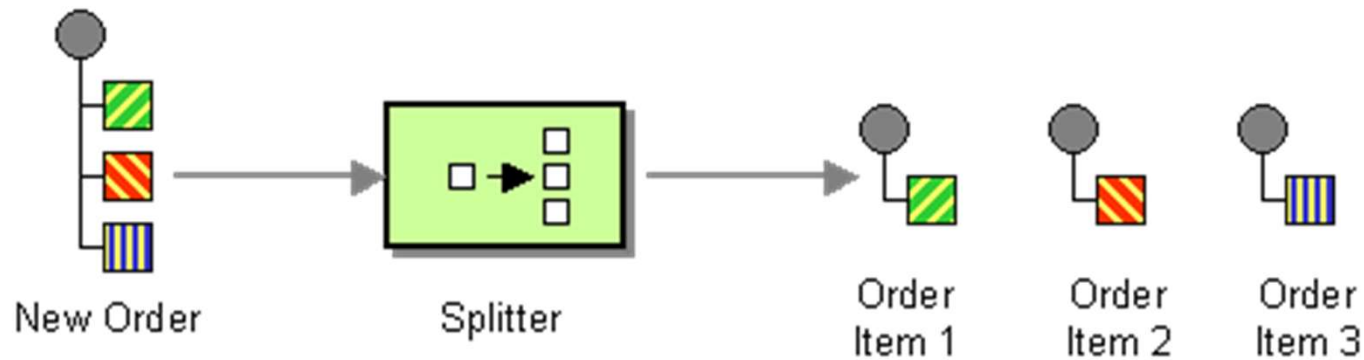


Message filter

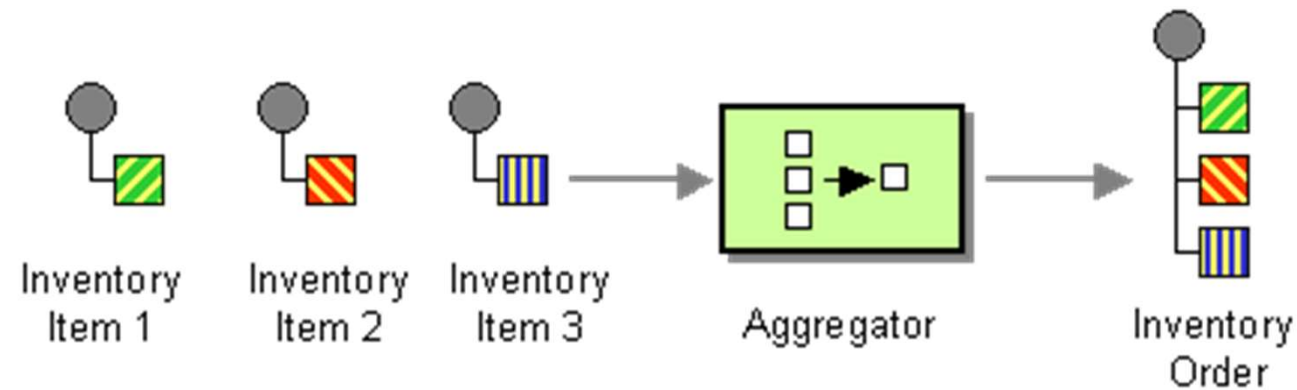
Recipient list



Message Routing



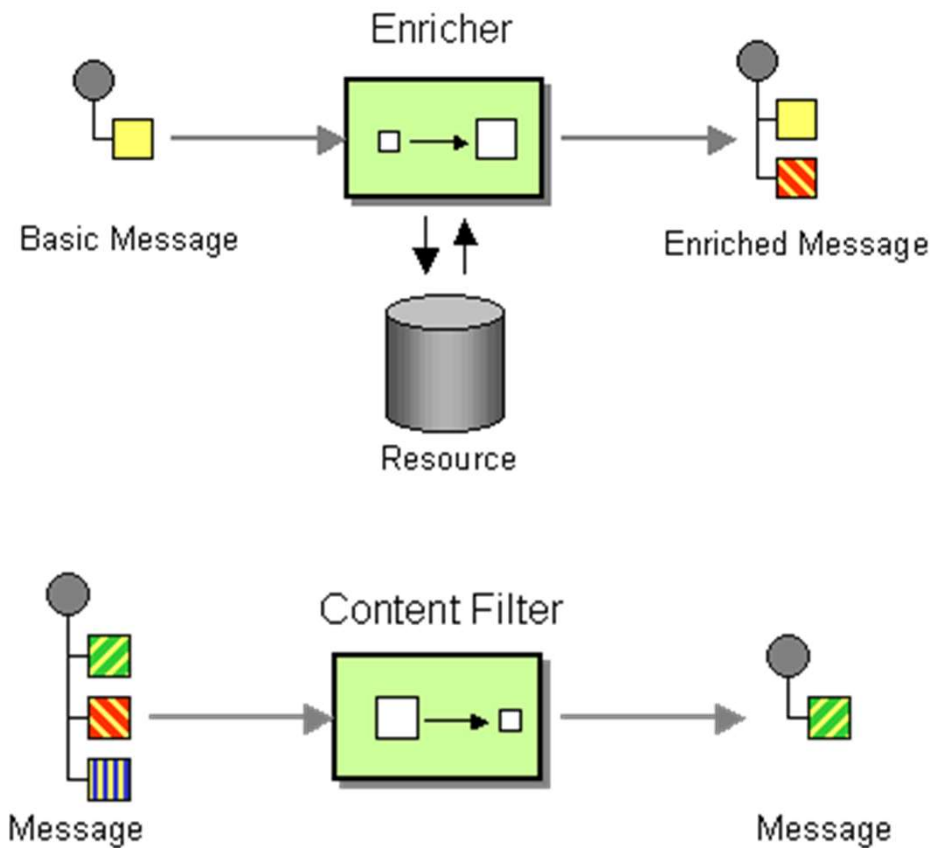
Splitter



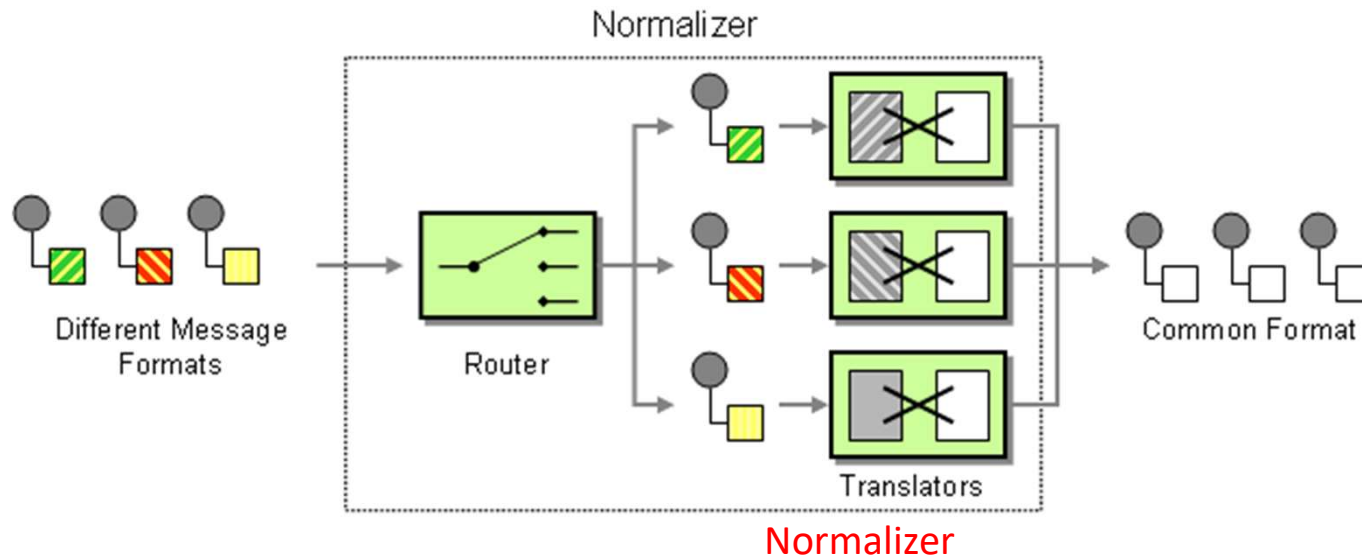
Aggregator



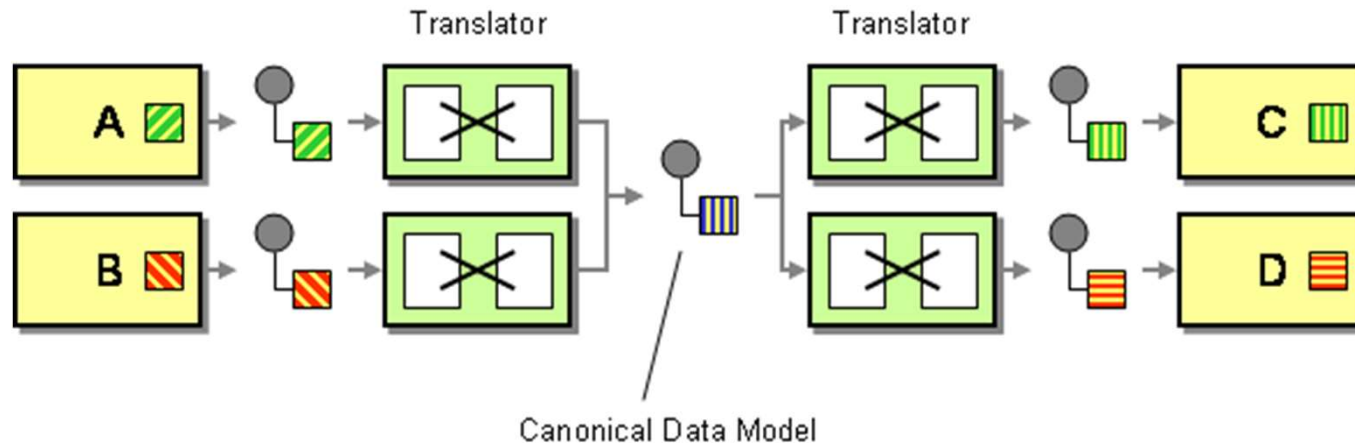
Message Transformation



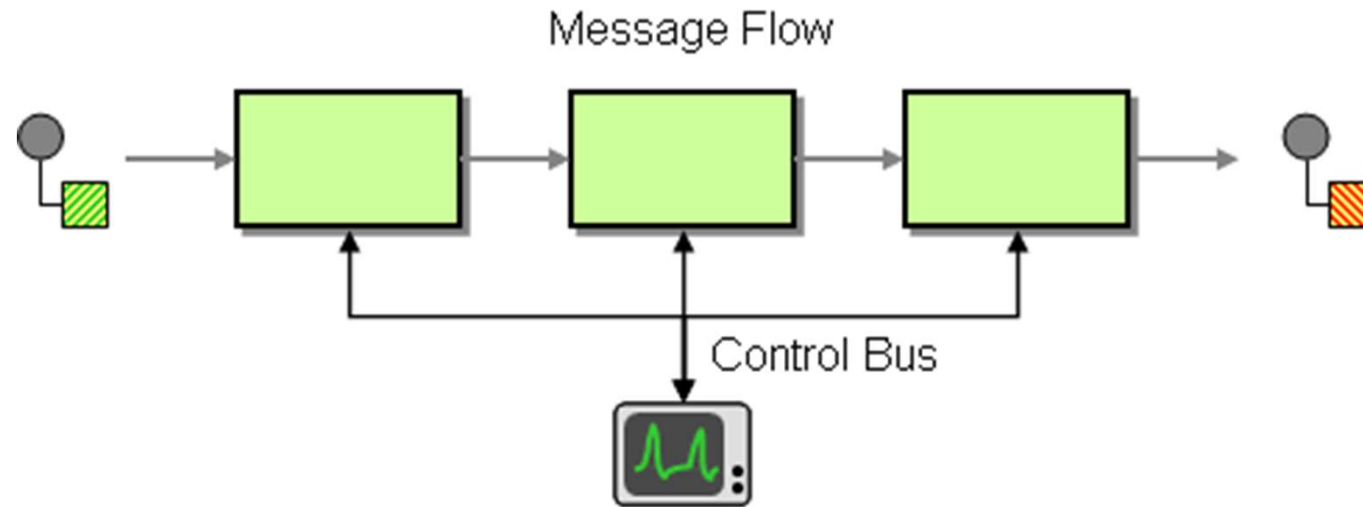
Message Transformation



Canonical Data Model



Management

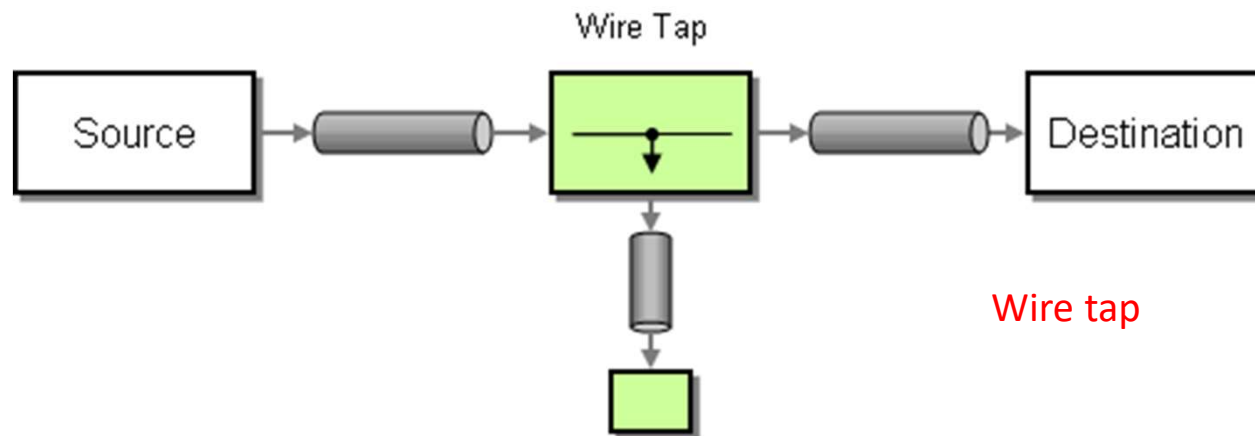
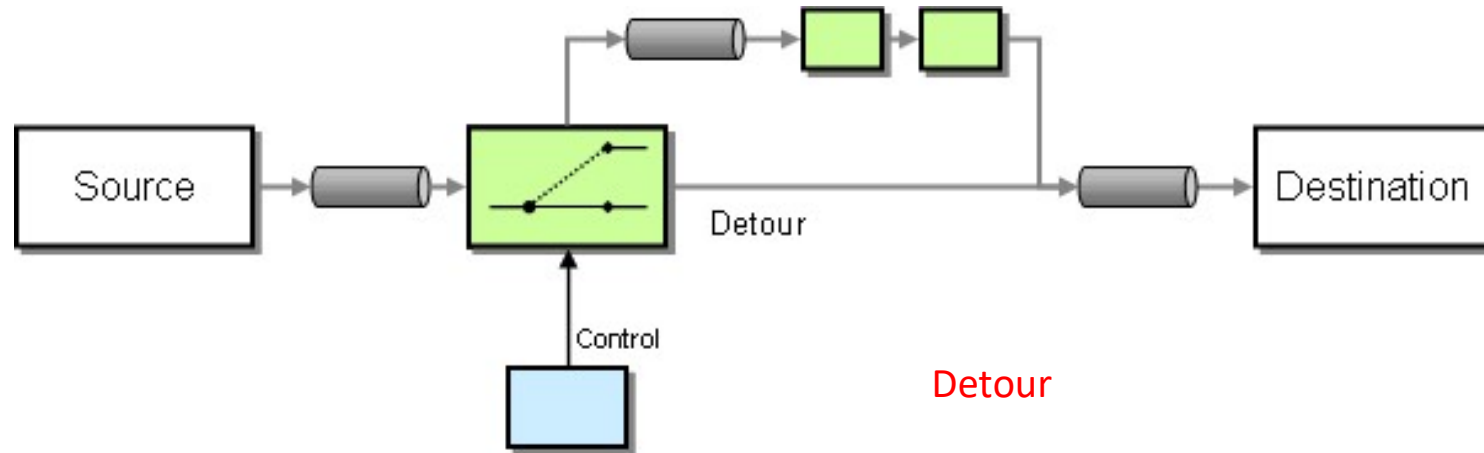


Use a *Control Bus* to manage an enterprise integration system.

