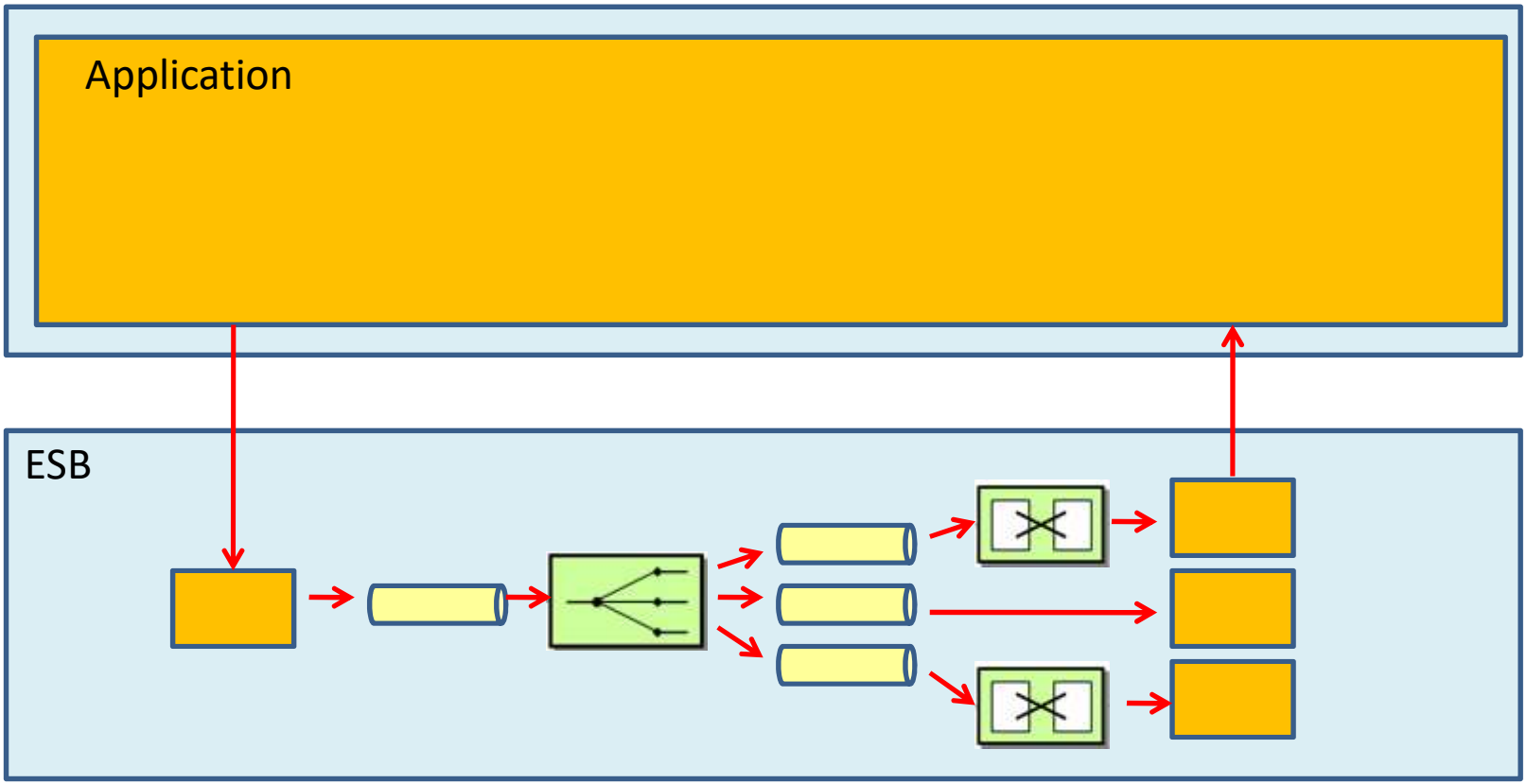
SPRING INTEGRATION

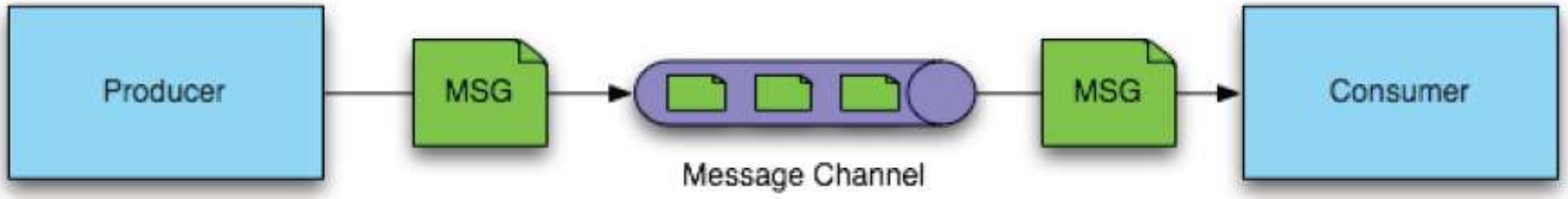
ESB

- Runs outside the application
  - Needs to be installed, started, stopped, monitored.

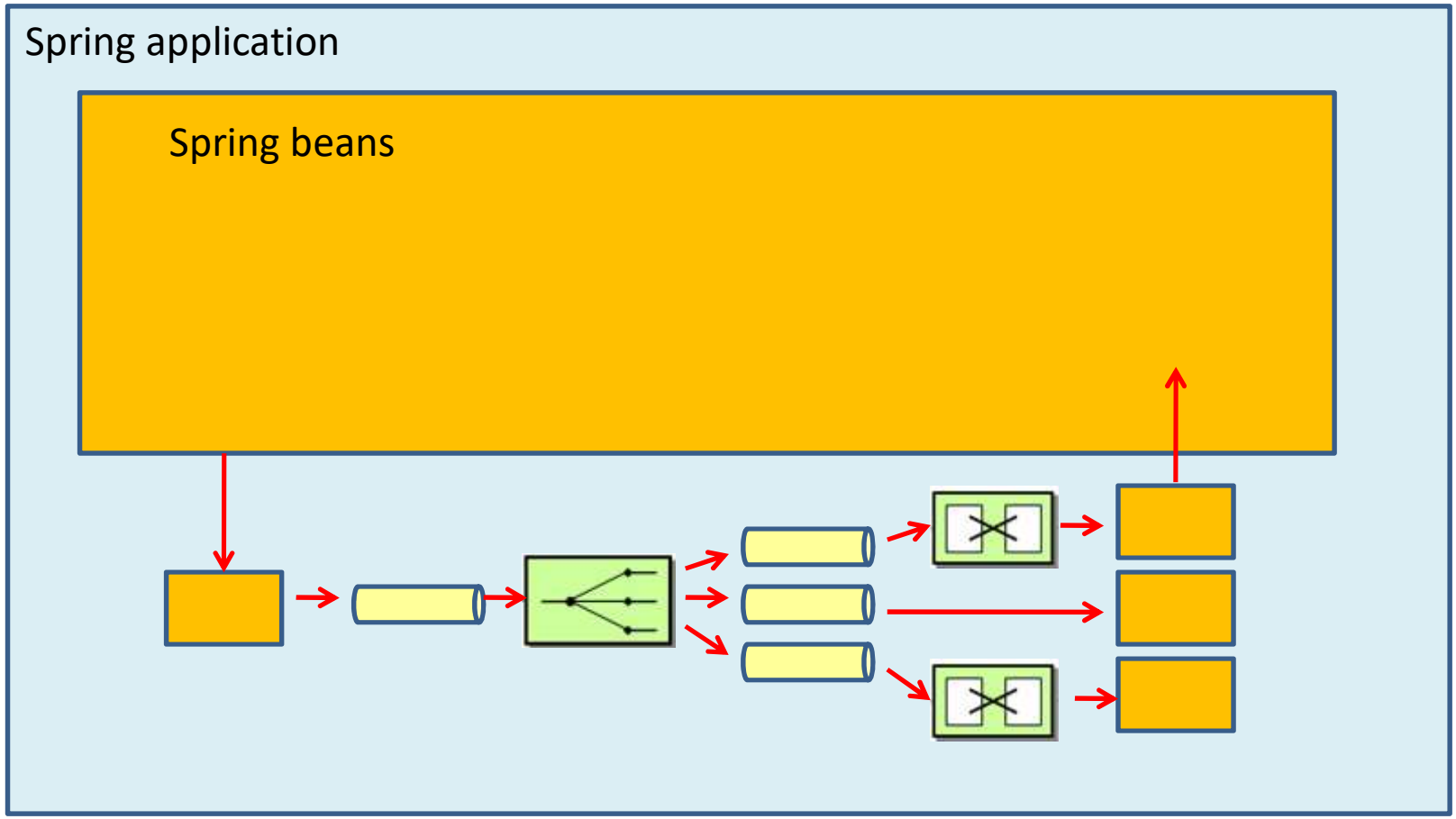


What is Spring Integration?

