Getting Started with Spring Integration

Mark Fisher, SpringSource
http://springsource.org/spring-integration

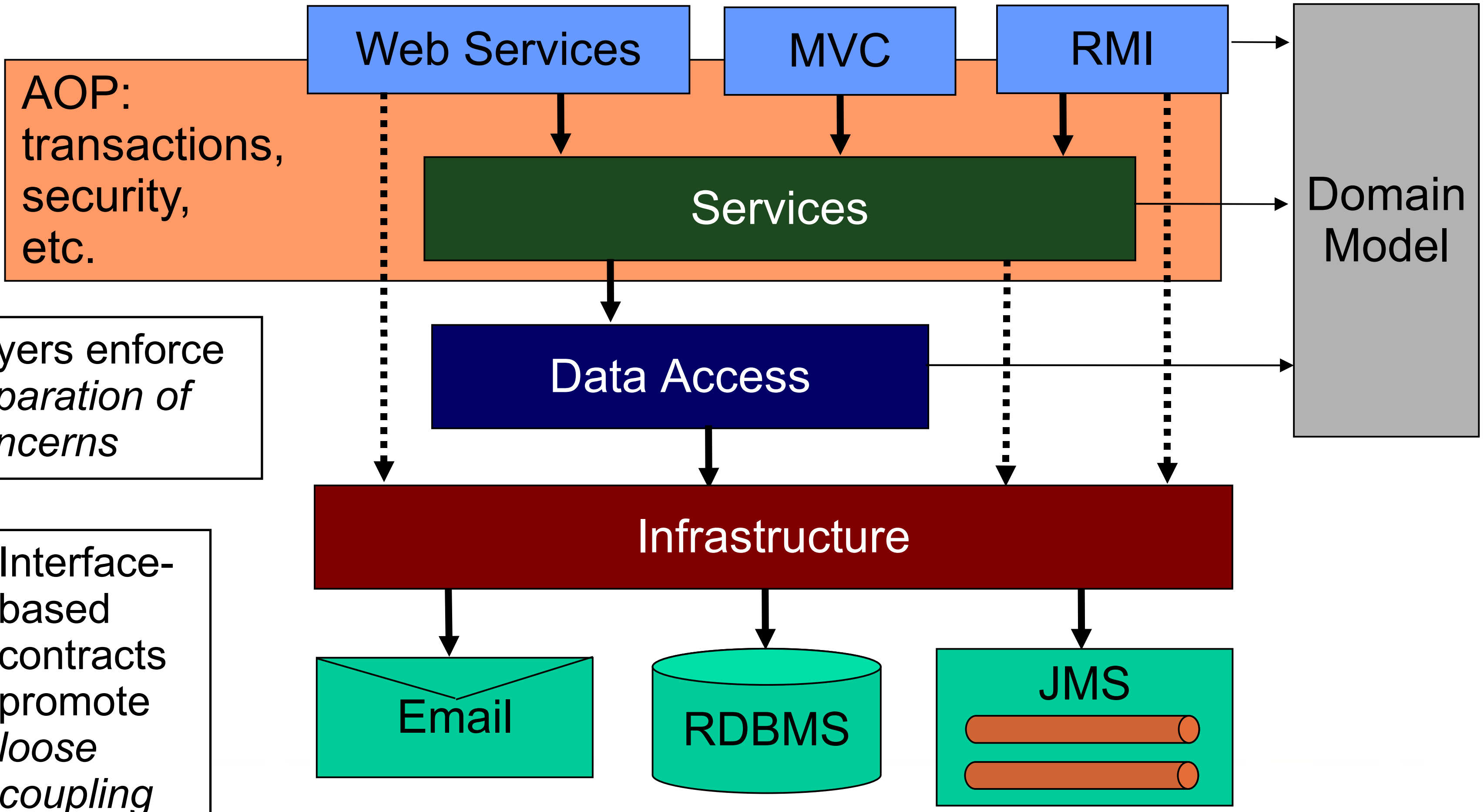
Topics

- Background
- Message Construction
- Channels and Endpoints
- Message Routing
- Adapters
- Roadmap

