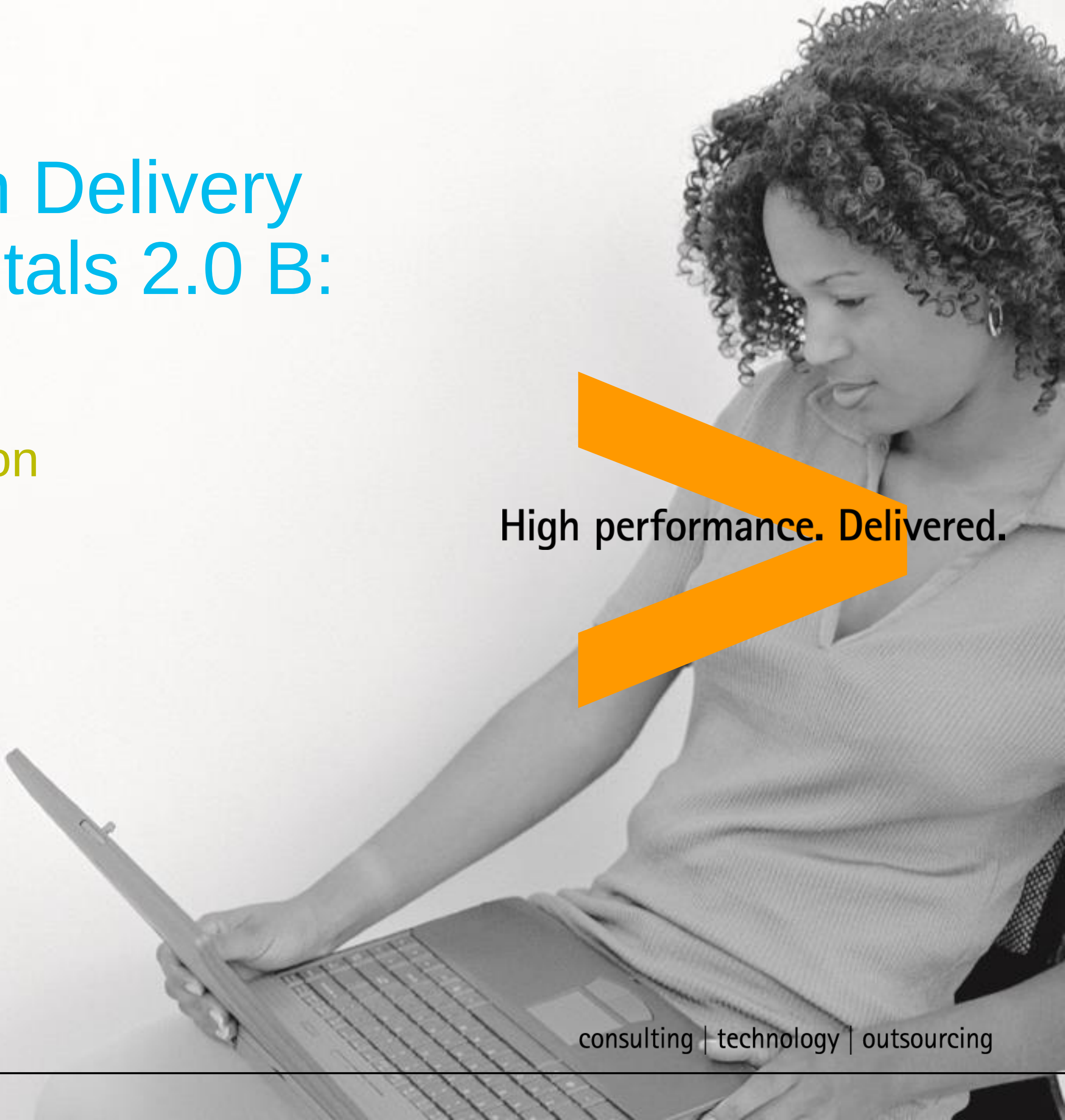
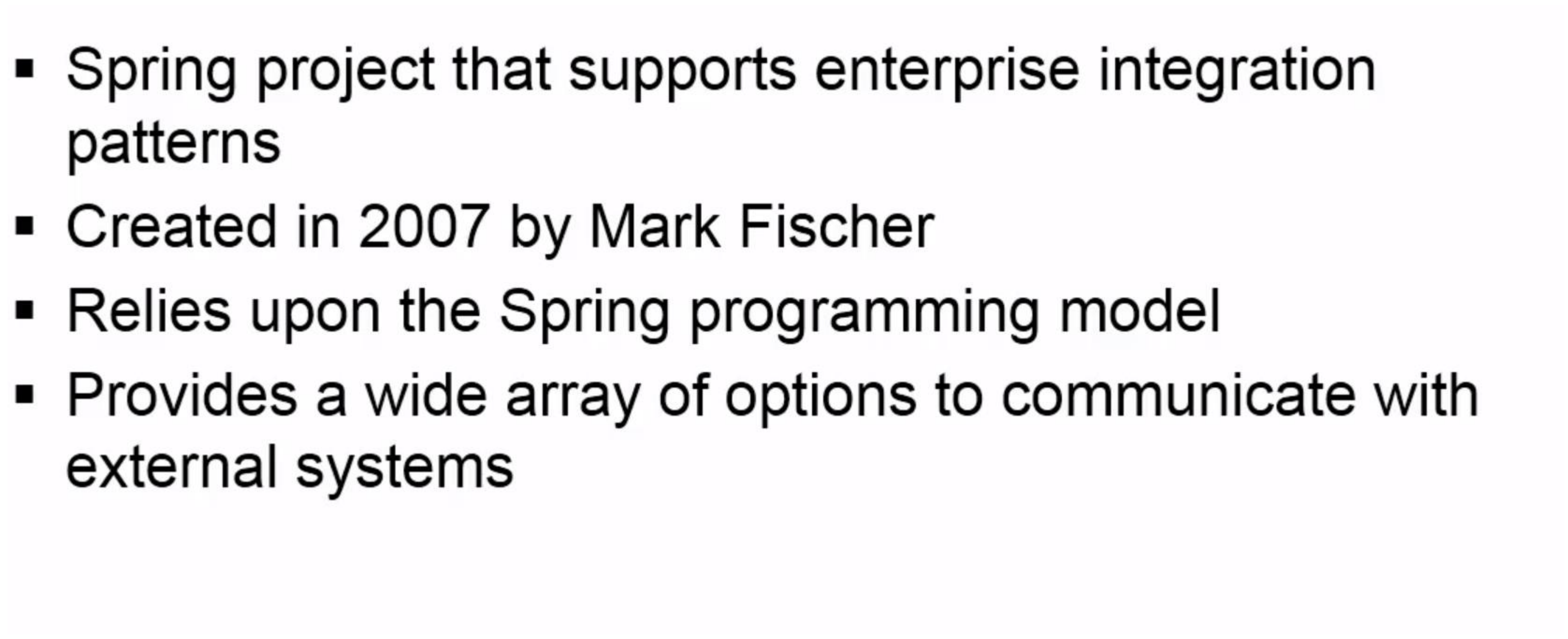
Application Delivery Fundamentals 2.0 B: Java