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

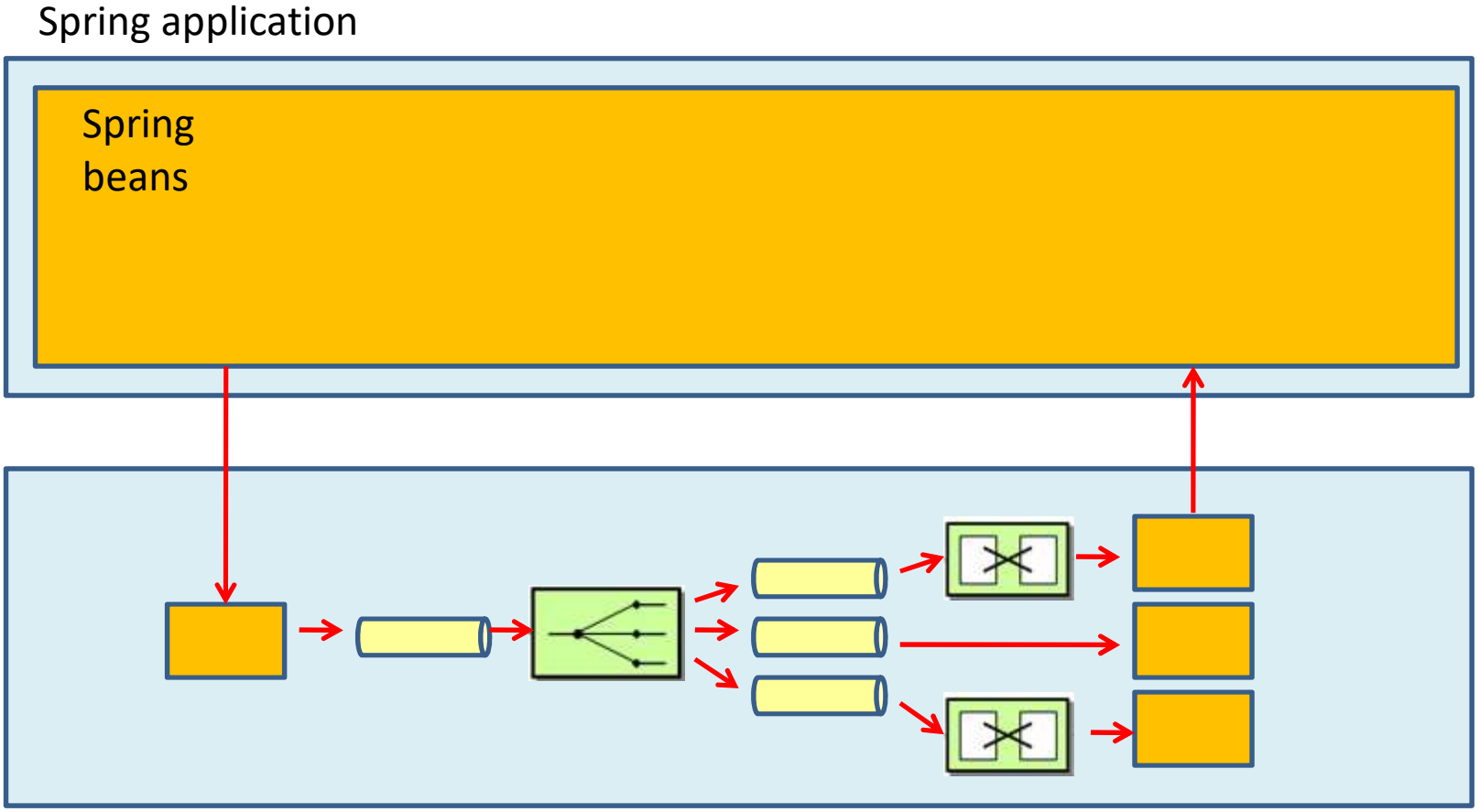


Using Spring Integration



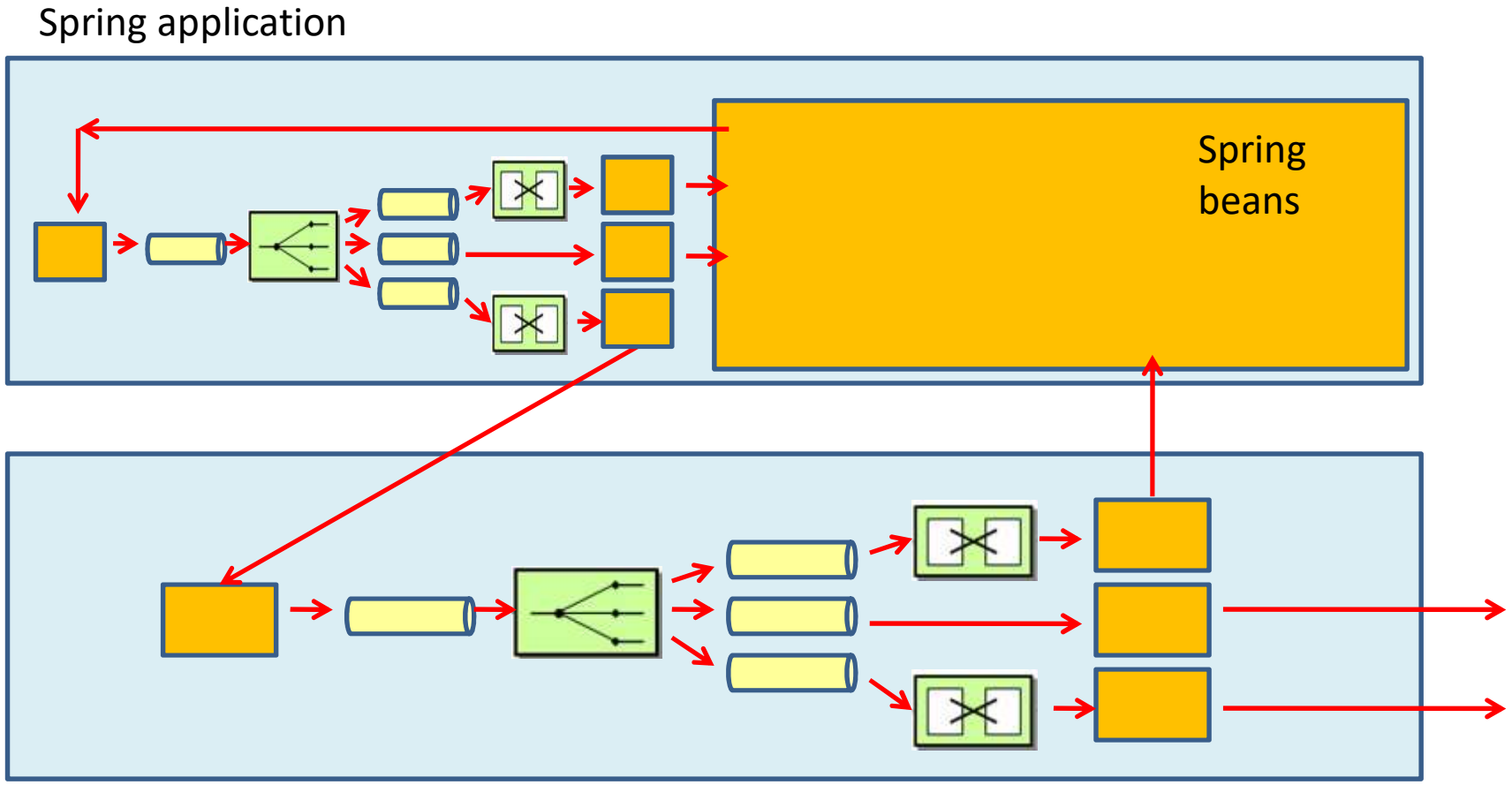
- Use SI inside your application

Using Spring Integration



- Use SI outside your application

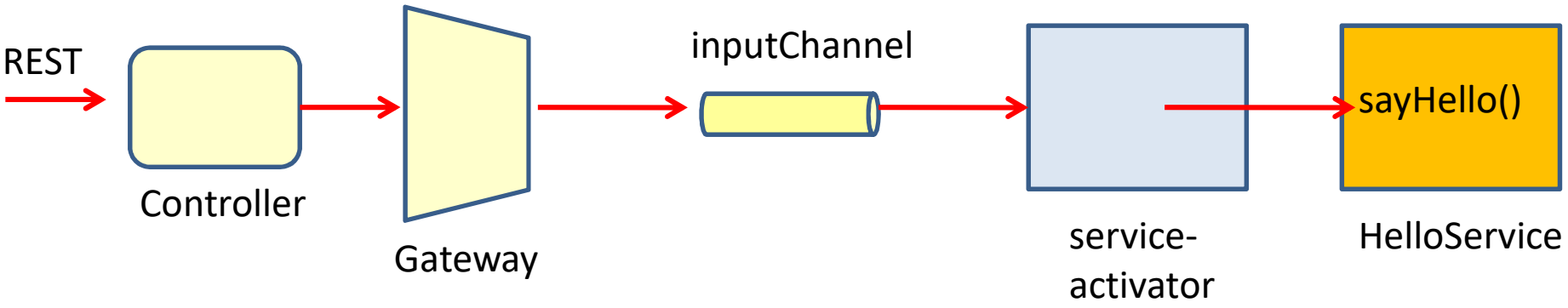
Using Spring Integration



- Use SI inside and outside your application

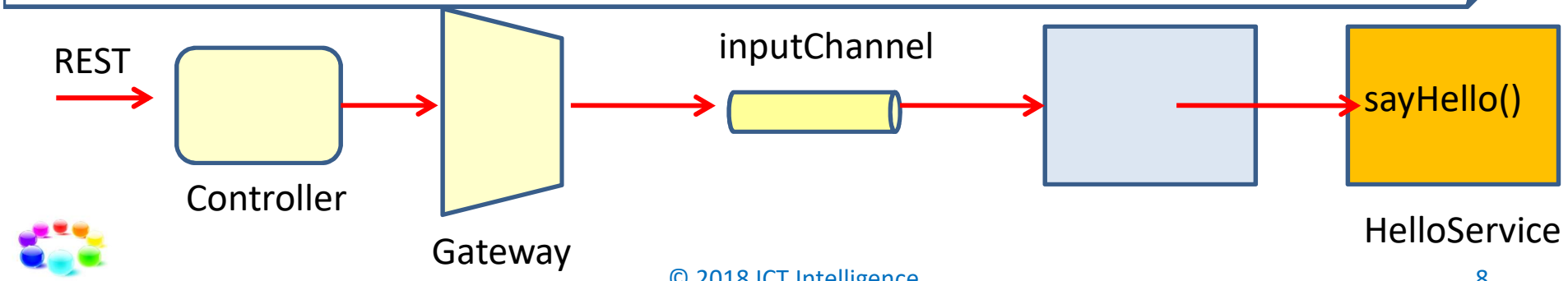
Spring integration Hello World

```
public class HelloService {

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



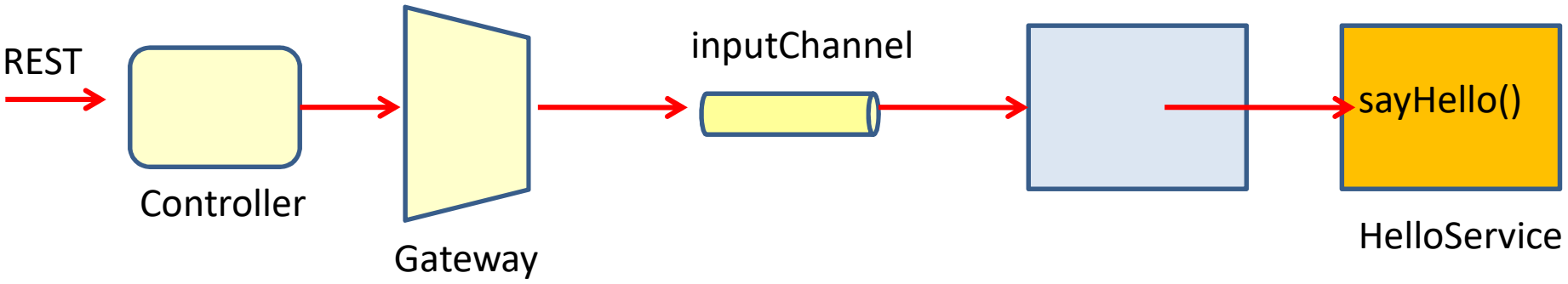
springconfiguration.xml

```
<?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="inputChannel"/>

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

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

</beans:beans>
```



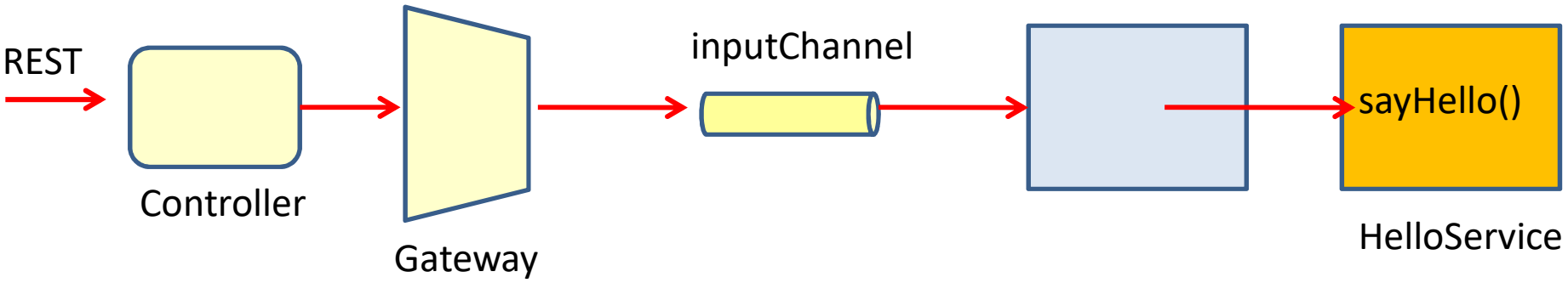
The gateway

```
@MessagingGateway
public interface GreetingGateway {

  @Gateway(requestChannel = "inputChannel")
  String handleRequest(String name);
}
```

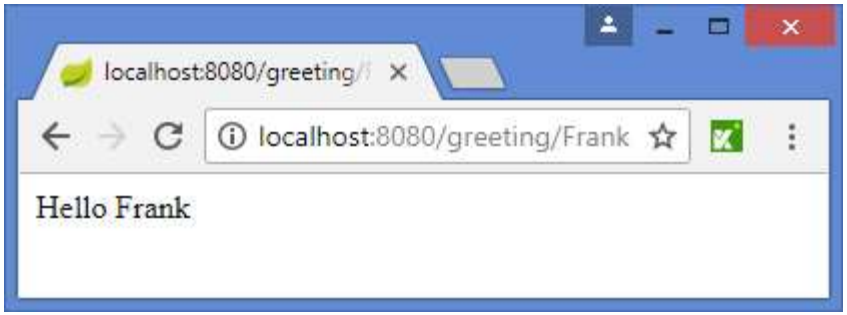


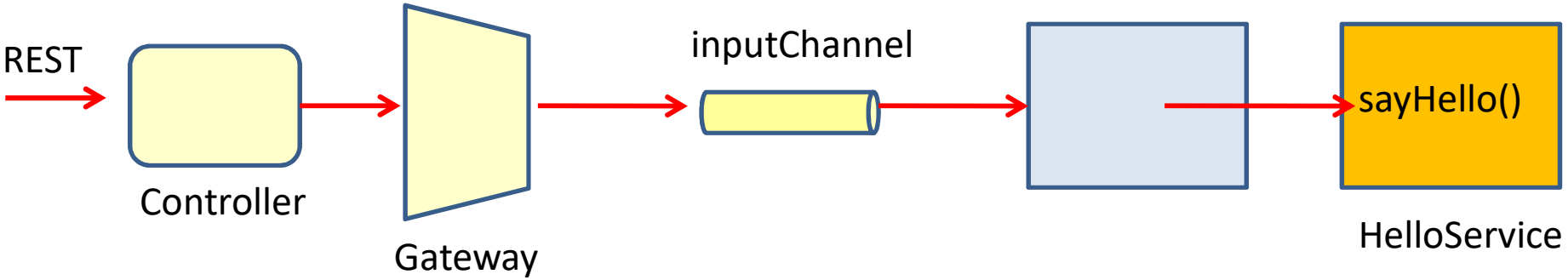
The controller

```
@RestController
public class Controller {
  @Autowired
  private GreetingGateway gateway;

