L'intégration dans le monde des applications Java

ParisJUG

Octobre 2013

Grégory Boissinot

- Directeur technique Zenika Paris
- Formateur certifié Springsource
- Committer Jenkins

@gboissinot

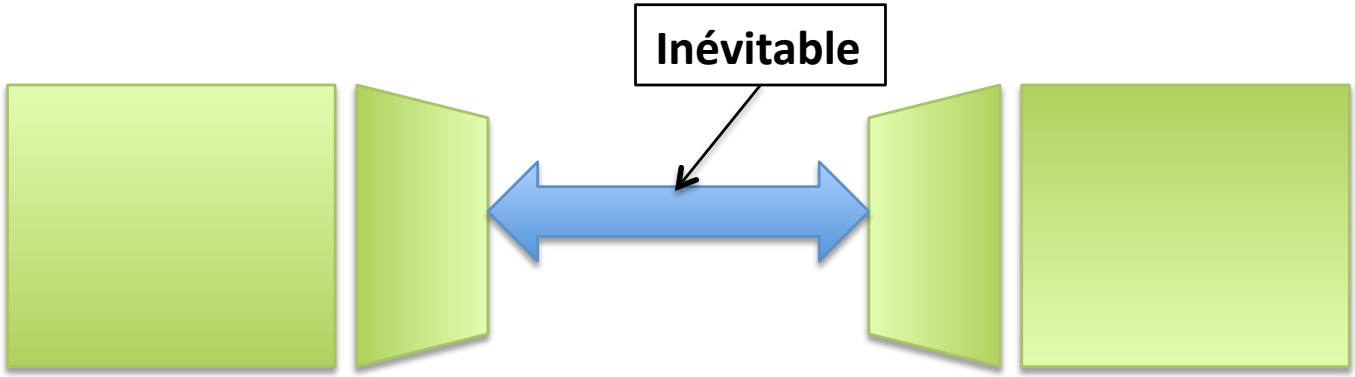
Guillaume Giamarchi

- Architecte technique chez Zenika
- Formateur et expert SOA, ESB et Apache Camel

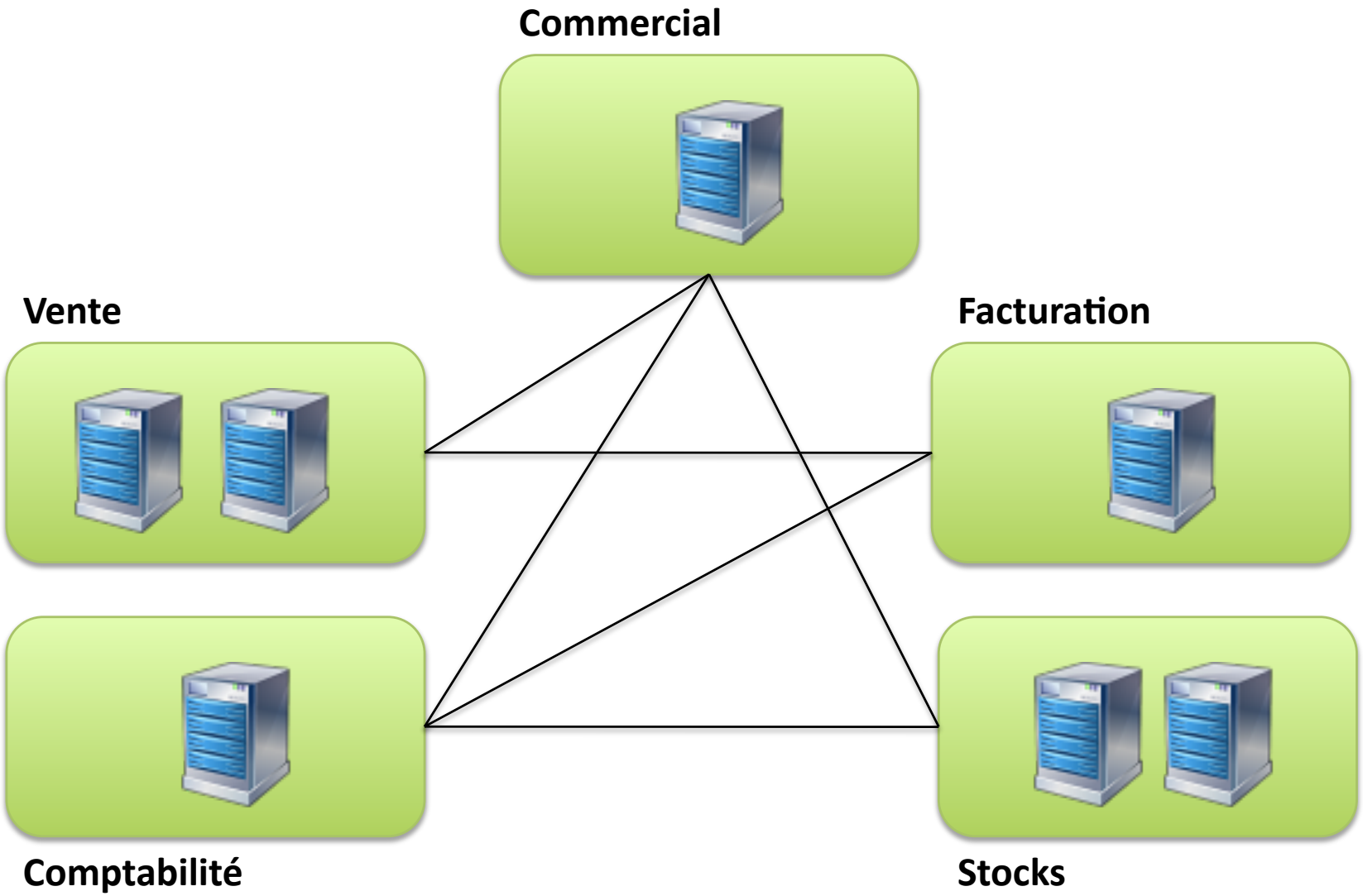
@ggiamarchi

Pourquoi le besoin d'intégration

- Très peu d'applications vivent en isolation
- Les fonctionnalités voulues par un utilisateur résident dans différents systèmes et dans différentes applications

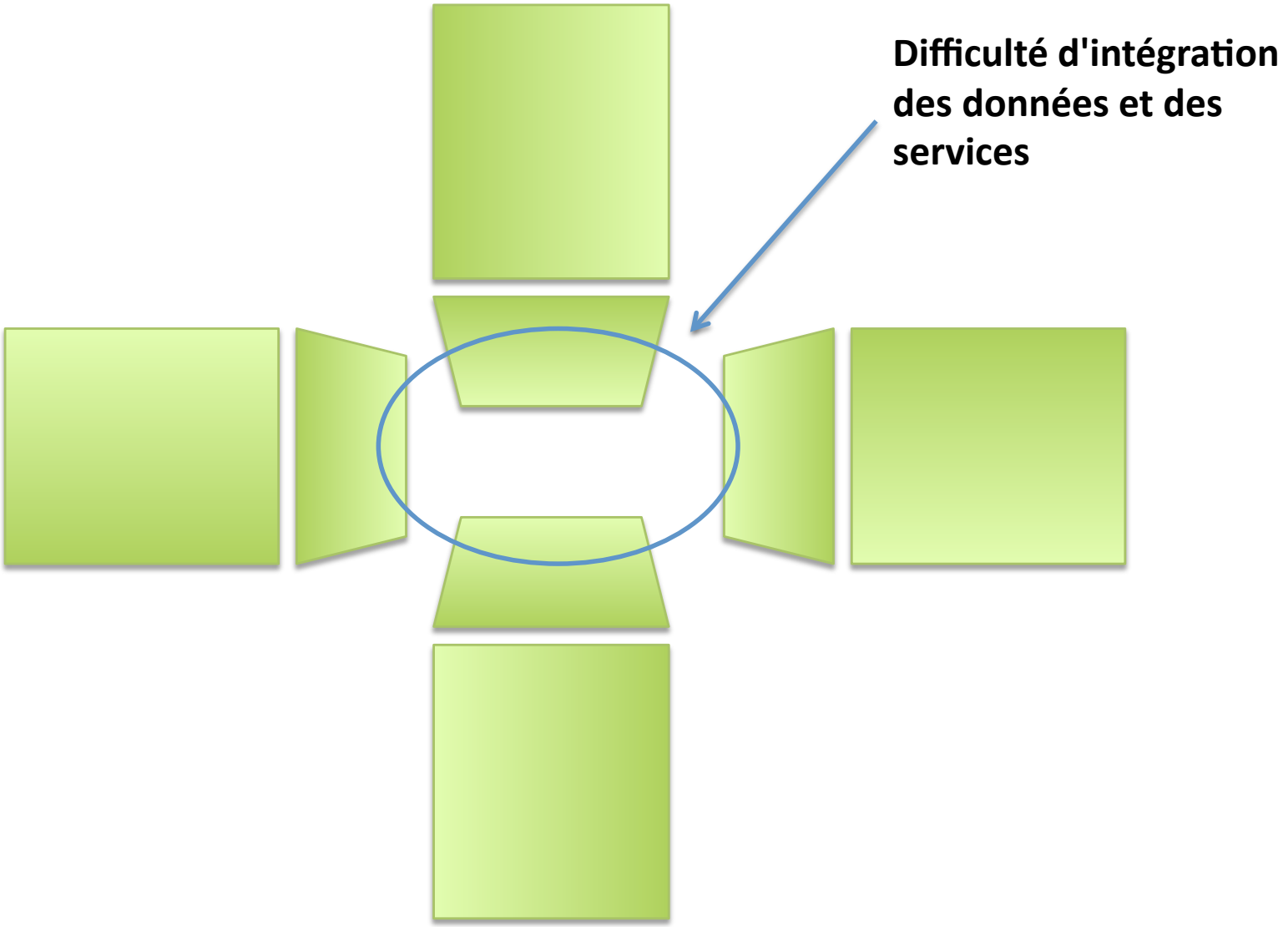


Exemple d'intégration d'un système d'information d'entreprise



Le problème des applications intégrées point à point

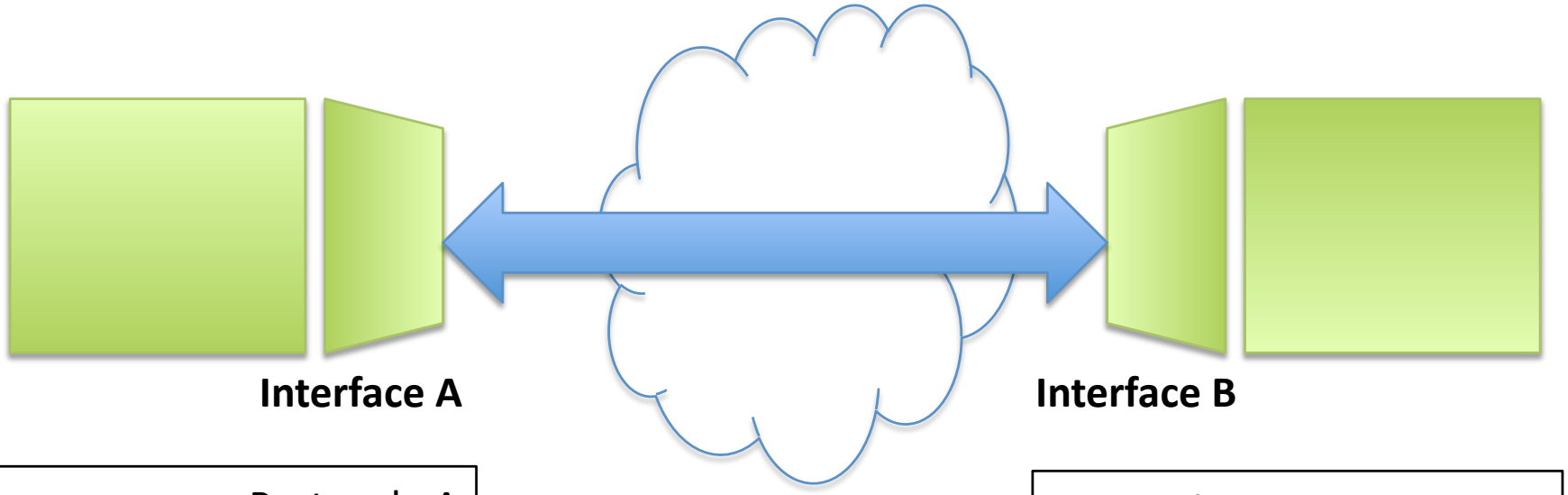
L'intégration avec des systèmes externes est un challenge



Autres challenges d'intégration

- Fiabilité des réseaux
- Lenteur des réseaux
- Changement continu (-- > inévitable)
  - Métier
  - Technique
  - Humain
- Les données échangées ainsi que le nombre d'applications à intégrer augmentent

Objectif d'intégration



Fournir un modèle d'intégration standard et efficace

Les différents styles d'intégration

File Transfer

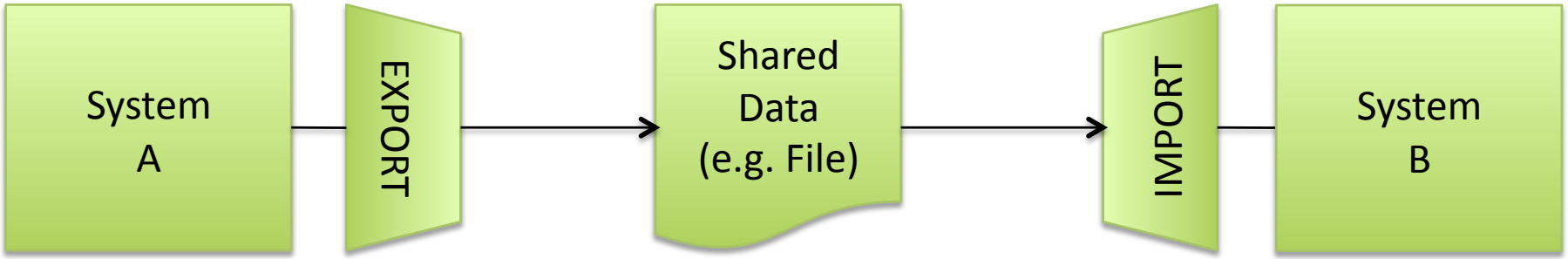
Shared Database

Remote Procedure Call (RPC)

Asynchronous Messaging Style

File Transfer

1970



Le système A exporte ses données dans un format de données commun, le fichier est ensuite lu par le système B

Avantages
- Simple à mettre en place pour des petits volumes
- Découplé physiquement, logiquement et temporellement
- Langages et systèmes indépendant

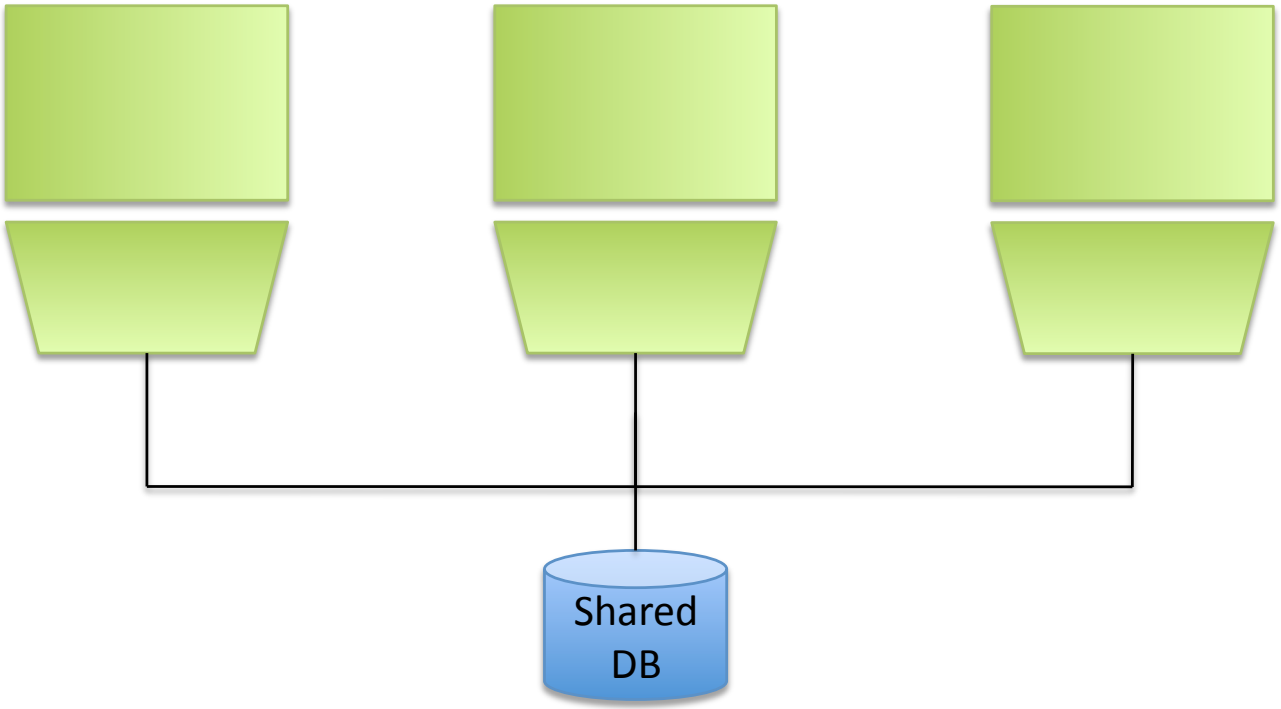
Inconvénients
- Peu fiable
- Temps de latence
- Contrat du nom du fichier, localisation, format des données
- Difficulté d'exploitation