Spring Integration

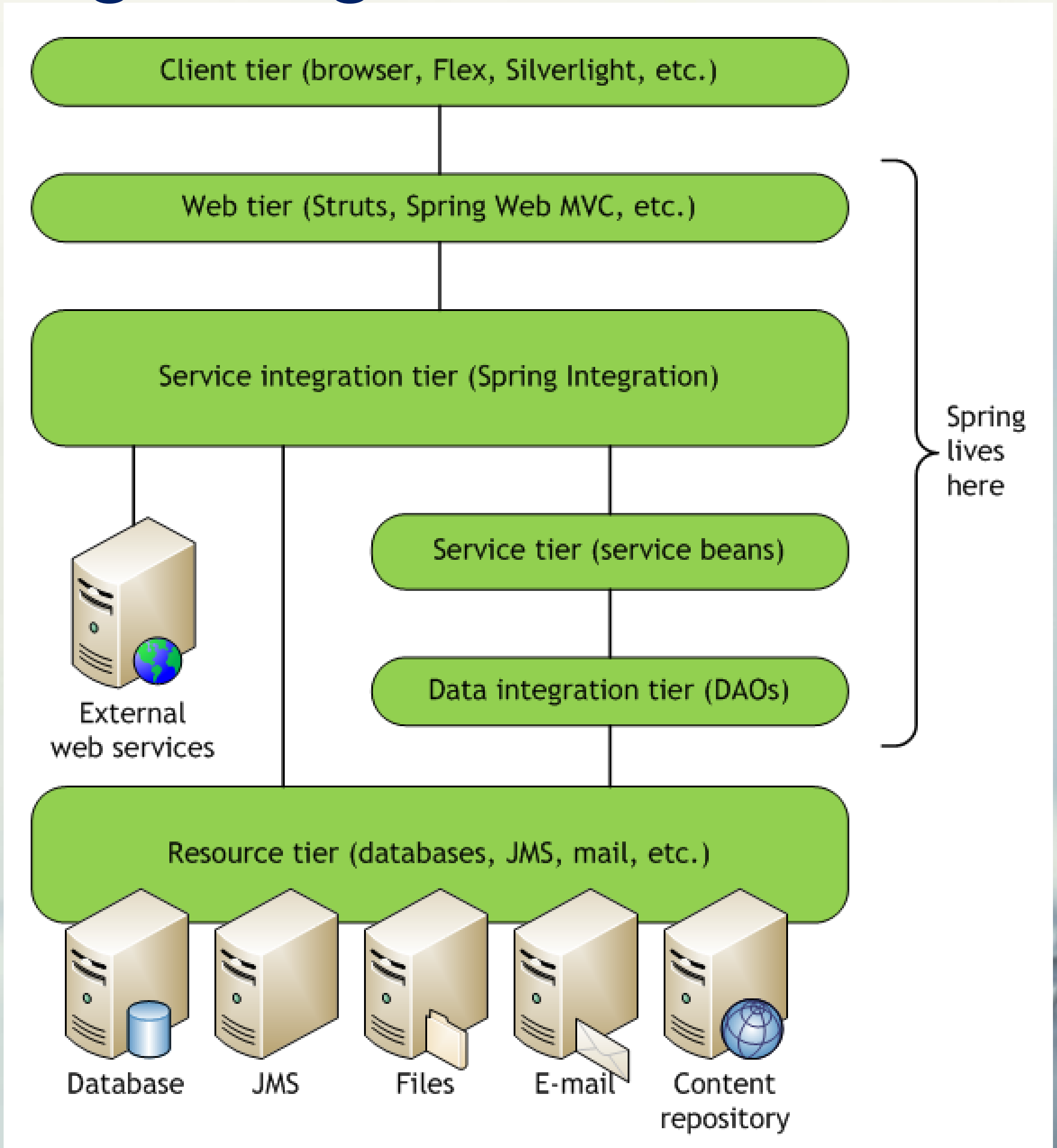

High performance. Delivered.
consulting | technology | outsourcing

Spring Integration



- Spring project that supports enterprise integration patterns
- Created in 2007 by Mark Fischer
- Relies upon the Spring programming model
- Provides a wide array of options to communicate with external systems

Spring Integration


Client tier (browser, Flex, Silverlight, etc.)
Web tier (Struts, Spring Web MVC, etc.)
Service integration tier (Spring Integration)
Service tier (service beans)
Data integration tier (DAOs)
Spring lives here
External web services
Resource tier (databases, JMS, mail, etc.)
Database
JMS
Files
E-mail
Content repository

Spring Integration Framework

- Spring Integration provides an extension of Spring’s plain old Java object (POJO) programming model to support the standard integration patterns.
- Spring Integration works in terms of the fundamental idioms of integration, including messages, channels, and endpoints.
- It enables messaging within Spring-based applications and integrates with external systems via Spring Integration’s adapter framework.

Spring Integration Framework

- The adapter framework provides a higher level of abstraction over Spring’s existing support for remote method invocation, messaging, scheduling and much more.

Exploring the Alternatives

- Mule provides a lightweight container and leverages a simple XML configuration file and Plain Old Java Objects (POJOs).
- ServiceMix is based on the Java Business Integration (JBI) standard and now supports OSGI.
- OpenESB is (http://open-esb.dev.java.net) the integration offering from Oracle based on the JBI and J2EE standards.
- OpenESB is designed to live in a J2EE application server like GlassFish.

Spring Integration Basics

- Spring Integration adds essentially three components to the core Spring Framework: messages, message channels, and endpoints.

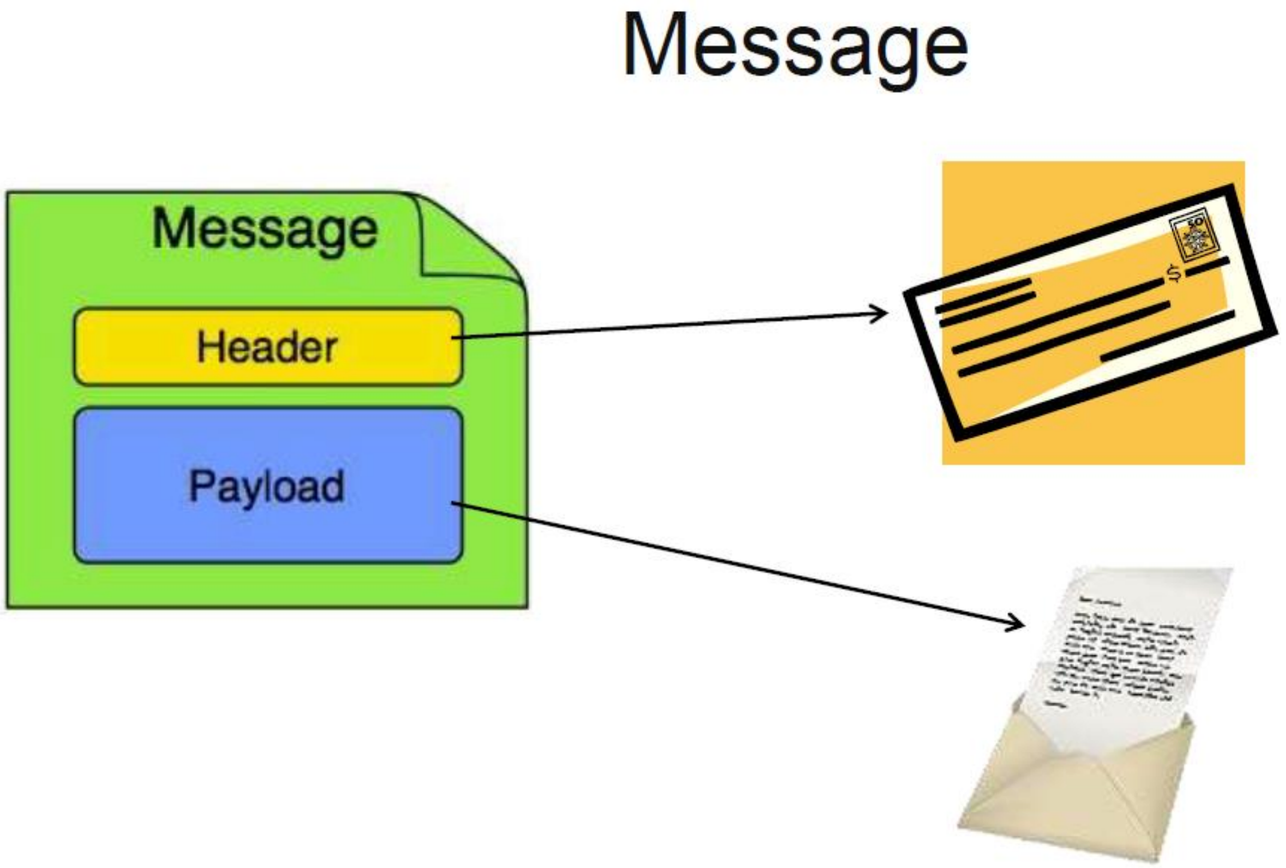
Goals

- Spring Integration is motivated by the following goals:
- Provide a simple model for implementing complex enterprise integration solutions.
- Facilitate asynchronous, message-driven behavior within a Spring-based application.
- Promote intuitive, incremental adoption for existing Spring users.

Principles

- Spring Integration is guided by the following principles:
- Components should be ***loosely coupled*** for modularity and testability.
- The framework should enforce ***separation of concerns*** between business logic and integration logic.
- Extension points should be abstract in nature but within well-defined boundaries to promote ***reuse*** and ***portability***.