Spring: Big Picture

- Inversion of Control
- Application code should be
  - Testable
  - Maintainable
  - Flexible
  - Robust
- Developers should be able to focus on the specific business domain, *not* infrastructure and plumbing

Layered Architecture



Layers enforce *separation of concerns*

Interface-based contracts promote *loose coupling*

Event-Driven Architecture

- Essentially Inversion of Control at runtime
  - Framework polls *or* listens to an event source
  - Framework notifies or invokes a service

Example: Spring JMS Message-Driven POJOs

```
<jms:listener-container transaction-manager="txManager">
    <jms:listener ref="orderService"
                  method="order"
                  destination="queue.orders"
                  response-destination="queue.confirmation"/>
</jms:listener-container>
```

```
public class OrderService {

        public OrderConfirmation order(Order o) {...}

}
```

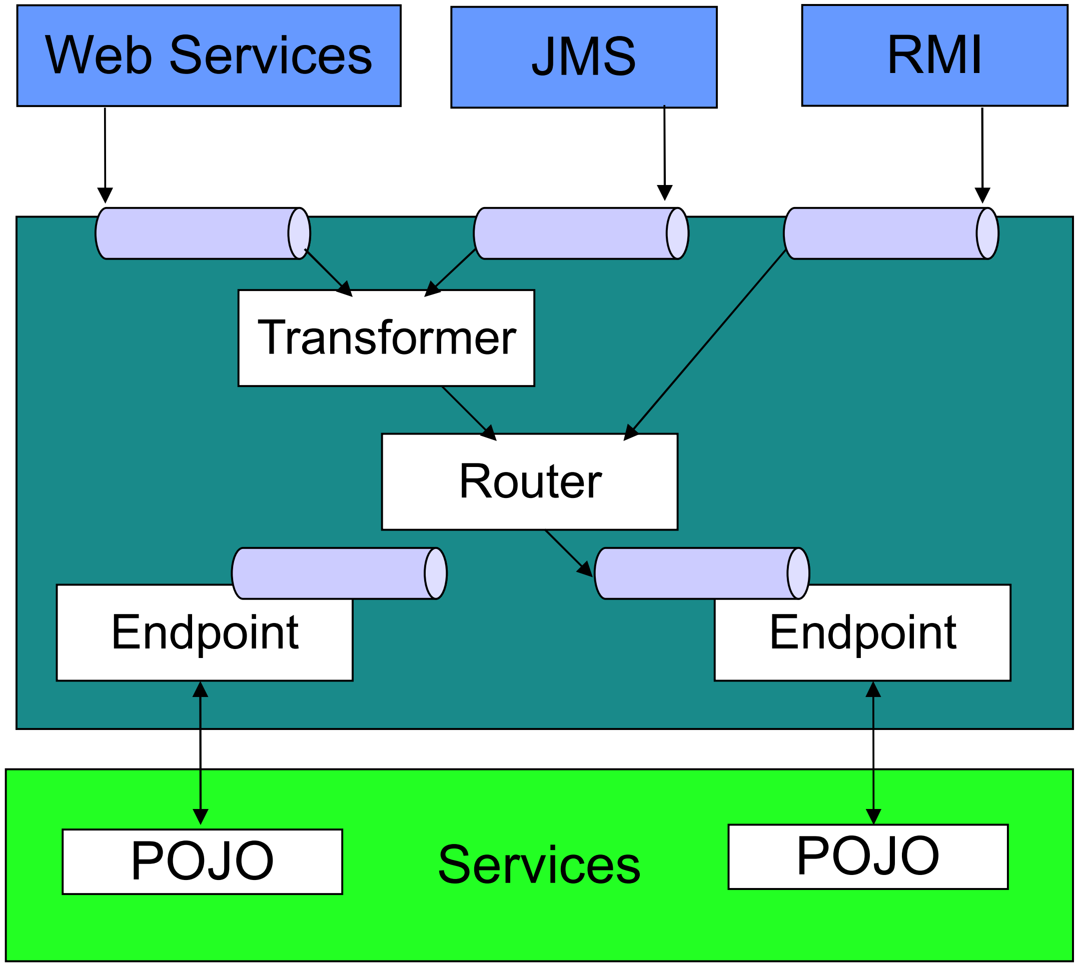
Event Driven SOA with Spring Integration