File Transfer

```
File sharedDataFile =
        new File("sharedLocation/sharedData.txt");

//SERVER
FileWriter writer = new FileWriter(sharedDataFile);
writer.write("myData");
writer.flush();
writer.close();

//CLIENT
Scanner in =
        new Scanner(new FileReader(sharedDataFile));
```

Shared Database

1980

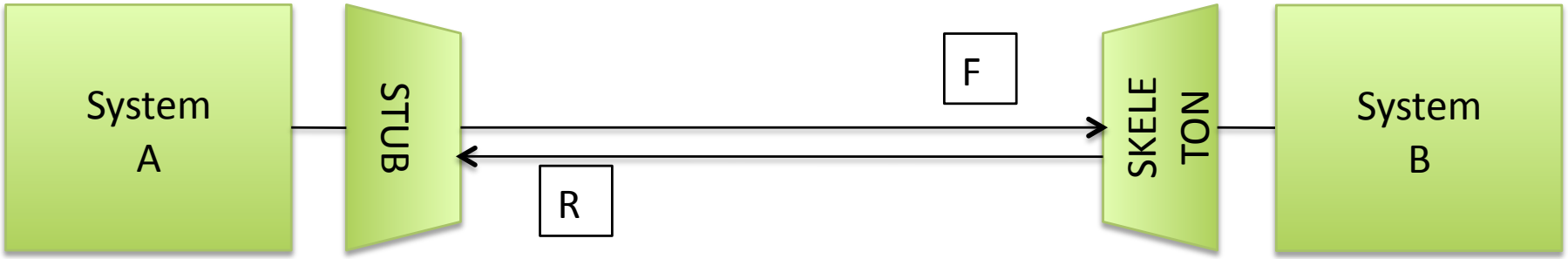


Les données sont centralisées dans un emplacement partagé par toute les applications

Avantages
- Simple
- Données consistantes (transactions)

Inconvénients
- Difficulté à trouver une représentation commune
- Lent

Remoting Procedure Call



Appel des systèmes à la demande pour réaliser une opération

Avantages
- Pratique et Rapide
- Proche du monde "Object Oriented"
- Les données sont échangées uniquement lorsqu'on en a besoin
- Découplé Physiquement

Inconvénients
- Difficilement interopérable
- Couplé logiquement
- Fragile (très couplé logiquement)
- Pas scalable (le thread d'envoi peut rester bloqué)

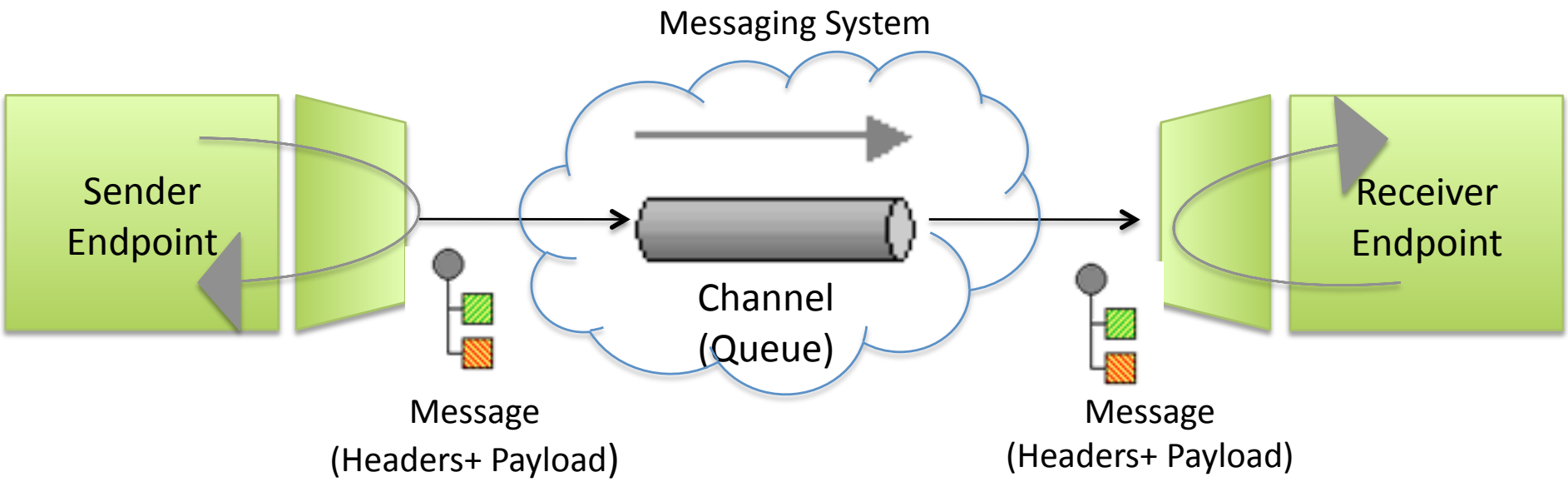
Remoting Procedure Call
Exemple de code

```
//Client side code
Registry registry =
     LocateRegistry.getRegistry(orderRegistry);

OrderService orderService =
   (OrderService)registry.lookup(orderRegistryName);

 SubmitOrderResult result =
     orderService.submitOrder(order);
```

Remarque

Spring Remoting facilite l'exposition et la consommation de services à travers différents protocoles

Asynchronous Messaging Style



- **Les systèmes envoient des données à travers des channels (ou event queue)**
- **L'envoie d'un message dans un channel est rapide ("fire-and-forget")**
- **Le channel assure le découplage des systèmes (Le channel garde les requêtes jusqu'à ce que le consommateur est prêt)**

Asynchronous Messaging Style
Exemple de code JMS

```
//Producer JMS
producer = session.createProducer(queueMessage)
producer.send(message);
otherBusiness()
```

```
//Consumer JMS
consumer = session.createConsumer(queueMessage)
Message message  = consumer.receive(3000)
```

Remarque
L'envoie et la réception de messages peuvent être facilité avec Spring JMS

Un écosystème de termes

Asynchronous Messaging

Fire-and-forget information exchange

Message Oriented Middleware (MOM) dans le cas d'un broker

L'apport d'un style de messaging asynchrone

- Communication Asynchrone
- Découplé logiquement
- Réglage de la cadence de traitement
- Fiabilité
- Logique d'intercepteurs

Attention au développement d'application asynchrone

Pas les mêmes paradigmes
que les application synchrones

- Contexte transactionnel
- Contexte de sécurité
- Gestion des erreurs

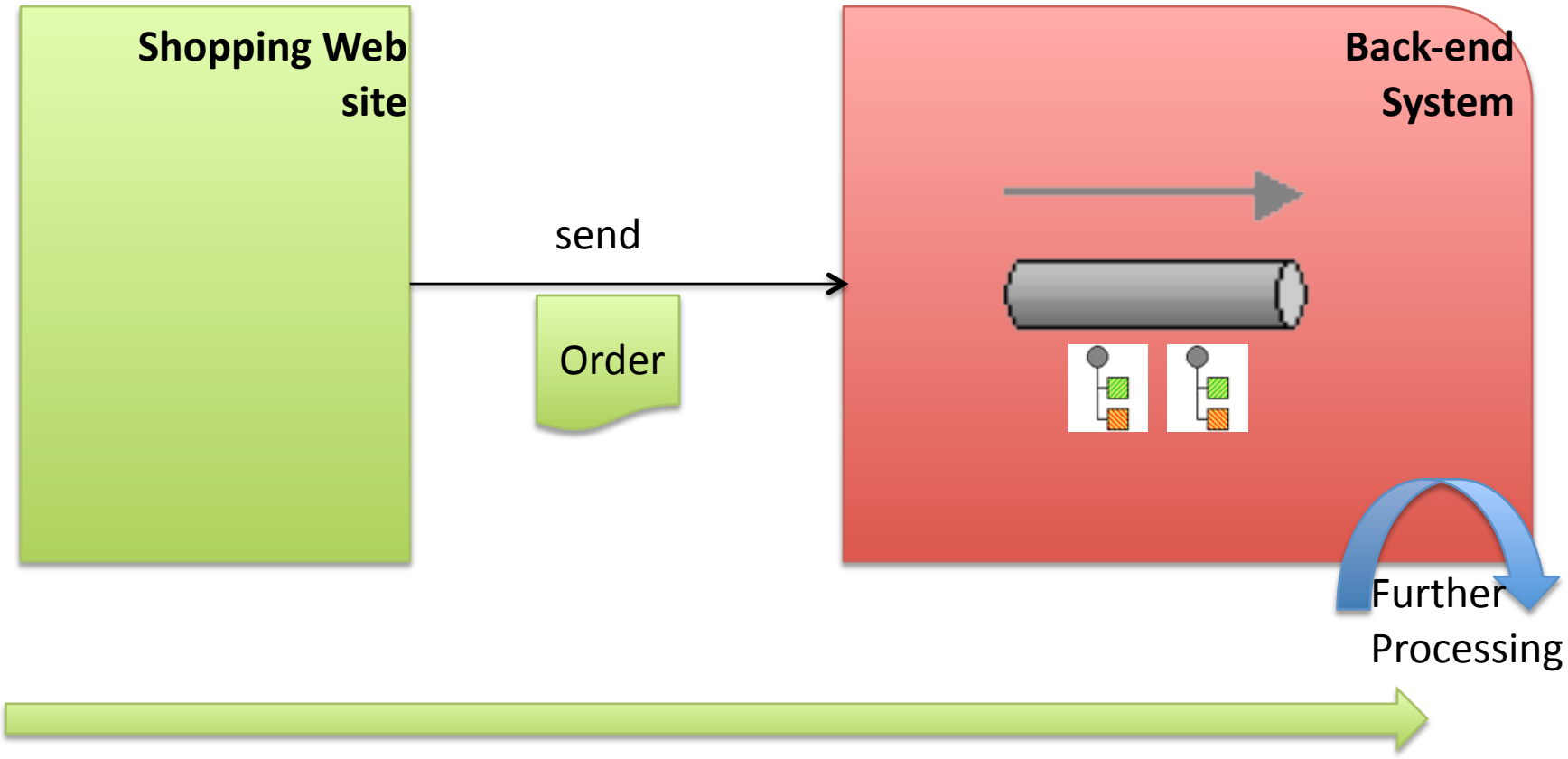
Exemples de cas d'utilisation du messaging

Simple producer – consumer

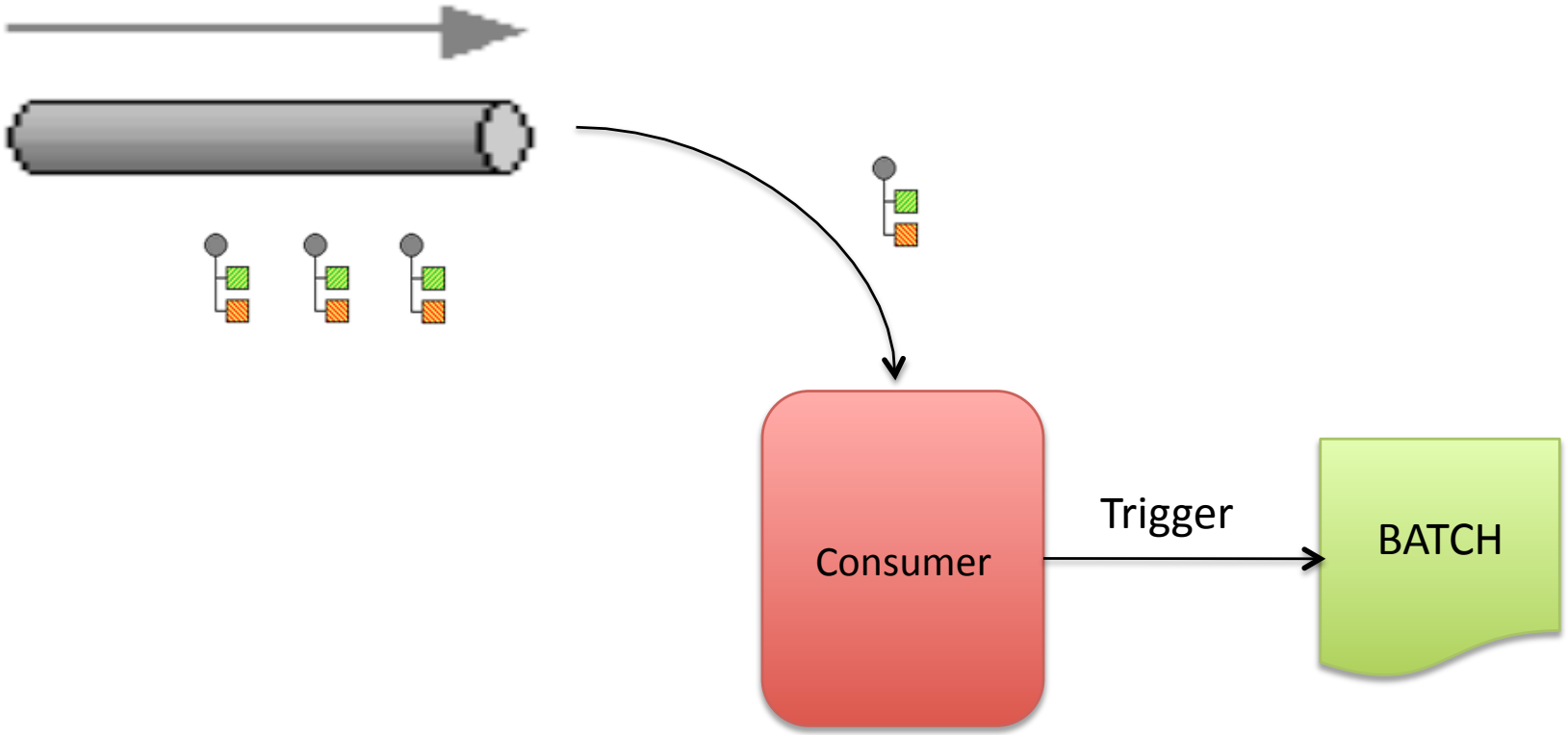
Pipelining

Message Distribution

Simple Producer – Consumer (1/3)

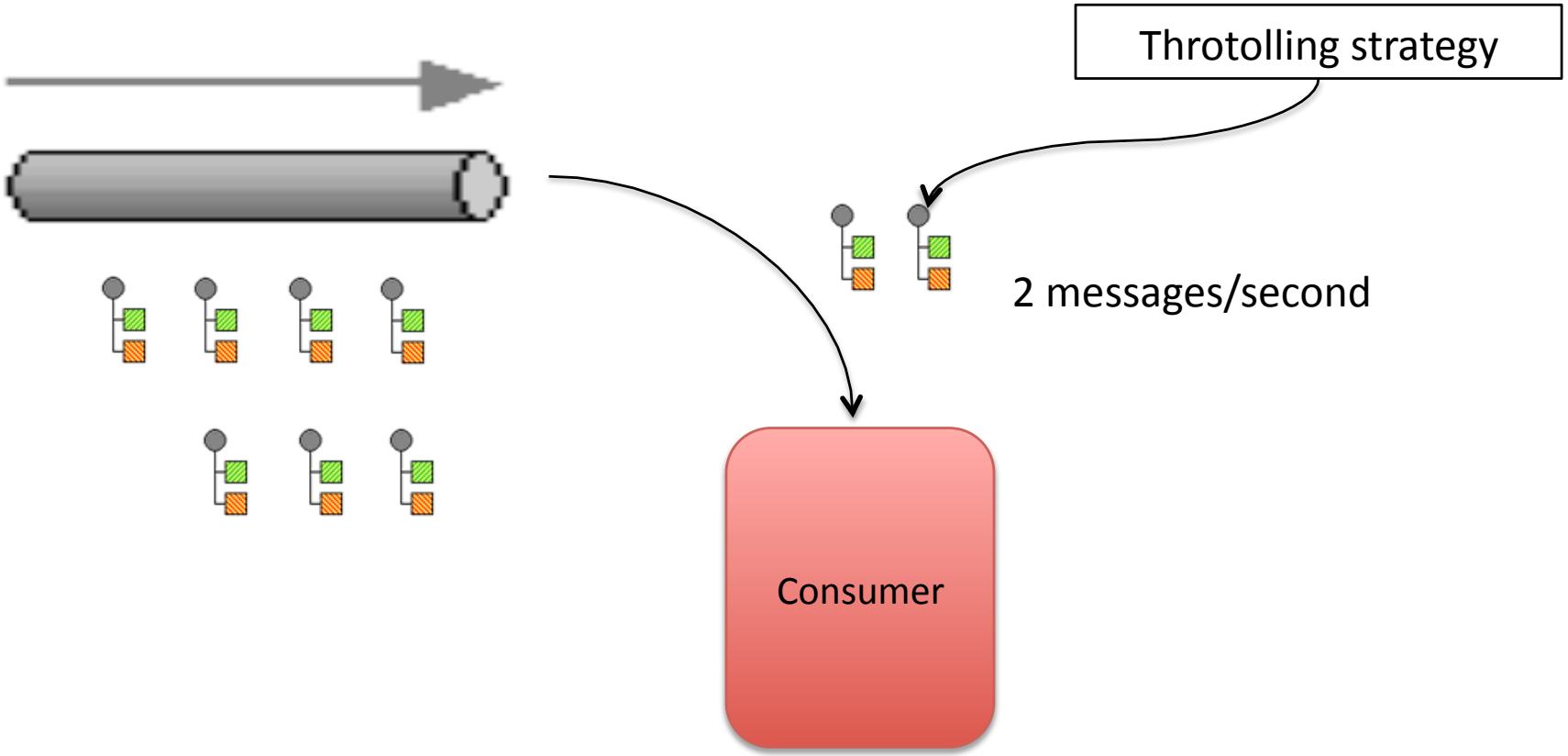


Simple Producer – Consumer (2/3)