Message



Messages

- A message is a generic wrapper for any Java object combined with metadata used by Spring Integration to handle the object.

  ```
  public interface
  org.springframework.integration.Message<T> {
  MessageHeaders getHeaders();
  T getPayload();
  }
  ```

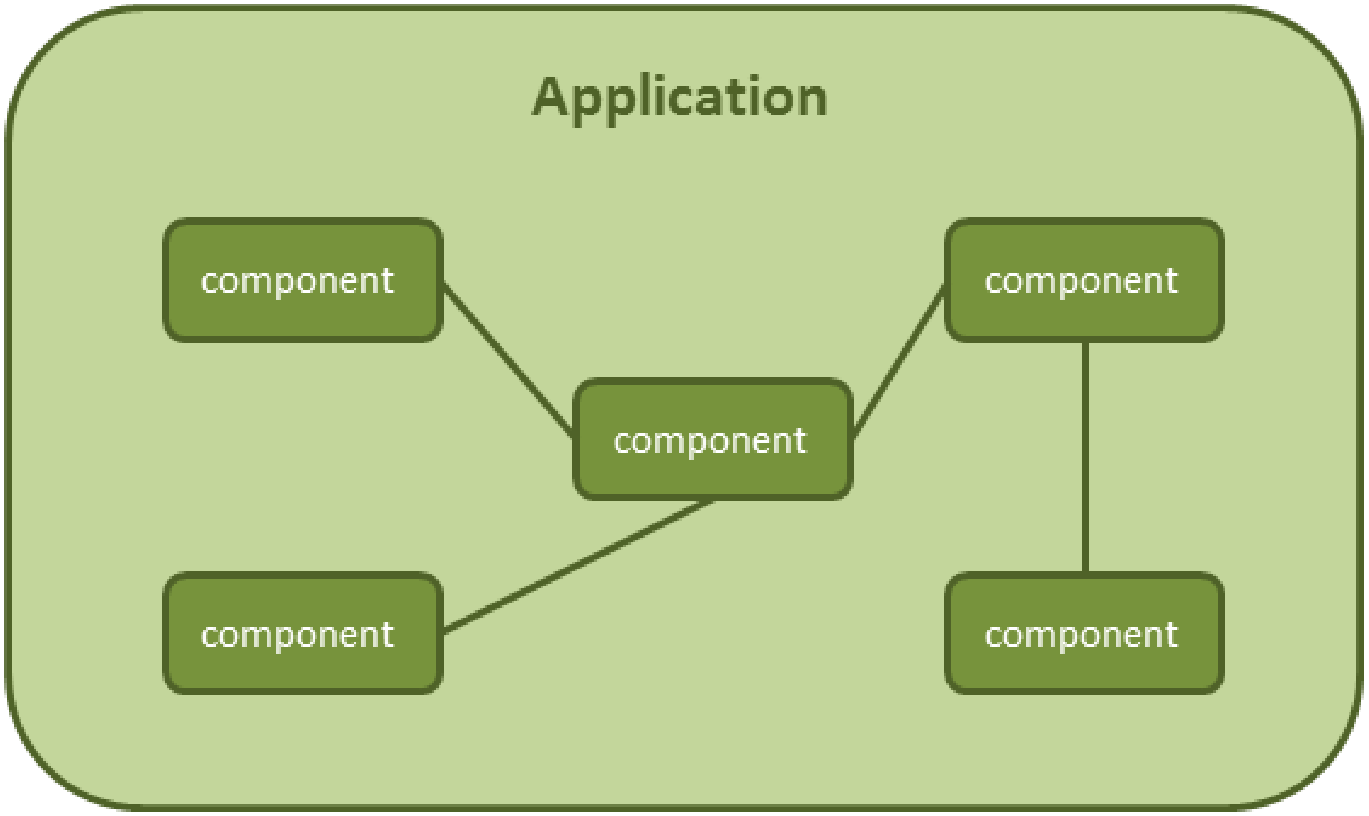
Headers

- The org.springframework.integration.MessageHeaders object is a String/Object Map that typically maintains values for message housekeeping chores.
- MessageHeaders are immutable and are usually created using the MessageBuilder API.
- There are a number of predefined entries (headers), including id, timestamp, correlation id, and priority.
- The headers can maintain values required by the adapter endpoints.
- Headers may be used any key/value pair required by the developer.

Payloads

- The message payload can be any POJO.
- Transformation support allows converting any payload into any type of format required by the message endpoint.

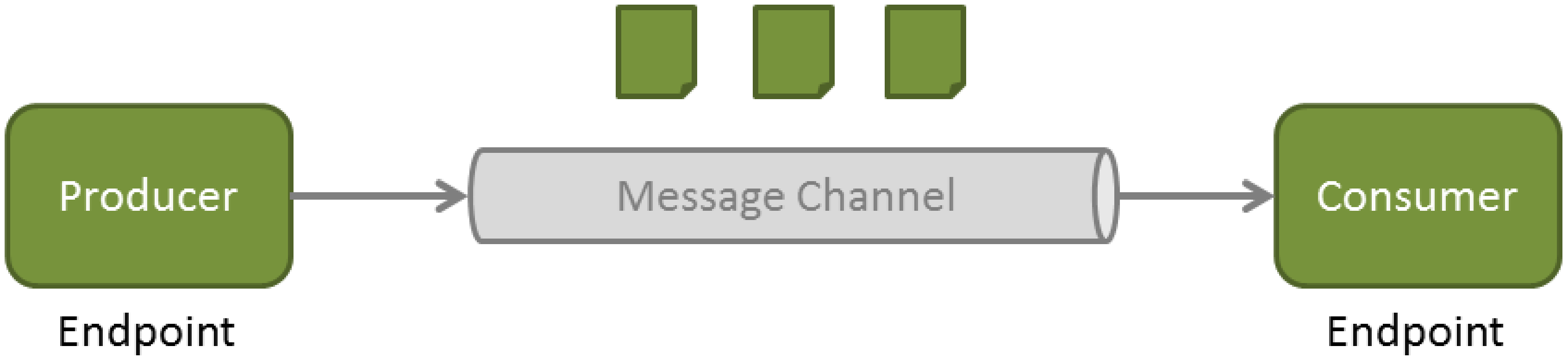
Message


Application
component
component
component
component
component

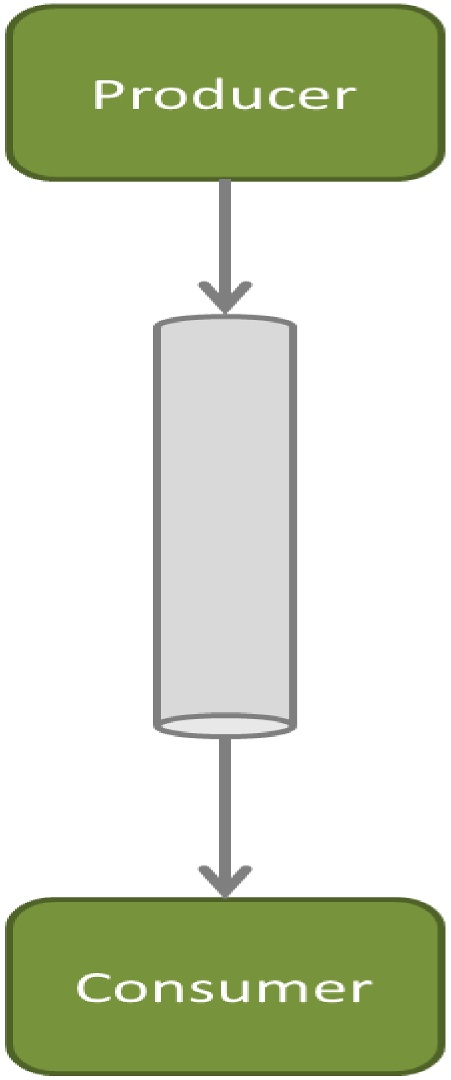
Message Channels

- A message channel is the component through which messages are moved.
- Message publishers send messages to the channel, and message consumers receive messages from the channel.
- The channel effectively decouples the producer and consumer.
- **There are two types of messaging scenarios:**
- ***point-topoint*, in which a message is received only once, by a single consumer;**
- ***publish/subscribe*, in which one or more consumers can attempt to receive a single message.**