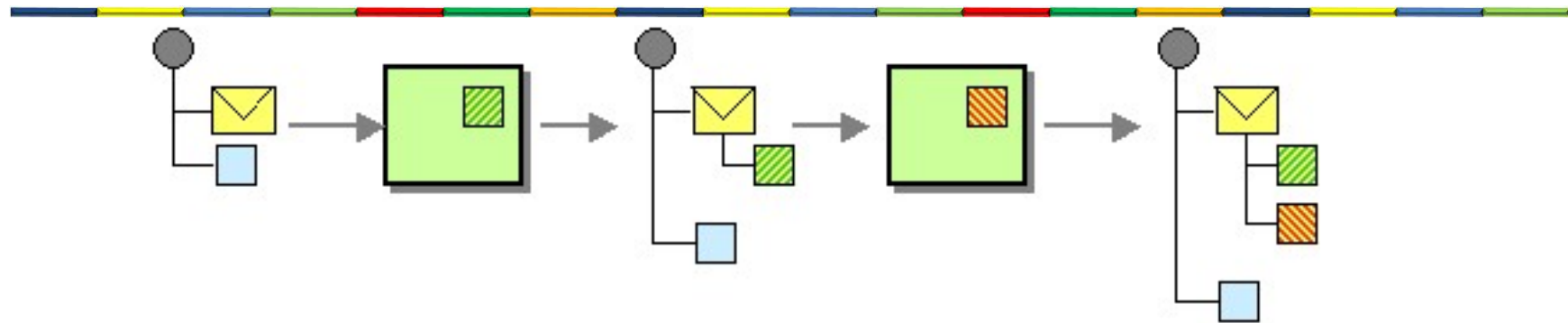
- same messaging mechanism
- separate channels to transmit management relevant data



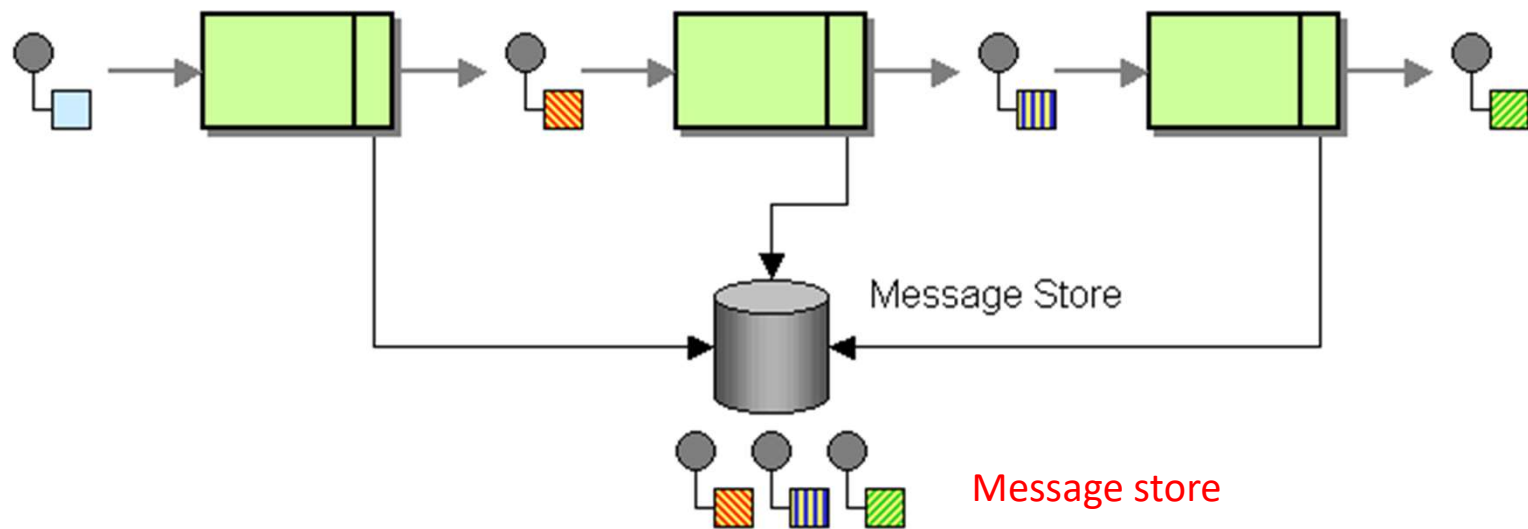
Management



Management



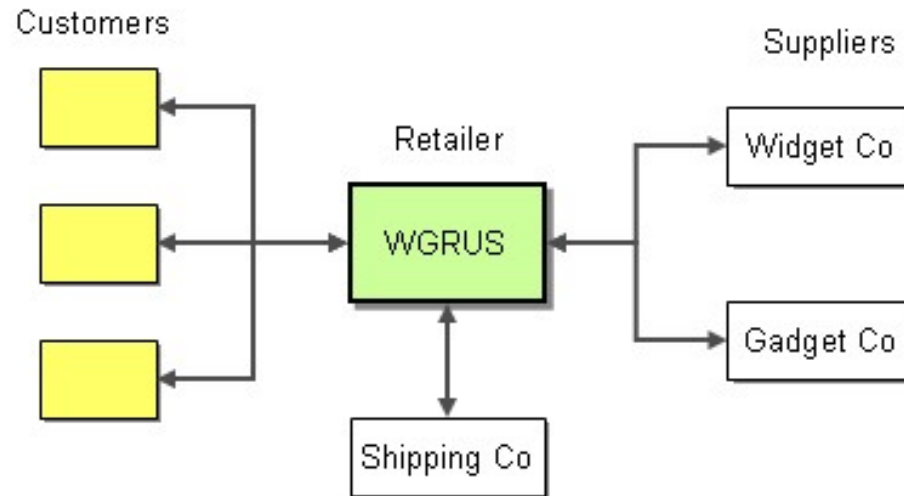
Message history



Message store



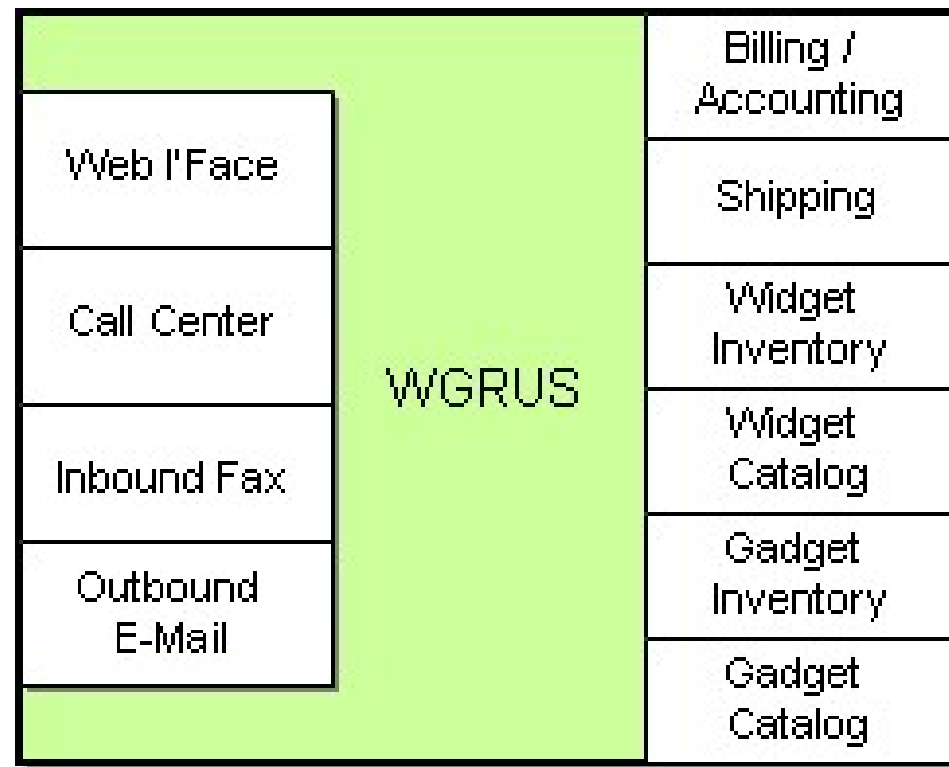
Example: Widgets&Gatchets 'R Us (WGRUS)



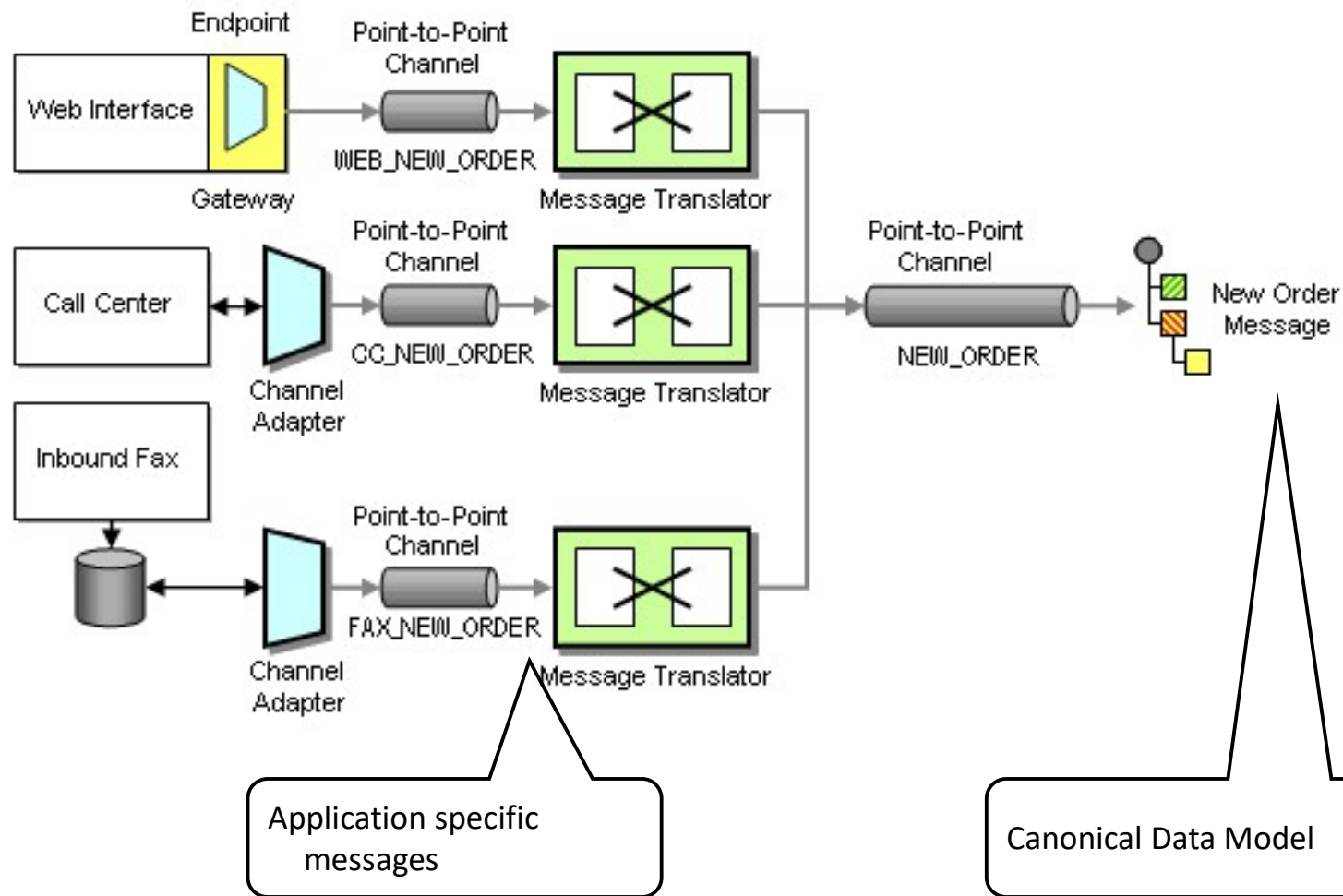
- **Take Orders:** Customers can place orders via Web, phone or fax
- **Process Orders:** Processing an order involves multiple steps, including verifying inventory, shipping the goods and invoicing the customer
- **Check Status:** Customers can check the order status
- **Change Address:** Customers can use a Web front-end to change their billing and shipping address
- **New Catalog:** The suppliers update their catalog periodically. WGRUS needs to update its pricing and availability based in the new catalogs.
- **Announcements:** Customers can subscribe to selective announcements from WGRUS.
- **Testing and Monitoring:** The operations staff needs to be able to monitor all individual components and the message flow between them.



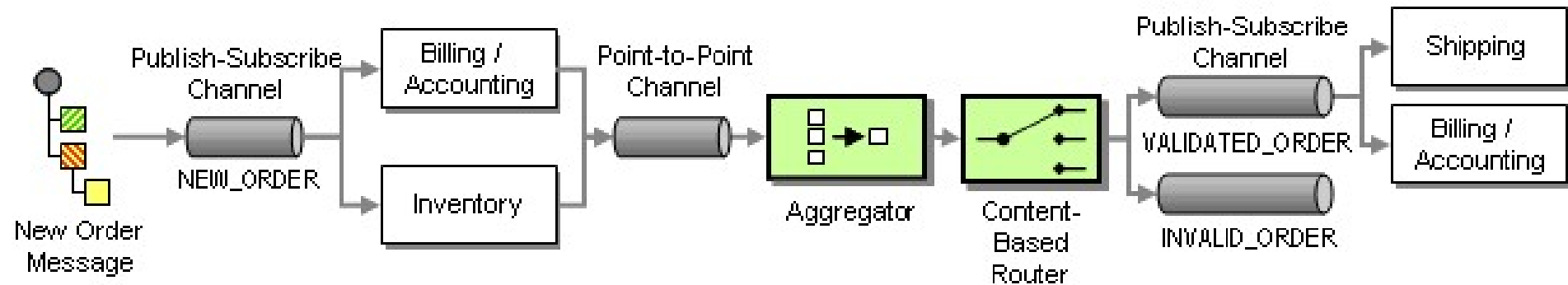
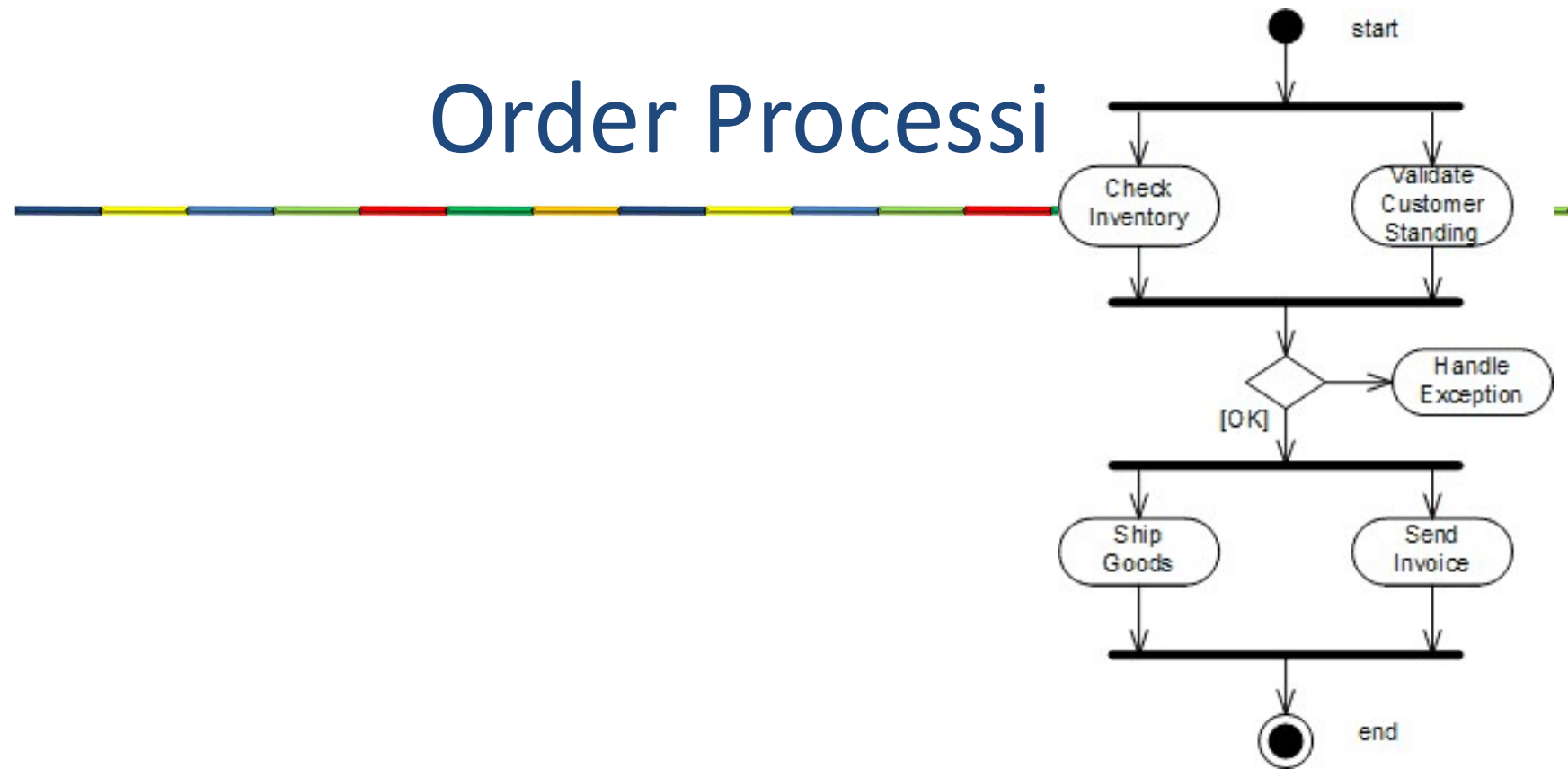
WGRUS internal IT infrastructure



