



padrões  
essenciais  
de  
mensageria  
para  
**integração**  
de sistemas

**helder**darocha

*helder@summa.com.br*

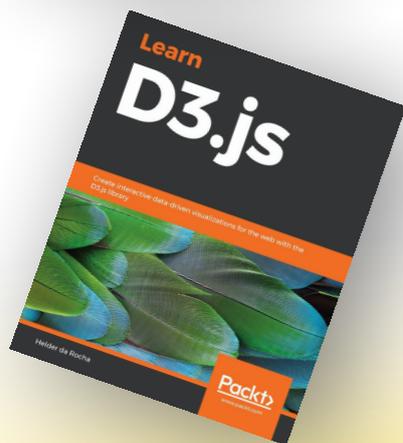
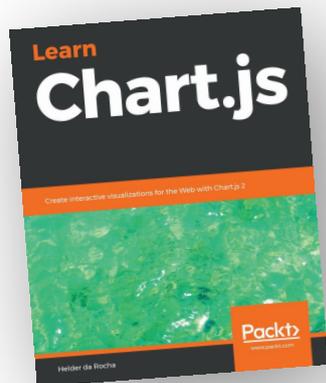
# Quem sou eu? Who am I? Кто я?

## Helder da Rocha

Tecnologia \* Ciência \* Arte

HTML & tecnologias Web desde 1995

Autor de cursos e livros sobre  
Java, XML e tecnologias Web



[argonavis.com.br](http://argonavis.com.br)

[helderदारocha.com.br](http://helderदारocha.com.br)



# Por que integrar sistemas?

- Aplicações interessantes raramente vivem **isoladas**
  - **Sincronizar** emails, calendários, etc.
  - **Vincular** portal a um aplicativo de controle de estoque
  - **Integrar** com serviços e dados na nuvem & IoT
- Viabilizar **comunicação** entre sistemas incompatíveis
- Não reinventar a roda
  - Reusar serviços que funcionam bem
  - Aproveitar pontos de integração



# Desafios das soluções de integração



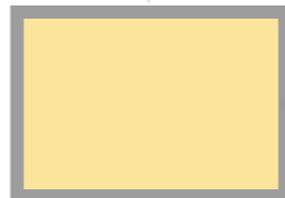
Redes **não são confiáveis**

Redes são **lentas**

Aplicações são **diferentes**

Aplicações **mudam**

Soluções são **complexas**



# Padrões de design

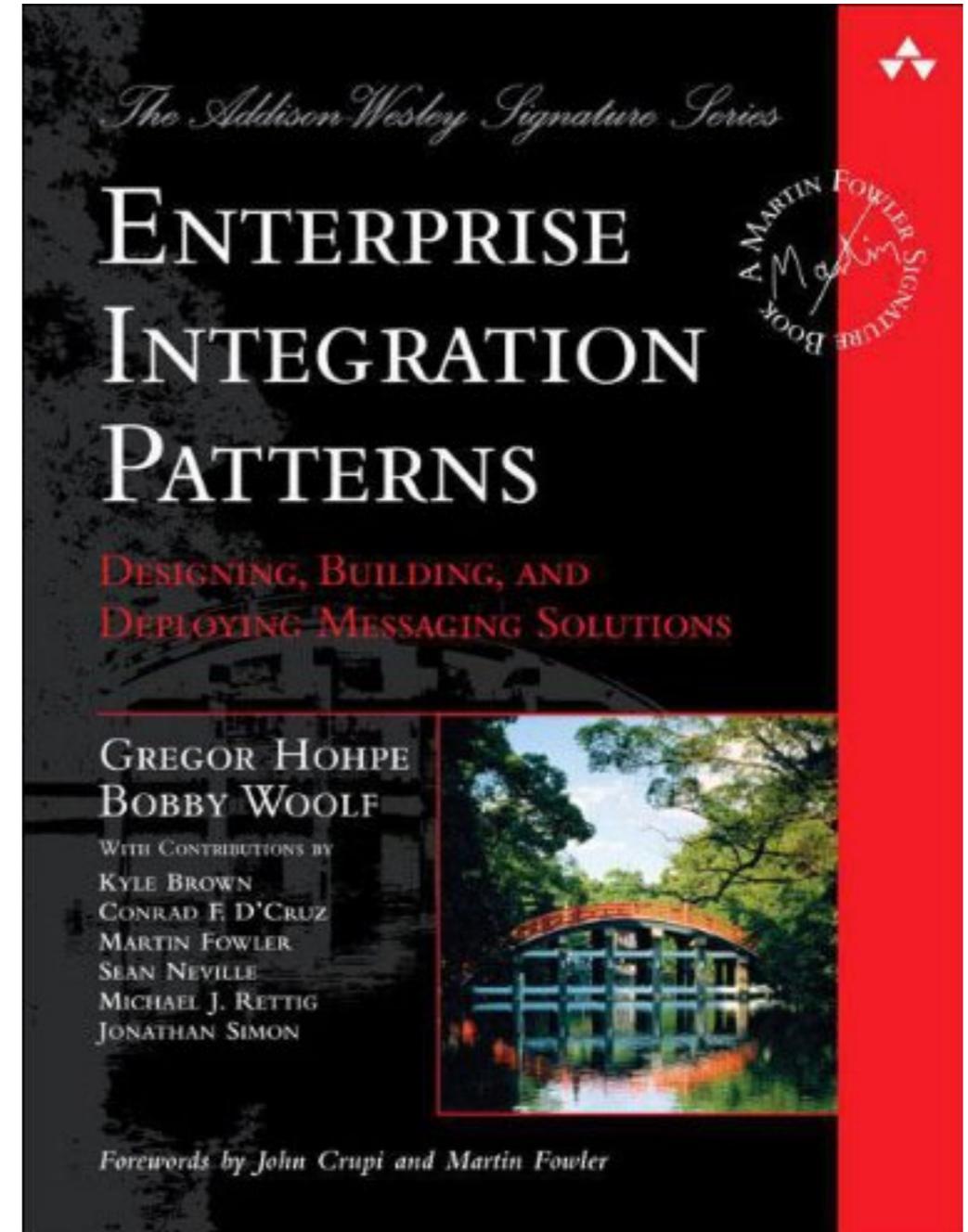
- **Abstração** de alto nível

- Nome / Ícone
- Contexto
- Problema / Solução / Exemplo
- Vocabulário
- Notação
- Padrões relacionados
- Conseqüências da aplicação do padrão



# Padrões de integração

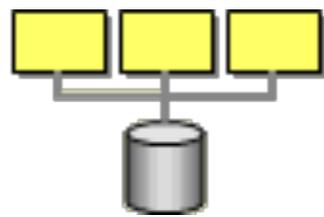
- 65 padrões de integração de sistemas
  - Aplicações em Java, .NET e outras plataformas
- 62 focam em soluções de mensageria
- Foco: minimizar o **acoplamento** entre componentes



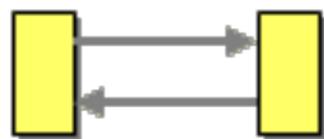
# 4 estratégias básicas de **integração**



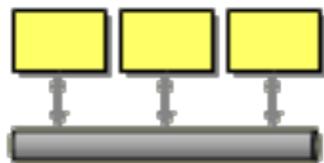
Transferir **arquivos** entre aplicações (1)



Compartilhar um **banco de dados** (2)



Usar interfaces + proxies **RPC** (3)

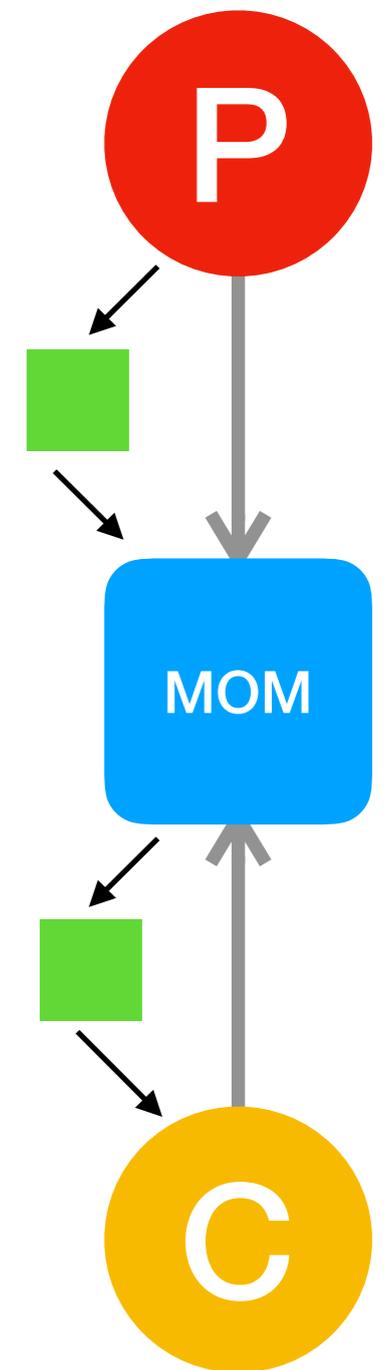


Usar **mensageria** (4)



# O que é **messaging**?

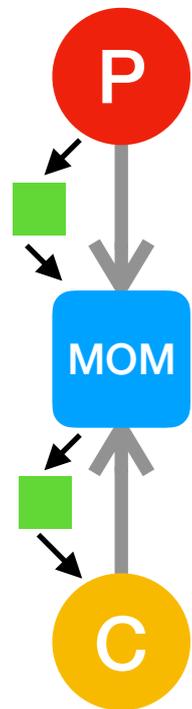
- Comunicação entre máquinas
  - Através de **mensagens** enviados em **canais** (filas) compartilhados entre as máquinas
  - Remetente, **produtor** de mensagens
  - Destinatário, **consumidor** de mensagens
- Mensagem
  - Estrutura de dados (objeto, string, tipo, bytes)
  - **Cabeçalho**, metadados
  - Corpo, **payload**



# O que é um **sistema de mensageria**?

- **Message Oriented Middleware**
  - **Mediator pattern** (GoF) entre consumidores e produtores
  - Administra o sistema (**canais** e **conexões**)

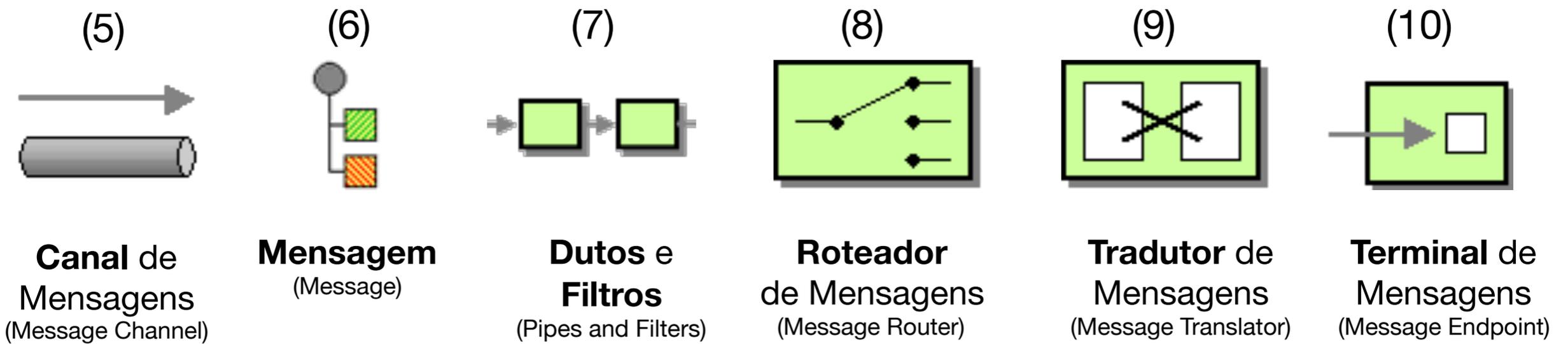
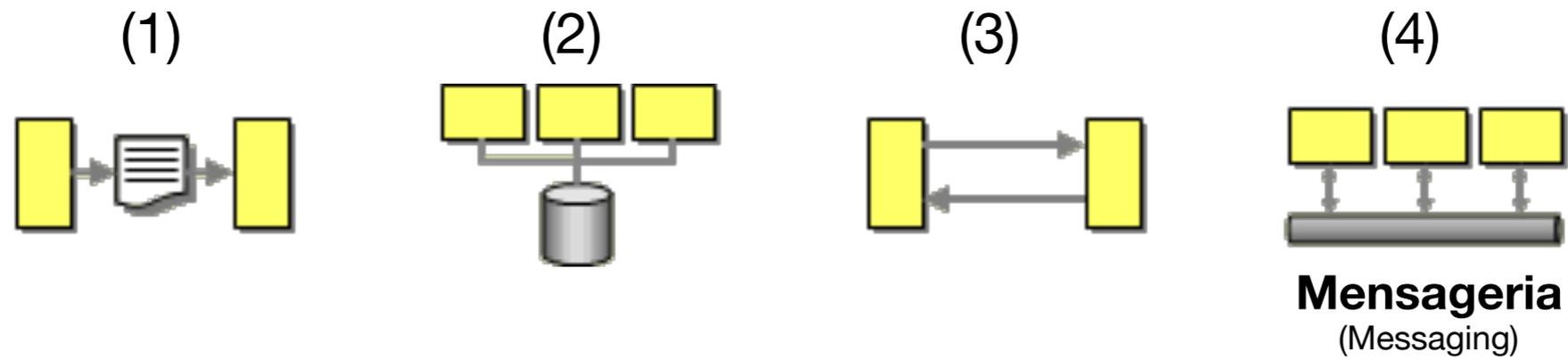
- Etapas