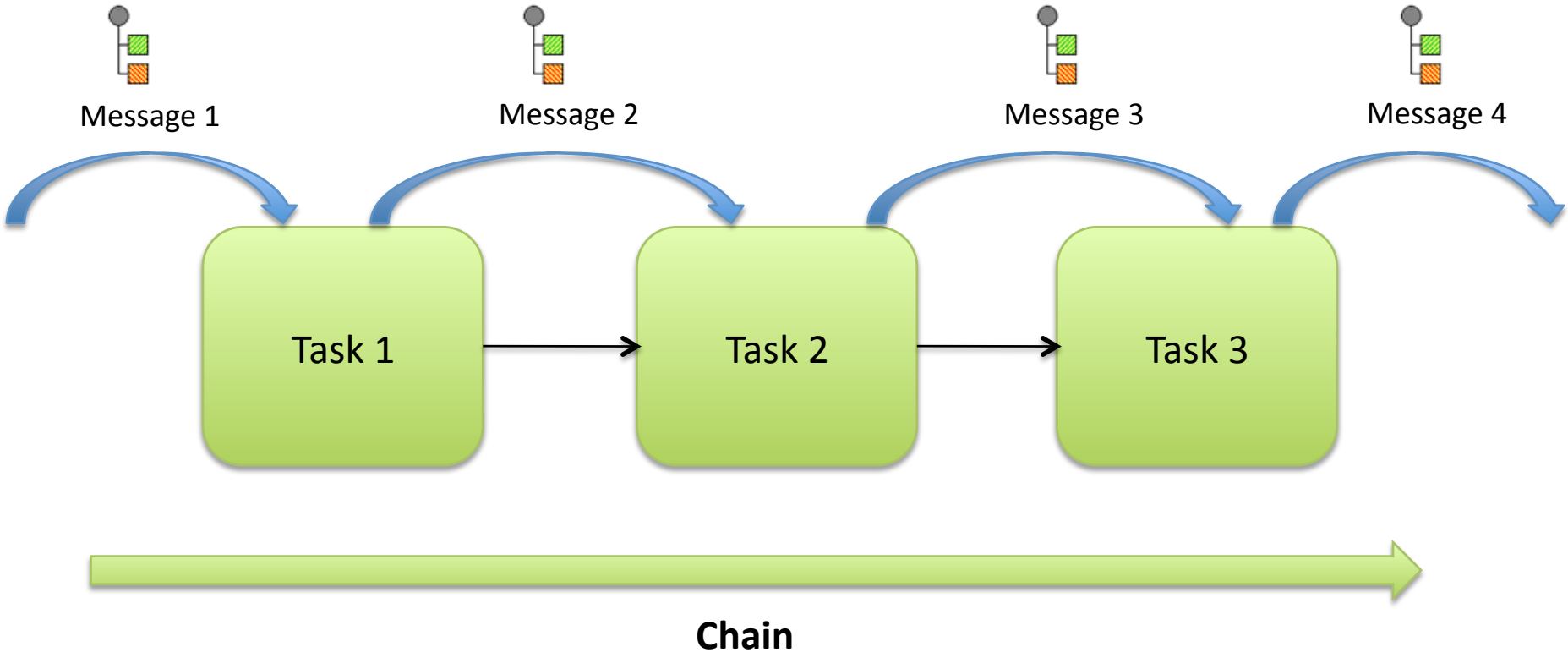
Chaque message consommé déclenche un batch

Simple Producer – Consumer (3/3)



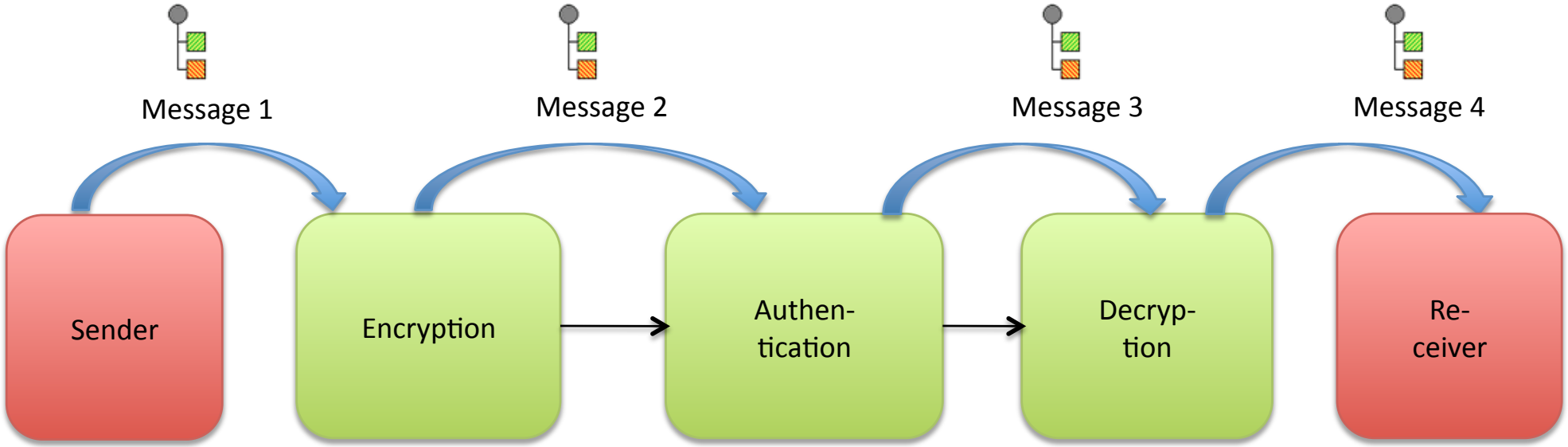
Le consommateur n'est jamais surchargé

Pipelining

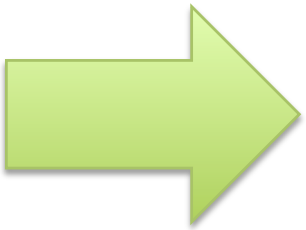


On envoie le message à travers de multiples systèmes

Pipelining à la sécurité

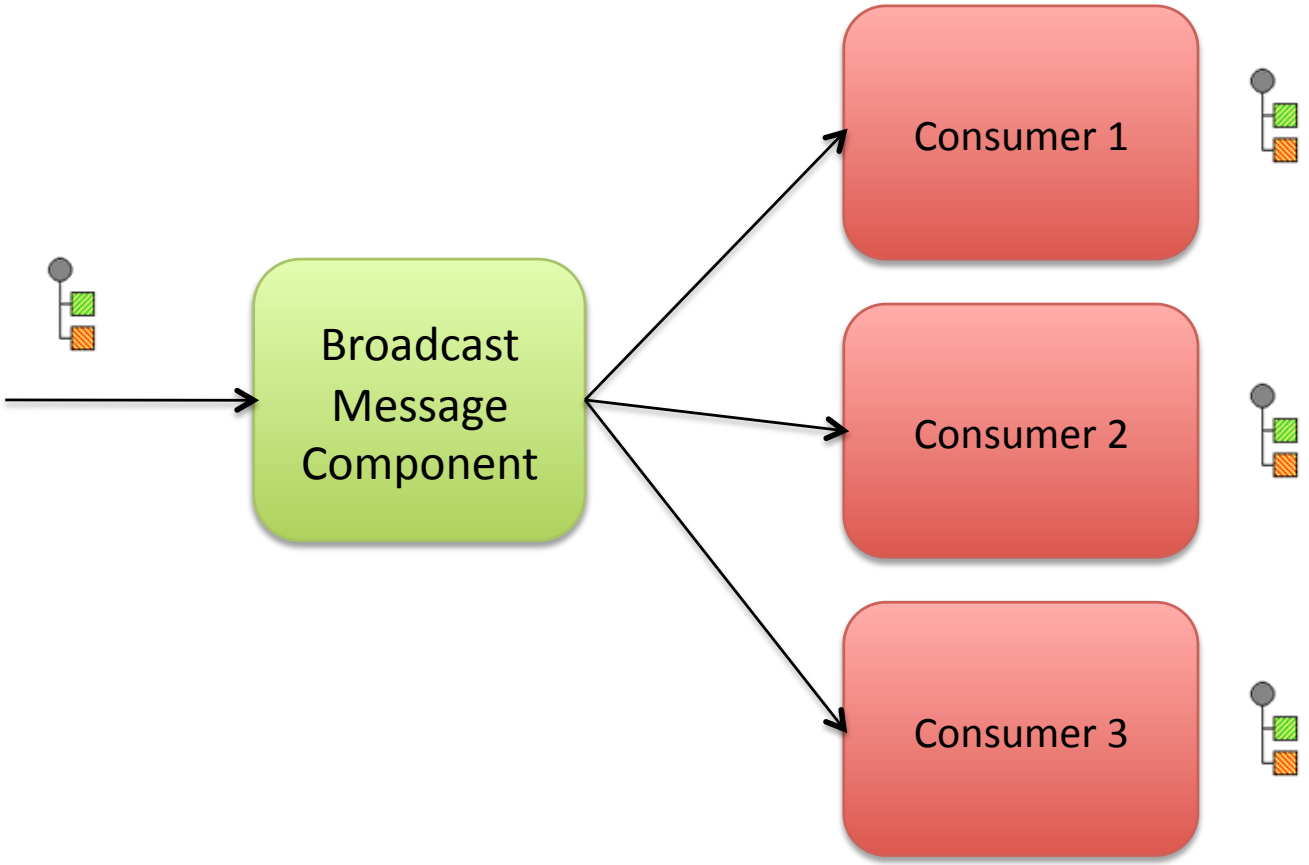


IMPLEMENTATION
MONOLITHIQUE



IMPLEMENTATION
A BASE DE MESSAGES

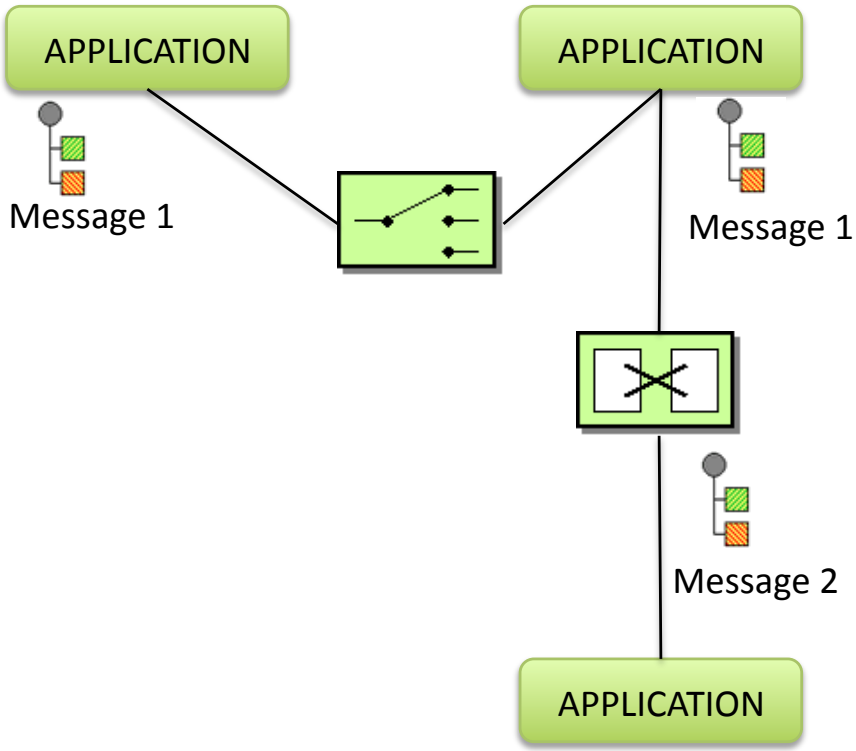
Message Distribution



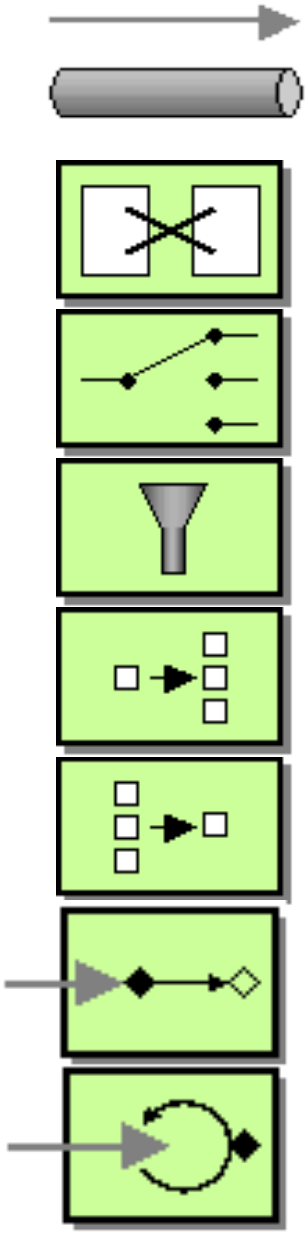
Le message est dupliqué.
Chaque consommateur reçoit une copie du message

Le besoin d'un "Messaging Pattern Language"

1. Transporter les messages **(Channel Patterns)**
2. Designer les Messages **(Message Patterns)**
3. Transfert des messages vers sa destination (**Router Patterns**)
4. Transformer les messages dans le bon format (**Transformation Patterns**)
5. Envoyer et réceptionner des messages **(Endpoint patterns)**
6. Gérer et Tester le système **(Management Patterns)**



Entreprise Integration Patterns (EIP)



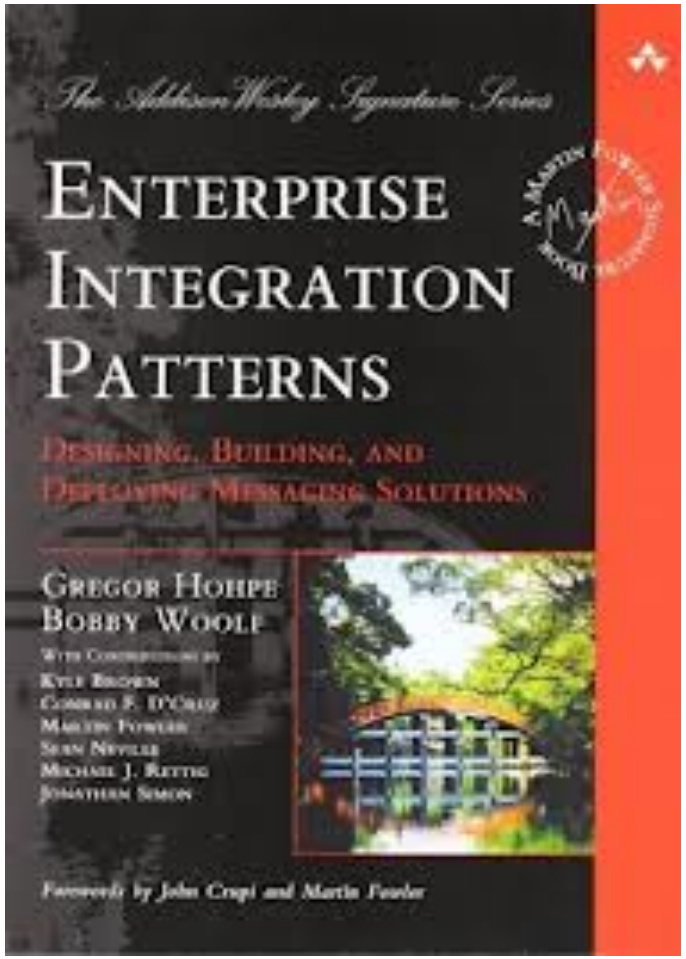
- Channel
- Transformer
- Router
- Filter
- Splitter
- Aggregator
- ServiceActivator
- Poller
- etc

- Des patterns pour l'intégration
- Facilite la communication entre les développeurs
  - Aidé par des diagrammes visuels
- Oriente une décision, un choix d'architecture