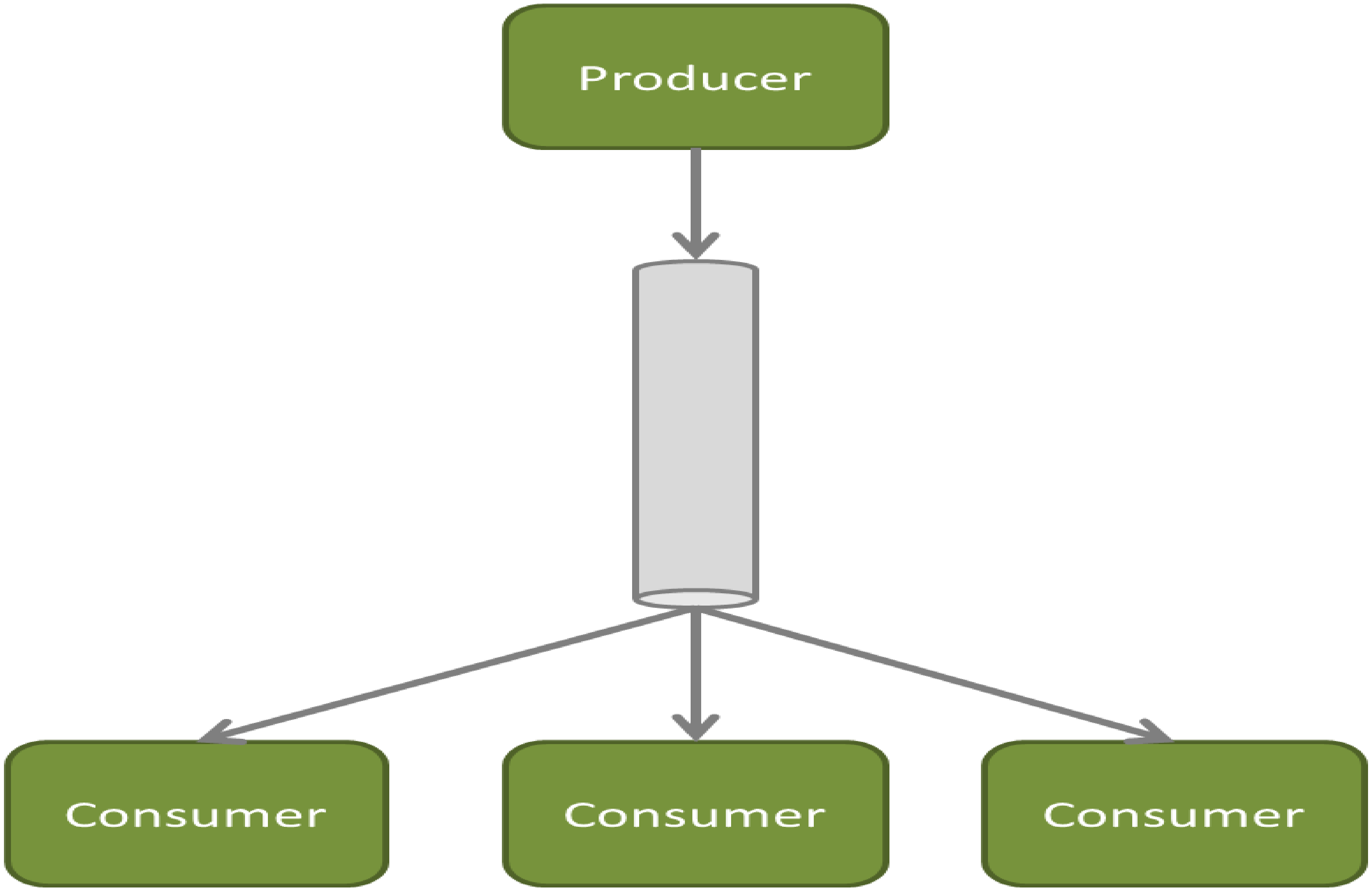
Message Channel



Point to Point

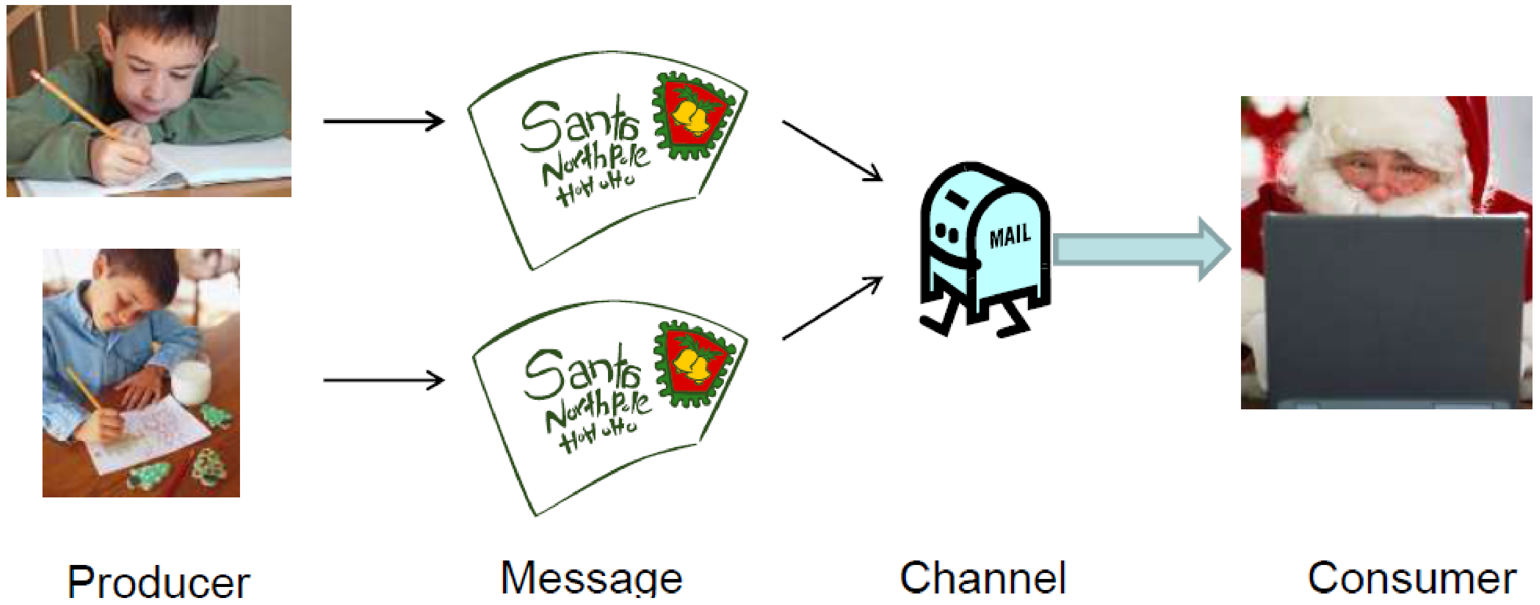


Publish and Subscribe



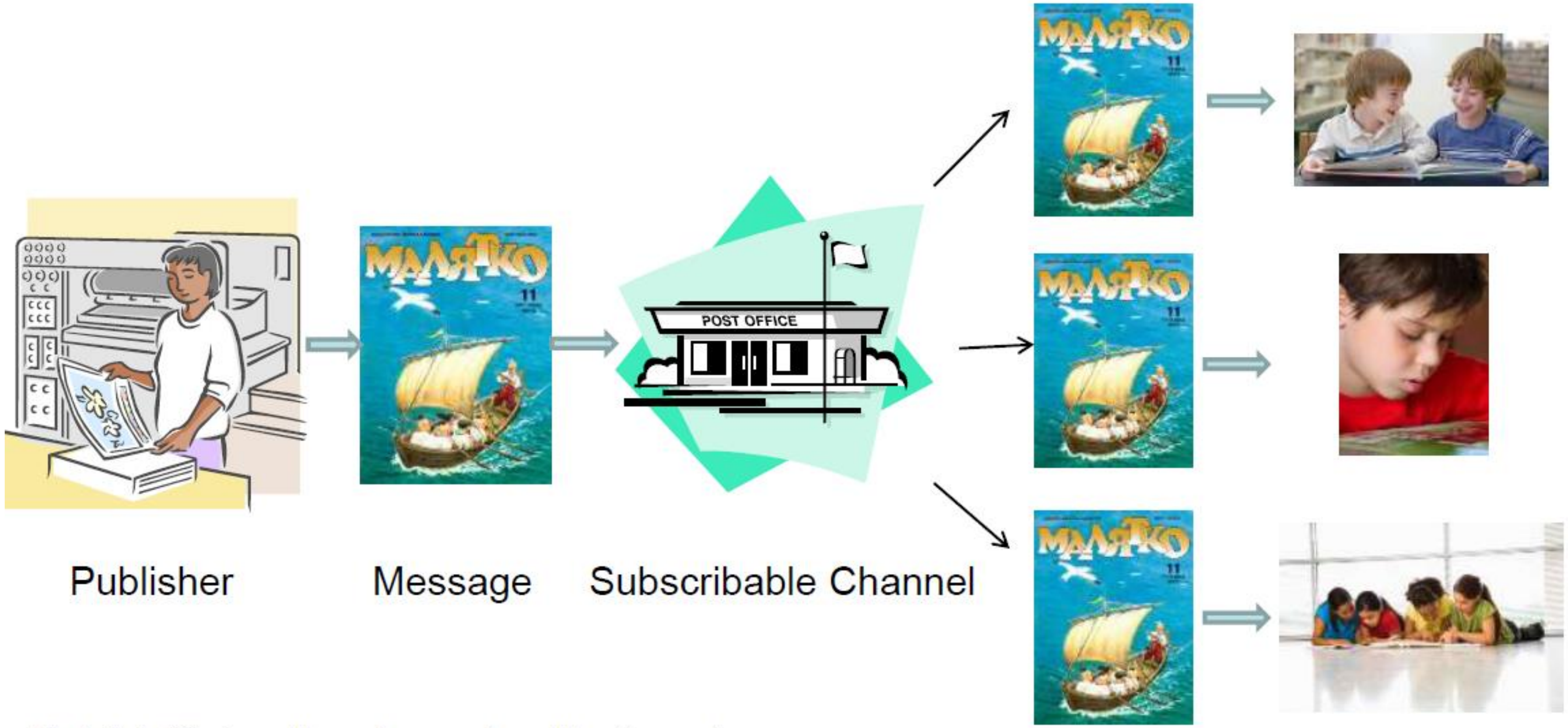
Message Channels

With a Point-to-Point channel,