- **Criar** – **produtor** cria a mensagem e preenche com dados
- **Enviar** – **produtor** adiciona mensagem a um canal
- **Entregar** – **sistema de mensageria** transfere a mensagem de uma máquina para a outra tornando-a disponível ao consumidor
- **Receber** – **consumidor** lê a mensagem do canal
- **Processar** – **consumidor** extrai os dados da mensagem



# Padrões de Mensageria (4)

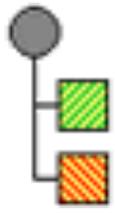


arquitetura

componentes

terminal



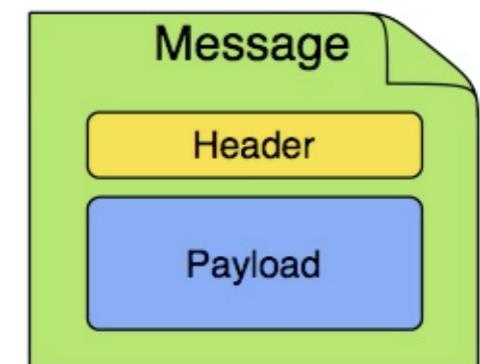
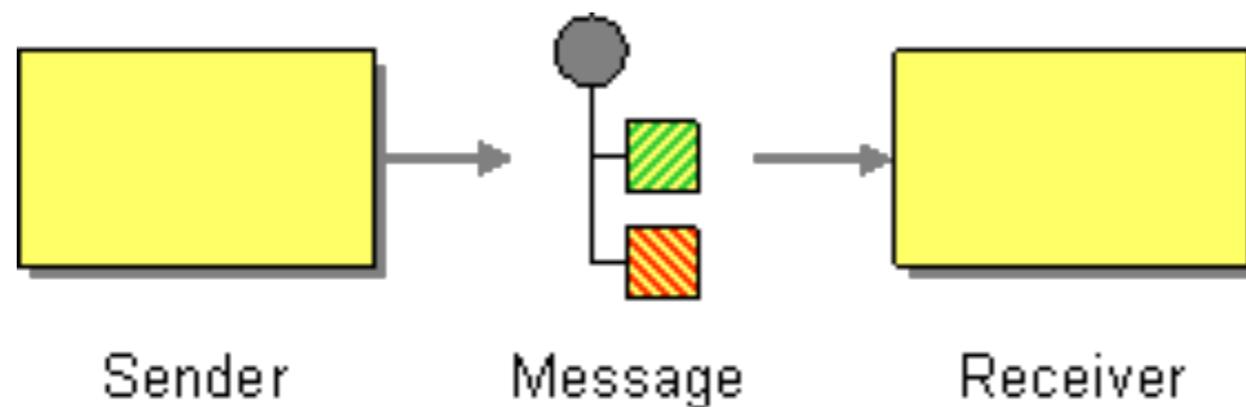


# Message

# 1

**PROBLEMA** “Como é que duas aplicações conectadas por um canal de mensagens podem trocar informação?”

**SOLUÇÃO** “Empacote a informação dentro de uma **Mensagem (Message)**, um registro de dados que o sistema de mensageria pode transmitir através de um canal de mensagens.”



"Envelope"

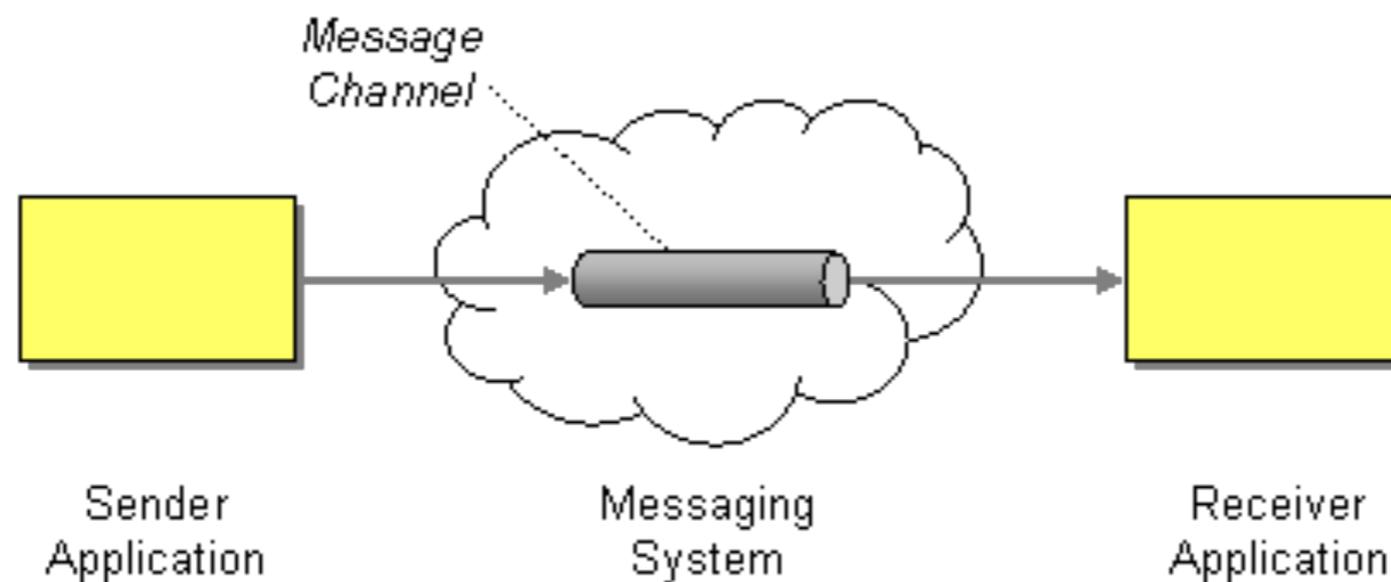




# Message Channel 2

**PROBLEMA** “Como pode uma aplicação comunicar-se com outra aplicação usando mensageria?”

**SOLUÇÃO** “Conectar as aplicações usando um **Canal de Mensagens (Message Channel)**, onde uma aplicação grava informação no canal e a outra lê do canal.”



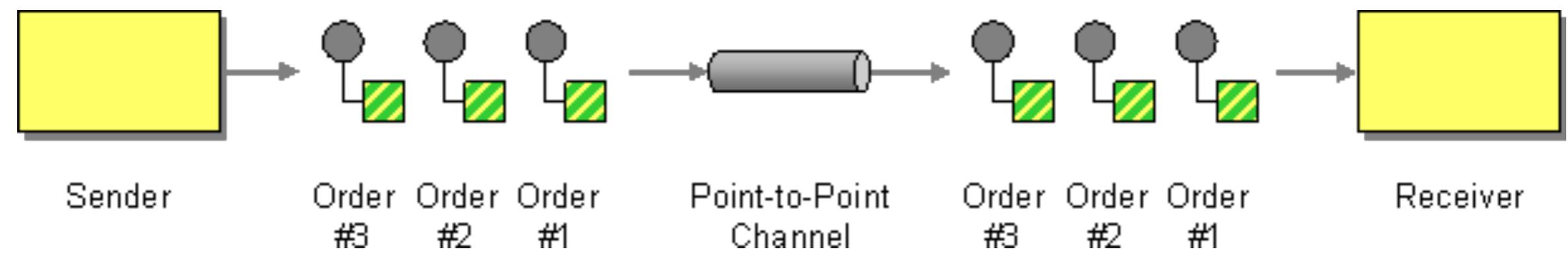
WWW.EAIPATTERNS.COM

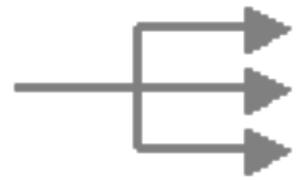
# → Point-to-point channel

3

**PROBLEMA** “Como o remetente pode ter certeza que apenas um destinatário irá receber a mensagem ou executar um comando?”

**SOLUÇÃO** “Envie a mensagem através de um **Canal Ponto-a-Ponto**, que garante que apenas um destinatário irá receber a mensagem.”



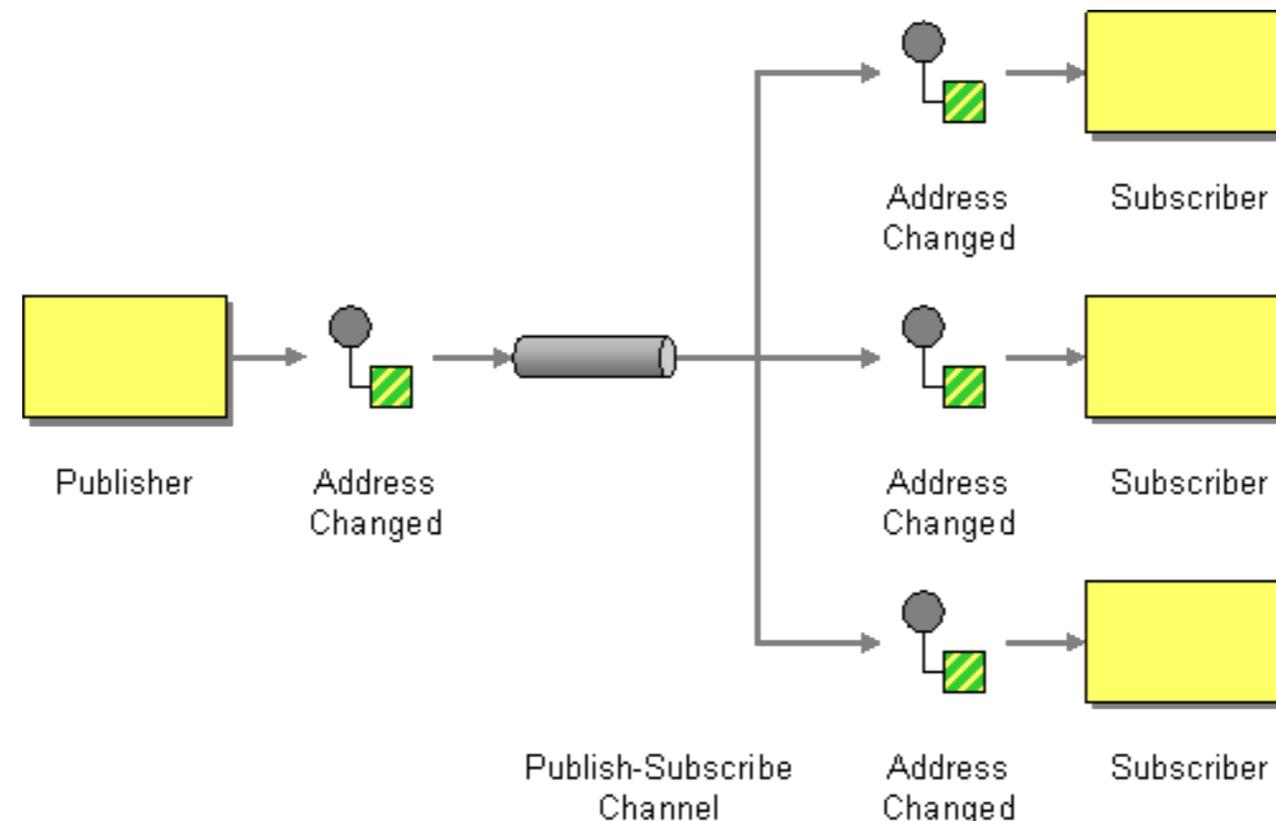


# Publish-subscribe channel

4

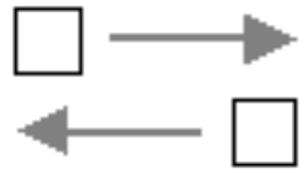
**PROBLEMA** “Como pode um remetente transmitir um evento a todos os destinatários interessados?”