- Challenges
  - Numerous data sources and targets
    - (File, JMS, WS, HTTP, Mail, etc)
  - Heterogeneous data formats
- Goals
  - Reuse existing service layer
  - Add integration components *incrementally*

Spring Integration Architecture

MessageChannels promote loose coupling between producers and consumers

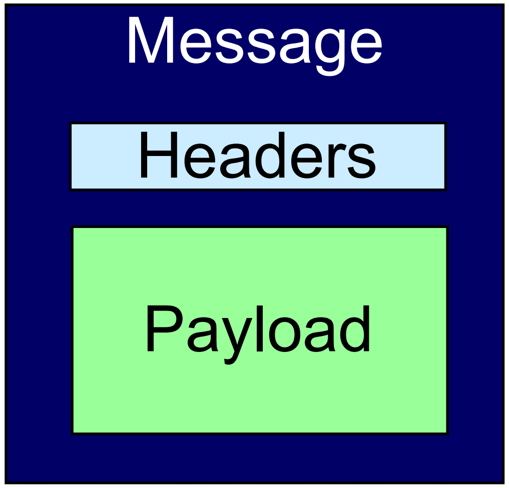
Message Endpoints enforce separation of business and integration logic (polling, transforming, routing, etc).



Message Construction

Message

- A generic package for any payload that can be transported via channels
- Headers provide information to other components that handle the message
  - Sequence Number
  - Sequence Size
  - Expiration Date
  - Correlation Identifier
  - Return Address
  - Transport Info



Message

```
public interface Message<T> {

    MessageHeaders getHeaders();

    T getPayload();

}
```

Message Headers

```
MessageHeaders headers = message.getHeaders();

String value = headers.get("key", String.class);

Object id = headers.getId();

long timestamp = headers.getTimestamp();

MessagePriority priority = headers.getPriority();
```

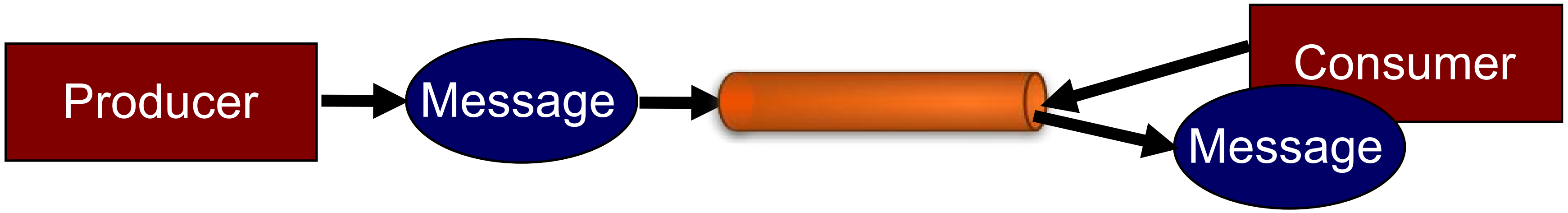
MessageBuilder

```
Message<String> message = MessageBuilder.withPayload("test")
        .setHeader("foo", 123)
        .setPriority(MessagePriority.HIGHEST)
        .build();


Message<String> copy = MessageBuilder.fromMessage(message)
        .setHeader("foo", 456)
        .setHeaderIfAbsent("bar", 789)
        .build();
```