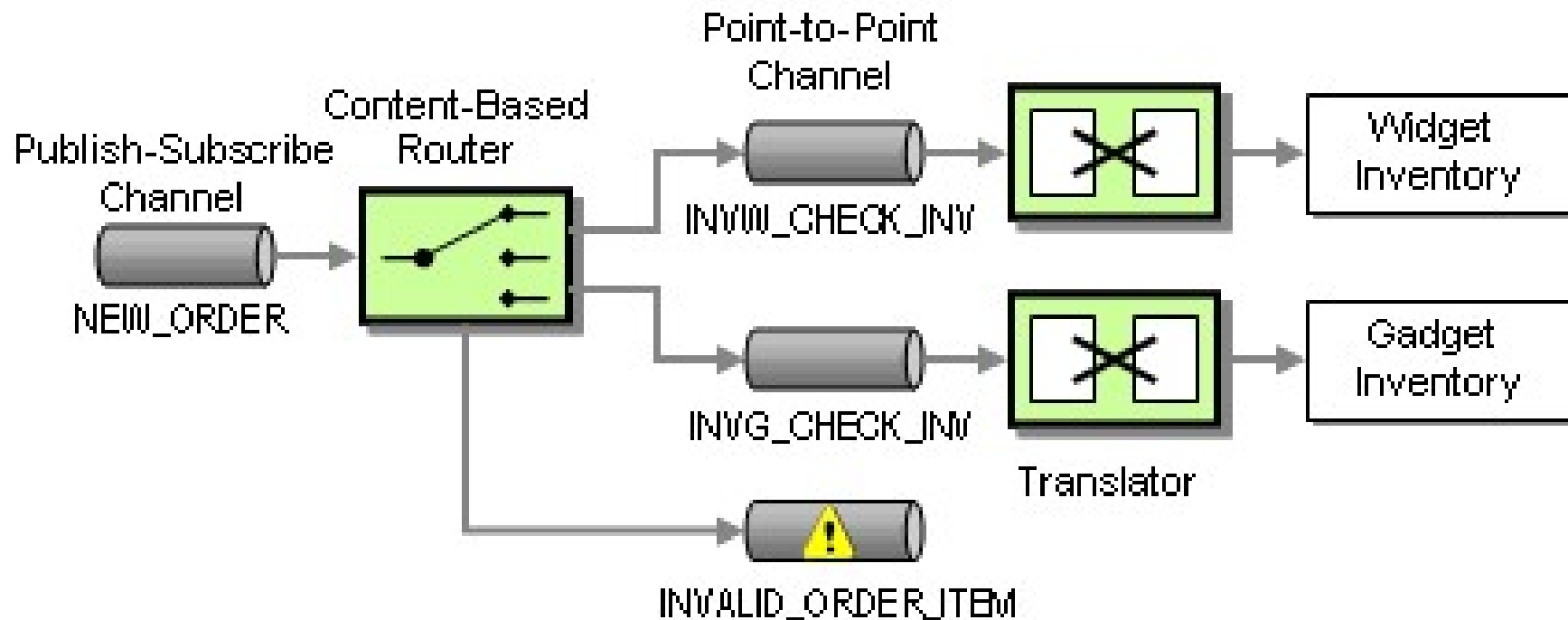
Taking orders from 3 different channels



Order Processi

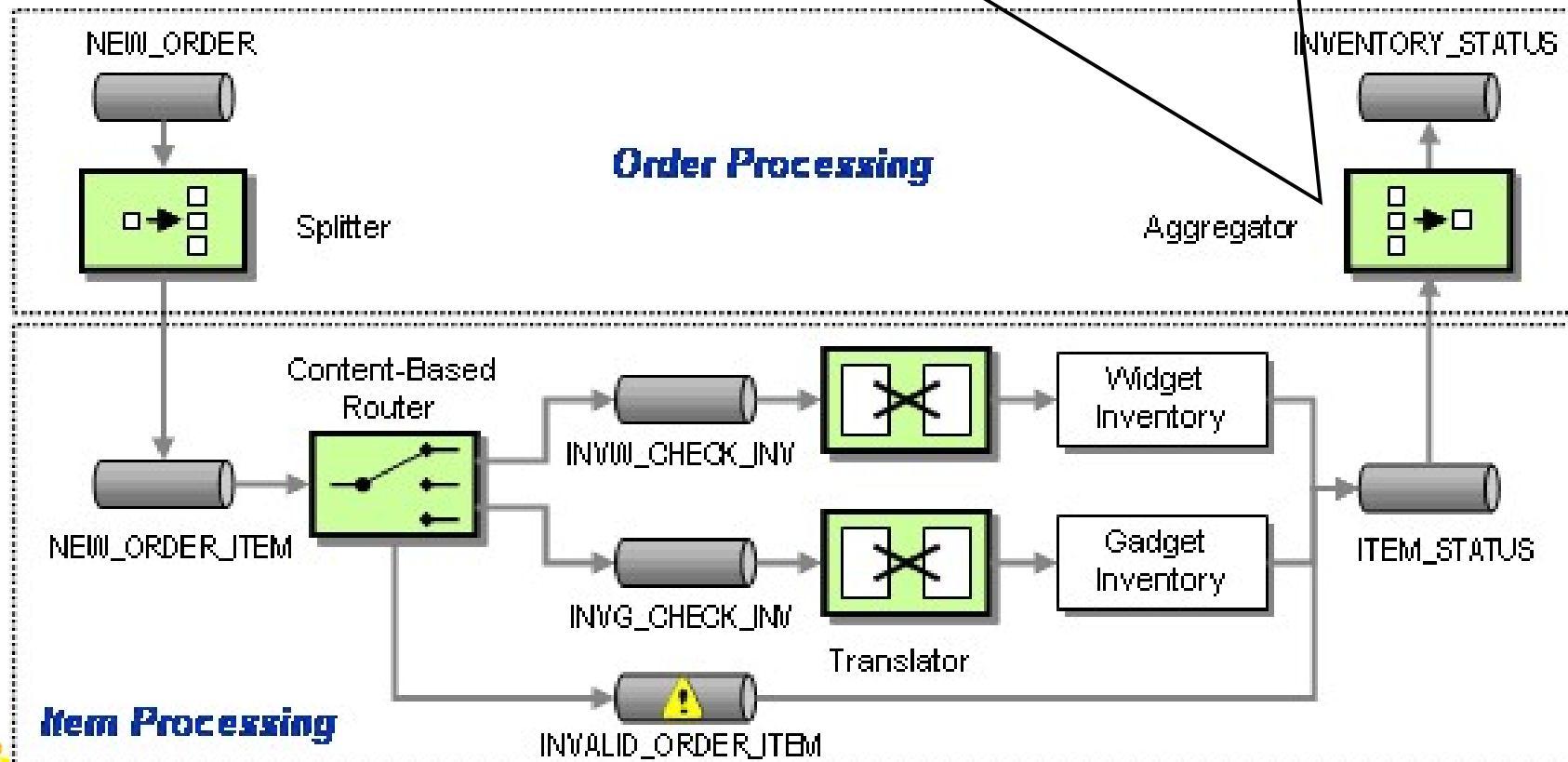


Routing the inventory request

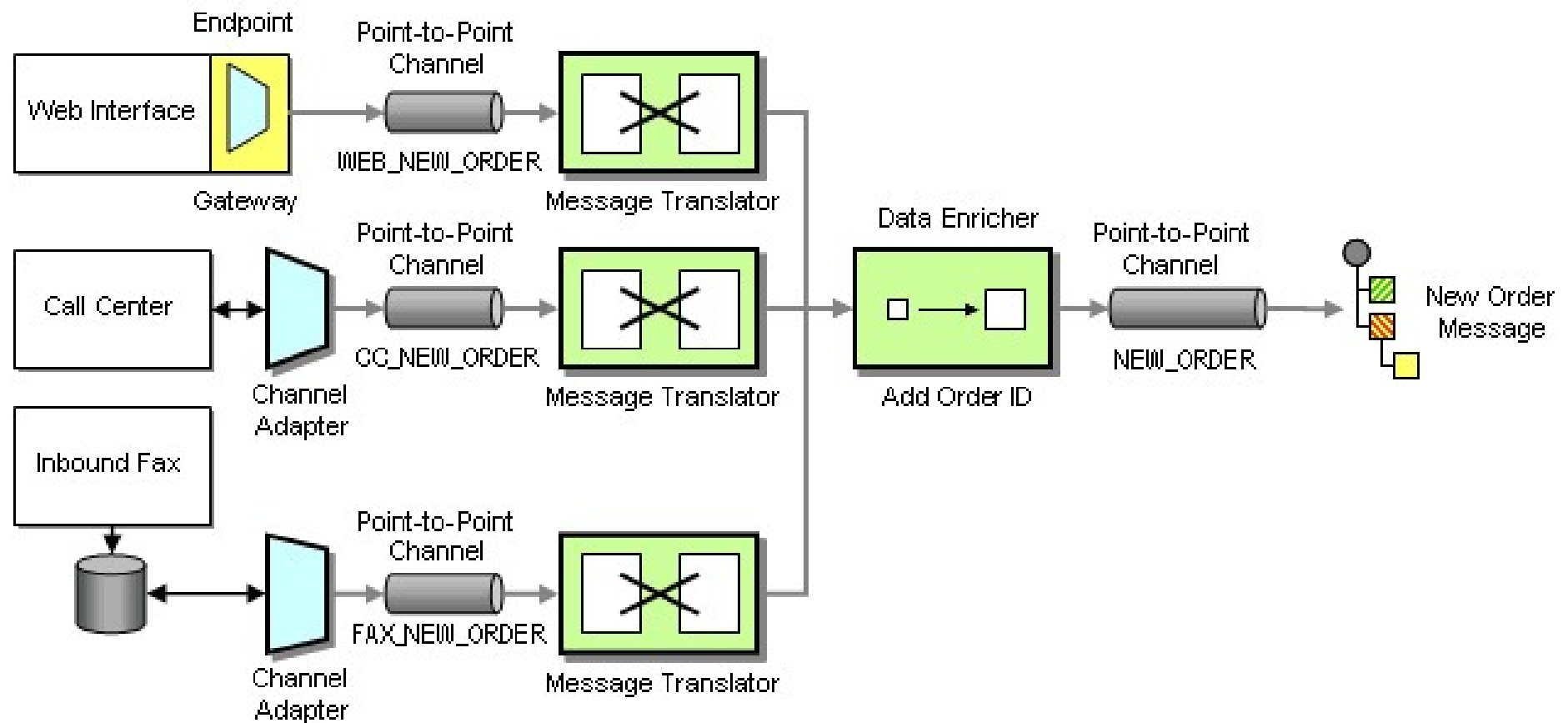


Orders can contain multiple items

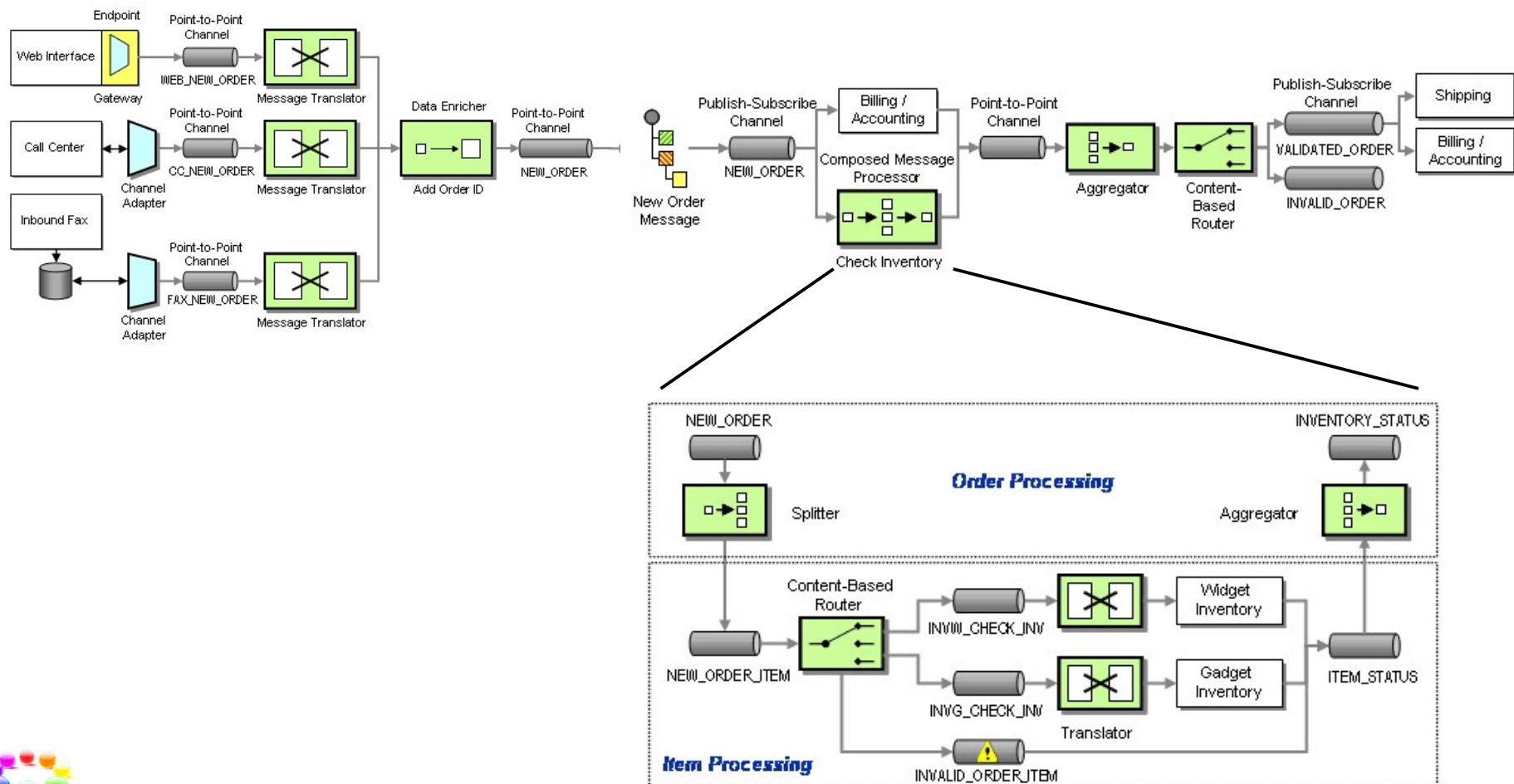
1. Correlation: which messages belong together? **We need an unique order ID**
2. Completeness: how do we know that all messages are received? **Count**
3. Aggregation algorithm: how do we combine the individual messages into one result message?
Append based on order ID