**SOLUÇÃO** “Envie o evento para um **Canal de Difusão**, que entrega uma cópia do evento a cada destinatário.”



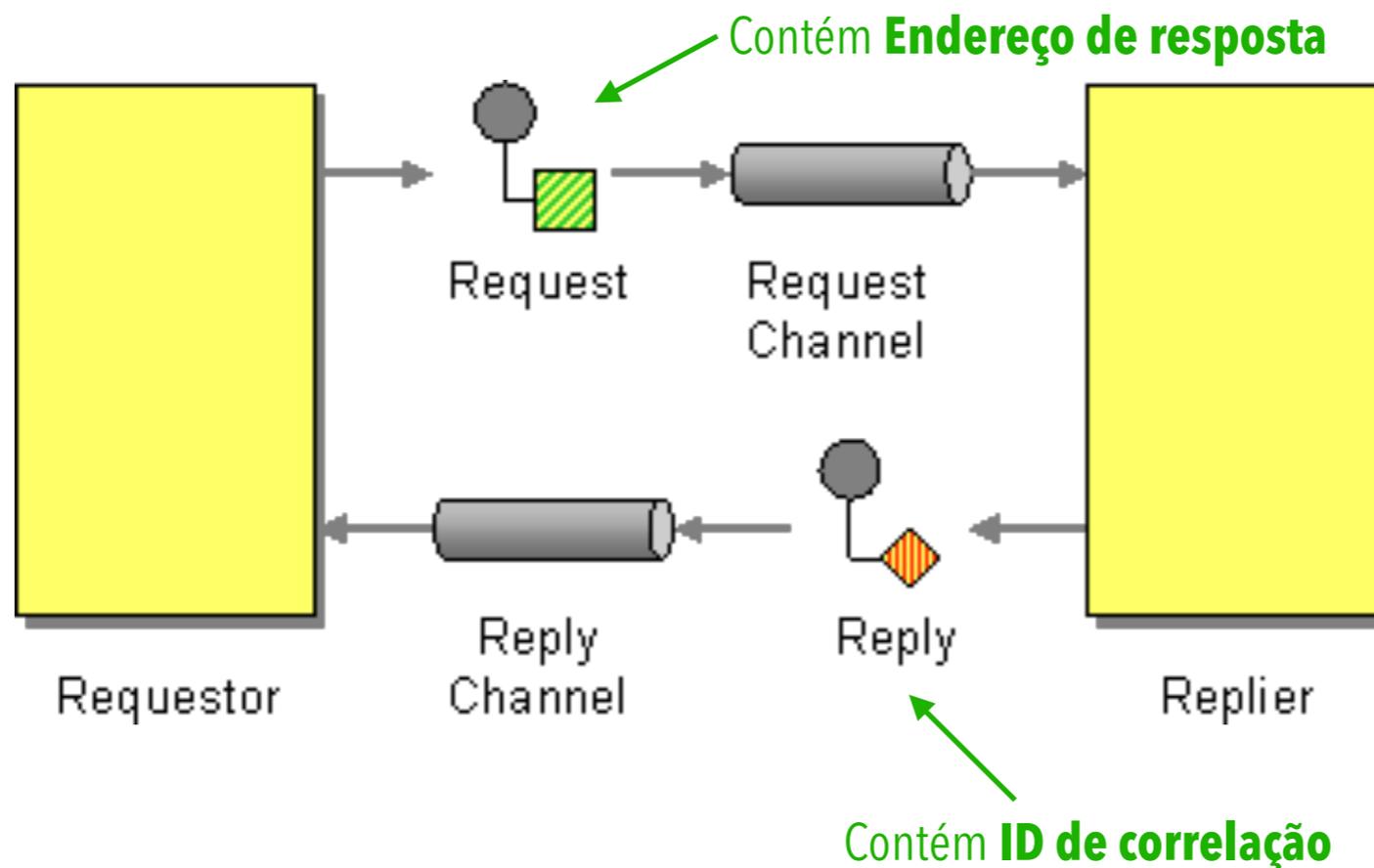
# Request-Reply

5



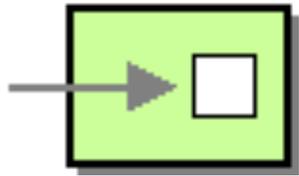
**PROBLEMA** “Quando uma aplicação envia uma mensagem, como ela pode obter uma resposta do servidor?”

**SOLUÇÃO** “Envie um **par de mensagens Requisição-Resposta**, cada uma no seu próprio canal”



Exchange  
:getIn()  
:getOut()

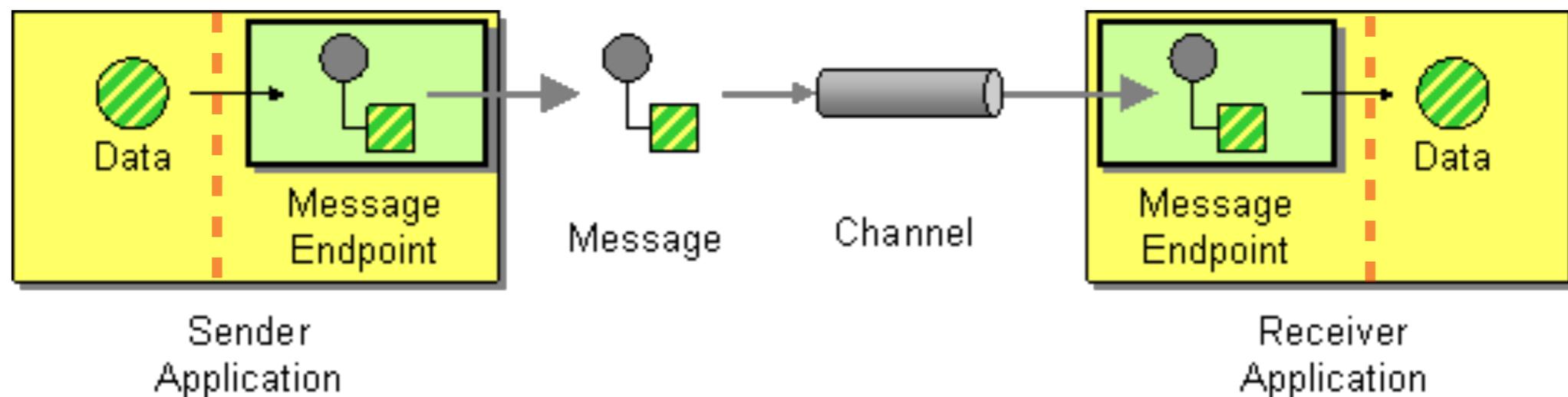


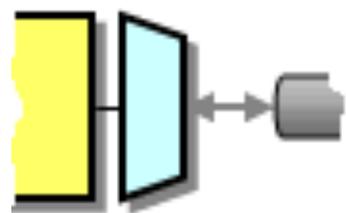


# Message Endpoint 6

**PROBLEMA** “Como pode uma aplicação conectar-se a um canal de mensageria para enviar e receber mensagens?”

**SOLUÇÃO** “Conecte uma aplicação a um canal de mensageria usando um **Terminal de Mensagens**, um cliente do sistema de mensageria que a aplicação pode usar para enviar ou receber mensagens.”



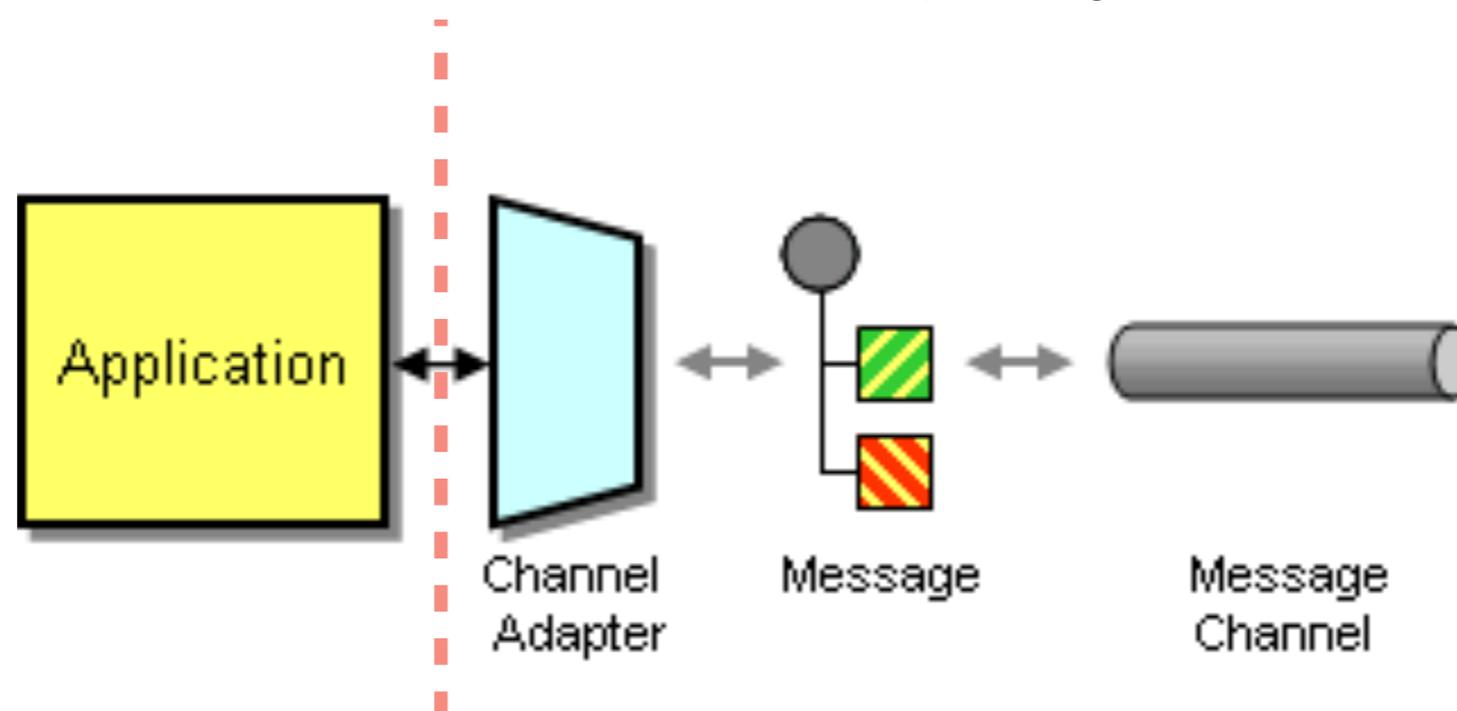


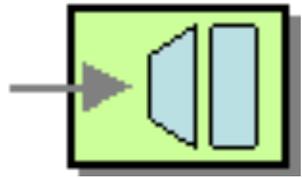
# Channel Adapter

7

**PROBLEMA** “Como conectar uma aplicação ao sistema de messaging para que ela possa enviar e receber mensagens?”

**SOLUÇÃO** “Use um **Adaptador de Canal** que possa acessar a API da aplicação ou seus dados, e publique mensagens em um canal, ou que possa receber mensagens enviadas para um canal e chamar funcionalidades dentro da aplicação”

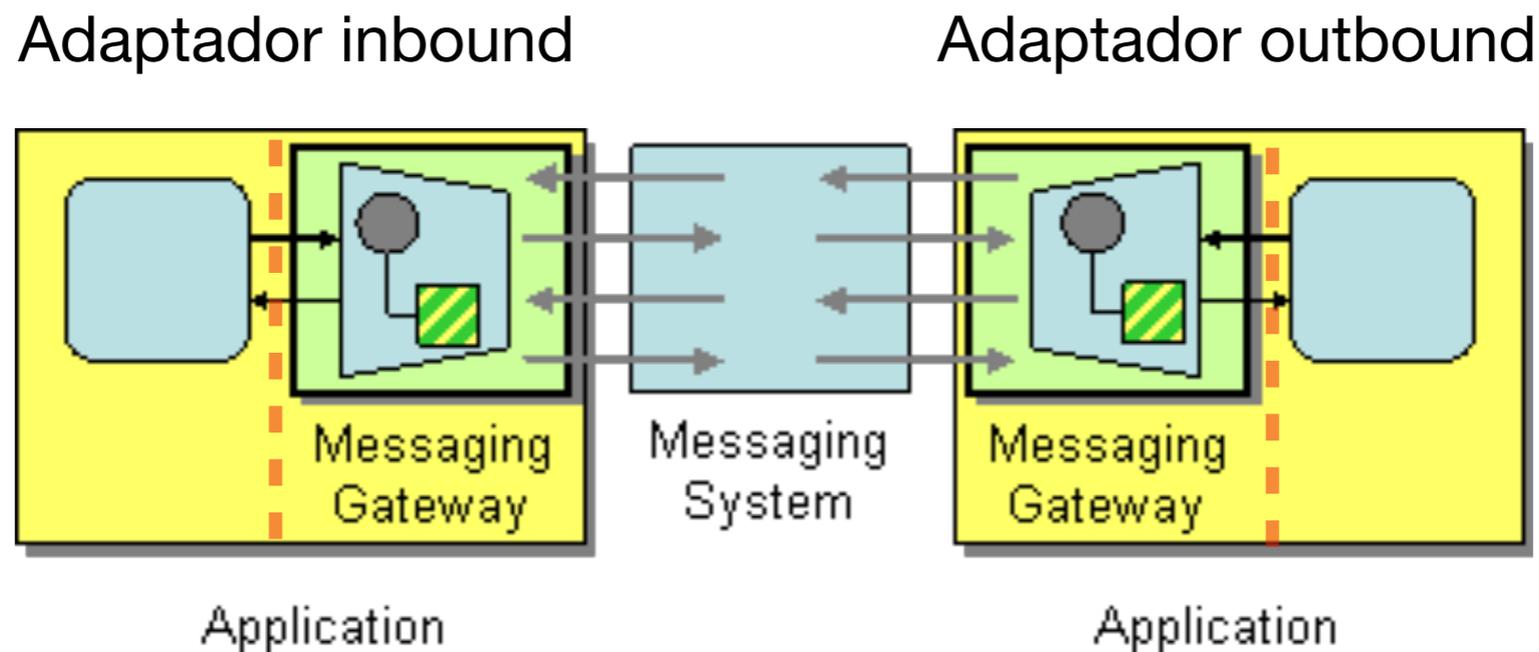


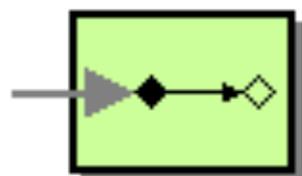


# Messaging Gateway 8

**PROBLEMA** “Como isolar o acesso ao sistema de mensageria do restante da aplicação?”

**SOLUÇÃO** “Use um **Gateway de Mensageria**, uma classe que encapsula chamadas específicas ao sistema de mensageria e expõe uma interface com métodos específicos ao domínio da aplicação”