Entreprise Integration Patterns Book

Gregor Hohpe

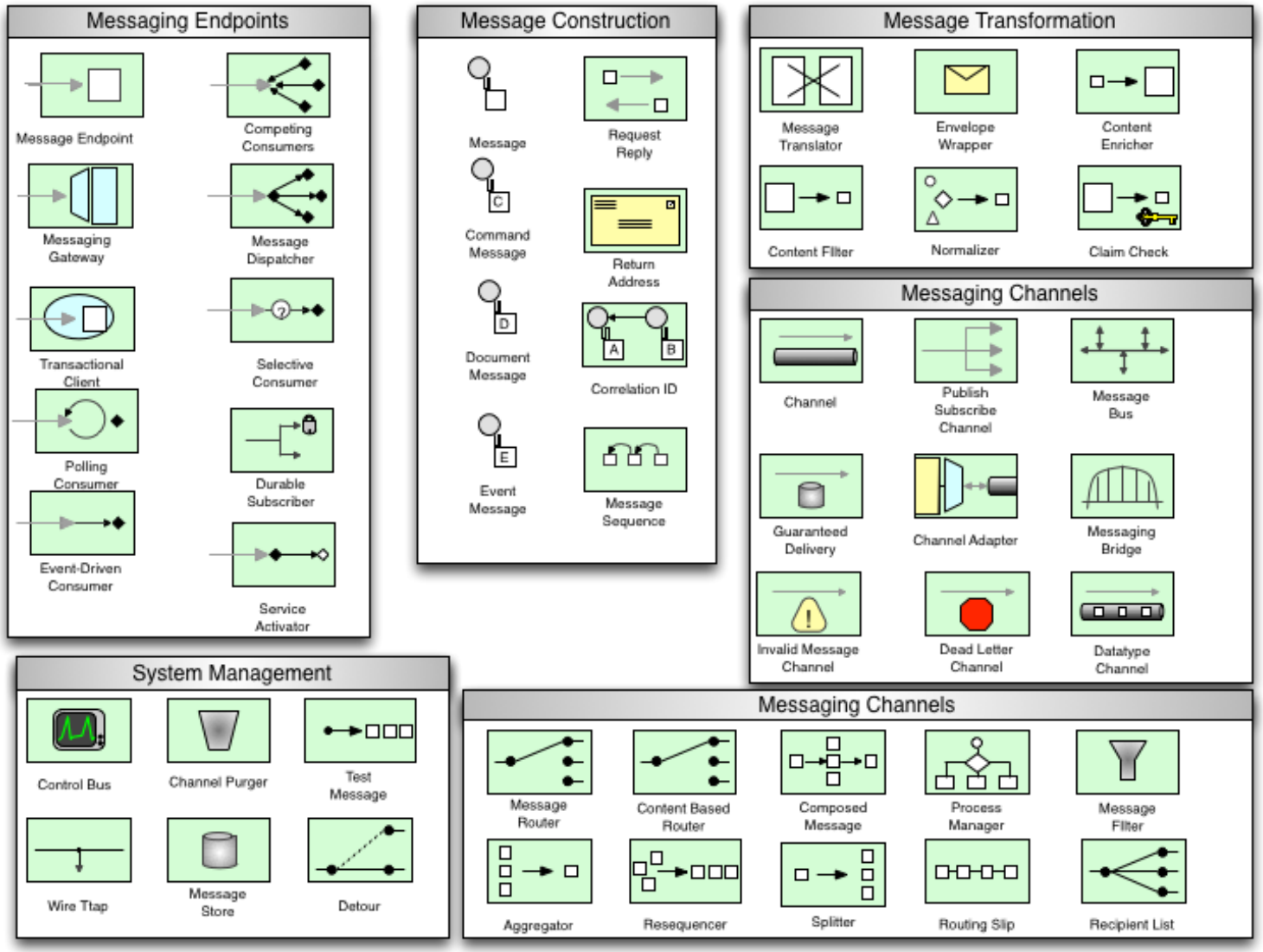
Bobby Woolf



- Un langage visuel (diagrammes et icônes)

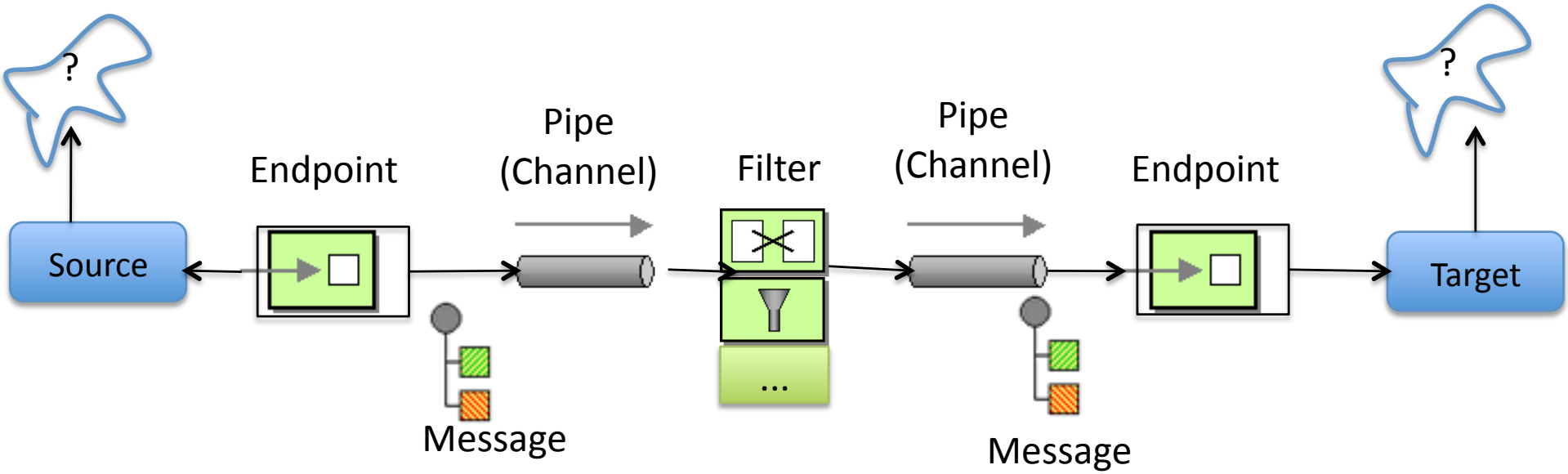
- A servi de fondation pour les différentes technologies de l'industrie open source d'intégration
  - Adoption d'un langage commun

Un catalogue complet de patterns

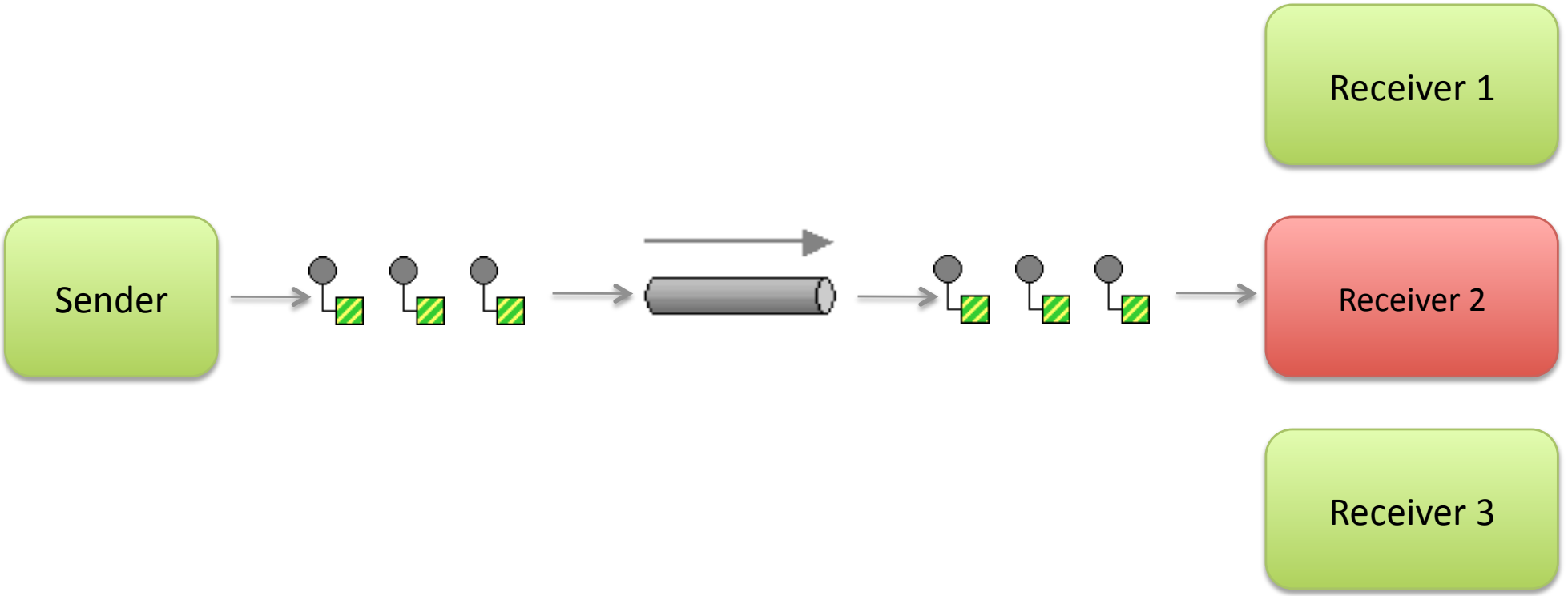


65 patterns

"Pipe and Filter Architecture" en EIP

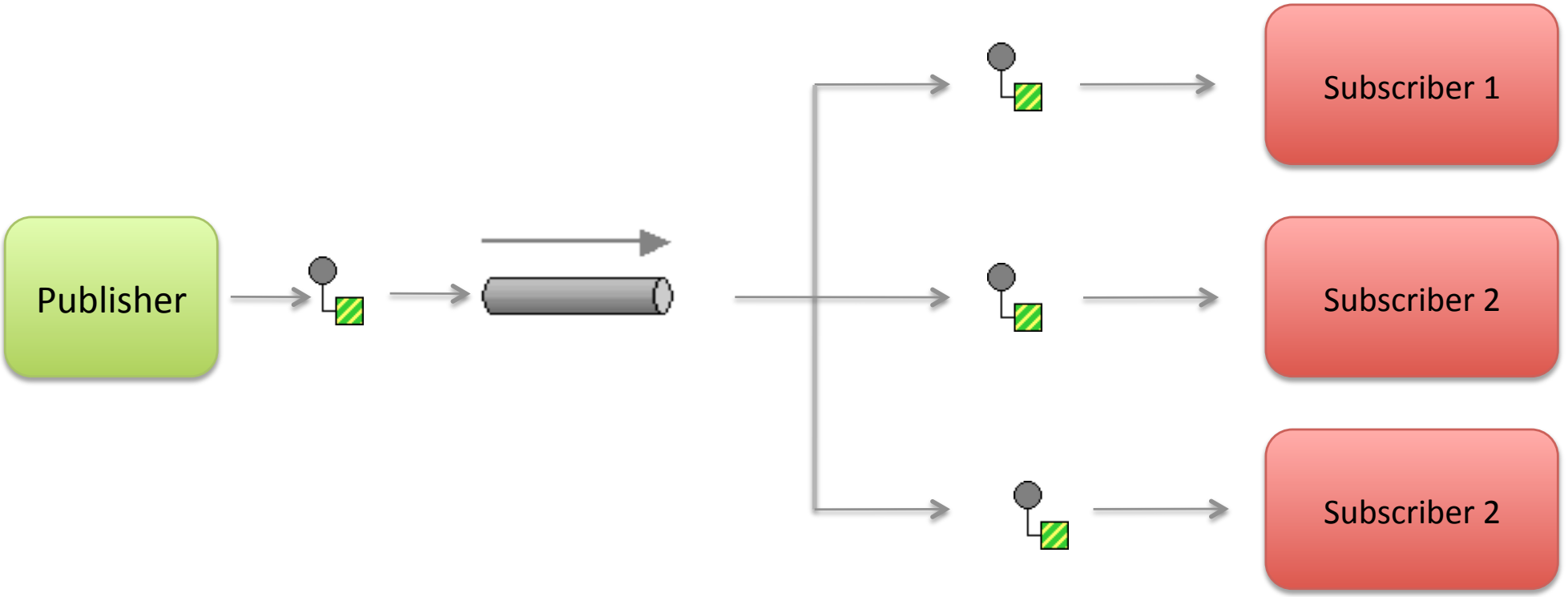


Point-to-point (P2P) Channel



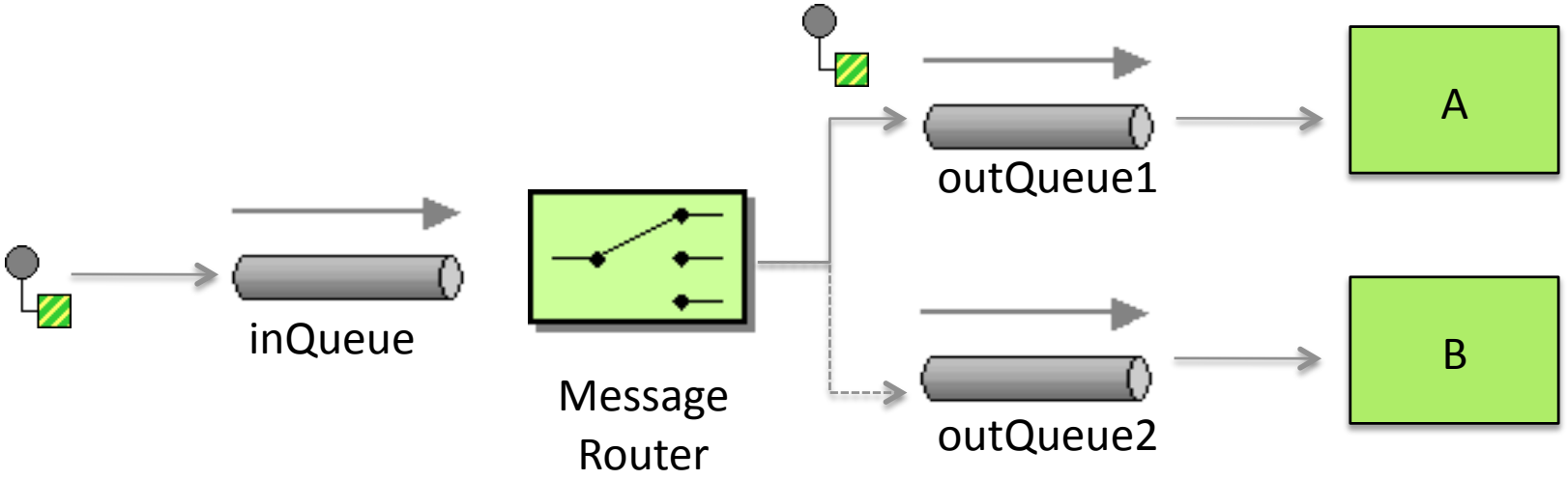
Un Point-to-point Channel garantit
qu'un seul consommateur consomme le message

Publish-subscribe Channel

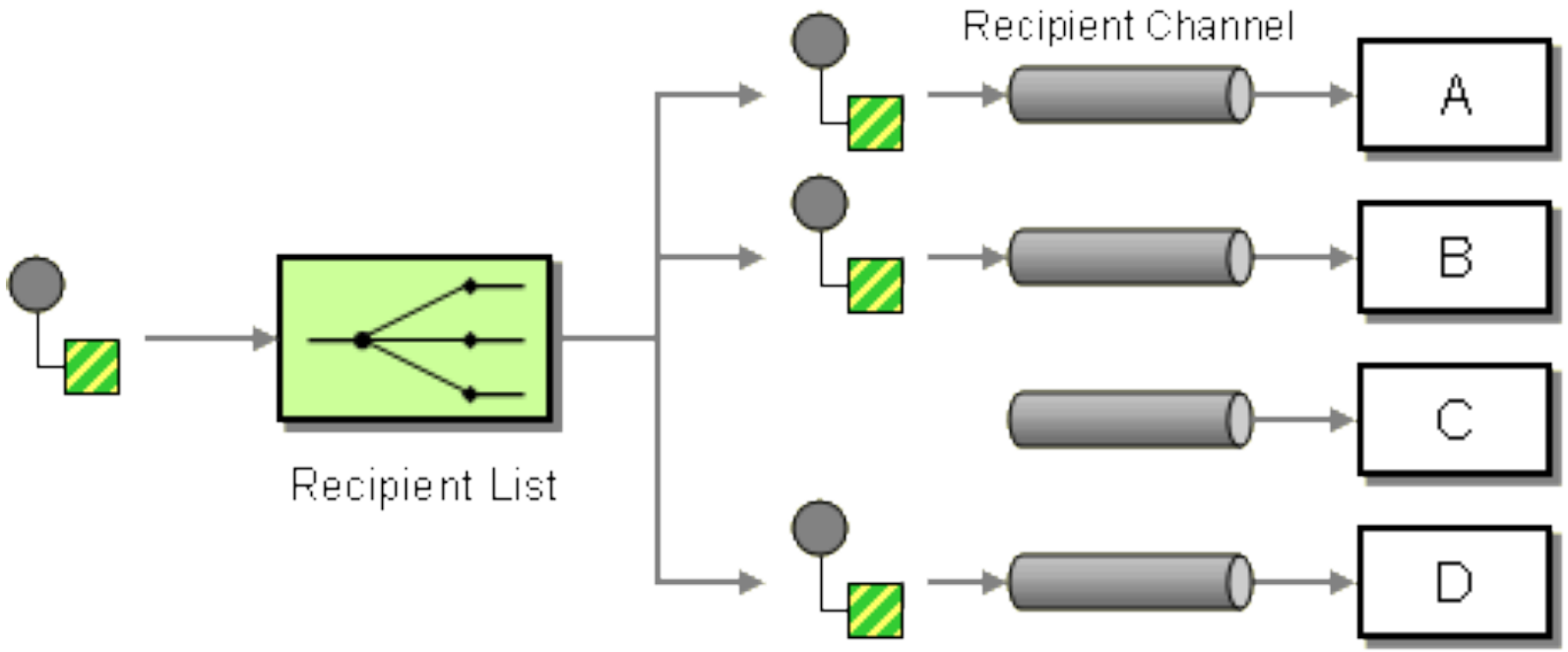


Un Publish-subscribe channel garantit que tous les consommateurs enregistrés reçoivent une copie du message