Add an unique order ID



Result so far

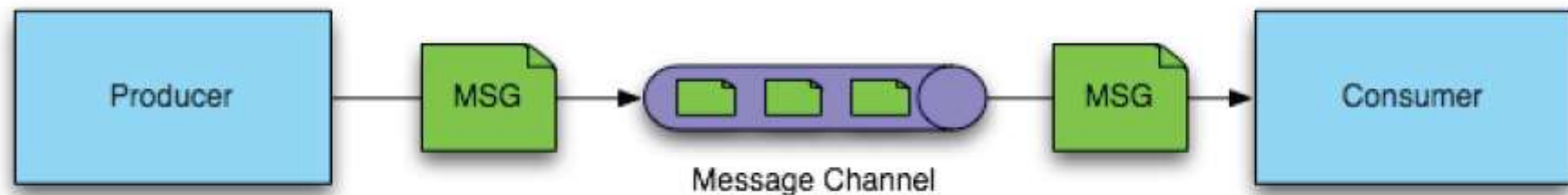


SPRING INTEGRATION

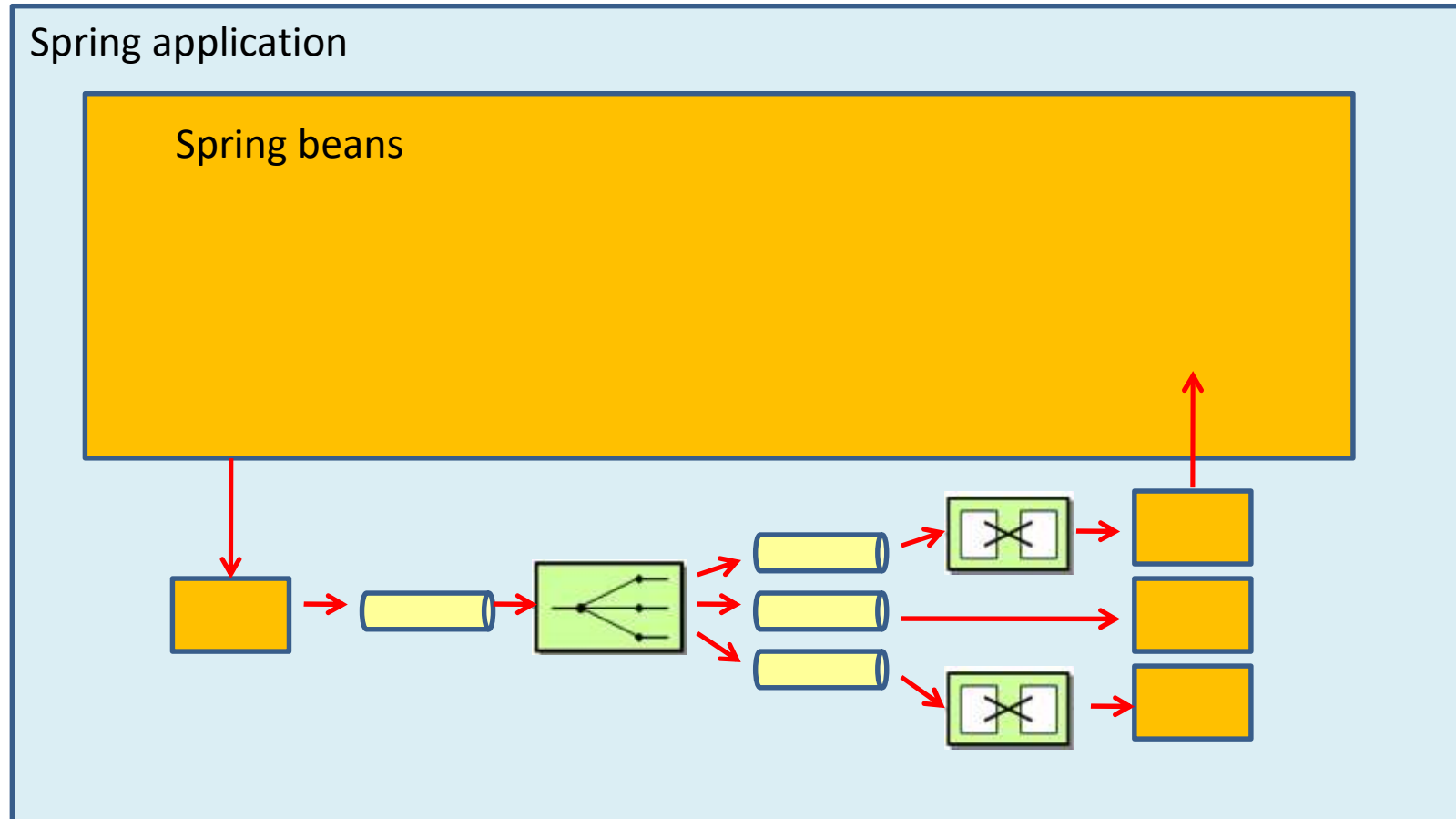


What is Spring Integration?

- Provides a simple model to implement complex enterprise integration solutions
- Facilitate asynchronous, parallel, message-driven behavior within a Spring-based application



Using Spring Integration

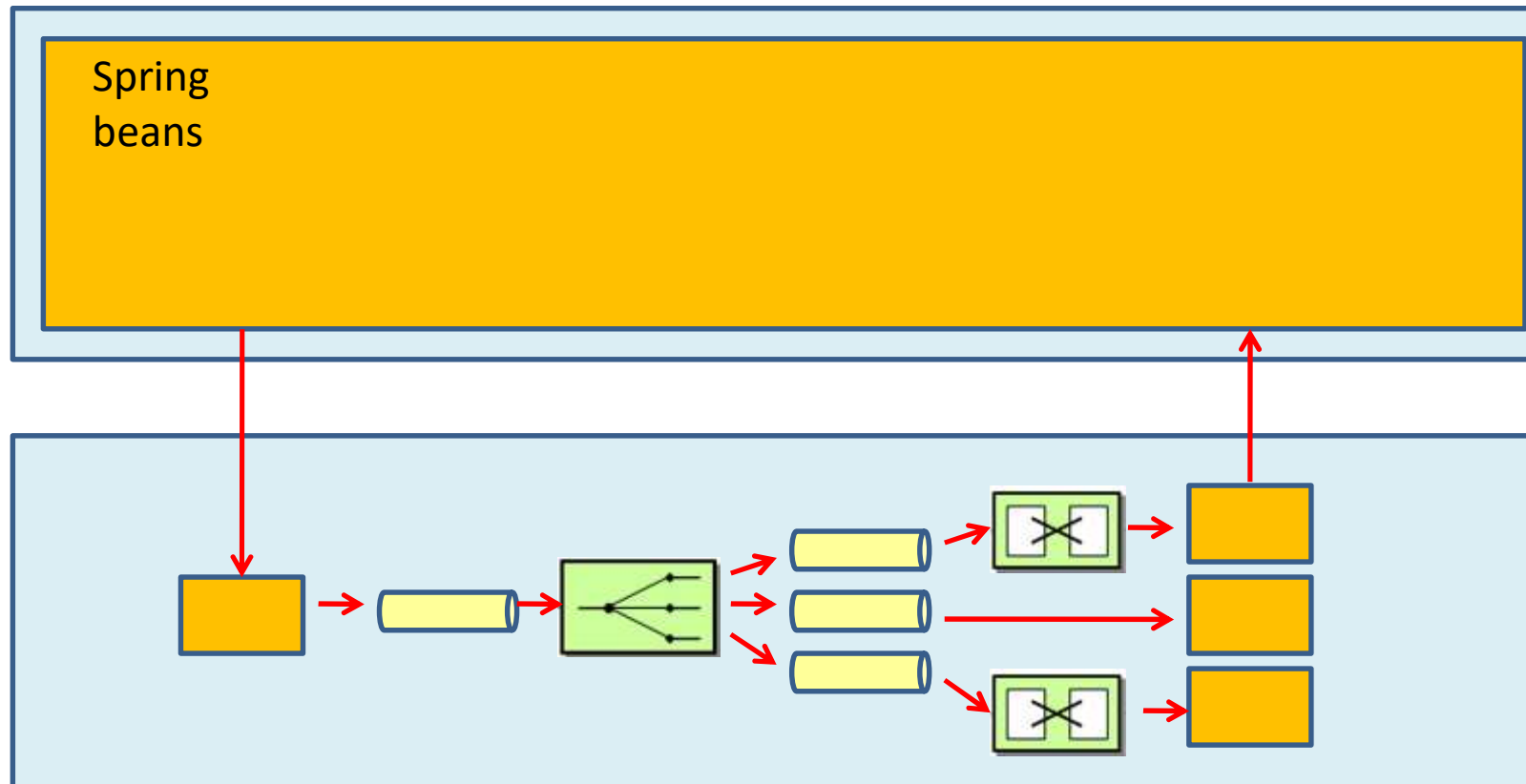


- Use SI inside your application



Using Spring Integration

Spring application

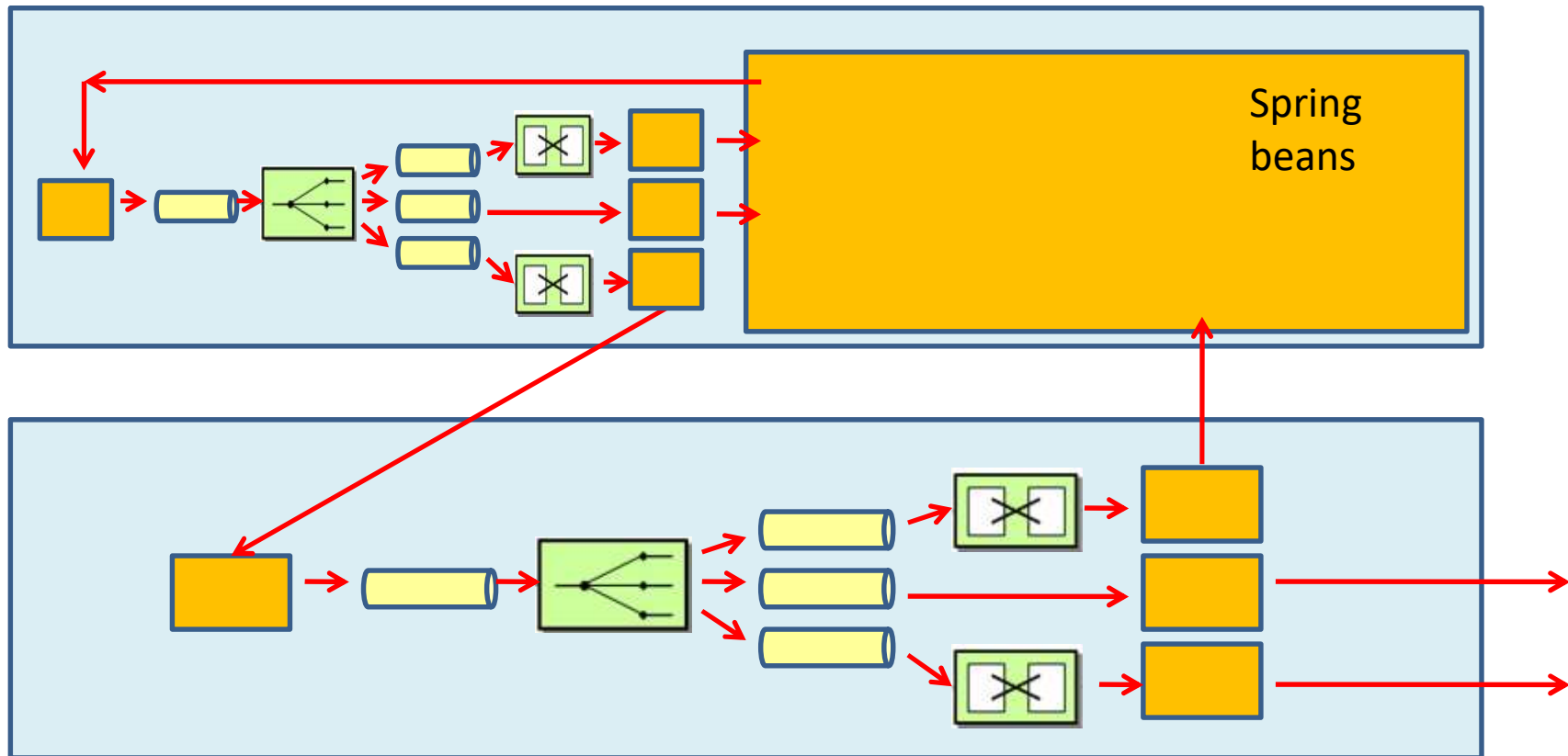


- Use SI outside your application



Using Spring Integration

Spring application



- Use SI inside and outside your application

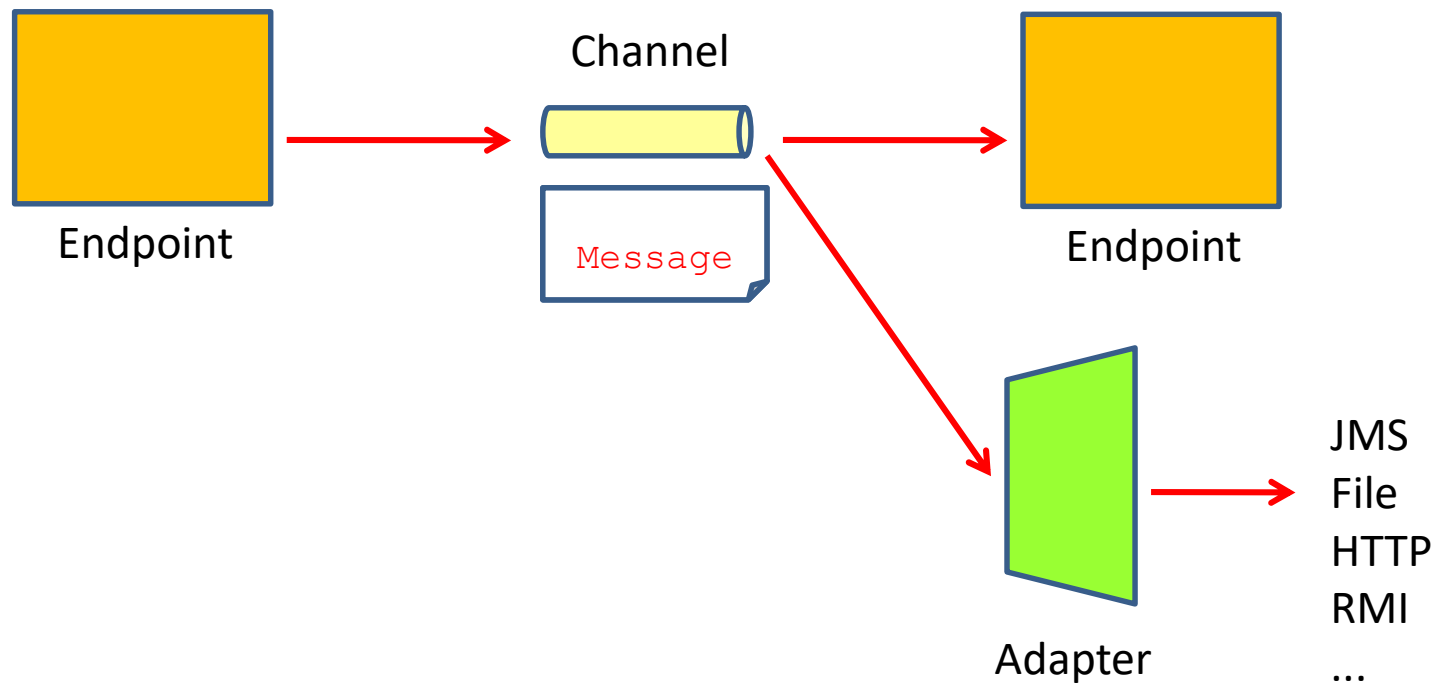


Difference with an ESB

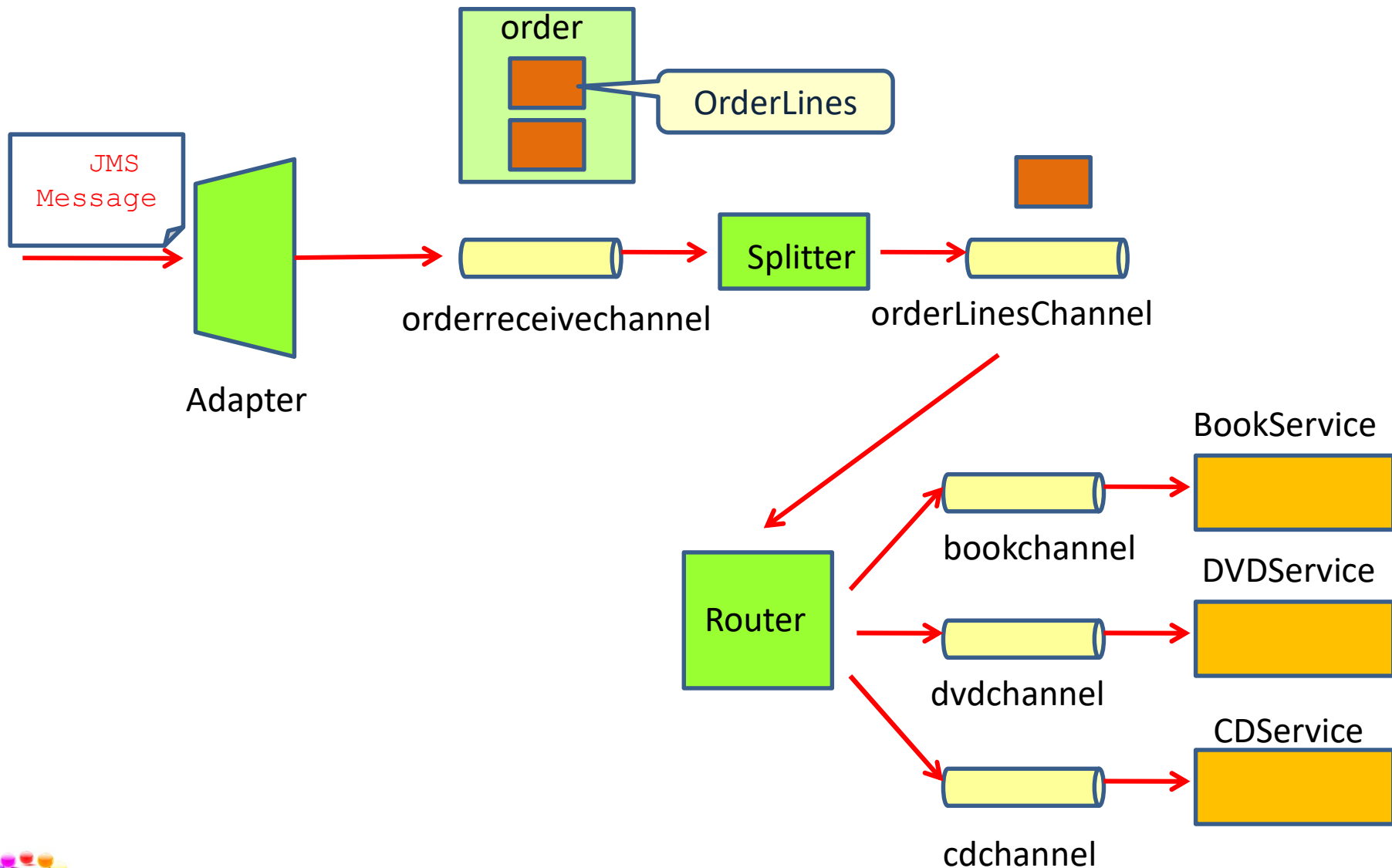
- ESB's run within its own VM
 - Spring Integration can run within an application
- You have to install ESB's
 - Spring integration is a library
- You have to start (and stop) ESB