Channels and Endpoints

Message Channel

- Decouples producers from consumers
- May be Point-to-Point or Publish/Subscribe
- Enables interception

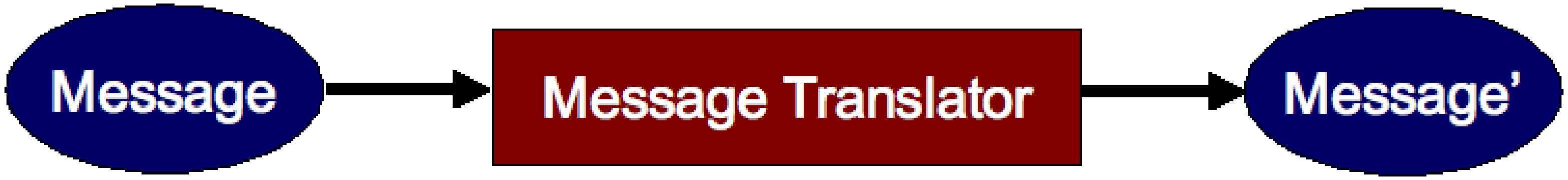


Message Channels

```
<channel id="sync-p2p"/>

<channel id="async-p2p"><queue capacity="50"/></channel>

<publish-subscribe-channel id="pubsub"/>

<channel id="priorityChannel">
   <priority-queue comparator="someComparator"/>
</channel>

<channel id="rendezvousChannel"><rendezvous-queue/></channel>
```

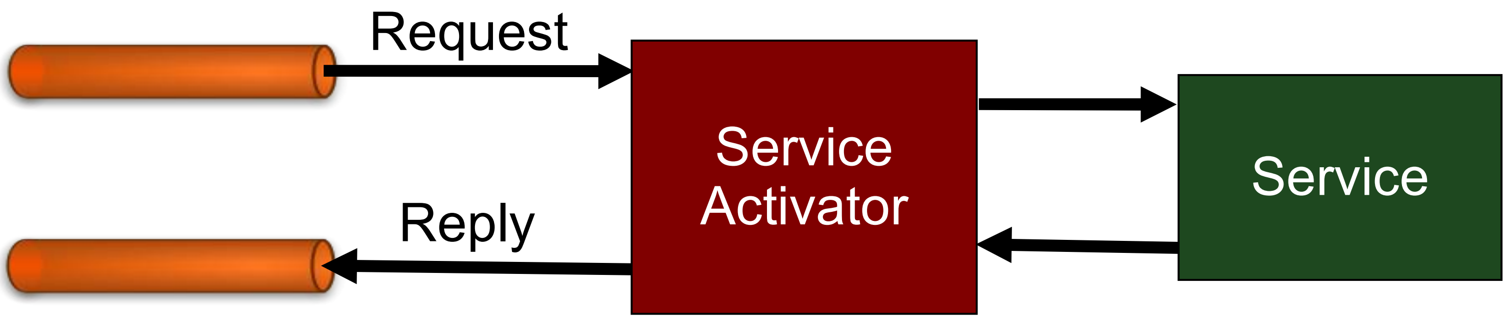
Message Translator

- Payload Transformer
  - converts the type or format of a Message
- Header Transformer
  - add-to or remove-from the MessageHeaders



Service Activator

- A Message Endpoint that invokes a service
- Supports multiple communication styles
  - one-way and request-reply
  - synchronous and asynchronous
- The service is unaware of the messaging system



Service Activator

```
<channel id="requests"/>
<channel id="quotes"/>

<service-activator input-channel="requests"
                   ref="loanBroker"
                   method="processRequest"
                   output-channel="quotes"/>

<beans:bean id=“loanBroker” class=“example.LoanBroker”/>
```

Annotation-Based Configuration

```
@MessageEndpoint
public class LoanBroker {

    @ServiceActivator(inputChannel=”x”, outputChannel=”y”)
    public LoanQuote processRequest(LoanRequest request) {
        LoanQuote quote = ...
        return quote;
    }

}
```