  @RequestMapping("/greeting/{name}")
  public String getGreeting(@PathVariable("name") String name) {
    String result = gateway.handleRequest(name);
    return result;
  }
}
```



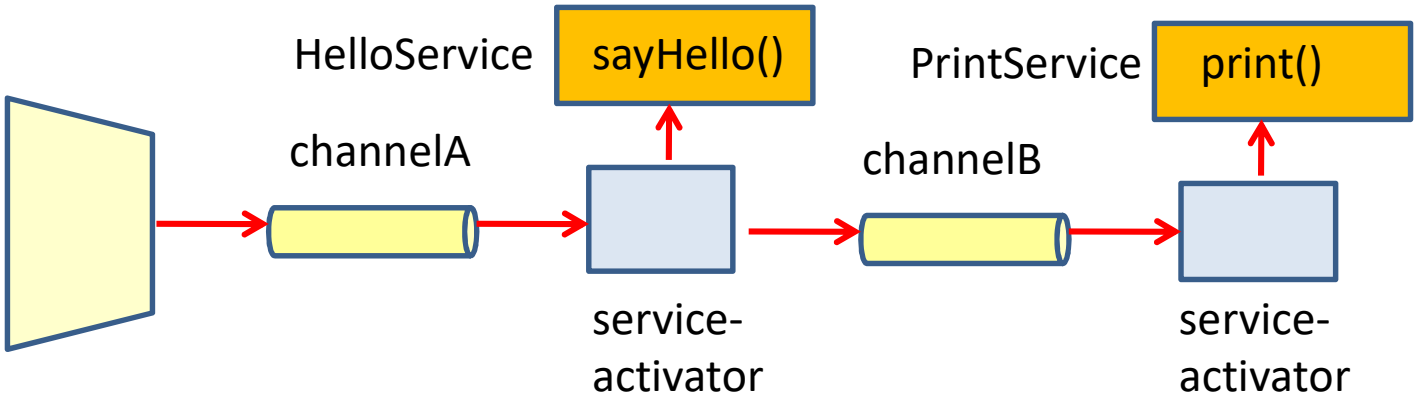
The output





Extending the application

```
<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"/>
```



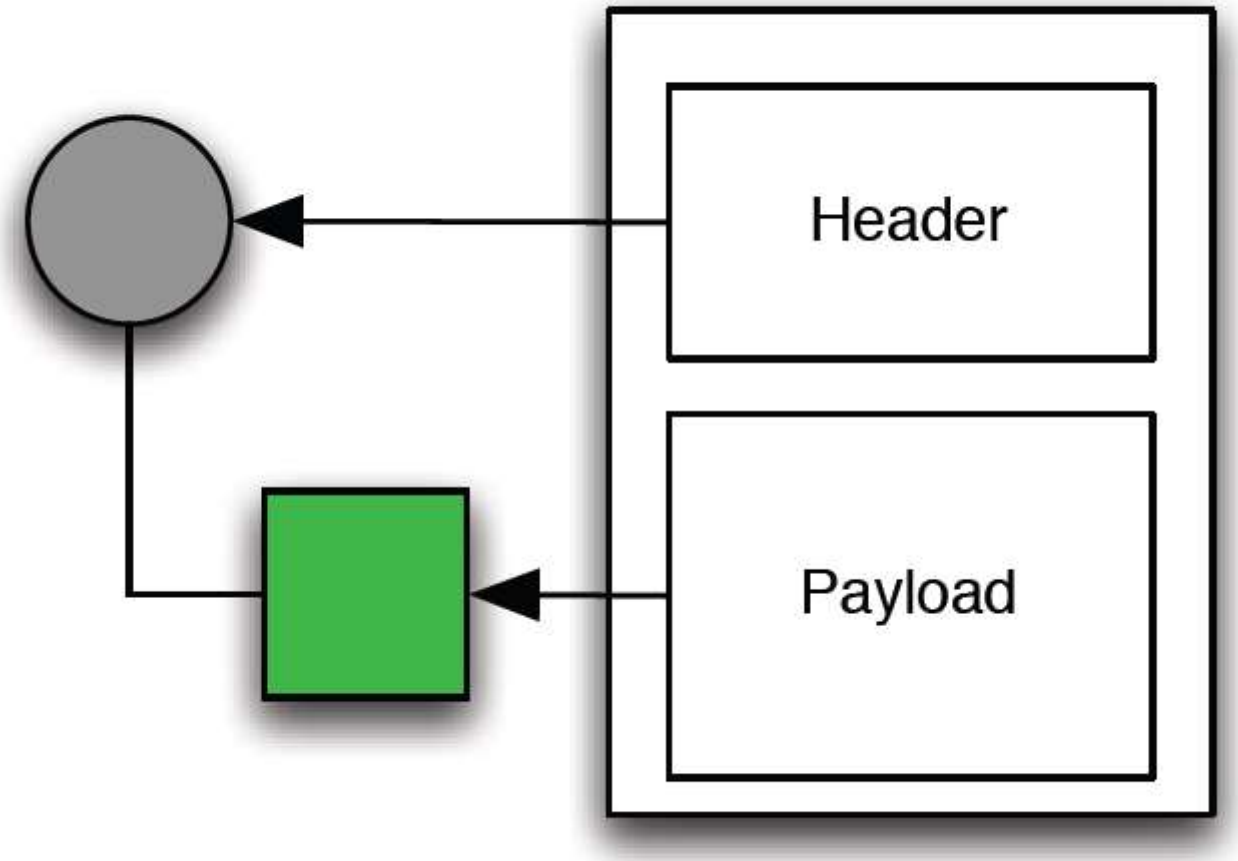
Extending the application

```
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);
  }
}
```

MESSAGES

Message



The Message interface

```
public interface Message<T> {

    T getPayload();

    MessageHeaders getHeaders();

}
```

Messages are immutable
There are no setter methods

```
public final class MessageHeaders implements Map<String, Object>, Serializable
{
   ...
}
```

MessageHeaders is a Map of Java objects

Creating a Message

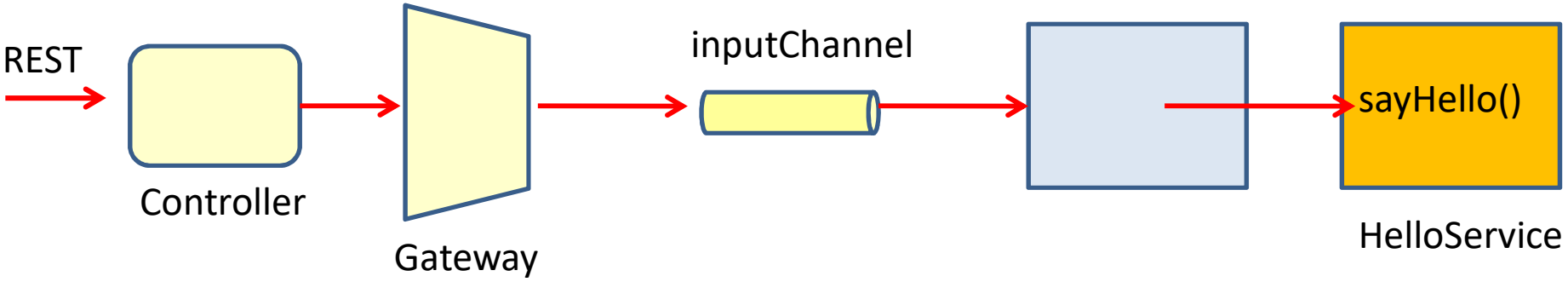
```
Message<String> helloMessage =
        MessageBuilder.withPayload("Hello, world!").build();
```

```
Message<String> helloMessage =
        MessageBuilder.withPayload("Hello, world!")
        .setHeader("my.custom.header", "HeaderValue")
        .build();
```

The gateway

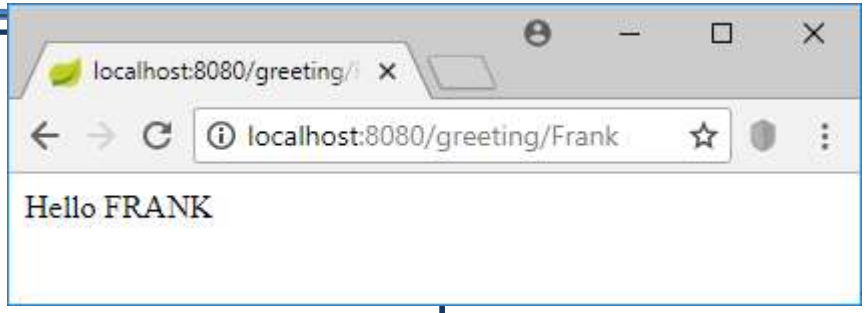
```java
@MessagingGateway
public interface GreetingGateway {

  @Gateway(requestChannel = "inputChannel")
  String handleRequest(Message<String> message);
}
```

Message instead of a String

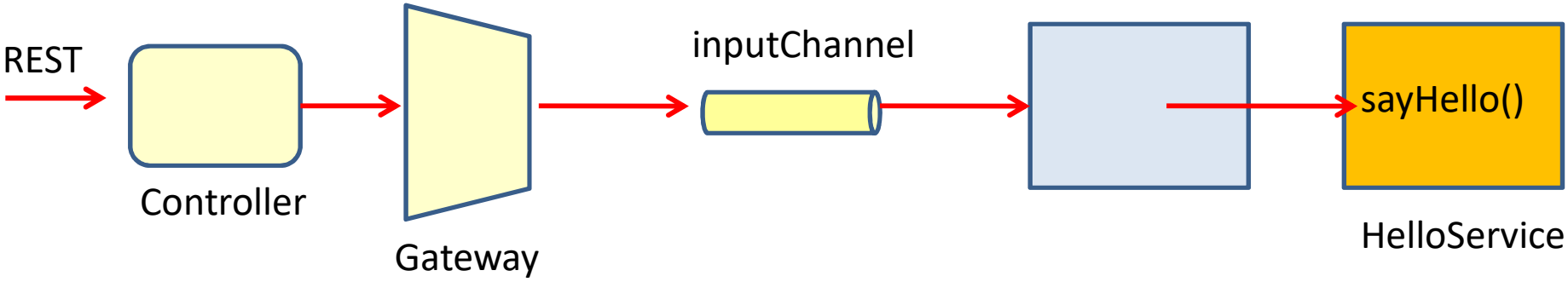


The controller

```
@RestController
public class Controller {
  @Autowired
  private GreetingGateway gateway;

  @RequestMapping("/greeting/{name}")
  public String getGreeting(@PathVariable("name") String name) {
    Message<String> helloMessage =
        MessageBuilder.withPayload(name.toUpperCase()).build();

    String result = gateway.handleRequest(helloMessage);
    return result;
  }
}
```



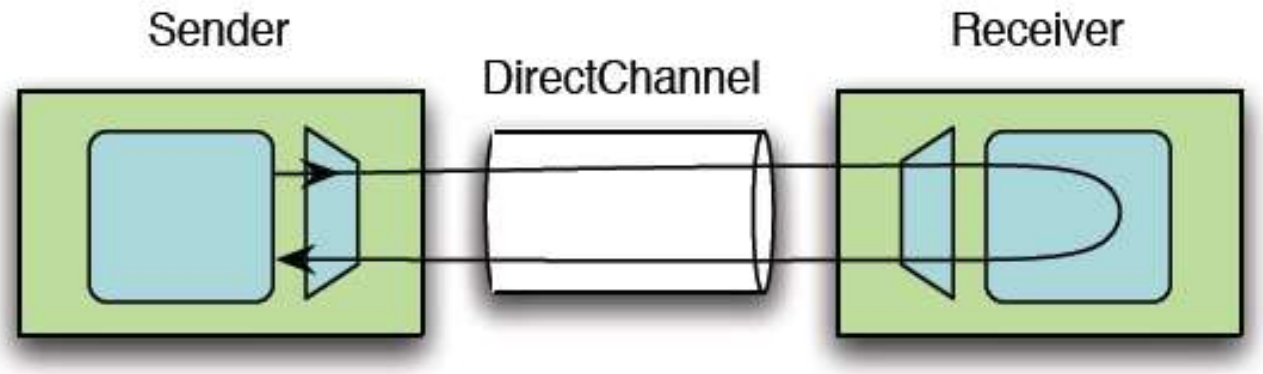
Message instead of a String



MESSAGE CHANNELS

Synchronous

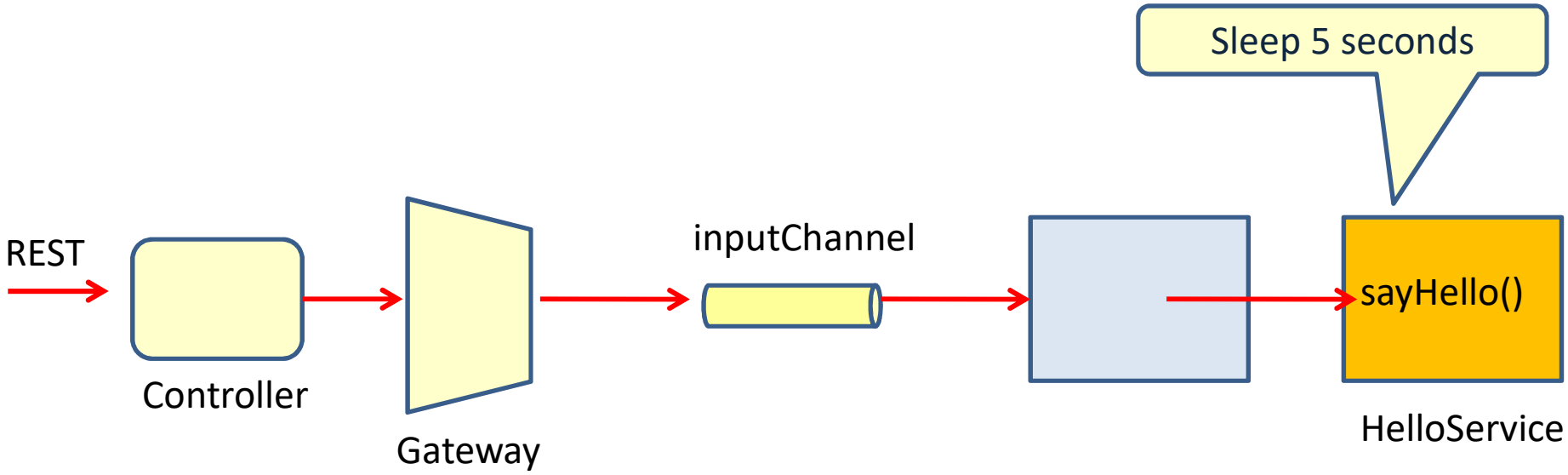
- A direct default channel is synchronous



Synchronous