Basic components



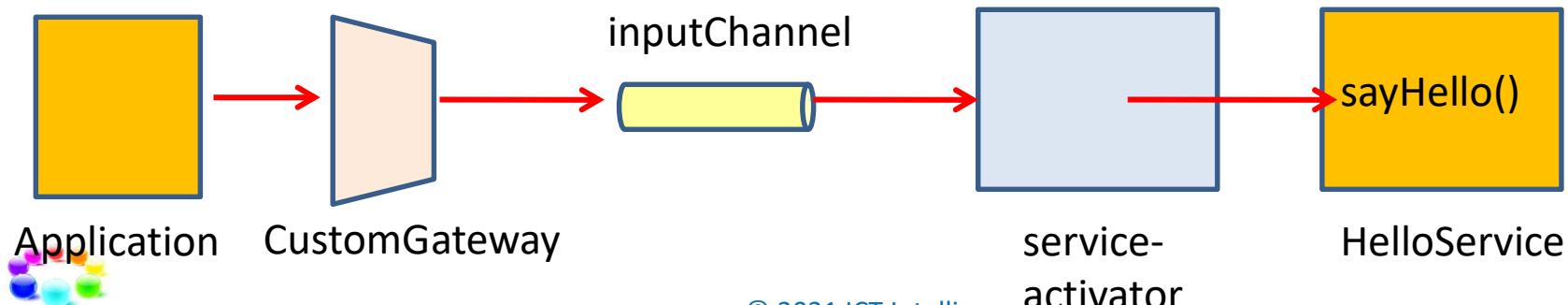
Spring Integration example



Spring integration Hello World

```
public class HelloService {  
    public void sayHello(String name){  
        System.out.println("Hello "+name);  
    }  
}
```

```
public interface CustomGateway {  
  
    public void process(String message);  
}
```



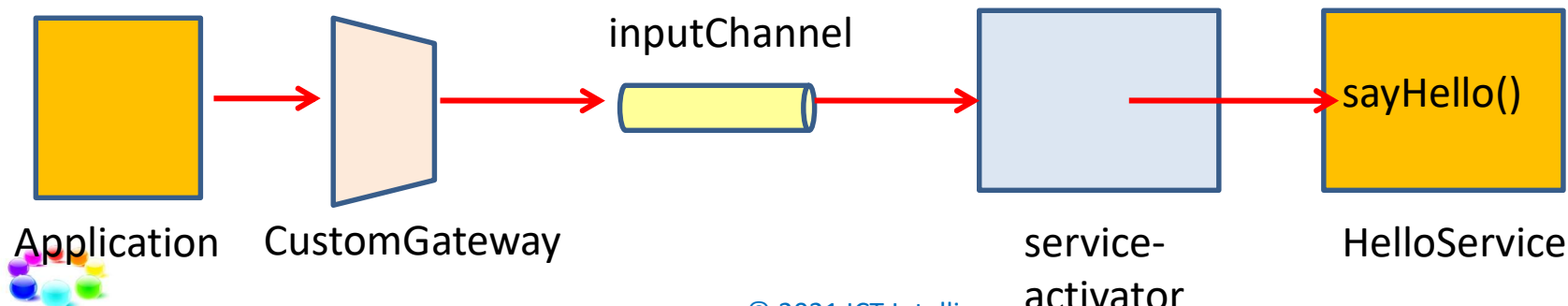
Integration-context.xml

```
<int:gateway service-interface="integration.CustomGateway"
              default-request-channel="inputChannel">
  <int:method name="process" />
</int:gateway>

<int:channel id="inputChannel" />

<int:service-activator
  input-channel="inputChannel" ref="helloService" method="sayHello" />

<bean id="helloService" class="integration.HelloService" />
```



The application

```
@SpringBootApplication
@ImportResource("integration-context.xml")
public class SpringIntegrationProjectApplication implements CommandLineRunner {

    @Autowired
    private CustomGateway gateway;

    public static void main(String[] args) {
        SpringApplication.run(SpringIntegrationProjectApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        gateway.process("World");
    }
}
```



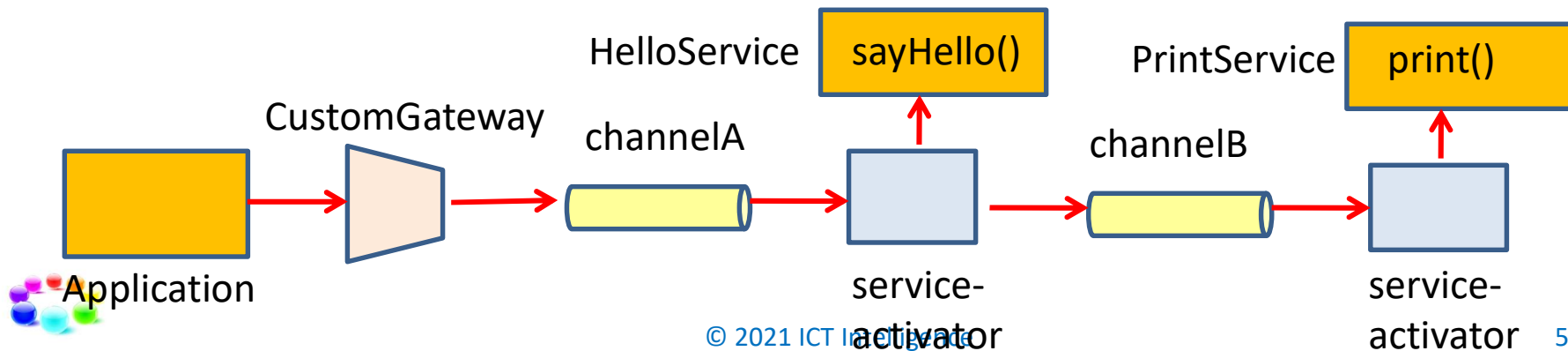
Extending the application

```
<int:gateway service-interface="integration.CustomGateway"
  default-request-channel="channelA">
  <int:method name="process" />
</int:gateway>

<channel id="channelA"/>
<channel id="channelB"/>

<service-activator input-channel="channelA"
  output-channel="channelB"
  ref="helloService"
  method="sayHello"/>
<service-activator input-channel="channelB"
  ref="printService"
  method="print"/>

<beans:bean id="helloService" class="integration.HelloService"/>
<beans:bean id="printService" class="integration.PrintService"/>
```



Extending the application

```
public class HelloService {  
  
    public String sayHello(String name) {  
        System.out.println("HelloService: receiving name "+name);  
        return "Hello "+ name;  
    }  
}
```

```
public class PrintService {  
  
    public void print(String message) {  
        System.out.println("Printing message: "+ message);  
    }  
}
```



Sending an Order

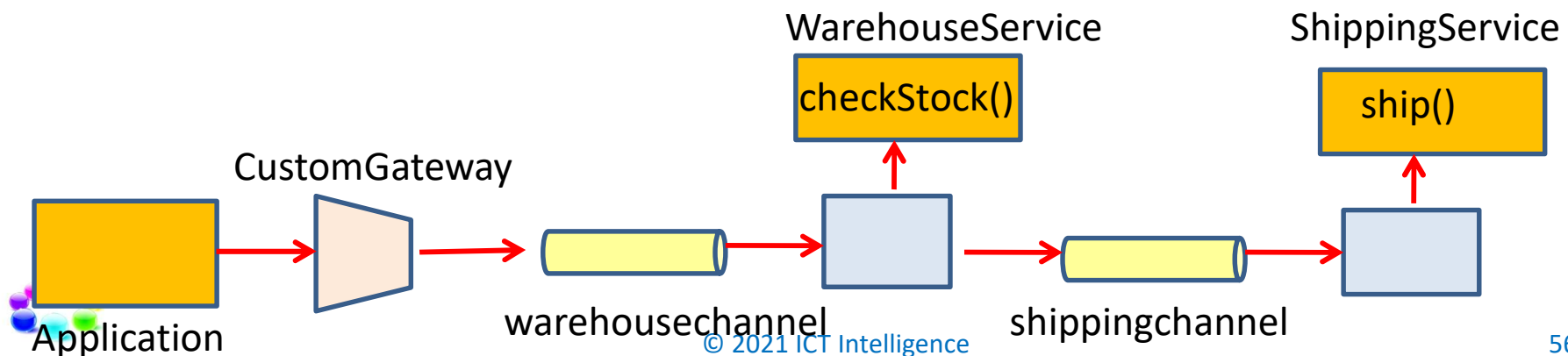
```
<int:gateway service-interface="integration.CustomGateway"
  default-request-channel="warehousechannel">
  <int:method name="process" />
</int:gateway>

<int:channel id="warehousechannel" />
<int:channel id="shippingchannel" />

<int:service-activator
  input-channel="warehousechannel" output-channel="shippingchannel"
  ref="warehouseservice" method="checkStock" />

<int:service-activator
  input-channel="shippingchannel" ref="shippingservice" method="ship" />

<bean id="warehouseservice" class="integration.WarehouseService" />
<bean id="shippingservice" class="integration.ShippingService" />
```



The services

```
public class WarehouseService {  
  
    public Order checkStock(Order order) {  
        System.out.println("WarehouseService: checking order "+order.toString());  
        return order;  
    }  
}
```

```
public class ShippingService {  
    public void ship(Order order) {  
        System.out.println("shipping: "+ order.toString());  
    }  
}
```

```
public class Order {  
    private String orderNumber;  
    private double amount;  
  
    public String toString(){  
        return "order: nr="+orderNumber+" amount="+amount;  
    }  
    ...  
}
```



The application

```
@SpringBootApplication
@ImportResource("integration-context.xml")
public class SpringIntegrationProjectApplication implements CommandLineRunner {

    @Autowired
    private CustomGateway gateway;

    public static void main(String[] args) {
        SpringApplication.run(SpringIntegrationProjectApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        Order order = new Order("H-234-X56", 1245.75);
        Message<Order> message = MessageBuilder.withPayload(order).build();
        gateway.process(message);
    }
}
```

WarehouseService: checking order order: nr=H-234-X56 amount=1245.75
shipping: order: nr=H-234-X56 amount=1245.75



Service Activator

```
<channel id="orderchannel" />

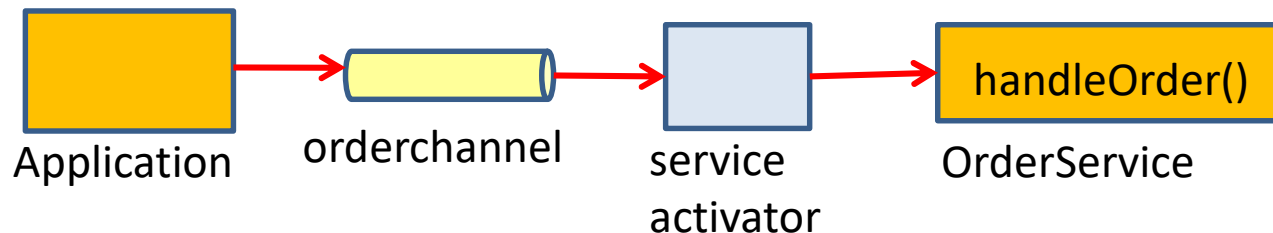
<service-activator input-channel="orderchannel" ref="orderservice"
    method="handleOrder" />

<beans:bean id="orderservice" class="integration.OrderService" />
```

```
public class OrderService {
    public void handleOrder(Message message) throws Exception {
        Order order = (Order) message.getPayload();
        System.out.println("OrderService receiving order: " + order.toString());
    }
}
```

The message as argument

OrderService receiving order: order: nr=H-234-X56 amount=1245.75



Service Activator

```
<channel id="orderchannel" />

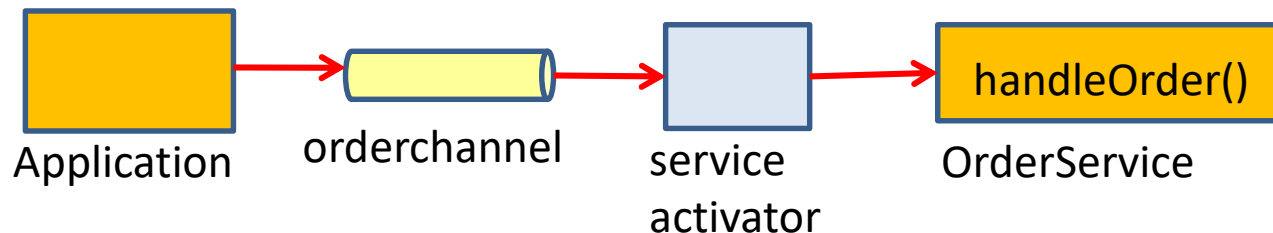
<service-activator input-channel="orderchannel" ref="orderservice"
    method="handleOrder" />

<beans:bean id="orderservice" class="integration.OrderService" />
```

The Order as argument

```
public class OrderService {
    public void handleOrder(Order order) throws Exception {
        System.out.println("OrderService receiving order: " + order.toString());
    }
}
```

OrderService receiving order: order: nr=H-234-X56 amount=1245.75



Multiple payloads

```
<channel id="orderchannel" />

<service-activator input-channel="orderchannel" ref="orderservice"
    method="handleOrder" />

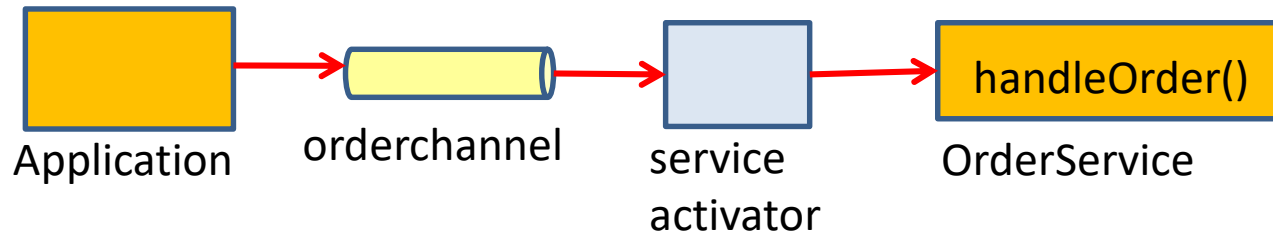
<beans:bean id="orderservice" class="integration.OrderService" />
```

Order

```
public class OrderService {
    public void handleOrder(Order order) throws Exception {
        System.out.println("OrderService receiving order: " + order.toString());
    }
}
```

RushOrder

```
public void handleOrder(RushOrder rushOrder) throws Exception {
    System.out.println("OrderService receiving rushorder: " +
        rushOrder.toString());
}
```



MESSAGE CHANNELS



Direct Channel

```
<channel id="inputChannel"/>
```

Default point-to-point
channel

- Point-to-point
- When there are multiple handlers subscribed to the same channel
 - A “round-robin” loadbalancer balances the messages
 - The loadbalancer will automatically send the message to a subsequent handler if the preceding handler throws an exception (failover)

```
<channel id="failFastChannel">  
  <dispatcher failover="false"/>  
</channel>
```