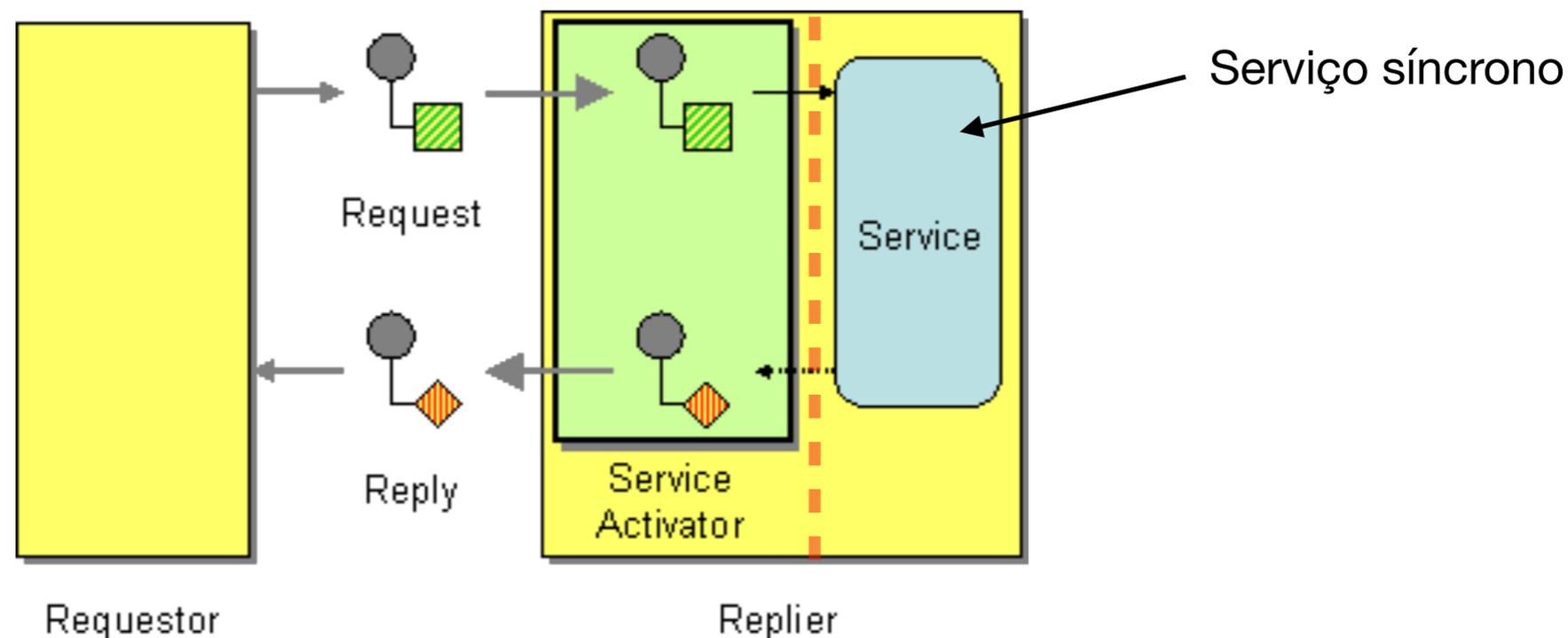


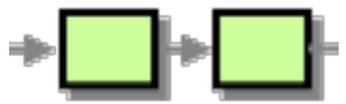
# Service Activator

9

**PROBLEMA** “Como projetar um serviço que possa ser chamado de forma síncrona (sem usar mensageria) ou de forma assíncrona (usando tecnologias de mensageria)?”

**SOLUÇÃO** “Projete um **Ativador de Serviços** que conecte as mensagens do canal ao serviço”

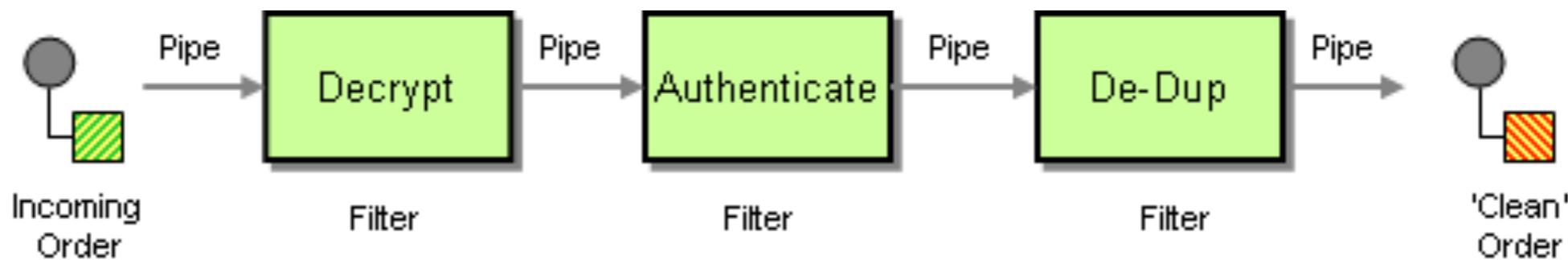


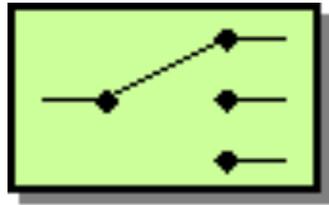


# Arquitetura dutos e filtros

# 10

- Padrão de arquitetura (PEAA\*)
  - Representa a **conexão** de **componentes** (filtros), em série, através de dutos (**canais**) que os conectam
  - Em vez de enviar a mensagem diretamente ao destinatário, pode-se **interceptá-la** em **etapas** intermediárias para validação do seu conteúdo, roteamento, transformação de dados, etc.
  - Cada **componente** age como filtro; cada **canal** age como duto.
  - Permite criar **rotas** (arquitetura de pipeline)

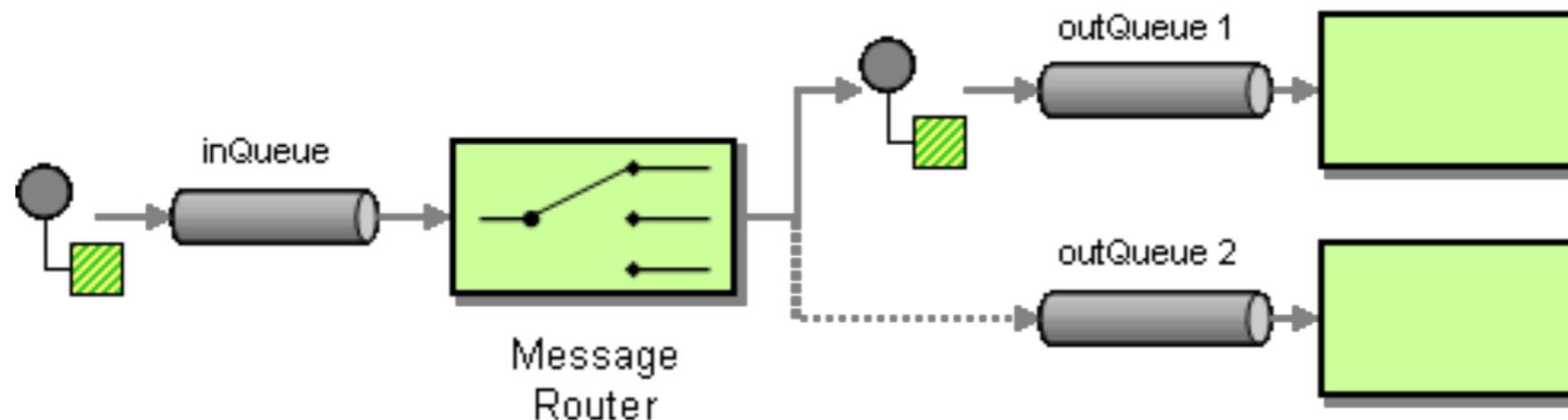




# Message Router 11

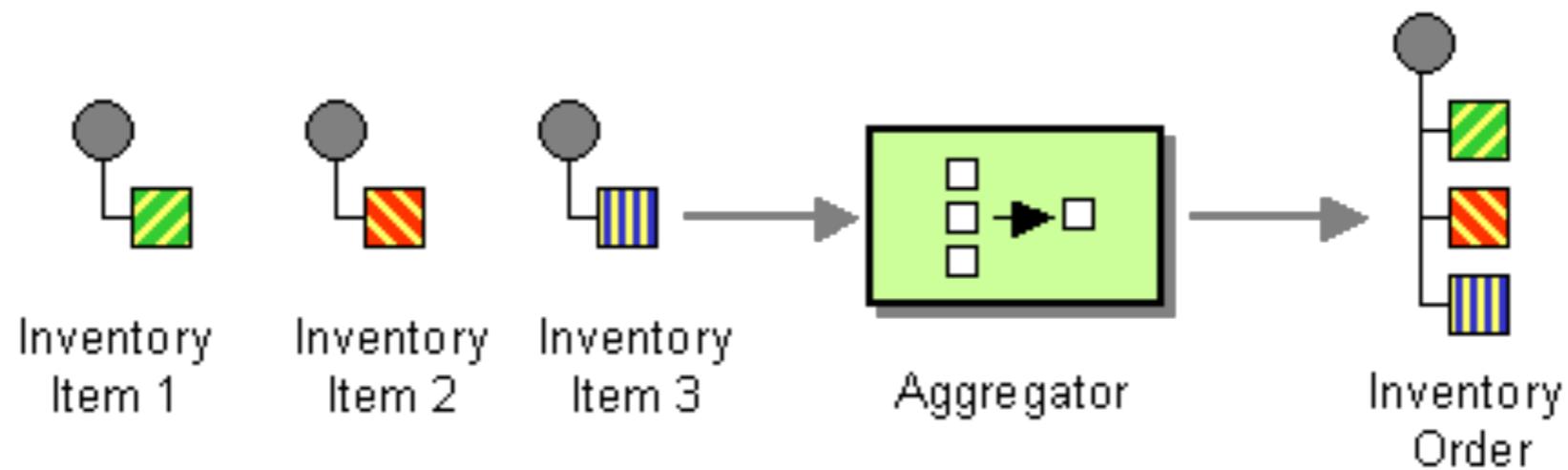
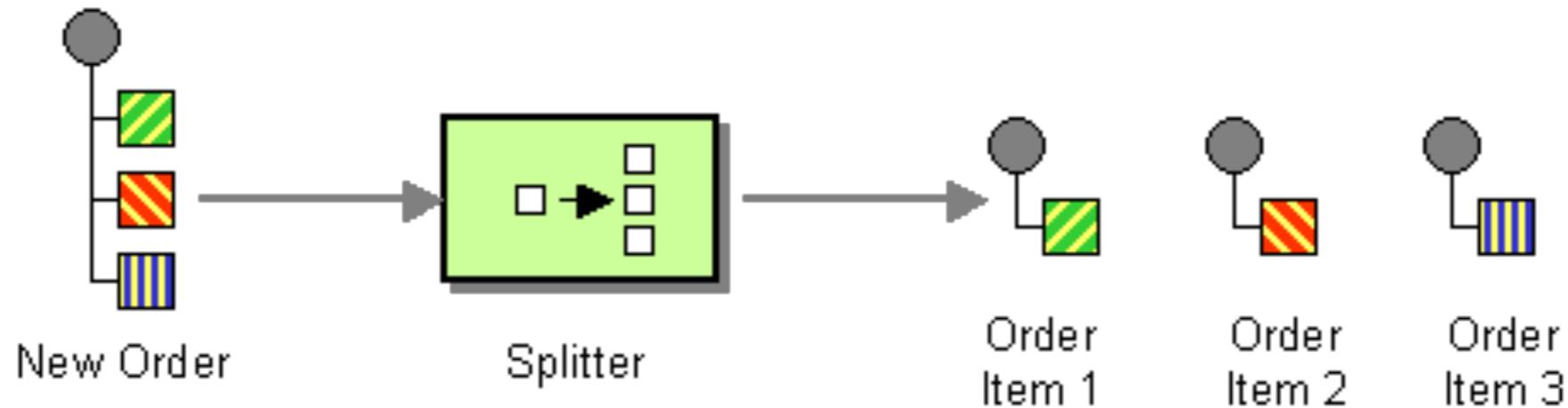
**PROBLEMA** “Como desacoplar passos individuais de processamento de forma que mensagens possam ser passadas para diferentes filtros dependendo de uma série de condições?”