```
public class HelloService {
  public String sayHello(String name) throws Exception {
    System.out.println("Hello " + name);
    Thread.sleep(5000);
    return "Hello " + name;
  }
}
```

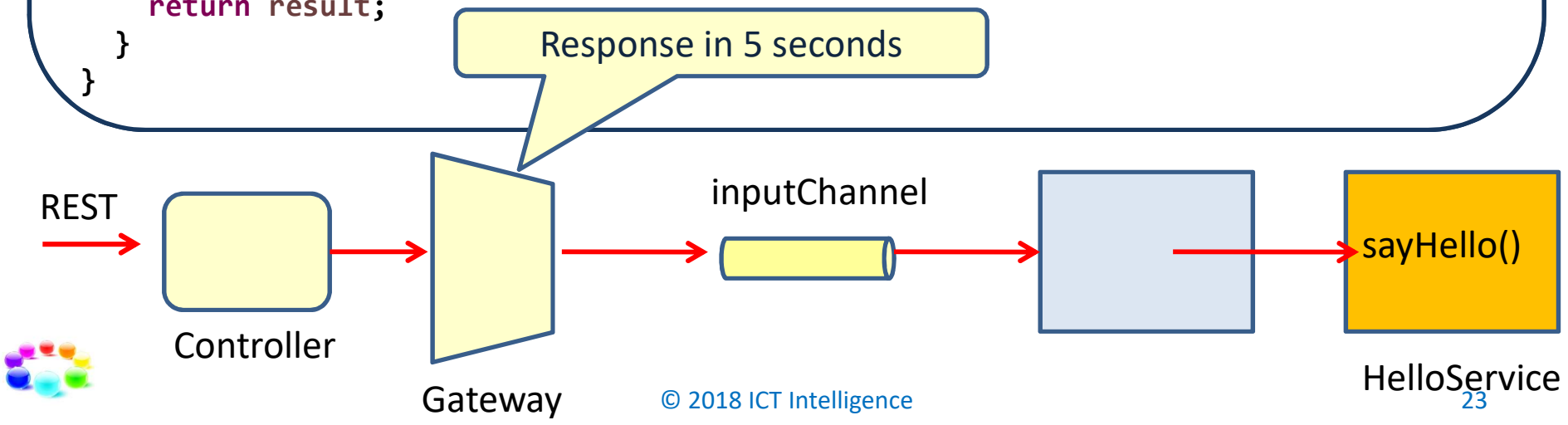
Sleep 5 seconds



Synchronous

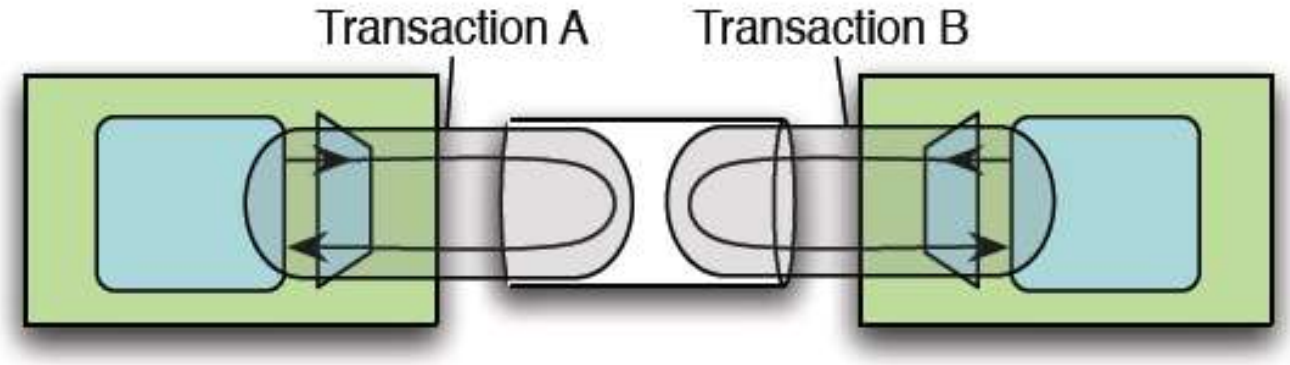
```
@RestController
public class Controller {
  @Autowired
  private GreetingGateway gateway;

  @RequestMapping("/greeting/{name}")
  public String getGreeting(@PathVariable("name") String name) {
    LocalTime localTime = LocalTime.now();

    System.out.println("time before sending message ="+ localTime);

    String result = gateway.handleRequest(name);
    localTime = LocalTime.now();
    System.out.println("time after sending message ="+ localTime);
    return result;
  }
}
```

```
time before sending message =12:03:33.027
Hello Frank
time after sending message =12:03:38.062
```



QueueChannel: Asynchronous

- A queue channel is asynchronous



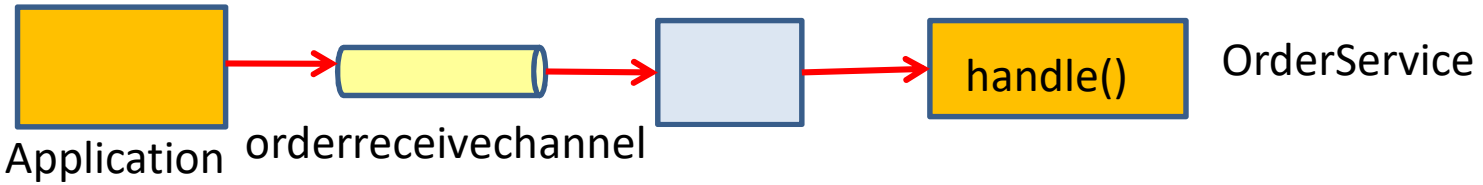
QueueChannel

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

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

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

Add a queue

Now we need a poller

```
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 needs 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

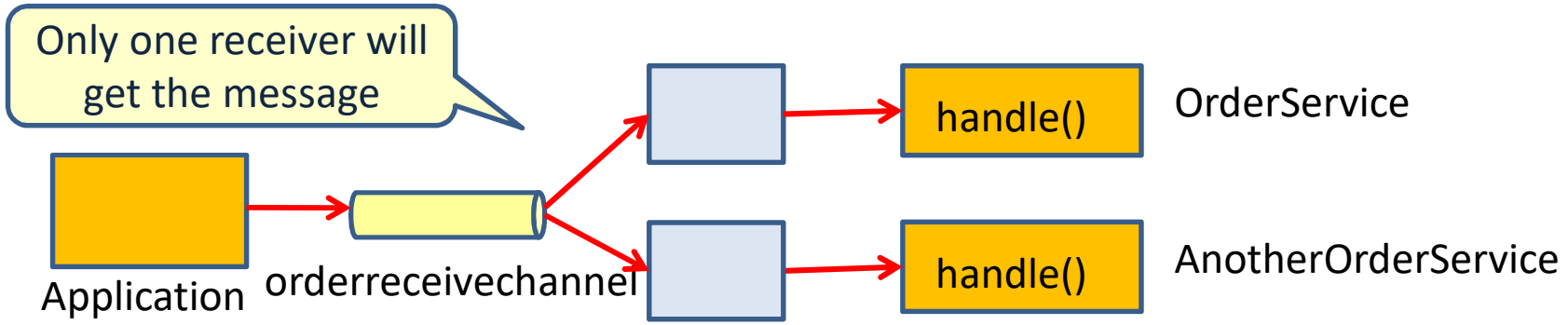
```
<channel id="stringOrNumberChannel"
         datatype="java.lang.String,java.lang.Number"/>
```

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
```

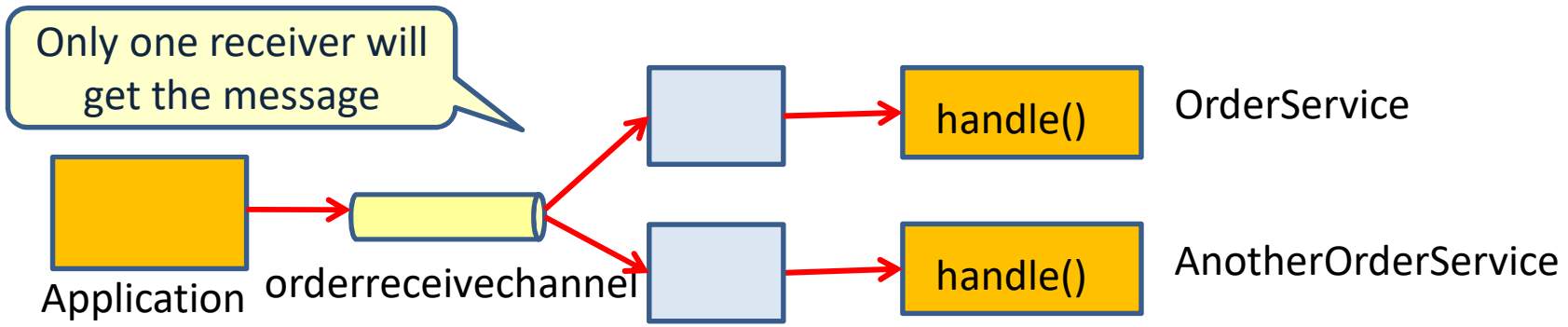


Point-to-point channel

```
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());
  }
}
```

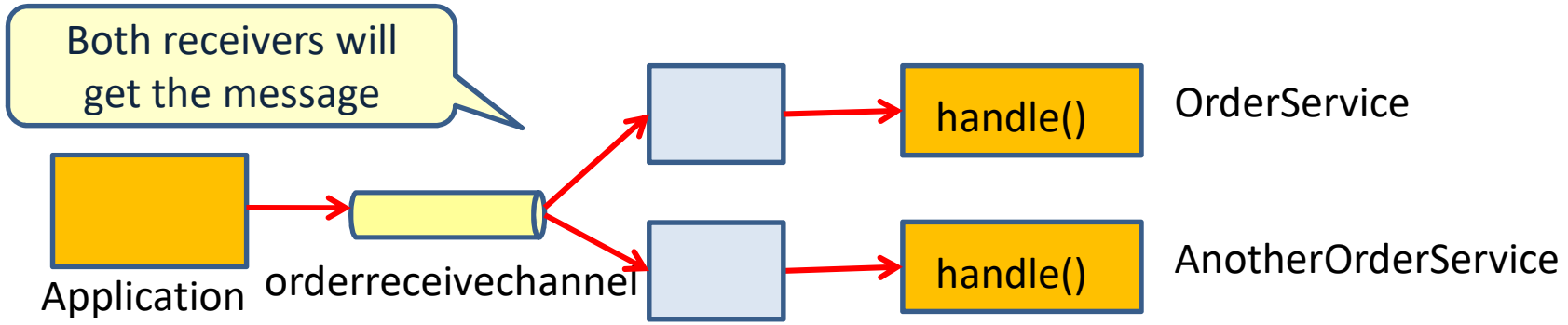
```
OrderService receiving order: order: nr=H-234-X56 amount=1245.75
```



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
```