at most one consumer can receive each Message sent to the channel.



Message Channels

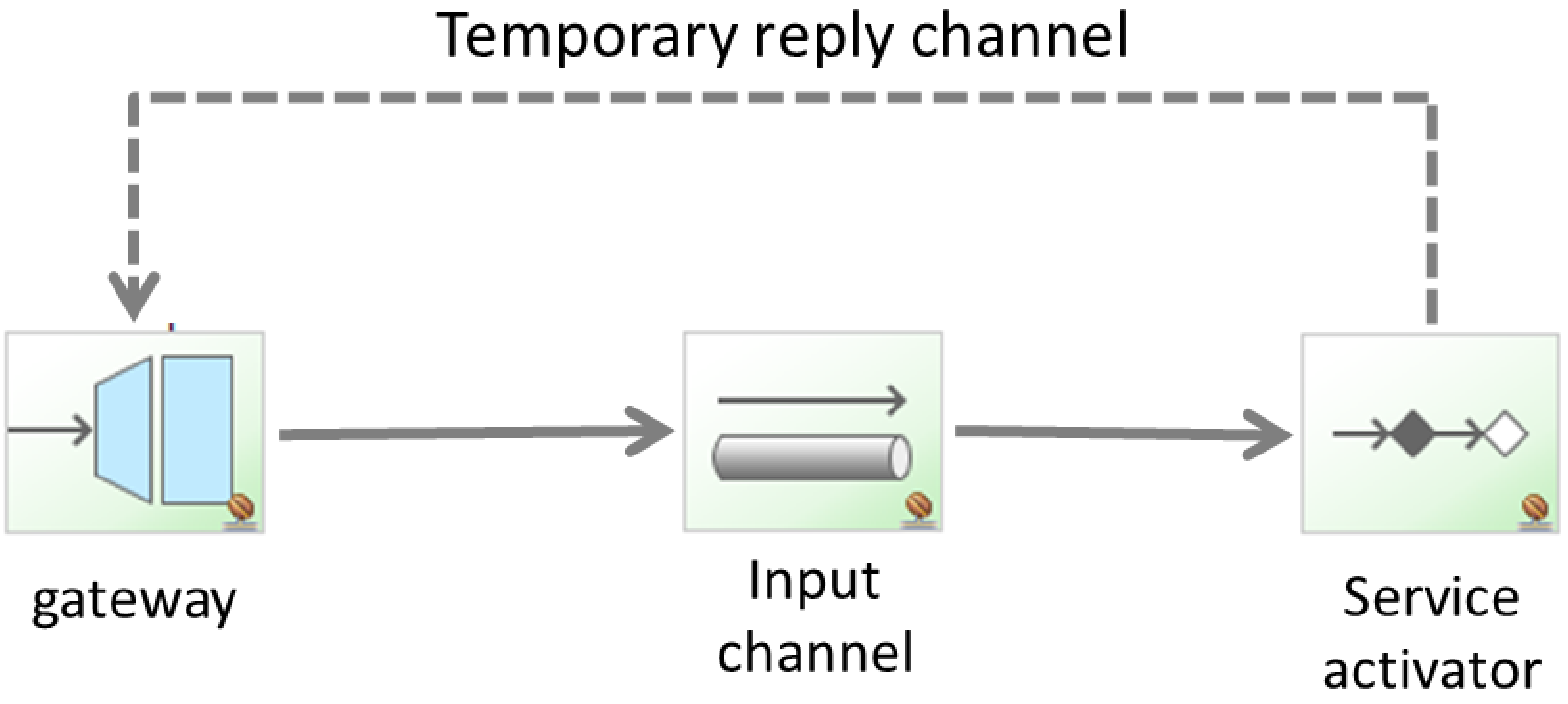
Publish / Subscribe



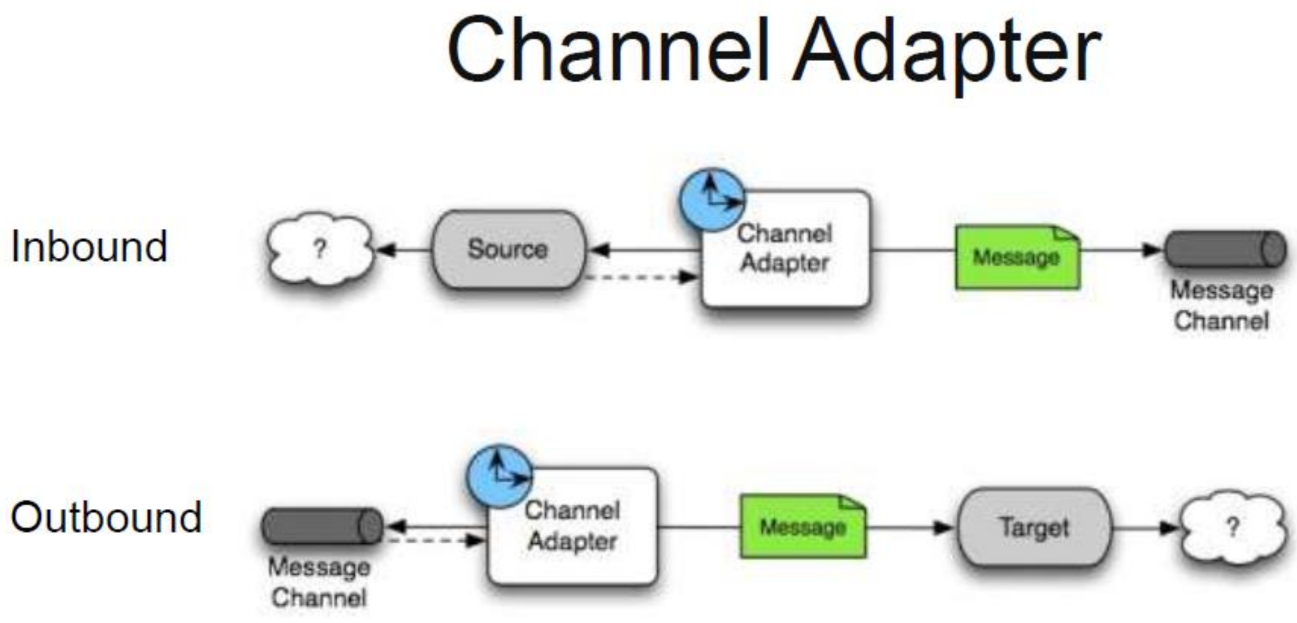
Publish/Subscribe channels will attempt
to broadcast each Message to all of its subscribers