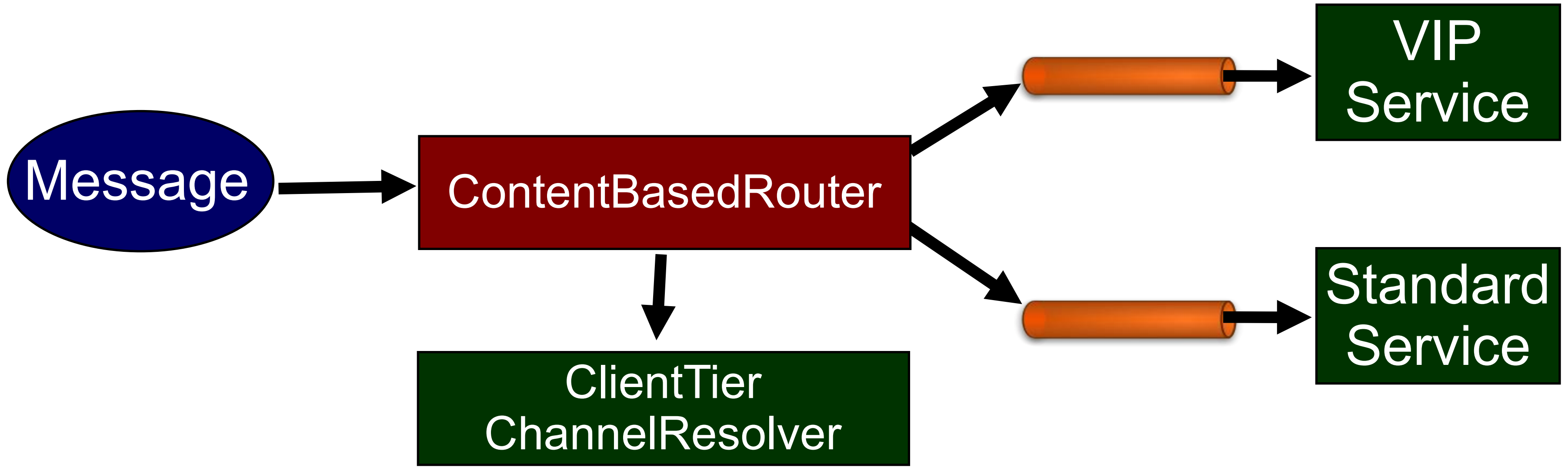
Polling and Transactions

```
<service-activator ref="loanBroker"
                   method="processRequest"
                   input-channel="requests"
                   output-channel="quotes">
   <poller task-executor="pool1">
      <interval-trigger interval="5000"/>
      <transactional propagation="REQUIRES_NEW"/>
   </poller>
</service-activator>
<pool-executor id="pool1" max-size="25"/>
<beans:bean id="transactionManager" ... />
```

Message Routing

Content Based Router

- Determine target channel based on
  - payload type
  - property value
  - header attribute



PayloadTypeRouter

```
typeMap.put(String.class, stringChannel);
typeMap.put(Integer.class, integerChannel);

PayloadTypeRouter router = new PayloadTypeRouter();
router.setPayloadTypeChannelMap(typeMap);

router.handleMessage(new StringMessage(“test”)); //  to ‘stringChannel’
router.handleMessage(new GenericMessage(123)); // to ‘integerChannel’
```

RecipientListRouter

```
List<MessageChannel> channels = new ArrayList<MessageChannel>();
channels.add(channel1);
channels.add(channel2);

RecipientListRouter router = new RecipientListRouter();
router.setChannels(channels);
Message<String> message = new StringMessage("test");

router.handleMessage(message); // will send to channel1 and channel2
```