Synchronous pub-sub

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

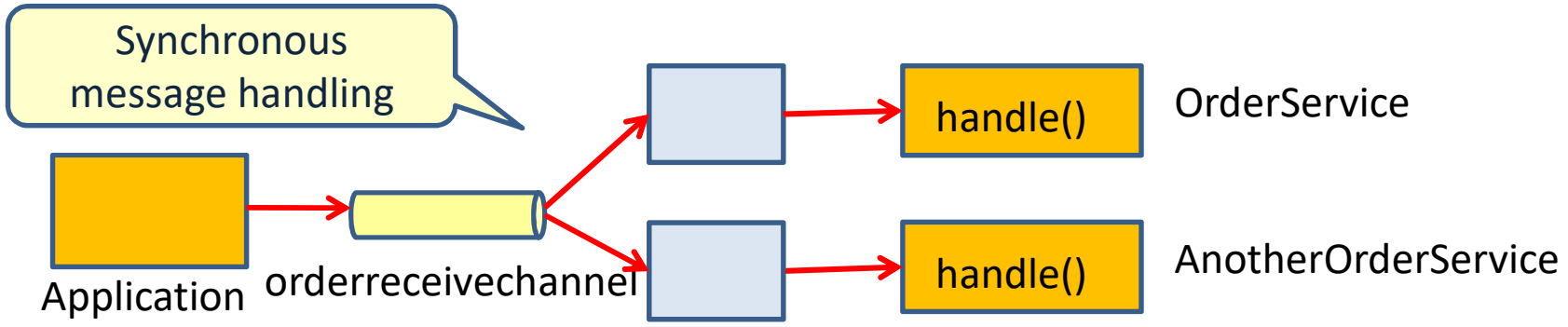
```
public class AnotherOrderService {
  public void handle(Order order) throws Exception {
    System.out.println("AnotherOrderService receiving order: "+ order.toString());
    Thread.sleep(5000);
  }
}
```

```
public class Application {

  public static void main(String[] args) {
    ...
    System.out.println("time before sending message ="
+DateFormat.getTimeInstance(DateFormat.DEFAULT).format(Calendar.getInstance().ge
tTime()));
    inputChannel.send(orderMessage);
    System.out.println("time after sending message ="
+DateFormat.getTimeInstance(DateFormat.DEFAULT).format(Calendar.getInstance().ge
tTime()));
  }
}
```

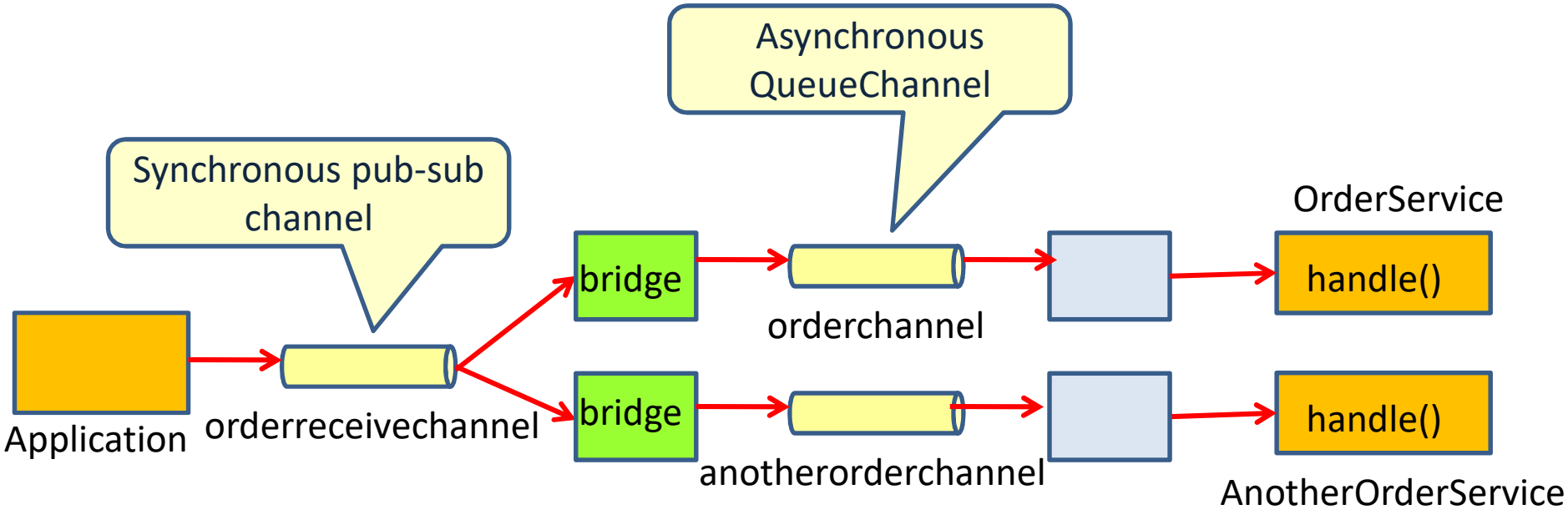
Synchronous pub-sub

```
<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" />
```

```
time before sending message =9:40:31
OrderService receiving order: order: nr=H-234-X56 amount=1245.75
AnotherOrderService receiving order: order: nr=H-234-X56 amount=1245.75
time after sending message =9:40:41
```



Asynchronous pub-sub



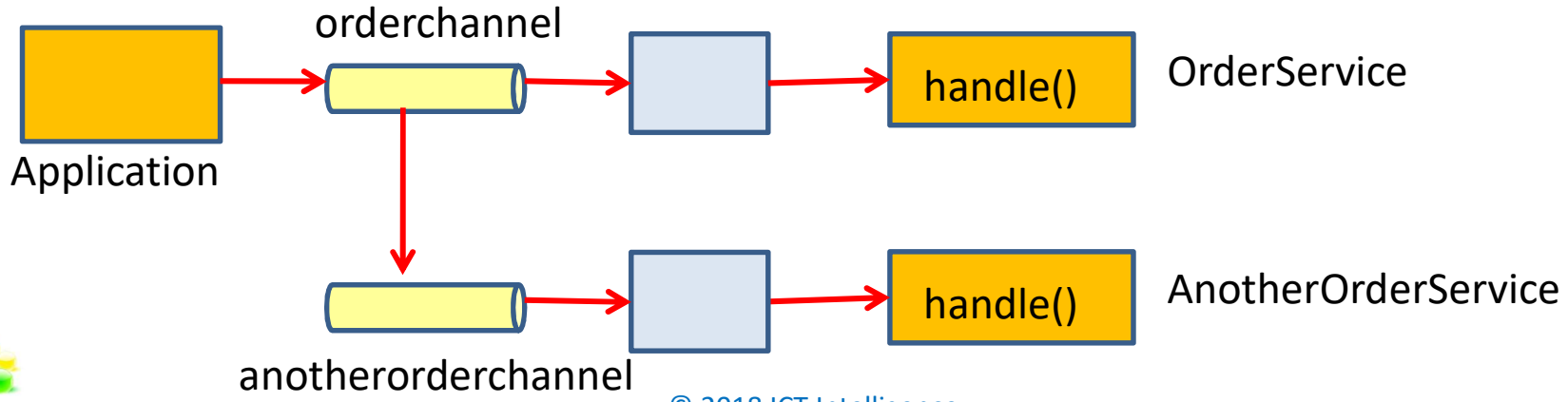

```
time before sending message =9:54:32
time after sending message =9:54:32
OrderService receiving order: order: nr=H-234-X56 amount=1245.75
AnotherOrderService receiving order: order: nr=H-234-X56 amount=1245.75
```

Asynchronous pub-sub

```
<channel id="orderchannel">
  <queue capacity="25" />
</channel>
<channel id="anotherorderchannel">
  <queue capacity="25" />
</channel>
<publish-subscribe-channel id="orderreceivechannel" />

<bridge input-channel="orderreceivechannel" output-channel="orderchannel" />
<bridge input-channel="orderreceivechannel" output-channel="anotherorderchannel" />

<service-activator input-channel="orderchannel" ref="orderservice"
                   method="handle">
  <poller>
    <interval-trigger interval="200" />
  </poller>
</service-activator>
<service-activator input-channel="anotherorderchannel"
                    ref="anotherorderservice" method="handle">
  <poller>
    <interval-trigger interval="200" />
  </poller>
</service-activator>
<beans:bean id="orderservice" class="integration.OrderService" />
<beans:bean id="anotherorderservice" class="integration.AnotherOrderService" />
```

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" />
```

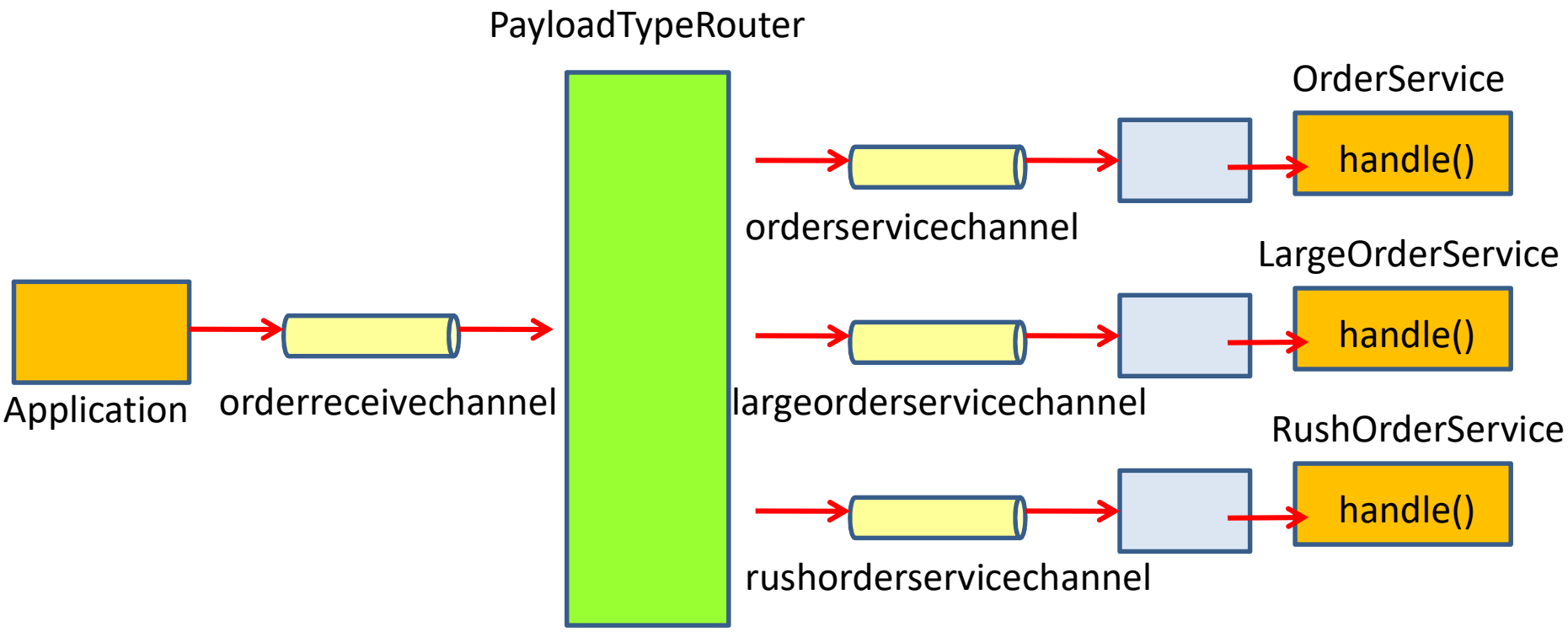


ROUTER

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());
  }
}
```

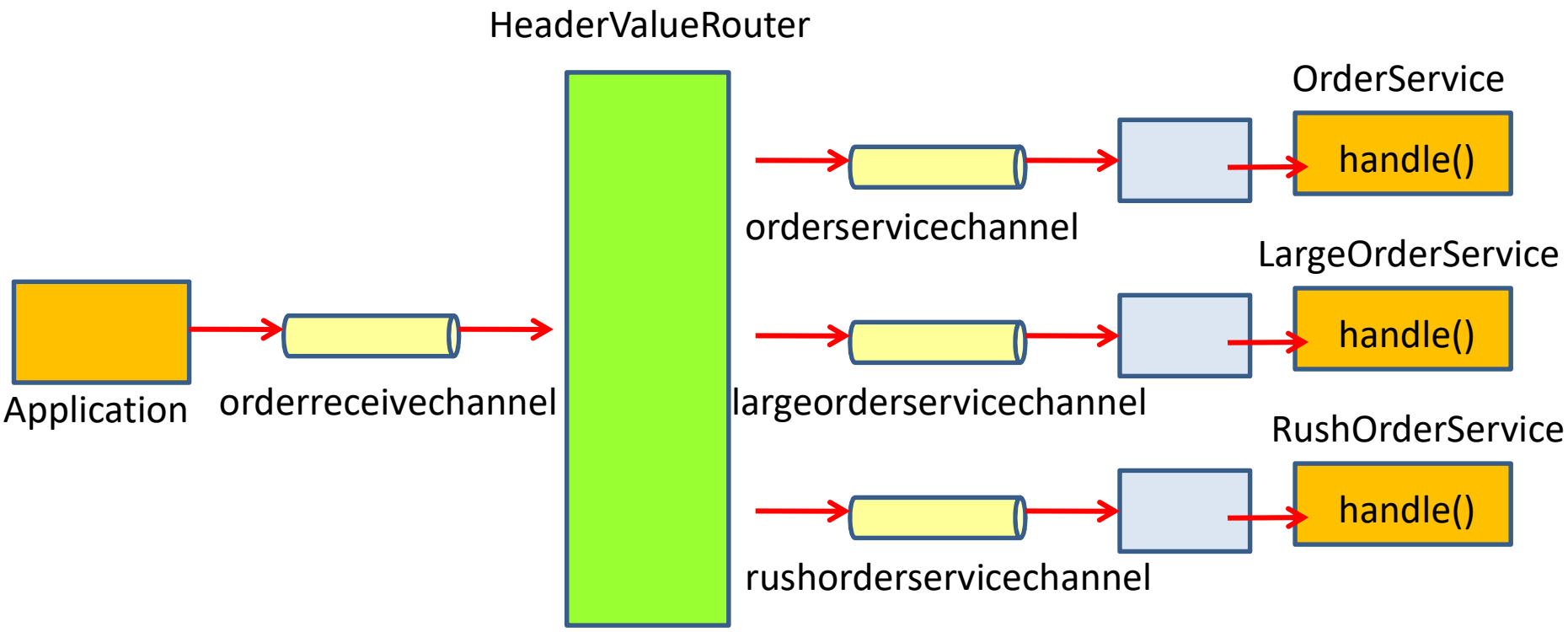
```
public class RushOrderService {
  public void handle(Order order) {
    System.out.println("RushOrderService receiving order: "+ order.toString());
  }
}
```