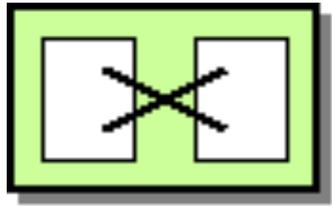
**SOLUÇÃO** “Use um filtro especial, um **Roteador de Mensagens (Message Router)** que consome uma Mensagem de um Canal de Mensagens e publique-a em outro Canal de Mensagens, dependendo de uma série de condições.”



Condições para rotear: algoritmo estático ou dinâmico, conteúdo, eventos externos, etc.

# Splitter e Agregator 12



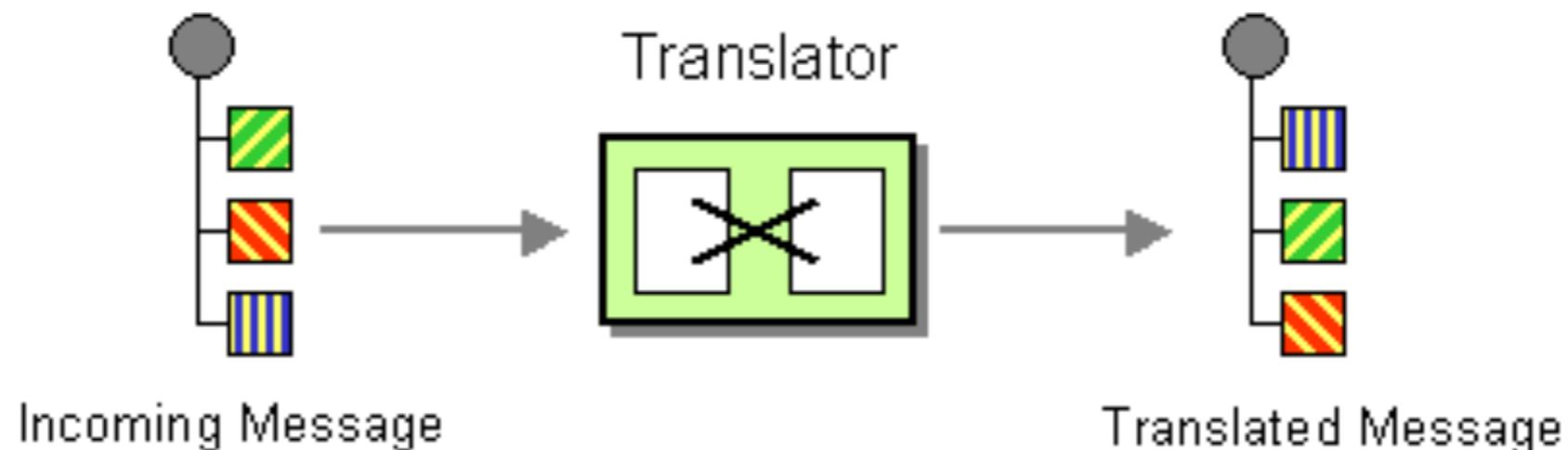


# Message Translator

# 13

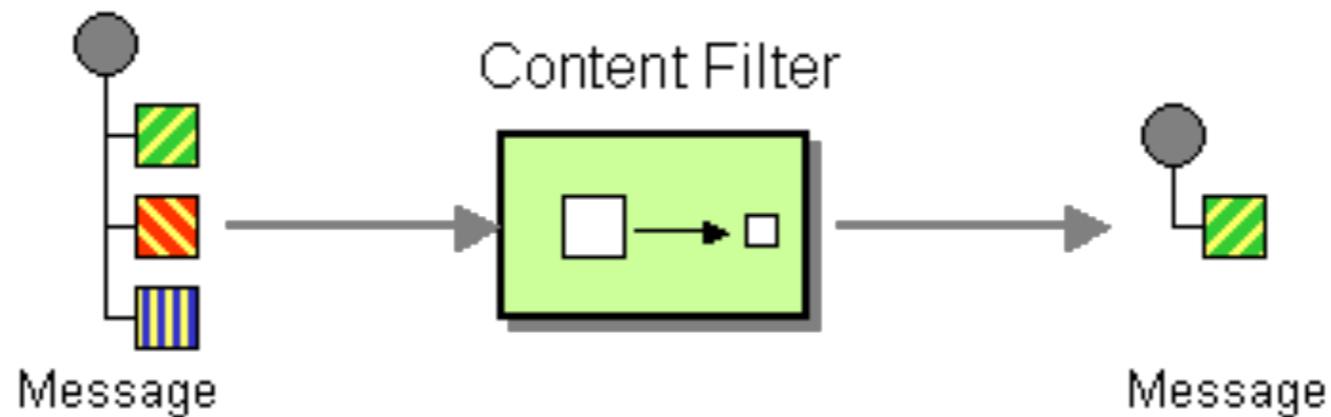
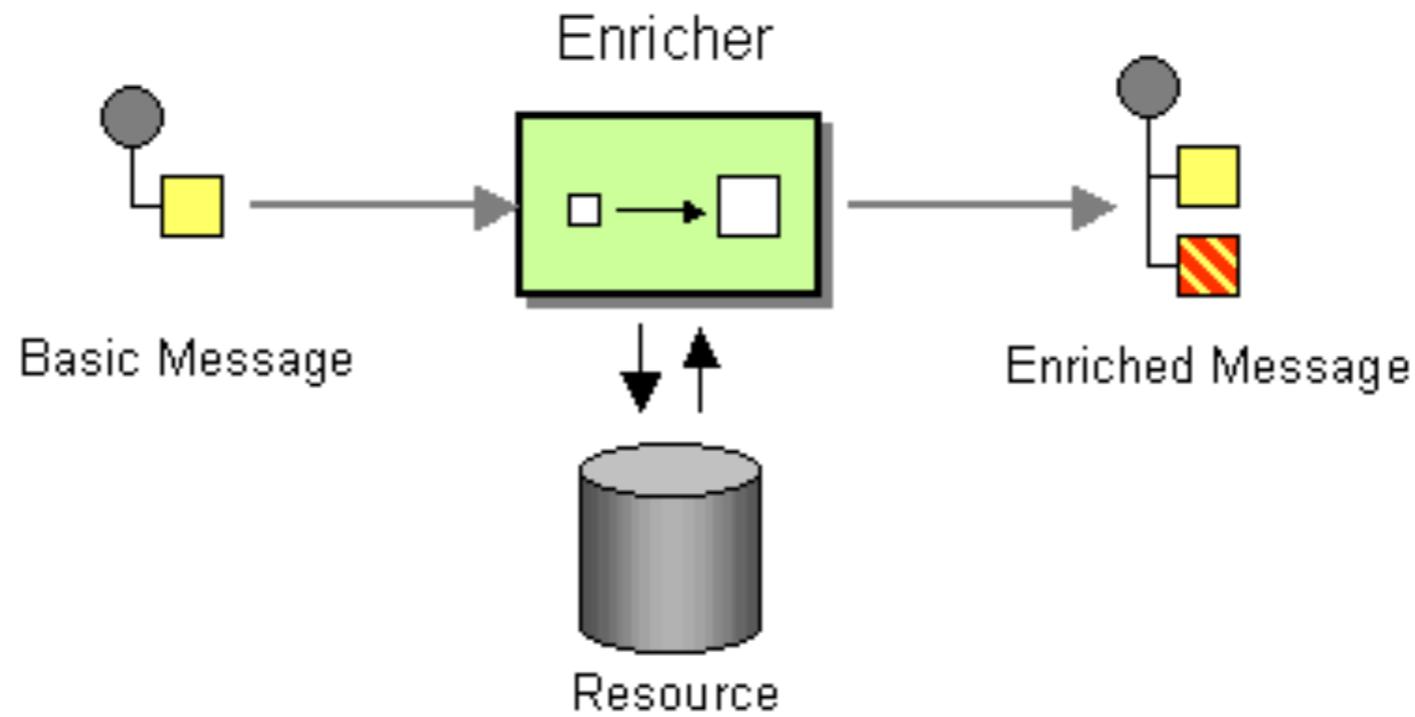
**PROBLEMA** “Como é possível realizar a comunicação entre sistemas que usam formatos de dados diferentes?”

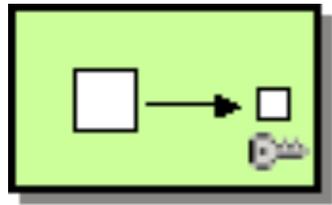
**SOLUÇÃO** “Use um filtro especial, um **Tradutor de Mensagens**, entre outros filtros ou aplicações para traduzir de um formato de dados para outro.”



# Content Enricher + Filter

14



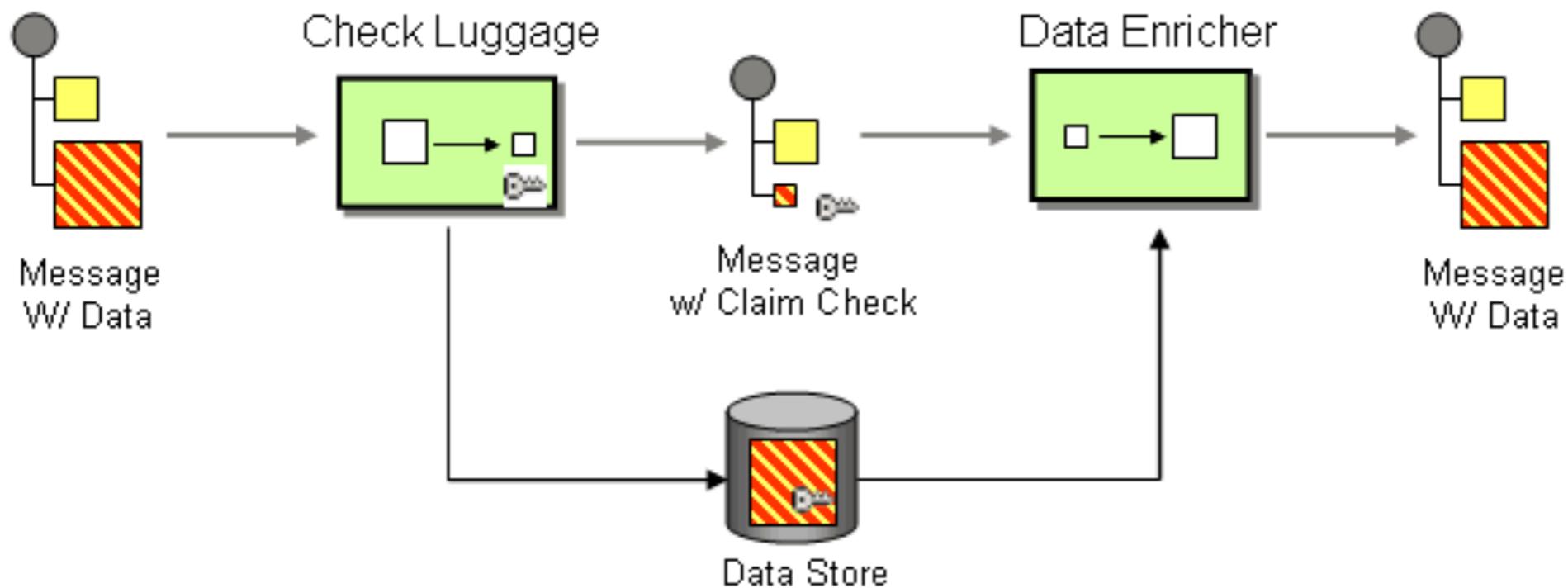


# Claim Check

15

**PROBLEMA** “Como podemos reduzir o volume de dados de uma mensagem enviada através do sistema sem sacrificar o conteúdo da informação?”