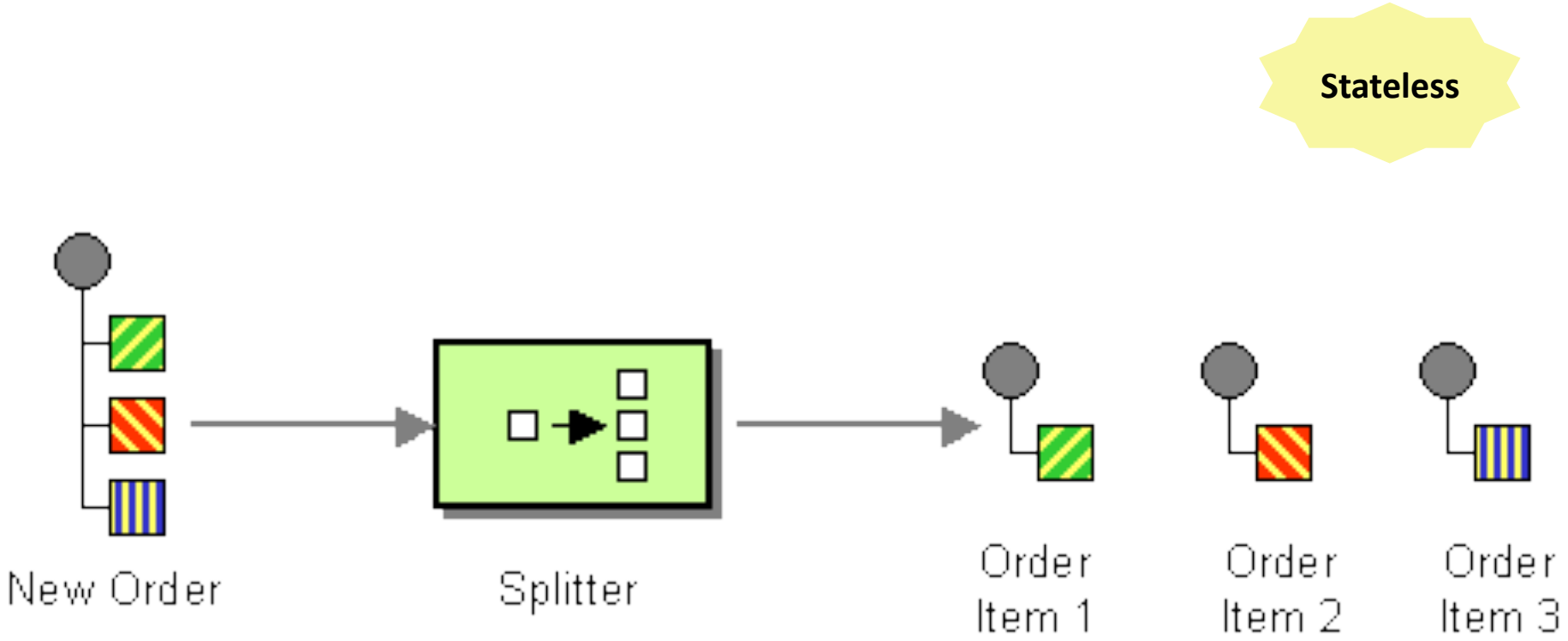
Message Router



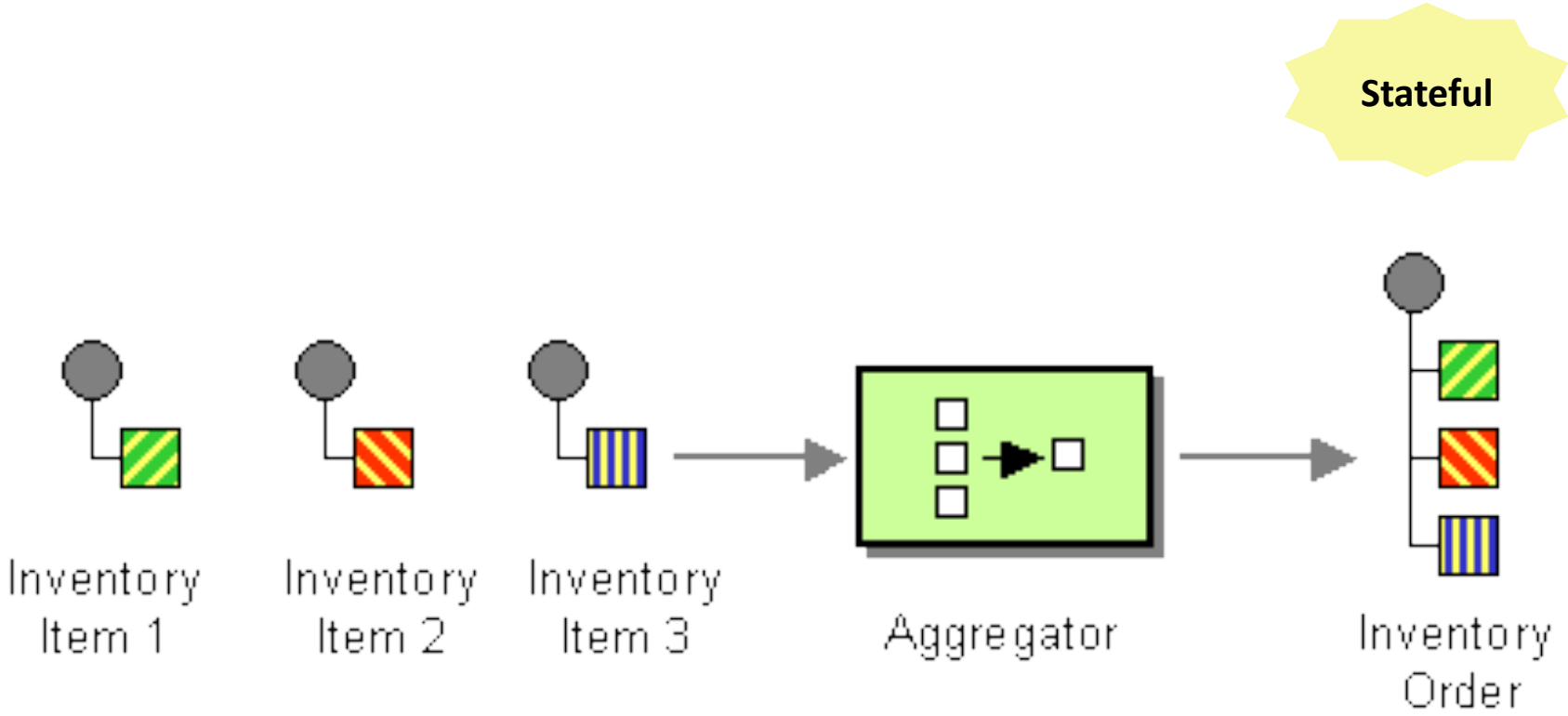
Recipient List



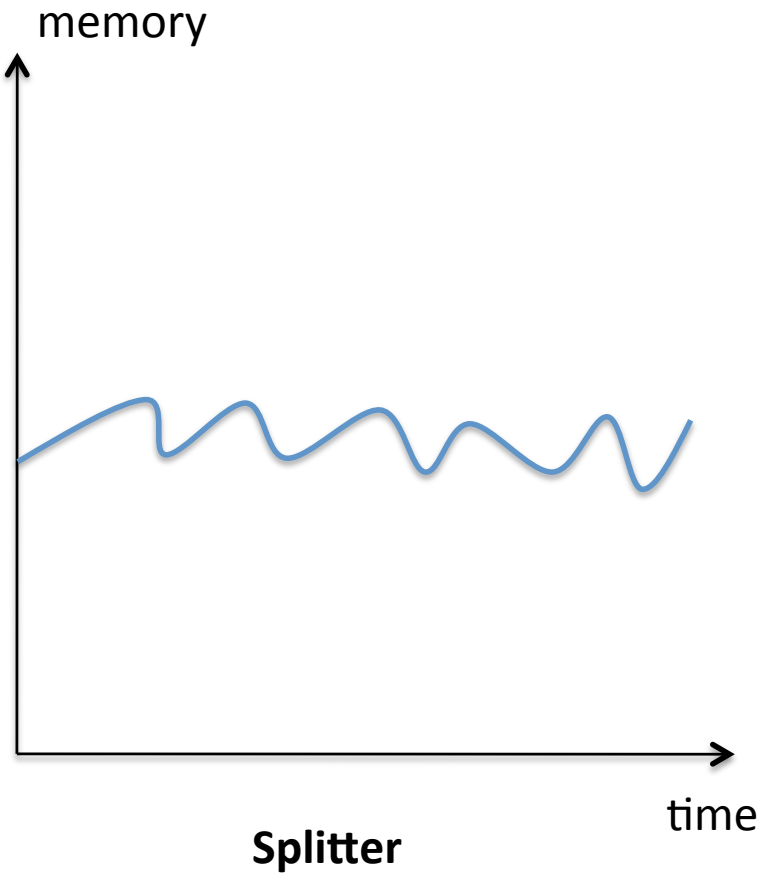
Splitter


Stateless
New Order
Splitter
Order Item 1
Order Item 2
Order Item 3

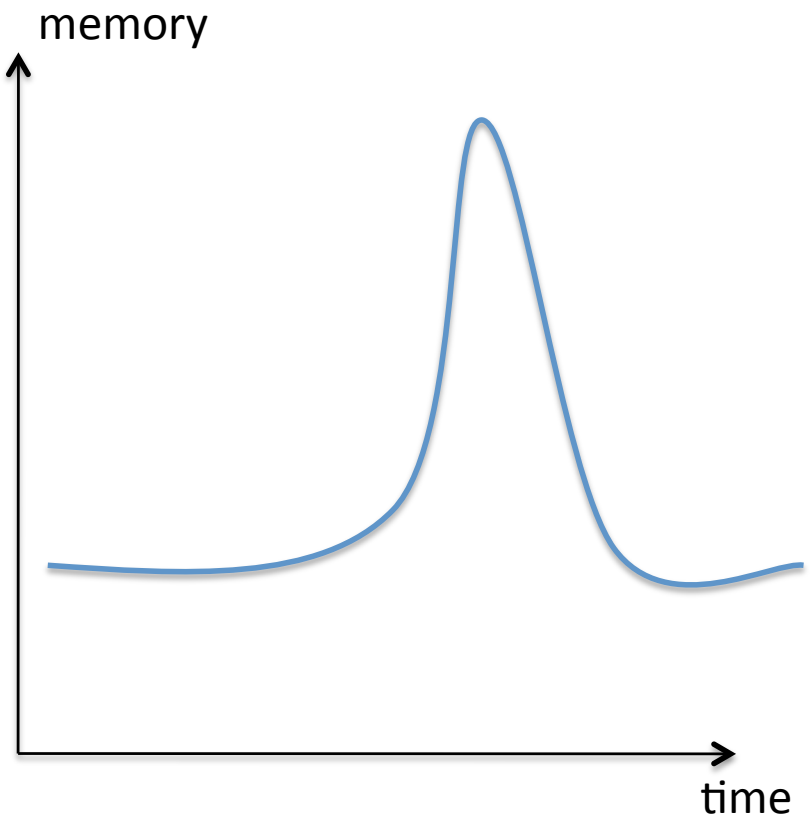
Aggregator


Stateful
Inventory Item 1
Inventory Item 2
Inventory Item 3
Aggregator
Inventory Order

Memory Heap
Splitter vs Aggregator

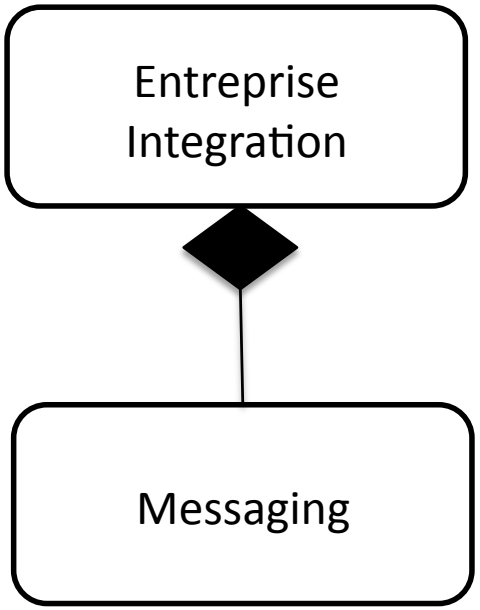


Splitter



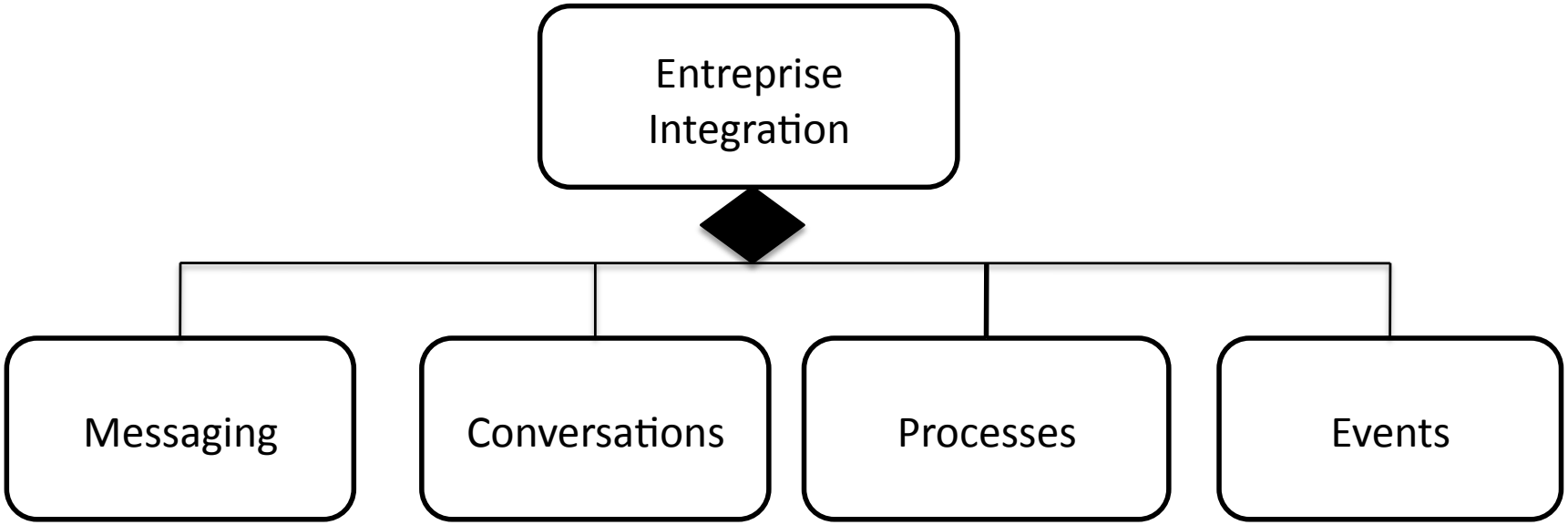
In-memory Aggregator

Est-ce que le messaging est l'unique partie de l'intégration d'entreprise?



- Un système de messaging
  - Faiblement couplé
  - Composable
  - Élégant
  - Scalable
- Mais
  - Plusieurs interactions?
  - Un flow à travers différents nœuds?
  - Transactions distribués (2PC)?
  - Gestion des erreurs?