No failover

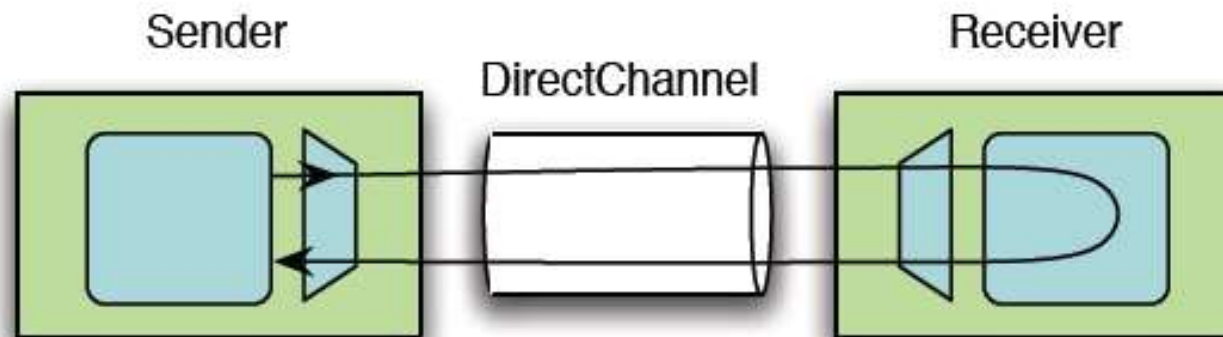
```
<channel id="channelWithFixedOrderSequenceFailover">  
  <dispatcher load-balancer="none"/>  
</channel>
```

No round-robin
balancer



Synchronous

- A direct default channel is synchronous



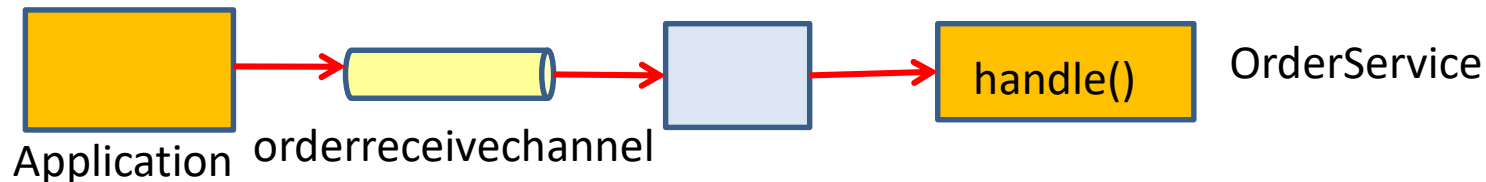
Synchronous

```
<channel id="orderreceivechannel" />

<service-activator input-channel="orderreceivechannel"
    ref="orderservice" method="handle" />
<beans:bean id="orderservice" class="integration.OrderService" />
```

```
public class OrderService {
    public void handle(Order order) throws Exception {
        System.out.println("OrderService receiving order: "+ order.toString());
        Thread.sleep(5000);
    }
}
```

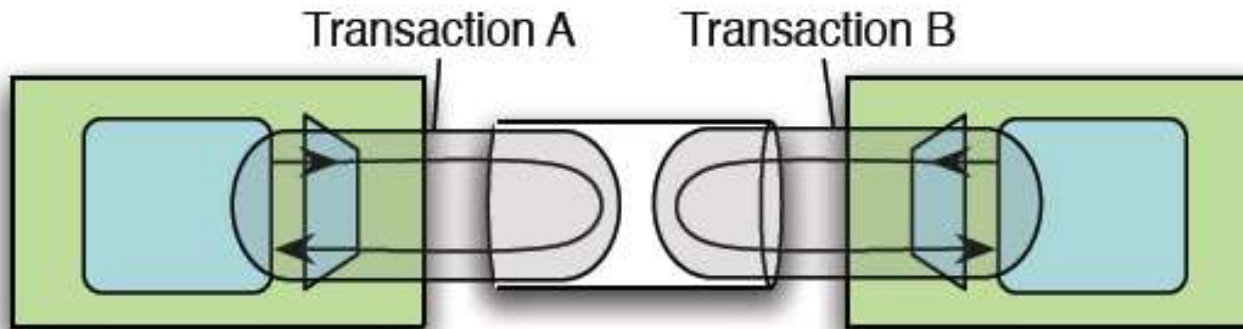
Sleep 5 seconds



```
time before sending message =8:54:15
OrderService receiving order: order: nr=H-234-X56 amount=1245.75
time after sending message =8:54:20
```

QueueChannel: Asynchronous

- A queue channel is asynchronous



QueueChannel

```
<channel id="orderreceivechannel" >  
  <queue capacity="25"/>  
</channel>
```

Add a queue

```
<service-activator input-channel="orderreceivechannel" ref="orderservice"  
  method="handle" >  
  <poller>  
    <interval-trigger interval="200"/>  
  </poller>  
</service-activator>
```

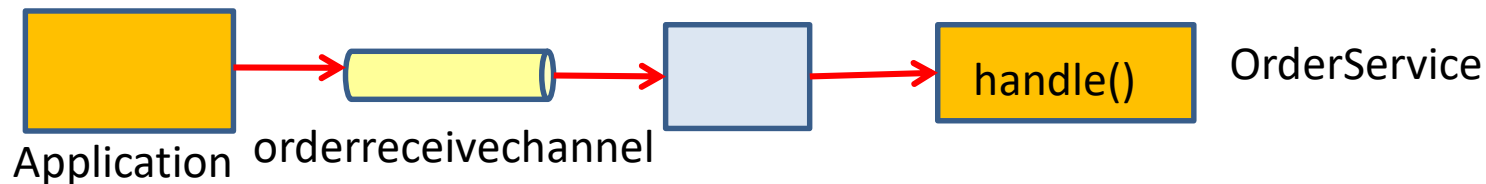
Now we need a poller

```
<beans:bean id="orderservice" class="integration.OrderService" />
```

time before sending message =9:22:30

time after sending message =9:22:30

OrderService receiving order: order: nr=H-234-X56 amount=1245.75



Poller

- We need a poller whenever the component need to be active
 - Getting a message from a QueueChannel
 - Reading files
 - Getting JMS messages

```
<poller>  
  <interval-trigger interval="200"/>  
</poller>
```

```
<poller>  
  <cron-trigger expression="30 * 9-17 * * MON-FRI"/>  
</poller>
```



Datatype channel

```
<channel id="numberChannel" datatype="java.lang.Number"/>
```

Datatype Channel that only accepts messages containing a certain payload type

```
<channel id="stringOrNumberChannel"  
  datatype="java.lang.String,java.lang.Number"/>
```

Accept multiple types



Point-to-point channel

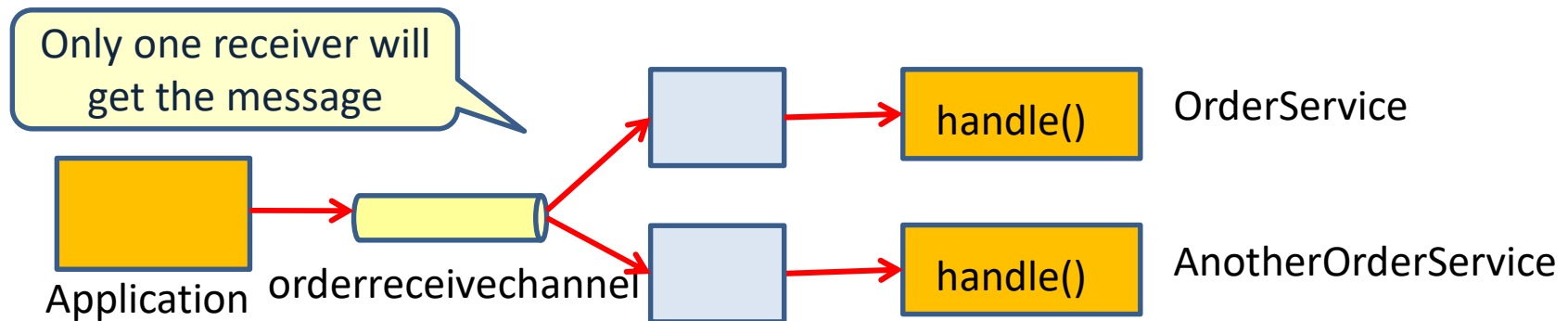
```
<channel id="orderreceivechannel" />

<service-activator input-channel="orderreceivechannel"
ref="orderservice" method="handle" />

<service-activator input-channel="orderreceivechannel"
ref="anotherorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="anotherorderservice" class="integration.AnotherOrderService" />
```

OrderService receiving order: order: nr=H-234-X56 amount=1245.75

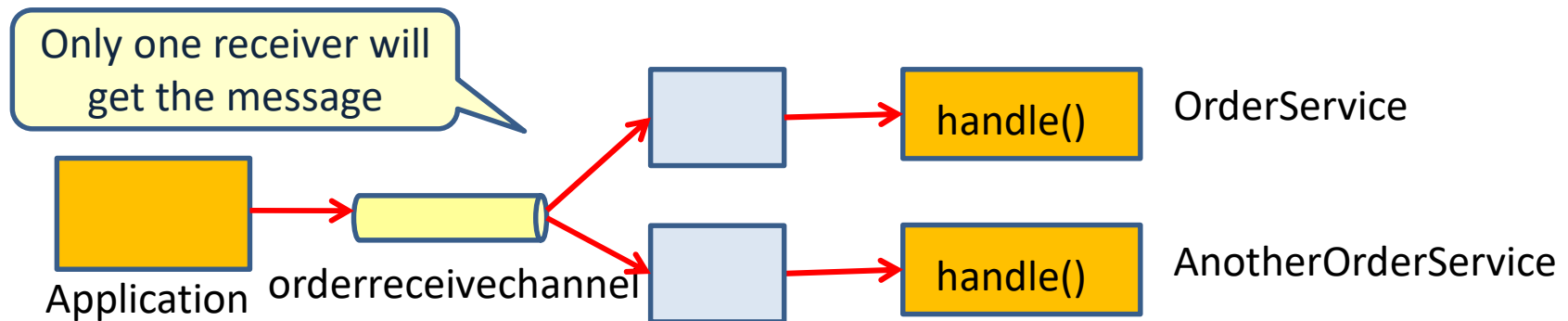


Point-to-point channel

```
public class OrderService {  
    public void handle(Order order) {  
        System.out.println("OrderService receiving order: "+ order.toString());  
    }  
}
```

```
public class AnotherOrderService {  
    public void handle(Order order) {  
        System.out.println("AnotherOrderService receiving order: "+ order.toString());  
    }  
}
```

OrderService receiving order: order: nr=H-234-X56 amount=1245.75



Publish-Subscribe channel

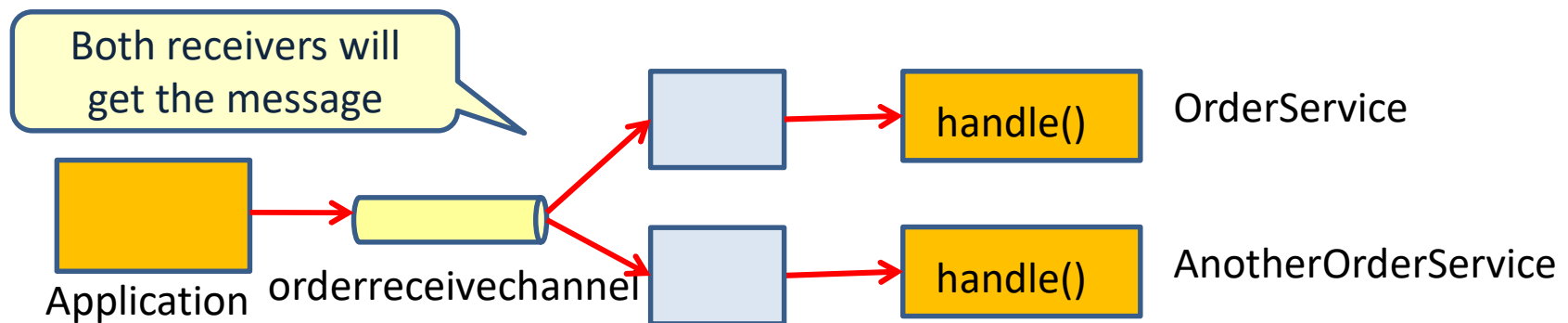
```
<publish-subscribe-channel id="orderreceivechannel" />

<service-activator input-channel="orderreceivechannel"
ref="orderservice" method="handle" />

<service-activator input-channel="orderreceivechannel"
ref="anotherorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="anotherorderservice" class="integration.AnotherOrderService" />
```

OrderService receiving order: order: nr=H-234-X56 amount=1245.75
AnotherOrderService receiving order: order: nr=H-234-X56 amount=1245.75

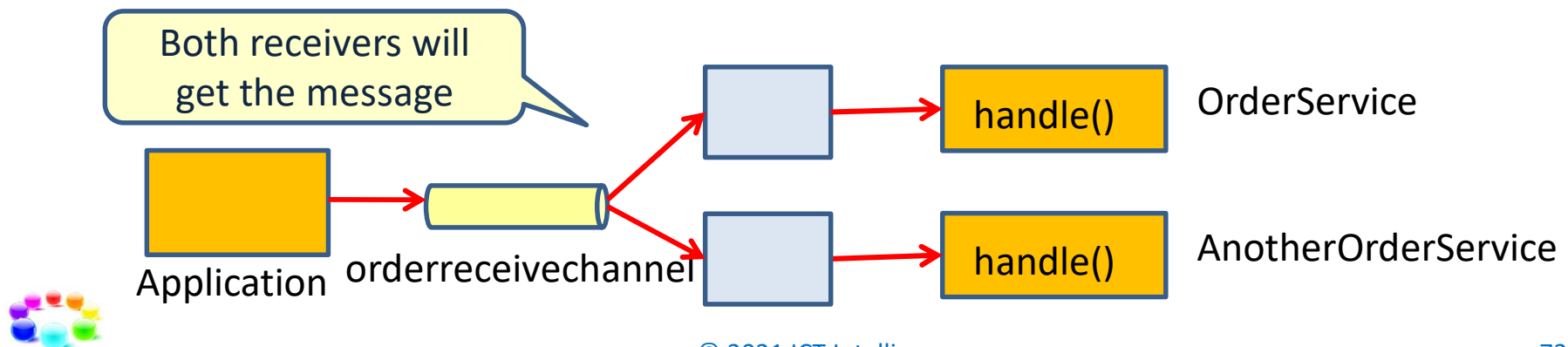


Pub-sub

```
public class OrderService {  
    public void handle(Order order) throws Exception {  
        System.out.println("OrderService receiving order: "+ order.toString());  
        Thread.sleep(5000);  
    }  
}
```

```
public class AnotherOrderService {  
    public void handle(Order order) throws Exception {  
        System.out.println("AnotherOrderService receiving order: "+ order.toString());  
        Thread.sleep(5000);  
    }  
}
```

OrderService receiving order: order: nr=H-234-X56 amount=1245.75
AnotherOrderService receiving order: order: nr=H-234-X56 amount=1245.75

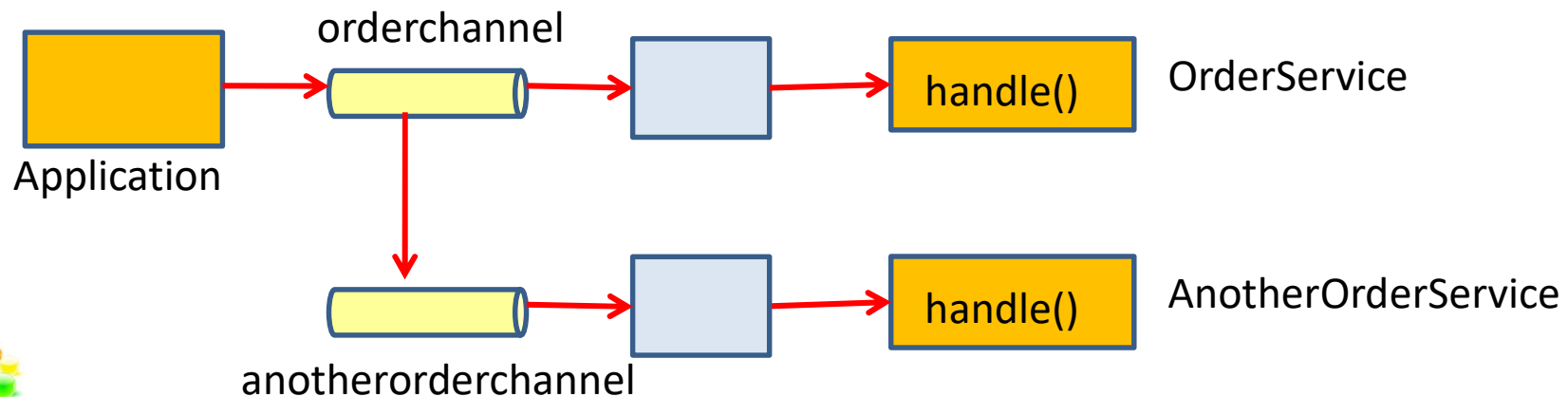


Wiretap

```
<channel id="orderchannel">
  <interceptors>
    <wire-tap channel="anotherorderchannel" />
  </interceptors>
</channel>
<channel id="anotherorderchannel" />

<service-activator input-channel="orderchannel"
  ref="orderservice" method="handle" />
<service-activator input-channel="anotherorderchannel"
  ref="anotherorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="anotherorderservice" class="integration.AnotherOrderService" />
```