Subscribers

Temporary Reply Channel

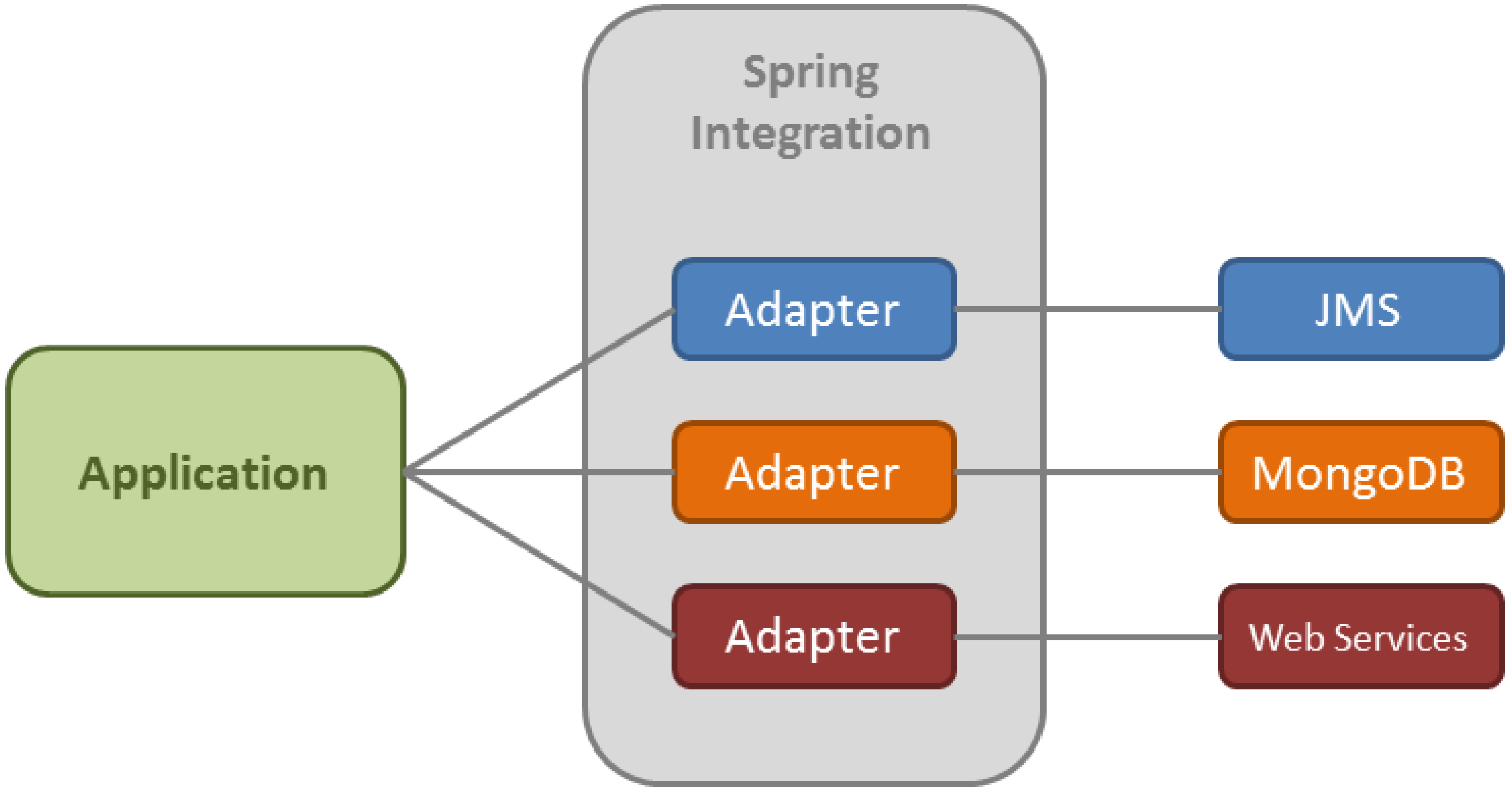


Channel Adapter

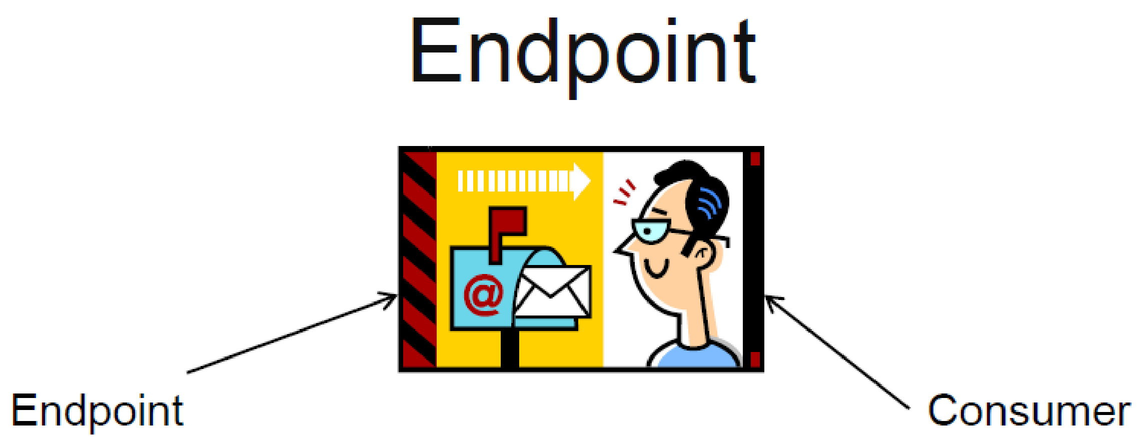


- An endpoint that connects a Message Channel to some other system or transport
- May be either inbound or outbound
- Typically, will do some mapping between the Message and whatever object or resource is received-from or sent-to the other system (File, HTTP Request, JMS Message, etc)

Channel Adapter



Endpoint



- A Message Endpoint represents the "filter" of a pipes-and-filters architecture.
- The endpoint's primary role is to connect application code to the messaging framework, and to do so in a non-invasive manner.

Message Endpoints

- A message endpoint is the abstraction layer between the application code and the messaging framework.
- An endpoint broadly defines all the types of components used in Spring Integration.
- It handles such tasks as producing and consuming messages, as well as interfacing with application code, external services, and applications.
- When data travels through a Spring Integration solution, it moves along channels from one endpoint to another.
- Data can come into the framework from external systems using a specific type of endpoint called an adapter.

Message Endpoints

- The main endpoint types supported by Spring Integration are as follows:
- *Transformer*: Converts the message content or structure.
- *Filter*: Determines if the message should be passed to the message channel.
- *Router*: Can determine which channel to send a particular message based on its content.
- *Splitter*: Can break an incoming message into multiple messages and send them to the appropriate channel.

Message Endpoints

- *Aggregator*: Can combine multiple messages into one. An aggregator is more complex than a splitter often required to maintain state.
- *Service activator*: Is the interface between the message channel and a service instance, many times containing the application code for business logic.
- *Channel adapter*: Is used to connect the message channel to another system or transport.

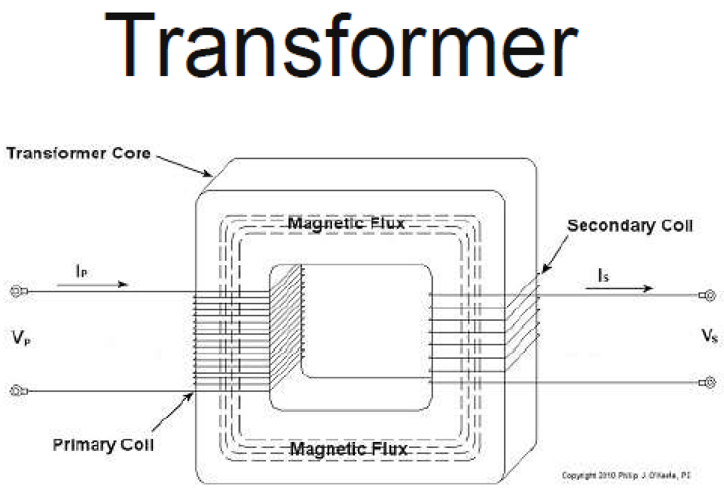
Integration with External Systems

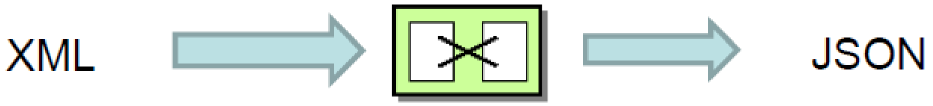
Spring Integration provides a number of endpoints used to interface with external systems, file systems etc.