MethodInvokingRouter

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

<channel id="odd"/>

<router ref="parityResolver" input-channel="numbers"/>
```

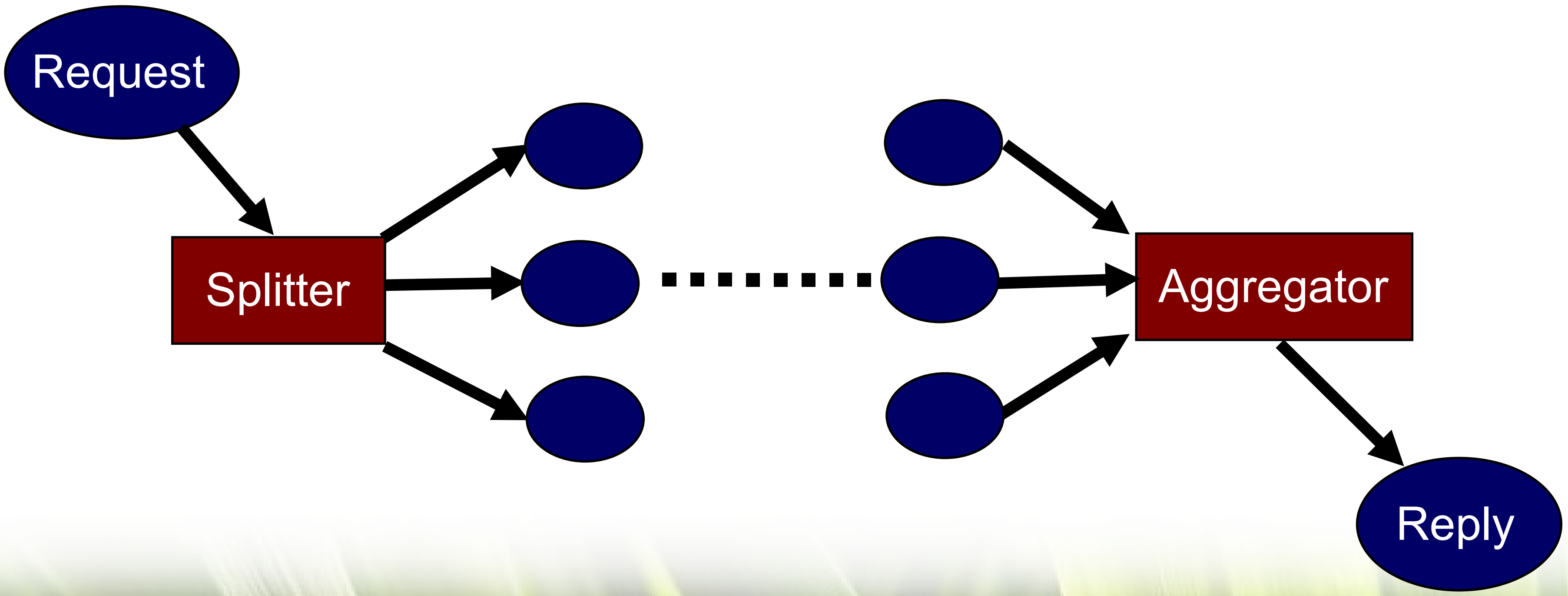
```
@Router
public String getParity(int i) {
   return (i % 2 == 0) ? "even" : "odd";
}
```

...or return a MessageChannel instance

...or return multiple Strings/MessageChannels

Splitter and Aggregator

- Divide coarse-grained message into sub-messages
- Delegate to distributed endpoints as necessary
- Recombine asynchronous reply messages



Splitter and Aggregator

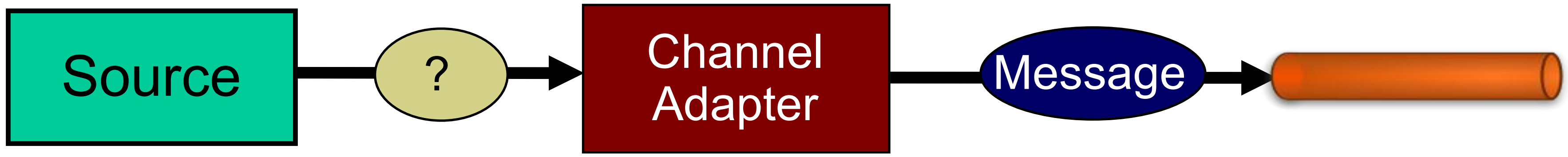
```
@Splitter
public List<OrderItem> splitOrder(PurchaseOrder order,
                                @Header("customerId") String customerId) {