ROUTER

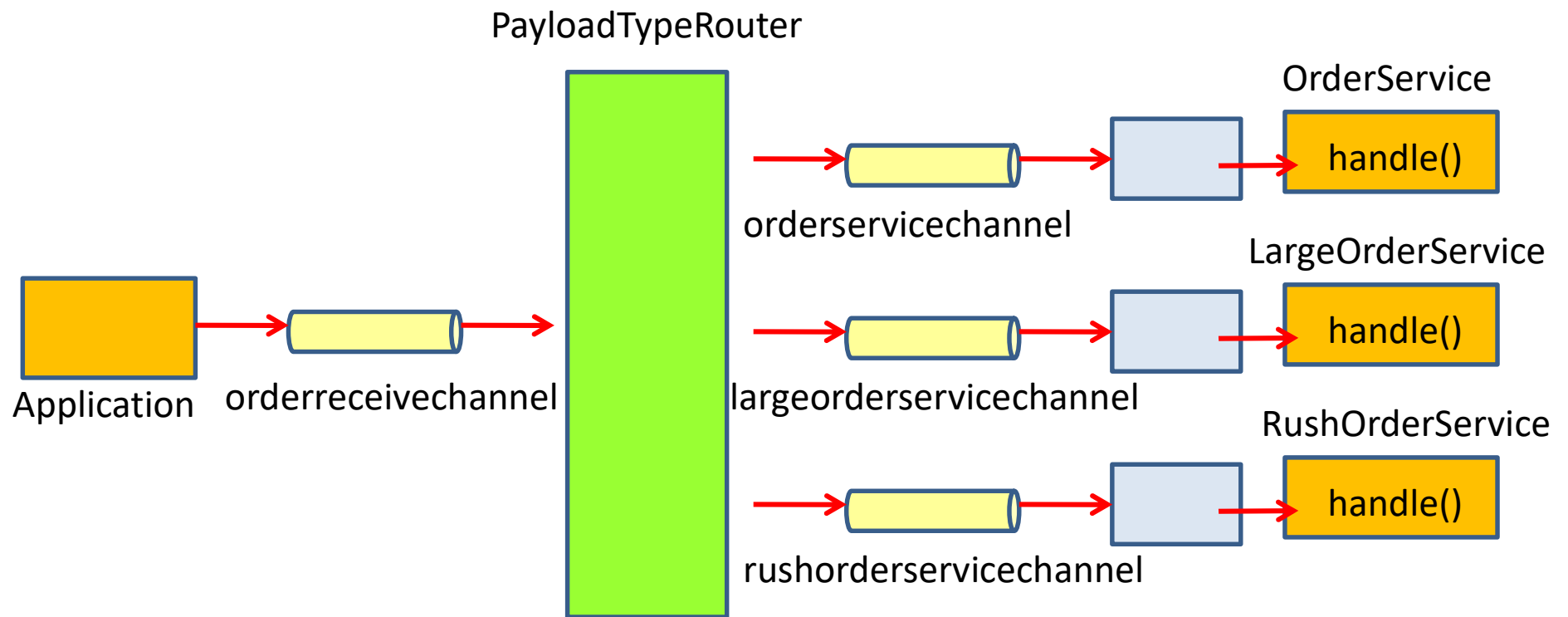


Routers

- Build-in routers
 - PayloadTypeRouter
 - HeaderValueRouter
 - RecipientListRouter
- Custom router



PayloadTypeRouter



PayloadTypeRouter

```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />
<channel id="rushorderservicechannel" />
<channel id="largeorderservicechannel" />

<payload-type-router input-channel="orderreceivechannel">
  <mapping type="integration.Order" channel="orderservicechannel" />
  <mapping type="integration.RushOrder" channel="rushorderservicechannel" />
  <mapping type="integration.LargeOrder" channel="largeorderservicechannel" />
</payload-type-router>

<service-activator input-channel="orderservicechannel"
  ref="orderservice" method="handle" />

<service-activator input-channel="rushorderservicechannel"
  ref="rushorderservice" method="handle" />

<service-activator input-channel="largeorderservicechannel"
  ref="largeorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="rushorderservice" class="integration.RushOrderService" />
<beans:bean id="largeorderservice" class="integration.LargeOrderService" />
```

The Payload types

```
public class Order {  
    private String orderNumber;  
    private double amount;  
  
    public String toString(){  
        return "order: nr="+orderNumber+" amount="+amount;  
    }  
    ...  
}
```

```
public class RushOrder extends Order{  
    public RushOrder(String orderNumber, double amount) {  
        super(orderNumber, amount);  
    }  
}
```

```
public class LargeOrder extends Order{  
    public LargeOrder(String orderNumber, double amount) {  
        super(orderNumber, amount);  
    }  
}
```



The services

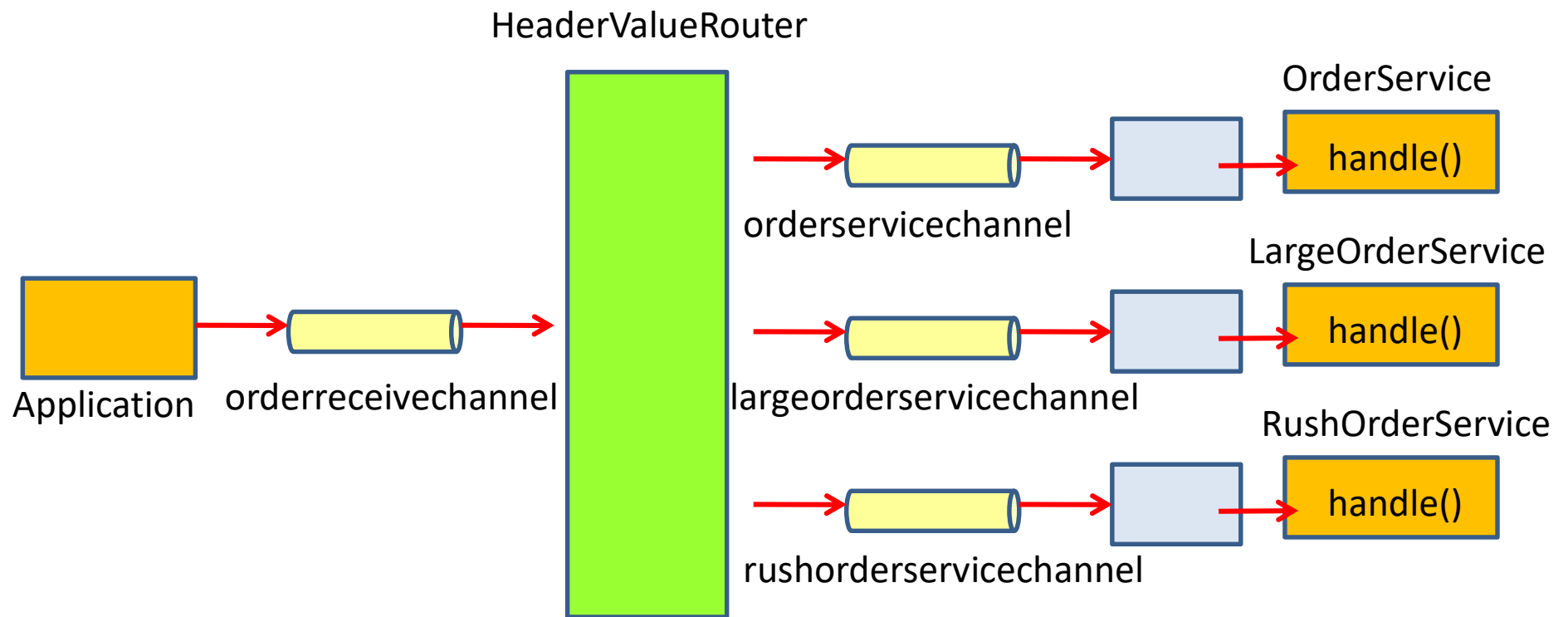
```
public class OrderService {  
    public void handle(Order order) {  
        System.out.println("OrderService receiving order: "+ order.toString());  
    }  
}
```

```
public class LargeOrderService {  
    public void handle(Order order) {  
        System.out.println("LargeOrderService receiving order: "+ order.toString());  
    }  
}
```

```
public class RushOrderService {  
    public void handle(Order order) {  
        System.out.println("RushOrderService receiving order: "+ order.toString());  
    }  
}
```



HeaderValueRouter



HeaderValueRouter

```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />
<channel id="rushorderservicechannel" />
<channel id="largeorderservicechannel" />

<header-value-router input-channel="orderreceivechannel"
    header-name="orderType">
    <mapping value="normal" channel="orderservicechannel" />
    <mapping value="rush" channel="rushorderservicechannel" />
    <mapping value="large" channel="largeorderservicechannel" />
</header-value-router>

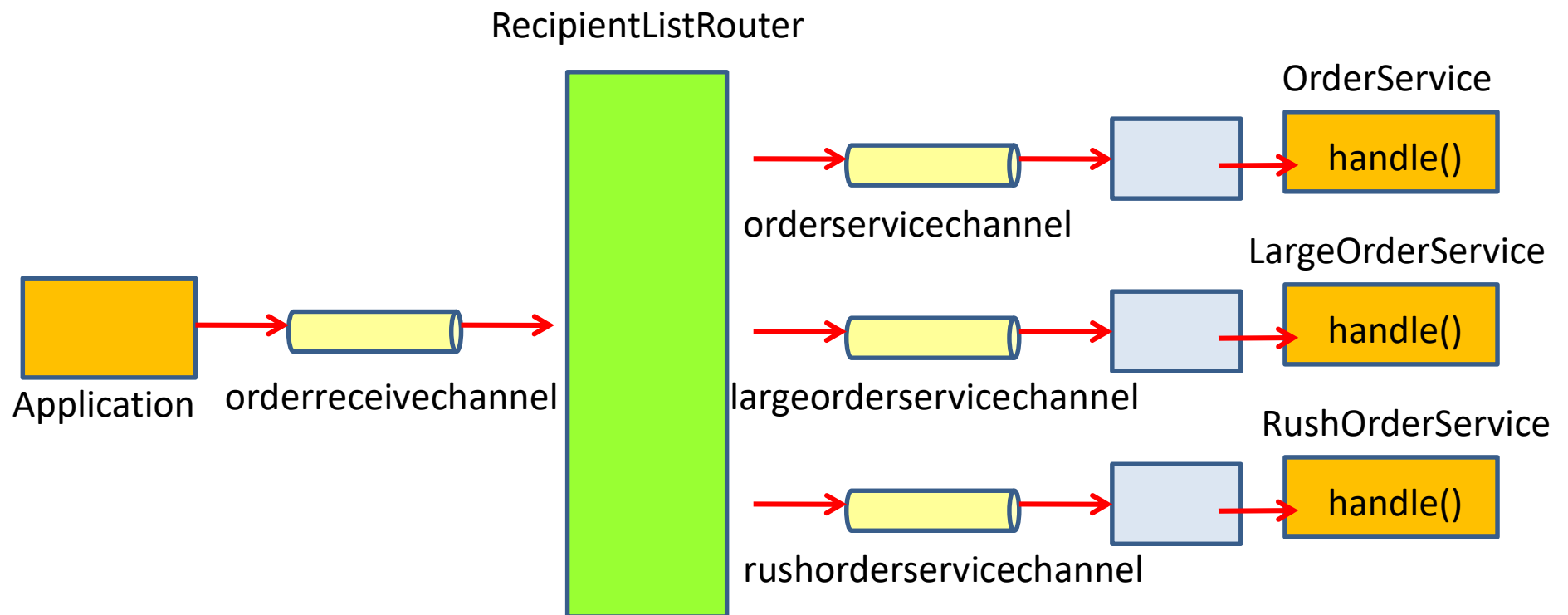
<service-activator input-channel="orderservicechannel"
    ref="orderservice" method="handle" />

<service-activator input-channel="rushorderservicechannel"
    ref="rushorderservice" method="handle" />

<service-activator input-channel="largeorderservicechannel"
    ref="largeorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="rushorderservice" class="integration.RushOrderService" />
<beans:bean id="largeorderservice" class="integration.LargeOrderService" />
```

RecipientListRouter



RecipientListRouter

```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />
<channel id="rushorderservicechannel" />
<channel id="largeorderservicechannel" />

<recipient-list-router id="customRouter" input-channel="orderreceivechannel"
    apply-sequence="true">
    <recipient channel="orderservicechannel" />
    <recipient channel="rushorderservicechannel" />
    <recipient channel="largeorderservicechannel" />
</recipient-list-router>

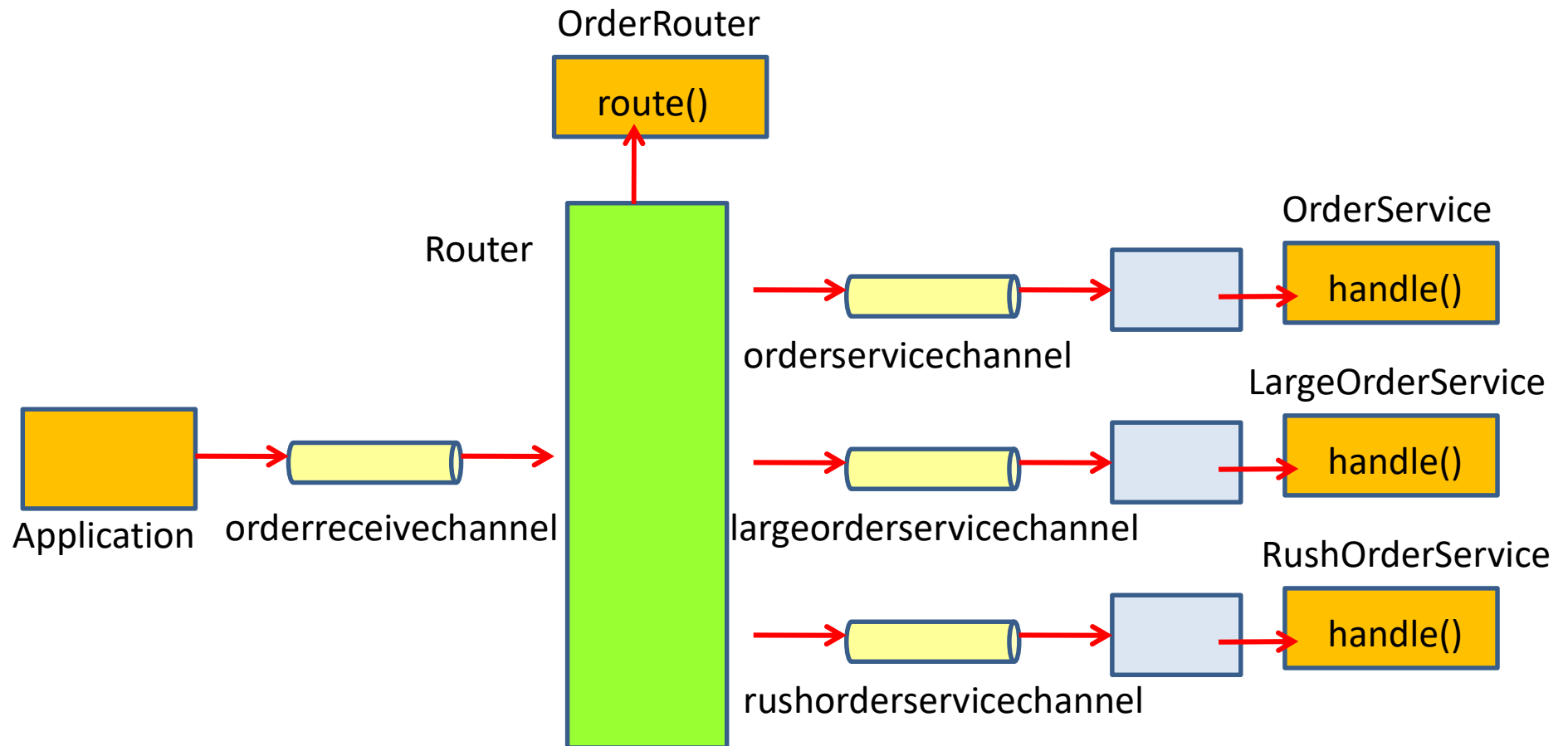
<service-activator input-channel="orderservicechannel"
    ref="orderservice" method="handle" />

<service-activator input-channel="rushorderservicechannel"
    ref="rushorderservice" method="handle" />

<service-activator input-channel="largeorderservicechannel"
    ref="largeorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="rushorderservice" class="integration.RushOrderService" />
<beans:bean id="largeorderservice" class="integration.LargeOrderService" />
```