**SOLUÇÃO** “Guarde os dados da mensagem em um repositório persistente e passe um **Recibo de Bagagem** para os componentes seguintes. Esses componentes poderão usar o Recibo para recuperar a informação armazenada”



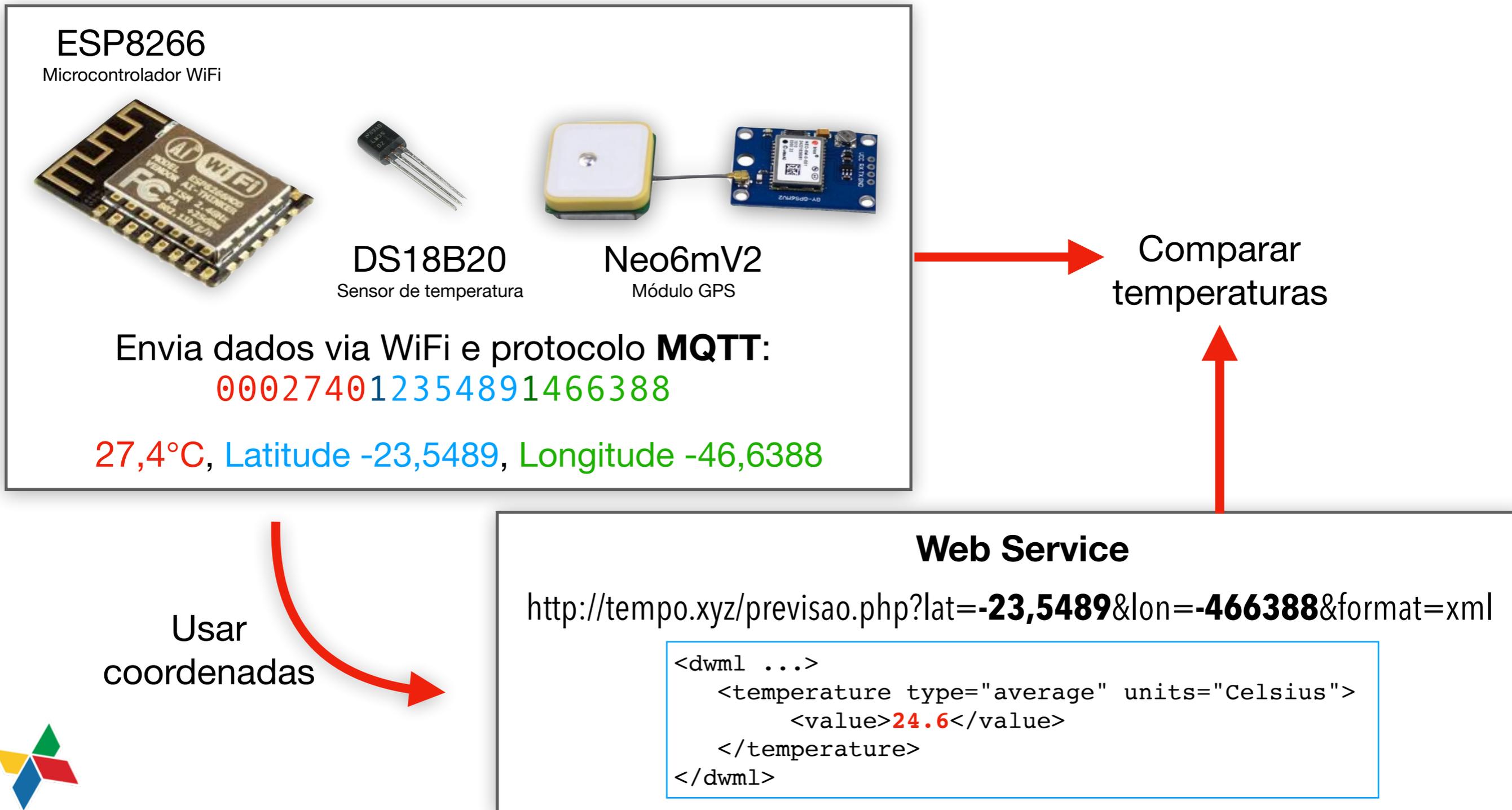
# Como usar os padrões?

- Os padrões são **abstrações de alto nível** que permitem **descrever a solução** de um problema de integração
- Use os padrões para **descrever a arquitetura de rotas** de uma solução de integração
- Não existe solução única
  - Mesmos resultados podem ser alcançados com arquiteturas diferentes (+ conseqüências diferentes)

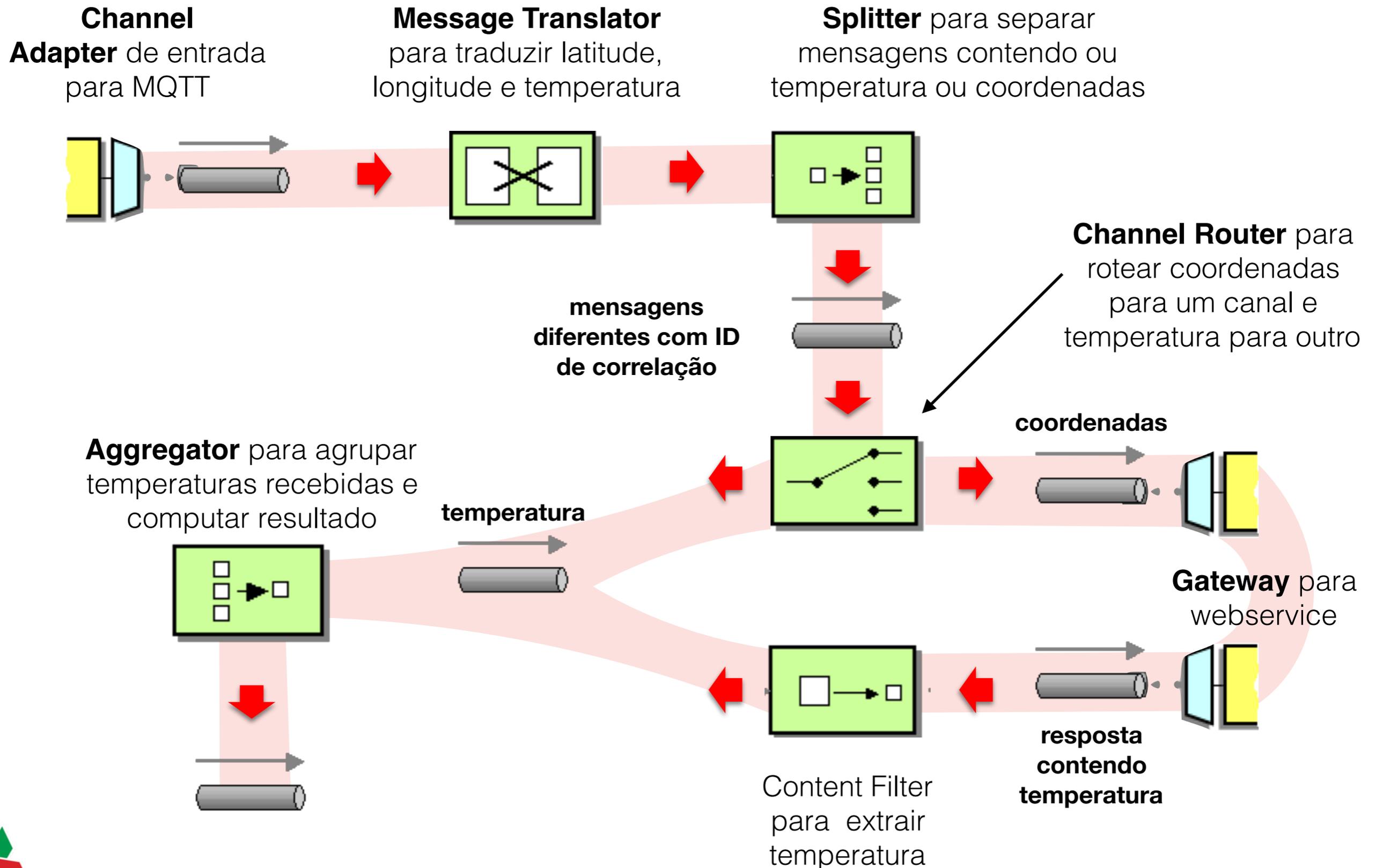


# Problema #1: integrar IoT e webservice

Calcular a **diferença** entre uma temperatura medida e a temperatura média prevista para uma localidade.



# Possível **solução** para problema #1



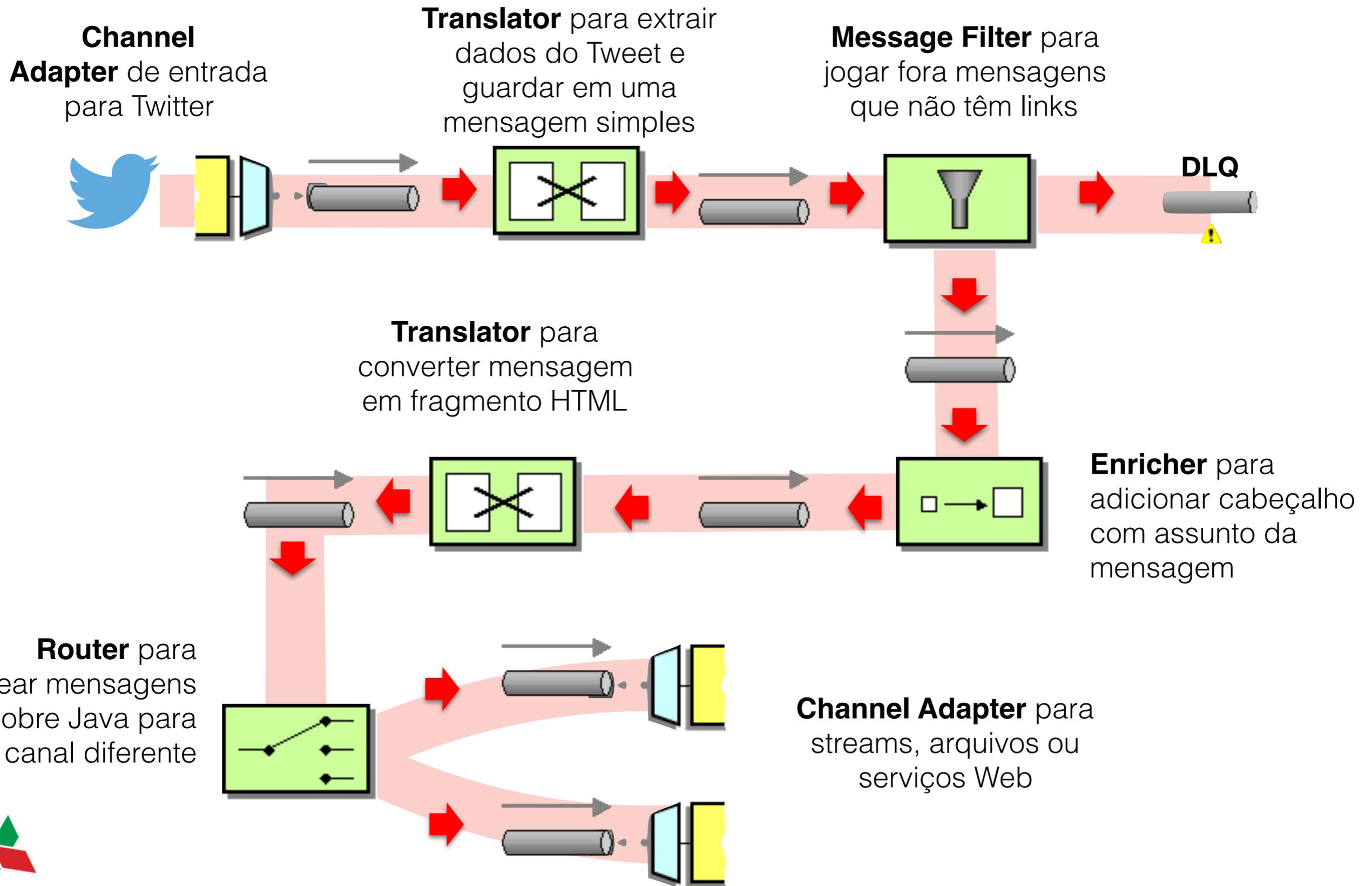
# Problema #2: integrar Twitter com página Web

Criar página Web contendo **tweets** selecionados em tempo real.