   // split the purchase order into order items…

}
```

```
@Aggregator
public PurchaseOrder aggregateOrder(List<OrderItem> items) {

   // aggregate the items into a single order object...

}
```

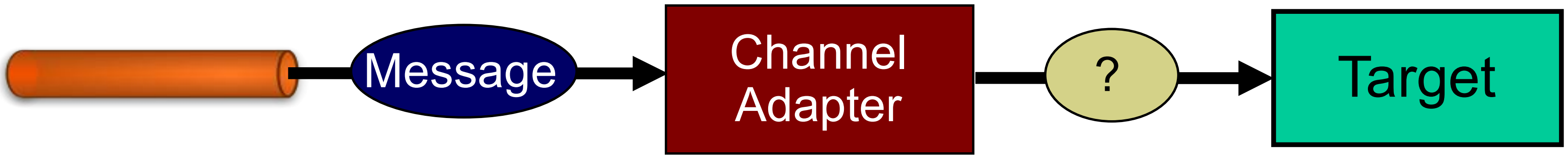
Adapters

Channel Adapter

- Connect a source to the messaging system



- Connect a target to the messaging system



File

```
<file:inbound-channel-adapter channel=“filesIn”
      directory=“${java.io.tmpdir}/test-input”>
  <poller max-messages-per-poll=“5”>
    <cron-trigger expression=“*/10 * * * * MON-FRI”/>
  </poller>
</file:inbound-channel-adapter>

<file:outbound-channel-adapter channel=“filesOut”
      directory=“${java.io.tmpdir}/test-output”/>
```

JMS

```
<jms:inbound-channel-adapter channel=“input”
      connection-factory=“connectionFactory”
      destination-name=“sourceQueueName”/>

<jms:outbound-channel-adapter channel=“output”
      destination=“targetQueue”/>

<jms:inbound-gateway request-channel=“inRequests”
      destination=“inboundRequestQueue”/>

<jms:outbound-gateway request-channel=“outRequests”
   reply-channel=“replies” jms-queue=“outQueue”/>
```

Method Invoking Adapters

```
<channel id=“channel”/>

<inbound-channel-adapter channel=“channel”
            ref=“reader” method=“read”>
   <poller max-messages-per-poll=“1”>
      <interval-trigger interval=“1000”/>
   </poller>
</inbound-channel-adapter>

<outbound-channel-adapter channel=“channel”
              ref=“writer” method=“write”/>
```

Other Adapters

- HTTP
- Web Services
- Mail
- RMI
- Spring ApplicationEvents
- ...and more in Spring Extensions
  - www.springsource.org/extensions

Spring Integration 2.0: Roadmap

- Building on Spring 3.0
- Expression Language support
  - Message-to-argument binding on methods
  - Routers and Transformers directly in XML
- TaskScheduler Juergenized
- RestTemplate/HTTP client-side API
- JDBC Adapters
- Groovy scripts for Routers, Transformers, etc.
- Process Manager (scope, state, and context)
- ???

Suggested Reading

- Enterprise Integration Patterns
  - Gregor Hohpe and Bobby Woolf (Addison Wesley, 2004)
- Pattern-Oriented Software Architecture, v.4
  - Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt (Wiley, 2007)
- Event-Based Programming
  - Ted Faison (Apress, 2006)
- Java Messaging
  - Eric Bruno (Charles River Media, 2006)
- Open Source ESBs in Action
  - Tijs Rademakers and Jos Dirksen (Manning, 2008)

Questions?

http://springsource.org/spring-integration