The application

```
    Order rushOrder = new RushOrder("H-234-X56",600.65);
    Order largeOrder = new LargeOrder("H-234-X56",30045.35);

    Message<Order> orderMessage = MessageBuilder.withPayload(order).build();
    Message<Order> rushOrderMessage =
                   MessageBuilder.withPayload(rushOrder).build();
    Message<Order> largeOrderMessage =
                   MessageBuilder.withPayload(largeOrder).build();

    gateway.handleRequest(orderMessage);
    gateway.handleRequest(rushOrderMessage);
    gateway.handleRequest(largeOrderMessage);
  }
}
```

```
OrderService receiving order: order: nr=H-234-X56 amount=1245.75
RushOrderService receiving order: order: nr=H-234-X56 amount=600.65
LargeOrderService receiving order: order: nr=H-234-X56 amount=30045.35
```

HeaderValueRouter



HeaderValueRouter

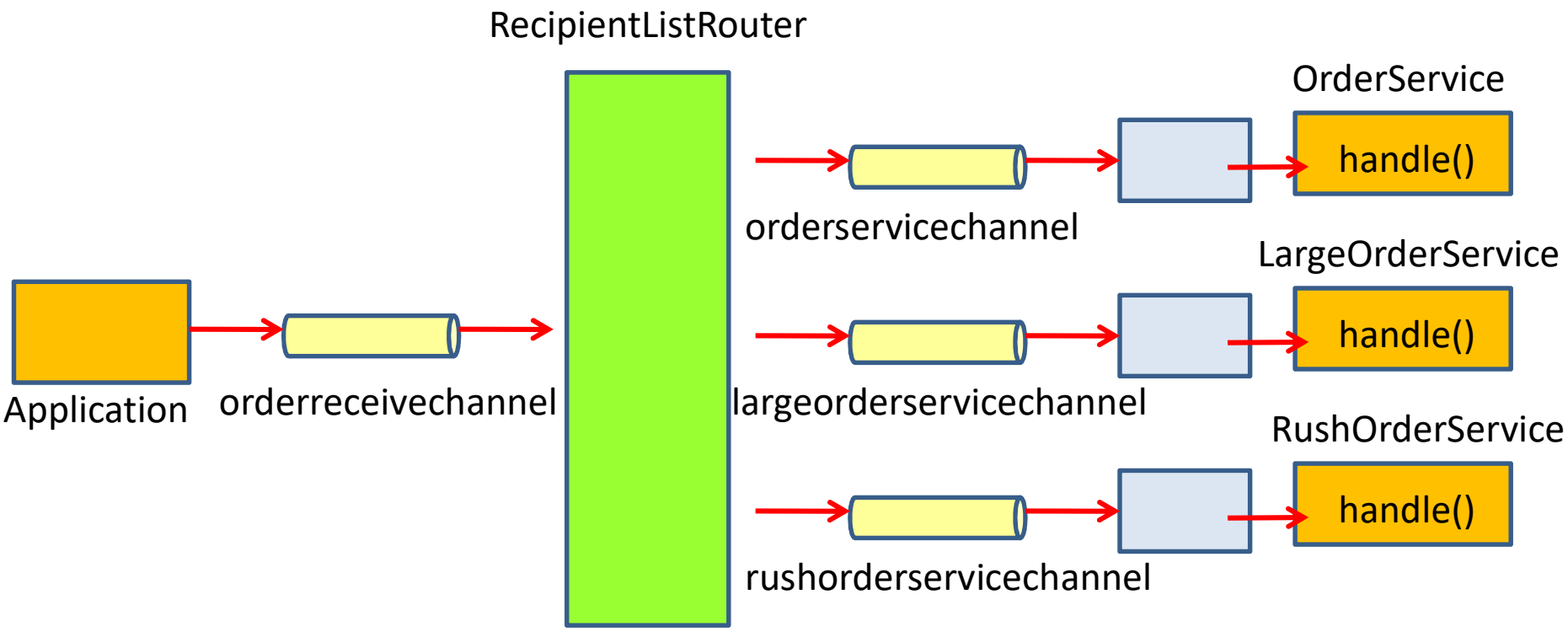
```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />
<channel id="rushorderservicechannel" />
<channel id="largeorderservicechannel" />

<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" />
```

The application

```
Order order = new Order("H-234-X56",1245.75);
Order rushOrder = new RushOrder("H-234-X57",600.65);
Order largeOrder = new LargeOrder("H-234-X58",30045.35);

Message<Order> orderMessage = MessageBuilder.withPayload(order)
                .setHeader("orderType", "normal").build();
Message<Order> rushOrderMessage = MessageBuilder.withPayload(rushOrder)
                .setHeader("orderType", "rush").build();
Message<Order> largeOrderMessage = MessageBuilder.withPayload(largeOrder)
                .setHeader("orderType", "large").build();

gateway.handleRequest(orderMessage);
gateway.handleRequest(rushOrderMessage);
gateway.handleRequest(largeOrderMessage);
```

```
OrderService receiving order: order: nr=H-234-X56 amount=1245.75
RushOrderService receiving order: order: nr=H-234-X57 amount=600.65
LargeOrderService receiving order: order: nr=H-234-X58 amount=30045.35
```

RecipientListRouter



RecipientListRouter

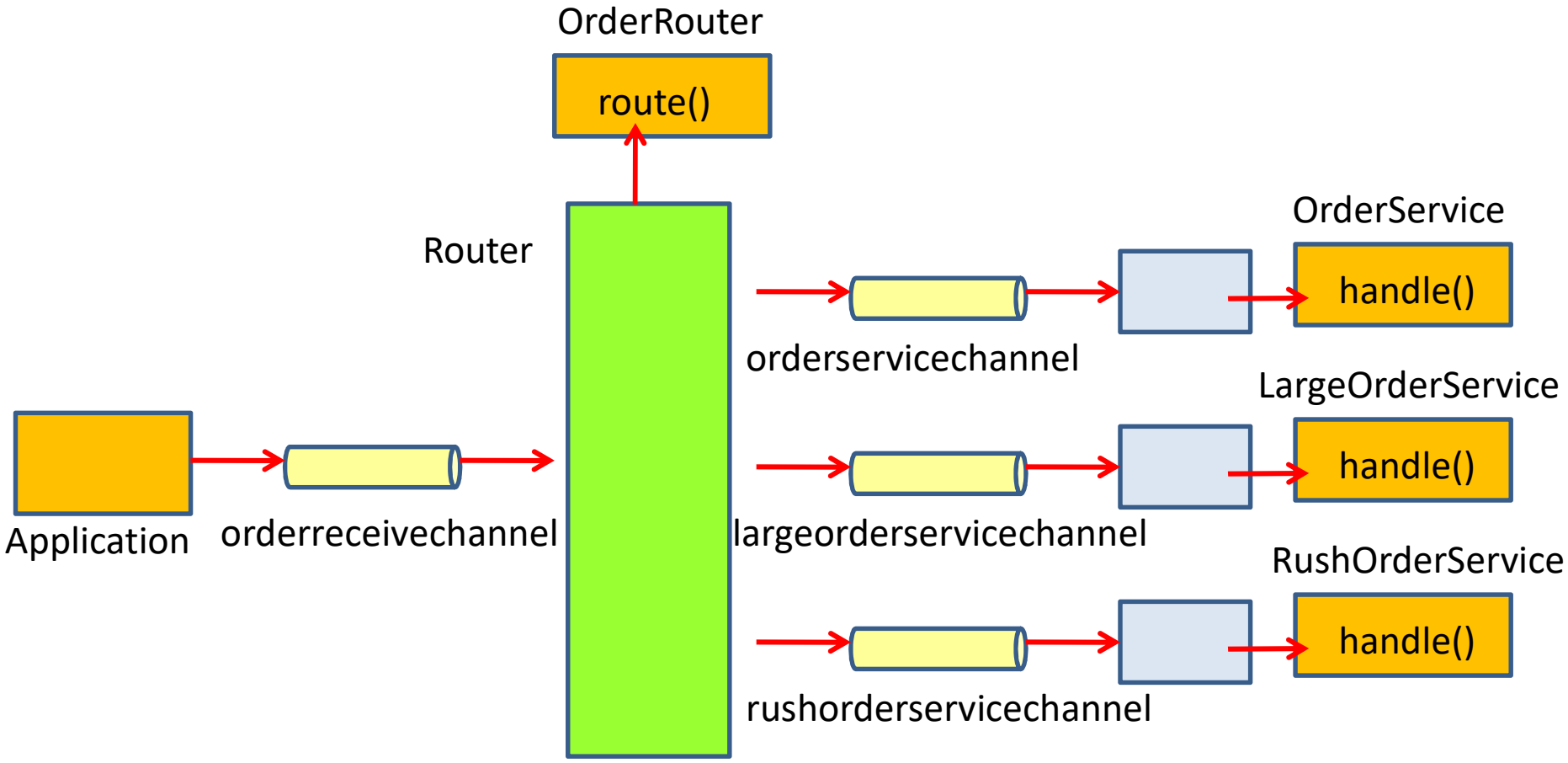
```
<channel id="orderreceivechannel" />
<channel id="orderservicechannel" />
<channel id="rushorderservicechannel" />
<channel id="largeorderservicechannel" />

<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" />
```

The application

```
Order order = new Order("H-234-X56",1245.75);
Order order2 = new Order("H-234-X57",600.65);

Message<Order> orderMessage = MessageBuilder.withPayload(order).build();
Message<Order> orderMessage2 = MessageBuilder.withPayload(order2).build();

gateway.handleRequest(orderMessage);
gateway.handleRequest (orderMessage2);
```

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

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;
  }
}
```

The application

```
Order order = new Order("H-234-X56",1245.75, true);
Order order2 = new Order("H-234-X57",600.65, false);
Order order3 = new Order("H-234-X58",50600.65, false);

Message<Order> orderMessage = MessageBuilder.withPayload(order).build();
Message<Order> orderMessage2 = MessageBuilder.withPayload(order2).build();
Message<Order> orderMessage3 = MessageBuilder.withPayload(order3).build();

gateway.handleRequest(orderMessage);
gateway.handleRequest(orderMessage2);
gateway.handleRequest(orderMessage3);
```

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;
  }
}
```