Custom Router bean



Custom Router bean

```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />
<channel id="rushorderservicechannel" />
<channel id="largeorderservicechannel" />

<router method="route" input-channel="orderreceivechannel">
  <beans:bean class="integration.OrderRouter" />
</router>

<service-activator input-channel="orderservicechannel"
ref="orderservice" method="handle" />

<service-activator input-channel="rushorderservicechannel"
ref="rushorderservice" method="handle" />

<service-activator input-channel="largeorderservicechannel"
ref="largeorderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="rushorderservice" class="integration.RushOrderService" />
<beans:bean id="largeorderservice" class="integration.LargeOrderService" />
```



The router bean

```
public class OrderRouter {  
    public String route(Order order) {  
        String destinationChannel = null;  
        if (order.isRush())  
            destinationChannel = "rushorderservicechannel";  
        else if (order.getAmount() > 20000)  
            destinationChannel = "largeorderservicechannel";  
        else  
            destinationChannel = "orderservicechannel";  
        return destinationChannel;  
    }  
}
```

RushOrderService receiving order: order: nr=H-234-X56 amount=1245.75
OrderService receiving order: order: nr=H-234-X57 amount=600.65
LargeOrderService receiving order: order: nr=H-234-X58 amount=50600.65



The router bean: multiple return values

```
public class OrderRouter {  
    public List<String> route(Order order) {  
        List<String> destinationChannels = new ArrayList<String>();  
        if (order.isRush())  
            destinationChannels.add("rushorderservicechannel");  
        if (order.getAmount() > 20000)  
            destinationChannels.add("largeorderservicechannel");  
        destinationChannels.add("orderservicechannel");  
        return destinationChannels;  
    }  
}
```

RushOrderService receiving order: order: nr=H-234-X56 amount=1245.75
OrderService receiving order: order: nr=H-234-X56 amount=1245.75
OrderService receiving order: order: nr=H-234-X57 amount=600.65
RushOrderService receiving order: order: nr=H-234-X58 amount=50600.65
LargeOrderService receiving order: order: nr=H-234-X58 amount=50600.65
OrderService receiving order: order: nr=H-234-X58 amount=50600.65



FILTER



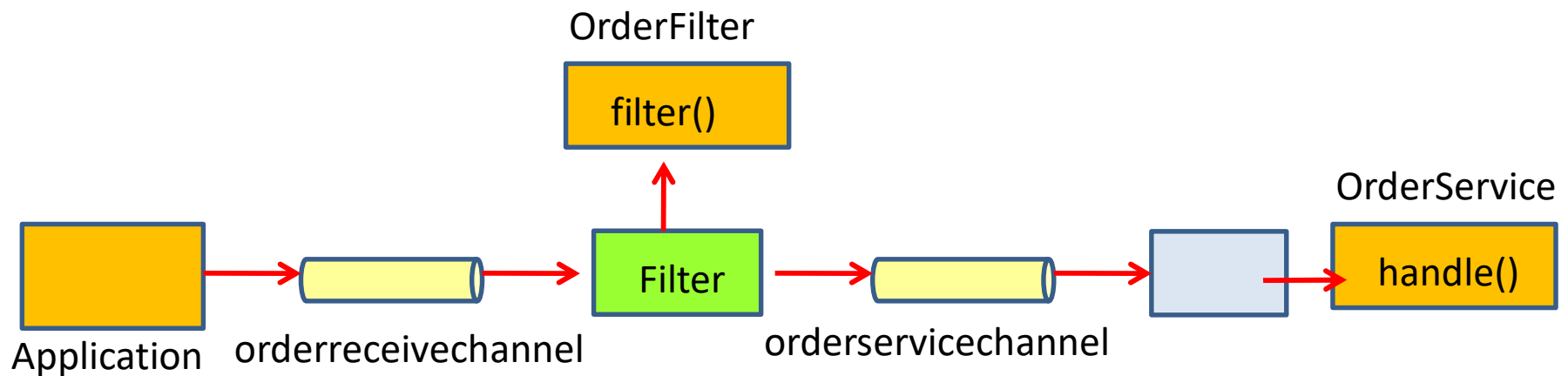
Filter

```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />

<filter input-channel="orderreceivechannel" output-channel="orderservicechannel"
        ref="orderfilter" method="filter"/>

<service-activator input-channel="orderservicechannel"
                   ref="orderservice" method="handle" />

<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="orderfilter" class="integration.OrderFilter" />
```



The Filter class

```
public class OrderFilter {  
    public boolean filter(Order order) {  
        if (order.getAmount() > 800)  
            return true;  
        else  
            return false;  
    }  
}
```



What to do with rejected messages?

```
<filter input-channel="orderreceivechannel" output-channel="orderservicechannel"
        ref="orderfilter" method="filter" throw-exception-on-rejection="true"/>
```

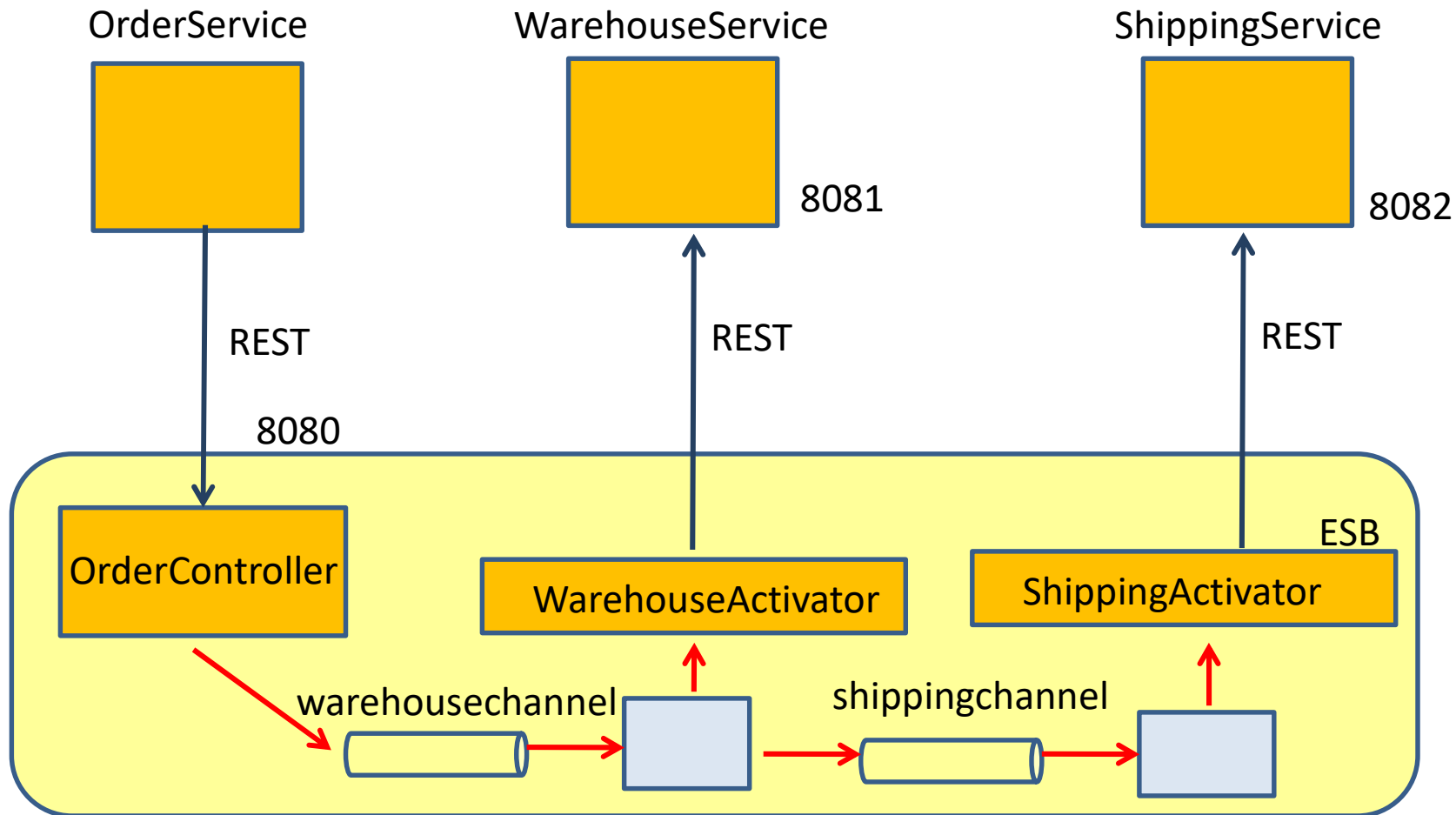
Throw an exception if
a message is rejected

```
<filter input-channel="orderreceivechannel" output-channel="orderservicechannel"
        ref="orderfilter" method="filter" discard-channel="rejectedMessages"/>
```

Send rejected
messages to another
channel



ESB with spring integration



ESB configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/integration"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/integration
    http://www.springframework.org/schema/integration/spring-integration.xsd">

  <channel id="warehousechannel"/>
  <channel id="shippingchannel"/>

  <service-activator input-channel="warehousechannel"
    output-channel="shippingchannel"
    ref="warehouseservice"
    method="checkStock"/>

  <service-activator input-channel="shippingchannel"
    ref="shippingservice"
    method="ship"/>

  <beans:bean id="warehouseservice" class="esb.WarehouseActivator"/>
  <beans:bean id="shippingservice" class="esb.ShippingActivator"/>

</beans:beans>
```

OrderController

```
@RestController
public class OrderController {
    @Autowired
    @Qualifier("warehousechannel")
    MessageChannel warehouseChannel;

    @PostMapping("/orders")
    public ResponseEntity<?> receiveOrder(@RequestBody Order order) {
        Message<Order> orderMessage = MessageBuilder.withPayload(order).build();
        warehouseChannel.send(orderMessage);
        return new ResponseEntity<Order>(order, HttpStatus.OK);
    }
}
```



The activator beans

```
public class WarehouseActivator {
```

```
    @Autowired
```

```
    RestTemplate restTemplate;
```

```
    public Order checkStock(Order order) {
```

```
        System.out.println("WarehouseService: checking order "+order.toString());
```

```
        restTemplate.postForLocation("http://localhost:8082/orders", order);
```

```
        return order;
```

```
    }
```

```
}
```

```
public class ShippingActivator {
```

```
    @Autowired
```

```
    RestTemplate restTemplate;
```

```
    public void ship(Order order) {
```

```
        System.out.println("shipping: "+ order.toString());
```

```
        restTemplate.postForLocation("http://localhost:8081/orders", order);
```

```
    }
```

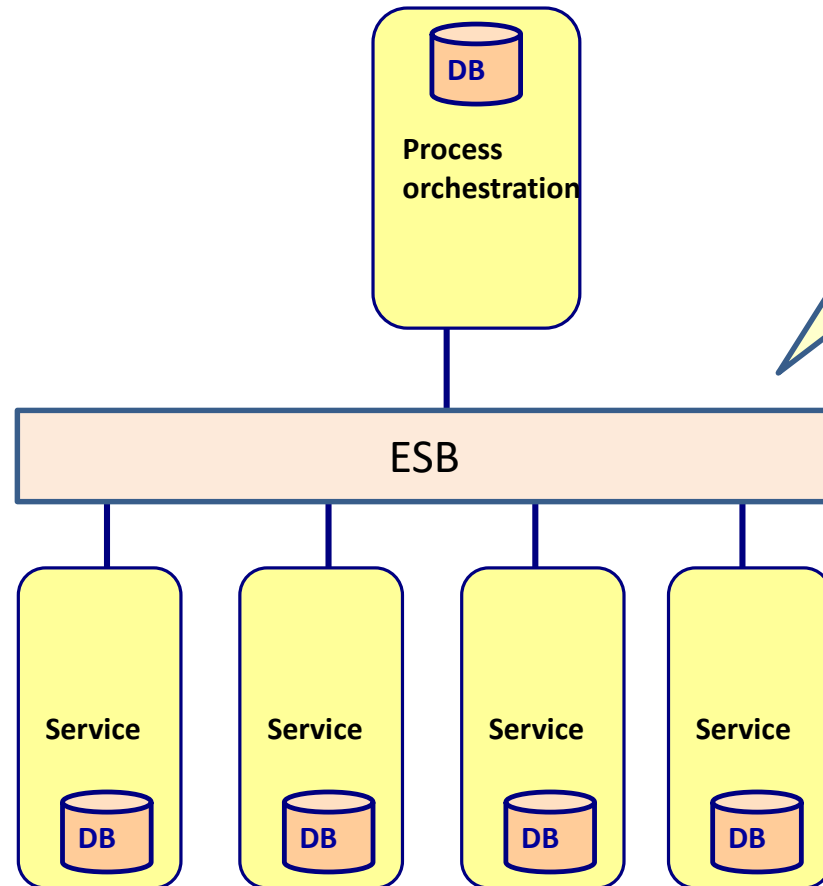
```
}
```



SUMMARY



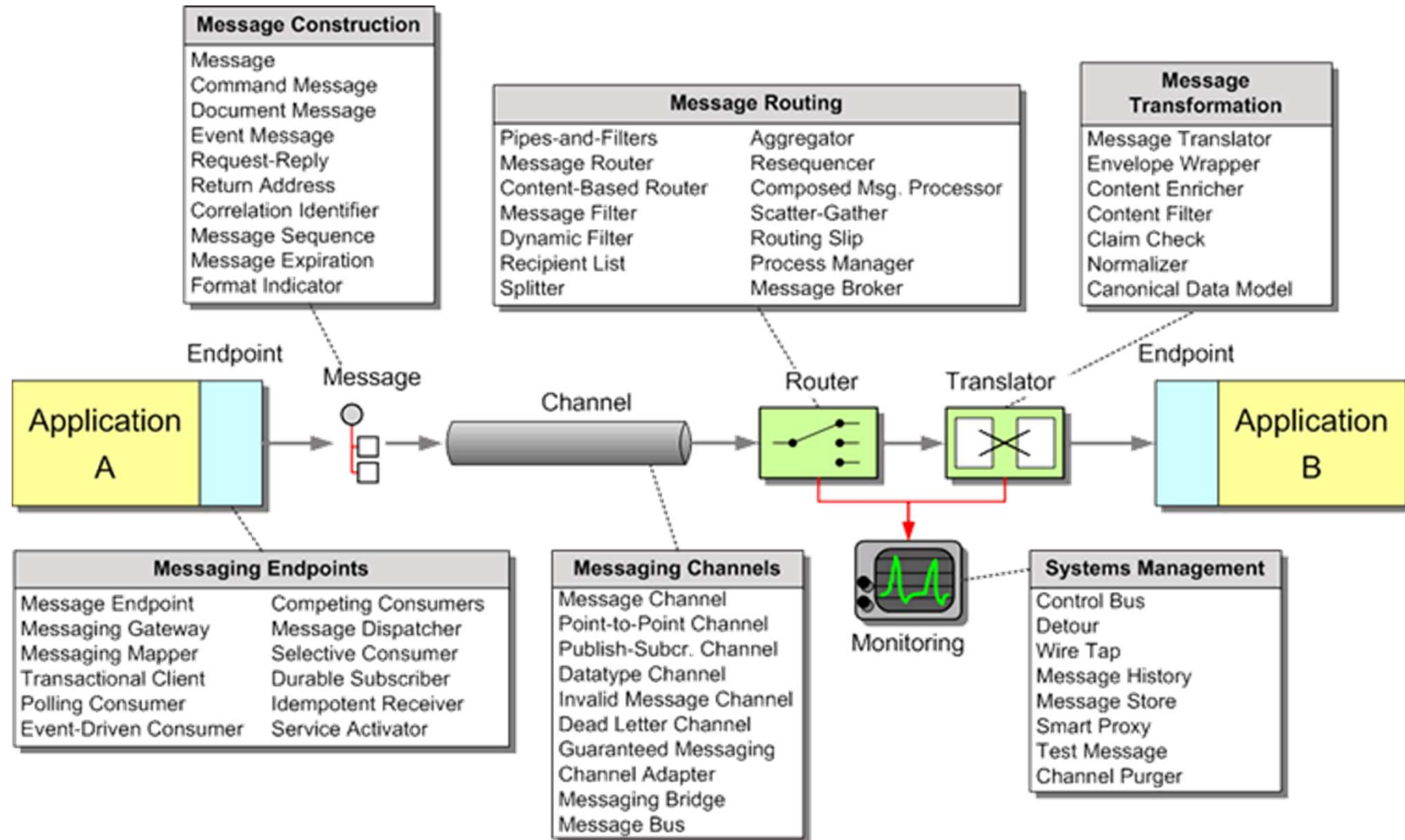
Service Oriented Architecture



An ESB works only when the scope is small and the integration logic is simple



Enterprise Integration Patterns



Connecting the parts of knowledge with the wholeness of knowledge

1. By externalizing integration logic from the application into an ESB, the applications become more loosely coupled.
2. Integration logic can be designed with a basic set of integration patterns.



3. **Transcendental consciousness** is the field that connects everything together.
4. **Wholeness moving within itself:** In Unity Consciousness, one realizes that everything else in creation is connected at the field of pure consciousness