Integration with External Systems

- AMQP (RabbitMQ)
- Spring ApplicationEvents
- Feed (RSS, ATOM, …)
- File
- FTP
- GemFire
- HTTP
- JDBC
- JMS
- JMX
- JPA (Hibernate, OpenJPA, …)
- Mail
- MongoDB
- MQTT
- Redis
- Resource
- RMI
- SFTP
- Stream
- Syslog
- TCP
- Twitter
- UDP
- Web Services
- XMPP

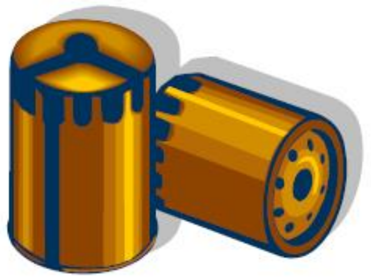
Transformer

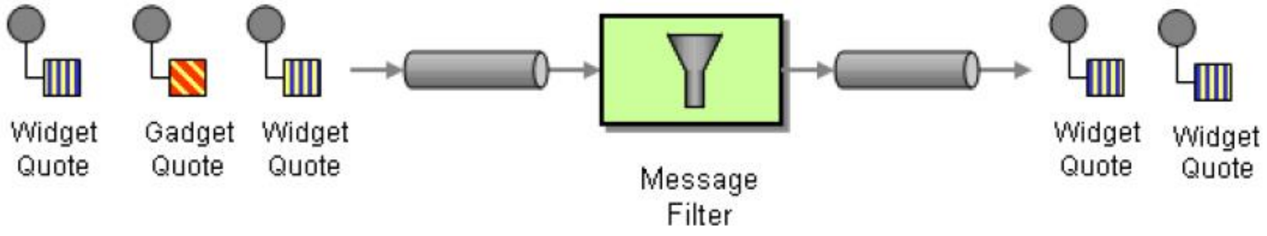




- Is responsible for converting a Message's content or structure and returning the modified Message.
- The most common type of transformer is one that converts the payload of the Message from one format to another (e.g. from XML Document to JSON).
- May be used to add, remove, or modify the Message's header values.

Filter

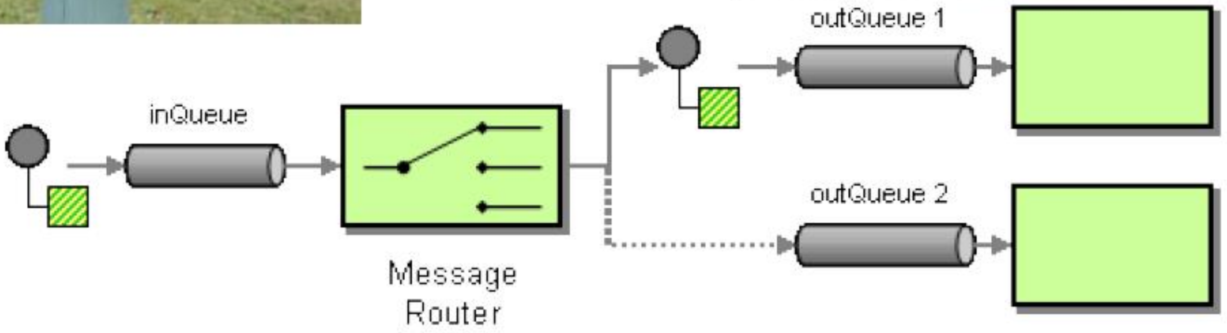




- Determines whether a Message should be passed to an output channel at all.
- Often used in conjunction with a Publish Subscribe channel, where multiple consumers may receive the same Message and use the filter to narrow down the set of Messages to be processed based on some criteria.