The application

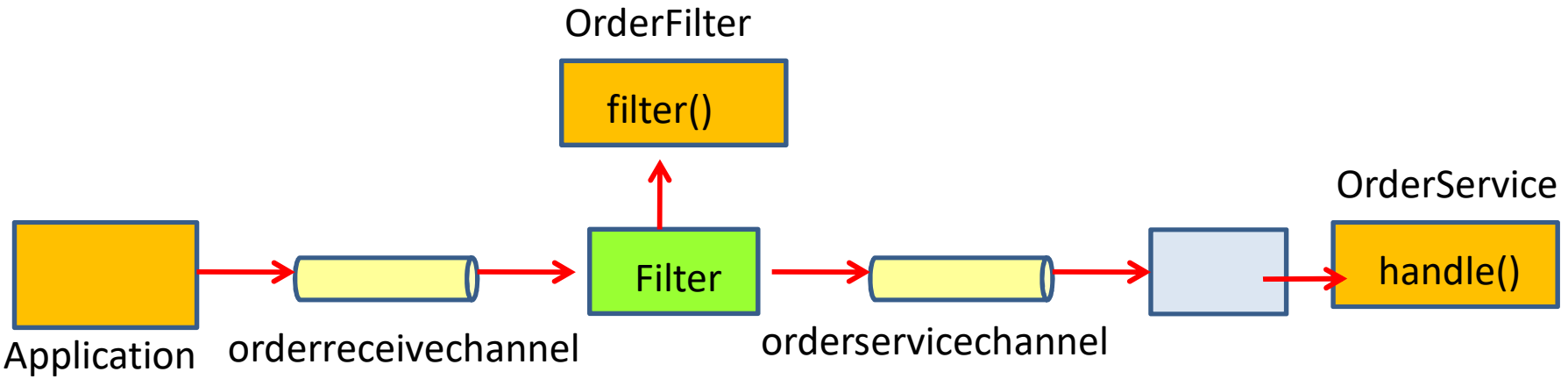
```
public class Application {
  public static void main(String[] args) {
    ApplicationContext context = new
      ClassPathXmlApplicationContext("/integration/springconfiguration.xml");
    Order order = new Order("H-234-X56",1245.75, true);
    Order order2 = new Order("H-234-X57",600.65, false);
    Order order3 = new Order("H-234-X58",50600.65, true);
    Message<Order> orderMessage = MessageBuilder.withPayload(order).build();
    Message<Order> orderMessage2 = MessageBuilder.withPayload(order2).build();
    Message<Order> orderMessage3 = MessageBuilder.withPayload(order3).build();
    MessageChannel inputChannel = context.getBean("orderreceivechannel",
                                    MessageChannel.class);
    inputChannel.send(orderMessage);
    inputChannel.send(orderMessage2);
    inputChannel.send(orderMessage3);
  }
}
```

```
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;
  }
}
```

The Order and the OrderService

```
public class Order {
  private String orderNumber;
  private double amount;

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

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

The application

```
        Order order = new Order("H-234-X56",1245.75);
        Order order2 = new Order("H-234-X57",600.65);
        Order order3 = new Order("H-234-X58",50600.65);

        Message<Order> orderMessage = MessageBuilder.withPayload(order).build();
        Message<Order> orderMessage2 = MessageBuilder.withPayload(order2).build();
        Message<Order> orderMessage3 = MessageBuilder.withPayload(order3).build();

        gateway.handleRequest(orderMessage);
        gateway.handleRequest(orderMessage2);
        gateway.handleRequest(orderMessage3);
    }
}
```

OrderService receiving order: order: nr=H-234-X56 amount=1245.75
OrderService receiving order: order: nr=H-234-X58 amount=50600.65

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

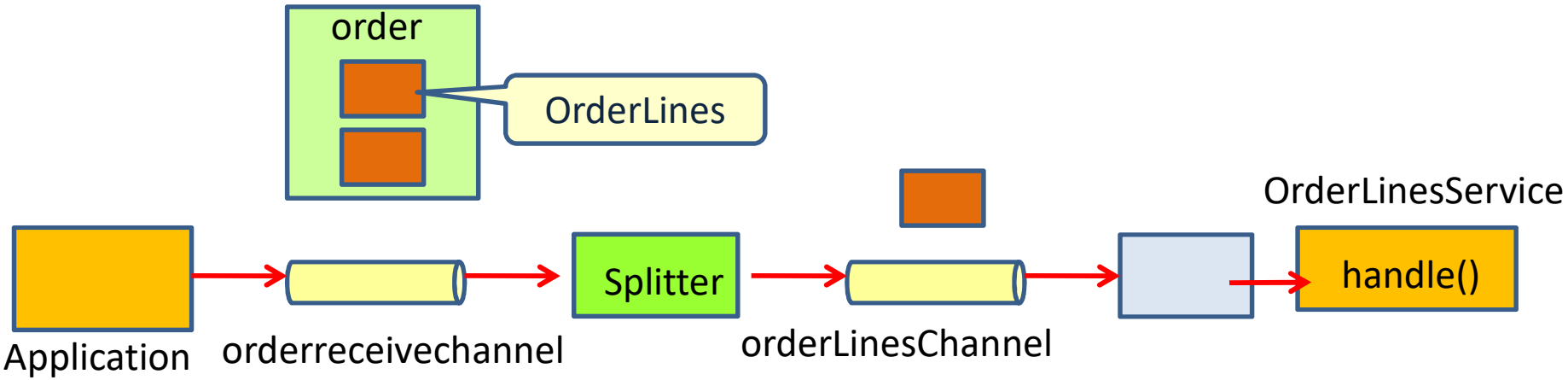
```
<filter input-channel="orderreceivechannel" output-channel="orderservicechannel"
        ref="orderfilter" method="filter" discard-channel="rejectedMessages"/>
```

Send rejected messages to another channel

SPLITTER AND AGGREGATOR

Splitter

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

<splitter input-channel="orderreceivechannel" output-channel="orderLineschannel"
          ref="splitterBean" method="split" />

<service-activator input-channel="orderLinesChannel"
                   ref="orderLinesService" method="handle" />

<beans:bean id="splitterBean" class="integration.OrderSplitter" />
<beans:bean id="orderLinesService" class="integration.OrderLinesService" />
```



The Splitter class

```
public class OrderSplitter {
  public Collection<OrderLine> split(Order order) {
    return order.getOrderLines();
  }
}
```

```
public class Order {
  private String orderNumber;
  private Collection<OrderLine> orderLines = new ArrayList<OrderLine>();
...
}
```

```
public class OrderLine {
  private int quantity;
  private Product product;
...
}
```

```
public class Product {
  private String nr;
  private String name;
  private double price;
...
}
```

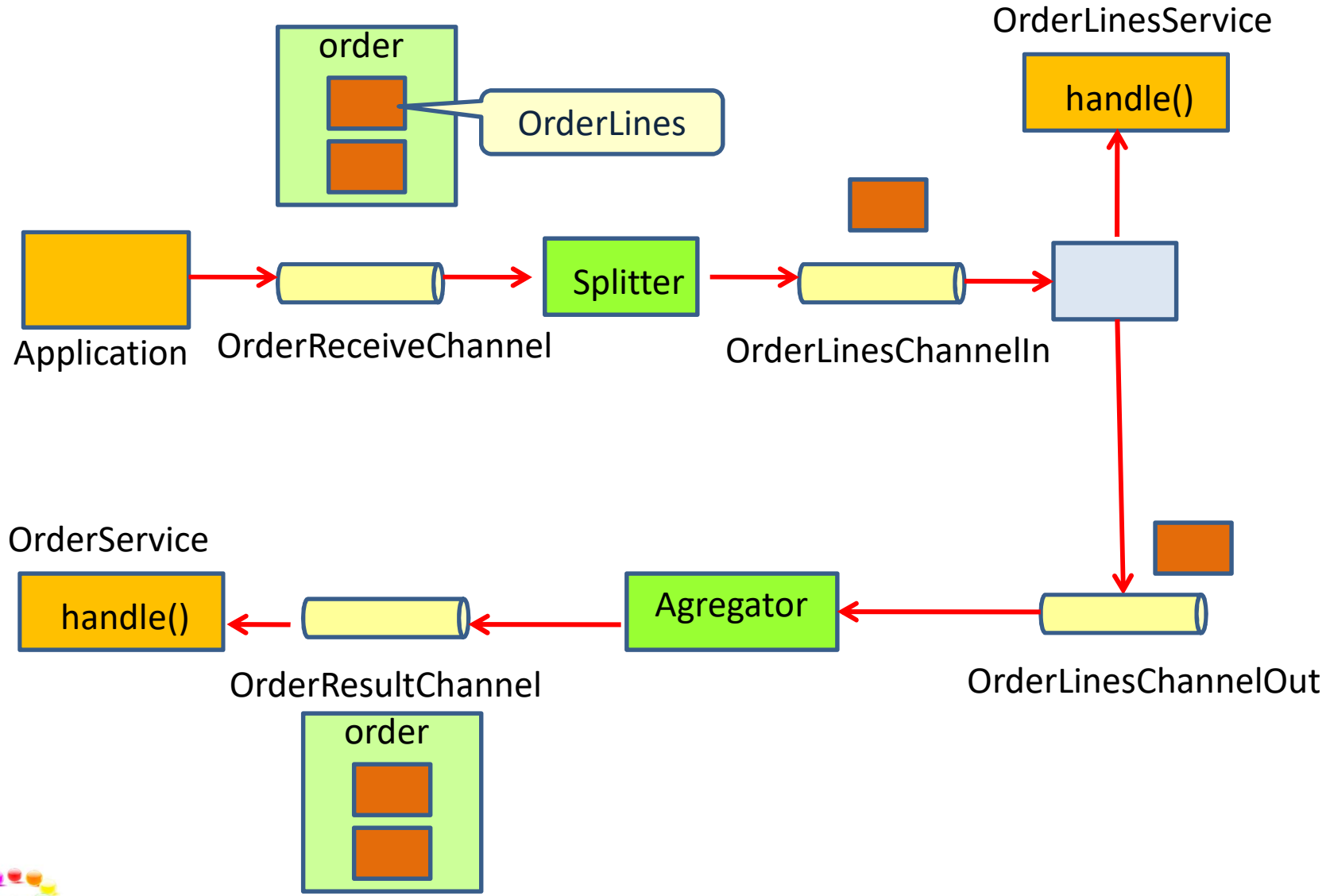
The application

```
Order order = new Order("H-234-X56");
order.addOrderLine(new OrderLine(4, new Product("XH-456","MP3
                   player",125.75)));
order.addOrderLine(new OrderLine(2, new Product("GH-428","LED 3D
                   TV",675.85)));
Message<Order> orderMessage = MessageBuilder.withPayload(order).build();

gateway.handleRequest(orderMessage);
```

```
OrderLinesService receiving orderline: quantity=4 product=MP3 player
sequenceNumber: 1
sequenceSize: 2
-----------------------------------
OrderLinesService receiving orderline: quantity=2 product=LED 3D TV
sequenceNumber: 2
sequenceSize: 2
-----------------------------------
```

Spring Integration adds sequenceNumber and sequenceSize

Aggregator



Aggregator

```
<channel id="orderReceiveChannel" />
<channel id="OrderLinesChannelIn" />
<channel id="OrderLinesChannelOut" />
<channel id="OrderResultChannel" />

<splitter input-channel="orderReceiveChannel"
          output-channel="OrderLinesChannelIn"
          ref="splitterBean" method="split" />

<service-activator input-channel="OrderLinesChannelIn"
                   output-channel="OrderLinesChannelOut"
                   ref="orderLinesService" method="handle" />

<aggregator input-channel="OrderLinesChannelOut"
            output-channel="OrderResultChannel"
            ref="aggegatorBean" method="aggregate"/>

<service-activator input-channel="OrderResultChannel"
                   ref="orderService" method="handle" />

<beans:bean id="splitterBean" class="integration.OrderSplitter" />
<beans:bean id="aggegatorBean" class="integration.OrderAggregator" />
<beans:bean id="orderLinesService" class="integration.OrderLinesService" />
<beans:bean id="orderService" class="integration.OrderService" />
```

The Splitter and Aggregator

```
public class OrderSplitter {
  public Collection<OrderLine> split(Order order) {
    return order.getOrderLines();
  }
}
```

```
public class OrderAggregator {
  public Order aggregate(Collection<OrderLine> orderlines) {
    Order order = new Order();
    for (OrderLine ol: orderlines){
      order.addOrderLine(ol);
    }
    return order;
  }
}
```

The Payload and Services

```
public class OrderLinesService {
  public OrderLine handle(OrderLine orderline) throws Exception {
    System.out.println("OrderLinesService receiving orderline: "+
           orderline.toString());
    return orderline;
  }
}
```

```
public class OrderService {
  public void handle(Order order)  {
    System.out.println("OrderService receiving order:");
    for (OrderLine ol : order.getOrderLines()){
      System.out.println(ol.getProduct().getName());
    }
  }
}
```

```
public class Order {
private Collection<OrderLine> orderLines = new ArrayList<OrderLine>();
...
}
```

```
public class OrderLine {
  private int quantity;
  private Product product;
...
}
```

```
public class Product {
  private String nr;
  private String name;
  private double price;
...
}
```

The application

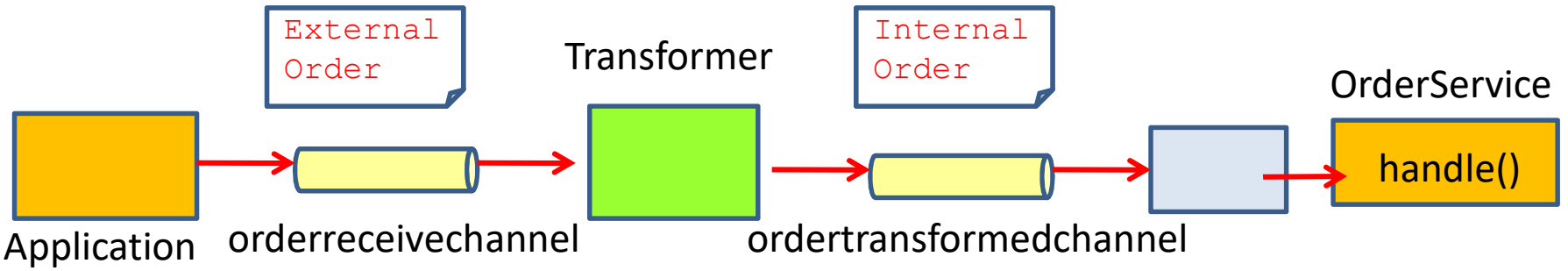
```
Order order = new Order();
order.addOrderLine(new OrderLine(4, new Product("XH-456","MP3
                   player",125.75)));
order.addOrderLine(new OrderLine(2, new Product("GH-428","LED 3D
                   TV",675.85)));
Message<Order> orderMessage = MessageBuilder.withPayload(order).build();

gateway.handleRequest(orderMessage);
```

```
OrderLinesService receiving orderline: quantity = 4 , product = MP3 player
OrderLinesService receiving orderline: quantity = 2 , product = LED 3D TV
OrderService receiving order:
MP3 player
LED 3D TV
```