Les autres parties de l'intégration



Les patterns de conversation (1/2)

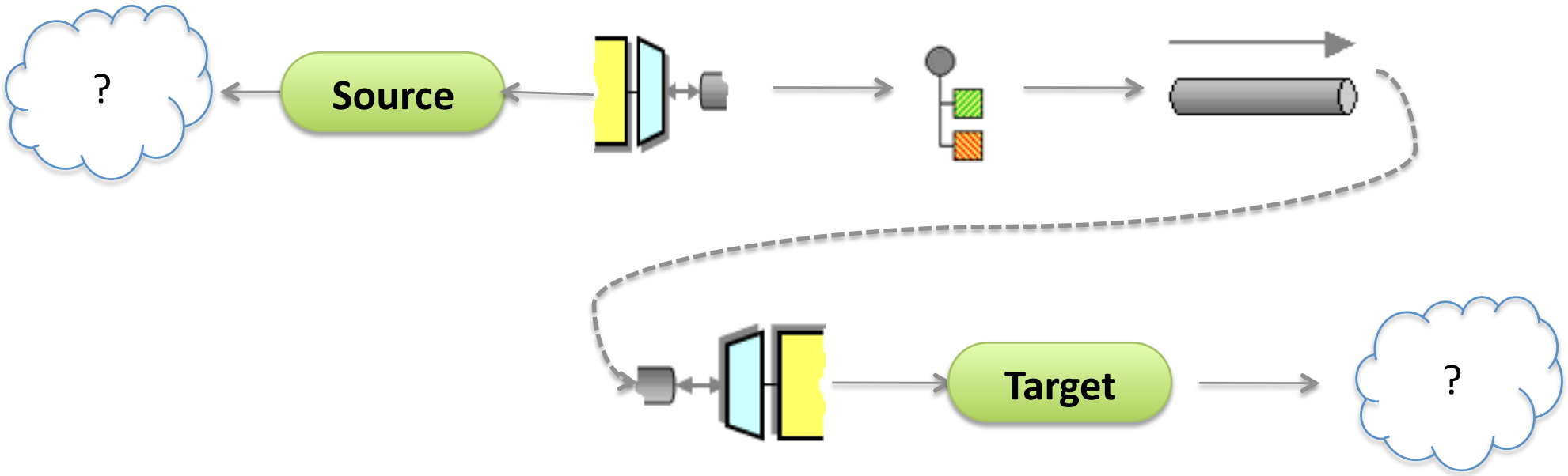
- **Messages**
  - Instantiating Message
  - Follow-on Message
  - Complete Message
  - Side Conversation (Sub conversation)
  - Acknowledge Message
- **Simple Conversation**
  - Reliable Delivery
  - Sync Request-reply
  - Async Request-reply message
  - Async Request-Poll for result
  - Subscribe-Notify
  - Tacit Agreement
  - Reaching Agreement

Les patterns de conversation (2/2)

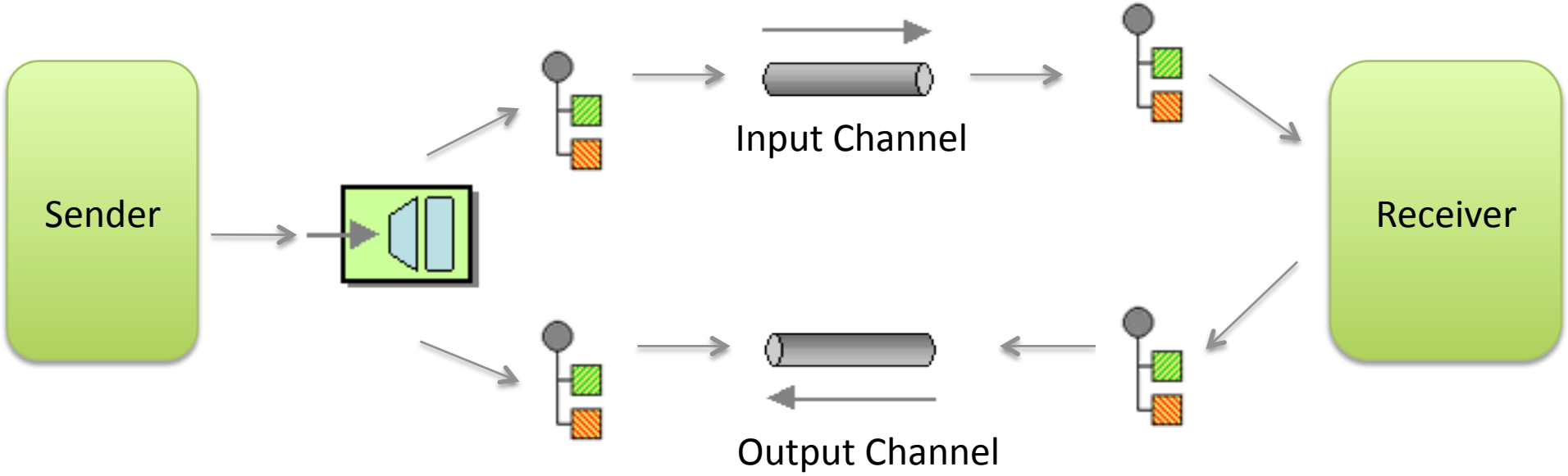
- **Coordinates Conv.**
  - Vote/Poll
  - Reaching Agreement/ Two phase vote
  - Unanimous aggreement
- **Etablishing Conv.**
  - Discovery
  - Introduction
  - Three-Way Handashable
  - Role negocation
  - Establishing trust
- **Renewing Interest**
  - Lease/Automatic
  - Expiration
  - Renewal Reminder
- **Exception Handling**
  - Two Phase Commit
  - Compensation Action
  - Retry / Resend (Idempotent receive)
  - Write-Offs

Channel Adapter

- Du code afin de cacher la complexité d'infrastructure entre une source ou une destination et le système de messaging



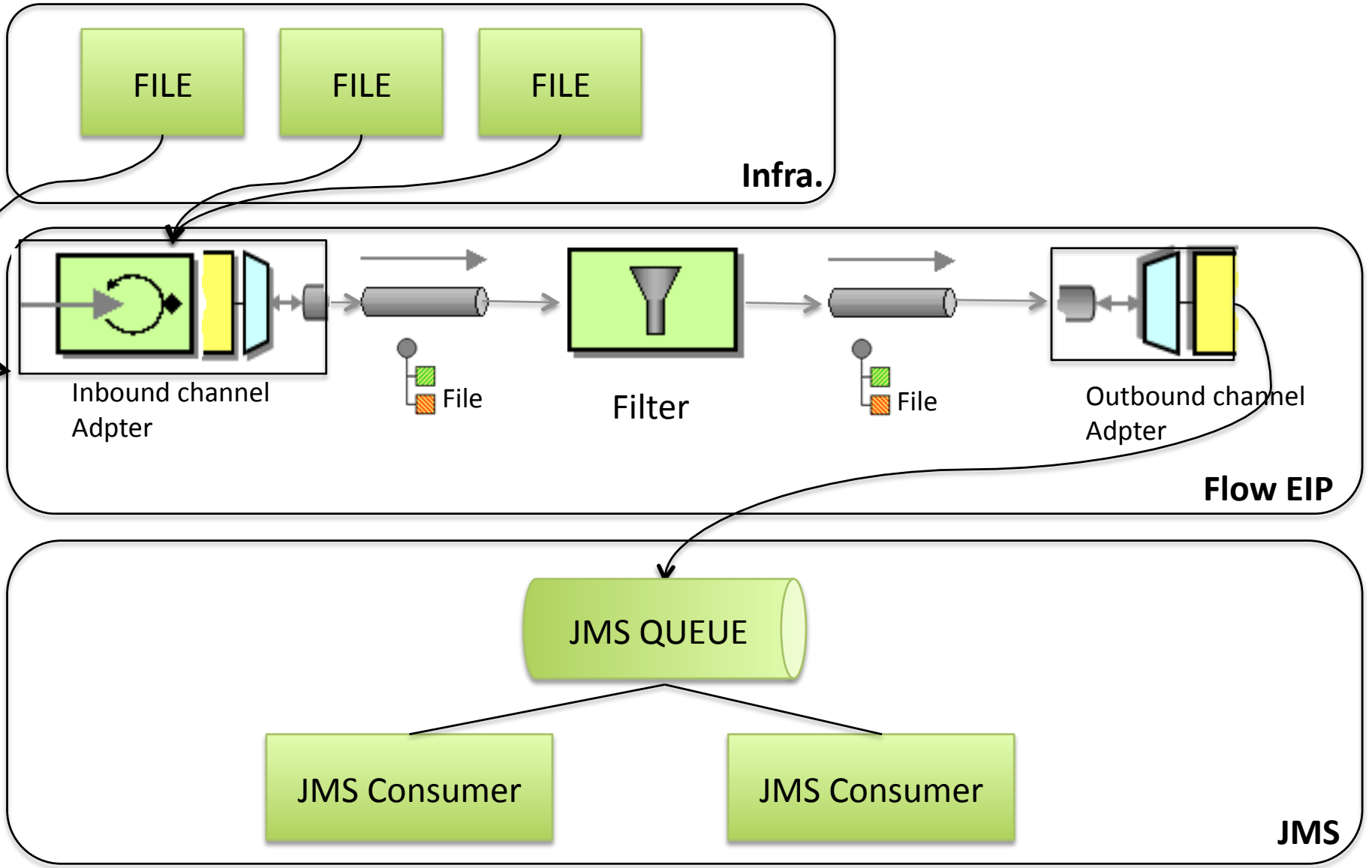
Messaging Gateway



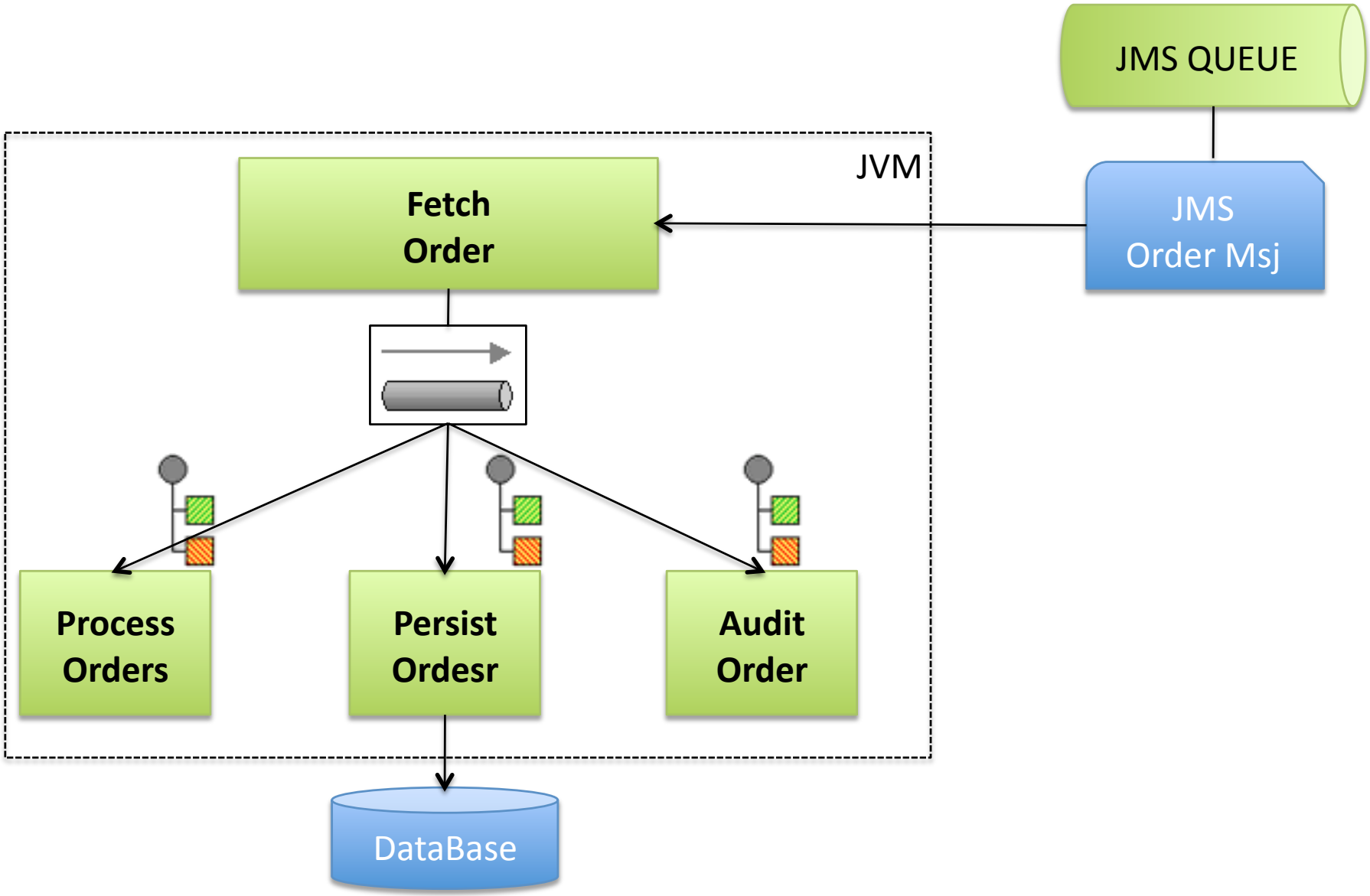
Façade d'envoie de messages
Synchrone ou Asynchrone

Quelques cas d'utilisation

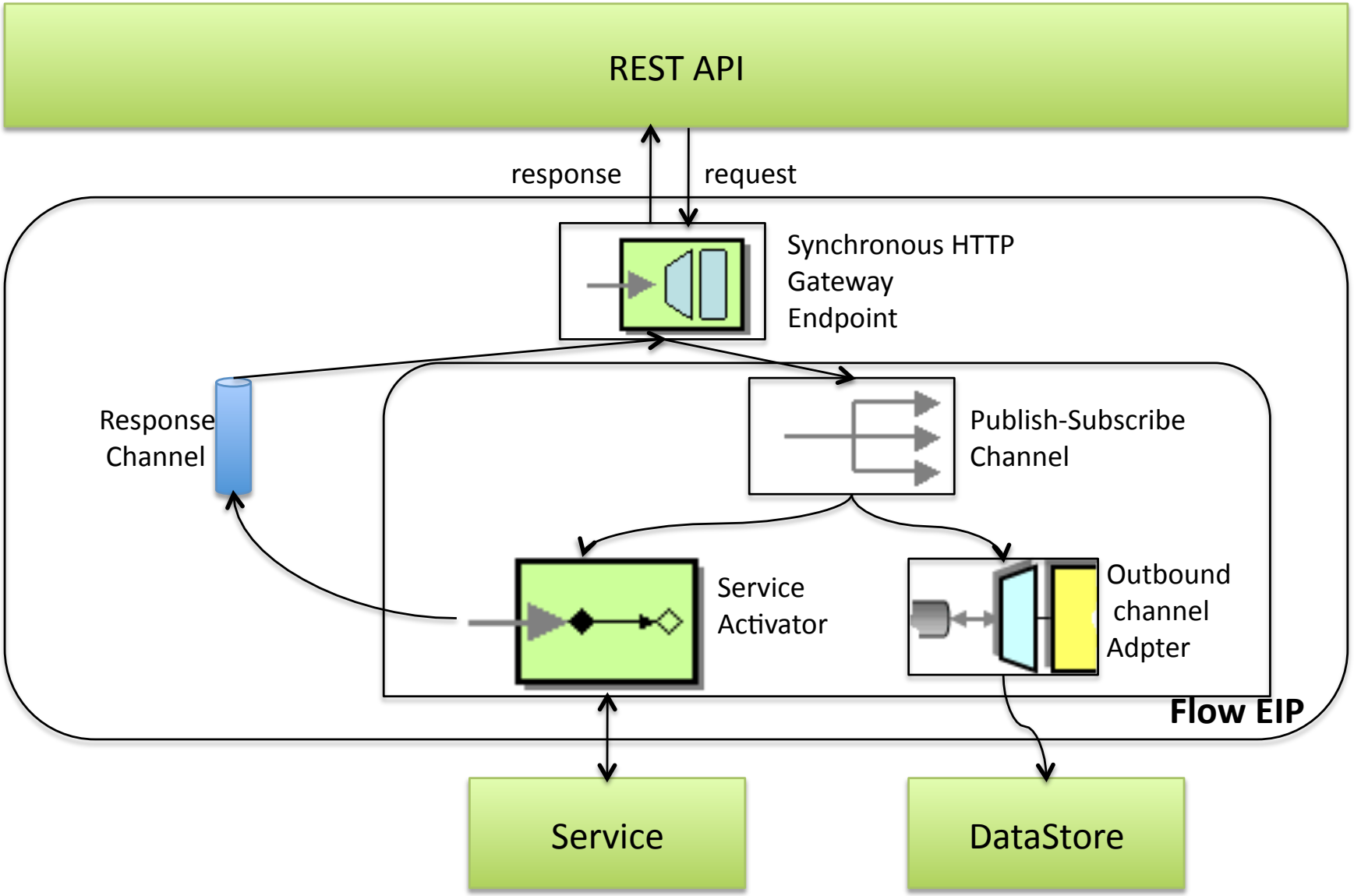
Un pont entre un environnement à base de fichiers et JMS