- Obter todas as mensagens com hashtag **#TheDevConf** ou **#TDC2019** no Twitter, periodicamente (polling)
- **Processar** apenas mensagens que tiverem [links](#)
- **Estruturar** o conteúdo das mensagens identificando **#hashtags**, **@usuários**, [links](#) e **remetentes**
- **Separar** mensagens que tratam de “**Java**” das demais mensagens
- **Disponibilizar** (arquivo de log e página Web) com **duas** listas de mensagens (Java e outras)



# Possível **solução** para problema #2



# Como implementar?

- Construir solução caseira usando Java e JMS
- Usar um **framework** que implemente padrões de integração de sistemas
- **Alternativas** populares para Java:

**Spring** Integration



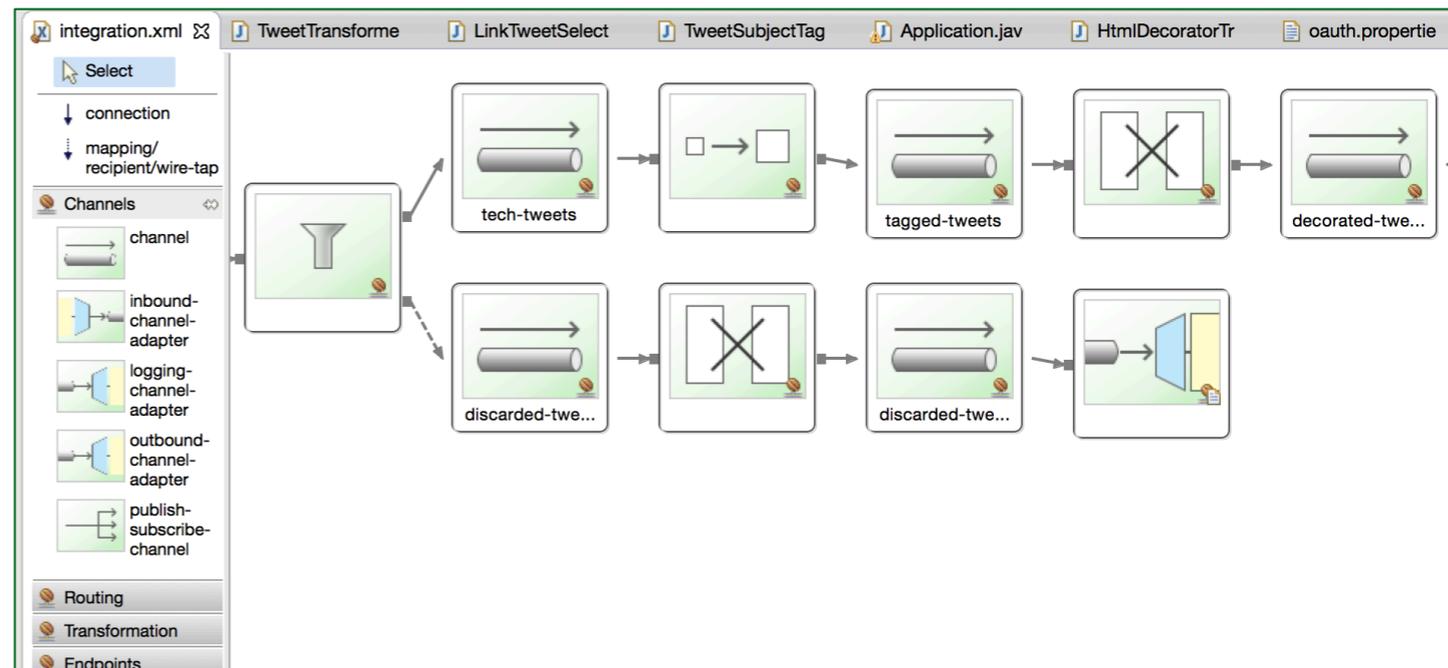
Apache **Camel** (+Spring Boot)



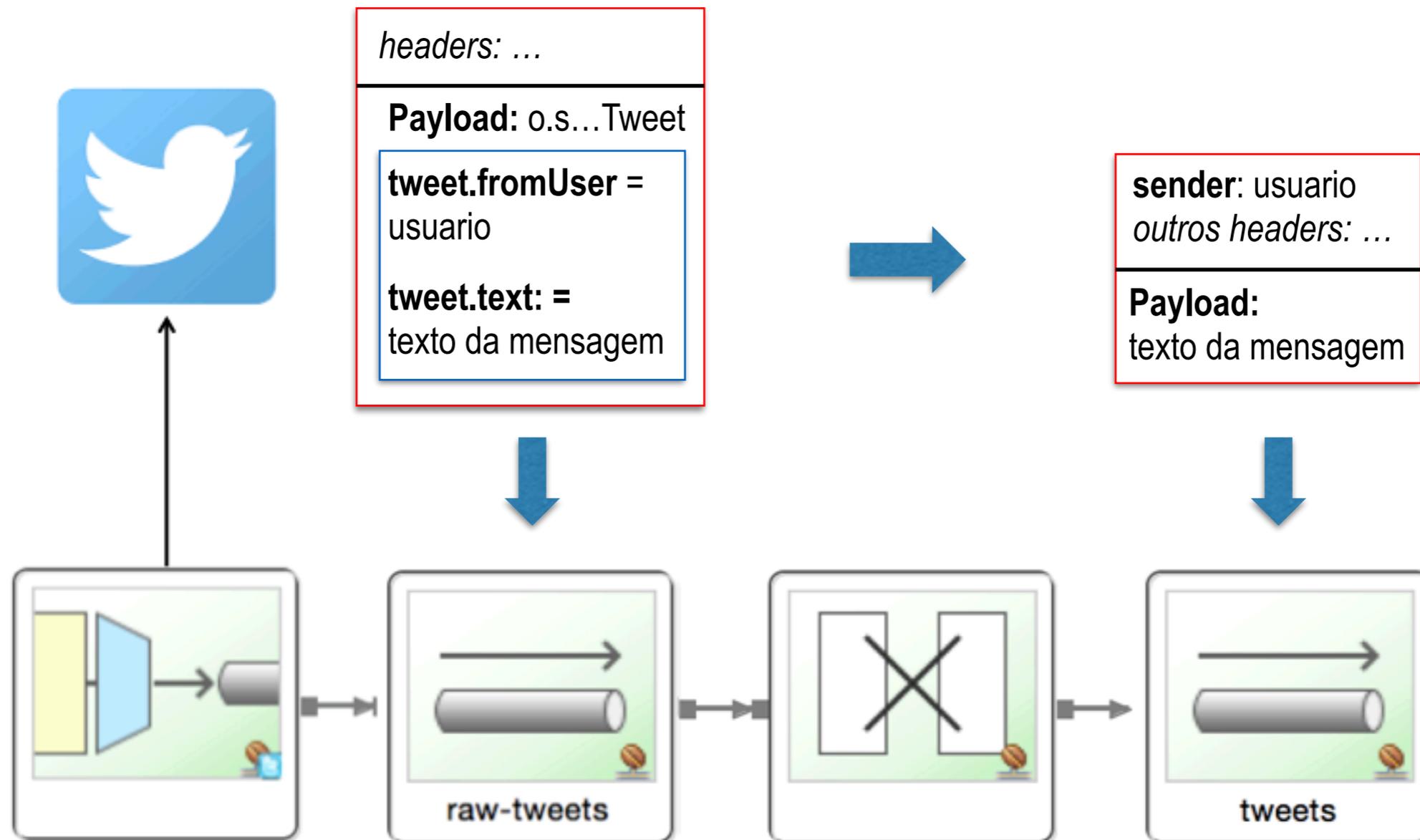


# Spring Framework

- Spring é uma **plataforma Java** construída sobre conceitos **injeção de dependências** (DI) e aspectos
- Componentes Spring vivem em um **container** e são interligados pelo framework usando **DI**
- **Spring Integration** - infraestrutura para mensageria que implementa principais padrões
- Ferramentas para a plataforma Eclipse: **STS**



# Inbound Channel Adapter



Faz query no **Twitter**:  
“**#TheDevConf OR #tdc2019**”  
Põe mensagens no canal **raw-tweets**

Translator: Extrai dados de objeto **Tweet** e cria novo objeto **Message**

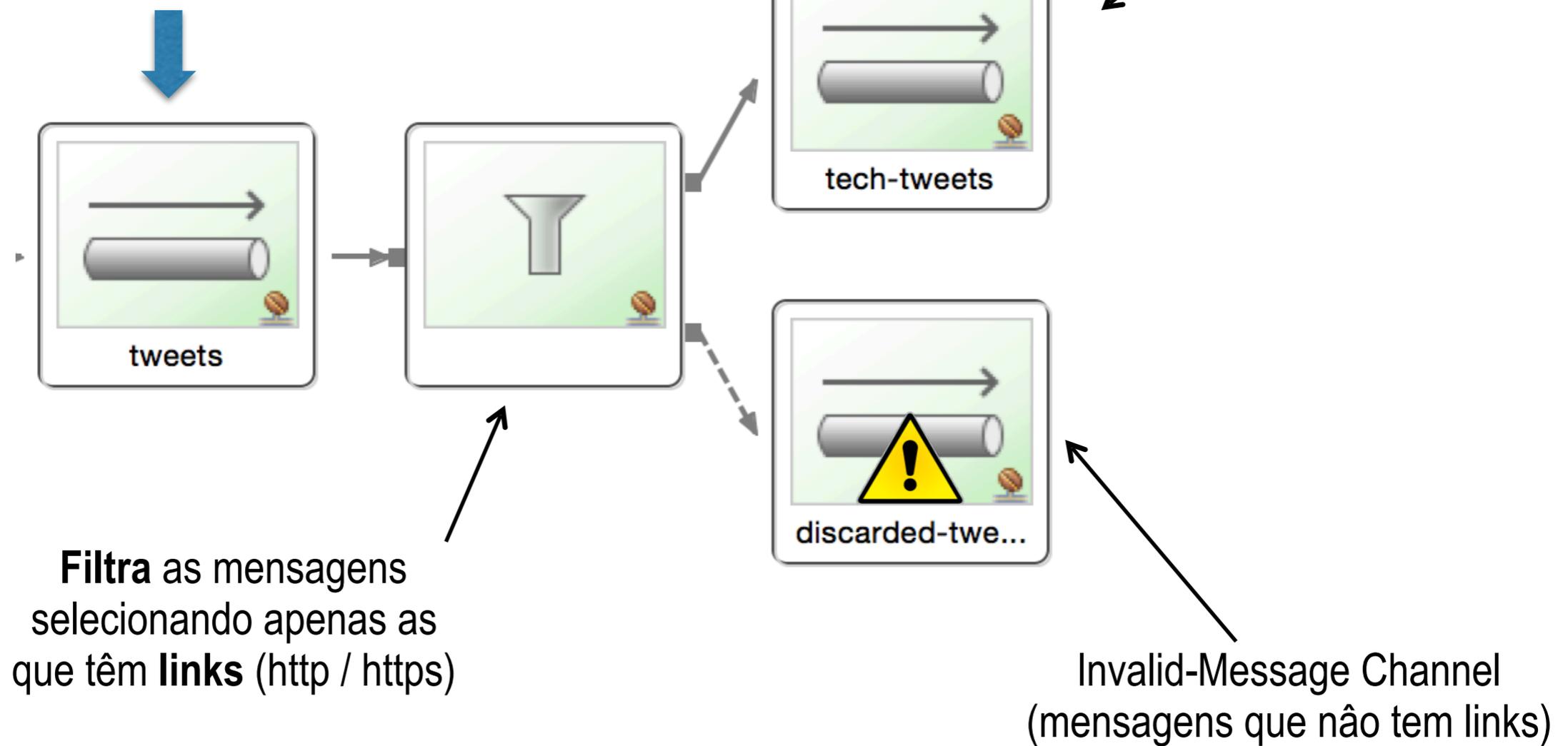


# Message Filter

**sender:** usuario  
*outros headers: ...*

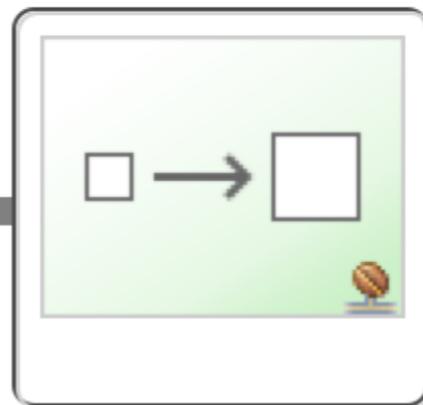
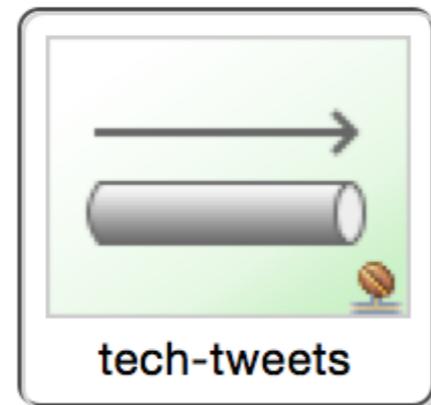
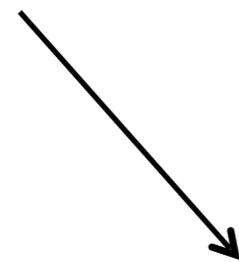
---