TRANSFORMATION

Transformer

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

<transformer  ref="orderTransformerBean" input-channel="orderreceivechannel"
              method="transform" output-channel="ordertransformedchannel"/>
<beans:bean id="orderTransformerBean" class="integration.OrderTransformer" />

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

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



The OrderTransformer

```
public class OrderTransformer {

  public InternalOrder transform (ExternalOrder order){
    if (order.getType().equals("large")){
      return new LargeOrder(order.getOrderNumber(), order.getAmount());
    }
    else if (order.getType().equals("rush")){
      return new RushOrder(order.getOrderNumber(), order.getAmount());
    }
    return null;
  }
}
```

```
public class InternalOrder {
  private String orderNumber;
  private double amount;
  private String type;

...
}
```

```
public class InternalOrder {
  private String orderNumber;
  private double amount;
...
}
```

The OrderService

```
public class OrderService {
  public void handleOrder(LargeOrder largeorder) throws Exception {
    System.out.println("OrderService receiving large order: "+
                       largeorder.toString());
  }
  public void handleOrder(RushOrder rushOrder) throws Exception {
    System.out.println("OrderService receiving rush order: "+
                       rushOrder.toString());
  }
}
```

The application

```
ExternalOrder order = new ExternalOrder("H-234-X56",1245.75,"large");
ExternalOrder order2 = new ExternalOrder("H-234-X57",600.65,"rush");

Message<ExternalOrder> message1 = MessageBuilder.withPayload(order).build();
Message<ExternalOrder> message2 =
    MessageBuilder.withPayload(order2).build();

gateway.handleRequest(message1);
gateway.handleRequest(message2);
```

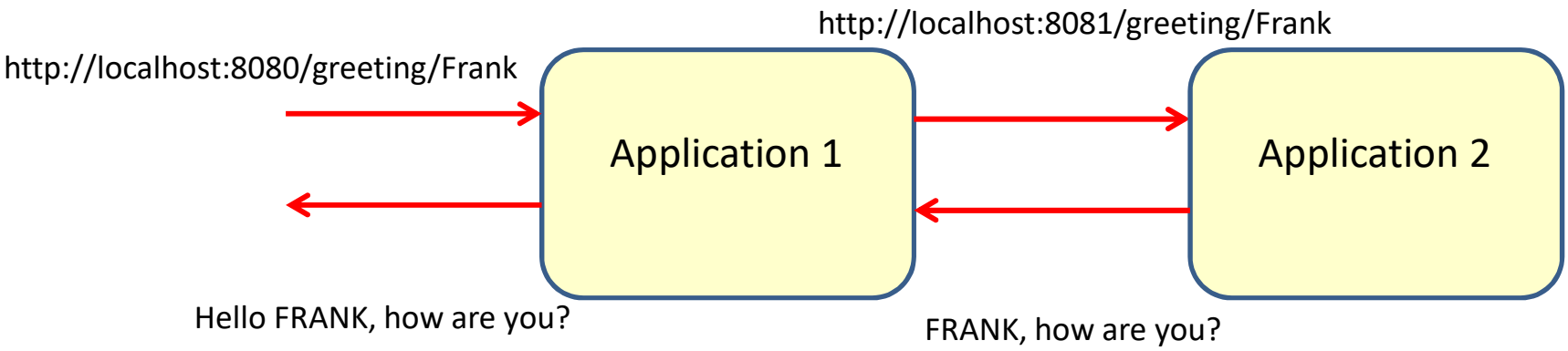
```
OrderService receiving large order: order: nr=H-234-X56 amount=1245.75
OrderService receiving rush order: order: nr=H-234-X57 amount=600.65
```

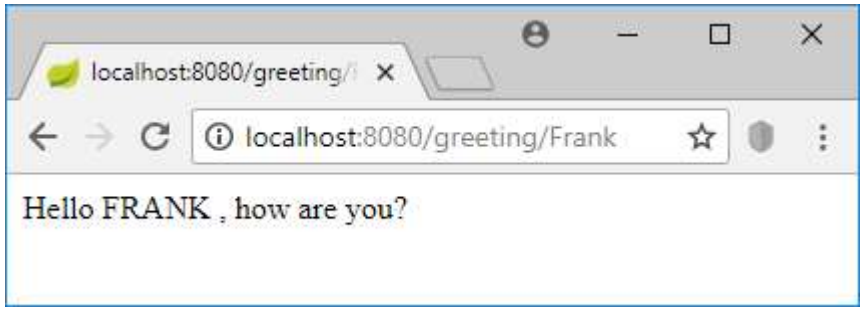
Adapters

Spring integration adapters

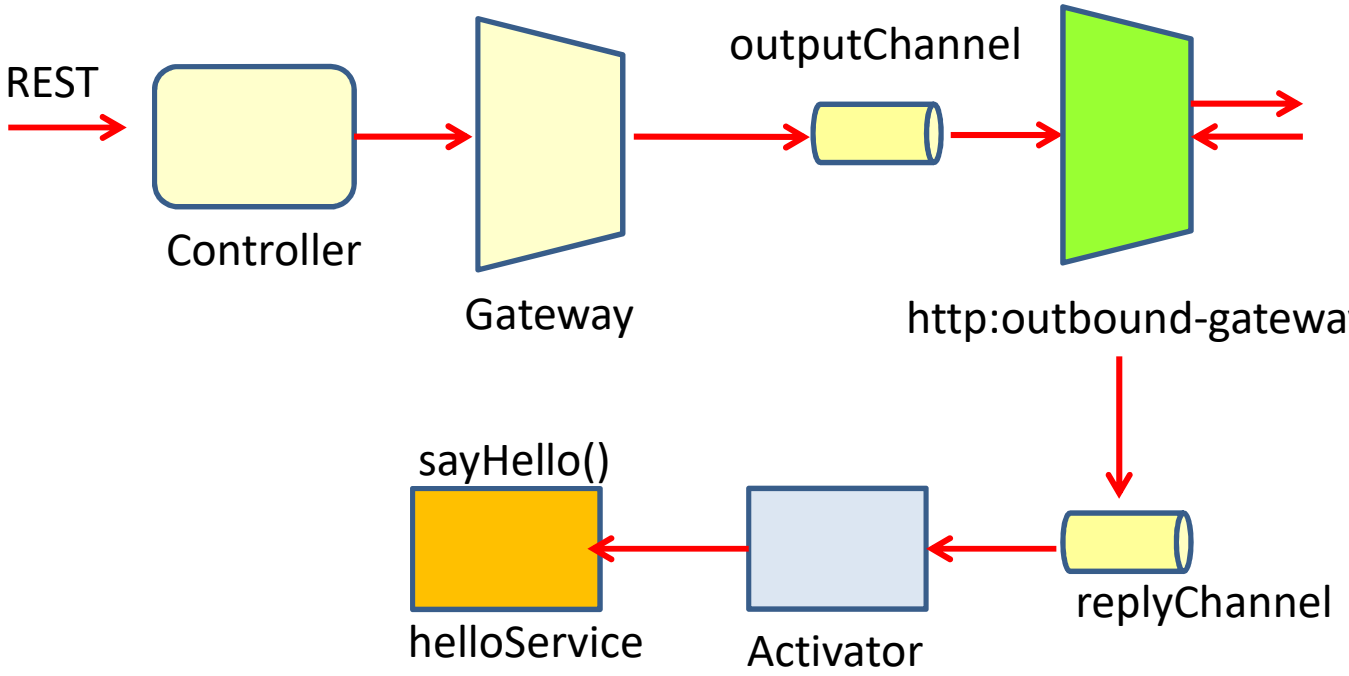
- File
- FTP
- HTTP
- Mail
- TCP and UDP
- JDBC
- JMS
- RMI
- Web services
- …

Http sender adapter





Application 1



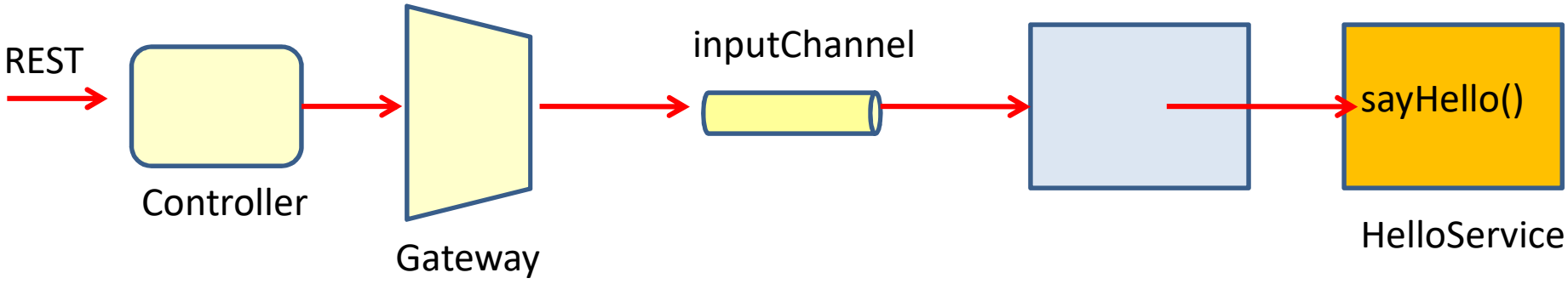

```
public class HelloService {
  public String sayHello(String name) throws Exception {
    System.out.println("Hello " + name);
    return "Hello " + name;
  }
}
```

Application 2

application.properties

```
server.port=8081
```

```
public class HelloService {
  public String sayHello(String name) throws Exception {
    System.out.println("Hi " + name);
    return name+" , how are you?";
  }
}
```



Application 1

```
<channel id="replyChannel"/>
<channel id="outputChannel"/>

<service-activator input-channel="replyChannel"
                   ref="helloService"
                   method="sayHello"/>

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

<int-http:outbound-gateway
      request-channel="outputChannel"
      reply-channel="replyChannel"
      url="http://localhost:8081/greeting/{name}"
      http-method="GET"
      expected-response-type="java.lang.String">
   <int-http:uri-variable name="name" expression="payload"/>
</int-http:outbound-gateway>
```

Application 1

```
@RestController
public class Controller {
  @Autowired
  private GreetingGateway gateway;

  @RequestMapping("/greeting/{name}")
  public String getGreeting(@PathVariable("name") String name) {
    Message<String> helloMessage =
                    MessageBuilder.withPayload(name.toUpperCase()).build();

    String result = gateway.handleRequest(helloMessage);
    return result;
  }
}
```

Application 1

```
@MessagingGateway
public interface GreetingGateway {

@Gateway(requestChannel = "outputChannel")
  String handleRequest(Message<String> message);
}
```

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

Application 2

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

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

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

application.properties

```
server.port=8081
```

```
public class HelloService {
  public String sayHello(String name) throws Exception {
    System.out.println("Hi " + name);
    return name+" , how are you?";
  }
}
```

Application 2

```
@RestController
public class Controller {
  @Autowired
  private GreetingGateway gateway;

  @RequestMapping("/greeting/{name}")
  public String getGreeting(@PathVariable("name") String name) {
    Message<String> helloMessage =
                    MessageBuilder.withPayload(name.toUpperCase()).build();

    String result = gateway.handleRequest(helloMessage);
    return result;
  }
}
```

```
@MessagingGateway
public interface GreetingGateway {

@Gateway(requestChannel = "inputChannel")
  String handleRequest(Message<String> message);
}
```

Main point

- Spring integration supports all different integration patterns:
  - Message channels
  - Routers
  - Filters
  - Splitters
  - Transformers
  - …
- Pure Consciousness is the home of all the laws of nature, field of all possibilities.

Connecting the parts of knowledge with the wholeness of knowledge

1. Spring integration is a framework that can run both inside and outside your application.
2. Spring integration separates the standard integration structure (in XML) from the specific integration logic (in POJO's).

3. **Transcendental consciousness** is the field of all possibilities.

- **Wholeness moving within itself:** In unity consciousness one realizes that the perfect underling structure of the entire creation is just the same structure of one's own pure consciousness.