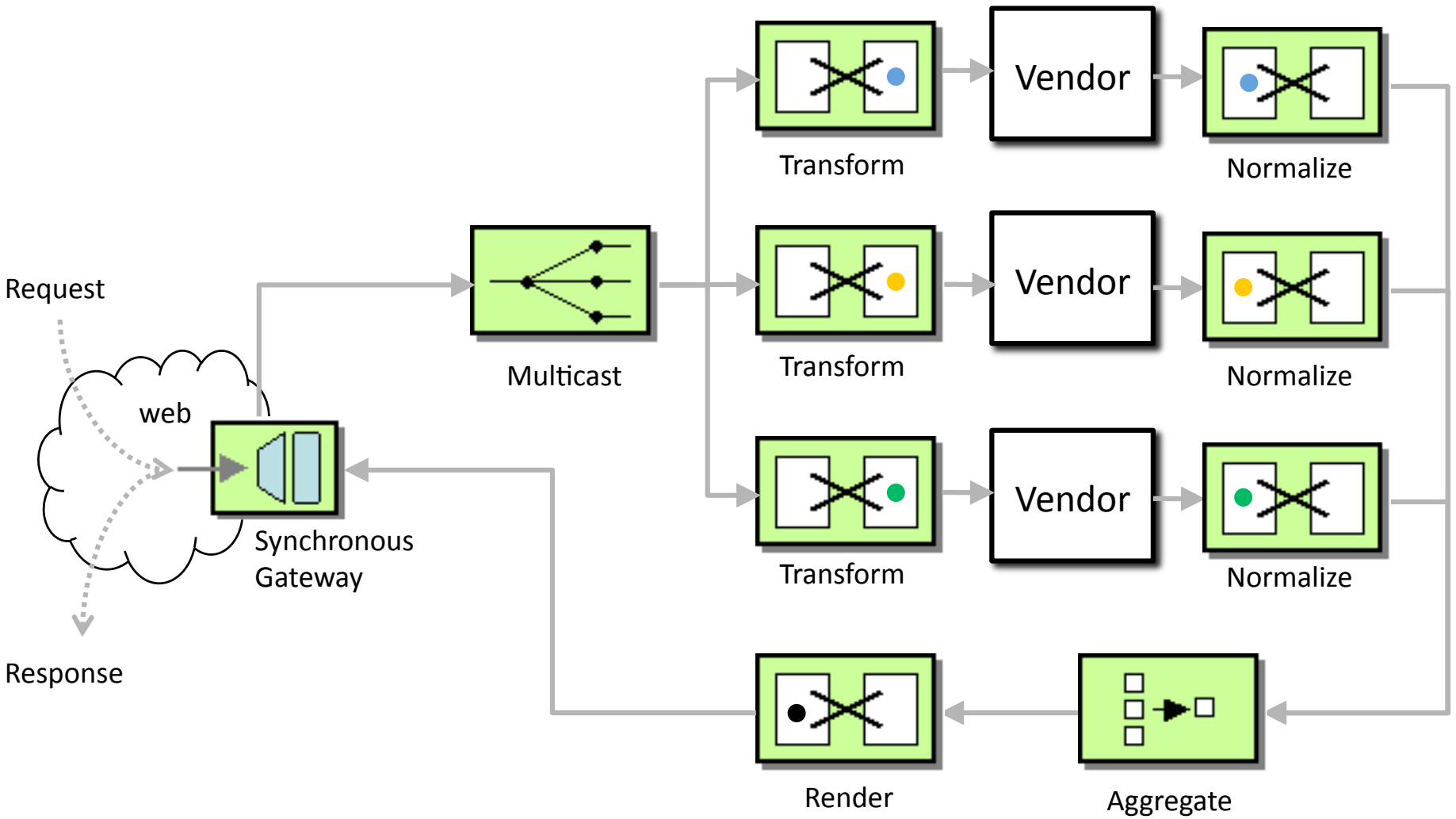
Distribution d'un traitement JMS



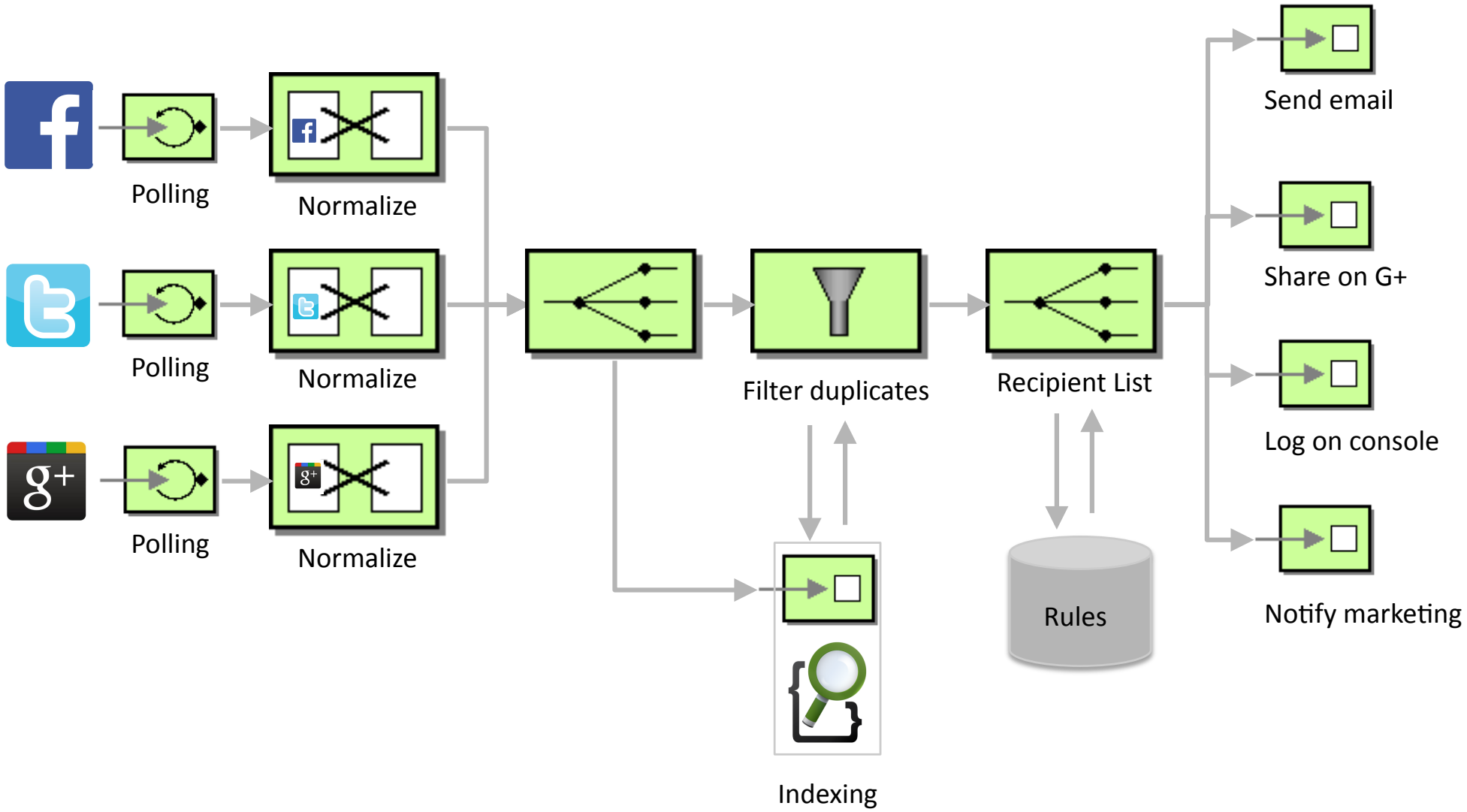
Traitement en // depuis une API REST



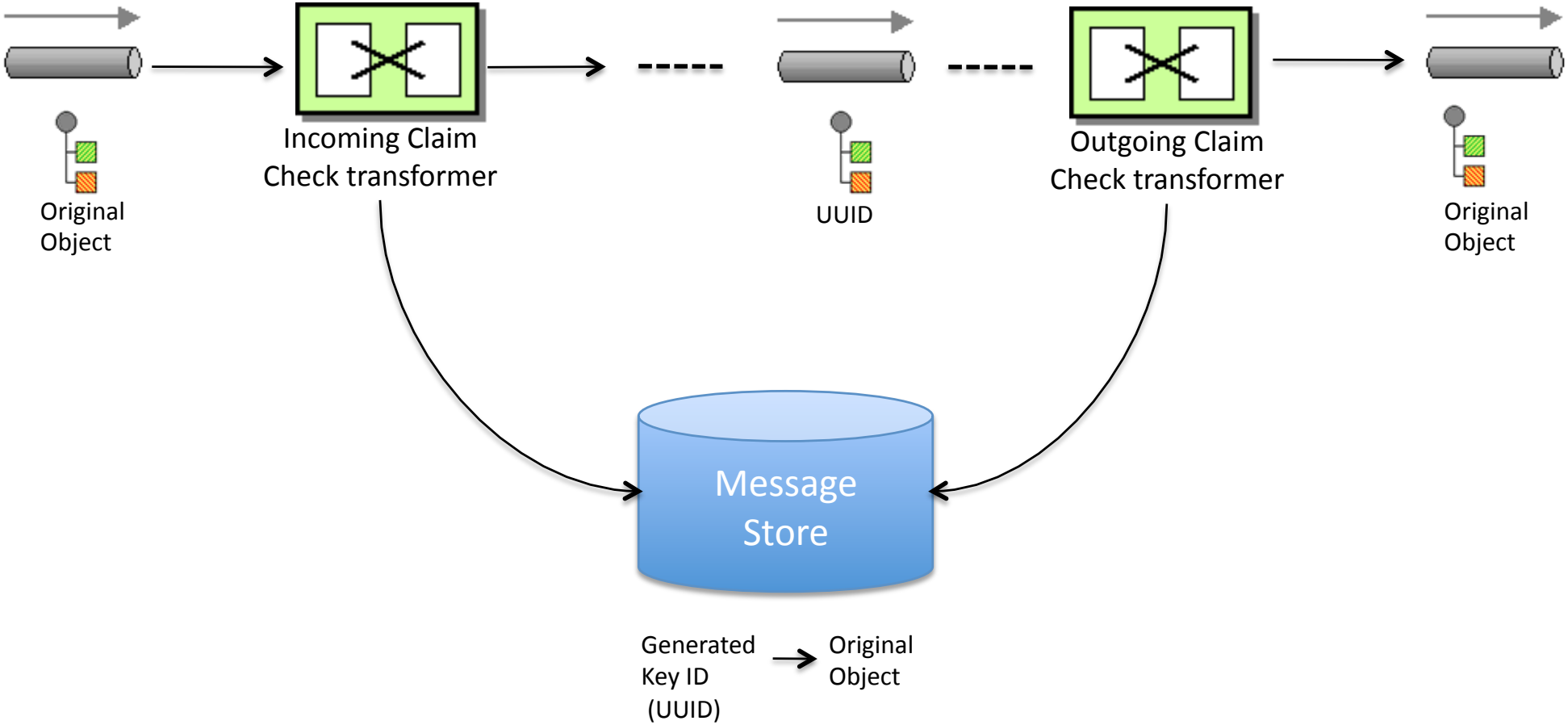
Comparateur de prix



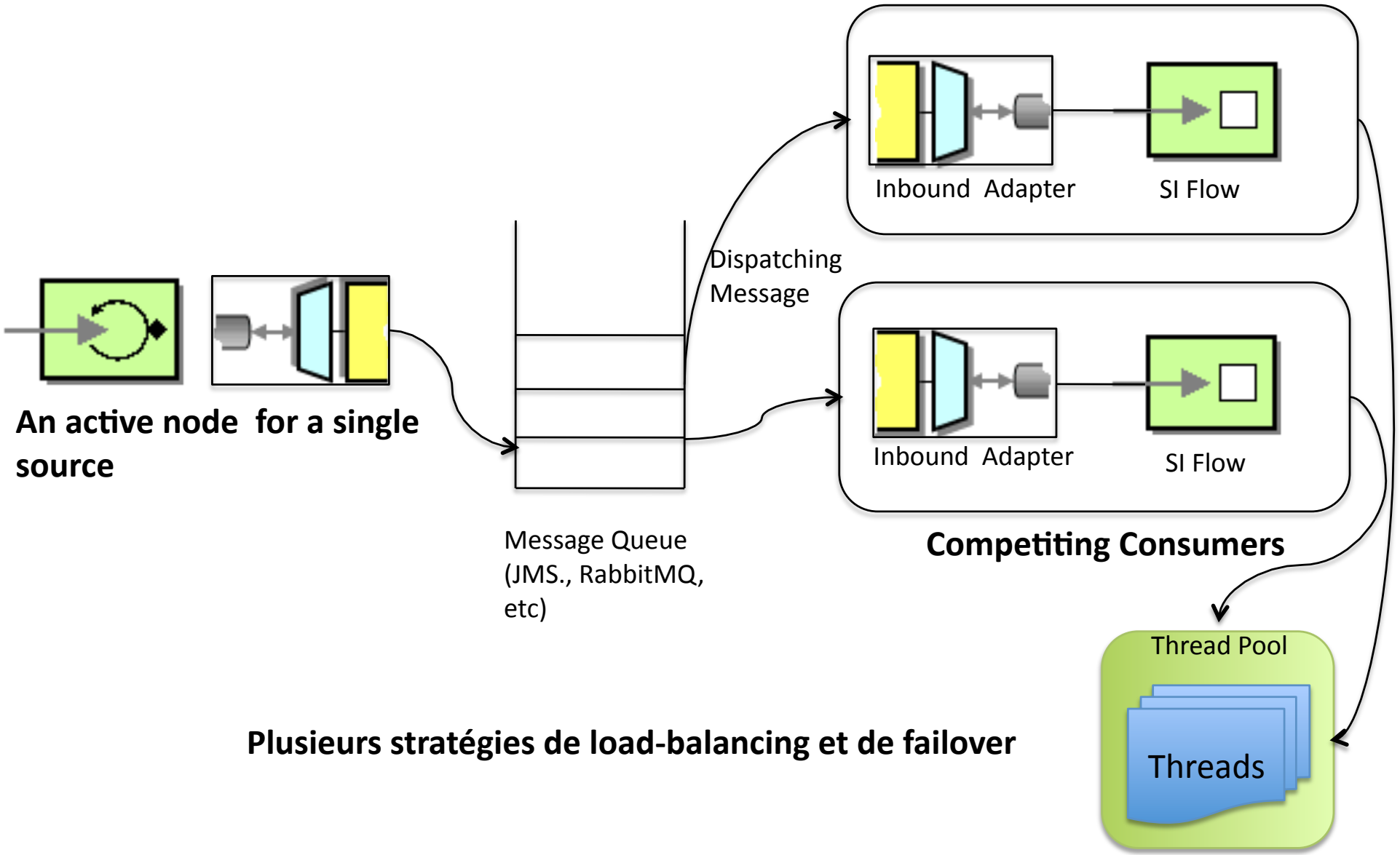
Social Crawler



Scaling Integration Flow
"Claim Check Pattern"



High Availability (HA) Architecture



Plusieurs stratégies de load-balancing et de failover

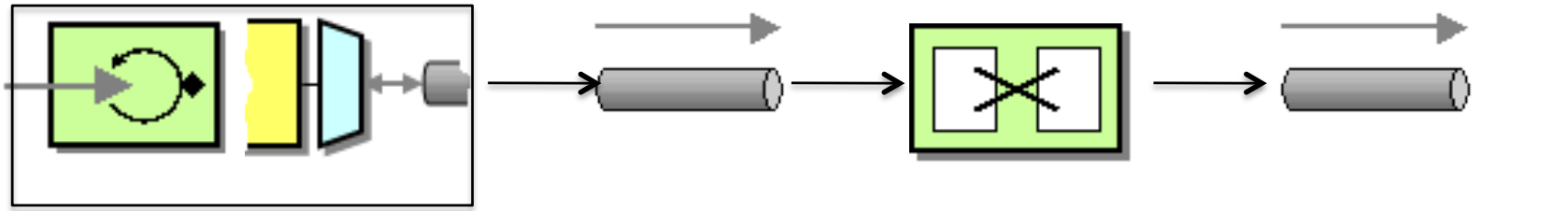
Quelques mots sur le monitoring

Message History

Wire-Tap

JMX

Message History

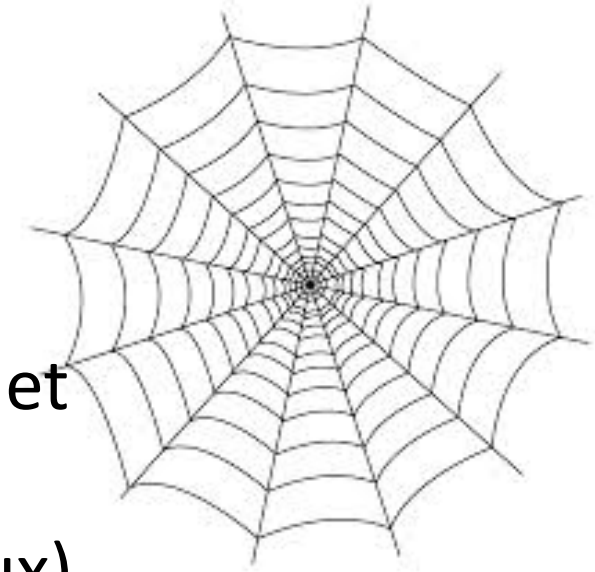


[name:poller, type:inbound-channel-adapter,
timestamp:1304966973309]

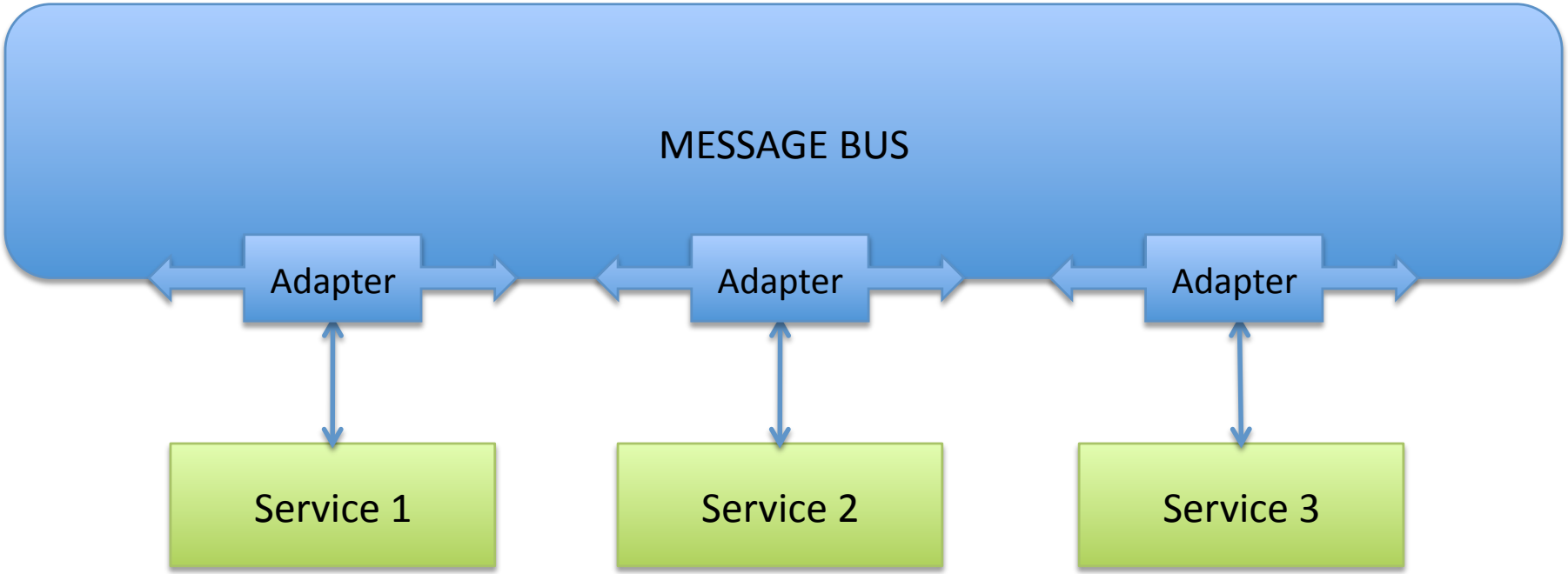
[name:inputChannel, type:channel,
timestamp:1304966973309]

[name:logerChannel, type:channel,
timestamp:1304966973309]

Et les ESB?