**Payload:**  
texto da mensagem



# Content Enricher

Apenas mensagens com links  
estão neste canal



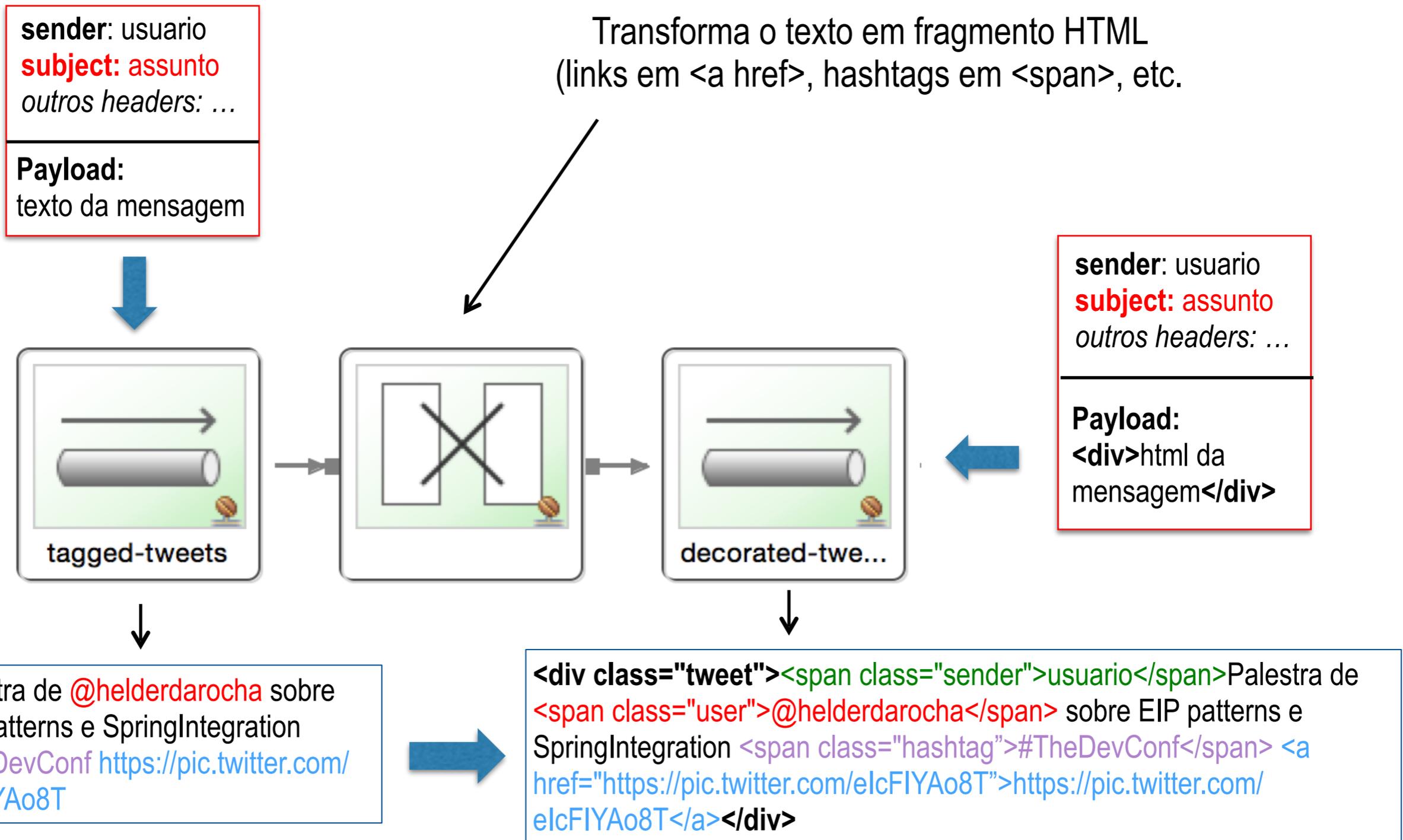
<b>sender:</b> usuario <b>subject:</b> assunto <i>outros headers: ...</i>
<b>Payload:</b> texto da mensagem



Adiciona um cabeçalho para  
**rotular** as mensagens  
por assunto ("java" ou "other")

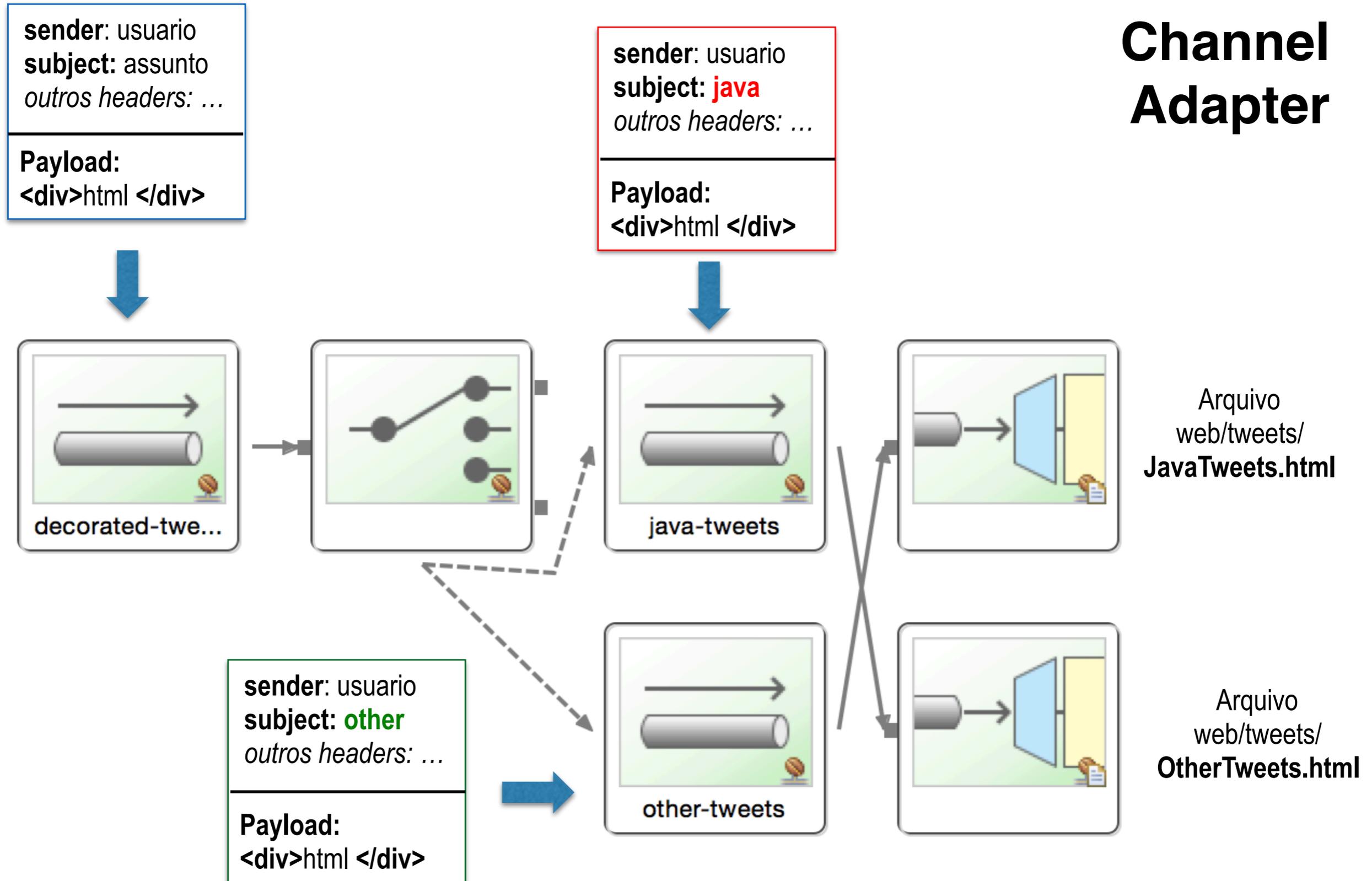


# Message Translator

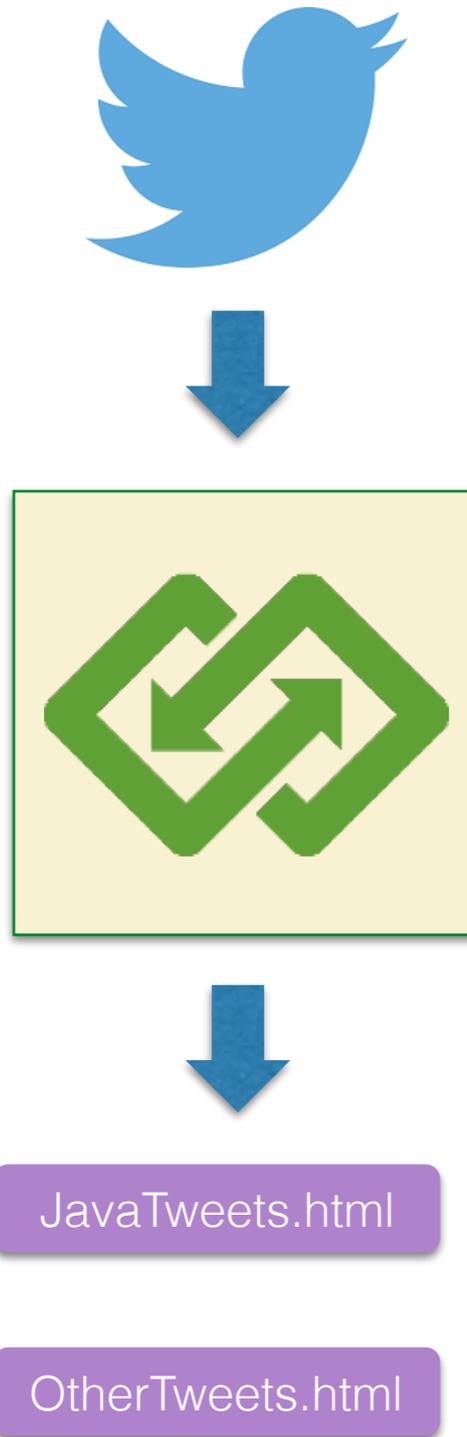


# Message Router

+ Outbound  
**Channel  
Adapter**



# Resultado



Ajax

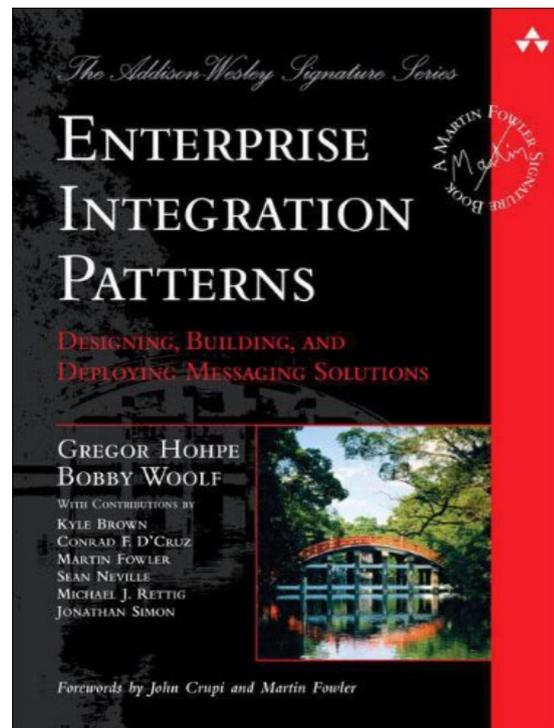
A screenshot of a web browser window. The address bar shows 'localhost:8080'. The page title is '#TheDevConf Twitter feeds'. The main content is titled '#thedeveloper and #tdc2019 tweets'. Below this, there is a section titled 'Java tweets (contain Java-related strings)'. This section contains three tweets, each with the user name highlighted in yellow. The first tweet is from 'Fabicanedo', the second from 'rafaelvcunha', and the third from 'rafaelvcunha'. Below this section is another section titled 'Other tweets', which contains five tweets, also with user names highlighted in yellow. The tweets are from 'DiogosGuedes', 'Dionei\_Piazza', 'myLIMS', 'educostadev', and 'pasimoes'. Each tweet includes the user's name, a handle, the text of the tweet, and a link to the original tweet.



# Conclusões

- **Mensageria** é uma estratégia eficiente para integrar aplicações que não foram projetadas para trabalhar juntas
- É **difícil** usar mensageria
- Padrões de mensageria promovem a **separação de responsabilidades** complexas (processamento, roteamento, etc.) facilitando a descrição da arquitetura de uma **solução** de integração
- Existem **produtos** que implementam os principais padrões, facilitando a implementação de soluções





# Referência

Gregor Hohpe, Bobby Woolf, et al  
**Enterprise Integration Patterns**

<http://www.eaipatterns.com/>



# E-book + código



<https://github.com/helderda Rocha/EIP-Course>



# padrões essenciais de mensageria para integração de sistemas

Baixe esta palestra (depois do TDC) em

**[http://www.argonavis.com.br/download/tdc\\_2019\\_eip.html](http://www.argonavis.com.br/download/tdc_2019_eip.html)**

+Links para outros recursos, código, referências

<https://github.com/helderda Rocha/tutorial-messaging-patterns>



**[www.summa.com.br](http://www.summa.com.br)**

**helder da rocha**

helder@summa.com.br