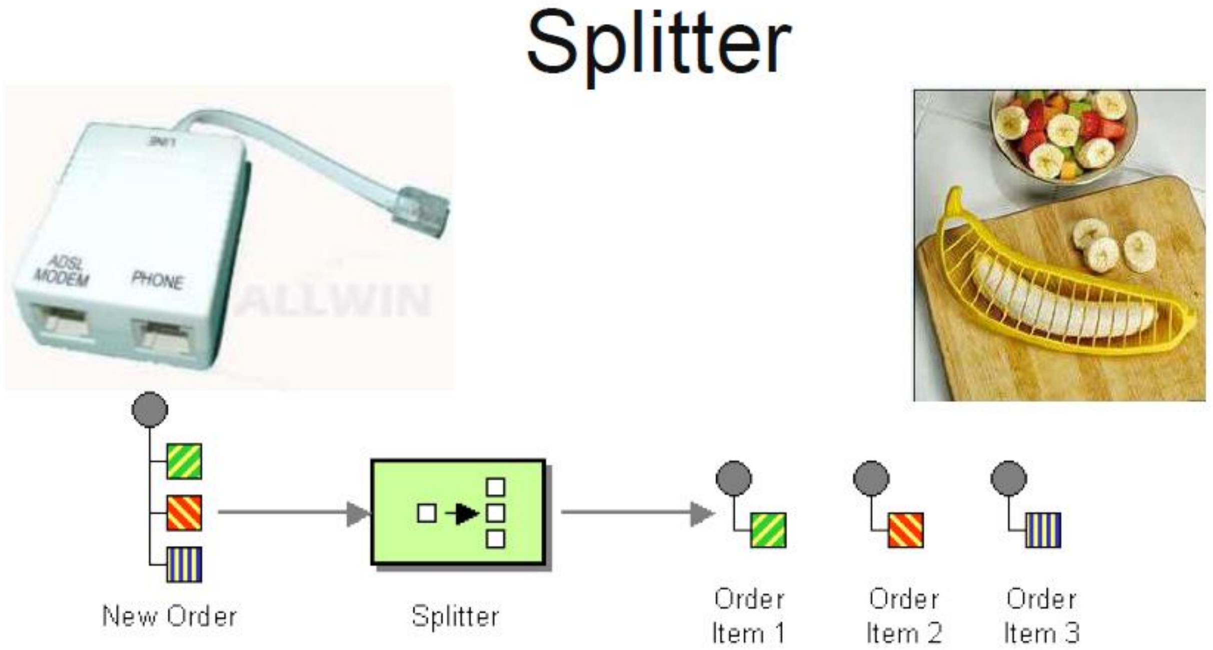
Router

Router



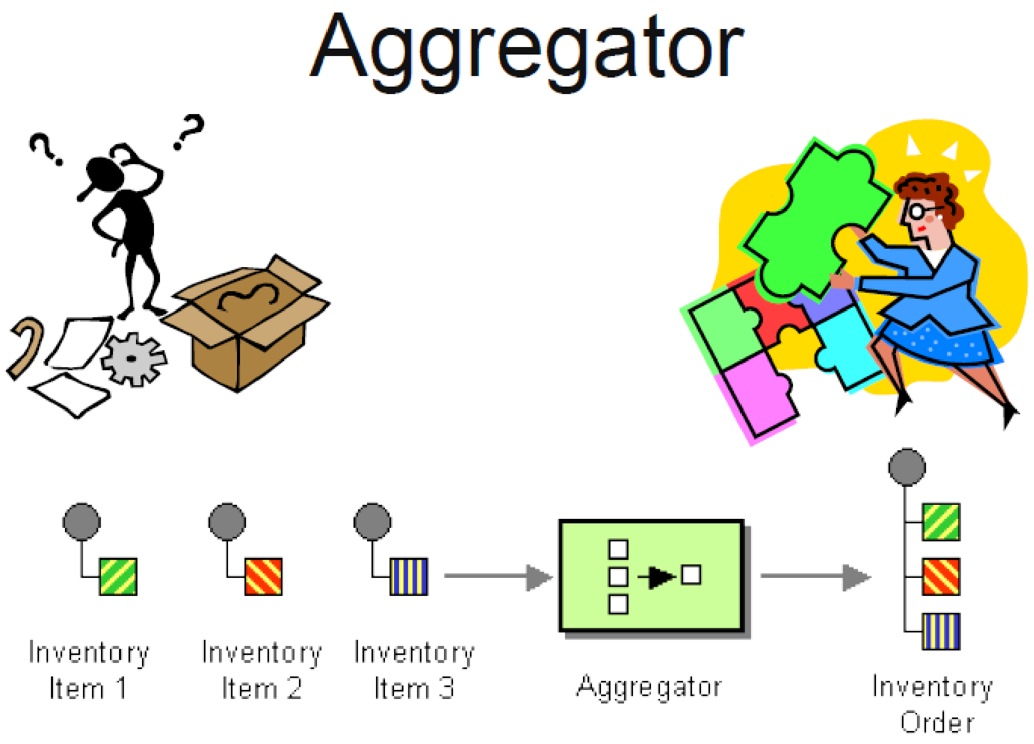
- Responsible for deciding what channel or channels should receive the Message next
- Often used as a dynamic alternative to a statically configured output channel
- Provides a proactive alternative to the reactive Message Filters used by multiple subscribers

Splitter



- Splits received Message into multiple Messages, and then sends each of those to its output channel.
- Typically used for dividing a "composite" payload object into a group of Messages containing the sub-divided payloads

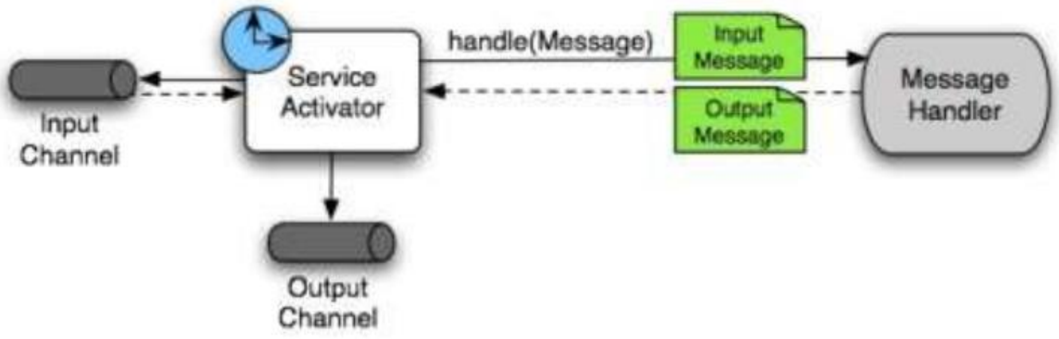
Aggregator



- A mirror-image of the Splitter
- Receives multiple Messages and combines them into a single Message
- Technically, is more complex than a Splitter, because it is required to maintain state (the Messages to-be-aggregated)

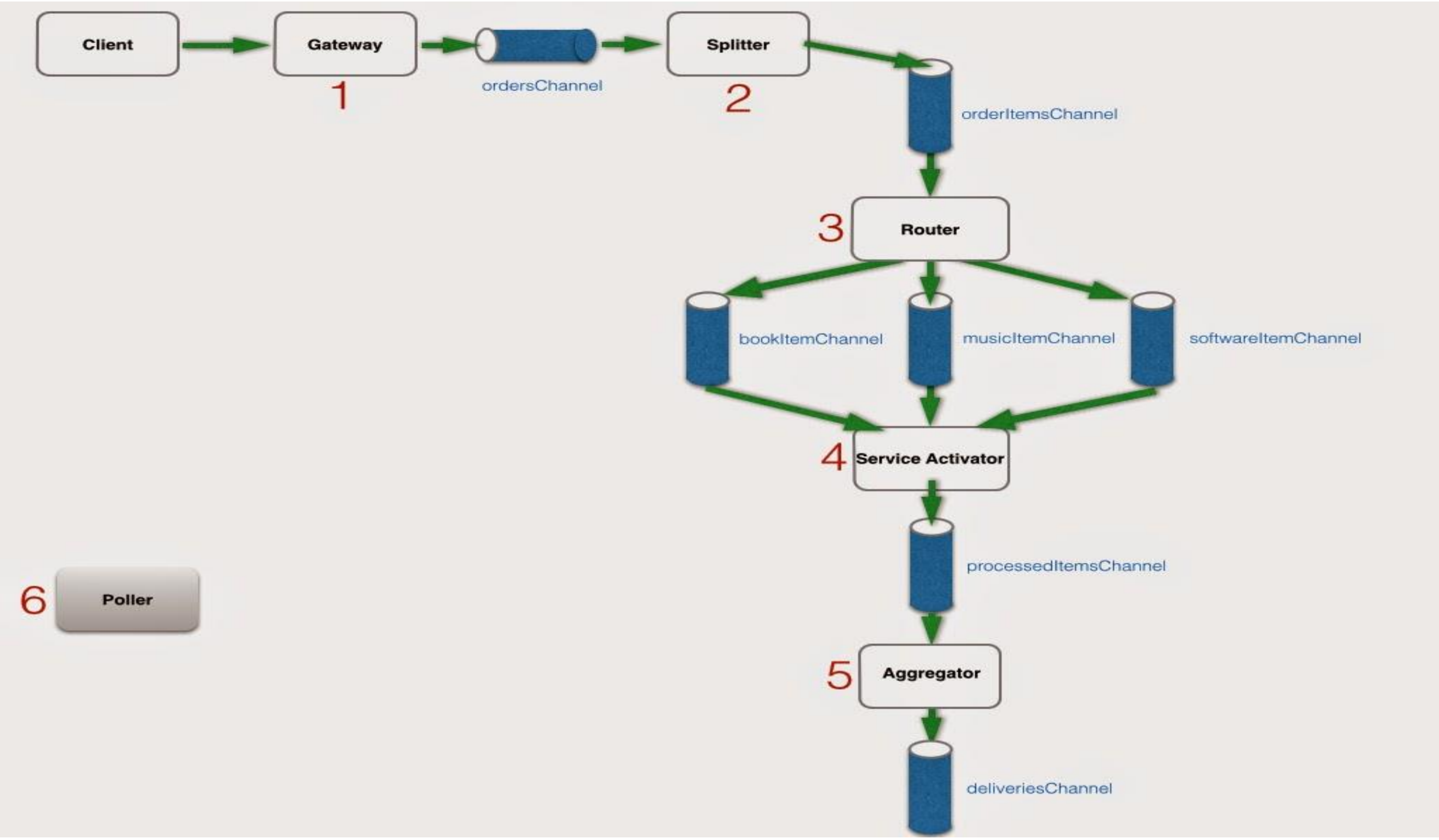
Service Activator

Service Activator



- A generic endpoint for connecting a service instance to the messaging system
- The input Message Channel must be configured
- If the service method to be invoked is capable of returning a value, an output Message Channel may also be provided.

Spring Integration



Event-Driven Architecture

- An event is a significant change in state.
- An event-driven architecture, this event get sent or published to all interested parties.
- The subscribing parties can look at the event and choose to respond or ignore it.
- The response may include invoking a service, running a business process, or publishing another event.

Event-Driven Architecture

- Event-driven architecture is loosely coupled, meaning that the publisher of the event has no knowledge of what the consumers do downstream after the event has been processed.
- Spring Integration is well suited for this type of architecture, as it captures events as messages.
- Event-driven architecture usually refers to simple event processing in which a published event leads to some downstream action.
- This type of processing is well suited to performing a real-time flow of work.

Event-Driven Architecture

- By contrast, complex event processing (CEP) usually involves evaluating an aggregation of events, typically coming from all levels of a business.
- This may require analysis, correlation, and pattern matching to determine the appropriate action to be taken.
- Business process management (BPM) is a typical use case for CEP.
- There can be a number of different events that occur and only a certain combination of events will require a response.
- Spring Integration has the basic building blocks for CEP, including routers and aggregators

Questions

Module Summary

- – Spring Integration Framework.
- – Message, Channel and Adapter
- – Understood the different Component Integration
- – Understood the Event-Driven Architecture