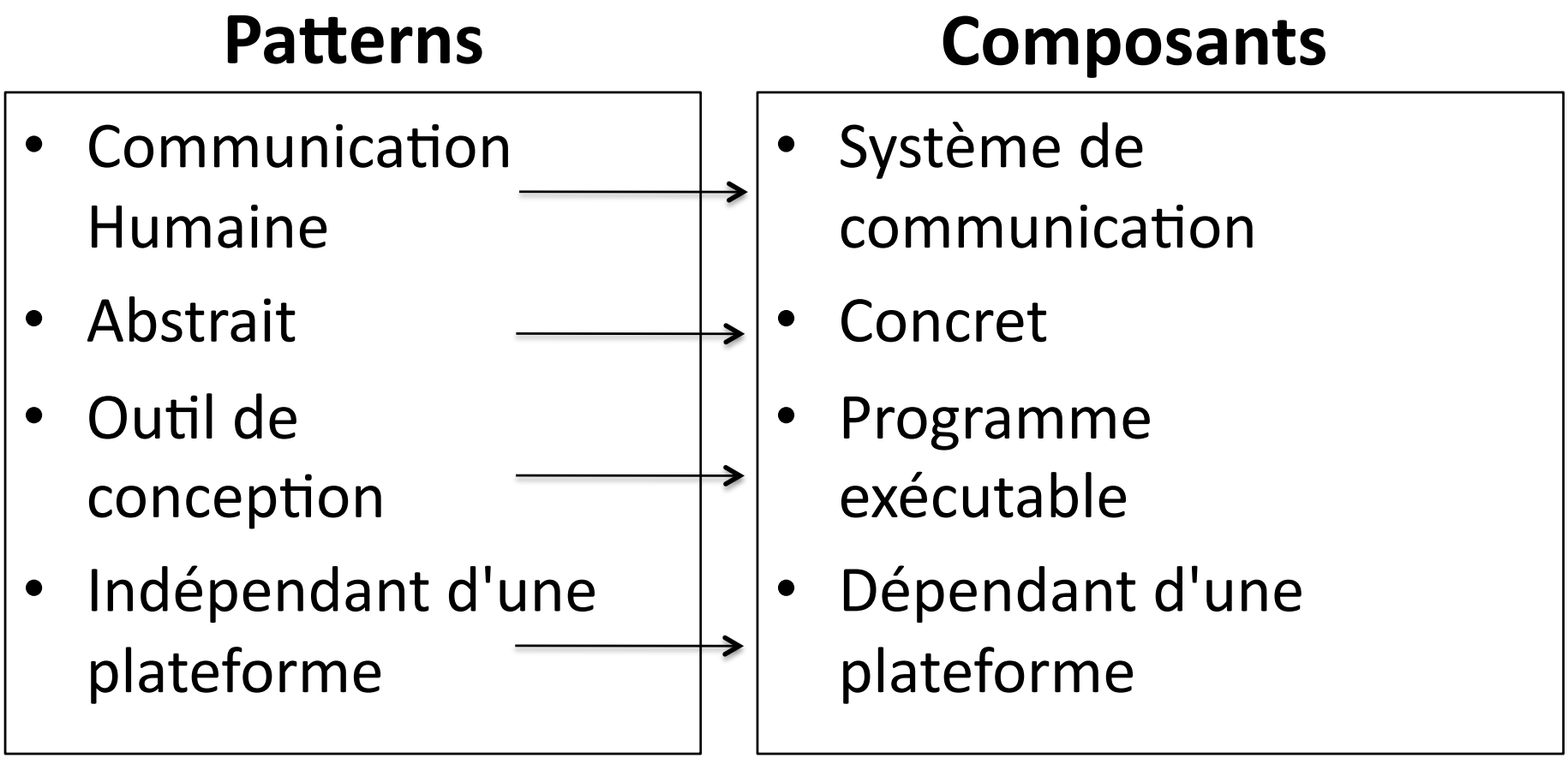


- Les entreprise Service Bus (ESB) permet de composer des applications SOA (moteur+ monitoring + gestion des flux)



Les frameworks d'intégration Java

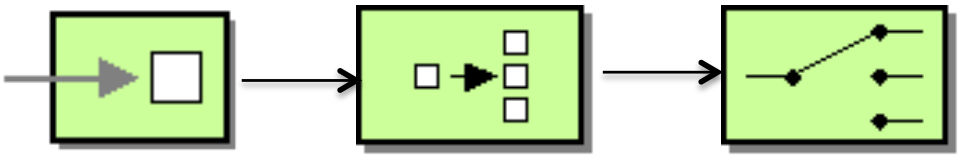
Projection des patterns sur des plateformes technologiques



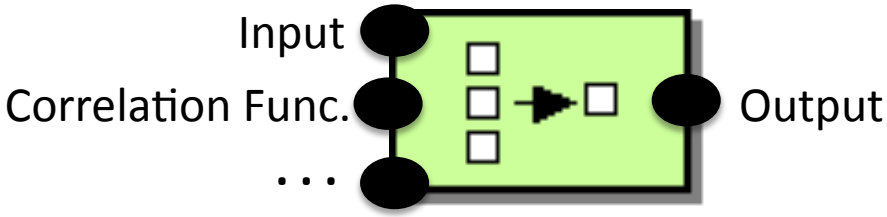
Deux concepts pour une même définition

Les patterns utilisable comme des composants

- Un modèle par composition pour le style "Pipe & Filter"



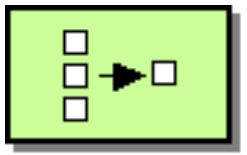
- On formalise facilement les entrées, les sorties et les autres propriétés



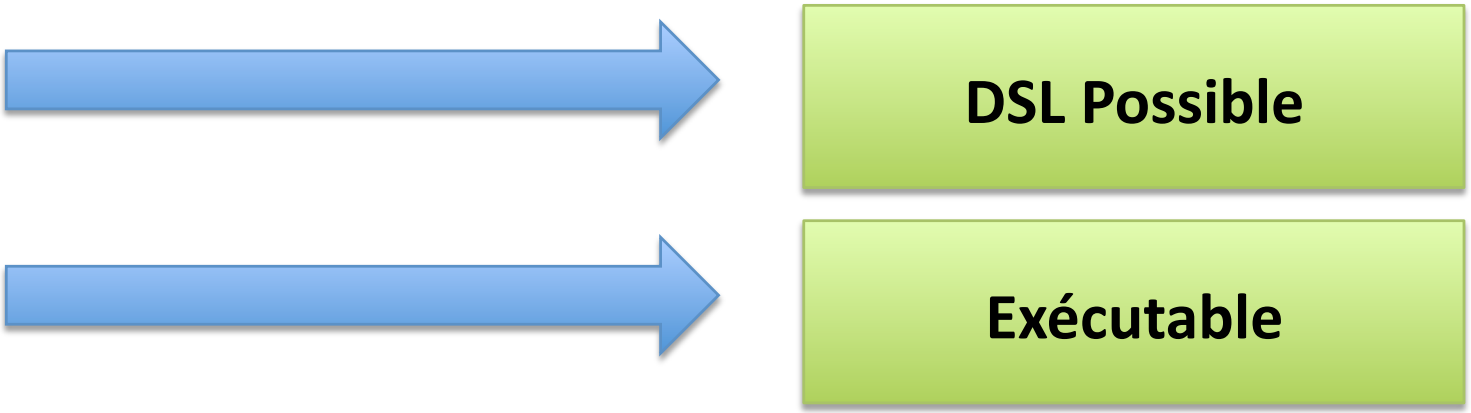
- Exprimable en différents langages

Le composant Aggregator en pratique

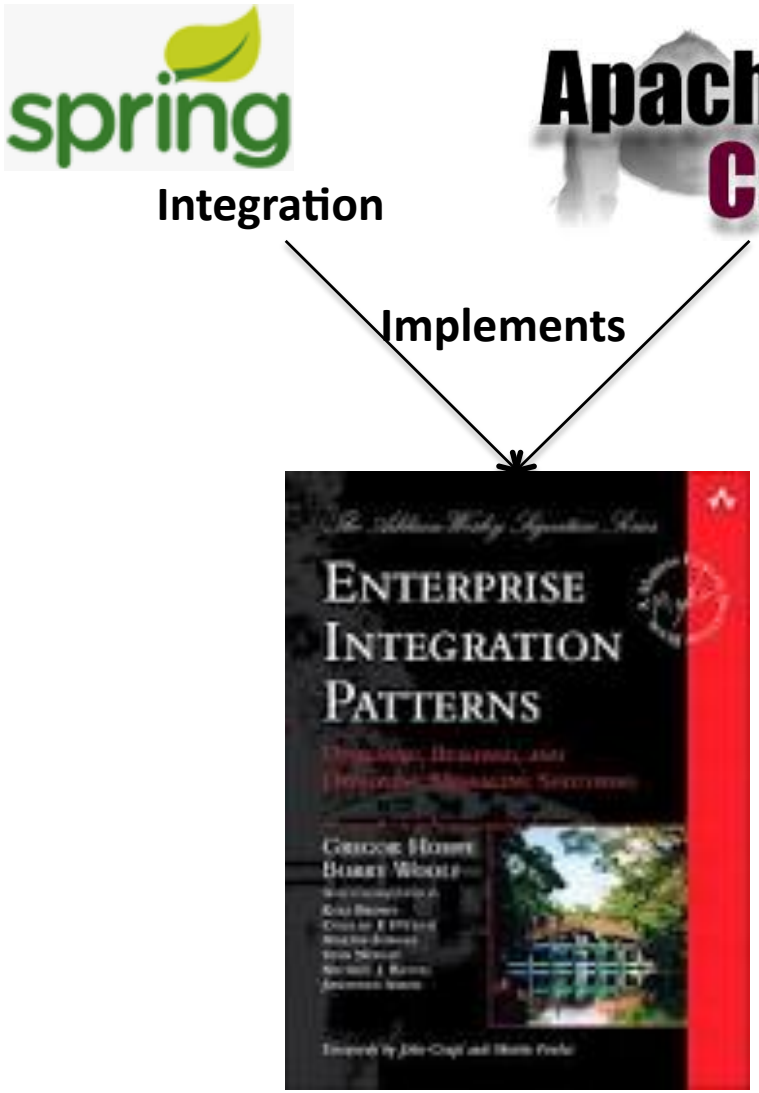
- Un composant avec un ensemble de propriétés
  - InputChannel
  - OutputChannel
  - Correlation Function
  - Completeness Condition
  - Aggregation algorithm



Aggregator

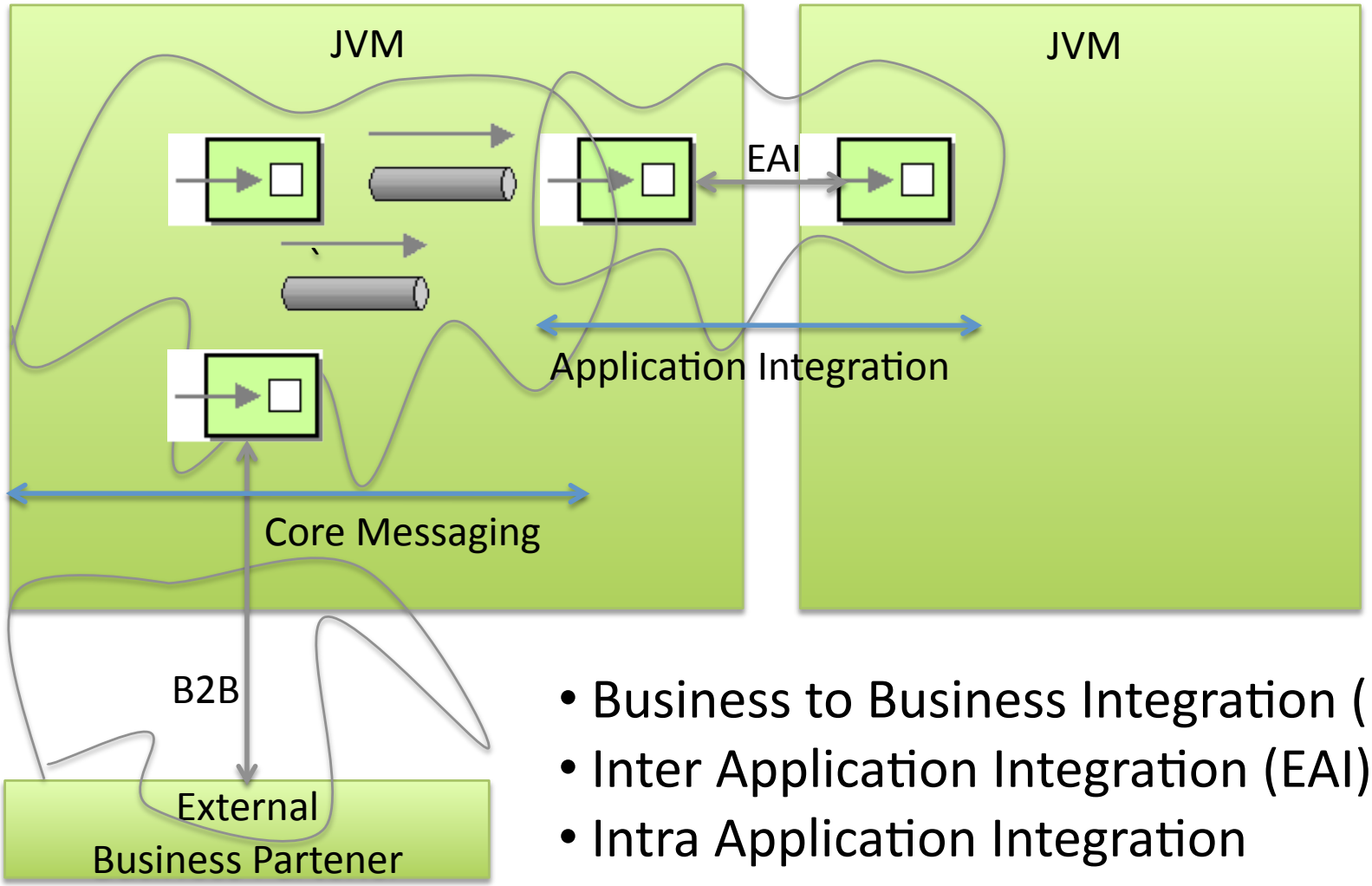


Les frameworks d'intégration Java



- Framework de médiation et de routing
- Pas un ESB

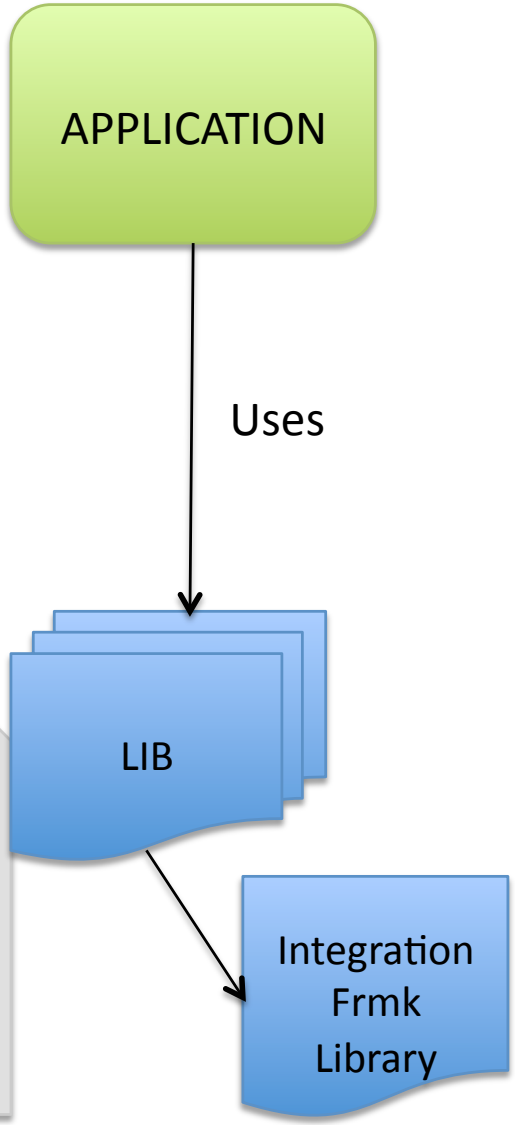
Un usage intra et inter application



- Business to Business Integration (B2B)
- Inter Application Integration (EAI)
- Intra Application Integration

Déploiement du framework d'intégration

- 1 simple librairie non invasive
- Pas de container/serveur (ou de broker)
- Aucune installation
- Pas de configuration logicielle




```
<!-- Exemple avec Spring Integration -->
<dependency>
   <groupId>org.springframework.integration</groupId>
   <artifactId>spring-integration-core</artifactId>
   <version>${spring.integration.version}</version>
</dependency>
```

Un déploiement comme "Middleware Integration platform"